Amazon Aurora로의 데이터베이스 마이그레이션

2016년 6월

고지 사항

이 문서는 정보 제공 목적으로만 제공됩니다. 본 문서의 발행일 당시 AWS의 현재 제품 및 실행방법을 설명하며, 예고 없이 변경될 수 있습니다. 고객은 본 문서에 포함된 정보나 AWS 제품 또는 서비스의 사용을 독립적으로 평가할 책임이 있으며, 각 정보 및 제품은 명시적이든 묵시적이든 어떠한 종류의 보증 없이 "있는 그대로" 제공됩니다. 본 문서는 AWS, 그 계열사, 공급업체 또는 라이선스 제공자로부터 어떠한 보증, 표현, 계약 약속, 조건 또는 보증을 구성하지 않습니다. 고객에 대한 AWS의 책임 및 의무는 AWS 계약에 준거합니다. 본 문서는 AWS와 고객 간의 어떠한 계약도 구성하지 않으며 이를 변경하지도 않습니다.

목차

요약

Amazon Aurora는 MySQL과 호환 가능한 엔터프라이즈급 관계형 데이터베이스 엔진입니다. Amazon Aurora는 기존 관계형 데이터베이스 엔진의 한계를 대폭 개선한 클라우드 네이티브 데이터베이스입니다. 본 백서는 기존 데이터베이스를 Amazon Aurora로 마이그레이션하는 모범 사례를 설명하고자 준비되었으며 애플리케이션의 운영 중단을 최소화하면서 오픈 소스 및 상용 데이터베이스를 Amazon Aurora로 마이그레이션하기 위한 마이그레이션 고려 사항과 단계별 프로세스를 제시합니다.

Amazon Aurora 소개

기존의 관계형 데이터베이스는 수십 년 동안 데이터 스토리지 및 지속성을 위한 기본 솔루션으로 사용되어 왔습니다. 이러한 데이터베이스 시스템은 획일화된 아키텍처에 의존하며 클라우드 인프라의 장점을 활용하도록 설계되어 있지 않습니다. 이러한 획일적 아키텍처는 비용, 유연성 및 가용성 영역을 비롯한 다양한 부문에서 많은 문제를 발생시킵니다. 이러한 문제를 해결하기 위해 AWS는 클라우드 인프라를 위해 새롭게 설계된 관계형 데이터베이스인 Amazon Aurora를 출시했습니다.

Amazon Aurora는 MySQL과 호환 가능한 관계형 데이터베이스 엔진으로서 하이엔드 상용 데이터베이스의 속도, 가용성 및 보안 성능과 오픈 소스 데이터베이스의 간편성 및 비용 효율성을 결합했습니다. Aurora는 MySQL에 비해 최대 5배 높은, 하이엔드 상용 데이터베이스 수준의 성능을 제공합니다. Amazon Aurora의 가격은 상용 엔진의 1/10 수준으로 책정되어 있습니다.

Amazon Aurora는 Amazon Relational Database Service(Amazon RDS) 플랫폼을 통해 제공됩니다. 다른 Amazon RDS 데이터베이스와 같이 Aurora는 완전관리형 데이터베이스 서비스입니다. Amazon RDS 플랫폼을 사용하면 하드웨어

프로비저닝, 소프트웨어 패치, 설정, 구성, 모니터링 및 백업과 같은 대부분의 데이터베이스 관리 작업이 완전히 자동화됩니다.

Amazon Aurora는 미션 크리티컬 워크로드를 위해 설계되어 기본적으로 고가용성을 제공합니다. 특정 리전 내의 여러 가용 영역에 걸쳐 구성되는 Aurora 데이터베이스 클러스터는 여러 물리적 데이터 센터에 대한 즉각적인 내구성 및 내결함성을 제공합니다. 가용 영역은 Amazon에서 운영하는 하나 이상의 고가용성 데이터 센터로 구성됩니다. 가용 영역은 서로 격리되며 지연 시간이 짧은 링크를 통해 연결됩니다. 데이터베이스 볼륨의 각 세그먼트는 이러한 가용 영역 전체에 걸쳐 6회 복제됩니다.

Aurora 클러스터 볼륨은 데이터베이스의 데이터 양이 늘어남에 따라 성능이나 가용성에 영향을 미치지 않고 자동으로 증가하므로 사전에 대규모 데이터베이스 용량을 예측하거나 프로비저닝할 필요가 없습니다. 단일 Aurora 클러스터 볼륨은 최대 64테라바이트(TB)까지 확장할 수 있으며 Aurora 클러스터 볼륨에서 사용한 공간에 대해서만 사용자에게 요금이 청구됩니다.

Aurora의 자동 백업 기능은 특정 시점으로의 데이터 복구를 지원하므로 데이터베이스를 보존 기간 중 어느 시점으로나 복원할 수 있습니다(최근 5분 전까지 가능). 자동 백업은 99.999999999%의 내구성을 제공하도록 설계된 Amazon Simple Storage Service(Amazon S3)에 저장됩니다. Amazon Aurora 백업은 지속적인 자동 증분 방식의 백업이며 데이터베이스 성능에 영향을 미치지 않습니다.

읽기 전용 복제본이 필요한 애플리케이션의 경우 각 Aurora 데이터베이스별로 매우 짧은 복제 지연 시간으로 최대 15개의 Aurora 복제본을 만들 수 있습니다. 이 복제본은 소스 인스턴스와 동일한 기본 스토리지를 공유하므로 비용을 낮춰주며 복제본 노드에서 쓰기를 수행할 필요가 없습니다.

Amazon Aurora는 매우 안전하며 AWS Key Management Service(KMS)를 통해 생성 및 관리되는 키를 사용하여 데이터베이스를 암호화할 수 있습니다. Amazon Aurora 암호화를 실행 중인 데이터베이스 인스턴스에서는 기본 스토리지에 저장된 데이터가 암호화되며, 동일한 클러스터에 있는 자동화된 백업, 스냅샷 및 복제본 또한 암호화됩니다. Amazon Aurora는 SSL(AES-256)을 사용하여 전송 중인 데이터를 보호합니다.

Aurora 기능에 대한 전체 목록은 Amazon Aurora 제품 페이지를 참조하십시오.[1] 풍부한 기능과 비용 효율성을 제공하는 Amazon Aurora는 미션 크리티컬 애플리케이션을 위한 필수 데이터베이스로 인기가 올라가고 있습니다.

데이터베이스 마이그레이션 고려 사항

데이터베이스는 대부분의 애플리케이션 아키텍처에서 중요한 구성 요소입니다. 새로운 플랫폼으로의 데이터베이스 마이그레이션은 애플리케이션 수명 주기에서 중요한 이벤트이며 애플리케이션의 기능, 성능 및 안정성에 영향을 미칠 수 있습니다. Amazon Aurora로의 첫 번째 마이그레이션 프로젝트를 시작하기 전에 몇 가지 중요한 사항을 고려해야 합니다.

마이그레이션 단계

데이터베이스 마이그레이션을 복잡한 과정이므로 반복 가능한 단계별 접근 방식이 권장됩니다.

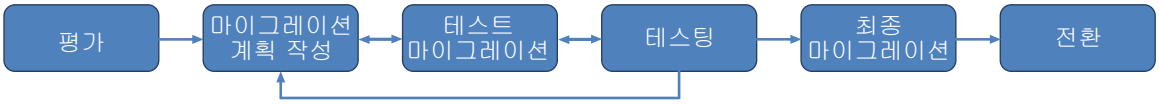

그림 1: 마이그레이션 단계

애플리케이션 고려 사항

Aurora 기능 평가

대부분의 애플리케이션은 다양한 관계형 데이터베이스 엔진과 연동되도록 설계되어 있지만 해당 애플리케이션이 구체적으로 Amazon Aurora와 연동되는지 확인해야 합니다. Amazon Aurora는 MySQL 5.6과 호환되도록 설계되었습니다. 따라서 현재 MySQL 데이터베이스에서 사용되는 대부분의 코드와 애플리케이션, 드라이버 및 도구를 거의 또는 전혀 변경하지 않고 그대로 Aurora에서 사용할 수 있습니다.

하지만 MyISAM 스토리지 엔진과 같은 일부 MySQL 기능은 Amazon Aurora에서 제공되지 않습니다. 또한, Aurora 서비스의 관리형 특성으로 인해 데이터베이스 노드에 대한 SSH 액세스가 제한되므로 데이터베이스 호스트에 타사 도구 또는 플러그인을 설치하는 기능에 영향을 미칠 수 있습니다.

성능 고려 사항

데이터베이스 성능은 데이터베이스를 새로운 플랫폼으로 마이그레이션할 때 고려할 핵심 사항 중 하나입니다. 따라서 많은 성공적인 데이터베이스 마이그레이션 프로젝트는 새로운 데이터베이스 플랫폼의 성능 평가에서 시작됩니다. Sysbench 및 TPC-C 벤치마크 실행을 통해 전반적인 데이터 성능에 대한 상당한 정보를 수집할 수 있지만 이러한 벤치마크가 특정 애플리케이션의

데이터 액세스 패턴을 정확하게 에뮬레이션해 주지는 않습니다. 보다 유용한 결과를 얻으려면 새 플랫폼에서 직접 쿼리(또는 쿼리의 일부)를 실행하여 시간에 민감한 워크로드의 성능을 테스트해야 합니다.

다음과 같은 전략을 고려하십시오.

- 현재 데이터베이스가 MySQL인 경우 다운타임을 두고 데이터베이스를 Amazon Aurora로 마이그레이션한 후 애플케이션의 테스트 또는 스테이징 버전을 사용하거나 생산 워크로드를 재실행하여 데이터베이스의 성능 테스트를 수행합니다.

- 비 MySQL 호환 엔진의 경우 선택적으로 가장 분주한 테이블을 Amazon Aurora에 복사한 후 해당 테이블에 대한 쿼리를 테스트합니다. 이 전략은 좋은 시작점이 될 수 있습니다. 물론 완전한 데이터 마이그레이션 후의 테스트는 새 플랫폼에서 애플리케이션의 실제 성능에 대한 전체적인 그림을 제공해 줄 것입니다.

Amazon Aurora는 MySQL에 비해 상당히 개선된, 상용 엔진 수준의 성능을 제공합니다. 이러한 성능은 데이터베이스 워크로드용으로 설계된 SSD 기반 가상 스토리지 계층과 데이터베이스 엔진을 긴밀하게 통합하여 실현됩니다. 이는 스토리지 시스템으로의 쓰기 작업을 줄이고 잠금 경합을 최소화하며 데이터베이스 프로세스 스레드에 의해 발생하는 지연을 제거해 줍니다. r3.8xlarge 인스턴스에서의 SysBench 테스트 결과 Amazon Aurora는 동일한 하드웨어에서 동일한 벤치마크를 수행한 MySQL보다 5배 빠른 초당 585,000회의 읽기 및 초당 107,000회의 쓰기 속도를 제공하는 것으로 나타났습니다.

Amazon Aurora가 기존의 MySQL보다 월등히 개선된 부분 중 하나는 높은 동시성의 워크로드입니다. Amazon Aurora에서의 워크로드 처리량을 극대화하려면 다수의 동시 쿼리를 처리할 수 있도록 애플리케이션의 아키텍처를 설계하는 것이 좋습니다.

샤딩 및 읽기 복제본 고려 사항

현재 데이터베이스가 여러 노드에 걸쳐 샤딩된 경우 마이그레이션 중에 이러한 샤드를 단일 Aurora 데이터베이스로 결합할 수 있습니다. 단일 Amazon Aurora 인스턴스는 최대 64TB까지 확장될 수 있으며 수천 개의 테이블을 지원합니다. 또한 표준 MySQL 데이터베이스보다 훨씬 높은 수준의 읽기 및 쓰기 속도를 지원합니다.

애플리케이션의 읽기/쓰기 작업이 빈번한 경우 Aurora 읽기 전용 복제본을 사용하여 마스터 데이터베이스로부터 읽기 전용 워크로드만 오프로딩하는 방식을 고려할 수 있습니다. 이렇게 하면 쓰기 작업에 대한 기본 데이터베이스의 동시성을 개선할 수 있으며 전반적인 읽기 및 쓰기 성능이 향상됩니다. 또한 읽기 전용 복제본을 사용하면 다중 AZ 구성에서 기본 인스턴스에 대해 더 작은 인스턴스를 사용할 수 있으므로 비용을 절감할 수 있는 동시에 데이터베이스 클러스트에 장애 조치 기능을 추가해 줍니다. Aurora 읽기 전용 복제본은 거의 0에 가까운 복제 지연 시간을 제공하며 최대 15개의 읽기 전용 복제본을 생성할 수 있습니다.

안정성 고려 사항

데이터베이스에 대한 중요한 고려 사항 중 하나는 고가용성과 재해 복구입니다. 애플리케이션의 복구 시간 목표(RTO)와 복구 지점 목표(RPO)를 결정하십시오. Amazon Aurora를 사용하면 이 두 요소 모두를 크게 개선할 수 있습니다.

Amazon Aurora는 대부분의 데이터베이스 충돌 시나리오에서 데이터베이스 재시작 시간을 60초 미만으로 줄여줍니다. 또한 Aurora는 버퍼 캐시를 데이터베이스 프로세스 외부로 이동하여 재시작 시 즉시 사용할 수 있도록 합니다. 드물게 발생하는 하드웨어 및 가용 영역 장애 시나리오에서 복구는 데이터베이스 플랫폼에 의해 자동으로 처리됩니다.

Aurora는 AWS 리전 내에서 제로 RPO 복구를 제공하도록 설계되었으며 이는 온프레미스 데이터베이스 시스템에 비해 대폭 개선된 성능입니다. Aurora는 3개의 가용 영역에 6개의 데이터 사본을 유지 관리하고 데이터 손실 없이 정상적인 AZ에서 데이터베이스 복구를 자동으로 시도합니다. Amazon Aurora 스토리지 내에서 데이터를 사용할 수 없는 드문 상황에서는 DB 스냅샷에서 복구하거나 새 인스턴스에 특정 시점으로의 복원 작업을 수행할 수 있습니다.

