aws

개발자 가이드

Amazon Keyspaces(Apache Cassandra용)

Amazon Keyspaces(Apache Cassandra용): 개발자 가이드

Table of Contents

Amazon Keyspaces(Apache Cassandra용)란 무엇입니까?

Amazon Keyspaces(Apache Cassandra용)는 고가용성의 확장 가능한 관리형 Apache Cassandra 호환 데이터베이스 서비스입니다. Amazon Keyspaces를 사용하면 서버를 프로비저닝, 패치 또는 관리할 필요가 없으며 소프트웨어를 설치, 유지 또는 운영할 필요도 없습니다.

Amazon Keyspaces는 서버리스이므로 사용한 리소스에 대해서만 비용을 지불하면 되며 서비스는 애플리케이션 트래픽에 따라 테이블을 자동으로 확장 및 축소합니다. 사실상 무제한의 처리량과 스토리지로 초당 수천 건의 요청을 처리하는 애플리케이션을 구축할 수 있습니다.

> **Note**
>
> Apache Cassandra는 대량의 데이터를 처리하도록 설계된 오픈 소스 와이드 컬럼 데이터 스토어입니다. 자세한 내용은 Apache Cassandra를 참조하세요.

Amazon Keyspaces를 사용하면 Cassandra 워크로드를 AWS 클라우드에서 쉽게 마이그레이션, 실행 및 확장할 수 있습니다. 인프라를 배포하거나 소프트웨어를 설치하지 않고도 AWS 관리 콘솔에서 몇 번의 클릭이나 코드 몇 줄만으로 Amazon Keyspace에 키스페이스와 테이블을 생성할 수 있습니다.

Amazon Keyspaces를 사용하면 현재 사용하는 것과 동일한 Cassandra 애플리케이션 코드 및 개발자 도구를 AWS 사용하여 기존 Cassandra 워크로드를 실행할 수 있습니다.

사용 가능한 AWS 리전 엔드포인트 목록은 Amazon Keyspace의 서비스 엔드포인트를 참조하십시오.

다음 섹션을 읽고 시작하면 도움이 됩니다.

주제

- Amazon Keyspaces: 작동 방식
- Amazon Keyspaces 사용 사례
- Cassandra 쿼리 언어(CQL)란 무엇입니까?

Amazon Keyspaces: 작동 방식

Amazon Keyspaces는 Cassandra를 관리하는 데 따르는 관리 오버헤드를 제거합니다. 이유를 이해하려면 먼저 Cassandra 아키텍처로 시작한 다음 Amazon Keyspaces와 비교해 보는 것이 좋습니다.

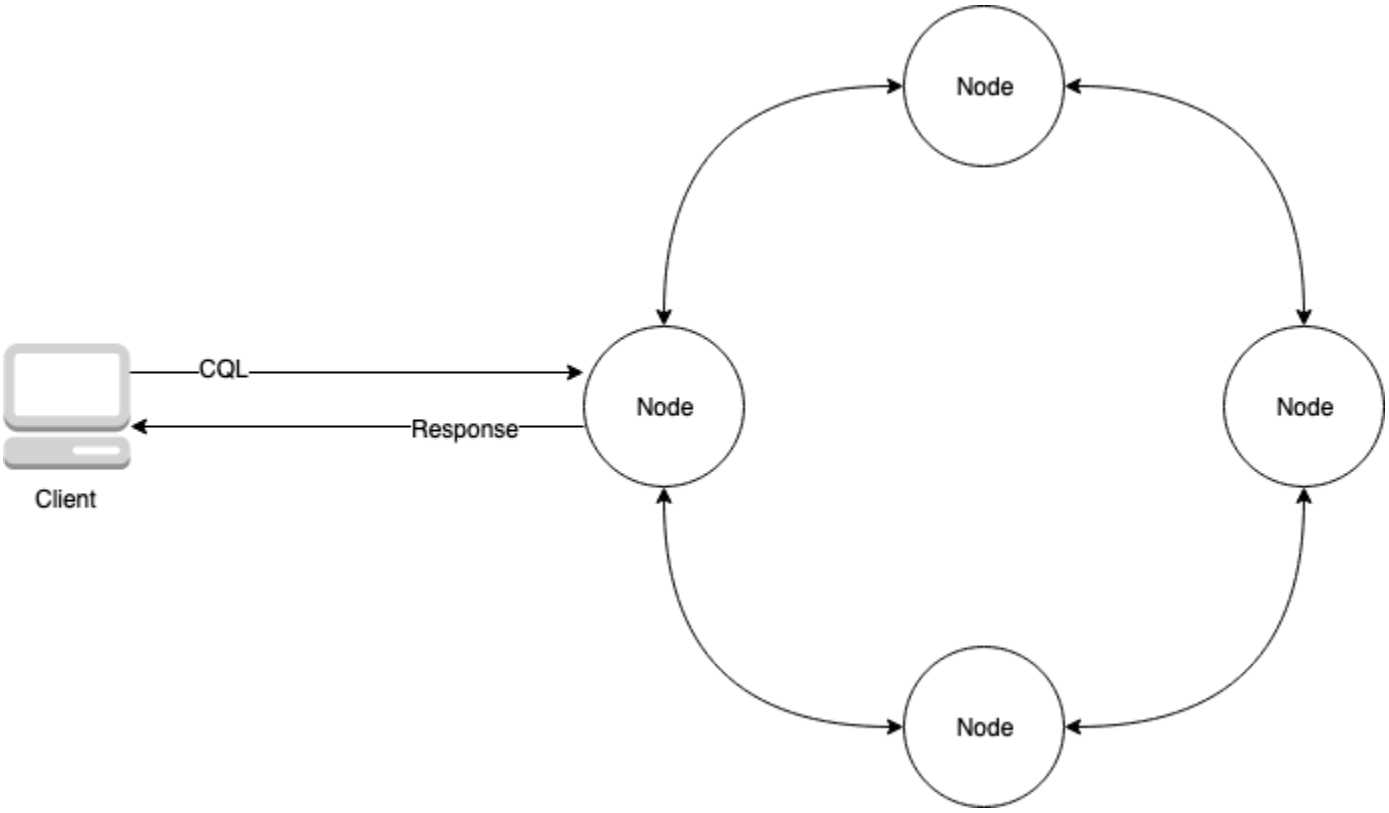
주제

상위 수준 아키텍처: Apache Cassandra vs. Amazon Keyspaces

기존 Apache Cassandra는 하나 이상의 노드로 구성된 클러스터에 배포됩니다. 클러스터 확장에 따라 각 노드를 관리하고 노드를 추가 및 제거하는 것은 사용자의 책임입니다.

클라이언트 프로그램은 노드 중 하나에 연결하고 Cassandra 쿼리 언어(CQL) 문을 실행하여 Cassandra에 액세스합니다. CQL은 관계형 데이터베이스에서 널리 사용되는 언어인 SQL과 유사합니다. Cassandra는 관계형 데이터베이스가 아니지만 CQL은 Cassandra에서 데이터를 쿼리하고 조작하기 위한 친숙한 인터페이스를 제공합니다.

다음 다이어그램은 네 개의 노드로 구성된 간단한 Apache Cassandra 클러스터를 보여 줍니다.



프로덕션 Cassandra 배포는 하나 이상의 물리적 데이터 센터에 있는 수백 대의 물리적 컴퓨터에서 실행되는 수백 개의 노드로 구성될 수 있습니다. 이로 인해 소프트웨어를 설치, 유지 관리 및 운영하는 것 외에도 서버를 프로비저닝, 패치 및 관리해야 하는 애플리케이션 개발자에게 운영상의 부담이 발생할 수 있습니다.

Amazon Keyspaces(Apache Cassandra용)를 사용하면 서버를 프로비저닝, 패치 또는 관리할 필요가 없으므로 더 나은 애플리케이션을 구축하는 데 집중할 수 있습니다. Amazon Keyspaces는 읽기 및 쓰기를 위한 두 가지 처리량 용량 모드(온디맨드 및 프로비저닝)를 제공합니다. 테이블의 처리량 용량 모드를 선택하여 워크로드의 예측 가능성과 변동성을 기반으로 읽기 및 쓰기 가격을 최적화할 수 있습니다.

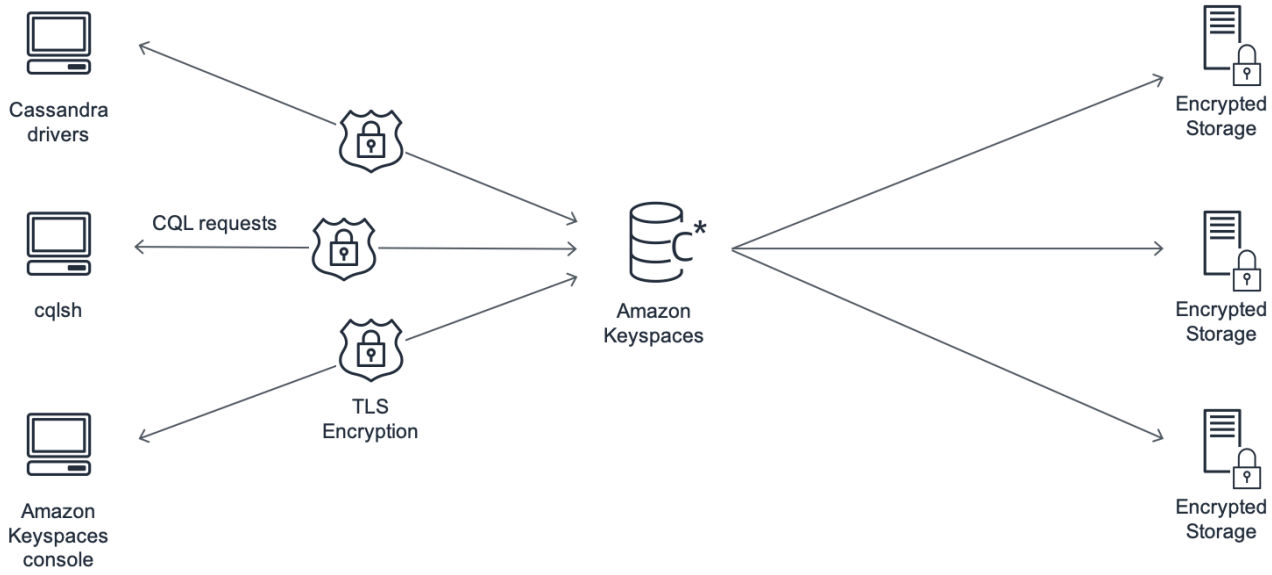
온디맨드 모드를 사용하면 애플리케이션이 실제로 수행하는 읽기 및 쓰기에 대한 비용만 지불합니다. 테이블의 처리량 용량을 미리 지정할 필요가 없습니다. Amazon Keyspaces는 애플리케이션 트래픽이 증가하거나 감소할 때 거의 즉시 수용하므로 트래픽을 예측할 수 없는 애플리케이션에 적합한 옵션입니다.

프로비저닝된 용량 모드를 사용하면 애플리케이션 트래픽을 예측할 수 있고 테이블의 용량 요구 사항을 미리 예측할 수 있는 경우 처리량 가격을 최적화하는 데 도움이 됩니다. 프로비저닝된 용량 모드를 사용하면 애플리케이션에서 수행할 것으로 예상되는 초당 읽기 및 쓰기 수를 지정합니다. 자동 크기 조정을 활성화하여 테이블의 프로비저닝된 용량을 자동으로 늘리거나 줄일 수 있습니다.

워크로드의 트래픽 패턴에 대해 자세히 알아보거나 테이블 트래픽이 많이 발생할 것으로 예상되는 주요 이벤트와 같이 트래픽이 크게 급증할 것으로 예상되는 경우 테이블의 용량 모드를 하루에 한 번 변경할 수 있습니다. 읽기 및 쓰기 용량 프로비저닝에 대한 자세한 내용은 the section called “읽기/쓰기 용량 모드” 섹션을 참조하세요.

Amazon Keyspaces(Apache Cassandra용)는 내구성과 고가용성을 위해 여러 가용 영역에 데이터 사본 3개를 저장합니다. 또한 보안에 매우 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다. 새 Amazon Keyspaces 테이블을 생성하고 모든 클라이언트 연결에 전송 계층 보안(TLS)이 필요한 경우 저장 시 암호화가 자동으로 활성화됩니다. 추가 AWS 보안 기능에는 모니터링 AWS Identity and Access Management, 가상 사설 클라우드 (VPC) 엔드포인트가 포함됩니다. 사용 가능한 모든 보안 기능의 개요는 보안 섹션을 참조하세요.

다음 다이어그램은 Amazon Keyspaces의 아키텍처를 보여 줍니다.



클라이언트 프로그램은 미리 결정된 엔드포인트(호스트 이름 및 포트 번호)에 연결하고 CQL 문을 발행하여 Amazon Keyspaces에 액세스합니다. 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

Cassandra 데이터 모델

Amazon Keyspaces에서 최적의 성능을 달성하려면 비즈니스 사례에 맞게 데이터를 모델링하는 방법이 매우 중요합니다. 잘못된 데이터 모델은 성능을 크게 저하시킬 수 있습니다.

CQL은 SQL과 비슷해 보이지만 Cassandra와 관계형 데이터베이스의 백엔드는 매우 다르므로 접근 방식도 달라야 합니다. 고려해야 할 몇 가지 중요한 문제는 다음과 같습니다.

스토리지

각 행은 레코드를 나타내고 각 열은 해당 레코드 내의 필드를 나타내는 테이블로 Cassandra 데이터를 시각화할 수 있습니다.

테이블 디자인: 쿼리 우선

CQL에는 `JOIN`이 없습니다. 따라서 비즈니스 사용 사례에 맞게 데이터의 형태와 데이터에 액세스하는 방법을 고려하여 테이블을 설계해야 합니다. 이로 인해 데이터가 중복되어 비정규화가 발생할 수 있습니다. 각 테이블을 특정 액세스 패턴에 맞게 특별히 디자인해야 합니다.

파티션

데이터는 디스크의 파티션에 저장됩니다. 데이터가 저장되는 파티션 수와 파티션 간에 데이터가 분산되는 방식은 파티션 키에 따라 결정됩니다. 파티션 키를 정의하는 방법은 쿼리 성능에 상당한 영향을 미칠 수 있습니다. 모범 사례는 the section called “파티션 키 설계” 단원을 참조하세요.

프라이머리 키

Cassandra에서 데이터는 키-값 페어로 저장됩니다. 이를 위해 모든 Cassandra 테이블에는 테이블의 각 행에 대한 키인 프라이머리 키가 있어야 합니다. 프라이머리 키는 필수 파티션 키와 선택적 클러스터링 열로 구성됩니다. 프라이머리 키를 구성하는 데이터는 테이블의 모든 레코드에서 고유해야 합니다.

- 파티션 키 - 프라이머리 키의 파티션 키 부분이 필요하며 데이터가 저장되는 클러스터의 파티션을 결정합니다. 파티션 키는 단일 열이거나 둘 이상의 열로 구성된 복합 값일 수 있습니다. 단일 열 파티션 키로 인해 단일 파티션 또는 극소수의 파티션에 대부분의 데이터가 포함되어 대부분의 디스크 I/O 작업이 수행될 경우 복합 파티션 키를 사용합니다.
- 클러스터링 열 - 프라이머리 키의 선택적 클러스터링 열 부분에 따라 각 파티션 내에서 데이터가 클러스터링되고 정렬되는 방식이 결정됩니다. 프라이머리 키에 클러스터링 열을 포함하는 경우 클러스터링 열에 하나 이상의 열이 있을 수 있습니다. 클러스터링 열에 여러 열이 있는 경우 정렬 순서는 클러스터링 열에 열이 나열된 순서(왼쪽에서 오른쪽)에 따라 결정됩니다.

NoSQL 설계 및 Amazon Keyspace에 대한 자세한 내용은 을 참조하십시오. the section called “NoSQL 설계” Amazon Keyspaces 및 데이터 모델링에 대한 자세한 내용은 을 참조하십시오. the section called “데이터 모델링”

애플리케이션에서 Amazon Keyspaces에 액세스

Amazon Keyspaces(Apache Cassandra용)는 Apache Cassandra 쿼리 언어(CQL) API를 구현하므로 이미 사용하고 있는 CQL 및 Cassandra 드라이버를 사용할 수 있습니다. 애플리케이션을 업데이트하는 것은 Amazon Keyspaces 서비스 엔드포인트를 가리키도록 Cassandra 드라이버 또는 `cqlsh` 구성을 업데이트하는 것만큼 쉽습니다. 필수 자격 증명에 대한 자세한 내용은 을 참조하십시오the section called “인증을 위한 IAM 자격 증명 AWS ”.

> **Note**
>
> 시작하는 데 도움이 되도록 Amazon Keyspaces end-to-end 코드 예제 리포지토리에서 다양한 Cassandra 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 연결하는 코드 샘플을 찾을 수 있습니다. GitHub

Cassandra 클러스터에 연결하여 테이블을 쿼리하는 다음 Python 프로그램을 고려해 보세요.

```
from cassandra.cluster import Cluster
#TLS/SSL configuration goes here

ksp = 'MyKeyspace'
tbl = 'WeatherData'

cluster = Cluster(['NNN.NNN.NNN.NNN'], port=NNNN)
session = cluster.connect(ksp)

session.execute('USE ' + ksp)

rows = session.execute('SELECT * FROM ' +  tbl)
for row in rows:
    print(row)
```

Amazon Keyspaces에서 동일한 프로그램을 실행하려면 다음을 수행해야 합니다.

- 클러스터 엔드포인트 및 포트 추가: 예를 들어 호스트를 `cassandra.us-east-2.amazonaws.com`과 같은 서비스 엔드포인트와 9142의 포트 번호로 교체할 수 있습니다.
- TLS/SSL 구성 추가: Cassandra 클라이언트 Python 드라이버를 사용하여 Amazon Keyspaces에 연결하기 위한 TLS/SSL 구성을 추가하는 방법에 대한 자세한 내용은 Cassandra Python 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스 섹션을 참조하세요.

Amazon Keyspaces 사용 사례

다음은 Amazon Keyspaces를 사용할 수 있는 몇 가지 방법입니다.

- 짧은 지연 시간이 필요한 애플리케이션 구축 — 산업 장비 유지 관리, 거래 모니터링, 차량 관리, 경로 최적화와 같이 single-digit-millisecond 지연 시간이 필요한 애플리케이션의 데이터를 고속으로 처리합니다.
- 오픈 소스 기술을 사용하여 애플리케이션 구축 — Java, Python, Ruby, Microsoft .NET, Node.js, PHP, C++, Perl, Go와 같은 다양한 프로그래밍 언어에 사용할 수 있는 오픈 소스 Cassandra API 및 드라이버를 AWS 사용하여 애플리케이션을 구축하십시오. 코드 예제는 라이브러리 및 도구 섹션을 참조하세요.
- 클라우드로 Cassandra 워크로드 이동 - Cassandra 테이블을 직접 관리하려면 시간과 비용이 많이 듭니다. Amazon Keyspaces를 사용하면 인프라를 관리하지 않고도 Cassandra 테이블을 설정, 보호 및 확장할 수 있습니다. AWS 클라우드 자세한 정보는 서버리스 리소스 관리을 참조하세요.

Cassandra 쿼리 언어(CQL)란 무엇입니까?

Cassandra 쿼리 언어(CQL)는 Apache Cassandra와 통신하기 위한 기본 언어입니다. Amazon Keyspaces(Apache Cassandra용)는 CQL 3.x API(버전 2.x와 이전 버전과 호환 가능)와 호환됩니다.

CQL 쿼리를 실행하려면 다음 중 한 가지 방법을 시도하면 됩니다.

- AWS Management Console에서 CQL 편집기를 사용합니다.
- 사용 및 cqlsh 확장. AWS CloudShell
- `cqlsh` 클라이언트에서 실행합니다.
- Apache 2.0 라이선스 Cassandra 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 실행합니다.

또한 AWS SDK와 를 사용하여 Amazon Keyspace에 액세스할 수 있습니다. AWS Command Line Interface

이러한 메서드를 사용하여 Amazon Keyspaces에 액세스하는 방법에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 액세스 섹션을 참조하세요.

CQL에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용)에 대한 CQL 언어 참조 섹션을 참조하세요.

Amazon Keyspaces(Apache Cassandra용)는 Apache Cassandra와 어떻게 비교됩니까?

Amazon Keyspace에 연결하려면 Amazon Virtual Private Cloud의 인터페이스 VPC 엔드포인트 (AWS PrivateLink) 를 사용하는 퍼블릭AWS 서비스 엔드포인트 또는 프라이빗 엔드포인트를 사용할 수 있습니다. 사용된 엔드포인트에 따라 Amazon Keyspaces는 다음 방법 중 하나로 클라이언트에 표시될 수 있습니다.

AWS 서비스 엔드포인트 연결

이는 모든 퍼블릭 엔드포인트를 통해 설정된 연결입니다. 이 경우 Amazon Keyspaces는 클라이언트에게 9노드 아파치 카산드라 3.11.2 클러스터로 나타납니다.

인터페이스 VPC 엔드포인트 연결

인터페이스 VPC 엔드포인트를 사용하여 설정된 프라이빗 연결입니다. 이 경우 Amazon Keyspaces는 클라이언트에게 3노드 아파치 카산드라 3.11.2 클러스터로 표시됩니다.

연결 유형 및 클라이언트에 표시되는 노드 수에 관계없이 Amazon Keyspace는 사실상 무제한 처리량과 스토리지를 제공합니다. 이를 위해 Amazon Keyspace는 노드를 로드 밸런서에 매핑하여 쿼리를 여러 기본 스토리지 파티션 중 하나로 라우팅합니다. 연결에 대한 자세한 내용은 the section called “작동 방식” 단원을 참조하십시오.

Amazon Keyspace는 데이터를 파티션에 저장합니다. 파티션은 솔리드 스테이트 드라이브 (SSD) 로 뒷받침되는 테이블에 스토리지를 할당하는 것입니다. Amazon Keyspace는 내구성과 고가용성을 위해 한 개 내의 여러 AWS 리전 가용 영역에 데이터를 자동으로 복제합니다. 처리량 또는 스토리지 요구 사항이 증가하면 Amazon Keyspaces가 파티션 관리를 처리하고 필요한 추가 파티션을 자동으로 프로비저닝합니다. 자세한 정보는 the section called “스토리지”을 참조하세요.

Amazon Keyspaces는 키스페이스 및 테이블 생성, 데이터 읽기, 데이터 쓰기 등 일반적으로 사용되는 모든 Cassandra 데이터 플레인 작업을 지원합니다. Amazon Keyspaces는 서버리스이므로 서버를 프로비저닝, 패치 또는 관리할 필요가 없습니다. 또한 소프트웨어를 설치, 유지 관리 또는 운영할 필요도 없습니다. 따라서 Amazon Keyspaces에서는 Cassandra 컨트롤 플레인 API 작업을 사용하여 클러스터 및 노드 설정을 관리할 필요가 없습니다.

Amazon Keyspaces는 복제 요소 및 일관성 수준과 같은 설정을 자동으로 구성하여 높은 가용성, 내구성 및 성능을 제공합니다. single-digit-millisecond Amazon Keyspace는 복원력을 높이고 로컬 읽기 지연 시간을 줄이기 위해 다중 지역 복제를 제공합니다.

주제

기능적 차이: Amazon Keyspaces와 Apache Cassandra

다음은 Amazon Keyspaces와 Apache Cassandra의 기능적 차이점입니다.

주제

Apache Cassandra API, 작업 및 데이터 유형

Amazon Keyspaces는 키스페이스 및 테이블 생성, 데이터 읽기, 데이터 쓰기 등 일반적으로 사용되는 모든 Cassandra 데이터 플레인 작업을 지원합니다. 현재 지원되는 내용을 보려면 Amazon Keyspaces에서 지원되는 Cassandra API, 작업, 함수, 데이터 유형 섹션을 참조하세요.

키스페이스 및 테이블의 비동기 생성 및 삭제

Amazon Keyspaces는 키스페이스 및 테이블 생성 및 삭제와 같은 데이터 정의 언어(DDL) 작업을 비동기적으로 수행합니다. 리소스 생성 상태를 모니터링하는 방법을 알아보려면 the section called “키스페이스 생성” 및 the section called “테이블 생성” 섹션을 참조하세요. CQL 언어 참조의 DDL 문 목록은 the section called “DDL 문” 섹션을 참조하세요.

인증 및 권한 부여

Amazon Keyspaces (Apache Cassandra용) 는 사용자 인증 및 권한 부여에 AWS Identity and Access Management (IAM) 을 사용하며, Apache Cassandra와 동일한 권한 부여 정책을 지원합니다. 따라서 Amazon Keyspaces는 Apache Cassandra의 보안 구성 명령을 지원하지 않습니다.

배치

Amazon Keyspaces는 최대 30개의 명령으로 일괄적으로 기록되지 않은 배치 명령을 지원합니다. 무조건 INSERT, UPDATE 또는 DELETE 명령만 일괄 처리할 수 있습니다. 로깅된 배치는 지원되지 않습니다.

클러스터 구성

Amazon Keyspaces는 서버리스이므로 구성할 클러스터, 호스트 또는 Java 가상 머신(JVM)이 없습니다. 압축, 캐싱, 가비지 수집 및 블룸 필터링에 대한 Cassandra의 설정은 Amazon Keyspaces에 적용할 수 없으며 지정된 경우 무시됩니다.

연결

기존 Cassandra 드라이버를 사용하여 Amazon Keyspaces와 통신할 수 있지만 드라이버를 다르게 구성해야 합니다. Amazon Keyspaces는 TCP 연결당 초당 최대 3,000개의 CQL 쿼리를 지원하지만 드라이버가 설정할 수 있는 연결 수에는 제한이 없습니다.

대부분의 오픈 소스 Cassandra 드라이버는 Cassandra에 연결 풀을 설정하고 연결 풀 전체에 걸쳐 쿼리를 로드 밸런싱합니다. Amazon Keyspaces는 드라이버에 9개의 피어 IP 주소를 노출하며 대부분의

드라이버의 기본 동작은 각 피어 IP 주소에 단일 연결을 설정하는 것입니다. 따라서 기본 설정을 사용하는 드라이버의 최대 CQL 쿼리 처리량은 초당 27,000개의 CQL 쿼리입니다.

이 수를 늘리려면 드라이버가 연결 풀에서 유지 관리하는 IP 주소당 연결 수를 늘리는 것이 좋습니다. 예를 들어 IP 주소당 최대 연결 수를 2로 설정하면 드라이버의 최대 처리량이 초당 54,000개의 CQL 쿼리로 두 배로 늘어납니다.

오버헤드를 허용하고 배포를 개선하려면 연결당 초당 500개의 CQL 쿼리를 사용하도록 드라이버를 구성하는 것이 좋습니다. 이 시나리오에서 초당 18,000개의 CQL 쿼리를 계획하려면 36개의 연결이 필요합니다. 9개 엔드포인트의 4개 연결에 대해 드라이버를 구성하면 초당 500개의 요청을 수행하는 36개의 연결이 제공됩니다. 연결 모범 사례에 대한 자세한 내용은 the section called “연결” 섹션을 참조하세요.

VPC 엔드포인트와 연결할 때 사용 가능한 엔드포인트가 적을 수 있습니다. 즉 드라이버 구성의 연결 수를 늘려야 합니다. VPC 연결 모범 사례에 대한 자세한 내용은 the section called “VPC 엔드포인트 연결” 섹션을 참조하세요.

IN 키워드

Amazon Keyspaces는 SELECT 문의 IN 키워드를 지원합니다. IN은 UPDATE 및 DELETE에서는 지원되지 않습니다. SELECT 문에서 IN 키워드를 사용하는 경우 SELECT 문에 키가 표시되는 순서대로 쿼리 결과가 반환됩니다. Cassandra에서는 결과가 사전순으로 정렬됩니다.

ORDER BY를 사용하면 페이지 매김이 비활성화된 상태에서 전체 순서 변경이 지원되지 않으며 페이지 내에서 결과가 정렬됩니다. IN 키워드에는 슬라이스 쿼리가 지원되지 않습니다. TOKENS는 IN 키워드에서 지원되지 않습니다. Amazon Keyspaces는 하위 쿼리를 생성하여 IN 키워드로 쿼리를 처리합니다. 각 하위 쿼리는 TCP 연결당 초당 3,000개의 CQL 쿼리 한도에 대한 연결로 간주됩니다. 자세한 정보는 the section called “IN SELECT 문”을 참조하세요.

CQL 쿼리 처리량 조정

Amazon Keyspaces는 TCP 연결당 초당 최대 3,000개의 CQL 쿼리를 지원하지만 드라이버가 설정할 수 있는 연결 수에는 제한이 없습니다.

대부분의 오픈 소스 Cassandra 드라이버는 Cassandra에 연결 풀을 설정하고 연결 풀 전체에 걸쳐 쿼리를 로드 밸런싱합니다. Amazon Keyspaces는 드라이버에 9개의 피어 IP 주소를 노출하며 대부분의 드라이버의 기본 동작은 각 피어 IP 주소에 단일 연결을 설정하는 것입니다. 따라서 기본 설정을 사용하는 드라이버의 최대 CQL 쿼리 처리량은 초당 27,000개의 CQL 쿼리입니다.

이 수를 늘리려면 드라이버가 연결 풀에서 유지 관리하는 IP 주소당 연결 수를 늘리는 것이 좋습니다. 예를 들어 IP 주소당 최대 연결 수를 2로 설정하면 드라이버의 최대 처리량이 초당 54,000개의 CQL 쿼리로 두 배로 늘어납니다.

연결 모범 사례에 대한 자세한 내용은 the section called “연결” 섹션을 참조하세요.

VPC 엔드포인트와 연결할 경우 사용 가능한 엔드포인트 수가 더 적습니다. 즉 드라이버 구성의 연결 수를 늘려야 합니다. VPC 엔드포인트 연결 모범 사례에 대한 자세한 내용은 을 참조하십시오. the section called “VPC 엔드포인트 연결”

FROZEN 컬렉션

Cassandra의 FROZEN 키워드는 컬렉션 데이터 유형의 여러 구성 요소를 BLOB처럼 처리되는 변경할 수 없는 단일 값으로 직렬화합니다. INSERT 및 UPDATE 문은 컬렉션 전체를 덮어씁니다.

Amazon Keyspaces는 기본적으로 최대 5개 수준의 프로즌 수집 중첩을 지원합니다. 자세한 정보는 the section called “Amazon Keyspaces 서비스 할당량”을 참조하세요.

Amazon Keyspaces는 조건부 UPDATE 또는 SELECT 문에서 프로즌 수집 전체를 사용하는 불평등 비교를 지원하지 않습니다. 컬렉션과 프로즌 수집의 동작은 Amazon Keyspaces에서 동일합니다.

클라이언트 측 타임스탬프가 있는 프로즌 수집을 사용하는 경우 쓰기 작업의 타임스탬프가 만료되거나 삭제되지 않은 기존 열의 타임스탬프와 동일한 경우 Amazon Keyspaces는 비교를 수행하지 않습니다. 대신 서버가 최신 라이터를 결정할 수 있으며 그러면 가장 최근 라이터가 승리합니다.

프로즌 수집에 대한 자세한 내용은 the section called “컬렉션 유형” 섹션을 참조하세요.

간단한 트랜잭션

Amazon Keyspaces(Apache Cassandra용)는 Apache Cassandra에서 간단한 트랜잭션(LWT)으로 알려진 INSERT, UPDATE 및 DELETE 명령에 대한 비교 및 설정 기능을 완벽하게 지원합니다. 서버리스 서비스인 Amazon Keyspaces(Apache Cassandra용)는 간단한 트랜잭션을 포함하여 모든 규모에서 일관된 성능을 제공합니다. Amazon Keyspaces를 사용하면 간단한 트랜잭션을 사용해도 성능이 저하되지 않습니다.

로드 밸런싱

system.peers 표 항목은 Amazon Keyspaces 로드 밸런서에 해당합니다. 최상의 결과를 얻으려면 라운드 로빈 로드 밸런싱 정책을 사용하고 애플리케이션의 요구 사항에 맞게 IP당 연결 수를 조정하는 것이 좋습니다.

페이지 매김

Amazon Keyspaces는 결과 집합에서 반환된 행 수가 아니라 요청을 처리하기 위해 읽은 행 수를 기준으로 결과 페이지를 매깁니다. 따라서 일부 페이지에는 필터링된 쿼리의 페이지 크기에서 지정한 수보다 적은 수의 행이 포함될 수 있습니다. 또한 Amazon Keyspaces는 1MB의 데이터를 읽은 후 자동으로 결과 페이지를 매겨 고객에게 한 자릿수 밀리초의 일관된 읽기 성능을 제공합니다. 자세한 정보는 the section called “결과 페이지 매김”을 참조하세요.

파티셔너

Amazon Keyspaces의 기본 파티셔너는 Cassandra 호환 `Murmur3Partitioner`입니다. 또한 Amazon Keyspaces `DefaultPartitioner` 또는 Cassandra 호환 `RandomPartitioner`를 사용할 수 있습니다.

Amazon Keyspaces를 사용하면 Amazon Keyspaces 데이터를 다시 로드하지 않고도 계정의 파티셔너를 안전하게 변경할 수 있습니다. 구성 변경이 완료된 후(약 10분 소요) 클라이언트는 다음에 연결할 때 새 파티셔너 설정을 자동으로 보게 됩니다. 자세한 정보는 the section called “파티셔너로 작업하기”을 참조하세요.

준비된 문

Amazon Keyspaces는 데이터 읽기 및 쓰기와 같은 데이터 조작 언어(DML) 작업에 준비된 문을 사용할 수 있도록 지원합니다. Amazon Keyspaces는 현재 테이블 및 키스페이스 생성과 같은 데이터 정의 언어(DDL) 작업에 준비된 문을 사용하는 것을 지원하지 않습니다. DDL 작업은 준비된 문 외부에서 실행해야 합니다.

범위 삭제

Amazon Keyspaces는 범위 내의 행 삭제를 지원합니다. 범위는 파티션 내의 연속된 행 집합입니다. WHERE 절을 사용하여 DELETE 작업에서 범위를 지정합니다. 범위를 전체 파티션으로 지정할 수 있습니다.

또한 관계 연산자(예: '>', '<')를 사용하거나 파티션 키를 포함하고 클러스터링 열을 하나 이상 생략하여 파티션 내 연속 행의 하위 집합이 될 범위를 지정할 수 있습니다. Amazon Keyspaces를 사용하면 한 번의 작업으로 범위 내에서 최대 1,000개의 행을 삭제할 수 있습니다. 또한 범위 삭제는 원자적이지만 분리되지는 않습니다.

시스템 테이블

Amazon Keyspaces는 Apache 2.0 오픈 소스 Cassandra 드라이버에 필요한 시스템 테이블을 채웁니다. 클라이언트에게 표시되는 시스템 테이블에는 인증된 사용자의 고유한 정보가 포함되어 있습니다. 시스템 테이블은 Amazon Keyspaces에 의해 완전히 제어되며 읽기 전용입니다.

시스템 테이블에 대한 읽기 전용 액세스가 필요하며 IAM 액세스 정책으로 시스템 테이블을 제어할 수 있습니다. 자세한 정보는 the section called “정책을 사용한 액세스 관리”을 참조하세요. Cassandra 드라이버 및 개발자 도구를 통해 AWS SDK 또는 Cassandra 쿼리 언어 (CQL) API 호출을 사용하는지 여부에 따라 시스템 테이블에 대한 태그 기반 액세스 제어 정책을 다르게 정의해야 합니다. 시스템 테이블의 태그 기반 액세스 제어에 대해 자세히 알아보려면 the section called “ 태그를 기반으로 한 Amazon Keyspaces 리소스 액세스” 섹션을 참조하세요.

Amazon VPC 엔드포인트를 사용하여 Amazon Keyspaces에 액세스하는 경우 Amazon Keyspaces가 볼 수 있는 권한을 가진 각 Amazon VPC 엔드포인트에 대한 항목이 `system.peers` 테이블에 표시됩니다. 따라서 Cassandra 드라이버는 `system.peers` 테이블의 제어 노드 자체에 대한 경고 메시지를 발행할 수 있습니다. 이 경고는 무시해도 됩니다.

타임스탬프

Amazon Keyspaces에서는 Apache Cassandra의 기본 타임스탬프와 호환되는 셀 수준 타임스탬프가 옵트인 기능입니다.

`USING TIMESTAMP` 절과 `WRITETIME` 함수는 테이블에 대해 클라이언트 측 타임스탬프가 설정된 경우에만 사용할 수 있습니다. Amazon Keyspaces의 클라이언트 측 타임스탬프에 대한 자세한 내용은 클라이언트 측 타임스탬프 섹션을 참조하세요.

Amazon Keyspaces에서 지원되는 Cassandra API, 작업, 함수, 데이터 유형

Amazon Keyspaces(Apache Cassandra용)는 Cassandra 쿼리 언어(CQL) 3.11 API(버전 2.x와 이전 버전과 호환 가능)와 호환됩니다.

Amazon Keyspaces는 키스페이스 및 테이블 생성, 데이터 읽기, 데이터 쓰기 등 일반적으로 사용되는 모든 Cassandra 데이터 플레인 작업을 지원합니다.

다음 섹션에는 지원되는 기능이 나열되어 있습니다.

주제

Cassandra API 지원

API 작업	지원
CREATE KEYSPACE	예
ALTER KEYSPACE	예
DROP KEYSPACE	예
CREATE TABLE	예
ALTER TABLE	예
DROP TABLE	예
CREATE INDEX	아니요
DROP INDEX	아니요
UNLOGGED BATCH	예
LOGGED BATCH	아니요
SELECT	예
INSERT	예
DELETE	예
UPDATE	예
USE	예

API 작업	지원
CREATE TYPE	아니요
ALTER TYPE	아니요
DROP TYPE	아니요
CREATE TRIGGER	아니요
DROP TRIGGER	아니요
CREATE FUNCTION	아니요
DROP FUNCTION	아니요
CREATE AGGREGATE	아니요
DROP AGGREGATE	아니요
CREATE MATERIALIZED VIEW	아니요
ALTER MATERIALIZED VIEW	아니요
DROP MATERIALIZED VIEW	아니요
TRUNCATE	아니요

Cassandra 컨트롤 플레인 API 지원

Amazon Keyspaces는 관리형이므로 클러스터 및 노드 설정을 관리하기 위한 Cassandra 컨트롤 플레인 API 작업이 필요하지 않습니다. 따라서 다음과 같은 Cassandra 기능은 적용되지 않습니다.

기능	이유
내구성이 뛰어난 쓰기 토글	모든 쓰기는 내구성이 뛰어납니다.
읽기 복구 설정	해당 사항 없음
GC 유예 기간(초)	해당 사항 없음

기능	이유
블룸 필터 설정	해당 사항 없음
컴팩션 설정	해당 사항 없음
컴프레션 설정	해당 사항 없음
캐싱 설정	해당 사항 없음
보안 설정	IAM으로 대체됨

Cassandra 데이터 플레인 API 지원

기능	지원
SELECT 및 INSERT 문에 대한 JSON 지원	예
정적 열	예
TTL(Time To Live)	예

Cassandra 함수 지원

지원되는 함수에 대한 자세한 내용은 the section called “기본 제공 함수” 섹션을 참조하십시오.

함수	지원
`Aggregate` 함수	아니요
`Blob` 변환	예
`Cast`	예
`Datetime` 함수	예
시간 변환 함수	예

함수	지원
TimeUuid 함수	예
Token	예
User defined functions (UDF)	아니요
Uuid	예

Cassandra 데이터 유형 지원

데이터 유형	지원	참고
ascii	예	
bigint	예	
blob	예	
boolean	예	
counter	예	
date	예	
decimal	예	
double	예	
float	예	
frozen	예	
inet	예	
int	예	
list	예	

데이터 유형	지원	참고
map	예	
set	예	
smallint	예	
text	예	
time	예	
timestamp	예	
timeuuid	예	
tinyint	예	
tuple	예	
user-defined types (UDT)	아니요	프로토콜 버퍼를 사용하여 UDT를 리팩터링하려면 Amazon Keyspaces 프로토콜 버퍼를 참조하십시오.
uuid	예	
varchar	예	
varint	예	

Amazon Keyspaces에서 지원되는 Apache Cassandra 일관성 수준

이 섹션의 항목에서는 Amazon Keyspaces(Apache Cassandra용)에서 읽기 및 쓰기 작업에 대해 지원되는 Apache Cassandra 일관성 수준에 대해 설명합니다.

주제

- 쓰기 일관성 수준
- 읽기 일관성

- 지원되지 않는 일관성 수준

쓰기 일관성 수준

Amazon Keyspaces는 내구성과 고가용성을 위해 여러 가용 영역에 모든 쓰기 작업을 세 번 복제합니다. 쓰기는 LOCAL_QUORUM 일관성 수준을 사용하여 승인되기 전에 안정적으로 저장됩니다. 1KB 쓰기당 프로비저닝된 용량 모드를 사용하는 테이블에는 1WCU(쓰기 용량 단위), 온디맨드 모드를 사용하는 테이블에는 1WRU(쓰기 요청 단위)가 청구됩니다.

cqlsh을 사용하면 다음 코드를 사용하여 현재 세션의 모든 쿼리에 대한 일관성을 LOCAL_QUORUM로 설정할 수 있습니다.

```
CONSISTENCY LOCAL_QUORUM;
```

일관성 수준을 프로그래밍 방식으로 구성하려면 적절한 Cassandra 클라이언트 드라이버를 사용하여 일관성을 설정할 수 있습니다. 예를 들어 4.x 버전 Java 드라이버를 사용하면 아래와 같이 app config 파일에 정합성 수준을 설정할 수 있습니다.

```
basic.request.consistency = LOCAL_QUORUM
```

3.x 버전 Java Cassandra 드라이버를 사용하는 경우 다음 코드 예제와 같이 .withQueryOptions(new QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM) 을 추가하여 세션의 일관성 수준을 지정할 수 있습니다.

```
Session session = Cluster.builder()
                      .addContactPoint(endPoint)
                      .withPort(portNumber)
                      .withAuthProvider(new SigV4AuthProvider("us-east-2"))
                      .withSSL()
                      .withQueryOptions(new
 QueryOptions().setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM)
                      .build()
                      .connect();
```

특정 쓰기 작업에 대한 일관성 수준을 구성하려면 Java 드라이버를 사용할 때 setConsistencyLevel 인수를 사용하여 QueryBuilder.insertInto를 호출할 때의 일관성을 정의하면 됩니다.

읽기 일관성

Amazon Keyspaces는 세 가지 읽기 일관성 수준(ONE, LOCAL_ONE, LOCAL_QUORUM)을 지원합니다. LOCAL_QUORUM 읽기 중에 Amazon Keyspaces는 이전에 성공한 모든 쓰기 작업의 최신 업데이트를 반영하는 응답을 반환합니다. 일관성 수준 ONE 또는 LOCAL_ONE를 사용하면 읽기 요청의 성능과 가용성을 개선할 수 있지만, 응답에 최근 완료된 쓰기 결과가 반영되지 않을 수 있습니다.

ONE 또는 LOCAL_ONE 일관성을 사용하는 읽기 사용량 4KB마다 프로비저닝된 용량 모드를 사용하는 테이블에는 0.5RCU(읽기 용량 단위), 온디맨드 모드를 사용하는 테이블에는 0.5RRU(읽기 요청 단위)가 청구됩니다. LOCAL_QUORUM 일관성을 사용하는 읽기 사용량 4KB마다 프로비저닝된 용량 모드를 사용하는 테이블에는 1RCU(읽기 용량 단위), 온디맨드 모드를 사용하는 테이블에는 1RRU(읽기 요청 단위)가 청구됩니다.

읽기 4KB당 테이블별 읽기 일관성 및 읽기 용량 처리량 모드를 기준으로 요금 청구

일관성 수준	프로비저닝됨	온디맨드
ONE	0.5RCU	0.5RRU
LOCAL_ONE	0.5RCU	0.5RRU
LOCAL_QUORUM	1RCU	1RRU

읽기 작업의 일관성을 다르게 지정하려면 Java 드라이버를 사용할 때 `setConsistencyLevel` 인수를 사용하여 `QueryBuilder.select`을 호출합니다.

지원되지 않는 일관성 수준

Amazon Keyspaces에서는 다음과 같은 일관성 수준을 지원하지 않으므로 예외가 발생합니다.

지원되지 않는 일관성 수준

Apache Cassandra	Amazon Keyspaces
EACH_QUORUM	지원되지 않음
QUORUM	지원되지 않음
ALL	지원되지 않음

Apache Cassandra	Amazon Keyspaces
TWO	지원되지 않음
THREE	지원되지 않음
ANY	지원되지 않음
SERIAL	지원되지 않음
LOCAL_SERIAL	지원되지 않음

Amazon Keyspaces(Apache Cassandra용) 액세스

콘솔을 사용하거나, `cqlsh` 클라이언트 AWS CloudShell, AWS SDK를 실행하여 프로그래밍 방식으로 또는 Apache 2.0 라이센스 Cassandra 드라이버를 사용하여 Amazon Keyspaces에 액세스할 수 있습니다. Amazon Keyspaces는 Apache Cassandra 3.11.2와 호환되는 드라이버와 클라이언트를 지원합니다. Amazon Keyspaces에 액세스하려면 먼저 설정을 완료한 다음 Amazon AWS Identity and Access Management Keyspace에 대한 IAM 자격 증명 액세스 권한을 부여해야 합니다.

설 AWS Identity and Access Management정

가입하여 다음을 수행하십시오. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

1. https://portal.aws.amazon.com/billing/signup을 여세요.
2. 온라인 지시 사항을 따르세요.

   등록 절차 중에는 전화를 받고 키패드로 인증 코드를 입력하는 과정이 있습니다.

   에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스 액세스 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/으로 가서 내 계정(My Account)을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

등록한 AWS 계정후에는 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 보호하고 AWS IAM Identity Center활성화하고 생성하십시오 AWS 계정 루트 사용자.

보안을 유지하세요. AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 AWS Management Console소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

루트 사용자를 사용하여 로그인하는 데 도움이 필요하면AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

   지침은 IAM 사용 설명서의 AWS 계정 루트 사용자 (콘솔) 에 대한 가상 MFA 디바이스 활성화를 참조하십시오.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

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

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

   를 ID 소스로 사용하는 방법에 대한 자습서는 사용 설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 액세스 권한이 있는 사용자로 로그인

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

  IAM Identity Center 사용자를 사용하여 로그인하는 데 도움이 필요하면 사용 설명서의 AWS 액세스 포털 로그인을 참조하십시오.AWS 로그인

추가 사용자에게 액세스 권한 할당

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

   지침은AWS IAM Identity Center 사용 설명서의 Create a permission set를 참조하세요.

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

   지침은AWS IAM Identity Center 사용 설명서의 Add groups를 참조하세요.

아마존 키스페이스 설정

Amazon Keyspaces 리소스에 대한 액세스는 IAM을 사용하여 관리합니다. IAM을 사용하면 Amazon Keyspace의 특정 리소스에 읽기 및 쓰기 권한을 부여하는 정책을 IAM 사용자, 역할 및 연동 ID에 연결할 수 있습니다.

IAM 자격 증명에 대한 권한 부여를 시작하려면 Amazon Keyspaces의 AWS 관리형 정책 중 하나를 사용할 수 있습니다.

- AmazonKeyspacesFullAccess— 이 정책은 Amazon Keyspace의 모든 리소스에 액세스하고 모든 기능에 대한 전체 액세스 권한을 부여합니다.
- AmazonKeyspacesReadOnlyAccess_v2 — 이 정책은 Amazon Keyspace에 읽기 전용 권한을 부여합니다.

관리형 정책에 정의된 작업에 대한 자세한 설명은 을 참조하십시오. the section called “AWS 관리형 정책”

IAM 자격 증명이 수행할 수 있는 작업의 범위를 제한하거나 자격 증명이 액세스할 수 있는 리소스를 제한하려면 `AmazonKeyspacesFullAccess` 관리형 정책을 템플릿으로 사용하는 사용자 지정 정책을 생성하고 필요하지 않은 모든 권한을 제거할 수 있습니다. 또한 특정 키스페이스 또는 테이블에 대한 액세스를 제한할 수 있습니다. Amazon Keyspaces에서 작업을 제한하거나 특정 리소스에 대한 액세스를 제한하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “Amazon Keyspaces에서 IAM을 사용하는 방법”

Amazon Keyspaces에 대한 IAM ID 액세스 권한을 부여하는 정책을 AWS 계정 생성하고 생성한 후 Amazon Keyspaces에 액세스하려면 다음 섹션 중 하나를 계속 진행하십시오.

- 콘솔 사용
- 사용: AWS CloudShell
- 프로그래밍 방식으로 연결

콘솔을 사용하여 Amazon Keyspaces 액세스

https://console.aws.amazon.com/keyspaces/home에서 Amazon Keyspaces용 콘솔에 액세스할 수 있습니다. AWS Management Console 액세스에 대한 자세한 내용은 IAM 사용 설명서의 AWS Management Console IAM 사용자 액세스 제어를 참조하십시오.

콘솔을 사용하여 Amazon Keyspaces에서 다음을 수행할 수 있습니다.

- 키스페이스와 테이블을 생성, 삭제, 관리합니다.
- 테이블의 모니터 탭에서 중요한 테이블 지표를 모니터링하세요.
  - 청구 가능 테이블 크기 (바이트)
  - 용량 지표
- CQL 편집기를 사용하여 쿼리를 실행합니다 (예: 데이터 삽입, 업데이트, 삭제).
- 계정의 파티셔너 구성을 변경합니다.
- 대시보드에서 계정의 성능 및 오류 지표를 볼 수 있습니다.

Amazon Keyspaces 키스페이스 및 테이블을 생성하고 샘플 애플리케이션 데이터로 설정하는 방법을 알아보려면 Amazon Keyspaces(Apache Cassandra용) 시작 섹션을 참조하세요.

Amazon AWS CloudShell Keyspace에 액세스하는 데 사용

AWS CloudShell 에서 직접 실행할 수 있는 브라우저 기반의 사전 인증된 셸입니다. AWS Management Console선호하는 셸 (Bash 또는 Z 셸) 을 사용하여 AWS 서비스에 대해 AWS CLI 명령을 실행할 수 있습니다. PowerShell 를 사용하여 Amazon Keyspaces를 `cqlsh` 사용하려면 를 설치해야 합니다. `cqlsh-expansion` `cqlsh-expansion`설치 지침은 을 참조하십시오the section called “`cqlsh-expansion` 사용”.

AWS CloudShell 에서 실행하면 콘솔에 AWS Management Console 로그인하는 데 사용한 AWS 자격 증명이 새 셸 세션에서 자동으로 사용할 수 있습니다. 이러한 AWS CloudShell 사용자 사전 인증을 통해 AWS CLI 버전 2 (셸의 컴퓨팅 환경에 사전 설치됨) 를 사용하여 `cqlsh` Amazon Keyspaces와 같은 AWS 서비스와 상호 작용할 때 자격 증명 구성을 건너뛸 수 있습니다.

IAM 권한 획득: AWS CloudShell

에서 제공하는 AWS Identity and Access Management액세스 관리 리소스를 사용하여 관리자는 IAM 사용자에게 환경 기능에 AWS CloudShell 액세스하고 사용할 수 있는 권한을 부여할 수 있습니다.

관리자가 사용자에게 액세스 권한을 부여하는 가장 빠른 방법은 AWS 관리형 정책을 사용하는 것입니다. AWS 관리형 정책은 AWS에서 생성 및 관리하는 독립 실행형 정책입니다. 다음과 같은 AWS 관리형 정책을 IAM ID에 연결할 CloudShell 수 있습니다.

- `AWSCloudShellFullAccess`: 모든 기능에 대한 전체 액세스 AWS CloudShell 권한과 함께 사용할 수 있는 권한을 부여합니다.

IAM 사용자가 수행할 수 있는 작업의 범위를 제한하려면 AWSCloudShellFullAccess 관리형 정책을 템플릿으로 AWS CloudShell사용하는 사용자 지정 정책을 생성할 수 있습니다. 에서 CloudShell 사용자가 수행할 수 있는 작업을 제한하는 방법에 대한 자세한 내용은 사용 설명서의 IAM 정책을 통한 AWS CloudShell 액세스 및 사용 관리를 참조하십시오.AWS CloudShell

> **Note**
>
> 또한 IAM 자격 증명에는 Amazon Keyspaces에 전화를 걸 수 있는 권한을 부여하는 정책이 필요합니다.

AWS 관리형 정책을 사용하여 Amazon Keyspaces 액세스 권한을 IAM ID에 부여하거나, 먼저 관리형 정책을 템플릿으로 사용하고 필요하지 않은 권한을 제거할 수 있습니다. 또한 특정 키스페이스 및 테이블에 대한 액세스를 제한하여 사용자 지정 정책을 생성할 수 있습니다. Amazon Keyspaces에 대한 다음과 같은 관리형 정책을 IAM ID에 연결할 수 있습니다.

- AmazonKeyspacesFullAccess— 이 정책은 모든 기능에 대한 전체 액세스 권한을 가진 Amazon Keyspace를 사용할 수 있는 권한을 부여합니다.

관리형 정책에 정의된 작업에 대한 자세한 설명은 을 참조하십시오the section called “AWS 관리형 정책”.

Amazon Keyspaces에서 작업을 제한하거나 특정 리소스에 대한 액세스를 제한하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “Amazon Keyspaces에서 IAM을 사용하는 방법”

를 사용하여 Amazon Keyspace와 상호 작용하기 AWS CloudShell

AWS CloudShell 에서 시작한 후에는 cqlsh 또는 명령줄 인터페이스를 사용하여 Amazon Keyspaces와 즉시 상호 작용을 시작할 수 있습니다. AWS Management Console을 (를) 아직 설치하지 않은 경우 the section called “cqlsh-expansion 사용” 자세한 단계를 참조하십시오. cqlsh-expansion

> **Note**
>
> in을 사용하는 cqlsh-expansion AWS CloudShell경우 셸 내에서 이미 인증되었으므로 전화를 걸기 전에 자격 증명을 구성하지 않아도 됩니다.

Amazon Keyspace에 연결하고 새 키스페이스를 생성합니다. 그런 다음 시스템 테이블을 읽고 키스페이스가 다음을 사용하여 생성되었는지 확인합니다. AWS CloudShell

1. 에서 탐색 표시줄에서 사용할 수 있는 다음 옵션을 CloudShell 선택하여 시작할 수 있습니다. AWS Management Console

   - CloudShell 아이콘을 선택합니다.
   - 검색 상자에 “cloudshell”을 입력하기 시작한 다음 옵션을 선택합니다. CloudShell

2. 다음 명령을 사용하여 Amazon Keyspaces에 대한 연결을 설정할 수 있습니다. *cassandra.us-east-1.amazonaws.com* 를 해당 지역의 올바른 엔드포인트로 바꿔야 합니다.

   ```
   cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
   ```

   접속에 성공하면 다음 예제와 비슷하게 출력됩니다.

   ```
   Connected to Amazon Keyspaces at cassandra.us-east-1.amazonaws.com:9142
   [cqlsh 6.1.0 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
   Use HELP for help.
   cqlsh current consistency level is ONE.
   cqlsh>
   ```

3. 이름을 mykeyspace 사용하여 새 키스페이스를 생성하십시오. 다음 명령을 사용하여 이 작업을 수행할 수 있습니다.

   ```
   CREATE KEYSPACE mykeyspace WITH REPLICATION = {'class': 'SingleRegionStrategy'};
   ```

4. 키스페이스가 생성되었는지 확인하려면 다음 명령을 사용하여 시스템 테이블에서 키스페이스를 읽을 수 있습니다.

   ```
   SELECT * FROM system_schema_mcs.keyspaces WHERE keyspace_name = 'mykeyspace';
   ```

   직접 호출이 성공하면 명령줄에 다음 출력과 비슷한 서비스의 응답이 표시됩니다.

   ```
    keyspace_name  | durable_writes | replication
   ----------------+----------------
   +-------------------------------------------------------------------------------------------
    mykeyspace     |           True | {'class':
    'org.apache.cassandra.locator.SimpleStrategy', 'replication_factor': '3'}
   ```

```
(1 rows)
```

프로그래밍 방식으로 Amazon Keyspace에 접속

이 주제에서는 Amazon Keyspaces에 프로그래밍 방식으로 접속하는 데 필요한 단계를 간략하게 설명합니다. IAM 자격 증명을 생성하는 방법을 안내하고 사용 가능한 AWS 서비스 엔드포인트를 나열합니다. 마지막 섹션에서는 cqlsh를 사용하여 Amazon Keyspaces에 접속하는 방법을 알아봅니다. 다양한 Apache Cassandra 드라이버를 사용하여 Amazon Keyspaces에 연결하는 방법에 대한 step-by-step 자습서는 을 참조하십시오. the section called “Cassandra 클라이언트 드라이버 사용” Amazon VPC 엔드포인트에서 Amazon Keyspace에 연결하는 방법을 보여주는 step-by-step 자습서는 을 참조하십시오. the section called “VPC 엔드포인트와 연결”

> **Note**
>
> 시작하는 데 도움이 되도록 Amazon Keyspaces end-to-end 코드 예제 리포지토리에서 다양한 Cassandra 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 연결하는 코드 샘플을 찾을 수 있습니다. GitHub

Amazon Keyspaces는 Apache Cassandra 3.11.2와 호환되는 드라이버와 클라이언트를 지원합니다. 의 설정 지침을 이미 완료했다고 가정합니다. AWS Amazon Keyspaces 액세스

이미 가지고 있다면 다음 주제를 참조하여 cqlsh를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스하는 방법을 알아보십시오. AWS 계정

주제

- Amazon Keyspaces에 프로그래밍 방식으로 액세스하기 위한 보안 인증 정보 생성
- Amazon Keyspaces의 서비스 엔드포인트
- cqlsh 사용하여 Amazon Keyspace에 접속
- AWS CLI 사용
- API 사용
- 아마존 키스페이스를 SDK와 함께 사용하기 AWS
- Cassandra 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스
- 튜토리얼: 아마존 Elastic Kubernetes Service에서 아마존 키스페이스에 연결

Amazon Keyspaces에 프로그래밍 방식으로 액세스하기 위한 보안 인증 정보 생성

Amazon Keyspaces 리소스에 프로그래밍 방식으로 액세스할 수 있는 보안 인증 정보를 사용자와 애플리케이션에 제공하려면 다음 중 하나를 수행할 수 있습니다.

- Cassandra가 인증 및 액세스 관리에 사용하는 기존 사용자 이름 및 비밀번호와 유사한 서비스별 자격 증명을 생성하십시오. AWS 서비스별 자격 증명은 특정 AWS Identity and Access Management (IAM) 사용자와 연결되며 해당 자격 증명을 생성한 서비스에만 사용할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 Amazon Keyspaces(Apache Cassandra용)에서 IAM 사용을 참조하세요.

  > **Warning**
  >
  > IAM 사용자는 장기 자격 증명을 보유하므로 보안상 위험이 있습니다. 이 위험을 줄이려면 이러한 사용자에게 작업을 수행하는 데 필요한 권한만 제공하고 더 이상 필요하지 않을 경우 이러한 사용자를 제거하는 것이 좋습니다.

- 보안 강화를 위해 모든 AWS 서비스에서 사용되는 IAM ID를 생성하고 임시 자격 증명을 사용하는 것이 좋습니다. Cassandra 클라이언트 드라이버용 Amazon Keyspaces SigV4 인증 플러그인을 사용하면 사용자 이름 및 암호 대신 IAM 액세스 키를 사용하여 Amazon Keyspaces에 대한 호출을 인증할 수 있습니다. Amazon Keyspaces SigV4 플러그인을 사용하여 IAM 사용자, 역할, 페더레이션 ID가 Amazon Keyspaces API 요청에서 인증할 수 있도록 지원하는 방법에 대해 자세히 알아보려면 AWS Signature Version 4 프로세스(SiGV4)를 참조하세요.

  다음 위치에서 SigV4 플러그인을 다운로드할 수 있습니다.

  - Java: https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin.
  - Node.js: https://github.com/aws/aws-sigv4-auth-cassandra-nodejs-driver-plugin.
  - Python: https://github.com/aws/aws-sigv4-auth-cassandra-python-driver-plugin.
  - Go: https://github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin.

  SigV4 인증 플러그인을 사용하여 연결을 설정하는 방법을 보여주는 코드 샘플은 the section called “Cassandra 클라이언트 드라이버 사용” 섹션을 참조하십시오.

주제

- 서비스별 보안 인증 정보 생성
- Amazon Keyspace용 AWS 자격 증명을 생성하고 구성하는 방법

서비스별 보안 인증 정보 생성

서비스별 보안 인증 정보는 Cassandra가 인증 및 액세스 관리에 사용하는 것과 같은 전통적인 사용자 이름 및 암호와 유사합니다. IAM 사용자는 서비스별 보안 인증 정보를 사용하여 특정 AWS 서비스에 액세스할 수 있습니다. 이러한 장기 자격 증명은 다른 AWS 서비스에 액세스하는 데 사용할 수 없습니다. 특정 IAM 사용자와 연결되어 있으며 다른 IAM 사용자는 사용할 수 없습니다.

> **Important**
>
> 서비스별 자격 증명은 특정 IAM 사용자와 관련된 장기 자격 증명으로, 해당 자격 증명이 생성된 서비스에만 사용할 수 있습니다. IAM 역할 또는 페더레이션 ID에 임시 자격 증명을 사용하여 모든 AWS 리소스에 액세스할 수 있는 권한을 부여하려면 Amazon Keyspaces용 SigV4 AWS 인증 플러그인을 통한 인증을 사용해야 합니다.

다음 절차 중 하나를 사용하여 서비스별 자격 증명을 생성하십시오.

콘솔을 사용하여 서비스별 보안 인증 정보 생성

콘솔을 사용하여 서비스별 보안 인증 정보 생성

1. 에 AWS Management Console 로그인하고 에서 AWS Identity and Access Management 콘솔을 엽니다. https://console.aws.amazon.com/iam/home
2. 탐색 창에서 사용자를 선택한 다음, 이전에 생성한 사용자 중 Amazon Keyspaces 권한(정책 연결)이 있는 사용자를 선택합니다.
3. 보안 인증 정보를 선택합니다. Amazon Keyspace용 보안 인증 정보에서 보안 인증 정보 생성을 선택하여 서비스별 보안 인증 정보를 생성합니다.

   이제 서비스별 자격 증명을 사용할 수 있습니다. 이 때가 암호를 다운로드하거나 볼 수 있는 유일한 시간입니다. 나중에 복구할 수 없습니다. 그러나 언제든지 암호를 재설정할 수 있습니다. 나중에 필요하므로 사용자와 암호를 안전한 위치에 저장하세요.

를 사용하여 서비스별 자격 증명을 생성하십시오. AWS CLI

를 사용하여 서비스별 자격 증명을 생성하려면 AWS CLI

서비스별 자격 증명을 생성하려면 먼저 () 를 다운로드, 설치 및 구성해야 합니다. AWS Command Line Interface AWS CLI

1. http://aws.amazon.com/cli AWS CLI 에서 다운로드하십시오.

   > **Note**
   >
   > 윈도우, macOS 또는 리눅스에서 AWS CLI 실행됩니다.

2. 사용 설명서의 AWS CLI 설치 및 AWS CLI 구성 지침을 따르십시오.AWS Command Line Interface

3. 를 사용하여 다음 명령을 실행하여 사용자가 `alice` Amazon Keyspace에 액세스할 수 있도록 서비스별 자격 증명을 생성합니다. AWS CLI

```
aws iam create-service-specific-credential \
    --user-name alice \
    --service-name cassandra.amazonaws.com
```

출력은 다음과 같습니다.

```
{
    "ServiceSpecificCredential": {
        "CreateDate": "2019-10-09T16:12:04Z",
        "ServiceName": "cassandra.amazonaws.com",
        "ServiceUserName": "alice-at-111122223333",
        "ServicePassword": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
        "ServiceSpecificCredentialId": "ACCAYFI33SINPGJEBYESF",
        "UserName": "alice",
        "Status": "Active"
    }
}
```

출력에서 `ServiceUserName` 및 `ServicePassword` 값을 기록해 둡니다. 이 값은 나중에 필요하므로 안전한 위치에 저장하세요.

> **Important**
>
> `ServicePassword`을 사용할 수 있는 유일한 시간입니다.

Amazon Keyspace용 AWS 자격 증명을 생성하고 구성하는 방법

AWS SDK 또는 Cassandra 클라이언트 드라이버와 SigV4 플러그인을 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스하려면 액세스 키가 있는 IAM 사용자 또는 역할이 필요합니다. AWS CLI프로그래밍 방식으로 사용하는 경우 AWS 프로그래밍 호출에서 ID를 확인할 수 있도록 AWS 액세스 키를 제공합니다. AWS 액세스 키는 액세스 키 ID (예: AKIAIOSFODNN7EXAMPLE) 와 보안 액세스 키 (예: bPxRfi WJALRXUTNFEMI/K7MDeng/ CYEXAMPLEKEY) 로 구성됩니다. AKIAIOSFODNN7EXAMPLE 이 주제에서는 이 프로세스의 필수 단계를 살펴봅니다.

보안 모범 사례에서는 제한된 권한을 가진 IAM 사용자를 생성하고 대신 IAM 역할을 특정 작업을 수행하는 데 필요한 권한과 연결하는 것이 좋습니다. 그러면 IAM 사용자가 일시적으로 IAM 역할을 맡아 필요한 작업을 수행할 수 있습니다. 예를 들어 Amazon Keyspaces 콘솔을 사용하는 계정의 IAM 사용자는 역할로 전환하여 콘솔에서 해당 역할의 권한을 일시적으로 사용할 수 있습니다. 사용자는 자신의 원래 권한을 포기하고 역할에 할당된 권한을 수임합니다. 사용자가 역할을 끝내면 원래 권한이 복원됩니다. 사용자가 역할을 수임하는 데 사용하는 자격 증명은 임시입니다. 반대로 IAM 사용자는 장기 자격 증명을 보유하므로 역할을 맡는 대신 권한을 직접 할당받을 경우 보안 위험이 발생할 수 있습니다. 이 위험을 줄이려면 이러한 사용자에게 작업을 수행하는 데 필요한 권한만 제공하고 더 이상 필요하지 않을 경우 이러한 사용자를 제거하는 것이 좋습니다. 역할에 대한 자세한 내용은 IAM 사용 설명서의 역할의 일반적인 시나리오: 사용자, 애플리케이션, 서비스를 참조하십시오.

주제

- 카산드라 클라이언트 드라이버용 AWS CLI, AWS SDK 또는 Amazon Keyspaces SigV4 플러그인에 필요한 자격 증명
- 계정의 Amazon Keyspaces에 프로그래밍 방식으로 액세스하기 위한 IAM 사용자 생성 AWS
- IAM 사용자의 액세스 키 생성
- IAM 사용자의 액세스 키 관리 방법
- IAM 역할 및 SigV4 플러그인을 사용하여 임시 보안 인증 정보로 Amazon Keyspaces에 접속

카산드라 클라이언트 드라이버용 AWS CLI, AWS SDK 또는 Amazon Keyspaces SigV4 플러그인에 필요한 자격 증명

IAM 사용자 또는 역할을 인증하는 데 필요한 다음 자격 증명은 다음과 같습니다.

`AWS_ACCESS_KEY_ID`

IAM 사용자 또는 역할과 관련된 AWS 액세스 키를 지정합니다.

Amazon Keyspace에 프로그래밍 방식으로 접속하려면 액세스 키 aws_access_key_id가 필요합니다.

AWS_SECRET_ACCESS_KEY

액세스 키와 연결된 보안 키를 지정합니다. 이는 액세스 키에 대한 기본적인 "암호"입니다.

Amazon Keyspace에 프로그래밍 방식으로 접속하려면 aws_secret_access_key가 필요합니다.

AWS_SESSION_TOKEN – 선택 사항

AWS Security Token Service 작업에서 직접 검색한 임시 보안 자격 증명을 사용하는 경우 필요한 세션 토큰 값을 지정합니다. 자세한 정보는 the section called “임시 보안 인증 정보를 사용하여 Amazon Keyspaces에 접속”을 참조하세요.

IAM 사용자로 연결하는 경우에는 aws_session_token이 필요하지 않습니다.

계정의 Amazon Keyspaces에 프로그래밍 방식으로 액세스하기 위한 IAM 사용자 생성 AWS

SDK 또는 SigV4 플러그인을 사용하여 Amazon AWS CLI Keyspaces에 프로그래밍 방식으로 액세스하기 위한 자격 증명을 얻으려면 먼저 IAM 사용자 또는 역할을 생성해야 합니다. AWS IAM 사용자를 생성하고 해당 IAM 사용자가 Amazon Keyspaces에 프로그래밍 방식으로 액세스할 수 있도록 구성하는 과정은 다음 단계와 같습니다.

1. AWS Management Console PowerShell, Windows용 도구에서 또는 API AWS CLI작업을 사용하여 사용자를 생성합니다. AWS 에서 사용자를 생성하면 자격 증명이 자동으로 생성됩니다. AWS Management Console
2. 프로그래밍 방식으로 사용자를 생성하는 경우 추가 단계에서 해당 사용자의 액세스 키(0액세스 키 ID 및 비밀 액세스 키)를 생성해야 합니다.
3. 사용자에게 Amazon Keyspace에 액세스할 수 있는 권한을 부여합니다.

사용자를 생성하는 데 필요한 권한에 대한 자세한 내용은 IAM 리소스에 액세스하는 데 필요한 권한을 참조하세요.

IAM 사용자 생성(콘솔)

를 AWS Management Console 사용하여 IAM 사용자를 생성할 수 있습니다.

프로그래밍 방식의 액세스 권한이 있는 IAM 사용자 생성(콘솔)

1. https://console.aws.amazon.com/iam/ 에서 AWS Management Console 로그인하고 IAM 콘솔을 엽니다.
2. 탐색 창에서 사용자(Users)와 사용자 추가(Add users)를 차례로 선택합니다.
3. 신규 사용자의 사용자 이름을 입력합니다. 의 로그인 이름입니다. AWS

   > **Note**
   >
   > 사용자 이름에는 최대 64개의 문자, 숫자 및 더하기(+), 등호(=), 쉼표(,), 마침표(.), 앳(@) 및 하이픈(-) 조합을 사용할 수 있습니다. 이름은 계정 내에서 고유해야 합니다. 대소문자는 구별하지 않습니다. 예를 들어 "TESTUSER"와 "testuser"라는 두 사용자를 만들 수는 없습니다.

4. 액세스 키 - 프로그래밍 방식 액세스를 선택하여 새 사용자를 위한 액세스 키를 생성합니다. 최종 페이지에 이르면 액세스 키를 보거나 다운로드할 수 있습니다.

   다음: 권한을 선택합니다.

5. 권한 설정 페이지에서 직접 기존 정책 연결을 선택하여 신규 사용자에게 권한을 부여합니다.

   이 옵션은 계정에서 사용할 수 있는 AWS 관리형 및 고객 관리형 정책 목록을 표시합니다. 검색 필드에 `keyspaces`를 입력하면 Amazon Keyspace와 관련된 정책만 표시됩니다.

   Amazon Keyspaces의 경우 사용 가능한 관리형 정책은 `AmazonKeyspacesFullAccess` 및 `AmazonKeyspacesReadOnlyAccess`입니다. 각 정책에 대한 자세한 내용은 the section called “AWS 관리형 정책” 섹션을 참조하십시오.

   테스트 목적으로 연결 자습서를 따르려면 새 IAM `AmazonKeyspacesReadOnlyAccess` 사용자에 대한 정책을 선택하십시오. 참고: 최소 권한 원칙을 따르고 특정 리소스에 대한 액세스를 제한하고 필요한 작업만 허용하는 사용자 지정 정책을 만드는 것이 좋습니다. IAM 정책에 대한 자세한 내용과 Amazon Keyspaces 정책 예제를 보려면 the section called “Amazon Keyspaces ID 기반 정책” 섹션을 참조하세요. 사용자 지정 권한 정책을 생성한 후에는 정책을 역할에 연결한 다음 사용자가 일시적으로 적절한 역할을 맡도록 하십시오.

   다음: 태그를 선택합니다.

6. 태그 추가(선택 사항) 페이지에서 사용자에 대한 태그를 추가하거나 다음: 검토를 선택할 수 있습니다.

7. 검토 페이지에서 이 시점까지 한 선택을 모두 확인할 수 있습니다. 계속 진행할 준비가 되었으면 사용자 생성을 선택합니다.
8. 사용자의 액세스 키(액세스 키 ID와 비밀 액세스 키)를 보려면 암호와 액세스 키 옆에 있는 Show(표시)를 선택합니다. 액세스 키를 저장하려면 .csv 다운로드(Download .csv)를 선택한 후 안전한 위치에 파일을 저장합니다.

> **Important**
>
> 비밀 액세스 키는 이 때만 확인 및 다운로드가 가능하기 때문에 이 정보가 있어야 사용자에게 SigV4 플러그인을 사용할 수 있습니다. 사용자의 새 액세스 키 ID와 비밀 액세스 키를 안전한 장소에 보관하세요. 이 단계 이후에는 보안 키에 다시 액세스할 수 없습니다.

IAM 사용자 생성(AWS CLI)

를 AWS CLI 사용하여 IAM 사용자를 생성할 수 있습니다.

프로그래밍 방식의 액세스 권한이 있는 IAM 사용자 생성(AWS CLI)

1. 다음 AWS CLI 코드를 사용하여 사용자를 생성합니다.
   - aws iam create-user
2. 사용자에게 프로그래밍 방식 액세스 권한을 부여합니다. 이를 위해서는 다음과 같은 방식으로 생성할 수 있는 액세스 키가 필요합니다.
   - AWS CLI: aws iam create-access-key
   - 윈도우용 도구 PowerShell: New-IAMAccessKey
   - IAM API: CreateAccessKey

> **Important**
>
> 비밀 액세스 키는 이 때만 확인 및 다운로드가 가능하기 때문에 이 정보가 있어야 사용자에게 SigV4 플러그인을 사용할 수 있습니다. 사용자의 새 액세스 키 ID와 비밀 액세스 키를 안전한 장소에 보관하세요. 이 단계 이후에는 보안 키에 다시 액세스할 수 없습니다.

3. 사용자 권한을 정의한 `AmazonKeyspacesReadOnlyAccess` 정책을 사용자에게 추가합니다. 주의: 모범 사례로, 사용자에게 직접 정책을 추가하는 대신 그룹에 사용자를 추가하고 그 그룹에 정책을 추가하여 사용자 권한을 관리하시는 것이 좋습니다.

   - AWS CLI: aws iam attach-user-policy

IAM 사용자의 액세스 키 생성

이미 IAM 사용자가 있으면 언제든지 새 액세스 키를 생성할 수 있습니다. IAM 사용자의 액세스 키 관리(예: 액세스 키 순환 방법)에 대한 자세한 내용은 IAM 사용자의 액세스 키 관리를 참조하세요.

IAM 사용자의 액세스 키 생성(콘솔)

1. https://console.aws.amazon.com/iam/ 에서 AWS Management Console 로그인하고 IAM 콘솔을 엽니다.
2. 탐색 창에서 사용자를 선택합니다.
3. 액세스 키를 생성하려는 사용자 이름을 선택합니다.
4. 사용자 요약 페이지에서 보안 인증 정보 탭을 선택합니다.
5. 액세스 키 섹션에서 Create access key(액세스 키 생성)를 선택합니다.

   새 액세스 키 페어를 보려면 표시를 선택합니다. 자격 증명은 다음과 비슷합니다.

   - 액세스 키 ID: AKIAIOSFODNN7EXAMPLE
   - 비밀 액세스 키: bPxRfi WJALRXUTNFEMI/K7MDENG/ CYEXAMPLEKEY

   > **Note**
   >
   > 이 대화 상자를 닫은 후에는 비밀 액세스 키에 다시 액세스할 수 없습니다.

6. 키 페어 파일을 다운로드하려면 [Download .csv file]을 선택합니다. 안전한 위치에 키를 저장합니다.
7. .csv 파일을 다운로드한 후 닫기를 선택합니다.

액세스 키를 생성하면 키 페어가 기본적으로 활성화되므로 해당 페어를 즉시 사용할 수 있습니다.

IAM 사용자의 액세스 키 관리 방법

액세스 키를 코드에 직접 포함하지 않는 것이 가장 좋은 방법입니다. AWS SDK와 AWS 명령줄 도구를 사용하면 알려진 위치에 액세스 키를 배치할 수 있으므로 코드에 보관하지 않아도 됩니다. 액세스 키를 다음 중 한 곳에 보관하십시오.

- 환경 변수 – 다중 테넌트 시스템에서 시스템 환경 변수가 아닌 사용자 환경 변수를 선택합니다.
- CLI 자격 증명 파일 - `credentials` 명령을 실행하면 `config` 및 `aws configure` 파일이 업데이트됩니다. `credentials` 파일은 Linux, macOS, Unix에서는 `~/.aws/credentials`에, Windows에서는 `C:\Users\USERNAME\.aws\credentials`에 저장됩니다. 이 파일에는 `default` 프로필 및 모든 명명된 프로필에 대한 자격 증명 세부 정보가 포함되어 있습니다.
- CLI 구성 파일 - `credentials` 명령을 실행하면 `config` 및 `aws configure` 파일이 업데이트됩니다. `config` 파일은 Linux, macOS, Unix에서는 `~/.aws/config`에, Windows에서는 `C:\Users\USERNAME\.aws\config`에 저장됩니다. 이 파일에는 기본 프로필 및 모든 명명된 프로필에 대한 구성 설정이 포함되어 있습니다.

액세스 키를 환경 변수로 저장하는 것은 the section called “Java 4.x용 인증 플러그인”의 전제 조건입니다. 클라이언트는 기본보안 인증 정보 공급자 체인을 사용하여 보안 인증 정보를 검색하며, 환경 변수로 저장된 액세스 키는 다른 모든 위치(예: 구성 파일)보다 우선합니다. 자세한 내용은 구성 설정 및 우선 순위를 참조하십시오.

다음은 기본 사용자에 대한 환경 변수를 구성할 수 있는 방법을 보여주는 예입니다.

Linux, macOS, or Unix

```
$ export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
$ export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
$ export AWS_SESSION_TOKEN=AQoDYXdzEJr...<remainder of security token>
```

환경 변수를 설정하면 사용되는 값이 변경되어 쉘 세션이 종료될 때까지 또는 변수를 다른 값으로 설정할 때까지 유지됩니다. 쉘의 스타트업 스크립트에서 변수를 설정하면 해당 변수가 향후 세션에서도 영구적으로 적용되도록 할 수 있습니다.

Windows Command Prompt

```
C:\> setx AWS_ACCESS_KEY_ID AKIAIOSFODNN7EXAMPLE
C:\> setx AWS_SECRET_ACCESS_KEY wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
C:\> setx AWS_SESSION_TOKEN AQoDYXdzEJr...<remainder of security token>
```

환경 변수를 설정하는 데 `set`을 사용하면 사용되는 값이 변경되어 현재 명령 프롬프트 세션이 종료될 때까지 또는 변수를 다른 값으로 설정할 때까지 유지됩니다. 환경 변수를 설정하는 데 `setx`를 사용하면 현재 명령 프롬프트 세션과 명령 실행 후 생성한 모든 명령 프롬프트 세션에서 사용되는 값이 변경됩니다. 명령을 실행하는 시점에 이미 실행 중인 다른 명령 쉘에는 영향을 주지 않습니다.

PowerShell

```
PS C:\> $Env:AWS_ACCESS_KEY_ID="AKIAIOSFODNN7EXAMPLE"
PS C:\> $Env:AWS_SECRET_ACCESS_KEY="wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"
PS C:\> $Env:AWS_SESSION_TOKEN="AQoDYXdzEJr...<remainder of security token>"
```

이전 예제와 같이 PowerShell 프롬프트에서 환경 변수를 설정하면 현재 세션 기간 동안만 값이 저장됩니다. 모든 PowerShell 세션과 명령 프롬프트 세션에서 환경 변수 설정을 유지하려면 제어판의 시스템 응용 프로그램을 사용하여 환경 변수 설정을 저장하십시오. 또는 PowerShell 프로필에 변수를 추가하여 향후 모든 PowerShell 세션에 사용할 변수를 설정할 수 있습니다. 환경 변수를 저장하거나 여러 세션에서 환경 변수를 유지하는 방법에 대한 자세한 내용은 PowerShell 설명서를 참조하십시오.

IAM 역할 및 SigV4 플러그인을 사용하여 임시 보안 인증 정보로 Amazon Keyspaces에 접속

보안 강화를 위해 임시 보안 인증 정보를 사용하여 SigV4 플러그인으로 인증할 수 있습니다. 대부분의 시나리오에서는 IAM 사용자와 같이 만료되지 않는 장기 액세스 키가 필요하지 않습니다. 대신, IAM 역할을 만들고 임시 보안 자격 증명을 생성할 수 있습니다. 임시 보안 자격 증명은 액세스 키 ID와 비밀 액세스 키로 구성되지만, 자격 증명이 만료되는 시간을 나타내는 보안 토큰을 포함합니다. 장기 액세스 키 대신 IAM 역할을 사용하는 방법에 대해 자세히 알아보려면 IAM 역할 (AWS API) 로 전환을 참조하십시오.

임시 보안 인증을 시작하려면 먼저 IAM 역할을 생성해야 합니다.

Amazon Keyspaces에 대한 읽기 전용 액세스를 부여하는 IAM 역할을 생성합니다.

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할, 역할 생성을 차례로 선택합니다.
3. 역할 생성 페이지의 신뢰할 수 있는 엔터티 유형 선택에서 AWS 서비스를 선택합니다. 사용 사례 선택에서 Amazon EC2를 선택한 후 다음을 선택합니다.
4. 권한 추가 페이지의 권한 정책에서 Amazon Keyspaces 읽기 전용 액세스를 선택한 후 다음을 선택합니다.

5. 이름 지정, 검토, 생성 페이지에서 역할 이름을 입력하고 신뢰할 수 있는 엔티티 선택 및 권한 추가 섹션을 검토합니다. 이 페이지에서 역할에 대한 선택적 태그를 추가할 수도 있습니다. 완료했으면 역할 생성을 선택하세요. Amazon EC2 인스턴스를 시작할 때 필요하므로 이 이름을 기억해 두십시오.

코드에서 임시 보안 자격 증명을 사용하려면 프로그래밍 방식으로 AWS Security Token Service API를 `AssumeRole` 호출하고 이전 단계에서 생성한 IAM 역할에서 생성된 자격 증명과 세션 토큰을 추출합니다. 그런 다음 해당 값을 후속 호출을 위한 자격 증명으로 사용합니다. AWS다음 예는 임시 보안 자격 증명을 사용하는 방법에 대한 유사 코드를 보여줍니다.

```
assumeRoleResult = AssumeRole(role-arn);
tempCredentials = new SessionAWSCredentials(
   assumeRoleResult.AccessKeyId,
   assumeRoleResult.SecretAccessKey,
   assumeRoleResult.SessionToken);
cassandraRequest = CreateAmazoncassandraClient(tempCredentials);
```

Python 드라이버를 사용하여 Amazon Keyspace에 액세스하는 임시 보안 인증을 구현하는 예제는 ???을 참조하세요.

`AssumeRole`, `GetFederationToken` 및 기타 API 작업을 호출하는 방법에 대한 자세한 내용은 AWS Security Token Service API 참조를 참조하세요. 이러한 호출의 결과에서 임시 보안 자격 증명 및 세션 토큰을 얻는 방법에 대한 자세한 내용은 사용하고 있는 SDK의 설명서를 참조하세요. 기본 AWS 설명서 페이지의 SDK 및 툴킷 섹션에서 모든 AWS SDK에 대한 설명서를 찾을 수 있습니다.

Amazon Keyspaces의 서비스 엔드포인트

주제

- 포트 및 프로토콜
- 글로벌 엔드포인트
- AWS GovCloud (US) Region FIPS 엔드포인트
- 중국 리전 엔드포인트

포트 및 프로토콜

Apache 2.0 라이선스를 획득한 Cassandra 드라이버를 사용하거나 AWS CLI 및 AWS SDK를 사용하여 `cqlsh` 클라이언트를 실행하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스할 수 있습니다.

아래 테이블에는 다양한 액세스 메커니즘의 포트와 프로토콜이 나와 있습니다.

프로그래밍 방식 액세스	포트	프로토콜
CQLSH	9142	TLS
Cassandra 드라이버	9142	TLS
AWS CLI	443	HTTPS
AWS SDK	443	HTTPS

TLS 연결의 경우 Amazon Keyspaces는 Starfield CA를 사용하여 서버에 대해 인증합니다. 자세한 내용은 the section called “TLS에 대한 cqlsh 연결을 수동으로 구성하는 방법” 또는 the section called “Cassandra 클라이언트 드라이버 사용” 장에서 드라이버의 시작하기 전에 섹션을 참조하세요.

글로벌 엔드포인트

Amazon Keyspaces는 다음 AWS 리전에서 사용할 수 있습니다. 이 표는 각 리전에서 사용 가능한 서비스 엔드포인트를 보여 줍니다.

리전 이름	리전	엔드포인트	프로토콜
미국 동부(오하이오)	us-east-2	cassandra.us-east-2.amazonaws.com	HTTPS 및 TLS
미국 동부(버지니아 북부)	us-east-1	cassandra.us-east-1.amazonaws.com cassandra-fips.us-east-1.amazonaws.com	HTTPS 및 TLS TLS
미국 서부(캘리포니아 북부)	us-west-1	cassandra.us-west-1.amazonaws.com	HTTPS 및 TLS
미국 서부(오레곤)	us-west-2	cassandra.us-west-2.amazonaws.com cassandra-fips.us-west-2.amazonaws.com	HTTPS 및 TLS

리전 이름	리전	엔드포인트	프로토콜
			TLS
아시아 태평양(홍콩)	ap-east-1	cassandra.ap-east-1.amazonaws.com	HTTPS 및 TLS
아시아 태평양(뭄바이)	ap-south-1	cassandra.ap-south-1.amazonaws.com	HTTPS 및 TLS
아시아 태평양(서울)	ap-northeast-2	cassandra.ap-northeast-2.amazonaws.com	HTTPS 및 TLS
아시아 태평양(싱가포르)	ap-southeast-1	cassandra.ap-southeast-1.amazonaws.com	HTTPS 및 TLS
아시아 태평양(시드니)	ap-southeast-2	cassandra.ap-southeast-2.amazonaws.com	HTTPS 및 TLS
아시아 태평양(도쿄)	ap-northeast-1	cassandra.ap-northeast-1.amazonaws.com	HTTPS 및 TLS
캐나다(중부)	ca-central-1	cassandra.ca-central-1.amazonaws.com	HTTPS 및 TLS
유럽(프랑크푸르트)	eu-central-1	cassandra.eu-central-1.amazonaws.com	HTTPS 및 TLS
유럽(아일랜드)	eu-west-1	cassandra.eu-west-1.amazonaws.com	HTTPS 및 TLS
유럽(런던)	eu-west-2	cassandra.eu-west-2.amazonaws.com	HTTPS 및 TLS

리전 이름	리전	엔드포인트	프로토콜	
유럽(파리)	eu-west-3	cassandra.eu-west-3.amazonaws.com	HTTPS 및 TLS	
유럽(스톡홀름)	eu-north-1	cassandra.eu-north-1.amazonaws.com	HTTPS 및 TLS	
중동(바레인)	me-south-1	cassandra.me-south-1.amazonaws.com	HTTPS 및 TLS	
남아메리카(상파울루)	sa-east-1	cassandra.sa-east-1.amazonaws.com	HTTPS 및 TLS	
AWS GovCloud(미국 동부)	us-gov-east-1	cassandra.us-gov-east-1.amazonaws.com	HTTPS 및 TLS	
AWS GovCloud(미국 서부)	us-gov-west-1	cassandra.us-gov-west-1.amazonaws.com	HTTPS 및 TLS	

AWS GovCloud (US) Region FIPS 엔드포인트

AWS GovCloud (US) Region에서 사용 가능한 FIPS 엔드포인트 자세한 내용은 AWS GovCloud (US) 사용 설명서의 Amazon Keyspaces를 참조하세요.

리전 이름	리전	FIPS 엔드포인트	프로토콜
AWS GovCloud(미국 동부)	us-gov-east-1	cassandra.us-gov-east-1.amazonaws.com	HTTPS 및 TLS

리전 이름	리전	FIPS 엔드포인트	프로토콜
AWS GovCloud(미국 서부)	us-gov-west-1	cassandra.us-gov-west-1.amazonaws.com	HTTPS 및 TLS

중국 리전 엔드포인트

AWS 중국 리전에서 사용할 수 있는 Amazon Keyspaces 엔드포인트는 다음과 같습니다.

이러한 엔드포인트에 액세스하려면 중국 리전에 고유한 별도의 계정 보안 인증 세트에 가입해야 합니다. 자세한 내용은 중국 가입, 계정 및 보안 인증을 참조하세요.

리전 이름	리전	엔드포인트	프로토콜
중국(베이징)	cn-north-1	cassandra.cn-north-1.amazonaws.com.cn	HTTPS 및 TLS
중국(닝샤)	cn-northwest-1	cassandra.cn-northwest-1.amazonaws.com.cn	HTTPS 및 TLS

`cqlsh` 사용하여 Amazon Keyspace에 접속

`cqlsh`를 사용하여 Amazon Keyspaces에 접속하려면 `cqlsh-expansion`를 사용할 수 있습니다. 이 도구 키트에는 Apache Cassandra와의 완전한 호환성을 유지하면서 Amazon Keyspaces에 사전 구성된 `cqlsh` 및 도우미와 같은 일반적인 Apache Cassandra 도구가 포함되어 있습니다. `cqlsh-expansion`는 SigV4 인증 플러그인을 통합하여 사용자 이름과 암호 대신 IAM 액세스 키를 사용하여 접속할 수 있습니다. Amazon Keyspaces는 서버리스이므로 전체 Apache Cassandra 배포는 설치하지 않고 `cqlsh` 스크립트만 설치하면 접속할 수 있습니다. 이 간단한 설치 패키지에는 Python을 지원하는 모든 플랫폼에 설치할 수 있는 `cqlsh-expansion` 및 클래식 `cqlsh` 스크립트가 포함되어 있습니다.

`cqlsh`에 대한 일반적인 정보는 `cqlsh`: CQL 쉘 섹션을 참조하세요.

주제

- cqlsh-expansion을 사용하여 Amazon Keyspace에 접속
- TLS에 대한 cqlsh 연결을 수동으로 구성하는 방법

cqlsh-expansion을 사용하여 Amazon Keyspace에 접속

cqlsh-expansion 설치 및 구성

1. cqlsh-expansion Python 패키지를 설치하려면 pip 명령을 실행할 수 있습니다. 그러면 종속성 목록이 포함된 파일과 함께 pip install을 사용하여 컴퓨터에 cqlsh-expansion 스크립트가 설치됩니다. --user flag가 pip에 플랫폼용 Python 사용자 설치 디렉터리를 사용하도록 지시합니다. Unix 기반 시스템에서는 ~/.local/ 디렉터리여야 합니다.

   cqlsh-expansion을 설치하려면 Python 3가 필요합니다. Python 버전을 확인하려면 Python --version를 사용하세요. 설치하려면 다음 명령을 실행할 수 있습니다.

   ```
   python3 -m pip install --user cqlsh-expansion
   ```

   결과는 다음과 비슷해야 합니다.

   ```
   Collecting cqlsh-expansion
     Downloading cqlsh_expansion-0.9.6-py3-none-any.whl (153 kB)
        ######################################## 153.7/153.7 KB 3.3 MB/s eta 0:00:00
   Collecting cassandra-driver
     Downloading cassandra_driver-3.28.0-cp310-cp310-
   manylinux_2_17_x86_64.manylinux2014_x86_64.whl (19.1 MB)
        ######################################## 19.1/19.1 MB 44.5 MB/s eta 0:00:00
   Requirement already satisfied: six>=1.12.0 in /usr/lib/python3/dist-packages (from
    cqlsh-expansion) (1.16.0)
   Collecting boto3
     Downloading boto3-1.29.2-py3-none-any.whl (135 kB)
        ######################################## 135.8/135.8 KB 17.2 MB/s eta 0:00:00
   Collecting cassandra-sigv4>=4.0.2
     Downloading cassandra_sigv4-4.0.2-py2.py3-none-any.whl (9.8 kB)
   Collecting botocore<1.33.0,>=1.32.2
     Downloading botocore-1.32.2-py3-none-any.whl (11.4 MB)
        ######################################## 11.4/11.4 MB 60.9 MB/s eta 0:00:00
   Collecting s3transfer<0.8.0,>=0.7.0
     Downloading s3transfer-0.7.0-py3-none-any.whl (79 kB)
        ######################################## 79.8/79.8 KB 13.1 MB/s eta 0:00:00
   Collecting jmespath<2.0.0,>=0.7.1
     Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
   Collecting geomet<0.3,>=0.1
     Downloading geomet-0.2.1.post1-py3-none-any.whl (18 kB)
   Collecting python-dateutil<3.0.0,>=2.1
     Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
   ```

```
     ######################################## 247.7/247.7 KB 33.1 MB/s eta 0:00:00
Requirement already satisfied: urllib3<2.1,>=1.25.4 in /usr/lib/python3/dist-
packages (from botocore<1.33.0,>=1.32.2->boto3->cqlsh-expansion) (1.26.5)
Requirement already satisfied: click in /usr/lib/python3/dist-packages (from
 geomet<0.3,>=0.1->cassandra-driver->cqlsh-expansion) (8.0.3)
Installing collected packages: python-dateutil, jmespath, geomet, cassandra-driver,
 botocore, s3transfer, boto3, cassandra-sigv4, cqlsh-expansion
  WARNING: The script geomet is installed in '/home/ubuntu/.local/bin' which is not
 on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
 warning, use --no-warn-script-location.
  WARNING: The scripts cqlsh, cqlsh-expansion and cqlsh-expansion.init are
 installed in '/home/ubuntu/.local/bin' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
 warning, use --no-warn-script-location.
Successfully installed boto3-1.29.2 botocore-1.32.2 cassandra-driver-3.28.0
 cassandra-sigv4-4.0.2 cqlsh-expansion-0.9.6 geomet-0.2.1.post1 jmespath-1.0.1
 python-dateutil-2.8.2 s3transfer-0.7.0
```

설치 디렉터리가 에 PATH 없는 경우 운영 체제의 지침에 따라 추가해야 합니다. 다음은 우분투 리눅스의 한 예입니다.

```
export PATH="$PATH:/home/ubuntu/.local/bin"
```

다음 명령을 실행하여 패키지가 설치되었는지 확인할 수 있습니다.

```
cqlsh-expansion --version
```

결과는 다음과 같아야 합니다.

```
cqlsh 6.1.0
```

2. cqlsh-expansion를 구성하려면 설치 후 스크립트를 실행하여 다음 단계를 자동으로 완료할 수 있습니다.

   1. .cassandra 디렉터리가 아직 없으면 사용자 홈 디렉터리에 생성합니다.
   2. 사전 구성된 cqlshrc 구성 파일을 .cassandra 디렉터리에 복사합니다.
   3. Starfield 디지털 인증서를 .cassandra 디렉터리에 복사합니다. Amazon Keyspaces는 이 인증서를 사용하여 전송 계층 보안(TLS) 과의 보안 연결을 구성합니다. 전송 중 암호화는 Amazon Keyspaces와 주고 받는 데이터를 암호화하여 추가 데이터 보호 계층을 제공합니다.

스크립트를 먼저 검토하려면 post_install.py의 Github 리포지토리에서 액세스할 수 있습니다.

다음 명령을 실행하여 스크립트를 사용할 수 있습니다.

```
cqlsh-expansion.init
```

> **Note**
>
> pip uninstall를 사용하여 cqlsh-expansion을 제거해도 설치 후 스크립트로 생성된 디렉터리와 파일은 제거되지 않으므로 수동으로 삭제해야 합니다.

cqlsh-expansion을 사용하여 Amazon Keyspace에 접속

1. 를 AWS 리전 구성하고 사용자 환경 변수로 추가합니다.

   Unix 기반 시스템에서 기본 지역을 환경 변수로 추가하려면 다음 명령을 실행할 수 있습니다. 이 예제에서는 미국 동부(버지니아 북부)를 사용합니다.

   ```
   export AWS_DEFAULT_REGION=us-east-1
   ```

   다른 플랫폼을 포함하여 환경 변수를 설정하는 방법에 대한 자세한 내용은 환경 변수 설정 방법을 참조하세요.

2. 서비스 엔드포인트를 찾습니다.

   리전에 적절한 서비스 엔드포인트를 선택합니다. Amazon Keyspaces에 사용할 수 있는 엔드포인트를 검토하려면 the section called “서비스 엔드포인트”을 참조하세요. 이 예에서는 엔드포인트 cassandra.us-east-1.amazonaws.com을 사용합니다.

3. 인증 방식을 구성합니다.

   보안을 강화하기 위한 권장 방법은 IAM 액세스 키(IAM 사용자, 역할, 페더레이션 ID)로 접속하는 것입니다.

   IAM 액세스 키로 접속하려면 먼저 다음 단계를 완료해야 합니다.

a. IAM 사용자를 생성하거나, 모범 사례에 따라 IAM 사용자가 위임할 수 있는 IAM 역할을 생성하십시오. IAM 액세스 키를 생성하는 방법에 대한 자세한 정보는 the section called “인증을 위한 IAM 자격 증명 AWS ” 섹션을 참조하세요.

b. 역할(또는 IAM 사용자)에게 최소한 Amazon Keyspace에 대한 읽기 전용 액세스 권한을 부여하는 IAM 정책을 생성합니다. IAM 사용자 또는 역할이 Amazon Keyspaces에 접속하는 데 필요한 권한에 대한 자세한 내용은 the section called “Amazon Keyspaces 테이블에 액세스” 섹션을 참조하세요.

c. 다음 예제와 같이 IAM 사용자의 액세스 키를 사용자 환경 변수에 추가합니다.

```
export AWS_ACCESS_KEY_ID=AKIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
```

다른 플랫폼을 포함하여 환경 변수를 설정하는 방법에 대한 자세한 내용은 환경 변수 설정 방법을 참조하세요.

> **Note**
>
> Amazon EC2 인스턴스에서 연결하는 경우, 인스턴스에서 Amazon Keyspace로의 트래픽을 허용하는 보안 그룹의 아웃바운드 규칙도 구성해야 합니다. EC2 아웃바운드 규칙을 보고 편집하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹에 규칙 추가를 참조하십시오.

4. `cqlsh-expansion` 및 SigV4 인증을 사용하여 Amazon Keyspaces에 접속합니다.

`cqlsh-expansion`를 사용하여 Amazon Keyspaces에 접속하려면 다음 명령을 사용합니다. 서비스 엔드포인트를 해당 리전의 올바른 엔드포인트로 바꿔야 합니다.

```
cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 --ssl
```

접속에 성공하면 다음 예제와 비슷하게 출력됩니다.

```
Connected to Amazon Keyspaces at cassandra.us-east-1.amazonaws.com:9142
[cqlsh 6.1.0 | Cassandra 3.11.2 | CQL spec 3.4.4 | Native protocol v4]
Use HELP for help.
cqlsh current consistency level is ONE.
cqlsh>
```

연결 오류가 발생하는 경우 문제 해결 정보를 the section called “Cqlsh 연결 오류” 참조하십시오.

- 서비스별 보안 인증 정보로 Amazon Keyspaces에 접속합니다.

  Cassandra가 인증에 사용하는 기존 사용자 이름 및 암호 조합으로 접속하려면 먼저 the section called “서비스별 보안 인증 정보”에 설명된 대로 Amazon Keyspaces에 대한 서비스별 보안 인증 정보를 생성해야 합니다. 또한 해당 사용자에게 Amazon Keyspace에 액세스할 수 있는 권한을 부여해야 합니다. 자세한 내용은 the section called “Amazon Keyspaces 테이블에 액세스” 섹션을 참조하세요.

  사용자에 대한 서비스별 보안 인증 정보와 권한을 생성한 후에는 일반적으로 사용자 디렉터리 경로 ~/.cassandra/에 있는 cqlshrc 파일을 업데이트해야 합니다. cqlshrc 파일에서 Cassandra [authentication] 섹션으로 이동하여 다음 예제와 같이 “;” 문자를 사용하여 [auth_provider]에 있는 SigV4 모듈과 클래스를 주석으로 처리합니다.

  ```
  [auth_provider]

  ; module = cassandra_sigv4.auth

  ; classname = SigV4AuthProvider
  ```

  cqlshrc 파일을 업데이트한 후 다음 명령을 사용하여 서비스별 보안 인증 정보로 Amazon Keyspace에 접속할 수 있습니다.

  ```
  cqlsh-expansion cassandra.us-east-1.amazonaws.com 9142 -u myUserName -
  p myPassword --ssl
  ```

정리

- pip uninstall 명령을 사용하여 cqlsh-expansion 패키지를 제거할 수 있습니다.

  ```
  pip3 uninstall cqlsh-expansion
  ```

  pip3 uninstall 명령을 실행해도 설치 후 스크립트로 생성된 디렉터리 및 관련 파일은 제거되지 않습니다. 설치 후 스크립트로 생성된 폴더와 파일을 제거하려면 .cassandra 디렉터리를 삭제하면 됩니다.

TLS에 대한 **cqlsh** 연결을 수동으로 구성하는 방법

Amazon Keyspaces는 전송 계층 보안(TLS)을 사용한 보안 연결만 허용합니다. 자동으로 인증서를 다운로드하고 사전 구성된 cqlshrc 구성 파일을 설치하는 cqlsh-expansion 유틸리티를 사용할 수 있습니다. 자세한 내용은 이 페이지의 the section called “cqlsh-expansion 사용”를 참조하세요.

인증서를 다운로드하고 연결을 수동으로 구성하려면 다음 단계를 사용하면 됩니다.

1. 다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드하고 sf-class2-root.crt를 로컬 또는 홈 디렉터리에 저장합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> 또한 Amazon 디지털 인증서를 사용하여 Amazon Keyspaces에 접속할 수 있으며, 클라이언트가 Amazon Keyspaces에 성공적으로 접속하면 계속 접속할 수 있습니다. Starfield 인증서는 이전 인증 기관을 사용하는 클라이언트에게 추가적인 이전 버전과의 호환성을 제공합니다.

2. 예를 들어 ${HOME}/.cassandra/cqlshrc와 같은 Cassandra 홈 디렉터리에서 cqlshrc 구성 파일을 열고 다음 줄을 추가합니다.

```
[connection]
port = 9142
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
validate = true
certfile =  path_to_file/sf-class2-root.crt
```

AWS CLI 사용

AWS Command Line Interface(AWS CLI)를 사용하면 명령줄에서 여러 AWS 서비스를 관리하고 스크립트를 통해 자동화할 수 있습니다. Amazon Keyspaces를 통해 AWS CLI를 데이터 언어 정의(DDL) 작업(예: 테이블 생성)에 사용할 수 있습니다. 또한 코드형 인프라(IaC) 서비스 및 AWS CloudFormation, Terraform 같은 도구를 사용할 수 있습니다.

Amazon Keyspaces에서 AWS CLI를 사용하려면 액세스 키 ID와 비밀 액세스 키를 얻어야 합니다. 자세한 내용은 the section called “인증을 위한 IAM 자격 증명 AWS ” 섹션을 참조하세요.

AWS CLI의 Amazon Keyspaces에 사용할 수 있는 모든 명령의 전체 목록을 보려면 AWS CLI 명령 레퍼런스를 참조하십시오.

주제

- AWS CLI 다운로드 및 구성
- Amazon Keyspaces와 함께 AWS CLI 사용

AWS CLI 다운로드 및 구성

AWS CLI는 https://aws.amazon.com/cli에 사용할 수 있습니다. Windows, macOS 또는 Linux에서 실행됩니다. AWS CLI 다운로드 후 다음 단계에 따라 설치 및 구성합니다.

1. AWS Command Line Interface 사용 설명서로 이동합니다.
2. AWS CLI 설치 및 AWS CLI 구성 지침을 따릅니다.

Amazon Keyspaces와 함께 AWS CLI 사용

명령줄 형식은 Amazon Keyspaces 작업 이름과 해당 작업에 대한 파라미터 순으로 구성됩니다. AWS CLI는 파라미터 값의 간편 구문과 JSON을 지원합니다. 다음 Amazon Keyspaces 예제는 AWS CLI 간편 구문을 사용합니다. 자세한 내용은 AWS CLI를 통한 간편 구문 사용을 참조하십시오.

다음 명령은 이름 카탈로그와 함께 키스페이스를 생성합니다.

```
aws keyspaces create-keyspace --keyspace-name 'catalog'
```

이 명령은 출력에 리소스 Amazon 리소스 이름(ARN)을 반환합니다.

```
{
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/"
}
```

다음 명령을 사용하여 키스페이스 카탈로그가 존재하는지 확인할 수 있습니다.

```
aws keyspaces get-keyspace --keyspace-name 'catalog'
```

명령의 출력은 다음 값을 반환합니다.

```
{
    "keyspaceName": "catalog",
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/"
}
```

다음 명령은 book_awards라는 이름의 테이블을 생성합니다. 테이블의 파티션 키는 year 열과 award 열로 구성되며 클러스터링 키는 category 열과 rank 열로 구성되고, 두 클러스터링 열 모두 오름차순 정렬 순서를 사용합니다. (읽기 쉽도록 이 섹션에서는 긴 명령이 여러 줄로 나누어져 있습니다.)

```
aws keyspaces create-table --keyspace-name 'catalog' --table-name 'book_awards'
            --schema-definition 'allColumns=[{name=year,type=int},
{name=award,type=text},{name=rank,type=int},
            {name=category,type=text}, {name=author,type=text},
{name=book_title,type=text},{name=publisher,type=text}],
            partitionKeys=[{name=year},
{name=award}],clusteringKeys=[{name=category,orderBy=ASC},{name=rank,orderBy=ASC}]'
```

이 명령은 다음 출력을 반환합니다.

```
{
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/table/
book_awards"
}
```

다음 명령을 사용하여 테이블의 메타데이터 속성을 확인할 수 있습니다.

```
aws keyspaces get-table --keyspace-name 'catalog' --table-name 'book_awards'
```

이 명령은 다음 출력을 반환합니다.

```
{
    "keyspaceName": "catalog",
    "tableName": "book_awards",
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/catalog/table/
book_awards",
    "creationTimestamp": 1645564368.628,
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "year",
```

```
                "type": "int"
            },
            {
                "name": "award",
                "type": "text"
            },
            {
                "name": "category",
                "type": "text"
            },
            {
                "name": "rank",
                "type": "int"
            },
            {
                "name": "author",
                "type": "text"
            },
            {
                "name": "book_title",
                "type": "text"
            },
            {
                "name": "publisher",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "year"
            },
            {
                "name": "award"
            }
        ],
        "clusteringKeys": [
            {
                "name": "category",
                "orderBy": "ASC"
            },
            {
                "name": "rank",
                "orderBy": "ASC"
            }
```

```
        ],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": 1645564368.628
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    }
}
```

복잡한 스키마가 포함된 테이블을 만들 때는 JSON 파일에서 테이블의 스키마 정의를 로드하는 것이 유용할 수 있습니다. 다음은 그 예시입니다. schema_definition.zip의 스키마 정의 예제 JSON 파일을 다운로드하고 schema_definition.json의 압축을 푼 다음 파일 경로를 기록해 둡니다. 이 예제에서는 스키마 정의 JSON 파일이 현재 디렉터리에 위치합니다. 다양한 파일 경로 옵션에 대해서는 파일에서 매개변수를 로드하는 방법을 참조하세요.

```
aws keyspaces create-table --keyspace-name 'catalog'
            --table-name 'book_awards' --schema-definition 'file://
schema_definition.json'
```

다음 예제는 추가 옵션을 사용하여 myTable이라는 이름의 간단한 테이블을 만드는 방법을 보여줍니다. 참고로 가독성을 높이기 위해 명령이 별도의 행으로 구분되어 있습니다. 이 명령은 테이블을 만드는 방법을 보여줍니다.

