
Amazon Aurora

Aurora 사용 설명서



Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Table of Contents

Aurora란 무엇입니까?	1
Aurora DB 클러스터	2
리전 및 가용 영역	3
리전 가용성	3
DB 클러스터의 현지 시간대	7
Aurora 연결 관리	9
Aurora 엔드포인트 유형	10
엔드포인트 보기	12
클러스터 엔드포인트 사용	12
리더 엔드포인트 사용	12
사용자 지정 엔드포인트 사용	12
사용자 지정 엔드포인트 만들기	15
사용자 지정 엔드포인트 보기	17
사용자 지정 엔드포인트 편집	22
사용자 지정 엔드포인트 삭제	24
사용자 지정 엔드포인트에 대한 종합 AWS CLI 예제	25
인스턴스 엔드포인트 사용	29
엔드포인트와 고가용성	29
DB 인스턴스 클래스	30
DB 인스턴스 클래스 유형	30
용어	31
하드웨어 사양	31
Aurora 스토리지 및 안정성	33
Aurora 스토리지 개요	34
클러스터 볼륨 콘텐츠	34
스토리지 확장	34
데이터 결제	34
안정성	34
Aurora 보안	36
Aurora DB 클러스터에 SSL 사용	36
Aurora를 위한 고가용성	36
Aurora 글로벌 데이터베이스	37
글로벌 데이터베이스 개요	37
글로벌 데이터베이스 생성	39
AWS 리전 추가	46
클러스터 제거	48
Aurora 글로벌 데이터베이스 삭제	50
글로벌 데이터베이스로 데이터 가져오기	51
Aurora 글로벌 데이터베이스에 연결	52
글로벌 데이터베이스에 대한 장애 조치	53
Global Database를 위한 성능 개선 도우미	53
여러 개의 보조 리전	54
다른 AWS 서비스와 함께 글로벌 데이터베이스 사용	54
Aurora를 사용한 복제	55
Aurora 복제본	55
Aurora MySQL	56
Aurora PostgreSQL	56
Aurora에 대한 DB 인스턴스 결제	56
온 디맨드 DB 인스턴스	57
예약 DB 인스턴스	57
환경 설정	69
AWS에 가입	69
IAM 사용자 생성	69
요구 사항 결정	71

보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다.	72
시작하기	74
Aurora MySQL DB 클러스터 생성 및 연결	74
Aurora MySQL DB 클러스터 생성	74
DB 클러스터의 인스턴스에 연결하기	81
샘플 DB 클러스터, DB 서브넷 그룹 및 VPC 삭제	82
Aurora PostgreSQL 클러스터 생성 및 연결	83
Aurora PostgreSQL DB 클러스터 생성	83
Aurora PostgreSQL DB 클러스터의 인스턴스에 연결	92
샘플 DB 클러스터, DB 서브넷 그룹 및 VPC 삭제	93
Aurora DB 클러스터 구성	95
DB 클러스터 생성	95
사전 조건	95
DB 클러스터 생성	96
사용 가능한 설정	106
사용 Aurora Serverless	111
Aurora Serverless의 장점	112
Aurora Serverless 사용 사례	112
Aurora Serverless의 제한 사항	113
Aurora Serverless에서 TLS/SSL 사용	114
Aurora Serverless 작동 방식	114
Aurora Serverless DB 클러스터 생성	120
Aurora Serverless DB 클러스터 복원	125
Aurora Serverless DB 클러스터 수정	127
Aurora Serverless DB 클러스터의 용량 설정	129
Aurora Serverless DB 클러스터 보기	131
데이터 API 사용	134
AWS CloudTrail을 사용하여 데이터 API 호출 로깅	157
쿼리 편집기 사용	158
DB 클러스터에 연결	162
Aurora MySQL	163
Aurora PostgreSQL	166
문제 해결	168
파라미터 그룹 작업	168
DB 클러스터와 DB 인스턴스 파라미터	170
DB 파라미터 그룹 생성	171
DB 클러스터 파라미터 그룹 만들기	172
DB 파라미터 그룹의 파라미터 수정	174
DB 클러스터 파라미터 그룹의 파라미터 수정	177
DB 파라미터 그룹 복사	179
DB 클러스터 파라미터 그룹 복사	180
DB 파라미터 그룹 나열	181
DB 클러스터 파라미터 그룹 나열	182
DB 파라미터 그룹의 파라미터 값 보기	183
DB 클러스터 파라미터 그룹의 파라미터 값 보기	184
파라미터 그룹 비교	186
DB 파라미터 값	186
데이터를 DB 클러스터로 마이그레이션하기	188
Aurora MySQL	189
Aurora PostgreSQL	189
Aurora DB 클러스터 관리	190
클러스터 중지 및 시작	190
클러스터의 중지 및 시작 개요	190
제한 사항	191
DB 클러스터 중지	191
DB 클러스터가 중지된 동안	192
DB 클러스터 시작	192

Aurora DB 클러스터 수정	193
콘솔, CLI, API를 사용하여 DB 클러스터 수정	193
DB 클러스터에서 DB 인스턴스 수정	194
사용 가능한 설정	196
해당 사항이 없는 설정	207
Aurora 복제본 추가	208
성능 및 확장 관리	211
스토리지 조정	212
인스턴스 조정	212
읽기 조정	212
연결 관리	212
쿼리 실행 계획 관리	213
Amazon RDS Proxy(미리 보기)를 사용한 연결 관리	213
RDS Proxy 개념 및 용어	214
RDS Proxy 계획 및 설정	217
RDS Proxy를 통해 연결	229
RDS Proxy 관리	230
모니터링	232
제한 사항	234
예제	234
문제 해결	236
Aurora DB 클러스터에서 데이터베이스 복제	239
제한 사항	239
데이터베이스 복제를 위한 기록 중 복사 프로토콜	240
원본 데이터베이스 삭제	241
AWS Management 콘솔을 통한 Aurora 클러스터 복제	241
AWS CLI를 통한 Aurora 클러스터 복제	242
계정 간 복제	243
AWS 서비스 통합	252
Aurora MySQL	253
Aurora PostgreSQL	253
Aurora 복제본에 Auto Scaling 사용	253
Aurora에서 기계 학습 사용	267
Aurora DB 클러스터 백업 및 복구	268
백업 및 복구에 대한 개요	268
백업 스토리지	270
DB 클러스터 스냅샷 생성	271
DB 클러스터 스냅샷에서 복원	273
스냅샷 복사	275
스냅샷 공유	284
Amazon S3로 스냅샷 데이터 내보내기	290
특정 시점으로 복구	303
스냅샷 삭제	305
Aurora DB 클러스터 유지 관리	306
업데이트 적용	307
유지 관리 기간	309
DB 클러스터의 유지 관리 기간 조정	310
Aurora MySQL 유지 관리 업데이트 빈도 선택	312
DB 인스턴스 재부팅	312
DB 인스턴스 삭제	313
삭제 방지	314
단일 DB 인스턴스가 있는 Aurora 클러스터	314
콘솔, CLI, API 사용	314
RDS 리소스에 태그 지정	315
개요	315
ARN 작업	319
ARN 생성	319

기존 ARN 가져오기	322
Aurora 업데이트	325
해당 Amazon Aurora 버전 식별	325
Aurora DB 클러스터 모니터링	327
모니터링 개요	327
모니터링 도구	328
CloudWatch를 사용하여 모니터링	329
CloudWatch Logs에 게시	335
DB 클러스터 보기	338
DB 클러스터 상태	343
DB 인스턴스 상태	344
Aurora DB 클러스터 지표 모니터링	346
Aurora 지표	346
Amazon RDS 콘솔에서 지표 보기	353
Amazon RDS 콘솔에서 사용 가능한 Aurora 지표	355
확장 모니터링	358
CloudWatch 측정치와 Enhanced Monitoring 측정치의 차이점	358
확장 모니터링 설정 및 활성화	358
확장 모니터링 보기	361
CloudWatch Logs를 사용하여 Enhanced Monitoring 보기	363
성능 개선 도우미	365
성능 개선 도우미 활성화	367
성능 개선 도우미의 액세스 제어	371
성능 개선 도우미 대시보드 사용	372
추가 사용자 인터페이스 기능	385
성능 개선 도우미 API	386
CloudWatch에 게시되는 지표	398
성능 개선 도우미 카운터	400
AWS CloudTrail를 사용하여 성능 개선 도우미 호출 로깅	406
Amazon Aurora 권장 사항 사용	408
권장 사항 대응	409
데이터베이스 활동 스트림 사용	412
활동 스트림 시작	413
활동 스트림 상태 가져오기	414
활동 스트림 중지	415
활동 스트림 모니터링	416
활동 스트림 액세스 관리	427
Amazon RDS 이벤트 알림 서비스 사용	429
Amazon RDS 이벤트 카테고리 및 이벤트 메시지	430
Amazon RDS 이벤트 알림 구독	436
Amazon RDS 이벤트 알림 구독의 목록 표시	438
Amazon RDS 이벤트 알림 구독 변경	439
Amazon RDS 이벤트 알림 구독에 대한 소스 식별자 추가	440
Amazon RDS 이벤트 알림 구독의 소스 식별자 제거	441
Amazon RDS 이벤트 알림 카테고리의 목록 표시	442
Amazon RDS 이벤트 알림 구독 삭제	443
Amazon RDS 이벤트 보기	444
콘솔	444
AWS CLI	444
API	444
.....	445
Amazon Aurora에 대한 CloudWatch 이벤트 및 EventBridge 이벤트	445
CloudWatch 이벤트로 Amazon RDS 이벤트 전송	445
DB 클러스터 이벤트	446
DB 인스턴스 이벤트	446
DB 파라미터 그룹 이벤트	447
DB 보안 그룹 이벤트	447

DB 클러스터 스냅샷 이벤트	448
DB 스냅샷 이벤트	448
데이터베이스 로그 파일	449
데이터베이스 로그 파일 보기 및 나열	449
데이터베이스 로그 파일 다운로드	450
데이터베이스 로그 파일 조사	451
CloudWatch Logs에 게시	451
REST를 사용하여 로그 파일 내용 읽기	451
MySQL 데이터베이스 로그 파일	452
PostgreSQL 데이터베이스 로그 파일	456
AWS CloudTrail을 사용하여 Amazon RDS API 호출 로깅	459
CloudTrail의 Amazon RDS 정보	459
Amazon RDS 로그 파일 항목 이해	460
Aurora MySQL 작업	463
Aurora MySQL의 개요	463
Amazon Aurora MySQL 성능 개선 사항	463
Aurora MySQL 및 지형 정보 데이터	464
Aurora MySQL 5.6과 Aurora MySQL 5.7의 비교	465
Aurora MySQL 5.7과 MySQL 5.7 비교	465
Aurora MySQL를 사용한 보안	466
Aurora MySQL을 사용한 마스터 사용자 권한	466
Aurora MySQL DB 클러스터에 SSL 사용	467
애플리케이션을 업데이트하여 새 SSL/TLS 인증서 획득	468
애플리케이션에서 SSL을 사용해 Aurora MySQL DB 클러스터에 연결하는지 여부 확인	469
클라이언트에서 연결 시 인증서 확인이 필요한지 여부 확인	469
애플리케이션 트러스트 스토어 업데이트	470
SSL 연결 설정을 위한 Java 코드 예시	471
Aurora MySQL로 데이터 마이그레이션	472
외부 MySQL 데이터베이스에서 Aurora MySQL로 마이그레이션	474
MySQL DB 인스턴스에서 Aurora MySQL로 마이그레이션	492
Aurora MySQL 관리	508
Amazon Aurora MySQL에 대한 성능 및 조정 관리	508
DB 클러스터 역주적	509
오류 삽입 쿼리를 사용하여 Amazon Aurora 테스트	522
빠른 DDL을 이용하는 Amazon Aurora에서의 테이블 수정	525
Aurora DB 클러스터를 위한 볼륨 상태 표시	526
Aurora MySQL용 Parallel Query	527
Parallel Query 개요	528
관리	530
병렬 쿼리 업그레이드	530
병렬 쿼리 클러스터 생성	530
병렬 쿼리 활성화 및 비활성화	534
성능 튜닝	536
스키마 객체 생성	536
병렬 쿼리 사용 확인	536
모니터링	539
병렬 쿼리 및 SQL 구조	540
Aurora MySQL의 고급 감사	552
고급 감사 활성화	552
감사 로그 보기	553
감사 로그 세부 정보	554
Aurora MySQL를 사용하는 단일 마스터 복제	554
Aurora 복제본	555
옵션	555
성능	556
고가용성	556
모니터링	557

여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제	557
Aurora와 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터 간의 복제	567
GTID 기반 복제 사용	580
멀티 마스터 클러스터 작업	584
멀티 마스터 클러스터의 개요	584
멀티 마스터 클러스터 생성	589
멀티 마스터 클러스터 관리	594
애플리케이션 고려 사항	596
성능 고려 사항	606
멀티 마스터 클러스터에 대한 접근 방식	607
Aurora MySQL과 AWS 서비스 통합	609
Aurora MySQL이 AWS 서비스에 액세스할 수 있도록 권한 부여	609
Amazon S3의 텍스트 파일에서 데이터 로드	620
Amazon S3의 텍스트 파일에 데이터 저장	627
Aurora MySQL에서 Lambda 함수 호출	633
CloudWatch Logs에 Aurora MySQL 로그 게시	639
Aurora MySQL에서 기계 학습 사용	642
Aurora MySQL 랩 모드	654
Aurora 랩 모드 기능	655
Amazon Aurora MySQL 모범 사례	655
연결되어 있는 DB 인스턴스 확인	656
T2 인스턴스 사용	656
AWS Lambda 함수 호출	657
비동기식 키 미리 가져오기 작업	657
Amazon Aurora MySQL에서 다중 스레드 복제 슬레이브 사용	659
Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 조정	659
MySQL 데이터베이스에서 Amazon Aurora를 재해 복구용으로 사용	662
감소된 종단 시간으로 MySQL에서 Amazon Aurora MySQL로 마이그레이션	662
Amazon Aurora MySQL에서 XA 트랜잭션 사용	663
해시 조인 작업	663
외래 키 작업	664
관련 주제	665
Aurora MySQL 참조	665
파라미터	665
적용할 수 없는 MySQL 파라미터 및 상태 변수	677
Aurora MySQL 이벤트	677
Aurora MySQL 격리 수준	679
저장 프로시저	683
Aurora MySQL 업데이트	686
Aurora MySQL 버전	686
Aurora MySQL 엔진 버전	686
Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치	687
제로 가동 중지 패치 적용	688
Aurora MySQL LTS(장기 지원) 릴리스	689
관련 주제	689
Amazon Aurora MySQL 2.0에 대한 데이터베이스 엔진 업데이트	690
Amazon Aurora MySQL 1.1에 대한 데이터베이스 엔진 업데이트	722
Aurora MySQL 업데이트를 통해 수정한 MySQL 버그	763
Aurora PostgreSQL 작업	773
Aurora PostgreSQL을 사용한 보안	773
암호 관리 제한	774
SSL을 이용한 Aurora PostgreSQL 데이터 보안	775
애플리케이션을 업데이트하여 새 SSL/TLS 인증서 획득	777
애플리케이션에서 SSL을 사용해 Aurora PostgreSQL DB 클러스터에 연결하는지 여부 확인	778
클라이언트에서 연결을 위해 인증서 확인이 필요한지 여부 확인	778
애플리케이션 트러스트 스토어 업데이트	779
다양한 유형의 애플리케이션에 대해 SSL/TLS 연결 사용	780

Aurora PostgreSQL로 데이터 마이그레이션	781
RDS PostgreSQL DB 스냅샷 마이그레이션	781
Aurora 읽기 전용 복제본을 사용한 RDS PostgreSQL DB 인스턴스 마이그레이션	783
S3 데이터를 Aurora PostgreSQL로 가져오기	791
Aurora PostgreSQL 관리	802
Aurora PostgreSQL DB 인스턴스 조정	802
최대 연결 수	802
오류 삽입 쿼리를 사용하여 Amazon Aurora PostgreSQL 테스트	803
Aurora DB 클러스터를 위한 볼륨 상태 표시	806
Aurora PostgreSQL을 사용한 복제	807
Aurora 복제본	807
복제 모니터링	808
논리적 복제 사용	808
Aurora PostgreSQL과 AWS 서비스 통합	812
PostgreSQL 데이터를 Amazon S3 파일로 내보내기	812
S3으로 내보내기 개요	813
내보낼 Amazon S3 파일 경로 지정	813
Amazon S3 버킷에 대한 액세스 권한 설정	814
aws_s3.export_query_to_s3 함수를 사용하여 쿼리 데이터 내보내기	817
함수 참조	819
Aurora PostgreSQL용 쿼리 실행 계획 관리	821
쿼리 계획 관리 활성화	822
쿼리 계획 관리 업그레이드	823
기본 사항	823
쿼리 계획 관리에 대한 모범 사례	826
dba_plans 보기에서 계획 검토	827
실행 계획 캡처	829
관리형 계획 사용	831
실행 계획 유지 관리	834
쿼리 계획 관리를 위한 파라미터 참조	838
쿼리 계획 관리를 위한 함수 참조	842
CloudWatch Logs에 Aurora PostgreSQL 로그 게시	847
Amazon CloudWatch에서 로그 이벤트 모니터링	850
Aurora PostgreSQL에서 기계 학습 사용	850
Aurora 기계 학습 활성화	851
자연어 처리에 Amazon Comprehend 사용	854
Amazon SageMaker 모델 교육을 위해 Amazon S3로 데이터 내보내기	855
Amazon SageMaker를 사용하여 자체 ML 모델 실행	855
Aurora 기계 학습 모범 사례	858
Aurora 기계 학습 모니터링	861
함수 참조	863
AWS CLI를 사용하여 수동으로 IAM 역할 설정	864
장애 조치 후 신속한 복구	868
클러스터 캐시 관리 구성	869
버퍼 캐시 모니터링	871
PostgreSQL DB 엔진 업그레이드	872
업그레이드 개요	873
PostgreSQL 버전 식별	873
메이저 버전 업그레이드	874
Aurora PostgreSQL 엔진 수동 업그레이드	876
마이너 버전 자동 업그레이드	877
PostgreSQL 확장 버전 업그레이드	877
Aurora PostgreSQL 모범 사례	878
빠른 장애 조치	878
스토리지 문제 해결	885
Kerberos 인증 사용	885
Kerberos 인증 개요	886

설정	886
도메인에서 DB 클러스터 관리	894
Kerberos 인증을 사용하여 연결	894
Aurora PostgreSQL 참조	895
Aurora PostgreSQL 파라미터	896
Aurora PostgreSQL 이벤트	904
Aurora PostgreSQL 업데이트	905
해당 버전 식별	906
업데이트	906
Aurora PostgreSQL의 엔진 버전	906
Aurora PostgreSQL의 확장 버전	926
Aurora 모범 사례	929
Amazon Aurora 기본 운영 지침	929
DB 인스턴스 RAM 권장 사항	929
Amazon Aurora 모니터링	930
DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업	930
Amazon Aurora 모범 사례 프레젠테이션 동영상	930
Aurora 개념 증명 수행	931
Aurora 개념 증명 개요	931
1. 목표 식별	931
2. 고객님의 워크로드 특성에 대한 이해	932
3. 콘솔 또는 CLI를 이용한 실습	933
콘솔을 이용한 실습	933
AWS CLI를 이용한 실습	933
4. Aurora 클러스터 생성	934
5. 스키마 설정	934
6. 데이터 가져오기	935
7. SQL 코드 포트	935
8. 구성 설정 지정	936
9. Aurora에 연결	936
10. 워크로드 실행	937
11. 성능 측정	938
12. Aurora 고가용성 실습	940
13. 다음에 수행할 작업	941
보안	943
데이터 보호	944
데이터 암호화	945
인터넷워크 트래픽 개인 정보	960
ID 및 액세스 관리	960
대상	961
자격 증명을 통한 인증	961
정책을 이용한 액세스 관리	963
Amazon Aurora에서 IAM을 사용하는 방식	964
자격 증명 기반 정책 예제	967
을 위한 IAM 데이터베이스 인증	976
문제 해결	993
Kerberos 인증	994
로깅 및 모니터링	995
규정 준수 확인	996
복원성	997
백업 및 복원	997
복제	997
Failover	997
인프라 보안	998
보안 그룹	998
퍼블릭 액세스 가능성	998
보안 모범 사례	998

보안 그룹을 통한 액세스 제어	999
VPC 보안 그룹	999
보안 그룹 시나리오	999
VPC 보안 그룹 생성	1000
DB 인스턴스 연결	1000
DB 클러스터 연결	1000
마스터 사용자 계정 권한	1001
서비스 연결 역할	1001
Amazon Aurora에 대한 서비스 연결 역할 권한	1002
Amazon Aurora에 대한 서비스 연결 역할 생성	1003
Amazon Aurora에 대한 서비스 연결 역할 편집	1004
Amazon Aurora에 대한 서비스 연결 역할 삭제	1004
Amazon Aurora와 Amazon VPC 사용	1005
Aurora용 VPC 생성	1005
VPC에서 DB 인스턴스에 액세스하는 시나리오	1011
VPC에서 DB 인스턴스를 사용한 작업	1015
자습서: DB 인스턴스에 사용할 Amazon VPC 생성	1020
활당량 및 제약 조건	1025
Amazon Aurora의 활당량	1025
Amazon Aurora의 명명 제약 조건	1026
Amazon Aurora 파일 크기 제한	1027
문제 해결	1028
DB 인스턴스에 연결할 수 없음	1028
DB 인스턴스 연결 테스트	1029
연결 인증 문제 해결	1029
보안 문제	1029
오류 메시지 "계정 속성을 불러오지 못했습니다. 일부 콘솔 기능이 손상되었을 수 있습니다."	1030
DB 인스턴스 소유작 역할 암호 재설정	1030
DB 인스턴스 중단 또는 재부팅	1030
파라미터 변경 사항이 적용 안 됨	1031
Aurora MySQL 메모리 부족 문제	1031
Aurora MySQL 복제 문제	1031
Read Replica 사이의 지연 문제 진단 및 해결	1031
MySQL 읽기 복제 오류 진단 및 해결	1033
Slave Down 또는 Disabled 오류	1034
No Space Left on Device(장치에 남은 공간 없음) 오류	1034
Amazon RDS API 참조	1036
Query API 사용	1036
쿼리 파라미터	1036
쿼리 요청 인증	1036
애플리케이션 문제 해결	1036
오류 검색	1037
문제 해결 팁	1037
문서 이력	1038
AWS Glossary	1054

Amazon Aurora란 무엇입니까?

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 최대 64 tebibytes (TiB)까지 자동으로 커집니다. 또한 Aurora는 데이터베이스 구성 및 관리의 가장 어려운 측면 중 하나인 데이터베이스 클러스터링 및 복제를 자동화하고 표준화합니다.

Aurora은 관리형 데이터베이스 서비스 Amazon Relational Database Service(Amazon RDS)의 일부입니다. Amazon RDS는 클라우드에서 관계형 데이터베이스를 보다 쉽게 설정, 작동 및 확장할 수 있게 해주는 웹 서비스입니다. Amazon RDS에 익숙하지 않은 경우 [Amazon Relational Database Service 사용 설명서](#)를 참조하십시오.

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

- Amazon RDS를 통해 새 데이터베이스 서버를 설정할 때 Aurora을 DB 엔진 옵션으로 선택합니다.
- Aurora는 관리 및 운영을 위해 익숙한 Amazon Relational Database Service(Amazon RDS) 기능을 활용합니다. Aurora는 Amazon RDS AWS Management 콘솔 인터페이스, AWS CLI 명령 및 API 작업을 사용하여 프로비저닝, 패치, 백업, 복구, 장애 감지 및 수리와 같은 일상적인 데이터베이스 작업을 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.
- 스냅샷을 생성 및 복원하거나 단방향 복제를 설정하여 MySQL용 Amazon RDS 및 PostgreSQL용 Amazon RDS의 데이터를 Aurora로에서 데이터를 가져올 수 있습니다. 기존 MySQL 애플리케이션용 Amazon RDS와 PostgreSQL 애플리케이션용 Amazon RDS를 Aurora로 전환할 수 있는 푸시 버튼식 마이그레이션 도구를 사용할 수 있습니다.

Amazon Aurora를 사용하기 전에 먼저 [Amazon Aurora 환경 설정 \(p. 69\)](#)에서 설명하는 단계를 마친 후에 [Amazon Aurora DB 클러스터 \(p. 2\)](#)에서 Aurora에 대한 개념과 기능을 검토해야 합니다.

주제

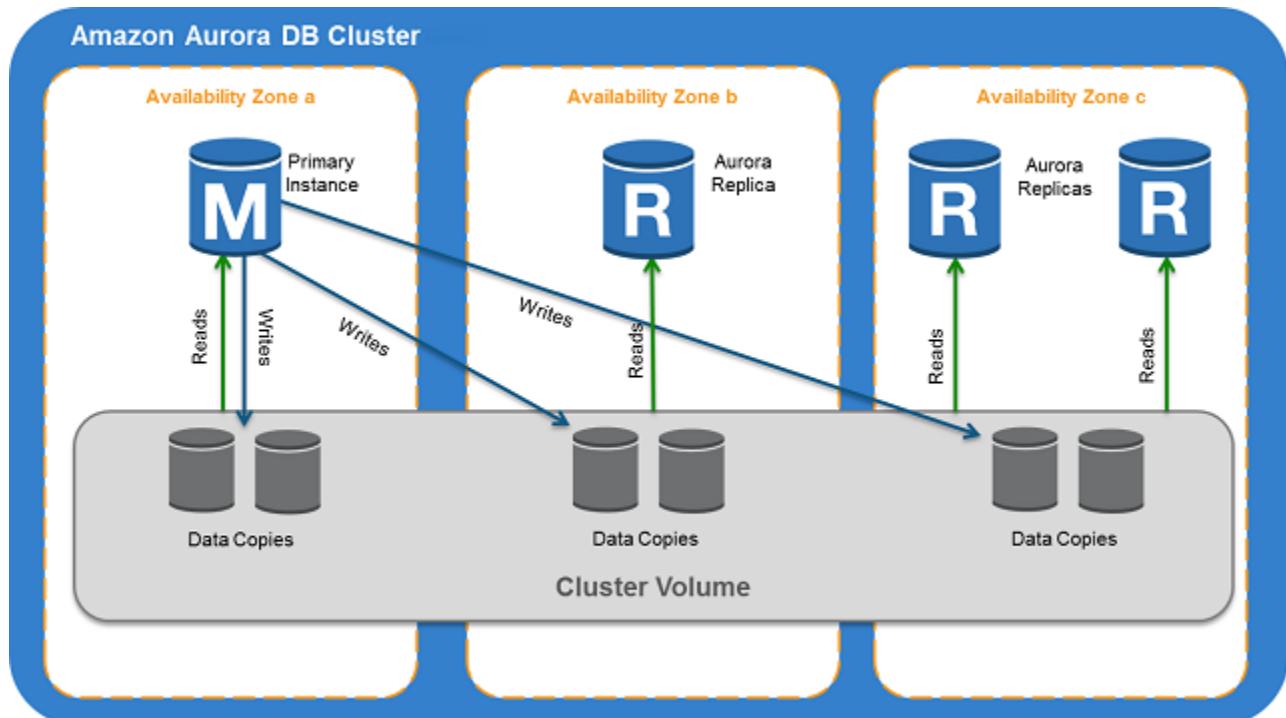
- [Amazon Aurora DB 클러스터 \(p. 2\)](#)
- [리전 및 가용 영역 \(p. 3\)](#)
- [Amazon Aurora 연결 관리 \(p. 9\)](#)
- [인스턴스 앤드포인트 사용 \(p. 29\)](#)
- [Aurora 앤드포인트가 고가용성으로 작동하는 방법 \(p. 29\)](#)
- [DB 인스턴스 클래스 \(p. 30\)](#)
- [Amazon Aurora 스토리지 및 안정성 \(p. 33\)](#)
- [Amazon Aurora 보안 \(p. 36\)](#)
- [Aurora을 위한 고가용성 \(p. 36\)](#)
- [Amazon Aurora 글로벌 데이터베이스 작업 \(p. 37\)](#)
- [Amazon Aurora를 사용한 복제 \(p. 55\)](#)
- [Aurora에 대한 DB 인스턴스 결제 \(p. 56\)](#)

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 DB 인스턴스 – 읽기 및 쓰기 작업을 지원하고, 클러스터 볼륨의 모든 데이터 수정을 실행합니다. Aurora DB 클러스터마다 기본 DB 인스턴스가 하나씩 있습니다.
- Aurora 복제본 – 기본 DB 인스턴스와 동일한 스토리지 볼륨에 연결되며 읽기 작업만 지원합니다. 각 Aurora DB 클러스터는 기본 DB 인스턴스에 대해 최대 15개까지 Aurora 복제본을 구성할 수 있습니다. Aurora 복제본을 별도의 가용 영역에 배포하여 고가용성을 유지할 수 있습니다. 기본 DB 인스턴스를 사용할 수 없는 상태가 되면 Aurora는 Aurora 복제본으로 자동 장애 조치합니다. Aurora 복제본에 대해 장애조치 우선 순위를 지정할 수 있습니다. 또한 Aurora 복제본은 기본 DB 인스턴스에서 읽기 워크로드를 오프로드할 수 있습니다.
- Aurora 멀티 마스터 클러스터에서는 각각의 DB 인스턴스가 읽기-쓰기 기능을 가집니다. 이 경우, 기본 인스턴스와 Aurora 복제본 간의 차이가 적용되지 않습니다. 클러스터가 단일 마스터 또는 멀티 마스터 복제를 사용할 수 있는 복제 토플로지에 대해 논의하기 위해 여기 나온 라이터 및 리더 DB 인스턴스를 호출했습니다.

다음은 클러스터 볼륨과 Aurora DB 클러스터에 속하는 기본 DB 인스턴스 및 Aurora 복제본 사이의 관계를 나타낸 다이어그램입니다.



Note

앞의 내용은 단일 마스터 복제를 사용하는 모든 Aurora 클러스터에 적용됩니다. 여기에는 프로비저닝된 클러스터, 병렬 쿼리 클러스터, 전역 데이터베이스 클러스터, 서비스 클러스터, 모든 MySQL 5.7 호환 및 PostgreSQL 호환 클러스터가 포함됩니다.

멀티 마스터 복제를 사용하는 Aurora 클러스터는 읽기-쓰기 DB 인스턴스와 읽기 전용 DB 인스턴스를 서로 다르게 정렬합니다. 멀티 마스터 클러스터에 있는 모든 DB 인스턴스는 쓰기 작업을 수행할 수 있습니다. 모든 쓰기 작업을 수행하는 단일 DB 클러스터는 없으며, 읽기 전용 DB 인스턴스도 없

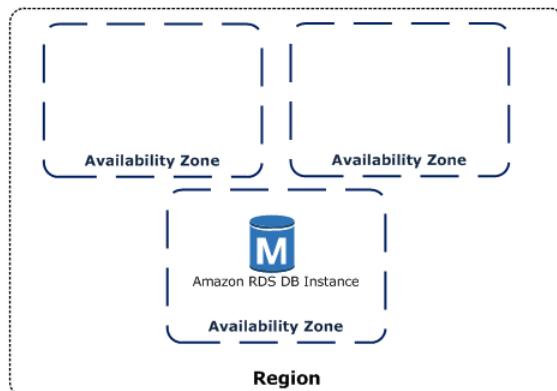
습니다. 따라서 기본 인스턴스 및 Aurora 복제본이라는 용어는 멀티 마스터 클러스터에는 적용되지 않습니다. 멀티 마스터 복제를 사용하는 클러스터에 대해 논의할 때는 라이터 DB 인스턴스와 리더 DB 인스턴스로 지칭합니다.

Aurora 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.

리전 및 가용 영역

Amazon 클라우드 컴퓨팅 리소스는 세계 각지의 여러 곳에서 호스팅됩니다. 이 위치들은 AWS 리전과 가용 영역으로 구성됩니다. 각 AWS 리전은 개별 지역 영역입니다. 각 AWS 리전에는 가용 영역이라는 여러 개의 격리된 위치가 있습니다. Amazon RDS는 여러 위치에 인스턴스와 같은 리소스와 데이터를 배치할 수 있는 기능을 제공합니다. 리소스는 특별하게 이를 지정하지 않을 경우 AWS 리전에 복제되지 않습니다.

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



각 AWS 리전이 서로 완전히 독립적이라는 점에 유의하십시오. 사용자의 모든 Amazon RDS 활동(데이터베이스 인스턴스 또는 사용할 수 있는 데이터베이스 인스턴스 목록 생성 등)은 현재 기본 AWS 리전에서만 실행됩니다. 기본 AWS 리전은 EC2_REGION 환경 변수를 설정하여 콘솔에서 변경하거나 AWS Command Line Interface에서 --region 파라미터를 사용해 재정의할 수 있습니다. 자세한 내용은 [AWS Command Line Interface 구성](#), 특히 환경 변수 및 명령줄 옵션을 참조하십시오.

Amazon RDS는 미국 정부 기관 및 고객이 더욱 민감한 워크로드를 클라우드로 이전할 수 있도록 설계된 AWS GovCloud (US-West)라는 특별한 AWS 리전을 지원합니다. AWS GovCloud (US-West)은 미국의 특정 규제 및 규정 준수 요건을 처리합니다. AWS GovCloud (US-West)에 대한 자세한 내용은 [AWS GovCloud \(US-West\)란?](#)을 참조하십시오.

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

리전 가용성

Note

Aurora MySQL 버전 5.7 및 Aurora PostgreSQL 버전 10.7에는 브라질의 DST(일광 절약 시간제)에 대한 최근 변경 사항을 반영하는 시간대 데이터가 없습니다. 업데이트된 버전이 나올 때까지 이 문제를 해결하려면 최근에 변경된 브라질 시간대의 예상 시간이 제대로 표시되지 않을 경우 DB 클러스터의 시간대 파라미터를 재설정합니다. 다음을 수행합니다.

- 남아메리카(상파울루) 리전 –에서 시간대를 America/Fortaleza로 설정합니다.
- 남아메리카(쿠아이바) 리전 –은 시간대를 America/Manaus로 설정합니다.

시간대를 변경하려면 [DB 파라미터 그룹의 파라미터 수정 \(p. 174\)](#)을 참조하십시오.

주제

- [Aurora MySQL 리전 가용성 \(p. 4\)](#)
- [Aurora PostgreSQL 리전 가용성 \(p. 5\)](#)

Aurora MySQL 리전 가용성

다음 표에는 현재 Aurora MySQL를 사용할 수 있는 리전이 나와 있습니다.

리전 이름	리전	Endpoint	Protocol
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 (캘리포니 아 북부)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
미국 서부 (오레곤)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
아시아 태 평양(홍콩)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
아시아 태 평양(뭄바 이)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
아시아 태 평양(서울)	ap-northeast-2	rds.ap-northeast-2.amazonaws.com	HTTPS
아시아 태 평양(싱가 포르)	ap-southeast-1	rds.ap-southeast-1.amazonaws.com	HTTPS
아시아 태 평양(시드 니)	ap-southeast-2	rds.ap-southeast-2.amazonaws.com	HTTPS
아시아 태 평양(도쿄)	ap-northeast-1	rds.ap-northeast-1.amazonaws.com	HTTPS
캐나다(중 부)	ca-central-1	rds.ca-central-1.amazonaws.com	HTTPS

리전 이름	리전	Endpoint	Protocol	
중국(베이징)	cn-north-1	rds.cn-north-1.amazonaws.com.cn	HTTPS	
중국(닝샤)	cn-northwest-1	rds.cn-northwest-1.amazonaws.com.cn	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(파리)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
유럽(스톡홀름)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
중동(바레인)	me-south-1	rds.me-south-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(US)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Aurora PostgreSQL 리전 가용성

다음 표에는 현재 Aurora PostgreSQL을 사용할 수 있는 리전이 나와 있습니다.

리전 이름	리전	Endpoint	Protocol	
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	
Africa(Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS	

Amazon Aurora Aurora 사용 설명서
리전 가용성

리전 이름	리전	Endpoint	Protocol	
아시아 태평양(홍콩)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS	
아시아 태평양(뭄바이)	ap-south-1	rds.ap-south-1.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	
중국(베이징)	cn-north-1	rds.cn-north-1.amazonaws.com.cn	HTTPS	
중국(닝샤)	cn-northwest-1	rds.cn-northwest-1.amazonaws.com.cn	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(파리)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
유럽(스톡홀름)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
중동(바레인)	me-south-1	rds.me-south-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 (US)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	HTTPS	

Amazon AuroraDB 클러스터의 로컬 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 DB 클러스터의 클러스터 파라미터 그룹에서 `time_zone` 파라미터를 이 단원의 뒤에 나오는 지원되는 값 중 하나로 설정합니다. DB 클러스터에 대한 `time_zone` 파라미터를 설정하면 DB 클러스터의 모든 인스턴스가 새로운 현지 시간대를 사용하도록 변경됩니다. 다른 DB 클러스터가 동일한 클러스터 파라미터 그룹을 사용하면 해당 DB 클러스터의 모든 인스턴스도 새로운 현지 시간대를 사용하도록 변경됩니다. 클러스터 수준 파라미터에 대한 자세한 내용은 [Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 \(p. 170\)](#) 단원을 참조하십시오.

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

AWS 리전 간 복제를 사용 중인 경우 복제 마스터 DB 클러스터와 복제본이 서로 다른 파라미터 그룹을 사용합니다. 파라미터 그룹은 AWS 리전에 고유합니다. 각 인스턴스에 대해 동일한 현지 시간대를 사용하려면 복제본 마스터와 복제본 모두의 파라미터 그룹에서 `time_zone` 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 복원할 경우 복원된 DB 클러스터의 현지 시간대는 복원된 DB 클러스터의 파라미터 그룹에서 설정한 시간대입니다.

현지 시간대를 다음 표에 나열된 값 중 하나로 설정할 수 있습니다. 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
<code>Africa/Harare</code>	시간대 설정이 1903년 2월 28일 21:49:40 GMT에서 1903년 2월 28일 21:55:48 GMT까지 잘못된 값을 반환할 수 있습니다.
<code>Africa/Monrovia</code>	
<code>Africa/Nairobi</code>	시간대 설정이 1939년 12월 31일 21:30:00 GMT에서 1959년 12월 31일 21:15:15 GMT까지 잘못된 값을 반환할 수 있습니다.
<code>Africa/Windhoek</code>	
<code>America/Bogota</code>	시간대 설정이 1914년 11월 23일 04:56:16 GMT에서 1914년 11월 23일 04:56:20 GMT까지 잘못된 값을 반환할 수 있습니다.
<code>America/Caracas</code>	
<code>America/Chihuahua</code>	
<code>America/Cuiaba</code>	
<code>America/Denver</code>	
<code>America/Fortaleza</code>	DB 클러스터가 남아메리카(상파울루) 리전에 있고 최근에 변경된 브라질 시간대의 예상 시간이 제대로 표시되지 않으면 DB 클러스터의 시간대 파라미터를 <code>America/Fortaleza</code> 로 재설정합니다.
<code>America/Guatemala</code>	
<code>America/Halifax</code>	시간대 설정이 1918년 10월 27일 05:00:00 GMT에서 1918년 10월 31일 05:00:00 GMT까지 잘못된 값을 반환할 수 있습니다.

Amazon Aurora Aurora 사용 설명서
DB 클러스터의 현지 시간대

시간대	참고
America/Manaus	DB 클러스터가 남아메리카(쿠아이바) 리전에 있고 최근에 변경된 브라질 시간대의 예상 시간이 제대로 표시되지 않으면 DB 클러스터의 시간대 파라미터를 America/Manaus로 재설정합니다.
America/Matamoros	
America/Monterrey	
America/Montevideo	
America/Phoenix	
America/Tijuana	
Asia/Ashgabat	
Asia/Baghdad	
Asia/Baku	
Asia/Bangkok	
Asia/Beirut	
Asia/Calcutta	
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	
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 Aurora 연결 관리

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 로드 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.

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 로드 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.

주제

- [Aurora 엔드포인트 유형 \(p. 10\)](#)
- [Aurora 클러스터의 엔드포인트 보기 \(p. 12\)](#)
- [클러스터 엔드포인트 사용 \(p. 12\)](#)
- [리더 엔드포인트 사용 \(p. 12\)](#)
- [사용자 지정 엔드포인트 사용 \(p. 12\)](#)
- [사용자 지정 엔드포인트 만들기 \(p. 15\)](#)
- [사용자 지정 엔드포인트 보기 \(p. 17\)](#)
- [사용자 지정 엔드포인트 편집 \(p. 22\)](#)
- [사용자 지정 엔드포인트 삭제 \(p. 24\)](#)
- [사용자 지정 엔드포인트에 대한 종합 AWS CLI 예제 \(p. 25\)](#)

Aurora 엔드포인트 유형

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

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-123456789012.us-east-1.rds.amazonaws.com:3306`

리더 엔드포인트

Aurora DB 클러스터의 리더 엔드포인트는 DB 클러스터에 대한 읽기 전용 연결 시 로드 밸런싱을 지원합니다. 쿼리와 같은 읽기 작업에 리더 엔드포인트를 사용합니다. 이 엔드포인트는 읽기 전용 Aurora 복

제본에서 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 로드 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-123456789012.us-east-1.rds.amazonaws.com:3306
```

사용자 지정 엔드포인트

Aurora 클러스터의 사용자 지정 엔드포인트는 선택한 DB 인스턴스 집합을 나타냅니다. 엔드포인트에 연결하면 Aurora가 로드 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 클러스터에 대해 최대 다섯 개의 사용자 지정 엔드포인트를 만들 수 있습니다. Aurora Serverless 클러스터에는 사용자 지정 엔드포인트를 사용할 수 없습니다.

사용자 지정 엔드포인트는 DB 인스턴스의 읽기 전용 또는 읽기-쓰기 기능 외의 조건을 기반으로 로드 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 인스턴스 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-123456789012.us-east-1.rds.amazonaws.com:3306
```

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 로드 밸런싱이 필요할 수 있습니다. 이 경우에는 여러 클라이언트를 구성하여 DB 클러스터에 속한 각기 다른 Aurora 복제본에 연결한 후 읽기 워크로드를 분산 시킬 수 있습니다. Aurora PostgreSQL에 대한 장애 조치 후 인스턴스 엔드포인트를 사용하여 연결 속도를 높이는 예제는 [Amazon Aurora PostgreSQL을 사용한 빠른 장애 조치 \(p. 878\)](#) 단원을 참조하십시오. Aurora MySQL에 대한 장애 조치 후 인스턴스 엔드포인트를 사용하여 연결 속도를 높이는 예제는 [MariaDB 커넥터/J 장애 조치 지원 - Amazon Aurora 사례](#) 단원을 참조하십시오.

다음은 Aurora MySQL DB 클러스터의 DB 인스턴스 엔드포인트를 나타낸 예제입니다.

mydbinstance.123456789012.us-east-1.rds.amazonaws.com:3306

Aurora 클러스터의 엔드포인트 보기

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

AWS CLI를 사용하여 [describe-db-clusters](#) 명령의 출력에서 라이터, 리더 및 사용자 지정 엔드포인트를 봅니다. 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 API로 [DescribeDbClusterEndpoints](#) 함수를 호출하여 엔드포인트를 검색합니다.

클러스터 엔드포인트 사용

각 Aurora 클러스터에는 기본 제공되는 단일 클러스터 엔드포인트가 있고 Aurora에서 이 엔드포인트의 이름과 기타 속성을 관리하기 때문에, 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 연결됩니다. 기본 인스턴스는 테이블과 인덱스를 만들고, `INSERT` 문을 실행하며, 기타 DDL 및 DML 작업을 수행할 수 있는 유일한 DB 인스턴스입니다.

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

리더 엔드포인트 사용

Aurora 클러스터의 읽기 전용 연결에 리더 엔드포인트를 사용합니다. 이 엔드포인트는 클러스터가 쿼리 집약적인 워크로드를 처리할 수 있도록 드는 로드 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 로드 밸런싱을 적용합니다. 개별 쿼리는 로드 밸런싱하지 않습니다. 각 쿼리를 로드 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 기본 인스턴스만 있고 Aurora 복제본이 없는 경우 리더 엔드포인트는 기본 인스턴스에 연결합니다. 이 경우 이 엔드포인트를 통해 쓰기 작업을 수행할 수 있습니다.

사용자 지정 엔드포인트 사용

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 앤드포인트를 사용하면 클러스터가 커지거나 줄어들 때 CNAME 레코드를 업데이트하지 않아도 됩니다. 또한 사용자 지정 앤드포인트는 암호화된 전송 계층 보안/Secure Sockets Layer(TLS/SSL) 연결을 사용할 수 있음을 뜻합니다.

각각의 특수 목적에 DB 인스턴스를 하나씩 사용하고 인스턴스 앤드포인트에 연결하는 대신에, 특수 DB 인스턴스 그룹을 여러 개 사용할 수 있습니다. 이 경우 각 그룹에 자체 사용자 지정 앤드포인트가 있습니다. 이러한 방식으로 Aurora는 프로덕션 또는 내부 쿼리 보고나 처리 등의 작업 전용 인스턴스 간에 로드 밸런싱을 수행할 수 있습니다. 사용자 지정 앤드포인트는 클러스터 내의 각 DB 인스턴스 그룹에 로드 밸런싱과 고가용성을 제공합니다. 그룹 내의 DB 인스턴스 중 하나를 사용할 수 없게 되면, Aurora는 동일한 앤드포인트와 연결된 다른 DB 인스턴스 중 하나로 후속 사용자 지정 앤드포인트 연결을 보냅니다.

주제

- [사용자 지정 앤드포인트의 속성 지정](#) (p. 13)
- [사용자 지정 앤드포인트의 멤버십 규칙](#) (p. 13)
- [사용자 지정 앤드포인트 관리](#) (p. 14)

사용자 지정 앤드포인트의 속성 지정

사용자 지정 앤드포인트 이름의 최대 길이는 63자입니다. 다음과 같은 이름 형식을 볼 수 있습니다.

endpointName.cluster-custom-*customerDnsIdentifier*.*dnsSuffix*

사용자 지정 앤드포인트 이름에는 클러스터 이름이 포함되지 않기 때문에, 클러스터 이름을 바꿀 경우 앤드포인트 이름을 변경할 필요가 없습니다. 동일한 리전에 있는 둘 이상의 클러스터에 동일한 사용자 지정 앤드포인트 이름을 재사용할 수 없습니다. 각 사용자 지정 앤드포인트에 특정 리전 내의 사용자 ID가 소유한 클러스터 전체에서 고유한 이름을 부여합니다.

각 사용자 지정 앤드포인트의 연결 유형은 해당 앤드포인트와 연결할 수 있는 DB 인스턴스를 결정합니다. 현재 이 유형은 READER, WRITER 또는 ANY일 수 있습니다. 사용자 지정 앤드포인트 유형에는 다음 고려 사항이 적용됩니다.

- 읽기 전용 Aurora 복제본인 DB 인스턴스만 READER 사용자 지정 앤드포인트의 일부일 수 있습니다. READER 유형은 단일 마스터 복제를 사용하는 클러스터에만 적용됩니다. 그러한 클러스터에 읽기 전용 DB 인스턴스가 여러 개 포함될 수 있기 때문입니다.
- 읽기 전용 Aurora 복제본과 읽기-쓰기 기본 인스턴스는 모두 ANY 사용자 지정 앤드포인트의 일부일 수 있습니다. Aurora는 확률이 동일한 연결된 모든 DB 인스턴스로 ANY 유형의 클러스터 앤드포인트 연결을 보냅니다. 읽기 전용 Aurora 복제본의 기본 인스턴스에 연결하는 경우에는 미리 결정할 수 없으므로, 읽기 전용 연결에만 이 종류의 앤드포인트를 사용합니다. ANY 유형은 복제 토플로지를 사용하는 클러스터에 적용됩니다.
- WRITER 유형은 멀티 마스터 클러스터에만 적용됩니다. 이러한 클러스터에는 읽기-쓰기 DB 인스턴스가 여러 개 포함될 수 있기 때문입니다.
- 클러스터의 복제 구성을 기반으로 적합하지 않은 유형의 사용자 지정 앤드포인트를 만들려고 하면 Aurora가 오류를 반환합니다.

사용자 지정 앤드포인트의 멤버십 규칙

사용자 지정 앤드포인트에 DB 인스턴스를 추가하거나 사용자 지정 앤드포인트에서 제거해도, 해당 DB 인스턴스로의 기존 연결은 활성 상태로 유지됩니다.

사용자 지정 앤드포인트에 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 앤드포인트 집합이 클러스터의 모든 DB 인스턴스를 포함하도록 할 수 있습니다. 각 사용자 지정 앤드포인트는 이러한 목록 유형 중 하나만 포함할 수 있습니다.

AWS Management 콘솔에서 Attach future instances added to this cluster(이 클러스터에 추가된 향후 인스턴스 첨부) 확인란에 선택이 표시됩니다. 이 확인란을 선택하지 않으면 사용자 지정 엔드포인트가 대화 상자에 지정된 DB 인스턴스만 포함하는 정적 목록을 사용합니다. 이 확인란을 선택하면 사용자 지정 엔드포인트가 제외 목록을 사용합니다. 이 경우 사용자 지정 엔드포인트는 대화 상자에서 선택하지 않은 채로 둔 인스턴스를 제외한 클러스터의 모든 DB 인스턴스를 나타냅니다(향후 추가하는 모든 인스턴스 포함). AWS CLI 및 Amazon RDS API에는 각 목록의 종류를 나타내는 파라미터가 있습니다. AWS CLI 또는 Amazon RDS API를 사용하는 경우, 목록에 개별 멤버를 추가하거나 제거할 수 없으며, 항상 전체 새 목록을 지정합니다.

장애 조치 또는 승격으로 인해 DB 인스턴스가 기본 인스턴스와 Aurora 복제본 간에 역할을 변경해도 Aurora는 정적 목록 또는 제외 목록에 지정된 DB 인스턴스를 변경하지 않습니다. 예를 들어 READER 유형의 사용자 지정 엔드포인트에는 Aurora 복제본이었다가 이후 기본 인스턴스로 승격된 DB 인스턴스가 포함될 수 있습니다. 사용자 지정 엔드포인트 유형(READER, WRITER, 또는 ANY)에 따라 해당 엔드포인트를 통해 수행할 수 있는 작업의 종류가 결정됩니다.

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, WRITER 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 관리

새로 생성된 Aurora 클러스터에는 사용자 지정 엔드포인트가 없기 때문에, 이러한 객체를 직접 만들고 관리해야 합니다. AWS Management 콘솔, 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 콘솔에서 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 엔드포인트 만들기

콘솔

AWS Management 콘솔을 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.

Create custom endpoint

Endpoint name

Endpoint name is case insensitive.
First character must be a letter. Can

Endpoint members



Filter database

AWS Management 콘솔에서 ANY 또는 READER 유형의 사용자 지정 엔드포인트는 선택할 수 없습니다.
AWS Management 콘솔을 통해 만드는 모든 사용자 지정 엔드포인트는 ANY 유형입니다.

AWS CLI

AWS CLI를 사용하여 사용자 지정 엔드포인트를 만들려면 [create-db-cluster-endpoint 명령](#)을 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.

Linux, OS X, Unix의 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
```

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 작업](#)을 실행합니다.

사용자 지정 엔드포인트 보기

콘솔

AWS Management 콘솔을 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.

The screenshot shows a user interface for managing endpoints. At the top, it says "Endpoints (0)". Below that is a search bar with a magnifying glass icon and the placeholder text "Filter filter custom endpoints". A table header row follows, containing a single column labeled "Endpoint". The main body of the table is currently empty.

해당 클러스터의 사용자 지정 앤드포인트를 만들면 엔드포인트 섹션 아래에 표시됩니다.

Endpoints (2)



Filter filter custom endpoints

Endpoint



.cluster-custo



.cluster-custo

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

RDS > Clusters:

Details

Endpoint name

Endpoint members



20

Filter endpoint members

클러스터에 추가된 새 DB 인스턴스가 앤드포인트에도 자동으로 추가되었는지 여부에 대한 추가 정보를 보려면 앤드포인트의 편집 대화 상자를 불러옵니다.

AWS CLI

AWS CLI를 사용하여 사용자 지정 앤드포인트를 보려면 [describe-db-cluster-endpoints](#) 명령을 실행합니다.

다음 명령은 지정된 리전의 지정된 클러스터와 연결된 사용자 지정 앤드포인트를 보여 줍니다. 출력에는 기본 제공 앤드포인트와 사용자 지정 앤드포인트가 모두 포함됩니다.

Linux, OS X, Unix의 경우:

```
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. WRITER 또는 READER의 `EndpointType`은 클러스터의 기본 제공 읽기-쓰기 및 읽기 전용 앤드포인트를 나타냅니다. CUSTOM의 `EndpointType`은 연결된 DB 인스턴스를 생성하고 선택한 앤드포인트를 나타냅니다. 앤드포인트 중 하나의 비어 있지 않은 `StaticMembers` 필드는 정확한 DB 인스턴스 집합과 연결되었음을 나타냅니다. 다른 앤드포인트의 비어 있지 않은 `ExcludedMembers` 필드는 앤드포인트가 `ExcludedMembers` 아래에 나열된 인스턴스 이외의 모든 DB 인스턴스와 연결되어 있음을 나타냅니다. 이 두 번째 종류의 사용자 지정 앤드포인트는 클러스터에 새 인스턴스를 추가할 때 이 새 인스턴스를 포함하도록 확장됩니다.

```
{
  "DBClusterEndpoints": [
    {
      "Endpoint": "custom-endpoint-demo.cluster-123456789012.ca-central-1.rds.amazonaws.com",
      "Status": "available",
      "DBClusterIdentifier": "custom-endpoint-demo",
      "EndpointType": "WRITER"
    },
    {
      "Endpoint": "custom-endpoint-demo.cluster-ro-123456789012.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-123456789012.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 \(p. 13\)](#)을 참조하십시오.

편집 작업으로 인한 변경이 진행 중인 동안에는 사용자 지정 엔드포인트에 연결하거나 사용할 수 없습니다. 엔드포인트 상태가 Available(사용 가능)로 돌아오고 다시 연결할 수 있으려면 몇 분이 걸릴 수 있습니다.

콘솔

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

RDS > Clusters: [REDACTED]

Edit endpoint:

Endpoint members



Filter database



DB instance name



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.

Linux, OS X, Unix의 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
```

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
```

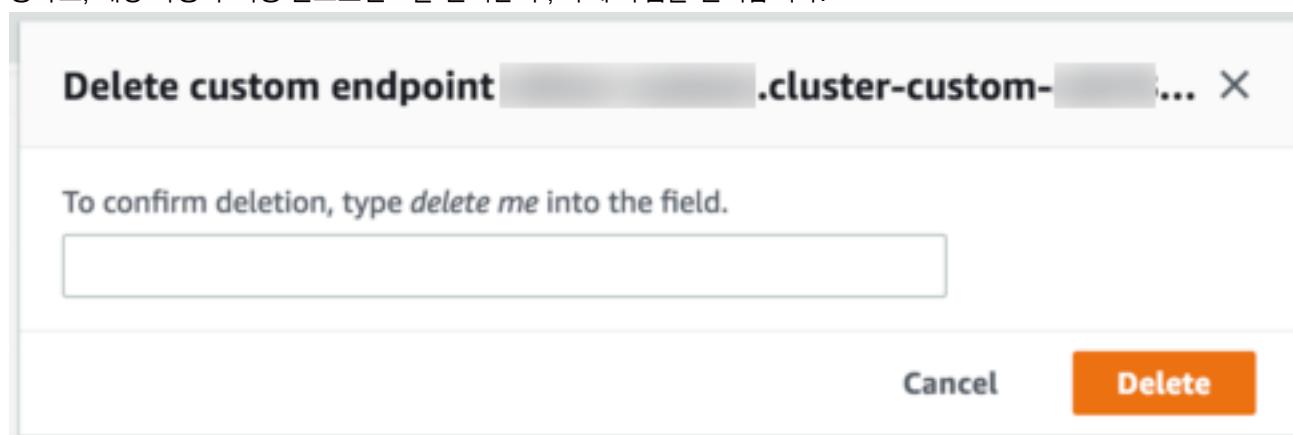
RDS API

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

사용자 지정 엔드포인트 삭제

콘솔

AWS Management 콘솔을 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.

Linux, OS X, Unix의 경우:

```
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.

사용자 지정 엔드포인트에 대한 종합 AWS CLI 예제

다음 자습서에서는 Unix 셸 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 있음을 보여 줍니다. 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.r4.4xlarge를 사용하지만, 마지막 두 DB 인스턴스는 db.r4.16xlarge를 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 \  
--engine-version 5.6.10a --master-username $MASTER_USER --master-user-password  
$MASTER_PW \  
--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.

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

리소스를 가장 많이 사용하는 쿼리에만 "더 큰" 인스턴스 두 개를 사용하려고 한다고 가정하겠습니다. 사용자 지정 읽기 전용 앤드포인트를 만든 후, 앤드포인트가 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, ANY 유형 대신에 READER 유형으로 지정했으므로 이 앤드포인트를 통해 해당 인스턴스에 접근할 수 없게 됩니다. 다음 예제에서는 앤드포인트 생성 및 수정 명령과, 사용자 지정 앤드포인트의 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-123456789012.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-123456789012.ca-central-1.rds.amazonaws.com",
    "DBClusterIdentifier": "custom-endpoint-demo"
}
```

클러스터의 기본 READER 앤드포인트는 "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 앤드포인트를 무시하고 다른 모든 DB 인스턴스에 연결되는 두 번째 사용자 지정 앤드포인트를 만들면 됩니다. 다음 예제에서는 사용자 지정 앤드포인트를 만든 후 제외 목록을 추가하여 이를 수행

합니다. 클러스터에 나중에 추가하는 다른 모든 DB 인스턴스는 이 앤드포인트에 자동으로 추가됩니다. ANY 유형은 이 앤드포인트가 총 여덟 개의 인스턴스(기본 인스턴스 하나와 Aurora 복제본 일곱 개)와 연결되어 있음을 뜻합니다. 이 예제에서 READER 유형을 사용한 경우, 사용자 지정 앤드포인트는 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-123456789012.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-1:111122223333:cluster-endpoint:small-instances",
    "DBClusterEndpointResourceIdentifier": "cluster-endpoint-6RDDXQOC3AKKZT2PRD7ST37BMY",
    "CustomEndpointType": "ANY",
    "Endpoint": "small-instances.cluster-custom-123456789012.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 앤드포인트 상태를 확인합니다. 클러스터에 여전히 EndpointType이 WRITER인 원래 클러스터 앤드포인트가 있으며, 관리, ETL 및 기타 쓰기 작업에 여전히 이 앤드포인트를 사용합니다. 여전히 원래 READER 앤드포인트가 있는데, 이 앤드포인트로의 각 연결을 "작거나" "큰" DB 인스턴스로 보낼 수 있으므로 이 인스턴스를 사용하지 않습니다. 사용자 지정 앤드포인트는 이 동작을 예측 가능하게 합니다. 지정한 앤드포인트를 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-123456789012.ca-central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        },
        {
            "EndpointType": "READER",
            "Endpoint": "small-instances.cluster-custom-123456789012.ca-central-1.rds.amazonaws.com",
            "Status": "available",
            "DBClusterIdentifier": "custom-endpoint-demo"
        }
    ]
}
```

```

        "EndpointType": "READER",
        "Endpoint": "custom-endpoint-demo.cluster-ro-123456789012.ca-
central-1.rds.amazonaws.com",
        "Status": "available",
        "DBClusterIdentifier": "custom-endpoint-demo"
    },
    {
        "Endpoint": "small-instances.cluster-custom-123456789012.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-123456789012.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 앤드포인트로의 연속적인 데이터베이스 연결이 Aurora 클러스터의 다양한 DB 인스턴스에 연결되는 방법을 보여 줍니다. small-instances 앤드포인트는 항상 이 클러스터에서 낮은 번호의 호스트인 db.r4.4xlarge DB 인스턴스에 연결됩니다.

```

$ mysql -h small-instances.cluster-custom-123456789012.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-123456789012.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-123456789012.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-123456789012.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-123456789012.ca-central-1.rds.amazonaws.com -u $MYUSER -p
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id      |
+-----+
| custom-endpoint-demo-09 |
+-----+
```

인스턴스 엔드포인트 사용

일상적인 운영에서 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.

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.

Aurora 엔드포인트가 고가용성으로 작동하는 방법

고가용성이 중요한 클러스터인 경우 읽기-쓰기 연결에 라이터 엔드포인트를, 읽기 전용 연결에 리더 엔드포인트를 사용합니다. 이러한 종류의 연결은 인스턴스 엔드포인트보다 DB 인스턴스 장애 조치를 더 잘 관리합니다. 인스턴스 엔드포인트는 DB 클러스터의 특정 DB 인스턴스에 연결되므로, 해당 DB 인스턴스를 사용할 수 없게 된 경우 애플리케이션의 로직이 다른 엔드포인트를 선택해야 합니다.

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 클러스터에서 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. 그런 다음 장애 조치 이후 인스턴스 클래스를 확인하고 해당 인스턴스 엔드포인트에 연결하면 됩니다.

장애 조치에 대한 자세한 내용은 [Aurora DB 클러스터의 내결함성 \(p. 268\)](#) 단월을 참조하십시오.

DB 인스턴스 클래스

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

인스턴스 클래스 요금에 대한 자세한 내용은 [Amazon RDS 요금](#)을 참조하십시오.

주제

- [DB 인스턴스 클래스 유형 \(p. 30\)](#)
- [DB 인스턴스 클래스 하드웨어 사양 관련 용어 \(p. 31\)](#)
- [Aurora에 사용 가능한 모든 DB 인스턴스 클래스의 하드웨어 사양 \(p. 31\)](#)

DB 인스턴스 클래스 유형

Amazon Aurora는 메모리 최적화 및 버스트 성능의 두 가지 인스턴스 클래스 유형을 지원합니다. Amazon EC2 인스턴스 유형에 대한 자세한 내용은 Amazon EC2 설명서의 [인스턴스 유형](#)을 참조하십시오.

다음은 사용 가능한 메모리 최적화 DB 인스턴스 클래스입니다.

- db.r5 – 메모리 집약적 애플리케이션에 최적화된 최신 세대 인스턴스 클래스입니다. 향상된 네트워킹 성능을 제공합니다. 전용 하드웨어 및 경량 하이퍼바이저 결합된 AWS Nitro System을 기반으로 합니다.
- db.r4 – 메모리 집약적 애플리케이션에 최적화된 현재 세대 인스턴스 클래스 향상된 네트워킹과 성능을 제공합니다.
- db.r3 – 메모리를 최적화하는 전세대 인스턴스 클래스 db.r3 인스턴스 클래스는 유럽(파리) 리전에서는 사용할 수 없습니다.

다음은 사용 가능한 버스트 가능 성능 DB 인스턴스 클래스입니다.

- db.t3 – CPU 사용률을 최대로 버스트할 수 있는 기능을 통해 기준 성능 수준을 제공하는 최신 세대 인스턴스 클래스입니다. 이 인스턴스 클래스는 이전의 db.t2 인스턴스 클래스보다 더 많은 컴퓨팅 용량을 제공합니다. 전용 하드웨어 및 경량 하이퍼바이저 결합된 AWS Nitro System을 기반으로 합니다.
- db.t2 – CPU 사용률을 최대로 버스트할 수 있는 기능을 통해 기준 성능 수준을 제공하는 현재 세대 인스턴스 클래스입니다. 이 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비 프로덕션 서버에만 사용하는 것 이 좋습니다.

Note

AWS Nitro System을 사용하는 DB 인스턴스 클래스(db.r5, db.t3)는 결합된 읽기 및 쓰기 워크로드에 대해 조절됩니다.

DB 인스턴스 클래스 하드웨어 사양 관련 용어

다음 용어는 DB 인스턴스 클래스의 하드웨어 사양을 기술하는 데 사용됩니다.

- vCPU – 가상 CPU(중앙 처리 디바이스)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 ECU로 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 대역폭(Mbps) – 초당 메가비트 단위로 최대 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 성능 – 다른 DB 인스턴스 클래스 대비 네트워크 속도입니다.

Aurora에 사용 가능한 모든 DB 인스턴스 클래스의 하드웨어 사양

아래 표에서 Amazon Aurora에 사용할 수 있는 Amazon RDS DB 인스턴스 클래스의 세부 정보를 확인하실 수 있습니다. 표 열에 있는 용어에 대한 자세한 설명은 [DB 인스턴스 클래스 하드웨어 사양 관련 용어 \(p. 31\)](#) 단원을 참조하십시오.

다음은 DB 인스턴스 클래스에 대한 DB 엔진 고려 사항입니다.

- Aurora db.r5 지원 – 여기 나온 인스턴스는 AWS GovCloud (US-West), AWS GovCloud(US-East) 및 중국(베이징)를 제외한 모든 Aurora 리전에서 사용할 수 있습니다.
 - 아래 표에 지정된 것처럼 Aurora MySQL 버전에서는 db.r5 인스턴스 클래스를 지원합니다.
 - Aurora PostgreSQL의 경우, PostgreSQL 9.6.12 이상 또는 PostgreSQL 10.6 이상과 호환되는 버전만 db.r5 인스턴스 클래스를 지원합니다.
- Aurora db.t3 지원
 - Aurora MySQL은 Aurora MySQL 1.15 이상 버전 및 모든 Aurora MySQL 2.x 버전에서 db.t3.medium 및 db.t3.small 인스턴스 클래스를 지원합니다. 이 인스턴스 클래스는 AWS GovCloud (US-West), AWS GovCloud(US-East) 및 중국(베이징)을 제외한 모든 Aurora 리전에서 Aurora MySQL에 사용할 수 있습니다.
 - Aurora MySQL db.r5, db.r4 및 db.t3 DB 인스턴스 클래스의 경우 클러스터의 어떤 인스턴스도 대기 중 인스턴스 수준 시스템 업데이트를 가질 수 없습니다. 대기 중 시스템 업데이트를 보려면 다음 AWS CLI 명령을 사용합니다.

```
aws rds describe-pending-maintenance-actions
```

- Aurora PostgreSQL은 PostgreSQL 10.7 이상과 호환되는 버전에서 db.t3.medium 인스턴스 클래스만 지원합니다. 여기 나온 인스턴스 클래스는 중국(닝샤)을 제외한 모든 Aurora 리전에서 Aurora PostgreSQL에 사용할 수 있습니다.

Amazon Aurora Aurora 사용 설명서
하드웨어 사양

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	최대 로컬 스토리지의 대역폭 (Mbps)	네트워크 성능	Aurora MySQL	Aurora PostgreSQL
db.r5 – 최신 세대 메모리 최적화 인스턴스 클래스							
db.r5.24xlarge	96	347	768	19,000	25Gbps	1.22 이상, 2.06 이상	예
db.r5.16xlarge	64	264	512	13,600	20Gbps	1.22 이상, 2.06 이상	아니요
db.r5.12xlarge	48	173	384	9,500	10Gbps	1.14.4 이상	예
db.r5.8xlarge	32	132	256	6,800	10Gbps	1.22 이상, 2.06 이상	아니요
db.r5.4xlarge	16	71	128	4,750	최대 10Gbps	1.14.4 이상	예
db.r5.2xlarge*	8	38	64	최대 4,750 개	최대 10Gbps	1.14.4 이상	예
db.r5.xlarge*	4	19	32	최대 4,750 개	최대 10Gbps	1.14.4 이상	예
db.r5.large*	2	10	16	최대 4,750 개	최대 10Gbps	1.14.4 이상	예
db.r4 – 현재 세대 메모리 최적화 인스턴스 클래스							
db.r4.16xlarge	64	195	488	14,000	25Gbps	1.14.4 이상	예
db.r4.8xlarge	32	99	244	7,000	10Gbps	1.14.4 이상	예
db.r4.4xlarge	16	53	122	3,500	최대 10Gbps	1.14.4 이상	예
db.r4.2xlarge	8	27	61	1,700	최대 10Gbps	1.14.4 이상	예
db.r4.xlarge	4	13.5	30.5	850	최대 10Gbps	1.14.4 이상	예
db.r4.large	2	7	15.25	425	최대 10Gbps	1.14.4 이상	예
db.r3 – 전세대 메모리 최적화 인스턴스 클래스							
db.r3.8xlarge	32	104	244	—	10Gbps	예	아니요

인스턴스 클래스	vCPU	ECU	메모리 (GiB)	최대 로컬 스토리지의 대역폭 (Mbps)	네트워크 성능	Aurora MySQL	Aurora PostgreSQL
db.r3.4xlarge	16	52	122	2,000	높음	예	아니요
db.r3.2xlarge	8	26	61	1,000	높음	예	아니요
db.r3.xlarge	4	13	30.5	500	중간	예	아니요
db.r3.large	2	6.5	15.25	—	중간	예	아니요
db.t3 – 최신 세대 버스트 성능 인스턴스 클래스							
db.t3.2xlarge*	8	변수	32	최대 2,048 개	최대 5Gbps	아니요	아니요
db.t3.xlarge*	4	변수	16	최대 2,048 개	최대 5Gbps	아니요	아니요
db.t3.large*	2	변수	8	최대 2,048 개	최대 5Gbps	아니요	아니요
db.t3.medium*	2	변수	4	최대 1,536 개	최대 5Gbps	1.14.4 이상	10.7 이상
db.t3.small*	2	변수	2	최대 1,536 개	최대 5Gbps	1.14.4 이상	아니요
db.t3.micro*	2	변수	1	최대 1,536 개	최대 5Gbps	아니요	아니요
db.t2 – 현재 세대 버스트 성능 인스턴스 클래스							
db.t2.medium	2	변수	4	—	중간	예	아니요
db.t2.small	1	변수	2	—	낮음	예	아니요

* 이러한 DB 인스턴스 클래스에서는 24시간마다 최소 한 번씩 30분간 최대 성능을 지원합니다. 이러한 인스턴스 유형의 기본 성능에 대한 자세한 내용은 Linux 인스턴스용 Amazon EC2 사용 설명서의 [Amazon EBS 최적화 인스턴스](#)를 참조하십시오.

Amazon Aurora 스토리지 및 안정성

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

주제

- [Aurora 스토리지 개요 \(p. 34\)](#)
- [클러스터 볼륨에 포함된 항목 \(p. 34\)](#)
- [Aurora 스토리지가 확장되는 방법 \(p. 34\)](#)
- [Aurora 데이터 스토리지 요금이 청구되는 방법 \(p. 34\)](#)
- [Amazon Aurora 안정성 \(p. 34\)](#)

Aurora 스토리지 개요

Aurora SSD(Solid State Drive)를 사용하는 단일 가상 볼륨인 클러스터 볼륨에 저장됩니다. 클러스터 볼륨은 동일한 AWS 리전에 속한 다중 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.

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.

Aurora 스토리지가 확장되는 방법

데이터베이스의 데이터 용량이 늘어날수록 Aurora 클러스터 볼륨도 자동 확장됩니다. Aurora 클러스터 볼륨 크기는 최대 64 tebibytes (TiB)까지 증가할 수 있습니다. 테이블 크기는 클러스터 볼륨 크기로 제한됩니다. 즉 Aurora DB 클러스터에 있는 테이블의 최대 크기는 64 TiB입니다.

이 자동 스토리지 확장과 고성능의 고도로 분산된 스토리지 하위 시스템 덕분에, Aurora는 주요 목표가 안정성과 고가용성인 중요한 엔터프라이즈 데이터에 적합합니다. 스토리지 비용과 이러한 기타 우선 순위의 균형을 조정하는 방법은 다음 단원을 참조하십시오.

Aurora 데이터 스토리지 요금이 청구되는 방법

Aurora 클러스터 볼륨은 최대 64 TiB까지 증가할 수 있지만 요금은 Aurora 클러스터 볼륨에서 사용한 공간에 대해서만 청구됩니다. 테이블 또는 파티션을 삭제하는 등으로 Aurora 데이터가 제거될 때 전체 할당 공간은 동일하게 유지됩니다. 사용 가능한 공간은 향후에 데이터 볼륨이 증가할 때 자동으로 재사용됩니다.

Note

스토리지 비용은 스토리지 "하이 워터 마크"(어떤 시점에서 Aurora 클러스터에 대해 할당 완료된 최대 금액)를 기반으로 하기 때문에 불필요한 이전 데이터를 제거하기 전에 대량의 새 데이터를 로드하거나 큰 볼륨의 임시 정보를 생성하는 ETL 사례를 방지하여 비용을 관리할 수 있습니다.

Aurora 클러스터에서 데이터를 제거하면 많은 양이 할당되었지만 사용되지 않은 공간이 발생하는 경우 하이 워터 마크를 재설정할 때 mysqldump과 같은 도구를 사용하여 논리적 데이터 덤프를 수행하고 새 클러스터로 복원해야 합니다. 스냅샷을 생성하고 복원하면 기본 스토리지의 물리적 레이아웃이 복원된 스냅샷에서 동일하게 유지되기 때문에 할당된 스토리지가 감소하지 않습니다.

Aurora 데이터 스토리지에 대한 요금 정보는 [Aurora용 Amazon RDS 요금](#)을 참조하십시오.

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 \(p. 35\)](#)
- [유지 가능한 캐시 워밍 \(p. 35\)](#)
- [충돌 복구 \(p. 35\)](#)

스토리지 자동 복구

Aurora은 3개의 가용 영역에 여러 개의 복사본을 보관하고 있기 때문에 디스크 결함으로 인한 데이터 손실 가능성이 최소화됩니다. Aurora는 클러스터 볼륨을 구성하고 있는 디스크 볼륨에서 결함을 자동 감지합니다. 예를 들어 디스크 볼륨 세그먼트에 결함이 발생하면 Aurora가 즉시 해당 세그먼트를 복구합니다. Aurora가 디스크 세그먼트를 복구할 때는 동일한 클러스터 볼륨을 구성하는 나머지 디스크 볼륨의 데이터를 사용하기 때문에 복구 세그먼트의 데이터도 이용 가능합니다. 결과적으로 Aurora는 데이터 손실을 방지할 뿐만 아니라 특정 시점으로 복구 기능을 사용해 디스크 결함을 복구할 필요성도 줄어듭니다.

유지 가능한 캐시 워밍

Aurora는 전원이 꺼진 데이터베이스를 가동하거나 결함 발생 이후 다시 시작할 때 버퍼풀 캐시를 "워밍"합니다. 즉, Aurora가 인 메모리 페이지 캐시에 저장된 기존 공통 쿼리 페이지를 이용해 버퍼풀을 미리 로드합니다. 이 경우 일반적인 데이터베이스 사용과 달리 버퍼풀의 "웜업" 필요성을 우회할 수 있기 때문에 성능이 향상되는 이점이 있습니다.

Aurora 페이지 캐시는 데이터베이스가 아닌 별도의 프로세스로 관리되기 때문에 데이터베이스와 상관없이 유지됩니다. 예상과 달리 데이터베이스에 결함이 발생하더라도 페이지 캐시가 메모리에 남아있기 때문에 데이터베이스를 재시작할 때 버퍼풀이 가장 최신 상태로 워밍됩니다.

충돌 복구

Aurora은 거의 순간적으로 충돌에서 복구하고 바이너리 로그없이 애플리케이션 데이터를 계속 제공하도록 설계되었습니다. Aurora는 병렬 스레드에서 비동기 방식으로 충돌 복구를 실행하므로 충돌 직후에도 데이터베이스를 열어두고 사용할 수 있습니다.

충돌 복구에 대한 자세한 내용은 [Aurora DB 클러스터의 내결함성 \(p. 268\)](#) 단원을 참조하십시오.

다음은 Aurora MySQL에서의 바이너리 로깅 및 충돌 복구 시 고려 사항입니다.

- Aurora 바이너리 로깅을 활성화하면 DB 인스턴스로 하여금 강제로 바이너리 로그 복구를 수행하도록 하므로 충돌 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.

- Aurora는 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 OFF로 설정하여 이진 로깅을 비활성화하는 것이 좋습니다. 이렇게 하면 복구 시간이 단축됩니다.

Aurora 이진 로깅 및 복제에 대한 자세한 내용은 [Amazon Aurora를 사용한 복제 \(p. 55\)](#) 단원을 참조하십시오. 다양한 MySQL 복제 유형의 영향에 대한 자세한 내용은 MySQL 설명서의 [Advantages and Disadvantages of Statement-Based and Row-Based Replication](#)을 참조하십시오.

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)을 사용합니다. IAM 자격 증명을 사용하여 AWS에 연결할 때는 Amazon RDS 관리 작업에 필요한 권한을 부여할 수 있는 IAM 정책이 IAM 계정에 반드시 필요합니다. 자세한 내용은 [Amazon Aurora의 Identity and Access Management\(IAM\) \(p. 960\)](#) 단원을 참조하십시오.

IAM 계정을 사용해 Amazon RDS 콘솔에 액세스하려면 먼저 IAM 계정으로 AWS Management 콘솔에 로그인한 다음 Amazon RDS에서 <https://console.aws.amazon.com/rds> 콘솔로 이동합니다.

- Aurora DB 클러스터는 Amazon Virtual Private Cloud(VPC)에서 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 Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#) 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 \(p. 466\)](#) 또는 [Amazon Aurora PostgreSQL를 사용한 보안 \(p. 773\)](#) 단원을 참조하십시오.

- 또한 Aurora MySQL에 IAM 데이터베이스 인증을 사용할 수도 있습니다.

IAM 데이터베이스 인증의 경우, IAM 사용자 또는 IAM 역할 및 인증 토큰을 이용해 Aurora MySQL 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 \(p. 976\)](#) 단원을 참조하십시오.

구성 보안에 대한 자세한 내용은 [Amazon Aurora의 보안 \(p. 943\)](#) 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 \(p. 466\)](#), [Amazon Aurora PostgreSQL를 사용한 보안 \(p. 773\)](#) 또는 [Aurora Serverless에서 TLS/SSL 사용 \(p. 114\)](#) 단원을 참조하십시오.

Aurora을 위한 고가용성

Aurora 아키텍처는 스토리지와 컴퓨팅을 분리합니다. 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 기능을 통해 하나 이상의 DB 인스턴스가 애플리케이션의 데이터베이스 요청을 처리할 수 있습니다.

Aurora는 DB 클러스터의 인스턴스가 여러 가용 영역에 걸쳐 있는지 여부와 상관없이, 단일 AWS 리전의 여러 가용 영역에 걸쳐 있는 DB 클러스터에 데이터의 복사본을 저장합니다. Aurora에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 관리 \(p. 190\)](#) 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.

다. 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 \(p. 3\)](#) 단원을 참조하십시오.

단일 마스터 복제를 사용하는 클러스터에서는 기본 인스턴스를 생성한 후에 최대 15개까지 Aurora 복제본을 생성할 수 있습니다. 이러한 읽기 전용 DB 인스턴스는 읽기 집약적인 애플리케이션에서 SELECT 쿼리를 지원합니다. DB 클러스터의 가용성을 향상하려면 복수의 가용 영역의 DB 클러스터에 기본 인스턴스와 Aurora Replicas를 분배하는 것이 바람직합니다. 가용 영역에서 리더 인스턴스를 만들면 Amazon RDS가 자동으로 해당 인스턴스를 프로비저닝하고 기본 인스턴스와 함께 최신 상태로 유지합니다.

RDS 콘솔을 사용하여 DB 클러스터 생성 시 다중 AZ를 지정함으로써 간편하게 다중 AZ 배포를 생성할 수 있습니다. DB 클러스터가 단일 가용 영역에 있는 경우에는 다중 AZ DB 클러스터가 다른 가용 영역에 또 다른 DB 인스턴스를 추가하도록 할 수 있습니다.

CLI를 사용하여 다중 AZ 배포를 지정할 수도 있습니다. AWS CLI [describe-db-instances](#) 명령 또는 Amazon RDS API [DescribeDBInstances](#) 작업을 사용하여 예비 복제본의 가용 영역(보조 AZ)을 볼 수 있습니다.

자세한 내용은 [리전 가용성 \(p. 3\)](#) 단원을 참조하십시오. [create-db-instance](#) AWS CLI 명령을 호출하여 DB 클러스터에서 Aurora 복제본을 생성하십시오. DB 클러스터의 이름을 --db-cluster-identifier 파라미터 값으로 포함합니다. 선택에 따라 --availability-zone 파라미터를 사용하여 Aurora 복제본의 가용 영역을 지정할 수 있습니다.

Aurora 복제본으로의 장애 조치에 대한 자세한 정보는 [Amazon Aurora 연결 관리 \(p. 9\)](#) 단원을 참조하십시오. DB 클러스터 생성에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

Amazon Aurora 글로벌 데이터베이스 작업

아래에서 Amazon Aurora 글로벌 데이터베이스에 대한 설명을 찾아볼 수 있습니다. 각각의 Aurora 글로벌 데이터베이스는 여러 AWS 리전에 걸쳐 있어, 지연 시간이 짧은 전역 읽기와 리전 전체 중단에 대한 재해 복구를 지원합니다.

주제

- [Aurora 글로벌 데이터베이스 개요 \(p. 37\)](#)
- [Aurora 글로벌 데이터베이스 생성 \(p. 39\)](#)
- [Aurora 글로벌 데이터베이스에 AWS 리전 추가 \(p. 46\)](#)
- [Aurora 글로벌 데이터베이스에서 클러스터 제거 \(p. 48\)](#)
- [Aurora 글로벌 데이터베이스 삭제 \(p. 50\)](#)
- [Aurora 글로벌 데이터베이스로 데이터 가져오기 \(p. 51\)](#)
- [Aurora 글로벌 데이터베이스에 연결 \(p. 52\)](#)
- [Aurora 글로벌 데이터베이스에 대한 장애 조치 \(p. 53\)](#)
- [Aurora Global Database를 위한 성능 개선 도우미 \(p. 53\)](#)
- [Aurora 글로벌 데이터베이스에서 여러 개의 보조 리전 사용 \(p. 54\)](#)
- [다른 AWS 서비스와 함께 Aurora 글로벌 데이터베이스 사용 \(p. 54\)](#)

Aurora 글로벌 데이터베이스 개요

Aurora Global Database는 데이터가 마스터로 사용되는 기본 AWS 리전 1개와 최대 5개의 읽기 전용 보조 AWS 리전으로 이루어집니다. Aurora는 일반적으로 1초 미만의 지연 시간으로 보조 AWS 리전으로 데이터를 복제합니다. 쓰기 연산을 기본 AWS 리전의 기본 DB 인스턴스에 직접 발행합니다. Aurora 글로벌 데이터베이스에서는 전용 인프라를 사용해 데이터를 복제합니다. 이를 통해 데이터베이스 리소스가 애플리케이션

워크로드를 처리하는 데 전적으로 사용할 수 있게 됩니다. 전 세계적으로 사용되는 애플리케이션은 읽기 지연 시간을 줄이기 위해 보조 AWS 리전에서 리더 인스턴스를 사용할 수 있습니다. 흔히 발생하지는 않지만 데이터베이스가 AWS 리전에서 성능이 저하되는 경우에는 1분 안에 모든 읽기-쓰기 워크로드를 처리할 수 있도록 보조 AWS 리전 중 하나를 승격시킬 수 있습니다.

데이터가 마스터로 사용되는 기본 AWS 리전의 Aurora 클러스터는 읽기 및 쓰기 작업을 둘 다 수행합니다. 보조 리전의 클러스터는 낮은 지연 시간 읽기를 활성화합니다. 다른 DB 인스턴스 중 하나(Aurora 복제본)를 추가하여 읽기 전용 워크로드를 처리함으로써 보조 클러스터를 독립적으로 확장할 수 있습니다. 재해 복구 시에는 Aurora Global Database에서 보조 클러스터를 제거한 후 전체 읽기 및 쓰기 작업을 허용하도록 승격할 수 있습니다.

기본 클러스터만 쓰기 작업을 수행합니다. 쓰기 작업을 수행하는 클라이언트는 기본 클러스터의 DB 클러스터 엔드포인트에 연결합니다.

주제

- [Aurora 글로벌 데이터베이스의 장점 \(p. 38\)](#)
- [Aurora Global Database에 적용되는 제한 사항 \(p. 38\)](#)

Aurora 글로벌 데이터베이스의 장점

Aurora 글로벌 데이터베이스는 다음과 같은 장점을 제공합니다.

- 보조 클러스터는 단일 Aurora 클러스터의 용량을 훨씬 능가하는 읽기 확장을 제공합니다.
각 보조 클러스터에는 단일 Aurora 클러스터에 대한 15개의 제한이 아닌 최대 16개의 읽기 전용 Aurora 복제본 인스턴스가 있을 수 있습니다.
- Aurora Global Database에서 수행되는 복제는 기본 DB 클러스터의 성능에 거의 영향을 미치지 않습니다. DB 인스턴스의 리소스는 애플리케이션 읽기/쓰기 워크로드 처리 전용입니다.
- 일반적으로 1초 미만의 최소 지연 시간으로 변경 내용이 AWS 리전 간에 복제됩니다.
- 보조 클러스터를 사용하면 재해 복구를 위한 빠른 장애 조치가 가능합니다. 일반적으로 보조 클러스터를 승격하여 1분 안에 쓰기 가능하도록 설정할 수 있습니다.
- 원격 AWS 리전의 애플리케이션은 보조 클러스터에서 읽을 때 큐리 지연 시간이 더 낮습니다.

Aurora Global Database에 적용되는 제한 사항

현재 Aurora 글로벌 데이터베이스에 적용되는 제한 사항은 다음과 같습니다.

- Aurora Global Database는 다음 버전에서 사용할 수 있습니다.
 - Aurora MySQL를 선택하십시오. MySQL 5.6과 호환되는 Aurora MySQL의 경우 버전 5.6.10a 또는 버전 1.22 이상을 사용할 수 있습니다. MySQL 5.7과 호환되는 Aurora MySQL의 경우 버전 2.07 이상을 사용할 수 있습니다.
 - PostgreSQL 10.11과 호환되는 Aurora PostgreSQL
- Aurora Global Database에 대해 db.r4 또는 db.r5 인스턴스 클래스를 사용할 수 있습니다. db.t2 또는 db.t3 인스턴스 클래스는 사용할 수 없습니다.
- Aurora Global Database는 다음 리전에서 사용할 수 있습니다.
 - 미국 동부(버지니아 북부)
 - 미국 동부(오하이오)
 - 미국 서부(캘리포니아 북부 지역)
 - 미국 서부(오레곤)
 - 유럽(아일랜드)
 - 유럽(런던)
 - 유럽(파리)

- 유럽(프랑크푸르트)
- 아시아 태평양(뭄바이)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 아시아 태평양(서울)
- 캐나다(중부)
- 보조 클러스터는 기본 클러스터와는 다른 AWS 리전에 있어야 합니다.
- 보조 클러스터와 동일한 리전에는 기본 클러스터의 Aurora MySQL 리전 간 읽기 전용 복제본을 생성할 수 없습니다. 리전 간 읽기 전용 복제본에 대한 자세한 내용은 [여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제 \(p. 557\)](#) 단원을 참조하십시오.
- 글로벌 데이터베이스 클러스터를 업그레이드하려면 기본 클러스터보다 먼저 보조 클러스터를 업그레이드 해야 합니다. 업그레이드에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 또는 [Aurora PostgreSQL용 PostgreSQL DB 엔진 업그레이드 \(p. 872\)](#)를 참조하십시오.

다음 기능은 Aurora 글로벌 데이터베이스에서 지원되지 않습니다.

- 복제. 복제에 대한 자세한 내용은 [Aurora DB 클러스터에서 데이터베이스 복제 \(p. 239\)](#) 단원을 참조하십시오.
- 역추적. 역추적에 대한 자세한 내용은 [Aurora DB 클러스터 역추적 \(p. 509\)](#) 단원을 참조하십시오.
- 병렬 쿼리. 병렬 쿼리에 대한 자세한 내용은 [Amazon Aurora MySQL용 Parallel Query 처리 \(p. 527\)](#) 단원을 참조하십시오.
- Aurora Serverless를 선택하십시오. Aurora Serverless에 대한 자세한 내용은 [사용 Amazon Aurora Serverless \(p. 111\)](#) 단원을 참조하십시오.
- 데이터베이스 활동 스트림. 데이터베이스 활동 스트림에 대한 자세한 내용은 [Aurora PostgreSQL에서 데이터베이스 활동 스트림 사용 \(p. 412\)](#) 단원을 참조하십시오.
- 글로벌 데이터베이스 내에 있는 DB 클러스터 종지 및 시작. Aurora 종지 및 시작에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 종지 및 시작 \(p. 190\)](#) 단원을 참조하십시오.

Aurora 글로벌 데이터베이스 생성

Aurora 글로벌 데이터베이스는 여러 AWS 리전에 걸쳐 있습니다. 먼저 읽기-쓰기 기본 클러스터와 함께 글로벌 데이터베이스 자체를 생성합니다. 그런 다음 다른 AWS 리전에서 읽기 전용 보조 클러스터를 하나 이상 추가합니다.

기본 및 보조 클러스터의 이름을 지정할 때는 기존 Aurora 클러스터가 다른 AWS 리전에 있더라도 이와는 다른 이름을 선택하십시오.

Important

Aurora 글로벌 데이터베이스를 생성하기 전에 Amazon Relational Database Service 사용 설명서의 [Amazon RDS 설정](#)의 계정, 네트워크 및 보안 설정 관련 설정 작업을 완료합니다.

콘솔

콘솔을 사용하여 Aurora MySQL 버전 5.6.10a용 글로벌 데이터베이스를 생성하려면 두 번째 프로세스를 참조하십시오.

AWS Management 콘솔을 사용하여 Aurora 글로벌 데이터베이스를 생성하려면

1. 기존 Aurora 클러스터로 시작하거나 새 클러스터를 생성합니다. Aurora MySQL 버전 1.22.0 이상, Aurora MySQL 버전 2.07 이상 또는 Aurora PostgreSQL 버전 10.11을 사용합니다. 클러스터 생성에 대한 자세한 내용은 [DB 클러스터 생성 및 Aurora MySQL DB 클러스터의 데이터베이스에 연결 \(p. 74\)](#)

또는 [DB 클러스터 생성 및 Aurora PostgreSQL DB 클러스터의 데이터베이스에 연결 \(p. 83\)](#)을 참조하십시오.

2. Aurora 클러스터가 생성되어 사용 가능해지면 하나 이상의 보조 클러스터를 생성합니다. 이렇게 하려면 [Aurora 글로벌 데이터베이스에 AWS 리전 추가 \(p. 46\)](#) 단원의 절차를 따르십시오. 각 보조 클러스터는 원래 클러스터와 다른 AWS 리전에 있습니다.

AWS Management 콘솔(버전 5.6.10a만 해당)을 사용하여 Aurora MySQL 글로벌 데이터베이스를 생성하려면

Aurora MySQL 버전 5.6.10a를 사용하려면 클러스터 생성 시 클러스터를 Aurora Global Database와 호환되도록 선택해야 합니다. 이 Aurora MySQL 버전으로 클러스터를 생성할 때에만 AWS Management 콘솔에서 글로벌 선택 항목을 볼 수 있습니다. Aurora MySQL 버전 1.22 이상에서는 클러스터를 생성할 때 특별한 선택 없이 Aurora Global Database가 있는 모든 클러스터를 사용할 수 있습니다.

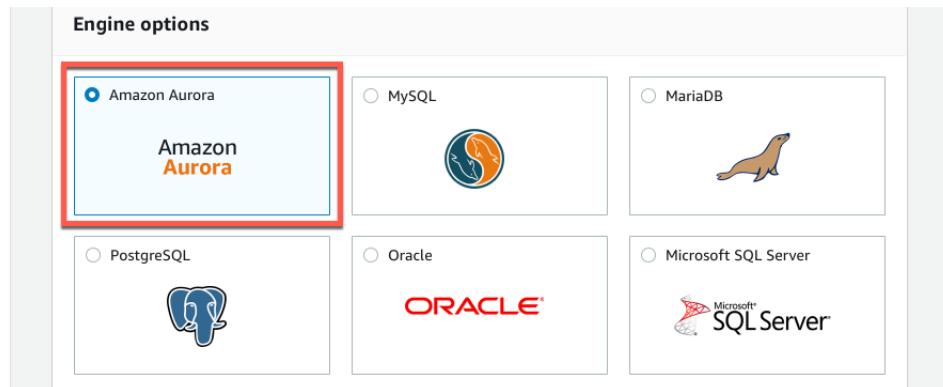
5.6.10a에서 1.22까지의 Aurora MySQL 버전에서는 클러스터 생성 시 특별한 선택을 하지 않습니다. 그러나 Aurora Global Database에서 해당 클러스터를 사용하려면 먼저 Aurora MySQL 버전 1.22 이상으로 업그레이드해야 합니다.

1. AWS Management 콘솔에 로그인하고 Aurora 글로벌 데이터베이스를 사용할 수 있는 리전에 대한 Amazon RDS 콘솔을 엽니다. [Aurora Global Database에 적용되는 제한 사항 \(p. 38\)](#)도 참조하십시오.
2. 데이터베이스 생성을 선택합니다.
3. 엔진 선택 페이지에서 Aurora 엔진을 선택하고 Database location(데이터베이스 위치)로 글로벌을 선택합니다. 예시는 다음 이미지를 참조하십시오.

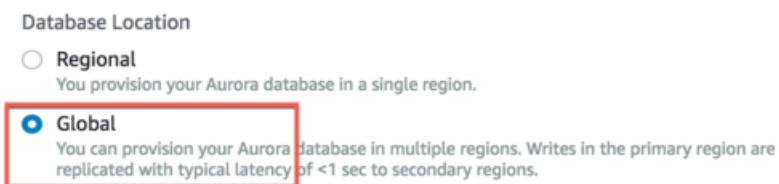
Note

Standard Create(표준 생성)가 선택되어 있는지 확인합니다. Standard create(표준 생성)를 고면 Aurora 글로벌 데이터베이스에 필요한 선택 사항이 표시됩니다. Easy Create(간편 생성)를 선택하지 마십시오.

- a. Amazon Aurora를 엔진으로 선택합니다.



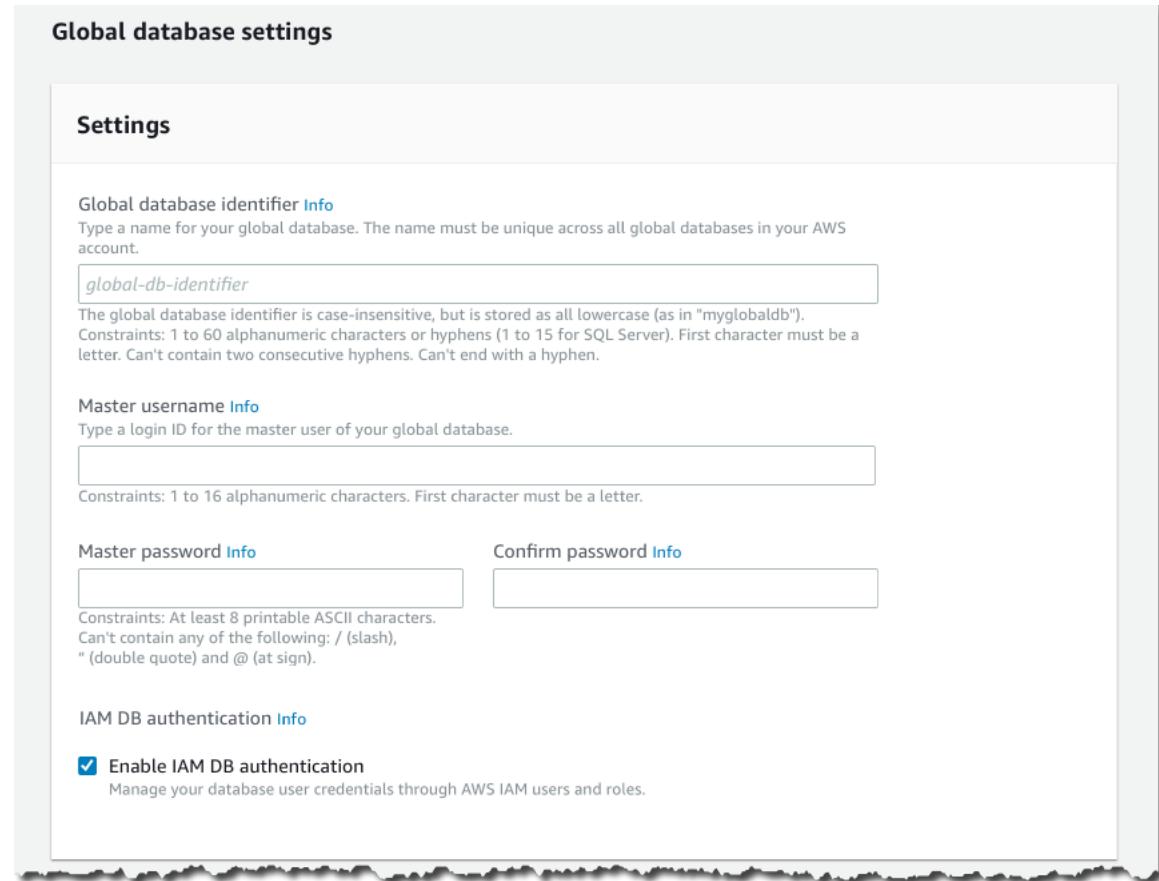
- b. MySQL과 호환되는 Amazon Aurora 에디션을 선택합니다.
- c. 버전 5.6.10a를 선택합니다.
- d. 데이터베이스 위치로 글로벌을 선택합니다.



Note

글로벌을 선택하면 글로벌 데이터베이스와 기본 Aurora 클러스터가 설정됩니다. 글로벌 데이터베이스가 생성되어 사용 가능한 상태가 되면 보조 AWS 리전을 추가할 수 있습니다.

4. 다른 Aurora 클러스터와 동일한 결정 프로세스에 따라 나머지 설정을 채웁니다. Aurora DB 클러스터 생성에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.



5. Create를 선택합니다.

기본 DB 클러스터가 생성되고 사용할 수 있게 되면 [Aurora 글로벌 데이터베이스에 AWS 리전 추가 \(p. 46\)](#)의 단계에 따라 보조 클러스터를 하나 이상 생성합니다.

AWS CLI

CLI를 사용하여 Aurora Global Database를 생성하려면 다음 절차 중 하나를 사용합니다.

- Aurora 글로벌 데이터베이스를 생성한 후 기본 AWS 리전 및 DB 인스턴스를 해당 AWS 리전에 추가합니다.
- 또는 기본 클러스터를 먼저 생성한 후, 글로벌 데이터베이스를 생성하여 기본 클러스터로 사용할 클러스터를 지정할 수도 있습니다.

Aurora Global Database를 생성한 다음 기본 AWS 리전 및 DB 인스턴스를 추가하려면

1. Aurora Global Database를 생성한 다음 기본 AWS 리전 클러스터를 추가합니다.

- a. Aurora 글로벌 데이터베이스 자체를 생성합니다. 처음에는 글로벌 데이터베이스에 아무런 클러스터도 연결되어 있지 않습니다. 다음 값을 사용합니다.
 - Aurora MySQL의 경우 --engine 파라미터에 대해 aurora를 지정합니다.
 - Aurora PostgreSQL의 경우 --engine 파라미터에 대해 aurora-postgresql을, --engine-version 파라미터에 대해 10.11을 지정합니다.

Linux, OS X, Unix의 경우:

```
aws rds create-global-cluster --region primary_region \
--global-cluster-identifier global_database_id \
--engine engine_type \
--engine-version version # optional
```

Windows의 경우:

```
aws rds create-global-cluster ^
--global-cluster-identifier global_database_id ^
--engine engine_type ^
--engine-version version # optional
```

추가 옵션에 대해 자세히 알아보려면 [create-global-cluster](#) 단원을 참조하십시오.

- b. 다음 예제와 같이 AWS CLI [describe-global-clusters](#) 명령을 사용하여 Aurora 글로벌 데이터베이스를 사용할 수 있는지 확인합니다.

```
aws rds describe-global-clusters --region primary_region --global-cluster-identifier global_database_id # ID is optional
```

결과에 상태가 available로 표시되면 다음 단계로 넘어갑니다.

- c. Aurora Global Database에 대한 기본 클러스터를 생성합니다. 이렇게 하려면 AWS CLI [create-db-cluster](#) 명령을 사용합니다. 글로벌 데이터베이스를 생성했을 때와 동일한 --global-cluster-identifier 값을 지정합니다.
 - Aurora MySQL의 경우 --engine 파라미터에 대해 aurora를 지정합니다.
 - Aurora PostgreSQL의 경우 --engine 파라미터에 대해 aurora-postgresql을, --engine-version 파라미터에 대해 10.11을 지정합니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster \
--region primary_region \
--db-cluster-identifier db_cluster_id \
--master-username master_userid \
--master-user-password master_password \
--engine { aurora | aurora-mysql | aurora-postgresql } \
--engine-version version \
--global-cluster-identifier global_database_id
```

Windows의 경우:

```
aws rds create-db-cluster ^
--region primary_region ^
--db-cluster-identifier db_cluster_id ^
--master-username master_userid ^
```

```
--master-user-password master_password ^
--engine { aurora | aurora-mysql | aurora-postgresql } ^
--engine-version version ^
--global-cluster-identifier global_database_id
```

Note

Aurora MySQL 버전 5.6.10a의 경우 --engine-mode 파라미터에 대해 global도 지정해야 합니다.

추가 옵션에 대해 자세히 알아보려면 [create-db-cluster](#) 단원을 참조하십시오.

- d. 다음 예와 같이 AWS CLI [describe-db-clusters](#) 명령을 사용하여 Aurora 클러스터를 사용할 수 있는지 확인합니다.

```
aws rds describe-db-clusters --region primary_region --db-cluster-
identifier db_cluster_id # ID is optional
```

결과에 상태가 available로 표시되면 다음 단계로 넘어갑니다.

2. 기본 클러스터에 대한 기본 DB 인스턴스를 생성합니다. 이렇게 하려면 AWS CLI [create-db-instance](#) 명령을 사용합니다.

- 다음과 같이 --engine 파라미터를 지정합니다.
 - Aurora MySQL 버전 1 또는 5.6.10a의 경우 --engine aurora를 사용합니다.
 - Aurora MySQL 버전 2의 경우 --engine aurora-mysql을 사용합니다.
 - Aurora PostgreSQL의 경우 --engine aurora-postgresql을 사용합니다.
 - 선택적으로 --engine-version 파라미터를 지정하여 사용할 특정 버전을 선택합니다.
 - Aurora MySQL 버전 5.6.10a의 경우 --engine-mode global을 지정합니다.

다음 예에서는 이 작업을 수행하는 방법을 보여줍니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-instance \
--db-cluster-identifier db_cluster_id \
--db-instance-class instance_class \
--db-instance-identifier db_instance_id \
--engine { aurora | aurora-mysql | aurora-postgresql } \
--engine-version version \
--region primary_region
```

Windows의 경우:

```
aws rds create-db-instance ^
--db-cluster-identifier db_cluster_id ^
--db-instance-class instance_class ^
--db-instance-identifier db_instance_id ^
--engine { aurora | aurora-mysql | aurora-postgresql } ^
--engine-version version ^
--region primary_region
```

--master-username 및 --master-user-password 파라미터 값은 기본 DB 클러스터에서 가져오므로 포함시키지 않아도 됩니다.

추가 옵션에 대해 자세히 알아보려면 [create-db-instance](#) 단원을 참조하십시오.

3. 다음 예제와 같이 AWS CLI [describe-db-clusters](#) 명령을 사용하여 Aurora 글로벌 데이터베이스의 기본 클러스터를 사용할 수 있는지 확인합니다.

```
aws rds describe-db-clusters --db-cluster-identifier sample-global-cluster
```

4. [describe-db-clusters](#) 결과에 상태가 `available`로 표시되면 기본 클러스터의 기본 인스턴스를 만들어 복제를 시작합니다. 이렇게 하려면 다음 예제에서와 같이 AWS CLI [create-db-instance](#) 명령을 사용하십시오.

Linux, OS X, Unix의 경우:

```
aws rds create-db-instance \
--db-cluster-identifier sample-global-cluster \
--db-instance-class instance_class \
--db-instance-identifier sample-replica-instance \
--engine aurora
```

Windows의 경우:

```
aws rds create-db-instance ^
--db-cluster-identifier sample-global-cluster ^
--db-instance-class instance_class ^
--db-instance-identifier sample-replica-instance ^
--engine aurora
```

DB 인스턴스가 생성되어 사용할 수 있게 되면 복제가 시작됩니다. AWS CLI [describe-db-instances](#) 명령을 호출하여 DB 인스턴스를 사용할 수 있는지 여부를 파악할 수 있습니다.

대체 절차로서, 먼저 기본 DB 클러스터를 생성한 다음 글로벌 데이터베이스를 생성할 수도 있습니다.

1. 빈 Aurora DB 클러스터를 생성합니다.

- Aurora MySQL 버전 5.6.10a의 경우 다음 파라미터와 값을 지정합니다.
 - `--engine aurora`
 - `--engine-version 5.6.10a`
 - `--engine-mode global`

이 설정은 `aurora --engine` 설정과 함께 사용해야 합니다.

- Aurora PostgreSQL의 경우 다음 파라미터와 값을 지정합니다.
 - `--engine aurora-postgresql`
 - `--engine-version 10.11`

Linux, OS X, Unix의 경우:

```
aws rds --region primary_region \
create-db-cluster \
--db-cluster-identifier primary_cluster_id \
--master-username master_userid \
--master-user-password master_password \
--engine { aurora | aurora-mysql | aurora-postgresql } \
--engine-version version

aws rds --region primary_region \
create-db-instance \
```

```
--db-instance-class instance_class \
--db-cluster-identifier primary_cluster_id \
--db-instance-identifier db_instance_id \
--engine { aurora | aurora-mysql | aurora-postgresql }
```

Windows의 경우:

```
aws rds --region primary_region ^
create-db-cluster ^
--db-cluster-identifier primary_cluster_id ^
--master-username master_userid ^
--master-user-password master_password ^
--engine { aurora | aurora-mysql | aurora-postgresql } ^
--engine-version version

aws rds --region primary_region ^
create-db-instance ^
--db-instance-class instance_class ^
--db-cluster-identifier primary_cluster_id ^
--db-instance-identifier db_instance_id ^
--engine { aurora | aurora-mysql | aurora-postgresql }
```

- 값을 사용하여 Aurora DB 클러스터 내에 DB 인스턴스를 생성합니다.

- Aurora MySQL 버전 5.6.10a의 경우 --engine aurora 및 --engine-mode global을 지정합니다.
- Aurora PostgreSQL의 경우 --engine aurora-postgresql을 지정합니다.

다음은 Aurora PostgreSQL 예제입니다.

Linux, OS X, Unix의 경우:

```
aws rds --region primary_region \
create-db-instance \
--db-instance-class instance_class \
--db-cluster-identifier primary_cluster_id \
--db-instance-identifier db_instance_id \
--engine aurora-postgresql
```

Windows의 경우:

```
aws rds --region primary_region ^
create-db-instance ^
--db-instance-class instance_class ^
--db-cluster-identifier primary_cluster_id ^
--db-instance-identifier db_instance_id ^
--engine aurora-postgresql
```

- 기본 DB 클러스터와 DB 인스턴스를 생성했고 사용할 수 있으면, Aurora Global Database를 생성할 AWS 리전에서 AWS CLI [create-global-cluster](#) 명령을 호출하여 Aurora Global Database를 생성합니다. --source-db-cluster-identifier 파라미터를 사용하여 이전에 생성한 기본 DB 클러스터를 지정합니다.

Linux, OS X, Unix의 경우:

```
aws rds create-global-cluster \
--global-cluster-identifier global_database_id \
--source-db-cluster-identifier primary_cluster_ARN
```

Windows의 경우:

```
aws rds create-global-cluster ^
--global-cluster-identifier global_database_id ^
--source-db-cluster-identifier primary_cluster_ARN
```

RDS API

RDS API를 사용하여 Aurora 글로벌 데이터베이스를 생성하려면 [CreateGlobalCluster](#) 작업을 실행합니다.

Aurora 글로벌 데이터베이스에 AWS 리전 추가

Aurora 글로벌 데이터베이스와 관련 기본 클러스터를 생성한 후 새 AWS 리전을 추가하여 글로벌 범위를 확대할 수 있습니다. 이 프로세스에는 하나 이상의 보조 Aurora 클러스터에 연결하는 작업이 수반됩니다. 각 보조 클러스터는 기본 클러스터 및 기타 보조 클러스터와 다른 AWS 리전에 있어야 합니다.

Note

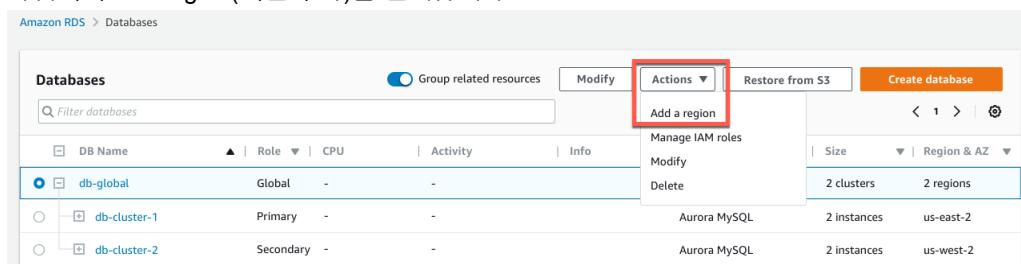
Aurora Global Database에 연결할 수 있는 보조 Aurora 클러스터의 최대 개수는 5개입니다. 따라서 글로벌 데이터베이스와 연결할 수 있는 대부분의 클러스터는 총 6개로 읽기-쓰기 인스턴스 및 선택적 읽기 인스턴스가 있는 기본 클러스터 하나와 읽기 전용 보조 클러스터 5개입니다.

글로벌 데이터베이스의 각 보조 클러스터에 대해 기본 클러스터의 최대 읽기 DB 인스턴스 수가 1씩 줄어듭니다. 예를 들어, 글로벌 데이터베이스에 보조 클러스터가 세 개 있는 경우 기본 클러스터의 최대 읽기 DB 인스턴스 수는 15개가 아닌 12개입니다. 기본 클러스터의 읽기 인스턴스 수와 보조 클러스터 수를 합한 수가 15인 경우 글로벌 데이터베이스에 보조 클러스터를 더 이상 추가할 수 없습니다.

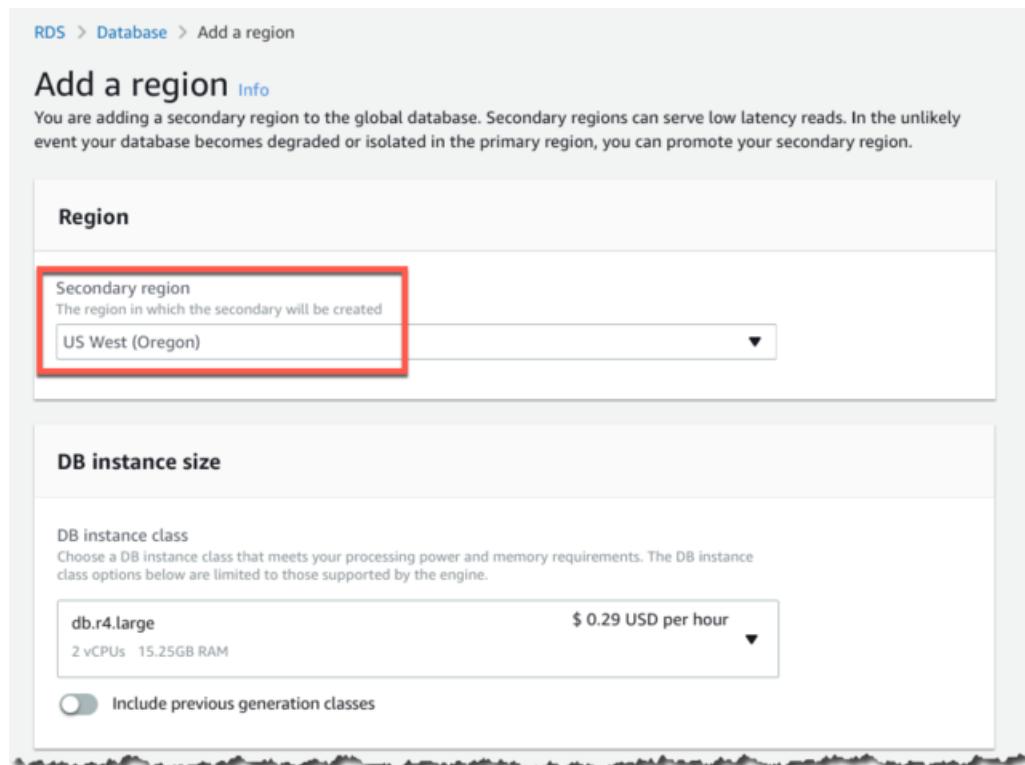
콘솔

AWS Management 콘솔을 사용하여 Aurora 글로벌 데이터베이스에 AWS 리전을 추가하려면

1. AWS Management 콘솔의 탐색 창에서 데이터베이스를 선택합니다.
2. 보조 클러스터를 생성할 Aurora 글로벌 데이터베이스의 확인란을 선택합니다. 기본 클러스터나 기본 클러스터 내부의 DB 인스턴스가 여전히 *Creating* 상태인 경우, 이러한 클러스터나 DB 인스턴스가 모두 *Available* 상태가 될 때까지 기다립니다.
3. 작업에서 *Add region(리전 추가)*을 선택합니다.



4. 리전 추가 페이지에서 보조 AWS 리전을 선택합니다.



5. 새 AWS 리전에서 보조 Aurora 클러스터의 나머지 필드를 작성한 후 리전 추가를 선택합니다.

AWS CLI

AWS CLI를 사용하여 Aurora 글로벌 데이터베이스에 AWS 리전을 추가하려면 [create-db-cluster 명령](#)을 실행합니다.

다음 명령은 새 Aurora 클러스터를 생성하고, 글로벌 데이터베이스에 읽기 전용 보조 클러스터로 연결한 후, 새 클러스터에 DB 인스턴스를 추가합니다. `create-db-cluster` 명령에 대한 `--global-cluster-identifier` 옵션은 새 클러스터를 연결할 글로벌 데이터베이스를 지정합니다. 다음과 같이 `--engine` 파라미터를 지정합니다.

- Aurora MySQL 버전 1 또는 5.6.10의 경우 `--engine aurora`를 사용합니다.
- Aurora MySQL 버전 2의 경우 `--engine aurora-mysql`을 사용합니다.
- Aurora PostgreSQL의 경우 `--engine aurora-postgresql`을 사용합니다.

암호화된 클러스터의 경우, 기본 AWS 리전과 일치하는 값을 사용하여 `--source-region` 옵션을 지정합니다.

다음은 Aurora PostgreSQL 예제입니다.

Linux, OS X, Unix의 경우:

```
aws rds --region secondary_region \
  create-db-cluster \
    --db-cluster-identifier secondary_cluster_id \
    --global-cluster-identifier global_database_id \
    --engine { aurora | 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 | 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 | 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 | aurora-mysql | aurora-postgresql }
```

RDS API

RDS API를 사용하여 Aurora 글로벌 데이터베이스에 새 AWS 리전을 추가하려면 [CreateDBCluster](#) 작업을 실행합니다. GlobalClusterIdentifier 파라미터를 사용하여 기존 글로벌 데이터베이스의 식별자를 지정합니다.

Aurora 글로벌 데이터베이스에서 클러스터 제거

Aurora 글로벌 데이터베이스에서 Aurora 클러스터를 제거하면 해당 클러스터가 리전 클러스터로 다시 전환되어, 모든 읽기-쓰기 기능이 부여되고 더 이상 기본 클러스터와 동기화되지 않습니다.

기본 클러스터가 성능 저하되거나 분리된 경우 글로벌 데이터베이스에서 Aurora 클러스터를 제거할 수 있습니다. Aurora 글로벌 데이터베이스에 대한 장애 조치를 수행하는 과정에는 원본 글로벌 데이터베이스에서 보조 클러스터 중 하나를 제거한 후 새 Aurora 글로벌 데이터베이스에서 기본 클러스터로 사용하는 작업이 수반됩니다. 자세한 내용은 [Aurora 글로벌 데이터베이스에 대한 장애 조치 \(p. 53\)](#) 단원을 참조하십시오.

글로벌 데이터베이스가 더 이상 필요하지 않다면 먼저 보조 클러스터를 모두 제거하고 나서 기본 클러스터를 제거한 후 글로벌 데이터베이스 자체를 삭제해야 합니다. 그런 다음 제거한 클러스터 중 일부 또는 모두를 삭제하거나 이들 클러스터 중 일부 또는 모두를 단일 리전 Aurora 클러스터로 계속 사용할 수 있습니다. 자세한 내용은 [Aurora 글로벌 데이터베이스 삭제 \(p. 50\)](#) 단원을 참조하십시오.

콘솔

AWS Management 콘솔로 Aurora Global Database에서 Aurora 클러스터를 제거하려면

- 데이터베이스 페이지에서 클러스터를 선택합니다.
- 작업에 대해 Remove from Global(글로벌에서 제거)을 선택합니다. 제거된 클러스터는 모든 읽기-쓰기 기능이 부여된 일반 Aurora 클러스터가 됩니다. 기본 클러스터와의 동기화 상태가 더 이상 유지되지 않습니다.



보조 클러스터가 여전히 글로벌 데이터베이스와 연결되어 있는 경우 기본 클러스터를 제거할 수 없습니다.

보조 클러스터를 모두 제거하거나 삭제한 후 동일한 방식으로 기본 클러스터를 제거할 수 있습니다.

클러스터를 Aurora 글로벌 데이터베이스에서 제거한 후에도 이들 클러스터는 글로벌, 기본 및 보조 레이블은 없지만 데이터베이스 페이지에 여전히 표시됩니다.

DB Name	Role	CPU	Activity	Info	Engine	Size	Region & A
db-global	Global	-	-		Aurora MySQL	0 clusters	0 regions
db-cluster-1	Global compatible	-	-		Aurora MySQL	2 instances	us-east-2
instance-name	Writer	6.72%	1 Selects/Sec		Aurora MySQL	db.r4.xlarge	us-east-2a
instance-name-us-west-2b	Reader	6.72%	1 Selects/Sec		Aurora MySQL	db.r4.xlarge	us-east-2b
dbname-cluster2	Cluster	-	-		Aurora MySQL	2 instances	us-east-2
instance-name1-1	Writer	6.72%	1 Selects/Sec		Aurora MySQL	db.r4.xlarge	us-east-2a
instance-name1-2-us-west-2b	Reader	6.72%	1 Selects/Sec		Aurora MySQL	db.r4.xlarge	us-east-2b
instance-master-name4	Instance	6.72%	1 Selects/Sec		PostgreSQL	db.r4.xlarge	us-east-2
instance-name4	Replica	6.72%	1 Selects/Sec		PostgreSQL	db.r4.xlarge	us-east-2a
dbname3	Instance	6.72%	1 Selects/Sec		MySQL	db.r4.xlarge	us-east-2b
dbname4	Instance	6.72%	1 Selects/Sec		Oracle Enterprise Edition	db.r4.xlarge	us-east-2

AWS CLI

AWS CLI를 사용하여 Aurora 글로벌 데이터베이스에서 Aurora 클러스터를 제거하려면 [remove-from-global-cluster](#) 명령을 실행합니다.

다음 명령은 Aurora 글로벌 데이터베이스에서 보조 클러스터를 제거한 후 기본 클러스터를 제거합니다. 제거 할 클러스터는 각 경우에서 --db-cluster-identifier 파라미터로 식별됩니다. Aurora 글로벌 데이터베이스는 --global-cluster-identifier 파라미터로 식별됩니다.

Linux, OS X, Unix의 경우:

```
aws rds --region secondary_region \
    remove-from-global-cluster \
    --db-cluster-identifier secondary_cluster_ARN \
    --global-cluster-identifier global_database_id
... repeat above command with a different --region for all secondary clusters ...

aws rds --region primary_region \
    remove-from-global-cluster \
    --db-cluster-identifier primary_cluster_ARN \
    --global-cluster-identifier global_database_id
```

Windows의 경우:

```
aws rds --region secondary_region ^
    remove-from-global-cluster ^
    --db-cluster-identifier secondary_cluster_ARN ^
    --global-cluster-identifier global_database_id
... repeat above command with a different --region for all secondary clusters ...

aws rds --region primary_region ^
```

```
remove-from-global-cluster ^
--db-cluster-identifier primary_cluster_ARN ^
--global-cluster-identifier global_database_id
```

RDS API

RDS API를 사용하여 Aurora 글로벌 데이터베이스에서 Aurora 클러스터를 제거하려면 [RemoveFromGlobalCluster](#) 작업을 실행합니다.

Aurora 글로벌 데이터베이스 삭제

Aurora Global Database를 삭제하기 전에 먼저 글로벌 데이터베이스의 클러스터를 제거하거나 삭제해야 합니다. 글로벌 데이터베이스는 일반적으로 비즈니스에 중요한 데이터를 포함하므로, 글로벌 데이터베이스와 이 데이터베이스에 연결된 클러스터를 한 번에 삭제할 수는 없습니다. 글로벌 데이터베이스에서 클러스터를 제거하는 방법에 대한 자세한 내용은 [Aurora 글로벌 데이터베이스에서 클러스터 제거 \(p. 48\)](#) 단원을 참조하십시오. 이렇게 하면 다시 독립 실행형 Aurora 클러스터가 됩니다. 제거한 후 클러스터를 삭제하는 절차는 [AuroraDB 클러스터에서 DB 인스턴스 삭제 \(p. 313\)](#) 단원을 참조하십시오.

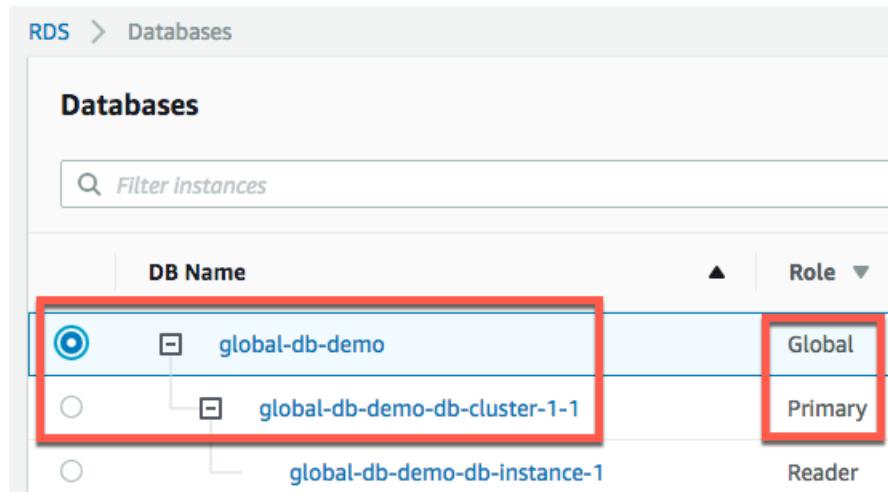
콘솔

AWS Management 콘솔을 사용하여 Aurora 글로벌 데이터베이스를 삭제하려면

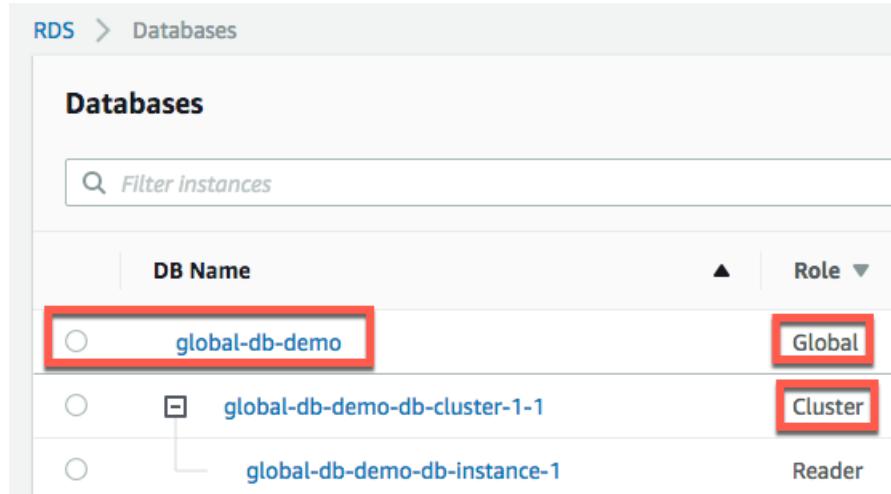
AWS Management 콘솔을 사용하여 Aurora Global Database를 삭제하려면 먼저 글로벌 데이터베이스와 연결된 모든 보조 클러스터를 제거하거나 삭제하고 기본 클러스터를 제거한 후 글로벌 데이터베이스 자체를 삭제합니다. Aurora 클러스터에서 쓰기 인스턴스를 삭제하면 클러스터 자체가 삭제됩니다. 글로벌 데이터베이스는 일반적으로 비즈니스에 중요한 데이터를 포함하므로, 글로벌 데이터베이스와 이 데이터베이스에 연결된 클러스터를 한 번에 삭제할 수는 없습니다. 글로벌 데이터베이스에서 클러스터 제거에 대한 지침은 [Aurora 글로벌 데이터베이스에서 클러스터 제거 \(p. 48\)](#) 단원을 참조하십시오. 제거된 클러스터를 삭제하는 절차는 [AuroraDB 클러스터에서 DB 인스턴스 삭제 \(p. 313\)](#) 단원을 참조하십시오. Aurora 클러스터에서 기본 인스턴스를 삭제하면 클러스터 자체가 삭제됩니다.

1. Aurora 글로벌 데이터베이스에서 다른 모든 클러스터가 제거되었는지 확인합니다.

글로벌 데이터베이스에 중첩된 클러스터가 포함된 경우 다음 스크린샷에서처럼 글로벌 데이터베이스를 아직 삭제할 수 없습니다. 글로벌 데이터베이스와 연결된 각 클러스터에 대해 [Aurora 글로벌 데이터베이스에서 클러스터 제거 \(p. 48\)](#)의 절차를 따릅니다.



글로벌 데이터베이스에 연결된 클러스터가 없어야 삭제를 진행할 수 있습니다. 다음 스크린샷은 `global-db-demo` 클러스터가 제거된 후 글로벌 데이터베이스와 더 이상 연결되지 않았음을 보여줍니다.



- 데이터베이스 페이지 또는 글로벌 데이터베이스의 세부 정보 페이지에서 작업을 선택한 후 삭제를 선택합니다.

AWS CLI

AWS CLI를 사용하여 Aurora 글로벌 데이터베이스의 일부로 포함된 클러스터를 삭제하려면 [delete-global-cluster](#) 명령을 실행합니다.

다음 명령은 지정된 글로벌 클러스터 ID의 Aurora 글로벌 데이터베이스를 삭제합니다.

Linux, OS X, Unix의 경우:

```
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 Global Database의 일부로 포함된 클러스터를 삭제하려면 [DeleteGlobalCluster](#) 작업을 실행합니다.

Aurora 글로벌 데이터베이스로 데이터 가져오기

데이터를 Aurora 글로벌 데이터베이스로 가져오려면 다음 기법 중 하나를 사용하십시오.

- Aurora 클러스터나 Amazon RDS DB 인스턴스의 스냅샷을 생성하고 Aurora Global Database의 기본 클러스터로 복원합니다. 현재는 스냅샷을 Aurora MySQL 버전 1, Aurora MySQL 버전 2 또는 Aurora PostgreSQL 10.11과 호환되는 소스에서 생성해야 합니다.

다음은 Aurora MySQL 5.6 인스턴스를 복원하는 예입니다.

RDS > Snapshots > Restore Snapshot

Restore DB Instance

You are creating a new DB Instance from a source DB Instance at a specified time. This new DB Instance will have the default DB Security Group and DB Parameter Groups.

Instance specifications

DB Engine
Name of the Database Engine

Aurora MySQL

DB Engine Version
Version Number of the Database Engine to be used for this instance

Aurora (MySQL)-5.6.10a (default)

Capacity type [Info](#)

- Provisioned**
You provision and manage the server instance sizes.
- Provisioned with Aurora parallel query enabled [Info](#)**
You provision and manage the server instance sizes, and Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6)
- Serverless [Info](#)**
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load (currently available for Aurora MySQL 5.6)
- 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.

- 기본 클러스터의 특정 시점 복원을 사용하여 클러스터 데이터를 이전 상태로 복원합니다. 자세한 내용은 [DB 클러스터를 지정된 시간으로 복원 \(p. 303\)](#) 단원을 참조하십시오.

Aurora 글로벌 데이터베이스의 중요한 특징은 항상 기본 클러스터로 지정하는 클러스터로 데이터를 가져온다는 점입니다. 해당 클러스터를 Aurora 글로벌 데이터베이스에 추가하기 이전이나 이후에 초기 데이터 가져오기를 수행할 수 있습니다. 기본 클러스터의 모든 데이터는 보조 클러스터에 자동으로 복제됩니다.

Aurora 글로벌 데이터베이스에 연결

읽기 전용 쿼리 트래픽의 경우 자신의 AWS 리전에 있는 Aurora 클러스터용 리더 엔드포인트에 연결합니다.

데이터 조작 언어(DML) 또는 데이터 정의 언어(DDL) 설명문을 실행하려면 기본 클러스터용 클러스터 엔드포인트에 연결합니다. 이 엔드포인트는 해당 애플리케이션과는 다른 AWS 리전에 있을 수도 있습니다.

다음 스크린샷에서처럼 AWS Management 콘솔에서 Aurora 글로벌 데이터베이스를 볼 때 모든 관련 클러스터와 연결된 범용 엔드포인트를 모두 볼 수 있습니다. 쓰기 작업용으로 사용하며 기본 클러스터와 연결된 단일 클러스터 엔드포인트가 있습니다. 기본 클러스터와 각 보조 클러스터에는 읽기 전용 쿼리에 사용하는 리더 엔드포인트가 있습니다. 지연 시간을 최소화하려면 자신의 AWS 리전이나 가장 가까운 AWS 리전의 리더 엔드포인트를 선택합니다. 다음은 Aurora MySQL 예제입니다.

Endpoint name	Status	Status	Port
gdemo-04-db-cluster-1-1.cluster-.us-east-2.rds.amazonaws.com	Available	Writer	3306
gdemo-04-db-cluster-1-1.cluster-ro-.us-east-2.rds.amazonaws.com	Available	Reader	3306

Aurora 글로벌 데이터베이스에 대한 장애 조치

Aurora 글로벌 데이터베이스는 기본 Aurora 클러스터보다 더 높은 수준의 장애 조치를 제공합니다. AWS 리전 하나에서 전체 클러스터를 사용할 수 없게 되면, 글로벌 데이터베이스의 다른 클러스터를 승격하여 읽기-쓰기 기능을 갖추도록 할 수 있습니다.

다른 AWS 리전의 클러스터가 기본 클러스터로 더 적합한 경우 장애 조치 메커니즘을 수동으로 활성화할 수 있습니다. 예를 들어 보조 클러스터 중 하나의 용량을 늘린 후 이 클러스터를 기본 클러스터로 승격할 수 있습니다. 또는 AWS 리전 간의 활동 균형이 변경될 수 있으므로, 기본 클러스터를 다른 AWS 리전으로 전환하면 쓰기 작업에 대한 지연 시간이 낮아질 수도 있습니다.

다음 단계는 Aurora 글로벌 데이터베이스에서 보조 클러스터 중 하나를 승격할 때 발생하는 이벤트의 시퀀스를 개괄적으로 설명한 것입니다.

1. 기본 클러스터가 사용할 수 없는 상태가 됩니다.
2. Aurora 글로벌 데이터베이스에서 보조 클러스터를 제거합니다. 이렇게 하면 이 클러스터가 승격되어 모든 읽기-쓰기 기능이 부여됩니다. 글로벌 데이터베이스에서 Aurora 클러스터를 제거하는 방법은 [Aurora 글로벌 데이터베이스에서 클러스터 제거 \(p. 48\)](#) 단원을 참조하십시오.
3. 쓰기 트래픽이 새로 승격된 클러스터로 향하도록 애플리케이션을 다시 구성합니다.

Important

이때 사용할 수 없게 된 클러스터에 대한 DML 문이나 다른 쓰기 작업의 실행을 중지합니다. 클러스터 간 데이터 불일치("브레인 분할" 시나리오)를 방지하려면 다시 온라인으로 전환되더라도 이전 기본 클러스터에 쓰지 마십시오.

4. 새로 승격된 클러스터를 기본 클러스터로 사용하여 새 Aurora 글로벌 데이터베이스를 생성합니다. Aurora 글로벌 데이터베이스 생성 방법은 [Aurora 글로벌 데이터베이스 생성 \(p. 39\)](#) 단원을 참조하십시오.
5. 문제가 발생한 클러스터의 AWS 리전을 Aurora 글로벌 데이터베이스에 추가합니다. 이제 AWS 리전에 새 보조 클러스터가 포함됩니다. Aurora 글로벌 데이터베이스에 AWS 리전을 추가하는 방법은 [Aurora 글로벌 데이터베이스에 AWS 리전 추가 \(p. 46\)](#) 단원을 참조하십시오.
6. 중단 문제를 해결했고 원본 AWS 리전을 기본 클러스터로 다시 할당할 준비가 되었으면, 동일한 단계를 역순으로 수행합니다.
 - a. Aurora 글로벌 데이터베이스에서 보조 클러스터 중 하나를 제거합니다.
 - b. 해당 클러스터를 원본 AWS 리전에 있는 새 Aurora 글로벌 데이터베이스의 기본 클러스터로 설정합니다.
 - c. 모든 쓰기 트래픽이 원본 AWS 리전의 기본 클러스터로 리디렉션됩니다.
 - d. AWS 리전을 추가하여 전과 동일한 AWS 리전에서 보조 클러스터를 하나 이상 설정합니다.

Aurora Global Database를 위한 성능 개선 도우미

Amazon RDS 성능 개선 도우미와 Aurora Global Database를 함께 사용할 수 있습니다. 이렇게 할 경우 성능 개선 도우미 보고서는 글로벌 데이터베이스에 있는 각 클러스터에 개별적으로 적용됩니다. 글로벌 데이터베

이스의 일부인 각 클러스터에 대해 성능 개선 도우미를 활성화하거나 비활성화할 수 있습니다. 이미 성능 개선 도우미를 사용하고 있는 글로벌 데이터베이스에 새로운 보조 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 리전을 전환할 수 있습니다. 그러나 AWS 리전을 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 리전을 전환한 후에 성능 개선 도우미 탐색 창에서 DB 인스턴스의 이름을 다시 선택하십시오.

글로벌 데이터베이스와 연결된 DB 인스턴스의 경우, 성능에 영향을 미치는 요인은 AWS 리전에 따라 다를 수 있습니다. 예를 들어 각 리전의 DB 인스턴스는 용량이 서로 다를 수 있습니다.

성능 개선 도우미 사용에 대한 자세한 내용은 [Amazon RDS 성능 개선 도우미 사용 \(p. 365\)](#) 단원을 참조하십시오.

Aurora 글로벌 데이터베이스에서 여러 개의 보조 리전 사용

Aurora 글로벌 데이터베이스에 보조 리전이 하나 이상 포함되어 있으면 기본 AWS 리전에 영향을 미치는 종단이 발생한 경우 장애 조치할 AWS 리전을 선택할 수 있습니다. 어떤 보조 리전을 선택할지 결정하는데 도움이 되는 방법으로 모든 보조 리전의 복제 지연 시간을 모니터링할 수 있습니다. 한 곳의 보조 리전이 다른 보조 리전에 비해 상당히 짧은 복제 지연 시간을 나타내는 경우 이는 연결된 Aurora 클러스터에 애플리케이션에 대한 전체 읽기-쓰기 워크로드를 인수할 데이터베이스 및 네트워크 용량이 충분히 있다는 좋은 의미로 해석할 수 있습니다.

글로벌 데이터베이스에 여러 개의 보조 리전이 있다면 기본 리전이 중단된 경우 모든 보조 리전의 연결을 해제해야 합니다. 그런 다음 이 보조 리전 중 하나를 새 기본 리전으로 승격하십시오. 끝으로 다른 보조 리전 각각에 새 클러스터를 생성하고 이 클러스터를 글로벌 데이터베이스에 연결합니다.

다른 AWS 서비스와 함께 Aurora 글로벌 데이터베이스 사용

경우에 따라 Aurora 글로벌 데이터베이스와 결합하여 다른 AWS 서비스에 액세스해야 할 수도 있습니다. 이러한 경우에는 연결된 모든 클러스터에 대해 해당 AWS 리전에 동일한 권한, 외부 함수 등이 필요합니다. 글로벌 데이터베이스의 Aurora 클러스터를 읽기 전용 복제 대상으로 시작할 수 있더라도, 나중에 기본 클러스터로 승격할 수 있습니다. 이러한 가능성을 준비하려면 글로벌 데이터베이스의 모든 Aurora 클러스터에 대해 다른 서비스에 필요한 쓰기 권한을 미리 설정합니다.

다음 목록에는 각 AWS 서비스에 대해 수행할 작업이 요약되어 있습니다.

Lambda 함수 호출

Aurora 글로벌 데이터베이스를 구성하는 모든 Aurora 클러스터에 대해 [Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출 \(p. 633\)](#)의 절차를 수행합니다.

Aurora 글로벌 데이터베이스의 각 클러스터에 대해 `aws_default_lambda_role` 클러스터 파라미터를 새 IAM 역할의 Amazon 리소스 이름(ARN)으로 설정합니다.

Aurora 글로벌 데이터베이스의 데이터베이스 사용자가 Lambda 함수를 호출하도록 허용하려면 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)에서 생성한 역할을 Aurora 글로벌 데이터베이스의 각 클러스터와 연결합니다.

Lambda로의 발신 연결을 허용하도록 Aurora 글로벌 데이터베이스의 각 클러스터를 구성합니다. 지침은 [Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 \(p. 619\)](#) 단원을 참조하십시오.

S3에서 데이터 로드

Aurora 글로벌 데이터베이스를 구성하는 모든 Aurora 클러스터에 대해 [Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 \(p. 620\)](#)의 절차를 수행합니다.

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.

Aurora 글로벌 데이터베이스의 데이터베이스 사용자가 Amazon S3에 액세스하도록 허용하려면 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)에서 생성한 역할을 글로벌 데이터베이스의 각 Aurora 클러스터와 연결합니다.

Amazon S3로의 발신 연결을 허용하도록 글로벌 데이터베이스의 각 Aurora 클러스터를 구성합니다. 자침은 [Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 \(p. 619\)](#) 단원을 참조하십시오.

쿼리한 데이터를 S3에 저장

Aurora 글로벌 데이터베이스를 구성하는 모든 Aurora 클러스터에 대해 [Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 \(p. 627\)](#)의 절차를 수행합니다.

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.

Aurora 글로벌 데이터베이스의 데이터베이스 사용자가 Amazon S3에 액세스하도록 허용하려면 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)에서 생성한 역할을 글로벌 데이터베이스의 각 Aurora 클러스터와 연결합니다.

Amazon S3로의 발신 연결을 허용하도록 글로벌 데이터베이스의 각 Aurora 클러스터를 구성합니다. 자침은 [Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 \(p. 619\)](#) 단원을 참조하십시오.

Amazon Aurora를 사용한 복제

Aurora에서는 몇 가지 복제 옵션이 제공됩니다. 다음 단원에서는 각 기법을 선택하는 방법과 시기를 설명합니다.

Aurora 복제본

Aurora 복제본은 Aurora DB 클러스터의 듀립 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 인스턴스 및 Aurora 복제본에는 클러스터 볼륨 데이터가 단 하나의 논리 볼륨으로 표시됩니다.

그 결과, 모든 Aurora 복제본은 복제본 지연 시간을 최소화하여 쿼리 결과에 대한 데이터를 동일하게 반환합니다. 이 지연 시간은 기본 인스턴스가 업데이트를 적용한 후 100밀리초 미만이지만 데이터베이스 변경률에 따라 달라집니다. 즉, 데이터베이스의 쓰기 연산이 많은 기간에는 복제본 지연 시간이 증가할 수 있습니다.

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 복제본이 기본 인스턴스로 승격됩니다. 기본 인스턴스에 대한 읽기 및 쓰기 요청이 예외적으로 실패하면서 인스턴스가 아주 잠시 중단되고, 이때 Aurora 복제본이 다시 부팅됩니다. Aurora DB 클러스터에 Aurora 복제본이 포함되어 있지 않으면 DB 인스턴스가 장애 이벤트에서 복구되는 기간 동안 DB 클러스터를 사용할 수 없게 됩니다. 하지만 Aurora 복제본을 승격시키는 것이 기본 인스턴스를 재생성하는 것보다 훨씬 빠릅니다. 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 내결합 \(p. 268\)](#) 단원을 참조하십시오.

Note

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

Aurora 복제본 생성 방법에 대한 자세한 내용은 [DB 클러스터에 Aurora 복제본 추가 \(p. 208\)](#) 단원을 참조하십시오.

Aurora MySQL를 사용한 복제

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

- 다른 AWS 리전에 속하는 Aurora MySQL DB 클러스터의 Aurora 읽기 전용 복제본을 생성하여 다른 AWS 리전에 Aurora MySQL DB 클러스터 2개를 설정합니다.
- MySQL 바이너리 로그(binlog) 복제를 사용하여 동일한 리전에 있는 Aurora MySQL DB 클러스터 2개
- Amazon RDS MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하여 마스터인 Aurora MySQL MySQL DB 인스턴스와 Amazon RDS DB 클러스터를 설정합니다. 일반적으로 이 방법은 진행 중인 복제 보다는 Aurora MySQL로의 마이그레이션에 사용됩니다.

Aurora MySQL을 사용한 복제에 대한 자세한 내용은 [Amazon Aurora MySQL를 사용하는 단일 마스터 복제 \(p. 554\)](#) 단원을 참조하십시오.

Aurora PostgreSQL를 사용한 복제

Aurora 외에도 마스터인 Amazon RDS PostgreSQL DB 인스턴스와 Aurora PostgreSQL DB 클러스터 사이에 복제를 설정하는 방법이 있습니다. Amazon RDS PostgreSQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하면 됩니다.

Aurora PostgreSQL을 사용한 복제에 대한 자세한 내용은 [Amazon Aurora PostgreSQL를 사용한 복제 \(p. 807\)](#) 단원을 참조하십시오.

Aurora에 대한 DB 인스턴스 결제

Aurora 클러스터의 Amazon RDS 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 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 참조하십시오.
- I/O 요청(월별 1백만 요청별) – 결제 주기에 요청한 총 스토리지 I/O 요청 수입니다. 해당됩니다.
- 백업 스토리지(월별 GiB별) – 백업 스토리지는 자동화된 데이터베이스 백업 및 생성한 활성 데이터베이스 스냅샷과 연결된 스토리지입니다. 백업 보존 기간을 연장하거나 추가 데이터베이스 스냅샷을 찍으면 데이

터베이스가 사용하는 백업 스토리지가 증가합니다. 초 단위 결제는 백업 스토리지(GB-월 단위로 측정됨)에는 적용되지 않습니다.

자세한 정보는 [Amazon Aurora DB 클러스터 백업 및 복구 \(p. 268\)](#) 단원을 참조하십시오.

- 데이터 전송(GB별) – DB 인스턴스와 인터넷 및 기타 AWS 리전 간의 양방향 데이터 전송입니다.

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.

Aurora요금에 대한 자세한 내용은 [Aurora 요금 페이지](#)를 참조하십시오.

주제

- [Aurora용 온디맨드 DB 인스턴스 \(p. 57\)](#)
- [Aurora용 예약 DB 인스턴스 \(p. 57\)](#)

Aurora용 온디맨드 DB 인스턴스

Amazon RDS 온디맨드 DB 인스턴스는 DB 인스턴스 클래스를 기반으로 청구됩니다(예: db.t2.small 또는 db.m4.large). Amazon RDS 요금에 대한 자세한 정보는 [Amazon RDS 제품 페이지](#)를 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 [DB 인스턴스 상태 \(p. 344\)](#) 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

DB 인스턴스가 다중 AZ 배포가 되도록 지정하면 Amazon RDS 요금 페이지에 게시된 다중 AZ 요금에 따라 청구됩니다.

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 인

스턴스가 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 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 예약 DB 인스턴스의 사양과 일치해야 합니다. 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.

요금을 포함하여 예약 DB 인스턴스에 대한 자세한 내용은 [Amazon RDS 예약 인스턴스](#)를 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.

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 결제 및 비용 관리 사용 설명서에서 [Amazon RDS 예약 DB 인스턴스](#)를 참조하십시오.

유연한 크기의 예약 DB 인스턴스

예약 DB 인스턴스를 구매할 때 지정해야 하는 것 중 하나가 인스턴스 클래스(db.m4.large 등)입니다. 인스턴스 클래스에 대한 자세한 내용은 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 참조하십시오.

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.m4.large의 예약 DB 인스턴스는 db.m4.large에 적용할 수 있지만, db.m5.large에는 적용할 수 없습니다. db.m4와 db.m5는 다른 인스턴스 클래스 유형이기 때문입니다. 이러한 예약 DB 인스턴스의 이점은 다중 AZ와 단일 AZ 구성 모두에게 적용됩니다.

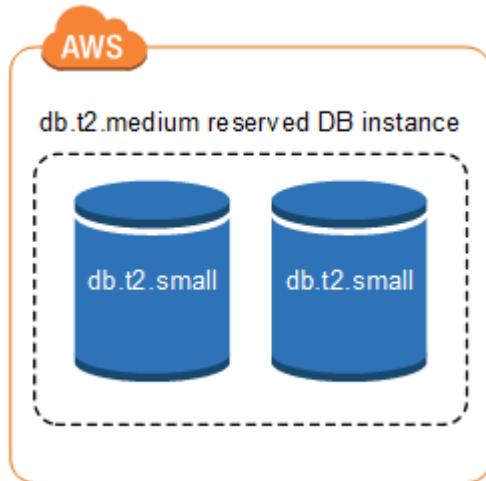
유연한 크기의 예약 DB 인스턴스는 다음 Aurora 데이터베이스 엔진에서 제공됩니다.

- Aurora MySQL
- Aurora PostgreSQL

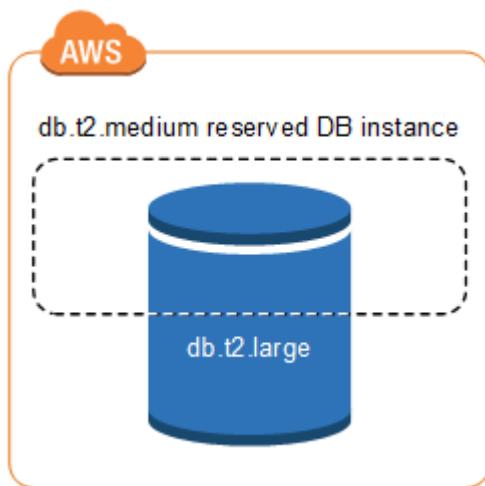
예약 DB 인스턴스의 크기에 따른 사용량은 정규화 유닛을 사용하여 비교할 수 있습니다. 예를 들어 db.m3.large DB 인스턴스 2개일 때 사용량의 정규화 유닛 1개는 db.m3.small 1개일 때 사용량의 정규화 유닛 8개와 같습니다. 다음 표는 DB 인스턴스 크기에 따른 정규화 유닛의 수를 나타낸 것입니다.

인스턴스 크기	단일 AZ 정규화 유닛	다중 AZ 정규화 유닛
micro	0.5	1
small	1	2
medium	2	4
large	4	8
xlarge	8	16
2xlarge	16	32
4xlarge	32	64
8xlarge	64	128
10xlarge	80	160
16xlarge	128	256

예약 DB 인스턴스로 db.t2.medium을 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.



예약 DB 인스턴스 결제의 예

예약 DB 인스턴스의 요금에는 스토리지, 백업, IO와 관련된 정기 비용이 포함되지 않습니다. 다음 예는 예약 DB 인스턴스의 월 총 요금을 보여 줍니다.

- 미국 동부(버지니아 북부)의 Aurora MySQL 예약 단일 AZ db.r4.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 선택한 경우, 미국 동부(버지니아 북부)의 Aurora MySQL 단일 AZ db.r4.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.

Note

이 예의 요금은 예제 요금이며 실제 요금과 다를 수 있습니다.

Aurora요금에 대한 자세한 내용은 [Aurora 요금 페이지](#)를 참조하십시오.

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.

예약 DB 인스턴스에 대해 선결제를 하면 사용할 리소스가 예약됩니다. 이러한 리소스는 예약되므로 사용 여부에 관계없이 리소스에 대한 요금이 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.

콘솔

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

사용 가능한 예약 DB 인스턴스 상품에 대한 요금과 정보를 가져오려면

1. AWS Management 콘솔에 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. 다중 AZ 배포에서 다중 AZ 배포를 사용할지 여부를 선택합니다.

Note

예약된 Amazon Aurora 인스턴스에서는 Multi-AZ deployment(다중 AZ 배포) 옵션을 항상 No로 설정합니다. 예약 DB 인스턴스에서 Amazon Aurora DB 클러스터를 생성할 때 DB 클러스터가 자동으로 다중 AZ로 생성됩니다.

7. [Term]에서 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 콘솔에 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. 다중 AZ 배포에서 다중 AZ 배포를 사용할지 여부를 선택합니다.

Note

예약된 Amazon Aurora 인스턴스에서는 Multi-AZ deployment(다중 AZ 배포) 옵션을 항상 No로 설정합니다. 예약 DB 인스턴스에서 Amazon Aurora DB 클러스터를 생성할 때 DB 클러스터가 자동으로 다중 AZ로 생성됩니다.

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

상품 유형을 선택하면 아래와 같이 요금 정보가 표시됩니다.

Purchase Reserved DB Instances

Choose from the options below, then enter the number of DB instances you wish to reserve with this order. When you are done, click the Continue button.

Options

Product description

aurora-mysql

DB instance class

db.r4.4xlarge — 16 vCPU, 122 GiB RAM

Multi AZ deployment

Multi-AZ deployment model is not applicable for this database engine and edition

- Yes
 No

Term

1 year

Offering type

All Upfront

Reserved Id (optional)

Optional tag to track your reservation

Number of DB instances

1

Pricing details

One-time payment (per instance)

Total one-time payment*

*Additional taxes may apply

Normalized units per hour [info](#)

32

Usage charges*

USD (hourly)

*Additional taxes may apply

This hourly rate is charged for every hour for each instance in the Reserved Instance term you purchase, regardless of instance usage

Charges for your usage will appear on your monthly bill.

Cancel

Continue

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. [Continue]를 선택합니다.

[Purchase Reserved DB Instance] 대화 상자가 나타나면서 아래와 같이 선택한 예약 DB 인스턴스 속성에 대한 요약 내용과 지불 기한이 각각 표시됩니다.

Purchase Reserved DB Instances

Summary of Purchase

You are about to purchase a Reserved DB Instance with the following information.

Region

US East (N. Virginia)

Product Description

aurora-mysql

DB Instance Class

db.r4.4xlarge

Offering Type

All Upfront

Multi AZ Deployment

No

Term

1 year

Reserved DB Instance

default

Quantity

1

Price Per Instance

[REDACTED]

Total Payment Due Now

[REDACTED]

⚠️ Purchasing this Reserved DB Instance will charge [REDACTED] to the payment method associated with this Amazon Web Services account. Are you sure you would like to proceed?

Cancel

Back

Purchase

11. 확인 페이지에서 자신이 예약한 DB 인스턴스를 확인합니다. 정보가 정확하면 [Purchase]를 선택하여 예약한 DB 인스턴스를 구매합니다.

또는 [Back]을 선택하여 예약한 DB 인스턴스를 편집합니다.

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 콘솔에 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 상품 보기

구매 가능한 DB 인스턴스 상품에 대한 정보를 가져오려면 AWS CLI 명령 `describe-reserved-db-instances-offerings`를 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.m1.large	y	1y	1820.00
USD	0.368	mysql	Partial Upfront		
OFFERING	649fd0c8-cf6d-47a0-bfa6-060f8e75e95f	db.m1.small	n	1y	227.50
USD	0.046	mysql	Partial Upfront		
OFFERING	123456cd-ab1c-47a0-bfa6-12345667232f	db.m1.small	n	1y	162.00
USD	0.00	mysql	All Upfront		
Recurring Charges:	Amount	Currency	Frequency		
Recurring Charges:	0.123	USD	Hourly		
OFFERING	123456cd-ab1c-37a0-bfa6-12345667232d	db.m1.large	y	1y	700.00
USD	0.00	mysql	All Upfront		
Recurring Charges:	Amount	Currency	Frequency		
Recurring Charges:	1.25	USD	Hourly		
OFFERING	123456cd-ab1c-17d0-bfa6-12345667234e	db.m1.xlarge	n	1y	4242.00
USD	2.42	mysql	No Upfront		

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

Example 예약 DB 인스턴스 구입

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

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

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

Linux, OS X, Unix의 경우:

```
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.m1.small	y	2011-12-19T00:30:23.247Z	1y
455.00 USD	0.092 USD	1	payment-pending	mysql	Partial Upfront

예약한 DB 인스턴스를 구매한 후에는 다음 예제와 같이 예약한 DB 인스턴스에 대한 정보를 가져올 수 있습니다.

Example 예약 DB 인스턴스 보기

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

```
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.m1.small	y	2011-12-09T23:37:44.720Z	1y
455.00 USD	0.092 USD	1	retired	mysql	Partial Upfront

RDS API

예약 DB 인스턴스에 대한 작업은 다음 예제와 같이 RDS API를 사용하여 진행할 수 있습니다.

Example 사용 가능한 예약 DB 인스턴스 상품 보기

구매할 수 있는 DB 인스턴스 예약 상품에 대한 정보를 가져오려면 Amazon RDS API 함수 `DescribeReservedDBInstancesOfferings`를 호출합니다.

```
https://rds.us-east-1.amazonaws.com/
?Action=DescribeReservedDBInstancesOfferings
&SignatureMethod=HmacSHA256
&SignatureVersion=4
```

```
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140411/us-east-1/rds/aws4_request
&X-Amz-Date=20140411T203327Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=545f04acffeb4b80d2e778526b1c9da79d0b3097151c24f28e83e851d65422e2
```

이 호출은 다음과 비슷한 출력을 반환합니다.

```
<DescribeReservedDBInstancesOfferingsResponse xmlns="http://rds.amazonaws.com/doc/2014-10-31/">
  <DescribeReservedDBInstancesOfferingsResult>
    <ReservedDBInstancesOfferings>
      <ReservedDBInstancesOffering>
        <Duration>31536000</Duration>
        <OfferingType>Partial Upfront</OfferingType>
        <CurrencyCode>USD</CurrencyCode>
        <RecurringCharges/>
        <FixedPrice>1820.0</FixedPrice>
        <ProductDescription>mysql</ProductDescription>
        <UsagePrice>0.368</UsagePrice>
        <MultiAZ>true</MultiAZ>
        <ReservedDBInstancesOfferingId>438012d3-4052-4cc7-b2e3-8d3372e0e706</
      ReservedDBInstancesOfferingId>
        <DBInstanceClass>db.m1.large</DBInstanceClass>
      </ReservedDBInstancesOffering>
      <ReservedDBInstancesOffering>
        <Duration>31536000</Duration>
        <OfferingType>Partial Upfront</OfferingType>
        <CurrencyCode>USD</CurrencyCode>
        <RecurringCharges/>
        <FixedPrice>227.5</FixedPrice>
        <ProductDescription>mysql</ProductDescription>
        <UsagePrice>0.046</UsagePrice>
        <MultiAZ>false</MultiAZ>
        <ReservedDBInstancesOfferingId>649fd0c8-cf6d-47a0-bfa6-060f8e75e95f</
      ReservedDBInstancesOfferingId>
        <DBInstanceClass>db.m1.small</DBInstanceClass>
      </ReservedDBInstancesOffering>
    </ReservedDBInstancesOfferings>
  </DescribeReservedDBInstancesOfferingsResult>
  <ResponseMetadata>
    <RequestId>5e4ec40b-2978-11e1-9e6d-771388d6ed6b</RequestId>
  </ResponseMetadata>
</DescribeReservedDBInstancesOfferingsResponse>
```

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

Example 예약 DB 인스턴스 구입

예약 DB 인스턴스를 구매하려면 Amazon RDS API 작업 [PurchaseReservedDBInstancesOffering](#)을 다음 파라미터와 함께 호출합니다.

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

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

```
https://rds.us-east-1.amazonaws.com/
?Action=PurchaseReservedDBInstancesOffering
&ReservedDBInstanceId=MyReservation
&ReservedDBInstancesOfferingId=438012d3-4052-4cc7-b2e3-8d3372e0e706
&DBInstanceCount=10
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140415/us-east-1/rds/aws4_request
&X-Amz-Date=20140415T232655Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=c2ac761e8c8f54a8c0727f5a87ad0a766fbb0024510b9aa34ea6d1f7df52fb11
```

이 호출은 다음과 비슷한 출력을 반환합니다.

```
<PurchaseReservedDBInstancesOfferingResponse xmlns="http://rds.amazonaws.com/
doc/2014-10-31/">
  <PurchaseReservedDBInstancesOfferingResult>
    <ReservedDBInstance>
      <OfferingType>Partial Upfront</OfferingType>
      <CurrencyCode>USD</CurrencyCode>
      <RecurringCharges/>
      <ProductDescription>mysql</ProductDescription>
      <ReservedDBInstancesOfferingId>649fd0c8-cf6d-47a0-bfa6-060f8e75e95f</
      ReservedDBInstancesOfferingId>
      <MultiAZ>true</MultiAZ>
      <State>payment-pending</State>
      <ReservedDBInstanceId>MyReservation</ReservedDBInstanceId>
      <DBInstanceCount>10</DBInstanceCount>
      <StartTime>2011-12-18T23:24:56.577Z</StartTime>
      <Duration>31536000</Duration>
      <FixedPrice>123.0</FixedPrice>
      <UsagePrice>0.123</UsagePrice>
      <DBInstanceClass>db.m1.small</DBInstanceClass>
    </ReservedDBInstance>
  </PurchaseReservedDBInstancesOfferingResult>
  <ResponseMetadata>
    <RequestId>7f099901-29cf-11e1-bd06-6fe008f046c3</RequestId>
  </ResponseMetadata>
</PurchaseReservedDBInstancesOfferingResponse>
```

예약한 DB 인스턴스를 구매한 후에는 다음 예제와 같이 예약한 DB 인스턴스에 대한 정보를 가져올 수 있습니다.

Example 예약 DB 인스턴스 보기

자신의 AWS 계정에 대해 예약된 DB 인스턴스에 대한 자세한 내용을 보려면 Amazon RDS API 작업 [DescribeReservedDBInstances](#)를 호출합니다.

```
https://rds.us-west-2.amazonaws.com/
?Action=DescribeReservedDBInstances
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-09-01
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140420/us-west-2/rds/aws4_request
&X-Amz-Date=20140420T162211Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=3312d17a4c43bcd209bc22a0778dd23e73f8434254abbd7ac53b89ade3dae88e
```

이 API는 다음과 비슷한 출력을 반환합니다.

```
<DescribeReservedDBInstancesResponse xmlns="http://rds.amazonaws.com/doc/2014-10-31/">
<DescribeReservedDBInstancesResult>
  <ReservedDBInstances>
    <ReservedDBInstance>
      <OfferingType>Partial Upfront</OfferingType>
      <CurrencyCode>USD</CurrencyCode>
      <RecurringCharges/>
      <ProductDescription>mysql</ProductDescription>
      <ReservedDBInstancesOfferingId>649fd0c8-cf6d-47a0-bfa6-060f8e75e95f</
      ReservedDBInstancesOfferingId>
      <MultiAZ>false</MultiAZ>
      <State>payment-failed</State>
      <ReservedDBInstanceId>MyReservation</ReservedDBInstanceId>
      <DBInstanceCount>1</DBInstanceCount>
      <StartTime>2010-12-15T00:25:14.131Z</StartTime>
      <Duration>31536000</Duration>
      <FixedPrice>227.5</FixedPrice>
      <UsagePrice>0.046</UsagePrice>
      <DBInstanceClass>db.m1.small</DBInstanceClass>
    </ReservedDBInstance>
    <ReservedDBInstance>
      <OfferingType>Partial Upfront</OfferingType>
      <CurrencyCode>USD</CurrencyCode>
      <RecurringCharges/>
      <ProductDescription>mysql</ProductDescription>
      <ReservedDBInstancesOfferingId>649fd0c8-cf6d-47a0-bfa6-060f8e75e95f</
      ReservedDBInstancesOfferingId>
      <MultiAZ>false</MultiAZ>
      <State>payment-failed</State>
      <ReservedDBInstanceId>MyReservation</ReservedDBInstanceId>
      <DBInstanceCount>1</DBInstanceCount>
      <StartTime>2010-12-15T01:07:22.275Z</StartTime>
      <Duration>31536000</Duration>
      <FixedPrice>227.5</FixedPrice>
      <UsagePrice>0.046</UsagePrice>
      <DBInstanceClass>db.m1.small</DBInstanceClass>
    </ReservedDBInstance>
  </ReservedDBInstances>
</DescribeReservedDBInstancesResult>
<ResponseMetadata>
  <RequestId>23400d50-2978-11e1-9e6d-771388d6ed6b</RequestId>
</ResponseMetadata>
</DescribeReservedDBInstancesResponse>
```

Amazon Aurora 환경 설정

Amazon Aurora를 처음 사용한다면 먼저 다음 작업을 완료해야 합니다.

1. AWS에 가입 (p. 69)
2. IAM 사용자 생성 (p. 69)
3. 요구 사항 결정 (p. 71)
4. 보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다. (p. 72)

AWS에 가입

Amazon RDS(AWS)에 가입하면 Amazon RDS를 포함한 AWS의 모든 서비스에 AWS 계정이 자동으로 등록됩니다. 사용한 서비스에 대해서만 청구됩니다.

Amazon RDS에서는 사용한 리소스에 대해서만 비용을 지불합니다. 생성한 Amazon RDS DB 클러스터는 실시간으로 활성화됩니다(샌드박스에서 실행되지 않음). 종료하기 전까지 해당 클러스터에 대해 표준 Amazon RDS 사용 요금이 발생합니다. Amazon RDS 사용 요금에 대한 자세한 정보는 [Amazon RDS 제품 페이지](#)를 참조하십시오. AWS를 처음 사용하는 고객인 경우에는 Amazon RDS를 무료로 사용해볼 수 있습니다. 자세한 정보는 [AWS 프리 티어](#) 단원을 참조하십시오.

이미 AWS 계정이 있다면 다음 작업으로 건너뛰십시오. AWS 계정이 없는 경우에는 아래 단계를 수행하여 계정을 만드십시오.

AWS 계정을 만들려면 다음을 수행합니다.

1. <https://portal.aws.amazon.com/billing/signup>을 엽니다.
2. 온라인 지시 사항을 따릅니다.

등록 절차 중 전화를 받고 전화 키패드를 사용하여 확인 코드를 입력하는 과정이 있습니다.

다음 작업에 필요하므로 AWS 계정 번호를 기록합니다.

IAM 사용자 생성

Amazon RDS 등의 AWS 서비스에 액세스하려면 자격 증명을 제공해야 합니다. 리소스에 대한 액세스 권한이 있는지 여부를 파악해야 하기 때문입니다. 콘솔은 암호를 요구합니다. AWS 계정에 대한 액세스 키를 생성하면 명령줄 인터페이스 또는 API에 액세스할 수 있습니다. 그러나 AWS 계정에 자격 증명을 사용하여 AWS에 액세스하지 말고, AWS Identity and Access Management(IAM)를 사용하는 것이 좋습니다. IAM 사용자를 생성하여 관리자 권한과 함께 IAM 그룹에 추가하거나, 이 사용자에게 관리자 권한을 부여하십시오. 그러면 IAM 사용자의 특정 URL이나 자격 증명을 사용하여 AWS에 액세스할 수 있습니다.

AWS에 가입했지만 IAM 사용자를 생성하지 않았다면 IAM 콘솔에서 생성할 수 있습니다.

관리자 사용자를 직접 생성하여 관리자 그룹에 추가하려면(콘솔)

1. <https://console.aws.amazon.com/iam/>에서 AWS 계정 이메일 주소 및 비밀번호를 AWS 계정 루트 사용자로 사용하여 IAM 콘솔에 로그인합니다.

Note

Administrator IAM 사용자를 사용하는 아래 모범 사례를 준수하고, 루트 사용자 자격 증명을 안전하게 보관해 두는 것이 좋습니다. 몇 가지 [계정 및 서비스 관리 작업](#)을 수행하려면 반드시 루트 사용자로 로그인해야 합니다.

2. 탐색 창에서 사용자와 사용자 추가를 차례로 선택합니다.
3. 사용자 이름에 **Administrator**를 입력합니다.
4. AWS Management 콘솔 액세스 옆의 확인란을 선택합니다. 그런 다음 Custom password(사용자 지정 암호)를 선택하고 텍스트 상자에 새 암호를 입력합니다.
5. (선택 사항) 기본적으로 AWS에서는 새 사용자가 처음 로그인할 때 새 암호를 생성해야 합니다. User must create a new password at next sign-in(사용자가 다음에 로그인할 때 새 암호를 생성해야 합니다) 옆에 있는 확인란의 선택을 취소하면 새 사용자가 로그인한 후 암호를 재설정할 수 있습니다.
6. Next: Permissions(다음: 권한)를 선택합니다.
7. 권한 설정 아래에서 그룹에 사용자 추가를 선택합니다.
8. 그룹 생성을 선택합니다.
9. 그룹 생성 대화 상자의 그룹 이름에 **Administrators**를 입력합니다.
10. Filter policies(정책 필터링)을 선택한 다음 AWS managed -job function(AWS 관리형 -job 함수)을 선택하여 테이블 내용을 필터링합니다.
11. 정책 목록에서 AdministratorAccess 확인란을 선택합니다. 그런 다음 Create group을 선택합니다.

Note

AdministratorAccess 권한을 사용하여 AWS Billing and Cost Management 콘솔에 액세스 하려면 먼저 결제에 대한 IAM 사용자 및 역할 액세스를 활성화해야 합니다. 이를 위해 [결제 콘솔에 액세스를 위임하기 위한 자습서 1단계](#)의 지침을 따르십시오.

12. 그룹 목록으로 돌아가 새 그룹의 확인란을 선택합니다. 목록에서 그룹을 확인하기 위해 필요한 경우 Refresh(새로 고침)를 선택합니다.
13. Next: Tags(다음: 태그)를 선택합니다.
14. (선택 사항) 태그를 키-값 페어로 연결하여 메타데이터를 사용자에게 추가합니다. IAM에서 태그 사용에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 개체 태그 지정](#)을 참조하십시오.
15. Next: Review(다음: 검토)를 선택하여 새 사용자에 추가될 그룹 멤버십의 목록을 확인합니다. 계속 진행 할 준비가 되었으면 Create user를 선택합니다.

이와 동일한 절차에 따라 그룹 및 사용자를 추가로 생성하고 사용자에게 AWS 계정 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 특정 AWS 리소스에 대한 사용자 권한을 제한하는 정책을 사용하는 방법을 알아보려면 [액세스 관리 및 정책 예제](#)를 참조하십시오.

이 새로운 IAM 사용자로 로그인하려면 먼저 AWS 콘솔에서 로그아웃한 후 다음 URL을 사용합니다. 여기에서 your_aws_account_id는 하이픈을 제외한 AWS 계정 번호를 나타냅니다. 예를 들어, AWS 계정 번호가 1234-5678-9012이면 계정 ID는 123456789012입니다.

```
https://your_aws_account_id.signin.aws.amazon.com/console/
```

방금 생성한 IAM 사용자 이름과 암호를 입력합니다. 로그인하면 탐색 모음에 "your_user_name @ your_aws_account_id"가 표시됩니다.

로그인 페이지의 URL에 AWS 계정 ID가 포함되지 않게 하려면 계정 별칭을 생성할 수 있습니다. IAM 대시보드에서 사용자 지정을 선택하고 회사 이름 등의 별칭을 입력합니다. 계정 별칭을 만든 후 로그인하려면 다음 URL을 사용하십시오.

```
https://your_account_alias.signin.aws.amazon.com/console/
```

본인 계정의 IAM 사용자 로그인 링크를 확인하려면 IAM 콘솔을 열고 대시보드의 [AWS Account Alias] 아래를 체크합니다.

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 할당합니다. 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.

보안 그룹을 생성하거나 DB 클러스터를 생성하려면 DB 클러스터 및 네트워크 요구 사항을 사전에 반드시 알고 있어야 합니다. 요구 사항의 예를 들면 다음과 같습니다.

- 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 자세한 내용은 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 참조하십시오.
- DB 클러스터는 가상 사설 클라우드(VPC)에 있습니다. DB 클러스터에 연결하려면 보안 그룹 규칙을 구성해야 합니다. 다음은 각 VPC 옵션의 규칙을 설명한 목록입니다.
 - 기본 VPC — 현재 리전에서 AWS 계정에 기본 VPC가 있는 경우 해당 VPC를 구성하여 DB 클러스터를 지원할 수 있습니다. DB 클러스터 생성 시 기본 VPC를 지정할 경우 다음과 같이 실행해야 합니다.
 - VPC 보안 그룹을 생성하여 데이터베이스를 이용해 애플리케이션 또는 서비스에서 Aurora DB 클러스터까지 연결 권한을 부여해야 합니다. VPC 보안 그룹을 생성하려면 VPC 콘솔에서 [Amazon EC2 API](#) 또는 보안 그룹 옵션을 사용해야 합니다. 자세한 정보는 [4단계: VPC 보안 그룹 만들기 \(p. 1019\)](#) 단원을 참조하십시오.
 - 기본 DB 서브넷 그룹을 지정해야 합니다. 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.
 - VPC 보안 그룹을 생성하여 데이터베이스를 이용해 애플리케이션 또는 서비스에서 Aurora DB 클러스터까지 연결 권한을 부여해야 합니다. VPC 보안 그룹을 생성하려면 VPC 콘솔에서 [Amazon EC2 API](#) 또는 보안 그룹 옵션을 사용해야 합니다. 자세한 정보는 [4단계: VPC 보안 그룹 만들기 \(p. 1019\)](#) 단원을 참조하십시오.
 - VPC가 DB 클러스터를 호스팅하려면 별도의 가용 영역에서 각각 최소 2개 이상씩 서브넷을 구성하는 등 특정 요구 사항을 충족해야 합니다. 자세한 정보는 [Amazon Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#) 단원을 참조하십시오.
 - DB 서브넷 그룹을 지정하여 DB 클러스터에서 VPC를 사용할 서브넷을 정의해야 합니다. 자세한 정보는 [VPC에서 DB 인스턴스를 사용한 작업 \(p. 1016\)](#)의 DB 서브넷 그룹 단원을 참조하십시오.
 - 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 정보는 [Aurora를 위한 고가용성 \(p. 36\)](#) 단원을 참조하십시오.
 - 귀하의 AWS 계정에 Amazon RDS 연산 수행에 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 정보는 [Amazon Aurora의 Identity and Access Management\(IAM\) \(p. 960\)](#) 단원을 참조하십시오.
 - 데이터베이스는 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.
 - 데이터베이스를 구성하려고 하는 리전은 어디입니까? 애플리케이션이나 웹 서비스에 가깝게 데이터베이스를 구성하면 네트워크 지연 시간을 줄일 수 있습니다.

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

보안 그룹을 생성하여 VPC 내의 DB 클러스터에 대한 액세스를 제공합니다.

DB 클러스터는 VPC에서 생성될 가능성이 가장 높습니다. 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.

DB 보안 그룹을 요구하는 레거시 DB 클러스터(VPC에서 실행되지 않음)가 아니라면 생성해야 하는 보안 그룹은 VPC 보안 그룹입니다. 2013년 3월 이후 AWS 계정을 생성하였다면 기본 VPC가 지원되므로 DB 클러스터를 이 VPC 내에 생성할 수 있습니다. VPC 내에 DB 클러스터를 생성하려면 클러스터에 액세스할 수 있는 규칙을 VPC 보안 그룹에 추가해야 합니다.

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 클러스터에 애플리케이션이 있다면 Amazon EC2 클러스터에 대해 설정한 VPC 보안 그룹을 사용할 수 있습니다.

VPC 보안 그룹의 생성 방법

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/vpc>에서 Amazon VPC 콘솔을 엽니다.
2. AWS Management 콘솔 상단 오른쪽 모서리에서 VPC 보안 그룹과 DB 클러스터를 생성할 리전을 선택합니다. 해당 리전의 Amazon VPC 리소스 목록에 1개 이상의 VPC와 몇 개의 서브넷이 있는 것으로 표시되어야 합니다. 그렇지 않으면 해당 리전에 기본 VPC가 없는 것입니다.
3. 탐색 창에서 [Security Groups]를 선택합니다.
4. [Create Security Group]을 선택합니다.
5. [Create Security Group] 창에서 보안 그룹의 Name tag, Group name 및 Description을 입력합니다. DB 클러스터를 생성하려는 VPC를 선택합니다. [Yes, Create]를 선택합니다.
6. 생성한 VPC 보안 그룹이라고 해도 여전히 선택할 수 있어야 합니다. 콘솔 화면 하단 세부 정보 창에 보안 그룹 세부 정보를 비롯해 인바운드 및 아웃바운드 규칙 작업에 대한 탭이 표시됩니다. 인바운드 규칙 탭을 선택합니다.
7. Inbound Rules 탭에서 [Edit]를 선택합니다. [Custom TCP Rule]을 [Type] 목록에서 선택합니다. 포트 범위 템스트 상자에 DB 클러스터에 사용할 포트 값을 입력한 후 소스 템스트 상자에 클러스터에 액세스하는 위치의 IP 주소 범위(CIDR 값)를 입력하거나, 보안 그룹 이름을 선택합니다.
8. IP 주소를 더 추가하거나 다른 포트 범위를 추가할 경우에는 Add another rule(다른 규칙 추가)을 선택합니다.
9. 필요에 따라 [Outbound Rules] 탭을 사용하여 아웃바운드 트래픽 규칙을 추가할 수도 있습니다.
10. 모두 마쳤으면 설정을 변경한 각 탭에서 저장을 클릭합니다.

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

마지막으로 VPC 서브넷에 대해 간략히 언급하자면, 기본 VPC를 사용하는 경우에는 VPC의 모든 서브넷을 아우르는 기본 서브넷 그룹이 이미 생성되어 있습니다. DB 클러스터를 생성할 때 기본값 VPC를 선택하고 DB 서브넷 그룹의 기본값을 사용할 수 있습니다.

모든 설정 요구 사항을 마쳤으면 이제 요구 사항과 생성한 보안 그룹을 사용하여 DB 클러스터를 시작할 수 있습니다. DB 클러스터를 만드는 방법에 대한 자세한 정보는 다음 표의 관련 문서를 참조하십시오.

Amazon Aurora Aurora 사용 설명서
보안 그룹을 생성하여 VPC 내의 DB 클
러스터에 대한 액세스를 제공합니다.

설정 후 테스트 Amazon Aurora DB 클러스터를 만들 수 있습니다. 지침은 [DB 클러스터 생성 및 Aurora MySQL DB 클러스터의 데이터베이스에 연결 \(p. 74\)](#)을 참조하십시오.

Amazon Aurora 시작하기

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

여기 나온 절차는 Aurora를 사용하여 시작하기 위한 기본 정보를 제공하는 자습서입니다. 후속 단원에서는 다양한 종류의 앤드포인트와 Aurora 클러스터를 확장/축소하는 방법을 포함하여 Aurora의 개념과 절차에 대한 보다 고급 정보를 제공합니다.

Important

DB 클러스터를 생성하려면 먼저 [Amazon Aurora 환경 설정 \(p. 69\)](#) 단원의 작업을 완료해야 합니다.

주제

- [DB 클러스터 생성 및 Aurora MySQL DB 클러스터의 데이터베이스에 연결 \(p. 74\)](#)
- [DB 클러스터 생성 및 Aurora PostgreSQL DB 클러스터의 데이터베이스에 연결 \(p. 83\)](#)

DB 클러스터 생성 및 Aurora MySQL DB 클러스터의 데이터베이스에 연결

Aurora MySQL DB 클러스터는 Amazon RDS 콘솔을 사용할 때 가장 쉽게 생성할 수 있습니다. DB 클러스터가 생성되었으면 이제 MySQL Workbench 같은 표준 MySQL 유ти리티를 사용해 DB 클러스터의 데이터베이스에 연결할 수 있습니다.

주제

- [Aurora MySQL DB 클러스터 생성 \(p. 74\)](#)
- [DB 클러스터의 인스턴스에 연결하기 \(p. 81\)](#)
- [샘플 DB 클러스터, DB 서브넷 그룹 및 VPC 삭제 \(p. 82\)](#)

Aurora MySQL DB 클러스터 생성

DB 클러스터를 생성하려면 먼저 Amazon VPC 서비스 및 Amazon RDS DB 서브넷 그룹을 기반으로 하는 가상 프라이빗 클라우드(VPC)가 있어야 합니다. VPC는 최소 2개의 사용 영역마다 서브넷이 1개 이상씩 있어야 합니다. 또한 AWS 계정의 기본 VPC를 사용하거나, 직접 자신의 VPC를 생성할 수도 있습니다. Amazon RDS 콘솔은 Amazon Aurora에 사용할 자신의 VPC를 생성하거나, Aurora DB 클러스터에 기존 VPC를 사용하는 방법도 쉽습니다.

Amazon RDS가 VPC와 DB 서브넷 그룹을 생성하도록 하지 않고 Aurora DB 클러스터에서 사용할 VPC와 DB 서브넷 그룹을 직접 생성하고 싶은 경우에는 [Amazon Aurora에 사용할 VPC의 생성 방법 \(p. 1005\)](#)의 지침을 따릅니다. 그렇지 않고 DB 클러스터를 생성한 후 Amazon RDS에서 VPC와 DB 서브넷 그룹을 생성하면 이번 주제의 지침을 따릅니다.

Note

새 콘솔 인터페이스를 데이터베이스 생성에 사용할 수 있습니다. 사용 중인 콘솔에 따라 New Console(새 콘솔) 또는 Original Console(기존 콘솔) 지침을 선택합니다. New Console(새 콘솔) 지침이 기본적으로 열립니다.

새 콘솔

Easy Create(간편 생성)를 활성화 또는 비활성화하여 AWS Management 콘솔에서 MySQL과 호환되는 Aurora 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 예제에서는 Easy Create(간편 생성)가 활성화되어 있습니다. Easy Create(간편 생성)를 활성화 되지 않은 상태에서 Aurora MySQL DB 클러스터를 생성하는 방법은 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

Easy Create(간편 생성)를 활성화한 상태에서 Aurora MySQL DB 클러스터를 생성하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 콘솔의 오른쪽 위 모서리에서 DB 인스턴스를 생성하려는 AWS 리전을 선택합니다.
- Aurora를 모든 AWS 리전에서 사용할 수 있는 것은 아닙니다. Aurora를 사용할 수 있는 AWS 리전을 보려면 [리전 가용성 \(p. 3\)](#) 단원을 참조하십시오.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. 데이터베이스 생성을 선택하고 Easy Create(간편 생성)가 선택되어 있는지 확인합니다.

Create database

Choose a database creation method Info

Standard Create

You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy Create

Use recommended best-practice configuration options can be changed after the database is created.

Configuration

5. Engine type(엔진 유형)에서 Amazon Aurora을 선택합니다.
6. Edition(에디션)에서 Amazon Aurora with MySQL 5.6 compatibility(MySQL 5.6 호환 Amazon Aurora)를 선택합니다.
7. DB instance size(DB 인스턴스 크기)에서 개발/테스트를 선택합니다.
8. DB 클러스터 식별자에 DB 클러스터의 이름을 입력하거나 기본 이름을 그대로 유지합니다.
9. 마스터 사용자 이름에 마스터 사용자의 이름을 입력하거나 기본 이름을 그대로 유지합니다.

데이터베이스 생성 페이지는 다음 이미지와 비슷해야 합니다.

Create database

Choose a database creation method Info

Standard Create

You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy Create

Use recommended best-practice configuration options can be changed after the database is created.

Configuration

Engine type Info

Amazon Aurora



MySQL



MariaDB



PostgreSQL



Oracle

ORACLE®

Microsoft SQL Server



Edition

Amazon Aurora with MySQL 5.6 compatibility

Amazon Aurora with PostgreSQL compatibility

DB instance size

Production

db.r5.2xlarge
8 vCPUs
64 GiB RAM

Dev/Test

db.r5.large
2 vCPUs
16 GiB RAM

Serverless

4-64 GiB RAM

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 this region.

10. DB 클러스터에서 자동 생성된 마스터 암호를 사용하려면 Auto generate a password(암호 자동 생성)를 선택해야 합니다.

마스터 암호를 입력하려면 Auto generate a password(암호 자동 생성) 확인란의 선택을 해제한 다음, 마스터 암호 및 암호 확인에 동일한 암호를 입력합니다.

11. (선택 사항) View default settings for Easy create(간편 생성 기본 설정 보기)를 엽니다.

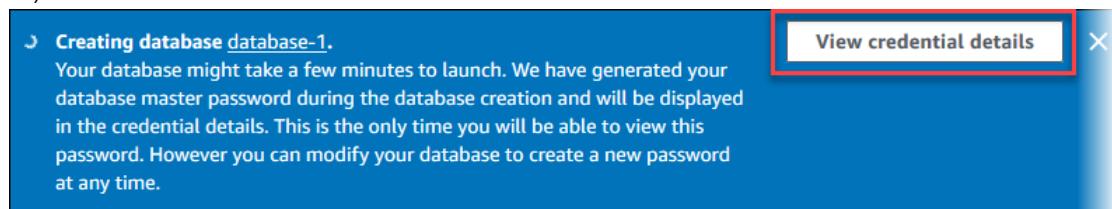
▼ View default settings for Easy create		
Configuration	Value	Editable after database is created
Database Location	Regional	No
Database Features	provisioned	No
Automatic Backups	Enabled	No

Easy Create(간편 생성)가 활성화되었을 때 사용되는 기본 설정을 검토할 수 있습니다. 데이터베이스 생성 도중 하나 이상의 설정을 변경하려면 Standard Create(표준 생성)를 선택하여 설정합니다. Editable after database creation(데이터베이스 생성 후 편집 가능) 열에는 데이터베이스 생성 후 어떤 옵션을 변경할 수 있는지 나와 있습니다. 이 열에 아니요로 표시된 설정을 변경하려면 Standard Create(표준 생성)를 사용합니다. 이 열에 예로 표시된 설정은 Standard Create(표준 생성)를 사용하거나 DB 인스턴스가 생성된 후 인스턴스를 수정하여 해당 설정을 변경할 수 있습니다.

12. 데이터베이스 생성을 선택하십시오.

자동 생성된 암호를 사용하기로 한 경우에는 데이터베이스 페이지에 View credential details(자격 증명 세부 정보 보기) 버튼이 나타납니다.

DB 인스턴스의 마스터 사용자 이름 및 암호를 보려면 View credential details(자격 증명 세부 정보 보기)를 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 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.

13. 데이터베이스에서 새 Aurora MySQL 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.

The screenshot shows the 'Databases' section of the AWS RDS console. There are two entries listed:

DB identifier	Role	Engine	Class	Status	CPU	Current activity
database-1	Cluster	Aurora MySQL	-	Creating		
database-1-instance-1	Reader	Aurora MySQL	db.r5.large	Creating		0 Sessions

The 'Status' column for both entries is circled in red.

기존 콘솔

Aurora MySQL DB 클러스터를 실행하는 방법

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 열니다.
2. AWS Management 콘솔 오른쪽 상단 모서리에서 DB 클러스터를 생성하려는 AWS 리전을 선택합니다. Aurora를 사용할 수 있는 AWS 리전을 보려면 [리전 가용성 \(p. 3\)](#) 단원을 참조하십시오.
3. 탐색 창에서 데이터베이스를 선택합니다.
 탐색 창이 닫혀 있는 경우 왼쪽 상단의 메뉴 아이콘을 선택하여 여십시오.
4. 데이터베이스 생성을 선택하여 엔진 선택 페이지를 여십시오.
5. 엔진 선택 페이지에서 Amazon Aurora를 선택하고 MySQL 호환 에디션을 선택합니다.

Select engine

Engine options

Amazon Aurora
Amazon Aurora

MySQL 

MariaDB 

PostgreSQL 

Oracle 

Microsoft SQL Server 

Amazon Aurora
Amazon Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at <\$1/day.

- Up to 5 times the throughput of MySQL and 3 times the throughput of PostgreSQL
- Up to 64TB of auto-scaling SSD storage
- 6-way replication across three Availability Zones
- Up to 15 Read Replicas with sub-10ms replica lag
- Automatic monitoring and failover in less than 30 seconds

Edition
 MySQL 5.6-compatible
 MySQL 5.7-compatible
 PostgreSQL-compatible

Only enable options eligible for RDS Free Usage Tier [info](#)

[Cancel](#) **Next**

- [Next]를 선택합니다.
- [Specify DB details] 페이지에서 다음과 같이 값을 설정합니다.
 - DB 인스턴스 클래스: db.r4.large
 - DB 인스턴스 식별자: gs-db-instance1
 - Master username: 영숫자 문자를 사용하여 DB 클러스터의 DB 인스턴스에 로그인할 때 사용할 마스터 사용자 이름을 입력합니다.
 - Master password 및 Confirm Password: [Master Password] 상자에 데이터베이스에 로그인할 때 사용할 마스터 사용자 암호를 8~41자의 인쇄 가능한 ASCII 문자(/, " 및 @ 제외)로 입력합니다. 그런 다음 [Confirm Password] 상자에 암호를 다시 한 번 입력합니다.

Specify DB details

Instance specifications

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#).

DB engine
Aurora - compatible with MySQL 5.7.12

DB instance class [info](#)
db.r4.large — 2 vCPU, 15.25 GiB RAM

Multi-AZ deployment [info](#)
 Create Replica in Different Zone
 No

Settings

DB instance identifier [info](#)
Specify a name that is unique for all DB instances owned by your AWS account in the current region.
gs-db-instance1

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance".

Master username [info](#)
Specify an alphanumeric string that defines the login ID for the master user.
myawsuser

Master Username must start with a letter.

Master password [info](#) Confirm password [info](#)
***** *****

Master Password must be at least eight characters long, as in "mypassword".

[Cancel](#) [Previous](#) [Next](#)

8. [Next]를 선택하고 [Configure Advanced Settings] 페이지에 다음과 같이 값을 설정합니다.

- Virtual Private Cloud(VPC): 기존 VPC가 있는 경우, VPC 식별자(예: vpc-a464d1c1)를 선택하여 해당 VPC를 Amazon Aurora DB 클러스터에 사용할 수 있습니다. 기존 VPC 사용 방법에 대한 자세한 내용은 [Amazon Aurora에 사용할 VPC의 생성 방법 \(p. 1005\)](#) 단원을 참조하십시오.

기존 VPC가 없다면 [Create a new VPC]를 선택하여 Amazon RDS에서 VPC를 새로 생성하도록 할 수 있습니다. 이 예에서는 [Create a new VPC] 옵션을 사용합니다.

- 서브넷 그룹: 기존 서브넷 그룹이 있으면 해당 서브넷 그룹 식별자(예: gs-subnet-group1)를 선택하여 Amazon Aurora DB 클러스터에 기존 서브넷 그룹을 사용할 수 있습니다.

기존 서브넷 그룹이 없다면 [Create a new subnet group]을 선택하여 Amazon RDS에서 서브넷 그룹을 새로 생성하도록 할 수 있습니다. 이 예에서는 [Create a new subnet group] 옵션을 사용합니다.

- 퍼블릭 액세스 가능성: Yes

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 No로 설정합니다.

- Availability zone: No Preference
- VPC 보안 그룹: 기존 VPC 보안 그룹이 한 개 이상 있을 경우 해당 VPC 보안 그룹 식별자(예: gs-security-group1)를 선택하여 이러한 VPC 보안 그룹을 Amazon Aurora DB 클러스터에 사용할 수 있습니다.

기존 VPC 보안 그룹이 없다면 Create a new Security group(새 보안 그룹 생성)을 선택하여 Amazon RDS에서 VPC 보안 그룹을 새로 생성하도록 할 수 있습니다. 이 예제에서는 Create a new Security group(새 보안 그룹 생성) 옵션을 사용합니다.

- DB 클러스터 식별자 gs-db-cluster1
- 데이터베이스 이름: sampledb

Note

이것은 기본 데이터베이스를 생성합니다. 추가 데이터베이스를 생성하려면 DB 클러스터에 연결한 다음 SQL 명령어 CREATE DATABASE를 사용하십시오.

- 데이터베이스 포트: 3306

Note

기업 방화벽 뒤에 있어서 Aurora MySQL 기본 포트인 3306 같은 기본 포트에 액세스하지 못할 수도 있습니다. 이런 경우에는 기업 방화벽이 허용하는 포트 값을 입력합니다. 나중에 Aurora DB 클러스터에 연결할 때도 필요하므로 이 포트 값을 기억해야 합니다.

- 나머지 값은 기본값으로 유지하고 데이터베이스 생성을 선택하여 DB 클러스터와 기본 인스턴스를 생성합니다.

DB 클러스터의 인스턴스에 연결하기

Amazon RDS가 DB 클러스터를 프로비저닝하여 기본 인스턴스를 생성한 후에는 무엇이든 표준 SQL 클라이언트 애플리케이션을 사용하여 DB 클러스터의 데이터베이스에 연결할 수 있습니다. 이 예에서는 MySQL 모니터 명령을 사용해 Aurora MySQL DB 클러스터의 데이터베이스에 연결합니다. 연결에 사용할 수 있는 GUI 기반 애플리케이션으로는 MySQL Workbench가 있습니다. 자세한 내용은 [MySQL Workbench 다운로드](#) 페이지 단원을 참조하십시오.

MySQL 모니터를 사용하여 Aurora MySQL DB 클러스터의 데이터베이스에 연결하려면

- AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
- 데이터베이스를 선택하고 DB 클러스터 이름을 선택하여 DB 클러스터 세부 정보를 표시합니다. Connectivity & security(연결 및 보안) 탭에서 쓰기 엔드포인트의 엔드포인트 이름에 대한 값을 복사합니다. 또한 엔드포인트의 포트 번호를 적어둡니다.

Endpoint name	Status	Type	Port
database-1.cluster-ro-... .us-east-2.rds.amazonaws.com	Available	Reader	3306
database-1.cluster-... .us-east-2.rds.amazonaws.com	Available	Writer	3306

3. 클라이언트의 명령 프롬프트에 다음 명령을 입력하여 MySQL 모니터로 Aurora MySQL DB 클러스터의 데이터베이스에 연결합니다. 명령에서 기본 인스턴스에 연결할 클러스터 엔드포인트 그리고 앞에서 생성한 마스터 사용자 이름을 입력합니다. (암호를 입력하라는 메시지가 나타납니다.) 3306 외에 다른 포트 값을 사용할 때는 -P 파라미터 자리에 대신 입력합니다.

```
PROMPT> mysql -h <cluster endpoint> -P 3306 -u <mymasteruser> -p
```

다음과 유사한 출력 화면이 표시되어야 합니다.

```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 350
Server version: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

DB 클러스터 연결에 대한 자세한 내용은 [Amazon Aurora MySQL DB 클러스터 연결 \(p. 163\)](#) 단원을 참조 하십시오.

샘플 DB 클러스터, DB 서브넷 그룹 및 VPC 삭제

생성한 샘플 DB 클러스터에 연결되면 이제 DB 클러스터, DB 서브넷 그룹 및 VPC(생성한 경우)를 삭제할 수 있습니다.

DB 클러스터를 삭제하는 방법

1. AWS Management 콘솔에 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.

DB 서브넷 그룹을 삭제하는 방법

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 서브넷 그룹을 선택한 다음, DB 서브넷 그룹을 선택합니다.
3. 삭제를 선택합니다.
4. 삭제를 선택합니다.

VPC를 삭제하는 방법

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
2. [Your VPCs]를 선택한 후 이 절차에서 생성한 VPC를 선택합니다.
3. 작업에서 Delete VPC(VPC 삭제)를 선택합니다.
4. 삭제를 선택합니다.

DB 클러스터 생성 및 Aurora PostgreSQL DB 클러스터의 데이터베이스에 연결

Aurora PostgreSQL DB 클러스터는 Amazon RDS 콘솔을 사용할 때 가장 쉽게 생성할 수 있습니다. DB 클러스터가 생성되었으면 이제 pgAdmin 같은 표준 PostgreSQL 유ти리티를 사용해 DB 클러스터의 데이터베이스에 연결할 수 있습니다.

주제

- [Aurora PostgreSQL DB 클러스터 생성 \(p. 83\)](#)
- [Aurora PostgreSQL DB 클러스터의 인스턴스에 연결 \(p. 92\)](#)
- [샘플 DB 클러스터, DB 서브넷 그룹 및 VPC 삭제 \(p. 93\)](#)

Aurora PostgreSQL DB 클러스터 생성

DB 클러스터를 생성하려면 먼저 Amazon VPC 서비스 및 Amazon RDS DB 서브넷 그룹을 기반으로 하는 가상 프라이빗 클라우드(VPC)가 있어야 합니다. VPC는 최소 2개의 사용 영역마다 서브넷이 1개 이상씩 있어야 합니다. 또한 AWS 계정의 기본 VPC를 사용하거나, 직접 자신의 VPC를 생성할 수도 있습니다. Amazon RDS 콘솔은 Amazon Aurora에 사용할 자신의 VPC를 생성하거나, Aurora DB 클러스터에 기존 VPC를 사용하는 방법도 쉽습니다.

Amazon RDS가 VPC와 DB 서브넷 그룹을 생성하도록 하지 않고 Amazon Aurora DB 클러스터에서 사용할 VPC와 DB 서브넷 그룹을 직접 생성하고 싶은 경우에는 [Amazon Aurora에 사용할 VPC의 생성 방법](#)

법 (p. 1005)의 지침을 따릅니다. 그렇지 않고 DB 클러스터를 생성한 후 Amazon RDS에서 VPC와 DB 서브 넷 그룹을 생성하려면 이번 주제의 지침을 따릅니다.

Note

새 콘솔 인터페이스를 데이터베이스 생성에 사용할 수 있습니다. 사용 중인 콘솔에 따라 New Console(새 콘솔) 또는 Original Console(기존 콘솔) 지침을 선택합니다. New Console(새 콘솔) 지침이 기본적으로 열립니다.

새 콘솔

Easy Create(간편 생성)를 활성화 또는 비활성화하여 AWS Management 콘솔에서 Aurora PostgreSQL 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.

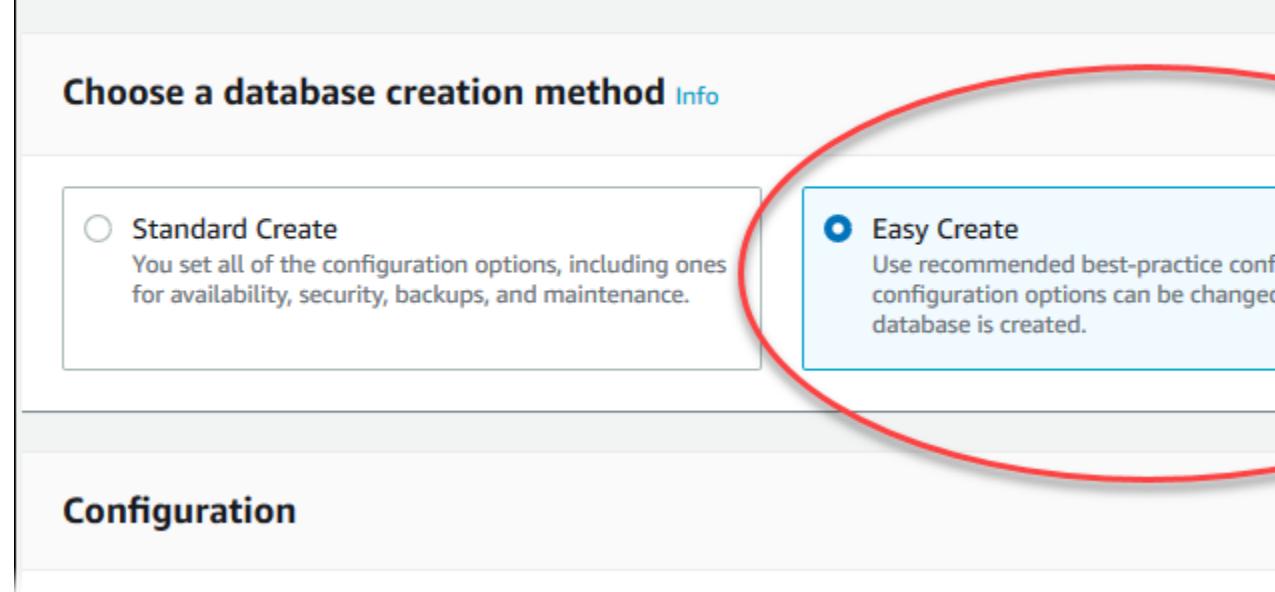
Note

이 예제에서는 Easy Create(간편 생성)가 활성화되어 있습니다. Easy Create(간편 생성)를 활성화되지 않은 상태에서 Aurora PostgreSQL DB 클러스터를 생성하는 방법은 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

Easy Create(간편 생성)를 활성화한 상태에서 Aurora PostgreSQL DB 클러스터를 생성하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 콘솔의 오른쪽 위 모서리에서 DB 인스턴스를 생성하려는 AWS 리전을 선택합니다.
Aurora를 모든 AWS 리전에서 사용할 수 있는 것은 아닙니다. Aurora를 사용할 수 있는 AWS 리전을 보려면 [리전 사용성 \(p. 3\)](#) 단원을 참조하십시오.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. 데이터베이스 생성을 선택하고 Easy Create(간편 생성)가 선택되어 있는지 확인합니다.

Create database



5. Engine type(엔진 유형)에서 Amazon Aurora을 선택합니다.
6. Edition(에디션)에서 PostgreSQL과 호환되는 Amazon Aurora을 선택합니다.
7. DB instance size(DB 인스턴스 크기)에서 개발/테스트를 선택합니다.
8. DB 클러스터 식별자에 DB 클러스터의 이름을 입력하거나 기본 이름을 그대로 유지합니다.
9. 마스터 사용자 이름에 마스터 사용자의 이름을 입력하거나 기본 이름을 그대로 유지합니다.

데이터베이스 생성 페이지는 다음 이미지와 비슷해야 합니다.

Create database

Choose a database creation method [Info](#)

Standard Create

You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy Create

Use recommended best-practice configuration options can be changed after the database is created.

Configuration

Engine type [Info](#)

Amazon Aurora



MySQL



MariaDB



PostgreSQL



Oracle

ORACLE®

Microsoft SQL Server



Edition

Amazon Aurora with MySQL 5.6 compatibility

Amazon Aurora with PostgreSQL compatibility

DB instance size

Production

db.r4.2xlarge
8 vCPUs
61 GiB RAM

Dev/Test

db.r4.large
2 vCPUs
15.25 GiB RAM

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 region.

10. DB 클러스터에서 자동 생성된 마스터 암호를 사용하려면 Auto generate a password(암호 자동 생성) 확인란을 선택해야 합니다.

마스터 암호를 입력하려면 Auto generate a password(암호 자동 생성) 확인란의 선택을 해제한 다음, 마스터 암호 및 암호 확인에 동일한 암호를 입력합니다.

11. (선택 사항) View default settings for Easy create(간편 생성 기본 설정 보기)를 엽니다.

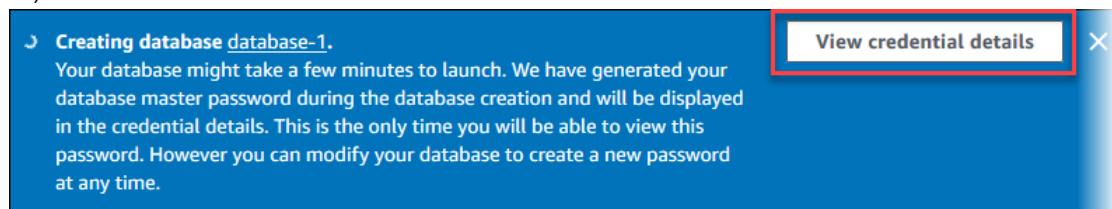
▼ View default settings for Easy create		
Configuration	Value	Editable after database is created
Database Location	Regional	No
Database Features	provisioned	No
Automatic Backups	Enabled	No

Easy Create(간편 생성)가 활성화되었을 때 사용되는 기본 설정을 검토할 수 있습니다. 데이터베이스 생성 도중 하나 이상의 설정을 변경하려면 Standard Create(표준 생성)를 선택하여 설정합니다. Editable after database creation(데이터베이스 생성 후 편집 가능) 열에는 데이터베이스 생성 후 어떤 옵션을 변경할 수 있는지 나와 있습니다. 이 열에 아니요로 표시된 설정을 변경하려면 Standard Create(표준 생성)를 사용합니다. 이 열에 예로 표시된 설정은 Standard Create(표준 생성)를 사용하거나 DB 인스턴스가 생성된 후 인스턴스를 수정하여 해당 설정을 변경할 수 있습니다.

12. 데이터베이스 생성을 선택하십시오.

자동 생성된 암호를 사용하기로 한 경우에는 데이터베이스 페이지에 View credential details(자격 증명 세부 정보 보기) 버튼이 나타납니다.

DB 인스턴스의 마스터 사용자 이름 및 암호를 보려면 View credential details(자격 증명 세부 정보 보기)를 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 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.

13. 데이터베이스에서 새 Aurora PostgreSQL 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.

DB identifier	Role	Engine	Class	Status	CPU	Curri
database-1	Cluster	Aurora PostgreSQL	-	Creating		
database-1-instance-1	Reader	Aurora PostgreSQL	db.r4.large	Creating		

기존 콘솔

Aurora PostgreSQL DB 클러스터를 시작하는 방법

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management 콘솔 오른쪽 상단 모서리에서 DB 클러스터를 생성하려는 AWS 리전을 선택합니다. Aurora를 사용할 수 있는 AWS 리전을 보려면 [리전 가용성 \(p. 3\)](#) 단원을 참조하십시오.
3. 탐색 창에서 데이터베이스를 선택합니다.
탐색 창이 닫혀 있는 경우 왼쪽 상단의 메뉴 아이콘을 선택하여 여십시오.
4. 데이터베이스 생성을 선택하여 엔진 선택 페이지를 여십시오.
5. 엔진 선택 페이지에서 Amazon Aurora를 선택하고 PostgreSQL 호환 에디션을 선택합니다.

RDS > Create database

Select engine

Engine options

Amazon Aurora
Amazon Aurora

MySQL


MariaDB


PostgreSQL


Oracle
ORACLE®

Microsoft SQL Server


Amazon Aurora

Amazon Aurora is a MySQL- and PostgreSQL-compatible enterprise-class database, starting at <\$1/day.

- Up to 5 times the throughput of MySQL and 3 times the throughput of PostgreSQL
- Up to 64TiB of auto-scaling SSD storage
- 6-way replication across three Availability Zones
- Up to 15 Read Replicas with sub-10ms replica lag
- Automatic monitoring and failover in less than 30 seconds

Edition

MySQL 5.6-compatible
Aurora Serverless and Parallel Query capacities are only available with this edition.

MySQL 5.7-compatible

PostgreSQL-compatible

Aurora global database feature is now available.
This feature is now available in our new database creation flow.

[Try it now](#)

Only enable options eligible for RDS Free Usage Tier [Info](#)

[Cancel](#) **Next**

6. [Next]를 선택합니다.
7. [Specify DB details] 페이지에서 다음과 같이 값을 설정합니다.

- DB 인스턴스 클래스: db.r4.large
- DB 인스턴스 식별자: aurora-postgres-db-instance1
- Master username: 영숫자 문자를 사용하여 DB 클러스터의 DB 인스턴스에 로그인할 때 사용할 마스터 사용자 이름을 입력합니다.
- Master password 및 Confirm Password: [Master Password] 상자에 데이터베이스에 로그인할 때 사용할 마스터 사용자 암호를 8~41자의 인쇄 가능한 ASCII 문자(/, " 및 @ 제외)로 입력합니다. 그런 다음 [Confirm Password] 상자에 암호를 다시 한 번 입력합니다.

RDS > Create database

Specify DB details

Instance specifications

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#)

DB engine

Aurora PostgreSQL

Capacity type [Info](#)

Provisioned

You provision and manage the server instance sizes.

Provisioned with Aurora parallel query enabled [Info](#)

You provision and manage the server instance sizes, and Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6)

Serverless [Info](#)

You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.

DB engine version [Info](#)

Aurora PostgreSQL (compatible with PostgreSQL 9.6.9)

DB instance class [Info](#)

db.r4.large — 2 vCPU, 15.25 GiB RAM

Multi-AZ deployment [Info](#)

Create Replica in Different Zone

No

Settings

DB instance identifier [Info](#)

Specify a name that is unique for all DB instances owned by your AWS account in the current region.

aurora-postgres-db-instance1

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance". Must contain from 1 to 63 alphanumeric characters or hyphens (1 to 15 for SQL Server). First character must be a letter. Cannot end with a hyphen or contain two consecutive hyphens.

Master username [Info](#)

Specify an alphanumeric string that defines the login ID for the master user.

myawsuser

Master Username must start with a letter. Must contain 1 to 16 alphanumeric characters.

Master password [Info](#)

Confirm password [Info](#)

Master Password must be at least eight characters long, as in "mypassword". Can be any printable ASCII character except "/", "", or "@".

8. [Next]를 선택하고 [Configure Advanced Settings] 페이지에 다음과 같이 값을 설정합니다.

- Virtual Private Cloud(VPC): 기존 VPC가 있는 경우, VPC 식별자(예: vpc-a464d1c1)를 선택하여 해당 VPC를 Amazon Aurora DB 클러스터에 사용할 수 있습니다. 기존 VPC 사용 방법에 대한 자세한 내용은 [Amazon Aurora에 사용할 VPC의 생성 방법 \(p. 1005\)](#) 단원을 참조하십시오.

기존 VPC가 없다면 [Create a new VPC]를 선택하여 Amazon RDS에서 VPC를 새로 생성하도록 할 수 있습니다. 이 예에서는 [Create a new VPC] 옵션을 사용합니다.

- 서브넷 그룹: 기존 서브넷 그룹이 있으면 해당 서브넷 그룹 식별자(예: gs-subnet-group1)를 선택하여 Amazon Aurora DB 클러스터에 기존 서브넷 그룹을 사용할 수 있습니다.

기존 서브넷 그룹이 없다면 [Create a new subnet group]을 선택하여 Amazon RDS에서 서브넷 그룹을 새로 생성하도록 할 수 있습니다. 이 예에서는 [Create a new subnet group] 옵션을 사용합니다.

- 퍼블릭 액세스 가능성: Yes

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 No로 설정합니다.

- Availability zone: No Preference
- VPC 보안 그룹: 기존 VPC 보안 그룹이 한 개 이상 있을 경우 해당 VPC 보안 그룹 식별자(예: gs-security-group1)를 선택하여 이러한 VPC 보안 그룹을 Amazon Aurora DB 클러스터에 사용할 수 있습니다.

기존 VPC 보안 그룹이 없다면 Create a new Security group(새 보안 그룹 생성)을 선택하여 Amazon RDS에서 VPC 보안 그룹을 새로 생성하도록 할 수 있습니다. 이 예제에서는 Create a new Security group(새 보안 그룹 생성) 옵션을 사용합니다.

- DB 클러스터 식별자: aurora-postgres-db-cluster1
- 데이터베이스 이름: sampledb

Note

이것은 기본 데이터베이스를 생성합니다. 추가 데이터베이스를 생성하려면 DB 클러스터에 연결한 다음 SQL 명령어 CREATE DATABASE를 사용하십시오.

- 데이터베이스 포트: 5432

Note

기업 방화벽 뒤에 있어서 Aurora PostgreSQL 기본 포트인 5432 같은 기본 포트에 액세스하지 못할 수도 있습니다. 이런 경우에는 기업 방화벽이 허용하는 포트 값을 입력합니다. 나중에 Aurora DB 클러스터에 연결할 때도 필요하므로 이 포트 값을 기억해야 합니다.

9. 나머지 값은 기본값으로 유지하고 데이터베이스 생성을 선택하여 DB 클러스터와 기본 인스턴스를 생성합니다.

Aurora PostgreSQL DB 클러스터의 인스턴스에 연결

Amazon RDS가 DB 클러스터를 프로비저닝하여 기본 인스턴스를 생성한 후에는 무엇이든 표준 SQL 클라이언트 애플리케이션을 사용하여 DB 클러스터의 데이터베이스에 연결할 수 있습니다.

Aurora PostgreSQL DB 클러스터의 데이터베이스에 연결하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택하고 DB 클러스터 이름을 선택하여 DB 클러스터 세부 정보를 표시합니다. Connectivity & security(연결 및 보안) 탭에서 쓰기 엔드포인트의 엔드포인트 이름에 대한 값을 복사합니다. 또한 엔드포인트의 포트 번호를 적어둡니다.

Endpoint name	Status	Type	Port
database-1.cluster-ro-... .us-west-1.rds.amazonaws.com	Available	Reader	5432
database-1.cluster-... .us-west-1.rds.amazonaws.com	Available	Writer	5432

3. 엔드포인트 및 포트를 사용하여 DB 클러스터에 연결하는 방법은 [Amazon Aurora PostgreSQL DB 클러스터 연결 \(p. 166\)](#) 단원을 참조하십시오.

샘플 DB 클러스터, DB 서브넷 그룹 및 VPC 삭제

생성한 샘플 DB 클러스터에 연결되면 이제 DB 클러스터, DB 서브넷 그룹 및 VPC(생성한 경우)를 삭제할 수 있습니다.

DB 클러스터를 삭제하는 방법

1. AWS Management 콘솔에 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.

DB 서브넷 그룹을 삭제하는 방법

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 서브넷 그룹을 선택한 다음, DB 서브넷 그룹을 선택합니다.
3. 삭제를 선택합니다.
4. 삭제를 선택합니다.

VPC를 삭제하는 방법

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
2. [Your VPCs]를 선택한 후 이 절차에서 생성한 VPC를 선택합니다.
3. 작업에서 Delete VPC(VPC 삭제)를 선택합니다.
4. 삭제를 선택합니다.

Amazon Aurora DB 클러스터 구성

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 \(p. 95\)](#)
- [사용 Amazon Aurora Serverless \(p. 111\)](#)
- [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#)
- [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#)
- [Amazon Aurora DB 클러스터로 데이터 마이그레이션 \(p. 188\)](#)

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 나타내는 클러스터 볼륨으로 구성됩니다. 기본적으로 DB 클러스터에는 기본 라이터 DB 인스턴스와 옵션으로 최대 15개의 Aurora 복제본(리더 DB 인스턴스)이 포함됩니다. Aurora DB 클러스터에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 \(p. 2\)](#) 단원을 참조하십시오.

다음 주제에서는 Aurora DB 클러스터를 생성하는 방법을 확인할 수 있습니다. 시작하려면 먼저 [DB 클러스터 사전 요구사항 \(p. 95\)](#) 단원을 참조하십시오.

Aurora DB 클러스터에 연결하는 간단한 지침은 [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#) 단원을 참조하십시오.

DB 클러스터 사전 요구사항

Important

Aurora DB 클러스터를 생성할 수 있으려면 먼저 [Amazon Aurora 환경 설정 \(p. 69\)](#) 단원의 작업들을 완료해야 합니다.

다음은 DB 클러스터를 생성할 때 필요한 사전 요구사항입니다.

VPC

Amazon Aurora DB 클러스터는 가용 영역이 최소 2개 이상인 AWS 리전의 Amazon VPC 서비스 기반 가상 사설 클라우드(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.

AWS Management 콘솔을 사용하여 Aurora DB 클러스터를 생성하는 경우에는 Amazon RDS에서 VPC를 자동으로 생성할 수 있습니다. 또는 Aurora DB 클러스터에서 기존 VPC를 사용하거나 새 VPC를 생성

할 수 있습니다. Amazon Aurora DB 클러스터에서 VPC를 사용하려면 2개 이상의 가용 영역마다 VPC에 서브넷이 1개 이상 있어야 합니다. 자세한 내용은 [Amazon Aurora에 사용할 VPC의 생성 방법 \(p. 1005\)](#) 단원을 참조하십시오. VPC에 대한 자세한 내용은 다음([Amazon Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#))을 참조하십시오.

Note

ClassicLink를 사용하여 VPC 및 Amazon Aurora DB 클러스터에 없는 EC2 인스턴스와 통신할 수 있습니다. 자세한 내용은 [VPC에 있지 않은 EC2 인스턴스가 VPC에 있는 DB 인스턴스에 액세스 \(p. 1014\)](#) 단원을 참조하십시오.

기본 VPC가 없거나 VPC를 생성하지 않았다면 콘솔을 사용해 Aurora 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 [Amazon Aurora에 사용할 VPC의 생성 방법 \(p. 1005\)](#) 단원을 참조하십시오.
- Aurora DB 클러스터에 대한 연결 권한을 부여할 수 있도록 VPC 보안 그룹을 지정합니다. 자세한 내용은 [VPC에서 DB 인스턴스를 사용한 작업 \(p. 1016\)](#) 단원을 참조하십시오.
- Aurora DB 클러스터에서 VPC의 서브넷 2개 이상을 사용할 수 있도록 RDS DB 서브넷 그룹을 지정합니다. 자세한 내용은 [DB 서브넷 그룹을 사용한 작업 \(p. 1016\)](#) 단원을 참조하십시오.

기타 사전 요구사항

AWS Identity and Access Management(IAM) 자격 증명을 사용하여 AWS에 연결할 경우 Amazon RDS 작업을 수행하는 데 필요한 사용 권한을 부여할 수 있는 IAM 정책이 IAM 계정에 필요합니다. 자세한 내용은 [Amazon Aurora의 Identity and Access Management\(IAM\) \(p. 960\)](#) 단원을 참조하십시오.

IAM 계정을 사용해 Amazon RDS 콘솔에 액세스하려면, 먼저 IAM 계정으로 AWS Management 콘솔에 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 필요로 하는 파라미터 설정으로 지정해야 합니다. DB 클러스터 파라미터 그룹 또는 DB 파라미터 그룹의 생성 또는 변경에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

DB 클러스터에 지정할 TCP/IP 포트 번호를 결정해야 합니다. 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.

DB 클러스터 생성

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 Aurora DB 클러스터를 생성할 수 있습니다.

Note

콘솔을 사용 중인 경우에는 새 콘솔 인터페이스를 데이터베이스 생성에 사용할 수 있습니다. 사용 중인 콘솔에 따라 New Console(새 콘솔) 또는 Original Console(기존 콘솔) 지침을 선택합니다. New Console(새 콘솔) 지침이 기본적으로 열립니다.

새운 콘솔

Easy create(간편 생성)를 활성화 또는 비활성화하여 AWS Management 콘솔에서 MySQL을 실행하는 DB 인스턴스를 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 예제에서는 Standard Create(표준 생성)가 활성화되고 Easy Create(간편 생성)는 활성화되지 않습니다. Easy create(간편 생성)를 활성화한 상태에서 Aurora MySQL DB 클러스터를 생성하는 방법은 [Amazon Aurora 시작하기 \(p. 74\)](#) 단원을 참조하십시오.

콘솔을 사용하여 Aurora DB 클러스터를 생성하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management 콘솔의 오른쪽 상단 모서리에서 DB 인스턴스를 생성하려는 AWS 리전을 선택합니다.

Aurora를 모든 AWS 리전에서 사용할 수 있는 것은 아닙니다. Aurora를 사용할 수 있는 AWS 리전을 보려면 [리전 가용성 \(p. 3\)](#) 단원을 참조하십시오.

3. 탐색 창에서 데이터베이스를 선택합니다.
4. 데이터베이스 생성을 선택하십시오.
5. Choose a database creation method(데이터베이스 생성 방법 선택)에서 Standard Create(표준 생성)를 선택합니다.
6. Engine options(엔진 옵션)에서 Amazon Aurora를 선택합니다.

Create database

Choose a database creation method Info

Standard Create

You set all of the configuration options, including ones for availability, security, backups, and maintenance.

Easy Create

Use recommended best-practice configuration options can be changed after the database is created.

Engine options

Engine type Info

Amazon Aurora



MySQL



MariaDB



PostgreSQL



Oracle

ORACLE®

Microsoft SQL Server



Edition

Amazon Aurora with MySQL compatibility

Amazon Aurora with PostgreSQL compatibility

Version Info

7. Edition(에디션)에서 다음 중 하나를 선택하십시오.

- MySQL과 호환되는 Amazon Aurora

- PostgreSQL과 호환되는 Amazon Aurora
8. MySQL과 호환되는 Amazon Aurora를 선택한 경우에는 Database features(데이터베이스 기능)에서 다음 중 하나를 선택합니다.
- 라이터 한 개, 리더 여러 개

자세한 내용은 [Amazon Aurora DB 클러스터 \(p. 2\)](#) 단원을 참조하십시오.

- 서비스

자세한 내용은 [사용 Amazon Aurora Serverless \(p. 111\)](#) 단원을 참조하십시오.

9. 템플릿에서 사용 사례에 맞는 템플릿을 선택합니다.

10. 마스터 암호를 입력하려면 다음과 같이 하십시오.

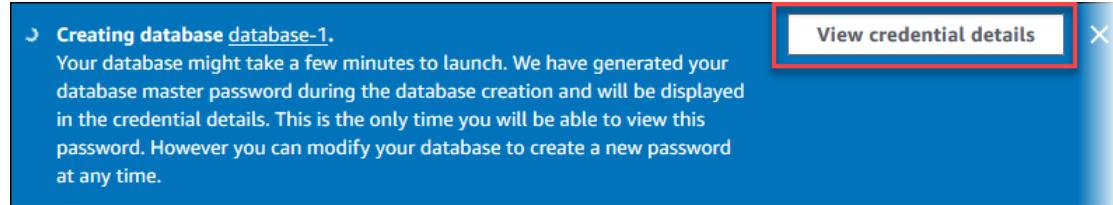
- a. 설정 섹션에서 Credential Settings(자격 증명 설정)를 엽니다.
- b. Auto generate a password(암호 자동 생성) 확인란의 선택을 취소합니다.
- c. (선택 사항) 마스터 사용자 이름 값을 변경하고 마스터 암호 및 암호 확인에 동일한 암호를 입력합니다.

기본적으로 새 DB 인스턴스는 마스터 사용자를 위해 자동 생성된 암호를 사용합니다.

11. 나머지 섹션에서 DB 클러스터 설정을 지정합니다. 각 설정에 대한 자세한 내용은 [Aurora DB 클러스터 설정 \(p. 106\)](#) 단원을 참조하십시오.
12. 데이터베이스 생성을 선택하십시오.

자동 생성된 암호를 사용하기로 결정한 경우에는 데이터베이스 페이지에 View credential details(자격 증명 세부 정보 보기) 버튼이 나타납니다.

DB 클러스터의 마스터 사용자 이름 및 암호를 보려면 View credential details(자격 증명 세부 정보 보기)를 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 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.

13. 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.

Amazon Aurora Aurora 사용 설명서 DB 클러스터 생성

The screenshot shows the 'Databases' section of the Amazon RDS console. A cluster named 'database-1' contains two instances: 'database-1-instance-1' (Role: Reader, Engine: Aurora MySQL, Class: db.r5.large) and 'database-1-instance-2' (Role: Writer, Engine: Aurora MySQL, Class: db.r5.large). Both instances are currently in a 'Creating' state, indicated by a circular progress bar icon next to the status column. A red circle highlights the 'Status' column header and the 'Creating' status of the second instance.

상태가 [available]로 변경되면 DB 클러스터의 기본 인스턴스에 연결할 수 있습니다. DB 인스턴스 클레스와 할당된 저장소에 따라 새 인스턴스를 사용할 수 있을 때까지 몇 분 정도 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 \(p. 338\)](#) 단원을 참조하십시오.

This screenshot displays the detailed configuration for the 'database-1' DB cluster. The 'Connectivity & security' tab is active. It lists two endpoints: 'database-1.cluster-ro-...' (Type: Reader, Port: 3306) and 'database-1.cluster-...' (Type: Writer, Port: 3306). The 'Writer' endpoint is highlighted with a red circle. Below the endpoints, there are tabs for 'Monitoring', 'Logs & events', 'Configuration', 'Maintenance & backups', and 'Tags'.

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.

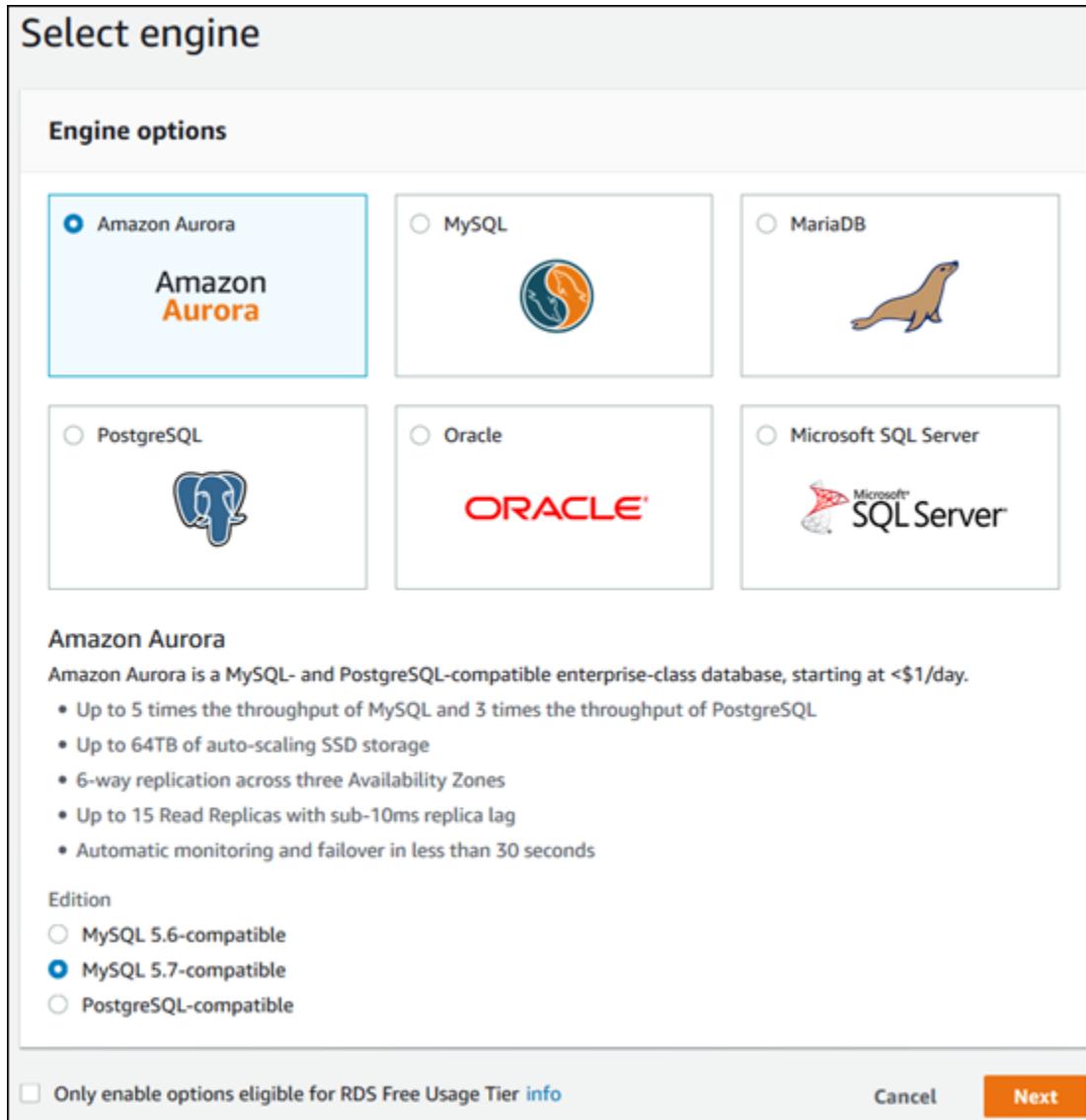
기존 콘솔

AWS Management 콘솔을 사용하여 Aurora DB 클러스터를 생성하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management 콘솔의 오른쪽 위 모서리에서 Aurora DB 클러스터를 만들 AWS 리전을 선택합니다.
3. 탐색 창에서 데이터베이스를 선택합니다.

탐색 창이 닫혀 있는 경우 왼쪽 상단의 메뉴 아이콘을 선택하여 여십시오.

4. 데이터베이스 생성을 선택하여 엔진 선택 페이지를 여십시오.
5. 엔진 선택 페이지에서 Aurora의 에디션을 선택합니다. MySQL 5.6 호환, MySQL 5.7 호환 또는 PostgreSQL 호환을 선택하십시오.



6. [Next]를 선택합니다.
7. DB 세부 정보 지정 페이지에서 DB 인스턴스 정보를 지정합니다. 각 설정에 대한 자세한 내용은 [Aurora DB 클러스터 설정 \(p. 106\)](#) 단원을 참조하십시오.

일반적인 [Specify DB details] 페이지는 다음과 같습니다.

Specify DB details

Instance specifications

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#).

DB engine
Aurora - compatible with MySQL 5.7.12

DB instance class [info](#)
db.r4.large — 2 vCPU, 15.25 GiB RAM

Multi-AZ deployment [info](#)
 Create Replica in Different Zone
 No

Settings

DB instance identifier [info](#)
Specify a name that is unique for all DB instances owned by your AWS account in the current region.
gs-db-instance1

DB instance identifier is case insensitive, but stored as all lower-case, as in "mydbinstance".

Master username [info](#)
Specify an alphanumeric string that defines the login ID for the master user.
myawsuser

Master Username must start with a letter.

Master password [info](#) Confirm password [info](#)
***** *****

Master Password must be at least eight characters long, as in "mypassword".

[Cancel](#) [Previous](#) [Next](#)

8. 마스터 암호를 한 번 더 입력한 후 [Next]를 선택합니다.
9. 고급 설정 구성 페이지에서 Aurora DB 클러스터 설정을 추가로 사용자 지정할 수 있습니다. 각 설정에 대한 자세한 내용은 [Aurora DB 클러스터 설정 \(p. 106\)](#) 단원을 참조하십시오.
10. Create database(데이터베이스 생성)을 선택하여 Aurora DB 클러스터를 생성한 후 닫기를 선택합니다.

Amazon RDS 콘솔의 DB 클러스터 목록에 새로운 DB 클러스터가 나타납니다. DB 클러스터를 만들고 사용할 준비가 될 때까지 DB 클러스터의 상태는 생성 중입니다. 상태가 '사용 가능'으로 변경되면 DB 클

러스터의 라이터 인스턴스에 연결할 수 있습니다. DB 클러스터 클래스와 할당된 저장소에 따라 새 클러스터를 사용할 수 있을 때까지 몇 분 정도 걸릴 수 있습니다.

새로 생성된 클러스터를 보려면 Amazon RDS 콘솔의 탐색 창에서 Databases(데이터베이스)를 선택하고 DB 클러스터를 선택하여 DB 클러스터 세부 정보를 표시하십시오. 자세한 내용은 [Amazon Aurora DB 클러스터 보기 \(p. 338\)](#) 단원을 참조하십시오.

The screenshot shows the AWS RDS console interface for managing a DB cluster named 'gs-db-cluster1'. The main area displays the DB identifier hierarchy:

DB identifier	Role	Engine
gs-db-cluster1	Regional	Aurora MySQL
gs-db-instance1	Writer	Aurora MySQL
gs-db-instance1-us-east-2b	Reader	Aurora MySQL

Below the DB identifier section, there are tabs for Connectivity & security, Monitoring, Logs & events, Configuration, and Maintenance. The Connectivity & security tab is selected. Under this tab, the 'Endpoints (2)' section is shown, listing two endpoint entries:

Endpoint name	Status	
gs-db-cluster1.rds.amazonaws.com	Available	(circled in red)
gs-db-cluster1.rds.amazonaws.com	Available	

클러스터의 포트와 엔드포인트를 기록합니다. 쓰기 또는 읽기 작업을 수행하는 모든 애플리케이션에 대해서는 JDBC 및 ODBC 연결 문자열에서 라이터 DB 클러스터의 엔드포인트와 포트를 사용하십시오.

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 요구사항 \(p. 95\)](#) 단원을 참조하십시오.

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 DB 클러스터 또는 DB 인스턴스를 생성할 때는 DB 클러스터 또는 DB 인스턴스 MySQL 호환성을 기반으로 --engine 옵션 값에 올바른 값을 지정하는지 확인하십시오.

- Aurora MySQL 5.7 DB 클러스터 또는 DB 인스턴스를 생성할 때는 --engine 옵션에 aurora-mysql을 지정해야 합니다.
- Aurora MySQL 5.6 DB 클러스터 또는 DB 인스턴스를 생성할 때는 --engine 옵션에 aurora을 지정해야 합니다.

다음 단계를 완료합니다.

1. 새 DB 클러스터에 대한 DB 서브넷 그룹과 VPC 보안 그룹 ID를 확인한 다음 `create-db-cluster` AWS CLI 명령을 호출하여 Aurora MySQL DB 클러스터를 생성하십시오.

예를 들어, 다음 명령을 사용하면 이름이 sample-cluster인 새 MySQL 5.7-호환 DB 클러스터가 생성됩니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-mysql \
  \
  --engine-version 5.7.12 --master-username user-name --master-user-
  password 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 5.7.12 --master-username user-name --master-user-password password
^
  \
  --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2
```

다음 명령을 사용하면 이름이 sample-cluster인 새 MySQL 5.6-호환 DB 클러스터가 생성됩니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora \
  \
  --engine-version 5.6.10a --master-username user-name --master-user-
  password 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 ^
```

```
--engine-version 5.6.10a --master-username user-name --master-user-  
password 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.

[create-db-instance](#) AWS CLI 명령을 호출하여 DB 클러스터를 위한 기본 인스턴스를 생성하십시오. DB 클러스터의 이름을 `--db-cluster-identifier` 옵션 값으로 포함하십시오.

예를 들어, 다음 명령을 사용하면 이름이 `sample-instance`인 새 MySQL 5.7-호환 DB 인스턴스가 생성됩니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance \  
    --db-cluster-identifier sample-cluster --engine aurora-mysql --db-instance-class  
    db.r4.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.r4.large
```

다음 명령을 사용하면 이름이 `sample-instance`인 새 MySQL 5.6-호환 DB 인스턴스가 생성됩니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance \  
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class  
    db.r4.large
```

Windows의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^  
    --db-cluster-identifier sample-cluster --engine aurora --db-instance-class  
    db.r4.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 생성하십시오.

예를 들어, 다음 명령을 사용하면 이름이 `sample-cluster`인 새 DB 클러스터가 생성됩니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-  
    postgresql \  
        --master-username user-name --master-user-password 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 ^  
    --master-username user-name --master-user-password 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.

[create-db-instance](#) AWS CLI 명령을 호출하여 DB 클러스터를 위한 기본 인스턴스를 생성하십시오. DB 클러스터의 이름을 `--db-cluster-identifier` 옵션 값으로 포함하십시오.

Linux, OS X, Unix의 경우:

```
aws rds create-db-instance --db-instance-identifier sample-instance ^  
    --db-cluster-identifier sample-cluster --engine aurora-postgresql --db-instance-class db.r4.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.r4.large
```

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 요구사항 \(p. 95\)](#) 단원을 참조하십시오.

새 DB 클러스터에 대한 DB 서브넷 그룹과 VPC 보안 그룹 ID를 확인한 다음 [CreateDBInstance](#) 작업을 호출하여 DB 클러스터를 생성하십시오.

Aurora MySQL DB 클러스터 또는 DB 인스턴스를 생성할 때는 DB 클러스터 또는 DB 인스턴스 MySQL 호환성을 기반으로 Engine 파라미터 값에 올바른 값을 지정하는지 확인하십시오.

- Aurora MySQL 5.7 DB 클러스터 또는 DB 인스턴스를 생성할 때는 Engine 파라미터에 `aurora-mysql`을 지정해야 합니다.
- Aurora MySQL 5.6 DB 클러스터 또는 DB 인스턴스를 생성할 때는 Engine 파라미터에 `aurora`을 지정해야 합니다.

Aurora PostgreSQL DB 클러스터 또는 DB 인스턴스를 생성할 때는 Engine 파라미터에 `aurora-postgresql`을 지정하십시오.

Aurora DB 클러스터 설정

다음 표에는 Aurora DB 인스턴스를 생성할 때 선택하는 설정에 대한 세부 정보가 나와 있습니다.

옵션	수행할 작업
[Availability zone]	특정 가용 영역의 지정 여부를 결정합니다. 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 (p. 3) 단원을 참조하십시오.

옵션	수행할 작업
Auto minor version upgrade	<p>DB 엔진에 대한 기본 마이너 버전 업그레이드가 있을 때 Aurora DB 클러스터에서 이를 자동으로 수신하도록 하려면 Enable auto minor version upgrade(마이너 버전 자동 업그레이드 활성화)를 선택합니다.</p> <p>마이너 버전 자동 업그레이드 설정은 Aurora PostgreSQL DB 클러스터에만 적용됩니다.</p> <p>Aurora PostgreSQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora PostgreSQL 데이터베이스 엔진 업데이트 (p. 905) 단원을 참조하십시오.</p> <p>Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL 데이터베이스 엔진 업데이트 (p. 686) 단원을 참조하십시오.</p>
역추적	<p>Aurora MySQL에만 적용됩니다. 역추적을 활성화하려면 Enable Backtrack(역추적 활성화)을 선택하고 역추적을 비활성화하려면 Disable Backtrack(역추적 비활성화)을 선택합니다.</p> <p>역추적을 이용하면 새 DB 클러스터를 만들지 않고 특정 시점으로 DB 클러스터를 되감을 수 있습니다. 기본적으로는 비활성화되어 있습니다. 역추적을 활성화할 경우 DB 클러스터를 역추적 할 수 있는 기간(대상 역추적 기간)도 함께 지정하십시오. 자세한 내용은 Aurora DB 클러스터 역추적 (p. 509) 단원을 참조하십시오.</p>
스냅샷으로 태그 복사	<p>스냅샷을 생성할 때 DB 인스턴스 태그를 DB 스냅샷에 복사하려면 이 옵션을 선택합니다.</p> <p>자세한 내용은 Amazon RDS 리소스에 태그 지정 (p. 315) 단원을 참조하십시오.</p>
Database authentication(데이터베이스 인증)	<p>귀하가 사용하고 싶은 데이터베이스 인증 옵션입니다.</p> <p>데이터베이스 암호로만 데이터베이스 사용자를 인증할 수 있도록 Password authentication(암호 인증)을 선택합니다.</p> <p>IAM 사용자 및 역할을 통해 데이터베이스 암호 및 사용자 자격증명으로 데이터베이스 사용자를 인증할 수 있도록 Password and IAM DB authentication(암호 및 IAM DB 인증)을 선택합니다. 자세한 정보는 을 위한 IAM 데이터베이스 인증 (p. 976) 단원을 참조하십시오.</p>
데이터베이스 포트	<p>애플리케이션과 유ти리티가 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.</p>

옵션	수행할 작업
[DB cluster identifier]	<p>선택한 AWS 리전에 속한 계정에 고유한 DB 클러스터 이름을 입력합니다. 이 식별자는 DB 클러스터에 대한 클러스터 엔드포인트 주소로 사용됩니다. 클러스터 엔드포인트에 대한 자세한 정보는 Amazon Aurora 연결 관리 (p. 9) 단원을 참조하십시오.</p> <p>DB 클러스터 식별자는 다음과 같은 제약 조건이 있습니다.</p> <ul style="list-style-type: none"> • 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.
DB 클러스터 파라미터 그룹	<p>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 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.</p>
DB 엔진 버전	<p>프로비저닝된 용량 유형에만 적용됩니다. DB 엔진의 버전 번호를 선택합니다.</p>
DB 인스턴스 클래스	<p>프로비저닝된 용량 유형에만 적용됩니다. DB 클러스터의 각 인스턴스 처리 및 메모리 요건을 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 (p. 30) 단원을 참조하십시오.</p>
DB 인스턴스 식별자	<p>DB 클러스터의 기본 인스턴스 이름을 입력합니다. 이 식별자는 DB 클러스터의 기본 인스턴스에 대한 엔드포인트 주소로 사용됩니다.</p> <p>DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다.</p> <ul style="list-style-type: none"> • 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. • AWS 리전별로 AWS 계정 하나당 모든 DB 인스턴스는 고유해야 합니다.
DB 파라미터 그룹	<p>파라미터 그룹을 선택합니다. Aurora에서 제공되는 기본 파라미터 그룹을 사용하거나 자체 파라미터 그룹을 생성할 수 있습니다. 파라미터 그룹에 대한 자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.</p>
삭제 방지 활성화	<p>DB 클러스터가 삭제되는 것을 방지하려면, 삭제 방지 활성화를 선택합니다. 콘솔을 사용하여 프로덕션 DB 클러스터를 생성할 경우 기본값으로 삭제 방지가 활성화됩니다.</p>
Enable encryption	<p>이 DB 클러스터의 저장 시 암호화를 활성화하려면 Enable encryption를 선택합니다. 자세한 내용은 Amazon Aurora 리소스 암호화 (p. 945) 단원을 참조하십시오.</p>

옵션	수행할 작업
Enable Enhanced Monitoring	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 단원을 참조하십시오.
Performance Insights 활성화	Enable Performance Insights를 선택하여 Amazon RDS Performance Insights를 활성화합니다. 자세한 내용은 Amazon RDS 성능 개선 도우미 사용 (p. 365) 단원을 참조하십시오.
Failover priority(장애 조치 우선 순위)	인스턴스의 장애 조치 우선 순위를 선택합니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 기본 인스턴스 장애로부터 복원할 때 이 우선 순위에 따라 승격할 Aurora Replicas 순서가 결정됩니다. 자세한 내용은 Aurora DB 클러스터의 내결합성 (p. 268) 단원을 참조하십시오.
Granularity	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
초기 데이터베이스 이름	<p>기본 데이터베이스의 이름을 입력합니다. 이름을 입력하지 않으면 Amazon RDS가 DB 클러스터에서 생성하려고 하는 데이터베이스를 생성하지 않습니다.</p> <p>Aurora MySQL의 경우, 기본 데이터베이스 이름에는 다음과 같은 제약 조건이 적용됩니다.</p> <ul style="list-style-type: none"> 1–64자의 영숫자로 구성되어야 합니다. 데이터베이스 엔진에서 예약한 단어는 사용할 수 없습니다. <p>Aurora PostgreSQL의 경우, 기본 데이터베이스 이름에는 다음과 같은 제약 조건이 적용됩니다.</p> <ul style="list-style-type: none"> 1–63자의 영숫자로 구성되어야 합니다. 글자나 밑줄로 시작해야 합니다. 이후 문자는 글자, 밑줄 또는 숫자(0–9)가 될 수 있습니다. 데이터베이스 엔진에서 예약한 단어는 사용할 수 없습니다. <p>추가 데이터베이스를 생성하려면, DB 클러스터에 연결한 다음 SQL 명령어 CREATE DATABASE를 사용하십시오. DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora DB 클러스터 연결 (p. 162) 단원을 참조하십시오.</p>
로그 내보내기	생성할 MySQL 또는 PostgreSQL 데이터베이스 로그 파일의 유형을 선택합니다. 자세한 내용은 MySQL 데이터베이스 로그 파일 (p. 452) 및 PostgreSQL 데이터베이스 로그 파일 (p. 456) 단원을 참조하십시오.
유지 관리 기간	기간 선택을 선택하고 시스템 유지 관리를 실행할 수 있는 주 단위 기간을 지정합니다. 또는 Amazon RDS가 임의로 기간을 지정하도록 하려면 기본 설정 없음을 선택합니다.

옵션	수행할 작업
마스터 키	[Encryption]을 [Enable encryption]으로 설정한 경우에만 사용 할 수 있습니다. 이 DB 클러스터를 암호화하는 데 사용할 마스터 키를 선택합니다. 자세한 내용은 Amazon Aurora 리소스 암호화 (p. 945) 단원을 참조하십시오.
옵션 그룹	Aurora은 기본값 옵션 그룹을 포함합니다.
Master password	DB 클러스터에 로그인할 암호를 입력합니다. <ul style="list-style-type: none"> Aurora MySQL의 경우, 암호는 8–41자의 인쇄 가능한 ASCII 문자로 구성되어야 합니다. Aurora PostgreSQL의 경우, 암호는 8–128자의 인쇄 가능한 ASCII 문자로 구성되어야 합니다. / , " , @ 또는 공백을 포함할 수 없습니다.
Master username	DB 클러스터에 로그인할 때 마스터 사용자 이름으로 사용할 이름을 입력합니다. <ul style="list-style-type: none"> 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.
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-role라는 이름의 역할을 생성하도록 기본값을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 단원을 참조하십시오.
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 (p. 3) 단원을 참조하십시오.
옵션 그룹	Aurora은 기본값 옵션 그룹을 포함합니다.
Performance Insights	MySQL 5.6에는 적용되지 않습니다. Amazon RDS Performance Insights를 사용해 Amazon Aurora DB 클러스터 로드를 모니터링하려면 Performance Insights 활성화를 선택하십시오. 성능 개선 도우미(Performance Insights)에 대한 자세한 내용은 Amazon RDS 성능 개선 도우미 사용 (p. 365) 단원을 참조하십시오.
[Publicly accessible]	DB 클러스터에 퍼블릭 IP 주소를 할당하려면 예를 선택하고, 그렇지 않으면 아니요를 선택합니다. DB 클러스터의 인스턴스는 퍼블릭과 프라이빗 DB 인스턴스를 모두 혼합하여 사용할 수 있습니다. 모든 사용자의 액세스에서 인스턴스를 습기는 방법에 대한 자세한 내용은 VPC에 있는 DB 인스턴스를 인터넷에서 습기기 (p. 1017) 단원을 참조하십시오.

옵션	수행할 작업
보존 기간	Aurora가 데이터베이스 백업 사본을 보존하는 기간을 1~35일로 선택합니다. 백업 사본은 데이터베이스를 마지막 특정 시점으로 복구(PITR)하는 데 사용됩니다.
Subnet Group	DB 클러스터에 사용할 DB 서브넷 그룹을 선택합니다. 자세한 내용은 DB 클러스터 사전 요구사항 (p. 95) 단원을 참조하십시오.
Amazon CloudWatch Logs에 게시할 로그 유형을 선택합니다.	Aurora MySQL에만 적용됩니다. 로그 내보내기 섹션에서 Amazon CloudWatch Logs에 게시하기 시작할 로그를 선택합니다. CloudWatch Logs로의 게시에 대한 자세한 내용은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 (p. 639) 단원을 참조하십시오.
Virtual Private Cloud(VPC)	DB 클러스터를 호스팅할 VPC를 선택합니다. Amazon RDS에서 VPC를 생성하도록 하려면 새 VPC 생성을 선택합니다. 자세한 내용은 DB 클러스터 사전 요구사항 (p. 95) 단원을 참조하십시오.
VPC 설정 그룹	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 요구사항 (p. 95) 단원을 참조하십시오.

사용 Amazon Aurora Serverless

Amazon Aurora Serverless는 Amazon Aurora에 대한 온디맨드 방식의 Auto Scaling 구성입니다. Aurora Serverless DB 클러스터는 애플리케이션의 필요에 따라 자동으로 시작 및 종료되고 컴퓨팅 용량이 확장 또는 축소되는 DB 클러스터입니다. Aurora Serverless는 부정기적이거나 간헐적이거나 예측할 수 없는 워크로드에 대해 비교적 간단하고 비용 효율적인 옵션을 제공합니다. 자동으로 시작하고 애플리케이션의 사용량에 맞춰 컴퓨팅 용량을 확장하고, 사용하지 않는 경우 종료되기 때문에 이러한 옵션을 제공할 수 있습니다.

Note

Aurora에 대한 비 서버리스 DB 클러스터를 프로비저닝된 DB 클러스터라고 합니다. Aurora Serverless 클러스터와 프로비저닝된 클러스터는 모두 동일한 종류의 분산형의 고용량 및 고가용성 스토리지 볼륨을 가지고 있습니다.

주제

- [Aurora Serverless의 장점 \(p. 112\)](#)
- [Aurora Serverless 사용 사례 \(p. 112\)](#)
- [Aurora Serverless의 제한 사항 \(p. 113\)](#)
- [Aurora Serverless에서 TLS/SSL 사용 \(p. 114\)](#)
- [Aurora Serverless 작동 방식 \(p. 114\)](#)
- [Aurora Serverless DB 클러스터 생성 \(p. 120\)](#)

- [Aurora Serverless DB 클러스터 복원 \(p. 125\)](#)
- [Aurora Serverless DB 클러스터 수정 \(p. 127\)](#)
- [Aurora Serverless DB 클러스터의 용량 설정 \(p. 129\)](#)
- [Aurora Serverless DB 클러스터 보기 \(p. 131\)](#)
- [Aurora Serverless에 데이터 API 사용 \(p. 134\)](#)
- [AWS CloudTrail을 사용하여 데이터 API 호출 로깅 \(p. 157\)](#)
- [Aurora Serverless에 쿼리 편집기 사용 \(p. 158\)](#)

Aurora Serverless의 장점

Aurora Serverless의 장점은 다음과 같습니다.

더 간단해짐

Aurora Serverless는 DB 인스턴스 및 용량 관리의 복잡성을 크게 줄여 줍니다.

확장 가능

Aurora Serverless는 클라이언트 연결을 종단하지 않고 필요에 따라 컴퓨팅 및 메모리 용량을 원활하게 확장합니다.

비용 효율성

Aurora Serverless를 사용하는 경우 사용한 데이터베이스 리소스에 대해서만 초 단위로 요금을 지불합니다.

고가용성 스토리지

Aurora Serverless는 6방향 복제가 가능하고 Aurora와 동일한 내결함성 분산 스토리지 시스템을 사용해 데이터 손실을 방지합니다.

Aurora Serverless 사용 사례

Aurora Serverless는 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 Serverless를 사용하는 경우 사용한 데이터베이스 리소스에 대해서만 초 단위로 요금을 지불합니다.

신규 애플리케이션

새 애플리케이션을 배포하는데 필요한 인스턴스 크기를 잘 모릅니다. Aurora Serverless를 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 Serverless를 사용하면 피크 또는 평균 용량을 더 이상 프로비저닝하지 않아도 됩니다.

예측 불가능한 워크로드

종일 데이터베이스를 사용하는 워크로드를 실행 중인데, 활동의 피크를 예측하기가 어렵습니다. 비가 내리기 시작하면 활동이 급증하는 트래픽 사이트를 예로 들 수 있습니다. Aurora Serverless를 사용하면 애

플리케이션의 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 Serverless를 사용하면 데이터베이스가 사용 중이 아닌 경우 자동으로 종료됩니다.

다중 테넌트 애플리케이션

Aurora Serverless를 사용하면 플랫에서 각 애플리케이션에 대한 데이터베이스 용량을 개별적으로 관리할 필요가 없습니다. Aurora Serverless는 사용자를 대신하여 개별 데이터베이스 용량을 관리해 줍니다.

Aurora Serverless의 제한 사항

Aurora Serverless에는 다음과 같은 제한 사항이 적용됩니다.

- Aurora Serverless는 다음과 같은 경우에만 사용할 수 있습니다.
 - MySQL 버전 5.6과 호환되는 Aurora
 - PostgreSQL 버전 10.7과 호환되는 Aurora
- 연결용 포트 번호는 다음과 같아야 합니다.
 - 5432(Aurora PostgreSQL일 때)
 - 5432(Aurora PostgreSQL일 때)
- Aurora Serverless DB 클러스터에는 퍼블릭 IP 주소를 부여할 수 없습니다. Amazon VPC 서비스를 기반으로 하는 가상 사설 클라우드(VPC) 내에서만 Aurora Serverless DB 클러스터에 액세스할 수 있습니다.
- 각 Aurora Serverless DB 클러스터에는 두 개의 AWS PrivateLink 엔드포인트가 필요합니다. VPC 내 AWS PrivateLink 엔드포인트의 제한에 도달하면 해당 VPC에서 더 이상 Aurora Serverless 클러스터를 생성할 수 없습니다. VPC 내 엔드포인트 제한을 확인하고 변경하는 방법에 대한 자세한 내용은 [Amazon VPC 제한](#)을 참조하십시오.
- Aurora Serverless에서 사용하는 DB 서브넷 그룹은 동일한 가용 영역에 하나 이상의 서브넷을 보유할 수 없습니다.
- Aurora Serverless DB 클러스터에서 사용하는 서브넷 그룹에 대한 변경 사항은 클러스터에 적용되지 않습니다.
- Aurora Serverless DB 클러스터에 대한 연결이 하루 이상 열려 있으면 자동으로 닫힙니다.
- Aurora Serverless는 다음과 같은 기능을 지원하지 않습니다.
 - [Amazon S3 버킷에서 데이터 로드](#) (p. 620)
 - [데이터를 Amazon S3 버킷에 저장](#) (p. 627)
 - [Aurora MySQL 네이티브 함수로 AWS Lambda 함수 호출](#) (p. 634)
 - [Aurora 복제본](#) (p. 554)
 - [역추적](#) (p. 509)
 - [멀티 마스터 클러스터](#) (p. 584)
 - [데이터베이스 복제](#) (p. 239)
 - [IAM 데이터베이스 인증](#) (p. 976)
 - [MySQL DB 인스턴스에서 스냅샷 복원](#) (p. 492)
 - [Amazon RDS 성능 개선 도우미](#) (p. 365)

Note

AWS Lambda에서 Aurora Serverless DB 클러스터에 액세스할 수 있습니다. AWS Lambda 작업에 대한 자세한 내용은 AWS Lambda 개발자 안내서에 있는 [Amazon VPC의 리소스에 액세스하도록 Lambda 함수 구성](#) 단원을 참조하십시오.

Aurora Serverless에서 TLS/SSL 사용

전송 계층 보안/보안 소켓 계층(TLS/SSL) 프로토콜을 사용하여 Aurora Serverless 클러스터에 연결할 수 있습니다. 이를 위해서는 [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#)에 설명된 것과 동일한 일반적인 절차를 사용합니다.

Note

Aurora Serverless 클러스터를 사용하면 Amazon RDS SSL/TLS 인증서를 다운로드하여 사용할 필요가 없습니다. 대신에 AWS Certificate Manager(ACM)의 인증서를 사용합니다. 자세한 내용은 [AWS Certificate Manager 사용 설명서](#)를 참조하십시오.

세션에서 클라이언트와 Aurora Serverless VPC 엔드포인트 간에 TLS를 사용할 수 있도록 할 수 있습니다. 이렇게 하려면 `--ssl-mode` 파라미터를 사용하여 클라이언트 측에서 요구 사항을 지정하십시오. SSL 세션 번수는 Aurora Serverless DB 클러스터에 대한 SSL 연결에 대해서는 설정되지 않습니다.

Aurora Serverless는 TLS 프로토콜 버전 1.0, 1.1 및 1.2를 지원합니다. 하지만 Aurora Serverless 데이터베이스를 TLS용으로 구성할 필요는 없습니다. 특히 SSL 관련 데이터베이스 사용자 권한에서 `REQUIRE` 절을 사용하지 마십시오. 이렇게 하면 사용자가 연결할 수 없습니다.

기본적으로 클라이언트 프로그램은 `--ssl-mode` 옵션을 통해 사용할 수 있는 추가 제어 기능으로 Aurora Serverless로 암호화된 연결을 설정합니다. 클라이언트 측에서 Aurora Serverless는 모든 SSL 모드를 지원합니다.

Note

중국(베이징) AWS 리전에서는 현재 Aurora Serverless 클러스터에 대한 TLS가 지원되지 않습니다.

`mysql` 및 `psql` 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.

`mysql` 또는 `psql` 클라이언트를 `--ssl-mode VERIFY_CA` 또는 `VERIFY_IDENTITY`와 함께 사용하는 경우 CA를 가리키는 `--ssl-ca` 옵션을 .pem 형식으로 지정하십시오. 사용할 수 있는 .pem 파일을 보려면 Amazon Trust Services에서 [Amazon 루트 CA 1 트러스트 스토어](#)를 다운로드하십시오.

Aurora Serverless는 와일드카드 인증서를 사용합니다. `mysql` 클라이언트를 사용하여 SSL 모드 `VERIFY_IDENTITY`로 연결하는 경우 현재는 MySQL 8.0 호환 `mysql` 명령을 사용해야 합니다.

Aurora Serverless 작동 방식

Aurora Serverless(프로비저닝된 DB 클러스터) 없이 Amazon Aurora로 작업하는 경우 DB 인스턴스 클래스 크기를 선택하고 읽기 처리량을 늘리기 위해 Aurora 복제본을 생성할 수 있습니다. 워크로드가 변경되면 DB

인스턴스 클래스 크기를 수정하고 Aurora 복제본 수를 변경할 수 있습니다. 예측한 워크로드를 기반으로 용량을 수동으로 조정할 수 있기 때문에 이 모델은 데이터베이스 워크로드를 예측할 수 있는 경우 잘 작동합니다.

그러나 일부 환경에서는 워크로드가 간헐적으로 발생하기 때문에 예측이 불가능합니다. 많은 워크로드가 불과 몇 분 또는 몇 시간 동안 지속될 수 있고, 적은 활동이 오래 지속되거나 활동이 아예 없는 기간이 있을 수 있습니다. 간헐적인 판매 이벤트를 진행하는 소매 웹 사이트, 필요한 경우 보고서를 생성하는 보고 데이터베이스, 개발 및 테스트 환경, 필요량이 불확실한 새 애플리케이션 등을 예로 들 수 있습니다. 이러한 경우와 기타 많은 경우 정확한 시점에 정확한 용량을 구성하기가 어려울 수 있습니다. 또한 사용하지 않는 용량에 대해 지불하는 경우에는 비용이 불필요하게 증가할 수 있습니다.

Aurora Serverless를 사용하면 DB 인스턴스 클래스 크기를 지정하지 않고도 데이터베이스 엔드포인트를 생성할 수 있습니다. 사용자가 최대 및 최소 용량을 설정합니다. Aurora Serverless를 사용하면 워크로드를 자동 조정되는 리소스 플릿으로 라우팅하는 프록시 플릿에 데이터베이스 엔드포인트가 연결됩니다. 프록시 플릿으로 인해 Aurora Serverless가 최소 및 최대 용량 사양에 따라 자동으로 리소스를 조정하는 동안 연결이 끊기지 않습니다. 프록시 플릿을 사용하기 위해 데이터베이스 클라이언트 애플리케이션을 변경할 필요가 없습니다. Aurora Serverless는 연결을 자동으로 관리합니다. 항상 요청을 서비스할 준비가 되어 있는 "웜" 리소스 풀을 사용하므로 조정이 신속합니다. 스토리지와 처리가 분리되므로 처리 용량을 0까지 축소할 수 있고 스토리지에 대한 비용만 지불하면 됩니다.

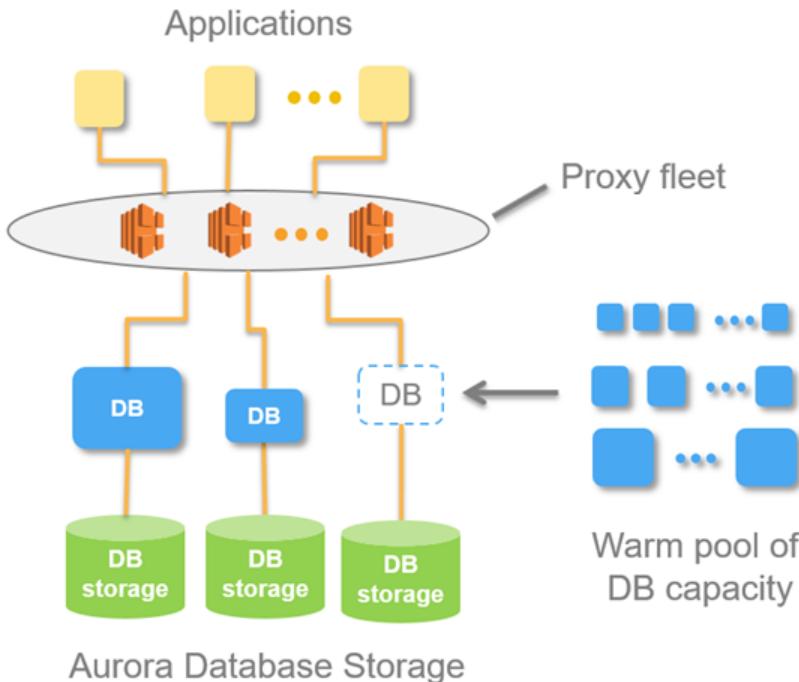
Aurora Serverless는 Aurora DB 클러스터에 새 serverless DB 엔진 모드를 도입합니다. 비 서버리스 DB 클러스터는 provisioned DB 엔진 모드를 사용합니다.

주제

- [Aurora Serverless 아키텍처 \(p. 115\)](#)
- [Aurora Serverless에서의 Auto Scaling \(p. 116\)](#)
- [Aurora Serverless 자동 일시 중지 및 다시 시작 \(p. 117\)](#)
- [용량 변경을 위한 제한 시간 조치 \(p. 117\)](#)
- [Aurora Serverless 및 파라미터 그룹 \(p. 117\)](#)
- [Aurora Serverless 및 유지 관리 \(p. 120\)](#)
- [Aurora Serverless 및 장애 조치 \(p. 120\)](#)
- [Aurora Serverless 및 스냅샷 \(p. 120\)](#)

Aurora Serverless 아키텍처

다음 이미지는 Aurora Serverless 아키텍처의 개요입니다.



데이터베이스 서버를 프로비저닝 및 관리하는 대신 ACU(Aurora 용량 단위)를 지정합니다. 각 ACU는 처리 용량과 메모리 용량의 조합입니다. 데이터베이스 스토리지는 표준 Aurora DB 클러스터의 스토리지와 동일한 10GiB~64 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 Serverless는 설정에 따라 CPU 사용률, 연결 및 가용 메모리 임계값에 대한 조정 규칙을 자동으로 생성합니다.

Aurora Serverless는 조정 시간을 최소화하기 위해 AWS 리전에서 리소스의 웜 풀을 관리합니다. Aurora Serverless가 Aurora DB 클러스터에 새 리소스를 추가하면 프록시 플릿을 사용하여 활성 클라이언트 연결을 새 리소스로 전환합니다. 특정 시간에 Aurora DB 클러스터에서 사용 중인 ACU에 대해서만 비용이 청구됩니다.

Aurora Serverless에서의 Auto Scaling

Aurora Serverless DB 클러스터에 할당된 용량은 클라이언트 애플리케이션에서 생성된 부하(CPU 사용률 및 연결 수)에 따라 원활하게 확장 및 축소됩니다. 또한 5분 간 연결이 없는 경우에는 용량이 0으로 조정됩니다.

CPU 또는 연결에서 용량 제약이 나타나면 Aurora Serverless가 확장됩니다. 또한 확장하면 해결할 수 있는 성능 문제를 감지해도 확장됩니다.

확장한 후 축소를 위한 휴지 기간은 15분입니다. 축소한 후 다시 축소하기 위한 휴지 기간은 310초입니다.

Note

확장할 경우에는 휴지 기간이 없습니다. Aurora Serverless는 확장 또는 축소 직후를 포함해 필요 시 언제든 확장할 수 있습니다.

조정점은 데이터베이스가 조정 작업을 안전하게 시작할 수 있는 시점입니다. 다음 조건에서는 Aurora Serverless가 조정점을 찾지 못할 수 있습니다.

- 장기간 쿼리 또는 트랜잭션이 진행 중인 경우

- 임시 테이블 또는 테이블 잠금이 사용 중인 경우

이러한 경우에는 Aurora Serverless가 조정 작업을 시작할 수 있도록 계속해서 조정점 찾기를 시도합니다. DB 클러스터가 조정되어야 한다고 결정하는 한, 이 작업을 수행합니다.

AWS Management 콘솔의 DB 클러스터에 대한 세부 정보에서 조정 이벤트를 확인할 수 있습니다. 또한 Amazon CloudWatch의 `ServerlessDatabaseCapacity` 지표를 사용하여 DB 클러스터에 할당된 현재 용량을 모니터링할 수도 있습니다.

autoscaling 동안 Aurora Serverless는 `EngineUptime` 지표를 재설정합니다. 재설정된 지표 값은 원활한 확장에 있어 어떤 문제도 나타내지 않으며, 이는 어떤 연결도 끊기지 않았다는 것을 의미합니다. 지표에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 지표 모니터링 \(p. 346\)](#) 단원을 참조하십시오.

Aurora Serverless 자동 일시 중지 및 다시 시작

활동 없이 지정된 시간이 경과하면 Aurora Serverless DB 클러스터를 일시 중지하도록 선택할 수 있습니다. DB 클러스터를 일시 중지하기 전에 활동 없이 경과하는 시간을 지정합니다. 기본값은 5분입니다. DB 클러스터 일시 중지를 비활성화할 수도 있습니다.

DB 클러스터가 일시 중지되면 컴퓨팅 또는 메모리 활동이 발생하지 않고 스토리지에 대한 요금만 청구됩니다. Aurora Serverless DB 클러스터가 일시 중지되었을 때 데이터베이스 연결이 요청되면 DB 클러스터가 자동으로 작동을 재개해 연결 요청을 처리합니다.

Note

DB 클러스터가 8일 이상 일시 중지된 경우 스냅샷을 사용하여 DB 클러스터를 백업할 수 있습니다. 이 경우 DB 클러스터에 대한 연결 요청이 있을 때 DB 클러스터가 복원됩니다.

용량 변경을 위한 제한 시간 조치

Aurora Serverless DB 클러스터의 용량을 변경할 수 있습니다. 용량 변경 시 Aurora Serverless는 이 변경에 대한 조정점을 찾으려고 합니다. Aurora Serverless는 조정점을 찾지 못하면 제한 시간을 초과하게 됩니다. 용량 변경 시간 제한을 초과할 때 취할 조치로 다음 중 한 가지를 지정할 수 있습니다.

- 용량 변경 강제 – 최대한 빨리 용량을 지정된 값에 설정합니다.
- 용량 변경 롤백 – 용량 변경을 취소합니다.

Important

용량 변경을 강제하면 Aurora 서비스가 조정점을 찾지 못하도록 방지하는 연결이 끊길 수 있습니다.

용량 변경에 대한 자세한 내용은 [Aurora Serverless DB 클러스터 수정 \(p. 127\)](#) 단원을 참조하십시오.

Aurora Serverless 및 파라미터 그룹

파라미터 그룹은 Aurora Serverless DB 클러스터에서 프로비저닝된 DB 클러스터와 서로 다르게 작동합니다. Aurora에서 직접 용량 설정을 관리합니다. 다른 종류의 Aurora 클러스터와 함께 사용하는 구성 프로시저, 기본 파라미터 값 등의 일부는 Aurora Serverless 클러스터에 적용되지 않습니다.

Aurora Serverless 클러스터의 DB 인스턴스에는 DB 파라미터 그룹이 아닌 연결된 DB 클러스터 파라미터 그룹만 있습니다. 서비스 클러스터는 DB 인스턴스가 Aurora Serverless 클러스터와 영구적으로 연결되어 있지 않으므로 DB 클러스터 파라미터 그룹에 의존합니다. Aurora는 필요에 따라 연결된 DB 인스턴스를 자동으로 확장합니다. 조정 작업에는 더 크거나 작은 용량에 적합하도록 파라미터 값을 수정하는 작업이 포함됩니다.

Aurora Serverless 클러스터의 구성 설정을 사용자 지정하려면 고유한 DB 클러스터 파라미터 그룹을 정의하고 이 그룹에 포함된 파라미터를 수정하면 됩니다. 클러스터 수준 파라미터와 다른 종류의 Aurora 클러스터에서 인스턴스 수준으로 적용되는 파라미터를 수정할 수 있습니다. 그러나 Aurora Serverless DB 클러스터와 연결된 DB 클러스터 파라미터 그룹을 수정하는 경우, 수정 사항은 다른 DB 클러스터 파라미터 그룹에 대해 적용되는 것과는 다른 방식으로 적용됩니다.

변경 사항을 Aurora Serverless DB 클러스터의 DB 클러스터 파라미터 그룹에 저장하면 변경 사항이 즉시 적용됩니다. 이 과정에서 Aurora Serverless는 다음 설정을 무시합니다.

- DB 클러스터 전체를 수정할 때 Aurora Serverless는 다음 설정을 무시합니다.
 - AWS Management 콘솔의 즉시 적용 설정
 - AWS CLI 명령인 [modify-db-cluster](#)에 있는 `--apply-immediately|--no-apply-immediately` 옵션
 - RDS API 연산인 [ModifyDBCluster](#)에 있는 `ApplyImmediately` 파라미터
- 파라미터 수정 시 Aurora Serverless는 AWS CLI 명령인 [modify-db-cluster-parameter-group](#) 및 [reset-db-cluster-parameter-group](#)에 있는 파라미터 목록의 `ApplyMethod` 값을 무시합니다.
- 파라미터 수정 시 Aurora Serverless는 RDS API 연산인 [ModifyDBClusterParameterGroup](#) 및 [ResetDBClusterParameterGroup](#)에 있는 파라미터 목록의 `ApplyMethod` 값을 무시합니다.

DB 클러스터 파라미터 그룹에 변경 사항을 적용하기 위해 Aurora Serverless는 DB 클러스터가 활성 상태인 경우 현재 용량으로 원활한 조정을 시작합니다. 일지 정지된 경우 DB 클러스터를 다시 시작합니다.

Important

Aurora는 `force-apply-capacity-change` 옵션으로 파라미터 그룹 변경에 대해 원활한 조정 작업을 수행합니다. 조정점을 찾을 수 없는 경우 Aurora Serverless가 조정점을 찾지 못하게 방해하는 연결이 삭제될 수 있습니다.

클러스터 수준 파라미터에 지원되는 엔진 모드를 보려면 [describe-engine-default-cluster-parameters](#) 명령 또는 RDS API 작업 [DescribeEngineDefaultClusterParameters](#)를 실행하십시오. 예를 들어 다음 Linux 명령은 Aurora MySQL Serverless 클러스터에 대해 설정할 수 있는 파라미터의 이름을 aurora5.6 기본 DB 클러스터 파라미터 그룹에서 추출합니다.

```
aws rds describe-engine-default-cluster-parameters \
--db-parameter-group-family aurora5.6 \
--query 'EngineDefaults.Parameters[*].{ParameterName:ParameterName, \
SupportedEngineModes:SupportedEngineModes} \
| [?contains(SupportedEngineModes, `serverless`) == `true`] \
| [*].{param:ParameterName}' \
--output text
```

다음 Linux 명령은 Aurora PostgreSQL Serverless 클러스터에 대해 설정할 수 있는 파라미터의 이름을 aurora-postgresql10 기본 DB 클러스터 파라미터 그룹에서 추출합니다.

```
aws rds describe-engine-default-cluster-parameters \
--db-parameter-group-family aurora-postgresql10 \
--query 'EngineDefaults.Parameters[*].{ParameterName:ParameterName, \
SupportedEngineModes:SupportedEngineModes} \
| [?contains(SupportedEngineModes, `serverless`) == `true`] \
| [*].{param:ParameterName}' \
--output text
```

다음 Linux 명령은 Serverless 클러스터에 설정할 수 있는 파라미터의 이름을 고객님이 생성한 DB 클러스터 파라미터 그룹에서 추출합니다.

```
aws rds describe-db-cluster-parameters \
--db-cluster-parameter-group-name my_cluster_param_group_name \
--query 'Parameters[*].{ParameterName:ParameterName,
SupportedEngineModes:SupportedEngineModes}
| [?contains(SupportedEngineModes, `serverless`) == `true`]
| [*].{param:ParameterName}' \
--output text
```

파라미터 그룹에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

Aurora MySQL Serverless DB 클러스터에서는 파라미터 값에 대한 수정이 다음 파라미터에 대해서만 적용됩니다. 다른 파라미터는 수정할 수 있지만 Aurora는 변경된 값을 사용하지 않습니다. 다른 모든 구성 파라미터의 경우 Aurora MySQL Serverless 클러스터는 기본값을 사용합니다. 이러한 기본값은 다른 종류의 Aurora 클러스터에 대해 다를 수 있습니다. 이러한 값은 Aurora에서 Aurora Serverless 클러스터를 확장하거나 축소함에 따라 변경될 수도 있습니다.

- `character_set_server`.
- `collation_server`.
- `general_log`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `innodb_file_format`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `innodb_file_per_table`.
- `innodb_large_prefix`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `innodb_lock_wait_timeout`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `innodb_monitor_disable`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `innodb_monitor_enable`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `innodb_monitor_reset`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `innodb_monitor_reset_all`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `innodb_print_all_deadlocks`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `lc_time_names`.
- `log_output`. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다. 이 설정의 기본값은 `FILE`입니다. 이 값은 변경할 수 없습니다.
- `log_queries_not_using_indexes`를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `log_warnings`를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `long_query_time`를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `lower_case_table_names`를 선택하십시오.
- `net_read_timeout`를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `net_retry_count`를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `net_write_timeout`를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `server_audit_logging`을 선택하십시오.
- `server_audit_events`를 선택하십시오.
- `server_audit_excl_users`를 선택하십시오.
- `server_audit_incl_users`를 선택하십시오.
- `slow_query_log`를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `sql_mode`를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.
- `time_zone`를 선택하십시오.

- tx_isolation를 선택하십시오. 이 설정은 이전에는 DB 인스턴스 파라미터 그룹에만 있었습니다.

Aurora Serverless 및 유지 관리

Aurora Serverless는 DB 클러스터가 최신 기능, 버그 수정 및 보안 업데이트를 유지할 수 있도록 정기적으로 유지 관리를 수행합니다. Aurora Serverless의 유지 관리는 가능하다면 종단 시간을 최소화하는 방식으로 이루어집니다.

유지 관리를 적용하려면 Aurora Serverless가 조정점을 찾아야 합니다. 조정점에 대한 자세한 내용은 [Aurora Serverless에서의 Auto Scaling \(p. 116\)](#) 단원을 참조하십시오.

Aurora Serverless DB 클러스터에 유지 관리가 필요한 경우 DB 클러스터가 7일 동안 유지 관리를 적용할 조정 지점을 찾으려고 시도합니다. 조정 지점을 찾을 수 없는 날마다 Aurora Serverless는 클러스터 이벤트를 생성하여 유지 관리를 적용하기 위해 조정해야 함을 알립니다. 알림에는 ScalingConfiguration 설정에 관계없이 유지 관리를 적용하기 위한 ForceApplyCapacityChange 제한 시간 작업과 함께 조정이 적용될 날짜가 포함됩니다. DB 클러스터에 RollbackCapacityChange가 해당 ScalingConfiguration에 대해 TimeoutAction으로 설정된 경우 Aurora Serverless는 이벤트에 포함된 시간 이전에 RollbackCapacityChange 시간 초과 작업을 사용하여 유지 관리를 적용하려고 합니다. DB 클러스터에 ForceApplyCapacityChange가 해당 ScalingConfiguration에 대해 TimeoutAction으로 설정된 경우 유지 관리를 적용하는 조정이 모든 시도에 이 클러스터를 사용합니다. ForceApplyCapacityChange를 사용하면 워크로드가 중단될 수 있습니다. 자세한 내용은 [용량 변경을 위한 제한 시간 조치 \(p. 117\)](#) 단원을 참조하십시오.

Note

그러면 유지 관리 기간이 Aurora Serverless에 적용되지 않습니다.

Aurora Serverless 및 장애 조치

Aurora Serverless DB 클러스터의 DB 인스턴스를 사용할 수 없게 되거나 가용 영역(AZ)에 장애가 발생하면 Aurora가 다른 AZ에서 DB 인스턴스를 재생성합니다. 이러한 기능을 자동 다중 AZ 장애 조치라고 부릅니다.

이러한 장애 조치 메커니즘은 Aurora 프로비저닝된 클러스터보다 시간이 오래 걸립니다. Aurora Serverless 장애 조치 시간은 해당 AWS 리전 내 다른 AZ의 수요 및 용량 가능성에 따라 다르기 때문에 현재 정의되어 있지 않습니다.

Aurora는 컴퓨팅 용량과 스토리지를 분리하기 때문에 클러스터의 스토리지 볼륨은 다중 AZ로 분산됩니다. 따라서 시스템 종단으로 DB 인스턴스 또는 연결된 AZ가 영향을 받더라도 데이터는 계속해서 사용할 수 있습니다.

Aurora Serverless 및 스냅샷

Aurora Serverless 클러스터의 클러스터 볼륨은 항상 암호화됩니다. 암호화 키를 선택할 수는 있으나 암호화는 끌 수 없습니다. Aurora Serverless 클러스터의 스냅샷을 복사하거나 공유하려면 고객님 고유의 KMS 키를 사용해 스냅샷을 암호화합니다.

Aurora Serverless DB 클러스터 생성

Aurora Serverless DB 클러스터를 생성하는 경우 해당 클러스터에 대한 최소 및 최대 용량을 설정할 수 있습니다. 각 용량 단위는 특정 컴퓨팅 및 메모리 구성과 동일합니다. Aurora Serverless는 CPU 사용률, 연결 및 가용 메모리 임계값에 대한 조정 규칙을 자동으로 생성합니다. 또한 활동이 없을 때 Aurora Serverless가 데 이터베이스를 일시 중지하고 활동이 다시 시작되면 다시 시작하도록 할지 여부를 설정할 수도 있습니다.

다음 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.

- Timeout action(제한 시간 조치) – 조정점을 찾지 못해 용량 수정 시간 제한을 초과한 경우 취할 조치입니다. Aurora는 용량 변경을 강제하여 용량을 최대한 빨리 지정된 값에 설정할 수 있습니다. 또는 용량 변경을 뒤로 밀어 변경을 취소할 수 있습니다. 자세한 내용은 [용량 변경을 위한 제한 시간 조치 \(p. 117\)](#) 단원을 참조하십시오.
- 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.

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 Aurora Serverless DB 클러스터를 생성할 수 있습니다.