비용 및 라이선스 고려 사항

데이터베이스의 소유 및 실행에는 관련 비용이 따릅니다. 데이터베이스 마이그레이션을 계획하기 전에 반드시 새 데이터베이스 플랫폼에 대한 총 소유 비용(TCO) 분석을 수행해야 합니다. 이상적으로 새 데이터베이스 플랫폼으로의 마이그레이션은 총 소유 비용을 낮추는 동시에 애플리케이션에 비슷하거나 더 나은 기능을 제공해야 합니다. 오픈 소스 데이터베이스 엔진(MySQL, Postgres)을 실행 중인 경우 대부분의 비용은 하드웨어, 서버 관리 및 데이터베이스 관리 작업과 관련된 비용입니다. 그러나 상용 데이터베이스 엔진(Oracle, SQL Server, DB2 등)을 실행 중인 경우 데이터베이스 라이선스가 비용의 상당한 부분을 차지합니다.

Aurora는 상용 엔진 대비 1/10의 비용으로 사용할 수 있으므로 Aurora로 이전하는 많은 애플리케이션의 TCO를 대폭 낮출 수 있습니다. MySQL 또는 Postgres와 같은 오픈 소스 엔진을 실행 중인 경우에도 Aurora의 뛰어난 성능 및 이중 용도의 읽기 전용 복제본을 사용하여, Amazon Aurora로 이동함으로써 의미 있는 비용 절감을 실현할 수 있습니다. 자세한 내용은 Amazon RDS for Aurora 요금 페이지를 참조하십시오.[2]

기타 마이그레이션 고려 사항

일단 애플리케이션 적합성, 성능, TCO 및 안정성 요소를 고려했으면 새 플랫폼으로 마이그레이션하는 데 무엇이 필요한지를 생각해야 합니다.

코드 변경 범위 예측

Amazon Aurora로 데이터베이스를 마이그레이션하는 과정에 수행해야 할 코드 및 스키마 변경의 양을 예측하는 것이 중요합니다. MySQL 호환 데이터베이스를 마이그레이션하는 경우 코드 변경이 거의 필요하지 않습니다. 그러나 비 MySQL 엔진을 마이그레이션할 때에는 스키마와 코드 변경 작업이 필요합니다. 이 문서의 후반부에 설명되어 있는 AWS Schema Conversion Tool을 사용하면 이러한 예측 작업에 도움이 될 수 있습니다.

마이그레이션 중의 애플리케이션 가용성

Amazon Aurora로 마이그레이션할 때에는 예측 가능한 애플리케이션 다운타임을 둔 접근 방식 또는 다운타임이 거의 없는 접근 방식 중에서 선택할 수 있습니다. 어느 접근 방식을 선택할지는 데이터베이스의 크기와 애플리케이션의 가용성 요구 사항에 따라 달라집니다. 어느 방식을 선택하든지 간에 데이터베이스 마이그레이션을 시작하기 전에 마이그레이션 프로세스가 애플리케이션과 비즈니스에 미치는 영향을 고려하는 것이 좋습니다. 다음 몇 개의 섹션에서는 두 방식 모두를 자세히 설명하고 있습니다.

데이터베이스 마이그레이션 프로세스 계획

이전 섹션에서는 Amazon Aurora로 데이터베이스를 마이그레이션하는 동안 생각해야 할 몇 가지 핵심 고려 사항을 다루었습니다. Aurora가 해당 애플리케이션에 적합한 솔루션이라는 것을 확인했으면 다음 단계는 사전 마이그레이션 접근 방식을 결정하고 데이터베이스 마이그레이션 계획을 작성하는 것입니다.

동종 마이그레이션

소스 데이터베이스가 MySQL 5.6 호환 데이터베이스(MySQL, MariaDB, Percona 등)인 경우 Aurora로의 마이그레이션은 매우 간단합니다.

다운타임을 둔 동종 마이그레이션

애플리케이션이 사용량이 적은 시간대에 예측 가능한 다운타임을 가질 수 있는 경우에는 다운타임을 둔 마이그레이션이 가장 간단한 옵션이자 가장 권장되는 접근 방식입니다. 대부분의 데이터베이스는 이미 잘 정의된 유지 관리 기간을 사용하고 있으므로 대부분의 마이그레이션 프로젝트가 이 범주에 포함됩니다. 다운타임을 둔 데이터베이스 마이그레이션에는 다음과 같은 옵션을 선택할 수 있습니다.

- **RDS 스냅샷 마이그레이션** – 소스 데이터베이스가 Amazon RDS MySQL 5.6에서 실행 중인 경우 간단히 해당 데이터베이스의 스냅샷을 Amazon Aurora로 마이그레이션할 수 있습니다. 다운타임을 둔 마이그레이션에서는 스냅샷과 마이그레이션이 진행되는 동안 애플리케이션을 중지하거나 데이터베이스 쓰기를 중지해야 합니다. 마이그레이션에 소요되는 시간은 주로 데이터베이스 크기에 따라 달라지며 생산 환경 마이그레이션 전에 테스트 마이그레이션을 실행하여 파악할 수 있습니다. 스냅샷 마이그레이션 옵션은 RDS 스냅샷 마이그레이션 섹션에 설명되어 있습니다. 다음 섹션에 설명되어 있는 스냅샷 마이그레이션 및 binlog 복제를 사용하여 다운타임이 거의 없는 마이그레이션을 수행할 수도 있습니다.
- **네이티브 MySQL 도구를 사용한 마이그레이션.** 네이티브 MySQL 도구를 사용하여 데이터 및 스키마를 Aurora로 마이그레이션할 수 있습니다. 이것은 데이터베이스 마이그레이션 프로세스를 좀 더 세부적으로 제어해야 하는 경우, 네이티브 MySQL 도구를 사용하는 것이 더 편한 경우 및 다른 마이그레이션 방법이 해당 사용 사례에서 원하는 대로 수행되지 않는 경우에 좋은 옵션입니다. 이 옵션 사용에 대한 모범 사례를 보려면 백서 Amazon RDS for Aurora 내보내기/가져오기 성능 모범 사례를 다운로드하십시오.[3]
- **AWS Database Migration Service(AWS DMS)를 사용한 마이그레이션.** 소스 데이터베이스를 Amazon Aurora로 마이그레이션하는 데 사용할 수 있는 또 다른 도구로는 AWS DMS를 사용한 일회성 마이그레이션이 있습니다. AWS DMS를 사용하여 데이터를 이동하려면 네이티브 MySQL 도구를 사용하여 소스에서 대상으로 데이터베이스 스키마를 복사해야 합니다. 단계별 프로세스에 대한 자세한 내용은 데이터 마이그레이션 섹션을 참조하십시오. AWS DMS를 사용하는 방법은 네이티브 MySQL 도구를 사용한 경험이 없는 경우에 좋은 옵션이 될 수 있습니다.

다운타임이 거의 없는 동종 마이그레이션

일부 시나리오에서는 다운타임을 최소화하면서 데이터베이스를 Aurora로 마이그레이션하기를 원할 수 있습니다. 다음의 두 경우가 그러한 시나리오의 예입니다.

- 데이터베이스가 비교적 크며 다운타임을 둔 옵션의 마이그레이션 시간이 애플리케이션 유지 관리 기간보다 긴 경우
- 테스트 용도로 소스 및 대상 데이터베이스를 병렬로 실행하고자 할 경우

이러한 경우 복제 기능을 사용하여 소스 MySQL 데이터베이스에서 Aurora로 변경 사항을 실시간으로 복제할 수 있습니다. 사용자는 다음의 두 가지 옵션 중 하나를 선택할 수 있습니다.

- **MySQL binlog 복제를 사용한 다운타임이 거의 없는 마이그레이션.** Amazon Aurora는 기존 MySQL binlog 복제를 지원합니다. MySQL 데이터베이스를 실행 중인 경우 기존 binlog 복제 설정에 대해 이미 잘 알고 있을 가능성이 높습니다. 이러한 경우에 해당하며 마이그레이션 프로세스를 더 세부적으로 제어하기를 원한다면 네이티브 도구를 사용한 일회성 데이터베이스 로드 및 binlog 복제의 조합이 Aurora로의 마이그레이션을 위한 익숙한 경로를 제공합니다.
- **AWS Database Migration Service(AWS DMS)를 사용한 다운타임이 거의 없는 마이그레이션.** 일회성 마이그레이션 지원 외에도 AWS DMS는 소스에서 대상으로의 변경 데이터 캡처(CDC)을 사용한 실시간 데이터 복제를 지원합니다. AWS DMS는 초기 데이터 복사, 복제 인스턴스 설정 및 복제 모니터링과 관련된 복잡성을 처리합니다. 초기 데이터베이스 마이그레이션이 완료되면 대상 데이터베이스는 선택한 기간 동안 소스 데이터베이스와 동기화된 상태를 유지합니다. binlog 복제에 익숙하지 않은 경우, 다운타임이 거의 없는 Amazon Aurora로의 동종 마이그레이션을 위한 차선의 옵션은 AWS DMS입니다. 자세한 내용은 AWS DMS 소개 및 일반 접근 방식을 참조하십시오.
- **RDS 스냅샷 마이그레이션 및 MySQL binlog 복제를 사용한 다운타임이 거의 없는 마이그레이션.** 소스 데이터베이스를 Amazon RDS MySQL 5.6에서 실행 중인 경우 해당 데이터베이스의 스냅샷을 Amazon Aurora로 마이그레이션한 후 소스와 대상 사이에 대한 binlog 복제를 시작할 수 있습니다. 이 마이그레이션 옵션에 대한 자세한 내용은*Amazon RDS 사용 설명서*의 Amazon Aurora를 사용한 복제를 참조하십시오.[4]

이기종 마이그레이션

비 MySQL 호환 데이터베이스(Oracle, SQL Server, PostgreSQL 등)를 Amazon Aurora로 마이그레이션하려는 경우 신속하고 간편한 마이그레이션이 되도록 도와줄 몇 가지 옵션이 있습니다.

스키마 마이그레이션

비 MySQL 호환 데이터베이스에서 Amazon Aurora로의 스키마 마이그레이션은 AWS Schema Conversion Tool을 사용하여 실행할 수 있습니다. 이 도구는 Oracle, Microsoft SQL Server 또는 PostgreSQL 데이터베이스의 데이터베이스 스키마를 Amazon RDS MySQL DB 인스턴스 또는 Amazon Aurora DB 클러스터로 변환할 수 있도록 해 주는 데스크톱 애플리케이션입니다. 소스 데이터베이스의 스키마를 자동으로 완전히 변환할 수 없는 경우 AWS Schema Conversion Tool을 사용하면 대상 Amazon RDS 데이터베이스에 동일한 스키마를 생성할 수 있는 방법에 대한 지침을 얻을 수 있습니다. 자세한 내용은 데이터베이스 스키마 마이그레이션 섹션을 참조하십시오.

데이터 마이그레이션

AWS Database Migration Service(AWS DMS)는 다운타임이 거의 없는 동종 마이그레이션을 지원할 뿐 아니라 이기종 데이터베이스 사이의 연속 복제도 지원하며 다운타임을 둔 마이그레이션 및 다운타임이 거의 없는 마이그레이션 모두에서 소스 데이터베이스를 대상 데이터베이스로 이동하는 데 선호되는 옵션입니다. 마이그레이션이 시작되면, AWS DMS는 데이터 유형 변환, 압축, 더 빠른 데이터 전송을 위한 병렬 전송과 같은 마이그레이션 프로세스 복잡성을 모두 관리하며 마이그레이션 프로세스 도중 발생하는 소스 데이터베이스의 데이터 변경 사항이 대상 데이터베이스에 자동으로 복제되도록 해 줍니다.

AWS DMS를 사용하는 것 외에도 Attunity Replicate, Tungsten Replicator, Oracle Golden Gate 등 다양한 타사 도구를 사용하여 데이터를 Amazon Aurora로 마이그레이션할 수 있습니다. 어느 도구를 선택하든 간에 마이그레이션에 필요한 도구 세트를 확정하기 전에 성능과 라이선스 비용을 고려해야 합니다.

Amazon Aurora로의 대규모 데이터베이스 마이그레이션

모든 데이터베이스 마이그레이션 프로젝트에서 대규모 데이터 세트의 마이그레이션 작업은 고유한 문제를 제기합니다. 대규모 데이터베이스 마이그레이션 프로젝트는 주로 다음과 같은 전략의 조합을 사용합니다.

- **연속 복제를 사용한 마이그레이션.** 일반적으로 대규모 데이터베이스는 소스에서 대상으로 데이터를 이동할 때 긴 다운타임이 요구됩니다. 다운타임을 줄이려면 먼저 소스의 기준선 데이터를 대상에 로드한 다음 복제(MySQL 네이티브 도구, AWS DMS 또는 타사 도구 사용)를 활성화하여 변경 사항을 포착해야 합니다.
- **정적 테이블 우선 복사.** 데이터베이스에서 참조 데이터가 포함된 큰 정적 테이블에 의존하는 경우 활성 데이터세트를 마이그레이션하기 전에 이러한 대형 테이블을 대상 데이터베이스에 마이그레이션할 수 있습니다. AWS DMS를 활용하여 테이블을 선택적으로 복사하거나 테이블을 내보낸 다음 수동으로 해당 테이블을 가져올 수 있습니다.
- **다단계 마이그레이션.** 수천 개의 테이블을 가진 대규모 데이터베이스의 마이그레이션을 여러 단계를 분할할 수 있습니다. 예를 들어, 데이터베이스가 대상 데이터베이스로 완전히 마이그레이션될 때까지 교차 결합 쿼리가 없는 테이블 세트를 주말마다 이동할 수 있습니다. 참고로 이를 달성하려면 데이터세트가 두 개의 개별 노드에 있는 동안 애플리케이션이 두 개의 데이터베이스에 동시에 연결하도록 설정을 변경해야 합니다. 이는 일반적인 마이그레이션 패턴은 아니지만 옵션 중 하나로 선택할 수 있습니다.
- **데이터베이스 정리.** 많은 대규모 데이터베이스에는 사용되지 않는 데이터 및 테이블이 포함되어 있습니다. 많은 경우 개발자와 DBA는 동일한 데이터베이스에 백업 테이블 사본을 유지하거나 사용되지 않는 테이블을 삭제하는 것을 잊습니다. 이유에 상관없이 데이터베이스 마이그레이션 프로젝트는 마이그레이션 전에 기존 데이터베이스를 정리할 수 있는 기회를 제공합니다. 일부 테이블이 사용되지 않는 경우 삭제하거나 다른 데이터베이스에 보관할 수 있습니다. 대형 테이블에서 오래된 데이터를 삭제하거나 해당 데이터를 플랫 파일로 보관할 수도 있습니다.