- 테이블의 용량 모드 설정
- 테이블 시점 복구 활성화
- 테이블의 기본 Time to Live(TTL) 값을 1년으로 설정
- 테이블에 태그 두 개 추가

```
aws keyspaces create-table --keyspace-name 'catalog' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},{name=name,type=text},
{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --capacity-specification
 'throughputMode=PROVISIONED,readCapacityUnits=5,writeCapacityUnits=5'
            --point-in-time-recovery 'status=ENABLED'
            --default-time-to-live '31536000'
            --tags 'key=env,value=test' 'key=dpt,value=sec'
```

이 예제에서는 암호화에 고객 관리형 키를 사용하고 열과 행의 만료 날짜를 설정할 수 있도록 TTL을 활성화한 새 테이블을 생성하는 방법을 보여줍니다. 이 샘플을 실행하려면 고객 관리형 AWS KMS 키의 리소스 ARN을 자체 키로 바꾸고 Amazon Keyspaces가 해당 키에 액세스할 수 있도록 해야 합니다.

```
aws keyspaces create-table --keyspace-name 'catalog' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},{name=name,type=text},
{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --encryption-specification
 'type=CUSTOMER_MANAGED_KMS_KEY,kmsKeyIdentifier=arn:aws:kms:us-
east-1:111222333444:key/11111111-2222-3333-4444-555555555555'
            --ttl 'status=ENABLED'
```

API 사용

AWS SDK 및 AWS Command Line Interface(AWS CLI)를 사용하여 Amazon Keyspaces와 대화형으로 작업할 수 있습니다. API를 데이터 언어 정의(DDL) 작업(예: 키스페이스 또는 테이블 생성)에 사용할 수 있습니다. 또한 코드형 인프라(IaC) 서비스 및 AWS CloudFormation, Terraform 같은 도구를 사용할 수 있습니다.

Amazon Keyspaces에서 AWS CLI를 사용하려면 액세스 키 ID와 비밀 액세스 키를 얻어야 합니다. 자세한 내용은 the section called “인증을 위한 IAM 자격 증명 AWS ” 섹션을 참조하세요.

API에서 Amazon Keyspaces에 사용할 수 있는 모든 작업의 전체 목록은 Amazon Keyspaces API 참조에서 확인할 수 있습니다.

아마존 키스페이스를 SDK와 함께 사용하기 AWS

AWS 소프트웨어 개발 키트 (SDK) 는 널리 사용되는 여러 프로그래밍 언어에 사용할 수 있습니다. 각 SDK는 개발자가 선호하는 언어로 애플리케이션을 쉽게 구축할 수 있도록 하는 API, 코드 예시 및 설명서를 제공합니다.

SDK 설명서	코드 예시
AWS SDK for C++	AWS SDK for C++ 코드 예제
AWS CLI	AWS CLI 코드 예제
AWS SDK for Go	AWS SDK for Go 코드 예제
AWS SDK for Java	AWS SDK for Java 코드 예제
AWS SDK for JavaScript	AWS SDK for JavaScript 코드 예제
AWS SDK for Kotlin	AWS SDK for Kotlin 코드 예제
AWS SDK for .NET	AWS SDK for .NET 코드 예제
AWS SDK for PHP	AWS SDK for PHP 코드 예제
AWS Tools for PowerShell	PowerShell 코드 예제를 위한 도구
AWS SDK for Python (Boto3)	AWS SDK for Python (Boto3) 코드 예제
AWS SDK for Ruby	AWS SDK for Ruby 코드 예제
AWS SDK for Rust	AWS SDK for Rust 코드 예제
AWS SDK for SAP ABAP	AWS SDK for SAP ABAP 코드 예제
AWS SDK for Swift	AWS SDK for Swift 코드 예제

가용성 예제

필요한 예제를 찾을 수 없습니까? 이 페이지 하단의 피드백 제공 링크를 사용하여 코드 예시를 요청하세요.

Cassandra 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스

많은 타사 오픈 소스 Cassandra 드라이버를 사용하여 Amazon Keyspaces에 접속할 수 있습니다. Amazon Keyspaces는 Apache Cassandra 버전 3.11.2를 지원하는 Cassandra 드라이버와 호환됩니다. 다음은 Amazon Keyspaces와 함께 사용하도록 테스트하고 권장한 드라이버 및 최신 버전입니다.

- `Java v3.3`
- `Java v4.17`
- `Python Cassandra-driver 3.29.1`
- `Node.js cassandra driver -v 4.7.2`
- `GO using GOCQL v1.6`
- `.NET CassandraCSharpDriver -v 3.20.1`

Cassandra 드라이버에 대한 자세한 내용은 Apache Cassandra Client 드라이버를 참조하세요.

> **Note**
>
> 시작하는 데 도움이 되도록 인기 있는 드라이버를 사용하여 Amazon Keyspace에 대한 연결을 설정하는 end-to-end 코드 예제를 보고 다운로드할 수 있습니다. 에서 Amazon Keyspaces 예제를 참조하십시오. GitHub

이 장의 튜토리얼에는 Amazon Keyspace에 대한 연결이 성공적으로 설정되었는지 확인하는 간단한 CQL 쿼리가 포함되어 있습니다. Amazon Keyspaces 엔드포인트에 연결한 후 키스페이스 및 테이블을 사용하는 방법을 알아보려면 CQL 언어 참조을 참조하세요. Amazon VPC 엔드포인트에서 Amazon Keyspace에 연결하는 방법을 보여주는 step-by-step 자습서는 을 참조하십시오. the section called “VPC 엔드포인트와 연결”

주제

- Cassandra. NET Core 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스
- Cassandra Go 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스
- Cassandra Perl 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스

Cassandra Java 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스

이 섹션에서는 Java 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 접속하는 방법을 소개합니다.

> **Note**
>
> 자바 17 및 DataStax 자바 드라이버 4.17은 현재 베타 지원만 가능합니다. 자세한 정보는 https://docs.datastax.com/en/developer/java-driver/4.17/upgrade_guide/을 참조하세요.

Amazon Keyspaces 리소스에 프로그래밍 방식으로 액세스할 수 있는 보안 인증 정보를 사용자와 애플리케이션에 제공하려면 다음 중 하나를 수행할 수 있습니다.

- 특정 AWS Identity and Access Management (IAM) 사용자와 연결된 서비스별 보안 인증 정보를 생성합니다.
- 보안 강화를 위해 모든 서비스에서 사용되는 IAM ID용 IAM 액세스 키를 생성하는 것이 좋습니다. AWS Cassandra 클라이언트 드라이버용 Amazon Keyspaces SigV4 인증 플러그인을 사용하면 사용자 이름 및 암호 대신 IAM 액세스 키를 사용하여 Amazon Keyspaces에 대한 호출을 인증할 수 있습니다. 자세한 정보는 the section called “인증을 위한 IAM 자격 증명 AWS ”을 참조하세요.

> **Note**
>
> Spring Boot와 함께 Amazon Keyspaces를 사용하는 방법에 대한 예는 https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/spring을 참조하세요.

주제

- 시작하기 전 준비 사항
- 서비스별 자격 증명을 사용하여 Apache Cassandra용 DataStax Java 드라이버를 사용하여 Amazon Keyspaces에 연결하는 방법에 대한 tep-by-step 자습서
- 아파치 카산드라용 4.x DataStax 자바 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결하는 방법에 대한 tep-by-step 자습서
- 아파치 카산드라용 3.x DataStax 자바 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결합니다.

시작하기 전 준비 사항

Amazon Keyspaces에 접속하려면 시작하기 전에 다음 작업을 수행해야 합니다.

1. Amazon Keyspaces에서는 클라이언트와의 연결을 보호하는 데 도움이 되는 전송 계층 보안(TLS)을 사용해야 합니다.

   a. 다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드하고 `sf-class2-root.crt`를 로컬 또는 홈 디렉터리에 저장합니다.

   ```
   curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
   ```

   > **Note**
   >
   > 또한 Amazon 디지털 인증서를 사용하여 Amazon Keyspaces에 접속할 수 있으며, 클라이언트가 Amazon Keyspaces에 성공적으로 접속하면 계속 접속할 수 있습니다. Starfield 인증서는 이전 인증 기관을 사용하는 클라이언트에게 추가적인 이전 버전과의 호환성을 제공합니다.

   b. Starfield 디지털 인증서를 trustStore 파일로 변환합니다.

   ```
   openssl x509 -outform der -in sf-class2-root.crt -out temp_file.der
   keytool -import -alias cassandra -keystore cassandra_truststore.jks -file
    temp_file.der
   ```

   이 단계에서는 키스토어의 암호를 생성하고 이 인증서를 신뢰해야 합니다. 대화형 명령은 다음과 같습니다.

   ```
   Enter keystore password:
   Re-enter new password:
   ```

```
Owner: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
 Inc.", C=US
Issuer: OU=Starfield Class 2 Certification Authority, O="Starfield
 Technologies, Inc.", C=US
Serial number: 0
Valid from: Tue Jun 29 17:39:16 UTC 2004 until: Thu Jun 29 17:39:16 UTC 2034
Certificate fingerprints:
  MD5:  32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
  SHA256:
 14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB
Signature algorithm name: SHA1withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86   F4 5B 55 AC DC D7 10 C2  ._.......[U.....
0010: 0E A9 88 E7                                        ....
]
[OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
 Inc.", C=US]
SerialNumber: [    00]
]
#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:2147483647
]
#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86   F4 5B 55 AC DC D7 10 C2  ._.......[U.....
0010: 0E A9 88 E7                                        ....
]
]
Trust this certificate? [no]:  y
```

2. JVM 인수에 trustStore 파일을 첨부합니다.

```
-Djavax.net.ssl.trustStore=path_to_file/cassandra_truststore.jks
-Djavax.net.ssl.trustStorePassword=my_password
```

서비스별 자격 증명을 사용하여 Apache Cassandra용 DataStax Java 드라이버를 사용하여 Amazon Keyspaces에 연결하는 방법에 대한 tep-by-step 자습서

다음 step-by-step 자습서는 서비스별 자격 증명을 사용하여 Cassandra용 Java 드라이버를 사용하여 Amazon Keyspace에 연결하는 방법을 안내합니다. 구체적으로 말하자면, 아파치 카산드라용 자바 드라이버 4.0 버전을 사용하게 될 것입니다. DataStax

주제

1단계: 사전 조건

이 자습서를 따르려면 서비스별 자격 증명을 생성하고 Apache Cassandra용 DataStax Java 드라이버를 Java 프로젝트에 추가해야 합니다.

- the section called “서비스별 보안 인증 정보”의 단계를 완료하여 Amazon Keyspaces IAM 사용자를 위한 서비스별 보안 인증 정보를 생성합니다. IAM 액세스 키를 인증에 사용하려면 the section called “Java 4.x용 인증 플러그인”을 참조하세요.
- 아파치 카산드라용 DataStax 자바 드라이버를 자바 프로젝트에 추가합니다. Apache Cassandra 3.11.2를 지원하는 드라이버 버전을 사용하고 있는지 확인하세요. 자세한 내용은 Apache Cassandra용 DataStax Java 드라이버 설명서를 참조하십시오.

2단계: 드라이버 구성

애플리케이션의 구성 파일을 생성하여 DataStax Java Cassandra 드라이버의 설정을 지정할 수 있습니다. 이 구성 파일은 기본 설정을 재정의하고 포트 9142를 사용하여 Amazon Keyspaces 서비스 엔드포인트에 접속하도록 드라이버에 지시합니다. 사용 가능한 서비스 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

구성 파일을 생성하고 애플리케이션의 리소스 폴더에 파일을 저장합니다(예: `src/main/resources/application.conf`). `application.conf`을 열고 다음 구성 설정을 열고 추가합니다.

1. 인증 제공자 - `PlainTextAuthProvider` 클래스를 사용하여 인증 제공자를 생성합니다. *`ServiceUser###`* 의 단계에 따라 서비스별 자격 증명을 생성할 때 얻은 사용자 이름 및 암호와 *`ServicePassword`*일치해야 합니다. 서비스별 보안 인증 정보 생성

> **Note**
>
> 드라이버 구성 파일에 자격 증명을 하드코딩하는 대신 Apache Cassandra용 DataStax Java 드라이버용 인증 플러그인을 사용하여 단기 자격 증명을 사용할 수 있습니다. 자세히 알아보려면 the section called “Java 4.x용 인증 플러그인”에 대한 안내를 따르세요.

2. 로컬 데이터 센터 - `local-datacenter`에 대한 값을 연결 대상 리전으로 설정합니다. 예를 들어 애플리케이션이 `cassandra.us-east-2.amazonaws.com`에 연결 중인 경우 로컬 데이터 센터를 `us-east-2`로 설정합니다. 사용 가능한 모든 AWS 리전에 대한 자세한 내용은 ??? 섹션을 참조하세요. 더 적은 수의 노드에 대해 로드 밸런싱을 수행하도록 `slow-replica-avoidance = false`을 설정합니다.
3. SSL/TLS — 클래스를 지정하는 한 줄로 구성 파일에 섹션을 EngineFactory 추가하여 SSL을 초기화합니다. `class = DefaultSslEngineFactory` trustStore 파일의 경로와 이전에 만든 암호를 입력합니다. Amazon Keyspaces는 피어의 `hostname-validation`를 지원하지 않으므로 이 옵션을 false로 설정합니다.

```
datastax-java-driver {

    basic.contact-points = [ "cassandra.us-east-2.amazonaws.com:9142"]
    advanced.auth-provider{
        class = PlainTextAuthProvider
        username = "ServiceUserName"
        password = "ServicePassword"
    }
    basic.load-balancing-policy {
        local-datacenter = "us-east-2"
        slow-replica-avoidance = false
    }

    advanced.ssl-engine-factory {
        class = DefaultSslEngineFactory
        truststore-path = "./src/main/resources/cassandra_truststore.jks"
        truststore-password = "my_password"
        hostname-validation = false
      }
}
```

> **Note**
>
> 구성 파일에 trustStore 경로를 추가하는 대신 애플리케이션 코드에 trustStore 경로를 직접 추가하거나 trustStore 경로를 JVM 인수에 추가할 수도 있습니다.

3단계: 샘플 애플리케이션 실행

이 코드 예제는 앞서 만든 구성 파일을 사용하여 Amazon Keyspaces에 대한 연결 풀을 생성하는 간단한 명령줄 애플리케이션을 보여줍니다. 간단한 쿼리를 실행하여 연결이 설정되었는지 확인합니다.

```
package <your package>;
// add the following imports to your project
import com.datastax.oss.driver.api.core.CqlSession;
import com.datastax.oss.driver.api.core.config.DriverConfigLoader;
import com.datastax.oss.driver.api.core.cql.ResultSet;
import com.datastax.oss.driver.api.core.cql.Row;

public class App
{

    public static void main( String[] args )
    {
        //Use DriverConfigLoader to load your configuration file
        DriverConfigLoader loader =
 DriverConfigLoader.fromClasspath("application.conf");
        try (CqlSession session = CqlSession.builder()
                .withConfigLoader(loader)
                .build()) {

            ResultSet rs = session.execute("select * from system_schema.keyspaces");
            Row row = rs.one();
            System.out.println(row.getString("keyspace_name"));
        }
    }
}
```

> **Note**
>
> `try` 블록을 사용하여 연결이 항상 닫히도록 설정합니다. `try` 블록을 사용하지 않는 경우 리소스 유출을 방지하기 위해 연결을 닫아야 합니다.

아파치 카산드라용 4.x DataStax 자바 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결하는 방법에 대한 tep-by-step 자습서

다음 섹션에서는 Apache Cassandra용 오픈 소스 4.x DataStax Java 드라이버용 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces (Apache Cassandra용) 에 액세스하는 방법을 설명합니다. 플러그인은 리포지토리에서 GitHub사용할 수 있습니다.

SigV4 인증 플러그인을 사용하면 Amazon Keyspaces에 접속할 때 사용자 또는 역할에 대한 IAM 보안 인증 정보를 사용할 수 있습니다. 이 플러그인은 사용자 이름과 비밀번호를 요구하는 대신 액세스 키를 사용하여 API 요청에 서명합니다. 자세한 정보는 the section called “인증을 위한 IAM 자격 증명 AWS ”을 참조하세요.

1단계: 사전 조건

이 튜토리얼을 따르려면 다음 작업을 수행해야 합니다.

- 아직 수행하지 않은 경우 the section called “인증을 위한 IAM 자격 증명 AWS ”의 단계에 따라 IAM 사용자 또는 역할의 보안 인증 정보를 생성합니다. 이 튜토리얼에서는 액세스 키가 환경 변수로 저장된다고 가정합니다. 자세한 정보는 the section called “액세스 키를 관리하는 방법”을 참조하세요.
- 아파치 카산드라용 DataStax 자바 드라이버를 자바 프로젝트에 추가하세요. Apache Cassandra 3.11.2를 지원하는 드라이버 버전을 사용하고 있는지 확인하세요. 자세한 내용은 Apache Cassandra용 DataStax 자바 드라이버 설명서를 참조하십시오.
- 애플리케이션에 인증 플러그인을 추가합니다. 인증 플러그인은 아파치 카산드라용 DataStax Java 드라이버 버전 4.x를 지원합니다. Apache Maven이나 Maven 종속성을 사용할 수 있는 빌드 시스템을 사용하는 경우 `pom.xml` 파일에 다음 종속성을 추가합니다.

> **Important**
>
> 저장소에 표시된 대로 플러그인 버전을 최신 버전으로 교체하십시오. GitHub

```
<dependency>
        <groupId>software.aws.mcs</groupId>
        <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin</artifactId>
        <version>4.0.9</version>
</dependency>
```

2단계: 드라이버 구성

애플리케이션에 대한 구성 파일을 생성하여 DataStax Java Cassandra 드라이버의 설정을 지정할 수 있습니다. 이 구성 파일은 기본 설정을 재정의하고 포트 9142를 사용하여 Amazon Keyspaces 서비스 엔드포인트에 접속하도록 드라이버에 지시합니다. 사용 가능한 서비스 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

구성 파일을 생성하고 애플리케이션의 리소스 폴더에 파일을 저장합니다(예: `src/main/resources/application.conf`). `application.conf`을 열고 다음 구성 설정을 열고 추가합니다.

1. 인증 제공자 - `advanced.auth-provider.class`를 `software.aws.mcs.auth.SigV4AuthProvider`의 새 인스턴스로 설정합니다. AuthProvider SigV4는 SigV4 인증을 수행하기 위해 플러그인에서 제공하는 인증 핸들러입니다.
2. 로컬 데이터 센터 - `local-datacenter`에 대한 값을 연결 대상 리전으로 설정합니다. 예를 들어 애플리케이션이 `cassandra.us-east-2.amazonaws.com`에 연결 중인 경우 로컬 데이터 센터를 `us-east-2`로 설정합니다. 사용 가능한 모든 정보는 을 참조하십시오. AWS 리전??? 사용 가능한 모든 노드에 대해 로드 밸런싱을 `slow-replica-avoidance = false` 수행하도록 설정합니다.
3. Idempotence - 실패한 `idempotence` 읽기/쓰기/준비/실행 요청을 항상 재시도하도록 `true` 드라이버를 구성하도록 응용 프로그램의 기본값을 설정합니다. 이는 실패한 요청을 재시도하여 일시적인 오류를 처리하는 데 도움이 되는 분산 응용 프로그램을 위한 모범 사례입니다.
4. SSL/TLS — 클래스를 지정하는 한 줄로 구성 파일에 섹션을 EngineFactory 추가하여 SSL을 초기화합니다. `class = DefaultSslEngineFactory` trustStore 파일의 경로와 이전에 만든 암호를 입력합니다. Amazon Keyspaces는 피어의 `hostname-validation`를 지원하지 않으므로 이 옵션을 false로 설정합니다.
5. 연결 - 설정을 통해 엔드포인트당 최소 3개의 로컬 연결을 생성합니다. `local.size = 3` 이는 애플리케이션이 오버헤드 및 트래픽 버스트를 처리하는 데 도움이 되는 모범 사례입니다. 예상 트래픽 패턴을 기반으로 애플리케이션에 필요한 엔드포인트당 로컬 연결 수를 계산하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “연결 구성 방법”
6. 재시도 정책 — Amazon Keyspaces 재시도 `AmazonKeyspacesExponentialRetryPolicy` 정책은 Cassandra 드라이버와 함께 제공되는 정책 대신 사용할 수 `DefaultRetryPolicy` 있습니다. 두 재시도 정책의 주요 차이점은 필요에 맞게 재시도 횟수를 구성할 수 있다는 것입니다. `AmazonKeyspacesExponentialRetryPolicy` 기본적으로 에 대한 재시도 횟수는 3으로 `AmazonKeyspacesExponentialRetryPolicy` 설정됩니다. 또한 Amazon Keyspaces 재시도 정책은 제네릭을 반환하지 않습니다. `NoHostAvailableException` 대신 Amazon Keyspaces

재시도 정책은 서비스에서 반환한 원래 예외를 다시 전달합니다. 재시도 정책을 구현하는 추가 코드 예제는 Github의 Amazon Keyspaces 재시도 정책을 참조하십시오.

7. 준비된 명령문 — 네트워크 사용을 `prepare-on-all-nodes` 최적화하려면 false로 설정하십시오.

```
datastax-java-driver {
    basic {
        contact-points = [ "cassandra.us-east-2.amazonaws.com:9142"]
        request {
            timeout = 2 seconds
            consistency = LOCAL_QUORUM
            page-size = 1024
            default-idempotence = true
        }
        load-balancing-policy {
            local-datacenter = "us-east-2"
            class = DefaultLoadBalancingPolicy
            slow-replica-avoidance = false
        }
    }
    advanced {
        auth-provider {
            class = software.aws.mcs.auth.SigV4AuthProvider
            aws-region = us-east-2
        }
        ssl-engine-factory {
            class = DefaultSslEngineFactory
            truststore-path = "./src/main/resources/cassandra_truststore.jks"
            truststore-password = "my_password"
            hostname-validation = false
        }
        connection {
      connect-timeout = 5 seconds
      max-requests-per-connection = 512
      pool {
                local.size = 3
      }
        }
       retry-policy {
           class =  com.aws.ssa.keyspaces.retry.AmazonKeyspacesExponentialRetryPolicy
     max-attempts = 3
     min-wait = 10 mills
```

```
     max-wait = 100 mills
       }
       prepared-statements {
     prepare-on-all-nodes = false
       }
    }
}
```

> **Note**
>
> 구성 파일에 trustStore 경로를 추가하는 대신 애플리케이션 코드에 trustStore 경로를 직접 추가하거나 trustStore 경로를 JVM 인수에 추가할 수도 있습니다.

3단계: 애플리케이션 실행

이 코드 예제는 앞서 만든 구성 파일을 사용하여 Amazon Keyspaces에 대한 연결 풀을 생성하는 간단한 명령줄 애플리케이션을 보여줍니다. 간단한 쿼리를 실행하여 연결이 설정되었는지 확인합니다.

```
package <your package>;
// add the following imports to your project
import com.datastax.oss.driver.api.core.CqlSession;
import com.datastax.oss.driver.api.core.config.DriverConfigLoader;
import com.datastax.oss.driver.api.core.cql.ResultSet;
import com.datastax.oss.driver.api.core.cql.Row;

public class App
{

    public static void main( String[] args )
    {
        //Use DriverConfigLoader to load your configuration file
        DriverConfigLoader loader =
 DriverConfigLoader.fromClasspath("application.conf");
        try (CqlSession session = CqlSession.builder()
                .withConfigLoader(loader)
                .build()) {

            ResultSet rs = session.execute("select * from system_schema.keyspaces");
            Row row = rs.one();
            System.out.println(row.getString("keyspace_name"));
        }
```

```
    }
}
```

> **Note**
>
> try 블록을 사용하여 연결이 항상 닫히도록 설정합니다. try 블록을 사용하지 않는 경우 리소스 유출을 방지하기 위해 연결을 닫아야 합니다.

아파치 카산드라용 3.x DataStax 자바 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결합니다.

다음 섹션에서는 Apache Cassandra용 3.x 오픈 소스 DataStax Java 드라이버용 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 액세스하는 방법을 설명합니다. 플러그인은 리포지토리에서 GitHub 사용할 수 있습니다.

SigV4 인증 플러그인을 사용하면 Amazon Keyspaces에 접속할 때 사용자 및 역할에 대한 IAM 보안 인증 정보를 사용할 수 있습니다. 이 플러그인은 사용자 이름과 비밀번호를 요구하는 대신 액세스 키를 사용하여 API 요청에 서명합니다. 자세한 정보는 the section called “인증을 위한 IAM 자격 증명 AWS ”을 참조하세요.

1단계: 사전 조건

이 코드 샘플을 실행하려면 먼저 다음 작업을 수행해야 합니다.

- the section called “인증을 위한 IAM 자격 증명 AWS ”의 단계에 따라 IAM 사용자 또는 역할의 보안 인증 정보를 생성합니다. 이 튜토리얼에서는 액세스 키가 환경 변수로 저장된다고 가정합니다. 자세한 정보는 the section called “액세스 키를 관리하는 방법”을 참조하세요.
- the section called “시작하기 전 준비 사항”의 단계에 따라 Starfield 디지털 인증서를 다운로드하고, trustStore 파일로 변환하고, JVM 인수의 trustStore 파일을 애플리케이션에 연결합니다.
- 아파치 카산드라용 DataStax 자바 드라이버를 자바 프로젝트에 추가하세요. Apache Cassandra 3.11.2를 지원하는 드라이버 버전을 사용하고 있는지 확인하세요. 자세한 내용은 Apache Cassandra용 DataStax 자바 드라이버 설명서를 참조하십시오.
- 애플리케이션에 인증 플러그인을 추가합니다. 인증 플러그인은 아파치 카산드라용 DataStax Java 드라이버 버전 3.x를 지원합니다. Apache Maven이나 Maven 종속성을 사용할 수 있는 빌드 시스템을 사용하는 경우 pom.xml 파일에 다음 종속성을 추가합니다. 저장소에 표시된 대로 플러그인 버전을 최신 버전으로 교체하십시오. GitHub

```
<dependency>
```

```
        <groupId>software.aws.mcs</groupId>
        <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin_3</artifactId>
        <version>3.0.3</version>
</dependency>
```

2단계: 애플리케이션 실행

이 코드 예제는 Amazon Keyspaces에 대한 연결 풀을 생성하는 간단한 명령줄 애플리케이션을 보여줍니다. 간단한 쿼리를 실행하여 연결이 설정되었는지 확인합니다.

```
package <your package>;
// add the following imports to your project

import software.aws.mcs.auth.SigV4AuthProvider;
import com.datastax.driver.core.Cluster;
import com.datastax.driver.core.ResultSet;
import com.datastax.driver.core.Row;
import com.datastax.driver.core.Session;

public class App
{

    public static void main( String[] args )
    {
        String endPoint = "cassandra.us-east-2.amazonaws.com";
        int portNumber = 9142;
        Session session = Cluster.builder()
                                 .addContactPoint(endPoint)
                                 .withPort(portNumber)
                                 .withAuthProvider(new SigV4AuthProvider("us-east-2"))

                                 .withSSL()
                                 .build()
                                 .connect();

        ResultSet rs = session.execute("select * from system_schema.keyspaces");
        Row row = rs.one();
        System.out.println(row.getString("keyspace_name"));
    }
}
```

사용 노트:

사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

Amazon Keyspace와 함께 Java Driver를 사용할 때 유용한 Java 드라이버 정책, 예제, 모범 사례는 https://github.com/aws-samples/amazon-keyspaces-java-driver-helpers 리포지토리를 참조하십시오.

Cassandra Python 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스

이 섹션에서는 Python 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 접속하는 방법을 소개합니다. Amazon Keyspaces 리소스에 프로그래밍 방식으로 액세스할 수 있는 보안 인증 정보를 사용자와 애플리케이션에 제공하려면 다음 중 하나를 수행할 수 있습니다.

- 특정 AWS Identity and Access Management (IAM) 사용자와 연결된 서비스별 보안 인증 정보를 생성합니다.
- 보안 강화를 위해 모든 AWS 서비스에서 사용되는 IAM 사용자 또는 역할을 위한 IAM 액세스 키를 생성하는 것이 좋습니다. Cassandra 클라이언트 드라이버용 Amazon Keyspaces SigV4 인증 플러그인을 사용하면 사용자 이름 및 암호 대신 IAM 액세스 키를 사용하여 Amazon Keyspaces에 대한 호출을 인증할 수 있습니다. 자세한 정보는 the section called “인증을 위한 IAM 자격 증명 AWS ”을 참조하세요.

주제

- 시작하기 전 준비 사항
- Apache Cassandra용 Python 드라이버와 서비스별 보안 인증 정보를 사용하여 Amazon Keyspaces에 접속합니다.
- 아파치 카산드라용 DataStax Python 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결합니다.

시작하기 전 준비 사항

시작하기 전에 다음 작업을 수행해야 합니다.

Amazon Keyspaces에서는 클라이언트와의 연결을 보호하는 데 도움이 되는 전송 계층 보안(TLS)을 사용해야 합니다. TLS를 사용하여 Amazon Keyspace에 접속하려면 Amazon 디지털 인증서를 다운로드하고 TLS를 사용하도록 Python 드라이버를 구성해야 합니다.

다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드하고 `sf-class2-root.crt`를 로컬 또는 홈 디렉터리에 저장합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> 또한 Amazon 디지털 인증서를 사용하여 Amazon Keyspaces에 접속할 수 있으며, 클라이언트가 Amazon Keyspaces에 성공적으로 접속하면 계속 접속할 수 있습니다. Starfield 인증서는 이전 인증 기관을 사용하는 클라이언트에게 추가적인 이전 버전과의 호환성을 제공합니다.


```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```


Apache Cassandra용 Python 드라이버와 서비스별 보안 인증 정보를 사용하여 Amazon Keyspaces에 접속합니다.

다음 코드 예제는 Python 클라이언트 드라이버 및 서비스별 자격 증명을 사용하여 Amazon Keyspaces에 접속하는 방법을 보여줍니다.

```
from cassandra.cluster import Cluster
from ssl import SSLContext, PROTOCOL_TLSv1_2 , CERT_REQUIRED
from cassandra.auth import PlainTextAuthProvider

ssl_context = SSLContext(PROTOCOL_TLSv1_2 )
ssl_context.load_verify_locations('path_to_file/sf-class2-root.crt')
ssl_context.verify_mode = CERT_REQUIRED
auth_provider = PlainTextAuthProvider(username='ServiceUserName',
 password='ServicePassword')
cluster = Cluster(['cassandra.us-east-2.amazonaws.com'], ssl_context=ssl_context,
 auth_provider=auth_provider, port=9142)
session = cluster.connect()
r = session.execute('select * from system_schema.keyspaces')
print(r.current_rows)
```

사용 노트:

1. "*path_to_file*/sf-class2-root.crt"을 첫 번째 단계에서 저장한 인증서 경로로 바꿉니다.
2. 의 단계에 따라 *ServiceUserName* and 가 서비스별 자격 증명을 생성할 때 얻은 사용자 이름 및 암호와 *ServicePassword*일치하는지 확인하십시오. 서비스별 보안 인증 정보 생성
3. 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

아파치 카산드라용 DataStax Python 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결합니다.

다음 섹션에서는 Apache Cassandra용 오픈 소스 DataStax Python 드라이버용 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces (Apache Cassandra용) 에 액세스하는 방법을 보여줍니다.

아직 수행하지 않은 경우 the section called “인증을 위한 IAM 자격 증명 AWS ”의 단계에 따라 IAM 역할의 보안 인증 정보를 생성합니다. 이 튜토리얼에서는 IAM 역할이 필요한 임시 보안 인증 정보를 사용합니다. 임시 보안 인증 정보에 대한 자세한 내용은 the section called “임시 보안 인증 정보를 사용하여 Amazon Keyspaces에 접속” 섹션을 참조하세요.

그런 다음 리포지토리에서 Python SigV4 인증 플러그인을 환경에 추가합니다. GitHub

```
pip install cassandra-sigv4
```

다음 코드 예제는 Cassandra용 오픈 소스 DataStax Python 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결하는 방법을 보여줍니다. 플러그인은 파이썬용 AWS SDK (Boto3) 에 따라 다릅니다. `boto3.session`을 통해 임시 보안 인증 정보가 생성됩니다.

```
from cassandra.cluster import Cluster
from ssl import SSLContext, PROTOCOL_TLSv1_2 , CERT_REQUIRED
from cassandra.auth import PlainTextAuthProvider
import boto3
from cassandra_sigv4.auth import SigV4AuthProvider

ssl_context = SSLContext(PROTOCOL_TLSv1_2)
ssl_context.load_verify_locations('path_to_file/sf-class2-root.crt')
ssl_context.verify_mode = CERT_REQUIRED

# use this if you want to use Boto to set the session parameters.
boto_session = boto3.Session(aws_access_key_id="AKIAIOSFODNN7EXAMPLE",
                             aws_secret_access_key="wJalrXUtnFEMI/K7MDENG/
bPxRfiCYEXAMPLEKEY",
                             aws_session_token="AQoDYXdzEJr...<remainder of token>",
                             region_name="us-east-2")
auth_provider = SigV4AuthProvider(boto_session)

# Use this instead of the above line if you want to use the Default Credentials and not
 bother with a session.
# auth_provider = SigV4AuthProvider()
```

```
cluster = Cluster(['cassandra.us-east-2.amazonaws.com'], ssl_context=ssl_context,
 auth_provider=auth_provider,
                   port=9142)
session = cluster.connect()
r = session.execute('select * from system_schema.keyspaces')
print(r.current_rows)
```

사용 노트:

1. "*path_to_file*/sf-class2-root.crt"을 첫 번째 단계에서 저장한 인증서 경로로 바꿉니다.
2. *aws_access_key_id*, *aws_secret_access_key*, *aws_session_token*이 `boto3.session`를 사용하여 얻은 `Access Key`, `Secret Access Key`, `Session Token`와 일치하는지 확인합니다. 자세한 내용은 AWS SDK for Python (Boto3)의 보안 인증 정보를 참조하세요.
3. 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

Cassandra Node.js 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스

이 섹션에서는 Node.js 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 접속하는 방법을 소개합니다. Amazon Keyspaces 리소스에 프로그래밍 방식으로 액세스할 수 있는 보안 인증 정보를 사용자와 애플리케이션에 제공하려면 다음 중 하나를 수행할 수 있습니다.

- 특정 AWS Identity and Access Management (IAM) 사용자와 연결된 서비스별 보안 인증 정보를 생성합니다.
- 보안 강화를 위해 모든 서비스에서 사용되는 IAM 사용자 또는 역할을 위한 IAM 액세스 키를 생성하는 것이 좋습니다. AWS Cassandra 클라이언트 드라이버용 Amazon Keyspaces SigV4 인증 플러그인을 사용하면 사용자 이름 및 암호 대신 IAM 액세스 키를 사용하여 Amazon Keyspaces에 대한 호출을 인증할 수 있습니다. 자세한 정보는 the section called “인증을 위한 IAM 자격 증명 AWS ”을 참조하세요.

주제

- 시작하기 전 준비 사항
- 아파치 카산드라용 Node.js DataStax 드라이버와 서비스별 자격 증명을 사용하여 Amazon Keyspaces에 연결합니다.
- 아파치 카산드라용 DataStax Node.js 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결합니다.

시작하기 전 준비 사항

시작하기 전에 다음 작업을 수행해야 합니다.

Amazon Keyspaces에서는 클라이언트와의 연결을 보호하는 데 도움이 되는 전송 계층 보안(TLS)을 사용해야 합니다. TLS를 사용하여 Amazon Keyspace에 접속하려면 Amazon 디지털 인증서를 다운로드하고 TLS를 사용하도록 Python 드라이버를 구성해야 합니다.

다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드하고 `sf-class2-root.crt`를 로컬 또는 홈 디렉터리에 저장합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> 또한 Amazon 디지털 인증서를 사용하여 Amazon Keyspaces에 접속할 수 있으며, 클라이언트가 Amazon Keyspaces에 성공적으로 접속하면 계속 접속할 수 있습니다. Starfield 인증서는 이전 인증 기관을 사용하는 클라이언트에게 추가적인 이전 버전과의 호환성을 제공합니다.


```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```


아파치 카산드라용 Node.js DataStax 드라이버와 서비스별 자격 증명을 사용하여 Amazon Keyspaces에 연결합니다.

TLS용 Starfield 디지털 인증서를 사용하고 서비스별 보안 인증 정보를 사용하여 인증하도록 드라이버를 구성합니다. 예:

```
const cassandra = require('cassandra-driver');
const fs = require('fs');
const auth = new cassandra.auth.PlainTextAuthProvider('ServiceUserName',
 'ServicePassword');
const sslOptions1 = {
         ca: [
                    fs.readFileSync('path_to_file/sf-class2-root.crt', 'utf-8')],
                    host: 'cassandra.us-west-2.amazonaws.com',
                    rejectUnauthorized: true
        };
const client = new cassandra.Client({
```

```
                    contactPoints: ['cassandra.us-west-2.amazonaws.com'],
                    localDataCenter: 'us-west-2',
                    authProvider: auth,
                    sslOptions: sslOptions1,
                    protocolOptions: { port: 9142 }
        });
const query = 'SELECT * FROM system_schema.keyspaces';

client.execute(query)
                     .then( result => console.log('Row from Keyspaces %s',
 result.rows[0]))
                     .catch( e=> console.log(`${e}`));
```

사용 노트:

1. "*path_to_file*/sf-class2-root.crt"을 첫 번째 단계에서 저장한 인증서 경로로 바꿉니다.
2. 의 단계에 따라 *ServiceUser### ###* 서비스별 자격 증명을 생성할 때 얻은 사용자 이름 및 암호와 *ServicePassword*일치하는지 확인하십시오. 서비스별 보안 인증 정보 생성
3. 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

아파치 카산드라용 DataStax Node.js 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 연결합니다.

다음 섹션에서는 Apache Cassandra용 오픈 소스 DataStax Node.js 드라이버용 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces (Apache Cassandra용) 에 액세스하는 방법을 보여줍니다.

아직 수행하지 않은 경우 the section called “인증을 위한 IAM 자격 증명 AWS ”의 단계에 따라 IAM 사용자 또는 역할의 보안 인증 정보를 생성합니다.

리포지토리에서 Node.js SiGV4 인증 플러그인을 애플리케이션에 추가합니다. GitHub 플러그인은 카산드라용 DataStax Node.js 드라이버 버전 4.x를 지원하며 Node.js SDK에 따라 다릅니다. AWS `AWSCredentialsProvider`을 통해 보안 인증 정보가 생성됩니다.

```
$ npm install aws-sigv4-auth-cassandra-plugin --save
```

이 코드 예제는 `SigV4AuthProvider`의 리전별 인스턴스를 인증 공급자로 설정하는 방법을 보여줍니다.

```
const cassandra = require('cassandra-driver');
```

```
const fs = require('fs');
const sigV4 = require('aws-sigv4-auth-cassandra-plugin');

const auth = new sigV4.SigV4AuthProvider({
    region: 'us-west-2',
    accessKeyId:'AKIAIOSFODNN7EXAMPLE',
    secretAccessKey: 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY'});

const sslOptions1 = {
  ca: [
      fs.readFileSync('path_to_filecassandra/sf-class2-root.crt', 'utf-8')],
  host: 'cassandra.us-west-2.amazonaws.com',
  rejectUnauthorized: true
};


const client = new cassandra.Client({
  contactPoints: ['cassandra.us-west-2.amazonaws.com'],
  localDataCenter: 'us-west-2',
  authProvider: auth,
  sslOptions: sslOptions1,
  protocolOptions: { port: 9142 }
});


const query = 'SELECT * FROM system_schema.keyspaces';

client.execute(query).then(
    result => console.log('Row from Keyspaces %s', result.rows[0]))
    .catch( e=> console.log(`${e}`));
```

사용 노트:

1. "*path_to_file*/sf-class2-root.crt"을 첫 번째 단계에서 저장한 인증서 경로로 바꿉니다.
2. *### KeyId # ##### #### ### #### # ##* 액세스 키와 AccessKey 일치하는지 확인하십시오. `AWSCredentialsProvider` 자세한 내용은 Node.js 형식의AWS SDK에서 Node.js 사용자 인증 정보 설정을 참조하십시오. JavaScript
3. 액세스 키를 코드 외부에 저장하는 경우의 모범 사례는 the section called “액세스 키를 관리하는 방법” 섹션을 참조하세요.
4. 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

Cassandra. NET Core 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스

이 섹션에서는 .NET Core 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 접속하는 방법을 소개합니다. 설정 단계는 환경 및 운영 체제에 따라 다르므로 그에 따라 수정해야 할 수도 있습니다. Amazon Keyspaces에서는 클라이언트와의 연결을 보호하는 데 도움이 되는 전송 계층 보안(TLS)을 사용해야 합니다. TLS를 사용하여 Amazon Keyspace에 접속하려면 Starfield 디지털 인증서를 다운로드하고 TLS를 사용하도록 드라이버를 구성해야 합니다.

1. Starfield 인증서를 다운로드하고 로컬 디렉터리에 저장한 다음 경로를 기록해 둡니다. 다음은 를 사용한 PowerShell 예제입니다.

```
$client = new-object System.Net.WebClient
$client.DownloadFile("https://certs.secureserver.net/repository/sf-class2-
root.crt","path_to_file\sf-class2-root.crt")
```

2. 너겟 콘솔을 사용하여 SharpDriver 너겟을 통해 카산드랙을 설치합니다.

```
PM> Install-Package CassandraCSharpDriver
```

3. 다음 예시는.NET Core C# 콘솔 프로젝트를 사용하여 Amazon Keyspace에 연결하고 쿼리를 실행합니다.

```
using Cassandra;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Security;
using System.Runtime.ConstrainedExecution;
using System.Security.Cryptography.X509Certificates;
using System.Text;
using System.Threading.Tasks;

namespace CSharpKeyspacesExample
{
    class Program
    {
        public Program(){}

        static void Main(string[] args)
        {
```

```
            X509Certificate2Collection certCollection = new
 X509Certificate2Collection();
            X509Certificate2 amazoncert = new X509Certificate2(@"path_to_file\sf-
class2-root.crt");
            var userName = "ServiceUserName";
            var pwd = "ServicePassword";
            certCollection.Add(amazoncert);

            var awsEndpoint =  "cassandra.us-east-2.amazonaws.com" ;

            var cluster = Cluster.Builder()
                     .AddContactPoints(awsEndpoint)
                     .WithPort(9142)
                     .WithAuthProvider(new PlainTextAuthProvider(userName, pwd))
                     .WithSSL(new
 SSLOptions().SetCertificateCollection(certCollection))
                     .Build();

            var session = cluster.Connect();
            var rs = session.Execute("SELECT * FROM system_schema.tables;");
            foreach (var row in rs)
            {
                var name = row.GetValue<String>("keyspace_name");
                Console.WriteLine(name);
            }
        }
    }
}
```

사용 노트:

a. "*path_to_file*/sf-class2-root.crt"을 첫 번째 단계에서 저장한 인증서 경로로 바꿉니다.

b. 의 단계에 따라 *ServiceUserName* and 가 서비스별 자격 증명을 생성할 때 얻은 사용자 이름 및 암호와 *ServicePassword*일치하는지 확인하십시오. 서비스별 보안 인증 정보 생성

c. 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

Cassandra Go 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스

이 섹션에서는 Go 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 접속하는 방법을 소개합니다. Amazon Keyspaces 리소스에 프로그래밍 방식으로 액세스할 수 있는 보안 인증 정보를 사용자와 애플리케이션에 제공하려면 다음 중 하나를 수행할 수 있습니다.

- 특정 AWS Identity and Access Management (IAM) 사용자와 연결된 서비스별 보안 인증 정보를 생성합니다.
- 보안 강화를 위해 모든 서비스에서 사용되는 IAM 사용자 및 역할에 대한 IAM 액세스 키를 생성하는 것이 좋습니다. AWS Cassandra 클라이언트 드라이버용 Amazon Keyspaces SigV4 인증 플러그인을 사용하면 사용자 이름 및 암호 대신 IAM 액세스 키를 사용하여 Amazon Keyspaces에 대한 호출을 인증할 수 있습니다. 자세한 정보는 the section called “인증을 위한 IAM 자격 증명 AWS ”을 참조하세요.

주제

시작하기 전 준비 사항

시작하기 전에 다음 작업을 수행해야 합니다.

Amazon Keyspaces에서는 클라이언트와의 연결을 보호하는 데 도움이 되는 전송 계층 보안(TLS)을 사용해야 합니다. TLS를 사용하여 Amazon Keyspace에 접속하려면 Amazon 디지털 인증서를 다운로드하고 TLS를 사용하도록 Python 드라이버를 구성해야 합니다.

다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드하고 `sf-class2-root.crt`를 로컬 또는 홈 디렉터리에 저장합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> 또한 Amazon 디지털 인증서를 사용하여 Amazon Keyspaces에 접속할 수 있으며, 클라이언트가 Amazon Keyspaces에 성공적으로 접속하면 계속 접속할 수 있습니다. Starfield 인증서는 이전 인증 기관을 사용하는 클라이언트에게 추가적인 이전 버전과의 호환성을 제공합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

Apache Cassandra용 Gocql 드라이버와 서비스별 보안 인증 정보를 사용하여 Amazon Keyspaces에 접속합니다.

1. 애플리케이션용 디렉터리를 생성합니다.

   ```
   mkdir ./gocqlexample
   ```

2. 새 디렉터리로 이동합니다.

   ```
   cd gocqlexample
   ```

3. 애플리케이션에 대한 파일을 생성합니다.

   ```
   touch cqlapp.go
   ```

4. Go 드라이버를 다운로드합니다.

   ```
   go get github.com/gocql/gocql
   ```

5. 다음 샘플 코드를 cqlapp.go 파일에 추가합니다.

   ```
   package main

   import (
        "fmt"
        "github.com/gocql/gocql"
        "log"
   )

   func main() {
   ```

```
    // add the Amazon Keyspaces service endpoint
    cluster := gocql.NewCluster("cassandra.us-east-2.amazonaws.com")
    cluster.Port=9142
    // add your service specific credentials
    cluster.Authenticator = gocql.PasswordAuthenticator{
            Username: "ServiceUserName",
            Password: "ServicePassword"}
    // provide the path to the sf-class2-root.crt
    cluster.SslOpts = &gocql.SslOptions{
            CaPath: "path_to_file/sf-class2-root.crt",
            EnableHostVerification: false,
     }

     // Override default Consistency to LocalQuorum
     cluster.Consistency = gocql.LocalQuorum
     cluster.DisableInitialHostLookup = false

     session, err := cluster.CreateSession()
     if err != nil {
            fmt.Println("err>", err)
     }
     defer session.Close()

     // run a sample query from the system keyspace
     var text string
     iter := session.Query("SELECT keyspace_name FROM system_schema.tables;").Iter()
     for iter.Scan(&text) {
            fmt.Println("keyspace_name:", text)
     }
     if err := iter.Close(); err != nil {
            log.Fatal(err)
     }
     session.Close()
}
```

사용 노트:

a. "*path_to_file*/sf-class2-root.crt"을 첫 번째 단계에서 저장한 인증서 경로로 바꿉니다.

b. 의 단계에 따라 *ServiceUserName* and 가 서비스별 자격 증명을 생성할 때 얻은 사용자 이름 및 암호와 *ServicePassword*일치하는지 확인하십시오. 서비스별 보안 인증 정보 생성

c. 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

6. 프로그램을 빌드합니다.

```
go build cqlapp.go
```

7. 프로그램을 실행합니다.

```
./cqlapp
```

Apache Cassandra용 Go 드라이버와 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 접속

다음 코드 샘플에서는 오픈 소스 Go 드라이버를 위한 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces(Apache Cassandra용) 에 액세스하는 방법을 설명합니다.

아직 수행하지 않은 경우 the section called “인증을 위한 IAM 자격 증명 AWS ”의 단계에 따라 IAM 사용자 또는 역할의 보안 인증 정보를 생성합니다.

리포지토리에서 Go SigV4 인증 플러그인을 애플리케이션에 추가합니다. GitHub 플러그인은 카산드라용 오픈 소스 Go 드라이버 버전 1.2.x를 지원하며 Go용 SDK에 따라 다릅니다. AWS

```
$ go mod init
$ go get github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin
```

이 코드 샘플에서 Amazon Keyspaces 엔드포인트는 `Cluster` 클래스로 표현됩니다. 클러스터의 인증 속성에 대한 `AwsAuthenticator`을 통해 보안 인증 정보가 생성됩니다.

```
package main

import (
        "fmt"
        "github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin/sigv4"
        "github.com/gocql/gocql"
        "log"
)

func main() {
    // configuring the cluster options
    cluster := gocql.NewCluster("cassandra.us-west-2.amazonaws.com")
    cluster.Port=9142
    var auth sigv4.AwsAuthenticator = sigv4.NewAwsAuthenticator()
    auth.Region = "us-west-2"
```

```
    auth.AccessKeyId = "AKIAIOSFODNN7EXAMPLE"
    auth.SecretAccessKey = "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY"

    cluster.Authenticator = auth

    cluster.SslOpts = &gocql.SslOptions{
            CaPath: "path_to_file/sf-class2-root.crt",
            EnableHostVerification: false,
    }
    cluster.Consistency = gocql.LocalQuorum
    cluster.DisableInitialHostLookup = false

    session, err := cluster.CreateSession()
    if err != nil {
     fmt.Println("err>", err)
     return
    }
    defer session.Close()

    // doing the query
    var text string
    iter := session.Query("SELECT keyspace_name FROM system_schema.tables;").Iter()
    for iter.Scan(&text) {
     fmt.Println("keyspace_name:", text)
    }
    if err := iter.Close(); err != nil {
     log.Fatal(err)
    }
}
```

사용 노트:

1. "*path_to_file*/sf-class2-root.crt"을 첫 번째 단계에서 저장한 인증서 경로로 바꿉니다.
2. *AccessKeyId* 및 *SecretAccessKey# #### ### ### #* 및 비밀 액세스 키와 일치하는지 확인하세요. AwsAuthenticator 자세한 내용은 AWS SDK for Go의 Go용 AWS SDK 구성을 참조하세요.
3. 액세스 키를 코드 외부에 저장하는 경우의 모범 사례는 the section called “액세스 키를 관리하는 방법” 섹션을 참조하세요.
4. 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

Cassandra Perl 클라이언트 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스

이 섹션에서는 Perl 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 접속하는 방법을 소개합니다. 이 코드 샘플에서는 Perl 5를 사용했습니다. Amazon Keyspaces에서는 클라이언트와의 연결을 보호하는 데 도움이 되는 전송 계층 보안(TLS)을 사용해야 합니다.

> **Important**
>
> 보안 연결을 만들기 위해 코드 샘플은 TLS 연결을 설정하기 전에 Starfield 디지털 인증서를 사용하여 서버를 인증합니다. Perl 드라이버는 서버의 Amazon SSL 인증서를 검증하지 않으므로 Amazon Keyspaces에 접속하고 있는지 확인할 수 없습니다. 두 번째 단계는 Amazon Keyspaces에 접속해야 할 때 TLS를 사용하도록 드라이버를 구성하고 클라이언트와 서버 간에 전송되는 데이터가 암호화되도록 하는 것입니다.

1. https://metacpan.org/pod/DBD::Cassandra에서 Cassandra DBI 드라이버를 다운로드하여 Perl 환경에 설치합니다. 정확한 단계는 환경에 따라 달라집니다. 다음은 일반적인 예제입니다.

```
cpanm DBD::Cassandra
```

2. 애플리케이션에 대한 파일을 생성합니다.

```
touch cqlapp.pl
```

3. 다음 샘플 코드를 cqlapp.pl 파일에 추가합니다.

```
use DBI;
my $user = "ServiceUserName";
my $password = "ServicePassword";
my $db = DBI->connect("dbi:Cassandra:host=cassandra.us-
east-2.amazonaws.com;port=9142;tls=1;",
$user, $password);

my $rows = $db->selectall_arrayref("select * from system_schema.keyspaces");
print "Found the following Keyspaces...\n";
for my $row (@$rows) {
      print join(" ",@$row['keyspace_name']),"\n";
}
```

```
$db->disconnect;
```

> **Important**
>
> 의 단계에 따라 *ServiceUser###* 서비스별 자격 증명을 생성할 때 얻은 사용자 이름 및 암호와 *ServicePassword*일치하는지 확인하십시오. 서비스별 보안 인증 정보 생성

> **Note**
>
> 사용 가능한 엔드포인트 목록은 the section called “서비스 엔드포인트” 섹션을 참조하세요.

4. 애플리케이션을 실행합니다.

```
perl cqlapp.pl
```

튜토리얼: 아마존 Elastic Kubernetes Service에서 아마존 키스페이스에 연결

이 자습서에서는 SiGV4 인증을 사용하여 Amazon Keyspace에 연결하는 컨테이너식 애플리케이션을 호스팅하도록 Amazon Elastic Kubernetes Service (Amazon EKS) 클러스터를 설정하는 데 필요한 단계를 안내합니다.

Amazon EKS는 자체 Kubernetes 컨트롤 플레인을 설치, 운영 및 유지 관리할 필요가 없는 관리형 서비스입니다. Kubernetes는 컨테이너화된 애플리케이션의 관리, 규모 조정 및 배포포를 자동화하는 오픈 소스 시스템입니다.

이 자습서는 컨테이너식 Java 애플리케이션을 구성, 구축 및 Amazon EKS에 배포하기 위한 step-by-step 지침을 제공합니다. 마지막 단계에서는 애플리케이션을 실행하여 Amazon Keyspaces 테이블에 데이터를 씁니다.

주제

- 자습서 사전 요구 사항
- 1단계: Amazon EKS 클러스터를 구성하고 IAM 권한을 설정합니다.
- 2단계: 애플리케이션 구성
- 3단계: 애플리케이션 이미지를 생성하고 Amazon ECR 리포지토리에 Docker 파일을 업로드합니다.
- 4단계: Amazon EKS에 애플리케이션을 배포하고 Amazon Keyspaces 테이블에 데이터를 씁니다.

- 5단계: (선택 사항) 정리

자습서 사전 요구 사항

튜토리얼을 시작하기 전에 다음 AWS 리소스를 생성하세요.

1. 이 자습서를 시작하기 전에 의 AWS 설정 지침을 따르십시오Amazon Keyspaces(Apache Cassandra용) 액세스. 이러한 단계에는 Amazon Keyspace에 액세스할 수 있는 AWS Identity and Access Management (IAM) 보안 주체 가입 AWS 및 생성이 포함됩니다.
2. 이 자습서의 뒷부분에서 Amazon EKS에서 실행되는 컨테이너식 애플리케이션에서 쓸 수 `user` 있는 이름과 이름을 가진 테이블을 사용하여 Amazon Keyspaces 키스페이스를 생성합니다. `aws` 를 사용하거나 를 사용하여 이 작업을 수행할 수 있습니다. AWS CLI `cqlsh`

   AWS CLI

   ```
   aws keyspaces create-keyspace --keyspace-name 'aws'
   ```

   키스페이스가 생성되었는지 확인하려면 다음 명령을 사용할 수 있습니다.

   ```
   aws keyspaces list-keyspaces
   ```

   테이블을 생성하려면 다음 명령을 사용할 수 있습니다.

   ```
   aws keyspaces create-table --keyspace-name 'aws' --table-name 'user' --schema-
   definition 'allColumns=[
               {name=username,type=text}, {name=fname,type=text},
   {name=last_update_date,type=timestamp},{name=lname,type=text}],
               partitionKeys=[{name=username}]'
   ```

   다음 명령을 사용하여 테이블이 생성되었는지 확인할 수 있습니다.

   ```
   aws keyspaces list-tables --keyspace-name 'aws'
   ```

   자세한 내용은 AWS CLI 명령 참조의 키스페이스 생성 및 테이블 생성을 참조하십시오.

   cqlsh

   ```
   CREATE KEYSPACE aws WITH replication = {'class': 'SimpleStrategy',
    'replication_factor': '3'}  AND durable_writes = true;
   ```

```
CREATE TABLE aws.user (
    username text PRIMARY KEY,
    fname text,
    last_update_date timestamp,
    lname text
);
```

테이블이 생성되었는지 확인하려면 다음 명령문을 사용할 수 있습니다.

```
SELECT * FROM system_schema.tables WHERE keyspace_name = "aws";
```

이 명령문의 출력에는 테이블이 나열되어야 합니다. 테이블이 생성될 때까지 지연이 발생할 수 있다는 점에 유의하십시오. 자세한 설명은 the section called “CREATE TABLE” 섹션을 참조하세요.

3. Fargate - 리눅스 노드 유형을 사용하여 Amazon EKS 클러스터를 생성합니다. Fargate는 Amazon Amazon EC2 인스턴스를 관리하지 않고도 쿠버네티스 파드를 배포할 수 있는 서버리스 컴퓨팅 엔진입니다. 모든 예제 명령에서 클러스터 이름을 업데이트할 필요 없이 이 자습서를 따르려면 Amazon EKS 사용 설명서의 Amazon EKS *eksctl* 시작하기-에 있는 지침에 따라 이름을 *my-eks-cluster* 사용하여 클러스터를 생성하십시오. 클러스터를 생성할 때 노드와 두 개의 기본 파드가 실행 중이고 정상인지 확인하십시오. 다음 명령어로 이 작업을 수행할 수 있습니다.

```
kubectl get pods -A -o wide
```

이 출력과 비슷한 내용이 표시될 것입니다.

```
NAMESPACE     NAME                       READY   STATUS    RESTARTS   AGE   IP
     NODE                                                  NOMINATED NODE   READINESS
 GATES
kube-system   coredns-1234567890-abcde   1/1     Running   0          18m
 192.0.2.0   fargate-ip-192-0-2-0.region-code.compute.internal   <none>
 <none>
kube-system   coredns-1234567890-12345   1/1     Running   0          18m
 192.0.2.1   fargate-ip-192-0-2-1.region-code.compute.internal   <none>
 <none>
```

4. 도커를 설치합니다. Amazon EC2 인스턴스에 Docker를 설치하는 방법에 대한 지침은 Amazon Elastic 컨테이너 레지스트리 사용 설명서의 Docker 설치를 참조하십시오.

Docker는 최신 Linux 배포 버전(Ubuntu 등)을 비롯하여 macOS 및 Windows 등 다양한 운영 체제에서 사용할 수 있습니다. 특정 운영 체제에 Docker를 설치하는 방법에 대한 자세한 내용은 Docker 설치 안내서를 참조하십시오.

5. Amazon ECR 리포지토리를 생성합니다. Amazon ECR은 선호하는 CLI와 함께 사용하여 Docker 이미지를 푸시하고, 가져오고, 관리할 수 있는 AWS 관리형 컨테이너 이미지 레지스트리 서비스입니다. Amazon ECR 리포지토리에 대한 자세한 내용은 Amazon Elastic 컨테이너 레지스트리 사용 설명서를 참조하십시오. 다음 명령을 사용하여 이름이 지정된 리포지토리를 생성할 수 있습니다. `my-ecr-repository`

```
aws ecr create-repository --repository-name my-ecr-repository
```

필수 단계를 완료한 후 the section called “1단계: Amazon EKS 클러스터 구성”로 진행합니다.

1단계: Amazon EKS 클러스터를 구성하고 IAM 권한을 설정합니다.

Amazon EKS 클러스터를 구성하고 Amazon EKS 서비스 계정을 Amazon Keyspaces 테이블에 연결하는 데 필요한 IAM 리소스를 생성합니다.

1. Amazon EKS 클러스터를 위한 오픈 ID 연결 (OIDC) 공급자를 생성합니다. 이는 서비스 계정의 IAM 역할을 사용하는 데 필요합니다. OIDC 공급자 및 OIDC 공급자 생성 방법에 대한 자세한 내용은 Amazon EKS 사용 설명서의 클러스터용 IAM OIDC 공급자 생성을 참조하십시오.

   a. 다음 명령을 사용하여 클러스터의 IAM OIDC 자격 증명 공급자를 생성합니다. 이 예제에서는 클러스터 이름이 다음과 같다고 가정합니다. `my-eks-cluster` 클러스터의 이름이 다른 경우 향후 모든 명령에서 이름을 업데이트해야 합니다.

```
eksctl utils associate-iam-oidc-provider --cluster my-eks-cluster --approve
```

   b. 다음 명령을 사용하여 OIDC ID 공급자가 IAM에 등록되었는지 확인합니다.

```
aws iam list-open-id-connect-providers --region aws-region
```

   결과는 다음과 비슷해야 합니다. 다음 단계에서 서비스 계정에 대한 신뢰 정책을 생성할 때 필요한 OIDC의 Amazon 리소스 이름 (ARN) 을 기록해 두십시오.

```
{
    "OpenIDConnectProviderList": [
```

```
        ..
        {
            "Arn": "arn:aws:iam::111122223333:oidc-provider/oidc.eks.aws-
region.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
        }
    ]
}
```

2. Amazon EKS 클러스터용 서비스 계정을 생성합니다. 서비스 계정은 파드에서 실행되는 프로세스에 대한 ID를 제공합니다. 파드는 컨테이너식 애플리케이션을 배포하는 데 사용할 수 있는 가장 작고 단순한 Kubernetes 객체입니다. 다음으로, 서비스 계정이 리소스에 대한 권한을 얻기 위해 맡을 수 있는 IAM 역할을 생성합니다. 해당 AWS 서비스에 대한 액세스 권한이 있는 IAM 역할을 수임할 수 있는 서비스 계정을 사용하도록 구성된 파드에서 모든 서비스에 액세스할 수 있다.

   a. 서비스 계정의 새 네임스페이스를 생성합니다. 네임스페이스는 이 자습서에서 만든 클러스터 리소스를 격리하는 데 도움이 됩니다. 다음 명령을 사용하여 새 네임스페이스를 만들 수 있습니다.

   ```
   kubectl create namespace my-eks-namespace
   ```

   b. 사용자 지정 네임스페이스를 사용하려면 Fargate 프로필에 연결해야 합니다. 다음 코드는 이에 대한 예입니다.

   ```
   eksctl create fargateprofile \
       --cluster my-eks-cluster \
       --name my-fargate-profile \
       --namespace my-eks-namespace \
       --labels *=*
   ```

   c. 다음 명령을 사용하여 Amazon EKS 클러스터의 `my-eks-namespace` 네임스페이스에 있는 이름을 `my-eks-serviceaccount` 사용하여 서비스 계정을 생성합니다.

   ```
   cat >my-serviceaccount.yaml <<EOF
   apiVersion: v1
   kind: ServiceAccount
   metadata:
     name: my-eks-serviceaccount
     namespace: my-eks-namespace
   EOF
   kubectl apply -f my-serviceaccount.yaml
   ```

d. 다음 명령을 실행하여 IAM 역할이 서비스 계정을 신뢰하도록 지시하는 신뢰 정책 파일을 생성합니다. 보안 주체가 역할을 맡으려면 먼저 이 신뢰 관계가 필요합니다. 파일을 다음과 같이 편집해야 합니다.

- 의 `Principal` 경우 IAM이 명령에 반환한 ARN을 입력합니다. `list-open-id-connect-providers` ARN에는 계정 번호와 지역이 포함됩니다.
- `condition`명세서에서 AWS 리전 및 OIDC ID를 바꾸십시오.
- 서비스 계정 이름과 네임스페이스가 올바른지 확인하십시오.

다음 단계에서 IAM 역할을 생성할 때 신뢰 정책 파일을 첨부해야 합니다.

```
cat >trust-relationship.json <<EOF
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.aws-region.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.aws-region.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:my-eks-
namespace:my-eks-serviceaccount",
                    "oidc.eks.aws-region.amazonaws.com/
id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}
EOF
```

선택 사항: `StringEquals` 또는 `StringLike` 조건에 여러 항목을 추가하여 여러 서비스 계정 또는 네임스페이스가 역할을 수임하도록 허용할 수도 있습니다. 서비스 계정이 다른 AWS 계정의 IAM 역할을 맡도록 허용하려면 Amazon EKS 사용 설명서의 계정 간 IAM 권한을 참조하십시오.

3. 맡을 Amazon EKS 서비스 `my-iam-role` 계정의 이름으로 IAM 역할을 생성합니다. 마지막 단계에서 생성한 신뢰 정책 파일을 역할에 연결합니다. 신뢰 정책은 IAM 역할이 신뢰할 수 있는 서비스 계정과 OIDC 공급자를 지정합니다.

```
aws iam create-role --role-name my-iam-role --assume-role-policy-document file://
trust-relationship.json --description "EKS service account role"
```

4. 액세스 정책을 첨부하여 Amazon Keyspace에 IAM 역할 권한을 할당합니다.

   a. 액세스 정책을 첨부하여 IAM 역할이 특정 Amazon Keyspaces 리소스에서 수행할 수 있는 작업을 정의합니다. 이 자습서에서는 애플리케이션이 Amazon Keyspaces 테이블에 데이터를 쓰기 때문에 AWS 관리형 정책을 `AmazonKeyspacesFullAccess` 사용합니다. 하지만 모범 사례로는 최소 권한 원칙을 구현하는 사용자 지정 액세스 정책을 생성하는 것이 좋습니다. 자세한 설명은 the section called “Amazon Keyspaces에서 IAM을 사용하는 방법” 섹션을 참조하세요.

```
aws iam attach-role-policy --role-name my-iam-role --policy-
arn=arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess
```

   다음 설명을 사용하여 정책이 IAM 역할에 성공적으로 연결되었는지 확인하십시오.

```
aws iam list-attached-role-policies --role-name my-iam-role
```

   결과는 다음과 같아야 합니다.

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonKeyspacesFullAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess"
        }
    ]
}
```

   b. 서비스 계정에 해당 계정이 맡을 수 있는 IAM 역할의 Amazon 리소스 이름 (ARN) 으로 주석을 답니다. 계정 ID로 역할 ARN을 업데이트해야 합니다.

```
kubectl annotate serviceaccount -n my-eks-namespace my-eks-serviceaccount
 eks.amazonaws.com/role-arn=arn:aws:iam::111122223333:role/my-iam-role
```

5. IAM 역할과 서비스 계정이 올바르게 구성되었는지 확인합니다.

   a. 다음 명령문을 사용하여 IAM 역할의 신뢰 정책이 올바르게 구성되었는지 확인합니다.

```
aws iam get-role --role-name my-iam-role --query Role.AssumeRolePolicyDocument
```

   결과는 다음과 비슷해야 합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::111122223333:oidc-provider/
oidc.eks.aws-region.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "oidc.eks.aws-region/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.amazonaws.com",
                    "oidc.eks.aws-region.amazonaws.com/id/
EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system:serviceaccount:my-eks-
namespace:my-eks-serviceaccount"
                }
            }
        }
    ]
}
```

   b. Amazon EKS 서비스 계정에 IAM 역할 주석이 달렸는지 확인합니다.

```
kubectl describe serviceaccount my-eks-serviceaccount -n my-eks-namespace
```

   결과는 다음과 비슷해야 합니다.

```
Name: my-eks-serviceaccount
Namespace:my-eks-namespace
Labels: <none>
Annotations: eks.amazonaws.com/role-arn: arn:aws:iam::111122223333:role/my-iam-
role
```

```
Image pull secrets: <none>
Mountable secrets: <none>
Tokens: <none>
[...]
```

Amazon EKS 서비스 계정, IAM 역할을 생성하고 필요한 관계 및 권한을 구성한 후 다음 단계로 진행하십시오. the section called “2단계: 애플리케이션 구성”

2단계: 애플리케이션 구성

이 단계에서는 SigV4 플러그인을 사용하여 Amazon Keyspace에 연결되는 애플리케이션을 구축합니다. Github의 Amazon Keyspaces 예제 코드 리포지토리에서 예제 자바 애플리케이션을 보고 다운로드할 수 있습니다. 또는 자체 애플리케이션을 사용하여 모든 구성 단계를 완료하여 따라할 수도 있습니다.

애플리케이션을 구성하고 필요한 종속성을 추가하세요.

1. 다음 명령을 사용하여 Github 리포지토리를 복제하여 예제 Java 애플리케이션을 다운로드할 수 있습니다.

   ```
   git clone https://github.com/aws-samples/amazon-keyspaces-examples.git
   ```

2. Github 리포지토리를 다운로드한 후 다운로드한 파일의 압축을 풀고 디렉토리에서 해당 파일로 이동합니다. `resources` `application.conf`

   a. 애플리케이션 구성

   이 단계에서는 SigV4 인증 플러그인을 구성합니다. 애플리케이션에서 다음 예제를 사용할 수 있습니다. 아직 생성하지 않았다면 IAM 액세스 키 (액세스 키 ID 및 보안 액세스 키) 를 생성하여 AWS 구성 파일이나 환경 변수에 저장해야 합니다. 자세한 지침은 the section called “인증을 위한 필수 자격 증명 AWS ” 섹션을 참조하세요. 필요에 따라 Amazon Keyspaces의 AWS 지역 및 서비스 엔드포인트를 업데이트하십시오. 더 많은 서비스 엔드포인트는 을 참조하십시오. the section called “서비스 엔드포인트” 신뢰 저장소 위치, 신뢰 저장소 이름 및 신뢰 저장소 암호를 자신의 것으로 바꾸십시오.

   ```
   datastax-java-driver {
     basic.contact-points = ["cassandra.aws-region.amazonaws.com:9142"]
     basic.load-balancing-policy.local-datacenter = "aws-region"
     advanced.auth-provider {
       class = software.aws.mcs.auth.SigV4AuthProvider
   ```

```
    aws-region = "aws-region"
  }
  advanced.ssl-engine-factory {
    class = DefaultSslEngineFactory
    truststore-path = "truststore_locationtruststore_name.jks"
    truststore-password = "truststore_password;"
  }
}
```

b. STS 모듈 종속성을 추가합니다.

이렇게 하면 애플리케이션이 제공해야 하는 AWS 자격 증명을 `WebIdentityTokenCredentialsProvider` 반환하는 a를 사용하여 서비스 계정이 IAM 역할을 수임할 수 있는 기능이 추가되었습니다. 다음 예제를 기반으로 이 작업을 수행할 수 있습니다.

```
        <dependency>
            <groupId>com.amazonaws</groupId>
            <artifactId>aws-java-sdk-sts</artifactId>
            <version>1.11.717</version>
        </dependency>
```

c. SigV4 종속성을 추가합니다.

이 패키지는 Amazon Keyspaces에 인증하는 데 필요한 SigV4 인증 플러그인을 구현합니다.

```
        <dependency>
            <groupId>software.aws.mcs</groupId>
            <artifactId>aws-sigv4-auth-cassandra-java-driver-plugin</
artifactId>
            <version>4.0.3</version>
        </dependency>
```

3. 로깅 종속성을 추가하십시오.

로그가 없으면 연결 문제를 해결할 수 없습니다. 이 자습서에서는 로깅 `slf4j` 프레임워크로 사용하고 로그 출력을 저장하는 `logback.xml` 데 사용합니다. 연결을 debug 설정하기 위해 로깅 수준을 로 설정했습니다. 다음 예제를 사용하여 종속성을 추가할 수 있습니다.

```
    <dependency>
        <groupId>org.slf4j</groupId>
        <artifactId>slf4j-api</artifactId>
```

```
            <version>2.0.5</version>
        </dependency>
```

다음 코드 스니펫을 사용하여 로깅을 구성할 수 있습니다.

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">

        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</
pattern>
        </encoder>
    </appender>

    <root level="debug">
        <appender-ref ref="STDOUT" />
    </rootv
</configuration>
```

> **Note**
>
> 연결 debug 실패를 조사하려면 레벨이 필요합니다. 애플리케이션에서 Amazon Keyspaces에 성공적으로 연결되면 로깅 수준을 `info` 또는 `warning` 필요에 따라 변경할 수 있습니다.

3단계: 애플리케이션 이미지를 생성하고 Amazon ECR 리포지토리에 Docker 파일을 업로드합니다.

이 단계에서는 예제 애플리케이션을 컴파일하고, Docker 이미지를 빌드하고, 이미지를 Amazon ECR 리포지토리로 푸시합니다.

애플리케이션을 빌드하고, Docker 이미지를 빌드하고, Amazon Elastic 컨테이너 레지스트리에 제출하십시오.

1. 다음을 정의하는 빌드의 환경 변수를 설정합니다. AWS 리전 예제의 지역을 자신의 지역으로 바꾸세요.

```
export CASSANDRA_HOST=cassandra.aws-region.amazonaws.com:9142
```

```
export CASSANDRA_DC=aws-region
```

2. 다음 명령을 사용하여 Apache Maven 버전 3.6.3 이상으로 애플리케이션을 컴파일하십시오.

```
mvn clean install
```

그러면 디렉터리에 모든 종속성이 포함된 JAR 파일이 생성됩니다. target

3. 다음 명령을 사용하여 다음 단계에 필요한 ECR 리포지토리 URI를 검색합니다. 지역을 사용 중이던 지역으로 업데이트해야 합니다.

```
aws ecr describe-repositories --region aws-region
```

출력은 다음 예와 같아야 합니다.

```
"repositories": [
 {
 "repositoryArn": "arn:aws:ecr:aws-region:111122223333:repository/my-ecr-
repository",
 "registryId": "111122223333",
 "repositoryName": "my-ecr-repository",
 "repositoryUri": "111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-
repository",
 "createdAt": "2023-11-02T03:46:34+00:00",
 "imageTagMutability": "MUTABLE",
 "imageScanningConfiguration": {
 "scanOnPush": false
 },
 "encryptionConfiguration": {
 "encryptionType": "AES256"
 }
 },
```

4. 애플리케이션의 루트 디렉터리에서 마지막 단계의 리포지토리 URI를 사용하여 Docker 이미지를 빌드합니다. 필요에 따라 Docker 파일을 수정합니다. 빌드 명령에서 계정 ID를 바꾸고 Amazon ECR my-ecr-repository 리포지토리가 위치한 지역으로 AWS 리전 설정해야 합니다.

```
docker build -t 111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-
repository:latest .
```

5. 인증 토큰을 검색하여 Docker 이미지를 Amazon ECR로 푸시합니다. 다음 명령을 사용하여 이 작업을 수행할 수 있습니다.

```
aws ecr get-login-password --region aws-region | docker login --username AWS --
password-stdin 111122223333.dkr.ecr.aws-region.amazonaws.com
```

6. 먼저 Amazon ECR 리포지토리에 있는 기존 이미지를 확인합니다. 다음 명령을 사용할 수 있습니다.

```
aws ecr describe-images --repository-name my-ecr-repository --region aws-region
```

그런 다음 Docker 이미지를 리포지토리로 푸시합니다. 다음 명령을 사용할 수 있습니다.

```
docker push 111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-repository:latest
```

4단계: Amazon EKS에 애플리케이션을 배포하고 Amazon Keyspaces 테이블에 데이터를 씁니다.

자습서의 이 단계에서는 애플리케이션에 대한 Amazon EKS 배포를 구성하고 애플리케이션이 실행 중이며 Amazon Keyspace에 연결할 수 있는지 확인합니다.

Amazon EKS에 애플리케이션을 배포하려면 라는 `deployment.yaml` 파일에서 모든 관련 설정을 구성해야 합니다. 그러면 Amazon EKS에서 이 파일을 사용하여 애플리케이션을 배포합니다. 파일의 메타데이터에는 다음 정보가 포함되어야 합니다.

- 애플리케이션 이름: 애플리케이션 이름. 이 자습서에서는 다음을 사용합니다`my-keyspaces-app`.
- 쿠버네티스 네임스페이스는 Amazon EKS 클러스터의 네임스페이스입니다. 이 자습서에서는 `my-eks-namespace` 다음을 사용합니다.
- 아마존 EKS 서비스 계정 이름 아마존 EKS 서비스 계정의 이름. 이 자습서에서는 다음을 사용합니다. `my-eks-serviceaccount`
- 이미지 이름: 애플리케이션 이미지의 이름. 이 자습서에서는 다음을 사용합니다`my-keyspaces-app`.
- 이미지 URI: 아마존 ECR의 도커 이미지 URI입니다.
- AWS 계정 ID, AWS 계정 ID.
- IAM 역할 ARN은 서비스 계정이 수임하도록 생성된 IAM 역할의 ARN입니다. 이 자습서에서는 를 사용합니다. `my-iam-role`
- AWS 리전 Amazon EKS 클러스터를 AWS 리전 생성한 Amazon EKS 클러스터의 구성입니다.

이 단계에서는 Amazon Keyspace에 연결하고 테이블에 데이터를 쓰는 애플리케이션을 배포 및 실행합니다.

1. `deployment.yaml` 파일을 구성합니다. 다음 값을 바꿔야 합니다.

   - `name`
   - `namespace`
   - `serviceAccountName`
   - `image`
   - `AWS_ROLE_ARN value`
   - 더 AWS 리전 인 `CASSANDRA_HOST`
   - `AWS_REGION`

   다음 파일을 예로 사용할 수 있습니다.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-keyspaces-app
  namespace: my-eks-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app: my-keyspaces-app
  template:
    metadata:
      labels:
        app: my-keyspaces-app
    spec:
      serviceAccountName: my-eks-serviceaccount
      containers:
      - name: my-keyspaces-app
        image: 111122223333.dkr.ecr.aws-region.amazonaws.com/my-ecr-
repository:latest
        ports:
        - containerPort: 8080
        env:
        - name: CASSANDRA_HOST
          value: "cassandra.aws-region.amazonaws.com:9142"
```

```
        - name: CASSANDRA_DC
          value: "aws-region"
        - name: AWS_WEB_IDENTITY_TOKEN_FILE
          value: /var/run/secrets/eks.amazonaws.com/serviceaccount/token
        - name: AWS_ROLE_ARN
          value: "arn:aws:iam::111122223333:role/my-iam-role"
        - name: AWS_REGION
          value: "aws-region"
```

2. deployment.yaml를 배포합니다.

```
kubectl apply -f deployment.yaml
```

결과는 다음과 같아야 합니다.

```
deployment.apps/my-keyspaces-app created
```

3. Amazon EKS 클러스터의 네임스페이스에 있는 파드의 상태를 확인합니다.

```
kubectl get pods -n my-eks-namespace
```

결과가 다음 예제와 비슷해야 합니다.

```
NAME                    READY STATUS RESTARTS AGE
my-keyspaces-app-123abcde4f-g5hij 1/1 Running 0 75s
```

자세한 내용은 다음 명령을 사용할 수 있습니다.

```
kubectl describe pod my-keyspaces-app-123abcde4f-g5hij -n my-eks-namespace
```

```
Name:                 my-keyspaces-app-123abcde4f-g5hij
Namespace:            my-eks-namespace
Priority:             2000001000
Priority Class Name:  system-node-critical
Service Account:      my-eks-serviceaccount
Node:                 fargate-ip-192-168-102-209.ec2.internal/192.168.102.209
Start Time:           Thu, 23 Nov 2023 12:15:43 +0000
Labels:               app=my-keyspaces-app
                      eks.amazonaws.com/fargate-profile=my-fargate-profile
```

```
                      pod-template-hash=6c56fccc56
Annotations:          CapacityProvisioned: 0.25vCPU 0.5GB
                      Logging: LoggingDisabled: LOGGING_CONFIGMAP_NOT_FOUND
Status:               Running
IP:                   192.168.102.209
IPs:
  IP:           192.168.102.209
Controlled By:  ReplicaSet/my-keyspaces-app-6c56fccc56
Containers:
  my-keyspaces-app:
    Container ID:
 containerd://41ff7811d33ae4bc398755800abcdc132335d51d74f218ba81da0700a6f8c67b
    Image:          111122223333.dkr.ecr.aws-region.amazonaws.com/
my_eks_repository:latest
    Image ID:       111122223333.dkr.ecr.aws-region.amazonaws.com/
my_eks_repository@sha256:fd3c6430fc5251661efce99741c72c1b4b03061474940200d0524b84a951439c
    Port:           8080/TCP
    Host Port:      0/TCP
    State:          Running
      Started:      Thu, 23 Nov 2023 12:15:19 +0000
      Finished:     Thu, 23 Nov 2023 12:16:17 +0000
    Ready:          True
    Restart Count:  1
    Environment:
      CASSANDRA_HOST:                cassandra.aws-region.amazonaws.com:9142
      CASSANDRA_DC:                  aws-region
      AWS_WEB_IDENTITY_TOKEN_FILE:  /var/run/secrets/eks.amazonaws.com/
serviceaccount/token
      AWS_ROLE_ARN:                  arn:aws:iam::111122223333:role/my-iam-role
      AWS_REGION:                    aws-region
      AWS_STS_REGIONAL_ENDPOINTS:    regional
    Mounts:
      /var/run/secrets/eks.amazonaws.com/serviceaccount from aws-iam-token (ro)
      /var/run/secrets/kubernetes.io/serviceaccount from kube-api-access-fssbf (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  aws-iam-token:
    Type:                     Projected (a volume that contains injected data from
 multiple sources)
```

```
    TokenExpirationSeconds:  86400
  kube-api-access-fssbf:
    Type:                    Projected (a volume that contains injected data from
 multiple sources)
    TokenExpirationSeconds:  3607
    ConfigMapName:           kube-root-ca.crt
    ConfigMapOptional:       <nil>
    DownwardAPI:             true
QoS Class:                   BestEffort
Node-Selectors:              <none>
Tolerations:                 node.kubernetes.io/not-ready:NoExecute op=Exists for
 300s
                             node.kubernetes.io/unreachable:NoExecute op=Exists for
 300s
Events:
  Type     Reason           Age                From               Message
  ----     ------           ----               ----               -------
  Warning  LoggingDisabled  2m13s              fargate-scheduler  Disabled logging
 because aws-logging configmap was not found. configmap "aws-logging" not found
  Normal   Scheduled        89s                fargate-scheduler  Successfully
 assigned my-eks-namespace/my-keyspaces-app-6c56fccc56-mgs2m to fargate-
ip-192-168-102-209.ec2.internal
  Normal   Pulled           75s                kubelet
 Successfully pulled image "111122223333.dkr.ecr.aws-region.amazonaws.com/
my_eks_repository:latest" in 13.027s (13.027s including waiting)
  Normal   Pulling          54s (x2 over 88s)  kubelet            Pulling image
 "111122223333.dkr.ecr.aws-region.amazonaws.com/my_eks_repository:latest"
  Normal   Created          54s (x2 over 75s)  kubelet            Created container
 my-keyspaces-app
  Normal   Pulled           54s                kubelet
 Successfully pulled image "111122223333.dkr.ecr.aws-region.amazonaws.com/
my_eks_repository:latest" in 222ms (222ms including waiting)
  Normal   Started          53s (x2 over 75s)  kubelet            Started container
 my-keyspaces-app
```

4. 파드의 로그를 확인하여 애플리케이션이 실행 중이고 Amazon Keyspaces 테이블에 연결할 수 있는지 확인하십시오. 다음 명령을 사용하여 이 작업을 수행할 수 있습니다. 배포 이름을 바꿔야 합니다.

```
kubectl logs -f my-keyspaces-app-123abcde4f-g5hij -n my-eks-namespace
```

아래 예와 같이 Amazon Keyspaces에 대한 연결을 확인하는 애플리케이션 로그 항목을 볼 수 있어야 합니다.

```
2:47:20.553 [s0-admin-0] DEBUG c.d.o.d.i.c.metadata.MetadataManager
 - [s0] Adding initial contact points [Node(endPoint=cassandra.aws-
region.amazonaws.com/1.222.333.44:9142, hostId=null, hashCode=e750d92)]
22:47:20.562 [s0-admin-1] DEBUG c.d.o.d.i.c.c.ControlConnection - [s0] Initializing
 with event types [SCHEMA_CHANGE, STATUS_CHANGE, TOPOLOGY_CHANGE]
22:47:20.564 [s0-admin-1] DEBUG c.d.o.d.i.core.context.EventBus - [s0] Registering
 com.datastax.oss.driver.internal.core.metadata.LoadBalancingPolicyWrapper$$Lambda
$812/0x0000000801105e88@769afb95 for class
 com.datastax.oss.driver.internal.core.metadata.NodeStateEvent
22:47:20.566 [s0-admin-1] DEBUG c.d.o.d.i.c.c.ControlConnection -
 [s0] Trying to establish a connection to Node(endPoint=cassandra.us-
east-1.amazonaws.com/1.222.333.44:9142, hostId=null, hashCode=e750d92)
```

5. Amazon Keyspaces 테이블에서 다음 CQL 쿼리를 실행하여 한 행의 데이터가 테이블에 기록되었는지 확인합니다.

```
SELECT * from aws.user;
```

다음 결과가 표시됩니다.

```
fname    | lname | username | last_update_date
----------+-------+----------+---------------------------
random    | k     | test     | 2023-12-07 13:58:31.57+0000
```

5단계: (선택 사항) 정리

다음 단계에 따라 이 자습서에서 만든 모든 리소스를 삭제하세요.

이 자습서에서 만든 리소스를 삭제하세요.

1. 배포를 삭제합니다. 다음 명령을 사용하여 그렇게 할 수 있습니다.

```
kubectl delete deployment my-keyspaces-app -n my-eks-namespace
```

2. Amazon EKS 클러스터와 여기에 포함된 모든 파드를 삭제합니다. 이렇게 하면 서비스 계정 및 OIDC ID 공급자와 같은 관련 리소스도 삭제됩니다. 다음 명령을 사용하여 이 작업을 수행할 수 있습니다.

```
eksctl delete cluster --name my-eks-cluster --region aws-region
```

3. Amazon Keyspace에 대한 액세스 권한이 있는 Amazon EKS 서비스 계정에 사용되는 IAM 역할을 삭제합니다. 먼저 역할에 연결된 관리형 정책을 제거해야 합니다.

```
aws iam detach-role-policy --role-name my-iam-role --policy-arn
 arn:aws:iam::aws:policy/AmazonKeyspacesFullAccess
```

그런 다음 다음 명령을 사용하여 역할을 삭제할 수 있습니다.

```
aws iam delete-role --role-name my-iam-role
```

자세한 내용은 IAM 사용 설명서의 IAM 역할 삭제 (AWS CLI) 를 참조하십시오.

4. Amazon ECR 리포지토리에 저장된 모든 이미지를 포함하여 해당 리포지토리를 삭제합니다. 다음 명령을 사용하여 이 작업을 수행할 수 있습니다.

```
aws ecr delete-repository \
      --repository-name my-ecr-repository \
      --force \
      --region aws-region
```

이미지가 포함된 리포지토리를 삭제하려면 `force` 플래그가 필요하다는 점에 유의하세요. 이미지를 먼저 삭제하려면 다음 명령을 사용하면 됩니다.

```
aws ecr batch-delete-image \
      --repository-name my-ecr-repository \
      --image-ids imageTag=latest \
      --region aws-region
```

자세한 내용은 Amazon Elastic 컨테이너 레지스트리 사용 설명서의 이미지 삭제를 참조하십시오.

5. Amazon Keyspaces 키스페이스 및 테이블을 삭제합니다. 키스페이스를 삭제하면 해당 키스페이스의 모든 테이블이 자동으로 삭제됩니다. 다음 옵션 중 하나를 사용하여 삭제할 수 있습니다.

AWS CLI

```
aws keyspaces delete-keyspace --keyspace-name 'aws'
```

키스페이스가 삭제되었는지 확인하려면 다음 명령을 사용할 수 있습니다.

```
aws keyspaces list-keyspaces
```

테이블을 먼저 삭제하려면 다음 명령을 사용할 수 있습니다.

```
aws keyspaces delete-table --keyspace-name 'aws' --table-name 'user'
```

다음 명령을 사용하여 테이블이 삭제되었는지 확인할 수 있습니다.

```
aws keyspaces list-tables --keyspace-name 'aws'
```

자세한 내용은 AWS CLI 명령 참조의 키스페이스 삭제 및 테이블 삭제를 참조하십시오.

cqlsh

```
DROP KEYSPACE IF EXISTS "aws";
```

키스페이스가 삭제되었는지 확인하려면 다음 명령문을 사용할 수 있습니다.

```
SELECT * FROM system_schema.keyspaces ;
```

이 명령문의 출력에 키스페이스가 나열되어서는 안 됩니다. 키스페이스가 삭제될 때까지 지연이 있을 수 있다는 점에 유의하십시오. 자세한 설명은 the section called “DROP KEYSPACE” 섹션을 참조하세요.

테이블을 먼저 삭제하려면 다음 명령을 사용할 수 있습니다.

```
DROP TABLE "aws.user"
```

다음 명령을 사용하여 테이블이 삭제되었는지 확인할 수 있습니다.

```
SELECT * FROM system_schema.tables WHERE keyspace_name = "aws";
```

이 명령문의 출력에 테이블이 나열되어서는 안 됩니다. 테이블이 삭제될 때까지 지연이 있을 수 있다는 점에 유의하십시오. 자세한 내용은 the section called “DROP TABLE”을(를) 참조하세요.

자습서: 인터페이스 VPC 엔드포인트를 사용하여 Amazon Keyspaces에 연결

이 자습서에서는 Amazon Keyspaces에 대한 인터페이스 VPC 엔드포인트를 설정 및 사용하는 절차를 살펴봅니다.

인터페이스 VPC 엔드포인트를 사용하면 Amazon VPC에서 실행되는 Virtual Private Cloud(VPC)와 Amazon Keyspaces 간에 프라이빗 통신이 가능해집니다. 인터페이스 VPC 엔드포인트는 VPC와 서비스 간의 개인 통신을 가능하게 하는 AWS 서비스인 에 의해 AWS PrivateLink구동됩니다. AWS 자세한 정보는 the section called “인터페이스 VPC 엔드포인트 사용”을 참조하세요.

주제

자습서 사전 조건 및 고려 사항

이 자습서를 시작하기 전에 의 AWS 설정 지침을 따르십시오. Amazon Keyspaces(Apache Cassandra용) 액세스 이러한 단계에는 Amazon Keyspace에 액세스할 수 있는 AWS Identity and Access Management (IAM) 보안 주체 가입 AWS 및 생성이 포함됩니다. IAM 사용자의 이름과 액세스 키는 이 자습서의 뒷부분에서 필요하므로 기록해 둡니다.

이 자습서의 뒷부분에서 VPC 엔드포인트를 사용하여 연결을 테스트할 `myKeyspace` 이름과 하나 이상의 테이블을 포함하는 키스페이스를 생성합니다. 자세한 지침은 시작하기에서 확인할 수 있습니다.

필수 단계를 완료한 후 1단계: Amazon EC2 인스턴스 시작로 진행합니다.

1단계: Amazon EC2 인스턴스 시작

이 단계에서는 기본 Amazon VPC에서 Amazon EC2 인스턴스를 시작합니다. 그런 다음 Amazon Keyspaces에 대해 VPC 엔드포인트를 생성하고 사용할 수 있습니다.

Amazon EC2 인스턴스를 시작하려면

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 인스턴스 시작을 선택하고 다음을 수행합니다.

   EC2 콘솔 대시보드의 인스턴스 시작 상자에서 인스턴스 시작을 선택한 다음 표시되는 옵션에서 인스턴스 시작을 선택합니다.

   이름 및 태그 아래의 이름에 인스턴스를 설명하는 이름을 입력합니다.

   애플리케이션 및 OS 이미지(Amazon Machine Image)에서:

   - 빠른 시작을 선택한 다음 Ubuntu를 선택합니다. 인스턴스의 운영 체제(OS)입니다.
   - Amazon Machine Image(AMI)에서 프리 티어 사용 가능으로 표시된 기본 이미지를 사용할 수 있습니다. Amazon Machine Image(AMI)는 기본 구성으로서 인스턴스의 템플릿 역할을 합니다.

   인스턴스 유형에서:

   - 인스턴스 유형 목록에서 기본적으로 선택된 t2.micro 인스턴스 유형을 선택합니다.

   키 페어(로그인)에서 키 페어 이름에 대해 이 자습서에 사용할 수 있는 다음 옵션 중 하나를 선택합니다.

   - Amazon EC2 키 페어가 없는 경우 새 키 페어 생성을 선택하고 지침을 따릅니다. 프라이빗 키 파일(.pem 파일)을 다운로드하라는 메시지가 표시됩니다. 나중에 Amazon EC2 인스턴스에 로그인할 때 이 파일이 필요하므로 파일 경로를 기록해 둡니다.
   - Amazon EC2 키 페어가 이미 있으면 키 페어 선택으로 이동하고 목록에서 해당 키 페어를 선택합니다. Amazon EC2 인스턴스에 로그인하려면 프라이빗 키 파일(.pem 파일)이 있어야 합니다.

   네트워크 설정에서:

   - 편집을 선택합니다.
   - 기존 보안 그룹 선택을 선택합니다.

- 보안 그룹 목록에서 기본값을 선택합니다. 이것이 VPC의 기본 보안 그룹입니다.

요약으로 계속 진행합니다.

- 요약 패널에서 인스턴스 구성 요약을 검토합니다. 준비가 완료되면 인스턴스 시작을 선택합니다.

3. 새 Amazon EC2 인스턴스의 완료 화면에서 인스턴스에 연결 타일을 선택합니다. 다음 화면에는 새 인스턴스에 연결하는 데 필요한 정보와 필요한 단계가 표시됩니다. 다음 정보를 기록해 둡니다.

   - 키 파일을 보호하기 위한 샘플 명령
   - 연결 문자열
   - 퍼블릭 IPv4 DNS 이름

   이 페이지의 정보를 기록한 후 이 자습서(2단계: Amazon EC2 인스턴스 구성)의 다음 단계를 계속할 수 있습니다.

> **Note**
>
> Amazon EC2 인스턴스가 사용 가능한 상태가 되는 데 몇 분 정도 걸립니다. 다음 단계로 이동하기 전에 인스턴스 상태가 `running`이고 모든 상태 확인이 통과되었는지 확인하십시오.

2단계: Amazon EC2 인스턴스 구성

Amazon EC2 인스턴스를 사용할 수 있는 경우 해당 인스턴스에 로그인하고 첫 사용이 가능한 상태로 준비할 수 있습니다.

> **Note**
>
> 다음 단계에서는 사용자가 Linux를 실행하는 컴퓨터에서 Amazon EC2 인스턴스로 연결되어 있다고 가정합니다. 다른 연결 방법은 Amazon EC2 사용 설명서의 Linux 인스턴스에 연결을 참조하십시오.

Amazon EC2 인스턴스를 구성하려면

1. Amazon EC2 인스턴스에 대한 인바운드 SSH 트래픽을 승인해야 합니다. 이를 위해 새로운 EC2 보안 그룹을 생성하여 EC2 인스턴스에 할당합니다.

   a. 탐색 창에서 보안 그룹을 선택합니다.

   b. 보안 그룹 생성을 선택합니다. 보안 그룹 생성 창에서 다음을 수행합니다.

      - 보안 그룹 이름 - 보안 그룹의 이름을 입력합니다. 예: `my-ssh-access`
      - 설명 - 보안 그룹에 대한 간단한 설명을 입력합니다.
      - VPC – 기본 VPC를 선택합니다.
      - 보안 그룹 규칙 섹션에서 규칙 추가를 선택하고 다음을 수행합니다.
        - 유형 - SSH를 선택합니다.
        - 소스 - 내 IP를 선택합니다.
        - 규칙 추가를 선택합니다.

      페이지 하단에서 구성 설정을 확인하고 보안 그룹 생성을 선택합니다.

   c. 탐색 창에서 인스턴스를 선택합니다.

   d. 1단계: Amazon EC2 인스턴스 시작에서 시작한 Amazon EC2 인스턴스를 선택합니다.

   e. 작업을 선택하고 보안을 선택한 다음 보안 그룹 변경을 선택합니다.

   f. 보안 그룹 변경에서 이 절차에서 앞서 생성한 보안 그룹을 선택합니다(예: `my-ssh-access`). 기존 `default` 보안 그룹도 선택되어야 합니다. 구성 설정을 확인하고 보안 그룹 할당을 선택합니다.

2. 다음 명령을 사용하여 프라이빗 키 파일에 액세스하지 못하도록 보호합니다. 이 단계를 건너뛰면 연결이 실패합니다.

```
chmod 400 path_to_file/my-keypair.pem
```

3. `ssh` 명령을 사용하여 Amazon EC2 인스턴스에 로그인합니다(아래 예 참조).

```
ssh -i path_to_file/my-keypair.pem ubuntu@public-dns-name
```

   프라이빗 키 파일(.pem 파일)과 인스턴스의 퍼블릭 DNS 이름을 지정해야 합니다. (1단계: Amazon EC2 인스턴스 시작 섹션을 참조하세요.)

   로그인 ID는 `ubuntu`입니다. 암호는 필요하지 않습니다.

Amazon EC2 인스턴스 연결 허용에 대한 자세한 내용 및 AWS CLI 지침은 Amazon EC2 사용 설명서의 Linux 인스턴스용 인바운드 트래픽 승인을 참조하십시오.

4. 최신 버전의 AWS Command Line Interface를 다운로드하여 설치합니다.

   a. unzip을 설치합니다.

   ```
   sudo apt install unzip
   ```

   b. AWS CLI를 사용하여 zip 파일을 다운로드합니다.

   ```
   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o
    "awscliv2.zip"
   ```

   c. 파일 압축을 폽니다.

   ```
   unzip awscliv2.zip
   ```

   d. 를 설치하십시오. AWS CLI

   ```
   sudo ./aws/install
   ```

   e. AWS CLI 설치 버전을 확인합니다.

   ```
   aws --version
   ```

   출력은 다음과 같아야 합니다.

   ```
   aws-cli/2.9.19 Python/3.9.11 Linux/5.15.0-1028-aws exe/x86_64.ubuntu.22 prompt/
   off
   ```

5. 다음 예와 같이 AWS 자격 증명을 구성합니다. 메시지가 표시되면 AWS 액세스 키 ID, 비밀 키, 기본 지역 이름을 입력합니다.

   ```
   aws configure

   AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
   AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
   Default region name [None]: us-east-1
   Default output format [None]:
   ```

6. VPC 엔드포인트가 올바르게 구성되었는지 확인하려면 Amazon Keyspaces에 cqlsh 연결을 사용해야 합니다. 에서 로컬 환경 또는 Amazon Keyspaces CQL 편집기를 사용하는 경우 VPC 엔드포인트 대신 퍼블릭 엔드포인트를 통해 연결이 자동으로 진행됩니다. AWS Management Console 이 자습서에서 VPC 엔드포인트 연결을 테스트하는 데 cqlsh를 사용하려면 cqlsh 사용하여 Amazon Keyspace에 접속의 설정 지침을 완료합니다.

이제 Amazon Keyspaces에 대한 VPC 엔드포인트를 생성할 준비가 되었습니다.

3단계: Amazon Keyspaces에 대한 VPC 엔드포인트 생성

이 단계에서는 AWS CLI를 사용하여 Amazon Keyspaces에 대한 VPC 엔드포인트를 생성합니다. VPC 콘솔을 사용하여 VPC 엔드포인트를 생성하려면 AWS PrivateLink 가이드의 VPC 엔드포인트 생성 지침을 따를 수 있습니다. 서비스 이름을 필터링할 때 **Cassandra**을 입력합니다.

를 사용하여 VPC 엔드포인트를 만들려면 AWS CLI

1. 시작하기 전에 퍼블릭 엔드포인트를 사용하여 Amazon Keyspaces와 통신할 수 있는지 확인합니다.

```
aws keyspaces list-tables --keyspace-name 'myKeyspace'
```

출력에는 지정된 키스페이스에 포함된 Amazon Keyspaces 테이블 목록이 표시됩니다. 보유 중인 테이블이 없는 경우 목록이 비어있습니다.

```
{
    "tables": [
        {
            "keyspaceName": "myKeyspace",
            "tableName": "myTable1",
            "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
catalog/table/myTable1"
        },
        {
            "keyspaceName": "myKeyspace",
            "tableName": "myTable2",
            "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
catalog/table/myTable2"
        }
    ]
}
```

2. Amazon Keyspaces가 현재 리전에서 VPC 엔드포인트를 생성하는 데 사용할 수 있는 서비스인지 확인하십시오. AWS (명령은 예제 출력에 이어 굵은 텍스트로 표시됩니다.)

```
aws ec2 describe-vpc-endpoint-services

{
    "ServiceNames": [
        "com.amazonaws.us-east-1.cassandra",
        "com.amazonaws.us-east-1.cassandra-fips"
    ]
}
```

예제 출력에서 Amazon Keyspaces는 사용할 수 있는 서비스 중 하나이므로 이에 대한 VPC 엔트포인트를 생성할 수 있습니다.

3. VPC 식별자 확인

```
aws ec2 describe-vpcs

{
    "Vpcs": [
        {
            "VpcId": "vpc-a1234bcd",
            "InstanceTenancy": "default",
            "State": "available",
            "DhcpOptionsId": "dopt-8454b7e1",
            "CidrBlock": "111.31.0.0/16",
            "IsDefault": true
        }
    ]
}
```

예제 출력에서 VPC ID는 vpc-a1234bcd입니다.

4. 필터를 사용하여 VPC의 서브넷에 대한 세부 정보를 수집합니다.

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=vpc-a1234bcd"

{
     {
    "Subnets":[
       {
          "AvailabilityZone":"us-east-1a",
```

```
            "AvailabilityZoneId":"use2-az1",
            "AvailableIpAddressCount":4085,
            "CidrBlock":"111.31.0.0/20",
            "DefaultForAz":true,
            "MapPublicIpOnLaunch":true,
            "MapCustomerOwnedIpOnLaunch":false,
            "State":"available",
            "SubnetId":"subnet-920aacf9",
            "VpcId":"vpc-a1234bcd",
            "OwnerId":"111122223333",
            "AssignIpv6AddressOnCreation":false,
            "Ipv6CidrBlockAssociationSet":[

            ],
            "SubnetArn":"arn:aws:ec2:us-east-1:111122223333:subnet/subnet-920aacf9",
            "EnableDns64":false,
            "Ipv6Native":false,
            "PrivateDnsNameOptionsOnLaunch":{
               "HostnameType":"ip-name",
               "EnableResourceNameDnsARecord":false,
               "EnableResourceNameDnsAAAARecord":false
            }
         },
         {
            "AvailabilityZone":"us-east-1c",
            "AvailabilityZoneId":"use2-az3",
            "AvailableIpAddressCount":4085,
            "CidrBlock":"111.31.32.0/20",
            "DefaultForAz":true,
            "MapPublicIpOnLaunch":true,
            "MapCustomerOwnedIpOnLaunch":false,
            "State":"available",
            "SubnetId":"subnet-4c713600",
            "VpcId":"vpc-a1234bcd",
            "OwnerId":"111122223333",
            "AssignIpv6AddressOnCreation":false,
            "Ipv6CidrBlockAssociationSet":[

            ],
            "SubnetArn":"arn:aws:ec2:us-east-1:111122223333:subnet/subnet-4c713600",
            "EnableDns64":false,
            "Ipv6Native":false,
            "PrivateDnsNameOptionsOnLaunch":{
               "HostnameType":"ip-name",
```

```
            "EnableResourceNameDnsARecord":false,
            "EnableResourceNameDnsAAAARecord":false
         }
      },
      {
         "AvailabilityZone":"us-east-1b",
         "AvailabilityZoneId":"use2-az2",
         "AvailableIpAddressCount":4086,
         "CidrBlock":"111.31.16.0/20",
         "DefaultForAz":true,
         "MapPublicIpOnLaunch":true,


       }
    ]
}
```

예제 출력에는 subnet-920aacf9 및 subnet-4c713600의 사용 가능한 두 개의 서브넷 ID가 있습니다.

5. VPC 엔드포인트를 생성합니다. --vpc-id 파라미터에 대해 이전 단계에서 VPC ID를 지정합니다. --subnet-id 파라미터에 대해 이전 단계에서 서브넷 ID를 지정합니다. --vpc-endpoint-type 파라미터를 사용하여 엔드포인트를 인터페이스로 정의합니다. 명령에 대한 자세한 내용은 AWS CLI 명령 참조의 create-vpc-endpoint 섹션을 참조하세요.

```
aws ec2 create-vpc-endpoint --vpc-endpoint-type Interface --vpc-id vpc-a1234bcd
 --service-name com.amazonaws.us-east-1.cassandra --subnet-id subnet-920aacf9
 subnet-4c713600

{
    "VpcEndpoint": {
        "VpcEndpointId": "vpce-000ab1cdef23456789",
        "VpcEndpointType": "Interface",
        "VpcId": "vpc-a1234bcd",
        "ServiceName": "com.amazonaws.us-east-1.cassandra",
        "State": "pending",
        "RouteTableIds": [],
        "SubnetIds": [
            "subnet-920aacf9",
            "subnet-4c713600"
        ],
        "Groups": [
            {
```

```
                "GroupId": "sg-ac1b0e8d",
                "GroupName": "default"
            }
        ],
        "IpAddressType": "ipv4",
        "DnsOptions": {
            "DnsRecordIpType": "ipv4"
        },
        "PrivateDnsEnabled": true,
        "RequesterManaged": false,
        "NetworkInterfaceIds": [
            "eni-043c30c78196ad82e",
            "eni-06ce37e3fd878d9fa"
        ],
        "DnsEntries": [
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz.cassandra.us-
east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz-us-
east-1a.cassandra.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz-us-
east-1c.cassandra.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz-us-
east-1b.cassandra.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "vpce-000ab1cdef23456789-m2b22rtz-us-
east-1d.cassandra.us-east-1.vpce.amazonaws.com",
                "HostedZoneId": "Z7HUB22UULQXV"
            },
            {
                "DnsName": "cassandra.us-east-1.amazonaws.com",
                "HostedZoneId": "ZONEIDPENDING"
            }
```

```
        ],
        "CreationTimestamp": "2023-01-27T16:12:36.834000+00:00",
        "OwnerId": "111122223333"
    }
}

}
```

4단계: VPC 엔드포인트 연결을 위한 권한 구성

이 단계의 절차는 Amazon Keyspaces와 함께 VPC 엔드포인트를 사용하기 위한 규칙 및 권한을 구성하는 방법을 보여 줍니다.

TCP 인바운드 트래픽을 허용하도록 새 엔드포인트에 대한 인바운드 규칙을 구성하려면

1. Amazon VPC 콘솔의 왼쪽 패널에서 엔드포인트를 선택하고 이전 단계에서 생성한 엔드포인트를 선택합니다.
2. 보안 그룹을 선택한 다음 이 엔드포인트와 연결된 보안 그룹을 선택합니다.
3. 인바운드 규칙을 선택한 다음 인바운드 규칙 편집을 선택합니다.
4. 유형이 CQLSH/CASSANDRA인 인바운드 규칙을 추가하십시오. 이렇게 하면 포트 범위가 자동으로 9142로 설정됩니다.
5. 새 인바운드 규칙을 저장하려면 규칙 저장을 선택합니다.

IAM 사용자 권한을 구성하려면

1. Amazon Keyspaces에 연결하는 데 사용된 IAM 사용자에게 적절한 권한이 있는지 확인합니다. AWS Identity and Access Management (IAM) 에서는 AWS 관리형 정책을 사용하여 IAM 사용자에게 Amazon `AmazonKeyspacesReadOnlyAccess` Keyspace에 대한 읽기 액세스 권한을 부여할 수 있습니다.

   a. AWS Management Console 로그인하고 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.

   b. IAM 콘솔 대시보드에서 Users(사용자)를 선택한 후 목록에서 해당 IAM 사용자를 선택합니다.

   c. 요약 페이지에서 권한 추가를 선택합니다.

   d. 기존 정책 직접 첨부를 선택합니다.

e. 정책 목록에서 [AmazonKeyspacesReadOnlyAccess] 를 선택한 후 [다음: 검토] 를 선택합니다.

f. 권한 추가를 선택합니다.

2. VPC 엔드포인트를 통해 Amazon Keyspaces에 액세스할 수 있는지 확인합니다.