DB 클러스터 생성에 대한 일반적인 내용은 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

Note

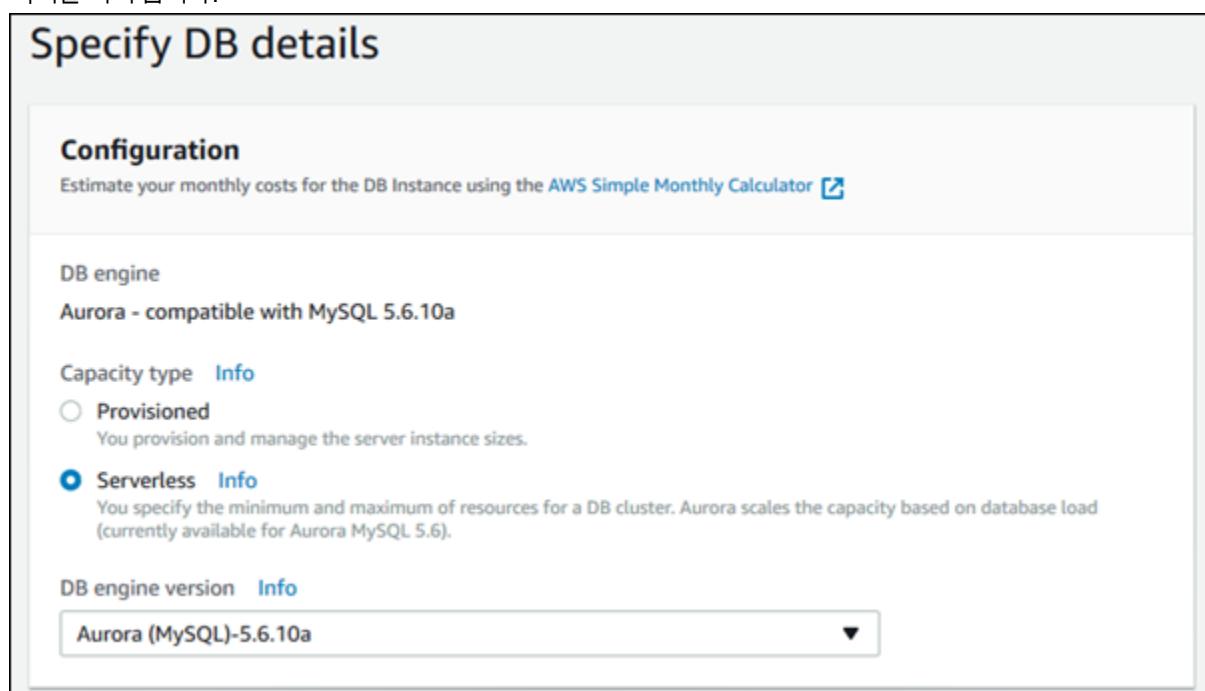
현재 Aurora Serverless를 모든 AWS 리전에서 사용할 수 있는 것은 아닙니다. Aurora Serverless에 대한 자세한 내용은 MySQL 호환 버전 또는 PostgreSQL 호환 버전 아래의 [요금](#)을 참조하십시오. Aurora Serverless 클러스터의 클러스터 볼륨은 항상 암호화됩니다. 암호화 키를 선택할 수는 있으나 암호화는 끌 수 없습니다. 따라서 암호화된 스냅샷에 대해 허용되지 않은 작업을 수행할 수 없습니다. 예를 들어 Aurora Serverless 클러스터의 스냅샷을 다른 AWS 리전에 복사할 수 없습니다.

콘솔

AWS Management 콘솔을 사용하여 새 Aurora Serverless DB 클러스터를 생성하려면 DB 세부 정보 지정 페이지에서 용량 유형에 서비스를 지정하십시오.

Aurora MySQL 예

다음 이미지는 Aurora MySQL 엔진에서 DB 클러스터를 생성할 때 서비스를 선택한 DB 세부 정보 지정 페이지를 나타냅니다.



고급 설정 구성 페이지의 용량 설정 섹션에서 값을 조정하여 Aurora Serverless DB 클러스터의 조정 구성을 구성할 수 있습니다.

다음 이미지는 Aurora MySQL Serverless DB 클러스터에서 조정할 수 있는 Capacity settings(용량 설정)을 나타낸 것입니다.

Capacity settings
Billing estimate is based on published prices. [Learn more](#)

Minimum Aurora capacity unit [Info](#)
1 2GB RAM

Maximum Aurora capacity unit [Info](#)
64 122GB RAM

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)
Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)
You are only charged for database storage while the compute capacity is paused
00 hours 05 minutes 00 seconds
Max: 24 hours

Aurora MySQL Serverless DB 클러스터에 대해 Data API를 활성화할 수도 있습니다. 자세한 내용은 [Aurora Serverless에 데이터 API 사용 \(p. 134\)](#) 단원을 참조하십시오.

Aurora PostgreSQL 예

Aurora PostgreSQL 엔진에서 먼저 PostgreSQL 버전 10.7과 호환되는 Aurora PostgreSQL의 DB 엔진 버전을 선택합니다. 그런 다음 용량 유형에서 서비스를 선택합니다.

Configuration
Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#)

DB engine
[Aurora PostgreSQL](#)

Capacity type [Info](#)

Provisioned
You provision and manage the server instance sizes.

Provisioned with Aurora parallel query enabled [Info](#)
You provision and manage the server instance sizes, and Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6)

Serverless [Info](#)
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.

DB engine version [Info](#)

[Aurora PostgreSQL \(compatible with PostgreSQL 10.7\)](#)

고급 설정 구성 페이지의 용량 설정 섹션에서 값을 조정하여 Aurora Serverless DB 클러스터의 조정 구성을 구성할 수 있습니다.

다음 이미지는 Aurora PostgreSQL Serverless DB 클러스터에서 조정할 수 있는 Capacity settings(용량 설정)을 나타낸 것입니다.

The screenshot shows the 'Capacity settings' section of the AWS Management Console. It includes fields for 'Minimum Aurora capacity unit' (set to 8, 16GB RAM) and 'Maximum Aurora capacity unit' (set to 384, 768GB RAM). Below these are sections for 'Additional scaling configuration' and 'Compute capacity pause settings'. The 'Compute capacity pause settings' section is expanded, showing a checkbox for pausing compute capacity after 5 minutes of inactivity, with a timer set to 00 hours, 05 minutes, and 00 seconds. A note indicates that compute capacity is paused during database storage.

AWS Management 콘솔을 사용하여 Aurora DB 클러스터 생성에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

Aurora Serverless DB 클러스터에 연결하려면 데이터베이스 엔드포인트를 사용합니다. 자세한 내용은 [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#) 단원의 지침을 참조하십시오.

Note

다음 오류 메시지가 표시되면 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 사용 \(p. 1001\)](#) 단원을 참조하십시오.

AWS CLI

AWS CLI를 사용하여 새 Aurora Serverless DB 클러스터를 생성하려면 `create-db-cluster` 명령을 실행하고 `--engine-mode` 옵션에 `serverless`를 지정하십시오.

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 MySQL 5.6 호환 Serverless DB 클러스터가 새롭게 생성됩니다. Aurora MySQL에서 유효한 용량 값은 1, 2, 4, 8, 16, 32, 64, 128 및 256입니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora --engine-version 5.6.10a \
--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 --engine-version 5.6.10a ^
--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 PostgreSQL 10.7-호환 Serverless DB 클러스터가 새롭게 생성됩니다. Aurora PostgreSQL에 유효한 용량 값은 2, 4, 8, 16, 32, 64, 192 및 384입니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora-postgresql
--engine-version 10.7 \
--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 10.7 ^
--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 DB 클러스터를 생성하려면 [CreateDBCluster](#) 작업을 실행하고 EngineMode 파라미터에 serverless를 지정하십시오.

최소 용량, 최대 용량 및 연결이 없는 경우 자동 일시 종지를 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 DB 클러스터 복원

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 프로비저닝된 DB 클러스터 스냅샷을 복원하는 경우 Aurora Serverless DB 클러스터를 구성할 수 있습니다.

Aurora Serverless DB 클러스터로 스냅샷을 복원할 때 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.
- Timeout action(제한 시간 조치) – 조정점을 찾지 못해 용량 수정 시간 제한을 초과한 경우 취할 조치입니다. Aurora는 용량 변경을 강제하여 용량을 최대한 빨리 지정된 값에 설정할 수 있습니다. 또는 용량 변경을 끝내기 위해 변경을 취소할 수 있습니다. 자세한 내용은 [용량 변경을 위한 제한 시간 조치 \(p. 117\)](#) 단원을 참조하십시오.
- 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 \(p. 273\)](#) 단원을 참조하십시오.

콘솔

AWS Management 콘솔을 사용하여 DB 클러스터 스냅샷을 Aurora DB 클러스터로 복원할 수 있습니다.

DB 클러스터 스냅샷을 Aurora DB 클러스터로 복원하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management 콘솔의 오른쪽 상단 모서리에서 원본 DB 클러스터를 호스팅하는 AWS 리전을 선택합니다.
3. 탐색 창에서 스냅샷을 선택한 다음 복원하려는 DB 클러스터 스냅샷을 선택합니다.
4. 작업에서 스냅샷 복원을 선택합니다.
5. DB 클러스터 복원 페이지에서 용량 유형으로 서비스를 선택합니다.

Restore DB Cluster

Instance specifications

DB Engine

Name of the Database Engine

Aurora MySQL

DB Engine Version

Version Number of the Database Engine to be used for this instance

Aurora (MySQL)-5.6.10a (default)

Capacity type [Info](#)

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 (currently available for Aurora MySQL 5.6).

6. DB 클러스터 식별자 필드에 복원한 DB 클러스터의 이름을 입력하고 다른 필드를 작성합니다.
7. 용량 설정 섹션에서 조정 구성을 수정합니다.

Capacity settings

Billing estimate is based on published prices. [Learn more](#) 

Minimum Aurora capacity unit [Info](#)

1

2GB RAM

Maximum Aurora capacity unit [Info](#)

64

122GB RAM

▼ Additional scaling configuration

Force scaling the capacity to the specified values when the timeout is reached [Info](#)

Enable to force capacity scaling as soon as possible. Disable to cancel the capacity changes when a timeout is reached

Pause compute capacity after consecutive minutes of inactivity [Info](#)

You are only charged for database storage while the compute capacity is paused

00

▼

hours

05

▼

minutes

00

▼

seconds

Max: 24 hours

8. DB 클러스터 복원을 선택합니다.

Aurora Serverless DB 클러스터에 연결하려면 데이터베이스 엔드포인트를 사용합니다. 자세한 내용은 [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#) 단원의 지침을 참조하십시오.

Note

다음 오류 메시지가 표시되면 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 사용 \(p. 1001\)](#) 단원을 참조하십시오.

AWS CLI

AWS CLI를 사용하여 DB 클러스터에서 복원할 때 Aurora Serverless DB 클러스터를 구성하려면 [restore-db-cluster-from-snapshot](#) CLI 명령을 실행하고 --engine-mode 옵션에 serverless를 지정하십시오.

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 복원합니다. `mynewdbcluster`라는 새 DB 클러스터로 복원합니다. DB 클러스터를 Aurora Serverless DB 클러스터로 복원하려면 --engine-mode 옵션을 serverless로 설정하십시오. 다음 예제는 --scaling-configuration 옵션에 대한 값도 지정합니다.

Linux, OS X, Unix의 경우:

```
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=1000,AutoP
```

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=1000,AutoP
```

RDS API

RDS API를 사용하여 DB 클러스터에서 복원할 때 Aurora Serverless DB 클러스터를 구성하려면 [RestoreDBClusterFromSnapshot](#) 작업을 실행하고 EngineMode 파라미터에 serverless를 지정하십시오.

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 DB 클러스터 수정

Aurora Serverless DB 클러스터를 구성한 후 AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 조정 구성을 수정할 수 있습니다.

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.

다음 특정 값을 설정할 수 있습니다.

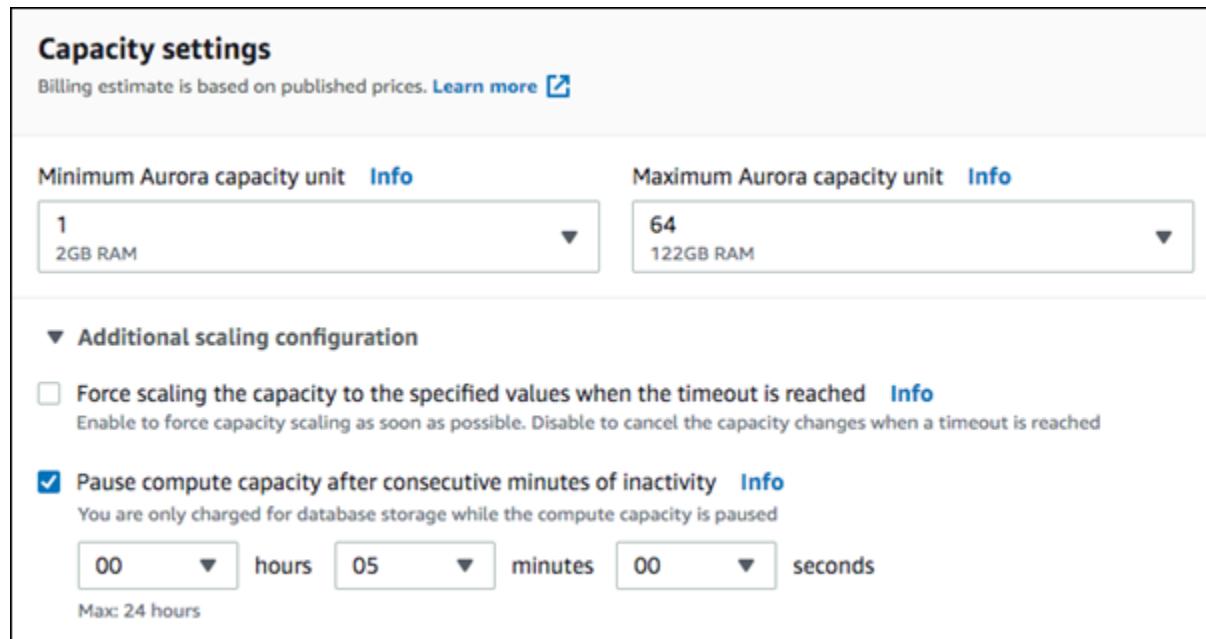
- 최소 Aurora 용량 단위 – Aurora Serverless는 이 용량 단위까지 용량을 줄일 수 있습니다.
- 최대 Aurora 용량 단위 – Aurora Serverless는 이 용량 단위까지 용량을 늘릴 수 있습니다.
- Timeout action(제한 시간 조치) – 조정점을 찾지 못해 용량 수정 시간 제한을 초과한 경우 취할 조치입니다. Aurora는 용량 변경을 강제하여 용량을 최대한 빨리 지정된 값에 설정할 수 있습니다. 또는 용량 변경을 끝내기 위해 변경을 취소할 수 있습니다. 자세한 내용은 [용량 변경을 위한 제한 시간 조치 \(p. 117\)](#) 단원을 참조하십시오.
- 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.

콘솔

AWS Management 콘솔을 사용하여 Aurora DB 클러스터의 조정 구성을 수정할 수 있습니다.

Aurora Serverless DB 클러스터를 수정하려면

1. <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 수정할 Aurora Serverless DB 클러스터를 선택합니다.
4. 작업에서 클러스터 수정을 선택합니다.
5. 용량 설정 섹션에서 조정 구성을 수정합니다.



6. [Continue]를 선택합니다.
7. Modify cluster(클러스터 수정)를 선택합니다.

변경 사항이 즉시 적용됩니다.

AWS CLI

AWS CLI를 사용하여 Aurora Serverless DB 클러스터의 조정 구성을 수정하려면 [modify-db-cluster](#) AWS CLI 명령을 실행하십시오. 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 DB 클러스터의 조정 구성을 수정합니다.

Linux, OS X, Unix의 경우:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster \
--scaling-configuration
MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange',AutoPa
```

Windows의 경우:

```
aws rds modify-db-cluster --db-cluster-identifier sample-cluster ^
--scaling-configuration
MinCapacity=8,MaxCapacity=64,SecondsUntilAutoPause=500,TimeoutAction='ForceApplyCapacityChange',AutoPa
```

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 DB 클러스터의 용량 설정

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 Aurora Serverless DB 클러스터의 용량을 특정 값으로 설정하십시오.

Aurora Serverless는 DB 클러스터의 워크로드를 기반으로 원활하게 조정합니다. 예를 들어, 신규 트랜잭션이 많은 경우처럼 경우에 따라서는 갑작스러운 워크로드 변동을 충족하기 위해 충분히 빠른 속도로 조정되지 않을 수 있습니다. 이러한 경우에는 용량을 명시적으로 설정할 수 있습니다. 용량을 명시적으로 설정하면 Aurora Serverless가 DB 클러스터를 자동으로 조정할 수 있습니다. 이 작업은 축소를 위한 휴지 기간에 근거하여 이루어집니다.

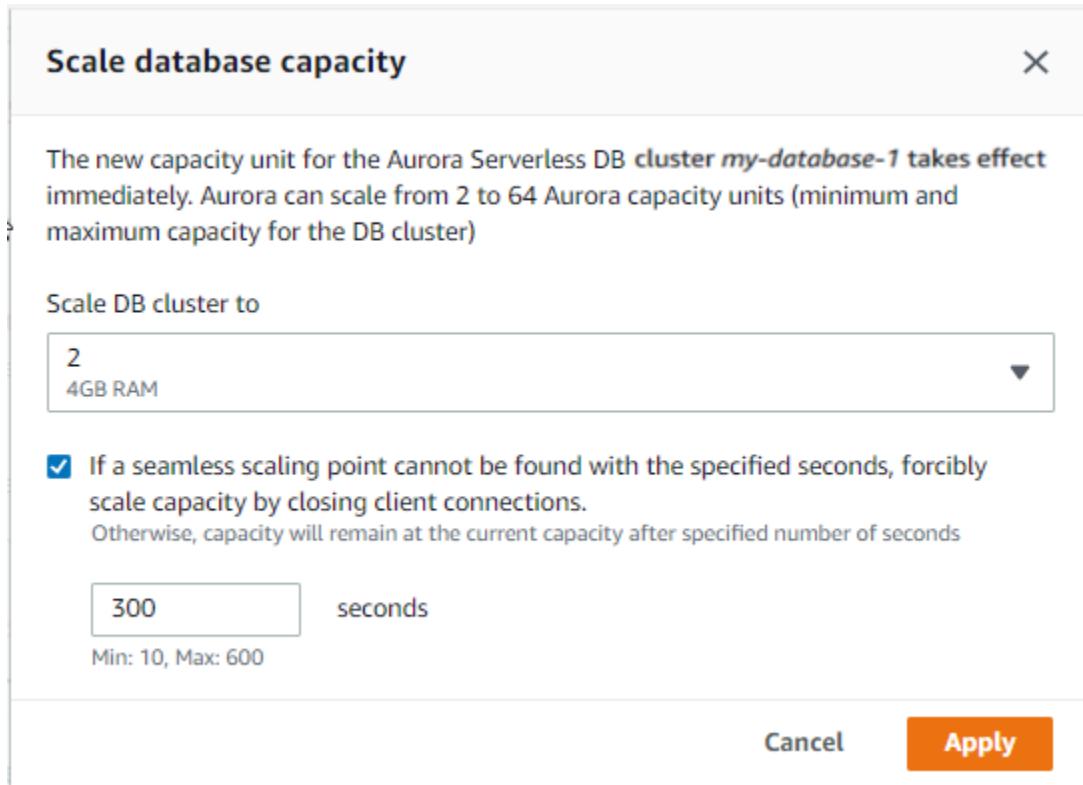
콘솔

AWS Management 콘솔을 사용하여 Aurora DB 클러스터의 용량을 설정할 수 있습니다.

Aurora Serverless DB 클러스터를 수정하려면

1. <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.

3. 수정할 Aurora Serverless DB 클러스터를 선택합니다.
4. 작업에서 Set capacity(용량 설정)를 선택합니다.
5. Scale database capacity(데이터베이스 용량 조정) 창에서 DB 클러스터에서 조정해야 할 용량을 설정합니다.



6. 옵션을 활성화 또는 비활성화하여 용량을 강제로 조정합니다. 옵션을 활성화하는 경우 기간을 지정하여 조정점을 찾으십시오. Aurora Serverless가 지정된 제한 시간 이전에 조정점을 찾기 시작합니다. 제한 시간에 도달하면 Aurora Serverless가 다음 작업 중 한 가지를 수행합니다.
 - 확인된 경우 제한 시간이 지나면 Aurora Serverless가 조정 용량을 강제로 변경합니다.
 - 확인되지 않은 경우 제한 시간이 지나면 Aurora Serverless가 조정 용량을 변경하지 않습니다.

Important

옵션이 활성화되면 Aurora 서비스가 조정점을 찾지 못하도록 방지하는 연결이 끊길 수 있습니다. 조정점 및 휴지 기간에 대한 자세한 내용은 [Aurora Serverless에서의 Auto Scaling \(p. 116\)](#) 단원을 참조하십시오.

7. 적용을 선택합니다.

AWS CLI

AWS CLI를 사용하여 Aurora Serverless DB 클러스터의 용량을 설정하려면 [modify-current-db-cluster-capacity](#) AWS CLI 명령을 실행하고 --capacity 옵션을 지정하십시오. 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 DB 클러스터의 용량을 64로 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 DB 클러스터 보기

Aurora Serverless DB 클러스터를 하나 이상 생성하면 유형이 서버리스인 DB 클러스터와 인스턴스인 DB 클러스터를 확인할 수 있습니다. 각 Aurora Serverless DB 클러스터에서 사용 중인 Aurora 용량 단위(ACU)의 현재 개수도 확인할 수 있습니다. 각 ACU는 처리 용량과 메모리 용량의 조합입니다.

Aurora Serverless DB 클러스터를 보려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management 콘솔의 상단 오른쪽에서 Aurora Serverless DB 클러스터를 생성한 AWS 리전을 선택합니다.
3. 탐색 창에서 데이터베이스를 선택합니다.

각 DB 클러스터의 DB 클러스터 유형은 역할 아래에 표시됩니다. Aurora Serverless DB 클러스터는 유형에 대해 서버리스라고 표시합니다. Aurora Serverless DB 클러스터의 현재 용량은 크기에서 확인할 수 있습니다.

DB identifier	Role	Engine	Region & AZ	Size	Status
database-1	Serverless	Aurora MySQL	us-east-1	0 capacity units	✓ /
my-database-1	Serverless	Aurora PostgreSQL	us-east-1	0 capacity units	✓ /
mysql3	Instance	MySQL	us-east-1c	db.t3.medium	✓ /
tests3aurora	Regional	Aurora MySQL	us-east-1	2 instances	✓ /
tests3aurora	Writer	Aurora MySQL	us-east-1a	db.t3.medium	✓ /
tests3aurora-us-east-1b	Reader	Aurora MySQL	us-east-1b	db.t3.medium	✓ /

4. Aurora Serverless DB 클러스터의 이름을 선택하면 세부 정보가 표시됩니다.

DB 클러스터에 연결하는 데 필요하므로 Connectivity & security(연결성 및 보안) 탭에서 데이터베이스 엔드포인트를 적어둡니다.

database-1

Summary

DB cluster id	CPU
database-1	
Role	Current activity
Serverless	

Connectivity & security Monitoring Logs & events Configuration

Connectivity & security

Endpoint & port

Endpoint	database-1. [REDACTED].us-east-1.rds.amazonaws.com
Port	3306

Network

VPC

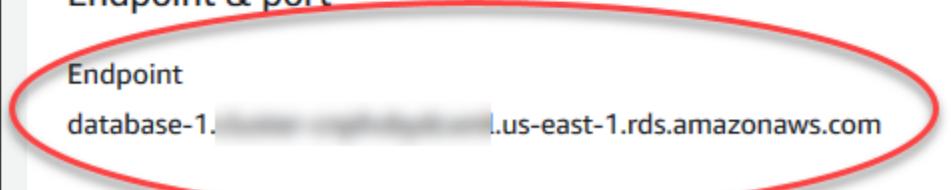
vpc-6

Subnet

default

Subnet

subnet



구성 탭을 선택하여 용량 설정을 확인합니다.

The screenshot shows the 'Configuration' tab selected in the top navigation bar. Under the 'Database' section, there is a 'Capacity settings' box highlighted with a red border. Inside this box, the following configuration details are listed:

- Minimum Aurora capacity unit: 2 capacity units
- Maximum Aurora capacity unit: 16 capacity units
- Pause compute capacity after consecutive minutes of inactivity: 5 minutes
- Force scaling the capacity to the specified values when the timeout is reached: Enabled

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.

The screenshot shows the 'Logs & events' tab selected in the top navigation bar. Under the 'Recent events (2)' section, two log entries are listed:

Time	System notes
Mon Aug 06 17:04:15 GMT-700 2018	The DB cluster has scaled from 8 capacity units to 4 capacity units.
Mon Aug 06 17:04:09 GMT-700 2018	Scaling DB cluster from 8 capacity units to 4 capacity units for this

또한 CloudWatch에서 Aurora Serverless DB 클러스터를 모니터링할 수도 있습니다. 특히, `ServerlessDatabaseCapacity` 지표를 사용하여 DB 클러스터에 할당된 용량을 모니터링할 수 있습니다. 또한 `CPUUtilization`, `DatabaseConnections`, `Queries` 등의 표준 Aurora CloudWatch 지표를 모두 모니터링할 수 있습니다. CloudWatch를 통해 Aurora 클러스터를 모니터링하는 방법에 대한 자세한 내용은 [Amazon CloudWatch에서 로그 이벤트 모니터링 \(p. 642\)](#) 단원을 참조하십시오.

Aurora가 데이터베이스 로그 중 일부 또는 전체를 CloudWatch에 게시하게 할 수 있습니다. 서비스 클러스터에 연결된 DB 클러스터 파라미터 그룹에 있는 `general_log`, `slow_query_log` 등의 구성 파라미터를 활성화하여 게시할 로그를 선택합니다. 프로비저닝된 클러스터와 달리 서비스 클러스터에서는 DB 클러스터 설정에서 CloudWatch로 업로드할 로그 유형을 지정할 필요가 없습니다. 서비스 클러스터는 모든 사용 로그를 자동으로 업로드합니다. 로그 구성 파라미터를 비활성화하면 CloudWatch에 대한 로그 게시가 중지됩니다. 더 이상 필요하지 않은 경우 CloudWatch에서 로그를 삭제할 수도 있습니다.

Aurora Serverless DB 클러스터에 연결하려면 데이터베이스 앤드포인트를 사용합니다. [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#)의 지침에 따라 Aurora Serverless DB 클러스터에 연결하십시오.

Note

Aurora Serverless를 사용하는 경우 DB 클러스터의 특정 DB 인스턴스에는 직접 연결할 수 없습니다.

Aurora Serverless에 데이터 API 사용

기본 제공된 데이터 API를 사용하여 Aurora Serverless DB 클러스터에 액세스할 수 있습니다. 이 API를 사용하면 AWS Lambda, AWS AppSync, AWS Cloud9 등 웹 서비스 기반 애플리케이션을 사용하여 Aurora Serverless에 액세스할 수 있습니다. 이러한 애플리케이션에 대한 자세한 내용은 [AWS Lambda](#), [AWS AppSync](#), [AWS Cloud9](#)에서 세부 정보를 참조하십시오.

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 API에 대한 모든 호출은 동기식입니다. 기본적으로 처리가 완료되지 않은 경우 호출은 시간 초과되어 45초 안에 종료됩니다. 호출 시간이 초과한 후에도 `continueAfterTimeout` 파라미터를 사용하여 SQL 문을 계속 실행할 수 있습니다.

이 API는 AWS Secrets Manager에 저장된 데이터베이스 자격 증명을 사용하므로, API 호출 시 자격 증명을 전달할 필요가 없습니다. API는 AWS Lambda를 사용하는 더 안전한 방법도 제공합니다. 가상 프라이빗 클라우드(VPC)의 리소스에 액세스하는 Lambda 함수를 구성할 필요 없이 DB 클러스터에 액세스할 수 있습니다. AWS Secrets Manager에 대한 자세한 정보는 AWS Secrets Manager 사용 설명서의 [AWS Secrets Manager란 무엇입니까?](#)를 참조하십시오.

Note

데이터 API를 활성화하면 Aurora Serverless에 쿼리 편집기를 사용할 수도 있습니다. 자세한 내용은 [Aurora Serverless에 쿼리 편집기 사용 \(p. 158\)](#) 단원을 참조하십시오.

데이터 API의 가능성

데이터 API는 다음과 같은 Aurora Serverless DB 클러스터에서만 사용할 수 있습니다.

- MySQL 버전 5.6과 호환되는 Aurora
- PostgreSQL 버전 10.7과 호환되는 Aurora

다음 표는 현재 Aurora Serverless에 데이터 API를 사용할 수 있는 AWS 리전을 보여줍니다. HTTPS 프로토콜을 사용하여 이러한 AWS 리전의 데이터 API에 액세스하십시오.

Region	링크
미국 동부(버지니아 북부)	rds-data.us-east-1.amazonaws.com
미국 동부(오하이오)	rds-data.us-east-2.amazonaws.com
미국 서부(오레곤)	rds-data.us-west-2.amazonaws.com
유럽(아일랜드)	rds-data.eu-west-1.amazonaws.com
아시아 태평양(도쿄)	rds-data.ap-northeast-1.amazonaws.com

데이터 API에 대한 액세스 권한 부여

사용자는 데이터 API에 액세스할 수 있는 권한이 있어야 합니다. 해당 사용자에게 미리 정의된 AWS Identity and Access Management(IAM) 정책인 `AmazonRDSDataFullAccess` 정책을 추가하여 데이터 API에 액세스할 수 있는 권한을 부여할 수 있습니다.

또한 데이터 API에 대한 액세스를 허가하는 IAM 정책을 생성할 수도 있습니다. 정책을 생성한 후에는 해당 정책을 데이터 API에 액세스해야 하는 각 사용자에게 추가합니다.

다음 정책은 사용자가 데이터 API에 액세스하는 데 필요한 최소 권한을 제공합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "SecretsManagerDbCredentialsAccess",  
            "Effect": "Allow",  
            "Action": [  
                "secretsmanager:GetSecretValue",  
                "secretsmanager:PutResourcePolicy",  
                "secretsmanager:PutSecretValue",  
                "secretsmanager>DeleteSecret",  
                "secretsmanager:DescribeSecret",  
                "secretsmanager:TagResource"  
            ],  
            "Resource": "arn:aws:secretsmanager:*:secret:rds-db-credentials/*"  
        },  
        {  
            "Sid": "RDSDataServiceAccess",  
            "Effect": "Allow",  
            "Action": [  
                "secretsmanager>CreateSecret",  
                "secretsmanager>ListSecrets",  
                "secretsmanager:GetRandomPassword",  
                "tag:GetResources",  
                "rds-data:BatchExecuteStatement",  
                "rds-data:BeginTransaction",  
                "rds-data:CommitTransaction",  
                "rds-data:ExecuteStatement",  
                "rds-data:RollbackTransaction"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

IAM 정책 생성에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 정책 생성](#)을 참조하십시오.

사용자에게 IAM 정책 추가에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 자격 증명 권한 추가 및 제거](#)를 참조하십시오.

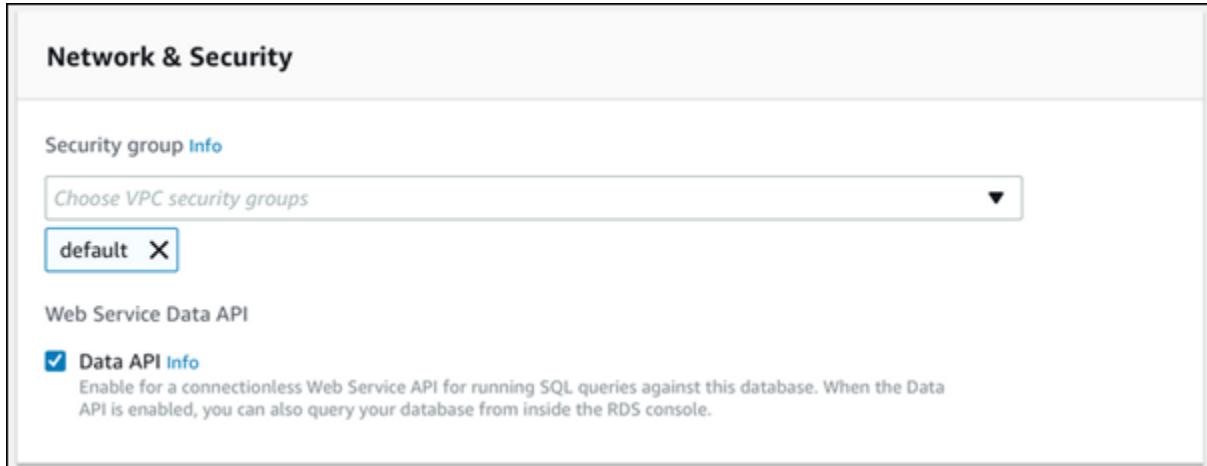
데이터 API 활성화

데이터 API를 사용하려면 Aurora Serverless DB 클러스터에 대해 데이터 API를 활성화합니다. DB 클러스터를 생성하거나 수정할 때 데이터 API를 활성화할 수 있습니다.

콘솔

Aurora Serverless DB 클러스터를 생성하거나 수정할 때 RDS 콘솔을 사용하여 데이터 API를 활성화할 수 있습니다. Aurora Serverless DB 클러스터를 생성할 때는 RDS 콘솔의 연결 섹션에서 데이터 API를 활성화합니다. Aurora Serverless DB 클러스터를 수정할 때는 RDS 콘솔의 네트워크 및 보안 섹션에서 데이터 API를 활성화합니다.

다음 스크린샷은 Aurora Serverless DB 클러스터를 수정할 때 활성화된 데이터 API를 보여줍니다.



지침은 [Aurora Serverless DB 클러스터 생성 \(p. 120\)](#) 및 [Aurora Serverless DB 클러스터 수정 \(p. 127\)](#) 단원을 참조하십시오.

AWS CLI

AWS CLI 명령을 사용하여 Aurora Serverless DB 클러스터를 생성하거나 수정할 때 `--enable-http-endpoint`를 지정하면 데이터 API가 활성화됩니다.

다음 AWS CLI 명령으로 `--enable-http-endpoint`를 지정할 수 있습니다.

- [create-db-cluster](#)
- [modify-db-cluster](#)

다음 예제는 데이터 API를 활성화하도록 `sample-cluster`를 수정합니다.

Linux, OS X, Unix의 경우:

```
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

Amazon RDS API 작업을 사용하여 Aurora Serverless DB 클러스터를 생성하거나 수정할 때 `EnableHttpEndpoint` 값을 `true`로 설정하여 데이터 API를 활성화할 수 있습니다.

다음 API 작업으로 `EnableHttpEndpoint` 값을 설정할 수도 있습니다.

- [CreateDBCluster](#)
- [ModifyDBCluster](#)

AWS Secrets Manager에 데이터베이스 자격 증명 저장

데이터 API를 호출할 때 AWS Secrets Manager의 보안 암호를 사용하여 Aurora Serverless DB 클러스터에 대한 자격 증명을 전달할 수 있습니다. 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. AWS Secrets Manager를 사용하여 Aurora DB 클러스터의 자격 증명을 포함하는 보안 암호를 생성합니다.

이에 관한 자침은 AWS Secrets Manager 사용 설명서의 [기본 암호 생성](#)을 참조하십시오.

2. AWS Secrets Manager 콘솔을 사용하여 생성한 보안 암호에 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.

데이터 API에 대한 Amazon VPC 엔드포인트(AWS PrivateLink) 생성

Amazon Virtual Private Cloud(Amazon VPC)를 사용하면 Aurora DB 클러스터 및 애플리케이션과 같은 AWS 리소스를 Virtual Private Cloud(VPC)로 시작할 수 있습니다. AWS PrivateLink는 Amazon 네트워크에서 Amazon 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 참조하십시오.

Amazon VPC 엔드포인트를 사용하여 데이터 API를 호출할 수 있습니다. Amazon VPC 엔드포인트를 사용하면 퍼블릭 IP 주소를 사용하지 않고도 Amazon VPC의 애플리케이션과 AWS 네트워크의 데이터 API 간 트래픽이 유지됩니다. 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 콘솔에 로그인한 다음 <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
2. 엔드포인트를 선택한 다음 엔드포인트 생성을 선택합니다.
3. 엔드포인트 생성 페이지의 서비스 범주에서 AWS 서비스를 선택합니다. 서비스 이름에 대해 `rds-data`를 선택합니다.

Create Endpoint

A VPC endpoint allows you to securely connect your VPC to another service.
An interface endpoint is powered by [PrivateLink](#), and uses an elastic network interface (ENI) as an entry point for traffic destined to the service.
A gateway endpoint serves as a target for a route in your route table for traffic destined for the service.

The screenshot shows the 'Create Endpoint' wizard. In the 'Service category' dropdown, 'AWS services' is selected. Below it are two other options: 'Find service by name' and 'Your AWS Marketplace services'. The 'Service Name' field contains 'com.amazonaws.eu-west-1.rds-data'. The main area shows a table of service names, owners, and types. The row for 'com.amazonaws.eu-west-1.rds-data' is highlighted with a red box.

Service Name	Owner	Type
com.amazonaws.eu-west-1.elastic-inferen...	amazon	Interface
com.amazonaws.eu-west-1.elasticfilesystem	amazon	Interface
com.amazonaws.eu-west-1.elasticfilesystem...	amazon	Interface
com.amazonaws.eu-west-1.elasticloadbal...	amazon	Interface
com.amazonaws.eu-west-1.elasticmapred...	amazon	Interface
com.amazonaws.eu-west-1.events	amazon	Interface
com.amazonaws.eu-west-1.execute-api	amazon	Interface
com.amazonaws.eu-west-1.glue	amazon	Interface
com.amazonaws.eu-west-1.kinesis-firehose	amazon	Interface
com.amazonaws.eu-west-1.kinesis-streams	amazon	Interface
com.amazonaws.eu-west-1.logs	amazon	Interface
com.amazonaws.eu-west-1.qldb.session	amazon	Interface
com.amazonaws.eu-west-1.rds-data	amazon	Interface
com.amazonaws.eu-west-1.rekognition	amazon	Interface
com.amazonaws.eu-west-1.s3	amazon	Gateway

- VPC의 경우 엔드포인트를 생성할 VPC를 선택합니다.

데이터 API를 호출하는 애플리케이션이 포함된 VPC를 선택합니다.

- 서브넷의 경우 애플리케이션을 실행 중인 AWS 서비스에서 사용하는 각 가용 영역(AZ)의 서브넷을 선택합니다.

The screenshot shows the 'Subnets' configuration screen. It lists three subnets under 'Availability Zone': 'eu-west-1a (euw1-az2)', 'eu-west-1b (euw1-az3)', and 'eu-west-1c (euw1-az1)'. Each subnet is associated with a specific Subnet ID: 'subnet-2c8ee364', 'subnet-cefa2594', and 'subnet-20c45946' respectively. All three checkboxes are checked.

Availability Zone	Subnet ID
eu-west-1a (euw1-az2)	subnet-2c8ee364
eu-west-1b (euw1-az3)	subnet-cefa2594
eu-west-1c (euw1-az1)	subnet-20c45946

Amazon VPC 엔드포인트를 생성하려면 엔드포인트에 액세스 할 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.

- Enable DNS name(DNS 이름 활성화)에서 이 엔드포인트에 대해 활성화를 선택합니다.

The screenshot shows the 'Enable DNS name' section. A checkbox labeled 'Enable for this endpoint' is checked. Below the checkbox, there is a note: 'To use private DNS names, ensure that the attributes 'Enable DNS hostnames' and 'Enable DNS Support' are set to 'true' for your VPC (vpc-c20d2da4). Learn more.'.

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 액세스 할 수 있습니다.

- 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 허용

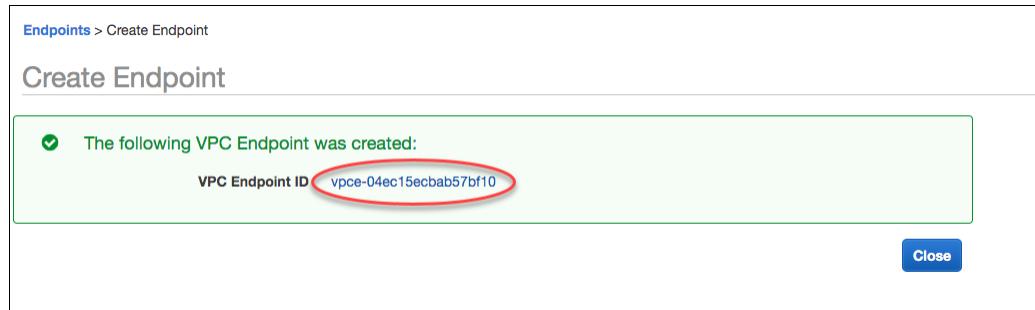
하는 보안 그룹을 선택합니다. 보안 그룹을 사용하면 VPC의 리소스에서 Amazon VPC 엔드포인트로 가는 트래픽을 제어할 수 있습니다.

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 콘솔에서 링크를 선택하여 엔드포인트 세부 정보를 봅니다.



엔드포인트 세부 정보 탭에는 Amazon VPC 엔드포인트를 만드는 동안 생성된 DNS 호스트 이름이 표시됩니다.

Endpoint: vpce-04ec15ecbab57bf10						
	Details	Subnets	Security Groups	Policy	Notifications	Tags
Endpoint ID	vpce-04ec15ecbab57bf10			VPC ID	vpce-c20d2da4	
Status	available			Status message	com.amazonaws.eu-west-1.rds-data	
Creation time	January 31, 2020 at 7:02:32 PM UTC-8			Service name	vpce-[REDACTED].rds-data.eu-west-1.vpce.amazonaws.com	
Endpoint type	Interface			DNS names	vpce-[REDACTED].eu-west-1.rds.rds-data.eu-west-1.vpce.amazonaws.com vpce-[REDACTED].eu-west-1.vpce.amazonaws.com vpce-[REDACTED].eu-west-1c.rds.rds-data.eu-west-1.vpce.amazonaws.com vpce-[REDACTED].eu-west-1.rds.rds-data.eu-west-1.vpce.amazonaws.com vpce-[REDACTED].eu-west-1.vpce.amazonaws.com rds-data.eu-west-1.amazonaws.com rds-data.eu-west-1.amazonaws.com	
Private DNS names enabled	true			Private DNS name	rds-data.eu-west-1.amazonaws.com	

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 [데이터 API 호출 \(p. 139\)](#) 단원을 참조하십시오.

데이터 API 호출

Aurora Serverless DB 클러스터에 대해 데이터 API를 활성화한 후, 데이터 API 또는 AWS CLI를 호출하여 DB 클러스터에 대해 SQL 문을 실행할 수 있습니다. 데이터 API는 AWS SDK에서 지원되는 프로그래밍 언어를 지원합니다. 자세한 내용은 [AWS 기반의 도구](#)를 참조하십시오.

데이터 API는 SQL 문을 실행하는 다음 작업을 제공합니다.

데이터 API 작업	AWS CLI 명령	설명
	<code>ExecuteStatement</code>	데이터베이스에 대해 SQL 문을 실행합니다.
	<code>BatchExecuteStatement</code>	대량 업데이트 및 삽입 작업을 위해 데이터 배열을 통해 일괄 SQL 문을 실행합니다. 파라미터 세트의 배열을 사용하여 DML 문을 실행할 수 있습니다. 일괄 SQL 문은 개별 삽입 및 업데이트 문을 통해 중요한 성능 개선을 제공합니다.

SQL 문을 독립적으로 실행하기 위해 두 작업을 모두 실행하거나 트랜잭션에서 실행할 수 있습니다. 데이터 API는 트랜잭션을 지원하기 위해 다음 작업을 제공합니다.

데이터 API 작업	AWS CLI 명령	설명
	<code>BeginTransaction</code>	SQL 트랜잭션을 시작합니다.
	<code>CommitTransaction</code>	SQL 트랜잭션을 종료하고 변경을 수행합니다.
	<code>RollbackTransaction</code>	트랜잭션의 롤백을 수행합니다.

SQL 문을 실행하고 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	Required	설명
	<code>resourceArn --resource-arn</code>	예	Aurora Serverless DB 클러스터의 Amazon 리소스 이름(ARN)입니다.
	<code>secretArn --secret-arn</code>	예	DB 클러스터에 대한 액세스를 활성화하는 보안 암호의 이름 또는 ARN.

`ExecuteStatement` 및 `BatchExecuteStatement`에 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 유형을 데이터 API 호출에서 지정하는 데이터 형식에 매핑합니다.

JDBC 데이터 형식	데이터 API 데이터 형식
<code>INTEGER, TINYINT, SMALLINT, BIGINT</code>	<code>LONG</code>
<code>FLOAT, REAL, DOUBLE</code>	<code>DOUBLE</code>
<code>DECIMAL</code>	<code>STRING</code>

JDBC 데이터 형식	데이터 API 데이터 형식
BOOLEAN, BIT	BOOLEAN
BLOB, BINARY, LONGVARBINARY, VARBINARY	BLOB
CLOB	STRING
다른 형식(날짜 및 시간과 관련된 형식 포함)	STRING

DECIMAL 또는 TIME과 같은 일부 특정 유형의 경우 String 값을 데이터베이스에 다른 유형으로 전달해야 한다고 데이터 API에 지시하는 힌트가 필요할 수도 있습니다. 이렇게 하려면 SqlParameter 데이터 유형의 typeHint에 값을 포함하면 됩니다. typeHint에 가능한 값은 다음과 같습니다.

- DECIMAL – 해당되는 String 파라미터 값은 DECIMAL 유형의 객체로 데이터베이스에 전송됩니다.
- TIMESTAMP – 해당되는 String 파라미터 값은 TIMESTAMP 유형의 객체로 데이터베이스에 전송됩니다. 승인된 형식은 YYYY-MM-DD HH:MM:SS[.FFF]입니다.
- TIME – 해당되는 String 파라미터 값은 TIME 유형의 객체로 데이터베이스에 전송됩니다. 승인된 형식은 HH:MM:SS[.FFF]입니다.
- DATE – 해당되는 String 파라미터 값은 DATE 유형의 객체로 데이터베이스에 전송됩니다. 승인된 형식은 YYYY-MM-DD입니다.

Note

현재, 데이터 API는 UUID(범용 고유 식별자) 배열을 지원하지 않습니다.

예제

- [AWS CLI를 사용하여 데이터 API 호출 \(p. 141\)](#)
- [Python 애플리케이션에서 데이터 API 호출 \(p. 148\)](#)
- [Java 애플리케이션에서 데이터 API 호출 \(p. 151\)](#)

Note

여기 나온 예제가 전부는 아닙니다.

AWS CLI를 사용하여 데이터 API 호출

AWS Command Line Interface(AWS CLI)를 사용하여 데이터 API를 호출할 수 있습니다.

다음 예제에서는 데이터 API용 AWS CLI를 사용합니다. 자세한 내용은 [AWS Command Line Interface 데이터 API 참조](#)를 참조하십시오.

각 예제에서 DB 클러스터 ARN을 Aurora Serverless DB 클러스터의 ARN으로 바꿉니다. 또한 DB 클러스터에 대한 액세스를 허용하는 AWS Secrets Manager의 암호 ARN으로 암호 ARN을 바꿉니다.

Note

AWS CLI는 응답을 JSON 형식으로 지정할 수 있습니다.

주제

- [SQL 트랜잭션 시작 \(p. 142\)](#)
- [SQL 문 실행 \(p. 142\)](#)

- 데이터 배열에서 일괄 SQL 문 실행 (p. 146)
- SQL 트랜잭션 커밋 (p. 147)
- SQL 트랜잭션 룰백 (p. 148)

SQL 트랜잭션 시작

aws rds-data begin-transaction CLI 명령을 사용하여 SQL 트랜잭션을 시작할 수 있습니다. 이 호출은 트랜잭션 식별자를 반환합니다.

Important

3분 안에 트랜잭션 ID를 사용하는 호출이 없는 경우 트랜잭션 시간이 초과됩니다. 커밋되기 전에 트랜잭션 시간이 초과되면 자동으로 룰백됩니다.
트랜잭션 내부의 DDL 문은 암시적 커밋을 발생시킵니다. --continue-after-timeout 옵션과 함께 별도의 execute-statement 명령으로 각 DDL 문을 실행하는 것이 좋습니다.

일반 옵션 외에 다음 옵션을 지정하십시오.

- --database (선택 사항) – 데이터베이스 이름.

예를 들어, 다음 CLI 명령은 SQL 트랜잭션을 시작합니다.

Linux, OS X, Unix의 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"
```

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 포함한 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-metadata입니다.
- --database (선택 사항) – 데이터베이스 이름.
- --continue-after-timeout | --no-continue-after-timeout (선택 사항) – 호출 시간이 초과된 후에 문을 계속 실행할지 나타내는 값. 기본값은 --no-continue-after-timeout입니다.

데이터 정의 언어(DDL) 문에 대해서는 오류 및 데이터 구조 손상 가능성은 피하기 위해 호출 시간이 초과된 후 문을 계속 실행하는 것이 좋습니다.

DB 클러스터는 각 호출에 대한 응답을 반환합니다.

Note

응답 크기 제한은 1MB입니다. 호출이 1MB를 초과하는 응답 데이터를 반환하면 호출이 종료됩니다.

예를 들어, 다음 CLI 명령은 단일 SQL 문을 실행하고 결과에서 메타데이터를 생략합니다(기본값).

Linux, OS X, Unix의 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"
```

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.

Linux, OS X, Unix의 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"
```

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.

Linux, OS X, Unix의 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\\\"}}]"
```

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. 호출 시간이 초과된 후 문을 계속 실행하려면 --continue-after-timeout 옵션을 지정하십시오.

Linux, OS X, Unix의 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
```

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.
- --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

응답 크기 제한은 1MB입니다. 호출이 1MB를 초과하는 응답 데이터를 반환하면 호출이 종료됩니다.

예를 들어, 다음 CLI 명령은 파라미터 세트를 이용해 데이터 배열에 대해 배치 SQL 문을 실행합니다.

Linux, OS X, Unix의 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"}}]]"
```

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-transaction으로 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.

Linux, OS X, Unix의 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"
```

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 룰백

aws rds-data rollback-transaction CLI 명령을 사용하여 aws rds-data begin-transaction으로 시작한 SQL 트랜잭션을 룰백할 수 있습니다. 트랜잭션을 룰백하면 변경 내용이 취소됩니다.

Important

트랜잭션 ID가 만기되었다면 트랜잭션이 자동으로 룰백된 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. 룰백하려는 트랜잭션의 트랜잭션 ID를 지정하십시오.

예를 들어 다음 AWS CLI 명령은 SQL 트랜잭션을 룰백합니다.

Linux, OS X, Unix의 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"
```

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 데이터 API 호출

Python 애플리케이션에서 데이터 API를 호출할 수 있습니다.

다음 예제에서는 Python용 AWS SDK(Boto)를 사용합니다. Boto에 대한 자세한 내용은 [Python용 AWS SDK\(Boto 3\) 설명서](#)를 참조하십시오.

각 예에서 DB 클러스터의 Amazon 리소스 이름(ARN)을 Aurora Serverless DB 클러스터의 ARN으로 바꿉니다. 또한 DB 클러스터에 대한 액세스를 허용하는 AWS Secrets Manager의 암호 ARN으로 암호 ARN을 바꿉니다.

주제

- [SQL 쿼리 실행 \(p. 149\)](#)
- [DML SQL 문 실행 \(p. 150\)](#)

- SQL 트랜잭션 실행 (p. 150)

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 끝납니다.

트랜잭션 ID를 지정하지 않으면 호출 결과가 자동으로 커밋됩니다.

다음 예제에서는 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. 호출 시간이 초과된 후 문을 계속 실행하려면 `continueAfterTimeout` 파라미터를 `true`로 설정하십시오.

Java 애플리케이션에서 데이터 API 호출

Java 애플리케이션에서 데이터 API를 호출할 수 있습니다.

다음 예제는 Java용 AWS SDK를 사용합니다. 자세한 내용은 [AWS SDK for Java Developer Guide](#) 단원을 참조하십시오.

각 예에서 DB 클러스터의 Amazon 리소스 이름(ARN)을 Aurora Serverless DB 클러스터의 ARN으로 바꿉니다. 또한 DB 클러스터에 대한 액세스를 허용하는 AWS Secrets Manager의 암호 ARN으로 암호 ARN을 바꿉니다.

주제

- [SQL 쿼리 실행 \(p. 151\)](#)
- [SQL 트랜잭션 실행 \(p. 152\)](#)
- [일괄 SQL 작업 실행 \(p. 153\)](#)

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. 호출 시간이 초과된 후 문을 계속 실행하려면 `continueAfterTimeout` 파라미터를 `true`로 설정하십시오.

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 API용 Java 클라이언트 라이브러리 사용(미리 보기)

이 시험판 설명서는 프리뷰 버전 서비스에 관한 것입니다. 변경될 수 있습니다.

데이터 API용 Java 클라이언트 라이브러리를 다운로드하여 사용할 수 있습니다. Java 클라이언트 라이브러리는 데이터 API를 사용할 수 있는 다른 방법을 제공합니다. 이 라이브러리를 사용해 클라이언트 측 클래스를 데이터 API의 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 추가하십시오.

```
<dependency>
<groupId>software.amazon.rdsdata</groupId>
<artifactId>rds-data-api-client-library-java</artifactId>
<version>1.0.1</version>
</dependency>
```

Java 클라이언트 라이브러리 예시

아래에는 데이터 API Java 클라이언트 라이브러리 사용에 관한 몇 가지 일반적인 예시가 나와 있습니다. 이 예시에서는 accountId와 balance라는 두 개의 열이 있는 accounts라는 테이블이 있다고 가정합니다. 또한 다음과 같은 데이터 전송 객체(DTO)도 있습니다.

```
public class Account {  
    String accountId;  
    double balance;  
    // 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("A-1", 1.1);  
var account2 = new Account("B-2", 100);  
client.forSql("INSERT INTO accounts(accountId, balance) VALUES(:accountId, :balance)")  
    .withParams(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, balance) VALUES(:accountId, :balance)")  
    .withParam("accountId", "A-1")  
    .withParam("balance", 12.2)  
    .execute();
```

다음은 입력 파라미터인 간단한 값으로 작업하는 또 다른 예시입니다.

```
client.forSql("INSERT INTO accounts(accountId, balance) VALUES(?, ?, "A-1", 12.2)  
    .execute();
```

클라이언트 라이브러리에서는 실행 결과가 반환될 때 DTO에 대한 자동 매핑을 제공합니다. 다음 예시에서는 실행 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);
```

데이터 API 문제 해결

공통 오류 메시지를 소개하는 다음 섹션부터는 데이터 API를 사용하면서 발생하는 문제를 해결하는 데 유용합니다.

Note

데이터 API와 관련하여 궁금한 사항이나 의견이 있으면 rds-data-api-feedback@amazon.com으로 이메일을 보내주십시오.

주제

- 트랜잭션 <transaction_ID>을 찾을 수 없습니다 (p. 156)
- 쿼리 패킷이 너무 큽니다 (p. 156)
- 데이터베이스 응답이 크기 제한을 초과했습니다 (p. 156)
- HttpEndpoint가 클러스터 <cluster_ID>에서 활성화되지 않았습니다 (p. 156)

트랜잭션 <transaction_ID>을 찾을 수 없습니다

위 오류 메시지는 데이터 API 호출 시 지정한 트랜잭션 ID를 찾을 수 없다는 것을 의미합니다. 이러한 문제가 발생하는 원인은 대부분 다음 중 한 가지입니다.

- 지정된 트랜잭션 ID가 [BeginTransaction](#) 호출에서 생성되지 않았습니다.
- 지정된 트랜잭션 ID가 만료되었습니다.

어떤 호출에서도 트랜잭션 ID를 3분 동안 사용하지 않으면 트랜잭션이 만료됩니다.

이러한 문제를 해결하려면 호출 시 유효한 트랜잭션 ID를 지정해야 합니다. 또한 각 트랜잭션 호출마다 마지막 호출 이후 3분 이내에 실행되어야 합니다.

트랜잭션 실행에 대한 자세한 내용은 [데이터 API 호출 \(p. 139\)](#) 단원을 참조하십시오.

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. 데이터 API는 데이터베이스에서 반환되는 결과 집합의 크기를 1MB로 제한합니다.

이러한 문제를 해결하려면 데이터 API 호출 시 반환되는 데이터 크기를 1MB 이하로 유지해야 합니다. 반환해야 하는 결과 집합의 크기가 1MB보다 높으면 쿼리에서 다수의 [ExecuteStatement](#) 호출을 LIMIT 절과 함께 사용할 수 있습니다.

LIMIT 절에 대한 자세한 내용은 MySQL 설명서에서 [SELECT 문](#)을 참조하십시오.

HttpEndpoint가 클러스터 <cluster_ID>에서 활성화되지 않았습니다

이러한 문제가 발생하는 원인은 대부분 다음 중 한 가지입니다.

- Aurora Serverless DB 클러스터에서 데이터 API가 활성화되어 있지 않습니다. Aurora Serverless DB 클러스터에서 데이터 API를 사용하려면 DB 클러스터에서 데이터 API를 먼저 활성화해야 합니다.
- 데이터 API가 활성화된 후 DB 클러스터의 이름이 바뀌었습니다.

DB 클러스터에서 데이터 API가 활성화되지 않았다면 활성화하십시오.

DB 클러스터에서 데이터 API가 활성화된 후 DB 클러스터 이름이 바뀌었다면 데이터 API를 비활성화한 후 다시 활성화하십시오.

데이터 API 활성화에 대한 자세한 내용은 [데이터 API 활성화 \(p. 135\)](#) 단원을 참조하십시오.

AWS CloudTrail을 사용하여 데이터 API 호출 로깅

데이터 API는 데이터 API의 사용자, 역할 또는 AWS 서비스가 수행한 작업을 기록하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 Amazon RDS 콘솔과 코드에서 데이터 API 작업으로의 호출을 포함해 데이터 API의 모든 API 호출을 이벤트로 캡처합니다. 추적을 생성하면 데이터 API 이벤트를 비롯하여 CloudTrail 이벤트를 Amazon S3 버킷으로 지속적으로 배포하도록 할 수 있습니다. 추적을 구성하지 않은 경우 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수도 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 User Guide](#)를 참조하십시오.

CloudTrail의 데이터 API 정보 작업

CloudTrail은 계정 생성 시 AWS 계정에서 활성화됩니다. 데이터 API에서 활동이 수행되면 해당 활동은 Event history(이벤트 기록)에서 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 AWS CloudTrail User Guide의 [CloudTrail 이벤트 기록을 사용하여 이벤트 보기](#)를 참조하십시오.

데이터 API의 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려는 경우 추적을 생성합니다. 추적은 CloudTrail이 Amazon S3 버킷으로 로그 파일을 전송할 수 있도록 합니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 AWS 리전의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 AWS CloudTrail User Guide에서 다음 주제를 참조하십시오.

- [추적 생성 개요](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에 대한 Amazon SNS 알림 구성](#)
- [여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기](#)

모든 데이터 API 작업은 CloudTrail이 기록하고 [Amazon RDS 데이터 서비스 API 참조](#)에 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. 자격 증명 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 자격 증명으로 했는지 여부
- 역할 또는 연합된 사용자에 대한 임시 보안 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 [CloudTrail userIdentity 요소](#)를 참조하십시오.

데이터 API 로그 파일 항목 이해

추적은 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 제공할 수 있는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함됩니다. 이벤트는 어떤 소스로부터의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 포함되어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 호출의 주문 스택 추적이 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 ExecuteStatement 작업을 보여주는 CloudTrail 로그 항목입니다.

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AKIAIOSFODNN7EXAMPLE",  
        "arn": "arn:aws:iam::123456789012:user/john Doe",  
        "accountId": "123456789012",  
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",  
        "userName": "john Doe"  
    },  
    "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"  
}
```

Aurora Serverless에 쿼리 편집기 사용

Aurora Serverless용 쿼리 편집기를 사용하면 RDS 콘솔에서 SQL 쿼리를 실행할 수 있습니다. Aurora Serverless DB 클러스터에서 데이터 조작 및 데이터 정의 문을 포함하여 유효한 SQL 문을 모두 실행할 수 있습니다.

쿼리 편집기에 데이터 API가 활성화된 Aurora Serverless DB 클러스터가 필요합니다. 데이터 API가 활성화된 Aurora Serverless DB 클러스터 생성에 대한 자세한 내용은 [Aurora Serverless에 데이터 API 사용 \(p. 134\)](#) 단원을 참조하십시오.

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.

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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "QueryEditor0",  
            "Effect": "Allow",  
            "Action": "rdsdata:ExecuteStatement",  
            "Resource": "arn:aws:rds:  
us-east-1:123456789012:cluster:my-database-1:  
executeStatement"  
        }  
    ]  
}
```

```
        "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 생성](#)을 참조하십시오.

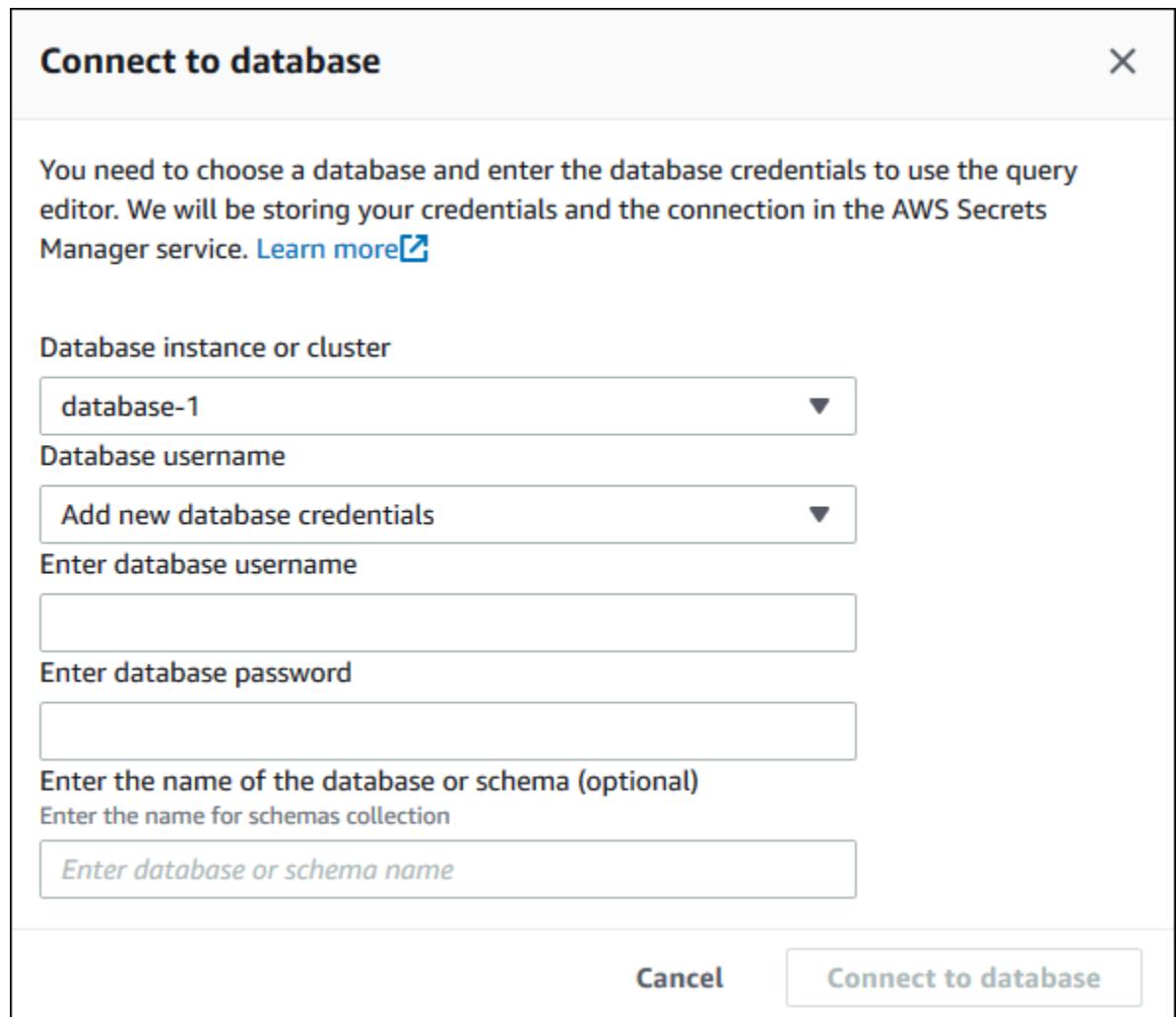
사용자에게 IAM 정책 추가에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [IAM 자격 증명 권한 추가 및 제거](#)를 참조하십시오.

쿼리 편집기에서 쿼리 실행

쿼리 편집기에서 Aurora Serverless DB 클러스터에 대해 SQL 문을 실행할 수 있습니다.

쿼리 편집기에서 쿼리를 실행하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management 콘솔의 상단 오른쪽에서 쿼리할 Aurora Serverless DB 클러스터를 생성한 AWS 리전을 선택합니다.
3. 탐색 창에서 데이터베이스를 선택합니다.
4. SQL 쿼리를 실행할 Aurora Serverless DB 클러스터를 선택합니다.
5. 작업에서 쿼리를 선택합니다. 이전에 데이터베이스에 연결하지 않은 경우 데이터베이스에 연결 페이지가 열립니다.



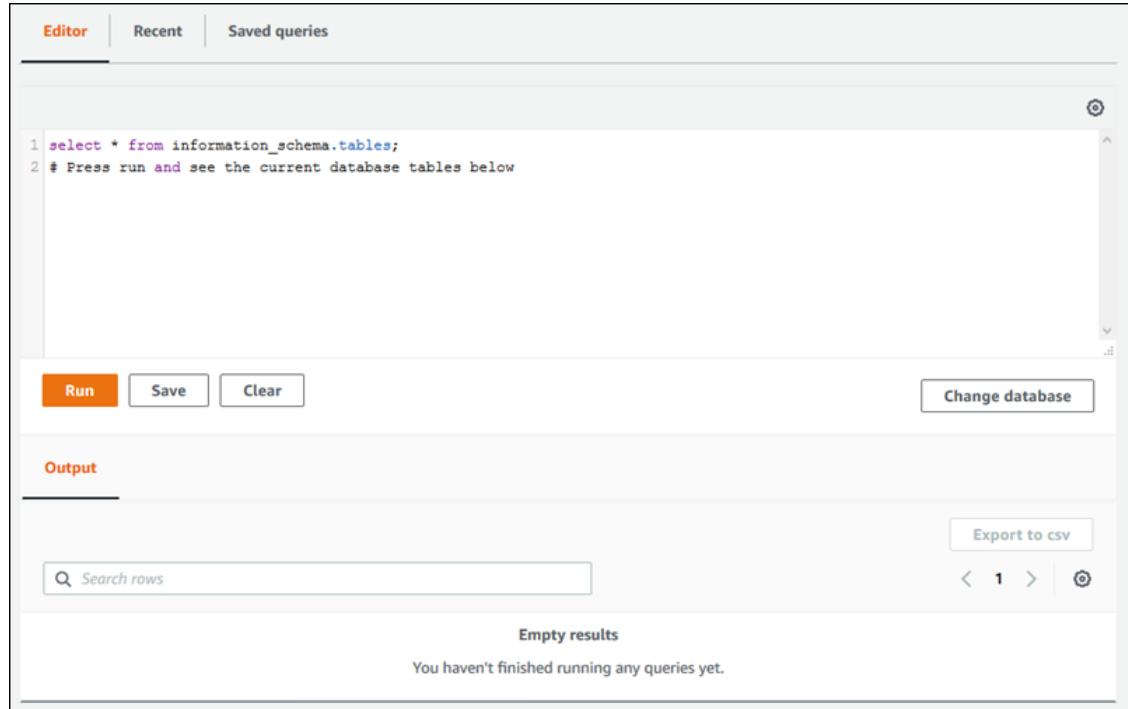
6. 다음 정보를 입력합니다.
 - a. 데이터베이스 인스턴스 또는 클러스터에서 SQL 쿼리를 실행할 Aurora Serverless DB 클러스터를 선택합니다.
 - b. Database username(데이터베이스 사용자 이름)에서 연결할 데이터베이스 사용자의 사용자 이름을 선택하거나 Add new database credentials(새 데이터베이스 자격 증명 추가)를 선택합니다. Add new database credentials(새 데이터베이스 자격 증명 추가)를 선택한 경우 Enter database username(데이터베이스 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 연결을 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.

Amazon Aurora Aurora 사용 설명서 쿼리 편집기 사용



```
1 select * from information_schema.tables;
2 # Press run and see the current database tables below
```

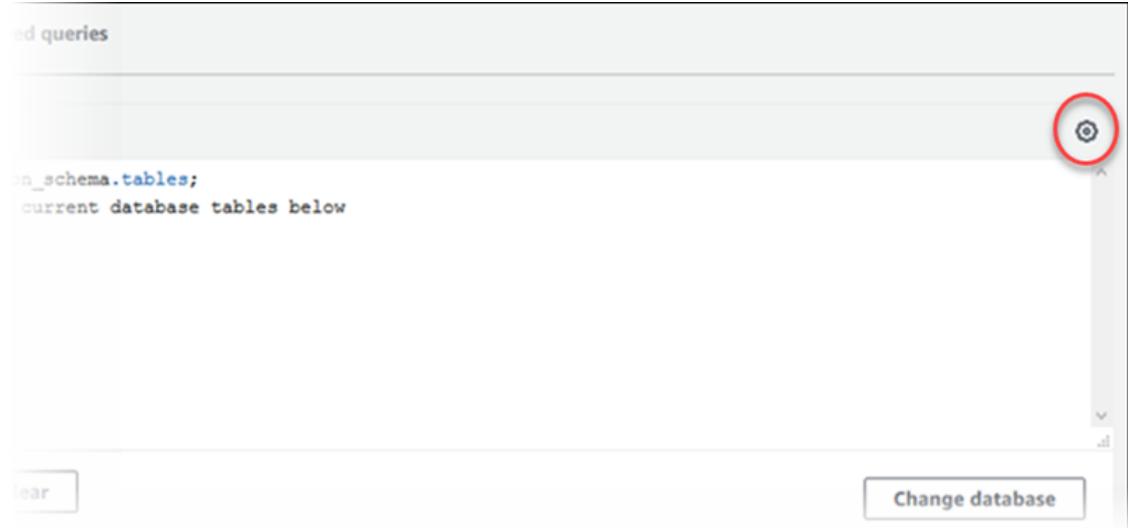
Run Save Clear Change database

Output

Search rows Export to csv < 1 > ⌂

Empty results
You haven't finished running any queries yet.

각 SQL 문은 자동으로 커밋되거나, 트랜잭션의 일부로 스크립트에서 SQL 문을 실행할 수 있습니다. 이 동작을 제어하려면 쿼리 창 위의 기어 모양 아이콘을 선택하십시오.

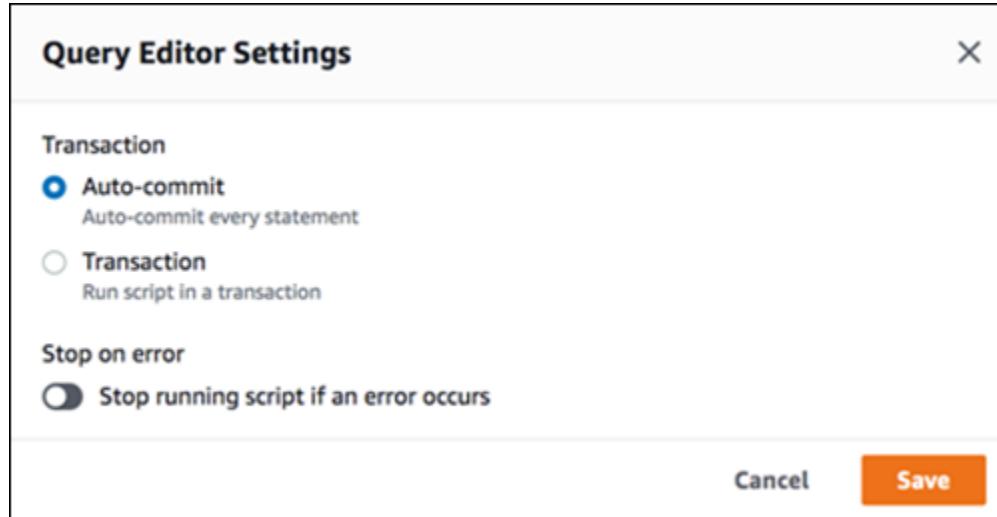


ed queries

```
1 select * from information_schema.tables;
2 current database tables below
```

Clear Change database

Query Editor Settings(쿼리 편집기 설정) 창이 나타납니다.



Auto-commit(자동 커밋)을 선택하면 모든 SQL 문이 자동으로 커밋됩니다. Transaction(트랜잭션)을 선택하면 스크립트에서 문 그룹을 실행할 수 있으며, 자동으로 커밋되지 않습니다. Transaction(트랜잭션)을 설정할 경우 실행을 선택하면 그룹의 문이 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 Change database(데이터베이스 변경)를 선택하십시오.

Amazon Aurora DB 클러스터 연결

MySQL 또는 PostgreSQL 데이터베이스에 연결할 때 사용 한 것과 동일한 도구를 사용하여 Aurora DB 클러스터에 연결할 수 있습니다. MySQL 또는 PostgreSQL DB 인스턴스에 연결되는 모든 스크립트, 유ти리티 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.

Amazon Aurora MySQL DB 클러스터 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 [사용자 계정 관리](#)를 참조하십시오. IAM 데이터베이스 인증 사용에 대한 자세한 정보는 [을 위한 IAM 데이터베이스 인증 \(p. 976\)](#) 단원을 참조하십시오.

MySQL 5.6과 호환되는 Amazon Aurora DB 클러스터에 연결되어 있으면 MySQL 버전 5.6과 호환되는 SQL 명령을 실행할 수 있습니다. MySQL 5.6 SQL 구문에 대한 자세한 정보는 [MySQL 5.6 참조 매뉴얼](#)을 참조하십시오.

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 매뉴얼](#)을 참조하십시오. Aurora MySQL 5.7에 적용되는 제한 사항에 대한 자세한 정보는 [Aurora MySQL 5.7과 MySQL 5.7 비교 \(p. 465\)](#)을 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 3306를 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.

Note

Aurora Serverless DB 클러스터인 경우 데이터베이스 엔드포인트에만 연결할 수 있습니다. 자세한 정보는 [사용 Amazon Aurora Serverless \(p. 111\)](#) 단원을 참조하십시오.

클러스터 엔드포인트(라이터 엔드포인트)를 보려면 Amazon RDS 콘솔에서 Databases(데이터베이스)를 선택하고 DB 클러스터의 이름을 선택하여 DB 클러스터 세부 정보를 표시하십시오.

The screenshot shows the AWS RDS console with the following details:

- Path:** RDS > Databases > aurora-cl-mysql
- Database Name:** aurora-cl-mysql
- Related:** Shows the DB identifier "aurora-cl-mysql" and its three instances: dbinstance4 (Writer), dbinstance1 (Reader), and dbinstance2 (Reader).
- Connectivity & security:** This tab is selected.
- Endpoints (2):**
 - Endpoint name: aurora-cl-mysql.cluster-ro.us-east-1.rds.amazonaws.com (Status: Available, Type: Reader, Port: 3306)
 - Endpoint name: aurora-cl-mysql.cluster.us-east-1.rds.amazonaws.com (Status: Available, Type: Writer, Port: 3306) - This endpoint is highlighted with a red border.

Aurora MySQL 연결 유ти리티

다음과 같은 연결 유ти리티를 사용할 수 있습니다.

- 명령줄 – MySQL 명령줄 유ти리티 같은 도구를 사용하여 Amazon Aurora DB 클러스터에 연결할 수 있습니다. MySQL 유ти리티 사용에 대한 자세한 정보는 MySQL 문서의 [mysql - The MySQL 명령줄 도구](#)를 참조하십시오.
- GUI – MySQL Workbench 유ти리티를 통해 UI 인터페이스를 사용하여 연결할 수 있습니다. 자세한 정보는 [MySQL Workbench 다운로드](#) 페이지 단원을 참조하십시오.
- 애플리케이션 – MariaDB Connector/J 유ти리티를 사용하여 Aurora DB 클러스터에 애플리케이션을 연결할 수 있습니다. 자세한 정보는 [MariaDB Connector/J 다운로드](#) 페이지 단원을 참조하십시오.

Note

MariaDB Connector/J 유ти리티를 Aurora Serverless 클러스터와 함께 사용하는 경우에는 연결 문자열에서 접두사 `jdbc:mariadb:aurora://`를 사용하십시오. `mariadb:aurora` 파라미터는

장애 조치 대상에 대한 자동 DNS 스캔을 건너뜁니다. 이 스캔 작업은 Aurora Serverless 클러스터에 필요 없기 때문에 연결을 구성할 때 시간을 지연시킵니다.

Amazon Aurora DB 인스턴스에 연결할 때는 SSL 암호화를 사용할 수 있습니다. 자세한 내용은 [MySQL DB 인스턴스와 함께 SSL 사용](#)을 참조하십시오.

Note

Amazon Aurora DB 클러스터는 Amazon Virtual Private Cloud(VPC)에서만 생성할 수 있으므로 Amazon Aurora DB 클러스터의 퍼블릭 엔드포인트 주소를 사용하려면 VPC에 없는 AWS 인스턴스의 Amazon Aurora DB 클러스터에 연결해야 합니다. 하지만 이제는 ClassicLink를 사용하여 VPC에 없는 Amazon EC2 인스턴스 및 Amazon Aurora DB 클러스터와 통신할 수 있습니다. 자세한 정보는 [VPC에 있지 않은 EC2 인스턴스가 VPC에 있는 DB 인스턴스에 액세스 \(p. 1014\)](#) 단원을 참조하십시오.

Aurora MySQL용 SSL과 연결

SSL을 사용하여 연결하려면 다음 절차에 따라 MySQL 유ти리티를 사용합니다. IAM 데이터베이스 인증을 사용하고 있다면, SSL 연결을 사용해야 합니다. 자세한 정보는 [을 위한 IAM 데이터베이스 인증 \(p. 976\)](#) 단원을 참조하십시오.

Note

SSL을 사용하여 클러스터 엔드포인트에 연결하려면 클라이언트 연결 유ти리티에서 Subject Alternative Names(SAN)를 지원해야 합니다. 클라이언트 연결 유ти리티에서 SAN을 지원하지 않는 경우, Aurora DB 클러스터에서 인스턴스에 직접 연결할 수 있습니다. Aurora 엔드포인트에 대한 자세한 정보는 [Amazon Aurora 연결 관리 \(p. 9\)](#) 단원을 참조하십시오.

MySQL 유ти리티를 사용하여 SSL에 DB 클러스터 연결하는 방법

1. Amazon RDS 서명 인증서의 퍼블릭 키를 다운로드합니다.

인증서 다운로드에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#) 단원을 참조하십시오.

2. MySQL 유ти리티에서 SSL이 포함된 DB 클러스터의 기본 인스턴스에 연결하려면 명령 프롬프트에 다음 명령을 입력합니다. -h 파라미터의 경우 기본 인스턴스의 엔드포인트 DNS 이름으로 대체합니다. --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 --ssl-ca=[full path]rds-combined-ca-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: 5.6.10-log MySQL Community Server (GPL)

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

Amazon RDS MySQL 연결 문자열 구성 및 SSL 연결용 퍼블릭 키 검색에 대한 일반적인 지침은 [MySQL 데이터베이스 엔진 기반 DB 인스턴스에 연결하기](#)를 참조하십시오.

Amazon Aurora PostgreSQL DB 클러스터 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 앤드포인트 및 포트 정보를 PostgreSQL DB 인스턴스에 연결하는 모든 스크립트, 유ти리티 또는 애플리케이션의 연결 문자열에 사용할 수 있습니다. 연결 문자열에는 호스트 파라미터로 기본 인스턴스 또는 Aurora 복제본 앤드포인트의 DNS 주소를 지정하십시오. 앤드포인트의 포트 번호를 포트 파라미터로 지정하십시오.

Amazon Aurora PostgreSQL DB 클러스터의 DB 인스턴스에 연결한 경우 PostgreSQL 버전 9.6.3과 호환되는 SQL 명령을 실행할 수 있습니다.

Aurora PostgreSQL DB 클러스터의 세부 정보 보기에서 클러스터 앤드포인트를 검색할 수 있습니다. PostgreSQL 연결 문자열에서 이 앤드포인트를 사용합니다. 앤드포인트는 DB 클러스터의 도메인 이름과 포트로 구성되어 있습니다. 예를 들어 앤드포인트 값이 `mycluster.cluster-123456789012.us-east-1.rds.amazonaws.com:5432`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 5432를 지정합니다

클러스터 앤드포인트는 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 앤드포인트와 무관한 고유한 앤드포인트를 가집니다. 이 고유 앤드포인트를 통해 클러스터의 특정 DB 인스턴스에 직접 연결할 수 있습니다. 클러스터 앤드포인트는 항상 기본 인스턴스를 가리킵니다. 기본 인스턴스에 결함이 발생하여 교체되면 클러스터 앤드포인트는 새로운 기본 인스턴스로 향하게 됩니다.

클러스터 앤드포인트(라이터 앤드포인트)를 보려면 Amazon RDS 콘솔에서 Databases(데이터베이스)를 선택하고 DB 클러스터의 이름을 선택하여 DB 클러스터 세부 정보를 표시하십시오.

The screenshot shows the AWS RDS console for managing databases. The main navigation bar at the top includes 'RDS', 'Databases', and the specific database name 'aurora-cl-postgresql'. On the right side of the top bar are 'Modify' and 'Actions' buttons. Below the navigation, the database name 'aurora-cl-postgresql' is displayed in a large, bold font. To the right of the database name are 'Modify' and 'Actions' buttons. A 'Related' section is present, followed by a search bar labeled 'Filter databases'. The main content area displays a table of database instances. The columns include 'DB identifier', 'Role', 'Engine', 'Region & AZ', and 'Size'. The table shows three entries: 'aurora-cl-postgresql' (Regional, Aurora PostgreSQL, us-east-1, 2 instances), 'aurora-cl-postgresql-instance-1' (Writer, Aurora PostgreSQL, us-east-1a, db.r5.large), and 'aurora-cl-postgresql-instance-1-us-east-1b' (Reader, Aurora PostgreSQL, us-east-1b, db.r5.large). Below the table are tabs for 'Connectivity & security' (which is selected), 'Monitoring', 'Logs & events', 'Configuration', 'Maintenance & backups', and 'Tags'. The 'Endpoints' section is highlighted, showing two endpoints: 'aurora-cl-postgresql.cluster-ro...' (us-east-1.rds.amazonaws.com) and 'aurora-cl-postgresql.cluster...' (us-east-1.rds.amazonaws.com). The 'aurora-cl-postgresql.cluster...' endpoint is highlighted with a red border. Below the endpoints is a 'Manage IAM roles' section.

Aurora PostgreSQL 연결 유ти리티

다음과 같은 연결 유ти리티를 사용할 수 있습니다.

- 명령줄 – PostgreSQL 대화식 터미널인 psql과 같은 도구를 사용하여 Amazon Aurora PostgreSQL DB 인스턴스에 연결할 수 있습니다. PostgreSQL 대화식 터미널 사용에 대한 자세한 정보는 PostgreSQL 설명서에서 [psql](#)을 참조하십시오.
- GUI – pgAdmin 유ти리티를 사용하여 UI 인터페이스로 PostgreSQL DB 인스턴스에 연결할 수 있습니다. 자세한 정보는 pgAdmin 웹사이트에서 [Download](#) 페이지 단원을 참조하십시오.
- 애플리케이션 – PostgreSQL JDBC 드라이버를 사용하여 애플리케이션을 PostgreSQL DB 인스턴스에 연결할 수 있습니다. 자세한 정보는 PostgreSQL JDBC 드라이버 웹사이트에서 [Download](#) 페이지 단원을 참조하십시오.

Note

Amazon Aurora PostgreSQL DB 클러스터는 Amazon Virtual Private Cloud(VPC)에서만 생성할 수 있으므로 Aurora PostgreSQL DB 클러스터의 퍼블릭 엔드포인트 주소를 사용하려면 VPC에

없는 AWS 인스턴스의 Aurora PostgreSQL DB 클러스터에 연결해야 합니다. 하지만 이제는 ClassicLink를 사용하여 VPC에 없는 Amazon EC2 인스턴스 및 Aurora PostgreSQL DB 클러스터와 통신할 수 있습니다. 자세한 정보는 [VPC에 있지 않은 EC2 인스턴스가 VPC에 있는 DB 인스턴스에 액세스 \(p. 1014\)](#) 단원을 참조하십시오.

Aurora 연결 장애 문제 해결

Note

Amazon Aurora MySQL DB 클러스터에 연결하는 데 도움이 되는 자세한 안내는 [Aurora 연결 관리](#) 핸드북에서 확인할 수 있습니다.

새 Aurora DB 클래스에 공통적으로 발생하는 연결 실패의 원인은 다음과 같습니다.

- DB 클러스터가 디바이스에서 연결할 수 없는 VPC를 사용하여 생성되었습니다. 이 문제를 해결하려면 디바이스에서 연결할 수 있는 VPC로 설정을 변경하거나, 디바이스에서 연결할 수 있는 VPC를 DB 클러스터에 새로 생성해야 합니다. 문제 해결 예는 [VPC 및 서브넷 생성 \(p. 1006\)](#) 단원을 참조하십시오.
- 기본 포트를 사용해 DB 클러스터를 생성했는데 기업 방화벽 규칙에 따라 기업 네트워크의 디바이스에서 해당 포트에 연결하는 것이 차단되었습니다. 이 오류를 수정하려면 인스턴스를 다른 포트로 다시 만들어야 합니다.
- IAM 데이터베이스 인증을 사용하고 있다면, IAM 데이터베이스 인증을 구성해야 합니다. 자세한 정보는 [을 위한 IAM 데이터베이스 인증 \(p. 976\)](#) 단원을 참조하십시오.

DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업

DB 인스턴스 및 Aurora 클러스터를 파라미터 그룹과 연결하여 DB 엔진 구성을 관리합니다. Amazon RDS는 새로 생성된 DB 인스턴스 및 Aurora 클러스터에 적용되는 기본 설정으로 파라미터 그룹을 정의합니다. 맞춤형 설정으로 자신만의 파라미터 그룹을 정의할 수 있습니다. 그런 다음 DB 인스턴스 및 Aurora 클러스터를 수정해 이 파라미터 그룹을 사용하게 할 수 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.

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 그룹을 지정하지 않고 DB 인스턴스를 만드는 경우 DB 인스턴스에서는 기본 DB 파라미터 그룹을 사용합니다. 이와 마찬가지로 DB 클러스터 파라미터 그룹을 지정하지 않고 Aurora DB 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. 그 대신에 자신만의 파라미터 설정을 선택하는 경우 자신만의 파라미터 그룹을 생성합니다. 고객님이 생성하는 파라미터 그룹에서 모든 DB 엔진 파라미터를 변경할 수 있는 것은 아닙니다.

자신만의 파라미터 그룹을 사용하고 싶다면 새 파라미터 그룹을 생성하고 사용하려는 파라미터를 수정하십시오. 그런 다음 DB 인스턴스 또는 DB 클러스터를 수정하여 새로운 파라미터 그룹을 사용하십시오. DB 파라미터 그룹 내의 파라미터를 업데이트하는 경우 변경 사항은 이 파라미터 그룹과 연결된 모든 DB 인스턴스에

적용됩니다. 이와 마찬가지로 DB 클러스터 파라미터 그룹 내의 파라미터를 업데이트하는 경우 변경 사항은 이 DB 클러스터 파라미터 그룹과 연결된 모든 Aurora 클러스터에 적용됩니다.

AWS CLI [rds-copy-db-parameter-group](#) 명령으로 기존 DB 파라미터 그룹을 복사할 수 있습니다. AWS CLI [copy-db-cluster-parameter-group](#) 명령으로 기존 DB 클러스터 파라미터 그룹을 복사할 수 있습니다. 기존 파라미터 그룹의 사용자 지정 파라미터 및 값 대부분을 새로운 파라미터 그룹에 포함할 때는 파라미터 그룹을 복사하는 것이 편리할 수 있습니다.

다음은 파라미터 그룹의 파라미터를 사용한 작업에 관한 몇 가지 주요 사항입니다.

- 동적 파라미터를 변경하고 파라미터 그룹을 저장하면 즉시 적용 설정에 관계없이 변경 내용이 바로 적용됩니다. 고정 파라미터를 변경하고 DB 파라미터 그룹을 저장한 후 DB 인스턴스를 수동으로 재부팅하면 파라미터 변경 내용이 적용됩니다. RDS 콘솔을 사용하거나 명시적으로 `RebootDbInstance` API 작업을 호출하여 DB 인스턴스를 재부팅할 수 있습니다(DB 인스턴스가 다른 AZ 배포에 있는 경우 장애 조치 없음). 고정 파라미터 변경 후 연결된 DB 인스턴스를 재부팅하도록 하면 `ModifyDBInstance`를 호출하여 DB 인스턴스 클래스를 변경하거나 스토리지를 조정하는 경우와 같이 잘못된 파라미터 구성이 API 호출에 영향을 주는 위험을 완화할 수 있습니다.

DB 인스턴스에서 연결된 DB 파라미터 그룹에 대한 최신 변경 내용을 사용하고 있지 않은 경우 AWS Management 콘솔에 DB 파라미터 그룹이 재시작 보류 중 상태로 표시됩니다. 재시작 보류 중 파라미터 그룹 상태로 인해 다음 번 유지 관리 기간 중에 자동 재부팅이 되지는 않습니다. 최신 파라미터 변경 내용을 이 DB 인스턴스에 적용하려면 해당 DB 인스턴스를 수동으로 재부팅해야 합니다.

- DB 인스턴스와 연결된 DB 파라미터 그룹을 변경하면 DB 인스턴스에서 새 DB 파라미터 그룹을 사용하기 전에 인스턴스를 수동으로 재부팅해야 합니다.

Note

DB 클러스터에 연결된 DB 클러스터 파라미터 그룹을 변경할 때는 재부팅하지 않아도 됩니다.

- 파라미터에 대해 수식, 변수, 함수 및 연산자로 구성된 정수 또는 정수 식으로 지정할 수 있습니다. 함수에 수학 로그식을 넣을 수 있습니다. 자세한 내용은 [DB 파라미터 값 \(p. 186\)](#) 단원을 참조하십시오.
- DB 인스턴스를 만들기 전 및 DB 인스턴스에 데이터베이스를 만들기 전에, 파라미터 그룹에 있는 데이터베이스의 문자 세트 또는 콜레이션과 관련된 파라미터를 모두 설정해야 합니다. 이렇게 하면 DB 인스턴스의 기본 데이터베이스와 새 데이터베이스가 지정한 문자 세트 및 데이터 정렬 값을 사용하게 됩니다. DB 인스턴스의 문자 세트 또는 데이터 정렬 파라미터를 변경해도 기존 데이터베이스에는 변경된 파라미터가 적용되지 않습니다.

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;
```

- 파라미터 그룹에 파라미터를 잘못 설정하면 성능 저하나 시스템 불안정 등의 의도하지 않은 부작용이 있을 수 있습니다. 데이터베이스 파라미터를 수정할 때 항상 주의를 기울이고 파라미터 그룹을 수정하기 전에 데이터를 백업하십시오. 파라미터 그룹 변경 내용을 프로덕션 DB 인스턴스에 적용하기 전에 테스트 DB 인스턴스에 적용해 봐야 합니다.
- Aurora 글로벌 데이터베이스의 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 글로벌 데이터베이스의 모든 클러스터 간에 시간대와 문자 세트에 대해 동일한 설정을 사용합니다.
- DB 엔진에서 지원되는 파라미터를 확인하기 위해 DB 클러스터에서 사용되는 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹에서 파라미터를 확인할 수 있습니다. 자세한 내용은 [DB 파라미터 그룹의 파라미터 값 보기 \(p. 183\)](#) 및 [DB 클러스터 파라미터 그룹의 파라미터 값 보기 \(p. 184\)](#) 단원을 참조하십시오.

주제

- [Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 \(p. 170\)](#)

- [DB 파라미터 그룹 생성 \(p. 171\)](#)
- [DB 클러스터 파라미터 그룹 만들기 \(p. 172\)](#)
- [DB 파라미터 그룹의 파라미터 수정 \(p. 174\)](#)
- [DB 클러스터 파라미터 그룹의 파라미터 수정 \(p. 177\)](#)
- [DB 파라미터 그룹 복사 \(p. 179\)](#)
- [DB 클러스터 파라미터 그룹 복사 \(p. 180\)](#)
- [DB 파라미터 그룹 나열 \(p. 181\)](#)
- [DB 클러스터 파라미터 그룹 나열 \(p. 182\)](#)
- [DB 파라미터 그룹의 파라미터 값 보기 \(p. 183\)](#)
- [DB 클러스터 파라미터 그룹의 파라미터 값 보기 \(p. 184\)](#)
- [파라미터 그룹 비교 \(p. 186\)](#)
- [DB 파라미터 값 \(p. 186\)](#)

Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터

Aurora에서는 다음과 같이 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 인스턴스로 포함하는 경우가 많습니다.

모든 Aurora 클러스터는 DB 클러스터 파라미터 그룹과 연결됩니다. 클러스터 내 각 DB 인스턴스는 DB 클러스터 파라미터 그룹에서 설정을 상속하며, DB 파라미터 그룹과 연결됩니다. Aurora는 고객님이 클러스터 또는 새로운 DB 인스턴스를 생성할 때 지정된 데이터베이스 엔진 및 버전에 근거하여 기본 파라미터 그룹을 배정합니다. 파라미터 그룹은 나중에 고객님이 생성하는 것으로 변경할 수 있습니다. 이 경우 파라미터 값을 편집할 수 있습니다.

DB 클러스터 파라미터 그룹에는 DB 파라미터 그룹에 속하는 모든 인스턴스 수준 파라미터의 기본값도 들어 있습니다. 이 기본값은 주로 Aurora Serverless 클러스터를 구성하기 위한 것입니다. 이 클러스터는 DB 파라미터 그룹이 아닌 DB 클러스터 파라미터 그룹에만 연결됩니다. DB 클러스터 파라미터 그룹에서 인스턴스 수준 파라미터 설정을 수정할 수 있습니다. 그런 다음 Aurora는 이 설정을 서비스 클러스터에 추가된 새 DB 인스턴스에 적용합니다. Aurora Serverless 클러스터에 대한 구성 설정과 수정할 수 있는 설정에 대해 자세히 알아보려면 [Aurora Serverless 및 파라미터 그룹 \(p. 117\)](#) 단원을 참조하십시오.

서비스가 아닌 클러스터의 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 API를 사용해 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 연산을 사용하십시오.

Aurora에서 제공되는 DB 클러스터 및 DB 인스턴스 파라미터는 데이터베이스 엔진 호환성에 따라 다릅니다.

데이터베이스 엔진	파라미터
Aurora MySQL	Aurora MySQL 파라미터 (p. 665) 단원을 참조하십시오. Aurora Serverless 클러스터의 경우 Aurora Serverless 및 파라미터 그룹 (p. 117) 단원에서 추가 세부 정보를 참조하십시오.
Aurora PostgreSQL	Amazon Aurora PostgreSQL 파라미터 (p. 896) 단원을 참조하십시오.

DB 파라미터 그룹 생성

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 새 DB 파라미터 그룹을 생성할 수 있습니다.

콘솔

DB 파라미터 그룹을 생성하려면

1. AWS Management 콘솔에 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.
파라미터 그룹 생성 창이 나타납니다.
4. 파라미터 그룹 패밀리 목록에서 DB 파라미터 그룹 패밀리를 선택합니다.
5. 유형 목록에서 DB 파라미터 그룹을 선택합니다.
6. 그룹 이름 상자에 새 DB 파라미터 그룹의 이름을 입력합니다.
7. 설명 상자에 새 DB 파라미터 그룹에 대한 설명을 입력합니다.
8. 생성을 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 5.6용 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

Linux, OS X, Unix의 경우:

```
aws rds create-db-parameter-group \
--db-parameter-group-name mydbparametergroup \
--db-parameter-group-family aurora5.6 \
--description "My new parameter group"
```

Windows의 경우:

```
aws rds create-db-parameter-group ^
--db-parameter-group-name mydbparametergroup ^
--db-parameter-group-family aurora5.6 ^
--description "My new parameter group"
```

다음과 비슷한 출력이 생성됩니다.

```
DBPARAMETERGROUP  mydbparametergroup  aurora5.6  My new parameter group
```

RDS API

DB 파라미터 그룹을 생성하려면 RDS API [CreateDBParameterGroup](#) 작업을 사용합니다.

다음 필수 파라미터를 포함합니다.

- DBParameterGroupName
- DBParameterGroupFamily
- Description

DB 클러스터 파라미터 그룹 만들기

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 새 DB 클러스터 파라미터 그룹을 생성할 수 있습니다.

콘솔

DB 클러스터 파라미터 그룹을 생성하려면

1. AWS Management 콘솔에 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.

파라미터 그룹 생성 창이 나타납니다.

4. 파라미터 그룹 패밀리] 목록에서 DB 파라미터 그룹 패밀리를 선택합니다.
5. 유형 목록에서 DB 클러스터 파라미터 그룹을 선택합니다.
6. 그룹 이름 상자에 새로운 DB 클러스터 파라미터 그룹의 이름을 입력합니다.
7. 설명 상자에 새 DB 클러스터 파라미터 그룹에 대한 설명을 입력합니다.
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"이라는 설명과 함께 mydbclusterparametergroup이라는 MySQL 버전 5.6용 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

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster-parameter-group \
--db-cluster-parameter-group-name mydbclusterparametergroup \
--db-parameter-group-family aurora5.6 \
--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 aurora5.6 ^
--description "My new cluster parameter group"
```

다음과 비슷한 출력이 생성됩니다.

```
DBCLUSTERPARAMETERGROUP  mydbclusterparametergroup  mysql5.6  My cluster new parameter
group
```

RDS API

DB 클러스터 파라미터 그룹을 생성하려면 RDS API [CreateDBClusterParameterGroup](#) 작업을 사용합니다.

다음 필수 파라미터를 포함합니다.

- `DBClusterParameterGroupName`
- `DBParameterGroupFamily`
- `Description`

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 않은 경우 RDS 콘솔에 DB 파라미터 그룹이 재시작 보류 중 상태로 표시됩니다. 최신 파라미터 변경 내용을 이 DB 인스턴스에 적용하려면 해당 DB 인스턴스를 수동으로 재부팅해야 합니다.

Connectivity Monitoring Logs & events Configuration

Instance

Configuration

DB instance id
oracle-instance1

Engine version
12.1.0.2.v14

Storage type
General Purpose (SSD)

IOPS
-

Storage
20 GiB

DB name
ORCL

License model
Bring Your Own License

Character set
AL32UTF8

Option groups
default:oracle-ee-12-1

ARN
[REDACTED]

Resource id
[REDACTED]

Created time
Fri Nov 30 2018 15:41:20 GMT-0800 (Pacific Standard Time)

Parameter group
testsqnet (pending-reboot)

Deletion protection
Disabled

콘솔

DB 파라미터 그룹을 수정하려면

1. AWS Management 콘솔에 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. 수정할 파라미터의 값을 변경합니다. 대화 상자 오른쪽 위의 화살표 키를 사용하여 파라미터를 스크롤할 수 있습니다.
- 기본 파라미터 그룹의 값은 변경할 수 없습니다.
6. Save changes(변경 사항 저장)를 선택합니다.

AWS CLI

DB 파라미터 그룹을 수정하려면 AWS CLI `modify-db-parameter-group` 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--db-parameter-group-name`
- `--parameters`

다음 예에서는 `mydbparametergroup`이라는 DB 파라미터 그룹에서 `max_connections` 및 `max_allowed_packet` 값을 수정합니다.

Example

Linux, OS X, Unix의 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"
```

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` 명령을 다음 필수 파라미터와 함께 사용합니다.

- DBParameterGroupName
- Parameters

DB 클러스터 파라미터 그룹의 파라미터 수정

고객이 생성한 DB 클러스터 파라미터 그룹의 파라미터 값은 수정할 수 있지만, 기본 DB 클러스터 파라미터 그룹의 파라미터 값은 변경할 수 없습니다. 고객이 생성한 DB 클러스터 파라미터 그룹의 파라미터를 변경하면 DB 클러스터 파라미터 그룹과 연결된 모든 DB 클러스터에 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 콘솔에 DB 인스턴스와 연결된 DB 클러스터 파라미터 그룹의 상태가 표시됩니다. 예를 들어, DB 인스턴스에서 연결된 DB 클러스터 파라미터 그룹에 대한 최신 변경 내용을 사용하고 있지 않은 경우 RDS 콘솔에 DB 클러스터 파라미터 그룹이 pending-reboot 상태로 표시됩니다. 최신 파라미터 변경 내용을 DB 인스턴스에 적용하려면 해당 DB 인스턴스를 수동으로 재부팅해야 합니다.

Details

Configurations

ARN
arn:aws:rds:us-west-2: [REDACTED]:db:cluster5

Engine
Aurora MySQL 5.6.10a

Created Time
Tue Aug 07 08:47:41 GMT-700 2018

DB Name
cluster5

Username
myadmin

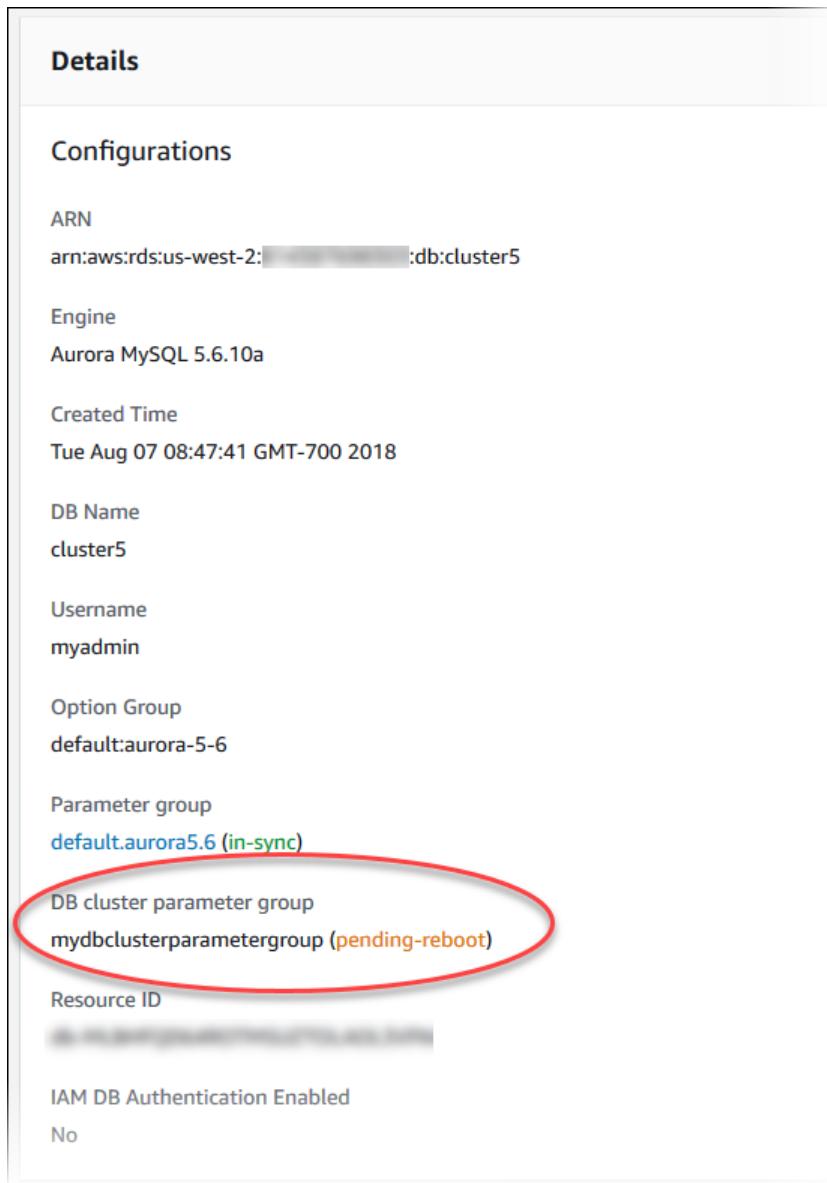
Option Group
default:aurora-5-6

Parameter group
[default.aurora5.6 \(in-sync\)](#)

DB cluster parameter group
[mydbclusterparametergroup \(pending-reboot\)](#)

Resource ID
[REDACTED]

IAM DB Authentication Enabled
No



콘솔

DB 클러스터 파라미터 그룹을 수정하려면

1. AWS Management 콘솔에 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(변경 사항 저장)를 선택합니다.

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 `mydbclusterparametergroup`이라는 DB 클러스터 파라미터 그룹에서 `server_audit_logging` 및 `server_audit_logs_upload` 값을 수정합니다.

Example

Linux, OS X, Unix의 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"
```

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`

DB 파라미터 그룹 복사

생성하는 사용자 지정 DB 파라미터 그룹을 복사할 수 있습니다. DB 파라미터 그룹을 이미 생성했으며 해당 그룹의 사용자 지정 파라미터와 값의 대부분을 새 DB 파라미터 그룹에 포함하려는 경우 파라미터 그룹을 복사하면 편리합니다. AWS CLI [copy-db-parameter-group](#) 명령 또는 RDS API [CopyDBParameterGroup](#) 작업을 사용하여 DB 파라미터 그룹을 복사할 수 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](#)의 파라미터 그룹 옵션이나 [describe-db-parameters](#) 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.

콘솔

DB 파라미터 그룹을 복사하려면

1. AWS Management 콘솔에 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 파라미터와 함께 사용합니다.

- `--source-db-parameter-group-identifier`
- `--target-db-parameter-group-identifier`
- `--target-db-parameter-group-description`

다음 예에서는 DB 파라미터 그룹 `mygroup1`을 복사하여 `mygroup2`라는 새 DB 파라미터 그룹을 생성합니다.

Example

Linux, OS X, Unix의 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"
```

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`

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 CLI [copy-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 생성하십시오. 이렇게 하면 파라미터 그룹이 새 DB 클러스터의 기본값으로 사용되기 전에 Amazon RDS에서 복사 작업을 완전히 마칠 수 있습니다. [Amazon RDS 콘솔](#)의 파라미터 그룹 옵션 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.

콘솔

DB 클러스터 파라미터 그룹을 복사하려면

1. AWS Management 콘솔에 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 `mygroup1`을 복사하여 `mygroup2`라는 새 DB 클러스터 파라미터 그룹을 생성합니다.

Example

Linux, OS X, Unix의 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"
```

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`

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 콘솔에 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.mysql5.5	mysql5.5	Default parameter group for MySQL5.5
DBPARAMETERGROUP	default.mysql5.6	mysql5.6	Default parameter group for MySQL5.6
DBPARAMETERGROUP	mydbparametergroup	mysql5.6	My new parameter group

다음은 mydbparamgroup1 파라미터 그룹을 설명하는 예제입니다.

Linux, OS X, Unix의 경우:

```
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	mysql5.5	My new parameter group
------------------	---------------------	----------	------------------------

RDS API

AWS 계정에 사용할 수 있는 모든 DB 파라미터 그룹을 나열하려면 RDS API [DescribeDBParameterGroups](#) 작업을 사용합니다.

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 콘솔에 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 그룹 유형에 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
```

다음과 같은 응답이 반환됩니다.

```
DBCLUSTERPARAMETERGROUPS      arn:aws:rds:us-west-2:1234567890:cluster-
pg:default.aurora5.6 default.aurora5.6      aurora5.6      Default cluster parameter
group for aurora5.6
DBCLUSTERPARAMETERGROUPS      arn:aws:rds:us-west-2:1234567890:cluster-
pg:mydbclusterparametergroup mydbclusterparametergroup      aurora5.6      My new cluster
parameter group
```

다음은 mydbclusterparametergroup 파라미터 그룹을 설명하는 예제입니다.

Linux, OS X, Unix의 경우:

```
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      arn:aws:rds:us-west-2:1234567890:cluster-
pg:mydbclusterparametergroup mydbclusterparametergroup      aurora5.6      My new cluster
parameter group
```

RDS API

AWS 계정에 사용할 수 있는 모든 DB 클러스터 파라미터 그룹을 나열하려면 RDS API [DescribeDBClusterParameterGroups](#) 작업을 사용합니다.

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 콘솔에 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 CLI `describe-db-parameters` 명령을 다음 필수 파라미터와 함께 사용합니다.

- `--db-parameter-group-name`

Example

다음 예에서는 `mydbparametergroup`이라는 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
DBPARAMETER	Type	Is Modifiable			
DBPARAMETER	allow-suspicious-udfs	false	engine-default	boolean	static
DBPARAMETER	auto_increment_increment	true	engine-default	integer	dynamic
DBPARAMETER	auto_increment_offset	true	engine-default	integer	dynamic
DBPARAMETER	binlog_cache_size	32768	system	integer	dynamic
DBPARAMETER	socket	/tmp/mysql.sock	system	string	static

RDS API

DB 파라미터 그룹의 파라미터 값을 보려면 RDS API `DescribeDBParameters` 명령을 다음 필수 파라미터와 함께 사용하십시오.

- `DBParameterGroupName`

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 콘솔에 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 그룹 유형에 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": [  
        {  
            "ApplyMethod": "pending-reboot",  
            "Description": "Controls whether user-defined functions that have only an xxx symbol for the main function can be loaded",  
            "DataType": "boolean",  
            "AllowedValues": "0,1",  
            "SupportedEngineModes": [  
                "provisioned"  
            ],  
            "Source": "engine-default",  
            "IsModifiable": false,  
            "ParameterName": "allow-suspicious-udfs",  
            "ApplyType": "static"  
        },  
        {  
            "ApplyMethod": "pending-reboot",  
            "Description": "Enables new features in the Aurora engine.",  
            "DataType": "boolean",  
            "IsModifiable": true,  
            "AllowedValues": "0,1",  
            "SupportedEngineModes": [  
                "provisioned"  
            ],  
            "Source": "engine-default",  
            "ParameterValue": "0",  
            "ParameterName": "aurora_lab_mode",  
            "ApplyType": "static"  
        },  
        ...  
    ]  
}
```

다음 예에서는 mydbclusterparametergroup이라는 DB 클러스터 파라미터 그룹에 대한 파라미터와 파라미터 값을 일반 텍스트 형식으로 나열합니다.

```
aws rds describe-db-cluster-parameters --db-cluster-parameter-group-name mydbclusterparametergroup --output text
```

다음과 같은 응답이 반환됩니다.

```
PARAMETERS 0,1 pending-reboot static boolean Controls whether user-defined functions that have only an xxx symbol for the main function can be loaded False allow-suspicious-udfs engine-default SUPPORTEDENGINEMODES provisioned  
PARAMETERS 0,1 pending-reboot static boolean Enables new features in the Aurora engine. True aurora_lab_mode 0 engine-default SUPPORTEDENGINEMODES provisioned
```

...

RDS API

DB 파라미터 그룹의 파라미터 값을 보려면 RDS API [DescribeDBClusterParameters](#) 명령을 다음 필수 파라미터와 함께 사용하십시오.

- `DBClusterParameterGroupName`

파라미터 그룹 비교

AWS Management 콘솔을 사용하여 동일한 DB 엔진 및 버전의 두 파라미터 그룹간 차이를 확인할 수 있습니다.

두 파라미터 그룹을 비교하려면

1. AWS Management 콘솔에 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.
4. 파라미터 그룹 작업에서 비교를 선택합니다.

Note

선택한 항목이 등등하지 않으면 비교를 선택할 수 없습니다. 예를 들어 MySQL 5.6과 MySQL 5.7 파라미터 그룹은 비교할 수 없습니다. DB 파라미터 그룹과 Aurora DB 클러스터 파라미터 그룹을 비교할 수 없습니다.

DB 파라미터 값

DB 파라미터의 값으로 다음 중 어느 것인든 지정할 수 있습니다.

- 정수 상수
- DB 파라미터 수식
- DB 파라미터 함수
- 문자열 상수
- `value={log(DBInstanceClassMemory/8187281418)*1000}`과 같은 로그식(로그 함수는 log base 2로 표시됨)

DB 파라미터 수식

DB 파라미터 수식은 정수 값 또는 부울 값으로 확인되는 식이며 종괄호({})로 둑입니다. DB 파라미터 함수에 대한 인수나 DB 파라미터 값에 대해 수식을 지정할 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 크기를 바이트 단위로 반환합니다.

DBInstanceClassMemory

현재 DB 인스턴스와 연결된 DB 인스턴스에 할당된 메모리에서 인스턴스를 관리하는 Amazon RDS 프로세스에 사용되는 메모리를 뺀 바이트 수를 반환합니다.

EndPointPort

DB 인스턴스에 연결하는 데 사용되는 포트의 번호를 반환합니다.

DB 파라미터 수식 연산자

DB 파라미터 수식은 나눗셈과 곱셈의 두 가지 연산자를 지원합니다.

나눗셈 연산자: /

나눗수를 나눗수로 나누어 정수 몫을 반환합니다. 몫의 소수는 잘리며 반올림되지 않습니다.

구문

```
dividend / divisor
```

나눗수 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

정수나 수식으로 파라미터 인수를 지정할 수 있습니다. 함수마다 인수가 하나 이상 있어야 합니다. 쉼표로 구분된 목록으로 여러 인수를 지정할 수 있습니다. 목록에 argument1, argument3과 같은 빈 멤버를 넣을 수 없습니다. 함수 이름은 대소문자를 구분하지 않습니다.

Note

현재는 AWS CLI에서 DB 파라미터 함수가 지원되지 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 값 예

다음 예에서는 DB 파라미터에 대한 값에 수식과 함수를 사용하는 방법을 보여 줍니다.

Warning

DB 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 백업하십시오. 파라미터 그룹 변경 내용을 프로덕션 DB 인스턴스에 적용하기 전에 특정 시점으로 복원을 사용하여 생성한 테스트 DB 인스턴스에 적용해 봐야 합니다.

Aurora MySQL table_definition_cache 파라미터 값에서 LEAST() 함수를 지정하여 정의 캐시에 저장될 수 있는 테이블 정의 수를 DBInstanceClassMemory/393040 또는 20,000 중 더 작은 값으로 설정할 수 있습니다.

```
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.

- Amazon RDS MySQL DB 인스턴스
- Amazon RDS 외부의 MySQL 데이터베이스
- MySQL과 호환되지 않는 데이터베이스

자세한 내용은 [Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 \(p. 472\)](#)을(를) 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 \(p. 781\)](#)을(를) 참조하십시오.

Amazon Aurora DB 클러스터 관리

이 섹션에서는 Aurora DB 클러스터를 관리 및 유지하는 방법을 보여줍니다. Aurora에는 복제 토플로지에 연결된 데이터베이스 서버의 클러스터가 포함되어 있습니다. 따라서 Aurora 관리를 위해서는 여러 개의 서버로 변경 사항을 배포하고 모든 Aurora 복제본이 마스터 서버를 따라 잡는지 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 \(p. 190\)](#)
- [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#)
- [DB 클러스터에 Aurora 복제본 추가 \(p. 208\)](#)
- [Aurora DB 클러스터의 성능 및 확장 관리 \(p. 211\)](#)
- [Amazon RDS Proxy\(미리 보기\)를 사용한 연결 관리 \(p. 213\)](#)
- [Aurora DB 클러스터에서 데이터베이스 복제 \(p. 239\)](#)
- [Aurora를 다른 AWS 서비스와 통합 \(p. 252\)](#)
- [Amazon Aurora DB 클러스터 백업 및 복구 \(p. 268\)](#)
- [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#)
- [DB 클러스터에서 DB 인스턴스 재부팅 \(p. 312\)](#)
- [AuroraDB 클러스터에서 DB 인스턴스 삭제 \(p. 313\)](#)
- [Amazon RDS 리소스에 태그 지정 \(p. 315\)](#)
- [Amazon RDS의 Amazon 리소스 이름\(ARN\)을 사용한 작업 \(p. 319\)](#)
- [Amazon Aurora 업데이트 \(p. 325\)](#)

Amazon Aurora DB 클러스터 중지 및 시작

Amazon Aurora 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 \(p. 190\)](#)
- [Aurora DB 클러스터 중지 및 시작에 대한 제한 사항 \(p. 191\)](#)
- [Aurora DB 클러스터 중지 \(p. 191\)](#)
- [Aurora DB 클러스터가 중지된 상태에서 가능한 작업 \(p. 192\)](#)
- [Aurora DB 클러스터 시작 \(p. 192\)](#)

Aurora DB 클러스터의 중지 및 시작 개요

Aurora 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.

다. 클러스터 내의 인스턴스 수와 관계없이 단일 작업만 관련된 모든 AWS Management 콘솔 절차를 수행할 수 있습니다.

DB 클러스터가 종지되어 있는 동안에는 지정된 보존 기간 내 클러스터 스토리지, 수동 스냅샷 및 자동 백업 스토리지에 대한 비용만 청구됩니다. DB 인스턴스 시간에 대해서는 비용이 청구되지 않습니다. Aurora은 7 일 후 DB 클러스터를 자동으로 시작하므로 필요한 유지 관리 업데이트보다 늦어지지 않습니다.

로드가 적은 Aurora 클러스터의 요금을 최소화하려면 Aurora 복제본을 모두 삭제하는 대신 클러스터를 종지할 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 종지를 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 종지할 수 없습니다.

Aurora DB 클러스터 종지 및 시작에 대한 제한 사항

다음과 같은 일부 Aurora 클러스터는 종지 및 시작이 불가능합니다.

- Aurora 글로벌 데이터베이스 (p. 37)의 일부인 클러스터는 종지 및 시작이 불가능합니다.
- Aurora 병렬 쿼리 (p. 527) 기능을 사용하는 클러스터는 종지 및 시작이 불가능합니다.

Aurora DB 클러스터 종지

Aurora DB 클러스터를 사용하거나 관리를 수행하려면 항상 실행 중인 Aurora DB 클러스터로 실행한 후 클러스터를 종지한 다음 클러스터를 다시 시작하십시오. 클러스터가 종지되어 있는 동안에는 DB 인스턴스 시간이 아니라 지정된 보존 기간 내 클러스터 스토리지, 수동 스냅샷 및 자동 백업 스토리지에 대한 비용이 청구됩니다.

종지 작업은 장애 조치 메커니즘 활성화를 피하기 위해 Aurora 복제본 인스턴스를 먼저 종지한 후 기본 인스턴스를 종지합니다.

다른 DB 클러스터의 데이터에 대한 복제 대상 역할을 하거나 복제 마스터로 작동하고 다른 클러스터로 데이터를 전송하는 DB 클러스터는 종지할 수 없습니다.

특정한 유형의 클러스터를 종지할 수 없습니다. 현재로서는 Aurora 전역 데이터베이스의 일부인 클러스터 또는 멀티 마스터 클러스터를 종지할 수 없습니다.

콘솔

Aurora 클러스터를 종지하려면

1. AWS Management 콘솔에 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 종지 작업을 수행하거나 종지하려는 DB 클러스터의 세부 정보 페이지로 이동하십시오.
3. Actions(작업)에서 Stop(종지)를 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 인스턴스를 종지하려면 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 \(p. 269\)](#) 단원을 참조하십시오.

클러스터가 중지된 동안 Aurora DB 클러스터 또는 해당 DB 인스턴스의 구성은 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.

Aurora은 다시 시작한 후 중지된 클러스터에 예약 유지보수를 적용합니다. 7일 후 Aurora은 중지된 클러스터를 자동으로 시작하므로 유지 관리 상태에서 너무 늦어지지 않는다는 점을 기억하십시오.

Aurora은 클러스터가 중지된 동안 기본 데이터를 변경할 수 없기 때문에 백업 자동화도 수행하지 않습니다. Aurora는 클러스터가 중지된 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.

콘솔

Aurora 클러스터를 시작하려면

1. AWS Management 콘솔에 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 이동하십시오.
3. Actions(작업)에는 Start(시작)을 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 클러스터를 시작하려면 다음 파라미터와 함께 [start-db-cluster](#) 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.

Amazon Aurora DB 클러스터 수정

백업 보존 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.

콘솔, CLI, API를 사용하여 DB 클러스터 수정

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터를 수정할 수 있습니다.

Note

Aurora의 경우, DB 클러스터 식별자, IAM DB 인증 및 새 마스터 암호 설정에 대한 변경 사항만이 즉시 적용 설정의 영향을 받습니다. 다른 모든 수정 사항은 즉시 적용 설정의 값에 상관없이 즉시 적용됩니다.

콘솔

DB 클러스터를 수정하려면

1. AWS Management 콘솔에 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 \(p. 196\)](#) 단원을 참조하십시오.

Note

AWS Management 콘솔에서는 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 설정 \(p. 196\)](#)의 설정 범위를 참조하십시오. AWS Management 콘솔에서 인스턴스 수준에서 전체 DB 클러스터를 수정하는 설정을 변경하려면 [DB 클러스터에서 DB 인스턴스 수정 \(p. 194\)](#) 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 \(p. 196\)](#) 단원을 참조하십시오.

Note

일부 설정은 DB 인스턴스에만 적용됩니다. 이러한 설정을 변경하려면 DB 클러스터에서 DB 인스턴스 수정 ([p. 194](#))의 지침을 따르십시오.

Example

다음 명령은 백업 보존 기간을 1주(7일)로 설정하여 `mydbcluster`를 수정합니다.

Linux, OS X, Unix의 경우:

```
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 \(p. 196\)](#) 단원을 참조하십시오.

Note

일부 설정은 DB 인스턴스에만 적용됩니다. 이러한 설정을 변경하려면 DB 클러스터에서 DB 인스턴스 수정 ([p. 194](#))의 지침을 따르십시오.

DB 클러스터에서 DB 인스턴스 수정

AWS Management 콘솔, 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 콘솔에서 **Apply Immediately**(즉시 적용) 옵션을 선택하거나, AWS CLI를 호출하는 경우 `--apply-immediately` 파라미터를 사용하거나, Amazon RDS API를 사용하는 경우 `ApplyImmediately` 파라미터를 `true`로 설정합니다.

변경 사항을 즉시 적용하지 않기로 선택하면 변경 사항이 보류 중 수정 사항 대기열로 보내집니다. 다음 유지 관리 기간에 대기열에 있는 보류 중 변경 사항이 적용됩니다. 변경 사항을 즉시 적용하기로 선택하면 새로운 변경 사항과 보류 중인 수정 사항 대기열에 있는 모든 변경 사항이 적용됩니다.

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.

콘솔

DB 클러스터에서 DB 인스턴스를 수정하려면

1. AWS Management 콘솔에 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 \(p. 196\)](#) 단원을 참조하십시오.

Note

일부 설정은 전체 DB 클러스터에 적용되며, 클러스터 수준에서 변경이 필요합니다. 이러한 설정을 변경하려면 [콘솔, CLI, API를 사용하여 DB 클러스터 수정 \(p. 193\)](#)의 지침을 따르십시오.

AWS Management 콘솔에서는 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 설정 \(p. 196\)](#)의 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 [Modify DB Instance]를 선택하여 변경 내용을 저장합니다.

그렇지 않으면 [Back]를 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 \(p. 196\)](#) 단원을 참조하십시오.

Note

일부 설정은 전체 DB 클러스터에 적용됩니다. 이러한 설정을 변경하려면 [콘솔, CLI, API를 사용하여 DB 클러스터 수정 \(p. 193\)](#)의 지침을 따르십시오.

Example

다음 코드는 DB 인스턴스 클래스를 db.r4.xlarge로 설정하여 mydbinstance를 수정합니다. 변경 사항은 --no-apply-immediately를 사용하여 다음 유지 관리 기간에 적용됩니다. 변경 사항을 바로 적용하려면 --apply-immediately를 사용합니다.

Linux, OS X, Unix의 경우:

```
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 사용해 MySQL 인스턴스를 수정하려면 [ModifyDBInstance](#) 작업을 호출합니다. DB 인스턴스 식별자와 수정하려는 설정 값을 지정하십시오. 각 파라미터에 대한 자세한 내용은 [Amazon Aurora에 대한 설정 \(p. 196\)](#) 단원을 참조하십시오.

Note

일부 설정은 전체 DB 클러스터에 적용됩니다. 이러한 설정을 변경하려면 [콘솔, CLI, API를 사용하여 DB 클러스터 수정 \(p. 193\)](#)의 지침을 따르십시오.

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.

설정 및 설명	방법	범위	자동 중지 참고 사항
<p>마이너 버전 자동 업그레이드</p> <p>사용 가능하게 되면 DB 인스턴스에 기본 마이너 엔진 버전 업그레이드를 자동으로 받을지 여부. 업그레이드는 예약된 유지 관리 기간 중에만 설치됩니다.</p> <p>마이너 버전 자동 업그레이드 설정은 Aurora PostgreSQL DB 클러스터에만 적용됩니다.</p> <p>Aurora PostgreSQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora PostgreSQL 데이터베이스 엔진 업데이트 (p. 905) 단원을 참조하십시오.</p> <p>Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL 데이터베이스 엔진 업데이트 (p. 686) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--auto-minor-version-upgrade</code> <code>--no-auto-minor-version-upgrade</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 ModifyDBInstance를 호출하고 <code>AutoMinorVersionUpgrade</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.
<p>백업 보존 기간</p> <p>자동 백업을 보존할 수. 최소값은 1입니다.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-</code></p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	자동 중지 참고 사항
자세한 내용은 백업 (p. 269) 단원을 참조하십시오.	<p><code>cluster</code>를 실행하고 <code>--backup-retention-period</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>BackupRetentionPeriod</code> 파라미터를 설정하십시오.</p>		
인증 기관 사용할 인증서.	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--ca-certificate-identifier</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBInstance</code>를 호출하고 <code>CACertificateIdentifier</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단됩니다. DB 인스턴스가 재부팅됩니다.
스냅샷으로 태그 복사 이 옵션을 선택하면 현재 DB 클러스터에 정의되어 있는 태그가 현재 DB 클러스터에서 생성된 DB 스냅샷으로 복사됩니다. 자세한 내용은 Amazon RDS 리소스에 태그 지정 (p. 315) 단원을 참조하십시오.	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--copy-tags-to-snapshot</code> 또는 <code>--no-copy-tags-to-snapshot</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>CopyTagsToSnapshot</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	자동 중지 참고 사항
<p>데이터 API AWS Lambda, AWS AppSync 등 웹 서비스 기반 애플리케이션을 사용하여 Aurora Serverless에 액세스할 수 있습니다. 이 설정은 Aurora Serverless DB 클러스터에만 적용됩니다.</p> <p>자세한 내용은 Aurora Serverless에 데이터 API 사용 (p. 134) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--enable-http-endpoint</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>EnableHttpEndpoint</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.
<p>Database authentication(데이터베이스 인증) 귀하가 사용하고 싶은 데이터베이스 인증 옵션입니다. 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 인증할 수 있도록 Password and IAM DB authentication(암호 및 IAM DB 인증)을 선택합니다. 자세한 정보는 을 위한 IAM 데이터베이스 인증 (p. 976) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--enable-iam-database-authentication</code> <code>--no-enable-iam-database-authentication</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>EnableIAMDatabaseAuthentication</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
데이터베이스 포트 DB 클러스터에 액세스하는 데 사용할 포트.	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--port</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>Port</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단됩니다. DB 클러스터의 모든 DB 인스턴스는 즉시 재부팅됩니다.
[DB cluster identifier] DB 클러스터 식별자입니다. 이 값은 소문자 문자열로 저장됩니다. DB 클러스터 식별자를 변경하면 DB 클러스터 엔드포인트가 변경되고 DB 클러스터의 DB 인스턴스 엔드포인트가 변경됩니다.	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--new-db-cluster-identifier</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>NewDBClusterIdentifier</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.
DB 클러스터 파라미터 그룹 DB 클러스터와 연결할 DB 클러스터 파라미터 그룹. 자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--db-cluster-parameter-group-name</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>DBClusterParameterGroupName</code> 파라미터를 설정하십시오.</p>	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
<p>DB 엔진 버전</p> <p>사용하려는 DB 엔진의 버전. 프로덕션 DB 클러스터를 업그레이드 하려면 먼저 테스트 DB 클러스터에서 업그레이드 프로세스를 테스트하여 지속 시간을 확인하고 애플리케이션의 유효성을 확인하는 것이 좋습니다.</p> <p>현재 이 설정만 사용하여 Aurora PostgreSQL DB 클러스터의 마이너 엔진 버전을 업그레이드할 수 있습니다.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--engine-version</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 ModifyDBCluster를 호출하고 <code>EngineVersion</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단됩니다.
<p>DB 인스턴스 클래스</p> <p>사용할 DB 인스턴스 클래스.</p> <p>자세한 내용은 DB 인스턴스 클래스 (p. 30) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--db-instance-class</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 ModifyDBInstance를 호출하고 <code>DBInstanceClass</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단됩니다.
<p>DB 인스턴스 식별자</p> <p>DB 인스턴스 식별자 이 값은 소문자 문자열로 저장됩니다.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--new-db-instance-identifier</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 ModifyDBInstance를 호출하고 <code>NewDBInstanceIdentifier</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단됩니다. DB 인스턴스가 재부팅됩니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
<p>DB 파라미터 그룹 DB 인스턴스와 연결하려는 DB 파라미터 그룹.</p> <p>자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--db-parameter-group-name</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBInstance</code>를 호출하고 <code>DBParameterGroupName</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다. 파라미터 그룹을 변경하는 경우 일부 파라미터에 대한 변경 내용은 재부팅 없이 DB 인스턴스에 즉시 적용됩니다. 다른 파라미터에 대한 변경 내용은 DB 인스턴스를 재부팅한 후에만 적용됩니다.
<p>삭제 방지</p> <p>DB 클러스터가 삭제되지 않도록 방지하기 위한 삭제 방지 활성화 자세한 내용은 AuroraDB 클러스터에서 DB 인스턴스 삭제 (p. 313) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--deletion-protection --no-deletion-protection</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>DeletionProtection</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	자동 중지 참고 사항
<p>확장 모니터링 DB 인스턴스가 실행되는 운영 체제에 대한 실시간 측정치 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다.</p> <p>자세한 내용은 확장 모니터링 (p. 358) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--monitoring-role-arn</code> 및 <code>--monitoring-interval</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBInstance</code>를 호출하고 <code>MonitoringRoleArn</code> 및 <code>MonitoringInterval</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.
<p>로그 내보내기 Amazon CloudWatch Logs에 게시할 로그 유형을 선택합니다.</p> <p>자세한 내용은 MySQL 데이터베이스 로그 파일 (p. 452) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--cloudwatch-logs-export-configuration</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>CloudwatchLogsExportConfiguration</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	자동 종지 참고 사항
<p>유지 관리 기간</p> <p>시스템 유지 관리를 실행하는 기간. 시스템 유지 관리는 업그레이드를 포함합니다(해당할 경우). 유지 관리 기간은 국제 표준 시(UTC)의 시작 시간과 시간 단위의 지속 기간으로 구성됩니다.</p> <p>이 기간을 현재 시간으로 설정하려면 대기 종인 변경 사항이 모두 적용될 수 있도록 현재 시간과 기간 종료 시간 사이에 최소 30분 이상 필요 합니다.</p> <p>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.</p> <p>자세한 내용은 Amazon RDS 유지 관리 기간 (p. 309) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔을 사용하여 DB 클러스터의 유지 관리 기간을 변경하려면 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193).</p> <p>AWS Management 콘솔을 사용하여 DB 인스턴스의 유지 관리 기간을 변경하려면 DB 클러스터에서 DB 인스턴스 수정 (p. 194).</p> <p>AWS CLI를 사용하여 DB 클러스터의 유지 관리 기간을 변경하려면 <code>modify-db-cluster</code>를 실행하고 <code>--preferred-maintenance-window</code> 옵션을 설정하십시오.</p> <p>AWS CLI를 사용하여 DB 인스턴스의 유지 관리 기간을 변경하려면 <code>modify-db-instance</code>를 실행하고 <code>--preferred-maintenance-window</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 DB 클러스터의 유지 관리 기간을 변경하려면 <code>ModifyDBCluster</code>를 호출하고 <code>PreferredMaintenanceWindow</code> 파라미터를 설정하십시오.</p> <p>RDS API를 사용하여 DB 인스턴스의 유지 관리 기간을 변경하려면 <code>ModifyDBInstance</code>를 호출하고 <code>PreferredMaintenanceWindow</code> 파라미터를 설정하십시오.</p>	<p>전체 DB 클러스터 또는 단일 DB 인스턴스</p>	<p>인스턴스가 종단될 수 있는 작업이 하나 이상 대기 중이고, 유지 관리 기간이 현재 시간을 포함하여 변경된 경우 대기 중인 작업들이 즉시 적용되고 인스턴스가 종단됩니다.</p>

설정 및 설명	방법	범위	자동 중지 참고 사항
<p>새 마스터 암호 마스터 사용자의 암호. 암호에는 8~41자의 영숫자 문자로 구성되어야 합니다.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--master-user-password</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>MasterUserPassword</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.
<p>성능 개선 도우미 데이터베이스 성능을 분석하고 문제를 해결할 수 있도록 DB 인스턴스 로드를 모니터링하는 도구인 성능 개선 도우미 활성화 여부.</p> <p>자세한 내용은 Amazon RDS 성능 개선 도우미 사용 (p. 365) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--enable-performance-insights</code> <code>--no-enable-performance-insights</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBInstance</code>를 호출하고 <code>EnablePerformanceInsights</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	자동 중지 참고 사항
<p>성능 개선 도우미 AWS KMS key identifier(성능 개선 도우미 AWS KMS 키 식별자)</p> <p>성능 개선 도우미 데이터의 암호화를 위한 AWS KMS 키 식별자 KMS 키 ID는 Amazon 리소스 이름(ARN), KMS 키 식별자 또는 KMS 암호화 키에 대한 KMS 키 별칭입니다.</p> <p>자세한 내용은 성능 개선 도우미 활성화 (p. 367) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--performance-insights-kms-key-id</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBInstance</code>를 호출하고 <code>PerformanceInsightsKMSKeyId</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.
<p>성능 개선 도우미 retention period(성능 개선 도우미 보존 기간)</p> <p>성능 개선 도우미 데이터를 보존할 시간(일). 유효한 값은 7 또는 731(2년)입니다.</p> <p>자세한 내용은 성능 개선 도우미 활성화 (p. 367) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--performance-insights-retention-period</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBInstance</code>를 호출하고 <code>PerformanceInsightsRetentionPeriod</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.
<p>Promotion tier(프로모션 티어)</p> <p>기존 기본 인스턴스에 결함이 발생한 후 Aurora 복제본을 단일 마스터 복제를 사용하는 클러스터의 기본 인스턴스로 승격할 순서를 지정하는 값입니다.</p> <p>자세한 내용은 Aurora DB 클러스터의 내결함성 (p. 268) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--promotion-tier</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBInstance</code>를 호출하고 <code>PromotionTier</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	가동 중지 참고 사항
<p>퍼블릭 액세스 가능성 DB 인스턴스에 퍼블릭 IP 주소를 부여할지 여부 (즉 VPC 외부에서 액세스할 수 있음). 공개적으로 액세스가 가능하려면 DB 인스턴스도 VPC의 퍼블릭 서브넷에 있어야 합니다. DB 인스턴스를 공개적으로 액세스할 수 없으면 VPC 내부에서만 액세스할 수 있습니다.</p> <p>자세한 내용은 VPC에 있는 DB 인스턴스를 인터넷에서 숨기기 (p. 1017) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, DB 클러스터에서 DB 인스턴스 수정 (p. 194) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-instance</code>를 실행하고 <code>--publicly-accessible</code> <code>--no-publicly-accessible</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBInstance</code>를 호출하고 <code>PubliclyAccessible</code> 파라미터를 설정하십시오.</p>	지정된 DB 인스턴스만 해당	이 변경 도중 인스턴스가 중단되지 않습니다.
<p>Scaling configuration(조정 구성)</p> <p>DB 클러스터의 조정 속성. DB 클러스터의 조정 속성은 serverless DB 엔진 모드에서만 수정할 수 있습니다. 이 설정은 Aurora MySQL에만 사용 가능합니다.</p> <p>Aurora Serverless에 대한 자세한 내용은 사용 Amazon Aurora Serverless (p. 111) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--scaling-configuration</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>ScalingConfiguration</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

설정 및 설명	방법	범위	자동 중지 참고 사항
<p>보안 그룹 DB 클러스터와 연결할 보안 그룹.</p> <p>자세한 내용은 보안 그룹을 통한 액세스 제어 (p. 999) 단원을 참조하십시오.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--vpc-security-group-ids</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>VpcSecurityGroupIds</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.
<p>대상 역추적 기간</p> <p>DB 클러스터를 역추적 가능한 시간(초). 이 설정은 Aurora MySQL에서만 사용 가능하며 역추적을 활성화한 상태에서 DB 클러스터가 생성된 경우에만 사용 가능합니다.</p>	<p>AWS Management 콘솔, 콘솔, CLI, API를 사용하여 DB 클러스터 수정 (p. 193) 사용.</p> <p>AWS CLI를 사용하여 <code>modify-db-cluster</code>를 실행하고 <code>--backtrack-window</code> 옵션을 설정하십시오.</p> <p>RDS API를 사용하여 <code>ModifyDBCluster</code>를 호출하고 <code>BacktrackWindow</code> 파라미터를 설정하십시오.</p>	전체 DB 클러스터	이 변경 도중 인스턴스가 중단되지 않습니다.

Amazon Aurora에 적용되지 않는 설정

다음과 같은 AWS CLI 명령 `modify-db-instance` 및 RDS API 작업 `ModifyDBInstance`의 설정은 Amazon Aurora에 적용되지 않습니다.

Note

AWS Management 콘솔에서는 Aurora에 대한 이러한 설정을 수정할 수 없습니다.

AWS CLI 설정	RDS API 설정
<code>--allocated-storage</code>	<code>AllocatedStorage</code>
<code>--allow-major-version-upgrade --no-allow-major-version-upgrade</code>	<code>AllowMajorVersionUpgrade</code>
<code>--copy-tags-to-snapshot --no-copy-tags-to-snapshot</code>	<code>CopyTagsToSnapshot</code>

AWS CLI 설정	RDS API 설정
--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
--option-group-name	OptionGroupName
--preferred-backup-window	PreferredBackupWindow
--processor-features	ProcessorFeatures
--storage-type	StorageType
--tde-credential-arn	TdeCredentialArn
--tde-credential-password	TdeCredentialPassword
--use-default-processor-features --no-use-default-processor-features	UseDefaultProcessorFeatures

Note

선후되는 백업 기간은 Aurora에 대해 설정 가능하지만 이 설정은 무시됩니다. Aurora 백업은 연속적이고 충분됩니다.

DB 클러스터에 Aurora 복제본 추가

단일 마스터 복제를 사용하는 Aurora DB 클러스터에는 기본 DB 인스턴스 1개와 최대 15개의 Aurora 복제본이 있습니다. 기본 DB 인스턴스는 읽기 및 쓰기 작업을 지원하고, 클러스터 볼륨에 대한 모든 데이터 수정을 수행합니다. Aurora 복제본은 기본 DB 인스턴스와 동일한 스토리지 볼륨에 연결되며 읽기 작업만 지원합니다. Aurora 복제본은 기본 DB 인스턴스에서 읽기 워크로드를 오프로드할 수 있습니다.

DB 클러스터의 가용성을 높이려면 여러 가용 영역에 걸쳐 DB 클러스터에 기본 인스턴스와 Aurora 복제본을 분배하는 것이 좋습니다. 자세한 정보는 [리전 가용성 \(p. 3\)](#) 단원을 참조하십시오.

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터에 Aurora 복제본을 추가할 수 있습니다.

DB 클러스터에서 Aurora 복제본을 제거하려면 [AuroraDB 클러스터에서 DB 인스턴스 삭제 \(p. 313\)](#)의 지침에 따라 Aurora 복제본 DB 인스턴스를 삭제하십시오.

Aurora 복제본에 대한 자세한 정보는 [Aurora 복제본 \(p. 55\)](#) 단원을 참조하십시오.

Note

또한 Amazon Aurora은 외부 데이터베이스 또는 RDS DB 인스턴스의 복제도 지원합니다. Amazon Aurora를 사용할 경우 RDS DB 인스턴스가 동일한 AWS 리전에 있어야 합니다. 자세한 정보는 [Amazon Aurora를 사용한 복제 \(p. 55\)](#) 단원을 참조하십시오.

콘솔

DB 클러스터에 Aurora 복제본을 추가하려면

1. AWS Management 콘솔에 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. 작업에서 Add reader(리더 추가)를 선택합니다.

Add reader(리더 추가) 페이지가 나타납니다.

4. Add reader(리더 추가) 페이지에서 Aurora 복제본에 대한 옵션을 지정합니다. 다음 표에서는 Aurora 복제본 설정을 보여 줍니다.

옵션	수행할 작업
[Availability zone]	특정 가용 영역의 지정 여부를 결정합니다. 이 목록에는 앞에서 지정한 DB 서브넷 그룹으로 매핑되어 있는 가용 영역만 포함됩니다. 가용 영역에 대한 자세한 정보는 리전 및 가용 영역 (p. 3) 단원을 참조하십시오.
[Publicly accessible]	Aurora 복제본에 퍼블릭 IP 주소를 할당하려면 yes를 선택하고, 그렇지 않으면 No를 선택합니다. 모든 사용자의 액세스에서 Aurora 복제본을 습기는 방법에 대한 자세한 정보는 VPC에 있는 DB 인스턴스를 인터넷에서 습기기 (p. 1017) 단원을 참조하십시오.
암호화	이 Aurora 복제본에 대해 암호화를 활성화하려면 Enable encryption 을 선택합니다. 자세한 정보는 Amazon Aurora 리소스 암호화 (p. 945) 단원을 참조하십시오.
DB 인스턴스 클래스	Aurora 복제본의 처리 및 메모리 요건을 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스 옵션에 대한 자세한 정보는 DB 인스턴스 클래스 (p. 30) 단원을 참조하십시오.
Aurora 복제본 소스	Aurora 복제본을 생성할 기본 인스턴스의 식별자를 선택합니다.
DB 인스턴스 식별자	선택한 AWS 리전의 계정에 대해 고유한 인스턴스의 이름을 입력합니다. 선택한 AWS 리전 및 DB 엔진을 포함해(예: aurora-read-instance1) 이름에 몇 가지 인텔리전스를 추가할 수 있습니다.
Priority	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 정보는 Aurora DB 클러스터의 내결합성 (p. 268) 단원을 참조하십시오.
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 정보는 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.

옵션	수행할 작업
확장 모니터링	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 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-role라는 이름의 역할을 생성하도록 기본값을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 단원을 참조하십시오.
Granularity	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
Auto minor version upgrade	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. 자동 마이너 버전 업그레이드 설정은 Aurora PostgreSQL DB 클러스터에만 적용됩니다. Aurora PostgreSQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora PostgreSQL 데이터베이스 엔진 업데이트 (p. 905) 단원을 참조하십시오. Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL 데이터베이스 엔진 업데이트 (p. 686) 단원을 참조하십시오.

5. Add reader(리더 추가)를 선택하여 Aurora 복제본을 생성합니다.

AWS CLI

DB 클러스터에 Aurora 복제본을 생성하려면 `create-db-instance` AWS CLI 명령을 실행하십시오. 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-2a인 새 MySQL 5.7-호환 Aurora 복제본이 생성됩니다.

Linux, OS X, Unix의 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.r4.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.r4.large ^
```

```
--availability-zone us-west-2a
```

다음 명령을 사용하면 이름이 sample-instance-us-west-2a인 새 MySQL 5.6-호환 Aurora 복제본이 생성됩니다.

Linux, OS X, Unix의 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.r4.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.r4.large
    ^
    --availability-zone us-west-2a
```

다음 명령을 사용하면 이름이 sample-instance-us-west-2a인 PostgreSQL 호환 Aurora 복제본이 생성됩니다.

Linux, OS X, Unix의 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.r4.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.r4.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 DB 클러스터의 성능 및 확장 관리

다음 옵션을 사용해 Aurora DB 클러스터 및 DB 인스턴스의 성능 및 확장을 관리할 수 있습니다.

주제

- [스토리지 조정 \(p. 212\)](#)
- [인스턴스 조정 \(p. 212\)](#)
- [읽기 조정 \(p. 212\)](#)
- [연결 관리 \(p. 212\)](#)

- 쿼리 실행 계획 관리 (p. 213)

스토리지 조정

Aurora 스토리지는 클러스터 볼륨에 저장된 데이터에 따라 자동 조정됩니다. 데이터가 증가함에 따라 클러스터 볼륨 스토리지도 최대 64 TiB까지 10기비바이트(GB)씩 확장됩니다.

클러스터 볼륨 크기는 1시간 단위로 확인하여 스토리지 비용을 측정할 수 있습니다. 요금에 대한 자세한 내용은 [Aurora 요금 페이지](#)를 참조하십시오.

테이블 또는 파티션을 삭제하는 등으로 Aurora 데이터가 제거될 때 전체 할당 공간은 동일하게 유지됩니다. 사용 가능한 공간은 향후에 데이터 볼륨이 증가할 때 자동으로 재사용됩니다.

Note

스토리지 비용은 어떤 시점에서든 Aurora 클러스터에 할당된 최대 양인 스토리지 “하이 워터 마크”를 토대로 합니다. 임시 정보를 대량 생성하는 ETL(Extract, Transform, Load) 작업을 방지하는 방법으로 비용을 관리할 수 있습니다. 마찬가지로 불필요한 오래된 데이터를 제거하기 전에 새 데이터를 대량으로 로드하는 ETL 작업을 방지하는 방법으로 비용을 관리할 수 있습니다.

Aurora 클러스터에서 데이터를 제거하면 많은 양이 할당되었지만 사용되지 않은 공간이 발생하는 경우 하이 워터 마크를 재설정할 때 `mysqldump`과 같은 도구를 사용하여 논리적 데이터 덤프를 수행하고 새 클러스터로 복원해야 합니다. 스냅샷을 생성하고 복원하면 기본 스토리지의 물리적 레이아웃이 복원된 스냅샷에서 동일하게 유지되기 때문에 할당된 스토리지가 감소하지 않습니다.

인스턴스 조정

필요에 따라 DB 클러스터의 DB 인스턴스마다 DB 인스턴스 클래스를 수정하여 Aurora DB 클러스터 규모를 조정할 수 있습니다. Aurora는 데이터베이스 엔진 호환성에 따라 Aurora에 최적화된 몇 가지 DB 인스턴스 클래스를 지원합니다.

데이터베이스 엔진	인스턴스 조정
Amazon Aurora MySQL	Aurora MySQL DB 인스턴스 조정 (p. 508) 참조
Amazon Aurora PostgreSQL	Aurora PostgreSQL DB 인스턴스 조정 (p. 802) 참조

읽기 조정

단일 마스터 복제를 사용하는 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 \(p. 208\)](#) 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 연결 (p. 508) 참조
Amazon Aurora PostgreSQL	Aurora PostgreSQL DB 인스턴스에 대한 최대 연결 (p. 802) 참조

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 \(p. 821\)](#) 단원을 참조하십시오.

Amazon RDS Proxy(미리 보기)를 사용한 연결 관리

이 문서는 Amazon RDS Proxy에 대한 시험판 설명서입니다. 변경될 수 있습니다.

Amazon RDS Proxy를 사용하면 애플리케이션이 데이터베이스 연결을 풀링하고 공유하여 확장성을 향상 할 수 있습니다. 애플리케이션 연결을 유지하면서 대기 DB 인스턴스에 자동으로 연결하여 데이터베이스 장애에 대한 애플리케이션의 탄력성을 높여줍니다. 또한 RDS Proxy는 데이터베이스에 AWS IAM(자격 증명 및 액세스 관리) 인증을 적용하고 Secrets Manager에 자격 증명을 안전하게 저장할 수 있게 해줍니다. RDS Proxy는 MySQL과 완벽하게 호환되며 코드 변경 없이 대부분의 애플리케이션에서 사용할 수 있습니다.

연결을 초과 구독하거나 빠른 속도로 새 연결을 생성할 경우 발생할 수 있는 예기치 않은 데이터베이스 트래픽 급증을 처리할 수 있습니다. RDS Proxy는 데이터베이스 연결 풀을 설정하고 매번 새 데이터베이스 연결을 여는 메모리 및 CPU 오버헤드 없이 이 풀에서 연결을 다시 사용합니다. 데이터베이스를 초과 구독으로부터 보호하기 위해 생성되는 데이터베이스 연결 수를 제어할 수 있습니다. RDS Proxy는 연결 풀에서 즉시 재공할 수 없는 애플리케이션 연결을 시퀀싱 또는 조절합니다. 대기 시간이 증가할 수 있지만 애플리케이션은 갑작스러운 데이터베이스 장애 또는 압도 없이 계속 확장될 수 있습니다. 연결 요청이 지정한 제한을 초과하는 경우 RDS Proxy는 사용 가능한 용량으로 처리될 수 있는 로드에 대해 예측 가능한 성능을 유지하는 수준에서 애플리케이션 연결을 거부합니다(부하 감소).

자격 증명을 처리하고 각 새 연결에 대한 보안 연결을 설정하는 데 필요한 오버헤드를 줄일 수 있습니다. RDS Proxy가 데이터베이스를 대신하여 해당 작업 중 일부를 처리할 수 있습니다.

Important

현재 Amazon RDS Proxy는 미리 보기로 제공됩니다. RDS Proxy를 프로덕션 워크로드에 사용하지 마십시오. 또한 RDS Proxy 미리 보기로 사용하는 데이터베이스에는 중요한 데이터를 저장하지 않는 것이 좋습니다. 미리 보기 중에 잘못된 결과, 데이터 손상 또는 그 두 가지를 모두 유발하는 문제가 발생할 수 있습니다. 미리 보기 기간 동안 AWS에서 사전 알림 없이 주요 사항을 변경할 수 있습니다. 여기에는 API 업그레이드 및 변경이 포함될 수 있습니다.

주제

- [RDS Proxy 개념 및 용어 \(p. 214\)](#)
- [RDS Proxy 계획 및 설정 \(p. 217\)](#)
- [RDS Proxy를 통해 데이터베이스에 연결 \(p. 229\)](#)
- [RDS Proxy 관리 \(p. 230\)](#)
- [RDS Proxy 모니터링 \(p. 232\)](#)
- [RDS Proxy 제한 사항 \(p. 234\)](#)
- [RDS Proxy용 명령줄 예제 \(p. 234\)](#)

- [RDS Proxy 문제 해결 \(p. 236\)](#)

RDS Proxy 개념 및 용어

RDS Proxy를 사용하여 Amazon RDS DB 인스턴스와 Aurora DB 클러스터에 대한 연결 관리를 간소화할 수 있습니다.

RDS Proxy는 클라이언트 애플리케이션과 데이터베이스 사이의 네트워크 트래픽을 처리합니다. 데이터베이스 프로토콜을 이해하고 애플리케이션의 SQL 작업과 데이터베이스의 결과 세트를 기반으로 동작을 조정하여 능동적으로 수행합니다. RDS Proxy는 데이터베이스 연결 관리를 위한 메모리 및 CPU 오버헤드를 줄입니다. 애플리케이션이 많은 연결을 동시에 열 때 데이터베이스에 필요한 메모리 및 CPU 리소스가 더 적습니다. 또한 오랫동안 유지 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 RDS DB 인스턴스 및 Aurora DB 클러스터용 데이터베이스 서버와 독립적입니다. 이러한 분리는 데이터베이스 서버의 오버헤드를 줄여 데이터베이스 워크로드를 처리하는 데 리소스가 투입될 수 있습니다.

주제

- [RDS Proxy 용어 개요 \(p. 214\)](#)
- [연결 풀링 \(p. 215\)](#)
- [RDS Proxy 보안 \(p. 215\)](#)
- [Failover \(p. 216\)](#)
- [트랜잭션 \(p. 217\)](#)

RDS Proxy 용어 개요

RDS Proxy는 연결 풀링 및 이 설명서에 설명된 다른 기능을 수행하기 위해 인프라를 처리합니다. RDS용 AWS Management 콘솔의 프록시 페이지에서 표시된 프록시를 볼 수 있습니다.

각 프록시는 단일 RDS DB 인스턴스 또는 Aurora DB 클러스터에 대한 연결을 처리합니다. RDS 다중 AZ DB 인스턴스 및 Aurora 프로비저닝된 클러스터의 경우 프록시가 현재 라이터 인스턴스를 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 Proxy는 다른 연결을 사용하여 세션에서 다른 트랜잭션을 실행하는 것이 안전한지 확인할 수 없으면 세션이 끝날 때까지 동일한 연결에서 세션을 유지합니다. 이 대체 동작을 고정이라고 합니다.

프록시에는 엔드포인트가 있습니다. RDS DB 인스턴스 또는 Aurora DB 클러스터로 작업할 때 인스턴스 또는 클러스터에 직접 연결하는 읽기-쓰기 엔드포인트 대신 이 엔드포인트에 연결합니다. Aurora 클러스터에 대한 특수 용도 엔드포인트는 계속 사용할 수 있습니다. 예를 들어 연결 풀링 없이 읽기-쓰기 연결을 위해 클러스터 엔드포인트에 연결할 수 있습니다. 로드 밸런싱된 읽기 전용 연결을 위해 리더 엔드포인트에 계속 연결할 수 있습니다. Aurora 클러스터의 특정 DB 인스턴스에 대한 진단 및 문제 해결을 위해 인스턴스 엔드포인트에 계속 연결할 수 있습니다. AWS Lambda와 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 엔드포인트를 지정합니다.

각 프록시는 대상 그룹을 포함합니다. 이 대상 그룹은 프록시가 연결할 수 있는 RDS DB 인스턴스 또는 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 RDS DB 인스턴스 또는 Aurora DB 클러스터 및 인스턴스를 해당 프록시의 대상이라고 합니다. 편의를 위해 AWS Management 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. 미리 보기 기간 중 RDS Proxy는 RDS MySQL 5.6 및 5.7 및 Aurora MySQL로 구성된 하나의 엔진 패밀리를 지원합니다. MySQL 엔진 패밀리에서 RDS Proxy는 현재 Aurora 프로비저닝된 클러스터, Aurora 병렬 쿼리 클러스터 및 Aurora 글로벌 데이터베이스를 지원합니다. 글로벌 데이터베이스의 경우 기본 AWS 리전에 대한 프록시를 생성할 수 있지만 읽기 전용 보조 AWS 리전에 대해서는 생성할 수 없습니다. RDS Proxy는 현재 Aurora 서비스 클러스터 및 Aurora 다중 마스터 클러스터를 지원하지 않습니다.

연결 폴링

각 프록시는 연결된 RDS 또는 Aurora 데이터베이스의 라이터 인스턴스에 대한 연결 폴링을 수행합니다. 연결 폴링은 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 TLS 핸드셰이킹, 인증, 협상 기능 등과 같은 새 연결을 여는 CPU 오버헤드가 포함됩니다. 연결 폴링은 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 Proxy가 하나의 기본 데이터베이스 연결을 사용하여 한 트랜잭션에 대한 모든 작업을 수행한 후 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 Transport Layer Security/Secure Sockets Layer(TLS/SSL) 및 AWS Identity and Access Management(IAM) 같은 기존의 RDS 보안 메커니즘을 사용합니다. 이러한 보안 기능에 대한 일반적인 내용은 [Amazon Aurora의 보안 \(p. 943\)](#) 단원을 참조하십시오. RDS 및 Aurora가 인증, 권한 부여 및 기타 보안 영역에서 작동하는 방식을 잘 모르는 경우 먼저 해당 리소스를 참조하십시오.

RDS Proxy는 클라이언트 애플리케이션과 기본 데이터베이스 간의 추가 보안 계층의 역할을 할 수 있습니다. 예를 들어 기본 DB 인스턴스가 TLS 1.0 또는 1.1만 지원하더라도 TLS 1.2를 사용하여 프록시에 연결할 수 있습니다. 프록시가 기본 사용자/암호 인증 방법을 사용하여 데이터베이스에 연결하더라도 IAM 역할을 사용하여 프록시에 연결할 수 있습니다. 이 기술을 사용하면 많은 비용을 들여 DB 인스턴스 자체를 마이그레이션 하지 않아도 데이터베이스 애플리케이션에 강력한 인증 요구 사항을 적용할 수 있습니다.

RDS Proxy가 사용하는 데이터베이스 자격 증명을 AWS Secrets Manager에 저장합니다. 프록시가 액세스하는 RDS DB 인스턴스 또는 Aurora DB 클러스터의 각 데이터베이스 사용자는 Secrets Manager에 해당하는 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. 이러한 보안 기능을 애플리케이션 코드에 데이터베이스 자격 증명을 포함하는 대신 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 Proxy를 사용하는 경우 TLS/SSL 인증서를 교체할 때 RDS Proxy 연결을 사용하는 애플리케이션을 업데이트할 필요가 없습니다.

RDS Proxy을 사용하면 세션에서 클라이언트와 RDS Proxy 엔드포인트 간에 TLS/SSL을 사용하도록 할 수 있습니다. RDS Proxy가 이렇게 하도록 하려면 클라이언트 측에서 `--ssl-mode` 파라미터를 이용해 요구 사항을 지정하십시오. SSL 세션 변수는 RDS Proxy 데이터베이스에 대한 SSL 연결에 대해 설정되지 않습니다.

RDS Proxy는 TLS 프로토콜 버전 1.0, 1.1 및 1.2를 지원합니다. 하지만 RDS Proxy 데이터베이스를 TLS용으로 구성할 필요는 없습니다. 특히 SSL 관련 데이터베이스 사용자 권한에서 `REQUIRE` 절을 사용하지 마십시오. 이렇게 하면 사용자가 연결할 수 없습니다.

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 .pem 형식의 CA 를 가리키는 `--ssl-ca` 옵션을 지정하십시오. 사용할 수 있는 .pem 파일을 보려면 Amazon Trust Services 에서 [Amazon 루트 CA 1 트러스트 스토어](#)를 다운로드하십시오.

RDS Proxy는 와일드카드 인증서를 사용합니다. `mysql` 클라이언트를 사용하여 SSL 모드 `VERIFY_IDENTITY`로 연결하는 경우 현재는 MySQL 8.0 호환 `mysql` 명령을 사용해야 합니다.

Failover

장애 조치는 원래 인스턴스를 사용할 수 없게 될 때 데이터베이스 인스턴스를 다른 인스턴스로 대체하는 고 가용성 기능입니다. 장애 조치는 데이터베이스 인스턴스 문제로 인해 발생할 수 있습니다. 또한 데이터베이스 업그레이드를 수행하는 경우와 같은 일반적인 유지 관리 절차의 일부로 발생할 수 있습니다. 장애 조치는 다중 AZ 구성의 RDS DB 인스턴스와 라이터 인스턴스 이외에 하나 이상의 리더 인스턴스가 있는 Aurora DB 클러스터에 적용됩니다.

RDS Proxy를 사용하지 않으면 장애 조치에 잠시 중단이 필요합니다. 중단 중에는 해당 데이터베이스에 대해 쓰기 작업을 수행할 수 없습니다. 모든 기존 데이터베이스 연결이 중단되고 애플리케이션이 해당 연결을 다시 열어야 합니다. 읽기 전용 DB 인스턴스를 사용할 수 없는 인스턴스 대신 사용할 수 있도록 승격하면 새 연결 및 쓰기 작업에 데이터베이스를 사용할 수 있게 됩니다.

프록시를 통해 연결하면 애플리케이션이 데이터베이스 장애 조치를 보다 효율적으로 수행할 수 있습니다. 원래 DB 인스턴스를 사용할 수 없게 되면 RDS Proxy가 유튜 애플리케이션 연결을 끊지 않고 대기 데이터베이스에 연결합니다. 이렇게 하면 장애 조치 프로세스를 단축하고 간소화할 수 있습니다. 그 결과 일반적인 재부팅 또는 데이터베이스 문제보다 애플리케이션 중단을 줄일 수 있는 빠른 장애 조치가 가능합니다.

DB 장애 조치 중에 RDS Proxy는 계속해서 동일한 IP 주소에서 연결을 수락하고 자동으로 연결을 새 마스터로 전달합니다. RDS Proxy를 통해 연결하는 클라이언트는 장애 조치, 로컬 DNS 캐싱, 연결 시간 초과, 어느

DB 인스턴스가 현재 라이터인지에 대한 불확실성 또는 연결을 닫지 않고 사용할 수 없게 된 이전 라이터에서의 쿼리 응답 대기로 인한 DNS 전파 지연에 영향을 받지 않습니다.

자체 연결 풀을 유지 관리하는 애플리케이션의 경우 RDS Proxy를 통해 연결한다는 것은 장애 조치 또는 기타 종단 중에 대부분의 연결이 활성 상태를 유지함을 의미합니다. 트랜잭션 또는 SQL 문 도중에 있는 연결만 취소됩니다. RDS Proxy는 즉시 새 연결을 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. 이는 설치 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.

- `autocommit` 설정을 활성화하면 각 개별 문 종료 후 연결 재사용이 발생할 수 있습니다.
- 반대로, `autocommit` 설정을 비활성화하면 세션에서 실행하는 첫 번째 문이 새 트랜잭션을 시작합니다. 따라서 `SELECT`, `INSERT`, `UPDATE` 및 기타 DML(데이터 조작 언어) 문의 시퀀스를 입력하면 `COMMIT`, `ROLLBACK`를 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.

세션을 다른 연결로 옮기는 것이 비실용적인 데이터베이스 요청을 RDS Proxy가 감지하면 나머지 세션 동안 해당 연결에 대해 멀티플렉싱을 해제합니다. RDS Proxy가 세션에서 멀티플렉싱이 실용적이라는 것을 확신 할 수 없는 경우에도 동일한 규칙이 적용됩니다. 이 작업을 고정이라고 합니다. 고정을 탐지하고 최소화하는 방법은 [고정 \(p. 230\)](#) 단원을 참조하십시오.

RDS Proxy 계획 및 설정

이 문서는 Amazon RDS Proxy에 대한 시험판 설명서입니다. 변경될 수 있습니다.

다음 섹션에서는 RDS Proxy 설정 방법을 찾을 수 있습니다. 각 프록시에 액세스할 수 있는 사용자 및 각 프록시가 DB 인스턴스에 연결하는 방법을 제어하는 관련 보안 옵션을 설정하는 방법도 확인할 수 있습니다.

주제

- [RDS Proxy와 함께 사용할 DB 인스턴스, 클러스터 및 애플리케이션 식별 \(p. 218\)](#)
- [네트워크 사전 요구 사항 설정 \(p. 218\)](#)
- [AWS Secrets Manager에서 데이터베이스 자격 증명 설정 \(p. 218\)](#)
- [AWS Identity and Access Management\(IAM\) 정책 설정 \(p. 219\)](#)
- [RDS Proxy 생성 \(p. 221\)](#)
- [RDS Proxy 보기 \(p. 223\)](#)
- [RDS Proxy 삭제 \(p. 224\)](#)
- [RDS Proxy 수정 \(p. 225\)](#)
- [새 데이터베이스 사용자 추가 \(p. 228\)](#)
- [데이터베이스 사용자 암호 변경 \(p. 229\)](#)

RDS Proxy와 함께 사용할 DB 인스턴스, 클러스터 및 애플리케이션 식별

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 적이 있는 DB 인스턴스 또는 클러스터는 프록시와 연결하기에 좋은 후보입니다. 프록시를 사용하면 애플리케이션은 많은 클라이언트 연결을 열 수 있고, 프록시는 DB 인스턴스 또는 클러스터에 대한 장기 연결을 더 적게 관리합니다.
- T2 또는 T3 같이 더 작은 규모의 AWS 인스턴스 클래스를 사용하는 DB 인스턴스 또는 클러스터의 경우 프록시를 사용하면 메모리 부족 상태를 방지하고 연결을 설정하는 데 필요한 CPU 오버헤드를 줄일 수 있습니다. 이러한 상태는 많은 수의 연결을 처리할 때 발생할 수 있습니다.
- 특정 Amazon CloudWatch 지표를 모니터링하여 DB 인스턴스 또는 클러스터가 연결 수 또는 연결 관리와 연결된 메모리에 대한 제한에 도달하는지 여부를 확인할 수 있습니다. 또한 특정 CloudWatch 지표를 모니터링하여 DB 인스턴스 또는 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 [RDS Proxy 모니터링 \(p. 232\)](#) 단원을 참조하십시오.
- AWS Lambda 함수도 프록시와 함께 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.
- PHP 및 Ruby on Rails와 같은 언어와 프레임워크를 사용하는 애플리케이션은 일반적으로 프록시와 함께 사용하는 것이 좋습니다. 이러한 애플리케이션은 일반적으로 많은 수의 데이터베이스 연결을 열고 닫으며 내장된 연결 풀링 메커니즘이 없습니다.
- 오랜 기간 동안 많은 수의 연결을 열린 상태로 유지하는 애플리케이션은 일반적으로 프록시와 함께 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.
- 모든 DB 인스턴스 및 클러스터에 대해 이러한 인증을 설정하기는 복잡하기 때문에 IAM 인증 Secrets Manager을 채택하지 않았을 수 있습니다. 그렇다면 기존 인증 방법을 그대로 두고 인증을 프록시에 위임 할 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.

네트워크 사전 요구 사항 설정

RDS Proxy를 사용하려면 가상 프라이빗 클라우드(VPC), 둘 이상의 서브넷, 동일한 VPC 내의 EC2 인스턴스, 인터넷 게이트웨이와 같은 네트워킹 리소스 집합이 있어야 합니다. RDS DB 인스턴스 또는 Aurora DB 클러스터에 성공적으로 연결했다면 필요한 네트워크 리소스가 이미 있는 것입니다. RDS 또는 Aurora를 막 시작하는 경우 [Amazon Aurora 환경 설정 \(p. 69\)](#)의 절차에 따라 데이터베이스에 연결하는 기본 사항을 학습하십시오. [Amazon Aurora 시작하기 \(p. 74\)](#)의 자습서를 따를 수도 있습니다.

AWS Secrets Manager에서 데이터베이스 자격 증명 설정

생성하는 각 프록시에 대해 먼저 Secrets Manager 서비스를 사용하여 사용자 이름 및 암호 자격 증명 집합을 저장합니다. 프록시가 RDS DB 인스턴스 또는 Aurora DB 클러스터에서 연결하는 각 데이터베이스 사용자 계정에 대해 별도의 Secrets Manager 비밀을 만듭니다.

Secrets Manager에서는 `username` 및 `password` 필드를 사용하여 이러한 비밀을 만듭니다. 이렇게 하면 프록시가 프록시와 연결된 RDS DB 인스턴스 또는 Aurora DB 클러스터에서 해당 데이터베이스 사용자에게 연결할 수 있습니다. 이렇게 하려면 다른 데이터베이스 자격 증명, RDS 데이터베이스 자격 증명 또는 다른 유형의 비밀 설정을 사용할 수 있습니다.

비밀에 `host` 및 `port` 같은 다른 필드가 존재하는 경우 프록시는 이를 무시합니다. 이러한 세부 정보는 프록시에서 자동으로 제공합니다. 다른 데이터베이스 자격 증명을 선택하는 경우 해당 옵션은 사용자 이름과 암

호를 입력하라는 메시지를 표시하지만 프록시 자체에 대한 설정에서 지정하는 다른 연결 세부 정보는 입력하지 않습니다. 다른 유형의 비밀을 선택하는 경우 `username` 및 `password`라는 키를 사용하여 비밀을 만듭니다.

비밀에 대한 교체 기간을 선택하지 마십시오.

비밀은 특정 데이터베이스 서버에 연결되지 않으므로 여러 데이터베이스 서버에서 동일한 자격 증명을 사용하면 여러 프록시에서 비밀을 다시 사용할 수 있습니다. 예를 들어 개발 및 테스트 서버 그룹에서 동일한 자격 증명을 사용할 수 있습니다.

비밀과 연결된 암호가 올바르지 않은 경우 Secrets Manager에서 연결된 비밀을 업데이트할 수 있습니다. 예를 들어 데이터베이스 계정의 암호가 변경되면 비밀을 업데이트할 수 있습니다. 비밀을 업데이트하기 전까지는 프록시를 통해 해당 데이터베이스 계정에 연결할 수 없습니다. 암호가 비밀에 저장된 자격 증명과 일치하는 다른 계정에는 계속 연결할 수 있습니다.

AWS CLI 또는 RDS API를 통해 프록시를 생성할 때 프록시가 액세스할 수 있는 모든 DB 사용자 계정에 해당 비밀의 Amazon 리소스 이름(ARN)을 지정합니다. AWS Management 콘솔에서는 설명하는 이름으로 비밀을 선택합니다.

Secrets Manager에서 비밀을 만드는 방법에 대한 자세한 내용은 Secrets Manager 설명서의 [비밀 만들기](#) 페이지를 참조하십시오. 다음 기법 중 한 가지를 사용할 수 있습니다. Secrets Manager에서 비밀을 만드는 방법에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 [비밀 만들기](#)를 참조하십시오. 다음 기법 중 한 가지를 사용할 수 있습니다.

- 콘솔에서 [Secrets Manager](#)에 로그인합니다.
- CLI를 사용하여 RDS Proxy에서 사용할 Secrets Manager 비밀을 만들려면 다음과 같은 명령을 사용합니다.

```
aws secretsmanager create-secret
--name "secret_name"
--description "secret_description"
--region region_name
--secret-string "{\"username\":\"db_user\",\"password\":\"db_user_password\"}"
```

예를 들어, 다음 명령은 이름이 `admin` 및 `app-user`인 두 데이터베이스 사용자에 대한 Secrets Manager 비밀을 만듭니다.

```
aws secretsmanager create-secret \
--name $ADMIN_SECRET --description 'db admin user' \
--secret-string '{"username":"admin","password":"choose_your_own_password"}'

aws secretsmanager create-secret \
--name $PROXY_SECRET --description 'application user' \
--secret-string '{"username":"app-user","password":"choose_your_own_password"}'
```

AWS Identity and Access Management(IAM) 정책 설정

Secrets Manager에서 비밀을 만든 후 해당 비밀에 액세스할 수 있는 IAM 정책을 생성합니다.

- 역할 생성 프로세스를 따릅니다. 데이터베이스에 역할 추가 단계를 포함합니다.
- 새 역할의 경우 인라인 정책 추가 단계를 수행합니다. 생성한 비밀의 ARN으로 대체하여 다음 JSON을 볼여넣습니다.

```
{
  "Version": "2012-10-17",
```

```
"Statement": [
    {
        "Sid": "VisualEditor0",
        "Effect": "Allow",
        "Action": [
            "secretsmanager:GetRandomPassword",
            "secretsmanager>CreateSecret",
            "secretsmanager>ListSecrets"
        ],
        "Resource": "*"
    },
    {
        "Sid": "VisualEditor1",
        "Effect": "Allow",
        "Action": "secretsmanager:*",
        "Resource": [
            "your_secret_ARN"
        ]
    }
]
```

- 선택적으로 이 IAM 정책에 대한 신뢰 정책을 편집합니다. 다음 JSON을 복사해 붙여넣습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        },
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "rds.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

다음 명령은 AWS CLI를 통해 동일한 작업을 수행합니다.

```
PREFIX=choose_an_identifier

aws iam create-role --role-name choose_role_name \
--assume-role-policy-document "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow\",\"Principal\":{\"Service\": [\"rds.amazonaws.com\"]},\"Action\":\"sts:AssumeRole\"}]}"
aws iam put-role-policy --role-name same_role_name_as_previous \
--policy-name $PREFIX-secret-reader-policy --policy-document
'{"Version":"2012-10-17","Statement":[{"Sid":"getsecretvalue","Effect":"Allow","Action":["secretsmanager:GetSecretValue","kms:Decrypt"],"Resource":"*"}]}'

aws kms create-key --description "$PREFIX-test-key" --policy "{\"Id\":\"$PREFIX-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\":\"*\"}]}
```

RDS Proxy 생성

지정된 DB 인스턴스 집합에 대한 연결을 관리하려면 프록시를 생성할 수 있습니다. 프록시를 RDS MySQL DB 인스턴스 또는 Aurora MySQL DB 클러스터와 연결할 수 있습니다.

AWS Management 콘솔

프록시를 생성하려면

1. AWS Management 콘솔에 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 모든 설정을 선택합니다. 공개 미리 보기의 경우 일부 선택 사항이 제한되거나 필수입니다. 다음 지침을 참고하십시오.

프록시 구성:

- Proxy identifier(프록시 식별자). AWS 계정 ID와 현재 AWS 리전에서 고유하게 선택한 이름을 지정합니다.
- Engine compatibility(엔진 호환성). 현재 MySQL이 유일한 선택 항목입니다.
- Require Transport Layer Security(전송 계층 보안 필요). 프록시가 모든 클라이언트 연결에 TLS/SSL을 적용하도록 하려면 이 설정을 선택합니다. 프록시에 암호화된 연결 또는 암호화되지 않은 연결을 사용하는 경우 프록시는 기본 데이터베이스에 연결할 때 동일한 암호화 설정을 사용합니다.
- Idle client connection timeout(유지 클라이언트 연결 시간 초과). 프록시가 연결을 종료하기 전에 클라이언트 연결이 유지 상태를 유지할 수 있는 기간을 선택합니다. 애플리케이션이 이전 요청이 완료된 후 지정된 시간 내에 새 요청을 제출하지 않으면 클라이언트 연결이 유지 상태로 간주됩니다. 기본 데이터베이스 연결은 열린 상태를 유지하고 연결 풀로 반환됩니다. 따라서 새 클라이언트 연결에 다시 사용할 수 있습니다.

대상 그룹 구성:

- 데이터베이스. 이 프록시를 통해 액세스할 RDS DB 인스턴스 또는 Aurora DB 클러스터를 하나 선택합니다. 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. 그런 다음 프록시를 다시 생성해보십시오.
- Connection pool maximum connections(연결 풀 최대 연결). 1-100 사이의 값을 지정합니다. 이 설정은 RDS Proxy가 연결에 사용할 수 있는 max_connections 값의 백분율을 나타냅니다. 이 DB 인스턴스 또는 클러스터에 하나의 프록시만 사용하려는 경우 100을 설정할 수 있습니다. RDS Proxy가 이 설정을 사용하는 방법에 대한 자세한 내용은 [연결 제한 및 시간 초과 \(p. 230\)](#) 단원을 참조하십시오.
- Session pinning filters(세션 고정 필터). 이 설정은 특정 애플리케이션의 성능 문제를 해결하기 위한 고급 설정입니다. 현재 EXCLUDE_VARIABLE_SETS가 유일한 선택 항목입니다. 애플리케이션이 특정 종류의 SQL 문으로 인해 연결을 재사용하지 않고 이러한 SQL 문에서 연결을 재사용하는 것이 애플리케이션의 정확성에 영향을 주지 않는지 확인할 수 있는 경우에만 필터를 선택합니다.
- Connection borrow timeout(연결 대여 시간 초과). 프록시가 사용 가능한 모든 연결을 사용하는 경우가 있을 경우 시간 초과 오류를 반환하기 전에 프록시가 연결을 사용할 수 있을 때까지 기다리는 시간을 지정할 수 있습니다. 이 설정은 프록시가 이미 최대 연결 수를 사용 중인 경우에만 적용됩니다.

연결:

- Secrets Manager ARNs(비밀 관리자 ARN). 이 프록시로 액세스하려는 RDS DB 인스턴스 또는 Aurora DB 클러스터와 연결된 Secrets Manager 비밀을 하나 이상 선택합니다.
- IAM 역할. 앞서 선택한 Secrets Manager 비밀에 액세스할 수 있는 권한이 있는 IAM 역할을 선택합니다. AWS Management 콘솔을 선택하여 새 IAM 역할을 만들어 사용할 수도 있습니다.
- IAM Authentication(IAM 인증). 프록시 연결에 대해 IAM 인증을 요구하지 아니면 허용하지 않을지 선택합니다.
- 서브넷. 이 필드는 VPC와 연결된 모든 서브넷으로 미리 채워집니다. 이 프록시에 필요하지 않은 서브넷을 모두 제거합니다. 서브넷은 두 개 이상 남겨 두어야 합니다.

추가 연결 구성:

- VPC 보안 그룹. 기존 VPC 보안 그룹을 선택합니다. AWS Management 콘솔을 선택하여 새 보안 그룹을 생성하여 사용할 수도 있습니다.

고급 구성:

- Enable enhanced logging(고급 로깅 사용). 프록시 호환성 또는 성능 문제를 해결하려면 이 설정을 사용하도록 설정할 수 있습니다. 이 설정을 사용하도록 설정하면 RDS Proxy는 SQL 문에 대한 자세한 정보를 로그에 포함합니다. 이 정보는 SQL 동작 또는 프록시 연결의 성능 및 확장성과 관련된 문제를 디버깅하는 데 도움이 됩니다. 디버그 정보에는 프록시를 통해 제출하는 SQL 문의 텍스트가 포함됩니다. 따라서 디버깅에 필요한 경우에만 그리고 로그에 나타나는 중요한 정보를 보호하기 위한 보안 조치가 있는 경우에만 이 설정을 사용하도록 설정합니다.
- 5. RDS Proxy is now available in internal Preview(이제 미리 보기에서 사용 가능)을 선택하여 미리 보기에 대한 사용 약관을 승인합니다.
- 6. Create proxy(프록시 생성)를 선택합니다.

AWS CLI

DB 인스턴스를 생성하려면 AWS CLI 명령 `create-db-proxy`를 사용합니다.

Example

Linux, OS X, Unix의 경우:

```
aws rds create-db-proxy \
--db-proxy-name proxy_name \
--role-arn iam_role \
--engine-family { MySQL } \
--vpc-subnet-ids space_separated_list \
[--vpc-security-group-ids space_separated_list] \
[--auth ProxyAuthenticationConfig_JSON_string] \
[--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 ^
```

```
--role-arn iam_role ^
--engine-family { MYSQL } ^
--vpc-subnet-ids space_separated_list ^
[--vpc-security-group-ids space_separated_list] ^
[--auth ProxyAuthenticationConfig_JSON_string] ^
[--require-tls | --no-require-tls] ^
[--idle-client-timeout value] ^
[--debug-logging | --no-debug-logging] ^
[--tags comma_separated_list]
```

프록시에 필요한 정보 및 연결을 생성하려면 다음 명령도 사용합니다.

```
aws rds create-db-proxy-target-group
  --name target_group_name
  --db-proxy-name proxy_name
  --connection-pool-defaults ConnectionPoolConfiguration
  --tags comma_separated_list
```

```
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), or
  [--db-cluster-endpoint endpoint_name]          # rds db cluster endpoint (all
  instances)
```

RDS API

RDS Proxy를 생성하려면 Amazon RDS API 함수 [CreateDBProxy](#)를 호출합니다. [AuthConfig](#) 데이터 구조와 함께 파라미터를 전달합니다.

또한 [CreateDBProxyTargetGroup](#) 함수를 호출하여 연결된 목표 그룹을 생성합니다. 그런 다음 [RegisterDBProxyTargets](#) 함수를 호출하여 RDS DB 인스턴스 또는 Aurora DB 클러스터를 대상 그룹에 연결합니다.

RDS Proxy 보기

하나 이상의 RDS 프록시를 생성한 후 모든 프록시를 보고 구성 세부 정보를 검토한 후 수정, 삭제할 항목을 선택할 수 있습니다.

특히 프록시를 사용하는 데이터베이스 애플리케이션의 연결 문자열에 사용할 프록시의 엔드포인트가 필요합니다.

AWS Management 콘솔

RDS Proxy를 보려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management 콘솔의 상단 오른쪽에서 RDS Proxy를 생성한 AWS 리전을 선택합니다.
3. 탐색 창에서 Proxies(프록시)를 선택합니다.
4. 세부 정보를 표시할 RDS 프록시의 이름을 선택합니다.
5. 세부 정보 페이지의 대상 그룹 섹션에 프록시가 특정 RDS DB 인스턴스 또는 Aurora DB 클러스터와 어떻게 연결되는지 표시됩니다. 기본 대상 그룹 페이지에 대한 링크를 따라 프록시와 데이터베이스 간 연결의 세부 정보를 볼 수 있습니다. 이 페이지에서는 최대 연결 비율, 연결 대여 시간 초과, 엔진 호환성, 세션 고정 필터 등 프록시를 생성할 때 지정한 설정을 볼 수 있습니다.

CLI

CLI를 사용하여 RDS Proxy를 보려면 `describe-db-proxies` 명령을 사용합니다. 기본적으로 AWS 계정이 소유한 모든 프록시가 표시됩니다. `--db-proxy-name` 파라미터를 지정하여 단일 프록시의 세부 정보를 볼 수 있습니다.

```
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.

- 프록시 목록을 가져오려면 `describe-db-proxies`를 실행합니다.
- 프록시의 이름을 파라미터로 사용하여 `describe-db-proxy-target-groups --db-proxy-name`을 실행합니다. 이 출력은 프록시가 사용할 수 있는 최대 연결 백분율과 같은 연결 파라미터를 표시합니다.
- `describe-db-proxy-targets`를 실행하여 반환된 대상 그룹과 연결된 RDS DB 인스턴스 또는 Aurora DB 클러스터의 세부 정보를 확인합니다.

RDS API

RDS API를 사용하여 프록시를 보려면 [DescribeDBProxies](#) 작업을 사용합니다. 이 작업은 [DBProxy](#) 데이터 형식의 값을 반환합니다.

[DescribeDBProxyTargetGroups](#) 작업을 사용하면 반환 값의 프록시 식별자를 사용하여 프록시 연결 설정의 세부 정보를 볼 수 있습니다. 이 작업은 [DBProxyTargetGroup](#) 데이터 형식의 값을 반환합니다.

[DescribeDBProxyTargets](#) 작업을 사용하면 프록시와 연결된 RDS 인스턴스 또는 Aurora DB 클러스터를 볼 수 있습니다. 이 작업은 [DBProxyTarget](#) 데이터 형식의 값을 반환합니다.

RDS Proxy 삭제

더 이상 필요하지 않은 프록시는 삭제할 수 있습니다. 프록시를 사용 중인 애플리케이션이 더 이상 관련이 없기 때문에 프록시를 삭제할 수 있습니다. 또는 프록시와 연결된 DB 인스턴스 또는 클러스터를 서비스 중단 상태로 전환하면 프록시를 삭제할 수 있습니다.

AWS Management 콘솔

프록시를 삭제하려면

1. AWS Management 콘솔에 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-proxy](#)를 사용합니다.

Example

```
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 [DeleteDBProxy](#)를 호출합니다. 관련 항목 및 연결을 삭제하려면 [DeleteDBProxyTargetGroup](#) 및 [DeregisterDBProxyTargets](#) 함수도 호출합니다.

RDS Proxy 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 콘솔

프록시에 대한 설정을 수정하려면

1. AWS Management 콘솔에 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.
 - 새 식별자를 입력하여 프록시 이름을 바꿀 수 있습니다.
 - TLS(전송 계층 보안) 요구 사항을 설정하거나 해제할 수 있습니다.
 - 유지 연결 시간 초과의 기간을 입력할 수 있습니다.
 - Secrets Manager 비밀을 추가하거나 제거할 수 있습니다. 이러한 비밀은 데이터베이스 사용자 이름 및 암호에 해당합니다.
 - Secrets Manager에서 비밀을 검색하는 데 사용되는 IAM 역할을 변경할 수 있습니다.
 - 프록시 연결에 대해 IAM 인증을 요구하거나 허용하지 않을 수 있습니다.
 - 프록시가 사용할 VPC 서브넷을 추가하거나 제거할 수 있습니다.
 - 프록시가 사용할 VPC 보안 그룹을 추가하거나 제거할 수 있습니다.
 - 고급 로깅을 사용하거나 사용하지 않도록 설정할 수 있습니다.
6. [Modify]를 선택합니다.

변경하려는 설정을 찾지 못한 경우 다음 절차를 계속하여 프록시의 대상 그룹을 업데이트합니다. 각 프록시에는 하나의 연결된 대상 그룹이 있으며, 이 그룹은 프록시와 함께 자동으로 생성됩니다.

프록시 대상 그룹에 대한 설정을 수정하려면

프록시와 연결된 대상 그룹은 물리적 데이터베이스 연결과 관련된 설정을 제어합니다. 대상 그룹은 프록시 세부 정보 페이지에서만 수정할 수 있으며 Proxies(프록시) 페이지의 목록에서는 수정할 수 없습니다.

1. Proxies(프록시) 페이지에서 프록시의 세부 정보 페이지로 이동합니다.

2. 대상 그룹에서 default 링크를 선택합니다. 현재 모든 프록시에는 default라는 단일 대상 그룹이 있습니다.
3. 기본 대상 그룹에 대한 세부 정보 페이지에서 수정을 선택합니다.
4. 대상 그룹 식별자, 데이터베이스 엔진과 같은 특정 속성은 고정되어 있습니다. 이러한 속성은 변경할 수 없습니다.

수정할 수 있는 속성의 새 설정을 선택합니다.

- 다른 RDS DB 인스턴스 또는 Aurora 클러스터를 선택할 수 있습니다.
- 프록시에서 사용할 수 있는 최대 연결 백분율을 조정할 수 있습니다.
- 세션 고정 필터를 선택할 수 있습니다. 이 설정을 사용하면 트랜잭션 수준 연결 재사용이 불충분하여 발생하는 성능 문제를 줄일 수 있습니다. 이 설정을 사용하려면 애플리케이션 동작과 RDS Proxy가 세션을 데이터베이스 연결에 고정하는 상황을 이해해야 합니다.
- 연결 대여 시간 초과 간격을 조정할 수 있습니다. 이 설정은 프록시에서 이미 최대 연결 수를 사용 중인 경우 시간 초과 오류를 반환하기 전에 프록시가 연결을 사용할 수 있을 때까지 기다리는 시간을 결정합니다.

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 사용하면 프록시에서 사용하는 Secrets Manager 비밀 집합, TLS 필요 여부, 유 휴 클라이언트 시간 초과, 디버깅을 위해 SQL 문에서 추가 정보를 기록할지 여부, Secrets Manager 비밀을 검색하는 데 사용되는 IAM 역할, 프록시에서 사용하는 보안 그룹 등의 속성을 변경할 수 있습니다.

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
```

`modify-db-proxy-target-group` 명령을 사용하여 연결 관련 설정을 수정하거나 대상 그룹의 이름을 바꿀 수 있습니다. 현재 모든 프록시에는 default라는 단일 대상 그룹이 있습니다. 이 대상 그룹으로 작업하는 경우 프록시 이름을 지정하고 대상 그룹 이름에 default를 지정합니다.

다음 예제에서는 대상 그룹을 사용하여 프록시에 대한 `MaxConnectionsPercent` 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 RDS DB 인스턴스 또는 Aurora DB 클러스터를 변경할 수 있습니다. 현재 각 프록시는 하나의 RDS DB 인스턴스 또는 Aurora DB 클러스터에 연결할 수 있습니다. 대상 그룹은 다중 AZ 구성의 모든 RDS DB 인스턴스 또는 Aurora 클러스터의 모든 DB 인스턴스에 대한 연결 세부 정보를 추적합니다.

다음 예제는 프록시가 cluster-56-2019-11-14-1399라는 Aurora 클러스터와 연결된 상태에서 시작합니다. 이 예제에서는 provisioned-cluster라는 다른 클러스터에 연결할 수 있도록 프록시를 변경하는 방법을 보여 줍니다. RDS DB 인스턴스로 작업하는 경우 --db-instance-identifier 옵션을 지정합니다. Aurora DB 클러스터로 작업하는 경우 대신 --db-cluster-identifier 옵션을 지정합니다.

```
$ 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-56-2019-11-14-1399"
        },
        {
            "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-56-2019-11-14-1399

$ 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 프록시에서 사용하는 Secrets Manager 비밀 집합, TLS 필요 여부, 유축 클라우드 이언트 시간 초과, 디버깅을 위해 SQL 문에서 추가 정보를 기록할지 여부, Secrets Manager 비밀을 검색하는 데 사용되는 IAM 역할, 프록시에서 사용하는 보안 그룹 등의 속성을 변경할 수 있습니다.

[ModifyDBProxyTargetGroup](#)을 사용하여 연결 관련 설정을 수정하거나 대상 그룹의 이름을 바꿀 수 있습니다. 현재 모든 프록시에는 default라는 단일 대상 그룹이 있습니다. 이 대상 그룹으로 작업하는 경우 프록시 이름을 지정하고 대상 그룹 이름에 default를 지정합니다.

[DeregisterDBProxyTargets](#) 및 [RegisterDBProxyTargets](#)를 사용하여 대상 그룹을 통해 프록시가 연결된 RDS DB 인스턴스 또는 Aurora DB 클러스터를 변경할 수 있습니다. 현재 각 프록시는 하나의 RDS DB 인스턴스 또는 Aurora DB 클러스터에 연결할 수 있습니다. 대상 그룹은 다중 AZ 구성의 모든 RDS DB 인스턴스 또는 Aurora 클러스터의 모든 DB 인스턴스에 대한 연결 세부 정보를 추적합니다.

새 데이터베이스 사용자 추가

프록시와 연결된 RDS DB 인스턴스 또는 Aurora 클러스터에 새 데이터베이스 사용자를 추가해야 할 경우가 있습니다. 그렇다면 Secrets Manager 비밀을 추가하거나 재사용하여 해당 사용자의 자격 증명을 저장합니다. 이렇게 하려면 다음 옵션 중 하나를 선택합니다.

- [AWS Secrets Manager에서 데이터베이스 자격 증명 설정 \(p. 218\)](#)에 설명된 절차를 사용하여 새 Secrets Manager 비밀을 만들습니다.
- 새 사용자가 기존 사용자를 대신하는 경우 기존 사용자의 프록시 Secrets Manager 비밀에 저장된 자격 증명을 업데이트합니다.

데이터베이스 사용자 암호 변경

프록시와 연결된 RDS DB 인스턴스 또는 Aurora 클러스터에서 데이터베이스 사용자의 암호를 변경해야 할 경우가 있습니다. 그렇다면 해당 Secrets Manager 비밀을 새 암호로 업데이트합니다.

RDS Proxy를 통해 데이터베이스에 연결

이 문서는 Amazon RDS Proxy에 대한 시험판 설명서입니다. 변경될 수 있습니다.

일반적으로 데이터베이스에 직접 연결하는 것과 동일한 방식으로 프록시를 통해 RDS DB 인스턴스 또는 Aurora DB 클러스터에 연결합니다. 가장 큰 차이점은 인스턴스 또는 클러스터 엔드포인트 대신 프록시 엔드포인트를 지정한다는 것입니다.

Aurora DB 클러스터의 경우 모든 프록시 연결은 읽기-쓰기 기능이 있으며 라이터 인스턴스를 사용합니다. 읽기 전용 연결에 리더 엔드포인트를 사용하는 경우 동일한 방법으로 리더 엔드포인트를 계속 사용합니다.

기본 인증을 사용하여 프록시에 연결

기본 인증을 사용하여 프록시에 연결하려면 다음 일반 절차를 따릅니다.

- 프록시 엔드포인트를 찾습니다. AWS Management 콘솔에서는 해당 프록시의 세부 정보 페이지에서 엔드포인트를 찾을 수 있습니다. AWS CLI를 사용하는 경우 `describe-db-proxies` 명령을 사용할 수 있습니다. 다음 예제에서는 이 작업을 수행하는 방법을 보여 줍니다.

```
# 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"
        }
    ]
]
```

- 클라이언트 애플리케이션의 연결 문자열에서 해당 엔드포인트를 호스트 파라미터로 지정합니다. 예를 들어 프록시 엔드포인트를 `mysql -h` 옵션의 파라미터로 지정합니다.
- 평소와 동일한 데이터베이스 사용자 이름과 암호를 제공합니다.

IAM 인증을 사용하여 프록시에 연결

IAM 인증을 사용하여 RDS DB 인스턴스 또는 Aurora 클러스터에 연결하는 동일한 일반 절차를 따릅니다. 인스턴스, 클러스터 또는 리더 엔드포인트 대신 프록시 엔드포인트를 지정합니다. 이 절차에 대한 자세한 내용은 [명령줄에서 DB 인스턴스에 연결: AWS CLI 및 mysql 클라이언트](#)를 참조하십시오.

RDS Proxy 관리

다음에서는 애플리케이션이 데이터베이스 연결을 가장 효율적으로 사용하고 최대 연결 재사용을 달성하는 데 도움이 되는 RDS Proxy 작업 및 구성의 측면에 대해 설명합니다. 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.

연결 제한 및 시간 초과

RDS Proxy는 RDS DB 인스턴스 또는 Aurora DB 클러스터에 대한 `max_connections` 설정을 사용합니다. 이 설정은 프록시가 한 번에 열 수 있는 연결의 전체 상한을 나타냅니다. Aurora 클러스터 및 RDS 다중 AZ 구성에서 프록시가 사용하는 `max_connections` 값은 Aurora 기본 인스턴스 또는 RDS 라이터 인스턴스에 대한 값입니다.

최대 연결에 대한 프록시 설정은 `max_connections` 값의 백분율을 나타냅니다. 여러 애플리케이션이 모두 동일한 데이터베이스를 사용하는 경우 `max_connections`의 특정 백분율을 갖는 각 애플리케이션에 대해 프록시를 사용하여 연결 할당량을 효과적으로 분배할 수 있습니다. 이렇게 할 경우 동일한 데이터베이스와 연결된 모든 프록시에 대해 백분율이 최대 100 이하인지 확인합니다.

RDS Proxy는 유휴 연결을 주기적으로 끊고 연결 풀로 반환합니다. 이 시간 초과 간격을 조정할 수 있습니다. 이렇게 하면 애플리케이션이 기한 경과 리소스를 처리하는 데 도움이 됩니다. 특히 애플리케이션이 잘못 하여 중요한 데이터베이스 리소스를 보유하면서 연결을 열어 둔 경우 더 그렇습니다.

연결 풀링

[연결 풀링 \(p. 215\)](#)에 설명된 대로 연결 풀링은 중요한 RDS Proxy 기능입니다. 다음에는 연결 풀링 및 트랜잭션 수준 연결 재사용(멀티플렉싱)을 가장 효율적으로 사용하는 방법을 알아봅니다.

연결 풀은 RDS Proxy에서 관리하므로 애플리케이션 코드를 변경하지 않고도 연결 풀을 모니터링하고 연결 제한 및 시간 초과 간격을 조정할 수 있습니다.

각 프록시마다 연결 풀에서 사용하는 연결 수에 대한 상한을 지정할 수 있습니다. 제한은 백분율로 지정합니다. 이 백분율은 데이터베이스에 구성된 최대 연결에 적용됩니다. 정확한 수는 DB 인스턴스 크기 및 구성 설정에 따라 다릅니다. 예를 들어 데이터베이스에 대한 최대 연결의 75%를 사용하도록 RDS Proxy를 구성했다고 가정합니다. MySQL의 경우 최대 값은 `max_connections` 구성 파라미터에 의해 정의됩니다. 이 경우 최대 연결의 나머지 25%는 다른 프록시에 할당하거나 프록시를 통과하지 않는 연결에 사용할 수 있습니다. 데이터베이스에 동시에 연결이 많지 않거나 일부 연결이 오랫동안 유휴 상태로 유지되는 경우 프록시는 특정 시점에 열린 연결을 최대 연결의 75% 미만으로 유지할 수 있습니다.

RDS DB 인스턴스 또는 Aurora MySQL 클러스터에 적용되는 `max_connections` 구성 설정을 업데이트하면 연결 풀에 사용할 수 있는 전체 연결 수가 변경됩니다.

프록시는 이러한 연결을 모두 미리 예약하지는 않습니다. 따라서 비교적 큰 백분율을 지정할 수 있으며 이러한 연결은 프록시가 필요로 할 때만 열립니다.

애플리케이션이 연결을 사용할 수 있을 때까지 기다리는 시간을 선택할 수 있습니다. 이 설정은 프록시를 생성할 때 `Connection borrow timeout`(연결 대여 시간 초과) 옵션으로 표시됩니다. 이 설정은 시간 초과 오류를 반환하기 전에 연결 풀에서 연결을 사용할 수 있을 때까지 기다리는 시간을 지정합니다. 연결 수가 최대일 때, 즉 연결 풀에서 사용할 수 있는 연결이 없을 때 적용됩니다. 또한 장애 조치 작업이 진행 중이기 때문에 사용 가능한 라이터 인스턴스가 없는 경우에도 적용됩니다. 이 설정을 사용하면 애플리케이션 코드에서 쿼리 시간 초과를 변경하지 않고도 애플리케이션에 가장 적합한 대기 기간을 제한할 수 있습니다.

고정

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 세션 이 종료할 때까지 동일한 기본 데이터베이스 연결을 사용합니다. RDS Proxy는 다른 세션에 적합하지 않은 세션 상태 변경을 감지하면 자동으로 클라이언트 연결을 특정 DB 연결에 고정합니다. 고정은 연결 재사용의 효과를 줄입니다. 모든 또는 거의 모든 연결이 고정되는 경우 애플리케이션 코드 또는 워크로드를 수정하여 고정을 유발하는 조건을 줄일 수 있습니다.

예를 들어 애플리케이션이 세션 변수 또는 구성 파라미터를 변경하는 경우 이후의 문이 새 변수 또는 파라미터를 사용하여 유효해질 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. 이 규칙은 사용자가 설정할 수 있는 모든 파라미터에 적용되는 것입니다. RDS Proxy는 문자 집합, 데이터 정렬, 시간대, 자동 커밋, 현재 데이터베이스, SQL 모드 및 session_track_schema 설정에 대한 변경 사항을 주목합니다. RDS Proxy는 이들을 수정할 때 세션을 고정하지 않습니다. 이 경우 RDS Proxy는 해당 설정에 대해 동일한 값을 가진 다른 세션에 대해서만 연결을 재사용합니다.

RDS Proxy 성능 투닝에는 고정을 최소화하여 트랜잭션 수준 연결 재사용(멀티플렉싱)을 최대화하려는 시도가 포함됩니다. 이를 위해 다음과 같은 기법을 사용할 수 있습니다.

- 고정을 일으킬 수 있는 불필요한 데이터베이스 요청을 피합니다.
- 모든 연결에서 변수와 구성 설정을 일관되게 설정합니다. 그렇게 하면 이후 세션에서 특정 설정이 있는 연결을 재사용할 가능성이 증가합니다.
- 프록시에 세션 고정 필터를 적용합니다. 이렇게 해도 애플리케이션의 올바른 작동에 영향을 주지 않음이 확인될 경우 특정 유형의 작업을 세션 고정에서 제외할 수 있습니다.
- CloudWatch 지표 DatabaseConnectionsCurrentlySessionPinned를 모니터링하여 고정 빈도를 확인합니다. 이 지표와 기타 CloudWatch 지표에 대한 자세한 내용은 [RDS Proxy 모니터링 \(p. 232\)](#) 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. AWS CLI 및 RDS API를 통해 초기화 쿼리 작업을 수행할 수 있습니다. 현재는 AWS Management 콘솔을 통해 이 속성을 설정할 수 없습니다.

다음과 같이 멀티플렉싱으로 예기치 않은 동작이 발생할 수 있는 상황에서 프록시는 세션을 현재 연결에 고정합니다.

- 텍스트 크기가 4KB를 초과하면 프록시가 세션을 고정합니다.
- 준비된 문을 사용하면 프록시가 세션을 고정합니다. 이 규칙은 준비된 문이 SQL 텍스트를 사용하는지 이진 프로토콜을 사용하는지 여부를 적용합니다.
- 명시적 LOCK TABLE, LOCK TABLES 또는 FLUSH TABLES WITH READ LOCK 문을 사용하면 프록시가 세션을 고정합니다.
- 사용자 변수 또는 시스템 변수(일부 예외)를 설정하면 프록시가 세션을 고정합니다. 이러한 상황으로 인해 연결 재사용이 너무 줄어들면 SET 작업에서 고정이 발생하지 않도록 선택할 수 있습니다.
- 임시 테이블을 생성하면 프록시가 세션을 고정합니다. 이렇게 하면 트랜잭션 경계에 관계없이 임시 테이블의 내용이 세션 전체에서 보존됩니다.
- ROW_COUNT(), FOUND_ROWS() 또는 LAST_INSERT_ID() 함수를 호출하면 고정이 발생하는 경우도 있고 그렇지 않은 경우도 있습니다. 이 동작은 MySQL 5.6 및 MySQL 5.7과 호환되는 Aurora MySQL 버전마다 다를 수 있습니다.

저장 프로시저 및 저장 함수를 호출해도 고정이 발생하지 않습니다. RDS Proxy는 이러한 호출로 인한 세션 상태 변경을 감지하지 못합니다. 따라서 애플리케이션이 저장된 루틴 내에서 세션 상태를 변경하지 않고 해

당 세션 상태에 의존하여 트랜잭션 간에 지속되어야 합니다. 예를 들어 저장 프로시저가 트랜잭션 간에 지속되도록 의도된 임시 테이블을 생성하는 경우 현재 해당 애플리케이션은 RDS Proxy와 호환되지 않습니다.

애플리케이션 동작에 대한 전문 지식이 있는 경우 특정 애플리케이션 문에서 고정 동작을 건너뛰도록 선택할 수 있습니다. 그러려면 프록시를 생성할 때 Session pinning filters(세션 고정 필터) 옵션을 선택하면 됩니다. 현재 다음 종류의 애플리케이션 문에 대해 세션 고정을 옵트아웃 할 수 있습니다.

- 세션 변수 및 구성 설정 설정

특정 프록시에서 고정이 발생하는 빈도에 대한 지표를 보려면 [RDS Proxy 모니터링 \(p. 232\)](#) 단원을 참조하십시오.

RDS Proxy 모니터링

이 문서는 Amazon RDS Proxy에 대한 시험판 설명서입니다. 변경될 수 있습니다.

Amazon CloudWatch를 사용하여 RDS Proxy를 모니터링할 수 있습니다. CloudWatch는 프록시에서 원시 데이터를 수집하여 읽기 가능하며 실시간에 가까운 지표로 처리합니다. CloudWatch 콘솔에서 이러한 지표를 찾으려면 지표를 선택한 다음 RDS, Per-Proxy Metrics(프록시별 지표)를 차례로 선택합니다.

Note

RDS는 프록시와 연결된 각 기본 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.

모든 RDS Proxy 지표는 그룹 proxy에 있으며 차원은 ProxyName입니다.

지표	유효 기간	설명
ClientConnectionsReceived	1분 이상	수신된 클라이언트 연결 요청 수입니다. 이 지표에 가장 유용한 통계는 합계입니다.
ClientConnectionsSetupSucceeded	1분 이상	TLS를 사용하거나 사용하지 않는 인증 메커니즘을 통해 성공적으로 설정된 클라이언트 연결 수입니다. 이 지표에 가장 유용한 통계는 합계입니다.
ClientConnectionsSetupFailedAuth	1분 이상	잘못된 인증 또는 TLS 구성 오류로 인해 실패한 클라이언트 연결 시도 횟수입니다. 이 지표에 가장 유용한 통계는 합계입니다.
ClientConnectionsClosed	1분 이상	닫은 클라이언트 연결 수입니다. 이 지표에 가장 유용한 통계는 합계입니다.

지표	유효 기간	설명
ClientConnections	1분	현재 클라이언트 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 합계입니다.
QueryRequests	1분 이상	수신된 쿼리 수입니다. 참고: 다중 문 쿼리는 하나의 쿼리로 계산됩니다. 이 지표에 가장 유용한 통계는 합계입니다.
DatabaseConnectionRequests	1분 이상	데이터베이스 연결을 생성하기 위한 요청 수입니다. 이 지표에 가장 유용한 통계는 합계입니다.
DatabaseConnectionsSetupSucceeded	1분 이상	TLS를 사용하거나 사용하지 않고 성공적으로 설정된 데이터베이스 연결 수입니다. 이 지표에 가장 유용한 통계는 합계입니다.
DatabaseConnectionsSetupFailed	1분 이상	실패한 데이터베이스 연결 요청 수입니다. 이 지표에 가장 유용한 통계는 합계입니다.
MaxDatabaseConnectionsAllowed	1분	허용되는 최대 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 합계입니다.
DatabaseConnections	1분	현재 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 합계입니다.
DatabaseConnectionsCurrentlyBorrowed	1분	현재 대여 상태인 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 합계입니다.
DatabaseConnectionsCurrentlySessionPinned	1분	클라이언트 요청의 세션 상태 변경 작업으로 인해 현재 고정된 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 합계입니다.

지표	유효 기간	설명
DatabaseConnectionsCurrentlyInTransaction	1분	트랜잭션의 현재 데이터베이스 연결 수입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 합계입니다.
TargetGroupWriterAvailableDuration	1분	1분 기간 동안 대상 그룹에 쓰기가 있는 시간(초)입니다. 이 지표는 1분마다 보고됩니다. 이 지표에 가장 유용한 통계는 Average(평균)입니다.

RDS Proxy 제한 사항

이 문서는 Amazon RDS Proxy에 대한 시험판 설명서입니다. 변경될 수 있습니다.

공개 미리 보기 중인 RDS Proxy에는 다음과 같은 제한 사항이 적용됩니다.

- 공개 미리 보기하는 다음 AWS 리전에서만 사용할 수 있습니다. 미국 동부(버지니아 북부), 미국 동부(오하이오), 미국 서부(오레곤), 아시아 태평양(도쿄), 유럽(아일랜드).
- 각 AWS 계정 ID에 대해 최대 20개의 프록시를 보유할 수 있습니다.
- Aurora 클러스터에서 연결 풀의 모든 연결은 Aurora 기본 인스턴스에서 처리됩니다. 읽기 집약적 워크로드에 대해 로드 밸런싱을 수행하려면 여전히 Aurora 클러스터에 대해 리더 엔드포인트를 직접 사용합니다.
- 현재 RDS Proxy는 MySQL 엔진 패밀리에서만 사용할 수 있습니다. 이 엔진 패밀리에는 RDS MySQL 5.6 및 5.7 그리고 Aurora 버전 1 및 2가 포함됩니다. 공개 미리 보기에서는 프록시가 PostgreSQL 데이터베이스를 지원하지 않습니다.
- RDS MySQL 8.0에는 RDS Proxy를 사용할 수 없습니다.
- Aurora 서비스 클러스터에는 RDS Proxy를 사용할 수 없습니다.
- Aurora 다중 마스터 클러스터에는 RDS Proxy를 사용할 수 없습니다.
- Amazon RDS MySQL 및 RDS Proxy에는 Aurora MySQL을 사용할 수 있습니다. EC2 인스턴스에서 실행되는 자체 관리형 MySQL 데이터베이스에는 사용할 수 없습니다.
- 현재 모든 프록시는 포트 3306에서 수신 대기합니다.
- RDS Proxy는 데이터베이스와 동일한 VPC에 있어야 합니다. 전용 테넌시가 있는 VPC는 사용할 수 없습니다. 데이터베이스에는 공개적으로 액세스할 수는 있지만 프록시에는 액세스할 수 없습니다.
- SQL 문 및 함수를 기반으로 세션을 데이터베이스 연결에 고정하기 위해 모든 논리가 구현된 것은 아닙니다. 최신 고정 동작은 [고정 \(p. 230\)](#) 단원을 참조하십시오.
- 프록시는 압축 모드를 지원하지 않습니다. 예를 들어 mysql 명령의 --compress 또는 -c 옵션에서 사용하는 압축을 지원하지 않습니다.

RDS Proxy용 명령줄 예제

이 문서는 Amazon RDS Proxy에 대한 시험판 설명서입니다. 변경될 수 있습니다.

mysql 연결 명령과 SQL 문의 조합이 어떻게 RDS Proxy와 상호 작용하는지 보려면 다음 예제를 살펴보십시오.

Example 장애 조치 시 연결 유지

이 예제에서는 데이터베이스를 재부팅하거나 데이터베이스가 문제로 인해 사용할 수 없게 되는 경우와 같은 장애 조치 중에 어떻게 열린 연결이 계속 작동하는지 보여 줍니다. 이 예제에서는 the-proxy라는 프록시와 DB 인스턴스 instance-8898 및 instance-9814가 있는 Aurora DB 클러스터를 사용합니다. Linux 명령 줄에서 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-id 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-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-id 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 5.6 또는 5.7과 호환되는 클러스터의 기본 파라미터 설정을 기반으로 자체 DB 클러스터 파라미터 그룹을 생성합니다. max_connections 설정에 값을 지정하여 기본값을 설정하는 공식을 재지정 합니다. DB 클러스터 파라미터 그룹을 자체 DB 클러스터와 연결합니다.

```
export REGION=us-east-1  
export CLUSTER_PARAM_GROUP=rds-proxy-mysql-56-max-connections-demo  
export CLUSTER_NAME=rds-proxy-mysql-56  
  
aws rds create-db-parameter-group --region $REGION \  
  --db-parameter-group-family aurora5.6 \  
  --db-parameter-group-name $CLUSTER_PARAM_GROUP \  
  --description "Aurora MySQL 5.6 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 Proxy 문제 해결

이 문서는 Amazon RDS Proxy에 대한 시험판 설명서입니다. 변경될 수 있습니다.

일반적인 문제 및 해결 방법

다음 증상 목록을 참조하여 RDS Proxy에서 발생할 수 있는 문제의 가능한 원인과 해결 방법을 확인할 수 있습니다. 공개 미리 보기 중에는 오류 처리 코드가 완료되지 않으므로 기본 서비스에서 암호 같은 오류 메시지가 나타날 수 있습니다.

새 프록시를 생성하는 동안 다음과 같은 문제가 발생할 수 있습니다.

오류 또는 증상	원인 또는 해결 방법
403: The security token included in the request is invalid	<ul style="list-style-type: none"> 새 IAM 역할을 생성하는 대신 기존 IAM 역할을 선택합니다.

프록시에 연결하는 동안 다음과 같은 문제가 발생할 수 있습니다.

오류 또는 증상	원인 또는 해결 방법
ERROR 2003 (HY000): Can't connect to MySQL server on ' proxy_endpoint ' (110)	<p>프록시 엔드포인트가 존재하지만 RDS DB 인스턴스 또는 Aurora DB 클러스터에 연결할 수 없습니다. 이 오류는 일반적으로 시간 초과 기간 후에 발생합니다. 몇 가지 가능한 이유는 다음과 같습니다.</p> <p>(110) 존재하지 않는 데이터베이스 사용자 이름을 지정했을 수 있습니다.</p> <ul style="list-style-type: none"> 잘못된 데이터베이스 사용자 암호를 입력했을 수 있습니다. 지정된 데이터베이스 사용자의 자격 증명을 포함하는 Secrets Manager 비밀이 없을 수 있습니다. Secrets Manager 비밀의 자격 증명이 해당 데이터베이스 사용자의 자격 증명과 일치하지 않을 수 있습니다. 해당 Secrets Manager 비밀을 검색할 수 있는 권한이 없을 수도 있습니다.
ERROR 2005 (HY000): Unknown MySQL server host ' proxy_endpoint ' (0)	<p>프록시 엔드포인트가 존재하지 않습니다. 이 오류는 일반적으로 즉시 발생합니다. 몇 가지 가능한 이유는 다음과 같습니다.</p> <ul style="list-style-type: none"> 프록시를 삭제했을 수 있습니다. 프록시 엔드포인트를 잘못 복사 또는 입력했을 수 있습니다. 잘못된 포트를 지정했을 수 있습니다. 현재 모든 프록시는 포트 3306에서 수신 대기합니다.
ERROR 2013 (HY000): Lost connection to MySQL server at 'reading initial communication packet', system error: 2	<p>프록시가 Secrets Manager 비밀의 자격 증명을 사용하여 데이터베이스에 로그인할 수 없습니다. 몇 가지 가능한 이유는 다음과 같습니다.</p> <ul style="list-style-type: none"> 프록시와 연결된 Secrets Manager 비밀에 연결하려는 사용자 이름이 포함된 비밀이 없습니다. 그렇다면 올바른 자격 증명을 포함하는 비밀을 프록시에 추가할 수 있습니다. 연결하려는 사용자 이름의 Secrets Manager 비밀에 잘못된 암호가 있습니다. 그렇다면 잘못된 자격 증명을 포함하는 비밀을 업데이트할 수 있습니다. 연결하려는 사용자가 아닌 다른 사용자의 Secrets Manager 비밀에 잘못된 암호가 있습니다. 그렇다면 잘못된 자격 증명을 포함하는 비밀을 제거할 수 있습니다.
ERROR 1045 (28000): Access denied for user ' DB_USER '@'%' (using password: YES)	<p>몇 가지 가능한 이유는 다음과 같습니다.</p> <ul style="list-style-type: none"> 프록시에서 사용하는 Amazon SageMaker 암호에는 교체 기간이 있습니다. RDS 프록시에 사용되는 암호에 대해 교체를 활성화하지 마십시오.

CloudWatch 로그

AWS Management 콘솔의 CloudWatch 아래에서 RDS Proxy 활동 로그를 찾을 수 있습니다. 각 프록시는 그 그룹 페이지에 항목이 있습니다.

Important

이러한 로그는 프로그래밍 방식 액세스가 아니라 사용자가 문제 해결을 위해 사용하기 위한 로그입니다. 로그의 형식과 내용이 변경될 수 있습니다(특히 미리 보기 중에).

프록시에 대한 연결 확인

다음 명령을 사용하여 연결 메커니즘의 모든 구성 요소가 다른 구성 요소와 통신할 수 있는지 확인할 수 있습니다.

프록시 자체를 검사합니다. 또한 연결된 대상 그룹과 대상 그룹에 지정된 대상을 검사합니다. 대상의 세부 정보가 프록시와 연결하려는 RDS DB 인스턴스 또는 Aurora DB 클러스터와 일치하는지 확인합니다.

```
aws rds describe-db-proxies --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-target-groups --db-proxy-name $DB_PROXY_NAME
aws rds describe-db-proxy-targets --db-proxy-name $DB_PROXY_NAME
```

다음 nc 명령이 성공을 보고하면 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 proxy_endpoint 3306
```

다음 명령을 사용하여 EC2 인스턴스에 필수 속성이 있는지 확인할 수 있습니다. 특히 EC2 인스턴스의 VPC는 프록시가 연결하는 RDS DB 인스턴스 또는 Aurora DB 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-value에 표시된 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
}
```

프록시와 연결된 RDS DB 인스턴스 또는 Aurora DB 클러스터를 검사합니다. Aurora DB 클러스터의 경우 클러스터의 각 DB 인스턴스도 검사합니다. 해당 속성이 프록시에 예상되는 것과 일치하는지 확인합니다(예: 포트가 3306).

```
aws rds describe-db-clusters --db-cluster-id $DB_CLUSTER_ID
aws rds describe-db-instances --db-instance-id $DB_INSTANCE_ID1
aws rds describe-db-instances --db-instance-id $DB_INSTANCE_ID2
```

Aurora DB 클러스터에서 데이터베이스 복제

데이터베이스 복제를 사용하면 Aurora DB 클러스터에 속한 데이터베이스를 모두 빠르고 비용 효과적으로 복제할 수 있습니다. 복제본 데이터베이스는 최초 생성 시 최소한의 추가 공간만 필요합니다.

데이터베이스 복제는 원본 데이터베이스 또는 복제본 데이터베이스에서 데이터가 변경되는 시점에 데이터가 복사되는 기록 중 복사 프로토콜을 사용합니다. 동일한 DB 클러스터에서 여러 복제본을 생성할 수 있습니다. 다른 복제본에서 추가 복제본을 생성할 수도 있습니다. Aurora 스토리지와 관련하여 기록 중 복사 프로토콜을 사용하는 방법에 대한 자세한 정보는 [데이터베이스 복제를 위한 기록 중 복사 프로토콜 \(p. 240\)](#) 단원을 참조하십시오.

다양한 사용 사례에서 특히 프로덕션 환경에 영향을 미치기를 원치 않을 경우 데이터베이스 복제를 사용할 수 있습니다. 다음은 몇 가지 예입니다.

- 스키마 변경 사항 또는 파라미터 그룹 변경 사항 등 변경 사항의 영향을 실험 및 평가하는 경우
- 데이터 내보내기 또는 분석 쿼리 실행과 같은 워크로드 집약적 작업을 수행하는 경우
- 비 프로덕션 환경에서 또는 테스트용으로 프로덕션 DB 클러스터의 복제본을 생성하는 경우

주제

- [제한 사항 \(p. 239\)](#)
- [데이터베이스 복제를 위한 기록 중 복사 프로토콜 \(p. 240\)](#)
- [원본 데이터베이스 삭제 \(p. 241\)](#)
- [AWS Management 콘솔을 통한 Aurora 클러스터 복제 \(p. 241\)](#)
- [AWS CLI를 통한 Aurora 클러스터 복제 \(p. 242\)](#)
- [계정 간 복제 \(p. 243\)](#)

제한 사항

데이터베이스 복제에는 다음에 설명하는 몇 가지 제한 사항이 있습니다.

- AWS 리전 간에 복제본 데이터베이스를 생성할 수 없습니다. 복제본 데이터베이스는 원본 데이터베이스와 동일한 리전에 생성해야 합니다.
- 현재 다른 복제본 기반의 복제본을 포함해 복사본 하나당 복제본 수는 15개로 제한됩니다. 이후에는 복사본만 생성할 수 있습니다. 그렇지만 복사본 하나가 최대 15개 복제본으로 구성될 수 있습니다.
- 현재 병렬 쿼리 기능이 없는 클러스터에서 병렬 쿼리가 활성화된 클러스터로 복제할 수 없습니다. 병렬 쿼리를 사용하는 클러스터에 데이터를 가져오려면 원본 클러스터의 스냅샷을 생성하여 병렬 쿼리 옵션이 활성화된 클러스터에 복원하십시오.
- 복제본에 다양한 Virtual Private Cloud(VPC)를 제공할 수 있습니다. 하지만 이러한 VPC의 서브넷을 동일한 가용 영역 세트에 매핑해야 합니다.

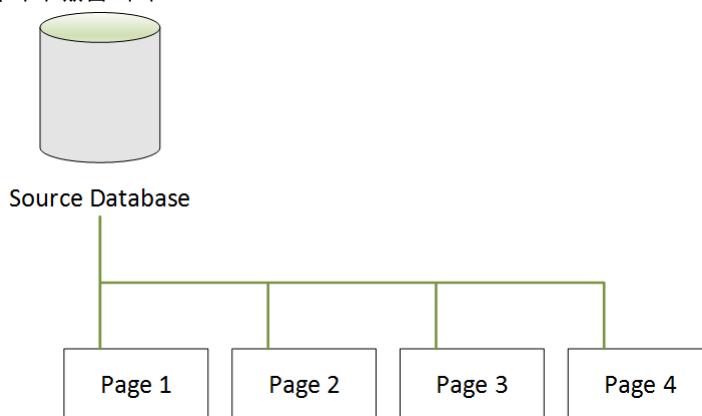
데이터베이스 복제를 위한 기록 중 복사 프로토콜

다음 시나리오는 기록 중 복사 프로토콜의 작동 방식을 보여줍니다.

- 데이터베이스 복제 전 (p. 240)
- 데이터베이스 복제 후 (p. 240)
- 원본 데이터베이스에 변경 사항이 발생하는 경우 (p. 240)
- 복제본 데이터베이스에 변경 사항이 발생하는 경우 (p. 241)

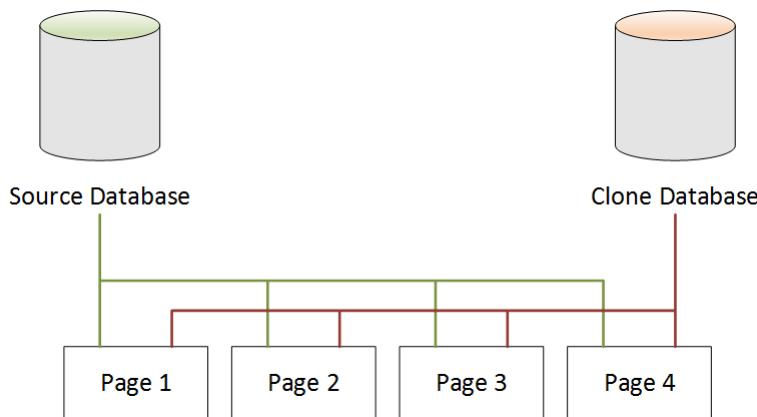
데이터베이스 복제 전

원본 데이터베이스에서 데이터가 페이지 단위로 저장됩니다. 다음 다이어그램에서 원본 데이터베이스에 4개 페이지가 있습니다.



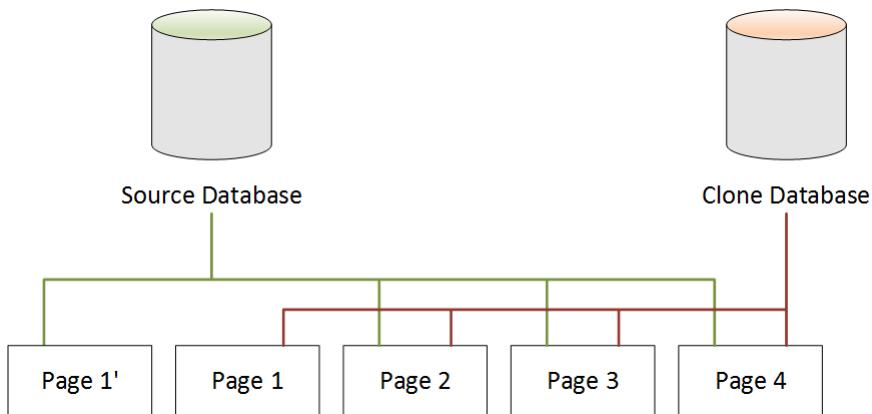
데이터베이스 복제 후

다음 다이어그램에 나오듯이 데이터베이스 복제 후 원본 데이터베이스에 변동이 없습니다. 원본 데이터베이스와 복제본 데이터베이스 모두 동일한 4개 페이지를 가리킵니다. 물리적으로 복사된 페이지가 없으며, 따라서 추가 스토리지가 필요하지 않습니다.



원본 데이터베이스에 변경 사항이 발생하는 경우

다음 예제에서 원본 데이터베이스가 Page 1에서 데이터를 변경합니다. 원래 Page 1에 기록하는 대신, 추가 스토리지를 사용하여 Page 1'이라는 새 페이지가 생성됩니다. 이제 원본 데이터베이스가 새로운 Page 1' 이외에 Page 2, Page 3, Page 4도 가리킵니다. 복제본 데이터베이스는 계속해서 Page 1~Page 4를 가리킵니다.



복제본 데이터베이스에 변경 사항이 발생하는 경우

다음 다이어그램에서 복제본 데이터베이스도 변경되었습니다. 이번에는 Page 4입니다. 원래 Page 4에 기록하는 대신, 추가 스토리지를 사용하여 Page 4'이라는 새 페이지가 생성됩니다. 원본 데이터베이스가 Page 1'과 Page 2~Page 4를 계속 가리키지만, 복제본 데이터베이스는 이제 Page 1~Page 3과 Page 4'도 가리킵니다.



두 번째 시나리오에서 예시한 대로, 데이터베이스 복제 이후 복제 생성 지점에 추가 스토리지가 필요하지 않습니다. 하지만 세 번째 및 네 번째 시나리오와 같이 원본 데이터베이스 및 복제본 데이터베이스가 변경될 경우 변경된 페이지만 생성됩니다. 시간이 경과하여 원본 데이터베이스와 복제본 데이터베이스 모두가 변경될 경우 변경 사항을 캡처하고 저장하기 위해 점점 더 많은 스토리지가 필요합니다.

원본 데이터베이스 삭제

하나 이상의 복제본 데이터베이스가 연결되어 있는 원본 데이터베이스를 삭제할 경우 복제본 데이터베이스는 영향을 받지 않습니다. 복제본 데이터베이스는 이전에 원본 데이터베이스가 소유하던 페이지를 계속 가리킵니다.

AWS Management 콘솔을 통한 Aurora 클러스터 복제

다음 프로시저에서는 AWS Management 콘솔을 사용하여 Aurora DB 클러스터를 복제하는 방법에 대해 설명합니다.

이번 지침은 복제본을 생성하는 것과 동일한 AWS 계정에서 소유하고 있는 DB 클러스터에 적용됩니다. DB 클러스터를 다른 AWS 계정에서 소유하고 있다면 [계정 간 복제](#) (p. 243) 단원을 참조하십시오.

AWS Management 콘솔을 사용해 AWS 계정에서 소유하는 DB 클러스터 복제본을 생성하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다. 복제본을 생성할 DB 클러스터를 선택합니다.
3. 작업에서 복제본 생성을 선택합니다.
4. 복제본 생성 페이지에서 복제본 DB 클러스터의 기본 인스턴스 이름을 DB 인스턴스 식별자로 입력합니다.

필요에 따라 복제본 DB 클러스터에 다른 설정을 구성합니다. 다양한 DB 클러스터 설정에 대한 자세한 정보는 [새운 콘솔 \(p. 96\)](#) 단원을 참조하십시오.

5. 복제본 생성을 선택해 복제본 DB 클러스터를 시작합니다.

AWS CLI를 통한 Aurora 클러스터 복제

다음 프로시저에서는 AWS CLI를 사용하여 Aurora DB 클러스터를 복제하는 방법에 대해 설명합니다.

AWS CLI를 사용하여 DB 클러스터 복제를 생성하려면

- `restore-db-cluster-to-point-in-time` AWS CLI 명령을 호출하고 다음 값을 입력합니다.
 - `--source-db-cluster-identifier` – 복제를 생성할 원본 DB 클러스터의 이름입니다.
 - `--db-cluster-identifier` – 복제 DB 클러스터의 이름입니다.
 - `--restore-type copy-on-write` – 복제 DB 클러스터를 생성하기 위해 나타내는 값입니다.
 - `--use-latest-restorable-time` – 최신 복원 가능한 백업 시간을 사용하도록 지정합니다.

다음 예제에서는 `sample-source-cluster`라는 DB 클러스터의 복제본을 생성합니다. 복제본 DB 클러스터의 이름은 `sample-cluster-clone`입니다.

Linux, OS X, Unix의 경우:

```
aws rds restore-db-cluster-to-point-in-time \
--source-db-cluster-identifier sample-source-cluster \
--db-cluster-identifier sample-cluster-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 sample-source-cluster ^
--db-cluster-identifier sample-cluster-clone ^
--restore-type copy-on-write ^
--use-latest-restorable-time
```

Note

`restore-db-cluster-to-point-in-time` AWS CLI 명령은 해당 DB 클러스터의 DB 인스턴스가 아닌 DB 클러스터만 복원합니다. `--db-cluster-identifier`에 복원된 DB 클러스터의 식별자를 지정하여 복원된 DB 클러스터의 DB 인스턴스를 생성하려면 `create-db-instance` 명령을 호출해야 합니다. `restore-db-cluster-to-point-in-time` 명령이 완료되고 DB 클러스터를 사용 가능할 때만 DB 인스턴스를 생성할 수 있습니다.

계정 간 복제

Amazon Aurora에서는 Aurora DB 클러스터를 다른 AWS 계정 또는 AWS 조직과 공유할 수 있습니다. 이러한 공유를 통해 DB 클러스터를 복제한 후 다른 계정 또는 조직에서 복제본에 액세스하는 것도 가능합니다.

예를 들어 프로덕션 환경과 테스트 환경에서 계정을 따로 사용하고 있다면 프로덕션 데이터의 복제본을 테스트 계정에서 생성할 수 있습니다. 그런 다음 복제본에서 다양한 파라미터를 사용해 실험하거나, 높은 비용의 온라인 분석 처리(OLAP) 쿼리를 실행하는 등 프로덕션 환경에 미치는 영향 없이 모든 작업을 처리할 수 있습니다. 그 밖에 기계 학습 모델을 훈련할 목적으로 외부 공급업체 같은 외부 업체에게 데이터베이스에 대한 액세스 권한을 부여할 수도 있습니다. 계정 간 복제는 데이터베이스 스냅샷을 생성하여 복원하는 것보다 훨씬 빠릅니다.

공유 권한을 승인할 때는 AWS 리소스 액세스 관리자(AWS RAM)를 사용할 수 있습니다.. AWS RAM을 통해 액세스를 제어하는 방법에 대한 자세한 내용은 [AWS RAM 사용 설명서](#) 단원을 참조하십시오.

계정 간 복제본을 만들려면 원본 클러스터를 소유하고 있는 AWS 계정과 복제본을 생성하는 계정에서 몇 가지 작업이 필요합니다. 먼저 소유 계정에서 다른 계정 1개 이상이 복제할 수 있도록 클러스터를 수정합니다. 일부 계정이 다른 AWS 조직에 속하는 경우에는 AWS가 공유 초대장을 생성하고 다른 계정이 사전에 초대장을 승인해야 합니다. 그러면 승인된 계정에서 클러스터를 복제할 수 있습니다. 이러한 프로세스 전체에서 클러스터는 고유한 Amazon 리소스 이름(ARN)으로 식별합니다.

데이터를 변경할 경우 추가되는 스토리지 공간에 대한 비용이 발생합니다. 원본 클러스터가 삭제되면 스토리지 비용이 나머지 복제된 클러스터로 동일하게 배분됩니다.

주제

- [계정 간 복제 제한 사항 \(p. 243\)](#)
- [다른 AWS 계정에서 클러스터를 복제하도록 허용 \(p. 243\)](#)
- [다른 AWS 계정에서 소유하고 있는 클러스터 복제 \(p. 246\)](#)

계정 간 복제 제한 사항

Aurora 계정 간 복제는 다음과 같은 제한 사항이 있습니다.

- Aurora Serverless 클러스터는 AWS 계정 간 복제가 불가능합니다.
- Aurora 글로벌 데이터베이스 클러스터는 AWS 계정 사이에서 복제할 수 없습니다.
- 공유 초대장을 보고 승인하려면 AWS CLI, Amazon RDS API 또는 AWS RAM 콘솔을 사용해야 합니다. 현재 Amazon RDS 콘솔에서는 이러한 절차를 수행할 수 없습니다.
- 계정 간 클러스터를 생성할 경우 새롭게 생성된 클러스터의 복제본을 추가로 생성하거나 복제된 클러스터를 다른 AWS 계정과 공유할 수 없습니다.
- 어떤 Aurora 클러스터든 계정 간 최대 복제본 수는 15개로 제한됩니다.
- 다른 AWS 계정과 공유할 때는 클러스터가 ACTIVE 상태이어야 합니다.
- Aurora 클러스터를 다른 AWS 계정과 공유하는 동안 클러스터 이름을 변경할 수 없습니다.
- 클러스터가 기본 RDS 키로 암호화되어 있으면 계정 간 복제본을 생성할 수 없습니다.
- 암호화된 클러스터를 공유할 때는 복제된 클러스터도 암호화해야 합니다. 이때 사용하는 암호화 키는 원본 클러스터에서 사용하는 암호화 키와 다를 수 있습니다. 또한 클러스터 소유 계정이 원본 클러스터에서 사용하는 AWS Key Management Service(AWS KMS) 키에 대한 액세스 권한을 부여해야 합니다.

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](#) 단원을 참조하십시오.

주제

- [다른 AWS 계정에게 클러스터를 복제할 수 있는 권한 부여](#) (p. 244)
- [소유하고 있는 클러스터에 대한 다른 AWS 계정의 공유 여부 확인](#) (p. 245)

다른 AWS 계정에게 클러스터를 복제할 수 있는 권한 부여

공유하고 있는 클러스터가 암호화되어 있으면 해당 클러스터의 고객 마스터 키(CMK)도 공유합니다. 임의 AWS 계정의 AWS Identity and Access Management(IAM) 사용자 또는 역할이 다른 계정의 CMK를 사용하도록 허용할 수 있습니다. 이를 위해서는 먼저 외부 계정(루트 사용자)을 AWS KMS를 통해 CMK의 키 정책에 추가합니다. 개별 IAM 사용자 또는 역할이 아니고 사용자 또는 역할을 소유한 외부 계정을 추가하는 것입니다. 또한 사용자가 생성하는 CMK만 공유할 수 있으며, 기본 RDS 서비스 키는 공유하지 못합니다. CMK 액세스 제어에 대한 자세한 내용은 [AWS KMS에 대한 인증 및 액세스 제어](#) 단원을 참조하십시오..

콘솔

AWS Management 콘솔을 사용해 클러스터 복제 권한을 부여하려면

1. AWS Management 콘솔에 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. Share DB cluster with other AWS accounts(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 AWS Management 콘솔이 해당 계정 ID의 소유자 또는 계정 존재 여부를 보고하지 않습니다.

자신의 AWS 계정과 다른 AWS 조직에 속하는 계정 숫자를 입력할 때는 주의하십시오. 원하는 계정과의 공유 여부를 바로 확인하십시오.

5. 확인 페이지에서 지정한 계정 ID가 정확한지 확인합니다. 확인 입력란에 share를 입력하여 확인합니다. 세부 정보 페이지에서 Accounts that this DB cluster is shared with(이 DB 클러스터를 공유하는 계정) 아래 지정된 AWS 계정 ID를 표시한 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

AWS CLI를 사용해 클러스터 복제 권한을 부여하려면

1. 필요한 파라미터 정보를 수집합니다. 클러스터 ARN과 다른 AWS 계정의 숫자 ID가 필요합니다.
2. RAM CLI 명령인 `create-resource-share`를 실행합니다.

Linux, OS X, Unix의 경우:

```
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 `create-resource-share`에서 `--allow-external-principals` 또는 `--no-allow-external-principals` 파라미터를 추가하십시오.

RAM API

RAM API를 사용해 클러스터 복제 권한을 부여하려면

- 필요한 파라미터 정보를 수집합니다. 클러스터 ARN과 다른 AWS 계정의 숫자 ID가 필요합니다.
- RAM API 작업인 [CreateResourceShare](#)를 호출한 후 다음 값을 지정합니다.
 - AWS 계정(들)의 계정 ID를 `principals` 파라미터로 지정합니다.
 - Aurora DB 클러스터(들)의 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

기본 RDS 키를 사용해 암호화된 클러스터를 재생성하려면

공유할 암호화 클러스터가 기본 RDS 키를 사용할 경우에는 고객 마스터 키(CMK)를 사용해 클러스터를 다시 생성해서 공유해야 합니다. DB 클러스터의 수동 스냅샷을 생성하여 새로운 클러스터로 복원하면 가능합니다. 다음 단계를 사용합니다.

- AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
- 탐색 창에서 스냅샷을 선택합니다.
- 자신의 스냅샷을 선택합니다.
- 작업에서 스냅샷 복사와 암호화 활성화를 차례대로 선택합니다.
- 마스터 키에서 새롭게 사용할 암호화 키를 선택합니다.
- 복사된 스냅샷을 복원합니다. 복원 방법으로는 [DB 클러스터 스냅샷에서 복원 \(p. 273\)](#)의 절차를 따릅시오. 새로운 DB 인스턴스는 새로운 암호화 키를 사용합니다.
- (선택 사항) 더 이상 필요 없다면 이전 DB 클러스터를 삭제합니다. 삭제 방법으로는 [DB 클러스터 스냅샷 삭제 \(p. 305\)](#)의 절차를 따릅시오. 단, 삭제 전에 새로운 클러스터에 필요한 데이터가 모두 있는지, 그리고 애플리케이션이 새로운 클러스터에 성공적으로 액세스하는지 확인하십시오.

소유하고 있는 클러스터에 대한 다른 AWS 계정의 공유 여부 확인

다른 사용자에게 클러스터 공유 권한이 있는지 확인할 수 있습니다. 따라서 클러스터가 계정 간 최대 복제 수 제한에 접근하는지 파악하는 데 효과적입니다.

AWS RAM 콘솔에서 리소스를 공유하는 방법에 대한 자세한 내용은 AWS RAM 사용 설명서에서 [자신이 소유한 리소스 공유](#) 단원을 참조하십시오.

AWS CLI

소유하고 있는 클러스터에 대한 다른 AWS 계정의 공유 여부를 AWS CLI를 사용해 확인하려면

- 자신의 계정 ID를 리소스 소유자로, 그리고 클러스터 ARN을 리소스 ARN으로 사용하여 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
```

RAM API

소유하고 있는 클러스터에 대한 다른 AWS 계정의 공유 여부를 RAM API를 사용해 확인하려면

- RAM API 작업인 [ListPrincipals](#)를 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 가져와야 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 액세스 단원](#)을 참조하십시오.

주제

- [다른 AWS 계정에서 소유하고 있는 클러스터에 대한 복제 초대장 보기 \(p. 246\)](#)
- [다른 AWS 계정에서 소유하고 있는 클러스터에 대한 공유 초대장 승인 \(p. 247\)](#)
- [다른 AWS 계정에서 소유하고 있는 Aurora 클러스터 복제 \(p. 248\)](#)
- [DB 클러스터의 계정 간 복제본 여부 확인 \(p. 251\)](#)

다른 AWS 계정에서 소유하고 있는 클러스터에 대한 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 작업을 하는 방법은 AWS RAM 사용 설명서에서 [공유 리소스에 대한 액세스 단원](#)을 참조하십시오.

AWS CLI

다른 AWS 계정에서 소유하고 있는 클러스터에 대한 복제 초대장을 AWS CLI를 사용해 보려면

- 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.

RAM API

다른 AWS 계정에서 소유하고 있는 클러스터에 대한 복제 초대장을 RAM API를 사용해 보려면

1. RAM API 작업인 `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. 초대장 승인 작업은 AWS CLI, RAM 및 RDS API 또는 RAM 콘솔을 사용합니다. 현재 RDS 콘솔에서는 이러한 절차를 수행할 수 없습니다.

AWS RAM 콘솔에서 초대장 작업을 하는 방법은 AWS RAM 사용 설명서에서 [공유 리소스에 대한 액세스 단원](#)을 참조하십시오.

콘솔

다른 AWS 계정에서 AWS CLI를 사용해 클러스터 공유 초대장을 승인하려면

1. 앞에서 한 것처럼 RAM CLI 명령인 `get-resource-share-invitations`를 실행하여 초대장 ARN을 찾습니다.
2. 다음과 같이 RAM CLI 명령인 `accept-resource-share-invitation`을 호출하여 초대장을 승인합니다.

Linux, OS X, Unix의 경우:

```
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
```

RAM 및 RDS API

RAM 및 RDS API를 사용해 다른 사용자의 클러스터 공유 초대장을 승인하려면

1. 앞에서 한 것처럼 RAM API 작업인 `GetResourceShareInvitations`를 호출하여 초대장 ARN을 찾습니다.
2. 해당 ARN을 `resourceShareInvitationArn` 파라미터로 RDS API 작업인 `AcceptResourceShareInvitation`에게 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 AWS Management 콘솔을 사용해 복제하려면

1. AWS Management 콘솔에 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. 작업 메뉴에서 복제본 생성을 선택합니다.
5. [AWS Management 콘솔을 통한 Aurora 클러스터 복제 \(p. 241\)](#) 단원의 절차에 따라 복제 클러스터의 설정을 마칩니다.
6. 필요하다면 복제 클러스터의 암호화를 활성화합니다. 복제할 클러스터가 암호화되어 있다면 복제 클러스터에서도 암호화를 활성화해야 합니다. 자신과 클러스터를 공유한 AWS 계정 역시 클러스터를 암호화하는 데 사용된 KMS 키를 공유해야 합니다. 복제본을 암호화할 때는 동일한 키를 사용하거나, 자신만의 고객 마스터 키(CMK)를 사용해도 좋습니다. 클러스터가 기본 RDS 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 AWS CLI를 사용해 복제하려면

1. 위와 같이 DB 클러스터를 소유하고 있는 AWS 계정의 초대장을 승인합니다.
2. 다음과 같이 RDS CLI 명령인 `restore-db-cluster-to-point-in-time`의 `source-db-cluster-identifier` 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.

Linux, OS X, Unix의 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
```

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 파라미터 값은 원본 DB 클러스터를 암호화할 때 사용한 것과 동일하거나, 혹은 사용자 고유의 고객 마스터 키(CMK)가 될 수도 있습니다. 단, 암호화 키를 사용할 수 있는 권한이 계정에게 있어야 합니다.

Linux, OS X, Unix의 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
```

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}}
    }
}
```

RDS API

다른 AWS 계정에서 소유하고 있는 Aurora 클러스터를 RDS API를 사용해 복제하려면

1. 위와 같이 DB 클러스터를 소유하고 있는 AWS 계정의 초대장을 승인합니다.
2. RDS API 작업인 [RestoreDBClusterToPointInTime](#)의 `SourceDBClusterIdentifier` 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` 파라미터 값은 원본 DB 클러스터를 암호화할 때 사용한 것과 동일하거나, 혹은 사용자 고유의 고객 마스터 키(CMK)가 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`에서 지정한 암호화 키를 사용해 새롭게 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:lambda:123456789012:alias/MyFunction"
            ]},
            "Action": [
                "kms>Encrypt",
                "kms>Decrypt"
            ],
            "Resource": "*"
        }
    ]
}
```

```
"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}}
}
]
```

DB 클러스터의 계정 간 복제본 여부 확인

DBClusters 객체는 각 클러스터의 계정 간 복제본 여부를 식별합니다. RDS CLI 명령인 [describe-db-clusters](#)를 실행할 때 `--include-shared` 옵션을 사용해 자신에게 복제 권한이 있는 클러스터를 알아볼 수 있습니다. 하지만 해당 클러스터의 구성 세부 정보는 대부분 볼 수 없습니다.

AWS CLI

AWS CLI를 사용해 DB 클러스터의 계정 간 복제본 여부를 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 `DBClusterArn` 필드에 자신과 다른 AWS 계정 번호의 항목이 존재할 수 있습니다. 이러한 경우 해당 항목은 다른 AWS 계정에서 소유하고 있지만 자신이 복제할 수 있는 클러스터라는 것을 의미 합니다. 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": "2019-05-01T21:17:54.106Z",
            "Engine": "aurora",
            "EngineVersion": "5.6.10a",
            "CrossAccountClone": false,
            ...
        },
        {
            "EarliestRestorableTime": "2019-04-09T16:01:07.398Z",
            "Engine": "aurora",
            "EngineVersion": "5.6.10a",
            "CrossAccountClone": true,
            ...
        },
        {
            "StorageEncrypted": false,
            "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-abcdefgh",
            "Engine": "aurora",
            "EngineVersion": "5.6.10a",
        }
    ]
}
```

}

RDS API

RDS API를 사용해 DB 클러스터의 계정 간 복제본 여부를 확인하려면

- RDS API 작업인 [DescribeDBClusters](#)를 호출합니다.

AWS 계정에서 소유하고 있는 기존 계정이라면 `CrossAccountClone` 필드에서 클러스터가 다른 AWS 계정에서 소유하고 있는 DB 클러스터의 복제본인지 알 수 있습니다. `DBClusterArn` 필드에서 AWS 계정 번호가 다른 항목은 복제 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": "2019-05-01T21:17:54.106Z",  
            "Engine": "aurora",  
            "EngineVersion": "5.6.10a",  
            "CrossAccountClone": false,  
            ...  
        },  
        {  
            "EarliestRestorableTime": "2019-04-09T16:01:07.398Z",  
            "Engine": "aurora",  
            "EngineVersion": "5.6.10a",  
            "CrossAccountClone": true,  
            ...  
        },  
        {  
            "StorageEncrypted": false,  
            "DBClusterArn": "arn:aws:rds:us-east-1:12345678:cluster:cluster-abcdefg",  
            "Engine": "aurora",  
            "EngineVersion": "5.6.10a"  
        }  
    ]  
}
```

Aurora를 다른 AWS 서비스와 통합

Aurora DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora를 다른 AWS 서비스와 통합하십시오.

주제

- [AWS 서비스를 Amazon Aurora MySQL과 통합 \(p. 253\)](#)
- [AWS 서비스를 Amazon Aurora PostgreSQL과 통합 \(p. 253\)](#)
- [Aurora 복제본에 Amazon Aurora Auto Scaling 사용 \(p. 253\)](#)
- [Amazon Aurora에서 기계 학습\(ML\) 기능 사용 \(p. 267\)](#)

AWS 서비스를 Amazon Aurora MySQL과 통합

Aurora MySQL DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora MySQL가 다른 AWS 서비스와 통합되었습니다. Aurora MySQL DB 클러스터는 AWS 서비스를 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- `lambda_sync` 또는 `lambda_async` 네이티브 함수를 사용하여 동기적 또는 비동기적으로 AWS Lambda 함수를 호출하십시오. 또는 `mysql.lambda_async` 프로시저를 사용하여 비동기적으로 AWS Lambda 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 복제본에 Amazon Aurora Auto Scaling 사용 \(p. 253\)](#)을 참조하십시오.

Aurora MySQL과 다른 AWS 서비스의 통합에 대한 자세한 내용은 [Amazon Aurora MySQL를 다른 AWS 서비스와 통합 \(p. 609\)](#) 단원을 참조하십시오.

AWS 서비스를 Amazon Aurora PostgreSQL과 통합

Aurora PostgreSQL DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora PostgreSQL가 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 복제본에 Amazon Aurora Auto Scaling 사용 \(p. 253\)](#) 단원을 참조하십시오.

Aurora PostgreSQL과 다른 AWS 서비스의 통합에 대한 자세한 내용은 [Amazon Aurora PostgreSQL를 다른 AWS 서비스와 통합 \(p. 812\)](#) 단원을 참조하십시오.

Aurora 복제본에 Amazon Aurora Auto Scaling 사용

연결 및 워크로드 요구 사항을 충족하기 위해 Aurora Auto Scaling은 단일 마스터 복제를 사용해 Aurora DB 클러스터에 대해 프로비저닝된 Aurora 복제본 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.

AWS Management 콘솔을 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 조정 정책을 적용할 수도 있습니다.

주제

- [시작하기 전 \(p. 254\)](#)

- Aurora Auto Scaling 정책 (p. 254)
- 조정 정책 추가 (p. 255)
- 조정 정책 편집 (p. 264)
- 조정 정책 삭제 (p. 266)
- 관련 주제 (p. 267)

시작하기 전

Aurora Auto Scaling을 Aurora DB 클러스터에 사용하려면 먼저 기본 인스턴스와 적어도 하나의 Aurora 복제본이 있는 Aurora DB 클러스터를 생성해야 합니다. Aurora Auto Scaling이 Aurora 복제본을 관리하더라도 처음에 적어도 하나의 Aurora 복제본이 Aurora DB 클러스터에 있어야 합니다. Aurora DB 클러스터 생성에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

DB 클러스터에 있는 모든 Aurora 복제본이 사용 가능한 상태일 경우에만 Aurora Auto Scaling이 DB 클러스터의 크기를 조정합니다. Aurora 복제본이 사용 가능 이외의 상태일 경우에는 Aurora Auto Scaling이 전체 DB 클러스터가 조정할 수 있는 상태가 될 때까지 기다립니다.

Aurora Auto Scaling이 새로운 Aurora 복제본을 추가할 때 새로운 Aurora 복제본은 기본 인스턴스에 사용되는 것과 동일한 DB 인스턴스 클래스입니다. DB 인스턴스 클래스에 대한 자세한 정보는 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 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 \(p. 268\)](#) 단원을 참조하십시오.

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. Aurora MySQL의 경우 MariaDB Connector/J 유ти리티와 같은 드라이버를 사용할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#) 단원을 참조하십시오.

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

서비스 연결 역할

Aurora Auto Scaling은 `AWSServiceRoleForApplicationAutoScaling_RDSCluster` 서비스 연결 역할을 사용합니다. 자세한 정보는 Application Auto Scaling 사용 설명서의 [Application Auto Scaling 서비스 연결 역할](#)을 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 값

에 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.

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.

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.

스케일 인 또는 스케일 아웃 휴지 기간을 지정하지 않으면 각 기본값은 300초가 됩니다.

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.

조정 정책 추가

AWS Management 콘솔, AWS CLI 또는 Application Auto Scaling API를 사용하여 조정 정책을 추가할 수 있습니다.

주제

- AWS Management 콘솔을 사용하여 조정 정책 추가 (p. 256)
- AWS CLI 또는 Application Auto Scaling API를 사용하여 조정 정책 추가 (p. 258)

AWS Management 콘솔을 사용하여 조정 정책 추가

AWS Management 콘솔을 사용하여 조정 정책을 Aurora DB 클러스터에 추가할 수 있습니다.

Aurora DB 클러스터에 Auto Scaling 정책을 추가하려면

1. AWS Management 콘솔에 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) [Additional Configuration]을 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.

Add Auto Scaling policy

Define an Auto Scaling policy to automatically add or remove [Aurora Replicas](#). We recommend using the Aurora reader endpoint or the MariaDB Connector to establish connections with new Aurora Replicas. [Learn more](#).

Policy details

Policy name
A name for the policy used to identify it in the console, CLI, API, notifications, and events.
 CPUScalingPolicy

Policy name must be 1 to 256 characters.

IAM role
The following service-linked role is used by Aurora Auto Scaling.
 AWSServiceRoleForApplicationAutoScaling_RDSCluster

Target metric
Only one Aurora Auto Scaling policy is allowed for one metric.

Average CPU utilization of Aurora Replicas [View metric](#)

Average connections of Aurora Replicas [View metric](#)

Target value
Specify the desired value for the selected metric. Aurora Replicas will be added or removed to keep the metric close to the specified value.
 40 %

[Additional configuration](#)

Cluster capacity details

Configure the minimum and maximum number of Aurora Replicas you want Aurora Auto Scaling to maintain.

Minimum capacity
Specify the minimum number of Aurora Replicas to maintain.
 5 Aurora Replicas

Maximum capacity
Specify the maximum number of Aurora Replicas to maintain. Up to 15 Aurora Replicas are supported.
 15 Aurora Replicas

[Cancel](#) [Add policy](#)

다음 대화 상자는 평균 연결 수 100을 기반으로 Auto Scaling 정책을 생성합니다. 정책은 최소 2개의 Aurora 복제본과 최대 8개의 Aurora 복제본을 지정합니다.

Add Auto Scaling policy

Define an Auto Scaling policy to automatically add or remove [Aurora Replicas](#). We recommend using the Aurora reader endpoint or the MariaDB Connector to establish connections with new Aurora Replicas. [Learn more](#).

Policy details

Policy name
A name for the policy used to identify it in the console, CLI, API, notifications, and events.

Policy name must be 1 to 256 characters.

IAM role
The following service-linked role is used by Aurora Auto Scaling.

Target metric
Only one Aurora Auto Scaling policy is allowed for one metric.
 Average CPU utilization of Aurora Replicas [View metric](#) 
 Average connections of Aurora Replicas [View metric](#) 

Target value
Specify the desired value for the selected metric. Aurora Replicas will be added or removed to keep the metric close to the specified value.
 connections

▶ Additional configuration

Cluster capacity details

Configure the minimum and maximum number of Aurora Replicas you want Aurora Auto Scaling to maintain.

Minimum capacity
Specify the minimum number of Aurora Replicas to maintain.
 Aurora Replicas

Maximum capacity
Specify the maximum number of Aurora Replicas to maintain. Up to 15 Aurora Replicas are supported.
 Aurora Replicas

[Cancel](#) [Add policy](#)

AWS CLI 또는 Application Auto Scaling API를 사용하여 조정 정책 추가

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 조정 차원 및 한계를 정의합니다. Application Auto Scaling은 Aurora 복제본의 수를 나타내는 `rds:cluster:ReadReplicaCount` 확장 가능 치수를 따라 Aurora DB 클러스터를 동적으로 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 사용하십시오.

- **--service-namespace** – 이 값을 rds로 설정하십시오.
- **--resource-id** – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 cluster이고 고유 식별자는 cluster:myscalablecluster와 같은 Aurora DB 클러스터의 이름입니다.
- **--scalable-dimension** – 이 값을 rds:cluster:ReadReplicaCount로 설정하십시오.
- **--min-capacity** – Application Auto Scaling에서 관리할 Aurora 복제본의 최소 수. 이 값은 0-15로 설정되어야 하며 max-capacity에 지정된 값과 같거나 작아야 합니다.
- **--max-capacity** – Application Auto Scaling에서 관리할 Aurora 복제본의 최대 수. 이 값은 0-15로 설정되어야 하며 min-capacity에 지정된 값과 같거나 커야 합니다.

Example

다음 예제에서는 이름이 myscaleablecluster인 Aurora DB 클러스터를 등록합니다. 등록은 1개에서 8개 까지 Aurora 복제본을 포함하도록 DB 클러스터 크기를 동적으로 조정해야 함을 나타냅니다.

Linux, OS X, Unix의 경우:

```
aws application-autoscaling register-scalable-target \
--service-namespace rds \
--resource-id cluster:myscaleablecluster \
--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:myscaleablecluster ^
--scalable-dimension rds:cluster:ReadReplicaCount ^
--min-capacity 1 ^
--max-capacity 8
```

RDS API

Application Auto Scaling로 Aurora DB 클러스터를 등록하려면 다음 파라미터와 함께 [RegisterScalableTarget](#) Application Auto Scaling API 작업을 사용하십시오.

- **ServiceNamespace** – 이 값을 rds로 설정하십시오.
- **ResourceID** – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 cluster이고 고유 식별자는 cluster:myscaleablecluster와 같은 Aurora DB 클러스터의 이름입니다.
- **ScalableDimension** – 이 값을 rds:cluster:ReadReplicaCount로 설정하십시오.
- **MinCapacity** – Application Auto Scaling에서 관리할 Aurora 복제본의 최소 수. 이 값은 0-15로 설정되어야 하며 MaxCapacity에 지정된 값과 같거나 작아야 합니다.
- **MaxCapacity** – Application Auto Scaling에서 관리할 Aurora 복제본의 최대 수. 이 값은 0-15로 설정되어야 하며 MinCapacity에 지정된 값과 같거나 커야 합니다.

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 참조하십시오.

대상 추적 조정 정책 구성을 정의하기 위해 다음과 같은 옵션을 사용할 수 있습니다.

주제

- [미리 정의된 지표 사용](#) (p. 260)
- [사용자 지정 지표 사용](#) (p. 261)
- [휴지 기간 사용](#) (p. 261)
- [스케일 인 활동 비활성화](#) (p. 262)

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 Aurora 지표](#) (p. 346) 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 TargetValue를 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-cluster라는 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

ScaleOutCooldown에 초 단위로 값을 지정하여 Aurora DB 클러스터를 확장하기 위한 휴지 기간을 추가할 수 있습니다. 마찬가지로 ScaleInCooldown에 초 단위로 값을 추가하여 Aurora DB 클러스터를 축소하기 위한 휴지 기간을 추가할 수 있습니다. ScaleInCooldown 및 ScaleOutCooldown에 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 스케일인 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 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. 조정 정책을 Aurora DB 클러스터에 등록하려면 AWS CLI 또는 Application Auto Scaling API를 사용할 수 있습니다.

AWS CLI

Aurora DB 클러스터에 조정 정책을 적용하려면 다음 파라미터와 함께 [put-scaling-policy](#) AWS CLI 명령을 사용하십시오.

- `--policy-name` – 조정 정책의 이름입니다.
- `--policy-type` – 이 값을 `TargetTrackingScaling`로 설정하십시오.
- `--resource-id` – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 `cluster`이고 고유 식별자는 `cluster:myscaleablecluster`와 같은 Aurora DB 클러스터의 이름입니다.
- `--service-namespace` – 이 값을 `rds`로 설정하십시오.
- `--scalable-dimension` – 이 값을 `rds:cluster:ReadReplicaCount`로 설정하십시오.
- `--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.

Linux, OS X, Unix의 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
```

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
```

RDS 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 `rds`로 설정하십시오.
- `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 `rds:cluster:ReadReplicaCount`로 설정하십시오.
- `PolicyType` – 이 값을 `TargetTrackingScaling`로 설정하십시오.
- `TargetTrackingScalingPolicyConfiguration` – Aurora DB 클러스터에 사용할 대상 추적 조정 정책 구성을 합니다.

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"
        }
    }
}
```

조정 정책 편집

AWS Management 콘솔, AWS CLI 또는 Application Auto Scaling API를 사용하여 조정 정책을 편집할 수 있습니다.

AWS Management 콘솔을 사용하여 조정 정책 편집

AWS Management 콘솔을 사용하여 조정 정책을 편집할 수 있습니다.

Aurora DB 클러스터의 Auto Scaling 정책을 편집하려면

1. AWS Management 콘솔에 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. Save를 선택합니다.

[Edit Auto Scaling policy] 대화 상자 샘플은 다음과 같습니다.

Edit Auto Scaling policy

Define an Auto Scaling policy to automatically add or remove [Aurora Replicas](#). We recommend using the Aurora reader endpoint or the MariaDB Connector to establish connections with new Aurora Replicas. [Learn more](#).

Policy details

Policy name

A name for the policy used to identify it in the console, CLI, API, notifications, and events.

CPUScalingPolicy

Policy name must be 1 to 256 characters.

IAM role

The following service-linked role is used by Aurora Auto Scaling.

AWSServiceRoleForApplicationAutoScaling_RDSCluster

Target metric

Only one Aurora Auto Scaling policy is allowed for one metric.

Average CPU utilization of Aurora Replicas [View metric](#)

Average connections of Aurora Replicas [View metric](#)

Target value

Specify the desired value for the selected metric. Aurora Replicas will be added or removed to keep the metric close to the specified value.

50

%

► Additional configuration

Cluster capacity details

Capacity values specified below apply to all the Aurora Auto Scaling policies for the DB cluster.

Minimum capacity

Specify the minimum number of Aurora Replicas to maintain.

1

Aurora Replicas

Maximum capacity

Specify the maximum number of Aurora Replicas to maintain. Up to 15 Aurora Replicas are supported.

6

Aurora Replicas

⚠ Changes to the capacity values will be applied to all the Auto Scaling policies for this DB cluster.

Cancel

Save

AWS CLI 또는 Application Auto Scaling API를 사용하여 조정 정책 편집

조정 정책을 적용하는 것과 같은 방식으로 AWS CLI 또는 Application Auto Scaling API를 사용하여 조정 정책을 편집할 수 있습니다.

- AWS CLI를 사용할 때는 `--policy-name` 파라미터에서 편집할 정책의 이름을 지정하십시오. 변경할 파라미터의 새로운 값을 지정합니다.
- Application Auto Scaling API를 사용할 때는 `PolicyName` 파라미터에서 편집하려는 정책의 이름을 지정하십시오. 변경할 파라미터의 새로운 값을 지정합니다.

자세한 정보는 [Aurora DB 클러스터에 조정 정책 적용 \(p. 262\)](#) 단원을 참조하십시오.

조정 정책 삭제

AWS Management 콘솔, AWS CLI 또는 Application Auto Scaling API를 사용하여 조정 정책을 삭제할 수 있습니다.

AWS Management 콘솔을 사용하여 조정 정책 삭제

AWS Management 콘솔을 사용하여 조정 정책을 삭제할 수 있습니다.

Aurora DB 클러스터의 Auto Scaling 정책을 삭제하려면

1. AWS Management 콘솔에 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 또는 Application Auto Scaling API를 사용하여 조정 정책 삭제

AWS CLI 또는 Application Auto Scaling API를 사용하여 Aurora DB 클러스터에서 조정 정책을 삭제할 수 있습니다.

AWS CLI

Aurora DB 클러스터에서 조정 정책을 삭제하려면 다음 파라미터와 함께 `delete-scaling-policy` AWS CLI 명령을 사용하십시오.

- `--policy-name` – 조정 정책의 이름입니다.
- `--resource-id` – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 `cluster`이고 고유 식별자는 `cluster:myscalablecluster`과 같은 Aurora DB 클러스터의 이름입니다.
- `--service-namespace` – 이 값을 `rds`로 설정하십시오.
- `--scalable-dimension` – 이 값을 `rds:cluster:ReadReplicaCount`로 설정하십시오.

Example

다음 예제에서는 `myscalablepolicy`라는 대상 추적 조정 정책을 `myscalablecluster`라는 Aurora DB 클러스터에서 삭제합니다.

Linux, OS X, Unix의 경우:

```
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 ^
```

RDS API

Aurora DB 클러스터에서 조정 정책을 삭제하려면 다음 파라미터와 함께 [DeleteScalingPolicy](#) Application Auto Scaling API 작업을 사용하십시오.

- **PolicyName** – 조정 정책의 이름입니다.
- **ServiceNamespace** – 이 값을 rds로 설정하십시오.
- **ResourceId** – Aurora DB 클러스터의 리소스 식별자. 이 파라미터의 경우 리소스 유형은 cluster이고 고유 식별자는 cluster:**myscalablecluster**와 같은 Aurora DB 클러스터의 이름입니다.
- **ScalableDimension** – 이 값을 rds:cluster:ReadReplicaCount로 설정하십시오.

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"
}
```

관련 주제

- [Aurora를 다른 AWS 서비스와 통합 \(p. 252\)](#)
- [Amazon Aurora DB 클러스터 관리 \(p. 190\)](#)

Amazon Aurora에서 기계 학습(ML) 기능 사용

아래는 Aurora 데이터베이스 애플리케이션에서 기계 학습(ML) 기능을 사용하는 방법에 대해 설명한 것입니다. 이 기능 덕분에 Amazon SageMaker 및 Amazon Comprehend 서비스를 사용하여 예측을 수행하는 데이터베이스 애플리케이션의 개발 과정이 간소화됩니다. ML 용어에서는 이러한 예측을 추론이라고 합니다.

이 기능은 여러 종류의 빠른 예측에 적합합니다. 짧은 자연 시간, 실시간 사용 사례(예: 부정 탐지), 광고 타겟팅, 제품 권장 사항 등을 예로 들 수 있습니다. 이 쿼리는 고객 프로필, 쇼핑 기록 및 제품 카탈로그 데이터를 Amazon SageMaker 모델로 전달합니다. 그런 다음 애플리케이션에서는 쿼리 결과로 반환되는 제품 권장 사항을 가져옵니다.

이 기능을 사용할 때 Amazon 기계 학습 서비스에서 사용할 수 있는 적절한 ML 모델, 노트북 등이 조직에 이미 있다면 도움이 됩니다. 데이터베이스 지식 및 ML 지식을 팀 구성원 사이에 배분할 수 있습니다. 데이터베이스 개발자는 애플리케이션의 SQL 및 데이터베이스 측에 집중할 수 있습니다. Aurora 기계 학습 기능 덕분에 애플리케이션은 저장된 함수 호출의 익숙한 데이터베이스 인터페이스를 통해 ML 처리를 사용할 수 있습니다.

주제

- [Aurora MySQL에서 기계 학습\(ML\) 사용 \(p. 642\)](#)
- [Aurora PostgreSQL에서 기계 학습\(ML\) 사용 \(p. 850\)](#)

Amazon Aurora DB 클러스터 백업 및 복구

이 단원에서는 Amazon Aurora DB 클러스터를 백업 및 복구하는 방법을 보여줍니다.

주제

- [Aurora DB 클러스터 백업 및 복원에 대한 개요 \(p. 268\)](#)
- [Aurora 백업 스토리지 사용량 파악 \(p. 270\)](#)
- [DB 클러스터 스냅샷 생성 \(p. 271\)](#)
- [DB 클러스터 스냅샷에서 복원 \(p. 273\)](#)
- [스냅샷 복사 \(p. 275\)](#)
- [DB 클러스터 스냅샷 공유 \(p. 284\)](#)
- [Amazon S3로 DB 스냅샷 데이터 내보내기 \(p. 290\)](#)
- [DB 클러스터를 지정된 시간으로 복원 \(p. 303\)](#)
- [스냅샷 삭제 \(p. 305\)](#)

Aurora DB 클러스터 백업 및 복원에 대한 개요

다음 섹션에서는 Aurora 백업과 AWS Management 콘솔을 사용한 Aurora DB 클러스터 복구 방법에 대한 정보를 찾아볼 수 있습니다.

Aurora DB 클러스터의 내결함성

Aurora DB 클러스터는 내결함성을 고려하여 설계되었습니다. 클러스터 볼륨은 단일 AWS 리전에 속하는 다른 가용 영역을 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.

단일 마스터 복제를 사용하는 DB 클러스터의 기본 인스턴스에 결함이 발생하면 Aurora가 다음 두 가지 방법 중 하나를 사용하여 자동으로 새로운 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.

다. 하지만, 일반적인 서비스 복구 시간은 120초 미만이지만 대부분 60초 미만에 복원됩니다. DB 클러스터의 가용성을 높이려면 최소 하나 이상의 Aurora 복제본을 둘 이상의 다른 가용 영역에 생성하는 것이 바람직합니다.

각 복제본에 우선 순위를 지정함으로써 장애 이후 기본 인스턴스로 승격할 Aurora 복제본 순서를 사용자 지정할 수 있습니다. 우선 순위 범위는 우선 순위가 가장 높은 0부터 가장 낮은 15까지입니다. 기본 인스턴스에 결합이 발생하면 Amazon RDS는 우선 순위가 더 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.

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

Note

Amazon Aurora는 외부 MySQL 데이터베이스 또는 RDS MySQL DB 인스턴스의 복제도 지원합니다. 자세한 내용은 [Aurora와 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터 간의 복제](#) (p. 567) 단원을 참조하십시오.

백업

Aurora은 클러스터 볼륨을 자동으로 백업한 후 백업 보존 기간 동안 복원 데이터를 보관합니다. Aurora 백업은 연속식 또는 증분식으로 이루어지기 때문에 백업 보존 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 전체 백업 보존 기간 중 복구 데이터를 누적 보관하기 때문에 백업 보존 기간을 넘어서까지 보관하려는 데이터는 스냅샷을 생성만 하면 됩니다. 새로운 DB 클러스터를 스냅샷에서 생성할 수 있기 때문입니다.

Note

- Amazon Aurora DB 클러스터의 경우, 기본 백업 보존 기간은 DB 클러스터 생성 방법과 상관없이 1일이 됩니다.
- Aurora에서 자동 백업을 비활성화할 수 없습니다. Aurora에 대한 백업 보존 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 [Aurora 백업 스토리지 사용량 파악](#) (p. 270) 단원을 참조하십시오. Aurora 백업 스토리지에 대한 요금 정보는 [Aurora용 Amazon RDS 요금](#)을 참조하십시오. 스냅샷과 연결된 Aurora 클러스터가 삭제된 후 해당 스냅샷을 저장하면 Aurora에 대해 표준 백업 스토리지 요금이 발생합니다.

데이터 복구

Aurora에서 유지되는 백업 데이터에서 또는 이전에 저장한 DB 클러스터 스냅샷에서 새 Aurora DB 클러스터를 생성하여 데이터를 복구할 수 있습니다. 백업 데이터에서 생성된 DB 클러스터의 새 사본을 백업 보존 기

간 종 임의 시점으로 빨리 복구할 수 있습니다. 백업 보존 기간 중 Aurora 백업의 연속 및 중분 특성은 복구 횟수를 늘리기 위해 데이터 스냅샷을 자주 캡처할 필요가 없다는 것을 의미합니다.

DB 인스턴스의 최근 또는 가장 빠른 복구 시간을 알아보려면 RDS 콘솔에서 [Latest Restorable Time](#) 또는 [Earliest Restorable Time](#) 값을 확인합니다. 이러한 값을 보는 방법은 [Amazon Aurora DB 클러스터 보기 \(p. 338\)](#) 단원을 참조하십시오. DB 클러스터의 최근 복구 시간은 DB 클러스터를 복구할 수 있는 가장 최근 시점을 나타내며 일반적으로 현재 시간에서 5분 이내입니다. 가장 빠른 복구 시간은 백업 보존 기간 내에서 클러스터 볼륨을 복구하려면 얼마나 후행해야 하는지 나타냅니다.

DB 클러스터의 복구가 언제 완료되었는지는 [Latest Restorable Time](#) 및 [Earliest Restorable Time](#) 값을 사용하여 확인할 수 있습니다. [Latest Restorable Time](#) 값과 [Earliest Restorable Time](#) 값은 복구 작업이 완료되기 전에는 NULL을 반환합니다. [Latest Restorable Time](#) 또는 [Earliest Restorable Time](#) 값이 NULL을 반환하면 백업 또는 복구 작업을 요청할 수 없습니다.

DB 클러스터를 특정 시점으로 복원에 대한 자세한 내용은 [DB 클러스터를 지정된 시간으로 복원 \(p. 303\)](#) 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 클러스터에서 데이터베이스 복제 \(p. 239\)](#) 단원을 참조하십시오.

역추적

이제 Aurora MySQL은 백업에서 데이터를 복구하지 않고도 특정 시간으로 DB 클러스터 "되감기"를 지원합니다. 자세한 내용은 [Aurora DB 클러스터 역추적 \(p. 509\)](#) 단원을 참조하십시오.

Aurora 백업 스토리지 사용량 파악

Aurora는 Aurora 백업 스토리지에 연속 백업(백업 보존 기간 내)과 스냅샷을 저장합니다. 백업 스토리지 사용량을 제어하려면 백업 보존 간격을 줄이거나 더 이상 필요하지 않은 경우 이전 수동 스냅샷을 제거하거나, 또는 두 가지 방법을 모두 사용할 수 있습니다. Aurora 백업에 대한 일반적인 정보는 [백업 \(p. 269\)](#) 단원을 참조하십시오. Aurora 백업 스토리지에 대한 요금 정보는 [Amazon Aurora 요금](#) 웹 페이지를 참조하십시오.

비용을 제어하기 위해 연속 백업과 수동 스냅샷(보존 기간 이후에도 계속 유지됨)에 사용되는 스토리지의 양을 모니터링할 수 있습니다. 그런 다음 백업 보존 간격을 줄이거나 더 이상 필요하지 않은 경우 수동 스냅샷을 제거할 수 있습니다.

다음과 같이 Amazon CloudWatch 지표인 `TotalBackupStorageBilled`, `SnapshotStorageUsed` 및 `BackupRetentionPeriodStorageUsed`를 사용하여 Aurora 백업에 사용되는 스토리지의 양을 검토 및 모니터링할 수 있습니다.

- `BackupRetentionPeriodStorageUsed`는 현재 연속 백업을 저장하는 데 사용되는 백업 스토리지 양(바이트)을 나타냅니다. 이 값은 보존 기간 중에 발생하는 변경의 양과 클러스터 볼륨의 크기에 따라 달라집니다. 하지만, 결제 목적상, 이 값은 보존 기간 동안 누적 클러스터 볼륨 크기를 초과하지 않습니다. 예를 들어 클러스터 `volumeBytesUsed` 크기가 107,374,182,400바이트(100 GiB)이고 보존 기간이 2일이면 `BackupRetentionPeriodStorageUsed`에 대한 최대 값은 214,748,364,800바이트(100GiB + 100GiB)입니다.
- `SnapshotStorageUsed`는 백업 보존 기간 이후에 수동 스냅샷을 저장하는 데 사용되는 백업 스토리지의 양(바이트)을 나타냅니다. 수동 스냅샷은 생성 타임스탬프가 보존 기간 내에 있는 동안에는 스냅샷 백업 스토리지에 포함되지 않습니다. 마찬가지로, 모든 자동 스냅샷도 스냅샷 백업 스토리지 계산에 포함되지 않습니다. 각 스냅샷의 크기는 스냅샷을 생성할 당시의 클러스터 볼륨 크기입니다. `SnapshotStorageUsed` 값은 유지하는 스냅샷 수와 각 스냅샷의 크기에 따라 달라집니다. 예

를 들어 보존 기간 이후에 계속 유지되는 수동 스냅샷이 하나 있고 해당 스냅샷을 생성할 당시의 클러스터 `volumeBytesUsed` 크기가 100GiB라고 가정해 보겠습니다. `SnapshotStorageUsed` 용량은 107,374,182,400이트(100GiB)입니다.

- `TotalBackupStorageBilled`는 `BackupRetentionPeriodStorageUsed` 및 `SnapshotStorageUsed`의 합계(바이트), 즉 당일의 클러스터 볼륨 크기와 똑같은 사용 가능한 백업 스토리지의 양을 나타냅니다. 사용 가능한 백업 스토리지는 최신 볼륨 크기와 같습니다. 예를 들어 클러스터 `VolumeBytesUsed` 크기가 100GiB이고, 보존 기간이 2일이고, 보존 기간이 경과한 수동 스냅샷 하나가 있는 경우 `TotalBackupStorageBilled`는 214,748,364,800바이트(200GiB + 100GiB - 100GiB)입니다.
- 이러한 지표는 각 Aurora DB 클러스터에 대해 독립적으로 계산됩니다.

[CloudWatch 콘솔](#)을 통해 CloudWatch 지표를 사용하여 Aurora 클러스터를 모니터링하고 보고서를 작성할 수 있습니다. CloudWatch 지표 사용법에 대한 자세한 내용은 [Amazon RDS 모니터링 개요 \(p. 327\)](#) 단원과 [Amazon RDS 지표 \(p. 331\)](#)의 지표 테이블을 참조하십시오.

Aurora DB 클러스터의 역추적 설정은 해당 클러스터의 백업 데이터 볼륨에 영향을 미치지 않습니다. Amazon은 역추적 데이터용 스토리지에 별도로 요금을 부과합니다. [Amazon Aurora 요금](#) 웹 페이지에서 역추적 요금 정보를 확인할 수도 있습니다.

스냅샷을 다른 사용자와 공유하는 경우 고객님이 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.

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

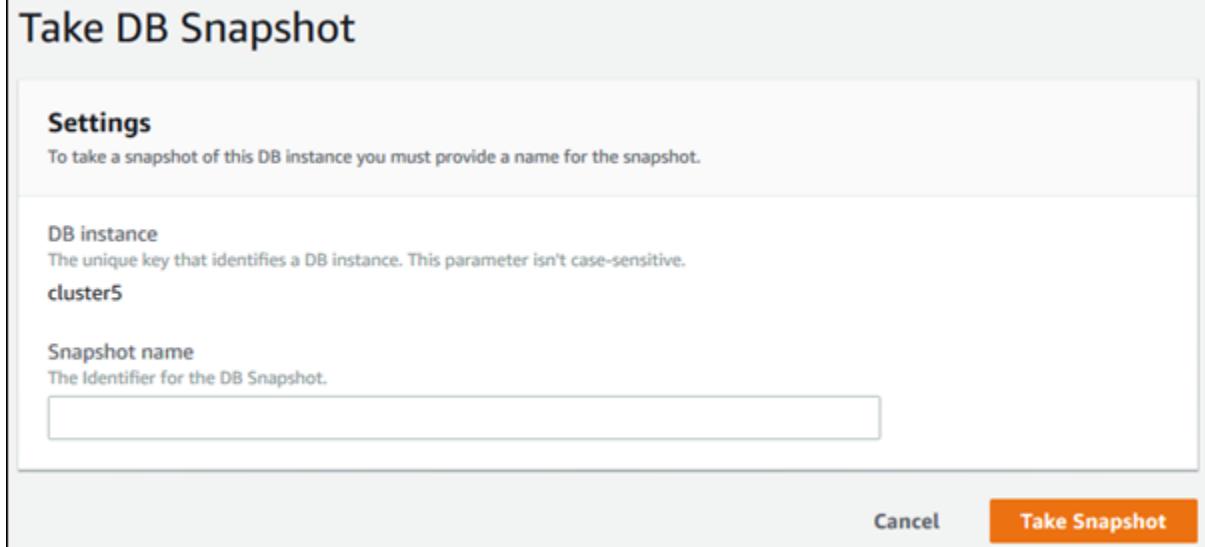
Amazon은 유지하는 Aurora 백업 및 스냅샷 데이터의 양과 유지 기간에 따라 요금을 청구합니다. Aurora 백업 및 스냅샷과 연결된 스토리지에 대한 자세한 내용은 [Aurora 백업 스토리지 사용량 파악 \(p. 270\)](#) 단원을 참조하십시오. Aurora 스토리지에 대한 요금 정보는 [Aurora용 Amazon RDS 요금](#)을 참조하십시오.

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 스냅샷을 생성할 수 있습니다.

콘솔

DB 클러스터 스냅샷을 생성하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. DB 인스턴스 목록에서 DB 클러스터의 라이터 인스턴스를 선택합니다.
4. 작업을 선택한 다음 스냅샷 만들기를 선택합니다.
[Take DB Snapshot] 창이 나타납니다.
5. 스냅샷 이름 상자에 DB 클러스터 스냅샷의 이름을 입력합니다.