Amazon Aurora에서의 파티션 및 샤드 통합

우수한 성능을 얻기 위해 데이터베이스를 여러 샤드 또는 기능 파티션으로 실행하고 있는 경우 이러한 파티션 또는 샤드를 단일 Aurora 데이터베이스로 통합할 수 있는 기회가 제공됩니다. 단일 Amazon Aurora 인스턴스는 최대 64TB까지 확장될 수 있으며 수천 개의 테이블을 지원합니다. 또한 표준 MySQL 데이터베이스보다 훨씬 높은 수준의 읽기 및 쓰기 속도를 지원합니다. 이러한 파티션을 단일 Aurora 인스턴스로 통합하면 총 소유 비용을 줄이고 데이터베이스 관리를 간소화할 뿐 아니라 교차 파티션 쿼리 성능을 크게 향상시킵니다.

- **기능 파티션.** 기능 파티션은 작업에 따라 서로 다른 전용 노드를 할당하는 것입니다. 예를 들어, 전자 상거래 애플리케이션에서 제품 카탈로그 데이터를 담당하는 데이터베이스 노드와 주문 포착 및 처리를 담당하는 또 다른 데이터베이스 노드를 가질 수 있습니다. 그 결과 이러한 파티션은 일반적으로 겹치지 않는 개별적인 스키마를 가지게 됩니다.

  통합 전략. 각 기능 파티션을 개별 스키마 형태로 대상 Aurora 인스턴스에 마이그레이션합니다. 소스 데이터베이스가 MySQL 호환인 경우 네이티브 MySQL 도구를 사용하여 스키마를 마이그레이션한 다음 AWS DMS에서 일회성 또는 연속 방식으로 복제를 사용하여 데이터를 마이그레이션합니다. 소스 데이터베이스가 비 MySQL 호환인 경우 AWS Schema Conversion Tool을 사용하여 스키마를 Aurora로 마이그레이션하고 AWS DMS를 사용하여 일회성 로드 또는 연속 복제를 수행합니다.

- **데이터 샤드.** 복수의 노드에 걸친 개별적인 데이터 세트에 동일한 스키마를 사용하는 경우 데이터베이스 샤드를 활용하고 있는 것입니다. 예를 들어, 트래픽이 많은 블로그 서비스의 경우 동일한 테이블 스키마를 유지하면서 사용자 활동과 데이터를 여러 데이터베이스 샤드로 분산할 수 있습니다.

  통합 전략. 모든 샤드가 동일한 데이터베이스 스키마를 공유하므로 대상 스키마는 한 번만 생성하면 됩니다. MySQL 호환 데이터베이스를 사용하는 경우 네이티브 도구를 사용하여 데이터베이스 스키마를 Aurora로 마이그레이션합니다. 비 MySQL 데이터베이스를 사용하는 경우 AWS Schema Conversion Tool을 사용하여 데이터베이스 스키마를 Aurora로 마이그레이션합니다. 일단 데이터베이스 스키마를 마이그레이션했으면 데이터베이스 샤드에 대한 쓰기를 중지하고 네이티브 도구 또는 일회성 AWS DMS 데이터 로드를 사용하여 개별 샤드를 Aurora로 마이그레이션하는 것이 좋습니다. 애플리케이션에 대한 쓰기 작업을 장기간 중지할 수 없는 경우 AWS DMS와 복제 기능을 사용할 수 있지만 이는 올바른 계획 작성과 테스팅을 수행한 후에 실시해야 합니다.

마이그레이션 옵션 요약

소스 데이터베이스 유형	다운타임을 둔 마이그레이션	다운타임이 거의 없는 마이그레이션
Amazon RDS MySQL	옵션 1: RDS 스냅샷 마이그레이션 옵션 2: 네이티브 도구를 사용한 수동 마이그레이션* 옵션 3: 네이티브 도구를 사용한 스키마 마이그레이션 및 AWS DMS를 사용한 데이터 로드	옵션 1: 네이티브 도구 + binlog 복제를 사용한 마이그레이션 옵션 2: RDS 스냅샷 마이그레이션 + binlog 복제 옵션 3: 네이티브 도구를 사용한 스키마 마이그레이션 + ASW DMS를 사용한 데이터 이동

소스 데이터베이스 유형	다운타임을 둔 마이그레이션	다운타임이 거의 없는 마이그레이션
MySQL Amazon EC2 또는 온프레미스	옵션 1: 네이티브 도구를 사용한 마이그레이션 옵션 2: 네이티브 도구를 사용한 스키마 마이그레이션 + AWS DMS를 사용한 데이터 로드	옵션 1: 네이티브 도구 + binlog 복제를 사용한 마이그레이션 옵션 2: 네이티브 도구를 사용한 스키마 마이그레이션 + ASW DMS를 사용한 데이터 이동
Oracle/SQL 서버	옵션 1: AWS Schema Conversion Tool + AWS DMS(권장) 옵션 2: 수동 또는 타사 도구를 사용한 스키마 변환 + 대상으로의 수동 또는 타사 데이터 로드	옵션 1: AWS Schema Conversion Tool + AWS DMS(권장) 옵션 2: 수동 또는 타사 도구를 사용한 스키마 변환 + 대상으로의 수동 또는 타사 데이터 로드 + 타사 도구를 사용한 복제
기타 비 MySQL 데이터베이스	옵션: 수동 또는 타사 도구를 사용한 스키마 변환 + 대상으로의 수동 또는 타사 데이터 로드	옵션: 수동 또는 타사 도구를 사용한 스키마 변환 + 대상으로의 수동 또는 타사 데이터 로드 + 타사 도구를 사용한 복제(GoldenGate 등)

* Mysql 네이티브 도구: mysqldump, SELECT INTO OUTFILE, mydumper/myloader와 같은 타사 도구

RDS 스냅샷 마이그레이션

스냅샷 마이그레이션을 사용하여 Aurora로 이동하려면 해당 MySQL 데이터베이스가 Amazon RDS MySQL 5.6에서 실행 중이어야 하며 데이터베이스의 RDS 스냅샷을 만들어야 합니다. 이 마이그레이션 방식은 온프레미스 데이터베이스 또는 Amazon Elastic Compute Cloud(Amazon EC2)에서 실행되는 데이터베이스에는 사용할 수 없습니다. 또한 5.6 이전 버전의 Amazon RDS MySQL 데이터베이스를 실행 중인 경우 사전 요구 사항으로 해당 버전을 5.6으로 업그레이드해야 합니다.

이 마이그레이션에 가장 큰 이점은 가장 적은 수의 단계를 필요로 하는 가장 단순한 방법이라는 점입니다. 특히 이 방식은 모든 스키마 객체, 보조 인덱스 및 저장된 프로시저를 모든 데이터베이스 데이터와 함께 마이그레이션합니다.

binlog 복제 없이 스냅샷 마이그레이션을 수행하는 동안에는 마이그레이션 동안 소스 데이터베이스가 변경되지 않도록 소스 데이터베이스가 오프라인 상태이거나 읽기 전용 모드여야 합니다. 다운타임을 추정하려면 데이터베이스의 기존 스냅샷을 사용하여 테스트 마이그레이션을 수행할 수 있습니다. 마이그레이션 시간이 다운타임 요구 사항 범위 내인 경우 이 방식이 가장 좋은 옵션이 될 수

있습니다. 일부 경우에는 AWS DMS 또는 네이티브 마이그레이션 도구를 사용한 마이그레이션이 스냅샷 마이그레이션을 사용하는 것보다 빠를 수 있습니다.

긴 다운타임이 허용되지 않는 경우에도 다운타임이 거의 없는 마이그레이션 효과를 실현할 수 있습니다. 이렇게 하려면 소스 데이터베이스가 업데이트되도록 허용하는 동안 RDS 데이터베이스의 스냅샷을 먼저 Amazon Aurora로 마이그레이션합니다. 그런 다음 MySQL binlog 복제를 사용하여 MySQL의 최신 상태를 Aurora로 전송합니다.

수동 또는 자동 DB 스냅샷을 마이그레이션할 수 있습니다. 일반적으로 다음 단계를 따라야 합니다.

1. Amazon RDS MySQL 5.6 인스턴스를 Aurora DB 클러스터로 마이그레이션하는 데 필요한 공간 크기를 결정합니다. 자세한 정보는 다음 섹션을 참조하십시오.
2. Amazon RDS 콘솔을 사용하여 Amazon RDS MySQL 5.6 인스턴스가 위치한 리전의 스냅샷을 생성합니다.
3. 콘솔에서 [**Migrate Database**] 기능을 사용하여 MySQL 5.6의 원본 DB 인스턴스의 DB 스냅샷으로 채워질 Amazon Aurora DB 클러스터를 생성합니다.

참고: 일부 MyISAM 테이블은 오류 없이 변환할 수 없으며 수동 변경이 필요할 수 있습니다. 예를 들어 InnoDB 엔진은 복합 키의 일부로 자동 증분 필드를 허용하지 않습니다. 또한 공간 인덱스는 현재 지원되지 않습니다.

스냅샷 마이그레이션을 위한 공간 요구 사항 예측

MySQL DB 인스턴스의 스냅샷을 Aurora DB 클러스터로 마이그레이션할 때 Aurora는 마이그레이션 전에 Amazon Elastic Block Store(EBS) 볼륨을 사용하여 스냅샷의 데이터를 포맷합니다. 일부 경우에는 마이그레이션할 데이터를 포맷하기 위해 추가 공간이 필요할 수도 있습니다. 마이그레이션 중에 공간 문제를 유발할 수 있는 두 가지 기능으로는 MyISAM 테이블과 `ROW_FORMAT=COMPRESSED` 옵션이 있습니다. 소스 데이터베이스에 이러한 기능을 사용하지 않는 경우 공간 문제가 발생하지 않으므로 이 섹션을 건너뛸 수 있습니다. 마이그레이션 중에 MyISAM 테이블은 InnoDB로 전환되고 모든 압축 테이블은 압축이 해제됩니다. 따라서 이러한 테이블의 추가 사본을 위한 충분한 공간이 필요합니다.

마이그레이션 볼륨의 크기는 스냅샷을 생성한 소스 MySQL 데이터베이스에 할당된 크기에 따라 다릅니다. 따라서 MyISAM 또는 압축 테이블이 전체 데이터베이스 크기에서 작은 비율을 차지하고 있으며 원본 데이터베이스에 충분한

공간이 있는 경우에는 특별한 공간 문제가 발생하지 않고 마이그레이션이 성공합니다. 그러나 원본 데이터베이스에 변환된 MyISAM 테이블 및 압축 테이블의 기타(비압축) 사본을 저장할 공간이 충분하지 않은 경우에 마이그레이션 볼륨의 크기가 충분하지 않을 수 있습니다. 이 경우 소스 Amazon RDS MySQL 데이터베이스를 수정하여 해당 테이블의 추가 사본을 위한 공간을 확보할 수 있도록 소스 데이터베이스 크기 할당을 늘리고 데이터베이스의 새 스냅샷을 생성한 후 새 스냅샷을 마이그레이션해야 합니다.

데이터를 DB 클러스터로 마이그레이션할 경우, 다음과 같은 지침과 제한 사항을 준수해야 합니다.

- Amazon Aurora는 최대 64TB의 스토리지를 지원하지만 스냅샷을 Aurora DB 클러스터로 마이그레이션하는 프로세스는 스냅샷의 Amazon EBS 볼륨 크기에 의해 제한되므로 최대 크기는 6TB로 제한됩니다.
- 원본 데이터베이스의 비 MyISAM 테이블의 최대 크기는 6TB입니다. 그러나 변환 도중 필요한 추가 공간 요구 사항이 있으므로 해당 MySQl DB 인스턴스에서 마이그레이션되는 MyISAM 및 압축 테이블의 크기는 3TB를 초과할 수 없습니다.

Amazon Aurora로 마이그레이션하기 전에 데이터베이스 스키마를 수정할 수 있습니다(MyISAM 테이블을 InnoDB로 변환한 다음 `ROW_FORMAT=COMPRESSED` 제거). 이 작업은 다음과 같은 경우에 도움이 됩니다.