```
aws keyspaces list-tables --keyspace-name 'my_Keyspace'
```

원하는 경우 Amazon Keyspace에 대한 몇 가지 다른 AWS CLI 명령을 시도해 볼 수 있습니다. 자세한 내용은 AWS CLI 명령 참조를 참조하세요.

> **Note**
>
> 다음 정책에 나와 있는 것처럼 IAM 사용자 또는 역할이 Amazon Keyspaces에 액세스하는 데 필요한 최소 권한은 시스템 테이블에 대한 읽기 권한입니다. 정책 기반 권한에 대한 자세한 내용은 the section called “ID 기반 정책 예제” 섹션을 참조하세요.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:555555555555:/keyspace/system*"
         ]
      }
   ]
}
```

3. IAM 사용자에게 VPC가 있는 Amazon EC2 인스턴스에 대한 읽기 액세스 권한을 부여합니다.

VPC 엔드포인트와 함께 Amazon Keyspaces를 사용하는 경우 Amazon Keyspaces에 액세스하는 IAM 사용자 또는 역할에 Amazon EC2 인스턴스와 VPC에 엔드포인트 및 네트워크 인터페이스 데이터를 수집할 수 있는 읽기 전용 권한을 부여해야 합니다. Amazon Keyspaces는 이 정보를 `system.peers` 테이블에 저장하고 이 정보를 사용하여 연결을 관리합니다.

> **Note**
>
> 관리형 정책 `AmazonKeyspacesReadOnlyAccess_v2` 및 `AmazonKeyspacesFullAccess`에는 Amazon Keyspaces가 Amazon EC2 인스턴스에 액세스하여 사용 가능한 인터페이스 VPC 엔드포인트에 대한 정보를 읽을 수 있도록 하는 데 필요한 권한이 포함되어 있습니다.

a. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.

b. IAM 콘솔 대시보드에서 정책을 선택합니다.

c. 정책 생성을 선택한 다음 JSON 탭을 선택합니다.

d. 다음 정책을 복사하고 다음: 태그를 선택합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListVPCEndpoints",
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcEndpoints"
         ],
         "Resource": "*"
      }
   ]
}
```

e. 다음: 검토를 선택하고 정책에 대한 이름 `keyspacesVPCendpoint`를 입력한 후 정책 생성을 선택합니다.

f. IAM 콘솔 대시보드에서 Users(사용자)를 선택한 후 목록에서 해당 IAM 사용자를 선택합니다.

g. 요약 페이지에서 권한 추가를 선택합니다.

h. 기존 정책 직접 첨부를 선택합니다.

i. 정책 목록에서 keyspacesVPCendpoint를 선택한 다음 다음: 검토를 선택합니다.

j. 권한 추가를 선택합니다.

4. Amazon Keyspaces `system.peers` 테이블이 VPC 정보로 업데이트되고 있는지 확인하려면 `cqlsh`를 사용하여 Amazon EC2 인스턴스에서 다음 쿼리를 실행합니다. 2단계에서 Amazon EC2 인스턴스에 아직 `cqlsh`를 설치하지 않은 경우 the section called “cqlsh-expansion 사용”의 지침을 따릅니다.

```
SELECT peer FROM system.peers;
```

출력은 해당 지역의 VPC 및 서브넷 설정에 따라 프라이빗 IP 주소가 있는 노드를 반환합니다. AWS

```
peer
---------------
112.11.22.123
112.11.22.124
112.11.22.125
```

> **Note**
>
> VPC 엔드포인트가 올바르게 구성되었는지 확인하려면 Amazon Keyspaces에 `cqlsh` 연결을 사용해야 합니다. AWS Management Console에서 로컬 환경 또는 Amazon Keyspaces CQL 편집기를 사용하는 경우 VPC 엔드포인트 대신 퍼블릭 엔드포인트를 통해 연결이 자동으로 진행됩니다. IP 주소가 9개인 경우 Amazon Keyspaces가 퍼블릭 엔드포인트 연결을 위해 `system.peers` 테이블에 자동으로 쓰는 항목입니다.

5단계: 다음을 사용하여 모니터링을 구성합니다. CloudWatch

이 단계에서는 Amazon을 사용하여 Amazon CloudWatch Keyspace에 대한 VPC 엔드포인트 연결을 모니터링하는 방법을 보여줍니다.

AWS PrivateLink 인터페이스 엔드포인트에 CloudWatch 대한 데이터 포인트를 게시합니다. 지표를 사용하여 시스템이 예상대로 수행되고 있는지 확인할 수 있습니다. 의 `AWS/PrivateLinkEndpoints` 네임스페이스에는 인터페이스 엔드포인트에 대한 메트릭이 CloudWatch 포함됩니다. 자세한 내용은 AWS PrivateLink가이드의 CloudWatch 메트릭을 참조하십시오.AWS PrivateLink

VPC 엔드포인트 CloudWatch 메트릭으로 대시보드를 만들려면

1. 에서 CloudWatch https://console.aws.amazon.com/cloudwatch/ 콘솔을 엽니다.

2. 탐색 창에서 대시보드를 선택합니다. 그런 다음 대시보드 생성을 선택합니다. 대시보드의 이름을 입력하고 생성을 선택합니다.
3. 위젯 추가에서 숫자를 선택합니다.
4. 메트릭에서 AWS/ 엔드포인트를 PrivateLink 선택합니다.
5. 엔드포인트 유형, 서비스 이름, VPC 엔드포인트 ID, VPC ID를 선택합니다.
6. 지표 `ActiveConnections` 및 `NewConnections`를 선택하고 위젯 생성을 선택합니다.
7. 대시보드를 저장합니다.

`ActiveConnections` 지표는 지난 1분 동안 엔드포인트가 수신한 동시 활성 연결 수로 정의됩니다. `NewConnections` 지표는 지난 1분 동안 엔드포인트를 통해 설정된 새 연결 수로 정의됩니다.

대시보드 생성에 대한 자세한 내용은 사용 설명서의 대시보드 생성을 참조하십시오. CloudWatch

6단계: (선택 사항) 애플리케이션의 연결 풀 크기를 구성하는 모범 사례

이 섹션에서는 애플리케이션의 쿼리 처리량 요구 사항을 기반으로 이상적인 연결 풀 크기를 결정하는 방법을 설명합니다.

Amazon Keyspaces는 TCP 연결당 초당 최대 3,000개의 CQL 쿼리를 허용합니다. 따라서 드라이버가 Amazon Keyspaces와 설정할 수 있는 연결 수에는 사실상 제한이 없습니다. 하지만 VPC 엔드포인트 연결과 함께 Amazon Keyspaces를 사용할 때는 연결 풀 크기를 애플리케이션의 요구 사항에 맞추고 사용 가능한 엔드포인트를 고려하는 것이 좋습니다.

연결 풀 크기는 클라이언트 드라이버에서 구성합니다. 예를 들어 로컬 풀 크기가 2이고 가용 영역 3개에 생성된 VPC 인터페이스 엔드포인트를 기반으로 드라이버는 쿼리를 위한 6개(제어 연결 포함 총 7개)의 연결을 설정합니다. 이 6개의 연결을 사용하면 초당 최대 18,000개의 CQL 쿼리를 지원할 수 있습니다.

애플리케이션이 초당 40,000개의 CQL 쿼리를 지원해야 하는 경우 필요한 쿼리 수를 기준으로 역방향으로 작업하여 필요한 연결 풀 크기를 결정합니다. 초당 40,000개의 CQL 쿼리를 지원하려면 로컬 풀 크기를 초당 최소 45,000개의 CQL 쿼리를 지원하는 5개 이상으로 구성해야 합니다.

네임스페이스의 `PerConnectionRequestRateExceeded` CloudWatch 지표를 사용하여 연결당 초당 최대 작업 수의 할당량을 초과하는지 모니터링할 수 있습니다. `AWS/Cassandra` `PerConnectionRequestRateExceeded` 지표는 Amazon Keyspaces에 대한 요청 중 연결당 요청 속도에 대한 할당량을 초과하는 요청 수를 보여 줍니다.

이 단계의 코드 예제는 인터페이스 VPC 엔드포인트를 사용할 때 연결 풀링을 추정하고 구성하는 방법을 보여 줍니다.

Java

Java 드라이버에서 풀당 연결 수를 구성할 수 있습니다. Java 클라이언트 드라이버 연결의 전체 예는 the section called “Cassandra Java 클라이언트 드라이버 사용” 섹션을 참조하세요.

클라이언트 드라이버가 시작되면 먼저 스키마 및 토폴로지 변경과 같은 관리 작업을 위한 제어 연결이 설정됩니다. 그런 다음 추가 연결이 생성됩니다.

다음 예제에서는 로컬 풀 크기 드라이버 구성을 2로 지정합니다. VPC 내 서브넷 3개에 걸쳐 VPC 엔드포인트를 생성하면 다음 공식과 같이 인터페이스 엔드포인트가 NewConnections 7인치가 됩니다. CloudWatch

```
NewConnections = 3 (VPC subnet endpoints created across) * 2 (pool size) + 1
 ( control connection)
```

```
datastax-java-driver {

    basic.contact-points = [ "cassandra.us-east-1.amazonaws.com:9142"]
    advanced.auth-provider{
        class = PlainTextAuthProvider
           username = "ServiceUserName"
           password = "ServicePassword"
    }
    basic.load-balancing-policy {
        local-datacenter = "us-east-1"
        slow-replica-avoidance = false
    }

    advanced.ssl-engine-factory {
        class = DefaultSslEngineFactory
        truststore-path = "./src/main/resources/cassandra_truststore.jks"
        truststore-password = "my_password"
        hostname-validation = false
      }
    advanced.connection {
        pool.local.size = 2
          }
}
```

활성 연결 수가 구성된 풀 크기(서브넷 간 집계) + 1개의 제어 연결과 일치하지 않으면 연결이 생성되지 않는 것입니다.

Node.js

Node.js 드라이버에서 풀당 연결 수를 구성할 수 있습니다. Node.js 클라이언트 드라이버 연결의 전체 예는 the section called “Cassandra Node.js 클라이언트 드라이버 사용” 섹션을 참조하세요.

다음 코드 예제의 경우 로컬 풀 크기 드라이버 구성을 1로 지정합니다. VPC 내 서브넷 4개에 걸쳐 VPC 엔드포인트를 생성하면 다음 공식과 같이 인터페이스 엔드포인트가 NewConnections 5인치가 됩니다. CloudWatch

```
NewConnections = 4 (VPC subnet endpoints created across) * 1 (pool size) + 1
 ( control connection)
```

```
const cassandra = require('cassandra-driver');
const fs = require('fs');
const types = cassandra.types;
const auth = new cassandra.auth.PlainTextAuthProvider('ServiceUserName',
 'ServicePassword');
const sslOptions1 = {
         ca: [
                    fs.readFileSync('/home/ec2-user/sf-class2-root.crt', 'utf-8')],
                    host: 'cassandra.us-east-1.amazonaws.com',
                    rejectUnauthorized: true
        };
const client = new cassandra.Client({
                   contactPoints: ['cassandra.us-east-1.amazonaws.com'],
                   localDataCenter: 'us-east-1',
                   pooling: { coreConnectionsPerHost: { [types.distance.local]:
 1 } },
                   consistency: types.consistencies.localQuorum,
                   queryOptions: { isIdempotent: true },
                   authProvider: auth,
                   sslOptions: sslOptions1,
                   protocolOptions: { port: 9142 }
        });
```

7단계: (선택 사항) 정리

이 자습서에서 생성한 리소스를 삭제하려면 다음 절차를 따르세요.

Amazon Keyspaces에 대한 VPC 엔드포인트를 제거하려면

1. Amazon EC2 인스턴스에 로그인합니다.
2. Amazon Keyspaces에 사용되는 VPC 엔드포인트 ID를 결정합니다. `grep` 파라미터를 생략하면 모든 서비스에 대한 VPC 엔드포인트 정보가 표시됩니다.

```
aws ec2 describe-vpc-endpoint-services | grep ServiceName | grep cassandra

{
    "VpcEndpoint": {
        "PolicyDocument": "{\"Version\":\"2008-10-17\",\"Statement\":[{\"Effect\":
\"Allow\",\"Principal\":\"*\",\"Action\":\"*\",\"Resource\":\"*\"}]}",
        "VpcId": "vpc-0bbc736e",
        "State": "available",
        "ServiceName": "com.amazonaws.us-east-1.cassandra",
        "RouteTableIds": [],
        "VpcEndpointId": "vpce-9b15e2f2",
        "CreationTimestamp": "2017-07-26T22:00:14Z"
    }
}
```

   예제 출력에서 VPC 엔드포인트 ID는 `vpce-9b15e2f2`입니다.

3. VPC 엔드포인트를 삭제합니다.

```
aws ec2 delete-vpc-endpoints --vpc-endpoint-ids vpce-9b15e2f2

{
    "Unsuccessful": []
}
```

   빈 어레이 `[]`는 성공을 나타냅니다(실패한 요청 없음).

Amazon EC2 인스턴스를 종료하려면

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 탐색 창에서 인스턴스를 선택합니다.
3. Amazon EC2 인스턴스를 선택합니다.
4. 작업을 선택하고 인스턴스 상태를 선택한 다음 종료를 선택합니다.

5. 확인 창에서 예, 종료합니다.를 선택합니다.

Amazon Keyspaces에 대한 크로스 계정 액세스 구성

리소스를 분리하고 다양한 환경(예: 개발 및 프로덕션)에서 사용하기 위해 별도로 AWS 계정을 생성하여 사용할 수 있습니다. 이 주제에서는 Amazon Virtual Private Cloud에서 인터페이스 VPC 엔드포인트를 통한 Amazon Keyspaces의 크로스 계정 액세스를 안내합니다. IAM 크로스 계정 액세스 구성에 대한 자세한 내용은 IAM 사용 설명서의 개별 개발 계정과 프로덕션 계정을 사용하는 예제 시나리오 섹션을 참조하세요.

Amazon Keyspaces와 프라이빗 VPC 엔드포인트에 대한 자세한 내용은 the section called “인터페이스 VPC 엔드포인트 사용” 섹션을 참조하세요.

주제

- 공유 VPC에서 Amazon Keyspaces에 대한 크로스 계정 액세스 구성
- 공유 VPC 없이 Amazon Keyspaces에 대한 크로스 계정 액세스 구성

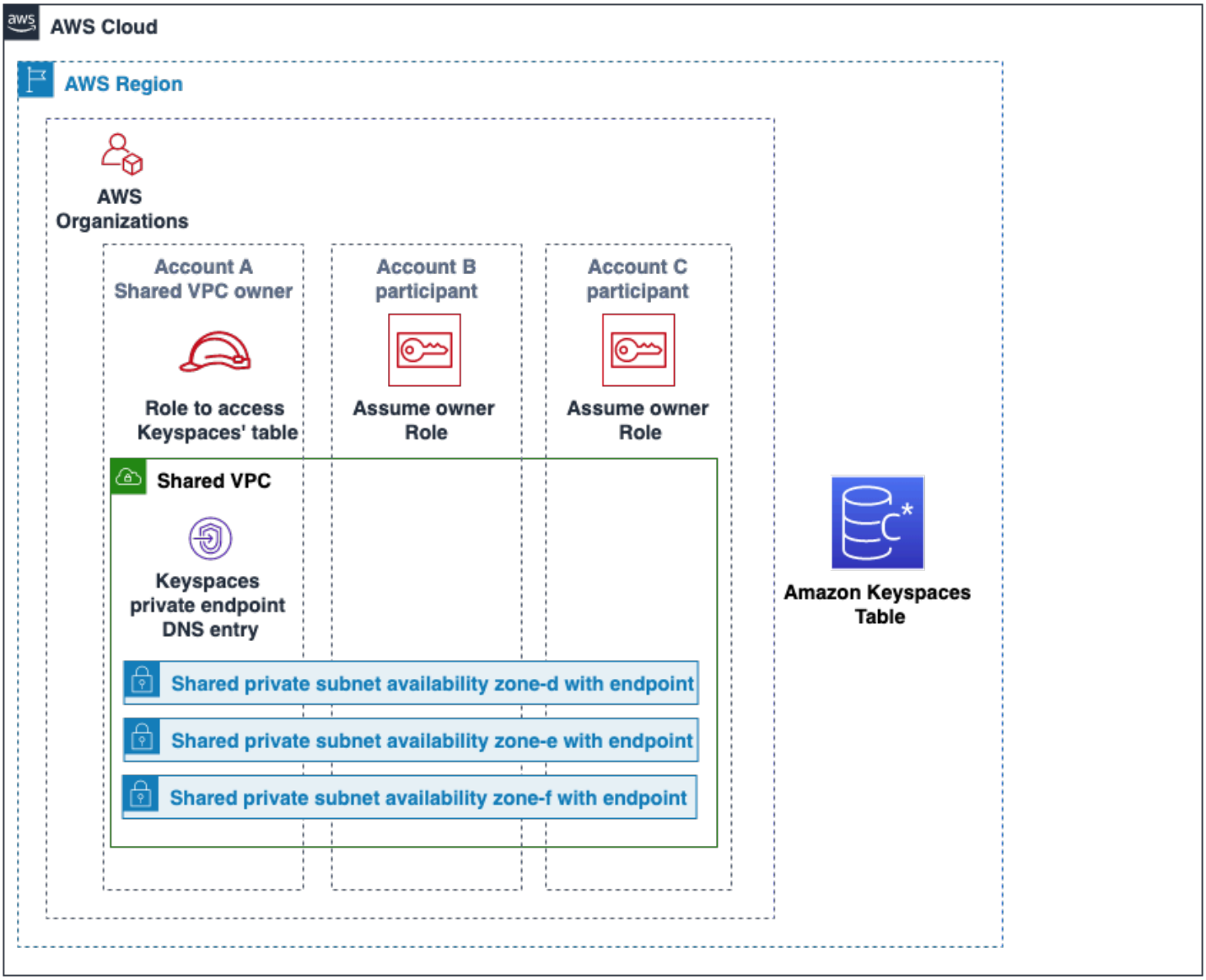
공유 VPC에서 Amazon Keyspaces에 대한 크로스 계정 액세스 구성

리소스를 애플리케이션과 분리하기 위해 다른 AWS 계정를 만들 수 있습니다. 예를 들어 Amazon Keyspaces 테이블에 대한 계정 하나, 개발 환경의 애플리케이션 계정 하나, 프로덕션 환경의 애플리케이션 계정 하나를 생성할 수 있습니다. 이 주제에서는 공유 VPC의 인터페이스 VPC 엔드포인트를 사용하여 Amazon Keyspaces에 대한 크로스 계정 액세스를 설정하는 데 필요한 구성 단계를 안내합니다.

Amazon Keyspaces의 VPC 엔드포인트를 구성하는 방법에 대한 자세한 단계는 the section called “3단계: Amazon Keyspaces에 대한 VPC 엔드포인트 생성” 섹션을 참조하세요.

이 예제에서는 공유 VPC에서 다음 세 개의 계정을 사용합니다.

- `Account A` — 이 계정에는 VPC 엔드포인트, VPC 서브넷, Amazon Keyspaces 테이블을 비롯한 인프라가 포함되어 있습니다.
- `Account B` — 이 계정에는 `Account A`에서 Amazon Keyspaces 테이블에 연결해야 하는 개발 환경의 애플리케이션이 포함되어 있습니다.
- `Account C` — 이 계정에는 `Account A`에서 Amazon Keyspaces 테이블에 연결해야 하는 프로덕션 환경의 애플리케이션이 포함되어 있습니다.



Account A는 Account B 및 Account C이 액세스해야 하는 리소스가 포함된 계정이므로, Account A는 신뢰하는 계정입니다. Account B 및 Account C는 Account A에서 리소스에 액세스해야 하는 보안 주체가 있는 계정이고 Account B 및 Account C은 신뢰 받는 계정입니다. 신뢰하는 계정은 IAM 역할을 공유하여 신뢰 받는 계정에 권한을 부여합니다. 다음 절차에서는 Account A에서 필요한 구성 단계를 간략하게 설명합니다.

Account A에 대한 구성

1. AWS Resource Access Manager을 사용하여 서브넷에 대한 리소스 공유를 만들고 Account B 및 Account C와 프라이빗 서브넷을 공유합니다.

   Account B 및 Account C는 공유된 서브넷에서 리소스를 보고 생성할 수 있습니다.

2. AWS PrivateLink에 의해 구동되는 Amazon Keyspaces 프라이빗 VPC 엔드포인트를 생성합니다. 이렇게 하면 Amazon Keyspaces 서비스 엔드포인트에 대한 공유 서브넷 전체와 DNS 항목에 걸쳐 공유되는 여러 엔드포인트가 생성됩니다.
3. Amazon Keyspaces 키스페이스 및 테이블을 생성합니다.
4. 다음 정책 예제와 같이 Amazon Keyspaces 테이블에 대한 전체 액세스 권한, Amazon Keyspaces 시스템 테이블에 대한 읽기 액세스 권한을 갖고, Amazon EC2 VPC 리소스를 설명할 수 있는 IAM 역할을 생성합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CrossAccountAccess",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeNetworkInterfaces",
                "ec2:DescribeVpcEndpoints",
                "cassandra:*"
            ],
            "Resource": "*"
        }
    ]
}
```

5. 다음 예와 같이 `Account B` 및 `Account C`가 신뢰할 수 있는 계정으로 간주할 수 있는 IAM 역할 신뢰 정책을 구성합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111111111111:root"
            },
            "Action": "sts:AssumeRole",
            "Condition": {}
        }
    ]
}
```

크로스 계정 IAM 정책에 대한 자세한 내용은 IAM 사용 설명서의 크로스 계정 정책을 참조하세요.

Account B 및 Account C에서의 구성

1. Account B및 Account C에서 새 역할을 만들고 Account A에서 생성된 공유 역할을 주체가 맡도록 허용하는 다음 정책을 연결합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "ec2.amazonaws.com"
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

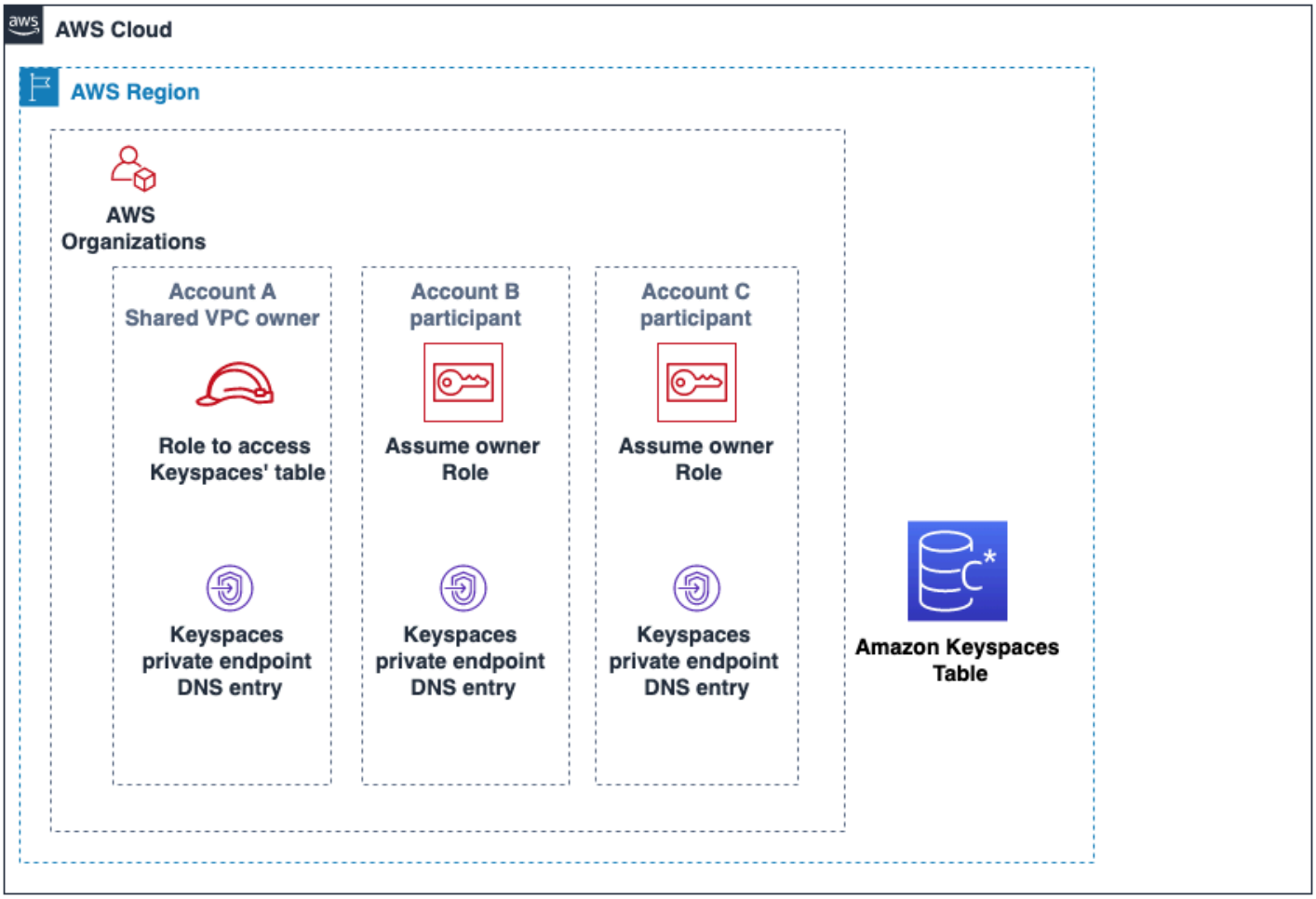
   주체가 공유 역할을 맡도록 허용하는 것은 AWS Security Token Service(AWS STS)의 AssumeRole API를 사용하여 구현됩니다. 자세한 내용은 IAM 사용자 설명서의 소유한 다른 AWS 계정의 IAM 사용자에 액세스 권한 제공을 참조하세요.

2. Account B 및 Account C에서는 SIGV4 인증 플러그인을 활용하는 애플리케이션을 생성할 수 있습니다. 그러면 애플리케이션이 공유 역할을 맡아 공유 VPC의 VPC 엔드포인트를 통해 Account A에 위치한 Amazon Keyspaces 테이블에 연결할 수 있습니다. SIGV4에 대한 자세한 내용은 the section called “보안 인증 정보 생성” 섹션을 참조하세요.

공유 VPC 없이 Amazon Keyspaces에 대한 크로스 계정 액세스 구성

Amazon Keyspaces 테이블과 프라이빗 VPC 엔드포인트를 서로 다른 계정에서 소유하고 있지만 VPC를 공유하지는 않는 경우에도 애플리케이션은 VPC 엔드포인트를 사용하여 크로스 계정을 연결할 수 있습니다. 계정이 VPC 엔드포인트를 공유하지 않으므로 Account A, Account B, Account C에 자체 VPC 엔드포인트가 필요합니다. Cassandra 클라이언트 드라이버에게 Amazon Keyspaces는 다중 노드 클러스터가 아닌 단일 노드처럼 보입니다. 연결 시 클라이언트 드라이버는 DNS 서버에 도달하여 계정의 VPC에서 사용 가능한 엔드포인트 중 하나를 반환합니다.

또한 퍼블릭 엔드포인트를 사용하거나 각 계정에 프라이빗 VPC 엔드포인트를 배포하여 공유 VPC 엔드포인트 없이 여러 계정의 Amazon Keyspaces 테이블에 액세스할 수 있습니다. 공유 VPC를 사용하지 않는 경우 각 계정에는 자체 VPC 엔드포인트가 필요합니다. 이 예제에서는 `Account A`, `Account B`, `Account C`에 자체 VPC 엔드포인트가 있어야 `Account A`의 테이블에 액세스할 수 있습니다. 이 구성에서 VPC 엔드포인트를 사용하는 경우 Amazon Keyspaces는 Cassandra 클라이언트 드라이버에 다중 노드 클러스터 대신 단일 노드 클러스터로 나타납니다. 연결 시 클라이언트 드라이버는 DNS 서버에 도달하여 계정의 VPC에서 사용 가능한 엔드포인트 중 하나를 반환합니다. 하지만 클라이언트 드라이버는 `system.peers` 테이블에 액세스하여 추가 엔드포인트를 검색할 수 없습니다. 사용 가능한 호스트가 적기 때문에 드라이버의 연결 수가 줄어듭니다. 이를 조정하려면 드라이버의 연결 풀 설정을 3배 늘립니다.



Amazon Keyspaces(Apache Cassandra용) 시작

이 자습서는 Apache Cassandra와 Amazon Keyspaces(Apache Cassandra용)를 처음 사용하는 경우 유용합니다. 이 자습서에서는 Amazon Keyspaces를 성공적으로 사용하는 데 필요한 모든 프로그램과 드라이버를 설치합니다.

다양한 Cassandra 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 프로그래밍 방식으로 연결하는 방법에 대한 자습서는 the section called “Cassandra 클라이언트 드라이버 사용” 섹션을 참조하세요.

주제

자습서 사전 조건 및 고려 사항

이 자습서를 시작하기 전에 의 AWS 설정 지침을 따르십시오Amazon Keyspaces(Apache Cassandra용) 액세스. 이러한 단계에는 Amazon AWS Keyspace에 액세스할 수 있는 AWS Identity and Access Management (IAM) 사용자 가입 및 생성이 포함됩니다.

또한 `cqlsh` 또는 Apache 2.0 라이선스 Cassandra 클라이언트 드라이버를 사용하여 자습서를 완료하는 경우 `cqlsh` 사용하여 Amazon Keyspace에 접속의 설정 지침을 완료합니다.

필수 단계를 완료한 후 자습서 1단계: Amazon Keyspaces에서 키스페이스 및 테이블 생성로 진행합니다.

자습서 1단계: Amazon Keyspaces에서 키스페이스 및 테이블 생성

이 섹션에서는 콘솔을 사용하여 키스페이스를 만들고 여기에 테이블을 추가합니다.

> **Note**
>
> 시작하기 전에 자습서 사전 조건을 모두 구성해야 합니다.

주제

키스페이스 생성

키스페이스는 하나 이상의 애플리케이션과 관련된 관련 테이블을 그룹화합니다. 키스페이스는 하나 이상의 테이블을 포함하며 포함된 모든 테이블의 복제 전략을 정의합니다. 키스페이스에 대한 자세한 내용은 다음 주제를 참조하세요.

- 키스페이스 사용: the section called “키스페이스 생성”
- 데이터 정의 언어(DDL) 문: Keyspaces
- Amazon Keyspaces(Apache Cassandra용)에 대한 할당량

키스페이스를 생성할 때 키스페이스 이름을 지정해야 합니다.

> **Note**
>
> 키스페이스의 복제 전략은 `SingleRegionStrategy`여야 합니다.
> `SingleRegionStrategy`는 해당 AWS 리전에서 가용 영역 3개에 데이터를 복제합니다.

콘솔 사용

콘솔을 사용하여 키스페이스를 생성하려면

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 탐색 창에서 Keyspaces를 선택합니다.
3. 키스페이스 생성을 선택합니다.
4. 키스페이스 이름 상자에 키스페이스 이름으로 **`myGSGKeyspace`**를 입력합니다.

   이름 제약 조건:

   - 비워 둘 수 없습니다.
   - 허용되는 문자: 영숫자 및 밑줄(_)

- 최대 길이는 48자입니다.

5. 키스페이스를 만들려면 키스페이스 생성을 선택합니다.
6. 다음을 수행하여 myGSGKeyspace 키스페이스가 생성되었는지 확인합니다.
   a. 탐색 창에서 Keyspaces를 선택합니다.
   b. 키스페이스 목록에서 myGSGKeyspace 키스페이스를 찾습니다.

CQL 사용

다음 절차는 CQL을 사용하여 키스페이스를 생성합니다.

CQL을 사용하여 키스페이스를 만들려면

1. 명령 쉘을 열고 다음을 입력합니다.

   cqlsh

2. 다음 CQL 명령을 사용하여 키스페이스를 생성합니다.

```
CREATE KEYSPACE IF NOT EXISTS "myGSGKeyspace"
   WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

   SingleRegionStrategy복제 요소 3을 사용하고 해당 지역의 3개 가용 AWS 영역에 데이터를 복제합니다.

> **Note**
>
> Amazon Keyspaces는 따옴표로 입력하지 않는 한 기본적으로 모든 입력을 소문자로 사용합니다. 이 경우에는 "myGSGKeyspace"를 참고합니다.

3. 키스페이스가 생성되었는지 확인합니다.

```
SELECT * from system_schema.keyspaces ;
```

   키스페이스가 나열되어야 합니다.

테이블 생성

테이블은 데이터를 구성하고 저장하는 곳입니다. 테이블의 프라이머리 키는 테이블에서 데이터를 분할하는 방법을 결정합니다. 프라이머리 키는 필수 파티션 키와 하나 이상의 선택적 클러스터링 열로 구성됩니다. 프라이머리 키를 구성하는 결합된 값은 테이블의 모든 데이터에서 고유해야 합니다. 테이블에 대한 자세한 내용은 다음 주제를 참조하세요.

- 테이블 작업: the section called “테이블 생성”
- DDL 문: 표
- 테이블 리소스 관리: 서버리스 리소스 관리
- 테이블 리소스 사용률 모니터링: the section called “를 통한 모니터링 CloudWatch”
- Amazon Keyspaces(Apache Cassandra용)에 대한 할당량

테이블을 생성할 때 다음을 지정합니다.

- 테이블의 이름
- 테이블에 있는 각 열의 이름 및 데이터 유형
- 테이블의 프라이머리 키
  - 파티션 키 - 필수
  - 클러스터링 컬럼 — 선택 사항

다음 절차를 사용하여 지정된 열, 데이터 유형, 파티션 키 및 클러스터링 열을 포함하는 테이블을 생성합니다.

콘솔 사용

다음 절차는 이러한 열과 데이터 유형을 포함하는 `employees_tbl` 테이블을 만듭니다.

```
ID            text
name          text
region        text
division      text
project       text
role          text
pay_scale     int
vacation_hrs  float
manager_id    text
```

콘솔을 사용하여 테이블을 생성하려면

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 탐색 창에서 Keyspaces를 선택합니다.
3. 이 테이블을 만들려는 키스페이스로 `myGSGKeyspace`를 선택합니다.
4. 테이블 생성을 선택합니다.
5. 테이블 이름 상자에 테이블 이름으로 **`employees_tbl`**을 입력합니다.

   이름 제약 조건:

   - 비워 둘 수 없습니다.
   - 허용되는 문자: 영숫자 및 밑줄(_)
   - 최대 길이는 48자입니다.
6. 열 섹션에서 이 테이블에 추가할 각 열에 대해 다음 단계를 반복합니다.

   다음 열과 데이터 유형을 추가합니다.

   ```
   id            text
   name          text
   region        text
   division      text
   project       text
   role          text
   pay_scale     int
   vacation_hrs  float
   manager_id    text
   ```

   a. 이름 - 열에 이름을 입력합니다.

      이름 제약 조건:

      - 비워 둘 수 없습니다.
      - 허용되는 문자: 영숫자 및 밑줄(_)
      - 최대 길이는 48자입니다.

   b. 유형 — 데이터 유형 목록에서 이 열의 데이터 유형을 선택합니다.

   c. 다른 열을 추가하려는 경우 열 추가를 선택합니다.

7. 파티션 키에서 파티션 키로 `id`를 선택합니다. 각 테이블에는 파티션 키가 필요합니다. 파티션 키는 하나 이상의 열로 구성될 수 있습니다.
8. 클러스터링 열로 `division`을 추가합니다. 클러스터링 열은 선택 사항이며 각 파티션 내의 정렬 순서를 결정합니다.
   a. 클러스터링 열을 추가하려면 클러스터링 열 추가를 선택합니다.
   b. 열 목록에서 division을 선택합니다. 순서 목록에서 ASC를 선택하여 이 열의 값을 오름차순으로 정렬합니다. (내림차순은 DESC를 선택합니다.)
9. 테이블 설정 섹션에서 기본 설정을 선택합니다.
10. 테이블 생성을 선택합니다.
11. 테이블이 생성되었는지 확인합니다.
    a. 탐색 창에서 테이블을 선택합니다.
    b. 테이블이 테이블 목록에 있는지 확인합니다.
    c. 테이블 이름을 선택합니다.
    d. 모든 열과 데이터 유형이 올바른지 확인합니다.

> **Note**
> 테이블에 추가한 순서와 동일한 순서로 열이 나열되지 않을 수 있습니다.

    e. 클러스터링 열에서 division true로 식별되는지 확인합니다. 다른 모든 테이블 열은 false여야 합니다.

CQL 사용

다음 절차는 CQL을 사용하여 다음 열과 데이터 유형을 포함하는 테이블을 만듭니다. `id` 열은 파티션 키가 됩니다.

```
id            text
name          text
region        text
division      text
project       text
role          text
pay_scale     int
vacation_hrs  float
```

```
manager_id    text
```

CQL을 사용하여 테이블을 생성하려면

1. 명령 쉘을 열고 다음을 입력합니다.

   cqlsh

2. cqlsh 프롬프트(cqlsh>)에서 테이블을 생성할 키스페이스를 지정합니다.

   ```
   USE "myGSGKeyspace" ;
   ```

3. 키스페이스 프롬프트(cqlsh:keyspace_name>)에서 명령 창에 다음 코드를 입력하여 테이블을 생성합니다.

   ```
   CREATE TABLE IF NOT EXISTS "myGSGKeyspace".employees_tbl (
      id text,
      name text,
      region text,
      division text,
      project text,
      role text,
      pay_scale int,
      vacation_hrs float,
      manager_id text,
      PRIMARY KEY (id,division))
      WITH CLUSTERING ORDER BY (division ASC) ;
   ```

   > **Note**
   >
   > ASC는 기본 클러스터링 순서입니다. 내림차순으로 DESC를 지정할 수도 있습니다.

   id 열은 파티션 키가 됩니다. 그러면 division은 오름차순(ASC)으로 정렬되는 클러스터링 열입니다.

4. 테이블이 생성되었는지 확인합니다.

   ```
   SELECT * from system_schema.tables WHERE keyspace_name='myGSGKeyspace' ;
   ```

테이블이 나열되어야 합니다.

5. 테이블 구조를 확인합니다.

```
SELECT * FROM system_schema.columns WHERE keyspace_name = 'myGSGKeyspace' AND
 table_name = 'employees_tbl' ;
```

모든 열과 데이터 유형이 예상한 대로인지 확인합니다. 열의 순서는 CREATE 문의 순서와 다를 수 있습니다.

테이블의 데이터에 대해 CRUD(생성, 읽기, 업데이트 및 삭제) 작업을 수행하려면 the section called “2단계: CRUD 작업”로 진행합니다.

자습서 2단계: 데이터 생성, 읽기, 업데이트 및 삭제(CRUD)

이 섹션에서는 콘솔의 CQL 편집기를 사용하여 테이블의 데이터에 대해 CRUD(생성, 읽기, 업데이트 및 삭제) 작업을 수행합니다. cqlsh를 사용하여 명령을 실행할 수도 있습니다.

주제

- 자습서: Amazon Keyspaces 테이블에 데이터 삽입 및 로드
- 자습서: Amazon Keyspaces 테이블에서 읽기
- 자습서: Amazon Keyspaces 테이블의 데이터 업데이트
- 자습서: Amazon Keyspaces 테이블의 데이터 삭제

자습서: Amazon Keyspaces 테이블에 데이터 삽입 및 로드

employees_tbl 테이블에 데이터를 만들려면 INSERT 문을 사용하여 행 하나를 추가합니다.

1. cqlsh를 사용하여 Amazon Keyspaces 테이블에 데이터를 쓰려면 먼저 현재 cqlsh 세션의 쓰기 일관성을 LOCAL_QUORUM으로 설정해야 합니다. 지원되는 일관성 수준에 대한 자세한 내용은 the section called “쓰기 일관성 수준” 섹션을 참조하세요. AWS Management Console에서 CQL 편집기를 사용하는 경우에는 이 단계가 필요하지 않음에 유의하십시오.

```
CONSISTENCY LOCAL_QUORUM;
```

2. 단일 레코드를 삽입하려면 CQL 편집기에서 다음 명령을 실행합니다.

```
INSERT INTO "myGSGKeyspace".employees_tbl (id, name, project, region, division,
 role, pay_scale, vacation_hrs, manager_id)
VALUES ('012-34-5678','Russ','NightFlight','US','Engineering','IC',3,12.5,
 '234-56-7890') ;
```

3. 다음 명령을 실행하여 데이터가 테이블에 올바르게 추가되었는지 확인합니다.

```
SELECT * FROM "myGSGKeyspace".employees_tbl ;
```

cqlsh를 사용하여 파일에서 여러 레코드를 삽입하려면

1. 다음 아카이브 파일 sampledata.zip에 포함된 샘플 데이터 파일(employees.csv)을 다운로드합니다. 이 CSV(쉼표로 구분된 값) 파일에는 다음 데이터가 포함됩니다. 파일을 저장한 경로를 기억해 두세요.

ID	name	project	region	division	role	pay_scale	vacation_hrs	manager_id
123-45-6789	Bob	NightFlight	US	Engineering	Intern	1	0	234-56-7890
234-56-7890	Bob	NightFlight	US	Engineering	Manager	6	72	789-01-2345
345-67-8901	Sarah	Storm	US	Engineering	IC	4	108	234-56-7890
456-78-9012	Beth	NightFlight	US	Engineering	IC	7	100.5	234-56-7890
567-89-0123	Ahmed	NightFlight	US	Marketing	IC	4	88	678-90-1234
678-90-1234	Alan	Storm	US	Marketing	Manager	3	18.4	789-01-2345
789-01-2345	Roberta	All	US	Executive	CEO	15	184	None

2. 명령 쉘을 열고 다음을 입력합니다.

   `cqlsh`

3. `cqlsh` 프롬프트(`cqlsh>`)에서 키스페이스를 지정합니다.

```
USE "myGSGKeyspace" ;
```

4. 쓰기 일관성을 `LOCAL_QUORUM`으로 설정합니다. 지원되는 일관성 수준에 대한 자세한 내용은 the section called “쓰기 일관성 수준” 섹션을 참조하세요.

```
CONSISTENCY LOCAL_QUORUM;
```

5. 키스페이스 프롬프트(`cqlsh:keyspace_name>`)에서 다음 쿼리를 실행합니다.

```
COPY employees_tbl
 (id,name,project,region,division,role,pay_scale,vacation_hrs,manager_id)
```

```
FROM 'path-to-the-csv-file/employees.csv' WITH delimiter=',' AND header=TRUE ;
```

6. 다음 쿼리를 실행하여 데이터가 테이블에 올바르게 추가되었는지 확인합니다.

```
SELECT * FROM employees_tbl ;
```

자습서: Amazon Keyspaces 테이블에서 읽기

이 자습서: Amazon Keyspaces 테이블에 데이터 삽입 및 로드 섹션에서는 SELECT 문을 사용하여 테이블에 데이터를 성공적으로 추가했는지 확인했습니다. 이 섹션에서는 SELECT의 용도를 조정하여 특정 열을 표시하고 특정 기준을 충족하는 행만 표시합니다.

SELECT 문의 일반적인 형식은 다음과 같습니다.

```
SELECT column_list FROM table_name [WHERE condition [ALLOW FILTERING]] ;
```

주제

- 테이블의 모든 데이터 선택
- 열 하위 집합 선택
- 행의 하위 집합 선택

테이블의 모든 데이터 선택

SELECT 문의 가장 간단한 형식은 테이블의 모든 데이터를 반환합니다.

> **Important**
>
> 프로덕션 환경에서는 일반적으로 테이블의 모든 데이터를 반환하는 이 명령을 실행하는 것이 가장 좋은 방법은 아닙니다.

모든 테이블의 데이터를 선택하려면

- 다음 쿼리를 실행합니다.

```
SELECT * FROM "myGSGKeyspace".employees_tbl ;
```

column_list에 대해 와일드카드 문자(*)를 사용하면 모든 열이 선택됩니다.

열 하위 집합 선택

열 하위 집합을 쿼리하려면

- id, name 및 manager_id 열을 검색하려면 다음 쿼리를 실행합니다.

```
SELECT name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;
```

출력에는 SELECT 문에 나열된 순서대로 지정된 열만 포함됩니다.

행의 하위 집합 선택

대규모 데이터 세트를 쿼리할 때는 특정 기준을 충족하는 레코드만 필요할 수 있습니다. 이렇게 하려면 SELECT 문 끝에 WHERE 절을 추가할 수 있습니다.

행 하위 집합을 쿼리하려면

- '234-56-7890' ID가 있는 직원에 대한 레코드만 검색하려면 다음 쿼리를 실행합니다.

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='234-56-7890' ;
```

위의 SELECT 문은 id가 234-56-7890인 행만 반환합니다.

WHERE 절 이해

WHERE 절은 데이터를 필터링하고 지정된 기준을 충족하는 데이터만 반환하는 데 사용됩니다. 지정된 기준은 단순 조건일 수도 있고 복합 조건일 수도 있습니다.

WHERE 절에서 조건을 사용하는 방법

- 단순 조건 - 단일 열

```
WHERE column_name=value
```

다음 조건 중 하나라도 충족되면 WHERE 절에 단순 조건을 사용할 수 있습니다.

- 열은 테이블의 프라이머리 키에 있는 유일한 열입니다.
- WHERE 절의 조건 뒤에 ALLOW FILTERING을 추가합니다.

  ALLOW FILTERING을 사용하면 성능이 일관되지 않을 수 있으며 특히 여러 파티션을 나눈 대형 테이블의 경우 더욱 그렇습니다.

- 복합 조건 - AND로 연결된 여러 개의 단순 조건

```
WHERE column_name1=value1 AND column_name2=value2 AND column_name3=value3...
```

다음 조건 중 하나라도 충족되면 WHERE 절에 복합 조건을 사용할 수 있습니다.

- WHERE 절의 열은 테이블의 프라이머리 키에 있는 열과 정확히 일치합니다. 더 많지도 적지도 않습니다.
- 다음 예와 같이 WHERE 절의 복합 조건 뒤에 ALLOW FILTERING을 추가합니다.

```
SELECT * FROM my_table WHERE col1=5 AND col2='Bob' ALLOW FILTERING ;
```

ALLOW FILTERING을 사용하면 성능이 일관되지 않을 수 있으며 특히 여러 파티션을 나눈 대형 테이블의 경우 더욱 그렇습니다.

사용해보기

자체 CQL 쿼리를 생성하여 employees_tbl 테이블에서 다음을 찾습니다.

- 모든 직원의 name, project 및 id를 찾습니다.
- 인턴 Bob이 어떤 프로젝트를 진행 중인지 찾습니다(결과물에 최소한 인턴의 이름, 프로젝트, 역할 포함).
- 고급: 애플리케이션을 생성하여 인턴 Bob과 매니저가 같은 직원을 모두 찾습니다. 힌트: 이 작업에는 쿼리가 두 개 이상 필요할 수 있습니다.
- 고급: 애플리케이션을 생성하여 프로젝트 NightFlight에서 작업하는 모든 직원 중에서 선택한 열을 찾습니다. 힌트: 이 문제를 해결하려면 여러 문이 필요할 수 있습니다.

자습서: Amazon Keyspaces 테이블의 데이터 업데이트

employees_tbl 테이블의 데이터를 업데이트하려면 UPDATE 문을 사용합니다.

UPDATE 문의 일반적인 형식은 다음과 같습니다.

```
UPDATE table_name SET column_name=new_value WHERE primary_key=value ;
```

Tip

- 다음 예제와 같이 쉼표로 구분된 column_names 및 값 목록을 사용하여 여러 열을 업데이트할 수 있습니다.

  ```
  UPDATE my_table SET col1='new_value_1', col2='new_value2' WHERE id='12345' ;
  ```

- 프라이머리 키가 여러 열로 구성된 경우 모든 프라이머리 키 열과 해당 값이 WHERE 절에 포함되어야 합니다.
- 프라이머리 키의 어떤 열도 업데이트할 수 없습니다. 업데이트하면 레코드의 프라이머리 키가 변경되기 때문입니다.

단일 셀을 업데이트하려면

employees_tbl 테이블을 사용하여 ID 567-89-0123을 가진 직원에게 급여 인상을 제공합니다.

```
UPDATE "myGSGKeyspace".employees_tbl SET pay_scale=5 WHERE id='567-89-0123' AND
 division='Marketing' ;
```

직원의 급여 규모가 현재 5인지 확인합니다.

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='567-89-0123' ;
```

사용해보기

고급: 회사에서 인턴인 Bob을 고용했습니다. 그의 역할이 'IC'이고 급여 규모가 2가 되도록 그의 기록을 변경합니다.

자습서: Amazon Keyspaces 테이블의 데이터 삭제

employees_tbl 테이블의 데이터를 삭제하려면 DELETE 문을 사용합니다.

행이나 파티션에서 데이터를 삭제할 수 있습니다. 삭제는 되돌릴 수 없으므로 데이터를 삭제할 때는 주의해야 합니다.

테이블에서 하나 또는 모든 행을 삭제해도 테이블은 삭제되지 않습니다. 따라서 데이터를 다시 채울 수 있습니다. 테이블을 삭제하면 테이블과 테이블 내의 모든 데이터가 삭제됩니다. 테이블을 다시 사용하려면 테이블을 다시 생성하여 데이터를 추가해야 합니다. 키스페이스를 삭제하면 키스페이스와 그 안에 있는 모든 테이블이 삭제됩니다. 키스페이스와 테이블을 사용하려면 키스페이스와 테이블을 다시 생성한 다음 데이터로 채워야 합니다.

셀 삭제

행에서 열을 삭제하면 지정된 셀의 데이터가 제거됩니다. SELECT 문을 사용하여 해당 열을 표시하면 데이터가 *null*로 표시되지만 해당 위치에 null 값이 저장되지는 않습니다.

하나 이상의 특정 열을 삭제하는 일반적인 구문은 다음과 같습니다.

```
DELETE column_name1[, column_name2...] FROM table_name WHERE condition ;
```

employees_tbl 테이블에서 CEO에게 관리자에 대한 권한이 "None"임을 알 수 있습니다. 먼저 셀에 데이터가 들어 있지 않도록 셀을 삭제합니다.

특정 셀을 삭제하려면

1. 다음 DELETE 쿼리를 실행합니다.

   ```
   DELETE manager_id FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND
    division='Executive';
   ```

2. 삭제가 예상대로 이루어졌는지 확인합니다.

   ```
   SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND
    division='Executive';
   ```

행 삭제

직원이 퇴직하는 경우와 같이 행 전체를 삭제해야 하는 경우가 있을 수 있습니다. 행을 삭제하는 일반적인 구문은 다음과 같습니다.

```
DELETE FROM table_name WHERE condition ;
```

행을 삭제하려면

1. 다음 DELETE 쿼리를 실행합니다.

```
DELETE FROM "myGSGKeyspace".employees_tbl WHERE id='456-78-9012' AND
 division='Engineering';
```

2. 삭제가 예상대로 이루어졌는지 확인합니다.

```
SELECT * FROM "myGSGKeyspace".employees_tbl WHERE id='456-78-9012' AND
 division='Engineering';
```

자습서 3단계: Amazon Keyspaces에서 테이블 및 키스페이스 삭제

필요하지 않은 테이블과 데이터에 대해 요금이 청구되지 않도록 하려면 사용하지 않는 테이블과 키스페이스를 모두 삭제합니다. 테이블을 삭제하면 테이블과 해당 데이터가 삭제되고 이에 대한 요금 발생이 중지됩니다. 하지만 키스페이스는 그대로 유지됩니다. 키스페이스를 삭제하면 키스페이스와 모든 해당 테이블이 삭제되고 이에 대한 요금 발생이 중지됩니다.

테이블 삭제

콘솔 또는 CQL을 사용하여 테이블을 삭제할 수 있습니다. 테이블을 삭제하면 테이블과 모든 해당 데이터가 삭제됩니다.

콘솔 사용

다음 절차에서는 AWS Management Console을 사용하여 테이블과 모든 해당 데이터를 삭제합니다.

콘솔을 사용하여 테이블을 삭제하려면

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 탐색 창에서 테이블을 선택합니다.
3. 삭제할 각 테이블의 이름 왼쪽에 있는 상자를 선택합니다.
4. 삭제를 선택합니다.
5. 테이블 삭제 화면에서 상자에 **Delete**를 입력합니다. 그런 다음 테이블 삭제를 선택합니다.
6. 테이블이 삭제되었는지 확인하려면 탐색 창에서 테이블을 선택하고 `employees_tbl` 테이블이 더 이상 목록에 없는지 확인합니다.

CQL 사용

다음 절차에서는 CQL을 사용하여 테이블과 모든 해당 데이터를 삭제합니다.

CQL을 사용하여 테이블을 삭제하려면

1. 명령 쉘을 열고 다음을 입력합니다.

   cqlsh

2. 키스페이스 프롬프트(cqlsh:*keyspace_name*>)에 다음 명령을 입력하여 테이블을 삭제합니다.

```
DROP TABLE IF EXISTS "myGSGKeyspace".employees_tbl ;
```

3. 테이블이 삭제되었는지 확인합니다.

```
SELECT * FROM system_schema.tables WHERE keyspace_name = 'myGSGKeyspace' ;
```

   테이블이 나열되어서는 안됩니다.

키스페이스 삭제

AWS Management Console 또는 CQL을 사용하여 키스페이스를 삭제할 수 있습니다. 키스페이스를 삭제하면 키스페이스와 모든 해당 테이블 및 데이터가 삭제됩니다.

AWS Management Console사용

다음 절차에서는 AWS Management Console을 사용하여 키스페이스와 모든 해당 테이블 및 데이터를 삭제합니다.

콘솔을 사용하여 키스페이스를 삭제하려면

1. 에 AWS Management Console로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 탐색 창에서 Keyspaces를 선택합니다.
3. 삭제할 각 키스페이스의 이름 왼쪽에 있는 상자를 선택합니다.
4. 삭제를 선택합니다.
5. 키스페이스 삭제 화면에서 상자에 **Delete**를 입력합니다. 그런 다음 키스페이스 삭제를 선택합니다.

6. 키스페이스 myGSGKeyspace가 삭제되었는지 확인하려면 탐색 창에서 Keyspaces를 선택하고 키스페이스가 더 이상 목록에 없는지 확인합니다. 키스페이스를 삭제했으므로 Tables 아래의 employees_tbl 테이블도 나열되지 않아야 합니다.

CQL 사용

다음 절차에서는 CQL을 사용하여 키스페이스와 모든 해당 테이블 및 데이터를 삭제합니다.

CQL을 사용하여 키스페이스를 삭제하려면

1. 명령 쉘을 열고 다음을 입력합니다.

   cqlsh

2. 키스페이스 프롬프트(cqlsh:*keyspace_name*>)에 다음 명령을 입력하여 키스페이스를 삭제합니다.

   ```
   DROP KEYSPACE IF EXISTS "myGSGKeyspace" ;
   ```

3. 키스페이스가 삭제되었는지 확인합니다.

   ```
   SELECT * from system_schema.keyspaces ;
   ```

   키스페이스가 나열되어서는 안 됩니다. 이 작업은 비동기 작업이므로 키스페이스가 삭제될 때까지 지연이 발생할 수 있습니다.

Amazon Keyspaces로 마이그레이션

Amazon Keyspaces (Apache Cassandra용) 로 마이그레이션하면 기업과 조직에 다양하고 매력적인 혜택을 누릴 수 있습니다. Amazon Keyspaces가 마이그레이션을 위한 매력적인 선택이 될 수 있는 몇 가지 주요 이점은 다음과 같습니다.

- 확장성 — Amazon Keyspace는 대규모 워크로드를 처리하고 증가하는 데이터 볼륨 및 트래픽을 수용할 수 있도록 원활하게 확장할 수 있도록 설계되었습니다. 기존 카산드라의 경우 확장이 온디맨드 방식으로 수행되지 않으므로 향후 최고점에 대한 계획이 필요합니다. Amazon Keyspaces를 사용하면 수요에 따라 테이블을 쉽게 확장하거나 축소할 수 있으므로 애플리케이션에서 성능 저하 없이 갑작스러운 트래픽 급증을 처리할 수 있습니다.
- 성능 — Amazon Keyspace는 지연 시간이 짧은 데이터 액세스를 제공하여 애플리케이션이 뛰어난 속도로 데이터를 검색하고 처리할 수 있도록 합니다. 분산 아키텍처는 읽기 및 쓰기 작업이 여러 노드에 분산되도록 하여 요청 속도가 높아도 일관되게 10밀리초 미만의 응답 시간을 제공합니다.
- 완전 관리형 — Amazon Keyspace는 에서 제공하는 완전 관리형 서비스입니다. AWS즉, 프로비저닝, 구성, 패치, 백업, 규모 조정 등 데이터베이스 관리의 운영 측면을 AWS 처리합니다. 따라서 개발자는 데이터베이스 관리 작업보다는 애플리케이션 개발에 더 집중할 수 있습니다.
- 서버리스 아키텍처 — Amazon Keyspace는 서버리스입니다. 사전 용량 프로비저닝 없이 사용한 용량에 대해서만 비용을 지불하면 됩니다. 관리할 서버나 선택할 인스턴스가 없습니다. 이 pay-per-request 모델은 용량을 프로비저닝하고 모니터링할 필요 없이 사용한 리소스에 대해서만 비용을 지불하므로 비용 효율성이 뛰어나고 운영 오버헤드가 최소화됩니다.
- 스키마를 통한 NoSQL 유연성 — Amazon Keyspace는 NoSQL 데이터 모델을 따르므로 스키마 설계에 유연성을 제공합니다. Amazon Keyspaces를 사용하면 정형, 반정형 및 비정형 데이터를 저장할 수 있으므로 다양하고 진화하는 데이터 유형을 처리하는 데 적합합니다. 또한 Amazon Keyspace는 쓰기 시 스키마 검증을 수행하여 데이터 모델을 중앙 집중식으로 발전시킬 수 있습니다. 이러한 유연성 덕분에 개발 주기를 단축하고 변화하는 비즈니스 요구 사항에 쉽게 적응할 수 있습니다.
- 고가용성 및 내구성 — Amazon Keyspace는 한 영역 내의 여러 가용 영역에 데이터를 복제하여 높은 가용성과 데이터 내구성을 보장합니다. AWS 리전복제, 장애 조치 및 복구를 자동으로 처리하여 데이터 손실이나 서비스 중단의 위험을 최소화합니다. Amazon Keyspace는 최대 99.999% 의 가용성 SLA를 제공합니다. Amazon Keyspace는 복원력을 높이고 로컬 읽기 지연 시간을 줄이기 위해 다중 지역 복제를 제공합니다.
- 보안 및 규정 준수 — Amazon Keyspaces는 세밀한 액세스 제어를 AWS Identity and Access Management 위해 와 통합됩니다. 저장 및 전송 시 암호화를 제공하여 데이터 보안을 개선하는 데 도움이 됩니다. 또한 Amazon Keyspace는 HIPAA, PCI DSS, GDPR을 비롯한 보안 표준 및 개인 정보 보호법을 준수하므로 규제 요구 사항을 충족할 수 있습니다.

- AWS 생태계와의 통합 — Amazon Keyspace는 AWS 에코시스템의 일부로서 AWS CloudFormation Amazon AWS 서비스등과 같은 다른 곳과 원활하게 통합됩니다. CloudWatch AWS CloudTrail이러한 통합을 통해 서버리스 아키텍처를 구축하고, 코드형 인프라를 활용하고, 실시간 데이터 기반 애플리케이션을 만들 수 있습니다.

Amazon Keyspace로의 마이그레이션에 대한 일반 고려 사항

- 마이그레이션을 더 작은 구성 요소로 나눕니다.

  원시 데이터 크기 측면에서 다음과 같은 마이그레이션 단위와 잠재적 공간을 고려합니다. 한 단계 이상의 단계에서 소량의 데이터를 마이그레이션하면 마이그레이션을 단순화하는 데 도움이 될 수 있습니다.

  - 클러스터별 - 모든 Cassandra 데이터를 한 번에 마이그레이션합니다. 이 접근 방식은 소규모 클러스터의 경우 괜찮을 수 있습니다.
  - 키스페이스 또는 테이블별 - 마이그레이션을 키스페이스 또는 테이블 그룹으로 나눕니다. 이 접근 방식을 사용하면 각 워크로드의 요구 사항에 따라 단계적으로 데이터를 마이그레이션할 수 있습니다.
  - 데이터별 - 데이터 크기를 더 줄이려면 특정 사용자 그룹 또는 제품에 대한 데이터를 마이그레이션하는 것을 고려합니다.
- 단순성을 기반으로 먼저 마이그레이션할 데이터의 우선 순위를 정합니다.

  먼저 더 쉽게 마이그레이션할 수 있는 데이터가 있는지 고려합니다. 예를 들어 특정 시간대에 변경되지 않는 데이터, 야간 배치 작업의 데이터, 오프라인 시간 동안 사용하지 않은 데이터 또는 내부 앱의 데이터 등이 여기에 해당하는지 고려합니다.

주제

아파치 카산드라에서 데이터를 마이그레이션하기 위한 지침

Apache Cassandra에서 Amazon Keyspaces로 성공적으로 마이그레이션하려면 사용 가능한 옵션을 신중하게 계획하고 비교하는 것이 좋습니다. 이 주제에서는 마이그레이션 프로세스의 작동 방식, 사용 가능한 도구, 다양한 마이그레이션 전략을 평가하여 요구 사항에 가장 적합한 전략을 선택하는 방법을 설명합니다.

주제

기능적 호환성

마이그레이션하기 전에 Apache Cassandra와 Amazon Keyspaces 간의 기능적 차이를 신중하게 고려하십시오. Amazon Keyspaces는 키스페이스 및 테이블 생성, 데이터 읽기, 데이터 쓰기 등 일반적으로 사용되는 모든 Cassandra 데이터 플레인 작업을 지원합니다. 하지만 Amazon Keyspace에서는 지원하지 않는 일부 카산드라 API도 있습니다. 지원되는 API에 대한 자세한 내용은 을 참조하십시오. the section called “지원되는 Cassandra API, 작업, 함수, 데이터 유형” Amazon Keyspace와 Apache Cassandra 간의 모든 기능적 차이에 대한 개요는 를 참조하십시오. the section called “Apache Cassandra와의 기능적 차이”

사용 중인 Cassandra API 및 스키마를 Amazon Keyspaces에서 지원되는 기능과 비교하려면 Amazon Keyspaces 도구 키트에서 사용할 수 있는 호환성 스크립트를 실행할 수 있습니다. GitHub

호환성 스크립트 사용 방법

1. 에서 GitHub호환성 Python 스크립트를 다운로드하여 기존 Apache Cassandra 클러스터에 액세스할 수 있는 위치로 이동합니다.
2. 호환성 스크립트는 와 유사한 매개 변수를 사용합니다. `CQLSH` 클러스터의 Cassandra 노드 중 하나에 연결하고 쿼리를 실행하는 데 사용하는 IP 주소 및 포트를 `--port` 입력하고 입력합니다. `--host` Cassandra 클러스터에서 인증을 사용하는 경우 및 도 제공해야 합니다. `-username -password` 호환성 스크립트를 실행하려면 다음 명령을 사용할 수 있습니다.

```
python toolkit-compat-tool.py --host hostname or IP -u "username" -p "password" --
port native transport port
```

아마존 키스페이스 가격 추정

이 섹션에서는 Amazon Keyspaces의 예상 비용을 계산하기 위해 Apache Cassandra 테이블에서 수집해야 하는 정보에 대한 개요를 제공합니다. 각 테이블에는 서로 다른 데이터 유형이 필요하고, 서로 다른 CQL 쿼리를 지원해야 하며, 고유한 읽기/쓰기 트래픽을 유지해야 합니다. 테이블을 기반으로 요구

사항을 고려하면 Amazon Keyspace의 테이블 수준 리소스 격리 및 읽기/쓰기 처리 용량 모드와 일치합니다. Amazon Keyspaces를 사용하면 테이블의 읽기/쓰기 용량 및 자동 조정 정책을 독립적으로 정의할 수 있습니다. 테이블 요구 사항을 이해하면 기능, 비용 및 마이그레이션 노력을 기반으로 마이그레이션할 테이블의 우선 순위를 정하는 데 도움이 됩니다.

마이그레이션하기 전에 다음 Cassandra 테이블 지표를 수집하십시오. 이 정보는 Amazon Keyspace에서의 워크로드 비용을 추정하는 데 도움이 됩니다.

- 테이블 이름 — 정규화된 키스페이스의 이름 및 테이블 이름.
- 설명 — 테이블에 대한 설명 (예: 테이블 사용 방법 또는 테이블에 저장된 데이터 유형)
- 초당 평균 읽기 — 오랜 기간 동안 테이블에 대해 좌표 수준에서 읽은 평균 횟수입니다.
- 초당 평균 쓰기 — 오랜 기간 동안 테이블에 대한 평균 좌표 수준 쓰기 횟수입니다.
- 평균 행 크기 (바이트) — 평균 행 크기 (바이트).
- 스토리지 크기 (GB) - 테이블의 원시 스토리지 크기입니다.
- 읽기 일관성 분석 — 최종 일관성 (LOCAL_ONE또는ONE) 을 사용하는 읽기와 강력한 일관성 () 을 사용하는 읽기의 비율입니다. LOCAL_QUORUM

이 표에는 마이그레이션을 계획할 때 취합해야 하는 테이블 관련 정보의 예가 나와 있습니다.

테이블 이름	설명	초당 평균 읽기	초당 평균 쓰기	평균 행 크기 (바이트)	스토리지 크기 (GB)	읽기 일관성 분석
마이키스페이스. 마이 테이블	장바구니 기록을 저장하는 데 사용됩니다.	10,000개	5,000	2,200	2,000	100% LOCAL_ONE
마이 키스페이스. 마이 테이블 2	최신 프로필 정보를 저장하는 데 사용됩니다.	20,000건	1,000	850	1,000	25% LOCAL_QUORUM 75% LOCAL_ONE

테이블 메트릭을 수집하는 방법

이 섹션에서는 기존 Cassandra 클러스터에서 필요한 테이블 메트릭을 수집하는 방법에 대한 단계별 지침을 제공합니다. 이러한 지표에는 행 크기, 테이블 크기, 초당 읽기/쓰기 요청 (RPS) 이 포함됩니다. 이를 통해 Amazon Keyspaces 테이블의 처리 용량 요구 사항을 평가하고 요금을 추정할 수 있습니다.

Cassandra 소스 테이블에서 테이블 지표를 수집하는 방법

1. 행 크기 결정

   행 크기는 Amazon Keyspace의 읽기 용량 및 쓰기 용량 사용률을 결정하는 데 중요합니다. 다음 다이어그램은 카산드라 토큰 범위에 걸친 일반적인 데이터 분포를 보여줍니다.

   에서 사용할 수 있는 행 크기 샘플러 스크립트를 사용하여 Cassandra 클러스터의 각 테이블에 대한 GitHub행 크기 메트릭을 수집할 수 있습니다. 이 스크립트는 구성 가능한 테이블 데이터 샘플 세트에 대한 행 크기의 최소, 최대, 평균 `cqlsh` 및 `awk` 표준 편차를 사용하고 계산하는 방식으로 Apache Cassandra에서 테이블 데이터를 내보냅니다. 행 크기 샘플러는 인수를 로 `cqlsh` 전달하므로 동일한 매개변수를 사용하여 Cassandra 클러스터를 연결하고 읽을 수 있습니다.

   다음 문은 이에 대한 예입니다.

```
./row-size-sampler.sh 10.22.33.44 9142 \\
   -u "username" -p "password" --ssl
```

   Amazon Keyspaces에서 행 크기를 계산하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “행 크기 계산”

2. 테이블 크기 결정

Amazon Keyspaces를 사용하면 스토리지를 미리 프로비저닝할 필요가 없습니다. Amazon Keyspaces는 테이블의 청구 가능 크기를 지속적으로 모니터링하여 스토리지 요금을 결정합니다. 스토리지는 월별 GB당 요금이 청구됩니다. Amazon Keyspaces 테이블 크기는 단일 복제본의 원시 크기 (압축되지 않은 크기) 를 기준으로 합니다. Amazon Keyspaces에서 테이블 크기를 모니터링하려면 의 각 테이블에 대해 표시되는 지표를 `BillableTableSizeInBytes` 사용할 수 있습니다. AWS Management Console

Amazon Keyspaces 테이블의 청구 가능 크기를 추정하려면 다음 두 가지 방법 중 하나를 사용할 수 있습니다.

- 평균 행 크기를 사용하고 행 수를 곱하십시오.

  평균 행 크기에 Cassandra 소스 테이블의 행 수를 곱하여 Amazon Keyspaces 테이블의 크기를 추정할 수 있습니다. 이전 섹션의 행 크기 샘플 스크립트를 사용하여 평균 행 크기를 캡처하십시오. 행 수를 `dsbulk count` 캡처하려면 등의 도구를 사용하여 원본 테이블의 총 행 수를 확인할 수 있습니다.

- 를 사용하여 테이블 메타데이터를 `nodetool` 수집할 수 있습니다.

  `Nodetool`Apache Cassandra 배포판에서 제공되는 관리 도구로, Cassandra 프로세스의 상태를 파악하고 테이블 메타데이터를 반환합니다. 테이블 크기에 대한 메타데이터를 `nodetool` 샘플링하여 Amazon Keyspace의 테이블 크기를 추정하는 데 사용할 수 있습니다. 사용할 명령은 입니다. `nodetool tablestats` Tablestats는 테이블의 크기와 압축률을 반환합니다. 테이블 크기는 테이블의 크기로 저장되며 이 크기를 로 나눌 수 있습니다. `tablelivespace` `compression ratio` 그런 다음 이 크기 값에 노드 수를 곱합니다. 마지막으로 복제 인자 (일반적으로 3) 로 나눕니다. 이것은 테이블 크기를 평가하는 데 사용할 수 있는 완전한 계산 공식입니다.

  ```
  ((tablelivespace / compression ratio) * (total number of nodes))/ (replication
   factor)
  ```

  Cassandra 클러스터에 12개의 노드가 있다고 가정해 보겠습니다. `nodetool tablestats`명령을 실행하면 `tablelivespace` 200GB의 a와 0.5의 a가 `compression ratio` 반환됩니다. 키스페이스의 복제 인자는 3입니다. 이 예제의 계산 방식은 다음과 같습니다.

  ```
  (200 GB / 0.5) * (12 nodes)/ (replication factor of 3)
                           = 4,800 GB / 3
  ```

```
                        = 1,600 GB is the table size estimate for Amazon
Keyspaces
```

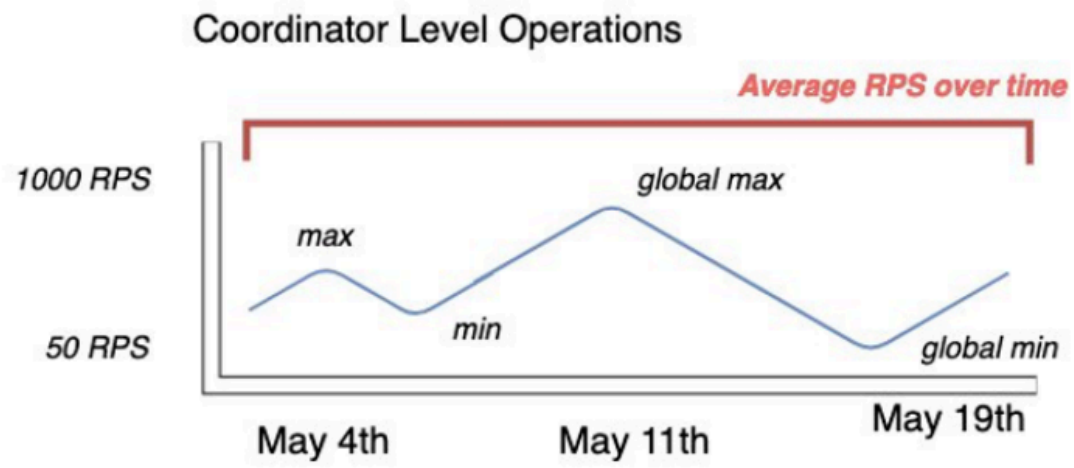
3. 읽기 및 쓰기 횟수를 캡처합니다.

   Amazon Keyspaces 테이블의 용량 및 규모 조정 요구 사항을 확인하려면 마이그레이션 전에 Cassandra 테이블의 읽기 및 쓰기 요청 비율을 캡처하십시오.

   Amazon Keyspaces는 서버리스이므로 사용한 만큼만 비용을 지불하면 됩니다. 일반적으로 Amazon Keyspace의 읽기/쓰기 처리량 요금은 요청 수와 크기를 기준으로 합니다. Amazon Keyspace에는 온디맨드 용량 모드와 프로비저닝 용량 모드의 두 가지 용량 모드가 있습니다. 온디맨드 용량 모드는 용량을 계획할 필요 없이 초당 수천 건의 요청을 처리할 수 있는 유연한 청구 옵션입니다. 이 옵션은 읽기 및 쓰기 요청에 대한 pay-per-request 요금을 제공하므로 사용한 만큼만 비용을 지불하면 됩니다. 프로비저닝된 처리량 용량 모드를 선택하는 경우 애플리케이션에 필요한 초당 읽기 및 쓰기 횟수를 지정합니다. 이를 통해 Amazon Keyspaces 사용이 정의된 요청 속도 이하로 유지되도록 관리하여 가격을 최적화하고 예측 가능성을 유지할 수 있습니다. 프로비저닝된 모드에서는 Auto Scaling을 통해 프로비저닝 속도를 자동으로 조정하여 규모를 늘리거나 줄여 운영 효율성을 높일 수 있습니다. 서버리스 리소스 관리에 대한 자세한 내용은 을 참조하십시오. 서버리스 리소스 관리

   Amazon Keyspaces에서 읽기 및 쓰기 처리 용량을 개별적으로 프로비저닝하므로 기존 테이블의 읽기 및 쓰기 요청 속도를 개별적으로 측정해야 합니다.

   기존 Cassandra 클러스터에서 가장 정확한 사용률 지표를 수집하려면 단일 데이터 센터의 모든 노드에 걸쳐 집계된 테이블에 대해 장기간에 걸친 코디네이터 수준의 읽기 및 쓰기 작업에 대한 초당 평균 요청 수 (RPS) 를 캡처하십시오. 다음 다이어그램과 같이 최소 몇 주 동안 평균 RPS를 캡처하면 트래픽 패턴의 최고점과 최저점을 캡처할 수 있습니다.



Cassandra 테이블의 읽기 및 쓰기 요청 비율을 결정하는 데는 두 가지 옵션이 있습니다.

- 기존 카산드라 모니터링 사용

  다음 표에 표시된 지표를 사용하여 읽기 및 쓰기 요청을 관찰할 수 있습니다. 단, 지표 이름은 사용 중인 모니터링 도구에 따라 변경될 수 있습니다.

측정기준	카산드라 JMX 메트릭
씁니다.	`org.apache.cassandra.metrics:type=ClientRequest,scope=Write,name=Latency#Count`
읽습니다	`org.apache.cassandra.metrics:type=ClientRequest,scope=Read,name=Latency#Count`

- `nodetool` 사용

  `nodetool tablestats`및 `nodetool info` 를 사용하여 테이블에서 평균 읽기 및 쓰기 작업을 캡처합니다. `tablestats`노드가 시작된 시점부터 총 읽기 및 쓰기 수를 반환합니다. `nodetool info`노드의 가동 시간을 초 단위로 제공합니다. 초당 평균 읽기 및 쓰기 수를 구하려면 읽기 및 쓰기 수를 노드 가동 시간 (초) 으로 나눕니다. 그런 다음 읽기의 경우 일관성 수준으로 나누고 쓰기의 경우 복제 인자로 나눕니다. 이러한 계산은 다음 공식으로 표현됩니다.

초당 평균 읽기 수 공식:

```
((number of reads * number of nodes in cluster) / read consistency quorum
 (2)) / uptime
```

초당 평균 쓰기 수 공식:

```
((number of writes * number of nodes in cluster) / replication factor of 3) /
 uptime
```

4주 동안 가동된 12노드 클러스터가 있다고 가정해 보겠습니다. `nodetool info`2,419,200초의 가동 시간을 반환하고 10억 개의 쓰기와 20억 개의 읽기를 `nodetool tablestats` 반환합니다. 이 예제의 계산 결과는 다음과 같습니다.

```
((2 billion reads * 12 in cluster) / read consistency quorum (2)) / 2,419,200
 seconds
=  12 billion reads / 2,419,200 seconds
=  4,960 read request per second
                        ((1 billion writes * 12 in cluster) / replication
 factor of 3) / 2,419,200 seconds
=  4 billion writes / 2,419,200 seconds
=  1,653 write request per second
```

4. 테이블의 용량 활용도를 결정하십시오.

   평균 용량 사용률을 추정하려면 먼저 Cassandra 소스 테이블의 평균 요청 비율과 평균 행 크기에서 시작하십시오.

   Amazon Keyspace는 RCU (읽기 용량 단위) 와 WCU (쓰기 용량 단위) 를 사용하여 테이블의 읽기 및 쓰기에 대한 프로비저닝된 처리 용량을 측정합니다. 이 추정치에서는 이러한 단위를 사용하여 마이그레이션 후 새 Amazon Keyspaces 테이블의 읽기 및 쓰기 용량 요구량을 계산합니다. 이 주제 후반부에서는 프로비저닝된 용량 모드와 온디맨드 용량 모드 중 선택이 청구에 미치는 영향에 대해 설명하겠습니다. 하지만 용량 사용률 추정치에서는 테이블이 프로비저닝 모드라고 가정합니다.

   RCU 1개는 최대 4KB 크기의 행에 대한 `LOCAL_ONE` 읽기 요청 1개 `LOCAL_QUORUM` 또는 읽기 요청 2개를 나타냅니다. 4KB보다 큰 행을 읽어야 하는 경우 읽기 작업에 추가 RCU가 사용됩니다. 필요한 총 RCU 수는 행 크기, 사용 `LOCAL_QUORUM` 또는 `LOCAL_ONE` 읽기 일관성 여부에

따라 달라집니다. 예를 들어 8KB 행을 읽으려면 읽기 일관성을 사용하는 RCU 2개가 필요하고 LOCAL_QUORUM 읽기 일관성을 선택한 LOCAL_ONE 경우 RCU 1개가 필요합니다.

하나의 WCU는 크기가 최대 1KB인 행에 대한 쓰기 1회를 나타냅니다. 모든 쓰기는 LOCAL_QUORUM 일관성을 사용하며 간단한 트랜잭션(LWT) 사용에 대한 추가 비용은 없습니다. 1KB보다 큰 행을 써야 하는 경우 쓰기 작업에 추가 WCU가 사용됩니다. 필요한 WCU의 총 수는 행 크기에 따라 달라집니다. 예를 들어 행 크기가 2KB인 경우 쓰기 요청 하나를 수행하려면 WCU 2개가 필요합니다.

다음 공식을 사용하여 필요한 RCU 및 WCU를 추정할 수 있습니다. RCU의 읽기 용량은 초당 읽기 당 읽은 행 수를 곱하고 평균 행 크기를 4KB로 나눈 다음 가장 가까운 정수로 반올림하여 구할 수 있습니다.

WCU의 쓰기 용량은 요청 수에 평균 행 크기를 1KB로 나눈 다음 가장 가까운 정수로 반올림하여 계산할 수 있습니다. 이는 다음 공식으로 표현됩니다.

```
Read requests per second * ROUNDUP((Average Row Size)/4096 per unit) =  RCUs per
 second

Write requests per second * ROUNDUP(Average Row Size/1024 per unit) = WCUs per
 second
```

예를 들어 카산드라 테이블에서 행 크기가 2.5KB인 4,960개의 읽기 요청을 수행하는 경우 Amazon Keyspace에는 4,960개의 RCU가 필요합니다. 현재 카산드라 테이블에서 2.5KB의 행 크기로 초당 1,653개의 쓰기 요청을 수행하고 있다면 Amazon Keyspaces에서 초당 4,959개의 WCU가 필요합니다. 이 예제는 다음 공식으로 표현됩니다.

```
4,960 read requests per second * ROUNDUP( 2.5KB /4KB bytes per unit)
= 4,960 read requests per second * 1 RCU
= 4,960 RCUs

1,653 write requests per second * ROUNDUP(2.5KB/1KB per unit)
= 1,653 requests per second * 3 WCUs
= 4,959 WCUs
```

를 eventual consistency 사용하면 각 읽기 요청의 처리 용량을 최대 절반까지 절약할 수 있습니다. 최종적으로 일관된 각 읽기는 최대 8KB를 소비할 수 있습니다. 다음 공식에 표시된 대로 이전 계산에 0.5를 곱하여 최종 일관된 읽기를 계산할 수 있습니다.

```
4,960 read requests per second * ROUNDUP( 2.5KB /4KB per unit) * .5
= 2,480 read request per second * 1 RCU
= 2,480 RCUs
```

5. Amazon Keyspaces의 월별 예상 요금을 계산해 보십시오.

   읽기/쓰기 용량 처리량을 기반으로 테이블의 월별 요금을 추정하려면 다양한 공식을 사용하여 온디맨드 모드와 프로비저닝 모드의 요금을 계산하고 테이블의 옵션을 비교해 보면 됩니다.

   프로비저닝된 모드 — 읽기 및 쓰기 용량 사용량은 초당 용량 단위를 기준으로 시간당 요금이 청구됩니다. 먼저 이 비율을 0.7로 나누어 기본 자동 크기 조정 목표 사용률인 70% 를 나타냅니다. 그런 다음 달력일 기준 30일, 하루 24시간, 지역별 요금제를 곱합니다. 이 계산은 다음 공식에 요약되어 있습니다.

   ```
   (read capacity per second / .7) * 24 hours * 30 days * regional rate
                   (write capacity per second / .7) * 24 hours * 30 days * regional
    rate
   ```

   온디맨드 모드 - 읽기 및 쓰기 용량은 요청당 요금이 청구됩니다. 먼저 요청 속도에 달력일 기준 30일, 하루 24시간을 곱합니다. 그런 다음 요청 단위 100만 개로 나눕니다. 마지막으로 지역 요금을 곱합니다. 이 계산은 다음 공식에 요약되어 있습니다.

   ```
   ((read capacity per second * 30 * 24 * 60 * 60) / 1 Million read request units) *
    regional rate
                   ((write capacity per second * 30 * 24 * 60 * 60) / 1 Million write
    request units) * regional rate
   ```

마이그레이션 전략을 선택합니다.

일반적으로 Apache Cassandra에서 Amazon Keyspaces로 마이그레이션할 때는 다음과 같은 세 가지 마이그레이션 전략 중에서 선택할 수 있습니다.

- 오프라인 — 이 마이그레이션에는 파란색/녹색 스타일의 애플리케이션 마이그레이션 배포를 통해 Cassandra에서 Amazon Keyspaces로 데이터 세트를 복사하는 작업이 포함됩니다. 마이그레이션 중에 애플리케이션이 다운타임을 어느 정도 감수할 수 있는 경우 이 옵션을 사용하면 마이그레이션 프로세스를 간소화할 수 있습니다. 오프라인 마이그레이션에 대한 자세한 내용은 을 참조하십시오.

  the section called “오프라인 마이그레이션”.

- 온라인 - 일반적으로 애플리케이션 로직에 직접 기록되는 이중 쓰기를 포함하는 카나리아 스타일 배포입니다. 마이그레이션 중에 다운타임이 전혀 없어야 하는 애플리케이션의 경우 실시간 읽기 및 쓰기가 한 데이터 소스에서 다른 데이터 소스로 전환되는 동안 데이터를 복사해야 합니다.
- 하이브리드 — 이 접근 방식을 사용하면 변경 내용을 거의 실시간으로 복제할 수 있지만 읽기와 쓰기의 전환은 애플리케이션에서 담당합니다.

사용 가능한 마이그레이션 전략을 자세히 검토한 후 요구 사항과 가용 리소스를 고려하여 프로세스를 단순화하는 옵션을 의사 결정 트리에 배치할 수 있습니다.

Amazon Keyspace로의 오프라인 마이그레이션

오프라인 마이그레이션은 마이그레이션을 수행하기 위해 다운타임을 감수할 수 있는 경우에 적합합니다. 기업에서는 일반적으로 패치 적용, 대규모 릴리즈 또는 하드웨어 업그레이드 또는 주요 업그레이드를 위한 다운타임을 위한 유지 관리 기간이 있습니다. 오프라인 마이그레이션은 이 창을 사용하여 데이터를 복사하고 애플리케이션 트래픽을 Apache Cassandra에서 Amazon Keyspace로 전환할 수 있습니다. 오프라인 마이그레이션은 Cassandra와 Amazon Keyspace에 동시에 통신할 필요가 없으므로 애플리케이션 수정 작업이 줄어듭니다. 또한 데이터 흐름이 일시 중지된 상태에서도 변형을 유지하지 않고도 정확한 상태를 복사할 수 있습니다.

이 예시에서는 다운타임을 최소화하기 위해 오프라인 마이그레이션 중에 Amazon Simple Storage Service (Amazon S3) 를 데이터 스테이징 영역으로 사용합니다. Spark Cassandra 커넥터 및 를 사용하여 Amazon S3에 Parquet 형식으로 저장한 데이터를 Amazon Keyspaces 테이블로 자동으로 가져올 수 있습니다. AWS Glue다음 섹션에서는 프로세스에 대한 개괄적인 개요를 보여줍니다. Github에서 이 프로세스의 코드 예제를 찾을 수 있습니다.

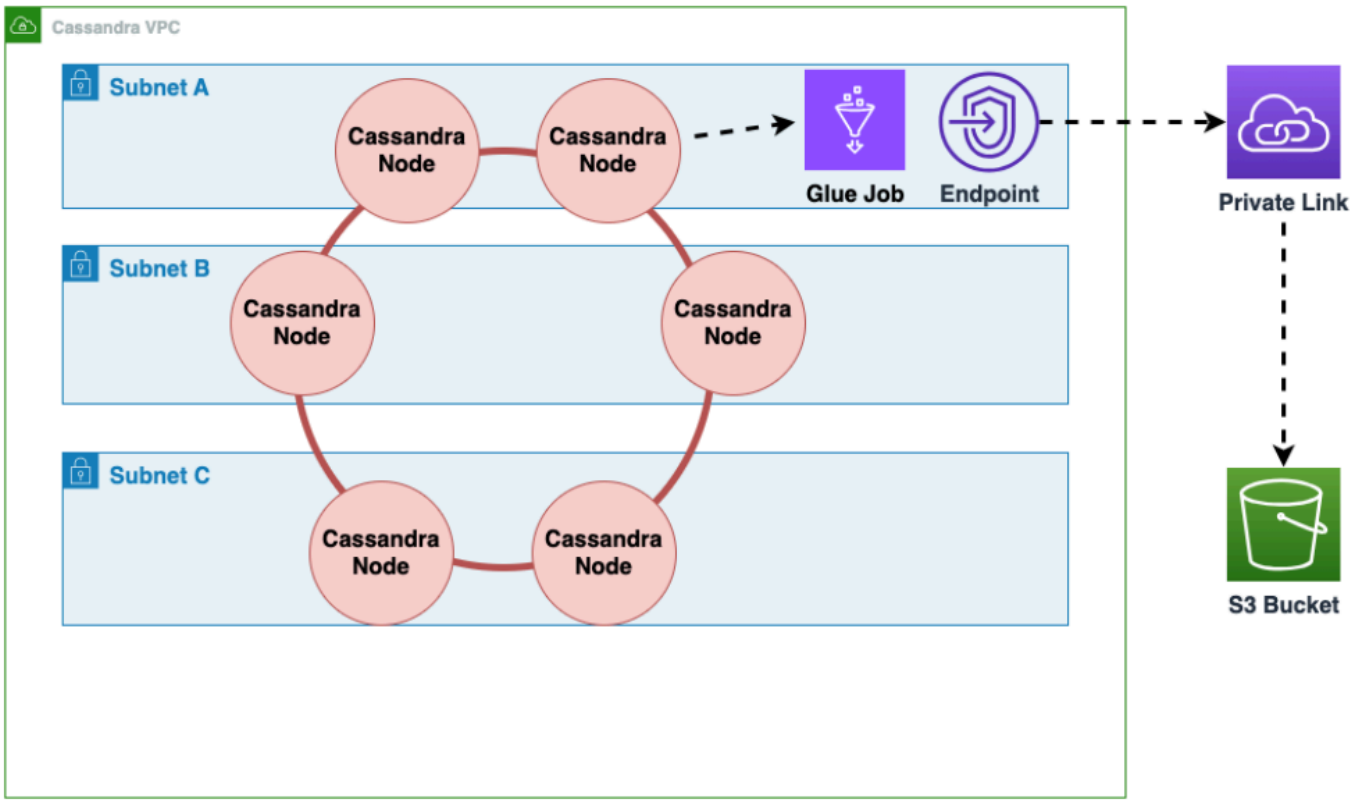
Amazon S3를 사용하여 Apache Cassandra에서 Amazon Keyspaces로 오프라인 마이그레이션을 진행하려면 다음 작업이 AWS Glue 필요합니다. AWS Glue

1. CQL 데이터를 추출 및 변환하여 Amazon S3 버킷에 저장하는 ETL 작업입니다.
2. 버킷에서 Amazon Keyspaces로 데이터를 가져오는 두 번째 작업입니다.
3. 세 번째 작업은 증분 데이터를 가져오는 것입니다.

Amazon Virtual Private Cloud에서 Amazon EC2를 실행하는 카산드라에서 Amazon Keyspace로 오프라인 마이그레이션을 수행하는 방법

1. 먼저 Cassandra에서 Parquet 형식으로 테이블 데이터를 내보내고 Amazon S3 버킷에 저장하는 AWS Glue 데 사용합니다. Cassandra를 실행하는 Amazon EC2 인스턴스가 있는 VPC에 대한

AWS Glue 커넥터를 사용하여 AWS Glue 작업을 실행해야 합니다. 그런 다음 Amazon S3 프라이빗 엔드포인트를 사용하여 Amazon S3 버킷에 데이터를 저장할 수 있습니다. 다음 다이어그램은 이러한 단계를 보여줍니다.



2. Amazon S3 버킷의 데이터를 셔플하여 데이터 무작위화를 개선하십시오. 데이터를 균등하게 가져오면 대상 테이블에 더 많은 트래픽을 분산시킬 수 있습니다. 이 단계는 Amazon Keyspace에 데이터를 삽입할 때 핫 키 패턴을 피하기 위해 큰 파티션 (1000개 이상의 행이 있는 파티션) 이 있는 Cassandra에서 데이터를 내보낼 때 필요합니다. 핫 키 문제는 Amazon `WriteThrottleEvents` Keyspaces에서 발생하며 이로 인해 로드 시간이 늘어납니다.

3. 다른 AWS Glue 작업을 사용하여 Amazon S3 버킷에서 Amazon Keyspace로 데이터를 가져올 수 있습니다. Amazon S3 버킷의 셔플 데이터는 Parquet 형식으로 저장됩니다.

Amazon Keyspaces로 데이터를 마이그레이션하기 위한 도구

다양한 도구를 사용하여 데이터를 Amazon Keyspace로 마이그레이션할 수 있습니다.

- 마이그레이션 도구
  - 대규모 마이그레이션의 경우 ETL (추출, 변환, 로드) 도구 사용을 고려해 보세요. AWS Glue 를 사용하여 데이터 변환 마이그레이션을 빠르고 효과적으로 수행할 수 있습니다.
  - Apache Cassandra Spark 커넥터를 사용하여 Amazon Keyspaces에 데이터를 쓰는 방법을 알아보려면 Apache Spark와 통합 섹션을 참조하세요.
  - cqlsh `COPY FROM` 명령을 사용하여 Amazon Keyspaces로 데이터를 빠르게 로드합니다. cqlsh는 Apache Cassandra에 포함되어 있으며 작은 데이터 세트 또는 테스트 데이터를 로드하는 데 가장 적합합니다. 자세한 step-by-step 지침은 을 참조하십시오. the section called “cqlsh를 사용하여 데이터 로드”
  - 또한 Apache Cassandra용 DataStax 벌크 로더를 사용하여 명령을 사용하여 Amazon Keyspaces에 데이터를 로드할 수 있습니다. `dsbulk` DSBulk는 cqlsh보다 더 강력한 가져오기 기능을 제공하며 리포지토리에서 사용할 수 있습니다. GitHub 자세한 step-by-step 지침은 을 참조하십시오. the section called “DSBulk를 사용하여 데이터 로드”

주제

- 자습서: cqlsh를 사용하여 Amazon Keyspaces에 데이터 로드
- 자습서: DSBulk를 사용하여 Amazon Keyspaces에 데이터 로드

자습서: cqlsh를 사용하여 Amazon Keyspaces에 데이터 로드

이 step-by-step 자습서는 명령을 사용하여 Apache Cassandra에서 Amazon Keyspaces로 데이터를 마이그레이션하는 방법을 안내합니다. `cqlsh COPY` 이 자습서에서는 다음 작업을 수행합니다.

주제

필수 조건

이 자습서를 시작하려면 먼저 다음 작업을 완료해야 합니다.

1. 아직 등록하지 않았다면 의 단계에 따라 가입하십시오. AWS 계정 the section called “설 AWS Identity and Access Management정”
2. the section called “콘솔을 사용하여 서비스별 보안 인증 정보 생성”의 단계에 따라 서비스별 보안 인증을 생성합니다.
3. Cassandra 쿼리 언어 쉘(cqlsh) 연결을 설정하고 the section called “cqlsh 사용하기”의 단계에 따라 Amazon Keyspaces에 연결할 수 있는지 확인합니다.

1단계: 소스 CSV 파일 및 대상 테이블 생성

이 자습서에서는 이름 `keyspaces_sample_table.csv`가 있는 쉼표로 구분된 값(CSV) 파일을 데이터 마이그레이션의 원본 파일로 사용합니다. 제공된 샘플 파일에는 이름이 `book_awards`인 테이블에 대한 몇 개의 데이터 행이 포함되어 있습니다.

1. 소스 파일을 생성합니다. 다음 옵션 중 하나를 선택할 수 있습니다.
   - 다음 아카이브 파일 samplemigration.zip에 포함된 샘플 CSV 파일 (`keyspaces_sample_table.csv`)을 다운로드합니다. 아카이브의 압축을 풀고 `keyspaces_sample_table.csv`의 경로를 기록해 둡니다.
   - Apache Cassandra 데이터베이스에 저장된 자체 데이터로 CSV 파일을 채우려면 다음 예와 같이 `cqlsh` 및 `COPY TO` 문을 사용하여 소스 CSV 파일을 채울 수 있습니다.

```
cqlsh localhost 9042 -u "username" -p "password" --execute
 "COPY mykeyspace.mytable TO 'keyspaces_sample_table.csv' WITH HEADER=true"
```

생성한 CSV 파일이 다음 요구 사항을 충족하는지 확인합니다.

- 첫 번째 행에는 열 이름이 포함됩니다.
- 소스 CSV 파일의 열 이름은 대상 테이블의 열 이름과 일치합니다.
- 데이터는 쉼표로 구분됩니다.
- 모든 데이터 값은 유효한 Amazon Keyspaces 데이터 유형입니다. the section called “데이터 타입” 섹션을 참조하십시오.

2. Amazon Keyspaces에서 대상 키스페이스 및 테이블을 생성합니다.

   a. `cqlsh`를 사용하여 Amazon Keyspaces에 연결하고 다음 예제의 서비스 엔드포인트, 사용자 이름 및 암호를 사용자 고유의 값으로 바꿉니다.

   ```
   cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "111122223333" -
   p "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" --ssl
   ```

   b. 다음 예제와 같이 이름 `catalog`를 사용하여 새 키스페이스를 생성합니다.

   ```
   CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
   ```

   c. 새 키스페이스를 사용할 수 있게 되면 다음 코드를 사용하여 대상 테이블 `book_awards`를 생성합니다.

   ```
   CREATE TABLE "catalog.book_awards" (
      year int,
      award text,
      rank int,
      category text,
      book_title text,
      author text,
      publisher text,
      PRIMARY KEY ((year, award), category, rank)
      );
   ```

Apache Cassandra가 원본 데이터 소스인 경우 헤더가 일치하는 Amazon Keyspaces 대상 테이블을 생성하는 간단한 방법은 다음 문과 같이 소스 테이블에서 `CREATE TABLE` 문을 생성하는 것입니다.

```
cqlsh localhost 9042  -u "username" -p "password" --execute "DESCRIBE
 TABLE mykeyspace.mytable;"
```

그런 다음 Cassandra 소스 테이블의 설명과 일치하는 열 이름 및 데이터 유형을 사용하여 Amazon Keyspaces에 대상 테이블을 생성합니다.

2단계: 데이터 준비

효율적인 전송을 위해 소스 데이터를 준비하는 것은 2단계 프로세스입니다. 먼저 데이터를 무작위화합니다. 두 번째 단계에서는 데이터를 분석하여 적절한 cqlsh 파라미터 값과 필요한 테이블 설정을 결정합니다.

데이터 무작위화

이 cqlsh COPY FROM 명령은 CSV 파일에 나타나는 것과 동일한 순서로 데이터를 읽고 씁니다. cqlsh COPY TO 명령을 사용하여 소스 파일을 만들면 데이터가 키 정렬된 순서대로 CSV에 기록됩니다. Amazon Keyspaces는 내부적으로 파티션 키를 사용하여 데이터를 분할합니다. Amazon Keyspaces에는 동일한 파티션 키에 대한 요청을 로드 밸런싱하는 데 도움이 되는 로직이 내장되어 있지만 순서를 무작위로 지정하면 데이터를 로드하는 것이 더 빠르고 효율적입니다. Amazon Keyspaces가 다른 파티션에 쓸 때 발생하는 내장된 로드 밸런싱을 활용할 수 있기 때문입니다.

쓰기를 파티션 전체에 균등하게 분산하려면 소스 파일의 데이터를 무작위화해야 합니다. 이 작업을 수행하는 애플리케이션을 작성하거나 Shuf와 같은 오픈 소스 도구를 사용할 수 있습니다. Shuf는 Linux 배포판, macOS(homebrew에 coreutils 설치), Windows(Windows Subsystem for Linux(WSL) 사용)에서 무료로 사용할 수 있습니다. 이 단계에서 열 이름이 있는 헤더 행이 섞이지 않도록 하려면 한 가지 추가 단계가 필요합니다.

헤더를 보존하면서 소스 파일을 무작위화하려면 다음 코드를 입력합니다.

```
tail -n +2 keyspaces_sample_table.csv | shuf -o keyspace.table.csv && (head
 -1 keyspaces_sample_table.csv && cat keyspace.table.csv ) > keyspace.table.csv1 &&
 mv keyspace.table.csv1 keyspace.table.csv
```

Shuf는 keyspace.table.csv라는 새 CSV 파일에 데이터를 다시 씁니다. 이제 필요 없는 keyspaces_sample_table.csv 파일을 삭제할 수 있습니다.

데이터 분석

데이터를 분석하여 평균 및 최대 행 크기를 결정합니다.

이렇게 하는 이유는 다음과 같습니다.

- 평균 행 크기는 전송할 총 데이터 양을 추정하는 데 도움이 됩니다.
- 데이터 업로드에 필요한 쓰기 용량을 프로비저닝하려면 평균 행 크기가 필요합니다.
- 각 행의 크기가 Amazon Keyspaces의 최대 행 크기인 1MB 미만인지 확인할 수 있습니다.

> **Note**
>
> 이 할당량은 파티션 크기가 아니라 행 크기를 나타냅니다. Apache Cassandra 파티션과 달리 Amazon Keyspaces 파티션은 크기가 사실상 바인딩되지 않을 수 있습니다. 파티션 키와 클러스터링 열에는 메타데이터를 위한 추가 스토리지가 필요하며 이를 행의 원시 크기에 추가해야 합니다. 자세한 설명은 the section called “행 크기 계산” 섹션을 참조하세요.

다음 코드는 AWK를 사용하여 CSV 파일을 분석하고 평균 및 최대 행 크기를 인쇄합니다.

```
awk -F, 'BEGIN {samp=10000;max=-1;}{if(NR>1){len=length($0);t+=len;avg=t/
NR;max=(len>max ? len : max)}}NR==samp{exit}END{printf("{lines: %d, average: %d bytes,
 max: %d bytes}\n",NR,avg,max);}' keyspace.table.csv
```

이 코드를 실행하면 다음과 같은 결과가 출력됩니다.

```
using 10,000 samples:
{lines: 10000, avg: 123 bytes, max: 225 bytes}
```

이 자습서의 다음 단계에서 평균 행 크기를 사용하여 테이블의 쓰기 용량을 프로비저닝합니다.

3단계: 테이블의 처리량 용량 설정

이 자습서에서는 설정된 시간 범위 내에서 데이터를 로드하도록 cqlsh를 조정하는 방법을 보여 줍니다. 얼마나 많은 읽기와 쓰기를 수행하는지 미리 알고 있기 때문에 프로비저닝된 용량 모드를 사용합니다. 데이터 전송을 완료한 후에는 애플리케이션의 트래픽 패턴에 맞게 테이블의 용량 모드를 설정해야 합니다. 용량 관리에 대한 자세한 내용은 서버리스 리소스 관리 섹션을 참조하세요.

프로비저닝된 용량 모드를 사용하면 테이블에 프로비저닝할 읽기 및 쓰기 용량을 미리 지정할 수 있습니다. 쓰기 용량은 시간당 청구되며 쓰기 용량 단위(WCU)로 측정됩니다. 각 WCU는 초당 1KB의 데이터 쓰기를 지원하기에 충분한 쓰기 용량입니다. 데이터를 로드할 때 쓰기 속도는 대상 테이블에 설정된 최대 WCU(매개 변수: `write_capacity_units`) 미만이어야 합니다.

기본적으로 테이블에 최대 40,000WCU를 프로비저닝하고 계정의 모든 테이블에 80,000WCU를 프로비저닝할 수 있습니다. 추가 용량이 필요한 경우 Service Quotas에서 할당량 증가를 요청할 수 있습니다. 할당량에 대한 자세한 내용은 할당량 섹션을 참조하세요.

삽입에 필요한 WCU의 평균 수를 계산합니다.

초당 1KB의 데이터를 삽입하려면 1WCU가 필요합니다. 360,000개의 행이 있는 CSV 파일에 1시간 내에 모든 데이터를 로드하려면 초당 100개의 행을 작성해야 합니다(360,000행/60분/60초 = 초당 100행). 각 행에 최대 1KB의 데이터가 있는 경우 초당 100개의 행을 삽입하려면 테이블에 100WCU를 프로비저닝해야 합니다. 각 행에 1.5KB의 데이터가 있는 경우 초당 한 행을 삽입하려면 WCU 두 개가 필요합니다. 따라서 초당 100개의 행을 삽입하려면 200개의 WCU를 프로비저닝해야 합니다.

초당 행 하나를 삽입하는 데 필요한 WCU 수를 결정하려면 평균 행 크기(바이트)를 1024로 나누고 가장 가까운 정수로 반올림합니다.

예를 들어 평균 행 크기가 3000바이트인 경우 초당 한 행을 삽입하려면 WCU 3개가 필요합니다.

```
ROUNDUP(3000 / 1024) = ROUNDUP(2.93) = 3 WCUs
```

데이터 로드 시간 및 용량 계산

이제 CSV 파일의 평균 크기와 행 수를 알았으므로 주어진 시간 동안 데이터를 로드하는 데 필요한 WCU 수와 다양한 WCU 설정을 사용하여 CSV 파일의 모든 데이터를 로드하는 데 걸리는 대략적인 시간을 계산할 수 있습니다.

예를 들어 파일의 각 행이 1KB이고 CSV 파일에 1,000,000개의 행이 있는 경우 1시간 내에 데이터를 로드하려면 해당 시간 동안 테이블에 최소 278개의 WCU를 프로비저닝해야 합니다.

```
1,000,000 rows * 1 KBs = 1,000,000 KBs
1,000,000 KBs / 3600 seconds =277.8 KBs / second = 278 WCUs
```

프로비저닝된 용량 설정 구성

테이블을 생성할 때 또는 ALTER TABLE CQL 명령을 사용하여 테이블의 쓰기 용량 설정을 지정할 수 있습니다. 다음은 ALTER TABLE CQL 문을 사용하여 테이블의 프로비저닝된 용량 설정을 변경하는 구문입니다.

```
ALTER TABLE mykeyspace.mytable WITH custom_properties={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 100,
 'write_capacity_units': 278}} ;
```

전체 언어 참조는 the section called “ALTER TABLE” 섹션을 참조하세요.

4단계: `cqlsh COPY FROM` 설정 구성

이 섹션에서는 `cqlsh COPY FROM`의 파라미터 값을 결정하는 방법을 간략하게 설명합니다. `cqlsh COPY FROM` 명령은 이전에 준비한 CSV 파일을 읽고 CQL을 사용하여 Amazon Keyspaces에 데이터를 삽입합니다. 이 명령은 행을 나누어 작업자 집합 간에 `INSERT` 작업을 분배합니다. 각 작업자는 Amazon Keyspaces와 연결을 설정하고 이 채널을 통해 `INSERT` 요청을 보냅니다.

`cqlsh COPY` 명령에는 작업자 간에 작업을 균등하게 분배하기 위한 내부 로직이 없습니다. 하지만 작업이 균등하게 분배되도록 수동으로 구성할 수 있습니다. 먼저 다음과 같은 주요 cqlsh 파라미터를 검토합니다.

- DELIMITER - 쉼표 이외의 구분 기호를 사용한 경우 이 파라미터를 설정할 수 있습니다. 기본값은 쉼표입니다.
- INGESTRATE - `cqlsh COPY FROM`이 초당 처리를 시도하는 대상 행 수입니다. 지정되지 않은 경우 기본값은 100,000입니다.
- NUMPROCESSES - cqlsh가 `COPY FROM` 작업을 위해 생성하는 하위 작업자 프로세스의 수입니다. 이 설정의 최대값은 16이며, 기본값은 `num_cores - 1`입니다. 여기서 `num_cores`는 cqlsh를 실행하는 호스트의 프로세싱 코어 수입니다.
- MAXBATCHSIZE – 배치 크기에 따라 단일 배치에서 대상 테이블에 삽입되는 최대 행 수가 결정됩니다. 설정되지 않은 경우 cqlsh는 삽입된 20개 행의 배치를 사용합니다.
- CHUNKSIZE - 하위 작업자에게 전달되는 작업 단위의 크기입니다. 기본적으로 5,000으로 설정됩니다.
- MAXATTEMPTS - 실패한 작업자 청크를 다시 시도할 수 있는 최대 횟수입니다. 최대 시도 횟수에 도달하면 실패한 레코드가 새 CSV 파일에 기록되며 실패를 조사한 후 나중에 다시 실행할 수 있습니다.

대상 테이블에 프로비저닝한 WCU 수를 기반으로 `INGESTRATE`를 설정합니다. `cqlsh COPY FROM` 명령의 `INGESTRATE`는 제한이 아니라 대상 평균입니다. 즉, 설정한 수치를 초과할 수 있으며 종종 그럴 수도 있습니다. 버스트를 허용하고 데이터 로드 요청을 처리할 수 있는 충분한 용량을 확보하려면 `INGESTRATE`를 테이블 쓰기 용량의 90% 설정합니다.

```
INGESTRATE = WCUs * .90
```

그런 다음 `NUMPROCESSES` 파라미터를 시스템의 코어 수보다 하나 적은 것으로 설정합니다. 다음 코드를 실행하여 시스템의 코어 수를 확인할 수 있습니다.

```
python -c "import multiprocessing; print(multiprocessing.cpu_count())"
```

이 자습서에서는 다음 값을 사용합니다.

```
NUMPROCESSES = 4
```

각 프로세스는 작업자를 생성하고 각 작업자는 Amazon Keyspaces에 대한 연결을 설정합니다. Amazon Keyspaces는 모든 연결에서 초당 최대 3,000개의 CQL 요청을 지원할 수 있습니다. 즉, 각 작업자가 처리하는 요청이 초당 3,000개 미만인지 확인해야 합니다.