6. Take Snapshot(스냅샷 만들기)을 선택합니다.

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

Linux, OS X, Unix의 경우:

```
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 콘솔에서 클러스터에 대한 세부 정보 페이지에 있는 Maintenance & backups(유지 관리 및 백업) 탭의 Snapshots(스냅샷) 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 풀링할 수 있습니다.

DB 클러스터 스냅샷에서 복원

Amazon RDS는 개별 데이터베이스가 아닌 전체 DB 클러스터를 백업하여 DB 인스턴스의 스토리지 볼륨 스냅샷을 생성합니다. 이 DB 클러스터 스냅샷에서 복원하여 DB 클러스터를 생성할 수 있습니다. DB 클러스터를 복원하는 경우 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.

Note

공유되고 동시에 암호화된 DB 클러스터 스냅샷에서는 DB 클러스터를 복원할 수 없습니다. 대신 DB 클러스터 스냅샷의 복사본을 생성하고 해당 복사본에서 DB 클러스터를 복원할 수 있습니다.

파라미터 그룹 고려 사항

복원된 DB 클러스터를 바른 파라미터 그룹과 연결할 수 있도록 생성한 DB 클러스터 스냅샷에 대한 파라미터 그룹을 유지하는 것이 좋습니다. DB 클러스터를 복원할 때 파라미터 그룹을 지정할 수 있습니다.

보안 그룹 고려 사항

DB 클러스터를 복원할 경우 기본 보안 그룹은 복원된 클러스터와 연결되도록 기본 설정되어 있습니다.

Note

- AWS CLI를 사용 중이라면 `restore-db-cluster-from-snapshot` 명령에 `--vpc-security-group-ids` 옵션을 포함하여 클러스터에 연결할 사용자 지정 보안 그룹을 지정할 수 있습니다.
- Amazon RDS API를 사용 중이라면 `RestoreDBClusterFromSnapshot` 작업에 `VpcSecurityGroupIds.VpcSecurityGroupId.N` 파라미터를 포함할 수 있습니다.
- Amazon RDS 콘솔에는 복원 중에 사용자 지정 보안 그룹을 연결할 수 있는 옵션이 없습니다.

복원이 완료되고 새 DB 클러스터가 사용 가능하게 되는 즉시 복원의 소스 스냅샷에서 사용하는 사용자 지정 보안 그룹을 연결할 수 있습니다. RDS 콘솔, AWS CLI `modify-db-cluster` 명령 또는 `ModifyDBCluster` Amazon RDS API 작업을 통해 DB 클러스터를 수정하여 이러한 변경을 적용해야 합니다. 자세한 내용은 [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.

Amazon Aurora 고려 사항

Aurora에서 DB 클러스터 스냅샷을 DB 클러스터로 복원합니다.

또한 Aurora MySQL에서는 DB 클러스터 스냅샷을 Aurora Serverless DB 클러스터로 복원할 수 있습니다. 자세한 내용은 [Aurora Serverless DB 클러스터 복원 \(p. 125\)](#) 단원을 참조하십시오.

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 Parallel Query 처리 \(p. 527\)](#) 단원을 참조하십시오.

스냅샷에서 복구

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 스냅샷에서 DB 클러스터 복원할 수 있습니다.

콘솔

DB 클러스터 스냅샷에서 DB 클러스터를 복원하려면

1. AWS Management 콘솔에 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.
5. DB 인스턴스 복원 페이지의 DB 인스턴스 식별자에 복원된 DB 클러스터의 이름을 입력합니다.
6. [Restore DB Instance]를 선택합니다.
7. DB 클러스터의 기능을 스냅샷을 생성할 때 원본으로 사용한 DB 클러스터의 기능으로 복원하려면, 보안 그룹을 사용하도록 DB 클러스터를 수정해야 합니다. 다음 단계에서는 사용자의 DB 클러스터가 VPC에 있다고 가정합니다. DB 클러스터가 VPC에 있지 않은 경우 EC2 Management Console을 사용하여 DB 클러스터에 필요한 보안 그룹을 찾아야 합니다.
 - a. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
 - b. 탐색 창에서 [Security Groups]를 선택합니다.
 - c. DB 클러스터에 사용할 보안 그룹을 선택합니다. 필요에 따라 이 보안 그룹을 EC2 인스턴스에 대한 보안 그룹과 연결하는 규칙을 추가합니다. 자세한 내용은 [동일한 VPC에 있는 EC2 인스턴스가 VPC 내에 있는 DB 인스턴스에 액세스 \(p. 1012\)](#) 단원을 참조하십시오.

AWS CLI

DB 클러스터 스냅샷에서 DB 클러스터를 복원하려면 AWS CLI 명령 `restore-db-cluster-from-snapshot`을 사용합니다.

이 예제에서는 `mydbclustersnapshot`이라는 이전에 생성된 DB 클러스터 스냅샷에서 복원합니다. `mynewdbcluster`라는 새 DB 클러스터로 복원합니다.

Example

Linux, OS X, Unix의 경우:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier mynewdbcluster \
--snapshot-identifier mydbclustersnapshot \
--engine aurora/aurora-postgresql
```

Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
```

```
--db-cluster-identifier mynewdbcluster ^
--snapshot-identifier mydbclustersnapshot ^
--engine aurora/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 기본 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. [create-db-instance](#) AWS CLI 명령을 호출하여 DB 클러스터를 위한 기본 인스턴스를 생성하십시오. 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 기본 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. RDS API 작업 [CreateDBInstance](#)를 호출하여 DB 클러스터의 기본 인스턴스를 만듭니다. DB 클러스터의 이름을 `DBClusterIdentifier` 파라미터 값으로 포함합니다.

스냅샷 복사

Amazon RDS에서는 자동 또는 수동 DB 클러스터 스냅샷을 복사할 수 있습니다. 스냅샷을 복사한 후 사본은 수동 스냅샷입니다.

동일한 AWS 리전 내의 시냅스들을 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 \(p. 284\)](#) 단원을 참조하십시오.

Note

Amazon은 유지하는 Aurora 백업 및 스냅샷 데이터의 양과 유지 기간에 따라 요금을 청구합니다. Aurora 백업 및 스냅샷과 연결된 스토리지에 대한 자세한 내용은 [Aurora 백업 스토리지 사용량 피약 \(p. 270\)](#) 단원을 참조하십시오. Aurora 스토리지에 대한 요금 정보는 [Aurora용 Amazon RDS 요금](#)을 참조하십시오.

제한 사항

다음은 스냅샷을 복사할 때 적용되는 몇몇 제한 사항입니다.

- AWS 리전 중국(베이징) 또는 중국(닝샤)로/에서는 스냅샷을 복사할 수 없습니다.
- AWS GovCloud(US-East)과 AWS GovCloud (US-West) 간에는 스냅샷을 복사할 수 있지만, AWS GovCloud (US) 리전과 다른 AWS 리전 간에는 스냅샷을 복사할 수 없습니다.
- 대상 스냅샷이 사용 가능해지기 전에 원본 스냅샷을 삭제할 경우 스냅샷 복사가 실패할 수 있습니다. 원본 스냅샷을 삭제하기 전에 대상 스냅샷의 상태가 AVAILABLE인지 확인하십시오.
- 계정당 단일 대상 리전으로 최대 5개의 스냅샷 복사 요청이 진행될 수 있습니다.
- 관련된 리전과 복사할 데이터 양에 따라 리전 간 스냅샷 복사를 완료하는 데 몇 시간이 걸릴 수 있습니다. 지정된 원본 AWS 리전에서 리전 간 DB 스냅샷 복사 요청이 많은 경우 진행 중인 복사 중 일부가 완료될 때까지 Amazon RDS에서 해당 원본 AWS 리전에 대한 새로운 리전 간 복사 요청을 대기열에 넣을 수 있습니다. 대기열에 있는 복사 요청에 대한 진행 정보는 표시되지 않습니다. 복사가 시작되면 진행 정보가 표시됩니다.

스냅샷 보존

DB 클러스터에 대한 자동 스냅샷을 비활성화하거나 DB 클러스터를 삭제하면 보존 기간이 끝날 때 Amazon RDS에서 자동 스냅샷이 삭제됩니다. 자동 DB 스냅샷을 더 오랜 기간 동안 유지하려면 자동 DB 스냅샷을 복사하여 수동 DB 스냅샷을 만들습니다. 그러면 사용자가 삭제할 때까지 스냅샷이 보존됩니다. 기본 스토리지 공간을 초과하는 경우 수동 스냅샷에 Amazon RDS 스토리지 비용이 적용될 수 있습니다.

백업 스토리지 비용에 대한 자세한 내용은 [Amazon RDS 요금](#)을 참조하십시오.

공유 스냅샷 복사

다른 AWS 계정이 공유하는 스냅샷을 복사할 수 있습니다. 다른 AWS 계정이 공유한 암호화된 스냅샷을 복사하려는 경우 스냅샷을 암호화하는 데 사용된 KMS 암호화 키에 대한 액세스 권한이 있어야 합니다.

동일한 AWS 리전에서는 암호화 여부와 관계없이 공유 DB 클러스터 스냅샷만 복사할 수 있습니다. 자세한 내용은 [암호화된 스냅샷 공유 \(p. 285\)](#) 단원을 참조하십시오.

암호화 처리

AWS KMS 암호화 키를 사용하여 암호화된 스냅샷을 복사할 수 있습니다. 암호화된 스냅샷을 복사할 경우 스냅샷의 사본도 암호화해야 합니다. 동일한 AWS 리전 내에 암호화된 스냅샷을 복사하는 경우, 원본 스냅샷과 동일한 KMS 암호화 키를 사용하여 사본을 암호화하거나, 다른 KMS 암호화 키를 지정할 수 있습니다. 리전 간에 암호화된 스냅샷을 복사하는 경우 KMS 키가 리전마다 다르므로 이 복사에는 원본 스냅샷에 사용한 것과 동일한 KMS 암호화 키를 사용할 수 없습니다. 대신에 대상 AWS 리전에 유효한 KMS 키를 지정해야 합니다.

소스 스냅샷은 복사 프로세스 전체에서 암호화를 유지합니다. 자세한 내용은 [Amazon Aurora 암호화된 DB 클러스터의 제한 \(p. 946\)](#) 단원을 참조하십시오.

Note

Amazon Aurora DB 클러스터 스냅샷의 경우 암호화되지 않은 DB 클러스터 스냅샷은 스냅샷을 복사할 때 암호화 할 수 없습니다.

AWS 리전 간 스냅샷 복사

원본 스냅샷의 AWS 리전과는 다른 AWS 리전에 스냅샷을 복사할 때 복사는 전체 스냅샷 복사입니다. 전체 스냅샷 복사에는 DB 인스턴스 복원에 필요한 모든 데이터 및 메타데이터가 포함됩니다.

관련된 AWS 리전과 복사할 데이터 양에 따라 리전 간 스냅샷 복사를 완료하는 데 몇 시간이 걸릴 수 있습니다. 경우에 따라 지정된 원본 AWS 리전으로부터 대량의 리전 간 스냅샷 복사 요청이 있을 수 있습니다. 이러

한 경우 진행 중인 복사 중 일부가 완료될 때까지 Amazon RDS에서 해당 원본 AWS 리전의 새로운 리전 간 복사 요청을 대기열에 넣을 수 있습니다. 대기열에 있는 복사 요청에 대한 진행 정보는 표시되지 않습니다. 복사가 시작되면 진행 정보가 표시됩니다.

Note

Aurora에서는 충분 스냅샷 복사를 지원하지 않습니다. Aurora DB 클러스터 스냅샷 복사본은 항상 전체 복사본입니다.

파라미터 그룹 고려 사항

리전 간에 스냅샷을 복사하는 경우 복사에는 원래 DB 클러스터에서 사용된 파라미터 그룹이 포함되지 않습니다. 스냅샷을 복원하여 새 DB 클러스터를 생성하면 이 DB 클러스터가 생성되는 AWS 리전의 기본 파라미터 그룹이 DB 인스턴스 또는 DB 클러스터에 배정됩니다. 새 DB 클러스터에 원본과 같은 파라미터를 지정하려면 다음을 수행해야 합니다.

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 새로운 또는 기존 파라미터 그룹을 추가합니다.

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. 원본 AWS 리전 밖으로 DB 클러스터 스냅샷을 복사하면 Amazon RDS 데이터 전송 요금이 발생합니다.

데이터 전송 요금에 대한 자세한 내용은 [Amazon RDS 요금](#)을 참조하십시오.

새 AWS 리전에 DB 클러스터 스냅샷 사본이 생성된 후 DB 클러스터 스냅샷 사본은 해당 AWS 리전의 다른 모든 DB 클러스터 스냅샷과 똑같이 동작합니다.

콘솔

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 최소하려면 대상 DB 클러스터 스냅샷이 [copying] 상태에 있는 동안 해당 DB 클러스터 스냅샷을 삭제합니다.

DB 클러스터 스냅샷을 복사하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 열니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 복사하려는 DB 클러스터 스냅샷의 확인란을 선택합니다.
4. 작업에서 스냅샷 복사를 선택합니다. [Make Copy of DB Snapshot] 페이지가 나타납니다.

Make Copy of DB Snapshot?

Settings

Source DB Snapshot

DB Snapshot Identifier for the automated snapshot being copied.

mydbinstancesnapshot

Destination Region [Info](#)

US East (N. Virginia)

New DB Snapshot Identifier

DB Snapshot Identifier for the new snapshot

Target Option Group (Optional) [Info](#)

No preference

Copy Tags [Info](#)

i Please note that depending on the amount of data to be copied and the Region you choose, this operation could take several hours to complete and the display on the progress bar could be delayed until setup is complete.

Encryption

Encryption [Info](#)

Enable encryption [Learn more](#)

Select to encrypt the given instance. Master key ids and aliases appear in the list after they have been created using the Key Management Service(KMS) console.

Disable encryption

[Cancel](#)

[Copy Snapshot](#)

5. (선택 사항) DB 클러스터 스냅샷을 다른 AWS 리전에 복사하려면 [Destination Region]에서 해당 AWS 리전을 선택합니다.
6. [New DB Snapshot Identifier]에 DB 클러스터 스냅샷 사본의 이름을 입력합니다.
7. 스냅샷의 태그와 값들을 스냅샷 사본에 복사하려면 [Copy Tags]를 선택합니다.
8. [Enable Encryption]에 대해 다음 옵션 중 하나를 선택합니다.
 - DB 클러스터 스냅샷이 암호화되지 않았고 사본도 암호화하지 않으려면 Disable encryption(암호화 비 활성화)을 선택합니다.
 - DB 클러스터 스냅샷이 암호화되지 않았지만 사본을 암호화하려면 암호화 활성화를 선택합니다. 이 경우 [Master Key]에 대해 DB 클러스터 스냅샷 사본을 암호화할 때 사용할 KMS 키 쇠팔자를 지정합니다.

- DB 클러스터 스냅샷이 암호화된 경우 암호화 활성화를 선택합니다. 이 경우 [Yes]가 이미 선택되어 있으므로 사본을 암호화해야 합니다. [Master Key]에 대해 DB 클러스터 스냅샷 사본을 암호화할 때 사용할 KMS 키 식별자를 지정합니다.
9. [Copy Snapshot]을 선택합니다.

AWS CLI 또는 Amazon RDS API를 사용하여 암호화되지 않은 DB 클러스터 스냅샷 복사

다음 단원의 절차에 따라 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 TargetDBClusterSnapshotIdentifier로 식별된 해당 DB 클러스터 스냅샷을 삭제합니다.

AWS CLI

DB 클러스터 스냅샷을 복사하려면 AWS CLI `copy-db-cluster-snapshot` 명령을 사용하십시오. 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 원본 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

Linux, OS X, Unix의 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
```

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 원본 AWS 리전의 ARN 형식이어야 합니다.
- **TargetDBClusterSnapshotIdentifier** – DB 클러스터 스냅샷의 새 사본의 식별자입니다.

다음 코드는 us-west-1 리전에 이름이 myclustersnapshotcopy인 arn:aws:rds:us-east-1:123456789012:cluster-snapshot:aurora-cluster1-snapshot-20130805 스냅샷 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 단원의 절차에 따라 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 TargetDBClusterSnapshotIdentifier로 식별된 해당 DB 클러스터 스냅샷을 삭제합니다.

AWS CLI

DB 클러스터 스냅샷을 복사하려면 AWS CLI [copy-db-cluster-snapshot](#) 명령을 사용하십시오. 스냅샷을 다른 AWS 리전에 복사하려는 경우, 스냅샷을 복사할 AWS 리전에서 명령을 실행합니다.

다음 옵션을 사용하여 암호화된 DB 클러스터 스냅샷을 복사할 수 있습니다.

- **--source-region** – 스냅샷을 다른 AWS 리전에 복사하려는 경우 암호화된 DB 클러스터 스냅샷을 복사할 소스 AWS 리전을 지정합니다.

스냅샷을 다른 AWS 리전에 복사하려고 하며 source-region을 지정하지 않은 경우, 대신 pre-signed-url 옵션을 지정해야 합니다. pre-signed-url 값은 DB 클러스터 스냅샷을 복사해올 원본 AWS 리전에서 CopyDBClusterSnapshot 작업이 호출되도록 하는 서명 버전 4의 서명된 요청이 포함된 URL이어야 합니다. pre-signed-url에 대한 자세한 내용은 [copy-db-cluster-snapshot](#)을 참조하십시오.

- **--source-db-cluster-snapshot-identifier** – 복사할 암호화된 DB 클러스터 스냅샷의 식별자입니다. 스냅샷을 다른 AWS 리전에 복사하려는 경우, 이 식별자는 원본 AWS 리전의 ARN 형식이어야 합니다. 그럴 경우 source-db-cluster-snapshot-identifier에 지정된 AWS 리전은 --source-region에 대해 지정된 AWS 리전과 일치해야 합니다.
- **--target-db-cluster-snapshot-identifier** – 암호화된 DB 클러스터 스냅샷의 새 사본의 식별자입니다.

- **--kms-key-id** – DB 클러스터 스냅샷의 사본을 암호화하는 데 사용할 키의 KMS 키 식별자입니다.

DB 클러스터 스냅샷이 암호화되어 있으며, 동일한 AWS 리전에 스냅샷을 복사하려고 하고, 사본을 암호화하는 데 사용할 새 KMS 암호화 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 예제에서는 암호화된 DB 클러스터 스냅샷을 us-west-2 리전에서 us-east-1 리전으로 복사합니다. 이 명령은 us-east-1 리전에서 호출됩니다.

Example

Linux, OS X, Unix의 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 \
--source-region us-west-2 \
--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 ^
--source-region us-west-2 ^
--kms-key-id my-us-east-1-key
```

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 원본 AWS 리전의 ARN 형식이어야 합니다.
- **TargetDBClusterSnapshotIdentifier** – 암호화된 DB 클러스터 스냅샷의 새 사본의 식별자입니다.
- **KmsKeyId** – DB 클러스터 스냅샷의 사본을 암호화하는 데 사용할 키의 KMS 키 식별자입니다.

DB 클러스터 스냅샷이 암호화되어 있으며, 동일한 AWS 리전에 스냅샷을 복사하려고 하고, 사본을 암호화하는 데 사용할 새 KMS 암호화 키를 지정하려는 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 원본 AWS 리전에서 CopyDBClusterSnapshot 작업이 호출되도록 하는 서명 버전 4의 서명된 요청이 포함된 URL이어야 합니다. 미리 서명된 URL 사용에 대한 자세한 내용은 [CopyDBClusterSnapshot](#)을 참조하십시오.

미리 서명된 URL을 수동이 아닌 자동으로 생성하려면 그 대신 AWS CLI `copy-db-cluster-snapshot` 명령을 `--source-region` 옵션과 함께 사용합니다.

다음 코드 예제에서는 암호화된 DB 클러스터 스냅샷을 us-west-2 리전에서 us-east-1 리전으로 복사합니다. 이 작업은 us-east-1 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
```

계정 간에 DB 클러스터 스냅샷 복사

Amazon RDS API `ModifyDBClusterSnapshotAttribute` 및 `CopyDBClusterSnapshot` 작업을 사용하여 여러 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 `AttributeName` 파라미터에 `restore`를 지정하고 `ValuesToAdd` 파라미터에 계정 B의 ID를 지정하여 `ModifyDBClusterSnapshotAttribute`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 IAM 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 `AttributeName` 파라미터에 `restore`를 지정하고 `ValuesToAdd` 파라미터에 대상 계정의 ID를 지정하여 [ModifyDBClusterSnapshotAttribute](#)를 호출합니다.

계정 987654321을 사용하는 다음 예를 실행하면 123451234512 및 123456789012라는 두 개의 AWS 계정 식별자가 manual-snapshot1라는 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=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36ddbb3
```

2. 대상 계정에서 [CopyDBClusterSnapshot](#)을 호출하고, `SourceDBClusterSnapshotIdentifier` 파라미터를 사용하여 복사할 DB 클러스터 스냅샷의 ARN을 지정합니다. 이 ARN에는 원본 계정의 ID가 포함되어야 합니다.

계정 123451234512를 사용하는 다음 예를 실행하면 계정 987654321에서 DB 클러스터 스냅샷 aurora-cluster1-snapshot-20130805가 복사되고, dbclustersnapshot1라는 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 `AttributeName` 파라미터에 `restore`를 지정하고 `ValuesToAdd` 파라미터에 대상 계정의 ID를 지정하여 `ModifyDBClusterSnapshotAttribute`를 호출합니다.

계정 987654321을 사용하는 다음 예를 실행하면 123451234512 및 123456789012라는 두 개의 AWS 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=ef38f1ce3dab4e1dbf113d8d2a265c67d17ece1999ffd36be85714ed36ddbb3
```

2. DB 클러스터 스냅샷의 원본 계정에서 KMS 키의 키 정책을 업데이트합니다. 먼저 대상 계정의 ARN을 `Principal`로 추가한 다음 `kms:CreateGrant` 작업을 허용합니다. 자세한 내용은 [AWS KMS 암호화 키에 대한 액세스 허용 \(p. 285\)](#) 단원을 참조하십시오.
3. 대상 계정에서 IAM 사용자를 선택하거나 만들고, 해당 사용자에 IAM 정책을 연결합니다. 그러면 사용자가 KMS 키를 사용하여 암호화된 DB 클러스터 스냅샷을 복사할 수 있습니다. 자세한 내용은 [암호화된 스냅샷을 복사할 수 있도록 IAM 정책 만들기 \(p. 286\)](#) 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 123451234512를 사용하는 다음 예를 실행하면 계정 987654321에서 DB 클러스터 스냅샷 `aurora-cluster1-snapshot-20130805`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 RDS를 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.

스냅샷 복사에 대한 자세한 정보는 [스냅샷 복사 \(p. 275\)](#) 단원을 참조하십시오. DB 클러스터 스냅샷에서 DB 인스턴스를 복원하는 방법에 대한 자세한 정보는 [DB 클러스터 스냅샷에서 복원 \(p. 273\)](#) 단원을 참조하십시오.

DB 클러스터 스냅샷에서 DB 클러스터를 복원하는 방법에 대한 자세한 정보는 [Aurora DB 클러스터 백업 및 복원에 대한 개요 \(p. 268\)](#)을 참조하십시오.

최대 20개의 다른 AWS 계정과 수동 스냅샷을 공유할 수 있습니다. 암호화되지 않은 수동 스냅샷을 퍼블릭으로 공유할 수 있습니다. 그러면 모든 AWS 계정에서 해당 스냅샷을 사용할 수 있습니다. 스냅샷을 퍼블릭으로 공유할 경우 비공개 정보가 퍼블릭 스냅샷에 포함되지 않도록 유의하십시오.

수동 스냅샷을 다른 AWS 계정과 공유할 경우 다음과 같은 제한이 적용됩니다.

- AWS Command Line Interface(AWS CLI) 또는 Amazon RDS API를 사용하여 공유 스냅샷에서 DB 인스턴스 또는 DB 클러스터를 복원할 때는 공유 스냅샷의 Amazon 리소스 이름(ARN)을 스냅샷 식별자로 지정해야 합니다.

암호화된 스냅샷 공유

[Amazon Aurora 리소스 암호화 \(p. 945\)](#)에 설명된 대로 AES-256 암호화 알고리즘을 사용하여 "저장 상태"에서 암호화된 DB 클러스터 스냅샷을 공유할 수 있습니다. 이렇게 하려면 다음 단계를 따라야 합니다.

1. 스냅샷을 암호화하는 데 사용한 AWS Key Management Service(AWS KMS) 암호화 키를 해당 스냅샷에 액세스할 수 있도록 하려는 계정과 공유합니다.

KMS 키 정책에 다른 계정을 추가하여 AWS KMS 암호화 키를 다른 AWS 계정과 공유할 수 있습니다. 키 정책 업데이트에 관한 세부 정보는 AWS KMS 개발자 안내서의 [키 정책](#) 단원을 참조하십시오. 키 정책 생성의 예는 이번 주제 후반부의 [AWS KMS 암호화 키에 대한 액세스 허용 \(p. 285\)](#) 단원을 참조하십시오.

2. AWS Management 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 암호화된 스냅샷을 다른 계정과 공유합니다.

이러한 제한은 암호화된 스냅샷 공유에 적용됩니다.

- 암호화된 스냅샷을 퍼블릭으로는 공유할 수 없습니다.
- 스냅샷을 공유한 AWS 계정의 기본 AWS KMS 암호화 키를 사용하여 암호화된 스냅샷은 공유할 수 없습니다.

AWS KMS 암호화 키에 대한 액세스 허용

다른 AWS 계정이 사용자 계정에서 공유된 암호화된 DB 클러스터 스냅샷을 복사하려면 스냅샷을 공유하는 계정에 스냅샷을 암호화한 KMS 키에 대한 액세스 권한이 있어야 합니다. 다른 AWS 계정이 AWS KMS 키에 액세스할 수 있도록 허용하려면 KMS 키에 대한 키 정책을 공유하는 AWS 계정의 ARN으로 업데이트합니다. 즉, KMS 키 정책에서 `Principal`을 포함하고 `kms:CreateGrant` 작업을 허용합니다.

AWS 계정에 사용자의 KMS 암호화 키에 대한 액세스 권한을 부여한 후에는, 암호화된 스냅샷을 복사하려면 해당 AWS 계정에서 AWS Identity and Access Management(IAM) 사용자를 만들어야 합니다(아직 없는 경우). 또한 IAM 사용자가 해당 KMS 키를 사용하여 암호화된 DB 클러스터 스냅샷을 복사할 수 있도록 허용하는 IAM 정책을 이 AWS 계정에 연결해야 합니다. KMS 보안 제약으로 인해 이 계정은 IAM 사용자여야 하며 루트 AWS 계정 자격 증명일 수 없습니다.

다음 키 정책 예에서는 사용자 111122223333이 KMS 암호화 키의 소유자이고 사용자 444455556666이 키를 공유하는 계정입니다. 업데이트된 이 키 정책으로 인해 AWS 계정이 KMS 키에 액세스할 수 있습니다.

다. 사용자 444455556666에 대한 AWS 계정 자격 증명의 ARN을 정책에 대한 Principal로 포함하고 kms:CreateGrant 작업을 허용했습니다.

```
{  
    "Id": "key-policy-1",  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "Allow use of the key",  
            "Effect": "Allow",  
            "Principal": {"AWS": [  
                "arn:aws:iam::111122223333:user/KeyUser",  
                "arn:aws:iam::444455556666: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::111122223333:user/KeyUser",  
                "arn:aws:iam::444455556666:root"  
            ]},  
            "Action": [  
                "kms:CreateGrant",  
                "kms>ListGrants",  
                "kms:RevokeGrant"  
            ],  
            "Resource": "*",  
            "Condition": {"Bool": {"kms:GrantIsForAWSResource": true}}  
        }  
    ]  
}
```

암호화된 스냅샷을 복사할 수 있도록 IAM 정책 만들기

외부 AWS 계정에 사용자의 KMS 키에 대한 액세스 권한을 부여한 후에는 해당 AWS 계정의 소유자가 이 계정에 대해 생성된 IAM 사용자가 해당 KMS 키를 사용하여 암호화된 스냅샷을 복사할 수 있게 하는 정책을 만들 수 있습니다.

다음 예에서는 AWS 계정 444455556666에 대해 IAM 사용자에게 추가할 수 있으며 IAM 사용자가 us-west-2 리전에서 KMS 키 c989c1dd-a3f2-4a5d-8d96-e793d082ab26을 사용하여 암호화된 AWS 계정 111122223333에서 공유 스냅샷을 복사할 수 있게 하는 정책을 보여 줍니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowUseOfTheKey",  
            "Effect": "Allow",  
            "Action": [  
                "kms:Encrypt",  
                "kms:Decrypt",  
                "kms:ReEncrypt*",  
                "kms:GenerateDataKey*",  
                "kms:DescribeKey"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

```
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms>CreateGrant",
        "kms:RetireGrant"
    ],
    "Resource": [ "arn:aws:kms:us-west-2:111122223333:key/c989c1dd-a3f2-4a5d-8d96-e793d082ab26" ]
},
{
    "Sid": "AllowAttachmentOfPersistentResources",
    "Effect": "Allow",
    "Action": [
        "kms>CreateGrant",
        "kms>ListGrants",
        "kms:RevokeGrant"
    ],
    "Resource": [ "arn:aws:kms:us-west-2:111122223333:key/c989c1dd-a3f2-4a5d-8d96-e793d082ab26" ],
    "Condition": {
        "Bool": {
            "kms:GrantIsForAWSResource": true
        }
    }
}
]
```

키 정책 업데이트에 관한 세부 정보는 AWS KMS 개발자 안내서의 [키 정책](#) 단원을 참조하십시오.

스냅샷 공유

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터 스냅샷을 공유할 수 있습니다.

콘솔

Amazon RDS 콘솔을 사용하여 최대 20개의 AWS 계정과 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 콘솔에 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. 작업에서 스냅샷 공유를 선택합니다.
5. DB 스냅샷 공개 여부에 대해 다음 옵션 중 하나를 선택합니다.
 - 소스가 암호화되어 있지 않은 경우 Public(퍼블릭)을 선택하여 모든 AWS 계정이 수동 DB 클러스터 스냅샷에서 DB 클러스터를 복원하도록 허용하거나, Private(프라이빗)을 선택하여 지정한 AWS 계정만 수동 DB 클러스터 스냅샷에서 DB 클러스터를 복원하도록 허용합니다.

Warning

DB snapshot visibility(DB 스냅샷 가시성)를 Public(퍼블릭)으로 설정한 경우 모든 AWS 계정은 수동 DB 클러스터 스냅샷에서 DB 클러스터를 복원하고 사용자 데이터에 액세스할 수 있습니다. 프라이빗 정보가 포함된 수동 DB 클러스터 스냅샷을 Public(퍼블릭)으로 공유하지 마십시오.

- 원본 DB 클러스터가 암호화되어 있는 경우 암호화된 스냅샷을 퍼블릭으로 공유할 수 없으므로 DB snapshot visibility(DB 스냅샷 가시성)는 Private(프라이빗)으로 설정됩니다.

- 수동 스냅샷에서 DB 클러스터를 복원하도록 허용할 계정의 AWS 계정 식별자를 AWS 계정 ID에 입력한 다음 Add(추가)를 선택합니다. 이 과정을 반복하여 AWS 계정 식별자를 최대 20개까지 추가합니다.

허용된 계정 목록에 AWS 계정 식별자를 추가하다가 실수하는 경우, 잘못된 AWS 계정 식별자의 오른쪽에 있는 [Delete]를 선택하여 목록에서 해당 식별자를 삭제할 수 있습니다.

Snapshot permissions

Preferences

You are sharing an unencrypted DB snapshot. When you share an unencrypted DB snapshot, you give the other account permission to make a copy of the DB snapshot and to restore a database from your DB snapshot.

DB snapshot
testoracletags-snap

DB snapshot visibility
 Private
 Public

AWS account ID

AWS account ID	Delete
Please add AWS account ID	

- 수동 스냅샷을 복원할 수 있도록 허용할 모든 AWS 계정의 식별자를 추가한 후 [Save]를 선택하여 변경 내용을 저장합니다.

AWS 계정과 수동 DB 클러스터 스냅샷 공유를 중지하려면

- AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
- 탐색 창에서 [Snapshots]를 선택합니다.
- 공유를 중지할 수동 스냅샷을 선택합니다.
- 작업을 선택한 다음 스냅샷 공유를 선택합니다.
- AWS 계정의 권한을 제거하려면 권한 있는 계정 목록에서 해당 계정의 AWS 계정 식별자에 대해 [Delete]를 선택합니다.

Snapshot permissions

Preferences

You are sharing an unencrypted DB snapshot. When you share an unencrypted DB snapshot, you give the other account permission to make a copy of the DB snapshot and to restore a database from your DB snapshot.

DB snapshot

testoracletags-snap

DB snapshot visibility

- Private
- Public

AWS account ID

Add

AWS account ID

Delete

Delete

Cancel

Save

- [Save]를 선택하여 변경 사항을 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 계정의 ID 목록을 추가합니다.

다음 예제에서는 두 AWS 계정 식별자 123451234512 및 1234567890120에게 `manual-cluster-snapshot1`이라는 DB 클러스터 스냅샷을 복원할 수 있는 권한을 부여하고 `all` 속성 값을 제거하여 DB 스냅샷을 프라이빗으로 표시합니다.

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-add '[ "111122223333", "444455556666" ]'
```

목록에서 AWS 계정 식별자를 제거하려면 `--values-to-remove` 파라미터를 사용합니다. 다음 예는 AWS 계정 ID 444455556666의 스냅샷 복원을 방지합니다.

```
aws rds modify-db-cluster-snapshot-attribute \
--db-cluster-snapshot-identifier manual-cluster-snapshot1 \
--attribute-name restore \
--values-to-remove '[ "444455556666" ]'
```

RDS API

Amazon RDS API를 사용하여 수동 DB 클러스터 스냅샷을 다른 AWS 계정과 공유할 수도 있습니다. 이를 위해 [ModifyDBClusterSnapshotAttribute](#) 작업을 호출합니다. AttributeName에 대해 restore를 지정하고 ValuesToAdd 파라미터를 사용하여 수동 스냅샷의 복원 권한이 있는 AWS 계정을 위한 ID 목록을 추가합니다.

수동 스냅샷을 퍼블릭으로 설정하고 모든 AWS 계정에서 복원할 수 있도록 하려면 a11 값을 사용합니다. 단, 일부 AWS 계정에만 제공할 비공개 정보가 포함된 수동 스냅샷에 대해 a11 값을 추가하지 않도록 유의합니다. 또한 암호화된 스냅샷에 대해 a11을 지정하지 마십시오. 그러한 스냅샷을 퍼블릭으로 설정하는 것은 지원되지 않습니다.

AWS 계정의 공유 권한을 제거하려면 AttributeName을 restore로 설정한 [ModifyDBClusterSnapshotAttribute](#) 작업과 valuesToRemove 파라미터를 사용합니다. 수동 스냅샷을 비공개로 하려면 restore 속성에 대한 값 목록에서 a11 값을 제거합니다.

스냅샷을 복원할 수 있도록 허용된 모든 AWS 계정을 나열하려면 [DescribeDBClusterSnapshotAttributes](#) API 작업을 사용합니다.

Amazon S3로 DB 스냅샷 데이터 내보내기

DB 스냅샷 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. 내보내기 프로세스는 백그라운드에서 실행되며 활성 DB 클러스터의 성능에는 영향을 주지 않습니다.

DB 스냅샷을 내보낼 때 Amazon Aurora는 해당 스냅샷에서 데이터를 추출하여 계정의 Amazon S3 버킷에 저장합니다. 데이터는 압축되고 일관된 Apache Parquet 형식으로 저장됩니다.

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.

스냅샷 내보내기는 다음 AWS 리전에서 지원됩니다.

- 미국 동부(버지니아 북부)
- 미국 동부(오하이오)
- 미국 서부(오레곤)
- 유럽(아일랜드)
- 아시아 태평양(도쿄)

Note

S3 내보내기가 지원되지 않는 AWS 리전에서 지원되는 리전으로 스냅샷을 복사한 다음 복사본을 내보낼 수 있습니다. S3 버킷이 복사본과 동일한 AWS 리전에 있어야 합니다.

다음 목록에는 스냅샷 데이터를 Amazon S3로 내보내는 것이 지원되는 엔진 버전이 나와 있습니다.

Aurora MySQL

- MySQL 5.6과 호환되는 버전 1.19.2, 1.19.3, 1.19.4 및 1.19.5
- MySQL 5.7과 호환되는 버전 2.04.4, 2.04.5 및 2.04.6

Aurora MySQL 엔진에 대한 자세한 정보는 [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#) 단원을 참조하십시오.

Aurora PostgreSQL

- PostgreSQL 9.6.11과 호환되는 버전 1.4

- PostgreSQL 9.6.12와 호환되는 버전 1.5
- PostgreSQL 10.6과 호환되는 버전 2.2
- PostgreSQL 10.7과 호환되는 버전 2.3

Aurora PostgreSQL 엔진에 대한 자세한 정보는 [Amazon Aurora PostgreSQL의 엔진 버전 \(p. 906\)](#) 단원을 참조하십시오.

주제

- [스냅샷 데이터 내보내기 개요 \(p. 291\)](#)
- [Amazon S3 버킷에 대한 액세스 권한 설정 \(p. 291\)](#)
- [Amazon S3 버킷으로 스냅샷 내보내기 \(p. 293\)](#)
- [스냅샷 내보내기 모니터링 \(p. 295\)](#)
- [스냅샷 내보내기 작업 취소 \(p. 297\)](#)
- [Amazon S3 버킷으로 내보내기를 할 때 데이터 변환 \(p. 298\)](#)

스냅샷 데이터 내보내기 개요

다음 절차에서는 DB 스냅샷 데이터를 Amazon S3 버킷으로 내보내는 방법을 간략하게 보여 줍니다. 자세한 내용은 다음 섹션을 참조하십시오.

DB 스냅샷 데이터를 Amazon S3로 내보내려면

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 버킷이 스냅샷과 동일한 AWS 리전에 있어야 합니다. 자세한 내용은 [내보낼 Amazon S3 버킷 식별 \(p. 292\)](#) 단원을 참조하십시오.
 - b. 서버 측 암호화를 위한 AWS Key Management Service(AWS KMS) 키를 생성합니다. KMS 키는 스냅샷 내보내기 작업에서 S3에 내보내기 데이터를 기록할 때 KMS 서버 측 암호화를 설정하는 데 사용됩니다. 자세한 정보는 [Amazon Aurora 리소스 암호화 \(p. 945\)](#) 단원을 참조하십시오.
 - c. 스냅샷 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 \(p. 292\)](#) 단원을 참조하십시오.
3. 콘솔 또는 `start-export-task` CLI 명령을 사용하여 Amazon S3로 스냅샷을 내보냅니다. 자세한 내용은 [Amazon S3 버킷으로 스냅샷 내보내기 \(p. 293\)](#) 단원을 참조하십시오.
 4. Amazon S3 버킷에서 내보낸 데이터에 액세스하려면 Amazon Simple Storage Service 콘솔 사용 설명서의 [객체 업로드, 다운로드 및 관리](#)를 참조하십시오.

Amazon S3 버킷에 대한 액세스 권한 설정

DB 스냅샷 데이터를 Amazon S3 파일로 내보내려면 먼저 스냅샷에 Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다. 그런 다음 Amazon Aurora 서비스가 Amazon S3 버킷에 쓰기 작업을 할 수 있게 허용하는 IAM 역할을 생성합니다.

주제

- [내보낼 Amazon S3 버킷 식별 \(p. 292\)](#)

- IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공 (p. 292)

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 콘솔 사용 설명서의 [S3 버킷의 속성을 보려면 어떻게 합니까?](#), [Amazon S3 버킷에서 기본 암호화를 활성화하려면 어떻게 해야 합니까?](#), [S3 버킷을 어떻게 생성합니까?](#)를 참조하십시오.

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.

이를 위해 해당 버킷에 대한 액세스 권한을 부여하는 IAM 정책을 생성합니다. 그런 다음 IAM 역할을 생성하여 정책을 이 역할에 연결합니다. 나중에 스냅샷 내보내기 작업에 IAM 역할을 할당합니다.

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.

Amazon Aurora에서 S3 버킷으로의 파일 전송을 허용하려면 정책에 다음과 같은 필수 작업을 포함시킵니다.

- s3:PutObject*
- s3:GetObject*
- s3>ListBucket
- s3>DeleteObject*
- s3:GetBucketLocation

버킷에서 S3 버킷 및 객체를 식별하려면 정책에 다음 리소스를 포함시킵니다. 다음 리소스 목록은 Amazon S3에 액세스하기 위한 Amazon 리소스 이름(ARN) 형식을 보여 줍니다.

- arn:aws:s3:::*your-s3-bucket*
- arn:aws:s3:::*your-s3-bucket*/*

Amazon Aurora에 대한 IAM 정책을 생성하는 방법에 대한 자세한 내용은 [IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 \(p. 979\)](#) 단원을 참조하십시오. IAM 사용 설명서의 [자습서: 첫 번째 고객 관리형 정책 생성 및 연결](#)도 참조하십시오.

다음 AWS CLI 명령은 이 옵션으로 ExportPolicy라는 IAM 정책을 생성합니다. *your-s3-bucket*라는 버킷에 대한 액세스 권한을 부여합니다.

Note

정책을 생성한 후 정책의 ARN을 기록해 둍니다. IAM 역할에 정책을 연결할 때 이후 단계에 ARN이 필요합니다.

```
aws iam create-policy --policy-name ExportPolicy --policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {
```

```
{  
    "Effect": "Allow",  
    "Action": [  
        "s3>ListBucket",  
        "s3:GetBucketLocation"  
    ],  
    "Resource": [  
        "arn:aws:s3:::*"  
    ]  
},  
{  
    "Effect": "Allow",  
    "Action": [  
        "s3:PutObject*",  
        "s3:GetObject*",  
        "s3>DeleteObject*"  
    ],  
    "Resource": [  
        "arn:aws:s3:::your-s3-bucket",  
        "arn:aws:s3:::your-s3-bucket/*"  
    ]  
}  
]  
}  
}'
```

2. IAM 역할 생성. 이렇게 하면 Aurora이 사용자 대신 이 IAM 역할을 수임하여 Amazon S3 버킷에 액세스 할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자에게 권한을 위임하기 위한 역할 생성](#)을 참조하십시오.

다음 예제에서는 AWS CLI 명령을 사용해 `rds-s3-export-role`이라는 역할을 생성하는 방법을 보여 줍니다.

```
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
```

Amazon S3 버킷으로 스냅샷 내보내기

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 S3으로 데이터를 추출합

니다. 이 단계 동안의 작업 진행 상황은 STARTING으로 표시됩니다. 작업이 Amazon S3으로 데이터 내보내기로 전환되면 진행 상황이 IN_PROGRESS로 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. 테이블에 분할에 적합한 열이 없으면 내보내기가 더 느린 단일 스레드 프로세스가 됩니다.

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 스냅샷을 Amazon S3로 내보낼 수 있습니다.

콘솔

DB 스냅샷을 내보내려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Snapshots(스냅샷)을 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 데이터의 양을 선택합니다.
 - 스냅샷의 모든 데이터를 내보내려면 모두를 선택합니다.
 - 스냅샷의 특정 부분을 내보내려면 Partial(부분)을 선택합니다. 스냅샷의 어떤 부분을 내보낼지 식별하려면 Identifiers(식별자)에 하나 이상의 테이블을 입력합니다.
 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 선택합니다. [IAM 역할을 사용하여 Amazon S3 버킷에 대한 액세스 권한 제공 \(p. 292\)](#)의 단계에 따라 역할을 생성한 경우에는 해당 역할을 선택합니다.
 10. 마스터 키에 내보낸 데이터를 암호화하는 데 사용할 키의 ARN을 입력합니다.
 11. Export to Amazon S3(Amazon S3로 내보내기)를 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 스냅샷을 Amazon S3로 내보내려면 `start-export-task` 명령을 다음과 같은 필수 옵션과 함께 사용합니다.

- `--export-task-identifier`
- `--source-arn`
- `--s3-bucket-name`
- `--iam-role-arn`
- `--kms-key-id`

다음 예제에서 스냅샷 내보내기 작업의 이름은 `my_snapshot_export`이고, 이 작업은 스냅샷을 `my_export_bucket`이라는 S3 버킷으로 내보냅니다.

Example

Linux, OS X, Unix의 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 my_export_bucket \
--iam-role-arn iam_role \
--kms-key-id master_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 my_export_bucket ^
--iam-role-arn iam_role ^
--kms-key-id master_key
```

샘플 출력은 다음과 같습니다.

```
{
    "Status": "STARTING",
    "IamRoleArn": "iam_role",
    "ExportTime": "2019-08-12T01:23:53.109Z",
    "S3Bucket": "my_export_bucket",
    "PercentProgress": 0,
    "KmsKeyId": "master_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 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 스냅샷 내보내기를 모니터링할 수 있습니다.

콘솔

DB 스냅샷 내보내기를 모니터링하려면

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

2. 탐색 창에서 Snapshots(스냅샷)을 선택합니다.
3. 스냅샷 내보내기 목록을 모니터링하려면 Exports in Amazon S3(Amazon S3의 내보내기) 탭을 선택합니다.
4. 특정 스냅샷 내보내기에 대한 정보를 보려면 해당 내보내기 작업을 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 스냅샷 내보내기를 모니터링하려면 [describe-export-tasks](#) 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": "awsexamplebucket",
            "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": "awsexamplebucket1",
            "PercentProgress": 100,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/h3yCo8nzbEXAMPLEKEY",
            "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 edgcuc-export 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": "awsexamplebucket2",
            "PercentProgress": 0,
            "KmsKeyId": "arn:aws:kms:AWS_Region:123456789012:key/2Zp9Utk/h3yCo8nzbEXAMPLEKEY",
        }
    ]
}
```

```
"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 `describe-export-tasks` 명령과 함께 `--export-task-identifier` 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.

스냅샷 내보내기 작업 취소

AWS Management 콘솔, 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 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Snapshots(스냅샷)을 선택합니다.
3. Exports in Amazon S3(Amazon S3에서 내보내기) 탭을 선택합니다.
4. 취소할 스냅샷 내보내기 작업을 선택합니다.
5. 취소를 선택합니다.
6. 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": "awsexamplebucket",
    "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.

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 참조하십시오.

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 참고하십시오.

Note

- 일부 문자는 데이터베이스 테이블 열 이름에서 지원되지 않습니다. 열 이름에 다음 문자가 포함되어 있는 테이블은 내보내기를 수행하는 동안 건너뛰기가 됩니다.

```
,;{}()\n\t=
```

- 데이터에 500MB에 근접하거나 이 보다 큰 값이 포함되어 있으면 내보내기가 실패합니다.
- 데이터베이스, 스키마 또는 테이블 이름에 공백이 포함되어 있으면 부분 내보내기가 지원되지 않습니다. 그러나 전체 DB 스냅샷을 내보낼 수는 있습니다.

주제

- Parquet로 MySQL 데이터 유형 매핑 (p. 299)
- Parquet로 PostgreSQL 데이터 유형 매핑 (p. 301)

Parquet로 MySQL 데이터 유형 매핑

다음 표는 데이터를 변환해 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 형식			
BIGINT	INT64		
BIGINT UNSIGNED	FIXED_LEN_BYTE_ARRAY	DECIMAL(20,0)	Parquet는 서명된 유형만 지원하므로 매핑에는 BIGINT_UNSIGNED 유형을 저장하기 위해 추가 바이트(8 + 1)가 필요합니다.
BIT	BYTE_ARRAY		
DECIMAL	INT32	DECIMAL (p,s)	소스 값이 2^{31} 보다 작으면 INT32로 저장됩니다.
	INT64	DECIMAL (p,s)	소스 값이 2^{31} 이상이지만 2^{63} 미만인 경우 INT64로 저장됩니다.
	FIXED_LEN_BYTE_ARRAY	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_ARRAY 유형의 문자열로 변환되고 UTF8로 인코딩됩니다.
DOUBLE	DOUBLE		
FLOAT	DOUBLE		
INT	INT32		
INT UNSIGNED	INT64		
MEDIUMINT	INT32		
MEDIUMINT UNSIGNED	INT64		
NUMERIC	INT32	DECIMAL (p,s)	소스 값이 2^{31} 미만이면 INT32로 저장됩니다.

소스 데이터 유형	Parquet 프리미티브 유형	논리적 유형 주석	변환 노트
	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		
TINYINT	INT32		
TINYINT UNSIGNED	INT32		
문자열 데이터 형식			
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 형식			
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 형식			
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 데 이터 값으로 인해 복잡해

PostgreSQL 데이터 형식	Parquet 프리미티브 유형	논리적 유형 주석	매핑 노트
			지는 것을 피하기 위한 것입니다.
DOUBLE PRECISION	DOUBLE		
INTEGER	INT32		
MONEY	BYTE_ARRAY	STRING	
REAL	FLOAT		
SERIAL	INT32		
SMALLINT	INT32	INT_16	
SMALLSERIAL	INT32	INT_16	
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			
DATE	BYTE_ARRAY	STRING	
INTERVAL	BYTE_ARRAY	STRING	
TIME	BYTE_ARRAY	STRING	

PostgreSQL 데이터 형식	Parquet 프리미티브 유형	논리적 유형 주석	매핑 노트
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		
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	

DB 클러스터를 지정된 시간으로 복원

DB 클러스터를 특정 시점으로 복원하여 새로운 DB 클러스터를 생성할 수 있습니다. DB 클러스터를 특정 시점으로 복원할 경우, 새 DB 클러스터에는 기본 DB 보안 그룹이 적용됩니다. 사용자 지정 DB 보안 그룹을 DB 클러스터에 적용해야 하는 경우에는 DB 인스턴스를 사용할 수 있게 된 후 AWS Management 콘솔, AWS CLI `modify-db-cluster` 명령 또는 Amazon RDS API `ModifyDBCluster` 작업을 사용하여 이들을 명시적으로 적용해야 합니다.

Note

Aurora DB 클러스터 백업 및 복원에 대한 자세한 내용은 [Aurora DB 클러스터 백업 및 복원에 대한 개요 \(p. 268\)](#) 단원을 참조하십시오. Aurora MySQL의 경우 프로비저닝된 DB 클러스터를 Aurora Serverless DB 클러스터에 복원할 수 있습니다. 자세한 내용은 [Aurora Serverless DB 클러스터 복원 \(p. 125\)](#) 단원을 참조하십시오.

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터를 특정 시점으로 복원할 수 있습니다.

콘솔

DB 클러스터를 지정된 시간으로 복원하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 복원하려는 DB 클러스터의 를 선택합니다.
4. 작업에서 특정 시점으로 복구를 선택합니다.

DB 인스턴스 시작 창이 나타납니다.

5. 최근 복원 가능 시간을 선택하여 가능한 최근 시간으로 복원하거나, 사용자 지정을 선택하여 시간을 선택합니다.

사용자 지정을 선택한 경우 클러스터를 복원하려는 날짜와 시간을 입력합니다.

6. DB 인스턴스 식별자에 복원된 DB 인스턴스의 이름을 입력한 후, 다른 옵션을 설정합니다.
7. [Launch DB Instance]를 선택합니다.

AWS CLI

DB 클러스터를 특정 시점으로 복원하려면, AWS CLI 명령 `restore-db-cluster-to-point-in-time`을 사용하여 새 DB 클러스터를 만듭니다.

Example

Linux, OS X, Unix의 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
```

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
```

RDS API

DB 클러스터를 특정 시간으로 복원하려면, Amazon RDS API `RestoreDBClusterToPointInTime` 작업을 다음 파라미터와 함께 호출합니다.

- `SourceDBClusterIdentifier`
- `DBClusterIdentifier`
- `RestoreToTime`

스냅샷 삭제

Amazon RDS에서 관리하는 DB 클러스터 스냅샷이 더 이상 필요하지 않으면 삭제할 수 있습니다.

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 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 스냅샷을 선택합니다.
3. 삭제하고 싶은 DB 클러스터 스냅샷을 선택합니다.
4. 작업에서 스냅샷 삭제를 선택합니다.
5. 확인 페이지에서 삭제를 선택합니다.

AWS CLI

AWS CLI 명령 `delete-db-cluster-snapshot`을 사용하여 DB 클러스터 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.

Linux, OS X, Unix의 경우:

```
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 `DeleteDBClusterSnapshot`을 사용하여 DB 클러스터 스냅샷을 삭제할 수 있습니다.

다음 파라미터를 사용하여 DB 클러스터 스냅샷을 삭제할 수 있습니다.

- `DBClusterSnapshotIdentifier` – DB 클러스터 스냅샷의 식별자입니다.

Amazon Aurora DB 클러스터 유지 관리

Amazon RDS는 Amazon RDS 리소스를 정기적으로 유지 관리합니다. 이러한 유지 관리에는 DB 클러스터의 기본 하드웨어, 기본 운영 체제(OS) 또는 데이터베이스 엔진 버전에 대한 업데이트가 수반되는 경우가 많습니다. 운영 체제 업데이트는 보안상 가장 빈번하게 발생하며 최대한 빠른 시간 내에 실행해야 합니다.

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 패치 이 포함됩니다. 이때 보안 및 인스턴스 안정성과 관련된 패치에 대해 필수 패치 작업으로 자동 예약됩니다. 이러한 패치 작업은 찾지는 않아서(수 개월에 한 번 정도) 유지 관리 기간에 비하면 매우 적은 편입니다.

RDS 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 DB 클러스터에 대해 유지 관리 업데이트를 사용할 수 있는지 여부를 확인할 수 있습니다. 업데이트가 있는 경우에는 다음과 같이 Amazon RDS 콘솔에서 DB 클러스터의 유지 관리 열에 사용 가능 여부가 표시됩니다.

Current activity	Maintenance	VPC	Multi-AZ
0 Connections	none	vpc-2aed394c	No
0 Connections	next window	vpc-2aed394c	No
0.02 Sessions	none	vpc-2aed394c	No

DB 클러스터에 대해 유지 관리 업데이트가 제공되지 않는 경우 그에 대한 열 값은 없음입니다.

DB 클러스터에 대해 유지 관리 업데이트가 제공되지 않는 경우 다음과 같은 열 값이 가능합니다.

- 필수 – 유지 관리 작업은 리소스에 적용되며 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.
- 작업 없음

Note

일부 OS 업데이트가 필수로 표시됩니다. 필수 업데이트를 연기할 경우에는 Amazon RDS에서 언제 업데이트가 수행될지를 알려 주는 알림이 수신됩니다. 기타 업데이트는 사용 가능으로 표시되며 이러한 업데이트는 무한정으로 연기할 수 있습니다.

조치를 취하려면 DB 클러스터를 선택하여 세부 정보를 표시한 후 Maintenance & backups(유지 관리 및 백업)을 선택하십시오. 그러면 보류 중인 유지 관리 항목이 표시됩니다.

The screenshot shows the AWS RDS console with the 'Maintenance & backups' tab selected. In the 'Pending maintenance' section, there is one item listed:

Description	Type	Status	Apply date
Automatic minor version upgrade to postgres 9.6.11	db-upgrade	next window	February 25th 2019, 3:28:00 am UTC-8 (local)

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 \(p. 309\)](#) 단원을 참조하십시오.

Amazon Aurora 엔진으로의 업데이트에 대한 자세한 내용과 엔진을 업그레이드하고 패치를 적용하는 방법을 보려면 [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#) 및 [Amazon Aurora PostgreSQL 데이터베이스 엔진 업데이트 \(p. 905\)](#) 단원을 참조하십시오.

DB 클러스터의 업데이트 적용

Amazon RDS를 사용하여 유지 관리 작업을 적용하는 시기를 선택할 수 있습니다. RDS 콘솔, AWS Command Line Interface(AWS CLI) 또는 RDS API를 사용해 Amazon RDS가 업데이트를 적용하는 시기를 결정할 수 있습니다.

콘솔

DB 클러스터의 업데이트를 관리하려면

1. AWS Management 콘솔에 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 업그레이드
 - Upgrade at next window(다음에 업그레이드)

Note

다음에 업그레이드를 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

Linux, OS X, Unix의 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
```

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 --opt-in-type에 undo-opt-in을 지정합니다. 유지 관리 작업이 이미 시작된 경우 --opt-in-type에 undo-opt-in을 지정할 수 없습니다. 유지 관리 작업을 취소하려면 [modify-db-instance](#) AWS CLI 명령을 실행하고 --no-auto-minor-version-upgrade를 지정합니다.

하나 이상의 대기 중인 업데이트가 있는 리소스 목록을 반환하려면, [describe-pending-maintenance-actions](#) AWS CLI 명령을 사용합니다.

Example

Linux, OS X, Unix의 경우:

```
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

Linux, OS X, Unix의 경우:

```
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

모든 DB 클러스터에는 시스템 변경 내용이 적용되는 주 단위 유지 관리 기간이 있습니다. 유지 관리 기간은 요청이나 필요에 따라 수정하거나 소프트웨어 패치를 적용하는 시기를 조정할 수 있는 기간입니다. 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.

지역별로 8시간 블록 시간 중에서 30분 유지 관리 시간이 임의로 선택됩니다. DB 클러스터 생성 시 기본 유지 관리 기간을 지정하지 않으면 Amazon RDS에서 임의로 선택한 요일에 30분 유지 관리 기간을 배정합니다.

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 완료하면서 다중 AZ 장애 조치가 필요한 경우가 있을 수 있습니다.

다음에서 기본 유지 관리 기간이 할당된 리전별 시간 블록을 확인할 수 있습니다.

리전 이름	Region	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

리전 이름	Region	시간 블록
미국 서부(오레곤)	us-west-2	06:00–14:00 UTC
아시아 태평양(홍콩)	ap-east-1	06:00–14:00 UTC
아시아 태평양(뭄바이)	ap-south-1	17:30–01:30 UTC
아시아 태평양(오사카-로컬)	ap-northeast-3	22:00–23:59 UTC
아시아 태평양(서울)	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
캐나다(중부)	ca-central-1	03:00–11:00 UTC
중국(베이징)	cn-north-1	06:00–14:00 UTC
중국(닝샤)	cn-northwest-1	06:00–14:00 UTC
유럽(프랑크푸르트)	eu-central-1	23:00–07:00 UTC
유럽(아일랜드)	eu-west-1	22:00–06:00 UTC
유럽(런던)	eu-west-2	22:00–06:00 UTC
유럽(파리)	eu-west-3	23:59–07:29 UTC
유럽(스톡홀름)	eu-north-1	23:00–07:00 UTC
중동(바레인)	me-south-1	06:00–14:00 UTC
남아메리카(상파울루)	sa-east-1	00:00–08:00 UTC
AWS GovCloud (US-West)	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 콘솔에 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. 작업에서 클러스터 수정을 선택합니다.

5. 유지 관리 섹션에서 유지 관리 기간을 업데이트합니다.

6. [Continue]를 선택합니다.

확인 페이지에서 변경 내용을 검토합니다.

7. 유지 관리 기간에 변경 사항을 즉시 적용하려면 즉시 적용을 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.

Linux, OS X, Unix의 경우:

```
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 = my-cluster`
- `PreferredMaintenanceWindow = Tue:04:00-Tue:04:30`

Example

다음은 유지 관리 기간을 화요일 4:00–4:30 AM UTC로 설정하는 코드 예제입니다.

```
https://rds.us-west-2.amazonaws.com/
?Action=ModifyDBCluster
&DBClusterIdentifier=my-cluster
&PreferredMaintenanceWindow=Tue:04:00-Tue:04:30
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140725/us-east-1/rds/aws4_request
&X-Amz-Date=20161017T161457Z
```

&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=d6d1c65c2e94f5800ab411a3f7336625820b103713b6c063430900514e21d784

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 우선 순위에 따라 다릅니다. 업그레이드를 자주 수행하지 않아도 되는 Aurora MySQL 장기 안정성(LTS) 릴리스에 대한 자세한 내용은 [Aurora MySQL LTS\(장기 지원\) 릴리스 \(p. 689\)](#) 단원을 참조하십시오.

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 MySQL Amazon RDS MySQL을 모두 사용하며 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 업그레이드 할 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 Amazon RDS 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 Aurora MySQL DB 클러스터를 LTS 버전보다 최신인 Aurora MySQL 버전으로 업그레이드하여 Aurora가 중요한 업그레이드를 더 자주 적용하도록 활성화할 수 있습니다. 이렇게 하면 최신 성능 향상, 버그 수정 및 기능을 보다 신속하게 사용할 수 있습니다.

DB 클러스터에서 DB 인스턴스 재부팅

일반적으로 유지 관리를 이유로 DB 인스턴스를 재부팅해야 할 수 있습니다. 예를 들어 특정 내용을 수정하거나 DB 인스턴스 또는 해당 DB 클러스터와 연결된 DB 파라미터 그룹을 변경하는 경우 변경 내용을 적용하려면 인스턴스를 재부팅해야 합니다.

Note

DB 인스턴스에서 연결된 DB 파라미터 그룹에 대한 최신 변경 내용을 사용하고 있지 않은 경우 AWS Management 콘솔에 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.

니다. 파라미터 그룹에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

DB 인스턴스를 재부팅하면 데이터베이스 엔진 서비스가 재시작됩니다. DB 인스턴스를 재부팅하면 DB 인스턴스 상태가 rebooting으로 설정되면서 잠시 종단됩니다.

사용 가능 상태가 아닌 경우 DB 인스턴스를 재부팅할 수 없습니다. 백업이 진행 중이거나 이전에 수정을 요청했거나 유지 관리 기간 작업 등 여러 원인으로 인해 데이터베이스를 사용할 수 없습니다.

Important

Amazon Aurora DB 클러스터의 기본 인스턴스를 재부팅하면 RDS에서 해당 DB 클러스터의 모든 Aurora 복제본도 자동으로 다시 시작됩니다. Aurora DB 클러스터의 기본 인스턴스를 재부팅하면 장애 조치가 발생하지 않습니다. Aurora 복제본을 재부팅하면 장애 조치가 발생하지 않습니다. Aurora DB 클러스터에 대한 장애 조치를 수행하려면 AWS CLI 명령 [failover-db-cluster](#) 또는 API 작업 [FailoverDBCluster](#)를 호출하십시오.

콘솔

DB 인스턴스를 재부팅하려면

1. AWS Management 콘솔에 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 [Cancel]을 선택합니다.

AWS CLI

AWS CLI를 사용하여 DB 인스턴스를 재부팅하려면 [reboot-db-instance](#) 명령을 호출하십시오.

Example 간편한 재부팅

Linux, OS X, Unix의 경우:

```
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 호출하십시오.

AuroraDB 클러스터에서 DB 인스턴스 삭제

DB 클러스터 또는 Amazon Aurora 복제본의 기본 DB 인스턴스를 포함하여 DB 클러스터에서 DB 인스턴스를 삭제할 수 있습니다. DB 인스턴스를 삭제하려면 인스턴스의 이름을 지정해야 합니다.

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 정보는 [여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제 \(p. 557\)](#) 단원을 참조하십시오.

삭제 방지

사용자가 DB 클러스터를 삭제할 수 없도록 삭제 방지를 활성화할 수 있습니다. AWS Management 콘솔을 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. 삭제 방지 활성화 및 비활성화에 대한 자세한 내용은 [콘솔, CLI, API를 사용하여 DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.

Aurora는 콘솔, CLI, API 중 어느 곳에서 작업을 수행하든 간에 DB 클러스터에 대한 삭제 방지를 강제 시행합니다. 삭제 방어가 활성화된 DB 클러스터를 삭제하려 해도 삭제를 할 수가 없습니다. 클러스터를 확실히 삭제할 수 있으려면 클러스터를 수정하고 삭제 방지를 비활성화해야 합니다.

단일 DB 인스턴스가 있는 Aurora 클러스터

Aurora 클러스터의 마지막 DB 인스턴스를 삭제하려 하는 경우 이 작동의 결과는 어떤 방법을 사용하느냐에 따라 달라집니다. AWS Management 콘솔을 통해 마지막 DB 인스턴스를 삭제할 수 있지만, 이렇게 하면 DB 클러스터도 삭제됩니다. DB 클러스터가 삭제 방지를 활성화한 경우에도 AWS CLI 또는 API를 통해 마지막 DB 인스턴스를 삭제할 수 있습니다. 이 경우 DB 클러스터 자체는 여전히 존재하고 데이터는 보존됩니다. 새로운 DB 인스턴스를 클러스터에 연결하여 데이터에 다시 액세스할 수 있습니다.

콘솔, CLI, API를 사용하여 DB 인스턴스 삭제

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 인스턴스를 삭제할 수 있습니다.

콘솔

DB 인스턴스를 삭제하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 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 `--db-instance-identifier` 옵션과 함께 `delete-db-instance` 명령을 호출하십시오.

Example

Linux, OS X, Unix의 경우:

```
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.

Amazon RDS 리소스에 태그 지정

Amazon RDS 태그를 사용하여 Amazon RDS 리소스에 메타데이터를 추가할 수 있습니다. IAM 정책과 함께 이러한 태그를 사용하여 Amazon RDS 리소스에 대한 액세스를 관리하고 Amazon RDS 리소스에 적용 가능한 작업을 제어할 수 있습니다. 마지막으로 비슷하게 태그가 지정된 리소스에 대한 비용을 그룹화하여 이러한 태그로 비용을 추적할 수 있습니다.

모든 Amazon RDS 리소스에 태깅할 수 있습니다.

- DB 인스턴스
- DB 클러스터
- 읽기 전용 복제본
- DB 스냅샷
- DB 클러스터 스냅샷
- 예약 DB 인스턴스
- 이벤트 구독
- DB 옵션 그룹
- DB 파라미터 그룹
- DB 클러스터 파라미터 그룹
- DB 보안 그룹
- DB 서브넷 그룹

IAM 정책으로 태그가 지정된 리소스 액세스 관리에 대한 자세한 내용은 [Amazon Aurora의 Identity and Access Management\(IAM\) \(p. 960\)](#) 단원을 참조하십시오.

Amazon RDS 리소스 태그 개요

Amazon RDS 태그는 사용자가 정의하고 Amazon RDS 리소스와 연결하는 이름-값 페어입니다. 이 이름을 키라고 합니다. 키 값을 제공하는 것은 선택 사항입니다. 태그를 사용하여 Amazon RDS 리소스에 임의의 정보를 배정할 수 있습니다. 범주 정의 등에 태그 키를 사용할 수 있으며 태그 값은 해당 범주의 항목일 수 있습니다. 예를 들어, 태그 키를 “project”로 정의하고 태그 값을 “Salix”로 정의하여 Amazon RDS 리소스가 Salix project에 배정됨을 나타냅니다. 또한 태그를 사용하여 `environment=test` 또는 `environment=production` 등의 키를 사용해 Amazon RDS 리소스를 테스트나 프로덕션에 사용되도록 지정할 수도 있습니다. Amazon RDS 리소스와 연결된 메타데이터를 더 쉽게 추적할 수 있게 일관성 있는 태그 키 세트를 사용하는 것이 좋습니다.

태그를 사용하여 비용 구조를 반영하도록 AWS 대금을 구성하십시오. 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 결제 및 비용 관리 정보의 [비용 할당 및 태그 지정](#)을 참조하십시오.

각 Amazon RDS 리소스에는 해당 Amazon RDS 리소스에 배정되는 모든 태그를 포함하는 태그 세트가 있습니다. 태그 세트는 최대 50개의 태그를 포함하거나 비어 있을 수 있습니다. Amazon RDS 리소스의 기존 태그와 동일한 키를 갖는 태그를 리소스에 추가하면 새 값이 이전 값을 덮어씁니다.

AWS에서는 태그에 의미론적 의미를 적용하지 않습니다. 태그는 엄격히 문자열로 해석됩니다. Amazon RDS에서는 사용자가 리소스를 만들 때 사용하는 설정에 따라 DB 인스턴스 또는 기타 Amazon RDS 리소스에서 태그를 설정할 수 있습니다. 예를 들어 Amazon RDS에서 DB 인스턴스가 프로덕션용인지, 아니면 테스트용인지를 나타내는 태그를 추가할 수 있습니다.

- 태그 키는 태그의 필수 이름입니다. 문자열 값은 길이가 1~128자(유니코드 문자)이며 "aws:" 또는 "rds:"로 시작할 수 없습니다. 문자열에는 유니코드 문자 집합, 숫자, 공백, '_', '.', ':', '/', '=', '+', '-', '@'(Java regex: "^([\p{L}][\p{Z}]\p{N}_:=+\-\-]*\$")만 포함할 수 있습니다.
- 태그 값은 태그의 선택적 문자열 값입니다. 문자열 값은 길이가 1~256자(유니코드 문자)이며 "aws:"로 시작할 수 없습니다. 문자열에는 유니코드 문자 집합, 숫자, 공백, '_', '.', ':', '/', '=', '+', '-', '@'(Java regex: "^([\p{L}][\p{Z}]\p{N}_:=+\-\-]*\$")만 포함할 수 있습니다.

값은 태그 세트에서 고유할 필요는 없으며 null일 수 있습니다. 예를 들어 `project=Trinity` 및 `cost-center=Trinity`의 태그 세트에 키-값 페어가 있을 수 있습니다.

Note

스냅샷에 태그를 추가할 수 있지만 이 그룹화는 청구서에 반영되지 않습니다.

AWS Management 콘솔, 명령줄 인터페이스 또는 Amazon RDS API를 사용하여 Amazon RDS 리소스에서 태그를 추가, 나열 및 삭제할 수 있습니다. 명령줄 인터페이스나 Amazon RDS API를 사용할 때는 작업하려는 Amazon RDS 리소스에 대한 Amazon 리소스 이름(ARN)을 제공해야 합니다. ARN 생성에 대한 자세한 내용은 [Amazon RDS의 ARN 구성 \(p. 319\)](#) 주제단원을 참조하십시오.

권한 부여 목적으로 태그가 캐시됩니다. 이 때문에 Amazon RDS 리소스의 태그에 대한 추가나 업데이트가 제공되는 데 몇 분 정도 걸릴 수 있습니다.

콘솔

Amazon RDS 리소스에 태그를 지정하는 프로세스는 모든 리소스에서 비슷합니다. 다음 절차에서는 Amazon RDS DB 인스턴스에 태그를 지정하는 방법을 보여줍니다.

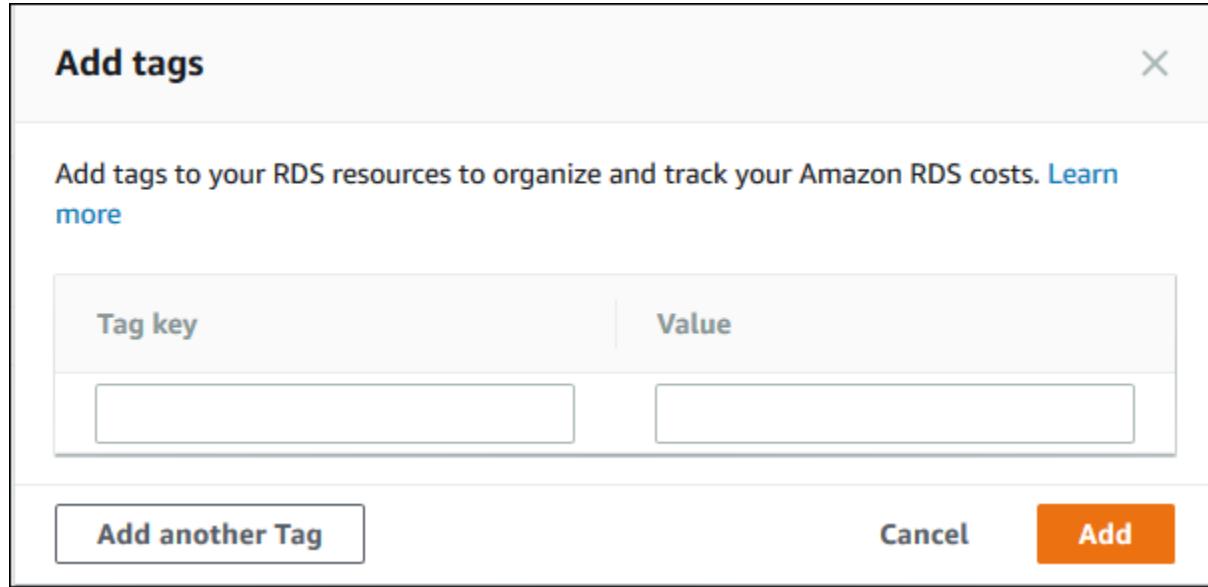
DB 인스턴스에 태그를 추가하려면

1. AWS Management 콘솔에 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. 추가를 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. 추가를 선택합니다.

DB 인스턴스에서 태그를 삭제하려면

1. AWS Management 콘솔에 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.

Tags (1)		Edit	Remove	Add
<input type="text"/>	Filter tag key	<	1	>
<input checked="" type="checkbox"/> Tag key	Value			
<input checked="" type="checkbox"/> workload-type	other			

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 \(p. 319\)](#) 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 [ListTagsForResource](#)를 사용합니다.
- Amazon RDS 리소스에서 태그를 제거하려면 [RemoveTagsFromResource](#) 작업을 사용합니다.

필수 ARN을 생성하는 방법에 대해 자세히 알아보려면 [Amazon RDS의 ARN 구성 \(p. 319\)](#) 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. Key 및 Value 값은 대/소문자를 구분합니다. 예를 들어, project=Trinity와 PROJECT=Trinity는 서로 다른 두 개의 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.
키	키는 태그의 필수 이름입니다. 문자열 값은 길이가 1~128자(유니코드 문자)이며 "rds:" 또는 "aws:"로 시작할 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/Trinity 와 project/Xanadu처럼 키는 같지만 값은 다른 키-페어가 있을 수 없습니다.

태그 지정 요소	설명
값	<p>값은 태그의 선택적 값입니다. 문자열 값은 길이가 1~256자(유니코드 문자)이며 "rds:" 또는 "aws:"로 시작할 수 없습니다. 문자열에는 유니코드 문자, 숫자, 공백, '_', '.', '/', '=', '+', '-'(Java regex: "^([\p{L}]\p{Z} \p{N}...:=+\-\])\$")만 포함될 수 있습니다.</p> <p>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.</p>

Amazon RDS의 Amazon 리소스 이름(ARN)을 사용한 작업

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.

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	리전	Endpoint	Protocol
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 (캘리포니 아 북부)	us-west-1	rds.us-west-1.amazonaws.com	HTTPS
미국 서부 (오레곤)	us-west-2	rds.us-west-2.amazonaws.com	HTTPS
Africa (Cape Town)	af-south-1	rds.af-south-1.amazonaws.com	HTTPS
아시아 태 평양(홍콩)	ap-east-1	rds.ap-east-1.amazonaws.com	HTTPS
아시아 태 평양(뭄바 이)	ap-south-1	rds.ap-south-1.amazonaws.com	HTTPS
아시아 태 평양(오사 카-로컬)	ap-northeast-3	rds.ap-northeast-3.amazonaws.com	HTTPS

Amazon Aurora Aurora 사용 설명서 ARN 생성

리전 이름	리전	Endpoint	Protocol	
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	
중국(베이징)	cn-north-1	rds.cn-north-1.amazonaws.com.cn	HTTPS	
중국(닝샤)	cn-northwest-1	rds.cn-northwest-1.amazonaws.com.cn	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(파리)	eu-west-3	rds.eu-west-3.amazonaws.com	HTTPS	
유럽(스톡홀름)	eu-north-1	rds.eu-north-1.amazonaws.com	HTTPS	
중동(바레인)	me-south-1	rds.me-south-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(US)	us-gov-west-1	rds.us-gov-west-1.amazonaws.com	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:

리소스 유형	ARN 형식
	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> 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

리소스 유형	ARN 형식
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
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> 예: arn:aws:rds:us-east-2:123456789012:subgrp:my-subnet-10

기존 ARN 가져오기

AWS Management 콘솔, AWS Command Line Interface(AWS CLI) 또는 RDS API를 사용해 RDS 리소스에 대한 ARN을 가져올 수 있습니다.

콘솔

AWS Management 콘솔에서 ARN을 확인하려면 ARN을 보려는 리소스를 탐색하거나, 해당 리소스의 세부 정보를 확인합니다. 예를 들어, 다음과 같이 DB 인스턴스 세부 정보의 구성 탭에서 DB 인스턴스의 ARN을 가져올 수 있습니다.

Connectivity Monitoring Logs & events Configuration

Instance

Configuration

DB instance id
oracle-instance1

Engine version
12.1.0.2.v14

Storage type
General Purpose (SSD)

IOPS
-

Storage
20 GiB

DB name
ORCL

License model
Bring Your Own License

Character set
AL32UTF8

Option groups
default:oracle-ee-12-1

ARN
arn:aws:rds:us-west-2:██████████:db:oracle-instance1

Resource id

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
<code>describe-event-subscriptions</code>	EventSubscriptionArn
<code>describe-certificates</code>	CertificateArn
<code>describe-db-parameter-groups</code>	DBParameterGroupArn
<code>describe-db-cluster-parameter-groups</code>	DBClusterParameterGroupArn
<code>describe-db-instances</code>	DBInstanceArn
<code>describe-db-security-groups</code>	DBSecurityGroupArn
<code>describe-db-snapshots</code>	DBSnapshotArn
<code>describe-events</code>	SourceArn
<code>describe-reserved-db-instances</code>	ReservedDBInstanceArn
<code>describe-db-subnet-groups</code>	DBSubnetGroupArn
<code>describe-db-clusters</code>	DBClusterArn
<code>describe-db-cluster-snapshots</code>	DBClusterSnapshotArn

예를 들어, 다음 AWS CLI 명령은 DB 인스턴스에 대한 ARN을 가져옵니다.

Example

Linux, OS X, Unix의 경우:

```
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. [AWS Management 콘솔](#)에서 유지 관리 기간 설정을 확인하거나 변경할 수 있습니다.

아래에서 Amazon Aurora에 대한 일반적인 업데이트에 대한 정보를 확인할 수 있습니다. Amazon Aurora에 적용되는 일부 업데이트는 Aurora에서 지원하는 데이터베이스 엔진으로 국한됩니다. Aurora의 데이터베이스 엔진 업데이트에 대한 자세한 내용은 다음 표를 참조하십시오.

데이터베이스 엔진	업데이트
Amazon Aurora MySQL	Amazon Aurora MySQL 데이터베이스 엔진 업데이트 (p. 686) 단원을 참조하십시오.
Amazon Aurora PostgreSQL	Amazon Aurora PostgreSQL 데이터베이스 엔진 업데이트 (p. 905) 단원을 참조하십시오.

해당 Amazon Aurora 버전 식별

Amazon Aurora에는 모든 Aurora DB 클러스터에서 사용할 수 있는 Aurora 일반 기능인 특정 기능이 포함되어 있습니다. 그 밖에도 Aurora에는 Aurora가 지원하는 일부 데이터베이스 엔진에서만 사용할 수 있는 기

능들도 포함되어 있습니다. 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 번호에는 다음 형식을 사용합니다.

```
<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	<pre>SELECT AURORA_VERSION();</pre> <pre>SHOW @@aurora_version;</pre>
Amazon Aurora PostgreSQL	<pre>SELECT AURORA_VERSION();</pre>

Amazon Aurora DB 클러스터 모니터링

이 단원에서는 Amazon Aurora를 모니터링하는 방법을 보여줍니다. Aurora는 복제 토플로지에 연결된 데이터베이스 서버의 클러스터를 사용합니다. Aurora 클러스터를 모니터링하려면 보통 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 사용자는 하나의 DB 인스턴스에서 이루어진 변경 사항이 다른 인스턴스에 적용되기까지의 시간인 복제 지연 시간을 측정하여 클러스터의 전반적인 상태를 모니터링할 수 있습니다.

주제

- [Amazon RDS 모니터링 개요 \(p. 327\)](#)
- [Amazon Aurora DB 클러스터 보기 \(p. 338\)](#)
- [DB 클러스터 상태 \(p. 343\)](#)
- [DB 인스턴스 상태 \(p. 344\)](#)
- [Amazon Aurora DB 클러스터 지표 모니터링 \(p. 346\)](#)
- [확장 모니터링 \(p. 358\)](#)
- [Amazon RDS 성능 개선 도우미 사용 \(p. 365\)](#)
- [Amazon Aurora 권장 사항 사용 \(p. 408\)](#)
- [Aurora PostgreSQL에서 데이터베이스 활동 스트림 사용 \(p. 412\)](#)
- [Amazon RDS 이벤트 알림 서비스 사용 \(p. 429\)](#)
- [Amazon RDS 이벤트 보기 \(p. 444\)](#)
- [Amazon Aurora에 대한 CloudWatch 이벤트 및 Amazon EventBridge 이벤트 가져오기 \(p. 445\)](#)
- [Amazon Aurora 데이터베이스 로그 파일 \(p. 449\)](#)
- [AWS CloudTrail을 사용하여 Amazon RDS API 호출 로깅 \(p. 459\)](#)

Amazon RDS 모니터링 개요

모니터링은 Amazon RDS와 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 발생하는 다중 지점 실패를 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분으로부터 모니터링 데이터를 수집해야 합니다. Amazon RDS 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 작성하는 것이 좋습니다.

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

다음 단계에서는 다양한 시간과 다양한 부하 조건에서 성능을 측정하여 환경에서 일반 Amazon RDS 성능의 기준선을 설정합니다. Amazon RDS를 모니터링할 때 과거 모니터링 데이터를 저장할 것을 고려해야 합니다. 이 저장된 데이터는 현재 성능 데이터와 비교하고, 일반 성능 패턴과 성능 이상을 식별하고, 문제 해결 방법을 제안하는 기준이 됩니다.

예를 들어 Amazon RDS에서 네트워크 처리량, 읽기, 쓰기 및/또는 메타데이터 작업에 대한 I/O, 클라이언트 연결, DB 인스턴스에 대한 버스트 크레딧 밸런스 등을 모니터링할 수 있습니다. 성능이 설정된 기준을 벗어

날 경우 워크로드에 대한 데이터베이스 가용성을 최적화하기 위해 클라이언트에 사용 가능한 DB 인스턴스 및 읽기 전용 복제본의 수 또는 DB 인스턴스의 인스턴스 클래스를 변경해야 할 수 있습니다.

일반적으로 성능 지표에 허용되는 값은 기준이 무엇인지 그리고 애플리케이션 무엇을 수행하는지에 따라 달립니다. 기준과의 일관된 차이 또는 추세를 조사하십시오. 특정 지표 유형에 대한 참고 정보는 다음과 같습니다.

- CPU 또는 RAM 사용량이 많음 – CPU 또는 RAM 사용량이 많을 경우 해당 애플리케이션의 목표와 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 – 인스턴스 성능 저하 및 응답 시간 지연과 함께 사용자 연결 수가 많을 경우 데이터베이스 연결 제한을 고려해 봅니다. DB 인스턴스에 대한 최적의 사용자 연결 수는 해당 인스턴스 클래스와, 수행하는 작업의 복잡성에 따라 달립니다. DB 인스턴스를 `User Connections` 파라미터가 0(무제한)이 아닌 다른 값으로 설정된 파라미터 그룹과 연결하여 데이터베이스 연결 수를 지정할 수 있습니다. 기존 파라미터 그룹을 사용하거나 새로 하나 만들 수 있습니다. 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.
- 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.

모니터링 도구

AWS는 Amazon RDS를 모니터링하는 데 사용할 수 있는 다양한 도구를 제공합니다. 이를 도구 중에는 모니터링을 자동으로 수행하도록 구성할 수 있는 도구도 있지만, 수동 작업이 필요한 도구도 있습니다. 모니터링 작업을 최대한 자동화하는 것이 좋습니다.

자동 모니터링 도구

다음과 같은 자동 모니터링 도구를 사용하여 Amazon RDS를 관찰하고 문제 발생 시 보고할 수 있습니다.

- Amazon RDS 이벤트 – DB 인스턴스, DB 클러스터, DB 클러스터 스냅샷, DB 파라미터 그룹 또는 DB 보안 그룹에 변경 사항이 있을 경우 알려주는 Amazon RDS 이벤트에 가입합니다. 자세한 내용은 [Amazon RDS 이벤트 알림 서비스 사용 \(p. 429\)](#) 단원을 참조하십시오.
- 데이터베이스 로그 파일 – Amazon RDS 콘솔 또는 Amazon RDS API 작업을 사용해 데이터베이스 로그 파일을 보거나, 다운로드하거나, 모니터링합니다. 또한 데이터베이스 테이블에 로드된 데이터베이스로 그 파일 중 일부에 쿼리를 요청할 수도 있습니다. 자세한 내용은 [Amazon Aurora 데이터베이스 로그 파일 \(p. 449\)](#) 단원을 참조하십시오.
- Amazon RDS 확장된 모니터링 — 운영 체제에 대한 지표를 실시간으로 확인합니다. 자세한 내용은 [확장 모니터링 \(p. 358\)](#) 단원을 참조하십시오.

또한 Amazon RDS는 추가 모니터링 기능을 위해 Amazon CloudWatch와 통합됩니다.

- Amazon CloudWatch 지표 – Amazon RDS는 각각의 활성 데이터베이스 인스턴스에 대해 1분마다 CloudWatch로 지표를 전송합니다. CloudWatch에서 Amazon RDS 지표에 대해서는 추가 요금이 청구되지 않습니다. 자세한 내용은 [the section called “DB 인스턴스 측정치 보기” \(p. 336\)](#) 단원을 참조하십시오.
- Amazon CloudWatch 경보 – 특정 기간 동안 단일 Amazon RDS 지표를 감시하고, 설정한 임계값을 기준으로 측정치의 값에 따라 하나 이상의 작업을 수행할 수 있습니다. 자세한 내용은 [Amazon CloudWatch로 모니터링 \(p. 329\)](#) 단원을 참조하십시오.

- Amazon CloudWatch Logs – 대부분의 DB 엔진에서 CloudWatch Logs의 데이터베이스 로그 파일을 모니터링 및 저장하고 액세스할 수 있습니다. 자세한 내용은 [Amazon CloudWatch Logs User Guide](#) 단원을 참조하십시오.

수동 모니터링 도구

Amazon RDS 모니터링의 또 한 가지 중요한 부분은 CloudWatch 경보에 포함되지 않는 항목을 수동으로 모니터링해야 한다는 점입니다. Amazon RDS, CloudWatch, AWS Trusted Advisor 및 다른 AWS 콘솔 대시보드에서 AWS 환경의 상태를 한눈에 파악할 수 있습니다. 또한 DB instance에서 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 스토리지의 양
 - DB 인스턴스에 대해 사용 중인 메모리 및 CPU 양
 - DB 인스턴스에서 주고 받는 네트워크 트래픽 양
- AWS Trusted Advisor 대시보드에서는 다음과 같은 비용 최적화, 보안, 내결함성과 성능 개선 확인을 살펴볼 수 있습니다.
 - Amazon RDS 유후 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 헤더 페이지에 표시되는 항목은 다음과 같습니다.
 - 현재 경보 및 상태
 - 경보 및 리소스 그래프
 - 서비스 상태

또한 CloudWatch를 사용하여 다음 작업을 수행할 수도 있습니다.

- [사용자 정의 대시보드](#)를 만들어 원하는 서비스 모니터링.
- 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악.
- 모든 AWS 리소스 지표 검색 및 찾아보기.
- 문제에 대해 알려주는 경보 생성 및 편집.

Amazon CloudWatch로 모니터링

Amazon RDS로부터 원시 데이터를 수집하여 읽기 가능한 실시간 지표로 처리하는 Amazon CloudWatch를 사용하여 DB 인스턴스를 모니터링할 수 있습니다. Amazon RDS 지표 데이터는 기본적으로 1분 단위로 CloudWatch에 자동 전송됩니다. CloudWatch에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 [Amazon CloudWatch란 무엇입니까?](#)를 참조하십시오.

15일 동안 60초(1분)의 데이터 요소를 사용할 수 있으므로 기록 정보에 액세스하고 웹 애플리케이션이나 서비스가 어떻게 수행되고 있는지 더 잘 파악할 수 있습니다. CloudWatch 지표 보존 기간에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 [지표 보존 기간](#)을 참조하십시오.

Note

Amazon RDS Performance Insights를 사용하는 경우 추가 지표를 사용할 수 있습니다. 자세한 내용은 [Amazon CloudWatch에 게시되는 성능 개선 도우미 지표 \(p. 398\)](#) 단원을 참조하십시오.

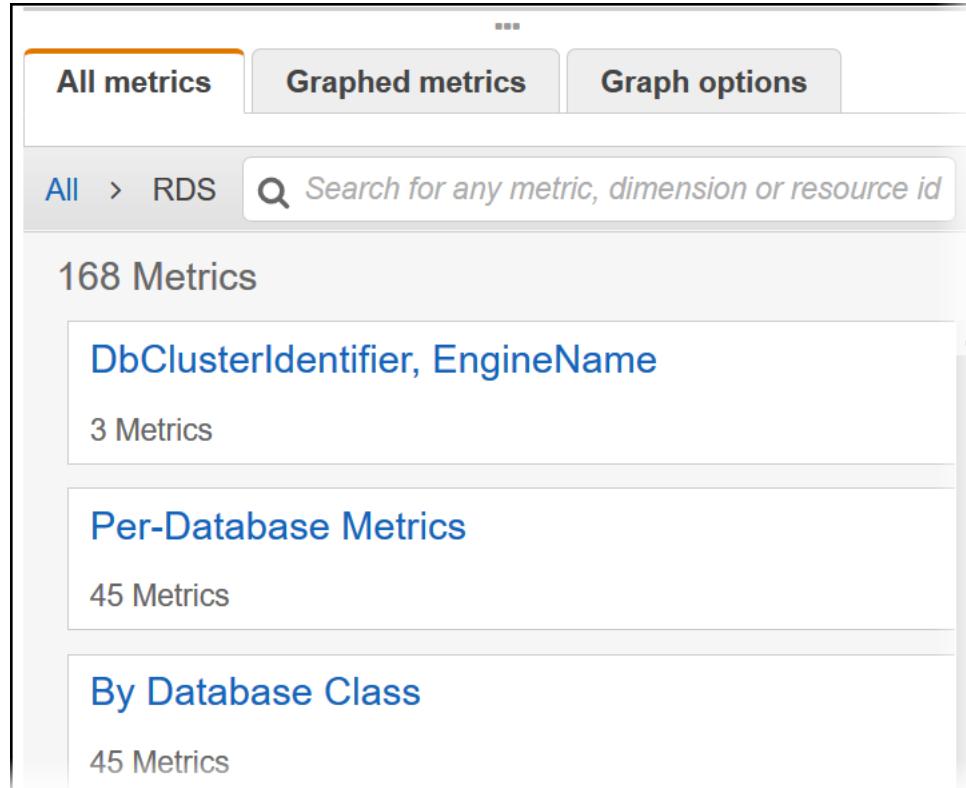
Amazon RDS 지표 및 차원

Amazon RDS 리소스를 사용할 때 Amazon RDS에서 1분마다 지표 및 차원을 Amazon CloudWatch에 보냅니다. 다음 절차에 따라 Amazon RDS에 대한 지표를 볼 수 있습니다.

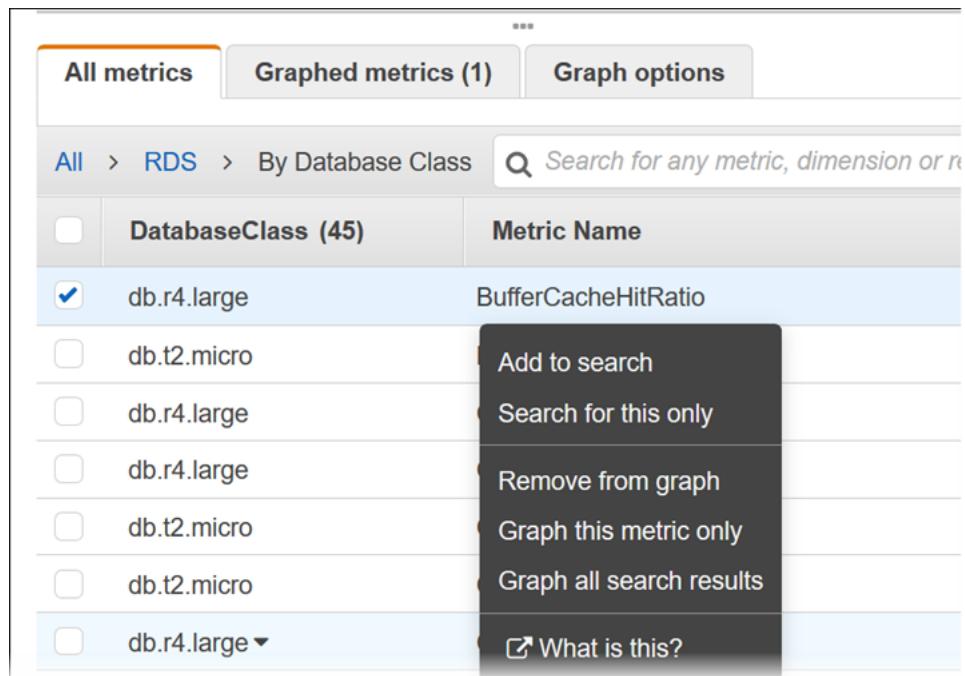
Amazon CloudWatch 콘솔을 사용한 메트릭 확인

측정치는 먼저 서비스 네임스페이스별로 그룹화된 다음, 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

1. <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
2. 필요한 경우 AWS 리전을 변경합니다. 탐색 모음에서 AWS 리소스가 상주하는 AWS 리전을 선택합니다. 자세한 내용은 [리전 및 앤드포인트](#)를 참조하십시오.
3. 탐색 창에서 [Metrics]를 선택합니다. RDS 측정치 네임스페이스를 선택합니다.



4. 측정치 차원(예: 데이터베이스 클래스별)을 선택합니다.
5. 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.



AWS CLI를 사용하여 지표를 보려면

- 명령 프롬프트에서 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace AWS/RDS
```

Amazon RDS 지표

AWS/RDS 네임스페이스에는 다음 지표가 포함되어 있습니다.

지표	설명
BinLogDiskUsage	마스터에서 이진 로그가 차지하는 디스크 공간 크기. MySQL 읽기 전용 복제본에 적용. 단위: 바이트
BurstBalance	사용할 수 있는 범용 SSD(gp2) 버스트-버킷 I/O 크레딧 비율 단위: 백분율
CPUUtilization	CPU 사용 백분율. 단위: 백분율
CPUCreditUsage	(T2 인스턴스) CPU 사용률을 위해 인스턴스에서 소비되는 CPU 크레딧의 수입니다. 하나의 CPU 크레딧은 1분 또는 이와 동등한 vCPU, 사용률 및 시간의 조합 동안 100%의 사용률로 실행되는 vCPU 하나에 해당됩니다. 예를 들어, CPU 크레딧 하나는 2분 동안 50%의 사용률로 실행되는 vCPU 하나 또는 2분 동안 25%의 사용률로 실행되는 vCPU 2개에 해당합니다.

지표	설명
	<p>CPU 크레딧 측정치는 5분 간격으로만 제공됩니다. 5분 이상의 시간을 지정할 경우 Sum 통계 대신 Average 통계를 사용하십시오.</p> <p>단위: 크레딧 (vCPU-분)</p>
CPUCreditBalance	<p>(T2 인스턴스) 시작 이후 인스턴스가 누적한 획득 CPU 크레딧 수입니다. T2 스탠다드의 경우 CPUCreditBalance에 누적된 시작 크레딧 수도 포함됩니다.</p> <p>크레딧은 획득 이후에 크레딧 밸런스에 누적되고, 소비 시 크레딧 밸런스에서 소멸됩니다. 크레딧 밸런스는 최대 한도(인스턴스 크기에 따라 결정)가 있습니다. 한도에 도달하면 새로 획득한 크레딧이 모두 삭제됩니다. T2 스탠다드의 경우 시작 크레딧은 한도에 포함되지 않습니다.</p> <p>CPUCreditBalance의 크레딧은 인스턴스가 기준 CPU 사용률 이상으로 버스터를 하는 데 소비할 수 있습니다.</p> <p>인스턴스가 실행 중인 동안 CPUCreditBalance의 크레딧은 만료되지 않습니다. 인스턴스가 종지되면 CPUCreditBalance는 지속되지 않고 모든 누적된 크레딧이 삭제됩니다.</p> <p>CPU 크레딧 측정치는 5분 간격으로만 제공됩니다.</p> <p>단위: 크레딧 (vCPU-분)</p>
DatabaseConnections	<p>사용 중인 데이터베이스 연결 수.</p> <p>지표 값에는 아직 데이터베이스에서 정리되지 않은 끊어진 데이터베이스 연결이 포함되지 않을 수 있습니다. 따라서 데이터베이스에서 기록한 데이터베이스 연결 수가 지표 값보다 클 수 있습니다.</p> <p>단위: 개수</p>
DiskQueueDepth	<p>디스크 액세스를 대기 중인 I/O(읽기/쓰기 요청) 수.</p> <p>단위: 개수</p>
FailedSQLServerAgentJobs	<p>최근 1분간 실패한 SQL Server 에이전트 작업의 수입니다.</p> <p>단위: 수/분</p>
FreeableMemory	<p>사용 가능한 RAM 크기.</p> <p>Aurora의 경우 이 지표는 /proc/meminfo의 MemAvailable 필드 값을 보고합니다.</p> <p>단위: 바이트</p>
FreeStorageSpace	<p>사용 가능한 스토리지 공간 크기.</p> <p>단위: 바이트</p>
MaximumUsedTransactionIDs	<p>사용된 최대 전역 트랜잭션 ID. PostgreSQL에 적용됩니다.</p> <p>단위: 개수</p>

지표	설명
<code>NetworkReceiveThroughput</code>	DB 인스턴스 수신 네트워크 트래픽(고객 데이터베이스 트래픽과 모니터링 및 복제에 사용된 Amazon RDS 트래픽을 모두 포함). 단위: 바이트/초
<code>NetworkTransmitThroughput</code>	DB 인스턴스 송신 네트워크 트래픽(고객 데이터베이스 트래픽과 모니터링 및 복제에 사용된 Amazon RDS 트래픽을 모두 포함). 단위: 바이트/초
<code>OldestReplicationSlotLag</code>	수신된 WAL 데이터를 기준으로 가장 지연된 복제본의 지연 크기. PostgreSQL에 적용됩니다. 단위: 메가바이트
<code>ReadIOPS</code>	초당 평균 디스크 읽기 I/O 연산 수 단위: 개수/초
<code>ReadLatency</code>	디스크 I/O 연산당 평균 처리 시간. 단위: 초
<code>ReadThroughput</code>	초당 디스크에서 읽은 평균 바이트 수. 단위: 바이트/초
<code>ReplicaLag</code>	원본 DB 인스턴스를 기준으로 읽기 전용 복제본 DB 인스턴스의 지연 시간. MySQL, MariaDB, Oracle, PostgreSQL 및 SQL Server 읽기 전용 복제본에 적용됩니다. 단위: 초
<code>ReplicationSlotDiskUsage</code>	복제 슬롯 파일에 사용된 디스크 공간. PostgreSQL에 적용됩니다. 단위: 메가바이트
<code>SwapUsage</code>	DB 인스턴스에서 사용된 스왑 공간 크기. SQL 서버에는 이 지표를 사용 할 수 없습니다. 단위: 바이트
<code>TransactionLogsDiskUsage</code>	트랜잭션 로그에 사용된 디스크 공간. PostgreSQL에 적용됩니다. 단위: 메가바이트
<code>TransactionLogsGeneration</code>	초당 생성되는 트랜잭션 로그의 크기. PostgreSQL에 적용됩니다. 단위: 바이트/초
<code>WriteIOPS</code>	초당 평균 디스크 쓰기 I/O 연산 수 단위: 개수/초
<code>WriteLatency</code>	디스크 I/O 연산당 평균 처리 시간. 단위: 초

지표	설명
WriteThroughput	초당 디스크에 쓴 평균 바이트 수. 단위: 바이트/초

Amazon RDS 차원

다음 표의 차원을 사용하여 Amazon RDS 지표 데이터를 필터링할 수 있습니다.

차원	설명
DBInstanceIdentifier	이 차원은 특정 데이터베이스 인스턴스에 대해 요청하는 데이터를 필터링합니다.
DBClusterIdentifier	이 차원은 특정 Amazon Aurora DB 클러스터에 대해 요청하는 데이터를 필터링합니다.
DBClusterIdentifier, Role	이 차원은 인스턴스 역할(WRITER/READER)별로 지표를 집계하여 특정 Aurora DB 클러스터에 대해 요청하는 데이터를 필터링합니다. 예를 들어 클러스터에 속하는 모든 READER 인스턴스에 대한 지표를 집계할 수 있습니다.
DatabaseClass	이 차원은 특정 데이터베이스 클래스의 모든 인스턴스에 대해 요청하는 데이터를 필터링합니다. 예를 들어 데이터베이스 클래스 db.m1.small에 속하는 모든 인스턴스에 대한 지표를 집계할 수 있습니다.
EngineName	이 차원은 식별된 엔진 이름에 대해 요청하는 데이터만 필터링합니다. 예를 들어 엔진 이름이 mysql인 모든 인스턴스에 대한 지표를 집계할 수 있습니다.
SourceRegion	이 차원은 지정된 리전에 대해 요청한 데이터를 필터링합니다. 예를 들어 us-east-1 리전의 모든 인스턴스에 대한 지표를 집계할 수 있습니다.

Amazon RDS를 모니터링하는 CloudWatch 경보 생성

경보가 상태를 변경하면 Amazon SNS 메시지를 보내는 CloudWatch 경보를 만들 수 있습니다. 경보는 지정한 기간에 단일 지표를 감시하고 여러 기간에 지정된 임계값에 대한 지표 값을 기준으로 작업을 하나 이상 수행합니다. 이 작업은 Amazon SNS 주제나 Auto Scaling 정책으로 전송되는 알림입니다.

경보는 지속적인 상태 변경에 대해서만 작업을 호출합니다. CloudWatch 경보는 특정 상태가 되었다고 해서 작업을 호출하지는 않습니다. 이러한 상태가 변경되어 지정한 기간 동안 유지되어야 합니다. 다음 절차에서는 Amazon RDS에 대한 경보를 만드는 방법을 보여 줍니다.

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 복제본에 Amazon Aurora Auto Scaling 사용 \(p. 253\)](#) 단원을 참조하십시오.

CloudWatch 콘솔을 사용한 경보 설정

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
2. [Alarms]를 선택한 다음 [Create Alarm]을 선택합니다. 그러면 [Create Alarm Wizard]가 시작됩니다.
3. RDS 지표를 선택하고 Amazon RDS 지표를 스크롤하여 경보를 생성할 지표를 찾습니다. 이 대화 상자에서 Amazon RDS 지표를 표시하려면 리소스의 식별자를 검색합니다. 지표를 선택하여 경보를 생성한 다음 다음을 선택합니다.
4. 지표에 대한 이름, 설명 및 다음 경우 항상 값을 입력합니다.
5. 경보 상태에 도달하면 CloudWatch에서 이메일을 보내도록 하려면 이 경보가 발생할 경우 항상:에서 상태가 ALARM입니다.를 선택합니다. Send a notification to:(다음 주소로 알림 전송)에서 기존 SNS 주제를 선택합니다. Create topic(주제 생성)을 선택한 경우 새 이메일 구독 목록에 대한 이름 및 이메일 주소를 설정할 수 있습니다. 이 목록은 향후 경보를 위해 필드에 저장되고 표시됩니다.

Note

새 Amazon SNS 주제를 생성하기 위해 주제 생성을 사용할 경우 이메일 주소는 알림을 받기 전에 검증되어야 합니다. 이메일은 경보가 경보 상태에 입력될 때만 전송됩니다. 이러한 경보 상태 변경이 이메일이 검증되기 전에 발생할 경우에는 알림을 받지 못합니다.

6. 이제 경보 미리 보기 영역에서 생성할 경보를 미리 볼 수 있습니다. 경보 생성을 선택합니다.

AWS CLI를 사용하여 경보를 설정하려면

- [put-metric-alarm](#)을 호출합니다. 자세한 내용은 [AWS CLI Command Reference](#) 단원을 참조하십시오.

CloudWatch API를 사용하여 경보를 설정하려면

- [PutMetricAlarm](#)을 호출합니다. 자세한 내용은 [Amazon CloudWatch API Reference](#) 단원을 참조하십시오.

Amazon CloudWatch Logs에 데이터베이스 엔진 로그 게시

Amazon CloudWatch Logs의 로그 그룹에 로그 데이터를 게시하도록 Amazon RDS 데이터베이스 엔진을 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 CloudWatch Logs 에이전트로 관리할 수 있는 내구성이 뛰어난 스토리지에 로그 레코드를 저장할 수 있습니다. 예를 들어 로그 레코드를 호스트에서 로그 서비스로 교체할 시기를 결정할 수 있으므로 필요할 때 원시 로그에 액세스할 수 있습니다.

엔진별 정보는 다음을 참조하십시오.

- the section called “[CloudWatch Logs에 Aurora MySQL 로그 게시](#)” (p. 639)
- the section called “[CloudWatch Logs에 Aurora PostgreSQL 로그 게시](#)” (p. 847)

Note

로그 데이터 게시를 활성화하려면 먼저 AWS Identity and Access Management(IAM)에 서비스 연결 역할이 있어야 합니다. 서비스 연결 역할에 대한 자세한 내용은 [Amazon Aurora에 서비스 연결 역할 사용 \(p. 1001\)](#) 단원을 참조하십시오.

CloudWatch 로그 통합 구성

CloudWatch Logs에 데이터베이스 로그 파일을 게시하려면 게시할 로그를 선택합니다. 새 DB 인스턴스를 만들 때 [Advanced Settings] 섹션에서 이를 선택합니다. 기존 DB 인스턴스를 수정하여 게시를 시작해도 됩니다.

The screenshot shows the 'Log Exports' configuration page. It includes sections for enabling CloudWatch Logs exports and selecting log types. A 'Log Exports Role' section is also visible, with 'RDS Service Linked Role' highlighted.

Log Exports

Enable CloudWatch Logs exports

Error Log
 General Log
 Slow Query Log
 Audit Log

Log Exports Role

RDS Service Linked Role

게시를 활성화하면 Amazon RDS는 모든 DB 인스턴스 로그 레코드를 로그 그룹으로 계속 스트리밍합니다. 예를 들어 게시하는 각 로그 유형에 대해 `/aws/rds/instance/log_type` 로그 그룹이 있습니다. 이 로그 그룹은 로그를 생성하는 데이터베이스 인스턴스와 동일한 AWS 리전에 있습니다.

로그 레코드를 게시한 후 CloudWatch Logs를 사용하여 레코드를 검색 및 필터링할 수 있습니다. 로그 검색 및 필터링에 관한 자세한 내용은 [로그 데이터 검색 및 필터링](#)을 참조하십시오.

DB 인스턴스 측정치 보기

Amazon RDS는 DB 인스턴스의 상태를 모니터링할 수 있도록 지표를 제공합니다. DB 인스턴스 측정치와 운영 체제(OS) 측정치를 모두 모니터링할 수 있습니다.

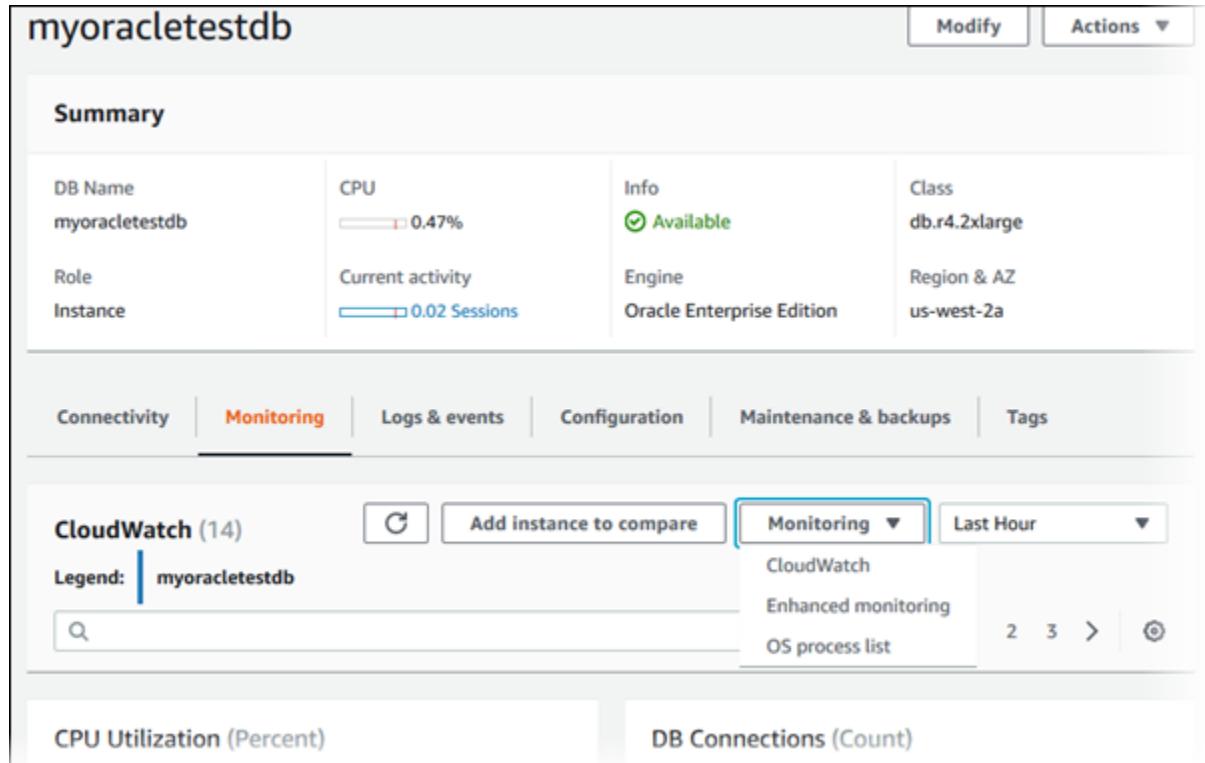
이 단원에서는 RDS 콘솔 및 CloudWatch를 사용하여 DB 인스턴스에 대한 지표를 확인하는 방법을 자세히 설명합니다. CloudWatch Logs를 사용하여 DB 인스턴스의 운영 체제에 대한 지표를 모니터링하는 방법에 대한 자세한 내용은 [확장 모니터링 \(p. 358\)](#) 단원을 참조하십시오.

콘솔을 사용하여 측정치 보기

DB 인스턴스에 대한 DB 및 OS 측정치를 보려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 정보가 필요한 DB 인스턴스의 이름을 선택하여 세부 정보를 표시합니다.
4. [Monitoring] 탭을 선택합니다.
5. 모니터링에서 측정치를 표시할 방법을 지정하는 옵션을 선택합니다.
 - CloudWatch – Amazon CloudWatch에서 사용 가능한 DB 인스턴스 측정치를 요약하여 표시합니다. 각 지표에는 특정 시간대에서 지표를 모니터링한 그래프도 포함되어 있습니다.

- 확장 모니터링 – 확장 모니터링을 활성화한 상태로 DB 인스턴스에 대해 사용 가능한 OS 지표를 요약하여 표시합니다. 각 지표에는 특정 시간대에서 지표를 모니터링한 그래프도 포함되어 있습니다.
- OS 프로세스 목록 – 선택한 인스턴스에서 실행되는 각 프로세스의 세부 정보를 표시합니다.



Tip

시간 범위 목록을 사용하여 그래프로 표시된 측정치의 시간 범위를 선택할 수 있습니다. 그래프를 선택하여 더 세부적인 보기 를 불러올 수 있습니다. 측정치별 필터를 데이터에 적용할 수도 있습니다.

CLI 또는 API를 사용하여 DB 인스턴스 지표 보기

Amazon RDS는 CloudWatch 측정치와 통합되어 다양한 DB 인스턴스 측정치를 제공합니다. CloudWatch 지표는 RDS 콘솔, AWS CLI 또는 API를 사용하여 볼 수 있습니다.

전체 Amazon RDS 지표 목록은 Amazon CloudWatch 사용 설명서의 [Amazon RDS 차원 및 지표](#)에서 확인할 수 있습니다.

CloudWatch CLI를 사용하여 DB 측정치 보기

Note

다음 CLI 예제를 실행하려면 CloudWatch 명령줄 도구가 필요합니다. CloudWatch에 대한 자세한 내용을 보거나 개발자 도구를 다운로드하려면 [Amazon CloudWatch 제품 페이지](#)를 참조하십시오. 이 예제에 나온 StartTime 및EndTime 값은 설명을 듬기 위해 지정되었습니다. 따라서 DB 인스턴스의 올바른 시작 및 종료 시간 값으로 대체해야 합니다.

DB 인스턴스의 사용량 및 성능 통계를 보는 방법

- CloudWatch 명령 mon-get-stats를 다음 파라미터와 함께 사용합니다.

```
PROMPT>mon-get-stats FreeStorageSpace --dimensions="DBInstanceIdentifier=mydbinstance"  
--statistics= Average  
--namespace="AWS/RDS" --start-time 2009-10-16T00:00:00 --end-time 2009-10-16T00:02:00
```

CloudWatch API를 사용하여 DB 측정치 보기

이 예제에 나온 StartTime 및 EndTime 값은 설명을 돋기 위해 지정되었습니다. 따라서 DB 인스턴스의 올 바른 시작 및 종료 시간 값으로 대체해야 합니다.

DB 인스턴스의 사용량 및 성능 통계를 보는 방법

- 다음 파라미터와 함께 CloudWatch API GetMetricStatistics를 호출합니다.
 - Statistics.member.1 = Average
 - Namespace = AWS/RDS
 - StartTime = 2009-10-16T00:00:00
 - EndTime = 2009-10-16T00:02:00
 - Period = 60
 - MeasureName = FreeStorageSpace

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 RDS에서는 콘솔의 탐색 창에 있는 Databases(데이터베이스)를 선택하면 DB 클러스터에 관한 세부 정보를 볼 수 있습니다. 또한 Databases(데이터베이스) 페이지에서 Amazon Aurora DB 클러스터에 속한 DB 인스턴스의 세부 정보를 볼 수 있습니다.

Databases(데이터베이스) 목록에는 해당 AWS 계정의 모든 DB 클러스터가 표시됩니다. DB 클러스터를 선택하면 DB 클러스터에 대한 정보와 함께 해당 DB 클러스터에 속해 있는 DB 인스턴스의 목록이 표시됩니다. 목록에서 DB 인스턴스의 식별자를 선택하여 RDS 콘솔에서 해당 DB 인스턴스에 대한 정보 페이지로 바로 이동할 수 있습니다.

DB 클러스터의 세부 정보 페이지를 보려면 탐색 창에서 Databases(데이터베이스)를 선택한 후 DB 클러스터의 이름을 선택합니다.

콘솔의 탐색 창에서 Databases(데이터베이스)를 선택하여 Databases(데이터베이스) 목록으로 이동해 DB 클러스터를 수정할 수 있습니다. DB 클러스터를 수정하려면 Databases(데이터베이스) 목록에서 DB 클러스터를 선택한 후 수정을 선택합니다.

DB 클러스터에 속한 DB 인스턴스를 수정하려면 콘솔의 탐색 창에서 Databases(데이터베이스)를 선택하여 Databases(데이터베이스) 목록으로 이동합니다.

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.

The screenshot shows the AWS RDS console interface for managing an Aurora DB cluster named `aurora-test`. The main table displays the following information:

DB identifier	Role	Engine	Region & AZ
<code>aurora-test</code>	Regional	Aurora MySQL	us-east-1
<code>dbinstance4</code>	Writer	Aurora MySQL	us-east-1a
<code>dbinstance1</code>	Reader	Aurora MySQL	us-east-1b
<code>dbinstance2</code>	Reader	Aurora MySQL	us-east-1b
<code>dbinstance3</code>	Reader	Aurora MySQL	us-east-1a

Below the table, there are several tabs: Connectivity & security (highlighted in orange), Monitoring, Logs & events, Configuration, Maintenance & backups, and Tags.

In the 'Endpoints' section, there are two entries:

- `aurora-test.cluster-ro-... .us-east-1.rds.amazonaws.com`
- `aurora-test.cluster-... .us-east-1.rds.amazonaws.com`

`dbinstance4` DB 인스턴스 식별자의 링크를 클릭하면 Amazon RDS 콘솔에 다음 이미지와 같이 `dbinstance4` DB 인스턴스에 대한 세부 정보 페이지가 표시됩니다.

Related			
<input type="text"/> Filter databases			
DB identifier	Role	Engine	
aurora-test	Regional	Aurora MySQL	
dbinstance4	Writer	Aurora MySQL	
dbinstance1	Reader	Aurora MySQL	
dbinstance2	Reader	Aurora MySQL	
dbinstance3	Reader	Aurora MySQL	

Connectivity & security

Monitoring Logs & events Configuration Maintenance Tags

Endpoint & port

Endpoint
dbinstance4. [REDACTED].us-east-1.rds.amazonaws.com

Port
3306

Network
Available
us-east-1
VPC
vpc-

AWS CLI

AWS CLI를 사용하여 DB 클러스터 정보를 보려면 [describe-db-clusters](#) 명령을 사용하십시오. 예를 들어 다음 AWS CLI 명령은 구성된 AWS 계정에 대해 us-east-1 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",  
      "Endpoint": "sample-cluster1.cluster-123456789012.us-east-1.rds.amazonaws.com",  
      "AllocatedStorage": 1,  
      "DBClusterIdentifier": "sample-cluster1",  
      "MasterUsername": "mymasteruser",  
      "EarliestRestorableTime": "2016-03-30T03:35:42.563Z".  
    }  
  ]  
}
```

```
"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": "5.6.10a",
"DBClusterParameterGroup": "default.aurora5.6",
"BackupRetentionPeriod": 1,
"AvailabilityZones": [
    "us-east-1b",
    "us-east-1c",
    "us-east-1d"
],
"LatestRestorableTime": "2016-03-31T20:06:08.903Z",
"PreferredMaintenanceWindow": "wed:08:15-wed:08:45"
},
{
    "Status": "available",
    "Engine": "aurora",
    "Endpoint": "aurora-sample.cluster-123456789012.us-east-1.rds.amazonaws.com",
    "AllocatedStorage": 1,
    "DBClusterIdentifier": "aurora-sample-cluster",
    "MasterUsername": "mymasteruser",
    "EarliestRestorableTime": "2016-03-30T10:21:34.826Z",
    "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": "5.6.10a",
```

```
        "DBClusterParameterGroup": "default.aurora5.6",
        "BackupRetentionPeriod": 1,
        "AvailabilityZones": [
            "us-east-1b",
            "us-east-1c",
            "us-east-1d"
        ],
        "LatestRestorableTime": "2016-03-31T20:00:11.491Z",
        "PreferredMaintenanceWindow": "sun:03:53-sun:04:23"
    }
]
```

RDS API

Amazon RDS API를 사용하여 DB 클러스터 정보를 보려면 [DescribeDBClusters](#) 작업을 사용하십시오. 예를 들어 다음 Amazon RDS API 명령은 us-east-1 리전에 있는 모든 DB 클러스터의 정보를 열거합니다.

```
https://rds.us-east-1.amazonaws.com/
?Action=DescribeDBClusters
&MaxRecords=100
&SignatureMethod=HmacSHA256
&SignatureVersion=4
&Version=2014-10-31
&X-Amz-Algorithm=AWS4-HMAC-SHA256
&X-Amz-Credential=AKIADQKE4SARGYLE/20140722/us-east-1/rds/aws4_request
&X-Amz-Date=20140722T200807Z
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date
&X-Amz-Signature=2d4f2b9e8abc31122b5546f94c0499bba47de813cb875f9b9c78e8e19c9afe1b
```

이 작업은 다음 출력을 반환합니다.

```
<DescribeDBClustersResponse xmlns="http://rds.amazonaws.com/doc/2014-10-31/">
<DescribeDBClustersResult>
    <DBClusters>
        <DBCluster>
            <Engine>aurora5.6</Engine>
            <Status>available</Status>
            <BackupRetentionPeriod>0</BackupRetentionPeriod>
            <DBSubnetGroup>my-subgroup</DBSubnetGroup>
            <EngineVersion>5.6.10a</EngineVersion>
            <Endpoint>sample-cluster2.cluster-cbfvmb0y5fy.us-east-1.rds.amazonaws.com</
Endpoint>
            <DBClusterIdentifier>sample-cluster2</DBClusterIdentifier>
            <PreferredBackupWindow>04:45-05:15</PreferredBackupWindow>
            <PreferredMaintenanceWindow>sat:05:56-sat:06:26</PreferredMaintenanceWindow>
            <DBClusterMembers/>
            <AllocatedStorage>15</AllocatedStorage>
            <MasterUsername>awsuser</MasterUsername>
        </DBCluster>
        <DBCluster>
            <Engine>aurora5.6</Engine>
            <Status>available</Status>
            <BackupRetentionPeriod>0</BackupRetentionPeriod>
            <DBSubnetGroup>my-subgroup</DBSubnetGroup>
            <EngineVersion>5.6.10a</EngineVersion>
            <Endpoint>sample-cluster3.cluster-cefgqfx9y5fy.us-east-1.rds.amazonaws.com</
Endpoint>
            <DBClusterIdentifier>sample-cluster3</DBClusterIdentifier>
            <PreferredBackupWindow>07:06-07:36</PreferredBackupWindow>
            <PreferredMaintenanceWindow>tue:10:18-tue:10:48</PreferredMaintenanceWindow>
            <DBClusterMembers>
```

```
<DBClusterMember>
  <IsClusterWriter>true</IsClusterWriter>
  <DBInstanceIdentifier>sample-cluster3-master</DBInstanceIdentifier>
</DBClusterMember>
<DBClusterMember>
  <IsClusterWriter>false</IsClusterWriter>
  <DBInstanceIdentifier>sample-cluster3-read1</DBInstanceIdentifier>
</DBClusterMember>
</DBClusterMembers>
<AllocatedStorage>15</AllocatedStorage>
<MasterUsername>awsuser</MasterUsername>
</DBCluster>
</DBClusters>
</DescribeDBClustersResult>
<ResponseMetadata>
  <RequestId>d682b02c-1383-11b4-a6bb-172dfac7f170</RequestId>
</ResponseMetadata>
</DescribeDBClustersResponse>
```

DB 클러스터 상태

DB 클러스터 상태는 DB 클러스터의 상태를 나타냅니다. Amazon RDS 콘솔, AWS CLI 명령 [describe-db-clusters](#) 또는 API 작업 [DescribeDBClusters](#)을 사용하여 DB 클러스터의 상태를 볼 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 정보는 [DB 클러스터의 업데이트 적용 \(p. 307\)](#) 단원을 참조하십시오.

다음 표에서 DB 클러스터에 대한 가능한 상태 값을 확인하십시오.

DB 클러스터 상태	설명
사용 가능	DB 클러스터에 문제가 없으며 사용할 수 있습니다.
백업 중	현재 DB 클러스터를 백업 중입니다.
역추적	현재 DB 클러스터를 백트랙 중입니다. 이 상태는 Aurora MySQL에만 적용됩니다.
cloning-failed	DB 클러스터 복제에 실패했습니다.
생성 중	DB 클러스터를 생성 중입니다. 생성 중인 DB 클러스터에는 액세스할 수 없습니다.
삭제 중	DB 클러스터를 삭제 중입니다.
failing-over	기본 인스턴스에서 Aurora Replica에 대한 장애 조치가 수행 중입니다.
inaccessible-encryption-credentials	DB 클러스터를 암호화 또는 해독하는 데 사용되는 AWS KMS 키는 액세스할 수 없습니다.
유지 관리	Amazon RDS는 DB 클러스터에 유지 관리 업데이트를 적용합니다. 이 상태는 RDS가 한참 전에 미리 예약하는 DB 클러스터 수준의 유지 관리에 사용됩니다.
migrating	DB 클러스터 스냅샷이 DB 클러스터로 복원 중입니다.

DB 클러스터 상태	설명
migration-failed	マイグ레이션이 실패했습니다.
수정 중	고객의 DB 클러스터 수정 요청으로 인해 DB 클러스터를 수정 중입니다.
promoting	읽기 전용 복제본이 독립형 DB 클러스터로 승격 중입니다.
이름을 바꾸는 중	고객의 이름 바꾸기 요청으로 인해 DB 클러스터 이름을 바꾸는 중입니다.
마스터 자격 증명 재설정 중	고객의 재설정 요청으로 인해 DB 클러스터 사용자 자격 증명을 재설정하는 중입니다.
시작 중	DB 클러스터를 시작 중입니다.
중단됨	DB 클러스터가 중단됩니다.
중지 중	DB 클러스터를 중지 중입니다.
update-iam-db-auth	DB 클러스터에 대한 IAM 권한 부여가 업데이트 중입니다.
upgrading	DB 클러스터 엔진 버전은 현재 업그레이드 중입니다.

DB 인스턴스 상태

DB 인스턴스의 상태는 DB 인스턴스의 상태를 나타냅니다. Amazon RDS 콘솔, AWS CLI 명령 [describe-db-instances](#) 또는 [DescribeDBInstances](#) API 작업을 사용하여 DB 인스턴스 상태를 확인할 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 클러스터의 업데이트 적용 \(p. 307\)](#) 단원을 참조하십시오.

다음 표에서 DB 인스턴스에 가능한 상태 값을 찾으십시오. 각 상태별 요금도 표시됩니다. DB 인스턴스와 스토리지에만 요금이 청구되는지, 스토리지에만 청구되는지, 혹은 청구되지 않는지 표시됩니다. 모든 DB 인스턴스 상태의 백업 사용에는 항상 청구됩니다.

DB 인스턴스 상태	청구	설명
사용 가능	청구	DB 인스턴스에 문제가 없으며 사용할 수 있습니다.
백업 중	청구	현재 DB 인스턴스를 백업 중입니다.
역추적	청구	현재 DB 인스턴스를 역추적 중입니다. 이 상태는 Aurora MySQL에만 적용됩니다.
Enhanced Monitoring 구 성 중	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	DB 인스턴스를 생성 중입니다. 생성 중인 DB 인스턴스에는 액세스할 수 없습니다.
deleting	미청구	DB 인스턴스를 삭제 중입니다.
실패	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 해독하는 데 사용되는 AWS KMS 키는 액세스할 수 없습니다.
네트워크 호환 장애	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.
옵션 그룹 호환 장애	청구	Amazon RDS 옵션 그룹 변경을 적용하려고 시도했으나 적용하지 못했습니다. 또한 Amazon RDS는 이전 옵션 그룹 상태를 둘백하지 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.
파라미터 호환 장애	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.
복원 호환 장애	미청구	Amazon RDS가 특정 시점으로 복원을 수행할 수 없습니다. 임시 테이블 사용, MySQL에서 MyISAM 테이블 사용 또는 MariaDB에서 Aria 테이블 사용은 이러한 상태가 발생하는 일반적 원인 중 하나입니다.
유지 관리	청구	Amazon RDS는 DB 인스턴스에 유지 관리 업데이트를 적용합니다. 이 상태는 RDS가 한참 전에 미리 예약하는 인스턴스 수준의 유지 관리에 사용됩니다.
수정 중	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.
재부팅 중	청구	고객의 요청 또는 DB 인스턴스 재부팅이 필요한 Amazon RDS 프로세스로 인해 DB 인스턴스를 재부팅 중입니다.
이름을 바꾸는 중	청구	고객의 이름 바꾸기 요청으로 인해 DB 인스턴스 이름을 바꾸는 중입니다.
마스터 자격 증명 재설정 중	청구	고객의 재설정 요청으로 인해 DB 인스턴스 사용자 자격 증명을 재설정하는 중입니다.

DB 인스턴스 상태	청구	설명
restore-error	청구	특정 시점으로 복원을 시도하거나 스냅샷에서 복원을 시도할 때 DB 인스턴스에서 오류가 발생했습니다.
시작 중	요금 부과 스토리지	DB 인스턴스를 시작하는 중입니다.
중단됨	요금 부과 스토리지	DB 인스턴스가 종지되었습니다.
종지 중	요금 부과 스토리지	DB 인스턴스를 종지 중입니다.
스토리지가 꽉 참	청구	DB 인스턴스에 할당된 스토리지 용량이 거의 다 찼습니다. 이것은 중요한 상태이므로 즉각 이 문제를 수정하는 것이 좋습니다. 그러면 DB 인스턴스를 수정하여 스토리지를 확장하십시오. 이러한 상황을 피하려면 스토리지 공간이 부족해지면 경고하도록 Amazon CloudWatch 경보를 설정하십시오.
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.
upgrading	청구	데이터베이스 엔진 버전은 현재 업그레이드 중입니다.

Amazon Aurora DB 클러스터 지표 모니터링

Amazon Aurora은 모니터링을 통해 Aurora DB 클러스터의 상태와 성능을 평가할 수 있는 Amazon CloudWatch 지표를 다양하게 제공합니다. Amazon RDS 관리 콘솔, AWS CLI CloudWatch API와 같은 다양한 도구를 사용하여 Aurora 지표를 볼 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 모니터링 \(p. 327\)](#) 단원을 참조하십시오.

Note

Amazon RDS Performance Insights를 사용하는 경우 추가 지표를 사용할 수 있습니다. 자세한 내용은 [Amazon CloudWatch에 게시되는 성능 개선 도우미 지표 \(p. 398\)](#) 단원을 참조하십시오.

Amazon Aurora 지표

Amazon Aurora에서 사용할 수 있는 지표는 아래와 같습니다.

Amazon Aurora 지표

AWS/RDS 네임스페이스에는 Amazon Aurora에서 실행 중인 데이터베이스 개체에 적용되는 다음 지표가 포함되어 있습니다.

지표	설명	적용 대상
<code>ActiveTransactions</code>	<p>현재 Aurora 데이터베이스 인스턴스에서 1초마다 실행되고 있는 평균 트랜잭션 수입니다.</p> <p>기본적으로 Aurora에서는 이 지표를 활성화하지 않습니다. 이 값의 측정을 시작하려면 특정 DB 인스턴스의 DB 파라미터 그룹에서 <code>innodb_monitor_enable='all'</code>을 설정하십시오.</p>	Aurora MySQL
<code>AuroraBinlogReplicaLag</code>	<p>소스 DB 클러스터를 기준으로 MySQL과 호환되는 Aurora에서 실행 중인 복제본 DB 클러스터의 지연 시간입니다.</p> <p>이 지표는 MySQL <code>Seconds_Behind_Master</code> 명령의 <code>SHOW SLAVE STATUS</code> 필드의 값을 보고합니다. 이 측정치는 여러 AWS 리전에 걸쳐 복제 중인 Aurora DB 클러스터들 사이의 복제 지연을 모니터링하는 데 유용합니다. 자세한 내용은 Aurora MySQL 복제를 참조하십시오.</p>	Aurora MySQL
<code>AuroraGlobalDBReplicatedWriteIO</code>	<p>Aurora 글로벌 데이터베이스의 기본 AWS 리전에서 보조 AWS 리전의 클러스터 볼륨으로 복제된 쓰기 I/O 작업 수입니다.</p> <p>글로벌 데이터베이스의 보조 AWS 리전에 대한 청구 계산은 <code>VolumeWriteIOPS</code>를 사용하여 클러스터 내에서 수행된 쓰기를 설명합니다. 글로벌 데이터베이스의 기본 AWS 리전에 대한 청구 계산은 <code>VolumeWriteIOPS</code>를 사용하여 해당 클러스터 내의 쓰기 활동을 설명하고,</p> <p><code>AuroraGlobalDBReplicatedWriteIO</code>는 글로벌 데이터베이스 내에서 리전 간 복제를 설명합니다.</p> <p>단위: 바이트</p>	Aurora MySQL
<code>AuroraGlobalDBDataTransferBytes</code>	<p>Aurora 글로벌 데이터베이스에서 마스터 AWS 리전에서 보조 AWS 리전으로 전송된 다시 실행 로그 데이터의 양입니다.</p> <p>단위: 바이트</p>	Aurora MySQL
<code>AuroraGlobalDBReplicationLag</code>	<p>Aurora 글로벌 데이터베이스의 경우 기본 AWS 리전에서 업데이트를 복제할 때의 지연 시간(밀리초)입니다.</p>	Aurora MySQL

지표	설명	적용 대상
	단위: 밀리초	
AuroraReplicaLag	Aurora 복제본에서 기본 인스턴스에서 업데이트를 복제할 때의 지연 시간(ms).	Aurora MySQL 및 Aurora PostgreSQL
AuroraReplicaLagMaximum	기본 인스턴스와 DB 클러스터의 각 Aurora DB 인스턴스 사이에 발생하는 최대 지연 시간(밀리초)입니다.	Aurora MySQL 및 Aurora PostgreSQL
AuroraReplicaLagMinimum	기본 인스턴스와 DB 클러스터의 각 Aurora DB 인스턴스 사이에 발생하는 최소 지연 시간(밀리초)입니다.	Aurora MySQL 및 Aurora PostgreSQL
BacktrackChangeRecordsCreationRate	DB 클러스터를 위해 5분간 생성된 역추적 변경 레코드의 수.	Aurora MySQL
BacktrackChangeRecordsStored	DB 클러스터에 의해 사용된 역추적 변경 레코드의 실제 수.	Aurora MySQL
BacktrackWindowActual	대상 역추적 기간과 실제 역추적 기간 사이의 차이.	Aurora MySQL
BacktrackWindowAlert	정해진 시간 동안 실제 역추적 기간이 대상 역추적 기간보다 작은 횟수.	Aurora MySQL
BackupRetentionPeriodStorageUsed	Aurora DB 클러스터의 백업 보존 기간 내 특정 시점 복원 기능을 지원하는 데 사용되는 총 백업 스토리지 양(단위: 바이트)입니다. TotalBackupStorageBilled 지표를 통해 보고되는 총계에 포함됩니다. 각 Aurora 클러스터마다 개별적으로 계산됩니다. 자침은 Aurora 백업 스토리지 사용량 파악 (p. 270) 단원을 참조하십시오.	Aurora MySQL 및 Aurora PostgreSQL
BinLogDiskUsage	마스터에서 이진 로그가 차지하는 디스크 공간 크기(바이트 단위).	Aurora MySQL
BlockedTransactions	데이터베이스에서 1초마다 차단되는 평균 트랜잭션 수	Aurora MySQL
BufferCacheHitRatio	버퍼 캐시에서 처리하는 요청 비율.	Aurora MySQL 및 Aurora PostgreSQL
CommitLatency	커밋 작업의 지연 시간(ms).	Aurora MySQL 및 Aurora PostgreSQL
CommitThroughput	초당 커밋 작업의 평균 수.	Aurora MySQL 및 Aurora PostgreSQL

지표	설명	적용 대상
CPUCreditBalance	<p>한 인스턴스가 모은 CPU 크레딧 수입니다. 이 지표는 db.t2.small 및 db.t2.medium 인스턴스에만 적용됩니다. Aurora MySQL DB 인스턴스가 지정된 속도로 기준 성능을 넘어 얼마나 오래 버스트 할 수 있는지 결정하는 데 사용됩니다.</p> <p>Note</p> <p>CPU 크레딧 지표는 5분 간격으로 보고됩니다.</p>	Aurora MySQL
CPUCreditUsage	<p>지정된 기간 동안 소비한 CPU 크레딧 수입니다. 이 지표는 db.t2.small 및 db.t2.medium 인스턴스에만 적용됩니다. 이것은 Aurora MySQL DB 인스턴스에 할당된 가상 CPU에서 처리 명령에 실제 CPU를 사용한 시간을 씩별합니다.</p> <p>Note</p> <p>CPU 크레딧 지표는 5분 간격으로 보고됩니다.</p>	Aurora MySQL
CPUUtilization	Aurora DB 인스턴스의 CPU 사용률입니다.	Aurora MySQL 및 Aurora PostgreSQL
DatabaseConnections	Aurora DB 인스턴스에 대한 연결 수입니다.	Aurora MySQL 및 Aurora PostgreSQL
DDLLatency	생성, 변경, 하락 요청 등 — 데이터 정의 언어(DDL) 요청의 지연 시간(밀리초)입니다.	Aurora MySQL
DDLTroughput	초당 평균 DDL 요청 수	Aurora MySQL
Deadlocks	데이터베이스 1초마다 발생하는 평균 교착 수	Aurora MySQL 및 Aurora PostgreSQL
DeleteLatency	삭제 쿼리의 지연 시간(ms).	Aurora MySQL
DeleteThroughput	초당 삭제 쿼리의 평균 수.	Aurora MySQL
DiskQueueDepth	디스크 액세스를 대기 중인 읽기/쓰기 요청 수.	Aurora PostgreSQL
DMLLatency	삽입, 업데이트 및 삭제의 지연 시간(ms).	Aurora MySQL

지표	설명	적용 대상
DMLThroughput	초당 평균 삽입, 업데이트 및 삭제 수.	Aurora MySQL
EngineUptime	인스턴스 실행 시간(초).	Aurora MySQL 및 Aurora PostgreSQL
FreeableMemory	사용 가능한 RAM 크기(바이트).	Aurora MySQL 및 Aurora PostgreSQL
FreeLocalStorage	<p>사용 가능한 로컬 스토리지 양(바이트)입니다.</p> <p>다른 DB 엔진과 달리 Aurora DB 인스턴스의 경우, 이 지표는 각 DB 인스턴스에 사용 가능한 스토리지 크기를 보고합니다. 이 값은 DB 인스턴스 클래스에 좌우됩니다(오늘에 대한 자세한 내용은 Amazon RDS 제품 페이지 참조). DB 인스턴스 클래스를 큰 것으로 선택하면 인스턴스의 여유 스토리지 공간을 늘릴 수 있습니다.</p>	Aurora MySQL 및 Aurora PostgreSQL
InsertLatency	삽입 쿼리의 지연 시간(ms).	Aurora MySQL
InsertThroughput	초당 삽입 쿼리의 평균 수.	Aurora MySQL
LoginFailures	초당 평균 로그인 실패 수	Aurora MySQL
MaximumUsedTransactionIDs	<p>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 랙 어라운드를 방지할 목적으로 데이터베이스가 읽기 전용 모드로 강제 전환됩니다. 자세한 내용은 PostgreSQL 설명서에서 Preventing Transaction ID Wraparound Failures를 참조하십시오.</p>	Aurora PostgreSQL
NetworkReceiveThroughput	Aurora MySQL DB 클러스터의 인스턴스 하나가 클라이언트에서 수신하는 네트워크 처리량(초당 바이트 수)입니다. 이 처리량에서 Aurora DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.	Aurora MySQL 및 Aurora PostgreSQL

지표	설명	적용 대상
NetworkThroughput	Aurora MySQL DB 클러스터의 인스턴스 하나가 클라이언트에서 수신하고 클라이언트로 전송하는 네트워크 처리량(bps). 이 처리량에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.	Aurora MySQL 및 Aurora PostgreSQL
NetworkTransmitThroughput	Aurora DB 클러스터의 인스턴스 하나가 클라이언트로 전송하는 네트워크 처리량(초당 바이트 수)입니다. 이 처리량에서 DB 클러스터의 인스턴스와 클러스터 볼륨 간 네트워크 트래픽은 제외됩니다.	Aurora MySQL 및 Aurora PostgreSQL
Queries	초당 실행 쿼리의 평균 수.	Aurora MySQL
RDSToAuroraPostgreSQLReplicaLag	기본 RDS PostgreSQL 인스턴스에서 클러스터의 다른 노드로 업데이트를 복제할 때 지연 시간(초)입니다.	Aurora PostgreSQL
ReadIOPS	초당 평균 디스크 I/O 연산 수. PostgreSQL과 호환되는 Aurora는 1분 간격으로 읽기 IOPS와 쓰기 IOPS를 따로 보고합니다.	Aurora PostgreSQL
ReadLatency	디스크 I/O 연산당 평균 처리 시간.	Aurora PostgreSQL
ReadThroughput	초당 디스크에서 읽은 평균 바이트 수.	Aurora PostgreSQL
ResultSetCacheHitRatio	Resultset 캐시에서 처리되는 평균 요청 수	Aurora MySQL
SelectLatency	선택 쿼리의 지연 시간(ms).	Aurora MySQL
SelectThroughput	초당 평균 select 쿼리 수	Aurora MySQL
SnapshotStorageUsed	Aurora DB 클러스터에 대해 백업 보존 기간 경과 후 모든 Aurora 스냅샷에 사용된 총 백업 스토리지 양(단위: 바이트)입니다. TotalBackupStorageBilled 지표를 통해 보고되는 총계에 포함됩니다. 각 Aurora 클러스터마다 개별적으로 계산됩니다. 자침은 Aurora 백업 스토리지 사용량 파악 (p. 270) 을 참조하십시오.	Aurora MySQL 및 Aurora PostgreSQL
SwapUsage	Aurora PostgreSQL DB 인스턴스에서 사용된 스왑 공간 크기입니다.	Aurora PostgreSQL

지표	설명	적용 대상
TotalBackupStorageBilled	<p>특정 Aurora DB 클러스터에 대한 총 백업 스토리지 양(단위: 바이트)입니다. 이 양에 대해 요금이 청구됩니다.</p> <p><code>BackupRetentionPeriodStorageUsed</code> 및 <code>SnapshotStorageUsed</code> 지표로 측정되는 백업 스토리지를 포함합니다. 각 Aurora 클러스터마다 개별적으로 계산됩니다. 자침은 Aurora 백업 스토리지 사용량 파악 (p. 270)을 참조하십시오.</p>	Aurora MySQL 및 Aurora PostgreSQL
TransactionLogsDiskUsage	Aurora PostgreSQL DB 인스턴스에서 트랜잭션 로그가 차지하는 디스크 공간 크기입니다.	Aurora PostgreSQL
UpdateLatency	업데이트 쿼리의 지연 시간(ms).	Aurora MySQL
UpdateThroughput	초당 업데이트 쿼리의 평균 수.	Aurora MySQL
AuroraVolumeBytesLeftTotal	<p>바이트 단위로 측정되는 클러스터 볼륨의 잔여 가용 공간입니다. 클러스터 볼륨이 확장됨에 따라 이 값은 감소합니다. 이 값이 0에 도달하면 클러스터에서 공간 부족 오류를 보고합니다. 이 값은 Aurora 클러스터가 64 TiB 크기 한도에 근접하는지 여부를 감지하고 싶을 때 <code>VolumeBytesUsed</code>보다 더 쉽고 안정적으로 모니터링할 수 있습니다.</p> <p><code>AuroraVolumeBytesLeftTotal</code>에서는 내부 정리에 사용되는 스토리지와 스토리지 결제에 영향을 미치지 않는 기타 할당을 고려합니다.</p>	Aurora MySQL
VolumeBytesUsed	<p>Aurora DB 인스턴스가 사용하는 스토리지 양(바이트 단위)입니다.</p> <p>이 값은 Aurora DB 클러스터의 요금에 영향을 미칩니다(요금에 대한 자세한 내용은 Amazon RDS 제품 페이지를 참조하십시오).</p> <p>이 값에는 스토리지 결제에 영향을 미치지 않는 일부 내부 스토리 할당이 반영되지 않습니다. 따라서 <code>VolumeBytesUsed</code>를 64 TiB 스토리지 한도와 비교하는 대신에 <code>AuroraVolumeBytesLeftTotal</code>이 0에 근접하고 있는지 여부를 테스트하여 공간 부족 문제를 더 정확하게 예상할 수 있습니다.</p>	Aurora MySQL 및 Aurora PostgreSQL

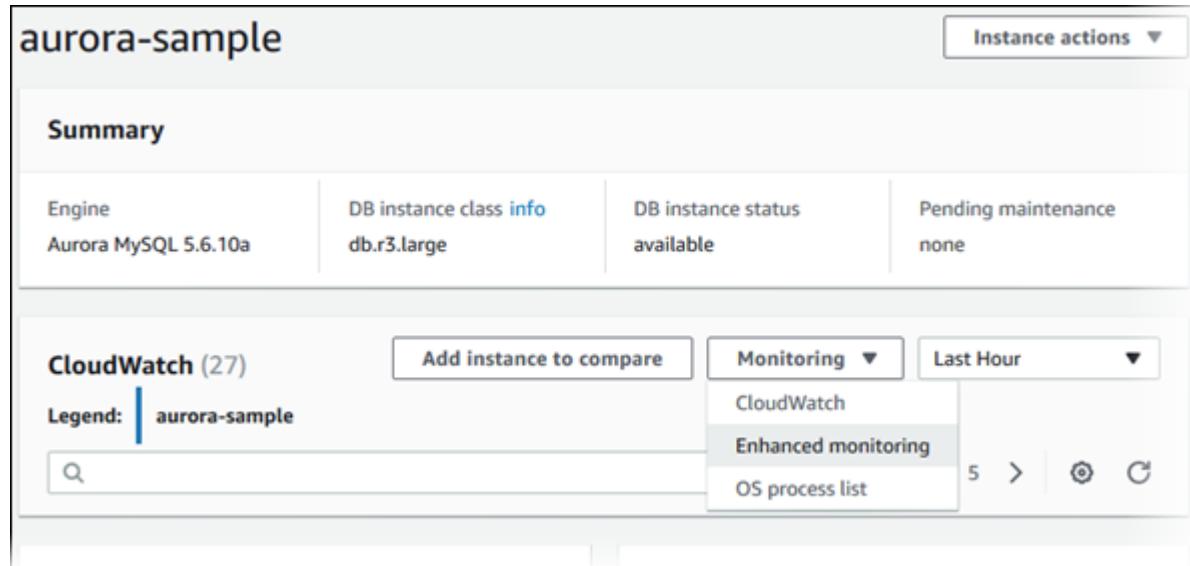
지표	설명	적용 대상
VolumeReadIOPS	<p>클러스터 볼륨에서 요금이 청구된 읽기 I/O 작업의 수로, 5분마다 보고됩니다.</p> <p>요금이 청구된 읽기 작업은 클러스터 볼륨 수준에서 계산되며, Aurora DB 클러스터의 모든 인스턴스에 대해 집계된 후 5분 간격으로 보고됩니다. 이 값은 5분 동안의 읽기 작업 측정치 값을 사용하여 계산됩니다. 요금 부과된 읽기 작업 측정치의 값을 300로 나눠서 초당 요금 부과된 읽기 작업의 양을 확인할 수 있습니다. 예를 들어 요금 부과된 읽기 작업이 13,686을 반환할 경우 초당 요금 부과된 읽기 작업은 $45(13,686 / 300 = 45.62)$입니다.</p> <p>버퍼 캐시에 없는 데이터베이스 페이지를 요청하는 쿼리에 대해서는 요금 부과된 읽기 작업이 발생하므로 스토리지에서 로드해야 합니다. 쿼리 결과를 스토리지에서 읽은 후 버퍼 캐시로 로드하면 요금 부과된 작업이 급증할 수 있습니다.</p>	Aurora MySQL 및 Aurora PostgreSQL
VolumeWriteIOPS	5분 간격으로 보고되는 클러스터 볼륨에 대한 평균 디스크 쓰기 I/O 작업 수입니다. 요금 부과된 쓰기 작업이 계산되는 방법에 대한 자세한 내용은 위에 나와 있는 VolumeReadIOPS에 대한 설명을 참조하십시오.	Aurora MySQL 및 Aurora PostgreSQL
WriteIOPS	초당 평균 디스크 I/O 연산 수. Aurora PostgreSQL은 1분의 간격을 두고 읽기 IOPS와 쓰기 IOPS를 따로 보고합니다.	Aurora PostgreSQL
WriteLatency	디스크 I/O 연산당 평균 처리 시간.	Aurora PostgreSQL
WriteThroughput	초당 디스크에 쓴 평균 바이트 수.	Aurora PostgreSQL

Amazon RDS 콘솔에서 Aurora 메트릭보기

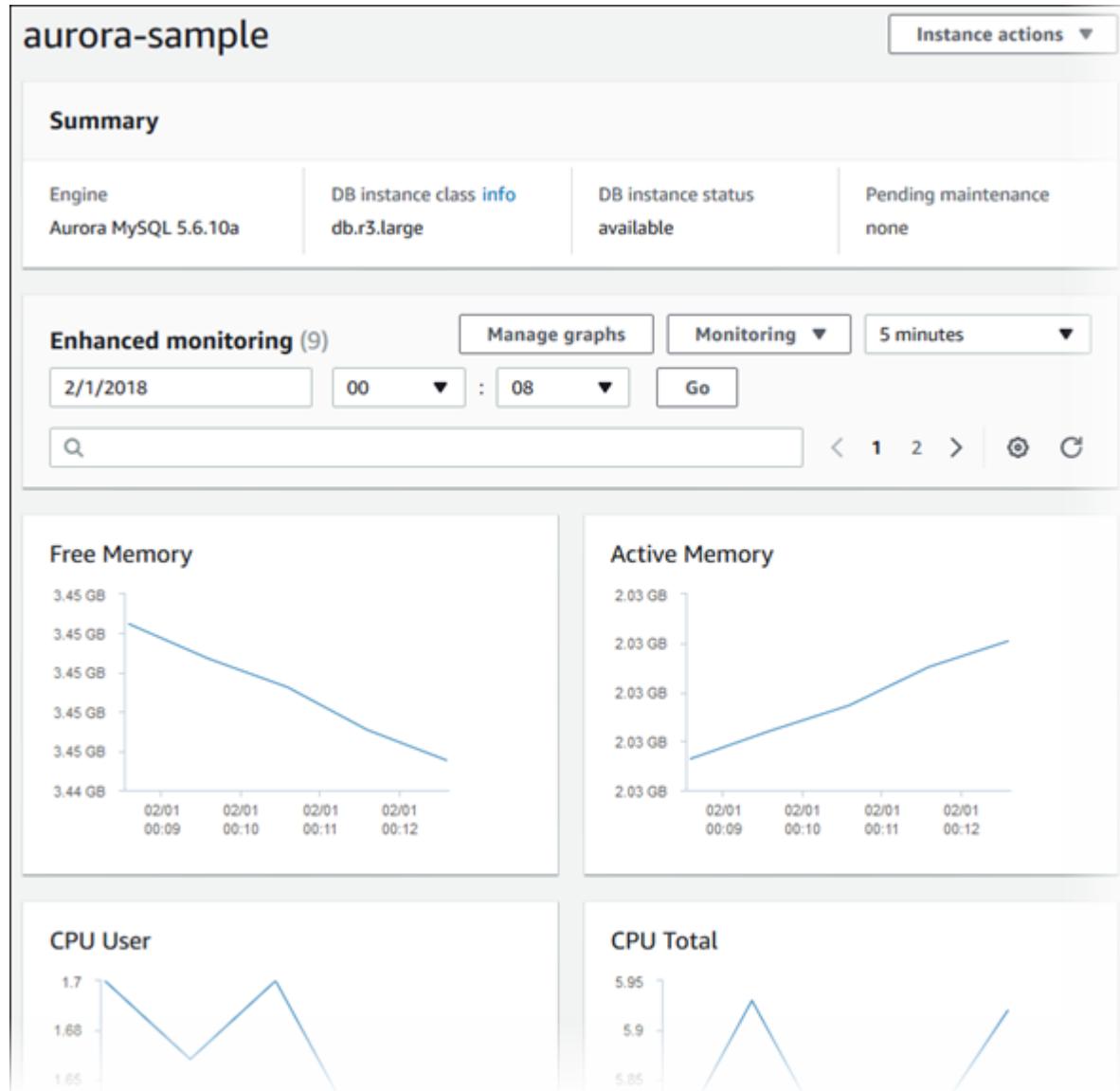
Aurora DB 클러스터의 상태와 성능을 모니터링하면, Amazon RDS 콘솔의 Amazon Aurora가 제공하는 지표의 일부(전부는 아님)를 확인할 수 있습니다. Amazon RDS 콘솔에 사용 가능한 Aurora 지표의 상세한 목록은 [Amazon RDS 콘솔에서 사용 가능한 Aurora 지표 \(p. 355\)](#)를 참조하십시오.

Amazon RDS 콘솔에서 Aurora 측정치를 확인하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 인스턴스를 선택합니다.
3. 세부 정보를 모니터링하고자 하는 DB 인스턴스 이름을 선택합니다.
4. CloudWatch 섹션의 [Monitoring]에서 측정치 표시 방식을 지정하는 다음 옵션 중 하나를 선택합니다.
 - Cloudwatch – CloudWatch 지표의 요약을 보여줍니다. 각 지표에는 특정 시간대에서 지표를 모니터링 한 그래프도 포함되어 있습니다. 자세한 내용은 [Amazon CloudWatch로 모니터링 \(p. 329\)](#) 단원을 참조하십시오.
 - 확장 모니터링 – 확장 모니터링을 활성화한 상태에서 Aurora DB 인스턴스에 사용 가능한 OS 지표를 요약하여 표시합니다. 각 지표에는 특정 시간대에서 지표를 모니터링한 그래프도 포함되어 있습니다. 자세한 내용은 [확장 모니터링 \(p. 358\)](#) 단원을 참조하십시오.
 - OS 프로세스 목록 – DB 인스턴스 또는 DB 클러스터에서 실행하는 프로세스와 관련 지표(CPU 비율, 메모리 사용량 등)가 표시됩니다.



5. 다음 이미지는 [Enhanced monitoring]이 선택된 상태에서 지표 보기를 표시합니다.



Amazon RDS 콘솔에서 사용 가능한 Aurora 지표

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.

다음 Aurora지표는 Amazon RDS 콘솔에서 사용할 수 없습니다.

- AuroraBinlogReplicaLag
- DeleteLatency
- DeleteThroughput
- EngineUptime
- InsertLatency
- InsertThroughput

- NetworkThroughput
- Queries
- UpdateLatency
- UpdateThroughput

또한 일부 Aurora 지표는 특정 인스턴스 클래스에서만, 또는 DB 인스턴스에서만, 또는 다른 이름 및 측정 단위와 함께 표시됩니다.

- CPUCreditBalance 및 CPUCreditUsage 측정치는 db.t2.small 및 db.t2.medium 인스턴스에 대해서만 표시됩니다.
- 다른 이름과 함께 표시되는 다음 지표는 다음과 같이 나열됩니다.

지표	표시 이름
AuroraReplicaLagMaximum	최대 복제 지연 시간
AuroraReplicaLagMinimum	최소 복제 지연 시간
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 측정치는 Amazon RDS 콘솔에서 바이트가 아니라 MB 단위로 표시됩니다.
 - FreeableMemory
 - FreeLocalStorage
 - NetworkReceiveThroughput
 - NetworkTransmitThroughput

최신 지표 보기

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.

범주	지표
SQL	ActiveTransactions BlockedTransactions BufferCacheHitRatio CommitLatency

범주	지표
	CommitThroughput DatabaseConnections DDLLatency DDLThroughput Deadlocks DMLLatency DMLThroughput LoginFailures ResultSetCacheHitRatio SelectLatency SelectThroughput
시스템	AuroraReplicaLag AuroraReplicaLagMaximum AuroraReplicaLagMinimum CPUCreditBalance CPUCreditUsage CPUUtilization FreeableMemory FreeLocalStorage NetworkReceiveThroughput
배포	AuroraReplicaLag BufferCacheHitRatio ResultSetCacheHitRatio SelectThroughput

Note

Amazon RDS 콘솔에서 최신 지표 보기의 SQL 카테고리 아래에 표시된 실패한 SQL 문 지표는 Amazon Aurora에 적용되지 않습니다.

확장 모니터링

Amazon RDS는 DB 인스턴스가 실행되는 운영 체제(OS)에 대한 측정치를 실시간으로 제공합니다. 콘솔을 사용하여 DB 인스턴스에 대한 측정치를 보거나, 선택한 모니터링 시스템의 Amazon CloudWatch Logs에서 Enhanced Monitoring JSON 출력을 사용할 수 있습니다.

기본적으로 확장 모니터링 지표는 CloudWatch Logs 로그에 30일간 저장됩니다. 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 변경](#)을 참조하십시오.

확장 모니터링 사용 비용은 다음과 같은 몇 가지 요인에 따라 달라집니다.

- Amazon CloudWatch Logs이 제공하는 프리 티어를 초과하는 Enhanced Monitoring에 대해서만 비용이 청구됩니다.

요금에 대한 자세한 내용은 [Amazon CloudWatch 요금](#)을 참조하십시오.

- 모니터링 간격이 작을수록 OS 측정치가 더 자주 보고되고 모니터링 비용이 증가합니다.
- 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.

CloudWatch 측정치와 Enhanced Monitoring 측정치의 차이점

CloudWatch는 DB 인스턴스의 하이퍼바이저에서 CPU 사용률에 대한 측정치를 수집하고, Enhanced Monitoring에서는 인스턴스의 에이전트에서 측정치를 수집합니다. 하이퍼바이저 계층에서는 소량의 작업만 수행하므로 두 측정치 간의 차이점을 확인할 수 있습니다. DB 인스턴스에서 사용하는 인스턴스 클래스가 적을수록 단일 물리적 인스턴스에서 하이퍼바이저 계층에 의해 관리되는 가상 머신(VM)의 수가 더 많아지므로 차이가 더 커질 수 있습니다. Enhanced Monitoring 측정치는 DB 인스턴스의 여러 프로세스 또는 스레드에서 CPU를 사용하는 방법을 확인하려는 경우에 유용합니다.

확장 모니터링 설정 및 활성화

확장 모니터링을 설정하고 활성화하려면 아래 나열된 단계를 수행하십시오.

시작하기 전

Enhanced Monitoring은 사용자를 대신하여 CloudWatch Logs에 OS 측정치 정보를 보낼 수 있는 권한이 필요합니다. AWS Identity and Access Management(IAM) 역할을 사용하여 확장 모니터링에 필요한 권한을 부여합니다.

콘솔에서 Enhanced Monitoring을 처음으로 활성화할 때 [Monitoring Role] 속성에 대한 [Default] 옵션을 선택하여 RDS에서 필요한 IAM 역할을 생성하도록 할 수 있습니다. 그러면 RDS에서 이름이 `rds-monitoring-role`인 역할을 자동으로 생성하고 해당 역할을 지정된 DB 인스턴스 또는 일기 전용 복제본에 사용합니다.

Enhanced Monitoring을 활성화하기 전에 필요한 역할을 생성한 다음 Enhanced Monitoring을 활성화할 때 새 역할을 이름을 지정할 수도 있습니다. AWS CLI 또는 RDS API를 사용하여 Enhanced Monitoring을 활성화할 경우 이 필수 역할을 생성해야 합니다.

적절한 IAM 역할을 생성하여 Amazon RDS에게 사용자를 대신하여 Amazon CloudWatch Logs 서비스와 통신하도록 허용하려면 다음 단계를 수행합니다.

확장 모니터링을 활성화하는 사용자는 `PassRole` 권한을 부여받아야 합니다. 자세한 내용은 IAM 사용 설명서의 [사용자에게 AWS 서비스에 역할을 전달할 수 있는 권한 부여](#)에 있는 예제 2를 참조하십시오.

Amazon RDS Enhanced Monitoring에 대한 IAM 역할을 생성하려면

1. <https://console.aws.amazon.com>에서 **IAM 콘솔**을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. [Create role]을 선택합니다.
4. [AWS service] 탭을 선택한 다음 서비스 목록에서 [RDS]를 선택합니다.
5. RDS - Enhanced Monitoring(RDS - 확장 모니터링)을 선택한 다음 Next: Permissions(다음: 권한)를 선택합니다.
6. Attached permissions policy(연결된 권한 정책) 페이지에 `AmazonRDSEnhancedMonitoringRole`이 표시되었는지 확인한 다음 Next: Tags(다음: 태그)를 선택합니다.
7. Add tags(태그 추가) 페이지에서 Next: Review(다음: 검토)를 선택합니다.
8. Role Name(역할 이름)에 `emaccess`와 같은 역할의 이름을 입력한 다음 Create role(역할 생성)을 선택합니다.

Enhanced Monitoring 활성화 및 비활성화

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 확장 모니터링을 활성화하거나 비활성화할 수 있습니다.

콘솔

DB 클러스터 또는 읽기 전용 복제본을 생성할 때 또는 DB 클러스터를 수정할 때 확장 모니터링을 활성화할 수 있습니다. 확장 모니터링을 활성화하기 위해 DB 인스턴스를 수정하는 경우 DB 인스턴스를 재부팅하지 않아도 변경 내용이 적용됩니다.

다음 작업 중 하나를 수행할 때 RDS 콘솔에서 확장 모니터링을 활성화할 수 있습니다.

- DB 클러스터 생성 – 추가 구성아래의 모니터링 섹션에서 확장 모니터링을 활성화할 수 있습니다.
- 읽기 전용 복제본 생성 – 모니터링 섹션에서 확장 모니터링을 활성화할 수 있습니다.
- DB 인스턴스 수정 – 모니터링 섹션에서 확장 모니터링을 활성화할 수 있습니다.

RDS 콘솔을 사용하여 Enhanced Monitoring을 활성화하려면 [Monitoring] 섹션으로 스크롤한 후 다음 작업을 수행합니다.

1. DB 인스턴스 또는 읽기 전용 복제본에 대해 확장 모니터링 활성화를 선택합니다.
2. 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.
3. 세부 수준 속성을 DB 인스턴스 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.

Enhanced Monitoring을 비활성화하려면 [Disable enhanced monitoring]을 선택합니다.

Monitoring

Enhanced monitoring

Enable enhanced monitoring
Enhanced monitoring metrics are useful when you want to see how different processes or threads use the CPU.

Disable enhanced monitoring

Monitoring Role	Granularity
rds-monitoring-role	60 sec... ▾

Enable Enhanced Monitoring의 경우 DB 인스턴스를 다시 시작하지 않아도 됩니다.

Note

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 확장 모니터링을 활성화하려면 다음 명령에서 `--monitoring-interval` 옵션을 0 이외의 값으로 설정하고 `--monitoring-role-arn` 옵션을 [시작하기 전 \(p. 358\)](#)에서 생성된 역할로 설정합니다.

- [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 60입니다.

AWS CLI를 사용하여 확장 모니터링을 비활성화하려면 다음 명령에서 `--monitoring-interval` 옵션을 0으로 설정합니다.

Example

다음 예제에서는 DB 인스턴스에 대해 확장 모니터링을 활성화합니다.

Linux, OS X, Unix의 경우:

```
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
```

RDS API

RDS API를 사용하여 확장 모니터링을 활성화하려면 다음 작업에서 `MonitoringInterval` 파라미터를 0 이외의 값으로 설정하고 `MonitoringRoleArn` 파라미터를 [시작하기 전 \(p. 358\)](#)에서 생성된 역할로 설정합니다.

- [CreateDBInstance](#)
- [CreateDBInstanceReadReplica](#)
- [ModifyDBInstance](#)

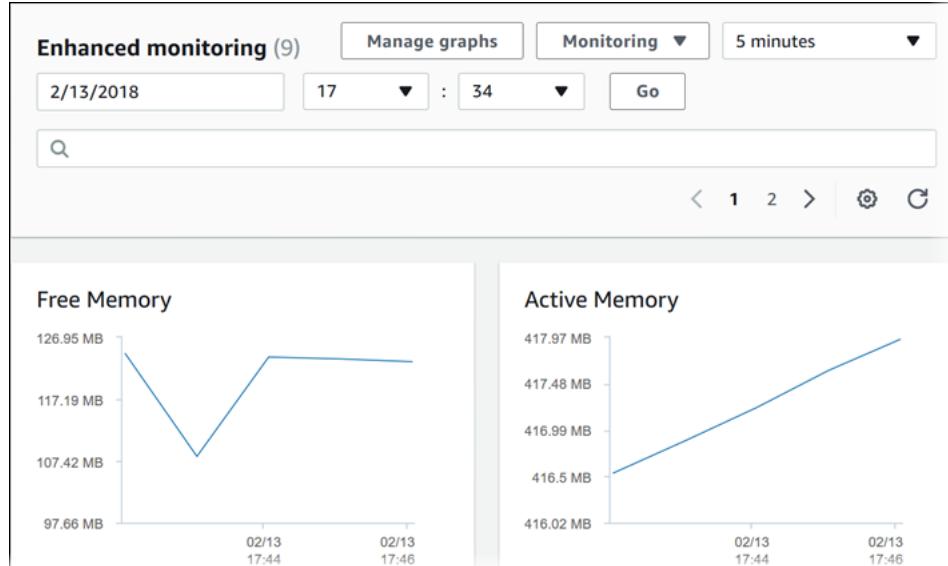
`MonitoringInterval` 파라미터는 확장 모니터링 지표가 수집되는 시점 간격(초)을 지정합니다. 이 파라미터의 유효한 값은 0, 1, 5, 10, 15, 30 및 60입니다.

RDS API를 사용하여 확장 모니터링을 비활성화하려면 이러한 작업에서 `MonitoringInterval` 파라미터를 0으로 설정합니다.

확장 모니터링 보기

모니터링에서 확장 모니터링을 선택하면 RDS 콘솔에서 확장 모니터링이 보고하는 OS 측정치를 볼 수 있습니다.

확장 모니터링 페이지가 다음과 같이 표시됩니다.



DB 인스턴스에서 실행 중인 프로세스에 대한 자세한 정보를 보려면 [Monitoring]에 대해 [OS process list]를 선택합니다.

프로세스 목록 보기는 다음과 같이 표시됩니다.

Process List						
<input type="text"/> Filter process list						
NAME	VIRT	RES	CPU%	MEM%	VMLIMIT	
postgres [3181] ^t	283.55 MB	17.11 MB	0.02	1.72		
postgres:						
rdsadmin	384.7 MB	9.51 MB	0.02	0.95		
rdsadmin						
localhost(40156)						
idle [2953] ^t						

프로세스 목록 보기에서 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 processes] – 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 Logs를 사용하여 Enhanced Monitoring 보기 \(p. 363\)](#) 단원을 참조하십시오.

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 Logs를 사용하여 Enhanced Monitoring 보기

DB 인스턴스에 대한 Enhanced Monitoring을 활성화한 후 CloudWatch Logs를 사용하여 DB 인스턴스에 대한 측정치를 볼 수 있습니다. 각 로그 스트림에는 모니터링 중인 단일 DB 인스턴스가 표시됩니다. 로그 스트림 식별자는 DB 인스턴스에 대한 리소스 식별자(DBResourceID)입니다.

Enhanced Monitoring 로그 데이터를 보려면

1. <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.
2. 필요한 경우 DB 인스턴스가 있는 리전을 선택합니다. 자세한 내용은 Amazon Web Services 일반 참조의 [리전 및 엔드포인트](#)를 참조하십시오.
3. 탐색 창에서 로그를 선택합니다.
4. 로그 그룹 목록에서 RDSOSMetrics를 선택합니다.
5. 로그 스트림 목록에서 보려는 로그 스트림을 선택합니다.

사용 가능한 OS 측정치

다음 표에는 Amazon CloudWatch Logs에서 사용 가능한 OS 측정치가 나와 있습니다.

Aurora의 지표

그룹	측정치	설명
General	engine	DB 인스턴스에 대한 데이터베이스 엔진
	instanceID	DB 인스턴스 식별자
	instanceResourceID	DB 인스턴스에 대해 리전별로 고유하고 변경 불가능한 식별자이며, 로그 스트림 식별자로도 사용됩니다.
	numVCpus	DB 인스턴스의 가상 CPU 수
	timestamp	측정치를 가져온 시간
	uptime	DB 인스턴스가 활성 상태로 유지된 시간
	version	OS 측정치 스트림 JSON 형식의 버전
cpuUtilization	guest	게스트 프로그램에서 사용 중인 CPU의 비율
	idle	유휴 상태인 CPU의 비율
	irq	소프트웨어 인터럽트에서 사용 중인 CPU의 비율
	nice	가장 낮은 우선순위로 실행 중인 프로그램에서 사용 중인 CPU의 비율
	steal	다른 가상 머신에서 사용 중인 CPU의 비율
	system	커널에서 사용 중인 CPU의 비율
	total	사용 중인 CPU의 총 비율입니다. 이 값에는 nice 값이 포함됩니다.
	user	사용자 프로그램에서 사용 중인 CPU의 비율

그룹	측정치	설명
	wait	I/O 액세스를 대기 중인 동안 사용되지 않은 CPU의 비율
fileSys	maxFiles	파일 시스템에 대해 생성될 수 있는 최대 파일 수
	mountPoint	파일 시스템의 경로
	name	파일 시스템의 이름
	total	파일 시스템에 사용 가능한 총 디스크 공간(KB)
	used	파일 시스템에서 파일에 사용된 디스크 공간의 양(KB)
	usedFilePercent	사용 중인 사용 가능한 파일의 비율
	usedFiles	파일 시스템의 파일 수
	usedPercent	사용 중인 파일 시스템 디스크 공간의 비율
loadAverageMinutes	fifteen	마지막 15분 동안 CPU 시간을 요청한 프로세스 수
	five	마지막 5분 동안 CPU 시간을 요청한 프로세스 수
	one	마지막 1분 동안 CPU 시간을 요청한 프로세스 수
memory	active	할당된 메모리의 양(KB)
	buffers	스토리지 디바이스에 쓰기 이전에 I/O 요청을 버퍼링하는 데 사용되는 메모리의 양(KB)
	cached	파일 시스템 기반 I/O를 캐시하는 데 사용된 메모리의 양
	dirty	수정되었지만 스토리지의 관련 데이터 블록에 기록되지 않은 RAM의 메모리 페이지 양(KB)
	free	할당되지 않은 메모리의 양(KB)
	hugePagesFree	사용 가능한 방대한 페이지 수입니다. 방대한 페이지는 Linux 커널의 기능입니다.
	hugePagesRsvd	커밋된 방대한 페이지의 수
	hugePagesSize	각 방대한 페이지 단위의 크기(KB)
	hugePagesSurp	총계 대비 사용 가능한 초과 방대한 페이지 수
	hugePagesTotal	시스템에 대한 방대한 페이지의 총 크기(KB)입니다.
	inactive	가장 적게 사용되는 메모리 페이지의 양(KB)
	mapped	프로세스 주소 공간 내에 메모리 매핑되는 총 파일 시스템 콘텐츠 양(KB)
	pageTables	페이지 표에 사용된 메모리의 양(KB)
	slab	재사용 가능한 커널 데이터 구조의 양(KB)
	total	총 메모리 양(KB)
	writeback	RAM에서 지원 스토리지에 아직 기록 중인 더티 페이지의 양(KB)

그룹	측정치	설명
network	interface	DB 인스턴스에 대해 사용 중인 네트워크 인터페이스의 식별자
	rx	초당 수신된 바이트 수
	tx	초당 업로드된 바이트 수
processList	cpuUsedPc	프로세스에서 사용된 CPU 비율
	id	프로세스의 식별자
	memoryUsedPc	프로세스에 사용된 메모리의 양(KB)
	name	프로세스의 이름입니다.
	parentID	프로세스의 상위 프로세스에 대한 프로세스 식별자
	rss	프로세스에 할당된 RAM의 양(KB)
	tgid	스레드 그룹 식별자이며, 스레드가 속한 프로세스 ID를 나타내는 숫자입니다. 이 식별자는 동일한 프로세스에서 스레드를 그룹화하는 데 사용됩니다.
swap	VIRT	프로세스에 할당된 가상 메모리의 양(KB)
	swap	사용 가능한 스왑 메모리 양(KB)
	swap in	디스크에서 스왑된 메모리 양(KB)
	swap out	디스크로 스왑된 총 메모리 양(KB)
	free	사용 가능한 스왑 메모리 양(KB)
tasks	committed	캐시 메모리로 사용된 스왑 메모리의 양(KB)
	blocked	차단되는 작업 수
	running	실행 중인 작업 수
	sleeping	절전 상태인 작업 수
	stopped	중단된 작업 수
	total	총 작업 수
	zombie	상위 작업은 활성화되었지만 비활성 상태인 하위 작업 수

Amazon RDS 성능 개선 도우미 사용

Amazon RDS 성능 개선 도우미는 데이터베이스 성능을 분석하고 문제를 해결할 수 있도록 Amazon RDS DB 인스턴스 로드를 모니터링합니다. Amazon RDS 성능 개선 도우미는 현재 다음 DB 엔진에서 사용할 수 있습니다.

- MySQL과 호환되는 Amazon Aurora 버전 2.04.2 및 2.x 상위 버전(MySQL 5.7과 호환됨)
- MySQL과 호환되는 Amazon Aurora 버전 1.17.3 및 1.x 상위 버전(MySQL 5.6과 호환)
- PostgreSQL과 호환되는 Amazon Aurora
- Amazon RDS for MariaDB 버전 10.4.8 및 그 이상의 10.4 버전, 버전 10.3.13 및 그 이상의 10.3 버전, 버전 10.2.21 및 그 이상의 10.2 버전

- Amazon RDS for MySQL 버전 8.0.17 및 그 이상의 8.0 버전, 버전 5.7.22 및 그 이상의 5.7 버전, 버전 5.6.41 및 그 이상의 5.6 버전
- Amazon RDS for Microsoft SQL Server(SQL Server 2008을 제외한 모든 버전)
- Amazon RDS for PostgreSQL 버전 10 및 11
- Amazon RDS for Oracle(모든 버전)

Note

Amazon RDS 성능 개선 도우미는 MariaDB 버전 10.0 또는 10.1이나 MySQL 버전 5.5에서는 지원되지 않습니다.

MariaDB 및 MySQL용 Amazon RDS의 경우 db.t2.micro, db.t2.small, db.t3.micro, db.t3.small DB 인스턴스 클래스에서는 성능 개선 도우미가 지원되지 않습니다.

Aurora MySQL의 경우 db.t2 또는 db.t3 DB 인스턴스 클래스에서는 성능 개선 도우미가 지원되지 않습니다.

병렬 쿼리를 위해 활성화된 Aurora MySQL DB 클러스터에 대해서는 성능 개선 도우미가 지원되지 않습니다.

성능 개선 도우미(Performance Insights)는 기존 Amazon RDS 모니터링 기능을 확장한 것으로서 데이터베이스 성능을 표시하여 성능 문제를 분석하는데 효과적입니다. 성능 개선 도우미 대시보드가 데이터베이스 부하를 시각화하여 대기 시간, SQL 문, 호스트 또는 사용자를 기준으로 부하를 필터링합니다.

성능 개선 도우미는 Amazon RDS를 사용해 작업하는 모든 DB 엔진을 위한 콘솔 생성 마법사 내에서 기본적으로 활성화되어 있습니다. DB 인스턴스의 데이터베이스가 2개 이상인 경우에는 모든 데이터베이스에 대한 성능 데이터가 DB 인스턴스에 집계됩니다.

성능 개선 도우미의 중앙 지표는 DB 엔진에서 활성 세션의 평균 수를 의미하는 DB Load입니다. DB Load 지표는 1초마다 수집됩니다. 활성 세션이란 DB 엔진에게 작업을 제출하여 현재 응답 대기 중인 연결 세션을 말합니다. 예를 들어 DB 엔진에게 SQL 쿼리를 제출하면 DB 엔진이 이 쿼리를 처리하는 동안 해당하는 데이터베이스 세션이 활성화됩니다.

DB Load와 대기 이벤트 데이터를 결합하여 활성 세션의 전체 상태를 이해할 수 있습니다. 대기 이벤트는 SQL 문 실행이 계속되려면 특정 이벤트가 발생하기를 기다려야 하는 조건을 말합니다. 예를 들어 잠긴 리소스의 잠금이 해제될 때까지 대기해야 SQL 문이 실행될 수 있습니다. 대기 이벤트는 DB 엔진마다 다릅니다.

- Aurora MySQL에 가장 일반적으로 사용되는 대기 이벤트 목록을 보려면 [Aurora MySQL 이벤트 \(p. 677\)](#) 단원을 참조하십시오.
- 모든 MySQL 대기 이벤트에 대한 자세한 내용은 MySQL 설명서의 [대기 이벤트 요약 테이블](#)을 참조하십시오.
- Aurora PostgreSQL에 가장 일반적으로 사용되는 대기 이벤트 목록을 보려면 [Amazon Aurora PostgreSQL 이벤트 \(p. 904\)](#) 단원을 참조하십시오.
- 모든 PostgreSQL 대기 이벤트에 대한 자세한 내용은 PostgreSQL 설명서의 [PostgreSQL 대기 이벤트](#)를 참조하십시오.

세션 정보는 수집 및 집계 후 대시보드에 평균 활성 세션 차트로 표시됩니다. 평균 활성 세션 차트에는 최대 CPU 값이 선으로 표시되기 때문에 활성 세션이 최대 값을 초과하는지 알 수 있습니다. 최대 CPU 값은 DB 인스턴스에서 vCPU(가상 CPU) 코어의 수로 결정됩니다.

평균 활성 세션 차트의 로드가 최대 CPU 선을 상회하는 경우가 찾아지고 CPU가 기본 대기 상태라면 시스템 CPU에서 과부하가 발생한 것입니다. 이러한 경우 연결 수를 인스턴스에 맞게 조절하거나, CPU 부하가 높은 SQL 쿼리를 모두 조정하거나, 인스턴스 클래스의 크기를 늘리는 것이 좋습니다. 대기 상태가 찾고 일관성 있게 나타난다면 병목 현상 또는 리소스 유지 문제가 있을 수 있으므로 이를 해결해야 합니다. 로드가 최대 CPU 선을 넘지 않는다 하더라도 이러한 문제가 나타날 수 있습니다.

다음 비디오에서 성능 개선 도우미의 개요를 볼 수 있습니다.

성능 개선 도우미를 사용하여 Amazon Aurora PostgreSQL의 성능 분석

주제

- [성능 개선 도우미 활성화 \(p. 367\)](#)
- [성능 개선 도우미의 액세스 제어 \(p. 371\)](#)
- [성능 개선 도우미 대시보드 사용 \(p. 372\)](#)
- [추가 사용자 인터페이스 기능 \(p. 385\)](#)
- [성능 개선 도우미 API \(p. 386\)](#)
- [Amazon CloudWatch에 게시되는 성능 개선 도우미 지표 \(p. 398\)](#)
- [성능 개선 도우미 카운터 \(p. 400\)](#)
- [AWS CloudTrail를 사용하여 성능 개선 도우미 호출 로깅 \(p. 406\)](#)

성능 개선 도우미 활성화

성능 개선 도우미를 사용하려면 DB 인스턴스에서 이 기능을 활성화해야 합니다.

Aurora 글로벌 데이터베이스와 함께 성능 개선 도우미를 사용하는 경우 각 AWS 리전에서 DB 인스턴스에 대해 개별적으로 성능 개선 도우미를 활성화해야 합니다. 세부 정보는 [Aurora Global Database를 위한 성능 개선 도우미 \(p. 53\)](#) 단원을 참조하십시오.

콘솔

새 DB 인스턴스를 생성할 때 콘솔을 사용하여 성능 개선 도우미를 활성화할 수 있습니다. 성능 개선 도우미를 활성화하도록 DB 인스턴스를 수정할 수도 있습니다.

주제

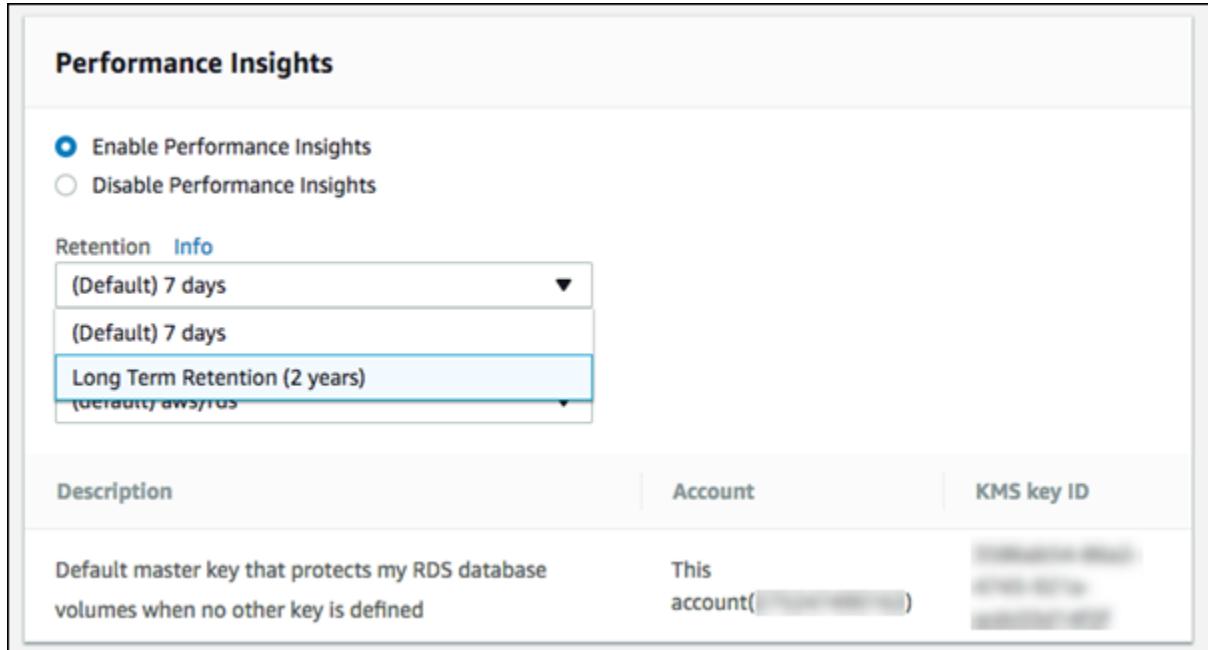
- [DB 인스턴스를 생성할 때 콘솔을 사용하여 성능 개선 도우미 활성화 \(p. 367\)](#)
- [DB 인스턴스를 수정 때 콘솔을 사용하여 성능 개선 도우미 활성화 \(p. 368\)](#)

DB 인스턴스를 생성할 때 콘솔을 사용하여 성능 개선 도우미 활성화

새 DB 인스턴스를 생성할 때 성능 개선 도우미(성능 개선 도우미) 섹션에서 Enable 성능 개선 도우미(성능 개선 도우미)를 선택하면 성능 개선 도우미가 활성화됩니다.

DB 인스턴스를 생성하려면 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#)의 DB 엔진에 대한 지침을 따르십시오.

다음 이미지는 성능 개선 도우미 섹션의 스크린샷입니다.



Enable 성능 개선 도우미(성능 개선 도우미 활성화)를 선택할 때 다음 옵션이 있습니다.

- 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. 7일(기본값) 또는 2년을 선택합니다.
- 마스터 키 – AWS Key Management Service(AWS KMS) 키를 지정합니다. 성능 개선 도우미는 AWS KMS 키를 사용하여 잠재적으로 민감한 모든 데이터를 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 [Amazon Aurora 리소스 암호화 \(p. 945\)](#) 단원을 참조하십시오.

DB 인스턴스를 수정 때 콘솔을 사용하여 성능 개선 도우미 활성화

콘솔을 사용하여 성능 개선 도우미를 활성화하도록 DB 인스턴스를 수정할 수 있습니다.

콘솔을 사용하여 DB 인스턴스의 성능 개선 도우미를 활성화하려면

- AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
- 데이터베이스를 선택합니다.
- 수정하려는 DB 인스턴스를 선택한 후 수정을 선택합니다.
- 성능 개선 도우미(성능 개선 도우미) 섹션에서 Enable 성능 개선 도우미(성능 개선 도우미 활성화).

Enable 성능 개선 도우미(성능 개선 도우미 활성화)를 선택할 때 다음 옵션이 있습니다.

- 보존 – 성능 개선 도우미 데이터를 보존할 시간입니다. 7일(기본값) 또는 2년을 선택합니다.
 - 마스터 키 – AWS Key Management Service(AWS KMS) 키를 지정합니다. 성능 개선 도우미는 AWS KMS 키를 사용하여 잠재적으로 민감한 모든 데이터를 암호화합니다. 데이터는 암호화된 상태로 전송 및 저장됩니다. 자세한 내용은 [Amazon Aurora 리소스 암호화 \(p. 945\)](#) 단원을 참조하십시오.
- [Continue]를 선택합니다.
 - Scheduling of Modifications(수정 사항 예약)에서 다음 중 하나를 선택합니다.
 - 예약된 다음 유지 관리 기간에 적용 – 성능 개선 도우미 수정 사항을 다음 유지 관리 기간에 적용합니다.
 - 즉시 적용 – 성능 개선 도우미 수정 사항을 최대한 빨리 적용합니다.

7. Modify Instance(DB 인스턴스)를 선택합니다.

AWS CLI

[create-db-instance](#) AWS CLI 명령을 사용하여 새 DB 인스턴스를 생성하면 `--enable-performance-insights`를 지정할 때 성능 개선 도우미가 활성화됩니다.

다음 AWS CLI 명령을 사용해 `--enable-performance-insights` 값을 지정할 수도 있습니다.

- [create-db-instance-read-replica](#)
- [modify-db-instance](#)
- [restore-db-instance-from-s3](#)

다음 절차에서는 AWS CLI를 사용하여 DB 인스턴스의 성능 개선 도우미를 활성화하는 방법을 설명합니다.

AWS CLI를 사용하여 DB 인스턴스의 성능 개선 도우미를 활성화하려면

- [modify-db-instance](#) AWS CLI 명령을 호출하고 다음 값을 입력합니다.
 - `--db-instance-identifier` DB 인스턴스의 이름.
 - `--enable-performance-insights`

다음 예제에서는 `sample-db-instance`에 대한 성능 개선 도우미를 활성화합니다.

Linux, OS X, Unix의 경우:

```
aws rds modify-db-instance \
  --db-instance-identifier sample-db-instance \
  --enable-performance-insights
```

Windows의 경우:

```
aws rds modify-db-instance ^
  --db-instance-identifier sample-db-instance ^
  --enable-performance-insights
```

성능 개선 도우미를 활성화할 때 선택적으로 `--performance-insights-retention-period` 옵션을 사용하여 성능 개선 도우미 데이터를 보존할 시간을 일 단위로 지정할 수 있습니다. 유효한 값은 7(기본값) 또는 731(2년)입니다.

다음 예제에서는 `sample-db-instance`의 성능 개선 도우미를 활성화하고 성능 개선 도우미 데이터가 2년 동안 보존되도록 지정합니다.

Linux, OS X, Unix의 경우:

```
aws rds modify-db-instance \
  --db-instance-identifier sample-db-instance \
  --enable-performance-insights \
  --performance-insights-retention-period 731
```

Windows의 경우:

```
aws rds modify-db-instance ^
--db-instance-identifier sample-db-instance ^
--enable-performance-insights ^
--performance-insights-retention-period 731
```

RDS API

[CreateDBInstance](#) 작업 Amazon RDS API 작업을 사용하여 새 DB 인스턴스를 생성할 때 `EnablePerformanceInsights`를 `True`로 설정하면 Performance Insights가 활성화됩니다.

다음 API 작업으로 `EnablePerformanceInsights` 값을 지정할 수도 있습니다.

- [ModifyDBInstance](#)
- [CreateDBInstanceReadReplica](#)
- [RestoreDBInstanceFromS3](#)

성능 개선 도우미를 활성화할 때 선택적으로 `PerformanceInsightsRetentionPeriod` 파라미터를 사용하여 성능 개선 도우미 데이터를 보존할 시간을 일 단위로 지정할 수 있습니다. 유효한 값은 7(기본값) 또는 731(2년)입니다.

MariaDB 또는 MySQL용 Aurora MySQL에서 성능 개선 도우미에 대해 성능 스키마 활성화

MariaDB 또는 MySQL용 Aurora MySQL의 경우, 성능 개선 도우미는 성능 스키마 기능이 활성화되면 자세한 정보를 제공합니다. 성능 개선 도우미를 활성화한 상태에서 MariaDB 또는 MySQL용 Aurora MySQL DB 인스턴스를 생성하면 성능 스키마가 자동으로 활성화됩니다. 성능 개선 도우미를 활성화한 상태에서 DB 인스턴스를 생성하면 성능 스키마 파라미터의 다음 하위 집합이 지정된 값으로 자동으로 설정됩니다.

파라미터 이름	파라미터 값
<code>performance_schema</code>	1
<code>performance-schema-consumer-events-waits-current</code>	ON
<code>performance-schema-instrument</code>	<code>wait/%=ON</code>
<code>performance-schema-consumer-global instrumentation</code>	ON
<code>performance-schema-consumer-thread-instrumentation</code>	ON

파라미터 그룹에 `performance_schema` 파라미터에 대해 명시적으로 설정된 값이 없는 경우에만 성능 스키마가 자동으로 활성화됩니다. `performance_schema` 파라미터를 검사한 후 원본 값이 `user`인 경우 값을 설정할 수 있습니다. 성능 스키마 파라미터가 자동으로 설정되도록 하려면 `performance_schema` 파라미터 값을 재설정합니다. AWS Management 콘솔에서 파라미터를 확인하거나 AWS CLI [describe-db-parameters](#) 명령을 실행하여 파라미터 값의 원본을 확인할 수 있습니다.

performance_schema 파라미터 값을 변경할 때 DB 인스턴스를 재부팅해야 합니다. 성능 개선 도우미를 활성화한 상태에서 새 DB 인스턴스를 생성할 경우 performance_schema 파라미터는 기본적으로 1(활성화)로 설정됩니다.

성능 스키마를 활성화하지 않은 상태에서는 성능 개선 도우미에 데이터베이스 부하가 MySQL 프로세스의 복록 상태별로 구분되어 표시됩니다. 성능 스키마를 활성화한 상태에서는 성능 개선 도우미에 데이터베이스 부하가 세부 대기 이벤트별로 구분되어 표시됩니다.

자세한 내용은 [성능 개선 도우미 대시보드 사용 \(p. 372\)](#) 단원을 참조하십시오.

성능 개선 도우미의 액세스 제어

성능 개선 도우미에 액세스하려면 AWS Identity and Access Management(IAM)의 적절한 권한이 있어야 합니다. 액세스 권한은 다음 두 가지 옵션을 사용해 부여할 수 있습니다.

1. AmazonRDSFullAccess 관리 정책을 IAM 사용자 또는 역할에 연결합니다.
2. 사용자 지정 IAM 정책을 생성해 IAM 사용자 또는 역할에 연결합니다.

AmazonRDSFullAccess 관리 정책

AmazonRDSFullAccess는 모든 Amazon RDS API 작업에 대한 액세스 권한을 부여하는 AWS 관리 정책입니다. 이 정책은 Amazon RDS 콘솔에서 사용하는 관련 서비스(예: Amazon SNS를 사용한 이벤트 알림)에 대한 액세스 권한도 부여합니다.

또한 AmazonRDSFullAccess에는 성능 개선 도우미 사용에 필요한 모든 권한이 들어 있습니다. 이 정책을 IAM 사용자 또는 역할에 연결하면 수신자가 기타 콘솔 기능과 함께 성능 개선 도우미를 사용할 수 있습니다.

사용자 지정 IAM 정책 사용

AmazonRDSFullAccess 정책이 포함된 완전한 액세스 권한이 없는 사용자의 경우, 사용자 관리형 IAM 정책을 생성 또는 수정하여 성능 개선 도우미에 대한 액세스 권한을 부여할 수 있습니다. IAM 사용자 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 정책을 선택합니다.
3. [Create policy]를 선택합니다.
4. [Create Policy] 페이지에서 JSON 탭을 선택합니다.
5. 다음을 복사하여 붙여 넣습니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "pi:*",  
            "Resource": "arn:aws:pi:***:metrics/rds/*"  
        }  
    ]  
}
```

6. [Review policy]를 선택합니다.

Note

현재 이 정책을 입력하면 [Visual editor] 탭에 pi 리소스가 인식되지 않는다는 경고가 표시됩니다. 이 경고는 무시해도 됩니다.

- 정책의 이름과 설명(선택 사항)을 지정한 다음 [Create policy]를 선택합니다.

이제 IAM 사용자 또는 역할에 해당 정책을 연결할 수 있습니다. 다음 절차에서는 이러한 목적에 사용할 수 있는 IAM 사용자가 이미 있다고 가정합니다.

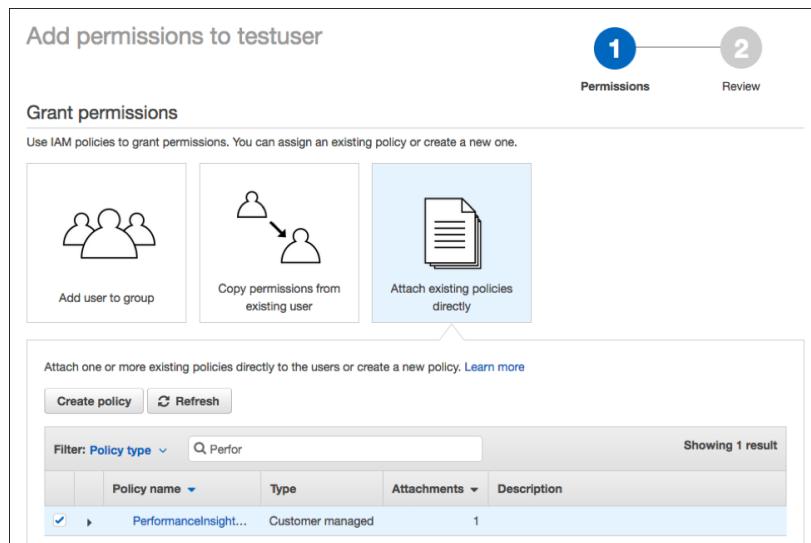
IAM 사용자에게 정책을 연결하려면

- <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
- 탐색 창에서 사용자를 선택합니다.
- 목록에서 기존 사용자를 선택합니다.

Important

사용자가 성능 개선 도우미를 사용하려면 사용자 지정 정책 외에 Amazon RDS에 대한 액세스 권한이 있어야 합니다. 예를 들어 `AmazonRDSReadOnlyAccess` 사전 정의 정책은 Amazon RDS에 대한 읽기 전용 액세스를 제공합니다. 자세한 내용은 [정책을 이용한 액세스 관리 \(p. 963\)](#) 단원을 참조하십시오.

- [Summary] 페이지에서 [Add permissions]를 선택합니다.
- [Attach existing policies directly]를 선택합니다. [Search]에 다음과 같이 정책 이름의 첫 문자 몇 개를 입력합니다.



- 정책을 선택하고 [Next: Review]를 선택합니다.
- [Add permissions]를 선택합니다.

성능 개선 도우미 대시보드 사용

성능 개선 도우미 대시보드에는 성능 문제를 분석하여 해결할 수 있는 데이터베이스 성능 정보가 포함됩니다. 메인 대시보드 페이지에서 데이터베이스 부하에 대한 정보를 확인할 수 있습니다. 또한 특정 대기 상태, SQL 쿼리, 호스트 또는 사용자에 대한 세부 정보를 알아볼 수 있습니다.

주제

- [성능 개선 도우미 대시보드 열기 \(p. 373\)](#)
- [성능 개선 도우미 대시보드 구성 요소 \(p. 375\)](#)
- [성능 개선 도우미 대시보드를 사용한 데이터베이스 부하 분석 \(p. 379\)](#)
- [실행 중인 쿼리에 대한 통계 분석 \(p. 380\)](#)

- 성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트 보기 (p. 383)

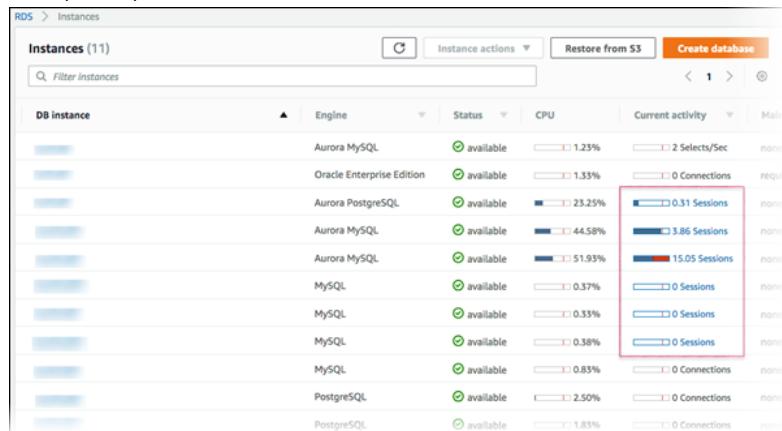
성능 개선 도우미 대시보드 열기

성능 개선 도우미 대시보드는 다음 절차에 따라 확인할 수 있습니다.

AWS Management Console에서 성능 개선 도우미 대시보드를 보려면

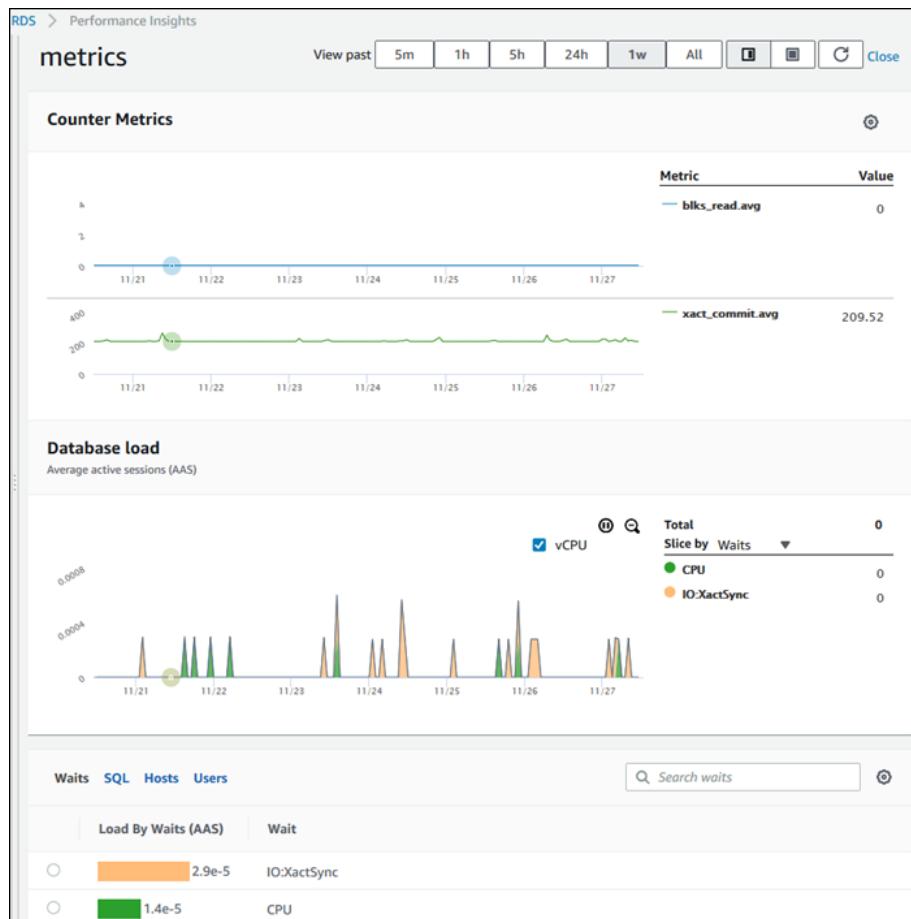
1. <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [성능 개선 도우미]를 선택합니다.
3. DB 인스턴스를 선택합니다. 선택한 DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.

성능 개선 도우미가 활성화되면 DB 인스턴스의 경우 DB 인스턴스 목록에서 세션 항목을 선택하여 대시보드에 접속할 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 유 휴 상태입니다. 로드가 증가하면 막대가 파란색으로 채워집니다. 로드에서 DB 인스턴스 클래스의 가상 CPU(vCPU) 수를 전달하면 막대가 빨간색으로 바뀌고 병목 가능성을 나타냅니다.



다음 스크린샷은 DB 인스턴스의 대시보드입니다.

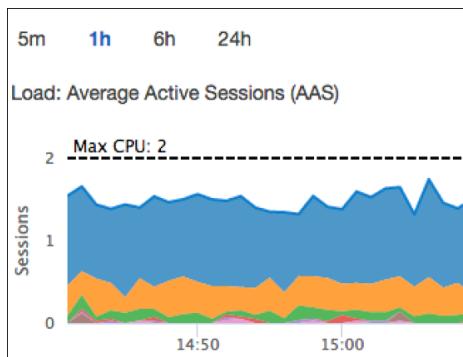
Amazon Aurora Aurora 사용 설명서 성능 개선 도우미 대시보드 사용



성능 개선 도우미 대시보드는 기본적으로 마지막 60분 동안 수집된 데이터를 표시합니다. 하지만 마지막 5분, 60분, 5시간, 24시간 또는 1주 동안 데이터를 표시하도록 설정할 수 있습니다. 사용 가능한 모든 데이터를 볼 수도 있습니다.

Performance Insight 대시보드는 새 데이터로 자동으로 고쳐집니다. 새로 고침 속도는 표시되는 데이터의 양에 따라 다릅니다.

- 5분은 5초마다 새로 고칩니다.
- 1시간 및 5시간은 1분마다 새로 고칩니다.
- 24시간은 5분마다 새로 고칩니다.
- 1주는 1시간마다 새로 고칩니다.



성능 개선 도우미 대시보드 구성 요소

대시보드는 세 부분으로 나뉩니다.

1. 카운터 지표 차트 – 특정 성능 카운터 지표의 데이터를 보여 줍니다.
2. 평균 활성 세션 차트 – 데이터베이스 부하와 DB 인스턴스 용량을 비교하여 최대 CPU 선으로 표시합니다.
3. 상위 부하 항목 테이블 – 데이터베이스 부하에 가장 많이 영향을 미치는 항목을 보여 줍니다.

카운터 지표 차트

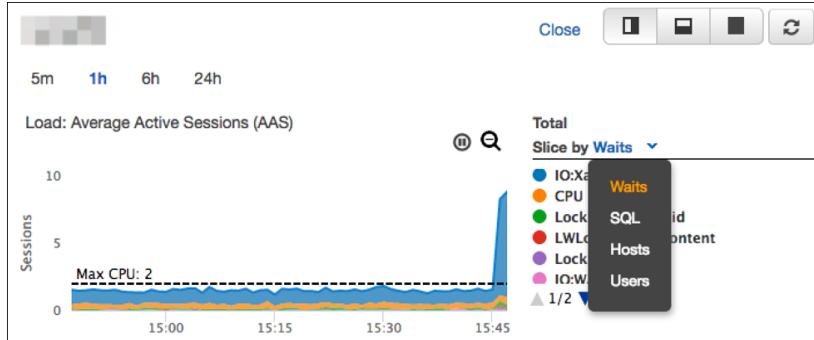
카운터 지표 차트에는 성능 카운터의 데이터가 표시됩니다. 표시되는 기본 지표는 `blk.read.avg` 및 `xact.commit.avg`입니다. 차트 오른쪽 상단 모서리에 있는 기어 모양 아이콘을 선택하여 표시할 성능 카운터를 선택하십시오.



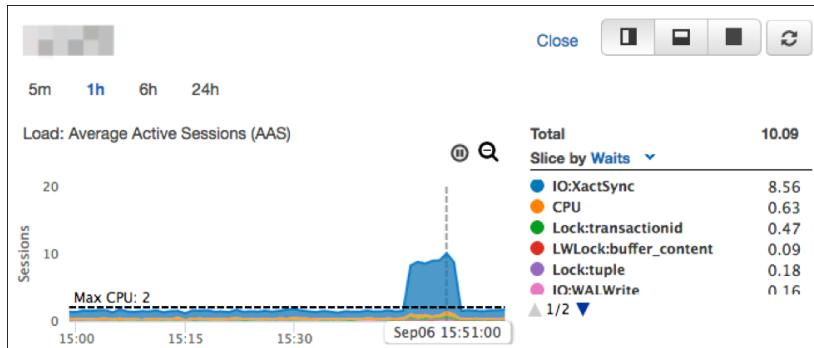
자세한 내용은 [성능 개선 도우미 카운터 \(p. 400\)](#) 단원을 참조하십시오.

평균 활성 세션 차트

평균 활성 세션 차트는 데이터베이스 부하와 DB 인스턴스 용량의 비교 방식을 최대 CPU 선으로 표시합니다. 기본적으로 부하는 대기 상태를 기준으로 구분된 활성 세션으로 표시됩니다. 또한 SQL 쿼리, 호스트 또는 사용자로 구분된 활성 세션으로 부하를 표시하도록 선택할 수도 있습니다.

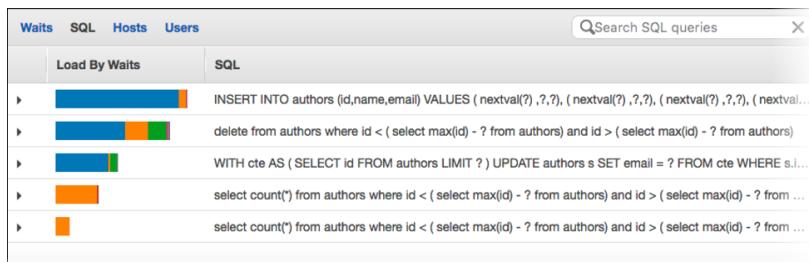


선택한 기간의 항목 세부 정보를 범례에 표시하려면 평균 활성 세션 차트의 임의 항목 위로 마우스 포인터를 가져가면 됩니다.



상위 부하 항목 테이블

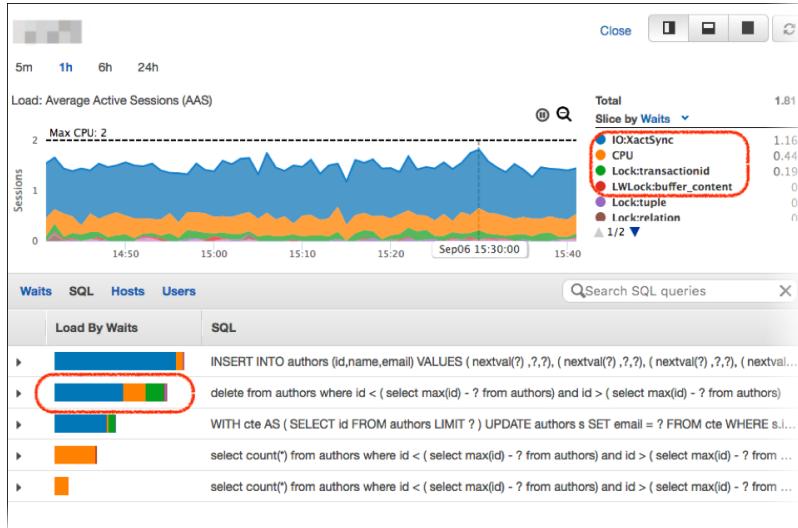
[Top Load Items] 테이블에는 데이터베이스 부하에 영향을 끼치는 상위 항목이 표시됩니다. 기본적으로 데이터베이스 부하에 영향을 미치는 상위 SQL 쿼리가 표시됩니다. 쿼리는 구조적으로 유사한 실제 쿼리가 다수 요약되어 표시되지만 다른 파라미터가 존재할 가능성도 있습니다. 그 밖에 최상위 대기 상태, 호스트 또는 사용자를 표시하도록 선택할 수도 있습니다.



각 상위 로드 항목과 연결된 데이터베이스 로드 비율(%)은 DB Load by Waits(대기별 DB 로드) 열에 표시됩니다. 이 열에는 현재 평균 활성 세션 차트에서 어떤 구분 기준을 선택하는 그 기준에 따라 해당 항목의 로드가 반영됩니다. 예를 들어, 평균 활성 세션 차트가 호스트별로 구분되어 있고 상위 부하 항목 테이블에서 SQL 쿼리를 살펴보고 있다고 가정합니다. 이 경우 DB Load by Waits(대기별 DB 로드) 막대는 관련 호스트에서 쿼리가 나타내는 로드를 반영합니다. 평균 활성 세션 차트의 해당 호스트에 컬러 코드로 매핑합니다.

또 다른 예로, 평균 활성 세션 차트가 대기 상태별로 구분되어 있고 상위 부하 항목 테이블에서 SQL 쿼리를 살펴보고 있다고 가정합니다. 이 경우 DB Load by Waits(대기별 DB 로드) 막대는 쿼리가 영향을 미치는 대

기상태의 정도를 크기, 세그먼트 및 컬러 코드로 표시합니다. 해당 쿼리에 영향을 미치는 대기 상태도 표시합니다.



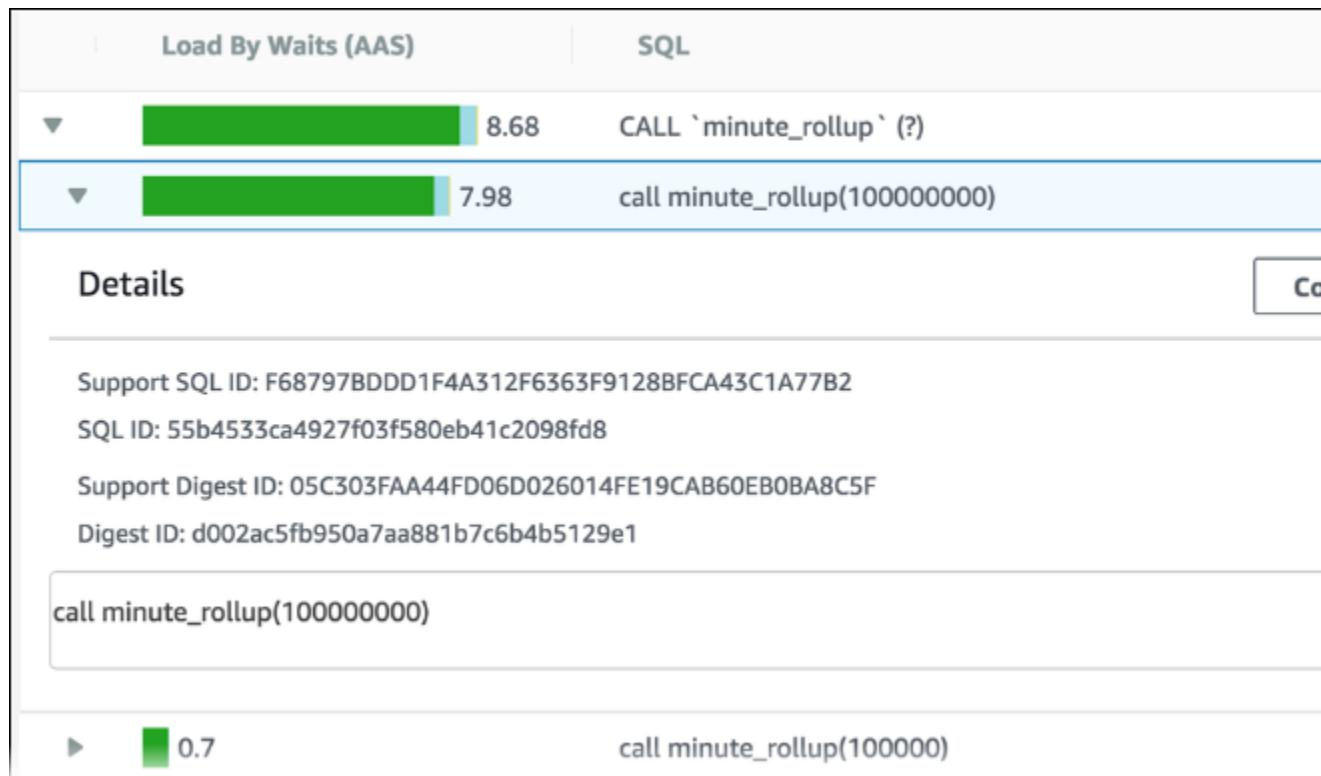
Top Load Items(상위 부하 항목) 테이블에서 SQL 문과 관련된 다음과 같은 식별자(ID) 유형을 볼 수 있습니다.

- SQL ID – 데이터베이스가 SQL 문을 고유하게 식별하기 위해 사용하는 ID입니다.
- SQL ID 지원 – SQL ID의 해시 값입니다. 이 값은 AWS Support를 사용할 때 SQL ID를 참조하는 용도로만 사용됩니다. AWS Support는 실제 SQL ID 및 SQL 텍스트에 액세스할 수 없습니다.
- 다이제스트 ID – 데이터베이스가 SQL 다이제스트를 고유하게 식별하기 위해 사용하는 ID입니다. SQL 다이제스트에는 리터럴이 제거되고 공백이 표준화된 SQL 문이 하나 이상 포함될 수 있습니다. 리터럴은 둘 음표(?)로 대체됩니다.

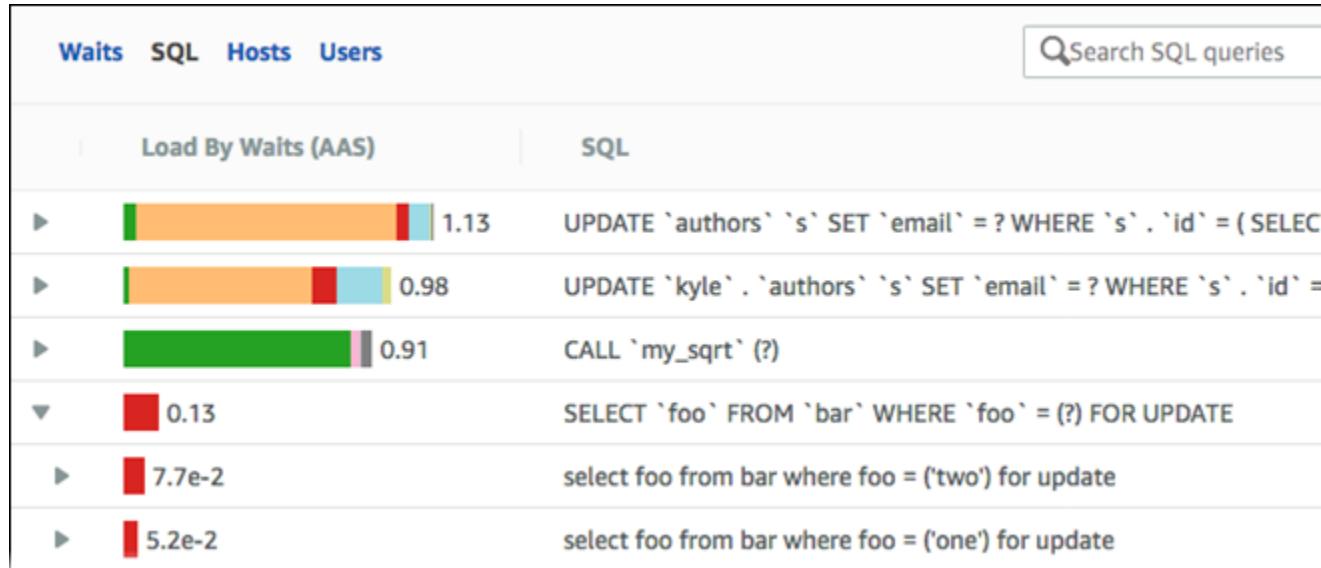
Aurora MySQL 및 Aurora PostgreSQL DB 인스턴스의 경우, 다이제스트 ID를 사용하여 특정 SQL 다이제스트를 찾을 수 있습니다.

- Support Digest ID(다이제스트 ID 지원) – 다이제스트 ID의 해시 값입니다. 이 값은 AWS Support를 사용할 때 다이제스트 ID를 참조하는 용도로만 사용됩니다. AWS Support는 실제 다이제스트 ID 및 SQL 텍스트에 액세스할 수 있는 권한이 없습니다.

Top Load Items(상위 부하 항목) 테이블에서 상위 문을 열어 ID를 볼 수 있습니다. 다음은 열린 상위 문의 스크린샷입니다.



기본 설정 아이콘을 선택하여 Top Load Items(상위 부하 항목) 테이블에 표시된 ID를 제어할 수 있습니다.



기본 설정 아이콘을 선택하면 기본 설정 창이 열립니다.



Top Load Items(상위 부하 항목) 테이블에 표시하려는 ID를 활성화한 후, 저장을 선택합니다.

성능 개선 도우미 대시보드를 사용한 데이터베이스 부하 분석

평균 활성 세션 차트가 병목 현상을 보일 때는 부하가 발생하는 위치를 찾아낼 수 있습니다. 이렇게 하려면 평균 활성 세션 차트 아래에 있는 상위 부하 항목 테이블을 살펴봅니다. 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.

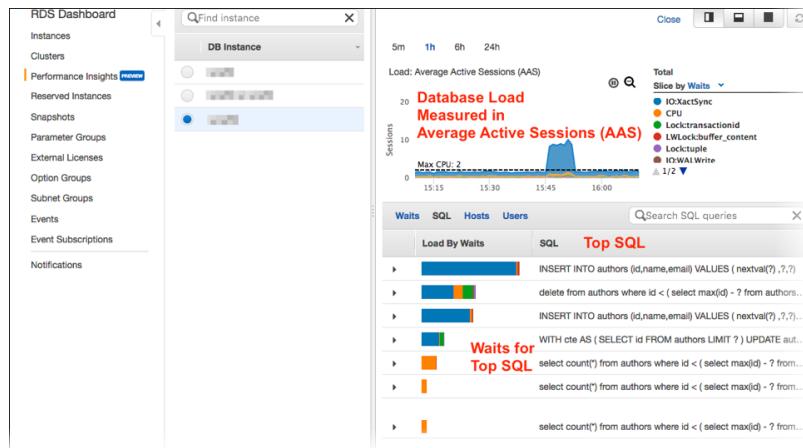
성능 문제를 진단하는 일반 워크플로우는 다음과 같습니다.

1. 평균 활성 세션 차트를 보면서 데이터베이스 부하가 최대 CPU 선을 상회하는지 모니터링합니다.
2. 상회하는 경우가 있으면 평균 활성 세션 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 IO:XactSync 대기 시간은 빈번한 문제입니다. [CPU] 대기 시간은 비교적 빈번하지는 않지만 부하에 여전히 중요한 역할을 합니다.

상위 부하 항목 테이블의 [SQL] 탭에서 처음 4개의 룰업 쿼리는 첫 번째 상태와 매우 밀접한 관계가 있습니다. 따라서 이러한 쿼리의 하위 쿼리까지 세부 정보를 확인해야 합니다. 이러한 쿼리가 성능 문제에 어떻게 영향을 끼치고 있는지 알아보려면 이렇게 확인해야 합니다.

마지막 3개의 룰업 쿼리는 CPU 부하에 중요한 역할을 하며, CPU 부하가 문제인지 여부를 조사하는 쿼리가 됩니다.



실행 중인 쿼리에 대한 통계 분석

Amazon RDS 성능 개선 도우미의 Top Load Items(상위 로드 항목) 섹션에서 실행 중인 쿼리에 대한 통계를 검색할 수 있습니다. 이 통계를 보려면 상위 SQL을 확인하십시오. 성능 개선 도우미는 가장 흔한 쿼리에 대해서만 통계를 수집합니다. 이러한 쿼리는 대개 성능 개선 도우미 대시보드에 표시된 로드별 상위 쿼리에 해당합니다.

주제

- [Aurora PostgreSQL에 대한 SQL 다이제스트 통계 \(p. 380\)](#)
- [실행 중인 SQL 문의 Aurora 지표 분석 \(p. 382\)](#)

Aurora PostgreSQL에 대한 SQL 다이제스트 통계

SQL 다이제스트 통계를 보려면 `pg_stat_statements` 라이브러리를 로드해야 합니다. 이 라이브러리는 PostgreSQL 10과 호환되는 Aurora PostgreSQL DB 클러스터에 대해 기본적으로 로드됩니다. 그러나 이 라이브러리를 PostgreSQL 9.6과 호환되는 Aurora PostgreSQL DB 클러스터에 대해 수동으로 활성화해야 합니다. 이 라이브러리를 수동으로 활성화하려면 DB 인스턴스와 연결된 DB 파라미터 그룹의 `shared_preload_libraries`에 `pg_stat_statements`를 추가하십시오. 그런 다음 DB 인스턴스를 재부팅합니다. 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

Note

성능 개선 도우미는 `pg_stat_activity`에서 잘리지 않은 쿼리에 대한 통계만 수집할 수 있습니다. 기본적으로 Aurora PostgreSQL 데이터베이스는 1,024바이트보다 긴 쿼리를 자릅니다. DB 인스턴스와 연결된 DB 파라미터 그룹에서 `track_activity_query_size` 파라미터를 변경하여 쿼리 크기를 늘릴 수 있습니다. 이 파라미터를 변경하면 DB 인스턴스를 재부팅해야 합니다. 성능 개선 도우미에는 쿼리를 수집할 때 5,120이트 제한이 있으며, 이는 통계 수집에도 영향을 미칩니다. 메모리 제한으로 인해 5,120바이트 제한이 필요합니다.

다음 SQL 다이제스트 통계는 Aurora PostgreSQL DB 인스턴스에 제공됩니다.

지표	Unit
<code>db.sql_tokenized.stats.calls_per_sec</code>	초당 호출 수
<code>db.sql_tokenized.stats.rows_per_sec</code>	초당 행
<code>db.sql_tokenized.stats.total_time_per_sec</code>	초당 평균 활성 실행(AAE)

지표	Unit
db.sql_tokenized.stats.shared_blk_hit_per_sec	초당 대량 히트 수
db.sql_tokenized.stats.shared_blk_read_per_sec	초당 대량 읽기 수
db.sql_tokenized.stats.shared_blk_dirtied_per_sec	초당 대량 더티 수
db.sql_tokenized.stats.shared_blk_written_per_sec	초당 대량 쓰기 수
db.sql_tokenized.stats.local_blk_hit_per_sec	초당 로컬 대량 히트 수
db.sql_tokenized.stats.local_blk_read_per_sec	초당 로컬 대량 읽기 수
db.sql_tokenized.stats.local_blk_dirtied_per_sec	초당 로컬 대량 더티 수
db.sql_tokenized.stats.local_blk_written_per_sec	초당 로컬 대량 쓰기 수
db.sql_tokenized.stats.temp_blk_written_per_sec	초당 임시 쓰기 수
db.sql_tokenized.stats.temp_blk_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

다음 지표에서는 SQL 문의 호출당 통계를 제공합니다.

지표	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_blk_hit_per_call	호출당 대량 히트 수
db.sql_tokenized.stats.shared_blk_read_per_call	호출당 대량 읽기 수
db.sql_tokenized.stats.shared_blk_written_per_call	호출당 대량 쓰기 수
db.sql_tokenized.stats.shared_blk_dirtied_per_call	호출당 대량 더티 수
db.sql_tokenized.stats.local_blk_hit_per_call	호출당 로컬 대량 히트 수
db.sql_tokenized.stats.local_blk_read_per_call	호출당 로컬 대량 읽기 수
db.sql_tokenized.stats.local_blk_dirtied_per_call	호출당 로컬 대량 더티 수
db.sql_tokenized.stats.local_blk_written_per_call	호출당 로컬 대량 쓰기 수
db.sql_tokenized.stats.temp_blk_written_per_call	호출당 임시 대량 쓰기 수
db.sql_tokenized.stats.temp_blk_read_per_call	호출당 임시 대량 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)

이러한 지표에 대한 자세한 내용은 PostgreSQL 설명서의 [pg_stat_statements](#)를 참조하십시오.

실행 중인 SQL 문의 Aurora 지표 분석

AWS Management 콘솔에서 SQL 탭을 선택하여 실행 중인 SQL 쿼리의 지표를 확인할 수 있습니다.

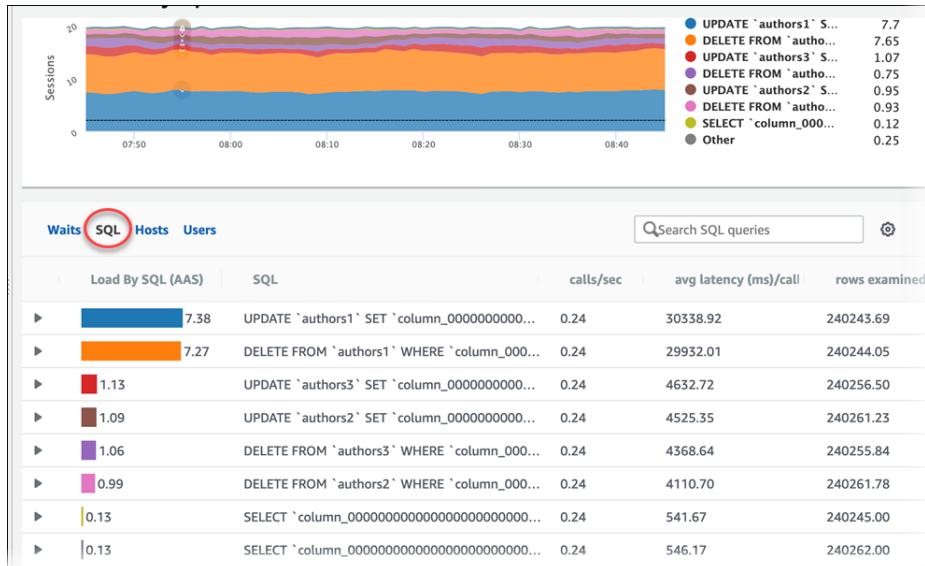
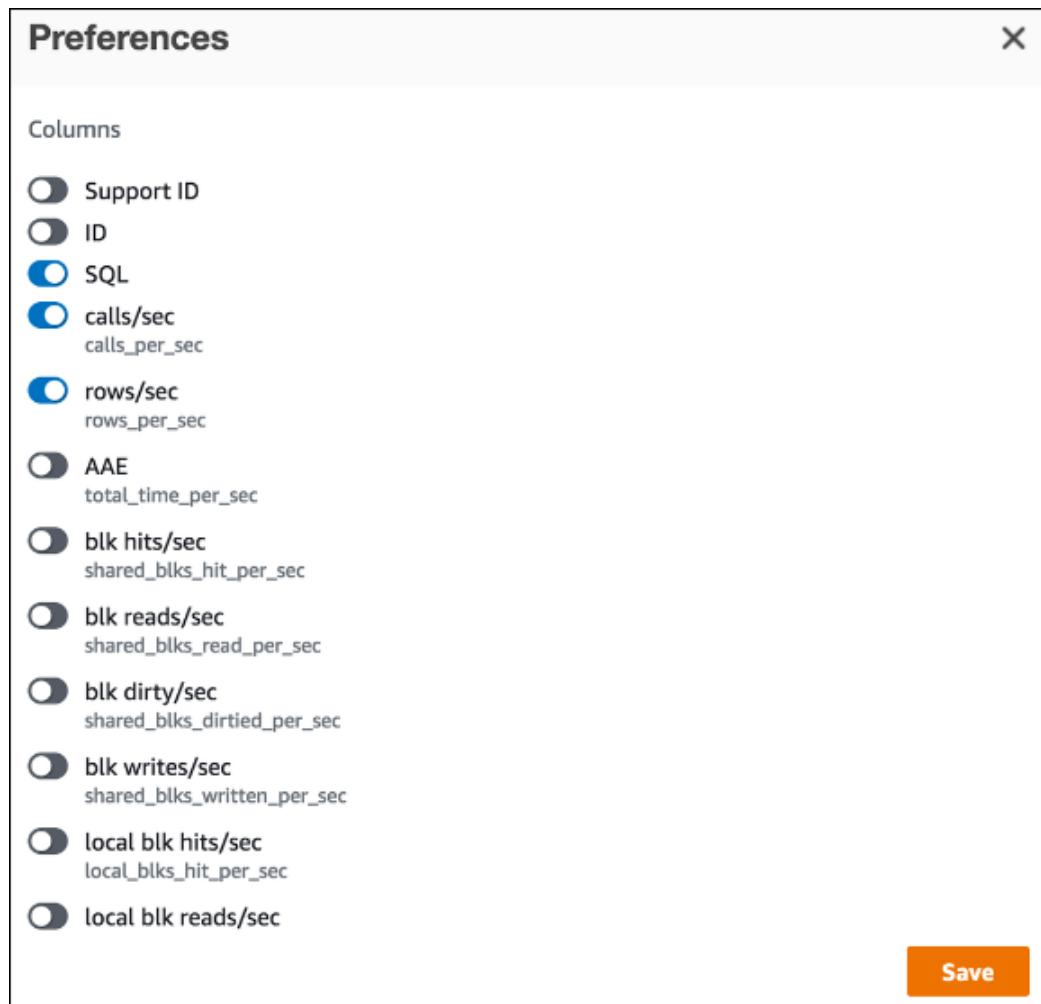


차트 오른쪽 상단 모서리에 있는 기어 모양 아이콘을 선택하여 표시할 통계를 선택하십시오.

다음 스크린샷은 Aurora PostgreSQL의 기본 설정을 보여줍니다.



성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트 보기

기본적으로 Top Load Items(상위 부하 항목) 표의 각 행에는 각 SQL 문에 대해 500바이트의 SQL 텍스트가 표시됩니다. SQL 문이 500바이트 이상인 경우 성능 개선 도우미 대시보드에서 해당 문을 열어 더 많은 SQL 문을 볼 수 있습니다. 성능 개선 도우미 대시보드는 SQL 문 하나에 최대 4,096바이트를 표시할 수 있습니다. 표시되는 SQL 문을 복사할 수 있습니다. 4,096바이트보다 큰 텍스트를 보려면 Download full SQL(전체 SQL 다운로드)을 선택하십시오. 그러면 DB 엔진 최대 크기까지 SQL 텍스트를 볼 수 있습니다.

SQL 텍스트의 한도는 DB 엔진에 달려 있습니다. 다음과 같은 제한이 적용됩니다.

- Aurora MySQL 5.7 – 4,096바이트
- Aurora MySQL 5.6 – 1,024바이트
- Aurora PostgreSQL – `track_activity_query_size` DB 인스턴스 파라미터로 설정

Aurora PostgreSQL DB 인스턴스의 경우, `track_activity_query_size` DB 인스턴스 파라미터를 최대 102,400바이트까지 설정하여 SQL 텍스트의 한도를 제어할 수 있습니다. AWS Management 콘솔을 사용해 이 파라미터로 설정한 한도까지 SQL 텍스트를 다운로드할 수 있습니다. 자세한 내용은 [Aurora PostgreSQL DB 인스턴스에 대한 SQL 텍스트 한도 설정 \(p. 385\)](#) 단원을 참조하십시오.

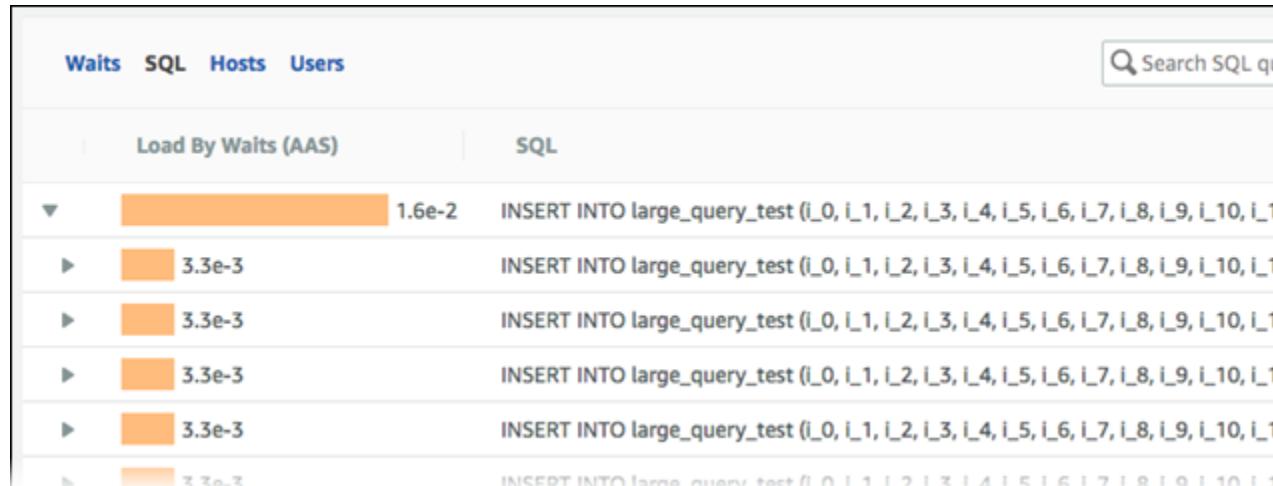
Important

현재는 AWS Management 콘솔에서 더 많은 SQL 텍스트를 보고 다운로드할 수만 있습니다. AWS 성능 개선 도우미 CLI 및 API는 최대 500바이트의 텍스트를 반환할 수 있습니다.

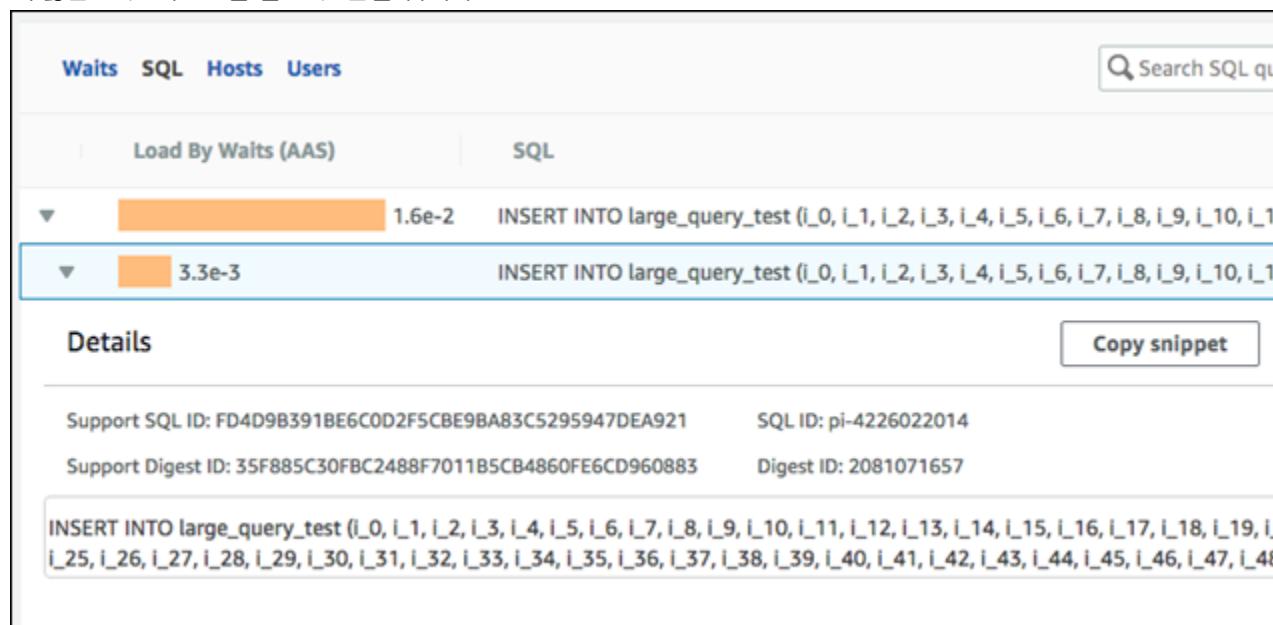
성능 개선 도우미 대시보드에서 더 많은 SQL 텍스트를 보려면

1. <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
 2. 탐색 창에서 [성능 개선 도우미]를 선택합니다.
 3. DB 인스턴스를 선택합니다. 선택한 DB 인스턴스에 대한 성능 개선 도우미 대시보드가 표시됩니다.

500바이트 이상의 텍스트가 있는 SQL 문은 다음 이미지와 유사합니다.



- #### 4. 더 많은 SQL 텍스트를 볼 SQL 문을 엽니다.



성능 개선 도우미 대시보드는 각 SQL 문에 최대 4,096바이트를 표시할 수 있습니다.

5. (선택 사항) Copy snippet(코드 조각 복사)을 선택하여 표시된 SQL 문을 복사하거나 Download full SQL(전체 SQL 다운로드)를 선택하여 DB 엔진 최대 크기까지 SQL 텍스트를 볼 수 있는 SQL 문을 다운로드합니다.

Note

SQL 문을 복사하거나 다운로드하려면 팝업 차단 기능을 비활성화하십시오.

Aurora PostgreSQL DB 인스턴스에 대한 SQL 텍스트 한도 설정

Aurora PostgreSQL DB 인스턴스의 경우, 성능 개선 도우미 대시보드에 표시될 수 있는 SQL 텍스트의 한도를 제어할 수 있습니다.

이를 위해서는 `track_activity_query_size` DB 인스턴스 파라미터를 수정해야 합니다. Aurora PostgreSQL 버전 9.6.0에서 `track_activity_query_size` 파라미터에 대한 기본 설정은 1,024바이트입니다. Aurora PostgreSQL 10 이상 버전에서 `track_activity_query_size` 파라미터에 대한 기본 설정은 4096바이트입니다.

바이트 수를 늘려 성능 개선 도우미 대시보드에서 볼 수 있는 SQL 텍스트 크기를 늘릴 수 있습니다. 파라미터의 한도는 10,240바이트입니다. `track_activity_query_size` DB 파라미터에 대한 자세한 내용은 PostgreSQL 설명서에서 [런타임 통계](#)를 참조하십시오.

파라미터를 수정하려면 Aurora PostgreSQL DB 인스턴스와 연결된 파라미터 그룹에서 파라미터 설정을 변경하십시오.

Aurora PostgreSQL DB 인스턴스에서 기본 파라미터 그룹을 사용 중인 경우, 다음 절차를 완료하십시오.

- 적절한 DB 엔진 및 DB 엔진 버전에 대해 새로운 DB 인스턴스 파라미터 그룹을 생성합니다.
- 새 파라미터 그룹에 파라미터를 설정합니다.
- 새 파라미터 그룹을 DB 인스턴스에 연결합니다.

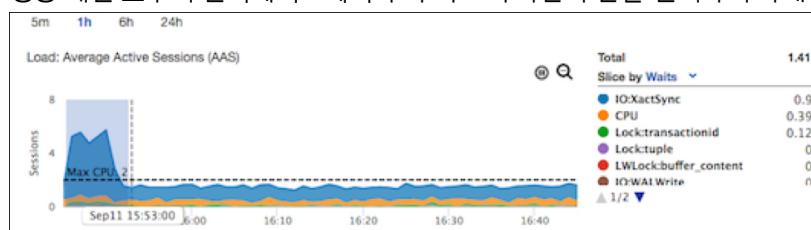
DB 인스턴스 파라미터 설정에 대한 자세한 내용은 [DB 파라미터 그룹의 파라미터 수정 \(p. 174\)](#) 단원을 참조하십시오.

추가 사용자 인터페이스 기능

성능 개선 도우미 사용자 인터페이스의 다른 기능을 사용해 성능 데이터를 분석할 수 있습니다.

클릭하여 끌어 확대

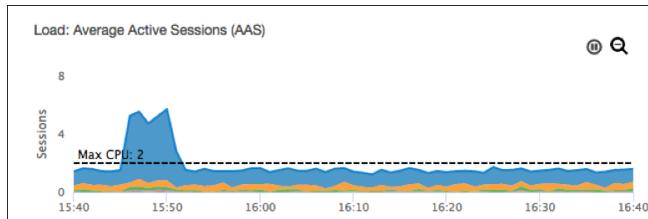
성능 개선 도우미 인터페이스에서 부하 차트의 작은 부분을 선택하여 확대해 자세히 볼 수 있습니다.



부하 차트의 한 부분을 확대하려면 시작 시간을 선택하고 원하는 기간 끝까지 끕니다. 이렇게 하면 선택한 영역이 강조 표시됩니다. 마우스를 놓으면 부하 차트의 선택한 영역이 확대되고 [Top N] 테이블이 다시 계산됩니다.

일시 중지 및 축소

부하 차트의 오른쪽 맨 위에는 [Pause] 및 [Zoom out] 도구가 있습니다.



[Pause]를 선택하면 부하 차트가 자동 새로 고침을 중지합니다. [Pause]를 다시 선택하면 부하 차트가 자동 새로 고침을 다시 시작합니다.

[Zoom out]을 선택하면 부하 차트가 다음으로 가장 큰 시간 간격으로 축소됩니다.

성능 개선 도우미 API

지원되는 엔진 유형에 대해 성능 개선 도우미가 활성화된 경우 Amazon RDS 성능 개선 도우미 API는 RDS 인스턴스의 성능에 대한 가시성을 제공합니다. Amazon CloudWatch Logs는 AWS 서비스에 대해 판매된 모니터링 지표를 위한 신뢰할 수 있는 소스를 제공합니다. 성능 개선 도우미는 평균 활성 세션 수로 측정되어 API 소비자에게 2D 시계열 데이터 세트로 제공되는 도메인별 데이터베이스 부하 보기를 제공합니다. 데이터의 시간 차원은 쿼리된 시간 범위 내 각 시점에 대한 데이터베이스 부하 데이터를 제공합니다. 각 시점에서는 요청된 차원에 관해 해당 시점에서 측정되는 전체 부하를 분해합니다(예: SQL, Wait-event, User 또는 Host).

Amazon RDS 성능 개선 도우미는 데이터베이스 성능을 분석하고 관련 문제를 해결할 수 있도록 Amazon RDS DB 인스턴스를 모니터링합니다. 성능 개선 도우미 데이터를 볼 수 있는 한 가지 방법은 AWS Management 콘솔에서 보는 것입니다. 또한 성능 개선 도우미는 사용자가 자신의 데이터를 쿼리할 수 있도록 퍼블릭 API도 제공합니다. API는 데이터를 데이터베이스에 오프로드하거나 성능 개선 도우미 데이터를 기준 모니터링 대시보드에 추가하거나 모니터링 도구를 빌드하는 데 사용할 수 있습니다. 성능 개선 도우미 API를 사용하려면 Amazon RDS DB 인스턴스 중 하나에서 성능 개선 도우미를 활성화하십시오. 성능 개선 도우미 활성화에 대한 자세한 내용은 [성능 개선 도우미 활성화 \(p. 367\)](#) 단원을 참조하십시오.

성능 개선 도우미 API에서는 다음과 같은 작업을 제공합니다.

성능 개선 도우미 작업	AWS CLI 명령	설명
	<code>aws pi describe-dimension-keys</code>	특정 기간에 대해 지표의 상위 N개 차원 키를 검색합니다.
	<code>aws pi get-resource-metrics</code>	일정 기간 동안 데이터 소스 집합에 대한 성능 개선 도우미 지표를 검색합니다. 특정 차원 그룹 및 차원을 제공하고 각 그룹에 집계 및 필터링 기준을 제공할 수 있습니다.

성능 개선 도우미 API에 대한 자세한 내용은 [Amazon RDS 성능 개선 도우미 API 참조](#)를 참조하십시오.

성능 개선 도우미용 AWS CLI

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.

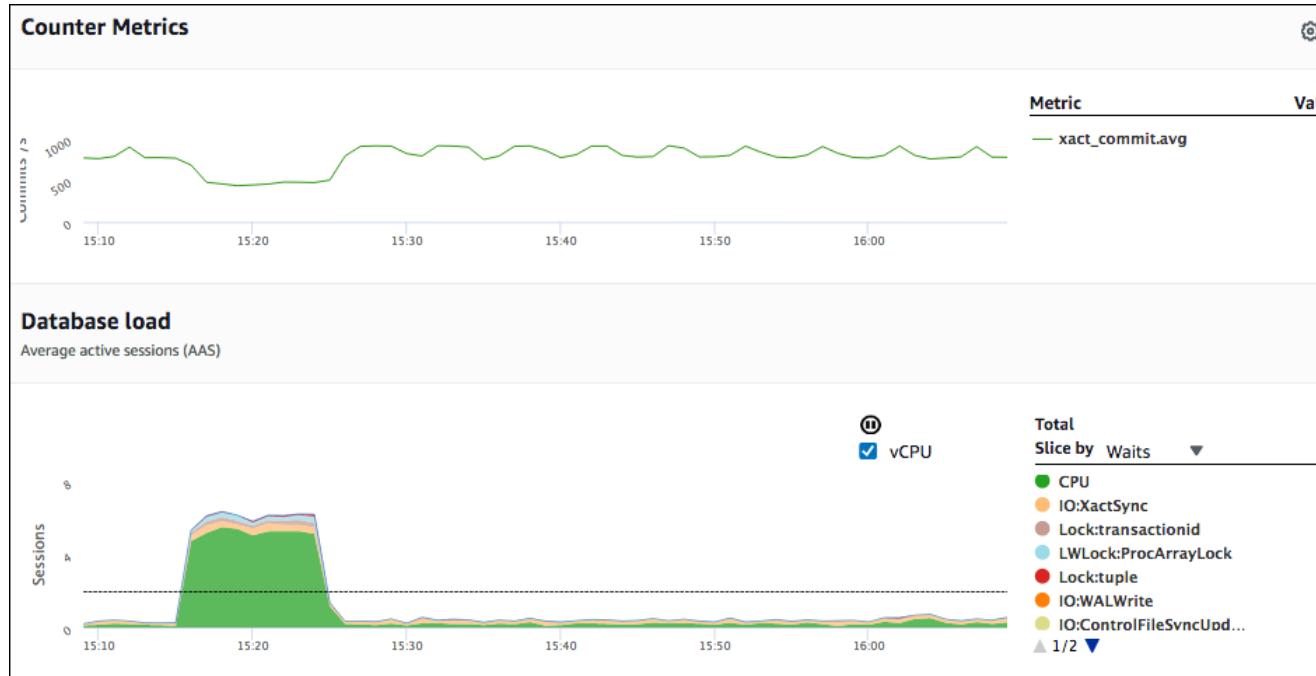
```
aws pi help
```

AWS CLI가 설치되어 있지 않은 경우 설치에 관한 자세한 정보는 AWS CLI 사용 설명서의 [AWS 명령줄 인터페이스 설치](#)를 참조하십시오.

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 콘솔에서는 성능 개선 도우미 대시보드의 두 곳에서 `GetResourceMetrics`를 사용합니다. `GetResourceMetrics`는 Counter Metrics(카운터 지표) 차트를 입력하는 데 사용되고 Database Load(데이터베이스 로드)에서는 다음 이미지처럼 보입니다.



`GetResourceMetrics`에서 반환하는 모든 지표는 표준 시계열 지표이며 한 가지 예외가 있습니다. 그 예외는 성능 개선 도우미의 핵심 지표인 `db.load`입니다. 이 지표는 Database Load(데이터베이스 부하) 차트에 표시됩니다. `db.load` 지표는 '차원'이라는 하위 구성 요소로 구분할 수 있기 때문에 다른 시계열 지표와 다릅니다. 앞의 이미지에서 `db.load`는 `db.load`를 구성하는 대기 상태에 따라 구분되고 그룹화됩니다.

Note

`GetResourceMetrics`에서는 `db.sampleload`도 반환할 수 있지만 `db.load` 지표는 대부분의 경우 적절합니다.

`GetResourceMetrics`에서 반환하는 카운터 지표에 대한 자세한 내용은 [성능 개선 도우미 카운터 \(p. 400\)](#)를 참조하십시오.

지표에 대해서는 다음 계산이 지원됩니다.

- 평균 - 일정 기간 동안 지표의 평균 값입니다. `.avg`를 지표 이름에 추가합니다.
- 최소 - 일정 기간 동안 지표의 최소 값입니다. `.min`을 지표 이름에 추가합니다.
- 최대 - 일정 기간 동안 지표의 최대 값입니다. `.max`을 지표 이름에 추가합니다.
- 합계 - 일정 기간 동안 지표 값의 합계입니다. `.sum`을 지표 이름에 추가합니다.
- 샘플 수 - 일정 기간 동안 지표가 수집된 횟수입니다. `.sample_count`를 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](#) 단원을 참조하십시오.

--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 사용 예입니다.

```
{  
    "Metric": "string",  
    "GroupBy": {  
        "Group": "string",  
        "Dimensions": ["string", ...],  
        "Limit": integer  
    },  
    "Filter": {"string": "string"  
              ...}  
}
```

성능 개선 도우미에 대한 AWS CLI 예시

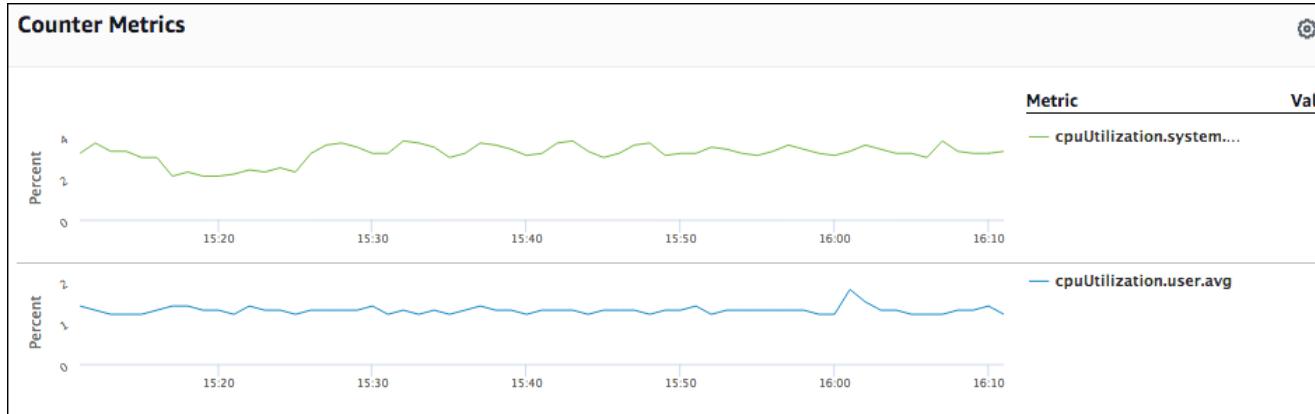
다음은 성능 개선 도우미에 대한 AWS CLI 사용을 보여주는 예입니다.

주제

- [카운터 지표 검색 \(p. 388\)](#)
- [상위 대기 이벤트에 대한 DB 평균 로드 검색 \(p. 391\)](#)
- [상위 SQL에 대한 DB 평균 로드 검색 \(p. 393\)](#)
- [SQL을 기준으로 필터링된 DB 평균 로드 검색 \(p. 395\)](#)

카운터 지표 검색

다음 스크린샷은 AWS Management 콘솔에 표시되는 카운터 지표 차트 2개를 나타낸 것입니다.



다음 예에서는 카운터 지표 차트 2개를 생성하기 위해 AWS Management 콘솔이 사용하는 것과 동일한 데이터를 수집하는 방법을 보여줍니다.

Linux, OS X, Unix의 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"}]'
```

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.

Linux, OS X, Unix의 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
```

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 쿼리의 경우 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 avg입니다. Amazon CloudWatch와 마찬가지로 지원되는 함수는 min, max, total 및 avg입니다.

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 콘솔에서 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 검색](#) (p. 388)의 명령과 동일합니다. 그러나 query.json 파일의 컨텐츠는 다음과 같습니다.

```

[
    {
        "Metric": "db.load.avg",
        "GroupBy": { "Group": "db.wait_event", "Limit": 7 }
    }
]

```

다음 명령을 실행합니다.

Linux, OS X, Unix의 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
```

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 GroupBy에 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": 0.3833333333333336
                },
                {
                    "Timestamp": 1540857780.0, //Minute 3
                    "Value": 0.2666666666666666
                }
                //... 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-select * from customers where customer_id = 123와 같은 SQL 문
- db.sql_tokenized-select * from customers where customer_id = ?와 같은 토큰화된 SQL 문

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 검색 \(p. 391\)](#)의 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.

Linux, OS X, Unix의 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
```

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 MetricList에 11개의 항목이 있는데(전체 1개, 최상위 토큰화 SQL 10개) 각 항목에는 시간당 DataPoints가 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 콘솔에서는 이 ID를 지원 ID라고 합니다. 이렇게 부르는 이유는 이 ID가 데이터베이스 관련 문제 해결을 지원하기 위해 AWS Support가 검토하는 데이터이기 때문입니다. AWS는 데이터 보안 및 개인 정보 보호를 중대 사안으로 간주하므로 거의 모든 데이터는 AWS KMS 키로 암호화되어 저장됩니다. 그러므로 AWS 내부의 어느 누구도 이 데이터를 볼 수 없습니다. 앞의 예에서 tokenized.statement와 tokenized.db_id 모두 암호화되어 저장됩니다. 데이터베이스 관련 문제가 있는 경우 AWS Support가 지원 ID를 참조하여 도움을 드릴 수 있습니다.

쿼리 시 GroupBy에서 Group을 지정하면 편리할 수 있습니다. 그러나 반환되는 데이터에 대한 더 세분화된 제어를 위해서는 차원 목록을 지정하십시오. 예를 들어 db.sql_tokenized.statement만 필요한 경우에 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 짹이 되는 세션만 고려합니다.

이 예에서 해당되는 API 쿼리는 [상위 SQL에 대한 DB 평균 로드 검색 \(p. 393\)](#) 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" }  
  }  
]
```

Linux, OS X, Unix의 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
```

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.

Amazon CloudWatch에 게시되는 성능 개선 도우미 지표

성능 개선 도우미는 Amazon CloudWatch에 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.

지표	설명
DBLoad	DB 엔진에 대한 활성 세션 수입니다. 일반적으로 사용자는 활성 세션의 평균 개수에 대한 데이터를 원합니다. 성능 개선 도우미에서 이 데이터는 db.load.avg로 쿼리됩니다.
DBLoadCPU	대기 이벤트 유형이 CPU인 활성 세션 수입니다. 성능 개선 도우미에서 이 데이터는 db.load.avg로 쿼리되며 대기 이벤트 유형인 CPU를 기준으로 필터링됩니다.

지표	설명
DBLoadNonCPU	대기 이벤트 유형이 CPU가 아닌 활성 세션 수입니다.

Note

이 지표는 DB 인스턴스에 로드가 있는 경우에만 CloudWatch에 게시됩니다.

CloudWatch 콘솔, AWS CLI 또는 CloudWatch API를 사용하여 이러한 지표를 검사할 수 있습니다.

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": "2018-07-19T21:30:00Z",
      "Unit": "None",
      "Average": 2.1
    },
    {
      "Timestamp": "2018-07-19T21:34:00Z",
      "Unit": "None",
      "Average": 1.7
    },
    {
      "Timestamp": "2018-07-19T21:35:00Z",
      "Unit": "None",
      "Average": 2.8
    },
    {
      "Timestamp": "2018-07-19T21:31:00Z",
      "Unit": "None",
      "Average": 1.5
    },
    {
      "Timestamp": "2018-07-19T21:32:00Z",
      "Unit": "None",
      "Average": 1.8
    },
    {
      "Timestamp": "2018-07-19T21:29:00Z",
      "Unit": "None",
      "Average": 3.0
    },
    {
      "Timestamp": "2018-07-19T21:33:00Z",
      "Unit": "None",
      "Average": 2.4
    }
  ],
  "Label": "DBLoad"
}
```

CloudWatch에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 [Amazon CloudWatch란 무엇입니까?](#)를 참조하십시오.

성능 개선 도우미 카운터

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 데 이터베이스 로드를 연관 지으면 성능 문제를 식별하고 분석하는 데 도움이 됩니다.

주제

- [성능 개선 도우미 운영 체제 카운터 \(p. 400\)](#)
- [Aurora MySQL용 성능 개선 도우미 카운터 \(p. 402\)](#)
- [Aurora PostgreSQL용 성능 개선 도우미 카운터 \(p. 405\)](#)

성능 개선 도우미 운영 체제 카운터

Aurora PostgreSQL용 성능 개선 도우미에서 다음 운영 체제 카운터를 사용할 수 있습니다. [CloudWatch Logs를 사용하여 Enhanced Monitoring 보기 \(p. 363\)](#)에서 이러한 지표의 정의를 확인할 수 있습니다.

카운터	Type	지표
active	메모리	os.memory.active
버퍼	메모리	os.memory.buffers
캐시됨	메모리	os.memory.cached
더티	메모리	os.memory.dirty
사용 가능	메모리	os.memory.free
hugePagesFree	메모리	os.memory.hugePagesFree
hugePagesRsvd	메모리	os.memory.hugePagesRsvd
hugePagesSize	메모리	os.memory.hugePagesSize
hugePagesSurp	메모리	os.memory.hugePagesSurp
hugePagesTotal	메모리	os.memory.hugePagesTotal
비활성	메모리	os.memory.inactive
매핑됨	메모리	os.memory.mapped
pageTables	메모리	os.memory.pageTables
슬래브	메모리	os.memory.slab
총합	메모리	os.memory.total
writeback	메모리	os.memory.writeback
게스트	cpuUtilization	os.cpuUtilization.guest
유 휴	cpuUtilization	os.cpuUtilization.idle

카운터	Type	지표
irq	cpuUtilization	os.cpuUtilization.irq
nice	cpuUtilization	os.cpuUtilization.nice
도용	cpuUtilization	os.cpuUtilization.steal
시스템	cpuUtilization	os.cpuUtilization.system
총합	cpuUtilization	os.cpuUtilization.total
사용자	cpuUtilization	os.cpuUtilization.user
대기	cpuUtilization	os.cpuUtilization.wait
avgQueueLen	diskIO	os.diskIO.avgQueueLen
avgReqSz	diskIO	os.diskIO.avgReqSz
await	diskIO	os.diskIO.await
readIOsPS	diskIO	os.diskIO.readIOsPS
readKb	diskIO	os.diskIO.readKb
readKbPS	diskIO	os.diskIO.readKbPS
rrqmPS	diskIO	os.diskIO.rrqmPS
tps	diskIO	os.diskIO.tps
util	diskIO	os.diskIO.util
writelOsPS	diskIO	os.diskIO.writelOsPS
writeKb	diskIO	os.diskIO.writeKb
writeKbPS	diskIO	os.diskIO.writeKbPS
wrqmPS	diskIO	os.diskIO.wrqmPS
차단됨	작업	os.tasks.blocked
실행 중	작업	os.tasks.running
절전	작업	os.tasks.sleeping
중단됨	작업	os.tasks.stopped
총합	작업	os.tasks.total
좀비	작업	os.tasks.zombie
1	loadAverageMinute	os.loadAverageMinute.one
15	loadAverageMinute	os.loadAverageMinute.fifteen
5	loadAverageMinute	os.loadAverageMinute.five
캐시됨	스왑	os.swap.cached
사용 가능	스왑	os.swap.free

카운터	Type	지표
in	스왑	os.swap.in
out	스왑	os.swap.out
총합	스왑	os.swap.total
maxFiles	fileSys	os.fileSys.maxFiles
usedFiles	fileSys	os.fileSys.usedFiles
usedFilePercent	fileSys	os.fileSys.usedFilePercent
usedPercent	fileSys	os.fileSys.usedPercent
used	fileSys	os.fileSys.used
총합	fileSys	os.fileSys.total
rx	network	os.network.rx
tx	network	os.network.tx
numVCpus	general	os.general.numVCpus

Aurora MySQL용 성능 개선 도우미 카운터

Aurora MySQL용 성능 개선 도우미에서는 다음 데이터베이스 카운터를 사용할 수 있습니다.

주제

- [Aurora MySQL용 기본 카운터 \(p. 402\)](#)
- [기본이 아닌 Aurora MySQL용 카운터 \(p. 403\)](#)

Aurora MySQL용 기본 카운터

MySQL 설명서의 [서버 상태 변수](#)에서 이러한 기본 카운터에 대한 정의를 확인하실 수 있습니다.

카운터	Type	Unit	지표
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
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
질문	SQL	초당 쿼리 수	db.SQL.Questions
Select_full_join	SQL	초당 쿼리 수	db.SQL.Select_full_join

카운터	Type	Unit	지표
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
Sort_scan	SQL	초당 쿼리 수	db.SQL.Sort_scan
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
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
Innodb_buffer_pool_pages_data	Cache	페이지	db.Cache.Innodb_buffer_pool_pages_data
Innodb_buffer_pool_pages_free	Cache	페이지	db.Cache.Innodb_buffer_pool_pages_free
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	Type	지표	설명	정의
innodb_buffer_pool_hits	Count	db.Cache.innodb_buffer_pool_hits	InnoDB 버퍼풀에서 총족할 수 있었던 읽기의 수입니다.	innodb_buffer_pool_read_requests - innodb_buffer_pool_reads
innodb_buffer_pool_hit_rate	Count	db.Cache.innodb_buffer_pool_hit_rate	InnoDB가 총족할 수 있었던 읽기의 비율입니다.	100 * innodb_buffer_pool_read_requests / (innodb_buffer_pool_read_requests + innodb_buffer_pool_reads)
innodb_buffer_pool_usage	Count	db.Cache.innodb_buffer_pool_usage	InnoDB 버퍼풀의 비율입니다. Note 압축된 테이블을 사용하는 경우 이 값은 달라질 수 있습니다. 자세한 내용은 MySQL 설명서에서 서버 상태 변수 의 Innodb_buffer_pool_pages_data 및 Innodb_buffer_pool_pages_total에 대한 정보를 참조하십시오.	Innodb_buffer_pool_pages_data / Innodb_buffer_pool_pages_total * 100.0
query_cache_hits	Count	db.Cache.query_cache_hits	MySQL 결과값 캐시(쿼리 캐시)의 적중률입니다.	Qcache_hits / (Qcache_hits + Com_select) * 100
innodb_rows_inserted	Count	db.SQL.innodb_rows_inserted	InnoDB 행 추가 연산입니다.	db.SQL.Innodb_rows_inserted + db.SQL.Innodb_rows_deleted + db.SQL.Innodb_rows_updated
active_transactions	Count	db.Transaction.active_transactions	활성 트랜잭션입니다.	SELECT COUNT(1) AS active_transactions FROM INFORMATION_SCHEMA.INNODB_TRX
innodb_deadlocks	Count	db.Locks.innodb_deadlocks	고려상태인 총 개수입니다.	SELECT COUNT AS innodb_deadlocks FROM INFORMATION_SCHEMA.INNODB_METRIC WHERE NAME='lock_deadlocks'
innodb_lock_timeouts	Count	db.Locks.innodb_lock_timeouts	시간을 초과한 고착 상태의 총 개수입니다.	SELECT COUNT AS innodb_lock_timeouts FROM INFORMATION_SCHEMA.INNODB_METRIC WHERE NAME='lock_timeouts'

카운터	Type	지표	설명	정의
innodb_row_lock_waits	타이머	db.Locks.innodb_row_lock_waits	내가 아닌 행에 대한 행 잠금의 총 개수입니다.	SELECT COUNT AS innodb_row_lock_waits FROM INFORMATION_SCHEMA.INNODB_METRIC WHERE NAME='lock_row_lock_waits'

Aurora PostgreSQL용 성능 개선 도우미 카운터

Aurora PostgreSQL용 성능 개선 도우미에서는 다음 데이터베이스 카운터를 사용할 수 있습니다.

주제

- [Aurora PostgreSQL용 기본 카운터 \(p. 405\)](#)
- [기본이 아닌 Aurora PostgreSQL용 카운터 \(p. 406\)](#)

Aurora PostgreSQL용 기본 카운터

PostgreSQL 설명서의 [통계 보기](#)에서 이러한 기본 지표에 대한 정의를 확인하실 수 있습니다.

카운터	Type	Unit	지표
tup_deleted	SQL	초당 투플 수	db.SQL.tup_deleted
tup_fetched	SQL	초당 투플 수	db.SQL.tup_fetched
tup_inserted	SQL	초당 투플 수	db.SQL.tup_inserted
tup_returned	SQL	초당 투플 수	db.SQL.tup_returned
tup_updated	SQL	초당 투플 수	db.SQL.tup_updated
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
active_transactions	트랜잭션	트랜잭션	db.Transactions.active_transactions
blocked_transactions	트랜잭션	트랜잭션	db.Transactions.blocked_transactions
max_used_xact_ids	트랜잭션	트랜잭션	db.Transactions.max_used_xact_ids
xact_commit	트랜잭션	초당 커밋 수	db.Transactions.xact_commit
xact_rollback	트랜잭션	초당 롤백 수	db.Transactions.xact_rollback
blk_read_time	IO	밀리초	db.IO.blk_read_time
blkls_read	IO	초당 블록 수	db.IO.blks_read

카운터	Type	Unit	지표
buffers_backend	IO	초당 블록 수	db.IO.buffers_backend
buffers_backend_fsync	IO	초당 블록 수	db.IO.buffers_backend_fsync
buffers_clean	IO	초당 블록 수	db.IO.buffers_clean
blk_hit	Cache	초당 블록 수	db.Cache.blks_hit
buffers_alloc	Cache	초당 블록 수	db.Cache.buffers_alloc
temp_files	Temp	분당 파일 수	db.Temp.temp_files
numbackends	User	Connections	db.User.numbackends
deadlocks	동시성	분당 교착 상태의 수	db.Concurrency.deadlocks
archived_count	WAL	분당 파일 수	db.WAL.archived_count
archive_failed_count	WAL	분당 파일 수	db.WAL.archive_failed_count

기본이 아닌 Aurora PostgreSQL용 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 자연 시간에 대한 계산 시 2개 이상의 기본 카운터가 사용됩니다.

카운터	Type	지표	설명	정의
checkpoint_sync_time	체크포인트	db.Checkpoint.sync_time	Amazon Aurora에 등록되는 체크포인트 처리 중 일부에서 소요된 총 시간입니다.	checkpoint_sync_time / (checkpoints_timed + checkpoints_req)
checkpoint_write_time	체크포인트	db.Checkpoint.write_time	Amazon Aurora에 등록되는 체크포인트 처리 중 일부에서 소요된 총 시간입니다.	checkpoint_write_time / (checkpoints_timed + checkpoints_req)
read_latency	IO	db.IO.read_latency	아이언스턴스의 백엔드에서 데이터 파일 블록을 읽는 데 소요된 시간입니다.	blk_read_time / blks_read

AWS CloudTrail를 사용하여 성능 개선 도우미 호출 로깅

성능 개선 도우미는 사용자, 역할 또는 성능 개선 도우미의 AWS 서비스에서 수행한 작업을 기록하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 성능 개선 도우미의 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. 추적을 구성하지 않은 경우 이벤트 기록에서 CloudTrail 콘솔의 최신 이벤트를 볼 수도 있습니다. CloudTrail에서 수집한 데이터를 사용하여 특정 정보를 확인할 수 있습니다. 이 정보에는 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 User Guide](#)를 참조하십시오.

CloudTrail의 성능 개선 도우미 정보 작업

CloudTrail은 계정 생성 시 AWS 계정에서 활성화됩니다. 성능 개선 도우미에서 활동이 수행되면 해당 활동은 Event history(이벤트 기록)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 User Guide의 [이벤트 기록에서 CloudTrail 이벤트 보기](#)를 참조하십시오.

성능 개선 도우미의 이벤트를 포함해 AWS 계정의 이벤트를 계속 기록하려면 trail을 생성합니다. 추적은 CloudTrail이 Amazon S3 버킷으로 로그 파일을 전송할 수 있도록 합니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 AWS 리전의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 AWS CloudTrail User Guide의 다음 주제를 참조하십시오.

- [추적 생성 개요](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에 대한 Amazon SNS 알림 구성](#)
- [여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기](#)

모든 성능 개선 도우미 작업은 CloudTrail이 기록하며 [성능 개선 도우미 API 참조](#)에 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. 자격 증명 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 자격 증명으로 했는지 여부
- 역할 또는 연합된 사용자에 대한 임시 보안 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 [CloudTrail userIdentity 요소](#)를 참조하십시오.

성능 개선 도우미 로그 파일 항목 이해

추적은 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 제공할 수 있는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함됩니다. 이벤트는 원본의 단일 요청을 나타냅니다. 각 이벤트에는 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 관한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 호출의 주문 스택 추적이 아니기 때문에 특정 순서로 표시되지 않습니다.

다음 예제는 `GetResourceMetrics` 작업을 보여주는 CloudTrail 로그 항목입니다.