- 마이그레이션 프로세스를 가속화하고 싶은 경우
- 프로비저닝에 얼마만큼의 공간이 필요한지를 정확히 모르는 경우
- 데이터 마이그레이션을 시도했지만 프로비저닝 공간 부족으로 마이그레이션이 실패한 경우

이러한 변경 작업은 생산 Amazon RDS MySQL 데이터베이스 인스턴스가 아닌 생산 스냅샷으로부터 복원된 데이터베이스 인스턴스에 수행해야 합니다. 이 작업을 수행하는 데 대한 자세한 내용은 *Amazon RDS 사용 설명서*의 Amazon Aurora로의 데이터 마이그레이션에 필요한 공간 감소를 참조하십시오.[5]

콘솔을 사용한 DB 스냅샷 마이그레이션

Amazon RDS MySQL DB 인스턴스의 DB 스냅샷을 마이그레이션하여 Aurora DB 클러스터를 생성할 수 있습니다. 새로운 DB 클러스터는 원본 Amazon RDS MySQL DB 인스턴스의 데이터로 채워집니다. 이때 DB 스냅샷은 MySQL 5.6을 실행하는 RDS DB 인스턴스에서 생성된 스냅샷이어야 하며 암호화되지 않아야

합니다. DB 스냅샷 생성에 대한 자세한 내용은 *Amazon RDS 사용 설명서*의 DB 스냅샷 생성을 참조하십시오.[6]

해당 DB 스냅샷이 Aurora DB 클러스터를 구성하려는 리전에 속하지 않을 경우, Amazon RDS 콘솔을 사용하여 DB 스냅샷을 해당 리전으로 복사합니다. DB 스냅샷 복사에 대한 자세한 내용은 *Amazon RDS 사용 설명서*의 DB 스냅샷 복사를 참조하십시오.[7]

콘솔을 사용하여 MySQL 5.6 DB 스냅샷을 마이그레이션하려면 다음을 수행합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/rds/의 Amazon RDS 콘솔을 엽니다.
2. **[Snapshots]**를 선택합니다.
3. **[Snapshots]** 페이지에서 Aurora DB 클러스터로 마이그레이션하려는 Amazon RDS MySQL 5.6 스냅샷을 선택합니다.
4. **[Migrate Database]**를 선택합니다.
5. **[Migrate Database]** 페이지에서 다음 그림에 표시된 것과 같이 환경 및 프로세싱 요구 사항과 일치하는 값을 지정합니다. 이러한 옵션에 대한 자세한 내용은 *Amazon RDS 사용 설명서*의 콘솔을 사용한 DB 스냅샷 마이그레이션을 참조하십시오.[8]

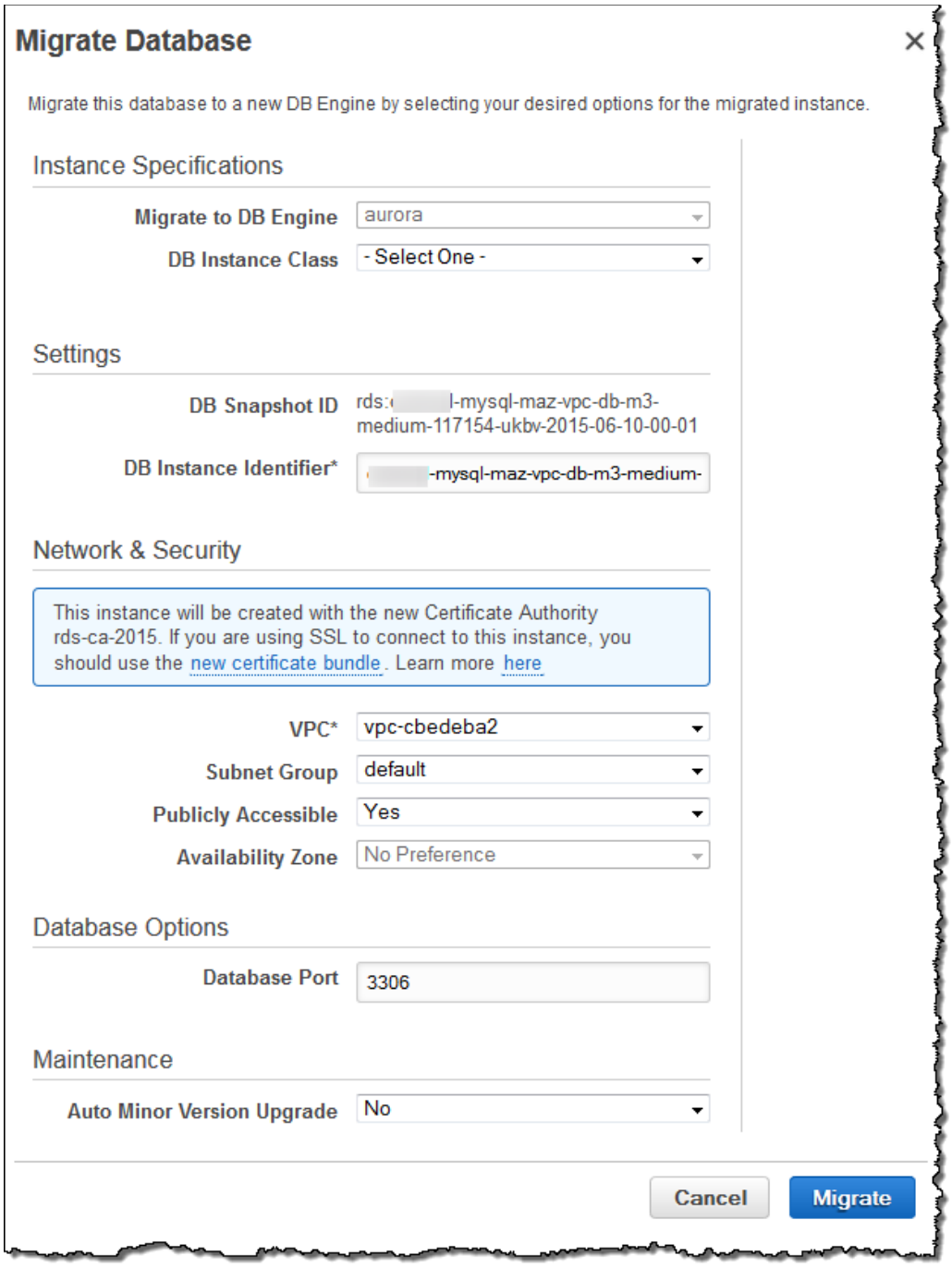

그림 2: 스냅샷 마이그레이션

6. **[Migrate]**를 클릭하여 DB 스냅샷을 마이그레이션합니다.

인스턴스 목록에서 적절한 화살표 아이콘을 클릭하여 DB 클러스터 세부 정보를 표시하고 마이그레이션 진행 상태를 모니터링합니다. 이 세부 정보 패널에는 DB 클러스터의 기본 인스턴스에 연결하는 데 사용된 클러스터 엔드포인트가

표시됩니다. Amazon Aurora DB 클러스터 연결에 대한 자세한 내용은 *Amazon RDS 사용 설명서*의 Amazon Aurora DB 클러스터 연결을 참조하십시오.[9]

데이터베이스 스키마 마이그레이션

RDS DB 스냅샷 마이그레이션은 전체 스키마 및 데이터 모두를 새 Aurora 인스턴스로 마이그레이션합니다. 그러나 소스 데이터베이스 위치 또는 애플리케이션 업타임 요구 사항이 RDS 스냅샷 마이그레이션의 사용을 허용하지 않을 경우, 실제 데이터를 이동할 수 있으려면 먼저 소스 데이터베이스에서 대상 데이터베이스로 데이터베이스 스키마를 마이그레이션해야 합니다. 데이터베이스 스키마는 전체 데이터베이스의 논리적 보기를 나타내는 골격 구조로서 일반적으로 다음과 같은 요소를 포함합니다.

- 데이터베이스 스토리지 객체: 테이블, 열, 제약 조건, 인덱스, 시퀀스, 사용자 정의 유형 및 데이터 유형
- 데이터베이스 코드 객체: 함수, 프로시저, 패키지, 트리거, 뷰, 구체화된 보기, 이벤트, SQL 스칼라 함수, SQL 인라인 함수, SQL 테이블 기능, 속성, 변수, 상수, 테이블 유형, 퍼블릭 유형, 프리이빗 유형, 커서, 예외, 매개 변수 및 기타 객체

대부분의 경우, 데이터베이스 스키마는 비교적 정적으로 유지되므로 데이터베이스 스키마 마이그레이션 도중 다운타임이 필요하지 않습니다. 소스 데이터베이스의 스키마는 소스 데이터베이스가 실행되는 동안에도 성능에 영향을 주지 않고 추출될 수 있습니다. 애플리케이션 또는 개발자가 데이터베이스 스키마를 자주 변경하는 경우, 마이그레이션이 진행되는 동안 이러한 변경을 일시 중지하게 하거나 스키마 마이그레이션 프로세스에 이러한 요소를 반영해야 합니다.

소스 데이터베이스의 유형에 따라 다음 섹션에 설명된 기술을 사용하여 데이터베이스 스키마를 마이그레이션할 수 있습니다. 스키마 마이그레이션의 사전 요구 사항 중 하나로 대상 Aurora 데이터베이스가 생성되어 사용 가능한 상태여야 합니다.

동종 스키마 마이그레이션

소스 데이터베이스가 MySQL 5.6 호환이며 Amazon RDS, Amazon EC2 또는 AWS 외부에서 실행 중인 경우 네이티브 MySQL 도구를 사용하여 스키마를 내보내고 가져올 수 있습니다.

- **데이터베이스 스키마 내보내기.** mysqldump 클라이언트 유틸리티를 사용하여 데이터베이스 스키마를 내보낼 수 있습니다. 이 유틸리티를

실행하려면 소스 데이터베이스에 연결하여 mysqldump 명령의 출력을 플랫 파일로 리디렉션해야 합니다. -no-data 옵션을 사용하면 실제 테이블 데이터 없이 데이터베이스 스키마만 내보낼 수 있습니다. 전체 mysqldump명령 참조는 mysqldump—데이터베이스 백업 프로그램을 참조하십시오.[10]

```
mysqldump -u source_db_username -p --no-data --routines --
triggers -databases source_db_name > DBSchema.sql
```

- **Aurora로 데이터베이스 스키마 가져오기.** 스키마를 Aurora 인스턴스로 가져오려면 MySQL 명령줄 클라이언트(또는 해당 Windows 클라이언트)에서 Aurora 데이터베이스에 연결하고 내보내기 파일의 내용을 MySQL로 전달합니다.

```
mysql -h aurora-cluster-endpoint -u username -p <
DBSchema.sql
```

다음을 참고하십시오.

- 소스 데이터베이스가 저장된 프로시저, 트리거 및 보기를 포함하는 경우 덤프 파일에서 DEFINER 구문을 제거해야 합니다. 이 작업을 위한 간단한 Perl 명령이 아래에 나와 있습니다. 이 명령을 수행하면 현재 연결된 사용자의 모든 트리거, 뷰 및 저장된 프로시저를 DEFINER로 생성합니다. 이로 인한 모든 보안 영향을 평가해야 합니다.

```
$perl -pe 's/\sDEFINER=`[^`]+`@`[^`]+`//' < DBSchema.sql >
DBSchemaWithoutDEFINER.sql
```

- Amazon Aurora는 InnoDB 테이블만 지원합니다. 소스 데이터베이스에 MyISAM 테이블이 있는 경우, CREATE TABLE 명령을 실행할 때 Aurora는 자동으로 엔진을 InnoDB로 변경합니다.
- Amazon Aurora는 압축 테이블(즉, ROW_FORMAT=COMPRESSED로 생성된 테이블)을 지원하지 않습니다. 소스 데이터베이스에 압축 테이블이 있는 경우 CREATE TABLE 명령을 실행할 때 Aurora는 자동으로 ROW_FORMAT을 COMPACT로 변경합니다.

MySQL 5.6 호환 소스 데이터베이스의 스키마를 Amazon Aurora로 가져왔으면 다음 단계는 소스에서 대상으로 실제 데이터를 복사하는 것입니다. 자세한 내용은 이 문서 뒷부분의 AWS DMS 소개 및 일반 접근 방식을 참조하십시오.

이기종 스키마 마이그레이션

소스 데이터베이스의 MySQL 호환이 아닌 경우 스키마를 Amazon Aurora와 호환되는 형식으로 변환해야 합니다. 한 데이터베이스 엔진에서 다른 데이터베이스 엔진으로 스키마를 변환하는 것은 간단한 작업이 아니며 데이터베이스 및 애플리케이션 코드를 일부 재작성해야 할 수도 있습니다. 스키마를 변환하고 Amazon Aurora로 마이그레이션하는 옵션은 크게 두 가지가 있습니다.

- **AWS Schema Conversion Tool.** AWS Schema Conversion Tool은 소스 데이터베이스 스키마 및 보기, 저장된 프로시저 및 함수를 포함한 사용자 정의 코드 대부분을 대상 데이터베이스와 호환되는 형식으로 자동 변환하여 이기종 데이터베이스 마이그레이션을 쉽게 만들어 줍니다.[11] 자동으로 변환할 수 없는 코드는 명확하게 표시되므로 수동으로 변환할 수 있습니다. 이 도구를 사용하여 Oracle 또는 Microsoft SQL Server에서 실행되는 소스 데이터베이스를 Amazon RDS 또는 Amazon EC2의 Amazon Aurora, MySQL 또는 PostgreSQL 대상 데이터베이스로 변환할 수 있습니다. AWS Schema Conversion Tool을 사용하여 Oracle, SQL Server 또는 PostgreSQL 스키마를 Aurora 호환 형식으로 변환하는 것이 권장되는 방식입니다.
- **수동 스키마 마이그레이션 및 타사 도구.** 소스 데이터베이스가 Oracle, SQL Server 또는 PostgreSQL이 아닌 경우 소스 데이터베이스 스키마를 Aurora로 수동 마이그레이션하거나 타사 도구를 사용하여 스키마를 MySQL 5.6 호환 형식으로 마이그레이션할 수 있습니다. 수동 스키마 마이그레이션은 소스 스키마의 크기 및 복잡성에 따라 상당한 노력이 필요할 수 있는 작업입니다. 그러나 수동 스키마 변환은 대부분의 경우 비용 절감, 성능 이점 및 Amazon Aurora가 제공하는 기타 개선 사항을 선사하므로 충분히 가치가 있는 작업입니다.

AWS Schema Conversion Tool을 사용한 스키마 마이그레이션

AWS Schema Conversion Tool은 소스 데이터베이스의 데이터베이스 스키마를 Amazon Aurora와 호환되는 형식으로 자동 변환할 수 있는 프로젝트 기반 사용자 인터페이스를 제공합니다. 실제 생산 마이그레이션 전에 AWS Schema Conversion Tool을 사용하여 데이터베이스 마이그레이션에 필요한 작업량을 평가하고 파일럿 마이그레이션을 수행할 것이 적극 권장됩니다.

다음 설명은 AWS Schema Conversion Tool을 사용하는 방법에 대한 개략적인 안내를 제공합니다. 자세한 지침은 *AWS Schema Conversion Tool 사용 설명서*의 시작하기 섹션을 참조하십시오.[12]

1. 먼저, 도구를 설치합니다. AWS Schema Conversion Tool은 Microsoft Windows, Mac OS X, Ubuntu Linux 및 Fedora Linux에서 사용할 수 있습니다.