INGESTRATE와 마찬가지로 작업자는 설정한 수를 초과하여 버스트하는 경우가 많으며 클록 초로 제한되지 않습니다. 따라서 버스트를 고려하려면 각 작업자가 초당 2,500개의 요청을 처리하게 지정하도록 cqlsh 파라미터를 설정합니다. 작업자에게 분배된 작업량을 계산하려면 다음 지침을 사용합니다.

- INGESTRATE를 NUMPROCESSES로 나눕니다.
- INGESTRATE/NUMPROCESSES > 2,500인 경우 INGESTRATE를 낮추면 이 공식이 true가 됩니다.

```
INGESTRATE / NUMPROCESSES <= 2,500
```

샘플 데이터의 업로드를 최적화하도록 설정을 구성하기 전에 cqlsh 기본 설정을 검토하고 기본 설정을 사용하는 것이 데이터 업로드 프로세스에 어떤 영향을 미치는지 살펴보겠습니다. cqlsh COPY FROM은 CHUNKSIZE를 사용하여 작업자에게 배포할 작업 청크(INSERT 문)를 만들기 때문에 작업이 자동으로 균등하게 분배되지 않습니다. INGESTRATE 설정에 따라 일부 작업자는 가만히 앉아 있을 수 있습니다.

작업자 간에 작업을 균등하게 분배하고 각 작업자가 초당 2,500개의 요청 속도를 최적화할 수 있도록 하려면 입력 파라미터를 변경하여 CHUNKSIZE, MAXBATCHSIZE 및 INGESTRATE를 설정해야 합니다. 데이터 로드 중에 네트워크 트래픽 사용률을 최적화하려면 MAXBATCHSIZE에 대해 최대값인 30에 가까운 값을 선택합니다. CHUNKSIZE를 100으로 MAXBATCHSIZE를 25로 변경하면 10,000개의 행이 네 명의 작업자 사이에 균등하게 분산됩니다(10,000/2500 = 4).

다음 코드 예제에서는 이를 보여 줍니다.

```
INGESTRATE = 10,000
NUMPROCESSES = 4
CHUNKSIZE = 100
MAXBATCHSIZE. = 25
Work Distribution:
```

```
Connection 1 / Worker 1 : 2,500 Requests per second
Connection 2 / Worker 2 : 2,500 Requests per second
Connection 3 / Worker 3 : 2,500 Requests per second
Connection 4 / Worker 4 : 2,500 Requests per second
```

요약하면 `cqlsh COPY FROM` 파라미터를 설정할 때 다음 공식을 사용합니다.

- INGESTRATE = write_capacity_units * .90
- NUMPROCESSES = num_cores -1(기본값)
- INGESTRATE / NUMPROCESSES = 2,500(true 문이어야 합니다.)
- MAXBATCHSIZE = 30(기본값은 20입니다. Amazon Keyspaces는 최대 30개의 배치를 허용합니다.)
- CHUNKSIZE = (INGESTRATE / NUMPROCESSES) / MAXBATCHSIZE

`NUMPROCESSES`, `INGESTRATE` 및 `CHUNKSIZE` 계산을 마쳤고 이제 데이터를 로드할 준비가 되었습니다.

5단계: **`cqlsh COPY FROM`** 명령 실행

`cqlsh COPY FROM` 명령을 실행하려면 다음 단계를 완료합니다.

1. cqlsh를 사용하여 Amazon Keyspaces에 연결합니다.
2. 다음 코드를 사용하여 키스페이스를 선택합니다.

   ```
   USE catalog;
   ```

3. 쓰기 일관성을 `LOCAL_QUORUM`으로 설정합니다. 데이터 내구성을 보장하기 위해 Amazon Keyspaces는 다른 쓰기 일관성 설정을 허용하지 않습니다. 다음 코드를 확인합니다.

   ```
   CONSISTENCY LOCAL_QUORUM;
   ```

4. 다음 코드 예제를 사용하여 `cqlsh COPY FROM` 구문을 준비합니다.

   ```
   COPY book_awards FROM './keyspace.table.csv' WITH HEADER=true
   AND INGESTRATE=calculated ingestrate
   AND NUMPROCESSES=calculated numprocess
   AND MAXBATCHSIZE=20
   AND CHUNKSIZE=calculated chunksize;
   ```

5. 이전 단계에서 준비한 문을 실행합니다. cqlsh는 구성한 모든 설정을 다시 실행합니다.

a. 설정이 입력과 일치하는지 확인합니다. 다음 예를 참조하세요.

```
Reading options from the command line: {'chunksize': '120', 'header': 'true',
 'ingestrate': '36000', 'numprocesses': '15', 'maxbatchsize': '20'}
Using 15 child processes
```

b. 다음 예와 같이 전송된 행 수와 현재 평균 비율을 검토합니다.

```
Processed: 57834 rows; Rate: 6561 rows/s; Avg. rate: 31751 rows/s
```

c. cqlsh가 데이터 업로드를 완료하면 다음 예와 같이 데이터 로드 통계 요약(읽은 파일 수, 런타임, 건너뛴 행 수)을 검토합니다.

```
15556824 rows imported from 1 files in 8 minutes and 8.321 seconds (0 skipped).
```

자습서의 마지막 단계에서는 Amazon Keyspaces에 데이터를 업로드했습니다.

> **Important**
>
> 이제 데이터를 전송했으니 대상 테이블의 용량 모드 설정을 애플리케이션의 일반 트래픽 패턴에 맞게 조정합니다. 용량을 변경하기 전까지는 프로비저닝된 용량에 대해 시간당 요금이 부과됩니다.

문제 해결

데이터 업로드가 완료된 후 행을 건너뛰었는지 확인합니다. 이렇게 하려면 원본 CSV 파일의 소스 디렉터리로 이동하여 다음 이름의 파일을 검색합니다.

```
import_yourcsvfilename.err.timestamp.csv
```

cqlsh는 건너뛴 데이터 행을 해당 이름의 파일에 기록합니다. 파일이 소스 디렉터리에 있고 그 안에 데이터가 있는 경우 이 행은 Amazon Keyspaces에 업로드되지 않았습니다. 이러한 행을 다시 시도하려면 먼저 업로드 중에 발생한 오류가 있는지 확인하고 그에 따라 데이터를 조정합니다. 이러한 행을 다시 시도하려면 프로세스를 다시 실행하면 됩니다.

일반적인 오류

행이 로드되지 않는 가장 일반적인 이유는 용량 오류와 구문 분석 오류입니다.

Amazon Keyspaces에 데이터를 업로드할 때 잘못된 요청 오류

다음 예제의 소스 테이블에는 카운터 열이 포함되어 있으며 이로 인해 cqlsh COPY 명령의 배치 호출이 로그됩니다. 로그된 배치 호출은 Amazon Keyspaces에서 지원되지 않습니다.

```
Failed to import 10 rows: InvalidRequest - Error from server: code=2200 [Invalid query]
 message="Only UNLOGGED Batches are supported at this time.",  will retry later,
 attempt 22 of 25
```

이 오류를 해결하려면 DSBulk를 사용하여 데이터를 마이그레이션합니다. 자세한 설명은 the section called "DSBulk를 사용하여 데이터 로드" 섹션을 참조하세요.

Amazon Keyspaces에 데이터를 업로드할 때 구문 분석기 오류

다음 예는 ParseError로 인해 건너뛴 행을 보여 줍니다.

```
Failed to import 1 rows: ParseError - Invalid ... –
```

이 오류를 해결하려면 가져올 데이터가 Amazon Keyspaces의 테이블 스키마와 일치하는지 확인해야 합니다. 가져오기 파일에 구문 분석 오류가 있는지 검토합니다. 오류를 격리하는 INSERT 문을 사용하여 단일 데이터 행을 사용해 볼 수 있습니다.

Amazon Keyspaces에 데이터를 업로드할 때 용량 오류

```
Failed to import 1 rows: WriteTimeout - Error from server: code=1100 [Coordinator node
 timed out waiting for replica nodes' responses]
 message="Operation timed out - received only 0 responses." info={'received_responses':
 0, 'required_responses': 2, 'write_type': 'SIMPLE', 'consistency':
 'LOCAL_QUORUM'}, will retry later, attempt 1 of 100
```

Amazon Keyspaces는 처리량 용량이 충분하지 않아 쓰기 요청이 실패하는 경우를 나타내기 위해 ReadTimeout 및 WriteTimeout 예외를 사용합니다. 용량 부족 예외를 진단하는 데 도움이 되도록 Amazon Keyspace는 ReadThrottledEvents Amazon에 WriteThrottleEvents 게시하고 지표를 제공합니다. CloudWatch 자세한 설명은 the section called "를 통한 모니터링 CloudWatch" 섹션을 참조하세요.

Amazon Keyspaces에 데이터를 업로드할 때 cqlsh 오류

cqlsh 오류를 해결하는 데 도움이 되도록 --debug 플래그를 사용하여 실패한 명령을 다시 실행합니다.

호환되지 않는 버전의 cqlsh를 사용하는 경우 다음 오류가 표시됩니다.

```
AttributeError: 'NoneType' object has no attribute 'is_up'
Failed to import 3 rows: AttributeError - 'NoneType' object has no attribute 'is_up',
 given up after 1 attempts
```

다음 명령을 실행하여 올바른 cqlsh 버전이 설치되어 있는지 확인합니다.

```
cqlsh --version
```

출력에 대해 다음과 같은 내용이 표시되어야 합니다.

```
cqlsh 5.0.1
```

Windows를 사용하는 경우 `cqlsh`의 모든 인스턴스를 `cqlsh.bat`로 바꿉니다. 예를 들어 Windows에서 cqlsh 버전을 확인하려면 다음 명령을 실행합니다.

```
cqlsh.bat --version
```

cqlsh 클라이언트가 서버로부터 유형에 상관없이 연속으로 세 개의 오류를 수신한 후 Amazon Keyspaces에 대한 연결이 실패합니다. cqlsh 클라이언트가 실패하고 다음 메시지가 표시됩니다.

```
Failed to import 1 rows: NoHostAvailable - , will retry later, attempt 3 of 100
```

이 오류를 해결하려면 가져올 데이터가 Amazon Keyspaces의 테이블 스키마와 일치하는지 확인해야 합니다. 가져오기 파일에 구문 분석 오류가 있는지 검토합니다. 오류를 격리하는 INSERT 문을 사용하여 단일 데이터 행을 사용해 볼 수 있습니다.

클라이언트는 자동으로 연결 재설정을 시도합니다.

자습서: DSBulk를 사용하여 Amazon Keyspaces에 데이터 로드

이 step-by-step 자습서는 에서 제공되는 대량 로더 (DSBulk) 를 사용하여 Apache Cassandra에서 Amazon Keyspaces로 데이터를 마이그레이션하는 방법을 DataStax 안내합니다. GitHub 이 자습서에서는 다음 단계를 완료합니다.

주제

- 필수 조건
- 1단계: 소스 CSV 파일 및 대상 테이블 생성

- 2단계: 데이터 준비
- 3단계: 테이블의 처리량 용량 설정
- 4단계: DSBulk 설정 구성
- 5단계: DSBulk load 명령 실행

필수 조건

이 자습서를 시작하려면 먼저 다음 작업을 완료해야 합니다.

1. 아직 계정을 등록하지 않았다면 의 단계에 따라 계정을 등록하십시오. AWS the section called “설 AWS Identity and Access Management정”
2. the section called “인증을 위한 IAM 자격 증명 AWS ”의 단계에 따라 보안 인증을 생성합니다.
3. JKS 신뢰 저장소 파일을 생성합니다.

   a. 다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드하고 `sf-class2-root.crt`를 로컬 또는 홈 디렉터리에 저장합니다.

   ```
   curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
   ```

   > **Note**
   >
   > 또한 Amazon 디지털 인증서를 사용하여 Amazon Keyspaces에 접속할 수 있으며, 클라이언트가 Amazon Keyspaces에 성공적으로 접속하면 계속 접속할 수 있습니다. Starfield 인증서는 이전 인증 기관을 사용하는 클라이언트에게 추가적인 이전 버전과의 호환성을 제공합니다.

   b. Starfield 디지털 인증서를 trustStore 파일로 변환합니다.

   ```
   openssl x509 -outform der -in sf-class2-root.crt -out temp_file.der
   keytool -import -alias cassandra -keystore cassandra_truststore.jks -file
    temp_file.der
   ```

   이 단계에서는 키스토어의 암호를 생성하고 이 인증서를 신뢰해야 합니다. 대화형 명령은 다음과 같습니다.

   ```
   Enter keystore password:
   Re-enter new password:
   ```

```
Owner: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
 Inc.", C=US
Issuer: OU=Starfield Class 2 Certification Authority, O="Starfield
 Technologies, Inc.", C=US
Serial number: 0
Valid from: Tue Jun 29 17:39:16 UTC 2004 until: Thu Jun 29 17:39:16 UTC 2034
Certificate fingerprints:
  MD5:  32:4A:4B:BB:C8:63:69:9B:BE:74:9A:C6:DD:1D:46:24
  SHA1: AD:7E:1C:28:B0:64:EF:8F:60:03:40:20:14:C3:D0:E3:37:0E:B5:8A
  SHA256:
 14:65:FA:20:53:97:B8:76:FA:A6:F0:A9:95:8E:55:90:E4:0F:CC:7F:AA:4F:B7:C2:C8:67:75:21:FB
Signature algorithm name: SHA1withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3
Extensions:
#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86   F4 5B 55 AC DC D7 10 C2  ._.......[U.....
0010: 0E A9 88 E7                                        ....
]
[OU=Starfield Class 2 Certification Authority, O="Starfield Technologies,
 Inc.", C=US]
SerialNumber: [    00]
]
#2: ObjectId: 2.5.29.19 Criticality=false
BasicConstraints:[
  CA:true
  PathLen:2147483647
]
#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: BF 5F B7 D1 CE DD 1F 86   F4 5B 55 AC DC D7 10 C2  ._.......[U.....
0010: 0E A9 88 E7                                        ....
]
]
Trust this certificate? [no]:  y
```

4. Cassandra 쿼리 언어 쉘(cqlsh) 연결을 설정하고 the section called “cqlsh 사용하기”의 단계에 따라 Amazon Keyspaces에 연결할 수 있는지 확인합니다.
5. DSBulk를 다운로드하여 설치합니다.

a. DSBulk를 다운로드하려면 다음 코드를 사용할 수 있습니다.

```
curl -OL https://downloads.datastax.com/dsbulk/dsbulk-1.8.0.tar.gz
```

b. 그런 후 다음 예제와 같이 tar 파일의 압축을 풀고 PATH에 DSBulk를 추가합니다.

```
tar -zxvf dsbulk-1.8.0.tar.gz
# add the DSBulk directory to the path
export PATH=$PATH:./dsbulk-1.8.0/bin
```

c. DSBulk에서 사용할 설정을 저장할 `application.conf` 파일을 생성합니다. `./dsbulk_keyspaces.conf`로 다음 예제를 저장할 수 있습니다. 로컬 노드에 있지 않은 경우 `localhost`를 로컬 Cassandra 클러스터의 연락처(예: DNS 이름 또는 IP 주소)로 바꿉니다. 나중에 `dsbulk load` 명령에서 지정해야 하므로 파일 이름과 경로를 기록해 둡니다.

```
datastax-java-driver {
  basic.contact-points = [ "localhost"]
  advanced.auth-provider {
        class = software.aws.mcs.auth.SigV4AuthProvider
        aws-region = us-east-1
  }
}
```

d. SiGV4 지원을 활성화하려면 다음 GitHub예와 같이 음영 처리된 `jar` 파일을 에서 다운로드하여 DSBulk `lib` 폴더에 저장합니다.

```
curl -O -L https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin/
releases/download/4.0.6-shaded-v2/aws-sigv4-auth-cassandra-java-driver-
plugin-4.0.6-shaded.jar
```

1단계: 소스 CSV 파일 및 대상 테이블 생성

이 자습서에서는 이름 `keyspaces_sample_table.csv`가 있는 쉼표로 구분된 값(CSV) 파일을 데이터 마이그레이션의 원본 파일로 사용합니다. 제공된 샘플 파일에는 이름이 `book_awards`인 테이블에 대한 몇 개의 데이터 행이 포함되어 있습니다.

1. 소스 파일을 생성합니다. 다음 옵션 중 하나를 선택할 수 있습니다.

- 다음 아카이브 파일 samplemigration.zip에 포함된 샘플 CSV 파일(keyspaces_sample_table.csv)을 다운로드합니다. 아카이브의 압축을 풀고 keyspaces_sample_table.csv의 경로를 기록해 둡니다.
- Apache Cassandra 데이터베이스에 저장된 자체 데이터로 CSV 파일을 채우려면 다음 예와 같이 dsbulk unload를 사용하여 소스 CSV 파일을 채울 수 있습니다.

```
dsbulk unload -k mykeyspace -t mytable -f ./my_application.conf
 > keyspaces_sample_table.csv
```

생성한 CSV 파일이 다음 요구 사항을 충족하는지 확인합니다.

- 첫 번째 행에는 열 이름이 포함됩니다.
- 소스 CSV 파일의 열 이름은 대상 테이블의 열 이름과 일치합니다.
- 데이터는 쉼표로 구분됩니다.
- 모든 데이터 값은 유효한 Amazon Keyspaces 데이터 유형입니다. the section called “데이터 타입” 섹션을 참조하십시오.

2. Amazon Keyspaces에서 대상 키스페이스 및 테이블을 생성합니다.

a. cqlsh를 사용하여 Amazon Keyspaces에 연결하고 다음 예제의 서비스 엔드포인트, 사용자 이름 및 암호를 사용자 고유의 값으로 바꿉니다.

```
cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "111122223333" -
p "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" --ssl
```

b. 다음 예제와 같이 이름 catalog를 사용하여 새 키스페이스를 생성합니다.

```
CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
```

c. 새 키스페이스가 사용 가능 상태가 되면 다음 코드를 사용하여 대상 테이블 book_awards를 생성합니다. 비동기 리소스 생성 및 리소스가 사용 가능한지 확인하는 방법에 대한 자세한 내용은 the section called “키스페이스 생성” 섹션을 참조하세요.

```
CREATE TABLE catalog.book_awards (
    year int,
    award text,
    rank int,
    category text,
    book_title text,
```

```
    author text,
    publisher text,
    PRIMARY KEY ((year, award), category, rank)
    );
```

Apache Cassandra가 원본 데이터 소스인 경우 헤더가 일치하는 Amazon Keyspaces 대상 테이블을 생성하는 간단한 방법은 다음 문과 같이 소스 테이블에서 CREATE TABLE 문을 생성하는 것입니다.

```
cqlsh localhost 9042  -u "username" -p "password" --execute "DESCRIBE
 TABLE mykeyspace.mytable;"
```

그런 다음 Cassandra 소스 테이블의 설명과 일치하는 열 이름 및 데이터 유형을 사용하여 Amazon Keyspaces에 대상 테이블을 생성합니다.

2단계: 데이터 준비

효율적인 전송을 위해 소스 데이터를 준비하는 것은 2단계 프로세스입니다. 먼저 데이터를 무작위화합니다. 두 번째 단계에서는 데이터를 분석하여 적절한 dsbulk 파라미터 값과 필요한 테이블 설정을 결정합니다.

데이터 무작위화

이 dsbulk 명령은 CSV 파일에 나타나는 것과 동일한 순서로 데이터를 읽고 씁니다. dsbulk 명령을 사용하여 소스 파일을 만들면 데이터가 키 정렬된 순서대로 CSV에 기록됩니다. Amazon Keyspaces는 내부적으로 파티션 키를 사용하여 데이터를 분할합니다. Amazon Keyspaces에는 동일한 파티션 키에 대한 요청을 로드 밸런싱하는 데 도움이 되는 로직이 내장되어 있지만 순서를 무작위로 지정하면 데이터를 로드하는 것이 더 빠르고 효율적입니다. Amazon Keyspaces가 다른 파티션에 쓸 때 발생하는 내장된 로드 밸런싱을 활용할 수 있기 때문입니다.

쓰기를 파티션 전체에 균등하게 분산하려면 소스 파일의 데이터를 무작위화해야 합니다. 이 작업을 수행하는 애플리케이션을 작성하거나 Shuf와 같은 오픈 소스 도구를 사용할 수 있습니다. Shuf는 Linux 배포판, macOS(homebrew에 coreutils 설치), Windows(Windows Subsystem for Linux(WSL) 사용)에서 무료로 사용할 수 있습니다. 이 단계에서 열 이름이 있는 헤더 행이 섞이지 않도록 하려면 한 가지 추가 단계가 필요합니다.

헤더를 보존하면서 소스 파일을 무작위화하려면 다음 코드를 입력합니다.

```
tail -n +2 keyspaces_sample_table.csv | shuf -o keyspace.table.csv && (head
 -1 keyspaces_sample_table.csv && cat keyspace.table.csv ) > keyspace.table.csv1 &&
 mv keyspace.table.csv1 keyspace.table.csv
```

Shuf는 `keyspace.table.csv`라는 새 CSV 파일에 데이터를 다시 씁니다. 이제 필요 없는 `keyspaces_sample_table.csv` 파일을 삭제할 수 있습니다.

데이터 분석

데이터를 분석하여 평균 및 최대 행 크기를 결정합니다.

이렇게 하는 이유는 다음과 같습니다.

- 평균 행 크기는 전송할 총 데이터 양을 추정하는 데 도움이 됩니다.
- 데이터 업로드에 필요한 쓰기 용량을 프로비저닝하려면 평균 행 크기가 필요합니다.
- 각 행의 크기가 Amazon Keyspaces의 최대 행 크기인 1MB 미만인지 확인할 수 있습니다.

> **Note**
>
> 이 할당량은 파티션 크기가 아니라 행 크기를 나타냅니다. Apache Cassandra 파티션과 달리 Amazon Keyspaces 파티션은 크기가 사실상 바인딩되지 않을 수 있습니다. 파티션 키와 클러스터링 열에는 메타데이터를 위한 추가 스토리지가 필요하며 이를 행의 원시 크기에 추가해야 합니다. 자세한 정보는 the section called “행 크기 계산”을 참조하세요.

다음 코드는 AWK를 사용하여 CSV 파일을 분석하고 평균 및 최대 행 크기를 인쇄합니다.

```
awk -F, 'BEGIN {samp=10000;max=-1;}{if(NR>1){len=length($0);t+=len;avg=t/
NR;max=(len>max ? len : max)}}NR==samp{exit}END{printf("{lines: %d, average: %d bytes,
 max: %d bytes}\n",NR,avg,max);}' keyspace.table.csv
```

이 코드를 실행하면 다음과 같은 결과가 출력됩니다.

```
using 10,000 samples:
{lines: 10000, avg: 123 bytes, max: 225 bytes}
```

최대 행 크기가 1MB를 초과하지 않는지 확인합니다. 그럴 경우 행을 분할하거나 데이터를 압축하여 행 크기를 1MB 미만으로 가져와야 합니다. 이 자습서의 다음 단계에서 평균 행 크기를 사용하여 테이블의 쓰기 용량을 프로비저닝합니다.

3단계: 테이블의 처리량 용량 설정

이 자습서에서는 설정된 시간 범위 내에서 데이터를 로드하도록 DSBulk 를 조정하는 방법을 보여 줍니다. 얼마나 많은 읽기와 쓰기를 수행하는지 미리 알고 있기 때문에 프로비저닝된 용량 모드를 사용합니다. 데이터 전송을 완료한 후에는 애플리케이션의 트래픽 패턴에 맞게 테이블의 용량 모드를 설정해야 합니다. 용량 관리에 대한 자세한 내용은 서버리스 리소스 관리 섹션을 참조하세요.

프로비저닝된 용량 모드를 사용하면 테이블에 프로비저닝할 읽기 및 쓰기 용량을 미리 지정할 수 있습니다. 쓰기 용량은 시간당 청구되며 쓰기 용량 단위(WCU)로 측정됩니다. 각 WCU는 초당 1KB의 데이터 쓰기를 지원하기에 충분한 쓰기 용량입니다. 데이터를 로드할 때 쓰기 속도는 대상 테이블에 설정된 최대 WCU(매개 변수: `write_capacity_units`) 미만이어야 합니다.

기본적으로 테이블에 최대 40,000WCU를 프로비저닝하고 계정의 모든 테이블에 80,000WCU를 프로비저닝할 수 있습니다. 추가 용량이 필요한 경우 Service Quotas에서 할당량 증가를 요청할 수 있습니다. 할당량에 대한 자세한 내용은 할당량 섹션을 참조하세요.

삽입에 필요한 WCU의 평균 수를 계산합니다.

초당 1KB의 데이터를 삽입하려면 1WCU가 필요합니다. 360,000개의 행이 있는 CSV 파일에 1시간 내에 모든 데이터를 로드하려면 초당 100개의 행을 작성해야 합니다(360,000행/60분/60초 = 초당 100행). 각 행에 최대 1KB의 데이터가 있는 경우 초당 100개의 행을 삽입하려면 테이블에 100WCU를 프로비저닝해야 합니다. 각 행에 1.5KB의 데이터가 있는 경우 초당 한 행을 삽입하려면 WCU 두 개가 필요합니다. 따라서 초당 100개의 행을 삽입하려면 200개의 WCU를 프로비저닝해야 합니다.

초당 행 하나를 삽입하는 데 필요한 WCU 수를 결정하려면 평균 행 크기(바이트)를 1024로 나누고 가장 가까운 정수로 반올림합니다.

예를 들어 평균 행 크기가 3000바이트인 경우 초당 한 행을 삽입하려면 WCU 3개가 필요합니다.

```
ROUNDUP(3000 / 1024) = ROUNDUP(2.93) = 3 WCUs
```

데이터 로드 시간 및 용량 계산

이제 CSV 파일의 평균 크기와 행 수를 알았으므로 주어진 시간 동안 데이터를 로드하는 데 필요한 WCU 수와 다양한 WCU 설정을 사용하여 CSV 파일의 모든 데이터를 로드하는 데 걸리는 대략적인 시간을 계산할 수 있습니다.

예를 들어 파일의 각 행이 1KB이고 CSV 파일에 1,000,000개의 행이 있는 경우 1시간 내에 데이터를 로드하려면 해당 시간 동안 테이블에 최소 278개의 WCU를 프로비저닝해야 합니다.

```
1,000,000 rows * 1 KBs = 1,000,000 KBs
1,000,000 KBs / 3600 seconds =277.8 KBs / second = 278 WCUs
```

프로비저닝된 용량 설정 구성

테이블을 생성할 때 또는 ALTER TABLE 명령을 사용하여 테이블의 쓰기 용량 설정을 지정할 수 있습니다. 다음은 ALTER TABLE 명령을 사용하여 테이블의 프로비저닝된 용량 설정을 변경하는 구문입니다.

```
ALTER TABLE catalog.book_awards WITH custom_properties={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 100, 'write_capacity_units':
 278}} ;
```

전체 언어 참조는 the section called “CREATE TABLE” 및 the section called “ALTER TABLE” 섹션을 참조하세요.

4단계: DSBulk 설정 구성

이 섹션에서는 Amazon Keyspaces에 데이터를 업로드하기 위해 DSBulk를 구성하는 데 필요한 단계를 간략하게 설명합니다. 구성 파일을 사용하여 DSBulk를 구성합니다. 구성 파일은 명령줄에서 직접 지정합니다.

1. Amazon Keyspaces로 마이그레이션하기 위한 DSBulk 구성 파일을 생성합니다. 이 예에서는 파일 이름 dsbulk_keyspaces.conf를 사용합니다. DSBulk 구성 파일에 다음 설정을 지정합니다.

   a. *PlainTextAuthProvider* - PlainTextAuthProvider 클래스를 사용하여 인증 제공자를 생성합니다. ServiceUserName 및 ServicePassword는 the section called “보안 인증 정보 생성”의 단계에 따라 서비스별 보안 인증을 생성할 때 얻은 사용자 이름 및 암호와 일치해야 합니다.

   b. *local-datacenter*— 의 값을 연결 AWS 리전 중인 local-datacenter 파일로 설정합니다. 예를 들어 애플리케이션이 cassandra.us-east-2.amazonaws.com에 연결 중인 경우 로컬 데이터 센터를 us-east-2로 설정합니다. 사용 가능한 모든 항목에 AWS 리전대해서는 을 참조하십시오the section called “서비스 엔드포인트”. 복제본을 피하려면 slow-replica-avoidance를 false로 설정합니다.

   c. *SSLEngineFactory* - SSL/TLS를 구성하려면 class = DefaultSslEngineFactory로 클래스를 지정하는 한 줄로 구성 파일에 섹션을 추가하여 SSLEngineFactory를 초기화합니다. cassandra_truststore.jks의 경로와 이전에 만든 암호를 입력합니다.

d. *consistency* - 일관성 수준을 LOCAL_QUORUM으로 설정합니다. 다른 쓰기 일관성 수준은 지원되지 않습니다. 자세한 내용은 the section called “지원되는 Cassandra 일관성 수준” 섹션을 참조하세요.

e. Java 드라이버에서 풀당 연결 수를 구성할 수 있습니다. 이 예제에서는 advanced.connection.pool.local.size를 3으로 설정합니다.

다음은 전체 샘플 구성 파일입니다.

```
datastax-java-driver {
basic.contact-points = [ "cassandra.us-east-2.amazonaws.com:9142"]
advanced.auth-provider {
    class = PlainTextAuthProvider
    username = "ServiceUserName"
    password = "ServicePassword"
}

basic.load-balancing-policy {
    local-datacenter = "us-east-2"
    slow-replica-avoidance = false
}

basic.request {
    consistency = LOCAL_QUORUM
    default-idempotence = true
}
advanced.ssl-engine-factory {
    class = DefaultSslEngineFactory
    truststore-path = "./cassandra_truststore.jks"
    truststore-password = "my_password"
    hostname-validation = false
  }
advanced.connection.pool.local.size = 3
}
```

2. DSBulk load 명령의 파라미터를 검토합니다.

a. *executor.maxPerSecond* - load 명령이 초당 동시에 처리하려고 시도하는 최대 행 수입니다. 설정하지 않으면 이 설정이 -1로 비활성화됩니다.

대상 테이블에 프로비저닝한 WCU 수를 기반으로 executor.maxPerSecond를 설정합니다. load 명령의 executor.maxPerSecond는 제한이 아니라 대상 평균입니다. 즉, 설정한

수치를 초과할 수 있으며 종종 그럴 수도 있습니다. 버스트를 허용하고 데이터 로드 요청을 처리할 수 있는 충분한 용량을 확보하려면 executor.maxPerSecond를 테이블 쓰기 용량의 90% 설정합니다.

```
executor.maxPerSecond = WCUs * .90
```

이 자습서에서는 executor.maxPerSecond를 5로 설정했습니다.

> **Note**
>
> DSBulk 1.6.0 이상을 사용하는 경우 대신 dsbulk.engine.maxConcurrentQueries를 사용할 수 있습니다.

b. DSBulk load 명령에 대해 이러한 추가 파라미터를 구성합니다.

- *batch-mode* - 이 파라미터는 파티션 키별로 작업을 그룹화하도록 시스템에 지시합니다. 핫 키 시나리오와 원인이 발생할 수 있으므로 배치 모드를 비활성화하는 것이 좋습니다WriteThrottleEvents.
- *driver.advanced.retry-policy-max-retries* - 실패한 쿼리를 재시도할 횟수를 결정합니다. 설정하지 않은 경우 기본값은 10입니다. 필요에 따라 이 값을 조정할 수 있습니다.
- *driver.basic.request.timeout* - 시스템에서 쿼리가 반환되기를 기다리는 시간(분)입니다. 설정하지 않은 경우 기본값은 "5분"입니다. 필요에 따라 이 값을 조정할 수 있습니다.

5단계: DSBulk **load** 명령 실행

이 자습서의 마지막 단계에서는 Amazon Keyspaces에 데이터를 업로드합니다.

DSBulk load 명령을 실행하려면 다음 단계를 완료합니다.

1. 다음 코드를 실행하여 csv 파일의 데이터를 Amazon Keyspaces 테이블에 업로드합니다. 이전에 생성한 애플리케이션 구성 파일의 경로를 업데이트해야 합니다.

```
dsbulk load -f ./dsbulk_keyspaces.conf  --connector.csv.url keyspace.table.csv
 -header true --batch.mode DISABLED --executor.maxPerSecond 5 --
driver.basic.request.timeout "5 minutes" --driver.advanced.retry-policy.max-
retries 10 -k catalog -t book_awards
```

2. 출력에는 성공한 작업과 실패한 작업을 자세히 설명하는 로그 파일의 위치가 포함됩니다. 이 파일은 다음 디렉터리에 저장됩니다.

```
Operation directory: /home/user_name/logs/UNLOAD_20210308-202317-801911
```

3. 로그 파일 항목에는 다음 예와 같은 지표가 포함됩니다. 행 수가 csv 파일의 행 수와 일치하는지 확인합니다.

```
total | failed | rows/s | p50ms | p99ms | p999ms
   200 |      0 |    200 | 21.63 | 21.89 |  21.89
```

Important

이제 데이터를 전송했으니 대상 테이블의 용량 모드 설정을 애플리케이션의 일반 트래픽 패턴에 맞게 조정합니다. 용량을 변경하기 전까지는 프로비저닝된 용량에 대해 시간당 요금이 부과됩니다. 자세한 내용은 the section called “읽기/쓰기 용량 모드”을(를) 참조하세요.

SDK를 사용하는 Amazon 키스페이스의 코드 예제 AWS

다음 코드 예제는 AWS 소프트웨어 개발 키트 (SDK) 와 함께 Amazon Keyspaces를 사용하는 방법을 보여줍니다.

작업은 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 작업은 개별 서비스 함수를 호출하는 방법을 보여 주며 관련 시나리오와 교차 서비스 예시에서 컨텍스트에 맞는 작업을 볼 수 있습니다.

시나리오는 동일한 서비스 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여주는 코드 예시입니다.

AWS SDK 개발자 안내서 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

시작하기

Hello Amazon Keyspaces

다음 코드 예시에서는 Amazon Keyspaces를 시작하는 방법을 보여줍니다.

.NET

AWS SDK for .NET

> **Note**
>
> 자세한 내용은 에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
namespace KeyspacesActions;

public class HelloKeyspaces
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
```

```
        // Set up dependency injection for Amazon Keyspaces (for Apache
 Cassandra).
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
                services.AddAWSService<IAmazonKeyspaces>()
                .AddTransient<KeyspacesWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<HelloKeyspaces>();

        var keyspacesClient =
 host.Services.GetRequiredService<IAmazonKeyspaces>();
        var keyspacesWrapper = new KeyspacesWrapper(keyspacesClient);

        Console.WriteLine("Hello, Amazon Keyspaces! Let's list your keyspaces:");
        await keyspacesWrapper.ListKeyspaces();
    }
}
```

- API 세부 정보는 AWS SDK for .NET API ListKeyspaces참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.keyspaces.KeyspacesClient;
import software.amazon.awssdk.services.keyspaces.model.KeyspaceSummary;
import software.amazon.awssdk.services.keyspaces.model.KeyspacesException;
import software.amazon.awssdk.services.keyspaces.model.ListKeyspacesRequest;
import software.amazon.awssdk.services.keyspaces.model.ListKeyspacesResponse;
import java.util.List;

/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class HelloKeyspaces {
    public static void main(String[] args) {
        Region region = Region.US_EAST_1;
        KeyspacesClient keyClient = KeyspacesClient.builder()
                .region(region)
                .build();

        listKeyspaces(keyClient);
    }

    public static void listKeyspaces(KeyspacesClient keyClient) {
        try {
            ListKeyspacesRequest keyspacesRequest =
 ListKeyspacesRequest.builder()
                    .maxResults(10)
                    .build();

            ListKeyspacesResponse response =
 keyClient.listKeyspaces(keyspacesRequest);
            List<KeyspaceSummary> keyspaces = response.keyspaces();
            for (KeyspaceSummary keyspace : keyspaces) {
                System.out.println("The name of the keyspace is " +
 keyspace.keyspaceName());
            }

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
```

```
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API ListKeyspaces참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
Before running this Kotlin code example, set up your development environment,
 including your credentials.

For more information, see the following documentation topic:

https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
*/

suspend fun main() {
    listKeyspaces()
}

suspend fun listKeyspaces() {
    val keyspacesRequest =
        ListKeyspacesRequest {
            maxResults = 10
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.listKeyspaces(keyspacesRequest)
        response.keyspaces?.forEach { keyspace ->
            println("The name of the keyspace is ${keyspace.keyspaceName}")
        }
```

```
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 ListKeyspaces.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
import boto3


def hello_keyspaces(keyspaces_client):
    """
    Use the AWS SDK for Python (Boto3) to create an Amazon Keyspaces (for Apache
 Cassandra)
    client and list the keyspaces in your account.
    This example uses the default settings specified in your shared credentials
    and config files.

    :param keyspaces_client: A Boto3 Amazon Keyspaces Client object. This object
 wraps
                             the low-level Amazon Keyspaces service API.
    """
    print("Hello, Amazon Keyspaces! Let's list some of your keyspaces:\n")
    for ks in keyspaces_client.list_keyspaces(maxResults=5).get("keyspaces", []):
        print(ks["keyspaceName"])
        print(f"\t{ks['resourceArn']}")


if __name__ == "__main__":
    hello_keyspaces(boto3.client("keyspaces"))
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 ListKeyspaces.

코드 예시

- SDK를 사용한 Amazon 키스페이스 관련 작업 AWS
  - AWS SDK 또는 CreateKeyspace CLI와 함께 사용
  - AWS SDK 또는 CreateTable CLI와 함께 사용
  - AWS SDK 또는 DeleteKeyspace CLI와 함께 사용
  - AWS SDK 또는 DeleteTable CLI와 함께 사용
  - AWS SDK 또는 GetKeyspace CLI와 함께 사용
  - AWS SDK 또는 GetTable CLI와 함께 사용
  - AWS SDK 또는 ListKeyspaces CLI와 함께 사용
  - AWS SDK 또는 ListTables CLI와 함께 사용
  - AWS SDK 또는 RestoreTable CLI와 함께 사용
  - AWS SDK 또는 UpdateTable CLI와 함께 사용
- SDK를 사용하는 Amazon Keyspace의 시나리오 AWS
  - SDK를 사용하여 Amazon Keyspaces 키스페이스 및 테이블 시작하기 AWS

SDK를 사용한 Amazon 키스페이스 관련 작업 AWS

다음 코드 예제는 SDK를 사용하여 개별 Amazon Keyspaces 작업을 수행하는 방법을 보여줍니다. AWS 이들 발췌문은 Amazon Keyspaces API를 직접적으로 호출하며, 컨텍스트에서 실행되어야 하는 더 큰 프로그램에서 발췌한 코드입니다. 각 예제에는 코드 GitHub 설정 및 실행 지침을 찾을 수 있는 링크가 포함되어 있습니다.

다음 예제에는 가장 일반적으로 사용되는 작업만 포함되어 있습니다. 전체 목록은 Amazon Keyspaces(Apache Cassandra용) API 참조를 참고하세요.

예

- AWS SDK 또는 CreateKeyspace CLI와 함께 사용
- AWS SDK 또는 CreateTable CLI와 함께 사용
- AWS SDK 또는 DeleteKeyspace CLI와 함께 사용
- AWS SDK 또는 DeleteTable CLI와 함께 사용

- AWS SDK 또는 GetKeyspace CLI와 함께 사용
- AWS SDK 또는 GetTable CLI와 함께 사용
- AWS SDK 또는 ListKeyspaces CLI와 함께 사용
- AWS SDK 또는 ListTables CLI와 함께 사용
- AWS SDK 또는 RestoreTable CLI와 함께 사용
- AWS SDK 또는 UpdateTable CLI와 함께 사용

AWS SDK 또는 **CreateKeyspace** CLI와 함께 사용

다음 코드 예제는 CreateKeyspace의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Create a new keyspace.
/// </summary>
/// <param name="keyspaceName">The name for the new keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
public async Task<string> CreateKeyspace(string keyspaceName)
{
    var response =
        await _amazonKeyspaces.CreateKeyspaceAsync(
            new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}
```

- API 세부 정보는 AWS SDK for .NET API CreateKeyspace참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void createKeySpace(KeyspacesClient keyClient, String
keyspaceName) {
       try {
           CreateKeyspaceRequest keyspaceRequest =
CreateKeyspaceRequest.builder()
                   .keyspaceName(keyspaceName)
                   .build();

           CreateKeyspaceResponse response =
keyClient.createKeyspace(keyspaceRequest);
           System.out.println("The ARN of the KeySpace is " +
response.resourceArn());

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API 세부 정보는 AWS SDK for Java 2.x API CreateKeyspace참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createKeySpace(keyspaceNameVal: String) {
    val keyspaceRequest =
        CreateKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createKeyspace(keyspaceRequest)
        println("The ARN of the KeySpace is ${response.resourceArn}")
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 CreateKeyspace.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
```

```
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def create_keyspace(self, name):
        """
        Creates a keyspace.

        :param name: The name to give the keyspace.
        :return: The Amazon Resource Name (ARN) of the new keyspace.
        """
        try:
            response = self.keyspaces_client.create_keyspace(keyspaceName=name)
            self.ks_name = name
            self.ks_arn = response["resourceArn"]
        except ClientError as err:
            logger.error(
                "Couldn't create %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.ks_arn
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 CreateKeyspace.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **CreateTable** CLI와 함께 사용

다음 코드 예제는 CreateTable의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Create a new Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be
created.</param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
            PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
```

```
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }
```

- API 세부 정보는 AWS SDK for .NET API CreateTable참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void createTable(KeyspacesClient keyClient, String keySpace,
String tableName) {
        try {
            // Set the columns.
            ColumnDefinition defTitle = ColumnDefinition.builder()
                    .name("title")
                    .type("text")
                    .build();

            ColumnDefinition defYear = ColumnDefinition.builder()
                    .name("year")
                    .type("int")
                    .build();

            ColumnDefinition defReleaseDate = ColumnDefinition.builder()
                    .name("release_date")
                    .type("timestamp")
                    .build();

            ColumnDefinition defPlot = ColumnDefinition.builder()
                    .name("plot")
                    .type("text")
```

```
                .build();

        List<ColumnDefinition> colList = new ArrayList<>();
        colList.add(defTitle);
        colList.add(defYear);
        colList.add(defReleaseDate);
        colList.add(defPlot);

        // Set the keys.
        PartitionKey yearKey = PartitionKey.builder()
                .name("year")
                .build();

        PartitionKey titleKey = PartitionKey.builder()
                .name("title")
                .build();

        List<PartitionKey> keyList = new ArrayList<>();
        keyList.add(yearKey);
        keyList.add(titleKey);

        SchemaDefinition schemaDefinition = SchemaDefinition.builder()
                .partitionKeys(keyList)
                .allColumns(colList)
                .build();

        PointInTimeRecovery timeRecovery = PointInTimeRecovery.builder()
                .status(PointInTimeRecoveryStatus.ENABLED)
                .build();

        CreateTableRequest tableRequest = CreateTableRequest.builder()
                .keyspaceName(keySpace)
                .tableName(tableName)
                .schemaDefinition(schemaDefinition)
                .pointInTimeRecovery(timeRecovery)
                .build();

        CreateTableResponse response = keyClient.createTable(tableRequest);
        System.out.println("The table ARN is " + response.resourceArn());

    } catch (KeyspacesException e) {
        System.err.println(e.awsErrorDetails().errorMessage());
        System.exit(1);
    }
```

```
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API CreateTable참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun createTable(
    keySpaceVal: String?,
    tableNameVal: String?,
) {
    // Set the columns.
    val defTitle =
        ColumnDefinition {
            name = "title"
            type = "text"
        }

    val defYear =
        ColumnDefinition {
            name = "year"
            type = "int"
        }

    val defReleaseDate =
        ColumnDefinition {
            name = "release_date"
            type = "timestamp"
        }

    val defPlot =
        ColumnDefinition {
            name = "plot"
            type = "text"
```

```
        }

    val colList = ArrayList<ColumnDefinition>()
    colList.add(defTitle)
    colList.add(defYear)
    colList.add(defReleaseDate)
    colList.add(defPlot)

    // Set the keys.
    val yearKey =
        PartitionKey {
            name = "year"
        }

    val titleKey =
        PartitionKey {
            name = "title"
        }

    val keyList = ArrayList<PartitionKey>()
    keyList.add(yearKey)
    keyList.add(titleKey)

    val schemaDefinitionOb =
        SchemaDefinition {
            partitionKeys = keyList
            allColumns = colList
        }

    val timeRecovery =
        PointInTimeRecovery {
            status = PointInTimeRecoveryStatus.Enabled
        }

    val tableRequest =
        CreateTableRequest {
            keyspaceName = keySpaceVal
            tableName = tableNameVal
            schemaDefinition = schemaDefinitionOb
            pointInTimeRecovery = timeRecovery
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createTable(tableRequest)
```

```
        println("The table ARN is ${response.resourceArn}")
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 CreateTable.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def create_table(self, table_name):
        """
        Creates a table in the  keyspace.
        The table is created with a schema for storing movie data
        and has point-in-time recovery enabled.
```

```
        :param table_name: The name to give the table.
        :return: The ARN of the new table.
        """
        try:
            response = self.keyspaces_client.create_table(
                keyspaceName=self.ks_name,
                tableName=table_name,
                schemaDefinition={
                    "allColumns": [
                        {"name": "title", "type": "text"},
                        {"name": "year", "type": "int"},
                        {"name": "release_date", "type": "timestamp"},
                        {"name": "plot", "type": "text"},
                    ],
                    "partitionKeys": [{"name": "year"}, {"name": "title"}],
                },
                pointInTimeRecovery={"status": "ENABLED"},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["resourceArn"]
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 CreateTable.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **DeleteKeyspace** CLI와 함께 사용

다음 코드 예제는 DeleteKeyspace의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Delete an existing keyspace.
/// </summary>
/// <param name="keyspaceName"></param>
/// <returns>A Boolean value indicating the success of the action.</returns>
public async Task<bool> DeleteKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
        new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.HttpStatusCode == HttpStatusCode.OK;
}
```

- API 세부 정보는 AWS SDK for .NET API DeleteKeyspace참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void deleteKeyspace(KeyspacesClient keyClient, String
 keyspaceName) {
        try {
            DeleteKeyspaceRequest deleteKeyspaceRequest =
 DeleteKeyspaceRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();

            keyClient.deleteKeyspace(deleteKeyspaceRequest);

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API DeleteKeyspace참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteKeyspace(keyspaceNameVal: String?) {
    val deleteKeyspaceRequest =
        DeleteKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteKeyspace(deleteKeyspaceRequest)
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 DeleteKeyspace.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def delete_keyspace(self):
        """
        Deletes the keyspace.
        """
        try:
            self.keyspaces_client.delete_keyspace(keyspaceName=self.ks_name)
            self.ks_name = None
        except ClientError as err:
            logger.error(
                "Couldn't delete keyspace %s. Here's why: %s: %s",
                self.ks_name,
```

```
            err.response["Error"]["Code"],
            err.response["Error"]["Message"],
        )
        raise
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 DeleteKeyspace.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **DeleteTable** CLI와 함께 사용

다음 코드 예제는 DeleteTable의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Delete an Amazon Keyspaces table.
/// </summary>
/// <param name="keyspaceName">The keyspace containing the table.</param>
/// <param name="tableName">The name of the table to delete.</param>
/// <returns>A Boolean value indicating the success of the action.</returns>
```

```
    public async Task<bool> DeleteTable(string keyspaceName, string tableName)
    {
        var response = await _amazonKeyspaces.DeleteTableAsync(
            new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }
```

- API 세부 정보는 AWS SDK for .NET API DeleteTable참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    public static void deleteTable(KeyspacesClient keyClient, String
keyspaceName, String tableName) {
        try {
            DeleteTableRequest tableRequest = DeleteTableRequest.builder()
                    .keyspaceName(keyspaceName)
                    .tableName(tableName)
                    .build();

            keyClient.deleteTable(tableRequest);

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API DeleteTable참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun deleteTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    val tableRequest =
        DeleteTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteTable(tableRequest)
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 DeleteTable.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
```

```
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def delete_table(self):
        """
        Deletes the table from the keyspace.
        """
        try:
            self.keyspaces_client.delete_table(
                keyspaceName=self.ks_name, tableName=self.table_name
            )
            self.table_name = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",
                self.table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 DeleteTable.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetKeyspace** CLI와 함께 사용

다음 코드 예제는 GetKeyspace의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/// <summary>
/// Get data about a keyspace.
/// </summary>
/// <param name="keyspaceName">The name of the keyspace.</param>
/// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
public async Task<string> GetKeyspace(string keyspaceName)
{
    var response = await _amazonKeyspaces.GetKeyspaceAsync(
        new GetKeyspaceRequest { KeyspaceName = keyspaceName });
    return response.ResourceArn;
}
```

- API 세부 정보는 AWS SDK for .NET API GetKeyspace참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void checkKeyspaceExistence(KeyspacesClient keyClient, String
keyspaceName) {
       try {
           GetKeyspaceRequest keyspaceRequest = GetKeyspaceRequest.builder()
                   .keyspaceName(keyspaceName)
                   .build();

           GetKeyspaceResponse response =
keyClient.getKeyspace(keyspaceRequest);
           String name = response.keyspaceName();
           System.out.println("The " + name + " KeySpace is ready");

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API 세부 정보는 AWS SDK for Java 2.x API GetKeyspace참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun checkKeyspaceExistence(keyspaceNameVal: String?) {
    val keyspaceRequest =
        GetKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }
    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        val response: GetKeyspaceResponse =
 keyClient.getKeyspace(keyspaceRequest)
        val name = response.keyspaceName
        println("The $name KeySpace is ready")
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 GetKeyspace.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
```

```
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def exists_keyspace(self, name):
        """
        Checks whether a keyspace exists.

        :param name: The name of the keyspace to look up.
        :return: True when the keyspace exists. Otherwise, False.
        """
        try:
            response = self.keyspaces_client.get_keyspace(keyspaceName=name)
            self.ks_name = response["keyspaceName"]
            self.ks_arn = response["resourceArn"]
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Keyspace %s does not exist.", name)
                exists = False
            else:
                logger.error(
                    "Couldn't verify %s exists. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return exists
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 GetKeyspace.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **GetTable** CLI와 함께 사용

다음 코드 예제는 GetTable의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
    }
```

- API 세부 정보는 AWS SDK for .NET API GetTable참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void checkTable(KeyspacesClient keyClient, String keyspaceName,
String tableName)
           throws InterruptedException {
       try {
           boolean tableStatus = false;
           String status;
           GetTableResponse response = null;
           GetTableRequest tableRequest = GetTableRequest.builder()
                   .keyspaceName(keyspaceName)
                   .tableName(tableName)
                   .build();

           while (!tableStatus) {
               response = keyClient.getTable(tableRequest);
               status = response.statusAsString();
               System.out.println(". The table status is " + status);

               if (status.compareTo("ACTIVE") == 0) {
                   tableStatus = true;
               }
               Thread.sleep(500);
           }

           List<ColumnDefinition> cols =
response.schemaDefinition().allColumns();
           for (ColumnDefinition def : cols) {
               System.out.println("The column name is " + def.name());
               System.out.println("The column type is " + def.type());
           }

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
```

```
            System.exit(1);
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API GetTable참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun checkTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    var tableStatus = false
    var status: String
    var response: GetTableResponse? = null

    val tableRequest =
        GetTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }
    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        while (!tableStatus) {
            response = keyClient.getTable(tableRequest)
            status = response!!.status.toString()
            println(". The table status is $status")
            if (status.compareTo("ACTIVE") == 0) {
                tableStatus = true
            }
            delay(500)
        }
        val cols: List<ColumnDefinition>? =
 response!!.schemaDefinition?.allColumns
```

```
        if (cols != null) {
            for (def in cols) {
                println("The column name is ${def.name}")
                println("The column type is ${def.type}")
            }
        }
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 GetTable.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def get_table(self, table_name):
```

```
        """
        Gets data about a table in the keyspace.

        :param table_name: The name of the table to look up.
        :return: Data about the table.
        """
        try:
            response = self.keyspaces_client.get_table(
                keyspaceName=self.ks_name, tableName=table_name
            )
            self.table_name = table_name
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Table %s does not exist.", table_name)
                self.table_name = None
                response = None
            else:
                logger.error(
                    "Couldn't verify %s exists. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return response
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 GetTable.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **ListKeyspaces** CLI와 함께 사용

다음 코드 예제는 ListKeyspaces의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Lists all keyspaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeyspaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keyspace in paginator.Keyspaces)
        {

Console.WriteLine($"{keyspace.KeyspaceName,-30}\t{keyspace.ResourceArn}");
        }
    }
```

- API 세부 정보는 AWS SDK for .NET API ListKeyspaces참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void listKeyspacesPaginator(KeyspacesClient keyClient) {
       try {
           ListKeyspacesRequest keyspacesRequest =
ListKeyspacesRequest.builder()
                   .maxResults(10)
                   .build();

           ListKeyspacesIterable listRes =
keyClient.listKeyspacesPaginator(keyspacesRequest);
           listRes.stream()
                   .flatMap(r -> r.keyspaces().stream())
                   .forEach(content -> System.out.println(" Name: " +
content.keyspaceName()));

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API 세부 정보는 AWS SDK for Java 2.x API ListKeyspaces참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun listKeyspacesPaginator() {
    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient
            .listKeyspacesPaginated(ListKeyspacesRequest {})
            .transform { it.keyspaces?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name: ${obj.keyspaceName}")
            }
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 ListKeyspaces.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
```

```
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def list_keyspaces(self, limit):
        """
        Lists the keyspaces in your account.

        :param limit: The maximum number of keyspaces to list.
        """
        try:
            ks_paginator = self.keyspaces_client.get_paginator("list_keyspaces")
            for page in ks_paginator.paginate(PaginationConfig={"MaxItems":
limit}):
                for ks in page["keyspaces"]:
                    print(ks["keyspaceName"])
                    print(f"\t{ks['resourceArn']}")
        except ClientError as err:
            logger.error(
                "Couldn't list keyspaces. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 ListKeyspaces.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **ListTables** CLI와 함께 사용

다음 코드 예제는 ListTables의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new
ListTablesRequest { KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
        });

        return response.Tables;
    }
```

- API 세부 정보는 AWS SDK for .NET API ListTables참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void listTables(KeyspacesClient keyClient, String keyspaceName)
{
       try {
           ListTablesRequest tablesRequest = ListTablesRequest.builder()
                   .keyspaceName(keyspaceName)
                   .build();

           ListTablesIterable listRes =
keyClient.listTablesPaginator(tablesRequest);
           listRes.stream()
                   .flatMap(r -> r.tables().stream())
                   .forEach(content -> System.out.println(" ARN: " +
content.resourceArn() +
                           " Table name: " + content.tableName()));

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API 세부 정보는 AWS SDK for Java 2.x API ListTables참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun listTables(keyspaceNameVal: String?) {
    val tablesRequest =
        ListTablesRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient
            .listTablesPaginated(tablesRequest)
            .transform { it.tables?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println(" ARN: ${obj.resourceArn} Table name: ${obj.tableName}")
            }
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 ListTables.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
```

```
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def list_tables(self):
        """
        Lists the tables in the keyspace.
        """
        try:
            table_paginator = self.keyspaces_client.get_paginator("list_tables")
            for page in table_paginator.paginate(keyspaceName=self.ks_name):
                for table in page["tables"]:
                    print(table["tableName"])
                    print(f"\t{table['resourceArn']}")
        except ClientError as err:
            logger.error(
                "Couldn't list tables in keyspace %s. Here's why: %s: %s",
                self.ks_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 ListTables.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **RestoreTable** CLI와 함께 사용

다음 코드 예제는 RestoreTable의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Restores the specified table to the specified point in time.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
 string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
```

```
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }
```

- API 세부 정보는 AWS SDK for .NET API RestoreTable참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void restoreTable(KeyspacesClient keyClient, String
keyspaceName, ZonedDateTime utc) {
        try {
            Instant myTime = utc.toInstant();
            RestoreTableRequest restoreTableRequest =
RestoreTableRequest.builder()
                    .restoreTimestamp(myTime)
                    .sourceTableName("Movie")
                    .targetKeyspaceName(keyspaceName)
                    .targetTableName("MovieRestore")
                    .sourceKeyspaceName(keyspaceName)
                    .build();

            RestoreTableResponse response =
keyClient.restoreTable(restoreTableRequest);
            System.out.println("The ARN of the restored table is " +
response.restoredTableARN());

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
```

```
        }
    }
```

- API 세부 정보는 AWS SDK for Java 2.x API RestoreTable참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun restoreTable(
    keyspaceName: String?,
    utc: ZonedDateTime,
) {
    // Create an aws.smithy.kotlin.runtime.time.Instant value.
    val timeStamp =
        aws.smithy.kotlin.runtime.time
            .Instant(utc.toInstant())
    val restoreTableRequest =
        RestoreTableRequest {
            restoreTimestamp = timeStamp
            sourceTableName = "MovieKotlin"
            targetKeyspaceName = keyspaceName
            targetTableName = "MovieRestore"
            sourceKeyspaceName = keyspaceName
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.restoreTable(restoreTableRequest)
        println("The ARN of the restored table is ${response.restoredTableArn}")
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 RestoreTable.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def restore_table(self, restore_timestamp):
        """
        Restores the table to a previous point in time. The table is restored
        to a new table in the same keyspace.

        :param restore_timestamp: The point in time to restore the table. This
 time
                                  must be in UTC format.
        :return: The name of the restored table.
        """
        try:
            restored_table_name = f"{self.table_name}_restored"
            self.keyspaces_client.restore_table(
```

```
                sourceKeyspaceName=self.ks_name,
                sourceTableName=self.table_name,
                targetKeyspaceName=self.ks_name,
                targetTableName=restored_table_name,
                restoreTimestamp=restore_timestamp,
            )
        except ClientError as err:
            logger.error(
                "Couldn't restore table %s. Here's why: %s: %s",
                restore_timestamp,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return restored_table_name
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 RestoreTable.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

AWS SDK 또는 **UpdateTable** CLI와 함께 사용

다음 코드 예제는 UpdateTable의 사용 방법을 보여줍니다.

작업 예제는 대규모 프로그램에서 발췌한 코드이며 컨텍스트에 맞춰 실행해야 합니다. 다음 코드 예제에서는 컨텍스트 내에서 이 작업을 확인할 수 있습니다.

- 키스페이스 및 테이블 시작하기

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
    /// <summary>
    /// Updates the movie table to add a boolean column named watched.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to change.</param>
    /// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
    public async Task<string> UpdateTable(string keyspaceName, string tableName)
    {
        var newColumn = new ColumnDefinition { Name = "watched", Type =
"boolean" };
        var request = new UpdateTableRequest
        {
            KeyspaceName = keyspaceName,
            TableName = tableName,
            AddColumns = new List<ColumnDefinition> { newColumn }
        };
        var response = await _amazonKeyspaces.UpdateTableAsync(request);
        return response.ResourceArn;
    }
```

- API 세부 정보는 AWS SDK for .NET API UpdateTable참조를 참조하십시오.

Java

SDK for Java 2.x

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
   public static void updateTable(KeyspacesClient keyClient, String keySpace,
String tableName) {
       try {
           ColumnDefinition def = ColumnDefinition.builder()
                   .name("watched")
                   .type("boolean")
                   .build();

           UpdateTableRequest tableRequest = UpdateTableRequest.builder()
                   .keyspaceName(keySpace)
                   .tableName(tableName)
                   .addColumns(def)
                   .build();

           keyClient.updateTable(tableRequest);

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- API 세부 정보는 AWS SDK for Java 2.x API UpdateTable참조를 참조하십시오.

Kotlin

SDK for Kotlin

> **Note**
>
> 자세한 내용은 다음과 같습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
suspend fun updateTable(
    keySpace: String?,
    tableNameVal: String?,
) {
    val def =
        ColumnDefinition {
            name = "watched"
            type = "boolean"
        }

    val tableRequest =
        UpdateTableRequest {
            keyspaceName = keySpace
            tableName = tableNameVal
            addColumns = listOf(def)
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient.updateTable(tableRequest)
    }
}
```

- API 세부 정보는 Kotlin API용AWS SDK 레퍼런스를 참조하세요 UpdateTable.

Python

SDK for Python(Boto3)

> **Note**
>
> 자세한 내용은 여기에서 확인할 수 있습니다. GitHub AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""

    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def update_table(self):
        """
        Updates the schema of the table.

        This example updates a table of movie data by adding a new column
        that tracks whether the movie has been watched.
        """
        try:
            self.keyspaces_client.update_table(
                keyspaceName=self.ks_name,
                tableName=self.table_name,
                addColumns=[{"name": "watched", "type": "boolean"}],
            )
```

```
        except ClientError as err:
            logger.error(
                "Couldn't update table %s. Here's why: %s: %s",
                self.table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

- API에 대한 자세한 내용은 파이썬용AWS SDK (Boto3) API 레퍼런스를 참조하십시오 UpdateTable.

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오. 아마존 키스페이스를 SDK와 함께 사용하기 AWS 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

SDK를 사용하는 Amazon Keyspace의 시나리오 AWS

다음 코드 예제는 SDK를 사용하여 Amazon Keyspaces에서 일반적인 시나리오를 구현하는 방법을 보여줍니다. AWS 이러한 시나리오에서는 Amazon Keyspaces 내에서 여러 함수를 호출하여 특정 태스크를 수행하는 방법을 보여줍니다. 각 시나리오에는 코드 설정 및 실행 방법에 대한 지침을 찾을 수 있는 링크가 포함되어 있습니다. GitHub

예

- SDK를 사용하여 Amazon Keyspaces 키스페이스 및 테이블 시작하기 AWS

SDK를 사용하여 Amazon Keyspaces 키스페이스 및 테이블 시작하기 AWS

다음 코드 예제는 다음과 같은 작업을 수행하는 방법을 보여줍니다.

- 키스페이스와 테이블을 생성합니다. 테이블 스키마에는 동영상 데이터가 저장되며 point-in-time 복구가 활성화되어 있습니다.
- SiGV4 인증을 통한 보안 TLS 연결을 사용하여 키스페이스에 연결합니다.
- 테이블을 쿼리합니다. 영화 데이터를 추가, 검색 및 업데이트합니다.
- 테이블을 업데이트 하세요. 열을 추가하여 시청한 영화를 추적합니다.
- 테이블을 이전 상태로 복원하고 리소스를 정리합니다.

.NET

AWS SDK for .NET

> **Note**
>
> 더 많은 정보가 있습니다 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
global using System.Security.Cryptography.X509Certificates;
global using Amazon.Keyspaces;
global using Amazon.Keyspaces.Model;
global using KeyspacesActions;
global using KeyspacesScenario;
global using Microsoft.Extensions.Configuration;
global using Microsoft.Extensions.DependencyInjection;
global using Microsoft.Extensions.Hosting;
global using Microsoft.Extensions.Logging;
global using Microsoft.Extensions.Logging.Console;
global using Microsoft.Extensions.Logging.Debug;
global using Newtonsoft.Json;


namespace KeyspacesBasics;

/// <summary>
/// Amazon Keyspaces (for Apache Cassandra) scenario. Shows some of the basic
/// actions performed with Amazon Keyspaces.
/// </summary>
public class KeyspacesBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for the Amazon service.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
```

```
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonKeyspaces>()
            .AddTransient<KeyspacesWrapper>()
            .AddTransient<CassandraWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<KeyspacesBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        var keyspacesWrapper =
host.Services.GetRequiredService<KeyspacesWrapper>();
        var uiMethods = new UiMethods();

        var keyspaceName = configuration["KeyspaceName"];
        var tableName = configuration["TableName"];

        bool success; // Used to track the results of some operations.

        uiMethods.DisplayOverview();
        uiMethods.PressEnter();

        // Create the keyspace.
        var keyspaceArn = await keyspacesWrapper.CreateKeyspace(keyspaceName);

        // Wait for the keyspace to be available. GetKeyspace results in a
        // resource not found error until it is ready for use.
        try
        {
            var getKeyspaceArn = "";
            Console.Write($"Created {keyspaceName}. Waiting for it to become
available. ");
            do
            {
```

```
                getKeyspaceArn = await
keyspacesWrapper.GetKeyspace(keyspaceName);
                Console.Write(". ");
            } while (getKeyspaceArn != keyspaceArn);
        }
        catch (ResourceNotFoundException)
        {
            Console.WriteLine("Waiting for keyspace to be created.");
        }

        Console.WriteLine($"\nThe keyspace {keyspaceName} is ready for use.");

        uiMethods.PressEnter();

        // Create the table.
        // First define the schema.
        var allColumns = new List<ColumnDefinition>
        {
            new ColumnDefinition { Name = "title", Type = "text" },
            new ColumnDefinition { Name = "year", Type = "int" },
            new ColumnDefinition { Name = "release_date", Type = "timestamp" },
            new ColumnDefinition { Name = "plot", Type = "text" },
        };

        var partitionKeys = new List<PartitionKey>
        {
            new PartitionKey { Name = "year", },
            new PartitionKey { Name = "title" },
        };

        var tableSchema = new SchemaDefinition
        {
            AllColumns = allColumns,
            PartitionKeys = partitionKeys,
        };

        var tableArn = await keyspacesWrapper.CreateTable(keyspaceName,
tableSchema, tableName);

        // Wait for the table to be active.
        try
        {
            var resp = new GetTableResponse();
            Console.Write("Waiting for the new table to be active. ");
```

```
            do
            {
                try
                {
                    resp = await keyspacesWrapper.GetTable(keyspaceName,
tableName);
                    Console.Write(".");
                }
                catch (ResourceNotFoundException)
                {
                    Console.Write(".");
                }
            } while (resp.Status != TableStatus.ACTIVE);

            // Display the table's schema.
            Console.WriteLine($"\nTable {tableName} has been created in
{keyspaceName}");
            Console.WriteLine("Let's take a look at the schema.");
            uiMethods.DisplayTitle("All columns");
            resp.SchemaDefinition.AllColumns.ForEach(column =>
            {
                Console.WriteLine($"{column.Name,-40}\t{column.Type,-20}");
            });

            uiMethods.DisplayTitle("Cluster keys");
            resp.SchemaDefinition.ClusteringKeys.ForEach(clusterKey =>
            {

Console.WriteLine($"{clusterKey.Name,-40}\t{clusterKey.OrderBy,-20}");
            });

            uiMethods.DisplayTitle("Partition keys");
            resp.SchemaDefinition.PartitionKeys.ForEach(partitionKey =>
            {
                Console.WriteLine($"{partitionKey.Name}");
            });

            uiMethods.PressEnter();
        }
        catch (ResourceNotFoundException ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
```

```
        // Access Apache Cassandra using the Cassandra drive for C#.
        var cassandraWrapper =
host.Services.GetRequiredService<CassandraWrapper>();
        var movieFilePath = configuration["MovieFile"];

        Console.WriteLine("Let's add some movies to the table we created.");
        var inserted = await cassandraWrapper.InsertIntoMovieTable(keyspaceName,
tableName, movieFilePath);

        uiMethods.PressEnter();

        Console.WriteLine("Added the following movies to the table:");
        var rows = await cassandraWrapper.GetMovies(keyspaceName, tableName);
        uiMethods.DisplayTitle("All Movies");

        foreach (var row in rows)
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            var plot = row.GetValue<string>("plot");
            var release_date = row.GetValue<DateTime>("release_date");
            Console.WriteLine($"{release_date}\t{title}\t{year}\n{plot}");
            Console.WriteLine(uiMethods.SepBar);
        }

        // Update the table schema
        uiMethods.DisplayTitle("Update table schema");
        Console.WriteLine("Now we will update the table to add a boolean field
called watched.");

        // First save the current time as a UTC Date so the original
        // table can be restored later.
        var timeChanged = DateTime.UtcNow;

        // Now update the schema.
        var resourceArn = await keyspacesWrapper.UpdateTable(keyspaceName,
tableName);
        uiMethods.PressEnter();

        Console.WriteLine("Now let's mark some of the movies as watched.");

        // Pick some files to mark as watched.
        var movieToWatch = rows[2].GetValue<string>("title");
        var watchedMovieYear = rows[2].GetValue<int>("year");
```

```
        var changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[6].GetValue<string>("title");
        watchedMovieYear = rows[6].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[9].GetValue<string>("title");
        watchedMovieYear = rows[9].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[10].GetValue<string>("title");
        watchedMovieYear = rows[10].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        movieToWatch = rows[13].GetValue<string>("title");
        watchedMovieYear = rows[13].GetValue<int>("year");
        changedRows = await cassandraWrapper.MarkMovieAsWatched(keyspaceName,
tableName, movieToWatch, watchedMovieYear);

        uiMethods.DisplayTitle("Watched movies");
        Console.WriteLine("These movies have been marked as watched:");
        rows = await cassandraWrapper.GetWatchedMovies(keyspaceName, tableName);
        foreach (var row in rows)
        {
            var title = row.GetValue<string>("title");
            var year = row.GetValue<int>("year");
            Console.WriteLine($"{title,-40}\t{year,8}");
        }
        uiMethods.PressEnter();

        Console.WriteLine("We can restore the table to its previous state but
that can take up to 20 minutes to complete.");
        string answer;
        do
        {
            Console.WriteLine("Do you want to restore the table? (y/n)");
            answer = Console.ReadLine();
        } while (answer.ToLower() != "y" && answer.ToLower() != "n");

        if (answer == "y")
```

```
        {
            var restoredTableName = $"{tableName}_restored";
            var restoredTableArn = await keyspacesWrapper.RestoreTable(
                keyspaceName,
                tableName,
                restoredTableName,
                timeChanged);
            // Loop and call GetTable until the table is gone. Once it has been
            // deleted completely, GetTable will raise a
ResourceNotFoundException.
            bool wasRestored = false;

            try
            {
                do
                {
                    var resp = await keyspacesWrapper.GetTable(keyspaceName,
restoredTableName);
                    wasRestored = (resp.Status == TableStatus.ACTIVE);
                } while (!wasRestored);
            }
            catch (ResourceNotFoundException)
            {
                // If the restored table raised an error, it isn't
                // ready yet.
                Console.Write(".");
            }
        }

        uiMethods.DisplayTitle("Clean up resources.");

        // Delete the table.
        success = await keyspacesWrapper.DeleteTable(keyspaceName, tableName);

        Console.WriteLine($"Table {tableName} successfully deleted from
{keyspaceName}.");
        Console.WriteLine("Waiting for the table to be removed completely. ");

        // Loop and call GetTable until the table is gone. Once it has been
        // deleted completely, GetTable will raise a ResourceNotFoundException.
        bool wasDeleted = false;

        try
        {
```

```
            do
            {
                var resp = await keyspacesWrapper.GetTable(keyspaceName,
 tableName);
            } while (!wasDeleted);
        }
        catch (ResourceNotFoundException ex)
        {
            wasDeleted = true;
            Console.WriteLine($"{ex.Message} indicates that the table has been
 deleted.");
        }

        // Delete the keyspace.
        success = await keyspacesWrapper.DeleteKeyspace(keyspaceName);
        Console.WriteLine("The keyspace has been deleted and the demo is now
 complete.");
    }
}
```

```
namespace KeyspacesActions;

/// <summary>
/// Performs Amazon Keyspaces (for Apache Cassandra) actions.
/// </summary>
public class KeyspacesWrapper
{
    private readonly IAmazonKeyspaces _amazonKeyspaces;

    /// <summary>
    /// Constructor for the KeyspaceWrapper.
    /// </summary>
    /// <param name="amazonKeyspaces">An Amazon Keyspaces client object.</param>
    public KeyspacesWrapper(IAmazonKeyspaces amazonKeyspaces)
    {
        _amazonKeyspaces = amazonKeyspaces;
    }

    /// <summary>
    /// Create a new keyspace.
    /// </summary>
```

```
    /// <param name="keyspaceName">The name for the new keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new keyspace.</returns>
    public async Task<string> CreateKeyspace(string keyspaceName)
    {
        var response =
            await _amazonKeyspaces.CreateKeyspaceAsync(
                new CreateKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }


    /// <summary>
    /// Create a new Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace where the table will be
created.</param>
    /// <param name="schema">The schema for the new table.</param>
    /// <param name="tableName">The name of the new table.</param>
    /// <returns>The Amazon Resource Name (ARN) of the new table.</returns>
    public async Task<string> CreateTable(string keyspaceName, SchemaDefinition
schema, string tableName)
    {
        var request = new CreateTableRequest
        {
            KeyspaceName = keyspaceName,
            SchemaDefinition = schema,
            TableName = tableName,
            PointInTimeRecovery = new PointInTimeRecovery { Status =
PointInTimeRecoveryStatus.ENABLED }
        };

        var response = await _amazonKeyspaces.CreateTableAsync(request);
        return response.ResourceArn;
    }


    /// <summary>
    /// Delete an existing keyspace.
    /// </summary>
    /// <param name="keyspaceName"></param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.DeleteKeyspaceAsync(
```

```
            new DeleteKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Delete an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to delete.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> DeleteTable(string keyspaceName, string tableName)
    {
        var response = await _amazonKeyspaces.DeleteTableAsync(
            new DeleteTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response.HttpStatusCode == HttpStatusCode.OK;
    }


    /// <summary>
    /// Get data about a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>The Amazon Resource Name (ARN) of the keyspace.</returns>
    public async Task<string> GetKeyspace(string keyspaceName)
    {
        var response = await _amazonKeyspaces.GetKeyspaceAsync(
            new GetKeyspaceRequest { KeyspaceName = keyspaceName });
        return response.ResourceArn;
    }


    /// <summary>
    /// Get information about an Amazon Keyspaces table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the Amazon Keyspaces table.</param>
    /// <returns>The response containing data about the table.</returns>
    public async Task<GetTableResponse> GetTable(string keyspaceName, string
tableName)
    {
        var response = await _amazonKeyspaces.GetTableAsync(
```

```
            new GetTableRequest { KeyspaceName = keyspaceName, TableName =
tableName });
        return response;
    }


    /// <summary>
    /// Lists all keyspaces for the account.
    /// </summary>
    /// <returns>Async task.</returns>
    public async Task ListKeyspaces()
    {
        var paginator = _amazonKeyspaces.Paginators.ListKeyspaces(new
ListKeyspacesRequest());

        Console.WriteLine("{0, -30}\t{1}", "Keyspace name", "Keyspace ARN");
        Console.WriteLine(new string('-', Console.WindowWidth));
        await foreach (var keyspace in paginator.Keyspaces)
        {

Console.WriteLine($"{keyspace.KeyspaceName,-30}\t{keyspace.ResourceArn}");
        }
    }


    /// <summary>
    /// Lists the Amazon Keyspaces tables in a keyspace.
    /// </summary>
    /// <param name="keyspaceName">The name of the keyspace.</param>
    /// <returns>A list of TableSummary objects.</returns>
    public async Task<List<TableSummary>> ListTables(string keyspaceName)
    {
        var response = await _amazonKeyspaces.ListTablesAsync(new
ListTablesRequest { KeyspaceName = keyspaceName });
        response.Tables.ForEach(table =>
        {

Console.WriteLine($"{table.KeyspaceName}\t{table.TableName}\t{table.ResourceArn}");
        });

        return response.Tables;
    }
```

```
    /// <summary>
    /// Restores the specified table to the specified point in time.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to restore.</param>
    /// <param name="timestamp">The time to which the table will be restored.</
param>
    /// <returns>The Amazon Resource Name (ARN) of the restored table.</returns>
    public async Task<string> RestoreTable(string keyspaceName, string tableName,
 string restoredTableName, DateTime timestamp)
    {
        var request = new RestoreTableRequest
        {
            RestoreTimestamp = timestamp,
            SourceKeyspaceName = keyspaceName,
            SourceTableName = tableName,
            TargetKeyspaceName = keyspaceName,
            TargetTableName = restoredTableName
        };

        var response = await _amazonKeyspaces.RestoreTableAsync(request);
        return response.RestoredTableARN;
    }


    /// <summary>
    /// Updates the movie table to add a boolean column named watched.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table to change.</param>
    /// <returns>The Amazon Resource Name (ARN) of the updated table.</returns>
    public async Task<string> UpdateTable(string keyspaceName, string tableName)
    {
        var newColumn = new ColumnDefinition { Name = "watched", Type =
 "boolean" };
        var request = new UpdateTableRequest
        {
            KeyspaceName = keyspaceName,
            TableName = tableName,
            AddColumns = new List<ColumnDefinition> { newColumn }
        };
        var response = await _amazonKeyspaces.UpdateTableAsync(request);
        return response.ResourceArn;
    }
```

```
}
```

```
using System.Net;
using Cassandra;

namespace KeyspacesScenario;

/// <summary>
/// Class to perform CRUD methods on an Amazon Keyspaces (for Apache Cassandra)
 database.
///
/// NOTE: This sample uses a plain text authenticator for example purposes only.
/// Recommended best practice is to use a SigV4 authentication plugin, if
 available.
/// </summary>
public class CassandraWrapper
{
    private readonly IConfiguration _configuration;
    private readonly string _localPathToFile;
    private const string _certLocation = "https://certs.secureserver.net/
repository/sf-class2-root.crt";
    private const string _certFileName = "sf-class2-root.crt";
    private readonly X509Certificate2Collection _certCollection;
    private X509Certificate2 _amazoncert;
    private Cluster _cluster;

    // User name and password for the service.
    private string _userName = null!;
    private string _pwd = null!;

    public CassandraWrapper()
    {
        _configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load test settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();

        _localPathToFile = Path.GetTempPath();
```

```
        // Get the Starfield digital certificate and save it locally.
        var client = new WebClient();
        client.DownloadFile(_certLocation, $"{_localPathToFile}/
{_certFileName}");

        //var httpClient = new HttpClient();
        //var httpResult = httpClient.Get(fileUrl);
        //using var resultStream = await httpResult.Content.ReadAsStreamAsync();
        //using var fileStream = File.Create(pathToSave);
        //resultStream.CopyTo(fileStream);

        _certCollection = new X509Certificate2Collection();
        _amazoncert = new X509Certificate2($"{_localPathToFile}/
{_certFileName}");

        // Get the user name and password stored in the configuration file.
        _userName = _configuration["UserName"]!;
        _pwd = _configuration["Password"]!;

        // For a list of Service Endpoints for Amazon Keyspaces, see:
        // https://docs.aws.amazon.com/keyspaces/latest/devguide/
programmatic.endpoints.html
        var awsEndpoint = _configuration["ServiceEndpoint"];

        _cluster = Cluster.Builder()
            .AddContactPoints(awsEndpoint)
            .WithPort(9142)
            .WithAuthProvider(new PlainTextAuthProvider(_userName, _pwd))
            .WithSSL(new SSLOptions().SetCertificateCollection(_certCollection))
            .WithQueryOptions(
                new QueryOptions()
                    .SetConsistencyLevel(ConsistencyLevel.LocalQuorum)
                    .SetSerialConsistencyLevel(ConsistencyLevel.LocalSerial))
            .Build();
    }

    /// <summary>
    /// Loads the contents of a JSON file into a list of movies to be
    /// added to the Apache Cassandra table.
    /// </summary>
    /// <param name="movieFileName">The full path to the JSON file.</param>
    /// <returns>A list of movie objects.</returns>
```

```
    public List<Movie> ImportMoviesFromJson(string movieFileName, int numToImport
= 0)
    {
        if (!File.Exists(movieFileName))
        {
            return null!;
        }

        using var sr = new StreamReader(movieFileName);
        string json = sr.ReadToEnd();

        var allMovies = JsonConvert.DeserializeObject<List<Movie>>(json);

        // If numToImport = 0, return all movies in the collection.
        if (numToImport == 0)
        {
            // Now return the entire list of movies.
            return allMovies;
        }
        else
        {
            // Now return the first numToImport entries.
            return allMovies.GetRange(0, numToImport);
        }
    }

    /// <summary>
    /// Insert movies into the movie table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="movieTableName">The Amazon Keyspaces table.</param>
    /// <param name="movieFilePath">The path to the resource file containing
    /// movie data to insert into the table.</param>
    /// <returns>A Boolean value indicating the success of the action.</returns>
    public async Task<bool> InsertIntoMovieTable(string keyspaceName, string
movieTableName, string movieFilePath, int numToImport = 20)
    {
        // Get some movie data from the movies.json file
        var movies = ImportMoviesFromJson(movieFilePath, numToImport);

        var session = _cluster.Connect(keyspaceName);

        string insertCql;
```

```
        RowSet rs;

        // Now we insert the numToImport movies into the table.
        foreach (var movie in movies)
        {
            // Escape single quote characters in the plot.
            insertCql = $"INSERT INTO {keyspaceName}.{movieTableName}
(title, year, release_date, plot) values($${movie.Title}$$, {movie.Year},
'{movie.Info.Release_Date.ToString("yyyy-MM-dd")}', $${movie.Info.Plot}$$)";
            rs = await session.ExecuteAsync(new SimpleStatement(insertCql));
        }

        return true;
    }

    /// <summary>
    /// Gets all of the movies in the movies table.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing movie data.</returns>
    public async Task<List<Row>> GetMovies(string keyspaceName, string tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT * FROM
{keyspaceName}.{tableName}"));

            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return null!;
        }
    }

    /// <summary>
    /// Mark a movie in the movie table as watched.
    /// </summary>
```

```
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <param name="title">The title of the movie to mark as watched.</param>
    /// <param name="year">The year the movie was released.</param>
    /// <returns>A set of rows containing the changed data.</returns>
    public async Task<List<Row>> MarkMovieAsWatched(string keyspaceName, string
tableName, string title, int year)
    {
        var session = _cluster.Connect();
        string updateCql = $"UPDATE {keyspaceName}.{tableName} SET watched=true
WHERE title = $${title}$$ AND year = {year};";
        var rs = await session.ExecuteAsync(new SimpleStatement(updateCql));
        var rows = rs.GetRows().ToList();
        return rows;
    }

    /// <summary>
    /// Retrieve the movies in the movies table where watched is true.
    /// </summary>
    /// <param name="keyspaceName">The keyspace containing the table.</param>
    /// <param name="tableName">The name of the table.</param>
    /// <returns>A list of row objects containing information about movies
    /// where watched is true.</returns>
    public async Task<List<Row>> GetWatchedMovies(string keyspaceName, string
tableName)
    {
        var session = _cluster.Connect();
        RowSet rs;
        try
        {
            rs = await session.ExecuteAsync(new SimpleStatement($"SELECT
title, year, plot FROM {keyspaceName}.{tableName} WHERE watched = true ALLOW
FILTERING"));

            // Extract the row data from the returned RowSet.
            var rows = rs.GetRows().ToList();
            return rows;
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
            return null!;
        }
    }
```

```
}
```

- API 세부 정보는 AWS SDK for .NET API 참조의 다음 주제를 참조하십시오.
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

Java

SDK for Java 2.x

> **Note**
>
> 더 많은 것이 있어요 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 * Before running this Java (v2) code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * Before running this Java code example, you must create a
 * Java keystore (JKS) file and place it in your project's resources folder.
```

```
 *
 * This file is a secure file format used to hold certificate information for
 * Java applications. This is required to make a connection to Amazon Keyspaces.
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/keyspaces/latest/devguide/using_java_driver.html
 *
 * This Java example performs the following tasks:
 *
 * 1. Create a keyspace.
 * 2. Check for keyspace existence.
 * 3. List keyspaces using a paginator.
 * 4. Create a table with a simple movie data schema and enable point-in-time
 * recovery.
 * 5. Check for the table to be in an Active state.
 * 6. List all tables in the keyspace.
 * 7. Use a Cassandra driver to insert some records into the Movie table.
 * 8. Get all records from the Movie table.
 * 9. Get a specific Movie.
 * 10. Get a UTC timestamp for the current time.
 * 11. Update the table schema to add a ‘watched’ Boolean column.
 * 12. Update an item as watched.
 * 13. Query for items with watched = True.
 * 14. Restore the table back to the previous state using the timestamp.
 * 15. Check for completion of the restore action.
 * 16. Delete the table.
 * 17. Confirm that both tables are deleted.
 * 18. Delete the keyspace.
 */

public class ScenarioKeyspaces {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    /*
     * Usage:
     * fileName - The name of the JSON file that contains movie data. (Get this
 file
     * from the GitHub repo at resources/sample_file.)
     * keyspaceName - The name of the keyspace to create.
     */
    public static void main(String[] args) throws InterruptedException,
 IOException {
```

```
        String fileName = "<Replace with the JSON file that contains movie
data>";
        String keyspaceName = "<Replace with the name of the keyspace to
create>";
        String titleUpdate = "The Family";
        int yearUpdate = 2013;
        String tableName = "Movie";
        String tableNameRestore = "MovieRestore";
        Region region = Region.US_EAST_1;
        KeyspacesClient keyClient = KeyspacesClient.builder()
                .region(region)
                .build();

        DriverConfigLoader loader =
DriverConfigLoader.fromClasspath("application.conf");
        CqlSession session = CqlSession.builder()
                .withConfigLoader(loader)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Keyspaces example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("1. Create a keyspace.");
        createKeySpace(keyClient, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        Thread.sleep(5000);
        System.out.println("2. Check for keyspace existence.");
        checkKeyspaceExistence(keyClient, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("3. List keyspaces using a paginator.");
        listKeyspacesPaginator(keyClient);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Create a table with a simple movie data schema and
enable point-in-time recovery.");
        createTable(keyClient, keyspaceName, tableName);
        System.out.println(DASHES);
```

```
        System.out.println(DASHES);
        System.out.println("5. Check for the table to be in an Active state.");
        Thread.sleep(6000);
        checkTable(keyClient, keyspaceName, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. List all tables in the keyspace.");
        listTables(keyClient, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Use a Cassandra driver to insert some records into
the Movie table.");
        Thread.sleep(6000);
        loadData(session, fileName, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Get all records from the Movie table.");
        getMovieData(session, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9. Get a specific Movie.");
        getSpecificMovie(session, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("10. Get a UTC timestamp for the current time.");
        ZonedDateTime utc = ZonedDateTime.now(ZoneOffset.UTC);
        System.out.println("DATETIME = " + Date.from(utc.toInstant()));
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("11. Update the table schema to add a watched Boolean
column.");
        updateTable(keyClient, keyspaceName, tableName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("12. Update an item as watched.");
        Thread.sleep(10000); // Wait 10 secs for the update.
```

```
        updateRecord(session, keyspaceName, titleUpdate, yearUpdate);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("13. Query for items with watched = True.");
        getWatchedData(session, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("14. Restore the table back to the previous state
using the timestamp.");
        System.out.println("Note that the restore operation can take up to 20
minutes.");
        restoreTable(keyClient, keyspaceName, utc);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("15. Check for completion of the restore action.");
        Thread.sleep(5000);
        checkRestoredTable(keyClient, keyspaceName, "MovieRestore");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("16. Delete both tables.");
        deleteTable(keyClient, keyspaceName, tableName);
        deleteTable(keyClient, keyspaceName, tableNameRestore);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("17. Confirm that both tables are deleted.");
        checkTableDelete(keyClient, keyspaceName, tableName);
        checkTableDelete(keyClient, keyspaceName, tableNameRestore);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("18. Delete the keyspace.");
        deleteKeyspace(keyClient, keyspaceName);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("The scenario has completed successfully.");
        System.out.println(DASHES);
    }
```

```
   public static void deleteKeyspace(KeyspacesClient keyClient, String
keyspaceName) {
       try {
           DeleteKeyspaceRequest deleteKeyspaceRequest =
DeleteKeyspaceRequest.builder()
                   .keyspaceName(keyspaceName)
                   .build();

           keyClient.deleteKeyspace(deleteKeyspaceRequest);

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static void checkTableDelete(KeyspacesClient keyClient, String
keyspaceName, String tableName)
           throws InterruptedException {
       try {
           String status;
           GetTableResponse response;
           GetTableRequest tableRequest = GetTableRequest.builder()
                   .keyspaceName(keyspaceName)
                   .tableName(tableName)
                   .build();

           // Keep looping until table cannot be found and a
ResourceNotFoundException is
           // thrown.
           while (true) {
               response = keyClient.getTable(tableRequest);
               status = response.statusAsString();
               System.out.println(". The table status is " + status);
               Thread.sleep(500);
           }

       } catch (ResourceNotFoundException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
       }
       System.out.println("The table is deleted");
   }
```

```
   public static void deleteTable(KeyspacesClient keyClient, String
keyspaceName, String tableName) {
       try {
           DeleteTableRequest tableRequest = DeleteTableRequest.builder()
                   .keyspaceName(keyspaceName)
                   .tableName(tableName)
                   .build();

           keyClient.deleteTable(tableRequest);

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static void checkRestoredTable(KeyspacesClient keyClient, String
keyspaceName, String tableName)
           throws InterruptedException {
       try {
           boolean tableStatus = false;
           String status;
           GetTableResponse response = null;
           GetTableRequest tableRequest = GetTableRequest.builder()
                   .keyspaceName(keyspaceName)
                   .tableName(tableName)
                   .build();

           while (!tableStatus) {
               response = keyClient.getTable(tableRequest);
               status = response.statusAsString();
               System.out.println("The table status is " + status);

               if (status.compareTo("ACTIVE") == 0) {
                   tableStatus = true;
               }
               Thread.sleep(500);
           }

           List<ColumnDefinition> cols =
response.schemaDefinition().allColumns();
           for (ColumnDefinition def : cols) {
               System.out.println("The column name is " + def.name());
               System.out.println("The column type is " + def.type());
```

```
            }

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void restoreTable(KeyspacesClient keyClient, String
keyspaceName, ZonedDateTime utc) {
        try {
            Instant myTime = utc.toInstant();
            RestoreTableRequest restoreTableRequest =
RestoreTableRequest.builder()
                    .restoreTimestamp(myTime)
                    .sourceTableName("Movie")
                    .targetKeyspaceName(keyspaceName)
                    .targetTableName("MovieRestore")
                    .sourceKeyspaceName(keyspaceName)
                    .build();

            RestoreTableResponse response =
keyClient.restoreTable(restoreTableRequest);
            System.out.println("The ARN of the restored table is " +
response.restoredTableARN());

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void getWatchedData(CqlSession session, String keyspaceName) {
        ResultSet resultSet = session
                .execute("SELECT * FROM \"" + keyspaceName + "\".\"Movie\" WHERE
watched = true ALLOW FILTERING;");
        resultSet.forEach(item -> {
            System.out.println("The Movie title is " + item.getString("title"));
            System.out.println("The Movie year is " + item.getInt("year"));
            System.out.println("The plot is " + item.getString("plot"));
        });
    }
```

```
   public static void updateRecord(CqlSession session, String keySpace, String
titleUpdate, int yearUpdate) {
       String sqlStatement = "UPDATE \"" + keySpace
               + "\".\"Movie\" SET watched=true WHERE title = :k0 AND year
= :k1;";
       BatchStatementBuilder builder =
BatchStatement.builder(DefaultBatchType.UNLOGGED);
       builder.setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM);
       PreparedStatement preparedStatement = session.prepare(sqlStatement);
       builder.addStatement(preparedStatement.boundStatementBuilder()
               .setString("k0", titleUpdate)
               .setInt("k1", yearUpdate)
               .build());

       BatchStatement batchStatement = builder.build();
       session.execute(batchStatement);
   }

   public static void updateTable(KeyspacesClient keyClient, String keySpace,
String tableName) {
       try {
           ColumnDefinition def = ColumnDefinition.builder()
                   .name("watched")
                   .type("boolean")
                   .build();

           UpdateTableRequest tableRequest = UpdateTableRequest.builder()
                   .keyspaceName(keySpace)
                   .tableName(tableName)
                   .addColumns(def)
                   .build();

           keyClient.updateTable(tableRequest);

       } catch (KeyspacesException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static void getSpecificMovie(CqlSession session, String keyspaceName)
{
       ResultSet resultSet = session.execute(
```

```
                "SELECT * FROM \"" + keyspaceName + "\".\"Movie\" WHERE title =
'The Family' ALLOW FILTERING ;");
        resultSet.forEach(item -> {
            System.out.println("The Movie title is " + item.getString("title"));
            System.out.println("The Movie year is " + item.getInt("year"));
            System.out.println("The plot is " + item.getString("plot"));
        });
    }

    // Get records from the Movie table.
    public static void getMovieData(CqlSession session, String keyspaceName) {
        ResultSet resultSet = session.execute("SELECT * FROM \"" + keyspaceName +
"\".\"Movie\";");
        resultSet.forEach(item -> {
            System.out.println("The Movie title is " + item.getString("title"));
            System.out.println("The Movie year is " + item.getInt("year"));
            System.out.println("The plot is " + item.getString("plot"));
        });
    }

    // Load data into the table.
    public static void loadData(CqlSession session, String fileName, String
keySpace) throws IOException {
        String sqlStatement = "INSERT INTO \"" + keySpace + "\".\"Movie\" (title,
year, plot) values (:k0, :k1, :k2)";
        JsonParser parser = new JsonFactory().createParser(new File(fileName));
        com.fasterxml.jackson.databind.JsonNode rootNode = new
ObjectMapper().readTree(parser);
        Iterator<JsonNode> iter = rootNode.iterator();
        ObjectNode currentNode;
        int t = 0;
        while (iter.hasNext()) {

            // Add 20 movies to the table.
            if (t == 20)
                break;
            currentNode = (ObjectNode) iter.next();

            int year = currentNode.path("year").asInt();
            String title = currentNode.path("title").asText();
            String plot = currentNode.path("info").path("plot").toString();

            // Insert the data into the Amazon Keyspaces table.
```

```
            BatchStatementBuilder builder =
BatchStatement.builder(DefaultBatchType.UNLOGGED);
            builder.setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM);
            PreparedStatement preparedStatement = session.prepare(sqlStatement);
            builder.addStatement(preparedStatement.boundStatementBuilder()
                    .setString("k0", title)
                    .setInt("k1", year)
                    .setString("k2", plot)
                    .build());

            BatchStatement batchStatement = builder.build();
            session.execute(batchStatement);
            t++;
        }

        System.out.println("You have added " + t + " records successfully!");
    }

    public static void listTables(KeyspacesClient keyClient, String keyspaceName)
{
        try {
            ListTablesRequest tablesRequest = ListTablesRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();

            ListTablesIterable listRes =
keyClient.listTablesPaginator(tablesRequest);
            listRes.stream()
                    .flatMap(r -> r.tables().stream())
                    .forEach(content -> System.out.println(" ARN: " +
content.resourceArn() +
                            " Table name: " + content.tableName()));

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void checkTable(KeyspacesClient keyClient, String keyspaceName,
String tableName)
            throws InterruptedException {
        try {
            boolean tableStatus = false;
```

```
            String status;
            GetTableResponse response = null;
            GetTableRequest tableRequest = GetTableRequest.builder()
                    .keyspaceName(keyspaceName)
                    .tableName(tableName)
                    .build();

            while (!tableStatus) {
                response = keyClient.getTable(tableRequest);
                status = response.statusAsString();
                System.out.println(". The table status is " + status);

                if (status.compareTo("ACTIVE") == 0) {
                    tableStatus = true;
                }
                Thread.sleep(500);
            }

            List<ColumnDefinition> cols =
response.schemaDefinition().allColumns();
            for (ColumnDefinition def : cols) {
                System.out.println("The column name is " + def.name());
                System.out.println("The column type is " + def.type());
            }

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createTable(KeyspacesClient keyClient, String keySpace,
String tableName) {
        try {
            // Set the columns.
            ColumnDefinition defTitle = ColumnDefinition.builder()
                    .name("title")
                    .type("text")
                    .build();

            ColumnDefinition defYear = ColumnDefinition.builder()
                    .name("year")
                    .type("int")
                    .build();
```

```
            ColumnDefinition defReleaseDate = ColumnDefinition.builder()
                    .name("release_date")
                    .type("timestamp")
                    .build();

            ColumnDefinition defPlot = ColumnDefinition.builder()
                    .name("plot")
                    .type("text")
                    .build();

            List<ColumnDefinition> colList = new ArrayList<>();
            colList.add(defTitle);
            colList.add(defYear);
            colList.add(defReleaseDate);
            colList.add(defPlot);

            // Set the keys.
            PartitionKey yearKey = PartitionKey.builder()
                    .name("year")
                    .build();

            PartitionKey titleKey = PartitionKey.builder()
                    .name("title")
                    .build();

            List<PartitionKey> keyList = new ArrayList<>();
            keyList.add(yearKey);
            keyList.add(titleKey);

            SchemaDefinition schemaDefinition = SchemaDefinition.builder()
                    .partitionKeys(keyList)
                    .allColumns(colList)
                    .build();

            PointInTimeRecovery timeRecovery = PointInTimeRecovery.builder()
                    .status(PointInTimeRecoveryStatus.ENABLED)
                    .build();

            CreateTableRequest tableRequest = CreateTableRequest.builder()
                    .keyspaceName(keySpace)
                    .tableName(tableName)
                    .schemaDefinition(schemaDefinition)
                    .pointInTimeRecovery(timeRecovery)
```

```
                    .build();

            CreateTableResponse response = keyClient.createTable(tableRequest);
            System.out.println("The table ARN is " + response.resourceArn());

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void listKeyspacesPaginator(KeyspacesClient keyClient) {
        try {
            ListKeyspacesRequest keyspacesRequest =
ListKeyspacesRequest.builder()
                    .maxResults(10)
                    .build();

            ListKeyspacesIterable listRes =
keyClient.listKeyspacesPaginator(keyspacesRequest);
            listRes.stream()
                    .flatMap(r -> r.keyspaces().stream())
                    .forEach(content -> System.out.println(" Name: " +
content.keyspaceName()));

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void checkKeyspaceExistence(KeyspacesClient keyClient, String
keyspaceName) {
        try {
            GetKeyspaceRequest keyspaceRequest = GetKeyspaceRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();

            GetKeyspaceResponse response =
keyClient.getKeyspace(keyspaceRequest);
            String name = response.keyspaceName();
            System.out.println("The " + name + " KeySpace is ready");

        } catch (KeyspacesException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void createKeySpace(KeyspacesClient keyClient, String
 keyspaceName) {
        try {
            CreateKeyspaceRequest keyspaceRequest =
 CreateKeyspaceRequest.builder()
                    .keyspaceName(keyspaceName)
                    .build();

            CreateKeyspaceResponse response =
 keyClient.createKeyspace(keyspaceRequest);
            System.out.println("The ARN of the KeySpace is " +
 response.resourceArn());

        } catch (KeyspacesException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- API 세부 정보는 AWS SDK for Java 2.x API 참조의 다음 주제를 참조하십시오.
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

Kotlin

SDK for Kotlin

> **Note**
>
> 더 많은 것이 있어요 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

```
/**
 Before running this Kotlin code example, set up your development environment,
 including your credentials.

 For more information, see the following documentation topic:

 https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

 This example uses a secure file format to hold certificate information for
 Kotlin applications. This is required to make a connection to Amazon Keyspaces.
 For more information, see the following documentation topic:

 https://docs.aws.amazon.com/keyspaces/latest/devguide/using_java_driver.html

 This Kotlin example performs the following tasks:

 1. Create a keyspace.
 2. Check for keyspace existence.
 3. List keyspaces using a paginator.
 4. Create a table with a simple movie data schema and enable point-in-time
 recovery.
 5. Check for the table to be in an Active state.
 6. List all tables in the keyspace.
 7. Use a Cassandra driver to insert some records into the Movie table.
 8. Get all records from the Movie table.
 9. Get a specific Movie.
 10. Get a UTC timestamp for the current time.
 11. Update the table schema to add a ‘watched’ Boolean column.
 12. Update an item as watched.
 13. Query for items with watched = True.
 14. Restore the table back to the previous state using the timestamp.
```

```
 15. Check for completion of the restore action.
 16. Delete the table.
 17. Confirm that both tables are deleted.
 18. Delete the keyspace.
 */

/*
   Usage:
     fileName - The name of the JSON file that contains movie data. (Get this
 file from the GitHub repo at resources/sample_file.)
     keyspaceName - The name of the keyspace to create.
 */
val DASHES: String = String(CharArray(80)).replace("\u0000", "-")

suspend fun main() {
    val fileName = "<Replace with the JSON file that contains movie data>"
    val keyspaceName = "<Replace with the name of the keyspace to create>"
    val titleUpdate = "The Family"
    val yearUpdate = 2013
    val tableName = "MovieKotlin"
    val tableNameRestore = "MovieRestore"

    val loader = DriverConfigLoader.fromClasspath("application.conf")
    val session =
        CqlSession
            .builder()
            .withConfigLoader(loader)
            .build()

    println(DASHES)
    println("Welcome to the Amazon Keyspaces example scenario.")
    println(DASHES)

    println(DASHES)
    println("1. Create a keyspace.")
    createKeySpace(keyspaceName)
    println(DASHES)

    println(DASHES)
    delay(5000)
    println("2. Check for keyspace existence.")
    checkKeyspaceExistence(keyspaceName)
    println(DASHES)
```

```
    println(DASHES)
    println("3. List keyspaces using a paginator.")
    listKeyspacesPaginator()
    println(DASHES)

    println(DASHES)
    println("4. Create a table with a simple movie data schema and enable point-
in-time recovery.")
    createTable(keyspaceName, tableName)
    println(DASHES)

    println(DASHES)
    println("5. Check for the table to be in an Active state.")
    delay(6000)
    checkTable(keyspaceName, tableName)
    println(DASHES)

    println(DASHES)
    println("6. List all tables in the keyspace.")
    listTables(keyspaceName)
    println(DASHES)

    println(DASHES)
    println("7. Use a Cassandra driver to insert some records into the Movie
 table.")
    delay(6000)
    loadData(session, fileName, keyspaceName)
    println(DASHES)

    println(DASHES)
    println("8. Get all records from the Movie table.")
    getMovieData(session, keyspaceName)
    println(DASHES)

    println(DASHES)
    println("9. Get a specific Movie.")
    getSpecificMovie(session, keyspaceName)
    println(DASHES)

    println(DASHES)
    println("10. Get a UTC timestamp for the current time.")
    val utc = ZonedDateTime.now(ZoneOffset.UTC)
    println("DATETIME = ${Date.from(utc.toInstant())}")
    println(DASHES)
```

```
    println(DASHES)
    println("11. Update the table schema to add a watched Boolean column.")
    updateTable(keyspaceName, tableName)
    println(DASHES)

    println(DASHES)
    println("12. Update an item as watched.")
    delay(10000) // Wait 10 seconds for the update.
    updateRecord(session, keyspaceName, titleUpdate, yearUpdate)
    println(DASHES)

    println(DASHES)
    println("13. Query for items with watched = True.")
    getWatchedData(session, keyspaceName)
    println(DASHES)

    println(DASHES)
    println("14. Restore the table back to the previous state using the
timestamp.")
    println("Note that the restore operation can take up to 20 minutes.")
    restoreTable(keyspaceName, utc)
    println(DASHES)

    println(DASHES)
    println("15. Check for completion of the restore action.")
    delay(5000)
    checkRestoredTable(keyspaceName, "MovieRestore")
    println(DASHES)

    println(DASHES)
    println("16. Delete both tables.")
    deleteTable(keyspaceName, tableName)
    deleteTable(keyspaceName, tableNameRestore)
    println(DASHES)

    println(DASHES)
    println("17. Confirm that both tables are deleted.")
    checkTableDelete(keyspaceName, tableName)
    checkTableDelete(keyspaceName, tableNameRestore)
    println(DASHES)

    println(DASHES)
    println("18. Delete the keyspace.")
```

```
    deleteKeyspace(keyspaceName)
    println(DASHES)

    println(DASHES)
    println("The scenario has completed successfully.")
    println(DASHES)
}

suspend fun deleteKeyspace(keyspaceNameVal: String?) {
    val deleteKeyspaceRequest =
        DeleteKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteKeyspace(deleteKeyspaceRequest)
    }
}

suspend fun checkTableDelete(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    var status: String
    var response: GetTableResponse
    val tableRequest =
        GetTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }

    try {
        KeyspacesClient { region = "us-east-1" }.use { keyClient ->
            // Keep looping until the table cannot be found and a
 ResourceNotFoundException is thrown.
            while (true) {
                response = keyClient.getTable(tableRequest)
                status = response.status.toString()
                println(". The table status is $status")
                delay(500)
            }
        }
    } catch (e: ResourceNotFoundException) {
        println(e.message)
```

```
    }
    println("The table is deleted")
}

suspend fun deleteTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    val tableRequest =
        DeleteTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient.deleteTable(tableRequest)
    }
}

suspend fun checkRestoredTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    var tableStatus = false
    var status: String
    var response: GetTableResponse? = null

    val tableRequest =
        GetTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        while (!tableStatus) {
            response = keyClient.getTable(tableRequest)
            status = response!!.status.toString()
            println("The table status is $status")

            if (status.compareTo("ACTIVE") == 0) {
                tableStatus = true
            }
            delay(500)
        }
```

```
        val cols = response!!.schemaDefinition?.allColumns
        if (cols != null) {
            for (def in cols) {
                println("The column name is ${def.name}")
                println("The column type is ${def.type}")
            }
        }
    }
}

suspend fun restoreTable(
    keyspaceName: String?,
    utc: ZonedDateTime,
) {
    // Create an aws.smithy.kotlin.runtime.time.Instant value.
    val timeStamp =
        aws.smithy.kotlin.runtime.time
            .Instant(utc.toInstant())
    val restoreTableRequest =
        RestoreTableRequest {
            restoreTimestamp = timeStamp
            sourceTableName = "MovieKotlin"
            targetKeyspaceName = keyspaceName
            targetTableName = "MovieRestore"
            sourceKeyspaceName = keyspaceName
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.restoreTable(restoreTableRequest)
        println("The ARN of the restored table is ${response.restoredTableArn}")
    }
}

fun getWatchedData(
    session: CqlSession,
    keyspaceName: String,
) {
    val resultSet = session.execute("SELECT * FROM \"$keyspaceName\".
\"MovieKotlin\" WHERE watched = true ALLOW FILTERING;")
    resultSet.forEach { item: Row ->
        println("The Movie title is ${item.getString("title")}")
        println("The Movie year is ${item.getInt("year")}")
        println("The plot is ${item.getString("plot")}")
```

```
    }
}

fun updateRecord(
    session: CqlSession,
    keySpace: String,
    titleUpdate: String?,
    yearUpdate: Int,
) {
    val sqlStatement =
        "UPDATE \"$keySpace\".\"MovieKotlin\" SET watched=true WHERE title = :k0
 AND year = :k1;"
    val builder = BatchStatement.builder(DefaultBatchType.UNLOGGED)
    builder.setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM)
    val preparedStatement = session.prepare(sqlStatement)
    builder.addStatement(
        preparedStatement
            .boundStatementBuilder()
            .setString("k0", titleUpdate)
            .setInt("k1", yearUpdate)
            .build(),
    )
    val batchStatement = builder.build()
    session.execute(batchStatement)
}

suspend fun updateTable(
    keySpace: String?,
    tableNameVal: String?,
) {
    val def =
        ColumnDefinition {
            name = "watched"
            type = "boolean"
        }

    val tableRequest =
        UpdateTableRequest {
            keyspaceName = keySpace
            tableName = tableNameVal
            addColumns = listOf(def)
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
```

```
        keyClient.updateTable(tableRequest)
    }
}

fun getSpecificMovie(
    session: CqlSession,
    keyspaceName: String,
) {
    val resultSet =
        session.execute("SELECT * FROM \"$keyspaceName\".\"MovieKotlin\" WHERE
 title = 'The Family' ALLOW FILTERING ;")

    resultSet.forEach { item: Row ->
        println("The Movie title is ${item.getString("title")}")
        println("The Movie year is ${item.getInt("year")}")
        println("The plot is ${item.getString("plot")}")
    }
}

// Get records from the Movie table.
fun getMovieData(
    session: CqlSession,
    keyspaceName: String,
) {
    val resultSet = session.execute("SELECT * FROM \"$keyspaceName\".
\"MovieKotlin\";")
    resultSet.forEach { item: Row ->
        println("The Movie title is ${item.getString("title")}")
        println("The Movie year is ${item.getInt("year")}")
        println("The plot is ${item.getString("plot")}")
    }
}

// Load data into the table.
fun loadData(
    session: CqlSession,
    fileName: String,
    keySpace: String,
) {
    val sqlStatement =
        "INSERT INTO \"$keySpace\".\"MovieKotlin\" (title, year, plot) values
 (:k0, :k1, :k2)"
    val parser = JsonFactory().createParser(File(fileName))
    val rootNode = ObjectMapper().readTree<JsonNode>(parser)
```

```
    val iter: Iterator<JsonNode> = rootNode.iterator()
    var currentNode: ObjectNode

    var t = 0
    while (iter.hasNext()) {
        if (t == 50) {
            break
        }

        currentNode = iter.next() as ObjectNode
        val year = currentNode.path("year").asInt()
        val title = currentNode.path("title").asText()
        val info = currentNode.path("info").toString()

        // Insert the data into the Amazon Keyspaces table.
        val builder = BatchStatement.builder(DefaultBatchType.UNLOGGED)
        builder.setConsistencyLevel(ConsistencyLevel.LOCAL_QUORUM)
        val preparedStatement: PreparedStatement = session.prepare(sqlStatement)
        builder.addStatement(
            preparedStatement
                .boundStatementBuilder()
                .setString("k0", title)
                .setInt("k1", year)
                .setString("k2", info)
                .build(),
        )

        val batchStatement = builder.build()
        session.execute(batchStatement)
        t++
    }
}

suspend fun listTables(keyspaceNameVal: String?) {
    val tablesRequest =
        ListTablesRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient
            .listTablesPaginated(tablesRequest)
            .transform { it.tables?.forEach { obj -> emit(obj) } }
            .collect { obj ->
```

```
                println(" ARN: ${obj.resourceArn} Table name: ${obj.tableName}")
            }
    }
}

suspend fun checkTable(
    keyspaceNameVal: String?,
    tableNameVal: String?,
) {
    var tableStatus = false
    var status: String
    var response: GetTableResponse? = null

    val tableRequest =
        GetTableRequest {
            keyspaceName = keyspaceNameVal
            tableName = tableNameVal
        }
    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        while (!tableStatus) {
            response = keyClient.getTable(tableRequest)
            status = response!!.status.toString()
            println(". The table status is $status")
            if (status.compareTo("ACTIVE") == 0) {
                tableStatus = true
            }
            delay(500)
        }
        val cols: List<ColumnDefinition>? =
 response!!.schemaDefinition?.allColumns
        if (cols != null) {
            for (def in cols) {
                println("The column name is ${def.name}")
                println("The column type is ${def.type}")
            }
        }
    }
}

suspend fun createTable(
    keySpaceVal: String?,
    tableNameVal: String?,
) {
    // Set the columns.
```

```
    val defTitle =
        ColumnDefinition {
            name = "title"
            type = "text"
        }

    val defYear =
        ColumnDefinition {
            name = "year"
            type = "int"
        }

    val defReleaseDate =
        ColumnDefinition {
            name = "release_date"
            type = "timestamp"
        }

    val defPlot =
        ColumnDefinition {
            name = "plot"
            type = "text"
        }

    val colList = ArrayList<ColumnDefinition>()
    colList.add(defTitle)
    colList.add(defYear)
    colList.add(defReleaseDate)
    colList.add(defPlot)

    // Set the keys.
    val yearKey =
        PartitionKey {
            name = "year"
        }

    val titleKey =
        PartitionKey {
            name = "title"
        }

    val keyList = ArrayList<PartitionKey>()
    keyList.add(yearKey)
    keyList.add(titleKey)
```

```
    val schemaDefinitionOb =
        SchemaDefinition {
            partitionKeys = keyList
            allColumns = colList
        }

    val timeRecovery =
        PointInTimeRecovery {
            status = PointInTimeRecoveryStatus.Enabled
        }

    val tableRequest =
        CreateTableRequest {
            keyspaceName = keySpaceVal
            tableName = tableNameVal
            schemaDefinition = schemaDefinitionOb
            pointInTimeRecovery = timeRecovery
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createTable(tableRequest)
        println("The table ARN is ${response.resourceArn}")
    }
}

suspend fun listKeyspacesPaginator() {
    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        keyClient
            .listKeyspacesPaginated(ListKeyspacesRequest {})
            .transform { it.keyspaces?.forEach { obj -> emit(obj) } }
            .collect { obj ->
                println("Name: ${obj.keyspaceName}")
            }
    }
}

suspend fun checkKeyspaceExistence(keyspaceNameVal: String?) {
    val keyspaceRequest =
        GetKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }
    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
```

```
        val response: GetKeyspaceResponse =
 keyClient.getKeyspace(keyspaceRequest)
        val name = response.keyspaceName
        println("The $name KeySpace is ready")
    }
}

suspend fun createKeySpace(keyspaceNameVal: String) {
    val keyspaceRequest =
        CreateKeyspaceRequest {
            keyspaceName = keyspaceNameVal
        }

    KeyspacesClient { region = "us-east-1" }.use { keyClient ->
        val response = keyClient.createKeyspace(keyspaceRequest)
        println("The ARN of the KeySpace is ${response.resourceArn}")
    }
}
```

- API 세부 정보는 AWS SDK for Kotlin API reference의 다음 주제를 참조하세요.
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

Python

SDK for Python(Boto3)

> **Note**
>
> 더 많은 것이 있어요 GitHub. AWS 코드 예제 리포지토리에서 전체 예제를 찾고 설정 및 실행하는 방법을 배워보세요.