```
{  
    "eventVersion": "1.05",  
    "userIdentity": {  
        "type": "IAMUser",  
        "principalId": "AKIAIOSFODNN7EXAMPLE",  
        "arn": "arn:aws:iam::123456789012:user/johndoe",  
        "accountId": "123456789012",  
        "accessKeyId": "AKIAI44QH8DHBEEXAMPLE",  
        "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": "9fffbef15c-96b5-4fe6-bed9-9fccff1a0525",
"eventID": "08908de0-2431-4e2e-ba7b-f5424f908433",
"eventType": "AwsApiCall",
"recipientAccountId": "123456789012"
}
```

Amazon Aurora 권장 사항 사용

Amazon Aurora에서는 DB 인스턴스, DB 클러스터, DB 클러스터 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.

다음 표에서 이러한 권장 사항의 예를 확인할 수 있습니다.

유형	설명	권장 사항	추가 정보
기본이 아닌 사용자 지정 메모리 파라미터	DB 파라미터 그룹에서는 기본값에서 너무 많이 벗어나는 설정으로 인해 성능이 떨어지고 오류가 발생할 수 있습니다. DB 인스턴스에서 사용하는 DB 파라미터 그룹에서는 사용자 지정 메모리 파라미터를 기본값으로 설정하는 것이 좋습니다.		DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168)
MySQL DB 인스턴스에 대해 활성화된 변경 버퍼링	DB 파라미터 그룹에서 변경 버퍼링을 활성화 하였습니다.	변경 버퍼링을 통해 MySQL DB 인스턴스는 보조 인덱스 유지에 필요한 일부 쓰기를 연기할 수 있습니다. 이러한 구성은 성능을 약간 향상 시킬 수 있지만 충돌 복구가 크게 지연되는 원인이 될 수 있습니다. 충돌 복구 중에 보조 인덱스는 최신 상태이어야 합니다. 따라서 변경 버퍼링의 이점보다 매우 오래 지속될 가능성이 있는 충돌 복구 이벤트로 인한 손해가 더 크므로 변경 버퍼링을 비활성화하는 것이 좋습니다.	AWS 데이터베이스 블로그의 Amazon RDS for MySQL에 대한 파라미터 구성 모범 사례, PART 1: 성능 관련 파라미터
테이블에 로깅 하기	DB 파라미터 그룹은 로깅 출력력을 TABLE로 설정합니다.	로깅 출력력을 TABLE로 설정하면 이 파라미터를 FILE로 설정하는 것보다 더 많은 스토리지를 사용합니다. 스토리지 한도에 도달하지 않게 하려면 로깅 출력	MySQL 데이터베이스 로그 파일 (p. 452)

유형	설명	권장 사항	추가 정보
		파라미터를 FILE로 설정하는 것이 좋습니다.	
DB 인스턴스 가 한 개인 DB 클러스터	DB 클러스터에는 DB 인스턴스가 한 개만 포 함되어 있습니다.	성능 및 가용성 향상을 위해서는 동일한 DB 인스턴스 클래스가 다른 가용 영역 에 있는 또 다른 DB 인스턴스를 추가하 는 것이 좋습니다.	Aurora을 위한 고가용 성 (p. 36)
한 개의 가용 영역에 있는 DB 클러스터	DB 클러스터는 모든 DB 인스턴스가 동일 가용 영역에 있습니다.	가용성 향상을 위해서는 동일한 DB 인 스턴스 클래스가 다른 가용 영역에 있는 또 다른 DB 인스턴스를 추가하는 것이 좋습니다.	Aurora을 위한 고가용 성 (p. 36)
구식이 된 DB 클러스터	DB 클러스터에서 이전 엔진 버전을 실행하고 있습니다.	DB 클러스터는 최신 보안 및 기능 수정 사항이 포함된 최신 마이너 버전으로 유 지하는 것이 좋습니다. 메이저 버전 업 그레이드와 달리 마이너 버전 업그레이 드에는 DB 엔진의 이전 마이너 버전(메 이저 버전은 동일)과의 역호환이 가능 한 변경 사항만이 포함됩니다. 최근 엔 진 버전으로 업그레이드하는 것이 좋습 니다.	Amazon Aurora DB 클러스터 유지 관 리 (p. 306)
서로 다른 파 라미터 그룹이 있는 DB 클러 스터	DB 클러스터에는 DB 인스턴스에 서로 다른 DB 파라미터 그룹이 할당되어 있습니다.	서로 다른 파라미터 그룹으로 인해 DB 인스턴스가 서로 호환되지 않을 수 있습 니다. 문제를 피하고 유지 관리를 더 쉽 게 하려면 DB 클러스터의 모든 DB 인 스턴스에 동일한 파라미터 그룹을 사용 하는 것이 좋습니다.	DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168)
서로 다른 DB 인스턴스 클러 스터가 있는 DB 클러스터	DB 클러스터에 서로 다른 DB 인스턴스 클 래스를 사용하는 DB 인스턴스가 있습니다.	DB 인스턴스에 서로 다른 DB 인스턴스 클래스를 사용하면 문제가 발생할 수 있 습니다. 예를 들어 덜 강력한 DB 인스턴 스 클래스가 승격하여 더 강력한 DB 인 스턴스 클래스를 교체하는 경우 성능이 떨어질 수 있습니다. 문제를 피하고 유 지 관리를 더 쉽게 하려면 DB 클러스터 의 모든 DB 인스턴스에 동일한 DB 인 스턴스 클래스를 사용하는 것이 좋습니 다.	Aurora 복제본 (p. 55)

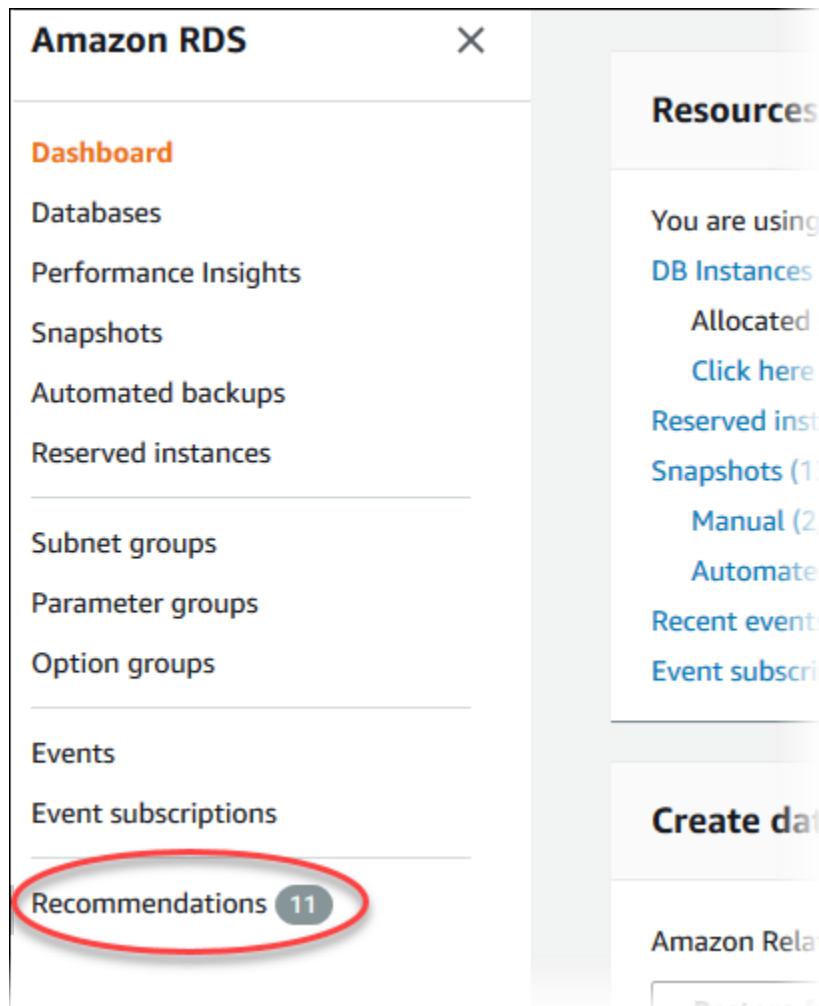
Amazon Aurora는 리소스가 생성되거나 수정될 때 리소스에 대해 권장 사항을 생성합니다. Amazon Aurora 역시 리소스를 주기적으로 스캔하고 권장 사항을 생성합니다.

Amazon Aurora 권장 사항에 대응

AWS Management 콘솔에서 권장 사항을 확인할 수 있습니다. 즉시 권장 조치를 수행하거나 다음 유지 관리 기간으로 예약하거나 무시할 수 있습니다.

Amazon Aurora 권장 사항에 대응하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Recommendations(권장 사항)를 선택합니다.



Recommendations(권장 사항) 페이지가 표시됩니다.

Recommendations

Active (2) | Dismissed (0) | Scheduled (0) | Applied (0)

▶ Heterogeneous instance classes in DB cluster (1)
DB clusters which have heterogeneous instance types [Info](#)

▶ Heterogeneous parameters for DB cluster (1)
DB clusters which have heterogeneous parameter groups assigned to its instances [Info](#)

3. Recommendations(권장 사항) 페이지에서 다음 중 하나를 선택합니다.

- Active(활성) – 적용, 무시 또는 예약할 수 있는 현재 권장 사항을 표시합니다.
- Dismissed(무시됨) – 무시된 권장 사항을 표시합니다. Dismissed(무시됨)를 선택할 때 무시된 권장 사항을 적용할 수 있습니다.
- 예약됨 – 예약되었지만 아직 적용되지 않은 권장 사항을 표시합니다. 이러한 권장 사항은 다음 예약된 유지 관리 기간에 적용됩니다.
- Applied(적용됨) – 현재 적용된 권장 사항을 표시합니다.

권장 사항 목록에서 섹션을 열어 해당 섹션의 권장 사항을 볼 수 있습니다.

The screenshot shows the 'Recommendations' section of the Amazon Aurora console. At the top, there are four tabs: 'Active (2)', 'Dismissed (0)', 'Scheduled (0)', and 'Applied (0)'. The 'Active (2)' tab is selected. Below the tabs, there is a section titled '▼ Heterogeneous instance classes in DB cluster (1)'. This section contains a link 'DB clusters which have heterogeneous instance types' and an 'Info' button. The main content area is titled 'DB clusters' and includes a search bar 'Filter recommendations', a 'Dismiss' button, a 'Schedule' button, and an 'Apply now' button. A table lists one resource, 'cluster-1', under the 'Resource' column. The 'Recommendation' column for 'cluster-1' states: 'Your DB cluster cluster-1 has DB instances that use different DB instance classes. Using different DB instance classes for DB instances can cause problems. To avoid problems and for easier maintenance, we recommend using db.t3.small for all of the DB instances in the DB cluster.' The 'Affected resources' column shows 'cluster-1-reader'. Navigation arrows and a refresh icon are also present at the bottom of the table.

각 섹션의 권장 사항 표시에 대한 기본 설정을 구성하려면 기본 설정 아이콘을 선택합니다.

This screenshot is identical to the one above, showing the 'Recommendations' section for the 'Heterogeneous instance classes in DB cluster' section. However, the 'Apply now' button in the top right corner of the 'DB clusters' table is circled in red, indicating it is the target for configuration.

표시되는 기본 설정 창에서 표시 옵션을 설정할 수 있습니다. 이러한 옵션에는 표시된 열과 페이지에 표시할 권장 사항 수가 포함됩니다.

4. 활성 권장 사항 관리:

- 활성을 선택하고 하나 이상의 섹션을 열어 권장 사항을 봅니다.
- 하나 이상의 권장 사항을 선택하고 Apply now(지금 적용)(즉시 적용), Schedule(일정)(다음 유지 관리 기간에 적용) 또는 Dismiss(무시)를 선택합니다.

권장 사항에 대해 지금 적용 버튼이 표시되지만 사용할 수 없는 경우(회색으로 표시) DB 인스턴스를 사용할 수 없습니다. DB 인스턴스 상태가 사용 가능한 경우 권장 사항을 즉시 적용할 수 있습니다. 예를 들어 DB 인스턴스의 상태가 수정 중인 경우 권장 사항을 즉시 적용할 수 없습니다. 이 경우 DB 인스턴스가 사용 가능할 때까지 기다렸다가 권장 사항을 적용합니다.

권장 사항에 대해 지금 적용 버튼이 표시되지 않는 경우 Recommendations(권장 사항) 페이지를 사용하여 권장 사항을 적용할 수 없습니다. DB 인스턴스를 수정하여 수동으로 권장 사항을 적용할 수 있습니다.

DB 클러스터 수정에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.

Note

지금 적용을 선택하면 간단한 DB 인스턴스 중단이 발생할 수 있습니다.

Aurora PostgreSQL에서 데이터베이스 활동 스트림 사용

데이터베이스 활동을 모니터링하면 데이터베이스를 보호하고 규정 준수 및 규제 요건을 충족하는 데 도움이 됩니다. PostgreSQL과 호환되는 Amazon Aurora에서 데이터베이스 활동을 모니터링하는 한 가지 방법은 데이터베이스 활동 스트림을 사용하는 것입니다. 데이터베이스 활동 스트림은 관계형 데이터베이스에서 데이터베이스 활동의 데이터 스트림을 거의 실시간으로 제공합니다. 데이터베이스 활동 스트림을 타사 모니터링 도구와 통합하면 데이터베이스 활동을 모니터링하고 감사할 수 있습니다.

외부 보안 위협 외에도 관리형 데이터베이스는 데이터베이스 관리자(DBA)의 내부자 위험으로부터 보호해야 합니다. 데이터베이스 활동 스트림은 데이터베이스 활동 스트림에 대한 DBA 액세스를 제어하여 내부 위험으로부터 데이터베이스를 보호합니다. 따라서 데이터베이스 활동 스트림의 수집, 전송, 저장 및 후속 처리는 데이터베이스를 관리하는 DBA의 액세스 범위를 벗어납니다.

Aurora PostgreSQL의 데이터베이스 활동 스트림이 데이터베이스를 대신하여 생성된 Amazon Kinesis 데이터 스트림으로 푸시됩니다. Kinesis에서, Amazon CloudWatch 또는 애플리케이션이 규정 준수 관리를 위해 활동 스트림을 사용할 수 있습니다. 이러한 규정 준수 애플리케이션으로는 Imperva의 SecureSphere Database Audit and Protection, McAfee의 Data Center Security Suite 또는 IBM의 InfoSphere Guardium이 있습니다. 이러한 애플리케이션은 활동 스트림 정보를 사용하여 알림을 생성하고 Amazon Aurora 데이터베이스의 모든 활동에 대한 감사를 제공할 수 있습니다.

데이터베이스 활동 스트림에는 다음과 같은 제한과 요구 사항이 있습니다.

- 현재 이러한 스트림은 PostgreSQL과 호환되는 Aurora 버전 2.3 이상에서만 지원됩니다.
- 데이터베이스 활동 스트림은 [Aurora에 사용 가능한 모든 DB 인스턴스 클래스의 하드웨어 사양 \(p. 31\)](#)의 Aurora PostgreSQL에 나열된 DB 인스턴스 클래스를 지원합니다. 단, t3.medium 인스턴스 클래스는 지원하지 않습니다.
- 다음 AWS 리전에는 지원되지 않습니다.
 - 중국(북경) 지역, cn-north-1
 - 중국(닝샤) 리전, cn-northwest-1
 - AWS GovCloud(미국 동부), us-gov-east-1
 - AWS GovCloud(미국 서부), us-gov-west-1
- 활동 스트림은 항상 암호화되므로 AWS Key Management Service(AWS KMS)를 사용해야 합니다.

주제

- [활동 스트림 시작 \(p. 413\)](#)
- [활동 스트림 상태 가져오기 \(p. 414\)](#)
- [활동 스트림 중지 \(p. 415\)](#)
- [데이터베이스 활동 스트림 모니터링 \(p. 416\)](#)

- 데이터베이스 활동 스트림에 대한 액세스 관리 (p. 427)

활동 스트림 시작

DB 클러스터 수준에서 활동 스트림을 시작하여 클러스터의 모든 DB 인스턴스에서 데이터베이스 활동을 모니터링합니다. 클러스터에 추가된 모든 DB 인스턴스도 자동으로 모니터링됩니다.

활동 스트림을 시작하면 변경 또는 액세스와 같은 각 데이터베이스 활동 이벤트가 활동 스트림 이벤트를 생성합니다. 액세스 이벤트는 CONNECT 및 SELECT 같은 SQL 명령에서 생성됩니다. 변경 이벤트는 CREATE 및 INSERT 같은 SQL 명령에서 생성됩니다. 각 활동 스트림 이벤트의 내구성을 위해 암호화하고 저장합니다. 데이터베이스 세션이 데이터베이스 활동 이벤트를 동기식으로 또는 비동기식으로 처리하도록 선택할 수 있습니다.

- 동기 모드 – 동기 모드에서 데이터베이스 세션이 활동 스트림 이벤트를 생성하면 세션은 이벤트가 내구성을 가질 때까지 차단됩니다. 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.

- 비동기 모드 – 비동기 모드에서 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.

콘솔

활동 스트림을 시작하려면

- <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
- 탐색 창에서 데이터베이스를 선택합니다.
- 사용할 DB 클러스터를 선택합니다.
- 작업의 경우 Start activity stream(활동 스트림 시작)을 선택합니다. Database Activity Stream(데이터베이스 활동 스트림) 창이 나타납니다.
- Database Activity Stream(데이터베이스 활동 스트림) 창에 다음 설정을 입력하십시오.
 - 마스터 키의 경우, AWS KMS 키 목록에서 키를 선택하십시오.

마스터 키는 키를 암호화하는 데 사용되고, 이 키가 로깅된 데이터베이스 활동을 암호화합니다. 기본 키가 아닌 마스터 키를 선택해야 합니다. 암호화 키와 AWS KMS에 대한 자세한 내용은 AWS Key Management Service Developer Guide의 [AWS Key Management Service란 무엇입니까?](#)를 참조하십시오.

- Database activity stream mode(데이터베이스 활동 스트림 모드)의 경우 Asynchronous(비동기) 또는 Synchronous(동기)를 선택하십시오.
- Apply immediately(즉시 적용)를 선택합니다.

Schedule for the next maintenance window(다음 유지 관리 기간 예약)을 선택하면 데이터베이스가 즉시 재시작되지 않습니다. 대신, PENDING REBOOT 상태로 유지됩니다. 이 경우 다음 유지 관리 기간이나 수동 재시작 시까지 데이터베이스 유지 관리 스트림이 시작되지 않습니다.

설정 입력을 완료하면 계속을 선택하십시오.

클러스터의 DB 인스턴스 상태는 데이터베이스 활동 스트림이 구성 중임을 표시합니다.

AWS CLI

DB 클러스터에 대한 데이터베이스 활동 스트림을 시작하려면 [start-activity-stream](#) AWS CLI 명령을 사용하여 DB 클러스터를 구성하십시오. --region 파라미터를 사용하여 DB 클러스터의 AWS 리전을 식별하십시오. --apply-immediately 파라미터는 선택 항목입니다.

Linux, OS X, Unix의 경우:

```
aws rds --region us-west-2 \
    start-activity-stream \
    --mode sync \
    --kms-key-id MY_KMS_KEY_ARN \
    --resource-arn MY_CLUSTER_ARN \
    --apply-immediately \
    --profile MY_PROFILE_CREDENTIALS
```

Windows의 경우:

```
aws rds --region us-west-2 ^
    start-activity-stream ^
    --mode sync ^
    --kms-key-id MY_KMS_KEY_ARN ^
    --resource-arn MY_CLUSTER_ARN ^
    --apply-immediately ^
    --profile MY_PROFILE_CREDENTIALS
```

활동 스트림 상태 가져오기

콘솔 또는 AWS CLI를 사용하여 활동 스트림의 상태를 가져올 수 있습니다.

콘솔

DB 클러스터의 활동 스트림 상태를 가져오려면

1. <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택한 후 DB 클러스터를 선택합니다.
3. 구성 탭을 선택하고 Database activity stream(데이터베이스 활동 스트림)에서 상태를 확인하십시오.

AWS CLI

[describe-db-clusters](#) CLI 요청에 대한 응답으로 DB 클러스터의 활동 스트림 구성은 가져올 수 있습니다. 다음 예제에서 ActivityStreamKinesisStreamName, ActivityStreamStatus, ActivityStreamKmsKeyId 및 ActivityStreamMode에 대한 값을 참조하십시오.

요청은 다음과 같습니다.

```
aws rds --region us-west-2 describe-db-clusters --db-cluster-identifier my-cluster --
profile MY_PROFILE_CREDENTIALS
```

응답에는 데이터베이스 활동 스트림에 대한 다음 항목이 포함됩니다.

```
{
  "DBClusters": [
```

```
{  
    "DBClusterIdentifier": "my-cluster",  
    . . .  
    "ActivityStreamKinesisStreamName": "aws-rds-das-cluster-A6TSYXITZCZXJHIRVFUBZ5LTWY",  
    "ActivityStreamStatus": "starting",  
    "ActivityStreamKmsKeyId": "12345678-abcd-efgh-ijkl-bd041f170262",  
    "ActivityStreamMode": "sync",  
    "DbClusterResourceId": "cluster-ABCD123456"  
    . . .  
}  
]
```

활동 스트림 중지

콘솔 또는 AWS CLI를 사용하여 활동 스트림을 중지할 수 있습니다.

DB 클러스터를 삭제하면 활동 스트림이 자동으로 중지된다는 점에 유의하십시오.

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 선택하십시오.
4. 작업의 경우 Stop activity stream(작업 스트림 중지)을 선택합니다. Database Activity Stream(데이터베이스 활동 스트림) 창이 나타납니다.
 - a. Apply immediately(즉시 적용)를 선택합니다.

Schedule for the next maintenance window(다음 유지 관리 기간 예약)을 선택하면 데이터베이스가 즉시 재시작되지 않습니다. 대신, PENDING REBOOT 상태로 유지됩니다. 이 경우 다음 유지 관리 기간이나 수동 재시작 시까지 데이터 활동 스트림이 비활성화되지 않습니다.

- b. [Continue]를 선택합니다.

AWS CLI

DB 클러스터에 대한 데이터베이스 활동 스트림을 중지하려면 AWS CLI 명령 `stop-activity-stream`을 사용하여 DB 클러스터를 구성하십시오. `--region` 파라미터를 사용하여 DB 클러스터의 AWS 리전을 식별하십시오. `--apply-immediately` 파라미터는 선택 항목입니다.

Linux, OS X, Unix의 경우:

```
aws rds --region us-west-2 \  
    stop-activity-stream \  
    --resource-arn MY_CLUSTER_ARN \  
    --apply-immediately \  
    --profile MY_PROFILE_CREDENTIALS
```

Windows의 경우:

```
aws rds --region us-west-2 ^  
    stop-activity-stream ^  
    --resource-arn MY_CLUSTER_ARN ^
```

```
--apply-immediately ^  
--profile MY_PROFILE_CREDENTIALS
```

데이터베이스 활동 스트림 모니터링

데이터베이스 활동 스트림은 데이터베이스의 모든 활동을 모니터링하고 보고합니다. Amazon Kinesis를 사용하여 활동 스트림이 수집되고 보안 서버로 전송됩니다. Kinesis에서 활동 스트림이 모니터링되거나, 나중에 추가 분석을 위해 다른 서비스 및 애플리케이션에 사용될 수 있습니다.

다음 활동 범주가 모니터링되고 활동 스트림 감사 로그에 기록됩니다.

- SQL 명령 – 모든 SQL 명령, prepared 문, PostgreSQL 함수 및 SQL용 프로시저 언어(PL/SQL)로 된 함수가 감사 대상입니다.
- 다른 데이터베이스 정보 – 모니터링되는 활동에는 전체 SQL 문, 파라미터, 바인드 변수, DML 명령의 영향을 받은 행의 행 수, 액세스된 객체 및 고유한 데이터베이스 이름이 포함됩니다.
- 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 사용하여 사용자에게 알립니다. 결함이 발생하면 DB 인스턴스를 종료하거나 계속 진행할 수 있습니다.

주제

- [Kinesis에서 활동 스트림에 액세스 \(p. 416\)](#)
- [감사 로그 내용 및 예 \(p. 417\)](#)
- [AWS SDK를 사용하여 활동 스트림 처리 \(p. 421\)](#)

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 Amazon CloudWatch 같은 소비자 애플리케이션에 연결할 수 있습니다. Imperva, McAfee 또는 IBM의 규정 준수 관리 애플리케이션에 연결할 수도 있습니다.

Kinesis에서 활동 스트림에 액세스하려면

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Kinesis 스트림 목록에서 활동 스트림을 선택합니다.

활동 스트림의 이름에는 접두사 `aws-rds-das-`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 탭을 선택하십시오.

AWS CLI를 사용하여 활동 스트림의 전체 Kinesis 스트림 이름을 찾으려면 `describe-db-clusters` CLI 요청을 사용하고 응답에서 `DBActivityStreamKinesisStreamName` 값을 메모하십시오.

3. 데이터베이스 활동을 관찰하려면 모니터링을 선택하십시오.

Amazon Kinesis 사용에 대한 자세한 내용은 [Amazon Kinesis 데이터 스트림이란 무엇입니까?](#)를 참조하십시오.

감사 로그 내용 및 예

모니터링되는 데이터베이스 활동 이벤트는 Kinesis 활동 스트림에 JSON 문자열로 표시됩니다. 구조는 `DatabaseActivityMonitoringRecord`를 포함하는 JSON 객체로 구성되며, 여기에는 `databaseActivityEventList` 활동 이벤트 배열이 포함됩니다.

주제

- [감사 로그 예제 \(p. 417\)](#)
- [활동 이벤트 레코드 \(p. 419\)](#)
- [databaseActivityEventList JSON 배열 \(p. 419\)](#)

감사 로그 예제

다음은 활동 이벤트 레코드의 해독된 JSON 감사 로그 샘플입니다.

Example CONNECT SQL 문의 활동 이벤트 레코드

다음은 `psql` 클라이언트(clientApplication)가 `CONNECT` SQL 문(command)을 사용하여 로그인한 활동 이벤트 레코드입니다.

```
{  
    "type": "DatabaseActivityMonitoringRecord",  
    "clusterId": "cluster-4HNY5V4RRNPCKYB7ICFKE5JBQQ",  
    "instanceId": "db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",  
    "databaseActivityEventList": [  
        {  
            "logTime": "2019-05-23 01:31:28.610198+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"  
        }  
    ]  
}
```

Example CREATE TABLE 문의 활동 이벤트 레코드

다음은 `CREATE TABLE` 이벤트의 예입니다.

```
{
```

```
"type": "DatabaseActivityMonitoringRecord",
"clusterId": "cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
"instanceId": "db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
"databaseActivityEventList": [
    {
        "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": "pgsql",
        "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"
    }
]
}
```

Example SELECT 문의 활동 이벤트 레코드

다음은 SELECT 이벤트의 예입니다.

```
{
    "type": "DatabaseActivityMonitoringRecord",
    "clusterId": "cluster-4HNY5V4RRNPKKYB7ICFKE5JBQQ",
    "instanceId": "db-FZJTMYKCXQBUUZ6VLU7NW3ITCM",
    "databaseActivityEventList": [
        {
            "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": "pgsql",
            "exitCode": null,
            "class": "READ",
            "serverVersion": "2.3.1",
            "serverType": "PostgreSQL",
            "type": "record"
        }
    ]
}
```

```

        "serviceName": "Amazon Aurora PostgreSQL-Compatible edition",
        "serverHost": "172.31.3.192",
        "netProtocol": "TCP",
        "dbProtocol": "Postgres 3.0",
        "type": "record"
    }
]
}

```

활동 이벤트 레코드

감사 로그 활동 이벤트 레코드는 다음 정보가 포함된 JSON 객체입니다.

JSON 필드	데이터 형식	설명
<code>type</code>	문자열	JSON 레코드 형식입니다. 이 값은 <code>DatabaseActivityMonitoringRecord</code> 입니다.
<code>clusterId</code>	문자열	DB 클러스터 ID
<code>instanceId</code>	문자열	DB 인스턴스 ID.
<code>databaseActivityEventList</code>	문자열 배열	<p>base64 바이트 배열로 암호화된 JSON 객체입니다. 이 콘텐츠를 해독하려면 다음 단계를 따르십시오.</p> <ol style="list-style-type: none"> 데이터베이스 활동 스트림의 마스터 키를 사용하여 키 JSON 필드의 값을 해독합니다. 1단계에 해독한 키를 사용하여 <code>databaseActivityEventList</code> 객체를 해독합니다.

databaseActivityEventList JSON 배열

감사 로그 페이지는 암호화된 `databaseActivityEventList` JSON 배열입니다. 다음 표는 해독된 감사 로그의 `DatabaseActivityEventList` 배열에 각 활동 이벤트의 필드를 영문자순으로 나열합니다.

Field	데이터 형식	설명
<code>class</code>	문자열	<p>활동 이벤트의 클래스입니다. 유효한 값은 다음과 같습니다.</p> <ul style="list-style-type: none"> • 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.

Field	데이터 형식	설명
clientApplication	문자열	클라이언트가 보고한 대로 클라이언트가 연결에 사용한 애플리케이션입니다. 클라이언트는 이 정보를 제공할 필요가 없으므로 값은 null일 수 있습니다.
command	문자열	명령 세부 정보가 없는 SQL 명령의 이름.
commandText	문자열	<p>사용자가 전달한 실제 SQL 문. 이 필드는 연결 또는 연결 해제 레코드를 제외한 모든 유형의 레코드에 사용되며 이 경우 값은 null입니다.</p> <p>민감한 데이터</p> <p>민감한 데이터를 포함하여 전체 SQL 텍스트가 표시됩니다. 그러나 데이터베이스 사용자 암호는 다음 SQL 문에서와 같이 컨텍스트에서 판별할 수 있는 경우 삭제됩니다.</p> <pre>ALTER ROLE role-name WITH password</pre>
databaseName	문자열	사용자가 연결된 데이터베이스입니다.
dbProtocol	문자열	데이터베이스 프로토콜.
dbUserName	문자열	클라이언트가 인증한 데이터베이스 사용자.
exitCode	int	세션 종료 레코드에 사용되는 값. 정리 종료의 경우, 여기에 종료 코드가 포함됩니다. 일부 결합 시나리오에서는 종료 코드를 항상 얻을 수 있는 것은 아닙니다. PostgreSQL이 <code>exit()</code> 을 수행하거나 연산자가 <code>kill -9</code> 같은 명령을 수행하는 경우가 그 예입니다.
logTime	문자열	감사 코드 경로에 기록된 타임스탬프.
netProtocol	문자열	네트워크 통신 프로토콜.
objectName	문자열	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	문자열	<p>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.</p> <ul style="list-style-type: none"> • COMPOSITE TYPE • FOREIGN TABLE • FUNCTION • INDEX • MATERIALIZED VIEW • SEQUENCE • TABLE • TOAST TABLE • VIEW • UNKNOWN

Field	데이터 형식	설명
paramList	문자열	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	문자열	데이터베이스의 log_hostname 파라미터 설정에 따라 클라이언트 IP 주소 또는 호스트 이름.
remotePort	문자열	클라이언트 포트 번호.
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.
serverType	문자열	데이터베이스 서버 유형(예: PostgreSQL).
serverVersion	문자열	데이터베이스 서버 버전.
serviceName	문자열	서비스의 이름(예: Amazon Aurora PostgreSQL-Compatible edition).
sessionId	int	의사 고유 세션 식별자.
statementId	int	클라이언트의 SQL 문에 대한 ID. 카운터는 세션 수준에 있으며 클라이언트가 입력한 각 SQL 문 단위로 증가합니다.
substatementId	int	SQL 하위 구문의 ID입니다. 이 값은 statementId 필드로 식별되는 각 SQL 문에 대해 포함된 하위 명령문을 계산합니다.
type	문자열	이벤트 유형입니다. 유효한 값은 record 또는 heartbeat입니다.

AWS SDK를 사용하여 활동 스트림 처리

AWS SDK를 사용하여 프로그래밍 방식으로 활동 스트림을 처리할 수 있습니다. 다음은 Kinesis 데이터 스트림을 처리하는 방법을 알 수 있는, 제대로 작동하는 Java 및 Python 예제입니다.

Java

```

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.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 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 statementId;
        String substatementId;
        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,
getBytes(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 through $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 zlib
import boto3
import base64
import json
import aws_encryption_sdk
from Crypto.Cipher import AES
from aws_encryption_sdk import DefaultCryptoMaterialsManager
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

aws_access_key_id="YOUR_ACCESS_KEY"
aws_secret_access_key="YOUR_SECRET_KEY"
key_id = "your_key_id"
stream_name = "YOUR_KINESIS_STREAM_NAME"
region_name = 'YOUR_REGION_NAME'
cluster_id = "YOUR_CLUSTER_ID"

class MyRawMasterKeyProvider(RawMasterKeyProvider):
    provider_id = "BC"

    def __new__(cls, *args, **kwargs):
        obj = super(RawMasterKeyProvider, cls).__new__(cls)
        return obj

    def __init__(self, wrapping_key):
        RawMasterKeyProvider.__init__(self)
        self.wrapping_key = wrapping_key

    def _get_raw_key(self, key_id):
        return self.wrapping_key

def decrypt(decoded, plaintext):
    wrapping_key =
        WrappingKey(wrapping_algorithm=WrappingAlgorithm.AES_256_GCM_IV12_TAG16_NO_PADDING,
                    wrapping_key=plaintext,
                    wrapping_key_type=EncryptionKeyType.SYMMETRIC)
    my_key_provider = MyRawMasterKeyProvider(wrapping_key)
    my_key_provider.add_master_key("DataKey")
    with aws_encryption_sdk.stream(
            mode='d',
            source=decoded,

materials_manager=DefaultCryptoMaterialsManager(master_key_provider=my_key_provider)
    ) as decryptor:
```

```
for chunk in decryptor:
    d = zlib.decompressobj(16 + zlib.MAX_WBITS)
    decompressed_database_stream = d.decompress(chunk)
    print decompressed_database_stream

if __name__ == '__main__':
    session = boto3.session.Session(
        aws_access_key_id=aws_access_key_id,
        aws_secret_access_key=aws_secret_access_key
    )

    kms = session.client('kms', region_name=region_name)

    client = session.client('kinesis', region_name=region_name)

    response = client.describe_stream(StreamName=stream_name)
    shard_it = {}
    for idx, shard in enumerate(response['StreamDescription']['Shards']):
        shared_iterator = client.get_shard_iterator(
            StreamName=stream_name,
            ShardId=response['StreamDescription']['Shards'][idx]['ShardId'],
            ShardIteratorType='LATEST',
        )["ShardIterator"]
        shard_it[idx] = shared_iterator

    while True:
        rows = []
        for shared_iterator in shard_it:
            response = client.get_records(ShardIterator=shard_it[shared_iterator],
                                           Limit=10000)
            for record in response['Records']:
                record_data = record['Data']
                record_data = json.loads(record_data)
                key = record_data['key']
                decoded = base64.b64decode(record_data['DatabaseActivityEvents'])
                decoded_data_key = base64.b64decode(record_data['key'])
                decrypt_result = kms.decrypt(CiphertextBlob=decoded_data_key,
                                              EncryptionContext={"aws:rds:dbc-id": cluster_id})
                plaintext = decrypt_result['Plaintext']
                cipher = AES.new(plaintext, AES.MODE_GCM)
                decrypt(decoded, decrypt_result['Plaintext'])
            shard_it[shared_iterator] = response["NextShardIterator"]
```

데이터베이스 활동 스트림에 대한 액세스 관리

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 Identity and Access Management\(IAM\)](#) (p. 960) 단원을 참조하십시오. IAM 정책 생성에 대한 자세한 내용은 [IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용](#) (p. 979) 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:ConfigureDBActivityStreams`를 사용하십시오. 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 시작하는 것을 거부합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenyStartActivityStreams",  
            "Effect": "Deny",  
            "Action": "rds:StartActivityStream",  
            "Resource": "*"  
        }  
    ]  
}
```

```
        "Sid":"DenyStartActivityStreams",
        "Effect":"Deny",
        "Action":"rds:StartActivityStream",
        "Resource": "*"
    }
]
```

Example 데이터베이스 활동 스트림 중지를 거부하는 정책

다음 IAM 정책 예제는 사용자 또는 역할이 활동 스트림을 중지하는 것을 거부합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "DenyStopActivityStreams",
            "Effect": "Deny",
            "Action": "rds:StopActivityStream",
            "Resource": "*"
        }
    ]
}
```

Amazon RDS 이벤트 알림 서비스 사용

주제

- [Amazon RDS 이벤트 카테고리 및 이벤트 메시지 \(p. 430\)](#)
- [Amazon RDS 이벤트 알림 구독 \(p. 436\)](#)
- [Amazon RDS 이벤트 알림 구독의 목록 표시 \(p. 438\)](#)
- [Amazon RDS 이벤트 알림 구독 변경 \(p. 439\)](#)
- [Amazon RDS 이벤트 알림 구독에 대한 소스 식별자 추가 \(p. 440\)](#)
- [Amazon RDS 이벤트 알림 구독의 소스 식별자 제거 \(p. 441\)](#)
- [Amazon RDS 이벤트 알림 카테고리의 목록 표시 \(p. 442\)](#)
- [Amazon RDS 이벤트 알림 구독 삭제 \(p. 443\)](#)

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.

Amazon RDS는 구독 가능한 카테고리로 이벤트를 그룹화합니다. 따라서 해당 카테고리의 이벤트가 발생했을 때 이에 대한 알림 메시지를 받을 수 있습니다. 구독 가능한 이벤트 범주로는 DB 인스턴스, DB 클러스터, DB 클러스터 스냅샷, DB 파라미터 그룹, DB 보안 그룹 등이 있습니다. 예를 들어 임의의 DB 인스턴스에 대한 백업 카테고리를 구독할 경우 백업 관련 이벤트가 발생하여 DB 인스턴스에 영향을 끼칠 때마다 알림 메시지가 수신됩니다. 혹은 DB 보안 그룹의 구성 변경 카테고리를 구독하면 DB 보안 그룹이 변경될 때마다 메시지가 수신됩니다. 또한 이벤트 알림 메시지 구독이 변경되어도 알림 메시지가 수신됩니다.

Amazon Aurora의 경우 이벤트는 DB 클러스터와 DB 인스턴스 수준 모두에서 발생하므로 Aurora DB 클러스터 또는 Aurora DB 인스턴스를 구독하면 이벤트를 수신할 수 있습니다.

이벤트 알림 메시지는 구독 생성 시 입력한 주소로 보내집니다. 모든 이벤트 알림 메시지를 수신하거나 프로덕션 DB 인스턴스의 중요 이벤트만 수신하는 등 다른 구독을 복수로 생성하는 것도 가능합니다. Amazon RDS 콘솔에서 활성화 옵션을 아니요로 설정하거나, AWS CLI 또는 Amazon RDS API를 통해 `Enabled` 파라미터를 `false`로 설정하면 구독을 삭제하지 않고 알림 기능을 쉽게 끌 수 있습니다.

Important

Amazon RDS는 이벤트 스트림에서 전송된 이벤트 순서를 보장하지 않습니다. 이벤트 순서는 변경될 수 있습니다.

Note

SMS 문자 메시지를 사용한 Amazon RDS 이벤트 알림 서비스는 현재 미국 동부(버지니아 북부) 리전에서 주제 Amazon 리소스 이름(ARN)과 Amazon RDS 리소스로 제공되고 있습니다. SNS 문자 메시지 사용에 대한 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 [Amazon SNS를 통한 SMS 알림 메시지 전송 및 수신](#)을 참조하십시오.

Amazon RDS에서는 Amazon SNS 주제의 ARN을 사용하여 각 구독을 식별합니다. Amazon RDS 콘솔은 구독 생성 시 ARN을 생성합니다. 그 밖에 CLI 또는 API를 사용하는 경우에는 구독을 생성하면서 Amazon SNS 콘솔이나 Amazon SNS API를 통해 ARN을 생성합니다.

Amazon RDS 이벤트 알림에 대한 결제는 Amazon Simple Notification Service(Amazon SNS)를 통해 이루어집니다. 이벤트 알림 사용 시 Amazon SNS 요금이 적용됩니다. Amazon SNS 결제에 대한 자세한 내용은 [Amazon Simple Notification Service 요금](#)을 참조하십시오.

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.
2. Amazon RDS가 구독 생성 시 제출한 이메일 주소로 송인 이메일 또는 SMS 메시지를 전송합니다. 구독 여부를 확인하려면 전송된 알림의 링크를 선택합니다.
3. 구독 여부를 확인하면 Amazon RDS 콘솔의 [My Event Subscriptions] 영역에 구독 상태가 업데이트됩니다.
4. 이제부터 이벤트 알림 메시지가 수신됩니다.

Note

Amazon SNS가 구독된 HTTP 또는 HTTPS 앤드포인트에 알림을 전송할 때 앤드포인트에 전송된 POST 메시지에는 JSON 문서를 포함하는 메시지 본문이 있습니다. 자세한 내용은 Amazon Simple Notification Service 개발자 안내서의 [Amazon SNS 메시지 및 JSON 형식](#)을 참조하십시오.

AWS Lambda를 사용하여 DB 인스턴스의 이벤트 알림을 처리할 수 있습니다. 자세한 내용은 AWS Lambda 개발자 안내서의 [Amazon RDS와 함께 AWS Lambda 사용](#)을 참조하십시오.

다음 섹션부터는 수신되는 모든 카테고리와 이벤트에 대해 살펴보겠습니다. 또한 Amazon RDS 이벤트 구독 및 구독 작업에 대한 정보도 제공합니다.

Amazon RDS 이벤트 카테고리 및 이벤트 메시지

Amazon RDS는 Amazon RDS 콘솔, AWS CLI 또는 API를 사용하여 구독할 수 있는 이벤트 카테고리가 매우 많습니다. 이러한 카테고리는 각각 DB 인스턴스, DB 스냅샷, DB 보안 그룹 또는 DB 파라미터 그룹 등 다양한 소스 유형에 적용됩니다.

다음 표는 DB 인스턴스가 소스 유형일 때 이벤트 카테고리와 이벤트 목록을 나타냅니다.

Category	Amazon RDS 이벤트 ID	설명
가용성	RDS-EVENT-0006	재시작된 DB 인스턴스.
가용성	RDS-EVENT-0004	DB 인스턴스 종료.
가용성	RDS-EVENT-0022	MySQL 또는 MariaDB 재시작 중 오류가 발생했습니다.

Category	Amazon RDS 이벤트 ID	설명
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 (p. 509) 단원을 참조하십시오.
역추적	RDS-EVENT-0132	실제 역추적 기간은 대상 역추적 기간과 같습니다.
backup	RDS-EVENT-0001	DB 인스턴스 백업.
backup	RDS-EVENT-0002	완료된 DB 인스턴스 백업.
구성 변경	RDS-EVENT-0009	DB 인스턴스가 보안 그룹에 추가되었습니다.
구성 변경	RDS-EVENT-0024	DB 인스턴스가 다중 AZ DB 인스턴스로 전환 중입니다.
구성 변경	RDS-EVENT-0030	DB 인스턴스가 단일 AZ DB 인스턴스로 전환 중입니다.
구성 변경	RDS-EVENT-0012	수정을 데이터베이스 인스턴스 클래스에 적용.
구성 변경	RDS-EVENT-0018	현재 이 DB 인스턴스의 스토리지 설정이 변경 중입니다.
구성 변경	RDS-EVENT-0011	이 DB 인스턴스의 파라미터 그룹이 변경되었습니다.
구성 변경	RDS-EVENT-0092	이 DB 인스턴스의 파라미터 그룹 업데이트가 완료되었습니다.
구성 변경	RDS-EVENT-0028	이 DB 인스턴스의 자동 백업이 비활성화되었습니다.
구성 변경	RDS-EVENT-0032	이 DB 인스턴스의 자동 백업이 활성화되었습니다.
구성 변경	RDS-EVENT-0033	마스터 사용자 이름과 일치하는 사용자가 [숫자]명 있습니다. 특정 호스트에 연결되지 않은 사용자가 재설정되었습니다.
구성 변경	RDS-EVENT-0025	DB 인스턴스가 다중 AZ DB 인스턴스로 전환되었습니다.
구성 변경	RDS-EVENT-0029	DB 인스턴스가 단일 AZ DB 인스턴스로 전환되었습니다.
구성 변경	RDS-EVENT-0014	이 DB 인스턴스의 클래스가 변경되었습니다.
구성 변경	RDS-EVENT-0017	이 DB 인스턴스의 스토리지 설정이 변경되었습니다.
구성 변경	RDS-EVENT-0010	DB 인스턴스가 보안 그룹에서 제거되었습니다.
구성 변경	RDS-EVENT-0016	DB 인스턴스의 마스터 암호가 재설정되었습니다.
구성 변경	RDS-EVENT-0067	DB 인스턴스의 마스터 암호 재설정 시도가 실패했습니다.
구성 변경	RDS-EVENT-0078	Enhanced Monitoring 구성이 변경되었습니다.
생성	RDS-EVENT-0005	생성된 DB 인스턴스.
삭제	RDS-EVENT-0003	DB 인스턴스가 삭제되었습니다.

Category	Amazon RDS 이벤트 ID	설명
장애 조치	RDS-EVENT-0034	최근에 DB 인스턴스에 장애 조치가 발생하였기 때문에 Amazon RDS가 요청한 장애 조치를 실행하지 않습니다.
장애 조치	RDS-EVENT-0013	예비 인스턴스의 승격 원인이었던 다른 AZ 장애 조치가 시작되었습니다.
장애 조치	RDS-EVENT-0015	예비 인스턴스의 승격 원인이었던 다른 AZ 장애 조치가 완료되었습니다. DNS가 새로운 기본 DB 인스턴스로 이전하는 데 몇 분 걸릴 수 있습니다.
장애 조치	RDS-EVENT-0065	인스턴스가 부분적 장애 조치에서 복구되었습니다.
장애 조치	RDS-EVENT-0049	다른 AZ 장애 조치가 완료되었습니다.
장애 조치	RDS-EVENT-0050	성공적인 인스턴스 복구 후 다른 AZ 활성화가 시작되었습니다.
장애 조치	RDS-EVENT-0051	다른 AZ 활성화가 완료되었습니다. 이제 데이터베이스에 액세스할 수 있습니다.
결함	RDS-EVENT-0031	호환되지 않는 구성 또는 기본 스토리지 문제로 인해 DB 인스턴스에 장애가 발생했습니다. DB 인스턴스에 대해 특정 시점으로 복구를 시작합니다.
결함	RDS-EVENT-0036	DB 인스턴스가 호환되지 않는 네트워크에 있습니다. 특정 서브넷 ID 중 일부가 잘못되었거나 존재하지 않습니다.
결함	RDS-EVENT-0035	DB 인스턴스에 잘못된 파라미터가 있습니다. 예를 들어 이 인스턴스 클래스의 메모리 관련 파라미터가 너무 높게 설정되어 있어 DB 인스턴스가 시작되지 않는 경우 고객이 할 수 있는 작업은 메모리 파라미터를 수정하고 DB 인스턴스를 재부팅하는 것입니다.
결함	RDS-EVENT-0058	Statspack 사용자 계정인 PERFSTAT 생성 종 오류가 발생하였습니다. Statspack 옵션을 추가하기 전에 해당 계정을 삭제하십시오.
결함	RDS-EVENT-0079	Enhanced Monitoring을 활성화하려면 Enhanced Monitoring IAM 역할이 있어야 합니다. Enhanced Monitoring IAM 역할을 생성하는 방법에 대한 자세한 내용은 Amazon RDS Enhanced Monitoring에 대한 IAM 역할을 생성하려면 (p. 359) 단원을 참조하십시오.
결함	RDS-EVENT-0080	구성을 변경하는 동안 오류가 발생하여 Enhanced Monitoring이 비활성화되었습니다. Enhanced Monitoring IAM 역할이 잘못 구성된 것 같습니다. Enhanced Monitoring IAM 역할을 생성하는 방법에 대한 자세한 내용은 Amazon RDS Enhanced Monitoring에 대한 IAM 역할을 생성하려면 (p. 359) 단원을 참조하십시오.

Category	Amazon RDS 이벤트 ID	설명
결함	RDS-EVENT-0082	Aurora가 Amazon S3 버킷에서 백업 데이터를 복사할 수 없습니다. Amazon S3 버킷에 액세스하기 위한 Aurora의 권한이 잘못 구성된 것 같습니다. 자세한 내용은 Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 (p. 475) 단원을 참조하십시오.
적은 스토리지	RDS-EVENT-0089	DB 인스턴스가 할당된 스토리지의 90% 이상을 사용하였습니다. [Free Storage Space] 측정치를 사용하여 DB 인스턴스에 대한 스토리지 공간을 모니터링할 수 있습니다. 자세한 내용은 DB 인스턴스 측정치 보기 (p. 336) 단원을 참조하십시오.
적은 스토리지	RDS-EVENT-0007	DB 인스턴스에 할당된 스토리지를 모두 사용하였습니다. 이 문제를 해결하려면 DB 인스턴스에 스토리지를 추가 할당해야 합니다. 자세한 내용은 RDS FAQ 단원을 참조하십시오. [Free Storage Space] 측정치를 사용하여 DB 인스턴스에 대한 스토리지 공간을 모니터링할 수 있습니다. 자세한 내용은 DB 인스턴스 측정치 보기 (p. 336) 단원을 참조하십시오.
유지 관리	RDS-EVENT-0026	DB 인스턴스의 오프라인 유지 관리가 진행 중입니다. 따라서 현재 DB 인스턴스는 사용할 수 없습니다.
유지 관리	RDS-EVENT-0027	DB 인스턴스의 오프라인 유지 관리가 완료되었습니다. 이제 DB 인스턴스를 사용할 수 있습니다.
유지 관리	RDS-EVENT-0047	DB 인스턴스의 패치 작업이 완료되었습니다.
유지 관리	RDS-EVENT-0155	DB 인스턴스에 사용 가능한 DB 엔진 마이너 버전 업그레이드가 있습니다.
알림	RDS-EVENT-0044	연산자 관련 알림 메시지입니다. 자세한 내용은 이벤트 메시지 단원을 참조하십시오.
알림	RDS-EVENT-0048	DB 인스턴스의 패치 작업이 지연되었습니다.
알림	RDS-EVENT-0054	사용 중인 MySQL 스토리지 엔진이 InnoDB가 아닙니다. Amazon RDS는 MySQL 스토리지 엔진으로 InnoDB를 권장합니다. MySQL 스토리지 엔진에 대한 자세한 내용은 Amazon RDS MySQL에 대해 지원되는 스토리지 엔진 을 참조하십시오.
알림	RDS-EVENT-0055	DB 인스턴스의 테이블 수가 Amazon RDS의 권장 모범 사례를 초과하였습니다. DB 인스턴스의 테이블 수를 줄이십시오. 권장 모범 사례에 대한 자세한 내용은 Amazon Aurora 모범 사례 (p. 929) 단원을 참조하십시오.
알림	RDS-EVENT-0056	DB 인스턴스의 데이터베이스 수가 Amazon RDS의 권장 모범 사례를 초과하였습니다. DB 인스턴스의 데이터베이스 수를 줄이십시오. 권장 모범 사례에 대한 자세한 내용은 Amazon Aurora 모범 사례 (p. 929) 단원을 참조하십시오.
알림	RDS-EVENT-0087	DB 인스턴스가 종단되었습니다.

Category	Amazon RDS 이벤트 ID	설명
알림	RDS-EVENT-0088	DB 인스턴스가 시작되었습니다.
알림	RDS-EVENT-0154	DB 인스턴스가 중지 최대 허용 시간 초과로 인해 시작 중입니다.
알림	RDS-EVENT-0157	대상 인스턴스 클래스는 원본 DB 인스턴스에 있는 데이터베이스의 수를 지원할 수 없기 때문에 RDS는 DB 인스턴스 클래스를 수정할 수 없습니다. “인스턴스에 N 대 이터베이스가 있지만, 변환 후에는 N만 지원할 것입니다.”라는 오류 메시지가 표시됩니다.
알림	RDS-EVENT-0158	DB 인스턴스가 업그레이드할 수 없는 상태입니다.
읽기 전용 복제본	RDS-EVENT-0045	읽기 전용 복제 프로세스에서 오류가 발생하였습니다. 자세한 내용은 이벤트 메시지 단원을 참조하십시오. 읽기 전용 복제본 오류의 문제 해결에 대한 자세한 내용은 MySQL 또는 MariaDB 읽기 전용 복제본 문제 해결 을 참조하십시오.
읽기 전용 복제본	RDS-EVENT-0046	읽기 전용 복제본이 복제를 재개했습니다. 이 메시지는 읽기 전용 복제본을 처음 생성할 때 나타나거나 복제 가능한 정상 여부를 확인하는 모니터링 메시지로 나타납니다. RDS-EVENT-0045 알림 메시지 후에 이 메시지가 표시되면 오류 이후, 또는 복제가 중단되었다가 다시 시작된 것입니다.
읽기 전용 복제본	RDS-EVENT-0057	읽기 전용 복제본의 복제가 종료되었습니다.
읽기 전용 복제본	RDS-EVENT-0062	읽기 전용 복제본의 복제가 수동으로 중지되었습니다.
읽기 전용 복제본	RDS-EVENT-0063	읽기 전용 복제본의 복제가 재설정되었습니다.
복구	RDS-EVENT-0020	DB 인스턴스 복구가 시작되었습니다. 복구 시간은 복구 할 데이터 용량에 따라 달라집니다.
복구	RDS-EVENT-0021	DB 인스턴스 복구가 완료되었습니다.
복구	RDS-EVENT-0023	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 복구 할 데이터 용량에 따라 달라집니다.
복구	RDS-EVENT-0053	다중 AZ 인스턴스 복구가 완료되었습니다.
복원	RDS-EVENT-0008	DB 인스턴스가 DB 스냅샷에서 복원되었습니다.
복원	RDS-EVENT-0019	DB 인스턴스가 특정 시점으로 백업에서 복원되었습니다.

다음 표는 DB 파라미터 그룹이 소스 유형일 때 이벤트 카테고리와 이벤트 목록을 나타냅니다.

Category	RDS 이벤트 ID	설명
구성 변경	RDS-EVENT-0037	파라미터 그룹 설정이 변경되었습니다.

다음 표는 DB 보안 그룹이 소스 유형일 때 이벤트 카테고리와 이벤트 목록을 나타냅니다.

Category	RDS 이벤트 ID	설명
구성 변경	RDS-EVENT-0038	보안 그룹 설정이 변경되었습니다.
결함	RDS-EVENT-0039	[사용자]가 소유한 보안 그룹이 없습니다. 보안 그룹에 대한 권한 부여가 취소되었습니다.

다음 표에는 Aurora DB 클러스터가 원본 유형일 때 이벤트 카테고리와 이벤트 목록이 나와 있습니다.

Category	RDS 이벤트 ID	설명
장애 조치	RDS-EVENT-0069	DB 클러스터에 대한 장애 조치가 실패했습니다.
장애 조치	RDS-EVENT-0070	DB 클러스터에 대한 장애 조치가 다시 시작되었습니다.
장애 조치	RDS-EVENT-0071	DB 클러스터에 대한 장애 조치를 마쳤습니다.
장애 조치	RDS-EVENT-0072	DB 클러스터에 대한 장애 조치가 동일한 가용 영역 내에서 시작되었습니다.
장애 조치	RDS-EVENT-0073	DB 클러스터에 대한 장애 조치가 가용 영역 전체에서 시작되었습니다.
결함	RDS-EVENT-0083	Aurora가 Amazon S3 버킷에서 백업 데이터를 복사할 수 없습니다. Amazon S3 버킷에 액세스하기 위한 Aurora의 권한이 잘못 구성된 것 같습니다. 자세한 내용은 Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 (p. 475) 단원을 참조하십시오.
유지 관리	RDS-EVENT-0156	DB 클러스터에 사용 가능한 DB 엔진 마이너 버전 업그레이드가 있습니다.
알림	RDS-EVENT-0076	Aurora DB 클러스터로 마이그레이션하지 못했습니다.
알림	RDS-EVENT-0077	Aurora DB 클러스터로 마이그레이션하는 중에 원본 데이터베이스의 테이블을 InnoDB로 변환하지 못했습니다.
알림	RDS-EVENT-0141	Aurora Serverless DB 클러스터에 대해 조정이 시작되었습니다.
알림	RDS-EVENT-0142	Aurora Serverless DB 클러스터에 대해 조정이 완료되었습니다.
알림	RDS-EVENT-0143	Aurora Serverless DB 클러스터에 대해 조정이 실패하였습니다.
알림	RDS-EVENT-0144	Aurora Serverless DB 클러스터에서 자동 일시 중지가 시작되었습니다.
알림	RDS-EVENT-0145	Aurora Serverless DB 클러스터가 일시 중지되었습니다.
알림	RDS-EVENT-0146	Aurora Serverless DB 클러스터 일시 중지가 취소되었습니다.

Category	RDS 이벤트 ID	설명
알림	RDS-EVENT-0147	Aurora Serverless DB 클러스터 재개가 시작되었습니다.
알림	RDS-EVENT-0148	Aurora Serverless DB 클러스터 재개가 완료되었습니다.
알림	RDS-EVENT-0149	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.

다음 표에는 Aurora DB 클러스터 스냅샷이 원본 유형일 때 이벤트 카테고리와 이벤트 목록이 나와 있습니다.

Category	RDS 이벤트 ID	설명
백업	RDS-EVENT-0074	수동 DB 클러스터 스냅샷 생성이 시작되었습니다.
백업	RDS-EVENT-0075	수동 DB 클러스터 스냅샷이 생성되었습니다.
알림	RDS-EVENT-0162	DB 클러스터 스냅샷 내보내기 작업이 실패했습니다.
알림	RDS-EVENT-0163	DB 클러스터 스냅샷 내보내기 작업이 취소되었습니다.
알림	RDS-EVENT-0164	DB 클러스터 스냅샷 내보내기 작업이 완료되었습니다.
백업	RDS-EVENT-0168	자동화된 클러스터 스냅샷을 생성하는 중입니다.
백업	RDS-EVENT-0169	자동화된 클러스터 스냅샷이 생성되었습니다.
생성	RDS-EVENT-0170	DB 클러스터가 생성되었습니다.
삭제	RDS-EVENT-0171	DB 클러스터가 삭제되었습니다.
알림	RDS-EVENT-0172	DB 클러스터 이름이 [이전 DB 클러스터 이름]에서 [새 DB 클러스터 이름]으로 변경되었습니다.

Amazon RDS 이벤트 알림 구독

임의의 DB 인스턴스, DB 스냅샷, DB 보안 그룹 또는 DB 파라미터 그룹에 대한 이벤트 발생 여부를 알 수 있도록 Amazon RDS 알림 구독을 생성할 수 있습니다. 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. 이 두 가지는 [SourceType](소스 유형)과 [SourceIdentifier](이벤트를 발생시키는 Amazon RDS 소

스)에서 지정됩니다.SourceType 및 SourceIdentifier(예를 들면 SourceType = db-instance 및 SourceIdentifier = myDBInstance1)를 둘 다 지정하는 경우 지정된 원본에 대한 모든 DB 인스턴스 이벤트가 수신됩니다. 하지만 SourceType만 지정하고 SourceIdentifier를 지정하지 않으면 모든 Amazon RDS 원본 중 해당 원본 유형의 이벤트만 알림 메시지로 받게 됩니다. SourceType 또는 SourceIdentifier를 지정하지 않으면, 고객 계정에 속하는 모든 Amazon RDS 원본에서 생성된 이벤트에 대한 알림을 수신합니다.

Note

이벤트 알림을 전달하는 데 최대 5분 정도 걸릴 수 있습니다.

Amazon RDS 이벤트 알림은 암호화되지 않은 SNS 주제에만 사용할 수 있습니다. 암호화된 SNS 주제를 지정하면 해당 주제에 대한 이벤트 알림이 전송되지 않습니다.

콘솔

RDS 이벤트 알림 구독 방법

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 이벤트 구독을 선택합니다.
3. [Event subscriptions] 창에서 [Create event subscription]을 선택합니다.
4. [Create event subscription] 대화 상자에서 다음과 같이 실행합니다.
 - a. 이름에서 이벤트 알림 구독 이름을 입력합니다.
 - b. 알림 받을 대상에서 Amazon SNS 주제의 기존 Amazon SNS ARN을 선택하거나 주제 생성을 선택하여 주제와 수신자 목록의 이름을 입력합니다.
 - c. [Source type]에서 원본 형식을 선택합니다.
 - d. [Yes]를 선택하여 구독을 활성화합니다. 구독만 생성하고 알림 메시지 전송은 아직 원하지 않을 경우에는 [No]를 선택합니다.
 - e. 선택한 소스 유형에 따라 이벤트 알림 메시지를 수신하고자 하는 이벤트 카테고리와 소스를 선택합니다.
 - f. Create를 선택합니다.

Amazon RDS 콘솔에 현재 구독 생성 중으로 나옵니다.

Name	Status	Source Type	Enabled
Configchangerspgres	active	Instances	Yes
Test	creating	Instances	Yes

AWS CLI

RDS 이벤트 알림을 구독하려면 AWS CLI `create-event-subscription` 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

- `--subscription-name`
- `--sns-topic-arn`

Example

Linux, OS X, Unix의 경우:

```
aws rds create-event-subscription \
--subscription-name myeventsSubscription \
--sns-topic-arn arn:aws:sns:us-east-1:802#####:myawsuser-RDS \
--enabled
```

Windows의 경우:

```
aws rds create-event-subscription ^
--subscription-name myeventsSubscription ^
--sns-topic-arn arn:aws:sns:us-east-1:802#####: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 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 콘솔에 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.

Event subscriptions (2)		
	Name	Status
<input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Create event subscription"/>		
<input type="text" value="Filter event subscriptions"/>		
< 1 >	Filter	Clear
	Name	Status
<input type="checkbox"/>	Configchangerdpgres	<input checked="" type="checkbox"/> active
<input type="checkbox"/>	Postgresnotification	<input checked="" type="checkbox"/> active

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 myfirsteventsubscription을 설명하는 예제입니다.

```
aws rds describe-event-subscriptions --subscription-name myfirsteventsubscription
```

API

현재 Amazon RDS 이벤트 알림 구독을 표시하려면 Amazon RDS API [DescribeEventSubscriptions](#) 작업을 사용합니다.

Example

다음은 최대 100개의 이벤트 구독을 나열하는 코드 예제입니다.

```
https://rds.us-east-1.amazonaws.com/  
?Action=DescribeEventSubscriptions  
&MaxRecords=100  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4  
&Version=2014-10-31  
&X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=AKIADQKE4SARGYLE/20140428/us-east-1/rds/aws4_request  
&X-Amz-Date=20140428T161907Z  
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date  
&X-Amz-Signature=4208679fe967783ala149c826199080a066085d5a88227a80c6c0cadb3e8c0d4
```

다음은 myfirsteventsubscription을 설명하는 예제입니다.

```
https://rds.us-east-1.amazonaws.com/  
?Action=DescribeEventSubscriptions  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4  
&SubscriptionName=myfirsteventsubscription  
&Version=2014-10-31  
&X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=AKIADQKE4SARGYLE/20140428/us-east-1/rds/aws4_request  
&X-Amz-Date=20140428T161907Z  
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date  
&X-Amz-Signature=4208679fe967783ala149c826199080a066085d5a88227a80c6c0cadb3e8c0d4
```

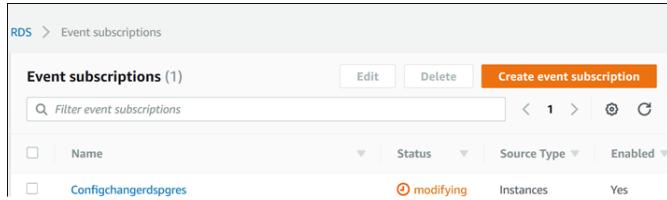
Amazon RDS 이벤트 알림 구독 변경

구독을 생성한 후에는 구독 이름, 소스 식별자, 카테고리 또는 주제 ARN을 변경할 수 있습니다.

콘솔

Amazon RDS 이벤트 알림 구독을 변경하려면

1. AWS Management 콘솔에 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. [Edit]를 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 `myeventsSubscription`을 사용할 수 있습니다.

Linux, OS X, Unix의 경우:

```
aws rds modify-event-subscription \
--subscription-name myeventsSubscription \
--enabled
```

Windows의 경우:

```
aws rds modify-event-subscription ^
--subscription-name myeventsSubscription ^
--enabled
```

API

Amazon RDS 이벤트를 수정하려면 Amazon RDS API 작업 [ModifyEventSubscription](#)을 호출합니다. 다음 필수 파라미터를 포함합니다.

- `SubscriptionName`

Amazon RDS 이벤트 알림 구독에 대한 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 변경 \(p. 439\)](#) 단원을 참조하십시오.

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 `myrdseventsSubscription` 구독에 소스 식별자 `mysqlDb`를 추가합니다.

Linux, OS X, Unix의 경우:

```
aws rds add-source-identifier-to-subscription \
--subscription-name myrdseventsSubscription \
--source-identifier mysqlDb
```

Windows의 경우:

```
aws rds add-source-identifier-to-subscription ^
--subscription-name myrdseventsSubscription ^
--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 구독의 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 변경 \(p. 439\)](#) 단원을 참조하십시오.

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 `myrdseventsSubscription` 구독에서 소스 식별자 `mysqlDb`를 제거합니다.

Linux, OS X, Unix의 경우:

```
aws rds remove-source-identifier-from-subscription \
--subscription-name myrdseventsSubscription \
--source-identifier mysqlDb
```

Windows의 경우:

```
aws rds remove-source-identifier-from-subscription ^
--subscription-name myrdsseventssubscription ^
--source-identifier mysqladb
```

API

Amazon RDS 이벤트 알림 구독에서 소스 식별자를 제거하려면 Amazon RDS API [RemoveSourceIdentifierFromSubscription](#) 명령을 사용합니다. 다음 필수 파라미터를 포함합니다.

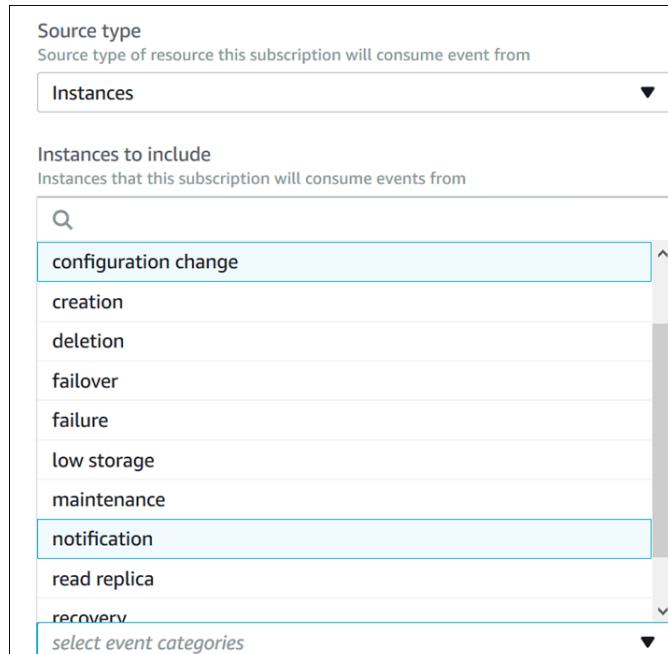
- `SubscriptionName`
- `SourceIdentifier`

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 변경 \(p. 439\)](#) 단원을 참조하십시오.



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 콘솔에 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. 삭제를 선택합니다.
5. Amazon RDS 콘솔에 현재 구독 삭제 중으로 나옵니다.

The screenshot shows the 'Event subscriptions (2)' section in the AWS Management Console. At the top, there are three buttons: 'Edit', 'Delete' (which is circled in red), and 'Create event subscription'. Below these are search and filter options. The main area displays a table with two rows:

Name	Status
Configchangerdpgres	active
Postgresnotification	active

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 RDS 이벤트 보기

Amazon RDS에서는 DB 인스턴스, DB 스냅샷, DB 보안 그룹 및 DB 파라미터 그룹과 관련된 이벤트 레코드를 유지합니다. 여기에는 이벤트 날짜 및 시간, 이벤트의 원본 이름 및 유형, 이벤트 관련 메시지 등의 정보가 포함됩니다.

지난 24시간 동안 발생한 이벤트를 표시하는 AWS Management 콘솔을 통해 RDS 리소스에 대한 이벤트를 검색할 수 있습니다. [describe-events](#) AWS CLI 명령 또는 [DescribeEvents](#) RDS API 작업을 사용하여 RDS 리소스에 대한 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 CloudWatch 이벤트에 Amazon RDS 이벤트를 보낼 수 있습니다. 자세한 내용은 [Amazon Aurora에 대한 CloudWatch 이벤트 및 Amazon EventBridge 이벤트 가져오기](#) (p. 445) 단원을 참조하십시오.

콘솔

지난 24시간 동안 발생한 모든 Amazon RDS 인스턴스 이벤트를 보려면

1. AWS Management 콘솔에 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. [Filter] 목록을 사용하여 이벤트를 유형별로 필터링하고 [Filter] 목록 오른쪽에 있는 텍스트 상자를 사용하여 결과를 세부적으로 필터링할 수 있습니다. 예를 들어 다음 스크린샷에서는 DB 인스턴스 이벤트 유형별로 필터링되고 문자 **1318**을 포함된 이벤트 목록을 보여 줍니다.

Identifier	Type
feb1318	DB cluster snapshots
feb1318	DB cluster snapshots

AWS CLI

지난 7일 동안 발생한 모든 Amazon RDS 인스턴스 이벤트를 보려면

[describe-events](#) AWS CLI 명령을 호출하고 --duration 파라미터를 10080으로 설정하여 지난 7일 동안 발생한 모든 Amazon RDS 인스턴스 이벤트를 볼 수 있습니다.

```
aws rds describe-events --duration 10080
```

API

지난 14일 동안 발생한 모든 Amazon RDS 인스턴스 이벤트를 보려면

DescribeEvents RDS API 작업을 호출하고 Duration 파라미터를 20160으로 설정하여 지난 14일 동안 발생한 모든 Amazon RDS 인스턴스 이벤트를 볼 수 있습니다.

```
https://rds.us-west-2.amazonaws.com/  
?Action=DescribeEvents  
&Duration=20160  
&MaxRecords=100  
&SignatureMethod=HmacSHA256  
&SignatureVersion=4  
&Version=2014-10-31  
&X-Amz-Algorithm=AWS4-HMAC-SHA256  
&X-Amz-Credential=AKIADQKE4SARGYLE/20140421/us-west-2/rds/aws4_request  
&X-Amz-Date=20140421T194733Z  
&X-Amz-SignedHeaders=content-type;host;user-agent;x-amz-content-sha256;x-amz-date  
&X-Amz-Signature=8e313cabcd9766c56a2886b5b298fd944e0b7cfa248953c82705fdd0374f27
```

Amazon Aurora에 대한 CloudWatch 이벤트 및 Amazon EventBridge 이벤트 가져오기

Amazon CloudWatch Events 및 Amazon EventBridge로 AWS 서비스를 자동화하여 애플리케이션 활용 성 문제나 리소스 변경 같은 시스템 이벤트에 대응할 수 있습니다. AWS 서비스 이벤트는 거의 실시간으로 CloudWatch 이벤트 및 EventBridge로 전송됩니다. 어떤 이벤트에 관심이 있으며 이벤트가 규칙과 일치할 때 어떤 자동화된 작업을 수행할지를 표시하는 간단한 규칙을 작성할 수 있습니다.

AWS Lambda 함수나 Amazon SNS 주제 등 다양한 대상을 설정하고 JSON 형식으로 된 이벤트를 받을 수 있습니다. 자세한 내용은 [Amazon CloudWatch Events 사용 설명서](#) 및 [Amazon EventBridge 사용 설명서](#)를 참조하십시오.

예를 들어, DB 인스턴스가 생성되거나 삭제될 때마다 CloudWatch 이벤트 또는 Amazon EventBridge에 이벤트를 전송하도록 Amazon Aurora를 구성할 수 있습니다.

주제

- [CloudWatch 이벤트로 Amazon RDS 이벤트 전송 \(p. 445\)](#)
- [DB 클러스터 이벤트 \(p. 446\)](#)
- [DB 인스턴스 이벤트 \(p. 446\)](#)
- [DB 파라미터 그룹 이벤트 \(p. 447\)](#)
- [DB 보안 그룹 이벤트 \(p. 447\)](#)
- [DB 클러스터 스냅샷 이벤트 \(p. 448\)](#)
- [DB 스냅샷 이벤트 \(p. 448\)](#)

CloudWatch 이벤트로 Amazon RDS 이벤트 전송

Amazon RDS 이벤트를 CloudWatch 이벤트로 전송하기 위한 CloudWatch 이벤트 규칙을 생성할 수 있습니다.

다음 단계를 사용하여 AWS 서비스에서 발생하는 이벤트에서 트리거되는 CloudWatch 이벤트 규칙을 생성합니다.

이벤트에서 트리거되는 규칙을 생성하려면

1. <https://console.aws.amazon.com/cloudwatch/>에서 CloudWatch 콘솔을 엽니다.

2. 탐색 창의 이벤트 아래에서 규칙을 선택합니다.
3. [Create rule]을 선택합니다.
4. [Event Source]에서 다음을 수행합니다.
 - a. 이벤트 패턴을 선택합니다.
 - b. 서비스 이름에서 Relational Database Service(RDS)를 선택합니다.
 - c. 이벤트 유형에서 이벤트를 트리거할 Amazon RDS 리소스 유형을 선택합니다. 예를 들어, DB 인스턴스가 이벤트를 트리거하는 경우 RDS DB Instance Event(RDS DB 인스턴스 이벤트)를 선택합니다.
5. 대상에서 Add Target(대상 추가)를 선택한 다음 CloudWatch 로그 그룹을 선택합니다.
6. 로그 그룹에 이벤트를 저장할 로그 그룹의 이름을 입력합니다.
7. 세부 정보 구성을 선택합니다. 규칙 정의에 규칙의 이름과 규칙에 대한 설명을 입력합니다.
8. 규칙 생성을 선택합니다.

DB 클러스터 이벤트

다음은 DB 클러스터 엔드포인트 예제입니다.

```
{  
  "version": "0",  
  "id": "844e2571-85d4-695f-b930-0153b71dcba42",  
  "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",  
    "SourceIdentifier": "rds:my-db-cluster",  
    "Message": "Database cluster has been patched"  
  }  
}
```

DB 인스턴스 이벤트

다음은 DB 인스턴스 이벤트의 예제입니다.

```
{  
  "version": "0",  
  "id": "68f6e973-1a0c-d37b-f2f2-94a7f62ffd4e",  
  "detail-type": "RDS DB Instance Event",  
  "source": "aws.rds",  
  "account": "123456789012",  
  "time": "2018-09-27T22:36:43Z",  
  "region": "us-east-1",  
  "resources": [  
    "arn:aws:rds:us-east-1:123456789012:instance:my-db-instance"  
  ],  
  "detail": {  
    "EventCategories": [  
      "notification"  
    ],  
    "SourceType": "INSTANCE",  
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:instance:my-db-instance",  
    "Date": "2018-09-27T22:36:43.882Z",  
    "SourceIdentifier": "rds:my-db-instance",  
    "Message": "Database instance has been patched"  
  }  
}
```

```
"resources": [
    "arn:aws:rds:us-east-1:123456789012:db:my-db-instance"
],
"detail": {
    "EventCategories": [
        "failover"
    ],
    "SourceType": "DB_INSTANCE",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:db:my-db-instance",
    "Date": "2018-09-27T22:36:43.292Z",
    "SourceIdentifier": "rds:my-db-instance",
    "Message": "A Multi-AZ failover has completed."
}
}
```

DB 파라미터 그룹 이벤트

다음은 DB 파라미터 그룹 이벤트의 예제입니다.

```
{
    "version": "0",
    "id": "844e2571-85d4-695f-b930-0153b71dc42",
    "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",
        "SourceIdentifier": "rds:my-db-param-group",
        "Message": "Updated parameter time_zone to UTC with apply method immediate"
    }
}
```

DB 보안 그룹 이벤트

다음은 DB 보안 그룹 이벤트의 예제입니다.

```
{
    "version": "0",
    "id": "844e2571-85d4-695f-b930-0153b71dc42",
    "detail-type": "RDS DB Security 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:secgrp:my-security-group"
    ],
}
```

```
"detail": {  
    "EventCategories": [  
        "configuration change"  
    ],  
    "SourceType": "SECURITY_GROUP",  
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:secgrp:my-security-group",  
    "Date": "2018-10-06T12:26:13.882Z",  
    "SourceIdentifier": "rds:my-security-group",  
    "Message": "Applied change to security group"  
}  
}
```

DB 클러스터 스냅샷 이벤트

다음은 DB 클러스터 스냅샷 이벤트의 예제입니다.

```
{  
    "version": "0",  
    "id": "844e2571-85d4-695f-b930-0153b71dc42",  
    "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": "rds:my-db-cluster-snapshot",  
        "Message": "Creating manual cluster snapshot"  
    }  
}
```

DB 스냅샷 이벤트

다음은 DB 스냅샷 이벤트의 예제입니다.

```
{  
    "version": "0",  
    "id": "844e2571-85d4-695f-b930-0153b71dc42",  
    "detail-type": "RDS DB 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:snapshot:rds:my-db-snapshot"  
    ],  
    "detail": {  
        "EventCategories": [  
    }
```

```
        "deletion"
    ],
    "SourceType": "SNAPSHOT",
    "SourceArn": "arn:aws:rds:us-east-1:123456789012:snapshot:rds:my-db-snapshot",
    "Date": "2018-10-06T12:26:13.882Z",
    "SourceIdentifier": "rds:my-db-snapshot",
    "Message": "Deleted manual snapshot"
}
}
```

Amazon Aurora 데이터베이스 로그 파일

Amazon RDS 콘솔, AWS Command Line Interface(AWS CLI) 또는 Amazon RDS API를 사용하여 데이터베이스 로그를 보고 다운로드하고 조사할 수 있습니다. 트랜잭션 로그 보기, 다운로드 또는 조사는 지원되지 않습니다.

엔진별 정보는 다음을 참조하십시오.

- MySQL 데이터베이스 로그 파일 (p. 452)
- PostgreSQL 데이터베이스 로그 파일 (p. 456)

Note

경우에 따라 로그에 숨겨진 데이터가 포함됩니다. 따라서 AWS Management 콘솔은 콘텐츠를 로그 파일에 표시할 수 있지만 다운로드할 때 로그 파일이 비어 있을 수 있습니다.

데이터베이스 로그 파일 보기 및 나열

Amazon RDS 콘솔을 사용하여 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.

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 [Logs] 섹션을 찾습니다.
6. 로그 섹션에서 표시할 로그를 선택한 다음 보기를 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

Amazon RDS 콘솔, 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

Linux, OS X, Unix의 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
```

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

Amazon RDS 콘솔을 사용하여 로그 파일의 내용을 모니터링할 수 있습니다.

콘솔

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 보기 를 선택합니다.

Amazon CloudWatch Logs에 데이터베이스 로그 게시

DB 인스턴스 로그를 보고 다운로드하는 것 외에도 Amazon CloudWatch Logs에 로그를 게시할 수 있습니다. CloudWatch Logs에서는 로그 데이터에 대한 실시간 분석을 수행하고, 매우 내구력 있는 스토리지에 데이터를 저장하며, CloudWatch Logs 에이전트를 사용하여 데이터를 관리할 수 있습니다. 보존 기간을 지정하지 않는 한 AWS는 CloudWatch Logs에 게시된 로그 데이터를 무기한 보존합니다. 자세한 내용은 [CloudWatch Logs에서 로그 데이터 보존 기간 변경](#)을 참조하십시오.

엔진별 정보는 다음을 참조하십시오.

- the section called “CloudWatch Logs에 Aurora MySQL 로그 게시” (p. 639)
- the section called “CloudWatch Logs에 Aurora PostgreSQL 로그 게시” (p. 847)

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///////////
WEaOAIXLhngC5zp9CyB1R6abwKrXHVR5efnAVN3XvR7IwqKYalFSn6UyJuEFTft9nObglx4QJ+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)

MySQL 데이터베이스 로그 파일

MySQL 오류 로그, 느린 쿼리 로그 및 일반 로그를 모니터링할 수 있습니다. MySQL 오류 로그는 기본적으로 생성됩니다. DB 파라미터 그룹에서 파라미터를 설정하여 느린 쿼리와 일반 로그를 생성할 수 있습니다. Amazon RDS는 모든 MySQL 로그 파일을 교체하며 각 유형에 대한 간격은 다음에 제공됩니다.

Amazon RDS 콘솔, Amazon RDS API, AWS CLI 또는 AWS SDK를 통해 MySQL 로그를 직접 모니터링할 수 있습니다. 또한, 주 데이터베이스에 있는 데이터베이스 테이블로 로그를 전송하고 그 테이블을 쿼리하여 MySQL 로그에 액세스할 수 있습니다. mysqlbinlog 유ти리티를 사용하여 이진 로그를 다운로드할 수 있습니다.

파일 기반 데이터베이스 로그 보기, 다운로드 및 조사 방법에 대한 자세한 내용은 [Amazon Aurora 데이터베이스 로그 파일 \(p. 449\)](#) 단원을 참조하십시오.

MySQL 오류 로그 액세스

MySQL 오류 로그는 `mysql-error.log` 파일에 기록됩니다. Amazon RDS 콘솔을 사용하거나 Amazon RDS API, Amazon RDS CLI 또는 AWS SDK를 사용하여 로그를 검색하면 `mysql-error.log`를 볼 수 있습니다. `mysql-error.log`는 5분마다 플러시되며 해당 콘텐츠가 `mysql-error-running.log`에 추가됩니다. 그런 다음, `mysql-error-running.log` 파일은 1시간마다 교체되고 마지막 30일 동안 생성된 시간별 파일이 보존됩니다. 보존 기간은 Amazon RDS와 Aurora 간에 다릅니다.

각 로그 파일이 생성된 시간(UTC)이 파일 이름에 추가됩니다. 로그 파일에는 타임스탬프도 포함되어 있어, 로그 항목이 작성된 시간을 확인하는데 도움이 됩니다.

MySQL에서는 시작, 종료 및 오류 발생 시에만 오류 로그에 데이터가 기록됩니다. 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.

MySQL 느린 쿼리 및 일반 로그 액세스

DB 파라미터 그룹에서 파라미터를 설정하면 MySQL 느린 쿼리 로그와 일반 로그가 파일이나 데이터베이스 테이블에 기록될 수 있습니다. DB 파라미터 그룹의 생성 및 변경에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 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 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 로그에 기록할 쿼리의 최단 실행 시간 값(초)을 지정하십시오. 기본값은 10초이고, 최소값은 0초입니다. `log_output = FILE` 인 경우에는 마이크로초 단위까지 부동 소수점 값을 지정할 수 있습니다. `log_output = TABLE`인 경우에는 초 단위로 정수 값을 지정해야 합니다. 실행 시간이 `long_query_time` 값을 초과하는 쿼리만 로그에 기록됩니다. 예를 들어 `long_query_time`을 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. 기본값은 0입니다. 인덱스를 사용하지 않는 쿼리는 실행 시간이 `long_query_time` 파라미터의 값보다 짧아도 로그에 기록됩니다.
- `log_output option`: `log_output` 파라미터에 대해 다음 옵션 중 하나를 지정할 수 있습니다.
 - TABLE(기본값)- `mysql.general_log` 테이블에는 일반 쿼리를, `mysql.slow_log` 테이블에는 느린 쿼리를 씁니다.
 - FILE- 파일 시스템에 일반 쿼리 로그와 느린 쿼리 로그를 모두 씁니다. 로그 파일은 매시간 순환됩니다.
 - NONE- 로깅을 비활성화합니다.

로깅을 사용하는 경우, Amazon RDS는 테이블 로그를 순환하거나 로그 파일을 정기적으로 삭제합니다. 이러한 예방 조치를 취하면 데이터베이스 사용에 방해가 되거나 성능에 영향을 미치는 큰 로그 파일이 생성될 가능성을 줄일 수 있습니다. `FILE` 및 `TABLE` 로깅 접근 방식 교체 및 삭제는 다음과 같습니다.

- `FILE` 로깅을 사용하는 경우, 로그 파일은 매시간 검사되며 24시간 이상 지난 로그 파일은 삭제됩니다. 경우에 따라 삭제 후 나머지 로그 파일의 총 크기가 DB 인스턴스에 할당된 공간 중 2%의 임계값을 초과할 수 있습니다. 이러한 경우 로그 파일의 전체 크기가 임계값 이하로 작아질 때까지 가장 큰 로그 파일부터 차례대로 삭제됩니다.
- `TABLE` 로깅이 활성화된 경우 경우에 따라 로그 테이블이 24시간마다 순환됩니다. 테이블 로그에서 사용되는 공간이 할당된 스토리지 공간 중 20% 이상을 차지하거나 모든 로그의 총 크기가 10GB 이상일 경우 이 순환이 발생합니다. DB 인스턴스에 대해 사용된 공간의 양이 DB 인스턴스의 할당된 스토리지 공간 중 90% 이상을 차지할 경우, 로그 순환을 위한 임계값은 줄어듭니다. 테이블 로그에서 사용되는 공간이 할당된 스토리지 공간 중 10% 이상을 차지하거나 모든 로그의 총 크기가 5GB 이상일 경우, 로그 테이블은 순환됩니다. 공간 확보를 위해 로그 테이블이 순환되면 알림을 받으려면 `low_free_storage` 이벤트를 구독할 수 있습니다. 자세한 내용은 [Amazon RDS 이벤트 알림 서비스 사용 \(p. 429\)](#) 단원을 참조하십시오.

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_backup`으로 지정됩니다. `mysql.slow_log` 테이블에 대한 백업 로그 테이블 이름은 `mysql.slow_log_backup`으로 지정됩니다.

`mysql.rds_rotate_general_log` 절차를 호출하면 `mysql.general_log` 테이블을 순환할 수 있습니다. `mysql.rds_rotate_slow_log` 절차를 호출하면 `mysql.slow_log` 테이블을 순환할 수 있습니다.

데이터베이스 버전 업그레이드가 진행되는 동안 테이블 로그가 순환됩니다.

Amazon RDS 콘솔, Amazon RDS API, Amazon RDS CLI 또는 AWS SDK에서 로그를 사용하여 작업하려면 `log_output` 파라미터를 `FILE`로 설정합니다. MySQL 오류 로그와 같이, 이런 로그 파일은 매시간 순환됩니다. 이전 72시간 동안 생성된 로그 파일이 보존됩니다. 보존 기간은 Amazon RDS와 Aurora 간에 다릅니다.

느린 쿼리 및 일반 로그에 대한 자세한 내용은 MySQL 문서에서 다음 항목을 참조하십시오.

- [The Slow Query Log](#)

- The General Query Log

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 \(p. 639\)](#) 단원을 참조하십시오.

로그 파일 크기

MySQL 느린 쿼리 로그, 오류 로그 및 일반 로그 파일 크기는 DB 인스턴스에 대해 할당된 스토리지 공간의 2% 이하로 제한됩니다. 이 임계값을 유지하기 위해 로그는 매시간 자동으로 순환되면서 24시간 이상 지난 로그 파일은 제거됩니다. 오래된 로그 파일을 제거한 후 로그 파일의 총 크기가 임계값을 초과하는 경우에는 로그 파일의 전체 크기가 임계값 이하로 작아질 때까지 가장 큰 로그 파일부터 차례대로 삭제됩니다.

MySQL의 경우 다시 실행 로그에 기록되는 BLOB에 대한 크기 제한이 있습니다. 이러한 제한을 감안하려면 MySQL DB 인스턴스에 대한 `innodb_log_file_size` 파라미터가 테이블에서 발견된 BLOB 데이터의 최대 크기 및 동일한 테이블에서 다른 가변 길이 필드(VARCHAR, VARBINARY, TEXT)의 길이 보다 10배 커야 합니다. 파라미터 값 설정 방법에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오. 다시 실행 로그 BLOB 크기 제한에 대한 자세한 내용은 [MySQL 5.6.20의 변경 내용](#) 단원을 참조하십시오.

테이블 기반 MySQL 로그 관리

DB 파라미터 그룹을 만들고 `log_output` 서버 파라미터를 `TABLE`로 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. 이를 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 RDS에서는 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.

이진 로깅 형식

Amazon RDS에서 MySQL은 MySQL 버전 5.6 이상에 대해 행 기반, 설명문 기반 및 혼합 바이너리 로깅 형식을 지원합니다. 기본 바이너리 로깅 형식은 혼합입니다. MySQL 버전 5.1 및 5.5를 실행하는 DB 인스턴스의 경우 혼합 이진 로깅만 지원됩니다. 다양한 MySQK 바이너리 로그 형식에 대한 자세한 내용은 MySQL 설명서에서 [바이너리 로깅 형식](#)을 참조하십시오.

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 Replication](#)을 참조하십시오.

Important

이진 로깅 형식을 행 기반으로 설정하면 이진 로그 파일이 매우 커질 수 있습니다. 큰 이진 로그 파일은 DB 인스턴스에 사용할 수 있는 스토리지의 양을 줄이므로, DB 인스턴스의 복원 작업 수행에 필요한 시간이 늘어날 수 있습니다.

설명문 기반 복제는 원본 DB 인스턴스와 읽기 전용 복제본 간의 불일치를 초래할 수 있습니다. 자세한 내용은 MySQL 설명서의 [Determination of Safe and Unsafe Statements in Binary Logging](#)을 참조하십시오.

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 인스턴스에 사용되는 파라미터 그룹을 선택합니다.

기본 파라미터 그룹을 수정할 수 없습니다. DB 인스턴스에서 기본 파라미터 그룹을 사용 중인 경우 새 파라미터 그룹을 생성하여 DB 인스턴스와 연결합니다.

DB 파라미터 그룹에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

4. 파라미터 그룹 작업에서 편집을 선택합니다.
5. binlog_format 파라미터를 선택한 바이너리 로깅 형식(ROW, STATEMENT 또는 MIXED)으로 설정합니다.
6. 변경 내용 저장을 선택하여 업데이트를 DB 파라미터 그룹에 저장합니다.

Important

`default.mysql5.6`, `default.mysql5.7`, `default.mysql8.0` DB 파라미터 그룹을 변경하면 그 파라미터 그룹을 사용하는 모든 MySQL 버전 DB 인스턴스에 변경 내용이 적용됩니다. 한 AWS 리전에서 다양한 MySQL 5.6, 5.7, 8.0 DB 인스턴스에 대해 각기 다른 이진 로깅 형식을 지정하려면 자체 DB 파라미터 그룹을 만들어야 합니다. 이 파라미터 그룹은 서로 다른 로깅 형식을 식별하고 그 DB 파라미터 그룹을 의도한 DB 인스턴스에 할당합니다.

MySQL 이진 로그 액세스

`mysqlbinlog` 유ти리티를 사용하여 MySQL 5.6 이상을 실행하는 Amazon RDS 인스턴스로부터 이진 로그를 다운로드하거나 스트리밍할 수 있습니다. 이진 로그를 로컬 컴퓨터로 다운로드하면 `mysql` 유ти리티를 사용하여 로그를 재생하는 것과 같은 작업을 수행할 수 있습니다. `Mysqlbinlog` 유ти리티 사용에 대한 자세한 내용은 [Using mysqlbinlog to Back Up Binary Log Files](#)를 참조하십시오.

Amazon RDS 인스턴스에 대해 `mysqlbinlog` 유ти리티를 실행하려면 다음 옵션을 사용합니다.

- `--read-from-remote-server` 옵션을 지정합니다.
- `--host`: 인스턴스의 엔드포인트에서 DNS 이름을 지정합니다.
- `--port`: 인스턴스에서 사용되는 포트를 지정합니다.
- `--user`: 복제 슬레이브 권한이 부여된 MySQL 사용자를 지정합니다.
- `--password`: 사용자의 암호를 지정하거나, 유ти리티에서 암호 입력을 요구하는 메시지가 표시되도록 암호 값을 생략합니다.
- 파일이 이진 형식으로 다운로드되도록 하려면 `--raw` 옵션을 지정합니다.
- `--result-file`: 원시 출력을 수신할 로컬 파일을 지정합니다.
- 하나 이상의 이진 로그 파일의 이름을 지정합니다. 사용 가능한 로그의 목록을 획득하려면 SQL 명령 `SHOW BINARY LOGS`를 사용합니다.
- 이진 로그 파일을 스트리밍하려면 `--stop-never` 옵션을 지정합니다.

Mysqlbinlog 옵션에 대한 자세한 내용은 [mysqlbinlog - Utility for Processing Binary Log Files](#) 단원을 참조하십시오.

예를 들어 다음을 참조하십시오.

Linux, OS X, Unix의 경우:

```
mysqlbinlog \
--read-from-remote-server \
--host=MySQL56Instance1.cg034hpkmmt.region.rds.amazonaws.com \
--port=3306 \
--user ReplUser \
--password \
--raw \
--result-file=/tmp/ \
binlog.00098
```

Windows의 경우:

```
mysqlbinlog ^
--read-from-remote-server ^
--host=MySQL56Instance1.cg034hpkmmt.region.rds.amazonaws.com ^
--port=3306 ^
--user ReplUser ^
--password ^
--raw ^
--result-file=/tmp/ ^
binlog.00098
```

Amazon RDS는 보통은 최대한 빨리 이진 로그를 제거하지만, mysqlbinlog가 액세스할 수 있도록 인스턴스에서 이진 파일을 여전히 사용할 수 있어야 합니다. RDS가 이진 파일을 보존할 시간을 지정하려면 `mysql.rds_set_configuration` 저장 프로시저를 사용하고 로그를 다운로드하기에 충분한 시간으로 기간을 지정합니다. 보존 기간을 설정한 후, DB 인스턴스의 스토리지 사용량을 모니터링하여 보존된 이진 로그가 너무 많은 스토리지를 차지하지 않도록 합니다.

Note

`mysql.rds_set_configuration` 저장 프로시저는 MySQL 5.6 이상 버전에서만 사용할 수 있습니다.

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;
```

PostgreSQL 데이터베이스 로그 파일

Amazon RDS for PostgreSQL에서는 쿼리 및 오류 로그를 생성합니다. RDS PostgreSQL은 autovacuum 정보 및 `rds_admin` 작업을 오류 로그에 작성합니다. PostgreSQL은 로그 연결, 연결 해제 및 체크포인트를 오류 로그에 기록합니다. 자세한 내용은 PostgreSQL 문서에서 [Error Reporting and Logging](#)을 참조하십시오.

DB 인스턴스에 대한 로깅 파라미터를 설정하려면 DB 파라미터 그룹에서 파라미터를 설정하고 해당 파라미터 그룹을 DB 인스턴스에 연결합니다. 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업](#) (p. 168) 단원을 참조하십시오.

주제

- [로그 보존 기간 설정 \(p. 457\)](#)
- [로그 파일 회전 설정 \(p. 457\)](#)
- [쿼리 로깅 사용 \(p. 457\)](#)

로그 보존 기간 설정

시스템 로그의 보존 기간을 설정하려면 `rds.log_retention_period` 파라미터를 사용합니다. DB 인스턴스와 연결된 DB 파라미터 그룹에서 `rds.log_retention_period`를 찾을 수 있습니다. 이 파라미터의 단위는 분입니다. 예를 들어 1,440으로 설정하면 로그가 하루 동안 유지됩니다. 기본값은 4,320(3일)입니다. 최대값은 10,080(7일)입니다. 인스턴스에는 보존되는 로그 파일을 포함하기에 충분하게 할당된 스토리지가 있어야 합니다.

DB 인스턴스의 스토리지가 임계값에 도달하면 Amazon Aurora가 이전 PostgreSQL 로그를 압축합니다. Aurora는 gzip 압축 유ти리티를 사용하여 파일을 압축합니다. gzip에 대한 자세한 내용은 [gzip 웹 사이트](#)를 참조하십시오. 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.

Note

스토리지가 너무 부족해지면 Aurora는 보존 기간이 만료되기 전에 압축된 PostgreSQL 로그를 삭제할 수 있습니다. 로그가 일찍 삭제되면 다음과 같은 메시지가 표시됩니다.

The oldest PostgreSQL log files were deleted due to local storage constraints.

이전 로그를 유지하려면 Amazon CloudWatch Logs에 게시합니다. 자세한 내용은 [Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시 \(p. 847\)](#) 단원을 참조하십시오. CloudWatch 게시를 설정하면 로그가 CloudWatch Logs에 게시될 때까지 Aurora에서 해당 로그를 삭제하지 않습니다.

로그 파일 회전 설정

PostgreSQL 로그 파일 회전을 제어하려면 DB 인스턴스와 연결된 DB 파라미터 그룹에서 `log_rotation_age` 및 `log_rotation_size`의 두 파라미터를 설정하십시오. 이 두 설정은 별도의 새로그 파일이 생성되는 시기를 제어합니다. 로그 파일 이름은 `log_filename` 파라미터의 파일 이름 패턴을 기반으로 합니다.

시간에 따라 로그 파일 회전을 제어하려면 `log_rotation_age` 파라미터를 1분에서 1,440분(24시간) 사이로 설정합니다. `log_rotation_age` 기본값은 60분입니다. `log_rotation_age` 파라미터를 60분 미만으로 설정한 경우 `log_filename` 파라미터도 분 형식(`postgresql.log.%Y-%m-%d-%H%M`)으로 설정합니다.

파일 크기를 기반으로 로그 파일 회전을 제어하려면 `log_rotation_size` 파라미터를 50,000~1,000,000KB 사이로 설정합니다. 기본값은 100,000KB입니다. `log_filename` 파라미터도 분 형식으로 설정하는 것이 좋습니다. 이렇게 하면 `log_rotation_age` 파라미터가 60분 이상인 경우 1시간 이내에 새 로그 파일을 생성할 수 있습니다.

쿼리 로깅 사용

PostgreSQL DB 인스턴스에 대한 쿼리 로깅을 활성화하려면 DB 인스턴스와 연결된 DB 파라미터 그룹에서 `log_statement` 및 `log_min_duration_statement`의 두 파라미터를 설정합니다.

`log_statement` 파라미터는 어떤 SQL 문이 로그에 기록되는지 제어합니다. DB 인스턴스에서 문제를 디버깅할 때 모든 문을 기록하도록 이 파라미터를 `all`로 설정하는 것이 좋습니다. 기본 값은 `none`입니다. 모든 데이터 정의 언어(DDL) 문(CREATE, ALTER, DROP 등)을 기록하려면 이 값을 `ddl`로 설정합니다. 모든 DDL 및 데이터 수정 언어(DML) 문(INSERT, UPDATE, DELETE 등)을 기록하려면 이 값을 `mod`로 설정합니다.

`log_min_duration_statement` 파라미터는 로그에 기록할 문의 한계를 밀리초 단위로 설정합니다. 이 파라미터 설정보다 오래 실행되는 모든 SQL 문이 로그에 기록됩니다. 이 파라미터는 기본적으로 비활성화되어 있고 -1로 설정됩니다. 이 파라미터를 활성화하면 최적화되지 않은 쿼리를 찾는데 도움이 될 수 있습니다.

쿼리 로깅을 설정하려면 다음 단계를 수행합니다.

1. `log_statement` 파라미터를 `all`로 설정합니다. 다음 예제에서는 `postgres.log` 파일에 기록되는 정보를 보여줍니다.

```
2013-11-05 16:48:56 UTC::@[2952]:LOG: received SIGHUP, reloading configuration files
2013-11-05 16:48:56 UTC::@[2952]:LOG: parameter "log_statement" changed to "all"
```

쿼리를 실행할 때 `postgres.log` 파일에 추가 정보가 기록됩니다. 다음 예제에서는 쿼리 후 파일에 기록되는 정보의 유형을 보여줍니다.

```
2013-11-05 16:41:07 UTC::@[2955]:LOG: checkpoint starting: time
2013-11-05 16:41:07 UTC::@[2955]:LOG: checkpoint complete: wrote 1 buffers (0.3%); 
0 transaction log file(s) added, 0 removed, 1 recycled; write=0.000 s, sync=0.003 s,
total=0.012 s; sync files=1, longest=0.003 s, average=0.003 s
2013-11-05 16:45:14 UTC:[local]:master@postgres:[8839]:LOG: statement: SELECT d.datname
as "Name",
pg_catalog.pg_get_userbyid(d.datdba) as "Owner",
pg_catalog.pg_encoding_to_char(d.encoding) as "Encoding",
d.datcollate as "Collate",
d.datctype as "Ctype",
pg_catalog.array_to_string(d.datacl, E'\n') AS "Access privileges"
FROM pg_catalog.pg_database d
ORDER BY 1;
2013-11-05 16:45:
```

2. `log_min_duration_statement` 파라미터를 설정합니다. 다음 예제에서는 이 파라미터가 1로 설정되어 있을 때 `postgres.log` 파일에 기록되는 정보를 보여줍니다.

```
2013-11-05 16:48:56 UTC::@[2952]:LOG: received SIGHUP, reloading configuration files
2013-11-05 16:48:56 UTC::@[2952]:LOG: parameter "log_min_duration_statement" changed to
"1"
```

지속 시간 파라미터 설정을 초과하는 쿼리를 실행할 때 `postgres.log` 파일에 추가 정보가 기록됩니다. 다음 예제에서는 쿼리 후 파일에 기록되는 정보의 유형을 보여줍니다.

```
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: statement: SELECT
c2.relname, i.indisprimary, i.indisunique, i.indisclustered, i.indisvalid,
pg_catalog.pg_get_indexdef(i.indexrelid, 0, true),
pg_catalog.pg_get_constraintdef(con.oid, true), contype, condeferrable, condeferred,
c2.reltablespace
FROM pg_catalog.pg_class c, pg_catalog.pg_class c2, pg_catalog.pg_index i
LEFT JOIN pg_catalog.pg_constraint con ON (conrelid = i.indrelid AND conindid =
i.indexrelid AND contype IN ('p','u','x'))
WHERE c.oid = '1255' AND c.oid = i.indrelid AND i.indexrelid = c2.oid
ORDER BY i.indisprimary DESC, i.indisunique DESC, c2.relname;
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: duration: 3.367 ms
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: statement: SELECT
c.oid::pg_catalog.regclass FROM pg_catalog.pg_class c, pg_catalog.pg_inherits i WHERE
c.oid=i.inhparent AND i.inhrelid = '1255' ORDER BY inhseqno;
```

```
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: duration: 1.002 ms
2013-11-05 16:51:10 UTC:[local]:master@postgres:[9193]:LOG: statement:
    SELECT c.oid::pg_catalog.regclass FROM pg_catalog.pg_class c,
        pg_catalog.pg_inherits i WHERE c.oid=i.inhrelid AND i.inhparent = '1255' ORDER BY
        c.oid::pg_catalog.regclass::pg_catalog.text;
2013-11-05 16:51:18 UTC:[local]:master@postgres:[9193]:LOG: statement: select proname
    from pg_proc;
2013-11-05 16:51:18 UTC:[local]:master@postgres:[9193]:LOG: duration: 3.469 ms
```

AWS CloudTrail을 사용하여 Amazon RDS API 호출 로깅

Amazon RDS는 Amazon RDS의 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 Amazon RDS 콘솔의 호출 및 Amazon RDS API 코드 호출 등 Amazon RDS에 대한 모든 API 호출을 이벤트로 캡처합니다. 추적을 생성하면 Amazon RDS에 대한 이벤트를 포함해 CloudTrail 이벤트를 Amazon S3 버킷에 지속적으로 제공할 수 있습니다. 추적을 구성하지 않은 경우 Event history(이벤트 기록)에서 CloudTrail 콘솔의 최신 이벤트를 볼 수도 있습니다. CloudTrail에서 수집하는 정보를 사용하여 Amazon RDS에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

CloudTrail에 대한 자세한 내용은 [AWS CloudTrail User Guide](#)를 참조하십시오.

CloudTrail의 Amazon RDS 정보

CloudTrail은 계정 생성 시 AWS 계정에서 활성화됩니다. Amazon RDS에서 활동이 수행되면 해당 활동은 이벤트 기록에서 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 [CloudTrail 이벤트 기록에서 이벤트 보기](#)를 참조하십시오.

Amazon RDS 이벤트를 비롯하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성하십시오. 추적은 CloudTrail이 Amazon S3 버킷으로 로그 파일을 전송할 수 있도록 합니다. 콘솔에서 추적을 생성하면 기본적으로 모든 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하십시오.

- [추적 생성 개요](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에 대한 Amazon SNS 알림 구성](#)
- [여러 리전에서 CloudTrail 로그 파일 받기 및 여러 계정에서 CloudTrail 로그 파일 받기](#)

모든 Amazon RDS 작업이 CloudTrail에서 로깅되고 [Amazon RDS API Reference](#)에 문서화됩니다. 예를 들어, CreateDBInstance, ModifyDBInstance 및 CreateDBParameterGroup 작업을 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 자격 증명 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 IAM 사용자 자격 증명으로 했는지 여부
- 역할 또는 연합된 사용자에 대한 임시 보안 자격 증명을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 [CloudTrail userIdentity 요소](#)를 참조하십시오.

Amazon RDS 로그 파일 항목 이해

추적은 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 제공할 수 있도록 해 주는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함됩니다. 이벤트는 어떤 소스로부터의 단일 요청을 나타내며 요청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보가 들어 있습니다. CloudTrail 로그 파일은 퍼블릭 API 호출의 주문 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": "5.6.39",  
        "dBInstancePort": 0,  
        "optionGroupMemberships": [  
            {  
                "status": "in-sync",  
                "optionGroupName": "default:mysql-5-6"  
            }  
        ],  
        "dBParameterGroups": [  
            {  
                "dBParameterGroupName": "default.mysql5.6",  
                "parameterApplyStatus": "in-sync"  
            }  
        ],  
        "monitoringInterval": 0,  
        "dBInstanceClass": "db.m1.small",  
        "allocatedStorage": 20,  
        "engine": "mysql",  
        "instanceClass": "db.m1.small",  
        "instanceType": "db.m1.small",  
        "masterUser": "myawsuser",  
        "maxAllocatedStorage": 20,  
        "maxStorage": 20,  
        "storage": 20  
    }  
}
```

```
"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"  
}
```

Amazon Aurora MySQL 작업

Amazon Aurora MySQL는 완전 관리 형태로 MySQL과 호환되는 관계형 데이터베이스 엔진으로서 고사양 상업용 데이터베이스의 속도 및 안정성이 오픈 소스 데이터베이스의 간편성 및 비용 효율성과 결합되었습니다. Aurora MySQL은 MySQL을 즉시 대체할 수 있고 새로 배포하는 MySQL이든, 혹은 기존에 배포한 MySQL이든 상관없이 설치, 조작 및 조정이 간편하고 비용 효율적이기 때문에 비즈니스와 애플리케이션에 더욱 많은 시간을 투자할 수 있습니다. Amazon RDS는 프로비저닝, 패치, 백업, 복구, 결함 감지, 수리 등 일상적인 데이터베이스 작업을 처리할 수 있는 Aurora 관리 기능을 지원합니다. 또한 Amazon RDS는 기존 Amazon RDS for MySQL 애플리케이션을 Aurora MySQL로 전환할 수 있는 푸시 버튼식 마이그레이션 도구도 제공합니다.

주제

- [Amazon Aurora MySQL의 개요 \(p. 463\)](#)
- [Amazon Aurora MySQL를 사용한 보안 \(p. 466\)](#)
- [새 SSL/TLS 인증서를 사용해 Aurora MySQL DB 클러스터에 연결할 애플리케이션을 업데이트 \(p. 468\)](#)
- [Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 \(p. 472\)](#)
- [Amazon Aurora MySQL 관리 \(p. 508\)](#)
- [Amazon Aurora MySQL용 Parallel Query 처리 \(p. 527\)](#)
- [Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 \(p. 552\)](#)
- [Amazon Aurora MySQL를 사용하는 단일 마스터 복제 \(p. 554\)](#)
- [Aurora 멀티 마스터 클러스터 작업 \(p. 584\)](#)
- [Amazon Aurora MySQL를 다른 AWS 서비스와 통합 \(p. 609\)](#)
- [Amazon Aurora MySQL 랩 모드 \(p. 654\)](#)
- [Amazon Aurora MySQL 모범 사례 \(p. 655\)](#)
- [Amazon Aurora MySQL 참조 \(p. 665\)](#)
- [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#)

Amazon Aurora MySQL의 개요

다음 단원에는 Amazon Aurora MySQL의 개요가 나와 있습니다.

주제

- [Amazon Aurora MySQL 성능 개선 사항 \(p. 463\)](#)
- [Amazon Aurora MySQL 및 지형 정보 데이터 \(p. 464\)](#)
- [Aurora MySQL 5.6과 Aurora MySQL 5.7의 비교 \(p. 465\)](#)
- [Aurora MySQL 5.7과 MySQL 5.7 비교 \(p. 465\)](#)

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.

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 인데스 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 이 기능이 MySQL의 공간 기능에 어떻게 상용하는지 설명합니다.

- Aurora MySQL 1.x는 MySQL 5.6과 동일한 [공간 데이터 유형](#) 및 [공간 관계 기능](#)을 지원합니다.
- Aurora MySQL 2.x는 MySQL 5.7과 동일한 [공간 데이터 유형](#) 및 [공간 관계 기능](#)을 지원합니다.
- Aurora MySQL 1.x 및 2.x는 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 5.6에서는 InnoDB 테이블에 대한 공간 인덱싱 기능이 제공되지 않지만 MySQL 5.7에서는 제공됩니다. Aurora MySQL 1.x 및 2.x에서는 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.

다음 데이터 정의 언어(DDL) 문은 공간 데이터 유형을 사용하는 열에 인덱스를 생성할 목적으로 지원됩니다.

테이블 생성

`CREATE TABLE` 문에서 `SPATIAL INDEX` 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

`ALTER TABLE` 문에서 `SPATIAL INDEX` 키워드를 사용하여 기존 테이블의 열에 공간 인덱스를 추가할 수 있습니다. 다음은 그 한 예입니다.

```
ALTER TABLE test ADD SPATIAL INDEX(shape);
```

인덱스 생성

`CREATE INDEX` 문에서 `SPATIAL` 키워드를 사용하여 기존 테이블의 열에 공간 인덱스를 추가할 수 있습니다. 다음은 그 한 예입니다.

```
CREATE SPATIAL INDEX shape_index ON test (shape);
```

Aurora MySQL 5.6과 Aurora MySQL 5.7의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 5.6에서 지원되지만, 이러한 기능은 Aurora MySQL 5.7에서 현재 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. Aurora MySQL 5.7에서 AWS Lambda 함수를 비동기식으로 호출할 수 있습니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

현재, Aurora MySQL 5.7은 Aurora MySQL 버전 1.16 이상에서 추가된 기능을 지원하지 않습니다. Aurora MySQL 버전 1.16에 대한 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.

Aurora MySQL 5.7 초기 릴리스에서는 성능 스키마를 사용할 수 없습니다. 성능 스키마를 지원하려면 Aurora 2.03 이상으로 업그레이드하십시오.

Aurora MySQL 5.7과 MySQL 5.7 비교

다음 기능은 MySQL 5.7.12에서 지원되지만 Aurora MySQL 5.7에서는 현재 지원되지 않습니다.

- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- `CREATE TABLESPACE` SQL 문
- X 프로토콜

이러한 기능에 대한 자세한 내용은 [MySQL 5.7 설명서](#)를 참조하십시오.

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)을 사용합니다. IAM 자격 증명을 사용하여 AWS에 연결할 때는 Amazon RDS 관리 작업에 필요한 권한을 부여할 수 있는 IAM 정책이 IAM 계정에 반드시 필요합니다. 자세한 내용은 [Amazon Aurora의 Identity and Access Management\(IAM\) \(p. 960\)](#) 단원을 참조하십시오.

IAM 계정을 사용해 Amazon RDS 콘솔에 액세스하려면 먼저 IAM 계정으로 AWS Management 콘솔에 로그인해야 합니다. 그런 다음 <https://console.aws.amazon.com/rds>에서 Amazon RDS 콘솔로 이동합니다.

- Aurora MySQL DB 클러스터는 Amazon Virtual Private Cloud(VPC)에서 생성해야 합니다. Aurora MySQL DB 클러스터에서 DB 인스턴스의 엔드포인트 및 포트에 연결할 수 있는 디바이스 또는 Amazon EC2 인스턴스를 제어하려면 VPC 보안 그룹을 사용합니다. 이 엔드포인트와 포트는 Secure Sockets Layer(SSL) 방식으로 연결할 수 있습니다. 그 밖에도 기업의 방화벽 규칙을 통해 기업에서 이용하는 디바이스의 DB 인스턴스 연결 여부를 제어하는 것도 가능합니다. VPC에 대한 자세한 정보는 [Amazon Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#) 단원을 참조하십시오.

지원되는 VPC 테넌시는 Aurora MySQL DB 클러스터에서 사용하는 인스턴스 클래스에 따라 다릅니다. default VPC 테넌시로 VPC가 공유된 하드웨어에서 실행됩니다. dedicated VPC 테넌시를 사용하여 VPC는 전용 하드웨어 인스턴스에서 실행됩니다. Aurora MySQL은 인스턴스 클래스 기반으로 다음 VPC 테넌시를 지원합니다.

- db.r3 인스턴스 클래스는 default 및 dedicated VPC 테넌시를 모두 지원합니다.
- db.r4 인스턴스 클래스는 default 및 dedicated VPC 테넌시만 지원합니다.
- db.r2 인스턴스 클래스는 default VPC 테넌시만 지원합니다.

인스턴스 클래스에 대한 자세한 정보는 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 참조하십시오. default 및 dedicated VPC 테넌시에 대한 자세한 정보는 Amazon Elastic Compute Cloud 사용 설명서의 [전용 인스턴스](#)를 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 정보는 [을 위한 IAM 데이터베이스 인증 \(p. 976\)](#) 단원을 참조하십시오.

Amazon Aurora MySQL을 사용한 마스터 사용자 권한

Amazon Aurora MySQL DB 인스턴스를 생성할 때 마스터 사용자는 다음과 같은 기본 권한을 갖습니다.

- 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
- REPLICATION CLIENT
- REPLICATION SLAVE
- SELECT
- SHOW DATABASES
- SHOW VIEW
- TRIGGER
- UPDATE

DB 클러스터를 생성할 때는 각 DB 클러스터의 관리 서비스를 위해 rdsadmin 사용자가 만들어집니다. rdsadmin 계정을 삭제하려고 하거나, 계정 이름 또는 암호를 변경하려고 하거나, 계정 권한을 변경하려고 하면 오류가 발생합니다.

Aurora MySQL DB 클러스터의 관리를 위해 기본 kill 및 kill_query 명령은 사용이 제한됩니다. 대신, Amazon RDS 명령으로 rds_kill 및 rds_kill_query를 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 SSL 사용

Amazon Aurora MySQL DB 클러스터는 Amazon RDS MySQL DB 인스턴스와 동일한 프로세스 및 퍼블릭 키를 사용하여 애플리케이션의 Secure Sockets Layer(SSL) 연결을 지원합니다.

Amazon RDS가 SSL 인증서를 생성한 후 Amazon RDS가 인스턴스를 프로비저닝할 때 DB 인스턴스에 인증서를 설치합니다. 인증 기관이 서명하는 SSL 인증서에는 SSL 인증서에는 스푸핑 공격으로부터 보호해주는 SSL 인증서를 위한 일반 이름(CN)으로 DB 인스턴스 엔드포인트가 포함되어 있습니다. 그 결과 클라이언트가 Subject Alternative Names(SAN)를 지원할 경우 SSL을 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 \(p. 947\)](#) 단원을 참조하십시오.

Aurora MySQL 5.6은 TLS(전송 계층 보안) 버전 1.0을 지원하지 않습니다. Aurora MySQL 5.7은 TLS 버전 1.0, 1.1, 1.2를 지원합니다.

SAN과 SSL을 지원하는 클라이언트로서 MariaDB Connector/J 클라이언트를 권장합니다. 자세한 정보는 [MariaDB Connector/J 다운로드](#) 페이지 단원을 참조하십시오.

기본 mysql 클라이언트를 사용하여 연결을 암호화하려면 `--ssl-ca` parameter를 사용하여 mysql 클라이언트를 실행하고 다음과 같은 퍼블릭 키 등을 참조합니다.

MySQL 5.7 이상인 경우:

```
mysql -h myinstance.c9akciq32.rds-us-east-1.amazonaws.com  
--ssl-ca=[full path]rds-combined-ca-bundle.pem --ssl-mode=VERIFY_IDENTITY
```

MySQL 5.6 이전인 경우:

```
mysql -h myinstance.c9akciq32.rds-us-east-1.amazonaws.com  
--ssl-ca=[full path]rds-combined-ca-bundle.pem --ssl-verify-server-cert
```

특정 사용자 계정에 대한 SSL 연결을 요구할 수 있습니다. 예를 들면 MySQL 버전에 따라 다음 문 중 하나를 사용하여 사용자 계정 `encrypted_user`에 대한 SSL 연결을 요구할 수 있습니다.

MySQL 5.7 이상인 경우:

```
ALTER USER 'encrypted_user'@'%' REQUIRE SSL;
```

MySQL 5.6 이전인 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 SSL을 필수 또는 선택 사항으로 지정할 수 있습니다. RDS Proxy 사용에 대한 자세한 내용은 [Amazon RDS Proxy\(미리 보기\)](#)를 사용한 연결 관리 (p. 213) 단원을 참조하십시오.

Note

MySQL과의 SSL 연결에 대한 자세한 정보는 [MySQL 문서](#)를 참조하십시오.

새 SSL/TLS 인증서를 사용해 Aurora MySQL DB 클러스터에 연결할 애플리케이션을 업데이트

2019년 9월 19일부터 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. 이전 CA 인증서는 2020년 3월 5일에 만료됩니다. 아래에서 새 인증서를 사용하기 위해 애플리케이션을 업데이트하는 방법에 관한 정보를 찾으실 수 있습니다. 애플리케이션에서 SSL/TLS를 사용해 Aurora DB 클러스터에 연결하는 경우 2020년 3월 5일 이전에 다음 단계를 수행하셔야 합니다. 이렇게 하면 애플리케이션과 Aurora DB 클러스터 간에 연결이 중단되는 것을 방지할 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 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 \(p. 949\)](#) 단원을 참조하십시오. 인증서 다운로드에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#) 단원을 참조하십시오. Aurora MySQL DB 클러스터에서 SSL/TLS를 사용하는 방법에 관한 자세한 내용은 [Aurora MySQL DB 클러스터에 SSL 사용 \(p. 467\)](#) 단원을 참조하십시오.

주제

- [애플리케이션에서 SSL을 사용해 Aurora MySQL DB 클러스터에 연결하는지 여부 확인 \(p. 469\)](#)
- [클라이언트에서 연결 시 인증서 확인이 필요한지 여부 확인 \(p. 469\)](#)
- [애플리케이션 트러스트 스토어 업데이트 \(p. 470\)](#)
- [SSL 연결 설정을 위한 Java 코드 예시 \(p. 471\)](#)

애플리케이션에서 SSL을 사용해 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 SSL/TLS를 사용하는지 여부를 확인하십시오. 성능 스키마 활성화에 대한 자세한 내용은 MySQL 문서의 [성능 스키마 빠른 시작](#)을 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 고객님 자신의 세션(admin)과 webapp1로 로그인된 애플리케이션에서 모두 SSL을 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)
```

Aurora MySQL 버전 1(MySQL 5.6과 호환됨)을 사용 중이라면 애플리케이션에서 SSL을 사용하여 연결하는지 여부를 서버 측에서 확인할 수 있습니다. 이러한 버전의 경우 애플리케이션의 연결 방법을 검토하여 SSL이 사용되는지 여부를 확인할 수 있습니다. 다음 섹션에서 클라이언트 연결 구성을 검토하는 방법에 대한 자세한 내용을 볼 수 있습니다.

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 속성](#)을 참조하십시오.

MySQL 커넥터/J 8.0을 사용 중인 경우 다음 예시와 같이 연결 속성에서 `sslMode`가 `VERIFY_CA` 또는 `VERIFY_IDENTITY`로 설정되어 있다면 SSL 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);
```

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](#) 단원을 참조하십시오.

MySQL 5.7 또는 MySQL 8.0 클라이언트를 사용 중인 경우 다음 예시와 같이 `--ssl-mode` 옵션에 대해 `VERIFY_CA` 또는 `VERIFY_IDENTITY`를 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-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

SSL/TLS 연결을 위해 JDBC를 사용하는 애플리케이션에 대해 트러스트 스토어를 업데이트할 수 있습니다.

JDBC 애플리케이션에 대해 트러스트 스토어를 업데이트하려면

- 모든 AWS 리전에서 작동하는 2019 루트 인증서를 다운로드하고 이 파일을 트러스트 스토어 디렉터리에 저장하십시오.

루트 인증서 다운로드에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#) 단원을 참조하십시오.

2. 다음 명령을 사용하여 인증서를 .der 형식으로 변환합니다.

```
openssl x509 -outform der -in rds-ca-2019-root.pem -out rds-ca-2019-root.der
```

파일 이름을 다운로드한 파일 이름으로 바꿉니다.

3. 다음 명령을 사용하여 인증서를 키 스토어로 가져옵니다.

```
keytool -import -alias rds-root -keystore clientkeystore -file rds-ca-2019-root.der
```

4. 키 스토어가 성공적으로 업데이트되었는지 확인하십시오.

```
keytool -list -v -keystore clientkeystore.jks
```

메시지가 표시되면 메타스토어 암호를 입력하십시오.

출력에 다음 사항이 포함되어 있어야 합니다.

```
rds-root, date, trustedCertEntry,  
Certificate fingerprint (SHA1):  
D4:OD:DB:29:E3:75:0D:FF:A6:71:C3:14:0B:BF:5F:47:8D:1C:80:96  
# This fingerprint should match the output from the below command  
openssl x509 -fingerprint -in rds-ca-2019-root.pem -noout
```

애플리케이션에서 mysql JDBC를 사용 중인 경우 애플리케이션에서 다음 속성을 설정하십시오.

```
System.setProperty("javax.net.ssl.trustStore", "certs");  
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

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
```

SSL 연결 설정을 위한 Java 코드 예시

다음 코드 예시에서는 JDBC를 사용하여 서버 인증서를 확인하는 SSL 연결을 설정하는 방법을 보여줍니다.

```
public class MySQLSSSTest {  
  
    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.cn1e62e2e7kwh.us-east-1.rds.amazonaws.com:3306", properties);  
        Statement stmt=connection.createStatement();  
  
        ResultSet rs=stmt.executeQuery("SELECT 1 from dual");  
  
        return;  
    }  
}
```

Important

데이터베이스 연결에서 SSL/TLS를 사용함을 확인하고 애플리케이션 트러스트 스토어를 업데이트한 후에는 데이터베이스에서 rds-ca-2019 인증서를 사용하도록 업데이트할 수 있습니다. 지침은 [DB 인스턴스를 수정하여 CA 인증서 업데이트 \(p. 950\)](#)의 3단계를 참조하십시오.

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"을 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	마이그레이션 유형	솔루션
Amazon RDS MySQL DB 인스턴스	물리적	먼저 MySQL DB 인스턴스의 Aurora MySQL 읽기 전용 복제본을 생성하여 RDS 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 둑립형 Aurora MySQL DB 클러스터로 사용하도록 지정할 수 있습니다. 세부 정보는 Aurora 읽기 전용 복제본을 사용하여 MySQL DB 인스턴스의 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션 (p. 499) 단원을 참조하십시오.
RDS MySQL DB 스냅샷	물리적	Amazon RDS MySQL DB 스냅샷의 데이터를 Amazon Aurora MySQL DB 클러스터로 직접 마이그레이션할 수 있습니다. 세부 정보는 DB 스냅샷을 사용하여 MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 (p. 492) 단원을 참조하십시오.
Amazon RDS 외부의 MySQL 데이터베이스	논리적	<code>mysqldump</code> 유틸리티로 데이터 덤프를 생성하고 해당 데이터를 기준 Amazon Aurora MySQL DB 클러스터로 가져올 수 있습니다. 세부 정보는 mysqldump를 사용하여 MySQL에서 Amazon Aurora로 마이그레이션 (p. 492) 단원을 참조하십시오.

マイグレーション 原因	マイグレーション ユーティリティ	ソルーション
Amazon RDS 外部の MySQL データベース	物理的	データベースから Amazon Simple Storage Service(Amazon S3) バケットにバックアップファイルを複数作成する後、該当ファイルを Amazon Aurora MySQL DB クラスターで復元できます。`mysqldump` を使用してデータをマイグレーションするよりも、この方法が手短かです。詳細は Amazon S3 バケットを使用して MySQL からデータをマイグレーション (p. 475) を参照してください。
Amazon RDS 外部の MySQL データベース	論理的	データベースのデータをテキストファイルに保存して Amazon S3 バケットに保管し、その後 `LOAD DATA FROM S3` MySQL 命令を使用して既存の Aurora MySQL DB クラスターに該当データをロードできます。詳細は Amazon S3 バケットのテキストファイルから Amazon Aurora MySQL DB クラスターへのデータロード (p. 620) を参照してください。
MySQL と互換性がないデータベース	論理的	AWS Database Migration Service(AWS DMS)を使用して MySQL の互換性がないデータベースのデータをマイグレーションできます。AWS DMS に関する詳細は AWS Database Migration Service について (p. 619) を参照してください。

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 memcached를 사용하는 경우 마이그레이션 전에 memcached를 제거하십시오.

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 유ти리티로 데이터 덤프를 생성하고 해당 데이터를 기존 Amazon Aurora MySQL DB 클러스터로 가져올 수 있습니다. 자세한 내용은 [mysqldump를 사용하여 MySQL에서 Amazon Aurora로 마이그레이션 \(p. 492\)](#) 단원을 참조하십시오.
 - 데이터베이스에서 Amazon S3 버킷으로 전체 및 종분 백업 파일을 복사한 다음 해당 파일에서 Amazon Aurora MySQL DB 클러스터를 복원할 수 있습니다. mysqldump를 사용하여 데이터를 마이그레이션하는 것보다 이 방법이 훨씬 더 빠를 것입니다. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션

소스 MySQL 버전 5.5, 5.6 또는 5.7 데이터베이스에서 Amazon S3 버킷으로 전체 및 충분 백업 파일을 복사한 다음, 이 파일에서 Amazon Aurora MySQL DB 클러스터를 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.

Note

Amazon S3 버킷과 Amazon Aurora MySQL DB 클러스터는 같은 AWS 리전에 있어야 합니다.

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 추가해야 합니다

- 사용자 계정
- 함수
- 저장 프로시저
- 시간대 정보. 시간대 정보는 Amazon Aurora MySQL DB 클러스터의 로컬 운영 체제에서 로드됩니다. 자세한 내용은 [Amazon AuroraDB 클러스터의 로컬 시간대 \(p. 7\)](#) 단원을 참조하십시오.

암호화된 원본 데이터베이스에서 복원할 수는 없지만 마이그레이션되는 데이터를 암호화할 수 있습니다. 또한 데이터를 마이그레이션 프로세스 동안 암호화하지 않은 상태로 유지할 수도 있습니다.

기본 MySQL 데이터 딕터리 외부에서 정의된 테이블이 있는 원본 데이터베이스에서 마이그레이션할 수 없습니다.

또 마이그레이션 프로세스 동안 이진수 로그 복제를 사용, 가동 중지를 최소화할지 여부를 결정합니다. 이진수 로그 복제를 사용하면, 외부 MySQL 데이터베이스는 데이터가 Aurora MySQL DB 클러스터로 마이그레이션 되는 동안 트랜잭션에 개방된 상태를 유지합니다. Aurora MySQL DB 클러스터를 생성한 후, 이진수 로그 복제를 사용하여 Aurora MySQL DB 클러스터와 백업 이후 발생한 트랜잭션을 동기화합니다. Aurora MySQL DB 클러스터가 MySQL 데이터베이스를 따라 잡았을 때, 새 트랜잭션에 대한 Aurora MySQL DB 클러스터로 완전히 전환해 마이그레이션을 완료합니다.

주제

- [시작하기 전에 \(p. 475\)](#)
- [Amazon Aurora MySQL DB 클러스터로 복원할 파일 백업 \(p. 477\)](#)
- [Amazon S3 버킷에서 Amazon Aurora MySQL DB 클러스터를 복원하려면 \(p. 479\)](#)
- [복제를 사용하여 Amazon Aurora MySQL DB 클러스터와 MySQL 데이터베이스를 동기화 \(p. 487\)](#)

시작하기 전에

데이터를 Amazon S3 버킷으로 복사하고 해당 파일로 DB 클러스터를 복원하려면 먼저 다음과 같이 해야 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 [Download Percona XtraBackup](#)에서 설치할 수 있습니다.

Note

MySQL 5.7 마이그레이션을 위해서는 Percona XtraBackup 2.4를 사용해야 합니다. 그 이전에 출시된 MySQL 버전인 경우 Percona XtraBackup 2.3 또는 2.4를 사용하십시오.

필요한 권한

MySQL 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션하려면 몇 가지 권한이 필요합니다.

- Amazon RDS에서 Amazon S3 버킷으로 새 클러스터를 생성하도록 요청하는 사용자는 AWS 계정의 버킷을 나열할 권한이 있어야 합니다. AWS Identity and Access Management(IAM) 정책을 사용하여 사용자에게 이 권한을 부여합니다.
- Amazon RDS는 Amazon Aurora MySQL DB 클러스터를 생성하기 위해 사용된 파일을 저장하는 Amazon S3 버킷에 대신 액세스할 수 있는 권한을 요구합니다. IAM 서비스 역할을 사용하여 Amazon RDS에 필요한 권한을 부여합니다.
- 요청한 사용자에게는 AWS 계정의 IAM 역할을 나열할 권한도 있어야 합니다.
- 요청한 사용자가 IAM 서비스 역할을 만들거나 Amazon RDS에 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 KMS 키에 대한 RDS 액세스 권한을 부여합니다. 자침은 [AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 614\)](#)을 참조하십시오.

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.

```
{  
    "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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowS3AccessRole",  
            "Effect": "Allow",  
            "Action": "iam:PassRole",  
            "Resource": "  
        }  
    ]  
}
```

```
        "Resource": "arn:aws:iam::123456789012:role/S3Access"
    ]
}
```

IAM 사용자 권한에 대한 자세한 내용은 [정책을 이용한 액세스 관리 \(p. 963\)](#) 단원을 참조하십시오.

IAM 서비스 역할 생성

새 역할 생성 옵션을 선택하여 AWS Management 콘솔에서 역할을 생성할 수 있습니다(이 주제 후반부에서 설명). 이 옵션을 선택하고 새 역할의 이름을 지정하면 Amazon RDS는 Amazon RDS가 그 이름으로 Amazon S3 버킷에 액세스하는 데 필요한 IAM 서비스 역할을 생성합니다.

또는 다음 절차에 따라 수동으로 역할을 만들 수도 있습니다.

Amazon RDS가 Amazon S3에 액세스할 수 있도록 IAM 역할을 만들려면

1. [Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 610\)](#)의 단계를 수행합니다.
2. [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)의 단계를 수행합니다.
3. [IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 \(p. 616\)](#)의 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 충분 백업 디렉터리 및 파일을 Amazon S3 버킷으로 업로드할 수 있습니다.

Percona XtraBackup을 사용하여 전체 백업 만들기

Amazon S3에서 복원하여 Amazon Aurora MySQL DB 클러스터를 생성할 수 있는 MySQL 데이터베이스 파일의 전체 백업을 만들려면 Percona XtraBackup 유ти리티(xtrabackup)를 사용하여 데이터베이스를 백업하십시오.

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
```

Percona XtraBackup 유ти리티를 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 충분 백업 사용

Amazon Aurora MySQL은 Percona XtraBackup을 사용하여 만든 전체 및 충분 백업을 모두 지원합니다. 이 미 Percona XtraBackup을 사용하여 MySQL 데이터베이스 파일의 전체 및 충분 백업을 수행 중인 경우 전체 백업을 만들고 백업 파일을 Amazon S3로 업로드할 필요가 없습니다. 대신, 전체 및 충분 백업에 대한 기준 백업 디렉터리 및 파일을 Amazon S3 버킷으로 복사하여 시간을 크게 절약할 수 있습니다. Percona XtraBackup을 사용한 충분 백업 만들기에 대한 자세한 내용은 [Incremental Backup](#)을 참조하십시오.

기존의 전체 및 충분 백업 파일을 Amazon S3 버킷으로 복사할 때 기본 디렉터리의 콘텐츠를 반복적으로 복사해야 합니다. 이러한 콘텐츠에는 전체 백업이 포함되며 모든 충분 백업 디렉터리 및 파일도 포함됩니다. 이 복사본은 Amazon S3 버킷의 디렉터리 구조를 보존해야 합니다. Aurora는 모든 파일과 디렉터리를 반복합니다. Aurora는 각 충분 백업에 포함된 `xtrabackup-checkpoints` 파일을 사용하여 기본 디렉터리를 식별하고 로그 시퀀스 번호(LSN) 범위를 기준으로 충분 백업의 순서를 정렬합니다.

파일을 만들고 Amazon S3 버킷에 업로드하는 방법에 대한 자세한 내용은 Amazon S3 시작 안내서의 [Amazon Simple Storage Service 시작하기](#)를 참조하십시오.

백업 고려 사항

파일을 Amazon S3 버킷으로 업로드할 때, 서버 측 암호화를 사용해 데이터를 암호화할 수 있습니다. 이어서 이 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 참조하십시오.

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 여려 개로 나누어야 합니다.

Amazon RDS는 Amazon S3 버킷에 업로드되는 소스 파일을 100만 개로 제한합니다. 일부 경우, 모든 전체 및 충분 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 충분 백업 파일을 저장하십시오.

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 .xbstream을 지정합니다.

Aurora는 알파벳 순서뿐 아니라 자연수 순서로도 백업 파일을 사용합니다. xtrabackup 명령을 실행할 때는 항상 split 옵션을 사용하여 적절한 순서로 백업 파일을 작성하고 이름을 붙여야 합니다.

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 대한 자세한 내용은 Percona 웹 사이트의 [Percona XtraBackup - Documentation](#) 및 [The xtrabackup Binary](#)를 참조하십시오.

Aurora에서는 Percona XtraBackup을 사용한 충분 백업을 지원합니다. Percona XtraBackup을 사용한 충분 백업 만들기에 대한 자세한 내용은 [Incremental Backup](#)을 참조하십시오.

Amazon S3 버킷에서 Amazon Aurora MySQL DB 클러스터를 복원하려면

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 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. Amazon RDS 콘솔의 오른쪽 위 모서리에서 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. 엔진 선택 페이지에서 Amazon Aurora를 선택하고 MySQL 호환성 에디션을 선택한 후 다음을 선택합니다.

[Specify source backup details] 페이지가 나타납니다.

Specify source backup details

Source database specifications

Source engine

mysql

Source engine version

5.6

S3 bucket

Refresh

S3 bucket

- Select one -

S3 folder path prefix (optional) [Info](#)

IAM role

Refresh

Create a new role

Yes

- [Specify source backup details]에서 다음을 지정합니다.

옵션	수행할 작업
[Source engine]	Aurora MySQL은 현재 mysql 데이터베이스 엔진의 백업 파일에서만 복원하도록 지원합니다.
[Source engine version]	원본 데이터베이스의 MySQL 버전을 선택합니다.
S3 버킷	백업 파일이 포함된 Amazon S3 버킷을 선택합니다.
[S3 folder path prefix (optional)]	Amazon S3 버킷에 저장된 파일의 파일 경로 접두사를 지정합니다. [S3 Bucket Prefix]는 선택 사항입니다. 접두사를 지정하지 않는 경우, Aurora MySQL은 Amazon S3 버킷의 루트 폴더에 있는 모든 파일과 폴더를 사용하여 DB 클러스터를 생성합니다. 접두사를 지정하는 경우, Aurora MySQL은 Amazon S3 버킷에서 지정된 접두사로 전체 경로가 시작되는 파일과 폴더를 사용하여 DB 클러스터를 생성합니다. Aurora는 백업 파일을 찾기 위해 Amazon S3 버킷의 다른 하위 폴더를 탐색하지 않습니다. [S3 버킷 접두사]로 확인된 폴

옵션	수행할 작업
	<p>더의 파일만 사용합니다. Amazon S3 버킷의 하위 폴더에 백업 파일을 저장하는 경우, 파일이 저장되어 있는 폴더의 전체 경로를 나타내는 접두사를 지정해야 합니다.</p> <p>예를 들어, <code>backups</code>라는 이름의 Amazon S3 버킷 하위 폴더에 백업 파일을 저장한다고 가정해 보십시오. 또 <code>gzip_backup1</code>, <code>gzip_backup2</code> 식으로 각 디렉터리에 여러 백업 파일 세트가 각각 존재한다고 가정해 보십시오. 이 경우 <code>gzip_backup1</code> 폴더의 파일로 복원하려면 <code>backups/gzip_backup1</code>을 접두사로 지정합니다.</p>
새 역할 생성	<p>예를 선택하여 새 IAM 역할을 생성하거나 아니오를 선택하여 기존 IAM 역할을 생성하여 사용자 대신 Aurora에 Amazon S3에 대한 액세스 권한을 부여하십시오. 자세한 내용은 필요한 권한 (p. 476) 단원을 참조하십시오.</p>
[IAM role name]	<p>이 옵션은 [Create a new role]을 [Yes]로 설정한 경우에만 사용할 수 있습니다.</p> <p>생성할 새 IAM 역할의 이름을 입력하십시오. 새 역할은 Amazon Aurora이 사용자 대신 Amazon S3에 액세스하도록 권한을 부여하는 데 사용됩니다. 자세한 내용은 필요한 권한 (p. 476) 단원을 참조하십시오.</p>
IAM 역할	<p>이 옵션은 Create a new role(새 역할 생성)을 아니오로 설정한 경우에만 사용할 수 있습니다.</p> <p>사용자 대신 Aurora에 Amazon S3에 대한 액세스 권한을 부여하기 위해 생성한 IAM 역할을 선택합니다. IAM 역할을 아직 생성하지 않은 경우 새 역할 생성을 예로 설정하여 하나를 생성할 수 있습니다. 자세한 내용은 필요한 권한 (p. 476) 단원을 참조하십시오.</p>
Allow access to KMS key	<p>이 옵션은 [Create a new role]을 [Yes]로 설정한 경우에만 사용할 수 있습니다.</p> <p>백업 파일을 암호화하지 않았다면 [No]를 선택합니다.</p> <p>Amazon S3으로 백업 파일을 업로드할 때 AES-256(SSE-S3)로 암호화를 했다면 아니오를 선택합니다. 이 경우, 데이터가 자동으로 복호화됩니다.</p> <p>Amazon S3으로 백업 파일을 업로드할 때 AWS-KMS (SSE-KMS) 서버 측 암호화로 암호화를 했다면 예를 선택합니다. 다음은 [Master key]에 대한 올바른 마스터 키를 선택합니다. AWS Management 콘솔은 Amazon RDS를 활성화하여 데이터를 복호화하는 IAM 정책을 생성합니다.</p> <p>자세한 내용은 Amazon S3 개발자 안내서의 서버 측 암호화를 사용하여 데이터 보호 단원을 참조하십시오.</p>

6. [Next]를 선택합니다.
7. DB 세부 정보 지정 페이지에서 DB 클러스터 정보를 지정합니다.

일반적인 [Specify DB details] 페이지는 다음과 같습니다.

Specify DB details

Configuration

Estimate your monthly costs for the DB Instance using the [AWS Simple Monthly Calculator](#)

DB engine

Aurora - compatible with MySQL 5.6

Capacity type [Info](#)

Provisioned

You provision and manage the server instance sizes.

Provisioned with Aurora parallel query enabled [Info](#)

You provision and manage the server instance sizes, and Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6)

Serverless [Info](#)

You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load.

DB engine version [Info](#)

Aurora (MySQL)-5.6.10a

DB instance class [Info](#)

- Select one -

Multi-AZ deployment [Info](#)

Create Replica in Different Zone

No

다음 표는 DB 인스턴스 설정을 나타냅니다.

옵션	수행할 작업
용량 유형	DB 인스턴스에 대한 용량을 수동으로 관리하려면 프로비저닝됨을 선택합니다. 워크로드가 변경된 경우 인스턴스에 대한 DB 인스턴스 클래스를 변경해야 할 수 있습니다. DB 인스턴스의 용량을 수동으로 관리하려면 Provisioned with Aurora parallel query enabled(Aurora 병렬 쿼리가 활성화된 상태에서 프로비저닝됨)를 선택합니다. 이 옵션을 통해 Aurora는 처리를 Aurora 스토리지 계층으로 하향 푸시하여 분석 쿼리의 성능을 향상시킵니다(현재 Aurora MySQL 5.6에 사용 가능). 자세한 내용은 Amazon Aurora MySQL용 Parallel Query 처리 (p. 527) 단원을 참조하십시오.
DB 엔진 버전	프로비저닝된 용량 유형에만 적용됩니다. DB 엔진의 버전 번호를 선택합니다.

옵션	수행할 작업
DB 인스턴스 클래스	DB 클러스터의 각 인스턴스 처리 및 메모리 요건을 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 (p. 30) 단원을 참조하십시오.
다중 AZ 배포	장애 조치 지원을 위해 다른 가용 영역에 Aurora 복제본을 생성할지 여부를 결정합니다. 다른 영역에 복제본 생성을 선택한 경우 Amazon RDS가 DB 클러스터를 위한 기본 인스턴스보다는 다른 가용 영역의 DB 클러스터에 Aurora Replica를 생성합니다. 다중 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 (p. 3) 단원을 참조하십시오.
DB 인스턴스 식별자	DB 클러스터의 기본 인스턴스 이름을 입력합니다. 이 식별자는 DB 클러스터의 기본 인스턴스에 대한 엔드포인트 주소로 사용됩니다. DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다. <ul style="list-style-type: none"> • 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다. • 첫 번째 문자는 글자이어야 합니다. • 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. • AWS 리전별로 AWS 계정 하나당 모든 DB 인스턴스는 고유해야 합니다.
Master username	영숫자 문자를 사용해 DB 클러스터에 로그온하기 위해 마스터 사용자 이름으로 사용할 이름을 입력합니다.
Master password	마스터 사용자 암호로 인쇄 가능한 ASCII 문자(/, " 및 @ 제외) 8~41자를 포함하는 암호를 입력합니다.

8. [Next]를 선택합니다.
9. 고급 설정 구성 페이지에서 Aurora MySQL DB 클러스터 설정을 추가로 사용자 지정할 수 있습니다. 다음 표는 DB 클러스터의 고급 설정을 나타냅니다.

옵션	수행할 작업
Virtual Private Cloud(VPC)	DB 클러스터를 호스팅할 VPC를 선택합니다. Amazon RDS에서 VPC를 생성하도록 하려면 [Create a New VPC]를 선택합니다. 자세한 내용은 이번 주제에서 전반부 DB 클러스터 사전 요구사항 (p. 95) 단원을 참조하십시오.
Subnet Group	DB 클러스터에 사용할 DB 서브넷 그룹을 선택합니다. 자세한 내용은 이번 주제에서 전반부 DB 클러스터 사전 요구사항 (p. 95) 단원을 참조하십시오.
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 습기는 방법에 대한 자세한 내용은 VPC에 있는 DB 인스턴스를 인터넷에서 습기기 (p. 1017) 단원을 참조하십시오.

옵션	수행할 작업
[Availability zone]	특정 가용 영역의 지정 여부를 결정합니다. 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 (p. 3) 단원을 참조하십시오.
VPC 보안 그룹	Amazon RDS가 자동으로 VPC 보안 그룹을 생성하도록 하려면 Create new VPC security group(새 VPC 보안 그룹 생성)을 선택합니다. 또는 [Select existing VPC security groups]를 선택하고 하나 이상의 VPC 보안 그룹을 지정하여 DB 클러스터에 대한 네트워크 액세스에 보안을 적용합니다. 자세한 내용은 이번 주제에서 전반부 DB 클러스터 사전 요구사항 (p. 95) 단원을 참조하십시오.
DB Cluster Identifier	선택한 AWS 리전에 속한 계정 고유의 DB 클러스터 이름을 입력합니다. 이 식별자는 DB 클러스터에 대한 클러스터 엔드포인트 주소로 사용됩니다. 클러스터 엔드포인트에 대한 자세한 내용은 Amazon Aurora 연결 관리 (p. 9) 단원을 참조하십시오. DB 클러스터 식별자는 다음과 같은 제약 조건이 있습니다. <ul style="list-style-type: none">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.
데이터베이스 이름	기본 데이터베이스의 이름을 최대 64자의 영숫자 문자로 입력합니다. 이름을 입력하지 않으면 Amazon RDS가 DB 클러스터에서 생성하려고 하는 데이터베이스를 생성하지 않습니다. 추가 데이터베이스를 생성하려면, DB 클러스터에 연결한 다음 SQL 명령어 CREATE DATABASE를 사용하십시오. DB 클러스터 연결에 대한 자세한 내용은 Amazon Aurora DB 클러스터 연결 (p. 162) 단원을 참조하십시오.
데이터베이스 포트	애플리케이션과 유ти리티가 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.
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 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.

옵션	수행할 작업
DB 클러스터 파라미터 그룹	DB 클러스터 파라미터 그룹을 선택합니다. Aurora는 기본 값 DB 클러스터 파라미터 그룹을 제공하며, 자체 DB 클러스터 파라미터 그룹을 생성할 수도 있습니다. DB 클러스터 파라미터 그룹에 대한 자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.
옵션 그룹	Aurora은 기본값 옵션 그룹을 포함합니다.
암호화	이 DB 클러스터에 대해 비활성화되어 있는 암호화를 활성화하려면 Enable Encryption 을 선택합니다. 자세한 내용은 Amazon Aurora 리소스 암호화 (p. 945) 단원을 참조하십시오.
마스터 키	[Encryption]을 [Enable Encryption]으로 설정한 경우에만 사용할 수 있습니다. 현재 DB 클러스터를 암호화하는 데 사용할 마스터 키를 선택합니다. 자세한 내용은 Amazon Aurora 리소스 암호화 (p. 945) 단원을 참조하십시오.
Priority	인스턴스의 장애 조치 우선 순위를 선택합니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 기본 인스턴스 장애로부터 복원할 때 이 우선 순위에 따라 승격할 Aurora Replicas 순서가 결정됩니다. 자세한 내용은 Aurora DB 클러스터의 내결합성 (p. 268) 단원을 참조하십시오.
백업 보존 기간	Aurora가 데이터베이스 백업 사본을 보존하는 기간을 1~35 일로 선택합니다. 백업 사본은 데이터베이스를 마지막 특정 시점으로 복구(PITR)하는 데 사용됩니다.
역추적	역추적을 활성화하려면 역추적 활성화를 선택하고 역추적을 비활성화하려면 역추적 비활성화를 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 (p. 509) 단원을 참조하십시오.
확장 모니터링	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 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-role라는 이름의 역할을 생성하도록 기본값을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 단원을 참조하십시오.
Granularity	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
로그 내보내기	Amazon CloudWatch Logs에 게시를 시작할 로그를 선택합니다. CloudWatch Logs로의 게시에 대한 자세한 내용은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 (p. 639) 단원을 참조하십시오.

옵션	수행할 작업
Auto minor version upgrade	이 설정은 Aurora MySQL DB 클러스터에 적용되지 않습니다. Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL 데이터베이스 엔진 업데이트 (p. 686) 단원을 참조하십시오.
유지 관리 기간	[Select window]를 선택하고 시스템 유지 관리를 실행할 수 있는 주 단위 기간을 지정합니다. 또는 Amazon RDS가 임의로 기간을 지정하도록 하려면 기본 설정 없음을 선택합니다.
삭제 방지 활성화	DB 클러스터가 삭제되지 않도록 방지하려면, 삭제 방지 활성화를 선택합니다. 콘솔을 사용하여 프로덕션 DB 클러스터를 생성할 경우 기본값으로 삭제 방지가 활성화됩니다.

- 데이터베이스 생성을 선택하여 Aurora DB 인스턴스를 시작한 다음 닫기를 선택하여 마법사를 닫으십시오.

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 \(p. 338\)](#) 단원을 참조하십시오.

database-1

Modify

Related

Filter databases

DB identifier	Role	Engine	Region
database-1	Regional	Aurora MySQL	us-east-1
database-1-instance-1	Writer	Aurora MySQL	us-east-1

Connectivity & security Monitoring Logs & events Configuration Maintenance & backup

Endpoints (2)

Edit Delete Create custom

Filter endpoint

Endpoint name	Status	Type
database-1.cluster-ro-... .us-east-2.rds.amazonaws.com	Available	Read
database-1.cluster-i... .us-east-2.rds.amazonaws.com	Available	Write

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 클러스터와 MySQL 데이터베이스를 동기화

マイグレーション 동안 가동 정지가 거의 없도록 만들기 위해, MySQL 데이터베이스에 커밋된 트랜잭션을 Aurora MySQL DB 클러스터로 복제할 수 있습니다. 복제를 사용하여 DB 클러스터가 마이그레이션 동안 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 (p. 488)
- 복제를 사용하여 Amazon Aurora MySQL DB 클러스터와 외부 MySQL 데이터베이스를 동기화 (p. 489)

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 건너뛰고 [복제를 사용하여 Amazon Aurora MySQL DB 클러스터와 외부 MySQL 데이터베이스를 동기화 \(p. 489\)](#)의 지시로 이동합니다.

다음은 암호화된 복제를 사용하기 위한 사전 조건입니다.

- 외부 MySQL 마스터 데이터베이스에서 SSL(Secure Socket Layer)를 활성화 시켜야 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,

- 암호화 복제를 준비해야 합니다.

- 외부 MySQL 마스터 데이터베이스에서 SSL을 활성화 시키지 않았고 클라이언트 키와 인증서가 준비되지 않았다면, MySQL 데이터베이스 서버의 SSL을 활성화하고, 필요한 클라이언트 키와 인증서를 생성합니다.
- 외부 마스터에 SSL이 활성화되어 있다면, 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 마스터 데이터베이스의 SSL을 구성할 때 사용하는 인증 기관(CA) 키가 있어야 합니다.

자세한 정보는 MySQL 문서의 [Creating SSL Certificates and Keys Using openssl](#)을 참조하십시오.

인증 기관(CA) 인증서, 클라이언트 키, 클라이언트 인증서가 필요합니다.

- SSL을 사용하여 마스터 사용자로 Aurora MySQL DB 클러스터에 연결하십시오.

SSL을 이용한 Aurora MySQL DB 클러스터 연결에 대한 자세한 정보는 [Aurora MySQL DB 클러스터에 SSL 사용 \(p. 467\)](#)을 참조하십시오.

- `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 길니다.

```
call mysql.rds_import_binlog_ssl_material(
  '{"ssl_ca": "-----BEGIN CERTIFICATE-----\nAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V\nhz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEZoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr\nlsLnBItntckiJ7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0Fzz\nqaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221Cb5IMucxXPkX4rWi+z7wB3Rb\nBQoQzd8v7yeb7Oz1PnW0yN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE\n-----END CERTIFICATE-----\n", "ssl_cert": "-----BEGIN CERTIFICATE-----\nAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V\nhz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEZoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr\nlsLnBItntckiJ7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0Fzz\nqaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221Cb5IMucxXPkX4rWi+z7wB3Rb\nBQoQzd8v7yeb7Oz1PnW0yN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE"}')
```

```
-----END CERTIFICATE-----\n","ssl_key":"-----BEGIN RSA PRIVATE KEY-----\nAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V\nhz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/d6RJhJOI0iBXr\nlsLnBItnckiJ7FbtJMxLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/cQk+0Fzz\nqaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkYQS3xqC0+FmUZofz221CBt5IMucxXPkX4rWi+z7wB3Rb\nBQoQzd8v7yeb70z1PnWoyN0qFU0XA246RA8QFYiCNYwI3f05p6KLxEXAMPLE\n-----END RSA PRIVATE KEY-----\n"}');
```

자세한 내용은 [mysql_rds_import_binlog_ssl_material](#) 및 [Aurora MySQL DB 클러스터에 SSL 사용 \(p. 467\)](#) 단원을 참조하십시오.

Note

프로시저를 실행한 후 파일에 암호를 저장합니다. 이 파일을 나중에 삭제하려면 [mysql_rds_remove_binlog_ssl_material](#) 저장 프로시저를 실행할 수 있습니다.

복제를 사용하여 Amazon Aurora MySQL DB 클러스터와 외부 MySQL 데이터베이스를 동기화

복제를 사용하여 Amazon Aurora MySQL DB 클러스터와 MySQL 데이터베이스를 동기화 할 수 있습니다.

복제를 사용하여 Aurora MySQL DB 클러스터와 MySQL 데이터베이스를 동기화하려면,

- 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 활성화 시켜 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.

- AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
- 탐색 창에서 [Events]를 선택합니다.
- [Events] 목록에서 [Recovered from Binary log filename] 이벤트의 위치를 확인합니다.

Events (3)		
Identifier	Type	Event
testingest	Instances	DB instance created
testingest	Instances	Binlog position from crash recovery is OFF.000001 532
testingest-cluster	DB clusters	Recovered from Binary log filename 'mysqld-bin.00001'

다음 단계에서 복제를 시작할 위치를 지정합니다.

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 SSL이 포함되어 있지 않다면, 복제 연결이 자동으로 암호화되지 않은 연결로 돌아갈 수 있습니다.

4. Amazon RDS Management Console에서 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 RDS_SQL_DB_<host_name>
```

호스트 이름은 Aurora MySQL DB 클러스터 엔드포인트의 DNS 이름입니다.

5. mysql.rds_set_external_master 저장 프로시저를 실행해 이진수 로그 복제를 활성화 합니다. 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
);
```

파라미터에 대한 자세한 내용은 [mysql.rds_set_external_master](#)를 참조하십시오.

mysql_binary_log_file_name 및 mysql_binary_log_file_location의 경우, 앞에서 확인한 [Recovered from Binary log filename] 이벤트의 위치를 사용합니다.

Aurora MySQL DB 클러스터의 데이터가 암호화되어 있지 않다면, ssl_encryption 파라미터를 0으로 설정해야 합니다. 데이터가 암호화되어 있으면 ssl_encryption 파라미터를 1로 설정합니다.

다음은 암호화된 데이터가 있는 Aurora MySQL DB 클러스터에 대해 프로시저를 실행하는 예제입니다.

```
CALL mysql.rds_set_external_master(
    'Externaldb.some.com',
    3306,
    'rep1_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 마스터 데이터베이스보다 얼마나 지연되어 있는지 모니터링하십시오. 이렇게 하려면 Aurora MySQL DB 클러스터에 연결하고 다음 명령을 실행하십시오.

```
SHOW SLAVE STATUS;
```

명령 출력의 Seconds Behind Master 필드는 Aurora MySQL DB 클러스터가 MySQL 마스터보다 얼마나 지연되었는지 보여줍니다. 이 값이 0인 경우, Aurora MySQL DB 클러스터는 마스터를 따라 잡습니다. 그러면 다음 단계로 이동해 복제를 중지할 수 있습니다.

- MySQL 복제 마스터 데이터베이스를 연결하고, 복제를 중지합니다. 이렇게 하려면 다음 명령을 실행합니다.

```
CALL mysql.rds_stop_replication;
```

mysqldump를 사용하여 MySQL에서 Amazon Aurora로 마이그레이션

Amazon Aurora MySQL은 MySQL 호환 데이터베이스이므로 mysqldump 유ти리티를 사용하여 MySQL 또는 MariaDB 데이터베이스에서 기존의 Aurora MySQL DB 클러스터로 데이터를 복사할 수 있습니다.

초대형 MySQL 데이터베이스를 사용하는 방법은 [자동 종지 시간을 단축하여 Amazon RDS MySQL 또는 MariaDB DB 인스턴스로 데이터 가져오기](#)를 참조하십시오. 소량의 데이터를 갖는 MySQL 데이터베이스는 [MySQL 또는 MariaDB DB 데이터를 Amazon RDS MySQL 또는 MariaDB DB 인스턴스로 가져오기](#)를 참조하십시오.

DB 스냅샷을 사용하여 MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션

다음과 같은 방법으로 Amazon RDS MySQL DB 스냅샷의 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션(복사)할 수 있습니다.

주제

- [RDS MySQL 스냅샷을 Aurora로 마이그레이션 \(p. 492\)](#)
- [Aurora 읽기 전용 복제본을 사용하여 MySQL DB 인스턴스의 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션 \(p. 499\)](#)

Note

Amazon Aurora MySQL는 MySQL과 호환되기 때문에 MySQL 데이터베이스와 Amazon Aurora MySQL DB 클러스터 사이에 복제를 설정하면 MySQL 데이터베이스에서 데이터를 마이그레이션할 수 있습니다. 복제를 사용해 MySQL 데이터베이스의 데이터를 마이그레이션하려면 MySQL 데이터베이스에서 MySQL 버전 5.5 이상을 실행하는 것이 좋습니다. 자세한 내용은 [Amazon Aurora를 사용한 복제 \(p. 55\)](#) 단원을 참조하십시오.

RDS MySQL 스냅샷을 Aurora로 마이그레이션

Amazon RDS MySQL DB 인스턴스의 DB 스냅샷을 마이그레이션하면 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 새 Aurora MySQL DB 클러스터에 원본 Amazon RDS MySQL DB 인스턴스의 데이터가 채워집니다. DB 스냅샷은 MySQL 버전 5.6 또는 5.7을 실행하는 Amazon RDS DB 인스턴스에서 생성해야 합니다.

DB 스냅샷을 마이그레이션하는 방법은 수동과 자동이 있습니다. DB 클러스터를 생성한 후에는 옵션으로 Aurora 복제본을 생성할 수도 있습니다.

MySQL DB 인스턴스 및 Aurora DB 클러스터에서 동일한 버전의 MySQL을 실행 중인 경우 MySQL 스냅샷을 Aurora DB 클러스터로 직접 복원할 수 있습니다. 예를 들면 MySQL 버전 5.6 스냅샷을 Aurora MySQL 버전 5.6으로 복원할 수 있지만, MySQL 버전 5.6 스냅샷은 Aurora MySQL 버전 5.7로만 복원할 수 있습니다.

MySQL 버전 5.6 스냅샷을 Aurora MySQL 버전 5.7로 마이그레이션하려는 경우 다음 방법 중 하나로 마이그레이션을 수행할 수 있습니다.

- MySQL 버전 5.6 스냅샷을 Aurora MySQL 버전 5.6으로 마이그레이션하고 Aurora MySQL 버전 5.6 DB 클러스터의 스냅샷을 생성한 다음, Aurora MySQL 버전 5.6 스냅샷을 Aurora MySQL 버전 5.7로 복원하십시오.
- MySQL 버전 5.6 스냅샷을 MySQL 버전 5.7로 업그레이드하고, MySQL 버전 5.7 DB 인스턴스의 스냅샷을 생성한 다음, MySQL 버전 5.7 스냅샷을 Aurora MySQL 버전 5.7로 복원하십시오.

Note

또한 소스 MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하여 MySQL DB 인스턴스를 Aurora MySQL DB 클러스터로 마이그레이션할 수도 있습니다. 자세한 내용은 [Aurora 읽기 전용 복제본을 사용하여 MySQL DB 인스턴스의 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션 \(p. 499\)](#) 단원을 참조하십시오.

MySQL과 Aurora MySQL 간의 비호환성 관련 문제는 다음과 같습니다.

- MySQL 버전 5.7 스냅샷을 Aurora MySQL 버전 5.6으로 마이그레이션할 수 없습니다.
- MySQL 5.6.40 또는 5.7.22로 생성된 DB 스냅샷은 Aurora MySQL로 마이그레이션할 수 없습니다.

일반적으로 다음 단계를 반드시 따라야 합니다.

- Aurora MySQL DB 클러스터에 프로비저닝할 공간 크기를 결정합니다. 자세한 내용은 [필요한 공간 크기 \(p. 493\)](#) 단원을 참조하십시오.
- 콘솔을 사용하여 Amazon RDS MySQL 인스턴스가 위치한 AWS 리전에 스냅샷을 생성하십시오. DB 스냅샷 생성에 대한 자세한 내용은 [DB 스냅샷 생성](#) 단원을 참조하십시오.
- DB 스냅샷이 DB 클러스터와 동일한 AWS 리전에 속하지 않을 때는 Amazon RDS 콘솔을 사용하여 DB 스냅샷을 해당 AWS 리전으로 복사하십시오. DB 스냅샷 복사에 대한 자세한 내용은 [DB 스냅샷 복사](#) 단원을 참조하십시오.
- 콘솔을 사용하여 DB 스냅샷을 마이그레이션한 후 원본 MySQL DB 인스턴스와 동일한 데이터베이스로 Aurora MySQL DB 클러스터를 생성하십시오.

Warning

Amazon RDS는 AWS 계정마다 스냅샷 사본을 한 번에 각 AWS 리전의 스냅샷 사본 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 필요할 경우가 간혹 있습니다.

MyISAM 테이블이 아니거나 압축되지 않은 테이블은 최대 크기가 16TB로 제한됩니다. MyISAM 테이블이 있는 경우 Aurora는 Aurora MySQL과 호환되도록 테이블을 변환하기 위해 볼륨에 추가 공간이 필요합니다. 압축된 테이블이 있으면 Aurora는 이 테이블을 Aurora 클러스터 볼륨에 저장하기 전에 확장할 추가 공간이 필

요합니다. 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.

테이블 유형	제한 또는 지침
MyISAM 테이블	<p>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.</p> <p>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.</p> <pre>alter table <schema>.<table_name> engine=innodb, algorithm=copy;</pre>
압축된 테이블	<p>Aurora MySQL에서는 압축된 테이블(ROW_FORMAT=COMPRESSED로 만든 테이블)이 지원되지 않습니다.</p> <p>공간이 부족할 가능성을 줄이거나 마이그레이션 프로세스 속도를 높이려면 ROW_FORMAT을 DEFAULT, COMPACT, DYNAMIC 또는 REDUNDANT로 설정하여 압축된 테이블을 확장합니다. 자세한 내용은 https://dev.mysql.com/doc/refman/5.6/en/innodb-row-format.html을 참조하십시오.</p>

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. ' +
    '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)		

콘솔

Amazon RDS MySQL DB 인스턴스의 DB 스냅샷을 마이그레이션하면 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 새 Aurora MySQL DB 클러스터에 원본 Amazon RDS MySQL DB 인스턴스의 데이터가 채

위입니다. DB 스냅샷은 MySQL 버전 5.6 또는 5.7을 실행하는 Amazon RDS DB 인스턴스에서 생성해야 합니다. DB 스냅샷 생성에 대한 자세한 내용은 [DB 스냅샷 생성](#) 단원을 참조하십시오.

데이터를 찾고자 하는 AWS 리전에 DB 스냅샷이 없을 때는 Amazon RDS 콘솔을 사용하여 DB 스냅샷을 이 AWS 리전으로 복사하십시오. DB 스냅샷 복사에 대한 자세한 내용은 [DB 스냅샷 복사](#) 단원을 참조하십시오.

AWS Management 콘솔을 사용하여 DB 스냅샷을 마이그레이션할 경우, 콘솔에서 DB 클러스터와 기본 인스턴스 모두를 생성하는 데 필요한 작업이 따릅니다.

AWS Key Management Service(AWS KMS) 암호화 키를 사용하여 새 Aurora MySQL DB 클러스터가 유휴 상태에서 암호화되도록 선택할 수도 있습니다.

AWS Management 콘솔을 사용하여 MySQL DB 스냅샷을 마이그레이션하려면

1. AWS Management 콘솔에 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 Instances를 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 스토리지 및 용량이 있는 DB 인스턴스 클래스를 선택합니다(예: db.r3.large). Aurora 클러스터 볼륨은 데이터베이스의 데이터 양이 증가함에 따라 최대 크기인 64 tebibytes (TiB)까지 자동으로 증가합니다. 따라서 현재 스토리지 요구 사항에 맞는 DB 인스턴스 클래스를 선택해야 합니다. 자세한 내용은 [Aurora 스토리지 개요 \(p. 34\)](#) 단원을 참조하십시오.
 - DB 인스턴스 식별자: 선택한 AWS 리전의 계정에 대해 고유한 DB 클러스터의 이름을 입력합니다. 이 식별자는 DB 클러스터에 속한 인스턴스의 엔드포인트 주소로 사용됩니다. 선택한 AWS 리전 및 DB 엔진을 포함해(예: **aurora-cluster1**) 이름에 몇 가지 인텔리전스를 추가할 수 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.
- AWS 리전별로 AWS 계정 하나당 모든 DB 인스턴스는 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. 기존 VPC 사용 방법에 대한 자세한 내용은 [Amazon Aurora에 사용할 VPC의 생성 방법 \(p. 1005\)](#) 단원을 참조하십시오.

기존 VPC가 없다면 [Create a new VPC]를 선택하여 Amazon RDS에서 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 [Create a new subnet group]을 선택하여 Amazon RDS에서 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. Amazon RDS에서 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 활성을 선택합니다. 암호화 활성을 선택하면 마스터 키 값으로 AWS KMS 암호화 키를 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 MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#) 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 연결하는데 사용할 클러스터 앤드포인트가 표시됩니다. Aurora MySQL DB 클러스터 연결에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#) 단원을 참조하십시오.

AWS CLI

Amazon RDS MySQL DB 인스턴스의 DB 스냅샷을 마이그레이션하면 Aurora DB 클러스터를 생성할 수 있습니다. 그러면 DB 스냅샷의 데이터가 새 DB 클러스터로 채워집니다. DB 스냅샷은 MySQL 버전 5.6 또는 5.7을 실행하는 Amazon RDS DB 인스턴스에서 가져와야 합니다. 자세한 내용은 [DB 스냅샷 생성](#) 단원을 참조하십시오.

DB 스냅샷이 데이터를 마이그레이션하고자 하는 AWS 리전에 없는 경우에는 DB 스냅샷을 해당 AWS 리전으로 복사합니다. 자세한 내용은 [DB 스냅샷 복사](#) 단원을 참조하십시오.

`restore-db-cluster-from-snapshot` 명령을 다음 파라미터와 함께 사용하여 Amazon RDS MySQL DB 인스턴스의 DB 스냅샷으로 Aurora DB 클러스터를 생성할 수 있습니다.

- `--db-cluster-identifier`

생성할 DB 클러스터의 이름입니다.

- MySQL 5.7-호환 DB 클러스터의 경우 --engine aurora-mysql 또는 MySQL 5.6-호환 DB 클러스터의 경우 --engine aurora
- kms-key-id

DB 스냅샷 암호화 여부에 따라 DB 클러스터를 암호화할 수 있는 AWS Key Management Service(AWS KMS) 암호화 키입니다.

- 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

マイグレーション할 DB 스냅샷의 ARN(Amazon 리소스 이름)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.

Linux, OS X, Unix의 경우:

```
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 예에서는 ARN이 `mydbsnapshotARN`으로 설정된 DB 스냅샷에서 이름이 `mydbcluster`인 MySQL 5.6-호환 DB 클러스터를 생성합니다.

Linux, OS X, Unix의 경우:

```
aws rds restore-db-cluster-from-snapshot \
--db-cluster-identifier mydbcluster \
--snapshot-identifier mydbsnapshotARN \
--engine aurora
```

Windows의 경우:

```
aws rds restore-db-cluster-from-snapshot ^
```

```
--db-cluster-identifier mydbcluster ^
--snapshot-identifier mydbsnapshotARN ^
--engine aurora
```

Aurora 읽기 전용 복제본을 사용하여 MySQL DB 인스턴스의 데이터를 Amazon Aurora MySQL DB 클러스터로 마이그레이션

Amazon RDS는 MySQL DB 엔진의 바이너리 로그 복제 기능을 사용하여 원본 MySQL DB 인스턴스의 Aurora 읽기 전용 복제본이라고 하는 특수한 유형의 DB 클러스터를 만들 수 있습니다. 원본 MySQL DB 인스턴스에 적용된 업데이트는 Aurora 읽기 전용 복제본에 비동기 방식으로 복제됩니다.

이 기능을 사용하여 원본 MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성해 MySQL DB 인스턴스에서 Aurora MySQL DB 클러스터로 마이그레이션하는 것을 권장합니다. MySQL DB 인스턴스와 Aurora 읽기 전용 복제본 간 복제 지연이 0이라면 클라이언트 애플리케이션을 Aurora 읽기 전용 복제본으로 이동한 다음 Aurora 읽기 전용 복제본을 독립형 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 마이그레이션에는 상당한 시간이 드는데, 데이터 테비바이트(TiB)당 몇 시간 정도가 소요됩니다.

Aurora를 사용할 수 있는 리전 목록은 AWS General Reference의 [Amazon Aurora](#) 단원을 참조하십시오.

MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하면 Amazon RDS는 원본 MySQL DB 인스턴스의 DB 스냅샷을 생성합니다(Amazon RDS에 대해 프라이빗이며 요금이 발생하지 않음). 그런 다음 Amazon RDS는 DB 스냅샷의 데이터를 Aurora 읽기 전용 복제본으로 마이그레이션합니다. 데이터가 DB 스냅샷에서 새로운 Aurora MySQL DB 클러스터로 마이그레이션된 후 Amazon RDS는 MySQL DB 인스턴스와 Aurora MySQL DB 클러스터 간 복제를 시작합니다. 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 [DB 스냅샷을 사용하여 MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 \(p. 492\)](#) 단원을 참조하십시오.

MySQL DB 인스턴스 하나에 Aurora 읽기 전용 복제본 하나만 둘 수 있습니다.

Note

Amazon Aurora MySQL과 복제 마스터인 RDS MySQL DB 인스턴스의 MySQL 데이터베이스 엔진 버전의 기능 차이로 인해 복제 문제가 발생할 수 있습니다. 오류가 발생하면 [Amazon RDS 커뮤니티 포럼](#) 또는 AWS Support에서 지원을 받을 수 있습니다.

MySQL 읽기 전용 복제본에 대한 자세한 내용은 [MariaDB, MySQL 및 PostgreSQL DB 인스턴스의 읽기 전용 복제본 작업](#)을 참조하십시오.

Aurora 읽기 전용 복제본 생성

MySQL DB 인스턴스의 Aurora 읽기 전용 복제본은 콘솔 또는 AWS CLI를 사용해 생성할 수 있습니다.

콘솔

원본 MySQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하는 방법

1. AWS Management 콘솔에 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 요구를 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스 옵션에 대한 자세한 정보는 DB 인스턴스 클래스 (p. 30) 단원을 참조하십시오.
다중 AZ 배포	장애 조치 지원을 위해 대상 AWS 리전의 다른 가용 영역에 새 DB 클러스터의 예비 복제본을 만들려면 다른 영역에 복제본 생성을 선택합니다. 다중 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 (p. 3) 단원을 참조하십시오.
DB 인스턴스 식별자	<p>Aurora 읽기 전용 복제본 DB 클러스터의 기본 인스턴스 이름을 입력합니다. 이 식별자는 새 DB 클러스터의 기본 인스턴스에 대한 앤드포인트 주소로 사용됩니다.</p> <p>DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다.</p> <ul style="list-style-type: none"> 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자이어야 합니다. 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. AWS 리전별로 각 AWS 계정의 모든 DB 인스턴스에 대해 고유해야 합니다. <p>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.</p>
Virtual Private Cloud(VPC)	DB 클러스터를 호스팅할 VPC를 선택합니다. Amazon RDS가 자동으로 VPC를 생성하도록 하려면 새 VPC 생성을 선택합니다. 자세한 내용은 DB 클러스터 사전 요구사항 (p. 95) 단원을 참조하십시오.
Subnet Group	DB 클러스터에 사용할 DB 서브넷 그룹을 선택합니다. Amazon RDS가 자동으로 DB 서브넷 그룹을 생성하도록 하려면 새 DB 서브넷 그룹 생성을 선택합니다. 자세한 내용은 DB 클러스터 사전 요구사항 (p. 95) 단원을 참조하십시오.
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 습기는 방법에 대한 자세한 내용은 VPC에 있는 DB 인스턴스를 인터넷에서 습기기 (p. 1017) 단원을 참조하십시오.
[Availability zone]	특정 가용 영역의 지정 여부를 결정합니다. 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 (p. 3) 단원을 참조하십시오.

옵션	설명
VPC 보안 그룹	Amazon RDS가 자동으로 VPC 보안 그룹을 생성하도록 하려면 Create new VPC security group(새 VPC 보안 그룹 생성)을 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 요구사항 (p. 95) 단원을 참조하십시오.
데이터베이스 포트	애플리케이션과 유ти리티가 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 클러스터의 DB 파라미터 그룹을 선택합니다. Aurora는 기본 DB 파라미터 그룹을 제공하며, DB 파라미터 그룹을 직접 생성할 수도 있습니다. DB 파라미터 그룹에 대한 자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.
DB 클러스터 파라미터 그룹	Aurora MySQL DB 클러스터의 DB 클러스터 파라미터 그룹을 선택합니다. Aurora는 기본 DB 클러스터 파라미터 그룹을 제공하며, DB 클러스터 파라미터 그룹을 직접 생성할 수도 있습니다. DB 클러스터 파라미터 그룹에 대한 자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.
암호화	새 Aurora DB 클러스터를 암호화하지 않으려면 [Disable encryption]을 선택합니다. 새 Aurora DB 클러스터를 유 휴 상태에서 암호화하려면 [Enable encryption]을 선택합니다. 암호화 활성을 선택하면 마스터 키 값으로 AWS KMS 암호화 키를 선택해야 합니다. MySQL DB 인스턴스가 암호화되어 있지 않으면 유 휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. MySQL DB 인스턴스가 암호화되어 있으면 지정된 암호화 키를 사용하여 유 휴 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.
Priority	DB 클러스터의 장애 조치 우선 순위를 선택합니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 기본 인스턴스의 결합으로부터 복구할 때 이 우선 순위에 따라 Aurora 복제본이 승격되는 순서가 결정됩니다. 자세한 내용은 Aurora DB 클러스터의 내결함성 (p. 268) 단원을 참조하십시오.
백업 보존 기간	Aurora가 데이터베이스 백업 사본을 보존하는 기간을 1~35 일로 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 확장 모니터링 (p. 358) 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-role라는 이름의 역할을 생성하도록 기본값을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 단원을 참조하십시오.
Granularity	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
Auto minor version upgrade	이 설정은 Aurora MySQL DB 클러스터에 적용되지 않습니다. Aurora MySQL의 엔진 업데이트에 대한 자세한 내용은 Amazon Aurora MySQL 데이터베이스 엔진 업데이트 (p. 686) 단원을 참조하십시오.
유지 관리 기간	[Select window]를 선택하고 시스템 유지 관리를 실행할 수 있는 주 단위 기간을 지정합니다. 또는 Amazon RDS가 임의로 기간을 지정하도록 하려면 기본 설정 없음을 선택합니다.

6. [Create read replica]를 선택합니다.

AWS CLI

원본 MySQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면 `create-db-cluster` 및 `create-db-instance` AWS CLI 명령을 사용하여 새 Aurora MySQL DB 클러스터를 생성하십시오. `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.

Linux, OS X, Unix의 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:master-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:master-mysql-
instance
```

콘솔을 사용하여 Aurora 읽기 전용 복제본을 생성하면 Amazon RDS에서 DB 클러스터 Aurora 읽기 전용 복제본의 기본 인스턴스를 자동으로 생성합니다. AWS CLI를 사용하여 Aurora 읽기 전용 복제본을 생성할 경

우 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

Linux, OS X, Unix의 경우:

```
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

원본 MySQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면 `CreateDBCluster` 및 `CreateDBInstance` Amazon RDS API 명령을 사용하여 새 Aurora DB 클러스터와 기본 인스턴스를 생성하십시오. 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.

`CreateDBCluster` Amazon RDS API 명령을 다음 파라미터와 함께 사용하여 원본 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 Key Management Service(AWS KMS) 암호화 키입니다.

- MySQL DB 인스턴스가 암호화되어 있지 않으면 유 휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. 안 그러면 계정용 기본 암호화 키를 사용할 때 DB 클러스터가 유 휴 상태에서 암호화됩니다.
- MySQL DB 인스턴스가 암호화되어 있으면 지정된 암호화 키를 사용하여 유 휴 상태에서 DB 클러스터가 암호화되도록 암호화 키를 지정합니다. 안 그러면 MySQL DB 인스턴스용 암호화 키를 사용할 때 DB 클러스터가 유 휴 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 \(Amazon RDS\)](#) 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 DB 클러스터를 생성합니다.

Example

```
https://rds.us-east-1.amazonaws.com/
?Action=CreateDBCluster
&DBClusterIdentifier=myreadreplicacluster
&DBSubnetGroupName=mysubnetgroup
&Engine=aurora
&ReplicationSourceIdentifier=mysqlmasterARN
&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 Amazon RDS에서 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=bee4aab750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

Aurora 읽기 전용 복제본 보기

AWS Management 콘솔 또는 AWS CLI를 사용하여 Aurora MySQL DB 클러스터의 MySQL과 Aurora MySQL의 복제 관계를 볼 수 있습니다.

콘솔

Aurora 읽기 전용 복제본의 마스터 MySQL DB 인스턴스를 보는 방법

1. AWS Management 콘솔에 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. 마스터 MySQL DB 인스턴스 정보가 [Replication source] 필드에 나타납니다.

aurora-mysql-db-cluster

Details

ARN

arn:aws:rds: [REDACTED]:aurora-mysql-db-cluster

DB cluster

aurora-mysql-db-cluster (available)

DB cluster role

Replica

Replication source

arn:aws:rds: [REDACTED]:mydbinstance3

Cluster endpoint

aurora-mysql-db-cluster. [REDACTED] rds.amazonaws.com

Reader endpoint

aurora-mysql-db-cluster. [REDACTED] rds.amazonaws.com

Port

3306

AWS CLI

AWS CLI를 사용하여 Aurora MySQL DB 클러스터의 MySQL과 Aurora MySQL의 복제 관계를 보려면 [describe-db-clusters](#) 및 [describe-db-instances](#) 명령을 사용하십시오.

어떤 MySQL DB 인스턴스가 마스터인지 결정하려면 [describe-db-clusters](#)를 사용하고 --db-cluster-identifier 옵션에서 Aurora 읽기 전용 복제본의 클러스터 식별자를 지정하십시오. 복제 마스터인 DB 인스턴스의 ARN 출력에서 ReplicationSourceIdentifier 요소를 참조하십시오.

어느 DB 클러스터가 Aurora 읽기 전용 복제본인지 확인하려면 [describe-db-instances](#)를 사용하여 --db-instance-identifier 옵션을 MySQL DB 인스턴스의 인스턴스 식별자를 지정하십시오. Aurora 읽기 전용 복제본의 DB 클러스터 식별자의 출력에서 ReadReplicaDBClusterIdentifiers 요소를 참조하십시오.

Example

Linux, OS X, Unix의 경우:

```
aws rds describe-db-clusters \
--db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances \
--db-instance-identifier mysqlmaster
```

Windows의 경우:

```
aws rds describe-db-clusters ^
--db-cluster-identifier myreadreplicacluster
```

```
aws rds describe-db-instances ^
--db-instance-identifier mysqlmaster
```

Aurora 읽기 전용 복제본 승격

マイグレーション이 완료되면 Aurora 읽기 전용 복제본을 독립형 DB 클러스터로 승격하여 클라이언트 애플리케이션을 Aurora 읽기 전용 복제본의 엔드포인트로 보낼 수 있습니다. Aurora 엔드포인트에 대한 자세한 내용은 [Amazon Aurora 연결 관리 \(p. 9\)](#) 단원을 참조하십시오. 승격은 상당히 빨리 완료되며, 승격 중에 Aurora 읽기 전용 복제본에 대한 읽기 및 쓰기가 가능합니다. 다만 승격 도중에 마스터 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 중단한다음 Aurora 읽기 전용 복제본에서의 복제 지연이 0이 될 때까지 기다리십시오. Aurora 읽기 전용 복제본의 복제본 지연 시간은 Aurora 읽기 전용 복제본에서 SHOW SLAVE STATUS 명령을 호출하고 Seconds behind master 값을 읽어 조회할 수 있습니다.

마스터에 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.

승격 후, 탐색 창에서 인스턴스를 선택하고 Aurora 읽기 전용 복제본에 Promoted Read Replica cluster to stand-alone database cluster(읽기 전용 복제본 클러스터를 독립형 데이터베이스 클러스터로 승격) 이벤트가 있는지 확인하여 승격이 완료되었는지 확인하십시오. 승격이 완료된 후에는 마스터 MySQL DB 인스턴스와 Aurora 읽기 전용 복제본의 링크가 해제되고, 원한다면 DB 인스턴스를 안전하게 삭제할 수 있습니다.

콘솔

Aurora 읽기 전용 복제본을 Aurora DB 클러스터로 승격시키려면

1. AWS Management 콘솔에 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 Replica]를 선택합니다.

AWS CLI

Aurora 읽기 전용 복제본을 독립형 DB 클러스터로 승격시키려면 `promote-read-replica-db-cluster` AWS CLI 명령을 사용하십시오.

Example

Linux, OS X, Unix의 경우:

```
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 \(p. 508\)](#)
- [Aurora DB 클러스터 역추적 \(p. 509\)](#)
- [오류 삽입 쿼리를 사용하여 Amazon Aurora 테스트 \(p. 522\)](#)
- [빠른 DDL을 이용하는 Amazon Aurora에서의 테이블 수정 \(p. 525\)](#)
- [Aurora DB 클러스터를 위한 볼륨 상태 표시 \(p. 526\)](#)

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 조정 \(p. 212\)](#) 단원을 참조하십시오.

DB 클러스터의 DB 인스턴스마다 DB 인스턴스 클래스를 수정하여 Aurora MySQL DB 클러스터의 규모를 조정할 수 있습니다. Aurora MySQL은 Aurora에 최적화된 몇 가지 DB 인스턴스 클래스를 지원합니다. Aurora MySQL에서 지원하는 DB 인스턴스 클래스의 사양 정보는 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 참조하십시오.

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, `max_connections` 파라미터의 값을 최대 16,000까지 설정하여 Aurora MySQL DB 인스턴스의 최대 연결 수를 늘릴 수 있습니다.

인스턴스 클래스	max_connec 기본값		
db.t2.small	45		
db.t2.medium	90		
db.t3.small	45		
db.t3.medium	90		
db.r3.large	1000		
db.r3.xlarge	2000		

인스턴스 클래스	max_connec 기본값		
db.r3.2xlarge	3000		
db.r3.4xlarge	4000		
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		

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` 값은 DB 인스턴스에 사용 가능한 메모리 용량을 나타냅니다. Aurora는 Aurora 관리 구성 요소에 대해 각 인스턴스에 일부 메모리를 예약합니다. `DBInstanceClassMemory`에 대한 이 조정으로 인해 연결된 DB 인스턴스 클래스에 대한 전체 메모리에 수식이 사용된 경우보다 `max_connections` 값이 낮아집니다.

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 DB 클러스터 역추적

MySQL과 호환되는 Amazon Aurora에서는 백업에서 데이터를 복구하지 않고도 특정 시간으로 DB 클러스터를 되감을 수 있습니다.

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 돋기 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 \(p. 268\)](#) 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 클러스터를 업데이트하면 변경 레코드가 생성됩니다. Aurora는 대상 기간에 대한 변경 레코드를 보존하며 이 레코드를 저장하기 위해 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 가능성성이 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 정보는 [기존 역추적 검색 \(p. 521\)](#) 단원을 참조하십시오.

역추적 제한

역추적에는 다음과 같은 제한이 적용됩니다.

- 역추적 기능이 활성화된 상태에서 생성한 DB 클러스터에서만 역추적이 가능합니다. 새 DB 클러스터를 만들거나 DB 클러스터의 스냅샷을 복원할 때 역추적 기능을 활성화할 수 있습니다. 역추적 기능을 활성화된 DB 클러스터의 경우 역추적 기능이 활성화된 복제 DB 클러스터를 생성할 수 있습니다. 현재는 역추적 기능이 비활성화된 상태에서 생성한 DB 클러스터에서 역추적이 불가능합니다.
- 역추적 기간 제한은 72시간입니다.
- 역추적은 전체 DB 클러스터에 영향을 미칩니다. 예를 들어 단일 테이블 또는 단일 데이터 업데이트를 선택적으로 역추적할 수 없습니다.
- 이진 로그(binlog) 복제에는 역추적이 지원되지 않습니다. 구성하거나 역추적을 사용하기 전에는 반드시 교차 리전 복제를 비활성화해야 합니다.
- 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 클러스터에서 데이터베이스 복제 \(p. 239\)](#) 단원을 참조하십시오.
- 역추적하면 DB 인스턴스가 잠시 종단됩니다. 역추적 작업을 시작하기 전에는 새 읽기 또는 쓰기 요청이 없도록 애플리케이션을 종단하거나 일시 종지해야 합니다. 역추적 작업 중에 Aurora는 데이터베이스를 일시 종지하고 열려 있는 연결을 닫으며 커밋되지 않은 읽기 및 쓰기를 모두 종단합니다. 그리고 역추적 작업이 완료되기를 기다립니다.
- 다음 AWS 리전에서는 역추적이 지원되지 않습니다.
 - 중국(닝샤)
 - 아시아 태평양(홍콩)
 - 유럽(스톡홀름)
 - 중동(바레인)
 - 남아메리카(상파울루)
- 역추적을 지원하지 않는 AWS 리전에서는 역추적 사용 클러스터의 리전 간 스냅샷을 복원할 수 없습니다.
- Aurora 멀티 마스터 클러스터에서 역추적을 사용할 수 없습니다.

역추적은 MySQL 5.6과 호환되는 Aurora MySQL 1.*에 사용할 수 있습니다. 또한 MySQL 5.7과 호환되는 Aurora MySQL 2.06 이상에도 사용할 수 있습니다. Aurora MySQL 2.* 버전 요구 사항으로 인해 역추적 설정이 활성화된 Aurora MySQL 1.* 클러스터를 생성한 경우 역추적 호환 버전인 Aurora MySQL 2.*로만 업그레이드할 수 있습니다. 이러한 요구 사항은 다음 유형의 업그레이드 경로에 영향을 미칩니다.

- Aurora MySQL 1.* DB 클러스터의 스냅샷을 역추적 호환 버전인 Aurora MySQL 2.*로만 복원할 수 있습니다.
- Aurora MySQL 1.* DB 클러스터에서 특정 시점으로 복구를 수행하는 방법으로만 역추적 호환 버전인 Aurora MySQL 2.*로 복원할 수 있습니다.

이러한 업그레이드 요구 사항은 Aurora MySQL 1.* 클러스터에 대한 역추적을 끈다 해도 여전히 적용됩니다.

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 지정하십시오. Aurora는 이 시간을 지원하기 위해 변경 레코드를 충분히 보존해야 합니다.

콘솔

새 DB 클러스터를 생성할 때 콘솔을 사용하여 역추적을 구성할 수 있습니다. 역추적을 활성화하도록 DB 클러스터를 수정할 수도 있습니다.

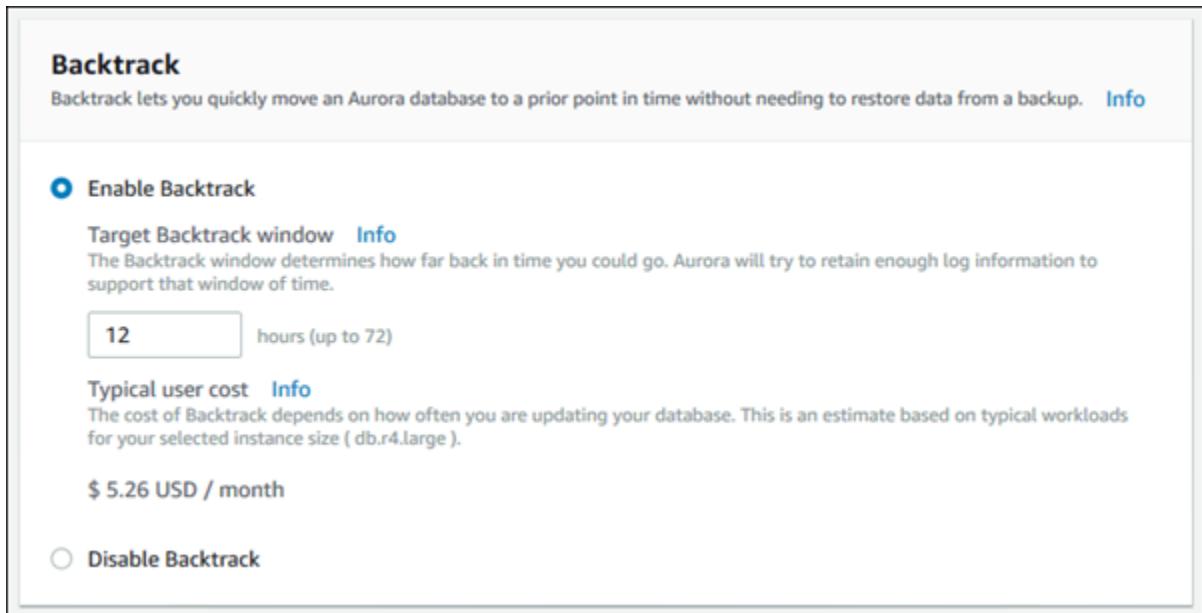
주제

- DB 클러스터 생성 시 콘솔로 역추적 구성 (p. 512)
- DB 클러스터 수정 시 콘솔로 역추적 구성 (p. 512)

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 \(p. 95\)](#) 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.

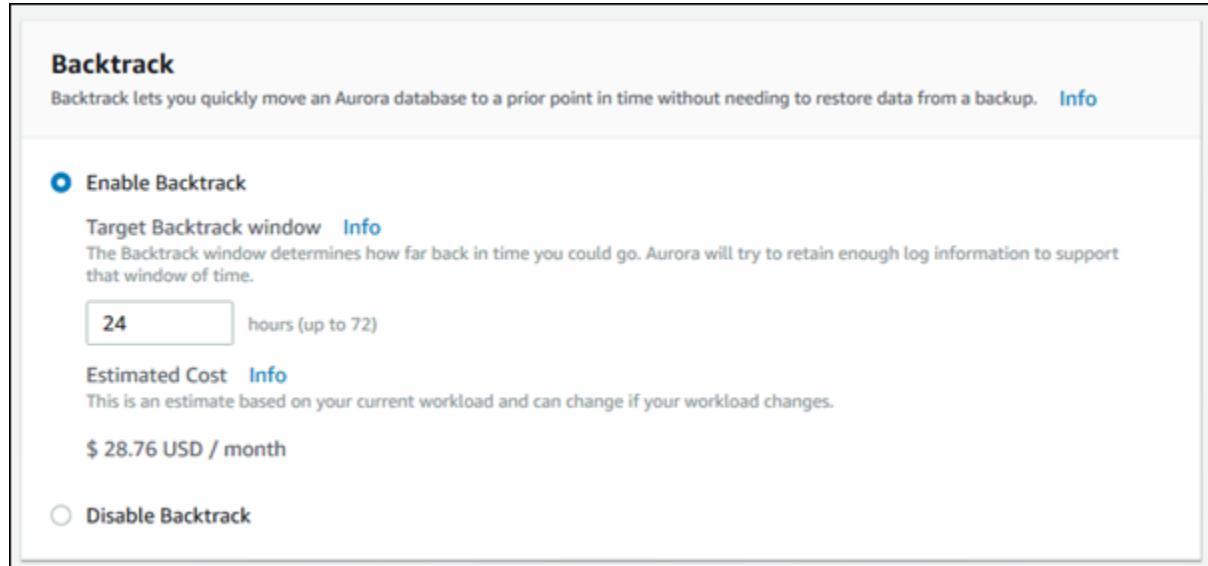
콘솔을 사용하여 DB 클러스터의 역추적을 수정하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 수정할 클러스터를 선택하고 수정을 선택합니다.
4. 역추적 섹션에 역추적이 비활성화되어 있으면 역추적 활성화를 선택합니다.

Note

현재는 역추적 기능이 활성화된 상태에서 생성한 DB 클러스터에만 역추적을 활성화할 수 있습니다. 역추적 기능이 비활성화된 상태에서 생성한 DB 클러스터에는 역추적 섹션이 보이지 않습니다.

5. 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.
6. [Continue]를 선택합니다.
 7. 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) 설정을 적용합니다.
 - 즉시 적용 – Target Backtrack window(대상 역추적 기간) 설정을 가급적 빨리 적용합니다.
 8. 클러스터 설정을 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 정합니다. 자세한 정보는 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

다음 AWS CLI 명령으로 `--backtrack-window` 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.

Linux, OS X, Unix의 경우:

```
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 정보는 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

다음 API 작업을 사용하여 `BacktrackWindow` 값을 지정할 수도 있습니다.

- [ModifyDBCluster](#)
- [RestoreDBClusterFromS3](#)
- [RestoreDBClusterFromSnapshot](#)
- [RestoreDBClusterToPointInTime](#)

Note

현재는 역추적 기능이 활성화된 상태에서 생성한 DB 클러스터에만 역추적을 활성화할 수 있습니다.

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

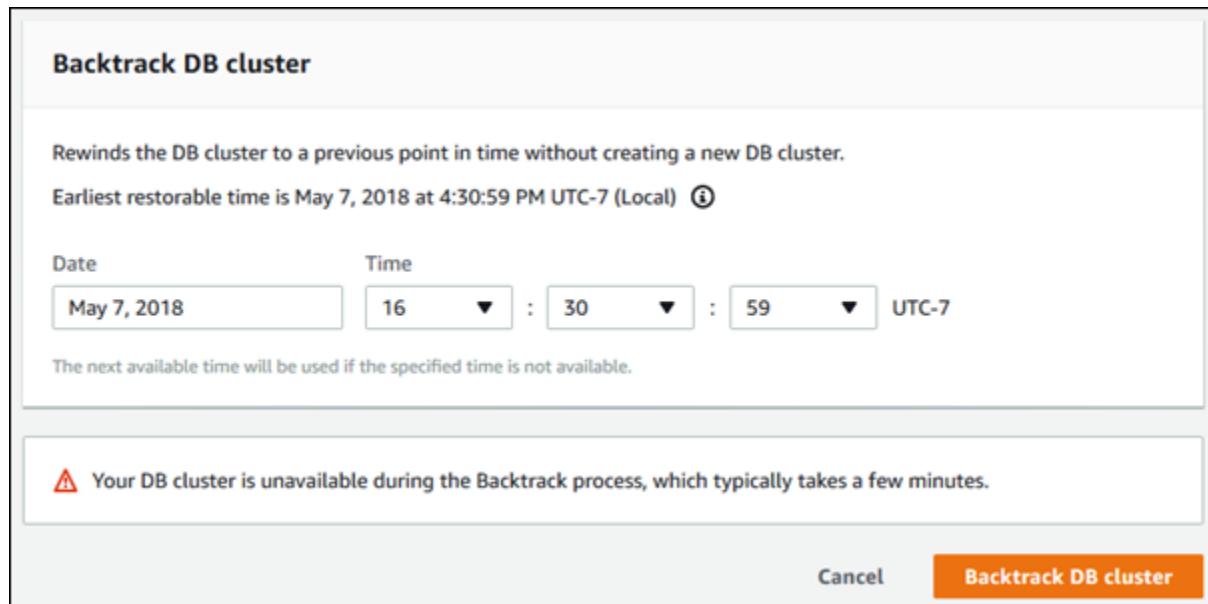
데이터베이스 복제본의 경우 복제본이 생성된 날짜와 시간 이전으로 DB 클러스터를 역추적할 수 없습니다. 데이터베이스 복제에 대한 자세한 정보는 [Aurora DB 클러스터에서 데이터베이스 복제 \(p. 239\)](#) 단원을 참조하십시오.

콘솔

다음 절차에서는 콘솔을 사용하여 DB 클러스터의 역추적 작업을 수행하는 방법을 설명합니다.

콘솔로 역추적 작업을 수행하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 인스턴스를 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.

Linux, OS X, Unix의 경우:

```
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.

역추적 모니터링

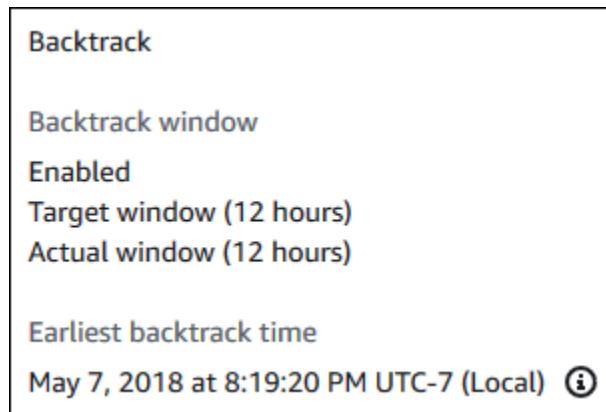
DB 클러스터에 대한 역추적 정보를 보고 역추적 지표를 모니터링할 수 있습니다.

콘솔

콘솔을 이용하여 역추적 정보를 보고 역추적 지표를 모니터링하려면

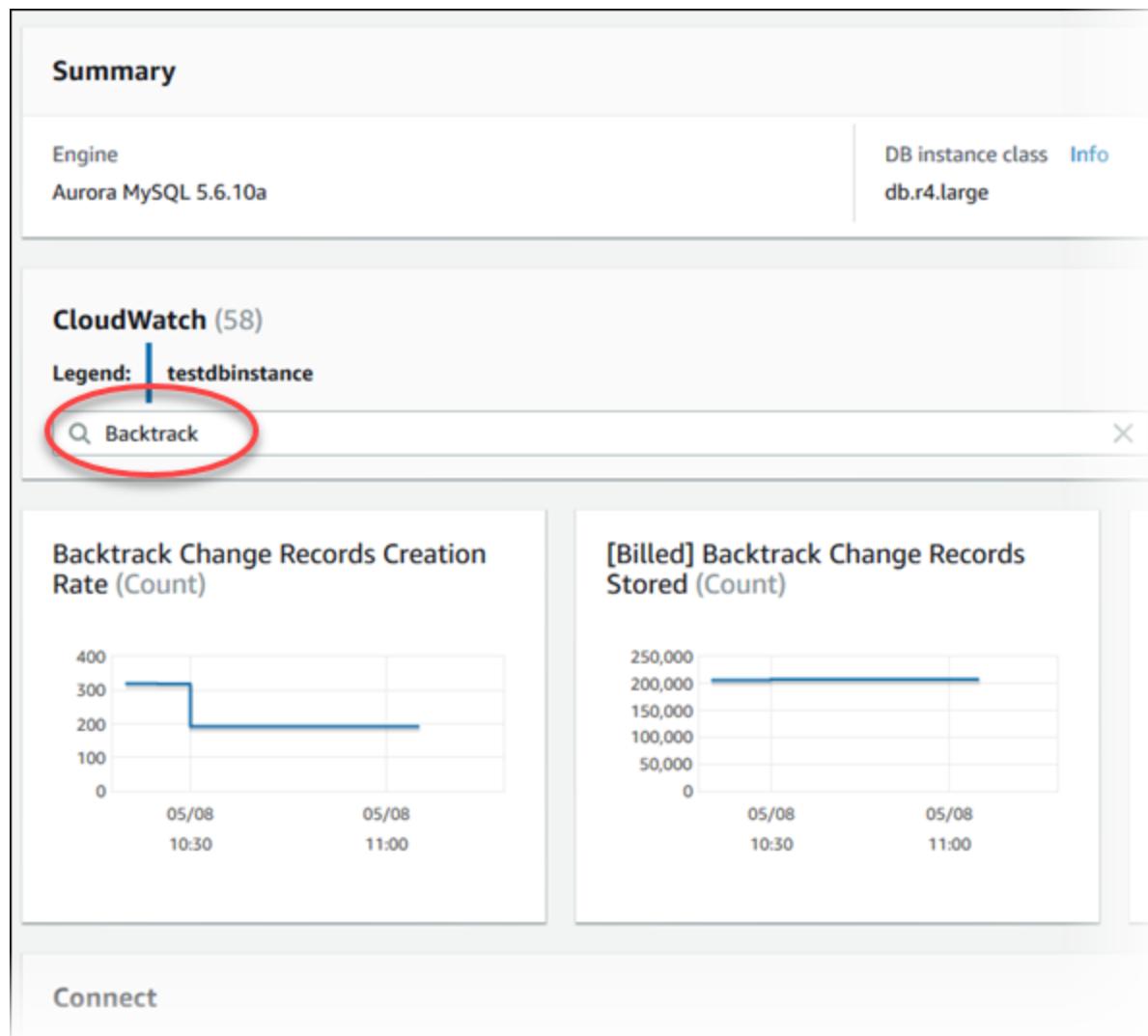
1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 데이터베이스를 선택합니다.
3. 정보를 열 DB 클러스터 이름을 선택합니다.

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 인스턴스를 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 표상으로 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.

Linux, OS X, Unix의 경우:

```
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 콘솔에 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.

Enabled

- Yes
- No

Target

Send notifications to

- ARN
- New email topic
- New SMS topic

Topic name

Name of the topic.

With these recipients

Email addresses or phone numbers of SMS enabled devices to send the notifications to

e.g. user@domain.com

Source

Source type

Source type of resource this subscription will consume event from



Instances to include

Instances that this subscription will consume events from

- All instances
- Select specific instances

Specific instances



Event categories to include

Event categories that this subscription will consume events from

- All event categories
- Select specific event categories

520

Specific event



10. Create를 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.

Linux, OS X, Unix의 경우:

```
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.

DB 클러스터의 역추적 비활성화

DB 클러스터의 역추적 기능을 비활성화할 수 있습니다.

콘솔

콘솔을 사용하여 DB 클러스터의 역추적을 비활성화할 수 있습니다.

콘솔을 사용하여 DB 클러스터의 역추적 기능을 비활성화하려면

1. AWS Management 콘솔에 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 0(제로)으로 설정하면 AWS CLI를 사용하여 DB 클러스터의 역추적 기능을 비활성화할 수 있습니다.

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` – 0.

다음은 `sample-cluster`를 `--backtrack-window`으로 설정함으로써 0의 역추적 기능을 비활성화하는 예제입니다.

Linux, OS X, Unix의 경우:

```
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.

오류 삽입 쿼리를 사용하여 Amazon Aurora 테스트

오류 삽입 쿼리를 사용하여 Amazon Aurora DB 클러스터의 내결합성을 테스트할 수 있습니다. 오류 삽입 쿼리는 Amazon Aurora 인스턴스에 SQL 명령으로 실행되며 오류 삽입 쿼리를 통해 다음 이벤트 중 하나의 발생에 대한 시뮬레이션을 예약하는 데 사용됩니다.

- 라이터 또는 리더 DB 인스턴스의 충돌
- Aurora 복제본의 실패
- 디스크 실패
- 디스크 정체

오류 삽입 쿼리가 충돌을 지정하면 Aurora 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 앤드포인트에 연결하여 오류 삽입 쿼리를 Aurora 복제본 중 하나로 제출할 수 있습니다. 자세한 정보는 [Amazon Aurora 연결 관리 \(p. 9\)](#) 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 콘솔의 DB 클러스터에 대한 장애 조치 인스턴스 작업을 선택하거나 `failover-db-cluster` AWS CLI 명령 또는 `FailoverDBCluster` RDS API 작업을 사용하십시오.

구문

```
ALTER SYSTEM CRASH [ INSTANCE | DISPATCHER | NODE ];
```

옵션

이 오류 삽입 쿼리에는 다음 충돌 유형 중 하나가 사용됩니다.

- **INSTANCE**—Amazon Aurora 인스턴스용 MySQL 호환 데이터베이스의 충돌이 시뮬레이션됩니다.
- **DISPATCHER**—Aurora DB 클러스터용 마스터 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 지정된 시간 간격 동안 DB 클러스터의 특정 Aurora 복제본 또는 모든 Aurora 복제본에 대한 요청이 모두 차단됩니다. 시간 간격이 경과되면 해당 Aurora 복제본이 마스터 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 TO와 Aurora 복제본의 이름을 지정하십시오. 기본 실패 유형은 TO ALL입니다.
- **quantity**—Aurora 복제본 실패 시뮬레이션 시간 길이. 이 간격 값 뒤에 시간 단위를 지정해야 합니다. 지정된 단위의 시간 동안 시뮬레이션이 발생합니다. 예를 들어 20 MINUTE를 지정하면 20분 동안 시뮬레이션이 실행됩니다.

Note

Aurora 복제본 실패 이벤트에 대한 시간 간격을 지정할 때는 각별히 주의하십시오. 시간 간격을 너무 길게 지정하고 마스터 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 종에 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 정보는 [Aurora DB 클러스터를 위한 볼륨 상태 표시 \(p. 526\)](#) 단원을 참조하십시오.
- **NODE index**—실패 이벤트를 시뮬레이션할 특정 스토리지 노드. 가용 스토리지 노드의 범위를 초과하는 경우, 지정할 수 있는 최대 인덱스 값을 알려주는 오류가 발생합니다. 자세한 정보는 [Aurora DB 클러스터를 위한 볼륨 상태 표시 \(p. 526\)](#) 단원을 참조하십시오.
- **quantity**—디스크 실패를 시뮬레이션하는 시간 길이. 이 간격 값 뒤에 시간 단위를 지정해야 합니다. 지정된 단위의 시간 동안 시뮬레이션이 발생합니다. 예를 들어 20 MINUTE를 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 종에 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 임의의 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 MINUTE를 지정하면 20분 동안 시뮬레이션이 실행됩니다.

빠른 DDL을 이용하는 Amazon Aurora에서의 테이블 수정

MySQL에서 많은 데이터 정의 언어(DDL) 작업이 상당한 성능 임팩트를 가집니다. 성능 임팩트는 최근의 온라인 DDL 개선에도 불구하고 발생합니다.

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 롤 적용
- 동시에 발생하는 DML 처리량 표시

In Amazon Aurora에서 빠른 DDL을 이용해 ALTER TABLE 작업을 거의 동시에 실행할 수 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 만들입니다.

Important

현재 Aurora MySQL에 대해 빠른 DDL을 사용하려면 Aurora 랩 모드를 활성화해야 합니다. 프로덕션 DB 클러스터에는 빠른 DDL을 사용하지 않는 것이 좋습니다. Aurora 랩 모드 활성화에 대한 내용은 [Amazon Aurora MySQL 랩 모드 \(p. 654\)](#) 단원을 참조하십시오.

Note

빠른 DDL은 Aurora 버전 1.12 이상에서 사용할 수 있습니다. Aurora 버전에 대한 자세한 내용은 [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#) 단원을 참조하십시오.

제한 사항

현재 빠른 DDL에는 다음과 같은 제한 사항이 있습니다.

- 빠른 DDL은 기존 테이블 끝까지 기본값 없이 null이 허용된 열 추가만 지원합니다.
- 빠른 DDL은 분할된 테이블을 지원하지 않습니다.
- 빠른 DDL은 중복 행 형식을 사용하는 InnoDB 테이블을 지원하지 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.

Note

최대 레코드 크기 확인이 Aurora 1.15 버전에 추가되었습니다.

구문

```
ALTER TABLE tbl_name ADD COLUMN col_name column_definition
```

옵션

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.

Aurora 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 대해 자세히 알아보려면, AWS 데이터베이스 블로그의 [Aurora 스토리지 엔진 소개](#)를 참조하십시오.

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 정보는 [오류 삽입 쿼리를 사용하여 Amazon Aurora 테스트 \(p. 522\)](#) 단원을 참조하십시오.

SHOW VOLUME STATUS 문을 사용하여 오류를 피할 수 있습니다. 이 문은 두 서버 상태 변수, Disks 및 Nodes를 반환합니다. 이러한 변수는 DB 클러스터 블록에 대해 각각 논리 데이터 블록과 스토리지 노드의 총 수를 표시합니다.

Note

SHOW VOLUME STATUS 문은 Aurora 버전 1.12 및 이후 버전에서 사용 가능합니다. Aurora 버전에 대한 자세한 내용은 [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#) 단원을 참조하십시오.

구문

```
SHOW VOLUME STATUS
```

예

다음 예는 전형적인 SHOW VOLUME STATUS 결과를 보여줍니다.

```
mysql> SHOW VOLUME STATUS;
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Disks         | 96    |
| Nodes         | 74    |
+-----+-----+
```

Amazon Aurora MySQL용 Parallel Query 처리

아래에서 MySQL과 호환되는 Amazon Aurora의 병렬 쿼리 성능 최적화에 대한 설명을 확인할 수 있습니다. 이 기능은 특정 데이터 집약적인 쿼리를 위한 특수한 실행 경로를 사용하면서 Aurora 공유 스토리지 아키텍처를 활용합니다. 현재, MySQL 5.6과 호환되는 Aurora MySQL 버전은 병렬 쿼리를 지원합니다. 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 \(p. 528\)](#)
- [병렬 쿼리 클러스터 관리 \(p. 530\)](#)
- [병렬 쿼리를 위한 업그레이드 고려 사항 \(p. 530\)](#)
- [병렬 쿼리에서 작동하는 DB 클러스터 생성 \(p. 530\)](#)
- [병렬 쿼리 활성화 및 비활성화 \(p. 534\)](#)
- [병렬 쿼리를 위한 성능 튜닝 \(p. 536\)](#)
- [병렬 쿼리를 활용하기 위해 스키마 객체 생성 \(p. 536\)](#)
- [어떤 문이 병렬 쿼리를 사용하는지 확인 \(p. 536\)](#)
- [병렬 쿼리 모니터링 \(p. 539\)](#)
- [병렬 쿼리가 SQL 구조에서 작동하는 방법 \(p. 540\)](#)

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.

병렬 쿼리 기능이 활성화되면, Aurora MySQL 엔진이 힌트 또는 테이블 속성과 같은 SQL 변경 필요 없이도 쿼리가 혜택을 얻을 수 있는 경우를 자동으로 결정합니다. 다음 단원에서는 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 완료되기까지 몇 분 또는 몇 시간이 걸리는 장기 실행 쿼리에 가장 많은 혜택을 제공합니다. Aurora MySQL은 일반적으로 비용이 많이 드는 쿼리의 경우 또는 쿼리 캐싱, 버퍼풀 캐싱 또는 인덱스 조회와 같은 다른 최적화 기법이 더 적절한 경우에는 병렬 쿼리 최적화를 수행하지 않습니다. 병렬 쿼리가 사용될 것으로 예상될 때 사용되지 않고 있는 경우 [어떤 문이 병렬 쿼리를 사용하는지 확인 \(p. 536\)](#)를 참조하십시오.

주제

- [혜택 \(p. 528\)](#)
- [아키텍처 \(p. 528\)](#)
- [제한 사항 \(p. 529\)](#)

혜택

병렬 쿼리를 사용하면 Aurora MySQL 테이블에서 데이터 집약적인 분석 쿼리를 실행할 수 있으며, 많은 사례에서 쿼리 처리를 위한 전통적인 분업에 비해 수치상 성능 향상이 나타났습니다.

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, Aurora는 결과 집합에 필요한 열 값만 포함된 간소화된 퓨플을 전송합니다.
- 푸시 다운 기능 실행, 행 필터링 및 WHERE 절에 대한 열 프로젝션으로 인한 헤드 노드에 대한 CPU 사용량의 감소.
- 버퍼풀에서의 메모리 압력 감소. 병렬 쿼리에 의해 처리된 페이지는 버퍼풀에 추가되지 않으므로, 버퍼풀에서 자주 사용되는 데이터를 제거하는 데이터 집약적인 스캔의 발생 기회가 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 트래픽 감소. Aurora MySQL은 이러한 기법을 사용하여 다시 실행 처리와 같은 쓰기 집약적인 작업의 속도를 높입니다. 병렬 쿼리는 읽기 작업에도 동일한 원칙을 적용합니다.

Note

Aurora MySQL 병렬 쿼리의 아키텍처는 다른 데이터베이스 시스템에서 이름이 비슷한 기능의 아키텍처와는 다릅니다. Aurora MySQL 병렬 쿼리는 대칭적 다중 처리(SMP)를 수반하지 않으며 때

라서 데이터베이스 서버의 CPU 용량에 의존하지 않습니다. 병렬 실행은 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 전송하고 그 단일 노드의 단일 스레드에서 모든 추가 처리를 수행하게 됩니다. 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.

제한 사항

다음 제한 사항이 병렬 쿼리 기능에 적용됩니다.

- 현재, MySQL 5.6과 호환되는 Aurora MySQL 버전은 병렬 쿼리를 지원합니다. (PostgreSQL 데이터베이스 엔진에는 "병렬 쿼리"라고도 하는 관련 없는 기능이 있음을 유념하십시오.)
- 현재, 다음 인스턴스 클래스에서만 병렬 쿼리를 사용할 수 있습니다.
 - db.r3 시리즈의 모든 인스턴스 클래스.
 - db.r4 시리즈의 모든 인스턴스 클래스.
 - db.r5 시리즈에서 Aurora MySQL가 지원하는 모든 인스턴스 클래스.

Note

병렬 쿼리를 위한 db.t2 또는 db.t3 인스턴스를 생성할 수 없습니다.

- 병렬 쿼리 옵션은 다음 리전에서 사용할 수 있습니다.
 - 미국 동부(버지니아 북부)
 - 미국 동부(오하이오)
 - 미국 서부(오레곤)
 - 유럽(아일랜드)
 - 아시아 태평양(도쿄)
- 병렬 쿼리를 사용할 경우 다음 설명에 따라 새 클러스터를 생성하거나 기존 Aurora MySQL 클러스터 스냅샷에서 복원해야 합니다.
 - 성능 개선 도우미 기능은 병렬 쿼리를 위해 활성화된 클러스터에는 현재 사용할 수 없습니다.
 - 역추적 기능은 병렬 쿼리를 위해 활성화된 클러스터에는 현재 사용할 수 없습니다.
 - 병렬 쿼리를 위해 활성화된 DB 클러스터는 중단 및 시작할 수 없습니다.
 - 현재, 분할된 테이블은 병렬 쿼리에 대해 지원되지 않습니다. 병렬 쿼리 클러스터에서는 분할된 테이블을 사용할 수 있습니다. 이러한 테이블에 대한 쿼리에서는 비병렬 쿼리 실행 경로를 사용합니다.

Note

일부 쿼리 블록이 분할된 테이블을 참조하는 경우에도, 조인 쿼리, 통합 쿼리 또는 다른 멀티파트 쿼리는 병렬 쿼리를 부분적으로 사용할 수 있습니다. 분할되지 않은 테이블만 참조하는 쿼리 블록은 병렬 쿼리 최적화를 사용할 수 있습니다.

- 병렬 쿼리를 사용하여 작업하려면, 현재 테이블이 InnoDB 스토리지 엔진의 Antelope 파일 형식을 요구하는 COMPACT 행 형식을 사용해야 합니다.
- TEXT, BLOB 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.

JSON 데이터 형식은 MySQL 5.7과 호환되는 Aurora에서만 사용할 수 있는 BLOB 같은 형식입니다. 병렬 쿼리는 MySQL 5.6과 호환되는 Aurora에서만 사용할 수 있습니다. 그러므로, 이 형식은 현재 병렬 쿼리 기능을 포함한 클러스터의 테이블에 존재할 수 없습니다.

- 현재, 병렬 쿼리는 전체 텍스트 검색 인덱스를 포함한 테이블에 사용되지 않습니다(쿼리가 그러한 인덱싱된 열을 참조하든 MATCH() 연산자를 사용하든 관계 없이).
- 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.
- 현재, 상관관계가 있는 하위 쿼리는 병렬 쿼리 최적화를 사용할 수 없습니다.
- 병렬 쿼리는 쓰기 또는 잠금이 없는 SELECT 문에서 REPEATABLE READ 격리 수준에서만 작동합니다. 예를 들어, 병렬 쿼리는 UPDATE 또는 DELETE 문의 SELECT FOR UPDATE 또는 WHERE 절에서는 작동하지 않습니다.
- 병렬 쿼리는 대기 중인 빠른 온라인 데이터 정의 언어(DDL) 작업이 없는 테이블에서만 사용할 수 있습니다.
- 병렬 쿼리 기능은 WHERE 절에서 거의 대부분 연산자 및 기능에서 작동합니다. 호환되는 작업을 보여주는 예는 [병렬 쿼리가 SQL 구조에서 작동하는 방법 \(p. 540\)](#)을 참조하십시오.
- 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 변수 \(p. 539\)](#)를 사용하여 활성 세션의 수를 모니터링할 수 있습니다. 상태 변수 `Aurora_pg_max_concurrent_requests`를 쿼리하여 주어진 DB 인스턴스의 동시 세션에 대한 제한을 확인할 수 있습니다.
- 현재, 병렬 쿼리는 AWS Identity and Access Management(IAM) 데이터베이스 인증을 지원하지 않습니다.

병렬 쿼리 클러스터 관리

병렬 쿼리를 위해 활성화된 클러스터 관리는 설정 단계(전체 Aurora MySQL 클러스터 생성 또는 복원) 및 클러스터 전체에 걸쳐 얼마나 광범위하게 병렬 쿼리를 활성화할지 결정이 필요합니다.

병렬 쿼리가 유용한 경우(예를 들어, 테이블을 분할되지 않게 하거나 전체 텍스트 검색 인덱스를 제거하기 위해) 일부 대용량 테이블의 새 버전을 만들어야 할 수도 있습니다. 세부 정보는 [병렬 쿼리를 활용하기 위해 스키마 객체 생성 \(p. 536\)](#) 단원을 참조하십시오.

설정이 완료된 후, 지속적인 관리를 위해 성능 모니터링 및 병렬 쿼리 사용의 장애물 제거가 수반됩니다. 해당 지침은 [병렬 쿼리를 위한 성능 튜닝 \(p. 536\)](#) 단원을 참조하십시오.

병렬 쿼리를 위한 업그레이드 고려 사항

현재, 병렬 쿼리 기능을 활성화하기 위한 Aurora MySQL 클러스터의 실행 중 업그레이드를 수행할 수 없습니다. 병렬 쿼리를 사용할 경우 새 클러스터를 생성하거나 기존 Aurora MySQL 5.6 클러스터 스냅샷에서 복원해야 합니다.

병렬 쿼리에서 작동하는 DB 클러스터 생성

병렬 쿼리를 사용하여 Aurora MySQL 클러스터를 생성하려면, 새 인스턴스를 추가하거나 다른 Aurora MySQL 클러스터에서 수행하는 것과 동일한 AWS Management 콘솔 및 AWS CLI 기법을 사용하는 다른 관리 작업을 수행합니다. 병렬 쿼리에서 작동하는 새 클러스터를 생성할 수 있습니다. 또한 MySQL 5.6과 호환되는 데이터베이스의 스냅샷에서 복원하여 병렬 쿼리에서 작동하는 DB 클러스터도 생성할 수 있습니다. 새 Aurora MySQL 클러스터를 생성하는 절차에 익숙하지 않은 경우, [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#)에서 배경 정보를 확인할 수 있습니다.

하지만, 생성 옵션은 다음과 같이 각기 다릅니다.

- Aurora MySQL 엔진 버전을 선택하는 경우 MySQL 5.6과 호환되는 최신 엔진을 선택해야 합니다. 현재, MySQL 5.6과 호환되는 Aurora MySQL 버전은 병렬 쿼리를 지원합니다.
- DB 클러스터를 생성하거나 복원하는 경우 parallelquery 엔진 모드를 선택해야 합니다.

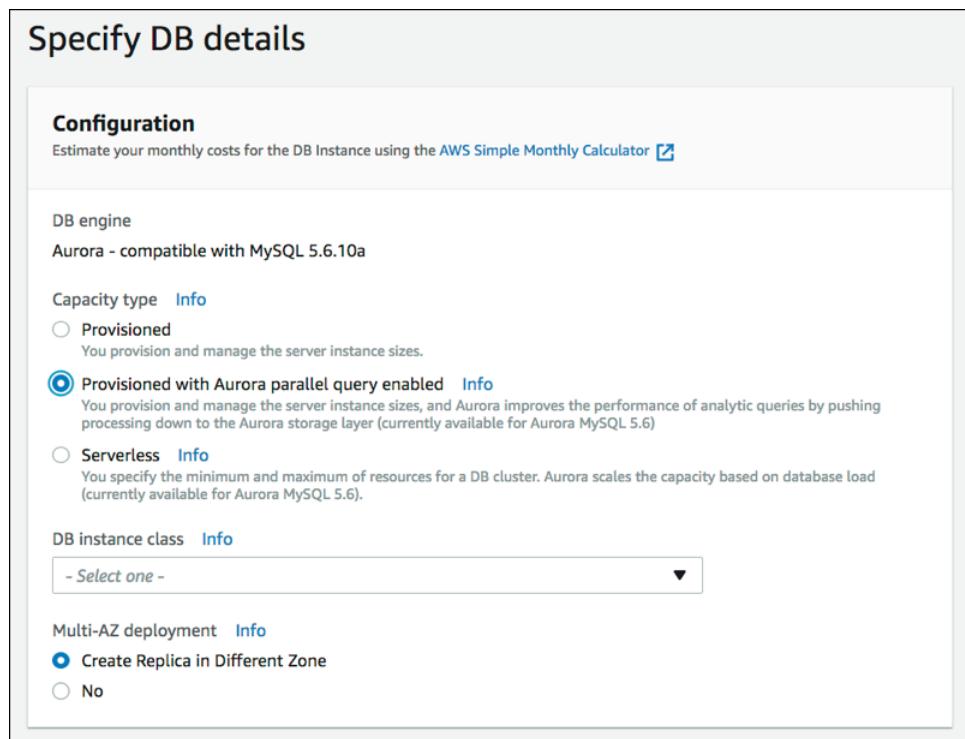
새 클러스터를 생성하든 스냅샷에서 복원하든, 새 DB 인스턴스를 추가하기 위해 다른 Aurora MySQL 클러스터에서 수행하는 것과 동일한 기법을 사용합니다.

콘솔을 사용하여 병렬 쿼리 클러스터 생성

다음 설명에 따라 콘솔을 사용하여 새 병렬 쿼리 클러스터를 생성할 수 있습니다.

AWS Management 콘솔을 사용하여 병렬 쿼리 클러스터를 생성하려면

1. [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#)의 일반 AWS Management 콘솔 절차를 따르십시오.
2. 엔진 선택 화면에서 Aurora의 MySQL 5.6 호환 에디션을 선택합니다.
3. DB 세부 정보 지정 화면에서, 다음 스냅샷에서처럼 용량 유형에 대하여 Aurora 병렬 쿼리가 활성화된 상태로 프로비저닝 완료를 선택합니다.



AWS Management 콘솔을 사용하여 스냅샷을 병렬 쿼리 클러스터에 복원하려면

1. MySQL 5.6 호환 데이터베이스 인스턴스의 스냅샷을 찾습니다.
2. [DB 클러스터 스냅샷에서 복원 \(p. 273\)](#)의 일반 AWS Management 콘솔 절차를 따르십시오.
3. DB 엔진 모드에 대하여 아래 스냅샷과 같이 parallelquery를 선택합니다.

Restore DB Instance

You are creating a new DB Instance from a source DB Instance at a specified time. This new DB Instance will have the default DB Security Group and DB Parameter Groups.

Instance specifications

DB Engine
Name of the Database Engine
Aurora MySQL

DB Engine Version
Version Number of the Database Engine to be used for this instance
Aurora (MySQL)-5.6.10a (default)

Capacity type [Info](#)

Provisioned
You provision and manage the server instance sizes.

Provisioned with Aurora parallel query enabled [Info](#)
You provision and manage the server instance sizes, and Aurora improves the performance of analytic queries by pushing processing down to the Aurora storage layer (currently available for Aurora MySQL 5.6)

Serverless [Info](#)
You specify the minimum and maximum of resources for a DB cluster. Aurora scales the capacity based on database load (currently available for Aurora MySQL 5.6).

새 클러스터가 병렬 쿼리를 사용할 수 있는지 확인하려면

- 앞에 나온 기법을 사용하여 클러스터를 생성하거나 복제합니다.
- `aurora_pq_supported` 구성 설정이 맞는지 확인합니다.

```
mysql> select @@aurora_pq_supported;
+-----+
| @@aurora_pq_supported |
+-----+
| 1 |
+-----+
```

CLI를 사용하여 병렬 쿼리 클러스터 생성

다음 설명에 따라 CLI를 사용하여 새 병렬 쿼리 클러스터를 생성할 수 있습니다.

AWS CLI를 사용하여 병렬 쿼리 클러스터를 생성하려면

- Aurora MySQL 버전, AWS 인스턴스 클래스, AWS 리전, 가용 영역 등의 어떤 조합을 병렬 쿼리 클러스터에 사용할 수 있는지 확인합니다. 그렇게 하려면, JSON 형식으로 출력을 생성하는 `aws rds describe-orderable-db-instance-options` 명령을 사용합니다. 다음 코드 예제는 지정된 AWS 리전에서 병렬 쿼리 클러스터에 사용할 수 있는 조합을 확인하는 방법을 보여줍니다.

```
aws rds describe-orderable-db-instance-options --
engine aurora --query 'OrderableDBInstanceOptions[?
contains(SupportedEngineModes, `parallelquery`)==`true`].
{Engine:Engine,EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes,Region:Region} --region us-east-1'
```

위의 명령은 다음과 비슷한 출력을 생성합니다.

```
[  
  {  
    "DBInstanceClass": "db.r3.2xlarge",  
    "EngineVersion": "5.6.10a",  
    "Engine": "aurora",  
    "SupportedEngineModes": [  
      "parallelquery"  
    ]  
  },  
  {  
    "DBInstanceClass": "db.r3.4xlarge",  
    "EngineVersion": "5.6.10a",  
  ...  
]
```

2. Amazon Aurora DB 클러스터 생성 (p. 95)의 일반 AWS CLI 절차를 따르십시오.

3. 다음 옵션 세트를 지정하십시오.

- --engine 옵션의 경우 aurora를 사용합니다.
- --engine-mode 옵션의 경우 parallelquery를 사용합니다. --engine-mode 파라미터는 create-db-cluster 작업에 적용됩니다. 그러면 클러스터의 엔진 모드가 후속 create-db-instance 작업에 의해 자동으로 사용됩니다.
- --engine-version 옵션의 경우 5.6.10a를 사용합니다.

다음 코드 예제에서는 작업 방법을 보여줍니다.

```
aws rds create-db-cluster --db-cluster-identifier $CLUSTER_ID  
  --engine aurora --engine-mode parallelquery --engine-version 5.6.10a \  
  --master-username $MASTER_USER_ID --master-user-password $MASTER_USER_PW \  
  --db-subnet-group-name $SUBNET_GROUP --vpc-security-group-ids $SECURITY_GROUP  
  
aws rds create-db-instance --db-instance-identifier ${INSTANCE_ID}-1 \  
  --engine aurora \  
  --db-cluster-identifier $CLUSTER_ID --db-instance-class $INSTANCE_CLASS
```

4. 생성하거나 복원한 클러스터에서 병렬 쿼리 기능을 사용할 수 있는지 확인합니다.
aurora_pq_supported 구성 설정이 맞는지 확인합니다.

```
mysql> select @@aurora_pq_supported;  
+-----+  
| @@aurora_pq_supported |  
+-----+  
|          1 |  
+-----+
```

AWS CLI를 사용하여 스냅샷을 병렬 쿼리 클러스터에 복원하려면

1. Aurora MySQL 버전, AWS 인스턴스 클래스, AWS 리전, 가용 영역 등의 어떤 조합을 병렬 쿼리 클러스터에 사용할 수 있는지 확인합니다. 그렇게 하려면, JSON 형식으로 출력을 생성하는 aws rds describe-orderable-db-instance-options 명령을 사용합니다. 다음 코드 예제에서는 작업 방법을 보여줍니다.

```
# See choices for a specified AWS Region.  
aws rds describe-orderable-db-instance-options --engine aurora --query  
  'OrderableDBInstanceOptions[?contains(SupportedEngineModes,`parallelquery`)==`true`].  
  {Engine:Engine,EngineVersion:EngineVersion,SupportedEngineModes:SupportedEngineModes,DBInstanceClas  
  --region us-east-1
```

```
[  
  {  
    "DBInstanceClass": "db.r3.2xlarge",  
    "EngineVersion": "5.6.10a",  
    "Engine": "aurora",  
    "SupportedEngineModes": [  
      "parallelquery"  
    ]  
  },  
  {  
    "DBInstanceClass": "db.r3.4xlarge",  
    "EngineVersion": "5.6.10a",  
    ...  
}
```

2. MySQL 5.6 호환 데이터베이스의 스냅샷을 찾습니다.
3. [DB 클러스터 스냅샷에서 복원 \(p. 273\)](#)의 일반 AWS CLI 절차를 따르십시오.
4. --engine-mode 옵션의 경우 parallelquery를 지정합니다. 다음 코드 예제에서는 작업 방법을 보여줍니다.

```
aws rds restore-db-instance-from-db-snapshot \  
  --db-instance-identifier mynewdbinstance \  
  --db-snapshot-identifier mydbsnapshot \  
  --engine-mode parallelquery
```

5. 생성하거나 복원한 클러스터에서 병렬 쿼리 기능을 사용할 수 있는지 확인합니다.
`aurora_pq_supported` 구성 설정이 맞는지 확인합니다.

```
mysql> select @@aurora_pq_supported;  
+-----+  
| @@aurora_pq_supported |  
+-----+  
|          1 |  
+-----+
```

병렬 쿼리 활성화 및 비활성화

`aurora_pq` 옵션을 사용하여 DB 인스턴스에 대하여 전역 수준과 세션 수준에서 모두 동적으로 병렬 쿼리를 활성화 및 비활성화할 수 있습니다. 병렬 쿼리 기능을 사용할 수 있는 클러스터에서는 파라미터가 기본적으로 활성화됩니다.

```
mysql> select @@aurora_pq;  
+-----+  
| @@aurora_pq |  
+-----+  
|          1 |  
+-----+
```

세션 수준에서(예를 들어, `mysql` 명령줄을 통해 혹은 JDBC 또는 ODBC 애플리케이션 내에서) `aurora_pq` 파라미터를 전환하려면, 표준 메서드를 사용하여 클라이언트 구성 설정을 변경합니다. 예를 들어, 표준 MySQL 클라이언트에서는 명령어 `set session aurora_pq = {'ON'/'OFF'}`입니다. 또한 JDBC 구성 또는 애플리케이션 코드에 세션 수준 파라미터를 추가하여 동적으로 병렬 쿼리를 활성화하거나 비활성화할 수 있습니다.

현재, `aurora_pq` 파라미터에 대한 전역 설정을 변경할 때는 개별 DB 인스턴스에 대해서가 아니라 전체 클러스터에 대하여 이를 수행해야 합니다. 클러스터 수준에서 `aurora_pq` 파라미터를 전환하려면, [DB 파라미터 그룹](#)에 대한 전역 설정을 업데이트하는 절차를 따르십시오.

터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168)에 설명된 대로 파라미터 그룹으로 작업하기 위한 기술을 사용합니다. 다음 단계를 따르십시오.

1. 사용자 지정 클러스터 파라미터 그룹을 만듭니다.
2. `aurora_pq`를 원하는 값으로 업데이트합니다.
3. 사용자 지정 클러스터 파라미터 그룹을 병렬 쿼리 기능을 사용하려는 Aurora 클러스터에 연결합니다.
4. 클러스터의 모든 DB 인스턴스를 다시 시작합니다.

[ModifyDBClusterParameterGroup](#) API 작업 또는 AWS Management 콘솔을 사용하여 병렬 쿼리 파라미터를 수정할 수 있습니다.

Note

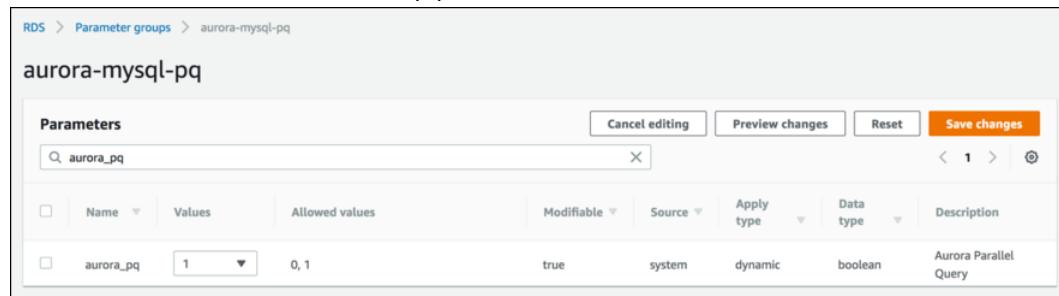
병렬 쿼리가 활성화되면 Aurora MySQL이 각 쿼리의 실행 시간에 병렬 쿼리를 사용할지 여부를 결정합니다. 조인, 통합, 하위 쿼리 등의 경우에는 Aurora MySQL이 각 쿼리 블록의 실행 시간에 병렬 쿼리를 사용할지 여부를 결정합니다. 자세한 내용은 [어떤 문이 병렬 쿼리를 사용하는지 확인](#) (p. 536) 및 [병렬 쿼리가 SQL 구조에서 작동하는 방법](#) (p. 540) 단원을 참조하십시오.

콘솔을 사용하여 DB 인스턴스에 대하여 병렬 쿼리 활성화 및 비활성화

파라미터 그룹으로 작업하여 DB 인스턴스 수준에서 병렬 쿼리를 활성화하거나 비활성화할 수 있습니다.

AWS Management 콘솔을 사용하여 Aurora MySQL 클러스터에 대하여 병렬 쿼리를 활성화하거나 비활성화하려면

1. [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업](#) (p. 168)에 설명된 대로 사용자 지정 파라미터 그룹을 생성합니다.
2. 아래 스크린샷과 같이 `aurora_pq`를 0(비활성화) 또는 1(활성화)로 업데이트합니다. 병렬 쿼리 기능을 사용할 수 있는 클러스터에서는 `aurora_pq`가 기본적으로 활성화됩니다.



3. 사용자 지정 클러스터 파라미터 그룹을 병렬 쿼리 기능을 사용하려는 Aurora DB 클러스터에 연결합니다.

CLI를 사용하여 DB 인스턴스에 대하여 병렬 쿼리 활성화 및 비활성화

`modify-db-cluster-parameter-group` 명령을 사용하여 병렬 쿼리 파라미터를 수정할 수 있습니다.

CLI를 사용하여 DB 인스턴스에 대하여 병렬 쿼리를 활성화하거나 비활성화하려면

- `modify-db-cluster-parameter-group` 명령을 사용하여 병렬 쿼리 파라미터를 수정합니다.

또한 세션 수준에서도(예를 들어, mysql 명령줄을 통해 혹은 JDBC 또는 ODBC 애플리케이션 내에서) 병렬 쿼리를 활성화하거나 비활성화할 수 있습니다. 그렇게 하려면, 표준 메서드를 사용하여 클라이언트 구성 설정을 변경합니다. 예를 들어, 표준 MySQL 클라이언트에서는 명령이 `set session aurora_pq = {'ON'/'OFF'}`입니다.

또한 JDBC 구성 또는 애플리케이션 코드에 세션 수준 파라미터를 추가하여 동적으로 병렬 쿼리를 활성화하거나 비활성화할 수 있습니다.

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.

- 어떤 쿼리가 병렬 쿼리를 사용하는지 모니터링합니다.
- 병렬 쿼리가 가장 데이터 집약적이고 오래 실행되는 쿼리에 사용되고 있는지 확인합니다.
- 병렬 쿼리를 활성화하여 예상되는 쿼리에 적용하고 워크로드에 맞는 적절한 수준의 동시성으로 실행되도록 클러스터에 대한 조건을 미세 조정합니다.

병렬 쿼리를 활용하기 위해 스키마 객체 생성

병렬 쿼리는 테이블이 `ROW_FORMAT=Compact` 설정을 사용해야 하기 때문에, Aurora 구성 설정에 `INNODB_FILE_FORMAT` 구성 옵션에 대한 변경 사항이 있는지 확인합니다. (`CREATE TABLE` 문의 대체 `ROW_FORMAT` 설정을 위해서는 `INNODB_FILE_FORMAT` 구성 옵션이 이미 'Barracuda'로 설정되어 있어야 합니다.) `SHOW TABLE STATUS` 문을 실행하여 데이터베이스의 모든 테이블에 대한 행 형식을 확인합니다.

현재 병렬 쿼리를 위해서는 테이블이 분할되어 있지 않아야 합니다. 따라서, `CREATE TABLE` 문 및 `SHOW CREATE TABLE` 출력을 확인하고 `PARTITION BY` 절을 제거합니다. 기존의 분할된 테이블에 대해서는, 먼저 동일한 열 정의 및 인덱스를 사용하여 데이터를 분할되지 않은 테이블에 복사합니다. 그런 다음, 분할되지 않은 테이블이 기존 쿼리 및 ETL 워크플로우에서 사용되도록 이전 테이블과 새 테이블의 이름을 변경합니다.

병렬 쿼리가 더 많은 테이블에서 작동할 수 있도록 스키마를 변경하기 전에, 병렬 쿼리로 인해 그러한 테이블의 성능이 정말로 증가되는지 확인하기 위한 테스트를 수행합니다. 또한, 병렬 쿼리를 위한 스키마 요구 사항이 원하는 목표와 다른 식으로 양립될 수 있는지 확인합니다.

예를 들어, `ROW_FORMAT=Compressed`에서 `ROW_FORMAT=Compact`으로 전환하기 전에 원본 테이블과 새 테이블에 대한 워크로드의 성능을 테스트합니다. 또한 데이터 볼륨 증가와 같은 다른 잠재적 영향도 고려합니다.

어떤 문이 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 사용할 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 버퍼풀에서 전부 사용될 수도 있습니다.

클러스터 성능을 모니터링하거나 튜닝할 때, 병렬 쿼리가 적절한 컨텍스트에서 사용되고 있는지 여부를 결정해야 합니다. 이 기능을 활용하기 위해 데이터베이스 스키마, 설정, SQL 쿼리 또는 심지어 클러스터 토플로지 및 애플리케이션 연결 설정을 조정할 수도 있습니다.

쿼리가 병렬 쿼리를 사용하고 있는지 확인하려면, [EXPLAIN](#) 문을 실행하여 쿼리 실행 계획("실행 계획"이라고도 함)을 확인합니다. SQL 문, 절 및 표현식이 병렬 쿼리의 EXPLAIN 출력에 어떠한 영향을 미치는지에 대한 예는 [병렬 쿼리가 SQL 구조에서 작동하는 방법 \(p. 540\)](#)를 참조하십시오.

다음 예는 기존의 실행 계획과 병렬 쿼리 계획 간에 차이점을 보여줍니다. 이 쿼리는 TPC-H 벤치마크의 쿼리 3입니다. 이 단원의 곳곳에 나오는 샘플 쿼리의 대부분이 TPC-H 데이터 세트의 테이블을 사용합니다.

```
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;
```

병렬 쿼리가 비활성화되어 있으면, 쿼리가 다음과 같은 실행 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 활성화된 후에는, 이 실행 계획의 두 단계가 EXPLAIN 출력의 Extra 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 [병렬 쿼리가 SQL 구조에서 작동하는 방법 \(p. 540\)](#)을 참조하십시오.

다음 예제 출력은 콜드 버퍼풀이 있는 db.r4.2xlarge 인스턴스에서 이전 쿼리를 실행한 결과를 보여줍니다. 병렬 쿼리 사용 시 쿼리가 상당히 더 빠르게 실행됩니다.

Note

타이밍은 많은 환경 요인에 의존하며 이 예제 쿼리는 초기 버전의 병렬 쿼리를 사용하여 실행되었기 때문에, 결과가 다를 수도 있습니다. 본인의 고유한 환경, 워크로드 등에서의 결과를 확인하기 위해 항상 본인의 성능 테스트를 수행하십시오.

```
-- Without parallel query
+-----+-----+-----+
| l_orderkey | revenue | o_orderdate | o_shipppriority |
+-----+-----+-----+
| 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_shipppriority |
+-----+-----+-----+
| 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 사용하십시오. 실제 워크로드의 경우, 동시성 한도와 같은 추가 요인이 적용됩니다.

병렬 쿼리 모니터링

Amazon Aurora DB 클러스터 지표 모니터링 (p. 346)에 설명된 Amazon CloudWatch 지표 외에도 Aurora는 다른 전역적 상태 변수를 제공합니다. 이러한 전역적 상태 변수를 사용하여 병렬 쿼리 실행을 모니터링하고 옵티마이저가 주어진 상황에서 병렬 쿼리를 사용할 수 있거나 사용할 수 없는 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 실행된 쿼리와 반드시 일대일 매핑되는 것은 아닙니다. 예를 들어, 실행 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. 그런 다음 병렬 쿼리가 예상되는 쿼리를 위해 실행되도록, 클러스터 설정, 쿼리 혼합, 병렬 쿼리가 활성화된 DB 인스턴스 등을 조정할 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
<code>Aurora_pq_request_attempted</code>	요청된 병렬 쿼리 세션의 수입니다. 이 값은 하위 쿼리 및 조인과 같은 SQL 구조에 따라, 쿼리 당 두 개 이상의 세션을 나타낼 수 있습니다.
<code>Aurora_pq_request_executed</code>	병렬 쿼리 세션의 수가 성공적으로 실행됩니다.
<code>Aurora_pq_request_failed</code>	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.
<code>Aurora_pq_pages_pushed_down</code>	병렬 쿼리가 헤드 노드로 네트워크 전송을 회피한 데이터 페이지의 수입니다(각각 16 KiB의 고정 크기).
<code>Aurora_pq_bytes_returned</code>	병렬 쿼리 동안 헤드 노드에 전송된 투플 데이터 구조를 위한 바이트 수입니다. <code>Aurora_pq_pages_pushed_down</code> 과 비교하기 위해 16,384로 나눕니다.
<code>Aurora_pq_request_not_chosen</code>	병렬 쿼리를 쿼리를 충족시키기 위해 선택되지 않은 횟수입니다. 이 값은 여러 개의 다른 더 세분화된 카운터의 합계입니다. 이 카운터는 쿼리가 실제로 수행되지 않는 경우에도 <code>EXPLAIN</code> 문에 의해 증가될 수 있습니다.
<code>Aurora_pq_request_not_chosen_below_min_rows</code>	테이블에 있는 행의 수로 인해 병렬 쿼리가 선택되지 않은 횟수입니다. 이 카운터는 쿼리가 실제로 수

	행되지 않는 경우에도 EXPLAIN 문에 의해 증가될 수 있습니다.
Aurora_pq_request_not_chosen_small_table	행의 수 및 평균 행 길이에 따라 결정된 테이블의 전체 크기로 인해 병렬 쿼리가 선택되지 않은 횟수입니다. 이 카운터는 쿼리가 실제로 수행되지 않는 경우에도 EXPLAIN 문에 의해 증가될 수 있습니다.
Aurora_pq_request_not_chosen_high_buffer_pool_pct	테이블 데이터 중 많은 양이(현재, 95% 이상) 이미 버퍼풀에 있었기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다. 이러한 경우, 옵티마이저는 버퍼풀에서 데이터 읽기가 더 효율적이라고 결정합니다. 이 카운터는 쿼리가 실제로 수행되지 않는 경우에도 EXPLAIN 문에 의해 증가될 수 있습니다.
Aurora_pq_request_not_chosen_few_pages_outside_buffer_pool	테이블 데이터 중 95% 미만이 버퍼풀에 있는 경우에도, 버퍼링되지 않은 테이블 데이터가 병렬 쿼리가 가치 있을 만큼 충분하지 않았기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다. 이 카운터는 쿼리가 실제로 수행되지 않는 경우에도 EXPLAIN 문에 의해 증가될 수 있습니다.
Aurora_pq_max_concurrent_requests	이 Aurora DB 인스턴스에서 동시에 실행될 수 있는 병렬 쿼리 세션의 최대 수입니다. 이 수는 AWS 인스턴스 클래스에 따라 결정되는 고정된 수입니다.
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_requests와 비교합니다.
Aurora_pq_request_throttled	동시 병렬 쿼리의 최대 수가 이미 특정 Aurora DB 인스턴스에서 실행 중이기 때문에 병렬 쿼리가 선택되지 않은 횟수입니다.
Aurora_pq_request_not_chosen_long trx	장기 실행 트랜잭션 내부에서 쿼리가 시작되는 종이어서, 비병렬 쿼리 실행 경로를 사용한 병렬 쿼리 요청의 수입니다. 이 카운터는 쿼리가 실제로 수행되지 않는 경우에도 EXPLAIN 문에 의해 증가될 수 있습니다.
Aurora_pq_request_not_chosen_unsupported_access	WHERE 절이 병렬 쿼리를 위한 기준에 부합되지 않기 때문에 비병렬 쿼리 실행 경로를 사용하는 병렬 쿼리 요청의 수입니다. 이 결과는 쿼리에 데이터 집약적인 스캔이 필요하지 않은 경우이거나 쿼리가 DELETE 또는 UPDATE 문인 경우에 발생할 수 있습니다.

병렬 쿼리가 SQL 구조에서 작동하는 방법

다음 섹션에서는 특정 SQL 문이 병렬 쿼리를 사용 또는 사용하지 않는 이유와 Aurora MySQL 기능이 병렬 쿼리와 상호 작용하는 방법에 대해 자세히 확인할 수 있습니다. 이러한 자세한 내용은 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.

주제

- EXPLAIN 문 (p. 541)
- WHERE 절 (p. 542)
- WHERE 절의 함수 호출 (p. 544)
- 집계 함수, GROUP BY 절 및 HAVING 절 (p. 545)
- 비교 연산자 (p. 545)
- 조인 (p. 546)
- 하위 쿼리 (p. 547)
- UNION (p. 547)
- 보기 (p. 548)
- 데이터 조작 언어(DML) 문 (p. 548)
- 트랜잭션 및 잠금 (p. 549)
- 인덱스 (p. 551)
- 내장된 캐싱 메커니즘 (p. 551)
- MyISAM 임시 테이블 (p. 551)

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 실행 계획에서 가장 중요한 항목입니다.

- key 열의 NULL 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 않아서 병렬 쿼리가 사용될 가능성이 낮음을 암시합니다.
- 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. 다음은 쿼리가 적은 수의 행만 스캔할 것이라는 기본 키의 특성을 기반으로, 옵티마이저가 추산할 수 있는 쿼리를 보여주는 예제입니다. 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 |
|     |             |       | 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 있습니다. 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 EXPLAIN 출력에서 extra 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 |
|   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) |
+-----+...+-----+
+
|
```

옵티マイ저가 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 |
+----+...+
```

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, LOWER()에 대한 호출이 WHERE 절에 나타나기 때문에 병렬 쿼리가 이 호출을 병렬화합니다. SUBSTR() 및 UPPER()에 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 WHERE 절의 CASE 표현식 및 LIKE 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) |
```

```
+----+...  
+  
+
```

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. 선택 목록 또는 HAVING 절에 있는 집계 함수 호출은 스토리지 계층으로 푸시 다운되지 않습니다. 하지만, 병렬 쿼리는 집계 함수에서도 그러한 쿼리의 성능을 여전히 향상 시킬 수 있습니다. 그렇게 하기 위해서 먼저 스토리지 계층에서 병렬적으로 원시 데이터 페이지로부터 열 값을 추출합니다. 그런 다음 이러한 값을 전체 데이터 페이지가 아니라 간소화된 티플 형식으로 다시 헤드 노드로 전송합니다. 항상 그렇듯이, 쿼리는 병렬 쿼리를 위해 최소 하나 이상의 WHERE 조건자가 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) |  
+----+...+  
+  
+
```

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 보여 줍니다. 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. 옵티マイ저는 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 실행됩니다. 따라서, 각 조인 쿼리는 동시성 한도 면에서 단일 병렬 쿼리 세션으로 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

해시 조인은 병렬 쿼리가 활성화된 클러스터에서 항상 사용할 수 있습니다. 병렬 쿼리는 해시 조인 최적화에서 이익을 얻는 리소스 집약적인 쿼리 유형에 일반적으로 사용됩니다.

```
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
|    |             | part        | ... | 20427936 | Using where; Using parallel query (2
|    |             |            |     |           | columns, 1 filters, 0 exprs; 0 extra)
|    |             | partsupp   | ... | 78164450 | Using where; Using join buffer (Block Nested
|    |             |            |     |           | Loop)
+-----+-----+...+-----+
```

하위 쿼리

외부 쿼리 블록 및 내부 하위 쿼리 블록은 각 블록에 대하여 테이블의 일반적인 특성, WHERE 절 등을 기반으로 각각 병렬 쿼리를 사용하거나 사용하지 않을 수 있습니다. 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)
+-----+-----+...+-----+
```

UNION

UNION 쿼리의 각 쿼리 블록은 UNION의 각 부분에 대하여 테이블의 일반적인 특성, WHERE 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 동시에 병렬 쿼리의 한도에 1로만 가산됩니다.

보기

옵티마이저는 기본 테이블을 사용하는 더 긴 쿼리로 보기로 쿼리를 재작성합니다. 따라서, 병렬 쿼리는 테이블 참조가 보기이든 실제 테이블이든 관계 없이 동일하게 작동합니다. 쿼리에 병렬 쿼리를 사용할지 여부 및 어떤 부분을 푸시 다운 할지에 대해 모두 동일한 고려 사항이 최종적으로 재작성된 쿼리에 적용됩니다.

예를 들어, 다음 실행 계획은 일반적으로 병렬 쿼리를 사용하지 않는 보기 정의를 보여줍니다. 보기의 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

SELECT 부분이 병렬 쿼리를 위한 다른 조건에 부합하는 경우, INSERT 문이 처리의 SELECT 단계에 병렬 쿼리를 사용할 수 있습니다.

```
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 버퍼풀에 있습니다. 그러므로, 테이블이 많은 수의 행을 삽입한 직후에는 병렬 쿼리에 적격하지 않을 수도 있습니다. 나중에 데이터가 정상 작동 중에 버퍼풀에서 제거된 후, 테이블에 대한 쿼리가 다시 병렬 쿼리를 사용하기 시작할 수 있습니다.

문의 SELECT 부분이 다른 식으로 병렬 쿼리에 적격할 수 있는 경우에도 CREATE TABLE AS 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 WHERE 절의 조건자에 관계 없이 DELETE 또는 UPDATE 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

병렬 쿼리는 REPEATABLE READ 격리 수준에서 실행된 문에만 적용됩니다. 이 격리 수준은 Aurora 읽기 DB 인스턴스의 기본값입니다. Aurora 기본 인스턴스에서 모든 격리 수준을 사용할 수 있습니다. Aurora 격리 수준에 대한 자세한 내용은 [Aurora MySQL 격리 수준 \(p. 679\)](#) 단원을 참조하십시오.

대형 트랜잭션이 완료된 후에는 테이블 통계가 기한 경과될 수 있습니다. Aurora가 행의 수를 정확하게 추산 할 수 있으려면 먼저 그런 기한 경과 통계에 ANALYZE TABLE 문이 필요할 수 있습니다. 대규모 DML 문 또한 테이블 데이터의 상당 부분을 버퍼풀로 가져올 수 있습니다. 이러한 데이터가 버퍼풀에 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 시작 되기 전에 쿼리가 최대 간격(현재 10분)보다 더 오래 실행되는 경우 이러한 유형의 시간 초과가 발생할 수 있습니다.

임시(1회) 쿼리를 수행하는 mysql 세션에 autocommit=1을 설정하여 실수로 장기 실행 트랜잭션을 시작 할 가능성을 줄일 수 있습니다. 읽기 보기 를 만들면 테이블에 대한 SELECT 문도 트랜잭션을 시작합니다. 읽기 보기 를 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 보기 를 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_txn 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_txn 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_txn 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_not_chosen_long_trx`를 확인합니다.

```
mysql> show global status like '%pq%trx%';
+-----+-----+
| Variable_name          | Value |
+-----+-----+
| Aurora_pq_not_chosen_long_trx | 4     |
+-----+-----+
```

`SELECT FOR UPDATE` 또는 `SELECT LOCK IN SHARE MODE` 구문과 같은 잠금을 획득하는 `SELECT` 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 |
+-----+-----+
```

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 TABLE`을 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.

내장된 캐싱 메커니즘

Aurora에는 내장된 캐싱 메커니즘(즉, 버퍼풀 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 버퍼풀에 페이지가 추가되거나 버퍼풀에 이미 존재하는 페이지가 제거되지 않습니다.

Aurora는 버퍼풀에 존재하는 테이블 데이터의 페이지 수와 그 수가 나타내는 테이블 데이터의 비율을 확인합니다. Aurora는 이 정보를 이용하여 병렬 쿼리를 사용하는(버퍼풀의 데이터를 사용하지 않음) 것이 더 효율적인지 여부를 결정합니다. 또는 Aurora는 버퍼풀에 캐시된 데이터를 사용하는 비병렬 쿼리 실행 경로를 사용할 수도 있습니다. 어떤 페이지가 캐시되고 데이터 집약적인 쿼리가 캐싱 및 제거에 어떠한 영향을 미치는지는 버퍼풀에 관련된 구성 설정에 따라 다릅니다. 따라서 선택은 버퍼풀 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 버퍼풀에 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 버퍼풀에 없어서 오래 걸릴 수 있습니다. 그런 다음 두 번째 및 이후 쿼리는 버퍼풀이 이제 "워밍업"되어서 훨씬 더 빨라집니다. 병렬 쿼리는 일반적으로 테이블에 대한 맨 첫 번째 쿼리로부터 일관된 성능을 보여줍니다. 성능 테스트를 수행할 때는 콜드(cold) 버퍼풀과 웜(warm) 버퍼풀에서 모두 비병렬 쿼리를 벤치마크합니다. 일부 경우, 웜 버퍼풀에서의 결과는 병렬 쿼리 시간과 잘 비교될 수 있습니다. 이러한 경우에는, 해당 테이블에 대한 쿼리의 빈도 및 해당 테이블의 데이터를 버퍼풀에 유지할 만한 가치가 있는지 여부와 같은 요소를 고려하십시오.

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 쿼리에서 활성화되거나 활성화되지 않는 쿼리의 버전들 간에 쿼리 시간이 비교 가능해지도록, 이 주제의 많은 병렬 쿼리 예제에 이 힌트가 포함되어 있습니다. 병렬 쿼리의 프로덕션용으로 이동할 때는 원본에서 이 힌트를 제거해야 합니다.

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 `EXPLAIN` 출력에서 `Using temporary`로 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 검토할 수 있습니다.

Note

CloudWatch Logs의 로그 그룹에 Aurora MySQL 일반 데이터, 느린 데이터, 감사 데이터 및 오류 로그 데이터를 게시할 수 있습니다. 자세한 정보는 [Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 \(p. 639\)](#) 단원을 참조하십시오.

고급 감사 활성화

이 단원에서 설명하는 파라미터를 사용하여 DB 클러스터에 대해 고급 감사를 활성화하고 구성할 수 있습니다.

고급 감사를 활성화 또는 비활성화하려면 `server_audit_logging` 파라미터를 사용하고 기록할 이벤트를 지정하려면 `server_audit_events` 파라미터를 사용합니다.

감사를 받을 사용자를 지정하려면 `server_audit_excl_users` 및 `server_audit_incl_users` 파라미터를 사용합니다. `server_audit_excl_users` 및 `server_audit_incl_users`가 비어 있을 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 해당 사용자만 제외하고 다른 모든 사용자가 감사됩니다. `server_audit_excl_users` 및 `server_audit_incl_users` 모두에 사용자를 추가할 경우 `server_audit_incl_users`에 더 높은 우선 순위가 부여되므로 해당 사용자가 감사됩니다.

DB 클러스터가 사용하는 파라미터 그룹에서 다음 파라미터를 설정하여 고급 감사를 구성합니다. [DB 파라미터 그룹의 파라미터 수정 \(p. 174\)](#) 단원에서 설명하는 절차에 따라 AWS Management 콘솔을 사용하여 DB 클러스터 파라미터를 수정할 수 있습니다. `modify-db-cluster-parameter-group` AWS CLI 명령 또는 `ModifyDBClusterParameterGroup` Amazon RDS API 명령을 사용하여 DB 클러스터 파라미터를 프로그래밍 방식으로 수정할 수 있습니다.

이들 파라미터를 수정할 때 DB 클러스터 재시작은 필요하지 않습니다.

server_audit_logging

고급 감사를 활성화 또는 비활성화합니다. 이 파라미터는 OFF가 기본값입니다. 고급 감사를 활성화하려면 ON으로 설정합니다.

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).

- 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.

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. 지정된 사용자 이름이 mysql.user 테이블 내 User 열의 해당 이름과 일치해야 합니다. 사용자 이름에 대한 자세한 정보는 [MySQL 설명서](#)를 참조하십시오.

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.

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. 지정된 사용자 이름이 mysql.user 테이블 내 User 열의 해당 이름과 일치해야 합니다. 사용자 이름에 대한 자세한 정보는 [MySQL 설명서](#)를 참조하십시오.

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.

감사 로그 보기

콘솔을 사용하여 감사 로그를 확인하고 다운로드할 수 있습니다. Databases(데이터베이스) 페이지에서 DB 인스턴스를 선택하여 세부 정보를 표시한 다음, 로그 섹션으로 스크롤합니다.

Logs (28)		
Filter name		
Name	Last written	Size
error/mysql-error-running.log	Fri Jan 12 15:00:00 GMT-800 2018	18.8 kB
error/mysql-error-running.log.2018-01-11.22	Thu Jan 11 14:00:00 GMT-800 2018	96.7 kB
error/mysql-error-running.log.2018-01-11.23	Thu Jan 11 14:30:00 GMT-800 2018	19.4 kB
error/mysql-error-running.log.2018-01-12.00	Thu Jan 11 15:30:00 GMT-800 2018	38 kB
error/mysql-error-running.log.2018-01-12.01	Thu Jan 11 16:30:00 GMT-800 2018	38.2 kB

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 정보는 [데이터베이스 로그 파일 보기 및 나열 \(p. 449\)](#) 및 [데이터베이스 로그 파일 다운로드 \(p. 450\)](#) 단원을 참조하십시오.

감사 로그 세부 정보

로그 파일은 UTF-8 형식입니다. 로그는 여러 파일로 작성되며, 파일 수는 인스턴스 크기에 따라 달라집니다. 최신 이벤트를 보려면 모든 감사 로드 파일을 확인해야 할 수 있습니다.

로그 항목이 순서대로 나열되지 않습니다. 정렬을 위해 타임스탬프 값을 사용할 수 있습니다.

로그 파일은 총 100MB에 도달하면 교체됩니다. 이 제한은 구성할 수 없습니다.

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.
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.
object	QUERY 이벤트의 경우 이 값은 실행된 쿼리를 나타냅니다. TABLE 이벤트의 경우 이 값은 테이블 이름을 나타냅니다.
retcode	기록된 작업의 반환 코드입니다.

Amazon Aurora MySQL를 사용하는 단일 마스터 복제

Aurora MySQL 복제 기능은 클러스터의 고가용성과 성능을 위한 핵심 기능입니다. Aurora는 최대 15개의 Aurora 복제본을 사용하여 클러스터를 손쉽게 생성하거나 크기를 조정할 수 있도록 지원합니다.

모든 복제본은 동일한 기반 데이터에서 작동합니다. 일부 데이터베이스 인스턴스가 오프라인 상태로 될 경우 다른 데이터베이스 인스턴스를 사용하여 계속 쿼리를 처리하거나, 필요한 경우 Writer를 인계받을 수 있습니다. Aurora는 읽기 전용 연결을 여러 데이터베이스에 자동으로 분산시켜 Aurora 클러스터가 쿼리 집약적인 워크로드를 지원하도록 합니다.

다음은 Aurora MySQL 복제의 작동 방식과 최적의 가용성 및 성능을 위해 복제 설정을 미세 조정하는 방법에 대한 정보입니다.

Note

다음과 같이 단일 마스터 복제를 사용하는 Aurora 클러스터 복제 기능에 대해 자세히 배울 수 있습니다. 이러한 유형의 클러스터는 Aurora의 기본값입니다. Aurora 멀티 마스터 클러스터에 대한 설명은 [Aurora 멀티 마스터 클러스터 작업 \(p. 584\)](#) 단원을 참조하십시오.

주제

- [Aurora 복제본 사용 \(p. 555\)](#)
- [Amazon Aurora MySQL의 복제 옵션 \(p. 555\)](#)
- [Amazon Aurora MySQL 복제를 위한 성능 고려 사항 \(p. 556\)](#)
- [Amazon Aurora MySQL 복제를 위한 고가용성 고려 사항 \(p. 556\)](#)
- [Amazon Aurora MySQL 복제 모니터링 \(p. 557\)](#)
- [여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제 \(p. 557\)](#)
- [Aurora와 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터 간의 복제 \(p. 567\)](#)
- [Aurora MySQL \(p. 580\)](#)

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 \(p. 55\)](#) 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 대조적으로 MySQL 읽기 전용 복제본은 마스터 DB 인스턴스부터 로컬 데이터 스토어에 이르는 모든 쓰기 작업을 단일 스레드에서 재 실행해야 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 READ를 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

종국(닝샤) 리전은 리전 간 읽기 전용 복제본을 지원하지 않습니다.

Amazon Aurora MySQL의 복제 옵션

다음 옵션들 중에서 복제를 설정할 수 있습니다.

- 다른 AWS 리전에 속하는 Aurora MySQL DB 클러스터의 Aurora 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 \(p. 557\)](#) 단원을 참조하십시오.

- MySQL 이진 로그(binlog) 복제를 사용하여 동일한 AWS 리전에 Aurora MySQL DB 클러스터 2개를 설정합니다.

자세한 정보는 [Aurora와 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터 간의 복제 \(p. 567\)](#) 단원을 참조하십시오.

- Amazon RDS MySQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하여 마스터인 Aurora MySQL DB 인스턴스와 Amazon RDS DB 클러스터를 설정합니다.

일반적으로 이 방법은 진행 중인 복제보다는 Aurora MySQL로의 마이그레이션에 사용됩니다. 자세한 정보는 [DB 스냅샷을 사용하여 MySQL DB 인스턴스에서 Amazon Aurora MySQL DB 클러스터로 데이터 마이그레이션 \(p. 492\)](#) 단원을 참조하십시오.

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

Aurora MySQL 1.17.4부터 다음 기능을 사용하여 Aurora MySQL 복제의 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. 따라서 CPU 용량이 많은 8xlarge 및 16xlarge 인스턴스 클래스에서만 기능을 사용할 수 있습니다. 이러한 인스턴스 클래스에서 기본적으로 활성화됩니다. `aurora_enable_replica_log_compression` 파라미터를 비활성화하여 이 기능을 제어할 수 있습니다. 예를 들어 라이터(writer) 노드가 최대 CPU 용량에 거의 도달한 경우 큰 인스턴스 클래스에 대해 복제본 로그 압축을 비활성화할 수 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_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.

Amazon Aurora MySQL 복제를 위한 고가용성 고려 사항

클러스터에 Aurora 복제본이 많을수록 높은 가용성을 보장할 수 있습니다. 일부 데이터베이스 인스턴스를 사용할 수 없는 경우에도 전체 데이터 복사본이 있는 데이터베이스 인스턴스를 항상 쿼리에 사용할 수 있습니다.

여러 개의 Aurora 복제본이 있을 경우, 기본 데이터베이스 인스턴스가 재시작되는 짧은 기간 동안 복제본을 사용할 수 없게 됩니다. 이러한 재시작은 유지 관리 작업 중에 또는 복제본이 마스터에 너무 시간이 지연될 때 발생할 수 있습니다. 복제본을 다시 시작하면 해당 데이터베이스 인스턴스에 대한 기존 연결이 중단됩니다. Aurora 클러스터를 다시 시작하면 모든 복제본이 동시에 사용 불가 상태가 됩니다.

Aurora MySQL 1.17.4부터는 다음 기능을 사용하여, 복제본이 다시 시작될 때도 고가용성을 보장할 수 있습니다.

가동 중지 없이 재시작(ZDR) 기능은 예를 들어 복제본이 마스터에 비해 너무 늦어지는 경우 Aurora MySQL 복제본이 다시 시작될 때 기존 연결을 유지합니다. 진행 중인 트랜잭션은 룰백되고, 애플리케이션에서 트랜

잭션을 재시도해야 합니다. 이 기능을 활성화하려면 클러스터 파라미터 그룹에서 `aurora_enable_zdr` 파라미터를 활성화하십시오. 이 파라미터는 기본적으로 비활성화됩니다.

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 Aurora MySQL DB 클러스터에서 기본 인스턴스의 `mysql.ro_replica_status` 테이블을 쿼리하여 `Replica_lag_in msec` 열의 값을 확인할 수 있습니다. 이 열 값은 Amazon CloudWatch에 AuroraReplicaLag 지표 값으로 제공됩니다. Aurora 복제본 지연은 Aurora MySQL DB 클러스터의 `INFORMATION_SCHEMA.REPLICA_HOST_STATUS` 테이블에 있는 각 Aurora 복제본에도 기록됩니다.

RDS 인스턴스 및 CloudWatch 지표 모니터링에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 모니터링 \(p. 327\)](#) 단원을 참조하십시오.

여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제

Amazon Aurora MySQL DB 클러스터를 원본 DB 클러스터와 다른 AWS 리전의 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 AWS 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. 다른 AWS 리전에 Aurora MySQL DB 클러스터의 읽기 전용 복제본을 만들 때는 다음 사항을 유의해야 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.
- 리전 간 시나리오에서는 AWS 리전 간 네트워크 채널이 더 길어지기 때문에 원본 DB 클러스터와 읽기 전용 복제본 간의 지연 시간이 증가합니다.
- 리전 간 복제를 위해 데이터를 전송할 때는 Amazon RDS 데이터 전송 요금이 발생합니다. 다음과 같은 리전 간 복제 작업에서 요금이 발생하는 이유는 소스 AWS 리전을 벗어나 데이터를 전송하기 때문입니다.
 - 읽기 전용 복제본을 생성할 때는 Amazon RDS가 원본 클러스터의 스냅샷을 캡처하여 해당 스냅샷을 읽기 전용 복제본이 있는 AWS 리전으로 전송합니다.
- 원본 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 요금](#)을 참조하십시오.

- 동일한 원본 DB 클러스터를 참조하는 읽기 전용 복제본에 대해 동시 생성 또는 삭제 작업을 여러 개 실행 할 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 전에 \(p. 558\)](#)
- [리전 간 읽기 전용 복제본인 Amazon Aurora MySQL DB 클러스터 만들기 \(p. 558\)](#)
- [Amazon Aurora MySQL 리전 간 복제본 보기 \(p. 564\)](#)
- [읽기 전용 복제본을 DB 클러스터로 승격 \(p. 565\)](#)
- [Amazon Aurora MySQL 리전 간 복제본 문제 해결 \(p. 566\)](#)

시작하기 전에

리전 간 읽기 전용 복제본인 Aurora MySQL DB 클러스터를 만들기 전에 먼저 원본 Aurora MySQL DB 클러스터에 대한 이진 로깅을 활성화해야 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 활성화하려면 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.

자세한 정보는 [Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 \(p. 170\)](#) 및 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

리전 간 읽기 전용 복제본인 Amazon Aurora MySQL DB 클러스터 만들기

AWS Management 콘솔, AWS Command Line Interface(AWS CLI) 또는 Amazon RDS API를 사용하여 리전 간 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 콘솔을 사용하여 Aurora MySQL용 리전 간 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 콘솔을 사용하여 리전 간 읽기 전용 복제본인 Aurora MySQL DB 클러스터를 만들려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. AWS Management 콘솔의 오른쪽 상단 모서리에서 원본 DB 클러스터를 호스팅하는 AWS 리전을 선택합니다.

3. 탐색 창에서 인스턴스를 선택합니다.
4. 리전 간 읽기 전용 복제본을 만들려는 DB 인스턴스의 확인란을 선택합니다. 작업에서 리전 간 읽기 전용 복제본 만들기를 선택합니다.
5. 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.
[Publicly accessible]	리전 간 읽기 전용 복제본 DB 클러스터에 퍼블릭 IP 주소를 할당하려면 예를 선택하고, 그렇지 않으면 아니요를 선택합니다.
암호화	이 DB 클러스터에 대해 비활성화되어 있는 암호화를 활성화하려면 [Enable Encryption]을 선택합니다. 자세한 정보는 Amazon Aurora 리소스 암호화 (p. 945) 단원을 참조하십시오.
마스터 키	[Encryption]을 [Enable Encryption]으로 설정한 경우에만 사용할 수 있습니다. 현재 DB 클러스터를 암호화하는 데 사용할 마스터 키를 선택합니다. 자세한 정보는 Amazon Aurora 리소스 암호화 (p. 945) 단원을 참조하십시오.
DB 인스턴스 클래스	DB 클러스터의 기본 인스턴스에 대한 처리 및 메모리 요구를 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스 옵션에 대한 자세한 정보는 DB 인스턴스 클래스 (p. 30) 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 (p. 3) 단원을 참조하십시오.
[Read replica source]	리전 간 읽기 전용 복제본을 만들 원본 DB 클러스터를 선택합니다.
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. <ul style="list-style-type: none">• 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다.• 첫 번째 문자는 글자이어야 합니다.• 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.• 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 마

옵션	설명
	스터 사용자 이름과 마스터 암호는 원본 DB 클러스터의 마스터 사용자 이름 및 마스터 암호와 동일합니다.
[DB cluster identifier]	<p>리전 간 일기 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 정보는 Amazon Aurora 연결 관리 (p. 9) 단원을 참조하십시오.</p> <p>DB 클러스터 식별자는 다음과 같은 제약 조건이 있습니다.</p> <ul style="list-style-type: none"> 1~63자의 영숫자 문자 또는 하이픈으로 구성되어야 합니다. 첫 번째 문자는 글자이어야 합니다. 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다. AWS 리전별로 각 AWS 계정의 모든 DB 클러스터에 대해 고유해야 합니다.
Priority	새 DB 클러스터의 기본 인스턴스에 대한 장애 조치의 우선 순위를 선택합니다. 기본 인스턴스 장애로부터 복원할 때 이 우선 순위에 따라 승격할 Aurora Replicas 순서가 결정됩니다. 값을 선택하지 않을 경우 기본값은 tier-1입니다. 자세한 정보는 Aurora DB 클러스터의 내결함성 (p. 268) 단원을 참조하십시오.
데이터베이스 포트	애플리케이션과 유ти리티가 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.
확장 모니터링	DB 클러스터가 실행되는 운영 체제에 대한 실시간 수집을 활성화하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 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-role라는 이름의 역할을 생성하도록 기본값을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 단원을 참조하십시오.
Granularity	[Enhanced Monitoring]을 [Enable enhanced monitoring]으로 설정한 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
Auto minor version upgrade	<p>이 설정은 Aurora MySQL DB 클러스터에 적용되지 않습니다.</p> <p>Aurora MySQL의 엔진 업데이트에 대한 자세한 정보는 Amazon Aurora MySQL 데이터베이스 엔진 업데이트 (p. 686) 단원을 참조하십시오.</p>

6. 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-identifier로 식별되는 DB 클러스터가 암호화되는 리전 간 복제의 경우 --kms-key-id 옵션 및 --storage-encrypted 옵션을 지정해야 합니다. --source-region 또는 --pre-signed-url 옵션도 지정해야 합니다. --source-region을 사용하면 복제할 암호화된 DB 클러스터가 포함된 소스 AWS 리전에서 실행할 수 있는 CreateDBCluster API 작업에 대한 유효한 요청인 미리 서명된 URL이 자동 생성됩니다. --pre-signed-url을 사용하려면 대신 미리 서명된 URL을 수동으로 생성해야 합니다. AWS KMS 키 ID는 읽기 전용 복제본을 암호화하는 데 사용되며, 대상 AWS 리전에 대해 유효한 KMS 암호화 키여야 합니다. 이러한 옵션에 대해 자세히 알아보려면 [create-db-cluster](#) 단원을 참조하십시오.

Note

--storage-encrypted를 지정하고 --kms-key-id의 값을 제공하여 암호화되지 않은 DB 클러스터에서 암호화된 읽기 전용 복제본으로 리전 간 복제를 설정할 수 있습니다. 이 경우 --source-region 또는 --pre-signed-url을 지정하지 않아도 됩니다.

--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.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster \
--db-cluster-identifier sample-replica-cluster \
--engine aurora \
--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 ^
--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.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster \
--db-cluster-identifier sample-replica-cluster \
--engine aurora \
```

```
--replication-source-identifier arn:aws:rds:us-west-2:123456789012:cluster:sample-  
master-cluster \  
--kms-key-id my-us-east-1-key \  
--source-region us-west-2 \  
--storage-encrypted
```

Windows의 경우:

```
aws rds create-db-cluster ^  
--db-cluster-identifier sample-replica-cluster ^  
--engine aurora ^  
--replication-source-identifier arn:aws:rds:us-west-2:123456789012:cluster:sample-  
master-cluster ^  
--kms-key-id my-us-east-1-key ^  
--source-region us-west-2 ^  
--storage-encrypted
```

2. 다음 예와 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 available로 표시되면 DB 클러스터의 기본 인스턴스를 만들어 복제를 시작합니다. 이렇게 하려면 다음 예제에서와 같이 AWS CLI [create-db-instance](#) 명령을 사용하십시오.

Linux, OS X, Unix의 경우:

```
aws rds create-db-instance \  
--db-cluster-identifier sample-replica-cluster \  
--db-instance-class db.r3.large \  
--db-instance-identifier sample-replica-instance \  
--engine aurora
```

Windows의 경우:

```
aws rds create-db-instance ^  
--db-cluster-identifier sample-replica-cluster ^  
--db-instance-class db.r3.large ^  
--db-instance-identifier sample-replica-instance ^  
--engine aurora
```

DB 인스턴스가 생성되어 사용할 수 있게 되면 복제가 시작됩니다. AWS CLI [describe-db-instances](#) 명령을 호출하여 DB 인스턴스를 사용할 수 있는지 여부를 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.

ReplicationSourceIdentifier로 식별되는 DB 클러스터가 암호화되는 리전 간 복제의 경우 KmsKeyId 파라미터를 지정하고 StorageEncrypted 파라미터를 true로 설정해야 합니다. PreSignedUrl 파라미터도 지정해야 합니다. 미리 서명된 URL은 복제할 암호화된 DB 클러스터가 포함된 소스 AWS 리전에서 실행할 수 있는 CreateDBCluster API 작업에 대한 유효한 요청이어야 합니다. KMS 키 ID는 읽기 전용 복제본을 암호화하는데 사용되며, 대상 AWS 리전에 대해 유효한 KMS 암호화 키여야 합니다. 미리 서명된 URL을 수동이 아닌 자동으로 생성하려면 AWS CLI [create-db-cluster](#) 명령을 --source-region 옵션과 함께 사용합니다.

Note

StorageEncrypted를 true로 지정하고 KmsKeyId의 값을 제공하여 암호화되지 않은 DB 클러스터에서 암호화된 읽기 전용 복제본으로 리전 간 복제를 설정할 수 있습니다. 이 경우 PreSignedUrl을 지정하지 않아도 됩니다.

MasterUsername 및 MasterUserPassword 파라미터 값은 원본 DB 클러스터에서 가져오므로 포함 시키지 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  
&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=a04c831a0b54b5e4cd236a90dc9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

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  
%253FA%253DCreateDBCluster  
%2526DestinationRegion%253Dus-east-1  
%2526KmsKeyId%253Dmy-us-east-1-key  
%2526ReplicationSourceIdentifier%253Darn%25253Aaws%25253Ards%25253Aus-  
west-2%25253A123456789012%25253Accluster%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
&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=a04c831a0b54b5e4cd236a90dc9f5fab7185eb3b72b5ebe9a70a4e95790c8b7
```

2. 다음 예와 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 기본 인스턴스를 만들어 복제를 시작합니다. 그러려면 다음 예와 같이 RDS API [CreateDBInstance](#) 작업을 사용하십시오.

```
https://rds.us-east-1.amazonaws.com/
?Action/CreateDBInstance
&DBClusterIdentifier=sample-replica-cluster
&DBInstanceClass=db.r3.large
&DBInstanceIdentifier=sample-replica-instance
&Engine=aurora
&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 있는지 여부를 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 리전 간 복제 관계를 확인할 수 있습니다. 응답에서 리전 간 읽기 전용 복제본 DB 클러스터의 DB 클러스터 식별자는 `ReadReplicaIdentifiers` 필드를 참조하고, 복제 마스터인 원본 DB 클러스터의 ARN은 `ReplicationSourceIdentifier` 요소를 참조하십시오.

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 오래 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 0에 도달하면 읽기 전용 복제본이 원본 DB 인스턴스를 따라잡은 것입니다.
2. Amazon RDS 콘솔의 Promote(승격) 옵션, AWS CLI 명령 `promote-read-replica-db-cluster` 또는 `PromoteReadReplicaDBCluster` Amazon RDS API 작업 등을 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 콘솔에 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 Replica]를 선택합니다.

AWS CLI

읽기 전용 복제본을 DB 클러스터로 승격하려면 AWS CLI `promote-read-replica-db-cluster` 명령을 사용합니다.

Example

Linux, OS X, Unix의 경우:

```
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 `PromoteReadReplicaDBCluster`를 호출합니다.

Amazon Aurora MySQL 리전 간 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 활성화합니다. 자세한 정보는 [시작하기 전에 \(p. 558\)](#) 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

AWS 리전에서는 원본 DB 클러스터당 읽기 전용 복제본인 리전 간 DB 클러스터를 하나만 만들 수 있습니다. 특정 AWS 리전에 읽기 전용 복제본인 리전 간 DB 클러스터를 새로 만들려면 기존 클러스터를 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.

Aurora와 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터 간의 복제

Amazon Aurora MySQL는 MySQL과 호환되기 때문에 MySQL 데이터베이스와 Amazon Aurora MySQL DB 클러스터 사이에 복제를 설정할 수 있습니다. 이때 MySQL 데이터베이스에서 실행되는 MySQL 버전은 5.5 이후를 사용하는 것이 좋습니다. Aurora MySQL DB 클러스터가 복제 마스터 또는 복제본인 경우에 복제를 설정할 수 있고, Amazon RDS MySQL DB 인스턴스, Amazon RDS 외부의 MySQL 데이터베이스 또는 다른 Aurora MySQL DB 클러스터로 복제할 수 있습니다.

다른 AWS 리전의 Amazon RDS MySQL DB 인스턴스 또는 Aurora MySQL DB 클러스터로도 복제할 수 있습니다. AWS 리전 간 복제를 수행할 경우 DB 클러스터와 DB 인스턴스에 공개 액세스가 가능한지 확인합니다. Aurora MySQL DB 클러스터는 VPC의 퍼블릭 서브넷에 포함되어야 합니다.

Aurora MySQL DB 클러스터와 다른 리전에 있는 Aurora MySQL DB 클러스터 간에 복제를 구성하려면 원본 DB 클러스터와 다른 AWS 리전에 Aurora MySQL DB 클러스터를 읽기 전용 복제본으로 생성합니다. 자세한 정보는 [여러 AWS 리전에 걸쳐 Amazon Aurora MySQL DB 클러스터 복제 \(p. 557\)](#) 단원을 참조하십시오.

Aurora MySQL 2.04 이상에서는 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 [Aurora MySQL \(p. 580\)](#) 단원을 참조하십시오.

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;
```

Aurora MySQL을 사용한 MySQL 복제 설정 단계는 다음과 같으며, 이 주제의 뒷부분에 자세히 설명되어 있습니다.

- 복제 마스터에 대한 이진 로깅 활성화 (p. 567)
- 더 이상 필요 없을 때까지 이진 로그를 복제 마스터에 보관 (p. 572)
- 복제 마스터의 스냅샷 만들기 (p. 573)
- 복제본 대상으로 스냅샷 로드 (p. 575)
- 복제본 대상에 대해 복제 활성화 (p. 576)
- 복제본 모니터링 (p. 578)

MySQL 또는 다른 Aurora DB 클러스터를 사용한 복제 설정

MySQL로 Aurora 복제를 설정하려면 다음 단계를 수행합니다.

1. 복제 마스터에 대한 이진 로깅 활성화

다음 데이터베이스 엔진의 복제 마스터에 대한 이진 로깅을 활성화하는 방법의 지침을 확인하십시오.

데이터베이스 엔진	지침
Aurora	Aurora MySQL DB 클러스터에 대한 이진 로깅을 활성화하려면

데이터베이스 엔진	지침
	<p><code>binlog_format</code> 파라미터를 ROW, STATEMENT, MIXED로 설정합니다. 특정 binlog 형식이 필요하지 않는 한 MIXED로 설정하는 것이 좋습니다. <code>binlog_format</code> 파라미터는 기본 클러스터 파라미터 그룹에 속하는 클러스터 수준 파라미터입니다. <code>binlog_format</code> 파라미터를 OFF에서 다른 값으로 변경하는 경우, Aurora DB 클러스터를 재부팅해야 변경 사항이 적용됩니다.</p> <p>자세한 정보는 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 (p. 170) 및 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.</p>
RDS MySQL	<p>Amazon RDS DB 인스턴스에 대한 이진 로깅을 활성화하려면</p> <p>Amazon RDS DB 인스턴스에 대한 이진 로깅을 직접 활성화할 수는 없지만, 다음 중 하나를 수행하여 활성화할 수 있습니다.</p> <ul style="list-style-type: none">• DB 인스턴스의 자동 백업을 활성화합니다. DB 인스턴스를 생성할 때 자동 백업을 활성화하거나 기존 DB 인스턴스를 수정하여 백업을 활성화할 수 있습니다. 자세한 내용은 Amazon RDS 사용 설명서의 DB 인스턴스 생성 단원을 참조하십시오.• DB 인스턴스의 읽기 전용 복제본을 생성합니다. 자세한 내용은 Amazon RDS 사용 설명서의 읽기 전용 복제본 작업을 참조하십시오.

데이터베이스 엔진	지침
MySQL(외부)	<p>암호화된 복제 설정</p> <p>Aurora MySQL 버전 5.6을 사용하여 데이터를 안전하게 복제하려면 암호화된 복제를 사용하십시오.</p> <p>현재 외부 MySQL 데이터베이스로 암호화된 복제는 Aurora MySQL 버전 5.6에만 지원됩니다.</p> <p>Note</p> <p>암호화된 복제를 사용할 필요가 없다면 이 단계를 건너뛸 수 있습니다.</p> <p>다음은 암호화된 복제를 사용하기 위한 사전 조건입니다.</p> <ul style="list-style-type: none"> 외부 MySQL 마스터 데이터베이스에서 SSL(Secure Socket Layer)를 활성화 시켜야 합니다. Aurora MySQL DB 클러스터에 대해 클라이언트 키와 클라이언트 인증서를 준비해야 합니다. <p>암호화 복제 중, Aurora MySQL DB 클러스터는 MySQL 데이터베이스 서버의 클라이언트 역할을 합니다. Aurora MySQL 클라이언트 인증서와 키는 .pem 형식의 파일입니다.</p> <ol style="list-style-type: none"> 암호화 복제를 준비해야 합니다. <ul style="list-style-type: none"> 외부 MySQL 마스터 데이터베이스에서 SSL을 활성화 시키지 않았고 클라이언트 키와 인증서가 준비되지 않았다면, MySQL 데이터베이스 서버의 SSL을 활성화하고, 필요한 클라이언트 키와 인증서를 생성합니다. 외부 마스터에 SSL이 활성화되어 있다면, 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 마스터 데이터베이스의 SSL을 구성할 때 사용하는 인증 기관(CA) 키가 있어야 합니다. <p>자세한 정보는 MySQL 문서의 Creating SSL Certificates and Keys Using openssl을 참조하십시오.</p> <p>인증 기관(CA) 인증서, 클라이언트 키, 클라이언트 인증서가 필요합니다.</p> <ol style="list-style-type: none"> SSL을 사용하여 마스터 사용자로 Aurora MySQL DB 클러스터에 연결하십시오. <p>SSL을 이용한 Aurora MySQL DB 클러스터 연결에 대한 자세한 정보는 Aurora MySQL DB 클러스터에 SSL 사용 (p. 467)을 참조하십시오.</p> <ol style="list-style-type: none"> <code>mysql.rds_import_binlog_ssl_material</code> 저장 프로시저를 실행하여 Aurora MySQL DB 클러스터로 SSL 정보를 가져오십시오. <p><code>ssl_material_value</code> 파라미터에서, Aurora MySQL DB 클러스터의 정보를 올바른 JSON 페이로드에 삽입하십시오.</p> <p>다음은 Aurora MySQL DB 클러스터에 SSL 정보를 가져오는 예제입니다. .pem 형식 파일은 본문 코드의 길이가 일반적으로 예제의 본문 코드 길이보다 길입니다.</p> <pre style="border: 1px solid black; padding: 10px;"> call mysql.rds_import_binlog_ssl_material('{"ssl_ca":"-----BEGIN CERTIFICATE-----\nAAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V\nhz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4xyyb/wB96xbiFveSFJuOp/\nd6RJhJOI0iBXr"}'); </pre>

Amazon Aurora Aurora 사용 설명서
Aurora와 MySQL 간의 복제 또는 Aurora
와 다른 Aurora DB 클러스터 간의 복제

데이터베이스 엔진	지침
	<pre>lsLnBItnckiJ7FbtJMXMLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/ cQk+0Fzz qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkyQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwi3f05p6KLxEXAMPLE -----END CERTIFICATE-----\n", "ssl_cert": "-----BEGIN CERTIFICATE----- AAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/ d6RJhJOI0iBXr lsLnBItnckiJ7FbtJMXMLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/ cQk+0Fzz qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkyQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwi3f05p6KLxEXAMPLE -----END CERTIFICATE-----\n", "ssl_key": "-----BEGIN RSA PRIVATE KEY----- AAAAB3NzaC1yc2EAAAQABAAQClKsfkNkuSevGj3eYhCe53pcjqP3maAhDFcvBS706V hz2ItxCih+PnDSUaw+WNQn/mZphTk/a/gU8jEzoOWbkM4yxyb/wB96xbiFveSFJuOp/ d6RJhJOI0iBXr lsLnBItnckiJ7FbtJMXMLvvwJryDUilBMTjYtwB+QhYXUMOzce5Pjz5/i8SeJtjnV3iAoG/ cQk+0Fzz qaeJAAHco+CY/5WrUBkrHmFJr6HcXkvJdWPkyQS3xqC0+FmUzofz221CBt5IMucxXPkX4rWi +z7wB3Rb BQoQzd8v7yeb7OzlPnWOyN0qFU0XA246RA8QFYiCNYwi3f05p6KLxEXAMPLE -----END RSA PRIVATE KEY-----\n"}');</pre>

자세한 정보는 [mysql_rds_import_binlog_ssl_material](#) 및 [Aurora MySQL DB 클러스터에 SSL 사용 \(p. 467\)](#) 단원을 참조하십시오.

Note

프로시저를 실행한 후 파일에 암호를 저장합니다. 이 파일을 나중에 삭제하려면 [mysql_rds_remove_binlog_ssl_material](#) 저장 프로시저를 실행할 수 있습니다.

외부 MySQL 데이터베이스에 대한 이진 로깅을 활성화하려면

1. 명령 셸에서 mysql 서비스를 중지합니다.