   상세한 다운로드 및 설치 지침은 사용 설명서의 설치 및 업데이트 섹션에서 찾을 수 있습니다.[13] AWS Schema Conversion Tool의 설치 위치는 중요합니다. 이 도구는 스키마 변환 및 적용을 위해 소스 및 대상 데이터베이스 모두에 직접 연결해야 합니다. AWS Schema Conversion Tool이 설치되는 데스크톱에 소스 및 대상 데이터베이스로의 네트워크 연결이 설정되어 있는지 확인하십시오.

2. JDBC 드라이버를 설치합니다. AWS Schema Conversion Tool은 소스 및 대상 데이터베이스에 연결하기 위해 JDBC 드라이버를 사용합니다. 이 도구를 사용하려면 이러한 JDBC 드라이버를 로컬 데스크톱에 다운로드해야 합니다. 드라이버 다운로드를 위한 지침은 *AWS Schema Conversion Tool 사용 설명서*의 필수 데이터베이스 드라이버에서 찾을 수 있습니다.[14] 또한 AWS Schema Conversion Tool을 위한 AWS 포럼에서 다양한 데이터베이스 엔진용 JDBC 드라이버 설정에 대한 지침을 확인하십시오.[15]

3. 대상 데이터베이스를 생성합니다. Amazon Aurora 대상 데이터베이스를 생성합니다. Amazon Aurora 데이터베이스 생성에 대한 지침은 *Amazon RDS 사용 설명서*의 Amazon Aurora DB 클러스터 생성을 참조하십시오.[16]

4. AWS Schema Conversion Tool을 열고 [**New Project Wizard**]를 시작합니다.

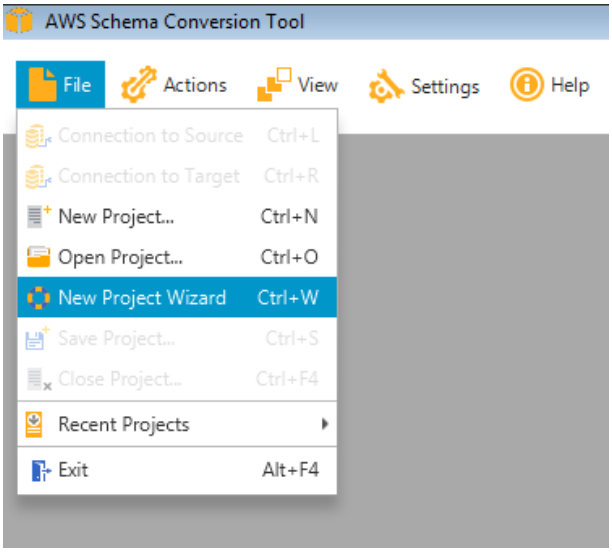

그림 3: AWS Schema Conversion Tool 프로젝트 생성

5. 소스 데이터베이스와 AWS Schema Conversion Tool 및 소스 데이터베이스의 테스트 연결을 구성합니다. 이 작업을 수행하려면 데스크톱에서 소스 데이터베이스에 연결할 수 있어야 하므로 적절한 네트워크 및 방화벽 설정이 구성되어 있는지 확인합니다.

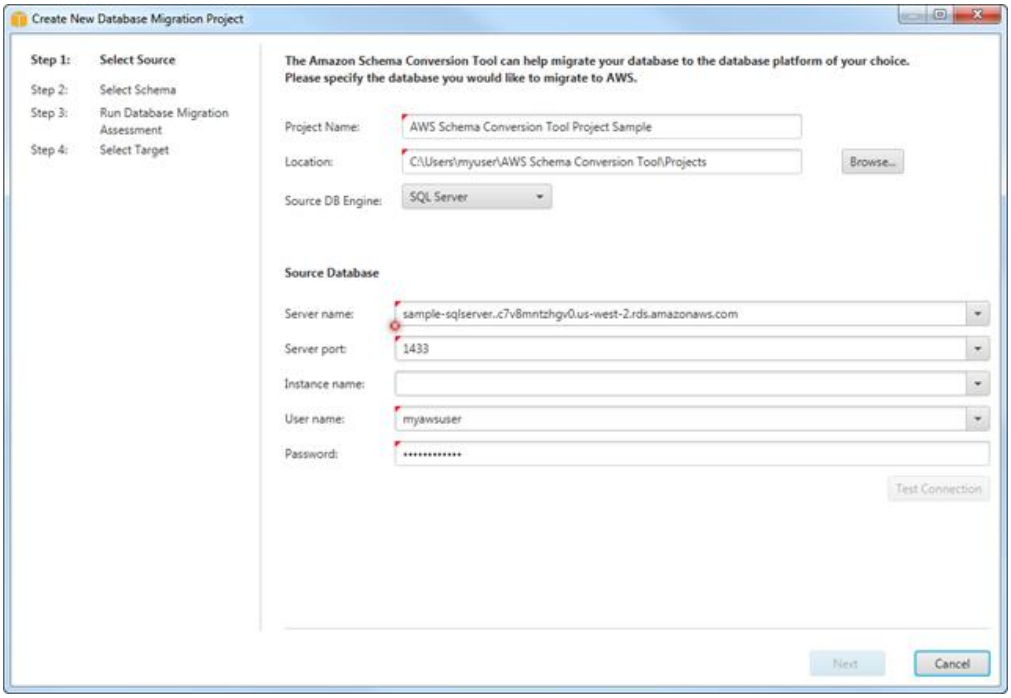

그림 4: Create New Database Migration Project 마법사

6. 다음 화면에서 Amazon Aurora로 변환할 소스 데이터베이스 스키마를 선택합니다.

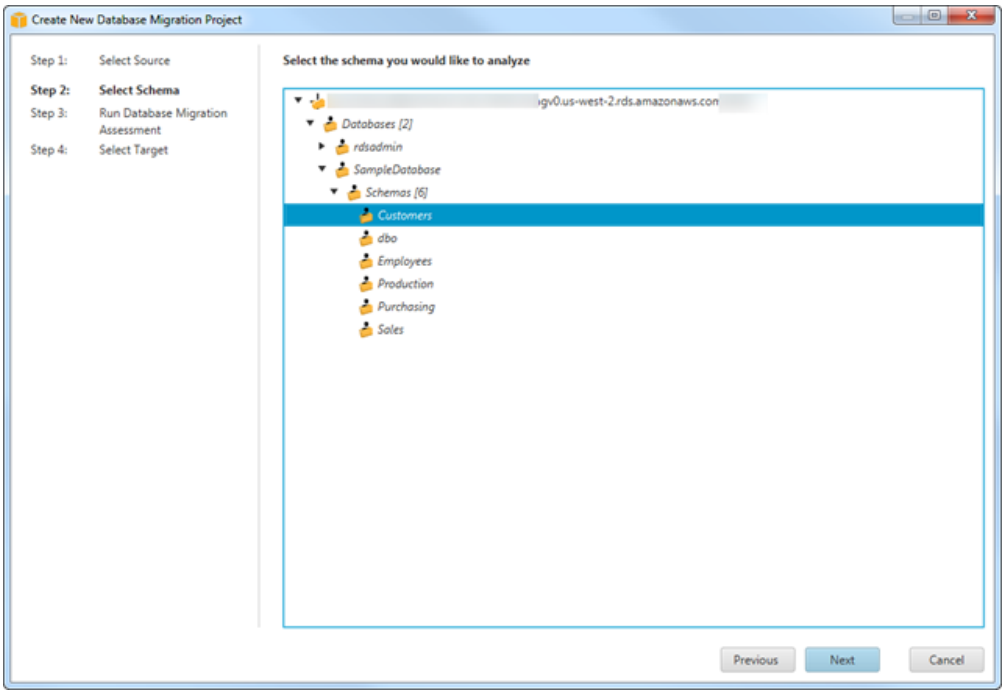

그림 5: 마이그레이션 마법사의 스키마 선택 단계

7. 데이터베이스 마이그레이션 평가 보고서를 실행합니다. 이 보고서는 소스 데이터베이스에서 대상 Amazon Aurora 인스턴스로의 스키마 변환에 대한 중요한 정보를 제공합니다. 보고서에는 모든 스키마 변환 작업이 요약되어 있으며 Aurora로 자동 변환할 수 없는 모든 스키마 요소에 대한 작업 항목을 세부적으로 설명하고 있습니다. 이 보고서는 또한 자동 변환되지 않는 항목에 대해 대상 데이터베이스에 동일한 코드를 작성하는 데 소요되는 예상 작업량 수치를 포함합니다.

   [Next]를 클릭하여 대상 데이터베이스를 구성합니다. 이 마이그레이션 보고서는 나중에 다시 볼 수 있습니다.

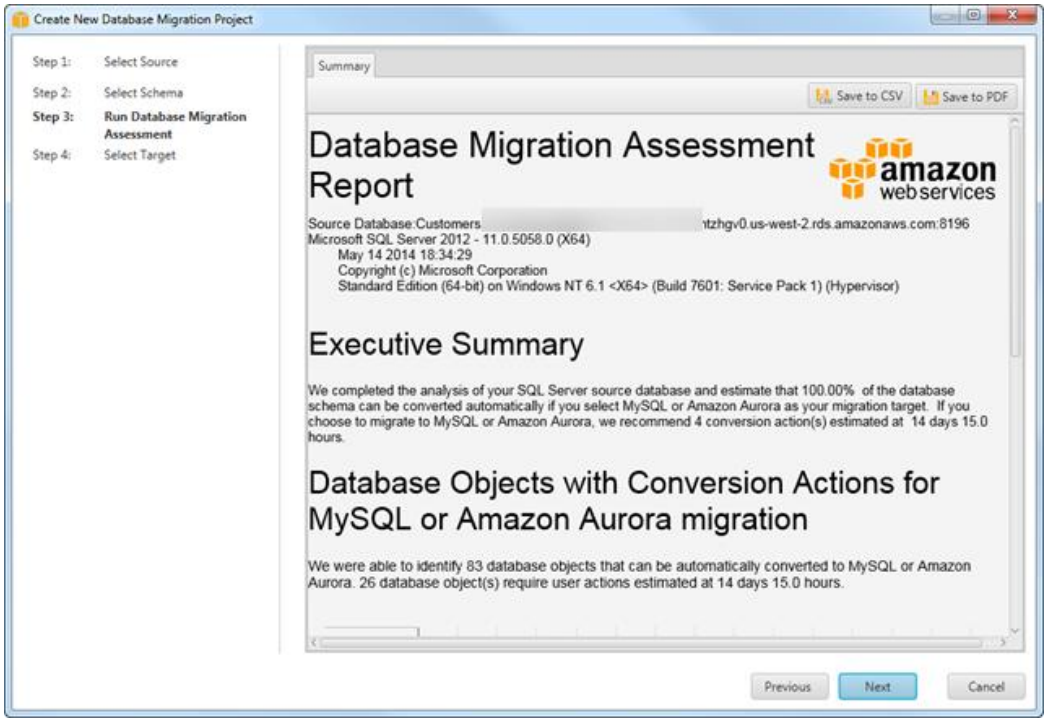

그림 6: 마이그레이션 보고서

8. 대상 Amazon Aurora 데이터베이스를 구성하고 AWS Schema Conversion Tool 및 소스 데이터베이스 사이의 연결을 테스트합니다. 이 작업을 수행하려면 데스크톱에서 대상 데이터베이스에 연결할 수 있어야 하므로 적절한 네트워크 및 방화벽 설정이 구성되어 있는지 확인합니다. **[Finish]**를 클릭하여 프로젝트 창으로 이동합니다.
9. 프로젝트 창으로 이동했다는 것은 소스 및 대상 데이터베이스 연결이 이미 설정되었으며 이제 세부 평가 보고서를 분석하고 스키마를 마이그레이션할 준비가 되었음을 의미합니다.
10. 소스 데이터베이스의 스키마를 표시하는 왼쪽 패널에서 평가 보고서를 생성할 스키마 객체를 선택합니다. 객체를 마우스 오른쪽 버튼으로 클릭하고 **[Create Report]**를 선택합니다.

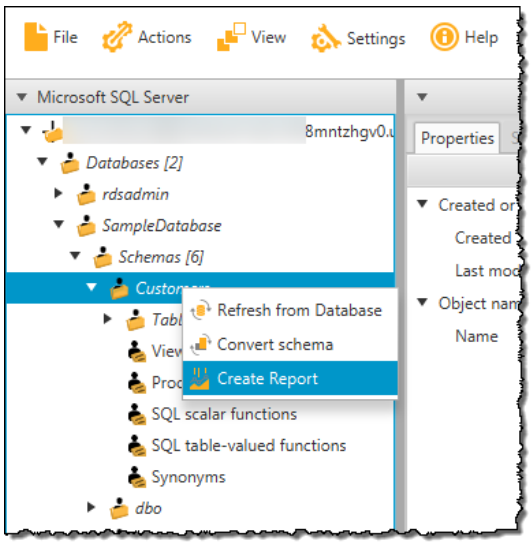

그림 7: 마이그레이션 보고서 생성

[Summary] 탭은 데이터베이스 마이그레이션 평가 보고서의 요약 정보를 표시합니다. 또한 자동 변환된 항목과 자동 변환할 수 없는 항목을 표시합니다.

대상 데이터베이스 엔진으로 자동 변환할 수 없는 스키마 항목의 경우 요약 정보에는 소스 데이터베이스와 동일한 스키마를 대상 DB 인스턴스에 생성하는 데 필요한 예상 작업량에 대한 정보가 포함됩니다. 보고서는 이러한 스키마 항목의 변환에 소요되는 예상 시간을 다음과 같은 범주로 표시합니다.

- **Simple** – 1시간 내에 완료할 수 있는 작업.
- **Medium** – 좀 더 복잡하며 1 ~ 4시간 내에 완료될 수 있는 작업.
- **Significant** – 매우 복잡하며 완료하는 데 4시간 이상이 소요되는 작업.