명령 프롬프트에서 대화형 시나리오를 실행합니다.

```
class KeyspaceScenario:
    """Runs an interactive scenario that shows how to get started using Amazon
 Keyspaces."""

    def __init__(self, ks_wrapper):
        """
        :param ks_wrapper: An object that wraps Amazon Keyspace actions.
        """
        self.ks_wrapper = ks_wrapper

    @demo_func
    def create_keyspace(self):
        """
        1. Creates a keyspace.
        2. Lists up to 10 keyspaces in your account.
        """
        print("Let's create a keyspace.")
        ks_name = q.ask(
            "Enter a name for your new keyspace.\nThe name can contain only
 letters, "
            "numbers and underscores: ",
            q.non_empty,
        )
        if self.ks_wrapper.exists_keyspace(ks_name):
            print(f"A keyspace named {ks_name} exists.")
        else:
            ks_arn = self.ks_wrapper.create_keyspace(ks_name)
            ks_exists = False
            while not ks_exists:
                wait(3)
                ks_exists = self.ks_wrapper.exists_keyspace(ks_name)
```

```
            print(f"Created a new keyspace.\n\t{ks_arn}.")
        print("The first 10 keyspaces in your account are:\n")
        self.ks_wrapper.list_keyspaces(10)

    @demo_func
    def create_table(self):
        """
        1. Creates a table in the keyspace. The table is configured with a schema
to hold
           movie data and has point-in-time recovery enabled.
        2. Waits for the table to be in an active state.
        3. Displays schema information for the table.
        4. Lists tables in the keyspace.
        """
        print("Let's create a table for movies in your keyspace.")
        table_name = q.ask("Enter a name for your table: ", q.non_empty)
        table = self.ks_wrapper.get_table(table_name)
        if table is not None:
            print(
                f"A table named {table_name} already exists in keyspace "
                f"{self.ks_wrapper.ks_name}."
            )
        else:
            table_arn = self.ks_wrapper.create_table(table_name)
            print(f"Created table {table_name}:\n\t{table_arn}")
            table = {"status": None}
            print("Waiting for your table to be ready...")
            while table["status"] != "ACTIVE":
                wait(5)
                table = self.ks_wrapper.get_table(table_name)
        print(f"Your table is {table['status']}. Its schema is:")
        pp(table["schemaDefinition"])
        print("\nThe tables in your keyspace are:\n")
        self.ks_wrapper.list_tables()

    @demo_func
    def ensure_tls_cert(self):
        """
        Ensures you have a TLS certificate available to use to secure the
connection
        to the keyspace. This function downloads a default certificate or lets
you
        specify your own.
        """
```

```
        print("To connect to your keyspace, you must have a TLS certificate.")
        print("Checking for TLS certificate...")
        cert_path = os.path.join(
            os.path.dirname(__file__), QueryManager.DEFAULT_CERT_FILE
        )
        if not os.path.exists(cert_path):
            cert_choice = q.ask(
                f"Press enter to download a certificate from
{QueryManager.CERT_URL} "
                f"or enter the full path to the certificate you want to use: "
            )
            if cert_choice:
                cert_path = cert_choice
            else:
                cert = requests.get(QueryManager.CERT_URL).text
                with open(cert_path, "w") as cert_file:
                    cert_file.write(cert)
        else:
            q.ask(f"Certificate {cert_path} found. Press Enter to continue.")
        print(
            f"Certificate {cert_path} will be used to secure the connection to
your keyspace."
        )
        return cert_path

    @demo_func
    def query_table(self, qm, movie_file):
        """
        1. Adds movies to the table from a sample movie data file.
        2. Gets a list of movies from the table and lets you select one.
        3. Displays more information about the selected movie.
        """
        qm.add_movies(self.ks_wrapper.table_name, movie_file)
        movies = qm.get_movies(self.ks_wrapper.table_name)
        print(f"Added {len(movies)} movies to the table:")
        sel = q.choose("Pick one to learn more about it: ", [m.title for m in
movies])
        movie_choice = qm.get_movie(
            self.ks_wrapper.table_name, movies[sel].title, movies[sel].year
        )
        print(movie_choice.title)
        print(f"\tReleased: {movie_choice.release_date}")
        print(f"\tPlot: {movie_choice.plot}")
```

```
    @demo_func
    def update_and_restore_table(self, qm):
        """
        1. Updates the table by adding a column to track watched movies.
        2. Marks some of the movies as watched.
        3. Gets the list of watched movies from the table.
        4. Restores to a movies_restored table at a previous point in time.
        5. Gets the list of movies from the restored table.
        """
        print("Let's add a column to record which movies you've watched.")
        pre_update_timestamp = datetime.utcnow()
        print(
            f"Recorded the current UTC time of {pre_update_timestamp} so we can
restore the table later."
        )
        self.ks_wrapper.update_table()
        print("Waiting for your table to update...")
        table = {"status": "UPDATING"}
        while table["status"] != "ACTIVE":
            wait(5)
            table = self.ks_wrapper.get_table(self.ks_wrapper.table_name)
        print("Column 'watched' added to table.")
        q.ask(
            "Let's mark some of the movies as watched. Press Enter when you're
ready.\n"
        )
        movies = qm.get_movies(self.ks_wrapper.table_name)
        for movie in movies[:10]:
            qm.watched_movie(self.ks_wrapper.table_name, movie.title, movie.year)
            print(f"Marked {movie.title} as watched.")
        movies = qm.get_movies(self.ks_wrapper.table_name, watched=True)
        print("-" * 88)
        print("The watched movies in our table are:\n")
        for movie in movies:
            print(movie.title)
        print("-" * 88)
        if q.ask(
            "Do you want to restore the table to the way it was before all of
these\n"
            "updates? Keep in mind, this can take up to 20 minutes. (y/n) ",
            q.is_yesno,
        ):
            starting_table_name = self.ks_wrapper.table_name
```

```
            table_name_restored =
self.ks_wrapper.restore_table(pre_update_timestamp)
            table = {"status": "RESTORING"}
            while table["status"] != "ACTIVE":
                wait(10)
                table = self.ks_wrapper.get_table(table_name_restored)
            print(
                f"Restored {starting_table_name} to {table_name_restored} "
                f"at a point in time of {pre_update_timestamp}."
            )
            movies = qm.get_movies(table_name_restored)
            print("Now the movies in our table are:")
            for movie in movies:
                print(movie.title)

    def cleanup(self, cert_path):
        """
        1. Deletes the table and waits for it to be removed.
        2. Deletes the keyspace.

        :param cert_path: The path of the TLS certificate used in the demo. If
the
                          certificate was downloaded during the demo, it is
removed.
        """
        if q.ask(
            f"Do you want to delete your {self.ks_wrapper.table_name} table and "
            f"{self.ks_wrapper.ks_name} keyspace? (y/n) ",
            q.is_yesno,
        ):
            table_name = self.ks_wrapper.table_name
            self.ks_wrapper.delete_table()
            table = self.ks_wrapper.get_table(table_name)
            print("Waiting for the table to be deleted.")
            while table is not None:
                wait(5)
                table = self.ks_wrapper.get_table(table_name)
            print("Table deleted.")
            self.ks_wrapper.delete_keyspace()
            print(
                "Keyspace deleted. If you chose to restore your table during the
"
                "demo, the original table is also deleted."
            )
```

```
            if cert_path == os.path.join(
                os.path.dirname(__file__), QueryManager.DEFAULT_CERT_FILE
            ) and os.path.exists(cert_path):
                os.remove(cert_path)
                print("Removed certificate that was downloaded for this demo.")

    def run_scenario(self):
        logging.basicConfig(level=logging.INFO, format="%(levelname)s:
 %(message)s")

        print("-" * 88)
        print("Welcome to the Amazon Keyspaces (for Apache Cassandra) demo.")
        print("-" * 88)

        self.create_keyspace()
        self.create_table()
        cert_file_path = self.ensure_tls_cert()
        # Use a context manager to ensure the connection to the keyspace is
 closed.
        with QueryManager(
            cert_file_path, boto3.DEFAULT_SESSION, self.ks_wrapper.ks_name
        ) as qm:
            self.query_table(qm, "../../../resources/sample_files/movies.json")
            self.update_and_restore_table(qm)
        self.cleanup(cert_file_path)

        print("\nThanks for watching!")
        print("-" * 88)


if __name__ == "__main__":
    try:
        scenario = KeyspaceScenario(KeyspaceWrapper.from_client())
        scenario.run_scenario()
    except Exception:
        logging.exception("Something went wrong with the demo.")
```

키스페이스 및 테이블 작업을 래핑하는 클래스를 정의합니다.

```
class KeyspaceWrapper:
    """Encapsulates Amazon Keyspaces (for Apache Cassandra) keyspace and table
 actions."""
```

```
    def __init__(self, keyspaces_client):
        """
        :param keyspaces_client: A Boto3 Amazon Keyspaces client.
        """
        self.keyspaces_client = keyspaces_client
        self.ks_name = None
        self.ks_arn = None
        self.table_name = None

    @classmethod
    def from_client(cls):
        keyspaces_client = boto3.client("keyspaces")
        return cls(keyspaces_client)


    def create_keyspace(self, name):
        """
        Creates a keyspace.

        :param name: The name to give the keyspace.
        :return: The Amazon Resource Name (ARN) of the new keyspace.
        """
        try:
            response = self.keyspaces_client.create_keyspace(keyspaceName=name)
            self.ks_name = name
            self.ks_arn = response["resourceArn"]
        except ClientError as err:
            logger.error(
                "Couldn't create %s. Here's why: %s: %s",
                name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return self.ks_arn


    def exists_keyspace(self, name):
        """
        Checks whether a keyspace exists.

        :param name: The name of the keyspace to look up.
```

```
        :return: True when the keyspace exists. Otherwise, False.
        """
        try:
            response = self.keyspaces_client.get_keyspace(keyspaceName=name)
            self.ks_name = response["keyspaceName"]
            self.ks_arn = response["resourceArn"]
            exists = True
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Keyspace %s does not exist.", name)
                exists = False
            else:
                logger.error(
                    "Couldn't verify %s exists. Here's why: %s: %s",
                    name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return exists


    def list_keyspaces(self, limit):
        """
        Lists the keyspaces in your account.

        :param limit: The maximum number of keyspaces to list.
        """
        try:
            ks_paginator = self.keyspaces_client.get_paginator("list_keyspaces")
            for page in ks_paginator.paginate(PaginationConfig={"MaxItems":
limit}):
                for ks in page["keyspaces"]:
                    print(ks["keyspaceName"])
                    print(f"\t{ks['resourceArn']}")
        except ClientError as err:
            logger.error(
                "Couldn't list keyspaces. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

```
    def create_table(self, table_name):
        """
        Creates a table in the  keyspace.
        The table is created with a schema for storing movie data
        and has point-in-time recovery enabled.

        :param table_name: The name to give the table.
        :return: The ARN of the new table.
        """
        try:
            response = self.keyspaces_client.create_table(
                keyspaceName=self.ks_name,
                tableName=table_name,
                schemaDefinition={
                    "allColumns": [
                        {"name": "title", "type": "text"},
                        {"name": "year", "type": "int"},
                        {"name": "release_date", "type": "timestamp"},
                        {"name": "plot", "type": "text"},
                    ],
                    "partitionKeys": [{"name": "year"}, {"name": "title"}],
                },
                pointInTimeRecovery={"status": "ENABLED"},
            )
        except ClientError as err:
            logger.error(
                "Couldn't create table %s. Here's why: %s: %s",
                table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return response["resourceArn"]


    def get_table(self, table_name):
        """
        Gets data about a table in the keyspace.

        :param table_name: The name of the table to look up.
        :return: Data about the table.
        """
        try:
```

```
            response = self.keyspaces_client.get_table(
                keyspaceName=self.ks_name, tableName=table_name
            )
            self.table_name = table_name
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.info("Table %s does not exist.", table_name)
                self.table_name = None
                response = None
            else:
                logger.error(
                    "Couldn't verify %s exists. Here's why: %s: %s",
                    table_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return response


    def list_tables(self):
        """
        Lists the tables in the keyspace.
        """
        try:
            table_paginator = self.keyspaces_client.get_paginator("list_tables")
            for page in table_paginator.paginate(keyspaceName=self.ks_name):
                for table in page["tables"]:
                    print(table["tableName"])
                    print(f"\t{table['resourceArn']}")
        except ClientError as err:
            logger.error(
                "Couldn't list tables in keyspace %s. Here's why: %s: %s",
                self.ks_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def update_table(self):
        """
        Updates the schema of the table.
```

```
        This example updates a table of movie data by adding a new column
        that tracks whether the movie has been watched.
        """
        try:
            self.keyspaces_client.update_table(
                keyspaceName=self.ks_name,
                tableName=self.table_name,
                addColumns=[{"name": "watched", "type": "boolean"}],
            )
        except ClientError as err:
            logger.error(
                "Couldn't update table %s. Here's why: %s: %s",
                self.table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def restore_table(self, restore_timestamp):
        """
        Restores the table to a previous point in time. The table is restored
        to a new table in the same keyspace.

        :param restore_timestamp: The point in time to restore the table. This
time
                                  must be in UTC format.
        :return: The name of the restored table.
        """
        try:
            restored_table_name = f"{self.table_name}_restored"
            self.keyspaces_client.restore_table(
                sourceKeyspaceName=self.ks_name,
                sourceTableName=self.table_name,
                targetKeyspaceName=self.ks_name,
                targetTableName=restored_table_name,
                restoreTimestamp=restore_timestamp,
            )
        except ClientError as err:
            logger.error(
                "Couldn't restore table %s. Here's why: %s: %s",
                restore_timestamp,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
```

```
            )
            raise
        else:
            return restored_table_name


    def delete_table(self):
        """
        Deletes the table from the keyspace.
        """
        try:
            self.keyspaces_client.delete_table(
                keyspaceName=self.ks_name, tableName=self.table_name
            )
            self.table_name = None
        except ClientError as err:
            logger.error(
                "Couldn't delete table %s. Here's why: %s: %s",
                self.table_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise


    def delete_keyspace(self):
        """
        Deletes the keyspace.
        """
        try:
            self.keyspaces_client.delete_keyspace(keyspaceName=self.ks_name)
            self.ks_name = None
        except ClientError as err:
            logger.error(
                "Couldn't delete keyspace %s. Here's why: %s: %s",
                self.ks_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
```

키스페이스에 대한 TLS 연결을 생성하고, SigV4로 인증하고, 키스페이스의 테이블에 CQL 쿼리를 전송하는 클래스를 정의합니다.

```
class QueryManager:
    """
    Manages queries to an Amazon Keyspaces (for Apache Cassandra) keyspace.
    Queries are secured by TLS and authenticated by using the Signature V4
 (SigV4)
    AWS signing protocol. This is more secure than sending username and password
    with a plain-text authentication provider.

    This example downloads a default certificate to secure TLS, or lets you
 specify
    your own.

    This example uses a table of movie data to demonstrate basic queries.
    """

    DEFAULT_CERT_FILE = "sf-class2-root.crt"
    CERT_URL = f"https://certs.secureserver.net/repository/sf-class2-root.crt"

    def __init__(self, cert_file_path, boto_session, keyspace_name):
        """
        :param cert_file_path: The path and file name of the certificate used for
 TLS.
        :param boto_session: A Boto3 session. This is used to acquire your AWS
 credentials.
        :param keyspace_name: The name of the keyspace to connect.
        """
        self.cert_file_path = cert_file_path
        self.boto_session = boto_session
        self.ks_name = keyspace_name
        self.cluster = None
        self.session = None

    def __enter__(self):
        """
        Creates a session connection to the keyspace that is secured by TLS and
        authenticated by SigV4.
        """
        ssl_context = SSLContext(PROTOCOL_TLSv1_2)
```

```
        ssl_context.load_verify_locations(self.cert_file_path)
        ssl_context.verify_mode = CERT_REQUIRED
        auth_provider = SigV4AuthProvider(self.boto_session)
        contact_point = f"cassandra.
{self.boto_session.region_name}.amazonaws.com"
        exec_profile = ExecutionProfile(
            consistency_level=ConsistencyLevel.LOCAL_QUORUM,
            load_balancing_policy=DCAwareRoundRobinPolicy(),
        )
        self.cluster = Cluster(
            [contact_point],
            ssl_context=ssl_context,
            auth_provider=auth_provider,
            port=9142,
            execution_profiles={EXEC_PROFILE_DEFAULT: exec_profile},
            protocol_version=4,
        )
        self.cluster.__enter__()
        self.session = self.cluster.connect(self.ks_name)
        return self

    def __exit__(self, *args):
        """
        Exits the cluster. This shuts down all existing session connections.
        """
        self.cluster.__exit__(*args)

    def add_movies(self, table_name, movie_file_path):
        """
        Gets movies from a JSON file and adds them to a table in the keyspace.

        :param table_name: The name of the table.
        :param movie_file_path: The path and file name of a JSON file that
 contains movie data.
        """
        with open(movie_file_path, "r") as movie_file:
            movies = json.loads(movie_file.read())
        stmt = self.session.prepare(
            f"INSERT INTO {table_name} (year, title, release_date, plot) VALUES
 (?, ?, ?, ?);"
        )
        for movie in movies[:20]:
            self.session.execute(
                stmt,
```

```
                parameters=[
                    movie["year"],
                    movie["title"],
                    date.fromisoformat(movie["info"]
["release_date"].partition("T")[0]),
                    movie["info"]["plot"],
                ],
            )

    def get_movies(self, table_name, watched=None):
        """
        Gets the title and year of the full list of movies from the table.

        :param table_name: The name of the movie table.
        :param watched: When specified, the returned list of movies is filtered
 to
                        either movies that have been watched or movies that have
 not
                        been watched. Otherwise, all movies are returned.
        :return: A list of movies in the table.
        """
        if watched is None:
            stmt = SimpleStatement(f"SELECT title, year from {table_name}")
            params = None
        else:
            stmt = SimpleStatement(
                f"SELECT title, year from {table_name} WHERE watched = %s ALLOW
 FILTERING"
            )
            params = [watched]
        return self.session.execute(stmt, parameters=params).all()

    def get_movie(self, table_name, title, year):
        """
        Gets a single movie from the table, by title and year.

        :param table_name: The name of the movie table.
        :param title: The title of the movie.
        :param year: The year of the movie's release.
        :return: The requested movie.
        """
        return self.session.execute(
            SimpleStatement(
                f"SELECT * from {table_name} WHERE title = %s AND year = %s"
```

```
            ),
            parameters=[title, year],
        ).one()

    def watched_movie(self, table_name, title, year):
        """
        Updates a movie as having been watched.

        :param table_name: The name of the movie table.
        :param title: The title of the movie.
        :param year: The year of the movie's release.
        """
        self.session.execute(
            SimpleStatement(
                f"UPDATE {table_name} SET watched=true WHERE title = %s AND year
= %s"
            ),
            parameters=[title, year],
        )
```

- API 세부 정보는 AWS SDK for Python (Boto3) API 참조의 다음 주제를 참조하십시오.
  - CreateKeyspace
  - CreateTable
  - DeleteKeyspace
  - DeleteTable
  - GetKeyspace
  - GetTable
  - ListKeyspaces
  - ListTables
  - RestoreTable
  - UpdateTable

AWS SDK 개발자 가이드 및 코드 예제의 전체 목록은 을 참조하십시오아마존 키스페이스를 SDK와 함께 사용하기 AWS. 이 주제에는 시작하기에 대한 정보와 이전 SDK 버전에 대한 세부 정보도 포함되어 있습니다.

Amazon Keyspaces(Apache Cassandra용) 라이브러리 및 도구

이 섹션에서는 Amazon Keyspaces(Apache Cassandra용) 라이브러리, 코드 예제, 도구에 대한 정보를 제공합니다.

주제

라이브러리 및 예제

Amazon Keyspaces 오픈 소스 라이브러리 및 개발자 도구는 GitHub의 AWS 및 AWS 샘플 리포지토리에서 찾을 수 있습니다.

Amazon Keyspaces(Apache Cassandra용) 개발자 도구 키트

이 리포지토리는 Amazon Keyspaces에 유용한 개발자 도구가 포함된 도커 이미지를 제공합니다. 예를 들어 모범 사례가 포함된 CQLSHRC 파일, cqlsh에 대한 선택적 AWS 인증 확장, 일반적인 작업을 수행하기 위한 도우미 도구가 포함되어 있습니다. 이 도구 키트는 Amazon Keyspaces에 최적화되어 있지만 Apache Cassandra 클러스터와도 호환됩니다.

https://github.com/aws-samples/amazon-keyspaces-toolkit.

Amazon Keyspaces(Apache Cassandra용) 예제

이 리포지토리는 Amazon Keyspaces 예제 코드의 공식 목록입니다. 리포지토리는 언어별 섹션으로 세분화되어 있습니다(예제 참조). 각 언어마다 고유한 예제 하위 섹션이 있습니다. 이 예제는 애플리케이션을 구축할 때 사용할 수 있는 일반적인 Amazon Keyspaces 서비스 구현 및 패턴을 보여줍니다.

https://github.com/aws-samples/amazon-keyspaces-examples/.

AWS Signature Version 4(SigV4) 인증 플러그인

플러그인을 사용하면 AWS Identity and Access Management(IAM) 사용자 및 역할을 사용하여 Amazon Keyspaces에 대한 액세스를 관리할 수 있습니다.

Java: https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin.

Node.js: https://github.com/aws/aws-sigv4-auth-cassandra-nodejs-driver-plugin.

Python: https://github.com/aws/aws-sigv4-auth-cassandra-python-driver-plugin.

Go: https://github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin.

강조 표시된 샘플 및 개발자 도구 리포지토리

다음은 Amazon Keyspaces(Apache Cassandra용)에서 사용되는 유용한 커뮤니티 도구 모음입니다.

Amazon Keyspaces 프로토콜 버퍼

Amazon Keyspaces에 프로토콜 버퍼 (Protobuf)를 사용하여 Apache Cassandra 사용자 정의 유형(UDT)에 대한 대안을 제공할 수 있습니다. Protobuf는 구조화된 데이터를 직렬화하는 데 사용되는 무료 오픈 소스 크로스 플랫폼 데이터 형식입니다. CQL BLOB 데이터 유형을 사용하여 Protobuf 데이터를 저장하고 UDT를 리팩터링하는 동시에 여러 애플리케이션 및 프로그래밍 언어에 걸쳐 구조화된 데이터를 보존할 수 있습니다.

이 리포지토리는 Amazon Keyspaces에 연결하고, 새 테이블을 생성하고, Protobuf 메시지가 포함된 행을 삽입하는 코드 예제를 제공합니다. 그러면 행 읽기의 일관성이 향상됩니다.

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/protobuf-user-defined-types

AWS CloudFormationAmazon Keyspaces(Apache Cassandra용) 지표에 대한 Amazon CloudWatch 대시보드를 생성하는 템플릿

이 리포지토리는 Amazon Keyspaces에 대한 CloudWatch 지표를 빠르게 설정할 수 있는 AWS CloudFormation 템플릿을 제공합니다. 이 템플릿을 사용하여 일반적으로 사용되는 지표와 함께 배포 가능한 사전 구축된 CloudWatch 대시보드를 제공하여 더 쉽게 시작할 수 있습니다.

https://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates.

AWS Lambda과 함께 Amazon Keyspaces(Apache Cassandra용) 사용

리포지토리에는 Lambda에서 Amazon Keyspaces에 접속하는 방법을 보여주는 예제가 포함되어 있습니다. 다음은 일부 예제입니다.

C#/.NET: https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/dotnet/datastax-v3/connection-lambda.

Java: https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/connection-lambda.

Python Lambda에서 Amazon Keyspaces를 배포하고 사용하는 방법을 보여주는 또 다른 Lambda 예제는 다음 리포지토리에서 제공됩니다.

https://github.com/aws-samples/aws-keyspaces-lambda-python

Spring과 함께 Amazon Keyspaces(Apache Cassandra용) 사용

다음은 Spring Boot와 함께 Amazon Keyspaces를 사용하는 방법을 보여주는 예제입니다.

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/java/datastax-v4/spring

Scala와 함께 Amazon Keyspaces(Apache Cassandra용) 사용

다음은 Scala에서 SigV4 인증 플러그인을 사용하여 Amazon Keyspaces에 접속하는 방법을 보여주는 예제입니다.

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/scala/datastax-v4/connection-sigv4

AWS Glue과 함께 Amazon Keyspaces(Apache Cassandra용) 사용

다음은 AWS Glue와 함께 Amazon Keyspaces를 사용하는 방법을 보여주는 예제입니다.

https://github.com/aws-samples/amazon-keyspaces-examples/tree/main/scala/datastax-v4/aws-glue

Amazon Keyspaces(Apache Cassandra용) Cassandra 쿼리 언어(CQL) - AWS CloudFormation 변환기

이 패키지는 Apache Cassandra 쿼리 언어(CQL) 스크립트를 AWS CloudFormation(CloudFormation) 템플릿으로 변환하기 위한 명령줄 도구를 구현하며, 이를 통해 CloudFormation 스택에서 Amazon Keyspaces 스키마를 쉽게 관리할 수 있습니다.

https://github.com/aws/amazon-keyspaces-cql-to-cfn-converter.

Java용 Apache Cassandra를 위한 Amazon Keyspaces(Apache Cassandra용) 도우미

이 리포지토리에는 Amazon Keyspaces(Apache Cassandra용)와 함께 DataStax Java 드라이버를 사용할 때의 드라이버 정책, 예제, 모범 사례가 포함되어 있습니다.

https://github.com/aws-samples/amazon-keyspaces-java-driver-helpers.

Amazon Keyspaces(Apache Cassandra용) 빠른 컴프레션 데모

이 리포지토리는 더 빠른 성능과 낮은 처리량 및 스토리지 비용을 위해 대형 객체를 압축, 저장, 읽기/쓰기를 실행하는 방법을 보여줍니다.

https://github.com/aws-samples/amazon-keyspaces-compression-example.

Amazon Keyspaces(Apache Cassandra용) 및 Amazon S3 코덱 데모

사용자 지정 Amazon S3 코덱은 UUID 포인터를 Amazon S3 객체에 투명하게 사용자 구성에 따라 매핑하는 기능을 지원합니다.

https://github.com/aws-samples/amazon-keyspaces-large-object-s3-demo.

Amazon Keyspaces와 Apache Spark의 통합

Apache Spark는 대규모 데이터 분석을 위한 오픈 소스 엔진입니다. Apache Spark를 사용하면 Amazon Keyspaces에 저장된 데이터에 대한 분석을 보다 효율적으로 수행할 수 있습니다. 또한 Amazon Keyspaces를 사용하여 Spark의 분석 데이터에 대한 일관된 한 자리 밀리초 단위 읽기 액세스 권한을 애플리케이션에 제공할 수 있습니다. 오픈 소스 Spark Cassandra 커넥터는 Amazon Keyspaces와 Spark 간의 데이터 읽기 및 쓰기를 간소화합니다.

Spark Cassandra 커넥터에 대한 Amazon Keyspaces의 지원은 완전관리형 서버리스 데이터베이스 서비스를 사용하여 Spark 기반 분석 파이프라인에서 Cassandra 워크로드 실행을 간소화합니다. Amazon Keyspaces를 사용하면 Spark가 테이블과 동일한 기본 인프라 리소스를 놓고 경쟁하는 것에 대해 걱정할 필요가 없습니다. Amazon Keyspaces 테이블은 애플리케이션 트래픽에 따라 자동으로 확장 및 축소됩니다.

다음 자습서는 Spark Cassandra 커넥터를 사용하여 Amazon Keyspaces에 데이터를 읽고 쓰는 데 필요한 단계와 모범 사례를 안내합니다. 이 자습서에서는 Spark Cassandra 커넥터를 사용하여 파일에서 데이터를 로드하고 Amazon Keyspaces 테이블에 데이터를 기록하여 Amazon Keyspaces로 데이터를 마이그레이션하는 방법을 보여 줍니다. 그런 다음 자습서에서는 Spark Cassandra 커넥터를 사용하여 Amazon Keyspaces에서 데이터를 다시 읽는 방법을 보여 줍니다. 이렇게 하면 Spark 기반 분석 파이프라인에서 Cassandra 워크로드를 실행할 수 있습니다.

주제

Spark Cassandra 커넥터를 사용하여 Amazon Keyspaces에 연결하기 위한 사전 요구 사항

Spark Cassandra 커넥터를 사용하여 Amazon Keyspaces에 연결하기 전에 다음을 설치했는지 확인해야 합니다. Amazon Keyspaces와 Spark Cassandra 커넥터의 호환성은 다음 권장 버전을 통해 테스트되었습니다.

- Java 버전 8
- Scala 2.12
- Spark 3.4
- Cassandra 커넥터 2.5 이상
- Cassandra 드라이버 4.12

1. Scala를 설치하려면 https://www.scala-lang.org/download/scala2.html에서 지침을 따릅니다.
2. Spark 3.4.1을 설치하려면 이 예제를 따릅니다.

```
curl -o spark-3.4.1-bin-hadoop3.tgz -k https://dlcdn.apache.org/spark/spark-3.4.1/
spark-3.4.1-bin-hadoop3.tgz

# now to untar
tar -zxvf spark-3.4.1-bin-hadoop3.tgz

# set this variable.
export SPARK_HOME=$PWD/spark-3.4.1-bin-hadoop3
```

1단계: Apache Cassandra Spark 커넥터와의 통합을 위한 Amazon Keyspaces 구성

이 단계에서는 계정의 파티셔너가 Apache Spark 커넥터와 호환되는지 확인하고 필요한 IAM 권한을 설정합니다. 다음 모범 사례는 테이블에 충분한 읽기/쓰기 용량을 프로비저닝하는 데 도움이 됩니다.

1. `Murmur3Partitioner` 파티셔너가 계정의 기본 파티셔너인지 확인합니다. 이 파티셔너는 Spark Cassandra 커넥터와 호환됩니다. 파티셔너와 파티셔너 변경 방법에 대한 자세한 내용은 the section called “파티셔너로 작업하기” 섹션을 참조하세요.

2. Apache Spark와 함께 인터페이스 VPC 엔드포인트를 사용하여 Amazon Keyspaces에 대한 IAM 권한을 설정합니다.

   - 아래 나열된 IAM 정책 예제에 나와 있는 것처럼 사용자 테이블에 읽기/쓰기 권한과 시스템 테이블에 대한 읽기 권한을 할당합니다.
   - VPC 엔드포인트를 통해 Spark로 Amazon Keyspaces에 액세스하는 클라이언트의 경우 system.peers 테이블을 사용 가능한 인터페이스 VPC 엔드포인트로 채워야 합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select",
            "cassandra:Modify"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      },
      {
         "Sid":"ListVPCEndpoints",
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcEndpoints"
         ],
         "Resource":"*"
      }
   ]
}
```

3. Amazon Keyspaces 테이블에 Spark Cassandra 커넥터의 트래픽을 지원할 수 있는 충분한 읽기/쓰기 처리량 용량을 구성하려면 다음 모범 사례를 고려합니다.

   - 시나리오를 테스트하는 데 도움이 되도록 온디맨드 용량 사용을 시작합니다.

- 프로덕션 환경의 테이블 처리량 비용을 최적화하려면 커넥터의 트래픽에 속도 제한기를 사용하고 자동 크기 조정을 통해 프로비저닝된 용량을 사용하도록 테이블을 구성합니다. 자세한 내용은 the section called “Auto Scaling을 통한 처리 용량 관리” 섹션을 참조하세요.
- Cassandra 드라이버와 함께 제공되는 고정 속도 제한기를 사용할 수 있습니다. AWS 샘플 리포지토리에 Amazon Keyspaces에 맞게 조정된 몇 가지 속도 제한기가 있습니다.
- 용량 관리에 대한 자세한 내용은 the section called “읽기/쓰기 용량 모드” 섹션을 참조하세요.

2단계: Apache Cassandra Spark 커넥터 구성

Apache Spark는 다양한 방식으로 구성할 수 있는 범용 컴퓨팅 플랫폼입니다. Amazon Keyspaces와 통합되도록 Spark 및 Spark Cassandra 커넥터를 구성하려면 다음 섹션에 설명된 최소 구성 설정으로 시작한 다음 나중에 워크로드에 맞게 크기를 늘리는 것이 좋습니다.

- Spark 파티션 크기를 8MB보다 작게 생성합니다.

  Spark에서 파티션은 병렬로 실행할 수 있는 원자 데이터 청크를 나타냅니다. Spark Cassandra 커넥터를 사용하여 Amazon Keyspaces에 데이터를 쓰는 경우 Spark 파티션이 작을수록 작업에서 기록할 레코드 양도 줄어듭니다. Spark 작업에서 오류가 여러 번 발생하는 경우 지정된 재시도 횟수를 모두 사용한 후에는 작업이 실패합니다. 큰 작업을 재생하고 많은 데이터를 재처리하지 않으려면 Spark 파티션의 크기를 작게 유지합니다.
- 재시도 횟수가 많고 실행자당 동시 쓰기 횟수를 적게 사용합니다.

  Amazon Keyspaces는 작업 제한 시간이 초과되면 용량 부족 오류를 Cassandra 드라이버에 반환합니다. Spark Cassandra 커넥터가 `MultipleRetryPolicy`를 사용하여 요청을 투명하게 재시도하기 때문에 구성된 제한 시간 기간을 변경하여 용량 부족으로 인한 시간 초과를 해결할 수 없습니다. 재시도 횟수가 드라이버의 연결 풀에 과부하를 주지 않도록 하려면 실행자당 동시 쓰기 횟수를 줄이고 재시도 횟수가 많아야 합니다. 다음 코드 조각은 이에 대한 예입니다.

```
spark.cassandra.query.retry.count = 500
spark.cassandra.output.concurrent.writes = 3
```

- 총 처리량을 세분화하여 여러 Cassandra 세션에 분배합니다.
  - Cassandra Spark 커넥터는 각 Spark 실행기에 대해 하나의 세션을 생성합니다. 이 세션을 필요한 처리량과 필요한 연결 수를 결정하는 규모의 단위라고 생각하면 됩니다.
  - 실행자당 코어 수와 작업당 코어 수를 정의할 때는 낮은 수준에서 시작하여 필요에 따라 늘립니다.

- 일시적 오류가 발생하는 경우에도 처리할 수 있도록 Spark 작업 실패를 설정합니다. 애플리케이션의 트래픽 특성 및 요구 사항을 숙지한 후에는 `spark.task.maxFailures`를 제한된 값으로 설정하는 것이 좋습니다.
- 예를 들어 다음 구성에서는 실행자당, 세션별로 두 개의 동시 작업을 처리할 수 있습니다.

```
spark.executor.instances = configurable -> number of executors for the session.
spark.executor.cores = 2 -> Number of cores per executor.
spark.task.cpus = 1 -> Number of cores per task.
spark.task.maxFailures = -1
```

- 일괄 처리를 끕니다.
  - 임의 액세스 패턴을 개선하려면 일괄 처리를 해제하는 것이 좋습니다. 다음 코드 조각은 이에 대한 예입니다.

```
spark.cassandra.output.batch.size.rows = 1 (Default = None)
spark.cassandra.output.batch.grouping.key = none (Default = Partition)
spark.cassandra.output.batch.grouping.buffer.size = 100 (Default = 1000)
```

- **`SPARK_LOCAL_DIRS`**를 충분한 공간이 있는 빠른 로컬 디스크로 설정합니다.
  - 기본적으로 Spark는 맵 출력 파일과 복원력이 있는 분산 데이터 세트(RDD)를 `/tmp` 폴더에 저장합니다. Spark 호스트의 구성에 따라 이로 인해 디바이스에 남은 공간 없음 스타일 오류가 발생할 수 있습니다.
  - `SPARK_LOCAL_DIRS` 환경 변수를 `/example/spark-dir`이라는 디렉터리로 설정하려면 다음 명령을 사용할 수 있습니다.

```
export SPARK_LOCAL_DIRS=/example/spark-dir
```

3단계: 애플리케이션 구성 파일 생성

Amazon Keyspaces와 함께 오픈 소스 Spark Cassandra 커넥터를 사용하려면 DataStax Java 드라이버에 연결하는 데 필요한 설정이 포함된 애플리케이션 구성 파일을 제공해야 합니다. 서비스별 보안 인증 또는 SigV4 플러그인을 사용하여 연결할 수 있습니다.

아직 Starfield 디지털 인증서를 trustStore 파일로 변환하지 않았다면 이를 수행합니다. Java 드라이버 연결 자습서의 the section called “시작하기 전 준비 사항”에서 세부 단계를 따를 수 있습니다. trustStore 파일 경로와 암호는 애플리케이션 구성 파일을 생성할 때 필요하므로 기록해 둡니다.

SigV4 인증을 사용하여 연결

이 섹션에서는 AWS 보안 인증과 SigV4 플러그인을 사용하여 연결할 때 사용할 수 있는 예제 `application.conf` 파일을 보여 줍니다. 아직 생성하지 않았다면 IAM 액세스 키(액세스 키 ID 및 비밀 액세스 키)를 생성하여 AWS 구성 파일에 저장하거나 환경 변수로 저장해야 합니다. 자세한 지침은 the section called “인증을 위한 필수 자격 증명 AWS ” 섹션을 참조하세요.

다음 예제에서는 trustStore 파일의 파일 경로를 바꾸고 암호를 바꿉니다.

```
datastax-java-driver {
        basic.contact-points = ["cassandra.us-east-1.amazonaws.com:9142"]
        basic.load-balancing-policy {
            class = DefaultLoadBalancingPolicy
            local-datacenter = us-east-1
            slow-replica-avoidance = false
        }
        basic.request {
              consistency = LOCAL_QUORUM
        }
        advanced {
                auth-provider = {
                   class = software.aws.mcs.auth.SigV4AuthProvider
                   aws-region = us-east-1
                 }
            ssl-engine-factory {
                class = DefaultSslEngineFactory
                truststore-path = "path_to_file/cassandra_truststore.jks"
                truststore-password = "password"
        hostname-validation=false
            }
   }
        advanced.connection.pool.local.size = 3
}
```

이 구성 파일을 `/home/user1/application.conf`로 업데이트하고 저장합니다. 다음 예제는 이 경로를 사용합니다.

서비스별 보안 인증으로 연결

이 섹션에서는 서비스별 보안 인증을 사용하여 연결할 때 사용할 수 있는 예제 `application.conf` 파일을 보여 줍니다. Amazon Keyspaces에 대해 서비스별 보안 인증을 생성하지 않았다면 이를 생성해야 합니다. 자세한 지침은 the section called “서비스별 보안 인증 정보” 섹션을 참조하세요.

다음 예제에서 username 및 password를 고유한 보안 인증으로 바꿉니다. 또한 trustStore 파일의 파일 경로를 바꾸고 암호를 바꿉니다.

```
datastax-java-driver {
        basic.contact-points = ["cassandra.us-east-1.amazonaws.com:9142"]
        basic.load-balancing-policy {
            class = DefaultLoadBalancingPolicy
            local-datacenter = us-east-1
        }
        basic.request {
              consistency = LOCAL_QUORUM
        }
        advanced {
            auth-provider = {
            class = PlainTextAuthProvider
                    username = "username"
                    password = "password"
                    aws-region = "us-east-1"
            }
            ssl-engine-factory {
                class = DefaultSslEngineFactory
                truststore-path = "path_to_file/cassandra_truststore.jks"
                truststore-password = "password"
                hostname-validation=false
            }
            metadata = {
                schema {
                     token-map.enabled = true
                }
            }
        }
}
```

코드 예제와 함께 사용할 수 있도록 이 구성 파일을 /home/user1/application.conf로 업데이트하고 저장합니다.

고정 속도로 연결

Spark 실행자당 고정 속도를 적용하려면 요청 조절기를 정의하면 됩니다. 이 요청 조절기는 초당 요청 속도를 제한합니다. Spark Cassandra 커넥터는 실행자당 Cassandra 세션을 배포합니다. 다음 공식을 사용하면 테이블에 대해 일관된 처리량을 달성하는 데 도움이 될 수 있습니다.

```
max-request-per-second * numberOfExecutors = total throughput against a table
```

이 예제를 이전에 만든 애플리케이션 구성 파일에 추가할 수 있습니다.

```
datastax-java-driver {
  advanced.throttler {
    class = RateLimitingRequestThrottler

    max-requests-per-second = 3000
    max-queue-size = 30000
    drain-interval = 1 millisecond
  }
}
```

4단계: Amazon Keyspaces에서 원본 데이터 및 대상 테이블 준비

이 단계에서는 샘플 데이터와 Amazon Keyspaces 테이블을 사용하여 소스 파일을 생성합니다.

1. 소스 파일을 생성합니다. 다음 옵션 중 하나를 선택할 수 있습니다.
   - 이 자습서에서는 이름 `keyspaces_sample_table.csv`가 있는 쉼표로 구분된 값(CSV) 파일을 데이터 마이그레이션의 원본 파일로 사용합니다. 제공된 샘플 파일에는 이름이 `book_awards`인 테이블에 대한 몇 개의 데이터 행이 포함되어 있습니다.
     - 다음 아카이브 파일 samplemigration.zip에 포함된 샘플 CSV 파일(`keyspaces_sample_table.csv`)을 다운로드합니다. 아카이브의 압축을 풀고 `keyspaces_sample_table.csv`의 경로를 기록해 둡니다.
   - 자체 CSV 파일을 따라 Amazon Keyspaces에 데이터를 쓰려면 데이터가 무작위로 지정되었는지 확인하십시오. 데이터베이스에서 직접 읽거나 플랫 파일로 내보내는 데이터는 일반적으로 파티션과 프라이머리 키를 기준으로 정렬됩니다. 정렬된 데이터를 Amazon Keyspaces로 가져오면 Amazon Keyspaces 파티션의 더 작은 세그먼트에 데이터가 기록되어 트래픽이 고르지 않게 분산될 수 있습니다. 이로 인해 성능이 저하되고 오류율이 높아질 수 있습니다.

     반대로 데이터를 무작위화하면 트래픽을 파티션 전체에 더 균등하게 분산하여 Amazon Keyspaces의 기본 제공 로드 밸런싱 기능을 활용할 수 있습니다. 데이터를 랜덤화하는 데 사용할 수 있는 다양한 도구가 있습니다. 오픈소스 도구인 Shuf를 사용하는 예제는 데이터 마이그레이션 자습서에서 the section called “2단계: 데이터 준비” 섹션을 참조하세요. 다음은 데이터를 `DataFrame`으로 셔플하는 방법을 보여 주는 예제입니다.

```
import org.apache.spark.sql.functions.randval
shuffledDF = dataframe.orderBy(rand())
```

2. Amazon Keyspaces에서 대상 키스페이스 및 테이블을 생성합니다.

   a. cqlsh를 사용하여 Amazon Keyspaces에 연결하고 다음 예제의 서비스 엔드포인트, 사용자 이름 및 암호를 사용자 고유의 값으로 바꿉니다.

   ```
   cqlsh cassandra.us-east-2.amazonaws.com 9142 -u "111122223333" -
   p "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY" --ssl
   ```

   b. 다음 예제와 같이 이름 catalog를 사용하여 새 키스페이스를 생성합니다.

   ```
   CREATE KEYSPACE catalog WITH REPLICATION = {'class': 'SingleRegionStrategy'};
   ```

   c. 새 키스페이스가 사용 가능 상태가 되면 다음 코드를 사용하여 대상 테이블 book_awards를 생성합니다. 비동기 리소스 생성 및 리소스가 사용 가능한지 확인하는 방법에 대한 자세한 내용은 the section called “키스페이스 생성” 섹션을 참조하세요.

   ```
   CREATE TABLE catalog.book_awards (
      year int,
      award text,
      rank int,
      category text,
      book_title text,
      author text,
      publisher text,
      PRIMARY KEY ((year, award), category, rank)
      );
   ```

5단계: Apache Cassandra Spark 커넥터를 사용하여 Amazon Keyspaces 데이터 쓰기 및 읽기

이 단계에서는 먼저 샘플 파일의 데이터를 Spark Cassandra 커넥터를 사용하여 DataFrame으로 로드합니다. 다음으로 DataFrame의 데이터를 Amazon Keyspaces 테이블에 작성합니다. 예를 들어 이 부분을 독립적으로 사용하여 데이터를 Amazon Keyspaces 테이블로 마이그레이션할 수도 있습니다. 마지막으로 Spark Cassandra 커넥터를 사용하여 테이블의 데이터를 DataFrame으로 읽습니다. 예를 들

어 이 부분을 독립적으로 사용하여 Amazon Keyspaces 테이블에서 데이터를 읽고 Apache Spark로 데이터 분석을 수행할 수도 있습니다.

1. 다음 예제와 같이 Spark Shell을 시작합니다. 이 예제에서는 SigV4 인증을 사용하고 있습니다.

```
./spark-shell --files application.conf --conf
 spark.cassandra.connection.config.profile.path=application.conf
 --packages software.aws.mcs:aws-sigv4-auth-cassandra-java-driver-
plugin:4.0.5,com.datastax.spark:spark-cassandra-connector_2.12:3.1.0 --conf
 spark.sql.extensions=com.datastax.spark.connector.CassandraSparkExtensions
```

2. 다음 코드를 사용하여 Spark Cassandra 커넥터를 가져옵니다.

```
import org.apache.spark.sql.cassandra._
```

3. CSV 파일에서 데이터를 읽고 DataFrame에 저장하려면 다음 코드 예제를 사용할 수 있습니다.

```
var df =
 spark.read.option("header","true").option("inferSchema","true").csv("keyspaces_sample_tabl
```

다음 명령을 사용하여 결과를 표시할 수 있습니다.

```
scala> df.show();
```

결과는 다음과 비슷해야 합니다.

```
+----------------+----+-----------+----+-----------------+--------------------
+-------------+
|           award|year|   category|rank|           author|          book_title|
 publisher|
+----------------+----+-----------+----+-----------------+--------------------
+-------------+
|Kwesi Manu Prize|2020|    Fiction|   1|        Akua Mansa|   Where did you go?|
SomePublisher|
|Kwesi Manu Prize|2020|    Fiction|   2|       John Stiles|           Yesterday|
Example Books|
|Kwesi Manu Prize|2020|    Fiction|   3|        Nikki Wolf|Moving to the Cha...|
 AnyPublisher|
|            Wolf|2020|Non-Fiction|   1|       Wang Xiulan|    History of Ideas|
Example Books|
```

```
|            Wolf|2020|Non-Fiction|   2|Ana Carolina Silva|       Science Today|
SomePublisher|
|            Wolf|2020|Non-Fiction|   3| Shirley Rodriguez|The Future of Sea...|
 AnyPublisher|
|     Richard Roe|2020|    Fiction|   1| Alejandro Rosalez|         Long Summer|
SomePublisher|
|     Richard Roe|2020|    Fiction|   2|       Arnav Desai|             The Key|
Example Books|
|     Richard Roe|2020|    Fiction|   3|     Mateo Jackson|    Inside the Whale|
 AnyPublisher|
+----------------+----+-----------+----+------------------+--------------------
+-------------+
```

다음 예제와 같이 DataFrame에서 데이터의 스키마를 확인할 수 있습니다.

```
scala> df.printSchema
```

결과는 다음과 같아야 합니다.

```
root
|-- award: string (nullable = true)
|-- year: integer (nullable = true)
|-- category: string (nullable = true)
|-- rank: integer (nullable = true)
|-- author: string (nullable = true)
|-- book_title: string (nullable = true)
|-- publisher: string (nullable = true)
```

4. 다음 명령을 사용하여 Amazon Keyspaces 테이블에 DataFrame의 데이터를 씁니다.

```
df.write.cassandraFormat("book_awards", "catalog").mode("APPEND").save()
```

5. 데이터가 저장되었는지 확인하려면 다음 예제와 같이 데이터를 데이터 프레임으로 다시 읽을 수 있습니다.

```
var newDf = spark.read.cassandraFormat("book_awards", "catalog").load()
```

그러면 현재 데이터 프레임에 포함된 데이터를 표시할 수 있습니다.

```
scala> newDf.show()
```

해당 명령의 결과는 다음과 같아야 합니다.

```
+--------------------+-----------------+---------------+----------+-------------
+----+----+
|          book_title|           author|          award|  category|
 publisher|rank|year|
+--------------------+-----------------+---------------+----------+-------------
+----+----+
|         Long Summer| Alejandro Rosalez|     Richard Roe|    Fiction|
SomePublisher|   1|2020|
|    History of Ideas|       Wang Xiulan|           Wolf|Non-Fiction|Example
 Books|   1|2020|
|   Where did you go?|        Akua Mansa|Kwesi Manu Prize|    Fiction|
SomePublisher|   1|2020|
|    Inside the Whale|     Mateo Jackson|     Richard Roe|    Fiction|
 AnyPublisher|   3|2020|
|           Yesterday|       John Stiles|Kwesi Manu Prize|    Fiction|Example
 Books|   2|2020|
|Moving to the Cha...|        Nikki Wolf|Kwesi Manu Prize|    Fiction|
 AnyPublisher|   3|2020|
|The Future of Sea...| Shirley Rodriguez|           Wolf|Non-Fiction|
 AnyPublisher|   3|2020|
|       Science Today|Ana Carolina Silva|           Wolf|Non-Fiction|
SomePublisher|   2|2020|
|             The Key|       Arnav Desai|     Richard Roe|    Fiction|Example
 Books|   2|2020|
+--------------------+-----------------+---------------+----------+-------------
+----+----+
```

Amazon Keyspaces와 함께 Spark Cassandra 커넥터를 사용할 때 발생하는 일반적인 오류 문제 해결

Amazon Virtual Private Cloud를 사용하고 Amazon Keyspaces에 연결하는 경우 Spark 커넥터를 사용할 때 발생하는 가장 일반적인 오류는 다음과 같은 구성 문제로 인해 발생합니다.

- VPC에서 사용되는 IAM 사용자 또는 역할에는 Amazon Keyspaces의 `system.peers` 테이블에 액세스하는 데 필요한 권한이 없습니다. 자세한 내용은 the section called “인터페이스 VPC 엔드포인트 정보로 system.peers 테이블 항목 채우기” 섹션을 참조하세요.

- IAM 사용자 또는 역할에는 사용자 테이블에 대한 필수 읽기/쓰기 권한과 Amazon Keyspaces의 시스템 테이블에 대한 읽기 액세스 권한이 없습니다. 자세한 내용은 the section called “1단계: Amazon Keyspaces 구성” 섹션을 참조하세요.
- Java 드라이버 구성은 SSL/TLS 연결을 생성할 때 호스트 이름 확인을 비활성화하지 않습니다. 예제는 the section called “2단계: 드라이버 구성” 섹션을 참조하세요.

자세한 연결 문제 해결 단계는 the section called “VPC 엔드포인트 연결 오류” 섹션을 참조하세요.

또한 Amazon CloudWatch 지표를 사용하면 Amazon Keyspaces의 Spark Cassandra 커넥터 구성과 관련된 문제를 해결하는 데 도움이 될 수 있습니다. CloudWatch와 함께 Amazon Keyspaces를 사용하는 방법에 대한 자세한 내용은 the section called “를 통한 모니터링 CloudWatch” 섹션을 참조하세요.

다음 섹션에서는 Spark Cassandra 커넥터를 사용할 때 관찰할 수 있는 가장 유용한 지표를 설명합니다.

PerConnectionRequestRateExceeded

Amazon Keyspaces에는 연결당 초당 3,000개의 요청 할당량이 있습니다. 각 Spark 실행자는 Amazon Keyspaces와 연결을 설정합니다. 재시도를 여러 번 실행하면 연결당 요청 속도 할당량이 소진될 수 있습니다. 이 할당량을 초과하면 Amazon Keyspaces는 CloudWatch에서 `PerConnectionRequestRateExceeded` 지표를 내보냅니다.

다른 시스템 또는 사용자 오류와 함께 PerConnectionRequestRateExceeded 이벤트가 나타나는 경우 Spark에서 연결당 할당된 요청 수를 초과하여 여러 번 재시도를 실행하고 있는 것일 수 있습니다.

다른 오류가 없는 `PerConnectionRequestRateExceeded` 이벤트가 표시되면 처리량을 늘리기 위해 드라이버 설정에서 연결 수를 늘리거나 Spark 작업의 실행자 수를 늘려야 할 수 있습니다.

StoragePartitionThroughputCapacityExceeded

Amazon Keyspaces의 할당량은 파티션당 초당 1,000개의 WCU 또는 WRU/초당 3,000개의 RCU 또는 RRU입니다. `StoragePartitionThroughputCapacityExceeded` CloudWatch 이벤트가 표시되는 경우 로드 시 데이터가 무작위화되지 않았음을 의미할 수 있습니다. 데이터 셔플 방법에 대한 예는 the section called “4단계: 원본 데이터 및 대상 테이블 준비” 섹션을 참조하세요.

일반적인 오류 및 경고

Amazon Virtual Private Cloud를 사용하고 있고 Amazon Keyspaces에 연결하는 경우 Cassandra 드라이버는 `system.peers` 테이블의 제어 노드 자체에 대한 경고 메시지를 발행할 수 있습니다. 자세한 내용은 the section called “일반적인 오류 및 경고” 섹션을 참조하세요. 이 경고는 무시해도 됩니다.

Amazon Keyspaces(Apache Cassandra용) 문제 해결

다음 섹션에서는 Amazon Keyspaces(Apache Cassandra용) 사용 시 발생할 수 있는 일반적인 구성 문제를 해결하는 방법에 대한 정보를 제공합니다.

IAM 액세스 관련 문제 해결 지침은 the section called “문제 해결”을 참조하세요.

보안 모범 사례에 대한 자세한 내용은 the section called “보안 모범 사례” 섹션을 참조하세요.

주제

Amazon Keyspace의 일반 오류 문제 해결

일반적인 오류가 발생하나요? 다음에서는 몇 가지 일반적인 문제와 이에 대한 해결 방법에 대해 설명합니다.

일반 오류

여러 가지 이유로 인해 발생할 수 있는 다음과 같은 최상위 예외 중 하나가 발생합니다.

- `NoNodeAvailableException`
- `NoHostAvailableException`
- `AllNodesFailedException`

이러한 예외는 클라이언트 드라이버에 의해 생성되며 제어 연결을 설정하거나 읽기/쓰기/준비/실행/배치 요청을 수행할 때 발생할 수 있습니다.

제어 연결을 설정하는 동안 오류가 발생하면 애플리케이션에 지정된 모든 연락처에 연결할 수 없다는 신호입니다. 쿼리 읽기/쓰기/준비/실행을 수행하는 동안 오류가 발생하면 해당 요청에 대한 재시도 횟수가 모두 소진되었음을 나타냅니다. 기본 재시도 정책을 사용하는 경우 각 재시도는 다른 노드에서 시도됩니다.

최상위 Java 드라이버 예외에서 기본 오류를 분리하는 방법

이러한 일반 오류는 연결 문제 또는 읽기/쓰기/준비/실행 작업을 수행할 때 발생할 수 있습니다. 분산 시스템에서는 일시적 장애가 발생할 것으로 예상되므로 요청을 다시 시도하여 처리해야 합니다. Java 드라이버는 연결 오류가 발생해도 자동으로 재시도하지 않으므로 응용 프로그램에서 드라이버 연결을 설정할 때 재시도 정책을 구현하는 것이 좋습니다. 연결 모범 사례에 대한 자세한 개요는 를 참조하십시오. the section called “연결”

기본적으로 Java 드라이버는 모든 요청에 대해 `idempotence` false로 설정됩니다. 즉, Java 드라이버는 실패한 읽기/쓰기/준비 요청을 자동으로 재시도하지 않습니다. 드라이버에 실패한 요청을 `idempotence` 재시도하도록 `true` 설정하고 드라이버에 지시하려면 몇 가지 다른 방법을 사용할 수 있습니다. 다음은 Java 애플리케이션에서 단일 요청에 대해 프로그래밍 방식으로 idempotence를 설정하는 방법의 한 가지 예입니다.

```
Statement s = new SimpleStatement("SELECT * FROM my_table WHERE id = 1");
s.setIdempotent(true);
```

또는 다음 예제와 같이 전체 Java 응용 프로그램의 기본 항등성을 프로그래밍 방식으로 설정할 수 있습니다.

```
// Make all statements idempotent by default:
cluster.getConfiguration().getQueryOptions().setDefaultIdempotence(true);
//Set the default idempotency to true in your Cassandra configuration
basic.request.default-idempotence = true
```

또 다른 권장 사항은 애플리케이션 수준에서 재시도 정책을 생성하는 것입니다. 이 경우 애플리케이션은 요청을 `NoNodeAvailableException` 캐치하여 재시도해야 합니다. 지수 백오프는 10ms에서 시작하여 최대 100ms까지 작업하고 모든 재시도에 대해 총 1초로 10회 재시도하는 것이 좋습니다.

또 다른 옵션은 Github에서 사용할 수 있는 Java 드라이버 연결을 설정할 때 Amazon Keyspaces 지수 재시도 정책을 적용하는 것입니다.

기본 재시도 정책을 사용할 때 둘 이상의 노드에 대한 연결을 설정했는지 확인하십시오. Amazon Keyspace에서 다음 쿼리를 사용하여 이 작업을 수행할 수 있습니다.

```
SELECT * FROM system.peers;
```

이 쿼리에 대한 응답이 비어 있으면 Amazon Keyspaces에서 단일 노드를 사용하고 있음을 나타냅니다. 기본 재시도 정책을 사용하는 경우 기본 재시도는 항상 다른 노드에서 이루어지므로 재시도는 없

습니다. VPC 엔드포인트를 통한 연결 설정에 대한 자세한 내용은 을 참조하십시오. the section called "VPC 엔드포인트 연결"

Datastax 4.x Cassandra 드라이버를 사용하여 Amazon Keyspaces에 연결하는 방법을 보여주는 step-by-step 자습서는 을 참조하십시오. the section called "Java 4.x용 인증 플러그인"

Amazon Keyspaces의 연결 문제 해결

연결에 문제가 있으신가요? 다음에서는 몇 가지 일반적인 문제와 이에 대한 해결 방법에 대해 설명합니다.

Amazon Keyspaces 엔드포인트에 연결하는 오류

연결 실패 및 연결 오류로 인해 다른 오류 메시지가 표시될 수 있습니다. 다음 섹션에서는 가장 일반적인 시나리오를 다룹니다.

주제

- cqlsh를 사용하여 Amazon Keyspaces에 연결할 수 없음
- Cassandra 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 연결할 수 없음

cqlsh를 사용하여 Amazon Keyspaces에 연결할 수 없음

cqlsh를 사용하여 Amazon Keyspaces 엔드포인트에 연결하려고 하는데 **`Connection error`**와 함께 연결이 실패합니다.

Amazon Keyspaces 테이블에 연결하려고 하는데 cqlsh가 제대로 구성되지 않은 경우 연결이 실패합니다. 다음 섹션에서는 cqlsh를 사용하여 연결을 설정하려고 할 때 연결 오류가 발생하는 가장 일반적인 구성 문제의 예를 제공합니다.

> **Note**
>
> VPC에서 Amazon Keyspaces에 연결하려는 경우 추가 권한이 필요합니다. VPC 엔드포인트를 사용하여 연결을 성공적으로 구성하려면 the section called "VPC 엔드포인트와 연결"의 단계를 따르세요.

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하는데 연결 **`timed out`** 오류가 발생합니다.

올바른 포트를 제공하지 않은 경우 다음과 같은 오류가 발생할 수 있습니다.

```
#  cqlsh cassandra.us-east-1.amazonaws.com 9140 -u "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.199': error(None,
 "Tried connecting to [('3.234.248.199', 9140)]. Last error: timed out")})
```

이 문제를 해결하려면 연결에 포트 9142를 사용하고 있는지 확인합니다.

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하는데 **Name or service not known** 오류가 발생합니다.

철자가 틀렸거나 존재하지 않는 엔드포인트를 사용한 경우가 이에 해당할 수 있습니다. 다음 예에서는 엔드포인트 이름의 철자가 틀립니다.

```
#  cqlsh cassandra.us-east-1.amazon.com 9142 -u "USERNAME" -p "PASSWORD" --ssl
Traceback (most recent call last):
  File "/usr/bin/cqlsh.py", line 2458, in >module>
    main(*read_options(sys.argv[1:], os.environ))
  File "/usr/bin/cqlsh.py", line 2436, in main
    encoding=options.encoding)
  File "/usr/bin/cqlsh.py", line 484, in __init__
    load_balancing_policy=WhiteListRoundRobinPolicy([self.hostname]),
  File "/usr/share/cassandra/lib/cassandra-driver-internal-only-3.11.0-bb96859b.zip/
cassandra-driver-3.11.0-bb96859b/cassandra/policies.py", line 417, in __init__
socket.gaierror: [Errno -2] Name or service not known
```

퍼블릭 엔드포인트를 사용하여 연결할 때 이 문제를 해결하려면 the section called “서비스 엔드포인트”에서 사용 가능한 엔드포인트를 선택하고 엔드포인트 이름에 오류가 없는지 확인합니다. VPC 엔드포인트를 사용하여 연결하는 경우 cqlsh 구성에서 VPC 엔드포인트 정보가 올바른지 확인합니다.

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하지만 **OperationTimedOut** 오류를 수신합니다.

Amazon Keyspaces에서는 강력한 보안을 보장하기 위해 연결에 SSL을 활성화해야 합니다. 다음 오류를 받는 경우 SSL 매개 변수가 누락될 수 있습니다.

```
# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD"
Connection error: ('Unable to connect to any servers', {'3.234.248.192':
 OperationTimedOut('errors=Timed out creating connection (5 seconds),
 last_host=None',)})
#
```

이 문제를 해결하려면 cqlsh 연결 명령에 다음 플래그를 추가합니다.

```
--ssl
```

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하는데 **SSL transport factory requires a valid certfile to be specified** 오류를 수신합니다.

이 경우 SSL/TLS 인증서 경로가 누락되어 다음 오류가 발생합니다.

```
# cat .cassandra/cqlshrc
[connection]
port = 9142
factory = cqlshlib.ssl.ssl_transport_factory
#


# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD" --ssl
Validation is enabled; SSL transport factory requires a valid certfile to be specified.
 Please provide path to the certfile in [ssl] section as 'certfile' option in /
root/.cassandra/cqlshrc (or use [certfiles] section) or set SSL_CERTFILE environment
 variable.
#
```

이 문제를 해결하려면 컴퓨터에 있는 certfile에 경로를 추가합니다.

```
certfile =  path_to_file/sf-class2-root.crt
```

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하지만 **No such file or directory** 오류를 수신합니다.

컴퓨터에 있는 인증서 파일의 경로가 잘못되어 다음과 같은 오류가 발생할 수 있습니다.

```
# cat .cassandra/cqlshrc
[connection]
port = 9142
factory = cqlshlib.ssl.ssl_transport_factory

[ssl]
validate = true
certfile = /root/wrong_path/sf-class2-root.crt
#
```

```
# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.192': IOError(2, 'No
 such file or directory')})
#
```

이 문제를 해결하려면 컴퓨터에 있는 certfile에 대한 경로가 올바른지 확인합니다.

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하지만 **[X509] PEM lib** 오류를 수신합니다.

SSL/TLS 인증서 파일 `sf-class2-root.crt`가 유효하지 않아 다음 오류가 발생하는 경우가 이에 해당할 수 있습니다.

```
# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.241':
 error(185090057, u"Tried connecting to [('3.234.248.241', 9142)]. Last error: [X509]
 PEM lib (_ssl.c:3063)")})
#
```

이 문제를 해결하려면 다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드합니다. `sf-class2-root.crt`를 로컬에서 또는 홈 디렉터리에 저장합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하지만 **unknown** SSL 오류를 수신합니다.

SSL/TLS 인증서 파일 `sf-class2-root.crt`가 비어 있어 다음 오류가 발생하는 경우가 이에 해당할 수 있습니다.

```
# cqlsh cassandra.us-east-1.amazonaws.com -u "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.220': error(0,
 u"Tried connecting to [('3.234.248.220', 9142)]. Last error: unknown error
 (_ssl.c:3063)")})
#
```

이 문제를 해결하려면 다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드합니다. `sf-class2-root.crt`를 로컬에서 또는 홈 디렉터리에 저장합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하지만 **SSL: CERTIFICATE_VERIFY_FAILED** 오류를 수신합니다.

SSL/TLS 인증서 파일을 확인할 수 없어 다음 오류가 발생하는 경우가 이에 해당할 수 있습니다.

```
Connection error: ('Unable to connect to any servers', {'3.234.248.223':
 error(1, u"Tried connecting to [('3.234.248.223', 9142)]. Last error: [SSL:
 CERTIFICATE_VERIFY_FAILED] certificate verify failed (_ssl.c:727)")})
```

이 문제를 해결하려면 다음 명령을 사용하여 인증서 파일을 다시 다운로드합니다. `sf-class2-root.crt`를 로컬에서 또는 홈 디렉터리에 저장합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하지만 **Last error: timed out** 오류를 수신합니다.

Amazon EC2 보안 그룹에서 Amazon Keyspaces에 대한 아웃바운드 규칙을 구성하지 않은 경우일 수 있으며 그 결과 다음 오류가 발생합니다.

```
# cqlsh cassandra.us-east-1.amazonaws.com 9142 -u  "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.206': error(None,
 "Tried connecting to [('3.234.248.206', 9142)]. Last error: timed out")})
#
```

이 문제가 Amazon EC2 인스턴스의 구성으로 인한 것인지 아닌지 확인하기 위해 예를 `cqlsh` 들어 다음 명령을 사용하여 키스페이스에 연결해 볼 수 있습니다. AWS CLI

```
aws keyspaces list-tables --keyspace-name 'my_keyspace'
```

이 명령도 제한 시간이 초과되면 Amazon EC2 인스턴스가 제대로 구성되지 않은 것입니다.

Amazon Keyspace에 액세스할 수 있는 충분한 권한이 있는지 확인하려면 를 사용하여 연결할 수 있습니다. AWS CloudShell `cqlsh` 연결이 설정되면 Amazon EC2 인스턴스를 구성해야 합니다.

이 문제를 해결하려면 Amazon EC2 인스턴스에 Amazon Keyspace로의 트래픽을 허용하는 아웃바운드 규칙이 있는지 확인하십시오. 그렇지 않은 경우 EC2 인스턴스에 대한 새 보안 그룹을 생성하고 Amazon Keyspaces 리소스로의 아웃바운드 트래픽을 허용하는 규칙을 추가해야 합니다. Amazon

Keyspace로의 트래픽을 허용하도록 아웃바운드 규칙을 업데이트하려면 유형 드롭다운 메뉴에서 CQLSH/CASSANDRA를 선택합니다.

아웃바운드 트래픽 규칙을 사용하여 새 보안 그룹을 생성한 후에는 이를 인스턴스에 추가해야 합니다. 인스턴스를 선택한 다음 작업, 보안, 보안 그룹 변경을 차례로 선택합니다. 아웃바운드 규칙을 사용하여 새 보안 그룹을 추가하되, 기본 그룹도 계속 사용할 수 있도록 해야 합니다.

EC2 아웃바운드 규칙을 보고 편집하는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 보안 그룹에 규칙 추가를 참조하십시오.

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하지만 **Unauthorized** 오류를 수신합니다.

이는 IAM 사용자 정책에서 Amazon Keyspaces 권한이 누락되어 다음 오류가 발생하는 경우일 수 있습니다.

```
# cqlsh cassandra.us-east-1.amazonaws.com 9142 -u  "testuser-at-12345678910" -p
 "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.241':
 AuthenticationFailed('Failed to authenticate to 3.234.248.241: Error from server:
 code=2100 [Unauthorized] message="User arn:aws:iam::12345678910:user/testuser has no
 permissions."',)})
#
```

이 문제를 해결하려면 IAM 사용자 `testuser-at-12345678910`에게 Amazon Keyspaces에 액세스할 권한이 있는지 확인합니다. Amazon Keyspaces에 액세스 권한을 부여하는 IAM 정책의 예는 the section called “ID 기반 정책 예제” 섹션을 참조하세요.

IAM 액세스 관련 문제 해결 지침은 the section called “문제 해결” 섹션을 참조하세요.

cqlsh를 사용하여 Amazon Keyspaces에 연결하려고 하지만 **Bad credentials** 오류를 수신합니다.

사용자 이름이나 암호가 잘못되어 다음 오류가 발생하는 경우가 이에 해당할 수 있습니다.

```
# cqlsh cassandra.us-east-1.amazonaws.com 9142 -u  "USERNAME" -p "PASSWORD" --ssl
Connection error: ('Unable to connect to any servers', {'3.234.248.248':
 AuthenticationFailed('Failed to authenticate to 3.234.248.248: Error from server:
 code=0100 [Bad credentials] message="Provided username USERNAME and/or password are
 incorrect"',)})
#
```

이 문제를 해결하려면 코드의 ### ## 및 ##가 서비스별 보안 인증을 생성할 때 얻은 사용자 이름 및 암호와 일치하는지 확인합니다.

> **Important**
>
> cqlsh로 연결하려고 할 때 오류가 계속 표시되면 `--debug` 옵션을 사용하여 명령을 다시 실행하고 AWS Support에 문의할 때 자세한 출력을 포함합니다.

Cassandra 클라이언트 드라이버를 사용하여 Amazon Keyspaces에 연결할 수 없음

다음 섹션에서는 Cassandra 클라이언트 드라이버로 연결할 때 발생하는 가장 일반적인 오류를 보여줍니다.

DataStax Java 드라이버를 사용하여 Amazon Keyspaces 테이블에 연결하려고 하는데 오류가 발생합니다. **`NodeUnavailableException`**

요청을 시도한 연결이 끊어지면 다음 오류가 발생합니다.

```
[com.datastax.oss.driver.api.core.NodeUnavailableException: No connection
 was available to Node(endPoint=vpce-22ff22f2f22222fff-aa1bb234.cassandra.us-
west-2.vpce.amazonaws.com/11.1.1111.222:9142, hostId=1a23456b-
c77d-8888-9d99-146cb22d6ef6, hashCode=123ca4567)]
```

이 문제를 해결하려면 하트비트 값을 찾아 더 높으면 30초로 낮추십시오.

```
advanced.heartbeat.interval = 30 seconds
```

그런 다음 관련 타임아웃을 찾아 값이 5초 이상으로 설정되어 있는지 확인합니다.

```
advanced.connection.init-query-timeout = 5 seconds
```

드라이버 및 SigV4 플러그인을 사용하여 Amazon Keyspaces 테이블에 연결하려고 하지만 **`AttributeError`** 오류를 수신합니다.

보안 인증이 올바르게 구성되지 않은 경우 다음 오류가 발생합니다.

```
cassandra.cluster.NoHostAvailable: ('Unable to connect to any servers',
 {'44.234.22.154:9142': AttributeError("'NoneType' object has no attribute
 'access_key'")})
```

이 문제를 해결하려면 SigV4 플러그인을 사용할 때 IAM 사용자 또는 역할과 관련된 보안 인증을 전달하고 있는지 확인합니다. SigV4 플러그인에는 다음 보안 인증이 필요합니다.

- AWS_ACCESS_KEY_ID— IAM 사용자 또는 역할과 관련된 AWS 액세스 키를 지정합니다.
- AWS_SECRET_ACCESS_KEY - 액세스 키와 연결된 보안 키를 지정합니다. 이는 액세스 키에 대한 기본적인 "암호"입니다.

액세스 키와 SigV4 플러그인에 대한 자세한 내용은 the section called “인증을 위한 IAM 자격 증명 AWS ” 섹션을 참조하세요.

드라이버를 사용하여 Amazon Keyspaces 테이블에 연결하려고 하지만 **PartialCredentialsError** 오류를 수신합니다.

AWS_SECRET_ACCESS_KEY가 누락된 경우 다음 오류가 발생할 수 있습니다.

```
cassandra.cluster.NoHostAvailable: (‘Unable to connect to any servers’,
 {‘44.234.22.153:9142’:
 PartialCredentialsError(‘Partial credentials found in config-file, missing:
 aws_secret_access_key’)})
```

이 문제를 해결하려면 SigV4 플러그인을 사용할 때 AWS_ACCESS_KEY_ID 및 AWS_SECRET_ACCESS_KEY를 모두 전달하고 있는지 확인합니다. 액세스 키와 SigV4 플러그인에 대한 자세한 내용은 the section called “인증을 위한 IAM 자격 증명 AWS ” 섹션을 참조하세요.

드라이버를 사용하여 Amazon Keyspaces 테이블에 연결하려고 하지만 **Invalid signature** 오류를 수신합니다.

잘못된 보안 인증을 사용한 경우 다음과 같은 오류가 발생할 수 있습니다.

```
cassandra.cluster.NoHostAvailable: (‘Unable to connect to any servers’,
 {‘44.234.22.134:9142’:
 AuthenticationFailed(‘Failed to authenticate to 44.234.22.134:9142: Error from server:
 code=0100
 [Bad credentials] message=“Authentication failure: Invalid signature”’)})
```

이 문제를 해결하려면 전달하는 보안 인증이 Amazon Keyspaces에 액세스하도록 구성한 IAM 사용자 또는 역할과 연결되어 있는지 확인합니다. 액세스 키와 SigV4 플러그인에 대한 자세한 내용은 the section called “인증을 위한 IAM 자격 증명 AWS ” 섹션을 참조하세요.

VPC 엔드포인트 연결이 제대로 작동하지 않음

VPC 엔드포인트를 사용하여 Amazon Keyspaces에 연결하려고 하는데 토큰 맵 오류가 발생하거나 처리량이 낮습니다.

VPC 엔드포인트 연결이 올바르게 구성되지 않은 경우가 이에 해당할 수 있습니다.

이러한 문제를 해결하려면 다음 구성 세부 정보를 확인합니다. step-by-step 자습서를 따라 Amazon Keyspaces의 인터페이스 VPC 엔드포인트를 통한 연결을 구성하는 방법을 알아보려면 을 참조하십시오. the section called “VPC 엔드포인트와 연결”

1. Amazon Keyspaces에 연결하는 데 사용되는 IAM 엔티티가 다음 예와 같이 사용자 테이블에 대한 읽기/쓰기 액세스 권한과 시스템 테이블에 대한 읽기 권한을 가지고 있는지 확인합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select",
            "cassandra:Modify"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

2. 다음 예와 같이 Amazon Keyspaces에 연결하는 데 사용되는 IAM 엔티티에 Amazon EC2 인스턴스의 VPC 엔드포인트 정보에 액세스하는 데 필요한 읽기 권한이 있는지 확인합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListVPCEndpoints",
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcEndpoints"
         ],
         "Resource":"*"
      }
   ]
```

```
}
```

> **Note**
>
> 관리형 정책 `AmazonKeyspacesReadOnlyAccess_v2` 및 `AmazonKeyspacesFullAccess`에는 Amazon Keyspaces가 Amazon EC2 인스턴스에 액세스하여 사용 가능한 인터페이스 VPC 엔드포인트에 대한 정보를 읽을 수 있도록 하는 데 필요한 권한이 포함되어 있습니다.

VPC 엔드포인트에 대한 자세한 내용은 the section called “Amazon Keyspaces에 인터페이스 VPC 엔드포인트 사용” 섹션을 참조하세요.

3. 이 예와 같이 Java 드라이버의 SSL 구성이 호스트 이름 검증을 false로 설정하는지 확인합니다.

```
hostname-validation = false
```

드라이버 구성에 대한 자세한 내용은 the section called “2단계: 드라이버 구성” 섹션을 참조하세요.

4. VPC 엔드포인트가 올바르게 구성되었는지 확인하기 위해 VPC 내에서 다음 문을 실행할 수 있습니다.

> **Note**
>
> 로컬 개발자 환경 또는 Amazon Keyspaces CQL 편집기는 퍼블릭 엔드포인트를 사용하기 때문에 이 구성을 확인하기 위해 사용할 수 없습니다.

```
SELECT peer FROM system.peers;
```

출력은 이 예와 비슷해야 하며 VPC 설정 및 AWS 지역에 따라 프라이빗 IP 주소가 있는 2~6개의 노드를 반환해야 합니다.

```
peer
---------------
 192.0.2.0.15
 192.0.2.0.24
 192.0.2.0.13
```

```
 192.0.2.0.7
 192.0.2.0.8

(5 rows)
```

cassandra-stress를 사용하여 연결할 수 없음

cassandra-stress 명령을 사용하여 Amazon Keyspaces에 연결하려고 하지만 **SSL context** 오류를 수신합니다.

이 문제는 Amazon Keyspaces에 연결하려고 하지만 trustStore가 제대로 설정되지 않은 경우 발생합니다. Amazon Keyspaces에서는 클라이언트와의 연결을 보호하는 데 도움이 되는 전송 계층 보안(TLS)을 사용해야 합니다.

이 경우 다음 오류가 표시됩니다.

```
Error creating the initializing the SSL Context
```

이 문제를 해결하려면 이 주제 the section called “시작하기 전 준비 사항”에 나와 있는 대로 trustStore를 설정하는 지침을 따르세요.

trustStore가 설정되면 다음 명령을 사용하여 연결할 수 있어야 합니다.

```
./cassandra-stress user profile=./profile.yaml n=100 "ops(insert=1,select=1)"
 cl=LOCAL_QUORUM -node "cassandra.eu-north-1.amazonaws.com" -port native=9142
 -transport ssl-alg="PKIX" truststore="./cassandra_truststore.jks" truststore-
password="trustStore_pw" -mode native cql3 user="user_name" password="password"
```

IAM ID를 사용하여 연결할 수 없음

IAM ID를 사용하여 Amazon Keyspaces 테이블에 연결하려고 하지만 **Unauthorized** 오류를 수신합니다.

정책을 구현하고 사용자에게 필요한 권한을 먼저 부여하지 않고 IAM ID(예: IAM 사용자)를 사용하여 Amazon Keyspaces 테이블에 연결하려고 하면 이 오류가 발생합니다.

이 경우 다음 오류가 표시됩니다.

```
Connection error: ('Unable to connect to any servers', {'3.234.248.202':
 AuthenticationFailed('Failed to authenticate to 3.234.248.202:
```

```
Error from server: code=2100 [Unauthorized] message="User
 arn:aws:iam::1234567890123:user/testuser has no permissions."',)})
```

이 문제를 해결하려면 IAM 사용자의 권한을 확인합니다. 표준 드라이버를 사용하여 연결하려면 사용자는 최소한 시스템 테이블에 대한 SELECT 액세스 권한을 가지고 있어야 합니다. 이는 대부분의 드라이버가 연결 시 시스템 키스페이스/테이블을 읽기 때문입니다.

Amazon Keyspaces 시스템 및 사용자 테이블에 액세스 권한을 부여하는 IAM 정책의 예는 the section called “Amazon Keyspaces 테이블에 액세스” 섹션을 참조하세요.

IAM과 관련된 문제 해결 섹션을 검토하려면 the section called “문제 해결” 섹션을 참조하세요.

cqlsh를 사용하여 데이터를 가져오려고 하는데 Amazon Keyspaces 테이블에 대한 연결이 끊어졌습니다.

cqlsh를 사용하여 Amazon Keyspaces에 데이터를 업로드하려고 하지만 연결 오류를 수신합니다.

cqlsh 클라이언트가 서버로부터 유형에 상관없이 연속으로 세 개의 오류를 수신한 후 Amazon Keyspaces에 대한 연결이 실패합니다. cqlsh 클라이언트가 실패하고 다음 메시지가 표시됩니다.

```
Failed to import 1 rows: NoHostAvailable - , will retry later, attempt 3 of 100
```

이 오류를 해결하려면 가져올 데이터가 Amazon Keyspaces의 테이블 스키마와 일치하는지 확인해야 합니다. 가져오기 파일에 구문 분석 오류가 있는지 검토합니다. 오류를 격리하는 INSERT 문을 사용하여 단일 데이터 행을 사용해 볼 수 있습니다.

클라이언트는 자동으로 연결 재설정을 시도합니다.

Amazon Keyspaces의 용량 관리 문제 해결

서버리스 용량에 문제가 있으신가요? 다음에서는 몇 가지 일반적인 문제와 이에 대한 해결 방법에 대해 설명합니다.

서버리스 용량 오류

이 섹션에서는 서버리스 용량 관리와 관련된 오류를 인식하는 방법과 해결 방법을 설명합니다. 예를 들어 애플리케이션이 프로비저닝된 처리량 용량을 초과하면 용량 부족 이벤트가 관찰될 수 있습니다.

Apache Cassandra는 여러 노드에서 실행되도록 설계된 클러스터 기반 소프트웨어이므로 처리량 용량과 같은 서버리스 기능과 관련된 예외 메시지가 없습니다. 대부분의 드라이버는 Apache Cassandra에

서 사용할 수 있는 오류 코드만 이해하므로 Amazon Keyspaces는 호환성을 유지하기 위해 동일한 오류 코드 세트를 사용합니다.

Cassandra 오류를 기본 용량 이벤트에 매핑하려면 Amazon을 사용하여 관련 Amazon Keyspaces CloudWatch 지표를 모니터링할 수 있습니다. 클라이언트 측 오류를 초래하는 용량 부족 이벤트는 이벤트를 일으킨 리소스에 따라 다음 세 그룹으로 분류할 수 있습니다.

- 테이블 - 테이블에 대해 프로비저닝된 용량 모드를 선택했는데 애플리케이션이 프로비저닝된 처리량을 초과하는 경우 용량 부족 오류가 발생할 수 있습니다. 자세한 정보는 the section called “읽기/쓰기 용량 모드”을 참조하세요.
- 파티션 - 특정 파티션의 트래픽이 3,000RCU 또는 1,000WCU를 초과하는 경우 용량 부족 이벤트가 발생할 수 있습니다. 트래픽을 파티션 전체에 균일하게 분배하는 것이 가장 좋습니다. 자세한 정보는 the section called “데이터 모델링”을 참조하세요.
- 연결 - 연결당 초당 최대 작업 수의 할당량을 초과하면 처리량이 부족해질 수 있습니다. 처리량을 늘리려면 드라이버로 연결을 구성할 때 기본 연결 수를 늘릴 수 있습니다.

  Amazon Keyspace의 연결을 구성하는 방법을 알아보려면 을 참조하십시오. the section called “연결 구성 방법” VPC 엔드포인트를 통한 연결 최적화에 대한 자세한 내용은 을 참조하십시오. the section called “VPC 엔드포인트 연결”

클라이언트 측 오류를 반환하는 용량 부족 이벤트를 일으키는 리소스를 확인하려면 Amazon Keyspaces 콘솔에서 대시보드를 확인할 수 있습니다. 기본적으로 콘솔은 테이블의 용량 탭에 있는 용량 및 관련 CloudWatch 지표 섹션에서 가장 일반적인 용량 및 트래픽 관련 지표를 집계하여 보여줍니다.

Amazon을 사용하여 자체 대시보드를 만들려면 다음 Amazon CloudWatch Keyspaces 지표를 확인하십시오.

- `PerConnectionRequestRateExceeded` - 연결당 요청 속도에 대한 할당량을 초과하는 Amazon Keyspaces에 대한 요청입니다. Amazon Keyspaces에 대한 각 클라이언트 연결은 초당 최대 3000개의 CQL 요청을 지원할 수 있습니다. 여러 연결을 생성하여 초당 3000개 이상의 요청을 수행할 수 있습니다.
- `ReadThrottleEvents` - 테이블의 읽기 용량을 초과하는 Amazon Keyspaces에 대한 요청입니다.
- `StoragePartitionThroughputCapacityExceeded` - 파티션의 처리량 용량을 초과하는 Amazon Keyspaces 스토리지 파티션에 대한 요청입니다. Amazon Keyspaces 스토리지 파티션은 초당 최대 1000 WCU/WRU 및 초당 3000 RCU/RRU를 지원할 수 있습니다. 이러한 예외를 완화하려면 데이터 모델을 검토하여 더 많은 파티션에 읽기/쓰기 트래픽을 분산하는 것이 좋습니다.

- `WriteThrottleEvents` - 테이블의 읽기 용량을 초과하는 Amazon Keyspaces에 대한 요청입니다.

자세히 CloudWatch 알아보려면 을 참조하십시오the section called “를 통한 모니터링 CloudWatch”. Amazon Keyspace에 사용할 수 있는 모든 CloudWatch 지표 목록은 을 참조하십시오. the section called “지표 및 차원”

> **Note**
>
> Amazon GitHub Keyspaces에서 일반적으로 관찰되는 모든 지표를 보여주는 사용자 지정 대시보드를 시작하려면 샘플 리포지토리에 있는 사전 구축된 CloudWatch 템플릿을 사용할 수 있습니다.AWS

주제

- 클라이언트 드라이버로부터 NoHostAvailable 용량 부족 오류가 발생했습니다.
- 데이터를 가져오는 동안 쓰기 제한 시간 오류가 발생함
- 키스페이스 또는 테이블의 실제 스토리지 크기를 볼 수 없음

클라이언트 드라이버로부터 **NoHostAvailable** 용량 부족 오류가 발생했습니다.

테이블에 대한 **Read_Timeout** 또는 **Write_Timeout** 예외가 표시됩니다.

용량이 충분하지 않은 상태에서 Amazon Keyspaces 테이블에 반복적으로 쓰거나 읽으려고 하면 드라이버와 관련된 클라이언트 측 오류가 발생할 수 있습니다.

프로비저닝된 처리량 지표와 실제 처리량 지표, 테이블의 용량 부족 이벤트를 모니터링하는 CloudWatch 데 사용합니다. 예를 들어 처리량 용량이 충분하지 않은 읽기 요청은 `Read_Timeout` 예외와 함께 실패하고 `ReadThrottleEvents` 지표에 게시됩니다. 처리량 용량이 충분하지 않은 쓰기 요청은 `Write_Timeout` 예외와 함께 실패하고 `WriteThrottleEvents` 지표에 게시됩니다. 이러한 지표에 대한 자세한 내용은 the section called “지표 및 차원” 섹션을 참조하세요.

문제를 해결하려면 다음 옵션을 고려하세요.

- 애플리케이션이 소비할 수 있는 최대 처리량 용량인 테이블에 대한 프로비저닝된 처리량을 늘립니다. 자세한 정보는 the section called “읽기 용량 단위 및 쓰기 용량 단위”을 참조하세요.
- 자동 크기 조정을 통해 서비스가 사용자 대신 처리량 용량을 관리하도록 합니다. 자세한 정보는 the section called “Auto Scaling을 통한 처리 용량 관리”을 참조하세요.

- 테이블에 온디맨드 용량 모드를 선택했습니다. 자세한 정보는 the section called “온디맨드 용량 모드”을 참조하세요.

계정의 기본 용량 할당량을 늘려야 하는 경우 할당량 섹션을 참조하세요.

파티션 용량 초과와 관련된 오류가 표시됩니다.

오류가 표시되면 파티션 용량이 일시적으로 `StoragePartitionThroughputCapacityExceeded` 초과된 것입니다. 이는 적응형 용량 또는 온디맨드 용량에 의해 자동으로 처리될 수 있습니다. 이러한 오류를 완화하려면 데이터 모델을 검토하여 읽기/쓰기 트래픽을 더 많은 파티션에 분산하는 것이 좋습니다. Amazon Keyspaces 스토리지 파티션은 초당 최대 1000 WCU/WRU 및 초당 3000 RCU/RRU를 지원할 수 있습니다. 읽기/쓰기 트래픽을 더 많은 파티션에 분산하도록 데이터 모델을 개선하는 방법에 대한 자세한 내용은 the section called “데이터 모델링” 섹션을 참조하세요.

동일한 논리적 파티션에 정적 및 비정적 데이터를 포함하는 동시 쓰기 작업의 비율이 높아진 경우에도 `Write_Timeout` 예외가 발생할 수 있습니다. 트래픽이 동일한 논리적 파티션 내에서 정적 및 비정적 데이터를 포함하는 여러 개의 동시 쓰기 작업을 실행할 것으로 예상되는 경우 정적 데이터와 비정적 데이터를 따로 쓰는 것이 좋습니다. 데이터를 별도로 쓰는 것도 처리량 비용을 최적화하는 데 도움이 됩니다.

연결 요청 속도 초과와 관련된 오류가 표시됩니다.

이 오류는 다음 원인 중 하나로 `PerConnectionRequestRateExceeded` 인해 발생합니다.

- 세션당 구성된 연결이 충분하지 않을 수 있습니다.
- VPC 엔드포인트 권한이 올바르게 구성되어 있지 않기 때문에 사용 가능한 피어보다 연결 수가 적을 수 있습니다. VPC 엔드포인트 정책에 대한 자세한 내용은 the section called “Amazon Keyspaces에 인터페이스 VPC 엔드포인트 사용” 섹션을 참조하세요.
- 4.x 드라이버를 사용하는 경우 호스트 이름 검증이 활성화되어 있는지 확인합니다. 드라이버는 기본적으로 TLS 호스트 이름 확인을 활성화합니다. 이 구성으로 인해 Amazon Keyspaces가 드라이버에 단일 노드 클러스터로 표시됩니다. 호스트 이름 인증을 끄는 것이 좋습니다.

다음 모범 사례를 따라 연결 및 처리량을 최적화하는 것이 좋습니다.

- CQL 쿼리 처리량 조정 구성

  Amazon Keyspaces는 TCP 연결당 초당 최대 3,000개의 CQL 쿼리를 지원하지만 드라이버가 설정할 수 있는 연결 수에는 제한이 없습니다.

대부분의 오픈 소스 Cassandra 드라이버는 Cassandra에 연결 풀을 설정하고 연결 풀 전체에 걸쳐 쿼리를 로드 밸런싱합니다. Amazon Keyspaces는 드라이버에게 9개의 피어 IP 주소를 노출합니다. 대부분의 드라이버의 기본 동작은 각 피어 IP 주소마다 하나의 연결을 설정하는 것입니다. 따라서 기본 설정을 사용하는 드라이버의 최대 CQL 쿼리 처리량은 초당 27,000개의 CQL 쿼리입니다.

이 수를 늘리려면 드라이버가 연결 풀에서 유지 관리하는 IP 주소당 연결 수를 늘리는 것이 좋습니다. 예를 들어 IP 주소당 최대 연결 수를 2로 설정하면 드라이버의 최대 처리량이 초당 54,000개의 CQL 쿼리로 두 배로 늘어납니다.

- 단일 노드 연결을 최적화합니다.

  기본적으로 대부분의 오픈 소스 Cassandra 드라이버는 세션을 설정할 때 `system.peers` 테이블에 표시된 모든 IP 주소에 대해 하나 이상의 연결을 설정합니다. 하지만 특정 구성에서는 드라이버가 단일 Amazon Keyspaces IP 주소에 연결될 수 있습니다. 이는 드라이버가 피어 노드 (예: DataStax Java 드라이버) 의 SSL 호스트 이름 검증을 시도하거나 VPC 엔드포인트를 통해 연결할 때 발생할 수 있습니다.

  여러 IP 주소에 연결할 때 드라이버와 동일한 가용성 및 성능을 얻으려면 다음을 수행하는 것이 좋습니다.

  - 원하는 클라이언트 처리량에 따라 IP당 연결 수를 9개 이상으로 늘립니다.
  - 동일한 노드에서 재시도가 실행되도록 하는 사용자 지정 재시도 정책을 생성합니다.
  - VPC 엔드포인트를 사용하는 경우 Amazon Keyspaces에 연결하는 데 사용되는 IAM 엔티티에 엔드포인트 및 네트워크 인터페이스 정보에 대한 VPC를 쿼리할 수 있는 액세스 권한을 부여합니다. 이렇게 하면 로드 밸런싱이 개선되고 읽기/쓰기 처리량이 늘어납니다. 자세한 정보는 ???을 참조하세요.

데이터를 가져오는 동안 쓰기 제한 시간 오류가 발생함

`cqlsh COPY` 명령을 사용하여 데이터를 업로드할 때 제한 시간 오류가 발생합니다.

```
Failed to import 1 rows: WriteTimeout - Error from server: code=1100 [Coordinator node
 timed out waiting for replica nodes' responses]
 message="Operation timed out - received only 0 responses." info={'received_responses':
 0, 'required_responses': 2, 'write_type': 'SIMPLE', 'consistency':
 'LOCAL_QUORUM'}, will retry later, attempt 1 of 100
```

Amazon Keyspaces는 처리량 용량이 충분하지 않아 쓰기 요청이 실패하는 경우를 나타내기 위해 `ReadTimeout` 및 `WriteTimeout` 예외를 사용합니다. 용량 부족 예외를 진단하는 데 도움이 되도록 Amazon Keyspace는 Amazon에 다음 지표를 게시합니다. CloudWatch

- `WriteThrottleEvents`
- `ReadThrottledEvents`
- `StoragePartitionThroughputCapacityExceeded`

데이터 로드 중 용량 부족 오류를 해결하려면 작업자당 쓰기 속도 또는 총 수집 속도를 낮춘 다음 행 업로드를 다시 시도합니다. 자세한 정보는 the section called “4단계: `cqlsh COPY FROM` 설정 구성”을 참조하세요. 보다 강력한 데이터 업로드 옵션을 원하면 리포지토리에서 사용할 수 있는 DSBulk를 사용해 보십시오. GitHub step-by-step지침은 을 참조하십시오the section called “DSBulk를 사용하여 데이터 로드”.

키스페이스 또는 테이블의 실제 스토리지 크기를 볼 수 없음

키스페이스 또는 테이블의 실제 스토리지 크기를 볼 수 없습니다.

테이블의 스토리지 크기에 대한 자세한 내용은 을 참조하십시오the section called “테이블 수준에서 비용 평가”. 테이블의 행 크기 계산을 시작하여 저장소 크기를 추정할 수도 있습니다. 행 크기 계산에 대한 자세한 지침은 the section called “행 크기 계산”에서 확인할 수 있습니다.

Amazon Keyspaces의 데이터 정의 언어 문제 해결

리소스 생성에 문제가 있으신가요? 다음에서는 몇 가지 일반적인 문제와 이에 대한 해결 방법에 대해 설명합니다.

데이터 정의 언어 오류

Amazon Keyspaces는 데이터 정의 언어(DDL) 작업을 비동기적으로 수행합니다(예: 키스페이스 및 테이블 생성 및 삭제). 애플리케이션이 준비가 되기 전에 리소스를 사용하려고 하면 작업이 실패합니다.

에서 새 키스페이스 및 테이블의 생성 상태를 모니터링할 수 있으며 AWS Management Console, 이는 키스페이스나 테이블이 보류 또는 활성 상태인 시기를 나타냅니다. 또한 시스템 스키마 테이블을 쿼리하여 프로그래밍 방식으로 새 키스페이스 또는 테이블의 생성 상태를 모니터링할 수 있습니다. 키스페이스 또는 테이블을 사용할 준비가 되면 시스템 스키마에 해당 키스페이스나 테이블이 표시됩니다.

> **Note**
>
> 를 사용하여 키스페이스 생성을 최적화하려면 이 유틸리티를 사용하여 AWS CloudFormation CQL 스크립트를 템플릿으로 변환할 수 있습니다. CloudFormation 이 도구는 리포지토리에서 사용할 수 있습니다. GitHub

주제

- 새 키스페이스를 생성했지만 해당 키스페이스를 보거나 액세스할 수 없음
- 새 테이블을 생성했지만 해당 테이블을 보거나 액세스할 수 없음
- Amazon Keyspaces point-in-time 복구 (PITR) 를 사용하여 테이블을 복원하려고 하는데 복원에 실패했습니다.
- INSERT/UPDATE를 사용하여 사용자 지정 TTL(Time To Live) 설정을 편집하려고 하지만 작업이 실패함
- Amazon Keyspaces 테이블에 데이터를 업로드하려고 하는데 열 수를 초과한다는 오류가 발생함
- Amazon Keyspaces 테이블에서 데이터를 삭제하려고 하는데 해당 범위에서 삭제가 실패함

새 키스페이스를 생성했지만 해당 키스페이스를 보거나 액세스할 수 없음

새 키스페이스에 액세스하려는 애플리케이션에서 오류가 발생합니다.

아직 비동기적으로 생성 중인 새로 생성된 Amazon Keyspaces 키스페이스에 액세스하려고 하면 오류가 발생합니다. 다음은 오류의 예입니다.

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured
 keyspace mykeyspace"
```

새 키스페이스를 사용할 준비가 되면 확인하는 권장 설계 패턴은 Amazon Keyspaces 시스템 스키마 테이블(system_schema_mcs.*)을 폴링하는 것입니다.

자세한 정보는 the section called “키스페이스 생성”을 참조하세요.

새 테이블을 생성했지만 해당 테이블을 보거나 액세스할 수 없음

새 테이블에 액세스하려는 애플리케이션에서 오류가 발생합니다.

아직 비동기적으로 생성 중인 새로 생성된 Amazon Keyspaces 테이블에 액세스하려고 하면 오류가 발생합니다. 예를 들어 아직 사용할 수 없는 테이블을 쿼리하려고 하면 `unconfigured table` 오류로 실패합니다.

```
InvalidRequest: Error from server: code=2200 [Invalid query] message="unconfigured
 table mykeyspace.mytable"
```

`sync_table()`을 사용하여 테이블을 보려고 하면 `KeyError`로 실패합니다.

```
KeyError: 'mytable'
```

새 테이블을 사용할 준비가 되면 확인하는 권장 설계 패턴은 Amazon Keyspaces 시스템 스키마 테이블(system_schema_mcs.*)을 폴링하는 것입니다.

다음은 생성 중인 테이블의 예제 출력입니다.

```
user-at-123@cqlsh:system_schema_mcs> select table_name,status from
 system_schema_mcs.tables where keyspace_name='example_keyspace' and
 table_name='example_table';

table_name | status

------------+----------

example_table | CREATING

(1 rows)
```

다음은 활성화된 테이블의 예제 출력입니다.

```
user-at-123@cqlsh:system_schema_mcs> select table_name,status from
 system_schema_mcs.tables where keyspace_name='example_keyspace' and
 table_name='example_table';

table_name | status

------------+----------

example_table | ACTIVE

(1 rows)
```

자세한 정보는 the section called “테이블 생성”을 참조하세요.

Amazon Keyspaces point-in-time 복구 (PITR) 를 사용하여 테이블을 복원하려고 하는데 복원에 실패했습니다.

Amazon Keyspaces 테이블을 PITR (point-in-time 복구 포함) 으로 복원하려고 할 때 복원 프로세스가 시작되었지만 성공적으로 완료되지 않은 경우 이 특정 테이블의 복원 프로세스에 필요한 모든 필수 권한을 구성하지 않았을 수 있습니다.

사용자 권한 외에도 Amazon Keyspaces는 보안 주체를 대신하여 복원 프로세스 중에 작업을 수행할 수 있는 권한을 요구할 수 있습니다. 테이블이 고객 관리형 키로 암호화되거나 들어오는 트래픽을 제한하는 IAM 정책을 사용하는 경우가 이에 해당합니다.

예를 들어 IAM 정책에서 조건 키를 사용하여 소스 트래픽을 특정 엔드포인트 또는 IP 범위로 제한하는 경우 복원 작업이 실패합니다. Amazon Keyspaces가 보안 주체를 대신하여 테이블 복원 작업을 수행하도록 허용하려면 IAM 정책에 `aws:ViaAWSService` 글로벌 조건 키를 추가해야 합니다.

테이블 복원 권한에 대한 자세한 내용은 the section called “권한 복원” 섹션을 참조하세요.

INSERT/UPDATE를 사용하여 사용자 지정 TTL(Time To Live) 설정을 편집하려고 하지만 작업이 실패함

사용자 지정 TTL 값을 삽입하거나 업데이트하려는 경우 다음 오류가 발생하여 작업이 실패할 수 있습니다.

```
TTL is not yet supported.
```

`INSERT` 또는 `UPDATE` 작업을 사용하여 행 또는 열의 사용자 지정 TTL 값을 지정하려면 먼저 테이블에 TTL을 활성화해야 합니다. `ttl` 사용자 지정 속성을 사용하여 테이블에 TTL을 활성화할 수 있습니다.

테이블에 대한 사용자 지정 TTL 설정을 활성화하는 방법에 대한 자세한 내용은 the section called “사용자 지정 속성을 사용하여 기존 테이블에 대한 TTL(Time To Live)을 활성화하는 방법” 섹션을 참조하세요.

Amazon Keyspaces 테이블에 데이터를 업로드하려고 하는데 열 수를 초과한다는 오류가 발생함

데이터를 업로드하고 있는데 동시에 업데이트할 수 있는 열 수를 초과했습니다.

이 오류는 테이블 스키마가 최대 크기인 350KB를 초과할 때 발생합니다. 자세한 정보는 할당량을 참조하세요.

Amazon Keyspaces 테이블에서 데이터를 삭제하려고 하는데 해당 범위에서 삭제가 실패함

파티션 키로 데이터를 삭제하려고 하는데 범위 삭제 오류가 발생했습니다.

이 오류는 한 번의 삭제 작업으로 1,000개 이상의 행을 삭제하려고 할 때 발생합니다.

```
Range delete requests are limited by the amount of items that can be deleted in a
 single range.
```

자세한 정보는 the section called “범위 삭제”을 참조하세요.

단일 파티션 내에서 1,000개가 넘는 행을 삭제하려면 다음 옵션을 고려해 보세요.

- 파티션별 삭제 - 파티션 대다수가 1,000행 미만인 경우 파티션별로 데이터를 삭제해 볼 수 있습니다. 파티션에 1,000개 이상의 행이 포함된 경우 대신 클러스터링 열을 기준으로 삭제를 시도합니다.
- 클러스터링 열별 삭제 - 모델에 클러스터링 열이 여러 개 있는 경우 열 계층 구조를 사용하여 여러 행을 삭제할 수 있습니다. 클러스터링 열은 중첩 구조이므로 최상위 열에 대해 작업하여 많은 행을 삭제할 수 있습니다.
- 개별 행별 삭제 - 행을 반복하고 전체 프라이머리 키(파티션 열 및 클러스터링 열)로 각 행을 삭제할 수 있습니다.
- 행을 여러 파티션으로 분할하는 것이 가장 좋습니다. Amazon Keyspaces에서는 처리량을 테이블 파티션에 분산하는 것이 좋습니다. 이렇게 하면 물리적 리소스 전체에 데이터와 액세스가 균등하게 분산되므로 처리량이 가장 좋습니다. 자세한 정보는 the section called “데이터 모델링”을 참조하세요.