```
sudo service mysqld stop
```

2. my.cnf 파일을 편집합니다(이 파일은 보통 /etc에 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 마스터-복제본 관계에서 서버의 고유 식별자를 제공합니다.

암호화된 복제가 필요 없다면, binlogs를 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
```

데이터베이스 엔진	지침
	<pre>sync_binlog=1</pre>
	<p>암호화된 복제가 필요하다면, SSL과 binlogs를 활성화시켜 외부 MySQL 데이터베이스를 시작합니다.</p> <p>/etc/my.cnf 파일 항목에는 MySQL 데이터베이스 서버의 .pem 파일 위치가 포함되어 있습니다.</p> <pre>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</pre>
	<p>또한 MySQL DB 인스턴스의 sql_mode 옵션을 0으로 설정하거나, my.cnf 파일에 이 옵션이 포함되어어서는 안 됩니다.</p> <p>외부 MySQL 데이터베이스 레코드에 연결되어 있는 동안 외부 MySQL 데이터베이스의 이진수 로그 위치를 기록합니다.</p>
	<pre>mysql> SHOW MASTER STATUS;</pre>
	<p>다음과 유사하게 출력되어야 합니다:</p> <pre>+-----+-----+-----+ +-----+ File Position Binlog_Do_DB Binlog_Ignore_DB Executed_Gtid_Set +-----+-----+-----+ mysql-bin.000031 107 +-----+-----+-----+ 1 row in set (0.00 sec)</pre>
	<p>자세한 정보는 MySQL 문서의 Setting the Replication Master Configuration 단원을 참조하십시오.</p>
	<p>3. mysql 서비스를 시작합니다.</p> <pre>sudo service mysqld start</pre>

2. 더 이상 필요 없을 때까지 이진 로그를 복제 마스터에 보관

MySQL binlog 복제를 사용할 경우 Amazon RDS에서 복제 프로세스를 관리하지 않습니다. 따라서 변경 사항이 복제본에 적용된 이후까지 복제 마스터의 binlog 파일이 보관되는지 확인해야 합니다. 이렇게 유지 관리해야 오류 발생 시 마스터 데이터베이스를 복원할 수 있습니다.

다음 데이터베이스 엔진의 이진 로그를 보관하는 방법에 대한 지침을 확인하십시오.

데이터베이스 엔진	지침
Aurora	<p>Aurora MySQL DB 클러스터에 이진 로그를 보관하려면</p> <p>Aurora MySQL DB 클러스터의 binlog 파일에 대한 액세스 권한이 없습니다. 따라서 Amazon RDS에서 binlog 파일을 삭제하기 이전에 변경 사항이 복제본에 적용되도록 복제 마스터에 binlog 파일을 보관할 기간을 충분히 길게 선택해야 합니다. Aurora MySQL DB 클러스터에 binlog 파일을 최대 90일 동안 보관할 수 있습니다.</p> <p>MySQL 데이터베이스 또는 RDS MySQL DB 인스턴스를 복제본으로 사용하여 복제를 설정하고 복제본을 생성할 데이터베이스가 매우 큰 경우, 복제본에 대한 데이터베이스의 초기 복사가 완료되고 복제 지연이 0에 도달할 때까지 binlog 파일을 보관하도록 기간을 길게 선택합니다.</p> <p>binlog 보관 기간을 설정하려면 mysql_rds_set_configuration 프로시저를 사용하여 'binlog retention hours' 구성 파라미터와 함께 DB 클러스터에 binlog 파일을 보관할 시간(최대 2160시간(90일))을 지정합니다. 다음 예에서는 binlog 파일의 보관 기간을 6일로 설정합니다.</p> <div style="border: 1px solid black; padding: 5px;"><pre>CALL mysql.rds_set_configuration('binlog retention hours', 144);</pre></div> <p>복제를 시작한 후 복제본에 대해 SHOW SLAVE STATUS 명령을 실행하고 Seconds behind master 필드를 선택하여 변경 사항이 복제본에 적용되었는지 확인할 수 있습니다. Seconds behind master 필드가 0이면 복제 지연이 없는 것입니다. 복제 지연이 없는 경우 binlog retention hours 구성 파라미터를 더 짧은 기간으로 설정하여 binlog 파일을 보관할 기간을 줄입니다.</p> <p>2160보다 큰 'binlog retention hours' 값을 지정하면 2160이 사용됩니다.</p>
RDS MySQL	<p>Amazon RDS DB 인스턴스에 이진 로그를 보관하려면</p> <p>이전 단원에서 설명한 Aurora MySQL DB 클러스터와 같은 방법으로 binlog 보관 시간을 설정하여 Amazon RDS DB 인스턴스에 binlog 파일을 보관할 수 있습니다.</p> <p>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 복제본에 대한 mysql_rds_stop_replication 프로시저를 호출합니다(mysql.rds_stop_replication 프로시저는 MySQL 버전 5.5, 5.6 이상 및 5.7 이상에서만 사용할 수 있음). 복제가 중지된 동안 Amazon RDS에서는 복제 마스터에서 binlog 파일을 삭제하지 않습니다. 영구 복제본을 사용하여 복제를 설정한 후 복제 마스터와 영구 복제본 사이의 복제 지연(Seconds behind master(마스터보다 지연된 시간(단위: 초)) 필드)이 0에 도달한 경우 읽기 전용 복제본을 삭제할 수 있습니다.</p>
MySQL(외부)	<p>외부 MySQL 데이터베이스에 이진 로그를 보관하려면</p> <p>외부 MySQL 데이터베이스의 binlog 파일은 Amazon RDS에서 관리하지 않으므로 삭제할 때 까지 보관됩니다.</p>

데이터베이스 엔진	지침
	복제를 시작한 후 복제본에 대해 <code>SHOW SLAVE STATUS</code> 명령을 실행하고 <code>Seconds behind master</code> 필드를 선택하여 변경 사항이 복제본에 적용되었는지 확인할 수 있습니다. <code>Seconds behind master</code> 필드가 0이면 복제 지연이 없는 것입니다. 복제 지연이 없는 경우 이전 binlog 파일을 삭제할 수 있습니다.

3. 복제 마스터의 스냅샷 만들기

복제 마스터의 스냅샷은 데이터의 기본 사본을 복제본으로 로드한 후 해당 지점에서 복제를 시작하는 데 사용됩니다.

다음 데이터베이스 엔진의 복제 마스터의 스냅샷을 생성하는 방법에 대한 지침을 확인하십시오.

데이터베이스 엔진	지침
Aurora	<p>Aurora MySQL DB 클러스터의 스냅샷을 만들려면</p> <ol style="list-style-type: none"> Amazon Aurora DB 클러스터의 DB 클러스터 스냅샷을 생성합니다. 자세한 정보는 DB 클러스터 스냅샷 생성 (p. 271) 단원을 참조하십시오. 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 이진수로 깊이 활성화됩니다. 자세한 내용은 DB 클러스터 스냅샷에서 복원 (p. 273) 단원을 참조하십시오. 콘솔에서 Databases(데이터베이스)를 선택하고 복원된 Aurora DB 클러스터의 기본 인스턴스(writer)를 선택하여 세부 정보를 표시합니다. [Recent Events]로 스크롤합니다. binlog 파일 이름과 위치를 포함한 이벤트 메시지가 표시됩니다. 이벤트 메시지의 형식은 다음과 같습니다. <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>Binlog position from crash recovery is binlog-file-name binlog-position</pre> </div> <p>복제 시작 위치에 해당하는 binlog 파일 이름 및 위치 값을 저장합니다.</p> <p>AWS CLI에서 <code>describe-events</code> 명령을 호출하여 binlog 파일 이름 및 위치를 얻을 수도 있습니다. 다음은 <code>describe-events</code> 명령과 명령 출력의 예입니다.</p> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>PROMPT> aws rds describe-events</pre> </div> <div style="border: 1px solid black; padding: 5px; margin-top: 10px;"> <pre>{ "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" }] }</pre> </div>

데이터베이스 엔진	지침
	<p>MySQL 오류 로그에서 마지막 MySQL binlog 파일 위치 또는 <code>DB_CRASH_RECOVERY_BINLOG_POSITION</code> 항목을 확인하여 binlog 파일 이름과 위치를 알 수도 있습니다.</p> <p>4. 복제본 대상이 다른 AWS 계정이 소유한 Aurora DB 클러스터이거나, 외부 MySQL 데이터베이스이거나, RDS MySQL DB 인스턴스인 경우 Amazon Aurora DB 클러스터 스냅샷에서 데이터를 로드할 수 없습니다. 대신 MySQL 클라이언트를 사용하여 DB 클러스터에 연결하고 <code>mysqldump</code> 명령을 실행하여 Amazon Aurora DB 클러스터의 덤프를 생성할 수 있습니다. 생성한 Amazon Aurora DB 클러스터 사본에 대해 <code>mysqldump</code> 명령을 실행해야 합니다. 다음은 예제입니다.</p> <pre>PROMPT> mysqldump --databases <database_name> --single-transaction --order-by-primary -r backup.sql -u <local_user> -p</pre> <p>5. 새로 생성된 Aurora DB 클러스터에서 데이터 덤프 생성을 마치고 나면 해당 DB 클러스터는 더 이상 필요하지 않으므로 삭제합니다.</p>
RDS MySQL	<p>Amazon RDS DB 인스턴스의 스냅샷을 만들려면</p> <ol style="list-style-type: none"> 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 <code>mysql_rds_stop_replication</code> 프로시저를 실행하여 복제를 중지합니다. 읽기 전용 복제본이 중지됨 상태인 동안 읽기 전용 복제본에 연결하고 <code>SHOW SLAVE STATUS</code> 명령을 실행합니다. <code>Relay_Master_Log_File</code> 필드에서 현재 이진 로그 파일 이름을 검색하고 <code>Exec_Master_Log_Pos</code> 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.
MySQL(외부)	<p>외부 MySQL 데이터베이스의 스냅샷을 생성하려면</p> <ol style="list-style-type: none"> 스냅샷을 생성하기 전에 스냅샷의 binlog 위치가 마스터 인스턴스의 최신 데이터와 일치하는지 확인해야 합니다. 이렇게 하려면 먼저 다음 명령을 사용하여 인스턴스에 대한 쓰기 작업을 중지해야 합니다. <pre>mysql> FLUSH TABLES WITH READ LOCK;</pre> 다음과 같이 <code>mysqldump</code> 명령을 사용하여 MySQL 데이터베이스의 덤프를 생성합니다. <pre>PROMPT> sudo mysqldump --databases <database_name> --master-data=2 --single-transaction \ --order-by-primary -r backup.sql -u <local_user> -p</pre> 스냅샷을 생성한 후 다음 명령을 사용하여 MySQL 데이터베이스에서 테이블의 잠금을 해제합니다. <pre>mysql> UNLOCK TABLES;</pre>

4. 복제본 대상으로 스냅샷 로드

Amazon RDS 외부에 있는 MySQL 데이터베이스 덤프에서 데이터를 로드할 계획이라면 EC2 인스턴스를 생성하여 덤프 파일을 복사한 후 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.

다음 데이터베이스 엔진의 복제본 대상에 복제 마스터의 스냅샷을 로드하는 방법에 대한 지침을 확인하십시오.

데이터베이스 엔진	지침
Aurora	<p>Aurora MySQL DB 클러스터로 스냅샷을 로드하려면</p> <ul style="list-style-type: none"> 복제 마스터의 스냅샷이 DB 클러스터 스냅샷인 경우 DB 클러스터 스냅샷에서 복원하여 새 Aurora MySQL DB 클러스터를 복제본 대상으로 생성할 수 있습니다. 자세한 정보는 DB 클러스터 스냅샷에서 복원 (p. 273) 단원을 참조하십시오. 복제 마스터의 스냅샷이 DB 스냅샷인 경우 DB 스냅샷의 데이터를 새 Aurora MySQL DB 클러스터로 마이그레이션할 수 있습니다. 자세한 정보는 Amazon Aurora DB 클러스터로 데이터 마이그레이션 (p. 188) 단원을 참조하십시오. 복제 마스터의 스냅샷이 mysqldump 명령의 출력인 경우 다음 단계를 따릅니다. <ol style="list-style-type: none"> mysqldump 명령의 출력을 복제 마스터에서 Aurora MySQL DB 클러스터에도 연결 가능한 위치로 복사합니다. mysql 명령을 사용하여 Aurora MySQL DB 클러스터에 연결합니다. 다음은 예제입니다. <pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <ol style="list-style-type: none"> 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>
RDS MySQL	<p>Amazon RDS DB 인스턴스로 스냅샷을 로드하려면</p> <ol style="list-style-type: none"> mysqldump 명령의 출력을 복제 마스터에서 MySQL DB 인스턴스에도 연결 가능한 위치로 복사합니다. mysql 명령을 사용하여 MySQL DB 인스턴스에 연결합니다. 다음은 예제입니다. <pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <ol style="list-style-type: none"> mysql 프롬프트에서 source 명령을 실행하고 데이터베이스 덤프 파일의 이름을 전달하여 MySQL DB 인스턴스로 데이터를 로드합니다. 예를 들면 다음과 같습니다. <pre>mysql> source backup.sql;</pre>
MySQL(외부)	<p>외부 MySQL 데이터베이스로 스냅샷을 로드하려면</p> <p>DB 스냅샷 또는 DB 클러스터 스냅샷을 외부 MySQL 데이터베이스로 로드할 수 없습니다. 대신 mysqldump 명령의 출력을 사용해야 합니다.</p>

데이터베이스 엔진	지침
	<p>1. mysqldump 명령의 출력을 복제 마스터에서 MySQL 데이터베이스에도 연결 가능한 위치로 복사합니다.</p> <p>2. mysql 명령을 사용하여 MySQL 데이터베이스에 연결합니다. 다음은 예제입니다.</p> <pre>PROMPT> mysql -h <host_name> -port=3306 -u <db_master_user> -p</pre> <p>3. mysql 프롬프트에서 source 명령을 실행하고 데이터베이스 덤프 파일의 이름을 전달하여 MySQL 데이터베이스로 데이터를 로드합니다. 다음은 예제입니다.</p> <pre>mysql> source backup.sql;</pre>

5. 복제본 대상에 대해 복제 활성화

복제를 활성화하기 전에 Aurora MySQL DB 클러스터 또는 RDS 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.

또한 복제에만 사용되는 사용자 ID를 생성할 수도 있습니다. 다음은 예제입니다.

```
mysql> CREATE USER 'repl_user'@'<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 'repl_user'@'<domain_name>';
```

암호화된 복제를 사용해야 한다면, 복제 사용자에게 SSL 연결이 반드시 필요합니다. 예를 들어, 다음 문 중 하나를 사용하여 사용자 계정 repl_user에서 SSL을 연결합니다.

```
GRANT USAGE ON *.* TO 'repl_user'@'<domain_name>' REQUIRE SSL;
```

Note

REQUIRE SSL이 포함되어 있지 않으면, 복제 연결이 자동으로 암호화되지 않은 연결로 돌아갈 수 있습니다.

다음 데이터베이스 엔진의 복제를 활성화하는 방법에 대한 지침을 확인하십시오.

데이터베이스 엔진	지침
Aurora	<p>Aurora MySQL DB 클러스터에서 복제를 활성화하려면</p> <p>1. DB 클러스터 스냅샷에서 DB 클러스터 복제본 대상을 생성한 경우 DB 클러스터 복제본 대상에 연결하고 SHOW MASTER STATUS 명령을 실행합니다. File 필드에서 현재 이진로그 파일 이름을 검색하고 Position 필드에서 로그 파일 위치를 검색합니다.</p>

Amazon Aurora Aurora 사용 설명서
Aurora와 MySQL 간의 복제 또는 Aurora
와 다른 Aurora DB 클러스터 간의 복제

데이터베이스 엔진	지침
	<p>DB 스냅샷에서 DB 클러스터 복제본 대상을 생성한 경우 복제 시작 위치에 해당하는 binlog 파일 및 binlog 위치가 필요합니다. 복제 마스터의 스냅샷을 생성할 때 SHOW SLAVE STATUS 명령에서 이 값을 검색했습니다.</p> <p>2. DB 클러스터에 연결하고 mysql_rds_set_external_master 프로시저와 mysql_rds_start_replication 프로시저를 실행하여 이전 단계의 이진수 로그 파일 이름과 위치를 사용하여 복제 마스터로 복제를 시작합니다. 다음은 예제입니다.</p> <div style="border: 1px solid black; padding: 10px;"><pre>CALL mysql.rds_set_external_master ('mydbinstance.123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;</pre></div>
RDS MySQL	<p>Amazon RDS DB 인스턴스에서 복제를 활성화하려면</p> <p>1. DB 스냅샷에서 DB 인스턴스 복제본 대상을 생성한 경우 복제 시작 위치에 해당하는 binlog 파일 및 binlog 위치가 필요합니다. 복제 마스터의 스냅샷을 생성할 때 SHOW SLAVE STATUS 명령에서 이 값을 검색했습니다.</p> <p>2. DB 인스턴스에 연결하고 mysql_rds_set_external_master 프로시저와 mysql_rds_start_replication 프로시저를 실행하여 이전 단계의 이진수 로그 파일 이름과 위치를 사용하여 복제 마스터로 복제를 시작합니다. 다음은 예제입니다.</p> <div style="border: 1px solid black; padding: 10px;"><pre>CALL mysql.rds_set_external_master ('mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', 3306, 'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0); CALL mysql.rds_start_replication;</pre></div>

데이터베이스 엔진	지침
MySQL(외부)	<p>외부 MySQL 데이터베이스에서 복제를 활성화 하려면</p> <p>1. 복제 시작 위치에 해당하는 binlog 파일 및 binlog 위치를 검색합니다. 복제 마스터의 스냅샷을 생성할 때 SHOW SLAVE STATUS 명령에서 이 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 예제입니다.</p> <pre>-- -- Position to start replication or point-in-time recovery from -- -- CHANGE MASTER TO MASTER_LOG_FILE='mysql-bin-changelog.000031', MASTER_LOG_POS=107;</pre> <p>2. 예를 들어, 외부 MySQL 복제본 대상에 연결하고 CHANGE MASTER TO 및 START SLAVE를 실행하여 이전 단계의 이진수 로그 파일 이름과 위치를 사용하여 복제 마스터로 복제를 시작합니다.</p> <pre>CHANGE MASTER TO MASTER_HOST = 'mydbcluster.cluster-123456789012.us-east-1.rds.amazonaws.com', MASTER_PORT = 3306, MASTER_USER = 'repl_user', MASTER_PASSWORD = '<password>', MASTER_LOG_FILE = 'mysql-bin-changelog.000031', MASTER_LOG_POS = 107; START SLAVE;</pre>

6. 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 서비스 사용 \(p. 429\)](#) 단원을 참조하십시오.

또한 복제본 대상에 연결하고 SHOW SLAVE STATUS 명령을 실행하여 복제본 대상이 복제 마스터보다 얼마나 지연되었는지 모니터링할 수 있습니다. 명령 출력의 Seconds Behind Master 필드는 복제본 대상이 마스터보다 얼마나 지연되었는지 알려줍니다.

Aurora와 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터 간의 복제 중지

MySQL DB 인스턴스, 외부 MySQL 데이터베이스 또는 다른 Aurora DB 클러스터와의 binlog 복제 중지 단계는 다음과 같으며, 이 주제의 뒷부분에 자세히 설명되어 있습니다.

- 복제본 대상의 Binlog 복제 중지 (p. 578)
- 복제 마스터에 대한 이진 로깅 비활성화 (p. 579)

1. 복제본 대상의 Binlog 복제 중지

다음 데이터베이스 엔진의 binlog 복제를 중지하는 방법에 대한 지침을 확인하십시오.

데이터베이스 엔진	지침
Aurora	<p>Aurora MySQL DB 클러스터 복제본 대상에 대한 binlog 복제를 중지하려면</p> <p>복제본 대상인 Aurora DB 클러스터에 연결하고, mysql_rds_stop_replication 프로시저를 호출합니다. <code>mysql.rds_stop_replication</code> 프로시저는 MySQL 5.5 이상, 5.6 이상, 5.7 이상 버전에서만 사용할 수 있습니다.</p>
RDS MySQL	<p>Amazon RDS DB 인스턴스에 대한 binlog 복제를 중지하려면</p> <p>복제본 대상인 RDS DB 클러스터에 연결하고, mysql_rds_stop_replication 프로시저를 호출합니다. <code>mysql.rds_stop_replication</code> 프로시저는 MySQL 5.5 이상, 5.6 이상, 5.7 이상 버전에서만 사용할 수 있습니다.</p>
MySQL(외부)	<p>외부 MySQL 데이터베이스에서 binlog 복제를 중지하는 방법</p> <p>MySQL 데이터베이스에 연결하고 <code>STOP REPLICATION</code> 명령을 호출합니다.</p>

2. 복제 마스터에 대한 이진 로깅 비활성화

다음 데이터베이스 엔진의 복제 마스터에 대한 이진 로깅을 비활성화하는 방법의 지침을 확인하십시오.

데이터베이스 엔진	지침
Aurora	<p>Amazon Aurora DB 클러스터에 대한 이진 로깅을 비활성화하는 방법</p> <ol style="list-style-type: none"> 복제 마스터인 Aurora DB 클러스터를 연결하고, binlog 보존 시간 프레임을 0으로 설정합니다. binlog 보관 기간을 설정하려면, 다음 예제와 같이 mysql_rds_set_configuration 프로시저를 사용하여 'binlog retention hours' 구성 파라미터를 DB 클러스터에 binlog 파일을 보관할 시간(이 예제에서는 0)과 함께 지정합니다. <pre>CALL mysql.rds_set_configuration('binlog retention hours', 0);</pre> <ol style="list-style-type: none"> 복제 마스터에서 <code>binlog_format</code> 파라미터를 OFF로 설정합니다. <code>binlog_format</code> 파라미터는 기본 클러스터 파라미터 그룹에 속하는 클러스터 수준 파라미터입니다. <p><code>binlog_format</code> 파라미터 값을 변경한 후에는 DB 클러스터를 재부팅해야만 변경 사항이 적용됩니다.</p> <p>자세한 정보는 Amazon Aurora DB 클러스터와 DB 인스턴스 파라미터 (p. 170) 및 DB 파라미터 그룹의 파라미터 수정 (p. 174) 단원을 참조하십시오.</p>
RDS MySQL	<p>Amazon RDS DB 인스턴스에 대한 이진 로깅을 비활성화하려면</p> <p>Amazon RDS DB 인스턴스에 대한 이진 로깅을 직접 비활성화할 수는 없지만, 다음을 수행하여 비활성화할 수 있습니다.</p> <ol style="list-style-type: none"> DB 인스턴스의 자동 백업을 비활성화합니다. 기존 DB 인스턴스를 수정하고 백업 보존 기간을 0으로 설정하여 자동 백업을 비활성화할 수 있습니다. 자세한 내용은 Amazon Relational Database Service 사용 설명서의 Amazon RDS DB 인스턴스 수정 및 백업 작업을 참조하십시오. 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.

데이터베이스 엔진	지침
MySQL(외부)	<p>외부 MySQL 데이터베이스에 대한 이진 로깅을 비활성화하는 방법</p> <p>MySQL 데이터베이스에 연결하고 <code>STOP REPLICATION</code> 명령을 호출합니다.</p> <p>1. 명령 셸에서 mysql 서비스를 중지합니다.</p> <pre>sudo service mysqld stop</pre> <p>2. my.cnf 파일을 편집합니다(이 파일은 보통 /etc에 있음).</p> <pre>sudo vi /etc/my.cnf</pre> <p><code>log_bin</code> 섹션에서 <code>server_id</code> 및 [<code>mysqld</code>]옵션을 삭제합니다.</p> <p>자세한 정보는 MySQL 문서의 Setting the Replication Master Configuration 단원을 참조하십시오.</p> <p>3. mysql 서비스를 시작합니다.</p> <pre>sudo service mysqld start</pre>

Aurora MySQL

아래에서는 Aurora MySQL 클러스터와 외부 원본 사이에 이진 로그(binlog) 복제를 통해 전역 트랜잭션 식별자(GTID)를 사용하는 방법을 배울 수 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, 오픈소스 MySQL 데이터베이스 또는 다른 AWS 리전의 Aurora DB 클러스터일 수 있습니다. 이러한 종류의 복제를 구성하는 방법에 대해 알아보려면 [Aurora와 MySQL 간의 복제 또는 Aurora와 다른 Aurora DB 클러스터 간의 복제 \(p. 567\)](#) 단원을 참조하십시오.

binlog 복제를 사용하고 있지만 MySQL을 사용한 GTID 기반 복제에 대해 잘 알지 못하는 경우 MySQL 설명서의 [전역 트랜잭션 식별자를 사용한 복제](#)에서 배경 정보를 참조하십시오.

Note

GTID 기반 복제는 Aurora MySQL 2.04 이후 버전의 MySQL 5.7 호환 클러스터에서 지원됩니다.
GTID 기반 복제는 Aurora MySQL 1 버전의 MySQL 5.6 호환 클러스터에서는 지원되지 않습니다.

주제

- [전역 트랜잭션 식별자\(GTID\) 개요 \(p. 581\)](#)
- [GTID 기반 복제 파라미터 \(p. 581\)](#)
- [Aurora MySQL 클러스터에 대한 GTID 기반 복제 구성 \(p. 582\)](#)
- 및 [Aurora MySQL DB 클러스터에 대해 GTID 기반 복제 비활성화 \(p. 582\)](#)

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 마스터 및 복제 본이 일치하는지를 쉽게 확인할 수 있습니다.

외부 MySQL 호환 데이터베이스에서 Aurora 클러스터로 복제할 때 일반적으로 Aurora에서 GTID 기반 복제를 사용합니다. 오픈소스 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	<p>OFF는 새 트랜잭션을 익명 트랜잭션(GTID가 없음)으로 지정하며, 트랜잭션을 복제하려면 익명이어야 합니다.</p> <p>OFF_PERMISSIVE는 새 트랜잭션을 익명 트랜잭션(GTID가 없음)으로 지정하지만, 모든 트랜잭션을 복제할 수 있습니다.</p> <p>ON_PERMISSIVE는 새 트랜잭션을 GTID 트랜잭션으로 지정하지만, 모든 트랜잭션을 복제할 수 있습니다.</p> <p>ON은 새 트랜잭션을 GTID 트랜잭션으로 지정하고, 트랜잭션을 복제하려면 GTID 트랜잭션이어야 합니다.</p>
enforce_gtid_consistency	OFF, ON, WARN	<p>OFF는 트랜잭션이 GTID 일관성을 위반하는 것을 허용합니다.</p> <p>ON은 트랜잭션이 GTID 일관성을 위반하지 않도록 합니다.</p>

파라미터	유효한 값	설명
		WARN은 트랜잭션의 GTID 일관성을 위반하는 것을 허용하지만, 위반이 발생할 경우 경고를 생성합니다.

GTID 기반 복제의 경우 Aurora MySQL DB 클러스터의 DB 클러스터 파라미터 그룹에 대해 이 설정을 사용하십시오.

- ON 및 ON_PERMISSIVE는 RDS DB 인스턴스 또는 Aurora MySQL 클러스터에서 밖으로 복제하는 경우에만 적용됩니다. 이 두 값으로 인해 외부 데이터베이스로 복제되는 트랜잭션에 대해 RDS DB 인스턴스 또는 Aurora DB 클러스터가 GTID를 사용하게 됩니다. ON은 외부 데이터베이스에서도 GTID 기반 복제를 사용할 것을 요구합니다. ON_PERMISSIVE로 인해 외부 데이터베이스에서 GTID 기반 복제는 선택 사항입니다.
- OFF_PERMISSIVE가 설정된 경우 이는 RDS DB 인스턴스 또는 Aurora DB 클러스터가 외부 데이터베이스에서 안으로 복제하는 것을 수락할 수 있음을 뜻합니다. 외부 데이터베이스가 GTID 기반 복제를 사용하는지 않으면 이렇게 할 수 있습니다.
- OFF가 설정된 경우 이는 RDS DB 인스턴스 또는 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_PERMISSIVE로 설정하는 것이 좋습니다. 이 설정을 통해 복제 원본의 GTID 설정에 관계없이 외부 데이터베이스에서 안으로 복제하는 것이 가능해집니다.

파라미터 그룹에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

Note

AWS Management 콘솔 콘솔에서 gtid_mode 파라미터는 gtid-mode로 표시됩니다.

Aurora MySQL 클러스터에 대한 GTID 기반 복제 구성

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 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#)의 절차를 따르십시오.

및 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 뜻

하는 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 \(p. 683\)](#) 단원을 참조하십시오.

Aurora MySQL DB 클러스터에 대해 GTID 기반 복제를 비활성화 하려면

1. Aurora 기본 인스턴스에서 다음 절차를 수행합니다.

```
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 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 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. a. Aurora 기본 인스턴스에서 SHOW MASTER STATUS 명령을 실행합니다.

다음과 유사하게 출력되어야 합니다.

File	Position
mysql-bin-changelog.000031	107

출력에서 파일 및 위치를 메모합니다.

- b. 각 읽기 전용 복제본에서 이전 단계에서 메모한 마스터의 파일 및 위치 정보를 사용하여 다음 쿼리를 실행합니다.

```
SELECT MASTER_POS_WAIT(file, position);
```

예를 들어, 파일 이름이 mysql-bin-changelog.000031이고 위치가 107일 경우 다음 명령문을 실행합니다.

```
SELECT MASTER_POS_WAIT(mysql-bin-changelog.000031, 107);
```

읽기 전용 복제본이 지정된 위치에 전달되면 쿼리가 즉시 반환합니다. 그렇지 않으면 함수가 대기합니다. 쿼리가 모든 읽기 전용 복제본에 대해 반환을 완료하면 다음 단계를 진행합니다.

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

Aurora 멀티 마스터 클러스터 작업

다음과 같이 Aurora 멀티 마스터 클러스터에 대해 자세히 배울 수 있습니다. 멀티 마스터 클러스터에서는 모든 DB 인스턴스가 읽기-쓰기 기능을 가집니다. 멀티 마스터 클러스터는 단일 마스터 클러스터와는 다른 가용성 특성, 데이터베이스 기능 지원, 모니터링 및 문제 해결 절차를 가지고 있습니다.

주제

- [Aurora 멀티 마스터 클러스터의 개요 \(p. 584\)](#)
- [Aurora 멀티 마스터 클러스터 생성 \(p. 589\)](#)
- [Aurora 멀티 마스터 클러스터 관리 \(p. 594\)](#)
- [Aurora 멀티 마스터 클러스터를 위한 애플리케이션 고려 사항 \(p. 596\)](#)
- [Aurora 멀티 마스터 클러스터의 성능 고려 사항 \(p. 606\)](#)
- [Aurora 멀티 마스터 클러스터에 대한 접근 방식 \(p. 607\)](#)

Aurora 멀티 마스터 클러스터의 개요

다음과 같은 배경 정보를 사용하면 새 Aurora 클러스터를 설정할 때 멀티 마스터 클러스터를 선택할지, 아니면 단일 마스터 클러스터를 선택할지 결정하는 데 도움이 됩니다. 정보를 토대로 결정을 내리려면 스키마 설계 및 애플리케이션 로직이 멀티 마스터 클러스터에서 가장 잘 작동하도록 하려면 어떻게 적용시켜야 하는지를 먼저 이해해야 합니다.

새 Amazon Aurora 클러스터 각각에 대해 단일 마스터 클러스터와 멀티 마스터 클러스터 중 어떤 것을 생성할 것인지 결정할 수 있습니다.

대부분의 Aurora 클러스터 유형은 단일 마스터 클러스터입니다. 예를 들어 프로비저닝된 클러스터, Aurora Serverless 클러스터 및 글로벌 데이터베이스 클러스터는 모두 단일 마스터 클러스터입니다. 단일 마스터 클러스터에서 단일 DB 인스턴스는 모든 쓰기 작업을 수행하며, 기타 모든 DB 인스턴스는 읽기 전용입니다. 라이터 DB 인스턴스가 사용 불가 상태가 되면 장애 조치 메커니즘이 읽기 전용 인스턴스 중 하나를 새 라이터로 승격합니다.

멀티 마스터 클러스터에 있는 모든 DB 인스턴스는 쓰기 작업을 수행할 수 있습니다. 읽기-쓰기 기본 인스턴스 및 다중 읽기 전용 Aurora 복제본 개념은 적용되지 않습니다. 라이터 DB 인스턴스가 사용 불가 상태가 될 때 어떤 장애 조치도 없습니다. 또 다른 라이터 DB 인스턴스가 실패한 인스턴스의 작업을 인계하기 위해 즉시 사용 가능한 상태가 되기 때문입니다. 단일 마스터 클러스터에서 제공되는 고가용성(장애 조치 동안 짧은 가동 중지)과 구분하기 위해 이러한 유형의 가용성을 지속적인 가용성이라고 합니다.

멀티 마스터 클러스터는 여러 측면에서 다른 종류의 Aurora 클러스터(예: 프로비저닝된 클러스터, Aurora Serverless 클러스터, 병렬 쿼리 클러스터)와 다르게 작동합니다. 멀티 마스터 클러스터에서는 영역에서 고가용성, 모니터링, 연결 관리 및 데이터베이스 기능 같은 여러 요소를 고려합니다. 예를 들어 데이터베이스 쓰기 작업에서 짧은 가동 중지도 허용할 수 없는 애플리케이션에서 멀티 마스터 클러스터는 라이터 인스턴스가 사용 불가 상태가 될 때 중단을 피하는 데 도움이 될 수 있습니다. 멀티 마스터 클러스터는 장애 조치 메커니즘을 사용하지 않습니다. 또 다른 DB 인스턴스를 읽기-쓰기 기능을 갖도록 승격시킬 필요가 없기 때문입니다. 멀티 마스터 클러스터에서 단일 기본 인스턴스 대신에 모든 DB 인스턴스에 대한 DML 처리량, 지연 시간 및 교착 상태와 관련된 지표를 검토합니다.

현재 멀티 마스터 클러스터에서는 MySQL 5.6과 호환 가능한 Aurora MySQL 버전 1이 필요합니다.

멀티 마스터 클러스터를 생성하려면 클러스터를 생성할 때 Database features(데이터베이스 기능) 아래의 Multiple writers(다중 라이터)를 선택합니다. 이렇게 하면 DB 인스턴스 간의 복제, 가용성 및 성능을 다른 종

류의 Aurora 클러스터와 다르게 동작 시킬 수 있습니다. 이러한 옵션은 클러스터 수명 동안 효과가 유지됩니다. 멀티 마스터 클러스터에 해당되는 전문적인 사용 사례를 이해해야 합니다.

주제

- [멀티 마스터 클러스터 용어 \(p. 585\)](#)
- [멀티 마스터 클러스터 아키텍처 \(p. 586\)](#)
- [멀티 마스터 클러스터를 위한 권장 워크로드 \(p. 587\)](#)
- [멀티 마스터 클러스터의 장점 \(p. 587\)](#)
- [멀티 마스터 클러스터의 제한 사항 \(p. 588\)](#)

멀티 마스터 클러스터 용어

다음 정의를 확인하여 멀티 마스터 클러스터에 대한 용어를 이해할 수 있습니다. 이러한 용어들은 멀티 마스터 클러스터를 위한 설명서 전반에서 사용됩니다.

라이터

쓰기 작업을 수행할 수 있는 DB 인스턴스입니다. Aurora 멀티 마스터 클러스터에서는 모든 DB 인스턴스가 라이터입니다. 이는 오직 하나의 DB 인스턴스만 라이터 역할을 할 수 있는 Aurora 단일 마스터 클러스터와 확연하게 다른 점입니다. 단일 마스터 클러스터에서는 라이터가 사용 불가 상태가 되면 장애 조치 메커니즘이 또 다른 DB 인스턴스를 새 라이터로 승격합니다. 멀티 마스터 클러스터에서는 애플리케이션이 실패한 DB 인스턴스에서 클러스터의 또 다른 DB 인스턴스로 쓰기 작업을 리디렉션할 수 있습니다.

멀티 마스터

각 DB 인스턴스가 읽기 및 쓰기 작업을 모두 수행할 수 있는 Aurora 클러스터의 아키텍처입니다. 단일 마스터와 이를 비교하십시오. 멀티 마스터 클러스터는 멀티 테넌트 애플리케이션용 워크로드 같이 조각화된 워크로드에 가장 적합합니다.

단일 마스터

Aurora 클러스터의 기본 아키텍처입니다. 단일 DB 인스턴스(기본 인스턴스)는 쓰기 작업을 수행합니다. 기타 모든 DB 인스턴스(Aurora 복제본)는 읽기 전용 쿼리 트래픽을 처리합니다. 이를 멀티 마스터와 비교하십시오. 이 아키텍처는 범용 애플리케이션에 적합합니다. 이러한 애플리케이션에서는 단일 DB 인스턴스가 모든 데이터 조작 언어(DML) 및 데이터 정의 언어(DDL) 문을 처리합니다. 확장성 문제는 거의 SELECT 쿼리가 관련됩니다.

쓰기 충돌

다른 DB 인스턴스가 동시에 동일한 데이터 페이지를 수정할 때 발생하는 상황입니다. Aurora은 애플리케이션에 쓰기 충돌을 교착 상태 오류로 보고합니다. 이 오류 조건은 트랜잭션의 룰백을 야기합니다. 애플리케이션은 오류 코드를 탐지하여 트랜잭션을 다시 시도해야 합니다.

Aurora 멀티 마스터 클러스터에서 주된 설계 고려 사항 및 성능 튜닝 목표는 쓰기 충돌을 최소화하는 방식으로 DB 인스턴스 간에 쓰기 작업을 나누는 것입니다. 바로 이것이 멀티 마스터 클러스터가 샤딩된 애플리케이션에 적합한 이유입니다. 쓰기 충돌 메커니즘에 대한 자세한 내용은 [멀티 마스터 클러스터에서의 충돌 해결 \(p. 606\)](#) 단원을 참조하십시오.

샤딩

조각화된 워크로드의 특정 클래스입니다. 데이터는 많은 파티션, 테이블, 데이터베이스 또는 별도 클러스터로 물리적으로 나뉘어집니다. 데이터의 특정 부분에 대한 컨테이너를 샤드라고 합니다. Aurora 멀티 마스터 클러스터에서 각 샤드는 특정 DB 인스턴스가 관리하며, 하나의 DB 인스턴스가 여러 개의 샤드를 책임질 수 있습니다. 샤딩된 스키마 설계는 Aurora 멀티 마스터 클러스터에서 연결을 관리하는 방법과 잘 매핑됩니다.

샤드

샤딩된 배포 내의 세부 수준 단위입니다. 테이블, 관련 테이블 세트, 데이터베이스, 파티션 또는 전체 클러스터가 세부 수준 단위가 될 수 있습니다. Aurora 멀티 마스터 클러스터에서는 샤딩된 애플리케이션의

데이터를 단일 Aurora 공유 스토리지 볼륨으로 통합하여 데이터베이스를 지속적으로 사용 가능한 상태로 만들고 데이터를 관리가 쉬운 상태로 만들 수 있습니다. 각 DB 인스턴스에서 어떤 샤드가 관리되는지 확인합니다. 데이터를 물리적으로 재구성하지 않고 언제라도 이 매핑을 변경할 수 있습니다.

리샤딩

샤딩된 데이터를 물리적으로 재구성하여 서로 다른 DB 인스턴스가 고유한 테이블 또는 데이터베이스를 처리할 수 있도록 하는 것입니다. 워크로드 변화나 DB 인스턴스 결합에 대한 응답으로 Aurora 멀티 마스터 클러스터 내에서 데이터를 물리적으로 재구성할 필요가 없습니다. 클러스터의 모든 DB 인스턴스가 공유 스토리지 볼륨을 통해 모든 데이터베이스 및 테이블에 액세스할 수 있기 때문에 리샤딩 작업을 피할 수 있습니다.

멀티테넌트

조각화된 워크로드의 특정 클래스입니다. 각 고객, 클라이언트 또는 사용자에 대한 데이터가 별도의 테이블 또는 데이터베이스에 유지됩니다. 이 설계는 격리를 보장하고 개별 사용자의 수준에서 용량 및 리소스를 관리하는 데 도움이 됩니다.

BYOS(Bring-Your-Own-Shard)

데이터베이스 스키마와 샤딩을 사용하는 연결 애플리케이션을 이미 가지고 있는 경우입니다. Aurora 멀티 마스터 클러스터로 이러한 배포를 비교적 쉽게 전달할 수 있습니다. 이 경우에는 서버 통합 및 고가용성 같은 Aurora 이점을 조사하는 데 노력을 집중할 수 있습니다. 쓰기 요청에서 여러 연결을 처리하는 새 애플리케이션로 직을 생성할 필요가 없습니다.

GRAW(Global Read-After-Write)

모든 읽기 작업에서 항상 데이터의 최신 상태를 볼 수 있도록 동기화를 도입하는 설정입니다. 기본적으로 멀티 마스터 클러스터의 읽기 작업에서 보이는 데이터에서는 복제 지연(보통 몇 밀리초)이 발생합니다. 동일한 데이터가 다른 DB 인스턴스에서 동시에 수정되기 때문에 이렇게 짧은 간격 동안 한 DB 인스턴스의 쿼리가 상태 데이터를 검색할 수 있습니다. 이 설정을 활성화하려면 `aurora_mm_session_consistency_level`을 기본 설정인 `INSTANCE_RAW`에서 `REGIONAL_RAW`으로 변경합니다. 이렇게 하면 읽기 및 쓰기를 수행하는 DB 인스턴스에 관계 없이 읽기 작업에서 클러스터 전반의 일관성이 보장됩니다. GRAW 모드에 대한 자세한 내용은 [멀티 마스터 클러스터의 일관성 모델 \(p. 599\)](#) 단원을 참조하십시오.

멀티 마스터 클러스터 아키텍처

멀티 마스터 클러스터는 다른 종류의 Aurora 클러스터와 다른 아키텍처를 가지고 있습니다. 멀티 마스터 클러스터에서는 모든 DB 인스턴스가 읽기-쓰기 기능을 가집니다. 다른 종류의 Aurora 클러스터는 모든 쓰기 작업을 수행하는 단일 전용 DB 인스턴스를 갖는 반면, 기타 모든 DB 인스턴스는 읽기 전용으로 `SELECT` 쿼리만 처리합니다. 멀티 마스터 클러스터는 기본 인스턴스 또는 읽기 전용 Aurora 복제본을 갖지 않습니다.

애플리케이션은 어떤 DB 인스턴스가 어떤 요청을 처리하는지 제어합니다. 따라서 멀티 마스터 클러스터에서는 DML 및 DDL 문을 발급하기 위해 개별 인스턴스 엔드포인트를 연결합니다. 이와 달리 다른 종류의 Aurora 클러스터들은 일반적으로 모든 쓰기 작업은 단일 클러스터 엔드포인트로, 모든 읽기 작업은 단일 리더 엔드포인트로 보냅니다.

Aurora 멀티 마스터 클러스터를 위한 기반 스토리지는 단일 마스터 클러스터를 위한 스토리지와 비슷합니다. 데이터는 자동 확장되는 매우 안정적인 공유 스토리지 볼륨에 여전히 저장됩니다. 중요한 차이는 DB 인스턴스의 수와 유형에 있습니다. 멀티 마스터 클러스터에는 N개의 읽기-쓰기 노드가 있습니다. 현재 N의 최대 값은 2입니다.

멀티 마스터 클러스터에는 전담 읽기 전용 노드가 없습니다. 따라서 Aurora 복제본에 대한 Aurora 절차 및 가이드라인이 멀티 마스터 클러스터에는 적용되지 않습니다. 다른 DB 인스턴스에 읽기 및 쓰기 워크로드를 배치하기 위해 일시적으로 DB 인스턴스를 읽기 전용으로 만들 수 있습니다. 그 방법은 [인스턴스 읽기 전용 모드 사용 \(p. 605\)](#) 단원을 참조하십시오.

멀티 마스터 클러스터 노드는 대기 시간과 지역 시간이 짧은 Aurora 복제를 사용해 연결됩니다. 멀티 마스터 클러스터는 전체-전체(all-to-all) 피어 투 피어 복제를 사용합니다. 복제는 라이터 간에 직접 작동합니다. 모든 라이터는 다른 모든 라이터에 대한 변경 사항을 복제합니다.

멀티 마스터 클러스터의 DB 인스턴스는 재시작과 복원을 독립적으로 처리합니다. 한 라이터가 다시 시작되면 다른 라이터들을 다시 시작할 필요가 없습니다. 세부 정보는 [Aurora 멀티 마스터 클러스터를 위한 고가용 성 고려 사항 \(p. 595\)](#) 단원을 참조하십시오.

멀티 마스터 클러스터는 모든 데이터베이스 인스턴스 내에서 이루어진 데이터에 대한 모든 변경 사항을 계속해서 추적합니다. 측정 단위는 데이터 페이지로, 16KB로 크기가 고정되어 있습니다. 이러한 변경에는 테이블 데이터, 보조 인덱스 및 시스템 테이블에 대한 수정이 포함됩니다. 변경은 Aurora 내부 정리 작업의 결과일 수도 있습니다. Aurora는 여러 물리적 복사본 간에 일관성을 보장합니다. Aurora는 각 데이터 페이지에 대해 공유 스토리지 볼륨 및 DB 인스턴스의 메모리에 이러한 복사본을 유지합니다.

두 개의 DB 인스턴스가 거의 동시에 동일한 데이터 페이지를 수정하려고 시도하는 경우에 쓰기 충돌이 발생합니다. 큐럼 투표 메커니즘을 사용하여 가장 빠른 변경 요청이 승인됩니다. 이 변경 사항은 영구 스토리지에 저장됩니다. 변경이 승인되지 않은 DB 인스턴스는 시도된 변경을 포함한 전체 트랜잭션을 롤백합니다. 트랜잭션을 롤백하면 데이터가 일관된 상태로 유지되고, 애플리케이션에서 항상 예측 가능한 방식으로 데이터를 볼 수 있습니다. 애플리케이션은 교착 상태를 탐지하여 전체 트랜잭션을 다시 시도할 수 있습니다.

쓰기 충돌 및 이와 관련된 성능 오버헤드를 최소화하는 방법은 [멀티 마스터 클러스터에서의 충돌 해결 \(p. 606\)](#) 단원을 참조하십시오.

멀티 마스터 클러스터를 위한 권장 워크로드

멀티 마스터 클러스터는 특정 종류의 워크로드에서 가장 잘 작동합니다.

액티브-패시브 워크로드

액티브-패시브 워크로드에서는 한 번에 하나의 DB 인스턴스에서 모든 읽기 및 쓰기 작업이 수행됩니다. Aurora 클러스터의 다른 모든 DB 인스턴스가 비축됩니다. 원래 액티브 DB 인스턴스가 사용 불가 상태가 되면 모든 읽기 및 쓰기 작업이 다른 DB 인스턴스로 즉시 전환합니다. 이러한 구성에서는 쓰기 작업의 가동 중지가 최소화됩니다. 다른 DB 인스턴스들은 장애 조치를 수행하지 않고 애플리케이션의 모든 처리를 인계할 수 있습니다.

액티브-액티브 워크로드

액티브-액티브 워크로드에서는 한 번에 모든 DB 인스턴스에 대해 읽기 및 쓰기 작업이 수행됩니다. 이 구성에서는 다른 DB 인스턴스가 동시에 동일한 기반 데이터를 수정하지 못하도록 보통 워크로드를 조각화합니다. 이를 통해 쓰기 충돌 가능성을 최소화합니다.

멀티 마스터 클러스터는 조각화된 워크로드를 위해 설계된 애플리케이션 로직에서 잘 작동합니다. 이러한 유형의 워크로드에서는 데이터베이스 인스턴스, 데이터베이스 테이블 또는 테이블 파티션에 따라 쓰기 작업을 분할합니다. 예를 들어 특정 DB 인스턴스에 각각을 할당하여 동일한 클러스터에서 여러 개의 애플리케이션을 실행할 수 있습니다. 아니면 여러 개의 소형 테이블(예: 온라인 서비스의 각 사용자를 위한 하나의 테이블)을 사용하는 애플리케이션을 실행할 수 있습니다. 다른 DB 인스턴스의 쓰기 작업이 동일한 테이블 내의 중첩된 행에 대한 동시 업데이트를 수행하지 않도록 스키마를 설계하는 것이 가장 좋습니다. 샤딩된 애플리케이션은 이러한 종류의 아키텍처의 한 예입니다.

액티브-액티브 워크로드의 설계에 대한 예제는 [샤딩된 데이터베이스에서 멀티 마스터 클러스터 사용 \(p. 607\)](#) 단원을 참조하십시오.

멀티 마스터 클러스터의 장점

Aurora 멀티 마스터 클러스터를 통해 다음과 같은 장점을 활용할 수 있습니다.

- 멀티 마스터 클러스터는 이미 뛰어난 Aurora의 가용성을 한층 개선합니다. 클러스터의 다른 DB 인스턴스가 다시 시작되는 문제를 초래하지 않고도 읽기-쓰기 DB 인스턴스를 다시 시작할 수 있습니다. 읽기-쓰기 DB 인스턴스가 사용 불가 상태가 될 때 장애 조치 프로세스 및 관련 지연이 발생하지 않습니다.
- 멀티 마스터 클러스터는 샤딩된 멀티 테넌트 애플리케이션에 적합합니다. 데이터를 관리할 때 복잡한 리纱딩 작업을 피할 수 있습니다. 적은 수의 클러스터나 DB 인스턴스에 샤딩된 애플리케이션을 통합하는 것이 가능할 수 있습니다. 세부 정보는 [샤딩된 데이터베이스에서 멀티 마스터 클러스터 사용 \(p. 607\)](#) 단원을 참조하십시오.

- Aurora는 트랜잭션이 커밋될 때가 아니라 즉시 쓰기 충돌을 탐지합니다. 쓰기 충돌 메커니즘에 대한 자세한 내용은 [멀티 마스터 클러스터에서의 충돌 해결 \(p. 606\)](#) 단원을 참조하십시오.

멀티 마스터 클러스터의 제한 사항

Note

Aurora 멀티 마스터 클러스터는 지속적인 가용성이 요구되는 사용 사례에 매우 적합합니다. 따라서 이러한 클러스터를 모든 워크로드에 일반적으로 적용할 수 없을 수 있습니다. Aurora 단일 마스터 클러스터에서 더 큰 DB 인스턴스 클래스를 사용하면 성능, 확장성 및 가용성에 대한 요구 사항을 충족할 수 있습니다. 이 경우에는 프로비저닝된 클러스터나 Aurora Serverless 클러스터를 사용하는 방법을 고려하십시오.

AWS 및 Aurora 제한 사항

다음과 같은 제한 사항은 현재 AWS와 멀티 마스터 클러스터에서 사용할 수 있는 Aurora 기능에 적용되고 있습니다.

- 현재 하나의 멀티 마스터 클러스터에 최대 두 개의 DB 인스턴스를 가질 수 있습니다.
- 또한 멀티 마스터 클러스터의 모든 DB 인스턴스들이 동일한 AWS 리전에 있어야 합니다.
- 멀티 마스터 클러스터에 나온 교차 리전 복제본을 사용할 수 없습니다.
- Stop 작업은 멀티 마스터 클러스터에서 사용할 수 없습니다.
- 유지 가능한 버퍼 풀로도 알려진 Aurora 유지 가능 페이지 캐시는 멀티 마스터 클러스터에서 지원되지 않습니다.
- 멀티 마스터 클러스터는 연결에 대해 어떤 로드 밸런싱도 수행하지 않습니다. 애플리케이션은 여러 DB 인스턴스 엔드포인트로 읽기 및 쓰기 작업을 분산하기 위해 자체 연결 관리 로직을 구현해야 합니다. 일반적으로 BYOS(Bring-Your-Own-Shard) 애플리케이션에는 각 샤프드를 특정 연결에 매핑하는 로직이 이미 있습니다. 애플리케이션에서 연결 관리 로직을 적용하는 방법은 [멀티 마스터 클러스터의 연결 관리 \(p. 598\)](#) 단원을 참조하십시오.
- 멀티 마스터 클러스터에서는 DB 인스턴스 간의 조정 시 약간의 처리 및 네트워크 오버헤드가 발생합니다. 이러한 오버헤드는 쓰기 집약적 애플리케이션과 읽기 집약적 애플리케이션에서 다음과 같은 결과를 초래 합니다.
 - 처리량 장점은 여러 쓰기 작업이 동시에 이루어지는 바쁜 클러스터에서 가장 두드러집니다. 많은 경우에 단일 기본 인스턴스를 가진 기존의 Aurora 클러스터는 한 클러스터에 대한 쓰기 트래픽을 처리할 수 있습니다. 이러한 경우에 멀티 마스터 클러스터의 장점은 성능보다는 고가용성에서 대부분 나타납니다.
 - 단일 쿼리 성능은 보통 이에 상응하는 단일 마스터 클러스터에서보다 낮습니다.
- 단일 마스터 클러스터에서 생성된 스냅샷을 가져와서 멀티 마스터 클러스터에서 복원할 수 없으며, 그 반대도 불가능합니다. 대신에 한 종류의 클러스터에서 다른 종류의 클러스터로 모든 데이터를 전송하려면 AWS Database Migration Service(AWS DMS) 같은 명령이나 mysqldump 명령을 통해 만들어진 로직 덤프를 사용합니다.
- 멀티 마스터 클러스터에서는 병렬 쿼리, Aurora Serverless 또는 글로벌 데이터베이스 기능을 사용할 수 없습니다.

멀티 마스터 측면은 클러스터에서 영구적으로 적용되는 옵션입니다. 멀티 마스터 클러스터와 다른 종류의 클러스터(예: Aurora Serverless 또는 병렬 쿼리) 사이에서 기존 Aurora 클러스터를 전환할 수 없습니다.

- ZDP(Zero-Downtime Patching) 및 ZDR(Zero-Downtime Restart) 기능은 멀티 마스터 클러스터에서 사용할 수 없습니다.
- AWS Lambda, Amazon S3 및 AWS Identity and Access Management 같은 다른 AWS 서비스와의 통합은 멀티 마스터 클러스터에서 사용할 수 없습니다.
- Performance Insights 기능은 멀티 마스터 클러스터에서 사용할 수 없습니다.
- 멀티 마스터 클러스터를 복제할 수 없습니다.
- 멀티 마스터 클러스터에서 역추적 기능을 사용할 수 없습니다.

데이터베이스 엔진 제한 사항

다음과 같은 제한 사항은 멀티 마스터 클러스터에서 사용할 수 있는 데이터베이스 엔진에 적용되고 있습니다.

- 멀티 마스터 클러스터와의 이진 로그(binlog) 복제를 수행할 수 없습니다. 이러한 제한으로 인해 멀티 마스터 클러스터에서는 전역 트랜잭션 ID(GTID) 복제를 사용할 수 없습니다.
- 이벤트 스케줄러는 멀티 마스터 클러스터에서 사용할 수 없습니다.
- 해시 조인 최적화는 멀티 마스터 클러스터에서 지원되지 않습니다.
- 쿼리 캐시는 멀티 마스터 클러스터에서 사용할 수 없습니다.
- 멀티 마스터 클러스터에서 특정한 SQL 언어 기능을 사용할 수 없습니다. SQL 차이점에 대한 전체 목록과 이러한 제한을 극복하기 위해 SQL 코드를 적용하는 방법은 [멀티 마스터 클러스터를 위한 SQL 고려 사항 \(p. 597\)](#) 단원을 참조하십시오.

Aurora 멀티 마스터 클러스터 생성

Aurora 클러스터를 생성하는 시점에 멀티 마스터 아키텍처와 단일 마스터 아키텍처 중에서 선택을 합니다. 다음 절차는 멀티 마스터 아키텍처를 선택해야 하는 경우를 보여줍니다. 이전에 생성된 Aurora 클러스터가 없는 경우에는 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#)의 일반 절차를 확인할 수 있습니다.

콘솔

AWS Management 콘솔에서 Aurora 멀티 마스터 클러스터를 생성하려면 다음과 같이 선택을 합니다. 첫 번째 화면에서 Aurora 클러스터를 선택합니다.

Amazon RDS > Databases > Create database

Create database

Database settings



Quick create [Info](#)

Provides the fastest way to get started with your database. You can modify

Engine options

Amazon Aurora

Amazon
Aurora

MySQL



PostgreSQL



Oracle

ORACLE®

또한 MySQL 5.6 호환성 및 위치 선택 리전:

Edition

- Amazon Aurora with MySQL compatibility
- Amazon Aurora with PostgreSQL compatibility

DB engine version [Info](#)

Aurora MySQL 1.21.0 (Compatible with MySQL 5.6)

Select engine version Aurora MySQL 1.21.0 (Compatible with MySQL 5.6) to use create parallel query, multiple writers, serverless, or global databases.

Database location

Regional [Info](#)

You provision your Aurora database in a single region.

Global [Info](#)

You can provision your Aurora database in multiple regions. Writes in the primary typical latency of <1 sec to secondary regions.

두 번째 화면에서 Database features(데이터베이스 기능) 아래의 Multiple writers(다중 라이터)를 선택합니다.

Database features

One writer and multiple readers

The readers connect to the same storage volume as the writer instance and supports only read operations. [Info](#)

Multiple writers

Supports read and write operations, and performs all of the data modifications to the cluster volume. [Info](#)

One writer

You provi...
Aurora im...
pushing p...

Serverless

You speci...
resources
on databa...

클러스터의 다른 설정값을 입력합니다. 절차에서 이 부분은 [DB 클러스터 생성 \(p. 96\)](#)에서 Aurora 클러스터를 생성하기 위한 일반 지침과 동일합니다.

멀티 마스터 클러스터를 생성한 후에는 다음과 같은 [DB 클러스터에 Aurora 복제본 추가 \(p. 208\)](#)의 절차에 따라 두 개의 DB 인스턴스를 추가합니다. 멀티 마스터 클러스터 내의 모든 DB 인스턴스에서는 동일한 AWS 인스턴스 클래스를 사용합니다.

멀티 마스터 클러스터 및 관련 DB 인스턴스를 생성한 후에는 AWS Management 콘솔 데이터베이스 페이지에 다음과 같이 클러스터가 나타납니다. 모든 DB 인스턴스는 Writer(라이터) 역할을 보여줍니다.

The screenshot shows the 'Databases' section of the Amazon RDS console. At the top, there is a search bar labeled 'Filter databases'. Below it, a table lists database instances. The columns are 'DB Name', 'Role', and 'CPU'. There are three instances listed under a single cluster:

DB Name	Role	CPU
db-multi-master	Cluster	-
instance-name1	Writer	6.72
instance-name2	Writer	6.72

AWS CLI

AWS CLI에서 멀티 마스터 클러스터를 생성하려면 `create-db-cluster` AWS CLI 명령을 실행하고 옵션 `--engine_mode=multimaster`를 포함시킵니다.

다음 명령은 멀티 마스터 복제를 통해 Aurora 클러스터를 생성하기 위한 구문을 보여줍니다. Aurora 클러스터를 생성하기 위한 일반적인 절차는 [DB 클러스터 생성 \(p. 96\)](#) 단원을 참조하십시오.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora \
--engine-version 5.6.10a --master-username user-name --master-user-password password \
--db-subnet-group-name my_subnet_group --vpc-security-group-ids my_vpc_id \
--engine-mode multimaster
```

Windows의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-cluster --engine aurora ^
--engine-version 5.6.10a --master-username user-name --master-user-password password ^
--db-subnet-group-name my_subnet_group --vpc-security-group-ids my_vpc_id ^
--engine-mode multimaster
```

멀티 마스터 클러스터를 생성한 후에는 다음과 같은 [DB 클러스터에 Aurora 복제본 추가 \(p. 208\)](#)의 절차에 따라 두 번째 DB 인스턴스를 추가합니다. 멀티 마스터 클러스터 내의 모든 DB 인스턴스에서는 동일한 AWS 인스턴스 클래스를 사용합니다.

RDS API

RDS API를 이용해 멀티 마스터 클러스터를 생성하려면 [CreateDBCluster](#) 작업을 실행합니다. `EngineMode` 파라미터에 값 `multimaster`를 지정합니다. Aurora 클러스터를 생성하기 위한 일반적인 절차는 [DB 클러스터 생성 \(p. 96\)](#) 단원을 참조하십시오.

멀티 마스터 클러스터를 생성한 후에는 다음과 같은 [DB 클러스터에 Aurora 복제본 추가 \(p. 208\)](#)의 절차에 따라 두 개의 DB 인스턴스를 추가합니다. 멀티 마스터 클러스터 내의 모든 DB 인스턴스에서는 동일한 AWS 인스턴스 클래스를 사용합니다.

멀티 마스터 클러스터에 DB 인스턴스 추가

멀티 마스터 클러스터가 장점을 발휘하려면 하나 이상의 DB 인스턴스가 필요합니다. [DB 클러스터에 Aurora 복제본 추가 \(p. 208\)](#)의 절차를 사용하여 나중에 또 다른 DB 인스턴스를 생성할 수 있습니다. 멀티 마스터 클러스터의 차이점이라면 새 DB 인스턴스가 읽기 전용 Aurora 복제본 대신에 읽기-쓰기 기능을 가진다는 사실입니다. 멀티 마스터 클러스터 내의 모든 DB 인스턴스에서는 동일한 AWS 인스턴스 클래스를 사용합니다.

Aurora 멀티 마스터 클러스터 관리

다른 종류의 Aurora 클러스터에서와 동일한 방식으로 Aurora 멀티 마스터 클러스터에서 대부분의 관리 및 운영 작업을 수행합니다. 다음 단원에서는 운영 및 관리 측면에서 멀티 마스터 클러스터의 차이점과 고유한 특징에 대해 설명합니다.

주제

- [Aurora 멀티 마스터 클러스터 모니터링 \(p. 594\)](#)
- [멀티 마스터 클러스터의 데이터 수집 성능 \(p. 594\)](#)
- [멀티 마스터 클러스터에서 데이터 가져오기 \(p. 595\)](#)
- [Aurora 멀티 마스터 클러스터를 위한 고가용성 고려 사항 \(p. 595\)](#)
- [멀티 마스터 클러스터와 다른 클러스터 간의 복제 \(p. 596\)](#)
- [멀티 마스터 클러스터 업그레이드 \(p. 596\)](#)

Aurora 멀티 마스터 클러스터 모니터링

MySQL 및 Aurora 단일 마스터 클러스터에서 지원되는 모니터링 및 진단 기능은 대부분 멀티 마스터 클러스터에서도 지원됩니다.

- MySQL 오류 로그, 일반 로그 및 느린 쿼리 로그.
- SHOW 명령, 상태 변수, InnoDB 실행 시간 상태 테이블 같이 MySQL에 내장된 진단 기능.
- MySQL 성능 스키마.
- 고급 감사.
- CloudWatch 지표.
- 확장 모니터링.

Aurora 멀티 마스터 클러스터는 다음과 같은 모니터링 기능을 현재 지원하지 않고 있습니다.

- Performance Insights.

멀티 마스터 클러스터의 데이터 수집 성능

멀티 마스터 클러스터에서 DML 작업을 위한 모범 사례 중 하나는 트랜잭션을 작고 간결하게 유지하는 것입니다. 또한 특정 테이블 또는 데이터베이스에 대한 쓰기 작업을 특정한 DB 인스턴스로 라우팅하는 것입니다.

대량 가져오기를 수행하려면 트랜잭션 크기에 대한 지침을 완화해야 할 수도 있습니다. 그러나 쓰기 충돌의 발생 가능성을 최소화하기 위해 여전히 쓰기 작업을 분산할 수 있습니다.

대량 가져오기에서 쓰기 워크로드를 분산하려면

1. 스키마에서 각 데이터베이스, 테이블 또는 기타 객체에 대해 별도의 mysqldump 명령을 실행합니다. 덤픽 중인 객체를 반영하는 이름을 가진 파일에 각 mysqldump의 결과를 저장합니다. 대신에 mydumper 같이 여러 개의 테이블을 병렬로 자동 덤픽할 수 있는 전문적인 덤프 및 가져오기 도구를 사용할 수도 있습니다.
2. 각 데이터 파일마다 별도의 mysql 세션을 실행하여 해당되는 스키마 객체를 처리하는 적절한 인스턴스 엔드포인트를 연결합니다. 다시 말하지만 이 대신에 myloader 같은 전문적인 병렬 가져오기 명령을 사용할 수도 있습니다.
3. 멀티 마스터 클러스터에서는 한 세션이 끝나기를 기다렸다가 다음 세션을 시작하는 것이 아니라 DB 인스턴스 전반에 걸쳐 병렬로 가져오기 세션을 실행합니다.

다음 기법을 사용하여 Aurora 멀티 마스터 클러스터로 데이터를 가져올 수 있습니다.

- 명령문이 Aurora에서 지원되지 않는 어떤 기능도 사용하지 않는 경우, 다른 MySQL 호환 서버에서 Aurora 멀티 마스터 클러스터로 논리적(SQL 형식) 덤프를 가져올 수 있습니다. 예를 들어 MySQL 전체 텍스트 검색(FTS) 인덱스를 포함하는 테이블에서의 논리적 덤프는 작동하지 않습니다. FTS 기능이 멀티 마스터 클러스터에서 지원되지 않기 때문입니다.
- 데이터를 Aurora 멀티 마스터 클러스터로 마이그레이션하기 위해 DMS 같은 관리형 서비스를 사용할 수 있습니다.
- MySQL과 호환되지 않는 서버에서 Aurora 멀티 마스터 클러스터로 마이그레이션을 하려면 이기종 Aurora 마이그레이션을 위한 기존 지침을 따르십시오.
- Aurora 멀티 마스터 클러스터는 SQL 형식으로 MySQL과 호환되는 논리적 덤프를 만들 수 있습니다. 이러한 형식을 이해할 수 있는 모든 마이그레이션 도구(예: AWS DMS)는 Aurora 멀티 마스터 클러스터에서 나온 데이터 덤프를 사용할 수 있습니다.
- Aurora은 멀티 마스터 클러스터에서 binlog 마스터 또는 작업자로서 이진 로깅을 지원하지 않습니다. 멀티 마스터 클러스터에서는 binlog 기반의 CDC 도구를 사용할 수 없습니다.
- MySQL과 호환되지 않는 서버에서 마이그레이션을 할 때는 AWS DMS의 연속 변경 데이터 캡처(CDC) 기능을 사용하여 멀티 마스터 클러스터를 복제할 수 있습니다. 이러한 유형의 복제에서는 SQL 문을 대상 클러스터로 전송하기 때문에 binlog 복제에 대한 제한이 적용되지 않습니다.

마이그레이션 기법 및 권장 사항에 대한 자세한 내용은 [Amazon Aurora 마이그레이션 핸드북](#) AWS 백서를 참조하십시오. 핸드북에 나와 있는 마이그레이션 방법 중 몇 가지는 Aurora 멀티 마스터 클러스터에 적용되지 않을 수 있지만, 이 백서는 Aurora 마이그레이션이라는 주제에 대한 전반적인 지식을 제공하는 중요한 소스입니다.

멀티 마스터 클러스터에서 데이터 가져오기

멀티 마스터 클러스터의 스냅샷을 저장하고 이를 또 다른 멀티 마스터 클러스터에 복원할 수 있습니다. 현재로서는 멀티 마스터 클러스터 스냅샷을 단일 마스터 클러스터로 복원할 수 없습니다.

멀티 마스터 클러스터에서 단일 마스터 클러스터로 데이터를 마이그레이션하려면 mysqldump 같은 도구를 통해 논리적 덤프를 사용해 복원합니다.

멀티 마스터 클러스터는 이진 로그 복제를 위한 소스 또는 대상으로 사용할 수 없습니다.

Aurora 멀티 마스터 클러스터를 위한 고가용성 고려 사항

Aurora 멀티 마스터 클러스터에서 DB 인스턴스는 다른 인스턴스의 재시작을 초래하지 않고도 다시 시작할 수 있습니다. 따라서 Aurora 단일 마스터 클러스터에서보다 읽기-쓰기 및 읽기 전용 연결에서 더 뛰어난 가용성을 제공합니다. 이러한 가용성 수준을 지속적인 가용성이라고 합니다. 멀티 마스터 클러스터에서는 라이터 DB 인스턴스에 문제가 생겼을 때 쓰기 가용성에 차질이 발생하는 일이 없습니다. 모든 클러스터 인스턴스에

서 쓰기가 가능하기 때문에 멀티 마스터 클러스터는 장애 조치 메커니즘을 사용하지 않습니다. 멀티 마스터 클러스터에서 DB 인스턴스에 문제가 발생하면 애플리케이션이 상태가 좋은 나머지 인스턴스로 워크로드를 리디렉션할 수 있습니다.

단일 마스터 클러스터에서 기본 인스턴스를 다시 시작하면 장애 조치 메커니즘이 새로운 기본 인스턴스를 승격할 때까지 쓰기 작업이 사용 불가 상태가 됩니다. 또한 클러스터의 모든 Aurora 복제본이 다시 시작되므로 읽기 전용 작업이 잠시 중지됩니다.

멀티 마스터 클러스터에서 애플리케이션의 가동 중지를 최소화하려면 AQL 수준에서 자주 상태 확인을 하십시오. 멀티 마스터 클러스터의 DB 인스턴스가 사용 불가 상태가 되면 예상되는 중단 시간과 워크로드에서 쓰기 작업의 긴급성에 따라 어떤 조치를 취할지 결정할 수 있습니다. 중단 시간이 짧고 쓰기 작업이 긴급하지 않다고 예상되면 DB 인스턴스가 복구되어 해당 DB 인스턴스에 의해 워크로드가 정상적으로 처리되는 등 작동이 재개될 때까지 기다릴 수 있습니다. 아니면 다른 DB 인스턴스로 해당 워크로드를 리디렉션할 수 있습니다. 기반 데이터는 클러스터의 모든 DB 인스턴스에서 항상 가용 상태를 유지합니다. 널리 분산된 Aurora 스토리지 볼륨은 드물게 전체 AZ에 영향을 미치는 장애가 발생한 경우에도 데이터의 연속적인 가용성을 유지합니다. 사용할 수 없는 DB 인스턴스에서 다른 인스턴스로 쓰기 작업을 전환하기 위한 타이밍을 결정할 때 고려 사항은 [활성 대기 구성으로 멀티 마스터 클러스터 사용 \(p. 608\)](#) 단원을 참조하십시오.

멀티 마스터 클러스터와 다른 클러스터 간의 복제

멀티 마스터 클러스터는 수신 또는 발신 이진 로그 복제를 지원하지 않습니다.

멀티 마스터 클러스터 업그레이드

Aurora 멀티 마스터 클러스터는 다른 종류의 Aurora 클러스터와 동일한 버전 넘버링 체계(메이저 및 마이너 버전 번호 포함)를 사용합니다. 하지만 Enable auto minor version upgrade(마이너 버전 자동 업그레이드 활성화) 설정은 멀티 마스터 클러스터에 적용되지 않습니다.

Aurora 멀티 마스터 클러스터를 업그레이드할 때 보통 업그레이드 절차를 통해 현재 버전에서 차기 버전으로 데이터베이스 엔진을 이동시킵니다. 버전 번호가 1보다 큰 증분값에 따라 증가하는 Aurora 버전으로 업그레이드를 하는 경우에는 업그레이드에서 다단계 접근 방식을 사용합니다. 각 DB 인스턴스가 차기 버전으로 업그레이드되고, 차기 버전은 그 다음 차기 버전으로 업그레이드 되면서 지정된 업그레이드 버전에 도달할 때 까지 이 과정이 계속됩니다.

이러한 접근 방식은 이전 버전과 신규 버전 간에 이전 버전과 호환되지 않는 변경이 있는지 여부에 따라 달라집니다. 예를 들어 시스템 스키마에 대한 업데이트는 이전 버전과 호환되지 않는 변경으로 간주됩니다. 릴리스 정보를 참조하여 특정 버전에 이전 버전과 호환되지 않는 변경이 포함되어 있는지 여부를 확인할 수 있습니다.

이전 버전과 신규 버전 간에 이전 버전과 호환되지 않는 변경이 없으면 각 DB 인스턴스가 개별적으로 업그레이드 및 재시작됩니다. 업그레이드가 시간차를 두고 이루어지기 때문에 전체 클러스터에서 어떤 가동 중단도 발생하지 않습니다. 업그레이드 프로세스 동안 언제든 최소 하나의 DB 인스턴스를 사용할 수 있습니다.

이전 버전과 신규 버전 간에 이전 버전과 호환되지 않는 변경이 있는 경우에는 Aurora이 오프라인 모드에서 업그레이드를 수행합니다. 모든 클러스터 노드가 동시에 업그레이드 및 재시작됩니다. 이전 버전의 엔진이 신규 버전의 시스템 테이블로 쓰기 작업을 하지 못하도록 하기 위해 클러스터에서 약간의 가동 중지가 발생할 수 있습니다.

제로 가동 중지 패치 적용(ZDP)은 현재 Aurora 멀티 마스터 클러스터에서 지원되지 않고 있습니다.

Aurora 멀티 마스터 클러스터를 위한 애플리케이션 고려 사항

아래에서는 멀티 마스터 클러스터와 단일 마스터 클러스터 간의 기능 지원 또는 동작의 차이로 인해 애플리케이션에서 필요할 수 있는 변경 작업을 확인할 수 있습니다.

주제

- [멀티 마스터 클러스터를 위한 SQL 고려 사항 \(p. 597\)](#)

- 멀티 마스터 클러스터의 연결 관리 (p. 598)
- 멀티 마스터 클러스터의 일관성 모델 (p. 599)
- 멀티 마스터 클러스터 및 트랜잭션 (p. 599)
- 멀티 마스터 클러스터에서의 쓰기 충돌 및 교착 상태 (p. 600)
- 멀티 마스터 클러스터 및 쓰기 잠금 (p. 601)
- 멀티 마스터 클러스터에서 DDL 작업 수행 (p. 601)
- 자동 증분 열 사용 (p. 603)
- 멀티 마스터 클러스터 기능 참조 (p. 603)

멀티 마스터 클러스터를 위한 SQL 고려 사항

다음은 멀티 마스터 클러스터에서 사용할 수 있는 SQL 언어 기능에 적용되는 중요 제한 사항입니다.

- 멀티 마스터 클러스터에서는 행 레이아웃을 바꾸는 특정한 설정 또는 열 유형을 사용할 수 없습니다. `innodb_large_prefix` 구성 옵션을 사용할 수 없습니다. `MEDIUMTEXT`, `MEDIUMBLOB`, `LONGTEXT` 또는 `LONGBLOB` 같은 열 구성을 사용할 수 없습니다.
- 멀티 마스터 클러스터에서 외래 키 열이 포함된 CASCADE 절을 사용할 수 없습니다.
- 멀티 마스터 클러스터에는 전체 텍스트 검색(FTS) 인덱스를 가진 어떤 테이블도 포함시킬 수 없습니다. 이러한 테이블은 멀티 마스터 클러스터에서 생성 또는 가져오기가 불가능합니다.
- DDL은 멀티 마스터 클러스터와 단일 마스터 클러스터에서 다르게 작동합니다. 예를 들어 빠른 DDL 메커니즘은 멀티 마스터 클러스터에서는 사용할 수 없습니다. 테이블에서 DDL 작업이 이루어지고 있는 동안 멀티 마스터 클러스터에서 테이블로 쓰기 작업이 불가능합니다. DDL 차이점에 대한 자세한 내용은 [멀티 마스터 클러스터에서 DDL 작업 수행 \(p. 601\)](#) 단원을 참조하십시오.
- 멀티 마스터 클러스터에서 `SERIALIZABLE` 트랜잭션 격리 수준을 사용할 수 없습니다. Aurora 단일 마스터 클러스터에서는 기본 인스턴스에서 이러한 격리 수준을 사용할 수 있습니다.
- 자동 증분 열은 `auto_increment_increment` 및 `auto_increment_offset` 파라미터를 사용해 처리됩니다. 파라미터 값은 미리 결정되며 구성이 불가능합니다. 파라미터 `auto_increment_increment`은 Aurora 클러스터의 최대 인스턴스 수인 16으로 설정됩니다. 하지만 멀티 마스터 클러스터는 현재 DB 인스턴스의 수를 더 낮은 한도를 두고 있습니다. 세부 정보는 [자동 증분 열 사용 \(p. 603\)](#) 단원을 참조하십시오.

Aurora 멀티 마스터 클러스터에서 애플리케이션을 적용할 때는 마이그레이션 때와 동일한 방식으로 접근하십시오. 특정한 SQL 기능의 사용을 중단하고 다른 SQL 기능에 맞게 애플리케이션 로직을 변경해야 할 수 있습니다.

- `CREATE TABLE` 문에서 `MEDIUMTEXT`, `MEDIUMBLOB`, `LONGTEXT` 또는 `LONGBLOB`으로 정의된 모든 열을 오프 페이지 스토리지가 필요하지 않은 데 짧은 유형으로 변경합니다.
- `CREATE TABLE` 문에서 모든 외래 키 선언에서 CASCADE 절을 제거합니다. `INSERT` 또는 `DELETE` 문을 통해 CASCADE 효과를 에뮬레이션해야 할 경우에는 애플리케이션 로직을 추가합니다.
- 사용되고 있는 모든 InnoDB 전체 텍스트 검색(FTS) 인덱스를 제거합니다. `SELECT` 문의 `MATCH()` 작업에 대한 소스 코드와 DDL 문의 `FULLTEXT` 키워드를 확인합니다. `INFORMATION_SCHEMA.INNODB_SYS_TABLES` 시스템 테이블에서 나온 모든 테이블 이름에 문자열 `FTS_`이 포함되어 있는지 확인합니다.
- 애플리케이션에서 `CREATE TABLE` 및 `DROP TABLE` 같은 DDL 작업의 횟수를 확인합니다. DDL 작업은 멀티 마스터 클러스터에서 오버헤드가 더 크므로 작은 DDL 문이 다수 실행되는 것을 방지합니다. 예를 들어 필요한 테이블을 미리 생성할 수 있는 기회를 모색합니다. 멀티 마스터 클러스터에서의 DDL 차이에 대한 자세한 내용은 [멀티 마스터 클러스터에서 DDL 작업 수행 \(p. 601\)](#) 단원을 참조하십시오.
- 자동 증분 열의 사용을 검토합니다. 멀티 마스터 클러스터는 자동 증분 열의 값 순서가 다른 종류의 Aurora 클러스터와 다릅니다. DDL 문의 `AUTO_INCREMENT` 키워드, `SELECT` 문의 함수 이름 `last_insert_id()`, 사용자 지정 구성 설정의 이름 `innodb_autoinc_lock_mode`를 확인합니다. 차이점과 이를 처리할 수 있는 방법에 대한 자세한 내용은 [자동 증분 열 사용 \(p. 603\)](#) 단원을 참조하십시오.

- `SERIALIZABLE` 키워드에서 코드를 확인합니다. 멀티 마스터 클러스터에서 이러한 트랜잭션 격리 수준을 사용할 수 없습니다.

멀티 마스터 클러스터의 연결 관리

멀티 마스터 클러스터의 연결에서 중요하게 고려할 것은 사용 가능한 DNS 엔드포인트의 수와 유형입니다. 멀티 마스터 클러스터에서는 다른 종류의 Aurora 클러스터에서는 거의 사용하지 않는 인스턴스 엔드포인트를 종종 사용합니다.

Aurora 멀티 마스터 클러스터에는 다음과 같은 종류의 엔드포인트가 있습니다.

클러스터 엔드포인트

이 유형의 엔드포인트는 항상 읽기-쓰기 기능을 갖춘 DB 인스턴스를 가리킵니다. 각 멀티 마스터 클러스터마다 하나의 클러스터 엔드포인트가 있습니다.

멀티 마스터 클러스터의 애플리케이션에는 보통 특정 DB 인스턴스에 대한 연결을 관리하는 로직이 포함되어 있기 때문에 이 엔드포인트를 사용해야 할 일은 거의 없습니다. 이 엔드포인트는 관리 작업을 위해 멀티 마스터 클러스터를 연결할 때 가장 유용합니다.

또한 클러스터에서 DB 인스턴스의 상태를 모를 때 클러스터 토플로지를 검토하기 위해 이 엔드포인트를 연결할 수도 있습니다. 자세한 절차는 [클러스터 토플로지 설명 \(p. 605\)](#) 단원을 참조하십시오.

DB 인스턴스 엔드포인트

이 유형의 엔드포인트는 특정한 명명 DB 인스턴스를 연결합니다. Aurora 멀티 마스터 클러스터에서 애플리케이션은 일반적으로 거의 모든 연결에서 DB 인스턴스 엔드포인트를 사용합니다. 샤드와 클러스터의 DB 인스턴스 간의 매핑에 따라 각 SQL 문에서 사용할 DB 인스턴스를 결정합니다. 각 DB 인스턴스마다 이러한 엔드포인트가 하나씩 있습니다. 따라서 멀티 마스터 클러스터에는 이러한 엔드포인트가 하나 이상 있고, 멀티 마스터 클러스터에서 DB 인스턴스가 추가 또는 제거됨에 따라 이 숫자가 바뀝니다.

DB 인스턴스 엔드포인트의 사용 방법은 단일 마스터 클러스터와 멀티 마스터 클러스터 간에 차이가 있습니다. 단일 마스터 클러스터에서는 보통 이러한 엔드포인트를 자주 사용하지 않습니다.

사용자 지정 엔드포인트

이러한 유형의 엔드포인트는 선택 사항입니다. 특정한 목적으로 DB 인스턴스를 그룹화하기 위해 사용자 지정 엔드포인트를 하나 이상 생성할 수 있습니다. 이 엔드포인트에 연결하면 Aurora가 매번 다른 DB 인스턴스의 IP 주소를 반환합니다. 멀티 마스터 클러스터에서는 보통 사용자 지정 엔드포인트를 사용하여 일기 작업에서 주로 사용할 DB 인스턴스 집합을 지정합니다. 멀티 마스터 클러스터에서 쓰기 작업을 로드 밸런싱할 때는 사용자 지정 엔드포인트를 사용하지 않는 것이 좋습니다. 쓰기 충돌이 발생할 가능성성이 높아지기 때문입니다.

멀티 마스터 클러스터에는 리더 엔드포인트가 없습니다. 가능한 동일한 테이블에 대해 정상적으로 쓰기 작업을 하는 동일한 DB 인스턴스 엔드포인트를 사용하여 `SELECT` 쿼리를 실행합니다. 이렇게 하면 버퍼풀에서 캐시된 데이터를 보다 효과적으로 사용하고, 클러스터 내의 복제 지연으로 인해 데이터의 기한이 경과되는 잠재적 문제를 방지할 수 있습니다. 동일한 테이블로 쓰기 작업을 수행하는 동일한 DB 인스턴스에서 `SELECT` 문의 위치를 찾을 수 없고 특정 쿼리에서 엄격한 쓰기 후 읽기 보장이 요구되는 경우에는 [멀티 마스터 클러스터의 일관성 모델 \(p. 599\)](#)에 설명되어 있는 전역 쓰기 후 읽기(GRAW) 메커니즘을 사용해 이러한 쿼리를 실행하는 것이 좋습니다.

Aurora 및 MySQL 연결 관리의 일반적인 모범 사례는 [Amazon Aurora 마이그레이션 핸드북](#) AWS 백서를 참조하십시오.

멀티 마스터 클러스터에서 읽기 전용 DB 인스턴스를 에뮬레이션하는 방법은 [인스턴스 읽기 전용 모드 사용 \(p. 605\)](#) 단원을 참조하십시오.

Aurora 멀티 마스터 클러스터에서 사용자 지정 DNS 엔드포인트를 생성하고 드라이버 및 커넥터를 설계할 때는 여기 나온 지침을 따르십시오.

- DDL, DML 및 DCL 문에서는 라운드 로빈 또는 랜덤 방식으로 작동하는 엔드포인트 또는 연결 라우팅 기법을 사용하지 않도록 합니다.
- 이러한 트랜잭션이 클러스터의 다른 쓰기 트래픽과 충돌하지 않을 것이 확실해질 때까지는 장기 실행 쓰기 쿼리와 장기 쓰기 트랜잭션을 피합니다.
- 자동 커밋된 트랜잭션을 사용하는 것이 좋습니다. 가능한 전역 또는 세션 수준에서 autocommit=0 설정을 사용을 피하십시오. 프로그래밍 언어에서 데이터베이스 커넥터 또는 데이터베이스 프레임워크를 사용할 때는 이러한 커넥터 또는 프레임워크를 사용하는 애플리케이션에서 autocommit가 활성화되어 있는지 확인합니다. 필요할 경우, 코드 전반의 논리적 지점에서 COMMIT 문을 추가하여 트랜잭션을 간략하게 합니다.
- 전역 읽기 일관성 또는 쓰기 후 읽기 보장이 요구되는 경우에는 [멀티 마스터 클러스터의 일관성 모델 \(p. 599\)](#)에 설명되어 있는 전역 쓰기 후 읽기(GRAW)에 대한 권장 사항을 따르십시오.
- 가능한 이러한 클러스터 엔드포인트는 DDL 및 DCL 문에서 사용하십시오. 이러한 클러스터 엔드포인트는 개별 DB 인스턴스의 호스트 이름에 대한 종속성을 최소화하는 데 도움이 됩니다. DML 문에서와 같이 테이블 또는 데이터베이스에 따라 DDL 및 DCL 문을 나눌 필요가 없습니다.

멀티 마스터 클러스터의 일관성 모델

Aurora 멀티 마스터 클러스터는 세션 수준에서 구성 가능한 전역 쓰기 후 읽기(GRAW) 모드를 지원합니다. 이 설정에서는 각 쿼리에 대해 일관된 읽기 보기 만들기 위해 추가적인 동기화가 필요합니다. 덕분에 쿼리는 항상 최신 데이터를 보게 됩니다. 기본적으로 멀티 마스터 클러스터에서는 복제 지연이 있기 때문에 데이터가 업데이트된 후 몇 밀리초 동안에는 DB 인스턴스가 이전 데이터를 보게 될 수 있습니다. 그 결과로 쿼리가 기다려야 하는 경우가 발생하더라도 애플리케이션이 다른 DB 인스턴스가 변경한 최신 데이터를 보고 있는 쿼리에 의존하는 있을 때는 이 기능을 사용합니다.

Note

동일한 DB 인스턴스를 사용하여 데이터에 대해 쓰기 후 읽기를 수행하는 경우에는 복제 지연이 쿼리 결과에 영향을 미치지 않습니다. 따라서 GRAW 기능은 서로 다른 DB 인스턴스를 통해 여러 개의 쓰기 작업을 동시에 발행하는 애플리케이션에 주로 적용됩니다.

GRAW 모드를 사용할 때는 기본적으로 모든 쿼리에 대해 이를 활성화해서는 안 됩니다. 전역적으로 일관된 읽기는 로컬 읽기보다 확연히 속도가 느립니다. 따라서 필요한 쿼리에서만 선택적으로 GRAW를 사용하십시오.

GRAW를 사용할 때는 다음을 반드시 고려하십시오.

- GRAW에는 클러스터 전반의 일관된 읽기 보기 구축하는 비용으로 인해 성능 오버헤드가 수반됩니다. 트랜잭션이 먼저 클러스터 전반에서 일관된 특정 시점을 결정한 다음, 복제가 이 시점을 확인해야 합니다. 전체 지연 시간은 워크로드에 따라 다르지만, 보통 수십 밀리초 범위 내에 있습니다.
- 트랜잭션 내에서 GRAW 모드를 변경할 수 없습니다.
- 명시적 트랜잭션 없이 GRAW를 사용하면 전역적으로 일관된 읽기 보기 구축하는 비용으로 인해 각 개별 쿼리에서 성능 오버헤드가 발생합니다.
- GRAW가 활성화되어 있으면 성능 패널티가 읽기와 쓰기에 모두 적용됩니다.
- 명시적 트랜잭션과 함께 GRAW를 사용하면 각 트랜잭션에 대해 전역적으로 일관된 읽기 보기 구축하는 비용으로 인한 오버헤드가 각 트랜잭션마다 한 번씩 적용됩니다(트랜잭션이 시작될 때). 트랜잭션에서 이후에 실행된 쿼리는 GRAW 없이 실행되는 경우처럼 속도가 빠릅니다. 여러 개의 연속 문이 동일한 읽기 기능을 사용할 수 있는 경우에는 전반적인 성능 개선을 위해 단일 트랜잭션에 이들을 래핑할 수 있습니다. 따라서 패널티가 쿼리마다 적용되는 것이 아니라 트랜잭션별로 한 번만 적용됩니다.

멀티 마스터 클러스터 및 트랜잭션

표준 Aurora MySQL 지침이 Aurora 멀티 마스터 클러스터에 적용됩니다. Aurora MySQL 데이터베이스 엔진은 수명이 짧은 SQL 문에 최적화되어 있습니다. 이러한 유형의 문은 보통 온라인 트랜잭션 처리(OLTP) 애플리케이션과 연관되어 있습니다.

특히 쓰기 트랜잭션을 가능한 짧게 만드십시오. 이렇게 하면 쓰기 충돌의 위험을 줄일 수 있습니다. 충돌 해결 메커니즘은 충돌이 드문 경우에 가장 잘 작동한다는 점에서 낙관적입니다. 하지만 충돌이 발생하면 막대한 오버헤드가 발생한다는 단점이 있습니다.

특정 워크로드에서는 대규모 트랜잭션이 유리합니다. 예를 들어 단일 문 트랜잭션이 아니라 멀티 기가바이트 트랜잭션을 사용하여 실행할 때 대량 데이터 가져오기의 속도가 훨씬 빨라집니다. 이러한 워크로드를 실행하는 동안 허용할 수 없는 수준의 충돌이 발생할 경우에는 다음 옵션을 고려하십시오.

- 트랜잭션 크기를 줄입니다.
- 배치 작업이 중복되어 다른 워크로드와 충돌을 유발하지 않도록 배치 작업을 다시 예약하거나 다시 배열합니다. 가능하다면 피크 이외 시간에 실행되도록 배치 작업을 다시 예약합니다.
- 충돌을 야기 중인 다른 트랜잭션과 동일한 라이터 인스턴스에서 실행되도록 배치 작업을 리팩터링합니다. 충돌 중인 트랜잭션이 동일한 인스턴스에서 실행될 때 트랜잭션 엔진이 행에 대한 액세스를 관리합니다. 이 경우, 스토리지 수준의 쓰기 충돌이 발생하지 않습니다.

멀티 마스터 클러스터에서의 쓰기 충돌 및 교착 상태

멀티 마스터 클러스터의 중요한 성능 측면 중 하나는 쓰기 충돌 횟수입니다. Aurora 스토리지 서브시스템에서 이러한 문제가 발생하면 애플리케이션이 교착 오류를 수신하고 교착 상태에 대한 일반적인 오류 처리를 수행합니다. Aurora는 이러한 충돌이 드문 경우일 때 가장 잘 작동하는 잠금 없는 낙관적 알고리즘을 사용합니다.

멀티 마스터 클러스터에서는 모든 DB 인스턴스가 공유 스토리지 볼륨에 쓰기 작업을 할 수 있습니다. 수정하는 모든 데이터 페이지에서 Aurora는 여러 사용 영역(AZ)으로 몇 개의 복사본을 자동 분산합니다. 여러 개의 DB 인스턴스가 매우 짧은 시간 내에 동일한 데이터 페이지를 수정하려고 시도할 때 쓰기 충돌이 발생할 수 있습니다. Aurora 스토리지 서브시스템은 변경 사항이 중복된 것을 탐지하고 쓰기 작업을 마무리하기 전에 충돌 해결 작업을 수행합니다.

Aurora는 16KiB로 크기가 고정되어 있는 물리적 데이터 페이지의 수준에서 쓰기 충돌을 탐지합니다. 따라서 충돌은 다른 행에 영향을 미치는 변경 작업에 대해서도 발생할 수 있습니다(동일한 데이터 페이지 내에 두 행이 모두 존재하는 경우).

충돌이 발생하면 정리 작업 시 DB 인스턴스 중 하나에서 변경을 취소하기 위한 추가 작업이 필요합니다. 애플리케이션 관점에서 보자면 충돌을 야기한 트랜잭션은 교착 상태에 봉착하고 Aurora는 전체 트랜잭션을 롤백합니다. 애플리케이션은 오류 코드 1213을 수신합니다.

트랜잭션의 실행을 취소하면 Aurora 스토리지 서브시스템에 변경 사항이 이미 적용된 다른 많은 데이터 페이지를 수정해야 할 수도 있습니다. 트랜잭션에서 얼마나 많은 데이터가 변경되었느냐에 따라 실행 취소에 막대한 오버헤드가 수반될 수 있습니다. 따라서 쓰기 충돌의 잠재적 가능성은 최소화하는 것이 Aurora 멀티 마스터 클러스터를 설계할 때 중요하게 고려해야 할 사항입니다.

일부 충돌은 개시한 변경 작업의 결과입니다. SQL 문, 트랜잭션 및 트랜잭션 롤백이 이러한 변경에 해당됩니다. 스키마 설계와 애플리케이션의 연결 관리 로직을 통해 이러한 유형의 충돌을 최소화할 수 있습니다.

다른 충돌들은 SQL 문과 내부 서버 스레드 모두에서 동시에 변경 작업이 수행될 때 발생합니다. 이러한 충돌은 파악하기 힘든 내부 서버 활동으로 인한 것이기 때문에 예측이 어렵습니다. 이러한 충돌을 야기하는 두 가지 중요한 내부 활동 유형으로는 가비지 수집(일명 제거)과 Aurora에서 자동으로 수행되는 트랜잭션 롤백이 있습니다. 예를 들어 Aurora는 충돌이 복구되는 동안, 또는 클라이언트 연결이 끊기는 경우에 자동으로 롤백을 수행합니다.

트랜잭션 롤백은 이미 수행된 페이지 변경을 물리적으로 되돌립니다. 롤백은 원래 트랜잭션과 똑같은 방식으로 페이지 변경 사항을 만듭니다. 롤백에는 시간이 걸리는데, 원래 트랜잭션 만큼의 시간이 여러 차례 필요할 수 있습니다. 롤백이 진행되는 동안 만들어진 변경 사항이 트랜잭션과 충돌할 수 있습니다.

가비지 수집은 Aurora MySQL 트랜잭션 엔진에서 사용하는 동시성 제어 방법인 다중 버전 동시성 제어(MVCC)를 통해 수행해야 합니다. MVCC에서는 데이터 변형으로 새로운 행 버전이 만들어지고, 데이터베이스 데이터에 대한 동시 액세스를 허용하면서도 트랜잭션 격리를 달성할 수 있도록 여러 버전의 행을 유지

합니다. 더 이상 필요가 없을 때 행 버전이 삭제됩니다(제거). 다시 말하지만, 제거 프로세스는 트랜잭션과 충돌 가능성이 있는 페이지 변경 사항을 만들어냅니다. 워크로드에 따라 데이터베이스에서 가비지 수집을 기다리는 변경 사항의 대기열인 제거 지연 시간이 발생할 수 있습니다. 지연 시간이 크게 증가하면 SQL 문의 제출을 중지한 경우라도 데이터베이스에서 제거 작업을 완료하는 데 상당한 시간이 소요될 수 있습니다.

내부 서버 스레드에서 쓰기 충돌이 발생하면 Aurora가 자동으로 재시도를 합니다. 반대로, 애플리케이션은 충돌이 발생한 모든 트랜잭션에서 재시도 로직을 처리해야 합니다.

동일한 DB 인스턴스에서 여러 개의 트랜잭션으로 인해 이러한 유형의 변경 중복이 발생할 때 Aurora는 표준 트랜잭션 동시성 규칙을 사용합니다. 예를 들어 동일한 DB 인스턴스의 두 트랜잭션이 동일한 행을 수정하면 이들 중 하나는 대기를하게 됩니다. 대기 시간이 구성된 제한 시간(innodb_lock_wait_timeout, 기본 50초)보다 길어지면 대기 중인 트랜잭션이 “Lock wait timeout exceeded(잠금 대기 제한 시간 초과)” 메시지와 함께 작동을 중단합니다.

멀티 마스터 클러스터 및 쓰기 잠금

Aurora 멀티 마스터 클러스터는 다음과 같은 형태의 쓰기 잠금을 지원합니다.

```
SELECT ... FOR UPDATE  
SELECT ... LOCK IN SHARE MODE
```

쓰기 잠금에 대한 자세한 내용은 [MySQL 참조 매뉴얼](#)을 참조하십시오.

쓰기 잠금 작업은 모든 노드에서 지원되지만, 잠금 범위는 명령이 실행된 노드에 국한됩니다. 한 라이터에서 쓰기 잠금을 수행했다고 해서 다른 라이터가 잠금 상태인 행을 액세스 또는 수정하는 것을 막을 수는 없습니다. 이러한 한계에도 불구하고 쓰기 잠금은 라이터 간에 엄격한 워크로드 범위 구분을 보장해야 하는 사용 사례(예: 샤딩된 데이터베이스 또는 멀티 테넌트 데이터베이스)에서 여전히 효과적일 수 있습니다.

다음 지침을 참고하십시오.

- 노드는 언제라도 지연 없이 즉시 자체 변경 사항을 확인할 수 있다는 것을 기억하십시오. 가능하다면 GRAW 요구 사항을 제거할 수 있도록 동일한 노드에 읽기 및 쓰기 작업을 배치할 수 있습니다.
- 전역적으로 일관된 결과와 함께 읽기 전용 쿼리를 실행해야 하는 경우에는 GRAW를 사용합니다.
- 읽기 전용 쿼리에서 전역적 일관성이 아니라 데이터 가시성이 중요한 경우에는 GRAW를 사용하거나 읽기 작업을 시작하기 전에 대기 시한을 두십시오. 예를 들어 단일 애플리케이션 스레드가 서로 다른 두 노드에 대해 연결 C1 및 C2를 유지할 수 있습니다. 이 애플리케이션은 C1에서는 쓰기 작업을, C2에서는 읽기 작업을 수행합니다. 이러한 경우에 애플리케이션은 GRAW를 사용하여 즉시 읽기 쿼리를 발행하거나 잠시 기다렸다가 읽기 쿼리를 발행할 수 있습니다. 대기 시간은 복제 지연(일반적으로 약 20–30ms)과 같거나 더 길어야 합니다.

쓰기 후 읽기 기능은 `aurora_mm_session_consistency_level` 세션 변수를 사용해 제어됩니다. 유효 값은 로컬 일관성 모드(기본)의 경우 `INSTANCE_RAW`, 클러스터 전반 일관성 모드의 경우 `REGIONAL_RAW`입니다.

멀티 마스터 클러스터에서 DDL 작업 수행

SQL 데이터 정의 언어(DDL) 문은 멀티 마스터 클러스터를 특별히 고려한 것입니다. 이들 문은 때로 기반 데이터의 대대적인 재구성을 야기합니다. 이러한 대규모의 변경은 공유 스토리지 볼륨의 많은 데이터 페이지에 영향을 미칠 수 있습니다. 테이블 및 기타 스키마 객체의 정의는 `INFORMATION_SCHEMA` 테이블에 보관됩니다. Aurora는 여러 개의 DDL 인스턴스가 동시에 DDL 문을 실행할 때 쓰기 충돌이 발생하는 일이 없도록 이러한 테이블에 대한 변경 사항을 특별한 방식으로 처리합니다.

DDL 문에서 Aurora는 클러스터의 특별 서버 프로세스로 실행을 자동 위임합니다. Aurora는 중앙에서 `INFORMATION_SCHEMA` 테이블을 변경하므로 이 메커니즘을 통해 DDL 문 간의 쓰기 충돌을 막을 수 있습니다.

DDL 작업은 해당 테이블에 대한 동시 쓰기를 금지합니다. 테이블에서 DDL 작업이 수행되는 동안 멀티 마스터 클러스터의 모든 DB 인스턴스들은 DDL 문이 완료될 때까지 해당 테이블에 대해 읽기 전용 액세스만 가능하도록 권한이 제한됩니다.

다음과 같은 DDL 활동은 Aurora 단일 마스터 클러스터와 멀티 마스터 클러스터에서 동일합니다.

- DB 인스턴스에서 DDL을 실행하면 다른 인스턴스들이 테이블을 사용해 적극적으로 연결을 종단하는 결과가 초래됩니다.
- MyISAM 또는 MEMORY 스토리지 엔진을 사용해 어떤 노드에서든 세션 수준의 임시 테이블을 생성할 수 있습니다.
- DB 인스턴스에 충분한 용량의 로컬 임시 스토리지가 없는 경우에는 규모가 엄청나게 큰 테이블에서 DDL 작업은 실패합니다.

멀티 마스터 클러스터에서는 다음과 같은 DDL 성능 고려 사항에 유의하십시오.

- 애플리케이션에서 짧은 DDL 문을 대량으로 발행하는 것을 피해야 합니다. 가능한 데이터베이스, 테이블, 파티션, 열 등을 미리 생성합니다. 복제 오버헤드는 일반적으로 속도가 매우 빠른 간단한 DDL 문에 엄청난 성능 오버헤드를 야기할 수 있습니다. 변경 사항이 클러스터의 모든 DB 인스턴스에 복제될 때까지 문 실행이 완료되지 않습니다. 예를 들어 멀티 마스터 클러스터는 빈 테이블을 생성하거나, 테이블을 삭제하거나, 많은 테이블이 포함된 스키마를 삭제할 때 다른 Aurora 클러스터보다 더 많은 시간이 걸립니다.

대규모의 DDL 작업을 수행해야 하는 경우, 여러 스레드를 통해 문을 병렬로 발행하여 네트워크 및 조정 오버헤드를 줄일 수 있습니다.

- DDL 문의 전체 시간에서 복제 지연이 차지하는 비중이 작기 때문에 길이가 긴 DDL 문은 영향을 덜 받습니다.
- 세션 수준의 임시 테이블에서 DDL의 성능은 Aurora 단일 마스터 및 멀티 마스터 클러스터에서 일주 비슷합니다. 임시 테이블에서 작업은 로컬로 수행되고, 동기식 복제 오버헤드가 발생하지 않습니다.

멀티 마스터 클러스터에서 Percona 온라인 스키마 사용

pt-online-schema-change 도구는 멀티 마스터 클러스터에서 작동합니다. 가장 비차단적인 방식으로 테이블 수정을 실행하는 데 우선 순위를 두는 경우에 이 도구가 유용할 수 있습니다. 그러나 스키마 변경 프로세스로 인한 쓰기 충돌 위험에 유의하십시오.

개괄적으로 살펴볼 때 pt-online-schema-change 도구는 다음과 같이 작동합니다.

1. 원하는 구조를 가진 빈 테이블을 새로 생성합니다.
2. 원래 테이블에서 DELETE, INSERT 및 UPDATE 트리거를 생성하여 새 테이블을 토대로 원래 테이블에 대한 모든 데이터 변경을 취소합니다.
3. 트리거를 사용해 이후의 테이블 변경을 자동으로 처리하면서 동시에 작은 청크를 사용하여 새 테이블로 기존 행을 이동시킵니다.
4. 모든 데이터가 이동되고 나면 트리거를 삭제하고 이름을 다시 명명하여 테이블을 전환합니다.

데이터가 새 테이블로 전달되는 동안 경합 지점이 발생할 수 있습니다. 처음 생성된 새 테이블은 완전히 비어 있기 때문에 잠금 핫 포인트가 될 수 있습니다. 다른 종류의 데이터베이스 시스템에서도 마찬가지입니다. 트리거는 동기식이기 때문에 핫 포인트에서의 영향이 쿼리로 다시 전파될 수 있습니다.

멀티 마스터 클러스터에서는 이러한 영향이 더 잘 눈에 보일 수 있습니다. 왜냐하면 새 테이블이 잠금 경합을 유발하는 것은 물론이고, 쓰기 충돌 가능성도 높아지 때문입니다. 처음에 테이블에는 매우 적은 수의 페이지가 포함되는데, 이는 곧 쓰기 작업이 로컬로 수행되어 충돌 가능성이 높다는 의미입니다. 테이블이 확장된 후에는 쓰기 작업이 분산되어 쓰기 충돌이 더 이상 문제가 되지 않아야 합니다.

멀티 마스터 클러스터에서 온라인 스키마 변경 도구를 사용할 수 있습니다. 그러나 보다 신중한 테스트가 필요할 수 있고, 이후 워크로드에 미치는 영향이 작업을 시작하고 첫 몇 분 동안에 조금 더 눈에 띌 수 있습니다.

자동 증분 열 사용

Aurora 멀티 마스터 클러스터는 기존의 구성 파라미터인 `auto_increment_increment`와 `auto_increment_offset`을 사용하여 자동 증분 열을 처리합니다. 자세한 내용은 [MySQL 참조 매뉴얼](#)을 참조하십시오.

파라미터 값은 미리 결정되며 변경이 불가능합니다. 즉, `auto_increment_increment` 파라미터는 모든 종류의 Aurora 클러스터에서 DB 인스턴스의 최대 값인 16으로 하드코딩되어 있습니다.

하드코딩 된 증가 설정으로 인해 단일 마스터 클러스터에서보다 자동 증분 값이 훨씬 신속하게 사용됩니다. 해당 테이블이 단일 DB 인스턴스에서만 수정되는 경우라 하더라도 마찬가지입니다. 최고의 결과를 위해서는 자동 증분 열에서 INT 대신에 BIGINT 데이터 유형을 항상 사용하십시오.

멀티 마스터 클러스터에서는 다음 속성을 가진 자동 증분 열을 허용하도록 애플리케이션 로직을 준비해야 합니다.

- 이 값들은 불연속적입니다.
- 빈 테이블에서는 값이 1부터 시작하지 않을 수 있습니다.
- 1보다 큰 증분값에 따라 값이 증가합니다.
- 단일 마스터 클러스터에서보다 훨씬 신속하고 대대적으로 값이 사용됩니다.

다음 예제는 멀티 마스터 클러스터의 자동 증분 값 순서가 예상과 어떻게 다를 수 있는지를 보여줍니다.

```
mysql> create table autoinc (id bigint not null auto_increment, s varchar(64), primary key (id));

mysql> insert into autoinc (s) values ('row 1'), ('row 2'), ('row 3');
Query OK, 3 rows affected (0.02 sec)

mysql> select * from autoinc order by id;
+----+-----+
| id | s    |
+----+-----+
|  2 | row 1 |
| 18 | row 2 |
| 34 | row 3 |
+----+-----+
3 rows in set (0.00 sec)
```

`AUTO_INCREMENT` 테이블 속성을 변경할 수 있습니다. 기본값이 아닌 값을 사용하는 것은 테이블에 이미 존재하는 어떤 기본 키 값보다 값이 큰 경우에만 주효합니다. 더 작은 값을 사용하여 테이블의 빈 간격을 채울 수는 없습니다. 이렇게 하면 변경 사항이 임시적으로만 적용되거나 전혀 적용되지 않습니다. 이러한 활동은 MySQL 5.6에서 유래한 것으로, Aurora 구현에는 해당되지 않습니다.

멀티 마스터 클러스터 기능 참조

아래에서 Aurora 멀티 마스터 클러스터에 해당되는 명령, 절차 및 상태 변수에 대한 빠른 참조를 확인할 수 있습니다.

쓰기 후 읽기 사용

쓰기 후 읽기 기능은 `aurora_mm_session_consistency_level` 세션 변수를 사용해 제어됩니다. 유효 값은 로컬 일관성 모드(기본)의 경우 `INSTANCE_RAW`, 클러스터 전반 일관성 모드의 경우 `REGIONAL_RAW`입니다.

예를 들면 다음과 같습니다.

```
mysql> select @@aurora_mm_session_consistency_level;
+-----+
| @@aurora_mm_session_consistency_level |
+-----+
| INSTANCE_RAW |
+-----+
1 row in set (0.01 sec)

mysql> set session aurora_mm_session_consistency_level = 'REGIONAL_RAW';
Query OK, 0 rows affected (0.00 sec)

mysql> select @@aurora_mm_session_consistency_level;
+-----+
| @@aurora_mm_session_consistency_level |
+-----+
| REGIONAL_RAW |
+-----+
1 row in set (0.03 sec)
```

DB 인스턴스 읽기-쓰기 모드 확인

멀티 마스터 클러스터에서는 모든 노드가 읽기-쓰기 모드에서 작동합니다. `innodb_read_only` 변수는 항상 0을 반환합니다. 다음 예제에서 알 수 있듯이 멀티 마스터 클러스터에서 DB 인스턴스에 연결할 때 DB 인스턴스는 읽기-쓰기 기능을 가지고 있다고 보고합니다.

```
$ mysql -h mysql -A -h multi-master-instance-1.example123.us-east-1.rds.amazonaws.com
mysql> select @@innodb_read_only;
+-----+
| @@innodb_read_only |
+-----+
|          0 |
+-----+
mysql> quit;
Bye

$ mysql -h mysql -A -h multi-master-instance-2.example123.us-east-1.rds.amazonaws.com
mysql> select @@innodb_read_only;
+-----+
| @@innodb_read_only |
+-----+
|          0 |
+-----+
```

노드 이름 및 역할 확인

`aurora_server_id` 상태 변수를 사용하여 현재 연결된 DB 인스턴스의 이름을 확인할 수 있습니다. 다음 예제에서 그 방법을 설명합니다.

```
mysql> select @@aurora_server_id;
+-----+
| @@aurora_server_id |
+-----+
| mmr-demo-test-mm-3-1 |
+-----+
1 row in set (0.00 sec)
```

멀티 마스터 클러스터에서 모든 DB 인스턴스에 대해 이 정보를 찾으려면 [클러스터 토플로지 설명 \(p. 605\)](#) 단원을 참조하십시오.

클러스터 토플로지 설명

information_schema.replica_host_status 테이블에서 선택하여 멀티 마스터 클러스터 토플로지를 설명할 수 있습니다. 멀티 마스터 클러스터와 단일 마스터 클러스터의 차이점은 다음과 같습니다.

- `has_primary` 열은 노드의 역할을 식별합니다. 멀티 마스터 클러스터에서 이 값은 모든 DDL 및 DCL 문을 처리하는 DB 인스턴스에 적용됩니다. Aurora는 이러한 요청을 멀티 마스터 클러스터의 DB 인스턴스 중 하나에 전달합니다.
- `replica_lag_in_milliseconds` 열은 모든 DB 인스턴스에 대한 복제 지연을 보고합니다.
- `last_reported_status` 열은 DB 인스턴스의 상태를 보고합니다. Online, Recovery 또는 Offline일 수 있습니다.

예를 들면 다음과 같습니다.

```
mysql> select server_id, has_primary, replica_lag_in_milliseconds, last_reported_status
    -> from information_schema.replica_host_status;
+-----+-----+-----+-----+
| server_id | has_primary | replica_lag_in_milliseconds | last_reported_status |
+-----+-----+-----+-----+
| mmr-demo-test-mm-3-1 | true | 37.302 | Online |
| mmr-demo-test-mm-3-2 | false | 39.907 | Online |
+-----+-----+-----+-----+
```

인스턴스 읽기 전용 모드 사용

Aurora 멀티 마스터 클러스터에서는 보통 관련 테이블에서 쓰기 작업을 수행하는 특정한 DB 인스턴스에 SELECT 문을 발행합니다. 이렇게 하면 복제 지연으로 인한 일관성 문제를 방지하고 버퍼풀에서 테이블 및 인덱스 데이터를 최대한 재사용할 수 있습니다.

여러 테이블에서 쿼리 집약적인 워크로드를 실행해야 하는 경우에는 멀티 마스터 클러스터 내에서 하나 이상의 DB 인스턴스를 읽기 전용으로 지정할 수 있습니다.

실행 시간에 전체 DB 인스턴스를 읽기 전용 모드로 전환하려면 `mysql.rds_set_read_only` 저장 절차를 호출하십시오.

```
mysql> select @@read_only;
+-----+
| @@read_only |
+-----+
| 0 |
+-----+
1 row in set (0.00 sec)
mysql> call mysql.rds_set_read_only(1);
Query OK, 0 rows affected (0.00 sec)
mysql> select @@read_only;
+-----+
| @@read_only |
+-----+
| 1 |
+-----+
1 row in set (0.00 sec)
mysql> call mysql.rds_set_read_only(0);
Query OK, 0 rows affected (0.00 sec)
mysql> select @@read_only;
+-----+
| @@read_only |
+-----+
```

```
|          0 |
+-----+
1 row in set (0.00 sec)
```

저장 절차를 호출하는 것은 `SET GLOBAL read_only = 0|1`을 실행하는 것과 같습니다. 이러한 설정은 실행 시간에만 적용되며, 엔진이 재시작되면 적용되지 않습니다. DB 인스턴스의 파라미터 그룹에서 `read_only` 파라미터를 `true`으로 설정하여 DB 인스턴스를 읽기 전용으로 영구적으로 설정할 수 있습니다.

Aurora 멀티 마스터 클러스터의 성능 고려 사항

단일 마스터 클러스터와 멀티 마스터 클러스터 모두 Aurora 엔진이 OLTP 워크로드에 최적화되어 있습니다. OLTP 애플리케이션은 주로 매우 선택적인 랜덤 액세스 쿼리를 가지고 있고 수명이 짧은 트랜잭션으로 이루어져 있습니다. 이러한 작업을 동시에 다수 실행하는 워크로드에서 Aurora를 최대한 활용할 수 있습니다.

항상 100%의 사용률로 실행하지 않도록 하십시오. 이렇게 하면 Aurora가 내부 유지 관리 작업을 따라가게 됩니다. 멀티 마스터 클러스터가 얼마나 바쁘고 유지 관리 작업이 어느 정도 필요한지 측정하는 방법은 [Aurora 멀티 마스터 클러스터 모니터링 \(p. 594\)](#) 단원을 참조하십시오.

주제

- [멀티 마스터 클러스터의 쿼리 성능 \(p. 606\)](#)
- [멀티 마스터 클러스터에서의 충돌 해결 \(p. 606\)](#)
- [버퍼풀 및 딕셔너리 캐시 사용 최적화 \(p. 607\)](#)

멀티 마스터 클러스터의 쿼리 성능

멀티 마스터 클러스터는 전담 읽기 전용 노드 또는 읽기 전용 DNS 엔드포인트를 제공하지 않으므로 읽기 전용 DB 인스턴스 그룹을 생성하여 원하는 용도로 사용하는 것이 가능합니다. 자세한 내용은 [인스턴스 읽기 전용 모드 사용 \(p. 605\)](#) 단원을 참조하십시오.

다음과 같은 접근 방식을 사용하여 멀티 마스터 클러스터에서 쿼리 성능을 최적화할 수 있습니다.

- 관련 테이블, 데이터베이스 또는 쿼리에 관여하는 기타 스마트 객체가 포함된 샤크를 처리하는 DB 인스턴스에서 `SELECT` 문을 실행합니다. 이 기법을 사용하면 버퍼풀에서 데이터를 최대한 재사용할 수 있습니다. 또한 하나 이상의 DB 인스턴스에서 동일한 데이터가 캐시되는 것을 방지할 수 있습니다. 이 기법에 대한 자세한 내용은 [버퍼풀 및 딕셔너리 캐시 사용 최적화 \(p. 607\)](#) 단원을 참조하십시오.
- 읽기-쓰기 워크로드 격리가 필요한 경우에는 [인스턴스 읽기 전용 모드 사용 \(p. 605\)](#)에서 설명한 대로 하나 이상의 DB를 읽기 전용으로 지정합니다. 해당 인스턴스 엔드포인트를 연결하거나 모든 읽기 전용 인스턴스에 연결된 사용자 지정 엔드포인트를 정의하여 이러한 DB 인스턴스로 읽기 전용 세션을 보낼 수 있습니다.
- 모든 DB 인스턴스로 읽기 전용 쿼리를 분산합니다. 이러한 접근 방식은 최소한의 효율성을 제공합니다. 따라서 개발 및 테스트 단계에서 프로덕션 단계로 이동할 때는 가능한 다른 접근 방식 중 하나를 사용하십시오.

멀티 마스터 클러스터에서의 충돌 해결

멀티 마스터 클러스터를 위한 많은 모범 사례들이 쓰기 충돌 가능성 줄이는데 초점을 맞추고 있습니다. 쓰기 충돌 해결에는 네트워크 오버헤드가 수반됩니다. 또한 애플리케이션은 오류 조건을 처리하고 트랜잭션을 재시도해야 합니다. 이렇게 원치 않는 결과를 최대한 최소화하도록 하십시오.

- 가능하면 특정 테이블과 관련 인덱스에 대한 모든 변경 작업에서 동일한 DB 인스턴스를 사용하십시오. 오직 하나의 DB 인스턴스만 데이터 페이지를 수정한 경우, 해당 페이지를 변경해도 쓰기 충돌이 야기되지 않습니다. 이러한 액세스 패턴은 샤딩된 데이터베이스 배포나 멀티 테넌트 데이터베이스 배포에서 흔히 볼 수 있습니다. 따라서 멀티 마스터 클러스터를 사용하도록 이러한 배포를 전환하는 것은 비교적 쉽습니다.

- 멀티 마스터 클러스터에는 리더 엔드포인트가 없습니다. 리더 엔드포인트는 수신 연결을 로드 밸런싱하기 때문에 특정 연결에서 어떤 DB 인스턴스가 처리 중인지 파악하기 위해 애쓸 필요가 없습니다. 멀티 마스터 클러스터에서 연결을 관리하려면 각 연결에서 어떤 DB 인스턴스가 사용되는지 알아야 합니다. 따라서 특정 데이터베이스 또는 테이블에 대한 수정을 동일한 DB 인스턴스로 항상 라우팅할 수 있습니다.
- 적은 양의 데이터(16-KB 페이지 1개)에서 발생한 쓰기 충돌로 인해 전체 트랜잭션을 롤백할 정도로 대대적 인 작업이 야기될 수 있습니다. 따라서 멀티 마스터 클러스터의 트랜잭션을 비교적 간략하고 소규모로 유지하는 것이 좋습니다. OLTP 애플리케이션을 위한 이러한 모범 사례는 Aurora 멀티 마스터 클러스터에서 특히 중요합니다.

페이지 수준에서 충돌이 탐지됩니다. 다른 DB 인스턴스에서 제안된 변경 작업이 페이지 내의 다른 열을 변경하면서 충돌이 발생할 수 있습니다. 시스템에서 수행된 모든 페이지 변경은 충돌 탐지 대상입니다. 이러한 규칙은 소스가 사용자 트랜잭션이든 서버 백그라운드 프로세스이든 관계 없이 적용됩니다. 또한 테이블, 보조 인덱스, 실행 취소 공간 등 데이터 페이지의 출처에 관계 없이 적용됩니다.

각 DB 인스턴스가 스키마 객체 집합에 대한 모든 쓰기 작업을 처리하도록 쓰기 작업을 분할할 수 있습니다. 이 경우에 각 데이터 페이지에 대한 모든 변경은 하나의 특정 인스턴스에 의해 이루어집니다.

버퍼 폴 및 딕셔너리 캐시 사용 최적화

멀티 마스터 클러스터의 각 DB 인스턴스는 버퍼 폴, 테이블 핸들러 캐시, 테이블 딕셔너리 캐시 같이 별도의 인 메모리 버퍼와 캐시를 유지합니다. 각 DB 인스턴스에서 버퍼 및 캐시의 내용과 회전 양은 해당 인스턴스에서 처리되는 SQL 문에 따라 다릅니다.

효율적인 메모리 사용은 멀티 마스터 클러스터의 성능 향상과 I/O 비용 절감에 도움이 됩니다. 샤딩된 설계를 사용하여 특정 DB 인스턴스에서 각 샤드로 데이터와 쓰기를 물리적으로 분리하십시오. 이렇게 하면 각 DB 인스턴스에서 버퍼 캐시를 가장 효율적으로 사용할 수 있습니다. 테이블의 SELECT 문을 해당 테이블에 대해 쓰기 작업을 수행하는 동일한 DB 인스턴스에 할당하십시오. 이렇게 하면 이들 쿼리가 해당 DB 인스턴스에서 캐시된 데이터를 재사용하는 데 도움이 됩니다. 다수의 테이블 또는 파티션이 있는 경우에는 이 기법을 통해 각 DB 인스턴스에서 메모리에 보관되는 고유한 테이블 핸들러 및 딕셔너리 객체의 수를 줄일 수 있습니다.

Aurora 멀티 마스터 클러스터에 대한 접근 방식

다음 단원에서는 멀티 마스터 클러스터에 적합한 특정 배포에서 추해야 할 접근 방식을 확인할 수 있습니다. 이러한 접근 방식에는 DB 인스턴스가 중복되지 않는 데이터 부분에서 쓰기 작업을 수행하도록 워크로드를 분할하는 방법이 포함되어 있습니다. 이를 통해 쓰기 충돌 가능성은 최소화할 수 있습니다. 쓰기 충돌은 멀티 마스터 클러스터의 성능 튜닝 및 문제 해결에서 주로 초점을 맞추는 부분입니다.

주제

- [샤딩된 데이터베이스에서 멀티 마스터 클러스터 사용 \(p. 607\)](#)
- [샤딩 없이 멀티 마스터 클러스터 사용 \(p. 608\)](#)
- [활성 대기 구성으로 멀티 마스터 클러스터 사용 \(p. 608\)](#)

샤딩된 데이터베이스에서 멀티 마스터 클러스터 사용

샤딩은 Aurora 멀티 마스터 클러스터에 적합한 특정 스키마 설계 유형입니다. 샤딩된 아키텍처에서는 특정한 스키마 객체 그룹을 업데이트하도록 각 DB 인스턴스가 할당됩니다. 따라서 여러 개의 DB 인스턴스가 동시에 변경으로 인한 충돌 없이 동일한 공유 스토리지 볼륨에 쓰기 작업을 할 수 있습니다. 각 DB 인스턴스는 여러 샤드의 쓰기 작업을 처리할 수 있습니다. 애플리케이션 구성을 업데이트하여 언제라도 샤드에 대한 DB 인스턴스 매핑을 변경할 수 있습니다. 이 과정에서 데이터베이스 스토리지를 재구성하거나 DB 인스턴스를 재구성할 필요가 없습니다.

샤딩된 스키마 설계를 사용하는 애플리케이션은 Aurora 멀티 마스터 클러스터에서 사용하기 적합합니다. 샤딩된 시스템에서 데이터를 물리적으로 분할하면 쓰기 충돌을 피하는 데 도움이 됩니다. 각 샤드를 파티션, 테

이를 또는 데이터베이스 같은 스키마 객체에 매핑합니다. 애플리케이션은 특정 샤드의 모든 쓰기 작업을 해당되는 DB 인스턴스로 보냅니다.

BYOS(Bring-Your-Own-Shard)는 샤딩 및 분할된 데이터베이스를 이미 가지고 있고 애플리케이션이 이에 액세스할 수 있는 사용 사례를 설명합니다. 샤드는 이미 물리적으로 분리되어 있습니다. 따라서 스키마 설계를 변경하지 않고 Aurora 멀티 마스터 클러스터로 워크로드를 손쉽게 이동할 수 있습니다. 각 테넌트가 전용 테이블, 테이블 세트 또는 전체 데이터베이스를 사용하는 멀티 테넌트 데이터베이스에도 마찬가지로 간단한 마이그레이션 경로가 적용됩니다.

일대일 또는 다대일 방식으로 DB 인스턴스에 샤드 또는 테넌트를 매핑합니다. 각 DB 인스턴스는 샤드를 하나 이상 처리합니다. 샤딩된 설계는 주로 쓰기 작업에 적용됩니다. 동급의 성능을 갖춘 DB 인스턴스라면 어떤 샤드에 대해서든 `SELECT` 쿼리를 실행할 수 있습니다.

시간이 지나면서 샤드 중 하나가 훨씬 활발하게 가동된다고 가정해 봅시다. 워크로드를 재분배하기 위해 해당 샤드를 책임지는 DB 인스턴스를 전환할 수 있습니다. Aurora이 아닌 시스템에서는 데이터를 다른 서버로 물리적으로 이동시켜야 할 수 있습니다. Aurora 멀티 마스터 클러스터에서는 사용되지 않은 컴퓨팅 용량이 있는 다른 DB 인스턴스로 샤드에 대한 모든 쓰기 작업을 보냄으로써 이와 같은 재샤딩 작업을 수행할 수 있습니다. Aurora 공유 스토리지 모델에서는 데이터를 물리적으로 재구성할 필요가 없습니다.

샤딩 없이 멀티 마스터 클러스터 사용

스키마 설계가 데이터베이스, 테이블, 파티션 같이 물리적으로 분리된 컨테이너로 데이터를 세분하지 않을 경우에도 여전히 멀티 마스터 클러스터에서 이를 사용할 수 있습니다.

약간의 성능 오버헤드가 발생할 수 있으며, 쓰기 충돌이 교착 상태로 처리될 때 애플리케이션에서 간헐적인 트랜잭션 럭백을 처리해야 할 수 있습니다. 쓰기 충돌은 크기가 작은 테이블에서 쓰기 작업을 수행하는 동안에 발생하기 쉽습니다. 테이블에 포함된 데이터 페이지 수가 적은 경우에는 기본 키 범위의 다른 부분에서 나온 행이 동일한 데이터 페이지에 있는 경우가 발생할 수 있습니다. 이러한 중복은 다른 DB 인스턴스가 이러한 행들을 동시에 변경하는 경우에 쓰기 충돌로 이어질 수 있습니다.

또한 이 경우에는 보조 인덱스의 수를 최소화해야 합니다. 테이블에서 인덱싱된 열을 변경하면 Aurora이 관련 보조 인덱스에서 해당되는 변경을 수행합니다. 보조 인덱스와 관련 테이블 간에 행의 순서와 그룹화가 다르기 때문에 인덱스 변경은 쓰기 충돌을 야기할 수 있습니다.

이 기법을 사용해도 여전히 쓰기 충돌이 발생할 수 있기 때문에 Amazon은 가능하면 다른 접근 방식을 사용할 것을 권장하고 있습니다. 데이터를 서로 다른 스키마 객체로 세분하는 대안적 데이터베이스 설계를 사용할 수 있는지 확인하십시오.

활성 대기 구성으로 멀티 마스터 클러스터 사용

활성 대기 구성은 다른 DB 인스턴스와 계속해서 동기화되고 매우 신속하게 작업을 인계할 준비가 되어 있는 DB 인스턴스입니다. 이 구성은 단일 DB 인스턴스가 전체 워크로드를 처리할 수 있는 상황에서 뛰어난 가용성을 유지하는 데 도움이 됩니다.

읽기-쓰기 및 읽기 전용 트래픽을 포함해 모든 트래픽을 단일 DB 인스턴스로 보내서 활성 대기 구성으로 멀티 마스터 클러스터를 사용할 수 있습니다. DB 인스턴스가 사용 불가 상태가 되면 애플리케이션은 문제를 탐지하고 다른 DB 인스턴스로 모든 연결을 전환합니다. 이 경우, Aurora은 장애 조치를 수행하지 않습니다. 다른 DB 인스턴스가 이미 읽기-쓰기 연결을 수락할 수 있는 준비가 되어 있기 때문입니다. 단일 DB 인스턴스에 한번만 쓰기 작업을 수행하는 방법으로 쓰기 충돌을 피할 수 있습니다. 따라서 멀티 마스터 클러스터를 이런 방식으로 사용하기 위해 샤딩된 데이터베이스 스키마를 갖출 필요가 없습니다.

Tip

애플리케이션에서 짧은 일시 중지를 허용할 수 있는 경우에는 DB 인스턴스가 사용 불가 상태가 된 후 몇 초 동안 기다렸다가 다른 인스턴스로 쓰기 트래픽을 리디렉션할 수 있습니다. 재시작으로 인해 사용 불가 상태가 된 인스턴스도 약 10-20 초 후면 다시 사용 가능한 상태가 됩니다. 인스턴스를 신속하게 재시작할 수 없는 경우에 Aurora는 해당 인스턴스에 대해 복구 작업을 개시할 수 있습니다. 인스턴스는 종료 시 종료 과정의 일환으로 몇 가지 추가적인 정리 작업을 수행합니다. 인스턴스가 재시작되고 있는 동안에 다른 인스턴스에 대한 쓰기 작업을 시작한 경우에는 쓰기 충돌이 발생할

수 있습니다. 새 인스턴스의 SQL 문과 복구 작업(재시작 또는 종료된 인스턴스에서의 룰백 및 제거) 간에 이러한 충돌이 발생할 수 있습니다.

Amazon Aurora MySQL를 다른 AWS 서비스와 통합

Aurora MySQL DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora MySQL가 다른 AWS 서비스와 통합되었습니다. Aurora MySQL DB 클러스터는 AWS 서비스를 사용하여 다음과 같은 작업을 수행할 수 있습니다.

- `lambda_sync` 또는 `lambda_async` 네이티브 함수를 사용하여 동기적 또는 비동기적으로 AWS Lambda 함수를 호출하십시오. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- `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 \(p. 620\)](#) 단원을 참조하십시오.
- `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 \(p. 627\)](#) 단원을 참조하십시오.
- Application Auto Scaling을 사용하여 Aurora 복제본을 자동으로 추가 또는 제거하십시오. 자세한 내용은 [Aurora 복제본에 Amazon Aurora Auto Scaling 사용 \(p. 253\)](#)를 참조하십시오.
- Amazon Comprehend로 감성 분석을 수행하거나 Amazon SageMaker로 광범위한 기계 학습 알고리즘을 수행하십시오. 자세한 내용은 [Amazon Aurora에서 기계 학습\(ML\) 기능 사용 \(p. 267\)](#) 단원을 참조하십시오.

Aurora에는 AWS Identity and Access Management(IAM)을 사용하여 다른 AWS 서비스에 액세스할 수 있는 기능이 있습니다. 필요한 권한이 있는 IAM 역할을 만든 다음 해당 역할을 DB 클러스터와 연결하여 다른 AWS 서비스에 액세스하도록 허용하는 방법에 대한 세부 정보 및 지침은 [Amazon Aurora MySQL이 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여 \(p. 609\)](#) 단원을 참조하십시오.

Amazon Aurora MySQL이 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여

Note

Amazon Aurora MySQL 1.8 이후 버전에서 다른 AWS 서비스와 통합할 수 있습니다. 일부 통합 기능은 Aurora MySQL의 최신 버전에서만 사용할 수 있습니다. Aurora 버전에 대한 자세한 내용은 [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#) 단원을 참조하십시오.

Aurora MySQL DB 클러스터가 사용자를 대신하여 다른 서비스에 액세스하려면 AWS Identity and Access Management(IAM) 역할을 생성하고 구성하십시오. 이 역할은 다른 AWS 서비스에 액세스할 수 있도록 DB 클러스터의 데이터베이스 사용자에 권한을 부여합니다. 자세한 내용은 [AWS 서비스에 액세스할 수 있는 IAM 역할 설정 \(p. 610\)](#) 단원을 참조하십시오.

또한 대상 AWS 서비스로의 아웃바운드 연결을 허용하도록 Aurora DB 클러스터를 구성해야 합니다. 자세한 내용은 [Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 \(p. 619\)](#) 단원을 참조하십시오.

이렇게 하면 데이터베이스 사용자가 다른 AWS 서비스를 사용하여 다음 작업을 수행할 수 있습니다.

- `lambda_sync` 또는 `lambda_async` 네이티브 함수를 사용하여 동기적 또는 비동기적으로 AWS Lambda 함수를 호출하십시오. 또는 `mysql.lambda_async` 프로시저를 사용하여 비동기적으로 AWS Lambda 함

수를 호출하십시오. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.

- `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 \(p. 620\)](#) 단원을 참조하십시오.
- `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 \(p. 627\)](#) 단원을 참조하십시오.
- 로그 데이터를 Amazon CloudWatch Logs MySQL로 내보냅니다. 자세한 내용은 [Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 \(p. 639\)](#) 단원을 참조하십시오.
- Application Auto Scaling을 사용하여 Aurora 복제본을 자동으로 추가 또는 제거하십시오. 자세한 내용은 [Aurora 복제본에 Amazon Aurora Auto Scaling 사용 \(p. 253\)](#) 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 다음을 참조하십시오.
 - [Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 610\)](#)
 - [AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 612\)](#)
 - [CloudWatch Logs 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 613\)](#)
 - [AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 614\)](#)
2. IAM 역할을 만들고 만든 정책을 연결합니다. 자세한 내용은 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#) 단원을 참조하십시오.
3. IAM 역할을 Aurora DB 클러스터와 연결합니다. 자세한 내용은 [IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 \(p. 616\)](#) 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 필수 버킷 및 객체 권한이 나와 있습니다.

기능	버킷 권한	객체 권한
<code>LOAD DATA FROM S3</code>	<code>ListBucket</code>	<code>GetObject</code> <code>GetObjectVersion</code>
<code>LOAD XML FROM S3</code>	<code>ListBucket</code>	<code>GetObject</code> <code>GetObjectVersion</code>
<code>SELECT INTO OUTFILE S3</code>	<code>ListBucket</code>	<code>AbortMultipartUpload</code> <code>DeleteObject</code> <code>GetObject</code> <code>ListMultipartUploadParts</code>

기능	버킷 권한	객체 권한
		PutObject

Note

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. Aurora가 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 콘솔](#)을 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 example-bucket라는 Amazon S3 버킷에 액세스할 수 있게 허용하려면 Amazon 리소스 이름(ARN) 값을 arn:aws:s3:::example-bucket으로 설정하십시오.

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) Add additional permissions(권한 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. [Review policy]를 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 생성 \(p. 615\)](#)의 단계를 수행합니다.

AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성

Aurora이 사용자를 대신하여 AWS Lambda 함수를 호출하는 데 필요한 최소 권한을 제공하는 IAM 정책을 만들 수 있습니다.

아래 정책은 Aurora이 사용자를 대신하여 AWS Lambda 함수를 호출하는 데 필요한 권한을 추가합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToExampleFunction",  
            "Effect": "Allow",  
            "Action": "lambda:InvokeFunction",  
            "Resource":  
                "arn:aws:lambda:<region>:<123456789012>:function:<example_function>"  
        }  
    ]  
}
```

다음 단계를 사용하여 Aurora이 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 콘솔](#)을 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.

InvokeFunction이 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_function라는 Lambda 함수에 액세스하도록 허용하려고 하는 경우 ARN 값을 arn:aws:lambda:::function:example_function으로 설정하십시오.

AWS Lambda에 대한 액세스 정책을 정의하는 방법에 대한 자세한 내용은 [AWS Lambda의 인증 및 액세스 제어](#) 단원을 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. [Review policy]를 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 생성 \(p. 615\)](#)의 단계를 수행합니다.

CloudWatch Logs 리소스에 액세스할 수 있는 IAM 정책 생성

Aurora는 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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogGroups",  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogGroup",  
                "logs>PutRetentionPolicy"  
            ],  
            "Resource": [  
                "arn:aws:logs:*:log-group:/aws/rds/*"  
            ]  
        },  
        {  
            "Sid": "EnableCreationAndManagementOfRDSCloudwatchLogStreams",  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogStream",  
                "logs>PutLogEvents",  
                "logs>DescribeLogStreams",  
                "logs>GetLogEvents"  
            ],  
            "Resource": [  
                "arn:aws:logs:*:log-group:/aws/rds/:log-stream:/*"  
            ]  
        }  
    ]  
}
```

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 콘솔](#)을 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)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 [Log Group Name]에 대해 log-group:/aws/rds/*를 입력하고 [Add]를 선택합니다.
 8. [log-stream]에 대해 [Add ARN]을 선택합니다.
 9. [Add ARN(s)] 대화 상자에서 다음 값을 입력합니다.
 - 로그 그룹 이름 – log-group:/aws/rds/*
 - 로그 스트림 – log-stream
 - 로그 Stream Name – *
 10. [Add ARN(s)] 대화 상자에서 [Add]를 선택합니다.
 11. [Review policy]를 선택합니다.
 12. [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.
 13. [Create policy]를 선택합니다.
 14. [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)의 단계를 수행합니다.

AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성

Aurora은 데이터베이스 백업 암호화에 사용되는 AWS Key Management Service 키에 액세스할 수 있습니다. 하지만 Aurora에서 KMS 키에 액세스하도록 허용하는 IAM 정책을 먼저 생성해야 합니다.

아래 정책은 Aurora이 사용자를 대신하여 KMS 키에 액세스하는 데 필요한 권한을 추가합니다.

```
{  
    "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 콘솔](#)을 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-2와 같은 AWS 리전을 입력하십시오.
 - 계정 – 사용자 계정 번호를 입력하십시오.
 - 로그 스트림 이름 – KMS 키 ID를 입력하십시오.
8. [Add ARN(s)] 대화 상자에서 [Add]를 선택합니다.
9. [Review policy]를 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 생성 \(p. 615\)](#)의 단계를 수행합니다.

Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성

Aurora가 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 콘솔](#)을 엽니다.
2. 탐색 창에서 역할을 선택합니다.
3. [Create role]을 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: Permissions(다음: 권한)를 선택합니다.
7. Attach permissions policies(권한 정책 추가) 페이지에서 검색 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 \(p. 610\)](#)
 - [AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 612\)](#)
 - [CloudWatch Logs 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 613\)](#)
 - [AWS KMS 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 614\)](#)
9. Next: Tags(다음: 태그)를 선택한 후 Next: Review(다음: 검토)를 선택합니다.
10. 역할 이름에 IAM 역할의 이름을 입력합니다(예: RDSLoadFromS3). Role description(역할 설명) 값(선택 사항)을 추가할 수도 있습니다.

11. [Create Role]을 선택합니다.
12. [IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 \(p. 616\)](#)의 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 생성 \(p. 615\)](#)에서 생성한 역할을 해당 DB 클러스터와 연결합니다.

IAM 역할을 DB 클러스터와 연결하려면 다음 두 가지 작업을 수행합니다.

- 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.
- 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.

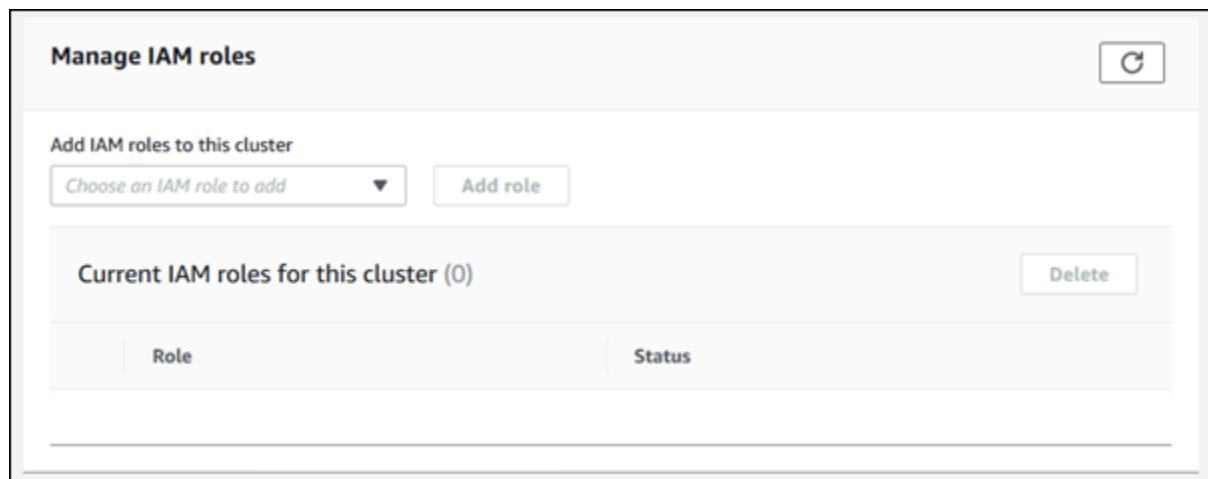
클러스터 수준 파라미터	설명
<code>aws_default_lambda_role</code>	DB 클러스터에서 Lambda 함수를 호출하는 데 사용됩니다.
<code>aws_default_logs_role</code>	이 파라미터는 DB 클러스터의 로그 데이터를 Amazon CloudWatch Logs로 내보낼 때 더 이상 필요하지 않습니다. Aurora MySQL은 이제 필요한 권한에 서비스 연결된 역할을 사용합니다. 서비스 연결 역할에 대한 자세한 내용은 Amazon Aurora에 서비스 연결 역할 사용 (p. 1001) 단원을 참조하십시오.
<code>aws_default_s3_role</code>	DB 클러스터에서 LOAD DATA FROM S3, LOAD XML FROM S3 또는 SELECT INTO OUTFILE S3 문을 호출하는 데 사용됩니다. 이 파라미터에 지정된 IAM 역할은 적절한 문에 대해 <code>aurora_load_from_s3_role</code> 또는 <code>aurora_select_into_s3_role</code> 에 IAM 역할이 지정되지 않은 경우에만 사용됩니다. 이전 버전의 Aurora에서는 이 파라미터에 지정된 IAM 역할이 항상 사용됩니다.
<code>aurora_load_from_s3_role</code>	DB 클러스터에서 LOAD DATA FROM S3 또는 LOAD XML FROM S3 문을 호출하는 데 사용됨. 이 파라미터에 대해 지정된 IAM 역할이 없는 경우 <code>aws_default_s3_role</code> 에 지정된 IAM 역할이 사용됩니다. 이전 버전의 Aurora에서는 이 파라미터를 사용할 수 없습니다.
<code>aurora_select_into_s3_role</code>	DB 클러스터에서 SELECT INTO OUTFILE S3 문을 호출하는 데 사용됨. 이 파라미터에 대해 지정된 IAM 역할이 없는 경우

클러스터 수준 파라미터	설명
	<p>aws_default_s3_role에 지정된 IAM 역할이 사용됩니다.</p> <p>이전 버전의 Aurora에서는 이 파라미터를 사용할 수 없습니다.</p>

Amazon RDS 클러스터가 사용자 대신 다른 AWS 서비스와 통신하도록 허용하는 IAM 역할을 연결하려면 다음 단계를 수행하십시오.

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. Connectivity & security(연결성 및 보안) 탭에 있는 Manage IAM roles(IAM 역할 관리) 섹션의 이 클러스터에 IAM 역할 추가에서 추가할 역할을 선택합니다.



5. [Add role]을 선택합니다.
6. (선택 사항) IAM 역할과 DB 클러스터의 연결을 중지하고 관련 권한을 제거하려면 해당 역할을 선택하고 삭제를 선택합니다.
7. RDS 콘솔의 탐색 창에서 [Parameter groups]를 선택합니다.
8. 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 5.6 호환 DB 클러스터는 aurora5.6을 선택하고 또 는 Aurora MySQL 5.7 호환 DB 클러스터는 aurora-mysql5.7을 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.

Create parameter group

Parameter group details
To create a parameter group, select a parameter group family, then name and describe your parameter group

Parameter group family
DB family that this DB parameter group will apply to

Type

Group name
Identifier for the DB parameter group

Description
Description for the DB parameter group

Cancel **Create**

- f. Create를 선택합니다.
9. 파라미터 그룹 페이지에서 DB 클러스터 파라미터 그룹을 선택하고 Parameter group actions(파라미터 그룹 작업)에서 편집을 선택합니다.
10. 적절한 클러스터 수준 파라미터를 관련 IAM 역할 ARN 값으로 설정합니다. 예를 들어, aws_default_s3_role 파라미터는 arn:aws:iam::123456789012:role/AllowAuroraS3Role로 설정할 수 있습니다.
11. Save changes(변경 사항 저장)를 선택합니다.
12. DB 클러스터의 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. 클러스터 수정을 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 파라미터 \(p. 665\)](#) 단원을 참조하십시오.

AWS CLI를 사용하여 IAM 역할을 DB 클러스터와 연결하려면

1. 다음과 같이 AWS CLI에서 add-role-to-db-cluster 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
```

- 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 AWS CLI에서 `create-db-cluster-parameter-group` 명령을 호출하십시오.

```
PROMPT> aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name AllowAWSAccess \
    --db-parameter-group-family aurora5.6 --description "Allow access to Amazon S3 and AWS Lambda"
```

Aurora MySQL 5.7 호환 DB 클러스터의 경우 `--db-parameter-group-family aurora-mysql5.7`을 지정하십시오.

- 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"
```

- 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 파라미터 \(p. 665\)](#) 단원을 참조하십시오.

Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화

AWS Lambda 함수를 호출하거나 Amazon S3에서 파일을 액세스하려면 Aurora DB 클러스터의 네트워크 구성에서 이러한 서비스에 대한 엔드포인트로의 아웃바운드 연결을 허용해야 합니다. Aurora는 서비스 엔드포인트에 연결할 수 없는 경우 다음의 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
```

AWS Lambda 함수를 호출하거나 Amazon S3에서 파일에 액세스할 때 이러한 메시지가 표시된다면, Aurora DB 클러스터가 퍼블릭인지 프라이빗인지 확인하십시오. Aurora DB 클러스터가 프라이빗이라면, 연결을 허용하도록 구성해야 합니다.

Aurora DB 클러스터를 퍼블릭으로 설정하려면, 퍼블릭 액세스 가능으로 표시해야 합니다. 이러한 경우 AWS Management 콘솔의 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 인스턴스를 사용한 작업 \(p. 1015\)](#) 단원을 참조하십시오. 퍼블릭 Amazon VPC 서브넷에 대한 자세한 내용은 [VPC 및 서브넷](#) 단원을 참조하십시오.

Aurora DB 클러스터가 퍼블릭 액세스 상태가 아니며 VPC 퍼블릭 서브넷에 있다면, 프라이빗 상태입니다. 프라이빗인 DB 클러스터가 있고 AWS Lambda 함수를 호출하거나 Amazon S3 파일에 액세스하고자 할 수 있습니다. 그런 경우 NAT(Network Address Translation)를 통해 인터넷 주소에 연결할 수 있도록 클러스터를 구성합니다. 대신 Amazon S3의 대안으로 DB 클러스터의 라우트 테이블과 관련된 Amazon S3의 VPC 종단점을 갖도록 VPC를 구성하면 됩니다. VPC에서 NAT를 구성하는 방법에 대한 자세한 내용은 [NAT 게이트웨이](#)를 참조하십시오. VPC 종단점 구성에 대한 자세한 내용은 [VPC Endpoints](#)를 참조하십시오.

관련 주제

- [Aurora를 다른 AWS 서비스와 통합 \(p. 252\)](#)
- [Amazon Aurora DB 클러스터 관리 \(p. 190\)](#)

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.

Note

Amazon Aurora MySQL 1.8 및 이후 버전에서는 Amazon S3 버킷에서는 데이터를 텍스트 파일에서 테이블로 로드할 수 있습니다. Aurora MySQL 버전에 대한 자세한 내용은 [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#) 단원을 참조하십시오.

이 기능은 현재 Aurora Serverless 클러스터에서 사용할 수 없습니다.

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 클러스터가 Amazon S3에 액세스하도록 허용하는 버킷 및 객체 권한을 제공하는 AWS Identity and Access Management(IAM) 정책을 생성하십시오. 지침은 [Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 610\)](#)을 참조하십시오.
2. IAM 역할을 생성하고 [Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 610\)](#)에서 생성한 IAM 정책을 새 IAM 역할에 연결하십시오. 지침은 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#) 단원을 참조하십시오.

3. DB 클러스터에서 사용자 지정 DB 클러스터 파라미터 그룹을 사용 중인지 확인하십시오.

사용자 지정 DB 클러스터 파라미터 그룹 생성에 대한 자세한 내용은 [DB 클러스터 파라미터 그룹 만들기 \(p. 172\)](#) 단원을 참조하십시오.

4. `aurora_load_from_s3_role` 또는 `aws_default_s3_role` DB 클러스터 파라미터를 새 IAM 역할의 Amazon 리소스 이름(ARN)으로 설정하십시오. `aurora_load_from_s3_role`에 대해 지정된 IAM 역할이 없는 경우 Aurora는 `aws_default_s3_role`에 지정된 IAM 역할을 사용합니다.

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 \(p. 170\)](#) 단원을 참조하십시오.

5. Aurora MySQL DB 클러스터의 데이터베이스 사용자가 Amazon S3에 액세스하도록 허용하려면 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)에서 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 \(p. 616\)](#) 단원을 참조하십시오.
6. Amazon S3으로의 아웃바운드 연결을 허용하도록 Aurora MySQL DB 클러스터를 구성하십시오. 자침은 [Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 \(p. 619\)](#) 단원을 참조하십시오.

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 문을 실행할 수 있도록 `LOAD FROM S3` 권한이 부여되어야 합니다. DB 클러스터의 마스터 사용자 이름에는 `LOAD FROM S3` 권한이 기본적으로 부여됩니다. 다음 문을 사용하여 다른 사용자에게 권한을 부여할 수 있습니다.

```
GRANT LOAD FROM S3 ON *.* TO 'user'@'domain-or-ip-address'
```

`LOAD FROM S3` 권한은 Amazon Aurora에만 특정하며 MySQL 데이터베이스 또는 RDS MySQL DB 인스턴스에서는 사용할 수 없습니다. 복제 마스터인 Aurora DB 클러스터와 복제 클라이언트인 MySQL 데이터베이스 간에 복제를 설정한 경우, `GRANT LOAD FROM S3` 문으로 인해 오류가 생겨 복제가 중단됩니다. 오류를 건너뛰고 복제를 계속 진행하셔도 됩니다. RDS MySQL DB 인스턴스에서 오류를 건너뛰려면 `mysql_rds_skip_repl_error` 프로시저를 사용하십시오. 외부 MySQL 데이터베이스에서 오류를 건너뛰려면 `SET GLOBAL sql_slave_skip_counter` 문을 사용합니다.

Amazon S3 버킷의 경로 지정

Amazon S3 버킷에 저장된 파일의 경로를 지정하는 구문은 다음과 같습니다.

```
s3-region://bucket-name/file-name-or-prefix
```

경로에는 다음 값이 포함됩니다.

- `region`(선택 사항) – 에서 로드할 Amazon S3 버킷이 포함된 AWS 리전입니다. 이 값은 선택 사항입니다. `region` 값을 지정하지 않으면 Aurora는 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 \(p. 623\)](#) 단원을 참조하십시오.

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.

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,...]
```

파라미터

다음은 `LOAD DATA FROM S3` 문에서 사용되는 필수 파라미터 및 선택 파라미터의 목록입니다. MySQL 문서의 [LOAD DATA INFILE Syntax](#) 단원에서 이러한 파라미터에 대한 자세한 내용을 볼 수 있습니다.

- **FILE | PREFIX | MANIFEST** – 단일 파일에서 데이터를 로드할지, 지정된 접두사와 일치하는 모든 파일에서 데이터를 로드할지, 지정된 매니페스트의 모든 파일에서 데이터를 로드할지를 식별합니다. **FILE**은 기본값입니다.
- **S3-URI** – 로드 할 텍스트나 매니페스트 파일의 URI를 지정하거나 사용할 Amazon S3 접두사를 지정합니다. [Amazon S3 버킷의 경로 지정 \(p. 621\)](#)에서 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 세트를 식별합니다.
- **FIELDS | COLUMNS** – 입력 파일의 필드 또는 열을 구분하는 방법을 식별합니다. 필드는 기본적으로 탭으로 구분됩니다.
- **LINES** – 입력 파일의 줄을 구분하는 방법을 식별합니다. 줄은 기본적으로 캐리지 리턴으로 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 ROWS를 사용하여 입력 파일의 첫 두 데이터 행을 건너뛸 수 있습니다. PREFIX를 사용하는 경우 IGNORE는 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 값은 저장할 수 있습니다.

예를 들어, 다음 문은 입력 파일의 첫 번째 열을 table1의 첫 번째 열로 로드하고, table1에 있는 table_column2 열의 값을 100으로 나눈 두 번째 열의 입력 값으로 설정합니다.

```
LOAD DATA FROM S3 's3://mybucket/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 table1의 첫 두 열을 입력 파일의 첫 두 열의 값으로 설정한 다음, table1에 있는 column3의 값을 현재 타임스탬프로 설정합니다.

```
LOAD DATA FROM S3 's3://mybucket/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 매니페스트 파일을 지정할 수 있습니다. MANIFEST 키워드와 LOAD DATA FROM S3 문을 함께 사용하려면 Aurora 버전이 1.11 이상이어야 합니다.

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 **customer.manifest** 매니페스트를 이용해 LOAD DATA FROM S3 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 실행하여 로드한 파일을 확인하려면, WHERE 절로 Amazon S3 URI의 기록에서 문에 사용된 메니페스트 파일을 필터링하십시오. 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. <ul style="list-style-type: none">LOAD DATA FROM S3 FILE 문에 대한 단일 데이터 파일LOAD DATA FROM S3 PREFIX 문에 대한 여러 데이터 파일에 매핑되는 Amazon S3 접두사LOAD 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.
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.

예제

다음 문은 Aurora DB 클러스터와 동일한 리전에 있는 Amazon S3 버킷에서 데이터를 로드합니다. 이 문은 dbbucket Amazon S3 버킷에 있는 파일 customerdata.txt에서 쉼표로 구분된 데이터를 읽은 다음 테이블 store-schema.customer-table로 해당 데이터를 로드합니다.

```
LOAD DATA FROM S3 's3://dbbucket/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 my-data Amazon S3 버킷의 employee-data 객체 접두사와 일치하는 모든 파일에서 쉼표로 구분된 데이터를 읽은 다음 employees 테이블로 해당 데이터를 로드합니다.

```
LOAD DATA FROM S3 PREFIX 's3-us-west-2://my-data/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 로드 합니다.

```
LOAD DATA FROM S3 MANIFEST 's3-us-west-2://aurora-bucket/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

다음은 LOAD DATA FROM S3 문에서 사용되는 필수 파라미터 및 선택 파라미터의 목록입니다. MySQL 문서의 [LOAD XML Syntax](#) 단원에서 이러한 파라미터에 대한 자세한 내용을 볼 수 있습니다.

- FILE | PREFIX – 단일 파일에서 데이터를 로드할지, 지정된 접두사와 일치하는 모든 파일에서 데이터를 로드할지 식별합니다. FILE은 기본값입니다.
- REPLACE | IGNORE – 입력 행이 데이터베이스 테이블의 기존 행과 고유 키 값이 동일한 경우 수행할 작업을 결정합니다.
 - 입력 행이 테이블의 기존 행을 대체하도록 하려면 REPLACE를 지정합니다.

- 입력 행을 무시하려는 경우 IGNORE를 지정하십시오. IGNORE는 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 세트를 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 LINES를 사용하여 텍스트 파일의 첫 번째 줄을 건너뛰거나 IGNORE 2 ROWS를 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 table1의 첫 번째 열로 로드하고, table1에 있는 table_column2 열의 값을 100으로 나눈 두 번째 열의 입력 값으로 설정합니다.

```
LOAD XML FROM S3 's3://mybucket/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 table1의 첫 두 열을 입력 파일의 첫 두 열의 값으로 설정한 다음, table1에 있는 column3의 값을 현재 타임스탬프로 설정합니다.

```
LOAD XML FROM S3 's3://mybucket/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.

관련 주제

- Amazon Aurora MySQL를 다른 AWS 서비스와 통합 (p. 609)
- Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 (p. 627)
- Amazon Aurora DB 클러스터 관리 (p. 190)
- Amazon Aurora DB 클러스터로 데이터 마이그레이션 (p. 188)

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 직접 저장할 수 있습니다. 이 기능을 통해 먼저 데이터를 클라이언트로 가져온 다음 클라이언트에서 Amazon S3으로 복사하는 과정을 생략할 수 있습니다. LOAD DATA FROM S3 문은 이 문에 의해 만들어진 파일을 사용하여 데이터를 Aurora DB 클러스터에 로드할 수 있습니다.

다. 자세한 내용은 [Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 \(p. 620\)](#) 단원을 참조하십시오.

Note

이 기능은 현재 Aurora Serverless 클러스터에서 사용할 수 없습니다.

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 클러스터가 Amazon S3에 액세스하도록 허용하는 버킷 및 객체 권한을 제공하는 AWS Identity and Access Management(IAM) 정책을 생성하십시오. 지침은 [Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 610\)](#)을 참조하십시오.
2. IAM 역할을 생성하고 [Amazon S3 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 610\)](#)에서 생성한 IAM 정책을 새 IAM 역할에 연결하십시오. 지침은 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#) 단원을 참조하십시오.
3. `aurora_select_into_s3_role` 또는 `aws_default_s3_role` DB 클러스터 파라미터를 새 IAM 역할의 Amazon 리소스 이름(ARN)으로 설정하십시오. `aurora_select_into_s3_role`에 대해 지정된 IAM 역할이 없는 경우 Aurora는 `aws_default_s3_role`에 지정된 IAM 역할을 사용합니다.

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 \(p. 170\)](#) 단원을 참조하십시오.

4. Aurora MySQL DB 클러스터의 데이터베이스 사용자가 Amazon S3에 액세스하도록 허용하려면 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)에서 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 \(p. 616\)](#) 단원을 참조하십시오.

5. Amazon S3으로의 아웃바운드 연결을 허용하도록 Aurora MySQL DB 클러스터를 구성하십시오. 지침은 [Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 \(p. 619\)](#) 단원을 참조하십시오.

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 문을 실행할 수 있도록 `SELECT INTO S3` 권한이 부여되어야 합니다. DB 클러스터의 마스터 사용자 이름에는 `SELECT INTO S3` 권한이 기본적으로 부여됩니다. 다음 문을 사용하여 다른 사용자에게 권한을 부여할 수 있습니다.

```
GRANT SELECT INTO S3 ON *.* TO 'user'@'domain-or-ip-address'
```

`SELECT INTO S3` 권한은 Amazon Aurora MySQL에만 특정하며 MySQL 데이터베이스 또는 RDS MySQL DB 인스턴스에서는 사용할 수 없습니다. 복제 마스터인 Aurora MySQL DB 클러스터와 복제 클라이언트인

MySQL 데이터베이스 간에 복제를 설정한 경우, GRANT SELECT INTO S3 문으로 인해 오류가 생겨 복제가 중단됩니다. 오류를 건너뛰고 복제를 계속 진행하셔도 됩니다. RDS MySQL DB 인스턴스에서 오류를 건너뛰려면 `mysql_rds_skip_repl_error` 프로시저를 사용하십시오. 외부 MySQL 데이터베이스에서 오류를 건너뛰려면 `SET GLOBAL sql_slave_skip_counter` 문을 사용합니다.

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 Aurora는 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 매니페스트 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 \(p. 623\)](#) 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"
    }
  ]
}
```

```
{  
    "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 파일 또는 암호화된 파일은 지원되지 않습니다.

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}]  
    [PROCEDURE procedure_name(argument_list)]  
INTO OUTFILE S3 's3_uri'  
[CHARACTER SET charset_name]  
[export_options]  
[MANIFEST {ON | OFF}]  
[OVERWRITE {ON | OFF}]  
  
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

다음은 SELECT INTO OUTFILE S3 문에서 사용되는 Aurora에 고유한 필수 파라미터 및 선택 파라미터의 목록입니다.

- s3-uri – 사용할 Amazon S3 접두사의 URI를 지정합니다. [Amazon S3 버킷의 경로 지정 \(p. 629\)](#)에서 설명하는 구문을 사용하여 URI를 지정합니다.
- FORMAT {CSV|TEXT} [HEADER] – 선택적으로 데이터를 CSV 형식으로 저장합니다. TEXT 옵션은 기본 값이며 기존 MySQL 내보내기 형식을 생성합니다. csv 옵션은 쉼표로 구분된 데이터 값을 생성합니다.

CSV 형식은 [RFC-4180](#)의 사양을 따릅니다. 선택적 키워드인 HEADER를 지정하는 경우 출력 파일에는 헤더 행이 한 줄 포함됩니다. 이 헤더 행의 레이블은 SELECT 문의 열 이름에 해당합니다. AWS ML 서비스에서 사용할 훈련 데이터 모델에 CSV 파일을 사용할 수 있습니다. AWS ML 서비스에 내보낸 Aurora 데이터를 사용하는 방법에 대한 자세한 내용은 [Amazon SageMaker 모델 교육을 위해 Amazon S3로 데이터 내보내기 \(p. 648\)](#) 단원을 참조하십시오.

- 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 \(p. 620\)](#).

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 \(p. 629\)](#) 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.

MySQL 설명서의 [SELECT Syntax](#) 및 [LOAD DATA INFILE Syntax](#) 섹션에서 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 선택한 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 s3-uri 파라미터에도 서로 다른 file_prefix를 지정해야 합니다. 여러 문을 사용하여 선택할 데이터를 분할하면 실행 오류 시 보다 쉽게 복구할 수 있습니다. 특정 문을 실행하는 동안 오류가 발생할 경우 해당 부분의 데이터만 다시 선택하여 Amazon S3에 업로드하면 되기 때문입니다. 여러 문을 사용할 경우 한 트랜잭션을 장시간 실행하지 않아도 되므로 성과를 개선할 수 있습니다.
- s3-uri 파라미터에 동일한 file_prefix를 사용하는 여러 SELECT INTO OUTFILE S3 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-uri에 지정된 동일한 파일 접두사를 포함하는 기존의 데이터 파일을 모두 제거하거나 SELECT INTO OUTFILE S3 쿼리에서 OVERWRITE ON을 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 ON을 지정하는 것입니다. 매니페스트 파일은 문이 기록하는 마지막 파일입니다. 즉, 매니페스트 파일이 만들어지면 문이 실행을 완료한 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.

예제

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 만들립니다. 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 기준 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 기준 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;
```

관련 주제

- Aurora를 다른 AWS 서비스와 통합 (p. 252)
- Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 (p. 620)
- Amazon Aurora DB 클러스터 관리 (p. 190)
- Amazon Aurora DB 클러스터로 데이터 마이그레이션 (p. 188)

Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출

네이티브 함수 또는 저장 프로시저를 사용하여 MySQL과 호환되는 Amazon Aurora DB 클러스터로부터 AWS Lambda 함수를 호출할 수 있습니다. Aurora MySQL에서 Lambda 함수를 호출하기 전에 Aurora DB 클러스터가 Lambda에 액세스해야 합니다.

최근의 Aurora MySQL 버전에서는 저장 프로시저가 더 이상 사용되지 않습니다. 다음 Aurora MySQL 버전 중 하나를 사용 중인 경우 Aurora MySQL 네이티브 함수를 사용할 것을 적극 권장합니다.

- MySQL 5.6 호환 클러스터의 경우 Aurora MySQL 버전 1.16 이상.
- MySQL 5.7 호환 클러스터의 경우 Aurora MySQL 버전 2.06 이상.

주제

- Aurora에 Lambda에 대한 액세스 권한 부여 (p. 633)
- Aurora MySQL 네이티브 함수로 Lambda 함수 호출 (p. 634)
- Aurora MySQL 저장 프로시저로 Lambda 함수 호출 (p. 636)

Aurora에 Lambda에 대한 액세스 권한 부여

Aurora MySQL에서 Lambda 함수를 호출하기 전에 Aurora MySQL DB 클러스터에 Lambda 액세스 권한을 부여해야 합니다.

Aurora MySQL에 Lambda에 대한 액세스 권한을 부여하려면

1. Aurora MySQL DB 클러스터가 Lambda 함수를 호출하도록 허용하는 권한을 제공하는 AWS Identity and Access Management(IAM) 정책을 생성하십시오. 지침은 [AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 612\)](#)을 참조하십시오.
2. IAM 역할을 생성하고 [AWS Lambda 리소스에 액세스할 수 있는 IAM 정책 생성 \(p. 612\)](#)에서 생성한 IAM 정책을 새 IAM 역할에 연결하십시오. 지침은 [Amazon Aurora에서 AWS 서비스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#) 단원을 참조하십시오.
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 \(p. 170\)](#) 단원을 참조하십시오.

- Aurora MySQL DB 클러스터의 데이터베이스 사용자가 Lambda 함수를 호출하도록 허용하려면 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)에서 생성한 역할을 해당 DB 클러스터와 연결하십시오. IAM 역할과 DB 클러스터 연결에 대한 자세한 내용은 [IAM 역할을 Amazon Aurora MySQL DB 클러스터와 연결 \(p. 616\)](#) 단원을 참조하십시오.

클러스터가 Aurora 글로벌 데이터베이스의 일부로 포함되는 경우 해당 역할을 글로벌 데이터베이스의 각 Aurora 클러스터와 연결합니다.

- Lambda으로의 아웃바운드 연결을 허용하도록 Aurora MySQL DB 클러스터를 구성하십시오. 지침은 [Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 \(p. 619\)](#) 단원을 참조하십시오.

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 1.16 및 이후 버전을 사용할 때 `lambda_sync` 및 `lambda_async` 네이티브 함수를 호출할 수 있습니다. Aurora MySQL 버전에 대한 자세한 내용은 [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#) 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.

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 호출된 함수의 실행 결과를 알아야 하는 경우, 동기식 함수 `lambda_sync`를 사용합니다. 다른 작업으로 넘어가기 전에 실행 결과를 알 필요가 없는 경우, 비동기식 함수 `lambda_async`를 사용합니다.

네이티브 함수를 호출하는 사용자에게는 `Invoke Lambda` 권한이 있어야 합니다. 사용자에게 이 권한을 부여하려면 마스터 사용자로 DB 인스턴스에 연결하고 다음 명령문을 실행합니다.

```
GRANT INVOKELAMBDA ON *.* TO user@domain-or-ip-address
```

다음 명령문을 실행하여 이 권한을 호출할 수 있습니다.

```
REVOKE INVOKELAMBDA ON *.* FROM user@domain-or-ip-address
```

`lambda_sync` 함수의 구문

`RequestResponse` 호출 유형으로 `lambda_sync` 함수를 동기식으로 호출합니다. 함수가 JSON 페이로드에서 Lambda 호출 결과를 반환합니다. 함수에는 다음과 같은 구문이 있습니다.

```
lambda_sync (  
    lambda_function_ARN,  
    JSON_payload
```

)

Note

트리거를 사용하여 데이터 수정 문에서 Lambda를 호출할 수 있습니다. 트리거는 SQL 문장 한 번이 아니라 수정된 행당 한 번 실행되며 한 번에 한 행씩 실행됩니다. 트리거 실행은 동기식이며 데이터 수정 문은 트리거 실행이 완료될 때까지 반환되지 않습니다.
쓰기 트래픽이 높은 테이블에서 트리거로 AWS Lambda 기능을 호출할 때는 주의하십시오.
INSERT, UPDATE, DELETE 트리거는 행마다 활성화됩니다. INSERT, UPDATE, DELETE 트리거가 있는 테이블에서 쓰기가 많은 워크로드는 AWS Lambda 기능에 다양한 호출을 야기합니다.

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 MySQL은 JSON 구문 분석을 지원하지 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-1:868710585169:function:BasicTestLambda',
    '{"operation": "ping"}');
```

lambda_async 함수의 구문

Event 호출 유형으로 lambda_async 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 MySQL은 JSON 구문 분석을 지원하지 않습니다. Lambda 함수가 숫자 또는 문자열과 같은 원자 값을 반환할 때 JSON 구문 분석이 필요하지 않습니다.

lambda_async 함수의 예

lambda_async를 기반으로 하는 다음 쿼리는 함수 ARN을 사용하여 Lambda 함수 BasicTestLambda를 비동기식으로 호출합니다. 함수에 대한 페이로드는 {"operation": "ping"}입니다.

```
SELECT lambda_async(
    'arn:aws:lambda:us-east-1:868710585169:function:BasicTestLambda',
    '{"operation": "ping"}');
```

관련 주제

- Aurora를 다른 AWS 서비스와 통합 (p. 252)
- Amazon Aurora DB 클러스터 관리 (p. 190)
- AWS Lambda 개발자 안내서

Aurora MySQL 저장 프로시저로 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.

Aurora MySQL 버전 고려 사항

Aurora MySQL 버전 1.8 이상에서는 이러한 저장 프로시저 대신 네이티브 함수 메서드를 사용하여 Lambda 함수를 호출할 수 있습니다. Amazon Aurora 버전 1.16으로 시작하면 저장된 프로시저 mysql.lambda_async는 더 이상 사용되지 않습니다. Aurora 버전 1.16 이상을 사용하는 경우 네이티브 Lambda 함수로 대신 작업하는 것이 좋습니다. 네이티브 함수 구성에 대한 자세한 내용은 [Lambda 함수를 호출하기 위해 네이티브 함수로 작업 \(p. 634\)](#). 단원을 참조하십시오.

현재 Aurora MySQL 버전 2.*에서는 네이티브 함수 기술을 사용하여 Lambda 함수를 호출할 수 없습니다. Aurora MySQL 5.7 호환성 클러스터의 경우 다음 섹션에 기술된 저장 프로시저 기술을 사용하십시오.

Lambda 함수 호출을 위한 mysql.lambda_async 프로시저 작업

mysql.lambda_async 프로시저는 Lambda 함수를 비동기 방식으로 호출하는 기본 제공 저장 프로시저입니다. 이 프로시저를 사용하려면 데이터베이스 사용자가 mysql.lambda_async 저장 프로시저에 대한 실행 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.

예제

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 테이블에서 트리거로 AWS Lambda 기능을 호출할 때는 주의하십시오. INSERT, UPDATE, DELETE 트리거는 행마다 활성화됩니다. INSERT, UPDATE, DELETE 트리거가 있는 테이블에서 쓰기가 많은 워크로드는 AWS Lambda 기능에 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
```

```
    );
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!');
```

관련 주제

- Aurora를 다른 AWS 서비스와 통합 (p. 252)
- Amazon Aurora DB 클러스터 관리 (p. 190)
- AWS Lambda 개발자 안내서

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 설명서에 있는 [일반 쿼리와 느린 쿼리 로그 출력 대상 선택](#)을 확인하십시오. Aurora MySQL 감사 로그 활성화에 대한 자세한 내용은 [고급 감사 활성화 \(p. 552\)](#) 단원을 참조하십시오.

Note

다음에 유의하십시오.

- 중국(닝샤) 리전의 경우 CloudWatch Logs에 로그를 게시할 수 없습니다.
- 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 활성화하고 클러스터 수준의 DB 파라미터 `server_audit_logs_upload`를 1로 설정하는 것입니다. `server_audit_logs_upload` 파라미터의 기본 값은 0입니다.

대안인 이 방법을 사용하는 경우 CloudWatch Logs에 액세스하고 `aws_default_logs_role` 를 러스터 수준 파라미터를 이 역할에 대한 ARN으로 설정하는 IAM 역할을 보유해야 합니다. 역할에 대한 상세 정보는 [AWS 서비스에 액세스할 수 있는 IAM 역할 설정 \(p. 610\)](#) 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 \(p. 1001\)](#) 단원을 참조하십시오.

- 감사 로그를 CloudWatch Logs로 내보내고 싶지 않은 경우, 감사 로그를 내보내는 모든 방법이 비활성화되었는지 확인하십시오. AWS Management 콘솔, AWS CLI, RDS API, `server_audit_logs_upload` 파라미터가 그러한 방법에 해당합니다.
- Aurora Serverless 클러스터에 대한 절차는 프로비저닝된 클러스터에 대한 절차와 약간 다릅니다. 서비스 클러스터는 사용자가 구성 파라미터를 통해 활성화하는 모든 종류의 로그를 자동으로 업로드할 수 있습니다. 따라서 DB 클러스터 파라미터 그룹에서 다양한 로그 유형을 켜고 끄는 방식으로 서비스 클러스터에 대한 로그 업로드를 활성화 또는 비활성화합니다. AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 클러스터 자체의 설정을 수정할 수는 없습니다. 서비스 클러스터에 대해 MySQL 로그를 활성화하는 방법에 관한 자세한 내용은 [Aurora Serverless 및 파라미터 그룹 \(p. 117\)](#) 단원을 참조하십시오.

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.

Linux, OS X, Unix의 경우:

```
aws rds modify-db-cluster \
  --db-cluster-identifier mydbcluster \
  --cloudwatch-logs-export-configuration '{"EnableLogTypes": \
  ["error", "general", "slowquery", "audit"]}'
```

Windows의 경우:

```
aws rds modify-db-cluster ^
  --db-cluster-identifier mydbcluster ^
  --cloudwatch-logs-export-configuration '{"EnableLogTypes": \
  ["error", "general", "slowquery", "audit"]}'
```

Example

다음 명령은 CloudWatch Logs에 로그 파일을 게시하도록 Aurora MySQL DB 클러스터를 생성합니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster \
  --db-cluster-identifier mydbcluster \
  --engine aurora \
  --enable-cloudwatch-logs-exports '[ "error", "general", "slowquery", "audit" ]'
```

Windows의 경우:

```
aws rds create-db-cluster ^
  --db-cluster-identifier mydbcluster ^
  --engine aurora ^
  --enable-cloudwatch-logs-exports '[ "error", "general", "slowquery", "audit" ]'
```

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.

DB 클러스터의 모든 DB 인스턴스에 있는 모든 이벤트가 서로 다른 로그 스트림을 사용하는 로그 그룹으로 이동합니다.

지정된 이름이 있는 로그 그룹이 존재할 경우 Aurora는 이 로그 그룹을 사용하여 Aurora DB 클러스터의 로그 데이터를 내보냅니다. AWS CloudFormation 같은 자동 구성은 미리 정의된 로그 보존 기간, 메트릭 필터 및 고객 액세스 권한이 있는 로그 그룹을 생성할 수 있습니다. 또는 기본 로그 보존 기간인 만기 없음을 CloudWatch Logs에서 사용하면 새 로그 그룹이 자동으로 생성됩니다. CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용하여 로그 보존 기간을 변경할 수 있습니다. CloudWatch Logs의 로그 보존 기간 변경에 대한 자세한 내용은 [CloudWatch Logs에서 로그 데이터 보존 변경](#)을 참조하십시오.

CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용하여 DB 클러스터의 로그 이벤트 안에서 정보를 검색할 수 있습니다. 로그 데이터 검색 및 필터링에 관한 자세한 내용은 [로그 데이터 검색 및 필터](#)를 참조하십시오.

Aurora MySQL에서 기계 학습(ML) 사용

Aurora 기계 학습은 SQL 언어를 사용해 기계 학습 기반 예측을 데이터베이스 애플리케이션에 추가할 수 있게 지원합니다. Aurora 기계 학습은 Aurora 데이터베이스와 AWS 기계 학습(ML) 서비스인 Amazon SageMaker 및 Amazon Comprehend 간의 고도로 최적화된 통합을 이용합니다.

Aurora 기계 학습의 이점은 다음과 같습니다.

- ML 기반 예측을 기준 데이터베이스 애플리케이션에 추가할 수 있습니다. 사용자 지정 통합을 빌드하거나 별도 도구를 학습할 필요가 없습니다. 기계 학습 처리를 저장된 함수에 대한 호출로 SQL 쿼리에 직접 포함할 수 있습니다.
- ML 통합으로 ML 서비스가 트랜잭션 데이터 작업을 할 수 있게 신속히 지원할 수 있습니다. 기계 학습 작업을 수행하기 위해 데이터를 데이터베이스 밖으로 이동할 필요가 없습니다. 데이터베이스 애플리케이션에서 사용하기 위해 기계 학습의 결과를 변환하거나 다시 가져올 필요가 없습니다.
- 기존 거버넌스 정책을 사용하여 기본 데이터 및 생성된 통찰력에 대한 액세스 권한을 어떤 사용자에게 부여할지 제어할 수 있습니다.

AWS ML 서비스는 자체 프로덕션 환경에서 설정 및 실행되는 관리형 서비스입니다. 현재 Aurora 기계 학습은 감성 분석을 위한 Amazon Comprehend와 광범위한 ML 알고리즘을 위한 Amazon SageMaker와 통합됩니다.

Amazon Comprehend에 대한 일반적인 내용은 [Amazon Comprehend 단원](#)을 참조하십시오. Aurora 및 Amazon Comprehend를 함께 사용하는 것에 대한 자세한 내용은 [감성 감지를 위해 Amazon Comprehend 사용 \(p. 651\)](#) 단원을 참조하십시오.

Amazon SageMaker에 대한 일반적인 내용은 [Amazon SageMaker 단원](#)을 참조하십시오. Aurora 및 Amazon SageMaker를 함께 사용하는 것에 대한 자세한 내용은 [Amazon SageMaker를 사용하여 자체 ML 모델 실행 \(p. 649\)](#) 단원을 참조하십시오.

주제

- [Aurora 기계 학습의 사전 조건 \(p. 643\)](#)
- [Aurora 기계 학습 활성화 \(p. 643\)](#)
- [Amazon SageMaker 모델 교육을 위해 Amazon S3로 데이터 내보내기 \(p. 648\)](#)
- [Amazon SageMaker를 사용하여 자체 ML 모델 실행 \(p. 649\)](#)
- [감성 감지를 위해 Amazon Comprehend 사용 \(p. 651\)](#)
- [Aurora 기계 학습의 성능 요인 \(p. 652\)](#)
- [Aurora 기계 학습 모니터링 \(p. 653\)](#)
- [Aurora 기계 학습의 제한 사항 \(p. 654\)](#)

Aurora 기계 학습의 사전 조건

현재 Aurora 기계 학습에서는 클러스터가 Aurora MySQL 데이터베이스 엔진을 사용할 것을 요구합니다. 이 기능은 Aurora MySQL 2.07.0 이상을 실행하는 모든 Aurora 클러스터에서 사용할 수 있습니다. 이전 Aurora 클러스터를 이러한 릴리스 중 하나로 업그레이드하고 이 클러스터에서 이 기능을 사용할 수 있습니다.

Aurora 기계 학습 활성화

ML 기능 활성화에는 다음과 같은 단계가 수반됩니다.

- 애플리케이션에 사용하려는 ML 알고리즘의 종류에 따라 Aurora 클러스터가 Amazon 기계 학습 서비스인 Amazon SageMaker 또는 Amazon Comprehend에 액세스할 수 있게 합니다.
- Amazon SageMaker의 경우 Aurora CREATE FUNCTION 문을 사용하여 추론 기능에 액세스하는 저장 함수를 설정합니다.

Note

Aurora 기계 학습에는 감성 분석을 위해 Amazon Comprehend를 호출하는 내장 함수가 포함되어 있습니다. Amazon Comprehend만 사용하는 경우 CREATE FUNCTION 문을 실행할 필요가 없습니다.

주제

- Amazon Comprehend 및 Amazon SageMaker에 대한 IAM 액세스 권한 설정 (p. 644)
- Aurora 기계 학습 서비스 호출을 위한 SQL 권한 부여 (p. 648)
- Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 (p. 648)

Amazon Comprehend 및 Amazon SageMaker에 대한 IAM 액세스 권한 설정

Amazon SageMaker 및 Amazon Comprehend 서비스에 액세스하려면 먼저 Aurora MySQL 클러스터가 AWS ML 서비스에 액세스할 수 있게 하십시오. Aurora MySQL DB 클러스터가 AWS ML 서비스에 사용자 대신 액세스하게 하려면 AWS Identity and Access Management(IAM) 역할을 생성 및 구성하십시오. 이 역할은 Aurora MySQL 데이터베이스의 사용자에게 AWS ML 서비스에 액세스할 수 있는 권한을 부여합니다.

AWS Management 콘솔을 사용하는 경우 AWS는 사용자를 대신하여 IAM 설정을 자동으로 수행합니다. 다음 정보를 건너뛰고 [콘솔을 사용해 Aurora DB 클러스터를 Amazon S3, Amazon SageMaker 또는 Amazon Comprehend에 연결 \(p. 644\)](#)의 절차를 따라도 됩니다.

AWS CLI 또는 RDS API를 사용하여 Amazon SageMaker 또는 Amazon Comprehend에 대해 IAM 역할을 설정하는 작업은 다음 단계로 구성됩니다.

1. IAM 정책을 생성하여 Aurora MySQL에서 호출할 수 있는 Amazon SageMaker 엔드포인트를 지정하거나 Amazon Comprehend에 액세스할 수 있게 하십시오.
2. IAM 역할을 생성하여 Aurora MySQL 데이터베이스 클러스터가 AWS ML 서비스에 액세스할 수 있게 허용하십시오. 위에서 생성한 IAM 정책은 IAM 역할에 연결됩니다.
3. Aurora MySQL 데이터베이스 클러스터가 AWS ML 서비스에 액세스할 수 있게 허용하려면 위에서 생성한 IAM 역할을 데이터베이스 클러스터에 연결하십시오.
4. 데이터베이스 애플리케이션이 AWS ML 서비스를 호출할 수 있게 허용하려면 특정 데이터베이스 사용자에게도 권한을 부여해야 합니다. Amazon SageMaker의 경우 엔드포인트에 대한 호출이 저장 함수 내부에 래핑되어 있으므로 저장 함수의 EXECUTE 권한을 이 함수를 호출하는 모든 데이터베이스 사용자에게도 부여해야 합니다.

사용자를 대신하여 Aurora MySQL DB 클러스터에서 다른 AWS 서비스에 액세스하도록 허용하는 방법에 대한 일반적인 내용은 [Amazon Aurora MySQL이 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있도록 권한 부여 \(p. 609\)](#) 단원을 참조하십시오.

콘솔을 사용해 Aurora DB 클러스터를 Amazon S3, Amazon SageMaker 또는 Amazon Comprehend에 연결

Aurora 기계 학습에서는 DB 클러스터가 Amazon S3, Amazon SageMaker 및 Amazon Comprehend의 일부 조합을 사용할 것을 요구합니다. Amazon Comprehend는 감성 분석을 위한 것이고, Amazon SageMaker는 광범위한 기계 학습 알고리즘을 위한 것입니다. Aurora 기계 학습의 경우 Amazon S3는 교육 Amazon SageMaker 모델 전용입니다. 사용할 수 있는 교육된 모델이 아직 없고 교육이 자신의 책임인 경우에는 Aurora 기계 학습에서 Amazon S3만 사용하면 됩니다. DB 클러스터를 이 서비스에 연결하려면 각 Amazon 서비스에 대해 AWS Identity and Access Management(IAM) 역할을 설정해야 합니다. IAM 역할을 통해 DB 클러스터의 사용자는 해당 서비스로 인증할 수 있습니다.

Amazon S3, Amazon SageMaker 또는 Amazon Comprehend에 대해 IAM 역할을 생성하려면 필요한 각 서비스에 대해 다음 단계를 반복하십시오.

DB 클러스터를 Amazon 서비스에 연결하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 Databases(데이터베이스)를 선택한 다음 사용하려는 Aurora MySQL DB 클러스터를 선택합니다.
3. Connectivity & security(연결 및 보안) 탭을 선택합니다.

4. IAM 역할 관리 섹션에서 Select a service to connect to this cluster(이 클러스터에 연결할 서비스 선택)를 선택합니다.
5. 드롭다운 목록에서 연결할 서비스를 선택합니다.
 - Amazon S3
 - Amazon Comprehend
 - Amazon SageMaker
6. Connect service(서비스 연결)를 선택합니다.
7. Connect service(서비스 연결) 창에서 특정 서비스에 대한 필수 정보를 입력하십시오.
 - Amazon SageMaker의 경우 Amazon SageMaker 엔드포인트의 Amazon 리소스 이름(ARN)을 입력하십시오. 엔드포인트가 무엇을 나타내는지에 관한 자세한 내용은 [Amazon SageMaker 호스팅 서비스에서 모델 배포 단원](#)을 참조하십시오.

[Amazon SageMaker 콘솔](#)의 탐색 창에서 엔드포인트를 선택한 다음 사용하려는 엔드포인트의 ARN을 복사합니다.

- Amazon Comprehend의 경우 추가 파라미터는 지정하지 않습니다.
- Amazon S3의 경우 사용할 Amazon S3 버킷의 ARN을 입력하십시오.

Amazon S3 버킷 ARN의 형식은 `arn:aws:s3:::bucket_name`입니다. 사용하는 Amazon S3 버킷이 교육 Amazon SageMaker 모델에 대한 요구 사항을 포함하도록 설정되어 있는지 확인하십시오. 모델을 교육할 때 Aurora DB 클러스터에서는 데이터를 Amazon S3 버킷으로 내보낼 수 있는 권한뿐 아니라 데이터를 버킷에서 가져올 수 있는 권한도 요구합니다.

Amazon S3 버킷 ARN에 대한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 [정책에서 리소스 지정](#)을 참조하십시오. Amazon SageMaker에서 Amazon S3 버킷을 사용하는 방법에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서의 [1단계: Amazon Amazon S3 버킷 생성](#)을 참조하십시오.

8. Connect service(서비스 연결)를 선택합니다.
9. Aurora는 새 IAM 역할을 생성하여 이 클러스터에 대한 현재 IAM 역할의 DB 클러스터 목록에 추가합니다. IAM 역할의 상태는 처음에는 진행 중으로 되어 있습니다. IAM 역할 이름은 연결된 각 서비스에 대해 다음과 같은 패턴으로 자동 생성됩니다.
 - Amazon S3 IAM 역할 이름 패턴은 `rds-cluster_ID-S3-policy-timestamp`입니다.
 - Amazon SageMaker IAM 역할 이름 패턴은 `rds-cluster_ID-SageMaker-policy-timestamp`입니다.
 - Amazon Comprehend IAM 역할 이름 패턴은 `rds-cluster_ID-Comprehend-policy-timestamp`입니다.

또한 Aurora는 새 IAM 정책을 생성하여 역할에 연결합니다. 정책 이름은 유사한 이름 지정 규칙을 따르며 타임스탬프도 있습니다.

[Amazon SageMaker에 액세스할 수 있는 IAM 정책 생성\(AWS CLI 전용\)](#)

Note

AWS Management 콘솔 사용 시 Aurora는 IAM 정책을 자동으로 생성합니다. 이 경우 이 섹션을 건너뛸 수 있습니다.

아래 정책은 Aurora MySQL이 사용자를 대신하여 Amazon SageMaker 함수를 호출하는 데 필요한 권한을 추가합니다. 데이터베이스 애플리케이션이 Aurora MySQL 클러스터에서 액세스하는 데 필요한 모든 Amazon SageMaker 엔드포인트를 하나의 정책에서 지정할 수 있습니다. 이 정책을 통해 Amazon SageMaker 엔드포인트에 대해 AWS 리전을 지정할 수 있습니다. 그러나 Aurora MySQL 클러스터는 클러스터와 동일한 AWS 리전에 배포된 Amazon SageMaker 모델만 호출할 수 있습니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToInvokeRCFEndPoint",  
            "Effect": "Allow",  
            "Action": "sagemaker:InvokeEndpoint",  
            "Resource": "arn:aws:sagemaker:region:123456789012:endpoint/endpointName"  
        }  
    ]  
}
```

다음 명령은 AWS CLI를 통해 동일한 작업을 수행합니다.

```
aws iam put-role-policy --role-name role_name --policy-name policy_name --policy-document  
'{"Version": "2012-10-17", "Statement": [ { "Sid": "AllowAuroraToInvokeRCFEndPoint",  
    "Effect": "Allow", "Action": "sagemaker:InvokeEndpoint", "Resource":  
    "arn:aws:sagemaker:region:123456789012:endpoint/endpointName }]}'
```

Amazon Comprehend에 액세스할 수 있는 IAM 정책 생성(AWS CLI 전용)

Note

AWS Management 콘솔 사용 시 Aurora는 IAM 정책을 자동으로 생성합니다. 이 경우 이 섹션을 건너뛸 수 있습니다.

아래 정책은 Aurora MySQL이 사용자를 대신하여 AWS Amazon Comprehend 함수를 호출하는 데 필요한 권한을 추가합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToInvokeComprehendDetectSentiment",  
            "Effect": "Allow",  
            "Action": [  
                "comprehend:DetectSentiment",  
                "comprehend:BatchDetectSentiment"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

다음 명령은 AWS CLI를 통해 동일한 작업을 수행합니다.

```
aws iam put-role-policy --role-name role_name --policy-name policy_name  
--policy-document '{ "Version": "2012-10-17", "Statement": [ { "Sid":  
    "AllowAuroraToInvokeComprehendDetectSentiment", "Effect": "Allow", "Action":  
    [ "comprehend:DetectSentiment", "comprehend:BatchDetectSentiment" ], "Resource": "*" }]}'
```

Amazon Comprehend에 대한 액세스 권한을 부여하는 IAM 정책을 생성하려면

1. [IAM 관리 콘솔](#)을 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 Comprehend를 선택합니다.
5. 작업에서 Detect Sentiment(감성 감지) 및 BatchDetectSentiment를 선택합니다.
6. 정책 검토를 선택합니다.
7. 이름에서 IAM 정책의 이름을 입력합니다. IAM 역할을 만들어 DB 클러스터와 연결할 때 이 이름을 사용합니다. Description 값(선택 사항)을 추가할 수도 있습니다.
8. [Create policy]를 선택합니다.
9. [Amazon Aurora에서 AWS 서비스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)의 절차를 완료합니다.

Amazon SageMaker 및 Amazon Comprehend에 액세스할 수 있는 IAM 역할 생성

IAM 역할을 생성한 후 ML 서비스에 액세스하기 위해 데이터베이스 사용자를 대신하여 Aurora MySQL 클러스터가 맡을 수 있는 IAM 역할을 생성하십시오. IAM 역할을 생성하려면 AWS Management 콘솔 또는 AWS CLI를 사용하면 됩니다. IAM 역할을 생성하여 이 역할에 이전 정책을 연결하려면 [Amazon Aurora에서 AWS 서비스에 액세스하도록 허용하는 IAM 역할 생성 \(p. 615\)](#)에 설명된 단계를 따르십시오. IAM 역할에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [IAM 역할](#)을 참조하십시오.

인증을 위해서는 전역 IAM 역할만 사용할 수 있습니다. 데이터베이스 사용자 또는 세션과 연결된 IAM 역할은 사용할 수 없습니다. 이 요구 사항은 Lambda 및 Amazon S3 서비스와의 Aurora 통합에 대한 요구 사항과 동일합니다.

IAM 역할을 Aurora MySQL DB 클러스터와 연결(AWS CLI 전용)

Note

AWS Management 콘솔 사용 시 Aurora는 IAM 정책을 자동으로 생성합니다. 이 경우 이 섹션을 건너뛸 수 있습니다.

마지막 단계는 IAM 역할을 Aurora MySQL DB 클러스터와 연결된 IAM 정책에 연결하는 것입니다. IAM 역할을 Aurora DB 클러스터와 연결하려면 다음 두 가지 작업을 수행합니다.

1. AWS Management 콘솔, [add-role-to-db-cluster](#) AWS CLI 명령 또는 [AddRoleToDBCluster](#) RDS API 작업을 사용하여 DB 클러스터의 연결된 역할 목록에 역할을 추가하십시오.
2. 관련 AWS ML 서비스에 대한 클러스터 수준 파라미터를 연결된 IAM 역할의 ARN으로 설정합니다. Aurora 클러스터에서 사용하려는 AWS ML 서비스에 따라 `aws_default_sagemaker_role` 또는 `aws_default_comprehend_role` 파라미터를 사용하거나 이 두 파라미터를 모두 사용하십시오.

클러스터 수준 파라미터는 DB 클러스터 파라미터 그룹으로 그룹화됩니다. 이전 클러스터 파라미터를 설정하려면 기존 사용자 지정 DB 클러스터 그룹을 사용하거나 새로운 클러스터 그룹을 생성하십시오. 새 DB 클러스터 파라미터 그룹을 만들려면 다음과 같이 AWS CLI에서 `create-db-cluster-parameter-group` 명령을 호출하십시오.

```
PROMPT> aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name AllowAWSAccessToExternalServices \
    --db-parameter-group-family aurora-mysql5.7 --description "Allow access to Amazon S3, AWS Lambda, AWS SageMaker, and AWS Comprehend"
```

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 AllowAWSAccessToExternalServices \
--parameters
"ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraS3Role,ApplyMethod=pending-reboot" \
--parameters
"ParameterName=aws_default_sagemaker_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraSageMakerRole,ApplyMethod=pending-reboot" \
--parameters
"ParameterName=aws_default_comprehend_role,ParameterValue=arn:aws:iam::123456789012:role/
AllowAuroraComprehendRole,ApplyMethod=pending-reboot"
```

새로운 DB 클러스터 파라미터 그룹을 사용하도록 DB 클러스터를 수정합니다. 그런 다음 클러스터를 재부팅합니다. 아래에서는 이 작업을 수행하는 방법을 보여줍니다.

```
PROMPT> aws rds modify-db-cluster --db-cluster-identifier your_cluster_id --db-cluster-
parameter-group-nameAllowAWSAccessToExternalServices
PROMPT> aws rds failover-db-cluster --db-cluster-identifier your_cluster_id
```

인스턴스가 재부팅되면 IAM 역할이 DB 클러스터와 연결됩니다.

Aurora 기계 학습 서비스 호출을 위한 SQL 권한 부여

필요한 IAM 정책 및 역할을 생성하여 이 역할을 Aurora MySQL DB 클러스터에 연결한 후에는 Amazon SageMaker를 위한 Aurora 기계 학습 저장 함수와 Amazon Comprehend를 위한 내장 함수를 호출할 수 있는 권한을 개별 데이터베이스 사용자에게 부여합니다.

네이티브 함수를 호출하는 데이터베이스 사용자에게는 INVOK SAGEMAKER 또는 INVOK COMPREHEND 권한을 부여해야 합니다. 사용자에게 이 권한을 부여하려면 마스터 사용자로 DB 인스턴스에 연결하고 다음 문을 실행합니다. 데이터베이스 사용자에 대한 적절한 세부 정보로 대체하십시오.

```
GRANT INVOK SAGEMAKER ON *.* TO user@domain-or-ip-address
GRANT INVOK COMPREHEND ON *.* TO user@domain-or-ip-address
```

Amazon SageMaker의 경우 사용자 정의 함수에서는 파라미터를 추론 산출을 위해 모델로 전송하고 호출할 엔드포인트 이름을 구성하도록 정의합니다. 엔드포인트를 호출하려는 각 데이터베이스 사용자를 위해 Amazon SageMaker에 구성된 저장 함수에 대한 EXECUTE 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
```

Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화

또한 Amazon SageMaker 및 Amazon Comprehend는 외부 AWS 서비스이므로 대상 AWS 서비스로의 아웃 바운드 연결을 허용하도록 Aurora DB 클러스터를 구성해야 합니다. 자세한 내용은 [Amazon Aurora MySQL에서 다른 AWS 서비스로의 네트워크 통신 활성화 \(p. 619\)](#) 단원을 참조하십시오.

VPC 엔드포인트를 사용해 Amazon S3에 연결할 수 있습니다. 이때 AWS PrivateLink를 사용해 Aurora를 AWS 기계 학습 서비스 또는 Amazon S3에 연결할 수는 없습니다.

Amazon SageMaker 모델 교육을 위해 Amazon S3로 데이터 내보내기

팀에서 기계 학습 작업을 어떻게 분배하느냐에 따라 이 작업을 수행하지 않을 수도 있습니다. 다른 누군가가 사용자에게 Amazon SageMaker 모델을 제공하는 경우 사용자는 이 섹션을 건너뛸 수 있습니다.

Amazon SageMaker 모델을 교육하려면 데이터를 Amazon S3 버킷으로 내보냅니다. Amazon S3 버킷은 Jupyter Amazon SageMaker 노트북 인스턴스가 모델을 배포하기 전에 모델을 교육하는 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 기계 학습은 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-4180](#)의 사양을 따릅니다.
- 선택적 키워드인 HEADER를 지정하는 경우 출력 파일에는 헤더 행이 한 줄 포함됩니다. 이 헤더 행의 레이블은 SELECT 문의 열 이름에 해당합니다.
- CSV 및 HEADER라는 키워드를 식별자로 계속 사용할 수 있습니다.

SELECT INTO의 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']  
]
```

Amazon SageMaker를 사용하여 자체 ML 모델 실행

Amazon SageMaker는 종합 관리형 기계 학습 서비스입니다. Amazon SageMaker를 사용해 데이터 과학자와 개발자는 기계 학습 모델을 빠르고 쉽게 빌드하고 교육할 수 있습니다. 그런 다음 모델을 프로덕션 지원 호스팅 환경으로 직접 배포할 수 있습니다. Amazon SageMaker는 데이터 소스에 쉽게 액세스할 수 있도록 통합된 Jupyter 작성 노트북 인스턴스를 제공합니다. 이로써 서버용 하드웨어 인프라를 관리하지 않고도 탐색 및 분석을 수행할 수 있습니다. 또한 분산된 환경 내 초대용량 데이터 세트에 대해 효율적으로 실행되도록 최적화된 일반 기계 학습 알고리즘을 제공합니다. BYOM(Bring-Your-Own-Algorithm) 및 프레임워크 기본 원칙을 통해 Amazon SageMaker는 특정 워크플로에 맞게 조정되는 유연한 분산형 교육 옵션을 제공합니다.

현재 Aurora 기계 학습은 text/csv의 ContentType를 통해 쉼표로 구분된 값 형식을 읽고 쓸 수 있는 모든 Amazon SageMaker 엔트포인트를 지원합니다. 현재 이 형식을 수용하는 내장된 Amazon SageMaker 알고리즘은 Random Cut Forest, Linear Learner, 1P, XGBoost, 3P입니다. 알고리즘이 항목 하나당 여러 개의 출력을 반환하는 경우 Aurora 기계 학습 함수는 첫 번째 항목만 반환합니다. 이 첫 번째 항목은 대표적인 결과가 될 것으로 예상됩니다.

Aurora 기계 학습은 항상 Aurora 클러스터와 동일한 AWS 리전에 있는 Amazon SageMaker 엔드포인트를 호출합니다. 그러므로 단일 리전 Aurora 클러스터의 경우 항상 Aurora MySQL 클러스터와 동일한 AWS 리전에 모델을 배포하십시오.

Aurora 글로벌 데이터베이스를 사용 중인 경우 글로벌 데이터베이스의 일부인 각 AWS 리전의 서비스 간에 동일한 통합을 설정하십시오. 특히 글로벌 데이터베이스의 모든 AWS 리전에 대해 다음 조건이 만족되었는지 확인하십시오.

- 각 AWS 리전의 글로벌 데이터베이스에 대해 Amazon SageMaker, Amazon Comprehend 또는 Lambda와 같은 외부 서비스에 액세스할 수 있는 적절한 IAM 역할을 구성하십시오.
- 동일한 엔드포인트 이름으로 배포된 동일한 교육 Amazon SageMaker 모델이 모든 AWS 리전에 있는지 확인하십시오. 기본 AWS 리전의 Aurora 기계 학습 함수에 대해 CREATE FUNCTION 문을 실행하기 전에 확인해야 합니다. 글로벌 데이터베이스에서는 기본 AWS 리전에서 실행하는 모든 CREATE FUNCTION 문이 모든 보조 리전에서도 즉시 실행됩니다.

추론을 위해 Amazon SageMaker에 배포된 모델을 사용하려면 저장 함수에 대해 익숙한 MySQL 데이터 정의 언어(DDL) 문을 사용하여 사용자 정의 함수를 생성합니다. 각 저장 함수는 이 모델을 호스팅하는 Amazon SageMaker 엔드포인트를 대표합니다. 이러한 함수를 정의할 때 모델에 대한 입력 파라미터, 호출할 특정 Amazon SageMaker 엔드포인트, 반환 유형을 지정합니다. 이 함수에서는 입력 파라미터에 근거하여 모델을 실행한 후에 Amazon SageMaker 엔드포인트가 계산한 추론을 반환합니다. 모든 Aurora 기계 학습 저장 함수에서는 숫자형 또는 VARCHAR를 반환합니다. BIT를 제외한 모든 숫자형을 사용할 수 있습니다. JSON, BLOB, TEXT, DATE 등 다른 유형은 허용되지 않습니다. 모델 교육을 위해 Amazon S3로 내보낸 입력 파라미터와 동일한 모델 입력 파라미터를 사용하십시오.

```
CREATE FUNCTION function_name (arg1 type1, arg2 type2, ...) -- variable number of arguments
    [DEFINER = user]                                         -- same as existing MySQL
CREATE FUNCTION
    RETURNS mysql_type          -- For example, INTEGER, REAL, ...
    [SQL SECURITY { DEFINER | INVOKER } ]                         -- same as existing MySQL
CREATE FUNCTION
    ALIAS AWS_SAGEMAKER_INVOKE_ENDPOINT   -- ALIAS replaces the stored function body. Only
    AWS_SAGEMAKER_INVOKE_ENDPOINT is supported for now.
    ENDPOINT NAME 'endpoint_name'                                -- default is 10,000
    [MAX_BATCH_SIZE max_batch_size];
```

이것은 기존 CREATE FUNCTION DDL 문의 변형입니다. Amazon SageMaker 함수를 정의하는 CREATE FUNCTION 문에서 함수 본문을 정의해서는 안 됩니다. 그 대신에 새로운 키워드인 ALIAS를 함수 본문이 일반적으로 배치되는 곳에 지정하십시오. 현재 Aurora 기계 학습은 이 확장 구문에 대해 aws_sagemaker_invoke_endpoint만 지원합니다. endpoint_name 파라미터를 지정해야 합니다. 선택적 파라미터인 max_batch_size는 Amazon SageMaker에 대한 실제 일괄 처리 요청에서 처리되는 입력의 최대 개수를 제한합니다. Amazon SageMaker 엔드포인트는 모델마다 다양한 특성이 있을 수 있습니다. max_batch_size 파라미터는 너무 큰 입력으로 인한 오류를 방지하거나 Amazon SageMaker가 응답을 더 빨리 반환하게 하는 데 도움이 될 수 있습니다. max_batch_size 파라미터는 ML 요청 처리에 사용되는 내부 버터의 크기에 영향을 미칩니다. max_batch_size에 너무 큰 값을 지정하면 DB 인스턴스에 상당한 메모리 오버헤드가 발생할 수 있습니다.

MANIFEST 설정을 기본값인 OFF로 두는 것이 좋습니다. 사용자는 MANIFEST ON 옵션을 사용할 수 있지만 일부 Amazon SageMaker 기능에서는 이 옵션을 통해 내보낸 CSV를 직접 사용할 수 없습니다. 매니페스트 형식은 Amazon SageMaker의 예상 매니페스트 형식과 호환되지 않습니다.

각 Amazon SageMaker 모델에 대해 별도 저장 함수를 생성합니다. 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 실행할 수 있는지 제어할 수 있습니다. 또한 최대 배치 크기를 나타내는 파라미터를 지정하여 런타임 특성을 제어할 수 있습니다.

현재 모든 Aurora 기계 학습 함수에는 NOT DETERMINISTIC 속성이 있습니다. 이 속성을 명시적으로 지정하지 않으면 Aurora가 NOT DETERMINISTIC으로 자동 설정합니다. 이러한 요구 사항이 부과되는 이유는 ML 모델이 데이터베이스에 대한 알림 없이 변경될 수 있기 때문입니다. 변경된 경우 Aurora 기계 학습 함수에 대한 호출을 통해 단일 트랜잭션 내에서 동일 입력에 대해 다른 결과가 반환될 수 있습니다.

CREATE FUNCTION 문에서는 CONTAINS SQL, NO SQL, READS SQL DATA 또는 MODIFIES SQL DATA 특성을 사용할 수 없습니다.

다음은 이상을 감지하기 위해 Amazon SageMaker 엔드포인트를 호출하는 방법에 관한 예시입니다. Amazon SageMaker 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 Amazon SageMaker 함수에 대한 문자 집합 요구 사항

문자열을 반환하는 Amazon SageMaker 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 Amazon SageMaker 함수에서는 반환 값에 대해 utf8mb4 문자 집합을 사용합니다. 반환 값에서는 ML 함수가 반환 유형에 다른 문자 집합을 둑시적으로 또는 명시적으로 선언한다 하더라도 이러한 문자 집합을 사용합니다. ML 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 ML 함수 결과는 쿼리 중에 문자열이 내부적으로 처리되는 방식으로 인해 잘릴 수 있습니다.

감성 감지를 위해 Amazon Comprehend 사용

Amazon Comprehend에서는 기계 학습을 사용하여 텍스트 데이터에 있는 통찰력과 관계를 찾습니다. 기계 학습 경험 또는 전문성이 전혀 없다 하더라도 이 AWS 기계 학습 서비스를 사용할 수 있습니다. Aurora 기계 학습에서는 데이터베이스에 저장된 텍스트의 감성 분석을 위해 Amazon Comprehend를 사용합니다. 예를 들어 Amazon Comprehend를 사용해 콜센터 수신 통화 관련 문서를 분석하여 감성을 감지하고 발신자-에이전트 간 역학을 더 잘 이해할 수 있습니다. AWS 기계 학습 블로그의 [콜센터 통화 분석](#)이라는 게시물에서 더 자세한 설명을 볼 수 있습니다.

또한 단일 쿼리를 사용하여 데이터베이스의 기타 정보에 대한 분석을 감성 분석과 결합할 수 있습니다. 예를 들어 다음 사항이 합쳐진 문제에 대한 수신 통화 센터의 평균 감성을 감지할 수 있습니다.

- 30일 이상 미해결 상태
- 특정 제품 또는 기능과 관련된 문제
- 소셜 미디어 영향력이 가장 큰 고객에게 발생한 문제

Aurora 기계 학습에서 Amazon Comprehend를 사용하는 것은 SQL 함수를 호출하는 것만큼 쉽습니다. Aurora 기계 학습에서는 Amazon Comprehend를 통한 감성 분석을 수행할 수 있고 톤 두 가지 내장 Amazon Comprehend 함수 aws_comprehend_detect_sentiment() 및 aws_comprehend_detect_sentiment_confidence()를 제공합니다. 사용자가 분석하는 각 텍스트 조각에 이 함수를 사용하면 감성과 신뢰도 수준을 확인하는 데 도움이 됩니다.

```
-- Returns one of 'POSITIVE', 'NEGATIVE', 'NEUTRAL', 'MIXED'  
aws_comprehend_detect_sentiment(  
    input_text  
    ,language_code  
    [,max_batch_size] -- default is 25. should be greater than 0  
)  
  
-- Returns a double value that indicates confidence of the result of  
aws_comprehend_detect_sentiment.  
aws_comprehend_detect_sentiment_confidence(  
    input_text  
    ,language_code  
    [,max_batch_size] -- default is 25. should be greater than 0.  
)
```

`max_batch_size`는 Amazon Comprehend 함수 호출의 성능을 조정하는 데 도움이 됩니다. 대용량 배치는 Aurora 클러스터의 더 큰 메모리 사용으로 인한 더 빠른 성능을 상쇄합니다. 자세한 내용은 [Aurora 기계 학습의 성능 요인 \(p. 652\)](#) 단원을 참조하십시오.

Amazon Comprehend의 감성 감지 함수에 대한 파라미터 및 반환 유형에 관한 자세한 내용은 [DetectSentiment](#)를 참조하십시오.

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;
```

Note

Amazon Comprehend는 현재 일부 AWS 리전에서만 사용할 수 있습니다. Amazon Comprehend를 사용할 수 있는 AWS 리전을 확인하려면 [AWS 리전 테이블 페이지](#)를 참조하십시오.

Aurora 기계 학습의 성능 요인

Aurora 기계 학습 함수 호출 시 대부분의 작업은 외부 ML 서비스 내에서 이루어집니다. 이러한 분리를 통해 Aurora 클러스터와 독립적으로 기계 학습 서비스의 리소스를 확장할 수 있습니다. Aurora 내에서는 주로 함수 호출 자체를 최대한 효율적으로 수행하는 데 집중하십시오.

쿼리 캐시

Aurora MySQL 쿼리 캐시는 ML 함수에 대해서는 작동하지 않습니다. Aurora MySQL은 ML 함수를 호출하는 모든 SQL 문에 대한 쿼리 캐시에 쿼리 결과를 저장하지 않습니다.

Aurora 기계 학습 함수 호출을 위한 배치 최적화

Aurora 클러스터에서 영향을 미칠 수 있는 주요 Aurora 기계 학습 성능 측면은 Aurora 기계 학습 저장 함수에 대한 호출에 대한 배치 모드 설정입니다. 기계 학습 함수는 일반적으로 상당한 오버헤드가 필요하므로 행마다 별도의 외부 서비스를 호출하는 것은 불가능합니다. Aurora 기계 학습은 여러 행에 대한 외부 Aurora 기계 학습 서비스 호출을 단일 배치로 결합하여 이러한 오버헤드를 최소화할 수 있습니다. Aurora 기계 학습은 모

든 입력 행에 대한 응답을 수신하고 실행 중인 쿼리에 대해 한 번에 한 개의 행씩 응답을 제공합니다. 이러한 최적화를 통해 결과를 변경하지 않고도 Aurora 쿼리의 처리량 및 지연 시간을 개선할 수 있습니다.

Amazon SageMaker 엔드포인트에 연결된 Aurora 저장 함수를 생성할 때 배치 크기 파라미터를 정의합니다. 이 파라미터는 Amazon SageMaker에 대한 모든 기본 호출을 위해 전송되는 행의 수에 영향을 미칩니다. 대량의 행을 처리하는 쿼리의 경우 각 행에 대해 별도의 Amazon SageMaker 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 Amazon SageMaker 함수에 적용할 수 있는 경우 EXPLAIN PLAN 문에서 산출하는 실행 계획을 확인함으로써 이를 구별할 수 있습니다. 이 경우 실행 계획의 extra 열에는 Batched machine learning이 포함됩니다. 다음 예에서는 배치 모드를 사용하는 Amazon SageMaker 함수 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 기계 학습 함수 평가를 위한 배치 최적화는 다음 사례에 적용됩니다.

- select 목록 또는 SELECT 문의 WHERE 절 내 함수 호출. 아래에 설명된 것과 같이 몇 가지 예외가 있습니다.
- INSERT 및 REPLACE 문의 VALUES 목록 내 함수 호출
- UPDATE 문 내 SET 값의 ML 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 기계 학습 모니터링

Aurora 기계 학습 배치 실행의 성능을 모니터링하기 위해 Aurora MySQL은 다음과 같이 쿼리할 수 있는 몇 가지 전역 변수를 포함합니다.

```
show status like 'Aurora_ml%';
```

FLUSH STATUS 문을 사용해 이 상태 변수를 재설정할 수 있습니다. 따라서 모든 숫자는 변수를 마지막으로 재설정한 이후의 합계, 평균 등을 나타냅니다.

Aurora_ml_logical_response_cnt

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 ML 서비스에서 수신하는 총 응답 횟수입니다.

Aurora_ml_actual_request_cnt

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 ML 서비스에서 수신하는 총 요청 횟수입니다.

Aurora_ml_actual_response_cnt

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 ML 서비스에서 수신하는 총 응답 횟수입니다.

Aurora_ml_cache_hit_cnt

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 Aurora MySQL이 ML 서비스에서 수신하는 총 내부 캐시 히트 횟수입니다.

Aurora_ml_single_request_cnt

DB 인스턴스 사용자가 실행하는 모든 쿼리에 걸쳐 비일괄 모드로 평가되는 ML 함수의 총 개수입니다.

Aurora 기계 학습 함수에서 호출하는 Amazon SageMaker 작업의 성능을 모니터링하는 방법에 대한 자세한 내용은 [Amazon SageMaker](#)를 참조하세요.

Aurora 기계 학습의 제한 사항

Aurora 기계 학습에는 다음과 같은 제한 사항이 적용됩니다.

항상 생성되는 열에는 Aurora 기계 학습 함수를 사용할 수 없습니다. 동일한 제한 사항이 모든 Aurora MySQL 저장 함수에 적용됩니다. 이 함수는 이 바이너리 로그(binlog) 형식과 호환되지 않습니다. 생성되는 열에 대한 자세한 내용은 [MySQL 설명서](#)를 참조하십시오.

--binlog-format=STATEMENT 설정은 Aurora 기계 학습 함수 호출에 대해 예외를 발생시킵니다. 오류가 발생하는 이유는 Aurora 기계 학습이 모든 ML 함수를 비결정적인 것으로 간주하고 비결정적인 저장 함수는 이 binlog 형식과 호환되지 않기 때문입니다. 이 binlog 형식에 대한 자세한 내용은 [MySQL 설명서](#)를 참조하십시오.

Amazon Aurora MySQL 랩 모드

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](#) (p. 655) 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 [DB 파라미터 그룹의 파라미터 설정](#) (p. 174) 단원을 참조하십시오. 파라미터 그룹 및 Amazon Aurora에 대한 자세한 내용은 [Aurora MySQL 파라미터](#) (p. 665) 단원을 참조하십시오.

Aurora 랩 모드 기능

다음 표에는 Aurora 랩 모드를 활성화했을 때 현재 사용할 수 있는 Aurora 기능이 나와 있습니다. 이러한 기능을 사용하려면 먼저 Aurora 랩 모드를 활성화해야 합니다.

기능	설명
배치화 스캔	Aurora MySQL 스캔 배치화는 인 메모리 스캔 지향 쿼리의 속도를 크게 높입니다. 이 기능은 일괄 처리로 테이블 전체 스캔, 인덱스 전체 스캔 및 인덱스 범위 스캔의 성능을 향상 시킵니다.
해시 조인	이 기능은 동등 조인을 사용하여 많은 양의 데이터를 조인해야 하는 경우 쿼리 성능을 향상 시킬 수 있습니다. Aurora MySQL 버전 1에는 랩 모드가 필요합니다. Aurora MySQL 버전 2에는 랩 모드 없이 이 기능을 사용할 수 있습니다. 이 기능 사용에 대한 자세한 내용은 Aurora MySQL에서 해시 조인 작업 (p. 663) 단원을 참조하십시오.
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. 테이블 복사를 위해 임시 스토리지 를 사용하지 않으므로 스몰 인스턴스 클래스의 라지 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 (p. 525) 단원을 참조하십시오.

Amazon Aurora MySQL 모범 사례

이 주제에서는 Amazon Aurora MySQL DB 클러스터 사용 또는 이 클러스터로의 데이터 마이그레이션과 관련된 모범 사례와 옵션에 대해 설명합니다.

주제

- [연결되어 있는 DB 인스턴스 확인 \(p. 656\)](#)
- [T2 인스턴스 사용 \(p. 656\)](#)
- [AWS Lambda 함수 호출 \(p. 657\)](#)
- [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#)
- [Amazon Aurora MySQL에서 다중 스레드 복제 슬레이브 사용 \(p. 659\)](#)
- [Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 조정 \(p. 659\)](#)
- [MySQL 데이터베이스에서 Amazon Aurora를 재해 복구용으로 사용 \(p. 662\)](#)
- [감소된 중단 시간으로 MySQL에서 Amazon Aurora MySQL로 마이그레이션 \(p. 662\)](#)
- [Amazon Aurora MySQL에서 XA 트랜잭션 사용 \(p. 663\)](#)
- [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#)
- [Aurora MySQL에서 외래 키 작업 \(p. 664\)](#)

- 관련 주제 (p. 665)

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';
```

Aurora 복제본에 연결된 경우 `innodb_read_only` 변수는 ON로 설정되며 기본 인스턴스에 연결된 경우 OFF로 설정됩니다.

이 접근 방식은 애플리케이션 코드에 논리를 추가하여 작업의 균형을 조정하거나 쓰기 작업에 올바른 연결이 사용되고 있는지 확인하려는 경우에 유용할 수 있습니다. 이 기법은 단일 마스터 복제를 사용하는 Aurora 클러스터에만 적용됩니다. 멀티 마스터 클러스터의 경우에는 모든 DB 인스턴스가 `innodb_read_only=OFF` 설정을 가집니다.

T2 인스턴스 사용

`db.t2.small` 또는 `db.t2.medium` DB 인스턴스 클래스를 사용하는 Amazon Aurora MySQL 인스턴스는 연장된 시간 동안 높은 워크로드를 지원하지 않는 애플리케이션에 가장 적합합니다. T2 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. `db.t2.small` 및 `db.t2.medium` DB 인스턴스 클래스는 개발 및 테스트 서버 또는 기타 비 프로덕션 서버에만 사용하는 것이 좋습니다. T2 인스턴스에 대한 세부 정보는 [T2 인스턴스](#)를 참조하십시오.

MySQL 성능 스키마를 Amazon Aurora MySQL T2 인스턴스에서 활성화하지 마십시오. 성능 스키마가 활성화된 경우 T2 인스턴스의 메모리가 부족할 수 있습니다.

T2 인스턴스를 Aurora MySQL DB 클러스터의 DB 인스턴스로 사용할 때는 다음을 권장합니다.

- DB 클러스터에서 T2 인스턴스를 DB 인스턴스 클래스로 사용하는 경우에는 DB 클러스터의 모든 인스턴스에서 동일한 DB 인스턴스 클래스를 사용하는 것이 좋습니다. 예를 들어 `db.t2.medium`을 기본 인스턴스로 사용한다면 Aurora Replicas에도 `db.t2.medium`을 사용하는 것이 좋습니다.
- 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 R3 DB 인스턴스 클래스 중 하나를 사용하도록 하는 것이 좋습니다.(컴퓨팅 확장).

지표 모니터링에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 지표 모니터링 \(p. 346\)](#) 단원을 참조하십시오.

- 단일 마스터 복제를 사용하는 Aurora MySQL DB 클러스터의 경우에는 기본 인스턴스와 Aurora 복제본 간의 복제본 지연 시간(AuroraReplicaLag)을 모니터링합니다.

Aurora Replica가 기본 인스턴스 이전에 CPU 크레딧이 바닥나면 기본 인스턴스 지연 시간으로 인해 Aurora Replica가 다시 시작하는 일이 빈번해집니다. 애플리케이션에 Aurora MySQL DB 클러스터의 Aurora 복제본 간에 많은 양의 읽기 작업이 분산되어 있는 동시에 기본 인스턴스의 쓰기 작업의 양이 최소한일 때 일반적으로 나타나는 결과입니다.

복제 지연이 지속적으로 증가하는 경우에는 DB 클러스터의 Aurora Replicas에 대한 CPU 크레딧 잔고가 소진되지 않도록 해야 합니다.

CPU 크레딧 잔고가 지속 가능한 수준에 있지 않다면 DB 인스턴스를 수정하여 지원되는 R3 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 OFF가 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 R3 DB 인스턴스 클래스(컴퓨팅 확장) 중 하나를 사용하도록 인스턴스를 수정할 수 있습니다.

AWS Lambda 함수 호출

Amazon Aurora 버전 1.16 이상을 사용하는 경우 `lambda_sync` 및 `lambda_async` 네이티브 함수를 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 \(p. 633\)](#) 단원을 참조하십시오.

Amazon Aurora의 비동기식 키 미리 가져오기 작업

Note

비동기식 키 미리 가져오기(AKP) 기능은 Amazon Aurora MySQL 1.15 이후 버전에서 사용할 수 있습니다. Aurora MySQL 버전에 대한 자세한 정보는 [Amazon Aurora MySQL 데이터베이스 엔진 업데이트 \(p. 686\)](#)를 참조하십시오.

Amazon Aurora는 AKP를 사용하여 여러 인덱스 사이에 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 행 까지 식별합니다. 그런 다음 쿼리를 실행하기에 앞서 백그라운드 스레드를 사용하여 식별된 행이 포함된 페이징을 메모리에 비동기식으로 로드합니다.

비동기식 키 미리 가져오기(AKP) 활성화

MySQL 서버 변수 `aurora_use_key_prefetch`를 `on`으로 설정하여 AKP 기능을 활성화할 수 있습니다. 기본적으로 이 값은 `on`으로 설정됩니다. 하지만 먼저 BKA 조인 알고리즘을 활성화하고 비용 기반 MRR 기능을 비활성화해야만 AKP를 활성화할 수 있습니다. 이를 위해서는 `optimizer_switch` MySQL 서버 변수의 값을 다음과 같이 설정해야 합니다.

- batched_key_access를 on로 설정합니다. 이 값은 BKA 조인 알고리즘의 사용을 제어합니다. 기본적으로 이 값은 off로 설정됩니다.
- mrr_cost_based를 off로 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 옵티マイ저 스위치에 대한 자세한 정보는 MySQL 설명서에서 [Switchable Optimizations](#)를 참조하십시오.

비동기식 키 미리 가져오기를 위한 쿼리 최적화

쿼리의 AKP 기능 사용 여부를 확인할 수 있습니다. 방법은 쿼리를 실행하기 전에 EXPLAIN 문에서 EXTENDED 키워드를 사용하여 쿼리를 프로파일링하는 것입니다. EXPLAIN 문이 지정한 쿼리에 사용할 실행 계획에 대한 정보를 제공합니다.

EXPLAIN 문 출력에서 Extra 열은 실행 계획에 추가되는 정보에 대한 설명입니다. AKP 기능이 쿼리에 사용 할 테이블에 적용되는 경우 이 열에 다음 값 중 하나가 포함됩니다.

- Using Key Prefetching
- Using join buffer (Batched Key Access with Key Prefetching)

다음은 EXPLAIN 문에 EXTENDED 키워드를 사용하여 AKP를 이용할 수 있는 쿼리의 실행 계획을 확인하는 예제입니다.

```
mysql> explain extended 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.supplier.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)

```

확장된 EXPLAIN 출력 형식에 대한 자세한 정보는 MySQL 제품 설명서에서 [Extended EXPLAIN Output Format](#)을 참조하십시오.

Amazon Aurora MySQL에서 다중 스레드 복제 슬레이브 사용

Aurora MySQL DB 클러스터를 복제 슬레이브로 사용하는 경우 Aurora은 기본적으로 단일 스레드 복제를 사용합니다. Amazon Aurora가 다중 스레드 복제를 금지하지는 않지만 Aurora MySQL는 다중 스레드 복제와 관련된 여러 문제점을 MySQL로부터 물려받았습니다. 프로덕션 환경에서는 다중 스레드 복제를 사용하지 않는 것이 좋습니다. 다중 스레드 복제를 사용할 경우 모든 사용을 철저히 테스트할 것을 권장합니다.

Amazon Aurora에서의 복제에 대한 자세한 정보는 [Amazon Aurora를 사용한 복제 \(p. 55\)](#) 단원을 참조하십시오.

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. Aurora를 사용하여 MySQL DB 인스턴스에

대한 읽기 조정을 수행하려면 Amazon Aurora MySQL DB 클러스터를 생성한 후 이 클러스터를 MySQL DB 인스턴스의 복제 슬레이브로 설정합니다. 이러한 설정은 Amazon RDS 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 \(p. 95\)](#) 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 마스터 인스턴스에서 binlog를 유지 관리합니다. 이렇게 유지 관리해야 오류 발생 시 마스터 인스턴스를 복원할 수 있습니다.

Important

자체 관리형 복제를 사용하는 경우, 발생할 수 있는 복제 문제를 직접 모니터링하고 해결해야 합니다. 자세한 정보는 [Read Replica 사이의 지역 문제 진단 및 해결 \(p. 1031\)](#) 단원을 참조하십시오.

Note

Amazon Aurora MySQL DB 클러스터에서 복제를 시작하는 데 필요한 권한이 제한되고 Amazon RDS 마스터 사용자를 사용할 수 없습니다. 이 때문에 Amazon RDS `mysql_rds_set_external_master` 및 `mysql_rds_start_replication` 절차를 사용하여 Amazon Aurora MySQL DB 클러스터와 MySQL DB 인스턴스 간 복제를 설정해야 합니다.

외부 마스터 인스턴스와 Amazon RDS 상의 MySQL DB 인스턴스 사이에서 복제 시작

- 원본 MySQL DB 인스턴스를 읽기 전용으로 설정합니다.

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

- 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

- `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 Amazon RDS MySQL 또는 MariaDB DB 인스턴스로 데이터 가져오기](#) 절차를 사용하려고 할 수도 있습니다.

Linux, OS X, Unix의 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>
```

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.

mysql 명령의 --host, --user (-u), --port 및 -p 옵션을 사용하여 Aurora DB 클러스터에 연결 할 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.com입니다. 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;
```

복제와 함께 사용할 백업을 만드는 자세한 정보는 MySQL 문서의 [Backing Up a Master or Slave by Making It Read Only](#) 단원을 참조하십시오.

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 정보는 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'@'mydomain.com' IDENTIFIED BY '<password>';
```

7. 외부 MySQL 인스턴스의 경우 복제 사용자에게 REPLICATION CLIENT 및 REPLICATION SLAVE 권한을 부여합니다. 예를 들어 도메인의 'repl_user' 사용자를 위해 모든 데이터베이스에서 REPLICATION CLIENT 및 REPLICATION SLAVE 권한을 부여하려면 다음 명령을 실행합니다.

```
GRANT REPLICATION CLIENT, REPLICATION SLAVE ON *.* TO 'repl_user'@'mydomain.com'  
IDENTIFIED BY '<password>;'
```

8. 복제를 설정하기 전에 Aurora MySQL DB 클러스터의 수동 스냅샷을 복제 슬레이브로 생성합니다. 복제 슬레이브인 DB 클러스터와 복제를 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. 마스터 사용자로서 Amazon Aurora DB 클러스터에 연결하고 `mysql_rds_set_external_master` 절차를 사용하여 원본 MySQL 데이터베이스를 식별하십시오. 2단계에서 결정한 마스터 로그 파일 이름과 마스터 로그 위치를 사용합니다. 다음은 예제입니다.

```
CALL mysql.rds_set_external_master ('mymasterserver.mydomain.com', 3306,  
'repl_user', '<password>', 'mysql-bin-changelog.000031', 107, 0);
```

- 10 Amazon Aurora DB 클러스터에서 `mysql_rds_start_replication` 절차를 실행하여 복제를 시작하십시오.

```
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. 그리고 나면 Aurora 복제본에 연결하여 데이터에 대한 읽기 조정을 수행할 수 있습니다. Aurora 복제본 생성에 대한 자세한 정보는 [DB 클러스터에 Aurora 복제본 추가 \(p. 208\)](#) 단원을 참조하십시오.

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 Aurora를 사용하려면 Amazon Aurora DB 클러스터를 생성한 후 이 클러스터를 MySQL DB 인스턴스의 복제 슬레이브로 설정합니다. 이러한 설정은 Amazon RDS 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 복제 슬레이브로 설정하는 방법에 대한 자세한 정보는 [Amazon Aurora를 사용하여 MySQL 데이터베이스 읽기 조정 \(p. 659\)](#)의 절차를 따르십시오.

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 [자동 중지 시간을 단축하여 Amazon RDS MySQL 또는 MariaDB DB 인스턴스로 데이터 가져오기](#) 절차를 사용하십시오. 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 Amazon RDS 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.

Amazon Aurora MySQL에서 XA 트랜잭션 사용

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.

Aurora MySQL에서 해시 조인 작업

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 double과 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.

- 좌/우 외부 조인은 지원되지 않습니다.
- 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 조인을 활성화하려면 MySQL 서버 변수 optimizer_switch를 on으로 설정하십시오. optimizer_switch 파라미터는 해시 조인이 기본적으로 on으로 설정됩니다. 다음은 해시 조인을 활성화하는 방법을 설명한 예제입니다.

```
mysql> SET optimizer_switch='hash_join=on';
```

이 설정을 사용하면 옵티マイ저는 비용, 쿼리 특성 및 리소스 가용성을 기반으로 해시 조인을 사용하도록 선택합니다. 비용 견적이 정확하지 않으면 옵티マイ저가 해시 조인을 선택하게 할 수 있습니다. 그렇게 하려면 MySQL 서버 변수 hash_join_cost_based를 off으로 설정합니다. 다음은 옵티マイ저가 해시 조인을 선택하도록 하는 방법을 설명한 예제입니다.

```
mysql> SET optimizer_switch='hash_join_cost_based=off';
```

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 table은 해시 테이블을 작성하는 데 사용하는 테이블이며 Hash Join Outer table은 해시 테이블을 프로브하는 데 사용하는 테이블입니다.

확장된 EXPLAIN 출력 형식에 대한 자세한 정보는 MySQL 제품 설명서에서 [Extended EXPLAIN Output Format](#)을 참조하십시오.

Aurora MySQL에서 외래 키 작업

`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 0로 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 `foreign_key_checks` 변수의 일부로 0 변수를 `init_connect`으로 설정하지 않아야 합니다.

- `mysqldump`와 같은 논리적 백업으로부터 복원이 실패하거나 불완전할 경우, 같은 세션에서 다른 작업을 시작하기 전에 `foreign_key_checks`가 1로 설정되어 있는지 확인합니다. 논리적 백업은 시작할 때 `foreign_key_checks`를 0으로 설정합니다.

관련 주제

- Amazon Aurora DB 클러스터 관리 (p. 190)

Amazon Aurora MySQL 참조

이 참조에는 Aurora MySQL 파라미터 및 상태 변수에 대한 정보가 포함되어 있습니다.

주제

- [Aurora MySQL 파라미터](#) (p. 665)
- [적용할 수 없는 MySQL 파라미터 및 상태 변수](#) (p. 677)
- [Aurora MySQL 이벤트](#) (p. 677)
- [Aurora MySQL 격리 수준](#) (p. 679)
- [Aurora MySQL 저장 프로시저](#) (p. 683)

Aurora MySQL 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 클러스터를 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 파라미터 그룹을 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 콘솔, 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.

DB 파라미터 그룹에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업](#) (p. 168) 단원을 참조하십시오. Aurora Serverless 클러스터에 대한 규칙 및 제한 사항은 [Aurora Serverless 및 파라미터 그룹](#) (p. 117) 단원을 참조하십시오.

주제

- [클러스터 수준 파라미터](#) (p. 666)
- [인스턴스 수준 파라미터](#) (p. 668)

클러스터 수준 파라미터

다음 표에는 전체 Aurora MySQL DB 클러스터에 적용되는 모든 파라미터가 나와 있습니다.

파라미터 이름	수정 가능	참고
aurora_enable_replica_log_compression	예	Aurora 글로벌 데이터베이스의 일부로 포함된 클러스터에는 적용되지 않습니다.
aurora_enable_repl_bin_log_filtering	예	Aurora 글로벌 데이터베이스의 일부로 포함된 클러스터에는 적용되지 않습니다.
aurora_enable_zdr	예	자세한 내용은 Amazon Aurora MySQL 복제를 위한 고가용성 고려 사항 (p. 556) 단원을 참조하십시오.
aurora_load_from_s3_role	예	자세한 내용은 Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 (p. 620) 단원을 참조하십시오.
aurora_select_into_s3_role	예	자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 (p. 627) 단원을 참조하십시오.
auto_increment_increment	예	
auto_increment_offset	예	
aws_default_lambda_role	예	자세한 내용은 Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출 (p. 633) 단원을 참조하십시오.
aws_default_s3_role	예	
binlog_checksum	예	
binlog_format	예	
binlog_row_image	아니요	
binlog_rows_query_log_events	아니요	
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	
collation_connection	예	
collation_server	예	

파라미터 이름	수정 가능	참고
completion_type	예	
default_storage_engine	아니요	Aurora MySQL 클러스터는 모든 데이터에 InnoDB 스토리지 엔진을 사용합니다.
innodb_autoinc_lock_mode	예	
innodb_checksums	아니요	
innodb_cmp_per_index_enabled	예	
innodb_commit_concurrency	예	
innodb_data_home_dir	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
innodb_file_per_table	예	
innodb_flush_log_at_trx_commit	예	
innodb_ft_max_token_size	예	
innodb_ft_min_token_size	예	
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_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	
innodb_sync_array_size	예	
innodb_sync_spin_loops	예	
innodb_table_locks	예	
innodb_undo_directory	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
innodb_undo_logs	예	

파라미터 이름	수정 가능	참고
innodb_undo_tablespaces	아니요	
lc_time_names	예	
lower_case_table_names	예	
master-info-repository	예	
master_verify_checksum	예	
server_audit_events	예	
server_audit_excl_users	예	
server_audit_incl_users	예	
server_audit_logging	예	로그를 Amazon CloudWatch Logs에 업로드하는 방법에 관한 지침은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 (p. 639) 단원을 참조하십시오.
server_id	아니요	
skip-character-set-client-handshake	예	
skip_name_resolve	아니요	
sync_frm	예	
time_zone	예	

인스턴스 수준 파라미터

다음 표에는 Aurora MySQL DB 클러스터의 특정 DB 클러스터에 적용되는 모든 파라미터가 나와 있습니다.

파라미터 이름	수정 가능	참고
allow-suspicious-udfs	아니요	
aurora_lab_mode	예	자세한 내용은 Amazon Aurora MySQL 랩 모드 (p. 654) 단원을 참조하십시오.
aurora_oom_response	예	자세한 내용은 Amazon Aurora MySQL 메모리 부족 문제 (p. 1031) 단원을 참조하십시오.
aurora_read_replica_read_committed	예	Aurora 복제본에 대해 READ COMMITTED 격리 수준을 활성화하고 격리 동작을 변경하여 장기 실행 쿼리 중 제거 지역 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 종 Aurora에서 테이블 데이터를 재편하므로

파라미터 이름	수정 가능	참고
		장기 실행 쿼리에서는 동일 행의 사본을 두 개 이상 반환할 수 있습니다. 자세한 내용은 Aurora MySQL 격리 수준 (p. 679) 단원을 참조하십시오.
<code>autocommit</code>	예	
<code>automatic_sp_privileges</code>	예	
<code>back_log</code>	예	
<code>basedir</code>	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
<code>binlog_cache_size</code>	예	
<code>binlog_max_flush_queue_time</code>	예	
<code>binlog_order_commits</code>	예	
<code>binlog_stmt_cache_size</code>	예	
<code>bulk_insert_buffer_size</code>	예	
<code>concurrent_insert</code>	예	
<code>connect_timeout</code>	예	
<code>core-file</code>	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
<code>datadir</code>	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
<code>default_time_zone</code>	아니요	
<code>default_tmp_storage_engine</code>	예	
<code>default_week_format</code>	예	
<code>delay_key_write</code>	예	
<code>delayed_insert_limit</code>	예	
<code>delayed_insert_timeout</code>	예	
<code>delayed_queue_size</code>	예	
<code>div_precision_increment</code>	예	
<code>end_markers_in_json</code>	예	
<code>enforce_gtid_consistency</code>	가끔	Aurora MySQL 2.04 이후 버전에서 수정 가능.
<code>eq_range_index_dive_limit</code>	예	

파라미터 이름	수정 가능	참고
event_scheduler	예	
explicit_defaults_for_timestamp	예	
flush	아니요	
flush_time	예	
ft_boolean_syntax	아니요	
ft_max_word_len	예	
ft_min_word_len	예	
ft_query_expansion_limit	예	
ft_stopword_file	예	
general_log	예	로그를 CloudWatch Logs에 업로드하는 것에 관한 자침은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 (p. 639) 단원을 참조하십시오.
general_log_file	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
group_concat_max_len	예	
gtid-mode	가끔	Aurora MySQL 2.04 이후 버전에서 수정 가능.
host_cache_size	예	
init_connect	예	
innodb_adaptive_hash_index	예	
innodb_adaptive_max_sleep_delay	예	Aurora에 대해 innodb_thread_concurrency는 항상 0 이므로 이 파라미터를 수정해도 효과가 없습니다.
innodb_autoextend_increment	예	
innodb_buffer_pool_dump_at_shutdown	아니요	
innodb_buffer_pool_dump_now	아니요	
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	

파라미터 이름	수정 가능	참고
innodb_change_buffer_max_size	아니요	Aurora MySQL은 InnoDB 변경 버퍼를 전혀 사용하지 않습니다.
innodb_compression_failure_threshold_pct	아니요	
innodb_compression_level	예	
innodb_compression_pad_pct_max	예	
innodb_concurrency_tickets	예	Aurora에 대해 innodb_thread_concurrency는 항상 0 이므로 이 파라미터를 수정해도 효과가 없습니다.
innodb_file_format	예	
innodb_flush_log_at_timeout	아니요	
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	
innodb_lock_wait_timeout	예	
innodb_log_compressed_pages	아니요	
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	
innodb_random_read_ahead	예	

파라미터 이름	수정 가능	참고
innodb_read_ahead_threshold	예	
innodb_read_io_threads	아니요	
innodb_read_only	아니요	Aurora MySQL은 클러스터 유형에 따라 읽기 전용 및 읽기-쓰기 상태의 DB 인스턴스를 관리합니다. 예를 들어 프로비저닝된 클러스터는 읽기-쓰기 상태의 DB 인스턴스(기본 인스턴스)를 관리하며, 그 밖에 다른 클러스터 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_concurrency는 항상 0 이므로 이 파라미터를 수정해도 효과가 없습니다.
interactive_timeout	예	
join_buffer_size	예	
keep_files_on_create	예	
key_buffer_size	예	
key_cache_age_threshold	예	
key_cache_block_size	예	
key_cache_division_limit	예	
local_infile	예	
lock_wait_timeout	예	
log-bin	아니요	
log_bin_trust_function_creators	예	
log_bin_use_v1_row_events	예	
log_error	아니요	

파라미터 이름	수정 가능	참고
log_output	예	
log_queries_not_using_indexes	예	
log_slave_updates	아니요	
log_throttle_queries_not_using_indexes	예	
log_warnings	예	
long_query_time	예	
low_priority_updates	예	
max_allowed_packet	예	
max_binlog_cache_size	예	
max_binlog_size	아니요	
max_binlog_stmt_cache_size	예	
max_connect_errors	예	
max_connections	예	자세한 내용은 Aurora MySQL DB 인스턴스에 대한 최대 연결 (p. 508) 단원을 참조하십시오.
max_delayed_threads	예	
max_error_count	예	
max_heap_table_size	예	
max_insert_delayed_threads	예	
max_join_size	예	
max_length_for_sort_data	예	
max_prepared_stmt_count	예	
max_seeks_for_key	예	
max_sort_length	예	
max_sp_recursion_depth	예	
max_tmp_tables	예	
max_user_connections	예	
max_write_lock_count	예	
metadata_locks_cache_size	예	
min_examined_row_limit	예	
myisam_data_pointer_size	예	
myisam_max_sort_file_size	예	

파라미터 이름	수정 가능	참고
myisam_mmap_size	예	
myisam_sort_buffer_size	예	
myisam_stats_method	예	
myisam_use_mmap	예	
net_buffer_length	예	
net_read_timeout	예	
net_retry_count	예	
net_write_timeout	예	
old-style-user-limits	예	
old_passwords	예	
optimizer_prune_level	예	
optimizer_search_depth	예	
optimizer_switch	예	
optimizer_trace	예	
optimizer_trace_features	예	
optimizer_trace_limit	예	
optimizer_trace_max_mem_size	예	
optimizer_trace_offset	예	
performance_schema	예	
pid_file	아니요	
plugin_dir	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
port	아니요	Aurora MySQL은 연결 속성을 관리하여 클러스터에 속한 모든 DB 인스턴스에 일관된 설정을 적용합니다.
preload_buffer_size	예	
profiling_history_size	예	
query_alloc_block_size	예	
query_cache_limit	예	
query_cache_min_res_unit	예	
query_cache_size	예	
query_cache_type	예	

파라미터 이름	수정 가능	참고
<code>query_cache_wlock_invalidate</code>	예	
<code>query_prealloc_size</code>	예	
<code>range_alloc_block_size</code>	예	
<code>read_buffer_size</code>	예	
<code>read_only</code>	아니요	Aurora MySQL은 클러스터 유형에 따라 읽기 전용 및 읽기-쓰기 상태의 DB 인스턴스를 관리합니다. 예를 들어 프로비저닝된 클러스터는 읽기-쓰기 상태의 DB 인스턴스(기본 인스턴스)를 관리하며, 그 밖에 다른 클러스터 인스턴스는 읽기 전용(Aurora 복제본)입니다.
<code>read_rnd_buffer_size</code>	예	
<code>relay-log</code>	아니요	
<code>relay_log_info_repository</code>	예	
<code>relay_log_recovery</code>	아니요	
<code>safe-user-create</code>	예	
<code>secure_auth</code>	예	
<code>secure_file_priv</code>	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
<code>skip-slave-start</code>	아니요	
<code>skip_external_locking</code>	아니요	
<code>skip_show_database</code>	예	
<code>slave_checkpoint_group</code>	예	
<code>slave_checkpoint_period</code>	예	
<code>slave_parallel_workers</code>	예	
<code>slave_pending_jobs_size_max</code>	예	
<code>slave_sql_verify_checksum</code>	예	
<code>slow_launch_time</code>	예	
<code>slow_query_log</code>	예	로그를 CloudWatch Logs에 업로드하는 것에 관한 지침은 Amazon CloudWatch Logs에 Amazon Aurora MySQL 로그 게시 (p. 639) 단원을 참조하십시오.
<code>slow_query_log_file</code>	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
<code>socket</code>	아니요	

파라미터 이름	수정 가능	참고
sort_buffer_size	예	
sql_mode	예	
sql_select_limit	예	
stored_program_cache	예	
sync_binlog	아니요	
sync_master_info	예	
sync_relay_log	예	
sync_relay_log_info	예	
sysdate-is-now	예	
table_cache_element_entry_ttl	아니요	
table_definition_cache	예	
table_open_cache	예	
table_open_cache_instances	예	
temp_pool	예	
thread_handling	아니요	
thread_stack	예	
timed_mutexes	예	
tmp_table_size	예	
tmpdir	아니요	Aurora MySQL은 관리형 인스턴스를 사용하기 때문에 사용자가 직접 파일 시스템에 액세스하지 않습니다.
transaction_alloc_block_size	예	
transaction_prealloc_size	예	
tx_isolation	예	
updatable_views_with_limit	예	
validate_password	아니요	
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	

파라미터 이름	수정 가능	참고
wait_timeout	예	

적용할 수 없는 MySQL 파라미터 및 상태 변수

Aurora MySQL과 MySQL 간의 아키텍처 차이 때문에 일부 MySQL 파라미터와 상태 변수는 Aurora MySQL에 적용되지 않습니다.

다음 MySQL 파라미터는 Aurora MySQL에 적용되지 않습니다.

- innodb_adaptive_flushing
- innodb_adaptive_flushing_lwm
- innodb_change_buffering
- innodb_checksum_algorithm
- innodb_doublewrite
- 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_max_dirty_pages_pct
- innodb_use_native_aio
- innodb_write_io_threads
- thread_cache_size

다음 MySQL 상태 변수는 Aurora MySQL에 적용되지 않습니다.

- innodb_buffer_pool_bytes_dirty
- innodb_buffer_pool_pages_dirty
- innodb_buffer_pool_pages_flushed

Note

이러한 목록은 포괄적이지 않습니다.

Aurora MySQL 이벤트

다음은 Aurora MySQL의 일반 대기 이벤트입니다.

Note

MySQL 대기 이벤트에 사용되는 이름 지정 규칙에 대한 자세한 내용은 MySQL 설명서의 [성능 스키마 장비 이름 지정 규칙](#)을 참조하십시오.

io/aurora_redo_log_flush

이 대기 이벤트에서는 세션이 Aurora 스토리지에 데이터를 쓰고 있습니다. 일반적으로 이 대기 이벤트는 Aurora MySQL의 쓰기 I/O 작업을 위한 것입니다.

io/aurora_respond_to_client

이 대기 이벤트에서는 스레드가 클라이언트에 결과 집합을 반환하고 있습니다.

io/file/csv/data

이 대기 이벤트에는 쉼표로 구분된 값(CSV) 형식으로 테이블에 쓰는 스레드가 있습니다. CSV 테이블 사용을 확인하십시오. 일반적으로 이 이벤트는 테이블에서 log_output을 설정하는 중에 발생합니다.

io/file/innodb/innodb_data_file

이 대기 이벤트에서는 스토리지에 대한 I/O 작업을 대기 중인 스레드가 있습니다. 이 이벤트는 I/O 집약적인 워크로드에서 일반적으로 발생합니다. 이 대기 이벤트가 비교적 큰 비율로 나타나는 SQL 문은 디스크 집약적인 쿼리를 실행하는 중일 수 있습니다. 또는 InnoDB 버퍼풀에서는 충족할 수 없는 데이터를 요청하는 중일 수 있습니다. 이를 알아내려면 쿼리 계획과 캐시 적중률을 확인해야 합니다. 자세한 내용은 MySQL 설명서의 [버퍼링 및 캐싱](#)을 참조하십시오.

io/file/sql/binlog

이 대기 이벤트에서는 binlog 파일에서 디스크에 쓰는 동안 대기 중인 스레드가 있습니다.

io/socket/sql/client_connection

이 대기 이벤트에서는 스레드가 새 연결을 처리하고 있습니다.

io/table/sql/handler

이 이벤트는 테이블 I/O 대기 이벤트입니다. 일반적으로 이러한 이벤트가 발생한 이후에는 파일 I/O 이벤트와 같은 중첩 이벤트가 발생할 수 있습니다. 성능 스키마의 'atom' 및 'molecule' 이벤트에 대한 자세한 내용은 MySQL 설명서의 [성능 스키마 Atom 및 Molecule 이벤트](#)를 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 이벤트](#)를 참조하십시오.

synch/cond/mysys/my_thread_var::suspend

이 대기 이벤트에서는 특정 조건에서 스레드가 대기 중일 때 스레드가 일시 중지됩니다. 예를 들어, 스레드가 테이블 수준 잠금을 대기 중일 때 이 이벤트가 발생합니다. 워크로드를 조사하여 DB 인스턴스에서 테이블 잠금을 획득 중인 스레드를 확인하는 것이 좋습니다. MySQL의 테이블 잠금에 대한 자세한 내용은 MySQL 설명서의 [테이블 잠금 문제](#)를 참조하십시오.

synch/cond/sql/MDL_context::COND_wait_status

이 대기 이벤트에서는 테이블 메타데이터 잠금을 대기 중인 스레드가 있습니다. 자세한 내용은 MySQL 설명서의 [잠금 작업 최적화](#) 단원을 참조하십시오.

synch/cond/sql/MYSQL_BIN_LOG::COND_done

이 대기 이벤트에서는 binlog 파일에서 디스크에 쓰는 동안 대기 중인 스레드가 있습니다. 변경률이 매우 높은 데이터베이스에서 바이너리 로깅 경합이 발생할 수 있습니다.

synch/mutex/innodb/aurora_lock_thread_slot_futex

이 대기 이벤트에 InnoDB 레코드 잠금을 대기 중인 스레드가 있습니다. 이 이벤트가 표시될 경우 데이터베이스에서 워크로드 충돌을 확인하십시오. 자세한 내용은 MySQL 설명서의 [InnoDB 잠금](#)을 참조하십시오.

synch/mutex/innodb/buf_pool_mutex

이 대기 이벤트에서는 스레드가 InnoDB 버퍼풀에서 잠금을 획득했습니다.

synch/mutex/sql/LOCK_open

이 대기 이벤트에서는 LOCK_open을 사용하여 데이터 딕셔너리에서 다양한 객체를 보호하고 있습니다. 이 대기 이벤트는 이러한 잠금을 획득하기 위해 대기 중인 스레드가 있음을 나타냅니다. 일반적으로 이 이벤트는 데이터 딕셔너리 경합으로 인해 발생합니다.

synch/mutex/sql/LOCK_table_cache

이 대기 이벤트에서는 테이블 캐시 인스턴스에 대한 잠금 획득을 대기 중인 스레드가 있습니다. 자세한 내용은 MySQL 설명서의 [MySQL에서 테이블을 열고 닫는 방법](#)을 참조하십시오.

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 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 로그](#) 단원을 참조하십시오.

synch/mutex/sql/MYSQL_BIN_LOG::LOCK_log

이 대기 이벤트에서는 스레드가 바이너리 로그 파일을 능동적으로 잠그고 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.

synch/rwlock/innodb/dict

이 대기 이벤트에서는 InnoDB 데이터 딕셔너리에 보관된 rwlock을 대기 중인 스레드가 있습니다.

synch/rwlock/innodb/dict sys RW lock

이 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.

Aurora MySQL 격리 수준

아래에서 Aurora MySQL 클러스터의 DB 인스턴스가 격리의 데이터베이스 속성을 구현하는 방식을 알아볼 수 있습니다. 이를 통해 Aurora MySQL 기본 동작이 엄격한 일관성과 높은 성능 사이에 균형을 이루는 방식을 이해할 수 있습니다. 또한 워크로드의 특성에 따라 언제 기본 설정을 변경할지 결정할 수 있습니다.

라이터 인스턴스에 사용 가능한 격리 수준

Aurora MySQL 단일 마스터 클러스터의 기본 인스턴스에서 또는 Aurora MySQL 멀티 마스터 클러스터의 DB 인스턴스에서 REPEATABLE READ, READ COMMITTED, READ UNCOMMITTED, SERIALIZABLE 격리 수준을 사용할 수 있습니다. 이러한 격리 수준은 RDS 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 SHOW ENGINE INNODB STATUS 명령의 출력에서 history list length의 값이 커지는 것입니다. CloudWatch의 RollbackSegmentHistoryListLength 지표를

사용하여 이 값을 모니터링할 수 있습니다. 이 조건으로 인해 보조 인덱스의 효율성이 떨어져 전반적인 쿼리 성능이 하락하고 스토리지 공간이 낭비될 수 있습니다.

이러한 문제가 발생하면 Aurora MySQL 구성 설정인 `aurora_read_replica_read_committed`를 사용하여 Aurora 복제본에서 READ COMMITTED 격리 수준을 사용할 수 있습니다. 이 설정을 사용하면 장기 실행 쿼리를 테이블을 수정하는 트랜잭션과 동시에 수행함으로 인해 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 Amazon RDS 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 사용 사례에서는 `aurora_read_replica_read_committed` 설정을 사용할 수 있습니다. 정밀도 및 반복성이 중요하고 결과 집합이 작은 짧은 쿼리에는 이 설정을 사용하지 않는 것이 좋습니다.

리더에 대해 READ COMMITTED 활성화

Aurora 복제본에 대해 READ COMMITTED 격리 수준을 활성화하려면 `aurora_read_replica_read_committed` 구성 설정을 활성화하십시오. 특정 Aurora 복제본에 연결된 상태에서 이 설정을 세션 수준에서 활성화하십시오. 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 활성화하여 대화형 즉석(일회성) 쿼리를 수행할 수 있습니다. 또한 다른 애플리케이션에 대한 기본값을 변경하지 않은 상태에서 READ COMMITTED 격리 수준에서 이점을 얻는 보고 또는 데일리 분석 애플리케이션을 실행할 수 있습니다.

클러스터에서 1개 이상의 Aurora 복제본에 대해 이 기능을 영구적으로 활성화할 수 있습니다. 이렇게 하려면 연결된 DB 인스턴스에서 사용하는 DB 파라미터 그룹에서 `aurora_read_replica_read_committed` 설정을 켜십시오. 장기 실행 쿼리를 실행하는 DB 인스턴스에서 이 설정을 활성화하십시오. 이 경우 인스턴스 앤드포인트에 연결하여 쿼리가 원하는 DB 인스턴스에서 실행되게 하십시오.

`aurora_read_replica_read_committed` 설정이 활성화된 경우 `SET TRANSACTION ISOLATION LEVEL` 명령을 사용하여 적절한 트랜잭션에 격리 수준을 지정하십시오.

```
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 인스턴스 또는 멀티 마스터 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 `DELETE`와 같은 SQL 문

에서 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	<code>INSERT INTO big_table SELECT * FROM other_table LIMIT 1000000; COMMIT;</code>		
T2		Q1: <code>SELECT COUNT(*) FROM big_table;</code>	Q2: <code>SELECT COUNT(*) FROM big_table;</code>
T3	<code>INSERT INTO big_table (c1, c2) VALUES (1, 'one more row'); COMMIT;</code>		
T4		Q1이 지금 완료되면 결과는 1,000,000입니다.	Q2가 지금 완료되면 결과는 1,000,000 또는 1,000,001입니다.
T5	<code>DELETE FROM big_table LIMIT 2; COMMIT;</code>		
T6		Q1이 지금 완료되면 결과는 1,000,000입니다.	Q2가 지금 완료되면 결과는 1,000,000 또는

시간	Aurora 기본 인스턴스의 DML 문	READ COMMITTED로 Aurora 기본 인스턴스에 대해 쿼리	READ COMMITTED로 Aurora 복제본에 대해 쿼리
			1,000,001 또는 999,999 또는 999,998입니다.
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 999,999 이거나 이보다 더 높은 수일 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;
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.

기본 인스턴스의 READ COMMITTED에서는 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 COMMITTED인 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 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 있습니다. 따라서 행 수는 예상과 다를 수 있습니다.

Q5 및 Q6의 결과는 서로 같거나 약간 다를 수 있습니다. READ COMMITTED에서 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 저장 프로시저

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. Aurora MySQL에서 전역 트랜잭션 식별자(GTID)에 근거하여 복제를 사용하는 방법을 알아보려면 [Aurora MySQL \(p. 580\)](#) 단원을 참조하십시오.

주제

- [mysql.rds_set_master_auto_position \(p. 683\)](#)
- [mysql.rds_set_external_master_with_auto_position \(p. 684\)](#)
- [mysql.rds_skip_transaction_with_gtid \(p. 685\)](#)

[mysql.rds_set_master_auto_position](#)

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 0입니다.
- 1 – GTID 기반 복제 방법을 사용합니다.

사용 시 주의 사항

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

마스터 사용자는 `mysql.rds_set_master_auto_position` 절차를 실행해야 합니다.

Aurora의 경우, 이 절차는 Aurora MySQL 2.04 버전과 MySQL 5.7-호환 버전에서 지원됩니다. Aurora MySQL 1.1 또는 1.0에서는 GTID 기반 복제가 지원되지 않습니다.

mysql.rds_set_external_master_with_auto_position

Aurora MySQL 기본 인스턴스가 외부 MySQL 인스턴스에서 수신되는 복제를 수락하도록 구성하십시오. 또한 이 절차에서는 전역 트랜잭션 식별자(GTID)를 기반으로 한 복제를 구성합니다.

이 절차는 Amazon RDS MySQL과 Aurora MySQL 모두에서 사용할 수 있으며, 컨텍스트에 따라 다르게 작동합니다. Aurora MySQL에서 사용하는 경우, 이 절차에서는 지연된 복제를 구성하지 않습니다. 이러한 제한 사항이 있는 이유는 Amazon RDS MySQL에서는 지연된 복제를 지원하지만 Aurora MySQL에서는 지원하지 않기 때문입니다.

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 마스터가 될 Aurora의 외부에서 실행 중인 MySQL 인스턴스의 호스트 이름 또는 IP 주소입니다.

host_port

복제 마스터로 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 지정하십시오.

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_name에 지정된 사용자 ID의 암호입니다.

ssl_encryption

이 옵션은 현재 구현되지 않습니다. 기본값은 0입니다.

사용 시 주의 사항

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

마스터 사용자는 `mysql.rds_set_external_master_with_auto_position` 절차를 실행해야 합니다. 마스터 사용자는 복제 대상의 역할을 하는 Aurora MySQL DB 클러스터의 기본 인스턴스에서 이 절차를 실행합니다. 이것은 외부 MySQL DB 인스턴스 또는 Aurora MySQL DB 클러스터의 복제 대상이 될 수 있습니다.

Aurora의 경우, 이 절차는 Aurora MySQL 2.04 버전과 MySQL 5.7 호환 버전에서 지원됩니다. Aurora MySQL 1.1 또는 1.0에서는 GTID 기반 복제가 지원되지 않습니다.

`mysql.rds_set_external_master_with_auto_position`을 실행하기 전에 외부 MySQL DB 인스턴스가 복제 마스터가 되도록 구성합니다. 외부 MySQL 인스턴스에 연결하려면 `replication_user_name`

및 replication_user_password의 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 마스터로 구성하려면

- 선택한 MySQL 클라이언트를 사용하여 외부 MySQL 인스턴스에 연결하고 복제에 사용할 사용자 계정을 생성합니다. 다음은 예제입니다.

```
CREATE USER 'repl_user'@'mydomain.com' IDENTIFIED BY 'SomePassW0rd'
```

- 외부 MySQL 인스턴스의 경우 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'  
IDENTIFIED BY 'SomePassW0rd'
```

`mysql.rds_set_external_master_with_auto_position`을 호출하면 Amazon RDS는 특정 정보를 기록합니다. 이 정보는 `mysql.rds_history` 및 `mysql.rds_replication_status` 테이블에 있는 "set master"의 시간, 사용자 및 작업입니다.

문제의 원인으로 알려진 특정 GTID 기반 트랜잭션을 건너뛰려면

[mysql.rds_skip_transaction_with_gtid \(p. 685\)](#) 저장 프로시저를 사용할 수 있습니다. GTID 기반 복제 작업에 대한 자세한 내용은 [Aurora MySQL \(p. 580\)](#) 단원을 참조하십시오.

예

Aurora 기본 인스턴스에서 다음 예제를 실행하면 Aurora 클러스터가 Aurora 외부에서 실행 중인 MySQL 인스턴스의 읽기 전용 복제본의 역할을하도록 구성됩니다.

```
call mysql.rds_set_external_master_with_auto_position(  
    'Externaldb.some.com',  
    3306,  
    'repl_user'@'mydomain.com',  
    'SomePassW0rd');
```

mysql.rds_skip_transaction_with_gtid

Aurora 기본 인스턴스에서 지정된 전역 트랜잭션 식별자(GTID)를 사용하여 트랜잭션 복제를 건너뜁니다.

특정 GTID 트랜잭션이 문제의 원인으로 알려진 경우 재해 복구를 위해 이 절차를 사용할 수 있습니다. 이 저장 절차를 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 시 주의 사항

Aurora MySQL DB 클러스터의 경우, 기본 인스턴스에 연결된 상태에서 이 저장 프로시저를 호출합니다.

마스터 사용자는 `mysql.rds_skip_transaction_with_gtid` 절차를 실행해야 합니다.

Aurora의 경우, 이 절차는 Aurora MySQL 2.04 버전과 MySQL 5.7 호환 버전에서 지원됩니다. Aurora MySQL 1.1 또는 1.0에서는 GTID 기반 복제가 지원되지 않습니다.

Amazon Aurora MySQL 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.

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. [AWS Management 콘솔](#)에서 유지 관리 기간 설정을 확인하거나 변경할 수 있습니다.

Aurora MySQL 버전

MySQL과 호환되는 Aurora은 MySQL과 호환되는 데이터베이스 엔진이지만 Aurora MySQL에는 Aurora MySQL에서만 사용할 수 있는 기능 및 Aurora MySQL DB 클러스터에만 제공되는 기능이 포함되어 있습니다.

Aurora 버전은 `<### ##>.<### ##>.<## ##>` 형식을 사용합니다. `AURORA_VERSION` 시스템 변수를 쿼리하여 Aurora 인스턴스 버전을 확인할 수 있습니다. Aurora 버전을 확인하려면 다음 쿼리 중 하나를 사용하십시오.

```
select AURORA_VERSION();
select @@aurora_version;
```

Aurora MySQL 엔진 버전

Aurora MySQL 2.03.2부터는 Aurora 엔진 버전에 다음 구문이 제공됩니다.

```
<mysql-major-version>.mysql_aurora.<aurora-mysql-version>
```

예를 들어 Aurora MySQL 2.03.2의 엔진 버전은 다음과 같습니다.