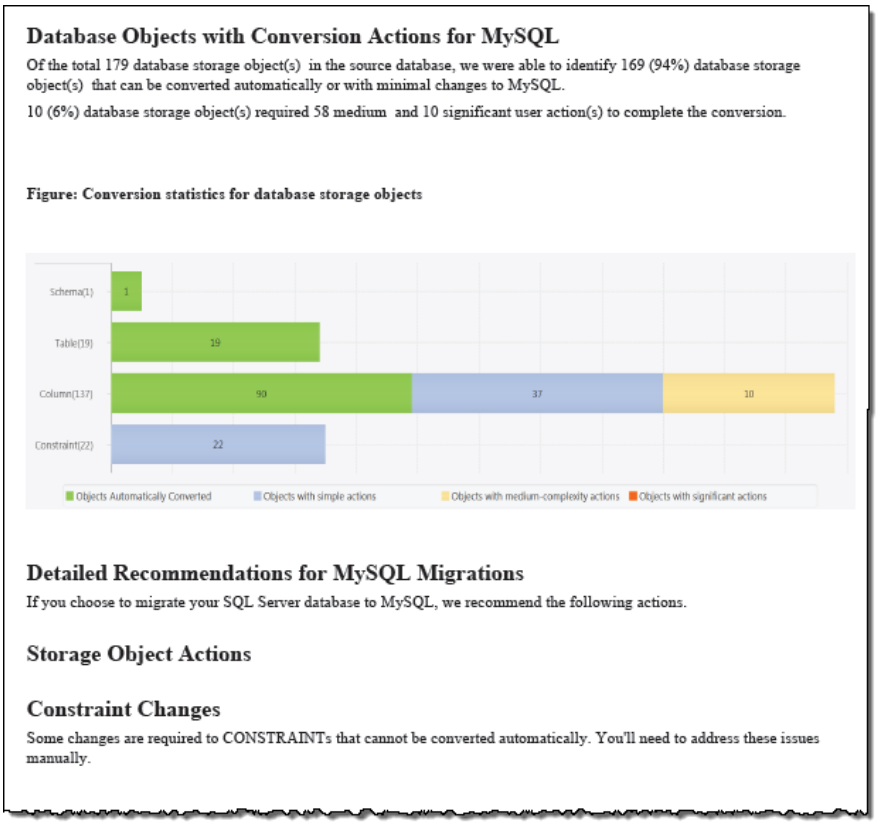

그림 8: 마이그레이션 보고서

중요: 데이터베이스 마이그레이션 프로젝트에 소요되는 작업량을 평가할 때에는 이 평가 보고서가 중요한 고려 대상 중 하나입니다. 평가 보고서를 자세히 조사하여 데이터베이스 스키마에서 어떤 코드 변경이 필요하며 변경 사항이 애플리케이션 기능 및 디자인에 어떤 영향을 미칠 것인지 결정합니다.

11. 다음 단계는 스키마 변환입니다. 변환된 스키마는 대상 데이터베이스로 즉시 적용되지 않습니다. 대신 변환된 스키마를 대상 데이터베이스에 명시적으로 적용할 때까지 로컬로 저장됩니다. 소스 데이터베이스의 스키마를 변환하려면 프로젝트의 왼쪽 패널에서 변환할 스키마 객체를 선택합니다. 다음 그림에 표시된 것과 같이 객체를 마우스 오른쪽 버튼으로 클릭하고 **[Convert schema]**를 선택합니다.

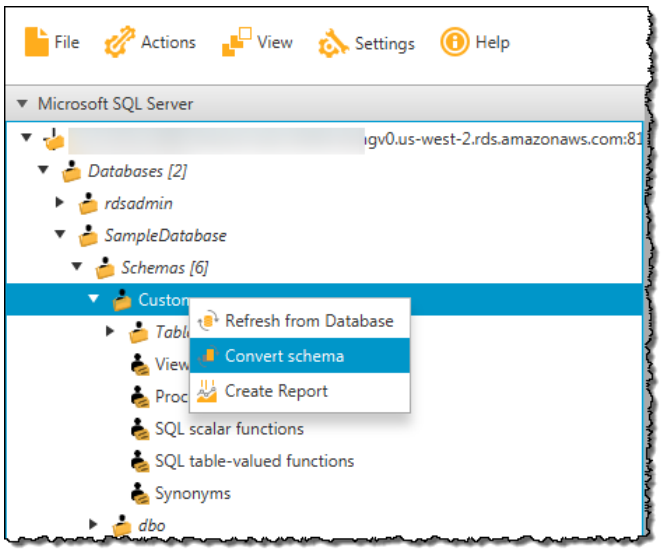

그림 9: 스키마 변환

이 작업은 변환된 스키마를 프로젝트 창의 오른쪽 패널에 추가하고 AWS Schema Conversion Tool에 의해 자동 변환된 객체를 표시합니다.

12. 다양한 방식으로 평가 보고서에 있는 작업 항목을 응답할 수 있습니다.

- 동일한 스키마를 수동으로 추가합니다. 프로젝트의 오른쪽 패널에서 **[Apply to database]**를 선택하여 대상 DB 인스턴스로 자동 변환될 수 있는 스키마 부분을 작성할 수 있습니다. 대상 DB 인스턴스에 작성된 스키마는 자동 변환될 수 없는 항목을 포함하지 않습니다. 그러한 항목은 데이터베이스 마이그레이션 평가 보고서에 나열되어 있습니다.

  대상 DB 인스턴스에 스키마를 적용했으면 자동 변환할 수 없는 항목에 대한 스키마를 대상 DB 인스턴스에 수동으로 생성할 수 있습니다. 일부 경우에는 대상 DB 인스턴스에 동일한 스키마를 생성할 수 없습니다. 해당 DB 엔진에서 제공되는 기능을 대상 DB 인스턴스에서 사용할 수 있도록 애플리케이션 및 데이터베이스의 일부를 재설계해야 할 수 있습니다. 또 다른 경우에는 간단히 자동으로 변환할 수 없는 스키마를 무시할 수 있습니다.

  주의: 스키마를 대상 DB 인스턴스에 수동으로 생성할 경우 모든 수작업의 사본을 저장해 두기 전까지 **[Apply to database]**를 선택하지 마십시오. 프로젝트에서 대상 DB 인스턴스로 스키마를 적용하면 대상 DB 인스턴스에 있는 동일한 이름의 스키마를 덮어쓰며 수동으로 추가한 모든 업데이트가 손실됩니다.

- 소스 데이터베이스 스키마를 수정하고 프로젝트의 스키마를 새로 고칩니다. 일부 항목의 경우 소스 데이터베이스의 데이터베이스 스키마를 해당

애플리케이션 아키텍처와 호환되며 대상 DB 인스턴스의 DB 엔진으로 자동 변환될 수 있는 스키마로 수정하는 방법이 가장 이상적일 수 있습니다. 소스 데이터베이스의 스키마를 업데이트하고 해당 업데이트가 애플리케이션과 호환되는 것을 확인했으면 프로젝트의 왼쪽 패널에서 **[Refresh from Database]**를 선택하여 소스 데이터베이스의 스키마를 업데이트합니다. 그런 다음 업데이트된 스키마를 변환하고 데이터베이스 마이그레이션 평가 보고서를 다시 생성할 수 있습니다. 업데이트된 스키마에 대한 작업 항목은 더 이상 표시되지 않습니다.

13. 변환된 스키마를 대상 Aurora 인스턴스에 적용할 준비가 되면 프로젝트의 오른쪽 패널에서 스키마 요소를 선택합니다. 다음 그램에 표시된 것과 같이 스키마 요소를 마우스 오른쪽 버튼으로 클릭하고**[Apply to database]**를 선택합니다.

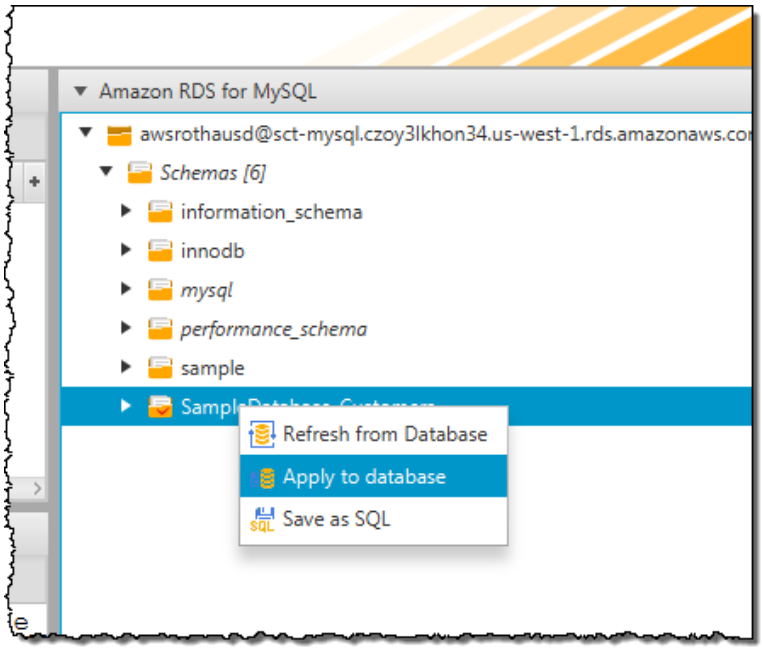

그림 10: 데이터베이스에 스키마 적용

참고: 변환된 스키마를 대상 DB 인스턴스에 처음 적용할 때 AWS Schema Conversion Tool은 대상 DB 인스턴스에 별도의 스키마(`AWS_ORACLE_EXT` 또는 `AWS_SQLSERVER_EXT`)를 추가합니다. 이 스키마는 변환된 스키마를 대상 DB 인스턴스에 쓸 때 필요한 소스 데이터베이스의 시스템 기능을 구현합니다. 이 스키마를 수정하지 마십시오. 수정하면 대상 DB 인스턴스에 쓴 변환된 스키마에서 예기치 않은 결과가 발생할 수 있습니다. 스키마가 대상 DB 인스턴스로 완전히 마이그레이션되었으며 더 이상 AWS Schema

> Conversion Tool이 필요하지 않은 경우 `AWS_ORACLE_EXT` 또는 `AWS_SQLSERVER_EXT` 스키마를 삭제할 수 있습니다.

AWS Schema Conversion Tool은 마이그레이션 툴킷에 추가되는 사용이 간편한 도구입니다. AWS Schema Conversion Tool과 관련된 추가 모범 사례는 *AWS Schema Conversion Tool 사용 설명서*의 모범 사례 주제를 참조하십시오.[17]

데이터 마이그레이션

소스 데이터베이스에서 대상 Aurora 데이터베이스로 데이터베이스 스키마를 복사했으면 다음 단계는 실제 데이터를 소스에서 대상으로 마이그레이션하는 것입니다. 데이터 마이그레이션은 다양한 도구를 사용하여 수행할 수 있지만 간단하며 작업에 필요한 모든 기능을 제공하는 AWS Database Migration Service(AWS DMS)를 사용하여 데이터를 이동하는 것이 좋습니다.

AWS DMS 소개 및 일반 접근 방식

AWS Database Migration Service(AWS DMS)를 사용하면 최소한의 다운타임으로 간단하게 생산 데이터베이스를 AWS로 마이그레이션할 수 있습니다. 애플리케이션은 데이터베이스를 마이그레이션하는 동안 실행 상태를 그대로 유지할 수 있습니다. 또한, AWS Database Migration Service는 마이그레이션 도중 및 후에 발생한 소스 데이터베이스의 데이터 변경이 대상에 연속적으로 복제될 수 있도록 해 줍니다. 마이그레이션 작업은 AWS Management Console에서 몇 분이면 설정할 수 있습니다. AWS Database Migration Service를 사용하면 Oracle, SQL Server, MySQL, PostgreSQL, Amazon Aurora, MariaDB 및 Amazon Redshift와 같이 널리 사용되는 데이터베이스 플랫폼 간에 데이터를 마이그레이션할 수 있습니다.

이 서비스는 Oracle에서 Oracle로의 동종 마이그레이션뿐 아니라 Oracle에서 Amazon Aurora 또는 SQL Server에서 MySQL로의 마이그레이션과 같은 이기종 데이터베이스 플랫폼 간 마이그레이션도 지원합니다. 마이그레이션은 일회성으로 수행하거나 고객이 복잡한 소프트웨어를 설치 또는 구성할 필요 없이 데이터베이스 간의 연속 복제를 유지할 수 있습니다.

AWS DMS는 온프레미스, Amazon EC2 기반 또는 Amazon RDS 기반으로 실행되는 데이터베이스에 사용할 수 있습니다. 그러나 소스 데이터베이스와 대상 데이터베이스 모두가 온프레미스 환경에 있는 경우에는 AWS DMS를 사용할 수 없습니다. 두 엔드포인트 중 하나는 AWS에 있어야 합니다.

AWS DMS는 특정 버전의 Oracle, SQL Server, Amazon Aurora, MySQL 및 PostgreSQL을 지원합니다. 현재 지원되는 버전은 AWS Database Migration Service 사용 설명서를 참조하십시오.[18] 하지만 이 백서는 마이그레이션 대상으로 Amazon Aurora만 중점적으로 다루고 있습니다.

마이그레이션 방식

AWS DMS는 다음과 같은 세 가지 데이터 마이그레이션 방식을 제공합니다.

기존 데이터의 마이그레이션. 이 방식은 대상 데이터베이스에 테이블을 생성하고 대상에 필요한 메타데이터를 자동으로 정의하며 해당 테이블을 소스 데이터베이스의 데이터로 채웁니다("전체 로드"). 테이블의 데이터는 효율성 향상을 위해 병렬로 로드됩니다. 테이블은 동종 마이그레이션의 경우에만 생성되며 AWS DMS는 보조 인덱스를 자동 생성하지 않습니다. 자세한 내용을 보려면 다음 내용을 계속 읽으십시오.

기존 데이터의 마이그레이션 및 지속적인 변경 복제. 이 방식은 위에 설명된 전체 로드를 수행하고 추가적으로 전체 로드 중에 소스 데이터베이스에 발생한 지속적인 변경 사항을 캡처하여 복제 인스턴스에 저장합니다. 전체 로드가 완료되면 대상 데이터베이스가 소스 데이터베이스 수준으로 업데이트될 때까지 저장된 변경 사항을 대상 데이터베이스에 적용됩니다. 또한, 추후 소스 데이터베이스에 발생하는 모든 변경 사항은 대상 데이터베이스로 복제되어 동기화 상태를 유지합니다. 이 마이그레이션 방식은 매우 적은 다운타임을 유지하면서 데이터베이스 마이그레이션을 수행하려고 할 때 매우 유용합니다.