워크로드가 많은 삭제 작업을 계획할 때는 다음 권장 사항도 고려해 보세요.

- Amazon Keyspaces를 사용하면 파티션에 사실상 무제한의 행이 포함될 수 있습니다. 이를 통해 기존 Cassandra 지침인 100MB보다 “더 넓게” 파티션을 확장할 수 있습니다. 시간이 지남에 따라 시계열 또는 원장의 데이터가 1기가바이트 이상 증가하는 것은 드문 일이 아닙니다.
- Amazon Keyspaces를 사용하면 과중한 워크로드로 인해 삭제 작업을 수행해야 할 때 고려할 압축 전략이나 삭제 표시가 없습니다. 읽기 성능에 영향을 주지 않으면서 원하는 만큼 데이터를 삭제할 수 있습니다.

Amazon Keyspaces(Apache Cassandra용)에서 서버리스 리소스 관리

Amazon Keyspaces(Apache Cassandra용)는 서버리스입니다. Amazon Keyspaces는 클러스터의 노드를 통해 워크로드의 스토리지 및 컴퓨팅 리소스를 배포, 관리 및 유지 관리하는 대신 스토리지 및 읽기/쓰기 처리량 리소스를 테이블에 직접 할당합니다.

이 장에서는 Amazon Keyspaces의 서버리스 리소스 관리에 대한 세부 정보를 제공합니다. Amazon에서 서버리스 리소스를 모니터링하는 방법을 CloudWatch 알아보려면 을 참조하십시오the section called “를 통한 모니터링 CloudWatch”.

주제

- Amazon Keyspaces의 스토리지
- Amazon Keyspaces의 읽기/쓰기 용량 모드
- Amazon Keyspaces 자동 크기 조정을 통해 처리 용량을 자동으로 관리합니다.
- Amazon Keyspaces에서 버스트 용량을 효과적으로 사용하기
- Amazon Keyspace의 용량 사용량을 추정하는 방법

Amazon Keyspaces의 스토리지

Amazon Keyspaces(Apache Cassandra용)는 테이블에 저장된 실제 데이터를 기반으로 테이블에 스토리지를 자동으로 프로비저닝합니다. 테이블에 스토리지를 미리 프로비저닝할 필요는 없습니다. Amazon Keyspaces는 애플리케이션이 데이터를 쓰고, 업데이트하고, 삭제함에 따라 테이블 스토리지를 자동으로 확장 및 축소합니다. 기존 Apache Cassandra 클러스터와 달리 Amazon Keyspaces는 압축과 같은 저수준 시스템 작업을 지원하기 위해 추가 스토리지가 필요하지 않습니다. 사용하는 스토리지에 대해서만 비용을 지불합니다.

Amazon Keyspaces는 기본적으로 복제 인수 3으로 키스페이스를 구성합니다. 복제 인수는 수정할 수 없습니다. Amazon Keyspace는 AWS 고가용성을 위해 여러 가용 영역에 테이블 데이터를 자동으로 세 번 복제합니다. Amazon Keyspaces 스토리지의 GB당 요금에는 이미 복제가 포함되어 있습니다. 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 요금을 참조하세요.

Amazon Keyspaces는 테이블 크기를 지속적으로 모니터링하여 스토리지 요금을 결정합니다. Amazon Keyspaces가 청구 대상 데이터 크기를 계산하는 방법에 대한 자세한 내용은 the section called “행 크기 계산” 섹션을 참조하세요.

Amazon Keyspaces의 읽기/쓰기 용량 모드

Amazon Keyspaces에는 테이블에서 읽기 및 쓰기 처리를 위한 두 가지 읽기/쓰기 용량 모드가 있습니다.

- 온디맨드(기본값)
- 프로비저닝됨

선택하는 읽기/쓰기 용량 모드는 읽기 및 쓰기 처리량에 대한 청구 방식과 테이블 처리량 용량 관리 방식을 제어합니다.

주제

온디맨드 용량 모드

Amazon Keyspaces(Apache Cassandra용) 온디맨드 용량 모드는 용량 계획 없이 초당 수천 개의 요청을 처리할 수 있는 유연한 청구 옵션입니다. 이 옵션은 읽기 및 쓰기 요청에 대한 pay-per-request 요금을 제공하므로 사용한 만큼만 비용을 지불하면 됩니다.

온디맨드 모드를 선택하면 Amazon Keyspaces는 테이블의 처리량 용량을 이전에 도달한 트래픽 수준까지 즉시 확장하고 애플리케이션 트래픽이 감소할 때 다시 축소할 수 있습니다. 워크로드의 트래픽 수준이 새로운 피크를 기록할 경우에는 서비스가 신속하게 조정을 수행하여 테이블의 처리량 용량을 증가시킵니다. 새 테이블과 기존 테이블 모두에 대해 온디맨드 용량 모드를 활성화할 수 있습니다.

온디맨드 모드는 다음 중 하나에 해당되는 경우에 유용한 옵션입니다.

- 알 수 없는 워크로드를 포함하는 테이블을 새로 만들 경우
- 애플리케이션 트래픽이 예측 불가능한 경우
- 사용한 만큼에 대해서만 지불하는 요금제를 사용하려는 경우

온디맨드 모드를 시작하려면 콘솔을 사용하거나 몇 줄의 Cassandra 쿼리 언어(CQL) 코드를 사용하여 온디맨드 용량 모드를 사용하도록 새 테이블을 만들거나 기존 테이블을 업데이트하면 됩니다. 자세한 정보는 the section called “표”을 참조하세요.

주제

- 읽기 요청 단위 및 쓰기 요청 단위
- 피크 트래픽 및 크기 조정 속성
- 온디맨드 용량 모드의 초기 처리량

읽기 요청 단위 및 쓰기 요청 단위

온디맨드 용량 모드 테이블을 사용하면 애플리케이션에서 미리 사용할 것으로 예상되는 읽기 및 쓰기 처리량을 지정할 필요가 없습니다. Amazon Keyspaces에서는 읽기 요청 단위(RRU) 및 쓰기 요청 단위(WRU)의 측면에서 테이블에서 수행하는 읽기 및 쓰기에 대해 요금이 부과됩니다.

- RRU 1은 최대 4KB 크기의 행에 대한 LOCAL_QUORUM 읽기 요청 1회 또는 LOCAL_ONE 읽기 요청 2회를 나타냅니다. 4KB보다 큰 행을 읽어야 하는 경우 읽기 작업에 추가 RRU가 사용됩니다. 필요한 총 RRU 수는 행 크기, LOCAL_QUORUM 또는 LOCAL_ONE 읽기 일관성을 사용하려는지 여부에 따라 다릅니다. 예를 들어 8KB 행을 읽으려면 LOCAL_QUORUM 읽기 일관성을 사용하는 2RRU가 필요하고 LOCAL_ONE 읽기 일관성을 선택한 경우 1RRU가 필요합니다.
- WRU 1은 최대 1KB 크기의 행에 대한 1회의 쓰기를 나타냅니다. 모든 쓰기는 LOCAL_QUORUM 일관성을 사용하며 간단한 트랜잭션(LWT) 사용에 대한 추가 비용은 없습니다. 1KB보다 큰 행을 써야 하는 경우 쓰기 작업에 추가 WRU가 사용됩니다. 필요한 WRU의 총 수는 행 크기에 따라 달라집니다. 예를 들어 행 크기가 2KB인 경우 쓰기 요청 하나를 수행하려면 2WRU가 필요합니다.

지원되는 일관성 수준에 대한 자세한 내용은 the section called “지원되는 Cassandra 일관성 수준” 섹션을 참조하세요.

피크 트래픽 및 크기 조정 속성

온디맨드 용량 모드를 사용하는 Amazon Keyspaces 테이블은 애플리케이션의 트래픽 볼륨에 따라 자동으로 조정됩니다. 온디맨드 용량 모드의 테이블은 이전 피크 트래픽의 최대 2배 용량을 즉시 수용합니다. 예를 들어 애플리케이션의 트래픽 패턴은 초당 5,000~10,000회 LOCAL_QUORUM 읽기 사이에서 다양할 수 있으며 이때 초당 10,000회 읽기가 이전 트래픽 피크입니다.

이 패턴을 사용하면 온디맨드 용량 모드는 초당 최대 20,000회 읽기의 지속적인 트래픽을 즉시 수용할 수 있습니다. 애플리케이션이 초당 20,000회 읽기 트래픽을 지속하는 경우 해당 피크가 새로운 이전 피크가 되어 후속 트래픽은 초당 최대 40,000회 읽기에 도달할 수 있습니다.

테이블에서 이전 피크의 2배 이상이 필요한 경우 Amazon Keyspaces가 트래픽 볼륨이 증가함에 따라 자동으로 추가 용량을 할당합니다. 이렇게 하면 테이블에 추가 요청을 처리하기에 충분한 처리량 용량을 확보할 수 있습니다. 하지만 30분 이내에 이전 피크의 두 배를 초과하면 처리량 용량 부족 오류가 발생할 수 있습니다.

예를 들어 애플리케이션의 트래픽 패턴이 초당 5,000~10,000회의 강력히 일관된 읽기 사이에서 다양하다고 가정합니다. 이때 초당 20,000회 읽기는 이전에 도달한 트래픽 피크입니다. 이 경우 서비스에서는 초당 최대 40,000회의 읽기 횟수를 늘리기 전에 최소 30분 이상 트래픽 증가 간격을 두는 것이 좋습니다.

테이블의 읽기 및 쓰기 용량 사용량을 추정하는 방법을 알아보려면 을 참조하십시오the section called “용량 소비 추정”.

계정의 기본 할당량 및 할당량을 높이는 방법에 대한 자세한 내용은 할당량 섹션을 참조하세요.

온디맨드 용량 모드의 초기 처리량

온디맨드 용량 모드가 활성화된 새 테이블을 생성하거나 처음으로 기존 테이블을 온디맨드 용량 모드로 전환하는 경우 이전에 온디맨드 용량 모드를 사용하여 트래픽을 처리하지 않았더라도 테이블의 이전 피크 설정은 다음과 같습니다.

- 온디맨드 용량 모드로 새로 만든 테이블: 이전 피크는 2,000WRU와 6,000RRU였습니다. 이전 피크의 최대 두 배까지 즉시 늘릴 수 있습니다. 이렇게 하면 새로 생성된 온디맨드 테이블에서 최대 4,000WRU와 12,000RRU를 처리할 수 있습니다.
- 온디맨드 용량 모드로 전환된 기존 테이블: 이전 피크는 테이블에 대해 프로비저닝된 이전 WCU 및 RCU의 절반이거나 온디맨드 용량 모드를 통해 새로 생성된 테이블의 설정 중 더 높은 값입니다.

프로비저닝된 처리량 용량 모드

프로비저닝된 처리량 용량 모드를 선택하는 경우 애플리케이션에 필요한 초당 읽기 및 쓰기 횟수를 지정합니다. 이를 통해 Amazon Keyspaces 사용이 정의된 요청 속도 이하로 유지되도록 관리하여 가격을 최적화하고 예측 가능성을 유지할 수 있습니다. 프로비저닝된 처리량을 위한 자동 크기 조정에 대한 자세한 내용은 the section called “Auto Scaling을 통한 처리 용량 관리” 섹션을 참조하세요.

프로비저닝된 처리량 용량 모드는 다음 중 하나에 해당되는 경우에 유용한 옵션입니다.

- 애플리케이션 트래픽이 예측 가능한 경우
- 트래픽이 일관되거나 점진적으로 변화하는 애플리케이션을 실행할 경우
- 용량 요구 사항을 예측하여 가격을 최적화할 수 있습니다.

읽기 용량 단위 및 쓰기 용량 단위

프로비저닝된 처리량 용량 모드 테이블의 경우 읽기 용량 단위(RCU) 및 쓰기 용량 단위(WCU)의 측면에서 처리량을 지정합니다.

- RCU 1은 최대 4KB 크기의 행에 대해 초당 LOCAL_QUORUM 읽기 1회 또는 초당 LOCAL_ONE 읽기 2회를 나타냅니다. 4KB보다 큰 행을 읽어야 하는 경우 읽기 작업에 추가 RCU가 사용됩니다.

  필요한 총 RCU 수는 행 크기, LOCAL_QUORUM 또는 LOCAL_ONE 읽기를 원하는지 여부에 따라 다릅니다. 예를 들어 행 크기가 8KB인 경우 초당 LOCAL_QUORUM 읽기 1회를 유지하려면 2RCU개가 필요하고, LOCAL_ONE 읽기를 선택하면 1RCU가 필요합니다.
- WCU 1은 최대 1KB 크기의 행에 대한 초당 1회의 쓰기를 나타냅니다. 모든 쓰기는 LOCAL_QUORUM 일관성을 사용하며 간단한 트랜잭션(LWT) 사용에 대한 추가 비용은 없습니다. 1KB보다 큰 행을 써야 하는 경우 쓰기 작업에 추가 WCU가 사용됩니다.

  필요한 WCU의 총 수는 행 크기에 따라 달라집니다. 예를 들어 행 크기가 2KB인 경우 초당 쓰기 요청 하나를 유지하려면 2WCU가 필요합니다. 테이블의 읽기 및 쓰기 용량 사용량을 추정하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “용량 소비 추정”.

애플리케이션이 더 큰 행(1MB의 Amazon Keyspaces 최대 행 크기까지)을 읽거나 쓸 경우 더 많은 용량 단위를 사용합니다. 행 크기를 추정하는 방법에 대한 자세한 내용은 the section called “행 크기 계산” 섹션을 참조하세요. 예를 들어 6RCU 및 6WCU로 프로비저닝된 테이블을 생성한다고 가정하겠습니다. 이렇게 설정하면 애플리케이션에서 다음을 수행할 수 있습니다.

- 초당 최대 24KB의 LOCAL_QUORUM 읽기 수행(4KB × 6RCU)
- 초당 최대 48KB의 LOCAL_ONE 읽기 수행(읽기 처리량의 2배)
- 초당 최대 6KB 쓰기(1KB × 6WCU)

프로비저닝된 처리량은 애플리케이션이 테이블에서 사용할 수 있는 최대 처리량 용량의 양입니다. 애플리케이션이 프로비저닝된 처리량 용량을 초과하는 경우 용량 부족 오류가 발생할 수 있습니다.

예를 들어 처리량 용량이 충분하지 않은 읽기 요청은 `Read_Timeout` 예외와 함께 실패하고 `ReadThrottleEvents` 지표에 게시됩니다. 처리량 용량이 충분하지 않은 쓰기 요청은 `Write_Timeout` 예외와 함께 실패하고 `WriteThrottleEvents` 지표에 게시됩니다.

CloudWatch Amazon을 사용하여 프로비저닝된 처리량 지표와 실제 처리량 지표와 용량 부족 이벤트를 모니터링할 수 있습니다. 이러한 지표에 대한 자세한 내용은 the section called “지표 및 차원” 섹션을 참조하세요.

> **Note**
>
> 용량 부족으로 오류가 반복되면 클라이언트 측 드라이버별 예외가 발생할 수 있습니다. 예를 들어 DataStax Java 드라이버가 a와 함께 실패합니다. `NoHostAvailableException`

테이블의 처리량 용량 설정을 변경하려면 CQL을 사용한 AWS Management Console 또는 `ALTER TABLE` 문을 사용할 수 있습니다. 자세한 내용은 the section called “ALTER TABLE” 섹션을 참조하세요.

계정의 기본 할당량 및 할당량을 높이는 방법에 대한 자세한 내용은 할당량 섹션을 참조하세요.

용량 모드 관리 및 보기

Amazon Keyspaces 시스템 키스페이스에서 시스템 테이블을 쿼리하여 테이블에 대한 용량 모드 정보를 검토할 수 있습니다. 또한 테이블이 온디맨드 또는 프로비저닝된 처리량 용량 모드를 사용하는지 확인할 수 있습니다. 테이블이 프로비저닝된 처리량 용량 모드로 구성된 경우 테이블에 대한 프로비저닝된 처리량 용량을 확인할 수 있습니다.

예

```
SELECT * from system_schema_mcs.tables where keyspace_name = 'mykeyspace' and
 table_name = 'mytable';
```

온디맨드 용량 모드로 구성된 테이블은 다음을 반환합니다.

```
                {
                    'capacity_mode': {
                        'last_update_to_pay_per_request_timestamp':
 '1579551547603',
```

```
                        'throughput_mode': 'PAY_PER_REQUEST'
                    }
                }
```

온디맨드 처리량 용량 모드로 구성된 테이블은 다음을 반환합니다.

```
                {
                    'capacity_mode': {
                        'last_update_to_pay_per_request_timestamp':
'1579048006000',
                        'read_capacity_units': '5000',
                        'throughput_mode': 'PROVISIONED',
                        'write_capacity_units': '6000'
                    }
                }
```

`last_update_to_pay_per_request_timestamp` 값은 밀리초 단위로 측정됩니다.

테이블의 프로비저닝된 처리량 용량을 변경하려면 the section called “ALTER TABLE”를 사용합니다.

용량 모드 변경 시 고려 사항

테이블을 프로비저닝된 용량 모드에서 온디맨드 용량 모드로 전환할 경우 Amazon Keyspaces에서 테이블 및 파티션의 구조가 다양하게 변경됩니다. 이 프로세스는 몇 분 정도 걸릴 수 있습니다. 전환 기간 동안 테이블은 이전에 프로비저닝된 WCU 및 RCU 양과 일치하는 처리량을 제공합니다.

온디맨드 용량 모드에서 프로비저닝된 용량 모드로 다시 전환할 경우 테이블이 온디맨드 용량 모드로 설정되었을 때 도달한 이전 피크와 일치하는 처리량을 제공합니다.

> **Note**
>
> 24시간 동안 용량 모드를 프로비저닝에서 온디맨드로 한 번만 전환할 수 있습니다.

Amazon Keyspaces 자동 크기 조정을 통해 처리 용량을 자동으로 관리합니다.

대부분의 데이터베이스 워크로드는 원래 주기적으로 반복되거나 미리 예측하기가 어렵습니다. 낮 시간 동안에는 대부분의 사용자가 활성 상태인 소셜 네트워킹 앱을 예로 들어 보겠습니다. 이러한 데이터베이스에서는 주간 활동을 처리할 수 있어야 하지만, 밤에는 동일한 수준의 처리량이 필요 없습니다.

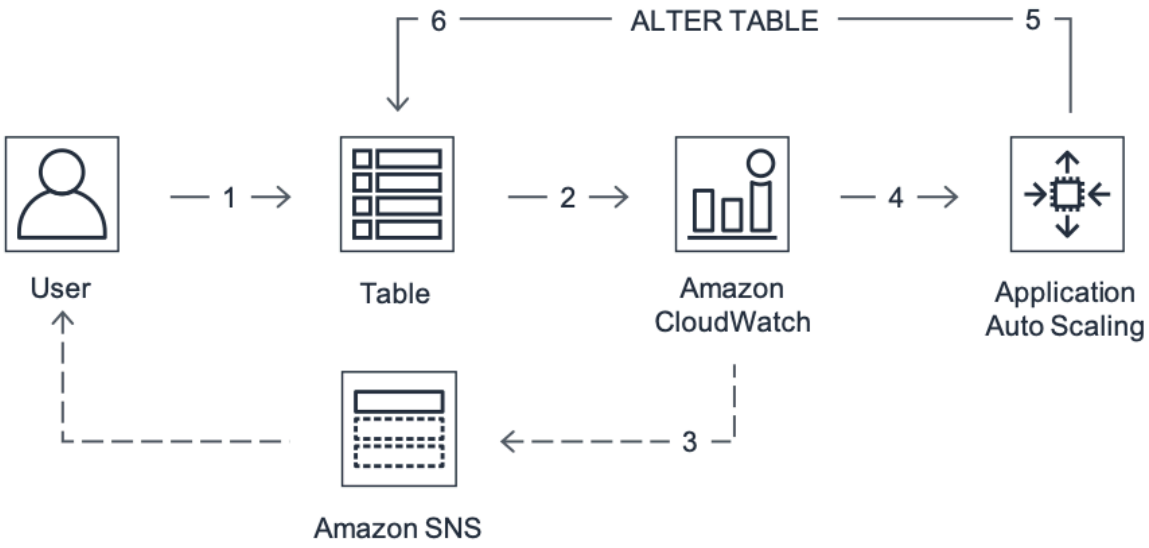
빠른 속도로 도입 중인 새로운 모바일 게임 앱을 또 다른 예로 들 수 있습니다. 이 게임의 인기가 높아지면 사용 가능한 데이터베이스 리소스 양을 초과하여 성능이 느려지고 고객 불만이 발생할 것입니다. 이러한 종류의 워크로드는 대개 사용량 변화에 따라 수동 개입을 통해 데이터베이스 리소스의 규모를 늘리거나 줄여야 합니다.

Amazon Keyspaces(Apache Cassandra용)는 실제 애플리케이션 트래픽에 따라 처리량 용량을 자동으로 조정하여 가변 워크로드에 대한 처리량 용량을 효율적으로 프로비저닝할 수 있도록 지원합니다. Amazon Keyspaces는 Application Auto Scaling 서비스를 사용하여 자동으로 테이블의 읽기 및 쓰기 용량을 늘리거나 줄입니다. Application Auto Scaling에 대한 자세한 내용은 Application Auto Scaling 사용 설명서를 참조하세요.

> **Note**
>
> Amazon Keyspaces Auto Scaling의 사용을 신속하게 시작하려면 the section called “콘솔 사용” 섹션을 참조하세요. 카산드라 쿼리 언어 (CQL) 를 사용하여 Amazon Keyspaces 규모 조정 정책을 관리하려면 을 참조하십시오. the section called “CQL 사용” CLI를 사용하여 Amazon Keyspaces 조정 정책을 관리하는 방법을 알아보려면 을 참조하십시오. the section called “CLI 사용”

Amazon Keyspaces Auto Scaling 작동 방식

아래 그림에는 Amazon Keyspaces Auto Scaling이 테이블의 처리량 용량을 관리하는 방법이 간단히 소개되어 있습니다.



테이블에 대해 Auto Scaling을 활성화하기 위해 규모 조정 정책을 만듭니다. 이 규모 조정 정책을 통해 읽기 용량이나 쓰기 용량(또는 둘 다)을 조정할 것인지 여부와 테이블에 대해 할당된 용량 단위의 최댓값 및 최솟값 설정을 지정할 수 있습니다.

규모 조정 정책은 목표 사용률도 정의합니다. 목표 사용률이란 특정 시점에 프로비저닝된 용량 단위에 대한 소비된 용량 단위의 비율을 백분율로 나타낸 값입니다. Auto Scaling은 목표 추적 알고리즘을 사용하여 실제 워크로드에 따라 테이블의 프로비저닝된 처리량을 확대 또는 축소합니다. 이렇게 하면 실제 용량 사용률을 목표 사용률과 비슷하게 유지할 수 있습니다.

읽기 및 쓰기 용량에 대해 20%와 90% 사이에서 Auto Scaling 목표 사용률 값을 설정할 수 있습니다. 기본 목표 사용률은 70% 입니다. 트래픽이 빠르게 변하고 용량을 더 빨리 스케일 업하려는 경우 목표 사용률을 낮게 설정할 수 있습니다. 애플리케이션 트래픽이 더 느리게 변하고 처리량 비용을 줄이려는 경우 목표 사용률을 더 높게 설정할 수도 있습니다.

조정 정책에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 Application Auto Scaling에 대한 대상 추적 조정 정책을 참조하십시오.

조정 정책을 생성하면 Amazon Keyspaces는 사용자를 대신하여 두 쌍의 Amazon CloudWatch 경보를 생성합니다. 각 쌍은 프로비저닝되고 사용된 처리량 설정의 상한값과 하한값을 나타냅니다. 이러한 CloudWatch 경보는 테이블의 실제 사용률이 일정 기간 동안 목표 사용률에서 벗어날 때 트리거됩니다. CloudWatchAmazon에 대해 자세히 알아보려면 Amazon CloudWatch 사용 설명서를 참조하십시오.

CloudWatch 알람 중 하나가 트리거되면 Amazon Simple Notification Service (Amazon SNS) 에서 알림을 보냅니다 (활성화한 경우). 그러면 CloudWatch 경보가 Application Auto Scaling을 호출하여 조정

정책을 평가합니다. 그러면 Amazon Keyspaces에 Alt Table 요청이 전송되어 테이블의 프로비저닝된 용량을 적절히 확장하거나 축소합니다. Amazon SNS 알림에 대한 자세한 내용은 Amazon SNS 알림 설정을 참조하세요.

Amazon Keyspaces는 테이블의 프로비저닝된 처리량 용량이 목표 사용률에 근접하도록 동적으로 확장하거나 축소함으로써 Alter Table 요청을 처리합니다.

> **Note**
>
> Amazon Keyspaces Auto Scaling은 실제 워크로드가 몇 분 동안 지속적으로 상승 (또는 감소) 상태를 유지할 때만 프로비저닝된 처리량 설정을 수정합니다. 의 목표 추적 알고리즘은 목표 사용률을 장기적으로 사용자가 선택한 값 안팎으로 유지되도록 합니다. 짧은 기간 동안 갑자기 급증하는 활동은 테이블에 기본 제공되는 버스트 용량으로 처리합니다.

멀티 리전 테이블에서 Auto Scaling이 작동하는 방식

프로비저닝된 용량 모드에서 모든 멀티 리전 테이블의 모든 테이블 복제본에 대해 항상 충분한 읽기 및 쓰기 용량을 확보하려면 Amazon Keyspaces Auto Scaling을 구성하는 것이 좋습니다. AWS 리전

Auto Scaling과 함께 프로비저닝 모드에서 멀티 리전 테이블을 사용하는 경우 단일 테이블 복제본에 대해 Auto Scaling을 비활성화할 수 없습니다. 하지만 테이블의 읽기 전용 Auto Scaling 설정을 지역별로 조정할 수 있습니다. 예를 들어 테이블이 복제되는 각 지역에 대해 서로 다른 읽기 용량 및 읽기 Auto Scaling 설정을 지정할 수 있습니다.

지정된 리전의 테이블 복제본에 대해 구성한 읽기 Auto Scaling 설정이 테이블의 일반 Auto Scaling 설정을 덮어씁니다. 하지만 모든 지역에서 쓰기를 복제하기에 충분한 용량을 확보하려면 모든 테이블 복제본에서 쓰기 용량이 동기화된 상태로 유지되어야 합니다.

Amazon Keyspaces Auto Scaling은 해당 지역의 사용량을 AWS 리전 기반으로 각 테이블의 프로비저닝된 용량을 독립적으로 업데이트합니다. 따라서 Auto Scaling이 활성화된 경우 다중 지역 테이블에 대한 각 지역의 프로비저닝 용량이 다를 수 있습니다.

Amazon Keyspaces 콘솔, AWS CLI API 또는 CQL을 사용하여 멀티 리전 테이블 및 해당 복제본의 오토 스케일링 설정을 구성할 수 있습니다. 다중 지역 테이블의 Auto Scaling 설정을 생성하고 업데이트하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “다중 리전 복제 사용 방법”.

> **Note**
>
> 다중 리전 테이블에 자동 크기 조정을 사용하는 경우 항상 Amazon Keyspaces API 작업을 사용하여 자동 조정 설정을 구성해야 합니다. Application Auto Scaling API 작업을 직접 사용하여 Auto Scaling 설정을 구성하는 경우 다중 지역 테이블을 지정할 수 없습니다. AWS 리전 이로 인해 구성이 지원되지 않을 수 있습니다.

사용 노트

Amazon Keyspaces Auto Scaling을 사용하려면 먼저 다음 내용을 이해해야 합니다.

- Amazon Keyspaces Auto Scaling은 Auto Scaling 정책에 따라 필요한 만큼 자주 읽기 용량이나 쓰기 용량을 늘릴 수 있습니다. 모든 Amazon Keyspaces 할당량은 할당량에서 설명한 것처럼 효력이 유지됩니다.
- Amazon Keyspaces Auto Scaling은 프로비저닝된 처리량 설정을 사용자가 수동으로 수정하는 것도 허용합니다. 이러한 수동 조정은 조정 정책에 연결된 기존 CloudWatch 경보에는 영향을 미치지 않습니다.
- 콘솔을 사용하여 프로비저닝된 처리량 용량이 있는 테이블을 생성하는 경우 Amazon Keyspaces Auto Scaling이 기본적으로 활성화됩니다. 언제든지 Auto Scaling 설정을 변경할 수 있습니다. 자세한 정보는 the section called “콘솔 사용”을 참조하세요.
- 를 AWS CloudFormation 사용하여 조정 정책을 생성하는 경우 스택이 스택 템플릿과 AWS CloudFormation 동기화되도록 조정 정책을 관리해야 합니다. Amazon Keyspace에서 조정 정책을 변경하면 스택이 재설정될 때 AWS CloudFormation 스택 템플릿의 원래 값으로 해당 정책을 덮어씁니다.
- Amazon Keyspaces 자동 크기 조정을 모니터링하는 CloudTrail 데 사용하는 경우 구성 검증 프로세스의 일부로 Application Auto Scaling에서 이루어진 호출에 대한 알림이 표시될 수 있습니다. 이러한 알림을 필터링하려면 이러한 유효성 검사를 위한 `application-autoscaling.amazonaws.com`을 포함하는 `invokedBy` 필드를 사용합니다.

콘솔을 통한 Amazon Keyspaces Auto Scaling 정책 관리

콘솔을 사용하여 새 테이블과 기존 테이블에 대해 Amazon Keyspaces Auto Scaling을 활성화할 수 있습니다. 또한 콘솔을 사용하여 Auto Scaling 설정을 수정하거나 Auto Scaling을 비활성화할 수 있습니다.

> **Note**
>
> 스케일 인 및 스케일 아웃 휴지 시간 설정과 같은 고급 기능을 사용하려면 CQL 또는 the () 를 사용하여 Amazon Keyspaces 조정 정책을 AWS Command Line Interface 프로그래밍 방식으로 AWS CLI 관리하십시오. 자세한 내용은 카산드라 쿼리 언어 (CQL) 를 사용한 Amazon Keyspaces 오토 스케일링 관리 또는 CLI를 사용한 Amazon Keyspaces 규모 조정 정책 관리를 참조하세요.

주제

시작하기 전에: 사용자에게 Amazon Keyspaces Auto Scaling에 대한 권한 부여

시작하려면 사용자에게 자동 규모 조정 설정을 생성하고 관리할 수 있는 적절한 권한이 있는지 확인합니다. AWS Identity and Access Management (IAM) 에서는 Amazon Keyspaces 조정 정책을 AWS `AmazonKeyspacesFullAccess` 관리하려면 관리형 정책이 필요합니다.

> **Important**
>
> 테이블에서 자동 크기 조정을 비활성화하려면 `application-autoscaling:*` 권한이 필요합니다. 테이블을 삭제하려면 먼저 테이블의 Auto Scaling을 해제해야 합니다.

Amazon Keyspaces 콘솔 액세스 및 Amazon Keyspaces Auto Scaling을 위해 IAM 사용자를 설정하려면 다음 정책을 추가합니다.

`AmazonKeyspacesFullAccess` 정책 연결

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/iam/ 에서 IAM 콘솔을 엽니다.
2. IAM 콘솔 대시보드에서 Users(사용자)를 선택한 후 목록에서 해당 IAM 사용자를 선택합니다.

3. 요약 페이지에서 권한 추가를 선택합니다.
4. 기존 정책 직접 첨부를 선택합니다.
5. 정책 목록에서 원하는 AmazonKeyspacesFullAccess정책을 선택한 후 다음: 검토를 선택합니다.
6. 권한 추가를 선택합니다.

Amazon Keyspaces Auto Scaling이 활성화된 새 테이블 만들기

> **Note**
>
> Amazon Keyspaces Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 설명은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

Auto Scaling이 활성화된 새 테이블 만들기

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 탐색 창에서 테이블을 선택한 다음 테이블 생성을 선택합니다.
3. 테이블 세부 정보 섹션의 테이블 생성 페이지에서 키스페이스를 선택하고 새 테이블의 이름을 입력합니다.
4. 열 섹션에서 테이블의 스키마를 생성합니다.
5. 기본 키 섹션에서 테이블의 기본 키를 정의하고 선택적 클러스터링 열을 선택합니다.
6. 테이블 설정 섹션에서 설정 사용자 지정을 선택합니다.
7. 읽기/쓰기 용량 설정을 계속합니다.
8. 용량 모드에서 프로비저닝됨을 선택합니다.
9. 읽기 용량 섹션에서 자동 규모 조정이 선택되어 있는지 확인합니다.

   이 단계에서는 테이블의 최소 및 최대 읽기 용량 단위와 목표 사용률을 선택합니다.

   - 최소 용량 단위 - 테이블이 항상 지원할 준비가 되어 있어야 하는 최소 처리량 수준 값을 입력합니다. 값은 1에서 계정에 대해 초당 최대 처리량 할당량(기본값 40,000) 사이여야 합니다.
   - 최대 용량 단위 - 테이블에 프로비저닝하려는 최대 처리량을 입력합니다. 값은 1에서 계정에 대해 초당 최대 처리량 할당량(기본값 40,000) 사이여야 합니다.

- 목표 사용률 — 목표 사용률을 20% ~ 90% 사이로 입력합니다. 트래픽이 정의된 목표 사용률을 초과하면 용량이 자동으로 스케일 업됩니다. 트래픽이 정의된 목표 미만으로 떨어지면 자동으로 다시 스케일 다운됩니다.

> **Note**
>
> 계정의 기본 할당량 및 할당량을 높이는 방법에 대한 자세한 내용은 할당량 섹션을 참조하세요.

10. 쓰기 용량 섹션에서 읽기 용량에 대해 이전 단계에서 정의한 것과 동일한 설정을 선택하거나 용량 값을 수동으로 구성합니다.
11. 테이블 생성을 선택합니다. Auto Scaling 파라미터로 테이블이 생성됩니다.

기존 테이블에서 Amazon Keyspaces Auto Scaling 활성화

> **Note**
>
> Amazon Keyspaces Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 설명은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

기존 테이블에 Amazon Keyspaces Auto Scaling 활성화

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 작업하려는 테이블을 선택하고 용량 탭으로 이동합니다.
3. 용량 설정 섹션에서 편집을 선택합니다.
4. 용량 모드에서 테이블이 프로비저닝된 용량 모드를 사용하고 있는지 확인합니다.
5. 자동 규모 조정을 선택하고 Amazon Keyspaces Auto Scaling이 활성화된 새 테이블 만들기의 6단계를 참조하여 읽기 및 쓰기 용량을 편집합니다.
6. Auto Scaling 설정이 정의되면 저장을 선택합니다.

Amazon Keyspaces Auto Scaling 설정 수정 또는 비활성화

를 사용하여 Amazon Keyspaces 자동 조정 설정을 수정할 수 있습니다. AWS Management Console 이렇게 하려면 편집하려는 테이블을 선택하고 용량 탭으로 이동하십시오. 용량 설정 섹션에서 편집을 선택합니다. 이제 읽기 용량 또는 쓰기 용량 섹션에서 설정을 수정할 수 있습니다. 이러한 설정에 대한 자세한 내용은 Amazon Keyspaces Auto Scaling이 활성화된 새 테이블 만들기 섹션을 참조하세요.

Amazon Keyspaces 자동 크기 조정을 비활성화하려면 자동 크기 조정 확인란의 선택을 취소하십시오. 자동 스케일링을 비활성화하면 테이블이 Application Auto Scaling에서 확장 가능한 타겟으로 등록 취소됩니다. Application Auto Scaling이 Amazon Keyspaces 테이블에 액세스하는 데 사용하는 서비스 연결 역할을 삭제하려면 the section called “Amazon Keyspaces에 대한 서비스 연결 역할 삭제”의 단계를 따릅니다.

> **Note**
>
> Application Auto Scaling에서 사용하는 서비스 연결 역할을 삭제하려면 계정 내 모든 테이블에서 전체 테이블의 자동 크기 조정을 비활성화해야 합니다. AWS 리전

콘솔에서 Amazon Keyspaces Auto Scaling 활동 보기

사용량 및 성능에 대한 지표를 생성하는 Amazon을 사용하여 Amazon CloudWatch Keyspaces 자동 크기 조정이 리소스를 어떻게 사용하는지 모니터링할 수 있습니다. Application Auto Scaling 사용 설명서의 단계에 따라 CloudWatch 대시보드를 생성하십시오.

카산드라 쿼리 언어 (CQL) 를 사용한 Amazon Keyspaces 오토 스케일링 관리

카산드라 쿼리 언어 (CQL) 를 사용하여 Amazon Keyspaces 테이블에 대한 오토 스케일링 설정을 생성하고 관리하려면 다음을 사용할 수 있습니다. `cqlsh` 이 항목에서는 CQL을 사용하여 프로그래밍 방식으로 관리할 수 있는 Auto Scaling 작업에 대한 개요를 제공합니다.

이 항목에 설명된 CQL 문에 대한 자세한 내용은 을 참조하십시오. the section called “DDL 문”

주제

- 시작하기 전 준비 사항
- CQL을 사용하여 자동 크기 조정이 가능한 새 테이블을 생성합니다.
- CQL을 사용하여 기존 테이블의 자동 크기 조정을 활성화합니다.

- CQL을 사용하여 테이블의 Amazon Keyspaces 자동 스케일링 구성을 확인하십시오.
- CQL을 사용하여 테이블에 대한 Amazon Keyspaces 자동 스케일링을 비활성화합니다.

시작하기 전 준비 사항

시작하기 전에 다음 작업을 수행해야 합니다.

권한 구성

사용자가 Auto Scaling 설정을 생성하고 관리하는 데 적절한 권한을 구성하지 않았다면 이를 수행해야 합니다. AWS Identity and Access Management (IAM) 에서 Amazon Keyspaces 조정 정책을 AWS `AmazonKeyspacesFullAccess` 관리하려면 관리형 정책이 필요합니다. 자세한 단계는 the section called “시작하기 전에: 사용자에게 Amazon Keyspaces Auto Scaling에 대한 권한 부여” 단원을 참조하세요.

`cqlsh` 구성

아직 설치하지 않았다면 설치하고 구성해야 합니다. `cqlsh` 이렇게 하려면 의 지침을 따르십시오the section called “`cqlsh-expansion` 사용”. 그런 다음 AWS CloudShell 를 사용하여 다음 섹션의 명령을 실행할 수 있습니다.

CQL을 사용하여 자동 크기 조정이 가능한 새 테이블을 생성합니다.

새 Amazon Keyspaces 테이블을 생성할 때 명령문에서 테이블의 쓰기 또는 읽기 용량에 대해 Auto Scaling을 자동으로 활성화할 수 있습니다. `CREATE TABLE` 이렇게 하면 Amazon Keyspace가 사용자를 대신하여 Application Auto Scaling에 연락하여 테이블을 확장 가능한 대상으로 등록하고 프로비저닝된 쓰기 또는 읽기 용량을 조정할 수 있습니다.

다중 리전 테이블을 생성하고 테이블 복제본에 대해 다양한 Auto Scaling 설정을 구성하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “기본 설정 (CQL) 이 포함된 다중 지역 테이블 만들기”

> **Note**
>
> Amazon Keyspaces Auto Scaling을 사용하려면 사용자를 대신하여 자동 조정 작업을 수행할 서비스 연결 역할 (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`) 이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 설명은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

테이블에 대한 Auto Scaling 설정을 프로그래밍 방식으로 구성하려면 Amazon Keyspaces Auto Scaling의 파라미터가 포함된 AUTOSCALING_SETTINGS 명령문을 사용하십시오. 파라미터는 Amazon Keyspaces가 테이블의 프로비저닝된 처리량을 조정하도록 지시하는 조건과 취해야 할 추가 선택적 조치를 정의합니다. 이 예시에서는 mytable의 자동 스케일링 설정을 정의합니다.

정책에는 다음 요소가 포함됩니다.

- AUTOSCALING_SETTINGS— Amazon Keyspace가 사용자를 대신하여 처리 용량을 조정할 수 있는지 여부를 지정합니다. 다음 값이 필요합니다.
  - provisioned_write_capacity_autoscaling_update:
    - minimum_units
    - maximum_units
  - provisioned_read_capacity_autoscaling_update:
    - minimum_units
    - maximum_units
  - scaling_policy— Amazon Keyspace는 대상 추적 정책을 지원합니다. 대상 추적 정책을 정의하려면 다음 파라미터를 구성합니다.
    - target_value— Amazon Keyspaces Auto Scaling은 프로비저닝된 용량에 대한 소비 용량의 비율이 이 값 또는 그 근방을 유지하도록 보장합니다. target_value를 백분율로 지정합니다.
    - disableScaleIn: (선택 사항) 테이블의 비활성화 또는 활성화 scale-in 여부를 boolean 지정하는 A. 이 매개변수는 기본적으로 비활성화되어 있습니다. scale-in켜려면 boolean 값을 로 설정합니다FALSE. 즉, 사용자 대신 테이블의 용량이 자동으로 축소됩니다.
    - scale_out_cooldown – 스케일 아웃 활동은 테이블의 프로비저닝된 처리량을 늘립니다. 스케일 아웃 활동을 위한 휴지 기간을 추가하려면 scale_out_cooldown에 대한 값(초)을 지정합니다. 값을 지정하지 않는 경우 기본값은 0입니다. 대상 추적 및 휴지 기간에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 대상 추적 조정 정책을 참조하십시오.
    - scale_in_cooldown – 스케일 인 활동은 테이블의 프로비저닝된 처리량을 줄입니다. 스케일 인 활동을 위한 휴지 기간을 추가하려면 scale_in_cooldown에 대한 값(초)을 지정합니다. 값을 지정하지 않는 경우 기본값은 0입니다. 대상 추적 및 휴지 기간에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 대상 추적 조정 정책을 참조하십시오.

> **Note**
>
> target_value의 용도를 자세히 이해하기 위해 쓰기 용량 단위 200으로 할당 처리량을 설정한 테이블을 예로 들어 보겠습니다. 이 테이블에 대해 target_value가 70퍼센트인 확장 정책을 만들려고 합니다.
> 테이블에 대한 쓰기 트래픽을 시작했더니 실제 쓰기 처리량이 150 용량 단위였다고 가정해 보겠습니다. 현재 consumed-to-provisioned 비율은 (150/200), 즉 75% 입니다. 이 비율이 목표를 초과하므로 Auto Scaling은 프로비저닝된 쓰기 용량을 215로 늘려 비율을 (150/ 215), 즉 69.77% 로 늘립니다. 즉, 비율과 최대한 비슷하지만 초과하지는 않습니다. target_value

mytable의 경우 읽기 및 쓰기 용량을 모두 50% 로 TargetValue 설정했습니다. Amazon Keyspaces Auto Scaling은 테이블의 프로비저닝 처리량을 용량 단위 5~10개 범위 내에서 조정하여 consumed-to-provisioned 비율이 50% 또는 그 근방을 유지하도록 합니다. 읽기 용량의 경우 값을 1~60초로 설정합니다. ScaleOutCooldown ScaleInCooldown

다음 명령문을 사용하여 자동 크기 조정이 활성화된 새 Amazon Keyspaces 테이블을 생성할 수 있습니다.

```
CREATE TABLE mykeyspace.mytable(pk int, ck int, PRIMARY KEY (pk, ck))
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 1,
        'write_capacity_units': 1
    }
} AND AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50
            }
        }
    },
    'provisioned_read_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
```

```
                'target_value': 50,
                'scale_in_cooldown': 60,
                'scale_out_cooldown': 60
            }
        }
    }
};
```

CQL을 사용하여 기존 테이블의 자동 크기 조정을 활성화합니다.

기존 Amazon Keyspaces 테이블의 경우 명령문을 사용하여 테이블의 쓰기 또는 읽기 용량에 대해 Auto Scaling을 활성화할 수 있습니다. ALTER TABLE 현재 온디맨드 용량 모드에 있는 테이블을 업데이트하는 경우 필요한 용량보다 더 capacity_mode 높습니다. 테이블이 이미 프로비저닝된 용량 모드에 있는 경우 이 필드를 생략할 수 있습니다.

> **Note**
>
> Amazon Keyspaces Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(AWSServiceRoleForApplicationAutoScaling_CassandraTable)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 설명은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

다음 예제에서 명령문은 온디맨드 용량 모드에 있는 mytable 테이블을 업데이트합니다. 명령문은 테이블의 용량 모드를 Auto Scaling이 활성화된 프로비저닝 모드로 변경합니다.

쓰기 용량은 5~10개 용량 단위 범위 내에서 구성되며, 목표 값은 50% 입니다. 읽기 용량도 5~10개 용량 단위 범위 내에서 구성되며, 목표값은 50% 입니다. 읽기 용량의 경우 값을 scale_out_cooldown scale_in_cooldown 1~60초로 설정합니다.

```
ALTER TABLE mykeyspace.mytable
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 1,
        'write_capacity_units': 1
    }
} AND AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
```

```
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50
            }
        }
    },
    'provisioned_read_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50,
                'scale_in_cooldown': 60,
                'scale_out_cooldown': 60
            }
        }
    }
};
```

CQL을 사용하여 테이블의 Amazon Keyspaces 자동 스케일링 구성을 확인하십시오.

테이블의 Auto Scaling 구성 세부 정보를 보려면 다음 명령을 사용하십시오.

```
SELECT * FROM system_schema_mcs.autoscaling WHERE keyspace_name = 'mykeyspace' AND
 table_name = 'mytable';
```

이 명령의 출력은 다음과 같습니다.

```
 keyspace_name | table_name | provisioned_read_capacity_autoscaling_update

                                                                                |
 provisioned_write_capacity_autoscaling_update
--------------+------------
+-------------------------------------------------------------------------------------------------
+-------------------------------------------------------------------------------------------------
 mykeyspace    | mytable    | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 60, 'disable_scale_in': false, 'target_value':
 50, 'scale_in_cooldown': 60}}} | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 0, 'disable_scale_in': false, 'target_value': 50,
 'scale_in_cooldown': 0}}}
```

CQL을 사용하여 테이블에 대한 Amazon Keyspaces 자동 스케일링을 비활성화합니다.

언제든지 테이블의 Amazon Keyspaces 자동 크기 조정을 해제할 수 있습니다. 더 이상 테이블의 읽기 또는 쓰기 용량을 확장할 필요가 없는 경우 Amazon Keyspaces가 테이블의 읽기 또는 쓰기 용량 설정을 계속 수정하지 않도록 Auto Scaling을 끄는 것을 고려해야 합니다. 명령문을 사용하여 테이블을 업데이트할 수 있습니다. `ALTER TABLE`

다음 명령문은 mytable 테이블의 쓰기 용량에 대한 자동 스케일링을 끕니다. 또한 사용자 대신 생성된 CloudWatch 경보도 삭제됩니다.

```
ALTER TABLE mykeyspace.mytable
WITH AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'autoscaling_disabled': true
    }
};
```

> **Note**
>
> Application Auto Scaling에서 사용하는 서비스 연결 역할을 삭제하려면 계정 내 모든 테이블에서 전체 테이블의 자동 크기 조정을 비활성화해야 합니다. AWS 리전

CLI를 사용한 Amazon Keyspaces 규모 조정 정책 관리

Amazon Keyspaces Auto Scaling 설정을 프로그래밍 방식으로 업데이트하고 관리하려면 AWS Command Line Interface (AWS CLI) 또는 API를 사용할 수 있습니다. AWS 카산드라 쿼리 언어 (CQL)를 사용하여 Amazon Keyspaces 자동 조정 정책을 관리하려면 을 참조하십시오. the section called “CQL 사용” 이 항목에서는 를 사용하여 프로그래밍 방식으로 관리할 수 있는 Auto Scaling 작업에 대한 개요를 제공합니다. AWS CLI

이 주제에 설명된 Amazon Keyspaces AWS CLI 명령에 대한 자세한 내용은 AWS CLI 명령 참조를 참조하십시오.

주제

- 시작하기 전 준비 사항
- 를 사용하여 자동 크기 조정이 포함된 새 테이블을 생성하십시오. AWS CLI
- 를 사용하여 기존 테이블의 자동 크기 조정을 활성화합니다. AWS CLI
- 다음을 사용하여 테이블의 Amazon Keyspaces 자동 스케일링 구성을 확인하십시오. AWS CLI

- 다음을 사용하여 테이블에 대한 Amazon Keyspaces 자동 크기 조정을 끕니다. AWS CLI

시작하기 전 준비 사항

시작하기 전에 다음 작업을 수행해야 합니다.

권한 구성

사용자가 Auto Scaling 설정을 생성하고 관리하는 데 적절한 권한을 구성하지 않았다면 이를 수행해야 합니다. AWS Identity and Access Management (IAM) 에서 Amazon Keyspaces 조정 정책을 AWS `AmazonKeyspacesFullAccess` 관리하려면 관리형 정책이 필요합니다. 자세한 단계는 the section called “시작하기 전에: 사용자에게 Amazon Keyspaces Auto Scaling에 대한 권한 부여” 단원을 참조하세요.

AWS CLI 설치

AWS CLI를 아직 설치하지 않았다면 이를 설치하고 구성해야 합니다. 이렇게 하려면 AWS Command Line Interface 사용 설명서로 이동하여 다음 지침을 따르십시오.

- AWS CLI 설치
- AWS CLI 구성

를 사용하여 자동 크기 조정이 포함된 새 테이블을 생성하십시오. AWS CLI

새 Amazon Keyspaces 테이블을 생성할 때 작업에서 테이블의 쓰기 또는 읽기 용량에 대해 Auto Scaling을 자동으로 활성화할 수 있습니다. `CreateTable` 이렇게 하면 Amazon Keyspace가 사용자를 대신하여 Application Auto Scaling에 연락하여 확장 가능한 대상으로 지정한 테이블을 등록하고 프로비저닝된 쓰기 또는 읽기 용량을 조정할 수 있습니다.

Auto Scaling 구성을 사용하여 다중 지역 테이블을 생성하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “Auto Scaling (CLI) 을 사용하여 프로비저닝 모드에서 새 멀티 리전 테이블 생성”.

> **Note**
>
> Amazon Keyspaces Auto Scaling을 사용하려면 사용자를 대신하여 자동 조정 작업을 수행할 서비스 연결 역할 (`AWSServiceRoleForApplicationAutoScaling_CassandraTable`) 이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 설명은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

테이블에 대한 Auto Scaling 설정을 프로그래밍 방식으로 구성하려면 Amazon Keyspaces Auto Scaling의 파라미터를 정의하는 autoScalingSpecification 작업을 사용하십시오. 파라미터는 Amazon Keyspaces가 테이블의 프로비저닝된 처리량을 조정하도록 지시하는 조건과 취해야 할 추가 선택적 조치를 정의합니다. 이 예시에서는 mytable의 자동 스케일링 설정을 정의합니다.

정책에는 다음 요소가 포함됩니다.

- autoScalingSpecification— Amazon Keyspace가 사용자를 대신하여 용량 처리량을 조정할 수 있는지 여부를 지정합니다. 읽기 용량과 쓰기 용량에 대해 각각 Auto Scaling을 활성화할 수 있습니다. 그런 다음 다음 파라미터를 지정해야 합니다autoScalingSpecification.
  - writeCapacityAutoScaling— 최대 및 최소 쓰기 용량 단위.
  - readCapacityAutoScaling— 최대 및 최소 읽기 용량 단위.
  - scalingPolicy— Amazon Keyspace는 대상 추적 정책을 지원합니다. 대상 추적 정책을 정의하려면 다음 파라미터를 구성합니다.
    - targetValue— Amazon Keyspaces Auto Scaling은 프로비저닝된 용량에 대한 소비 용량의 비율이 이 값 또는 그 근방을 유지하도록 보장합니다. targetValue를 백분율로 지정합니다.
    - disableScaleIn: (선택 사항) 테이블의 비활성화 또는 활성화 scale-in 여부를 boolean 지정하는 A. 이 매개변수는 기본적으로 비활성화되어 있습니다. scale-in켜려면 boolean 값을 로 설정합니다FALSE. 즉, 사용자 대신 테이블의 용량이 자동으로 축소됩니다.
    - scaleOutCooldown – 스케일 아웃 활동은 테이블의 프로비저닝된 처리량을 늘립니다. 스케일 아웃 활동을 위한 휴지 기간을 추가하려면 ScaleOutCooldown에 대한 값(초)을 지정합니다. 기본값은 0입니다. 대상 추적 및 휴지 기간에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 대상 추적 조정 정책을 참조하십시오.
    - scaleInCooldown – 스케일 인 활동은 테이블의 프로비저닝된 처리량을 줄입니다. 스케일 인 활동을 위한 휴지 기간을 추가하려면 ScaleInCooldown에 대한 값(초)을 지정합니다. 기본값은 0입니다. 대상 추적 및 휴지 기간에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 대상 추적 조정 정책을 참조하십시오.

> **Note**
>
> TargetValue의 용도를 자세히 이해하기 위해 쓰기 용량 단위 200으로 할당 처리량을 설정한 테이블을 예로 들어 보겠습니다. 이 테이블에 대해 TargetValue가 70퍼센트인 확장 정책을 만들려고 합니다.
> 테이블에 대한 쓰기 트래픽을 시작했더니 실제 쓰기 처리량이 150 용량 단위였다고 가정해 보겠습니다. 현재 consumed-to-provisioned 비율은 (150/200), 즉 75% 입니다. 이 비율이 목표

를 초과하므로 Auto Scaling은 프로비저닝된 쓰기 용량을 215로 늘려 비율을 (150/ 215), 즉 69.77% 로 늘립니다. 즉, 비율과 최대한 비슷하지만 초과하지는 않습니다. `TargetValue`

mytable의 경우 읽기 및 쓰기 용량을 모두 50% 로 `TargetValue` 설정했습니다. Amazon Keyspaces Auto Scaling은 테이블의 프로비저닝된 처리량을 용량 단위 5~10개 범위 내에서 조정하여 consumed-to-provisioned 비율이 50% 또는 그 근방을 유지하도록 합니다. 읽기 용량의 경우 값을 1~60초로 설정합니다. `ScaleOutCooldown` `ScaleInCooldown`

복잡한 Auto Scaling 설정이 포함된 테이블을 생성할 때는 JSON 파일에서 Auto Scaling 설정을 로드하는 것이 좋습니다. 다음 예제의 경우 auto-scaling.zip 에서 예제 JSON 파일을 다운로드하고 파일 경로를 기록하여 `auto-scaling.json` 압축을 풀 수 있습니다. 이 예제에서 JSON 파일은 현재 디렉터리에 있습니다. 다양한 파일 경로 옵션에 대해서는 파일에서 매개변수를 로드하는 방법을 참조하세요.

```
aws keyspaces create-table --keyspace-name mykeyspace --table-name mytable
            \ --schema-definition 'allColumns=[{name=pk,type=int},
{name=ck,type=int}],partitionKeys=[{name=pk},{name=ck}]'
            \ --capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1
            \ --auto-scaling-specification file://auto-scaling.json
```

를 사용하여 기존 테이블의 자동 크기 조정을 활성화합니다. AWS CLI

기존 Amazon Keyspaces 테이블의 경우 작업을 사용하여 테이블의 쓰기 또는 읽기 용량에 대해 Auto Scaling을 활성화할 수 있습니다. `UpdateTable` 다중 지역 테이블의 Auto Scaling 설정을 업데이트하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “다중 지역 테이블 (CLI) 의 프로비저닝된 용량 및 Auto Scaling 설정 업데이트”.

> **Note**
>
> Amazon Keyspaces Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 설명은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

다음 명령을 사용하여 기존 테이블에 대해 Amazon Keyspaces 자동 크기 조정을 활성화할 수 있습니다. 테이블의 Auto Scaling 설정은 JSON 파일에서 로드됩니다. 다음 예제의 경우 auto-scaling.zip 에서

예제 JSON 파일을 다운로드하고 파일 경로를 기록하여 auto-scaling.json 압축을 풀 수 있습니다. 이 예제에서 JSON 파일은 현재 디렉터리에 있습니다. 다양한 파일 경로 옵션에 대해서는 파일에서 매개변수를 로드하는 방법을 참조하세요.

다음 예제에 사용된 Auto Scaling 설정에 대한 자세한 내용은 을 참조하십시오the section called “를 사용하여 자동 크기 조정이 포함된 새 테이블을 생성하십시오. AWS CLI”.

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
            \ --capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1
            \ --auto-scaling-specification file://auto-scaling.json
```

다음을 사용하여 테이블의 Amazon Keyspaces 자동 스케일링 구성을 확인하십시오. AWS CLI

테이블의 Auto Scaling 구성을 보려면 get-table-auto-scaling-settings 작업을 사용할 수 있습니다. 다음 CLI 명령은 이에 대한 예입니다.

```
aws keyspaces get-table-auto-scaling-settings --keyspace-name mykeyspace --table-name
 mytable
```

이 명령의 출력은 다음과 같습니다.

```
{
    "keyspaceName": "mykeyspace",
    "tableName": "mytable",
    "resourceArn": "arn:aws:cassandra:us-east-1:5555-5555-5555:/keyspace/mykeyspace/
table/mytable",
    "autoScalingSpecification": {
        "writeCapacityAutoScaling": {
            "autoScalingDisabled": false,
            "minimumUnits": 5,
            "maximumUnits": 10,
            "scalingPolicy": {
                "targetTrackingScalingPolicyConfiguration": {
                    "disableScaleIn": false,
                    "scaleInCooldown": 0,
                    "scaleOutCooldown": 0,
                    "targetValue": 50.0
                }
```

```
            }
        },
        "readCapacityAutoScaling": {
            "autoScalingDisabled": false,
            "minimumUnits": 5,
            "maximumUnits": 10,
            "scalingPolicy": {
                "targetTrackingScalingPolicyConfiguration": {
                    "disableScaleIn": false,
                    "scaleInCooldown": 60,
                    "scaleOutCooldown": 60,
                    "targetValue": 50.0
                }
            }
        }
    }
}
```

다음을 사용하여 테이블에 대한 Amazon Keyspaces 자동 크기 조정을 끕니다. AWS CLI

언제든지 테이블의 Amazon Keyspaces 자동 크기 조정을 해제할 수 있습니다. 더 이상 테이블의 읽기 또는 쓰기 용량을 확장할 필요가 없는 경우 Amazon Keyspaces가 테이블의 읽기 또는 쓰기 용량 설정을 계속 수정하지 않도록 Auto Scaling을 끄는 것을 고려해야 합니다. 작업을 통해 테이블을 업데이트할 수 있습니다. `UpdateTable`

다음 명령은 테이블의 읽기 용량에 대한 Auto Scaling을 끕니다. 또한 사용자를 대신하여 생성된 CloudWatch 경보도 삭제됩니다.

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
            \ --auto-scaling-specification
 readCapacityAutoScaling={autoScalingDisabled=true}
```

> **Note**
>
> Application Auto Scaling에서 사용하는 서비스 연결 역할을 삭제하려면 계정 내 모든 테이블에서 전체 테이블의 자동 크기 조정을 비활성화해야 합니다. AWS 리전

Amazon Keyspaces에서 버스트 용량을 효과적으로 사용하기

Amazon Keyspaces는 버스트 용량을 제공해 파티션당 처리량 프로비저닝에서 어느 정도 유연성을 제공합니다. 파티션의 처리량을 완전히 사용하지 않을 때마다 Amazon Keyspaces는 나중에 사용량 급증을 처리하기 위해 처리량 버스트에 사용하지 않은 용량 일부를 예약해 둡니다.

Amazon Keyspaces는 현재 5분(300초)에 해당하는 미사용 읽기 및 쓰기 용량을 비축합니다. 가끔 발생하는 읽기 또는 쓰기 작업의 버스트 중 이러한 추가 용량 단위는 빠르게, 테이블에 대해 정의한 초당 프로비저닝된 처리량 용량보다도 빠르게 소모될 수 있습니다.

Amazon Keyspaces는 또한 사전 통지 없이 배경 유지 관리와 다른 작업에 버스트 용량을 사용할 수도 있습니다.

향후 버스트 용량의 세부 정보가 변경될 수도 있습니다.

Amazon Keyspace의 용량 사용량을 추정하는 방법

Amazon Keyspaces에서 데이터를 읽거나 쓸 때 쿼리에서 소비하는 읽기/쓰기 요청 단위 (RRU/WRU) 또는 읽기/쓰기 용량 단위 (RCU/WCU) 의 양은 Amazon Keyspaces가 쿼리를 실행하기 위해 처리해야 하는 총 데이터 양에 따라 달라집니다. 경우에 따라 Amazon Keyspace가 쿼리를 처리하기 위해 읽어야 했던 데이터의 하위 집합이 클라이언트에 반환되는 데이터가 될 수 있습니다. 조건부 쓰기의 경우 조건부 검사가 실패하더라도 Amazon Keyspaces는 쓰기 용량을 소비합니다.

요청에 대해 처리되는 총 데이터 양을 추정하려면 인코딩된 행 크기와 총 행 수를 고려해야 합니다. 이 주제에서는 Amazon Keyspaces가 쿼리를 처리하는 방식과 이것이 용량 소비에 미치는 영향을 보여주기 위해 일반적인 시나리오와 액세스 패턴의 몇 가지 예를 다룹니다. 예제를 따라 테이블의 용량 요구 사항을 추정하고 Amazon을 사용하여 이러한 사용 사례의 읽기 및 쓰기 용량 사용량을 CloudWatch 관찰할 수 있습니다.

Amazon Keyspace의 인코딩된 행 크기를 계산하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “행 크기 계산”

주제

- 간단한 트랜잭션
- Amazon의 읽기 및 쓰기 용량 사용량 추정 CloudWatch

범위 쿼리

범위 쿼리의 읽기 용량 사용량을 살펴보기 위해 온디맨드 용량 모드를 사용하는 다음 예제 표를 사용합니다.

```
pk1 | pk2 | pk3 | ck1 | ck2 | ck3 | value
-----+-----+-----+-----+-----+-----+-------
a | b | 1 | a | b | 50 | <any value that results in a row size larger than 4KB>
a | b | 1 | a | b | 60 | value_1
a | b | 1 | a | b | 70 | <any value that results in a row size larger than 4KB>
```

이제 이 테이블에서 다음 쿼리를 실행합니다.

```
SELECT * FROM amazon_keyspaces.example_table_1 WHERE pk1='a' AND pk2='b' AND pk3=1 AND
 ck1='a' AND ck2='b' AND ck3 > 50 AND ck3 < 70;
```

쿼리에서 다음 결과 세트를 수신하고 Amazon Keyspace에서 수행하는 읽기 작업은 정합성 보장 모드에서 RRU 2개를 소비합니다. LOCAL_QUORUM

```
pk1 | pk2 | pk3 | ck1 | ck2 | ck3 | value
-----+-----+-----+-----+-----+-----+-------
a | b | 1 | a | b | 60 | value_1
```

Amazon Keyspace는 2개의 RRU를 사용하여 값이 있는 행을 평가하고 쿼리를 ck3=60 처리합니다ck3=70. 하지만 Amazon Keyspaces는 쿼리에 지정된 WHERE 조건이 참인 행, 즉 값이 있는 행만 반환합니다. ck3=60 쿼리에 지정된 범위를 평가하기 위해 Amazon Keyspace는 범위의 상한과 일치하는 행을 읽지만 (이 경우ck3 = 70) 결과에 해당 행을 반환하지는 않습니다. 읽기 용량 소비량은 반환된 데이터가 아니라 쿼리를 처리할 때 읽은 데이터를 기준으로 합니다.

쿼리 제한

LIMIT절을 사용하는 쿼리를 처리할 때 Amazon Keyspace는 쿼리에 지정된 조건을 일치시키려고 할 때 최대 페이지 크기까지 행을 읽습니다. Amazon Keyspaces에서 첫 페이지의 LIMIT 값을 충족하는 충분한 매칭 데이터를 찾을 수 없는 경우 페이지를 매긴 호출이 한 번 이상 필요할 수 있습니다. 다음 페이지에서 계속 읽으려면 페이지 매김 토큰을 사용할 수 있습니다. 기본 페이지 크기는 1MB입니다.

LIMIT조항을 사용할 때 읽기 용량을 적게 소비하려면 페이지 크기를 줄일 수 있습니다. 페이지 매김에 대한 자세한 내용은 the section called “결과 페이지 매김” 섹션을 참조하세요.

예를 들어 다음 쿼리를 살펴보겠습니다.

```
SELECT * FROM my_table WHERE partition_key=1234 LIMIT 1;”
```

페이지 크기를 설정하지 않으면 Amazon Keyspaces는 행 1개만 반환하더라도 1MB의 데이터를 읽습니다. Amazon Keyspace가 한 행만 읽도록 하려면 이 쿼리의 페이지 크기를 1로 설정하면 됩니다. 이 경우, Amazon Keyspaces는 T ime-to-live 설정 또는 클라이언트 측 타임스탬프를 기반으로 만료된 행이 없는 한 하나의 행만 읽습니다. 읽기 용량을 적게 사용하려면 페이지 크기를 LIMIT 값과 동일하게 설정하여 Amazon Keyspace가 읽는 데이터 양을 줄이는 것이 좋습니다.

테이블 스캔

ALLOW FILTERING옵션을 사용한 쿼리와 같이 전체 테이블 스캔이 발생하는 쿼리는 결과로 반환되는 것보다 더 많은 읽기를 처리하는 쿼리의 또 다른 예입니다. 또한 읽기 용량 소비는 반환된 데이터가 아니라 읽은 데이터를 기준으로 합니다.

테이블 스캔 예제에서는 온디맨드 용량 모드에서 다음 예제 테이블을 사용합니다.

```
pk | ck | value
---+----+---------
pk | 10 | <any value that results in a row size larger than 4KB>
pk | 20 | value_1
pk | 30 | <any value that results in a row size larger than 4KB>
```

Amazon Keyspace는 기본적으로 4개의 파티션이 있는 온디맨드 용량 모드에서 테이블을 생성합니다. 이 예제 테이블에서는 모든 데이터가 한 파티션에 저장되고 나머지 세 개의 파티션은 비어 있습니다.

이제 테이블에서 다음 쿼리를 실행합니다.

```
SELECT * from amazon_keyspaces.example_table_2;
```

이 쿼리로 인해 Amazon Keyspaces가 테이블의 4개 파티션을 모두 스캔하고 정합성 모드에서 6개의 RRU를 소비하는 테이블 스캔 작업이 발생합니다. LOCAL_QUORUM 먼저, Amazon Keyspace는 세 개의 행을 읽는 데 RRU 3개를 사용합니다. pk=‘pk’ 그러면 Amazon Keyspace는 테이블의 빈 파티션 3개를 스캔하는 데 추가로 RRU 3개를 사용합니다. 이 쿼리는 테이블 스캔으로 이어지기 때문에 Amazon Keyspaces는 데이터가 없는 파티션을 포함하여 테이블의 모든 파티션을 스캔합니다.

간단한 트랜잭션

경량 트랜잭션 (LWT) 을 사용하면 테이블 데이터에 대해 조건부 쓰기 작업을 수행할 수 있습니다. 조건부 업데이트 작업은 현재 상태를 평가하는 조건에 따라 레코드를 삽입, 업데이트 및 삭제할 때 유용합니다.

Amazon Keyspace에서는 모든 쓰기 작업에 LOCAL_QUORUM 일관성이 필요하며 LWT 사용에 따른 추가 비용은 없습니다. LWT의 차이점은 LWT 조건 검사 결과가 FALSE인 경우 쓰기 용량 단위를 소비한다는 것입니다. 소비되는 쓰기 용량 단위 수는 행 크기에 따라 달라집니다. 행 크기가 2KB인 경우 실패한 조건부 쓰기는 쓰기 용량 단위 2개를 소비합니다. 행이 현재 테이블에 없는 경우 작업은 쓰기 용량 단위 1개를 소비합니다. 의 `ConditionalCheckFailed` 메트릭을 모니터링하여 LWT 상태 검사 실패로 인해 소비된 용량을 CloudWatch 확인할 수 있습니다.

Amazon의 읽기 및 쓰기 용량 사용량 추정 CloudWatch

CloudWatch 대시보드를 사용하여 읽기 및 쓰기 용량 사용량을 추정하고 모니터링할 수 있습니다. Amazon Keyspace에 사용할 수 있는 지표에 대한 자세한 내용은 을 참조하십시오. the section called “지표 및 차원”

특정 명세서에서 소비한 읽기 및 쓰기 용량 단위를 모니터링하려면 다음 단계를 따르세요. CloudWatch

1. 샘플 데이터를 사용하여 새 테이블을 생성합니다.
2. 테이블에 대한 Amazon Keyspaces CloudWatch 대시보드를 구성합니다. 시작하려면 Github에서 제공하는 대시보드 템플릿을 사용할 수 있습니다.
3. 예를 들어 `ALLOW FILTERING` 옵션을 사용하여 CQL 문을 실행하고 대시보드에서 전체 테이블 스캔에 사용된 읽기 용량 단위를 확인합니다.

Amazon Keyspaces(Apache Cassandra용)의 키스페이스, 테이블 및 행으로 작업하기

이 장에서는 Amazon Keyspaces(Apache Cassandra용)의 키스페이스, 테이블, 행 등의 작업에 대한 세부 정보를 제공합니다. Amazon에서 키스페이스와 테이블을 모니터링하는 방법을 CloudWatch 알아보려면 을 참조하십시오the section called “를 통한 모니터링 CloudWatch”.

주제

- Amazon Keyspaces에서 키스페이스로 작업하기
- Amazon Keyspaces에서 테이블로 작업하기
- Amazon Keyspaces에서 행으로 작업하기
- Amazon Keyspaces에서 쿼리로 작업하기
- Amazon Keyspaces에서 파티셔너 사용하기
- Amazon Keyspaces 리소스의 태그 및 레이블을 사용한 작업

Amazon Keyspaces에서 키스페이스로 작업하기

이 섹션에서는 Amazon Keyspaces(Apache Cassandra용)의 키스페이스 작업에 대한 세부 정보를 제공합니다.

주제

- Amazon Keyspaces에서 시스템 키스페이스로 작업하기
- Amazon Keyspaces에서 키스페이스 생성

Amazon Keyspaces에서 시스템 키스페이스로 작업하기

Amazon Keyspaces는 네 가지 시스템 키스페이스를 사용합니다.

- `system`
- `system_schema`
- `system_schema_mcs`
- `system_multiregion_info`

다음 섹션에서는 Amazon Keyspace에서 지원되는 시스템 키스페이스 및 시스템 테이블에 대한 세부 정보를 제공합니다.

`system`

이것은 Cassandra 키스페이스입니다. Amazon Keyspaces는 다음 테이블을 사용합니다.

테이블 이름	열 이름	설명
`local`	`key, bootstrapped, broadcast_address, cluster_name, cql_version, data_center, gossip_generation, host_id, listen_address, native_protocol_version, partitioner, rack, release_version, rpc_address, schema_version, thrift_version, tokens, truncated_at`	로컬 키스페이스에 대한 정보
`peers`	`peer, data_center, host_id, preferred_ip, rack, release_version, rpc_address, schema_version, tokens`	이 테이블을 쿼리하여 사용 가능한 엔드포인트를 확인합니다. 예를 들어, 퍼블릭 엔드포인트를 통해 연결하는 경우 사용 가능한 IP 주소 9개의 목록이 표시됩니다. FIPS 엔드포인트를 통해 연결하는 경우 세 개의 IP 주소 목록이 표시됩니다. AWS PrivateLink VPC 엔드포인트를 통해 연결하는 경우 구성한 IP 주소 목록이 표시됩니다. 자세한 설명은 the section called “인터페이스 VPC 엔드

테이블 이름	열 이름	설명
		포인트 정보로 system.peers 테이블 항목 채우기" 섹션을 참조하세요.
size_estimates	keyspace_name, table_name, range_start, range_end, mean_partition_size, partitions_count	이 테이블은 모든 테이블의 각 토큰 범위에 대한 파티션의 총 크기와 수를 정의합니다. 이는 예상 파티션 크기를 사용하여 작업을 분배하는 Apache Cassandra Spark 커넥터에 필요합니다.
prepared_statements	prepared_id, logged_keyspace, query_string	이 테이블에는 저장된 쿼리에 대한 정보가 들어 있습니다.

system_schema

이것은 Cassandra 키스페이스입니다. Amazon Keyspaces는 다음 테이블을 사용합니다.

테이블 이름	열 이름	설명
keyspaces	keyspace_name, durable_writes, replication	특정 키스페이스에 대한 정보
tables	keyspace_name, table_name, bloom_filter_fp_chance, caching, comment, compaction, compression, crc_check_chance, dclocal_read_repair_chance, default_time_to_li	특정 테이블에 대한 정보

테이블 이름	열 이름	설명
	ve, extensions, flags, gc_grace_seconds, id, max_index_interval, memtable_flush_period_in_ms, min_index_interval, read_repair_chance, speculative_retry	
columns	keyspace_name, table_name, column_name, clustering_order, column_name_bytes, kind, position, type	특정 열에 대한 정보

system_schema_mcs

이것은 Amazon Keyspaces 특정 설정에 대한 정보를 저장하는 AWS Amazon Keyspaces 키스페이스입니다.

테이블 이름	열 이름	설명
keyspaces	keyspace_name, durable_writes, replication	이 테이블을 쿼리하여 프로그래밍 방식으로 키스페이스가 생성되었는지 확인할 수 있습니다. 자세한 설명은 the section called “키스페이스 생성” 섹션을 참조하세요.
tables	keyspace_name, creation_time, speculative_retry, cdc, gc_grace_	이 테이블을 쿼리하여 특정 테이블의 상태를 확인할 수 있습니다. 자세한 설명은 the

테이블 이름	열 이름	설명
	seconds, crc_check_chance, min_index_interval, bloom_filter_fp_chance, flags, custom_properties, dclocal_read_repair_chance, table_name, caching, default_time_to_live, read_repair_chance, max_index_interval, extensions, compaction, comment, id, compression, memtable_flush_period_in_ms, status	section called “테이블 생성” 섹션을 참조하세요. 또한 이 테이블을 쿼리하여 Amazon Keyspace에만 적용되고 저장되는 설정을 나열할 수 있습니다. custom_properties 예: • capacity_mode • client_side_timestamps • encryption_specification • point_in_time_recovery • ttl
tables_history	keyspace_name, table_name, event_time, creation_time, custom_properties, event	이 테이블을 쿼리하여 특정 테이블의 스키마 변경 사항에 대해 알아봅니다.
columns	keyspace_name, table_name, column_name, clustering_order, column_name_bytes, kind, position, type	이 테이블은 system_schema 키스페이스의 Cassandra 테이블과 동일합니다.

테이블 이름	열 이름	설명
tags	resource_id, keyspace_name, resource_name, resource_type, tags	이 테이블을 쿼리하여 키스페이스에 태그가 있는지 알아봅니다. 자세한 설명은 the section called “CQL을 사용하여 신규 또는 기존 키스페이스 및 테이블에 태그 추가” 섹션을 참조하세요.
autoscaling	keyspace_name, table_name, provisioned_read_capacity_autoscaling_update, provisioned_write_capacity_autoscaling_update	프로비저닝된 테이블의 Auto Scaling 설정을 가져오려면 이 테이블을 쿼리하세요. 참고로 이러한 설정은 테이블이 활성화되기 전까지는 사용할 수 없습니다. 이 테이블을 쿼리하려면 WHERE 절에 keyspace_name 및 table_name 을 지정해야 합니다. 자세한 설명은 the section called “CQL 사용” 섹션을 참조하세요.

system_multiregion_info

다중 리전 복제에 대한 정보를 저장하는 Amazon Keyspaces 키스페이스입니다.

테이블 이름	열 이름	설명
tables	keyspace_name, table_name, region, status	이 테이블에는 다중 지역 테이블에 대한 정보 (예: 테이블이 AWS 리전 복제되는 위치 및 테이블 상태) 가 들어 있습니다. 또한 이 테이블을 쿼리하여 저장되어 있는 Amazon Keyspace에만 적용되는 설

테이블 이름	열 이름	설명
		정을 나열할 수 있습니다. `custom_properties` 예: • `capacity_mode` 이 테이블을 쿼리하려면 `WHERE` 절에 `keyspace_name` 및 `table_name` 을 지정해야 합니다. 자세한 설명은 the section called “다중 리전 키스페이스 생성(CQL)” 섹션을 참조하세요.
`autoscaling`	`keyspace_name, table_name, provisioned_read_capacity_autoscaling_update, provisioned_write_capacity_autoscaling_update, region`	이 테이블을 쿼리하여 멀티 리전 프로비저닝 테이블의 Auto Scaling 설정을 가져오세요. 참고로 이러한 설정은 테이블이 활성화되기 전까지는 사용할 수 없습니다. 이 테이블을 쿼리하려면 `WHERE` 절에 `keyspace_name` 및 `table_name` 을 지정해야 합니다. 자세한 설명은 the section called “CQL 사용” 섹션을 참조하세요.

Amazon Keyspaces에서 키스페이스 생성

Amazon Keyspaces는 키스페이스 생성 및 삭제와 같은 데이터 정의 언어(DDL) 작업을 비동기적으로 수행합니다.

에서 새 키스페이스의 생성 상태를 모니터링할 수 있습니다. 이 AWS Management Console 상태는 키스페이스가 보류 중이거나 활성화된 시기를 나타냅니다. 또한 `system_schema_mcs` 키스페이스를 사용하여 프로그래밍 방식으로 새 키스페이스의 생성 상태를 모니터링할 수 있습니다. 키스페이스를 사용할 준비가 되면 `system_schema_mcs keyspaces` 테이블에 키스페이스가 표시됩니다.

새 키스페이스를 사용할 준비가 되면 확인하는 권장 설계 패턴은 Amazon Keyspaces `system_schema_mcs keyspaces` 테이블(system_schema_mcs.*)을 폴링하는 것입니다. 키스페이스에 대한 DDL 문 목록은 CQL 언어 참조의 the section called “Keyspaces” 섹션을 참조하세요.

다음 쿼리는 키스페이스가 성공적으로 생성되었는지 여부를 보여 줍니다.

```
SELECT * FROM system_schema_mcs.keyspaces WHERE keyspace_name = 'mykeyspace';
```

성공적으로 생성된 키스페이스의 경우 쿼리 출력은 다음과 같습니다.

```
keyspace_name | durable_writes  | replication
--------------+-----------------+--------------
   mykeyspace | true            |{...} 1 item
```

Amazon Keyspaces에서 테이블로 작업하기

이 섹션에서는 Amazon Keyspaces(Apache Cassandra용)의 테이블 작업에 대한 세부 정보를 제공합니다.

주제

- Amazon Keyspaces에서 테이블 생성
- Amazon Keyspaces에서 멀티 리전 테이블 사용하기
- Amazon Keyspaces의 정적 열

Amazon Keyspaces에서 테이블 생성

Amazon Keyspaces는 테이블 생성 및 삭제와 같은 데이터 정의 언어(DDL) 작업을 비동기적으로 수행합니다. 테이블이 보류 중이거나 활성 AWS Management Console 상태인 시기를 나타내는 에서 새 테이블의 생성 상태를 모니터링할 수 있습니다. 또한 시스템 스키마 테이블을 사용하여 프로그래밍 방식으로 새 테이블의 생성 상태를 모니터링할 수 있습니다.

테이블을 사용할 준비가 되면 시스템 스키마에서 테이블이 활성 상태로 표시됩니다. 새 테이블을 사용할 준비가 되면 확인하는 권장 설계 패턴은 Amazon Keyspaces 시스템 스키마 테이블

(system_schema_mcs.*)을 폴링하는 것입니다. 테이블에 대한 DDL 문 목록은 CQL 언어 참조의 the section called “표” 섹션을 참조하세요.

다음 쿼리는 테이블 상태를 보여 줍니다.

```
SELECT keyspace_name, table_name, status FROM system_schema_mcs.tables WHERE
 keyspace_name = 'mykeyspace' AND table_name = 'mytable';
```

아직 생성 중이고 보류 중인 테이블의 경우 쿼리 출력은 다음과 같습니다.

```
keyspace_name | table_name | status
--------------+------------+--------
   mykeyspace |    mytable | CREATING
```

성공적으로 생성되고 활성화된 테이블의 경우 쿼리 출력은 다음과 같습니다.

```
keyspace_name | table_name | status
--------------+------------+--------
   mykeyspace |    mytable | ACTIVE
```

Amazon Keyspaces에서 멀티 리전 테이블 사용하기

다중 리전 테이블에는 다음 두 가지 방법 중 하나로 쓰기 처리 용량이 구성되어 있어야 합니다.

- WRU (쓰기 요청 단위) 단위로 측정되는 온디맨드 용량 모드
- Auto Scaling이 포함된 프로비저닝된 용량 모드 (쓰기 용량 단위 (WCU) 단위로 측정

Auto Scaling과 함께 프로비저닝된 용량 모드 또는 온디맨드 용량 모드를 사용하면 멀티 리전 테이블에 대해 복제된 쓰기를 수행할 수 있는 충분한 용량을 확보할 수 있습니다. AWS 리전

> **Note**
>
> 지역 중 하나에서 테이블의 용량 모드를 변경하면 모든 복제본의 용량 모드가 변경됩니다.

기본적으로 Amazon Keyspace는 멀티 리전 테이블에 온디맨드 모드를 사용합니다. 온디맨드 모드를 사용하면 애플리케이션에서 수행할 것으로 예상되는 읽기 및 쓰기 처리량을 지정할 필요가 없습니다.

Amazon Keyspaces는 이전에 도달한 트래픽 수준으로 워크로드를 늘리거나 줄일 때 워크로드를 즉시 수용합니다. 워크로드의 트래픽 수준이 새로운 최고점에 도달하면 Amazon Keyspace는 워크로드를 수용하도록 빠르게 적응합니다.

테이블에 프로비저닝된 용량 모드를 선택하는 경우 애플리케이션에 필요한 초당 읽기 용량 단위 (RCU) 및 쓰기 용량 단위 (WCU) 수를 구성해야 합니다.

다중 지역 테이블의 처리 용량 요구 사항을 계획하려면 먼저 각 지역에 필요한 초당 WCU 수를 추정해야 합니다. 그런 다음 테이블이 복제되는 모든 지역의 쓰기를 추가하고 이 합계를 사용하여 각 지역의 용량을 프로비저닝합니다. 이는 한 지역에서 수행되는 모든 쓰기가 각 복제 지역에서도 반복되어야 하기 때문에 필요합니다.

테이블 용량이 충분하지 않아 모든 지역의 쓰기를 처리할 수 없는 경우 용량 예외가 발생합니다. 또한 지역 간 복제 대기 시간도 늘어날 것입니다.

예를 들어 미국 동부 (버지니아 북부) 에서 초당 5개, 미국 동부 (오하이오) 에서 초당 10개, 유럽 (아일랜드) 에서 초당 5개가 예상되는 다중 지역 테이블의 경우 테이블은 미국 동부 (버지니아 북부), 미국 동부 (오하이오), 유럽 (아일랜드) 등 각 지역에서 20개의 WCU를 사용할 것으로 예상해야 합니다. 즉, 이 예시에서는 각 테이블 복제본에 대해 20개의 WCU를 프로비저닝해야 합니다. Amazon을 사용하여 테이블의 용량 사용량을 모니터링할 수 CloudWatch 있습니다. 자세한 설명은 the section called “를 통한 모니터링 CloudWatch” 섹션을 참조하세요.

각 다중 지역 쓰기 요금은 WCU의 1.25배로 청구되므로 이 예에서는 총 75개의 WCU에 대한 요금이 청구됩니다. 요금에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 요금을 참조하세요.

Amazon Keyspaces Auto Scaling을 통한 프로비저닝 용량에 대한 자세한 내용은 을 참조하십시오. the section called “Auto Scaling을 통한 처리 용량 관리”

> **Note**
>
> 테이블이 Auto Scaling과 함께 프로비저닝된 용량 모드에서 실행되는 경우, 프로비저닝된 쓰기 용량은 각 지역의 Auto Scaling 설정 내에서 유동적으로 허용될 수 있습니다.

Amazon Keyspaces의 정적 열

클러스터링 열이 있는 Amazon Keyspaces 테이블에서 `STATIC` 키워드를 사용하여 정적 열을 생성할 수 있습니다. 정적 열에 저장된 값은 논리적 파티션의 모든 행에서 공유됩니다. 이 열의 값을 업데이트하면 Amazon Keyspaces는 파티션의 모든 행에 변경 사항을 자동으로 적용합니다.

이 섹션에서는 정적 열에 쓸 때 인코딩된 데이터 크기를 계산하는 방법을 설명합니다. 이 프로세스는 행의 비정적 열에 데이터를 쓰는 프로세스와는 별도로 처리됩니다. 정적 데이터에 대한 크기 할당량 외에도 정적 열에 대한 읽기 및 쓰기 작업은 테이블의 측정 및 처리량 용량에도 독립적으로 영향을 미칩니다.

Amazon Keyspaces의 논리적 파티션당 정적 열 크기 계산

이 섹션에서는 Amazon Keyspaces의 인코딩된 정적 열 크기를 추정하는 방법에 대한 세부 정보를 제공합니다. 인코딩된 크기는 요금 및 할당량 사용량을 계산할 때 사용됩니다. 또한 테이블의 프로비저닝된 처리량 용량 요구 사항을 계산할 때는 인코딩된 크기를 사용해야 합니다. Amazon Keyspaces 내에서 인코딩된 정적 열 크기를 계산하려는 경우 다음 지침을 사용할 수 있습니다.

- 파티션 키는 최대 2048바이트의 데이터를 포함할 수 있습니다. 파티션 키의 각 키 열에는 최대 3바이트의 메타데이터가 필요합니다. 이러한 메타데이터 바이트는 파티션당 1MB의 정적 데이터 크기 할당량에 포함됩니다. 정적 데이터 크기를 계산할 때는 각 파티션 키 열이 전체 3바이트의 메타데이터를 사용한다고 가정해야 합니다.
- 데이터 유형에 따른 정적 열 데이터 값의 원시 크기를 사용합니다. 데이터 유형에 대한 자세한 내용은 the section called “데이터 타입” 섹션을 참조하세요.
- 메타데이터의 정적 데이터 크기에 104바이트를 추가합니다.
- 클러스터링 열과 일반, 기본이 아닌 키 열은 정적 데이터 크기에 포함되지 않습니다. 행 내 비정적 데이터의 크기를 추정하는 방법에 대한 자세한 내용은 the section called “행 크기 계산” 섹션을 참조하세요.

인코딩된 정적 열의 총 크기는 다음 공식을 기반으로 합니다.

```
partition key columns + static columns + metadata = total encoded size of static data
```

모든 열의 유형이 정수인 다음 테이블을 예로 들어 보겠습니다. 테이블에는 파티션 키 열 2개, 클러스터링 열 2개, 일반 열 1개 및 정적 열 1개가 있습니다.

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2
 int, reg_col1 int, static_col1 int static, primary key((pk_col1, pk_col2),ck_col1,
 ck_col2));
```

이 예제에서는 다음 문의 정적 데이터 크기를 계산합니다.

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, static_col1) values(1,2,6);
```

이 쓰기 작업에 필요한 총 바이트를 추정하려면 다음 단계를 사용하면 됩니다.

1. 열에 저장된 데이터 유형의 바이트와 메타데이터 바이트를 더하여 파티션 키 열의 크기를 계산합니다. 모든 파티션 키 열에 대해 이 과정을 반복합니다.

   a. 파티션 키(pk_col1)의 첫 번째 열 크기를 계산합니다.

   ```
   4 bytes for the integer data type + 3 bytes for partition key metadata = 7
    bytes
   ```

   b. 파티션 키(pk_col2)의 두 번째 열 크기를 계산합니다.

   ```
   4 bytes for the integer data type + 3 bytes for partition key metadata = 7
    bytes
   ```

   c. 두 열을 모두 더하여 파티션 키 열의 총 예상 크기를 구합니다.

   ```
   7 bytes + 7 bytes = 14 bytes for the partition key columns
   ```

2. 정적 열의 크기를 더합니다. 이 예제에서는 정수를 저장하는 정적 열이 하나뿐입니다(4바이트 필요).
3. 마지막으로 정적 열 데이터의 인코딩된 총 크기를 구하려면 프라이머리 키 열과 정적 열의 바이트를 더하고 메타데이터에 대해 104바이트를 더 추가합니다.

   ```
   14 bytes for the partition key columns + 4 bytes for the static column + 104 bytes
    for metadata = 122 bytes.
   ```

동일한 문을 사용하여 정적 및 비정적 데이터를 업데이트할 수도 있습니다. 쓰기 작업의 총 크기를 추정하려면 먼저 비정적 데이터 업데이트의 크기를 계산해야 합니다. 그런 다음 the section called “행 크기 계산”의 예제와 같이 행 업데이트 크기를 계산하고 결과를 더합니다.

이 경우 총 2MB를 쓸 수 있습니다. 즉 1MB는 최대 행 크기 할당량이고 1MB는 논리적 파티션당 최대 정적 데이터 크기 할당량입니다.

동일한 문에서 정적 및 비정적 데이터의 총 업데이트 크기를 계산하려면 다음 공식을 사용하면 됩니다.

```
(partition key columns + static columns + metadata = total encoded size of static data)
 + (partition key columns + clustering columns + regular columns + row metadata = total
 encoded size of row)
= total encoded size of data written
```

모든 열의 유형이 정수인 다음 테이블을 예로 들어 보겠습니다. 테이블에는 파티션 키 열 2개, 클러스터링 열 2개, 일반 열 1개 및 정적 열 1개가 있습니다.

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2
 int, reg_col1 int, static_col1 int static, primary key((pk_col1, pk_col2),ck_col1,
 ck_col2));
```

이 예제에서는 다음 문과 같이 테이블에 행을 쓸 때의 데이터 크기를 계산합니다.

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, ck_col1, ck_col2, reg_col1,
 static_col1) values(2,3,4,5,6,7);
```

이 쓰기 작업에 필요한 총 바이트를 추정하려면 다음 단계를 사용하면 됩니다.

1. 앞에 표시된 대로 정적 데이터의 인코딩된 총 크기를 계산합니다. 이 예제에서는 122바이트입니다.
2. the section called “행 크기 계산”의 단계에 따라 비정적 데이터의 업데이트를 기반으로 인코딩된 행의 총 크기를 추가합니다. 이 예제에서 행 업데이트의 총 크기는 134바이트입니다.

   ```
   122 bytes for static data + 134 bytes for nonstatic data = 256 bytes.
   ```

Amazon Keyspaces의 정적 데이터 읽기/쓰기 작업 측정

Cassandra에서 정적 데이터는 개별 행이 아니라 논리적 파티션과 연결됩니다. Amazon Keyspaces의 논리적 파티션은 여러 물리적 스토리지 파티션에 걸쳐 있을 수 있으며 사실상 크기에 제한이 없습니다. 결과적으로 Amazon Keyspaces는 정적 데이터와 비정적 데이터에 대한 쓰기 작업을 별도로 측정합니다. 또한 정적 데이터와 비정적 데이터를 모두 포함하는 쓰기에는 데이터 일관성을 제공하기 위한 추가 기본 작업이 필요합니다.

정적 데이터와 비정적 데이터를 혼합하여 쓰기 작업을 수행하는 경우 두 개의 개별 쓰기 작업이 발생합니다. 하나는 비정적 데이터에 대한 것이고 다른 하나는 정적 데이터에 대한 것입니다. 이는 온디맨드 및 프로비저닝된 읽기/쓰기 용량 모드 모두에 적용됩니다.

다음 예제는 정적 열이 있는 Amazon Keyspaces의 테이블에 대해 프로비저닝된 처리량 용량 요구 사항을 계산할 때 필요한 읽기 용량 단위(RCU) 및 쓰기 용량 단위(WCU)를 추정하는 방법에 대한 세부 정보를 제공합니다. 다음 공식을 사용하여 정적 데이터와 비정적 데이터가 모두 포함된 쓰기를 테이블에서 처리하는 데 필요한 용량을 추정할 수 있습니다.

```
2 x WCUs required for nonstatic data + 2 x WCUs required for static data
```

예를 들어 애플리케이션이 초당 27KB의 데이터를 쓰고 각 쓰기에 25.5KB의 비정적 데이터와 1.5KB의 정적 데이터가 포함되는 경우 테이블에는 56WCU(2 x 26WCU + 2 x 2WCU)가 필요합니다.

Amazon Keyspaces는 여러 행의 읽기와 동일하게 정적 및 비정적 데이터의 읽기를 측정합니다. 따라서 동일한 작업에서 정적 및 비정적 데이터를 읽는 비용은 읽기를 수행하기 위해 처리된 데이터의 총 크기를 기준으로 합니다.

Amazon에서 서버리스 리소스를 모니터링하는 방법을 CloudWatch 알아보려면 을 참조하십시오the section called “를 통한 모니터링 CloudWatch”.

Amazon Keyspaces에서 행으로 작업하기

이 섹션에서는 Amazon Keyspaces(Apache Cassandra용)의 행 작업에 대한 세부 정보를 제공합니다. 테이블은 Amazon Keyspaces의 기본 데이터 구조이며 테이블의 데이터는 열과 행으로 구성됩니다.

주제

Amazon Keyspaces에서의 행 크기 계산

Amazon Keyspaces는 한 자릿수 밀리초의 읽기 및 쓰기 성능을 제공하고 여러 AWS 가용 영역에 데이터를 안정적으로 저장하는 완전 관리형 스토리지를 제공합니다. Amazon Keyspaces는 모든 행과 프라이머리 키 열에 메타데이터를 연결하여 효율적인 데이터 액세스와 고가용성을 지원합니다.

이 섹션에서는 Amazon Keyspaces의 인코딩된 행 크기를 추정하는 방법에 대한 세부 정보를 제공합니다. 인코딩된 행 크기는 청구서 및 할당량 사용량을 계산할 때 사용됩니다. 또한 테이블의 프로비저닝된 처리량 용량 요구 사항을 계산할 때는 인코딩된 행 크기를 사용해야 합니다. Amazon Keyspaces 내에서 인코딩된 행 크기를 계산하려는 경우 다음 지침을 사용할 수 있습니다.

- 프라이머리 키가 아닌 열인 일반 열, 클러스터링 열 또는 `STATIC` 열의 경우 데이터 유형에 따른 셀 데이터의 원시 크기를 사용하고 필요한 메타데이터를 추가합니다. Amazon Keyspaces에서 지원되는 데이터 유형에 대한 자세한 내용은 the section called “데이터 타입” 섹션을 참조하세요. Amazon Keyspaces가 데이터 유형 값과 메타데이터를 저장하는 방식의 몇 가지 주요 차이점은 다음과 같습니다.
- 각 열 이름에 필요한 공간은 열 식별자를 사용하여 저장되고 열에 저장된 각 데이터 값에 추가됩니다. 열 식별자의 스토리지 값은 테이블의 전체 열 수에 따라 달라집니다.

  - 1-62개 열: 1바이트
  - 63-124개 열: 2바이트
  - 125-186개 열: 3바이트

  62개 열을 추가할 때마다 1바이트를 추가합니다. Amazon Keyspaces에서는 단일 INSERT 또는 UPDATE 문으로 최대 225개의 일반 열을 수정할 수 있다는 점에 유의하세요. 자세한 내용은 the section called “Amazon Keyspaces 서비스 할당량” 섹션을 참조하세요.
- 파티션 키는 최대 2048바이트의 데이터를 포함할 수 있습니다. 파티션 키의 각 키 열에는 최대 3바이트의 메타데이터가 필요합니다. 행 크기를 계산할 때는 각 파티션 키 열이 전체 3바이트의 메타데이터를 사용한다고 가정해야 합니다.
- 클러스터링 열은 최대 850바이트의 데이터를 저장할 수 있습니다. 데이터 값의 크기 외에도 각 클러스터링 열에는 메타데이터에 대한 데이터 값 크기의 최대 20%가 필요합니다. 행 크기를 계산할 때는 클러스터링 열 데이터 값 5바이트당 메타데이터 1바이트를 추가해야 합니다.
- Amazon Keyspaces는 각 파티션 키와 클러스터링 키 열의 데이터 값을 두 번 저장합니다. 추가 오버헤드는 효율적인 쿼리와 내장된 인덱싱에 사용됩니다.
- Cassandra ASCII, TEXT 및 VARCHAR 문자열 데이터 유형은 모두 UTF-8 바이너리 인코딩의 유니코드를 사용하여 Amazon Keyspaces에 저장됩니다. Amazon Keyspaces의 문자열 크기는 UTF-8 인코딩된 바이트 수와 같습니다.
- Cassandra INT, BIGINT, SMALLINT 및 TINYINT 데이터 유형은 Amazon Keyspaces에 가변 길이(유효 자릿수 최대 38자리)의 데이터 값으로 저장됩니다. 앞과 끝의 0은 잘립니다. 이러한 데이터 유형의 크기는 유효 숫자 2자리당 약 1바이트+1바이트입니다.
- Amazon Keyspaces의 BLOB은 값의 원시 바이트 길이와 함께 저장됩니다.
- Null 값 또는 Boolean 값의 크기는 1바이트입니다.
- LIST 또는 MAP과 같은 컬렉션 데이터 유형을 저장하는 열에는 내용에 관계없이 3바이트의 메타데이터가 필요합니다. LIST 또는 MAP의 크기는 (열 ID) + 합계(중첩 요소 크기) + (3바이트)입니다. 빈 LIST 또는 MAP의 크기는 (열 ID) + (3바이트)입니다. 또한 각 개별 LIST 또는 MAP 요소에는 1바이트의 메타데이터를 필요로 합니다.
- STATIC 열 데이터는 최대 행 크기인 1MB에 포함되지 않습니다. 정적 열의 데이터 크기를 계산하려면 the section called “논리적 파티션당 정적 열 크기 계산” 섹션을 참조하세요.
- 기능이 켜져 있을 때 각 행의 모든 열에 대해 클라이언트 측 타임스탬프가 저장됩니다. 이러한 타임스탬프는 약 20~40바이트(데이터에 따라 다름)를 차지하며 행의 스토리지 및 처리량 비용에 영향을 줍니다. 자세한 내용은 the section called “Amazon Keyspaces의 클라이언트 측 타임스탬프” 섹션을 참조하세요.
- 행 메타데이터의 경우 각 행 크기에 100바이트를 추가합니다.

인코딩된 데이터 행의 총 크기는 다음 공식을 기반으로 합니다.

```
partition key columns + clustering columns + regular columns + row metadata = total
 encoded size of row
```

> **Important**
>
> 모든 열 메타데이터(예: 열 ID, 파티션 키 메타데이터, 클러스터링 열 메타데이터, 클라이언트 측 타임스탬프 및 행 메타데이터)는 최대 행 크기인 1MB에 포함됩니다.

모든 열의 유형이 정수인 다음 테이블을 예로 들어 보겠습니다. 테이블에는 파티션 키 열 2개, 클러스터링 열 2개, 일반 열 1개가 있습니다. 이 테이블에는 5개의 열이 있으므로 열 이름 식별자에 필요한 공간은 1바이트입니다.

```
CREATE TABLE mykeyspace.mytable(pk_col1 int, pk_col2 int, ck_col1 int, ck_col2 int,
 reg_col1 int, primary key((pk_col1, pk_col2),ck_col1, ck_col2));
```

이 예제에서는 다음 문과 같이 테이블에 행을 쓸 때의 데이터 크기를 계산합니다.

```
INSERT INTO mykeyspace.mytable (pk_col1, pk_col2, ck_col1, ck_col2, reg_col1)
 values(1,2,3,4,5);
```

이 쓰기 작업에 필요한 총 바이트를 추정하려면 다음 단계를 사용하면 됩니다.

1. 열에 저장된 데이터 유형의 바이트와 메타데이터 바이트를 더하여 파티션 키 열의 크기를 계산합니다. 모든 파티션 키 열에 대해 이 과정을 반복합니다.

   a. 파티션 키(pk_col1)의 첫 번째 열 크기를 계산합니다.

   ```
   (2 bytes for the integer data type) x 2 + 1 byte for the column id + 3 bytes
    for partition key metadata = 8 bytes
   ```

   b. 파티션 키(pk_col2)의 두 번째 열 크기를 계산합니다.

   ```
   (2 bytes for the integer data type) x 2 + 1 byte for the column id + 3 bytes
    for partition key metadata = 8 bytes
   ```

   c. 두 열을 모두 더하여 파티션 키 열의 총 예상 크기를 구합니다.

```
8 bytes + 8 bytes = 16 bytes for the partition key columns
```

2. 열에 저장된 데이터 유형의 바이트와 메타데이터 바이트를 더하여 클러스터링 열의 크기를 계산합니다. 모든 클러스터링 열에 대해 이 과정을 반복합니다.

   a. 클러스터링 열(ck_col1)의 첫 번째 열 크기를 계산합니다.

   ```
   (2 bytes for the integer data type) x 2 + 20% of the data value (2 bytes) for
    clustering column metadata + 1 byte for the column id  = 6 bytes
   ```

   b. 클러스터링 열(ck_col2)의 두 번째 열 크기를 계산합니다.

   ```
   (2 bytes for the integer data type) x 2 + 20% of the data value (2 bytes) for
    clustering column metadata + 1 byte for the column id = 6 bytes
   ```

   c. 두 열을 모두 더하여 클러스터링 열의 총 예상 크기를 구합니다.

   ```
   6 bytes + 6 bytes = 12 bytes for the clustering columns
   ```

3. 일반 열의 크기를 더합니다. 이 예제에서는 한 자리 정수를 저장하는 열이 하나뿐이며 이 경우 열 ID로 2바이트에 1바이트가 필요합니다.
4. 마지막으로 인코딩된 총 행 크기를 구하려면 모든 열의 바이트를 더하고 행 메타데이터에 대해 100바이트를 더 추가합니다.

   ```
   16 bytes for the partition key columns + 12 bytes for clustering columns + 3 bytes
    for the regular column + 100 bytes for row metadata = 131 bytes.
   ```

Amazon CloudWatch로 서버리스 리소스를 모니터링하는 방법을 알아보려면 the section called “를 통한 모니터링 CloudWatch” 섹션을 참조하세요.

Amazon Keyspaces에서 쿼리로 작업하기

이 섹션에서는 Amazon Keyspaces(Apache Cassandra용)에서 쿼리 작업 작업을 수행하는 방법을 소개합니다. 데이터를 쿼리, 변환 및 관리하는 데 사용할 수 있는 CQL 문은 SELECT, INSERT, UPDATE 및 DELETE입니다. 다음 주제에서는 쿼리 작업 시 사용할 수 있는 보다 복잡한 몇 가지 옵션에 대해 설명합니다. 예제를 포함한 전체 언어 구문은 the section called “DML 문” 섹션을 참조하세요.

주제

Amazon Keyspaces에서 **SELECT** 문과 함께 **IN** 연산자 사용

SELECT IN

SELECT 문을 사용하여 테이블의 데이터를 쿼리할 수 있습니다. 문은 테이블에서 하나 이상의 행에 대해 하나 이상의 열을 읽고 요청과 일치하는 행이 포함된 결과 집합을 반환합니다. SELECT 문에는 읽고 결과 집합에서 반환할 열을 결정하는 select_clause가 포함되어 있습니다. 절에는 데이터를 반환하기 전에 데이터를 변환하는 지침이 포함될 수 있습니다. 선택적 WHERE 절은 쿼리해야 하는 행을 지정하며 프라이머리 키의 일부인 열에 대한 관계로 구성됩니다. Amazon Keyspaces는 WHERE 절의 IN 키워드를 지원합니다. 이 섹션에서는 예제를 사용하여 Amazon Keyspaces가 IN 키워드로 SELECT 문을 처리하는 방법을 보여 줍니다.

이 예제는 Amazon Keyspaces가 IN 키워드가 포함된 SELECT 문을 하위 쿼리로 분류하는 방법을 보여 줍니다. 이 예제에서는 이름이 my_keyspace.customers인 테이블을 사용합니다. 테이블에는 프라이머리 키 열 department_id 1개, 클러스터링 열 sales_region_id 및 sales_representative_id 2개, customer_name 열에 고객 이름이 포함된 열 1개가 있습니다.

```
SELECT * FROM my_keyspace.customers;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  0            |       0         |            0            |    a
  0            |       0         |            1            |    b
  0            |       1         |            0            |    c
  0            |       1         |            1            |    d
  1            |       0         |            0            |    e
  1            |       0         |            1            |    f
  1            |       1         |            0            |    g
  1            |       1         |            1            |    h
```

이 테이블을 사용하면 다음 SELECT 문을 실행하여 WHERE 절의 IN 키워드로 관심 있는 부서 및 판매 리전의 고객을 찾을 수 있습니다. 다음 문은 이에 대한 예입니다.

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (0, 1) AND sales_region_id
 IN (0, 1);
```

Amazon Keyspaces는 다음 출력과 같이 이 문을 네 개의 하위 쿼리로 나눕니다.

```
SELECT * FROM my_keyspace.customers WHERE department_id = 0 AND sales_region_id = 0;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  0            |        0        |           0             |    a
  0            |        0        |           1             |    b

SELECT * FROM my_keyspace.customers WHERE department_id = 0 AND sales_region_id = 1;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  0            |        1        |           0             |    c
  0            |        1        |           1             |    d

SELECT * FROM my_keyspace.customers WHERE department_id = 1 AND sales_region_id = 0;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  1            |        0        |           0             |    e
  1            |        0        |           1             |    f

SELECT * FROM my_keyspace.customers WHERE department_id = 1 AND sales_region_id = 1;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  1            |        1        |           0             |    g
  1            |        1        |           1             |    h
```

IN 키워드를 사용하면 Amazon Keyspaces는 다음과 같은 경우에 결과를 자동으로 페이지로 나눕니다.

- 10회마다 하위 쿼리가 처리된 후
- 1MB의 논리적 IO를 처리한 후
- PAGE SIZE를 구성한 경우 Amazon Keyspaces는 집합 PAGE SIZE를 기반으로 처리할 쿼리 수를 읽은 후 페이지를 매깁니다.
- LIMIT 키워드를 사용하여 반환되는 행 수를 줄이면 Amazon Keyspaces는 집합 LIMIT를 기반으로 처리할 쿼리 수를 읽은 후 페이지를 매깁니다.

다음 테이블은 예제를 통해 이를 설명하는 데 사용됩니다.

페이지 매김에 대한 자세한 내용은 the section called “결과 페이지 매김” 섹션을 참조하세요.

```
SELECT * FROM my_keyspace.customers;

 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  2            |        0        |          0              |    g
  2            |        1        |          1              |    h
  2            |        2        |          2              |    i
  0            |        0        |          0              |    a
  0            |        1        |          1              |    b
  0            |        2        |          2              |    c
  1            |        0        |          0              |    d
  1            |        1        |          1              |    e
  1            |        2        |          2              |    f
  3            |        0        |          0              |    j
  3            |        1        |          1              |    k
  3            |        2        |          2              |    l
```

이 테이블에서 다음 문을 실행하여 페이지 매김의 작동 방식을 확인할 수 있습니다.

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (0, 1, 2, 3) AND
 sales_region_id IN (0, 1, 2) AND sales_representative_id IN (0, 1);
```

Amazon Keyspaces는 이 문을 24개의 하위 쿼리로 처리합니다. 이 쿼리에 포함된 모든 IN 용어의 Cartesian 제품의 카디널리티가 24이기 때문입니다.

```
 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  0            |        0        |          0              |    a
  0            |        1        |          1              |    b
  1            |        0        |          0              |    d
  1            |        1        |          1              |    e

---MORE---
 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  2            |        0        |          0              |    g
  2            |        1        |          1              |    h
  3            |        0        |          0              |    j
```

```
---MORE---
 department_id | sales_region_id | sales_representative_id | customer_name
---------------+-----------------+-------------------------+--------------
  3            |        1        |          1              |    k
```

이 예제는 IN 키워드와 함께 SELECT 문에서 ORDER BY 절을 사용하는 방법을 보여 줍니다.