```
5.7.mysql_aurora.2.03.2
```

Note

모든 1.x Aurora MySQL 엔진 버전은 Community MySQL 5.6.10a와 유선 호환됩니다. 모든 2.x Aurora MySQL 엔진 버전은 Community MySQL 5.7.12와 유선 호환됩니다.

Aurora MySQL 2.x의 경우 Aurora MySQL 버전 2.03.1 이하에 대한 엔진 버전은 5.7.12입니다.
Aurora MySQL 1.x의 경우 1.19.0 이전의 Aurora MySQL에 대한 엔진 버전은 5.6.10a입니다.

일부 AWS CLI 명령 및 RDS API 작업에서 Aurora MySQL 엔진 버전을 지정할 수 있습니다. 예를 들면 AWS CLI 명령 [create-db-cluster](#) 및 [modify-db-cluster](#) 실행 시 --engine-version 옵션을 지정할 수 있습니다. RDS API 작업 [CreateDBCluster](#) 및 [ModifyDBCluster](#) 실행 시 EngineVersion 파라미터를 지정할 수 있습니다.

Aurora MySQL 1.19.0 및 2.03.2 이전의 엔진 버전을 업데이트하는 프로세스는 클러스터에서 보류 중인 유지 관리 작업을 적용하는 것입니다. 이 프로세스에서는 AWS Management 콘솔에 표시되는 Aurora MySQL 엔진 버전이 변경되지 않습니다.

Aurora MySQL 1.19.0 및 2.03.2 이전 버전의 경우, AWS Management 콘솔에 표시된 엔진 버전이 클러스터에서 엔진을 업그레이드한 후에도 동일하게 유지됩니다. Aurora MySQL 1.19.0 이상 또는 2.03.2 이상에서는 AWS Management 콘솔의 엔진 버전에 Aurora 버전도 포함됩니다. 클러스터를 업그레이드하면 표시된 값이 변경됩니다. AWS CloudFormation을 통해 관리되는 Aurora 클러스터의 경우, EngineVersion 설정의 이 변경 사항은 AWS CloudFormation에 의한 작업을 트리거할 수 있습니다. AWS CloudFormation에서 EngineVersion 설정에 대한 변경 내용을 처리하는 방식에 대한 자세한 내용은 [AWS CloudFormation 설명서](#)를 참조하십시오.

Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치

DB 클러스터의 마이너 버전을 업그레이드하거나 DB 클러스터에 패치를 적용하는 방법에는 두 가지가 있습니다.

- [엔진 버전 수정 \(p. 687\)](#)(버전 2.03.2 이상으로만 업그레이드)
- [보류 중인 유지 관리를 Aurora MySQL DB 클러스터에 적용 \(p. 688\)](#)

엔진 버전 수정

Amazon Aurora MySQL 버전 2.03.2 이상으로 업그레이드하는 경우 DB 클러스터의 마이너 버전을 업그레이드할 수 있습니다. 이 작업을 수행하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 DB 클러스터의 엔진 버전을 수정하면 됩니다.

DB 클러스터의 엔진 버전을 수정하려면

- Amazon RDS 콘솔 사용 – 다음 단계를 수행합니다.
 1. Amazon RDS 콘솔에 로그인하고 Databases(데이터베이스)를 선택합니다.
 2. 수정할 DB 클러스터를 선택합니다.
 3. 수정을 선택합니다.
 4. DB 엔진 버전 상자에서 Aurora MySQL 엔진 버전을 변경합니다.
 5. [Continue]를 수정 사항을 요약한 내용을 확인합니다.
 6. (선택 사항) 즉시 적용을 선택하여 변경 내용을 즉시 적용합니다.
 7. 확인 페이지에서 클러스터 수정을 선택합니다.
- AWS CLI 사용 – [modify-db-cluster](#) AWS CLI 명령을 호출한 후, --db-cluster-identifier 옵션에 DB 클러스터 이름을 지정하고 --engine-version 옵션에 엔진 버전을 지정합니다.

예를 들면 Aurora MySQL 버전 2.03.2로 업그레이드하려면 --engine-version 옵션을 5.7.mysql_aurora.2.03.2로 설정합니다. DB 클러스터의 엔진 버전을 즉시 업데이트하려면 --apply-immediately 옵션을 설정합니다.

- Amazon RDS API 사용 – [ModifyDBCluster](#) API 작업을 호출한 후 DBClusterIdentifier 파라미터에 DB 클러스터 이름을 지정하고 EngineVersion 파라미터에 엔진 버전을 지정합니다. DB 클러스터의 엔진 버전을 즉시 업데이트하려면 ApplyImmediately 파라미터를 true로 설정합니다.

보류 중인 유지 관리를 Aurora MySQL DB 클러스터에 적용

Aurora MySQL 버전 1.x 버전으로 업그레이드하는 경우 새 데이터베이스 엔진 버전 및 패치가 DB 클러스터의 사용 가능한 유지 관리 업그레이드로 표시됩니다. 사용 가능한 유지 관리 작업을 적용하여 DB 클러스터의 데이터베이스 버전을 업그레이드하거나 패치를 적용할 수 있습니다. 먼저 프로덕션 이외 DB 클러스터에서 업데이트를 적용하여 새 버전의 변경 사항이 인스턴스 및 애플리케이션에 어떤 영향을 주는지 알아보는 것이 좋습니다.

보류되었던 유지 관리 작업을 적용하려면

- Amazon RDS 콘솔 사용 – 다음 단계를 수행합니다.
 1. Amazon RDS 콘솔에 로그인하고 Databases(데이터베이스)를 선택합니다.
 2. [available] 유지 관리 업그레이드를 표시하는 DB 클러스터를 선택합니다.
 3. DB 클러스터의 데이터베이스 버전을 즉시 업데이트하려면 작업에서 지금 업그레이드를 선택하고, 다음 번 DB 클러스터 유지 관리 기간 동안 DB 클러스터에 대한 데이터베이스 버전을 업데이트하려면 다음에 업그레이드를 선택합니다.
- AWS CLI를 사용하여 `apply-pending-maintenance-action` AWS CLI 명령을 호출하고 `--resource-id` 옵션에 대해 DB 클러스터의 Amazon 리소스 이름(ARN)을 지정하고, `--apply-action` 옵션을 `system-update`로 지정합니다. DB 클러스터의 데이터베이스 버전을 즉시 업데이트하려면 `--opt-in-type` 옵션을 `immediate`로 설정하고, 다음 번 클러스터 유지 관리 기간에 DB 클러스터의 데이터베이스 버전을 업데이트하려면 `next-maintenance`로 설정합니다.
- Amazon RDS API 사용 – `ApplyPendingMaintenanceAction` API 작업을 호출하고 `ResourceId` 파라미터에 DB 클러스터의 ARN을 지정하고, `ApplyAction` 파라미터에 `system-update`를 지정합니다. DB 클러스터의 데이터베이스 버전을 즉시 업데이트하려면 `OptInType` 파라미터를 `immediate`로 설정하고, 다음 번 클러스터 유지 관리 기간에 인스턴스의 데이터베이스 버전을 업데이트하려면 `next-maintenance`로 설정합니다.

Amazon RDS가 데이터베이스와 운영 체제 업데이트를 관리하는 방법에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 Aurora MySQL 버전이 1.14.x이지만 1.14.4보다 낮은 경우 1.14.4로만 업그레이드할 수 있습니다(db.r4 인스턴스 클래스 지원). 또한 1.14.x에서 1.17과 같이 더 높은 Aurora MySQL 마이너 버전으로 업그레이드하려면 1.14.x 버전이 1.14.4여야 합니다.

제로 가동 중지 패치 적용

제로 가동 중지 패치 적용(ZDP) 기능은 최선을 다해 엔진 패치 도중 클라이언트 연결을 유지하기 위해 노력합니다. ZDP가 성공적으로 실행될 경우 패치 적용 도중 애플리케이션 세션이 유지되고 데이터베이스 엔진이 재시작합니다. 데이터베이스 엔진 재시작 시 처리량이 약 5초간 떨어질 수 있습니다. ZDP는 MySQL 5.6과 호환되는 Aurora MySQL 1.13 이상부터 사용할 수 있습니다. 또한 MySQL 5.7과 호환되는 Aurora MySQL 2.07 이상부터 사용할 수도 있습니다.

다음 조건에서는 ZDP가 성공적으로 실행되지 않을 수 있습니다.

- 장기간 쿼리 또는 트랜잭션이 진행 중인 경우 이 경우 Aurora가 ZDP를 수행할 수 있다면 열린 트랜잭션이 모두 취소됩니다.
- 바이너리 로깅이 활성화되어 있거나 바이너리 로그 복제가 진행 중인 경우
- 열린 SSL 연결이 존재하는 경우
- 임시 테이블 또는 테이블 잠금이 사용 중인 경우(예: DDL 문 실행 중) 이 경우 Aurora가 ZDP를 수행할 수 있다면 열린 트랜잭션이 모두 취소됩니다.
- 보류 중인 파라미터 변경 사항이 존재하는 경우

Aurora MySQL 1.19 및 2.07부터 ZDP 메커니즘이 개선되기 시작했습니다. 이러한 개선 사항 덕분에 ZDP는 장기 실행 트랜잭션, 바이너리 로깅, 테이블 잠금 또는 임시 테이블이 있을 때 성공할 가능성이 더 높아집니다.

이러한 조건 때문에 ZDP 실행을 위한 적절한 시간 창을 확보할 수 없는 경우 패치 적용이 표준 동작으로 돌아갑니다.

Note

- ZDP는 클러스터의 기본 DB 인스턴스에만 적용됩니다. Aurora 복제본에는 ZDP가 적용되지 않습니다.
- 준비된 문은 ZDP를 방지하지 않지만 ZDP 실행 이후에는 유지되지 않습니다.

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 업그레이드 할 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 1년 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 다음 요구 사항과 비즈니스 요구가 있는 클러스터를 대상으로 합니다.

- 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

이 Aurora 사용 설명서의 게시일 현재 Aurora MySQL의 현재 LTS 릴리스는 다음과 같습니다.

- Aurora MySQL 버전 . 이 버전에 대한 자세한 내용은 [??? 단원](#)을 참조하십시오.
- Aurora MySQL 버전 . 이 버전에 대한 자세한 내용은 [??? 단원](#)을 참조하십시오.

관련 주제

- [Amazon Aurora MySQL 2.0에 대한 데이터베이스 엔진 업데이트 \(p. 690\)](#)
- [Amazon Aurora MySQL 1.1에 대한 데이터베이스 엔진 업데이트 \(p. 722\)](#)
- [Aurora MySQL 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그 \(p. 763\)](#)

- Aurora 랩 모드 기능 (p. 655)

Amazon Aurora MySQL 2.0에 대한 데이터베이스 엔진 업데이트

다음은 Amazon Aurora 2.0 데이터베이스 엔진 업데이트입니다.

- Aurora MySQL 데이터베이스 엔진 업데이트 2019-12-23(버전 2.07.1) (p. 690)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 2.07.0) (p. 691)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-22(버전 2.06.0) (p. 693)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-11(버전 2.05.0) (p. 696)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-20(버전 2.04.8) (p. 698)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-14(버전 2.04.7) (p. 699)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-09-19(버전 2.04.6) (p. 701)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-07-08(버전 2.04.5) (p. 702)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-29(버전 2.04.4) (p. 704)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-09(버전 2.04.3) (p. 705)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-02(버전 2.04.2) (p. 706)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-03-25(버전 2.04.1) (p. 708)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-03-25(버전 2.04) (p. 708)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-02-07 (p. 709) (버전 2.03.4)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-01-18 (p. 710) (버전 2.03.3)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-01-09 (p. 711)(버전 2.03.2)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-24 (p. 711)(버전 2.03.1)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-11 (p. 712)(버전 2.03)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-08 (p. 713)(버전 2.02.5)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-09-21 (p. 713)(버전 2.02.4)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-08-23 (p. 714)(버전 2.02.3)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-06-04 (p. 715)(버전 2.02.2)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-05-03 (p. 717)(버전 2.02)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-03-13 (p. 719)(버전 2.01.1)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-02-06 (p. 721)(버전 2.01)

Aurora MySQL 데이터베이스 엔진 업데이트 2019-12-23(버전 2.07.1)

버전: 2.07.1

Aurora MySQL 2.07.1이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*입니다.

현재 지원되는 Aurora MySQL 릴리스에서 Aurora MySQL 2.07.1로 스냅샷을 복원할 수 있습니다. 기존 Aurora MySQL 2.* 데이터베이스 클러스터를 Aurora MySQL 2.07.1로 업그레이드할 수 있는 옵션도 있습니다. 기존 Aurora MySQL 1.* 클러스터를 2.07.1로 직접 업그레이드할 수는 없습니다. 하지만 이 클러스터의 스냅샷을 Aurora MySQL 2.07.1로 복원할 수 있습니다.

Aurora MySQL가 이전 버전인 클러스터를 생성하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1], 중동(바레인) [me-south-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

높은 우선 순위:

- Aurora 특정 데이터베이스 추적 및 로깅 하위 시스템에서 사용 가능한 메모리를 낮추는 느린 메모리 누수 문제가 수정되었습니다.

일반적인 안정성 수정 사항:

- 중간 테이블을 내부적으로 사용하는 다중 테이블 조인 및 집계가 포함된 복잡한 쿼리를 실행하는 동안 발생하는 충돌 문제가 수정되었습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 2.07.0)

버전: 2.07.0

Aurora MySQL 2.07.0이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*입니다.

현재 지원되는 Aurora MySQL 릴리스에서 Aurora MySQL 2.07.0으로 스냅샷을 복원할 수 있습니다. 기존 Aurora MySQL 2.* 데이터베이스 클러스터를 Aurora MySQL 2.07.0으로 업그레이드할 수 있는 옵션도 있습니다. 기존 Aurora MySQL 1.* 클러스터를 2.07.0으로 직접 업그레이드할 수는 없습니다. 하지만 이 클러스터의 스냅샷을 Aurora MySQL 2.07.0으로 복원할 수 있습니다.

Aurora MySQL가 이전 버전인 클러스터를 생성하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1], 중동(바레인) [me-south-1], 남아메리카(상파울루) [sa-east-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표 할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

새로운 기능:

- 이제 글로벌 데이터베이스에서는 미국 동부(버지니아 북부) [us-east-1], 미국 동부(오하이오) [us-east-2], 미국 서부(캘리포니아 북부 지역) [us-west-1], 미국 서부(오레곤) [us-west-2], 유럽(아일랜드) [eu-west-1], 유럽(런던) [eu-west-2], 유럽(파리) [eu-west-3], 아시아 태평양(도쿄) [ap-northeast-1], 아시아 태평양(서울) [ap-northeast-2], 아시아 태평양(싱가포르) [ap-southeast-1], 아시아 태평양(시드니) [ap-southeast-2], 캐나다(중부) [ca-central-1], 유럽(프랑크푸르트) [eu-central-1], 아시아 태평양(뭄바이) [ap-south-1] AWS 리전에 배포된 데이터베이스 클러스터에 대해 보조 읽기 전용 복제본 리전을 추가할 수 있게 허용합니다.
- Amazon Aurora 기계 학습은 Aurora MySQL 데이터베이스 및 AWS 기계 학습(ML) 서비스 간의 가장 최적화된 통합입니다. Aurora 기계 학습에서는 개발자가 사용자 지정 통합을 구축하거나 별도의 도구를 배울 필요 없이 데이터베이스 개발을 위해 이미 사용 중인 익숙한 SQL 프로그래밍 언어를 사용해 ML 모델을 호출하여 다양한 ML 기반 예측을 자체 데이터베이스 애플리케이션에 추가할 수 있게 허용합니다. 자세한 내용은 [Amazon Aurora에서 기계 학습\(ML\) 기능 사용](#) 단원을 참조하십시오.
- 읽기 전용 복제본의 ANSI READ COMMITTED 격리 수준에 대한 지원을 추가하였습니다. 이 격리 수준을 통해 읽기 전용 복제본의 장기 실행 쿼리는 라이터 노드의 높은 쓰기 처리량에 영향을 미치지 않는 상태로 실행됩니다. 자세한 내용은 [Aurora MySQL 격리 수준](#)을 참조하십시오.

심각한 수정 사항:

- [CVE-2019-2922](#)
- [CVE-2019-2923](#)
- [CVE-2019-2924](#)
- [CVE-2019-2910](#)

우선 순위가 높은 수정 사항:

- 데이터베이스 가동 중지 시간이 늘어나는 원인이 되는 DDL 복구 관련 문제를 해결하였습니다. 다중 테이블 drop 문(예: DROP TABLE t1, t2, t3) 실행 이후 사용할 수 없게 되는 클러스터는 이 버전으로 업데이트해야 합니다.
- 데이터베이스 가동 중지 시간이 늘어나는 원인이 되는 DDL 복구 관련 문제를 해결하였습니다. INPLACE ALTER TABLE DDL 문 실행 이후 사용할 수 없게 되는 클러스터는 이 버전으로 업데이트해야 합니다.

일반적인 안정성 수정 사항:

- `information_schema.replica_host_status` 테이블에 일관성 없는 데이터를 생성한 문제를 해결하였습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- 버그 #26251621: 트리거 및 GCOL로 인한 잘못된 동작
- 버그 #22574695: ASSERTION '!TABLE || (!TABLE->READ_SET || BITMAP_IS_SET(TABLE->READ_SET, FIEL
- 버그 #25966845: 복제 키의 INSERT로 인해 교착 상태 발생
- 버그 #23070734: 동시 TRUNCATE 테이블로 인해 중단 발생
- 버그 #26191879: 외래 키 CASCADE에서 과도한 메모리 사용

- 버그 #20989615: INNODB AUTO_INCREMENT에서 동일한 값을 두 번 산출

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.07.0은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

현재 Aurora MySQL 2.07.0은 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-22(버전 2.06.0)

버전: 2.06.0

Aurora MySQL 2.06.0이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*입니다.

현재 지원되는 Aurora MySQL 릴리스에서 Aurora MySQL 2.06.0으로 스냅샷을 복원할 수 있습니다. 기존 Aurora MySQL 2.* 데이터베이스 클러스터를 Aurora MySQL 2.06.0으로 업그레이드 할 수 있는 옵션도 있습니다. 기존 Aurora MySQL 1.* 클러스터를 2.06.0으로 직접 업그레이드 할 수는 없습니다. 하지만 이 클러스터의 스냅샷을 Aurora MySQL 2.06.0으로 복원할 수 있습니다.

Aurora MySQL가 이전 버전인 클러스터를 생성하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1], 중동(바레인) [me-south-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

새로운 기능:

- Aurora MySQL 클러스터는 이제 db.r5.8xlarge, db.r5.16xlarge, db.r5.24xlarge 인스턴스 유형을 지원합니다. Aurora MySQL 클러스터의 인스턴스 유형에 대한 자세한 내용은 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 참조하십시오.
- 이제 해시 조인 기능은 정식 버전이며 Aurora 랩 모드 설정이 ON 상태일 필요가 없습니다. 이 기능은 동등 조인을 사용하여 많은 양의 데이터를 조인해야 하는 경우 이 기능으로 쿼리 성능을 향상시킬 수 있습니다. 이 기능 사용에 대한 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- 이제 핫 행 경합 기능은 정식 버전이며 Aurora 랩 모드 설정이 ON 상태일 필요가 없습니다. 이 기능은 동일한 페이지의 행에 대해 경합하는 트랜잭션이 많은 워크로드의 처리량을 크게 향상시킵니다.
- Aurora MySQL 2.06 이상에서는 백업에서 데이터를 복원하지 않고도 특정 시간으로 DB 클러스터 "되감기"하는 기능을 지원합니다. 역추적이라고 하는 이 기능을 사용하면 잘못된 테이블이나 잘못된 행을 삭제하는 등 사용자가 실수를 저지른 경우 빠르게 복구할 수 있습니다. 역추적은 대규모 데이터베이스라 할지라도 몇 초 내에 완료됩니다. [AWS 블로그](#)에서 개요를 읽고, 자세한 내용은 [Aurora DB 클러스터 역추적 \(p. 509\)](#) 단원을 참조하십시오.

심각한 수정 사항:

없음.

우선 순위가 높은 수정 사항:

CVE 수정 사항

- [CVE-2019-2805](#)
- [CVE-2019-2730](#)
- [CVE-2019-2739](#)
- [CVE-2019-2778](#)
- [CVE-2019-2758](#)
- [CVE-2018-3064](#)
- [CVE-2018-3058](#)
- [CVE-2018-2786](#)
- [CVE-2017-3653](#)
- [CVE-2017-3455](#)
- [CVE-2017-3465](#)
- [CVE-2017-3244](#)
- [CVE-2016-5612](#)

연결 처리

- DDL을 한 개 이상 실행하는 동안 클라이언트 연결 급증에 더 잘 대처하기 위해 데이터베이스 가용성을 개선하였습니다. 이러한 급증은 필요 시 추가 스레드를 일시적으로 생성하여 처리합니다. DDL 처리 중 연결 급증에 뒤이어 데이터베이스가 응답하지 않는 경우 업그레이드하는 것이 좋습니다.

엔진 다시 시작

- 엔진을 다시 시작하는 중에 사용 불가 상태가 지속되는 문제를 해결하였습니다. 이를 통해 버퍼풀 초기화 관련 문제가 해결됩니다. 이 문제는 드물게 발생하지만 지원되는 릴리스에 영향을 미칠 가능성이 있습니다.
- 대량의 쓰기 워크로드가 실행 중일 때 binlog 마스터로 구성된 데이터베이스가 다시 시작하는 원인이 되는 문제를 해결하였습니다.

일반적인 안정성 수정 사항:

- 캐시되지 않은 데이터에 액세스하는 쿼리가 평상시보다 더 느려질 수 있는 문제점을 개선하였습니다. 캐시되지 않은 데이터에 액세스하는 동안 알 수 없는 이유로 읽기 지연 시간이 늘어나는 고객은 이러한 문제를 겪을 때 업그레이드하는 것이 좋습니다.
- 데이터베이스 스냅샷에서 분할된 테이블을 복원하지 못하는 문제를 해결하였습니다. Aurora MySQL 1.* 데이터베이스의 스냅샷에서 복원된 데이터베이스 내 분할된 테이블에 액세스할 때 오류가 발생하는 문제를 겪는 고객은 이 버전을 사용하는 것이 좋습니다.
- DDL 쿼리가 라이터 DB 인스턴스에서 진행 중일 때 읽기 쿼리를 처리하는 스레드와 스키마 변경 사항을 적용하는 쿼리 사이의 잠금 경합을 해결하여 Aurora 복제본의 안정성을 높였습니다.
- DDL 작업을 통해 트리거되는 mysql.innodb_table_stats 테이블 업데이트와 관련된 안정성 문제를 해결하였습니다.
- Aurora 복제본의 임시 테이블에 대해 종첩된 쿼리가 실행될 때 ERROR 1836을 잘못 보고하는 문제를 해결하였습니다.

성능 개선 사항:

- 쿼리 캐시가 binlog 슬레이브에서 비활성화된 경우 캐시에 대한 불필요한 API 호출을 방지함으로써 binlog 복제의 성능을 높였습니다.

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.06.0은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

현재 Aurora MySQL 2.06.0은 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-11(버전 2.05.0)

버전: 2.05.0

Aurora MySQL 2.05.0이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*입니다.

현재 지원되는 Aurora MySQL 릴리스에서 Aurora MySQL 2.05.0으로 스냅샷을 복원할 수 있습니다. 기존 Aurora MySQL 2.* 데이터베이스 클러스터를 2.04.6까지 Aurora MySQL 2.05.0으로 업그레이드할 수 있는 옵션도 있습니다. 기존 Aurora MySQL 1.* 클러스터를 2.05.0으로 직접 업그레이드할 수는 없습니다. 하지만 이 클러스터의 스냅샷을 Aurora MySQL 2.05.0으로 복원할 수 있습니다.

Aurora MySQL가 이전 버전인 클러스터를 생성하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1], 유럽(스톡홀름) [eu-north-1], 중동(바레인) [me-south-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

심각한 수정 사항:

- CVE-2018-0734
- CVE-2019-2534
- CVE-2018-3155
- CVE-2018-2612
- CVE-2017-3599
- CVE-2018-3056

- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- sync_binlog 파라미터의 값이 1로 설정되지 않은 경우 마스터의 현재 binlog 파일에 있는 이벤트가 슬레이브에서 복제되지 않는 문제를 해결했습니다.

우선 순위가 높은 수정 사항:

- 데이터베이스 크기가 64TiB에 가까운 고객은 이 버전으로 업그레이드하여 Aurora 스토리지 한도에 가까운 볼륨에 영향을 미치는 안정성 버그로 인한 가동 중지를 방지할 것을 적극 권장합니다.
- 복제 지연 시간이 늘어나는 것을 방지하여 binlog 마스터에서 포그라운드 쿼리 성능을 향상하기 위해 `aurora_binlog_replication_max_yield_seconds` 파라미터의 기본값을 0으로 변경하였습니다.

MySQL 버그 수정 통합

- Bug#23054591: PURGE BINARY LOGS TO가 전체 binlog 파일을 읽고 있어 MySQL이 지연되고 있습니다.

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.05.0은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

현재 Aurora MySQL 2.05.0은 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-20(버전 2.04.8)

버전: 2.04.8

Aurora MySQL 2.04.8이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*입니다. 2.* Aurora MySQL 릴리스의 스냅샷을 Aurora MySQL 2.04.8로 복원할 수 있습니다. 기존 Aurora MySQL 2.* 데이터베이스 클러스터를 Aurora MySQL 2.04.8로 업그레이드할 수 있는 옵션도 있습니다.

Aurora MySQL가 이전 버전인 클러스터를 생성하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 아시아 태평양(홍콩) [ap-east-1], 중동(바레인) [me-south-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

새로운 기능:

- 읽기 전용 복제본 개선 사항:
 - 데이터를 효율적으로 전송함으로써 Aurora DB 클러스터 내 라이터 인스턴스에서 리더 인스턴스로 이동하는 네트워크 트래픽을 줄였습니다. 이 개선 사항은 복제본이 속도가 느려 다시 시작되는 것을 방지하는 데 도움이 되므로 기본적으로 활성화되어 있습니다. 이 기능의 파라미터는 `aurora_enable_repl_bin_log_filtering`입니다.
 - Aurora DB 클러스터 내 라이터 인스턴스에서 리더 인스턴스로 전송되는 네트워크 트래픽을 압축을 사용해 줄였습니다. 이 개선 사항은 8xlarge 및 16xlarge 인스턴스 클래스에 대해서만 기본적으로 활성화되어 있습니다. 그 이유는 이 인스턴스가 압축을 위한 추가 CPU 오버헤드를 허용할 수 있기 때문입니다. 이 기능의 파라미터는 `aurora_enable_replica_log_compression`입니다.

우선 순위가 높은 수정 사항:

- Aurora DB 클러스터 내에 리더 인스턴스가 있는 경우 대량 워크로드 처리 중 메모리 부족으로 인해 라이터가 다시 시작하는 것을 방지하는 Aurora 라이터 인스턴스의 메모리 관리를 개선하였습니다.
- 성능 스키마 객체에 동시에 액세스하는 동안 엔진이 재시작되는 스케줄러의 비결정적 조건이 수정되었습니다.

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.04.8은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.04.8은 현재, 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-14(버전 2.04.7)

버전: 2.04.7

Aurora MySQL 2.04.7이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*입니다.

현재 지원되는 Aurora MySQL 릴리스에서 Aurora MySQL 2.04.7로 스냅샷을 복원할 수 있습니다. 기존 Aurora MySQL 2.* 데이터베이스 클러스터를 Aurora MySQL 2.04.7로 업그레이드할 수 있는 옵션도 있습니다. 기존 Aurora MySQL 1.* 클러스터를 2.04.7로 직접 업그레이드할 수는 없습니다. 하지만 이 클러스터의 스냅샷을 Aurora MySQL 2.04.7로 복원할 수 있습니다.

Aurora MySQL가 이전 버전인 클러스터를 생성하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 아시아 태평양(홍콩) [ap-east-1], 중동(바레인) [me-south-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

우선 순위가 높은 수정 사항:

연결 처리

- DDL을 한 개 이상 실행하는 동안 클라이언트 연결 급증에 더 잘 대처하기 위해 데이터베이스 가용성을 개선하였습니다. 이러한 급증은 필요 시 추가 스레드를 일시적으로 생성하여 처리합니다. DDL 처리 중 연결 급증에 뛰어어 데이터베이스가 응답하지 않는 경우 업그레이드하는 것이 좋습니다.
- `Threads_running` 전역 상태 변수의 값이 잘못되는 문제를 해결했습니다.

엔진 다시 시작

- 엔진을 다시 시작하는 중에 사용 불가 상태가 지속되는 문제를 해결하였습니다. 이를 통해 버퍼 풀 초기화 관련 문제가 해결됩니다. 이 문제는 드물게 발생하지만 지원되는 릴리스에 영향을 미칠 가능성이 있습니다.

일반적인 안정성 수정 사항:

- 캐시되지 않은 데이터에 액세스하는 쿼리가 평상 시보다 더 느려질 수 있는 문제점을 개선하였습니다. 캐시되지 않은 데이터에 액세스하는 동안 알 수 없는 이유로 읽기 지연 시간이 늘어나는 고객은 이러한 문제를 겪을 때 업그레이드하는 것이 좋습니다.

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.04.7은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.04.7은 현재, 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-09-19(버전 2.04.6)

버전: 2.04.6

Aurora MySQL 2.04.6이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

기존 Aurora MySQL 2.* 데이터베이스 클러스터를 Aurora MySQL 2.04.6으로 업그레이드할 수 있는 옵션이 있습니다. Aurora MySQL 1.* 클러스터의 현재 위치 업그레이드는 허용되지 않습니다. 이러한 제한은 나중에 Aurora MySQL 2.* 버전에서 해제될 예정입니다. 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*의 스냅샷을 2.04.6으로 복원할 수 있습니다.

이전 버전의 Aurora MySQL을 사용하려면 AWS Management 콘솔, AWS CLI 또는 Amazon RDS API를 통해 엔진 버전을 지정하여 새로운 데이터베이스 클러스터를 생성하면 됩니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 유럽(런던) [eu-west-2], AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], and 아시아 태평양(홍콩) [ap-east-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- sync_binlog 파라미터의 값이 1로 설정되지 않은 경우 마스터의 현재 binlog 파일에 있는 이벤트가 슬레이브에서 복제되지 않는 문제를 해결했습니다.
- 복제 지연 시간이 늘어나는 것을 방지하여 binlog 마스터에서 포그라운드 쿼리 성능을 향상하기 위해 aurora_binlog_replication_max_yield_seconds 파라미터의 기본값을 0으로 변경하였습니다.

MySQL 버그 수정 통합

- Bug#23054591: PURGE BINARY LOGS TO가 전체 binlog 파일을 읽고 있어 MySQL이 지연되고 있습니다.

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.04.6은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

현재 Aurora MySQL 2.04.6은 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-07-08(버전 2.04.5)

버전: 2.04.5

Aurora MySQL 2.04.5가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

기존 Aurora MySQL 2.* 데이터베이스 클러스터를 Aurora MySQL 2.04.5로 업그레이드할 수 있는 옵션이 있습니다. Aurora MySQL 1.* 클러스터의 현재 위치 업그레이드는 허용되지 않습니다. 이러한 제한은 나중에 Aurora MySQL 2.* 버전에서 해제될 예정입니다. Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.* 및 2.04.*의 스냅샷을 Aurora MySQL 2.04.5로 복원할 수 있습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- 스토리지 볼륨을 늘리는 과정에서 데이터베이스가 다시 시작되는 원인이었던 경합 조건이 수정되었습니다.
- 볼륨을 여는 과정에서 데이터베이스가 다시 시작되는 원인이었던 내부 통신 장애가 수정되었습니다.
- 분할된 테이블에서 ALTER TABLE ALGORITHM=INPLACE에 대한 DDL 복구 지원 기능이 추가되었습니다.
- 데이터베이스가 다시 시작되는 원인이었던 ALTER TABLE ALGORITHM=COPY의 DDL 복구 문제가 수정되었습니다.
- 라이터에서 삭제 워크로드가 지나치게 높을 때 Aurora 복제본 안정성이 개선되었습니다.
- 전체 텍스트 검색 인덱스 동기화를 실행하는 스레드와 사전 캐시에서 전체 텍스트 검색 테이블을 제거하는 스레드 사이의 데드 래치로 인한 데이터베이스 재시작 문제가 수정되었습니다.
- DDL 복제 과정에서 Binlog 마스터에 대한 연결이 불안정할 때 Binlog 슬레이브에서 발생하는 안정성 문제가 수정되었습니다.
- 전체 텍스트 검색 코드에서 데이터베이스가 다시 시작되는 원인이었던 메모리 부족 문제가 수정되었습니다.
- 64TiB 볼륨을 모두 사용할 때 Aurora 라이터가 다시 시작되는 문제를 해결했습니다.
- 성능 스키마 기능에서 데이터베이스가 다시 시작되는 원인이었던 경합 조건이 수정되었습니다.
- 네트워크 프로토콜 관리의 오류를 처리할 때 중단된 연결로 인한 문제를 해결했습니다.

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.04.5는 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 7.7보다 20 배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.04.5는 현재, 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 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 확인 플러그인
- 쿼리 다시 쓰기 플러그인

- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-29(버전 2.04.4)

버전: 2.04.4

Aurora MySQL 2.04.4가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터를 생성할 때(스냅샷 복원 포함) MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. Aurora MySQL 1.* 클러스터를 인플레이스 업그레이드하거나 Amazon S3 백업에서 Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.04.4로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*의 스냅샷을 Aurora MySQL 2.04.4로 복원할 수 있습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1], 유럽(스톡홀름) [eu-north-1], 중국(ning)
샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- S3에서 Aurora로 데이터를 로드할 때 오류를 일으킬 수 있는 문제가 해결되었습니다.
- Aurora에서 S3로 데이터를 업로드할 때 오류를 일으킬 수 있는 문제가 해결되었습니다.
- 네트워크 프로토콜 관리의 오류를 처리할 때 중단된 연결로 인한 문제를 해결했습니다.
- 분할된 테이블을 처리할 때 충돌을 일으킬 수 있는 문제가 해결되었습니다.
- 일부 리전에서 성능 개선 도우미 기능을 사용할 수 없는 문제가 해결되었습니다.

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.04.4는 MySQL 4.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 7.7보다 20 배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.04.4는 현재, 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 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 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-09(버전 2.04.3)

버전: 2.04.3

Aurora MySQL 2.04.3이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터를 생성할 때(스냅샷 복원 포함) MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. Aurora MySQL 1.* 클러스터를 인플레이스 업그레이드하거나 Amazon S3 백업에서 Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.04.3으로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*의 스냅샷을 Aurora MySQL 2.04.3으로 복원할 수 있습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(نة) [cn-northwest-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- binlog 슬레이브로 구성된 Aurora 인스턴스에서 문제를 일으킬 수 있는 binlog 복제의 버그를 수정했습니다.
- 대규모 저장 루틴 처리 시 발생하는 메모리 부족 상태를 수정했습니다.
- 특정 유형의 ALTER TABLE 명령을 처리할 때 발생하는 오류를 수정했습니다.

- 네트워크 프로토콜 관리 시 발생한 오류로 인해 중단된 연결과 관련된 문제를 해결했습니다.

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.04.3은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등의 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20배 이상 향상된 쓰기 성능과 10배 이상 향상된 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.04.3은 현재 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-02(버전 2.04.2)

버전: 2.04.2

Aurora MySQL 2.04.2가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터를 생성할 때(스냅샷 복원 포함) MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. Aurora MySQL 1.* 클러스터를 인플레이스 업그레이드하거나 Amazon S3 백업에서 Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.04.2로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.0, 2.04.1의 스냅샷을 Aurora MySQL 2.04.2로 복원할 수 있습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(닝샤) [cn-northwest-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

Aurora MySQL 데이터베이스 클러스터를 업그레이드하는 방법에 대한 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- 사용자 지정 인증서를 사용하는 SSL binlog 복제에 대한 지원을 추가하였습니다. Aurora MySQL에서 SSL binlog 복제를 사용하는 방법에 관한 자세한 내용은 [mysql_rds_import_binlog_ssl_material](#) 단원을 참조하십시오.
- 전체 텍스트 검색 인덱스가 있는 테이블이 최적화되는 중에 Aurora 기본 인스턴스에서 발생하는 데드 래치를 수정했습니다.
- `SELECT(*)`를 사용하는 특정 쿼리의 성능이 보조 인덱스가 있는 테이블에서 영향을 받을 수 있는 Aurora 복제본의 문제를 수정했습니다.
- 오류 1032가 게시되게 한 조건을 수정했습니다.
- 여러 가지 데드 래치를 수정하여 Aurora 복제본의 안정성을 개선하였습니다.

MySQL 버그 수정 통합

- 버그 #24829050 - INDEX_MERGE_INTERSECTION 최적화로 인해 잘못된 쿼리 결과 산출

Aurora MySQL 버전 1과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 버전 1(MySQL 5.6과 호환됨)에서 지원되지만 현재 이 기능은 Aurora MySQL 버전 2(MySQL 5.7과 호환됨)에서는 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.04.2는 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등의 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20배 이상 향상된 쓰기 성능과 10배 이상 향상된 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.04.2는 현재 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 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 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문

Aurora MySQL 데이터베이스 엔진 업데이트 2019-03-25(버전 2.04.1)

버전: 2.04.1

Aurora MySQL 2.04.1이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터를 생성할 때(스냅샷 복원 포함) MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. Aurora MySQL 1.* 클러스터를 인플레이스 업그레이드하거나 Amazon S3 백업에서 Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.04.1로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 2.01.*, 2.02.*, 2.03.*, 2.04.0의 스냅샷을 Aurora MySQL 2.04.1로 복원할 수 있습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- 1.16보다 낮은 버전에 대한 Aurora MySQL 5.6 스냅샷을 최신 Aurora MySQL 5.7 클러스터로 복원할 수 없는 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-03-25(버전 2.04)

버전: 2.04

Aurora MySQL 2.04가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터를 생성할 때(스냅샷 복원 포함) MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. Aurora MySQL 1.* 클러스터를 인플레이스 업그레이드하거나 Amazon S3 백업에

서 Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.04.0으로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 1.19.*, 2.01.*, 2.02.*, 2.03.*의 스냅샷을 Aurora MySQL 2.04.0으로 복원할 수 있습니다.
Aurora MySQL 1.14.* 이하, 1.15.*, 1.16.*, 1.17.*, 1.18.*의 스냅샷을 Aurora MySQL 2.04.0으로 복원할 수 없
습니다. 이 제한 사항은 Aurora MySQL 2.04.1에서는 제거되었습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을
요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 리전에서 사용할 수 없습니다. 사용 가능
해지면 따로 발표할 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관
련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- GTID 기반 복제를 지원합니다. Aurora MySQL에서 GTID 기반 복제를 사용하는 방법에 대한 자세한 내용
은 [Aurora MySQL \(p. 580\)](#) 단원을 참조하십시오.
- 임시 테이블에 있는 행을 삭제 또는 업데이트하는 문에 InnoDB 하위 쿼리가 포함된 경우 Aurora 복제본에
서 `Running in read-only mode` 오류를 잘못 표시하는 문제를 수정했습니다.

MySQL 버그 수정 통합

- 버그 #26225783: 생성 테이블에서 MySQL 충돌(반복될 수 있음) -> INNODB: 세마포어 대기에 따른.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-02-07

버전: 2.03.4

Aurora MySQL 2.03.4가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora
MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터(스냅샷 복원 포함)를 생성할 때 MySQL 5.7 또는 MySQL 5.6과의 호환성을
선택할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.03.4로 인플레이스 업그레이드하거나 Amazon S3 백업에서
Aurora MySQL 2.03.4로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한
제한 사항을 제거할 계획입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을
요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서
사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관
련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- UTF8MB4 Unicode 9.0 액센트 구분 및 대소문자 구분 콜레이션 지원, `utf8mb4_0900_as_ci`.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-01-18

버전: 버전 2.03.3

Aurora MySQL 2.03.3이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터(스냅샷 복원 포함)를 생성할 때 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.03.3으로 인플레이스 업그레이드하거나 Amazon S3 백업에서 Aurora MySQL 2.03.3으로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- Aurora 복제본이 인덱스에서 역방향 스캔을 실행할 때 데드 래치 상태가 될 수 있는 문제를 해결했습니다.
- Aurora 기본 인스턴스가 분할된 테이블에서 현재 위치 DDL 작업을 실행할 때 Aurora 복제본이 다시 시작되는 문제를 해결했습니다.
- Aurora 기본 인스턴스에 대한 DDL 작업 후 쿼리 캐시 무효화 중에 Aurora 복제본이 다시 시작될 수 있는 문제를 해결했습니다.
- Aurora 기본 인스턴스가 테이블에서 잡림을 실행하는 동안 이 테이블에서 `SELECT` 쿼리 중에 Aurora 복제본이 다시 시작될 수 있는 문제를 해결했습니다.
- 인덱싱된 열만 액세스할 수 있는 MyISAM 임시 테이블에서 잘못된 결과가 나오는 문제를 해결했습니다.
- 약 40,000건의 쿼리를 수행한 후 주기적으로 `query_time` 및 `lock_time`에 대해 잘못된 대규모의 값을 생성하는 느린 로그의 문제를 해결했습니다.
- "tmp"라는 스키마로 인해 RDS MySQL에서 Aurora MySQL로의 마이그레이션이 정체될 수 있는 문제를 해결했습니다.
- 로그 순환 중에 감사 로그에 이벤트가 누락되는 문제를 해결했습니다.
- 랩 모드의 빠른 온라인 DDL 기능이 활성화될 때 Aurora 5.6 스냅샷에서 복원된 Aurora 기본 인스턴스가 다시 시작될 수 있는 문제를 해결했습니다.
- 사전 통계 스레드로 인해 CPU 사용량이 100%가 되는 문제를 해결했습니다.
- `CHECK TABLE` 문 실행 중에 Aurora 복제본이 다시 시작될 수 있는 문제를 해결했습니다.

MySQL 버그 수정 통합

- 버그 #25361251: SP에 있는 복제 키의 INSERT로 인한 오작동

- 버그 #26734162: BLOB INSERT + 복제 키 업데이트로 인한 오작동
- 버그 #27460607: INSERT SELECT의 원본 테이블이 비어 있을 때 IODKU 오작동 발생
- SELECT DISTINCT가 5.7에서 DISTINCT ROWS를 반환하지 않음(버그 #22343910)
- 오류 1093으로 인해 파생된 테이블 사용이 실패한 경우 조인된 테이블에서 삭제(버그 #23074801).
- GCOLS: 문자 집합 변경으로 인한 오작동(버그 #25287633).

Aurora MySQL 데이터베이스 엔진 업데이트 2019-01-09

버전: 2.03.2

Aurora MySQL 2.03.2가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터(스냅샷 복원 포함)를 생성할 때 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.03.2로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.03.2로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- Aurora 버전 선택기 – Aurora MySQL 2.03.2부터는 AWS Management 콘솔에서 MySQL 5.7 호환 Aurora 의 여러 버전 간에 선택할 수 있습니다. 자세한 내용은 [Aurora MySQL 엔진 버전 \(p. 686\)](#) 단원을 참조하십시오.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-24

버전: 2.03.1

Aurora MySQL 2.03.1이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

스냅샷에서 복원한 클러스터를 포함하여 새로운 Aurora MySQL DB 클러스터를 생성할 때는 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. MySQL 5.6 호환 스냅샷을 복원할 때, MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 2.01.*, 2.02.*, 2.03의 스냅샷을 Aurora MySQL 2.03.1로 복원할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.03.1로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.03.1로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

개선 사항

- 트랜잭션 교착(deadlock) 감지 실행 시 Aurora Writer가 다시 시작될 수 있는 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-11

버전: 2.03

Aurora MySQL 2.03이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

스냅샷에서 복원한 클러스터를 포함하여 새로운 Aurora MySQL DB 클러스터를 생성할 때는 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. MySQL 5.6 호환 스냅샷을 복원할 때, MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.03으로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.03으로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- 성능 스키마를 사용할 수 있습니다.
- 종료됨(killed) 상태의 좀비 세션이 CPU를 더 사용할 수 있는 문제를 수정했습니다.
- 읽기 전용 트랜잭션이 Aurora Writer의 레코드에 대한 잠금을 획득한 경우 발생하는 데드 래치 문제를 수정했습니다.
- 고객 워크로드가 없는 Aurora 복제본의 CPU 사용률이 높을 수 있는 문제를 수정했습니다.
- Aurora 복제본 또는 Aurora Writer를 다시 시작하게 만들 수 있는 여러 문제를 수정했습니다.
- 디스크 처리량 한도에 도달한 경우 진단 로깅을 건너뛰는 기능을 추가했습니다.
- Aurora Writer에서 binlog가 활성화된 경우 발생하는 메모리 누수 문제를 수정했습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- REVERSE SCAN ON A PARTITIONED TABLE DOES ICP - ORDER BY DESC(버그 #24929748).
- JSON_OBJECT CREATES INVALID JSON CODE(버그 #26867509).
- INSERTING LARGE JSON DATA TAKES AN INORDINATE AMOUNT OF TIME(버그 #22843444).

- PARTITIONED TABLES USE MORE MEMORY IN 5.7 THAN 5.6(버그 #25080442).

Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-08

버전: 2.02.5

Aurora MySQL 2.02.5가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

스냅샷에서 복원한 클러스터를 포함하여 새로운 Aurora MySQL DB 클러스터를 생성할 때는 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. MySQL 5.6 호환 스냅샷을 복원할 때, MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 2.01.*, 2.02.*의 스냅샷을 Aurora MySQL 2.02.5로 복원 할 수 있습니다. 또한 Aurora MySQL 2.01.* 또는 2.02.*에서 Aurora MySQL 2.02.5로 인플레이스 업그레이드를 수행할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.02.5로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.02.5로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 5.7의 이 릴리스에서는 성능 스키마가 비활성화됩니다. 성능 스키마를 지원하려면 Aurora 2.03으로 업그레이드하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- Aurora 복제본이 테이블에서 역방향 스캔을 수행할 때 다시 시작되는 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-09-21

버전: 2.02.4

Aurora MySQL 2.02.4가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

스냅샷에서 복원한 클러스터를 포함하여 새로운 Aurora MySQL DB 클러스터를 생성할 때는 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. MySQL 5.6 호환 스냅샷을 복원할 때, MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 2.01.*, 2.02.*의 스냅샷을 Aurora MySQL 2.02.4로 복원 할 수 있습니다. 또한 Aurora MySQL 2.01.* 또는 2.02.*에서 Aurora MySQL 2.02.4로 인플레이스 업그레이드를 수행할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.02.4로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.02.4로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 5.7의 이 릴리스에서는 성능 스키마가 비활성화됩니다. 성능 스키마를 지원하려면 Aurora 2.03으로 업그레이드하십시오.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- Aurora MySQL 5.6 스탠다드에서 복원된 테이블에서 전체 텍스트 검색 인덱스와 관련된 안정성 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-08-23

버전: 2.02.3

Aurora MySQL 2.02.3이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

스냅샷에서 복원한 클러스터를 포함하여 새로운 Aurora MySQL DB 클러스터를 생성할 때는 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. MySQL 5.6 호환 스냅샷을 복원할 때, MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 2.01.*, 2.02.*의 스냅샷을 Aurora MySQL 2.02.3으로 복원할 수 있습니다. 또한 Aurora MySQL 2.01.* 또는 2.02.*에서 Aurora MySQL 2.02.3으로 인플레이스 업그레이드를 수행할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.02.3으로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.02.3으로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 5.7의 이 릴리스에서는 성능 스키마가 비활성화됩니다. 성능 스키마를 지원하려면 Aurora 2.03으로 업그레이드하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Aurora MySQL 5.6과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 5.6에서 지원되지만, 이러한 기능은 Aurora MySQL 5.7에서 현재 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 이 함수는 Aurora MySQL 2.06 이상의 MySQL 5.7과 호환되는 클러스터에 대해 사용할 수 있습니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

현재, Aurora MySQL 2.01은 Aurora MySQL 버전 1.16 이상에서 추가된 기능을 지원하지 않습니다. Aurora MySQL 버전 1.16에 대한 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.01은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20 배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.01은 현재 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 전역 트랜잭션 식별자(GTID). Aurora MySQL은 버전 2.04 이상에서 GTID를 지원합니다.
- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문
- X 프로토콜

Aurora MySQL 2.x와 Aurora MySQL 1.x의 CLI 차이점

- Aurora MySQL 2.x의 엔진 이름은 `aurora-mysql`이고 Aurora MySQL 1.x의 엔진 이름은 계속해서 `aurora`입니다.
- Aurora MySQL 2.x의 엔진 버전은 5.7.12이고 Aurora MySQL 1.x의 엔진 버전은 계속해서 5.6.10a입니다.
- Aurora MySQL 2.x의 기본 파라미터 그룹은 `default.aurora-mysql5.7`이고 Aurora MySQL 1.x의 기본 파라미터 그룹은 계속해서 `default.aurora5.6`입니다.
- Aurora MySQL 2.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 `aurora-mysql5.7`이고 Aurora MySQL 1.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 계속해서 `aurora5.6`입니다.

전체 CLI 명령 세트 및 Aurora 2.x와 Aurora MySQL 1.x의 차이점은 Aurora MySQL 설명서를 참조하십시오.

개선 사항

- 레코드를 읽는 동안 낙관적 커서 사용이 복원될 때 Aurora 복제본이 다시 시작할 수 있는 문제를 수정했습니다.
- 인덱스 통계를 향상하기 위해 `innodb_stats_persistent_sample_pages` 파라미터 기본값을 128로 업데이트했습니다.
- Aurora 복제본이 Aurora 기본 인스턴스에서 동시 수정되는 작은 테이블에 액세스할 때 다시 시작될 수 있는 문제를 수정했습니다.
- 테이블 정의 캐시 비우기를 중지하기 위해 `ANALYZE TABLE`을 수정했습니다.
- 지역에 대한 포인트 쿼리를 검색 범위로 변환할 때 Aurora 기본 인스턴스 또는 Aurora 복제본이 다시 시작될 수 있는 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-06-04

버전: 2.02.2

Aurora MySQL 2.02.2이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

스냅샷에서 복원한 클러스터를 포함하여 새로운 Aurora MySQL DB 클러스터를 생성할 때는 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. MySQL 5.6 호환 스냅샷을 복원할 때, MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.14*, 1.15*, 1.16*, 1.17*, 2.01*, 2.02의 스냅샷을 Aurora MySQL 2.02.2로 복원할 수 있습니다. 또한 Aurora MySQL 2.01* 또는 2.02에서 Aurora MySQL 2.02.2로 인플레이스 업그레이드를 수행할 수 있습니다.

Aurora MySQL 1.* 클러스터를 Aurora MySQL 2.02.2로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.02.2로 복원하는 작업은 허용되지 않습니다. 나중에 Aurora MySQL 2.* 릴리스에서 이러한 제한 사항을 제거할 계획입니다.

Aurora MySQL 5.7의 이 릴리스에서는 성능 스키마가 비활성화됩니다. 성능 스키마를 지원하려면 Aurora 2.03으로 업그레이드하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Aurora MySQL 5.6과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 5.6에서 지원되지만, 이러한 기능은 Aurora MySQL 5.7에서 현재 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

현재, Aurora MySQL 2.01은 Aurora MySQL 버전 1.16 이상에서 추가된 기능을 지원하지 않습니다. Aurora MySQL 버전 1.16에 대한 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.01은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20 배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.01은 현재 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 전역 트랜잭션 식별자(GTID). Aurora MySQL은 버전 2.04 이상에서 GTID를 지원합니다.
- 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 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문
- X 프로토콜

Aurora MySQL 2.x와 Aurora MySQL 1.x의 CLI 차이점

- Aurora MySQL 2.x의 엔진 이름은 `aurora-mysql`이고 Aurora MySQL 1.x의 엔진 이름은 계속해서 `aurora`입니다.
- Aurora MySQL 2.x의 엔진 버전은 5.7.12이고 Aurora MySQL 1.x의 엔진 버전은 계속해서 5.6.10a입니다.
- Aurora MySQL 2.x의 기본 파라미터 그룹은 `default.aurora-mysql5.7`이고 Aurora MySQL 1.x의 기본 파라미터 그룹은 계속해서 `default.aurora5.6`입니다.
- Aurora MySQL 2.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 `aurora-mysql5.7`이고 Aurora MySQL 1.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 계속해서 `aurora5.6`입니다.

전체 CLI 명령 세트 및 Aurora 2.x와 Aurora MySQL 1.x의 차이점은 Aurora MySQL 설명서를 참조하십시오.

개선 사항

- Aurora 복제 진행 상황을 추적할 때 Aurora Writer가 가끔 다시 시작할 수 있는 문제를 수정했습니다.
- Aurora Writer의 테이블에서 인덱스 생성 또는 삭제 문을 실행한 후 파티셔닝된 테이블에 액세스할 때 Aurora 복제본이 다시 시작하거나 오류를 발생시키는 문제를 수정했습니다.
- Aurora Writer에서 ALTER 테이블 ADD/DROP 열 문을 실행하여 발생하는 변경 내용을 적용하는 동안 Aurora 복제본의 테이블에 액세스할 수 없는 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-05-03

버전: 2.02

Aurora MySQL 2.02가 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

스냅샷에서 복원한 클러스터를 포함하여 새로운 Aurora MySQL DB 클러스터를 생성할 때는 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. MySQL 5.6 호환 스냅샷을 복원할 때, MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.14*, 1.15*, 1.16*, 1.17*, 2.01*의 스냅샷을 Aurora MySQL 2.02로 복원할 수 있습니다. 또한 Aurora MySQL 2.01*에서 Aurora MySQL 2.02로 인플레이스 업그레이드를 수행할 수 있습니다.

Aurora MySQL 1.x 클러스터를 Aurora MySQL 2.02로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.02로 복원하는 작업은 허용되지 않습니다. 향후 Aurora MySQL 2.x 릴리스에서는 이러한 제한 사항을 없앨 계획입니다.

Aurora MySQL 5.7의 이 릴리스에서는 성능 스키마가 비활성화됩니다. 성능 스키마를 지원하려면 Aurora 2.03으로 업그레이드하십시오.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 AWS Premium Support 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Aurora MySQL 5.6과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 5.6에서 지원되지만, 이러한 기능은 Aurora MySQL 5.7에서 현재 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

현재, Aurora MySQL 2.01은 Aurora MySQL 버전 1.16 이상에서 추가된 기능을 지원하지 않습니다. Aurora MySQL 버전 1.16에 대한 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.01은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20 배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.01은 현재 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 전역 트랜잭션 식별자(GTID). Aurora MySQL은 버전 2.04 이상에서 GTID를 지원합니다.
- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문
- X 프로토콜

Aurora MySQL 2.x와 Aurora MySQL 1.x의 CLI 차이점

- Aurora MySQL 2.x의 엔진 이름은 `aurora-mysql`이고 Aurora MySQL 1.x의 엔진 이름은 계속해서 `aurora`입니다.
- Aurora MySQL 2.x의 엔진 버전은 5.7.12이고 Aurora MySQL 1.x의 엔진 버전은 계속해서 5.6.10aⁿⁿ입니다.
- Aurora MySQL 2.x의 기본 파라미터 그룹은 `default.aurora-mysql5.7`이고 Aurora MySQL 1.x의 기본 파라미터 그룹은 계속해서 `default.aurora5.6`입니다.
- Aurora MySQL 2.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 `aurora-mysql5.7`이고 Aurora MySQL 1.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 계속해서 `aurora5.6`입니다.

전체 CLI 명령 세트 및 Aurora 2.x와 Aurora MySQL 1.x의 차이점은 Aurora MySQL 설명서를 참조하십시오.

개선 사항

- INSERT 문을 실행시키고 Fast Insert 최적화를 이용할 때 Aurora Writer가 재시작되는 문제를 수정했습니다.
- Aurora 복제본에서 ALTER DATABASE 문을 실행시킬 때 Aurora 복제본이 재시작되는 문제를 수정했습니다.
- Aurora Writer에서 방금 삭제된 테이블에 쿼리를 실행할 때 Aurora 복제본이 재시작되는 문제를 수정했습니다.
- Aurora 복제본에서 innodb_adaptive_hash_index를 OFF로 설정할 때 Aurora 복제본이 재시작되는 문제를 수정했습니다.
- Aurora Writer에서 TRUNCATE TABLE 쿼리를 실행할 때 Aurora 복제본이 재시작되는 문제를 수정했습니다.
- INSERT 문을 실행할 때 Aurora Writer가 특정 상황에서 정지하는 문제를 수정했습니다. 여러 노드 클러스터에서는 장애 조치가 시작될 수 있습니다.
- 설정 세션 변수와 관련된 메모리 누수 문제를 수정했습니다.
- 생성된 열이 있는 테이블의 제거 실행 취소와 관련된 특정 상황에서 Aurora Writer가 정지하는 문제를 수정했습니다.
- 이진수 로깅이 활성화되어 있을 때 Aurora Writer가 때때로 재시작하는 문제를 수정했습니다.

MySQL 버그 수정 통합

- 좌측 조인이 외부 측에 잘못된 결과를 반환합니다(버그 #22833364).

Aurora MySQL 데이터베이스 엔진 업데이트 2018-03-13

버전: 2.01.1

Aurora MySQL 2.01.1이 정식 버전입니다. Aurora MySQL 2.x 버전은 MySQL 5.7과 호환 가능하고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

스냅샷에서 복원한 클러스터를 포함하여 새로운 Aurora MySQL DB 클러스터를 생성할 때는 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다. MySQL 5.6 호환 스냅샷을 복원할 때, MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.14*, 1.15*, 1.16*, 1.17*의 스냅샷을 Aurora MySQL 2.01.1로 복원할 수 있습니다.

Aurora MySQL 1.x 클러스터를 Aurora MySQL 2.01.1로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.01.1로 복원하는 작업은 허용되지 않습니다. 향후 Aurora MySQL 2.x 릴리스에서는 이러한 제한 사항을 없앨 계획입니다.

Aurora MySQL 5.7의 이 릴리스에서는 성능 스키마가 비활성화됩니다. 성능 스키마를 지원하려면 Aurora 2.03으로 업그레이드하십시오.

Aurora MySQL 5.6과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 5.6에서 지원되지만, 이러한 기능은 Aurora MySQL 5.7에서 현재 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.

- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

현재, Aurora MySQL 2.01.1은 Aurora MySQL 버전 1.16 이상에서 추가된 기능을 지원하지 않습니다. Aurora MySQL 버전 1.16에 대한 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.01.1은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20 배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.01.1은 현재, 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 전역 트랜잭션 식별자(GTID). Aurora MySQL은 버전 2.04 이상에서 GTID를 지원합니다.
- 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 확인 플러그인
- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문
- X 프로토콜

Aurora MySQL 2.x와 Aurora MySQL 1.x의 CLI 차이점

- Aurora MySQL 2.x의 엔진 이름은 `aurora-mysql`이고 Aurora MySQL 1.x의 엔진 이름은 계속해서 `aurora`입니다.
- Aurora MySQL 2.x의 엔진 버전은 5.7.12이고 Aurora MySQL 1.x의 엔진 버전은 계속해서 5.6.10nn입니다.
- Aurora MySQL 2.x의 기본 파라미터 그룹은 `default.aurora-mysql5.7`이고 Aurora MySQL 1.x의 기본 파라미터 그룹은 계속해서 `default.aurora5.6`입니다.
- Aurora MySQL 2.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 `aurora-mysql5.7`이고 Aurora MySQL 1.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 계속해서 `aurora5.6`입니다.

전체 CLI 명령 세트 및 Aurora 2.x와 Aurora MySQL 1.x의 차이점은 Aurora MySQL 설명서를 참조하십시오.

개선 사항

- MySQL 5.6 호환 스냅샷이 MySQL 5.7 호환성으로 복원되었을 때 Aurora 지정 데이터베이스 권한이 잘못 생성된 스냅샷 관련 문제를 해결했습니다.
- 1.17 스냅샷 복원에 대한 지원이 추가되었습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-02-06

버전: 2.01

Aurora MySQL 2.01이 정식 버전입니다. 앞으로 Aurora MySQL 2.x 버전은 MySQL 5.7과 호환되고, Aurora MySQL 1.x 버전은 MySQL 5.6과 호환됩니다.

새 Aurora MySQL DB 클러스터(스냅샷에서 복원된 클러스터 포함)를 생성할 때 MySQL 5.7 또는 MySQL 5.6과의 호환성을 선택할 수 있습니다.

Aurora MySQL 1.14*, 1.15*, 1.16*의 스냅샷을 Aurora MySQL 2.01로 복원할 수 있습니다.

Aurora MySQL 1.x 클러스터를 Aurora MySQL 2.01로 인플레이스 업그레이드하거나, Amazon S3 백업에서 Aurora MySQL 2.01로 복원하는 작업은 허용되지 않습니다. 향후 Aurora MySQL 2.x 릴리스에서는 이러한 제한 사항을 없앨 계획입니다.

Aurora MySQL 5.7의 이 릴리스에서는 성능 스키마가 비활성화됩니다. 성능 스키마를 지원하려면 Aurora 2.03으로 업그레이드하십시오.

Aurora MySQL 5.6과의 비교

다음 Amazon Aurora MySQL 기능은 Aurora MySQL 5.6에서 지원되지만, 이러한 기능은 Aurora MySQL 5.7에서 현재 지원되지 않습니다.

- 비동기식 키 미리 가져오기(AKP). 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 해시 조인. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- AWS Lambda 함수를 비동기식으로 호출하기 위한 네이티브 함수입니다. 자세한 내용은 [Aurora MySQL 네이티브 함수로 Lambda 함수 호출 \(p. 634\)](#) 단원을 참조하십시오.
- 배치화 스캔. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.
- Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션. 자세한 내용은 [Amazon S3 버킷을 사용하여 MySQL에서 데이터 마이그레이션 \(p. 475\)](#) 단원을 참조하십시오.

현재, Aurora MySQL 2.01은 Aurora MySQL 버전 1.16 이상에서 추가된 기능을 지원하지 않습니다. Aurora MySQL 버전 1.16에 대한 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 \(p. 742\)](#) 단원을 참조하십시오.

MySQL 5.7 호환성

Aurora MySQL 2.01은 MySQL 5.7과 유선 호환되며, JSON 지원, 공간 인덱스, 생성된 열 등과 같은 기능을 포함합니다. Aurora MySQL은 z축 곡선을 사용하는 공간 인덱싱의 기본 구현을 사용하여 MySQL 5.7보다 20 배 이상 높은 쓰기 성능과 10배 이상 높은 읽기 성능을 공간 데이터 세트에 제공합니다.

Aurora MySQL 2.01은 현재 다음과 같은 MySQL 5.7 기능을 지원하지 않습니다.

- 전역 트랜잭션 식별자(GTID). Aurora MySQL은 버전 2.04 이상에서 GTID를 지원합니다.
- 그룹 복제 플러그인
- 페이지 크기 증가
- 시작 시 InnoDB 버퍼풀 로딩
- InnoDB 풀 텍스트 구문 분석기 플러그인
- 멀티 소스 복제
- 온라인 버퍼풀 크기 조정
- 암호 확인 플러그인

- 쿼리 다시 쓰기 플러그인
- 복제 필터링
- CREATE TABLESPACE SQL 문
- X 프로토콜

Aurora MySQL 2.x와 Aurora MySQL 1.x의 CLI 차이점

- Aurora MySQL 2.x의 엔진 이름은 `aurora-mysql`이고 Aurora MySQL 1.x의 엔진 이름은 계속해서 `aurora`입니다.
- Aurora MySQL 2.x의 엔진 버전은 5.7.12이고 Aurora MySQL 1.x의 엔진 버전은 계속해서 5.6.10aⁿ입니다.
- Aurora MySQL 2.x의 기본 파라미터 그룹은 `default.aurora-mysql5.7`이고 Aurora MySQL 1.x의 기본 파라미터 그룹은 계속해서 `default.aurora5.6`입니다.
- Aurora MySQL 2.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 `aurora-mysql5.7`이고 Aurora MySQL 1.x의 DB 클러스터 파라미터 그룹 패밀리 이름은 계속해서 `aurora5.6`입니다.

전체 CLI 명령 세트 및 Aurora 2.x와 Aurora MySQL 1.x의 차이점은 Aurora MySQL 설명서를 참조하십시오.

Amazon Aurora MySQL 1.1에 대한 데이터베이스 엔진 업데이트

다음은 Amazon Aurora 1.1 데이터베이스 엔진 업데이트입니다.

- Aurora MySQL 데이터베이스 엔진 업데이트 2020-03-05 (p. 723)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-12-23(버전 1.22.1) (p. 724)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 1.22.0) (p. 724)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 1.21.0) (p. 727)
- Aurora MySQL 데이터베이스 엔진 업데이트 2020-03-05 (p. 729)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-11(버전 1.20.0) (p. 729)
- Aurora MySQL 데이터베이스 엔진 업데이트 2020-03-05 (p. 731)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-09-19(버전 1.19.5) (p. 731)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-06-05(버전 1.19.2) (p. 732)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-09(버전 1.19.1) (p. 733)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-02-07(버전 1.19.0) (p. 733)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-09-20 (p. 735)(버전 1.18.0)
- Aurora MySQL 데이터베이스 엔진 업데이트 2020-03-05 (p. 736)(버전 1.17.9)
- Aurora MySQL 데이터베이스 엔진 업데이트 2019-01-17 (p. 736)(버전 1.17.8)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-08 (p. 737)(버전 1.17.7)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-09-06 (p. 737)(버전 1.17.6)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-08-14 (p. 738)(버전 1.17.5)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-08-07 (p. 739)(버전 1.17.4)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-06-05 (p. 739)(버전 1.17.3)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-04-27 (p. 740)(버전 1.17.2)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-03-23 (p. 740)(버전 1.17.1)
- Aurora MySQL 데이터베이스 엔진 업데이트 2018-03-13 (p. 741)(버전 1.17)
- Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11 (p. 742)(버전 1.16)

- Aurora MySQL 데이터베이스 엔진 업데이트 2017-11-20 (p. 743)(버전 1.15.1)
- Aurora MySQL 데이터베이스 엔진 업데이트 2017-10-24 (p. 743)(버전 1.15)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2018-03-13 (p. 745)(버전 1.14.4)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2017-09-22 (p. 746)(버전 1.14.1)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2017-08-07 (p. 746)(버전 1.14)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2017-05-15 (p. 747)(버전 1.13)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2017-04-05 (p. 749)(버전 1.12)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2017-02-23 (p. 750)(버전 1.11)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2017-01-12 (p. 752)(버전 1.10.1)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-12-14 (p. 752)(버전 1.10)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-11-10 (p. 753)(버전 1.9.0, 1.9.1)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-10-26 (p. 754)(버전 1.8.1)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-10-18 (p. 754)(버전 1.8)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-09-20 (p. 755)(버전 1.7.1)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-08-30 (p. 756)(버전 1.7)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-06-01 (p. 757)(버전 1.6.5)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-04-06 (p. 757)(버전 1.6)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2016-01-11 (p. 759)(버전 1.5)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2015-12-03 (p. 759)(버전 1.4)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2015-10-16 (p. 760)(버전 1.2, 1.3)
- Aurora MySQL 데이터베이스 엔진 업데이트: 2015-08-24 (p. 762)(버전 1.1)

Aurora MySQL 데이터베이스 엔진 업데이트 2020-03-05

버전: 1.22.2

Aurora MySQL 1.22.2가 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*,
2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*입니다. Aurora MySQL 1.* 데이터베이스의 스냅샷을
Aurora MySQL 1.22.2로 복원할 수 있습니다.

이전 버전의 Aurora MySQL로 클러스터를 생성하려면 RDS 콘솔, AWS CLI 또는 Amazon RDS API를 통해
엔진 버전을 지정하십시오.

Note

이 버전은 현재 다음 리전에서 사용할 수 없습니다. AWS GovCloud(US-East) [us-gov-east-1], AWS
GovCloud (US-West) [us-gov-west-1]. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을
요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

높은 우선 순위:

- 인증서 교체 후 간헐적인 연결 실패 문제를 해결했습니다.
- 쓰기 로드가 많은 일부 데이터베이스 클러스터에서 복제가 더 오래 걸리는 문제를 해결했습니다.

- binlog_checksum 파라미터가 마스터 및 복제본에서 서로 다른 값으로 설정된 경우 논리적 복제가 끊어지는 문제를 해결했습니다.
- 읽기 전용 복제본에서 느린 로그 및 일반 로그가 제대로 회전하지 않을 수 있는 문제를 해결했습니다.
- ANSI 읽기 커밋된 격리 수준 동작의 문제를 해결했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-12-23(버전 1.22.1)

버전: 1.22.1

Aurora MySQL 1.22.1이 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*입니다. Aurora MySQL가 이전 버전인 클러스터를 생성 하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오. 기존 Aurora MySQL 1.* 데이터베이스 클러스터를 Aurora MySQL 1.22.1로 업그레이드할 수 있는 옵션이 있습니다.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1], 중동(바레인) [me-south-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

심각한 수정 사항:

- 테이블 잠금 및 임시 테이블과 관련된 엔진 복구를 방해했던 문제가 해결되었습니다.
- 임시 테이블을 사용할 때 이전 로그의 안정성이 향상되었습니다.

높은 우선 순위:

- Aurora 특정 데이터베이스 추적 및 로깅 하위 시스템에서 사용 가능한 메모리를 낮추는 느린 메모리 누수 문제가 수정되었습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 1.22.0)

버전: 1.22.0

Aurora MySQL 1.22.0이 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*입니다. Aurora MySQL가 이전 버전인 클러스터를 생성

하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오. 기존 Aurora MySQL 1.* 데이터베이스 클러스터를 Aurora MySQL 1.22.0으로 업그레이드할 수 있는 옵션이 있습니다.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1], 종동(바레인) [me-south-1], 남아메리카(상파울루) [sa-east-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표 할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

새로운 기능:

- Aurora MySQL 클러스터는 이제 r5.8xlarge, r5.16xlarge 및 r5.24xlarge 인스턴스 유형을 지원합니다.
- Binlog는 매우 큰 트랜잭션이 수반될 때 커밋 시간 지연을 줄이기 위한 새로운 개선 사항입니다.
- 현재 Aurora MySQL은 커밋 시 대규모 트랜잭션의 이벤트가 binlog에 쓰이는 기간을 최소화하는 메커니즘을 갖추고 있습니다. 이 메커니즘을 통해 이 기간 중 데이터베이스 총돌이 발생할 때 초래되는 오프라인 복구 지연을 효과적으로 방지할 수 있습니다. 또한 이 기능은 binlog 커밋 시 대규모 트랜잭션이 소규모 트랜잭션을 차단하는 문제도 해결합니다. 이 기능은 기본적으로 꺼져 있으며 워크로드에 필요한 경우 서비스 팀이 활성화할 수 있습니다. 활성화된 후에는 트랜잭션 크기가 500MB를 초과하면 트리거됩니다.
- 읽기 전용 복제본의 ANSI READ COMMITTED 격리 수준에 대한 지원을 추가하였습니다. 이 격리 수준을 통해 읽기 전용 복제본의 장기 실행 쿼리는 라이터 노드의 높은 쓰기 처리량에 영향을 미치지 않는 상태로 실행됩니다. 자세한 내용은 [Aurora MySQL 격리 수준](#)을 참조하십시오.
- 이제 글로벌 데이터베이스에서는 미국 동부(버지니아 북부) [us-east-1], 미국 동부(오하이오) [us-east-2], 미국 서부(캘리포니아 북부 지역) [us-west-1], 미국 서부(오레곤) [us-west-2], 유럽(아일랜드) [eu-west-1], 유럽(런던) [eu-west-2], 유럽(파리) [eu-west-3], 아시아 태평양(도쿄) [ap-northeast-1], 아시아 태평양(서울) [ap-northeast-2], 아시아 태평양(싱가포르) [ap-southeast-1], 아시아 태평양(시드니) [ap-southeast-2], 캐나다(중부) [ca-central-1], 유럽(프랑크푸르트) [eu-central-1], 아시아 태평양(뭄바이) [ap-south-1] AWS 리전에 배포된 데이터베이스 클러스터에 대해 보조 읽기 전용 복제본 리전을 추가할 수 있게 허용합니다.
- 이제 핫 행 경합 기능은 정식 버전이며 Aurora 랩 모드 설정이 ON 상태일 필요가 없습니다. 이 기능은 동일 한 페이지의 행에 대해 경합하는 트랜잭션이 많은 워크로드의 처리량을 크게 향상 시킵니다.
- 이 버전에서는 새 클러스터에 대해 최신 브라질 시간대 업데이트를 지원하도록 시간대 파일을 업데이트하였습니다.

심각한 수정 사항:

- [CVE-2019-2922](#)
- [CVE-2019-2923](#)
- [CVE-2019-2924](#)
- [CVE-2019-2910](#)

높은 우선 순위:

- [CVE-2019-2805](#)
- [CVE-2019-2730](#)

- CVE-2019-2740
- CVE-2018-3064
- CVE-2018-3058
- CVE-2017-3653
- CVE-2017-3464
- CVE-2017-3244
- CVE-2016-5612
- CVE-2016-5439
- CVE-2016-0606
- CVE-2015-4904
- CVE-2015-4879
- CVE-2015-4864
- CVE-2015-4830
- CVE-2015-4826
- CVE-2015-2620
- CVE-2015-0382
- CVE-2015-0381
- CVE-2014-6555
- CVE-2014-4258
- CVE-2014-4260
- CVE-2014-2444
- CVE-2014-2436
- CVE-2013-5881
- CVE-2014-0393
- CVE-2013-5908
- CVE-2013-5807
- CVE-2013-3806
- CVE-2013-3811
- CVE-2013-3804
- CVE-2013-3807
- CVE-2013-2378
- CVE-2013-2375
- CVE-2013-1523
- CVE-2013-2381
- CVE-2012-5615
- CVE-2014-6489
- 데이터베이스 가동 중지 시간이 늘어나는 원인이 되는 DDL 복구 구성 요소 관련 문제를 해결하였습니다.
AUTO_INCREMENT 열이 있는 테이블에서 TRUNCATE TABLE 쿼리를 실행한 이후 사용할 수 없게 되는 클러스터는 업데이트해야 합니다.
- 데이터베이스 가동 중지 시간이 늘어나는 원인이 되는 DDL 복구 구성 요소 관련 문제를 해결하였습니다.
다중 테이블에서 DROP TABLE 쿼리를 병렬로 실행한 이후 사용할 수 없게 되는 클러스터는 업데이트해야 합니다.

일반적인 안정성 수정 사항:

- 장기 실행 트랜잭션 중에 읽기 복제본이 다시 시작하는 원인이 되는 문제를 해결하였습니다. 여유 메모리 하락 가속화와 동시에 복제본이 다시 시작하는 문제를 겪는 고객은 이 버전으로 업그레이드하는 것을 고려해야 합니다.
- 읽기 전용 복제본의 임시 테이블에 대해 종첩된 쿼리가 실행될 때 `ERROR 1836`을 잘못 보고하는 문제를 해결하였습니다.
- Aurora 라이터 인스턴스에서 많은 양의 쓰기 워크로드가 실행 중일 때 Aurora 리더 인스턴스의 병렬 쿼리 가 중단되는 오류를 해결하였습니다.
- 대량의 쓰기 워크로드가 실행 중일 때 Binlog 마스터로 구성된 데이터베이스가 다시 시작하는 원인이 되는 문제를 해결하였습니다.
- 엔진을 다시 시작하는 중에 사용 불가 상태가 지속되는 문제를 해결하였습니다. 이를 통해 버퍼풀 초기화 관련 문제가 해결됩니다. 이 문제는 드물게 발생하지만 지원되는 릴리스에 영향을 미칠 가능성이 있습니다.
- `information_schema.replica_host_status` 테이블에 일관성 없는 데이터를 생성한 문제를 해결하였습니다.
- 병렬 쿼리와 표준 실행 경로 간의 교착 상태로 인해 리더 노드가 간헐적으로 다시 시작하는 문제를 해결하였습니다.
- 클라이언트 연결의 수가 `max_connections` 파라미터 값을 초과할 때의 데이터베이스 안정성을 개선하였습니다.
- 지원되지 않는 DDL 및 `LOAD FROM S3` 쿼리를 차단하여 리더 인스턴스의 안정성을 개선하였습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- 버그 #16346241 - ITEM_PARAM::QUERY_VAL_STR의 서버 충돌
- 버그 #17733850 - ITEM_NAME_CONST::ITEM_NAME_CONST()의 NAME_CONST() 충돌
- 버그 #20989615: INNODB AUTO_INCREMENT에서 동일한 값을 두 번 산출
- 버그 #20181776 - 액세스 제어에 와일드카드가 포함된 경우 가장 제한적인 호스트와 일치하지 않음
- Bug #27326796 - MYSQL이 PARSOPARS.CC 파일의 INNODB ASSERTION 실패와 충돌
- Bug #20590013 - FULLTEXT 인덱스가 있는데 이를 삭제하면 더 이상 온라인 DDL을 수행할 수 없음

Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 1.21.0)

버전: 1.21.0

Aurora MySQL 1.21.0이 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*입니다. Aurora MySQL가 이전 버전인 클러스터를 생성하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오. 기존 Aurora MySQL 1.* 데이터베이스 클러스터를 Aurora MySQL 1.21.0으로 업그레이드할 수 있는 옵션이 있습니다.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1], 유럽(스톡홀름) [eu-north-1], 중동(바레인) [me-south-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

심각한 수정 사항:

- [CVE-2018-0734](#)
- [CVE-2019-2534](#)
- [CVE-2018-2612](#)
- [CVE-2017-3599](#)
- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- [CVE-2015-4737](#)

높은 우선 순위:

- 데이터베이스 크기가 64TiB에 가까운 고객은 이 버전으로 업그레이드하여 Aurora 스토리지 한도에 가까운 볼륨에 영향을 미치는 안정성 버그로 인한 가동 중지를 방지할 것을 적극 권장합니다.

일반적인 안정성 수정 사항:

- Aurora 라이터 인스턴스에서 많은 양의 쓰기 워크로드가 실행 중일 때 Aurora 리더 인스턴스의 병렬 쿼리가 중단되는 오류를 해결했습니다.
- 라이터 인스턴스에 트랜잭션 커밋 트래픽이 가중되는 상황에서 트랜잭션이 장기간 실행 중이면 Aurora 리더 인스턴스의 여유 메모리가 줄어드는 문제를 해결했습니다.
- 이제 `aurora_disable_hash_join` 파라미터의 값은 데이터베이스 재시작 또는 호스트 대체 후에도 지속됩니다.
- Aurora 인스턴스의 메모리 부족 현상의 원인이 되는 전체 텍스트 검색 캐시 관련 문제를 해결했습니다. 전체 텍스트 검색을 사용하는 고객은 업그레이드해야 합니다.
- 해시 조인 기능이 활성화된 상태에서 인스턴스의 메모리가 얼마 남지 않았을 때를 대비해 데이터베이스의 안정성을 높였습니다. 해시 조인을 사용하는 고객은 업그레이드해야 합니다.
- “너무 많은 연결” 오류가 재부팅의 원인이 될 수 있는 쿼리 캐시 문제를 해결했습니다.
- 스왑 메모리 공간을 포함하여 불필요한 재부팅을 방지하도록 T2 인스턴스의 여유 메모리 계산 방식을 수정하였습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY
- Bug #17059925: UNION 문에서 행 검토 값이 잘못 계산되었습니다. 이러한 오류는 성능 스키마 문 테이블 (예: `events_statements_current`)의 `ROWS_EXAMINED` 열 값이 너무 커지는 것으로 나타났습니다.
- Bug #11827369: SELECT ... FROM DUAL 종합 하위 쿼리가 포함된 일부 쿼리로 인해 어설션이 발생하였습니다.
- 버그 #16311231: WHERE 절에 XOR 작업이 포함된 IN 절에 하위 쿼리가 들어 있는 쿼리의 경우 잘못된 결과가 반환되었습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2020-03-05

버전: 1.20.1

Aurora MySQL 1.20.1이 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*입니다. Aurora MySQL 1.* 데이터베이스의 스냅샷을 Aurora MySQL 1.20.1로 복원할 수 있습니다.

이전 버전의 Aurora MySQL로 클러스터를 생성하려면 RDS 콘솔, AWS CLI 또는 Amazon RDS API를 통해 엔진 버전을 지정하십시오.

Note

이 버전은 현재 다음 리전에서 사용할 수 없습니다. AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

높은 우선 순위:

- 인증서 교체 후 간헐적인 연결 실패 문제를 해결했습니다.
- 과중한 워크로드에서 장애 조치가 발생할 수 있는 연결 닫기 동시성과 관련된 문제를 해결했습니다.

일반적인 안정성 수정 사항:

- 중간 테이블을 내부적으로 사용하는 다중 테이블 조인 및 집계가 포함된 복잡한 쿼리를 실행하는 동안 발생하는 충돌 문제를 해결했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-11(버전 1.20.0)

버전: 1.20.0

Aurora MySQL 1.20.0이 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*입니다. Aurora MySQL가 이전 버전인 클러스터를 생성하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하십시오. 기존 Aurora MySQL 1.* 데이터베이스 클러스터를 1.19.5까지 Aurora MySQL 1.20.0으로 업그레이드할 수 있는 옵션이 있습니다.

Note

현재 이 버전은 AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1], 유럽(스톡홀름) [eu-north-1], 중동(바레인) [me-south-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

심각한 수정 사항:

- [CVE-2018-0734](#)
- [CVE-2019-2534](#)
- [CVE-2018-2612](#)
- [CVE-2017-3599](#)
- [CVE-2018-2562](#)
- [CVE-2017-3329](#)
- [CVE-2018-2696](#)
- [CVE-2015-4737](#)

높은 우선 순위:

- 데이터베이스 크기가 64TiB에 가까운 고객은 이 버전으로 업그레이드하여 Aurora 스토리지 한도에 가까운 볼륨에 영향을 미치는 안정성 버그로 인한 가동 중지를 방지할 것을 적극 권장합니다.

일반적인 안정성 수정 사항:

- Aurora 라이터 인스턴스에서 많은 양의 쓰기 워크로드가 실행 중일 때 Aurora 리더 인스턴스의 병렬 쿼리 가중단되는 오류를 해결했습니다.
- 라이터 인스턴스에 트랜잭션 커밋 트래픽이 가중되는 상황에서 트랜잭션이 장기간 실행 중이면 Aurora 리더 인스턴스의 여유 메모리가 줄어드는 문제를 해결했습니다.
- 이제 `aurora_disable_hash_join` 파라미터의 값은 데이터베이스 재시작 또는 호스트 대체 후에도 지속됩니다.
- Aurora 인스턴스의 메모리 부족 현상의 원인이 되는 전체 텍스트 검색 캐시 관련 문제를 해결했습니다. 전체 텍스트 검색을 사용하는 고객은 업그레이드해야 합니다.
- 해시 조인 기능이 활성화된 상태에서 인스턴스의 메모리가 얼마 남지 않았을 때를 대비해 데이터베이스의 안정성을 높였습니다. 해시 조인을 사용하는 고객은 업그레이드해야 합니다.
- “너무 많은 연결” 오류가 재부팅의 원인이 될 수 있는 쿼리 캐시 문제를 해결했습니다.
- 스왑 메모리 공간을 포함하여 불필요한 재부팅을 방지하도록 T2 인스턴스의 여유 메모리 계산 방식을 수정하였습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY
- Bug #17059925: UNION 문에서 행 검토 값이 잘못 계산되었습니다. 이러한 오류는 성능 스키마 문 테이블 (예: `events_statements_current`)의 `ROWS_EXAMINED` 열 값이 너무 커지는 것으로 나타났습니다.
- Bug #11827369: SELECT ... FROM DUAL 종합 하위 쿼리가 포함된 일부 쿼리로 인해 어설션이 발생하였습니다.
- 버그 #16311231: WHERE 절에 XOR 작업이 포함된 IN 절에 하위 쿼리가 들어 있는 쿼리의 경우 잘못된 결과가 반환되었습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2020-03-05

버전: 1.19.6

Aurora MySQL 1.19.6이 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*입니다. Aurora MySQL 1.* 데이터베이스의 스냅샷을 Aurora MySQL 1.19.6으로 복원할 수 있습니다.

이전 버전의 Aurora MySQL로 클러스터를 생성하려면 RDS 콘솔, AWS CLI 또는 Amazon RDS API를 통해 엔진 버전을 지정하십시오.

Note

이 버전은 현재 다음 리전에서 사용할 수 없습니다. AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

높은 우선 순위:

- 인증서 교체 후 간헐적인 연결 실패 문제를 해결했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-09-19(버전 1.19.5)

버전: 1.19.5

Aurora MySQL 1.19.5가 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

기존 데이터베이스 클러스터를 Aurora MySQL 1.19.5로 업그레이드 할 수 있는 옵션이 있습니다. Aurora MySQL 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.1, 1.19.2의 스냅샷을 Aurora MySQL 1.19.5로 복원할 수 있습니다.

이전 버전의 Aurora MySQL을 사용하려면 AWS Management 콘솔, AWS CLI 또는 RDS API를 통해 엔진 버전을 지정하여 새로운 데이터베이스 클러스터를 생성하면 됩니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 유럽(런던) [eu-west-2], AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1], 중국(닝샤) [cn-northwest-1], and 아시아 태평양(홍콩) [ap-east-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- 라이터 인스턴스에 트랜잭션 커밋 트래픽이 가중되는 상황에서 트랜잭션이 장기간 실행 중이면 Aurora 리더 인스턴스의 여유 메모리가 줄어드는 문제를 해결했습니다.
- Aurora 라이터 인스턴스에서 많은 양의 쓰기 워크로드가 실행 중일 때 Aurora 리더 인스턴스의 병렬 쿼리가 중단되는 오류를 해결했습니다.
- 이제 `aurora_disable_hash_join` 파라미터의 값은 데이터베이스 재시작 또는 호스트 대체 후에도 지속됩니다.
- Aurora 인스턴스의 메모리 부족 현상의 원인이 되는 전체 텍스트 검색 캐시 관련 문제를 해결했습니다.
- 볼륨 크기가 볼륨 한도인 64TiB에 근접할 때를 대비해 복구 워크플로우가 장애 조치 없이 완료될 수 있도록 160GB의 공간을 예약하여 데이터베이스의 안정성을 높였습니다.
- 해시 조인 기능이 활성화된 상태에서 인스턴스의 메모리가 얼마 남지 않았을 때를 대비해 데이터베이스의 안정성을 높였습니다.
- 초기에 재부팅되는 현상을 유발하는 여유 메모리 계산 방식을 수정하여 T2 인스턴스의 스왑 메모리 공간을 포함하도록 하였습니다.
- “너무 많은 연결” 오류가 재부팅의 원인이 될 수 있는 쿼리 캐시 문제를 해결했습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- CVE-2018-2696
- CVE-2015-4737
- Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY
- Bug #17059925: UNION 문에서 행 검토 값이 잘못 계산되었습니다. 이러한 오류는 성능 스키마 문 테이블 (예: `events_statements_current`)의 `ROWS_EXAMINED` 열 값이 너무 커지는 것으로 나타났습니다.
- Bug #11827369: SELECT ... FROM DUAL 종점 하위 쿼리가 포함된 일부 쿼리로 인해 어설션이 발생하였습니다.
- 버그 #16311231: WHERE 절에 XOR 작업이 포함된 IN 절에 하위 쿼리가 들어 있는 쿼리의 경우 잘못된 결과가 반환되었습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-06-05(버전 1.19.2)

버전: 1.19.2

Aurora MySQL 1.19.2가 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 1.17.8, 1.19.0, 1.19.1 또는 1.19.2에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.19.2로 업그레이드할 수 있지만 필수는 아닙니다. 이전 버전을 사용하려면 Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8 또는 Aurora MySQL 1.18에서 데이터베이스 클러스터를 새로 만드십시오. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1], 유럽(스톡홀름) [eu-north-1], 중국(ningxia) [cn-northwest-1], 아시아 태평양(홍콩) [ap-east-1] AWS 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- Amazon S3에서 Aurora로 데이터를 로드할 때 오류를 일으킬 수 있는 문제가 해결되었습니다.
- Aurora에서 Amazon S3로 데이터를 업로드할 때 오류를 일으킬 수 있는 문제가 해결되었습니다.
- 중단된 상태로 남아있는 좀비 세션을 생성한 문제가 해결되었습니다.
- 네트워크 프로토콜 관리의 오류를 처리할 때 중단된 연결로 인한 문제를 해결했습니다.
- 분할된 테이블을 처리할 때 충돌을 일으킬 수 있는 문제가 해결되었습니다.
- 트리거 생성의 binlog 복제와 관련된 문제가 수정되었습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-09(버전 1.19.1)

버전: 1.19.1

Aurora MySQL 1.19.1이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 1.17.8, 1.19.0 또는 1.19.1에서 생성할 수 있습니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.19.1로 업그레이드할 수 있지만 필수는 아닙니다. 이전 버전을 사용하려면 Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8 또는 Aurora MySQL 1.18에서 데이터베이스 클러스터를 새로 만드십시오. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표될 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

개선 사항

- binlog 슬레이브로 구성된 Aurora 인스턴스에서 문제를 일으킬 수 있는 binlog 복제의 버그를 수정했습니다.
- 특정 유형의 ALTER TABLE 명령을 처리할 때 발생하는 오류를 수정했습니다.
- 네트워크 프로토콜 관리 시 발생한 오류로 인해 중단된 연결과 관련된 문제를 해결했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-02-07(버전 1.19.0)

버전: 1.19.0

Aurora MySQL 1.19.0이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 1.17.8 또는 1.19.0에서 생성됩니다. 기존 데이터베이스

이스 클러스터를 Aurora MySQL 1.19.0으로 업그레이드할 수 있지만 필수는 아닙니다. 이전 버전을 사용하려면 Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16, Aurora MySQL 1.17.8 또는 Aurora MySQL 1.18.0에서 데이터베이스 클러스터를 새로 만드십시오. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

Note

DB 클러스터를 업그레이드하는 절차가 변경되었습니다. 자세한 내용은 [Amazon Aurora MySQL 관련 데이터베이스 업그레이드 및 패치 \(p. 687\)](#) 단원을 참조하십시오.

기능

- Aurora 버전 선택기 - Aurora MySQL 1.19.0부터는 Amazon RDS 콘솔에서 MySQL 5.6과 호환되는 Aurora의 여러 버전 중에서 선택할 수 있습니다. 자세한 내용은 [Aurora MySQL 엔진 버전 \(p. 686\)](#) 단원을 참조하십시오.

개선 사항

- Aurora 복제본의 CHECK TABLE 쿼리와 관련된 안정성 문제를 해결했습니다.
- 새로운 전역 사용자 변수인 `aurora_disable_hash_join`을 도입하여 해시 조인을 비활성화하였습니다.
- 다중 테이블 해시 조인 중에 출력을 생성할 때 발생하는 안정성 문제를 해결했습니다.
- 해시 조인 적용 가능성 점검 중 계획 변경으로 인해 잘못된 결과를 반환하는 문제를 해결했습니다.
- 가동 중단 제로화 패치 적용은 장기적인 실행 트랜잭션에서 지원됩니다. 이 개선 사항은 버전 1.19를 더 높은 버전으로 업그레이드할 때 효력이 발생합니다.
- 가동 중단 제로화 패치 적용은 현재 binlog가 활성화되어 있을 때 지원됩니다. 이 개선 사항은 버전 1.19를 더 높은 버전으로 업그레이드할 때 효력이 발생합니다.
- 워크로드와 관련이 없는 Aurora 복제본에서 CPU 사용률이 급증하는 원인이 되는 문제를 해결했습니다.
- 데이터베이스 재시작으로 이어지는 잠금 관리자의 교착 상태를 해결했습니다.
- Aurora 인스턴스의 안정성을 높이기 위해 잠금 관리자 구성 요소의 교착 상태를 해결했습니다.
- 잠금 관리자 구성 요소 내부에 있는 교착 상태 감지기의 안정성을 개선하였습니다.
- InnoDB에서 인덱스가 손상된 것을 감지한 경우 테이블에서 INSERT 작업은 허용되지 않습니다.
- Fast DDL에서 안정성 문제를 해결했습니다.
- 단일 행 하위 쿼리에 대한 배치화 스캔 중 메모리 소비를 줄임으로써 Aurora 안정성을 개선하였습니다.
- 시스템 변수 `foreign_key_checks`가 0으로 설정된 상태에서 외래 키가 드롭된 후 발생한 안정성 문제를 해결하였습니다.
- `table_definition_cache` 값에 대한 사용자의 변경 사항을 잘못 재정의하는 메모리 부족 방지 기능의 문제를 해결했습니다.
- 메모리 부족 방지 기능의 안정성 문제를 해결했습니다.
- `slow_query_log`의 `query_time` 및 `lock_time`을 가비지 값으로 설정하는 문제를 해결했습니다.
- 내부적으로 문자열 클레이션을 부적절하게 처리함으로 인해 트리거되는 병렬 쿼리 안정성 문제를 해결했습니다.
- 보조 인덱스 검색으로 인해 트리거되는 병렬 쿼리 안정성 문제를 해결했습니다.
- 다중 테이블 업데이트로 인해 트리거되는 병렬 쿼리 안정성 문제를 해결했습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- 버그 #32917: ORPHAN TEMP-POOL 파일을 감지하여 정상적으로 처리
- 버그 #63144: 존재하지 않는 경우 테이블 생성, 메타데이터 잠금이 너무 제한적임

Aurora MySQL 데이터베이스 엔진 업데이트 2018-09-20

버전: 1.18.0

Aurora MySQL 1.18.0이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여, MySQL 5.6과 호환되는 새로운 Aurora MySQL 병렬 쿼리 클러스터는 모두 Aurora MySQL 1.18.0에서 생성됩니다. 기존 병렬 쿼리 클러스터를 Aurora MySQL 1.18.0으로 업그레이드할 수 있지만 필수는 아닙니다. Aurora MySQL 1.14.4, Aurora MySQL 1.15.1, Aurora MySQL 1.16 또는 Aurora MySQL 1.17.6에서 새 DB 클러스터를 생성할 수 있습니다. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.18.0에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

Important

Aurora MySQL 1.18.0은 Aurora 병렬 쿼리 클러스터에만 적용됩니다. 프로비저닝된 5.6.10a 클러스터를 업그레이드하는 경우 결과 버전은 1.17.8입니다. 병렬 쿼리 5.6.10a 클러스터를 업그레이드하는 경우 결과 버전은 1.18.0입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

기능

- 이 릴리스에서는 새로운 클러스터와 복원된 스냅샷에 병렬 쿼리를 사용할 수 있습니다. 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 관점에서 볼 경우 아래로 밀어 내림). 병렬 쿼리가 없으면, 각 쿼리가 스캔한 모든 데이터를 Aurora MySQL 클러스터(헤드 노드) 내의 단일 노드로 가져오고 거기에서 모든 쿼리 처리를 수행합니다.
- 병렬 쿼리 기능이 활성화되면, Aurora MySQL 엔진이 힌트 또는 테이블 속성과 같은 SQL 변경 필요 없이도 쿼리가 혜택을 염을 수 있는 경우를 자동으로 결정합니다.

자세한 내용은 [Amazon Aurora MySQL용 Parallel Query 처리 \(p. 527\)](#) 단원을 참조하십시오.

- OOM Avoidance: 이 기능은 시스템 메모리를 모니터링하고, 데이터베이스의 다양한 구성요소에서 사용한 메모리를 추적합니다. 시스템 메모리가 부족해지면 데이터베이스가 메모리 부족(OOM: Out of Memory) 상태로 실행되지 않도록 하기 위해, 추적된 다양한 구성 요소에서 메모리를 해제하는 작업 목록을 수행함으로써 데이터베이스가 다시 시작되는 문제를 방지합니다. 이 최선의 기능은 t2 인스턴스에 대해 기본적으로 활성화되며, `aurora_oom_response`라는 새 인스턴스 파라미터를 통해 다른 인스턴스 클래스에서도 활성화할 수 있습니다. 이 인스턴스 파라미터는 메모리가 부족할 때 인스턴스가 취해야 할 작업을 쉼표로 구분해 놓은 문자열을 받습니다. 유효한 작업으로는 "print", "tune", "decline", "kill_query" 등이 있으며 이러한 작업을 조합할 수 있습니다. 빈 문자열은 취해야 할 조치가 없음을 의미하므로 해당 기능을 비활성화합니다. 이 기능의 기본 작업은 "print, tune"입니다. 사용 예제:
 - "print" – 많은 양의 메모리를 사용하는 쿼리만 인쇄합니다.
 - "tune" – 내부 테이블 캐시를 조정하여 일부 메모리를 시스템으로 돌려줍니다.
 - "decline" – 인스턴스 메모리가 부족해지면 새 쿼리를 거부합니다.
 - "kill_query" – 인스턴스 메모리가 하한값 이상이 될 때까지 메모리 사용량이 많은 순서로 쿼리를 종료합니다. 데이터 정의 언어(DDL) 설명문이 종료되지 않습니다.
 - "print, tune" – "print" 및 "tune"에 대해 설명한 작업을 수행합니다.

- "tune, decline, kill_query" – "tune", "decline", "kill_query"에 대해 설명한 작업을 수행합니다.

메모리 부족 상태 처리 및 기타 문제 해결 조언에 관한 자세한 내용은 [Amazon Aurora MySQL 메모리 부
족 문제 \(p. 1031\)](#) 단원을 참조하십시오.

Aurora MySQL 데이터베이스 엔진 업데이트 2020-03-05

버전: 1.17.9

Aurora MySQL 1.17.9가 정식 버전입니다. Aurora MySQL 1.* 버전은 MySQL 5.6과 호환되고 Aurora MySQL 2.* 버전은 MySQL 5.7과 호환됩니다.

현재 지원되는 Aurora MySQL 릴리스는 1.14.*, 1.15.*, 1.16.*, 1.17.*, 1.18.*, 1.19.*, 1.20.*, 1.21.*, 1.22.*, 2.01.*, 2.02.*, 2.03.*, 2.04.*, 2.05.*, 2.06.*, 2.07.*입니다. Aurora MySQL 1.* 데이터베이스의 스냅샷을 Aurora MySQL 1.17.9로 복원할 수 있습니다.

이전 버전의 Aurora MySQL로 클러스터를 생성하려면 RDS 콘솔, AWS CLI 또는 Amazon RDS API를 통해 엔진 버전을 지정하십시오.

Note

이 버전은 현재 다음 리전에서 사용할 수 없습니다. AWS GovCloud(US-East) [us-gov-east-1], AWS GovCloud (US-West) [us-gov-west-1]. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

높은 우선 순위:

- 인증서 교체 후 간헐적인 연결 실패 문제를 해결했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2019-01-17

버전: 1.17.8

Aurora MySQL 1.17.8이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여, MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 Aurora MySQL 1.17.8에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.17.8로 업그레이드할 수 있지만 필수는 아닙니다. 이전 버전을 사용 하려면 Aurora MySQL 1.14.4, 1.15.1, 1.16, 1.17.7 등에서 데이터베이스 클러스터를 새로 만드십시오. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.17.8에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- 재시작 후 Aurora 복제본에서 CPU 사용률이 증가하는 성능 문제를 수정했습니다.
- 해시 조인을 사용하는 SELECT 쿼리 관련 안정성 문제를 수정했습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- 버그 #13418638: 존재하지 않는 경우 테이블 생성, 메타데이터 잠금이 너무 제한적임

Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-08

버전: 1.17.7

Aurora MySQL 1.17.7이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여, MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 Aurora MySQL 1.17.7에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.17.7로 업그레이드할 수 있지만 필수는 아닙니다. 이전 버전을 사용 하려면 Aurora MySQL 1.14.4, 1.15.1, 1.16, 1.17.6 등에서 데이터베이스 클러스터를 새로 만드십시오. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.17.7에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- InnoDB 상태 변수 innodb_buffer_pool_size를 고객이 수정할 수 있도록 표시합니다.
- 장애 조치 과정에서 Aurora 클러스터에 발생하는 안정성 문제를 수정했습니다.
- TRUNCATE 작업 실패 후 발생하는 DDL 복구 문제를 수정하여 클러스터 가용성을 높였습니다.
- DDL 작업을 통해 트리거되는 mysql.innodb_table_stats 테이블 업데이트와 관련된 안정성 문제를 수정했습니다.
- DDL 연산 후 쿼리 캐시 무효화 과정에서 트리거되는 Aurora 복제본 안정성 문제를 수정했습니다.
- 백그라운드에서 정기적으로 실행되는 딕셔너리 캐시 제거 중 잘못된 메모리로 인해 트리거되는 안정성 문제를 수정했습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- 버그 #16208542: 누락 테이블을 초래하는 외래 키 열에서 인덱스 삭제.
- 버그 #76349: add_derived_key()의 메모리 누수.
- Bug #16862316: 파티션 분할된 테이블의 경우, 인덱스 병합 사용 여부에 따라 쿼리가 다른 결과를 반환할 수 있습니다.
- Bug #17588348: 인덱스 병합 최적화([인덱스 병합 최적화 참조](#))를 사용하는 쿼리를 HASH로 파티션 분할된 테이블에서 실행할 경우 잘못된 결과가 반환될 수 있습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-09-06

버전: 1.17.6

Aurora MySQL 1.17.6이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여, MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 Aurora MySQL 1.17.6에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.17.6으로 업그레이드할 수 있지만 필수는 아닙니다. 이전 버전을

사용 하려면 Aurora MySQL 1.14.4, 1.15.1, 1.16, 1.17.5 등에서 데이터베이스 클러스터를 새로 만드십시오.
AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.17.6에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- Aurora Writer가 동일한 테이블에서 DDL 작업을 수행하는 동안 SELECT 쿼리에 대한 Aurora Reader의 안정성 문제를 수정했습니다.
- 힐/메모리 엔진을 사용하는 임시 테이블에 대한 DDL 로그 생성 및 삭제로 인해 발생하는 안정성 문제를 수정했습니다.
- Binlog 마스터에 대한 연결이 불안정할 때 DDL 문이 복제되는 경우 Binlog 슬레이브에서 발생하는 안정성 문제를 수정했습니다.
- 느린 쿼리 로그에 쓸 때 발생하는 안정성 문제를 수정했습니다.
- 올바르지 않은 Aurora Reader 지연 시간 정보를 표시하는 복제본 상태 테이블 문제를 수정했습니다.

MySQL 커뮤니티 에디션 버그 픽스 통합

- **BINARY** 열의 이름을 변경하거나 기본값을 변경한 [ALTER TABLE](#) 문의 경우, 인플레이스가 아닌 테이블 복사를 사용하여 변경이 수행되었습니다. (버그 #67141, 버그 #14735373, 버그 #69580, 버그 #17024290)
- 그룹인 정규 테이블과 파생 테이블 간의 외부 조인으로 인해 서버 종료가 발생할 수 있습니다. (버그 #16177639)

Aurora MySQL 데이터베이스 엔진 업데이트 2018-08-14

버전: 1.17.5

Aurora MySQL 1.17.5가 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여, MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 Aurora MySQL 1.17.5에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.17.5로 업그레이드할 수 있지만 필수는 아닙니다. 이전 버전을 사용 하려면 Aurora MySQL 1.14.4, 1.15.1, 1.16, 1.17.4 등에서 데이터베이스 클러스터를 새로 만드십시오. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.17.5에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- 제로 가동 중지 패치 적용 기능을 사용하여 Aurora 클러스터에 패치를 적용한 후 Aurora Writer가 다시 시작될 수 있는 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-08-07

버전: 1.17.4

Aurora MySQL 1.17.4이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여, MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 Aurora MySQL 1.17.4에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.17.4로 업그레이드할 수 있지만 필수는 아닙니다. 이전 버전을 사용 하려면 Aurora MySQL 1.14.4, 1.15.1, 1.16, 1.17.3 등에서 데이터베이스 클러스터를 새로 만드십시오. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.17.4에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- 복제 개선:
 - binlog 레코드를 클러스터 복제본으로 전송하지 않아 네트워크 트래픽을 줄였습니다. 이 개선 사항은 기본적으로 활성화되어 있습니다.
 - 복제 메시지를 압축하여 네트워크 트래픽을 줄였습니다. 이 개선 사항은 8xlarge 및 16xlarge 인스턴스 클래스에 대해 기본적으로 활성화됩니다. 그런 대용량 인스턴스에서는 복제 메시지에 대한 높은 네트워크 트래픽을 발생하는 대용량의 쓰기 트래픽을 유지할 수 있습니다.
 - 복제본 쿼리 캐시에 대한 수정 사항
- ORDER BY LOWER(*col_name*) 데이터 정렬 중에 utf8_bin에서 잘못된 주문을 생성하던 문제가 해결되었습니다.
- 불안정성, 누락된 테이블을 비롯하여 DDL 문(특히, TRUNCATE TABLE 문)로 인해 Aurora 복제본에서 발생하던 문제를 해결했습니다.
- 스토리지 노드를 다시 시작할 때 소켓이 반개방 상태로 유지되던 문제를 해결했습니다.
- 다음과 같은 새로운 DB 클러스터 파라미터를 사용할 수 있습니다.
 - aurora_enable_zdr – Aurora 복제본에서 열려 있는 연결을 복제본 재시작 시 활성 상태로 유지합니다.
 - aurora_enable_replica_log_compression – 마스터와 Aurora 복제본 간의 네트워크 대역폭 활용률을 높이기 위해 복제 페이로드의 압축을 활성화합니다.
 - aurora_enable_repl_bin_log_filtering – Aurora 복제본이 마스터에서 사용할 수 없는 복제 코드에 대해 필터링을 활성화합니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-06-05

버전: 1.17.3

Aurora MySQL 1.17.3이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여, MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 Aurora MySQL 1.17.3에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.17.3으로 업그레이드할 수 있지만 필수는 아닙니다. Aurora MySQL 1.14.4, Aurora MySQL 1.15.1 또는 Aurora MySQL 1.16에서 새 DB 클러스터를 생성할 수 있습니다. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.17.3에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

Note

현재 이 버전은 AWS GovCloud (US-West) [us-gov-west-1] 및 중국(베이징) [cn-north-1] 리전에서 사용할 수 없습니다. 사용 가능해지면 따로 발표할 예정입니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- 레코드를 읽는 동안 낙관적 커서 사용이 복원될 때 Aurora 복제본이 다시 시작할 수 있는 문제를 수정했습니다.
- 성능 스키마가 활성화된 MySQL 세션을 종료하려고 할 때(`kill "<## ID>"`) Aurora Writer가 다시 시작하는 문제를 수정했습니다.
- 가비지 수집 임계값을 계산할 때 Aurora Writer가 다시 시작하는 문제를 수정했습니다.
- 로그 애플리케이션에서 Aurora 복제 진행 상황을 추적할 때 Aurora Writer가 가끔 다시 시작할 수 있는 문제를 수정했습니다.
- 자동 커밋이 꺼져 있을 때 잠재적으로 기한 경과 읽기가 발생할 수 있는 쿼리 캐시 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-04-27

버전: 1.17.2

Aurora MySQL 1.17.2가 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여, MySQL 5.6과 호환되는 새로운 Aurora MySQL 데이터베이스 클러스터는 모두 Aurora MySQL 1.17.2에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.17.2로 업그레이드할 수 있지만 필수는 아닙니다. Aurora MySQL 1.14.4, Aurora MySQL 1.15.1 또는 Aurora MySQL 1.16에서 새 DB 클러스터를 생성할 수 있습니다. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.17.2에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- 특정 DDL 파티션 작업 동안 재시작을 초래하는 문제를 수정했습니다.
- 네이티브 Aurora MySQL 함수를 사용하여 AWS Lambda 함수를 호출하는 지원 기능이 비활성화되는 문제를 수정했습니다.
- Aurora 복제본을 다시 시작하게 만드는 캐시 무효화 문제를 수정했습니다.
- 재시작을 초래하는 잠금 관리자의 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-03-23

버전: 1.17.1

Aurora MySQL 1.17.1이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 새로운 데이터베이스 클러스터는 모두 Aurora MySQL 1.17.1에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.17.1로 업그레이드할 수 있지만 필수는 아닙니다. Aurora MySQL 1.15.1, Aurora MySQL 1.16 또는 Aurora MySQL 1.17.에서 새 DB 클러스터를 생성할 수 있습니다. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora MySQL 버전 1.17.1에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. 이번 릴리스에서는 회귀는 물론 몇 가지 알려진 엔진 문제를 해결합니다.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

Note

최신 버전의 Aurora MySQL 엔진에 문제가 있습니다. 1.17.1로 업그레이드한 후 엔진 버전이 1.17로 잘못 보고됩니다. 1.17.1로 업그레이드한 경우 AWS Management 콘솔에서 DB 클러스터에 대한 유지 관리 열을 확인하여 업그레이드를 확인할 수 있습니다. `none`이라고 표시되면 엔진이 1.17.1로 업그레이드된 것입니다.

개선 사항

- 이진 로그가 자주 교체되는 경우 발생할 수 있는 대용량 이진 로드 인덱스 파일이 있는 상황에서 복구 시간이 더욱 길어지는 이진 로그 복원 문제를 해결했습니다.
- 분할된 테이블에 대한 비효율적인 쿼리 계획을 생성하는 쿼리 옵티마이저의 문제를 해결했습니다.
- 범위 쿼리로 인해 데이터베이스 엔진 재시작이 발생한 쿼리 옵티마이저의 문제를 해결했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트 2018-03-13

버전: 1.17

Aurora MySQL 1.17이 정식 버전입니다. Aurora MySQL 1.x 버전은 MySQL 5.6하고만 호환되며 MySQL 5.7과는 호환되지 않습니다. 스냅샷에서 복원되는 클러스터를 포함하여 새로운 5.6 호환 데이터베이스 클러스터는 모두 Aurora 1.17에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora 1.17로 업그레이드할 수 있지만 필수는 아닙니다. Aurora 1.14.1, Aurora 1.15.1 또는 Aurora 1.16.에서 새 DB 클러스터를 생성할 수 있습니다. 이렇게 하려면 AWS CLI 또는 Amazon RDS API를 사용하고 엔진 버전을 지정하면 됩니다.

Aurora 버전 1.17에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. AWS는 패치 적용 프로세스 도중 클라이언트 연결을 유지하기 위해 최선의 노력을 기울이는 제로 가동 중지 패치 적용을 지원합니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

질문이나 우려 사항이 있는 경우 커뮤니티 포럼이나 [AWS Premium Support](#) 통해 AWS Support 팀에 도움을 요청할 수 있습니다.

제로 가동 중지 패치 적용

제로 가동 중지 패치 적용(ZDP) 기능은 [최선을 다해 엔진 패치 도중 클라이언트 연결을 유지하기 위해 노력합니다](#). ZDP에 대한 자세한 내용은 [제로 가동 중지 패치 적용 \(p. 688\)](#) 단원을 참조하십시오.

새로운 기능

- Aurora MySQL에서 이제 잠금 관리자의 메모리 사용량을 최적화하는 압축 잠금을 지원합니다. 버전 1.17부터는 랩 모드를 활성화하지 않고 이 기능을 사용할 수 있습니다.

개선 사항

- 데이터베이스가 유휴 상태일 때에도 싱글 코어가 100%의 CPU 사용률을 차지할 수 있는 코어 수가 적은 인스턴스에서 주로 발견되는 문제를 해결합니다.
- Aurora 클러스터에서 이진 로그 가져오기 성능이 개선되었습니다.
- 테이블 통계, 영구 스토리지 및 충돌 쓰기를 시도하는 Aurora 복제본 문제를 해결했습니다.
- Aurora 복제본에서 쿼리 캐시가 예상대로 작동하지 않는 문제를 해결했습니다.
- 엔진 재시작으로 이어지는 잠금 관리자의 교착 상태를 해결했습니다.
- 엔진 재시작으로 이어지는 읽기 전용, 자동 커밋 트랜잭션으로 인한 잠금 문제를 해결했습니다.

- 일부 쿼리가 감사 로그에 쓰여지지 않는 문제를 해결했습니다.
- 장애 조치 시 특정 파티션 유지 관리 작업의 복구 문제를 해결했습니다.

MySQL 버그 수정 통합

- 복제 필터가 사용되는 경우 LAST_INSERT_ID가 올바르지 않게 복제됩니다(버그 #69861).
- 쿼리가 INDEX_MERGE 설정에 따라 다른 결과를 반환합니다(버그 #16862316).
- 저장된 루틴의 쿼리 처리 재실행, 비효율적인 쿼리 계획(버그 #16346367)
- INNODB FTS: FTS_CACHE_APPEND_DELETED_DOC_IDS에 어설션합니다(버그 #18079671).
- RBT_EMPTY(INDEX_CACHE->WORDS)를 ALTER TABLE 변경 열에 어설션합니다(버그 #17536995).
- 저장점이 연관되었을 경우 INNODB Fulltext 검색으로 레코드를 찾지 못합니다(버그 #70333, 버그 #17458835).

Aurora MySQL 데이터베이스 엔진 업데이트 2017-12-11

버전: 1.16

Aurora MySQL 1.16이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 새로운 데이터베이스 클러스터는 모두 Aurora 1.16에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora 1.16으로 업그레이드할 수 있지만 필수는 아닙니다. Aurora 1.14.1 또는 Aurora 1.15.1에서 새 DB 클러스터를 생성할 수 있습니다. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora 버전 1.16에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. AWS는 패치 적용 프로세스 도중 클라이언트 연결을 유지하기 위해 최선의 노력을 기울이는 제로 가동 중지 패치 적용을 실시하고 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

궁금하거나 걱정되는 점이 있다면 커뮤니티 포럼이나 AWS Premium Support(<http://aws.amazon.com/support>)를 통해 AWS Support 팀에게 도움을 요청할 수 있습니다.

제로 가동 중지 패치 적용

제로 가동 중지 패치 적용(ZDP) 기능은 [최선을 다해 엔진 패치 도중 클라이언트 연결을 유지하기 위해 노력합니다](#). ZDP에 대한 자세한 내용은 [제로 가동 중지 패치 적용 \(p. 688\)](#) 단원을 참조하십시오.

새로운 기능

- Aurora MySQL은 이제 네이티브 함수 `lambda_sync()`를 통해 동기식 AWS Lambda 호출을 지원합니다. 또한 비동기식 Lambda 호출의 기존 저장 프로시저의 대안으로 사용할 수 있는 네이티브 함수 `lambda_async()`도 사용할 수 있습니다. 자세한 내용은 [Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출 \(p. 633\)](#) 단원을 참조하십시오.
- Aurora MySQL은 이제 해시 조인을 지원하여 동등 조인 쿼리의 속도를 높입니다. Aurora의 비용 기반 최적화로 해시 조인을 사용할 시점을 자동으로 결정할 수 있고, 쿼리 계획에 강제로 사용할 수도 있습니다. 자세한 내용은 [Aurora MySQL에서 해시 조인 작업 \(p. 663\)](#) 단원을 참조하십시오.
- Aurora MySQL은 이제 스캔 배치화를 지원하여 인 메모리 스캔 지향 쿼리의 속도를 크게 높입니다. 이 기능은 일괄 처리로 테이블 전체 스캔, 인덱스 전체 스캔 및 인덱스 범위 스캔의 성능을 향상시킵니다.

개선 사항

- 마스터에서 방금 삭제된 테이블에서 쿼리를 실행할 때 읽기 전용 복제본이 충돌하는 문제를 수정했습니다.
- FULLTEXT 인덱스 수가 아주 많은 데이터베이스 클러스터에서 라이터를 다시 시작할 때 예상보다 복구가 길어지는 문제를 수정했습니다.

- 이전 로그 플러시가 binlog 이벤트에서 `LOST_EVENTS` 인시던트를 초래하는 문제를 수정했습니다.
- 성능 스키마가 활성화될 때 스케줄러의 안정성 문제를 수정했습니다.
- 임시 테이블을 사용하는 하위 쿼리가 부분적 결과를 반환할 수 있는 문제를 수정했습니다.

MySQL 버그 수정 통합

없음

Aurora MySQL 데이터베이스 엔진 업데이트 2017-11-20

버전: 1.15.1

Aurora MySQL 1.15.1이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 새로운 데이터베이스 클러스터는 모두 Aurora 1.15.1에서 생성됩니다. 기존 DB 클러스터를 Aurora 1.15.1로 업그레이드할 수 있지만 필수는 아닙니다. Aurora 1.14.1에서 새 DB 클러스터를 생성할 수 있습니다. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora 버전 1.15.1에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. AWS는 패치 적용 프로세스 도중 클라이언트 연결을 유지하기 위해 최선의 노력을 기울이는 제로 가동 중지 패치 적용을 실시하고 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

궁금하거나 걱정되는 점이 있다면 커뮤니티 포럼이나 AWS Premium Support(<http://aws.amazon.com/support>)를 통해 AWS Support 팀에게 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

제로 가동 중지 패치 적용

제로 가동 중지 패치 적용(ZDP) 기능은 [최선을 다해](#) 엔진 패치 도중 클라이언트 연결을 유지하기 위해 노력합니다. ZDP에 대한 자세한 내용은 [제로 가동 중지 패치 적용 \(p. 688\)](#) 단원을 참조하십시오.

개선 사항

- 적용 세그먼트 선택기에서 같은 세그먼트를 두 번 선택해 특정 조건에서 읽기 지연 시간이 폭증하는 읽기 요청에 대한 문제가 수정되었습니다.
- 스레드 스케줄러에 대한 Aurora MySQL의 최적화로 인해 발생하는 문제를 수정했습니다. 이 문제는 느린 로그에 쓰는 동안 허위 오류를 검토하며 이때 연결된 쿼리 자체는 정상적으로 작동합니다.
- 큰 볼륨(> 5TB)에서 발생하는 읽기 전용 복제본의 안정성 문제를 수정했습니다.
- 가짜 대기 연결 수로 인해 작업자 스레드 수가 계속 증가하는 문제를 수정했습니다.
- 삽입 워크로드 중 긴 세마포어 대기를 유발하는 테이블 풀 문제를 수정했습니다.
- Aurora MySQL 1.15에 포함된 다음 MySQL 버그 수정을 되돌렸습니다.
 - MySQL 인스턴스에서 "SYNC 인덱스 실행"이 지연됩니다(버그 #73816).
 - RBT_EMPTY(INDEX_CACHE->WORDS)를 ALTER TABLE 변경 열에 어설션합니다(버그 #17536995).
 - 저장점이 연관되었을 경우 InnoDB Fulltext 검색으로 레코드를 찾지 못합니다(버그 #70333).

MySQL 버그 수정 통합

없음

Aurora MySQL 데이터베이스 엔진 업데이트 2017-10-24

버전: 1.15

Aurora MySQL 1.15가 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 새로운 데이터베이스 클러스터는 모두 Aurora 1.15에서 생성됩니다. 기존 DB 클러스터를 Aurora 1.15로 업그레이드할 수 있지만 필수는 아닙니다. Aurora 1.14.1에서 새 DB 클러스터를 생성할 수 있습니다. AWS CLI 또는 Amazon RDS API를 사용하여 엔진 버전을 지정할 수 있습니다.

Aurora 버전 1.15에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. 업데이트 후에는 데이터베이스를 다시 시작해야 하므로 다운타임이 20-30초간 발생할 수 있습니다. 다운타임 후에 DB 클러스터를 다시 사용할 수 있습니다. DB 클러스터가 현재 Aurora 1.14 또는 Aurora 1.14.1을 실행하고 있는 경우에는 워크로드에 따라 Aurora MySQL의 제로 가동 중지 패치 적용 기능을 통해 업그레이드 도중에도 클라이언트와 Aurora MySQL 기본 인스턴스의 연결을 유지할 수 있습니다.

궁금하거나 걱정되는 점이 있다면 커뮤니티 포럼이나 AWS Premium Support(<http://aws.amazon.com/support>)를 통해 AWS Support 팀에게 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

제로 가동 중지 패치 적용

제로 가동 중지 패치 적용(ZDP) 기능은 [최선을 다해 엔진 패치 도중 클라이언트 연결을 유지하기 위해 노력합니다](#). ZDP에 대한 자세한 내용은 [제로 가동 중지 패치 적용 \(p. 688\)](#) 단원을 참조하십시오.

새로운 기능

- 비동기식 키 미리 가져오기 – 비동기식 키 미리 가져오기(AKP)는 필요하기 전에 메모리 키를 미리 가져와서 캐싱되지 않은 인덱스 조인 성능을 향상 시킬 수 있는 기능입니다. AKP가 주로 사용되는 사례로는 테이블 용량이 커질수록 인덱스 선택의 폭이 매우 제한적일 때 작은 용량의 외부 테이블과 큰 용량의 내부 테이블 사이의 인덱스 조인이 있습니다. 또한 Multi-Range Read(MRR) 인터페이스가 활성화되어 있을 때 보조-기본 인덱스를 조회하는 데도 AKP가 사용됩니다. 크기가 작아지면서 메모리 제약이 따르는 인스턴스는 경우에 따라 올바른 키 카디널리티를 지정하여 AKP를 사용할 수도 있습니다. 자세한 내용은 [Amazon Aurora의 비동기식 키 미리 가져오기 작업 \(p. 657\)](#) 단원을 참조하십시오.
- 빠른 DDL–[Aurora 1.13 \(p. 747\)](#)에서 릴리스된 기능을 기본값이 포함된 작업으로 확장했습니다. 이번 확장으로 기본값 유무에 상관없이 테이블 끝에 null 값이 허용되는 열을 추가하는 작업에도 빠른 DDL 기능이 적용됩니다. 이 기능은 여전히 Aurora 랩 모드에 있습니다. 자세한 내용은 [빠른 DDL을 이용하는 Amazon Aurora에서의 테이블 수정 \(p. 525\)](#) 단원을 참조하십시오.

개선 사항

- 이전에 WITHIN/CONTAINS 공간 쿼리를 최적화하는 도중 비어있는 결과 집합의 원인이 되었던 계산 오류가 수정되었습니다.
- 파라미터 그룹에서 SHOW VARIABLE 파라미터 값을 변경할 때마다 업데이트된 값이 표시되도록 innodb_buffer_pool_size 명령이 수정되었습니다.
- 적응형 해시 인덱싱이 비활성화되어 있고, 삽입할 레코드가 페이지의 첫 레코드일 때 빠른 DDL을 사용하여 변경된 테이블에 대한 대량 삽입 과정에서 기본 인스턴스의 안정성이 향상되었습니다.
- 사용자가 server_audit_events DB 클러스터 파라미터 값을 default로 설정하려고 할 때 Aurora의 안정성을 개선했습니다.
- Aurora 기본 인스턴스에서 ALTER TABLE 문을 실행한 후에도 다시 시작할 때까지 데이터베이스 문자 세트 변경 사항이 Aurora 복제본에 복제되지 않았던 문제를 수정했습니다.
- 이전에는 기본 인스턴스가 볼륨을 달았더라도 Aurora 복제본을 등록할 수 있었던 기본 인스턴스의 경합 조건을 수정하여 안정성을 높였습니다.
- 대용량 테이블에서 인덱스를 생성하는 과정에서 인덱스 빌드 도중 동시 데이터 조작 언어(DML) 설명문을 활성화하도록 잠금 프로토콜을 변경하여 기본 인스턴스의 성능이 향상되었습니다.
- ALTER TABLE RENAME 쿼리 도중 InnoDB 메타데이터 불일치 문제가 수정되어 안정성이 향상되었습니다. 예: 테이블 t1의 열(c1, c2) 이름이 동일한 ALTER 문 내에서 주기적으로 t1(c2, c3)으로 변경되는 경우
- Aurora 복제본에 활성 워크로드가 없어서 기본 인스턴스가 응답하지 않는 시나리오에서 Aurora 복제본의 안정성이 향상되었습니다.

- Aurora 복제본이 테이블을 명시적으로 잠금 처리하여 복제 스레드가 기본 인스턴스에서 수신되는 DDL 변경 사항을 적용하지 못하도록 차단하는 시나리오에서 Aurora 복제본의 가용성이 향상되었습니다.
- 별도의 세션 2개에서 외부 키와 열을 동시에 테이블에 추가하면서 빠른 DDL이 활성화되어 있을 때 기본 인스턴스의 안정성이 향상되었습니다.
- 쓰기 작업이 지나치게 많은 워크로드에서 제거될 때까지 실행 최소 레코드 자르기를 차단함으로써 기본 인스턴스에서 제거 스레드의 안정성이 향상되었습니다.
- 테이블을 삭제하는 트랜잭션의 커밋 프로세스에서 잠금 해제 순서를 수정하여 안정성이 향상되었습니다.
- Aurora 복제본에서 DB 인스턴스가 스타트업을 완료하지 못하고 포트 3306이 이미 사용 중이라고 메시지를 표시하던 결함 문제가 수정되었습니다.
- 일부 information_schema 테이블(innodb_trx, innodb_lock, innodb_lock_waits)에 대해 SELECT 쿼리가 실행되면서 클러스터 안정성을 떨어뜨렸던 경합 조건 문제가 수정되었습니다.

MySQL 버그 수정 통합

- CREATE USER가 플러그인 및 암호 해시를 허용하지만 암호 해시는 무시합니다(버그 #78033).
- 파티션 분할 엔진은 분할된 인덱스에서 정렬된 항목을 반환할 수 있도록 여러 필드를 읽기 비트 집합에 추가합니다. 이는 불필요한 필드까지 읽으려고 하면서 조인 버퍼의 원인이 됩니다. 분할 필드를 모두 read_set에 추가하지 않는 대신 read_set에서 이미 설정된 접두사 필드를 기준으로 정렬하여 버퍼 문제를 수정하였습니다. key_cmp를 실행하는 경우 첫 번째 필드를 읽어야 하도록 DBUG_ASSERT가 추가되었습니다(버그 #16367691).
- MySQL 인스턴스에서 "SYNC 인덱스 실행"이 지원됩니다(버그 #73816).
- RBT_EMPTY(INDEX_CACHE->WORDS)를 ALTER TABLE 변경 열에 어설션합니다(버그 #17536995).
- 저장점이 연관되었을 경우 InnoDB Fulltext 검색으로 레코드를 찾지 못합니다(버그 #70333).

Aurora MySQL 데이터베이스 엔진 업데이트: 2018-03-13

버전: 1.14.4

Aurora MySQL 1.14.4가 정식 버전입니다. AWS CLI 또는 Amazon RDS API를 사용하고 엔진 버전을 지정하여 Aurora 1.14.4에서 DB 클러스터를 새로 생성할 수 있습니다. 기존 1.14.x DB 클러스터를 Aurora 1.14.4로 업그레이드 할 수 있지만 필수는 아닙니다.

Aurora 버전 1.14.4에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. AWS는 패치 적용 프로세스 도중 클라이언트 연결을 유지하기 위해 최선의 노력을 기울이는 제로 가동 중지 패치 적용을 지원합니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

궁금하거나 걱정되는 점이 있다면 커뮤니티 포럼이나 AWS Premium Support(<http://aws.amazon.com/support>)를 통해 AWS Support 팀에게 도움을 요청할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

제로 가동 중지 패치 적용

제로 가동 중지 패치 적용(ZDP) 기능은 [최선을 다해 엔진 패치 도중 클라이언트 연결을 유지하기 위해 노력합니다](#). ZDP에 대한 자세한 내용은 [제로 가동 중지 패치 적용 \(p. 688\)](#) 단원을 참조하십시오.

새로운 기능

- Aurora MySQL이 이제 db.r4 인스턴스 클래스를 지원합니다.

개선 사항

- 대용량 binlog 이벤트를 쓸 때 LOST_EVENTS가 생성되는 문제를 해결했습니다.

MySQL 버그 수정 통합

- 무시할 수 있는 이벤트는 유효하지 않기 때문에 테스트 대상이 아닙니다(버그 #74683).
- NEW->OLD ASSERT FAILURE 'GTID_MODE > 0'(버그 #20436436)

Aurora MySQL 데이터베이스 엔진 업데이트: 2017-09-22

버전: 1.14.1

Aurora MySQL 1.14.1이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 새로운 데이터베이스 클러스터는 모두 Aurora MySQL 1.14.1에서 생성됩니다. Aurora MySQL 1.14.1은 기존 Aurora MySQL DB 클러스터의 필수 업그레이드 버전이기도 합니다. 자세한 내용은 AWS 개발자 포럼 웹 사이트에서 [Announcement: Extension to Mandatory Upgrade Schedule for Amazon Aurora](#) 단원을 참조하십시오.

Aurora MySQL 버전 1.14.1에서는 Aurora MySQL DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. 업데이트 후에는 데이터베이스를 다시 시작해야 하므로 다운타임이 20-30초간 발생할 수 있습니다. 다운타임 후에 DB 클러스터를 다시 사용할 수 있습니다. DB 클러스터가 현재 버전 1.13 이상을 사용하고 있는 경우에는 워크로드에 따라 Aurora MySQL의 제로 가동 중지 패치 기능을 통해 업그레이드 도중에도 클라이언트와 Aurora MySQL 기본 인스턴스의 연결을 유지할 수 있습니다.

궁금하거나 걱정되는 점이 있다면 커뮤니티 포럼이나 AWS Premium Support(<http://aws.amazon.com/support>)를 통해 AWS Support 팀에게 도움을 요청할 수 있습니다.

개선 사항

- Aurora MySQL 랩 모드에 유지된 Fast DDL 기능의 안정성을 개선하기 위해 삽입 및 제거와 관련된 교착 상태를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트: 2017-08-07

버전: 1.14

Aurora MySQL 1.14가 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 새로운 데이터베이스 클러스터는 모두 Aurora MySQL 1.14에서 생성됩니다. Aurora MySQL 1.14는 기존 Aurora MySQL DB 클러스터의 필수 업그레이드 버전이기도 합니다. 이전 버전의 Aurora MySQL 사용 중단 일정과 함께 별도의 공지 이메일을 발송해 드리겠습니다.

Aurora MySQL 버전 1.14에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. 업데이트 후에는 데이터베이스를 다시 시작해야 하므로 다운타임이 20-30초간 발생할 수 있습니다. 다운타임 후에 DB 클러스터를 다시 사용할 수 있습니다. DB 클러스터가 현재 버전 1.13을 사용하고 있는 경우에는 워크로드에 따라 Aurora의 제로 가동 중지 패치 기능을 통해 업그레이드 도중에도 클라이언트와 Aurora 기본 인스턴스의 연결을 유지할 수 있습니다.

궁금하거나 걱정되는 점이 있다면 커뮤니티 포럼이나 AWS Premium Support(<http://aws.amazon.com/support>)를 통해 AWS Support 팀에게 도움을 요청할 수 있습니다.

제로 가동 중지 패치 적용

제로 가동 중지 패치 적용(ZDP) 기능은 [최선을 다해 엔진 패치 도중 클라이언트 연결을 유지하기 위해 노력합니다](#). ZDP에 대한 자세한 내용은 [제로 가동 중지 패치 적용 \(p. 688\)](#) 단원을 참조하십시오.

개선 사항

- 기본 인덱스가 아닌 보조 인덱스에서 레코드가 발견되었을 때 발생하는 잘못된 "레코드를 찾을 수 없음" 오류가 수정되었습니다.

- 개별 쓰기 작업의 범위가 32개 페이지를 넘는 경우 너무 강력한 디펜시브 어설션 기능(1.12에서 추가됨)으로 인해 발생할 수 있는 안정성 문제가 수정되었습니다. 이러한 상황은 예를 들어 BLOB 값이 클 때 발생할 수 있습니다.
- 테이블 스페이스 캐시와 딕셔너리 캐시의 불일치로 인한 안정성 문제가 수정되었습니다.
- 기본 인스턴스에 연결할 수 있는 최대 시도 수를 초과한 경우 Aurora 복제본이 응답하지 않는 문제가 수정되었습니다. 이제 아무런 작업도 없는 시간이 기본 인스턴스의 상태 확인에 사용되는 하트비트 시간보다 클 경우에는 Aurora 복제본이 다시 시작됩니다.
- ALTER TABLE 같은 명령을 실행할 때 하나의 연결 세션에서 배타적 메타데이터 잠금(MDL)을 설정하려고 하면 동시성이 매우 높아져 발생할 수 있는 라이브록(livelock) 문제가 수정되었습니다.
- 논리적/병렬 미리 읽기가 발생할 경우 Aurora 읽기 전용 복제본의 안정성 문제가 수정되었습니다.
- LOAD FROM S3의 두 가지 개선 사항
 - 기존 재시도 외에 SDK 재시도를 추가로 사용함으로써 Amazon S3 제한 시간 오류를 개선하였습니다.
 - 용량이 매우 큰 파일이나 다수의 파일을 로드할 때 캐싱을 통해 클라이언트 상태를 재사용함으로써 성능을 최적화하였습니다.
- ALTER TABLE 작업 시 빠른 DDL을 사용하여 다음 안정성 문제가 수정되었습니다.
 - ALTER TABLE 문에 다수의 ADD COLUMN 명령이 있고, 열 이름이 오름차순을 따르지 않는 경우
 - 업데이트할 열의 이름 문자열과 내부 시스템 테이블에서 가져오는 해당 이름 문자열이 서로 널 종료 문자(/0)가 다른 경우
- B-트리 분할 작업일 때
- 테이블의 기본 키가 가변 길이일 때
- 전체 텍스트 검색(FTS) 인덱스 캐시를 기본 인스턴스의 인덱스 캐시와 일치시키는 데 시간이 너무 오래 걸릴 때 Aurora 복제본의 안정성 문제를 수정했습니다. 이러한 문제는 기본 인스턴스에서 새롭게 생성된 FTS 인덱스 항목 중 대다수가 아직 디스크로 내려쓰기되지 않은 경우에 발생할 수 있습니다.
- 인덱스 생성 중 발생할 수 있는 안정성 문제가 수정되었습니다.
- 연결 세션당 메모리 사용량을 추적하는 새로운 인프라와 메모리 부족(OOM) 회피 전략을 수립하는 데 사용되는 원격 측정이 추가되었습니다.
- Aurora 복제본에서 ANALYZE TABLE이 잘못 허용되었던 문제가 수정되었습니다. 현재 이 코드는 차단되었습니다.
- 논리적 미리 읽기와 제거 사이의 경쟁 상태로 인해 드물게 데드록(deadlock)이 발생하면서 생기는 안정성 문제가 수정되었습니다.

MySQL 버그 수정 통합

- 이전에는 전체 텍스트 검색이 파생 테이블(FROM 절의 하위 쿼리)과 결합되면서 서버 종료의 원인이 되었습니다. 하지만 이제는 전체 텍스트 작업이 파생 테이블에 따라 결정되는 경우 구체화된 테이블에서 전체 텍스트 검색을 실행할 수 없다는 오류가 서버에서 발생합니다. (버그 #68751, 버그 #16539903)

Aurora MySQL 데이터베이스 엔진 업데이트: 2017-05-15

버전: 1.13

Note

최초 출시 이후 Aurora MySQL 버전 1.13부터 SELECT INTO OUTFILE S3이라는 새로운 기능이 활성화되었으며, 이러한 변경을 반영하여 출시 정보가 업데이트되었습니다.

Aurora MySQL 1.13이 정식 버전입니다. 스냅샷에서 복원되는 클러스터를 포함하여 새로운 데이터베이스 클러스터는 모두 Aurora MySQL 1.13에서 생성됩니다. 기존 데이터베이스 클러스터를 Aurora MySQL 1.13으로 업그레이드 할 수 있지만 필수는 아닙니다. Aurora 버전 1.13에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. AWS는 패치 적용 프로세스 도중 클라이언트 연

결을 유지하기 위해 최선의 노력을 기울이는 제로 가동 중지 패치 적용을 실시하고 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

제로 가동 중지 패치 적용

제로 가동 중지 패치 적용(ZDP) 기능은 [최선을 다해 엔진 패치 도중 클라이언트 연결을 유지하기 위해 노력합니다](#). ZDP에 대한 자세한 내용은 [제로 가동 중지 패치 적용 \(p. 688\)](#) 단원을 참조하십시오.

새로운 기능:

- SELECT INTO OUTFILE S3 – 이제부터 Aurora MySQL는 쿼리 결과를 Amazon S3 버킷의 파일 1개 이상에 업로드할 수 있습니다. 자세한 내용은 [Amazon Aurora MySQL DB 클러스터에서 Amazon S3 버킷의 텍스트 파일로 데이터 저장 \(p. 627\)](#) 단원을 참조하십시오.

개선 사항:

- 복원 시간을 단축하기 위해 엔진 시작 시 CSV 형식 로그 파일 자르기를 구현했습니다. 이제 general_log_backup, general_log, slow_log_backup 및 slow_log 테이블은 데이터베이스 재 시작 시 유지되지 않습니다.
- **test**라는 이름의 데이터베이스의 마이그레이션이 실패하는 문제를 해결했습니다.
- 올바른 잠금 세그먼트를 재사용함으로써 잠금 관리자의 가비지 수집기의 안정성을 개선했습니다.
- 고착 감지 알고리즘 동안 잘못된 어설션을 제거하여 잠금 관리자의 안정성을 개선했습니다.
- 비동기식 복제를 재활성화하고 무부하 또는 일기 전용 워크로드에서 잘못된 복제 지연이 보고되는 관련 문제를 해결했습니다. 버전 1.10에서 복제 파이프라인이 향상되었습니다. 이러한 개선 사항은 Aurora 복제본의 버퍼 캐시에 로그 스트림 업데이트를 적용하기 위한 것입니다. 이는 Aurora 복제본의 읽기 성능 및 안정성을 개선했습니다.
- autocommit=OFF로 인해 서버 재부팅 시까지 예약된 이벤트가 차단되고 장기간 트랜잭션이 열린 상태로 유지되는 문제를 해결했습니다.
- 일반, 감사 및 느린 쿼리 로그가 비동기식 커밋에 의해 처리되는 쿼리를 로깅하지 못하는 문제를 해결했습니다.
- 논리적 미리 읽기(LRA) 기능의 성능을 최대 2.5배 개선했습니다. 이를 위해 B-트리의 중간 페이지 간에 미리 가져오기가 계속됩니다.
- 감사 변수가 불필요한 공간을 트리밍하는 파라미터 확인을 추가했습니다.
- SQL_CALC_FOUND_ROWS 옵션을 사용하고 FOUND_ROWS() 함수를 호출할 때 쿼리가 잘못된 결과를 반환할 수 있는, Aurora MySQL 버전 1.11에서 발생한 회귀 문제를 해결했습니다.
- 메타데이터 잠금 목록이 잘못 형성된 경우 안정성 문제를 해결했습니다.
- sql_mode가 PAD_CHAR_TO_FULL_LENGTH로 설정되고 명령 SHOW FUNCTION STATUS WHERE Db='**string**'이 실행될 때 안정성을 개선했습니다.
- Aurora 버전 업그레이드 후 잘못된 볼륨 일관성 검사로 인해 드물게 인스턴스가 나타나지 않는 경우를 해결했습니다.
- 사용자가 다수의 테이블을 보유하는 경우 Aurora Writer 성능이 저하되는, Aurora MySQL 버전 1.12에서 발생한 성능 문제를 해결했습니다.
- Aurora Writer가 binlog 슬레이브로 구성되고 연결 수가 16,000에 근접할 경우의 안정성 문제를 개선했습니다.
- Aurora 마스터에서 DDL을 실행 중일 때 메타데이터 잠금을 대기하는 도중 연결이 차단될 경우 드물게 Aurora 복제본이 다시 시작하는 문제를 해결했습니다.

MySQL 버그 수정 통합

- 빈 InnoDB 테이블의 경우, 테이블이 비어 있더라도 ALTER TABLE 문을 사용하여 auto_increment 값을 낮출 수 없습니다. (버그 #69882)

- MATCH() ... 긴 문자열을 AGAINST()의 인수로 사용하는 AGAINST 쿼리가 전체 텍스트 검색 인덱스를 사용하는 InnoDB 테이블에서 실행될 경우 오류가 발생할 수 있습니다. (버그 #17640261)
- ORDER BY 및 LIMIT과 조합으로 SQL_CALC_FOUND_ROWS를 처리할 경우 FOUND_ROWS()에 잘못된 결과가 발생할 수 있습니다. (버그 #68458, 버그 #16383173)
- 외부 키가 존재할 경우 ALTER TABLE이 열의 Null 허용 여부를 변경하도록 허용하지 않습니다. (버그 #77591)

Aurora MySQL 데이터베이스 엔진 업데이트: 2017-04-05

버전: 1.12

Aurora MySQL 1.12는 현재 스냅샷으로부터의 복구를 포함하는 새 DB 클러스터 생성의 기본 설정 버전입니다.

이것은 기존 클러스터에 대한 필수 업그레이드가 아닙니다. 집합 규모의 패치를 1.11에 적용 완료한 후에 기존 클러스터를 버전 1.12로 업그레이드하는 옵션이 제공됩니다(Aurora 1.11 출시 정보 (p. 750) 및 해당 [포럼 공지사항](#) 참조). Aurora 버전 1.12에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

새로운 기능

- 빠른 DDL – Aurora MySQL을 사용하여 이제 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. 테이블 복사를 위해 임시 스토리지를 사용하지 않으므로 스몰 인스턴스 클래스의 라지 테이블에 대해서도 DDL 문을 유용하게 만들습니다. 현재 빠른 DDL은 테이블 끝에서 기본값 없이 null이 허용된 열에 대해서만 지원됩니다. 이 기능은 현재 Aurora lab 모드에서 사용할 수 있습니다. 자세한 내용은 [빠른 DDL을 이용하는 Amazon Aurora에서의 테이블 수정 \(p. 525\)](#) 단원을 참조하십시오.
- 볼륨 상태 표시 – 새 모니터링 명령 SHOW VOLUME STATUS를 추가하여 볼륨의 노드 및 디스크 수를 표시합니다. 자세한 내용은 [Aurora DB 클러스터를 위한 볼륨 상태 표시 \(p. 526\)](#) 단원을 참조하십시오.

개선 사항

- 객체 잠금에 대해 할당된 메모리를 더욱 줄이는 압축 잠금에 대한 변경 실행. 이 개선은 lab 모드에서 사용할 수 있습니다.
- 데이터베이스가 유휴 상태이지만 trx_active_transactions 지표가 급속하게 감소하는 문제를 해결했습니다.
- 디스크 및 노드의 오류 시뮬레이션 시 오류 삽입 쿼리 구문과 관련된 잘못된 오류 메시지를 해결했습니다.
- 잠금 관리자의 경합 조건 및 데드 래치와 관련된 여러 문제를 해결했습니다.
- 쿼리 옵티마이저의 버퍼 오버플로우를 유발하는 문제를 해결했습니다.
- 기본 스토리지 노드의 가용 메모리가 부족한 경우 Aurora 읽기 전용 복제본의 안정성 문제를 해결했습니다.
- wait_timeout 파라미터 설정 후 유휴 상태의 연결이 지속되는 문제를 해결했습니다.
- 인스턴스 재부팅 후 query_cache_size가 예상하지 못한 값을 반환하는 문제를 해결했습니다.
- 쓰기가 스토리지로 진행되지 않는 이벤트에서 네트워크를 너무 자주 시험하는 진단 스레드의 결과로 발생하는 성능 문제를 해결했습니다.

MySQL 버그 수정 통합

- 비어 있는 상태로 인해 AUTO_INCREMENT 값이 재설정되는 동안 제거된 테이블을 재로드합니다. (버그 #21454472, 버그 #77743)

- purge_node_t 구조의 불일치로 인해 인덱스 기록이 블록에서 발견되지 않았습니다. 이러한 불일치로 "보조 인덱스 입력 업데이트 오류", "기록을 제거할 수 없음", "삭제 표시되지 않은 보조 인덱스 입력 제거를 시도함" 등의 경고 및 오류 메시지가 표시되었습니다. (버그 #19138298, 버그 #70214, 버그 #21126772, 버그 #21065746)
- qsort 작업에 대한 스택 크기 계산을 잘못하면 스택 오버플로우가 발생합니다. (버그 #73979)
- 룰백 시 인덱스에서 기록이 발견되지 않았습니다. (버그 #70214, 버그 #72419)
- 업데이트 CURRENT_TIMESTAMP의 ALTER TABLE 추가 열 TIMESTAMP가 ZERO-datas 삽입(버그 #17392)

Aurora MySQL 데이터베이스 엔진 업데이트: 2017-02-23

버전: 1.11

출시 후 빠른 시일 안에 모든 Aurora MySQL DB 클러스터를 최신 버전으로 패치할 예정입니다. DB 클러스터는 레거시 프로시저를 이용해 패치되며 다운타임은 약 5~30초입니다.

패치 작업은 사용자가 각 데이터베이스 인스턴스에 지정한 시스템 유지 관리 기간에 진행됩니다. 이 기간은 AWS Management 콘솔을 사용하여 확인하거나 변경할 수 있습니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

또는 DB 클러스터를 선택하고 클러스터 작업을 선택한 후 지금 업그레이드를 선택하여 AWS Management 콘솔에서 바로 패치를 적용하는 방법도 있습니다.

Aurora MySQL 버전 1.11에서는 Aurora DB 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

새로운 기능

- LOAD DATA FROM S3에 대한 MANIFEST 옵션 – 버전 1.8에서 LOAD DATA FROM S3이 제공되었습니다. 이 명령에 대한 옵션이 확장되어, Amazon S3에서 Aurora DB 클러스터에 로드할 때 매니페스트 파일을 사용하여 파일 목록을 지정할 수 있습니다. 따라서 FILE 옵션을 이용해 단일 파일에서 데이터를 로드하거나 PREFIX 옵션을 이용해 위치와 접두사가 같은 다수 파일에서 데이터를 로드하는 방식과는 달리, 하나 이상의 위치에 있는 특정 파일에서 데이터를 쉽게 불러올 수 있습니다. 매니페스트 파일 양식은 Amazon Redshift에서 사용하는 양식과 같습니다. LOAD DATA FROM S3를 MANIFEST 옵션과 함께 사용하는 방법에 대한 자세한 내용은 [매니페스트 파일을 이용해 로드할 데이터 파일 지정 \(p. 623\)](#)을 참조하십시오.
- 공간 인덱싱 기본 사용 – 이 기능은 버전 1.10의 랩 모드에서 공개되었으며, 이제 기본적으로 활성화됩니다. 공간 인덱싱은 공간 데이터를 사용하는 쿼리를 위한 대규모 데이터 세트에서의 쿼리 성능을 향상 시킵니다. 공간 인덱싱 사용에 대한 자세한 내용은 [Amazon Aurora MySQL 및 지형 정보 데이터 \(p. 464\)](#) 단원을 참조하십시오.
- 고급 감사 타이밍 변경 – 이 기능은 감사 데이터베이스 활동을 위한 고성능 시설을 제공하고자 버전 1.10.1에서 공개되었습니다. 이 버전에서는 감사 로그 타임스탬프의 정확성이 1초에서 1マイ크로초로 변경되었습니다. 더욱 정확한 타임스탬프는 감사 이벤트 발생 시 상황을 파악할 때 도움이 됩니다. 감사에 대한 자세한 내용은 [Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 \(p. 552\)](#) 단원을 참조하십시오.

개선 사항

- `thread_handling` 파라미터를 수정해 Aurora의 스레드 풀이 지원하는 유일한 모델인 `multiple-connections-per-thread`에만 설정할 수 있게 했습니다.
- `buffer_pool_size` 또는 `query_cache_size` 파라미터를 DB 클러스터의 총 메모리보다 크게 설정하면 발생하는 문제를 수정했습니다. 이런 상황이 되면, Aurora는 수정된 파라미터를 기본값으로 설정하기 때문에 DB 클러스터를 재시작하면 충돌하지 않게 됩니다.
- 테이블이 다른 거래에서 무효가 되면 쿼리 캐시에서 오래된 읽기 결과가 나오는 문제를 수정했습니다.
- 삭제 표시한 binlog 파일이 바로 삭제되도록 조정되었습니다.

- **tmp**라는 이름으로 생성된 데이터베이스가 단기 스토리지에 저장된 시스템 데이터베이스로 처리되며 Aurora 후발성 스토리지에서 유지되지 않는 문제가 수정되었습니다.
- SHOW TABLES가 특정 내부 시스템 테이블을 배제하도록 수정했습니다. 이러한 변경은 mysqldump가 SHOW TABLES에 있는 모든 파일을 잠그고, 그 결과 내부 시스템 테이블에 대한 쓰기가 금지되어 불필요한 페일오버가 발생하는 일을 막는 데 도움이 됩니다.
- 인수가 InnoDB 테이블의 열인 함수를 불러오는 쿼리에서 임시 테이블을 생성하면 Aurora 복제본이 재시작되는 문제가 수정되었습니다.
- Aurora 복제본 노드에서 메타데이터 잠금 충돌이 발생해 Aurora 복제본이 기본 DB 클러스터 뒤로 밀려 재시작하게 되는 문제가 수정되었습니다.
- 리더 노드의 복제 파이프라인에 있는 데드 래치 때문에 Aurora 복제본이 뒤로 밀려 재시작하게 되는 문제가 수정되었습니다.
- 1테라바이트(TB)가 넘는 암호화 볼륨에서 Aurora 복제본 랙이 과도해지는 문제가 수정되었습니다.
- 시스템 시간 읽는 방식을 개선해 Aurora 복제본 데드 래치 감지를 개선했습니다.
- 저자가 등록을 해제하면 Aurora 복제본이 한 번이 아닌 두 번 재시작하는 문제가 수정되었습니다.
- 임시 통계 때문에 고유하지 않은 인덱스 열에서 통계상의 불일치가 발생하면 Aurora 복제본에서 쿼리 성능이 느려지는 문제가 수정되었습니다.
- Aurora 복제가 관련 쿼리를 처리하고 있을 때 Aurora 복제본에서 DDL 인수를 복제하면 Aurora 복제본이 충돌하는 문제가 수정되었습니다.
- 버전 1.10에서 도입한 복제 파이프라인 개선 기본 설정을 활성화에서 비활성화로 변경했습니다. 이 개선 사항은 로그 스트림 업데이트를 Aurora 복제본의 버퍼 캐시에 적용하고자 도입되었습니다. Aurora 복제본의 읽기 성능과 안정성 개선에는 도움이 되지만, 특정 워크로드의 복제본 지연 시간을 높입니다.
- 진행 중인 DDL 인수와 대기 중인 Parallel Read Ahead가 같은 테이블에서 동시에 발생하면 DDL 거래 실행 단계에서 어설션 오류가 발생하는 문제가 수정되었습니다.
- DB 클러스터 재시작 후에도 존재할 수 있도록 일반 로그와 느린 쿼리 로그를 개선했습니다.
- ACL 모듈의 메모리 소비를 줄여 특정 장기 실행 쿼리의 메모리 부족 문제를 수정했습니다.
- 테이블에 비공간 인덱스가 있고, 쿼리에 공간 술어가 있으며, 플래너가 공간 조건을 인덱스에 부정확하게 밀어 넣으면 발생하는 재시작 문제를 수정했습니다.
- (LOB 같은) 외부에 저장된 초대형 지형 공간 객체의 삭제, 업데이트 또는 소거가 있으면 DB 클러스터가 재시작하는 문제가 수정되었습니다.
- ALTER SYSTEM SIMULATE ... FOR INTERVAL을 이용한 고장 시뮬레이션이 제대로 작동하지 않는 문제가 수정되었습니다.
- 잠금 관리자에서 잘못된 불변량에 유효하지 않은 어설션이 적용되어 발생하는 안정성 문제가 수정되었습니다.
- 버전 1.10에서 도입한 다음과 같은 두 가지 InnoDB 전체 텍스트 검색 개선이 비활성화되었습니다. 일부 까다로운 워크로드에서 안정성 문제가 발생하기 때문입니다.
 - 전체 텍스트 검색 인덱스 캐시 복제 속도 개선을 위해 Aurora 복제본에 대한 읽기 요청 후에만 캐시를 업데이트합니다.
 - FTS 캐시가 디스크에 동기화될 때 MySQL 쿼리가 너무 오래 종지되지 않도록, 캐시 크기가 전체 크기의 10%를 넘어서자마자 별도 스레드에 캐시 동기화 작업을 오프로드합니다. (버그 #22516559, #73816)

MySQL 버그 수정 통합

- ALTER 테이블 DROP 외래 키를 다른 DROP 연산과 동시에 실행하면 테이블이 사라집니다. (버그 #16095573)
- ORDER BY를 사용한 일부 INFORMATION_SCHEMA 쿼리가 예전처럼 파일 정렬 최적화를 사용하지 않습니다. (버그 #16423536)
- FOUND_ROWS ()가 잘못된 테이블 행 수를 반환합니다 (버그 #68458)
- 임시 테이블을 너무 많이 열면 오류가 발생하는 대신 서버가 고장납니다. (버그 #18948649)

Aurora MySQL 데이터베이스 엔진 업데이트: 2017-01-12

버전: 1.10.1

Aurora MySQL의 버전 1.10.1은 옵트인 버전이며 데이터베이스 인스턴스를 패치하는 데 사용되지 않습니다. 새 Aurora 인스턴스를 만들고 기존 인스턴스를 업그레이드하는 데 사용할 수 있습니다. [Amazon RDS 콘솔](#)에서 클러스터를 선택하고 클러스터 작업을 선택한 후 지금 업데이트를 선택하여 패치를 적용할 수 있습니다. 패치 적용 시 데이터베이스 재시작이 필요하여, 일반적으로 5~30초간 가동 중지 후 DB 클러스터 사용을 재개할 수 있습니다. 이 패치는 Aurora 클러스터의 모든 노드가 동시에 패치되는 클러스터 패치 적용 모델을 사용합니다.

새로운 기능

- 고급 감사 – Aurora MySQL은 데이터베이스 활동을 감사하는 데 사용할 수 있는 고성능 고급 감사 기능을 제공합니다. 고급 감사 활성화 및 사용에 대한 자세한 내용은 [Amazon Aurora MySQL DB 클러스터에서 고급 감사 사용 \(p. 552\)](#) 단원을 참조하십시오.

개선 사항

- 동일한 문에서 열을 만들고 해당 열에 인덱스를 추가할 때 공간 인덱싱 문제를 수정했습니다.
- DB 클러스터 재시작 시 공간 통계가 지속되지 않는 문제를 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-12-14

버전: 1.10

새로운 기능

- 가동 중지 없는 패치 – 이 기능을 사용하면 종단 시간 없이 DB 인스턴스에 패치를 적용할 수 있습니다. 즉 클라이언트 애플리케이션의 접속을 끊거나 데이터베이스를 다시 시작하는 일 없이 애플리케이션 업그레이드가 수행됩니다. 이러한 접근 방식은 유지 관리 기간 동안 Aurora DB 클러스터의 가용성을 향상 시킵니다. 성능 스키마에서 그와 같은 임시 데이터는 업그레이드 과정 중에 재설정된다는 점에 유의하십시오. 이 기능은 사용자 시작 패치뿐 아니라 유지 관리 기간 중 서비스 제공 패치에도 적용됩니다.

패치가 시작되면 해당 서비스에서는 열린 잠금, 트랜잭션 또는 임시 테이블이 없는지 확인한 다음, 적당한 기간 동안 대기하여 데이터베이스에 패치가 적용되어 다시 시작할 수 있도록 합니다. 패치 작업이 진행되는 동안(약 5초) 처리량이 떨어지기는 하지만 애플리케이션 세션은 보존됩니다. 적당한 기간을 찾을 수 없는 경우, 패치 작업은 표준 패치 동작으로 디풀트됩니다.

가동 중지 없는 패치는 다음 설명과 같은 일정한 한계 내에서 최대 한도로 이루어집니다.

- 이 기능은 현재 단일 노드 DB 클러스터 또는 다중 노드 DB 클러스터의 라이터 인스턴스에 대한 패치 작업에 사용할 수 있습니다.
- SSL 연결을 이 기능과 함께 사용하는 것은 지원하지 않습니다. 활성화된 SSL 연결이 있는 경우, Amazon Aurora MySQL은 가동 중지 없는 패치를 수행하는 대신에 SSL 연결이 종료되었는지 주기적으로 확인합니다. 연결이 종료되었다면 가동 중지 없는 패치가 수행됩니다. SSL 연결이 2초 이상 지속되는 경우, 가동 중지가 있는 표준 패치 작업이 수행됩니다.
- 이 기능은 Aurora 릴리스 1.10 이상에서 사용할 수 있습니다. 앞으로 우리는 가동 중지 없는 패치를 사용하여 적용할 수 없는 릴리스나 패치에 대해 알아볼 것입니다.
- 이 기능은 바이너리 로깅에 기반을 둔 복제가 활성화되어 있는 경우에는 해당되지 않습니다.
- 공간 인덱싱 – 공간 인덱싱은 공간 데이터를 사용하는 쿼리를 위한 대규모 데이터 세트에서의 쿼리 성능을 향상 시킵니다. 공간 인덱싱 사용에 대한 자세한 내용은 [Amazon Aurora MySQL 및 지형 정보 데이터 \(p. 464\)](#) 단원을 참조하십시오.

이 기능은 기본적으로 비활성화되어 있으며 Aurora 랩 모드를 활성화하면 사용할 수 있습니다. 자세한 내용은 [Amazon Aurora MySQL 랩 모드 \(p. 654\)](#) 단원을 참조하십시오.

- 복제 파이프라인 개선 – Aurora MySQL은 이제 개선된 메커니즘을 사용하여 Aurora Replica의 버퍼 캐시에 로그 스트림 업데이트를 적용합니다. 이 기능은 복제본에 상당한 읽기 로드가 있을 뿐 아니라 마스터에도 쓰기 로드가 심할 때 Aurora 복제본의 읽기 성능 및 안정성을 향상 시킵니다. 이 기능은 기본적으로 활성화되어 있습니다.
- 캐시된 읽기의 워크로드에 대한 처리량 향상 – Aurora MySQL은 이제 래치 프리 동시 알고리즘을 사용하여 읽기 뷰를 실행합니다. 이에 따라 버퍼 캐시가 처리하는 읽기 쿼리에 대한 처리량이 향상됩니다. 이를 포함한 여러 개선점 덕분에 Amazon Aurora MySQL는 SysBench 선택 전용 워크로드에 대한 MySQL 5.7의 초당 164K 읽기에 비해 초당 최대 625K 읽기의 처리량을 달성할 수 있습니다.
- 핫 행 경합이 있는 워크로드의 처리량 향상 – Aurora MySQL는 특히 핫 페이지 경합이 있는 경우(즉 같은 페이지에서 행에 대해 여러 트랜잭션이 경합하는 경우) 성능을 향상시키는 새로운 잠금 해제 알고리즘을 사용합니다. 이는 TPC-C 벤치마크를 사용한 테스트에서 MySQL 5.7에 비해 분당 트랜잭션 처리량이 최대 16배까지 향상되는 결과를 나타날 수 있습니다. 이 기능은 기본적으로 비활성화되어 있으며 Aurora 랩 모드를 활성화하면 사용할 수 있습니다. 자세한 내용은 [Amazon Aurora MySQL 랩 모드 \(p. 654\)](#) 단원을 참조하십시오.

개선 사항

- 전체 텍스트 검색 인덱스 캐시 복제 속도는 Aurora 복제본에 대한 읽기 요청 후에만 캐시를 업데이트하여 향상 시킬 수 있습니다. 이러한 방식을 통해 복제 스레드가 디스크에서 읽지 못하도록 방지합니다.
- 데이터베이스 이름 또는 테이블 이름에 특수 문자가 있는 테이블에 대한 Aurora 복제본에서 사전 캐시 무효화가 작동하지 않는 문제를 해결하였습니다.
- 스토리지 열 관리가 활성화된 상태에서 분산 스토리지 노드에 대한 데이터 마이그레이션 도중 발생하는 STUCK IO 문제를 해결하였습니다.
- 트랜잭션을 룰백 또는 커밋할 준비를 할 때 트랜잭션 잠금 대기 스레드에 대한 어설션 점검이 실패하는 경우 잠금 관리자에서 발생하는 문제를 해결하였습니다.
- 참조 개수를 사전 테이블 항목에 올바르게 업데이트하여 손상된 사전 테이블을 열 때 발생하는 문제를 해결하였습니다.
- DB 클러스터 최소 읽기 포인트를 느린 Aurora 복제본이 보류할 수 있는 버그를 수정하였습니다.
- 쿼리 캐시의 잠재적 메모리 누수 문제를 해결하였습니다.
- 저장된 프로시저의 IF 문에서 쿼리가 사용될 때 Aurora 복제본이 테이블에 행 수준 잠금을 배치하는 버그를 수정하였습니다.

MySQL 버그 수정 통합

- 파생된 테이블의 UNION이 '1=0/false' 절이 있는 잘못된 결과를 반환합니다. (버그 #69471)
- 저장된 프로시저 2차 실행 시 ITEM_FUNC_GROUP_CONCAT::FIX_FIELDS에서 서버가 충돌합니다. (버그 #20755389)
- 캐시 크기가 전체 크기의 10%를 넘어서자마자 별도 스레드에 캐시 동기화 작업을 오프로드하여 FTS 캐시 동기화 중에 MySQL 쿼리가 너무 오래 종지되는 일을 방지합니다. (버그 #22516559, #73816)

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-11-10

버전: 1.9.0, 1.9.1

새로운 기능

- 향상된 인덱스 빌드 – 이제 상향식으로 인덱스를 빌드하여 보조 인덱스를 빌드하므로 불필요한 페이지 분할이 방지됩니다. 따라서 인덱스를 만들거나 테이블을 다시 빌드하는 데 필요한 시간을 75%까지 줄일 수 있습니다.(db.r3.8xlarge DB 인스턴스 클래스 기준). 이 기능은 Aurora MySQL 버전 1.7의 랩 모드에 있었으며 Aurora 버전 1.9 이상에서는 기본값으로 활성화되어 있습니다. 자세한 내용은 [Amazon Aurora MySQL 랩 모드 \(p. 654\)](#) 단원을 참조하십시오.

- 압축 잠금(랩 모드) – 실행 시 잠금 관리자가 소모하는 메모리 양이 66%까지 대폭 감소합니다. 메모리 부족 예외가 발생하는 일 없이 더 많은 행 잠금을 얻을 수 있습니다. 이 기능은 기본적으로 비활성화되며 Aurora 랩 모드를 설정하여 활성화할 수 있습니다. 자세한 내용은 [Amazon Aurora MySQL 랩 모드 \(p. 654\)](#) 단원을 참조하십시오.

- 성능 스키마 – Aurora MySQL에 이제 성능 영향을 최소화한 성능 스키마 지원이 포함됩니다. SysBench를 이용한 자사 테스트에서 성능 스키마 이용 시 MySQL 성능이 60%까지 저하될 수 있습니다.

Aurora DB 클러스터의 SysBench 테스트에서는 MySQL보다 4배 적은 성능 영향을 보여주었습니다. db.r3.8xlarge DB 인스턴스 클래스를 실행한 결과 성능 스키마를 활성화했음에도 100K SQL 쓰기/초, 550K 이상의 SQL 읽기/초였습니다.

- 핫 행 경합 개선 – 이 기능은 다량의 연결에 의해 핫 행이 소량 액세스할 때 CPU 활용을 떨어뜨리고 처리량을 늘립니다. 또한 핫 행 경합이 있을 때 error 188을 제거합니다.
- 메모리 부족 취급 개선 – 필수적이지 않은 잠금 SQL 명령문을 실행하고 예약된 메모리 풀을 넘어갈 때, Aurora가 해당 SQL 명령문을 강제로 룸백합니다. 이 기능은 메모리를 비우고 메모리 부족 예외로 인한 엔진 충돌을 예방합니다.
- 스마트 읽기 선택기 – 실행 시 모든 읽기 작업에 대하여 각기 다른 세그먼트 중에서 최적의 스토리지 세그먼트를 선택함으로써 읽기 지연을 개선하며, 이로써 읽기 처리량이 개선됩니다. SysBench 테스트 결과 쓰기 워크로드 성능이 27%까지 올라갔습니다.

개선 사항

- 엔진 시동 시 Aurora 복제본이 잠금이 공유된 곳을 발견한 경우의 문제를 수정했습니다.
- 제거 시스템에서 뷰 포인터 읽기가 NULL인 경우 Aurora 복제본에서 잠재적 충돌을 수정했습니다.

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-10-26

버전: 1.8.1

개선 사항

- 트리거를 이용해 AWS Lambda를 호출하는 BULK INSERT 절차가 실패하는 문제를 수정했습니다.
- 자동 커밋이 전역적으로 깨진 경우 카탈로그 마이그레이션이 실패하는 문제를 수정했습니다.
- SSL을 이용해 Aurora로 연결이 실패하는 문제를 해결하였고 LogJam 공격에 대처하도록 Diffie-Hellman 그룹을 개선했습니다.

MySQL 버그 수정 통합

- LogJam 문제 때문에 OpenSSL이 Diffie-Hellman 키 길이 파라미터를 변경했습니다. (버그 #18367167)

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-10-18

버전: 1.8

새로운 기능

- AWS Lambda 통합 – 이제 mysql.lambda_async 프로시저를 사용하여 Aurora DB 클러스터에서 AWS Lambda 함수를 비동기적으로 호출할 수 있습니다. 자세한 내용은 [Amazon Aurora MySQL DB 클러스터에서 Lambda 함수 호출 \(p. 633\)](#) 단원을 참조하십시오.
- Amazon S3에서 데이터 로드 – 이제 LOAD DATA FROM S3 또는 LOAD XML FROM S3 명령을 사용하여 Amazon S3 버킷의 텍스트 또는 XML 파일을 Aurora DB 클러스터로 로드할 수 있습니다. 자세한 내용은 [Amazon S3 버킷의 텍스트 파일에서 Amazon Aurora MySQL DB 클러스터로 데이터 로드 \(p. 620\)](#) 단원을 참조하십시오.

- 카탈로그 마이그레이션 – 이제 Aurora에서 버전 관리를 지원하기 위해 클러스터 볼륨에 카탈로그 메타데이터를 유지합니다. 따라서 버전 간이나 복원 시에 카탈로그를 원활하게 마이그레이션할 수 있습니다.
- 클러스터 수준 유지 관리 및 패치 적용 – 이제 Aurora에서 전체 DB 클러스터에 대한 유지 관리 업데이트를 관리합니다. 자세한 내용은 [Amazon Aurora DB 클러스터 유지 관리 \(p. 306\)](#) 단원을 참조하십시오.

개선 사항

- 인플라이트 DDL 테이블에 대한 메타데이터 잠금을 허용하지 않을 경우 Aurora 복제본이 충돌하는 문제를 해결했습니다.
- `log_output=TABLE`일 때 느린 로그 및 일반 로그 CSV 파일의 로테이션을 간소화하기 위해 Aurora 복제본에 비 InnoDB 테이블 수정을 허용했습니다.
- 기본 인스턴스에서 Aurora 복제본으로 통계로 업데이트할 때 시간 지연을 해결했습니다. 이 수정을 적용하지 않을 경우 Aurora 복제본의 통계가 기본 인스턴스의 통계와 동기화될 수 없으므로 Aurora 복제본에 미달된 다른 쿼리 계획이 생성될 수 있습니다.
- Aurora 복제본에서 잠금을 획득하지 못하는 경합 조건을 해결했습니다.
- 기본 인스턴스를 등록 또는 등록 취소하는 Aurora 복제본에서 장애가 발생할 수 있는 경합 시나리오를 해결했습니다.
- 볼륨을 열거나 닫을 때 `db.r3.large` 인스턴스에서 교착 상태가 발생할 수 있는 경합 조건을 해결했습니다.
- Aurora 분산 스토리지 서비스에서 대량 쓰기 워크로드와 장애가 결합되어 발생할 수 있는 메모리 부족 문제를 해결했습니다.
- 장기 실행 트랜잭션이 존재할 때 제거 스레드 회전으로 인한 높은 CPU 사용 문제를 해결했습니다.
- 과도한 로드 조건에서 잠금에 대한 정보를 가져오기 위해 정보 스키마 쿼리를 실행할 때 발생하는 문제를 해결했습니다.
- 경합으로 인해 스토리지 노드에 대한 Aurora 쓰기가 중지되었다가 재시작/장애 조치되는 진단 프로세스 문제를 해결했습니다.
- `CREATE TABLE [if not exists]` 문을 처리하는 동안 충돌이 발생할 경우 충돌 복구 중에 성공적으로 생성된 테이블이 삭제될 수 있는 조건을 해결했습니다.
- 카탈로그 완화를 사용하여 일반 로그와 느린 로그를 디스크에 저장하지 않을 경우 로그 순환 절차가 중단되는 문제를 해결했습니다.
- 사용자가 사용자 정의 함수 내에서 임시 테이블을 만든 다음 선택한 쿼리 목록에서 사용자 정의 함수를 사용할 때 충돌이 발생하는 문제를 해결했습니다.
- GTID 이벤트를 재생할 때 발생하는 충돌을 해결했습니다. GTID는 Aurora MySQL에서 지원되지 않습니다.

MySQL 버그 수정 통합:

- 여러 인덱스를 포함하는 하나의 열에 모든 인덱스를 끌어서 놓은 경우 외래 키 제약 조건에서 인덱스를 요구할 때 InnoDB에서 DROP INDEX 작업을 차단하지 못합니다. (버그 #16896810)
- 외래 키 제약 조건 추가 충돌을 해결합니다. (버그 #16413976)
- 저장 프로시저에서 커서를 가져오는 동시에 테이블을 분석하거나 플러시할 때 발생하는 충돌을 해결했습니다. (버그 #18158639)
- 사용자가 테이블에서 AUTO_INCREMENT 값을 최대 자동 증분 열 값보다 작게 변경할 때 발생하는 자동 증분 버그를 해결했습니다. (버그 #16310273)

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-09-20

버전: 1.7.1

개선 사항

- InnoDB 전체 텍스트 검색 캐시가 가득 차 있을 때 Aurora 복제본이 중단되는 문제를 해결합니다.
- 스레드 폴의 작업자 스레드가 스스로를 기다릴 때 데이터베이스 엔진이 중단되는 문제를 해결합니다.
- 테이블의 메타데이터 잠금으로 인해 교착 상태가 발생할 때 Aurora 복제본이 중단되는 문제를 해결합니다.
- 스레드 폴의 두 작업자 스레드 간 경합 조건으로 인해 데이터베이스 엔진이 중단되는 문제를 해결합니다.
- 과도한 로드 조건에서 분산 스토리지 하위 시스템에 대한 쓰기 작업의 진전 상태를 모니터링 에이전트가 감지하지 못하는 경우 발생하는 불필요한 장애 조치의 문제를 해결합니다.

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-08-30

버전: 1.7.0

새로운 기능

- NUMA 인식 스케줄러 – Aurora MySQL 엔진의 작업 스케줄러가 이제 NUMA(Non-Uniform Memory Access)를 인식합니다. 이로 인해 db.r3.8xlarge DB 인스턴스 클래스에 대한 처리량 성능이 향상되어 CPU 간 소켓 경합이 최소화됩니다.
- 병렬 미리읽기가 백그라운드에서 비동기적으로 작동 – 전용 스레드를 사용하여 스레드 경합을 줄임으로써 성능을 개선하도록 병렬 미리읽기 기능을 수정했습니다.
- 향상된 인덱스 빌드(랩 모드) – 이제 상향식으로 인덱스를 빌드하여 보조 인덱스를 빌드하므로 불필요한 페이지 분할이 방지됩니다. 따라서 인덱스를 만들거나 테이블을 다시 빌드하는 데 필요한 시간을 줄일 수 있습니다. 이 기능은 기본적으로 비활성화되며 Aurora 랩 모드를 설정하여 활성화할 수 있습니다. 자세한 내용은 [Amazon Aurora MySQL 랩 모드 \(p. 654\)](#) 단원을 참조하십시오.

개선 사항

- 인스턴스에 대해 요청되는 연결 수가 급증할 경우 연결을 설정하는 데 오래 걸리던 문제를 해결했습니다.
- ALTER TABLE을 InnoDB를 사용하지 않는 분할된 테이블에서 실행할 경우 충돌이 발생하던 문제를 해결했습니다.
- 높은 쓰기 워크로드로 인해 장애 조치가 발생할 수 있던 문제를 해결했습니다.
- RENAME TABLE을 분할된 테이블에서 실행할 경우 오류가 발생하던 잘못된 어설션을 해결했습니다.
- 삽입 중심 워크로드 중에 트랜잭션을 롤백할 때 안정성이 향상되었습니다.
- 전체 텍스트 검색 인덱스가 Aurora 복제본에서 실행되지 않던 문제를 해결했습니다.

MySQL 버그 수정 통합

- LOCK_grant 잠금을 분할하여 확장성을 향상했습니다. (포트 WL #8355)
- 저장 프로시저의 SELECT에 커서를 두면 segfault가 발생합니다. (포트 버그 #16499751)
- MySQL에서는 일부 특수한 경우에 잘못된 결과를 제공합니다. (버그 #11751794)
- #11751794 버그에 대한 패치에 의해 GET_SEL_ARG_FOR_KEYPART에서 충돌이 발생합니다. (버그 #16208709)
- GROUP BY를 통한 간단한 쿼리에 대해 잘못된 결과가 표시됩니다. (버그 #17909656)
- 범위 조건자를 통한 semijoin 쿼리에서 추가 행이 표시됩니다. (버그 #16221623)
- IN 하위 쿼리 뒤에 ORDER BY 절을 추가하면 중복 행이 반환될 수 있습니다. (버그 #16308085)
- 쿼리에 대한 설명이 GROUP BY, MyISAM에 대한 간략 스캔과 충돌합니다. (버그 #16222245)
- 인용된 int 조건자를 사용하여 느슨한 인덱스 스캔을 수행하면 임의의 데이터가 반환됩니다. (버그 #16394084)

- 최적화 프로그램에서 느슨한 인덱스 스캔을 사용 중인 경우 임시 테이블을 생성하려고 하면 서버가 종료될 수 있습니다. (버그 #16436567)
- COUNT(DISTINCT)는 NULL 값을 계산하지 않지만 최적화 프로그램에서 느슨한 인덱스 스캔을 사용하는 경우에는 계산됩니다. (버그 #17222452)
- 쿼리에 MIN() MAX() 및 aggregate_function(DISTINCT)이 모두 포함되어 있고(예: SUM(DISTINCT)) 느슨한 인덱스 스캔을 사용하여 쿼리를 실행한 경우 MIN() MAX()의 결과 값이 잘못 설정되었습니다. (버그 #17217128)

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-06-01

버전: 1.6.5

새로운 기능

- 효율적인 이진 로그 저장 – 효율적인 이진 로그 저장 기능은 이제 모든 Aurora MySQL DB 클러스터에서 기본적으로 활성화되며 구성할 수 없습니다. 효율적인 이진 로그 저장은 2016년 4월 업데이트에 소개되었습니다. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트: 2016-04-06 \(p. 757\)](#) 단원을 참조하십시오.

개선 사항

- 기본 인스턴스에 과도한 워크로드가 발생하는 경우 Aurora 복제본의 안전성이 개선되었습니다.
- 분할된 테이블 및 테이블 이름에 특수 문자가 있는 테이블에서 쿼리를 실행할 때 Aurora 복제본의 안전성이 개선되었습니다.
- 보안 연결을 사용할 때 연결 문제가 수정되었습니다.

MySQL 버그 수정 통합

- 마스터 충돌 복구 후 슬레이브에서 복제를 진행할 수 없음(포트 버그 #17632285)

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-04-06

버전: 1.6

이 업데이트에는 다음의 기능 향상이 포함되어 있습니다.

새로운 기능

- 병렬 미리읽기 – 병렬 미리읽기는 이제 모든 Aurora MySQL DB 클러스터에서 기본적으로 활성화되며 구성할 수 없습니다. 병렬 미리읽기는 2015년 12월 업데이트에서 소개되었습니다. 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트: 2015-12-03 \(p. 759\)](#) 단원을 참조하십시오.

이 릴리스에는 병렬 미리읽기가 기본적으로 활성화되어 있을 뿐 아니라 병렬 미리읽기에 대한 다음과 같은 개선 사항이 포함되어 있습니다.

- 로직을 개선하여 병렬 미리읽기가 덜 적극적인 상태가 되도록 합니다. 이렇게 하면 DB 클러스터에 많은 병렬 워크로드가 발생할 경우 도움이 됩니다.
- 더 작은 테이블에서 안전성이 개선되었습니다.
- 효율적인 이진 로그 저장(랩 모드) – 이제 Aurora MySQL에서 MySQL 이진 로그 파일이 더 효율적으로 저장됩니다. 새로운 저장을 구현함으로써 이진 로그 파일을 이전보다 미리 삭제할 수 있으며, 이진 로그 복제 마스터인 Aurora MySQL DB 클러스터의 인스턴스에 대한 시스템 성능이 개선됩니다.

효율적인 이진 로그 저장을 활성화하려면 기본 인스턴스나 Aurora 복제본의 파라미터 그룹에서 `aurora_lab_mode` 파라미터를 1로 설정합니다. `aurora_lab_mode` 파라미터는 기본적으로

default.aurora5.6 파라미터 그룹에 속하는 인스턴스 수준 파라미터입니다. DB 파라미터 그룹 수정에 대한 자세한 내용은 [DB 파라미터 그룹의 파라미터 수정 \(p. 174\)](#) 단원을 참조하십시오. 파라미터 그룹 및 Aurora MySQL에 대한 자세한 내용은 [Aurora MySQL 파라미터 \(p. 665\)](#) 단원을 참조하십시오.

MySQL 이진 로그 복제 마스터 인스턴스인 Aurora MySQL DB 클러스터에서 인스턴스에 대한 효율적인 이진 로그 저장 기능만 설정합니다.

- AURORA_VERSION 시스템 변수 – 이제 AURORA_VERSION 시스템 변수를 쿼리하여 해당 Aurora MySQL DB 클러스터의 Aurora 버전을 확인할 수 있습니다.

Aurora 버전을 확인하려면 다음 쿼리 중 하나를 사용하십시오.

```
select AURORA_VERSION();
select @@aurora_version;
show variables like '%version';
```

DB 클러스터를 수정하는 경우 또는 [describe-db-engine-versions](#) AWS CLI 명령이나 [DescribeDBEngineVersions](#) API 작업을 호출하여 AWS Management 콘솔에서 Aurora 버전을 확인할 수도 있습니다.

- 잠금 관리자 메모리 사용량 측정치 – 잠금 관리자 메모리 사용량에 대한 정보가 이제 측정치로 제공됩니다.

잠금 관리자 메모리 사용량 지표를 가져오려면 다음 쿼리 중 하나를 사용합니다.

```
show global status where variable_name in ('aurora_lockmgr_memory_used');
select * from INFORMATION_SCHEMA.GLOBAL_STATUS where variable_name in
('aurora_lockmgr_memory_used');
```

개선 사항

- binlog 및 XA 트랜잭션 복구 중 안전성이 개선되었습니다.
- 많은 연결로 발생하는 메모리 문제가 수정되었습니다.
- Read Throughput, Read IOPS, Read Latency, Write Throughput, Write IOPS, Write Latency 및 Disk Queue Depth 지표의 정확성이 개선되었습니다.
- 충돌 후 대량 인스턴스가 느리게 시작되는 안전성 문제가 수정되었습니다.
- 동기화 메커니즘 및 캐시 제거에 관련된 데이터 사전의 동시성이 개선되었습니다.
- Aurora 복제본의 안정성 및 성능 개선 사항:
 - 기본 인스턴스에 대한 과도하거나 폭증하는 쓰기 워크로드 중 Aurora 복제본의 안정성 문제가 수정되었습니다.
 - db.r3.4xlarge 및 db.r3.8xlarge 인스턴스의 복제 지연이 개선되었습니다.
 - 애플리케이션 로그 레코드와 Aurora 복제본 동시 읽기 사이의 경합이 감소되어 성능이 개선되었습니다.
 - 새로 생성되거나 업데이트된 통계로 Aurora 복제본의 통계를 새로 고치는 문제가 수정되었습니다.
 - 기본 인스턴스에 트랜잭션이 많고 동일한 데이터의 Aurora 복제본에 대한 동시 읽기가 많은 경우 Aurora 복제본의 안정성이 개선되었습니다.
 - UPDATE 및 DELETE 문을 JOIN 문과 함께 실행할 때 Aurora 복제본의 안정성이 개선되었습니다.
 - INSERT ... SELECT 문을 실행하는 경우 Aurora 복제본의 안정성이 개선되었습니다.

MySQL 버그 수정 통합

- 백포트 버그 #18694052 5.6에서 '!M_ORDERED_REC_BUFFER' 어설션 실패 수정(포트 버그 #18305270)

- MEMCPY(), HA_PARTITION::POSITION의 SEGV 오류(포트 버그 # 18383840)
- 파티셔닝, INDEX_MERGE 및 NO PK에서 잘못된 결과 도출(포트 버그 # 18167648)
- FLUSH TABLES FOR EXPORT: HA_PARTITION::EXTRA의 어설션 오류(포트 버그 # 16943907)
- 가상 HA_ROWS HANDLER::MULTI_RANGE_READ_INFO_CONST에서 서버 충돌(포트 버그 # 16164031)
- SEL_ARG::RB_INSERT()에서 범위 최적화 프로그램 충돌(포트 버그 # 16241773)

Aurora MySQL 데이터베이스 엔진 업데이트: 2016-01-11

버전: 1.5

이 업데이트에는 다음의 기능 향상이 포함되어 있습니다.

개선 사항

- Aurora 스토리지 배포 중 유휴 인스턴스에 발생하는 10초 간의 쓰기 작업 중지 오류를 수정했습니다.
- 이제 innodb_file_per_table이 No로 설정된 경우 논리적 미리읽기가 작동합니다. 논리적 미리읽기에 대한 자세한 내용은 [Aurora MySQL 데이터베이스 엔진 업데이트: 2015-12-03 \(p. 759\)](#) 단원을 참조하십시오.
- Aurora 복제본이 기본 인스턴스에 다시 연결되는 문제를 해결했습니다. 이에 따라 오류 삽입 쿼리를 사용하여 Aurora 복제본을 테스트할 때 quantity 파라미터에 큰 값을 지정하는 경우 발생하는 문제도 해결되었습니다. 자세한 내용은 [Aurora 복제본 실패 테스트 \(p. 523\)](#) 단원을 참조하십시오.
- 속도가 느려 다시 시작되는 Aurora 복제본을 모니터링하는 기능이 개선되었습니다.
- Aurora 복제본의 속도를 느리게 하고 등록이 취소되게 한 다음, 다시 시작되도록 하는 문제를 해결했습니다.
- 교착 상태 중에 show innodb status 명령을 실행하는 경우 발생하는 문제를 해결했습니다.
- 쓰기 처리량이 높을 때 크기가 큰 인스턴스에 대한 장애 조치 문제를 해결했습니다.

MySQL 버그 수정 통합

- 데이터베이스 이름이 숫자로 시작하는 테이블에 영향을 주는 MySQL 전체 텍스트 검색 기능이 완전히 해결되지 않았던 문제를 해결했습니다. (포트 버그 #17607956)

Aurora MySQL 데이터베이스 엔진 업데이트: 2015-12-03

버전: 1.4

이 업데이트에는 다음의 기능 향상이 포함되어 있습니다.

새로운 기능

- 빠른 입력 기능 – 기본 키에 의해 정렬되는 병렬 입력을 빠르게 처리해 줍니다. 자세한 내용은 [Amazon Aurora MySQL 성능 개선 사항 \(p. 463\)](#) 단원을 참조하십시오.
- 대용량 데이터 세트 읽기 성능 – Aurora MySQL은 입출력(IO)이 많은 워크로드를 자동으로 탐지하여 추가 스레드를 시작함으로써 DB 클러스터의 성능을 높여 줍니다. Aurora 스케줄러가 입출력 작업을 확인하고 시스템의 최적 스레드 수를 자동으로 조정할지 결정하여 낮은 오버헤드로 입출력 중심 워크로드와 CPU 중심 워크로드 간을 빠르게 조정합니다.
- 병렬 미리 읽기 – 기본 인스턴스나 Aurora 복제본(범위 쿼리 포함)에서 사용할 수 있는 메모리에 비해 너무 큰 B-트리 스캔의 성능을 개선합니다. 병렬 미리읽기는 페이지 읽기 패턴을 자동으로 탐지하여 페이지가 필요하기 전에 버퍼 캐시에 페이지를 미리 가져옵니다. 병렬 미리읽기는 동일한 트랜잭션 내에서 동시에 여러 테이블에 작동합니다.

개선 사항:

- Aurora 스토리지 배포 종 발생하는 간단한 Aurora 데이터베이스 가용성 문제를 해결했습니다.
- max_connection 제한을 올바르게 적용했습니다.
- Aurora가 binlog 마스터이고 대용량 데이터 로드 이후 데이터베이스가 다시 시작되는 경우 binlog 제거 문제를 개선했습니다.
- 테이블 캐시에 발생하는 메모리 관리 문제를 해결했습니다.
- 빠른 복구를 위해 공유된 메모리 버퍼 캐시에 방대한 페이지에 대한 지원을 추가했습니다.
- 스레드 로컬 스토리지가 초기화되지 않는 문제를 해결했습니다.
- 기본적으로 16K 연결이 허용됩니다.
- 입출력 중심 워크로드에 대해 동적 스레드 풀이 지원됩니다.
- 쿼리 캐시에 UNION을 포함하는 뷰를 제대로 무효화할 때 발생하는 문제를 해결했습니다.
- 사전 통계 스레드에서 발생하는 안정성 문제를 해결했습니다.
- 캐시 제거와 관련하여 사전 하위 시스템에서 발생하는 메모리 누수 문제를 해결했습니다.
- 마스터에 쓰기 로드가 매우 낮은 경우 Aurora 복제본에서 읽기 지연 시간이 크게 발생하는 문제를 해결했습니다.
- 마스터에서 ALTER TABLE ... REORGANIZE PARTITION을 수행하는 것처럼 DDL 분할된 테이블에서 작업을 수행할 때 Aurora Replicas에서 발생하는 안정성 문제를 해결했습니다.
- 볼륨이 커지는 동안 Aurora 복제본에서 발생하는 안정성 문제를 해결했습니다.
- Aurora 복제본의 비클러스터 인덱스에서 스캔 수행 시 발생하는 성능 문제를 해결했습니다.
- Aurora 복제본의 속도를 느리게 하고 결국 등록이 취소되게 한 다음, 다시 시작되도록 하는 안정성 문제를 해결했습니다.

MySQL 버그 수정 통합

- FTSPARSE()의 SEGV 오류 (버그 #16446108)
- 열의 이름을 바꾸는 동안 InnoDB 데이터 사전이 업데이트되지 않습니다. (버그 #19465984)
- 테이블 이름을 다른 데이터베이스로 변경하면 FTS 총들이 발생합니다. (버그 #16834860)
- 잘린 테이블에서 트리거를 준비하지 못하면 오류 1054가 발생합니다. (버그 #18596756)
- 메타데이터를 변경하면 트리거 실행에 문제가 발생할 수 있습니다. (버그 #18684393)
- 길이가 긴 UTF8 VARCHAR 필드에 구조화가 선택되지 않습니다. (버그 #17566396)
- 제한 사항 X가 있는 ORDER BY 실행 시 실행 계획의 성능 저하. (버그 #16697792)
- 백포트(Backport) 버그#11765744 - 5.1, 5.5 및 5.6. (버그 #17083851)
- SQL/SQL_SHOW.CC에서의 뮤텍스 문제로 SIG6가 발생합니다. 소스는 FILL_VARIABLES와 유사합니다. (버그 #20788853)
- 백포트(Backport) 버그 #18008907 - 5.5 이상 버전 (버그 #18903155)
- MySQL 5.7에서 스택 오버플로우 오류에 대해 수정 적용 (버그 #19678930)

Aurora MySQL 데이터베이스 엔진 업데이트: 2015-10-16

버전: 1.2, 1.3

이 업데이트에는 다음의 기능 향상이 포함되어 있습니다.

수정 사항

- 오랜 시간 실행되는 트랜잭션이 있는 새로운 잠금 관리자에서 메모리 부족 문제 해결
- 비-RDS MySQL 데이터베이스를 복제할 때 발생하는 보안 취약성 해결

- 쿼리를 쓰기와 스토리지 오류와 함께 올바르게 재시도되도록 업데이트됨
- 복제 지연을 보다 정확하게 보고하도록 업데이트됨
- 많은 수의 동시 트랜잭션이 동일한 행을 수정하려고 하는 경우 경합을 줄임으로써 성능 개선
- 두 개의 테이블을 조인함으로써 생성된 뷰의 쿼리 캐시 무효화 해결
- UNCOMMITTED_READ가 격리된 트랜잭션의 쿼리 캐시 비활성화

개선 사항

- 워크 캐시에 있는 느린 카탈로그 쿼리의 성능 개선
- 사전 통계의 동시성 개선
- 새로운 쿼리 캐시 리소스 관리자, 익스텐트 관리, Amazon Aurora 스마트 스토리지에 저장되어 있는 파일, 로그 레코드의 배치 쓰기에 대한 안정성 개선

MySQL 버그 수정 통합

- innodb 내의 쿼리 삭제 시 결과적으로 어설션과의 충돌 발생 (버그 #1608883)
- 이벤트 스케줄러, 이벤트 실행 또는 새로운 연결에 대해 새로운 스레드를 생성하지 못한 경우 오류 로그에 아무 메시지도 작성되지 않습니다. (버그 #16865959)
- 한 연결이 기본 데이터베이스를 변경하고 동시에 다른 연결이 SHOW PROCESSLIST를 수행하는 경우, 첫 번째 연결의 기본 데이터베이스 메모리를 표시하려고 할 때 두 번째 연결에서 잘못된 메모리에 액세스 할 수 있습니다. (버그 #11765252)
- 설계 상 PURGE BINARY LOGS는 사용 중이거나 활성화 상태의 이진 로그 파일을 제거하지 않지만, 이러한 경우에도 아무런 메시지가 표시되지 않습니다. (버그 #13727933)
- 일부 문에서는 최적화 프로그램이 필요 없는 하위 쿼리 절을 제거하는 경우 메모리 누수가 발생할 수 있습니다. (버그 #15875919)
- 종료 중에 서버가 초기화되지 않은 뮤텍스를 잠그려고 시도할 수 있습니다. (버그 #16016493)
- GROUP_CONCAT() 및 ORDER BY 절을 사용하여 여러 열의 이름을 지정하는 준비된 문을 사용하면 서버가 종료될 수 있습니다. (버그 #16075310)
- 슬레이브 워커 스레드에 성능 스키마 장비가 누락되었습니다. (버그 #16083949)
- STOP SLAVE가 하나 이상의 상태 변수 Slave_retried_transactions, Slave_heartbeat_period, Slave_received_heartbeats, Slave_last_heartbeat 또는 Slave_running의 값을 검색하는 SHOW STATUS 와 같은 문과 함께 동시에 실행되는 경우 교착 상태가 발생할 수 있습니다. (버그 #16088188)
- 검색 용어에 따옴표가 있는 경우 부울 모드를 사용한 전체 텍스트 쿼리에서 0이 반환될 수 있습니다. (버그 #16206253)
- 하위 쿼리에서 조인의 ON 절에 하위 쿼리가 있는 준비된 문을 실행하는 경우 최적화 프로그램이 종복되는 하위 쿼리 절을 제거하려고 시도하면 어설션이 발생합니다. (버그 #16318585)
- GROUP_CONCAT 불안정성, ITEM_SUM::CLEAN_UP_AFTER_REMOVAL에서 충돌 발생 (버그 #16347450)
- INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD와 동일한 구조의 InnoDB 테이블을 생성하여 기본 InnoDB 전체 텍스트 검색(FTS) 불용어 목록을 변경하려고 시도하면 오류가 발생합니다. (버그 #16373868)
- 슬레이브의 클라이언트 스레드에서 FLUSH TABLES WITH READ LOCK을 실행한 후 마스터에서 일부 업데이트를 실행하면 SHOW SLAVE STATUS를 실행할 때 슬레이브가 멈춥니다. (버그 #16387720)
- 전체 텍스트 검색에서 "abc-def"와 같은 구분된 검색 문자열을 파싱할 때 이제 InnoDB에서 MyISAM과 동일한 단어 구분자를 사용합니다. (버그 #16419661)
- FTS_AST_TERM_SET_WILDCARD에서 충돌이 발생합니다. (버그 #16429306)
- FTS RQG 테스트 시 FTS_AST_VISIT()에서 SEGFAULT가 발생합니다. (버그 #16435855)
- 빌드 디버그 시 최적화 프로그램이 하위 쿼리를 가리키는 Item_ref를 제거하면 서버가 종료됩니다. (버그 #16509874)

- InnoDB 테이블에서 전체 텍스트 검색으로 + 또는 - 연산자와 결합된 리터럴 문구를 검색하면 오류가 발생합니다. (버그 #16516193)
- 서버가 --master-info-repository=TABLE relay-log-info-repository=TABLE 옵션을 사용하고 자동 커밋을 0 으로 설정하고 --skip-slave-start를 함께 사용하여 서버를 시작하면 START SLAVE에서 오류가 발생합니다. (버그 #16533802)
- InnoDB 전체 텍스트 검색(FTS)의 크기가 매우 크면 과도한 메모리 양을 소비할 수 있습니다. (버그 #16625973)
- 검색 문자열에 이진을 직접 사용하면 이진에 NULL 바이트 및 다른 의미 없는 문자가 포함될 수 있으므로 빌드 디버그 시 OPT_CHECK_ORDER_BY에서 어설션이 발생할 수 있습니다. (버그 #16766016)
- 일부 문에서는 최적화 프로그램이 필요 없는 하위 쿼리 절을 제거하는 경우 메모리 누수가 발생할 수 있습니다. (버그 #16807641)
- 슬레이브에 대한 새로운 연결에서 STOP SLAVE를 실행한 다음, 원래 연결을 사용하여 SHOW SLAVE STATUS를 실행하여 FLUSH TABLES WITH READ LOCK을 실행하면 교착 상태가 발생할 가능성이 있습니다. (버그 #16856735)
- 잘못된 구분 기호와 함께 GROUP_CONCAT()를 실행하면 서버가 종료될 수 있습니다. (버그 #16870783)
- 패턴이 해당 뮤텍스(Slave_heartbeat_period, Slave_last_heartbeat, Slave_received_heartbeats, Slave_retried_transactions, Slave_running)를 사용하는 상태 변수와 일치하지 않는 경우에도 서버가 SHOW STATUS LIKE '패턴' 문에 대해 LOCK_active_mi 및 active_mi->rli->data_lock 뮤텍스에서 과도한 잠금을 설정했습니다. (버그 #16904035)
- IN BOOLEAN MODE 수정자를 사용하여 전체 텍스트 검색을 실행하면 어설션이 발생합니다. (버그 #16927092)
- InnoDB 테이블에서 전체 텍스트 검색으로 + 부울 연산자를 사용하여 검색하면 오류가 발생합니다. (버그 #17280122)
- 4웨이 교착 상태: 좀비, binlog 삭제, 프로세스 목록 표시, binlog 표시 (버그 #17283409)
- 커밋 잠금을 기다리고 있는 SQL 스레드가 중단되었다가 다시 시작되는 경우 트랜잭션이 슬레이브에서 건너 뛰게 됩니다. (버그 #17450876)
- "종료되지 않는" 토큰으로 인해 InnoDB 전체 텍스트 검색 오류가 발생합니다. 문자열 및 문자열 길이가 문자열 비교를 위해 전달되어야 합니다. (버그 #17659310)
- 많은 수의 분할된 InnoDB 테이블이 MySQL 5.6 또는 5.7에서 사용될 때 이전 릴리스의 MySQL Server에서 사용될 때보다 동일한 테이블에서 사용하는 메모리 양이 많이 증가될 수 있습니다. (버그 #17780517)
- 전체 텍스트 쿼리의 경우 num_token이 max_proximity_item보다 적은지 확인하지 못하여 어설션이 발생할 수 있습니다. (버그 #18233051)
- 비어 있는 InnoDB 테이블이 많이 있는 경우 INFORMATION_SCHEMA 테이블 및 COLUMNS 테이블에 대한 특정 쿼리가 과도한 메모리 사용을 초래할 수 있습니다. (버그 #18592390)
- 트랜잭션을 커밋할 때 이제 플래그가 사용되어 스레드 자체를 확인하지 않고 스레드 생성 여부를 확인합니다. 이에 따라 특히 master_info_repository=TABLE과 함께 서버를 실행할 때 더 많은 리소스가 사용됩니다. (버그 #18684222)
- 마스터가 DML을 실행하는 동안 슬레이브에 있는 클라이언트 스레드가 FLUSH TABLES WITH READ LOCK을 실행하는 경우 동일한 클라이언트에 있는 SHOW SLAVE STATUS를 실행하면 차단되어 교착 상태가 발생합니다. (버그 #19843808)
- GROUP_CONCAT()로 주문하면 서버가 종료될 수 있습니다. (버그 #19880368)

Aurora MySQL 데이터베이스 엔진 업데이트: 2015-08-24

버전: 1.1

이 업데이트에는 다음의 기능 향상이 포함되어 있습니다.

- MySQL 데이터베이스(binlog 복제)를 사용하여 복제하는 경우 복제 안정성이 개선되었습니다. MySQL을 사용한 Aurora MySQL 복제에 대한 자세한 내용은 [Amazon Aurora를 사용한 복제 \(p. 55\)](#) 단원을 참조하십시오.

- 릴레이 로그의 크기에 대한 1기가바이트(GB) 제한이 복제 슬레이브인 Aurora MySQL DB 클러스터에 대해 수집됩니다. 이에 따라 Aurora DB 클러스터에 대한 파일 관리가 개선되었습니다.
- 미리읽기, 리커시브 외래 키 관계 및 Aurora 복제의 영역에서 안정성이 개선되었습니다.
- MySQL 버그 수정 통합.
 - 이름이 숫자로 시작하는 InnoDB 데이터베이스가 전체 텍스트 검색(FTS) 구문 분석기 오류를 일으킵니다. (버그 #17607956)
 - 이름이 숫자로 시작하는 데이터베이스에서 InnoDB 전체 텍스트 검색이 실패합니다. (버그 #17161372)
 - Windows에서 실행되는 InnoDB 데이터베이스의 경우, 전체 텍스트 검색(FTS) 객체 ID가 예상되는 16진수 형식이 아닙니다. (버그 #16559254)
 - MySQL 5.6에 도입된 코드 회귀가 DROP TABLE 및 ALTER TABLE 성능에 악영향을 미쳤습니다. 이로 인해 MySQL Server 5.5.x와 5.6.x 사이에서 성능 저하가 발생했을 수 있습니다. (버그 #16864741)
- 로깅을 단순화하여 필요한 스토리지 양 및 로그 파일 크기가 줄어들었습니다.

Aurora MySQL 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그

다음 섹션은 Aurora MySQL 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그를 설명한 것입니다.

주제

- [Aurora MySQL 2.x 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그 \(p. 763\)](#)
- [Aurora MySQL 1.x 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그 \(p. 764\)](#)

Aurora MySQL 2.x 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그

MySQL 5.7과 호환되는 버전의 Aurora에는 MySQL 5.7.12를 통한 모든 MySQL 버그 수정이 포함되어 있습니다. 다음 표에는 Aurora MySQL 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그와 이 버그들이 어느 업데이트에서 수정되었는지가 나와 있습니다.

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 2.07.0) (p. 691)	2.07.0	<ul style="list-style-type: none">버그 #26251621: 트리거 및 GCOL로 인한 잘못된 동작버그 #22574695: ASSERTION '!TABLE (!TABLE->READ_SET BITMAP_IS_SET(TABLE->READ_SET, FIELD))' 실패버그 #25966845: 복제 키의 INSERT로 인해 교착 상태 발생버그 #23070734: 동시 TRUNCATE 테이블로 인해 종단 발생버그 #26191879: 외래 키 CASCADE에서 과도한 메모리 사용버그 #20989615: INNODB AUTO_INCREMENT에서 동일한 값을 두 번 산출
Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-11(버전 2.05.0) (p. 696)	2.05.0	<ul style="list-style-type: none">Bug#23054591: PURGE BINARY LOGS TO가 전체 binlog 파일을 읽고 있어 MySQL이 지연되고 있습니다.
Aurora MySQL 데이터베이스 엔진 업데이트 2019-09-19(버전 2.04.6) (p. 701)	2.04.6	<ul style="list-style-type: none">Bug#23054591: PURGE BINARY LOGS TO가 전체 binlog 파일을 읽고 있어 MySQL이 지연되고 있습니다.

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-11 (p. 712)	2.03	<ul style="list-style-type: none"> REVERSE SCAN ON A PARTITIONED TABLE DOES ICP - ORDER BY DESC(버그 #24929748). JSON_OBJECT CREATES INVALID JSON CODE(버그 #26867509). INSERTING LARGE JSON DATA TAKES AN INORDINATE AMOUNT OF TIME(버그 #22843444). PARTITIONED TABLES USE MORE MEMORY IN 5.7 THAN 5.6(버그 #25080442).
Aurora MySQL 데이터베이스 엔진 업데이트 2018-05-03 (p. 717)	2.02	좌측 조인이 외부 측에 잘못된 결과를 반환합니다(버그 #22833364)
Aurora MySQL 데이터베이스 엔진 업데이트 2019-05-02(버전 2.04.2) (p. 706)	2.04.2	버그 #24829050 - INDEX_MERGE_INTERSECTION 최적화로 인해 잘못된 쿼리 결과 산출

Aurora MySQL 1.x 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그

MySQL 5.6과 호환되는 버전의 Aurora에는 MySQL 5.6.10을 통한 모든 MySQL 버그 수정이 포함되어 있습니다. 다음 표에는 Aurora MySQL 데이터베이스 엔진 업데이트를 통해 수정한 MySQL 버그와 이 버그들이 어느 업데이트에서 수정되었는지가 나와 있습니다.

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
Aurora MySQL 데이터베이스 엔진 업데이트: 2015-08-24 (p. 762)	1.1	<ul style="list-style-type: none"> 이름이 숫자로 시작하는 InnoDB 데이터베이스가 전체 텍스트 검색(FTS) 구문 분석기 오류를 일으킵니다. (버그 #17607956) 이름이 숫자로 시작하는 데이터베이스에서 InnoDB 전체 텍스트 검색이 실패합니다. (버그 #17161372) Windows에서 실행되는 InnoDB 데이터베이스의 경우, 전체 텍스트 검색(FTS) 객체 ID가 예상되는 16진수 형식이 아닙니다. (버그 #16559254) MySQL 5.6에 도입된 코드 회귀가 DROP TABLE 및 ALTER TABLE 성능에 악영향을 미쳤습니다. 이로 인해 MySQL Server 5.5.x와 5.6.x 사이에서 성능 저하가 발생했을 수 있습니다. (버그 #16864741)
Aurora MySQL 데이터베이스 엔진 업데이트: 2015-10-16 (p. 760)	1.2, 1.3	<ul style="list-style-type: none"> innodb 내의 쿼리 삭제 시 결과적으로 어설션과의 충돌 발생 (버그 #1608883) 이벤트 스케줄러, 이벤트 실행 또는 새로운 연결에 대해 새로운 스레드를 생성하지 못한 경우 오류 로그에 아무 메시지도 작성되지 않습니다. (버그 #16865959) 한 연결이 기본 데이터베이스를 변경하고 동시에 다른 연결이 SHOW PROCESSLIST를 수행하는 경우, 첫 번째 연결의 기본 데이터베이스 메모리를 표시하려고 할 때 두 번째 연결에서 잘못된 메모리에 액세스할 수 있습니다. (버그 #11765252)

데이터베이스 엔진 업 데이트	버전	수정된 MySQL 버그
		<ul style="list-style-type: none"> 설계 상 PURGE BINARY LOGS는 사용 중이거나 활성화 상태의 이진 로그 파일을 제거하지 않지만, 이러한 경우에도 아무런 메시지가 표시되지 않습니다. (버그 #13727933) 일부 문에서는 최적화 프로그램이 필요 없는 하위 쿼리 절을 제거하는 경우 메모리 누수가 발생할 수 있습니다. (버그 #15875919) 종료 중에 서버가 초기화되지 않은 뮤텍스를 잠그려고 시도할 수 있습니다. (버그 #16016493) GROUP_CONCAT() 및 ORDER BY 절을 사용하여 여러 열의 이름을 지정하는 준비된 문을 사용하면 서버가 종료될 수 있습니다. (버그 #16075310) 슬레이브 워커 스레드에 성능 스키마 장비가 누락되었습니다. (버그 #16083949) STOP SLAVE가 하나 이상의 상태 변수 Slave_retried_transactions, Slave_heartbeat_period, Slave_received_heartbeats, Slave_last_heartbeat 또는 Slave_running의 값을 검색하는 SHOW STATUS와 같은 문과 함께 동시에 실행되는 경우 교착 상태가 발생할 수 있습니다. (버그 #16088188) 검색 용어에 따옴표가 있는 경우 부울 모드를 사용한 전체 텍스트 쿼리에서 0이 반환될 수 있습니다. (버그 #16206253) 하위 쿼리에서 조인의 ON 절에 하위 쿼리가 있는 준비된 문을 실행하는 경우 최적화 프로그램이 중복되는 하위 쿼리 절을 제거하려고 시도하면 어설션이 발생합니다. (버그 #16318585) GROUP_CONCAT 불안정성, ITEM_SUM::CLEAN_UP_AFTER_REMOVAL에서 충돌 발생 (버그 #16347450) INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD 와 동일한 구조의 InnoDB 테이블을 생성하여 기본 InnoDB 전체 텍스트 검색(FTS) 불용어 목록을 변경하려고 시도하면 오류가 발생합니다. (버그 #16373868) 슬레이브의 클라이언트 스레드에서 FLUSH TABLES WITH READ LOCK을 실행한 후 마스터에서 일부 업데이트를 실행하면 SHOW SLAVE STATUS를 실행할 때 슬레이브가 멈춥니다. (버그 #16387720) 전체 텍스트 검색에서 "abc-def"와 같은 구분된 검색 문자열을 파싱할 때 이제 InnoDB에서 MyISAM과 동일한 단어 구분자를 사용합니다. (버그 #16419661) FTS_AST_TERM_SET_WILDCARD에서 충돌이 발생합니다. (버그 #16429306) FTS RQG 테스트 시 FTS_AST_VISIT()에서 SEGFAULT가 발생합니다. (버그 #16435855) 빌드 디버그 시 최적화 프로그램이 하위 쿼리를 가리키는 Item_ref 를 제거하면 서버가 종료됩니다. (버그 #16509874) InnoDB 테이블에서 전체 텍스트 검색으로 + 또는 - 연산자와 결합된 리터럴 문구를 검색하면 오류가 발생합니다. (버그 #16516193) 서버가 --master-info-repository=TABLE relay-log-info-repository=TABLE 옵션을 사용하고 자동 커밋을 0으로 설정하고 --skip-slave-start를 함께 사용하여 서버를 시작하면 START SLAVE에서 오류가 발생합니다. (버그 #16533802) InnoDB 전체 텍스트 검색(FTS)의 크기가 매우 크면 과도한 메모리 양을 소비할 수 있습니다. (버그 #16625973)

데이터베이스 엔진 업 데이트	버전	수정된 MySQL 버그
		<ul style="list-style-type: none"> • 검색 문자열에 이진을 직접 사용하면 이진에 NULL 바이트 및 다른 의미 없는 문자가 포함될 수 있으므로 빌드 디버그 시 OPT_CHECK_ORDER_BY에서 어설션이 발생할 수 있습니다. (버그 #16766016) • 일부 문에서는 최적화 프로그램이 필요 없는 하위 쿼리 절을 제거하는 경우 메모리 누수가 발생할 수 있습니다. (버그 #16807641) • 슬레이브에 대한 새로운 연결에서 STOP SLAVE를 실행한 다음, 원래 연결을 사용하여 SHOW SLAVE STATUS를 실행하여 FLUSH TABLES WITH READ LOCK을 실행하면 교착 상태가 발생할 가능성이 있습니다. (버그 #16856735) • 잘못된 구분 기호와 함께 GROUP_CONCAT()를 실행하면 서버가 종료될 수 있습니다. (버그 #16870783) • 패턴이 해당 뮤텍스(Slave_heartbeat_period, Slave_last_heartbeat, Slave_received_heartbeats, Slave_retried_transactions, Slave_running)를 사용하는 상태 변수와 일치하지 않는 경우에도 서버가 SHOW STATUS LIKE '패턴' 문에 대해 LOCK_active_mi 및 active_mi->rli->data_lock 뮤텍스에서 과도한 잠금을 설정했습니다. (버그 #16904035) • IN BOOLEAN MODE 수정자를 사용하여 전체 텍스트 검색을 실행하면 어설션 오류가 발생합니다. (버그 #16927092) • InnoDB 테이블에서 전체 텍스트 검색으로 + 부울 연산자를 사용하여 검색하면 오류가 발생합니다. (버그 #17280122) • 4웨이 교착 상태: 좀비, binlog 삭제, 프로세스 목록 표시, binlog 표시 (버그 #17283409) • 커밋 잠금을 기다리고 있는 SQL 스레드가 중단되었다가 다시 시작되는 경우 트랜잭션이 슬레이브에서 건너 뛰게 됩니다. (버그 #17450876) • "종료되지 않는" 토큰으로 인해 InnoDB 전체 텍스트 검색 오류가 발생합니다. 문자열 및 문자열 길이가 문자열 비교를 위해 전달되어야 합니다. (버그 #17659310) • 많은 수의 분할된 InnoDB 테이블이 MySQL 5.6 또는 5.7에서 사용될 때 이전 릴리스의 MySQL Server에서 사용될 때보다 동일한 테이블에서 사용하는 메모리 양이 많이 증가될 수 있습니다. (버그 #17780517) • 전체 텍스트 쿼리의 경우 num_token이 max_proximity_item보다 적은지 확인하지 못하여 어설션이 발생할 수 있습니다. (버그 #18233051) • 비어 있는 InnoDB 테이블이 많이 있는 경우 INFORMATION_SCHEMA 테이블 및 COLUMNS 테이블에 대한 특정 쿼리가 과도한 메모리 사용을 초래할 수 있습니다. (버그 #18592390) • 트랜잭션을 커밋할 때 이제 플래그가 사용되어 스레드 자체를 확인하지 않고 스레드 생성 여부를 확인합니다. 이에 따라 특히 master_info_repository=TABLE과 함께 서버를 실행할 때 더 많은 리소스가 사용됩니다. (버그 #18684222) • 마스터가 DML을 실행하는 동안 슬레이브에 있는 클라이언트 스레드가 FLUSH TABLES WITH READ LOCK을 실행하는 경우 동일한 클라이언트에 있는 SHOW SLAVE STATUS를 실행하면 차단되어 교착 상태가 발생합니다. (버그 #19843808)

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
		<ul style="list-style-type: none"> GROUP_CONCAT()로 주문하면 서버가 종료될 수 있습니다. (버그 #19880368)
Aurora MySQL 데이터베이스 엔진 업데이트: 2015-12-03 (p. 759)	1.4	<ul style="list-style-type: none"> FTSPARSE()의 SEGV 오류 (버그 #16446108) 열의 이름을 바꾸는 동안 InnoDB 데이터 사전이 업데이트되지 않습니다. (버그 #19465984) 테이블 이름을 다른 데이터베이스로 변경하면 FTS 충돌이 발생합니다. (버그 #16834860) 잘린 테이블에서 트리거를 준비하지 못하면 오류 1054가 발생합니다. (버그 #18596756) 메타데이터를 변경하면 트리거 실행에 문제가 발생할 수 있습니다. (버그 #18684393) 길이가 긴 UTF8 VARCHAR 필드에 구제화가 선택되지 않습니다. (버그 #17566396) 제한 사항 X가 있는 ORDER BY 실행 시 실행 계획의 성능 저하. (버그 #16697792) 백포트(Backport) 버그#11765744 - 5.1, 5.5 및 5.6. (버그 #17083851) SQL/SQL_SHOW.CC에서의 뮤텍스 문제로 SIG6가 발생합니다. 소스는 FILL_VARIABLES와 유사합니다. (버그 #20788853) 백포트(Backport) 버그 #18008907 - 5.5 이상 버전 (버그 #18903155) MySQL 5.7에서 스택 오버플로우 오류에 대해 수정 적용 (버그 #19678930)
Aurora MySQL 데이터베이스 엔진 업데이트: 2016-01-11 (p. 759)	1.5	<ul style="list-style-type: none"> 데이터베이스 이름이 숫자로 시작하는 테이블에 영향을 주는 MySQL 전체 텍스트 검색 기능이 완전히 해결되지 않았던 문제를 해결했습니다. (포트 버그 #17607956)
Aurora MySQL 데이터베이스 엔진 업데이트: 2016-04-06 (p. 757)	1.6	<ul style="list-style-type: none"> 백포트 버그 #18694052 5.6에서 'IM_ORDERED_REC_BUFFER' 어설션 실패 수정(포트 버그 #18305270) MEMCPY(), HA_PARTITION::POSITION의 SEGV 오류(포트 버그 # 18383840) 파티셔닝, INDEX_MERGE 및 NO PK에서 잘못된 결과 도출(포트 버그 # 18167648) FLUSH TABLES FOR EXPORT: HA_PARTITION::EXTRA의 어설션 오류(포트 버그 # 16943907) 가상 HA_ROWS HANDLER::MULTI_RANGE_READ_INFO_CONST에서 서버 충돌 (포트 버그 # 16164031) SEL_ARG::RB_INSERT()에서 범위 최적화 프로그램 충돌(포트 버그 # 16241773)
Aurora MySQL 데이터베이스 엔진 업데이트: 2016-06-01 (p. 757)	1.6.5	<ul style="list-style-type: none"> 마스터 충돌 복구 후 슬레이브에서 복제를 진행할 수 없음(포트 버그 #17632285)

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
Aurora MySQL 데이터베이스 엔진 업데이트: 2016-08-30 (p. 756)	1.7	<ul style="list-style-type: none"> LOCK_grant 잠금을 분할하여 확장성을 향상했습니다. (포트 WL #8355) 저장 프로시저의 SELECT에 커서를 두면 segfault가 발생합니다. (포트 버그 #16499751) MySQL에서는 일부 특수한 경우에 잘못된 결과를 제공합니다. (버그 #11751794) #11751794 버그에 대한 패치에 의해 GET_SEL_ARG_FOR_KEYPART에서- 충돌이 발생합니다. (버그 #16208709) GROUP BY를 통한 간단한 쿼리에 대해 잘못된 결과가 표시됩니다. (버그 #17909656) 범위 조건자를 통한 semijoin 쿼리에서 추가 행이 표시됩니다. (버그 #16221623) IN 하위 쿼리 뒤에 ORDER BY 절을 추가하면 중복 행이 반환될 수 있습니다. (버그 #16308085) 쿼리에 대한 설명이 GROUP BY, MyISAM에 대한 간략 스캔과 충돌 합니다. (버그 #16222245) 인용된 int 조건자를 사용하여 느슨한 인덱스 스캔을 수행하면 임의의 데이터가 반환됩니다. (버그 #16394084) 최적화 프로그램에서 느슨한 인덱스 스캔을 사용 중인 경우 임시 테이블을 생성하려고 하면 서버가 종료될 수 있습니다. (버그 #16436567) COUNT(DISTINCT)는 NULL 값을 계산하지 않지만 최적화 프로그램에서 느슨한 인덱스 스캔을 사용하는 경우에는 계산됩니다. (버그 #17222452) 쿼리에 MIN() MAX() 및 aggregate_function(DISTINCT)이 모두 포함되어 있고(예: SUM(DISTINCT)) 느슨한 인덱스 스캔을 사용하여 쿼리를 실행한 경우 MIN() MAX()의 결과 값이 잘못 설정되었습니다. (버그 #17217128)
Aurora MySQL 데이터베이스 엔진 업데이트: 2016-10-18 (p. 754)	1.8	<ul style="list-style-type: none"> 여러 인덱스를 포함하는 하나의 열에 모든 인덱스를 끌어서 놓은 경우 외래 키 제약 조건에서 인덱스를 요구할 때 InnoDB에서 DROP INDEX 작업을 차단하지 못합니다. (버그 #16896810) 외래 키 제약 조건 추가 충돌을 해결합니다. (버그 #16413976) 저장 프로시저에서 커서를 가져오는 동시에 테이블을 분석하거나 플러시할 때 발생하는 충돌을 해결했습니다. (버그 #18158639) 사용자가 테이블에서 AUTO_INCREMENT 값을 최대 자동 증분 열 값보다 작게 변경할 때 발생하는 자동 증분 버그를 해결했습니다. (버그 #16310273)
Aurora MySQL 데이터베이스 엔진 업데이트: 2016-10-26 (p. 754)	1.8.1	<ul style="list-style-type: none"> LogJam 문제 때문에 OpenSSL이 Diffie-Hellman 키 길이 파라미터를 변경했습니다. (버그 #18367167)

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
Aurora MySQL 데이터베이스 엔진 업데이트: 2016-12-14 (p. 752)	1.10	<ul style="list-style-type: none"> 파생된 테이블의 UNION이 '1=0/false' 절이 있는 잘못된 결과를 반환합니다. (버그 #69471) 저장된 프로시저 2차 실행 시 ITEM_FUNC_GROUP_CONCAT::FIX_FIELDS에서 서버가 충돌합니다. (버그 #20755389) 캐시 크기가 전체 크기의 10%를 넘어서자마자 별도 스레드에 캐시 동기화 작업을 오프로드하여 FTS 캐시 동기화 중에 MySQL 쿼리가 너무 오래 종지되는 일을 방지합니다. (버그 #22516559, #73816)
Aurora MySQL 데이터베이스 엔진 업데이트: 2017-02-23 (p. 750)	1.11	<ul style="list-style-type: none"> ALTER 테이블 DROP 외래 키를 다른 DROP 연산과 동시에 실행하면 테이블이 사라집니다. (버그 #16095573) ORDER BY를 사용한 일부 INFORMATION_SCHEMA 쿼리가 예전처럼 파일 정렬 최적화를 사용하지 않습니다. (버그 #16423536) FOUND_ROWS ()가 잘못된 테이블 행 수를 반환합니다 (버그 #68458) 임시 테이블을 너무 많이 열면 오류가 발생하는 대신 서버가 고장납니다. (버그 #18948649)
Aurora MySQL 데이터베이스 엔진 업데이트: 2017-04-05 (p. 749)	1.12	<ul style="list-style-type: none"> 비어 있는 상태로 인해 AUTO_INCREMENT 값이 재설정되는 동안 제거된 테이블을 재로드합니다. (버그 #21454472, 버그 #77743) purge_node_t 구조의 불일치로 인해 인덱스 기록이 룰백에서 발견되지 않았습니다. 이러한 불일치로 "보조 인덱스 입력 업데이트 오류", "기록을 제거할 수 없음", "삭제 표시되지 않은 보조 인덱스 입력 제거를 시도함" 등의 경고 및 오류 메시지가 표시되었습니다. (버그 #19138298, 버그 #70214, 버그 #21126772, 버그 #21065746) qsort 작업에 대한 스택 크기 계산을 잘못하면 스택 오버플로우가 발생합니다. (버그 #73979) 루백 시 인덱스에서 기록이 발견되지 않았습니다. (버그 #70214, 버그 #72419) 업데이트 CURRENT_TIMESTAMP의 ALTER TABLE 추가 열 TIMESTAMP가 ZERO-datas 삽입(버그 #17392)
Aurora MySQL 데이터베이스 엔진 업데이트: 2017-05-15 (p. 747)	1.13	<ul style="list-style-type: none"> 비어 있는 상태로 인해 AUTO_INCREMENT 값이 재설정되는 동안 제거된 테이블을 재로드합니다. (버그 #21454472, 버그 #77743) purge_node_t 구조의 불일치로 인해 인덱스 기록이 룰백에서 발견되지 않았습니다. 이러한 불일치로 "보조 인덱스 입력 업데이트 오류", "기록을 제거할 수 없음", "삭제 표시되지 않은 보조 인덱스 입력 제거를 시도함" 등의 경고 및 오류 메시지가 표시되었습니다. (버그 #19138298, 버그 #70214, 버그 #21126772, 버그 #21065746) qsort 작업에 대한 스택 크기 계산을 잘못하면 스택 오버플로우가 발생합니다. (버그 #73979) 루백 시 인덱스에서 기록이 발견되지 않았습니다. (버그 #70214, 버그 #72419) 업데이트 CURRENT_TIMESTAMP의 ALTER TABLE 추가 열 TIMESTAMP가 ZERO-datas 삽입(버그 #17392)
Aurora MySQL 데이터베이스 엔진 업데이트: 2017-08-07 (p. 746)	1.14	이전에는 전체 텍스트 검색이 파생 테이블(FROM 절의 하위 쿼리)과 결합되면서 서버 종료의 원인이 되었습니다. 하지만 이제는 전체 텍스트 작업이 파생 테이블에 따라 결정되는 경우 구체화된 테이블에서 전체 텍스트 검색을 실행할 수 없다는 오류가 서버에서 발생합니다. (버그 #68751, 버그 #16539903)

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
Aurora MySQL 데이터베이스 엔진 업데이트: 2018-03-13 (p. 745)	1.14.4	<ul style="list-style-type: none"> 무시할 수 있는 이벤트는 유효하지 않기 때문에 테스트 대상이 아닙니다(버그 #74683). NEW->OLD ASSERT FAILURE 'GTID_MODE > 0'(버그 #20436436)
Aurora MySQL 데이터베이스 엔진 업데이트 2017-10-24 (p. 743)	1.15	<ul style="list-style-type: none"> CREATE USER가 플러그인 및 암호 해시를 허용하지만 암호 해시는 무시합니다(버그 #78033). 파티션 분할 엔진은 분할된 인덱스에서 정렬된 항목을 반환할 수 있도록 여러 필드를 읽기 비트 집합에 추가합니다. 이는 불필요한 필드 까지 읽으려고 하면서 조인 버퍼의 원인이 됩니다. 분할 필드를 모두 read_set에 추가하지 않는 대신 read_set에서 이미 설정된 접두사 필드를 기준으로 정렬하여 버퍼 문제를 수정하였습니다. key_cmp 를 실행하는 경우 첫 번째 필드를 읽어야 하도록 DBUG_ASSERT가 추가되었습니다(버그 #16367691). MySQL 인스턴스에서 "SYNC 인덱스 실행"이 지연됩니다(버그 #73816). RBT_EMPTY(INDEX_CACHE->WORDS)를 ALTER TABLE 변경 열에 어설션합니다(버그 #17536995). 저장점이 연관되었을 경우 InnoDB Fulltext 검색으로 레코드를 찾지 못합니다(버그 #70333).
Aurora MySQL 데이터베이스 엔진 업데이트 2017-11-20 (p. 743)	1.15.1	<ul style="list-style-type: none"> 되돌림 — MySQL 인스턴스에서 "SYNC 인덱스 실행"이 지연됩니다(버그 #73816). 되돌림 — RBT_EMPTY(INDEX_CACHE->WORDS)를 ALTER TABLE 변경 열에 어설션합니다(버그 #17536995). 되돌림 — 저장점이 연관되었을 경우 InnoDB Fulltext 검색으로 레코드를 찾지 못합니다(버그 #70333).
Aurora MySQL 데이터베이스 엔진 업데이트 2018-03-13 (p. 741)	1.17	<ul style="list-style-type: none"> 복제 필터가 사용되는 경우 LAST_INSERT_ID가 올바르지 않게 복제됩니다(버그 #69861). 쿼리가 INDEX_MERGE 설정에 따라 다른 결과를 반환합니다(버그 #16862316). 저장된 루틴의 쿼리 처리 재실행, 비효율적인 쿼리 계획(버그 #16346367) InnoDB FTS: FTS_CACHE_APPEND_DELETED_DOC_IDS에 어설션합니다(버그 #18079671). RBT_EMPTY(INDEX_CACHE->WORDS)를 ALTER TABLE 변경 열에 어설션합니다(버그 #17536995). 저장점이 연관되었을 경우 InnoDB Fulltext 검색으로 레코드를 찾지 못합니다(버그 #70333, 버그 #17458835).
Aurora MySQL 데이터베이스 엔진 업데이트 2018-09-06 (p. 737)	1.17.6	<ul style="list-style-type: none"> BINARY 열의 이름을 변경하거나 기본값을 변경한 ALTER TABLE 문의 경우, 인플레이스가 아닌 테이블 복사를 사용하여 변경이 수행되었습니다. (버그 #67141, 버그 #14735373, 버그 #69580, 버그 #17024290) 그룹인 정규 테이블과 파생 테이블 간의 외부 조인으로 인해 서버 종료가 발생할 수 있습니다. (버그 #16177639)

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
Aurora MySQL 데이터베이스 엔진 업데이트 2018-10-08 (p. 737)	1.17.7	<ul style="list-style-type: none"> 누락 테이블을 초래하는 외래 키 열에서 인덱스 삭제. (버그 #16208542) add_derived_key()의 메모리 누수. (버그 #76349) 파티션 분할된 테이블의 경우, 인덱스 병합 사용 여부에 따라 쿼리가 다른 결과를 반환할 수 있습니다. (버그 #16862316) 인덱스 병합 최적화(인덱스 병합 최적화 참조)를 사용하는 쿼리를 HASH로 파티션 분할된 테이블에서 실행할 경우 잘못된 결과가 반환될 수 있습니다. (버그 #17588348)
Aurora MySQL 데이터베이스 엔진 업데이트 2019-01-17 (p. 736)	1.17.8	<ul style="list-style-type: none"> 버그 #13418638: 존재하지 않는 경우 테이블 생성, 메타데이터 잡금이 너무 제한적임
Aurora MySQL 데이터베이스 엔진 업데이트 2019-02-07(버전 1.19.0) (p. 733)	1.19.0	<ul style="list-style-type: none"> 버그 #32917: ORPHAN TEMP-POOL 파일을 감지하여 정상적으로 처리 버그 #63144: 존재하지 않는 경우 테이블 생성, 메타데이터 잡금이 너무 제한적임
Aurora MySQL 데이터베이스 엔진 업데이트 2019-09-19(버전 1.19.5) (p. 731)	1.19.5	<ul style="list-style-type: none"> CVE-2018-2696 CVE-2015-4737 Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: UNION 문에서 행 검토 값이 잘못 계산되었습니다. 이러한 오류는 성능 스키마 문 테이블(예: events_statements_current)의 ROWS_EXAMINED 열 값이 너무 클 때 명백하게 발생하였습니다. Bug #11827369: SELECT ... FROM DUAL 종합 하위 쿼리가 포함된 일부 쿼리로 인해 어설션이 발생하였습니다. Bug #16311231: WHERE 절에 XOR 작업이 포함된 IN 절에 하위 쿼리가 들어 있는 쿼리의 경우 잘못된 결과가 반환되었습니다.
Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-11(버전 1.20.0) (p. 729)	1.20.0	<ul style="list-style-type: none"> Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: UNION 문에서 행 검토 값이 잘못 계산되었습니다. 이러한 오류는 성능 스키마 문 테이블(예: events_statements_current)의 ROWS_EXAMINED 열 값이 너무 커지는 것으로 나타났습니다. Bug #11827369: SELECT ... FROM DUAL 종합 하위 쿼리가 포함된 일부 쿼리로 인해 어설션이 발생하였습니다. 버그 #16311231: WHERE 절에 XOR 작업이 포함된 IN 절에 하위 쿼리가 들어 있는 쿼리의 경우 잘못된 결과가 반환되었습니다.

데이터베이스 엔진 업데이트	버전	수정된 MySQL 버그
Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 1.21.0) (p. 727)	1.21.0	<ul style="list-style-type: none"> Bug #19929406: HANDLE_FATAL_SIGNAL (SIG=11) IN __MEMMOVE_SSSE3_BACK FROM STRING::COPY Bug #17059925: UNION 문에서 행 검토 값이 잘못 계산되었습니다. 이러한 오류는 성능 스키마 문 테이블(예: <code>events_statements_current</code>)의 <code>ROWS_EXAMINED</code> 열 값이 너무 커지는 것으로 나타났습니다. Bug #11827369: SELECT ... FROM DUAL 중첩 하위 쿼리가 포함된 일부 쿼리로 인해 어설션이 발생하였습니다. 버그 #16311231: WHERE 절에 XOR 작업이 포함된 IN 절에 하위 쿼리가 들어 있는 쿼리의 경우 잘못된 결과가 반환되었습니다.
Aurora MySQL 데이터베이스 엔진 업데이트 2019-11-25(버전 1.22.0) (p. 724)	1.22.0	<ul style="list-style-type: none"> 버그 #16346241 - ITEM_PARAM::QUERY_VAL_STR의 서버 충돌 버그 #17733850 - ITEM_NAME_CONST::ITEM_NAME_CONST()의 NAME_CONST() 충돌 버그 #20989615: INNODB AUTO_INCREMENT에서 동일한 값을 두 번 산출 버그 #20181776 - 액세스 제어에 와일드카드가 포함된 경우 가장 제한적인 호스트와 일치하지 않음 Bug #27326796 - MYSQL이 PARSONS.CC 파일의 INNODB ASSERTION 실패와 충돌 Bug #20590013 - FULLTEXT 인덱스가 있는데 이를 삭제하면 더 이상 온라인 DDL을 수행할 수 없음

Amazon Aurora PostgreSQL 작업

Amazon Aurora PostgreSQL은 완전 관리 형태로 PostgreSQL과 호환되고 ACID를 준수하는 관계형 데이터베이스 엔진으로서 고사양 상업용 데이터베이스의 속도 및 안정성이 오픈 소스 데이터베이스의 간편성 및 비용 효율성과 결합되었습니다. Aurora PostgreSQL은 PostgreSQL을 즉시 대체할 수 있고 새로 배포하는 PostgreSQL이든, 혹은 기존에 배포한 PostgreSQL이든 상관없이 설치, 조작 및 조정이 간편하고 비용 효율적이기 때문에 비즈니스와 애플리케이션에 더욱 많은 시간을 투자할 수 있습니다. Amazon RDS는 프로비저닝, 패치, 백업, 복구, 결함 감지, 수리 등 일상적인 데이터베이스 작업을 처리할 수 있는 Aurora 관리 기능을 지원합니다. 또한 Amazon RDS는 기존 Amazon RDS for PostgreSQL 애플리케이션을 Aurora PostgreSQL로 전환할 수 있는 푸시 버튼식 마이그레이션 도구도 제공합니다.

Aurora PostgreSQL은 다수의 산업 표준에서 작동할 수 있습니다. 예를 들어, 데이터베이스를 사용하여 HIPAA 준수 애플리케이션을 구축하고 AWS와 체결한 이행 중인 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 범위 내 AWS 서비스](#)를 참조하십시오.

주제

- [Amazon Aurora PostgreSQL을 사용한 보안 \(p. 773\)](#)
- [새 SSL/TLS 인증서를 사용해 Aurora PostgreSQL DB 클러스터에 연결할 애플리케이션을 업데이트 \(p. 777\)](#)
- [데이터를 PostgreSQL과 호환되는 Amazon Aurora로 마이그레이션 \(p. 781\)](#)
- [Amazon Aurora PostgreSQL 관리 \(p. 802\)](#)
- [Amazon Aurora PostgreSQL을 사용한 복제 \(p. 807\)](#)
- [Amazon Aurora PostgreSQL을 다른 AWS 서비스와 통합 \(p. 812\)](#)
- [Aurora PostgreSQL DB 클러스터의 데이터를 Amazon S3으로 내보내기 \(p. 812\)](#)
- [Aurora PostgreSQL용 쿼리 실행 계획 관리 \(p. 821\)](#)
- [Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시 \(p. 847\)](#)
- [Aurora PostgreSQL에서 기계 학습\(ML\) 사용 \(p. 850\)](#)
- [장애 조치 후 Aurora PostgreSQL용 클러스터 캐시 관리를 통한 신속한 복구 \(p. 868\)](#)
- [Aurora PostgreSQL용 PostgreSQL DB 엔진 업그레이드 \(p. 872\)](#)
- [Amazon Aurora PostgreSQL 모범 사례 \(p. 878\)](#)
- [Aurora PostgreSQL과 함께 Kerberos 인증 사용 \(p. 885\)](#)
- [Amazon Aurora PostgreSQL 참조 \(p. 895\)](#)
- [Amazon Aurora PostgreSQL 데이터베이스 엔진 업데이트 \(p. 905\)](#)

Amazon Aurora PostgreSQL을 사용한 보안

Amazon Aurora PostgreSQL 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)을 사용합니다. IAM 자격 증명을 사용하여 AWS에 연결할 때는 Amazon RDS 관리 작업에 필요한 권한을 부여할 수 있는 IAM 정책이 IAM 계정에 반드시 필요합니다. 자세한 내용은 [Amazon Aurora의 Identity and Access Management\(IAM\) \(p. 960\)](#) 단원을 참조하십시오.

IAM 계정을 사용해 Amazon RDS 콘솔에 액세스하려면 먼저 IAM 계정으로 AWS Management 콘솔에 로그인한 다음 Amazon RDS에서 <https://console.aws.amazon.com/rds> 콘솔로 이동합니다.

- Aurora DB 클러스터는 Amazon Virtual Private Cloud(VPC)에서 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 Secure Sockets Layer(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 Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#) 단원을 참조하십시오.

Aurora PostgreSQL은 기본 VPC일 때만 db.r4 및 db.t3 인스턴스 클래스를 지원합니다. 기본 VPC 테넌시 일 때는 VPC가 공유 하드웨어에서 실행됩니다. 하지만 전용 VPC 테넌시일 때는 VPC가 전용 하드웨어 인스턴스에서 실행됩니다. 인스턴스 클래스에 대한 자세한 내용은 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 참조하십시오. 기본 및 전용 VPC 테넌시에 대한 자세한 내용은 [Linux 인스턴스용 Amazon EC2 사용 설명서](#)에 서 [전용 인스턴스](#) 단원을 참조하십시오.

- Amazon Aurora DB 클러스터 로그인 및 사용 권한을 인증하려면 독립형 PostgreSQL 인스턴스와 동일한 접근 방식을 사용하십시오.

`CREATE ROLE`, `ALTER ROLE`, `GRANT`, `REVOKE` 등의 명령은 온프레미스 데이터베이스에서 작동하는 것과 마찬가지로 작동하며, 데이터베이스 스키마 테이블을 직접 수정할 때도 동일합니다. 자세한 내용은 PostgreSQL 설명서에서 [Client Authentication](#)을 참조하십시오.

Note

SCRAM(Salted Challenge Response Authentication Mechanism)은 Aurora PostgreSQL에서 지원되지 않습니다.

Amazon Aurora PostgreSQL DB 인스턴스를 생성할 때 마스터 사용자는 다음과 같은 기본 권한을 갖습니다.

- `LOGIN`
- `NOSUPERUSER`
- `INHERIT`
- `CREATEDB`
- `CREATEROLE`
- `NOREPLICATION`
- `VALID UNTIL 'infinity'`

DB 클러스터를 생성할 때는 각 DB 클러스터의 관리 서비스를 위해 `rdsadmin` 사용자가 만들어집니다. `rdsadmin` 계정에 대한 암호를 삭제하거나 이름 바꾸기를 하거나 변경하려고 시도하면 또는 권한을 변경하려고 시도하면 오류가 발생합니다.

암호 관리 제한

데이터베이스 사용자 암호를 관리할 수 있는 사람을 특정 역할로 제한할 수 있습니다. 이렇게 하면 클라이언트 측의 암호 관리를 더 잘 제어할 수 있습니다.

정적 파라미터 `rds.restrict_password_commands`로 제한된 암호 관리를 활성화하고 `rds_password`라는 역할을 사용합니다. `rds.restrict_password_commands` 파라미터를 1로 설정하면, `rds_password` 역할의 멤버인 사용자만 특정 SQL 명령을 실행할 수 있습니다. 제한된 SQL 명령은 데이터베이스 사용자 암호와 암호 만료 시간을 수정하는 명령입니다.

제한된 암호 관리를 사용하려면 DB 클러스터는 PostgreSQL 10.6 이상용 Amazon Aurora를 실행해야 합니다. `rds.restrict_password_commands` 파라미터는 정적이므로 이 파라미터를 변경하려면 데이터베이스를 다시 시작해야 합니다.

데이터베이스에 제한된 암호 관리가 활성화되어 있을 때 제한된 SQL 명령을 실행하려고 하면 ERROR: must be a member of `rds_password` to alter passwords 오류가 표시됩니다.

다음은 제한된 암호 관리가 활성화되어 있을 때 제한되는 몇 가지 SQL 명령 예입니다.

```
postgres=> CREATE ROLE myrole WITH PASSWORD 'mypassword';
postgres=> CREATE ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole WITH PASSWORD 'mypassword' VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole WITH PASSWORD 'mypassword';
postgres=> ALTER ROLE myrole VALID UNTIL '2020-01-01';
postgres=> ALTER ROLE myrole RENAME TO myrole2;
```

RENAME TO가 포함된 일부 ALTER ROLE 명령도 제한될 수 있습니다. 이러한 명령이 제한될 수 있는 이유는 MD5 암호가 있는 PostgreSQL 역할 이름을 바꾸면 암호가 지워지기 때문입니다.

rds_superuser 역할에는 기본적으로 rds_password 역할의 멤버십이 있으므로 변경할 수 없습니다. GRANT SQL 명령을 사용하여 다른 역할에 rds_password 역할의 멤버십을 제공할 수 있습니다. 암호 관리에만 사용하는 몇 가지 역할에만 rds_password 멤버십을 제공하는 것이 좋습니다. 이러한 역할에는 다른 역할을 수정할 CREATEROLE 속성이 필요합니다.

만료 및 클라이언트 측에 필요한 복잡성 등의 암호 요구 사항을 확인해야 합니다. 자체 클라이언트 측 유ти리티를 사용하여 암호 관련 변경을 제한하는 것이 좋습니다. 이 유ти리티에는 rds_password의 멤버이며 CREATEROLE 역할 속성이 있는 역할이 있어야 합니다.

SSL을 이용한 Aurora PostgreSQL 데이터 보안

Amazon RDS는 Aurora PostgreSQL DB 클러스터를 위한 SSL(Secure Socket Layer) 암호화를 지원합니다. SSL을 사용하여 애플리케이션과 Aurora PostgreSQL DB 클러스터 사이의 연결을 암호화할 수 있습니다. 또한 Aurora PostgreSQL DB 클러스터에 대한 모든 연결에서 SSL을 사용하도록 지정할 수도 있습니다.

SSL 지원 및 PostgreSQL 데이터베이스에 대한 일반적인 정보는 PostgreSQL 설명서의 [SSL 지원](#)을 참조하십시오. JDBC를 통한 SSL 연결 사용에 대한 자세한 내용은 PostgreSQL 설명서에서 [클라이언트 구성](#)을 참조하십시오.

주제

- [Aurora PostgreSQL DB 클러스터에 대한 SSL 연결 요구 \(p. 776\)](#)
- [SSL 연결 상태 확인 \(p. 776\)](#)

Aurora PostgreSQL의 모든 AWS 리전에서 SSL 지원 기능을 사용할 수 있습니다. Amazon RDS는 DB 클러스터가 생성될 때 Aurora PostgreSQL DB 클러스터의 SSL 인증서를 생성합니다. SSL 인증서 확인을 활성화하는 경우에는 SSL 인증서에 스폐핑 공격으로부터 보호해주는 SSL 인증서를 위한 일반 이름(CN)으로 DB 클러스터 엔드포인트가 포함됩니다.

SSL을 통해 Aurora PostgreSQL DB 클러스터에 연결하려면

1. 인증서를 다운로드합니다.

인증서 다운로드에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#) 단원을 참조하십시오.

2. 다음과 같이 사용 중인 운영 체제로 인증서를 가져옵니다.
3. SSL을 통해 Aurora PostgreSQL DB 클러스터에 연결합니다.

SSL을 사용하여 연결하면 클라이언트가 인증서 체인을 확인할지 여부를 선택할 수 있습니다. 연결 파라미터가 sslmode=verify-ca 또는 sslmode=verify-full을 지정하면 클라이언트는 RDS CA 인증서가 트러스트 스토어에 있거나 연결 URL에서 참조되게 할 것을 요구합니다. 이 요구 사항은 데이터베이스 인증서에 서명하는 인증서 체인을 확인하는 것입니다.

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 JDBC에서 사용되는 기본 sslmode 모드는 서로 다릅니다. libpq 기반 클라이언트는 prefer로 기본 설정되어 있고, JDBC 클라이언트는 verify-full로 기본 설정되어 있습니다.

`sslrootcert` 파라미터를 사용하여 인증서를 참조합니다(예: `sslrootcert=rds-ssl-ca-cert.pem`).

다음은 `psql`을 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 연결 요구

`rds.force_ssl` 파라미터를 사용하여 Aurora PostgreSQL DB 클러스터에 대한 연결 시 SSL을 사용하도록 요구할 수 있습니다. 기본적으로 `rds.force_ssl` 파라미터는 0(해제)으로 설정됩니다. `rds.force_ssl` 파라미터를 1(설정)로 설정하면 해당 DB 클러스터에 대한 연결 시 SSL을 요구합니다. `rds.force_ssl` 파라미터를 업데이트해도 PostgreSQL `ssl` 파라미터가 1(설정)로 설정되고, DB 클러스터의 `pg_hba.conf` 파일이 새로운 SSL 구성을 지원하도록 수정됩니다.

`rds.force_ssl` 파라미터 값은 DB 클러스터의 파라미터 그룹을 업데이트하여 설정할 수 있습니다. DB 클러스터의 파라미터 그룹이 기본 파라미터 그룹이 아니고 `rds.force_ssl` 파라미터를 1로 설정할 때 `ssl` 파라미터가 이미 1로 설정되어 있을 경우 DB 클러스터를 재부팅할 필요가 없습니다. 그렇지 않을 경우 변경 사항을 적용하려면 DB 클러스터를 재부팅해야 합니다. 파라미터 그룹에 대한 자세한 내용은 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

DB 클러스터에 대해 `rds.force_ssl` 파라미터를 1로 설정하면 연결 시 다음과 같이 SSL이 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 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 SSL이 사용 중인지 확인할 수 있습니다. 이 함수는 연결이 SSL을 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 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 기능](#)을 참조하십시오.

새 SSL/TLS 인증서를 사용해 Aurora PostgreSQL DB 클러스터에 연결할 애플리케이션을 업데이트

2019년 9월 19일부터 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. 이전 CA 인증서는 2020년 3 월 5일에 만료됩니다. 아래에서 새 인증서를 사용하기 위해 애플리케이션을 업데이트하는 방법에 관한 정보를 찾으실 수 있습니다. 애플리케이션에서 SSL/TLS를 사용해 Aurora DB 클러스터에 연결하는 경우 2020년 3월 5일 이전에 다음 단계를 수행하셔야 합니다. 이렇게 하면 애플리케이션과 Aurora DB 인스턴스 간에 연결이 중단되는 것을 방지할 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 \(p. 949\)](#) 단원을 참조하십시오. 인증서 다운로드에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#) 단원을 참조하십시오. PostgreSQL DB 클러스터에서 SSL/TLS를 사용하는 방법에 관한 자세한 내용은 [SSL을 이용한 Aurora PostgreSQL 데이터 보안 \(p. 775\)](#) 단원을 참조하십시오.

주제

- 애플리케이션에서 SSL을 사용해 Aurora PostgreSQL DB 클러스터에 연결하는지 여부 확인 (p. 778)
- 클라이언트에서 연결을 위해 인증서 확인이 필요한지 여부 확인 (p. 778)
- 애플리케이션 트러스트 스토어 업데이트 (p. 779)
- 다양한 유형의 애플리케이션에 대해 SSL/TLS 연결 사용 (p. 780)

애플리케이션에서 SSL을 사용해 Aurora PostgreSQL DB 클러스터에 연결하는지 여부 확인

`rds.force_ssl` 파라미터의 값에 대한 DB 클러스터 구성은 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 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 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, username, 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 username <> 'rdsadmin';
```

SSL/TLS 연결을 사용하는 행만 연결에 관한 정보와 함께 표시됩니다. 다음은 출력 샘플입니다.

```
datname | username | 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 JDBC에서 사용되는 기본 `sslmode` 모드는 서로 다릅니다. libpq 기반 클라이언트는 `prefer`로 기본 설정되어 있고, JDBC 클라이언트는 `verify-full`로 기본 설정되어 있습니다. 서버의 인증서는 `sslrootcert`에서 `sslmode`가 `require`, `verify-ca` 또는 `verify-full`로 설정된 경우에만 확인됩니다. 인증서가 잘못된 경우 오류가 발생합니다.

`PGSSLROOTCERT`를 사용해 `PGSSLMODE`가 `require`, `verify-ca` 또는 `verify-full`로 설정된 `PGSSLMODE` 환경 변수로 인증서를 확인하십시오.

```
PGSSLMODE=require PGSSLROOTCERT=/fullpath/rds-ca-2019-root.pem psql -h pgdbidentifier.cxxxxxxxxx.us-east-2.rds.amazonaws.com -U masteruser -d postgres
```

sslrootcert 인수를 사용해 sslmode가 require, verify-ca 또는 verify-full로 설정된 연결 문자열 형식의 sslmode로 인증서를 확인하십시오.

```
psql "host=pgdbidentifier.cxxxxxxxxx.us-east-2.rds.amazonaws.com sslmode=require  
sslrootcert=/full/path/rds-ca-2019-root.pem user=masteruser 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 업데이트 할 수 있습니다.

JDBC 애플리케이션에 대해 트러스트 스토어를 업데이트 하려면

- 모든 AWS 리전에서 작동하는 2019 루트 인증서를 다운로드하고 이 파일을 트러스트 스토어 디렉터리에 저장하십시오.

루트 인증서 다운로드에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화](#) (p. 947) 단원을 참조하십시오.

- 다음 명령을 사용하여 인증서를 .der 형식으로 변환합니다.

```
openssl x509 -outform der -in rds-ca-2019-root.pem -out rds-ca-2019-root.der
```

파일 이름을 다운로드한 파일 이름으로 바꿉니다.

- 다음 명령을 사용하여 인증서를 키 스토어로 가져옵니다.

```
keytool -import -alias rds-root -keystore clientkeystore -file rds-ca-2019-root.der
```

- 키 스토어가 성공적으로 업데이트되었는지 확인하십시오.

```
keytool -list -v -keystore clientkeystore.jks
```

메시지가 표시되면 키 스토어 암호를 입력하십시오.

출력에 다음 사항이 포함되어 있어야 합니다.

```
rds-root, date, trustedCertEntry,  
Certificate fingerprint (SHA1):  
D4:0D:DB:29:E3:75:0D:FF:A6:71:C3:14:0B:BF:5F:47:8D:1C:80:96  
# This fingerprint should match the output from the below command  
openssl x509 -fingerprint -in rds-ca-2019-root.pem -noout
```

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 설명서](#)를 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/TLS를 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 설명서](#)를 참조하십시오. PostgreSQL 데이터베이스에 연결하는 방법에 대한 짧은 자습서는 [Psycopg2 자습서](#)를 참조하십시오. [psycopg2 모듈 콘텐츠](#)에서 연결 명령이 수락하는 옵션에 대한 정보를 얻을 수 있습니다.

Important

데이터베이스 연결에서 SSL/TLS를 사용함을 확인하고 애플리케이션 트러스트 스토어를 업데이트한 후에는 데이터베이스에서 `rds-ca-2019` 인증서를 사용하도록 업데이트할 수 있습니다. 지침은 [DB 인스턴스를 수정하여 CA 인증서 업데이트](#) (p. 950)의 3단계를 참조하십시오.

데이터를 PostgreSQL과 호환되는 Amazon Aurora로 마이그레이션

기존 데이터베이스의 데이터를 PostgreSQL과 호환되는 Amazon Aurora DB 클러스터로 마이그레이션하기 위한 몇 가지 옵션이 있습니다. 마이그레이션할 데이터베이스와 데이터 크기에 따라서도 마이그레이션 옵션이 달라집니다. 마이그레이션 옵션은 다음과 같습니다.

RDS PostgreSQL DB 인스턴스에서 마이그레이션

Amazon RDS PostgreSQL DB 스냅샷의 데이터를 Aurora PostgreSQL DB 클러스터로 직접 마이그레이션할 수 있습니다. 자세한 내용은 [RDS PostgreSQL DB 스냅샷을 Aurora PostgreSQL DB 클러스터로 마이그레이션 \(p. 781\)](#) 단원을 참조하십시오.

또한 PostgreSQL DB 인스턴스의 Aurora PostgreSQL 읽기 전용 복제본을 생성하여 RDS PostgreSQL DB 인스턴스에서 마이그레이션할 수도 있습니다. PostgreSQL DB 인스턴스와 Aurora PostgreSQL 읽기 복제본 사이의 복제 지연 시간이 0이 되면 복제를 멈출 수 있습니다. 이때부터 읽기 및 쓰기 작업에서 Aurora 읽기 복제본을 듀립 실행형 Aurora PostgreSQL DB 클러스터로 사용할 수 있습니다. 자세한 내용은 [Aurora 읽기 전용 복제본을 사용하여 RDS PostgreSQL DB 인스턴스의 데이터를 Aurora PostgreSQL DB 클러스터로 마이그레이션 \(p. 783\)](#) 단원을 참조하십시오.

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 Service란 무엇입니까?](#)를 참조하십시오.

Amazon S3 데이터 가져오기

Amazon S3 데이터는 RDS PostgreSQL DB 인스턴스의 Aurora PostgreSQL DB 클러스터에 속한 테이블로 가져와서 마이그레이션할 수 있습니다. 자세한 내용은 [PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터 \(p. 791\)](#) 단원을 참조하십시오.

Aurora를 사용할 수 있는 AWS 리전 목록은 AWS General Reference에서 [Amazon Aurora](#) 단원을 참조하십시오.

RDS PostgreSQL DB 스냅샷을 Aurora PostgreSQL DB 클러스터로 마이그레이션

Aurora PostgreSQL DB 클러스터를 생성하여 RDS PostgreSQL DB 인스턴스의 DB 스냅샷을 마이그레이션 할 수 있습니다. 새롭게 생성된 Aurora PostgreSQL DB 클러스터는 원본 RDS PostgreSQL DB 인스턴스의 데이터로 채워집니다. 이때 DB 스냅샷은 PostgreSQL 9.6.1 또는 9.6.3 기반 RDS DB 인스턴스의 스냅샷이어야 합니다. DB 스냅샷 생성에 대한 자세한 내용은 [DB 스냅샷 생성](#) 단원을 참조하십시오.

경우에 따라 DB 스냅샷이 데이터를 저장할 AWS 리전에 속하지 않을 수도 있습니다. 이때는 Amazon RDS 콘솔을 사용해 DB 스냅샷을 해당 AWS 리전으로 복사하십시오. DB 스냅샷 복사에 대한 자세한 내용은 [DB 스냅샷 복사](#) 단원을 참조하십시오.

콘솔을 사용하여 DB 스냅샷을 마이그레이션할 경우, 콘솔에서 DB 클러스터와 기본 인스턴스 모두를 생성하는 데 필요한 작업이 따릅니다.

AWS Key Management Service(AWS KMS) 암호화 키를 사용하여 새 Aurora PostgreSQL DB 클러스터가 유휴 상태에서 암호화되도록 선택할 수도 있습니다. 이 옵션은 암호화되지 않은 DB 스냅샷에만 가능합니다.

RDS 콘솔을 사용해 PostgreSQL DB 스냅샷을 마이그레이션하려면

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

2. [Snapshots]를 선택합니다.
 3. 스냅샷 페이지에서 Aurora PostgreSQL DB 클러스터로 마이그레이션하려는 스냅샷을 선택합니다.
 4. [Migrate Database]를 선택합니다.
 5. [Migrate Database] 페이지에서 다음과 같이 값을 설정합니다.
 - DB 인스턴스 클래스: 데이터베이스에 필요한 스토리지 및 용량이 있는 DB 인스턴스 클래스를 선택합니다(예: db.r3.large). Aurora 클러스터 볼륨은 데이터베이스의 데이터 양이 증가함에 따라 최대 크기인 64 tebibytes (TiB)까지 자동으로 증가합니다. 따라서 현재 스토리지 요구 사항에 맞는 DB 인스턴스 클래스를 선택해야 합니다. 자세한 내용은 [Aurora 스토리지 개요 \(p. 34\)](#) 단원을 참조하십시오.
 - DB 인스턴스 식별자: 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.
 - AWS 리전별로 AWS 계정 하나당 모든 DB 인스턴스는 고유해야 합니다.
 - VPC: 기존 VPC가 있을 경우 해당 VPC 식별자(예: vpc-a464d1c1)를 선택하여 이 VPC를 Aurora PostgreSQL DB 클러스터에 사용할 수 있습니다. 기존 VPC 사용 방법에 대한 자세한 내용은 [Amazon Aurora에 사용할 VPC의 생성 방법 \(p. 1005\)](#) 단원을 참조하십시오.

기존 VPC가 없다면 [Create a new VPC]를 선택하여 Amazon RDS에서 VPC를 새로 생성하도록 할 수 있습니다.

- 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.

기존 서브넷 그룹이 없다면 [Create a new subnet group]을 선택하여 Amazon RDS에서 서브넷 그룹을 새로 생성하도록 할 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. 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 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 5432입니다.

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 활성: 새 Aurora PostgreSQL DB 클러스터가 "유휴 상태에서" 암호화하게 하려면 예를 선택합니다. 예를 선택하면 AWS KMS 암호화 키를 마스터 키 값으로 선택해야 합니다.
- 마이너 버전 자동 업그레이드: PostgreSQL DB 엔진의 마이너 버전 업그레이드가 있을 때 Aurora PostgreSQL DB 클러스터가 업그레이드를 자동으로 수신하도록 하려면 Enable auto minor version upgrade(마이너 버전 자동 업그레이드 활성화)를 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. [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 앤드포인트가 표시됩니다. Aurora PostgreSQL DB 클러스터 연결에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 연결 \(p. 162\)](#) 단원을 참조하십시오.

Aurora 읽기 전용 복제본을 사용하여 RDS PostgreSQL DB 인스턴스의 데이터를 Aurora PostgreSQL DB 클러스터로 마이그레이션

Aurora 읽기 전용 복제본을 사용하여 PostgreSQL DB 인스턴스에서 Aurora PostgreSQL DB 클러스터로 마이그레이션할 수 있습니다. RDS PostgreSQL DB 인스턴스에서 Aurora PostgreSQL DB 클러스터로 마이그레이션해야 할 때는 이러한 방식을 사용하는 것이 좋습니다.

이때 Amazon RDS는 PostgreSQL DB 엔진의 스트리밍 복제 기능을 사용해 PostgreSQL DB 인스턴스에 따라 특정 유형의 DB 클러스터를 생성합니다. 이러한 유형의 DB 클러스터를 Aurora 읽기 전용 복제본이라고 부릅니다. 원본 PostgreSQL DB 인스턴스에 적용된 업데이트는 Aurora 읽기 전용 복제본에 비동기 방식으로 복제됩니다.

주제

- [Aurora 읽기 전용 복제본을 사용한 데이터 마이그레이션 개요 \(p. 783\)](#)
- [Aurora 읽기 전용 복제본을 사용한 데이터 마이그레이션 준비 \(p. 784\)](#)
- [Aurora 읽기 전용 복제본 생성 \(p. 784\)](#)
- [Aurora 읽기 전용 복제본 승격 \(p. 790\)](#)

Aurora 읽기 전용 복제본을 사용한 데이터 마이그레이션 개요

RDS PostgreSQL DB 인스턴스에서 Aurora PostgreSQL DB 클러스터로 마이그레이션하려면 원본 PostgreSQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하는 것이 좋습니다. PostgreSQL DB 인스턴스와 Aurora PostgreSQL 읽기 복제본 사이의 복제 지연 시간이 0이 되면 복제를 멈출 수 있습니다. 이때 Aurora 읽기 복제본을 독립 실행형 Aurora PostgreSQL DB 클러스터로 승격시킬 수 있습니다. 이렇게 승격된 독립 실행형 DB 클러스터는 쓰기 부하를 허용합니다.

マイグレーションには、相当な時間がかかる場合があります。データベースのトランザクションが実行中の間は、Amazon RDS PostgreSQL DB インスタンスは Write Ahead Log(WAL) セグメントを記録します。Amazon RDS インスタンスにこのセグメントを適切に記録するためには、十分なストレージ容量が必要です。

PostgreSQL DB 인스턴스의 Aurora 읽기 전용 복제본을 생성하면 Amazon RDS는 원본 PostgreSQL DB 인스턴스의 DB 스냅샷을 만듭니다. 이 스냅샷은 Amazon RDS 전용이며 비용이 따로 발생하지 않습니다. 그런 다음 Amazon RDS가 데이터를 DB 스냅샷에서 Aurora 읽기 전용 복제본으로 마이그레이션합니다. DB 스냅샷 데이터가 새로운 Aurora PostgreSQL DB 클러스터로 마이그레이션된 후 RDS는 PostgreSQL DB 인스턴스와 Aurora PostgreSQL DB 클러스터 간 복제를 시작합니다.

PostgreSQL DB 인스턴스 하나에 Aurora 읽기 전용 복제본 하나만 만들 수 있습니다. Amazon RDS PostgreSQL 인스턴스의 Aurora 읽기 전용 복제본을 생성하려고 하는데 이미 읽기 전용 복제본이 있는 경우 요청이 거부됩니다.

Note

Aurora PostgreSQL과 복제 마스터인 RDS PostgreSQL DB 인스턴스의 PostgreSQL 엔진 버전 간 기능 차이로 인해 복제 문제가 발생할 수 있습니다. 해당하는 Aurora PostgreSQL 버전과 호환되는 Amazon RDS PostgreSQL 인스턴스에서만 복제할 수 있습니다. 예를 들어 지원되는 Aurora PostgreSQL 버전이 9.6.3인 경우 Amazon RDS PostgreSQL DB 인스턴스는 버전 9.6.1 이상을 실행해야 합니다. 오류가 발생하면 [Amazon RDS 커뮤니티 포럼](#) 또는 AWS Support에서 지원을 받을 수 있습니다.

PostgreSQL 읽기 전용 복제본에 대한 자세한 내용은 Amazon RDS 사용 설명서의 [읽기 전용 복제본 작업](#)을 참조하십시오.

Aurora 읽기 전용 복제본을 사용한 데이터 마이그레이션 준비

데이터를 RDS PostgreSQL 인스턴스에서 Aurora PostgreSQL 클러스터로 마이그레이션하려면 먼저 인스턴스의 스토리지 용량이 충분한지 확인해야 합니다. 스토리지 용량은 마이그레이션 과정에서 누적되는 Write Ahead Log(WAL) 세그먼트를 위한 것입니다. 이를 확인하기 위한 몇 가지 지표와 설명은 다음과 같습니다.

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

PostgreSQL DB 인스턴스의 Aurora 읽기 전용 복제본은 콘솔 또는 AWS CLI를 사용해 생성할 수 있습니다.

콘솔

원본 PostgreSQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면

1. AWS Management 콘솔에 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 PostgreSQL DB 인스턴스를 선택한 후 작업에서 Aurora 읽기 전용 복제본 생성을 선택합니다.

Databases			
	DB identifier	Role	Engine
<input type="radio"/>	<input checked="" type="checkbox"/> gs-db-cluster1	Regional	Aurora MySQL
<input type="radio"/>	js-cluster	Serverless	Aurora PostgreSQL
<input type="radio"/>	js-cluster-mysql	Serverless	Aurora MySQL
<input type="radio"/>	<input checked="" type="checkbox"/> js-cluster-mysql-1	Regional	Aurora MySQL
<input type="radio"/>	<input checked="" type="checkbox"/> js-db-cluster-1	Regional	Aurora PostgreSQL
<input checked="" type="radio"/>	jsdbinstance2	Instance	PostgreSQL

4. 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 요구를 정의한 DB 인스턴스 클래스를 선택합니다. DB 인스턴스 클래스 옵션에 대한 자세한 정보는 DB 인스턴스 클래스 (p. 30) 단원을 참조하십시오.
다중 AZ 배포	PostgreSQL에서는 사용할 수 없습니다.
DB 인스턴스 식별자	<p>Aurora 읽기 전용 복제본 DB 클러스터의 기본 인스턴스 이름을 입력합니다. 이 식별자는 새 DB 클러스터의 기본 인스턴스에 대한 엔드포인트 주소로 사용됩니다.</p> <p>DB 인스턴스 식별자는 다음과 같은 제약 조건이 있습니다.</p> <ul style="list-style-type: none">1~63자의 영숫자 또는 하이픈으로 구성되어야 합니다.첫 번째 문자는 글자이어야 합니다.하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.AWS 리전별로 각 AWS 계정의 모든 DB 인스턴스에 대해 고유해야 합니다. <p>Aurora 읽기 전용 복제본 DB 클러스터는 원본 DB 인스턴스의 스냅샷에서 생성됩니다. 따라서 Aurora 읽기 전용 복제본의 마스터 사용자 이름과 마스터 암호는 원본 DB 인스턴스의 마스터 사용자 이름 및 마스터 암호와 동일합니다.</p>

옵션	설명
가상 프라이빗 클라우드(VPC)	DB 클러스터를 호스팅할 VPC를 선택합니다. Amazon RDS에서 VPC를 생성하도록 하려면 새 VPC 생성을 선택합니다. 자세한 내용은 DB 클러스터 사전 요구사항 (p. 95) 단원을 참조하십시오.
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 요구사항 (p. 95) 단원을 참조하십시오.
퍼블릭 액세스 가능성	DB 클러스터에 퍼블릭 IP 주소를 할당하려면 예를 선택하고, 그렇지 않으면 아니요를 선택합니다. DB 클러스터의 인스턴스는 퍼블릭과 프라이빗 DB 인스턴스를 모두 혼합하여 사용할 수 있습니다. 모든 사용자의 액세스에서 인스턴스를 습기는 방법에 대한 자세한 내용은 VPC에 있는 DB 인스턴스를 인터넷에서 습기기 (p. 1017) 단원을 참조하십시오.
[Availability zone]	특정 가용 영역의 지정 여부를 결정합니다. 가용 영역에 대한 자세한 내용은 리전 및 가용 영역 (p. 3) 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 요구사항 (p. 95) 단원을 참조하십시오.
데이터베이스 포트	애플리케이션과 유ти리티가 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 클러스터의 DB 파라미터 그룹을 선택합니다. Aurora는 기본 DB 파라미터 그룹을 제공하며, DB 파라미터 그룹을 직접 생성할 수도 있습니다. DB 파라미터 그룹에 대한 자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.
DB 클러스터 파라미터 그룹	Aurora PostgreSQL DB 클러스터의 DB 클러스터 파라미터 그룹을 선택합니다. Aurora는 기본 DB 클러스터 파라미터 그룹을 제공하며, DB 클러스터 파라미터 그룹을 직접 생성할 수도 있습니다. DB 클러스터 파라미터 그룹에 대한 자세한 내용은 DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 (p. 168) 단원을 참조하십시오.
암호화	새 Aurora DB 클러스터를 유휴 상태에서 암호화하려면 [Enable encryption]을 선택합니다. 암호화 활성을 선택하면 마스터 키 값으로 AWS KMS 암호화 키도 선택해야 합니다.

옵션	설명
Priority	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 내결합성 (p. 268) 단원을 참조하십시오.
백업 보존 기간	-가 데이터베이스 백업 사본을 보존하는 기간을 1Aurora35 일로 선택합니다. 백업 사본은 데이터베이스를 마지막 특정 시점으로 복구(PITR)하는 데 사용됩니다.
확장 모니터링	DB 클러스터가 실행되는 운영 체제에 대한 실시간 지표 수집을 활성화 하려면 [Enable enhanced monitoring]을 선택합니다. 자세한 내용은 확장 모니터링 (p. 358) 단원을 참조하십시오.
역할 모니터링	확장 모니터링 활성화를 선택하는 경우에만 사용할 수 있습니다. 확장 모니터링에 사용할 AWS Identity and Access Management(IAM) 역할입니다. 자세한 내용은 확장 모니터링 설정 및 활성화 (p. 358) 단원을 참조하십시오.
Granularity	확장 모니터링 활성화를 선택하는 경우에만 사용할 수 있습니다. DB 클러스터에 대해 지표를 수집하는 간격(초)을 설정하십시오.
Auto minor version upgrade	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 엔진의 마이너 버전 업그레이드에만 적용됩니다. 시스템 안정성 유지를 위한 정기 패치에는 적용되지 않습니다.
유지 관리 기간	시스템 유지 관리를 실행하는 기간을 주 단위로 선택합니다.

5. [Create read replica]를 선택합니다.

AWS CLI

원본 PostgreSQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면 `create-db-cluster` 및 `create-db-instance` AWS CLI 명령을 사용하여 새 Aurora PostgreSQL DB 클러스터를 생성합니다. `create-db-cluster` 명령을 호출할 때는 원본 PostgreSQL DB 인스턴스의 Amazon 리소스 이름(ARN)을 식별하는 `--replication-source-identifier` 파라미터를 포함시키십시오. Amazon RDS ARN에 대한 자세한 내용은 AWS General Reference에서 [Amazon Relational Database Service\(Amazon RDS\)](#) 단원을 참조하십시오.

마스터 사용자 이름, 마스터 암호 또는 데이터베이스 이름은 지정하지 마십시오. Aurora 읽기 전용 복제본은 원본 PostgreSQL DB 인스턴스와 동일한 마스터 사용자 이름, 마스터 암호 및 데이터베이스 이름을 갖게 됩니다.

Linux, OS X, Unix의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine aurora-postgresql \
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 \
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:master-postgresql-instance
```

Windows의 경우:

```
aws rds create-db-cluster --db-cluster-identifier sample-replica-cluster --engine aurora-postgresql ^  
    --db-subnet-group-name mysubnetgroup --vpc-security-group-ids sg-c7e5b0d2 ^  
    --replication-source-identifier arn:aws:rds:us-west-2:123456789012:db:master-postgresql-instance
```

콘솔을 사용하여 Aurora 읽기 전용 복제본을 생성하면 RDS에서 DB 클러스터 Aurora 읽기 전용 복제본의 기본 인스턴스를 자동으로 생성합니다. 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` 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-postgresql`
사용할 데이터베이스 엔진입니다.

다음 예제에서는 이름이 `myreadreplicacluster`인 DB 클러스터에서 `myreadreplicainstance`라는 이름으로 기본 인스턴스를 생성합니다. 이때 `myinstanceclass`에서 지정한 DB 인스턴스 클래스를 사용합니다.

Example

Linux, OS X, Unix의 경우:

```
aws rds create-db-instance \  
    --db-cluster-identifier myreadreplicacluster \  
    --db-instance-class myinstanceclass \  
    --db-instance-identifier myreadreplicainstance \  
    --engine aurora-postgresql
```

Windows의 경우:

```
aws rds create-db-instance \  
    --db-cluster-identifier myreadreplicacluster \  
    --db-instance-class myinstanceclass \  
    --db-instance-identifier myreadreplicainstance \  
    --engine aurora-postgresql
```

RDS API

원본 PostgreSQL DB 인스턴스에서 Aurora 읽기 전용 복제본을 생성하려면 RDS API 작업인 `CreateDBCluster`와 `CreateDBInstance`를 사용해 새 Aurora DB 클러스터와 기본 인스턴스를 생성합니다. 마스터 사용자 이름, 마스터 암호 또는 데이터베이스 이름은 지정하지 마십시오. Aurora 읽기 전용 복제

본은 원본 PostgreSQL DB 인스턴스와 동일한 마스터 사용자 이름, 마스터 암호 및 데이터베이스 이름을 갖게 됩니다.

원본 PostgreSQL DB 인스턴스에서 Aurora 읽기 전용 복제본의 새 Aurora DB 클러스터를 생성할 수 있습니다. RDS API 작업인 [createDBCluster](#)를 다음 파라미터와 함께 사용하면 가능합니다.

- **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\(Amazon RDS\)](#) 단원을 참조하십시오.

- **VpcSecurityGroupIds**

DB 클러스터와 연결할 Amazon EC2 VPC 보안 그룹 목록입니다.

다음 예제에서는 소스 PostgreSQL DB 인스턴스에서 *myreadreplicacluster*라는 이름의 DB 클러스터를 생성합니다. 이 클러스터는 ARN이 *mysqlmasterARN*으로 설정됩니다. 또한 이름이 *mysubnetgroup*인 DB 서브넷 그룹과 이름이 *mysecuritygroup*인 VPC 보안 그룹에 연결됩니다.

Example

```
https://rds.us-east-1.amazonaws.com/
?Action=CreateDBCluster
&DBClusterIdentifier=myreadreplicacluster
&DBSubnetGroupName=mysubnetgroup
&Engine=aurora-postgresql
&ReplicationSourceIdentifier=mysqlmasterARN
&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 Amazon RDS에서 DB 클러스터 Aurora 읽기 전용 복제본의 기본 인스턴스를 자동으로 생성합니다. CLI를 사용하여 Aurora 읽기 전용 복제본을 생성할 경우 반드시 DB 클러스터용 기본 인스턴스를 명시적으로 생성해야 합니다. 기본 인스턴스는 클러스터에 생성된 최초의 DB 인스턴스입니다.

RDS API 작업인 [CreateDBInstance](#)를 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-postgresql**

사용할 엔진 이름입니다.

다음 예제에서는 이름이 **myreadreplicacluster**인 DB 클러스터에서 **myreadreplicainstance**라는 이름으로 기본 인스턴스를 생성합니다. 이때 **myinstanceclass**에서 지정한 DB 인스턴스 클래스를 사용합니다.

Example

```
https://rds.us-east-1.amazonaws.com/
?Action/CreateDBInstance
&DBClusterIdentifier=myreadreplicacluster
&DBInstanceClass=myinstanceclass
&DBInstanceIdentifier=myreadreplicainstance
&Engine=aurora-postgresql
&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=bee4aab750bf7dad0cd9e22b952bd6089d91e2a16592c2293e532eeaab8bc77
```

Aurora 읽기 전용 복제본 승격

マイグレーション이 완료된 후 Aurora 읽기 전용 복제본을 독립 실행형 DB 클러스터로 승격시키고 클라이언트 애플리케이션을 Aurora 읽기 전용 복제본의 엔드포인트로 보낼 수 있습니다. Aurora 엔드포인트에 대한 자세한 내용은 [Amazon Aurora 연결 관리 \(p. 9\)](#) 단원을 참조하십시오. 승격은 매우 신속하게 완료해야 합니다. 승격이 완료될 때까지 마스터 PostgreSQL DB 인스턴스를 삭제하거나 DB 인스턴스와 Aurora 읽기 전용 복제본의 링크를 해제할 수 없습니다.

Aurora 읽기 전용 복제본을 승격시키기 전에 원본 PostgreSQL DB 인스턴스에 대한 트랜잭션 쓰기를 종단합니다. 그런 다음 Aurora 읽기 전용 복제본에서 복제 지연 시간이 0이 될 때까지 기다립니다.

읽기 전용 복제본을 승격한 후 승격이 완료되었는지 확인합니다. 이를 위해 탐색 창에서 인스턴스를 선택한 후 해당 읽기 전용 복제본에 대한 Promoted Read Replica cluster to stand-alone database cluster(독립형 데이터베이스 클러스터로 승격된 읽기 전용 복제본 클러스터) 이벤트가 있는지 확인합니다. 승격이 완료된 후에는 마스터 PostgreSQL DB 인스턴스와 Aurora 읽기 전용 복제본의 링크가 해제됩니다. 이때 원한다면 DB 인스턴스를 안전하게 삭제할 수 있습니다.

콘솔

Aurora 읽기 전용 복제본을 Aurora DB 클러스터로 승격시키려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 인스턴스를 선택합니다.
3. Aurora 읽기 전용 복제본의 DB 인스턴스를 선택한 후 작업에 대해 승격을 선택합니다.
4. [Promote Read Replica]를 선택합니다.

AWS CLI

Aurora 읽기 전용 복제본을 독립형 DB 클러스터로 승격시키려면 `promote-read-replica-db-cluster` AWS CLI 명령을 사용하십시오.

Example

Linux, OS X, Unix의 경우:

```
aws rds promote-read-replica-db-cluster \
--db-cluster-identifier myreadreplicacluster
```

Windows의 경우:

```
aws rds promote-read-replica-db-cluster ^
--db-cluster-identifier myreadreplicacluster
```

PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터

PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터. 이를 위해서는 Aurora PostgreSQL가 제공하는 `aws_s3` PostgreSQL 확장을 사용합니다.

Note

Amazon S3에서 Aurora PostgreSQL로 가져오려면 데이터베이스에서 PostgreSQL 버전 10.7 이상을 실행 중이어야 합니다.

Amazon S3를 이용한 데이터 저장에 대한 자세한 내용은 Amazon Simple Storage Service 시작 안내서의 [버킷 생성](#)을 참조하십시오. 파일을 Amazon S3 버킷에 업로드하는 방법에 관한 지침은 Amazon Simple Storage Service 시작 안내서의 [버킷에 객체 추가](#)를 참조하십시오.

주제

- [Amazon S3 데이터 가져오기 개요](#) (p. 791)
- [Amazon S3 버킷에 대한 액세스 권한 설정](#) (p. 792)
- [aws_s3.table_import_from_s3](#) 함수를 사용하여 Amazon S3 데이터 가져오기 (p. 797)
- [함수 참조](#) (p. 799)

Amazon S3 데이터 가져오기 개요

Amazon S3 버킷에 저장된 데이터를 PostgreSQL 데이터베이스 테이블로 가져오려면 다음 단계를 따릅니다.

S3 데이터를 Aurora PostgreSQL로 가져오려면

- 필요한 PostgreSQL 확장을 설치합니다. 여기에는 `aws_s3` 및 `aws_commons` 확장이 포함됩니다. 이렇게 하려면 `psql`을 시작하고 다음 명령을 사용합니다.

```
psql=> CREATE EXTENSION aws_s3 CASCADE;
NOTICE: installing required extension "aws_commons"
```

`aws_s3` 확장에서는 Amazon S3 데이터를 가져오는 데 사용하는 `aws_s3.table_import_from_s3` (p. 799) 함수를 제공합니다. `aws_commons` 확장은 추가 헬퍼 함수를 제공합니다.

- 사용할 데이터베이스 테이블과 Amazon S3 파일을 식별합니다.

[aws_s3.table_import_from_s3 \(p. 799\)](#) 함수에는 가져온 데이터를 배치할 PostgreSQL 데이터베이스 테이블의 이름이 필요합니다. 또한 이 함수에서 가져올 Amazon S3 파일도 식별해야 합니다. 이 정보를 제공하려면 다음 단계를 따릅니다.

- 데이터를 배치할 PostgreSQL 데이터베이스 테이블을 식별합니다. 예를 들어 다음은 이 항목의 예제에 사용되는 샘플 t1 데이터베이스 테이블입니다.

```
psql=> CREATE TABLE t1 (col1 varchar(80), col2 varchar(80), col3 varchar(80));
```

- 가져올 Amazon S3 파일을 식별하려면 다음 정보를 가져옵니다.

- 버킷 이름 – 버킷은 Amazon S3 객체 또는 파일을 위한 컨테이너입니다.
- 파일 경로 – 파일 경로를 통해 파일이 Amazon S3 버킷에 배치됩니다.
- AWS 리전 – AWS 리전은 Amazon S3 버킷이 있는 위치입니다. 예를 들어 S3 버킷이 미국 동부 (버지니아 북부) 리전에 있는 경우 us-east-1을 사용합니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 사용 영역 \(p. 3\)](#) 단원을 참조하십시오.

이 정보를 얻는 방법에 대해 알아보려면 Amazon Simple Storage Service 시작 안내서에서 [객체 보기](#)를 참조하십시오. AWS CLI 명령 aws s3 cp를 사용하여 정보를 확인할 수 있습니다. 정보가 정확하면 이 명령이 Amazon S3 파일의 복사본을 다운로드합니다.

```
aws s3 cp s3://sample_s3_bucket/sample_file_path ./
```

- [aws_commons.create_s3_uri \(p. 801\)](#) 함수를 사용하여 Amazon S3 파일 정보를 보관할 aws_commons._s3_uri_1 구조를 생성합니다. 이 aws_commons._s3_uri_1 구조를 [aws_s3.table_import_from_s3 \(p. 799\)](#) 함수에 대한 호출의 파라미터로 제공합니다.

psql 예제에서 다음을 참조하십시오.

```
psql=> SELECT aws_commons.create_s3_uri(  
    'sample_s3_bucket',  
    'sample.csv',  
    'us-east-1'  
) AS s3_uri \gset
```

- Amazon S3 파일에 액세스할 수 있는 권한을 부여합니다.

Amazon S3 파일에서 데이터를 가져오려면 Aurora PostgreSQL DB 클러스터에 파일이 저장된 Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다. 이렇게 하려면 AWS Identity and Access Management(IAM) 역할 또는 보안 자격 증명을 사용합니다. 자세한 내용은 [Amazon S3 버킷에 대한 액세스 권한 설정 \(p. 792\)](#) 단원을 참조하십시오.

- [aws_s3.table_import_from_s3](#) 함수를 호출하여 Amazon S3 데이터를 가져옵니다.

위 준비 작업을 완료한 후에는 [aws_s3.table_import_from_s3 \(p. 799\)](#) 함수를 사용해 Amazon S3 데이터를 가져옵니다. 자세한 내용은 [aws_s3.table_import_from_s3](#) 함수를 사용하여 Amazon S3 데이터 가져오기 (p. 797) 단원을 참조하십시오.

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 \(p. 793\)](#)

- 보안 자격 증명을 사용해 Amazon S3 버킷에 액세스 (p. 796)
- Amazon S3 액세스 문제 해결 (p. 796)

IAM 역할을 사용해 Amazon S3 버킷에 액세스

Amazon S3 파일에서 데이터를 로드하기 전에 Aurora PostgreSQL DB 클러스터에 파일이 저장된 Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다. 그러면 추가 자격 증명 정보를 관리하거나 [aws_s3.table_import_from_s3 \(p. 799\)](#) 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.

IAM 역할을 통해 Amazon S3에 액세스할 수 있는 권한을 PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터에 부여하려면

1. IAM 정책을 생성합니다. 이 정책은 Aurora PostgreSQL DB 클러스터가 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 다음과 같은 리소스를 포함하여 Amazon S3 버킷 및 그 안의 객체를 식별합니다. 다음은 Amazon S3에 액세스하기 위한 Amazon 리소스 이름(ARN) 형식입니다.

- arn:aws:s3:::*your-s3-bucket*
- arn:aws:s3:::*your-s3-bucket*/*

PostgreSQL용 Aurora PostgreSQL에 대한 IAM 정책 생성에 대한 자세한 내용은 [IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 \(p. 979\)](#) 단원을 참조하십시오. IAM 사용 설명서의 [자습서: 첫 번째 고객 관리형 정책 생성 및 연결](#)도 참조하십시오.

다음 AWS CLI 명령은 이 옵션으로 rds-s3-import-policy라는 IAM 정책을 만듭니다. *your-s3-bucket*이라는 버킷에 대한 액세스 권한을 부여합니다.

Note

정책을 만든 후에 정책의 Amazon 리소스 이름(ARN)을 기록하십시오. IAM 역할에 정책을 연결할 때 이후 단계에 ARN이 필요합니다.

Example

Linux, OS X, Unix의 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:::your-s3-bucket",
            "arn:aws:s3:::your-s3-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:::your-s3-bucket",
                "arn:aws:s3:::your-s3-bucket/*"
            ]
        }
    ]
}'
```

2. IAM 역할 생성. Aurora PostgreSQL이 이 IAM 역할을 수임하여 사용자 대신 Amazon S3 버킷에 액세스 할 수 있도록 역할을 생성합니다. 자세한 내용은 IAM 사용 설명서의 역할을 만들어 [IAM 사용자에게 권한 위임 단원](#)을 참조하십시오.

다음 예제에서는 AWS CLI 명령을 사용해 `rds-s3-import-role`이라는 역할을 생성하는 방법을 보여 줍니다.

Example

Linux, OS X, Unix의 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"
        }
    ]
}'
```

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"  
        }  
    ]  
}'
```

3. 생성한 IAM 역할에 생성한 IAM 정책을 연결합니다.

다음 AWS CLI 명령은 앞서 생성한 정책을 `rds-s3-import-role`라는 역할에 연결합니다. `your-policy-arn`을 이전 단계에서 기록한 정책 ARN으로 바꿉니다.

Example

Linux, OS X, Unix의 경우:

```
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 콘솔 또는 AWS CLI를 사용합니다.

콘솔

콘솔을 사용하여 PostgreSQL DB 클러스터에 대해 IAM 역할을 추가하려면

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/rds/>에서 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 Add IAM roles to this instance(이 인스턴스에 IAM 역할 추가)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. `--feature-name` 옵션의 값에 대해 `s3Import`을 사용합니다.

Example

Linux, OS X, Unix의 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
```

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
```

보안 자격 증명을 사용해 Amazon S3 버킷에 액세스

원활 경우 IAM 역할을 사용하는 대신 보안 자격 증명을 사용해 Amazon S3 버킷에 대한 액세스 권한을 부여 여할 수 있습니다. 이렇게 하려면 [aws_s3.table_import_from_s3 \(p. 799\)](#) 함수 호출에 credentials 파라미터를 사용합니다.

credentials 파라미터는 AWS 자격 증명을 포함하는 `aws_commons._aws_credentials_1` 유형의 구조입니다. 다음과 같이 [aws_commons.create_aws_credentials \(p. 802\)](#) 함수를 사용해 `aws_commons._aws_credentials_1` 구조에서 액세스 키와 비밀 키를 설정하십시오.

```
psql=> 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 \(p. 799\)](#) 함수를 credentials 파라미터와 함께 사용해 데이터를 가져옵니다.

```
psql=> SELECT aws_s3.table_import_from_s3(
  't', '', '(format csv)',
  :'s3_uri',
  :'creds'
);
```

또는 [aws_commons.create_aws_credentials \(p. 802\)](#) 함수 호출 인라인을 `aws_s3.table_import_from_s3` 함수 호출에 포함할 수 있습니다.

```
psql=> 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 참조하십시오.

- [Amazon Aurora 자격 증명 및 액세스 문제 해결 \(p. 993\)](#)

- Amazon S3 문제 해결
- Amazon S3 및 IAM 문제 해결

aws_s3.table_import_from_s3 함수를 사용하여 Amazon S3 데이터 가져오기

[aws_s3.table_import_from_s3 \(p. 799\)](#) 함수를 호출하여 Amazon S3 데이터를 가져옵니다.

Note

다음 예제에서는 Amazon S3 버킷에 대한 액세스 권한을 제공하기 위해 IAM 역할 방법을 사용합니다. 따라서 aws_s3.table_import_from_s3 함수 호출에는 자격 증명 파라미터가 없습니다.

다음은 psql을 사용하는 일반적인 PostgreSQL의 예입니다.

```
psql=> 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 복사됩니다. 열 목록 사용에 대한 예시는 [사용자 지정 구분 기호를 사용하는 Amazon S3 파일 가져오기 \(p. 797\)](#) 단원을 참조하십시오.
- (format csv) – PostgreSQL COPY 인수입니다. 복사 프로세스에서는 [PostgreSQL COPY 명령의 인수 및 형식](#)을 사용합니다. 앞의 예에서 COPY 명령은 쉼표로 구분된 값(CSV) 파일 형식을 사용해 데이터를 복사합니다.
- s3_uri – Amazon S3 파일을 식별하는 정보가 포함된 구조입니다.
[aws_commons.create_s3_uri \(p. 801\)](#) 함수를 사용하여 s3_uri 구조를 생성하는 예제는 [Amazon S3 데이터 가져오기 개요 \(p. 791\)](#) 단원을 참조하십시오.

이 함수의 전체 참조는 [aws_s3.table_import_from_s3 \(p. 799\)](#) 단원을 참조하십시오.

다음 예제에서는 Amazon S3 데이터를 가져오기를 수행할 때 다양한 종류의 파일을 지정하는 방법을 보여줍니다.

주제

- [사용자 지정 구분 기호를 사용하는 Amazon S3 파일 가져오기 \(p. 797\)](#)
- [Amazon S3 압축\(gzip\) 파일 가져오기 \(p. 798\)](#)
- [인코딩된 Amazon S3 파일 가져오기 \(p. 799\)](#)

사용자 지정 구분 기호를 사용하는 Amazon S3 파일 가져오기

다음 예제에서는 사용자 지정 구분 기호를 사용하는 파일을 가져오는 방법을 보여줍니다. 또한 [aws_s3.table_import_from_s3 \(p. 799\)](#) 함수의 column_list 파라미터를 사용해 데이터베이스 테이블에서 데이터를 배치할 곳을 제어하는 방법을 보여줍니다.

이 예에서는 다음 정보가 Amazon S3 파일의 파일프로 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.

```
psql=> CREATE TABLE test (a text, b text, c text, d text, e text);
CREATE TABLE
```

2. [aws_s3.table_import_from_s3 \(p. 799\)](#) 함수의 다음과 같은 형식을 사용해 Amazon S3 파일에서 데이터를 가져옵니다.

[aws_commons.create_s3_uri \(p. 801\)](#) 함수 호출 인라인을 `aws_s3.table_import_from_s3` 함수 호출에 포함하여 파일을 지정할 수 있습니다.

```
psql=> SELECT aws_s3.table_import_from_s3(
    'test',
    'a,b,d,e',
    'DELIMITER ''|''',
    aws_commons.create_s3_uri('sampleBucket', 'pipeDelimitedSampleFile', 'us-east-2')
);
```

이제 데이터는 다음 열의 테이블에 있습니다.

```
psql=> 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.

파일이 다음 Amazon S3 메타데이터를 포함하고 있는지 확인합니다.

- 키: Content-Encoding
- 값: gzip

이 값을 Amazon S3 메타데이터에 추가하는 것에 관한 자세한 내용은 [Amazon Simple Storage Service 콘솔 사용 설명서의 S3 객체에 메타데이터를 추가하려면 어떻게 해야 합니까?](#)를 참조하십시오.

아래와 같이 gzip 파일을 PostgreSQL DB 인스턴스용 Aurora PostgreSQL DB 클러스터로 가져옵니다.

```
psql=> CREATE TABLE test_gzip(id int, a text, b text, c text, d text);
CREATE TABLE
psql=> SELECT aws_s3.table_import_from_s3(
    'test_gzip', '', '(format csv)',
    'myS3Bucket', 'test-data.gz', 'us-east-2'
);
```

인코딩된 Amazon S3 파일 가져오기

다음 예에서는 Windows-1252 인코딩이 있는 Amazon S3에서 파일을 가져오는 방법을 보여줍니다.

```
psql=> SELECT aws_s3.table_import_from_s3(
    'test_table', '', 'encoding ''WIN1252'''',
    aws_commons.create_s3_uri('sampleBucket', 'SampleFile', 'us-east-2')
);
```

함수 참조

Functions

- [aws_s3.table_import_from_s3 \(p. 799\)](#)
- [aws_commons.create_s3_uri \(p. 801\)](#)
- [aws_commons.create_aws_credentials \(p. 802\)](#)
- [구문 \(p. 802\)](#)
- [파라미터 \(p. 802\)](#)

aws_s3.table_import_from_s3

Amazon S3 데이터를 Aurora PostgreSQL 테이블로 가져옵니다. aws_s3 확장은 aws_s3.table_import_from_s3 함수를 제공합니다.

구문

세 가지 필수 파라미터는 table_name, column_list 및 options입니다. 이 파라미터에서는 데이터베이스 테이블을 식별하고 데이터가 테이블로 복사되는 방식을 지정합니다.

다음 파라미터도 사용할 수 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. 문제 해결 예는 [사용자 지정 구분 기호를 사용하는 Amazon S3 파일 가져오기 \(p. 797\)](#) 단원을 참조하십시오.

옵션

PostgreSQL COPY 명령에 대한 인수를 포함하는 필수 텍스트 스트링입니다. 이 인수에서는 데이터가 PostgreSQL 테이블에 복사되는 방식을 지정합니다. 자세한 내용은 [PostgreSQL COPY 설명서](#)를 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` – 파일의 Amazon S3 경로입니다.
- `region` – 파일이 위치한 AWS 리전입니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 가용 영역 \(p. 3\)](#) 단원을 참조하십시오.

자격 증명

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 \(p. 802\)](#) 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

파일의 Amazon S3 경로가 포함된 텍스트 문자열입니다.

region

파일이 위치한 AWS 리전이 포함된 텍스트 문자열입니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 가용 영역 \(p. 3\)](#) 단원을 참조하십시오.

액세스 키

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_s3.table_import_from_s3 \(p. 799\)](#) 함수의 s3_info 파라미터에서 aws_commons.create_s3_uri 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

파일의 Amazon S3 경로가 포함된 필수 텍스트 문자열입니다.

region

파일이 위치한 AWS 리전이 포함된 필수 텍스트 문자열입니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 가용 영역 \(p. 3\)](#) 단원을 참조하십시오.

aws_commons.create_aws_credentials

aws_commons._aws_credentials_1 구조에서 액세스 키와 비밀 키를 설정합니다. aws_s3.table_import_from_s3 (p. 799) 함수의 credentials 파라미터에서 aws_commons.create_aws_credentials 함수의 결과를 사용합니다.