데이터 변경만 복제. 이 방식은 소스 데이터베이스에서 복구 로그 파일의 변경 사항만 읽어 지속적으로 대상 데이터베이스에 해당 변경 사항을 적용합니다. 대상 데이터베이스를 사용할 수 없는 경우 이러한 변경 사항은 대상 데이터베이스를 사용할 수 있을 때까지 복제 인스턴스에 버퍼링됩니다.

AWS DMS에서 전체 로드 마이그레이션을 수행할 때 프로세싱 작업이 소스 데이터베이스에 부하를 유발하므로 해당 데이터베이스를 동시에 쿼리하는 애플리케이션의 성능에 영향을 미칠 수 있습니다. 이것이 문제가 되며 마이그레이션하는 동안 애플리케이션을 종료할 수 없는 경우에는 다음과 같은 접근 방식을 고려할 수 있습니다.

- 데이터베이스의 애플리케이션 부하가 가장 낮을 때 마이그레이션 실행
- 소스 데이터베이스의 읽기 전용 복제본을 생성한 다음 읽기 전용 복제본에서 AWS DMS 마이그레이션 수행

마이그레이션 절차

AWS DMS 사용에 대한 일반적인 개요는 다음과 같습니다.

1. 대상 데이터베이스를 생성합니다.
2. 스키마를 복사합니다.
3. AWS DMS 복제 인스턴스를 생성합니다.
4. 데이터베이스 소스 및 대상 엔드포인트를 정의합니다.
5. 마이그레이션 작업을 생성 및 실행합니다.

대상 데이터베이스 생성

*Amazon RDS 사용 설명서*의 Amazon Aurora DB 클러스터 생성에 설명된 절차를 사용하여 대상 Amazon Aurora 데이터베이스 클러스터를 생성합니다.[19] 대상 데이터베이스는 비즈니스 요구 사항과 일치하는 리전 및 인스턴스 유형으로 생성해야 합니다. 또한 마이그레이션 성능을 향상시키려면 대상 데이터베이스에 다중 AZ 배포가 활성화되어 있지 않아야 합니다. 이 옵션은 로드가 완료되면 활성화할 수 있습니다.

스키마 복사

또한, 이 대상 데이터베이스에 스키마를 생성해야 합니다. AWS DMS는 테이블 및 기본 키 생성을 포함한 기본적인 스키마 마이그레이션을 지원합니다. 그러나 AWS DMS는 대상 데이터베이스에 보조 인덱스, 외래 키, 저장된 프로시저, 사용자 계정 등을 자동 생성하지 않습니다. 전체 데이터베이스 마이그레이션 세부 정보는 데이터베이스 스키마 마이그레이션 섹션을 참조하십시오.

AWS DMS 복제 인스턴스 생성

AWS DMS 서비스를 사용하려면 VPC에서 실행되는 AWS DMS 복제 인스턴스를 생성해야 합니다. 이 인스턴스는 소스 데이터베이스에서 데이터를 읽고 지정된 테이블 매핑을 수행하며 대상 데이터베이스에 데이터를 씁니다. 일반적으로 큰 복제 인스턴스 크기를 사용하면 데이터베이스 마이그레이션 속도를 높일 수 있습니다(그러나 마이그레이션 속도는 소스 및 대상 데이터베이스 용량, 연결 지연 시간 등의 요소에 의해 영향을 받을 수도 있음). 또한 복제 인스턴스는 데이터베이스 마이그레이션이 완료되면 중지할 수 있습니다.

그림 11: AWS Database Migration Service

현재 AWS DMS는 복제 인스턴스로 T2 및 C4 인스턴스 클래스를 지원합니다. T2 인스턴스 클래스는 기준선 수준의 CPU 성능을 제공하고 기준선 위로 버스트할 수 있도록 설계된 저렴한 표준 인스턴스입니다. 이러한 인스턴스는 데이터베이스 마이그레이션 프로세스를 개발, 구성 및 테스트하는 데 적합하며 CPU 버스트 기능을 활용할 수 있는 주기적인 데이터 마이그레이션 작업에도 적합합니다. C4 인스턴스 클래스는 최고 수준의 프로세서 성능을 제공하고 매우 뛰어난 PPS(Packet Per Second) 성능, 낮은 네트워크 지터 및 짧은 네트워크 지연 시간을 실현하도록 설계되었습니다. 대규모 데이터베이스를 마이그레이션하면서 마이그레이션 시간을 최소화하려는 경우에는 C4 인스턴스 클래스를 사용해야 합니다.

일반적으로 전체 로드 작업은 AWS DMS 복제 인스턴스에서 상당한 양의 인스턴스 스토리지를 필요로 하지 않습니다. 그러나 전체 로드와 함께 복제를 수행하는 경우 전체 로드가 수행되는 동안 소스 데이터베이스에 대한 변경 사항이 AWS DMS 복제 인스턴스에 저장됩니다. 따라서 매우 크면서 마이그레이션이 진행되는 동안 많은 업데이트를 수신하는 소스 데이터베이스를 마이그레이션하는 경우 상당한 양의 인스턴스 스토리지가 소비될 수 있습니다. C4 인스턴스 제품군에는 100GB의 인스턴스 스토리지가 제공되며 T2 인스턴스 제품군에는 50GB가 제공됩니다. 일반적으로 이 정도 수준의 스토리지는 대부분의 마이그레이션 시나리오에 있어 충분합니다.

또한 매우 많은 트랜잭션 비율을 가진 매우 큰 데이터베이스를 마이그레이션하는 일부 극단적인 경우에는 AWS DMS 복제가 변경 속도를 따라잡지 못할 수 있습니다. 이러한 상황이 발생하면 수 분 동안 소스 데이터베이스에 대한 변경을 중지하여 복제 작업이 변경 속도를 따라잡을 수 있도록 한 다음 복제 작업을 대상 Aurora DB에 다시 연결해야 할 수 있습니다.

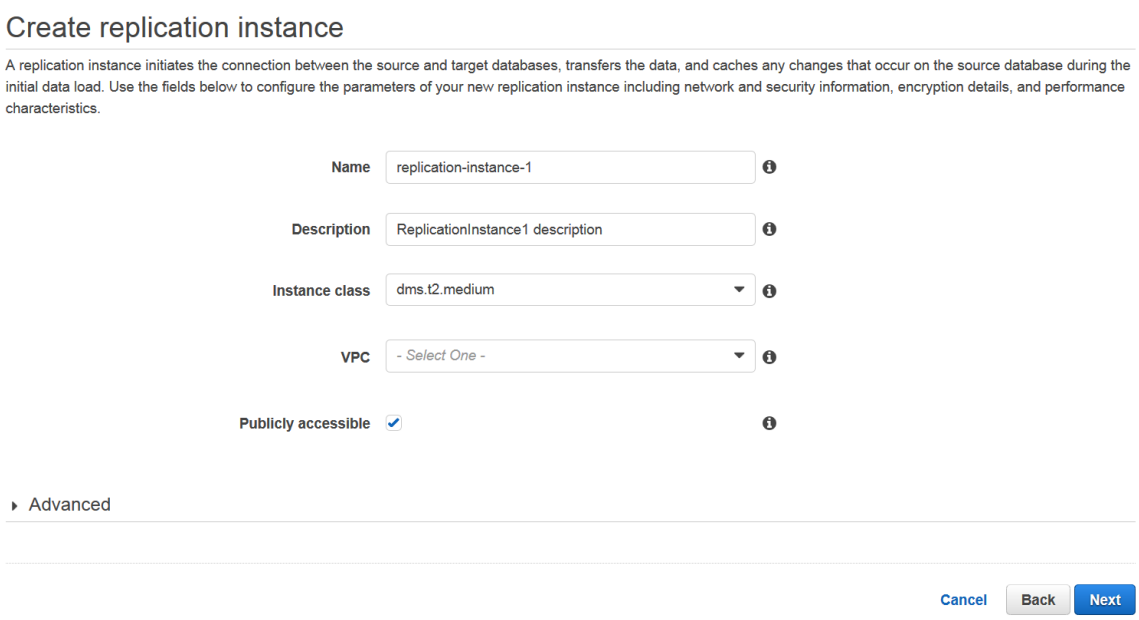

그림 12: AWS DMS 콘솔의 [Create replication instance] 페이지

데이터베이스 소스 및 대상 엔드포인트 정의

데이터베이스 엔드포인트는 데이터베이스 복제 인스턴스가 데이터베이스에 연결할 때 사용됩니다. 데이터베이스 마이그레이션을 수행하려면 소스 데이터베이스 엔드포인트와 대상 데이터베이스 엔드포인트 *모두*를 생성해야 합니다. 지정된 데이터베이스 엔드포인트는 온프레미스이거나 Amazon EC2 또는 Amazon RDS에서 실행되는 엔드포인트일 수 있지만 소스와 대상 모두가 온프레미스 환경에 있을 수는 없습니다.

정의하기 전에 데이터베이스 엔드포인트 연결을 테스트할 것이 적극 권장됩니다. 데이터베이스 엔드포인트를 생성하는 데 사용된 것과 동일한 페이지를 테스트에도 사용할 수 있습니다. 자세한 내용은 이 문서의 후반부에 설명되어 있습니다.

참고: 소스 스키마에 외래 키 제약 조건이 있는 경우 **[Advanced]** 섹션의 **[Extra connection attributes]**에 다음과 같은 항목을 입력해야 합니다.

```
initstmt=SET FOREIGN_KEY_CHECKS=0
```

이렇게 하면 대상 테이블이 로드되는 동안 외래 키 확인이 비활성화됩니다. 이는 부분 로드된 테이블에서 외래 키 확인이 실패하여 로드 작업이 중단되는 것을 방지해 줍니다.

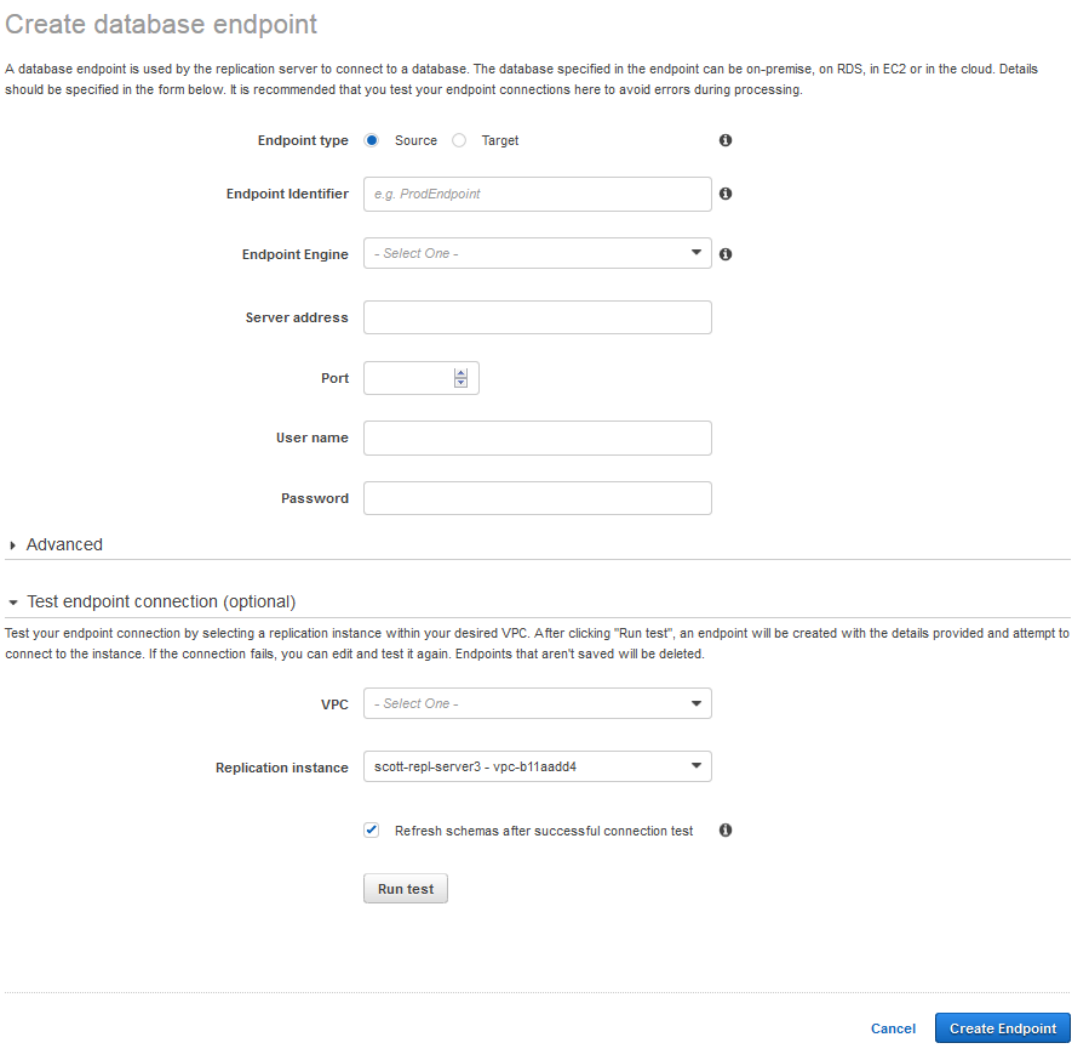

그림 13: AWS DMS 콘솔의 [Create database endpoint] 페이지

마이그레이션 작업 생성 및 실행

소스 데이터베이스 엔드포인트 및 대상 데이터베이스 엔드포인트를 성성하고 테스트했으면 데이터 마이그레이션을 수행할 작업을 생성할 수 있습니다. 작업을 생성할 때에는 생성한 복제 인스턴스, 데이터베이스 마이그레이션 방식 유형(앞에서 설명), 소스 데이터베이스 엔드포인트 및 Amazon Aurora 데이터베이스 클러스터를 위한 대상 데이터베이스 엔드포인트를 지정합니다.