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (3, 2, 1) ORDER BY
 sales_region_id DESC;

         department_id | sales_region_id | sales_representative_id | customer_name
        ---------------+-----------------+-------------------------+--------------
          3            |        2        |          2              |    l
          3            |        1        |          1              |    k
          3            |        0        |          0              |    j
          2            |        2        |          2              |    i
          2            |        1        |          1              |    h
          2            |        0        |          0              |    g
          1            |        2        |          2              |    f
          1            |        1        |          1              |    e
          1            |        0        |          0              |    d
```

하위 쿼리는 쿼리에 파티션 키 및 클러스터링 키 열이 표시된 순서대로 처리됩니다. 아래 예제에서는 파티션 키 값 ”2“에 대한 하위 쿼리가 먼저 처리된 다음 파티션 키 값 ”3“ 및 ”1“에 대한 하위 쿼리가 처리됩니다. 지정된 하위 쿼리의 결과는 쿼리의 정렬 절(있는 경우) 또는 테이블 생성 중에 정의된 테이블의 클러스터링 순서에 따라 정렬됩니다.

```
SELECT * FROM my_keyspace.customers WHERE department_id IN (2, 3, 1) ORDER BY
 sales_region_id DESC;

         department_id | sales_region_id | sales_representative_id | customer_name
        ---------------+-----------------+-------------------------+--------------
          2            |        2        |          2              |    i
          2            |        1        |          1              |    h
          2            |        0        |          0              |    g
          3            |        2        |          2              |    l
          3            |        1        |          1              |    k
          3            |        0        |          0              |    j
          1            |        2        |          2              |    f
          1            |        1        |          1              |    e
          1            |        0        |          0              |    d
```

Amazon Keyspaces의 결과 순서 지정

ORDER BY 절은 SELECT 문에서 반환된 결과의 정렬 순서를 지정합니다. 문은 열 이름 목록을 인수로 취하며 각 열에 대해 데이터의 정렬 순서를 지정할 수 있습니다. 순서 지정 절에는 클러스터링 열만 지정할 수 있으며 클러스터링이 아닌 열은 허용되지 않습니다.

반환된 결과에 대한 두 가지 사용 가능한 정렬 순서 옵션은 ASC(오름차순)와 DESC(내림차순) 정렬 순서입니다.

```
SELECT * FROM my_keyspace.my_table ORDER BY (col1 ASC, col2 DESC, col3 ASC);

 col1 | col2 | col3
------+------+------
  0   |  6   |  a
  1   |  5   |  b
  2   |  4   |  c
  3   |  3   |  d
  4   |  2   |  e
  5   |  1   |  f
  6   |  0   |  g
```

```
SELECT * FROM my_keyspace.my_table ORDER BY (col1 DESC, col2 ASC, col3 DESC);

 col1 | col2 | col3
------+------+------
  6   |  0   |  g
  5   |  1   |  f
  4   |  2   |  e
  3   |  3   |  d
  2   |  4   |  c
  1   |  5   |  b
  0   |  6   |  a
```

쿼리 문에 정렬 순서를 지정하지 않으면 클러스터링 열의 기본 순서가 사용됩니다.

순서 지정 절에서 사용할 수 있는 정렬 순서는 테이블 생성 시 각 클러스터링 열에 할당된 정렬 순서에 따라 달라집니다. 쿼리 결과는 테이블 생성 시 모든 클러스터링 열에 정의된 순서로만 정렬하거나 정의된 정렬 순서의 역순으로 정렬할 수 있습니다. 다른 가능한 조합은 허용되지 않습니다.

예를 들어 테이블의 CLUSTERING ORDER가 (col1 ASC, col2 DESC, col3 ASC)인 경우 ORDER BY에 대한 유효한 매개변수는 (col1 ASC, col2 DESC, col3 ASC) 또는 (col1 DESC, col2 ASC, col3 DESC)

입니다. CLUSTERING ORDER에 대한 자세한 내용은 the section called “CREATE TABLE” 아래의 table_options 섹션을 참조하세요.

Amazon Keyspaces의 결과 페이지 매김

Amazon Keyspaces는 SELECT 문을 처리하기 위해 읽은 데이터가 1MB를 초과하는 경우 SELECT 문의 결과를 자동으로 페이지 매김합니다. 페이지를 매기면 SELECT 문 결과는 크기가 1MB 이하인 데이터 '페이지'로 분리됩니다. 애플리케이션은 결과의 첫 번째 페이지를 처리한 다음 두 번째 페이지를 처리하고 이런 식으로 계속할 수 있습니다. 클라이언트는 여러 행을 반환하는 SELECT 쿼리를 처리할 때 항상 페이지 매김 토큰을 확인해야 합니다.

클라이언트가 1MB 이상의 데이터를 읽어야 하는 PAGE SIZE를 제공하는 경우 Amazon Keyspaces는 1MB 데이터 읽기 증분에 따라 결과를 여러 페이지로 자동 분할합니다.

예를 들어 행의 평균 크기가 100KB이고 PAGE SIZE를 20으로 지정하는 경우 Amazon Keyspaces는 10개 행(1000KB의 데이터 읽기)을 읽은 후 자동으로 데이터를 페이지 매깁니다.

Amazon Keyspaces는 결과 세트에 반환된 행 수가 아니라 요청을 처리하기 위해 읽는 행 수를 기준으로 결과 페이지를 매기기 때문에 필터링된 쿼리를 실행하는 경우 일부 페이지에는 행이 포함되지 않을 수 있습니다.

예를 들어 PAGE SIZE를 10으로 설정하고 Keyspaces가 30개의 행을 평가하여 SELECT 쿼리를 처리하면 Amazon Keyspaces는 세 페이지를 반환합니다. 행의 하위 집합만 쿼리와 일치하는 경우 일부 페이지의 행이 10개 미만일 수 있습니다. PAGE SIZEof LIMIT 쿼리가 읽기 용량에 미치는 영향에 대한 예는 을 참조하십시오the section called “쿼리 제한”.

Amazon Keyspaces에서 파티셔너 사용하기

Apache Cassandra에서 파티셔너는 클러스터에 저장되는 노드 데이터를 제어합니다. 파티셔너는 파티션 키의 해시된 값을 사용하여 숫자 토큰을 생성합니다. Cassandra는 이 토큰을 사용하여 노드 전체에 데이터를 분배합니다. 또한 클라이언트는 SELECT 작업 및 WHERE 절에서 이러한 토큰을 사용하여 읽기 및 쓰기 작업을 최적화할 수 있습니다. 예를 들어 클라이언트는 각 병렬 작업에서 쿼리할 고유한 토큰 범위를 지정하여 대형 테이블에서 병렬 쿼리를 효율적으로 수행할 수 있습니다.

Amazon Keyspaces는 세 가지 파티셔너를 제공합니다.

Murmur3Partitioner(기본값)

Apache Cassandra 호환 Murmur3Partitioner Murmur3Partitioner는 Amazon Keyspaces 및 Cassandra 1.2 이상 버전의 기본 Cassandra 파티셔너입니다.

RandomPartitioner

Apache Cassandra 호환 `RandomPartitioner` `RandomPartitioner`는 Cassandra 1.2 이전 버전의 기본 Cassandra 파티셔너입니다.

Keyspaces 기본 파티셔너

`DefaultPartitioner`는 `RandomPartitioner`와 동일한 `token` 함수 결과를 반환합니다.

파티셔너 설정은 계정 수준에서 리전별로 적용됩니다. 예를 들어 미국 동부(버지니아 북부)에서 파티셔너를 변경하면 이 리전의 동일한 계정에 있는 모든 테이블에 변경 내용이 적용됩니다. 파티셔너는 언제든지 안전하게 변경할 수 있습니다. 구성 변경을 완료하는 데 약 10분이 걸립니다. 파티셔너 설정을 변경할 때 Amazon Keyspaces 데이터를 다시 로드할 필요가 없습니다. 클라이언트는 다음 번에 연결할 때 새 파티셔너 설정을 자동으로 사용합니다.

AWS Management Console 또는 Cassandra 쿼리 언어(CQL)를 사용하여 파티셔너를 변경할 수 있습니다.

AWS Management Console

Amazon Keyspaces 콘솔을 사용하여 파티셔너를 변경하려면

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 구성을 선택합니다.
3. 구성 페이지에서 파티셔너 편집으로 이동합니다.
4. 사용 중인 Cassandra 버전과 호환되는 파티셔너를 선택합니다. 파티셔너 변경을 적용하는 데 약 10분이 걸립니다.

> **Note**
>
> 구성 변경이 완료되면 새 파티셔너 사용 요청을 위해 Amazon Keyspaces와의 연결을 끊었다가 다시 연결해야 합니다.

Cassandra Query Language (CQL)

1. 계정에 구성된 파티셔너를 보려면 다음 쿼리를 사용할 수 있습니다.

```
SELECT partitioner from system.local;
```

파티셔너가 변경되지 않은 경우 쿼리의 출력은 다음과 같습니다.

```
partitioner
-----------------------------------------
com.amazonaws.cassandra.DefaultPartitioner
```

2. 파티셔너를 Murmur3 파티셔너로 업데이트하려면 다음 문을 사용할 수 있습니다.

```
UPDATE system.local set
 partitioner='org.apache.cassandra.dht.Murmur3Partitioner' where key='local';
```

3. 이 구성 변경을 완료하는 데 약 10분이 걸립니다. 파티셔너가 설정되었는지 확인하려면 SELECT 쿼리를 다시 실행하면 됩니다. 최종 읽기 일관성으로 인해 응답에 최근 완료된 파티셔너 변경의 결과가 반영되지 않을 수 있습니다. 잠시 후 SELECT 작업을 다시 반복하면 응답이 최신 데이터를 반환합니다.

```
SELECT partitioner from system.local;
```

> **Note**
>
> 요청이 새 파티셔너를 사용하도록 Amazon Keyspaces와의 연결을 끊었다가 다시 연결해야 합니다.

Amazon Keyspaces 리소스의 태그 및 레이블을 사용한 작업

Amazon Keyspaces(Apache Cassandra용) 리소스에 태그를 사용하여 레이블을 지정할 수 있습니다. 태그를 사용하면 용도, 소유자, 환경 또는 다른 기준 등 다양한 방식으로 리소스를 분류할 수 있습니다. 태그를 사용하면 다음이 가능합니다.

- 지정한 태그를 기반으로 리소스를 신속하게 식별합니다.
- AWS 청구서를 태그별로 분류하여 참조하십시오.
- 태그를 기반으로 Amazon Keyspaces 리소스에 대한 액세스를 제어합니다. 태그를 사용하는 IAM 정책 예제는 the section called “Amazon Keyspaces 태그 기반 권한 부여” 섹션을 참조하세요.

태깅은 아마존 Elastic Compute Cloud (Amazon EC2), 아마존 심플 스토리지 AWS 서비스 (Amazon S3), 아마존 키스페이스 등과 같은 서비스에서 지원됩니다. 효율적으로 태그를 지정하면 특정 태그와 연결하여 서비스 전체에 대해 생성된 보고서를 통해 비용을 분석할 수 있습니다.

태그 지정을 시작하려면 다음을 수행합니다.

1. Amazon Keyspaces에 대한 태그 지정 제한를 파악합니다.
2. Amazon Keyspaces의 태그 지정 작업를 사용하여 태그를 생성합니다.
3. 활성 태그별 AWS 비용을 Amazon Keyspace에 대한 비용 할당 보고서 추적하는 데 사용합니다.

끝으로, 최적화된 태깅 전략을 따르는 것이 좋습니다. 자세한 내용은 AWS 태깅 전략을 참조하세요.

Amazon Keyspaces에 대한 태그 지정 제한

각 태그는 사용자가 정의하는 키와 값으로 구성됩니다. 다음과 같은 제한 사항이 있습니다.

- 각 Amazon Keyspaces 키스페이스 또는 테이블은 동일한 키에 대해 한 가지 태그만을 사용합니다. 기존 태그(동일한 키)를 추가하는 경우 기존 태그 값이 새 값으로 업데이트됩니다.
- 키스페이스에 적용된 태그는 해당 키스페이스 내의 테이블에 자동으로 적용되지 않습니다. 키스페이스 및 모든 테이블에 동일한 태그를 적용하려면 각 리소스에 개별적으로 태그를 지정해야 합니다.
- 다중 리전 키스페이스 또는 테이블을 생성하면 생성 프로세스 중에 정의한 모든 태그가 모든 리전의 모든 키스페이스와 테이블에 자동으로 적용됩니다. `ALTER KEYSPACE` 또는 `ALTER TABLE`을 사용하여 기존 태그를 변경하면 변경하려는 리전의 키스페이스 또는 테이블에만 업데이트가 적용됩니다.
- 하나의 값은 태그 카테고리(키) 내에서 서술자 역할을 수행합니다. Amazon Keyspaces에서는 값이 비어 있거나 null일 수 없습니다.
- 태그 키와 값은 대/소문자를 구분합니다.
- 최대 키 길이는 유니코드 문자 128자입니다.
- 최대 값 길이는 유니코드 문자 256자입니다.
- 허용되는 문자는 문자, 공백, 숫자, 특수 문자(`+ - = . _ : /`)입니다.
- 리소스당 최대 태그 수는 50개입니다.
- AWS 할당 태그 이름과 값에는 사용자가 할당할 수 없는 `aws:` 접두사가 자동으로 할당됩니다. AWS 할당 태그 이름은 태그 제한인 50개에 포함되지 않습니다. 비용 할당 보고서에는 사용자가 지정한 태그 이름인 `user:` 접두사가 포함됩니다.
- 태그를 소급해서 적용할 수 없습니다.

Amazon Keyspaces의 태그 지정 작업

Amazon Keyspaces(Apache Cassandra용) 콘솔, AWS CLI 또는 Cassandra 쿼리 언어(CQL)를 사용하여 키스페이스 및 테이블의 태그를 추가, 나열, 편집 또는 삭제할 수 있습니다. 이러한 사용자 정의 태그를 활성화하면 AWS Billing and Cost Management 콘솔에 표시되어 비용 할당을 추적할 수 있습니다. 자세한 내용은 Amazon Keyspace에 대한 비용 할당 보고서 섹션을 참조하세요.

일괄 편집을 위해 콘솔에서 Tag Editor를 사용할 수 있습니다. 자세한 내용은 AWS Resource Groups 사용 설명서에서 Tag Editor 작업을 참조하세요.

주제

콘솔을 사용하여 신규 또는 기존 키스페이스 및 테이블에 태그 추가

Amazon Keyspaces 콘솔을 사용하여 생성한 신규 키스페이스 및 테이블에 태그를 추가할 수 있습니다. 기존 테이블의 태그를 추가, 편집 또는 삭제할 수도 있습니다.

키스페이스를 생성할 때 키스페이스에 태그를 지정하려면(콘솔)

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 Keyspaces를 선택한 다음 키스페이스 생성을 선택합니다.
3. 키스페이스 생성 페이지에서 키스페이스의 이름을 입력합니다. 태그에 대한 키 및 값을 입력하려면 새 태그 추가를 선택합니다.
4. 키스페이스 생성을 선택합니다.

테이블을 생성할 때 테이블에 태그를 지정하려면(콘솔)

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 테이블을 선택한 다음 테이블 생성을 선택합니다.
3. 테이블 세부 정보 섹션의 테이블 생성 페이지에서 키스페이스를 선택하고 테이블의 이름을 입력합니다.

4. 스키마 섹션에서 테이블의 스키마를 생성합니다.
5. 테이블 설정 섹션에서 설정 사용자 지정을 선택합니다.
6. 테이블 태그 - 옵션 섹션으로 이동한 다음 새 태그 추가를 선택하여 새 태그를 생성합니다.
7. 테이블 생성을 선택합니다.

기존 리소스에 태그를 지정하려면(콘솔)

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 Keyspaces 또는 Tables를 선택합니다.
3. 목록에서 키스페이스 또는 테이블을 선택합니다. 그런 다음 태그 관리를 선택하여 태그를 추가, 편집 또는 삭제합니다.

태그 구조에 대한 자세한 내용은 Amazon Keyspaces에 대한 태그 지정 제한 섹션을 참조하세요.

AWS CLI를 사용하여 신규 또는 기존 키스페이스 및 테이블에 태그 추가

이 섹션의 예제는 키스페이스와 테이블을 생성할 때 AWS CLI를 사용하여 태그를 지정하는 방법, 기존 리소스에서 태그를 추가하거나 제거하는 방법, 태그를 나열하는 방법을 보여 줍니다.

다음 예제에서는 태그를 사용하여 새 테이블을 생성하는 방법을 보여 줍니다. 명령은 이미 존재하는 키스페이스 myKeyspace에 테이블 myTable을 생성합니다. 참고로 명령은 가독성을 높이기 위해 여러 줄로 나누어져 있습니다.

```
aws keyspaces create-table --keyspace-name 'myKeyspace' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},{name=name,type=text},
{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --tags 'key=key1,value=val1' 'key=key2,value=val2'
```

다음 예제에서는 기존 테이블에 새 태그를 추가하는 방법을 보여 줍니다.

```
aws keyspaces tag-resource --resource-arn 'arn:aws:cassandra:us-east-1:111222333444:/
keyspace/myKeyspace/table/myTable' --tags 'key=key3,value=val3' 'key=key4,value=val4'
```

다음 예제에서는 지정된 리소스의 태그를 나열하는 방법을 보여 줍니다.

```
aws keyspaces list-tags-for-resource --resource-arn 'arn:aws:cassandra:us-
east-1:111222333444:/keyspace/myKeyspace/table/myTable'
```

마지막 명령의 출력은 다음과 같습니다.

```
{
    "tags": [
        {
            "key": "key1",
            "value": "val1"
        },
        {
            "key": "key2",
            "value": "val2"
        },
        {
            "key": "key3",
            "value": "val3"
        },
        {
            "key": "key4",
            "value": "val4"
        }
    ]
}
```

CQL을 사용하여 신규 또는 기존 키스페이스 및 테이블에 태그 추가

다음 예제에서는 CQL을 사용하여 키스페이스 및 테이블 생성 시 태그를 지정하는 방법, 기존 리소스에 태그를 지정하는 방법 및 태그를 읽는 방법을 보여 줍니다.

다음 예제에서는 태그를 사용하여 새 키스페이스를 생성합니다.

```
CREATE KEYSPACE mykeyspace WITH TAGS = {'key1':'val1', 'key2':'val2'} ;
```

다음 예제에서는 태그를 사용하여 새 테이블을 생성합니다.

```
CREATE TABLE mytable(...) WITH TAGS = {'key1':'val1', 'key2':'val2'};
```

문의 리소스에 다른 명령을 사용하여 태그를 지정하려면

```
CREATE KEYSPACE mykeyspace WITH REPLICATION = {'class': 'Simple Strategy'} AND TAGS
 = {'key1':'val1', 'key2':'val2'};
```

다음 예제에서는 기존 키스페이스 및 테이블에 태그를 추가하거나 제거하는 방법을 보여 줍니다.

```
ALTER KEYSPACE mykeyspace ADD TAGS {'key1':'val1', 'key2':'val2'};
```

```
ALTER TABLE mytable DROP TAGS {'key1':'val1', 'key2':'val2'};
```

리소스에 첨부된 태그를 읽으려면 다음 CQL 문을 사용합니다.

```
SELECT * FROM system_schema_mcs.tags WHERE valid_where_clause;
```

이 WHERE 절은 필수이며 다음 형식 중 하나여야 합니다.

- keyspace_name = **'mykeyspace'** AND resource_type = 'keyspace'
- keyspace_name = **'mykeyspace'** AND resource_name = **'mytable'**
- resource_id = *arn*

예제:

다음 쿼리는 키스페이스에 태그가 있는지 여부를 보여 줍니다.

```
SELECT * FROM system_schema_mcs.tags WHERE keyspace_name = 'mykeyspace' AND
 resource_type = 'keyspace';
```

쿼리의 출력은 다음과 같습니다.

```
resource_id                                                    | keyspace_name |
 resource_name | resource_type | tags
---------------------------------------------------------------+---------------
+---------------+---------------+------
arn:aws:cassandra:us-east-1:123456789:/keyspace/mykeyspace/      | mykeyspace    |
 mykeyspace    | keyspace      | {'key1': 'val1', 'key2': 'val2'}
```

다음 쿼리는 테이블의 태그를 보여 줍니다.

```
SELECT * FROM system_schema_mcs.tags WHERE keyspace_name = 'mykeyspace' AND
 resource_name = 'mytable';
```

쿼리의 출력은 다음과 같습니다.

```
resource_id                                                                  |
 keyspace_name | resource_name | resource_type | tags
------------------------------------------------------------------------------
+--------------+--------------+--------------+------
arn:aws:cassandra:us-east-1:123456789:/keyspace/mykeyspace/table/mytable    |
 mykeyspace    | mytable       | table         | {'key1': 'val1', 'key2': 'val2'}
```

Amazon Keyspace에 대한 비용 할당 보고서

AWS에서는 태그를 사용하여 비용 할당 보고서에서 리소스 비용을 구성합니다. AWS에서는 두 가지 비용 할당 태그 유형을 제공합니다.

- AWS 생성 태그 AWS는 사용자를 위해 태그를 정의, 생성 및 적용합니다.
- 사용자 정의 태그로, 사용자가 태그를 정의, 생성 또는 적용합니다.

두 유형의 태그 모두 개별적으로 활성화해야만 Cost Explorer나 비용 할당 보고서에 표시됩니다.

AWS 생성 태그 활성화 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/billing/home#/에서 Billing and Cost Management 콘솔을 엽니다.
2. 탐색 창에서 비용 할당 태그를 선택합니다.
3. AWS-Generated Cost Allocation Tags(AWS 생성 비용 할당 태그)에서 Activate(활성화)를 선택합니다.

사용자 정의 태그 활성화 방법

1. AWS Management Console에 로그인한 후 https://console.aws.amazon.com/billing/home#/에서 Billing and Cost Management 콘솔을 엽니다.

2. 탐색 창에서 비용 할당 태그를 선택합니다.
3. 사용자 정의 비용 할당 태그에서 활성화를 선택합니다.

태그를 생성하여 활성화한 후 AWS에서는 사용 내역 및 비용별로 집계한 태그를 사용하여 비용 할당 보고서를 만듭니다. 비용 할당 보고서에는 각 결제 기간의 모든 AWS 비용이 포함되어 있습니다. 보고서에 태그가 지정된 리소스와 태그가 지정되지 않은 리소스가 모두 포함되어 있어 리소스에 대한 요금을 알아보기 쉽게 정리할 수 있습니다.

> **Note**
>
> 현재 Amazon Keyspaces에서 이전한 모든 데이터에 대해 비용 할당 보고서의 태그로 구분되지는 않습니다.

자세한 내용은 비용 할당 태그 사용을 참조하세요.

Amazon Keyspaces를 사용한 설계 및 아키텍처 설계 모범 사례

이 섹션을 통해 Amazon Keyspaces를 이용할 때 성능을 극대화하고 처리량 비용을 최소화할 수 있는 방법을 신속히 찾으십시오.

목차

Amazon Keyspaces NoSQL 설계

Amazon Keyspaces와 같은 NoSQL 데이터베이스 시스템은 데이터 관리에 키-값 페어 또는 문서 스토리지와 같은 대체 모델을 사용합니다. 관계형 데이터베이스 관리 시스템을 Amazon Keyspaces와 같은 NoSQL 데이터베이스 시스템으로 교체할 때 주요 차이점과 설계에 있어 특정 접근법을 이해하는 것이 중요합니다.

주제

관계형 데이터 설계와 NoSQL의 차이점

관계형 데이터베이스 시스템(RDBMS)과 NoSQL 데이터베이스는 각기 다른 장단점을 갖고 있습니다.

- RDBMS에서는 데이터를 유연하게 쿼리할 수 있지만, 쿼리 비용이 상대적으로 높으며 트래픽이 많은 상황에서는 확장성이 떨어집니다(the section called “데이터 모델링” 참조).
- Amazon Keyspaces와 같은 NoSQL 데이터베이스에서는 몇 가지 방법으로 데이터를 효율적으로 쿼리할 수 있지만, 그 외에는 쿼리 비용이 높고 속도가 느립니다.

이런 차이가 두 시스템의 데이터베이스 설계를 다르게 만듭니다.

- RDBMS의 경우, 세부적인 구현이나 성능을 걱정하지 않고 유연성을 목적으로 설계합니다. 일반적으로 쿼리 최적화가 스키마 설계에 영향을 미치지 않지만, 정규화가 중요합니다.
- Amazon Keyspaces의 경우, 가장 중요하고 범용적인 쿼리를 가능한 빠르고 저렴하게 수행할 수 있도록 스키마를 설계합니다. 사용자의 데이터 구조는 사용자 비즈니스 사용 사례의 특정 요구 사항에 적합하도록 만듭니다.

NoSQL 설계의 두 가지 주요 개념

NoSQL 설계에는 RDBMS 설계와 다른 사고 방식이 요구됩니다. RDBMS의 경우, 액세스 패턴을 생각하지 않고 정규화된 데이터 모델을 생성할 수 있습니다. 그런 후 나중에 새로운 질문과 쿼리에 대한 요구 사항이 생길 때 이를 확장할 수 있습니다. 각 데이터 유형을 고유의 테이블에 구성할 수 있습니다.

NoSQL 설계의 차이점

- 대조적으로 Amazon Keyspaces의 경우 대답해야 할 질문을 모르기 전까지는 스키마 설계를 시작할 수 없습니다. 사전에 비즈니스 문제와 애플리케이션 사용 사례를 이해해야 합니다.
- Amazon Keyspaces 애플리케이션에서는 가능한 적은 수의 테이블을 유지해야 합니다. 테이블 수가 적을수록 확장성이 향상되고 권한 관리가 줄어들며 Amazon Keyspaces 애플리케이션의 오버헤드가 감소합니다. 백업 비용도 전반적으로 낮게 유지할 수 있습니다.

NoSQL 설계에 접근

Amazon Keyspaces 애플리케이션 설계의 첫 번째 단계는 시스템이 충족해야 하는 특정 쿼리 패턴을 파악하는 것입니다.

특히 시작을 하기 앞서 애플리케이션 액세스 패턴에서 3가지 기본적인 속성을 이해하는 것이 중요합니다.

- 데이터 크기: 저장해야 할 데이터의 양과 한 번에 요청할 데이터의 양을 알면 가장 효과적으로 데이터를 파티션(분할)하는 방법을 결정할 수 있습니다.
- 데이터 모양: 쿼리를 처리할 때 데이터를 변화시키는 대신(RDBMS 시스템의 방식), NoSQL 데이터베이스는 데이터베이스의 모양이 쿼리 대상과 일치하도록 데이터를 구성합니다. 이는 속도와 확장성 향상에 중요한 요소입니다.
- 데이터 속도: Amazon Keyspaces는 프로세스 쿼리에 사용할 수 있는 물리적 파티션의 수를 늘리고, 해당 파티션에 효율적으로 데이터를 배포해 조정합니다. 사전에 피크 쿼리 로드를 알면 I/O 용량을 가장 효과적으로 사용할 수 있는 데이터 파티션(분할) 방법을 결정하는 데 도움이 될 수 있습니다.

특정 쿼리 요구 사항을 파악한 후, 성능을 결정하는 일반 원칙에 따라 데이터를 구성할 수 있습니다.

- 관련 데이터를 함께 유지합니다. 20년 전의 라우팅 테이블 최적화 연구에 따르면, 응답 시간 향상에 가장 중요한 한 가지 요소는 관련 데이터를 한 장소에 유지하는 "Locality of reference"입니다. 이는 현재 NoSQL 시스템에도 정확히 적용됩니다. 관련 데이터를 가까이 유지하는 것이 비용과 성능에 큰 영향을 미치기 때문입니다. 관련 데이터 항목을 여러 테이블로 분산시키는 대신 NoSQL 시스템에 가능한 가깝게 관련 항목을 유지해야 합니다.

  대체로 Amazon Keyspaces 애플리케이션에서는 가능한 적은 수의 테이블을 유지해야 합니다.

  단, 볼륨이 많은 시계열 데이터가 관여된 경우나 액세스 패턴이 아주 다른 데이터 세트는 해당되지 않습니다. 통상 반전된 인덱스의 단일 테이블로 간단한 쿼리를 활성화시켜 사용자의 애플리케이션에 필요한 복잡한 계층적 데이터 구조를 생성 및 검색할 수 있습니다.
- 정렬 순서를 사용합니다. 핵심 설계가 함께 정렬할 것을 요구하는 경우, 관련 항목을 그룹으로 묶어 효율적으로 쿼리할 수 있습니다. 이는 중요한 NoSQL 설계 전략입니다.
- 쿼리를 분산합니다. 많은 볼륨의 쿼리를 데이터베이스의 특정 부분에 집중시키지 않는 것이 중요합니다. I/O 용량을 초과할 수 있기 때문입니다. 대신 트래픽을 가능한 여러 파티션으로 분산시켜 '핫 스팟'이 방지되도록 데이터 키를 설계해야 합니다.

이런 일반 원칙은 Amazon Keyspaces에서 데이터를 효율적으로 모델링하기 위해 사용할 수 있는 범용 설계 패턴 가운데 일부로 전환됩니다.

Amazon Keyspaces(Apache Cassandra용)에 대한 클라이언트 드라이버 연결

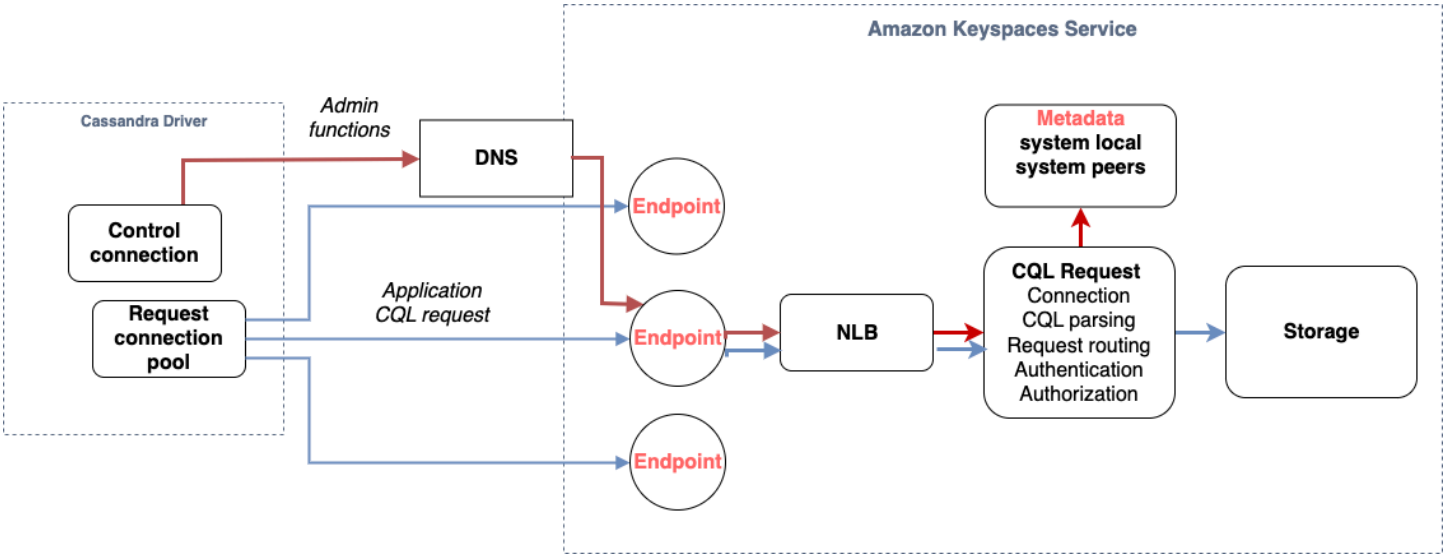
원하는 기존 Apache Cassandra 클라이언트 드라이버를 사용하여 Amazon Keyspaces와 통신할 수 있습니다. Amazon Keyspaces는 서버리스 서비스이므로 애플리케이션의 처리량 요구 사항에 맞게 클라이언트 드라이버의 연결 구성을 최적화하는 것이 좋습니다. 이 주제에서는 애플리케이션에 필요한 연결 수를 계산하는 방법과 연결 모니터링 및 오류 처리를 비롯한 모범 사례를 소개합니다.

주제

Amazon Keyspaces에서의 연결 작동 방식

이 섹션에서는 Amazon Keyspaces에서 클라이언트 드라이버 연결이 작동하는 방식에 대한 개요를 제공합니다. Cassandra 클라이언트 드라이버를 잘못 구성하면 Amazon Keyspaces에서 `PerConnectionRequestExceeded` 이벤트가 발생할 수 있으므로 이러한 연결 오류를 방지하려면 클라이언트 드라이버 구성에서 적절한 양의 연결을 구성해야 합니다.

Amazon Keyspace에 연결할 때 드라이버에는 초기 연결을 설정하기 위한 시드 엔드포인트가 필요합니다. Amazon Keyspaces는 DNS를 사용하여 사용 가능한 여러 엔드포인트 중 하나로 초기 연결을 라우팅합니다. 엔드포인트는 네트워크 로드 밸런서에 연결되며, 네트워크 로드 밸런서는 플릿의 요청 핸들러 중 하나로의 연결을 설정합니다. 초기 연결이 설정되면 클라이언트 드라이버는 `system.peers` 테이블에서 사용 가능한 모든 엔드포인트에 대한 정보를 수집합니다. 이 정보를 사용하여 클라이언트 드라이버는 나열된 엔드포인트에 대한 추가 연결을 생성할 수 있습니다. 클라이언트 드라이버가 만들 수 있는 연결 수는 클라이언트 드라이버 설정에 지정된 로컬 연결 수에 따라 제한됩니다. 기본적으로 대부분의 클라이언트 드라이버는 엔드포인트당 하나의 연결을 설정하고, Cassandra에 대한 연결 풀을 설정하고, 연결 풀 전체에 걸쳐 쿼리를 로드 밸런싱합니다. 동일한 엔드포인트에 여러 연결을 설정할 수 있지만 네트워크 로드 밸런서 뒤에서는 여러 개의 다른 요청 핸들러에 연결될 수 있습니다. 퍼블릭 엔드포인트를 통해 연결하는 경우 `system.peers` 테이블에 나열된 9개 엔드포인트 각각에 하나씩 연결을 설정하면 서로 다른 요청 핸들러에 9개의 연결이 생성됩니다.



Amazon Keyspaces에서 연결을 구성하는 방법

Amazon Keyspaces는 TCP 연결당 초당 최대 3,000개의 CQL 쿼리를 지원합니다. 드라이버가 설정할 수 있는 연결 수에는 제한이 없으므로 오버헤드, 트래픽 버스트, 효과적인 로드 밸런싱을 위해 연결당 초당 500개의 CQL 요청만 대상으로 하는 것이 좋습니다. 다음 단계에 따라 드라이버 연결이 애플리케이션의 요구 사항에 맞게 올바르게 구성되었는지 확인하세요.

드라이버가 연결 풀에서 유지 관리하는 IP 주소당 연결 수를 늘리십시오.

- 대부분의 Cassandra 드라이버는 Cassandra에 연결 풀을 설정하고 연결 풀 전체에 걸쳐 쿼리를 로드 밸런싱합니다. 대부분의 드라이버의 기본 동작은 각 엔드포인트마다 하나의 연결을 설정하는 것입니다. Amazon Keyspaces는 드라이버에 9개의 피어 IP 주소를 노출하므로 대부분의 드라이버의 기본 동작을 기준으로 하면 연결 수가 9개입니다. Amazon Keyspaces는 TCP 연결당 초당 최대 3,000개의 CQL 쿼리를 지원하므로, 기본 설정을 사용하는 드라이버의 최대 CQL 쿼리 처리량은 초당 27,000개의 CQL 쿼리입니다. 드라이버의 기본 설정을 사용하는 경우 하나의 연결에서 최대 CQL 쿼리 처리량인 초당 3,000개가 넘는 CQL 쿼리를 처리해야 할 수 있습니다. 이로 인해 `PerConnectionRequestExceeded` 이벤트가 발생할 수 있습니다.
- `PerConnectionRequestExceeded` 이벤트를 방지하려면 엔드포인트별로 추가 연결을 생성하여 처리량을 분산하도록 드라이버를 구성해야 합니다.
- Amazon Keyspaces의 모범 사례에서는 각 연결이 초당 500개의 CQL 쿼리를 지원할 수 있다고 가정합니다.
- 즉, 사용 가능한 9개 엔드포인트에 분산된 초당 약 27,000개의 CQL 쿼리를 지원해야 하는 프로덕션 애플리케이션의 경우 엔드포인트당 6개의 연결을 구성해야 합니다. 이렇게 하면 각 연결에서 초당 500개 이하의 요청을 처리할 수 있습니다.

애플리케이션의 요구 사항에 따라 드라이버에 대해 구성해야 하는 IP 주소당 연결 수를 계산하십시오.

애플리케이션에 엔드포인트별로 구성해야 하는 연결 수를 결정하려면 다음 예제를 고려해 보십시오. 10,000개의 INSERT, 5,000개의 SELECT, 5,000개의 DELETE 작업으로 구성된 초당 20,000개의 CQL 쿼리를 지원해야 하는 애플리케이션이 있습니다. 이 Java 애플리케이션은 Amazon Elastic Container Service(Amazon ECS)의 세 개 인스턴스에서 실행되며, 각 인스턴스마다 Amazon Keyspace에 대한 하나의 세션을 설정합니다. 드라이버에 구성해야 하는 연결 수를 추정하는 데 사용할 수 있는 계산에 사용되는 입력은 다음과 같습니다.

1. 애플리케이션이 지원해야 하는 초당 요청 수.
2. 사용 가능한 인스턴스 수에서 유지 관리 또는 장애 발생을 고려하여 하나를 뺀 수.
3. 사용 가능한 엔드포인트 수. 퍼블릭 엔드포인트를 통해 연결하는 경우 사용 가능한 엔드포인트는 9개입니다. VPC 엔드포인트를 사용하는 경우 사용 가능한 엔드포인트 수는 리전에 따라 2~5개입니다.
4. Amazon Keyspaces 모범 사례에서는 연결당 초당 500개의 CQL 쿼리를 사용합니다.
5. 결과를 반올림합니다.

이 예제의 공식은 다음과 같습니다.

```
20,000 CQL queries / (3 instances - 1 failure) / 9 public endpoints / 500 CQL queries
 per second = ROUND(2.22) = 3
```

이 계산에 따라 드라이버 구성에서 엔드포인트당 로컬 연결 3개를 지정해야 합니다. 원격 연결의 경우 엔드포인트당 하나의 연결만 구성합니다.

Amazon Keyspaces에서 VPC 엔드포인트를 통해 연결을 구성하는 방법

프라이빗 VPC 엔드포인트를 통해 연결하는 경우 사용 가능한 엔드포인트가 3개 있을 가능성이 높습니다. VPC 엔드포인트 수는 가용 영역 수 및 할당된 VPC의 서브넷 수에 따라 지역별로 다를 수 있습니다. 미국 동부(버지니아 북부) 지역에는 5개의 가용 영역이 있으며 Amazon Keyspaces 엔드포인트를 최대 5개까지 보유할 수 있습니다. 미국 서부(캘리포니아 북부) 지역에는 2개의 가용 영역이 있으며 Amazon Keyspaces 엔드포인트를 최대 2개까지 보유할 수 있습니다. 엔드포인트 수에 따라 규모가 달라지지는 않지만 드라이버 구성에서 설정해야 하는 연결 수가 늘어납니다. 다음 예제를 살펴보세요. 애플리케이션은 20,000개의 CQL 쿼리를 지원해야 하며 각 인스턴스가 Amazon Keyspaces에 대한 단일 세션을 설정하는 Amazon ECS의 세 개 인스턴스에서 실행됩니다. 유일한 차이는 서로 다른 엔드포인트에서 사용할 수 있는 엔드포인트의 수입니다. AWS 리전

미국 동부(버지니아 북부) 리전에서 필요한 연결:

```
20,000 CQL queries / (3 instances - 1 failure) / 5 private VPC endpoints / 500 CQL
 queries per second = 4 local connections
```

미국 서부(캘리포니아 북부) 리전에서 필요한 연결:

```
20,000 CQL queries / (3 instances - 1 failure) / 2 private VPC endpoints / 500 CQL
 queries per second = 10 local connections
```

> **Important**
>
> 프라이빗 VPC 엔드포인트를 사용하는 경우 Amazon Keyspaces가 사용 가능한 VPC 엔드포인트를 동적으로 검색하고 `system.peers` 테이블에 데이터를 입력하려면 추가 권한이 필요합니다. 자세한 정보는 the section called “인터페이스 VPC 엔드포인트 정보로 system.peers 테이블 항목 채우기”을 참조하세요.

다른 AWS 계정엔드포인트를 사용하는 프라이빗 VPC 엔드포인트를 통해 Amazon Keyspaces에 액세스하는 경우 Amazon Keyspaces 엔드포인트가 하나만 표시될 수 있습니다. 다시 말하지만 이는 Amazon Keyspaces에 대한 가능한 처리량 규모에는 영향을 미치지 않지만 드라이버 구성의 연결 수를 늘려야 할 수도 있습니다. 이 예제는 사용 가능한 단일 엔드포인트에 대한 동일한 계산을 보여줍니다.

```
20,000 CQL queries / (3 instances - 1 failure) / 1 private VPC endpoints / 500 CQL
 queries per second = 20 local connections
```

공유 VPC를 사용하여 Amazon Keyspaces에 대한 크로스 계정 액세스를 하는 방법에 대한 자세한 내용은 the section called “공유 VPC에서의 크로스 계정 액세스” 섹션을 참조하세요.

Amazon Keyspaces에서의 연결 모니터링 방식

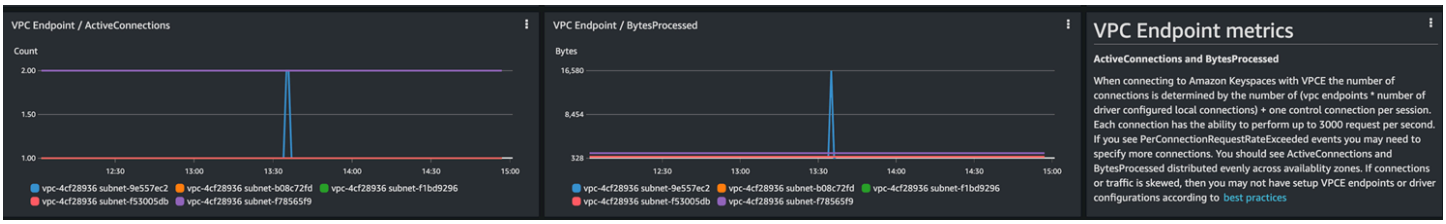
애플리케이션이 연결된 엔드포인트 수를 식별할 수 있도록 `system.peers` 테이블에 검색된 피어 수를 기록할 수 있습니다. 다음 예제는 연결이 설정된 후 피어 수를 인쇄하는 Java 코드의 예제입니다.

```
ResultSet result = session.execute(new SimpleStatement("SELECT * FROM system.peers"));

logger.info("number of Amazon Keyspaces endpoints:" + result.all().stream().count());
```

> **Note**
>
> CQL 콘솔 또는 AWS 콘솔은 VPC 내에 배포되지 않으므로 퍼블릭 엔드포인트를 사용합니다. 따라서 VPCE 외부에 있는 애플리케이션에서 `system.peers` 쿼리를 실행하면 피어가 9개가 되는 경우가 많습니다. 각 피어의 IP 주소를 인쇄하는 것도 유용할 수 있습니다.

VPCE Amazon 지표를 설정하여 VPC 엔드포인트를 사용할 때 피어 수를 관찰할 수도 있습니다. CloudWatch CloudWatch에서는 VPC 엔드포인트에 설정된 연결 수를 확인할 수 있습니다. Cassandra 드라이버는 각 엔드포인트에 대해 CQL 쿼리를 전송하기 위한 연결과 시스템 테이블 정보를 수집하기 위한 제어 연결을 설정합니다. 아래 이미지는 드라이버 설정에서 1개 연결을 구성하여 Amazon Keyspace에 연결한 후의 VPC 엔드포인트 CloudWatch 지표를 보여줍니다. 이 지표는 제어 연결 1개와 연결 5개(여러 가용 영역에 걸쳐 엔드포인트당 1개)로 구성된 6개의 활성 연결을 보여줍니다.



CloudWatch 그래프를 사용하여 연결 수 모니터링을 시작하려면 Amazon Keyspaces AWS CloudFormation 템플릿 GitHub 리포지토리에서 제공되는 이 템플릿을 배포하면 됩니다.

Amazon Keyspaces의 연결 오류를 처리하는 방법

연결 할당량당 요청 3,000개를 초과하는 경우 Amazon Keyspaces는 `PerConnectionRequestExceeded` 이벤트를 반환하고 Cassandra 드라이버는 `WriteTimeout` 또는 `ReadTimeout` 예외를 수신합니다. Cassandra 재시도 정책 또는 애플리케이션에서 지수 백오프를 적용하여 이 예외를 재시도해야 합니다. 추가 요청을 보내지 않으려면 지수 백오프를 제공해야 합니다.

기본 재시도 정책은 쿼리 계획에서 `try next host`을(를) 시도합니다. Amazon Keyspaces는 VPC 엔드포인트에 연결할 때 사용 가능한 엔드포인트가 1~3개 있을 수 있으므로 애플리케이션 로그에 `NoHostAvailableException`, `WriteTimeout`, `ReadTimeout` 예외가 표시될 수도 있습니다. Amazon Keyspaces에서 제공하는 재시도 정책을 사용할 수 있습니다. 이 정책은 여러 연결에 걸친 동일한 엔드포인트에서 재시도합니다.

Amazon Keyspaces Java 코드 예제 리포지토리에서 Java에 GitHub 대한 지수 재시도 정책의 예를 찾을 수 있습니다. GitHub의 Amazon Keyspaces 코드 예제 리포지토리에서 추가 언어 예제를 찾을 수 있습니다.

Amazon Keyspaces(Apache Cassandra용)에서의 데이터 모델링

이 주제에서는 Amazon Keyspaces(Apache Cassandra용)의 데이터 모델링 개념을 소개합니다. 이 섹션에서는 애플리케이션의 데이터 액세스 패턴에 맞는 데이터 모델 설계를 위한 권장 사항을 알아봅니다. Amazon Keyspaces를 사용할 때 데이터 모델링 모범 사례를 구현하면 성능이 향상되고 처리량 비용이 최소화됩니다.

NoSQL Workbench를 사용하면 데이터 모델을 보다 쉽게 시각화하고 설계할 수 있습니다.

주제

- Amazon Keyspaces에서 파티션 키를 효과적으로 사용하는 방법

Amazon Keyspaces에서 파티션 키를 효과적으로 사용하는 방법

Amazon Keyspaces 테이블의 각 행을 고유하게 식별하는 프라이머리 키는 데이터가 저장되는 파티션을 결정하는 하나 이상의 파티션 키 열과 파티션 내에서 데이터가 클러스터링되고 정렬되는 방식을 정의하는 하나 이상의 선택적 클러스터링 열로 구성될 수 있습니다.

파티션 키는 데이터가 저장되는 파티션 수와 파티션에 데이터가 분산되는 방식을 설정하므로 파티션 키를 어떻게 선택하느냐에 따라 쿼리 성능이 상당히 달라질 수 있습니다. 일반적으로 디스크의 모든 파티션에서 균일하게 작동하도록 애플리케이션을 설계해야 합니다.

애플리케이션의 읽기 및 쓰기 작업을 모든 파티션에 균등하게 분배하면 처리량 비용을 최소화하는 데 도움이 되며, 이는 온디맨드뿐 아니라 프로비저닝된 읽기/쓰기 용량 모드에도 적용됩니다. 예를 들어 프로비저닝된 용량 모드를 사용하는 경우, 애플리케이션에 필요한 액세스 패턴을 결정하고, 각 테이블에 필요한 읽기 용량 단위(RCU)와 쓰기 용량 단위(WCU)의 총계를 추정할 수 있습니다. Amazon Keyspaces는 특정 파티션 키에 대한 트래픽이 3,000RCU 및 1,000WCU를 초과하지 않는다면, 는 자동으로 사용자가 프로비저닝한 처리량을 사용해 액세스 패턴을 지원합니다.

Amazon Keyspaces는 버스트 용량 을 제공하여 파티션당 처리량 프로비저닝에서 추가적인 유연성을 제공합니다. 자세한 내용은 the section called “버스트 용량” 섹션을 참조하세요.

주제

- 쓰기 샤딩을 사용해 Amazon Keyspaces에서 워크로드를 고르게 배포

쓰기 샤딩을 사용해 Amazon Keyspaces에서 워크로드를 고르게 배포

Amazon Keyspaces의 파티션에 더 효과적으로 쓰기 작업을 배포하는 방법 중 하나는 공간 확장입니다. 여러 방법을 사용할 수 있습니다. 여러 파티션에 행을 분배하기 위해 난수를 쓰는 파티션 키 열을 추가할 수 있습니다. 또는 쿼리하는 항목을 기반으로 계산된 숫자를 사용할 수 있습니다.

복합 파티션 키와 무작위 값을 사용한 샤딩

파티션에 부하를 더 균등하게 분산하기 위한 한 가지 전략은 난수를 기록할 파티션 키 열을 추가하는 것입니다. 그러면 더 큰 공간으로 쓰기를 무작위화 할 수 있습니다.

예를 들어 날짜를 나타내는 단일 파티션 키가 있는 테이블은 다음과 같습니다.

```
CREATE TABLE IF NOT EXISTS tracker.blogs (
   publish_date date,
   title text,
   description int,
   PRIMARY KEY (publish_date));
```

이 테이블을 여러 파티션에 더 균등하게 분배하려면 난수를 저장하는 파티션 키 열 shard를 추가하면 됩니다. 예:

```
CREATE TABLE IF NOT EXISTS tracker.blogs (
   publish_date date,
   shard int,
   title text,
   description int,
   PRIMARY KEY ((publish_date, shard)));
```

데이터를 삽입할 때 shard 열의 1와(과) 200 사이의 난수를 선택할 수 있습니다. 이렇게 하면 (2020-07-09, 1), (2020-07-09, 2), (2020-07-09, 200)까지의 복합 파티션 키 값이 생성됩니다. 파티션 키를 임의 지정했기 때문에, 각 날짜의 테이블에 대한 쓰기가 모든 파티션 키 값에 고르게 분산됩니다. 이는 병렬 처리를 개선하고 전반적인 처리량을 높입니다.

하지만 지정된 날짜의 모든 행을 읽으려면, 모든 샤드의 행을 쿼리해 결과를 병합해야 합니다. 예를 들어 먼저 파티션 키 값 (2020-07-09, 1)에 대한 SELECT문을 생성합니다. 그런 다음 (2020-07-09, 2)에서 (2020-07-09, 200)까지에 대해 또 다른 SELECT문을 생성합니다. 마지막으로 애플리케이션은 모든 SELECT문 결과를 병합해야 합니다.

복합 파티션 키와 계산된 값을 사용한 샤딩

임의 지정 전략으로 쓰기 처리량을 크게 향상시킬 수 있습니다. 하지만 행을 작성할 때 shard 열에 어떤 값이 기록되었는지 모르기 때문에 특정 행을 읽기가 어렵습니다. 개별 행을 더 쉽게 읽을 수 있도록 만들려면 다른 전략을 사용합니다. 난수(임의의 수)를 사용해 여러 파티션으로 행을 배포하는 대신, 쿼리하고 싶은 내용을 토대로 계산할 수 있는 수를 사용합니다.

테이블의 파티션 키에 오늘 날짜를 사용한 앞의 예를 가정하겠습니다. 이제 각 행에 액세스가 가능한 title 열이 있고, 날짜에 더해 제목으로 행을 찾아야 하는 경우가 많다고 가정하겠습니다. 애플리케이션은 테이블에 행을 쓰기 전에 제목에 따라 해시 값을 계산하고 shard 열에 이를 입력할 수 있습니다. 계산 결과 임의 지정 전략에서 생산되는 결과와 유사하게 골고루 배포된 1-200 사이의 숫자가 생성될 것입니다.

제목의 문자에 대한 UTF-8 코드 포인트 값의 제품이나 모듈로 200 + 1과 같이 간단한 계산으로 충분합니다. 그러면 복합 파티션 키 값은 날짜와 계산 결과의 조합이 됩니다.

이러한 전략을 통해 쓰기가 파티션-키 값 및 물리적 파티션에 고르게 분산됩니다. 특정 행과 날짜에서 SELECT문을 손쉽게 수행할 수 있습니다. 특정 title 값에 대한 파티션-키 값을 계산할 수 있기 때문입니다.

지정된 날의 모든 열을 읽으려면 각 (2020-07-09, N)(여기서 N은 1~200) 키를 SELECT해야 하며, 애플리케이션은 모든 결과를 병합해야 합니다. 장점은 모든 워크로드를 취하는 하나의 '핫' 파티션 키 값이 발생하지 않도록 만든다는 것입니다.

Amazon Keyspaces 테이블 비용 최적화

이 섹션에서는 기존 Amazon Keyspaces 테이블의 비용을 최적화하는 방법에 대한 모범 사례를 다룹니다. 다음 전략을 검토하여 요구 사항에 가장 적합한 비용 최적화 전략을 확인하고 반복적으로 접근해야 합니다. 각 전략은 비용에 영향을 미칠 수 있는 요소, 비용 최적화 기회를 찾는 방법, 비용 절감을 위해 이러한 모범 사례를 구현하는 방법에 대한 규범적 지침의 개요를 제공합니다.

주제

- 테이블 수준에서 비용 평가
- 테이블 용량 모드 평가
- 테이블의 Application Auto Scaling 설정 평가
- 미사용 리소스 식별
- 테이블 사용 패턴 평가

- 적절한 규모의 프로비저닝을 위해 프로비저닝된 용량 평가

테이블 수준에서 비용 평가

에 있는 Cost Explorer 도구를 AWS Management Console 사용하면 읽기, 쓰기, 스토리지 및 백업 비용과 같은 유형별로 비용을 분류하여 확인할 수 있습니다. 월별 또는 일과 같은 기간별로 요약된 비용도 확인할 수 있습니다.

Cost Explorer에서 흔히 발생하는 문제 중 하나는 특정 테이블 하나에 대한 비용만 쉽게 검토할 수 없다는 것입니다. Cost Explorer에서는 특정 테이블의 비용을 기준으로 필터링하거나 그룹화할 수 없기 때문입니다. Amazon Keyspaces 콘솔의 테이블 모니터 탭에서 각 테이블의 청구 가능 테이블 크기 (바이트) 측정치를 확인할 수 있습니다. 테이블당 추가 비용 관련 정보가 필요한 경우 이 섹션에서는 Cost Explorer에서 태그 지정을 사용하여 개별 테이블 비용 분석을 수행하는 방법을 보여줍니다.

주제

- 단일 Amazon Keyspaces 테이블의 비용을 확인하는 방법
- Cost Explorer의 기본 보기
- Cost Explorer에서 테이블 태그를 사용하고 적용하는 방법

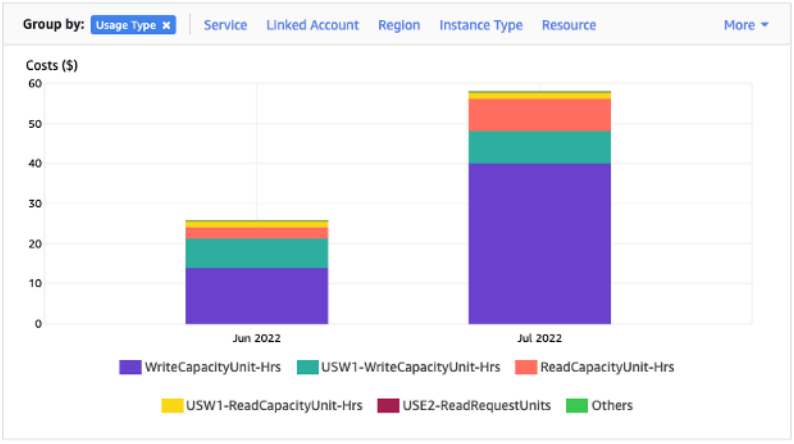
단일 Amazon Keyspaces 테이블의 비용을 확인하는 방법

콘솔에서 기본 키 스키마, 청구 가능한 테이블 크기, 용량 관련 지표 등 Amazon Keyspaces 테이블에 대한 기본 정보를 확인할 수 있습니다. 테이블 크기를 사용하여 테이블의 월별 스토리지 비용을 계산할 수 있습니다. 예를 들어, 미국 동부 (버지니아 북부) 의 경우 GB당 0.25 USD입니다. AWS 리전

테이블이 프로비저닝된 용량 모드를 사용하는 경우 현재 읽기 용량 단위(RCU) 및 쓰기 용량 단위(WCU) 설정도 반환됩니다. 이 정보를 사용하여 테이블의 현재 읽기 및 쓰기 비용을 계산할 수 있습니다. 특히 Amazon Keyspaces 자동 크기 조정으로 테이블을 구성한 경우 이러한 비용은 변경될 수 있습니다.

Cost Explorer의 기본 보기

Cost Explorer의 기본 보기는 처리량(throughput) 및 스토리지와 같은 사용된 리소스 비용을 보여주는 차트를 제공합니다. 월별 또는 일별 합계와 같이 기간별로 비용을 그룹화하도록 선택할 수 있습니다. 스토리지, 읽기, 쓰기 및 기타 카테고리의 비용도 세분화하여 비교할 수 있습니다.



Cost Explorer에서 테이블 태그를 사용하고 적용하는 방법

기본적으로 Cost Explorer는 여러 테이블의 비용을 합산하므로 특정 테이블에 대한 비용 요약을 제공하지 않습니다. 하지만 AWS 리소스 태깅을 사용하여 메타데이터 태그로 각 테이블을 식별할 수 있습니다. 태그는 프로젝트 또는 부서에 속한 모든 리소스를 식별하는 등 다양한 용도로 사용할 수 있는 키-값 쌍입니다. 자세한 정보는 the section called “태그 사용하기”을 참조하세요.

이 예시에서는 이름이 MyTable지정된 테이블을 사용합니다.

1. key는 table_name이고 값은 로 태그를 설정합니다. MyTable
2. Cost Explorer 탐색기에서 태그를 활성화한 후 태그 값을 필터링하여 각 테이블의 비용을 더 잘 파악할 수 있습니다.

> **Note**
>
> Cost Explorer에 태그가 표시되려면 하루나 이틀이 걸릴 수 있습니다.

콘솔에서 직접 메타데이터 태그를 설정하거나 CQL AWS CLI, the 또는 SDK를 사용하여 프로그래밍 방식으로 메타데이터 태그를 설정할 수 있습니다. AWS 조직의 새 테이블 생성 프로세스의 일부로 table_name 태그를 설정하도록 요구하는 것이 좋습니다. 자세한 정보는 the section called “Amazon Keyspace에 대한 비용 할당 보고서”을 참조하세요.

테이블 용량 모드 평가

이 섹션에서는 Amazon Keyspaces 테이블에 적절한 용량 모드를 선택하는 방법을 간략히 살펴봅니다. 각 모드는 처리량(throughput) 변화에 대한 대응력 및 사용량 청구 방식 측면에서 다양한 워크로드의 요구 사항을 충족하도록 조정됩니다. 결정을 내릴 때 이러한 요소들의 균형을 맞춰야 합니다.

주제

사용할 수 있는 테이블 용량 모드

Amazon Keyspaces 테이블을 생성할 때 온디맨드 또는 프로비저닝된 용량 모드를 선택해야 합니다. 자세한 정보는 the section called “읽기/쓰기 용량 모드”을 참조하세요.

온디맨드 용량 모드

온디맨드 용량 모드는 Amazon Keyspaces 테이블의 용량을 계획하거나 프로비저닝할 필요가 없도록 설계되었습니다. 이 모드에서 테이블은 리소스를 스케일 업 또는 다운할 필요 없이 요청을 즉시 수용합니다(테이블의 이전 최대 처리량(throughput)의 최대 2배).

온디맨드 테이블은 테이블에 대한 실제 요청 수를 계산하여 청구되므로 프로비저닝된 것이 아니라 사용한 만큼만 비용을 지불하면 됩니다.

프로비저닝된 용량 모드

프로비저닝된 용량 모드는 테이블이 요청에 사용할 수 있는 용량을 직접 또는 Application Auto Scaling의 도움으로 정의할 수 있는 보다 전통적인 모델입니다. 지정된 시간에 테이블에 대해 특정 용량이 프로비저닝되기 때문에 비용은 요청 수가 아닌 프로비저닝된 용량을 기준으로 결제됩니다. 할당된 용량을 초과하면 테이블이 요청을 거부하고 애플리케이션 사용자의 경험을 감소시킬 수도 있습니다.

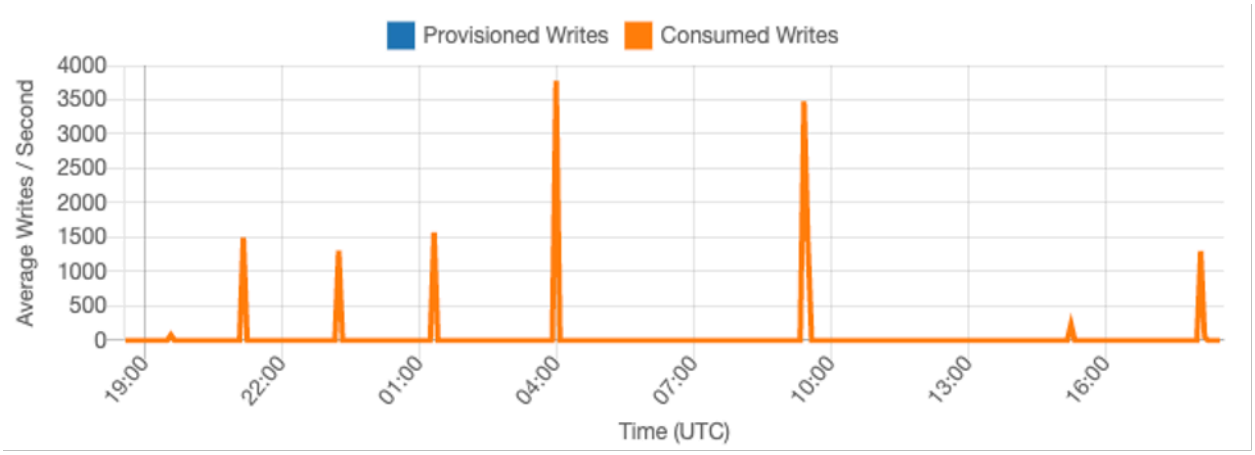
프로비저닝된 용량 모드에서는 두 가지를 모두 달성하기 위해 테이블을 과도하게 프로비저닝하지 않거나 과소 프로비저닝하지 않는 균형이 필요하며, 처리량 용량 부족 오류가 발생하지 않으며, 비용이 최적화되어야 합니다.

온디맨드 용량 모드를 선택하는 경우

비용을 최적화할 때 다음 그래프와 유사한 예측 불가능한 워크로드가 있는 경우 온디맨드 모드가 최선의 선택입니다.

이러한 유형의 워크로드에 영향을 미치는 요소는 다음과 같습니다.

- 예측할 수 없는 요청 타이밍(트래픽 급증 초래)
- 다양한 볼륨의 요청(배치 워크로드로 인한 요청)
- 지정된 1시간 동안 피크의 0 또는 18% 미만으로 떨어짐(개발 또는 테스트 환경의 결과)



위와 같은 특성을 가진 워크로드의 경우 Application Auto Scaling을 사용하여 트래픽 급증에 대응할 수 있는 테이블 용량을 충분히 유지하면 바람직하지 않은 결과가 발생할 수 있습니다. 테이블이 과도하게 프로비저닝되어 필요 이상으로 비용이 많이 들거나, 테이블이 제대로 프로비저닝되지 않아 요청으로 인해 불필요하게 저용량 처리량 오류가 발생할 수 있습니다. 이런 경우에는 온디맨드 테이블이 더 나은 선택입니다.

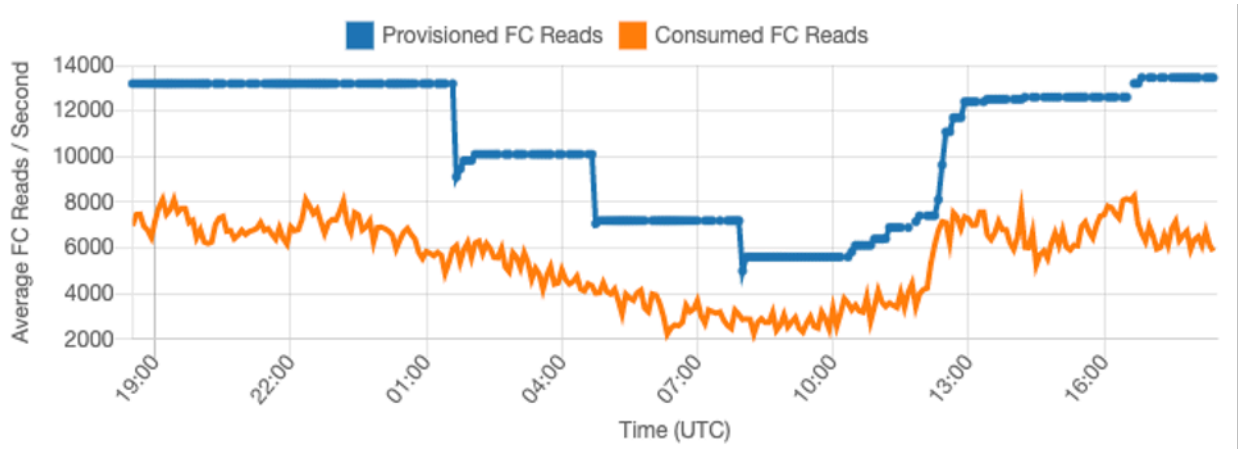
온디맨드 테이블은 요청에 따라 청구되기 때문에 비용 최적화를 위해 테이블 수준에서 더 이상 수행해야 할 작업이 없습니다. 워크로드에 여전히 위의 특성이 있는지 확인하기 위해 온디맨드 테이블을 정기적으로 평가해야 합니다. 워크로드가 안정화되면 프로비저닝 모드로 변경하여 비용 최적화를 유지하는 것이 좋습니다.

프로비저닝된 용량 모드를 선택하는 경우

프로비저닝된 용량 모드에 대한 이상적인 워크로드는 아래 그래프와 같이 사용 패턴이 더 예측 가능한 워크로드입니다.

예측 가능한 워크로드에 영향을 미치는 요소는 다음과 같습니다.

- 특정 시간 또는 일별 예측 가능한/주기적 트래픽
- 제한적인 단기 트래픽 폭주



지정된 시간 또는 일의 트래픽 볼륨이 더 안정적이기 때문에 프로비저닝된 용량을 테이블의 실제 사용된 용량에 비교적 가깝게 설정할 수 있습니다. 프로비저닝된 용량 테이블의 비용 최적화는 궁극적으로 테이블에서 `ThrottledRequests` 이벤트를 늘리지 않고 할당된 용량(파란색 선)을 사용된 용량(주황색 선)에 최대한 가깝게 만드는 연습입니다. 두 선 사이의 공간은 낭비된 용량뿐만 아니라 처리량 용량 부족 오류로 인해 발생할 수 있는 나쁜 사용자 경험에 대비한 용량을 나타냅니다.

Amazon Keyspaces는 프로비저닝된 용량 테이블에 대한 Application Auto Scaling을 제공하므로 사용자를 대신하여 자동으로 균형을 조정할 수 있습니다. 하루 종일 사용된 용량을 추적하고 몇 가지 변수를 기반으로 테이블의 프로비저닝된 용량을 구성할 수 있습니다.

최소 용량 단위

테이블의 최소 용량을 설정하여 처리량 용량 부족 오류의 발생을 줄일 수 있지만 테이블 비용이 줄어들지는 않습니다. 테이블 사용량이 낮은 기간이 있다가 갑자기 사용량이 급증한 경우 최소값으로 설정하면 Application Auto Scaling이 테이블 용량을 너무 낮게 설정하는 것을 방지할 수 있습니다.

최대 용량 단위

테이블의 최대 용량을 설정하여 테이블 확장을 의도한 것보다 높게 제한할 수 있습니다. 대규모 로드 테스트가 필요하지 않은 개발 또는 테스트 테이블에는 최대값을 적용하는 것이 좋습니다. 모든 테이블에 대해 최대값을 설정할 수 있지만 실수로 처리량 용량 부족 오류가 발생하지 않도록 프로덕션에서 사용할 때 테이블 기준선에 대해 이 설정을 정기적으로 평가해야 합니다.

목표 사용률

테이블의 목표 사용률을 설정하는 것은 프로비저닝된 용량 테이블에 대한 비용 최적화의 기본 수단입니다. 여기서 백분율 값을 낮게 설정하면 테이블이 오버프로비저닝되는 양이 늘어나 비용이 증가하지만 처리량 용량 부족 오류의 위험은 줄어듭니다. 여기서 백분율 값을 높게 설정하면 테이블이 오버프로비저닝되는 양이 줄어들지만 처리량 용량 부족 오류의 위험이 커집니다.

테이블 용량 모드를 선택할 때 고려해야 할 추가 요소

두 용량 모드 중 하나를 결정할 때 고려해야 할 몇 가지 추가 요소가 있습니다.

두 테이블 모드 중 하나를 결정할 때는 이러한 추가 할인이 테이블 비용에 얼마나 영향을 미치는지 고려합니다. 대부분의 경우 비교적 예측하기 어려운 워크로드라도 예약 용량이 있는 오버프로비저닝된 용량 테이블에서 실행하는 것이 더 저렴할 수 있습니다.

워크로드의 예측 가능성 향상

경우에 따라 워크로드에 예측 가능한 패턴과 예측할 수 없는 패턴이 모두 있는 것처럼 보일 수 있습니다. 온디맨드 테이블에서는 이러한 워크로드도 쉽게 지원할 수 있지만, 워크로드에 있는 예측할 수 없는 패턴을 개선할 수 있다면 비용을 더 줄일 수 있을 것입니다.

이러한 패턴의 가장 일반적인 원인 중 하나는 일괄 가져오기입니다. 이러한 유형의 트래픽은 테이블을 실행할 때 처리량 용량 부족 오류가 발생할 정도로 테이블의 기준 용량을 초과하는 경우가 많습니다. 프로비저닝된 용량 테이블에서 이와 같은 워크로드를 계속 실행하려면 다음 옵션을 사용해봅니다.

- 예약된 시간에 일괄 처리가 발생하는 경우 실행 전에 Application Auto Scaling 용량을 늘리도록 예약할 수 있습니다.
- 일괄 처리가 무작위로 발생하는 경우 최대한 빨리 실행하는 대신 실행 시간을 연장하는 것이 좋습니다.
- 가져오기 속도가 처음에는 느린 속도에서 시작해서 Application Auto Scaling이 테이블 용량 조정을 시작할 수 있을 때까지 몇 분 동안 서서히 오르도록 증가 기간을 가져오기에 추가합니다.

테이블의 Application Auto Scaling 설정 평가

이 섹션에서는 Amazon Keyspaces 테이블의 Application Auto Scaling 설정을 평가하는 방법을 간략히 살펴봅니다. Amazon Keyspaces Auto Scaling은 애플리케이션 트래픽과 대상 사용률 지표를 기반으로 테이블 처리량을 관리하는 기능입니다. 이렇게 하면 테이블이 애플리케이션 패턴에 필요한 용량을 확보할 수 있습니다.

Application Auto Scaling 서비스는 현재 테이블 사용률을 모니터링하고 이를 목표 사용률 값(`TargetValue`)과 비교합니다. 할당된 용량을 늘리거나 줄여야 할 시기가 되면 알려 줍니다.

주제

Application Auto Scaling 설정 이해하기

목표 사용률, 초기 단계 및 최종 값에 대한 올바른 값을 정의하려면 운영 팀의 참여가 필요합니다. 이렇게 하면 Application Auto Scaling 정책을 트리거하는 데 사용될 값을 과거 애플리케이션 사용량을 기반으로 적절하게 정의할 수 있습니다. 사용률 목표는 Application Auto Scaling 규칙이 적용되기 전에 일정 기간 동안 도달해야 하는 총 용량의 백분율입니다.

높은 사용률 목표(약 90%의 목표)를 설정하면 Application Auto Scaling이 활성화되기 전에 일정 기간 동안 트래픽이 90%보다 높아야 합니다. 애플리케이션이 매우 일정하고 트래픽 급증이 발생하지 않는 한 높은 사용률 목표를 사용해서는 안 됩니다.

매우 낮은 사용률(50% 미만의 목표)을 설정하면 애플리케이션이 Application Auto Scaling 정책을 실행하기 전에 프로비저닝된 용량의 50%에 도달해야 합니다. 애플리케이션 트래픽이 매우 빠른 속도로 증가하지 않는 한 이는 대개 용량 미사용 및 리소스 낭비로 이어집니다.

목표 사용률이 낮은(<= 50%) 테이블을 식별하는 방법

AWS CLI 또는 AWS Management Console 중 하나를 사용하여 Amazon Keyspaces 리소스에서 Application Auto Scaling 정책을 모니터링하고 식별할 수 있습니다. `TargetValues`

AWS CLI

1. 다음 명령을 실행하여 전체 리소스 목록을 반환합니다.

```
aws application-autoscaling describe-scaling-policies --service-namespace
 cassandra
```

이 명령은 모든 Amazon Keyspaces 리소스에 발행된 Application Auto Scaling 정책의 전체 목록을 반환합니다. 특정 테이블에서만 리소스를 검색하려는 경우 `–resource-id parameter`를 추가하면 됩니다. 예:

```
aws application-autoscaling describe-scaling-policies --service-namespace
 cassandra --resource-id "keyspace/keyspace-name/table/table-name”
```

2. 다음 명령을 실행하여 특정 테이블에 대한 Auto Scaling 정책만 반환하세요.

```
aws application-autoscaling describe-scaling-policies --service-namespace
 cassandra --resource-id "keyspace/keyspace-name/table/table-name”
```

Application Auto Scaling 정책에 대한 값은 아래에 강조 표시되어 있습니다. 과다 프로비저닝을 방지하기 위해 목표값을 50%보다 높게 설정해야 합니다. 다음과 유사한 결과가 출력되어야 합니다.

```
{
    "ScalingPolicies": [
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/keyspaces/table/table-name-scaling-policy",
            "PolicyName": $<full-gsi-name>”,
            "ServiceNamespace": "cassandra",
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:index:WriteCapacityUnits",
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "KeyspacesWriteCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:48.641000+10:00"
        },
        {
            "PolicyARN": "arn:aws:autoscaling:<region>:<account-
id>:scalingPolicy:<uuid>:resource/keyspaces/table/table-name/index/<index-
name>:policyName/$<full-gsi-name>-scaling-policy",
            "PolicyName":$<full-table-name>”,
            "ServiceNamespace": "cassandra",
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:index:ReadCapacityUnits",
```

```
            "PolicyType": "TargetTrackingScaling",
            "TargetTrackingScalingPolicyConfiguration": {
                "TargetValue": 70.0,
                "PredefinedMetricSpecification": {
                    "PredefinedMetricType": "CassandraReadCapacityUtilization"
                }
            },
            "Alarms": [
                ...
            ],
            "CreationTime": "2022-03-04T16:23:47.820000+10:00"
        }
    ]
}
```

AWS Management Console

1. AWS Management Console 로그인하고 시작하기에서 CloudWatch 서비스 페이지로 이동하십시오. AWS Management Console 필요한 AWS 리전 경우 적절한 항목을 선택합니다.
2. 왼쪽 탐색 메뉴에서 Tables(테이블)를 선택합니다. Tables(테이블) 페이지에서 테이블 이름을 선택합니다.
3. 테이블 세부 정보 페이지의 용량 탭에서 테이블의 Application Auto Scaling 설정을 검토합니다.

목표 사용률 값이 50% 이하이면 테이블 사용률 지표를 살펴보고 해당 지표가 과소 프로비저닝되었는지 과다 프로비저닝되었는지 확인해야 합니다.

계절적 변동이 있는 워크로드를 해결하는 방법

다음과 같은 시나리오를 생각해 보세요. 대부분의 경우 애플리케이션이 최소 평균값 미만으로 작동하지만 사용률 목표가 낮기 때문에 애플리케이션이 하루 중 특정 시간에 발생하는 이벤트에 신속하게 반응할 수 있고 충분한 용량이 확보되어 제한을 피할 수 있습니다. 이 시나리오는 일반적인 업무 시간(오전 9시~오후 5시)에는 사용량이 아주 많지만 업무 시간 이후에는 기본 수준에서 작동하는 애플리케이션이 있는 경우에 일반적입니다. 일부 사용자는 오전 9시 이전에 연결을 시작하므로 애플리케이션은 이 낮은 임계값을 사용하여 사용량이 많은 시간대에 필요한 용량에 빠르게 도달합니다.

이 시나리오는 다음과 같이 진행될 수 있습니다.

- 오후 5시에서 오전 9시 사이에는 ConsumedWriteCapacityUnits 단위가 90에서 100 사이로 유지됩니다.
- 사용자가 오전 9시 이전에 애플리케이션에 연결하기 시작하면 용량 단위가 크게 늘어납니다(지금까지 본 최대값은 1,500WCU).
- 평균적인 애플리케이션 사용량은 근무 시간 동안 800에서 1,200 사이로 다양합니다.

애플리케이션에 이전 시나리오가 적용되는 경우, 예약된 Application Auto Scaling을 사용하면 테이블에 구성된 Application Auto Scaling 규칙이 유지되면서도 필요한 특정 간격으로만 추가 용량을 프로비저닝하는 덜 적극적인 대상 사용률을 적용할 수 있습니다.

를 사용하여 다음 단계를 실행하여 시간과 요일을 기준으로 실행되는 스케줄링된 Auto Scaling 규칙을 생성할 수 있습니다. AWS CLI

1. Application Auto Scaling를 사용하여 Amazon Keyspaces 테이블을 확장 가능한 대상으로 등록합니다. 확장 가능한 목표란 Application Auto Scaling 이 스케일 아웃 또는 스케일 인할 수 있는 리소스입니다.

```
aws application-autoscaling register-scalable-target \
    --service-namespace cassandra \
    --scalable-dimension cassandra:table:WriteCapacityUnits \
    --resource-id keyspace/keyspace-name/table/table-name \
    --min-capacity 90 \
    --max-capacity 1500
```

2. 요구 사항에 따라 예약된 작업 설정

   시나리오를 다루려면 두 가지 규칙이 필요합니다. 하나는 스케일 업 규칙이고 다른 하나는 스케일 다운 규칙입니다. 예약된 작업을 스케일 업하기 위한 첫 번째 규칙은 다음 예제와 같습니다.

```
aws application-autoscaling put-scheduled-action \
    --service-namespace cassandra \
    --scalable-dimension cassandra:table:WriteCapacityUnits \
    --resource-id keyspace/keyspace-name/table/table-name \
    --scheduled-action-name my-8-5-scheduled-action \
    --scalable-target-action MinCapacity=800,MaxCapacity=1500 \
    --schedule "cron(45 8 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

   예약된 작업을 스케일 다운하기 위한 두 번째 규칙은 다음 예제와 같습니다.

```
aws application-autoscaling put-scheduled-action \
    --service-namespace cassandra \
    --scalable-dimension cassandra:table:WriteCapacityUnits \
    --resource-id keyspace/keyspace-name/table/table-name \
    --scheduled-action-name my-5-8-scheduled-down-action \
    --scalable-target-action MinCapacity=90,MaxCapacity=1500 \
    --schedule "cron(15 17 ? * MON-FRI *)" \
    --timezone "Australia/Brisbane"
```

3. 다음 명령을 실행하여 두 규칙이 모두 활성화되었는지 확인합니다.

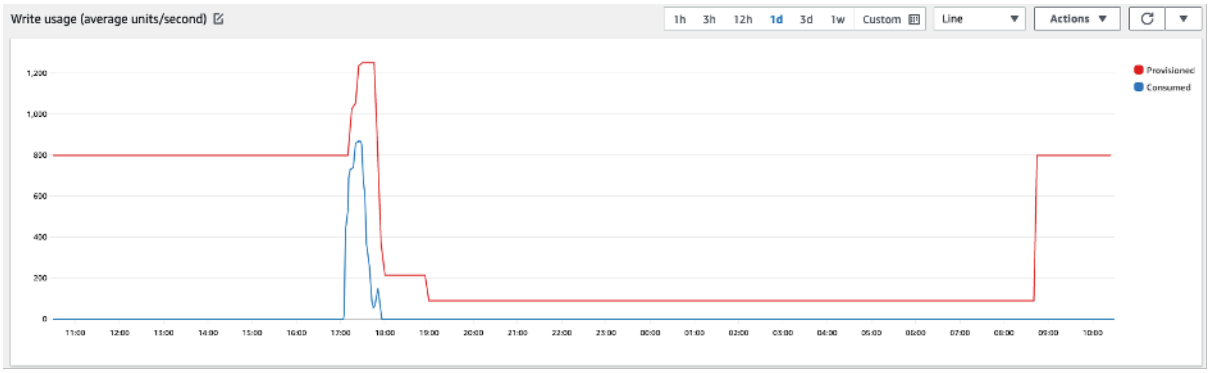
```
aws application-autoscaling describe-scheduled-actions --service-namespace
 cassandra
```

다음과 같은 결과가 나와야 합니다.

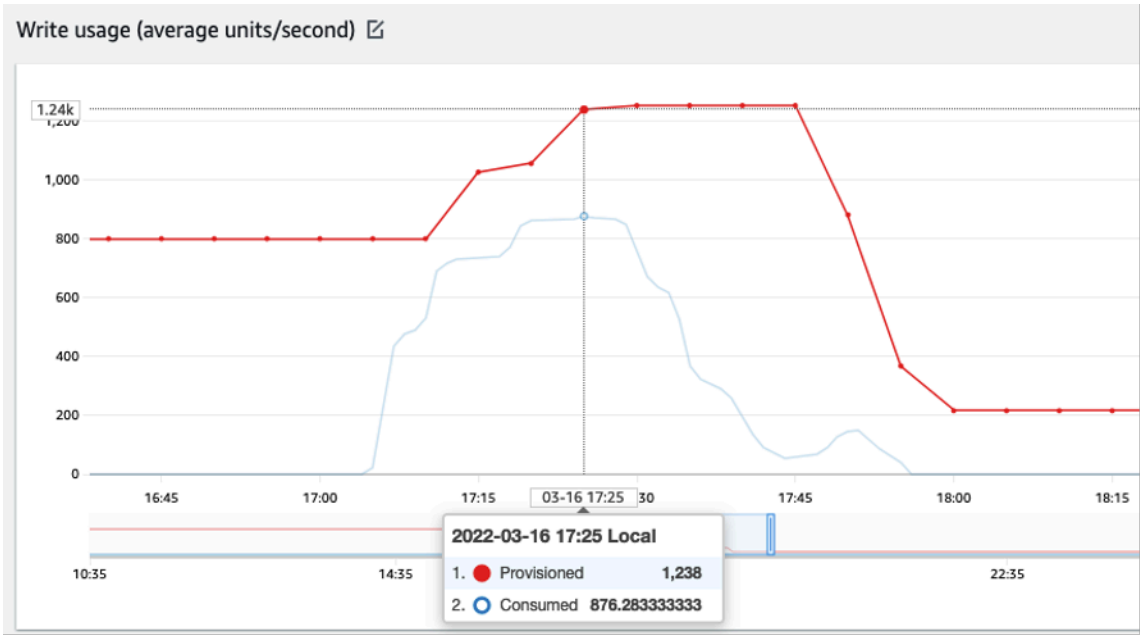
```
{
    "ScheduledActions": [
        {
            "ScheduledActionName": "my-5-8-scheduled-down-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/keyspaces/
table/table-name:scheduledActionName/my-5-8-scheduled-down-action",
            "ServiceNamespace": "cassandra",
            "Schedule": "cron(15 17 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 90,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:30:25.100000+10:00"
        },
        {
            "ScheduledActionName": "my-8-5-scheduled-action",
            "ScheduledActionARN":
 "arn:aws:autoscaling:<region>:<account>:scheduledAction:<uuid>:resource/keyspaces/
table/table-name:scheduledActionName/my-8-5-scheduled-action",
            "ServiceNamespace": "cassandra",
            "Schedule": "cron(45 8 ? * MON-FRI *)",
            "Timezone": "Australia/Brisbane",
```

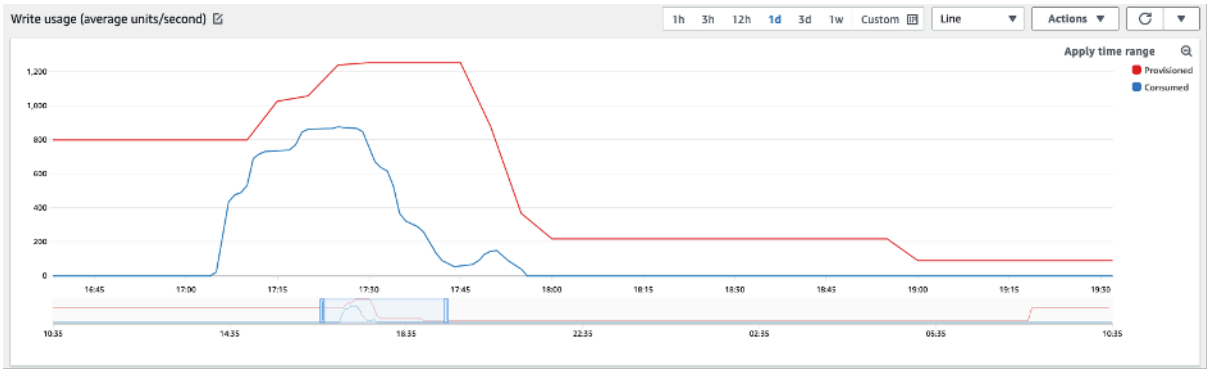
```
            "ResourceId": "keyspace/keyspace-name/table/table-name",
            "ScalableDimension": "cassandra:table:WriteCapacityUnits",
            "ScalableTargetAction": {
                "MinCapacity": 800,
                "MaxCapacity": 1500
            },
            "CreationTime": "2022-03-15T17:28:57.816000+10:00"
        }
    ]
}
```

다음 그림은 항상 70% 목표 사용률을 유지하는 샘플 워크로드를 보여 줍니다. Auto Scaling 규칙이 여전히 적용되고 있으며 처리량은 감소하지 않는다는 점에 유의하세요.



자세히 보면 애플리케이션에 급증이 발생하여 70% Auto Scaling 임계값이 트리거되고, Auto Scaling이 강제로 시작되어 테이블에 필요한 추가 용량을 제공하는 것을 볼 수 있습니다. 예약된 Auto Scaling 작업은 최대값 및 최소값에 영향을 미치며, 이를 설정하는 것은 사용자의 책임입니다.





알 수 없는 패턴으로 급증하는 워크로드를 해결하는 방법

이 시나리오에서는 애플리케이션 패턴을 아직 모르고 워크로드에 용량 처리량 부족 오류가 발생하지 않도록 하기 위해 애플리케이션이 매우 낮은 사용률 목표를 사용합니다.

대신 온디맨드 용량 모드를 사용하는 것을 고려해 봅니다. 온디맨드 테이블은 트래픽 패턴을 모르는 급증하는 워크로드에 적합합니다. 온디맨드 용량 모드를 사용하면 애플리케이션이 테이블에서 수행하는 데이터 읽기 및 쓰기에 대해 요청당 요금을 지불합니다. Amazon Keyspaces는 늘어나거나 줄어드는 워크로드를 즉시 수용하기 때문에 애플리케이션에서 수행할 것으로 예상되는 읽기 및 쓰기 처리량을 지정할 필요가 없습니다.

연결된 애플리케이션으로 워크로드를 해결하는 방법

이 시나리오에서 애플리케이션은 다른 시스템에 의존합니다. 예를 들어 애플리케이션 로직의 이벤트에 따라 트래픽이 크게 급증할 수 있는 일괄 처리 시나리오가 이에 해당합니다.

이러한 이벤트에 대응하여 특정 요구 사항에 따라 테이블 용량 및 `TargetValues`를 늘릴 수 있는 사용자 지정 Application Auto Scaling 로직을 개발하는 것을 고려해 보세요. Amazon EventBridge 및 Step Functions와 같은 AWS 서비스의 조합을 활용하고 사용하여 특정 애플리케이션 요구 사항에 대응할 수 있습니다.

미사용 리소스 식별

이 섹션에서는 미사용 리소스를 정기적으로 평가하는 방법을 간략히 살펴봅니다. 애플리케이션 요구 사항이 발전함에 따라 미사용 리소스가 없고 이로 인해 불필요한 Amazon Keyspaces 비용이 발생하지 않도록 해야 합니다. 아래에 설명된 절차는 Amazon CloudWatch 지표를 사용하여 미사용 리소스를 식별하고 비용을 줄이기 위한 조치를 취합니다.

Amazon Keyspace의 원시 데이터를 수집하여 읽기 가능한 거의 실시간 지표로 처리하는 를 사용하여 CloudWatch Amazon Keyspaces를 모니터링할 수 있습니다. 이러한 통계는 일정 기간 동안 유지되므로 기록 정보에 액세스하여 사용률을 더 잘 파악할 수 있습니다. 기본적으로 Amazon Keyspaces 메트릭 데이터는 자동으로 전송됩니다 CloudWatch . 자세한 내용은 Amazon이란 무엇입니까 CloudWatch? 를 참조하십시오. 및 Amazon CloudWatch 사용 설명서의 지표 보존.

주제

- 미사용 리소스를 식별하는 방법
- 미사용 테이블 리소스 식별
- 미사용 테이블 리소스 정리
- 미사용 point-in-time 복구 (PITR) 백업 정리

미사용 리소스를 식별하는 방법

사용하지 않는 테이블을 식별하려면 30일 동안 다음 CloudWatch 지표를 살펴보고 특정 테이블에 활성 읽기 또는 쓰기가 있는지 파악할 수 있습니다.

`ConsumedReadCapacityUnits`

일정 기간 동안 사용된 읽기 용량 단위의 수로, 이를 통해 사용된 용량이 얼마나 사용되었는지 추적할 수 있습니다. 테이블에 대해 소비된 총 읽기 용량을 검색할 수 있습니다.

`ConsumedWriteCapacityUnits`

일정 기간 동안 사용된 쓰기 용량 단위의 수로, 이를 통해 사용된 용량을 얼마나 사용했는지 추적할 수 있습니다. 테이블에 대해 소비된 총 쓰기 용량을 검색할 수 있습니다.

미사용 테이블 리소스 식별

CloudWatch Amazon은 미사용 리소스를 식별하는 데 사용할 수 있는 Amazon Keyspaces 테이블 지표를 제공하는 모니터링 및 관찰 서비스입니다. CloudWatch 지표는 를 AWS Management Console 통해서뿐만 아니라 를 통해서도 볼 수 있습니다. AWS Command Line Interface

AWS Command Line Interface

를 통해 테이블 지표를 보려면 다음 명령을 사용할 수 있습니다. AWS Command Line Interface

1. 먼저 테이블의 읽기를 평가합니다.

> **Note**
>
> 테이블 이름이 계정 내에서 고유하지 않은 경우 키스페이스의 이름도 지정해야 합니다.

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedReadCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/Cassandra --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

테이블을 미사용 테이블로 잘못 식별하지 않으려면 더 오랜 기간 동안 지표를 평가하세요. 적절한 시작 시간 및 종료 시간 범위(예: 30일)와 적절한 기간(예: 86400)을 선택합니다.

반환된 데이터에서 0보다 큰 모든 합계는 해당 기간 동안 수신된 읽기 트래픽을 평가 중인 테이블을 나타냅니다.

다음 결과에는 평가 기간에 읽기 트래픽을 수신한 테이블이 표시됩니다.

```
        {
            "Timestamp": "2022-08-25T19:40:00Z",
            "Sum": 36023355.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-12T19:40:00Z",
            "Sum": 38025777.5,
            "Unit": "Count"
```

```
        },
```

다음 결과에는 평가 기간에 읽기 트래픽을 수신하지 않은 테이블이 표시됩니다.

```
        {
            "Timestamp": "2022-08-01T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-20T19:50:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

2. 다음으로 테이블의 쓰기를 평가하세요.

```
aws cloudwatch get-metric-statistics --metric-name
ConsumedWriteCapacityUnits --start-time <start-time> --end-time <end-
time> --period <period> --namespace AWS/Cassandra --statistics Sum --
dimensions Name=TableName,Value=<table-name>
```

테이블을 미사용 테이블로 잘못 식별하지 않으려면 더 오랜 기간 동안 지표를 평가하는 것이 좋습니다. 적절한 시작 시간 및 종료 시간 범위(예: 30 days(30일))와 적절한 기간(예: 86400)을 선택합니다.

반환된 데이터에서 0보다 큰 모든 합계는 해당 기간 동안 수신된 읽기 트래픽을 평가 중인 테이블을 나타냅니다.

다음 결과에는 평가 기간에 쓰기 트래픽을 수신한 테이블이 표시됩니다.

```
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 41014457.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-18T20:15:00Z",
            "Sum": 40048531.0,
            "Unit": "Count"
        },
```

다음 결과에는 평가 기간에 쓰기 트래픽을 수신하지 않은 테이블이 표시됩니다.

```
        {
            "Timestamp": "2022-07-31T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
        {
            "Timestamp": "2022-08-19T20:15:00Z",
            "Sum": 0.0,
            "Unit": "Count"
        },
```

AWS Management Console

다음 단계에서는 AWS Management Console을 통해 리소스 사용률을 평가할 수 있습니다.

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/cloudwatch/ CloudWatch 서비스 페이지로 이동합니다. 필요한 경우 콘솔 오른쪽 상단에서 해당 AWS 리전 항목을 선택합니다.
2. 왼쪽 탐색 메뉴에서 지표를 선택한 다음 모든 지표를 선택합니다.
3. 위 작업을 수행하면 두 개의 패널이 있는 대시보드가 열립니다. 상단 패널에서는 현재 그래프로 표시된 지표를 확인할 수 있습니다. 하단에서는 그래프로 나타낼 수 있는 지표를 선택할 수 있습니다. 하단 패널에서 Amazon Keyspaces를 선택합니다.
4. Amazon Keyspaces 지표 선택 패널에서 테이블 지표 범주를 선택하면 현재 리전의 테이블에 해당하는 지표가 표시됩니다.
5. 메뉴를 아래로 스크롤하여 테이블 이름을 식별한 후 테이블에 해당하는 `ConsumedReadCapacityUnits` 및 `ConsumedWriteCapacityUnits` 지표를 선택합니다.
6. 그래프로 표시된 지표(2) 탭을 선택하고 통계 열을 합계로 조정합니다.
7. 테이블을 미사용 테이블로 잘못 식별하지 않으려면 더 오랜 기간 동안 테이블 지표를 평가하세요. 그래프 패널 상단에서 테이블을 평가할 적절한 기간(예: 1개월)을 선택합니다. 사용자 지정을 선택하고 드롭다운 메뉴에서 1개월을 선택한 다음 적용을 선택합니다.
8. 테이블의 그래프로 표시된 지표를 평가하여 사용 중인지 확인하세요. 지표가 0을 초과하면 평가된 기간 동안 테이블이 사용되었음을 나타냅니다. 읽기와 쓰기 모두에 대해 0에 있는 평면 그래프는 미사용 테이블을 나타냅니다.

미사용 테이블 리소스 정리

미사용 테이블 리소스를 식별한 경우 다음과 같은 방법으로 지속적인 비용을 줄일 수 있습니다.

> **Note**
>
> 미사용 테이블을 식별했지만 나중에 액세스해야 할 경우를 대비하여 계속 사용할 수 있으려면 온디맨드 모드로 전환하는 것이 좋습니다. 그렇지 않으면 테이블을 삭제하는 방안을 고려할 수 있습니다.

용량 모드

Amazon Keyspaces는 Amazon Keyspaces 테이블의 데이터 읽기, 쓰기, 저장에 대한 요금을 청구합니다.

Amazon Keyspaces에는 테이블에서 읽기 및 쓰기 처리를 위한 특정 결제를 위한 두 가지 용량 모드가 있습니다. 읽기/쓰기 용량 모드는 읽기 및 쓰기 처리량에 대한 청구 방법과 용량 관리 방법을 제어합니다.

온디맨드 모드 테이블의 경우 애플리케이션에서 수행할 것으로 예상되는 읽기 및 쓰기 처리량을 지정할 필요가 없습니다. Amazon Keyspaces에서는 읽기 요청 단위 및 쓰기 요청 단위의 측면에서 애플리케이션이 테이블에서 수행하는 읽기 및 쓰기에 대해 요금이 부과됩니다. 테이블에 활동이 없는 경우 처리량(throughput)에 대한 비용은 지불하지 않지만 스토리지 요금은 계속 부과됩니다.

테이블 삭제

미사용 테이블을 발견하여 삭제하려는 경우 먼저 데이터를 백업하거나 내보내는 것이 좋습니다.

백업을 통해 콜드 스토리지 계층화를 활용하여 비용을 더욱 AWS Backup 절감할 수 있습니다. 수명 주기를 사용하여 백업을 콜드 스토리지로 이동하는 방법에 대한 정보는 백업 관리 계획 설명서를 참조하세요.

테이블을 백업한 후 AWS Management Console 또는 AWS Command Line Interface를 통해 테이블을 삭제할 수 있습니다.

미사용 point-in-time 복구 (PITR) 백업 정리

Amazon Keyspace는 35일 동안 연속 백업을 제공하는 IP oint-in-time 복구를 제공하므로 실수로 인한 쓰기 또는 삭제를 방지할 수 있습니다. PITR 백업에는 비용이 발생합니다.

테이블에 더 이상 필요하지 않을 수 있는 백업이 활성화되어 있는지 확인하려면 시점 복구의 설명서를 참조하세요.

테이블 사용 패턴 평가

이 섹션에서는 Amazon Keyspaces 테이블을 효율적으로 사용하고 있는지 평가하는 방법을 간략히 살펴봅니다. Amazon Keyspaces에 최적화되지 않은 특정 사용 패턴이 있으며, 이러한 패턴은 성능 및 비용 측면에서 모두 최적화될 수 있는 여지가 있습니다.

주제

- 강력히 일관된 읽기 작업 줄이기
- Time to Live(TTL) 활성화

강력히 일관된 읽기 작업 줄이기

Amazon Keyspaces를 사용하면 요청별로 읽기 일관성을 구성할 수 있습니다. 기본적으로 읽기 요청은 최종적으로 일관됩니다. 최종 읽기 일관성은 최대 4KB의 데이터에 대해 0.5RCU의 요금이 부과됩니다.

분산 워크로드의 대부분은 유연하며 최종 일관성을 허용합니다. 하지만 강력히 일관된 읽기가 필요한 액세스 패턴이 있을 수 있습니다. 강력히 일관된 읽기는 최대 4KB의 데이터에 대해 1RCU의 요금이 부과되므로 읽기 비용이 두 배로 늘어납니다. Amazon Keyspaces는 동일한 테이블에서 두 정합성 모델을 모두 사용할 수 있는 유연성을 제공합니다.

워크로드와 애플리케이션 코드를 평가하여 강력히 일관된 읽기가 필요한 경우에만 사용되는지 확인할 수 있습니다.

Time to Live(TTL) 활성화

Time to Live(TTL)를 통해 테이블의 데이터를 자동으로 만료시켜 애플리케이션 로직을 간소화하고 스토리지 가격을 최적화할 수 있습니다. 더 이상 필요하지 않은 데이터는 설정한 TTL 값에 따라 테이블에서 자동으로 삭제됩니다.

적절한 규모의 프로비저닝을 위해 프로비저닝된 용량 평가

이 섹션에서는 Amazon Keyspaces 테이블에 적절한 규모의 프로비저닝이 있는지 평가하는 방법을 간략히 살펴봅니다. 워크로드가 발전함에 따라 운영 절차를 적절하게 수정해야 합니다. 특히 Amazon Keyspaces 테이블이 프로비저닝 모드로 구성되어 있고 테이블을 과다 프로비저닝하거나 과소 프로비저닝할 위험이 있는 경우에는 더욱 그렇습니다.

이 섹션에 설명된 절차에는 프로덕션 애플리케이션을 지원하는 Amazon Keyspaces 테이블에서 캡처해야 하는 통계 정보가 필요합니다. 애플리케이션 동작을 이해하려면 애플리케이션의 데이터 계절성을 포착할 수 있을 만큼 충분히 중요한 기간을 정의해야 합니다. 예를 들어 애플리케이션이 주간 패턴을 보이는 경우 3주 기간을 사용하면 애플리케이션 처리량 요구 사항을 분석할 시간을 충분히 확보할 수 있습니다.

어디서부터 시작해야 할지 모르겠다면 아래 계산에 최소 한 달 분량의 데이터 사용량을 사용해 보세요.

용량을 평가하는 동안 Amazon Keyspaces 테이블에 읽기 용량 단위(RCU)와 쓰기 용량 단위(WCU)를 별개로 구성할 수 있습니다.

주제

Amazon Keyspaces 테이블에서 소비 지표를 검색하는 방법

테이블 용량을 평가하려면 다음 CloudWatch 지표를 모니터링하고 적절한 차원을 선택하여 테이블 정보를 검색하십시오.

읽기 용량 단위	쓰기 용량 단위
`ConsumedReadCapacityUnits`	`ConsumedWriteCapacityUnits`
`ProvisionedReadCapacityUnits`	`ProvisionedWriteCapacityUnits`
`ReadThrottleEvents`	`WriteThrottleEvents`

또는 를 통해 이 작업을 수행할 수 AWS Management Console있습니다. AWS CLI

AWS CLI

테이블 사용량 지표를 검색하려면 먼저 CloudWatch API를 사용하여 일부 기록 데이터 포인트를 캡처해야 합니다.

먼저 두 개의 파일(`write-calc.json` 및 `read-calc.json`)을 만듭니다. 이러한 파일은 테이블에 대한 계산을 나타냅니다. 아래 표에 표시된 대로 일부 필드를 환경에 맞게 업데이트해야 합니다.

> **Note**
>
> 테이블 이름이 계정 내에서 고유하지 않은 경우 키스페이스의 이름도 지정해야 합니다.

필드 이름	정의	예
`<table-name>`	분석하려는 테이블 이름	SampleTable
`<period>`	사용률 목표를 평가하는 데 사용하는 기간(초 단위)	1시간인 경우 3,600 지정
`<start-time>`	평가 간격의 시작 부분으로, ISO8601 형식으로 지정	2022-02-21T23:00:00
`<end-time>`	평가 간격의 끝부분으로, ISO8601 형식으로 지정	2022-02-22T06:00:00

쓰기 계산 파일은 지정된 날짜 범위에 해당하는 기간 동안 프로비저닝되고 소비된 WCU 수를 검색합니다. 또한 분석에 사용할 수 있는 사용률도 생성합니다. `write-calc.json` 파일의 전체 내용은 다음 예제와 같은 형식으로 보입니다.

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Cassandra",
          "MetricName": "ProvisionedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
```

```
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedWCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Cassandra",
          "MetricName": "ConsumedWriteCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>""
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
    },
    {
      "Id": "m1",
      "Expression": "consumedWCU/PERIOD(consumedWCU)",
      "Label": "Consumed WCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedWCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

읽기 계산 파일은 비슷한 지표를 사용합니다. 이 파일은 지정된 날짜 범위에 해당하는 기간 동안 프로비저닝되고 소비된 RCU 수를 검색합니다. `read-calc.json` 파일의 콘텐츠는 다음 예제와 같아야 합니다.

```
{
  "MetricDataQueries": [
    {
      "Id": "provisionedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Cassandra",
          "MetricName": "ProvisionedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Average"
      },
      "Label": "Provisioned",
      "ReturnData": false
    },
    {
      "Id": "consumedRCU",
      "MetricStat": {
        "Metric": {
          "Namespace": "AWS/Cassandra",
          "MetricName": "ConsumedReadCapacityUnits",
          "Dimensions": [
            {
              "Name": "TableName",
              "Value": "<table-name>"
            }
          ]
        },
        "Period": <period>,
        "Stat": "Sum"
      },
      "Label": "",
      "ReturnData": false
```

```
    },
    {
      "Id": "m1",
      "Expression": "consumedRCU/PERIOD(consumedRCU)",
      "Label": "Consumed RCUs",
      "ReturnData": false
    },
    {
      "Id": "utilizationPercentage",
      "Expression": "100*(m1/provisionedRCU)",
      "Label": "Utilization Percentage",
      "ReturnData": true
    }
  ],
  "StartTime": "<start-time>",
  "EndTime": "<end-time>",
  "ScanBy": "TimestampDescending",
  "MaxDatapoints": 24
}
```

파일을 만들었으면 사용률 데이터 검색을 시작할 수 있습니다.

1. 쓰기 사용률 데이터를 검색하려면 다음 명령을 실행합니다.

```
aws cloudwatch get-metric-data --cli-input-json file://write-calc.json
```

2. 읽기 사용률 데이터를 검색하려면 다음 명령을 실행합니다.

```
aws cloudwatch get-metric-data --cli-input-json file://read-calc.json
```

두 쿼리의 결과는 분석에 사용할 수 있는 일련의 JSON 형식 데이터 포인트입니다. 결과는 지정한 데이터 포인트 수, 기간 및 고유한 워크로드 데이터에 따라 달라집니다. 다음 예제와 같을 수 있습니다.