구문

```
aws_commons.create_aws_credentials(  
    access_key text,  
    secret_key text,  
    session_token text  
)
```

파라미터

액세스 키

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 NULL입니다.

session_token

Amazon S3 파일 가져오기에 사용할 세션 토큰이 포함된 텍스트 문자열(선택 사항)입니다. 기본값은 NULL입니다. 선택 사항인 session_token을 제공하는 경우 임시 자격 증명을 사용할 수 있습니다.

Amazon Aurora PostgreSQL 관리

다음 단원에서는 Amazon Aurora PostgreSQL DB 클러스터의 성능 관리 및 조정 방법에 대해서 설명합니다.

주제

- Aurora PostgreSQL DB 인스턴스 조정 (p. 802)
- Aurora PostgreSQL DB 인스턴스에 대한 최대 연결 (p. 802)
- 오류 삽입 쿼리를 사용하여 Amazon Aurora PostgreSQL 테스트 (p. 803)
- Aurora DB 클러스터를 위한 볼륨 상태 표시 (p. 806)

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 조정 (p. 212) 단원을 참조하십시오.

DB 클러스터의 DB 인스턴스마다 DB 인스턴스 클래스를 수정하여 Aurora PostgreSQL DB 클러스터의 규모를 조정할 수 있습니다. Aurora PostgreSQL은 Aurora에 최적화된 몇 가지 DB 인스턴스 클래스를 지원합니다. Aurora PostgreSQL에서 지원하는 DB 인스턴스 클래스의 세부 사양은 Aurora에 사용 가능한 모든 DB 인스턴스 클래스의 하드웨어 사양 (p. 31) 단원을 참조하십시오.

Aurora PostgreSQL DB 인스턴스에 대한 최대 연결

Aurora PostgreSQL DB 인스턴스에 대해 허용되는 최대 연결 수는 DB 인스턴스의 인스턴스 수준 파라미터 그룹의 max_connections 파라미터로 결정됩니다. 기본적으로 이 값은 다음 등식으로 설정됩니다.

`LEAST({DBInstanceClassMemory/9531392}, 5000).`

`max_connections` 파라미터를 이 수식으로 설정하면 허용되는 연결 수가 인스턴스 크기에 따라 조정됩니다. 예를 들어, DB 인스턴스 클래스가 db.r4.large이고 메모리가 15.25기비바이트(GiB)라고 가정합니다. 허용되는 최대 연결 수는 다음 수식과 같이 1600입니다.

`LEAST((15.25 * 1000000000) / 9531392), 5000) = 1600`

다음 표에는 Aurora PostgreSQL에서 사용 가능한 각 DB 인스턴스 클래스에 대한 `max_connections`의 결과 기본값이 나와 있습니다. 인스턴스를 메모리가 더 많은 DB 인스턴스까지 확장하거나, `max_connections` 파라미터의 값을 최대 262,143까지 설정하여 Aurora PostgreSQL DB 인스턴스의 최대 연결 수를 늘릴 수 있습니다.

인스턴스 클래스	<code>max_connections</code> 기본값
db.r4.large	1600
db.r4.xlarge	3200
db.r4.2xlarge	5000
db.r4.4xlarge	5000
db.r4.8xlarge	5000
db.r4.16xlarge	5000
db.r5.large	1600
db.r5.xlarge	3300
db.r5.2xlarge	5000
db.r5.4xlarge	5000
db.r5.12xlarge	5000
db.r5.24xlarge	5000
db.t3.medium	420

Aurora PostgreSQL에서 지원하는 DB 인스턴스 클래스 목록과 각 클래스의 메모리 양은 [Aurora에 사용 가능한 모든 DB 인스턴스 클래스의 하드웨어 사양 \(p. 31\)](#) 단원을 참조하십시오.

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 실행되며 오류 삽입 쿼리를 통해 다음 이벤트 중 하나의 발생에 대한 시뮬레이션을 예약하는 데 사용됩니다.

- 라이터 또는 리더 DB 인스턴스의 충돌
- Aurora 복제본의 실패
- 디스크 실패
- 디스크 정체

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 정보는 [Amazon Aurora 연결 관리 \(p. 9\)](#) 단원을 참조하십시오.

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 콘솔의 DB 클러스터에 대한 장애 조치 인스턴스 작업을 선택하거나 `failover-db-cluster` AWS CLI 명령 또는 `FailoverDBCluster` RDS API 작업을 사용하십시오.

구문

```
SELECT aurora_inject_crash ('instance' | 'dispatcher' | 'node');
```

옵션

이 오류 삽입 쿼리에는 다음 충돌 유형 중 하나가 사용됩니다.

- '`instance`' — Amazon Aurora 인스턴스용 PostgreSQL 호환 데이터베이스의 충돌이 시뮬레이션됩니다.
- '`dispatcher`' — Aurora DB 클러스터용 마스터 인스턴스에서 디스패처 충돌이 시뮬레이션됩니다. 디스패처가 Amazon Aurora DB 클러스터용 클러스터 볼륨에 업데이트 쓰기 작업을 합니다.
- '`node`' — PostgreSQL 호환 데이터베이스 및 Amazon Aurora 인스턴스용 디스패처의 충돌이 모두 시뮬레이션됩니다.

충돌 유형은 대소문자를 구분하지 않습니다.

Aurora 복제본 실패 테스트

`aurora_inject_replica_failure()` 오류 삽입 쿼리 함수를 사용하여 Aurora 복제본의 실패를 시뮬레이션할 수 있습니다.

Aurora 복제본 실패가 발생하면 지정된 시간 간격 동안 DB 클러스터의 특정 Aurora 복제본 또는 모든 Aurora 복제본에 대한 요청이 모두 차단됩니다. 시간 간격이 경과되면 영향 받는 Aurora 복제본이 기본 인스턴스와 자동으로 동기화됩니다.

구문

```
SELECT aurora_inject_replica_failure(
    percentage_of_failure,
    quantity,
    'replica_name'
);
```

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.

- **quantity** — Aurora 복제본 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 복제본. 단일 Aurora 복제본의 실패를 시뮬레이션하려면 Aurora 복제본의 이름을 지정합니다. DB 클러스터의 모든 Aurora 복제본에 대한 오류를 시뮬레이션하려면 빈 문자열을 지정합니다.

복제본 이름을 식별하려면 `aurora_replica_status()` 함수의 `server_id` 열을 참조하십시오. 다음 예를 참조하십시오.

```
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,  
    quantity  
) ;
```

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.
- **index** — 실패 이벤트를 시뮬레이션할 특정 논리 데이터 블록. 가용 논리 데이터 블록 또는 스토리지 노드 데이터의 범위를 초과하는 경우 지정할 수 있는 최대 인덱스 값을 알려주는 오류가 발생합니다. 이 오류를 방지하려면 [Aurora DB 클러스터를 위한 볼륨 상태 표시 \(p. 806\)](#) 단원을 참조하십시오.
- **is_disk** — 삽입 실패가 논리 블록에서 발생했는지 아니면 스토리지 노드에서 발생했는지 여부 표시. `true`를 지정하면 삽입 실패가 논리 블록에서 발생했음을 의미합니다. `false`를 지정하면 삽입 실패가 스토리지 노드에서 발생했음을 의미합니다.
- **quantity** — Aurora 복제본 실패 시뮬레이션 시간 길이. 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 실패를 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,  
    quantity,  
    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.
- **index** — 실패 이벤트 시뮬레이션에 사용할 특정 논리 데이터 블록 또는 스토리지 노드.
가용 논리 데이터 블록 또는 스토리지 데이터 노드의 범위를 초과하는 경우 지정할 수 있는 최대 인덱스 값을 알려주는 오류가 발생합니다. 이 오류를 방지하려면 [Aurora DB 클러스터를 위한 볼륨 상태 표시 \(p. 806\)](#) 단원을 참조하십시오.
- **is_disk** — 삽입 실패가 논리 블록에서 발생했는지 아니면 스토리지 노드에서 발생했는지 여부 표시. `true`를 지정하면 삽입 실패가 논리 블록에서 발생했음을 의미합니다. `false`를 지정하면 삽입 실패가 스토리지 노드에서 발생했음을 의미합니다.
- **quantity** — Aurora 복제본 실패 시뮬레이션 시간 길이. 간격은 초 단위입니다. 예를 들어, 값이 20이면 시뮬레이션이 20초 동안 실행됩니다.
- **minimum** 및 **maximum** — 최대 및 최소 정체 지연 시간(밀리초). 정체 상태로 표시된 디스크 세그먼트는 시뮬레이션 기간 동안 최소 및 최대 범위 내에서 임의의 시간 동안 지연됩니다.

Aurora 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 대해 자세히 알아보려면, AWS 데이터베이스 블로그의 [Aurora 스토리지 엔진 소개](#)를 참조하십시오.

`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 수

`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 삽

입 쿼리에 대한 자세한 내용은 [오류 삽입 쿼리를 사용하여 Amazon Aurora PostgreSQL 테스트 \(p. 803\)](#) 단원을 참조하십시오.

Note

`aurora_show_volume_status()` 함수는 Aurora PostgreSQL 버전 10.11에서 사용할 수 있습니다. Aurora PostgreSQL 버전에 대한 자세한 내용은 [Amazon Aurora PostgreSQL의 엔진 버전 \(p. 906\)](#) 단원을 참조하십시오.

구문

```
SELECT * FROM aurora_show_volume_status();
```

예제

다음 예제에서는 일반적인 결과를 보여 줍니다.

```
customer_database=> SELECT * FROM aurora_show_volume_status();
disks | nodes
-----+-----
 96 |    45
```

Amazon Aurora PostgreSQL를 사용한 복제

아래에서 Amazon Aurora PostgreSQL를 이용한 복제에 관한 정보를 확인할 수 있습니다. 복제를 모니터링하는 방법에 관한 내용도 포함되어 있습니다.

주제

- [Aurora 복제본 사용 \(p. 807\)](#)
- [Aurora PostgreSQL 복제 모니터링 \(p. 808\)](#)
- [Aurora에서 PostgreSQL 논리적 복제 사용 \(p. 808\)](#)

Aurora 복제본 사용

Aurora 복제본은 Aurora DB 클러스터의 독립 엔드포인트로서, 읽기 작업을 조정하고 가용성을 높이는 데 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 라이터 DB 인스턴스 및 Aurora 복제본에는 클러스터 볼륨의 데이터가 단 하나의 논리 볼륨으로 표시됩니다. Aurora 복제본에 대한 자세한 내용은 [Aurora 복제본 \(p. 55\)](#) 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. 이와는 대조적으로 PostgreSQL 읽기 전용 복제본은 마스터 DB 인스턴스부터 로컬 데이터 스토어에 이르는 모든 쓰기 작업을 단일 스레드에서 적용해야 합니다. 이 제한은 대용량 쓰기 트래픽을 지원하는 PostgreSQL 읽기 전용 복제본의 기능에 영향을 끼칠 수 있습니다.

Note

Amazon Aurora DB 클러스터의 라이터 DB 인스턴스를 재부팅하면 해당 DB 클러스터의 Aurora 복제본도 자동으로 재부팅됩니다. 자동 재부팅을 통해 DB 클러스터 전반에 걸친 읽기/쓰기 일관성을 보장하는 진입점이 다시 구성됩니다.

Aurora PostgreSQL 복제 모니터링

읽기 조정과 고가용성을 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 DB 클러스터 모니터링 \(p. 327\)](#) 단원을 참조하십시오.

Aurora에서 PostgreSQL 논리적 복제 사용

PostgreSQL 논리적 복제를 통해 데이터베이스의 복제 및 동기화 부분을 세밀하게 제어할 수 있습니다. 예를 들어 논리적 복제를 사용해 데이터베이스의 개별 테이블을 복제할 수 있습니다.

아래에서 PostgreSQL 논리적 복제와 Amazon Aurora를 이용한 작업 방법에 관한 정보를 확인할 수 있습니다. 논리적 복제의 PostgreSQL 구현에 대한 자세한 내용은 PostgreSQL 설명서에 있는 [논리적 복제 및 논리적 디코딩 개념](#) 단원을 참조하십시오.

Note

논리적 복제는 Aurora PostgreSQL 버전 2.2.0(PostgreSQL 10.6과 호환됨) 이상에서 사용할 수 있습니다.

아래에서 PostgreSQL 논리적 복제와 Amazon Aurora를 이용한 작업 방법에 관한 정보를 확인할 수 있습니다.

주제

- [논리적 복제 구성 \(p. 808\)](#)
- [데이터베이스 테이블의 논리적 복제 관련 예시 \(p. 809\)](#)
- [AWS Database Migration Service를 이용한 논리적 복제 \(p. 810\)](#)

논리적 복제 구성

논리적 복제를 사용하려면 먼저 클러스터 파라미터 그룹에 `rds.logical_replication` 파라미터를 설정해야 합니다. 그런 다음 게시자 및 구독자를 설정합니다.

논리적 복제에서는 게시 및 구독 모델을 사용합니다. 게시자와 구독자는 노드입니다. 게시는 하나 이상의 데이터베이스 테이블에서 생성된 변경 사항 집합입니다. 게시자에 대한 게시를 지정합니다. 구독은 다른 또 하나의 데이터베이스와 이 데이터베이스가 구독하는 1개 이상의 게시에 대한 연결을 정의합니다. 구독자에 대한 구독을 지정합니다. 게시 및 구독을 통해 게시자 및 구독자 데이터베이스가 서로 연결됩니다.

Note

PostgreSQL 데이터베이스에 대해 논리적 복제를 수행하려면 AWS 사용자 계정에 `rds_superuser` 역할이 필요합니다.

Aurora에서 PostgreSQL 논리적 복제를 활성화하려면

- DB 클러스터 파라미터 그룹 만들기 ([p. 172](#))에 설명된 대로 논리적 복제에 사용할 새로운 DB 클러스터 파라미터 그룹을 생성하십시오. 다음 설정을 사용합니다.
 - 파라미터 그룹 패밀리에서 `aurora-postgres10` 또는 그 이상 버전을 선택합니다.
 - [Type]에서 [DB Cluster Parameter Group]을 선택합니다.

- DB 클러스터 파라미터 그룹의 파라미터 수정 (p. 177)에 설명된 대로 클러스터 파라미터 그룹을 수정합니다. `rds.logical_replication` 정적 파라미터를 1로 설정합니다.

`rds.logical_replication` 파라미터 활성화는 DB 클러스터의 성능에 영향을 미칩니다.

논리적 복제를 위해 게시자를 구성하려면

- 게시자의 클러스터 파라미터 그룹을 다음과 같이 설정합니다.
 - 게시자에 대해 기존 Aurora PostgreSQL DB 클러스터를 사용하려면 엔진 버전이 10.6 이상이어야 합니다. 다음을 수행합니다.
 - 클러스터 파라미터 그룹을 수정하여 논리적 복제를 활성화할 때 생성한 그룹으로 설정합니다. Aurora PostgreSQL DB 클러스터 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.
 - 정적 파라미터 변경 사항이 적용되도록 DB 클러스터를 다시 시작합니다. 클러스터 파라미터 그룹에는 정적 파라미터인 `rds.logical_replication`에 대한 변경 사항이 포함됩니다.
 - 게시자에 대해 새로운 Aurora PostgreSQL DB 클러스터를 사용하려면 다음 설정을 사용해 DB 클러스터를 생성하십시오. Aurora PostgreSQL DB 클러스터 생성에 대한 자세한 내용은 [DB 클러스터 생성 \(p. 96\)](#) 단원을 참조하십시오.
 - Amazon Aurora 엔진을 선택하고 PostgreSQL-compatible(PostgreSQL 호환) 에디션을 선택합니다.
 - DB 엔진 버전에서 PostgreSQL 10.6 이상 버전과 호환되는 Aurora PostgreSQL 엔진을 선택합니다.
 - DB 클러스터 파라미터 그룹에서 논리적 복제를 활성화할 때 생성한 그룹을 선택합니다.
 - 게시자에 대한 보안 그룹의 인바운드 규칙을 수정하여 구독자의 연결을 허용합니다. 이 작업은 대개 보안 그룹에 구독자의 IP 주소를 포함하는 것으로 완료됩니다. 보안 그룹 수정에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 [VPC용 보안 그룹](#)을 참조하십시오.

데이터베이스 테이블의 논리적 복제 관련 예시

논리적 복제를 구현하려면 PostgreSQL 명령인 `CREATE PUBLICATION` 및 `CREATE SUBSCRIPTION`을 사용하십시오.

이 예제의 경우, 테이블 데이터는 게시자인 Aurora PostgreSQL 데이터베이스에서 구독자인 PostgreSQL 데이터베이스로 복제됩니다. RDS PostgreSQL 데이터베이스 또는 Aurora PostgreSQL 데이터베이스가 구독자 데이터베이스가 될 수 있습니다. PostgreSQL 논리적 복제를 사용하는 애플리케이션이 구독자가 될 수도 있습니다. 논리적 복제 메커니즘이 설정된 후에는 게시자에 대한 변경 사항이 발생할 때마다 지속적으로 구독자에게 전송됩니다.

이 예시에 대해 논리적 복제를 설정하려면 다음과 같이 하십시오.

- Aurora PostgreSQL DB 클러스터를 게시자로 구성합니다. 이를 위해서는 [논리적 복제 구성 \(p. 808\)](#)에 서 게시자를 구성할 때 설명한 것처럼 새로운 Aurora PostgreSQL DB 클러스터를 생성해야 합니다.
- 게시자 데이터베이스를 설정합니다.

예를 들어 게시자 데이터베이스에서 다음 SQL 문을 사용해 테이블을 생성합니다.

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

- 다음 SQL 문을 사용해 게시자 데이터베이스에 데이터를 삽입합니다.

```
INSERT INTO LogicalReplicationTest VALUES (generate_series(1,10000));
```

- 게시자 데이터베이스에서 다음 SQL 문을 사용해 테이블을 생성합니다.

```
CREATE PUBLICATION testpub FOR TABLE LogicalReplicationTest;
```

5. 구독자를 생성합니다. 구독자 데이터베이스는 다음 중 하나일 수 있습니다.

- Aurora PostgreSQL 데이터베이스 버전 2.2.0(PostgreSQL 10.6과 호환 가능) 이상.
- PostgreSQL DB 엔진 버전이 10.4 이상인 PostgreSQL 데이터베이스용 Amazon RDS.

이 예제에서는 구독자로 PostgreSQL 데이터베이스용 Amazon RDS를 생성합니다. DB 인스턴스 생성에 대한 자세한 내용은 Amazon RDS 사용 설명서의 [DB 인스턴스 생성](#)을 참조하십시오.

6. 구독자 데이터베이스를 설정합니다.

이 예시에서는 다음 SQL 문을 사용해 게시자에 대해 생성한 것과 같은 테이블을 생성합니다.

```
CREATE TABLE LogicalReplicationTest (a int PRIMARY KEY);
```

7. 양쪽 데이터베이스에서 다음 SQL 문에 따라 게시자 측 테이블에 데이터가 있지만 구독자 측 테이블에는 데이터가 없다는 것을 확인합니다.

```
SELECT count(*) FROM LogicalReplicationTest;
```

8. 구독자에 대해 구독을 생성합니다.

다음과 같이 구독자 데이터베이스의 SQL 문과 게시자 클러스터에서의 설정을 사용합니다.

- 호스트 – 게시자 클러스터의 라이터 DB 인스턴스입니다.
- 포트 – 라이터 DB 인스턴스가 수신 대기하는 포트입니다. PostgreSQL의 기본값은 5432입니다.
- dbname – 게시자 클러스터의 DB 이름입니다.

```
CREATE SUBSCRIPTION testsub CONNECTION
  'host=publisher-cluster-writer-endpoint port=5432 dbname=db-name user=user
  password=password'
  PUBLICATION testpub;
```

구독이 생성된 후 논리적 슬롯이 게시자 측에 생성됩니다.

9. 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.

AWS Database Migration Service를 이용한 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 상업용 데이터베이스로 마이그레이션하십시오. AWS DMS에 대한 자세한 내용은 [AWS Database Migration Service 사용 설명서](#)를 참조하십시오.

다음 예시에서는 게시자인 Aurora PostgreSQL 데이터베이스에서 논리적 복제를 설정한 다음 마이그레이션을 위해 AWS DMS를 사용하는 방법을 보여줍니다. 이 예시에서는 [데이터베이스 테이블의 논리적 복제 관련 예시 \(p. 809\)](#)에서 생성된 동일한 게시자 및 구독자를 사용합니다.

AWS DMS를 이용해 논리적 복제를 설정하려면 Amazon RDS에서 게시자 및 구독자에 대한 세부 정보를 알아야 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](#) 단원을 참조하십시오.

2. AWS Management 콘솔에 로그인하고 <https://console.aws.amazon.com/dms>에서 AWS DMS 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 포함됩니다.

- VPC에서 라이터 DB 인스턴스의 VPC와 동일한 VPC를 선택합니다.
- Replication Subnet Group(복제 서브넷 그룹)에서 라이터 DB 인스턴스의 서브넷 그룹과 동일한 서브넷 그룹을 선택합니다.
- 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 작업에 모든 세부 정보를 지정하는 것에 관한 자세한 내용은 AWS Database Migration Service 사용 설명서의 [AWS DMS 작업 사용](#) 단원을 참조하십시오.

작업이 생성되고 나면 AWS DMS에서 게시자에서 구독자로 데이터가 마이그레이션되기 시작합니다.

Amazon Aurora PostgreSQL를 다른 AWS 서비스와 통합

Aurora PostgreSQL DB 클러스터를 확장하여 AWS 클라우드에서 추가 기능을 사용할 수 있도록 Amazon Aurora가 다른 AWS 서비스와 통합되었습니다. 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(Performance Insights) 대시보드가 데이터베이스 부하를 시각화하여 대기 시간, SQL 문, 호스트 또는 사용자를 기준으로 부하를 필터링합니다.

성능 개선 도우미(Performance Insights)에 대한 자세한 내용은 [Amazon RDS 성능 개선 도우미 사용 \(p. 365\)](#) 단원을 참조하십시오.

- Aurora Auto Scaling을 사용하여 Aurora 복제본을 자동으로 추가 또는 제거합니다. 자세한 내용은 [Aurora 복제본에 Amazon Aurora Auto Scaling 사용 \(p. 253\)](#) 단원을 참조하십시오.
- Amazon CloudWatch Logs에 로그 데이터를 게시하도록 Aurora PostgreSQL DB 클러스터를 구성합니다. CloudWatch Logs는 로그 레코드에 내구력이 뛰어난 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 \(p. 847\)](#) 단원을 참조하십시오.

Aurora PostgreSQL DB 클러스터의 데이터를 Amazon S3으로 내보내기

Aurora PostgreSQL DB 클러스터의 데이터를 쿼리하여 Amazon S3 버킷에 저장된 파일로 직접 내보낼 수 있습니다. 이를 위해서는 Aurora PostgreSQL가 제공하는 `aws_s3` PostgreSQL 확장을 사용합니다.

Note

Aurora PostgreSQL에서 Amazon S3으로 데이터를 내보내려면 데이터베이스에서 다음 PostgreSQL 엔진 버전 중 하나를 실행해야 합니다.

- PostgreSQL 버전 10.11 이상
- PostgreSQL 버전 11.6 이상

Amazon S3를 이용한 데이터 저장에 대한 자세한 내용은 Amazon Simple Storage Service 시작 안내서의 [버킷 생성](#)을 참조하십시오.

주제

- [Amazon S3으로 데이터 내보내기 개요 \(p. 813\)](#)
- [내보낼 Amazon S3 파일 경로 지정 \(p. 813\)](#)
- [Amazon S3 버킷에 대한 액세스 권한 설정 \(p. 814\)](#)
- [aws_s3.export_query_to_s3 함수를 사용하여 쿼리 데이터 내보내기 \(p. 817\)](#)
- [함수 참조 \(p. 819\)](#)

Amazon S3으로 데이터 내보내기 개요

Aurora PostgreSQL 데이터베이스에 저장된 데이터를 Amazon S3 버킷으로 내보내려면 다음 절차를 따르십시오.

Aurora PostgreSQL 데이터를 S3으로 내보내려면

1. 필요한 PostgreSQL 확장을 설치합니다. 여기에는 aws_s3 및 aws_commons 확장이 포함됩니다. 이렇게 하려면 psql을 시작하고 다음 명령을 사용합니다.

```
CREATE EXTENSION IF NOT EXISTS aws_s3 CASCADE;
```

aws_s3 확장에서는 Amazon S3으로 데이터를 내보내는 데 사용하는 [aws_s3.export_query_to_s3 \(p. 819\)](#) 함수를 제공합니다. aws_commons 확장은 추가 헬퍼 함수를 제공하기 위해 포함되어 있습니다.

2. 데이터를 내보내는데 사용할 Amazon S3 파일 경로를 식별합니다. 이 프로세스에 대한 자세한 내용은 [내보낼 Amazon S3 파일 경로 지정 \(p. 813\)](#) 단원을 참조하십시오.
3. 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 클러스터에 추가합니다.

이 프로세스에 대한 자세한 내용은 [Amazon S3 버킷에 대한 액세스 권한 설정 \(p. 814\)](#) 단원을 참조하십시오.

4. 데이터를 가져올 데이터베이스 쿼리를 식별합니다. aws_s3.export_query_to_s3 함수를 호출하여 쿼리 데이터를 내보냅니다.

앞의 준비 작업을 완료한 후 [aws_s3.export_query_to_s3 \(p. 819\)](#) 함수를 사용하여 쿼리 결과를 Amazon S3으로 내보냅니다. 이 프로세스에 대한 자세한 내용은 [aws_s3.export_query_to_s3 함수를 사용하여 쿼리 데이터 내보내기 \(p. 817\)](#) 단원을 참조하십시오.

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 S3를 이용한 데이터 저장에 대한 자세한 내용은 Amazon Simple Storage Service 시작 안내서의 [버킷 생성 및 객체 보기](#)를 참조하십시오.

- 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 `_partXX`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 \(p. 3\)](#) 단원을 참조하십시오.

내보내는 데이터를 저장할 위치에 대한 Amazon S3 파일 정보를 보관하려면 `aws_commons.create_s3_uri (p. 821)` 함수를 사용하여 다음과 같이 `aws_commons._s3_uri_1` 복합 구조를 생성할 수 있습니다.

```
psql=> SELECT aws_commons.create_s3_uri(
    'sample-bucket',
    'samplefilepath',
    'us-west-2'
) AS s3_uri_1 \gset
```

나중에 이 `s3_uri_1` 값을 `aws_s3.export_query_to_s3 (p. 819)` 함수에 대한 호출의 파라미터로 제공합니다. 예제는 `aws_s3.export_query_to_s3` 함수를 사용하여 쿼리 데이터 내보내기 (p. 817)을 참조하십시오.

Amazon S3 버킷에 대한 액세스 권한 설정

데이터를 Amazon S3으로 내보내려면 Aurora PostgreSQL DB 클러스터에 파일이 저장될 Amazon S3 버킷에 액세스할 수 있는 권한을 부여합니다.

이렇게 하려면 다음 절차를 따르십시오.

IAM 역할을 통해 Amazon S3에 액세스할 수 있는 권한을 Aurora PostgreSQL DB 클러스터에 부여하려면

- IAM 정책을 생성합니다.

이 정책은 Aurora PostgreSQL DB 클러스터가 Amazon S3에 액세스할 수 있도록 허용하는 권한을 버킷 및 객체에 부여합니다.

이 정책을 생성하는 과정에서 다음 단계를 수행하십시오.

- 다음과 같은 필수 작업을 정책에 포함하여 Aurora PostgreSQL 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:::your-s3-bucket/*`입니다.

PostgreSQL용 Aurora PostgreSQL에 대한 IAM 정책 생성에 대한 자세한 내용은 [IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 \(p. 979\)](#) 단원을 참조하십시오. IAM 사용 설명서의 [자습서: 첫 번째 고객 관리형 정책 생성 및 연결](#)도 참조하십시오.

다음 AWS CLI 명령은 이 옵션으로 `rds-s3-export-policy`라는 IAM 정책을 만듭니다. `your-s3-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 사용](#)을 참조하십시오.

모든 리소스에 액세스할 수 있는 정책은 생성하지 않는 것이 좋습니다. 이러한 액세스 권한은 데이터 보안에 위협이 될 수 있습니다. "Resource": "*"를 사용하여 모든 리소스에 액세스할 수 있는 권한을 `S3:PutObject`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"  
            ],  
            "Effect": "Allow",  
            "Resource": [  
                "arn:aws:s3:::your-s3-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 역할을 만들어 [IAM 사용자에게 권한 위임](#) 단원을 참조하십시오.

다음 예제에서는 AWS CLI 명령을 사용해 `rds-s3-export-role`이라는 역할을 생성하는 방법을 보여 줍니다.

```
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"  
        }  
    ]  
}'
```

'

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 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 콘솔 또는 AWS CLI를 사용합니다.

콘솔

콘솔을 사용하여 PostgreSQL DB 클러스터에 대해 IAM 역할을 추가하려면

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/rds/>에서 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 Add IAM roles to this instance(이 인스턴스에 IAM 역할 추가)에서 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. `--feature-name` 옵션의 값에 대해 `s3Export`를 사용합니다.

Example

Linux, OS X, Unix의 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
```

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.export_query_to_s3 함수를 사용하여 쿼리 데이터 내보내기

aws_s3.export_query_to_s3 (p. 819) 함수를 호출하여 PostgreSQL 데이터를 Amazon S3으로 내보냅니다.

주제

- [사전 조건 \(p. 817\)](#)
- [aws_s3.export_query_to_s3 호출 \(p. 817\)](#)
- [사용자 지정 구분 기호를 사용하는 CSV 파일로 내보내기 \(p. 818\)](#)
- [인코딩을 사용하여 이진 파일로 내보내기 \(p. 818\)](#)
- [Amazon S3 액세스 문제 해결 \(p. 819\)](#)

사전 조건

aws_s3.export_query_to_s3 함수를 사용하기 전에 다음 사전 조건을 충족해야 합니다.

- [Amazon S3으로 데이터 내보내기 개요 \(p. 813\)](#)에 설명된 대로 필요한 PostgreSQL 확장을 설치합니다.
- [내보낼 Amazon S3 파일 경로 지정 \(p. 813\)](#)에 설명된 대로 데이터를 내보낼 Amazon S3 위치를 결정합니다.
- [Amazon S3 버킷에 대한 액세스 권한 설정 \(p. 814\)](#)에 설명된 대로 Amazon S3에 대한 내보내기 액세스 권한이 DB 클러스터에 있는지 확인합니다.

다음 예제에서는 sample_table이라는 데이터베이스 테이블을 사용합니다. 이 예제에서는 sample-bucket이라는 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.export_query_to_s3 호출

다음은 aws_s3.export_query_to_s3 (p. 819) 함수를 호출하는 기본 방법을 보여줍니다. 파라미터가 달라도 이러한 예의 결과는 동일합니다. sample_table 테이블의 모든 행은 sample-bucket이라는 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 정보를 포함하는 구조입니다. 다음과 같이 구조를 생성하는 aws_commons.create_s3_uri (p. 821) 함수를 사용합니다.

```
psql=> SELECT aws_commons.create_s3_uri(
```

```
'sample-bucket',
'samplefilepath',
'us-west-2'
) AS s3_uri_1 \gset
```

또는 aws_commons.create_s3_uri 함수 호출 인라인을 aws_s3.query_export_to_s3 함수 호출에 포함할 수 있습니다.

```
SELECT * from aws_s3.query_export_to_s3('select * from sample_table',
aws_commons.create_s3_uri('sample-bucket', 'samplefilepath', '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 _partXX가 추가됩니다. XX는 2, 3 등을 나타냅니다. 예를 들어 데이터 파일을 저장하는 경로를 다음과 같이 지정한다고 가정합니다.

```
s3-us-west-2://my-bucket/my-prefix
```

내보내기 시 세 개의 데이터 파일을 만들어야 하는 경우 Amazon S3 버킷에 다음 데이터 파일이 포함됩니다.

```
s3-us-west-2://my-bucket/my-prefix
s3-us-west-2://my-bucket/my-prefix_part2
s3-us-west-2://my-bucket/my-prefix_part3
```

이 함수에 대한 전체 참조 및 이 함수를 호출하는 추가 방법은 [aws_s3.export_query_to_s3 \(p. 819\)](#) 단원을 참조하십시오. Amazon S3에서 파일에 액세스하는 방법에 대한 자세한 내용은 Amazon Simple Storage Service 시작 안내서의 [객체 보기](#)를 참조하십시오.

사용자 지정 구분 기호를 사용하는 CSV 파일로 내보내기

다음 예제에서는 [aws_s3.export_query_to_s3 \(p. 819\)](#) 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.export_query_to_s3 \(p. 819\)](#) 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');
```

Amazon S3 액세스 문제 해결

Amazon S3으로 데이터를 내보내려고 할 때 연결 문제가 발생할 경우 다음 권장 사항을 참조하십시오.

- [Amazon Aurora 자격 증명 및 액세스 문제 해결 \(p. 993\)](#)
- [Amazon Simple Storage Service 개발자 가이드의 Amazon S3 문제 해결](#)
- [IAM 사용 설명서의 Amazon S3 및 IAM 문제 해결](#)

함수 참조

Functions

- [aws_s3.export_query_to_s3 \(p. 819\)](#)
- [aws_commons.create_s3_uri \(p. 821\)](#)

aws_s3.export_query_to_s3

PostgreSQL 쿼리 결과를 Amazon S3 버킷으로 내보냅니다. aws_s3 확장은 `aws_s3.export_query_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.export_query_to_s3` 함수 사용 예는 [aws_s3.export_query_to_s3 함수를 사용하여 쿼리 데이터 내보내기 \(p. 817\)](#) 단원을 참조하십시오.

구문

```
aws_s3.query_export_to_s3(  
    query text,  
    s3_info aws_commons._s3_uri_1,  
    options 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` – 파일의 Amazon S3 경로입니다.
- `region` – 버킷이 있는 AWS 리전입니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 가용 영역 \(p. 3\)](#) 단원을 참조하십시오.

현재 이 값은 내보내는 DB 클러스터의 AWS 리전과 동일해야 합니다. 기본값은 내보내는 DB 클러스터의 AWS 리전입니다.

`aws_commons._s3_uri_1` 복합 구조를 생성하려면 [aws_commons.create_s3_uri \(p. 821\)](#) 함수를 참조하십시오.

옵션

PostgreSQL COPY 명령에 대한 인수를 포함하는 선택적 텍스트 문자열입니다. 이러한 인수는 내보낼 때 데이터를 복사하는 방법을 지정합니다. 자세한 내용은 [PostgreSQL COPY 설명서](#)를 참조하십시오.

대체 입력 파라미터

s3_info 파라미터 대신에 확장 파라미터 세트를 사용하면 테스트에 도움이 됩니다. 다음은 aws_s3.export_query_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  
)
```

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

파일의 Amazon S3 경로가 포함된 필수 텍스트 문자열입니다.

region

버킷이 있는 AWS 리전을 포함하는 선택적 텍스트 문자열입니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 가용 영역 \(p. 3\)](#) 단원을 참조하십시오.

현재 이 값은 내보내는 DB 클러스터의 AWS 리전과 동일해야 합니다. 기본값은 내보내는 DB 클러스터의 AWS 리전입니다.

옵션

PostgreSQL COPY 명령에 대한 인수를 포함하는 선택적 텍스트 문자열입니다. 이러한 인수는 내보낼 때 데이터를 복사하는 방법을 지정합니다. 자세한 내용은 [PostgreSQL COPY 설명서](#)를 참조하십시오.

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.

예제

```
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-bucket', 'samplefilepath');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-bucket', 'samplefilepath', 'us-west-2');
psql=> SELECT * from aws_s3.query_export_to_s3('select * from sample_table', 'sample-bucket', 'samplefilepath', 'us-west-2', 'format text');
```

aws_commons.create_s3_uri

Amazon S3 파일 정보를 저장할 `aws_commons._s3_uri_1` 구조를 생성합니다.

`aws_s3.export_query_to_s3` (p. 819) 함수의 `s3_info` 파라미터에서 `aws_commons.create_s3_uri` 함수의 결과를 사용합니다. `aws_commons.create_s3_uri` 함수 사용 예는 [내보낼 Amazon S3 파일 경로 지정](#) (p. 813) 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

파일의 Amazon S3 경로가 포함된 필수 텍스트 문자열입니다.

region

파일이 위치한 AWS 리전이 포함된 필수 텍스트 문자열입니다. AWS 리전 이름 및 연결된 값의 목록은 [리전 및 가용 영역](#) (p. 3) 단원을 참조하십시오.

Aurora PostgreSQL용 쿼리 실행 계획 관리

PostgreSQL과 호환되는 Amazon Aurora용 쿼리 계획 관리를 사용하면 쿼리 실행 계획을 변경하는 방식과 변경해야 하는 경우를 제어할 수 있습니다. 쿼리 계획 관리의 주요 목표는 두 가지입니다.

- 데이터베이스 시스템 변경 시 계획 회귀 방지
- 쿼리 최적화 프로그램에서 새 계획을 사용할 수 있는 시점 제어

쿼리 최적화의 품질 및 일관성은 관계형 데이터베이스 관리 시스템(RDBMS)의 성능 및 안정성에 큰 영향을 미칩니다. 쿼리 최적화 프로그램은 특정 시점에 SQL 문에 대한 쿼리 실행 계획을 생성합니다. 조건이 바뀜에 따라 최적화 프로그램은 성능을 저하하는 다른 계획을 선택할 수도 있습니다. 다수의 변경 사항으로 인해 쿼리 최적화 프로그램이 다른 계획을 선택하게 되고 결국 성능 역행으로 이어질 수 있습니다. 이러한 변경 사항으로는 통계, 제한 사항, 환경 설정, 쿼리 파라미터 바인딩, 소프트웨어 업그레이드에 대한 변경 사항을 들 수 있습니다. 성능 역행은 고성능 애플리케이션에 중요한 문제입니다.

쿼리 계획 관리 기능을 사용하면 관리하려는 설명문 세트에 대한 실행 계획을 제어할 수 있습니다. 다음을 수행할 수 있습니다.

- 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.

주제

- [Aurora PostgreSQL용 쿼리 계획 관리 활성화 \(p. 822\)](#)
- [쿼리 계획 관리 업그레이드 \(p. 823\)](#)
- [쿼리 계획 관리의 기본 사항 \(p. 823\)](#)
- [쿼리 계획 관리에 대한 모범 사례 \(p. 826\)](#)
- [apg_plan_mgmt.dba_plans 보기에서 계획 검토 \(p. 827\)](#)
- [실행 계획 캡처 \(p. 829\)](#)
- [관리형 계획 사용 \(p. 831\)](#)
- [실행 계획 유지 관리 \(p. 834\)](#)
- [쿼리 계획 관리를 위한 파라미터 참조 \(p. 838\)](#)
- [쿼리 계획 관리를 위한 함수 참조 \(p. 842\)](#)

Aurora PostgreSQL용 쿼리 계획 관리 활성화

쿼리 계획 관리는 Amazon Aurora PostgreSQL 버전 2.1.0(PostgreSQL 10.5와 호환) 이상에서 사용할 수 있습니다.

쿼리 계획 관리를 활성화하려면

1. <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 쿼리 계획 관리 파라미터에 사용할 새 인스턴스 수준의 파라미터 그룹을 생성합니다. 자세한 내용은 [DB 파라미터 그룹 생성 \(p. 171\)](#) 단원을 참조하십시오. 쿼리 계획 관리를 사용하려는 DB 인스턴스와 새 파라미터 그룹을 연결합니다. 자세한 내용은 [DB 클러스터에서 DB 인스턴스 수정 \(p. 194\)](#) 단원을 참조하십시오.
3. 쿼리 계획 관리 파라미터에 사용할 새 클러스터 수준의 파라미터 그룹을 생성합니다. 자세한 내용은 [DB 클러스터 파라미터 그룹 만들기 \(p. 172\)](#) 단원을 참조하십시오. 쿼리 계획 관리를 사용할 DB 클러스터와 새 클러스터 수준 파라미터 그룹을 연결합니다. 자세한 내용은 [콘솔, CLI, API를 사용하여 DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.
4. 클러스터 수준의 파라미터 그룹을 열고 `rds.enable_plan_management` 파라미터를 1로 설정합니다. 자세한 내용은 [DB 클러스터 파라미터 그룹의 파라미터 수정 \(p. 177\)](#) 단원을 참조하십시오.
5. DB 인스턴스를 다시 시작하여 이 새로운 설정을 활성화합니다.

6. psql과 같은 SQL 클라이언트를 사용하여 DB 인스턴스에 연결합니다.
7. DB 인스턴스용 apg_plan_mgmt 확장을 생성합니다. 다음은 그 한 예입니다.

```
psql my-database
my-database=> CREATE EXTENSION apg_plan_mgmt;
```

template1 기본 데이터베이스에서 apg_plan_mgmt 확장을 생성할 경우 생성하는 각각의 새 데이터 베이스에서 쿼리 계획 관리 확장을 사용할 수 있습니다.

Note

apg_plan_mgmt 확장을 생성하려면 rds_superuser 역할이 필요합니다. apg_plan_mgmt 확장을 생성하면 apg_plan_mgmt 역할이 생성됩니다. 사용자가 apg_plan_mgmt 확장을 관리하려면 apg_plan_mgmt 역할을 부여받아야 합니다.

쿼리 계획 관리 업그레이드

쿼리 계획 관리의 최신 버전은 2.0입니다. 이전 버전의 쿼리 계획 관리를 설치한 경우 버전 2.0으로 업그레이드할 것을 강력하게 권장합니다. 버전에 대한 자세한 내용은 [Amazon Aurora PostgreSQL의 확장 버전 \(p. 926\)](#) 단원을 참조하십시오.

업그레이드하려면 클러스터 또는 DB 인스턴스 수준에서 다음 명령을 실행하십시오.

```
ALTER EXTENSION apg_plan_mgmt UPDATE TO '2.0';
SELECT apg_plan_mgmt.validate_plans('update_plan_hash');
SELECT apg_plan_mgmt.reload();
```

쿼리 계획 관리의 기본 사항

설명문의 복잡성과 관계없이 쿼리 계획 관리를 사용하여 SELECT, INSERT, UPDATE 또는 DELETE 문을 관리할 수 있습니다. 준비된 SQL 문, 동적 SQL 문, 포함된 SQL 문 및 즉시 모드 SQL 문이 모두 지원됩니다. 파티셔닝된 테이블, 상속, 행 수준의 보안 및 recursive 공통 테이블 표현식(CTE)을 비롯하여 모든 PostgreSQL 언어 기능을 사용할 수 있습니다.

주제

- [수동 계획 캡처 수행 \(p. 823\)](#)
- [캡처된 계획 보기 \(p. 824\)](#)
- [관리형 설명문 및 SQL 해시를 이용한 작업 \(p. 824\)](#)
- [자동 계획 캡처 작업 \(p. 825\)](#)
- [계획 검증 \(p. 825\)](#)
- [성능을 개선하는 새 계획 승인 \(p. 825\)](#)
- [계획 삭제 \(p. 826\)](#)

수동 계획 캡처 수행

특정 설명문에 대한 계획을 캡처하려면 다음 예와 같이 수동 캡처 모드를 사용합니다.

```
/* Turn on manual capture */
SET apg_plan_mgmt.capture_plan_baselines = manual;
EXPLAIN SELECT COUNT(*) from pg_class;           -- capture the plan baseline
```

```
SET apg_plan_mgmt.capture_plan_baselines = off;      -- turn off capture
SET apg_plan_mgmt.use_plan_baselines =      true;    -- turn on plan usage
```

SELECT, INSERT, UPDATE 또는 DELETE 문을 실행하거나, 위와 같이 EXPLAIN 문을 포함할 수 있습니다. EXPLAIN을 사용하여 설명문 실행에 따른 오버헤드나 잠재적인 부작용 없이 계획을 캡처할 수 있습니다. 수동 캡처에 대한 자세한 내용은 [특정 SQL 문에 대해 계획을 수동으로 캡처 \(p. 830\)](#) 단원을 참조하십시오.

캡처된 계획 보기

앞의 예시에서 EXPLAIN SELECT가 실행될 때 최적화 프로그램이 계획을 저장합니다. 이를 위해 최적화 프로그램은 apg_plan_mgmt.dba_plans 보기에 행을 삽입하고 자율적인 트랜잭션에서 계획을 커밋합니다. apg_plan_mgmt 역할이 부여된 경우 apg_plan_mgmt.dba_plans 보기의 내용을 볼 수 있습니다. 다음 쿼리는 dba_plans 보기의 중요한 몇 가지 열을 표시합니다.

```
SELECT sql_hash, plan_hash, status, enabled, plan_outline, sql_text::varchar(40)
FROM apg_plan_mgmt.dba_plans
ORDER BY sql_text, plan_created;
```

표시된 각 행은 관리형 계획을 나타냅니다. 앞의 예는 다음 정보를 보여 줍니다.

- **sql_hash** – 계획 관련 관리형 설명문의 ID입니다.
- **plan_hash** – 관리형 계획의 ID입니다.
- **status** – 계획의 상태입니다. 최적화 프로그램은 승인된 계획을 실행할 수 있습니다.
- **enabled** – 계획이 활성화되어 사용할 수 있는지 아니면 비활성화되어 사용할 수 없는지를 나타내는 값입니다.
- **plan_outline** – 관리형 계획의 세부 정보입니다.

apg_plan_mgmt.dba_plans 보기에 대한 자세한 내용은 [apg_plan_mgmt.dba_plans 보기에서 계획 검토 \(p. 827\)](#) 단원을 참조하십시오.

관리형 설명문 및 SQL 해시를 이용한 작업

관리형 설명문은 쿼리 계획 관리 상태에서 최적화 프로그램에서 캡처한 SQL 문입니다. 수동 또는 자동 캡처를 사용하여 관리형 설명문으로 캡처할 SQL 문을 지정합니다.

- 수동 캡처의 경우 이전 예에서처럼 최적화 프로그램에 특정 설명문을 제공합니다.
- 자동 캡처의 경우 최적화 프로그램은 여러 번 실행되는 설명문에 대한 계획을 캡처합니다. 자동 캡처는 뒤에 소개하는 예에 나와 있습니다.

apg_plan_mgmt.dba_plans 보기에서는 SQL 해시 값을 사용하여 관리형 설명문을 식별할 수 있습니다. SQL 해시는 차이(예: 리터럴 값)를 없앤 SQL 문의 정규화된 표현에서 계산됩니다. 정규화 사용이란 여러 SQL 문이 리터럴 또는 파라미터 값만 다를 경우 apg_plan_mgmt.dba_plans 보기에서 동일 SQL 해시로 표현됨을 뜻합니다. 따라서 각 계획이 서로 다른 조건에서 최적 상태인 동일한 SQL 해시에 대해 여러 계획이 있을 수 있습니다.

최적화 프로그램은 SQL 문을 처리할 때 다음 규칙을 사용하여 정규화된 SQL 문을 생성합니다.

- **선행 블록 설명을 제거합니다.**
- EXPLAIN 키워드 및 EXPLAIN 옵션을 제거합니다(있는 경우).
- **후행 공백을 제거합니다.**
- **모든 리터럴을 제거합니다.**

- 가독성을 위해 공백과 대/소문자를 유지합니다.

예를 들면 다음 설명문을 사용합니다.

```
/*Leading comment*/ EXPLAIN SELECT /* Query 1 */ * FROM t WHERE x > 7 AND y = 1;
```

최적화 프로그램은 다음과 같이 이 설명문을 정규화합니다.

```
SELECT /* Query 1 */ * FROM t WHERE x > CONST AND y = CONST;
```

자동 계획 캡처 작업

애플리케이션의 모든 SQL 문에 대한 계획을 캡처하려는 경우 또는 수동 캡처를 사용할 수 없는 경우' 자동 계획 캡처를 사용합니다. 자동 계획 캡처를 사용하는 경우 기본적으로 최적화 프로그램은 두 번 이상 실행되는 설명문에 대해 계획을 캡처합니다. 자동 계획 캡처를 사용하려면 다음을 수행합니다.

1. DB 인스턴스에 대한 파라미터 그룹에서 `apg_plan_mgmt.capture_plan_baselines`을 `automatic`으로 설정하여 자동 계획 캡처를开启了. 자세한 내용은 [DB 파라미터 그룹의 파라미터 수정 \(p. 174\)](#) 단원을 참조하십시오.
2. DB 인스턴스를 다시 시작합니다.

애플리케이션이 실행될 때 최적화 프로그램은 두 번 이상 실행되는 모든 설명문에 대해 계획을 캡처합니다. 최적화 프로그램은 관리형 설명문의 캡처된 첫 번째 계획의 상태를 항상 `approved`로 설정합니다. 승인된 계획의 관리형 설명문 세트를 계획 기준이라고 합니다.

애플리케이션이 계속 실행됨에 따라 최적화 프로그램은 관리형 설명문에 대한 추가 계획을 찾을 수 있습니다. 최적화 프로그램은 캡처한 추가 계획을 `Unapproved` 상태로 설정합니다.

관리형 설명문에 대한 캡처된 모든 계획 세트를 계획 기록이라고 합니다. 나중에 `apg_plan_mgmt.evolve_plan_baselines` 함수 또는 `apg_plan_mgmt.set_plan_status` 함수를 사용하여 `Unapproved` 계획이 잘 수행되는지 여부를 판단하고 이러한 계획을 `Approved`, `Rejected` 또는 `Preferred`로 변경할 수 있습니다.

자동 계획 캡처를 꺼려면 DB 인스턴스에 대한 파라미터 그룹에서 `apg_plan_mgmt.capture_plan_baselines`을 `off`으로 설정하십시오. 그런 다음 설정이 적용되도록 데 이터베이스를 다시 시작합니다.

계획 캡처에 대한 자세한 내용은 [실행 계획 캡처 \(p. 829\)](#) 단원을 참조하십시오.

계획 검증

관리형 계획은 계획이 종속된 객체(예: 인덱스)가 제거되면 무효("기한 경과") 상태가 될 수 있습니다. 기한 경과 상태인 모든 계획을 찾아서 삭제하려면 `apg_plan_mgmt.validate_plans` 함수를 사용합니다.

```
SELECT apg_plan_mgmt.validate_plans('delete');
```

자세한 내용은 [계획 검증 \(p. 835\)](#) 단원을 참조하십시오.

성능을 개선하는 새 계획 승인

관리형 계획을 사용하는 경우 최적화 프로그램에서 검색된 더 새롭고 저렴한 비용의 계획이 계획 기준에 이미 포함된 최소 비용 계획보다 더 빠른지 확인할 수 있습니다. 성능 비교를 수행하고 필요에 따라 더 빠른 계획을 승인하려면 `apg_plan_mgmt.evolve_plan_baselines` 함수를 호출합니다.

다음 예제에서는 계획 기준의 최소 비용 계획보다 최소 10% 더 빠르며 활성화되어 있는 미승인 계획을 모두 자동으로 승인합니다.

```
SELECT apg_plan_mgmt.evolve_plan_baselines(
    sql_hash,
    plan_hash,
    1.1,
    'approve'
)
FROM apg_plan_mgmt.dba_plans
WHERE status = 'Unapproved' AND enabled = true;
```

`apg_plan_mgmt.evolve_plan_baselines` 함수는 실행 시 성능 통계를 수집하고 `planning_time_ms`, `execution_time_ms`, `cardinality_error`, `total_time_benefit_ms` 및 `execution_time_benefit_ms` 열의 `apg_plan_mgmt.dba_plans` 보기에 이러한 통계를 저장합니다. 또한 `apg_plan_mgmt.evolve_plan_baselines` 함수는 `last_verified` 또는 `last_validated` timestamps 열을 업데이트하므로 성능 통계가 수집된 가장 최근 시간을 볼 수 있습니다.

```
SELECT sql_hash, plan_hash, status, last_verified, sql_text::varchar(40)
FROM apg_plan_mgmt.dba_plans
ORDER BY last_verified DESC; -- value updated by evolve_plan_baselines()
```

계획 확인에 대한 자세한 내용은 [계획 성능 평가 \(p. 834\)](#) 단원을 참조하십시오.

계획 삭제

최적화 프로그램은 계획을 계획 보존 기간 동안 최소 비용 계획으로 선택했거나 실행하지 않은 경우 해당 계획을 자동으로 삭제합니다. 기본적으로 계획 보존 기간은 32일입니다. 계획 보존 기간을 변경하려면 `apg_plan_mgmt.plan_retention_period` 파라미터를 설정합니다.

또한 `apg_plan_mgmt.dba_plans` 보기의 내용을 검토하고 `apg_plan_mgmt.delete_plan` 함수를 사용하여 원하지 않는 계획을 모두 삭제할 수도 있습니다. 자세한 내용은 [계획 삭제 \(p. 837\)](#) 단원을 참조하십시오.

쿼리 계획 관리에 대한 모범 사례

사전 대비적이거나 사후 대응적인 계획 관리 스타일을 사용할 수 있습니다. 이러한 계획 관리 스타일은 새 계획의 사용이 승인되는 방식과 승인되는 경우를 대조합니다.

성능 역행 문제를 방지하기 위한 사전 대비형 계획 관리

사전 대비형 계획 관리를 통해 새 계획이 더 빠름을 확인한 후 해당 계획을 수동으로 승인합니다. 계획 성능 역행을 방지하려면 이 작업을 수행합니다. 사전 대비형 계획 관리를 진행하려면 다음 단계를 수행합니다.

- 개발 환경에서 성능 또는 시스템 처리량에 가장 큰 영향을 미치는 SQL 문을 식별합니다. 그런 다음 [특정 SQL 문에 대해 계획을 수동으로 캡처 \(p. 830\)](#) 및 [계획 자동 캡처 \(p. 830\)](#)에 설명된 대로 이러한 설명 문에 대해 계획을 캡처합니다.
- 개발 환경에서 캡처한 계획을 내보내고 프로덕션 환경으로 가져옵니다. 자세한 내용은 [계획 내보내기/가져오기 \(p. 837\)](#) 단원을 참조하십시오.
- 프로덕션 단계에서 애플리케이션을 실행하고 승인된 관리형 계획 사용을 적용합니다. 자세한 내용은 [관리형 계획 사용 \(p. 831\)](#) 단원을 참조하십시오. 애플리케이션이 실행되는 동안 최적화 프로그램에서 검색되는 새 계획도 추가합니다. 자세한 내용은 [계획 자동 캡처 \(p. 830\)](#) 단원을 참조하십시오.
- 미승인 계획을 분석하고 잘 수행되는 계획을 승인합니다. 자세한 내용은 [계획 성능 평가 \(p. 834\)](#) 단원을 참조하십시오.

5. 애플리케이션이 계속 실행되는 동안 최적화 프로그램은 해당되는 경우 새 계획을 사용하기 시작합니다.

성능 역행을 찾아내어 복구하기 위한 사후 대응형 계획 관리

사후 대응형 계획 관리를 사용하면 애플리케이션이 실행될 때 애플리케이션을 모니터링하여 성능 역행을 발생시키는 계획을 찾아냅니다. 역행을 찾아내면 좋지 않은 계획을 수동으로 거부하거나 수정합니다. 사후 대응형 계획 관리를 진행하려면 다음 단계를 수행합니다.

1. 애플리케이션이 실행되는 동안 관리형 계획의 사용을 적용하고 새로 검색된 계획을 미승인 계획으로 자동 추가합니다. 자세한 내용은 [관리형 계획 사용 \(p. 831\)](#) 및 [계획 자동 캡처 \(p. 830\)](#) 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 \(p. 835\)](#) 단원을 참조하십시오.

경우에 따라 좋지 않은 계획을 거부, 비활성화 또는 삭제하기보다는 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 \(p. 835\)](#) 단원을 참조하십시오.

apg_plan_mgmt.dba_plans 보기에서 계획 검토

퀴리 계획 관리는 `apg_plan_mgmt.dba_plans`라고 하는 데이터베이스 관리자(DBA)를 위한 새 SQL 보기 를 제공합니다. DB 인스턴스의 각 데이터베이스에는 고유한 `apg_plan_mgmt.dba_plans` 보기기가 있습니다.

이 보기에는 모든 관리형 설명문에 대한 계획 기록이 포함되어 있습니다. 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 참조하십시오.

Note

`apg_plan_mgmt.dba_plans` 보기에 대한 액세스는 `apg_plan_mgmt` 역할을 보유한 사용자로 제한됩니다.

관리형 계획 나열

관리형 계획을 나열하려면 `apg_plan_mgmt.dba_plans` 보기에서 SELECT 문을 사용합니다. 다음 예제는 `status`(승인 및 미승인 계획을 나타냄)와 같은 `dba_plans` 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)
```

apg_plan_mgmt.dba_plans 보기에 대한 참조

`apg_plan_mgmt.dba_plans` 보기의 계획 정보 열에는 다음이 포함됩니다.

dba_plans 열	설명
<code>cardinality_error</code>	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 (p. 843) 함수에 의해 작성됩니다.
<code>compatibility_level</code>	Aurora PostgreSQL 최적화 프로그램의 기능 수준입니다.
<code>created_by</code>	계획을 생성한 인증된 사용자(session_user)입니다.
<code>enabled</code>	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 (p. 845) 함수를 사용합니다.
<code>environment_variable</code>	최적화 프로그램이 계획이 캡처될 때 재정의한 PostgreSQL Grand Unified Configuration(GUC) 파라미터와 값입니다.
<code>estimated_startup_cost</code>	최적화 프로그램에서 테이블의 행을 전송하기 전 최적화 프로그램 설정 예상 비용입니다.
<code>estimated_total_cost</code>	최종 테이블 행을 전송하는 데 드는 최적화 프로그램 예상 비용입니다.
<code>execution_time_benefit</code>	최적화 시 실행 시간 편익(밀리초)입니다. 이 열은 apg_plan_mgmt.evolve_plan_baselines (p. 843) 함수에 의해 작성됩니다.
<code>execution_time_ms</code>	계획이 실행될 예상 시간(밀리초)입니다. 이 열은 apg_plan_mgmt.evolve_plan_baselines (p. 843) 함수에 의해 작성됩니다.
<code>has_side_effects</code>	SQL 문이 데이터 조작 언어(DML) 문이거나 VOLATILE 함수를 포함하는 SELECT 문임을 나타내는 값입니다.
<code>last_used</code>	이 값은 계획이 실행될 때마다 또는 계획이 쿼리 최적화 프로그램의 최소 비용 계획일 경우 현재 날짜로 업데이트됩니다. 이 값은 공유 메모리에 저장되고 정기적으로 디스크로 폴러시됩니다. 최신 값을 가져오려면 <code>last_used</code> 값을 읽는 대신 <code>apg_plan_mgmt.plan_last_used(sql_hash, plan_hash)</code> 함수를 호출하여 공유 메모리에서 날짜를 읽습니다. 자세한 내용은 apg_plan_mgmt.plan_retention_period (p. 841) 파라미터를 참조하십시오.
<code>last_validated</code>	<code>apg_plan_mgmt.validate_plans (p. 846)</code> 함수 또는 <code>apg_plan_mgmt.evolve_plan_baselines (p. 843)</code> 함수로 계획을 다시 생성할 수 있음이 확인된 최근 날짜 및 시간입니다.
<code>last_verified</code>	계획이 apg_plan_mgmt.evolve_plan_baselines (p. 843) 함수에 의해 지정된 파라미터에 대한 최적 수행 계획인 것으로 확인된 최근 날짜 및 시간입니다.
<code>origin</code>	<code>apg_plan_mgmt.capture_plan_baselines (p. 839)</code> 파라미터를 사용하여 계획을 캡처한 방법입니다. 유효 값에는 다음이 포함됩니다. M – 수동 계획 캡처 기능을 사용하여 계획을 캡처했습니다. A – 자동 계획 캡처 기능을 사용하여 계획을 캡처했습니다.
<code>param_list</code>	준비된 설명문인 경우 문으로 전달된 파라미터 값입니다.
<code>plan_created</code>	계획이 생성된 날짜 및 시간입니다.
<code>plan_hash</code>	계획 식별자입니다. <code>plan_hash</code> 및 <code>sql_hash</code> 의 조합은 특정 계획을 고유하게 식별합니다.

dba_plans 열	설명
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 (p. 843) 함수에 의해 작성됩니다.
queryId	pg_stat_statements 확장을 통해 계산된 설명문 해시입니다. 이 식별자는 객체 식별자(OID)에 종속되므로 안정형 또는 데이터베이스 독립형 식별자가 아닙니다.
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. <ul style="list-style-type: none"> 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 (p. 843) 함수를 사용합니다. Unapproved – 사용할 수 있는지 확인되지 않은 캡처된 계획입니다. 자세한 내용은 계획 성능 평가 (p. 834) 단원을 참조하십시오. Rejected – 최적화 프로그램에서 사용할 수 없는 계획입니다. 자세한 내용은 느린 계획 거부 또는 비활성화 (p. 835) 단원을 참조하십시오. 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 Preferred로 설정하십시오. 최적화 프로그램의 최소 비용이 Approved 계획이 아니면 Preferred 계획이 Approved 계획에 앞서 선택됩니다. 계획을 Preferred로 재설정하려면 apg_plan_mgmt.set_plan_status (p. 846) 함수를 사용합니다.
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	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 (p. 843) 함수에 의해 작성됩니다.

실행 계획 캡처

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 느리거나 변동성이 가장 큰) 계획을 캡처할 수 있습니다.

계획을 캡처하면 최적화 프로그램이 관리형 설명문에 대해 캡처된 첫 번째 계획의 상태를 approved로 설정합니다. 최적화 프로그램은 관리형 설명문에 대해 캡처된 추가 계획의 상태를 항상 unapproved로 설정합니다. 그러나 두 가지 이상의 계획이 approved 상태로 저장되는 경우가 가끔 있을 수 있습니다. 이런 일은 설

명문에 대해 여러 계획이 병렬적으로 생성될 때와 설명문의 첫 번째 계획이 커밋되기 전에 발생할 수 있습니다.

`dba_plans` 보기에서 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 다시 시작해야 합니다. 자세한 내용은 [apg_plan_mgmt.max_plans \(p. 839\)](#) 파라미터를 참조하십시오.

주제

- [특정 SQL 문에 대해 계획을 수동으로 캡처 \(p. 830\)](#)
- [계획 자동 캡처 \(p. 830\)](#)

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` 보기에서 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 \(p. 835\)](#) 단원을 참조하십시오. unapproved 및 approved 계획의 성능을 비교하고 이러한 계획을 승인, 거부 또는 삭제하려면 [계획 성능 평가 \(p. 834\)](#) 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 `psql`로 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`을 `automatic`으로 설정하여 자동 계획 캡처를 켭니다. 자세한 내용은 [DB 파라미터 그룹의 파라미터 설정 \(p. 174\)](#) 단원을 참조하십시오.
2. DB 인스턴스를 다시 시작합니다.
3. 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.

일부 파라미터를 수정하여 처리량에 가장 큰 영향을 미치거나 가장 오랫동안 실행되거나 변동성이 가장 큰 설명문에 대해 계획을 캡처할 수 있습니다. 하지만 이 확장된 자동 계획 캡처 모드에서는 상당한 성능 오버헤드가 발생합니다.

자동 계획 캡처를 고려면

- 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 \(p. 834\)](#) 단원을 참조하십시오.

관리형 계획 사용

최적화 프로그램에서 관리형 설명문에 대해 캡처한 계획을 사용하려면 파라미터 `apg_plan_mgmt.use_plan_baselines`을 `true`로 설정합니다. 다음은 로컬 인스턴스 예제입니다.

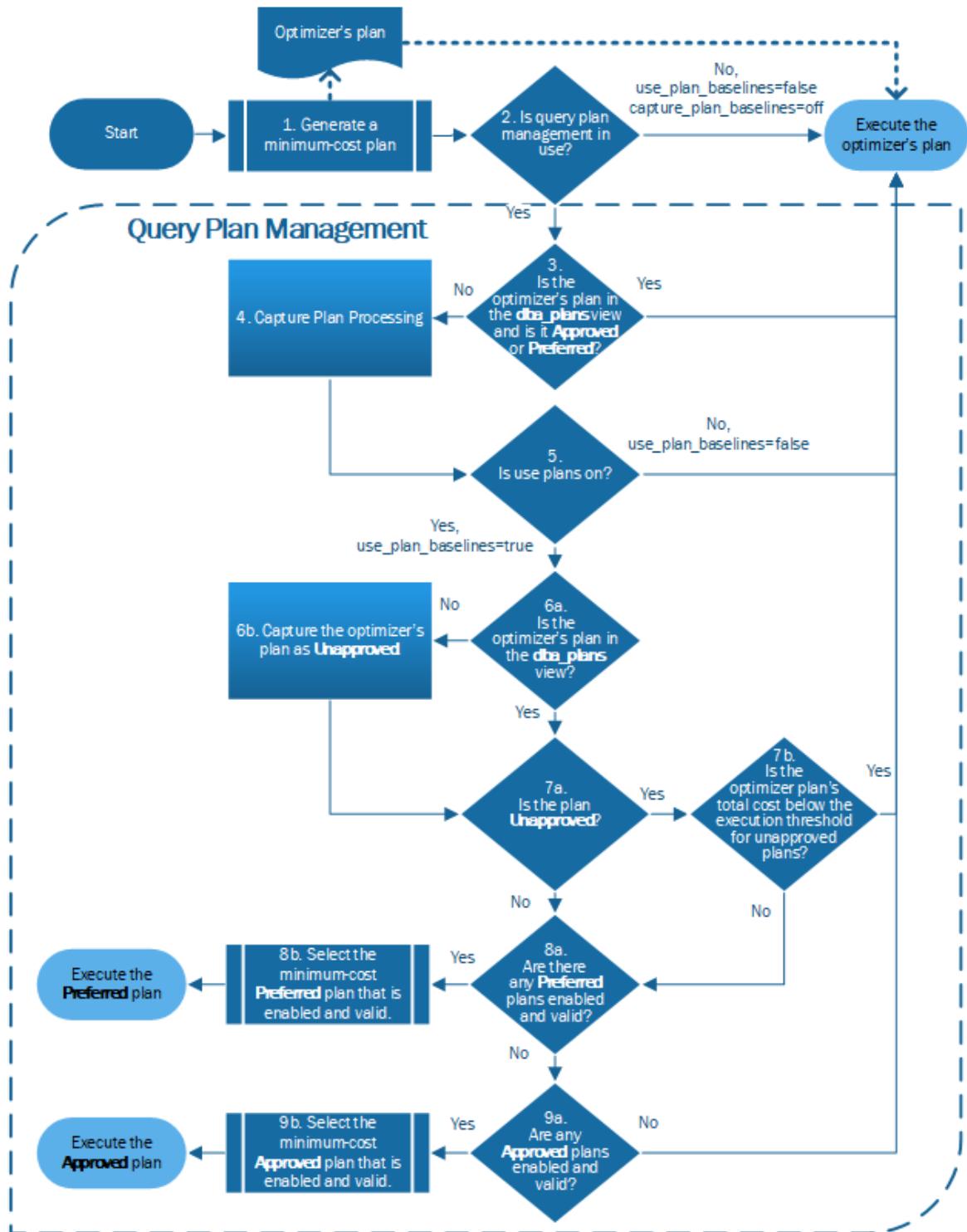
```
SET apg_plan_mgmt.use_plan_baselines = true;
```

애플리케이션이 실행되는 동안 이 설정을 사용하면 최적화 프로그램이 각각의 관리형 설명문에 대해 유효하고 활성화된 최소 비용, 기본 또는 승인된 계획을 사용합니다.

최적화 프로그램에서 실행할 계획을 선택하는 방식

실행 계획의 비용은 최적화 프로그램에서 서로 다른 계획을 비교할 때 계산하는 추정치입니다. 최적화 프로그램 비용은 계획에서 사용하는 CPU 및 I/O 작업을 포함하는 여러 요인에 따라 달라집니다. PostgreSQL 쿼리 플래너 비용에 대한 자세한 내용은 [PostgreSQL documentation on query planning](#)을 참조하십시오.

다음 흐름도는 쿼리 계획 관리 최적화 프로그램에서 실행할 계획을 선택하는 방식을 보여줍니다.



이 흐름은 다음과 같습니다.

- 최적화 프로그램은 모든 SQL 문을 처리할 때 최소 비용 계획을 생성합니다.
- 최적화 프로그램에서 쿼리 계획 관리 기능을 사용하지 않고 생성된 계획을 단순히 실행하기만 합니다. 다음 파라미터 설정 중 하나 또는 두 개를 설정하는 경우 최적화 프로그램이 쿼리 계획을 사용합니다.

- `apg_plan_mgmt.capture_plan_baselines`를 `manual` 또는 `automatic`으로
 - `apg_plan_mgmt.use_plan_baselines true`
3. 다음이 모두 `true`일 경우 최적화 프로그램이 생성된 계획을 즉시 실행합니다.
- 최적화 프로그램의 계획이 SQL 문의 `apg_plan_mgmt.dba_plans` 보기에 이미 있습니다.
 - 계획의 상태가 `approved` 또는 `preferred`입니다.
4. `apg_plan_mgmt.capture_plan_baselines` 파라미터가 `manual` 또는 `automatic`인 경우 최적화 프로그램이 계획 캡처 처리를 진행합니다.
- 최적화 프로그램의 계획 캡처 방식에 대한 자세한 내용은 [실행 계획 캡처 \(p. 829\)](#) 단원을 참조하십시오.
5. `apg_plan_mgmt.use_plan_baselines`가 `false`이면 최적화 프로그램이 생성된 계획을 실행합니다.
6. 최적화 프로그램의 계획이 `apg_plan_mgmt.dba_plans` 보기에 없으면 최적화 프로그램이 해당 계획을 새 `unapproved` 계획으로 캡처합니다.
7. 다음이 모두 `true`일 경우 최적화 프로그램이 생성된 계획을 실행합니다.
- 최적화 프로그램의 계획이 `rejected` 또는 `disabled` 계획이 아닙니다.
 - 계획의 총 비용이 미승인 실행 계획 임계값보다 작습니다.
- 최적화 프로그램이 거부된 상태의 계획이나 비활성화된 계획을 실행하지 않습니다. 대부분의 경우 최적화 프로그램이 미승인 계획을 실행하지 않습니다. 하지만 `apg_plan_mgmt.unapproved_plan_execution_threshold` 파라미터의 값을 설정하고 계획의 총 비용이 이 임계값보다 작으면 최적화 프로그램이 미승인 계획을 실행합니다. 자세한 내용은 [apg_plan_mgmt.unapproved_plan_execution_threshold \(p. 841\)](#) 파라미터를 참조하십시오.
8. 관리형 설명문에 활성화되고 유효한 기본 계획이 있는 경우, 최적화 프로그램에서 최소 비용 계획을 실행합니다.
- 유효 계획은 최적화 프로그램에서 실행할 수 있는 계획입니다. 관리형 계획은 다양한 이유로 유효하지 않은 상태가 될 수 있습니다. 예를 들면 종속된 객체(파티셔닝된 테이블의 파티션이나 인덱스)가 제거되면 계획이 유효하지 않게 됩니다.
9. 최적화 프로그램은 관리형 설명문의 활성화 및 유효 상태인 승인된 계획에서 최소 비용 계획을 결정합니다. 그런 다음 최적화 프로그램은 최소 비용 승인된 계획을 실행합니다.

최적화 프로그램에서 사용할 계획 분석

`apg_plan_mgmt.use_plan_baselines` 파라미터가 `true`로 설정된 경우 EXPLAIN ANALYZE SQL 문을 사용하여 최적화 프로그램에서 해당 설명문을 실행해야 하는 경우에 사용할 계획을 표시할 수 있습니다. 다음은 예제입니다.

```
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
```

최적화 프로그램이 실행할 계획을 표시하지만, 이 예제에서는 비용이 더 저렴한 계획을 찾았습니다. 이 경우, [계획 자동 캡처 \(p. 830\)](#)에 설명된 바와 같이 자동 계획 캡처를 켜서 이 새로운 최소 비용 계획을 캡처합니다.

최적화 프로그램이 새 계획을 Unapproved로 캡처합니다. `apg_plan_mgmt.evolve_plan_baselines` 함수를 사용하여 계획을 비교하고 승인됨, 거부됨 또는 비활성화됨으로 변경할 수 있습니다. 자세한 내용은 [계획 성능 평가 \(p. 834\)](#) 단원을 참조하십시오.

실행 계획 유지 관리

쿼리 계획 관리는 실행 계획을, 추가, 유지 관리 및 개선할 수 있는 기술과 함수를 제공합니다.

주제

- [계획 성능 평가 \(p. 834\)](#)
- [계획 검증 \(p. 835\)](#)
- [pg_hint_plan을 사용하여 계획 수정 \(p. 835\)](#)
- [계획 삭제 \(p. 837\)](#)
- [계획 내보내기/가져오기 \(p. 837\)](#)

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.

주제

- [더 나은 계획 승인 \(p. 834\)](#)
- [느린 계획 거부 또는 비활성화 \(p. 835\)](#)

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` 보기 를 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 비활성화합니다.

```
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 false로 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.

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.

pg_hint_plan을 사용하여 계획 수정

퀴리 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 생성하기 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 설명서](#)를 참조하십시오.
- GUC 변수 – 하나 이상의 비용 모델 파라미터 또는 다른 최적화 프로그램 파라미터(예: fromCollapse_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 설명문을 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 보기 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 `Preferred`가 아니면 최적화 프로그램은 `Preferred` 계획을 선택합니다.

계획 삭제

장시간 사용하지 않았거나 더 이상 관련이 없는 계획을 삭제합니다. 최적화 프로그램이 계획을 실행하거나 해당 계획을 설명문에 대한 최소 비용 계획으로서 선택할 때마다 최적화 프로그램이 업데이트하는 `last_used` 날짜가 각 계획에 지정되어 있습니다. 계획이 최근에 사용되었고 여전히 관련이 있는지 여부를 확인하려면 `last_used` 날짜를 사용합니다.

예를 들어 다음과 같이 `apg_plan_mgmt.delete_plan` 기능을 사용할 수 있습니다. 이렇게 하면 최소 비용 계획으로 선택하지 않았거나 31일 이상 실행하지 않은 모든 계획이 삭제됩니다. 하지만 이 예제에서는 명시적으로 거부된 계획을 삭제하지 않습니다.

```
SELECT SUM(apg_plan_mgmt.delete_plan(sql_hash, plan_hash))
FROM apg_plan_mgmt.dba_plans
WHERE last_used < (current_date - interval '31 days')
AND status <> 'Rejected';
```

더 이상 유효하지 않으며 다시는 유효 상태로 전환되지 않을 것으로 생각되는 계획을 모두 삭제하려면 `apg_plan_mgmt.validate_plans` 함수를 사용합니다. 자세한 내용은 [계획 검증 \(p. 835\)](#) 단원을 참조하십시오.

계획 삭제를 위한 자신만의 고유 정책을 구현할 수 있습니다. 현재 날짜인 `last_used`과 `apg_plan_mgmt.plan_retention_period` 파라미터의 값(기본값은 32일)보다 크면 계획이 자동으로 삭제됩니다. 더 긴 간격을 지정하거나, `delete_plan` 함수를 직접 호출하여 고유한 계획 보조 정책을 구현할 수 있습니다.

Important

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. DB 인스턴스 수준 파라미터 그룹에서 이 파라미터를 설정하고 DB 인스턴스를 다시 시작하여 변경 내용을 적용합니다. 자세한 내용은 [apg_plan_mgmt.max_plans \(p. 839\)](#) 파라미터를 참조하십시오.

계획 내보내기/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_baseline, 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,
estimated_rows = EXCLUDED.estimated_rows,
actual_rows = EXCLUDED.actual_rows,
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;
```

쿼리 계획 관리를 위한 파라미터 참조

apg_plan_mgmt 확장은 다음 파라미터를 제공합니다.

파라미터

- [apg_plan_mgmt.capture_plan_baseline \(p. 839\)](#)
- [apg_plan_mgmt.max_databases \(p. 839\)](#)
- [apg_plan_mgmt.max_plans \(p. 839\)](#)
- [apg_plan_mgmt.pgss_min_calls \(p. 840\)](#)
- [apg_plan_mgmt.pgss_min_mean_time_ms \(p. 840\)](#)
- [apg_plan_mgmt.pgss_min_stddev_time_ms \(p. 840\)](#)
- [apg_plan_mgmt.pgss_min_total_time_ms \(p. 841\)](#)
- [apg_plan_mgmt.plan_retention_period \(p. 841\)](#)
- [apg_plan_mgmt.unapproved_plan_execution_threshold \(p. 841\)](#)
- [apg_plan_mgmt.use_plan_baseline \(p. 842\)](#)

해당 수준에서 쿼리 계획 파라미터를 설정합니다.

- 모든 DB 인스턴스에 대해 동일한 설정을 제공하기 위해 클러스터 수준에서 설정합니다. 자세한 내용은 [DB 클러스터 파라미터 그룹의 파라미터 설정 \(p. 177\)](#) 단원을 참조하십시오.
- 설정을 개별 DB 인스턴스로 분리하기 위해 DB 인스턴스 수준에서 설정합니다. 자세한 내용은 [DB 파라미터 그룹의 파라미터 설정 \(p. 174\)](#) 단원을 참조하십시오.
- 해당 세션으로만 값을 분리하려면 psql에서와 같이 특정 클라이언트 세션에서 설정합니다.

클러스터 또는 DB 인스턴스 수준에서 `apg_plan_mgmt.max_databases` 및 `apg_plan_mgmt.max_plans` 파라미터를 설정해야 합니다.

apg_plan_mgmt.capture_plan_baselines

SQL 문에 대한 실행 계획 캡처를 활성화합니다.

```
SET apg_plan_mgmt.capture_plan_baselines = [off | automatic | manual]
```

값	설명
off	계획 캡처를 비활성화합니다. 이 값이 기본값입니다.
automatic	적격성 기준을 충족하는 후속 SQL 문에 대해 계획 캡처를 활성화합니다.
manual	후속 SQL 문에 대해 계획 캡처를 활성화합니다.

apg_plan_mgmt.max_databases

쿼리 계획 관리를 사용할 수 있는 최대 데이터베이스 객체 수를 설정합니다. 데이터베이스 객체는 CREATE DATABASE SQL 문을 사용하여 생성되는 항목입니다.

Important

클러스터 또는 DB 인스턴스 수준에서 `apg_plan_mgmt.max_databases`를 설정합니다. 새 값을 적용하려면 DB 인스턴스를 다시 시작해야 합니다.

Value	기본값	설명
양의 정수	10	양의 정수 값입니다.

apg_plan_mgmt.max_plans

`apg_plan_mgmt.dba_plans` 보기에서 캡처할 수 있는 최대 계획 수를 설정합니다.

Important

클러스터 또는 DB 인스턴스 수준에서 `apg_plan_mgmt.max_plans`을 설정합니다. 새 값을 적용하려면 DB 인스턴스를 다시 시작해야 합니다.

Value	기본값	설명
integer	1000	10보다 크거나 같은 양의 정수 값입니다.

apg_plan_mgmt.pgss_min_calls

이 파라미터는 더 이상 사용되지 않습니다.

계획 캡처에 적합한 최소 pg_stat_statements 호출 수를 설정합니다.

```
SET apg_plan_mgmt.pgss_min_calls = integer-value;
```

값	기본값	설명
양의 정수	2	2보다 크거나 같은 양의 정수 값입니다.

사용 시 주의사항

pg_stat_statements 확장을 설치해야 합니다. 자세한 내용은 [PostgreSQL pg_stats_statements 설명서](#)를 참조하십시오.

apg_plan_mgmt.pgss_min_mean_time_ms

이 파라미터는 더 이상 사용되지 않습니다.

계획 캡처에 적합한 pg_stat_statements mean_time의 최소 값입니다.

```
SET apg_plan_mgmt.pgss_min_mean_time_ms = double-value;
```

값	기본값	설명
양수	0.0	0.0보다 크거나 같은 양수 값입니다.

사용 시 주의사항

pg_stat_statements 확장을 설치해야 합니다. 자세한 내용은 [PostgreSQL pg_stats_statements 설명서](#)를 참조하십시오.

apg_plan_mgmt.pgss_min_stddev_time_ms

이 파라미터는 더 이상 사용되지 않습니다.

계획 캡처에 적합한 pg_stat_statements stddev_time의 최소 값입니다.

```
SET apg_plan_mgmt.pgss_min_stddev_time_ms = double-value;
```

값	기본값	설명
양수	0.0	0.0보다 크거나 같은 양수 값입니다.

사용 시 주의사항

`pg_stat_statements` 확장을 설치해야 합니다. 자세한 내용은 [PostgreSQL pg_stats_statements 설명서](#)를 참조하십시오.

apg_plan_mgmt.pgss_min_total_time_ms

이 파라미터는 더 이상 사용되지 않습니다.

계획 캡처에 적합한 `pg_stat_statements total_time`의 최소 값입니다.

```
SET apg_plan_mgmt.pgss_min_total_time_ms = double-value;
```

값	기본값	설명
양수	0.0	0.0보다 크거나 같은 양수 값입니다.

사용 시 주의사항

`pg_stat_statements` 확장을 설치해야 합니다. 자세한 내용은 [PostgreSQL pg_stats_statements 설명서](#)를 참조하십시오.

apg_plan_mgmt.plan_retention_period

계획이 실제로 삭제되기 전에 `apg_plan_mgmt.dba_plans` 보기에서 유지되는 일수입니다. 현재 날짜가 `last_used` 날짜 이후에 이 기간이 경과된 날짜인 경우 계획이 삭제됩니다.

```
SET apg_plan_mgmt.plan_retention_period_ = integer-value;
```

값	기본값	설명
양의 정수	32	32보다 크거나 같은 양의 정수 값으로, 일수를 나타냅니다.

apg_plan_mgmt.unapproved_plan_execution_threshold

추정되는 총 계획 비용 임계값으로, 최적화 프로그램은 이 값보다 낮은 미승인 계획을 실행합니다. 기본적으로 최적화 프로그램은 미승인 계획을 실행하지 않습니다. 하지만 가장 빠른 미승인 계획에 대해 실행 임계값을 설정할 수 있습니다. 이 설정을 사용하면 최적화 프로그램이 승인된 계획만을 실행할 때의 오버헤드를 피할 수 있습니다.

```
SET apg_plan_mgmt.unapproved_plan_execution_threshold = integer-value;
```

값	기본값	설명
양의 정수	0	0보다 크거나 같은 양의 정수 값입니다. 0 값은 <code>use_plan_baselines</code> 가 <code>true</code> 일 때 미승인 계획이 실행되지 않음을 의미합니다.

다음 예제에서는 `use_plan_baselines`가 `true`라도 추정 비용이 550보다 낮은 경우 최적화 프로그램이 미승인 계획을 실행합니다.

```
SET apg_plan_mgmt.unapproved_plan_execution_threshold = 550;
```

apg_plan_mgmt.use_plan_baselines

최적화 프로그램에서 관리형 설명문에 대해 관리형 계획을 사용하도록 설정합니다.

```
SET apg_plan_mgmt.use_plan_baselines = [true | false];
```

값	설명
true	관리형 계획을 사용하도록 설정합니다. SQL 문이 실행되고 이 설명문이 apg_plan_mgmt.dba_plans 보기의 관리형 설명문인 경우 최적화 프로그램은 다음 순서대로 관리형 계획을 선택합니다. <ol style="list-style-type: none">유효하고 활성화된 최소 비용 기본 계획유효하고 활성화된 최소 비용 승인 계획유효하고 활성화되었으며 임계값 (apg_plan_mgmt.unapproved_plan_execution_threshold 파라미터를 사용하여 설정한 경우)을 충족하는 최소 비용 미승인 계획최적화 프로그램에서 생성된 최소 비용 계획
false	(기본값) 관리형 계획을 사용하지 않습니다. 최적화 프로그램에서는 생성된 최소 비용 계획을 사용합니다.

사용 시 주의사항

use_plan_baselines가 true이면 최적화 프로그램에서 다음과 같이 실행을 결정합니다.

- 최적화 프로그램의 계획에 대한 추정 비용이 unapproved_plan_execution_threshold보다 낮으면 해당 계획을 실행합니다. 그렇지 않을 경우
- 계획이 approved 또는 preferred이면 해당 계획을 실행합니다. 그렇지 않을 경우
- 가능하면 최소 비용 preferred 계획을 실행합니다. 그렇지 않을 경우
- 가능하면 최소 비용 approved 계획을 실행합니다. 그렇지 않을 경우
- 최적화 프로그램의 최소 비용 계획을 실행합니다.

쿼리 계획 관리를 위한 함수 참조

apg_plan_mgmt 확장은 다음 함수를 제공합니다.

함수

- [apg_plan_mgmt.delete_plan \(p. 843\)](#)
- [apg_plan_mgmt.evolve_plan_baselines \(p. 843\)](#)
- [apg_plan_mgmt.plan_last_used \(p. 844\)](#)
- [apg_plan_mgmt.reload \(p. 845\)](#)
- [apg_plan_mgmt.set_plan_enabled \(p. 845\)](#)
- [apg_plan_mgmt.set_plan_status \(p. 846\)](#)
- [apg_plan_mgmt.validate_plans \(p. 846\)](#)

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 빠른지 여부 또는 쿼리 최적화 프로그램에서 최소 비용 계획으로 식별된 계획이 더 빠른지 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_speedup_factor	최소 속도 향상 인수는 계획이 승인되려면 이미 승인된 계획보다 몇 배 더 빨라야 하는지를 지정합니다. 또는 거부되거나 비활성화되려면 몇 배 더 느려야 하는지를 지정할 수도 있습니다. 이 값은 양수 부동 값입니다.

파라미터	설명
action	<p>함수가 수행해야 할 작업입니다. 유효 값에는 다음이 포함됩니다. 대/소문자를 구분하지 않습니다.</p> <ul style="list-style-type: none"> 'disable' – 최소 속도 향상 인수를 충족하지 않는 각각의 일치하는 계획을 비활성화합니다. 'approve' – 최소 속도 향상 인수를 충족하는 각각의 일치하는 계획을 활성화하고 상태를 <code>approved</code>로 설정합니다. 'reject' – 최소 속도 향상 인수를 충족하지 않는 각각의 일치하는 계획의 경우, 상태를 <code>rejected</code>로 설정합니다. NULL – 이 함수는 단순히 최소 속도 향상 인수를 충족하지 않아서 성능 이점을 없는 계획 수만을 반환합니다.

사용 시 주의사항

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 `last_verified` 타임스탬프가 있는 승인된 계획에서만 `apg_plan_mgmt.evolve_plan_baselines` 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 종분형 이점(또는 단점)이 `total_time_benefit_ms` 열의 `apg_plan_mgmt.dba_plans` 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, `apg_plan_mgmt.dba_plans` 보기의 `last_verified` 열이 `current_timestamp`로 업데이트됩니다. `last_verified` 타임스탬프를 사용하면 최근에 성능이 확인된 계획에 대해 이 함수가 다시 실행되지 않도록 할 수 있습니다.

apg_plan_mgmt.plan_last_used

공유 메모리에서 지정된 계획의 `last_used` 날짜를 반환합니다.

구문

```
apg_plan_mgmt.plan_last_used(
    sql_hash,
    plan_hash
)
```

반환 값

`last_used` 날짜를 반환합니다.

파라미터

파라미터	설명
<code>sql_hash</code>	계획의 관리형 SQL 문의 <code>sql_hash</code> ID.

파라미터	설명
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 시 주의사항

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): <ul style="list-style-type: none">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: <ul style="list-style-type: none">ApprovedUnapprovedRejectedPreferred 대문자를 사용하든 소문자를 사용하든 apg_plan_mgmt.dba_plans 보기에서 상태 값은 초기 대문자로 설정됩니다. 이러한 값에 대한 자세한 내용은 apg_plan_mgmt.dba_plans 보기에 대한 참조 (p. 827) 의 status를 참조하십시오.

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. 원활 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. <ul style="list-style-type: none">• 'disable' – 각각의 잘못된 계획이 비활성화됩니다.• 'delete' – 각각의 잘못된 계획이 삭제됩니다.• 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 시 주의사항

전체 apg_plan_mgmt.dba_plans 보기의 모든 관리형 설명문에 대해 모든 관리형 계획을 검증하려면 validate_plans(action) 형태를 사용합니다.

sql_hash가 지정된 관리형 설명문의 경우 plan_hash가 지정된 관리형 계획을 검증하려면 validate_plans(sql_hash, plan_hash, action) 형태를 사용합니다.

sql_hash가 지정된 관리형 설명문에 대해 모든 관리형 계획을 검증하려면 validate_plans(sql_hash, NULL, action) 형태를 사용합니다.

Amazon CloudWatch Logs에 Aurora PostgreSQL 로그 게시

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.

Note

다음에 유의하십시오.

- Aurora PostgreSQL에서는 버전 9.6.12 이상 및 버전 10.7 이상에 대해 로그를 CloudWatch Logs에 게시하는 기능을 지원합니다.
- Aurora PostgreSQL에서 postgresql 로그만 게시 할 수 있습니다. 업그레이드 로그 게시는 지원되지 않습니다.
- 중국(닝샤) 리전의 경우 CloudWatch Logs에 로그를 게시할 수 없습니다.
- 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 Logs로 내보내고 싶지 않은 경우, 감사 로그를 내보내는 모든 방법이 비활성화되었는지 확인하십시오. 이러한 방법으로는 AWS Management 콘솔, AWS CLI 및 RDS API가 해당합니다.

콘솔

콘솔을 사용하여 CloudWatch Logs에 Aurora PostgreSQL 로그를 게시할 수 있습니다.

콘솔에서 Aurora PostgreSQL 로그를 게시하려면

1. <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 로그 데이터를 게시하려는 Aurora PostgreSQL DB 클러스터를 선택합니다.
4. 수정을 선택합니다.
5. 로그 내보내기 부분에서 Postgresql 로그를 선택합니다.
6. 계속을 선택한 후, 요약 페이지에서 클러스터 수정을 선택합니다.

AWS CLI

AWS CLI를 사용하여 Aurora PostgreSQL 로그를 게시할 수 있습니다. 이 경우 [modify-db-cluster](#) AWS CLI 명령을 다음 옵션 중 하나를 포함해 실행할 수 있습니다.

- `--db-cluster-identifier`—DB 클러스터 식별자입니다.
- `--cloudwatch-logs-export-configuration`—DB 클러스터에 대하여 CloudWatch Logs로 내보내기를 활성화할 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.

Linux, OS X, Unix의 경우:

```
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 postgresql입니다.

Linux, OS X, Unix의 경우:

```
aws rds modify-db-cluster \
--db-cluster-identifier my-db-cluster \
--cloudwatch-logs-export-configuration '{"EnableLogTypes":["postgresql"]}'
```

Windows의 경우:

```
aws rds modify-db-cluster ^
--db-cluster-identifier my-db-cluster ^
--cloudwatch-logs-export-configuration '{\"EnableLogTypes\":[\"postgresql\"]}'
```

Note

Windows 명령 프롬프트를 사용하는 경우 백슬래시(\)를 접두사로 추가하여 JSON 코드에서 큰 따옴표(")를 이스케이프해야 합니다.

Example

다음 예제에서는 기존 Aurora PostgreSQL DB 클러스터를 수정하여 CloudWatch Logs에 로그 파일을 게시하는 것을 비활성화합니다. --cloudwatch-logs-export-configuration 값은 JSON 객체입니다. 이 객체의 키는 DisableLogTypes,이고 그 값은 postgresql입니다.

Linux, OS X, Unix의 경우:

```
aws rds modify-db-cluster \
--db-cluster-identifier mydbinstance \
--cloudwatch-logs-export-configuration '{"DisableLogTypes":["postgresql"]}'
```

Windows의 경우:

```
aws rds modify-db-cluster ^
--db-cluster-identifier mydbinstance ^
--cloudwatch-logs-export-configuration "{\"DisableLogTypes\":[\"postgresql\"]}"
```

Note

Windows 명령 프롬프트를 사용하는 경우 백슬래시(\)를 접두사로 추가하여 JSON 코드에서 큰 따옴표(")를 이스케이프해야 합니다.

RDS API

RDS API를 사용하여 Aurora PostgreSQL 로그를 게시할 수 있습니다. 이 경우 [ModifyDBCluster](#) 작업을 다음 옵션 중 하나를 포함해 실행할 수 있습니다.

- **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 PostgreSQL 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 활성화한 후에 Amazon CloudWatch Logs에서 이 이벤트를 모니터링할 수 있습니다. 모니터링에 대한 자세한 내용은 [CloudWatch 로그로 전송된 로그 데이터 보기](#)를 참조하십시오.

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 *my-db-cluster*라는 이름의 DB 클러스터에 postgresql 로그를 포함하도록 내보내기 함수를 구성하면, PostgreSQL 로그 데이터가 /aws/rds/cluster/*my-db-cluster*/postgresql 로그 그룹에 저장됩니다.

DB 클러스터의 모든 DB 인스턴스에 있는 모든 이벤트가 서로 다른 로그 스트림을 사용하는 로그 그룹으로 이동합니다.

지정된 이름이 있는 로그 그룹이 존재할 경우 Aurora는 이 로그 그룹을 사용하여 Aurora DB 클러스터의 로그 데이터를 내보냅니다. AWS CloudFormation 같은 자동 구성은 미리 정의된 로그 보존 기간, 메트릭 필터 및 고객 액세스 권한이 있는 로그 그룹을 생성할 수 있습니다. 또는 기본 로그 보존 기간인 만기 없음을 CloudWatch Logs에서 사용하면 새 로그 그룹이 자동으로 생성됩니다. CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용하여 로그 보존 기간을 변경할 수 있습니다. CloudWatch Logs의 로그 보존 기간 변경에 대한 자세한 내용은 [CloudWatch Logs에서 로그 데이터 보존 변경](#)을 참조하십시오.

CloudWatch Logs 콘솔, AWS CLI 또는 CloudWatch Logs API를 사용하여 DB 클러스터의 로그 이벤트 안에서 정보를 검색할 수 있습니다. 로그 데이터 검색 및 필터링에 관한 자세한 내용은 [로그 데이터 검색 및 필터](#)를 참조하십시오.

Aurora PostgreSQL에서 기계 학습(ML) 사용

Amazon Aurora 기계 학습은 SQL 언어를 사용해 기계 학습 기반 예측을 데이터베이스 애플리케이션에 추가할 수 있게 지원합니다. Aurora 기계 학습은 Aurora 데이터베이스와 AWS 기계 학습(ML) 서비스인 Amazon SageMaker 및 Amazon Comprehend 간의 고도로 최적화된 통합을 이용합니다.

Aurora 기계 학습의 이점은 다음과 같습니다.

- ML 기반 예측을 기존 데이터베이스 애플리케이션에 추가할 수 있습니다. 사용자 지정 통합을 빌드하거나 별도 도구를 학습할 필요가 없습니다. 기계 학습 처리를 함수에 대한 호출로 SQL 쿼리에 직접 포함할 수 있습니다.
- ML 통합으로 ML 서비스가 트랜잭션 데이터 작업을 할 수 있게 신속히 지원할 수 있습니다. 기계 학습 작업을 수행하기 위해 데이터를 데이터베이스 밖으로 이동할 필요가 없습니다. 데이터베이스 애플리케이션에서 사용하기 위해 기계 학습의 결과를 변환하거나 다시 가져올 필요가 없습니다.
- 기존 거버넌스 정책을 사용하여 기본 데이터 및 생성된 통찰력에 대한 액세스 권한을 어떤 사용자에게 부여할지 제어할 수 있습니다.

AWS 기계 학습 서비스는 자체 프로덕션 환경에서 설정하고 실행하는 관리형 서비스입니다. 현재 Aurora 기계 학습은 감성 분석을 위한 Amazon Comprehend와 광범위한 ML 알고리즘을 위한 Amazon SageMaker와 통합됩니다.

Amazon Comprehend에 대한 일반적인 내용은 [Amazon Comprehend 단원](#)을 참조하십시오. Aurora 및 Amazon Comprehend를 함께 사용하는 것에 대한 자세한 내용은 [자연어 처리에 Amazon Comprehend 사용 \(p. 854\)](#) 단원을 참조하십시오.

Amazon SageMaker에 대한 일반적인 내용은 [Amazon SageMaker 단원](#)을 참조하십시오. Aurora 및 Amazon SageMaker를 함께 사용하는 것에 대한 자세한 내용은 [Amazon SageMaker를 사용하여 자체 ML 모델 실행 \(p. 855\)](#) 단원을 참조하십시오.

Note

Aurora 기계 학습 for PostgreSQL은 Aurora 클러스터를 동일한 AWS 리전 내에 있는 Amazon SageMaker 또는 Amazon Comprehend 서비스에만 연결합니다.

주제

- [Aurora 기계 학습 활성화 \(p. 851\)](#)
- [자연어 처리에 Amazon Comprehend 사용 \(p. 854\)](#)
- [Amazon SageMaker 모델 교육을 위해 Amazon S3로 데이터 내보내기 \(p. 855\)](#)
- [Amazon SageMaker를 사용하여 자체 ML 모델 실행 \(p. 855\)](#)
- [Aurora 기계 학습 모범 사례 \(p. 858\)](#)
- [Aurora 기계 학습 모니터링 \(p. 861\)](#)
- [Aurora 기계 학습에 대한 PostgreSQL 함수 참조 \(p. 863\)](#)
- [AWS CLI를 사용하여 수동으로 Amazon SageMaker 및 Amazon Comprehend에 대한 IAM 역할 설정 \(p. 864\)](#)

Aurora 기계 학습 활성화

Aurora 기계 학습은 다음 Aurora PostgreSQL 버전에서 사용할 수 있습니다.

- PostgreSQL 엔진 버전 10.11을 실행하는 PostgreSQL과 호환되는 Aurora 버전 2.4
- PostgreSQL 엔진 버전 11.6 이상을 실행하는 PostgreSQL과 호환되는 Aurora 버전 3.1 이상

이전 Aurora 클러스터 업그레이드에 대한 자세한 내용은 [Aurora PostgreSQL용 PostgreSQL DB 엔진 업그레이드 \(p. 872\)](#) 단원을 참조하십시오.

ML 기능 활성화에는 다음과 같은 단계가 수반됩니다.

주제

- AWS 기계 학습 서비스에 대한 IAM 액세스 설정 (p. 852)
- 모델 추론을 위한 aws_ml 확장 설치 (p. 853)

AWS 기계 학습 서비스에 대한 IAM 액세스 설정

Amazon SageMaker 및 Amazon Comprehend 서비스에 액세스하기 전에 AWS Identity and Access Management(IAM) 역할을 설정합니다. 그런 다음 Aurora PostgreSQL 클러스터에 IAM 역할을 추가합니다. 이 역할은 Aurora PostgreSQL 데이터베이스의 사용자에게 AWS ML 서비스에 액세스할 수 있는 권한을 부여합니다.

여기에서 표시된 대로 AWS Management 콘솔을 사용하여 IAM 설정을 자동으로 수행할 수 있습니다. AWS CLI를 사용하여 IAM 서비스를 설정하려면 [AWS CLI를 사용하여 수동으로 Amazon SageMaker 및 Amazon Comprehend에 대한 IAM 역할 설정 \(p. 864\)](#) 단원을 참조하십시오.

콘솔을 사용하여 Aurora DB 클러스터를 AWS 서비스에 자동으로 연결

Aurora 기계 학습에서는 DB 클러스터가 Amazon S3, Amazon SageMaker 및 Amazon Comprehend의 일부 조합을 사용할 것을 요구합니다. Amazon Comprehend는 감성 분석을 위한 것이고, Amazon SageMaker는 광범위한 기계 학습 알고리즘을 위한 것입니다.

Aurora 기계 학습의 경우 Amazon S3만 사용하여 Amazon SageMaker 모델을 교육합니다. 사용할 수 있는 교육된 모델이 아직 없고 교육이 자신의 책임인 경우에는 Aurora 기계 학습에서 Amazon S3만 사용하면 됩니다.

DB 클러스터를 이 서비스에 연결하려면 각 Amazon 서비스에 대해 AWS Identity and Access Management(IAM) 역할을 설정해야 합니다. IAM 역할을 통해 DB 클러스터의 사용자는 해당 서비스로 인증 할 수 있습니다.

Amazon SageMaker, Amazon Comprehend 또는 Amazon S3에 대해 IAM 역할을 생성하려면 필요한 각 서비스에 대해 다음 단계를 반복하십시오.

DB 클러스터를 Amazon 서비스에 연결하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 열니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 사용하려는 Aurora PostgreSQL DB 클러스터를 선택합니다.
3. Connectivity & security(연결 및 보안) 탭을 선택합니다.
4. IAM 역할 관리 섹션에서 Select a service to connect to this cluster(이 클러스터에 연결할 서비스 선택)를 선택합니다.
5. 목록에서 연결할 서비스를 선택합니다.
 - Amazon S3
 - Amazon Comprehend
 - Amazon SageMaker
6. Connect service(서비스 연결)를 선택합니다.
7. Connect service(서비스 연결) 창에서 특정 서비스에 대한 필수 정보를 입력하십시오.
 - Amazon SageMaker의 경우 Amazon SageMaker 엔드포인트의 Amazon 리소스 이름(ARN)을 입력하십시오.

Amazon SageMaker 콘솔의 탐색 창에서 엔드포인트를 선택한 다음 사용하려는 엔드포인트의 ARN을 복사합니다. 엔드포인트가 무엇을 나타내는지에 관한 자세한 내용은 [Amazon SageMaker 호스팅 서비스에서 모델 배포](#) 단원을 참조하십시오.

- Amazon Comprehend의 경우 추가 파라미터는 지정하지 않습니다.

- Amazon S3의 경우 사용할 Amazon S3 버킷의 ARN을 입력하십시오.

Amazon S3 버킷 ARN의 형식은 `arn:aws:s3:::bucket_name`입니다. 사용하는 Amazon S3 버킷이 교육 Amazon SageMaker 모델에 대한 요구 사항을 포함하도록 설정되어 있는지 확인하십시오. 모델을 교육할 때 Aurora DB 클러스터에서는 데이터를 Amazon S3 버킷으로 내보낼 수 있는 권한뿐 아니라 데이터를 버킷에서 가져올 수 있는 권한도 요구합니다.

Amazon S3 버킷 ARN에 대한 자세한 내용은 Amazon Simple Storage Service 개발자 가이드의 [정책에서 리소스 지정](#)을 참조하십시오. Amazon SageMaker에서 Amazon S3 버킷을 사용하는 방법에 대한 자세한 내용은 Amazon SageMaker 개발자 안내서의 [1단계: Amazon S3 버킷 생성](#)을 참조하십시오.

- Connect service(서비스 연결)를 선택합니다.
- Aurora는 새 IAM 역할을 생성하여 이 클러스터에 대한 현재 IAM 역할의 DB 클러스터 목록에 추가합니다. IAM 역할의 상태는 처음에는 진행 중으로 되어 있습니다. IAM 역할 이름은 연결된 각 서비스에 대해 다음과 같은 패턴으로 자동 생성됩니다.
 - Amazon S3 IAM 역할 이름 패턴은 `rds-cluster_ID-S3-role-timestamp`입니다.
 - Amazon SageMaker IAM 역할 이름 패턴은 `rds-cluster_ID-SageMaker-role-timestamp`입니다.
 - Amazon Comprehend IAM 역할 이름 패턴은 `rds-cluster_ID-Comprehend-role-timestamp`입니다.

또한 Aurora는 새 IAM 정책을 생성하여 역할에 연결합니다. 정책 이름은 유사한 이름 지정 규칙을 따르며 타임스탬프도 있습니다.

모델 추론을 위한 aws_ml 확장 설치

필요한 IAM 역할을 생성하여 Aurora PostgreSQL DB 클러스터와 연결한 후 Amazon SageMaker 및 Amazon Comprehend 기능을 사용하는 함수를 설치합니다. `aws_ml` Aurora PostgreSQL 확장은 Amazon SageMaker와 직접 통신하는 `aws_sagemaker.invoke_endpoint` 함수를 제공합니다. `aws_ml` 확장은 Amazon Comprehend와 직접 통신하는 `aws_comprehend.detect_sentiment` 함수도 제공합니다.

특정 데이터베이스에 이러한 함수를 설치하려면 psql 프롬프트에서 다음 SQL 명령을 입력합니다.

```
psql>CREATE EXTENSION IF NOT EXISTS aws_ml CASCADE;
```

`template1` 기본 데이터베이스에서 `aws_ml` 확장을 생성할 경우 생성하는 각각의 새 데이터베이스에서 함수를 사용할 수 있습니다.

설치를 확인하려면 psql 프롬프트에서 다음을 입력합니다.

```
psql>\dx
```

Amazon Comprehend에 대한 IAM 역할을 설정하는 경우 다음과 같이 설정을 확인할 수 있습니다.

```
SELECT sentiment FROM aws_comprehend.detect_sentiment(null, 'I like it!', 'en');
```

`aws_ml` 확장을 설치하면 `aws_ml` 관리 역할이 생성되고 `rds_superuser` 역할에 부여됩니다. `aws_sagemaker` 서비스 및 `aws_comprehend` 서비스에 대해 별도의 스키마도 생성됩니다. `rds_superuser` 역할은 이러한 두 스키마의 OWNER로 구성됩니다.

사용자 또는 역할이 `aws_ml` 확장의 함수에 대한 액세스 권한을 얻으려면 해당 함수에 대한 `EXECUTE` 권한을 부여합니다. 필요한 경우 이후에 실행 권한을 취소할 수 있습니다. `EXECUTE` 권한은 기본적으로 PUBLIC

에서 이러한 스키마의 함수에 대해 취소됩니다. 멀티 테넌트 데이터베이스 구성에서 테넌트가 함수에 액세스 하지 못하도록 하려면 하나 이상의 ML 서비스 스키마에서 REVOKE USAGE를 사용합니다.

aws_ml 확장의 설치된 함수에 대한 참조는 [Aurora 기계 학습에 대한 PostgreSQL 함수 참조 \(p. 863\)](#) 단원을 참조하십시오.

자연어 처리에 Amazon Comprehend 사용

Amazon Comprehend는 기계 학습을 사용하여 텍스트에서 통찰력과 관계를 찾습니다. Amazon Comprehend는 자연어 처리를 사용하여 문서 내용에 대한 통찰력을 추출합니다. 문서에 있는 엔터티, 핵심 문구, 언어, 감정 및 기타 일반적인 요소를 인식하여 통찰력을 개발합니다. 이 Aurora 기계 학습 서비스는 기계 학습 경험이 거의 없어도 사용할 수 있습니다.

Aurora 기계 학습은 데이터베이스에 저장된 텍스트의 감성 분석에 Amazon Comprehend를 사용합니다. 감성은 텍스트로 표현된 의견입니다. 감성 분석은 감성을 식별하고 분류하여 주제 또는 제품 등의 항목에 대한 태도가 긍정적인지, 부정적인지 또는 중립인지를 결정합니다.

Note

Amazon Comprehend는 현재 일부 AWS 리전에서만 사용할 수 있습니다. Amazon Comprehend를 사용할 수 있는 AWS 리전을 확인하려면 AWS 사이트에서 [AWS 리전 표](#) 페이지를 참조하십시오.

예를 들어 Amazon Comprehend를 사용해 고객 센터 수신 통화 관련 문서를 분석하여 발신자 감성을 감지하고 발신자-에이전트 간 역학을 더 잘 이해할 수 있습니다. AWS 기계 학습 블로그의 [콜센터 통화 분석](#)이라는 게시물에서 더 자세한 설명을 볼 수 있습니다.

또한 단일 쿼리를 사용하여 데이터베이스의 기타 정보에 대한 분석을 감성 분석과 결합할 수 있습니다. 예를 들어 다음 사항이 합쳐진 문제에 대한 수신 통화 센터의 평균 감성을 감지할 수 있습니다.

- 30일 이상 미해결 상태
- 특정 제품 또는 기능과 관련된 문제
- 소셜 미디어 영향력이 가장 큰 고객에게 발생한 문제

Aurora 기계 학습에서 Amazon Comprehend를 사용하는 것은 SQL 함수를 호출하는 것만큼 쉽습니다. aws_ml 확장([모델 추론을 위한 aws_ml 확장 설치 \(p. 853\)](#))을 설치하면 Amazon Comprehend를 통해 감성 분석을 수행하는 [aws_comprehend.detect_sentiment \(p. 863\)](#) 함수가 제공됩니다.

분석하는 각 텍스트 조각에 이 함수를 사용하면 감성과 신뢰도 수준을 확인하는 데 도움이 됩니다. 일반적인 Amazon Comprehend 쿼리는 감성에 특정 값(POSITIVE 또는 NEGATIVE)이 있고 신뢰도 수준이 일정 비율 보다 큰 테이블 행을 찾습니다.

예를 들어 다음 쿼리에서는 데이터베이스 테이블 myTable.document 내 문서의 평균 감성을 확인하는 방법을 보여줍니다. 이 쿼리에서는 평가 신뢰도가 80% 이상인 문서만 고려합니다. 다음 예에서 감성 텍스트의 언어는 영어(en)입니다.

```
SELECT AVG(CASE s.sentiment
    WHEN 'POSITIVE' then 1
    WHEN 'NEGATIVE' then -1
    ELSE 0 END) as avg_sentiment, COUNT(*) AS total
FROM myTable, aws_comprehend.detect_sentiment (myTable.document, 'en') s
WHERE s.confidence >= 0.80;
```

테이블 행당 두 번 이상 감성 분석 요금이 청구되지 않도록 한 행당 한 번씩 분석 결과를 구체화할 수 있습니다. 관심있는 행에 대해 이 작업을 수행하십시오. 다음 예에서 감성 텍스트의 언어는 프랑스어(fr)입니다.

```
-- *Example:* Update the sentiment and confidence of French text.
--
```

```
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_nodes)) > 0 AND
    clinician_notes.sentiment IS NULL;
```

함수 호출 최적화에 대한 자세한 내용은 [Aurora 기계 학습 모범 사례 \(p. 858\)](#) 단원을 참조하십시오.

감성 감지 함수의 파라미터 및 반환 유형에 대한 자세한 내용은 [aws_comprehend.detect_sentiment \(p. 863\)](#) 단원을 참조하십시오.

Amazon SageMaker 모델 교육을 위해 Amazon S3로 데이터 내보내기

팀에서 기계 학습 작업을 어떻게 분배하느냐에 따라 모델 교육을 수행하지 않을 수도 있습니다. 다른 누군가가 사용자에게 Amazon SageMaker 모델을 제공하는 경우 사용자는 이 섹션을 건너뛸 수 있습니다.

Amazon SageMaker 모델을 교육하려면 데이터를 Amazon S3 버킷으로 내보냅니다. Amazon S3 버킷은 모델이 배포되기 전에 Amazon SageMaker에서 모델을 교육하는 데 사용됩니다. Aurora PostgreSQL DB 클러스터에서 데이터를 쿼리한 후 Amazon S3 버킷에 저장된 텍스트 파일에 직접 저장할 수 있습니다. 그런 다음 Amazon SageMaker에서 교육을 위해 Amazon S3 버킷의 데이터를 사용합니다. Amazon SageMaker 모델 교육에 대한 자세한 내용은 [Amazon SageMaker로 모델 교육](#)을 참조하십시오.

Note

Amazon SageMaker 모델 교육 또는 배치 점수 계산을 위해 S3 버킷을 생성할 때는 항상 S3 버킷 이름에 sagemaker 텍스트를 포함시킵니다. Amazon SageMaker용 S3 버킷 생성에 대한 자세한 내용은 [1단계: Amazon S3 버킷 생성](#)을 참조하십시오.

데이터 내보내기에 대한 자세한 내용은 [Aurora PostgreSQL DB 클러스터의 데이터를 Amazon S3으로 내보내기 \(p. 812\)](#) 단원을 참조하십시오.

Amazon SageMaker를 사용하여 자체 ML 모델 실행

Amazon SageMaker는 종합 관리형 기계 학습 서비스입니다. 데이터 과학자와 개발자는 Amazon SageMaker를 사용하여 기계 학습 모델을 빌드하고 교육합니다. 그런 다음 모델을 프로덕션용 호스팅 환경에 직접 배포할 수 있습니다.

Amazon SageMaker는 데이터 원본에 대한 액세스를 제공하므로 서버의 하드웨어 인프라를 관리하지 않고도 탐색 및 분석을 수행할 수 있습니다. Amazon SageMaker는 또한 분산 환경에서 매우 큰 데이터 세트에 대해 효율적으로 실행되도록 최적화된 공통 기계 학습 알고리즘을 제공합니다. BYOM(Bring-Your-Own-Algorithm) 및 프레임워크 기본 지원을 통해 Amazon SageMaker는 특정 워크플로에 맞게 조정되는 유연한 분산형 교육 옵션을 제공합니다.

Note

현재 Aurora 기계 학습은 text/csv의 ContentType 값을 통해 쉼표로 구분된 값(CSV) 형식을 읽고 쓸 수 있는 모든 Amazon SageMaker 엔드포인트를 지원합니다. 현재 이 형식을 수용하는 기본 제공 Amazon SageMaker 알고리즘은 Random Cut Forest, Linear Learner 및 XGBoost입니다.

사용 중인 모델을 Aurora PostgreSQL 클러스터와 동일한 AWS 리전에 배포해야 합니다. Aurora 기계 학습은 항상 Aurora 클러스터와 동일한 AWS 리전에서 Amazon SageMaker 엔드포인트를 호출합니다.

[모델 추론을 위한 aws_ml 확장 설치 \(p. 853\)](#)의 설명에 따라 aws_ml 확장을 설치하면 [aws_sagemaker.invoke_endpoint \(p. 863\)](#) 함수가 제공됩니다. 이 함수를 사용하여 Amazon SageMaker 모델을 호출하고 SQL 데이터베이스 애플리케이션 내에서 직접 모델 추론을 수행합니다.

주제

- [Amazon SageMaker 모델을 호출하는 사용자 정의 함수 생성 \(p. 856\)](#)
- [Amazon SageMaker 모델에 대한 입력으로 배열 전달 \(p. 857\)](#)
- [Amazon SageMaker 모델 호출 시 배치 크기 지정 \(p. 857\)](#)
- [다중 출력이 있는 Amazon SageMaker 모델 호출 \(p. 857\)](#)

Amazon SageMaker 모델을 호출하는 사용자 정의 함수 생성

각 SageMaker 모델에 대해 `aws_sagemaker.invoke_endpoint`를 호출하는 별도의 사용자 정의 함수를 생성합니다. 사용자 정의 함수는 이 모델을 호스팅하는 Amazon SageMaker 엔드포인트를 나타냅니다. 이 `aws_sagemaker.invoke_endpoint` 함수는 사용자 정의 함수 내에서 실행됩니다. 사용자 정의 함수는 다음과 같은 많은 장점을 제공합니다.

- 모든 ML 모델에 대해 `aws_sagemaker.invoke_endpoint`를 호출하는 대신 ML 모델에 고유한 이름을 지정할 수 있습니다.
- SQL 애플리케이션 코드의 한 곳에만 모델 엔드포인트 URL을 지정할 수 있습니다.
- 각 ML 함수에 대한 EXECUTE 권한을 독립적으로 제어할 수 있습니다.
- SQL 유형을 사용하여 모델 입력 및 출력 유형을 선언할 수 있습니다. SQL은 ML 모델에 전달된 인수의 수와 유형을 적용하고 필요한 경우 유형 변환을 수행합니다. SQL 유형을 사용하면 SQL NULL이 ML 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 FUNCTION`을 사용합니다. 함수를 정의할 때 다음을 지정합니다.

- 모델에 대한 입력 파라미터
- 호출할 특정 Amazon SageMaker 엔드포인트
- 반환 유형

사용자 정의 함수에서는 입력 파라미터에 근거하여 모델을 실행한 후에 Amazon SageMaker 엔드포인트가 계산한 추론을 반환합니다. 다음 예제에서는 두 개의 입력 파라미터가 있는 Amazon SageMaker 모델에 대해 사용자 정의 함수를 생성합니다.

```
CREATE FUNCTION classify_event (IN arg1 INT, IN arg2 DATE, OUT category INT)
AS $$$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        arg1, arg2
        )::INT
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

다음을 참조하십시오.

- `aws_sagemaker.invoke_endpoint` 함수 입력은 모든 데이터 형식에 대한 하나 이상의 파라미터가 될 수 있습니다.

파라미터에 대한 자세한 내용은 [aws_sagemaker.invoke_endpoint \(p. 863\)](#) 함수 참조를 참조하십시오.

- 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 실행을 활성화하려면 PARALLEL SAFE를 지정합니다. 자세한 내용은 [병렬 쿼리 처리 활용 \(p. 860\)](#) 단원을 참조하십시오.

- 함수의 실행 비용을 추정하려면 COST 5000을 지정합니다. 함수의 예상 실행 비용을 `cpu_operator_cost` 단위로 제공하는 양수를 사용합니다.

Amazon SageMaker 모델에 대한 입력으로 배열 전달

`aws_sagemaker.invoke_endpoint` (p. 863) 함수는 PostgreSQL 함수의 한계인 최대 100개의 입력 파라미터를 가질 수 있습니다. Amazon SageMaker 모델에 동일한 유형의 파라미터가 100개 이상 필요한 경우 모델 파라미터를 배열로 전달하십시오.

다음 예제에서는 배열을 Amazon SageMaker 회귀 모델에 대한 입력으로 전달하는 사용자 정의 함수를 생성합니다.

```
CREATE FUNCTION regression_model (params REAL[], OUT estimate REAL)
AS $$$
    SELECT aws_sagemaker.invoke_endpoint (
        'sagemaker_model_endpoint_name', NULL,
        params                               -- model input parameters as an array
        )::REAL                            -- cast output to REAL
$$ LANGUAGE SQL PARALLEL SAFE COST 5000;
```

Amazon SageMaker 모델 호출 시 배치 크기 지정

다음 예제에서는 배치 크기 기본값을 NULL로 설정하는 Amazon SageMaker 모델에 대한 사용자 정의 함수를 생성합니다. 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 퀴리 최적화 프로그램이 자동으로 최대 배치 크기를 선택합니다. 자세한 내용은 [Aurora 기계 학습 함수 호출에 대한 배치 모드 실행 최적화](#) (p. 858) 단원을 참조하십시오.
- 기본적으로 파라미터의 값으로 NULL을 전달하면 Amazon SageMaker에 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. `aws_sagemaker.invoke_endpoint`를 호출할 때 COALESCE를 사용하여 NULL을 원하는 NULL 대체 값으로 변환합니다. 이 예제에 있는 `amount` 파라미터의 경우 NULL 값은 0.0 으로 변환됩니다.

다중 출력이 있는 Amazon SageMaker 모델 호출

다음 예제에서는 다중 출력을 반환하는 Amazon SageMaker 모델에 대해 사용자 정의 함수를 생성합니다. 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.

Aurora 기계 학습 모범 사례

`aws_ml` 함수 호출 시 대부분의 작업은 외부 Aurora 기계 학습 서비스 내에서 이루어집니다. 이러한 분리를 통해 Aurora 클러스터와 독립적으로 기계 학습 서비스의 리소스를 확장할 수 있습니다. Aurora 내에서는 주로 사용자 정의 함수 호출 자체를 최대한 효율적으로 수행하는 데 집중하십시오. Aurora 클러스터에서 영향을 받을 수 있는 몇 가지 측면은 다음과 같습니다.

- `aws_ml` 함수 호출에 대한 `max_rows_per_batch` 설정
- ML 함수를 실행할 때 데이터베이스가 사용할 수 있는 최대 병렬도를 결정하는 데이터베이스 인스턴스의 가상 CPU 수
- 병렬 쿼리 실행을 제어하는 PostgreSQL 파라미터

주제

- [Aurora 기계 학습 함수 호출에 대한 배치 모드 실행 최적화 \(p. 858\)](#)
- [병렬 쿼리 처리 활용 \(p. 860\)](#)
- [구체화된 보기 및 구체화된 열 사용 \(p. 861\)](#)

Aurora 기계 학습 함수 호출에 대한 배치 모드 실행 최적화

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 최소화할 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 슬러인 경우 배치 모드 실행에 적합합니다.

사용자 정의 함수는 `LANGUAGE SQL` 함수여야 하며 `PARALLEL SAFE` 및 `COST 5000`을 지정해야 합니다.

주제

- [SELECT 문에서 FROM 절로 함수 마이그레이션 \(p. 859\)](#)
- [max_rows_per_batch 파라미터 사용 \(p. 859\)](#)
- [배치 모드 실행 확인 \(p. 860\)](#)

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 계획에 중첩 루프 조인 연산자가 있으면 Aurora가 `anomaly_score` 함수를 마이그레이션했음을 알 수 있습니다. 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 `false`로 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 `apg_enable_function_migration` 설정으로 `approved` 계획을 적용합니다. 이 적용은 `apg_enable_function_migration` GUC 파라미터 설정에 관계없이 발생합니다. 자세한 내용은 [Aurora PostgreSQL용 쿼리 실행 계획 관리 \(p. 821\)](#) 단원을 참조하십시오.

max_rows_per_batch 파라미터 사용

`aws_sagemaker.invoke_endpoint` (p. 863) 및 `aws_comprehend.detect_sentiment` (p. 863) 함수의 `max_rows_per_batch` 파라미터는 Aurora 기계 학습 서비스로 전송되는 행 수에 영향을 줍니다. 사용자 정의 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 `LIMIT` 절로 인해 `SELECT` 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. EXPLAIN에서 보고한 루프 수는 `FunctionScan` 연산자에서 행을 가져온 횟수를 나타냅니다. 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 솔어에서 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 ANALYZE를 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 처리 활용

많은 수의 행을 처리할 때 성능을 크게 향상시키려면 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 AWS 서비스(예: Amazon SageMaker)의 엔드포인트로 전송됩니다. 따라서 AWS 서비스 엔드포인트 뒤에 있는 인스턴스의 수와 유형은 유용하게 활용할 수 있는 병렬 처리 정도를 제한합니다. 2코어 인스턴스 클래스조차도 병렬 쿼리 처리를 통해 상당한 이점을 얻을 수 있습니다. 그러나 더 높은 K도에서 병렬 처리를 완전히 활용하려면 K 이상의 코어가 있는 데이터베이스 인스턴스 클래스가 필요합니다. 또한 K 배치 요청을 병렬로 처리할 수 있도록 AWS 서비스를 구성해야 합니다. Amazon SageMaker의 경우 충분한 고성능 인스턴스 클래스의 K 초기 인스턴스가 있도록 ML 모델에 대해 SageMaker 엔드포인트를 구성해야 합니다.

병렬 쿼리 처리를 활용하려면 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 ML 서비스를 배치 및 병렬로 호출할 수 있습니다. aws_comprehend.detect_sentiment 함수와 함께 다음 파라미터를 사용하여 4방향 병렬 처리가 있는 계획을 얻을 수 있습니다.

```
-- If you change either of the following two parameters, you must restart
-- the database instance for the changes to take effect.
--
-- 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 ML 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 [병렬 계획](#)을 참조하십시오.

구체화된 보기 및 구체화된 열 사용

데이터베이스에서 Amazon SageMaker 또는 Amazon Comprehend와 같은 AWS 서비스를 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 지역 시간은 일반적으로 배치 모드 함수를 호출하는 지역 시간보다 훨씬 짧습니다. 따라서 애플리케이션이 수행하는 모든 단일 행 `INSERT`에 대해 배치 모드 함수를 호출하면 애플리케이션의 지역 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 충분히 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

`aws_ml` 패키지의 함수를 모니터링하려면 `track_functions` 파라미터를 설정한 다음 [PostgreSQL pg_stat_user_functions](#) 보기 쿼리를 참조합니다.

Aurora 기계 학습 함수에서 호출하는 Amazon SageMaker 작업의 성능을 모니터링하는 방법에 대한 자세한 내용은 [Amazon SageMaker](#)를 참조하세요.

세션 수준에서 `track_functions`을 설정하려면 다음을 실행합니다.

```
SET track_functions = 'all';
```

다음 값 중 하나를 사용합니다.

- all – 인라인으로 배치되지 않은 C 언어 함수 및 SQL 언어 함수를 추적합니다. aws_ml 함수는 C로 구현 되기 때문에 이러한 함수를 추적하려면 all을 사용합니다.
- pl – 절차적 언어 함수만 추적합니다.
- none – 함수 통계 추적을 비활성화합니다.

track_functions를 활성화하고 사용자 정의 ML 함수를 실행한 후 pg_stat_user_functions 보기 퀼리하여 정보를 얻습니다. 보기에는 각 함수에 대한 calls, total_time 및 self_time의 수가 포함됩니다. aws_sagemaker.invoke_endpoint 및 aws_comprehend.detect_sentiment 함수에 대한 통계를 보려면 aws_로 시작하는 스키마 이름별로 결과를 필터링합니다.

```
run your statement
SELECT * FROM pg_stat_user_functions WHERE schemaname LIKE 'aws_%';
SELECT pg_stat_reset(); -- To clear statistics
```

aws_sagemaker.invoke_endpoint 함수를 호출하는 SQL 함수의 이름을 찾으려면 [PostgreSQL pg_proc 카탈로그](#) 테이블에서 함수의 소스 코드를 퀼리합니다.

```
SELECT proname FROM pg_proc WHERE prosrc LIKE '%invoke_endpoint%';
```

쿼리 계획 관리를 사용하여 ML 함수 모니터링

쿼리 계획 관리의 apg_plan_mgmt 확장을 사용하여 계획을 캡처한 경우 이러한 함수 이름을 참조하는 워크로드의 모든 명령문을 검색할 수 있습니다. 검색에서 plan_outline을 통해 배치 모드 실행이 사용되었는지 확인할 수 있습니다. 실행 시간 및 계획 비용과 같은 명령문 통계를 나열할 수도 있습니다. 배치 모드 함수 스캔을 사용하는 계획에는 계획 개요에 FuncScan 연산자가 포함되어 있습니다. 조인으로 실행되지 않는 함수에는 FuncScan 연산자가 포함되어 있지 않습니다.

쿼리 계획 관리에 대한 자세한 내용은 [Aurora PostgreSQL용 쿼리 실행 계획 관리 \(p. 821\)](#) 단원을 참조하십시오.

배치 모드를 사용하지 않는 aws_sagemaker.invoke_endpoint 함수에 대한 호출을 찾으려면 다음 문을 사용합니다.

```
\dx apg_plan_mgmt

SELECT sql_hash, plan_hash, status, environment_variables,
       sql_text::varchar(50), plan_outline
FROM pg_proc, apg_plan_mgmt.dba_plans
WHERE
    prosrc LIKE '%invoke_endpoint%' AND
    sql_text LIKE '%' || proname || '%' AND
    plan_outline NOT LIKE '%FuncScan%';
```

앞의 예에서는 SQL 함수를 호출하여 aws_sagemaker.invoke_endpoint 함수를 호출하는 워크로드의 모든 명령문을 검색합니다.

이러한 각 명령문에 대한 자세한 런타임 통계를 얻으려면 apg_plan_mgmt.get_explain_stmt 함수를 호출합니다.

```
SELECT apg_plan_mgmt.get_explain_stmt(sql_hash, plan_hash, 'analyze,verbose,buffers')
FROM pg_proc, apg_plan_mgmt.dba_plans
WHERE
    prosrc LIKE '%invoke_endpoint%' AND
    sql_text LIKE '%' || proname || '%' AND
    plan_outline NOT LIKE '%FuncScan%';
```

Aurora 기계 학습에 대한 PostgreSQL 함수 참조

Functions

- [aws_comprehend.detect_sentiment \(p. 863\)](#)
- [aws_sagemaker.invoke_endpoint \(p. 863\)](#)

[aws_comprehend.detect_sentiment](#)

Amazon Comprehend를 사용하여 감성 분석을 수행합니다. 사용법에 대한 자세한 내용은 [자연어 처리에 Amazon Comprehend 사용 \(p. 854\)](#) 단원을 참조하십시오.

구문

```
aws_comprehend.detect_sentiment (
    IN input_text varchar,
    IN language_code varchar,
    IN max_rows_per_batch int,
    OUT sentiment varchar,
    OUT confidence real)
)
```

입력 파라미터

input_text

감성을 감지할 텍스트입니다.

language_code

input_text의 언어입니다. 유효한 값은 [Amazon Comprehend에서 지원되는 언어](#)를 참조하십시오.

max_rows_per_batch

배치 모드 처리를 위한 배치당 최대 행 수입니다. 자세한 내용은 [Aurora 기계 학습 함수 호출에 대한 배치 모드 실행 최적화 \(p. 858\)](#) 단원을 참조하십시오.

출력 파라미터

sentiment

텍스트의 감성입니다. 유효한 값은 POSITIVE, NEGATIVE, NEUTRAL 또는 MIXED입니다.

confidence

sentiment 값에 대한 신뢰도입니다. 값의 범위는 1.0(100%) ~ 0.0(0%)입니다.

[aws_sagemaker.invoke_endpoint](#)

모델을 교육하고 Amazon SageMaker 서비스를 사용하여 프로덕션 환경에 배포하면 클라이언트 애플리케이션에서 aws_sagemaker.invoke_endpoint 함수를 사용하여 모델에서 추론을 가져옵니다. 모델은 지정된 엔드포인트에서 호스팅되어야 하며 데이터베이스 인스턴스와 동일한 AWS 리전에 있어야 합니다. 사용법에 대한 자세한 내용은 [Amazon SageMaker를 사용하여 자체 ML 모델 실행 \(p. 855\)](#) 단원을 참조하십시오.

구문

```
aws_sagemaker.invoke_endpoint(
    IN endpoint_name varchar,
    IN max_rows_per_batch int,
```

```
VARIADIC model_input "any",
OUT model_output varchar
)
```

입력 파라미터

endpoint_name

AWS 리전에 독립적인 엔드포인트 URL입니다.

max_rows_per_batch

배치 모드 처리를 위한 배치당 최대 행 수입니다. 자세한 내용은 [Aurora 기계 학습 함수 호출에 대한 배치 모드 실행 최적화 \(p. 858\)](#) 단원을 참조하십시오.

model_input

ML 모델에 대한 하나 이상의 입력 파라미터입니다. 모든 데이터 형식이 될 수 있습니다.

PostgreSQL을 사용하면 함수에 대해 최대 100개의 입력 파라미터를 지정할 수 있습니다. 배열 데이터 형식은 1차원이어야 하지만 Amazon SageMaker 모델에서 예상하는 만큼 많은 요소를 포함할 수 있습니다. Amazon SageMaker 모델에 대한 입력 수는 Amazon SageMaker 메시지 크기 제한인 5MB로만 제한됩니다.

출력 파라미터

model_output

SageMaker 모델의 출력 파라미터를 텍스트로 표시합니다.

사용 시 주의사항

`aws_sagemaker.invoke_endpoint` 함수는 동일한 AWS 리전의 모델 엔드포인트에만 연결됩니다. 데이터베이스 인스턴스에 여러 AWS 리전의 복제본이 있는 경우 항상 각 Amazon SageMaker 모델을 모든 AWS 리전에 배포하십시오.

`aws_sagemaker.invoke_endpoint`에 대한 호출은 데이터베이스 인스턴스에 대한 Amazon SageMaker IAM 역할을 사용하여 인증됩니다.

Amazon SageMaker 모델 엔드포인트는 개별 계정으로 범위가 지정되며 공개되지 않습니다. `endpoint_name` URL에는 계정 ID가 포함되어 있지 않습니다. Amazon SageMaker는 데이터베이스 인스턴스의 Amazon SageMaker IAM 역할에서 제공하는 인증 토큰에서 계정 ID를 결정합니다.

AWS CLI를 사용하여 수동으로 Amazon SageMaker 및 Amazon Comprehend에 대한 IAM 역할 설정

Note

AWS Management 콘솔을 사용하는 경우 AWS는 IAM 설정을 자동으로 수행합니다. 이 경우 다음 정보를 건너뛰고 [콘솔을 사용하여 Aurora DB 클러스터를 AWS 서비스에 자동으로 연결 \(p. 852\)](#)의 절차를 따라도 됩니다.

AWS CLI 또는 RDS API를 사용하여 Amazon SageMaker 또는 Amazon Comprehend에 대해 IAM 역할을 설정하는 작업은 다음 단계로 구성됩니다.

1. IAM 정책을 생성하여 Aurora PostgreSQL 클러스터에서 호출할 수 있는 Amazon SageMaker 엔드포인트를 지정하거나 Amazon Comprehend에 액세스할 수 있게 하십시오.
2. IAM 역할을 생성하여 Aurora PostgreSQL 데이터베이스 클러스터가 AWS ML 서비스에 액세스할 수 있게 허용하십시오. 또한 앞서 생성한 IAM 정책을 여기에서 생성한 IAM 역할에 연결합니다.

3. 이전에 생성한 IAM 역할을 Aurora PostgreSQL 데이터베이스 클러스터에 연결하여 AWS ML 서비스에 대한 액세스를 허용합니다.

주제

- [AWS CLI를 사용하여 Amazon SageMaker에 액세스할 수 있는 IAM 정책 생성 \(p. 865\)](#)
- [AWS CLI를 사용하여 Amazon Comprehend에 액세스할 수 있는 IAM 정책 생성 \(p. 865\)](#)
- [Amazon SageMaker 및 Amazon Comprehend에 액세스할 수 있는 IAM 역할 생성 \(p. 866\)](#)
- [AWS CLI를 사용하여 IAM 역할을 Aurora PostgreSQL DB 클러스터와 연결 \(p. 867\)](#)

AWS CLI를 사용하여 Amazon SageMaker에 액세스할 수 있는 IAM 정책 생성

Note

Aurora에서 IAM 정책을 자동으로 생성할 수 있습니다. 다음 정보를 건너뛰고 [콘솔을 사용하여 Aurora DB 클러스터를 AWS 서비스에 자동으로 연결 \(p. 852\)](#)의 절차를 따로도 됩니다.

아래 정책은 Aurora PostgreSQL이 사용자를 대신하여 Amazon SageMaker 함수를 호출하는 데 필요한 권한을 추가합니다. 데이터베이스 애플리케이션이 Aurora PostgreSQL 클러스터에서 액세스하는 데 필요한 모든 Amazon SageMaker 엔드포인트를 하나의 정책에서 지정할 수 있습니다.

Note

이 정책을 통해 Amazon SageMaker 엔드포인트에 대해 AWS 리전을 지정할 수 있습니다. 그러나 Aurora PostgreSQL 클러스터는 클러스터와 동일한 AWS 리전에 배포된 Amazon SageMaker 모델만 호출할 수 있습니다.

```
{ "Version": "2012-10-17", "Statement": [ { "Sid": "AllowAuroraToInvokeRCFEndPoint", "Effect": "Allow", "Action": "sagemaker:InvokeEndpoint", "Resource": "arn:aws:sagemaker:region:123456789012:endpoint/endpointName" } ] }
```

다음 AWS CLI 명령은 이 옵션으로 IAM 정책을 만듭니다.

```
aws iam create-policy --policy-name policy_name --policy-document '{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToInvokeRCFEndPoint",  
            "Effect": "Allow",  
            "Action": "sagemaker:InvokeEndpoint",  
            "Resource": "arn:aws:sagemaker:region:123456789012:endpoint/endpointName"  
        }  
    ]  
}'
```

다음 단계는 [Amazon SageMaker 및 Amazon Comprehend에 액세스할 수 있는 IAM 역할 생성 \(p. 866\)](#) 단원을 참조하십시오.

AWS CLI를 사용하여 Amazon Comprehend에 액세스할 수 있는 IAM 정책 생성

Note

Aurora에서 IAM 정책을 자동으로 생성할 수 있습니다. 다음 정보를 건너뛰고 [콘솔을 사용하여 Aurora DB 클러스터를 AWS 서비스에 자동으로 연결 \(p. 852\)](#)의 절차를 따로도 됩니다.

아래 정책은 Aurora PostgreSQL이 사용자를 대신하여 AWS Amazon Comprehend를 호출하는 데 필요한 권한을 추가합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowAuroraToInvokeComprehendDetectSentiment",  
            "Effect": "Allow",  
            "Action": [  
                "comprehend:DetectSentiment",  
                "comprehend:BatchDetectSentiment"  
            ],  
            "Resource": "*"  
        }  
    ]  
}
```

Amazon Comprehend에 대한 액세스 권한을 부여하는 IAM 정책을 생성하려면

1. [IAM 관리 콘솔](#)을 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 Comprehend를 선택합니다.
5. 작업에서 Detect Sentiment(감성 감지) 및 BatchDetectSentiment를 선택합니다.
6. 정책 검토를 선택합니다.
7. 이름에서 IAM 정책의 이름을 입력합니다. IAM 역할을 만들어 DB 클러스터와 연결할 때 이 이름을 사용합니다. Description 값(선택 사항)을 추가할 수도 있습니다.
8. [Create policy]를 선택합니다.

다음 단계는 [Amazon SageMaker 및 Amazon Comprehend에 액세스할 수 있는 IAM 역할 생성 \(p. 866\)](#) 단원을 참조하십시오.

Amazon SageMaker 및 Amazon Comprehend에 액세스할 수 있는 IAM 역할 생성

Note

Aurora에서 IAM 역할을 자동으로 생성할 수 있습니다. 다음 정보를 건너뛰고 [콘솔을 사용하여 Aurora DB 클러스터를 AWS 서비스에 자동으로 연결 \(p. 852\)](#)의 절차를 따라도 됩니다.

IAM 정책을 생성한 후 ML 서비스에 액세스하기 위해 데이터베이스 사용자를 대신하여 Aurora PostgreSQL DB 클러스터가 맡을 수 있는 IAM 역할을 생성하십시오. IAM 역할을 생성하려면 [IAM 사용자에게 권한을 위임할 역할 생성](#)에 설명된 단계를 따릅니다.

앞의 정책을 생성한 IAM 역할에 연결합니다. 자세한 내용은 [IAM 정책과 IAM 사용자 또는 역할의 연결 \(p. 981\)](#) 단원을 참조하십시오.

IAM 역할에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서의 [IAM 역할](#)을 참조하십시오.

다음 단계는 [AWS CLI를 사용하여 IAM 역할을 Aurora PostgreSQL DB 클러스터와 연결 \(p. 867\)](#) 단원을 참조하십시오.

AWS CLI를 사용하여 IAM 역할을 Aurora PostgreSQL DB 클러스터와 연결

Note

Aurora에서 DB 클러스터에 IAM 역할을 자동으로 연결할 수 있습니다. 다음 정보를 건너뛰고 [콘솔을 사용하여 Aurora DB 클러스터를 AWS 서비스에 자동으로 연결 \(p. 852\)](#)의 절차를 따라도 됩니다.

IAM 액세스를 설정하는 마지막 프로세스는 IAM 역할 및 IAM 정책을 Aurora PostgreSQL DB 클러스터에 연결하는 것입니다. 다음을 수행합니다.

1. DB 클러스터에 연결된 역할 목록에 역할을 추가합니다.

역할을 DB 클러스터와 연결하려면 AWS Management 콘솔 또는 `add-role-to-db-cluster` AWS CLI 명령을 사용합니다.

- 콘솔을 사용하여 PostgreSQL DB 클러스터에 대해 IAM 역할을 추가하려면

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 세부 정보를 표시하고자 하는 PostgreSQL DB 클러스터 이름을 선택합니다.
3. Connectivity & security(연결성 및 보안) 탭에 있는 IAM 역할 관리 섹션의 이 클러스터에 IAM 역할 추가에서 추가할 역할을 선택합니다.
4. 기능에서 SageMaker 또는 Comprehend를 선택합니다.
5. [Add role]을 선택합니다.

- 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. `--feature-name` 옵션 값에 대해 사용하고자 하는 서비스에 따라 SageMaker, Comprehend 또는 s3Export를 사용합니다.

Example

Linux, OS X, Unix의 경우:

```
aws rds add-role-to-db-cluster \
--db-cluster-identifier my-db-cluster \
--feature-name external-service \
--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 external-service ^
--role-arn your-role-arn ^
--region your-region
```

2. 각 AWS ML 서비스에 대한 클러스터 수준 파라미터를 연결된 IAM 역할의 ARN으로 설정합니다.

Aurora 클러스터에서 사용하려는 AWS ML 서비스에 따라 `electroencephalographic` 또는 `miscomprehended` 파라미터를 사용하거나 이 두 파라미터를 모두 사용하십시오.

클러스터 수준 파라미터는 DB 클러스터 파라미터 그룹으로 그룹화됩니다. 이전 클러스터 파라미터를 설정하려면 기존 사용자 지정 DB 클러스터 그룹을 사용하거나 새로운 그룹을 생성하십시오. 새 DB 클러스터

터 파라미터 그룹을 만들려면 다음과 같이 AWS CLI에서 `create-db-cluster-parameter-group` 명령을 호출하십시오.

```
aws rds create-db-cluster-parameter-group --db-cluster-parameter-group-name AllowAWSAccessToExternalServices \
    --db-parameter-group-family aurora-postgresql-group --description "Allow access to
Amazon S3, Amazon SageMaker, and Amazon Comprehend"
```

DB 클러스터 파라미터 그룹에서 적절한 클러스터 수준 파라미터 및 관련 IAM 역할 ARN 값을 설정합니다. 다음을 수행합니다.

```
aws rds modify-db-cluster-parameter-group \
    --db-cluster-parameter-group-name AllowAWSAccessToExternalServices \
    --parameters
    "ParameterName=aws_default_s3_role,ParameterValue=arn:aws:iam::123456789012:role/
    AllowAuroraS3Role,ApplyMethod=pending-reboot" \
    --parameters
    "ParameterName=aws_default_sagemaker_role,ParameterValue=arn:aws:iam::123456789012:role/
    AllowAuroraSageMakerRole,ApplyMethod=pending-reboot" \
    --parameters
    "ParameterName=aws_default_comprehend_role,ParameterValue=arn:aws:iam::123456789012:role/
    AllowAuroraComprehendRole,ApplyMethod=pending-reboot"
```

새로운 DB 클러스터 파라미터 그룹을 사용하도록 DB 클러스터를 수정합니다. 그런 다음 클러스터를 재부팅합니다. 아래에서는 이 작업을 수행하는 방법을 보여줍니다.

```
aws rds modify-db-cluster --db-cluster-identifier your_cluster_id --db-cluster-parameter-
group-nameAllowAWSAccessToExternalServices
aws rds failover-db-cluster --db-cluster-identifier your_cluster_id
```

인스턴스가 재부팅되면 IAM 역할이 DB 클러스터와 연결됩니다.

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.

주제

- [클러스터 캐시 관리 구성 \(p. 869\)](#)
- [버퍼 캐시 모니터링 \(p. 871\)](#)

클러스터 캐시 관리 구성

Note

Aurora PostgreSQL DB 클러스터 버전 9.6.11 이상과 버전 10.5 이상에서는 클러스터 캐시 관리 기능이 지원됩니다.

클러스터 캐시 관리를 구성하려면 다음 절차를 밟아야 합니다.

구성 절차

- [클러스터 캐시 관리 활성화 \(p. 869\)](#)
- [라이터 DB 인스턴스에 대해 승격 티어 우선 순위를 설정합니다. \(p. 870\)](#)
- [리더 DB 인스턴스에 대한 승격 티어 우선 순위 설정 \(p. 871\)](#)

Note

구성 절차를 마친 후 클러스터 캐시 관리 기능이 완전하게 실행되려면 최소 1분이 지나야 합니다.

클러스터 캐시 관리 활성화

DB 클러스터에서 클러스터 캐시 관리를 활성화하려면 다음과 같이 `apg_ccm_enabled` 파라미터를 1로 설정하여 파라미터 그룹을 수정합니다.

콘솔

클러스터 캐시 관리를 활성화하려면

1. AWS Management 콘솔에 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 활성화하려면 다음 필수 파라미터와 함께 AWS CLI `modify-db-cluster-parameter-group` 명령을 사용하십시오.

- `--db-cluster-parameter-group-name`
- `--parameters`

Example

Linux, OS X, Unix의 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"
```

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 대해 승격 티어 우선 순위를 설정합니다.

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. 유 효한 값은 0-15이며, 여기에서 0은 최우선 순위이고 15는 마지막 우선 순위입니다. 승격 티어에 대한 자세한 내용은 [Aurora DB 클러스터의 내결합성 \(p. 268\)](#) 단원을 참조하십시오.

콘솔

라이터 DB 인스턴스에 대해 승격 우선 순위를 설정하려면

1. AWS Management 콘솔에 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. 장애 조치 패널에서 우선 순위를 tier-0으로 선택합니다.
6. 계속을 선택하고 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

Linux, OS X, Unix의 경우:

```
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 인스턴스 한 개를 설정합니다. 이를 위해서는 라이터 DB 인스턴스와 동일한 인스턴스 클래스인 Aurora PostgreSQL 클러스터에서 리더를 선택해야 합니다. 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 최우선 순위이고 15는 마지막 우선 순위입니다.

콘솔

라이터 DB 인스턴스의 승격 우선 순위를 tier-0으로 설정하려면

1. AWS Management 콘솔에 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. 장애 조치 패널에서 우선 순위를 tier-0으로 선택합니다.
6. 계속을 선택하고 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` 명령을 호출하십시오.

- `--db-instance-identifier`
- `--promotion-tier`
- `--apply-immediately`

Example

Linux, OS X, Unix의 경우:

```
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 있

는 버퍼 캐시 콘텐츠를 검토하려면 PostgreSQL pg_buffercache 모듈을 사용하십시오. 자세한 내용은 [PostgreSQL pg_buffercache 설명서](#)를 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_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/buffers_found_last_scan) AS warm_percent
  FROM aurora_ccm_status();
```

Aurora PostgreSQL용 PostgreSQL DB 엔진 업그레이드

Aurora PostgreSQL에서 새 버전의 데이터베이스 엔진을 지원하는 경우 DB 클러스터를 새 버전으로 업그레이드할 수 있습니다. PostgreSQL DB 클러스터의 업그레이드에는 메이저 버전 업그레이드와 마이너 버전 업그레이드라는 두 가지 종류가 있습니다.

메이저 버전 업그레이드에는 기존 애플리케이션과 호환되지 않는 데이터베이스 변경 사항이 포함될 수 있습니다. 따라서 DB 인스턴스의 메이저 버전 업그레이드를 수동으로 수행해야 합니다. DB 클러스터를 수정하여 메이저 버전 업그레이드를 시작할 수 있습니다. 그러나 메이저 버전 업그레이드를 수행하기 전에 [메이저 버전 업그레이드를 준비하고 수행하는 방법 \(p. 874\)](#)에 설명된 단계를 수행하는 것이 좋습니다.

반대로 마이너 버전 업그레이드에는 기존 애플리케이션과 호환되는 변경 사항만 포함됩니다. DB 클러스터를 수정하여 마이너 버전 업그레이드를 수동으로 시작할 수 있습니다. 또는 DB 클러스터를 생성하거나 수정할 때 마이너 버전 자동 업그레이드 옵션을 활성화할 수 있습니다. 이렇게 하면 Aurora PostgreSQL에서 새 버전을 테스트 및 승인한 후 DB 클러스터가 자동으로 업그레이드됩니다. 자세한 내용은 [PostgreSQL 마이너 버전 자동 업그레이드 \(p. 877\)](#) 단원을 참조하십시오. 마이너 버전 업그레이드를 수동으로 수행하는 방법에 대한 자세한 내용은 [Aurora PostgreSQL 엔진 수동 업그레이드 \(p. 876\)](#) 단원을 참조하십시오.

Note

- 현재 Aurora PostgreSQL 9.6 마이너 버전 9.6.9 이상에서 Aurora PostgreSQL 10 마이너 버전 10.11 이상으로 메이저 버전 업그레이드를 수행할 수 있습니다.
- 논리적 복제 게시자 또는 구독자로 구성된 Aurora DB 클러스터의 경우 메이저 버전 업그레이드를 수행할 수 없습니다. 자세한 내용은 [Aurora에서 PostgreSQL 논리적 복제 사용 \(p. 808\)](#) 단원을 참조하십시오.

주제

- [Aurora PostgreSQL 업그레이드 개요 \(p. 873\)](#)

- PostgreSQL DB 클러스터 버전 식별 (p. 873)
- 메이저 버전 업그레이드를 준비하고 수행하는 방법 (p. 874)
- Aurora PostgreSQL 엔진 수동 업그레이드 (p. 876)
- PostgreSQL 마이너 버전 자동 업그레이드 (p. 877)
- PostgreSQL 확장 버전 업그레이드 (p. 877)

Aurora PostgreSQL 업그레이드 개요

메이저 버전 업그레이드에는 이전 버전의 데이터베이스와 호환되지 않는 데이터베이스 변경 사항이 포함될 수 있습니다. 이 기능을 사용하면 기존 애플리케이션이 올바르게 작동하지 않을 수 있습니다. 따라서 Amazon Aurora는 자동으로 메이저 버전 업그레이드를 적용하지 않습니다. 메이저 버전 업그레이드를 수행하려면 DB 클러스터를 수동으로 수정합니다.

DB 인스턴스를 안전하게 업그레이드하기 위해 Aurora PostgreSQL은 [PostgreSQL 설명서](#)에 설명된 pg_upgrade 유틸리티를 사용합니다. 라이터 업그레이드가 완료되면 각 리더 인스턴스는 새 메이저 버전으로 자동 업그레이드되는 동안 잠시 중단됩니다.

Aurora PostgreSQL은 업그레이드가 시작되기 전에 DB 클러스터 스냅샷을 생성합니다. 업그레이드가 완료된 후 이전 버전으로 돌아가려면 업그레이드 전에 생성된 이 스냅샷에서 DB 클러스터를 복원하거나, 업그레이드가 시작되기 전의 특정 시점으로 DB 클러스터를 복원할 수 있습니다. 자세한 내용은 [DB 클러스터 스냅샷에서 복원 \(p. 273\)](#) 또는 [DB 클러스터를 지정된 시간으로 복원 \(p. 303\)](#) 단원을 참조하십시오.

메이저 버전 업그레이드 프로세스 중에 복제된 볼륨이 할당됩니다. 스키마 비호환성 등의 이유로 업그레이드가 실패할 경우 Aurora PostgreSQL에서는 이 복제를 사용하여 업그레이드를 롤백합니다. 원본 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 유의하십시오.

- 업그레이드 중에 할당된 원래 볼륨과 복제된 볼륨 모두에 대한 결제 항목 및 지표가 표시될 수 있습니다. Aurora PostgreSQL은 클러스터 백업 보존 기간이 업그레이드 시간을 초과한 후 추가 볼륨을 정리합니다.
- 이 클러스터의 다음 리전 간 스냅샷 복사본은 충분 복사본 대신 전체 복사본입니다.

PostgreSQL DB 클러스터 버전 식별

DB 클러스터의 현재 Aurora PostgreSQL 버전 및 PostgreSQL 데이터베이스 엔진 버전을 확인하려면 [Amazon Aurora PostgreSQL의 해당 버전 식별 \(p. 906\)](#) 단원을 참조하십시오.

PostgreSQL 데이터베이스 엔진의 버전 번호 매기기 순서는 다음과 같습니다.

- PostgreSQL 버전 10 이상의 경우 엔진 버전 번호는 major.minor형식입니다. 메이저 버전 번호는 버전 번호의 정수 부분입니다. 마이너 버전 번호는 버전 번호의 소수 부분입니다.

메이저 버전 업그레이드는 10.minor에서 11.minor로의 업그레이드와 같이 버전 번호의 정수 부분이 증가합니다.
- PostgreSQL 버전 10 이전의 경우 엔진 버전 번호는 major.minor형식입니다. 버전 번호의 정수 부분과 첫 번째 소수 부분 모두가 메이저 엔진 버전 번호입니다. 예를 들어, 9.6이 메이저 버전입니다. 버전 번호의 세 번째 부분이 마이너 버전 번호입니다. 예를 들어, 버전 9.6.12의 경우 12가 마이너 버전 번호입니다.

메이저 버전 업그레이드는 버전 번호의 메이저 부분이 증가합니다. 예를 들어, 9.6.12에서 10.7로 업그레이드하는 것은 메이저 버전 업그레이드입니다. 여기서 9.6과 10이 메이저 버전 번호입니다.

DB를 업그레이드할 수 있는 메이저 엔진 버전을 확인하려면 `describe-db-engine-versions` CLI 명령을 사용합니다. 예를 들어, 다음 명령은 현재 Aurora PostgreSQL 엔진 버전 9.6.12를 실행 중인 DB 클러스터를 업그레이드하는 데 사용할 수 있는 메이저 엔진 버전을 표시합니다.

Example

Linux, OS X, Unix의 경우:

```
aws rds describe-db-engine-versions --engine aurora-postgresql --engine-version 9.6.12 \
--query 'DBEngineVersions[].[ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`]]'
```

Windows의 경우:

```
aws rds describe-db-engine-versions --engine aurora-postgresql --engine-version 9.6.12 ^
--query "DBEngineVersions[].[ValidUpgradeTarget[?IsMajorVersionUpgrade == `true`]]"
```

메이저 버전 업그레이드를 준비하고 수행하는 방법

메이저 버전 업그레이드에는 이전 버전의 데이터베이스와 호환되지 않는 데이터베이스 변경 사항이 포함될 수 있습니다. 이 기능을 사용하면 기존 애플리케이션이 올바르게 작동하지 않을 수 있습니다. 따라서 Amazon Aurora는 자동으로 메이저 버전 업그레이드를 적용하지 않습니다. 메이저 버전 업그레이드를 수행하려면 DB 클러스터를 수동으로 수정합니다.

프로덕션 DB 클러스터에 업그레이드를 적용하기 전에 모든 업그레이드를 철저하게 테스트하여 애플리케이션이 올바르게 작동하는지 확인해야 합니다.

Aurora PostgreSQL DB 클러스터를 업그레이드할 때는 다음 절차를 수행하는 것이 좋습니다.

1. 버전 호환 파라미터 그룹 준비 – 사용자 지정 DB 인스턴스 또는 DB 클러스터 파라미터 그룹을 사용하는 경우 다음 두 가지 옵션이 있습니다.

- 새 DB 엔진 버전에 대한 기본 DB 인스턴스 및/또는 DB 클러스터 파라미터 그룹을 지정할 수 있습니다.
- 또는 새 DB 엔진 버전에 대한 사용자 지정 파라미터 그룹을 직접 만들 수도 있습니다.

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 인스턴스를 재부팅해야 하는 경우 인스턴스의 파라미터 그룹 상태가 pending-reboot로 표시됩니다. 인스턴스의 파라미터 그룹 상태는 콘솔에서 보거나 [describe-db-instances](#) 또는 [describe-db-clusters](#) 같은 CLI 명령을 사용하여 볼 수 있습니다.

2. 지원되지 않는 사용 확인:

- 준비된 트랜잭션 – 업그레이드하기 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;
```

- Reg* 데이터 형식 – 업그레이드를 시도하기 전에 reg* 데이터 형식의 사용을 모두 제거하십시오. regtype 및 regclass 이외에는 reg* 데이터 형식을 업그레이드할 수 없습니다. pg_upgrade 유ти리티는 Amazon Aurora에서 업그레이드를 수행하는 데 사용하는 이 데이터 형식을 유지할 수 없습니다. pg_upgrade 유ти리티에 대한 자세한 내용은 [PostgreSQL 설명서](#)를 참조하십시오.

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');
```

3. 백업 수행 – 업그레이드 프로세스는 업그레이드 중에 DB 클러스터의 DB 클러스터 스냅샷을 생성합니다. 업그레이드 프로세스 전에 수동 백업을 수행하려면 [DB 클러스터 스냅샷 생성 \(p. 271\)](#) 단원을 참조하십시오.
4. 알 수 없는 데이터 형식 삭제 – 대상 버전에 따라 unknown 데이터 형식을 삭제합니다.

PostgreSQL 버전 10은 unknown 데이터 형식에 대한 지원이 중지되었습니다. 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."
```

문제가 되는 열을 제거하거나 지원되는 데이터 형식으로 변경할 수 있도록 데이터베이스에서 unknown 데이터 형식을 찾으려면 다음 SQL을 사용합니다.

```
SELECT DISTINCT data_type FROM information_schema.columns WHERE data_type ILIKE  
'unknown';
```

5. 테스트 실행 업그레이드 수행 – 프로덕션 데이터베이스에서 업그레이드를 수행하기 전에 프로덕션 데이터베이스의 복제본에서 메이저 버전 업그레이드를 테스트하는 것이 좋습니다. 종종 테스트 인스턴스를 생성하려면 최근 스냅샷에서 데이터베이스를 복원하거나 데이터베이스를 복제합니다. 자세한 내용은 [스냅샷에서 복구 \(p. 274\)](#) 또는 [Aurora DB 클러스터에서 데이터베이스 복제 \(p. 239\)](#) 단원을 참조하십시오.

테스트 실행 업그레이드를 수행하려면 [Aurora PostgreSQL 엔진 수동 업그레이드 \(p. 876\)](#) 단원을 참조하십시오.

6. 프로덕션 인스턴스 업그레이드 – 메이저 버전 업그레이드의 모의 실습이 성공한 경우 확신을 가지고 프로덕션 데이터베이스를 업그레이드해도 됩니다. 자세한 내용은 [Aurora PostgreSQL 엔진 수동 업그레이드 \(p. 876\)](#) 단원을 참조하십시오.

Note

업그레이드 프로세스 중에는 클러스터의 특정 시점으로 복원을 수행할 수 없습니다. Aurora PostgreSQL은 백업 보존 기간이 0보다 큰 경우 업그레이드 프로세스 중에 DB 클러스터 스냅샷을 생성합니다. 업그레이드가 시작되기 전과 인스턴스 자동 스냅샷이 완료된 후 시간으로 백업을 완료한 이후의 시간으로 특정 시점으로 복원을 수행할 수 있습니다.

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.log입니다. Amazon Aurora는 이러한 로그의 파일 이름에 타임스탬프를 추가합니다. 다른 로그와 마찬가지로 이러한 로그를 볼 수 있습니다. 자세한 내용은 [Amazon Aurora 데이터베이스 로그 파일 \(p. 449\)](#) 단원을 참조하십시오.

7. PostgreSQL 확장 업그레이드 – PostgreSQL 업그레이드 프로세스는 PostgreSQL 확장을 업그레이드하지 않습니다. 자세한 내용은 [PostgreSQL 확장 버전 업그레이드 \(p. 877\)](#) 단원을 참조하십시오.
8. 메이저 버전 업그레이드 완료 후 – 메이저 버전 업그레이드가 완료된 후에는 다음을 수행하는 것이 좋습니다.
 - ANALYZE 작업을 실행하여 pg_statistic 테이블을 새로 고칩니다.
 - 모든 것이 예상대로 작동하는지 확인하기 위해 유사한 워크로드로 업그레이드된 데이터베이스에서 애플리케이션을 테스트하는 것이 좋습니다. 업그레이드를 확인한 후 이 테스트 인스턴스를 삭제할 수 있습니다.

Aurora PostgreSQL 엔진 수동 업그레이드

Aurora PostgreSQL DB 클러스터의 업그레이드를 수행하려면 AWS Management 콘솔, AWS CLI 또는 RDS API에 대한 다음 지침을 사용합니다.

콘솔

콘솔을 사용하여 DB 클러스터의 엔진 버전을 업그레이드하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택한 다음 업그레이드하려는 DB 클러스터를 선택합니다.
3. [Modify]를 선택합니다. Modify DB cluster(DB 클러스터 수정) 페이지가 나타납니다.
4. DB 엔진 버전에서 새 버전을 선택합니다.
5. [Continue]를 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 \(p. 193\)](#) 단원을 참조하십시오.
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 CLI `modify-db-cluster` 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

Linux, OS X, Unix의 경우:

```
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.

PostgreSQL 마이너 버전 자동 업그레이드

DB 클러스터를 생성하거나 수정할 때 마이너 버전 자동 업그레이드 옵션을 활성화하면 자동으로 클러스터가 업그레이드될 수 있습니다.

각 PostgreSQL 메이저 버전에 대해 마이너 버전 한 개가 Amazon Aurora에서 자동 업그레이드 버전으로 지정됩니다. Amazon Aurora가 마이너 버전을 테스트하고 승인하면 유지 관리 기간 중에 자동으로 마이너 버전 업그레이드가 실행됩니다. Aurora는 새로 릴리스된 마이너 버전을 자동 업그레이드 버전으로 자동 설정하지 않습니다. Aurora가 최신 자동 업그레이드 버전을 지정하기 전에 다음과 같은 기준이 고려됩니다.

- 알려진 보안 문제
- PostgreSQL 커뮤니티 버전의 버그
- 마이너 버전 릴리스 이후 전반적인 플랫 안정성

다음 AWS CLI 명령 및 스크립트를 사용하여 현재 자동 업그레이드 마이너 버전을 결정할 수 있습니다.

```
aws rds describe-db-engine-versions --engine aurora-postgresql | grep -A 1 AutoUpgrade|  
grep -A 2 true |grep PostgreSQL | sort --unique | sed -e 's/"Description": "//g'
```

PostgreSQL DB 인스턴스는 다음 기준이 충족되면 유지 관리 기간 중에 자동으로 업그레이드됩니다.

- DB 클러스터에 마이너 버전 자동 업그레이드 옵션이 활성화되어 있습니다.
- DB 클러스터에서 현재 자동 업그레이드 마이너 버전보다 낮은 마이너 DB 엔진 버전을 실행 중입니다.

PostgreSQL 확장 버전 업그레이드

PostgreSQL 엔진 업그레이드에서는 PostgreSQL 확장 버전을 업그레이드하지 않습니다. 엔진 업그레이드 후 확장 버전을 업데이트하려면 `ALTER EXTENSION UPDATE` 명령을 사용합니다.

Note

Amazon RDS PostgreSQL DB 인스턴스에서 PostGIS 확장을 실행하는 경우 확장 버전을 업그레이드하기 전에 PostGIS 설명서의 [PostGIS 업그레이드 지침](#)을 따르십시오.

확장 버전을 업그레이드하려면 다음 명령을 사용하십시오.

```
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](#) 보기 사용하십시오.

```
SELECT * FROM pg_available_extension_versions;
```

Amazon Aurora PostgreSQL 모범 사례

이 주제에서는 Amazon Aurora PostgreSQL DB 클러스터 사용 또는 이 클러스터로의 데이터 마이그레이션과 관련된 모범 사례와 옵션에 대해 설명합니다.

Amazon Aurora PostgreSQL를 사용한 빠른 장애 조치

Aurora PostgreSQL을 사용하여 빠른 장애 조치를 실행하는 몇 가지 방법이 있습니다. 이 단원에서는 다음과 같은 방법을 다룹니다.

- TCP keepalive를 적극적으로 설정하여, 서버 응답을 기다리며 장기 실행 중인 쿼리가 장애 발생 시 읽기 제한 시간이 만료되기 전 제거되도록 합니다.
- Java DNS 캐싱 제한 시간을 적극적으로 설정하여, Aurora 읽기 전용 엔드포인트가 후속 연결 시도에서 읽기 전용 노드를 적절히 순환할 수 있도록 합니다.
- JDBC 연결 문자열에 사용되는 제한 시간 변수를 가능한 낮게 설정합니다. 단기 및 장기 실행 중인 쿼리에 별도의 연결 객체를 사용합니다.
- 제공된 읽기 및 쓰기 Aurora 엔드포인트를 사용하여 클러스터에 대한 연결을 설정합니다.
- RDS API를 사용하여 서버 측 장애에 대한 애플리케이션 응답을 테스트하고 패킷 폐기 도구를 사용하여 클라이언트 측 장애에 대한 애플리케이션 응답을 테스트합니다.

TCP Keepalives 파라미터 설정

TCP keepalive 프로세스는 단순합니다. TCP 연결을 설정하면 타이머 집합이 연결됩니다. keepalive 타이머가 0에 도달하면 keepalive 프로브 패킷을 전송합니다. keepalive 프로브에 대한 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. 이 작업을 통해 애플리케이션이 연결할 새 호스트를 선택하는 등 적절히 대응할 수 있습니다.

다음 TCP keepalive 파라미터를 설정해야 합니다.

- `tcp_keepalive_time`은 시간을 초 단위로 제어하며, 이후 소켓으로부터 데이터가 전송되지 않을 경우 (ACK는 데이터로 간주되지 않음) keepalive 패킷이 전송됩니다. 다음 설정을 권장합니다.

```
tcp_keepalive_time = 1
```

- `tcp_keepalive_intvl`은 초기 패킷이 전송된 후 후속 keepalive 패킷을 전송하는 시간 주기를 초 단위로 제어합니다(`tcp_keepalive_time` 파라미터를 사용해 설정). 다음 설정을 권장합니다.

```
tcp_keepalive_intvl = 1
```

- `tcp_keepalive_probes`는 애플리케이션에 알림이 전달되기 전 발생하는 승인되지 않은 keepalive 프로브의 개수입니다. 다음 설정을 권장합니다.

```
tcp_keepalive_probes = 5
```

이 설정은 데이터베이스가 응답을 종단하고 5초 안에 애플리케이션에 알려야 합니다. `keepalive` 패킷이 애플리케이션의 네트워크 내에서 자주 폐기되는 경우, `tcp_keepalive_probes` 값을 높게 설정할 수 있습니다. 이렇게 하면 실제 장애를 탐지하는 데 걸리는 시간이 늘어나지만 비교적 안정적이지 못한 네트워크에서 버퍼가 추가될 수 있습니다.

Linux에서 TCP keepalive 파라미터 설정

1. TCP keepalive 파라미터의 구성 방법을 테스트할 때는 다음 명령을 명령줄에 입력하는 방법이 좋습니다. 여기에서 제안하는 구성은 시스템 전체에 해당하는 것으로, `SO_KEEPALIVE` 옵션을 켜고 소켓을 생성하는 다른 모든 애플리케이션에 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, `/etc/sysctl.conf`에 다음 줄(사용자가 변경한 내용 포함)을 추가하여 이러한 설정을 유지해야 합니다.

```
tcp_keepalive_time = 1
tcp_keepalive_intvl = 1
tcp_keepalive_probes = 5
```

Windows에서 TCP keepalive 파라미터를 설정하는 방법은 [Things You May Want to Know About TCP Keepalive](#)를 참조하십시오.

애플리케이션에 빠른 장애 조치 구성

이 단원에서는 변경이 가능한 몇 가지 Aurora PostgreSQL 구성에 대해 설명합니다. JDBC 드라이버의 일반 설정과 구성에 대한 설명서는 [PostgreSQL JDBC 사이트](#)에서 확인할 수 있습니다.

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 TTL 값을 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 활용하려면 애플리케이션의 연결 문자열에 단일 호스트가 아닌 호스트 목록(다음 예제에서 굵게 강조 표시)이 있어야 합니다. 다음은 Aurora PostgreSQL 클러스터에 연결하는데 사용할 수 있는 연결 문자열의 예입니다.

```
jdbc:postgresql://myauroracluster.cluster-c9bfei4hjlr.us-east-1-
beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hjlr.us-east-1-beta.rds.amazonaws.com:5432
/postgres?user=<masteruser>&password=<masterpw>&loginTimeout=2
&connectTimeout=2&cancelSignalTimeout=2&socketTimeout=60
&tcpKeepAlive=true&targetServerType=master&loadBalanceHosts=true
```

최고의 가용성을 보장하고 RDS API에 대한 종속성을 피하는 최적의 연결 옵션은 데이터베이스에 대한 연결을 설정할 때 애플리케이션이 읽어 들일 호스트 문자열로 파일을 유지하는 것입니다. 이 호스트 문자열에는 클러스터에 제공된 모든 Aurora 엔드포인트가 있을 것입니다. Aurora 엔드포인트에 대한 자세한 내용은 [Amazon Aurora 연결 관리 \(p. 9\)](#) 단원을 참조하십시오. 예를 들어, 다음과 같이 파일에 로컬로 엔드포인트를 저장할 수 있습니다.

```
myauroracluster.cluster-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432,  
myauroracluster.cluster-ro-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432
```

애플리케이션이 이 파일에서 읽어 들여 JDBC 연결 문자열의 호스트 섹션을 채울 것입니다. DB 클러스터 이름을 바꾸면 이 엔드포인트가 변경됩니다. 애플리케이션이 이 이벤트가 발생하도록 처리하는지 확인하십시오.

다른 옵션은 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. 반면 Aurora 엔드포인트를 사용하면 연결 시도 한 회당 두 개의 노드만 시도합니다. DB 인스턴스 사용의 단점은 사용자가 클러스터에 노드를 추가하거나 제거하고 인스턴스 엔드포인트 목록이 무효로 되면 연결 드라이버가 연결할 올바른 호스트를 찾을 수 없다는 것입니다.

다음 파라미터를 적극적으로 설정하면 애플리케이션을 한 호스트에 연결하기 위해 너무 오래 기다리지 않도록 할 수 있습니다.

- `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 것을 고려해야 합니다.
- `socketTimeout` - 이 파라미터는 소켓이 읽기 작업을 대기하는 시간을 제어합니다. 쿼리가 이 값보다 길게 대기하지 않도록 이 파라미터를 전역 "쿼리 제한 시간"으로 사용할 수 있습니다. 한 가지 좋은 방법은, 수명이 짧은 쿼리를 실행하는 연결 핸들러를 두고 이 값을 낮게 설정하는 한편, 장기 실행 쿼리를 위한 다른 연결 핸들러를 두고 이 값을 훨씬 높게 설정하는 것입니다. 그러면 서버가 다운될 경우 TCP keepalive 파라미터를 이용해 장기 실행 쿼리를 삭제할 수 있습니다.
- `tcpKeepAlive` - 이 파라미터를 활성화하면 사용자가 설정한 TCP keepalive 파라미터가 적용됩니다.
- `targetServerType` - 이 파라미터를 사용하면 드라이버의 읽기(슬레이브) 또는 쓰기(마스터) 노드 연결 여부를 제어할 수 있습니다. 가능한 값은 `any`, `master`, `slave`, `preferSlave`입니다. `preferSlave` 값은 먼저 리더와의 연결 설정을 시도하지만 리더 연결을 설정할 수 없는 경우 폴백하고 라이터에 연결합니다.
- `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.

애플리케이션이 DB 클러스터의 DB 인스턴스에 연결하고 `aurora_replica_status` 함수를 쿼리하여 클러스터 라이터가 누구인지 파악하거나 클러스터에서 다른 리더 노드를 찾아낼 수 있습니다. 이 함수를 사용하면 연결할 호스트를 찾는데 걸리는 시간을 줄일 수 있으나 일부 시나리오에서 `aurora_replica_status` 함수가 일부 네트워크 장애 시나리오에서 오래되거나 불완전한 정보를 표시할 가능성이 있습니다.

애플리케이션이 연결할 노드를 찾도록 보장하는 좋은 방법 중 하나는 클러스터 라이터엔드포인트에 대한 연결을 시도한 후 읽기 가능한 연결을 설정할 수 있을 때까지 클러스터 리더엔드포인트 연결을 시도하는 것입니다. 사용자가 DB 클러스터 이름을 바꾸지 않는 한 이러한 엔드포인트는 변경되지 않기 때문에, 일반적으로 애플리케이션의 정적 멤버로 남아 있거나 애플리케이션이 읽어 들이는 리소스 파일에 저장될 수 있습니다.

이러한 엔드포인트 중 하나를 사용하여 연결을 설정하면 `aurora_replica_status` 함수를 호출하여 나머지 클러스터에 대한 정보를 가져올 수 있습니다. 예를 들어, 다음 명령은 `aurora_replica_status` 함수를 사용해 정보를 가져옵니다.

```
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           |
  vdl          | highest_lsn_rcvd | cur_replay_latency |
 now          |           last_update_time |
-----+-----+-----+-----+
-----+-----+-----+-----+
-----+-----+
      mynode-1 | 3e3c5044-02e2-11e7-b70d-95172646d6ca | 594220999 | 594221001 | 201421 | 2017-03-07
19:50:24.695322+00 | 2017-03-07 19:50:23+00
      mynode-2 | 1efdd188-02e4-11e7-becd-f12d7c88a28a | 594220999 | 594221001 | 201350 | 2017-03-07
19:50:24.695322+00 | 2017-03-07 19:50:23+00
      mynode-3 |           MASTER_SESSION_ID |
  594220999 |                           | 2017-03-07
19:50:24.695322+00 | 2017-03-07 19:50:23+00
(3 rows)
```

따라서 이를테면 연결 문자열의 호스트 섹션은 라이터와 리더 클러스터 엔드포인트로 시작할 수 있습니다.

```
myauroracluster.cluster-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432,
myauroracluster.cluster-ro-c9bfei4hj1rd.us-east-1-beta.rds.amazonaws.com:5432
```

이 시나리오에서 애플리케이션은 마스터나 슬레이브 노드 유형에 연결을 설정하려 할 것입니다. 연결이 설정되면 먼저 `SHOW transaction_read_only` 명령 결과에 대한 쿼리를 실행하여 노드의 읽기-쓰기 상태를 검사하는 것이 좋습니다.

쿼리의 반환 값이 OFF인 경우 마스터 노드에 성공적으로 연결된 것입니다. 반환 값이 ON이고, 애플리케이션에 읽기-쓰기 연결이 필요한 경우 `aurora_replica_status` 함수를 호출하여 `server_id`가 있는 `session_id='MASTER_SESSION_ID'`를 알아낼 수 있습니다. 이 함수는 마스터 노드의 이름을 반환합니다. 이 방법은 아래 설명된 'endpointPostfix'와 결합하여 사용할 수 있습니다.

한 가지 주의해야 할 점은 무효로 된 데이터가 있는 복제본에 연결할 가능성이 있다는 것입니다. 이 경우 `aurora_replica_status` 함수에 오래된 정보가 표시될 수 있습니다. 애플리케이션 수준에서 무효 임계값을 설정할 수 있으며, 서버 시간과 `last_update_time` 사이의 차이를 통해 확인할 수 있습니다. 일반적으로 애플리케이션은 `aurora_replica_status` 함수에 의해 반환된 충돌 정보로 인한 두 호스트 사이의 플립플롭을 방지해야 합니다. 다시 말해 애플리케이션이 `aurora_replica_status` 함수에 의해 반환된 데이터를 맹목적으로 따르는 대신 알려진 모든 호스트를 먼저 시도해야 합니다.

RDS API

[AWS Java SDK](#), 특히 `DescribeDbClusters` API를 사용하여 프로그래밍 방식으로 인스턴스 목록을 찾을 수 있습니다. java 8에서 이와 같이 검색하는 방법의 예는 다음과 같습니다.

```
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(", "));
```

pgJDBCEndpointStr에 다음과 같은 형식의 앤드포인트 목록이 포함됩니다.

```
my-node1.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432,
my-node2.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com:5432
```

'endpointPostfix' 변수는 애플리케이션이 설정하는 상수이거나, 클러스터의 단일 인스턴스에 대한 DescribeDBInstances API를 쿼리하여 가져올 수 있습니다. 이 값은 한 리전 내에서 개별 고객에 대해 상수로 유지되므로 API 호출을 저장하여 애플리케이션이 읽어 들이는 리소스 파일에서 이 상수를 유지할 것입니다. 위 예제에서는 다음과 같이 설정됩니다.

```
.cksc6xlmwcyw.us-east-1-beta.rds.amazonaws.com
```

가용성을 보장하기 위해 API가 응답하지 않거나 응답 시간이 너무 길 경우 DB 클러스터의 [Aurora 앤드포인트](#)를 사용하는 것을 기본값으로 설정하는 것이 좋습니다. 앤드포인트는 DNS 레코드를 업데이트하는 데 걸리는 시간 내에(일반적으로 30초 이내) 최신 상태가 됩니다. 이 역시 애플리케이션이 사용하는 리소스 파일에 저장할 수 있습니다.

장애 조치 테스트

어떤 경우든 DB 클러스터에 ≥ 2 DB 인스턴스가 포함되어야 합니다.

서버 측에서 특정 API가 애플리케이션의 응답 방식을 테스트하는 데 사용할 수 있는 종단을 유발할 수 있습니다.

- [FailoverDBCluster](#) - DB 클러스터의 새로운 DB 인스턴스를 라이터로 승격하려고 합니다.

```
public void causeFailover() {
/*
 * See http://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/basics.html for
more details on setting up an RDS client
*/
final AmazonRDS rdsClient = AmazonRDSClientBuilder.defaultClient();

FailoverDBClusterRequest request = new FailoverDBClusterRequest();
request.setDBClusterIdentifier("cluster-identifier");

rdsClient.failoverDBCluster(request);
}
```

- [RebootDBInstance](#) - 이 API에서는 장애 조치가 보장되지 않습니다. 하지만 이 경우 라이터에서 데이터베이스가 종료되고 연결 종단에 대한 애플리케이션의 대응 방식을 테스트하는 데 사용할 수 있습니다 (ForceFailover 파라미터는 Aurora 엔진에 적용되지 않으며, 대신 FailoverDBCluster API를 사용해야 함).

- [ModifyDBCluster](#) - 포트를 변경하면 클러스터의 노드가 새 포트에서 수신을 시작할 경우 종단을 유발합니다. 일반적으로 애플리케이션은 사용자의 애플리케이션만 포트 변경을 제어하고 해당 앤드포인트를 적절히 업데이트할 수 있는지 확인하고, 누군가 API 수준에서 변경할 때 포트를 수동으로 업데이트하도록 하거나, 애플리케이션에서 RDS API를 쿼리하여 포트가 변경되었는지 판단하는 방식으로 이와 같은 장애에 대응할 수 있습니다.
- [ModifyDBInstance](#) - DBInstanceClass 수정 시 종단이 발생합니다
- [DeleteDBInstance](#) - 마스터/라이터를 삭제하면 새 DB 인스턴스가 DB 클러스터의 라이터로 승격됩니다

Linux 사용 시 애플리케이션/클라이언트 측에서 포트 및 호스트 기반 패킷의 갑작스러운 폐기 또는 tcp keepalive 패킷이 전송되지 않거나 iptables를 이용해 수신되는 경우 애플리케이션이 어떻게 대응하는지 테스트할 수 있습니다.

빠른 장애 조치 예제

다음 코드 샘플은 애플리케이션이 Aurora PostgreSQL 드라이버 관리자를 설정하는 방식을 보여줍니다. 애플리케이션이 연결해야 할 때 `getConnection(...)`을 호출할 것입니다. 라이터를 찾을 수 없으나 `targetServerType`이 "마스터"로 설정된 경우 등에는 이 함수에 대한 호출이 유효한 호스트를 찾는데 실패할 수 있으며, 애플리케이션이 호출을 다시 시도해야 합니다. 이 경우 애플리케이션에 재시도 동작을 푸시하지 않도록 연결 풀러로 쉽게 래핑될 수 있습니다. 대다수 연결 풀러에서 JDBC 연결 문자열을 지정할 수 있으므로 애플리케이션이 `getJdbcConnectionString(...)`을 호출하고 연결 풀러로 이를 전달하여 Aurora PostgreSQL에서 빠른 장애 조치를 활용할 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 kill 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://%" +
    + "/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-c9bfei4hj1rd.us-east-1-
beta.rds.amazonaws.com:5432");
    newEndpointList.add("myauroracluster.cluster-ro-c9bfei4hj1rd.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(","));
}

```

}

스토리지 문제 해결

정렬이나 인덱스 생성 작업에 필요한 메모리 사용 가능한 메모리 양을 초과하면, Aurora PostgreSQL은 초과하는 데이터를 스토리지에 기록합니다. 데이터를 기록할 때 오류와 메시지 로그 저장에 사용하는 동일한 스토리지 공간을 사용합니다. 정렬이나 인덱스 생성 기능이 사용 가능한 메모리를 초과할 경우, 로컬 스토리지가 부족할 수 있습니다. Aurora PostgreSQL에서 스토리지 공간 부족 문제가 발생하면 데이터 정렬에 더 많은 메모리를 사용하도록 다시 구성하거나, PostgreSQL 로그 파일의 데이터 보존 기간을 줄일 수 있습니다. 로그 보존 기간 변경에 대한 자세한 내용은 [PostgreSQL 데이터베이스 로그 파일 \(p. 456\)](#)을 참조하십시오.

Aurora PostgreSQL과 함께 Kerberos 인증 사용

사용자가 PostgreSQL이 실행되는 DB 클러스터에 연결할 때 Kerberos 인증을 사용하여 사용자를 인증할 수 있습니다. 이 경우 DB 인스턴스는 AWS Managed Microsoft AD라고도 하는 Microsoft Active Directory용 AWS 디렉터리 서비스(Enterprise Edition)과 함께 작동하여 Kerberos 인증을 활성화합니다. 사용자가 신뢰하는 도메인에 가입된 PostgreSQL DB 클러스터를 사용하여 인증하면 AWS Directory Service를 사용하여 생성한 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.

Kerberos는 AWS Identity and Access Management (IAM)과 다른 인증 방법을 제공합니다. 데이터베이스는 Kerberos 또는 IAM 인증을 모두 사용할 수 있지만 둘 다 사용할 수는 없습니다. IAM 인증에 관한 자세한 내용은 [을 위한 IAM 데이터베이스 인증 \(p. 976\)](#) 단원을 참조하십시오.

Amazon Aurora는 다음 AWS 리전에서 PostgreSQL DB 클러스터에 대해 Kerberos 인증을 지원합니다.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(캘리포니아 북부 지역)
- 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(스톡홀름)
- 남아메리카(상파울루)

주제

- [PostgreSQL DB 클러스터에 대한 Kerberos 인증 개요 \(p. 886\)](#)
- [PostgreSQL DB 클러스터에 대해 Kerberos 인증 설정 \(p. 886\)](#)
- [도메인에서 DB 클러스터 관리 \(p. 894\)](#)

- Kerberos 인증을 사용하여 PostgreSQL 연결 (p. 894)

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 딕렉터리를 생성합니다. AWS Management 콘솔, AWS CLI 또는 AWS Directory Service API를 사용하여 딕렉터리를 생성할 수 있습니다.
2. AWS Managed Microsoft AD 딕렉터리를 호출할 수 있는 Amazon Aurora 권한을 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 리전에서 AWS 계정의 AWS Security Token Service(AWS STS) 엔드포인트를 활성화해야 합니다. AWS STS 엔드포인트는 기본적으로 모든 AWS 리전에서 활성화되어 있으므로 추가 작업 없이 사용할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 리전에서 AWS STS 활성화 및 비활성화](#)를 참조하십시오.

3. Microsoft Active Directory 도구를 사용하여 AWS Managed Microsoft AD 딕렉터리에서 사용자를 만들고 구성합니다. Active Directory에서 사용자를 생성하는 방법에 대한 자세한 내용은 AWS Directory Service 관리 안내서의 [AWS Managed Microsoft AD에서 사용자 및 그룹 관리](#)를 참조하십시오.
4. 딕렉터리와 DB 인스턴스를 다른 Virtual Private Cloud(VPC)에 배치하려면 VPC 피어링을 구성하십시오. 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.
 - [DB 클러스터 생성 및 Aurora PostgreSQL DB 클러스터의 데이터베이스에 연결](#) (p. 83)
 - [Amazon Aurora DB 클러스터 수정](#) (p. 193)
 - [DB 클러스터 스냅샷에서 복원](#) (p. 273)
 - [DB 클러스터를 지정된 시간으로 복원](#) (p. 303)

DB 클러스터를 생성하거나 수정한 경우 딕렉터리를 만들 때 생성된 도메인 식별자(d-* 식별자)를 제공합니다. 또한 생성한 IAM 역할의 이름도 제공하십시오. 딕렉터리와 동일한 VPC 또는 다른 VPC에 DB 클러스터를 배치할 수 있습니다.

6. RDS 마스터 사용자 자격 증명을 사용하여 PostgreSQL DB 클러스터에 연결합니다. 외부에서 식별 할 사용자를 PostgreSQL에서 생성합니다. 외부에서 식별되는 사용자는 Kerberos 인증을 사용하여 PostgreSQL DB 클러스터에 로그인할 수 있습니다.

PostgreSQL DB 클러스터에 대해 Kerberos 인증 설정

Microsoft Active Directory용 AWS 딕렉터리 서비스(Enterprise Edition)(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 딕렉터리 생성](#) (p. 887)
- [2단계: Amazon Aurora가 AWS Directory Service에 액세스할 수 있는 IAM 역할 생성](#) (p. 890)
- [3단계: 사용자 생성 및 구성](#) (p. 891)
- [4단계: VPC 피어링 구성](#) (p. 891)
- [5단계: PostgreSQL DB 클러스터 생성 또는 수정](#) (p. 892)
- [6단계: Kerberos 인증 PostgreSQL 로그인 생성](#) (p. 893)
- [7단계: PostgreSQL 클라이언트 구성](#) (p. 893)

1단계: AWS Managed Microsoft AD를 사용하여 딕터리 생성

AWS Directory Service는 AWS 클라우드에서 완전 관리형 Microsoft Active Directory를 생성합니다. AWS Managed Microsoft AD 딕터리를 생성할 때 AWS Directory Service에서 두 개의 도메인 컨트롤러와 DNS 서버가 자동으로 생성됩니다. 딕터리 서비스는 VPC 내 다른 서브넷에서 생성됩니다. 이러한 중복으로 인해 장애가 발생해도 딕터리에 액세스할 수 있습니다.

AWS Managed Microsoft AD 딕터리를 생성할 때 AWS Directory Service에서 다음 작업을 자동으로 수행합니다.

- VPC 내에서 Active Directory를 설정합니다.
- 사용자 이름 Admin과 지정된 암호를 사용하여 딕터리 관리자 계정을 생성합니다. 이 계정을 사용하여 딕터리를 관리할 수 있습니다.

Important

반드시 이 암호를 저장해야 합니다. AWS Directory Service에서는 이 암호를 저장하지 않으므로 암호를 검색하거나 다시 설정할 수 없습니다.

- 딕터리 컨트롤러에 대한 보안 그룹을 만듭니다.

Microsoft Active Directory용 AWS 딕터리 서비스(Enterprise Edition)를 시작하면 AWS에서 모든 딕터리의 객체를 포함하는 OU(조직 단위)를 생성합니다. 딕터리를 만들 때 입력한 NetBIOS 이름이 있는 이 OU는 도메인 루트에 있습니다. 도메인 루트는 AWS에서 소유하고 관리합니다.

AWS Managed Microsoft AD 딕터리를 사용하여 생성한 Admin 계정은 OU의 가장 일반적인 관리 활동에 대한 권한이 있습니다.

- 사용자 생성, 업데이트 또는 삭제
- 도메인(예: 파일 또는 인쇄 서비스)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 권리가 있습니다.

- DNS 구성 관리(레코드, 영역 및 전달자 추가, 제거 또는 업데이트)
- DNS 이벤트 로그 보기
- 보안 이벤트 로그 보기

AWS Managed Microsoft AD으로 딕터리를 생성하려면

1. [AWS Directory Service 콘솔 탐색 창에서 딕터리를 선택한 후 딕터리 설정을 선택합니다.](#)
2. AWS Managed Microsoft AD를 선택합니다. AWS Managed Microsoft AD는 현재 Amazon Aurora에서 사용하도록 지원되는 유일한 옵션입니다.
3. [Next]를 선택합니다.
4. 딕터리 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 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 암호를 다시 입력합니다.

Important

반드시 이 암호를 저장해야 합니다. AWS Directory Service에서는 이 암호를 저장하지 않으며 암호를 검색하거나 재설정할 수 없습니다.

5. [Next]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.

Subnets

디렉터리 서버에 대한 서브넷을 선택합니다. 두 서브넷이 서로 다른 가용 영역에 있어야 합니다.

7. [Next]를 선택합니다.
8. 디렉터리 정보를 검토합니다. 변경이 필요하면 이전을 선택하여 변경합니다. 정보가 올바르면 디렉터리 생성을 선택합니다.

Review & create

Review

Directory type	VPC
Microsoft AD	vpc-8b6b78e9 ([REDACTED])
Directory DNS name	Subnets
corp.example.com	subnet-75128d10 ([REDACTED], us-east-1a) subnet-f51665dd ([REDACTED], us-east-1b)
Directory NetBIOS name	
CORP	
Directory description	
My directory	

Pricing

Edition	Free trial eligible Learn more 30-day limited trial
Standard	
~USD [REDACTED] *	
* Includes two domain controllers, USD [REDACTED] /mo for each additional domain controller.	

[Cancel](#) [Previous](#) [Create directory](#)

디렉터리를 생성하는 데 몇 분 정도 걸립니다. 디렉터리가 생성된 경우 [Status] 값이 [Active]로 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.

The screenshot shows the 'Directories' section of the AWS Directory Service console. A Microsoft AD directory named 'd-90670a8d36' is selected. The 'Directory details' table includes the following information:

Directory type	VPC	Status
Microsoft AD	vpc-6594f31c	Active
Edition	Subnets	Last updated
Standard	subnet-7d36a227 subnet-a2ab49c6	Tuesday, January 7, 2020
Directory ID	Availability zones	Launch time
d-90670a8d36	us-east-1c, us-east-1d	Tuesday, January 7, 2020
Directory DNS name	DNS address	
corp.example.com	[REDACTED]	
Directory NetBIOS name		
CORP		
Description - Edit		
My directory		

Below the table, there are tabs for 'Application management' (which is active), 'Scale & share', 'Networking & security', and 'Maintenance'.

2단계: Amazon Aurora가 AWS Directory Service에 액세스할 수 있는 IAM 역할 생성

AWS Directory Service를 호출하는 Amazon Aurora의 경우 관리형 IAM 정책 `AmazonRDSDirectoryServiceAccess`를 사용하는 IAM 역할이 필요합니다. 이 역할을 사용하여 Amazon Aurora에서 AWS Directory Service를 호출할 수 있습니다.

AWS Management 콘솔을 사용하여 DB 인스턴스를 생성할 때 콘솔 사용자에게 `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 참조하십시오.

Note

RDS for Microsoft SQL Server에 대한 Windows 인증에 사용되는 IAM 역할은 Amazon Aurora에 사용할 수 없습니다.

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.

```
{  
    "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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Action": [  
                "ds:DescribeDirectories",  
                "ds:AuthorizeApplication",  
                "ds:UnauthorizeApplication",  
                "ds:GetAuthorizedApplicationDetails"  
            ],  
            "Effect": "Allow",  
            "Resource": "*"  
        }  
    ]  
}
```

3단계: 사용자 생성 및 구성

Active Directory 도메인 서비스 및 Active Directory Lightweight Directory Services 도구의 일부인 Active Directory 사용자 및 컴퓨터 도구를 사용하여 사용자를 생성할 수 있습니다. 이 경우 사용자는 디렉터리에 액세스할 수 있는 개별 사용자 또는 개체입니다.

AWS Directory Service 디렉터리에서 사용자를 생성하려면 Windows 기반 Amazon EC2 인스턴스에 연결되어 있어야 합니다. 또한 이 EC2 인스턴스는 AWS Directory Service 디렉터리의 멤버여야 합니다. 이와 동시에 사용자를 생성할 권한이 있는 사용자로 로그인한 상태이어야 합니다. 자세한 내용은 AWS Directory Service 관리 안내서의 [사용자 생성](#)을 참조하십시오.

4단계: VPC 피어링 구성

디렉터리와 DB 클러스터를 동일한 VPC에 배치하려면 이 단계를 건너뛰고 [5단계: PostgreSQL DB 클러스터 생성 또는 수정 \(p. 892\)](#) 단원으로 이동하십시오. 디렉터리와 DB 클러스터를 다른 VPC에 배치하려면 이 단계의 다음 지침에 따라 VPC 피어링을 구성하십시오.

동일한 AWS 계정에서 두 가지 VPC를 모두 소유한 경우에는 Amazon VPC Peering Guide의 [VPC 피어링이란?](#)에 있는 지침을 따르십시오. 구체적으로 다음 단계를 완료하십시오.

1. 네트워크 트래픽이 양방향으로 흐를 수 있도록 적절한 VPC 라우팅 규칙을 설정합니다.
2. DB 인스턴스의 보안 그룹이 이 보안 그룹에서 오는 수신 트래픽을 수신할 수 있게 하십시오.
3. 트래픽을 차단하는 네트워크 ACL(액세스 제어 목록) 규칙이 없어야 합니다.

다른 AWS 계정에서 VPC를 소유하는 경우 다음 단계를 완료하십시오.

1. [VPC 피어링이란?](#)의 지침에 따라 VPC 피어링을 구성합니다. 구체적으로 다음 단계를 완료하십시오.
 - a. 네트워크 트래픽이 양방향으로 흐를 수 있도록 적절한 VPC 라우팅 규칙을 설정합니다.
 - b. 트래픽을 차단하는 ACL 규칙이 없어야 합니다.
2. DB 클러스터를 생성하려는 AWS 계정과 디렉터리 공유를 시작합니다. 이 작업을 수행하려면 AWS Directory Service 관리 안내서의 [자습서: 원활한 EC2 도메인 조인을 위해 AWS 관리형 Microsoft AD 디렉터리를 공유](#)에 있는 지침을 따르십시오.
3. DB 클러스터용 계정을 사용하여 AWS Directory Service 콘솔에 로그인하고 계속하기 전에 도메인에 SHARED 상태가 있는지 확인합니다.
4. DB 클러스터의 계정을 사용하여 AWS Directory Service 콘솔에 로그인한 상태에서 디렉터리의 디렉터리 ID 값을 기록해둡니다.

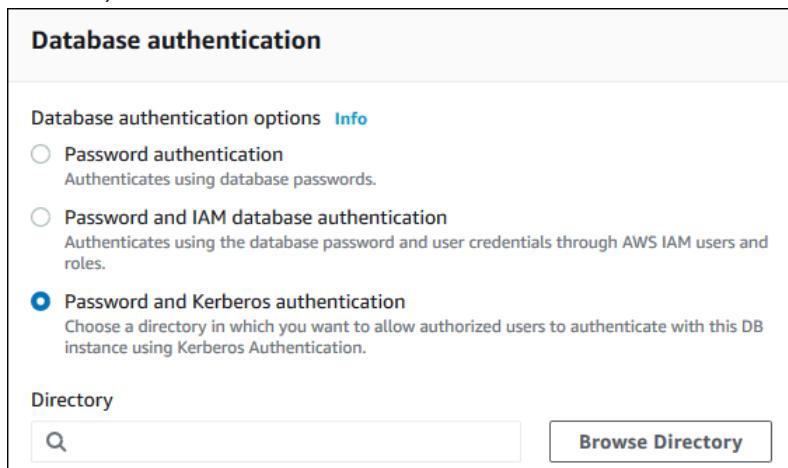
5단계: PostgreSQL DB 클러스터 생성 또는 수정

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 [DB 클러스터 생성 및 Aurora PostgreSQL DB 클러스터의 데이터베이스에 연결](#) (p. 83)을 참조하십시오.
- 콘솔, [modify-db-cluster](#) CLI 명령 또는 [ModifyDBCluster](#) RDS API 작업을 사용하여 기존 PostgreSQL DB 클러스터를 수정합니다. 지침은 [Amazon Aurora DB 클러스터 수정](#) (p. 193)을 참조하십시오.
- 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](#) (p. 273)을 참조하십시오.
- 콘솔, [restore-db-instance-to-point-in-time](#) CLI 명령 또는 [RestoreDBClusterToPointInTime](#) RDS API 작업을 사용하여 PostgreSQL DB 클러스터를 특정 시점으로 복원합니다. 지침은 [DB 클러스터를 지정된 시간으로 복원](#) (p. 303)을 참조하십시오.

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 송신을 허용하는 보안 그룹을 사용해야 합니다.

콘솔을 사용하여 DB 클러스터를 생성하는 경우 Database authentication(데이터베이스 인증) 섹션에서 Password and Kerberos authentication(암호 및 Kerberos 인증)을 선택합니다. Browse Directory(디렉터리 찾아보기)를 선택한 다음 디렉터리를 선택하거나 새 디렉터리 생성을 선택합니다.



콘솔을 사용하여 DB 클러스터를 수정하거나 복원할 때는 Kerberos authentication(Kerberos 인증) 섹션에서 디렉터리를 선택하거나 새 디렉터리 생성을 선택합니다.

Kerberos authentication

Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos authentication.

Directory

None

Create a new directory

By choosing a directory and continuing with database instance creation you authorize Amazon RDS to create the IAM role necessary for using Kerberos authentication

Refresh

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 구하가 생성한, 관리형 IAM 정책 AmazonRDS Directory Service Access를 사용하는 역할을 사용하십시오.

예를 들어, 다음 CLI 명령은 디렉터리를 사용하도록 DB 클러스터를 수정합니다.

```
aws rds modify-db-cluster --db-instance-identifier mydbinstance --domain d-Directory-ID --domain-iam-role-name role-name
```

Important

DB 클러스터를 수정하여 Kerberos 인증을 활성화하는 경우에는 변경 후 DB 클러스터를 재부팅하십시오.

6단계: Kerberos 인증 PostgreSQL 로그인 생성

다음은 다른 DB 클러스터의 경우와 같은 방법으로 RDS 마스터 사용자 자격 증명을 사용하여 PostgreSQL DB 클러스터에 연결합니다. DB 인스턴스는 AWS Managed Microsoft AD 도메인에 조인됩니다. 따라서 도메인 내 Microsoft Active Directory 사용자 및 그룹에서 PostgreSQL 로그인 및 사용자를 프로비저닝할 수 있습니다. 데이터베이스 권한을 관리하려면 이 로그인에 대해 표준 PostgreSQL 권한을 부여하고 취소하십시오.

Active Directory 사용자가 PostgreSQL로 인증할 수 있게 허용하려면 RDS 마스터 사용자 자격 증명을 사용하십시오. 다른 DB 클러스터의 경우와 같은 방법으로 이러한 자격 증명을 사용하여 PostgreSQL DB 클러스터에 연결합니다. 로그인한 후에는 PostgreSQL에서 외부에서 인증된 사용자를 생성하고 이 사용자에게 rds_ad 역할을 부여합니다.

```
CREATE USER "username@CORP.EXAMPLE.COM" WITH LOGIN;  
GRANT rds_ad TO "username@CORP.EXAMPLE.COM";
```

username 을 사용자 이름으로 대체하고 도메인 이름을 대문자로 포함합니다. 이제 도메인의 사용자(사람 및 애플리케이션)는 Kerberos 인증을 사용하여 도메인이 조인된 클라이언트 컴퓨터의 RDS PostgreSQL 클러스터에 연결할 수 있습니다.

7단계: PostgreSQL 클라이언트 구성

PostgreSQL 클라이언트를 구성하려면 다음 요구 사항을 충족하십시오.

- 도메인을 가리키도록 krb5.conf 파일(또는 동등한 파일)을 생성합니다.

- 클라이언트 호스트와 AWS Directory Service 간에 트래픽이 흐를 수 있는지 확인합니다. Netcat과 같은 네트워크 유ти리티를 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.

도메인에서 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. `--domain` 옵션의 경우 `none`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 해당 DB 인스턴스는 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` - AWS가 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` - AWS는 예약된 다음 유지 관리 기간 동안 DB 인스턴스에 대한 Kerberos 인증을 활성화하려 합니다.
- `pending-disable-kerberos` - 이 DB 인스턴스에 대한 Kerberos 인증 비활성화가 보류 중입니다.
- `pending-maintenance-disable-kerberos` - AWS는 예약된 다음 유지 관리 기간 동안 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` - AWS가 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 클러스터 연결 \(p. 166\)](#) 단원을 참조하십시오.

pgAdmin

pgAdmin을 사용하여 Kerberos 인증으로 PostgreSQL에 연결하려면 다음 단계를 수행하십시오.

- 클라이언트 컴퓨터에서 pgAdmin 애플리케이션을 실행합니다.
- [Dashboard] 탭에서 [Add New Server]를 선택합니다.
- [Create - Server] 대화 상자에서 pgAdmin의 서버를 식별하기 위해 [General] 탭에 이름을 입력합니다.
- 연결 탭에서 Aurora PostgreSQL 데이터베이스에 있는 다음 정보를 입력합니다.
 - 호스트에 엔드포인트를 입력합니다. 예를 들어, *PostgreSQL-endpoint.AWS-Region.rds.amazonaws.com* 형식을 사용합니다.
 - [Port]에 할당된 포트를 입력합니다.
 - Maintenance database(유지 관리 데이터베이스)에 클라이언트가 연결될 초기 데이터베이스의 이름을 입력합니다.
 - 사용자 이름에는 DB 클러스터를 생성할 때 입력한 사용자 이름을 입력합니다.
 - 암호에는 DB 클러스터를 생성할 때 입력했던 암호를 입력합니다.
- 저장을 선택합니다.

psql

psql을 사용하여 Kerberos 인증으로 PostgreSQL에 연결하려면 다음 단계를 수행하십시오.

- 명령 프롬프트에서 다음 명령을 실행합니다.

```
kinit username
```

*username*을 사용자 이름으로 대체합니다. 프롬프트에서 Microsoft Active Directory에 저장된 사용자 암호를 입력합니다.

- PostgreSQL DB 클러스터가 공개적으로 액세스 가능한 VPC를 사용하는 경우 DB 클러스터 엔드포인트의 프라이빗 IP 주소를 EC2 클라이언트의 /etc/hosts 파일에 넣습니다. 예를 들어 다음 명령은 프라이빗 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
```

- 다음 psql 명령을 사용하여 Microsoft Active Directory와 통합된 PostgreSQL DB 클러스터에 로그인합니다.

```
psql -U username@CORP.EXAMPLE.COM -p 5432 -h PostgreSQL-instance-endpoint.AWS-Region.rds.amazonaws.com postgres
```

Amazon Aurora PostgreSQL 참조

목차

- Amazon Aurora PostgreSQL 파라미터 (p. 896)
 - Aurora PostgreSQL 클러스터 수준 파라미터 (p. 896)
 - Aurora PostgreSQL 인스턴스 수준 파라미터 (p. 898)
- Amazon Aurora PostgreSQL 이벤트 (p. 904)

Amazon Aurora PostgreSQL 파라미터

Amazon Aurora 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 달릅니다. 따라서 Amazon Aurora DB 클러스터를 관리하는 데 사용하는 일부 파라미터는 엔진 클러스터에 적용되는 반면, 또 다른 일부 파라미터는 DB 클러스터의 특정 DB 인스턴스에만 적용됩니다.

클러스터 수준의 파라미터는 DB 클러스터 파라미터 그룹에서 관리됩니다. 인스턴스 수준의 파라미터는 DB 파라미터 그룹에서 관리됩니다. Aurora PostgreSQL DB 클러스터의 각 DB 인스턴스는 PostgreSQL 데이터베이스 엔진과 호환되기는 하지만, PostgreSQL 데이터베이스 엔진 파라미터의 일부는 클러스터 수준에서만 적용해야 하며 DB 클러스터 파라미터 그룹을 사용하여 관리됩니다. 클러스터 수준 파라미터는 DB 클러스터의 DB 인스턴스에 대한 DB 파라미터 그룹에 없으며 이 항목의 뒷부분에 나와 있습니다.

클러스터 수준 파라미터와 인스턴스 수준 파라미터는 Amazon RDS 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 확인할 수 있습니다. 예를 들어 DB 클러스터 파라미터 그룹에서 클러스터 수준 파라미터를 보려면 [describe-db-cluster-parameters](#) AWS CLI 명령을 사용합니다. DB 클러스터의 DB 인스턴스에 대한 DB 파라미터 그룹에서 인스턴스 수준 파라미터를 보려면 [describe-db-parameters](#) AWS CLI 명령을 사용합니다.

클러스터 수준 파라미터와 인스턴스 수준 파라미터는 모두 Amazon RDS 콘솔, AWS CLI 또는 Amazon RDS API를 사용하여 관리할 수 있습니다. 클러스터 수준 파라미터와 인스턴스 수준 파라미터를 관리하기 위한 명령은 서로 다릅니다. 다음 예를 참조하십시오.

- 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.

파라미터 그룹에 대한 자세한 정보는 [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 168\)](#) 단원을 참조하십시오.

Aurora PostgreSQL 클러스터 수준 파라미터

다음 표에는 전체 Aurora PostgreSQL DB 클러스터에 적용되는 모든 파라미터가 나와 있습니다.

파라미터 이름	수정 가능
<code>ansi_constraint_trigger_ordering</code>	예
<code>ansi_force_foreign_key_checks</code>	예
<code>ansi_qualified_update_set_target</code>	예
<code>apg_ccm_enabled</code>	예
<code>archive_command</code>	아니요
<code>archive_timeout</code>	아니요
<code>array_nulls</code>	예
<code>autovacuum</code>	예
<code>autovacuum_analyze_scale_factor</code>	예
<code>autovacuum_analyze_threshold</code>	예
<code>autovacuum_freeze_max_age</code>	예

파라미터 이름	수정 가능
<code>autovacuum_max_workers</code>	예
<code>autovacuum_multixact_freeze_max_age</code>	예
<code>autovacuum_naptime</code>	예
<code>autovacuum_vacuum_cost_delay</code>	예
<code>autovacuum_vacuum_cost_limit</code>	예
<code>autovacuum_vacuum_scale_factor</code>	예
<code>autovacuum_vacuum_threshold</code>	예
<code>autovacuum_work_mem</code>	예
<code>backslash_quote</code>	예
<code>client_encoding</code>	예
<code>data_directory</code>	아니요
<code>datestyle</code>	예
<code>default_tablespace</code>	예
<code>default_with_oids</code>	예
<code>extra_float_digits</code>	예
<code>huge_pages</code>	아니요
<code>intervalstyle</code>	예
<code>lc_monetary</code>	예
<code>lc_numeric</code>	예
<code>lc_time</code>	예
<code>log_autovacuum_min_duration</code>	예
<code>max_prepared_transactions</code>	예
<code>password_encryption</code>	아니요
<code>port</code>	아니요
<code>rds.enable_plan_management</code>	예
<code>rds.extensions</code>	아니요
<code>rds.force_autovacuum_logging_level</code>	예
<code>rds.force_ssl</code>	예
<code>rds.logical_replication</code>	예
<code>rds.restrict_password_commands</code>	예
<code>server_encoding</code>	아니요

파라미터 이름	수정 가능
ssl	예
synchronous_commit	예
timezone	예
track_commit_timestamp	예
vacuum_cost_delay	예
vacuum_cost_limit	예
vacuum_cost_page_hit	예
vacuum_cost_page_miss	예
vacuum_defer_cleanup_age	예
vacuum_freeze_min_age	예
vacuum_freeze_table_age	예
vacuum_multixact_freeze_min_age	예
vacuum_multixact_freeze_table_age	예
wal_buffers	예

Aurora PostgreSQL 인스턴스 수준 파라미터

다음 표에는 Aurora PostgreSQL DB 클러스터의 특정 DB 클러스터에 적용되는 모든 파라미터가 나와 있습니다.

파라미터 이름	수정 가능
apg_enable_not_in_transform	예
apg_enable_remove_redundant_inner_joins	예
apg_enable_semijoin_push_down	예
apg_plan_mgmt.capture_plan_baselines	예
apg_plan_mgmt.max_databases	예
apg_plan_mgmt.max_plans	예
apg_plan_mgmt.pgss_min_calls(사용되지 않음)	예
apg_plan_mgmt.pgss_min_mean_time_ms(사용되지 않음)	예
apg_plan_mgmt.pgss_min_stddev_time_ms(사용되지 않음)	예
apg_plan_mgmt.pgss_min_total_time_ms(사용되지 않음)	예
apg_plan_mgmt.plan_retention_period	예
apg_plan_mgmt.unapproved_plan_execution_threshold	예

파라미터 이름	수정 가능
apg_plan_mgmt.use_plan_baselines	예
application_name	예
authentication_timeout	예
auto_explain.log_analyze	예
auto_explain.log_buffers	예
auto_explain.log_format	예
auto_explain.log_min_duration	예
auto_explain.log_nested_statements	예
auto_explain.log_timing	예
auto_explain.log_triggers	예
auto_explain.log_verbose	예
auto_explain.sample_rate	예
backend_flush_after	예
bgwriter_flush_after	예
bytea_output	예
check_function_bodies	예
checkpoint_flush_after	예
checkpoint_timeout	아니요
client_min_messages	예
config_file	아니요
constraint_exclusion	예
cpu_index_tuple_cost	예
cpu_operator_cost	예
cpu_tuple_cost	예
cursor_tuple_fraction	예
db_user_namespace	아니요
deadlock_timeout	예
debug_pretty_print	예
debug_print_parse	예
debug_print_plan	예
debug_print_rewritten	예

파라미터 이름	수정 가능
default_statistics_target	예
default_transaction_deferrable	예
default_transaction_isolation	예
default_transaction_read_only	예
effective_cache_size	예
effective_io_concurrency	예
enable_bitmapscan	예
enable_hashagg	예
enable_hashjoin	예
enable_indexonlyscan	예
enable_indexscan	예
enable_material	예
enable_mergejoin	예
enable_nestloop	예
enable_seqscan	예
enable_sort	예
enable_tidscan	예
escape_string_warning	예
exit_on_error	아니요
force_parallel_mode	예
fromCollapse_limit	예
geqo	예
geqo_effort	예
geqo_generations	예
geqo_pool_size	예
geqo_seed	예
geqo_selection_bias	예
geqo_threshold	예
gin_fuzzy_search_limit	예
gin_pending_list_limit	예
hba_file	아니요

파라미터 이름	수정 가능
hot_standby_feedback	아니요
ident_file	아니요
idle_in_transaction_session_timeout	예
joinCollapse_limit	예
lc_messages	예
listen_addresses	아니요
lo_compat_privileges	아니요
log_connections	예
log_destination	예
log_directory	아니요
log_disconnections	예
log_duration	예
log_error_verbosity	예
log_executor_stats	예
log_file_mode	아니요
log_filename	예
log_hostname	예
log_line_prefix	아니요
log_lock_waits	예
log_min_duration_statement	예
log_min_error_statement	예
log_min_messages	예
log_parser_stats	예
log_planner_stats	예
log_replication_commands	예
log_rotation_age	예
log_rotation_size	예
log_statement	예
log_statement_stats	예
log_temp_files	예
log_timezone	아니요

파라미터 이름	수정 가능
<code>log_truncate_on_rotation</code>	아니요
<code>logging_collector</code>	아니요
<code>maintenance_work_mem</code>	예
<code>max_connections</code>	예
<code>max_files_per_process</code>	예
<code>max_locks_per_transaction</code>	예
<code>max_replication_slots</code>	예
<code>max_stack_depth</code>	예
<code>max_standby_archive_delay</code>	아니요
<code>max_standby_streaming_delay</code>	아니요
<code>max_wal_senders</code>	예
<code>max_worker_processes</code>	예
<code>min_parallel_relation_size</code>	예
<code>old_snapshot_threshold</code>	예
<code>operator_precedence_warning</code>	예
<code>parallel_setup_cost</code>	예
<code>parallel_tuple_cost</code>	예
<code>pg_hint_plan.debug_print</code>	예
<code>pg_hint_plan.enable_hint</code>	예
<code>pg_hint_plan.enable_hint_table</code>	예
<code>pg_hint_plan.message_level</code>	예
<code>pg_hint_plan.parse_messages</code>	예
<code>pg_stat_statements.max</code>	예
<code>pg_stat_statements.save</code>	예
<code>pg_stat_statements.track</code>	예
<code>pg_stat_statements.track_utility</code>	예
<code>pgaudit.log</code>	예
<code>pgaudit.log_catalog</code>	예
<code>pgaudit.log_level</code>	예
<code>pgaudit.log_parameter</code>	예
<code>pgaudit.log_relation</code>	예

파라미터 이름	수정 가능
<code>pgaudit.log_statement_once</code>	예
<code>pgaudit.role</code>	예
<code>postgis.gdal_enabled_drivers</code>	예
<code>quote_all_identifiers</code>	예
<code>random_page_cost</code>	예
<code>rds.force_admin_logging_level</code>	예
<code>rds.log_retention_period</code>	예
<code>rds.rds_superuser_reserved_connections</code>	예
<code>rds.superuser_variables</code>	아니요
<code>replacement_sort_tuples</code>	예
<code>restart_after_crash</code>	아니요
<code>row_security</code>	예
<code>search_path</code>	예
<code>seq_page_cost</code>	예
<code>session_replication_role</code>	예
<code>shared_buffers</code>	예
<code>shared_preload_libraries</code>	예
<code>sql_inheritance</code>	예
<code>ssl_ca_file</code>	아니요
<code>ssl_cert_file</code>	아니요
<code>ssl_ciphers</code>	아니요
<code>ssl_key_file</code>	아니요
<code>standard_conforming_strings</code>	예
<code>statement_timeout</code>	예
<code>stats_temp_directory</code>	아니요
<code>superuser_reserved_connections</code>	아니요
<code>synchronize_seqscans</code>	예
<code>syslog_facility</code>	아니요
<code>tcp_keepalives_count</code>	예
<code>tcp_keepalives_idle</code>	예
<code>tcp_keepalives_interval</code>	예

파라미터 이름	수정 가능
<code>temp_buffers</code>	예
<code>temp tablespaces</code>	예
<code>track_activities</code>	예
<code>track_activity_query_size</code>	예
<code>track_counts</code>	예
<code>track_functions</code>	예
<code>track_io_timing</code>	예
<code>transaction_deferrable</code>	예
<code>transaction_read_only</code>	예
<code>transform_null_equals</code>	예
<code>unix_socket_directories</code>	아니요
<code>unix_socket_group</code>	아니요
<code>unix_socket_permissions</code>	아니요
<code>update_process_title</code>	예
<code>wal_receiver_status_interval</code>	예
<code>wal_receiver_timeout</code>	예
<code>wal_sender_timeout</code>	예
<code>work_mem</code>	예
<code>xmlbinary</code>	예
<code>xmloption</code>	예

Amazon Aurora PostgreSQL 이벤트

다음은 Aurora PostgreSQL의 일반 대기 이벤트입니다.

BufferPin:BufferPin

이 대기 이벤트에서는 세션이 데이터 버퍼에 액세스하려고 대기 중이며 이 기간 동안 다른 세션은 해당 버퍼를 검사할 수 없습니다. 다른 프로세스에서 해당 버퍼로부터 마지막으로 읽은 데이터에 열린 커서를 유지하는 경우 버퍼 핀 대기가 연장될 수 있습니다.

Client:ClientRead

이 대기 이벤트에서는 세션이 애플리케이션 클라이언트에서 데이터를 받고 있습니다. 이 대기는 COPY 문을 사용한 대량 데이터 로드 중에, 또는 클라이언트와 데이터베이스 간의 많은 왕복 횟수를 사용하여 Aurora에 데이터를 전달하는 애플리케이션에서 일반적으로 발생할 수 있습니다. 트랜잭션당 높은 클라이언트 읽기 대기 횟수는 파라미터 전달 같은 과다한 왕복 시간을 나타낼 수도 있습니다. 이 값을 트랜잭션당 필요 문 수와 비교해야 합니다.

IO:DataFilePrefetch

이 대기 이벤트에서는 세션이 Aurora Storage의 비동기 미리 가져오기를 대기 중입니다.

IO:DataFileRead

이 대기 이벤트에서는 세션이 Aurora Storage에서 데이터를 읽고 있습니다. 이는 I/O 집약적인 워크로드에 대해 발생하는 일반적인 대기 이벤트일 수 있습니다. SQL 문에서 다른 SQL 문에 비해 상대적으로 높은 비율의 이 대기 이벤트를 보이는 경우 이 상황은 대량의 데이터를 읽어야 하는 비효율적인 쿼리 계획을 사용 중일 때 발생할 수 있습니다.

IO:XactSync

이 대기 이벤트에서는 세션이 COMMIT 또는 ROLLBACK을 실행 중이며, 현재 트랜잭션의 변경 내용이 유지되어야 합니다. Aurora는 Aurora 스토리지가 지속성을 확인할 때까지 대기 중입니다.

이 대기는 시스템에서 커밋 활동 비율이 매우 높은 경우에 흔히 발생합니다. 이따금 트랜잭션을 일괄적으로 커밋하도록 애플리케이션을 수정하면 이 대기가 완화되기도 합니다. DB 로드가 DB 인스턴스에 대한 가상 CPU(vCPU) 수를 초과하는 경우 CPU 대기와 동시에 이 대기를 볼 수 있습니다. 이 경우 CPU 집약적인 데이터베이스 워크로드가 존재하는 CPU에 대해 스토리지 지속성 경합이 벌어질 수도 있습니다. 이 시나리오를 완화하려는 경우 이러한 워크로드를 줄이거나 더 많은 vCPU를 사용하는 DB 인스턴스로 확장해 볼 수 있습니다.

Lock:transactionid

이 대기 이벤트에서는 세션이 다른 세션에 의해 수정된 데이터를 수정하려고 하며, 다른 세션의 트랜잭션이 커밋되거나 롤백되기를 기다리고 있습니다. 차단 및 대기 세션은 pg_locks 보기에서 살펴볼 수 있습니다.

LWLock:buffer_content

이 대기 이벤트에서는 다른 세션에서 특정 데이터 페이지에 대해 쓰기 잠금을 설정한 동안 세션에서 메모리 내 해당 페이지 읽기 또는 쓰기를 대기 중입니다. 여러 세션에 의한 동일 데이터 부분의 잦은 업데이트로 인해 단일 페이지(핫 페이지)에 대한 쓰기 경합이 심화되어 이 대기 이벤트가 일반적으로 발생할 수 있습니다. 외래 키 제약 조건의 과도한 사용으로 잠금 기간이 길어져서 경합이 증가할 수 있습니다. 외래 키 제약 조건 사용과 관련하여 buffer_content 대기가 발생하는 워크로드를 검토하여 해당 제약 조건이 필요한지 여부를 확인해야 합니다. 또는 상위 테이블에서 fillfactor를 줄이면 더 많은 블록 전반에 걸쳐 키를 분산하고 페이지에서 경합을 줄일 수 있습니다.

LWLock:SubtransControlLock

이 대기 이벤트에서 세션은 트랜잭션과 하위 트랜잭션 간의 상위/하위 관계를 조회하거나 조정작합니다. 하위 트랜잭션을 사용하는 가장 일반적인 두 가지 용도는 저장점과 PL/pgSQL 예외 블록입니다. 하위 트랜잭션 내에서 동시에 업데이트되는 데이터의 동시 쿼리가 많은 경우 대기 이벤트가 발생할 수 있습니다. 저장점 및 예외 블록의 사용을 줄일 수 있는지 또는 업데이트 중인 행에 대한 동시 쿼리를 줄일 수 있는지 조사해야 합니다.

모든 PostgreSQL 대기 이벤트의 전체 목록은 [PostgreSQL wait-event table](#)을 참조하십시오.

Amazon Aurora PostgreSQL 데이터베이스 엔진 업데이트

다음 항목에서 PostgreSQL과 호환되는 Amazon Aurora에 대한 버전 및 업데이트 정보를 확인할 수 있습니다. 일반적으로 Aurora에 적용되는 업데이트에 대한 자세한 내용은 [Amazon Aurora 업데이트 \(p. 325\)](#) 단원을 참조하십시오.

주제

- [Amazon Aurora PostgreSQL의 해당 버전 식별 \(p. 906\)](#)
- [Amazon Aurora PostgreSQL 엔진 버전 업그레이드 \(p. 906\)](#)

- Amazon Aurora PostgreSQL의 엔진 버전 (p. 906)
- Amazon Aurora PostgreSQL의 확장 버전 (p. 926)

Amazon Aurora PostgreSQL의 해당 버전 식별

Amazon Aurora에는 모든 Aurora DB 클러스터에서 사용할 수 있는 Aurora 일반 기능인 특정 기능이 포함되어 있습니다. 그 밖에도 Aurora에는 Aurora가 지원하는 일부 데이터베이스 엔진에서만 사용할 수 있는 기능들도 포함되어 있습니다. 이러한 기능들은 Aurora PostgreSQL 같이 해당 데이터베이스 엔진을 사용하는 Aurora DB 클러스터에만 제공됩니다.

Aurora 데이터베이스는 Aurora 버전 번호와 데이터베이스 엔진 버전 번호 등 두 가지 버전의 번호를 가지고 있습니다.

- Aurora 버전 번호를 확인하는 방법은 [해당 Amazon Aurora 버전 식별 \(p. 325\)](#) 단원을 참조하십시오.
- Aurora PostgreSQL DB 인스턴스의 데이터베이스 엔진 번호는 SERVER_VERSION 실행 시간 파라미터에 대한 쿼리를 실행하여 확인할 수 있습니다. 데이터베이스 엔진 버전 번호를 확인하려면 다음 쿼리를 사용합니다.

```
SHOW SERVER_VERSION;
```

Amazon Aurora PostgreSQL 엔진 버전 업그레이드

Amazon Aurora PostgreSQL 엔진 버전 업그레이드에 대한 자세한 내용은 [Aurora PostgreSQL용 PostgreSQL DB 엔진 업그레이드 \(p. 872\)](#) 단원을 참조하십시오.

Amazon Aurora PostgreSQL의 엔진 버전

아래에서 PostgreSQL과 호환되는 Aurora 데이터베이스 엔진의 지원 버전에 관한 정보를 확인할 수 있습니다. Aurora 데이터베이스는 Aurora 버전 번호와 데이터베이스 엔진 버전 번호 등 두 가지 버전의 번호를 가지고 있습니다. Aurora PostgreSQL 데이터베이스의 버전 번호를 확인하는 방법은 [Amazon Aurora PostgreSQL의 해당 버전 식별 \(p. 906\)](#) 단원을 참조하십시오.

다음 표에는 각 Aurora PostgreSQL 릴리스의 버전과 이와 호환되는 PostgreSQL의 버전이 나와 있습니다.

Aurora의 PostgreSQL 호환성	호환되는 PostgreSQL 릴리스
버전 3.1 (p. 907)	11.6
버전 3.0 (p. 909)	11.4
버전 2.4 (p. 910)	10.11
버전 2.3 (p. 912)	10.7
버전 2.2 (p. 913)	10.6
버전 2.1 (p. 914)	10.5
버전 2.0 (p. 916)	10.4
버전 1.6 (p. 917)	9.6.16
버전 1.5 (p. 918)	9.6.12
버전 1.4 (p. 919)	9.6.11

Aurora의 PostgreSQL 호환성	호환되는 PostgreSQL 릴리스
버전 1.3 (p. 920)	9.6.9
버전 1.2 (p. 922)	9.6.8
버전 1.1 (p. 923)	9.6.6 사용 중단
버전 1.0 (p. 924)	9.6.3 사용 중단

다음과 같은 Aurora PostgreSQL 버전이 지원됩니다.

버전 3.1

이 버전의 Aurora PostgreSQL은 PostgreSQL 11.6과 호환됩니다. 릴리스 11.6의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 11.6](#)을 참조하십시오.

이 릴리스에는 여러 가지 중요한 안정성 기능 향상이 포함되어 있습니다. 이전 PostgreSQL 11 엔진을 사용하는 Aurora PostgreSQL 클러스터를 이 릴리스로 업그레이드하는 것이 좋습니다.

이 엔진 버전에서는 다음과 같은 새로운 기능과 개선 사항을 찾을 수 있습니다.

새로운 기능

1. Amazon S3으로의 데이터 내보내기에 대한 지원. 자세한 내용은 [Aurora PostgreSQL DB 클러스터의 데이터를 Amazon S3으로 내보내기 \(p. 812\)](#) 단원을 참조하십시오.
2. Amazon Aurora 기계 학습에 대한 지원. 자세한 내용은 [Aurora PostgreSQL에서 기계 학습\(ML\) 사용 \(p. 850\)](#) 단원을 참조하십시오.
3. SQL 처리 기능 향상은 다음과 같습니다.
 - `apg_enable_not_in_transform` 파라미터를 통한 NOT IN의 최적화
 - `apg_enable_semijoin_push_down` 파라미터를 통한 해시 조인에 대한 세미 조인 필터 푸시다운 기능 향상
 - `apg_enable_remove_redundant_inner_joins` 파라미터를 통한 중복 내부 조인 제거의 최적화
 - `ansi_constraint_trigger_ordering`, `ansi_force_foreign_key_checks` 및 `ansi_qualified_update_set_target` 파라미터를 통한 개선된 ANSI 호환성 옵션

자세한 내용은 [Amazon Aurora PostgreSQL 파라미터 \(p. 896\)](#) 단원을 참조하십시오.

4. 새롭고 업데이트된 PostgreSQL 확장은 다음과 같습니다.
 - 새 `aws_ml` 확장. 자세한 내용은 [Aurora PostgreSQL에서 기계 학습\(ML\) 사용 \(p. 850\)](#) 단원을 참조하십시오.
 - 새 `aws_s3` 확장. 자세한 내용은 [Aurora PostgreSQL DB 클러스터의 데이터를 Amazon S3으로 내보내기 \(p. 812\)](#) 단원을 참조하십시오.
 - `apg_plan_mgmt` 확장에 대한 업데이트. 자세한 내용은 [Aurora PostgreSQL용 쿼리 실행 계획 관리 \(p. 821\)](#) 단원을 참조하십시오.

중요한 안정성 기능 향상

1. 드문 경우이지만 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 임시 테이블에 B-트리 인덱스를 생성하는 것과 관련된 버그를 수정했습니다.
2. Aurora PostgreSQL이 RDS PostgreSQL 인스턴스의 물리적 복제본 역할을 할 때 복제와 관련된 버그를 수정했습니다. 드문 경우이지만 이 버그로 인해 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 로그 쓰기 장애가 발생할 수 있습니다.
3. 드문 경우이지만 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 I/O 지연 시간이 높은 읽기 처리와 관련된 버그를 수정했습니다.

우선 순위가 높은 안정성 기능 향상

1. 읽기 노드의 로그-적용 프로세스에서 메모리 누수를 수정했습니다. 이 메모리 누수를 확인하려면 개선 사항 모니터링을 통해 "wal replay" 프로세스에 대한 "메모리 사용량의 %"를 관찰하십시오.
2. 스토리지에서 wal 세그먼트가 제대로 제거되지 않는 논리적 복제와 관련된 버그를 수정했습니다. 이 버그로 인해 스토리지 부풀림이 발생할 수 있습니다. 이를 모니터링하려면 TransactionLogDiskUsage 파라미터를 확인합니다.
3. B-트리 인덱스에서 미리 가져오기 작업을 수행하는 동안 Aurora에서 충돌이 발생할 수 있는 여러 가지 버그를 수정했습니다.
4. 논리적 복제를 사용할 때 Aurora 재시작이 시간 초과될 수 있는 버그를 수정했습니다.
5. 버퍼 캐시의 데이터 블록에 대해 수행되는 유효성 검사 기능을 개선했습니다. 이를 통해 Aurora에서 불일치를 감지하는 기능이 향상되었습니다.

추가 개선 사항 및 기능 향상

1. 쿼리 계획 관리 확장 `apg_plan_mgmt`에는 고도로 분할된 테이블에 대한 계획 생성을 관리하기 위한 개선된 알고리즘이 있습니다.
2. 버퍼 캐시 복구 알고리즘이 개선되어 캐시가 큰 인스턴스에서 시작 시간이 단축되었습니다.
3. PostgreSQL `IWLLock` 우선 순위 지정에 대한 변경 사항을 통해 트랜잭션 속도가 높은 워크로드에서 읽기-노드-적용 프로세스의 성능을 개선했습니다. 이러한 변경 사항은 PostgreSQL `ProcArray`가 큰 경합을 겪고 있는 동안 읽기-노드-적용 프로세스의 결핍을 방지합니다.
4. `vacuum`, 테이블 스캔 및 인덱스 스캔 중 배치 읽기 처리를 개선했습니다. 이로 인해 처리량이 향상되고 CPU 소비량이 낮아집니다.
5. PostgreSQL `SLRU` 자르기 작업을 재생하는 동안 읽기 노드가 충돌할 수 있는 버그를 수정했습니다.
6. 드문 경우이지만 Aurora 로그 레코드의 6개 복사본 중 하나에 의해 반환된 오류로 인해 데이터베이스 쓰기가 중지될 수 있는 버그를 수정했습니다.
7. 크기가 1Gb보다 큰 개별 트랜잭션으로 인해 엔진이 충돌할 수 있는 논리적 복제와 관련된 버그를 수정했습니다.
8. 클러스터 캐시 관리가 활성화된 경우 읽기 노드에서 메모리 누수를 수정했습니다.
9. 소스 스냅샷에 기록되지 않은 관계가 많은 경우 RDS PostgreSQL 스냅샷 가져오기가 중단될 수 있는 버그를 수정했습니다.
10. Aurora 스토리지 데몬이 과도한 I/O 로드에서 충돌할 수 있는 버그를 수정했습니다.
11. 읽기 노드에서 잘못된 트랜잭션 ID epoch를 쓰기 노드에 보고할 수 있는 읽기 노드에 대한 `hot_standby_feedback` 관련 버그를 수정했습니다. 이로 인해 쓰기 노드에서 `hot_standby_feedback`을 무시하고 읽기 노드에서 스냅샷을 무효화할 수 있습니다.
12. `CREATE DATABASE` 문 중에 발생하는 스토리지 오류가 제대로 처리되지 않는 버그를 수정했습니다. 이 버그로 인해 결과 데이터베이스에 액세스할 수 없었습니다. 올바른 동작은 데이터베이스 생성을 실패하고 사용자에게 적절한 오류를 반환하는 것입니다.
13. 읽기 노드가 쓰기 노드에 연결을 시도할 때 PostgreSQL 스냅샷 오버플로의 처리를 개선했습니다. 이렇게 변경하기 전에는 쓰기 노드가 스냅샷 오버플로 상태인 경우 읽기 노드가 조인할 수 없었습니다. 메시지는 PostgreSQL 로그 파일에 다음과 같은 형식으로 표시되었습니다. `DEBUG: recovery snapshot waiting for non-overflowed snapshot or until oldest active xid on standby is at least xxxxxxxx (now yyyyyyyy)` 스냅샷 오버플로는 개별 트랜잭션에서 64개 이상의 하위 트랜잭션을 생성할 때 발생합니다.
14. CTE에 NOT IN 클래스가 있을 때 오류가 잘못 발생하는 일반적인 테이블 표현식과 관련된 버그를 수정했습니다. 이 오류는 `CTE with NOT IN fails with ERROR: could not find CTE CTE-Name`입니다.
15. `aurora_replica_status` 테이블에서 잘못된 `last_error_timestamp` 값과 관련된 버그를 수정했습니다.
16. 임시 객체에 속한 블록으로 공유 버퍼를 채우지 않는 버그를 수정했습니다. 이러한 블록은 백엔드 로컬 버퍼에 올바르게 상주합니다.

17다음과 같이 확장이 변경되었습니다.

- pg_hint_plan을 버전 1.3.4로 업데이트했습니다.
- plprofiler 버전 4.1을 추가했습니다.
- pgTAP 버전 1.0.0을 추가했습니다.

버전 3.0

이 버전의 Aurora PostgreSQL은 PostgreSQL 11.4와 호환됩니다. 릴리스 11.4의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 11.4](#)를 참조하십시오.

Note

초기 릴리스의 경우 지원되는 AWS 리전은 us-east-1, us-east-2, us-west-2, eu-west-1, ap-northeast-1, ap-northeast-2입니다. 전체 AWS 리전 목록은 [Aurora PostgreSQL 리전 가용성 \(p. 5\)](#) 단원을 참조하십시오.

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. 이 릴리스는 [버전 2.3.5 \(p. 912\)](#)에 있는 모든 버그 수정, 기능, 개선 사항을 포함합니다.
2. 파티셔닝 – 파티셔닝 개선 사항에는 해시 파티셔닝 지원, 기본 파티션 생성 지원, 키 열 업데이트를 기반으로 하는 다른 파티션으로의 동적 행 이동이 포함됩니다.
3. 성능 – 성능 향상에는 인덱스 작성 중 병렬화, 구체화된 보기, 해시 조인 및 순차 스캔이 포함되어 작업 수행이 향상됩니다.
4. 저장 프로시저 – 이제 SQL 저장 프로시저에 임베디드 트랜잭션 지원이 추가되었습니다.
5. Autovacuum 개선 – 중요한 로깅을 제공하기 위해 log_autovacuum_min_duration 파라미터가 10초로 설정되고 rds.force_autovacuum_logging 파라미터는 기본적으로 ON으로 설정됩니다. autovacuum 효과를 높이기 위해, 더 큰 기본값을 제공하도록 autovacuum_max_workers 및 autovacuum_vacuum_cost_limit 파라미터의 값이 호스트 메모리 용량을 기반으로 계산됩니다.
6. 향상된 트랜잭션 제한 시간 – 파라미터 idle_in_transaction_session_timeout이 12시간으로 설정됩니다. 12시간 이상 유튜 상태였던 세션은 종료됩니다.
7. tsearch2 모듈은 더 이상 지원되지 않습니다. – 애플리케이션에서 tsearch2 함수를 사용하는 경우 핵심 PostgreSQL 엔진에서 제공하는 것과 동일한 함수를 사용하도록 업데이트하십시오. tsearch2 모듈에 대한 자세한 내용은 [PostgreSQL tsearch2](#) 단원을 참조하십시오.
8. chkpass 모듈은 더 이상 지원되지 않습니다. – chkpass 모듈에 대한 자세한 내용은 [PostgreSQL chkpass](#) 단원을 참조하십시오.
9. 다음 확장 기능을 업데이트하였습니다.
 - address_standardizer를 버전 2.5.1로 업데이트
 - address_standardizer_data_us를 버전 2.5.1로 업데이트
 - btree_gin을 버전 1.3으로 업데이트
 - citext를 버전 1.5로 업데이트
 - cube를 버전 1.4로 업데이트
 - hstore를 버전 1.5로 업데이트
 - ip4r을 버전 2.2로 업데이트
 - isn을 버전 1.2로 업데이트
 - orafce를 버전 3.7로 업데이트
 - pg_hint_plan을 버전 1.3.4로 업데이트
 - pg_prewarm을 버전 1.2로 업데이트
 - pg_repack을 버전 1.4.4로 업데이트
 - pg_trgm을 버전 1.4로 업데이트

- pgaudit을 버전 1.3으로 업데이트
- pgRouting을 버전 2.6.1로 업데이트
- pgtap을 버전 1.0.0으로 업데이트
- plcoffee를 버전 2.3.8로 업데이트
- plls를 버전 2.3.8로 업데이트
- plv8을 버전 2.3.8로 업데이트
- postgis를 버전 2.5.1로 업데이트
- postgis_tiger_geocoder를 버전 2.5.1로 업데이트
- postgis_topology를 버전 2.5.1로 업데이트
- rds_activity_stream을 버전 1.3으로 업데이트

버전 2.4

이 버전의 Aurora PostgreSQL은 PostgreSQL 10.11과 호환됩니다. 릴리스 10.11의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 10.11](#)을 참조하십시오.

이 릴리스에는 여러 가지 중요한 안정성 기능 향상이 포함되어 있습니다. 이전 PostgreSQL 10 엔진을 사용하는 Aurora PostgreSQL 클러스터를 이 릴리스로 업그레이드하는 것이 좋습니다.

이 엔진 버전에서는 다음과 같은 새로운 기능과 개선 사항을 찾을 수 있습니다.

새로운 기능

1. Amazon S3으로의 데이터 내보내기에 대한 지원. 자세한 내용은 [Aurora PostgreSQL DB 클러스터의 데이터를 Amazon S3으로 내보내기 \(p. 812\)](#) 단원을 참조하십시오.
2. Amazon Aurora 기계 학습에 대한 지원. 자세한 내용은 [Aurora PostgreSQL에서 기계 학습\(ML\) 사용 \(p. 850\)](#) 단원을 참조하십시오.
3. SQL 처리 기능 향상은 다음과 같습니다.
 - apg_enable_not_in_transform 파라미터를 통한 NOT IN의 최적화
 - apg_enable_semijoin_push_down 파라미터를 통한 해시 조인에 대한 세미 조인 필터 푸시다운 기능 향상
 - apg_enable_remove_redundant_inner_joins 파라미터를 통한 중복 내부 조인 제거의 최적화
 - ansi_constraint_trigger_ordering, ansi_force_foreign_key_checks 및 ansi_qualified_update_set_target 파라미터를 통한 개선된 ANSI 호환성 옵션

자세한 내용은 [Amazon Aurora PostgreSQL 파라미터 \(p. 896\)](#) 단원을 참조하십시오.

4. 새롭고 업데이트된 PostgreSQL 확장은 다음과 같습니다.
 - 새 aws_ml 확장. 자세한 내용은 [Aurora PostgreSQL에서 기계 학습\(ML\) 사용 \(p. 850\)](#) 단원을 참조하십시오.
 - 새 aws_s3 확장. 자세한 내용은 [Aurora PostgreSQL DB 클러스터의 데이터를 Amazon S3으로 내보내기 \(p. 812\)](#) 단원을 참조하십시오.
 - apg_plan_mgmt 확장에 대한 업데이트. 자세한 내용은 [Aurora PostgreSQL용 쿼리 실행 계획 관리 \(p. 821\)](#) 단원을 참조하십시오.

중요한 안정성 기능 향상

1. 드문 경우이지만 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 임시 테이블에 B-트리 인덱스를 생성하는 것과 관련된 버그를 수정했습니다.
2. Aurora PostgreSQL이 RDS PostgreSQL 인스턴스의 물리적 복제본 역할을 할 때 복제와 관련된 버그를 수정했습니다. 드문 경우이지만 이 버그로 인해 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 로그 쓰기 장애가 발생할 수 있습니다.

3. 드문 경우이지만 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 I/O 지연 시간이 높은 읽기 처리와 관련된 버그를 수정했습니다.

우선 순위가 높은 안정성 기능 향상

1. 읽기 노드의 로그-적용 프로세스에서 메모리 누수를 수정했습니다. 이 메모리 누수를 확인하려면 개선 사항 모니터링을 통해 "wal replay" 프로세스에 대한 "메모리 사용량의 %"를 관찰하십시오.
2. 스토리지에서 wal 세그먼트가 제대로 제거되지 않는 논리적 복제와 관련된 버그를 수정했습니다. 이 버그로 인해 스토리지 부풀림이 발생할 수 있습니다. 이를 모니터링하려면 TransactionLogDiskUsage 파라미터를 확인합니다.
3. B-트리 인덱스에서 미리 가져오기 작업을 수행하는 동안 Aurora에서 충돌이 발생할 수 있는 여러 가지 버그를 수정했습니다.
4. 논리적 복제를 사용할 때 Aurora 재시작이 시간 초과될 수 있는 버그를 수정했습니다.
5. 버퍼 캐시의 데이터 블록에 대해 수행되는 유효성 검사 기능을 개선했습니다. 이를 통해 Aurora에서 불일치를 감지하는 기능이 향상되었습니다.

추가 개선 사항 및 기능 향상

1. 쿼리 계획 관리 확장 `apg_plan_mgmt`에는 고도로 분할된 테이블에 대한 계획 생성을 관리하기 위한 개선된 알고리즘이 있습니다.
2. 버퍼 캐시 복구 알고리즘이 개선되어 캐시가 큰 인스턴스에서 시작 시간이 단축되었습니다.
3. PostgreSQL `LWLock` 우선 순위 지정에 대한 변경 사항을 통해 트랜잭션 속도가 높은 워크로드에서 읽기-노드-적용 프로세스의 성능을 개선했습니다. 이러한 변경 사항은 PostgreSQL `ProcArray`가 큰 경합을 겪고 있는 동안 읽기-노드-적용 프로세스의 결핍을 방지합니다.
4. `vacuum`, 테이블 스캔 및 인덱스 스캔 중 배치 읽기 처리를 개선했습니다. 이로 인해 처리량이 향상되고 CPU 소비량이 낮아집니다.
5. PostgreSQL `SLRU` 자르기 작업을 재생하는 동안 읽기 노드가 충돌할 수 있는 버그를 수정했습니다.
6. 드문 경우이지만 Aurora 로그 레코드의 6개 복사본 중 하나에 의해 반환된 오류로 인해 데이터베이스 쓰기가 중지될 수 있는 버그를 수정했습니다.
7. 크기가 1Gb보다 큰 개별 트랜잭션으로 인해 엔진이 충돌할 수 있는 논리적 복제와 관련된 버그를 수정했습니다.
8. 클러스터 캐시 관리가 활성화된 경우 읽기 노드에서 메모리 누수를 수정했습니다.
9. 소스 스냅샷에 기록되지 않은 관계가 많은 경우 RDS PostgreSQL 스냅샷 가져오기가 중단될 수 있는 버그를 수정했습니다.
10. Aurora 스토리지 데몬이 과도한 I/O 로드에서 충돌할 수 있는 버그를 수정했습니다.
11. 읽기 노드에서 잘못된 트랜잭션 ID epoch를 쓰기 노드에 보고할 수 있는 읽기 노드에 대한 `hot_standby_feedback` 관련 버그를 수정했습니다. 이로 인해 쓰기 노드에서 `hot_standby_feedback`을 무시하고 읽기 노드에서 스냅샷을 무효화할 수 있습니다.
12. `CREATE DATABASE` 문 중에 발생하는 스토리지 오류가 제대로 처리되지 않는 버그를 수정했습니다. 이 버그로 인해 결과 데이터베이스에 액세스할 수 없었습니다. 올바른 동작은 데이터베이스 생성을 실패하고 사용자에게 적절한 오류를 반환하는 것입니다.
13. 읽기 노드가 쓰기 노드에 연결을 시도할 때 PostgreSQL 스냅샷 오버플로의 처리를 개선했습니다. 이렇게 변경하기 전에는 쓰기 노드가 스냅샷 오버플로 상태인 경우 읽기 노드가 조인할 수 없었습니다. 메시지는 PostgreSQL 로그 파일에 다음과 같은 형식으로 표시되었습니다. `DEBUG: recovery snapshot waiting for non-overflowed snapshot or until oldest active xid on standby is at least xxxxxxxx (now yyyyyyyy)` 스냅샷 오버플로는 개별 트랜잭션에서 64개 이상의 하위 트랜잭션을 생성할 때 발생합니다.
14. CTE에 NOT IN 클래스가 있을 때 오류가 잘못 발생하는 일반적인 테이블 표현식과 관련된 버그를 수정했습니다. 이 오류는 `CTE with NOT IN fails with ERROR: could not find CTE CTE-Name`입니다.

- 15aurora_replica_status 테이블에서 잘못된 last_error_timestamp 값과 관련된 버그를 수정했습니다.
- 16.임시 객체에 속한 블록으로 공유 버퍼를 채우지 않는 버그를 수정했습니다. 이러한 블록은 백엔드 로컬 버퍼에 올바르게 상주합니다.
- 17.GIN 인덱스에서 vacuum 정리 성능을 개선했습니다
- 18.드문 경우이지만 복제 스트림이 유휴 상태인 경우에도 RDS PostgreSQL 인스턴스의 복제본 역할을 하는 동안 Aurora에서 100% CPU 사용률을 표시하는 버그를 수정했습니다.
- 19.PostgreSQL 11의 변경 사항을 백포트하여 분리된 임시 테이블의 정리를 개선했습니다. 드문 경우이지만 이 변경 사항이 없으면 분리된 임시 테이블로 인해 트랜잭션 ID 랩어라운드가 발생할 수 있습니다. 자세한 내용은 이 [PostgreSQL 커뮤니티 커밋](#)을 참조하십시오.
- 20.초기화되지 않은 시작 프로세스가 있는 동안 쓰기 인스턴스가 읽기 인스턴스의 복제 등록 요청을 수락할 수 있는 버그를 수정했습니다.
- 21.다음과 같이 확장이 변경되었습니다.
 - pg_hint_plan을 버전 1.3.3으로 업데이트했습니다.
 - plprofiler 버전 4.1을 추가했습니다.

버전 2.3

이 버전의 Aurora PostgreSQL은 PostgreSQL 10.7과 호환됩니다. 릴리스 10.7의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 10.7](#)을 참조하십시오.

패치 버전

- [버전 2.3.5 \(p. 912\)](#)
- [버전 2.3.3 \(p. 912\)](#)
- [버전 2.3.1 \(p. 913\)](#)
- [버전 2.3.0 \(p. 913\)](#)

버전 2.3.5

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. DB 인스턴스가 다시 시작하는 원인이 될 수 있는 버그를 수정했습니다.
2. 논리적 복제를 사용하는 중에 백엔드가 종료되면서 발생하는 충돌의 원인일 수 있는 버그를 수정했습니다.
3. 장애 조치 중에 읽기가 발생할 때 다시 시작되는 현상의 원인일 수 있는 버그를 수정했습니다.
4. 논리적 복제용 wal2json 플러그인에서 버그를 수정했습니다.
5. 일관성이 없는 메타데이터의 원인일 수 있는 버그를 수정했습니다.

버전 2.3.3

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. PostgreSQL 커뮤니티 보안 문제인 CVE-2019-10130에서 백포트가 수정되었습니다.
2. PostgreSQL 커뮤니티 보안 문제인 CVE-2019-10164에서 백포트가 수정되었습니다.
3. 데이터 스트리밍 작업에서 CPU 시간을 초과하여 사용할 수 있는 버그가 수정되었습니다.

4. B-트리 인덱스를 스캔하는 병렬 스레드가 디스크 읽기 이후 중단될 수 있는 버그가 수정되었습니다.
 5. 공통 테이블 표현식(CTE)에 대해 `not in Predicate`를 사용할 경우 "ERROR: bad levelsup for CTE"라는 오류 메시지가 반환될 수 있는 버그가 수정되었습니다.
 6. 일반화된 검색 트리(GIST) 인덱스를 수정할 때 읽기 노드 재생 프로세스가 중단될 수 있는 버그가 수정되었습니다.
 7. 읽기 노드로 장애 조치 이후 가시성 맵 페이지에 잘못된 동결 비트가 포함될 수 있는 버그가 수정되었습니다.
 8. 인덱스 유지 관리 과정에서 쓰기 노드와 읽기 노드 사이의 로그 트래픽에 최적화되었습니다.
 9. B-트리 인덱스 스캔 도중 읽기 노드에 대한 쿼리가 충돌을 일으킬 수 있는 버그가 수정되었습니다.
 10. 중복된 내부 조인 제거에 최적화된 쿼리가 충돌을 일으킬 수 있는 버그가 수정되었습니다.
11. `aurora_stat_memctx_usage` 함수가 이제 임의 컨텍스트 이름의 인스턴스 수를 보고합니다.
12. `aurora_stat_memctx_usage` 함수가 잘못된 결과를 보고했던 버그가 수정되었습니다.
13. 읽기 노드 재생 프로세스가 구성된 `max_standby_streaming_delay` 값을 넘어 충돌을 일으킨 쿼리를 종료할 때까지 대기하던 버그가 수정되었습니다.
14. 이제 활성 상태의 연결이 재생 프로세스와 충돌을 일으키면 추가 정보가 읽기 노드에 기록됩니다.
15. 파티션으로 분할된 테이블에서 삭제할 때 충돌을 일으킬 수 있는 PostgreSQL 커뮤니티 버그 #15677에 대해 백포트가 수정되었습니다.

버전 2.3.1

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. 엔진 충돌을 일으킨 I/O 미리 가져오기와 관련된 여러 버그가 해결되었습니다.

버전 2.3.0

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. 이제 Aurora PostgreSQL이 B-트리 인덱스를 스캔하는 동안 I/O 미리 가져오기를 수행합니다. 따라서 캐시 되지 않은 데이터에 대한 B-트리 스캔의 성능이 크게 향상됩니다.

개선 사항

1. "LWLocks를 너무 많이 가져옴" 오류로 인해 읽기 노드가 실패할 수 있는 버그가 해결되었습니다.
2. 클러스터의 쓰기 워크로드가 높을 때 읽기 노드를 시작하지 못하도록 만든 많은 문제를 해결했습니다.
3. `aurora_stat_memctx_usage()` 함수 사용 시 충돌이 발생할 수 있는 버그가 해결되었습니다.
4. 버퍼 캐시의 스레싱을 최소화하기 위해 테이블 스캔에서 사용하는 캐시 대체 전략이 개선되었습니다.

버전 2.2

이 버전의 Aurora PostgreSQL은 PostgreSQL 10.6과 호환됩니다. 릴리스 10.6의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 10.6](#)을 참조하십시오.

파치 버전

- [버전 2.2.1 \(p. 914\)](#)
- [버전 2.2.0 \(p. 914\)](#)

버전 2.2.1

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. 논리적 복제의 안정성이 향상되었습니다.
2. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "CLOG 세그먼트 123이 존재하지 않음: 그러한 파일 또는 디렉터리가 없음"과 같은 형식이 될 것입니다.
3. 지원되는 IAM 암호의 크기가 8KB로 늘어났습니다.
4. 높은 처리량 쓰기 워크로드에서 성능의 일관성이 향상되었습니다.
5. 다시 시작하는 중에 읽기 복제본이 충돌하는 원인이 될 수 있는 버그를 수정하였습니다.
6. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "SQL 오류: 관계의 과거 EOF를 읽으려고 시도하는 중"의 형식이 될 것입니다.
7. 다시 시작한 후 메모리 사용량 증가의 원인이 될 수 있는 버그를 수정하였습니다.
8. 다수의 하위 트랜잭션과의 트랜잭션이 실패하는 원인이 될 수 있는 버그를 수정하였습니다.
9. GIN 인덱스 사용 시 발생할 수 있는 장애에 대처하는 커뮤니티 PostgreSQL의 패치를 병합하였습니다. 자세한 내용은 <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbb49a493045c8d8086a9b15d95b8f18>를 참조하십시오.

10RDS for PostgreSQL에서 스냅샷 가져오기가 실패하는 원인이 될 수 있는 버그를 수정하였습니다.

버전 2.2.0

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. 제한된 암호 관리 기능을 추가했습니다. 제한된 암호 관리를 통해 `rds.restrict_password_commands` 파라미터와 `rds_password` 역할을 사용하여 사용자 암호 및 암호 만료 변경을 관리할 수 있는 사람을 제한할 수 있습니다. 자세한 내용은 [암호 관리 제한 \(p. 774\)](#) 단원을 참조하십시오.

버전 2.1

이 Aurora PostgreSQL 버전은 PostgreSQL 10.5와 호환됩니다. 릴리스 10.5의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 10.5](#)를 참조하십시오.

패치 버전

- [버전 2.1.1 \(p. 914\)](#)
- [버전 2.1.0 \(p. 915\)](#)

버전 2.1.1

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "CLOG 세그먼트 123이 존재하지 않음: 그러한 파일 또는 디렉터리가 없음"과 같은 형식이 될 것입니다.
2. 지원되는 IAM 암호의 크기가 8KB로 늘어났습니다.
3. 높은 처리량 쓰기 워크로드에서 성능의 일관성이 향상되었습니다.

4. 다시 시작하는 중에 읽기 복제본이 충돌하는 원인이 될 수 있는 버그를 수정하였습니다.
5. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "SQL 오류: 관계의 과거 EOF를 읽으려고 시도하는 중"의 형식이 될 것입니다.
6. 다시 시작한 후 메모리 사용량 증가의 원인이 될 수 있는 버그를 수정하였습니다.
7. 다수의 하위 트랜잭션과의 트랜잭션이 실패하는 원인이 될 수 있는 버그를 수정하였습니다.
8. GIN 인덱스 사용 시 발생할 수 있는 장애에 대처하는 커뮤니티 PostgreSQL의 패치를 병합하였습니다. 자세한 내용은 <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>를 참조하십시오.
9. RDS for PostgreSQL에서 스냅샷 가져오기가 실패하는 원인이 될 수 있는 버그를 수정하였습니다.

버전 2.1.0

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. Aurora 쿼리 계획 관리 기능이 일반 공개되었습니다. 이 기능을 통해 고객은 애플리케이션에서 사용하는 임의 또는 모든 쿼리 계획을 추적 및 관리하고, 쿼리 최적화 프로그램 계획 선택을 제어하며, 높고 안정적인 애플리케이션이 안정적으로 고성능을 발휘하도록 보장할 수 있습니다. 자세한 내용은 [Aurora PostgreSQL용 쿼리 실행 계획 관리 \(p. 821\)](#) 단원을 참조하십시오.
2. libprotobuf 확장을 1.3.0 버전으로 업데이트했습니다. 이 기능은 PostGIS 확장에 사용됩니다.
3. pg_similarity 확장을 1.0 버전으로 업데이트했습니다.
4. log_fdw 확장을 1.1 버전으로 업데이트했습니다.
5. pg_hint_plan 확장을 1.3.1 버전으로 업데이트했습니다.

개선 사항

1. 이제는 라이터와 리더 노드 간의 네트워크 트래픽이 압축되어 네트워크 사용률이 감소됩니다. 따라서 네트워크 포화로 인해 읽기 노드를 사용할 수 없게 될 확률이 감소합니다.
2. PostgreSQL 하위 트랜잭션에 대해 고성능의 확장 가능한 하위 시스템을 구현했습니다. 이로써 저장 포인트와 PL/pgSQL 예외 핸들러를 광범위하게 사용하는 애플리케이션의 성능이 향상됩니다.
3. rds_superuser 역할이 이제는 세션, 데이터베이스 또는 역할 수준에서 다음 파라미터를 설정할 수 있습니다.
 - log_duration
 - log_error_verbosity
 - log_executor_stats
 - log_lock_waits
 - log_min_duration_statement
 - log_min_error_statement
 - log_min_messages
 - log_parser_stats
 - log_planner_stats
 - log_replication_commands
 - log_statement_stats
 - log_temp_files
4. SQL 명령 "ALTER FUNCTION ... OWNER TO ..."가 실행되지 않고 "improper qualified name (too many dotted names)(잘못된 적격 이름(점으로 구분된 이름이 너무 많음))" 오류가 발생했던 버그를 수정했습니다.
5. 200만개를 초과하는 하위 트랜잭션을 포함하는 트랜잭션을 커밋하는 동안 충돌이 발생할 수 있는 버그를 수정했습니다.

6. Aurora Storage 볼륨을 사용할 수 없게 했던 GIN 인덱스 관련 커뮤니티 PostgreSQL 코드 버그를 수정했습니다.
7. PostgreSQL용 RDS 인스턴스의 Aurora PostgreSQL 복제본을 시작하지 못할 수 있고 "PANIC: could not locate a valid checkpoint record.(경고: 유효하지 않은 체크포인트 레코드를 찾지 못했습니다.)" 오류가 보고되던 버그를 수정했습니다.
8. 잘못된 파라미터를 `aurora_stat_backend_waits` 함수에 전달하여 충돌이 발생했던 문제를 수정했습니다.

알려진 문제

1. `pageinspect` 확장은 Aurora PostgreSQL에서 지원되지 않습니다.

버전 2.0

이 버전의 Aurora PostgreSQL는 PostgreSQL 10.4와 호환됩니다. 릴리스 10.5의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 10.4](#)를 참조하십시오.

패치 버전

- [버전 2.0.1 \(p. 916\)](#)
- [버전 2.0.0 \(p. 916\)](#)

버전 2.0.1

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "CLOG 세그먼트 123이 존재하지 않음: 그러한 파일 또는 디렉터리가 없음"과 같은 형식이 될 것입니다.
2. 지원되는 IAM 암호의 크기가 8KB로 늘어났습니다.
3. 높은 처리량 쓰기 워크로드에서 성능의 일관성이 향상되었습니다.
4. 다시 시작하는 중에 읽기 복제본이 충돌하는 원인이 될 수 있는 버그를 수정하였습니다.
5. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "SQL 오류: 관계 의 과거 EOF를 읽으려고 시도하는 중"의 형식이 될 것입니다.
6. 다시 시작한 후 메모리 사용량 증가의 원인이 될 수 있는 버그를 수정하였습니다.
7. 다수의 하위 트랜잭션과의 트랜잭션이 실패하는 원인이 될 수 있는 버그를 수정하였습니다.
8. GIN 인덱스 사용 시 발생할 수 있는 장애에 대처하는 커뮤니티 PostgreSQL의 패치를 병합하였습니다. 자세한 내용은 <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>를 참조하십시오.
9. RDS for PostgreSQL에서 스냅샷 가져오기가 실패하는 원인이 될 수 있는 버그를 수정하였습니다.

버전 2.0.0

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. 이 릴리스는 [버전 1.3 \(p. 920\)](#)에서 제공하는 모든 버그 수정, 기능, 개선 사항을 포함합니다.
2. 임시 파일 크기 제한은 사용자 구성이 가능합니다. `temp_file_limit` 파라미터를 수정하려면 `rds_superuser` 역할이 필요합니다.
3. PostGIS 확장에서 사용하는 GDAL 라이브러리를 업데이트했습니다.

4. `ip4r` 확장을 2.1.1 버전으로 업데이트했습니다.
5. `pg_repack` 확장을 1.4.3 버전으로 업데이트했습니다.
6. `plv8` 확장을 2.1.2 버전으로 업데이트했습니다.
7. 병렬 쿼리 - 새로운 Aurora PostgreSQL 버전 2.0 인스턴스를 생성하는 경우 병렬 쿼리가 `default_postgres10` 파라미터 그룹에 대해 활성화됩니다. `max_parallel_workers_per_gather` 파라미터는 기본적으로 2로 설정되지만, 특정 워크로드 요구 사항을 지원하도록 수정할 수 있습니다.
8. 쓰기 노드에서 특정 유형의 여유 공간이 변경된 후 읽기 노드가 충돌하는 원인이 될 수 있는 버그를 수정하였습니다.

버전 1.6

이 버전의 Aurora PostgreSQL은 PostgreSQL 9.6.16과 호환됩니다. 릴리스 9.6.16의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 9.6.16](#)을 참조하십시오.

이 릴리스에는 여러 가지 중요한 안정성 기능 향상이 포함되어 있습니다. 이전 PostgreSQL 9.6 엔진을 사용하는 Aurora PostgreSQL 클러스터를 이 릴리스로 업그레이드하는 것이 좋습니다.

이 엔진 버전에서는 다음과 같은 새로운 기능과 개선 사항을 찾을 수 있습니다.

새로운 기능

1. `apg_plan_mgmt` 확장에 대한 업데이트. 자세한 내용은 [Aurora PostgreSQL용 쿼리 실행 계획 관리](#) (p. 821) 단원을 참조하십시오.

중요한 안정성 기능 향상

1. 드문 경우이지만 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 임시 테이블에 B-트리 인덱스를 생성하는 것과 관련된 버그를 수정했습니다.
2. Aurora PostgreSQL이 RDS PostgreSQL 인스턴스의 물리적 복제본 역할을 할 때 복제와 관련된 버그를 수정했습니다. 드문 경우이지만 이 버그로 인해 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 로그 쓰기 장애가 발생할 수 있습니다.
3. 드문 경우이지만 복구 시간이 길어지고 가용성에 영향을 줄 수 있는 I/O 지연 시간이 높은 읽기 처리와 관련된 버그를 수정했습니다.

우선 순위가 높은 안정성 기능 향상

1. 읽기 노드의 로그-적용 프로세스에서 메모리 누수를 수정했습니다. 이 메모리 누수를 확인하려면 개선 사항 모니터링을 통해 "wal replay" 프로세스에 대한 "메모리 사용량의 %"를 관찰하십시오.
2. B-트리 인덱스에서 미리 가져오기 작업을 수행하는 동안 Aurora에서 충돌이 발생할 수 있는 여러 가지 버그를 수정했습니다.
3. 버퍼 캐시의 데이터 블록에 대해 수행되는 유효성 검사 기능을 개선했습니다. 이를 통해 Aurora에서 불일치를 감지하는 기능이 향상되었습니다.

추가 개선 사항 및 기능 향상

1. 쿼리 계획 관리 확장 `apg_plan_mgmt`에는 고도로 분할된 테이블에 대한 계획 생성을 관리하기 위한 개선된 알고리즘이 있습니다.
2. 버퍼 캐시 알고리즘이 개선되어 캐시가 큰 인스턴스에서 시작 시간이 단축되었습니다.
3. PostgreSQL `LWLock` 우선 순위 지정에 대한 변경 사항을 통해 트랜잭션 속도가 높은 워크로드에서 읽기-노드-적용 프로세스의 성능을 개선했습니다. 이러한 변경 사항은 PostgreSQL `ProcArray`가 큰 경합을 겪고 있는 동안 읽기-노드-적용 프로세스의 결핍을 방지합니다.
4. PostgreSQL `SLRU` 자르기 작업을 재생하는 동안 읽기 노드가 충돌할 수 있는 버그를 수정했습니다.

5. 드문 경우이지만 Aurora 로그 레코드의 6개 복사본 중 하나에 의해 반환된 오류로 인해 데이터베이스 쓰기가 중지될 수 있는 버그를 수정했습니다.
6. 클러스터 캐시 관리가 활성화된 경우 읽기 노드에서 메모리 누수를 수정했습니다.
7. 소스 스냅샷에 기록되지 않은 관계가 많은 경우 RDS PostgreSQL 스냅샷 가져오기가 중단될 수 있는 버그를 수정했습니다.
8. 읽기 노드에서 잘못된 트랜잭션 ID epoch를 쓰기 노드에 보고할 수 있는 읽기 노드에 대한 hot_standby_feedback 관련 버그를 수정했습니다. 이로 인해 쓰기 노드에서 hot_standby_feedback을 무시하고 읽기 노드에서 스냅샷을 무효화할 수 있습니다.
9. CREATE DATABASE 문 중에 발생하는 스토리지 오류가 제대로 처리되지 않는 버그를 수정했습니다. 이 오류로 인해 결과 데이터베이스에 액세스할 수 없었습니다. 올바른 동작은 데이터베이스 생성을 실패하고 사용자에게 적절한 오류를 반환하는 것입니다.
10. 읽기 노드가 쓰기 노드에 연결을 시도할 때 PostgreSQL 스냅샷 오버플로의 처리를 개선했습니다. 이렇게 변경하기 전에는 쓰기 노드가 스냅샷 오버플로 상태인 경우 읽기 노드가 조인할 수 없었습니다. 메시지는 PostgreSQL 로그 파일에 다음과 같은 형식으로 표시되었습니다. DEBUG: recovery snapshot waiting for non-overflowed snapshot or until oldest active xid on standby is at least **xxxxxxxx** (now **yyyyyyyy**) 스냅샷 오버플로는 개별 트랜잭션에서 64개 이상의 하위 트랜잭션을 생성할 때 발생합니다.
- 11 CTE에 NOT IN 클래스가 있을 때 오류가 잘못 발생하는 일반적인 테이블 표현식과 관련된 버그를 수정했습니다. 이 오류는 CTE with NOT IN fails with ERROR: could not find CTE **CTE-Name**입니다.
- 12 aurora_replica_status 테이블에서 잘못된 last_error_timestamp 값과 관련된 버그를 수정했습니다.
- 13 임시 객체에 속한 블록으로 공유 버퍼를 채우지 않는 버그를 수정했습니다. 이러한 블록은 백엔드 로컬 버퍼에 옮바르게 상주합니다.
- 14 드문 경우이지만 복제 스트림이 유휴 상태인 경우에도 RDS PostgreSQL 인스턴스의 복제본 역할을 하는 동안 Aurora에서 100% CPU 사용률을 표시하는 버그를 수정했습니다.
- 15 PostgreSQL 11의 변경 사항을 백포트하여 분리된 임시 테이블의 정리를 개선했습니다. 드문 경우이지만 이 변경 사항이 없으면 분리된 임시 테이블로 인해 트랜잭션 ID 랩어라운드가 발생할 수 있습니다. 자세한 내용은 이 [PostgreSQL 커뮤니티 커밋](#)을 참조하십시오.
- 16 초기화되지 않은 시작 프로세스가 있는 동안 쓰기 인스턴스가 읽기 인스턴스의 복제 등록 요청을 수락할 수 있는 버그를 수정했습니다.
- 17 다음과 같이 확장이 변경되었습니다.
 - pg_hint_plan을 버전 1.2.5로 업데이트했습니다.

버전 1.5

이 버전의 Aurora PostgreSQL은 PostgreSQL 9.6.12와 호환됩니다. 릴리스 9.6.12의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 9.6.12](#)를 참조하십시오.

패치 버전

- [버전 1.5.3 \(p. 918\)](#)
- [버전 1.5.2 \(p. 919\)](#)
- [버전 1.5.1 \(p. 919\)](#)
- [버전 1.5.0 \(p. 919\)](#)

버전 1.5.3

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. DB 인스턴스가 다시 시작하는 원인이 될 수 있는 버그를 수정했습니다.

2. 장애 조치 중에 읽기가 발생할 때 다시 시작되는 현상의 원인일 수 있는 버그를 수정했습니다.
3. 일관성이 없는 메타데이터의 원인일 수 있는 버그를 수정했습니다.

버전 1.5.2

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. PostgreSQL 커뮤니티 보안 문제인 CVE-2019-10130에서 백포트가 수정되었습니다.
2. 일반화된 검색 트리(GIST) 인덱스를 수정할 때 읽기 노드 재생 프로세스가 중단될 수 있는 버그가 수정되었습니다.
3. 읽기 노드로 장애 조치 이후 가시성 맵 페이지에 잘못된 동결 비트가 포함될 수 있는 버그가 수정되었습니다.
4. 오류 메시지인 "relation relation-name does not exist"가 잘못 보고되는 버그가 수정되었습니다.
5. 인덱스 유지 관리 과정에서 쓰기 노드와 읽기 노드 사이의 로그 트래픽에 최적화되었습니다.
6. B-트리 인덱스 스캔 도중 읽기 노드에 대한 쿼리가 충돌을 일으킬 수 있는 버그가 수정되었습니다.
7. `aurora_stat_memctx_usage` 함수가 이제는 임의 컨텍스트 이름의 인스턴스 수를 보고합니다.
8. `aurora_stat_memctx_usage` 함수가 잘못된 결과를 보고했던 버그가 수정되었습니다.
9. 읽기 노드 재생 프로세스가 구성된 `max_standby_streaming_delay`를 넘어 충돌을 일으킨 쿼리를 종료할 때까지 대기하던 버그가 수정되었습니다.
10. 이제 활성 상태의 연결이 재생 프로세스와 충돌을 일으키면 추가 정보가 읽기 노드에 기록됩니다.

버전 1.5.1

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. 엔진 충돌을 일으킨 I/O 미리 가져오기와 관련된 여러 버그가 해결되었습니다.

버전 1.5.0

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. 이제 Aurora PostgreSQL이 B-트리 인덱스를 스캔하는 동안 I/O 미리 가져오기를 수행합니다. 따라서 캐시 되지 않은 데이터에 대한 B-트리 스캔의 성능이 크게 향상됩니다.

개선 사항

1. 클러스터의 쓰기 워크로드가 높을 때 읽기 노드를 시작하지 못하도록 만든 많은 문제를 해결했습니다.
2. `aurora_stat_memctx_usage()` 함수 사용 시 충돌이 발생할 수 있는 버그가 해결되었습니다.
3. 버퍼 캐시의 스레싱을 최소화하기 위해 테이블 스캔에서 사용하는 캐시 대체 전략이 개선되었습니다.

버전 1.4

이 버전의 Aurora PostgreSQL은 PostgreSQL 9.6.11과 호환됩니다. 릴리스 9.6의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 9.6.11](#)을 참조하십시오.

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. `pg_similarity` 확장 버전 1.0.0에 대한 지원이 추가되었습니다.

개선 사항

1. 이 릴리스는 [버전 1.3 \(p. 920\)](#)에서 제공하는 모든 버그 수정, 기능, 개선 사항을 포함합니다.
2. 이제는 라이터와 리더 노드 간의 네트워크 트래픽이 압축되어 네트워크 사용률이 감소됩니다. 따라서 네트워크 포화로 인해 읽기 노드를 사용할 수 없게 될 확률이 감소합니다.
3. 높은 동시성 워크로드에서 하위 트랜잭션의 성능이 향상되었습니다.
4. `pg_hint_plan` 버전 1.2.3의 확장에 대한 업데이트.
5. 사용 중인 시스템에서 수백만 개의 하위 트랜잭션을 포함한 커밋(때로는 커밋 타임스탬프가 활성화된 상태)으로 인해 Aurora가 충돌하는 문제를 해결했습니다.
6. `VALUES`가 포함된 `INSERT`가 실패하면서 "관계의 과거 EOF 읽기 시도 중"이라는 메시지가 표시될 수 있는 문제를 해결했습니다.
7. `apg_plan_mgmt` 확장을 1.0.1 버전으로 업그레이드. 자세한 내용은 [apg_plan_mgmt 확장 버전 1.0.1 \(p. 927\)](#)을(를) 참조하십시오.

`apg_plan_mgmt` 확장은 쿼리 계획 관리와 함께 사용됩니다. `apg_plan_mgmt` 설치, 업그레이드 및 사용에 관한 자세한 내용은 [Aurora PostgreSQL용 쿼리 실행 계획 관리 \(p. 821\)](#) 단원을 참조하십시오.

버전 1.3

이 버전의 Aurora PostgreSQL은 PostgreSQL 9.6.9와 호환됩니다. 릴리스 9.6.9의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 9.6.9](#)를 참조하십시오.

패치 버전

- [버전 1.3.2 \(p. 920\)](#)
- [버전 1.3.0 \(p. 921\)](#)

버전 1.3.2

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. `ProcArrayGroupUpdate` 대기 이벤트를 추가했습니다.

개선 사항

1. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "CLOG 세그먼트 123이 존재하지 않음: 그러한 파일 또는 디렉터리가 없음"과 같은 형식이 될 것입니다.
2. 지원되는 IAM 암호의 크기가 8KB로 늘어났습니다.
3. 높은 처리량 쓰기 워크로드에서 성능의 일관성이 향상되었습니다.
4. 다시 시작하는 중에 읽기 복제본이 충돌하는 원인이 될 수 있는 버그를 수정하였습니다.
5. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "SQL 오류: 관계의 과거 EOF를 읽으려고 시도하는 중"의 형식이 될 것입니다.
6. 다시 시작한 후 메모리 사용량 증가의 원인이 될 수 있는 버그를 수정하였습니다.

7. 다수의 하위 트랜잭션과의 트랜잭션이 실패하는 원인이 될 수 있는 버그를 수정하였습니다.
8. GIN 인덱스 사용 시 발생할 수 있는 장애에 대처하는 커뮤니티 PostgreSQL의 패치를 병합하였습니다. 자세한 내용은 <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=9e66f2fb9a493045c8d8086a9b15d95b8f18>를 참조하십시오.
9. RDS for PostgreSQL에서 스냅샷 가져오기가 실패하는 원인이 될 수 있는 버그를 수정하였습니다.

버전 1.3.0

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

개선 사항

1. 이 릴리스는 [버전 1.2 \(p. 922\)](#)에서 제공하는 모든 버그 수정, 기능, 개선 사항을 포함합니다.
2. PostGIS 확장에서 사용하는 GDAL 라이브러리를 업데이트했습니다.
3. 다음 PostgreSQL 확장 기능을 업데이트했습니다.
 - ip4r을 버전 2.1.1로 업데이트했습니다.
 - pgaudit을 버전 1.1.1로 업데이트했습니다.
 - pg_repack을 버전 1.4.3으로 업데이트했습니다.
 - plv8을 버전 2.1.2로 업데이트했습니다.
4. 로컬 디스크 사용량이 높을 때 장애 조치를 잘못 발생시킬 수 있는 모니터링 시스템의 문제를 수정했습니다.
5. Aurora PostgreSQL이 반복적으로 충돌하여 다음을 보고하는 버그를 수정했습니다.

```
PANIC: new_record_total_len (8201) must be less than BLCKSZ (8192), rmid (6), info (32)
```

6. 대용량 버퍼 캐시 복구로 인해 Aurora PostgreSQL 읽기 노드가 클러스터에 다시 조인할 수 없는 버그를 수정했습니다. 이 문제는 r4.16xlarge. 외의 다른 인스턴스에서는 거의 발생할 가능성이 낮습니다.
7. 9.4 이전의 엔진 버전에서 가져온 빈 GIN 인덱스 리프 페이지에 삽입할 때 Aurora 스토리지 볼륨을 사용할 수 없게 되는 버그를 수정했습니다.
8. 드물게 트랜잭션 커밋 중에 충돌이 발생할 경우 커밋 트랜잭션의 CommitTs 데이터가 손실될 수 있는 버그를 수정했습니다. 트랜잭션의 실제 내구성은 이 버그의 영향을 받지 않았습니다.
9. PostGIS가 gserialized_gist_picksplit_2d() 함수에서 충돌할 수 있는 PostGIS 확장 기능의 버그를 수정했습니다.

10r4.8xI보다 작은 인스턴스에서 대규모 쓰기 트래픽을 수행할 때 읽기 전용 노드의 안정성이 향상되었습니다. 특히 라이터(writer)와 리더(reader) 사이의 네트워크 대역폭이 제한된 상황을 개선했습니다.

11RDS for PostgreSQL 인스턴스의 복제 대상 역할을 하는 Aurora PostgreSQL 인스턴스가 다음 오류와 충돌하는 버그를 수정했습니다.

```
FATAL: could not open file "base/16411/680897_vm": No such file or directory during "xlog redo at 782/3122D540 for Storage/TRUNCATE"
```

12.읽기 전용 노드에서 "aurora wal replay process"의 힙 크기가 계속 커지는 메모리 누수를 수정했습니다. 확장 모니터링을 통해 이를 관찰할 수 있습니다.

13PostgreSQL 로그에 다음 메시지가 보고되면서 Aurora PostgreSQL이 시작하지 못하는 버그를 수정했습니다.