또한 대상 데이터베이스에 이미 전체 스키마를 생성한 경우 **[Task Settings]**에서 **[Target table preparation mode]**를 기본값인 **[Drop tables on target]** 대신 **[Do nothing]**으로 변경해야 합니다. 기본값을 사용하면 테이블을 삭제하고 다시 생성할 때 외래 키 제약 조건과 같은 스키마 정의 요소가 유실될 수 있습니다.

작업을 생성할 때 대상 엔드포인트로 마이그레이션할 테이블과 함께 소스 스키마를 지정하는 테이블 매핑을 생성할 수 있습니다. 기본 매핑 방식은 모든 소스 테이블을 동일한 이름의 대상 테이블(있는 경우)로 마이그레이션합니다. 해당 테이블이 없는 경우 소스 테이블이 대상에 생성됩니다(작업 설정에 따라 다름). 특정 테이블만 마이그레이션하거나 필드 및 테이블 매핑 프로세스를 세부적으로 제어하려는 경우 JSON 파일을 사용하여 사용자 지정 매핑을 생성할 수도 있습니다. 또한 소스 엔드포인트에서 단일 스키마 또는 모든 스키마를 마이그레이션하도록 선택할 수 있습니다.

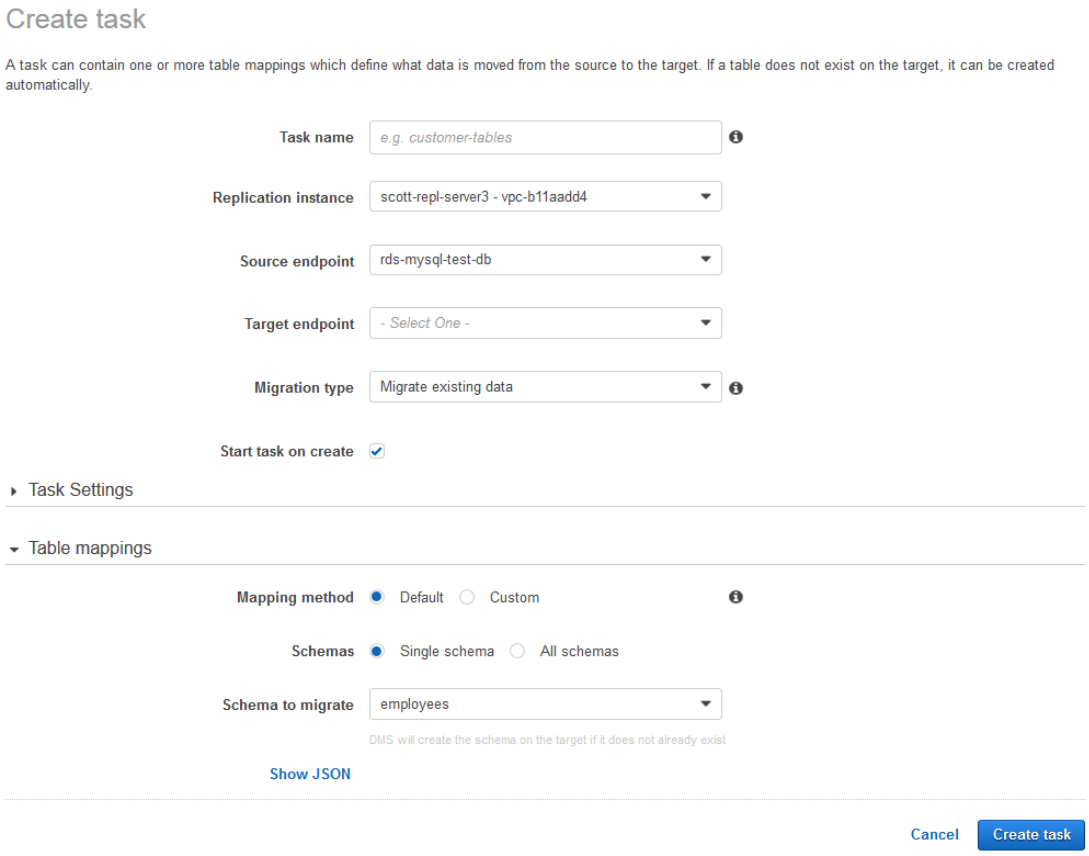

그림 14: AWS DMS 콘솔의 [Create task] 페이지

AWS Management Console을 사용하여 AWS Database Migration Service(AWS DMS) 작업의 진행 상태를 모니터링할 수 있습니다. 사용된 리소스 및 네트워크 연결도 모니터링할 수 있습니다. 다음 이미지에 표시된 것과 같이 AWS DMS 콘솔에는 작업 상태, 완료율, 경과 시간, 테이블 통계를 포함한 각 작업의 기본 통계가 표시됩니다.

또한, 작업을 선택하고 해당 작업에 대한 처리량, 초당 마이그레이션된 레코드 수, 디스크 및 메모리 사용 및 지연 시간 등의 성능 지표를 표시할 수 있습니다.

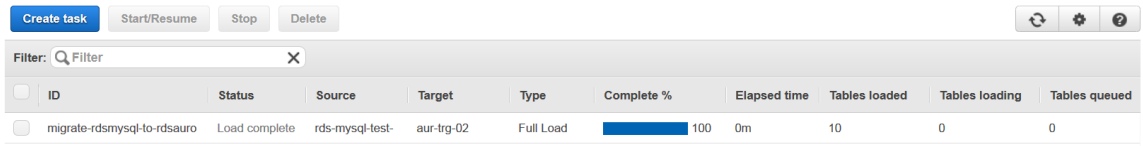

그림 15: AWS DMS 콘솔의 작업 상태

테스팅 및 전환

소스 데이터베이스에서 Amazon Aurora로 스키마와 데이터를 마이그레이션했으면 이제 마이그레이션 프로세스의 엔드-투-엔드 테스팅을 수행할 수 있습니다. 이 테스팅 접근 방식은 각 테스트 마이그레이션 후에 세부 조정되어야 하며 최종 마이그레이션 계획에는 마이그레이션된 데이터베이스에 대한 충분한 테스팅을 보장하는 테스트 계획이 포함되어야 합니다.

마이그레이션 테스팅

테스트 범주	목적
기본 수용 테스트	이러한 전환 전 테스트는 데이터 마이그레이션 프로세스 완료 시 자동으로 수행되어야 합니다. 기본 목적은 데이터 마이그레이션의 성공 여부를 확인하는 것입니다. 다음은 이러한 테스트의 일반적인 출력 내용입니다. • 처리된 총 항목 수 • 가져온 총 항목 수 • 건너뛴 총 항목 수 • 총 경고 수 • 총 오류 수 테스트를 통해 보고된 이러한 합계가 예상 값을 벗어난다면 마이그레이션이 성공하지 않았으며 다음 프로세스 단계 또는 테스팅 라운드로 이동하기 전에 문제를 해결해야 함을 의미합니다.
기능 테스트	이러한 전환 후 테스트는 Aurora를 데이터 스토리지로 사용하여 애플리케이션 기능을 연습 수행합니다. 테스트에는 자동 및 수동 테스트가 조합됩니다. 기능 테스트의 기본 목적은 Aurora로의 데이터 마이그레이션으로 인해 발생하는 애플리케이션 문제를 식별하는 것입니다.
비기능 테스트	이러한 전환 후 테스트는 다양한 수준의 부하에 따른 성능과 같은 애플리케이션의 비기능적 특성을 평가합니다.
사용자 수용 테스트	이러한 전환 후 테스트는 최종 데이터 마이그레이션 및 전환이 완료된 후 애플리케이션 사용자에 의해 실행되어야 합니다. 이러한 테스트의 목적은 애플리케이션이 조직의 기본 기능을 충족하기 위해 사용 가능한지를 최종 사용자가 결정하는 것입니다.

전환

최종 마이그레이션 및 테스팅을 완료했으면 애플리케이션을 Amazon Aurora 데이터베이스에 연결할 준비가 된 것입니다. 이 마이그레이션 단계를 *전환*이라고 합니다. 계획 및 테스팅 단계가 올바르게 실행되었으면 전환 과정에 예상치 못한 문제가 발생할 수 없습니다.

전환 전 작업

- 전환 기간 선택: 비즈니스의 중단을 최소화하면서 새 데이터베이스로 전환할 수 있는 시간대를 식별합니다. 일반적으로 데이터베이스 활동이 낮은 기간을 선택합니다(보통 야간 및/또는 주말).
- 변경 사항이 업데이트되는지 확인: 다운타임이 거의 없는 마이그레이션 접근 방식을 사용하여 소스 데이터베이스에서 대상 데이터베이스로 데이터베이스 변경을 복제하는 경우 모든 데이터베이스 변경이 캡처되며 대상 데이터베이스의 상태가 소스 데이터베이스보다 크게 뒤쳐지지 않는지 확인합니다.
- 애플리케이션 구성 변경을 위한 스크립트 준비: 전환을 수행하려면 애플리케이션 구성 파일의 데이터베이스 연결 세부 정보를 수정해야 합니다. 대규모의 복잡한 애플리케이션은 여러 위치의 연결 세부 사항에 대한 업데이트가 필요할 수 있습니다. 연결 구성을 신속하고 안정적으로 업데이트하는 데 필요한 스크립트가 있는지 확인합니다.
- 애플리케이션 중지: 소스 데이터베이스의 애플리케이션 프로세스를 중지하고 소스 데이터베이스를 읽기 전용 모드로 전환하여 더 이상 소스 데이터베이스에 대한 쓰기 작업이 수행될 수 없도록 합니다. 소스 데이터베이스 변경이 대상 데이터베이스에 완전히 적용되지 않은 경우 해당 변경 사항이 대상 데이터베이스에 완전히 전파될 때까지 기다립니다.
- 전환 전 테스트 실행: 전환 전 자동 테스트를 실행하여 데이터 마이그레이션이 성공했는지 확인합니다.

전환

- 전환 실행: 전환 전 확인이 성공적으로 완료되면 이제 애플리케이션을 Amazon Aurora에 연결할 수 있습니다. 전환 전 단계에서 생성한 스크립트를 실행하여 새 Aurora 데이터베이스에 연결하도록 애플리케이션 구성을 변경합니다.

- 애플리케이션 시작: 이 시점에서 애플리케이션을 시작할 수 있습니다. 애플리케이션이 실행되는 동안 사용자의 액세스를 중지하는 기능이 있는 경우 전환 후 확인 과정을 완료할 때까지 이 기능을 사용합니다.

전환 후 확인

- 전환 후 테스트 실행: 사전 정의된 자동 또는 수동 테스트 케이스를 실행하여 애플리케이션이 새 데이터베이스와 함께 예상대로 작동하는지 확인합니다. 좋은 전략은 데이터베이스에 쓰기 작업을 수행하는 테스트를 실행하기 전에 데이터베이스의 읽기 전용 기능부터 테스트하는 것입니다.
- 사용자 액세스 활성화 및 면밀한 모니터링: 테스트 케이스가 성공적으로 실행된 경우 애플리케이션에 사용자 액세스를 제공하여 마이그레이션 프로세스를 완료할 수 있습니다. 이 시기에는 애플리케이션과 데이터베이스 모두를 면밀하게 모니터링해야 합니다.

결론

Amazon Aurora는 클라우드용으로 설계된 고성능, 고가용성 및 엔터프라이즈급 데이터베이스입니다. Amazon Aurora를 활용하면 다른 오픈 소스 데이터베이스보다 더 뛰어난 성능 및 가용성을 얻고 대부분의 상용 데이터베이스보다 낮은 비용을 실현할 수 있습니다. 이 문서는 Amazon Aurora로의 데이터베이스 마이그레이션을 위한 최상의 방식을 식별하는 전략을 제안하며 이러한 마이그레이션의 계획 및 실행을 위한 절차를 자세히 설명하고 있습니다. 또한 이기종 마이그레이션 시나리오를 위한 권장 도구로 AWS Database Migration Service(AWS DMS)와 AWS Schema Conversion Tool이 제시됩니다. 이러한 강력한 도구는 데이터베이스 마이그레이션의 비용과 복잡성을 크게 줄일 수 있습니다.

기여자

이 문서의 작성에 도움을 준 개인 및 조직은 다음과 같습니다.

- Puneet Agarwal, Amazon Web Services 솔루션 아키텍트
- Scott Williams, Amazon Web Services 솔루션 아키텍트

참고 문헌

추가적인 도움이 필요하면 다음과 같은 자료를 참조하십시오.

- Amazon Aurora 제품 세부 정보
- Amazon Aurora FAQ
- Amazon Database Migration Service
- Amazon Database Migration Service FAQ

참고

[1] https://aws.amazon.com/rds/aurora/

[2] http://aws.amazon.com/rds/aurora/pricing/

[3] https://d0.awsstatic.com/product-marketing/Aurora/Aurora_Export_Import_Best_Practices_v1-3.pdf

[4] http://docs.aws.amazon.com/pt_br/AmazonRDS/latest/UserGuide/Aurora.Replication.html#Aurora.Overview.Replication.MySQLReplication

[5] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Migrate.html#USER_ImportAurora.PreImport

[6] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CreateSnapshot.html

[7] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/USER_CopySnapshot.html

[8] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Migrate.html#USER_ImportAuroraCluster.Console

[9] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.Connect.html

[10] https://dev.mysql.com/doc/refman/5.6/en/mysqldump.html

[11] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/Welcome.html

[12] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.GettingStarted.html

[13] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.Installing.html

[14] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.Installing.html#CHAP_SchemaConversionTool.Installing.JDBCDrivers

[15] https://forums.aws.amazon.com/forum.jspa?forumID=208

[16] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.CreateInstance.html

[17] http://docs.aws.amazon.com/SchemaConversionTool/latest/userguide/CHAP_SchemaConversionTool.BestPractices.html

[18] http://docs.aws.amazon.com/dms/latest/userguide/CHAP_Source.html

[19] http://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Aurora.CreateInstance.html