```
FATAL: Storage initialization failed.
```

14LWLock:buffer_content 및 IO:ControlFileSyncUpdate 이벤트에서 대기를 야기하는 대용량 쓰기 워크로드의 성능 한도를 수정했습니다.

15쓰기 노드에서 특정 유형의 여유 공간이 변경된 후 읽기 노드가 충돌하는 원인이 될 수 있는 버그를 수정하였습니다.

버전 1.2

이 버전의 Aurora PostgreSQL은 PostgreSQL 9.6.8과 호환됩니다. 릴리스 9.6.8의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 9.6.8을 참조하십시오.](#)

패치 버전

- [버전 1.2.2 \(p. 922\)](#)
- [버전 1.2.0 \(p. 922\)](#)

버전 1.2.2

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. `ProcArrayGroupUpdate` 대기 이벤트를 추가했습니다.

개선 사항

1. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "CLOG 세그먼트 123이 존재하지 않음: 그러한 파일 또는 디렉터리가 없음"과 같은 형식이 될 것입니다.
2. 지원되는 IAM 암호의 크기가 8KB로 늘어났습니다.
3. 높은 처리량 쓰기 워크로드에서 성능의 일관성이 향상되었습니다.
4. 다시 시작하는 중에 읽기 복제본이 충돌하는 원인이 될 수 있는 버그를 수정하였습니다.
5. 쿼리 실행 중 오류 발생의 원인이 될 수 있는 버그를 수정하였습니다. 보고되는 메시지는 "SQL 오류: 관계의 과거 EOF를 읽으려고 시도하는 중"의 형식이 될 것입니다.
6. 다시 시작한 후 메모리 사용량 증가의 원인이 될 수 있는 버그를 수정하였습니다.
7. 다수의 하위 트랜잭션과의 트랜잭션이 실패하는 원인이 될 수 있는 버그를 수정하였습니다.
8. GIN 인덱스 사용 시 발생할 수 있는 장애에 대처하는 커뮤니티 PostgreSQL의 패치를 병합하였습니다. 자세한 내용은 <https://git.postgresql.org/gitweb/?p=postgresql.git;a=commit;h=f9e66f2fbbb49a493045c8d8086a9b15d95b8f18>를 참조하십시오.
9. RDS for PostgreSQL에서 스냅샷 가져오기가 실패하는 원인이 될 수 있는 버그를 수정하였습니다.

버전 1.2.0

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. `aurora_stat_memctx_usage()` 함수를 도입했습니다. 이 함수는 각 PostgreSQL 백엔드의 내부 메모리 컨텍스트 사용을 보고합니다. 이 함수를 사용하여 특정 백엔드가 많은 양의 메모리를 소모하는 이유를 확인할 수 있습니다.

개선 사항

1. 이 릴리스는 [버전 1.1 \(p. 923\)](#)에서 제공하는 모든 버그 수정, 기능, 개선 사항을 포함합니다.
2. 다음 PostgreSQL 확장 기능 업데이트:
 - `pg_hint_plan` 버전 1.2.2로 업데이트
 - `plv8` 버전 2.1.0으로 업데이트

3. 라이터 및 리더 노드 간의 트래픽 효율성을 향상합니다.
4. 연결 설정 성능을 향상합니다.
5. 메모리 부족 오류가 발생할 때 PostgreSQL 오류 로그에 제공되는 진단 데이터를 향상합니다.
6. Amazon RDS for PostgreSQL에서 PostgreSQL과 호환되는 Aurora로 스냅샷을 가져올 때 안정성과 성능을 향상하기 위해 여러 부분을 수정했습니다.
7. Aurora PostgreSQL 읽기 노드의 안정성과 성능을 향상하기 위해 여러 부분이 수정되었습니다.
8. 수정하지 않으면 유튜 인스턴스가 Aurora 스토리지 볼륨에서 불필요한 읽기 트래픽을 생성할 수 있는 버그를 수정합니다.
9. 삽입 중에 중복 시퀀스 값이 발생할 수 있는 버그를 수정합니다. RDS for PostgreSQL에서 Aurora PostgreSQL로 스냅샷을 마이그레이션할 때만 문제가 발생합니다. 이 수정을 적용하면 마이그레이션을 수행할 때 문제가 도입되지 않습니다. 이 릴리스 이전에 마이그레이션된 인스턴스에는 여전히 중복 키 오류가 발생할 수 있습니다.
10. 복제를 사용하여 Aurora PostgreSQL로 마이그레이션된 RDS for PostgreSQL 인스턴스가 GIST 인덱스 삽입/업데이트를 수행하는 동안 메모리가 부족하게 되거나 GIST 인덱스에 기타 문제가 발생할 수 있는 버그를 수정합니다.
11. Vacuum이 데이터베이스의 해당 pg_database.datfrozenxid 값을 업데이트하는 데 실패할 수 있는 버그를 수정합니다.
12. 새로운 MultiXact를 생성하는 동안 발생하는 충돌(경합하는 행 수준 잠금)로 인해 Aurora PostgreSQL이 엔진 다시 시작 후 동일한 관계에 처음 액세스할 때 무한정으로 종단될 수 있는 버그를 수정합니다.
13. ffdw를 호출하는 동안 PostgreSQL 백엔드를 종료하거나 취소할 수 없는 버그를 수정합니다.
14. Aurora 스토리지 데몬이 하나의 vCPU를 항상 완전히 이용하는 버그를 수정합니다. 이 문제는 유튜 상태 일 때 CPU 사용량이 25–50퍼센트에 달할 수 있는 r4.large와 같은 비교적 작은 인스턴스 클래스에서 특히 뚜렷하게 발생합니다.
15. Aurora PostgreSQL 라이터 노드가 거짓으로 장애 조치할 수 있는 버그를 수정합니다.
16. 드문 시나리오에서 Aurora PostgreSQL 읽기 노드가 다음을 보고할 수 있는 버그를 수정합니다.

"FATAL: buffer_io 잠금이 없음"
17. 기한 경과된 relcache 항목이 관계의 vacuum을 중단시키고 시스템을 ID 랩어라운드에 가깝게 푸시할 수 있는 버그를 수정합니다. 이 수정은 향후 마이너 버전에서 릴리스될 예정인 PostgreSQL 커뮤니티 패치의 포트입니다.
18. 관계를 확장하는 동안 발생하는 결함으로 인해 부분적으로 확장된 관계를 스캔하는 동안 Aurora에서 충돌이 발생할 수 있는 버그를 수정합니다.

버전 1.1

이 버전의 Aurora PostgreSQL은 PostgreSQL 9.6.6과 호환됩니다. 릴리스 9.6.6의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 9.6.6](#)을 참조하십시오.

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

새로운 기능

1. aurora_stat_utils 확장 기능을 도입했습니다. 이 확장 기능에는 다음 두 가지 함수가 포함됩니다.
 - 이벤트 모니터링 대기를 위한 aurora_wait_report() 함수
 - 레코드 쓰기 모니터링 로그를 위한 aurora_log_report()
2. 다음 확장 기능에 대한 지원을 추가했습니다.
 - orafce 3.6.1
 - pgRouting 2.4.2
 - postgresql-hll 2.10.2

- prefix 1.2.6

개선 사항

1. 이 릴리스는 [버전 1.0.11 \(p. 925\)](#)에서 제공하는 모든 버그 수정, 기능, 개선 사항을 포함합니다.
2. 다음 PostgreSQL 확장 기능에 대한 업데이트:
 - postgis 확장 기능이 2.3.4 버전으로 업데이트됨
 - geos 라이브러리가 3.6.2 버전으로 업데이트됨
 - pg_repack이 1.4.2 버전으로 업데이트됨
3. pg_statistic 관계에 대한 액세스가 활성화되었습니다.
4. 'effective_ioConcurrency' guc 파라미터는 Aurora 스토리지에 적용되지 않기 때문에 비활성화되었습니다.
5. 'hot_standby_feedback' guc 파라미터가 수정 불가로 변경되고, 값이 '1'로 설정되었습니다.
6. 정리 작업 중에 힙 페이지 읽기 성능이 향상되었습니다.
7. 읽기 노드에서 스냅샷 충돌 해결 성능이 향상되었습니다.
8. 읽기 노드에서 트랜잭션 스냅샷 획득 성능이 향상되었습니다.
9. GIN 메타 페이지 업데이트에 대한 쓰기 성능이 향상되었습니다.
10. 시작 중 버퍼 캐시 복구 성능이 향상되었습니다.
11. 준비된 트랜잭션 복구 중 시작 시 데이터베이스 엔진 충돌을 일으키는 버그를 수정합니다.
12. 준비된 트랜잭션 수가 많은 경우 읽기 노드를 시작할 수 없는 버그를 수정합니다.
13. 읽기 노드에서 다음 오류를 보고할 수 있는 버그를 수정합니다.

오류: 트랜잭션 6080077의 상태를 액세스할 수 없음

세부 정보: * "pg_subtrans/005C" 파일을 열 수 없음: 해당 파일이나 디렉터리가 없음.

14 RDS PostgreSQL에서 Aurora PostgreSQL로 복제할 때 다음과 같은 오류를 일으킬 수 있는 버그를 수정합니다.

FATAL: buffer_content 잠금이 없음

컨텍스트: 스토리지/TRUNCATE: base/13322/8058750에 대해 46E/F1330870에서 0블록 플래그 7로 xlog 다시 실행

15 RDS PostgreSQL에서 Aurora PostgreSQL로 복제할 때 multixact WAL 레코드를 다시 재생하는 동안 Aurora PostgreSQL을 중단시킬 수 있는 버그를 수정합니다.

16 RDS PostgreSQL에서 Aurora PostgreSQL로 스냅샷을 가져올 때 안정성에서 다양한 부분이 향상되었습니다.

버전 1.0

이 버전의 Aurora PostgreSQL은 PostgreSQL 9.6.3과 호환됩니다. 버전 9.6.3의 개선 사항에 대한 자세한 내용은 [PostgreSQL 릴리스 9.6.3](#)을 참조하십시오.

이 버전에는 다음과 같은 패치 버전이 포함되어 있습니다.

패치 버전

- [버전 1.0.11 \(p. 925\)](#)
- [버전 1.0.10 \(p. 925\)](#)
- [버전 1.0.9 \(p. 925\)](#)
- [버전 1.0.8 \(p. 925\)](#)

- [버전 1.0.7 \(p. 926\)](#)

버전 1.0.11

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

1. 올바르지 않은 결과가 발생할 수 있는 병렬 쿼리 실행 문제를 해결합니다.
2. Amazon RDS for PostgreSQL에서 복제하는 도중 Aurora 스토리지 볼륨을 사용할 수 없게 만드는 가시성 맵 처리 문제를 해결합니다.
3. pg-repack 확장을 수정합니다.
4. 새로운 노드를 유지하는 개선 사항을 적용합니다.
5. 엔진 충돌이 발생할 수 있는 문제를 해결합니다.

버전 1.0.10

이 업데이트에는 새 기능이 포함되어 있습니다. 이제 Amazon RDS PostgreSQL DB 인스턴스를 Aurora PostgreSQL로 복제할 수 있습니다. 자세한 내용은 [Amazon Aurora PostgreSQL를 사용한 복제 \(p. 807\)](#) 단원을 참조하십시오.

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

1. 캐시가 존재하고 파라미터 변경으로 인해 버퍼 캐시, 스토리지 형식 또는 크기 불일치가 발생할 때 오류 로깅을 추가합니다.
2. 방대한 페이지에 대한 호환 가능한 파라미터 값이 있는 경우 엔진 재부팅의 원인이 되는 문제를 해결합니다.
3. 읽기 노드에서 Write Ahead Log(WAL)의 재생 동안 여러 개의 truncate table 문을 개선합니다.
4. 정적 메모리 오버헤드를 줄여 메모리 부족 오류를 줄입니다.
5. GiST 인덱스를 통한 삽입 수행 도중 메모리 부족 오류가 발생할 수 있는 문제를 해결합니다.
6. RDS PostgreSQL로부터의 스냅샷 가져오기를 개선하여 초기화되지 않은 페이지에서 vacuum을 수행해야 하는 필요성을 제거합니다.
7. 준비된 트랜잭션을 유발하여 엔진 충돌 이후 이전 상태로 돌아가는 문제를 해결합니다.
8. 읽기 노드 기한 경과를 방지하는 개선 사항을 적용합니다.
9. 엔진 다시 시작을 통해 다운타임을 줄이는 개선 사항을 적용합니다.
10. 엔진 충돌이 발생할 수 있는 문제를 해결합니다.

버전 1.0.9

이 엔진 업데이트에서 초기화되지 않은 페이지를 포함하는 RDS PostgreSQL로부터 스냅샷을 가져올 때 Aurora 스토리지 볼륨을 사용할 수 없게 될 수 있는 문제를 해결합니다.

버전 1.0.8

이 엔진 업데이트에서 다음 개선 사항을 확인할 수 있습니다.

1. `shared_preload_libraries` 인스턴스 파라미터에 `pg_hint_plan`이 포함된 경우 엔진을 시작하지 못하도록 하는 문제를 해결합니다.
2. 병렬 스캔 도중 발생할 수 있는 "Attempt to fetch heap block XXX is beyond end of heap (YYYY blocks)" 오류를 해결합니다.
3. `vacuum`에 대한 읽기 미리 가져오기의 효율성을 개선합니다.

4. 소스 스냅샷에 포함되지 않는 pg_internal.init 파일이 있는 경우 오류가 발생할 수 있는 RDS PostgreSQL에서 스냅샷 가져오기 문제를 해결합니다.
5. "aurora wal replay process (PID XXX) was terminated by signal 11: Segmentation fault" 메시지와 함께 읽기 노드 중들이 발생할 수 있는 문제를 해결합니다. 이 문제는 리더가 캐시되지 않은 가시성 맵 페이지에 대한 가시성 맵 변경 사항을 적용할 때 발생합니다.

버전 1.0.7

이는 Amazon PostgreSQL과 호환되는 Aurora의 첫 번째 일반 공개용 릴리스입니다.

Amazon Aurora PostgreSQL의 확장 버전

주제

- [apg_plan_mgmt 확장 버전 2.0 \(p. 926\)](#)
- [apg_plan_mgmt 확장 버전 1.0.1 \(p. 927\)](#)

apg_plan_mgmt 확장 버전 2.0

쿼리 계획 관리와 함께 apg_plan_mgmt 확장을 사용합니다. apg_plan_mgmt 설치, 업그레이드 및 사용에 관한 자세한 내용은 [Aurora PostgreSQL용 쿼리 실행 계획 관리 \(p. 821\)](#) 단원을 참조하십시오.

버전 2.0의 apg_plan_mgmt 확장 변경 사항은 다음과 같습니다.

새로운 확장 기능

1. 이제 파라미터가 있는지 여부에 관계없이 SQL 함수 내의 모든 쿼리를 관리할 수 있습니다.
2. 이제 파라미터가 있는지 여부에 관계없이 PL/pgSQL 함수 내의 모든 쿼리를 관리할 수 있습니다.
3. 이제 파라미터가 있는지 여부에 관계없이 일반 계획에서 쿼리를 관리할 수 있습니다. 일반 계획과 사용자 지정 계획에 대한 자세한 내용은 [PostgreSQL 설명서](#)의 PREPARE 문을 참조하십시오.
4. 이제 쿼리 계획 관리를 사용하여 쿼리 계획에서 특정 유형의 집계 방법을 사용하도록 할 수 있습니다.

확장 개선 사항

1. 이제 max_worker_processes 파라미터 설정에 최대 8KB를 곱한 크기로 계획을 저장할 수 있습니다. 이전에는 최대 계획 크기가 8KB였습니다.
2. JDBC에서와 같이 명명되지 않은 준비된 문에 대한 버그를 수정했습니다.
3. 이전에는 shared_preload_libraries에 로드되지 않았을 때 CREATE EXTENSION apg_plan_mgmt를 수행하려고 시도하면 백엔드 연결이 끊어졌습니다. 이제 오류 메시지가 표시되고 연결이 끊어지지 않습니다.
4. apg_plan_mgmt.plans table에서 cardinality_error의 기본값은 NULL이지만 apg_plan_mgmt.evolve_plan_baselines 함수 중에 -1로 설정할 수 있습니다. NULL은 이제 일관되게 사용됩니다.
5. 이제 임시 테이블을 참조하는 쿼리에 대한 계획이 저장됩니다.
6. 기본 최대 계획 수는 1000에서 10000으로 증가합니다.
7. pgss 파라미터 대신 자동 계획 캡처 모드를 사용해야 하므로 해당 파라미터는 더 이상 사용되지 않습니다.
 - apg_plan_mgmt.pgss_min_calls
 - apg_plan_mgmt.pgss_min_mean_time_ms
 - apg_plan_mgmt.pgss_min_stddev_time_ms
 - apg_plan_mgmt.pgss_min_total_time_ms

apg_plan_mgmt 확장 버전 1.0.1

버전 1.0.1의 apg_plan_mgmt 확장 변경 사항은 다음과 같습니다.

새로운 확장 기능

1. 새로운 `update_plan_hash` 파라미터를 `validate_plans` 함수에 사용할 수 있습니다. 이 파라미터를 통해 정확히 복제할 수 없는 계획의 `plan_hash`가 업데이트됩니다. 또한 `update_plan_hash` 파라미터를 통해 SQL을 다시 작성하여 계획을 수정할 수 있습니다. 그런 다음 원본 SQL에 대해 좋은 계획을 `Approved` 계획으로 등록할 수 있습니다. 다음은 `update_plan_hash`의 용례입니다.

```
UPDATE apg_plan_mgmt.dba_plans SET plan_hash = new_plan_hash, plan_outline
= good_plan_outline
WHERE sql_hash = bad_plan_sql_hash AND plan_hash = bad_plan_plan_hash;
SELECT apg_plan_mgmt.validate_plans(bad_plan_sql_hash, bad_plan_plan_hash,
'update_plan_hash');
SELECT apg_plan_mgmt.reload();
```

2. 지정된 SQL 문에 EXPLAIN 문의 텍스트를 생성하는 새로운 `get_explain_stmt` 함수를 사용할 수 있습니다. 이 함수에는 `sql_hash`, `plan_hash` 및 `explain_options` 파라미터가 포함되어 있습니다.

`explain_options` 파라미터는 아래와 같이 유효한 EXPLAIN 옵션의 쉼표로 구분된 목록일 수 있습니다.

```
analyze,verbose,buffers,hashes,format json
```

`explain_options` 파라미터가 NULL 또는 빈 문자열인 경우 `get_explain_stmt` 함수는 간단한 EXPLAIN 문을 생성합니다.

워크로드 전체 또는 일부에 대해 EXPLAIN 스크립트를 생성하려면 \a, \t 및 \o 옵션을 사용해 출력을 파일로 리디렉션하십시오. 예를 들어 `total_time`에 의해 DESC 순서로 정렬된 PostgreSQL `pg_stat_statements` 보기를 사용해 최상위(top-K) 문에 대해 EXPLAIN 스크립트를 생성할 수 있습니다.

3. 수집 병렬 쿼리 연산자의 정확한 위치는 비용에 따라 결정되고 시간이 지남에 따라 약간 변경될 수 있습니다. 이러한 차이로 인해 전체 계획이 무효화되는 것을 방지하기 위해 이제 쿼리 계획 관리에서는 수집 연산자가 계획 트리의 여러 곳으로 이동한다 하더라도 동일한 `plan_hash`를 계산합니다.
4. pl/pgsql 함수 내에 있는 파라미터화되지 않은 설명문에 대한 지원이 추가되었습니다.
5. 동일 클러스터에 있는 여러 데이터베이스에 apg_plan_mgmt 확장이 설치됨과 동시에 두 개 이상의 데이터베이스가 동시에 액세스되고 있으면 오버헤드가 감소합니다. 또한 이 릴리스에서는 계획이 공유 메모리에 저장되지 않게 만들었던 이 영역의 버그를 수정했습니다.

확장 개선 사항

1. `evolve_plan_baselines` 함수 개선 사항.
 - a. `evolve_plan_baselines` 함수는 이제 계획 내 모든 노드에 대해 `cardinality_error` 지표를 계산합니다. 이 지표를 사용해 카디널리티 추정 오류가 많고 계획 품질이 더 의심스러운 모든 계획을 식별 할 수 있습니다. `cardinality_error` 값이 높은 장기 실행 설명문은 쿼리 튜닝 우선 순위가 높은 후보자입니다.
 - b. `evolve_plan_baselines`에서 생성되는 보고서에는 이제 `sql_hash`, `plan_hash` 및 `status` 계획이 포함됩니다.
 - c. 이제 `evolve_plan_baselines`에서 앞서 `Rejected`된 계획을 승인하도록 허용할 수 있습니다.
 - d. 이제 `evolve_plan_baselines`에 대한 `speedup_factor`의 의미는 기준 계획에 대해 항상 상대적입니다. 예를 들어 이제 1.1이라는 값은 기준 계획보다 10퍼센트 더 빠르다는 것을 뜻합니다. 0.9라는 값은 기준 계획보다 10퍼센트 더 느리다는 것을 뜻합니다. 이러한 비교는 총 시간이 아닌 실행 시간만을 사용해 이루어집니다.

- e. 이제 `evolve_plan_baselines` 함수는 새로운 방식으로 캐시를 위임합니다. 즉 기준 계획을 실행한 후 기준 계획을 한 차례 실행하고, 이어서 후보 계획을 한 번 수행하는 방식입니다. 전에는 `evolve_plan_baselines`이 후보 계획을 두 번 실행하였습니다. 이 접근 방식을 통해 실행 시간(특히 느린 후보자 계획에 대한 실행 시간)이 크게 늘어났습니다. 그러나 기준 계획에서 사용되지 않는 인덱스를 후보 계획에서 사용하는 경우에는 후보 계획을 두 차례 실행하는 것이 더 안정적입니다.
2. 쿼리 계획 관리는 시스템 테이블 또는 보기, 임시 테이블 또는 쿼리 계획 관리의 자체 테이블을 참조하는 계획을 더 이상 저장하지 않습니다.
3. 버그 수정 사항으로 저장 시 계획을 즉시 캐싱, 백엔드 종료의 원인이었던 버그 수정 등이 있습니다.

Amazon Aurora 모범 사례

이 주제에서는 데이터의 사용 또는 Amazon Aurora DB 클러스터로의 데이터 마이그레이션에 대한 일반적인 모범 사례 및 옵션에 대해 설명합니다.

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 사례 (p. 655) 단원을 참조하십시오.
Amazon Aurora PostgreSQL	Amazon Aurora PostgreSQL 모범 사례 (p. 878) 단원을 참조하십시오.

Note

Aurora에 대한 일반적인 권장 사항은 [Amazon Aurora 권장 사항 사용 \(p. 408\)](#) 단원을 참조하십시오.

주제

- [Amazon Aurora 기본 운영 지침 \(p. 929\)](#)
- [DB 인스턴스 RAM 권장 사항 \(p. 929\)](#)
- [Amazon Aurora 모니터링 \(p. 930\)](#)
- [DB 파라미터 그룹 및 DB 클러스터 파라미터 그룹 작업 \(p. 930\)](#)
- [Amazon Aurora 모범 사례 프레젠테이션 동영상 \(p. 930\)](#)

Amazon Aurora 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 시스템 성능과 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 이후에 DB 인스턴스의 기본 IP 주소가 변경될 수 있기 때문에 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.
- 사용 사례에 대한 프로세스가 얼마나 오래 걸리는지 이해하고 DB 클러스터에 액세스하는 애플리케이션이 장애 조치 이후에 새 DB 클러스터에 자동으로 연결할 수 있는지 확인하려면 DB 클러스터에 대한 장애 조치를 테스트합니다.

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** – 이렇게 하여 5분 간격으로 보고되는 클러스터 볼륨에서 읽기 I/O 작업의 평균 수를 측정합니다. **VolumeReadIOPS**의 값은 작고 안정적이어야 합니다. 읽기 IO가 급격히 증가하거나 평소보다 높으면 DB 클러스터의 DB 인스턴스를 조사하여 IO 증가를 초래한 DB 인스턴스를 확인해야 합니다.
- **BufferCacheHitRatio** – 이 지표는 DB 클러스터에서 DB 인스턴스의 버퍼 캐시가 제공하는 요청 백분율을 측정합니다. 이 지표는 메모리에서 제공되는 데이터의 양에 대한 통찰력을 제공합니다. 적중률이 낮으면 이 DB 인스턴스에 대한 쿼리가 디스크보다 자주 발생한다는 것을 나타내는 좋은 지표입니다. 이 경우 워크로드를 조사하여 이 동작의 원인이 되는 쿼리를 확인해야 합니다.

워크로드를 조사한 후에 더 많은 메모리가 필요하다면 DB 인스턴스 클래스를 RAM이 더 큰 클래스로 확장하면 유용할 수 있습니다. 이렇게 한 후 위의 지표를 조사하고 필요에 따라 계속 확장할 수 있습니다. DB 클러스터 모니터링에 대한 자세한 정보는 [Amazon Aurora DB 클러스터 지표 모니터링 \(p. 346\)](#) 단원을 참조하십시오.

Amazon Aurora 모니터링

Amazon Aurora은 모니터링을 통해 Aurora DB 클러스터의 상태와 성능을 판별할 수 있는 다양한 Amazon CloudWatch 지표를 제공합니다. Amazon RDS 관리 콘솔, AWS CLI 및 CloudWatch API 등 다양한 도구를 사용하여 Aurora 지표를 볼 수 있습니다. 자세한 정보는 [Amazon Aurora DB 클러스터 모니터링 \(p. 327\)](#) 단원을 참조하십시오.

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 \(p. 268\)](#) 단원을 참조하십시오.

Amazon Aurora 모범 사례 프레젠테이션 동영상

2016 AWS Summit 컨퍼런스(시카고)에는 가용성이 높은 안전한 Amazon Aurora DB 클러스터를 생성 및 구성하는 모범 사례에 대한 프레젠테이션이 포함되었습니다. 프레젠테이션 비디오는 [여기](#)에서 볼 수 있습니다.

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 다양한 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 \(p. 211\)](#), [Amazon Aurora MySQL 성능 개선 사항 \(p. 463\)](#), [Amazon Aurora PostgreSQL 관리 \(p. 802\)](#), [Amazon RDS 성능 개선 도우미 사용 \(p. 365\)](#)와 같은 성능 관련 문서를 참조하십시오.

이 주제에서 제공하는 정보는 조직이 코드를 작성하고 스키마를 설계하며 MySQL 및 PostgreSQL 오픈 소스 데이터베이스 엔진을 지원하는 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 고객님의 애플리케이션 유형에 대해 Aurora 모범 사례 또는 사례 연구가 있는지 확인하십시오.

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. Aurora는 MySQL 5.6 및 MySQL 5.7뿐 아니라 PostgreSQL 9.6 및 PostgreSQL 10.4와도 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 서비스 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 자연 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 들어 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 방법](#)에 설명된 워크로드 통합입니다. 또 다른 한 가지는 Amazon Aurora 사용 설명서의 [Aurora 복제본에 Amazon Aurora Auto Scaling 사용 \(p. 253\)](#)에 설명된 수요 기반 확장입니다. 그밖에 성능 향상 또는 단순화된 데이터베이스 연산이 있습니다.

2. 고객님의 워크로드 특성에 대한 이해

고객님이 의도한 사용 사례의 컨텍스트에서 Aurora를 평가하십시오. Aurora는 OLTP(온라인 트랜잭션 처리) 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
- 자연 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. 또한 워크로드에 따라 이러한 클러스터는 초당 200,000건의 INSERT, UPDATE 및 DELETE 문을 처리할 수 있습니다. Aurora는 행 스토어 데이터베이스로서 대용량, 고처리량 및 고도로 병렬화된 OLTP 워크로드에 최적화되어 있습니다.

또한 Aurora는 OLTP 워크로드를 처리하는 동일 클러스터에서 보고 쿼리를 실행할 수 있습니다. Aurora는 최대 15개의 [복제본 \(p. 55\)](#)을 지원하는데, 각 복제본은 평균 10–20밀리초의 기본 인스턴스 내에 있습니다. 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 이용해 Aurora의 기능, 기타 AWS 서비스, AWS Management 콘솔 및 AWS CLI를 숙지하십시오. 또한 이러한 것들이 고객님이 개념 증명에 사용할 계획인 기타 도구와 어떻게 연동되는지 확인하십시오.

3. AWS Management 콘솔 또는 AWS CLI를 이용한 실습

다음 단계로 AWS Management 콘솔 또는 AWS CLI를 이용한 실습을 통해 이러한 도구와 Aurora를 숙지하십시오.

AWS Management 콘솔을 이용한 실습

Aurora 데이터베이스 클러스터를 이용한 다음과 같은 초기 활동은 주로 고객님이 AWS Management 콘솔 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\(p. 55\)](#) 단원을 참조하십시오. AWS 인스턴스 클래스를 변경하여 DB 인스턴스의 규모를 확장 또는 축소하십시오. 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 콘솔을 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 CLI를 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 위

한 AWS CloudFormation 동작은 새 인스턴스를 생성하고 데이터를 포함한 현재 인스턴스를 삭제하는 것입니다. 이 동작에 대한 자세한 내용은 AWS CloudFormation 사용 설명서의 [스택 리소스의 업데이트 동작](#) 단원을 참조하십시오.

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 \(p. 95\)](#) 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.

- Amazon Aurora를 선택하십시오.
- MySQL 5.6 호환성. 이러한 데이터베이스 엔진 및 버전의 조합은 다른 Aurora 기능과 가장 폭넓게 호환됩니다.
- quick create(빠른 생성)를 비활성화합니다. 개념 증명의 경우 나중에 동일한 또는 약간 다른 클러스터를 생성할 수 있도록 고객님이 선택하는 모든 설정을 속지하는 것이 좋습니다.
- 리전. Global(글로벌) 설정은 특정 고가용성 시나리오를 위한 것입니다. 초기에 기능 및 성능 실험을 마친 후 나중에 다시 시도할 수 있습니다.
- 라이터 한 개, 리더 여러 개. 이것은 가장 널리 사용되는 범용 클러스터입니다. 이 설정은 클러스터의 수명 주기 동안 지속됩니다. 따라서 나중에 서비스 또는 병렬 쿼리와 같은 다른 종류의 클러스터에 대한 실험을 수행하는 경우 다른 클러스터를 생성하고 각 클러스터에서 나온 결과를 상호 비교 및 대조하십시오.
- 개발/테스트 템플릿을 선택합니다. 이 선택은 개념 증명 활동에 중요하지는 않습니다.
- Instance Size(인스턴스 크기)에서 Memory Optimized(최적화된 메모리)와 xlarge 인스턴스 클래스 중 하나를 선택합니다. 나중에 인스턴스 클래스를 확장 또는 축소 조정할 수 있습니다.
- 다중 AZ 배포에서 다른 영역에 복제본 생성을 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 “마스터” 또는 “라이터”로 참조해서는 안 됩니다. 왜냐하면 서로 다른 DB 인스턴스는 이러한 역할을 필요에 따라 수임하기 때문입니다. clustername-az-serialnumber과 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 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 이

동하는 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 실행하려면 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 클러스터 연결 \(p. 162\)](#) 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 고려하십시오. 자세한 내용은 [AWS Schema Conversion Tool 사용 설명서](#)를 참조하십시오. 마이그레이션 및 포트 활동에 대한 일반적인 세부 정보는 AWS 백서인 [Aurora 마이그레이션 핸드북](#)을 참조하십시오.

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 \(p. 472\)](#) 또는 [데이터를 PostgreSQL과 호환되는 Amazon Aurora로 마이그레이션 \(p. 781\)](#) 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 설명서](#)를 참조하십시오.

마이그레이션 및 포트 활동에 대한 자세한 내용은 AWS 백서인 [Aurora 마이그레이션 핸드북](#)을 참조하십시오.

7. SQL 코드 포트

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 MySQL 또는 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 오픈소스 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 \(p. 936\)](#) 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.

マイグ레이션 및 포트 활동에 대한 자세한 내용은 AWS 백서인 [Aurora 마이그레이션 핸드북](#)을 참조하십시오.

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 소개](#)를 참조하십시오.

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 캐시 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 \(p. 170\)](#) 단원을 참조하십시오. Aurora 클러스터에 적용되거나 적용되지 않는 구성 설정에 대해서는 데이터베이스 엔진에 따라 [Aurora MySQL 파라미터 \(p. 665\)](#) 또는 [Amazon Aurora PostgreSQL 파라미터 \(p. 896\)](#)를 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.

쓰기 연산에 대해 클러스터 엔드포인트를 사용함으로써 읽기-쓰기 기능이 있는 클러스터에 있는 DB 인스턴스에 항상 연결할 수 있습니다. 기본적으로 Aurora 클러스터에 있는 단 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 종명 중에는 대개 필요하지 않습니다.

Aurora 복제본은 복제본 지연 시간이 대개 10-20밀리초라 하더라도 이 지연 시간의 적용을 받습니다. 복제지연 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 DML 문과 같은 쓰기 연산을 클라이언트 애플리케이션 코드 내 함수에 뮤습니다. 이렇게 하면 다양한 종류의 연산에서 특정 연결을 사용하게 할 수 있습니다.
- 쿼리 연산에 대해 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 열니다. 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 클러스터 연결 \(p. 162\)](#) 단원을 참조하십시오. 이 주제에 관해 자세히 알아보려면 [Aurora MySQL 데이터베이스 관리자용 핸드북 – 연결 관리](#)를 참조하십시오.

10. 워크로드 실행

スキ마, 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 의사결정을 내리는 것이 좋습니다. 가능한 경우 스키마, 쿼리 패턴 및 사용량에 근거하여 측정값을 수집하십시오.

가능한 한 애플리케이션이 실행될 실제 조건을 복제하십시오. 예를 들어 사용자는 일반적으로 Amazon EC2 클러스터와 동일한 AWS 리전과 동일한 Virtual Private Cloud(VPC)에 있는 Aurora 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 영역](#) 단원을 참조하십시오. 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 것으로 더 큰 비율의 샘플 데이터를 이용해 테스트를 실행 할 수 있습니다.

이 단계에서는 메모리 제한 및 연결 제한과 같은 구성 설정을 변경하면 결과가 어떻게 되는지 확인할 수 있습니다. [8. 구성 설정 지정 \(p. 936\)](#)에서 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 \(p. 271\)](#) 및 [DB 클러스터 스냅샷에서 복원 \(p. 273\)](#) 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.

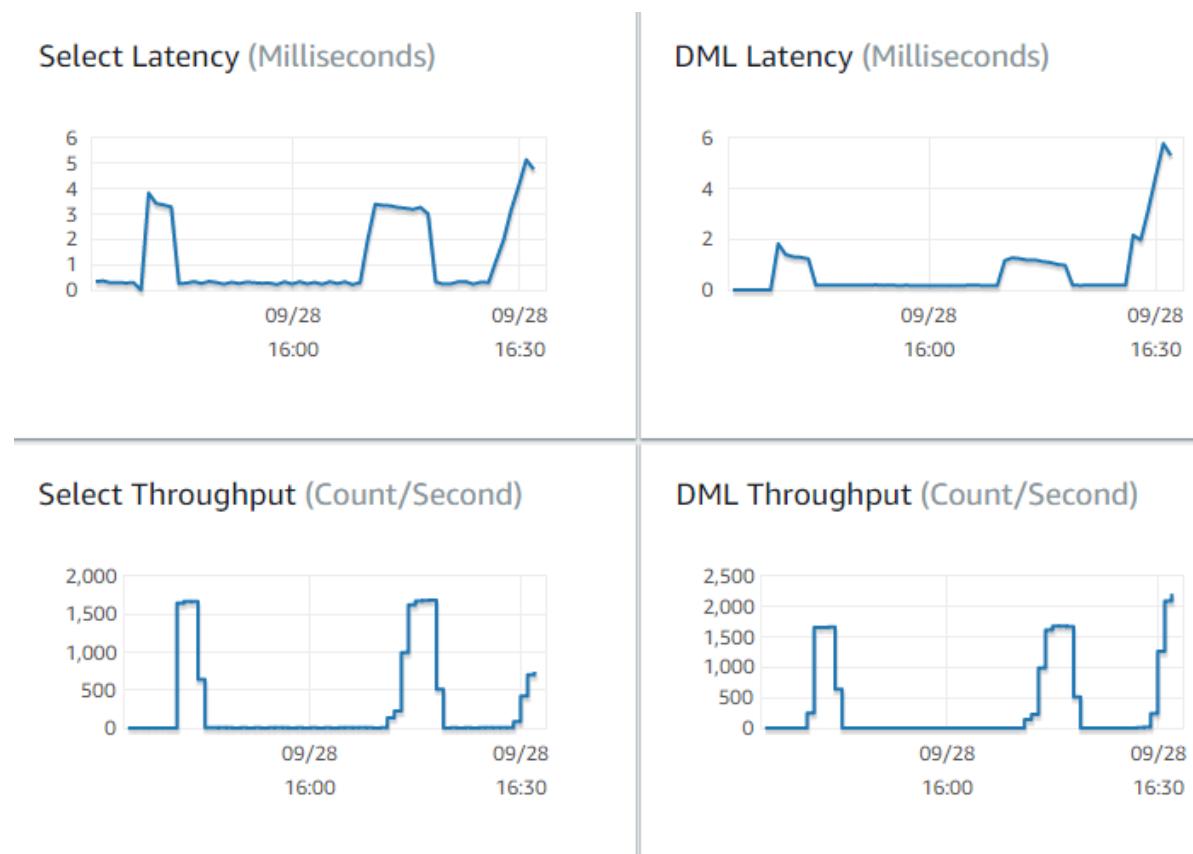
모니터링 탭에서 더 많은 사항을 선택할 수 있게 하려면 클러스터 설정에서 기본 모니터링 및 성능 개선 도우미를 활성화하십시오. 클러스터 설정 시 이를 선택하지 않았다면 이 선택 항목을 활성화할 수도 있습니다.

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 어떻게 분할되는지 확인할 수도 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.

이를 위해서는 아래에 설명된 대로 [RDS 콘솔](#)에서 해당 Aurora 클러스터의 모니터링 탭을 살펴보십시오.



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 동시에 쿼리, 더 큰 데이터에 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

기준 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 측정하여 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 복구되도록 하려면 의도적으로 문제를 도입해야 할 수 있습니다.

Tip

개념 증명의 경우 동일한 AWS 인스턴스 클래스로 Aurora 클러스터에서 모든 DB 인스턴스를 설정하십시오. 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 사용 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 테스트 \(p. 522\)](#) 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. 또한 DB 인스턴스가 더 많으면 읽기 집약적인 애플리케이션의 확장성이 향상됩니다. Aurora는 `SELECT`를 위한 다중 연결을 읽기 전용 Aurora 복제본 간에 분산할 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 작

업을 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 모방하십시오.

RTO(복원 시간 목표), RPO(복원 시점 목표) 및 지리적 종복성에 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 핸드북](#)을 참조하십시오.

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 알아보십시오.

이 시점에 이르면 Aurora를 이용한 벤치마크 테스트 실행의 절차 및 모범 사례를 이해해야 합니다. 성능 투닝을 추가로 수행할 필요가 있다는 것을 알게 될 수도 있습니다. 자세한 내용은 [Aurora DB 클러스터의 성능 및 확장 관리](#) (p. 211), [Amazon Aurora MySQL 성능 개선 사항](#) (p. 463), [Amazon Aurora PostgreSQL 관리](#) (p. 802) 및 [Amazon RDS 성능 개선 도우미 사용](#) (p. 365) 단원을 참조하십시오. 추가 투닝을 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 지역 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 관리](#)를 참조하십시오.

개념 증명을 마친 후 사용 사례가 Aurora에 적합하지 않다고 판단되면 다른 AWS 서비스를 고려하십시오.

- 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 설명서에 설명된 [Amazon S3에서 가져오기](#)
 - Amazon Aurora 사용 설명서에 설명된 [Amazon S3로 내보내기](#)
 - 그밖에 널리 사용되는 다수의 ETL 도구

Amazon Aurora의 보안

AWS에서 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 가장 보안에 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 귀하의 공동 책임입니다. [책임 분담 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- **클라우드의 보안** – AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호합니다. AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 타사 감사원은 정기적으로 [AWS 규제 준수 프로그램](#)의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon Aurora(Aurora)에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 [규정 준수 프로그램 제공 범위 내 AWS 서비스](#)를 참조하십시오.
- **클라우드 내 보안** – 귀하의 책임은 귀하가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 데이터의 민감도, 조직의 요구 사항 및 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Amazon Aurora 사용 시 책임 분담 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 항목에서는 보안 및 규정 준수 목표를 충족하도록 Amazon Aurora를 구성하는 방법을 보여줍니다. 또한 Amazon Aurora 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 제품을 사용하는 방법을 배우게 됩니다.

DB 클러스터에서 Amazon Aurora 리소스 및 데이터베이스에 대한 액세스를 관리할 수 있습니다. 액세스에 사용하는 방법은 사용자가 Amazon Aurora를 사용하여 수행해야 하는 작업 유형에 따라 다릅니다.

- 네트워크 액세스 제어를 최대한 강화할 목적으로 Amazon VPC 서비스에 따라 DB 클러스터를 가상 프라이빗 클라우드(VPC)에서 실행합니다. DB 클러스터를 VPC에서 생성하는 방법에 대한 자세한 내용은 [Amazon Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#) 단원을 참조하십시오.
- Amazon Aurora 리소스를 관리할 수 있는 사용자를 결정하는 권한을 배정하려면 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 사용자를 설정하는 방법에 대한 자세한 내용은 [IAM 사용자 생성 \(p. 69\)](#) 단원을 참조하십시오.

- 보안 그룹을 사용하여 어떤 IP 주소 또는 Amazon EC2 인스턴스가 DB 클러스터에 있는 데이터베이스에 연결할 수 있는지 제어합니다. DB 클러스터를 처음 생성하면, DB 인스턴스 방화벽에서 연결된 보안 그룹에서 지정한 규칙 이외의 데이터베이스 액세스를 차단합니다.
- Aurora MySQL 또는 Aurora PostgreSQL을 실행하는 DB 클러스터와 함께 SSL(Secure Socket Layer) 또는 TLS(전송 계층 보안) 연결을 사용합니다. DB 클러스터에서 SSL/TLS를 사용하는 방법에 대한 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#) 단원을 참조하십시오.
- Amazon Aurora 암호화를 사용해 DB 클러스터와 저장된 스냅샷을 보호합니다. Amazon Aurora 암호화는 업계 표준 AES-256 암호화 알고리즘을 사용해 DB 클러스터의 호스팅 서버에 저장된 데이터를 암호화합니다. 자세한 내용은 [Amazon Aurora 리소스 암호화 \(p. 945\)](#) 단원을 참조하십시오.
- 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 \(p. 466\)](#) 단원을 참조하십시오. Aurora PostgreSQL 사용 시 보안에 대한 자세한 내용은 [Amazon Aurora PostgreSQL를 사용한 보안 \(p. 773\)](#) 단원을 참조하십시오.

Aurora은 관리형 데이터베이스 서비스 Amazon Relational Database Service(Amazon RDS)의 일부입니다. Amazon RDS는 클라우드에서 관계형 데이터베이스를 보다 쉽게 설정, 작동 및 확장할 수 있게 해주는 웹 서비스입니다. Amazon RDS에 익숙하지 않은 경우 [Amazon RDS 사용 설명서](#)를 참조하십시오.

Aurora에는 고성능 스토리지 하위시스템이 포함됩니다. MySQL 호환 데이터베이스 엔진과 PostgreSQL 호환 데이터베이스 엔진은 고속 분산 스토리지의 이점에 따라 맞춤 설계되었습니다. 또한 Aurora은 데이터베이

스 구성 및 관리의 가장 어려운 측면 중 하나인 데이터베이스 클러스터링 및 복제를 자동화하고 표준화합니다.

Amazon RDS와 Aurora에서 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 정보는 [Amazon RDS에 대한 AWS Command Line Interface 레퍼런스](#)를 참조하십시오.

Note

사용 사례에 따라 보안을 구성하기만 하면 됩니다. 프로세스를 Amazon Aurora에서 관리하는 경우에는 보안 액세스를 구성할 필요 없습니다. 이러한 프로세스로는 백업 파일 생성, 마스터와 일기 전용 복제본 간 데이터 복제 등이 있습니다.

Amazon Aurora 리소스를 비롯해 DB 클러스터의 데이터베이스에 대한 액세스 관리는 아래 주제를 참조하십시오.

주제

- [Amazon Aurora의 데이터 보호 \(p. 944\)](#)
- [Amazon Aurora의 Identity and Access Management\(IAM\) \(p. 960\)](#)
- [Kerberos 인증 \(p. 994\)](#)
- [Amazon Aurora의 로깅 및 모니터링 \(p. 995\)](#)
- [Amazon Aurora 규정 준수 확인 \(p. 996\)](#)
- [Amazon Aurora의 복원성 \(p. 997\)](#)
- [Amazon Aurora 내 인프라 보안 \(p. 998\)](#)
- [Amazon Aurora 보안 모범 사례 \(p. 998\)](#)
- [보안 그룹을 통한 액세스 제어 \(p. 999\)](#)
- [마스터 사용자 계정 권한 \(p. 1001\)](#)
- [Amazon Aurora에 서비스 연결 역할 사용 \(p. 1001\)](#)
- [Amazon Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#)

Amazon Aurora의 데이터 보호

Amazon Aurora는 AWS [공동 책임 모델](#)을 준수하며, 여기에는 데이터 보호 관련 규정 및 지침이 포함됩니다. AWS는 모든 AWS 서비스를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. AWS는 고객 콘텐츠 및 개인 데이터의 처리를 위한 보안 구성을 제어 등 이 인프라에서 호스팅되는 데이터에 대한 제어권을 유지합니다. 데이터 컨트롤러 또는 데이터 처리자의 역할을 담당하는 AWS 고객과 APN 파트너는 AWS Cloud에 올린 모든 개인 데이터에 대해 책임을 집니다.

데이터를 보호하려면 AWS 계정 자격 증명을 보호하고 AWS Identity and Access Management(IAM)를 사용해 보안 주체를 설정하는 것이 좋습니다. 이 말은 각 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여된다는 것을 의미합니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정마다 멀티 팩터 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다.
- AWS CloudTrail로 API 및 사용자 활동 로깅을 설정합니다.
- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 컨트롤과 함께 사용합니다.
- Amazon S3에 저장된 개인 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- Aurora PostgreSQL에서는 데이터베이스 활동 스트림을 사용해 데이터베이스 활동을 모니터링하고 감사하여 데이터베이스에 대한 보호 수단을 제공할 뿐만 아니라 규정 준수 및 규제 요건을 충족합니다.

이름 필드와 같은 자유 형식 필드에 고객 계정 번호와 같은 중요 식별 정보를 절대 입력하지 마십시오. 여기에는 Amazon Aurora 또는 기타 AWS 서비스에서 콘솔, API, AWS CLI 또는 AWS SDK를 사용하여 작업하는 경우가 포함됩니다. Amazon Aurora 또는 기타 서비스에 입력하는 모든 데이터는 진단 로그에 포함하기 위해 선택될 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명 정보를 URL에 포함시키지 마십시오.

데이터 보호에 대한 자세한 내용은 AWS 보안 블로그에서 [AWS 공동 책임 모델 및 GDPR](#) 블로그 게시물을 참조하십시오.

주제

- [암호화를 사용하여 데이터 보호 \(p. 945\)](#)
- [인터넷워크 트래픽 개인 정보 \(p. 960\)](#)

암호화를 사용하여 데이터 보호

데이터베이스 리소스에 대한 암호화를 활성화할 수 있습니다. 또한 DB 클러스터에 대한 연결도 암호화가 가능합니다.

주제

- [Amazon Aurora 리소스 암호화 \(p. 945\)](#)
- [키 관리 \(p. 947\)](#)
- [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#)
- [SSL/TLS 인증서 교체 \(p. 949\)](#)

Amazon Aurora 리소스 암호화

Amazon Aurora DB 클러스터에서 암호화 옵션을 활성화하여 Amazon Aurora DB 클러스터와 저장 중인 스냅샷을 암호화할 수 있습니다. 유형 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 \(p. 945\)](#)
- [DB 클러스터에 대해 Amazon Aurora 암호화 활성화 \(p. 946\)](#)
- [Amazon Aurora 암호화 가능성 \(p. 946\)](#)
- [Amazon Aurora 암호화된 DB 클러스터의 제한 \(p. 946\)](#)

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 휴면 상태의 데이터 암호화를 위한 규정 준수 요구 사항을 만족하기 위해 Amazon Aurora 암호화를 사용할 수 있습니다.

Amazon Aurora 리소스의 암호화 및 암호화 해제에 사용되는 키를 관리하려면 [AWS Key Management Service\(AWS KMS\)](#)를 사용합니다. AWS KMS는 클라우드에 맞게 확장된 키 관리 시스템을 제공하기 위해 안전하고 가용성이 높은 하드웨어 및 소프트웨어를 결합합니다. AWS KMS를 사용하면 암호화 키를 생성하고 이 키를 사용할 수 있는 방법을 제어하는 정책을 정의할 수 있습니다. AWS KMS는 CloudTrail을 지원하도록 키가 적절하게 사용되고 있는지 확인하기 위해 키 사용을 감사할 수 있습니다. AWS KMS 키는 Amazon Aurora를 비롯해 Amazon S3, Amazon EBS, Amazon Redshift 등 지원되는 AWS 서비스에서 사용할 수 있습니다. AWS KMS를 지원하는 서비스 목록을 보려면 AWS Key Management Service 개발자 안내서에서 [지원되는 서비스](#) 단원을 참조하십시오.

Amazon Aurora 암호화된 DB 클러스터의 경우 모든 로그, 백업 및 스냅샷이 암호화됩니다. Amazon Aurora 암호화된 클러스터의 읽기 전용 복제본을 암호화할 수도 있습니다. 읽기 전용 복제본의 암호화는 AWS 리전의 KMS 마스터 키를 통해 보호를 받습니다.

DB 클러스터에 대해 Amazon Aurora 암호화 활성화

새로운 DB 클러스터에 대해 암호화를 활성화하려면 콘솔에서 암호화 활성화를 선택합니다. DB 클러스터 생성에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

`rds-create-db-cluster` AWS CLI 명령을 사용하여 암호화된 DB 클러스터를 생성할 경우 --storage-encrypted 파라미터를 true로 설정하십시오. [CreateDBCluster API](#) 작업을 사용할 경우 StorageEncrypted 파라미터를 true로 설정하십시오.

암호화된 DB 클러스터를 생성할 때 암호화 키에 대한 AWS KMS 키 식별자를 제공할 수도 있습니다. AWS KMS 키 식별자를 지정하지 않으면 Amazon Aurora는 새 DB 클러스터에 대해 기본 암호화 키를 사용합니다. AWS KMS는 AWS 계정의 Amazon Aurora에 대한 기본 암호화 키를 생성합니다. AWS 계정에는 AWS 리전마다 다른 기본 암호화 키가 있습니다.

암호화된 DB 클러스터를 생성한 후에는 해당 DB 클러스터에서 사용된 암호화 키의 유형을 변경할 수 없습니다. 따라서 암호화된 DB 클러스터를 생성하기 전에 암호화 키 요구 사항을 결정해야 합니다.

AWS CLI `create-db-cluster` 명령을 사용하여 암호화된 DB 클러스터를 생성할 경우 DB 클러스터의 KMS 키에 대한 Amazon 리소스 이름(ARN)으로 --kms-key-id 파라미터를 설정하십시오. RDS API `CreateDBCluster` 작업을 사용할 경우 DB 인스턴스의 KMS 키에 대한 ARN으로 `KmsKeyId` 파라미터를 설정하십시오.

다른 계정에 있는 키의 ARN을 사용하여 DB 클러스터를 암호화할 수 있습니다. 혹은 새 DB 클러스터를 암호화하는데 사용된 KMS 암호화 키를 소유한 동일한 AWS 계정으로 DB 클러스터를 생성할 수도 있습니다. 이 경우에는 전달하는 KMS 키 ID가 키의 ARN이 아닌 KMS 키 별칭일 수도 있습니다.

Important

경우에 따라 Amazon Aurora가 DB 클러스터 암호화 키에 대한 액세스 권한을 잃을 수도 있습니다. 예를 들어 키에 대한 RDS 액세스가 취소되면 Aurora는 액세스 권한을 잃습니다. 이때는 암호화된 DB 클러스터가 터미널 상태로 전환되기 때문에 백업 파일에서만 DB 클러스터를 복원할 수 있습니다. 데이터베이스에서 암호화된 데이터가 손실되지 않도록 보호하려면 암호화된 DB 클러스터에 대해 항상 백업을 활성화하는 것이 좋습니다.

Amazon Aurora 암호화 가능성

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.

Amazon Aurora 암호화된 DB 클러스터의 제한

Amazon Aurora 암호화된 DB 클러스터에는 다음과 같은 제한이 있습니다.

- 암호화된 DB 클러스터는 암호화를 비활성화하도록 수정할 수 없습니다.

- 암호화되지 않은 DB 클러스터를 암호화된 DB 클러스터로 변환할 수 없습니다. 하지만 암호화되지 않은 Aurora DB 클러스터 스냅샷을 암호화된 Aurora DB 클러스터로 복원할 수 있습니다. 암호화되지 않은 DB 클러스터 스냅샷에서 복원할 때 KMS 암호화 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 리전의 KMS 키 식별자를 지정해야 합니다. 왜냐하면 KMS 암호화 키는 이 키를 생성한 AWS 리전에 고유한 것이기 때문입니다.

소스 스냅샷은 복사 프로세스 전체에서 암호화를 유지합니다. AWS Key Management Service(KMS)는 봉투 암호화를 이용해 복사 프로세스가 진행되는 동안 데이터를 보호합니다. 봉투 암호화에 대한 자세한 내용은 [봉투 암호화](#)를 참조하십시오.

키 관리

AWS KMS 콘솔에서 [AWS Key Management Service\(AWS KMS\)](#)를 사용하여 Amazon Aurora 암호화된 DB 클러스터에 사용되는 키를 관리할 수 있습니다. 키를 완벽하게 제어하기 위해서는 고객 관리 키를 생성해야 합니다.

AWS KMS가 프로비저닝하는 기본 키는 삭제, 취소 또는 교체가 불가능합니다. 스냅샷을 공유한 AWS 계정의 기본 AWS KMS 암호화 키를 사용하여 암호화된 스냅샷은 공유할 수 없습니다.

[AWS CloudTrail](#)을 사용하여 고객 관리 키로 수행한 모든 작업의 감사 로그를 볼 수 있습니다.

Important

Aurora에서 Aurora가 액세스할 수 없는 키로 암호화된 DB 클러스터가 발견될 경우, Aurora는 DB 클러스터를 터미널 상태로 전환합니다. 이러한 상태에서는 DB 클러스터를 더 이상 사용하지 못하기 때문에 데이터베이스의 현재 상태를 복구할 수 없습니다. DB 클러스터를 복원하려면 Aurora의 암호화 키에 대한 액세스 권한을 다시 활성화한 후 백업 파일에서 DB 클러스터를 복원해야 합니다.

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. 각 DB 엔진에는 SSL/TLS를 구현하기 위한 고유한 프로세스가 있습니다. DB 클러스터에 대해 SSL/TLS를 구현하는 방법을 알아보려면 DB 엔진에 해당하는 아래 링크를 사용하십시오.

- [Amazon Aurora MySQL를 사용한 보안 \(p. 466\)](#)
- [Amazon Aurora PostgreSQL를 사용한 보안 \(p. 773\)](#)

Important

인증서 교체에 대한 자세한 내용은 [SSL/TLS 인증서 교체 \(p. 949\)](#) 단원을 참조하십시오.

Note

모든 인증서는 SSL/TLS 연결을 통한 다운로드에만 사용 가능합니다.

모든 AWS 리전에서 작동하는 루트 인증서는 다음 위치 중 한 곳에서 다운로드할 수 있습니다.

- <https://s3.amazonaws.com/rds-downloads/rds-ca-2019-root.pem>(CA-2019)
- <https://s3.amazonaws.com/rds-downloads/rds-ca-2015-root.pem>(CA-2015)

이 루트 인증서는 신뢰할 수 있는 루트 개체이므로 대부분의 경우에 작동하지만 애플리케이션에서 인증서 체인을 수락하지 않는 경우 실패할 수 있습니다. 애플리케이션에서 인증서 체인을 허용하지 않는 경우 이 단원의 뒤에 나오는 중간 인증서 목록에서 AWS 리전별 인증서를 다운로드하십시오.

중간 인증서와 루트 인증서를 모두 포함하는 인증서 번들은 <https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem>에서 다운로드할 수 있습니다.

애플리케이션이 Microsoft Windows에서 실행되어 PKCS7 파일이 필요한 경우에는 PKCS7 인증서 번들을 다운로드할 수 있습니다. 이 번들에는 중간 인증서와 루트 인증서가 포함되어 있으며, <https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.p7b>에서 다운로드가 가능합니다.

Note

Amazon RDS Proxy 및 Aurora Serverless 사용에서는 AWS Certificate Manager(ACM)의 인증서를 합니다. RDS Proxy를 사용하는 경우 Amazon RDS 인증서를 다운로드하거나 RDS Proxy 연결을 사용하는 애플리케이션을 업데이트할 필요가 없습니다. RDS Proxy에서 TLS/SSL을 사용하는 방법에 대한 자세한 내용은 [RDS Proxy에서 TLS/SSL 사용 \(p. 215\)](#) 단원을 참조하십시오.

Aurora Serverless을 사용하는 경우 Amazon RDS 인증서를 다운로드할 필요가 없습니다. Aurora Serverless에서 TLS/SSL을 사용하는 방법에 대한 자세한 내용은 [Aurora Serverless에서 TLS/SSL 사용 \(p. 114\)](#) 단원을 참조하십시오.

중간 인증서

자신의 AWS 리전에 연결하려면 중간 인증서를 사용해야 할 수도 있습니다. 예를 들어 SSL/TLS를 사용하여 AWS GovCloud(미국 서부) 리전에 연결하려면 중간 인증서를 사용해야 합니다. 특정 AWS 리전에 대한 중간 인증서가 필요한 경우 다음 테이블에서 인증서를 다운로드하십시오.

AWS 리전	CA-2019	CA-2015
아시아 태평양(홍콩)	rds-ca-2019-ap-east-1.pem	사용 가능한 인증서가 없음
아시아 태평양(뭄바이)	rds-ca-2019-ap-south-1.pem	rds-ca-2015-ap-south-1.pem
아시아 태평양(도쿄)	rds-ca-2019-ap-northeast-1.pem	rds-ca-2015-ap-northeast-1.pem
아시아 태평양(서울)	rds-ca-2019-ap-northeast-2.pem	rds-ca-2015-ap-northeast-2.pem
아시아 태평양(오사카-로컬)	rds-ca-2019-ap-northeast-3.pem	rds-ca-2015-ap-northeast-3.pem
아시아 태평양(싱가포르)	rds-ca-2019-ap-southeast-1.pem	rds-ca-2015-ap-southeast-1.pem
아시아 태평양(시드니)	rds-ca-2019-ap-southeast-2.pem	rds-ca-2015-ap-southeast-2.pem
캐나다(중부)	rds-ca-2019-ca-central-1.pem	rds-ca-2015-ca-central-1.pem
유럽(프랑크푸르트)	rds-ca-2019-eu-central-1.pem	rds-ca-2015-eu-central-1.pem
유럽(아일랜드)	rds-ca-2019-eu-west-1.pem	rds-ca-2015-eu-west-1.pem
유럽(런던)	rds-ca-2019-eu-west-2.pem	rds-ca-2015-eu-west-2.pem
유럽(파리)	rds-ca-2019-eu-west-3.pem	rds-ca-2015-eu-west-3.pem
유럽(스톡홀름)	rds-ca-2019-eu-north-1.pem	rds-ca-2015-eu-north-1.pem
중동(바레인)	rds-ca-2019-me-south-1.pem	사용 가능한 인증서가 없음
남아메리카(상파울루)	rds-ca-2019-sa-east-1.pem	rds-ca-2015-sa-east-1.pem
미국 동부(버지니아 북부)	rds-ca-2019-us-east-1.pem	rds-ca-2015-us-east-1.pem
미국 동부(오하이오)	rds-ca-2019-us-east-2.pem	rds-ca-2015-us-east-2.pem
미국 서부(캘리포니아 북부 지역)	rds-ca-2019-us-west-1.pem	rds-ca-2015-us-west-1.pem

AWS 리전	CA-2019	CA-2015
미국 서부(오레곤)	rds-ca-2019-us-west-2.pem	rds-ca-2015-us-west-2.pem

AWS GovCloud (US) 인증서

AWS GovCloud(미국) 리전의 루트 인증서는 다음 목록에서 다운로드할 수 있습니다.

[AWS GovCloud\(미국 동부\)\(루트 CA-2017\)](#)

[AWS GovCloud\(미국 서부\)\(루트 CA-2017\)](#)

AWS GovCloud(미국) 리전의 중간 인증서는 다음 목록에서 다운로드할 수 있습니다.

[AWS GovCloud\(US 동부\)\(CA-2017\)](#)

[AWS GovCloud\(US 서부\)\(CA-2017\)](#)

[AWS GovCloud\(US 서부\)\(CA-2012\)](#)

AWS GovCloud (US) 리전의 중간 인증서와 루트 인증서를 모두 포함하는 인증서 번들은 <https://s3.us-gov-west-1.amazonaws.com/rds-downloads/rds-combined-ca-us-gov-bundle.pem>에서 다운로드할 수 있습니다.

SSL/TLS 인증서 교체

Note

애플리케이션이 SSL(Secure Socket Layer) 또는 TLS(전송 계층 보안)를 사용하여 RDS DB에 연결하는 경우 2020년 3월 5일 이전에 다음 단계를 실행해야 합니다. 이렇게 하면 애플리케이션과 RDS DB 인스턴스 간에 연결이 중단되는 것을 방지할 수 있습니다.

현재 CA 인증서는 2020년 3월 5일에 만료됩니다. 인증서 확인과 함께 SSL/TLS를 사용하는 새 애플리케이션 연결은 2020년 3월 5일 이후에 실패합니다.

DB 인스턴스에서 새로운 CA 인증서를 사용하도록 업데이트하기 전에 RDS 데이터베이스에 연결하는 클라인트 또는 애플리케이션을 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 \(p. 947\)](#) 단원을 참조하십시오.

Note

Amazon RDS Proxy 및 Aurora Serverless 사용에서는 AWS Certificate Manager(ACM)의 인증서를 합니다. RDS Proxy를 사용하는 경우 SSL/TLS 인증서를 교체할 때 RDS Proxy 연결을 사용하는 애플리케이션을 업데이트할 필요가 없습니다. RDS Proxy에서 TLS/SSL을 사용하는 방법에 대한 자세한 내용은 [RDS Proxy에서 TLS/SSL 사용 \(p. 215\)](#) 단원을 참조하십시오.

Aurora Serverless를 사용하는 경우 SSL/TLS 인증서를 교체할 필요가 없습니다. Aurora Serverless에서 TLS/SSL을 사용하는 방법에 대한 자세한 내용은 [Aurora Serverless에서 TLS/SSL 사용 \(p. 114\)](#) 단원을 참조하십시오.

주제

- [DB 인스턴스를 수정하여 CA 인증서 업데이트 \(p. 950\)](#)
- [DB 인스턴스 유지 관리를 적용하여 CA 인증서 업데이트 \(p. 952\)](#)
- [트러스트 스토어로 인증서를 가져오기 위한 샘플 스크립트 \(p. 956\)](#)
- [CA 인증서 업데이트 되돌리기 \(p. 957\)](#)

- 새 DB 인스턴스에 대한 기본 CA 인증서 재정의 (p. 959)

DB 인스턴스를 수정하여 CA 인증서 업데이트

다음 단계를 완료하여 CA 인증서를 업데이트하십시오.

DB 인스턴스를 수정하여 CA 인증서를 업데이트하려면

1. [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#)에 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 SSL/TLS 인증서를 사용해 Aurora MySQL DB 클러스터에 연결할 애플리케이션을 업데이트 (p. 468)
- 새 SSL/TLS 인증서를 사용해 Aurora PostgreSQL DB 클러스터에 연결할 애플리케이션을 업데이트 (p. 777)

Linux 운영 체제의 트러스트 스토어를 업데이트하는 샘플 스크립트는 [트러스트 스토어로 인증서를 가져오기 위한 샘플 스크립트 \(p. 956\)](#) 단원을 참조하십시오.

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-2015에서 rds-ca-2019로 변경합니다.

Important

이 작업을 수행하면 기본적으로 DB 인스턴스가 재시작됩니다. 이 작업 중에 DB 인스턴스를 재시작하지 않으려면 `modify-db-instance` CLI 명령을 사용하여 `--no-certificate-rotation-restart` 옵션을 지정하면 됩니다.

이 옵션은 계획된 유지 관리 또는 계획되지 않은 유지 관리에 대해 다음에 데이터베이스를 다시 시작할 때까지 인증서를 교체하지 않습니다. 이 옵션은 SSL/TLS를 사용하지 않는 경우에만 권장됩니다.

인증서 만료 후 연결 문제가 발생하는 경우 콘솔에서 즉시 적용을 지정하거나 AWS CLI를 통해 `--apply-immediately` 옵션을 지정하여 즉시 적용 옵션을 사용합니다. 기본적으로 이 작업은 다음 유지 관리 기간 중에 실행되도록 예약되어 있습니다.

AWS Management 콘솔 또는 AWS CLI를 사용하여 DB 인스턴스의 CA 인증서를 rds-ca-2015에서 rds-ca-2019로 변경할 수 있습니다.

콘솔

DB 인스턴스의 CA를 rds-ca-2015에서 rds-ca-2019로 변경하려면

1. AWS Management 콘솔에 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.

The screenshot shows the AWS RDS console with the path: RDS > Databases > aurora-test > dbinstance1. The main title is "dbinstance1". On the left, there's a "Related" section with a search bar. Below it is a table with columns: DB identifier, Role, Engine, Region & AZ, and Size. The table contains four rows: aurora-test (Regional, Aurora MySQL, us-east-1, 3 instances), dbinstance4 (Writer, Aurora MySQL, us-east-1a, db.r5.large), dbinstance1 (Reader, Aurora MySQL, us-east-1b, db.r5.large), and dbinstance2 (Reader, Aurora MySQL, us-east-1b, db.r5.large). The dbinstance1 row is highlighted with a blue selection bar. At the bottom, there are tabs: Connectivity & security (which is active), Monitoring, Logs & events, Configuration, Maintenance, and Tags.

[Modify DB Instance] 페이지가 나타납니다.

4. 네트워크 및 보안 섹션에서 rds-ca-2019를 선택합니다.

The screenshot shows the "Certificate authority" section of the "Modify DB Instance" page. It asks for a certificate authority for the DB instance. A dropdown menu lists three options: rds-ca-2019 (selected), rds-ca-2015, and rds-ca-2019. Below the dropdown, a note says: "EC2 instances and devices outside of the VPC hosting the DB instance will connect to the DB instances. You can add more VPC security groups that specify which EC2 instances and devices can connect to the DB instance." There are two radio button options: "Yes" and "No".

5. [Continue]를 수정 사항을 요약한 내용을 확인합니다.

6. 변경 사항을 즉시 적용하려면 즉시 적용을 선택합니다.

Important

이 옵션을 선택하면 데이터베이스가 즉시 다시 시작됩니다.

7. 확인 페이지에서 변경 내용을 검토합니다. 변경 내용이 정확할 경우 [Modify DB Instance]를 선택하여 변경 내용을 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-2015에서 rds-ca-2019로 변경하려면 [modify-db-instance](#) 명령을 호출하십시오. DB 인스턴스 식별자 및 --ca-certificate-identifier 옵션을 지정하십시오.

Important

이 작업을 예약하는 경우 클라이언트 측 트러스트 스토어를 미리 업데이트했는지 확인하십시오.

Example

다음 코드에서는 CA 인증서를 rds-ca-2019로 설정하여 mydbinstance를 수정합니다. 변경 사항은 --no-apply-immediately를 사용하여 다음 유지 관리 기간에 적용됩니다. 변경 사항을 바로 적용하려면 --apply-immediately를 사용합니다.

Important

이 작업을 수행하면 기본적으로 DB 인스턴스가 재부팅됩니다. 이 작업 중에 DB 인스턴스를 재부팅하지 않으려면 modify-db-instance CLI 명령을 사용하여 --no-certificate-rotation-restart 옵션을 지정할 수 있습니다.

이 옵션은 계획된 유지 관리 또는 계획되지 않은 유지 관리에 대해 다음에 데이터베이스를 다시 시작할 때까지 인증서를 교체하지 않습니다. 이 옵션은 SSL/TLS를 사용하지 않는 경우에만 권장됩니다.

--apply-immediately를 사용하여 업데이트를 즉시 적용합니다. 기본적으로 이 작업은 다음 유지 관리 기간 중에 실행되도록 예약되어 있습니다.

Linux, OS X, Unix의 경우:

```
aws rds modify-db-instance \
--db-instance-identifier mydbinstance \
--ca-certificate-identifier rds-ca-2019 \
--no-apply-immediately
```

Windows의 경우:

```
aws rds modify-db-instance ^
--db-instance-identifier mydbinstance ^
--ca-certificate-identifier rds-ca-2019 ^
--no-apply-immediately
```

DB 인스턴스 유지 관리를 적용하여 CA 인증서 업데이트

DB 인스턴스 유지 관리를 적용하여 CA 인증서를 업데이트하려면 다음 단계를 수행합니다.

DB 인스턴스 유지 관리를 적용하여 CA 인증서를 업데이트하려면

1. [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#)에 설명된 대로 새 SSL/TLS 인증서를 다운로드합니다.
2. 새 SSL/TLS 인증서를 사용하도록 데이터베이스 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 SSL/TLS 인증서를 사용해 Aurora MySQL DB 클러스터에 연결할 애플리케이션을 업데이트 (p. 468)
- 새 SSL/TLS 인증서를 사용해 Aurora PostgreSQL DB 클러스터에 연결할 애플리케이션을 업데이트 (p. 777)

Linux 운영 체제의 트러스트 스토어를 업데이트하는 샘플 스크립트는 [트러스트 스토어로 인증서를 가져오기 위한 샘플 스크립트 \(p. 956\)](#) 단원을 참조하십시오.

Note

인증서 번들에는 이전 및 신규 CA의 인증서가 들어 있으므로 애플리케이션을 안전하게 업그레이드하고 전환 기간에 연결성을 유지할 수 있습니다.

3. DB 인스턴스 유지 관리를 적용하여 CA를 rds-ca-2015에서 rds-ca-2019로 변경합니다.

Important

변경 사항을 즉시 적용하도록 선택할 수 있습니다. 기본적으로 이 작업은 다음 유지 관리 기간 중에 실행되도록 예약되어 있습니다.

AWS Management 콘솔을 사용하면 DB 인스턴스 유지 관리를 적용하여 여러 DB 인스턴스에 대한 CA 인증서를 rds-ca-2015에서 rds-ca-2019로 변경할 수 있습니다.

[여러 DB 인스턴스에 유지 관리를 적용하여 CA 인증서 업데이트](#)

AWS Management 콘솔을 사용하여 여러 DB 인스턴스에 대한 CA 인증서를 변경합니다.

여러 DB 인스턴스에 대한 CA를 rds-ca-2015에서 rds-ca-2019로 변경하려면

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.

이전 CA 인증서를 사용 중인 DB 인스턴스가 하나 이상 있는 경우 페이지 상단에 다음 배너가 나타납니다.

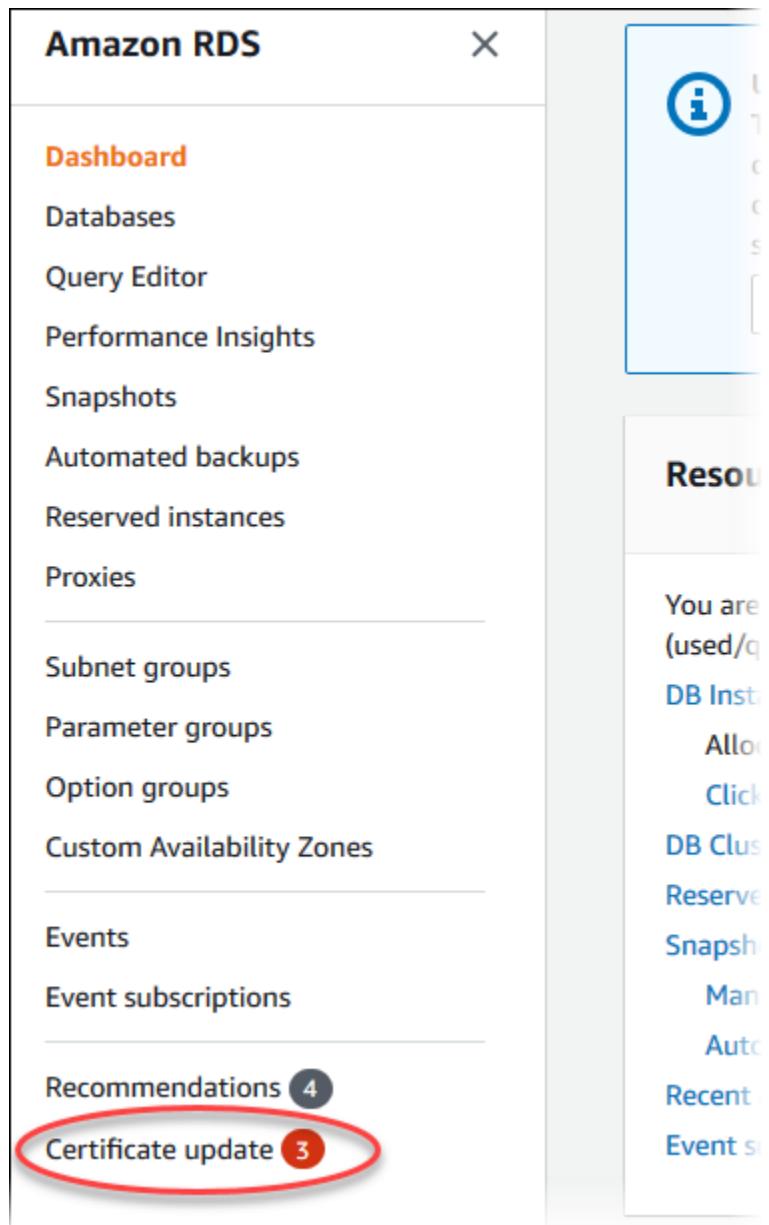


Update your Amazon RDS SSL/TLS certificates before March 5, 2020

To avoid interruption of your applications using RDS and Aurora databases, update the Certificate Authority (CA) certificates for these databases before March 5, 2020. We strongly recommend making your updates before February 5, 2020, to leave time for deployments, testing, and validation. New databases created after January 14, 2020, will default to using the new CA certificates. Make sure that you update your client applications with the new certificates first. Find the new CA certificates and info: [RDS](#) [Aurora](#)

[View pending maintenance actions](#)

탐색 창에는 영향을 받는 DB 인스턴스의 총 수를 보여 주는 Certificate update(인증서 업데이트) 옵션이 있습니다.



배너에서 View pending maintenance actions(보류 중인 유지 관리 작업 보기)를 선택하거나 탐색 창에서 Certificate update(인증서 업데이트)를 선택합니다.

Update your Amazon RDS SSL/TLS certificates(Amazon RDS SSL/TLS 인증서 업데이트) 페이지가 나타납니다.

The screenshot shows a list of databases requiring update. There are three entries:

DB identifier	DB cluster identifier	Status	Apply date
mydbinstancecf	-	Requires Update	-
mydbinstancecf2	-	Requires Update	-
oracledb	-	Requires Update	-

Note

이 페이지에는 현재 AWS 리전의 DB 인스턴스만 표시됩니다. 더 이상의 AWS 리전에 DB 인스턴스가 있는 경우 각 AWS 리전에서 이 페이지를 확인하여 이전 SSL/TLS 인증서가 있는 모든 DB 인스턴스를 확인합니다.

- 업데이트할 DB 인스턴스를 선택합니다.

Update at the next maintenance window(다음 유지 관리 기간에 업데이트)를 선택하여 다음 유지 관리 기간에 인증서 교체를 예약할 수 있습니다. Update now(지금 업데이트)를 선택하여 즉시 교체를 적용합니다.

Important

CA 인증서가 교체되면 이 작업은 DB 인스턴스를 재시작합니다.

인증서 만료 후 연결 문제가 발생하면 Update now(지금 업데이트) 옵션을 사용합니다.

- Update at the next maintenance window(다음 유지 관리 기간에 업데이트) 또는 Update now(지금 업데이트)를 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. 애플리케이션이 SSL/TLS 및 서버 인증서를 사용하여 연결하는지 확인하려면 [2단계: 새 SSL/TLS 인증서를 사용하도록 데이터베이스 애플리케이션을 업데이트 \(p. 952\)](#) 단원을 참조하십시오. 이러한 클라이언트 애플리케이션을 업데이트한 후 CA 인증서 교체를 확인할 수 있습니다.

Confirm rotation of CA certificate?

Before scheduling the CA certificate rotation, update client applications that connect to your database to use the new CA certificate. Not doing this will cause an interruption of connectivity between your applications and your database. [Get new CA certificates](#)

I understand that not doing so will break SSL/TLS connectivity to my database.

Cancel

Confirm

계속하려면 확인란을 선택한 다음 확인선택합니다.

5. 업데이트할 각 DB 인스턴스에 대해 3단계와 4단계를 반복합니다.

트러스트 스토어로 인증서를 가져오기 위한 샘플 스크립트

다음은 인증서 번들을 트러스트 스토어로 가져오는 샘플 셀 스크립트입니다.

주제

- [Linux에서 인증서를 가져오기 위한 샘플 스크립트 \(p. 956\)](#)
- [macOS에서 인증서를 가져오기 위한 샘플 스크립트 \(p. 956\)](#)

Linux에서 인증서를 가져오기 위한 샘플 스크립트

다음은 Linux 운영 체제에서 트러스트 스토어로 인증서 번들을 가져오는 샘플 셀 스크립트입니다.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi

truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -ss "https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem" > ${mydir}/
rds-combined-ca-bundle.pem
awk 'split_after == 1 {n++;split_after=0} /-----END CERTIFICATE-----/ {split_after=1}{print
> "rds-ca-" n ".pem"}' < ${mydir}/rds-combined-ca-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}/rds-combined-ca-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
```

macOS에서 인증서를 가져오기 위한 샘플 스크립트

다음은 macOS에서 트러스트 스토어로 인증서 번들을 가져오는 샘플 셀 스크립트입니다.

```
mydir=tmp/certs
if [ ! -e "${mydir}" ]
then
mkdir -p "${mydir}"
fi

truststore=${mydir}/rds-truststore.jks
storepassword=changeit

curl -ss "https://s3.amazonaws.com/rds-downloads/rds-combined-ca-bundle.pem" > ${mydir}/
rds-combined-ca-bundle.pem
split -p "-----BEGIN CERTIFICATE-----" ${mydir}/rds-combined-ca-bundle.pem rds-ca-
for CERT in rds-ca-*; do
alias=$(openssl x509 -noout -text -in $CERT | perl -ne 'next unless /Subject:/; s/.*\nCN = )//; print')
echo "Importing $alias"
keytool -import -file ${CERT} -alias "${alias}" -storepass ${storepassword} -keystore
${truststore} -noprompt
rm $CERT
done

rm ${mydir}/rds-combined-ca-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
```

CA 인증서 업데이트 되돌리기

AWS Management 콘솔 또는 AWS CLI를 사용하여 DB 인스턴스의 CA 인증서를 이전 상태로 되돌릴 수 있습니다. 2020년 3월 5일 이후에는 새 인증서로 업데이트를 되돌릴 수 없습니다.

콘솔

DB 인스턴스의 CA 인증서를 이전 상태로 되돌리려면

1. AWS Management 콘솔에 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.

The screenshot shows the AWS RDS console with the path: RDS > Databases > aurora-test > dbinstance1. The main title is "dbinstance1". On the left, there's a "Related" section with a search bar. Below it is a table with columns: DB identifier, Role, Engine, Region & AZ, and Size. The table contains four rows: aurora-test (Regional, Aurora MySQL, us-east-1, 3 instances), dbinstance4 (Writer, Aurora MySQL, us-east-1a, db.r5.lar), dbinstance1 (Reader, Aurora MySQL, us-east-1b, db.r5.lar), and dbinstance2 (Reader, Aurora MySQL, us-east-1b, db.r5.lar). The dbinstance1 row is highlighted with a blue selection bar. At the bottom, there are tabs for Connectivity & security, Monitoring, Logs & events, Configuration, Maintenance, and Tags.

[Modify DB Instance] 페이지가 나타납니다.

4. 네트워크 및 보안 섹션에서 rds-ca-2015를 선택합니다.

The screenshot shows the "Certificate authority" section of the "Modify DB Instance" wizard. It asks for a certificate authority for the DB instance. A dropdown menu shows three options: "rds-ca-2015" (selected), "rds-ca-2015" (highlighted with a blue border), and "rds-ca-2019". Below the dropdown is a note: "EC2 instances and devices outside of the VPC hosting the DB instance will connect to the DB instances. You may need to add more VPC security groups that specify which EC2 instances and devices can connect to the DB instance." There is also a "No" radio button.

5. [Continue]를 수정 사항을 요약한 내용을 확인합니다.
6. 변경 사항을 즉시 적용하려면 즉시 적용을 선택합니다.

Important

이 옵션을 선택하면 데이터베이스가 즉시 재시작됩니다.

7. 확인 페이지에서 변경 내용을 검토합니다. 변경 내용이 정확할 경우 [Modify DB Instance]를 선택하여 변경 내용을 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

DB 인스턴스의 CA 인증서를 이전 상태로 되돌리려면 [modify-db-instance](#) 명령을 호출하십시오. DB 인스턴스 식별자 및 --ca-certificate-identifier 옵션을 지정하십시오.

Important

이 작업을 예약하는 경우 클라이언트 측 트러스트 스토어를 미리 업데이트했는지 확인하십시오.

Example

다음 코드 예제에서는 CA 인증서를 rds-ca-2015로 설정하여 mydbinstance를 수정합니다. 변경 사항은 --no-apply-immediately를 사용하여 다음 유지 관리 기간에 적용됩니다. 변경 사항을 바로 적용하려면 --apply-immediately를 사용합니다.

Important

--certificate-rotation-restart 옵션이 지정될 때 이 --apply-immediately 옵션을 사용하면 중단이 발생합니다. --certificate-rotation-restart 옵션(기본값)은 인증서가 교체될 때 DB 인스턴스가 다시 시작되도록 지정합니다.

--no-certificate-rotation-restart 옵션은 인증서가 교체될 때 DB 인스턴스가 다시 시작되지 않도록 지정합니다. SSL/TLS를 사용하여 DB 인스턴스에 연결하지 않는 경우에만 --no-certificate-rotation-restart 옵션을 사용합니다.

Linux, OS X, Unix의 경우:

```
aws rds modify-db-instance \  
  --db-instance-identifier mydbinstance \  
  --ca-certificate-identifier rds-ca-2015 \  
  --no-apply-immediately
```

Windows의 경우:

```
aws rds modify-db-instance ^  
  --db-instance-identifier mydbinstance ^  
  --ca-certificate-identifier rds-ca-2015 ^  
  --no-apply-immediately
```

새 DB 인스턴스에 대한 기본 CA 인증서 재정의

2020년 1월 14일 이후에 생성된 모든 새 RDS DB 인스턴스에서는 기본적으로 새 인증서를 사용합니다. 새 DB 인스턴스를 일시적으로 수동 변경하여 이전(rds-ca-2015) 인증서를 사용할 수 있게 하려면 AWS Management 콘솔 또는 AWS CLI를 사용하면 됩니다. 2020년 1월 14일 이전에 생성된 모든 DB 인스턴스에서는 rds-ca-2019 인증서로 업데이트하기 전까지 rds-ca-2015 인증서를 사용합니다.

새 DB 인스턴스에 대해 시스템 기본 CA 인증서를 임시로 재정의하거나 재정의를 제거하려면 [modify-certificates](#) 명령을 사용합니다.

이 명령을 사용하면 RDS에서 제공하는 기본 인증서와 다른 새로운 DB 인스턴스에 대해 RDS가 승인한 SSL/TLS 인증서를 지정할 수 있습니다. 이 작업을 사용하여 새 DB 인스턴스가 RDS에서 제공하는 기본 인증서를 사용하도록 재정의를 제거할 수도 있습니다.

다음과 같은 경우 기본 인증서를 재정의해야 할 수 있습니다.

- 최신 CA 인증서를 지원하기 위해 애플리케이션을 이미 마이그레이션했지만, 새 CA 인증서는 아직 지정된 AWS 리전의 RDS 기본 CA 인증서가 아닙니다.
- RDS가 이미 지정된 AWS 리전에 대한 새로운 기본 CA 인증서로 이동했지만 아직 새 CA 인증서를 지원하는 중입니다. 이 경우 애플리케이션 변경을 완료하는 데 일시적으로 추가 시간이 필요합니다.

Note

RDS 콘솔을 사용하여 기본 CA 인증서를 재정의하거나 재정의를 제거할 수 없습니다.

Example

다음 코드 예제에서는 기본 CA 인증서를 rds-ca-2019로 설정합니다. 변경이 바로 적용됩니다.

```
aws rds modify-certificates --certificate-identifier rds-ca-2019
```

인터넷워크 트래픽 개인 정보

Amazon Aurora와 온프레미스 애플리케이션 사이 연결을 비롯해 Amazon Aurora와 동일한 AWS 리전의 다른 AWS 리소스 사이 연결이 보호를 받습니다.

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 VPN이란 무엇입니까?](#)를 참조하십시오.
- AWS Direct Connect 연결. 자세한 내용은 [AWS Direct Connect란 무엇입니까?](#)를 참조하십시오.

AWS 게시 API 작업을 사용하면 네트워크를 통해 Amazon Aurora에 액세스할 수 있습니다. 클라이언트가 TLS(전송 계층 보안) 1.0을 지원해야 합니다. TLS 1.2를 권장합니다. 클라이언트는 DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Diffie-Hellman Ephemeral)와 같은 PFS(전달 완전 보안)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다. 또한 IAM 보안 주체와 연결되어 있는 액세스 키 ID 및 보안 액세스 키를 사용해 요청서에 서명해야 합니다. 혹은 [AWS Security Token Service\(STS\)](#)를 사용해 요청서에 서명하기 위한 임시 보안 자격 증명을 생성할 수도 있습니다.

같은 리전에 있는 AWS 리소스 사이의 트래픽

Amazon Aurora의 Amazon Virtual Private Cloud(Amazon VPC) 엔드포인트는 Amazon Aurora에만 연결을 허용하는 VPC 내 논리적 개체입니다. Amazon VPC는 Amazon Aurora으로 요청을 라우팅하고, 응답을 다시 VPC로 라우팅합니다. 자세한 내용은 Amazon VPC 사용 설명서의 [VPC 엔드포인트](#)를 참조하십시오. VPC 엔드포인트에서 DB 클러스터 액세스를 제어하는 데 사용할 수 있는 샘플 정책은 [IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 \(p. 979\)](#) 단원을 참조하십시오.

Amazon Aurora의 Identity and Access Management(IAM)

AWS Identity and Access Management(IAM)은 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 Aurora 리소스를 사용하기 위해 인증(로그인) 및 권한 부여(권한 있음) 할 수 있는 사람을 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제

- [대상 \(p. 961\)](#)
- [자격 증명을 통한 인증 \(p. 961\)](#)
- [정책을 이용한 액세스 관리 \(p. 963\)](#)

- Amazon Aurora에서 IAM을 사용하는 방식 (p. 964)
- Amazon Aurora 자격 증명 기반 정책 예제 (p. 967)
- 을 위한 IAM 데이터베이스 인증 (p. 976)
- Amazon Aurora 자격 증명 및 액세스 문제 해결 (p. 993)

대상

AWS Identity and Access Management(IAM) 사용 방법은 Aurora에서 하는 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 \(p. 993\)](#) 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 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사가 Aurora에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 [Amazon Aurora에서 IAM을 사용하는 방식 \(p. 964\)](#) 단원을 참조하십시오.

IAM 관리자 – IAM 관리자는 Aurora에 대한 액세스 권한을 관리할 수 있는 정책을 작성하는 방법에 대해 자세히 알아보고 싶을 수 있습니다. IAM에서 사용할 수 있는 예제 Aurora 자격 증명 기반 정책 예제를 보려면 [Amazon Aurora 자격 증명 기반 정책 예제 \(p. 967\)](#) 단원을 참조하십시오.

자격 증명을 통한 인증

인증은 ID 자격 증명을 사용하여 AWS에 로그인하는 방식입니다. AWS Management 콘솔을 사용하여 로그인하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 콘솔 및 로그인 페이지](#)를 참조하십시오.

AWS 계정 루트 사용자, class="non-printable-char non-printable-space"> IAM class="non-printable-char non-printable-space"> 사용자로서 class="non-printable-char non-printable-space"> 또는 class="non-printable-char non-printable-space"> IAM class="non-printable-char non-printable-space"> 역할을 class="non-printable-char non-printable-space"> 수임하여 class="non-printable-char non-printable-space"> 인증(AWS에 class="non-printable-char non-printable-space"> 로그인)되어야 class="non-printable-char non-printable-space"> 합니다. 회사의 싱글 사인온(SSO) 인증을 사용하거나 Google 또는 Facebook을 사용하여 로그인할 수도 있습니다. 이러한 경우 관리자는 이전에 IAM 역할을 사용하여 자격 증명 연동을 설정한 것입니다. 다른 회사의 자격 증명을 사용하여 AWS에 액세스하면 간접적으로 역할을 가정하는 것입니다.

[AWS Management 콘솔](#)에 직접 로그인하려면 루트 사용자 이메일이나 IAM 사용자 이름과 비밀번호를 사용하십시오. 루트 사용자 또는 IAM 사용자 액세스 키를 사용하여 프로그래밍 방식으로 AWS에 액세스할 수 있습니다. AWS는 자격 증명을 사용하여 암호화 방식으로 요청에 서명할 수 있는 SDK 및 명령줄 도구를 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 이렇게 하려면 인바운드 API 요청을 인증하기 위한 프로토콜인 서명 버전 4를 사용합니다. 요청 인증에 대한 자세한 정보는 AWS General Reference의 [서명 버전 4 서명 프로세스](#) 단원을 참조하십시오.

사용하는 인증 방법에 상관 없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS는 멀티 팩터 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 IAM 사용 설명서의 [AWS에서 멀티 팩터 인증\(MFA\) 사용](#)을 참조하십시오.

AWS 계정 루트 사용자

AWS 계정을 처음 생성할 때는 해당 계정의 모든 AWS 서비스 및 리소스에 대한 완전한 액세스 권한이 있는 SSO(Single Sign-In) ID로 시작합니다. 이 자격 증명은 AWS 계정 루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소로 로그인하여 액세스합니다. 일상적인 작업은 물론 관리 작업에도 루트 사용자를 사용하지 않는 것이 좋습니다. 대신 [IAM 사용자를 처음 생성할 때만 루트 사용자를 사용하는 모범 사례](#)를 준

수하십시오. 그런 다음 루트 사용자 자격 증명을 안전하게 보관해 두고 몇 가지 계정 및 서비스 관리 작업을 수행할 때만 해당 자격 증명을 사용합니다.

IAM 사용자 및 그룹

[IAM 사용자](#)는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. IAM 사용자에게는 사용자 이름과 암호 또는 액세스 키 세트와 같은 장기 자격 증명이 있을 수 있습니다. 액세스 키를 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 사용자의 액세스 키 관리](#)를 참조하십시오. IAM 사용자의 액세스 키를 생성할 때는 키 페어를 보고 안전하게 저장해야 합니다. 향후에 보안 액세스 키를 복구할 수 없습니다. 그 대신 새 액세스 키 페어를 생성해야 합니다.

[IAM 그룹](#)은 IAM 사용자 컬렉션을 지정하는 ID입니다. 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, IAMAdmins이라는 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 자격 증명만 제공합니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자\(역할 대신\)를 생성하는 경우](#)를 참조하십시오.

IAM 데이터베이스 인증을 사용하여 DB 클러스터에 인증할 수 있습니다.

IAM 데이터베이스 인증은 Aurora에서 유효합니다. IAM을 사용한 DB 클러스터 인증에 대한 자세한 내용은 [을 위한 IAM 데이터베이스 인증 \(p. 976\)](#) 단원을 참조하십시오.

IAM 역할

[IAM 역할](#)은 특정 권한을 가지고 있는 AWS 계정 내 ID입니다. 이 역할은 IAM 사용자와 비슷하지만, 특정 개인과 연결되지 않습니다. [역할을 전환](#)하여 AWS Management 콘솔에서 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 내용은 IAM 사용 설명서의 [IAM 역할 사용](#)을 참조하십시오.

임시 자격 증명이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- **임시 IAM 사용자 권한** – IAM 사용자는 IAM 역할을 수임하여 특정 작업에 대한 다른 권한을 임시로 받을 수 있습니다.
- **연합된 사용자 액세스** – IAM 사용자를 생성하는 대신 AWS Directory Service의 기준 ID, 엔터프라이즈 사용자 디렉토리 또는 웹 ID 공급자를 사용할 수 있습니다. 이 사용자를 연합된 사용자라고 합니다. AWS에서는 [ID 공급자](#)를 통해 액세스가 요청되면 연합된 사용자에게 역할을 할당합니다. 연합된 사용자에 대한 자세한 내용은 IAM 사용 설명서의 [연합된 사용자 및 역할](#)을 참조하십시오.
- **교차 계정 액세스** – IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 할 수 있습니다. 역할은 교차 계정 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스를 사용하면 역할을 프록시로 사용하는 대신 리소스에 정책을 직접 연결할 수 있습니다. 교차 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 [IAM 역할과 리소스 기반 정책의 차이](#)를 참조하십시오.
- **AWS 서비스 액세스** – 서비스 역할은 서비스가 사용자를 대신하여 사용자 계정에서 작업을 수행하기 위해 수임하는 IAM 역할입니다. 일부 AWS 서비스 환경을 설정할 때 서비스에서 맙을 역할을 정의해야 합니다. 이 서비스 역할에는 서비스가 AWS 리소스에 액세스하는 데 필요한 모든 권한이 포함되어야 합니다. 서비스 역할은 서비스마다 다르지만 해당 서비스에 대한 문서화된 요구 사항을 충족하는 한 대부분의 경우 권한을 선택할 수 있습니다. 서비스 역할은 해당 계정 내 액세스 권한만 제공하며 다른 계정의 서비스에 대한 액세스 권한을 부여하는 데 사용될 수 없습니다. IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 예를 들어 Amazon Redshift에서 사용자 대신 Amazon S3 버킷에 액세스하도록 허용하는 역할을 생성한 후 해당 버킷에 있는 데이터를 Amazon Redshift 클러스터로 로드할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 서비스에 대한 권한을 위임할 역할 생성](#)을 참조하십시오.
- **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 인스턴

스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 자격 증명을 얻을 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 대한 부여](#)를 참조하십시오.

IAM 역할을 사용할지 여부를 알아보려면 IAM 사용 설명서의 [사용자 대신 IAM 역할을 생성해야 하는 경우](#)를 참조하십시오.

정책을 이용한 액세스 관리

정책을 생성하고 IAM 자격 증명 또는 AWS 리소스에 연결하여 AWS 액세스를 제어합니다. 정책은 자격 증명이나 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 엔터티(루트 사용자, IAM 사용자 또는 IAM 역할)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는지 여부를 결정합니다. 대부분의 정책은 AWS에 JSON 문서로서 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 내용은 IAM 사용 설명서의 [JSON 정책 개요](#)를 참조하십시오.

IAM 관리자는 AWS 리소스에 액세스할 수 있는 사람과 해당 리소스에 대해 수행할 수 있는 작업을 지정할 수 있습니다. 모든 IAM 엔터티(사용자 또는 역할)는 처음에는 권한이 없습니다. 다시 말해, 기본적으로 사용자는 아무 작업도 수행할 수 없으며, 자신의 암호를 변경할 수도 없습니다. 사용자에게 작업을 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 작업을 실행하기 위한 방법과 상관없이 작업을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management 콘솔, AWS CLI 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

자격 증명 기반 정책

자격 증명 기반 정책은 IAM 사용자, 역할 또는 그룹과 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 생성](#)을 참조하십시오.

자격 증명 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 사용자, 그룹 및 역할에게 독립적으로 추가할 수 있는 정책입니다. 관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함되어 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책과 인라인 정책의 선택](#)을 참조하십시오.

계정의 사용자에게 연결할 수 있는 다음 AWS 관리형 정책은 Amazon Aurora에 대해 고유합니다.

- `AmazonRDSReadOnlyAccess` – 지정된 AWS 계정의 모든 Amazon Aurora 리소스에 대한 읽기 전용 액세스를 부여합니다.
- `AmazonRDSFullAccess` – 지정된 AWS 계정의 모든 Amazon Aurora 리소스에 대한 전체 액세스를 부여합니다.

기타 정책 유형

AWS는 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 유형은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 자격 증명 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 엔터티에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 엔터티의 자격 증명 기반 정책의 교차와 그 권한 경계입니다. `Principal` 필드에서 사용자나 역할

을 보안 주체로 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 엔터티에 대한 권한 경계](#)를 참조하십시오.

- 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 기능을 활성화할 경우 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자를 비롯하여 멤버 계정의 엔터티에 대한 권한을 제한합니다. 조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 [SCP의 작동 방식](#) 단원을 참조하십시오.
- 세션 정책 – 세션 정책은 역할 또는 연합된 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 자격 증명 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 내용은 IAM 사용 설명서의 [세션 정책](#)을 참조하십시오.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지 여부를 결정하는 방법을 알아보려면 IAM 사용 설명서에서 [정책 평가 로직](#) 단원을 참조하십시오.

Aurora의 자격 증명 및 액세스 관리에 대해 자세히 알아보려면 다음 페이지로 진행하십시오.

- [Amazon Aurora에서 IAM을 사용하는 방식](#) (p. 964)
- [Amazon Aurora 자격 증명 및 액세스 문제 해결](#) (p. 993)

Amazon Aurora에서 IAM을 사용하는 방식

IAM을 사용하여 Aurora에 대한 액세스를 관리하려면 먼저 어떤 IAM 기능을 Aurora에 사용할 수 있는지를 이해해야 합니다. Aurora 및 기타 AWS 서비스가 IAM과 작용하는 방법의 상위 수준 보기를 얻으려면 IAM 사용 설명서에서 [IAM으로 작업하는 AWS 서비스](#) 단원을 참조하십시오.

주제

- [Aurora자격 증명 기반 정책](#) (p. 964)
- [Aurora리소스 기반 정책](#) (p. 966)
- [Aurora 태그 기반 권한 부여](#) (p. 966)
- [AuroraIAM 역할](#) (p. 966)

Aurora자격 증명 기반 정책

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스 및 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. Aurora는 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하십시오.

Actions

IAM 자격 증명 기반 정책의 Action 요소는 정책에 따라 허용되거나 거부되는 특정 작업에 대해 설명합니다. 일반적으로 정책 작업의 이름은 연결된 AWS API 작업의 이름과 동일합니다. 이 작업은 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에서 사용됩니다.

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 설명문에는 Action 또는 NotAction 요소가 있어야 합니다. Aurora는 이 서비스를 통해 수행할 수 있는 작업을 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, Describe라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "rds:Describe*"
```

Aurora 작업 목록을 보려면 IAM 사용 설명서에서 [Amazon RDS에서 정의한 작업](#) 단원을 참조하십시오.

리소스

텍스트를 사용합니다.

Resource 요소는 작업이 적용되는 객체를 지정합니다. 문에는 Resource 또는 NotResource 요소가 반드시 추가되어야 합니다. ARN을 사용하거나 문이 모든 리소스에 적용됨을 표시하는 와일드카드(*)를 사용하여 리소스를 지정합니다.

DB 인스턴스 리소스에는 다음 ARN이 있습니다.

```
arn:${Partition}:rds:${Region}:${Account}:{ResourceType}/${Resource}
```

ARN 형식에 대한 자세한 내용은 [Amazon 리소스 이름\(ARN\) 및 AWS 서비스 네임스페이스](#)를 참조하십시오.

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 사용해야 합니다.

```
"Resource": "*"
```

다양한 Amazon RDS API 작업에는 여러 리소스가 관여합니다. 예를 들어 CreateDBInstance는 DB 인스턴스를 생성합니다. DB 인스턴스를 생성할 때는 IAM 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 IAM 사용 설명서에서 [Amazon RDS에서 정의한 리소스](#) 단원을 참조하십시오. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 [Amazon RDS에서 정의한 작업](#) 단원을 참조하십시오.

조건 키

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 선택 사항입니다. 같음, 미만 등 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 빌드할 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 태그](#) 단원을 참조하십시오.

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 키](#)를 참조하십시오.

모든 RDS API 작업은 `aws:RequestedRegion` 조건 키를 지원합니다.

Aurora 조건 키 목록을 확인하려면 IAM 사용 설명서에서 [Amazon RDS의 조건 키](#) 단원을 참조하십시오. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 [Amazon RDS에서 정의한 작업](#) 단원을 참조하십시오.

예제

Aurora 자격 증명 기반 정책 예제를 보려면 [Amazon Aurora 자격 증명 기반 정책 예제 \(p. 967\)](#) 단원을 참조하십시오.

Aurora 리소스 기반 정책

Aurora는 리소스 기반 정책을 지원하지 않습니다.

Aurora 태그 기반 권한 부여

텍스트를 사용합니다.

태그를 Aurora 리소스에 연결하거나 요청을 통해 태그를 Aurora에 전달할 수 있습니다. 태그를 기반으로 액세스를 제어하려면 `rds:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 [조건 요소](#)에 태그 정보를 제공하십시오. Aurora 리소스 태그 지정에 대한 자세한 내용은 [조건 지정: 사용자 지정 태그 사용 \(p. 973\)](#) 단원을 참조하십시오.

리소스에서 태그를 기반으로 해당 리소스에 대한 액세스를 제한하는 자격 증명 기반 정책 예제를 보려면 [두 가지 값이 있는 특정 태그를 사용하는 리소스 작업에 대한 권한 부여 \(p. 971\)](#) 단원을 참조하십시오.

Aurora IAM 역할

로 바꿉니다.

[IAM 역할](#)은 특정 권한을 가지고 있는 AWS 계정 내 개체입니다.

Aurora에서 임시 자격 증명 사용

임시 자격 증명을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 수임하거나, 교차 계정 역할을 수입할 수 있습니다. `AssumeRole` 또는 `GetFederationToken` 같은 AWS STS API 작업을 호출하여 임시 보안 자격 증명을 가져옵니다.

Aurora는 임시 자격 증명 사용을 지원합니다.

서비스 연결 역할

[서비스 연결 역할](#)을 사용하면 AWS 제품이 다른 서비스의 리소스에 액세스하여 사용자 대신 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수는 없습니다.

Aurora에서는 서비스 연결 역할을 지원합니다. Aurora 서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 [Amazon Aurora에 서비스 연결 역할 사용 \(p. 1001\)](#) 단원을 참조하십시오.

서비스 역할

이 기능을 사용하면 서비스가 사용자를 대신해 [서비스 역할](#)을 맡을 수 있습니다. 이 역할을 사용하면 서비스는 다른 서비스의 리소스에 액세스해 사용자를 대신해 작업을 완료할 수 있습니다. 서비스 역할은 IAM 계정에 나타나고, 해당 계정이 소유합니다. 즉, IAM 관리자가 이 역할에 대한 권한을 변경할 수 있습니다. 그러나 권한을 변경하면 서비스의 기능이 손상될 수 있습니다.

Aurora는 서비스 역할을 지원합니다.

Amazon Aurora 자격 증명 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 Aurora 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management 콘솔, AWS CLI 또는 AWS API를 사용해 작업을 수행할 수 없습니다. IAM 관리자는 지정된 리소스에서 특정 API 작업을 수행할 수 있는 권한을 사용자와 역할에게 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한이 필요한 IAM 사용자 또는 그룹에 이러한 정책을 연결해야 합니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [JSON 탭에서 정책 생성](#)을 참조하십시오.

주제

- [정책 모범 사례 \(p. 967\)](#)
- [Aurora 콘솔 사용 \(p. 968\)](#)
- [사용자가 자신이 권한을 볼 수 있도록 허용 \(p. 968\)](#)
- [사용자에게 AWS 계정에서 DB 인스턴스를 생성하도록 허용 \(p. 969\)](#)
- [콘솔 사용에 필요한 권한 \(p. 970\)](#)
- [사용자가 모든 RDS 리소스에서 Describe 작업을 수행할 수 있도록 허용 \(p. 970\)](#)
- [사용자가 지정된 DB 파라미터 및 보안 그룹을 사용하는 DB 인스턴스를 생성할 수 있도록 허용 \(p. 970\)](#)
- [두 가지 값이 있는 특정 태그를 사용하는 리소스 작업에 대한 권한 부여 \(p. 971\)](#)
- [사용자의 DB 인스턴스 삭제 방지 \(p. 971\)](#)
- [정책 예: 조건 키 사용 \(p. 972\)](#)
- [조건 지정: 사용자 지정 태그 사용 \(p. 973\)](#)

정책 모범 사례

자격 증명 기반 정책은 매우 강력합니다. 이 정책은 계정에서 사용자가 Aurora 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부를 결정합니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. 자격 증명 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따르십시오.

- AWS 관리형 정책을 사용하여 시작하기 – Aurora 사용을 빠르게 시작하려면 AWS 관리형 정책을 사용하여 필요한 권한을 직원에게 부여합니다. 이 정책은 이미 계정에서 사용할 수 있으며 AWS에 의해 유지 관리 및 업데이트됩니다. 자세한 내용은 IAM 사용 설명서의 [AWS 관리형 정책으로 권한 사용 시작하기](#)를 참조하십시오.
- 최소 권한 부여 – 사용자 지정 정책을 생성할 때는 작업을 수행하는 데 필요한 권한만 부여합니다. 최소한의 권한 조합으로 시작하여 필요에 따라 추가 권한을 부여합니다. 처음부터 권한을 많이 부여한 후 나중에

줄이는 방법보다 이 방법이 안전합니다. 자세한 내용은 IAM 사용 설명서의 [최소 권한 부여](#)를 참조하십시오.

- 중요한 작업에 대해 MFA 활성화 – 보안을 강화하기 위해 IAM 사용자가 중요한 리소스 또는 API 작업에 액세스하려면 멀티 팩터 인증(MFA)을 사용해야 합니다. 자세한 내용은 IAM 사용 설명서의 [AWS에서 Multi-Factor Authentication\(MFA\) 사용하기](#)를 참조하십시오.
- 보안 강화를 위해 정책 조건 사용 – 실제로 가능한 경우, 자격 증명 기반 정책이 리소스에 대한 액세스를 허용하는 조건을 정의합니다. 예를 들어 요청을 할 수 있는 IP 주소의 범위를 지정하도록 조건을 작성할 수 있습니다. 지정된 날짜 또는 시간 범위 내에서만 요청을 허용하거나, SSL 또는 MFA를 사용해야 하는 조건을 작성할 수도 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하십시오.

Aurora 콘솔 사용

Amazon Aurora 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한은 AWS 계정에서 Aurora 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다. 최소 필수 권한보다 더욱 제한적인 자격 증명 기반 정책을 생성할 수 있습니다. 하지만 이러한 정책에서는 콘솔 기능이 개체(IAM 사용자 또는 역할)에 대해 의도한 대로 실행되지 않습니다.

해당 개체가 Aurora 콘솔을 여전히 사용할 수 있도록 하려면 AWS 관리형 정책도 개체에 연결합니다. 자세한 내용은 IAM 사용 설명서의 [사용자에게 권한 추가](#) 단원을 참조하십시오.

AmazonRDSReadOnlyAccess

AWS CLI 또는 AWS API만 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자가 자신이 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 자격 증명에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여 줍니다. 이 정책에는 콘솔이나 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로 이 작업을 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 ListPolicies",  
                "iam ListPolicy"  
            ]  
        }  
    ]  
}
```

```
        "iam>ListUsers"
    ],
    "Resource": "*"
}
]
```

사용자에게 AWS 계정에서 DB 인스턴스를 생성하도록 허용

다음은 ID가 123456789012인 사용자에게 AWS 계정에 대한 DB 인스턴스를 생성하도록 허용하는 정책 예제입니다. 이 정책에 따라 새 DB 인스턴스의 이름은 test로 시작해야 합니다. 또한 새 DB 인스턴스는 MySQL 데이터베이스 엔진 및 db.t2.micro DB 인스턴스 클래스를 사용해야 합니다. 또한 새로운 DB 인스턴스는 default로 시작하는 옵션 그룹과 DB 파라미터 그룹을, 그리고 default 서브넷 그룹을 사용해야 합니다.

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

정책은 다음 IAM 사용자 권한을 지정하는 단일 명령문을 포함합니다.

- 정책은 IAM 사용자가 [CreateDBInstance API](#) 작업을 사용하여 DB 인스턴스를 생성할 수 있도록 허용합니다. 이는 [create-db-instance](#) AWS CLI 명령과 AWS Management 콘솔에도 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), 사용자 계정 번호(위의 예제에서 사용하는 사용자 ID는 123456789012임), 그리고 리소스 유형이 포함됩니다. ARN 생성에 대한 자세한 내용은 [Amazon RDS의 Amazon 리소스 이름\(ARN\)을 사용한 작업 \(p. 319\)](#) 단원을 참조하십시오.

위의 예제에서 Resource 요소는 사용자 리소스에 대해 다음과 같은 정책 제약 조건을 지정합니다.

- 새 DB 인스턴스에 대한 DB 인스턴스 식별자는 test로 시작해야 합니다(예: testCustomerData1, test-region2-data).
- 새로운 DB 인스턴스의 옵션 그룹은 default로 시작해야 합니다.
- 새로운 DB 인스턴스의 DB 파라미터 그룹은 default로 시작해야 합니다.
- 새로운 DB 인스턴스의 서브넷 그룹은 default 서브넷 그룹이 되어야 합니다.
- Condition 요소는 DB 엔진은 MySQL이 되고, DB 인스턴스 클래스는 db.t2.micro가 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 [조건 키 \(p. 966\)](#) 단원을 참조하십시오. 이 예제에서는 rds:DatabaseEngine 및 rds:DatabaseClass 조건을 지정합니다. rds:DatabaseEngine의 유효 조건 값에 대한 정보는 [CreateDBInstance](#)의 Engine 파라미터 아래 목록을 참조하십시오. rds:DatabaseClass의 유효 조건 값에 대한 정보는 [Aurora에 사용 가능한 모든 DB 인스턴스 클래스의 하드웨어 사양 \(p. 31\)](#) 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 IAM 사용 설명서에서 [Amazon RDS에서 정의한 작업](#) 단원을 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 이용한 액세스 관리 \(p. 963\)](#) 참조).

AWS CLI 또는 Amazon RDS API만 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요가 없습니다.

다음 정책은 루트 AWS 계정의 모든 Amazon Aurora 리소스에 대한 모든 액세스 권한을 부여합니다.

AmazonRDSFullAccess

사용자가 모든 RDS 리소스에서 Describe 작업을 수행할 수 있도록 허용

다음 권한 정책은 사용자에게 `Describe`로 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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowRDSDescribe",  
            "Effect": "Allow",  
            "Action": "rds:Describe*",  
            "Resource": "*"  
        }  
    ]  
}
```

사용자가 지정된 DB 파라미터 및 보안 그룹을 사용하는 DB 인스턴스를 생성할 수 있도록 허용

다음 권한 정책은 사용자가 `mysql-production` DB 파라미터 그룹 및 `db-production` DB 보안 그룹을 사용해야 하는 DB 인스턴스만 생성할 수 있도록 허용하는 권한을 부여합니다.

{

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowMySQLProductionCreate",  
            "Effect": "Allow",  
            "Action": "rds:CreateDBInstance",  
            "Resource": [  
                "arn:aws:rds:us-west-2:123456789012:pg:mysql-production",  
                "arn:aws:rds:us-west-2:123456789012:secgrp:db-production"  
            ]  
        }  
    ]  
}
```

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` 태그가 있는 DB 인스턴스에서 `ModifyDBInstance` 및 `CreateDBSnapshot` API를 수행할 수 있는 권한을 허용합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowDevTestCreate",  
            "Effect": "Allow",  
            "Action": [  
                "rds:ModifyDBInstance",  
                "rds>CreateDBSnapshot"  
            ],  
            "Resource": "*",  
            "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.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "DenyDelete1",  
            "Effect": "Deny",  
            "Action": "rds>DeleteDBInstance",  
            "Resource": "arn:aws:rds:us-west-2:123456789012:db:my-mysql-instance"  
        }  
    ]  
}
```

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 m4.10xlarge를 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": "AllowAllOtherOperations",  
            "Effect": "Allow",  
            "Action": "rds:CreateDBInstance",  
            "Resource": "*",  
            "Condition": {  
                "StringNotEquals": {  
                    "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 stage인 태그를 값이 test, qa, production인 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 environment인 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
secgrp-tag	DB 보안 그룹
og-tag	DB 옵션 그룹
pg-tag	DB 파라미터 그룹

RDS 태그 식별자	적용 대상
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 environment이고 태그 값이 production인 DB 인스턴스에 적용됩니다.

```
"Condition": {"StringEquals": {"rds:db-tag/environment": ["production"]}}
```

태그 생성에 대한 자세한 내용은 [Amazon RDS 리소스에 태그 지정 \(p. 315\)](#) 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 IAM 사용 설명서에서 [Amazon RDS에서 정의한 작업](#) 단원을 참조하십시오.

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\(ARN\)을 사용한 작업 \(p. 319\)](#) 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 태그가 있는 DB 인스턴스에서 ModifyDBInstance 및 CreateDBSnapshot API를 수행할 수 있는 권한을 허용합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "AllowDevTestCreate",  
            "Effect": "Allow",  
            "Action": [  
                "rds:ModifyDBInstance",  
                "rds>CreateDBSnapshot"  
            ],  
            "Resource": "*",  
            "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": "*",  
            "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 devo와 동일한 stage라는 태그가 있거나 stage라는 태그가 없는 DB 인스턴스에서 API(AddTagsToResource 또는 RemoveTagsFromResource 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\(ARN\)을 사용한 작업 \(p. 319\)](#) 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"
            }
        }
    }
]
```

을 위한 IAM 데이터베이스 인증

AWS Identity and Access Management(IAM) 데이터베이스 인증을 사용하여 DB 클러스터에 인증할 수 있습니다. IAM 데이터베이스 인증은 AuroraMySQL 및 Aurora PostgreSQL에서 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.

IAM 데이터베이스 인증은 다음과 같은 이점이 있습니다.

- 데이터베이스를 오가는 네트워크 트래픽은 SSL(Secure Sockets Layer)을 통해 암호화됩니다.
- 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.

주제

- [IAM 데이터베이스 인증 방식의 가용성 \(p. 976\)](#)
- [IAM 데이터베이스 인증에 대한 MySQL 한도 \(p. 977\)](#)
- [IAM 데이터베이스 인증에 대한 PostgreSQL 한도 \(p. 977\)](#)
- [IAM 데이터베이스 인증의 활성화 및 비활성화 \(p. 977\)](#)
- [IAM 데이터베이스 액세스를 위한 IAM 정책 생성 및 사용 \(p. 979\)](#)
- [IAM 인증을 사용하여 데이터베이스 계정 생성 \(p. 982\)](#)
- [IAM 인증을 사용하여 DB 클러스터에 연결 \(p. 982\)](#)

IAM 데이터베이스 인증 방식의 가용성

IAM 데이터베이스 인증 방식은 다음과 같은 데이터베이스 엔진 및 인스턴스 클래스에서 사용할 수 있습니다.

- MySQL과 호환되는 Aurora 버전 1.10 이상. db.t2.small과 db.t3.small을 제외한 모든 DB 인스턴스 클래스가 지원됩니다.
- PostgreSQL과 호환되는 Aurora, PostgreSQL 버전 9.6.9 및 10.4 이상.

IAM 데이터베이스 인증에 대한 MySQL 한도

Aurora MySQL에 IAM 데이터베이스 인증을 사용할 경우 새로운 연결 수는 초당 최대 200개로 제한됩니다.

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. 추가 오버헤드로 인해 기존 연결까지 끊어질 수도 있습니다. Aurora MySQL의 최대 총 연결에 대한 자세한 정보는 [Aurora MySQL DB 인스턴스에 대한 최대 연결 \(p. 508\)](#) 단원을 참조하십시오.

현재, Aurora MySQL 병렬 쿼리는 IAM 데이터베이스 인증을 지원하지 않습니다.

MySQL 엔진을 사용할 경우 다음을 따르십시오.

- IAM 데이터베이스 인증 방식은 개인이 일시적으로 데이터베이스에 액세스하기 위한 메커니즘으로 사용하십시오.
- 재시도가 용이한 워크로드에서만 IAM 데이터베이스 인증 방식을 사용하십시오.
- 애플리케이션에 초당 200개 이상의 새 연결이 필요한 경우에는 IAM 데이터베이스 인증 방식을 사용하지 마십시오.

IAM 데이터베이스 인증에 대한 PostgreSQL 한도

PostgreSQL과 함께 IAM 데이터베이스 인증을 사용하는 경우 다음 제한 사항을 적어둡니다.

- 데이터베이스 클러스터에 대한 초당 최대 연결 수는 클러스터 유형 및 워크로드에 따라 제한할 수 있습니다.

IAM 데이터베이스 인증의 활성화 및 비활성화

DB 클러스터에서는 기본적으로 IAM 데이터베이스 인증이 비활성화되어 있습니다. IAM 데이터베이스 인증은 AWS Management 콘솔, AWS CLI 또는 API를 사용하여 활성화하거나 다시 비활성화할 수 있습니다.

콘솔

콘솔을 사용하여 IAM 인증을 통해 새 DB 클러스터를 생성하려면 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

각 생성 워크플로우에는 고급 설정 구성 페이지가 있으며, 여기서 IAM DB 인증을 활성화할 수 있습니다. 이 페이지의 [Database Options] 섹션에 있는 [Enable IAM DB Authentication]에서 [Yes]를 선택합니다.

기존 DB 클러스터에서 IAM 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

영향을 받는 모든 DB 인스턴스가 IAM 인증과 호환되는지 확인하십시오. 호환성 요구 사항은 [IAM 데이터베이스 인증 방식의 가용성 \(p. 976\)](#) 단원을 참조하십시오. Aurora DB 클러스터의 경우 클러스터의 모든 DB 인스턴스가 IAM과 호환되는 경우에만 IAM 인증을 활성화할 수 있습니다.

4. 수정을 선택합니다.

5. 데이터베이스 옵션 섹션의 IAM DB 인증에서 IAM DB 인증 또는 비활성화를 선택한 다음 계속을 선택합니다.
6. 변경 사항을 즉시 적용하려면 즉시 적용을 선택합니다.
7. 클러스터 수정을 선택합니다.

DB 클러스터를 복원하려면

1. <https://console.aws.amazon.com/rds/>에서 Amazon RDS 콘솔을 엽니다.
2. 탐색 창에서 [Snapshots]를 선택합니다.
3. 복원할 스냅샷을 선택한 다음 작업에서 스냅샷 복원을 선택합니다.
4. 설정 섹션에서 DB 인스턴스 ID에 DB 인스턴스의 ID를 입력합니다.
5. [Database options] 섹션의 [IAM DB authentication]에서 [Enable IAM DB authorization] 또는 [Disable]을 선택합니다.
6. [Restore DB Instance]를 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

영향을 받는 모든 DB 인스턴스가 IAM 인증과 호환되는지 확인하십시오. 호환성 요구 사항은 [IAM 데이터베이스 인증 방식의 가용성 \(p. 976\)](#) 단원을 참조하십시오. Aurora DB 클러스터의 경우 클러스터의 모든 DB 인스턴스가 IAM과 호환되는 경우에만 IAM 인증을 활성화할 수 있습니다.

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

영향을 받는 모든 DB 인스턴스가 IAM 인증과 호환되는지 확인하십시오. 호환성 요구 사항은 [IAM 데이터베이스 인증 방식의 가용성 \(p. 976\)](#) 단원을 참조하십시오. Aurora DB 클러스터의 경우 클러스터의 모든 DB 인스턴스가 IAM과 호환되는 경우에만 IAM 인증을 활성화할 수 있습니다.

DB 클러스터를 복원하는 경우에는 다음 API 작업 중 하나를 사용하십시오.

- [RestoreDBClusterToPointInTime](#)
- [RestoreDBClusterFromSnapshot](#)

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

IAM 사용자 또는 역할이 DB 클러스터에 연결할 수 있도록 허용하려면 IAM 정책을 생성해야 합니다. 그런 다음 정책을 IAM 사용자 또는 역할에 연결해야 합니다.

Note

IAM 정책에 대한 자세한 정보는 [Amazon Aurora의 Identity and Access Management\(IAM\) \(p. 960\)](#) 단원을 참조하십시오.

다음은 IAM 사용자가 IAM 데이터베이스 인증 방식을 사용해 DB 클러스터에 연결할 수 있도록 허용하는 정책 예제입니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "rds-db:connect"  
            ],  
            "Resource": [  
                "arn:aws:rds-db:us-east-2:1234567890:dbuser:cluster-ABCDEFGHIJKLM01234/db_user"  
            ]  
        }  
    ]  
}
```

Important

IAM 관리자 사용자는 IAM 정책에서 명시적 권한 없이도 DB 클러스터에 액세스할 수 있습니다. [IAM 사용자 생성 \(p. 69\)](#)의 예제에서는 IAM 관리자 사용자를 생성합니다. 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.

현재 IAM 콘솔에는 `rds-db:connect` 작업이 포함된 정책에 대한 오류가 표시됩니다. 이 오류는 무시할 수 있습니다.

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:connect를 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 DB 클러스터의 AWS 리전입니다. 정책 예제에서 사용되는 AWS 리전은 us-east-2입니다.
- **account-id**은 DB 클러스터의 AWS 계정 번호입니다. 정책 예제에서 사용되는 계정 번호는 1234567890입니다.
- **DbClusterResourceId**는 DB 클러스터의 식별자입니다. 이 식별자는 AWS 리전에 고유하며, 절대로 바꾸지 않습니다. 정책 예제에서 사용되는 식별자는 cluster-ABCDEFGHIJKLM01234입니다.

Amazon Aurora용 AWS Management 콘솔에서 DB 클러스터 리소스 ID를 찾으려면 DB 클러스터를 선택하여 세부 정보를 확인하십시오. 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]"
```

- **db-user-name**은 IAM 인증과 연결할 데이터베이스 계정 이름입니다. 정책 예제에서 사용되는 데이터베이스 계정은 db_user입니다.

다른 ARN을 구성하여 다양한 액세스 패턴을 지원할 수 있습니다. 다음 정책에서는 DB 클러스터에서 서로 다른 데이터베이스 계정 2개에 대한 액세스를 허용합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "rds-db:connect"  
            ],  
            "Resource": [  
                "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-ABCDEFGHIJKLM01234/  
jane_doe",  
                "arn:aws:rds-db:us-east-2:123456789012:dbuser:cluster-ABCDEFGHIJKLM01234/  
mary_roe"  
            ]  
        }  
    ]  
}
```

다음 정책에서는 "*" 문자를 사용하여 특정 AWS 계정과 AWS 리전의 모든 DB 클러스터 및 데이터베이스 계정을 일치시킵니다.

```
{
```

```
"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.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "rds-db:connect"
            ],
            "Resource": [
                "arn:aws:rds-db:us-east-2:123456789012:dbuser:*/jane_doe"
            ]
        }
    ]
}
```

IAM 사용자 또는 역할은 데이터베이스 사용자가 액세스할 수 있는 데이터베이스에만 액세스할 수 있습니다. 예를 들어 DB 클러스터에 이름이 `dev`인 데이터베이스와 `test`인 데이터베이스가 있다고 가정하겠습니다. 이 때 데이터베이스 사용자인 `jane_doe`가 `dev`에 대한 액세스 권한만 가지고 있다면 사용자 `jane_doe`와 함께 해당 DB 클러스터에 액세스하는 IAM 사용자 또는 역할도 `dev` 액세스 권한만 갖게 됩니다. 이러한 액세스 제한은 테이블, 뷰 등 다른 데이터베이스 객체에 대해서도 똑같이 적용됩니다.

IAM 정책과 IAM 사용자 또는 역할의 연결

데이터베이스 인증을 위한 IAM 정책을 생성하였으면 이제 정책을 IAM 사용자 또는 역할에 연결해야 합니다. 이번 주제에 대한 자습서는 IAM 사용자 안내서에서 [첫 번째 고객 관리형 정책 만들기 및 연결](#) 단원을 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 IAM 사용자를 얻게 될 것입니다.

Note

여러 IAM 사용자 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
```

IAM 사용자 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 IAM 사용자를 제거할 경우에는 DROP USER 문으로 데이터베이스 계정도 제거해야 합니다.

PostgreSQL에서 IAM 인증 사용

PostgreSQL에 IAM 인증을 사용하려면 DB 클러스터에 연결한 후 데이터베이스 사용자를 만들고 다음 예제와 같이 사용자에게 rds_iam 역할을 부여합니다.

```
CREATE USER db_userx;
GRANT rds_iam TO db_userx;
```

MySQL에서 IAM 인증 사용

MySQL에서는 AWS에서 제공하는 플러그인인 AWSAuthenticationPlugin에서 인증을 처리합니다. 이 플러그인은 IAM과 연동하여 IAM 사용자를 인증합니다. 다음 예제와 같이, DB 클러스터에 연결한 후 CREATE USER 문을 실행합니다.

```
CREATE USER jane_doe IDENTIFIED WITH AWSAuthenticationPlugin AS 'RDS';
```

IDENTIFIED WITH 절은 MySQL이 AWSAuthenticationPlugin을 사용하여 데이터베이스 계정(jane_doe)을 인증할 수 있도록 허용하는 데 필요합니다. AS 'RDS' 절은 인증 방법을 나타냅니다. 지정한 데이터베이스 계정과 IAM 사용자 또는 역할의 이름이 동일해야 합니다. 이 예에서 데이터베이스 계정과 IAM 사용자 또는 역할의 이름은 모두 jane_doe로 동일해야 합니다.

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 구성 사용하고 있는지, 그리고 DB 클러스터에서 IAM 데이터베이스 인증이 활성화되어 있는지 확인하십시오. 자세한 내용은 [IAM 데이터베이스 인증 방식의 가용성 \(p. 976\)](#) 및 [IAM 데이터베이스 인증의 활성화 및 비활성화 \(p. 977\)](#) 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.

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 General Reference의 [서명 버전 4 서명 프로세스](#) 참조) AWS CLI와 AWS SDK for Java는 생성되는 각 토큰에 자동으로 설명할 수 있습니다.

AWS Lambda 같은 다른 AWS 서비스에서 Amazon Aurora에 연결할 때 인증 토큰을 사용할 수 있습니다. 토큰을 사용하면 코드에 암호를 넣지 않아도 됩니다. 그 밖에 AWS SDK for Java를 사용하여 인증 토큰을 프로그래밍 방식으로 생성하고 프로그래밍 방식으로 서명하는 방법도 있습니다.

IAM 인증 토큰에 서명까지 마쳤으면 이제 Aurora DB 클러스터에 연결할 수 있습니다. 다음 단원에서는 명령 줄 도구 또는 AWS SDK for Java를 사용하여 연결하는 방법에 대해 알아보겠습니다.

자세한 내용은 [Use IAM authentication to connect with SQL Workbench/J to Amazon Aurora MySQL or Amazon RDS for MySQL](#) 단원을 참조하십시오.

주제

- [명령줄: AWS CLI 및 mysql Client에서 IAM 인증을 사용하여 DB 클러스터에 연결 \(p. 983\)](#)
- [명령줄: AWS CLI 및 psql Client에서 IAM 인증을 사용하여 DB 클러스터에 연결 \(p. 984\)](#)
- [IAM 인증 및 AWS SDK for Java를 사용하여 DB 클러스터에 연결 \(p. 985\)](#)

명령줄: AWS CLI 및 mysql Client에서 IAM 인증을 사용하여 DB 클러스터에 연결

아래 설명과 같이 AWS CLI 및 mysql 명령줄 도구를 사용하여 Aurora DB 클러스터에 연결할 수 있습니다.

주제

- [IAM 인증 토큰 생성 \(p. 983\)](#)
- [DB 클러스터에 연결 \(p. 983\)](#)

IAM 인증 토큰 생성

다음은 AWS CLI를 사용하여 서명이 되어 있는 인증 토큰을 생성하는 방법을 설명한 예제입니다.

```
aws rds generate-db-auth-token \
--hostname rdsmysql.cdgmuqiadpid.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 이름입니다.
- --region – DB 클러스터가 실행되는 AWS 리전입니다.
- --username – 액세스할 데이터베이스 계정입니다.

토큰에서 가장 앞의 일부 문자는 다음과 같은 모습입니다.

```
rdsmysql.cdgmuqiadpid.us-west-2.rds.amazonaws.com:3306/?Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

DB 클러스터에 연결

일반적인 연결 형식은 다음과 같습니다.

```
mysql --host=hostName --port=portNumber --ssl-ca=[full path]rds-combined-ca-bundle.pem --enable-cleartext-plugin --user=userName --password=authToken
```

파라미터는 다음과 같습니다.

- --host – 액세스할 DB 클러스터의 호스트 이름입니다.
- --port – DB 클러스터에 연결할 때 사용할 포트 이름입니다.

- **--ssl-ca** – 퍼블릭 키를 포함하는 SSL 인증서 파일입니다. 자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#) 단원을 참조하십시오.
- **--enable-cleartext-plugin** – 현재 연결에서 AWSAuthenticationPlugin을 사용하도록 지정하는 값입니다.
- **--user** – 액세스할 데이터베이스 계정입니다.
- **--password** – 서명이 되어 있는 IAM 인증 토큰입니다.

인증 토큰은 수백 자의 문자로 구성됩니다. 그렇기 때문에 명령줄에서는 다루기 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.

```
RDSHOST="rdsmysql.cdgmuqiadpid.us-west-2.rds.amazonaws.com"
TOKEN=$(aws rds generate-db-auth-token --hostname $RDSHOST --port 3306 --region us-west-2
--username jane_doe )

mysql --host=$RDSHOST --port=3306 --ssl-ca=/sample_dir/rds-combined-ca-bundle.pem --enable-
cleartext-plugin --user=jane_doe --password=$TOKEN
```

AWSAuthenticationPlugin을 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
|
| ...          | ...
+-----+
```

명령줄: AWS CLI 및 psql Client에서 IAM 인증을 사용하여 DB 클러스터에 연결

아래 설명과 같이 AWS CLI 및 psql 명령줄 도구를 사용하여 Aurora PostgreSQL DB 클러스터에 연결할 수 있습니다.

주제

- [IAM 인증 토큰 생성 \(p. 984\)](#)
- [Aurora PostgreSQL 클러스터에 연결 \(p. 985\)](#)

IAM 인증 토큰 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 `generated-db-auth-token` 명령을 사용하여 서명된 인증 토큰을 받고 이를 PGPASSWORD 환경 변수에 저장하는 방법을 보여 줍니다.

```
export RDSHOST="mypostgres-cluster.cluster-cdgmuiqiadpid.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 이름입니다.
- `--region` – DB 클러스터가 실행되는 AWS 리전입니다.
- `--username` – 액세스할 데이터베이스 계정입니다.

생성된 토큰에서 처음 몇 글자는 다음과 같은 모습입니다.

```
mypostgres-cluster.cluster-cdgmuiqiadpid.us-west-2.rds.amazonaws.com:5432/?  
Action=connect&DBUser=jane_doe&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Expires=900...
```

Aurora PostgreSQL 클러스터에 연결

`psql`을 사용한 일반적인 연결 형식은 다음과 같습니다.

```
psql "host=hostName port=portNumber sslmode=verify-full sslrootcert=certificateFile
dbname=DBName user=userName"
```

파라미터는 다음과 같습니다.

- `host` – 액세스할 DB 클러스터(클러스터 엔드포인트)의 호스트 이름입니다.
- `port` – DB 클러스터에 연결할 때 사용할 포트 이름입니다.
- `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 파일입니다. 자세한 내용은 [PostgreSQL DB 인스턴스와 함께 SSL 사용을 참조하십시오.](#)
- `dbname` – 액세스할 데이터베이스 계정입니다.
- `user` – 액세스할 데이터베이스 계정입니다.

다음 예제는 `psql`을 사용하여 연결하는 방법을 보여줍니다. 이 예제에서 `psql`은 이전 섹션에서 토큰이 생성될 때 설정한 환경 변수 `PGPASSWORD`를 사용합니다.

```
psql "host=$RDSHOST port=5432 sslmode=verify-full sslrootcert=/sample_dir/rds-combined-ca-
bundle.pem dbname=DBName user=jane_doe"
```

IAM 인증 및 AWS SDK for Java를 사용하여 DB 클러스터에 연결

아래 설명과 같이 AWS SDK for Java를 사용하여 명령줄에서 Aurora MySQL 또는 Aurora PostgreSQL DB 클러스터로 연결할 수 있습니다.

주제

- [IAM 인증 토큰 생성 \(p. 986\)](#)
- [IAM 인증 토큰 수동 구성 \(p. 986\)](#)
- [DB 클러스터에 연결 \(p. 989\)](#)

IAM 인증 토큰 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 체인](#)에서 찾은 첫 번째 AWS 액세스 키와 보안 키를 사용합니다. AWS 액세스 키에 대한 자세한 내용은 [IAM 사용자를 위한 액세스 키 관리](#) 단원을 참조하십시오.

`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.cdgmuqiadpid.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 `RdsIamAuthTokenGenerator`를 사용하는 것입니다. 이 클래스는 인증 토큰을 생성한 후 AWS 서명 버전 4를 사용해 서명까지 마칩니다. 자세한 내용은 AWS General Reference의 [서명 버전 4 서명 프로세스](#) 단원을 참조하십시오.

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.cdgmuiadpid.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'HHmmssZ").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.toHexString(calculateSignature(stringToSign,
new SigningKey(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 date
should be in format YYYYMMDDTHHMMSSZ
public static String createCanonicalString(String user, String accessKey, String date,
String date, 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", date);
    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.toHexString(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 date, String canonicalRequest,
String accessKey, String date, String region) throws Exception {
    String credentialScope = date + "/" + region + "/" + serviceName + "/aws4_request";
    return algorithm + '\n' + date + '\n' + credentialScope + '\n' +
BinaryUtils.toHexString(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 생성한 다음 이 토큰을 사용하여 MySQL을 실행하는 클러스터에 연결하는 방법을 보여주는 코드 예제입니다.

이 코드 예제를 실행하려면 [AWS SDK for Java](#)가 필요하며 이것은 AWS에서 받을 수 있습니다. 또한 다음이 필요합니다.

- MySQL Connector/J. 이 코드 예제는 `mysql-connector-java-5.1.33-bin.jar`을 사용하여 테스트되었습니다.
- 한 AWS 리전에 고유한 Amazon Aurora 중간 인증서입니다. (자세한 내용은 [SSL/TLS를 사용하여 DB 클러스터에 대한 연결 암호화 \(p. 947\)](#) 단원을 참조하십시오.) 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 – 한 AWS 리전에 특정한 Amazon Aurora용 SSL 인증서입니다.

AWS 리전 인증서를 다운로드하는 방법은 [중간 인증서 \(p. 948\)](#) 단원을 참조하십시오. SSL 인증서는 예제 실행 시 클래스 로더가 인증서를 찾을 수 있도록 이 Java 프로그램 파일과 동일한 디렉터리에 설치합니다.

다음은 [기본 자격 증명 공급자 체인](#)에서 AWS 자격 증명을 가져오는 코드 예제입니다.

```
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.cdgmuqiadpid.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-2015-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%2Fcna-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");
}

}
```

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 (p. 993)
- iam:PassRole을 수행할 권한이 없음 (p. 993)
- 액세스 키를 보기 원함 (p. 993)
- 관리자인데, 다른 사용자가 Aurora에 액세스할 수 있기를 원함 (p. 994)
- AWS 계정 외부의 사람이 Aurora 리소스에 액세스할 수 있기를 원함 (p. 994)

Aurora에서 작업을 수행할 권한이 없음

AWS Management 콘솔에서 작업을 수행할 권한이 없다는 메시지가 나타나는 경우 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 제공한 사람입니다.

다음 예제 오류는 mateojackson IAM 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 rds:GetWidget 작업을 사용하여 *my-example-widget* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

iam:PassRole을 수행할 권한이 없음

iam:PassRole 작업을 수행할 권한이 없다는 오류가 수신되면 관리자에게 문의하여 도움을 받아야 합니다. 관리자는 사용자 이름과 암호를 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 marymajor라는 IAM 사용자가 콘솔을 사용하여 Aurora에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 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 작업을 수행하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

액세스 키를 보기 원함

IAM 사용자 액세스 키를 생성한 후에는 언제든지 액세스 키 ID를 볼 수 있습니다. 하지만 보안 액세스 키는 다시 볼 수 없습니다. 보안 액세스 키를 잊어버린 경우 새로운 액세스 키 페어를 생성해야 합니다.

액세스 키는 액세스 키 ID(예: AKIAIOSFODNN7EXAMPLE)와 보안 액세스 키(예: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY)의 2가지 부분으로 구성됩니다. 사용자 이름 및 암호와 같이 액세스 키

ID와 보안 액세스 키를 함께 사용하여 요청을 인증해야 합니다. 사용자 이름과 암호를 관리하는 것처럼 안전하게 액세스 키를 관리합니다.

Important

[정식 사용자 ID를 찾는 데](#) 도움이 된다고 하더라도 액세스 키를 제3자에게 제공하지 마십시오. 이로 인해 다른 사람에게 계정에 대한 영구 액세스를 제공하게 될 수 있습니다.

액세스 키 페어를 생성할 때는 액세스 키 ID와 보안 액세스 키를 안전한 위치에 저장하라는 메시지가 나타납니다. 보안 액세스 키는 생성할 때만 사용할 수 있습니다. 하지만 보안 액세스 키를 잃어버린 경우 새로운 액세스 키를 IAM 사용자에게 추가할 수 있습니다. 최대 두 개의 액세스 키를 가질 수 있습니다. 이미 두 개가 있는 경우 새로 생성하려면 먼저 키 페어 하나를 삭제해야 합니다. 지침을 보려면 IAM 사용 설명서의 [액세스 키 관리](#)를 참조하십시오.

관리자인데, 다른 사용자가 Aurora에 액세스할 수 있기를 원함

다른 사용자가 Aurora에 액세스하도록 하려면 액세스 권한이 필요한 사용자 또는 애플리케이션에 대한 IAM 개체(사용자 또는 역할)를 만들어야 합니다. 다른 사용자들은 해당 엔터티에 대한 자격 증명을 사용해 AWS에 액세스합니다. 그런 다음 Aurora에서 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다.

즉시 시작하려면 IAM 사용 설명서의 [첫 번째 IAM 위임 사용자 및 그룹 생성](#)을 참조하십시오.

AWS 계정 외부의 사람이 Aurora 리소스에 액세스할 수 있기를 원함

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스하는 데 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 ACL(액세스 제어 목록)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 방식](#) (p. 964) 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 제공](#)을 참조하십시오.
- 리소스에 대한 액세스 권한을 제3자 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 [제3자가 소유한 AWS 계정에게 액세스 권한 제공](#)을 참조하십시오.
- 자격 증명 연동을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [외부에서 인증된 사용자에게 액세스 권한 제공\(자격 증명 연동\)](#)을 참조하십시오.
- 교차 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 [IAM 역할과 리소스 기반 정책의 차이](#)를 참조하십시오.

Kerberos 인증

Amazon Aurora에서는 Kerberos 및 Microsoft Active Directory를 사용하여 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 Aurora의 지원은 데이터베이스 사용자에게 SSO(Single Sign-On) 및 중앙 집중식 인증의 이점을 제공합니다. 사용자 자격 증명을 Active Directory에 보관할 수 있습니다. Active Directory는 여러 DB 인스턴스에 대한 자격 증명을 보관하고 관리할 수 있는 중앙 집중식 공간을 제공합니다.

데이터베이스 사용자가 두 가지 방법으로 DB 인스턴스에 대해 인증하도록 할 수 있습니다. Microsoft Active Directory용 AWS 디렉터리 서비스(Enterprise Edition) 또는 온프레미스 Active Directory에 저장된 자격 증명을 사용할 수 있습니다.

현재 Aurora는 PostgreSQL DB 클러스터에 대한 Kerberos 인증을 지원합니다. 자세한 내용은 [Aurora PostgreSQL과 함께 Kerberos 인증 사용 \(p. 885\)](#) 단원을 참조하십시오.

Amazon Aurora의 로깅 및 모니터링

모니터링은 Amazon Aurora와 사용자 AWS 솔루션의 안정성, 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 정책에 알림이 전송됩니다. CloudWatch 경보는 특정 상태에 있기 때문에 작업을 호출하지 않습니다. 대신, 상태가 변경되어 지정된 기간 동안 유지되어야 합니다. 자세한 내용은 [Amazon CloudWatch로 모니터링 \(p. 329\)](#) 단원을 참조하십시오.

AWS CloudTrail 로그

CloudTrail은 Amazon Aurora의 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공합니다. CloudTrail은 콘솔의 호출 및 Amazon RDS API 작업에 대한 코드 호출의 호출 등 Amazon Aurora에 대한 모든 API 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 내용은 [AWS CloudTrail을 사용하여 Amazon RDS API 호출 로깅 \(p. 459\)](#) 단원을 참조하십시오.

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 Enhanced Monitoring JSON 출력을 사용할 수 있습니다. 자세한 내용은 [확장 모니터링 \(p. 358\)](#) 단원을 참조하십시오.

Amazon RDS 성능 개선 도우미

성능 개선 도우미(Performance Insights)는 기존 Amazon Aurora 모니터링 기능을 확장한 것으로서 데이터베이스 성능을 표시하여 성능 문제를 분석하는 데 효과적입니다. 성능 개선 도우미 대시보드가 데이터베이스 부하를 시작화하여 대기 시간, SQL 문, 호스트 또는 사용자를 기준으로 부하를 필터링합니다. 자세한 내용은 [Amazon RDS 성능 개선 도우미 사용 \(p. 365\)](#) 단원을 참조하십시오.

데이터베이스 로그

AWS Management 콘솔, AWS CLI 또는 RDS API를 사용하여 데이터베이스 로그를 보고 다운로드하고 조사할 수 있습니다. 자세한 내용은 [Amazon Aurora 데이터베이스 로그 파일 \(p. 449\)](#) 단원을 참조하십시오.

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 내용은 [Amazon Aurora 권장 사항 사용 \(p. 408\)](#) 단원을 참조하십시오.

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 알

림 메시지 형식에 따라 이메일, 문자 또는 HTTP 엔드포인트 호출 등이 될 수 있습니다. 자세한 내용은 [Amazon RDS 이벤트 알림 서비스 사용 \(p. 429\)](#) 단원을 참조하십시오.

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 유튜 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

Aurora PostgreSQL에서는 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 요구를 충족할 수 있습니다. 자세한 내용은 [Aurora PostgreSQL에서 데이터베이스 활동 스트림 사용 \(p. 412\)](#) 단원을 참조하십시오.

Aurora 모니터링에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 모니터링 \(p. 327\)](#) 단원을 참조하십시오.

Amazon Aurora 규정 준수 확인

타사 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 범위 내 AWS 서비스](#) 단원을 참조하십시오. 일반 정보는 [AWS 규정 준수 프로그램](#)을 참조하십시오.

AWS Artifact를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 [AWS Artifact의 보고서 다운로드](#)를 참조하십시오.

Amazon Aurora 사용 시 규정 준수 책임은 데이터의 민감도, 조직의 규정 준수 목표 및 관련 법률과 규정에 따라 결정됩니다. AWS는 규정 준수에 도움이 되도록 다음 리소스를 제공합니다.

- [보안 및 규정 준수 빠른 시작 안내서](#) – 이 배포 가이드에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS 환경을 배포하기 위한 단계를 제공합니다.
- [HIPAA 보안 및 규정 준수 기술 백서 설계](#) – 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 만드는 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 서비스는 보안 업계 표준 및 모범 사례 준수 여부를 확인하는 데 도움이 되는 AWS 내 보안 상태에 대한 포괄적인 관점을 제공합니다.

Amazon Aurora의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전에서는 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 지연 시간, 높은 처리량 및 높은 종복성을 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, 내결합성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 [AWS 글로벌 인프라](#)를 참조하십시오.

AWS 글로벌 인프라 외에 Aurora도 데이터 복원성과 백업 요구 사항을 지원하는 여러 가지 기능을 제공합니다.

백업 및 복원

Aurora은 클러스터 볼륨을 자동으로 백업한 후 백업 보존 기간 동안 복원 데이터를 보관합니다. Aurora 백업은 연속식 또는 증분식으로 이루어지기 때문에 백업 보존 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 지나서 백업 파일을 보관하려면 클러스터 볼륨에서 데이터 스냅샷을 생성하는 방법도 있습니다. Aurora는 증분 복원 데이터를 백업 보존 기간이 끝날 때까지 보관합니다. 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 \(p. 268\)](#) 단원을 참조하십시오.

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 분산시킬 수 있습니다. DB 클러스터 볼륨은 DB 클러스터의 데이터 사본들로 구성됩니다. 하지만 DB 클러스터의 기본 DB 인스턴스 및 Aurora 복제본에는 클러스터 볼륨의 데이터가 단 하나의 논리 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 \(p. 55\)](#) 단원을 참조하십시오.

Failover

Aurora는 단일 AWS 리전에서 다중 가용 영역의 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 시스

템 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 [Aurora를 위한 고가용성 \(p. 36\)](#) 단원을 참조하십시오.

Amazon Aurora 내 인프라 보안

관리형 서비스인 Amazon RDS는 [Amazon Web Services: 보안 프로세스 개요](#) 백서에 설명된 AWS 글로벌 네트워크 보안 절차로 보호됩니다.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon Aurora에 액세스합니다. 클라이언트가 TLS(전송 계층 보안) 1.0을 지원해야 합니다. TLS 1.2 이상을 권장합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 PFS(전달 완전 보안, Perfect Forward Secrecy)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 [AWS Security Token Service\(AWS STS\)](#)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

또한 Aurora는 인프라 보안을 지원하는 기능을 제공합니다.

보안 그룹

보안 그룹은 DB 인스턴스에서 송수신되는 트래픽에 대한 액세스를 제어합니다. 기본적으로 DB 인스턴스에 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 인스턴스에 적용됩니다.

자세한 내용은 [보안 그룹을 통한 액세스 제어 \(p. 999\)](#) 단원을 참조하십시오.

퍼블릭 액세스 가능성

Amazon VPC 서비스 기반 Virtual Private Cloud(VPC)에서 DB 인스턴스를 시작할 때 해당 인스턴스의 퍼블릭 액세스를 켜거나 끌 수 있습니다. Public accessibility 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 인스턴스를 인터넷에서 숨기기 \(p. 1017\)](#) 단원을 참조하십시오.

Amazon Aurora 보안 모범 사례

AWS Identity and Access Management(IAM) 계정을 사용해 Amazon RDS API 작업, 특히 Amazon Aurora 리소스를 생성하거나, 수정하거나, 삭제하는 작업에 대한 액세스를 제어합니다. 이러한 리소스 중에는 DB 클러스터, 보안 그룹 및 파라미터 그룹이 있습니다. 또한 IAM을 사용해 DB 클러스터 백업 및 복구 같은 공통 관리 작업을 수행하는 작업을 제어합니다.

- Amazon Aurora 리소스를 관리하는 각 사용자에게 개별 IAM 계정을 할당합니다. Amazon Aurora 리소스 관리에 AWS 루트 자격 증명을 사용하지 마십시오. 본인을 포함하여 모두가 사용할 IAM 사용자를 만들어야 합니다.
- 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 보안 암호 교체](#) 단원을 참조하십시오.

IAM에 대한 자세한 내용은 [AWS Identity and Access Management](#) 단원을 참조하십시오. IAM 모범 사례에 대한 자세한 내용은 [IAM 모범 사례](#) 단원을 참조하십시오.

AWS Management 콘솔, 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.

보안 그룹을 통한 액세스 제어

보안 그룹은 DB 인스턴스에서 송수신되는 트래픽에 대한 액세스를 제어합니다. Aurora는 VPC 보안 그룹을 지원합니다.

VPC 보안 그룹

각 VPC 보안 그룹 규칙을 설정하면 특정 소스가 해당 VPC 보안 그룹과 연결되어 있는 VPC의 DB 인스턴스에 액세스할 수 있습니다. 소스는 주소 범위(예: 203.0.113.0/24) 또는 다른 VPC 보안 그룹일 수 있습니다. VPC 보안 그룹을 스스로 지정하면 소스 VPC 보안 그룹을 사용하는 모든 인스턴스(일반적으로 애플리케이션 서버)에서 수신 트래픽이 허용됩니다. VPC 보안 그룹에 인바운드와 아웃바운드 트래픽을 모두 제어하는 규칙이 있을 수 있지만, 아웃바운드 트래픽 규칙은 일반적으로 DB 인스턴스에 적용되지 않습니다. 아웃바운드 트래픽 규칙은 DB 인스턴스가 클라이언트로 작동하는 경우에만 적용됩니다. [Amazon EC2 API](#)를 사용하거나, 혹은 VPC 콘솔에서 보안 그룹 옵션을 선택하여 VPC 보안 그룹을 생성해야 합니다.

VPC의 인스턴스에 대한 액세스를 허용하는 VPC 보안 그룹과 관련된 규칙을 생성할 때 그 규칙이 액세스를 허용하는 주소들의 각 범위에 대해 포트를 지정해야 합니다. 예를 들어, VPC의 인스턴스에 대한 SSH 액세스를 활성화하고 싶다면 지정된 주소 범위와 관련해 TCP 포트 22에 대한 액세스를 허용하는 규칙을 생성해야 합니다.

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 생성하면 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.

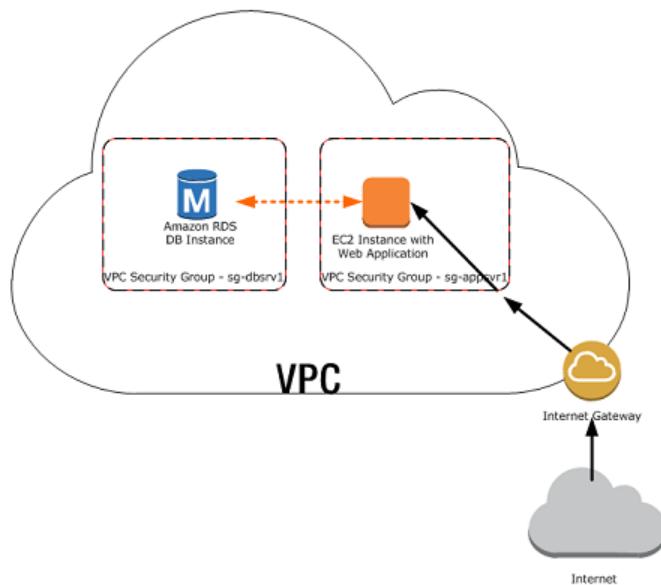
VPC 보안 그룹에 관한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 [보안 그룹](#)을 참조하십시오.

보안 그룹 시나리오

VPC에서 DB 인스턴스를 사용하는 일반적인 사례는 동일한 VPC의 Amazon EC2 인스턴스에서 실행이며 VPC 외부의 클라이언트 애플리케이션에서 액세스한 애플리케이션을 사용하여 데이터를 공유하는 것입니다. 이러한 시나리오에서는 AWS Management 콘솔의 RDS 및 VPC 페이지를 사용하거나, 혹은 RDS 및 EC2 API 작업을 사용하여 필요한 인스턴스와 보안 그룹을 생성합니다.

1. VPC 보안 그룹(예: sg-appsrv1)을 생성하고 클라이언트 애플리케이션의 IP 주소를 스스로 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-appsrv1)에 EC2 인스턴스를 추가합니다. VPC의 EC2 인스턴스는 DB 인스턴스와 VPC 보안 그룹을 공유합니다.
3. 두 번째 VPC 보안 그룹(예: sg-dbsrv1)을 생성하고 1단계에서 만든 VPC 보안 그룹(sg-appsrv1)을 스스로 지정해 새 규칙을 생성합니다.
4. 새 DB 인스턴스를 생성하고 이전 단계에서 생성한 VPC 보안 그룹(sg-dbsrv1)에 DB 인스턴스를 추가합니다. DB 인스턴스를 생성할 때 3단계에서 생성한 VPC 보안 그룹(sg-dbsrv1) 규칙에 대해 지정한 것과 동일한 보안 포트 번호를 사용하십시오.

다음 다이어그램은 이 시나리오를 보여 줍니다.



VPC 사용에 대한 자세한 내용은 [Amazon Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#) 단원을 참조하십시오.

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. \(p. 72\)](#) 및 Amazon Virtual Private Cloud 사용 설명서의 [보안 그룹](#)을 참조하십시오.

보안 그룹과 DB 인스턴스의 연결

보안 그룹은 RDS 콘솔의 수정 옵션을 사용하거나, `ModifyDBInstance` Amazon RDS API 또는 `modify-db-instance` AWS CLI 명령을 사용해 DB 인스턴스와 연결할 수 있습니다.

DB 클러스터에 있는 DB 인스턴스를 수정하는 것에 대한 자세한 내용은 [DB 클러스터에서 DB 인스턴스 수정 \(p. 194\)](#) 단원을 참조하십시오. DB 인스턴스를 DB 스냅샷에서 복원할 때 보안 그룹에서 고려해야 할 사항은 [보안 그룹 고려 사항 \(p. 273\)](#) 단원을 참조하십시오.

보안 그룹과 DB 클러스터의 연결

보안 그룹은 RDS 콘솔의 클러스터 수정 옵션을 사용하거나, `ModifyDBCluster` Amazon RDS API 또는 `modify-db-cluster` AWS CLI 명령을 사용해 DB 클러스터와 연결할 수 있습니다.

DB 클러스터 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.

마스터 사용자 계정 권한

새로운 DB 클러스터를 생성할 때 사용되는 기본 마스터 사용자는 해당 DB 클러스터에 대한 특정 권한을 갖습니다. 다음 표에는 마스터 사용자가 각 데이터베이스 엔진에 대해 갖는 권한 및 데이터베이스 역할이 나와 있습니다.

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 클러스터 수정에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 수정 \(p. 193\)](#) 단원을 참조하십시오.

데이터베이스 엔진	시스템 권한	데이터베이스 역할
Amazon Aurora MySQL	CREATE, DROP, GRANT OPTION, REFERENCES, EVENT, ALTER, DELETE, INDEX, INSERT, SELECT, UPDATE, CREATE TEMPORARY TABLES, LOCK TABLES, TRIGGER, CREATE VIEW, SHOW VIEW, LOAD FROM S3, SELECT INTO S3, ALTER ROUTINE, CREATE ROUTINE, EXECUTE, CREATE USER, PROCESS, SHOW DATABASES , RELOAD, REPLICATION CLIENT, REPLICATION SLAVE	—
Amazon Aurora PostgreSQL	LOGIN, NOSUPERUSER, INHERIT, CREATEDB, CREATEROLE, NOREPLICATION, VALID UNTIL 'infinity'	RDS_SUPERUSER

Amazon Aurora에 서비스 연결 역할 사용

Amazon Aurora은 AWS Identity and Access Management(IAM) 서비스 연결 역할을 사용합니다. 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.

서비스 연결 역할을 통해 Amazon Aurora 사용이 쉬워지는데 필요한 권한을 수동으로 추가할 필요가 없기 때문입니다. Amazon Aurora에서 서비스 연결 역할 권한을 정의하므로, 달리 정의되지 않은 한 Amazon Aurora에서만 해당 역할을 맡을 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며, 이 권한 정책은 다른 IAM 개체에 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 정보는 [IAM으로 작업하는 AWS 서비스](#)를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾으십시오. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

Amazon Aurora에 대한 서비스 연결 역할 권한

Amazon Aurora에서는 AWSServiceRoleForRDS – to allow Amazon RDS to call AWS services on behalf of your DB clusters인 서비스 연결 역할을 사용합니다.

AWSServiceRoleForRDS 서비스 연결 역할은 역할을 위임하기 위해 다음 서비스를 신뢰합니다.

- rds.amazonaws.com

역할 권한 정책은 Amazon Aurora가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "ec2:AuthorizeSecurityGroupIngress",  
                "ec2>CreateNetworkInterface",  
                "ec2>CreateSecurityGroup",  
                "ec2>DeleteNetworkInterface",  
                "ec2>DeleteSecurityGroup",  
                "ec2:DescribeAvailabilityZones",  
                "ec2:DescribeInternetGateways",  
                "ec2:DescribeSecurityGroups",  
                "ec2:DescribeSubnets",  
                "ec2:DescribeVpcAttribute",  
                "ec2:DescribeVpcs",  
                "ec2:ModifyNetworkInterfaceAttribute",  
                "ec2:ModifyVpcEndpoint",  
                "ec2:RevokeSecurityGroupIngress",  
                "ec2>CreateVpcEndpoint",  
                "ec2:DescribeVpcEndpoints",  
                "ec2:DeleteVpcEndpoints",  
                "ec2:AssignPrivateIpAddresses",  
                "ec2:UnassignPrivateIpAddresses"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "sns:Publish"  
            ],  
            "Resource": "*"  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogGroup"  
            ],  
            "Resource": [  
                "arn:aws:logs:***:log-group:/aws/rds/*",  
                "arn:aws:logs:***:log-group:/aws/docdb/*",  
                "arn:aws:logs:***:log-group:/aws/neptune/*"  
            ]  
        },  
        {  
            "Effect": "Allow",  
            "Action": [  
                "logs>CreateLogStream",  
                "logs:PutLogEvents",  
            ]  
        }  
    ]  
}
```

```
        "logs:DescribeLogStreams"
    ],
    "Resource": [
        "arn:aws:logs:*:log-group:/aws/rds/*:log-stream:*",
        "arn:aws:logs:*:log-group:/aws/docdb/*:log-stream:*",
        "arn:aws:logs:*log-group:/aws/neptune/*:log-stream:*
```

]

{

"Effect": "Allow",
 "Action": [

```
        "kinesis>CreateStream",
        "kinesis:PutRecord",
        "kinesis:PutRecords",
        "kinesis:DescribeStream",
        "kinesis:SplitShard",
        "kinesis:MergeShards",
        "kinesis>DeleteStream",
        "kinesis:UpdateShardCount"
    ],
    "Resource": [
        "arn:aws:kinesis:stream/aws-rds-das-*"
    ]
}
```

}

}

Note

IAM 개체(사용자, 그룹, 역할 등)가 서비스 연결 역할을 생성, 편집 또는 삭제할 수 있도록 권한을 구성해야 합니다. 다음 오류 메시지가 표시되는 경우:

리소스를 만들 수 없습니다. 서비스 연결 역할을 생성할 권리 있는지 확인하십시오. 그렇지 않은 경우 기다렸다가 나중에 다시 시도하십시오.

다음 권리 활성화되어 있는지 확인하십시오.

```
{
    "Action": "iam>CreateServiceLinkedRole",
    "Effect": "Allow",
    "Resource": "arn:aws:iam::role/aws-service-role/rds.amazonaws.com/
AWSRoleForRDS",
    "Condition": {
        "StringLike": {
            "iam:AWSServiceName": "rds.amazonaws.com"
        }
    }
}
```

자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권리](#)를 참조하십시오.

Amazon Aurora에 대한 서비스 연결 역할 생성

서비스 연결 역할은 수동으로 생성할 필요가 없습니다. create a DB cluster 시 Amazon Aurora에서 서비스 연결 역할을 생성합니다.

Important

Amazon Aurora 서비스가 서비스 연결 역할을 지원하기 시작한 December 1, 2017 이전에 이 서비스를 사용 종이었다면 Amazon Aurora에서 사용자 계정에 AWSRoleForRDS 역할을 이미 생성했습니다. 자세한 내용은 [내 IAM 계정에 표시되는 새 역할](#)을 참조하십시오.

이 서비스 연결 역할을 삭제한 다음 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. create a DB cluster 시 Amazon Aurora에서 서비스 연결 역할을 다시 생성합니다.

Amazon Aurora에 대한 서비스 연결 역할 편집

Amazon Aurora에서는 AWSServiceRoleForRDS 서비스 연결 역할을 편집하도록 허용하지 않습니다. 서비스 연결 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 그러나 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 편집](#) 단원을 참조하십시오.

Amazon Aurora에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제할 것을 권합니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 개체가 없도록 합니다. 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 있는지 확인하고 역할에서 사용되는 리소스를 모두 제거해야 합니다.

IAM 콘솔에서 서비스 연결 역할에 활성 세션이 있는지 확인하려면

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 엽니다.
2. IAM 콘솔의 탐색 창에서 역할을 선택합니다. 그런 다음 AWSServiceRoleForRDS 역할의 이름(확인란 아님)을 선택합니다.
3. 선택한 역할의 요약 페이지에서 Access Advisor(액세스 관리자) 탭을 선택합니다.
4. [Access Advisor] 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 되지 않으며, 역할이 사용되고 있는 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. 삭제를 선택합니다.

클러스터를 삭제하려면(CLI)

AWS CLI Command Reference의 [delete-db-cluster](#) 참조.

클러스터를 삭제하려면(API)

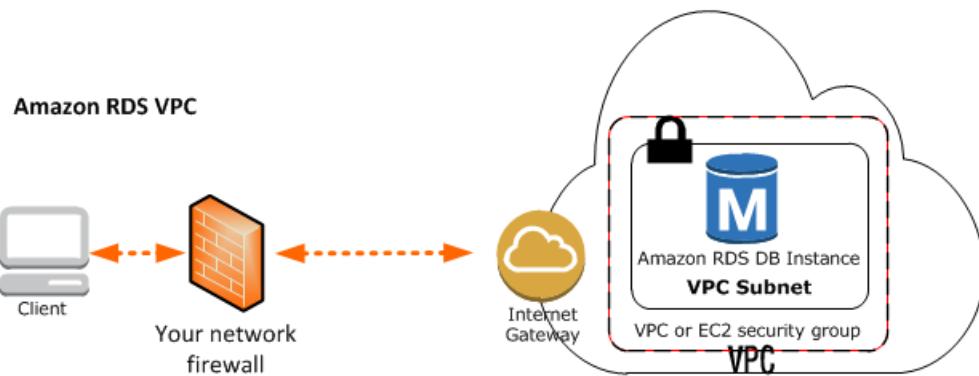
Amazon RDS API Reference의 [DeleteDBCluster](#) 참조.

IAM 콘솔, IAM CLI 또는 IAM API를 사용하여 AWSServiceRoleForRDS 서비스 연결 역할을 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스에 연결 역할 삭제 단원](#)을 참조하십시오.

Amazon Virtual Private Cloud VPC 및 Amazon Aurora

Amazon Virtual Private Cloud(Amazon VPC)에서는 Aurora DB 클러스터와 같은 AWS 리소스를 Virtual Private Cloud (VPC)로 시작할 수 있습니다.

Amazon VPC를 사용하는데 가상 네트워크 환경을 제어할 수 있는 경우: 자신의 IP 주소 범위를 선택하고 서브넷을 만들고 라우팅을 구성하고 제어 목록에 액세스할 수 있습니다. DB 인스턴스를 Amazon VPC에서 실행하는 데는 추가 비용이 들지 않습니다.



EC2-VPC 플랫폼만 지원하는 계정에는 기본 VPC가 있습니다. 달리 지정하지 않는 한 새 DB 인스턴스는 모두 기본 VPC에서 생성됩니다. 신규 Amazon Aurora 고객이거나, 전에 DB 인스턴스를 생성한 적이 전혀 없거나, 전에 사용한 적 없는 AWS 리전에서 DB 인스턴스를 생성하는 경우, 사용자는 틀림없이 EC2-VPC 플랫폼에 있고 기본 VPC가 있을 것입니다.

주제

- [Amazon Aurora에 사용할 VPC의 생성 방법 \(p. 1005\)](#)
- [VPC에서 DB 인스턴스에 액세스하는 시나리오 \(p. 1011\)](#)
- [VPC에서 DB 인스턴스를 사용한 작업 \(p. 1015\)](#)
- [자습서: DB 인스턴스에 사용할 Amazon VPC 생성 \(p. 1020\)](#)

이 문서에서는 Amazon Aurora DB 클러스터 관련 VPC 기능에 대해서만 설명합니다. Amazon VPC에 대한 자세한 내용은 [Amazon VPC 시작 안내서](#) 및 [Amazon VPC 사용 설명서](#)를 참조하십시오. 네트워크 주소 변환(NAT) 사용에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 [NAT 게이트웨이](#)를 참조하십시오.

Amazon Aurora에 사용할 VPC의 생성 방법

다음 섹션에서는 Amazon Aurora에서 사용할 VPC를 생성하는 방법에 대해 설명합니다.

Note

Amazon Aurora DB 클러스터에 연결하는 데 도움이 되는 자세한 안내는 [Aurora MySQL 데이터베이스 관리자 핸드북 - 연결 관리](#)에서 확인할 수 있습니다.

VPC 및 서브넷 생성

두 가용 영역에 걸쳐 있고 각 영역에 서브넷이 최소 1개 이상 있는 Virtual Private Cloud(VPC)에만 Amazon Aurora DB 클러스터를 생성할 수 있습니다. 또한 AWS 계정의 기본 VPC에 Aurora DB 클러스터를 생성하거나 사용자 정의 VPC를 따로 생성하여 사용하는 것도 가능합니다. 자세한 내용은 [Amazon Virtual Private Cloud VPC 및 Amazon Aurora \(p. 1005\)](#) 단원을 참조하십시오.

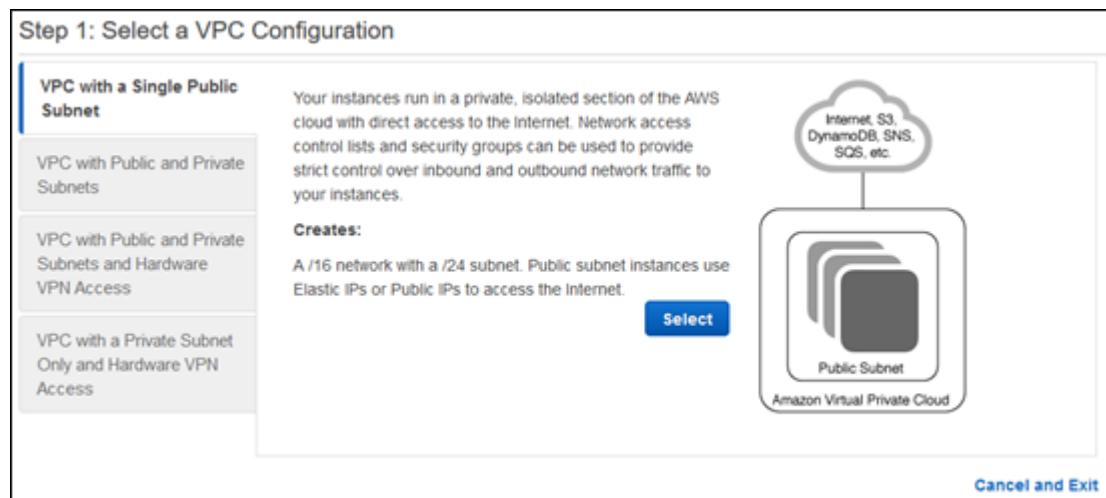
Amazon Aurora는 DB 클러스터에 사용할 VPC와 서브넷 그룹을 선택적으로 생성할 수 있습니다. 이렇게 하면 VPC를 한 번도 생성한 적이 없거나, 혹은 다른 VPC와 별도로 새로운 VPC를 생성할 때 유용하게 사용할 수 있습니다. Amazon Aurora에서 VPC와 서브넷 그룹을 생성할 때는 이 절차를 생략하고 [Aurora MySQL DB 클러스터 생성 \(p. 74\)](#) 단원을 참조하십시오.

Note

Aurora DB 클러스터에 사용할 모든 VPC 및 EC2 리소스는 [리전 및 가용 영역 \(p. 3\)](#)에 나열된 리전 중 하나에 있어야 합니다.

Aurora DB 클러스터에 사용할 VPC를 생성하는 방법

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
2. AWS Management 콘솔의 오른쪽 상단 모서리에서 VPC를 생성할 AWS 리전을 선택합니다. 이 예에서는 미국 동부(오하이오) 리전을 사용합니다.
3. 왼쪽 상단 모서리에서 [VPC Dashboard]를 선택합니다. VPC 마법사 시작을 선택하여 VPC 생성을 시작합니다.
4. VPC 생성 마법사에서 단일 퍼블릭 서브넷이 있는 VPC를 선택합니다. 선택을 선택합니다.



5. [Create VPC] 패널에서 다음과 같이 값을 설정합니다.

- IP CIDR block: 10.0.0.0/16
- VPC name: gs-cluster-vpc
- Public subnet: 10.0.0.0/24
- 가용 영역: us-east-1a
- Subnet name: gs-subnet1
- Enable DNS hostnames: Yes
- Hardware tenancy: Default

Step 2: VPC with a Single Public Subnet

IPv4 CIDR block*: (65531 IP addresses available)

IPv6 CIDR block: No IPv6 CIDR Block
 Amazon provided IPv6 CIDR block

VPC name:

Public subnet's IPv4 CIDR*: (251 IP addresses available)

Availability Zone*:

Subnet name:

You can add more subnets after AWS creates the VPC.

Service endpoints

Enable DNS hostnames*: Yes No

Hardware tenancy*:

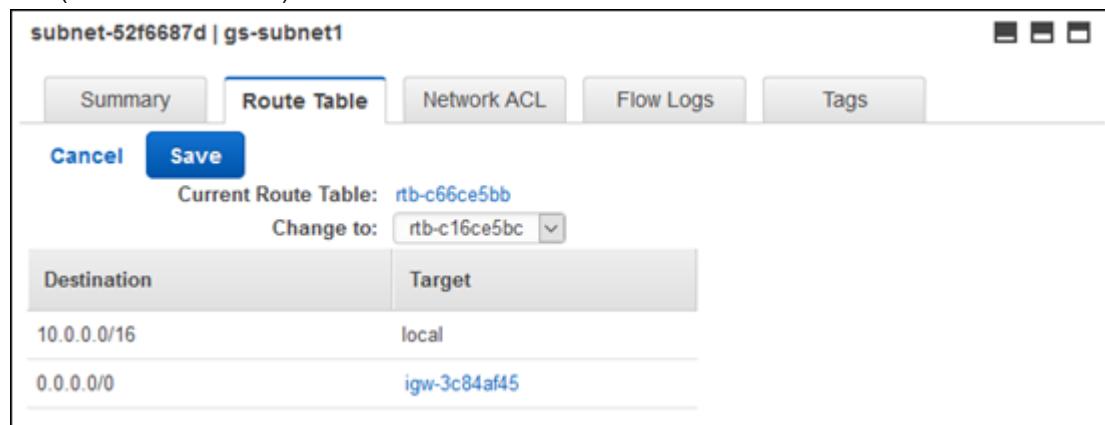
6. VPC 만들기를 선택합니다.
7. VPC 생성을 마치면 알림 페이지에서 닫기를 선택합니다.

서브넷을 추가로 생성하는 방법

1. 두 번째 서브넷을 VPC에 추가하려면 VPC 대시보드에서 서브넷과 서브넷 생성을 차례대로 선택합니다. Amazon Aurora DB 클러스터는 VPC 서브넷이 2개 이상 필요합니다.
2. 서브넷 생성 패널에서 다음과 같이 값을 설정합니다.
 - Name tag: gs-subnet2
 - VPC: 이전 단계에서 생성한 VPC를 선택합니다(예: vpc-a464d1c1 (10.0.0.0/16) | gs-cluster-vpc).
 - 가용 영역: us-east-1c
 - CIDR block: 10.0.1.0/24



3. 예, 생성을 선택합니다.
4. 두 번째로 생성한 서브넷이 첫 번째 서브넷과 동일한 라우팅 테이블을 사용하는지 확인하려면 VPC 대시보드에서 서브넷을 선택한 다음 VPC에 생성된 첫 번째 서브넷인 `gs-subnet1`을 선택합니다. 라우팅 테이블 탭을 선택하고 현재 라우팅 테이블(예: `rtb-c16ce5bc`)을 기록합니다.
5. 서브넷 목록에서 첫 번째 서브넷의 선택을 취소하고 두 번째 서브넷 `gs-subnet2`를 선택합니다. 라우팅 테이블 탭을 선택하고 편집을 선택합니다. Change to(다음과 같이 변경) 목록에서 이전 단계의 라우팅 테이블(예: `rtb-c16ce5bc`)을 선택합니다. 저장을 선택하여 선택 사항을 저장합니다.



보안 그룹 생성 및 인바운드 규칙 추가

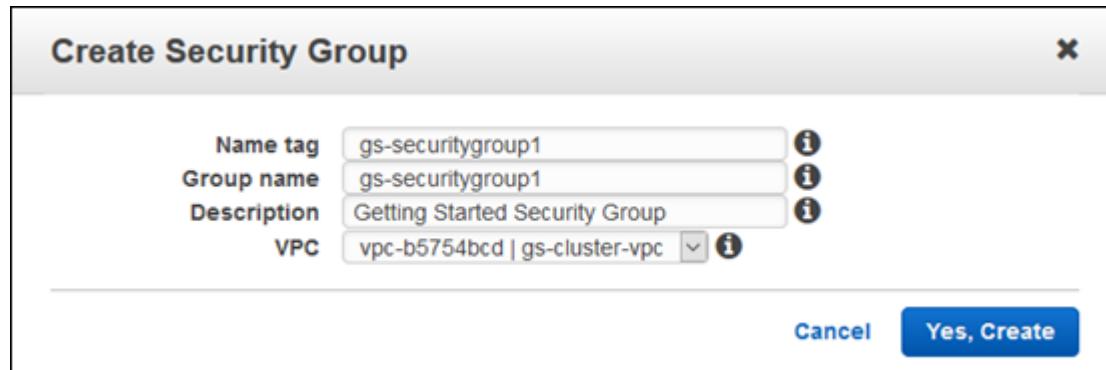
VPC와 서브넷 생성을 마쳤으면 다음 단계로 보안 그룹을 생성하고 인바운드 규칙을 추가해야 합니다.

보안 그룹을 생성하는 방법

Amazon Aurora DB 클러스터에 사용할 VPC의 마지막 생성 단계는 VPC 보안 그룹의 생성입니다. 이 보안 그룹으로 VPC의 DB 인스턴스에 액세스할 수 있는 네트워크 주소와 프로토콜을 식별할 수 있습니다.

1. VPC 대시보드에서 보안 그룹과 보안 그룹 생성을 차례대로 선택합니다.
2. [Create Security Group] 패널에서 다음과 같이 값을 설정합니다.
 - Name tag: `gs-securitygroup1`
 - 그룹 이름: `gs-securitygroup1`

- 설명: Getting Started Security Group
- VPC: 앞서 생성한 VPC를 선택합니다(예: vpc-b5754bcd | gs-cluster-vpc).



3. 예, 생성을 선택하여 보안 그룹을 생성합니다.

인바운드 규칙을 보안 그룹에 추가하려면

Aurora DB 클러스터에 연결하려면 인바운드 트래픽으로 연결할 수 있는 VPC 보안 그룹에 인바운드 규칙을 추가해야 합니다.

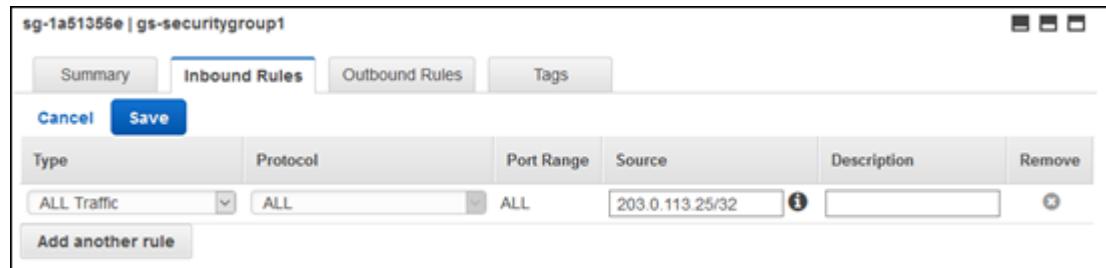
- Aurora 클러스터에 연결할 때 사용할 IP 주소를 측정합니다. <https://checkip.amazonaws.com>의 서비스를 사용하여 퍼블릭 IP 주소를 확인할 수 있습니다. 고정 IP 주소 없이 ISP 또는 방화벽을 경유하여 연결하는 경우에는 클라이언트 컴퓨터가 사용하는 IP 주소의 범위를 알아내야 합니다.

Warning

0.0.0.0/0을 사용하면 모든 IP 주소를 활성화하여 DB 클러스터에 액세스할 수 있습니다. 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션 환경에서는 특정 IP 주소나 주소 범위에서만 DB 클러스터 액세스를 허용하도록 설정해야 합니다.

- VPC 대시보드에서 보안 그룹을 선택하고 이전 절차에서 생성한 gs-securitygroup1 보안 그룹을 선택합니다.
- 인바운드 탭을 선택한 후 편집 버튼을 선택합니다.
- 새로운 인바운드 규칙에 대해 다음과 같이 값을 설정합니다.

- Type: All Traffic
- Source: 이전 단계의 IP 주소 또는 범위(예: 203.0.113.25/32)



5. 설정을 저장하려면 저장을 선택합니다.

DB 서브넷 그룹 만들기

마지막으로 Aurora DB 클러스터를 생성할 때 필요한 것이 바로 DB 서브넷 그룹입니다. DB 서브넷 그룹은 DB 클러스터가 사용할 서브넷을 이전 단계에서 생성한 VPC에서 식별합니다. DB 서브넷 그룹은 DB 클러스터를 배포하려는 AWS 리전에서 두 개 이상의 가용 영역에 한 개 이상의 서브넷을 포함해야 합니다.

Aurora DB 클러스터에 사용할 DB 서브넷 그룹을 생성하는 방법

1. <https://console.aws.amazon.com/rds>에서 Amazon Aurora 콘솔을 엽니다.
2. 서브넷 그룹을 선택한 후 DB 서브넷 그룹 생성을 선택합니다.
3. 새로운 DB 서브넷 그룹에 대해 다음과 같이 값을 설정합니다.
 - 이름: gs-subnetgroup1
 - 설명: Getting Started Subnet Group
 - VPC ID: 이전 절차에서 생성한 VPC를 선택합니다(예: gs-cluster-vpc (vpc-b5754bcd)).
4. Add all the subnets related to this VPC(이 VPC와 관련된 모든 서브넷 추가)를 선택하여 이전 단계에서 생성한 VPC의 서브넷을 추가합니다. 가용 영역 및 서브넷의 값을 선택한 다음 Add subnet(서브넷 추가)을 선택하여 각 서브넷을 개별적으로 추가할 수도 있습니다.

RDS > Subnet groups > Create DB subnet group

Create DB subnet group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select an existing VPC, you will be able to add subnets related to that VPC.

Subnet group details

Name: gs-subnetgroup1

Description: Getting Started Subnet Group

VPC: gs-cluster-vpc (vpc-b5754bcd)

Add subnets

Add subnet(s) to this subnet group. You may add subnets one at a time below or add all the subnets related to this VPC. You may make additions/edits after this group is created. A minimum of 2 subnets is required.

Add all the subnets related to this VPC

Availability zone: select an availability zone

Subnet: select a subnet Add subnet

Subnets in this subnet group (2)

Availability zone	Subnet ID	CIDR block	Action
us-east-1a	subnet-52f6687d	10.0.0.0/24	Remove
us-east-1c	subnet-5d069500	10.0.1.0/24	Remove

Cancel Create

5. 생성을 선택하여 서브넷 그룹을 생성합니다.

VPC에서 DB 인스턴스에 액세스하는 시나리오

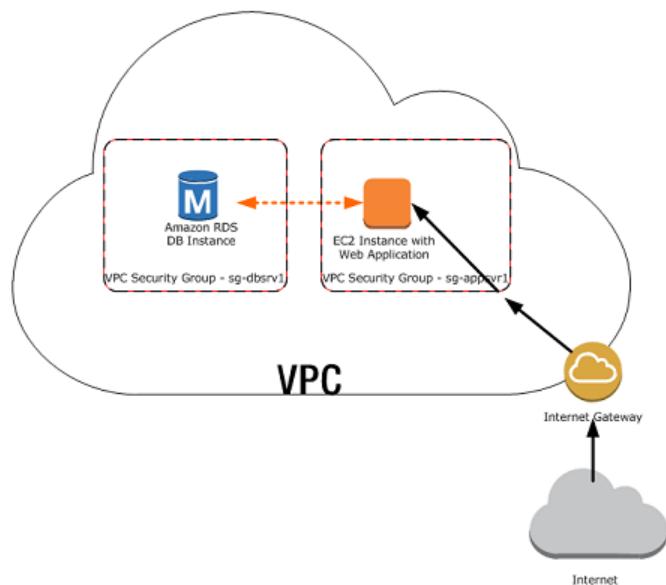
Amazon Aurora에서는 DB 인스턴스에 액세스하는 다음 시나리오를 지원합니다.

- 동일한 VPC에 있는 EC2 인스턴스 (p. 1012)
- 다른 VPC에 있는 EC2 인스턴스 (p. 1013)
- VPC에 있지 않은 EC2 인스턴스 (p. 1014)
- 클라이언트 애플리케이션이 인터넷을 통해 (p. 1015)

동일한 VPC에 있는 EC2 인스턴스가 VPC 내에 있는 DB 인스턴스에 액세스

VPC에서 DB 인스턴스를 사용하는 일반적인 사례는 동일한 VPC의 EC2 인스턴스에서 실행 중인 애플리케이션 서버와 데이터를 공유하는 것입니다. 이 사용자 시나리오는 AWS Elastic Beanstalk을 사용해 동일한 VPC에서 EC2 인스턴스와 DB 인스턴스를 만드는 경우입니다.

다음 다이어그램은 이 시나리오를 보여 줍니다.



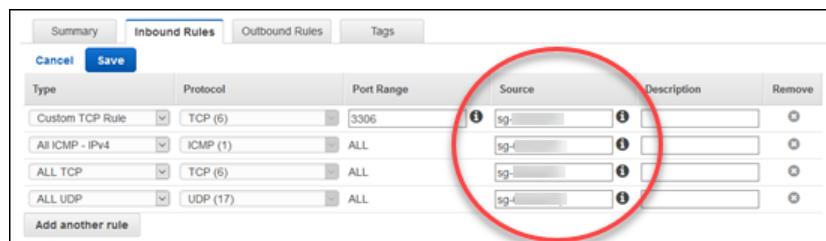
동일한 VPC에서 EC2 인스턴스와 DB 인스턴스 간 액세스를 관리하는 가장 간단한 방법은 다음과 같습니다.

- DB 인스턴스가 포함될 VPC 보안 그룹을 생성합니다. 이 보안 그룹을 사용해 DB 인스턴스에 대한 액세스를 제한할 수 있습니다. 예를 들어, DB 인스턴스를 만들 때 할당한 포트와 개발 등을 위해 DB 인스턴스에 액세스할 때 이용할 IP 주소를 사용해 TCP 액세스를 허용하는 이 보안 그룹을 위한 사용자 지정 규칙을 만들 수 있습니다.
- EC2 인스턴스(웹 서버 및 클라이언트)가 포함될 VPC 보안 그룹을 생성합니다. 이 보안 그룹은 필요할 경우 VPC의 라우팅 테이블을 통한 EC2 인스턴스 액세스를 허용할 수 있습니다. 예를 들어, 이 보안 그룹에서 TCP가 포트 22를 통해 EC2 인스턴스에 액세스하도록 허용하는 규칙을 설정할 수 있습니다.
- DB 인스턴스에 대한 보안 그룹에서 EC2 인스턴스에 대해 생성한 보안 그룹으로부터의 연결을 허용하는 사용자 지정 규칙을 만듭니다. 그러면 보안 그룹의 모든 구성원이 DB 인스턴스에 액세스하도록 허용됩니다.

이 시나리오에 대해 퍼블릭 서브넷과 프라이빗 서브넷 모두에서 VPC를 생성하는 방법을 보여주는 자습서는 [자습서: DB 인스턴스에 사용할 Amazon VPC 생성 \(p. 1020\)](#) 단원을 참조하십시오.

VPC 보안 그룹에서 다른 보안 그룹으로부터의 연결을 허용하는 규칙을 만들려면 다음을 수행합니다.

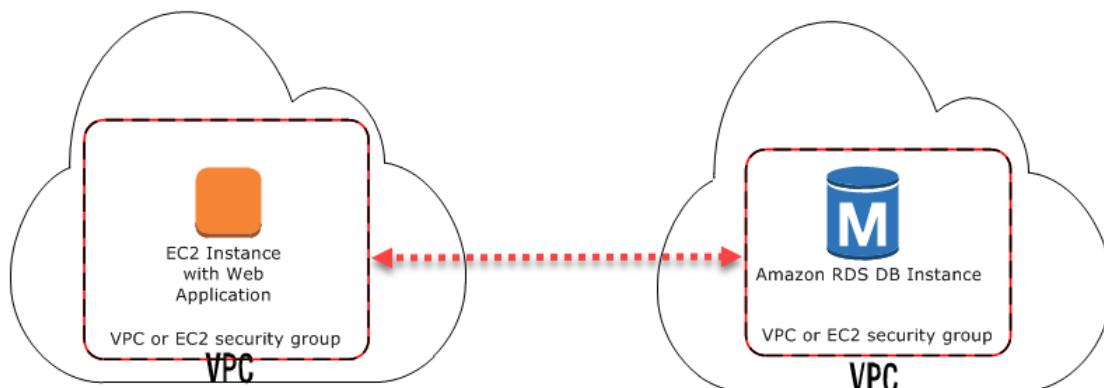
1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/vpc>에서 Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 [Security Groups]를 선택합니다.
3. 다른 보안 그룹의 구성원에 대한 액세스를 허용할 보안 그룹을 선택하거나 생성합니다. 위 시나리오에서 이 보안 그룹을 DB 인스턴스에 사용합니다. 인바운드 규칙 탭을 선택한 후 규칙 편집을 선택합니다.
4. 인바운드 규칙 편집 페이지에서 규칙 추가를 선택합니다.
5. 유형에서 모든 ICMP 옵션 중 하나를 선택합니다. [Source] 상자에 보안 그룹 ID를 입력하기 시작합니다. 입력을 시작하면 보안 그룹 목록이 제공됩니다. 이 보안 그룹에서 보호 중인 리소스에 대한 액세스를 허용할 구성원이 포함된 보안 그룹을 선택합니다. 위 시나리오에서 이 보안 그룹을 EC2 인스턴스에 사용합니다.
6. [Type]으로 [All TCP]를 지정하고 [Source] 상자에 사용자의 보안 그룹을 입력한 규칙을 만들어 TCP 프로토콜에 대해 단계를 반복합니다. UDP 프로토콜을 사용하려는 경우, [Type]으로 [All UDP]를 지정하고 [Source] 상자에 사용자의 보안 그룹을 입력하여 규칙을 만듭니다.
7. DB 인스턴스를 만들 때 사용한 포트(예: MySQL용 포트 3306)를 통한 액세스를 허용하는 사용자 지정 TCP 규칙을 만듭니다. 소스 상자에 사용할 보안 그룹 또는 IP 주소를 입력합니다.
8. 모두 완료했으면 저장을 선택합니다.



VPC에 있는 DB 인스턴스에 다른 VPC에 있는 EC2 인스턴스가 액세스

DB 인스턴스가 액세스 시 사용할 EC2 인스턴스와 다른 VPC에 있는 경우 DB 인스턴스에 액세스하기 위해 VPC 피어링을 사용할 수 있습니다.

다음 다이어그램은 이 시나리오를 보여 줍니다.

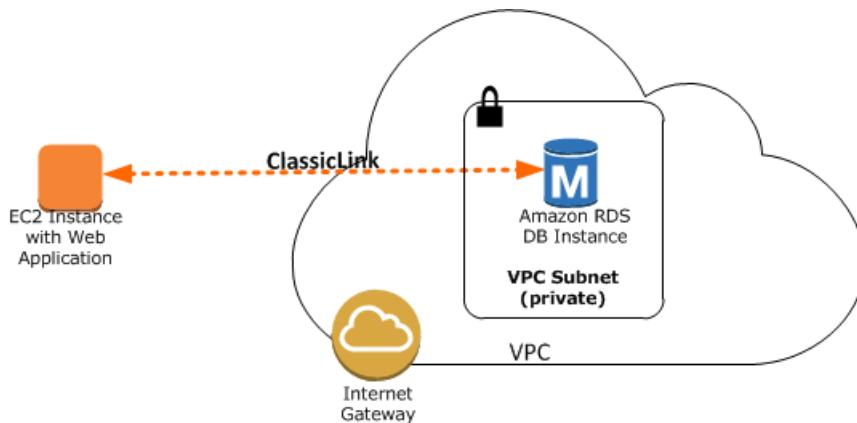


VPC 피어링 연결은 프라이빗 IP 주소를 사용하여 두 VPC 간에 트래픽을 라우팅할 수 있도록 하기 위한 두 VPC 사이의 네트워킹 연결입니다. 동일한 네트워크에 속하는 경우와 같이 VPC의 인스턴스가 서로 통신할 수 있습니다. 자체 VPC 간, 다른 AWS 계정에서 VPC를 사용하여 또는 다른 AWS 리전에서 VPC를 사용하여 VPC 피어링 연결을 만들 수 있습니다. VPC 피어링에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 [VPC 피어링](#) 단원을 참조하십시오.

VPC에 있지 않은 EC2 인스턴스가 VPC에 있는 DB 인스턴스에 액세스

ClassicLink를 사용하여 VPC에 있는 Amazon Aurora DB 인스턴스와 Amazon VPC에 있지 않은 EC2 인스턴스 간에 통신할 수 있습니다. ClassicLink를 사용하면 EC2 인스턴스상의 애플리케이션은 DB 인스턴스에 대한 엔드포인트를 사용하여 DB 인스턴스에 연결할 수 있습니다. ClassicLink는 무료로 사용할 수 있습니다.

다음 다이어그램은 이 시나리오를 보여 줍니다.



ClassicLink를 사용하여 IP 주소 범위를 정의하고 액세스 제어 목록(ACL)을 제어하여 네트워크 트래픽을 관리하는 논리적으로 분리되어 있는 데이터베이스에 EC2 인스턴스를 연결할 수 있습니다. VPC에서는 퍼블릭 IP 주소 또는 터널링을 사용하여 DB 인스턴스와 통신할 필요가 없습니다. 따라서 인스턴스 간 통신에서 처리량이 향상되고 연결 시 지연 시간이 짧아집니다.

VPC에 있는 DB 인스턴스와 VPC에 있지 않은 EC2 인스턴스 간에 ClassicLink를 활성화하려면

1. AWS Management 콘솔에 로그인한 다음 <https://console.aws.amazon.com/vpc>에서 Amazon VPC 콘솔을 엽니다.
2. 탐색 창에서 Your VPCs(VPCs)를 선택합니다.
3. DB 인스턴스가 사용하는 VPC를 선택합니다.
4. [Actions]에서 [Enable ClassicLink]를 선택합니다. 확인 대화 상자에서 [Yes, Enable]을 선택합니다.
5. EC2 콘솔에서 VPC에 있는 DB 인스턴스에 연결할 EC2 인스턴스를 선택합니다.
6. [Actions]에서 [ClassicLink]를 선택한 후 [Link to VPC]를 선택합니다.
7. [Link to VPC] 페이지에서 사용하려는 보안 그룹을 선택한 후 [Link to VPC]를 선택합니다.

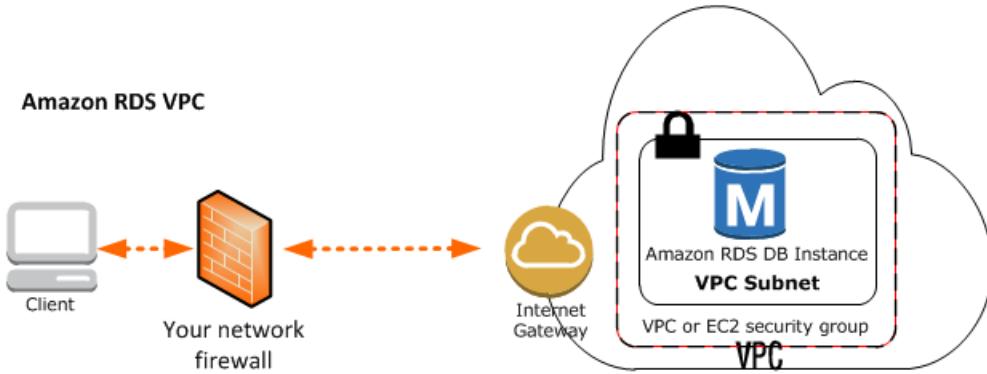
Note

ClassicLink 기능은 EC2-Classic을 지원하는 계정 및 리전의 콘솔에만 표시됩니다. 자세한 내용은 Linux 인스턴스용 Amazon EC2 사용 설명서에서 [ClassicLink](#) 단원을 참조하십시오.

클라이언트 애플리케이션이 인터넷을 통해 VPC에 있는 DB 인스턴스에 액세스

인터넷을 통해 클라이언트 애플리케이션에서 VPC에 있는 DB 인스턴스에 액세스하려면 단일 퍼블릭 서브넷만 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.
- VPC 및 서브넷과 연결된 Amazon Aurora DB 인스턴스입니다. Amazon RDS는 서브넷 내의 IP 주소를 DB 인스턴스에 할당합니다.
- VPC를 인터넷 및 다른 AWS 제품과 연결하는 인터넷 게이트웨이.
- DB 인스턴스와 연결된 보안 그룹. 보안 그룹의 인바운드 규칙은 클라이언트 애플리케이션이 사용자의 DB 인스턴스에 액세스할 수 있도록 해줍니다.

VPC에서 DB 인스턴스 생성에 대한 자세한 내용은 [VPC에 DB 인스턴스 만들기 \(p. 1017\)](#) 단원을 참조하십시오.

VPC에서 DB 인스턴스를 사용한 작업

, 사용자의 DB 인스턴스는 Virtual Private Cloud(VPC)에서 실행됩니다. VPC는 AWS 클라우드에 있는 다른 가상 네트워크와 논리적으로 격리되어 있는 가상 네트워크입니다. Amazon VPC에서는 Amazon Aurora DB 인스턴스 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.

VPC 내부와 Amazon Aurora DB 인스턴스를 포함하는 시나리오 목록은 [VPC에서 DB 인스턴스에 액세스하는 시나리오 \(p. 1011\)](#) 단원을 참조하십시오.

일반 Amazon Aurora 시나리오에서 사용할 수 있는 VPC를 생성하는 방법을 보여주는 자습서는 [자습서: DB 인스턴스에 사용할 Amazon VPC 생성 \(p. 1020\)](#) 단원을 참조하십시오.

VPC 내부에서 DB를 사용하여 작업을 수행하는 방법은 다음 항목을 참조하십시오.

주제

- VPC에서 DB 인스턴스를 사용한 작업 (p. 1016)
- DB 서브넷 그룹을 사용한 작업 (p. 1016)
- VPC에 있는 DB 인스턴스를 인터넷에서 습기기 (p. 1017)
- VPC에 DB 인스턴스 만들기 (p. 1017)

VPC에서 DB 인스턴스를 사용한 작업

다음은 VPC에서 DB 인스턴스를 사용하여 작업할 때 유용한 몇 가지 팁입니다.

- VPC는 최소 2개 이상의 서브넷을 보유해야 합니다. 이러한 서브넷은 DB 인스턴스를 배포하고자 하는 AWS 리전에 있는 2개의 다른 가용 영역에 있어야 합니다. 서브넷은 사용자가 지정할 수 있고 보안 및 운영상의 필요를 바탕으로 인스턴스를 그룹화할 수 있게 해주는 VPC IP 주소 범위의 한 부분입니다.
- VPC에서 DB 인스턴스에 공개적으로 액세스할 수 있도록 하려면 VPC 속성인 [DNS hostnames]와 [DNS resolution]을 활성화해야 합니다.
- VPC에는 사용자가 만드는 DB 서브넷 그룹이 있어야 합니다(자세한 내용은 다음 섹션 참조). 생성한 서브넷을 지정하여 DB 서브넷 그룹을 생성합니다. Amazon Aurora에서는 해당 DB 서브넷 그룹과 기본 가용 영역을 사용하여 DB 인스턴스에 할당할 서브넷과 해당 서브넷 내의 IP 주소를 선택합니다.
- VPC에는 DB 인스턴스에 대한 액세스를 허용하는 VPC 보안 그룹이 있어야 합니다.
- 각 서브넷의 CIDR 블록은 장애 조치와 컴퓨팅 조정을 포함한 유지 관리 활동 중에 Amazon Aurora가 사용할 예비 IP 주소를 수용할 만큼 충분히 커야 합니다.
- VPC의 [instance tenancy] 속성 값은 [default] 또는 [dedicated] 중 하나일 수 있습니다. 모든 기본 VPC에서 인스턴스 테넌시 속성이 기본값으로 설정되어 있으며, 기본 VPC는 어떤 DB 인스턴스 클래스라도 지원할 수 있습니다.

인스턴스 테넌시 속성이 전용으로 설정되어 있는 전용 VPC에 DB 인스턴스를 두 기로 선택할 경우, DB 인스턴스의 DB 인스턴스 클래스는 승인된 Amazon EC2 전용 인스턴스 유형 중 하나여야 합니다. 예를 들어 m3.medium EC2 전용 인스턴스는 db.m3.medium DB 인스턴스 클래스에 해당합니다. VPC의 인스턴스 테넌시에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 [EC2 전용 인스턴스 사용](#)을 참조하십시오.

전용 인스턴스에 존재할 수 있는 인스턴스 유형에 대한 자세한 내용은 EC2 요금 페이지에서 [Amazon EC2 전용 인스턴스](#)를 참조하십시오.

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 인스턴스에 대해 지정하는 서브넷(일반적으로 프라이빗)의 모음입니다. DB 서브넷 그룹을 통해 CLI 또는 API를 사용하여 DB 인스턴스를 생성할 때 특정 VPC를 지정할 수 있고, 콘솔을 사용하면 사용하려는 VPC와 서브넷만 선택할 수 있습니다.

각 DB 서브넷 그룹은 지정된 AWS 리전에서 두 개 이상의 가용 영역에 서브넷이 있어야 합니다. VPC에서 DB 인스턴스를 생성할 때는 DB 서브넷 그룹을 선택해야 합니다. Amazon Aurora에서는 해당 DB 서브넷 그룹과 기본 가용 영역을 사용하여 DB 인스턴스에 연결할 서브넷과 해당 서브넷 내의 IP 주소를 선택합니다. 다중 AZ 배포의 기본 DB 인스턴스에 오류가 있을 경우 Amazon Aurora는 해당 스탠바이를 승격한 후 다른 가용 영역 중 하나에서 서브넷의 IP 주소를 사용하여 새 스탠바이를 만들 수 있습니다.

Amazon Aurora가 VPC에 DB 인스턴스를 생성할 때는 DB 서브넷 그룹의 IP 주소를 사용하여 DB 인스턴스에 네트워크 인터페이스를 할당합니다. 하지만 기본 IP 주소가 장애 조치 중에 바뀌기 때문에 DB 인스턴스에 연결할 때 반드시 DNS 이름을 사용하는 것이 좋습니다.

Note

VPC에서 실행하는 각각의 DB 인스턴스에 대해 Amazon Aurora가 복구 작업에 사용할 수 있도록 DB 서브넷 그룹의 각 서브넷에 한 개 이상의 주소를 예약해야 합니다.

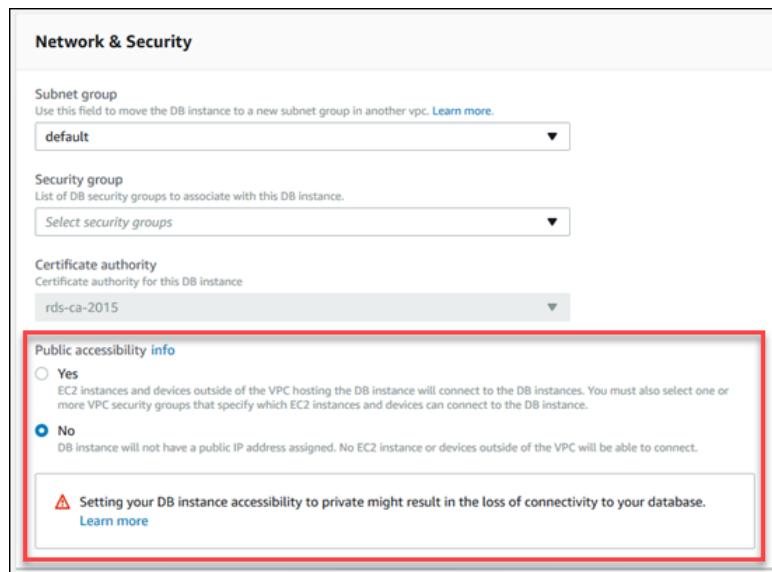
VPC에 있는 DB 인스턴스를 인터넷에서 숨기기

한 가지 일반적인 Amazon Aurora 시나리오는 일반에 공개되어 있는 웹 애플리케이션을 포함한 EC2 인스턴스와 공개적으로 액세스할 수 없는 데이터베이스를 포함한 DB 인스턴스가 있는 VPC를 사용하는 경우입니다. 예를 들어, 퍼블릭 서브넷과 프라이빗 서브넷이 있는 VPC를 생성할 수 있습니다. 웹 서버의 기능을 수행하는 Amazon EC2 인스턴스가 퍼블릭 서브넷에 배포될 수 있으며, DB 인스턴스는 프라이빗 서브넷에 배포됩니다. 이런 배포 환경에서는 웹 서버만 DB 인스턴스에 액세스할 수 있습니다. 이 시나리오에 대한 그림은 [동일한 VPC에 있는 EC2 인스턴스가 VPC 내에 있는 DB 인스턴스에 액세스 \(p. 1012\)](#) 단원을 참조하십시오.

VPC 내에서 DB 인스턴스를 시작하면 DB 인스턴스는 VPC 내부 트래픽에 대한 프라이빗 IP 주소를 가집니다. 이 프라이빗 IP 주소는 공개적으로 액세스할 수 없습니다. 퍼블릭 액세스 가능성 옵션을 사용하여 DB 인스턴스에 프라이빗 IP 주소 외에 퍼블릭 IP 주소가 있는지 여부를 지정할 수 있습니다. DB 인스턴스가 공개적으로 액세스할 수 있는 것으로 지정된 경우, 해당 DNS 엔드포인트는 DB 인스턴스의 VPC 내부 프라이빗 IP 주소와 DB 인스턴스의 VPC 외부 퍼블릭 IP 주소로 확인됩니다. DB 인스턴스에 대한 액세스는 궁극적으로 인스턴스에서 사용하는 보안 그룹에 의해 제어되며, DB 인스턴스에 할당된 보안 그룹에서 퍼블릭 액세스를 허용하지 않으면 퍼블릭 액세스가 허용되지 않습니다.

퍼블릭 액세스 가능성 옵션을 수정하여 퍼블릭 액세스 가능성을 켜거나 끄도록 DB 인스턴스를 수정할 수 있습니다. 이 파라미터는 다른 DB 인스턴스 파라미터와 같은 방법으로 수정합니다. 자세한 내용은 DB 엔진에 대한 수정을 참조하십시오.

다음 그림은 네트워크 보안 섹션의 퍼블릭 액세스 가능 옵션을 보여 줍니다.



퍼블릭 액세스 가능성 옵션을 설정하기 위해 DB 인스턴스를 수정하는 방법에 대한 자세한 내용은 [DB 클러스터에서 DB 인스턴스 수정 \(p. 194\)](#) 단원을 참조하십시오.

VPC에 DB 인스턴스 만들기

다음 절차에 따라 VPC에서 DB 인스턴스를 생성할 수 있습니다. 계정에 기본 VPC가 있으면 VPC 및 DB 서브넷 그룹이 자동으로 생성된 것이므로 3단계부터 시작하면 됩니다. AWS 계정에 기본 VPC가 없거나 추가 VPC를 만들려면 새 VPC를 만들 수 있습니다.

Note

VPC에서 DB 인스턴스에 공개적으로 액세스할 수 있도록 하려면 VPC 속성인 [DNS hostnames]와 [DNS resolution]을 활성화하여 VPC에 대한 DNS 정보를 업데이트해야 합니다. VPC 인스턴스에 대한 DNS 정보의 업데이트에 대한 자세한 내용은 [Updating DNS Support for Your VPC](#)를 참조하십시오.

다음 단계에 따라 VPC에서 DB 인스턴스를 만들 수 있습니다:

- 1 단계: VPC 생성 (p. 1018)
- 2단계: VPC에 서브넷 추가 (p. 1018)
- 3단계: DB 서브넷 그룹 만들기 (p. 1018)
- 4단계: VPC 보안 그룹 만들기 (p. 1019)
- 5단계: VPC에서 DB 인스턴스 만들기 (p. 1019)

1 단계: VPC 생성

AWS 계정에 기본 VPC가 없거나 추가 VPC를 만들려면 지침에 따라 새 VPC를 만듭니다. [프라이빗 서브넷과 퍼블릭 서브넷을 포함하는 VPC 생성 \(p. 1020\)](#) 단원을, 혹은 Amazon VPC 설명서에서 [1단계: VPC 생성](#) 단원을 참조하십시오.

2단계: VPC에 서브넷 추가

VPC를 만들었으면 두 개 이상의 가용 영역에 서브넷을 만들어야 합니다. DB 서브넷 그룹을 만들 때 이들 서브넷이 사용됩니다. 기본 VPC가 있는 경우에는 해당 AWS 리전의 각 가용 영역에 서브넷이 자동으로 생성됩니다.

VPC에서 서브넷을 생성하는 방법에 대한 지침은 [프라이빗 서브넷과 퍼블릭 서브넷을 포함하는 VPC 생성 \(p. 1020\)](#) 단원을 참조하십시오.

3단계: DB 서브넷 그룹 만들기

DB 서브넷 그룹은 사용자가 VPC에 대해 만든 다음 DB 인스턴스에 대해 지정하는 서브넷(일반적으로 프라이빗)의 모음입니다. DB 서브넷 그룹에서는 CLI 또는 API를 사용하여 DB 인스턴스를 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.

Note

DB 인스턴스에 공개적으로 액세스할 수 있도록 하려면 DB 서브넷 그룹의 서브넷에 인터넷 게이트웨이가 있어야 합니다. 서브넷용 인터넷 게이트웨이에 대한 자세한 내용은 Amazon VPC 설명서의 [인터넷 게이트웨이](#)를 참조하십시오.

VPC에서 DB 인스턴스를 생성할 때는 DB 서브넷 그룹을 선택해야 합니다. 그러면 Amazon Aurora에서 해당 DB 서브넷 그룹과 기본 가용 영역을 사용하여 서브넷과 해당 서브넷 내의 IP 주소를 선택합니다. Amazon Aurora에서는 탄력적 네트워크 인터페이스를 생성하여 해당 IP 주소가 있는 DB 인스턴스에 연결합니다. 다중 AZ 배포의 경우, 한 AWS 리전에 있는 두 개 이상의 가용 영역에 대한 서브넷을 정의하면 Amazon Aurora가 필요한 경우 다른 가용 영역에 새로운 예비 복제본을 만들 수 있습니다. 단일 AZ 배포의 경우도 어느 시점에 단일 AZ 배포를 다중 AZ 배포로 변환하려면 이 작업을 수행해야 합니다.

이 단계에서는 DB 서브넷 그룹을 만들고 VPC용으로 만든 서브넷을 추가합니다.

콘솔

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]를 선택합니다.
4. [Name]에 DB 서브넷 그룹의 이름을 입력합니다.
5. [Description]에 DB 서브넷 그룹에 대한 설명을 입력합니다.
6. [VPC]에 대해 생성된 VPC를 선택합니다.
7. 서브넷 추가 섹션에서 이 VPC와 관련된 모든 서브넷 추가를 선택합니다.

Create DB subnet group

To create a new Subnet Group give it a name, description, and select an existing VPC below. Once you select an existing VPC, you will be able to add subnets related to that VPC.

Subnet group details

Name: mydbsubnetgroup

Description: My DB Subnet Group

VPC: tutorial-vpc (vpc-971c12ee)

Add subnets

Add subnet(s) to this subnet group. You may add subnets one at a time below or add all the subnets related to this VPC. You may make additions/edits after this group is created. A minimum of 2 subnets is required.

Add all the subnets related to this VPC

Availability zone: select an availability zone

Subnet: select a subnet

Subnets in this subnet group (4)

Availability zone	Subnet ID	CIDR block	Action
us-east-1a	subnet-d8c8e7f4	10.0.2.0/24	Remove
us-east-1f	subnet-718fdc7d	10.0.3.0/24	Remove
us-east-1a	subnet-cbc8e7e7	10.0.1.0/24	Remove
us-east-1a	subnet-0ccde220	10.0.0.0/24	Remove

Note: The subnets listed here are associated with the selected VPC. You can remove them from this subnet group if needed.

Cancel **Create**

8. Create를 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.

4단계: VPC 보안 그룹 만들기

DB 인스턴스를 만들기 전에 DB 인스턴스와 연결할 VPC 보안 그룹을 만들어야 합니다. DB 인스턴스에 대한 보안 그룹을 생성하는 방법은 [프라이빗 DB 인스턴스에 대한 VPC 보안 그룹 생성 \(p. 1023\)](#) 단원을, 혹은 Amazon VPC 설명서에서 [VPC용 보안 그룹](#) 단원을 참조하십시오.

5단계: VPC에서 DB 인스턴스 만들기

이 단계에서는 DB 인스턴스를 만들고 이전 단계에서 만든 VPC 이름, DB 서브넷 그룹 및 VPC 보안 그룹을 사용합니다.

Note

VPC에서 DB 인스턴스에 공개적으로 액세스할 수 있도록 하려면 VPC 속성인 [DNS hostnames]와 [DNS resolution]을 활성화해야 합니다. VPC 인스턴스에 대한 DNS 정보를 업데이트하는 방법에 대한 자세한 내용은 [Updating DNS Support for Your VPC](#)를 참조하십시오.

DB 인스턴스 생성 방법에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 생성 \(p. 95\)](#) 단원을 참조하십시오.

Network & Security(네트워크 및 보안) 섹션에 메시지가 표시되면, 이전 단계에서 만든 VPC 이름, DB 서브넷 그룹 및 VPC 보안 그룹을 입력합니다.

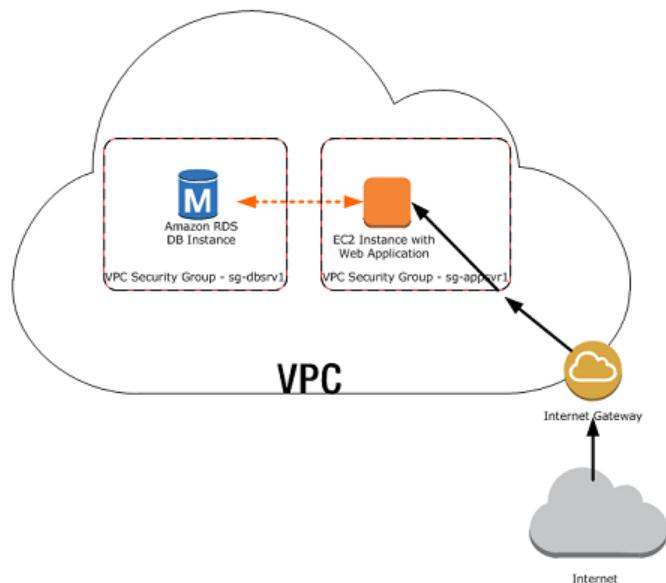
Note

현재 Aurora 클러스터에 대해서는 VPC 업데이트가 지원되지 않습니다.

자습서: DB 인스턴스에 사용할 Amazon VPC 생성

일반 시나리오에서는 동일한 VPC에서 실행 중인 웹 서버와 데이터를 공유하는 Amazon VPC DB 인스턴스가 포함됩니다. 이 자습서에서는 이 시나리오를 위한 VPC를 생성합니다.

다음 다이어그램은 이 시나리오를 보여 줍니다. 다른 시나리오에 대한 자세한 내용은 [VPC에서 DB 인스턴스에 액세스하는 시나리오 \(p. 1011\)](#)을(를) 참조하십시오.



공용 인터넷이 아닌 웹 서버에서만 DB 인스턴스를 사용할 수 있어야 하므로 퍼블릭 및 프라이빗 서브넷이 모두 있는 VPC를 생성합니다. 퍼블릭 서브넷에서 웹 서버를 호스팅하므로 웹 서버에서 퍼블릭 인터넷에 액세스할 수 있습니다. DB 인스턴스는 프라이빗 서브넷에서 호스팅됩니다. 동일한 VPC에서 호스팅되므로 웹 서버에서는 DB 인스턴스에 연결할 수 있지만, 퍼블릭 인터넷에서는 DB 인스턴스에 액세스할 수 없어 보다 강화된 보안이 가능합니다.

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 콘솔의 오른쪽 상단 모서리에서 VPC를 생성할 리전을 선택합니다. 이 예에서는 미국 서부(오레곤) 리전을 사용합니다.
3. 왼쪽 상단 모서리에서 [VPC Dashboard]를 선택합니다. VPC 생성을 시작하려면 VPC 마법사 시작을 선택합니다.
4. [Step 1: Select a VPC Configuration] 페이지에서 [VPC with Public and Private Subnets]를 선택한 후 [Select]를 선택합니다.

5. [Step 2: VPC with Public and Private Subnets] 페이지에서 다음 값을 설정합니다.

- IPv4 CIDR block: 10.0.0.0/16
- IPv6 CIDR block: No IPv6 CIDR Block
- VPC name: tutorial-vpc
- Public subnet's IPv4 CIDR: 10.0.0.0/24
- 가용 영역: us-west-2a
- Public subnet name: Tutorial public
- Private subnet's IPv4 CIDR: 10.0.1.0/24
- 가용 영역: us-west-2a
- Private subnet name: Tutorial Private 1
- Instance type: t2.small

Important

콘솔에 인스턴스 유형 상자가 표시되지 않으면 대신 NAT 인스턴스 사용을 선택합니다. 이 링크는 오른쪽에 있습니다.

Note

t2.small 인스턴스 유형이 표시되지 않으면 다른 인스턴스 유형을 선택할 수 있습니다.

- Key pair name: No key pair
- Service endpoints: Skip this field.
- Enable DNS hostnames: Yes
- Hardware tenancy: Default

6. 작업을 마쳤으면 [Create VPC]를 선택합니다.

추가 서브넷 생성

DB 인스턴스를 VPC에서 사용하기 위한 DB 서브넷 그룹을 만들려면 두 개의 프라이빗 서브넷 또는 두 개의 퍼블릭 서브넷이 있어야 합니다. 이 자습의 DB 인스턴스는 프라이빗으로 두 번째 프라이빗 서브넷을 VPC에 추가합니다.

서브넷을 추가로 생성하는 방법

1. <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
2. 두 번째 프라이빗 서브넷을 VPC에 추가하려면 VPC 대시보드를 선택하고 서브넷을 선택한 다음 서브넷 생성을 선택합니다.
3. 서브넷 생성 페이지에서 다음 값을 설정합니다.
 - Name tag: Tutorial private 2
 - VPC: 이전 단계에서 생성한 VPC를 선택합니다(예: vpc-*identifier* (10.0.0.0/16) | tutorial-vpc).
 - 가용 영역: us-west-2b

Note

첫 번째 프라이빗 서브넷에 대해 선택한 것과 다른 가용 영역을 선택합니다.

- IPv4 CIDR block: 10.0.2.0/24
4. 작업을 마쳤으면 생성을 선택합니다. 그런 다음 확인 페이지에서 닫기를 선택합니다.
 5. 두 번째로 생성한 프라이빗 서브넷이 첫 번째 프라이빗 서브넷과 동일한 라우팅 테이블을 사용하는지 확인하려면 [VPC Dashboard]를 선택하고 [Subnets]를 선택한 다음 VPC에 대해 생성한 첫 번째 프라이빗 서브넷인 Tutorial private 1을 선택합니다.

6. 서브넷 목록 아래에서 Route Table(라우팅 테이블) 탭을 선택하고 Route Table(라우팅 테이블)의 값을 기록합니다.—예: rtb-98b613fd
7. 서브넷 목록에서 첫 번째 프라이빗 서브넷 선택을 해제합니다.
8. 서브넷 목록에서 두 번째 프라이빗 서브넷 Tutorial private 2를 선택하고 [Route Table] 탭을 선택합니다.
9. 현재 라우팅 테이블과 첫 번째 프라이빗 서브넷에 사용한 라우팅 테이블이 동일하지 않은 경우 라우팅 테이블 연결 편집을 선택합니다. 라우팅 테이블 ID에서 앞서 기록한 라우팅 테이블을 선택합니다.—예: rtb-98b613fd 그런 다음 선택 사항을 저장하기 위해 저장을 선택합니다.

퍼블릭 웹 서버에 대해 VPC 보안 그룹 생성

이제 퍼블릭 액세스를 위한 보안 그룹을 생성합니다. VPC의 퍼블릭 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-securitygroup
 - 설명: Tutorial Security Group
 - VPC: 앞에서 생성한 VPC를 선택합니다(예: vpc-*identifier* (10.0.0.0/16) | tutorial-vpc).
4. 보안 그룹을 생성하려면 생성을 선택합니다. 그런 다음 확인 페이지에서 닫기를 선택합니다.

이 자습서에서 나중에 필요하므로 보안 그룹 ID를 적어둡니다.

인바운드 규칙을 보안 그룹에 추가하려면

1. VPC의 인스턴스에 연결할 때 사용할 IP 주소를 측정합니다. 퍼블릭 IP 주소를 확인하려면 <https://checkip.amazonaws.com>의 서비스를 사용합니다. IP 주소의 예는 203.0.113.25/32입니다.
2. 고정 IP 주소가 없는 방화벽 뒤나 ISP(인터넷 서비스 제공업체)를 통해 연결되어 있는 경우 클라이언트 컴퓨터가 사용하는 IP 주소의 범위를 찾아야 합니다.

Warning

0.0.0.0/0을 사용하면 모든 IP 주소에서 퍼블릭 인스턴스에 액세스할 수 있습니다. 이 방법은 테스트 환경에서 잠시 사용하는 것은 괜찮지만 프로덕션 환경에서는 안전하지 않습니다. 프로덕션에서는 특정 IP 주소나 주소 범위만 인스턴스에 액세스하도록 허용하십시오.

3. <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
4. [VPC Dashboard], [Security Groups]와 이전 절차에서 생성한 tutorial-securitygroup 보안 그룹을 차례로 선택합니다.
5. 보안 그룹 목록 아래에서 인바운드 규칙 탭을 선택한 후, 규칙 편집을 선택합니다.
6. 새로운 인바운드 규칙으로 다음 값을 설정하여 SSH(Secure Shell)에서 EC2 인스턴스에 액세스하도록 허용합니다. 이렇게 하면 EC2 인스턴스에 연결하여 웹 서버 및 다른 유ти리티를 설치하고 웹 서버의 컨텐츠를 업로드할 수 있습니다.
 - Type: SSH
 - [Source:] 1단계의 IP 주소 또는 범위(예: 203.0.113.25/32)
7. [Add another rule]을 선택합니다.

8. 새 인바운드 규칙에 다음 값을 설정하여 웹 서버에 대한 HTTP 액세스를 허용합니다.
 - Type: HTTP
 - 소스: 0.0.0.0/0.
9. 설정을 저장하려면 규칙 저장을 선택합니다. 그런 다음 확인 페이지에서 닫기를 선택합니다.

프라이빗 DB 인스턴스에 대한 VPC 보안 그룹 생성

DB 인스턴스를 프라이빗으로 유지하려면 프라이빗 액세스를 위한 보조 보안 그룹을 생성합니다. VPC의 프라이빗 인스턴스에 연결하려면 웹 서버로부터의 트래픽만을 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-db-securitygroup
 - 설명: Tutorial DB Instance Security Group
 - VPC: 앞에서 생성한 VPC를 선택합니다(예: vpc-**identifier** (10.0.0.0/16) | tutorial-vpc).
4. 보안 그룹을 생성하려면 생성을 선택합니다. 그런 다음 확인 페이지에서 닫기를 선택합니다.

인바운드 규칙을 보안 그룹에 추가하려면

1. <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 엽니다.
2. [VPC Dashboard], [Security Groups]와 이전 절차에서 생성한 tutorial-db-securitygroup 보안 그룹을 차례로 선택합니다.
3. 보안 그룹 목록 아래에서 인바운드 규칙 탭을 선택한 후, 규칙 편집을 선택합니다.
4. 인바운드 규칙 편집 페이지에서 규칙 추가를 선택합니다.
5. 새로운 인바운드 규칙으로 다음 값을 설정하여 EC2 인스턴스의 포트 3306에서 MySQL 트래픽을 허용합니다. 이렇게 하면 웹 서버에서 DB 인스턴스에 연결하여 웹 애플리케이션의 데이터를 데이터베이스에 저장하고 검색할 수 있습니다.
 - Type: MySQL/Aurora
 - Source: 이 자습서의 앞부분에서 만든 tutorial-securitygroup 보안 그룹의 식별자(예: sg-9edd5cfb)
6. 설정을 저장하려면 규칙 저장을 선택합니다. 그런 다음 확인 페이지에서 닫기를 선택합니다.

DB 서브넷 그룹 만들기

DB 서브넷 그룹은 사용자가 VPC에서 만든 다음 DB 인스턴스에 대해 지정하는 서브넷의 모음입니다. DB 서브넷 그룹에서 DB 인스턴스를 생성할 때 특정 VPC를 지정할 수 있습니다.

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. [Create DB subnet group] 페이지에서 [Subnet group details]에 이들 값을 설정합니다.

- 이름: tutorial-db-subnet-group
 - 설명: Tutorial DB Subnet Group
 - VPC: tutorial-vpc (vpc-*identifier*)
5. 서브넷 추가 섹션에서 이 VPC와 관련된 모든 서브넷 추가를 선택합니다.
 6. Create를 선택합니다.

새 DB 서브넷 그룹은 RDS 콘솔의 DB 서브넷 그룹 목록에 나타납니다. DB 서브넷 그룹을 클릭하면 창 하단에 있는 세부 정보 창에서 그룹과 연결된 모든 서브넷을 포함한 세부 정보를 확인할 수 있습니다.

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 \(p. 1025\)](#)
- [Amazon Aurora의 명명 제약 조건 \(p. 1026\)](#)
- [Amazon Aurora 파일 크기 제한 \(p. 1027\)](#)

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.

리소스	기본 할당량
DB 보안 그룹당 권한 부여	20
버스트 밸런스(1TiB 미만 인스턴스의 경우)	3000 IOPS
교차 리전 스냅샷 복사 요청	5
DB 클러스터	40
DB 클러스터 파라미터 그룹	50
DB 인스턴스	40
DB 서브넷 그룹	50
이벤트 구독	20
DB 클러스터당 AWS Identity and Access Management(IAM) 역할	5
DB 인스턴스당 IAM 역할	5
수동 스냅샷 수	100
수동 클러스터 스냅샷 수	100
파라미터 그룹 수	50
예약 DB 인스턴스	40
VPC(가상 프라이빗 클라우드) 보안 그룹별 규칙	50 인바운드, 50 아웃바운드
서브넷 그룹 1개당 서브넷 수	20
리소스당 태그	50

리소스	기본 할당량
VPC 보안 그룹	5

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 [서비스 할당량 콘솔](#)을 열어 추가 DB 인스턴스를 요청할 수 있습니다. 탐색 창에서 AWS services(AWS 서비스)를 선택합니다. Amazon RDS(Amazon 관계형 데이터베이스 서비스)를 선택하고, 할당량을 선택한 다음, 지침에 따라 할당량 증가를 요청합니다. 자세한 내용은 Service Quotas 사용 설명서의 [할당량 증가 요청](#)을 참조하십시오.

Amazon Aurora의 명명 제약 조건

다음 표는 Amazon Aurora의 명명 제약 조건을 설명한 것입니다.

리소스 또는 항목	Constraints
DB 클러스터 식별자	식별자에는 다음과 같은 명명 제약 조건이 적용됩니다. <ul style="list-style-type: none">1–63자의 영숫자 또는 하이픈으로 구성되어야 합니다.첫 번째 문자는 글자이어야 합니다.하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.AWS 리전별로 AWS 계정당 모든 DB 인스턴스가 고유해야 합니다.
초기 데이터베이스 이름	데이터베이스 이름 제약 조건은 Aurora MySQL과 PostgreSQL 간에 다릅니다. 자세한 내용은 각 DB 클러스터를 생성할 때 사용 가능한 설정을 참조하십시오.
마스터 사용자 이름	마스터 사용자 이름 제약 조건은 각 데이터베이스 엔진에 따라 다릅니다. 자세한 내용은 각 DB 클러스터를 생성할 때 사용 가능한 설정을 참조하십시오.
마스터 암호	마스터 데이터베이스 사용자의 암호에는 /, ", @ 또는 공백을 제외한 모든 인쇄 가능한 ASCII 문자가 포함될 수 있습니다. 마스터 암호 길이 제약 조건은 각 데이터베이스 엔진에 따라 다릅니다. 자세한 내용은 각 DB 클러스터를 생성할 때 사용 가능한 설정을 참조하십시오.
DB 파라미터 그룹 이름	이러한 이름에는 다음과 같은 제약 조건이 적용됩니다. <ul style="list-style-type: none">1–255자로 구성되어야 합니다.첫 번째 문자는 글자이어야 합니다.하이픈은 허용되지만 이름은 하이픈으로 끝나거나 하이픈이 2개 연속으로 이어져서는 안 됩니다.
DB 서브넷 그룹 이름	이러한 이름에는 다음과 같은 제약 조건이 적용됩니다. <ul style="list-style-type: none">1–255자로 구성되어야 합니다.영숫자, 스페이스, 하이픈, 밑줄, 마침표를 사용할 수 있습니다.

Amazon Aurora 파일 크기 제한

Aurora에서 테이블 크기 한도는 Aurora 클러스터 볼륨의 크기에 따라서만 제한을 받습니다. 따라서 Aurora MySQL DB 클러스터의 최대 테이블 크기는 64 tebibytes (TiB)이고 Aurora PostgreSQL DB 클러스터의 최대 테이블 크기는 32TiB입니다. 테이블 디자인 모범 사례(예: 대용량 테이블 분할)를 따르는 것이 좋습니다.

Aurora 문제 해결

다음 단원을 통해 Amazon RDS 및 Aurora의 DB 인스턴스와 관련해 발생한 문제를 해결할 수 있습니다.

주제

- [Amazon RDS DB 인스턴스에 연결할 수 없음 \(p. 1028\)](#)
- [Amazon RDS 보안 문제 \(p. 1029\)](#)
- [DB 인스턴스 소유자 역할 암호 재설정 \(p. 1030\)](#)
- [Amazon RDS DB 인스턴스 중단 또는 재부팅 \(p. 1030\)](#)
- [Amazon RDS DB 파라미터 변경 사항이 적용 안 됨 \(p. 1031\)](#)
- [Amazon Aurora MySQL 메모리 부족 문제 \(p. 1031\)](#)
- [Amazon Aurora MySQL 복제 문제 \(p. 1031\)](#)
- [No Space Left on Device\(장치에 남은 공간 없음\) 오류 \(p. 1034\)](#)

Amazon RDS API를 사용해 문제를 디버깅하는 방법에 대한 자세한 내용은 [Aurora에서 애플리케이션 문제 해결 \(p. 1036\)](#) 단원을 참조하십시오.

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 원본의 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. \(p. 72\)](#) 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 인스턴스를 인터넷에서 숨기기 \(p. 1017\)](#) 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.

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 바꾸고 `port`를 DB 인스턴스의 포트로 바꿈).

```
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.c6c8mn7fake0.us-west-2.rds.amazonaws.com 819
Connecting To sg-postgresql1.c6c8mn7fake0.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

DB 인스턴스에 연결할 수 있지만 인증 오류가 발생하는 경우 DB 인스턴스에 대한 마스터 사용자 암호를 재설정하고 싶을 수도 있을 것입니다. RDS 인스턴스를 수정하여 이 작업을 수행할 수 있습니다.

Amazon RDS 보안 문제

보안 문제를 피하려면 사용자 계정에 마스터 AWS 사용자 이름과 암호를 절대 사용하지 마십시오. 모범 사례에 따라 마스터 AWS 계정을 사용하여 AWS Identity and Access Management(IAM) 사용자를 생성하고 이런 사용자를 DB 사용자 계정에 할당하는 것이 좋습니다. 필요한 경우 마스터 계정을 사용하여 다른 사용자 계정을 만들 수도 있습니다.

IAM 사용자를 만드는 방법에 대한 자세한 정보는 [IAM 사용자 생성 \(p. 69\)](#) 단원을 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 있습니다. 기존 계정인 경우 DB 인스턴스 생성과 같은 특정 작업을 수행하기 위한 액세스 정책에 권한이 없을 수 있습니다. 이 문제를 해결하려면 IAM 관리자가 필요한 역할을 해당 계정에 제공해야 합니다. 자세한 내용은 [IAM 설명서](#)를 참조하십시오.

DB 인스턴스 소유자 역할 암호 재설정

마스터 암호를 재설정하여 DB 인스턴스에 할당된 권한을 재설정할 수 있습니다.

예를 들어 SQL Server 데이터베이스의 db_owner 역할의 암호를 분실했을 경우 DB 인스턴스 마스터 암호를 수정하여 db_owner 역할 암호를 재설정할 수 있습니다. DB 인스턴스 암호를 변경하면 DB 인스턴스에 다시 액세스할 수 있습니다. DB 인스턴스 암호를 변경하면 수정된 db_owner 암호를 사용하여 데이터베이스에 액세스할 수 있으며, 실수로 취소되었을 수 있는 db_owner 역할에 대한 권한을 복원할 수도 있습니다. Amazon RDS 콘솔, AWS CLI 명령 [modify-db-instance](#), 또는 [ModifyDBInstance API](#) 작업을 사용하여 DB 인스턴스 암호를 변경할 수 있습니다.

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. 재부팅은 DB 인스턴스를 수동으로 재부팅할 때 또는 DB 인스턴스 설정을 변경하여 이 변경 사항을 적용하기 위해 재부팅해야 할 때 발생할 수 있습니다.

DB 인스턴스 재부팅은 설정을 변경하여 재부팅해야 할 때 또는 수동으로 재부팅할 때 이 발생합니다. 설정을 변경하고 변경 사항을 즉시 적용할 것을 요청하는 경우에 재부팅이 수행되거나, 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 변경하고 즉시 적용이 `true`로 설정된 경우
- 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 사용하거나 [RebootDBInstance API](#) 작업을 명시적으로 호출하여 DB 인스턴스를 재부팅할 수 있습니다(DB 인스턴스가 Multi-AZ deployment에 있는 경우 장애 조치 없음). 정적 파라미터 변경 후 연결된 DB 인스턴스를 재부팅하도록 하면 잘못된 파라미터 구성이 API 호출에 영향을 주는 위험을 완화할 수 있습니다. 예를 들어, [ModifyDBInstance](#)를 호출하여 DB 인스턴스 클래스를 변경할 수 있습니다. 자세한 내용은 [DB 파라미터 그룹의 파라미터 수정 \(p. 174\)](#) 단원을 참조하십시오.

Amazon Aurora MySQL 메모리 부족 문제

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 메모리 부족(OOM: Out of Memory) 및 데이터베이스 재시작이 발생하지 않도록 이 메모리를 해제하는 작업 목록을 수행할 수 있습니다. 이 인스턴스 수준 파라미터는 메모리가 부족할 때 DB 인스턴스가 수행하는 첨표로 구분된 작업 문자열을 사용합니다. 유효 작업으로 `print`, `tune`, `decline`, `kill_query` 또는 이상을 조합한 것을 들 수 있습니다. 빈 문자열은 어떠한 작업도 수행하지 말아야 함을 의미하므로 해당 기능을 효과적으로 비활성화합니다.

다음은 `aurora_oom_response` 파라미터에 대한 사용량 예제입니다.

- `print` – 많은 양의 메모리를 사용하는 쿼리만 인쇄합니다.
- `tune` – 내부 테이블 캐시를 조정하여 일부 메모리를 시스템으로 돌려줍니다.
- `decline` – 인스턴스 메모리가 부족해지면 새 쿼리를 거부합니다.
- `kill_query` – 인스턴스 메모리가 하한값 이상이 될 때까지 메모리 사용량이 많은 순서로 쿼리를 종료합니다. 데이터 정의 언어(DDL) 설명문이 종료되지 않습니다.
- `print`, `tune` – `print` 및 `tune`에 대해 설명된 작업을 수행합니다.
- `tune`, `decline`, `kill_query` – `tune`, `decline` 및 `kill_query`에 대해 설명된 작업을 수행합니다.

db.t2.small DB 인스턴스 클래스의 경우, `aurora_oom_response` 파라미터는 기본값인 `print`, `tune`으로 설정됩니다. 다른 모든 DB 인스턴스 클래스의 경우, 파라미터 값은 기본값으로 비어 있습니다(비활성화).

Amazon Aurora MySQL 복제 문제

일부 MySQL 복제 문제도 Aurora MySQL에 적용됩니다. 이러한 문제를 진단하고 해결할 수 있습니다.

주제

- [Read Replica 사이의 지연 문제 진단 및 해결 \(p. 1031\)](#)
- [MySQL 읽기 복제 오류 진단 및 해결 \(p. 1033\)](#)
- [Slave Down 또는 Disabled 오류 \(p. 1034\)](#)

Read Replica 사이의 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 읽

기 전용 복제본의 복제 지연 시간은 0보다 큽니다. Amazon RDS AuroraBinlogReplicaLag 지표를 보고 Amazon CloudWatch에서 이 지연 시간을 모니터링할 수 있습니다.

AuroraBinlogReplicaLag 지표는 MySQL SHOW SLAVE STATUS 명령의 Seconds_Behind_Master 필드 값을 보고합니다. 자세한 정보는 [SHOW SLAVE STATUS](#) 단원을 참조하십시오.

AuroraBinlogReplicaLag 지표가 0에 도달하면 복제본이 원본 DB 인스턴스를 따라잡은 것입니다. AuroraBinlogReplicaLag 지표가 -1을 반환하는 경우 복제가 활성 상태가 아닐 수 있습니다. 복제 오류 문제를 해결하는 방법은 [MySQL 읽기 복제 오류 진단 및 해결 \(p. 1033\)](#) 단원을 참조하십시오.

AuroraBinlogReplicaLag 값이 -1인 경우 Seconds_Behind_Master 값을 결정할 수 없거나 이 값이 NULL이라는 의미일 수도 있습니다.

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 원본 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 [Replication Implementation Details](#)를 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 잡기고 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`을 0으로 설정하여 쿼리 캐시를 비활성화할 수 있습니다. 쿼리 캐시에 대한 자세한 정보는 [Query Cache Configuration](#)을 참조하십시오.
- InnoDB for MySQL의 읽기 전용 복제본에서 버퍼풀을 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 버퍼풀에 캐시합니다. 이 방법을 사용하면 복제본 지연 시간을 줄일 수 있습니다. 다음은 그 한 예입니다.

Linux, OS X, Unix의 경우:

```
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 모니터링하고, 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 \(p. 434\)](#), [RDS-EVENT-0046 \(p. 434\)](#) 및 [RDS-EVENT-0047 \(p. 433\)](#)을 포함하여 읽기 전용 복제본의 상태를 표시하는 이벤트도 생성됩니다. 이벤트와 이벤트 구독에 대한 자세한 정보는 [Amazon RDS 이벤트 알림 서비스 사용 \(p. 429\)](#) 단원을 참조하십시오. MySQL 오류 메시지가 반환되는 경우 [MySQL 오류 메시지 설명서](#)에 있는 오류를 확인하십시오.

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 지정하는데 사용됩니다. 원본 DB 인스턴스에 대한 `max_allowed_packet` 값이 읽기 전용 복제본에 대한 `max_allowed_packet` 값보다 작을 경우 복제 프로세스에서 오류가 발생하여 복제가 중지될 수 있습니다. 가장 흔한 오류는 `packet bigger than 'max_allowed_packet' bytes`입니다. 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 MySQL용 InnoDB, MariaDB 10.2 이상용 InnoDB, MariaDB 10.1 이하용 XtraDB 스토리지 엔진에만 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 [Determination of Safe and Unsafe Statements in Binary Logging](#)을 참조하십시오.

다음 단계를 통해 복제 오류를 해결할 수 있습니다.

- 논리적 오류가 발생하고 이 오류를 건너뛰어도 안전할 경우 [현재 복제 오류 건너뛰기](#)에 설명된 단계를 따르십시오. Aurora MySQL DB 인스턴스에서는 `mysql_rds_skip_repl_error` 프로시저를 포함한 버전이 실행 중이어야 합니다. 자세한 정보는 `mysql_rds_skip_repl_error` 단원을 참조하십시오.
- 이진 로그(binlog) 위치 문제가 발생하는 경우 `mysql_rds_next_master_log` 명령으로 복제본 재생 위치를 변경할 수 있습니다. Aurora MySQL DB 인스턴스에서 복제본 재생 위치를 변경하려면 `mysql_rds_next_master_log` 명령을 지원하는 버전을 실행하고 있어야 합니다. 버전 정보는 `mysql_rds_next_master_log`를 참조하십시오.
- 높은 DML 부하 때문에 일시적인 성능 문제가 발생할 경우 읽기 전용 복제본의 DB 파라미터 그룹에서 `innodb_flush_log_at_trx_commit` 파라미터를 2로 설정할 수 있습니다. 그러면 일시적으로 원자성, 일관성, 격리성 및 내구성(ACID)이 감소하지만 읽기 전용 복제본이 변화를 따라잡는데 도움이 될 수 있습니다.
- 읽기 전용 복제본을 삭제하고 앤드포인트가 이전 읽기 전용 복제본의 앤드포인트와 동일하게 유지되도록 동일한 DB 인스턴스 식별자를 사용하여 인스턴스를 생성할 수 있습니다.

복제 오류가 해결되면 Replication State가 replicating으로 변경됩니다. 자세한 내용은 [MySQL 읽기 전용 복제본의 문제 해결](#)을 참조하십시오.

Slave Down 또는 Disabled 오류

`mysql.rds_skip_repl_error` 명령을 호출하면 다음 오류 메시지가 표시될 수 있습니다. `Slave is down or disabled.`

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 재실행되기 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 마스터에서 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 720시간(30일))을 함께 지정합니다. 다음 예제에서는 binlog 파일의 보관 기간을 48시간으로 설정합니다.

```
CALL mysql.rds_set_configuration('binlog retention hours', 48);
```

No Space Left on Device(장치에 남은 공간 없음) 오류

다음과 같은 오류 메시지가 표시될 수 있습니다.

- Amazon Aurora MySQL에서:

```
ERROR 3 (HY000): Error writing file '/rdsdbdata/tmp/XXXXXXXXX' (Errcode: 28 - No space left on device)
```

- Amazon Aurora PostgreSQL에서:

```
ERROR: could not write block XXXXXXXX of temporary file: No space left on device.
```

Amazon Aurora DB 클러스터의 각 DB 인스턴스에서 로컬 SSD(Solid State Drive) 스토리지를 사용하여 세션에 대한 임시 테이블을 저장합니다. 임시 테이블에 대한 이 로컬 스토리지는 Aurora 클러스터 볼륨처럼 자동으로 증가하지 않습니다. 대신 로컬 스토리지의 양이 제한됩니다. DB 클러스터 내의 DB 인스턴스에 대한 DB 인스턴스 클래스를 기반으로 제한됩니다.

임시 테이블 및 로그에 사용할 수 있는 스토리지 양을 표시하려면 CloudWatch 지표 `FreeLocalStorage`를 사용할 수 있습니다. 이 지표는 각 인스턴스의 임시 볼륨에 대한 것이며 클러스터 볼륨을 나타내는 것이 아닙니다. 사용 가능한 지표에 대한 자세한 내용은 [Amazon Aurora DB 클러스터 지표 모니터링 \(p. 346\)](#) 단원을 참조하십시오.

경우에 따라 워크로드를 수정하여 필요한 임시 스토리지 양을 줄일 수는 없습니다. 이 경우 로컬 SSD 스토리지가 더 많은 DB 인스턴스 클래스를 사용하도록 DB 인스턴스를 수정합니다. 자세한 내용은 [DB 인스턴스 클래스 \(p. 30\)](#) 단원을 참조하십시오.

문제 해결에 대한 자세한 내용은 MySQL용 Aurora의 로컬 스토리지에 무엇이 저장되며, 로컬 스토리지 문제를 해결하려면 어떻게 해야 합니까? 또는 Amazon Aurora for PostgreSQL 스토리지에는 어떤 항목이 저장되며, 스토리지 문제를 해결하려면 어떻게 해야 합니까?를 참조하십시오.

Amazon RDS 애플리케이션 프로그래밍 인터페이스(API) 레퍼런스

Amazon Relational Database Service(Amazon RDS)는 AWS Management 콘솔과 AWS Command Line Interface(AWS CLI) 외에 애플리케이션 프로그래밍 인터페이스(API)도 제공합니다. API를 사용하여 Amazon RDS의 DB 인스턴스 및 기타 객체 관리 작업을 자동화할 수 있습니다.

- API 작업의 알파벳 순 목록은 [작업](#)을 참조하십시오.
- 데이터 형식에 대한 알파벳 순 목록은 [데이터 형식](#)을 참조하십시오.
- 공통 쿼리 파라미터 목록은 [공통 파라미터](#)를 참조하십시오.
- 오류 코드에 대한 설명은 [공통 오류](#)를 참조하십시오.

AWS CLI에 대한 자세한 정보는 [Amazon RDS에 대한 AWS Command Line Interface 레퍼런스](#)를 참조하십시오.

주제

- [Query API 사용 \(p. 1036\)](#)
- [Aurora에서 애플리케이션 문제 해결 \(p. 1036\)](#)

Query API 사용

다음 섹션에서는 Query API와 함께 사용되는 파라미터 및 요청 인증에 대해 설명합니다.

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 단원에서 [Amazon Relational Database Service\(RDS\)](#)를 참조하십시오.

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 프로세스](#) 단원을 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 \(p. 1037\)](#)
- [문제 해결 팁 \(p. 1037\)](#)

Amazon RDS DB 인스턴스 문제 해결에 대한 자세한 내용은 [Aurora 문제 해결 \(p. 1028\)](#) 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 발생뿐만 아니라 오류 코드 및 메시지를 쉽게 검색할 수 있는 간단한 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.

- <http://statusaws.amazon.com>에서 Amazon RDS가 해당 AWS 리전에서 일반적으로 작동하는지 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 있을 지에 대해 미리 알아보려면 샘플 요청이나 사용자 시나리오를 살펴보고 이러한 샘플이 유사한 작업을 하고 있는지 확인하십시오.

- **포럼 확인**

Amazon RDS와 관련하여 다른 사람들이 경험한 문제에 대한 해결책을 검색할 수 있는 개발 커뮤니티 포럼이 있습니다. 포럼을 보려면

<https://forums.aws.amazon.com/>

문서 이력

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 설명서](#)를 참조하십시오.

Note

2018년 8월 31일 이전의 Amazon Aurora은 Amazon Relational Database Service 사용 설명서로 작성되었습니다. Aurora의 이전 문서 기록은 Amazon Relational Database Service User Guide의 [문서 기록](#)을 참조하십시오.

update-history-change	update-history-description	update-history-date
이제 Aurora Global Database에서 Aurora PostgreSQL 지원 (p. 1038)	이제 PostgreSQL 데이터베이스 엔진에 대한 Aurora Global Database를 생성할 수 있습니다. Aurora 글로벌 데이터베이스는 여러 AWS 리전에 걸쳐 있어, 자연 시간이 짧은 전역 읽기와 리전 전체의 중단으로부터 재해 복구를 지원합니다. 자세한 내용은 Amazon Aurora 글로벌 데이터베이스 작업 단원 을 참조하십시오.	March 10, 2020
Aurora MySQL 버전 1.22.2 (p. 723)	Aurora MySQL 버전 1.22.2를 사용할 수 있습니다.	March 5, 2020
Aurora MySQL 버전 1.20.1 (p. 729)	Aurora MySQL 버전 1.20.1을 사용할 수 있습니다.	March 5, 2020
Aurora MySQL 버전 1.19.6 (p. 731)	Aurora MySQL 버전 1.19.6을 사용할 수 있습니다.	March 5, 2020
Aurora MySQL 버전 1.17.9 (p. 736)	Aurora MySQL 버전 1.17.9를 사용할 수 있습니다.	March 5, 2020
Aurora PostgreSQL에 대한 메이저 버전 업그레이드 지원 (p. 1038)	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 업그레이드 를 참조하십시오.	March 4, 2020
Aurora PostgreSQL에서 Kerberos 인증 지원 (p. 1038)	이제 사용자가 Aurora PostgreSQL DB 클러스터에 연결할 때 Kerberos 인증을 사용하여 사용자를 인증할 수 있습니다. 자세한 내용은 Aurora PostgreSQL에서 Kerberos 인증 사용 을 참조하십시오.	February 28, 2020

Aurora PostgreSQL 버전 3.1, 2.4 및 1.6 (p. 1038)	PostgreSQL과 호환되는 Amazon Aurora의 새 버전으로는 버전 3.1(PostgreSQL 11.6과 호환), 버전 2.4(PostgreSQL 10.11과 호환) 및 버전 1.6(PostgreSQL 9.6.16과 호환)이 있습니다. 자세한 내용은 Amazon Aurora PostgreSQL 데이터베이스 엔진 버전 을 참조하십시오.	February 11, 2020
데이터 API에서 AWS PrivateLink 지원 (p. 1038)	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) 생성 을 참조하십시오.	February 6, 2020
Aurora PostgreSQL에서 Aurora 기계 학습 지원 (p. 1038)	aws_ml Aurora PostgreSQL 확장은 데이터베이스 쿼리에서 감성 분석을 위해 Amazon Comprehend를 호출하고 자체 기계 학습 모델을 실행하기 위해 Amazon SageMaker를 호출하는데 사용하는 함수를 제공합니다. 자세한 내용은 Aurora에서 기계 학습(ML) 기능 사용 단원을 참조하십시오.	February 5, 2020
Aurora PostgreSQL에서 Amazon S3으로의 데이터 내보내기 지원 (p. 1038)	PostgreSQL과 호환되는 Aurora DB 클러스터에서 데이터를 쿼리하여 Amazon S3 버킷에 저장된 파일로 직접 내보낼 수 있습니다. 자세한 내용은 Aurora PostgreSQL DB 클러스터에서 Amazon S3으로 데이터 내보내기를 참조하십시오.	February 5, 2020
DB 스냅샷 데이터를 Amazon S3로 내보내도록 지원 (p. 1038)	Amazon Aurora는 MySQL 및 PostgreSQL에서 Amazon S3로 기존의 DB 스냅샷 데이터를 내보낼 수 있도록 지원합니다. 자세한 내용은 Amazon S3로 DB 스냅샷 데이터 내보내기 를 참조하십시오.	January 9, 2020
Aurora MySQL 버전 2.07.1 (p. 690)	Aurora MySQL 버전 2.07.1을 사용할 수 있습니다.	December 23, 2019
Aurora MySQL 버전 1.22.1 (p. 724)	Aurora MySQL 버전 1.22.1을 사용할 수 있습니다.	December 23, 2019

문서 이력의 Aurora MySQL 릴리스 정보 (p. 1038)	이 단원에는 2018년 8월 31일 이후 출시된 버전의 MySQL과 호환되는 Aurora 릴리스 정보에 대한 기록 항목이 포함되어 있습니다. 특정 버전의 전체 릴리스 정보를 보려면 기록 항목의 첫 번째 열에 있는 링크를 선택합니다.	December 13, 2019
Amazon RDS Proxy (p. 1038)	Amazon RDS Proxy를 사용하면 클러스터에서 연결 관리의 오버 헤드를 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 엔드 포인트를 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 관리 를 참조하십시오.	December 3, 2019
Aurora Serverless용 데이터 API에서 데이터 형식 매핑 힌트 지원 (p. 1038)	이제 힌트를 사용하여 String 값 을 다른 형식으로 데이터베이스에 전송하도록 Aurora Serverless용 데이터 API에 지시할 수 있습니다. 자세한 내용은 데이터 API 호출 을 참조하십시오.	November 26, 2019
Aurora Serverless용 데이터 API에서 Java 클라이언트 라이브러리 (평가판) 지원 (p. 1038)	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 사용 을 참조하십시오.	November 26, 2019
Aurora PostgreSQL은 FedRAMP HIGH 사용 자격이 있음 (p. 1038)	Aurora PostgreSQL은 FedRAMP HIGH 사용 자격이 있습니다. AWS 및 규정 준수 활동에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내 AWS 서비스 를 참조하십시오.	November 26, 2019
Aurora PostgreSQL 버전 3.0 (p. 1038)	Amazon Aurora PostgreSQL 버전 3.0을 사용할 수 있으며 PostgreSQL 11.4와 호환됩니다. 지원되는 AWS 리전으로 us-east-1, us-east-2, us-west-2, eu-west-1, ap-northeast-1, ap-northeast-2 등이 있습니다. 자세한 내용은 Amazon Aurora PostgreSQL 데이터베이스 엔진 버전 을 참조하십시오.	November 26, 2019

이제 Aurora 글로벌 데이터베이스에 엔진 모드 설정이 필요하지 않음 (p. 1038)

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 Global Database 작업 단원](#)을 참조하십시오.

November 25, 2019

성능 개선 도우미에서 실행 중인 Aurora PostgreSQL 쿼리에 대한 통계 분석을 지원 (p. 1038)

이제 Aurora PostgreSQL DB 인스턴스용 성능 개선 도우미를 사용하여 실행 중인 쿼리의 통계를 분석할 수 있습니다. 자세한 내용은 [실행 중인 쿼리에 대한 통계 분석 단원](#)을 참조하십시오.

November 25, 2019

Aurora 글로벌 데이터베이스에 더 많은 클러스터 (p. 1038)

이제 여러 개의 보조 리전을 Aurora 글로벌 데이터베이스에 추가할 수 있습니다. 더 넓은 지리 영역에 걸쳐 짧은 자연 시간 전역 읽기 및 재해 복구를 이용할 수 있습니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 [Amazon Aurora 글로벌 데이터베이스 작업 단원](#)을 참조하십시오.

November 25, 2019

Aurora 글로벌 데이터베이스는 Aurora MySQL 버전 2에서 사용 가능 (p. 1038)

Aurora MySQL 2.07에서부터 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 Global Database 작업 단원](#)을 참조하십시오.

November 25, 2019

Aurora MySQL에서 Aurora 기계 학습 지원 (p. 1038)	Aurora MySQL 2.07 이상에 서 감성 분석에 대해 Amazon Comprehend를, 광범위한 기계 학습 알고리즘에 대해 Amazon SageMaker를 호출할 수 있습니다. 쿼리의 저장 함수에 대한 호출을 포함하여 데이터베이스 애플리케이션에서 직접 결과를 사용하십시오. 자세한 내용은 Aurora에서 기계 학습(ML) 기능 사용 단원 을 참조하십시오.	November 25, 2019
Aurora MySQL 버전 2.07.0 (p. 691)	Aurora MySQL 버전 2.07.0을 사용할 수 있습니다.	November 25, 2019
Aurora MySQL 버전 1.22.0 (p. 724)	Aurora MySQL 버전 1.22.0을 사용할 수 있습니다.	November 25, 2019
Aurora MySQL 버전 1.21.0 (p. 727)	Aurora MySQL 버전 1.21.0을 사용할 수 있습니다.	November 25, 2019
Amazon Aurora MySQL 복제본에 대해 활성화된 READ COMMITTED 격리 수준 (p. 1038)	이제 Aurora MySQL 복제본에서 READ COMMITTED 격리 수준을 활성화할 수 있습니다. 이렇게 하려면 인스턴스 수준에서 <code>aurora_read_replica_read_committed_isolation_enabled</code> 구성 설정을 활성화해야 합니다. OLTP 클러스터의 장기 실행 쿼리에 READ COMMITTED 격리 수준을 사용하면 기록 목록 길이와 관련된 문제를 해결하는데 도움이 됩니다. 이 설정을 활성화하기 전에 Aurora 복제본의 격리 동작이 READ COMMITTED의 평상시 MySQL 구현과 어떻게 다른지 이해해야 합니다. 자세한 내용은 Aurora MySQL 격리 수준 단원 을 참조하십시오.	November 25, 2019
Aurora MySQL 버전 2.06.0 (p. 693)	Aurora MySQL 버전 2.06.0을 사용할 수 있습니다.	November 22, 2019
Aurora MySQL 버전 2.04.8 (p. 698)	Aurora MySQL 버전 2.04.8을 사용할 수 있습니다.	November 20, 2019
랩 모드 없이도 Aurora MySQL 해시 조인 사용 가능 (p. 1038)	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 이 기능으로 쿼리 성능을 향상시킬 수 있습니다. 이 기능 사용에 대한 자세한 내용은 Aurora MySQL의 해시 조인 작업 단원 을 참조하십시오.	November 19, 2019

랩 모드 없이도 Aurora MySQL 핫 행 경합 최적화 사용 가 능 (p. 1038)	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 향상 시킵니다. 이러한 향상에는 Aurora MySQL에서 사용하는 잠금 해제 알고리즘의 변경이 수반됩니다.	November 19, 2019
Aurora MySQL 2.*에서 역추적 지 원 (p. 1038)	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 완료 할 수 있습니다. AWS 블로그 에서 개요를 읽고 자세한 내용은 Aurora DB 클러스터 역추적 을 참조하십시오.	November 19, 2019
Aurora MySQL 2.*에서 더 많 은 db.r5 인스턴스 클래스 지 원 (p. 1038)	Aurora MySQL 클러스터는 이제 db.r5.8xlarge, db.r5.16xlarge, db.r5.24xlarge 인스턴스 유형을 지원합니다. Aurora MySQL 클러스터의 인스턴스 유형에 대한 자세한 내용은 DB 인스턴스 클래스 선택 단원을 참조하세요.	November 19, 2019
Aurora MySQL 버전 2.04.7 (p. 699)	Aurora MySQL 버전 2.04.7을 사용할 수 있습니다.	November 14, 2019
Aurora MySQL 버전 2.05.0 (p. 696)	Aurora MySQL 버전 2.05.0을 사용할 수 있습니다.	November 11, 2019
Aurora MySQL 버전 1.20.0 (p. 729)	Aurora MySQL 버전 1.20.0을 사용할 수 있습니다.	November 11, 2019
Aurora에 대한 청구 태그 지 원 (p. 1038)	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 태그 지정 을 참조하십시오. 태그 및 비용 분석에 태그를 사용하는 방법에 대한 일반적인 내용은 비용 할당 태그 사용 및 사용자 정의 비용 할당 태그를 참조하십시오.	October 23, 2019

Aurora PostgreSQL용 데이터 API (p. 1038)	이제 Aurora PostgreSQL에서는 Amazon Aurora Serverless DB 클러스터에서 데이터 API를 사용할 수 있도록 지원합니다. 자세한 내용은 Aurora Serverless용 데이터 API 사용 단원을 참조하십시오.	September 23, 2019
Aurora MySQL 버전 2.04.6 (p. 701)	Aurora MySQL 버전 2.04.6을 사용할 수 있습니다.	September 19, 2019
Aurora MySQL 버전 1.19.5 (p. 731)	Aurora MySQL 버전 1.19.5를 사용할 수 있습니다.	September 19, 2019
Aurora PostgreSQL가 CloudWatch 로그에 데이터베이스 로그 업로드 지원 (p. 1038)	Amazon CloudWatch Logs의 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 게시 를 참조하십시오.	August 9, 2019
Aurora MySQL를 위한 멀티 마스터 클러스터 (p. 1038)	Aurora MySQL 멀티 마스터 클러스터를 설정할 수 있습니다. 멀티 마스터 클러스터에서는 각각의 DB 인스턴스가 읽기-쓰기 기능을 가집니다. 자세한 내용은 Aurora 멀티 마스터 클러스터 작업 을 참조하십시오.	August 8, 2019
Aurora PostgreSQL에서 Aurora Serverless 지원 (p. 1038)	이제 Aurora PostgreSQL에서 Amazon Aurora Serverless를 사용할 수 있습니다. Aurora Serverless DB 클러스터는 애플리케이션 요구 사항을 기반으로 컴퓨팅 용량을 자동으로 시작, 종료, 확장 또는 축소합니다. 자세한 내용은 Amazon Aurora Serverless 사용 단원을 참조하십시오.	July 9, 2019
Aurora MySQL 버전 2.04.5 (p. 702)	Aurora MySQL 버전 2.04.5를 사용할 수 있습니다.	July 8, 2019

Aurora PostgreSQL 버전 2.3.3 및 1.5.2 (p. 1038)	PostgreSQL과 호환되는 Amazon Aurora 버전 2.3.3은 PostgreSQL 10.7와 호환하여 사용할 수 있습니다. PostgreSQL과 호환되는 Amazon Aurora 버전 1.5.2는 PostgreSQL 9.6.12와 호환하여 사용할 수 있습니다. 자세한 내용은 Amazon Aurora PostgreSQL 데이터베이스 엔진 버전 을 참조하십시오.	July 3, 2019
계정 간 Aurora MySQL 복제 (p. 1038)	이제 AWS 계정 사이에 Aurora MySQL DB 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.	July 2, 2019
Aurora PostgreSQL 버전 2.3.1 및 1.5.1 (p. 1038)	PostgreSQL과 호환되는 Amazon Aurora 버전 2.3.1은 PostgreSQL 10.7와 호환하여 사용할 수 있습니다. PostgreSQL과 호환되는 Amazon Aurora 버전 1.5.1은 PostgreSQL 9.6.12와 호환하여 사용할 수 있습니다. 자세한 내용은 Amazon Aurora PostgreSQL 데이터베이스 엔진 버전 을 참조하십시오.	July 2, 2019
db.t3 DB 인스턴스 클래스를 지원하는 Aurora PostgreSQL (p. 1038)	이제 db.t3 DB 인스턴스 클래스를 사용하는 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 단원을 참조하십시오.	June 20, 2019
Amazon S3에서 Aurora PostgreSQL에 필요한 데이터 가져오기 지원 (p. 1038)	이제 Amazon S3 파일의 데이터를 Aurora PostgreSQL DB 클러스터의 테이블로 가져올 수 있습니다. 자세한 내용은 Amazon S3 데이터를 Aurora PostgreSQL DB 클러스터로 가져오기 를 참조하십시오.	June 19, 2019
Aurora PostgreSQL에서는 이제 클러스터 캐시 관리를 통해 장애를 신속하게 복구합니다. (p. 1038)	이제 PostgreSQL과 호환되는 Aurora에서는 장애 조치가 이뤄지는 경우 클러스터 캐시 관리를 제공하여 기본 DB 인스턴스의 신속한 복구를 보장합니다. 자세한 내용은 장애 조치 후 클러스터 캐시 관리를 통한 신속한 복구 단원을 참조하십시오.	June 11, 2019

Aurora MySQL 버전 1.19.2 (p. 732)	Aurora MySQL 버전 1.19.2를 사용할 수 있습니다.	June 5, 2019
Aurora Serverless용 데이터 API를 일반적으로 사용 가능 (p. 1038)	데이터 API를 사용하여 웹 서비스 기반 애플리케이션으로 Aurora Serverless 클러스터에 액세스 할 수 있습니다. 자세한 내용은 Aurora Serverless용 데이터 API 사용 단원을 참조하십시오.	May 30, 2019
Aurora PostgreSQL, 데이터베이스 활동 스트림으로 데이터베이스 모니터링 지원 (p. 1038)	이제 PostgreSQL과 호환되는 Aurora에 데이터베이스 활동 스트림이 포함되어, 관계형 데이터베이스에서 데이터베이스 활동의 데이터 스트림을 거의 실시간으로 제공하게 됩니다. 자세한 내용은 데이터베이스 활동 스트림 사용 을 참조하십시오.	May 30, 2019
Aurora PostgreSQL 버전 2.3 (p. 1038)	PostgreSQL과 호환되는 Amazon Aurora 버전 2.3이 사용 가능하며 PostgreSQL 10.7과 호환됩니다. 자세한 내용은 버전 2.3 단원을 참조하십시오.	May 30, 2019
Aurora MySQL 버전 2.04.4 (p. 704)	Aurora MySQL 버전 2.04.4를 사용할 수 있습니다.	May 29, 2019
Amazon Aurora 권장 사항: (p. 1038)	Amazon Aurora에서 이제 Aurora 리소스에 대한 자동 권장 사항을 제공합니다. 자세한 내용은 Amazon Aurora 권장 사항 사용 단원을 참조하십시오.	May 22, 2019
Aurora PostgreSQL 버전 1.2.2, 1.3.2, 2.0.1, 2.1.1, 2.2.1 (p. 1038)	PostgreSQL과 호환되는 Amazon Aurora에 대한 다음 패치 버전은 현재 사용 가능하며 1.2.2, 1.3.2, 2.0.1, 2.1.1, 2.2.1 버전을 포함합니다. 자세한 내용은 Amazon Aurora PostgreSQL의 데이터베이스 엔진 버전 단원을 참조하십시오.	May 21, 2019
Aurora Global Database에 대한 성능 개선 도우미 지원 (p. 1038)	이제 Aurora Global Database에서 성능 개선 도우미를 사용할 수 있습니다. Aurora용 성능 개선 도우미에 대한 자세한 내용은 Amazon RDS 성능 개선 도우미 사용 단원을 참조하십시오. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 Aurora Global Database 작업 단원을 참조하십시오.	May 13, 2019
Aurora PostgreSQL 버전 1.4 (p. 1038)	PostgreSQL과 호환되는 Amazon Aurora 버전 1.4는 현재 사용 가능하며 PostgreSQL 9.6.11과 호환됩니다. 자세한 내용은 버전 1.4 단원을 참조하십시오.	May 9, 2019

Aurora MySQL 버전 2.04.3 (p. 705)	Aurora MySQL 버전 2.04.3을 사용할 수 있습니다.	May 9, 2019
Aurora MySQL 버전 1.19.1 (p. 733)	Aurora MySQL 버전 1.19.1을 사용할 수 있습니다.	May 9, 2019
Aurora MySQL 5.7에 성능 개선 도우미를 사용할 수 있습니다. (p. 1038)	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 사용 단원을 참조하십시오.	May 3, 2019
Aurora MySQL 버전 2.04.2 (p. 706)	Aurora MySQL 버전 2.04.2를 사용할 수 있습니다.	May 2, 2019
Aurora 글로벌 데이터베이스를 더 많은 AWS 리전에서 사용 가능 (p. 1038)	이제 Aurora를 사용할 수 있는 대부분의 AWS 리전에서 Aurora 글로벌 데이터베이스를 생성할 수 있습니다. Aurora 글로벌 데이터베이스에 대한 자세한 내용은 Amazon Aurora Global Database 작업 단원을 참조하십시오.	April 30, 2019
Aurora Serverless에 최소 용량 적용 (p. 1038)	Aurora Serverless 클러스터에 사용할 수 있는 최소 용량 설정 값은 1입니다. 전에는 최소 용량이 2였습니다. Aurora 서비스 용량 값 지정에 대한 자세한 내용은 Aurora Serverless DB 클러스터의 용량 설정 단원을 참조하십시오.	April 29, 2019
Aurora Serverless 제한 시간 조치 (p. 1038)	이제 Aurora Serverless 용량 변경 제한 시간이 초과되었을 때 취할 조치를 지정할 수 있습니다. 자세한 내용은 용량 변경에 대한 제한 시간 조치 단원을 참조하십시오.	April 29, 2019
초당 청구 (p. 1038)	Amazon RDS는 이제 온디맨드 인스턴스의 경우 AWS GovCloud (US)를 제외한 모든 AWS 리전에서 1초 증분 단위로 청구됩니다. 자세한 내용은 Aurora에 대한 DB 인스턴스 청구 단원을 참조하십시오.	April 25, 2019
Amazon S3에서 MySQL 5.7 백업 복원 (p. 1038)	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 마이그레이션 단원을 참조하십시오.	April 17, 2019

AWS 리전 간 Aurora Serverless 스냅샷 공유 (p. 1038)	Aurora Serverless에서 스냅샷은 항상 암호화됩니다. 자신의 AWS Key Management Service 키로 스냅샷을 암호화하는 경우 이제는 AWS 리전 간에 스냅샷을 복사하거나 공유할 수 있습니다. Aurora Serverless DB 클러스터의 스냅샷에 대한 자세한 내용은 Aurora Serverless 및 스냅샷 단원을 참조하십시오.	April 17, 2019
리전 간 Aurora Serverless 스냅샷 공유 (p. 1038)	Aurora Serverless에서 스냅샷은 항상 암호화됩니다. 자신의 KMS 키로 스냅샷을 암호화하는 경우 이제는 리전 간에 스냅샷을 복사하거나 공유할 수 있습니다. Aurora Serverless DB 클러스터의 스냅샷에 대한 자세한 내용은 Aurora 서비스 및 스냅샷 단원을 참조하십시오.	April 16, 2019
Aurora 개념 증명 자습서 (p. 1038)	개념 증명을 수행하여 Aurora에서 애플리케이션 및 워크로드를 시험하는 방법을 배울 수 있습니다. 전체 자습서는 Aurora 개념 증명 수행 단원을 참조하십시오.	April 16, 2019
Amazon S3 백업에서 복원하는 기능을 지원하는 Aurora Serverless (p. 1038)	이제 Amazon S3에서 Aurora Serverless 클러스터로 백업을 가져올 수 있습니다. 이 절차에 관한 자세한 내용은 Amazon S3 버킷을 사용해 MySQL에서 데이터 마이그레이션하기 를 참조하세요.	April 16, 2019
Aurora Serverless를 위한 새로운 수정 가능 파라미터 (p. 1038)	이제 Aurora Serverless 클러스터에 대한 <code>innodb_file_format</code> , <code>innodb_file_per_table</code> , <code>innodb_large_prefix</code> , <code>innodb_lock_wait_timeout</code> , <code>innodb_monitor_disable</code> , <code>innodb_monitor_enable</code> , <code>innodb_monitor_reset</code> , <code>innodb_monitor_reset_all</code> , <code>innodb_print_all_deadlocks</code> , <code>log_warnings</code> , <code>net_read_timeout</code> , <code>net_retry_count</code> , <code>net_write_timeout</code> , <code>sql_mode</code> , <code>tx_isolation</code> DB 파라미터를 수정할 수 있습니다. Aurora Serverless 클러스터용 구성 파라미터에 대한 자세한 내용은 Aurora Serverless 및 파라미터 그룹 단원을 참조하십시오.	April 4, 2019

Aurora PostgreSQL의 db.r5 DB 인스턴스 클래스 지원 (p. 1038)	이제 db.r5 DB 인스턴스 클래스를 사용하는 Aurora PostgreSQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 단원 을 참조하십시오.	April 4, 2019
Aurora PostgreSQL 논리적 복제 (p. 1038)	이제 PostgreSQL 논리적 복제를 사용해 Aurora PostgreSQL DB 클러스터용 데이터베이스의 일부를 복제할 수 있습니다. 자세한 내용은 PostgreSQL 논리적 복제 사용 단원 을 참조하십시오.	March 28, 2019
Aurora MySQL 버전 2.04.1 (p. 708)	Aurora MySQL 버전 2.04.1을 사용할 수 있습니다.	March 25, 2019
Aurora MySQL 버전 2.04 (p. 708)	Aurora MySQL 버전 2.04를 사용할 수 있습니다.	March 25, 2019
Aurora MySQL 2.04에 대한 GTID 지원 (p. 1038)	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 사용 단원 을 참조하십시오.	March 25, 2019
Aurora Serverless 로그를 Amazon CloudWatch에 업로드 (p. 1038)	이제 Aurora에서 데이터베이스 로그를 Aurora Serverless 클러스터용 CloudWatch에 업로드하도록 할 수 있습니다. 자세한 내용은 Aurora 서비스 DB 클러스터 단원 을 참조하십시오. 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 작업 단원 을 참조하십시오.	February 25, 2019
Aurora MySQL의 db.t3 DB 인스턴스 클래스 지원 (p. 1038)	이제 db.t3 DB 인스턴스 클래스를 사용하는 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 단원 을 참조하십시오.	February 25, 2019

Aurora MySQL의 db.r5 DB 인스턴스 클래스 지원 (p. 1038)	이제 db.r5 DB 인스턴스 클래스를 사용하는 Aurora MySQL DB 클러스터를 생성할 수 있습니다. 자세한 내용은 DB 인스턴스 클래스 단원을 참조하십시오.	February 25, 2019
Aurora MySQL용 성능 개선 도우미 카운터 (p. 1038)	이제 Aurora MySQL DB 인스턴스 용 성능 개선 도우미 차트에 성능 카운터를 추가할 수 있습니다. 자세한 내용은 성능 개선 도우미 대시보드 구성 요소 단원을 참조하십시오.	February 19, 2019
Aurora PostgreSQL 버전 2.2.0 (p. 1038)	PostgreSQL과 호환되는 Aurora 버전 2.2.0이 사용 가능하며 PostgreSQL 10.6과 호환됩니다. 자세한 내용은 버전 2.2.0 단원을 참조하십시오.	February 13, 2019
Aurora MySQL 버전 2.03.4 (p. 709)	Aurora MySQL 버전 2.03.4를 사용할 수 있습니다.	February 7, 2019
Aurora MySQL 버전 1.19.0 (p. 733)	Aurora MySQL 버전 1.19.0을 사용할 수 있습니다.	February 7, 2019
Amazon RDS 성능 개선 도우미에서 Aurora MySQL에 대해 더 많은 SQL 텍스트 보기기를 지원 (p. 1038)	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.	February 6, 2019
Amazon RDS 성능 개선 도우미에서 Aurora PostgreSQL에 대해 더 많은 SQL 텍스트 보기기를 지원 (p. 1038)	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.	January 24, 2019
Aurora MySQL 버전 2.03.3 (p. 710)	Aurora MySQL 버전 2.03.3을 사용할 수 있습니다.	January 18, 2019
Aurora MySQL 버전 1.17.8 (p. 736)	Aurora MySQL 버전 1.17.8을 사용할 수 있습니다.	January 17, 2019
Aurora MySQL 버전 2.03.2 (p. 711)	Aurora MySQL 버전 2.03.2를 사용할 수 있습니다.	January 9, 2019

Aurora 백업 결제 (p. 1038)	Amazon CloudWatch 지표 TotalBackupStorageBilled, SnapshotStorageUsed 및 BackupRetentionPeriodStorageUsed를 사용하여 Aurora 백업의 공간 사 용량을 모니터링할 수 있습니다. CloudWatch 지표를 사용하는 방 법에 대한 자세한 내용은 모니 터링 개요 를 참조하십시오. 백 업 데이터용 스토리지를 관리하 는 방법에 대한 자세한 내용은 Understanding Aurora Backup Storage Usage 단원을 참조하십 시오.	January 3, 2019
성능 개선 도우미 카운터 (p. 1038)	이제 성능 개선 도우미 차트에 성 능 카운터를 추가할 수 있습니다. 자세한 내용은 성능 개선 도우미 대시보드 구성 요소 단원을 참조 하십시오.	December 6, 2018
Aurora 글로벌 데이터베이 스 (p. 1038)	이제 Aurora 글로벌 데이터베이스 를 만들 수 있습니다. Aurora 글로 벌 데이터베이스는 여러 AWS 리 전에 걸쳐 있어, 자연 시간이 짧은 전역 읽기와 리전 전체의 종단으 로부터 재해 복구를 지원합니다. 자세한 내용은 Amazon Aurora 글 로벌 데이터베이스 작업 단원을 참조하십시오.	November 28, 2018
Aurora Serverless용 쿼리 편집기 (베타) (p. 1038)	Aurora Serverless 클러스터의 Amazon RDS 콘솔에서 SQL 문을 실행할 수 있습니다. 자세한 내용 은 Aurora Serverless용 쿼리 편집 기 사용 단원을 참조하십시오.	November 20, 2018
Aurora Serverless용 데이터 API(베타) (p. 1038)	데이터 API를 사용하여 웹 서비 스 기반 애플리케이션으로 Aurora Serverless 클러스터에 액세스 할 수 있습니다. 자세한 내용은 Using the Data API for Aurora Serverless 단원을 참조하십시오.	November 20, 2018
Aurora PostgreSQL의 쿼리 계획 관리 (p. 1038)	PostgreSQL과 호환되는 Aurora 이 이제 PostgreSQL 쿼리 실행 계획을 관리하는 데 사용할 수 있는 쿼리 계획 관리를 제공합 니다. 자세한 내용은 Managing Query Execution Plans for Aurora PostgreSQL 단원을 참조하십 시오.	November 20, 2018
Aurora PostgreSQL 버전 2.1 (p. 1038)	PostgreSQL과 호환되는 Aurora 버전 2.1을 사용 가능하며 PostgreSQL 10.5와 호환됩니다. 자세한 내용은 버전 2.1 단원을 참조하십시오.	November 20, 2018

Aurora Serverless에 대한 TLS 지원 (p. 1038)	Aurora Serverless 클러스터에서 TLS/SSL 암호화를 지원합니다. 자세한 내용은 TLS/SSL for Aurora Serverless 단원을 참조하십시오.	November 19, 2018
사용자 지정 엔드포인트 (p. 1038)	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 Connection Management 단원을 참조하십시오.	November 12, 2018
Aurora PostgreSQL의 IAM 인증 지원 (p. 1038)	PostgreSQL과 호환되는 Aurora이 이제 IAM 인증을 지원합니다. 자세한 내용은 IAM Database Authentication 단원을 참조하십시오.	November 8, 2018
Aurora MySQL 버전 2.03.1 (p. 711)	Aurora MySQL 버전 2.03.1을 사용할 수 있습니다.	October 24, 2018
복원 및 특정 시점으로 복구를 위한 사용자 지정 파라미터 그룹 (p. 1038)	이제 스냅샷을 복원하거나 특정 시점으로 복구 작업을 수행할 때 사용자 지정 파라미터 그룹을 지정할 수 있습니다. 자세한 내용은 DB 클러스터 스냅샷으로 복구 및 DB 클러스터를 지정된 시간으로 복구 단원을 참조하십시오.	October 15, 2018
Aurora MySQL 버전 2.03 (p. 712)	Aurora MySQL 버전 2.03을 사용할 수 있습니다.	October 11, 2018
Aurora MySQL 버전 2.02.5 (p. 713)	Aurora MySQL 버전 2.02.5를 사용할 수 있습니다.	October 8, 2018
Aurora MySQL 버전 1.17.7 (p. 737)	Aurora MySQL 버전 1.17.7을 사용할 수 있습니다.	October 8, 2018
Aurora DB 클러스터에 대한 삭제 방지 (p. 1038)	DB 클러스터에 대해 삭제 방지를 활성화하면 모든 사용자가 데이터베이스를 삭제할 수 없습니다. 자세한 내용은 DB 클러스터에서 DB 인스턴스 삭제 단원을 참조하십시오.	September 26, 2018
Aurora PostgreSQL 버전 2.0 (p. 1038)	PostgreSQL과 호환되는 Aurora 버전 2.0을 사용 가능하며 PostgreSQL 10.4와 호환됩니다. 자세한 내용은 버전 2.0 단원을 참조하십시오.	September 25, 2018

중지/시작 기능 Aurora (p. 1038)	이제 단일 작업으로 전체 Aurora 클러스터를 중지하거나 시작할 수 있습니다. 자세한 내용은 Aurora 클러스터 중단 및 시작 단원을 참조하십시오.	September 24, 2018
Aurora MySQL 버전 2.02.4 (p. 713)	Aurora MySQL 버전 2.02.4를 사용할 수 있습니다.	September 21, 2018
Aurora MySQL용 병렬 쿼리 기능 (p. 1038)	Aurora MySQL은 이제 Aurora 스토리지 인프라에서 쿼리에 대한 I/O 작업을 병렬화하는 옵션을 제공합니다. 이 기능은 데이터 집약적인 분석 쿼리의 속도를 높이는 데 이는 워크로드에서 가장 시간이 많이 걸리는 작업입니다. 자세한 내용은 Aurora MySQL에 대한 병렬 쿼리 작업 .	September 20, 2018
Aurora MySQL 버전 1.18.0 (p. 735)	Aurora MySQL 버전 1.18.0을 사용할 수 있습니다.	September 20, 2018
Aurora PostgreSQL 버전 1.3 (p. 1038)	이제 Aurora PostgreSQL 버전 1.3을 사용할 수 있으며 이 버전은 PostgreSQL 9.6.9와 호환됩니다. 자세한 내용은 버전 1.3 단원을 참조하십시오.	September 11, 2018
Aurora MySQL 버전 1.17.6 (p. 737)	Aurora MySQL 버전 1.17.6을 사용할 수 있습니다.	September 6, 2018
새 설명서 (p. 1038)	이 설명서는 Amazon Aurora 사용 설명서의 첫 번째 릴리스입니다.	August 31, 2018

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the AWS General Reference.