```
{
    "MetricDataResults": [
        {
            "Id": "utilizationPercentage",
            "Label": "Utilization Percentage",
            "Timestamps": [
                "2022-02-22T05:00:00+00:00",
```

```
                "2022-02-22T04:00:00+00:00",
                "2022-02-22T03:00:00+00:00",
                "2022-02-22T02:00:00+00:00",
                "2022-02-22T01:00:00+00:00",
                "2022-02-22T00:00:00+00:00",
                "2022-02-21T23:00:00+00:00"
            ],
            "Values": [
                91.55364583333333,
                55.066631944444445,
                2.6114930555555556,
                24.9496875,
                40.94725694444445,
                25.61819444444444,
                0.0
            ],
            "StatusCode": "Complete"
        }
    ],
    "Messages": []
}
```

> **Note**
>
> 짧은 기간과 긴 시간 범위를 지정하는 경우 스크립트에서 기본값인 24로 설정되어 있는 `MaxDatapoints` 값을 수정해야 할 수 있습니다. 이는 시간당 하나의 데이터 포인트, 하루에 24개의 데이터 포인트를 나타냅니다.

AWS Management Console

1. 에 AWS Management Console 로그인하고 시작하기에서 CloudWatch 서비스 페이지로 이동하십시오. AWS Management Console 필요한 AWS 리전 경우 적절한 항목을 선택합니다.
2. 왼쪽 탐색 메뉴에서 지표 섹션을 찾은 다음 모든 지표를 선택합니다.
3. 그러면 두 개의 패널이 있는 대시보드가 열립니다. 상단 패널에는 그래프가 표시되고 하단 패널에는 그래프로 표시하려는 지표가 표시됩니다. Amazon Keyspaces 패널을 선택합니다.
4. 하위 패널에서 테이블 지표 범주를 선택합니다. 그러면 현재 테이블이 표시됩니다 AWS 리전.
5. 메뉴를 아래로 스크롤하고 쓰기 작업 지표(`ConsumedWriteCapacityUnits` 및 `ProvisionedWriteCapacityUnits`)를 선택하여 테이블 이름을 식별합니다.

> **Note**
>
> 이 예제에서는 쓰기 작업 지표에 대해 설명하지만 다음 단계를 사용하여 읽기 작업 지표를 그래프로 표시할 수도 있습니다.

6. Graphed metrics (2)(그래프로 표시된 지표(2)) 탭을 선택하여 수식을 수정합니다. 기본적으로 그래프의 통계 함수 Average를 CloudWatch 선택합니다.
7. 그래프로 표시된 지표 두 개를 모두 선택한 상태에서(왼쪽의 확인란) Add math(수식 추가), Common(공통) 메뉴를 차례로 선택한 다음 Percentage 함수를 선택합니다. 절차를 두 번 반복합니다.

   처음으로 Percentage 함수를 선택할 때.

   두 번째로 Percentage 함수를 선택할 때.
8. 이제 하단 메뉴에 네 개의 지표가 있어야 합니다. `ConsumedWriteCapacityUnits` 계산 작업을 해봅시다. 일관성을 유지하려면 AWS CLI 섹션에서 사용한 이름과 이름을 일치시켜야 합니다. m1 ID를 클릭하고 이 값을 ConsumedWCU로 변경합니다.
9. 통계를 Average에서 Sum으로 변경합니다. 이 작업을 수행하면 ANOMALY_DETECTION_BAND라는 또 다른 지표가 자동으로 생성됩니다. 이 절차의 범위에서는 새로 생성된 ad1 지표에서 확인란을 선택 취소하여 이 절차를 무시해도 됩니다.
10. 8단계를 반복하여 m2 ID의 이름을 provisionedWCU로 변경합니다. 통계는 Average로 설정된 상태로 둡니다.
11. Expression1 레이블을 선택하고 값을 m1로 업데이트한 다음 레이블을 Consumed WCUs로 업데이트합니다.

> **Note**
>
> 데이터를 제대로 시각화하려면 m1(왼쪽의 확인란) 및 provisionedWCU만 선택했는지 확인하세요. Details(세부 정보)를 클릭하고 수식을 consumedWCU/PERIOD(consumedWCU)로 변경하여 수식을 업데이트합니다. 이 단계는 또 다른 ANOMALY_DETECTION_BAND 지표를 생성할 수도 있지만 이 절차의 범위에서는 무시해도 됩니다.

12. 이제 두 개의 그래픽이 생겼을 것입니다. 하나는 테이블의 프로비저닝된 WCU를 나타내고 다른 하나는 소비된 WCU를 나타냅니다.

13. Expression2 그래픽(e2)을 선택하여 백분율 수식을 업데이트합니다. 레이블 및 ID의 이름을 utilizationPercentage로 변경합니다. 수식의 이름을 100*(m1/provisionedWCU)와 일치하도록 변경합니다.
14. 사용률 패턴을 시각화하려면 utilizationPercentage를 제외한 모든 지표에서 확인란을 선택 취소하세요. 기본 간격은 1분으로 설정되어 있지만 필요에 따라 자유롭게 수정할 수 있습니다.

결과는 워크로드의 실제 데이터에 따라 달라집니다. 간격의 사용률이 100%를 초과하는 경우 처리량 용량 부족 오류 이벤트가 발생하기 쉽습니다. Amazon Keyspaces는 버스트 용량을 제공하지만 버스트 용량이 소진되는 즉시 100%를 초과하는 용량에 처리량 용량 부족 오류 이벤트가 발생합니다.

과소 프로비저닝된 Amazon Keyspaces 테이블을 식별하는 방법

대부분의 워크로드에서 테이블은 프로비저닝된 용량의 80%를 초과하여 지속적으로 소비하는 경우 과소 프로비저닝된 것으로 간주됩니다.

버스트 용량은 고객이 원래 프로비저닝된 것보다 더 많은(테이블에 정의된 초당 프로비저닝된 처리량보다 많은) RCU/WCU를 일시적으로 소비할 수 있도록 하는 Amazon Keyspaces 기능입니다. 버스트 용량은 특수 이벤트 또는 사용량 급증으로 인한 갑작스러운 트래픽 증가를 흡수하기 위해 만들어졌습니다. 이 버스트 용량 제한에 대한 자세한 내용은 the section called “버스트 용량”을(를) 참조하세요. 사용되지 않은 RCU와 WCU가 소진되었을 때 프로비저닝된 용량보다 더 많은 용량을 소비하려고 하면 용량 처리량 부족 오류 이벤트가 발생합니다. 애플리케이션 트래픽이 80% 사용률에 가까워지면 용량 처리량 부족 오류 이벤트의 위험이 훨씬 높아집니다.

80% 사용률 규칙은 데이터의 계절성 및 트래픽 증가에 따라 달라집니다. 다음 시나리오를 고려해 보세요.

- 지난 12개월 동안 트래픽이 약 90%의 사용률로 안정적이었다면 테이블의 용량이 적절한 것입니다.
- 애플리케이션 트래픽이 3개월 이내에 매월 8%씩 증가하면 100%에 도달하게 됩니다.
- 애플리케이션 트래픽이 4개월이 조금 넘는 기간 이내에 5%씩 증가해도 100%에 도달하게 됩니다.

위 쿼리의 결과는 사용률을 보여줍니다. 이를 참고하여 필요에 따라 테이블 용량을 늘리도록 선택하는 데 도움이 될 수 있는 다른 지표(예: 월별 또는 주별 증가율)를 추가로 평가해 보세요. 운영 팀과 협력하여 워크로드와 테이블에 적합한 비율을 정하세요.

일별 또는 주별로 데이터를 분석할 때 데이터가 왜곡되는 특수한 시나리오가 있습니다. 예를 들어 근무 시간 동안 사용량이 급증하다가 근무 시간 외에는 거의 0으로 떨어지는 계절성 애플리케이션의 경우,

프로비저닝된 용량을 늘리거나 줄일 하루 중 시간(및 요일)을 지정하는 Application Auto Scaling을 예약하면 효과를 볼 수 있습니다. 바쁜 시간을 처리하기 위해 더 높은 용량을 목표로 하는 대신 계절성이 덜 두드러지는 경우 Amazon Keyspaces 테이블 Auto Scaling 구성을 활용할 수도 있습니다.

과다 프로비저닝된 Amazon Keyspaces 테이블을 식별하는 방법

위 스크립트에서 얻은 쿼리 결과는 일부 초기 분석을 수행하는 데 필요한 데이터 포인트를 제공합니다. 데이터 세트의 여러 간격에 사용률이 20% 미만인 값이 표시되면 테이블이 과다 프로비저닝된 것일 수 있습니다. WCU 및 RCU 수를 줄여야 하는지 여부를 더 자세히 정의하려면 해당 간격의 다른 측정값을 다시 살펴봐야 합니다.

테이블에 낮은 사용 간격이 여러 개 있는 경우 Application Auto Scaling을 예약하거나 사용률을 기반으로 테이블에 대한 기본 Application Auto Scaling 정책을 구성하여 Application Auto Scaling 정책을 사용하면 이점을 얻을 수 있습니다.

사용률 대 높은 스로틀 비율 (간격의 Max (ThrottleEvents) /Min ())ThrottleEvents) 이 낮은 워크로드가 있는 경우, 워크로드가 매우 심해 특정 날짜 (또는 시간대) 에 트래픽이 크게 증가하지만 그 외에는 지속적으로 낮을 때 이런 현상이 발생할 수 있습니다. 이러한 시나리오에서는 예약된 Application Auto Scaling을 사용하는 것이 유용할 수 있습니다.

Amazon Keyspaces(Apache Cassandra용)와 함께 NoSQL Workbench 사용

NoSQL Workbench는 Amazon Keyspaces의 비관계형 데이터 모델을 보다 쉽게 설계하고 시각화할 수 있는 클라이언트 측 애플리케이션입니다. NoSQL Workbench 클라이언트는 Windows, macOS, Linux에서 사용할 수 있습니다.

데이터 모델 설계 및 리소스 자동 생성

NoSQL Workbench는 Amazon Keyspaces 데이터 모델을 설계하고 생성할 수 있는 포인트 앤 클릭 인터페이스를 제공합니다. 키스페이스, 테이블 및 열을 정의하여 처음부터 새 데이터 모델을 쉽게 생성할 수 있습니다. 또한 기존 데이터 모델을 가져와서 수정(예: 열 추가, 편집 또는 제거)을 수행하여 새 애플리케이션에 맞게 데이터 모델을 조정할 수 있습니다. 그러면 NoSQL Workbench를 사용하여 데이터 모델을 Amazon Keyspaces 또는 Apache Cassandra에 커밋하고 키스페이스와 테이블을 자동으로 생성할 수 있습니다. 데이터 모델을 구축하는 방법을 알아보려면 the section called “데이터 모델러” 섹션을 참조하세요.

데이터 모델 시각화

NoSQL Workbench를 사용하면 데이터 모델을 시각화하여 데이터 모델이 애플리케이션의 쿼리 및 액세스 패턴을 지원할 수 있는지 확인할 수 있습니다. 또한 협업, 문서 및 프레젠테이션을 위해 데이터 모델을 다양한 형식으로 저장하고 내보낼 수 있습니다. 자세한 내용은 the section called “데이터 비주얼라이저” 섹션을 참조하세요.

주제

- NoSQL Workbench 다운로드
- NoSQL Workbench로 시작하기
- 데이터 모델 구축 방법
- 데이터 모델을 시각화하는 방법
- Amazon Keyspaces와 Apache Cassandra에 데이터 모델을 커밋하는 방법
- NoSQL Workbench의 샘플 데이터 모델
- NoSQL Workbench 릴리스 기록

NoSQL Workbench 다운로드

다음 지침에 따라 NoSQL Workbench를 다운로드하여 설치합니다.

NoSQL Workbench를 다운로드 및 설치하려면

1. 다음 링크 중 하나를 사용하여 NoSQL Workbench를 무료로 다운로드합니다.

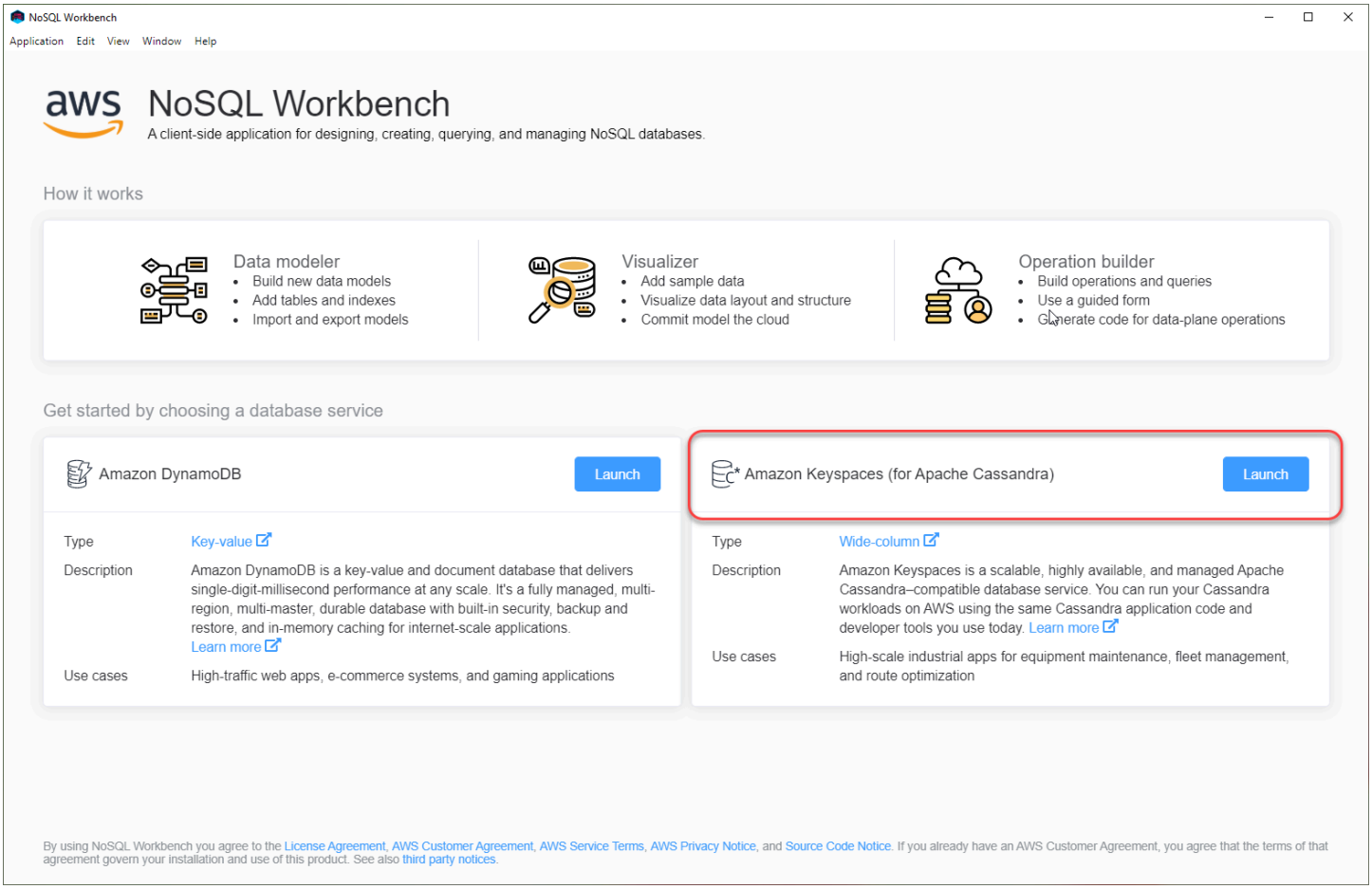
운영 체제	다운로드 링크
macOS	macOS용 다운로드
Linux*	Linux용 다운로드
Windows	Windows용 다운로드

   * NoSQL Workbench는 Ubuntu 12.04, Fedora 21, Debian 8 또는 이러한 Linux 배포의 최신 버전을 지원합니다.

2. 다운로드가 완료되면 애플리케이션을 시작하고 화면의 지침에 따라 설치를 완료합니다.

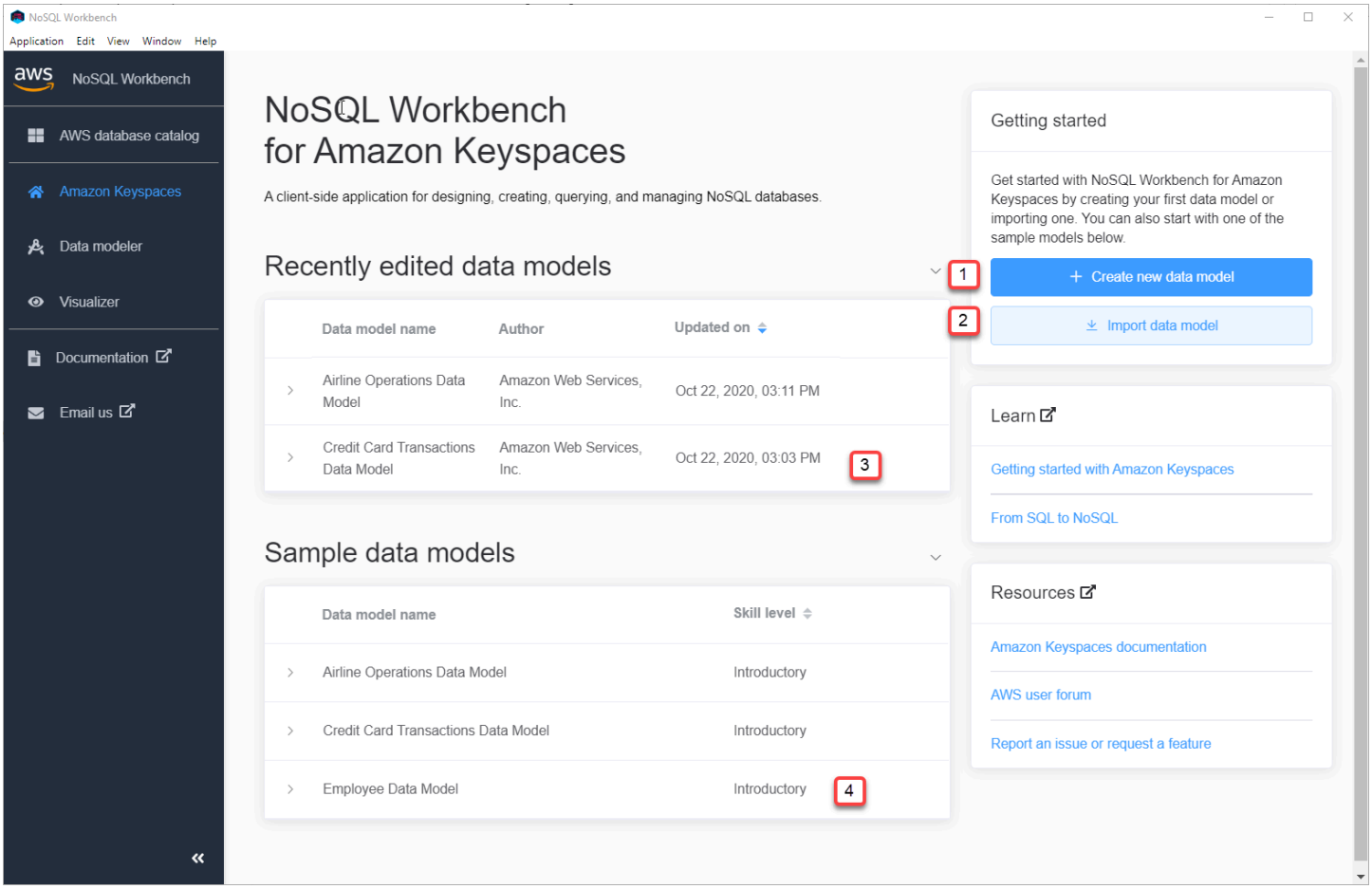
NoSQL Workbench로 시작하기

NoSQL Workbench를 시작하려면 NoSQL Workbench의 데이터베이스 카탈로그 페이지에서 Amazon Keyspaces를 선택한 다음 시작을 선택합니다.



그러면 Amazon Keyspaces의 NoSQL Workbench 홈 페이지가 열리고 다음과 같은 시작 옵션을 사용할 수 있습니다.

1. 새 데이터 모델을 생성합니다.
2. JSON 형식으로 기존 데이터 모델을 가져옵니다.
3. 최근에 편집한 데이터 모델을 엽니다.
4. 사용 가능한 샘플 모델 중 하나를 엽니다.



각 옵션은 NoSQL Workbench 데이터 모델러를 엽니다. 새 데이터 모델을 계속 생성하려면 the section called “데이터 모델 생성” 섹션을 참조하세요. 기존 데이터 모델을 편집하려면 the section called “데이터 모델 편집” 섹션을 참조하세요.

데이터 모델 구축 방법

NoSQL Workbench 데이터 모델러를 사용하여 애플리케이션의 데이터 액세스 패턴을 기반으로 새로운 데이터 모델을 설계할 수 있습니다. 데이터 모델러를 사용하여 새 데이터 모델을 설계하거나 NoSQL Workbench를 사용하여 생성된 기존 데이터 모델을 가져오고 수정할 수 있습니다. 데이터 모델러에는 데이터 모델링을 시작하는 데 도움이 되는 몇 가지 샘플 데이터 모델이 포함되어 있습니다.

주제

- NoSQL Workbench로 새 데이터 모델 구축
- NoSQL Workbench로 기존 데이터 모델 편집

NoSQL Workbench로 새 데이터 모델 구축

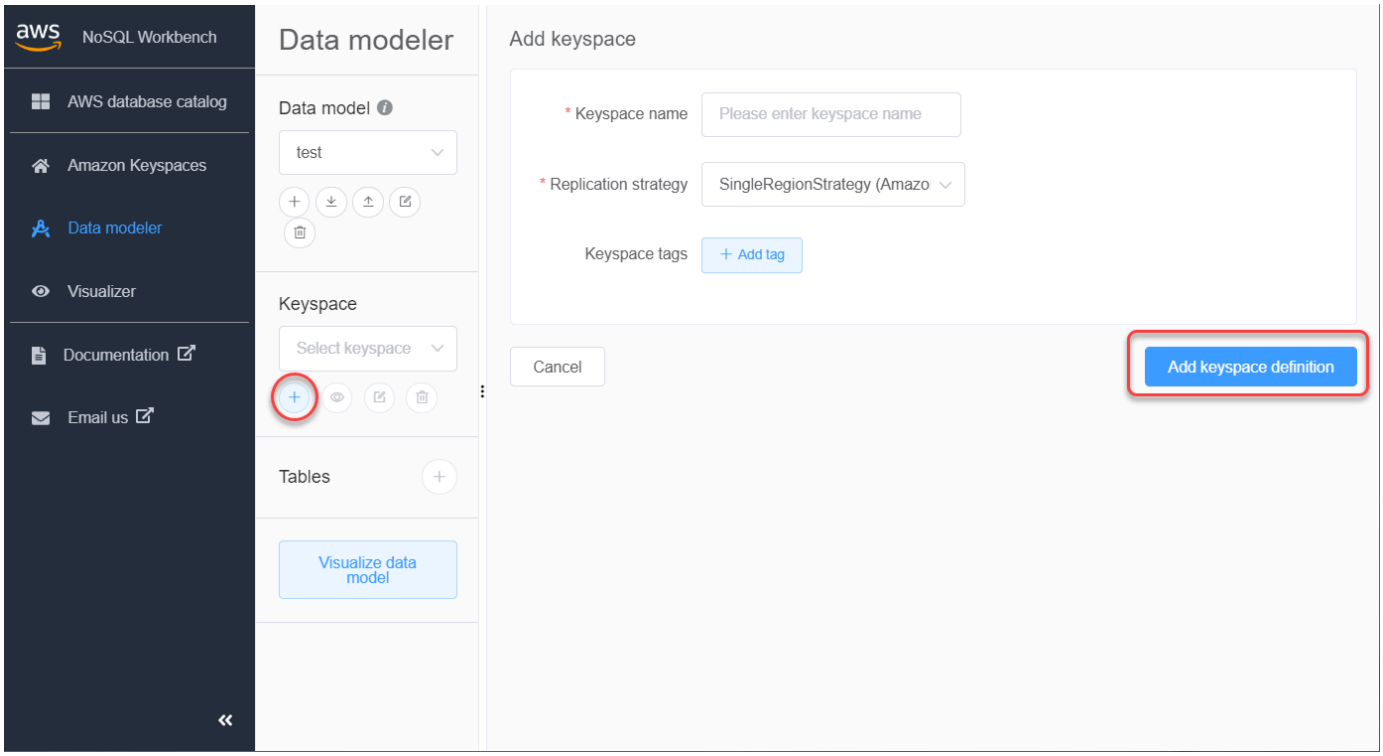
Amazon Keyspaces를 위한 새 데이터 모델을 생성하려면 NoSQL Workbench 데이터 모델러를 사용하여 키스페이스, 테이블 및 열을 생성할 수 있습니다. 다음 단계에 따라 새 데이터 모델을 생성합니다.

1. 새 키스페이스를 생성하려면 키스페이스 아래에서 더하기 기호를 선택합니다.

   이 단계에서 다음 속성과 설정을 선택합니다.

   - 키스페이스 이름 - 새 키스페이스의 이름을 입력합니다.
   - 복제 전략 - 키스페이스의 복제 전략을 선택합니다. Amazon Keyspaces는 SingleRegionStrategy를 사용하여 여러 AWS 가용 영역에 데이터를 자동으로 세 번 복제합니다. 데이터 모델을 Apache Cassandra 클러스터에 커밋할 계획인 경우 SimpleStrategy 또는 NetworkTopologyStrategy를 선택할 수 있습니다.
   - 키스페이스 태그 - 리소스 태그는 선택 사항이며 용도, 소유자, 환경 또는 다른 기준 등 다양한 방식으로 리소스를 분류할 수 있습니다. Amazon Keyspaces 리소스의 태그에 대한 자세한 내용은 the section called “태그 사용하기” 섹션을 참조하세요.

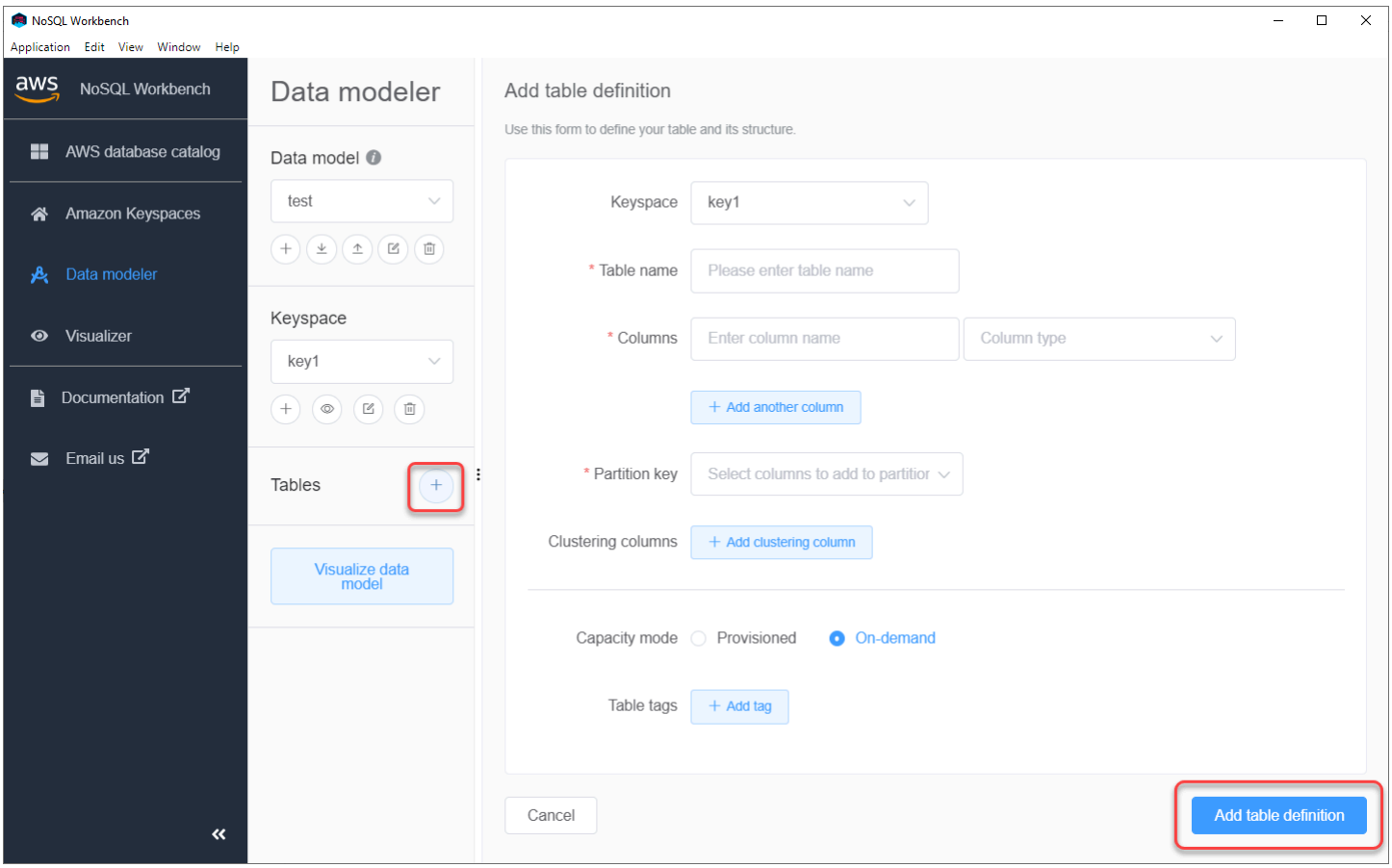
2. 키스페이스 정의 추가를 선택하여 키스페이스를 생성합니다.



3. 새 테이블을 생성하려면 Tables 옆의 더하기 기호를 선택합니다. 이 단계에서 다음 속성과 설정을 선택합니다.

- 테이블 이름 - 새 테이블의 이름
- 열 - 열 이름을 추가하고 데이터 유형을 선택합니다. 스키마의 모든 열에 대해 이 단계를 반복합니다.
- 파티션 키 - 파티션 키의 열을 선택합니다.
- 클러스터링 열 - 클러스터링 열을 선택합니다(선택 사항).
- 용량 모드 - 테이블에서 읽기/쓰기 용량 모드를 선택합니다. 프로비저닝된 용량 또는 온디맨드 용량을 선택할 수 있습니다. 용량 모드에 대한 자세한 내용은 the section called “읽기/쓰기 용량 모드” 섹션을 참조하세요.
- 테이블 태그 - 리소스 태그는 선택 사항이며 용도, 소유자, 환경 또는 다른 기준 등 다양한 방식으로 리소스를 분류할 수 있습니다. Amazon Keyspaces 리소스의 태그에 대한 자세한 내용은 the section called “태그 사용하기” 섹션을 참조하세요.

4. 테이블 정의 추가를 선택하여 새 테이블을 생성합니다.
5. 이 단계를 반복하여 추가 테이블을 생성합니다.
6. the section called “데이터 모델 시각화”를 계속해서 생성한 데이터 모델을 시각화합니다.



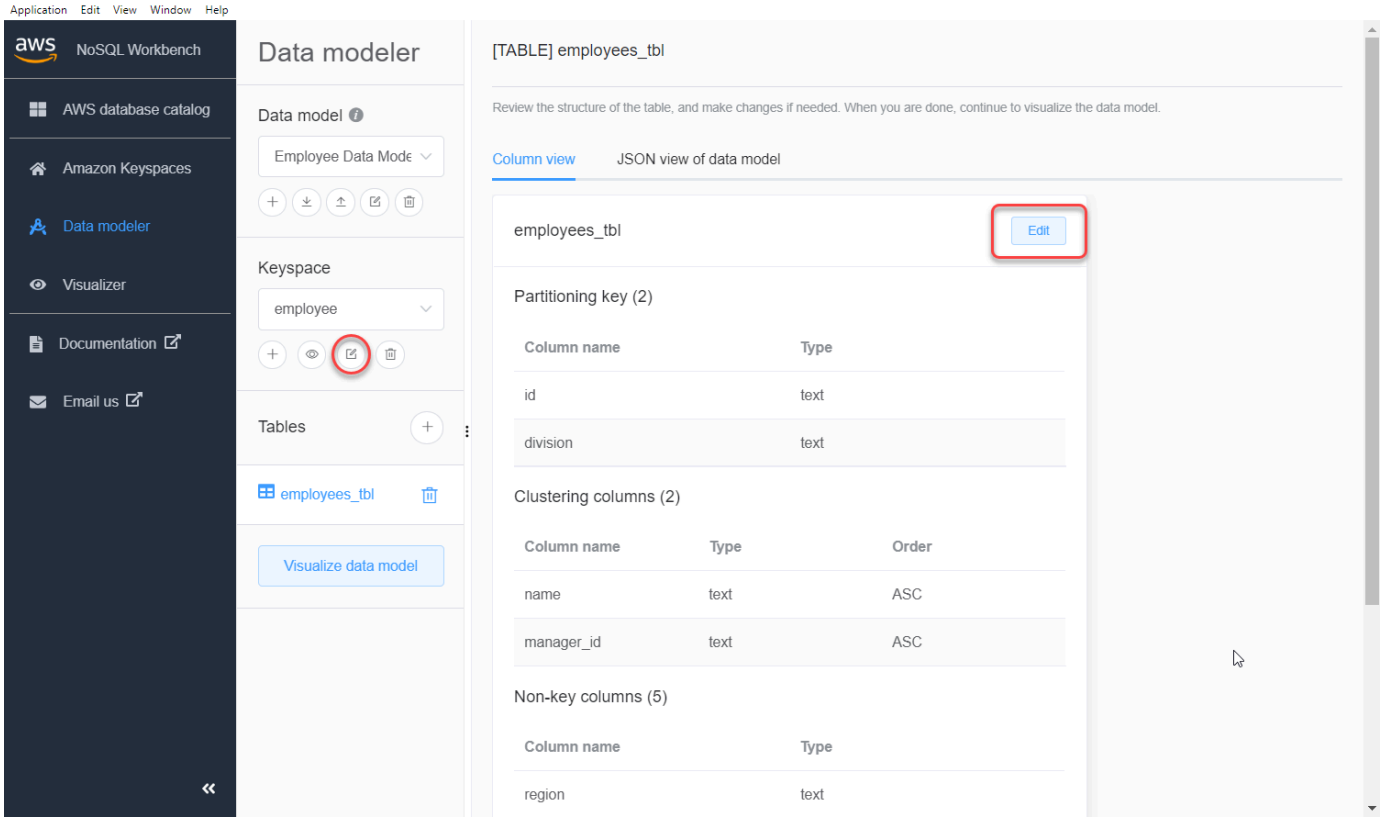
NoSQL Workbench로 기존 데이터 모델 편집

NoSQL Workbench 데이터 모델러를 사용하면 Amazon Keyspaces에서 기존 데이터 모델을 편집할 수 있습니다. 이는 파일에서 가져온 데이터 모델, 제공된 샘플 데이터 모델 또는 이전에 생성한 데이터 모델일 수 있습니다.

1. 키스페이스를 편집하려면 Keyspace에서 편집 기호를 선택합니다.

   이 단계에서 다음 속성과 설정을 편집할 수 있습니다.

   - 키스페이스 이름 - 새 키스페이스의 이름을 입력합니다.
   - 복제 전략 - 키스페이스의 복제 전략을 선택합니다. Amazon Keyspaces는 SingleRegionStrategy를 사용하여 여러 AWS 가용 영역에 데이터를 자동으로 세 번 복제합니다. 데이터 모델을 Apache Cassandra 클러스터에 커밋할 계획인 경우 SimpleStrategy 또는 NetworkTopologyStrategy를 선택할 수 있습니다.
   - 키스페이스 태그 - 리소스 태그는 선택 사항이며 용도, 소유자, 환경 또는 다른 기준 등 다양한 방식으로 리소스를 분류할 수 있습니다. Amazon Keyspaces 리소스의 태그에 대한 자세한 내용은 the section called “태그 사용하기” 섹션을 참조하세요.
2. 편집 저장을 선택하여 키스페이스를 업데이트합니다.



3. 테이블을 편집하려면 테이블 이름 옆의 편집을 선택합니다. 이 단계에서 다음 속성과 설정을 업데이트할 수 있습니다.

   - 테이블 이름 - 새 테이블의 이름
   - 열 - 열 이름을 추가하고 데이터 유형을 선택합니다. 스키마의 모든 열에 대해 이 단계를 반복합니다.
   - 파티션 키 - 파티션 키의 열을 선택합니다.
   - 클러스터링 열 - 클러스터링 열을 선택합니다(선택 사항).
   - 용량 모드 - 테이블에서 읽기/쓰기 용량 모드를 선택합니다. 프로비저닝된 용량 또는 온디맨드 용량을 선택할 수 있습니다. 용량 모드에 대한 자세한 내용은 the section called “읽기/쓰기 용량 모드” 섹션을 참조하세요.
   - 테이블 태그 - 리소스 태그는 선택 사항이며 용도, 소유자, 환경 또는 다른 기준 등 다양한 방식으로 리소스를 분류할 수 있습니다. Amazon Keyspaces 리소스의 태그에 대한 자세한 내용은 the section called “태그 사용하기” 섹션을 참조하세요.

4. 편집 저장을 선택하여 테이블을 업데이트합니다.
5. the section called “데이터 모델 시각화”를 계속해서 업데이트한 데이터 모델을 시각화합니다.

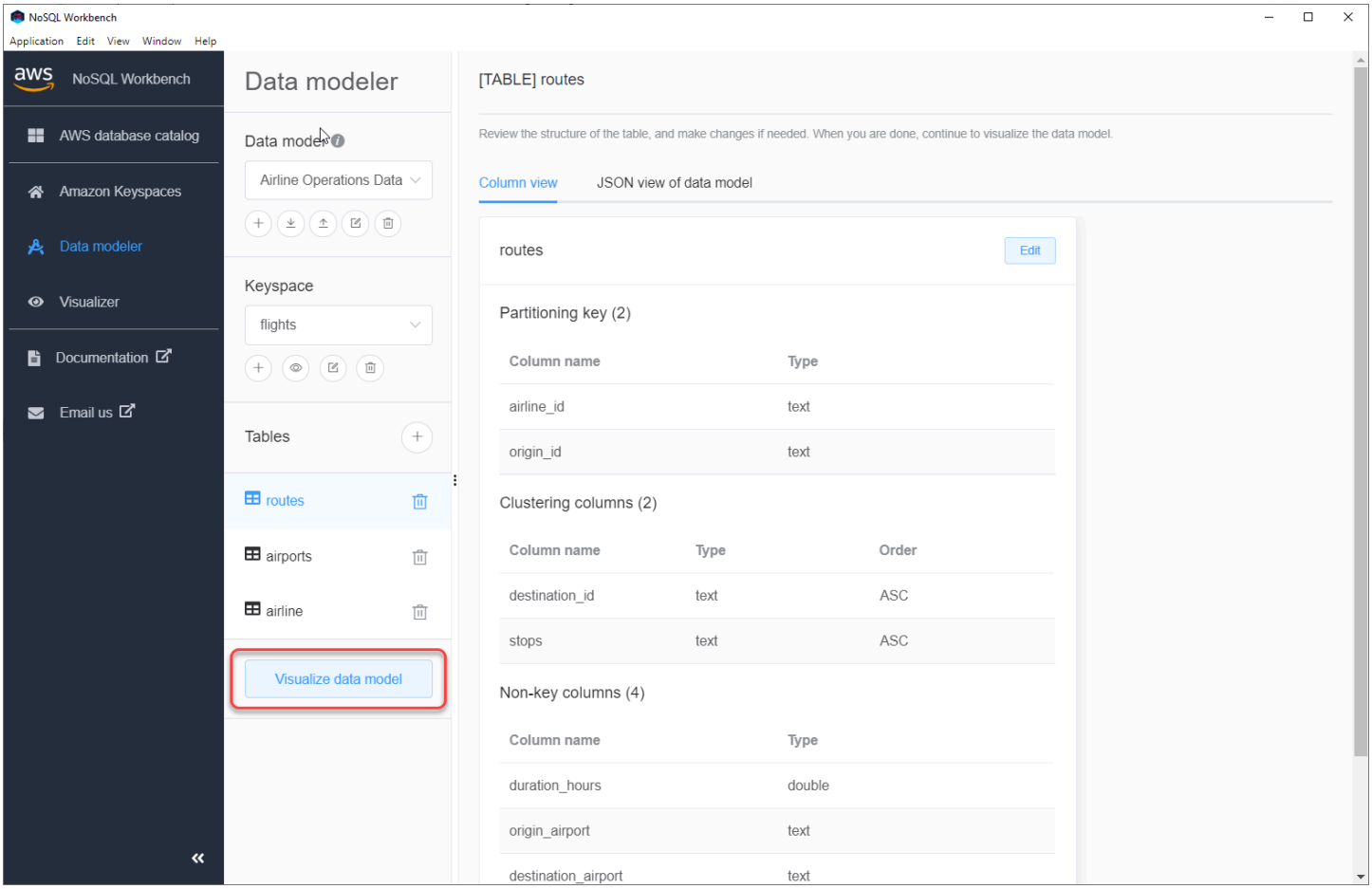
데이터 모델을 시각화하는 방법

NoSQL Workbench를 사용하면 데이터 모델을 시각화하여 데이터 모델이 애플리케이션의 쿼리 및 액세스 패턴을 지원할 수 있는지 확인할 수 있습니다. 또한 협업, 문서 및 프레젠테이션을 위해 데이터 모델을 다양한 형식으로 저장하고 내보낼 수 있습니다.

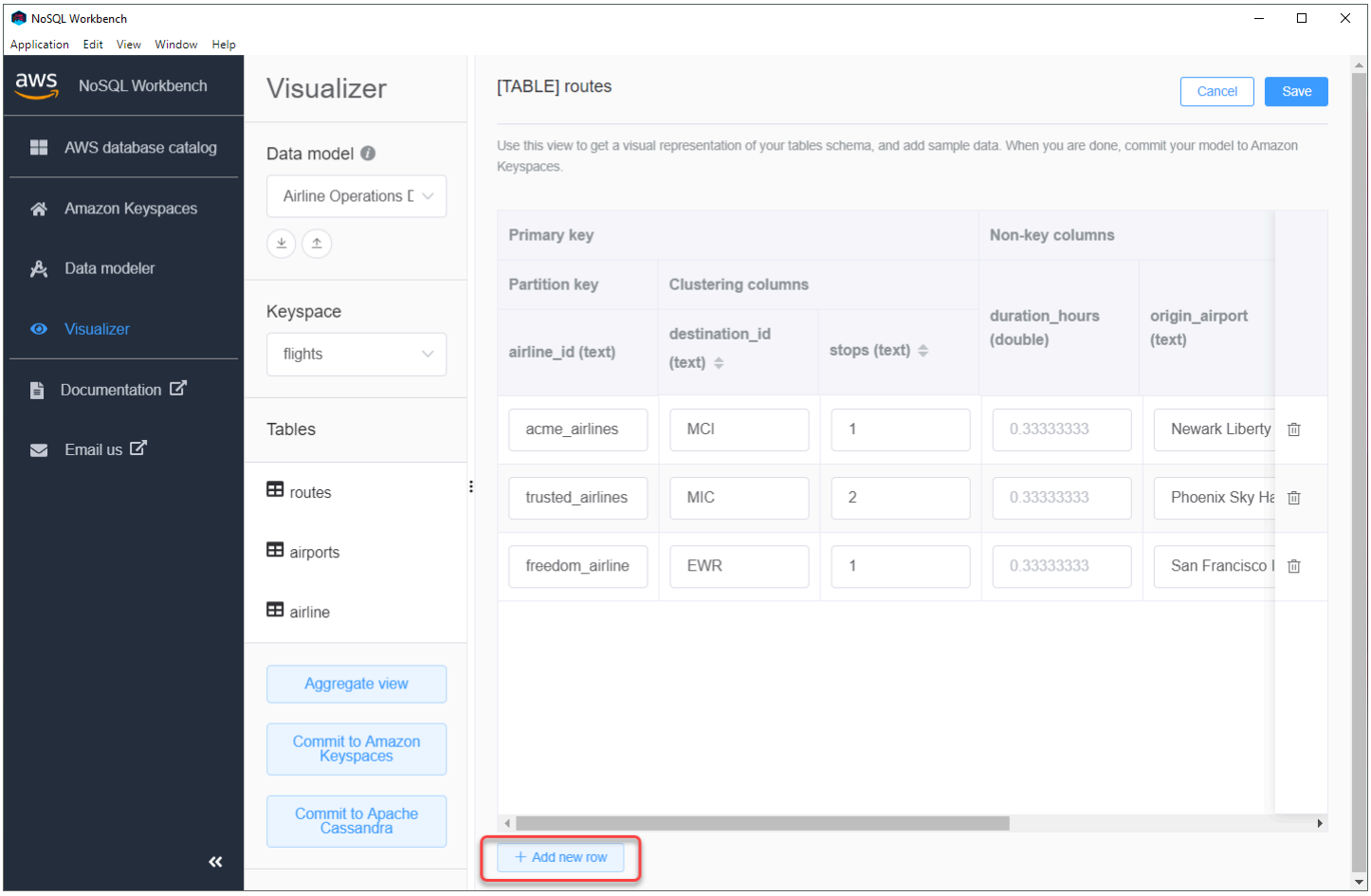
새 데이터 모델을 생성하거나 기존 데이터 모델을 편집한 후 모델을 시각화할 수 있습니다.

NoSQL Workbench로 데이터 모델 시각화

데이터 모델러에서 데이터 모델을 완성했으면 데이터 모델 시각화를 선택합니다.

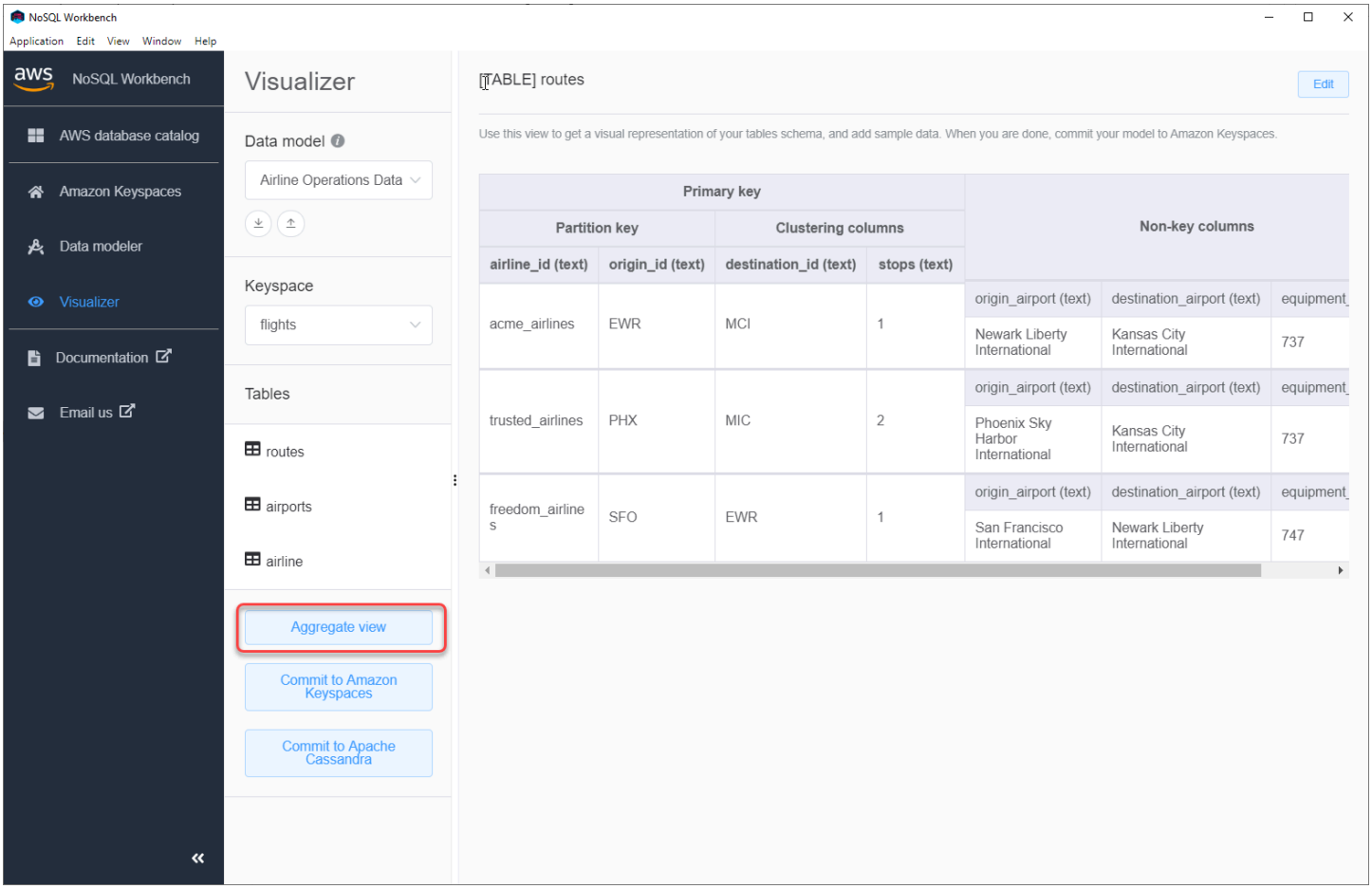


그러면 NoSQL Workbench의 데이터 비주얼라이저로 이동합니다. 데이터 비주얼라이저는 테이블 스키마를 시각적으로 표현하고 샘플 데이터를 추가할 수 있게 해줍니다. 샘플 데이터를 테이블에 추가하려면 모델에서 테이블을 선택한 다음 편집을 선택합니다. 새 데이터 행을 추가하려면 화면 하단에서 새 행 추가를 선택합니다. 완료되면 저장을 선택합니다.



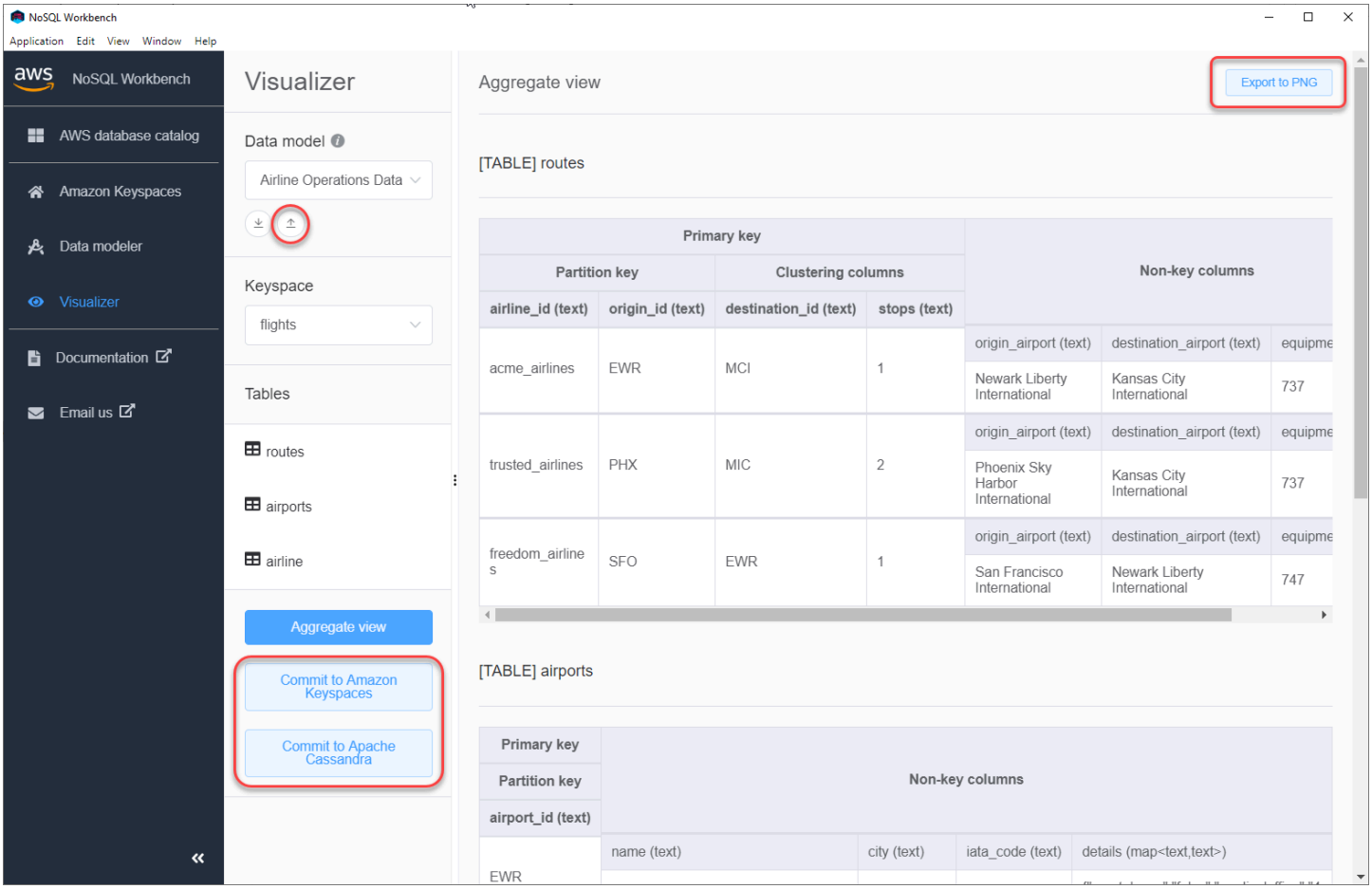
집계 보기

테이블의 스키마를 확인한 후 데이터 모델 시각화를 집계할 수 있습니다.



데이터 모델의 보기를 집계한 후 보기를 PNG 파일로 내보낼 수 있습니다. 데이터 모델을 JSON 파일로 내보내려면 데이터 모델 이름 아래에서 업로드 기호를 선택합니다.

> **Note**
>
> 디자인 과정에서 언제든지 JSON 형식으로 데이터 모델을 내보낼 수 있습니다.



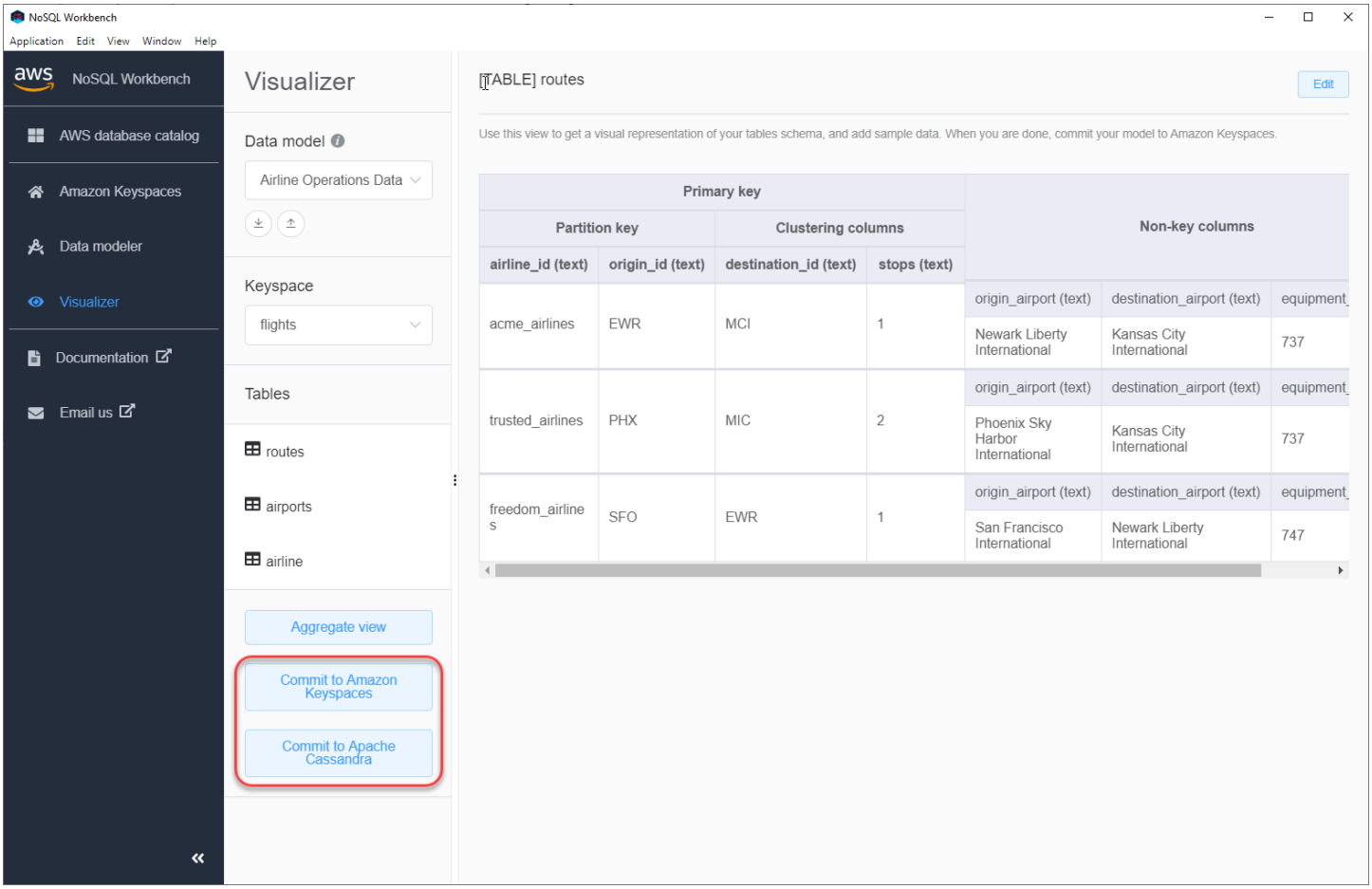
변경 사항을 커밋할 수 있는 옵션은 다음과 같습니다.

- Amazon Keyspaces에 커밋
- Apache Cassandra 클러스터에 커밋

변경 내용을 커밋하는 방법에 대한 자세한 내용은 the section called “데이터 모델 커밋” 섹션을 참조하세요.

Amazon Keyspaces와 Apache Cassandra에 데이터 모델을 커밋하는 방법

이 섹션에서는 Amazon Keypaces와 Apache Cassandra 클러스터에 완료된 데이터 모델을 커밋하는 방법을 설명합니다. 이 프로세스는 데이터 모델에서 정의한 설정을 기반으로 키스페이스와 테이블에 대한 서버측 리소스를 자동으로 생성합니다.



주제

- 시작하기 전에
- 서비스별 보안 인증을 사용하여 Amazon Keyspaces에 연결
- AWS Identity and Access Management(IAM) 보안 인증을 사용하여 Amazon Keyspaces에 연결
- 저장된 연결 사용
- Apache Cassandra에 커밋

시작하기 전에

Amazon Keyspaces에서는 클라이언트와의 연결을 보호하는 데 도움이 되는 전송 계층 보안(TLS)을 사용해야 합니다. TLS를 사용하여 Amazon Keyspaces에 연결하려면 시작하기 전에 다음 작업을 수행해야 합니다.

- 다음 명령을 사용하여 Starfield 디지털 인증서를 다운로드하고 `sf-class2-root.crt`를 로컬 또는 홈 디렉터리에 저장합니다.

```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```

> **Note**
>
> 또한 Amazon 디지털 인증서를 사용하여 Amazon Keyspaces에 접속할 수 있으며 클라이언트가 Amazon Keyspaces에 성공적으로 접속하면 계속 접속할 수 있습니다. Starfield 인증서는 이전 인증 기관을 사용하는 클라이언트에게 추가적인 이전 버전과의 호환성을 제공합니다.


```
curl https://certs.secureserver.net/repository/sf-class2-root.crt -O
```


인증서 파일을 저장한 후 Amazon Keyspaces에 연결할 수 있습니다. 한 가지 옵션은 서비스별 보안 인증을 사용하여 연결하는 것입니다. 서비스별 보안 인증은 특정 IAM 사용자와 연결된 사용자 이름 및 암호이며 지정된 서비스에서만 사용할 수 있습니다. 두 번째 옵션은 AWS 서명 버전 4 프로세스(SigV4)를 사용하는 IAM 보안 인증에 연결하는 것입니다. 이러한 옵션에 대해 자세히 알아보려면 the section called “보안 인증 정보 생성” 섹션을 참조하세요.

서비스별 보안 인증을 사용하여 연결하려면 the section called “서비스별 보안 인증을 사용하여 연결” 섹션을 참조하세요.

IAM 보안 인증을 사용하여 연결하려면 the section called “IAM 보안 인증을 사용하여 연결” 섹션을 참조하세요.

서비스별 보안 인증을 사용하여 Amazon Keyspaces에 연결

이 섹션에서는 서비스별 보안 인증을 사용하여 NoSQL Workbench에서 생성하거나 편집한 데이터 모델을 커밋하는 방법을 보여 줍니다.

1. 서비스별 보안 인증을 사용하여 새 연결을 생성하려면 사용자 이름 및 암호를 사용하여 연결 탭을 선택합니다.

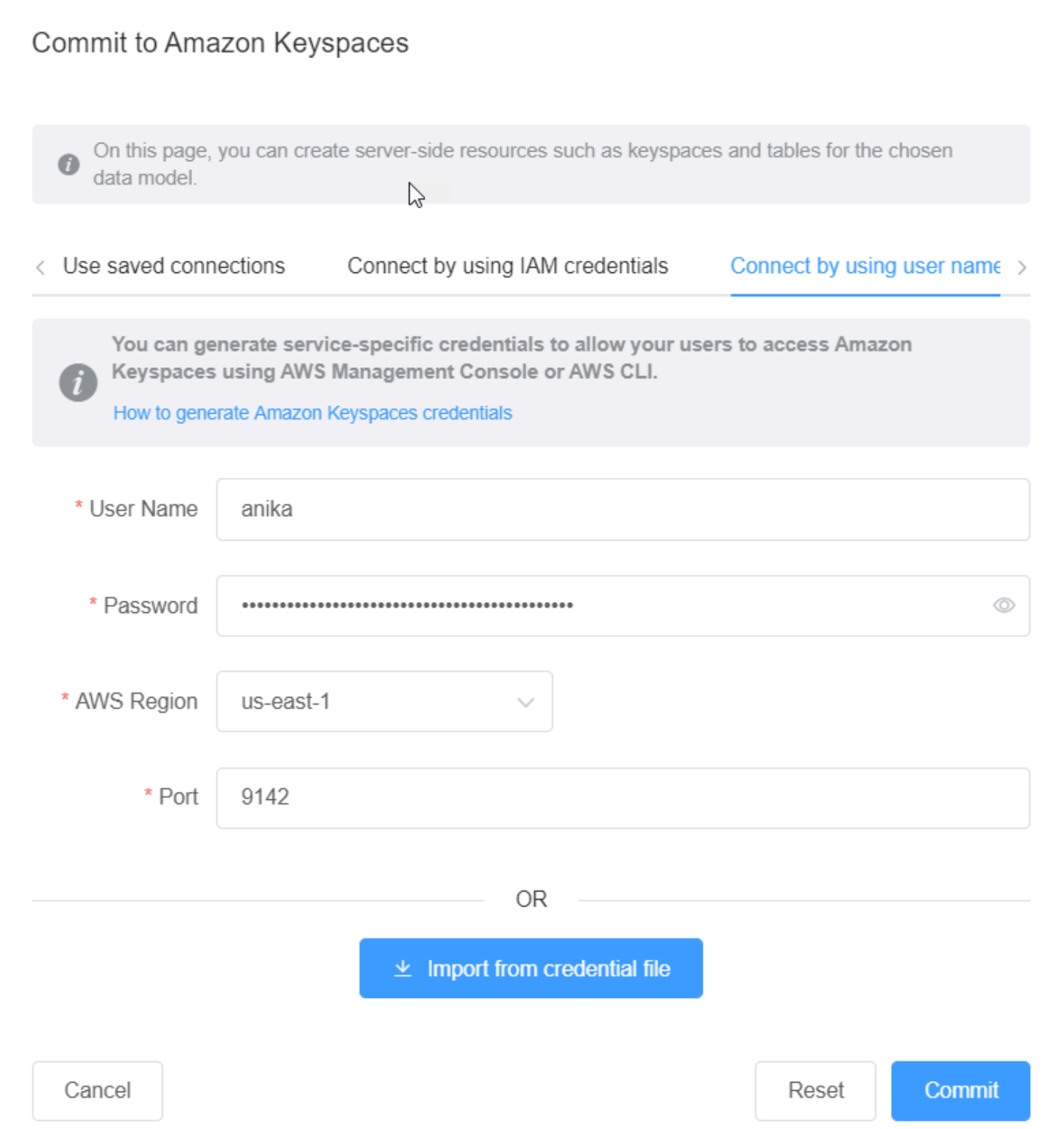
   - 시작하기 전에 the section called “서비스별 보안 인증 정보”에 설명된 프로세스를 사용하여 서비스별 보안 인증을 생성해야 합니다.

서비스별 보안 인증을 획득한 후에는 연결 설정을 계속할 수 있습니다. 다음 중 하나를 계속합니다.

- 사용자 이름 - 사용자 이름을 입력합니다.
- 암호 - 암호를 입력합니다.
- AWS 리전 - 사용 가능한 리전은 the section called “서비스 엔드포인트” 섹션을 참조하세요.
- 포트 - Amazon Keyspaces는 포트 9142를 사용합니다.

또는 파일에서 저장된 보안 인증을 가져올 수 있습니다.

2. 커밋을 선택하여 Amazon Keyspaces를 데이터 모델로 업데이트합니다.


Commit to Amazon Keyspaces
On this page, you can create server-side resources such as keyspaces and tables for the chosen data model.
Use saved connections
Connect by using IAM credentials
Connect by using user name
You can generate service-specific credentials to allow your users to access Amazon Keyspaces using AWS Management Console or AWS CLI.
How to generate Amazon Keyspaces credentials
* User Name
anika
* Password
* AWS Region
us-east-1
* Port
9142
OR
Import from credential file
Cancel
Reset
Commit

AWS Identity and Access Management(IAM) 보안 인증을 사용하여 Amazon Keyspaces에 연결

이 섹션에서는 IAM 보안 인증을 사용하여 NoSQL Workbench에서 생성하거나 편집한 데이터 모델을 커밋하는 방법을 보여 줍니다.

1. IAM 보안 인증을 사용하여 새 연결을 생성하려면 IAM 보안 인증을 사용하여 연결 탭을 선택합니다.

   - 시작하기 전에 다음 방법 중 하나를 사용하여 IAM 보안 인증을 생성해야 합니다.

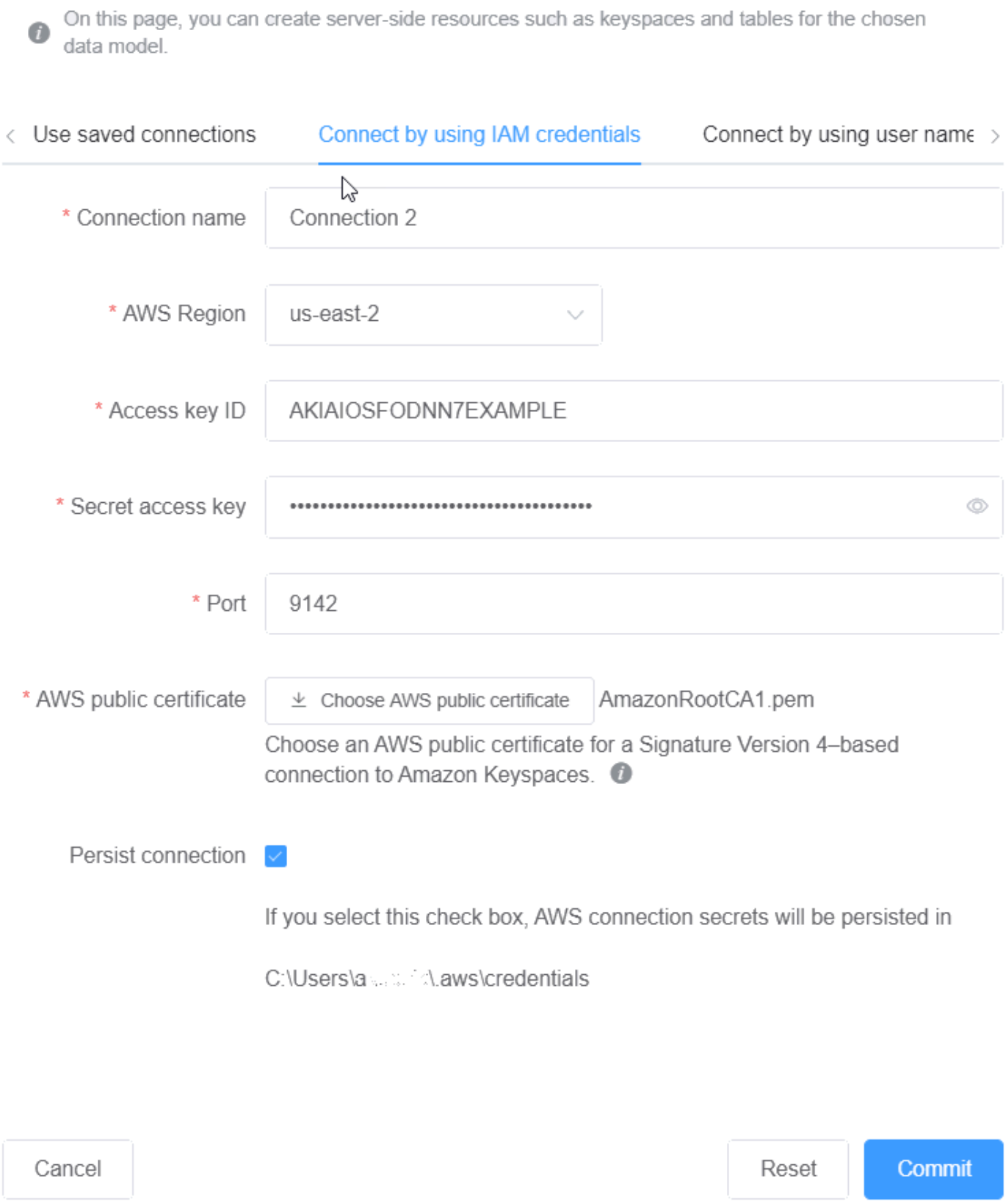
     - 콘솔 액세스의 경우 IAM 사용자 이름과 암호를 사용하여 IAM 로그인 페이지에서 AWS Management Console에 로그인합니다. 프로그래밍 방식 액세스 및 장기 보안 인증에 대한 대안을 포함한 AWS 보안 인증에 대한 자세한 내용은 IAM 사용 설명서의 AWS 보안 인증을 참조하세요. AWS 계정에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS에 로그인하는 방법을 참조하세요.
     - CLI 액세스를 위해서는 액세스 키 ID 및 비밀 액세스 키가 필요합니다. 가능하다면 장기 액세스 키 대신 임시 보안 인증을 사용합니다. 임시 보안 인증도 액세스 키 ID와 비밀 액세스 키로 구성되지만 보안 인증이 만료되는 시간을 나타내는 보안 토큰이 포함되어 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 리소스에서 임시 보안 인증 사용을 참조하세요.
     - API 액세스의 경우 액세스 키 ID 및 보안 액세스 키가 필요합니다. AWS 계정 루트 사용자 액세스 키 대신에 IAM 사용자 액세스 키를 사용합니다. 액세스 키 생성에 대한 자세한 내용은 IAM 사용 설명서의 IAM 사용자를 위한 액세스 키 관리를 참조하세요.

     자세한 내용은 IAM 사용자의 액세스 키 관리를 참조하세요.

   IAM 보안 인증을 획득한 후에는 연결 설정을 계속할 수 있습니다.

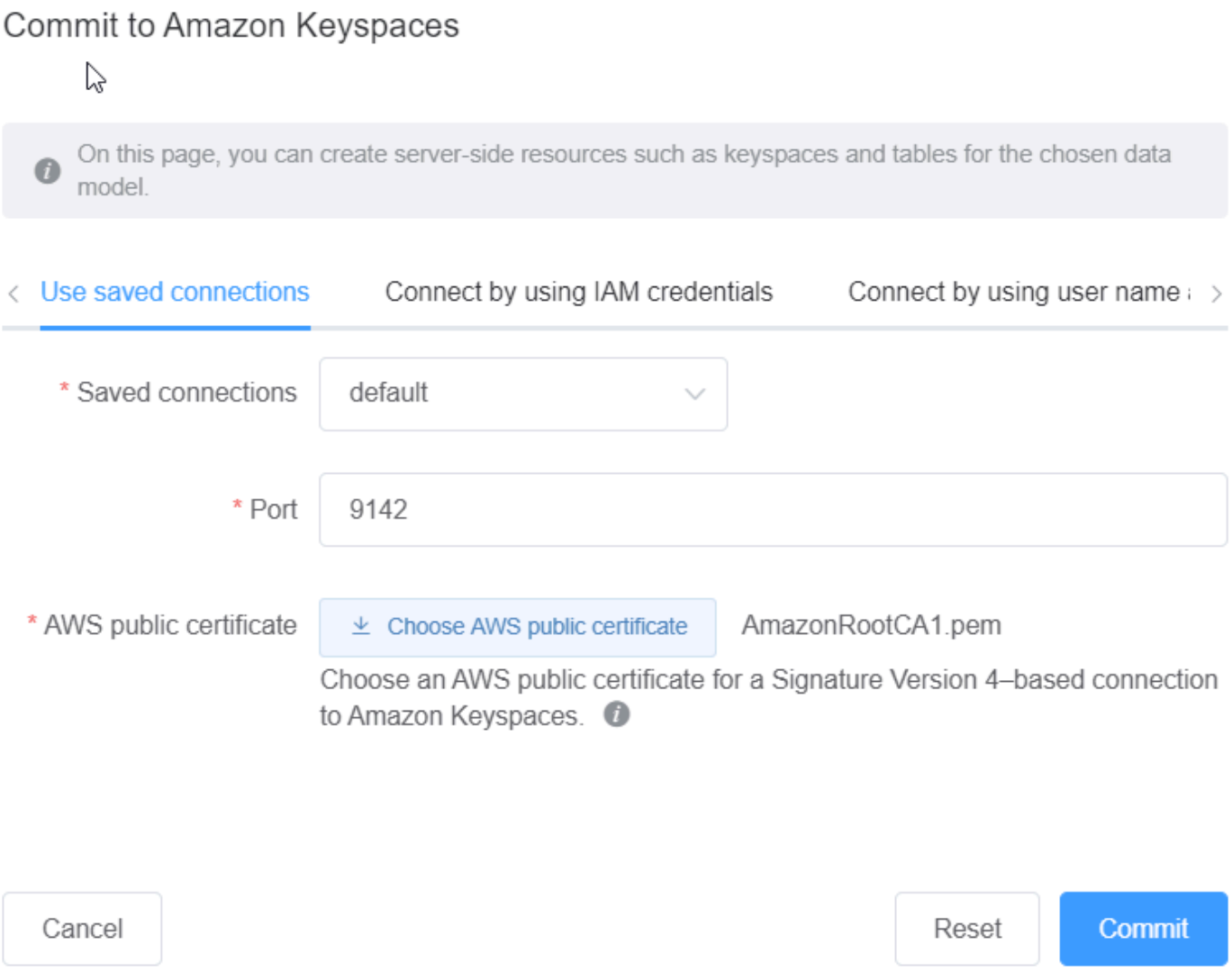
   - 연결 이름 - 연결 이름입니다.
   - AWS 리전 - 사용 가능한 리전은 the section called “서비스 엔드포인트” 섹션을 참조하세요.
   - 액세스 키 ID – 액세스 키 ID를 입력합니다.
   - 비밀 액세스 키 - 비밀 액세스 키를 입력합니다.
   - 포트 - Amazon Keyspaces는 포트 9142를 사용합니다.
   - AWS 퍼블릭 인증서 - 첫 번째 단계에서 다운로드한 AWS 인증서를 가리킵니다.
   - 연결 유지 - AWS 연결 암호를 로컬에 저장하려는 경우 이 확인란을 선택합니다.

2. 커밋을 선택하여 Amazon Keyspaces를 데이터 모델로 업데이트합니다.


On this page, you can create server-side resources such as keyspaces and tables for the chosen data model.
Use saved connections
Connect by using IAM credentials
Connect by using user name
* Connection name
Connection 2
* AWS Region
us-east-2
* Access key ID
AKIAIOSFODNN7EXAMPLE
* Secret access key
* Port
9142
* AWS public certificate
Choose AWS public certificate
AmazonRootCA1.pem
Choose an AWS public certificate for a Signature Version 4–based connection to Amazon Keyspaces.
Persist connection
If you select this check box, AWS connection secrets will be persisted in
C:\Users\a
\.aws\credentials
Cancel
Reset
Commit

저장된 연결 사용

이전에 Amazon Keyspaces에 대한 연결을 설정한 경우 이를 기본 연결로 사용하여 데이터 모델 변경을 커밋할 수 있습니다. 저장된 연결 사용 탭을 선택하고 업데이트를 계속 커밋합니다.

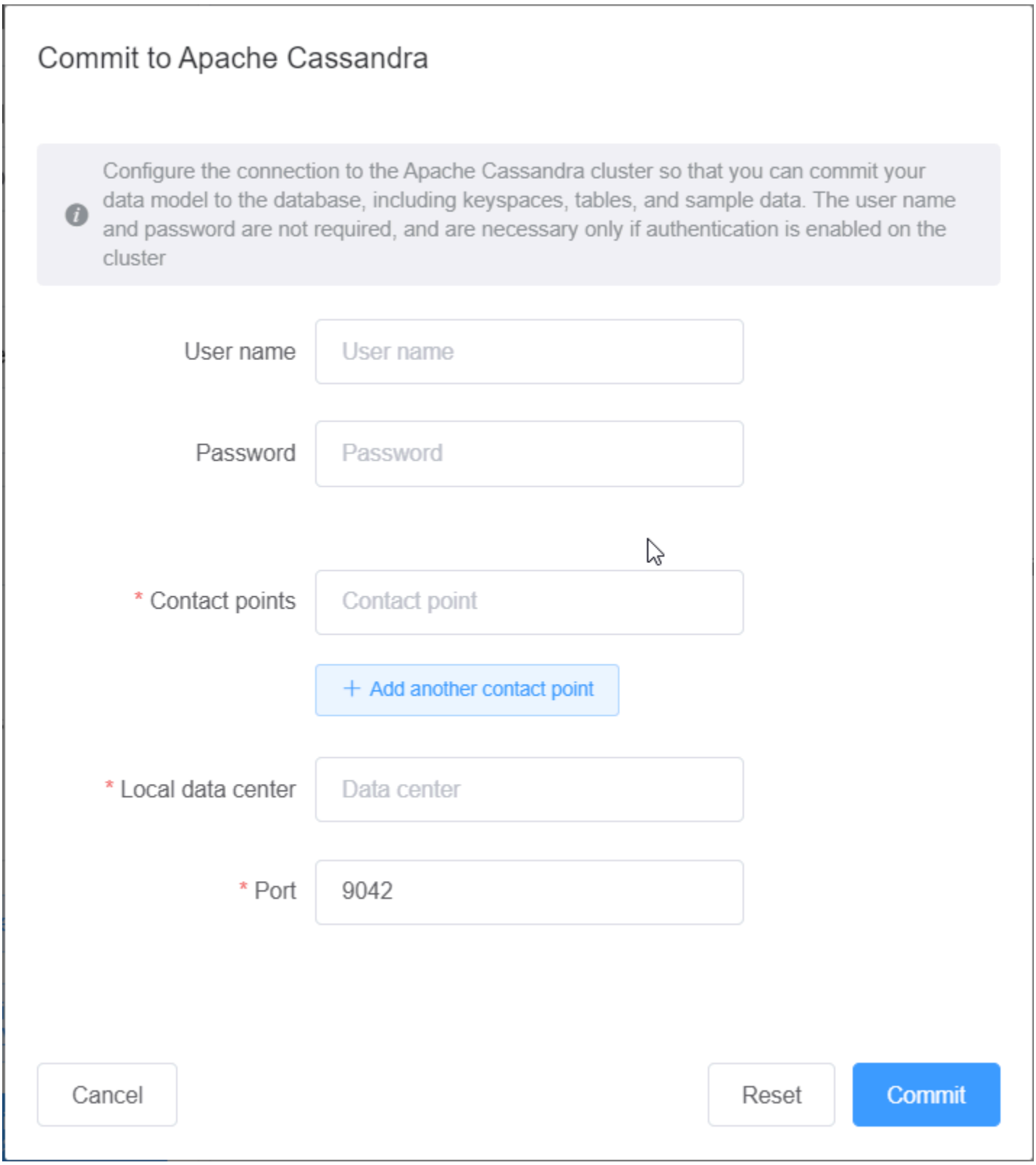


Apache Cassandra에 커밋

이 섹션에서는 NoSQL Workbench로 생성 또는 편집한 데이터 모델을 커밋하기 위해 Apache Cassandra 클러스터에 연결하는 방법을 안내합니다.

> **Note**
>
> `SimpleStrategy` 또는 `NetworkTopologyStrategy`로 생성된 데이터 모델만 Apache Cassandra 클러스터에 커밋할 수 있습니다. 복제 전략을 변경하려면 데이터 모델러에서 키스페이스를 편집합니다.

1.
   - 사용자 이름 - 클러스터에서 인증이 활성화된 경우 사용자 이름을 입력합니다.
   - 암호 - 클러스터에서 인증이 활성화된 경우 암호를 입력합니다.
   - Contact points - 연락처를 입력합니다.
   - 로컬 데이터 센터 - 로컬 데이터 센터의 이름을 입력합니다.
   - 포트 - 연결에는 포트 9042가 사용됩니다.
2. 커밋을 선택하여 Apache Cassandra 클러스터를 데이터 모델로 업데이트합니다.


Commit to Apache Cassandra
Configure the connection to the Apache Cassandra cluster so that you can commit your data model to the database, including keyspaces, tables, and sample data. The user name and password are not required, and are necessary only if authentication is enabled on the cluster
User name
User name
Password
Password
* Contact points
Contact point
+ Add another contact point
* Local data center
Data center
* Port
9042
Cancel
Reset
Commit

NoSQL Workbench의 샘플 데이터 모델

모델러 및 비주얼라이저의 홈 페이지에는 NoSQL Workbench와 함께 제공되는 여러 샘플 모델이 나와 있습니다. 이 섹션에서는 이러한 모델 및 사용 가능한 용도에 대해 설명합니다.

주제

직원 데이터 모델

이 데이터 모델은 직원 데이터베이스 애플리케이션을 위한 Amazon Keyspaces 스키마를 나타냅니다.

특정 회사의 직원 정보에 액세스하는 애플리케이션은 이 데이터 모델을 사용할 수 있습니다.

이 데이터 모델에서 지원되는 액세스 패턴은 다음과 같습니다.

- 지정된 ID로 직원 기록 검색
- 지정된 ID 및 부서로 직원 기록 검색
- 지정된 ID 및 이름으로 직원 기록 검색

신용카드 거래 데이터 모델

이 데이터 모델은 소매점에서의 신용 카드 거래를 위한 Amazon Keyspaces 스키마를 나타냅니다.

신용 카드 거래를 저장하면 매장의 장부 관리에 도움이 될 뿐만 아니라 매장 관리자가 구매 동향을 분석하여 예측 및 계획을 세우는 데도 도움이 됩니다.

이 데이터 모델에서 지원되는 액세스 패턴은 다음과 같습니다.

- 신용 카드 번호, 월, 연도, 날짜별로 거래 검색
- 신용 카드 번호, 범주, 날짜별로 거래 검색
- 범주, 위치, 신용 카드 번호별로 거래 검색
- 신용 카드 번호, 분쟁 상태로 거래 검색

항공사 운영 데이터 모델

이 데이터 모델은 공항, 항공사 및 비행 경로를 포함한 비행기 항공편에 대한 데이터를 보여 줍니다.

시연되는 Amazon Keyspaces 모델링의 주요 구성 요소는 키-값 페어, 와이드 열 데이터 저장소, 복합 키 및 일반적인 NoSQL 데이터 액세스 패턴을 보여 주기 위한 맵과 같은 복잡한 데이터 유형입니다.

이 데이터 모델에서 지원되는 액세스 패턴은 다음과 같습니다.

- 지정된 공항의 지정된 항공사에서 출발하는 노선 검색
- 지정된 목적지 공항의 노선 검색
- 직항편이 있는 공항 검색
- 공항 세부 정보 및 항공사 세부 정보 검색

NoSQL Workbench 릴리스 기록

다음 테이블은 NoSQL Workbench 클라이언트 측 애플리케이션의 각 릴리스별 중요 변경 사항을 설명합니다.

변경 사항	설명	날짜
Amazon Keyspaces용 NoSQL Workbench — 정식 버전	Amazon Keyspaces용 NoSQL Workbench가 정식 버전입니다.	2020년 10월 28일
릴리스된 NoSQL Workbench 평가판	NoSQL Workbench는 Amazon Keyspaces의 비관계형 데이터 모델을 보다 쉽게 설계하고 시각화할 수 있는 클라이언트 측 애플리케이션입니다. NoSQL Workbench 클라이언트는 Windows, macOS, Linux에서 사용할 수 있습니다. 자세한 내용은 Amazon Keyspaces용 NoSQL Workbench를 참조하세요.	2020년 10월 5일

Amazon Keyspaces(Apache Cassandra용)의 다중 리전 복제

Amazon Keyspaces 다중 지역 복제를 사용하면 선택한 영역에 걸쳐 자동화된 완전 관리형 액티브-액티브 복제를 통해 데이터를 복제할 수 있습니다. AWS 리전 액티브-액티브 복제를 사용하면 각 리전이 개별적으로 읽기 및 쓰기를 수행할 수 있습니다. 지역적 성능 저하를 통해 가용성과 복원력을 모두 개선하는 동시에 글로벌 애플리케이션의 로컬 읽기 및 쓰기 지연 시간이 짧은 이점을 누릴 수 있습니다.

다중 리전 복제를 사용하면 Amazon Keyspaces가 리전 간에 데이터를 비동기적으로 복제하며 데이터는 일반적으로 1초 이내에 리전 간에 전파됩니다. 또한 다중 리전 복제를 사용하면 충돌을 해결하고 데이터 분산 문제를 해결하는 어려운 작업이 필요 없어 애플리케이션에만 집중할 수 있습니다.

기본적으로 Amazon Keyspace는 내구성과 고가용성을 AWS 리전 위해 동일한 가용 영역 내 세 개의 가용 영역에 데이터를 복제합니다. 다중 지역 복제를 사용하면 원하는 지리적 위치 최대 6개까지 테이블을 복제하는 다중 지역 키스페이스를 생성할 수 있습니다. AWS 리전

주제

다중 지역 복제 사용의 이점

다중 지역 복제는 다음과 같은 이점을 제공합니다.

- 지연 시간이 10밀리초 미만인 글로벌 읽기 및 쓰기 — Amazon Keyspaces에서는 복제가 액티브-액티브 방식입니다. 규모에 상관없이 한 자릿수 밀리초 지연 시간으로 고객과 가장 가까운 리전에서 로컬로 읽기와 쓰기를 모두 제공할 수 있습니다. 전 세계 어디서나 빠른 응답 시간이 필요한 글로벌 애플리케이션에 Amazon Keyspaces 다중 리전 테이블을 사용할 수 있습니다.
- 비즈니스 연속성 개선 및 단일 지역 성능 저하로부터 보호 — 다중 지역 복제를 사용하면 다중 지역 키스페이스의 다른 지역으로 애플리케이션을 AWS 리전 리디렉션하여 단일 지역에서 성능 저하를 복구할 수 있습니다. Amazon Keyspaces는 액티브-액티브 복제를 제공하므로 읽기 및 쓰기에 영향을 주지 않습니다.

Amazon Keyspaces는 다중 리전 키스페이스에서 수행된 쓰기 기록을 유지하지만 모든 복제 리전으로 전파하지는 않습니다. 지역이 다시 온라인 상태가 되면 Amazon Keyspaces는 누락된 변경 사항을 자동으로 동기화하므로 애플리케이션에 영향을 주지 않고 복구할 수 있습니다.

- 지역 간 고속 복제 — 다중 지역 복제는 지역 간 데이터의 빠른 스토리지 기반 물리적 복제를 사용하며, 복제 지연은 일반적으로 1초 미만입니다.

  Amazon Keyspaces에서의 복제는 애플리케이션과 컴퓨팅 리소스를 공유하지 않기 때문에 데이터베이스 쿼리에 거의 또는 전혀 영향을 주지 않습니다. 즉, 애플리케이션에 영향을 주지 않고도 쓰기 처리량이 높은 사용 사례 또는 처리량이 갑자기 급증하거나 급증하는 사용 사례를 해결할 수 있습니다.

- 일관성 및 충돌 해결 — 특정 지역의 데이터에 대한 모든 변경 사항은 다중 지역 키스페이스의 다른 지역에 복제됩니다. 애플리케이션이 다른 리전에서 동시에 동일한 데이터를 업데이트하는 경우 충돌이 발생할 수 있습니다.

  최종 일관성을 제공하기 위해 Amazon Keyspaces는 셀 수준의 타임스탬프와 동시 업데이트 간에 최종 쓰기 우선 조정을 사용합니다. 충돌 해결은 완벽하게 관리되며 애플리케이션에 영향을 주지 않고 백그라운드에서 이루어집니다.

지원되는 구성 및 기능에 대한 자세한 내용은 the section called “사용 노트” 섹션을 참조하세요.

용량 모드 및 가격

다중 리전 키스페이스의 경우 온디맨드 용량 모드 또는 프로비저닝된 용량 모드를 사용할 수 있습니다. 자세한 설명은 the section called “읽기/쓰기 용량 모드” 섹션을 참조하세요.

온디맨드 모드의 경우 행당 최대 1KB의 데이터를 쓰려면 쓰기 요청 단위 (WRU) 1.25가 청구됩니다. 다중 리전 키스페이스의 각 리전에 대한 쓰기 요금이 청구됩니다. 예를 들어 두 리전이 있는 멀티 리전 키스페이스에 3KB의 데이터 행을 쓰려면 7.5WRU(3 * 1.25 * 2 = 7.5WRU)가 필요합니다. 또한 정적 데이터와 비정적 데이터를 모두 포함하는 쓰기에는 추가 쓰기 작업이 필요합니다.

프로비저닝 모드의 경우 행당 최대 1KB의 데이터를 쓰는 데 1.25WCU (쓰기 용량 단위) 가 청구됩니다. 다중 리전 키스페이스의 각 리전에 대한 쓰기 요금이 청구됩니다. 예를 들어 두 지역이 있는 멀티 리전 키스페이스에 초당 3KB의 데이터 행을 쓰려면 7.5 WCU (3 * 1.25 * 2 = 7.5 WCU) 가 필요합니다. 또한 정적 데이터와 비정적 데이터를 모두 포함하는 쓰기에는 추가 쓰기 작업이 필요합니다.

요금에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 요금을 참조하세요.

Amazon Keyspaces의 다중 리전 복제가 작동하는 방식

이 섹션에서는 Amazon Keyspaces 다중 리전 복제의 작동 방식에 대해 간략히 살펴봅니다. 요금에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 요금을 참조하세요.

주제

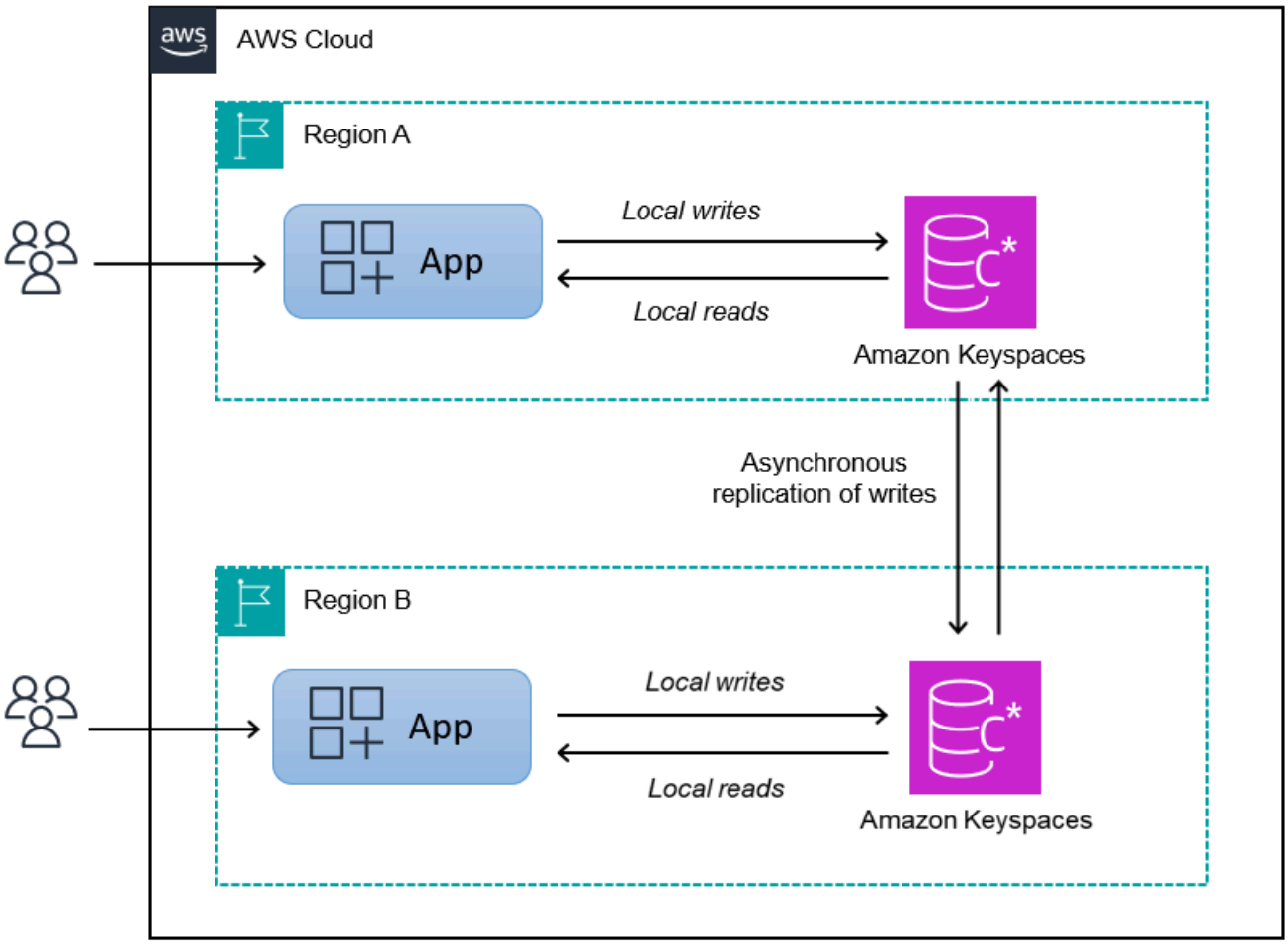
Amazon Keyspaces의 다중 리전 복제가 작동하는 방식

Amazon Keyspaces 다중 리전 복제는 독립적이고 지리적으로 분산된 AWS 리전에 데이터를 분산하는 데이터 복원력 아키텍처를 구현합니다. 액티브-액티브 복제를 사용하므로 각 리전이 개별적으로 읽기 및 쓰기를 수행할 수 있어 로컬 지연 시간이 짧습니다.

Amazon Keyspaces 다중 리전 키스페이스를 생성할 때 데이터를 복제할 리전을 최대 5개까지 추가로 선택할 수 있습니다. 다중 리전 키스페이스에서 생성하는 각 테이블은 Amazon Keyspaces가 이를 단일 단위로 간주하는 여러 개의 복제 테이블(리전당 한 개씩)로 구성됩니다.

모든 복제본은 동일한 테이블 이름과 동일한 프라이머리 키 스키마를 갖습니다. 애플리케이션이 한 리전의 로컬 테이블에 데이터를 쓰면 `LOCAL_QUORUM` 일관성 수준을 사용하여 데이터가 안정적으로 쓰여집니다. Amazon Keyspaces는 데이터를 다른 복제 리전에 비동기적으로 자동 복제합니다. 리전 간 복제 지연은 일반적으로 1초 미만이며 애플리케이션의 성능이나 처리량에 영향을 미치지 않습니다.

데이터를 쓴 후에는 `LOCAL_ONE/LOCAL_QUORUM` 일관성 수준이 있는 다른 복제 리전의 다중 리전 테이블에서 데이터를 읽을 수 있습니다. 지원되는 구성 및 기능에 대한 자세한 내용은 the section called “사용 노트” 섹션을 참조하세요.



다중 리전 복제 충돌 해결

Amazon Keyspaces 다중 리전 복제는 완전히 관리되므로 데이터 동기화 문제를 정리하기 위해 정기적으로 복구 작업을 실행하는 것과 같은 복제 작업을 수행할 필요가 없습니다. Amazon Keyspace는 충돌을 감지 및 AWS 리전 수정하여 서로 다른 테이블 간의 데이터 일관성을 모니터링하고 복제본을 자동으로 동기화합니다.

Amazon Keyspaces는 데이터 조정의 최종 쓰기 우선 방법을 사용합니다. 이러한 충돌 해결 메커니즘을 사용하여 다중 리전 키스페이스의 모든 리전은 최종 업데이트에 합의하며 모두 동일한 데이터를 갖는 상태가 됩니다. 조정 프로세스는 애플리케이션 성능에 영향을 미치지 않습니다. 충돌 해결을 지원하기 위해 다중 리전 테이블의 경우 클라이언트 측 타임스탬프가 자동으로 설정되며 해제할 수 없습니다. 자세한 정보는 클라이언트 측 타임스탬프을 참조하세요.

다중 리전 복제 재해 복구

Amazon Keyspaces 다중 지역 복제를 사용하면 쓰기가 각 지역에 비동기적으로 복제됩니다. 드문 경우지만 단일 리전의 성능 저하 또는 장애가 발생하는 경우 다중 리전 복제를 사용하면 애플리케이션에 거의 또는 전혀 영향을 주지 않고 재해를 복구할 수 있습니다. 재해 복구는 일반적으로 Recovery Time Objective(RTO) 및 Recovery Point Objective(RPO)를 사용하여 측정됩니다.

복구 시간 목표 – 재해 발생 후 시스템이 정상 작동 상태로 돌아가는 데 걸리는 시간입니다. RTO는 워크로드가 허용할 수 있는 가동 중지를 시간 단위로 측정합니다. 다중 리전 복제를 사용하여 영향을 받지 않는 리전으로 장애 조치하는 재해 복구 계획의 경우 RTO는 거의 0에 가까울 수 있습니다. RTO는 애플리케이션이 장애 상태를 감지하고 트래픽을 다른 리전으로 리디렉션하는 속도에 따라 제한됩니다.

복구 시점 목표 – 손실될 수 있는 데이터의 양(시간으로 측정)입니다. 다중 리전 복제를 사용하여 영향을 받지 않는 리전으로 장애 조치하는 재해 복구 계획의 경우 RPO는 일반적으로 10초 미만입니다. RPO는 장애 조치 대상 복제본에 대한 복제 지연 시간으로 제한됩니다.

Amazon Keyspaces에서의 복제는 액티브-액티브 방식이므로 리전 장애 또는 성능 저하가 발생하는 경우 보조 리전을 승격하거나 데이터베이스 장애 조치 절차를 수행할 필요가 없습니다. 대신 Amazon Route 53을 사용하여 가장 가까운 정상 리전으로 애플리케이션을 라우팅할 수 있습니다. Route 53에 대해 자세히 알아보려면 Amazon Route 53은 무엇인가요?를 참조하세요.

단일 AWS 리전 객체가 격리되거나 성능이 저하되는 경우 애플리케이션은 Route 53을 사용하여 트래픽을 다른 지역으로 리디렉션하여 다른 복제 테이블에 대해 읽기 및 쓰기를 수행할 수 있습니다. 또한 요청을 언제 다른 리전으로 리디랙션할지를 결정하는 사용자 지정 비즈니스 로직을 적용할 수 있습니다. 예를 들어 애플리케이션이 사용 가능한 여러 엔드포인트를 인식하도록 하는 경우를 들 수 있습니다.

리전이 다시 온라인 상태로 되면 Amazon Keyspaces는 해당 리전에서 보류 중인 쓰기 전파를 재개하여 다른 리전의 복제본 테이블로 전파합니다. 다시 온라인 상태로 된 리전에 대한 다른 복제본 테이블의 쓰기 전파도 재개됩니다.

다중 리전 키스페이스 및 테이블을 만드는 데 필요한 IAM 권한

다중 리전 키스페이스 및 테이블을 성공적으로 생성하려면 IAM 보안 주체가 서비스 연결 역할을 생성할 수 있어야 합니다. 이 서비스 연결 역할은 Amazon Keyspaces에서 사전 정의하는 고유한 유형의 IAM 역할입니다. 여기에는 Amazon Keyspaces가 사용자를 대신하여 작업을 수행하는 데 필요한 모든 권한이 포함되어 있습니다. 서비스 링크 역할에 대한 자세한 내용은 the section called “다중 리전 복제”을(를) 참조하세요.

다중 리전 복제에 필요한 서비스 연결 역할을 생성하려면 IAM 보안 주체에 대한 정책에 다음 요소가 필요합니다.

- `iam:CreateServiceLinkedRole` - 보안 주체가 수행할 수 있는 작업입니다.
- `arn:aws:iam::*:role/aws-service-role/replication.cassandra.amazonaws.com/AWSServiceRoleForKeyspacesReplication` - 작업을 수행할 수 있는 리소스입니다.
- `iam:AWSServiceName": "replication.cassandra.amazonaws.com`— 이 역할을 연결할 수 있는 유일한 AWS 서비스는 Amazon Keyspaces입니다.

다음은 다중 리전 키스페이스 및 테이블을 생성하는 데 필요한 최소 권한을 보안 주체에게 부여하는 정책의 예입니다.

```
{
            "Effect": "Allow",
            "Action": "iam:CreateServiceLinkedRole",
            "Resource": "arn:aws:iam::*:role/aws-service-role/
replication.cassandra.amazonaws.com/AWSServiceRoleForKeyspacesReplication",
            "Condition": {"StringLike": {"iam:AWSServiceName":
 "replication.cassandra.amazonaws.com"}}
}
```

다중 리전 키스페이스 및 테이블에 대한 추가 IAM 권한은 서비스 인증 참조의 Amazon Keyspaces(Apache Cassandra용)의 작업, 리소스 및 조건 키를 참조하세요.

다중 지역 복제 및 point-in-time 복구와 통합 (PITR)

IP oint-in-time 복구는 다중 지역 테이블에서 지원됩니다. PITR을 사용하여 다중 리전 테이블을 성공적으로 복원하려면 다음 조건을 충족해야 합니다.

- 소스 및 대상 테이블을 다중 리전 테이블로 구성해야 합니다.
- 소스 테이블의 키스페이스와 대상 테이블의 키스페이스에 대한 복제 리전은 동일해야 합니다.

소스 테이블을 사용할 수 있는 모든 리전에서 복원 문을 실행할 수 있습니다. Amazon Keyspaces는 각 리전의 대상 테이블을 자동으로 복원합니다. PITR에 대한 자세한 내용은 the section called “작동 방식” 섹션을 참조하세요.

다중 리전 복제 및 AWS 서비스와의 통합

Amazon CloudWatch 지표를 사용하여 서로 다른 AWS 리전 테이블 간의 복제 성능을 모니터링할 수 있습니다. 다음 지표는 다중 리전 키스페이스에 대한 지속적인 모니터링을 제공합니다.

- `ReplicationLatency` - 이 지표는 다중 리전 키스페이스의 한 복제 테이블에서 다른 복제 테이블로 `updates`, `inserts` 또는 `deletes`를 복제하는 데 걸린 시간을 측정합니다.

CloudWatch 지표 모니터링 방법에 대한 자세한 내용은 을 참조하십시오the section called “를 통한 모니터링 CloudWatch”.

Amazon Keyspaces 다중 리전 복제 사용 노트

Amazon Keyspaces와 함께 다중 리전 복제를 사용할 때는 다음 사항을 고려합니다.

- 사용 가능한 퍼블릭 중에서 최대 6개를 선택할 수 AWS 리전있습니다. AWS GovCloud (US) Regions, 기본적으로 비활성화되어 AWS 리전 있는 중국 지역은 지원되지 않습니다.
- 키스페이스의 복제 리전은 나중에 추가하거나 제거할 수 없으므로 신중하게 선택해야 합니다.
- 나중에 새 열을 추가할 수 없으므로 다중 리전 테이블을 만들기 전에 테이블 스키마를 마무리합니다.
- 저장 중 암호화의 경우 AWS 소유 키를 사용하십시오. 다중 리전 테이블에는 고객 관리형 키가 지원되지 않습니다. 자세한 내용을 알아보려면 다음 섹션을 참조하세요.

  the section called “작동 방식”.
- Amazon Keyspaces Auto Scaling과 함께 프로비저닝된 용량 관리를 사용하는 경우 Amazon Keyspaces API 작업을 사용하여 다중 리전 테이블을 생성하고 구성해야 합니다. Amazon Keyspace가 사용자를 대신하여 호출하는 기본 Application Auto Scaling API 작업에는 다중 지역 기능이 없습니다.

  자세한 정보는 the section called “다중 리전 복제 사용 방법”을 참조하세요. 프로비저닝된 다중 리전 테이블의 쓰기 용량 처리량을 추정하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “멀티 리전 테이블”
- 테이블에 TTL(Time To Live)이 필요한지 결정합니다. 나중에 켤 수 없습니다. 자세한 정보는 TTL(Time To Live)을 사용하여 데이터 만료을 참조하세요.
- 다중 리전 테이블의 선택된 리전에 데이터가 자동으로 복제되지만 클라이언트가 한 리전의 엔드포인트에 연결하여 `system.peers` 테이블을 쿼리하면 쿼리는 로컬 정보만 반환합니다. 쿼리 결과는 클라이언트에게 단일 데이터 센터 클러스터처럼 나타납니다.

- Amazon Keyspaces 다중 지역 복제는 비동기식이며 쓰기 일관성을 지원합니다. LOCAL_QUORUM LOCAL_QUORUM일관성을 유지하려면 로컬 리전에 있는 두 개의 복제본에 대한 행 업데이트가 안정적으로 지속되어야 클라이언트에 성공을 반환할 수 있습니다. 그러면 복제된 리전 (또는 리전) 에 대한 쓰기 전파가 비동기적으로 수행됩니다.

  Amazon Keyspaces 다중 지역 복제는 동기 복제 또는 일관성을 지원하지 않습니다. QUORUM
- 다중 리전 키스페이스 또는 테이블을 생성하면 생성 프로세스 중에 정의한 모든 태그가 모든 리전의 모든 키스페이스와 테이블에 자동으로 적용됩니다. ALTER KEYSPACE또는 ALTER TABLE 를 사용하여 기존 태그를 변경하면 변경하려는 지역의 키스페이스 또는 테이블에만 업데이트가 적용됩니다.
- CloudWatch Amazon은 복제된 각 지역에 대한 ReplicationLatency 지표를 제공합니다. 도착하는 행을 추적하고, 도착 시간을 초기 쓰기 시간과 비교하고, 평균을 계산하여 이 측정치를 계산합니다. 타이밍은 소스 지역 CloudWatch 내에 저장됩니다. 자세한 정보는 the section called “를 통한 모니터링 CloudWatch”을 참조하세요.

  평균 및 최대 시간을 확인하여 평균 및 최악의 복제 지연을 파악하는 것이 유용할 수 있습니다. 이 지연 시간에는 SLA가 없습니다.
- 온디맨드 모드에서 멀티 리전 테이블을 사용할 때 테이블 복제본에 새로운 트래픽 피크가 발생하면 비동기 쓰기 복제의 지연 시간이 증가할 수 있습니다. Amazon Keyspaces가 수신되는 애플리케이션 트래픽에 따라 단일 리전 온디맨드 테이블의 용량을 자동으로 조정하는 것과 마찬가지로, Amazon Keyspaces는 다중 리전 온디맨드 테이블 복제본의 용량을 수신하는 트래픽에 맞게 자동으로 조정합니다. Amazon Keyspace는 트래픽 볼륨이 증가함에 따라 더 많은 용량을 자동으로 할당하므로 복제 지연 시간은 일시적입니다. 모든 복제본이 트래픽 볼륨에 맞게 조정되면 복제 지연 시간이 정상으로 돌아올 것입니다. 자세한 정보는 the section called “피크 트래픽 및 크기 조정 속성”을 참조하세요.
- 프로비저닝 모드에서 다중 지역 테이블을 사용할 때 애플리케이션이 프로비저닝된 처리 용량을 초과하면 용량 부족 오류가 발생하고 복제 지연 시간이 증가할 수 있습니다. 모든 멀티 리전 테이블의 모든 테이블 복제본에 대해 항상 충분한 읽기 및 쓰기 용량을 확보하려면 Amazon Keyspaces Auto Scaling을 구성하는 것이 좋습니다. AWS 리전 Amazon Keyspaces Auto Scaling을 사용하면 실제 애플리케이션 트래픽에 따라 처리 용량을 자동으로 조정하여 가변 워크로드에 대한 처리 용량을 효율적으로 프로비저닝할 수 있습니다. 자세한 내용은 the section called “멀티 리전 테이블에서 Auto Scaling이 작동하는 방식”을(를) 참조하세요.

다중 리전 복제 사용 방법

Amazon Keyspaces (Apache Cassandra용) 콘솔, 카산드라 쿼리 언어 (CQL), SDK 및 () 를 사용하여 다중 리전 키스페이스와 테이블을 생성하고 관리할 수 있습니다. AWS AWS Command Line Interface AWS CLI

이 섹션에서는 온디맨드 및 프로비저닝 용량 모드를 모두 사용하여 콘솔, CQL, 를 사용하여 다중 리전 키스페이스와 테이블을 생성하는 방법에 대한 예를 제공합니다. AWS CLI 다중 리전 키스페이스에서 생성된 모든 테이블은 키스페이스의 다중 지역 설정을 자동으로 상속합니다.

이 섹션에는 또한 콘솔, CQL 및 를 사용하여 프로비저닝된 멀티 리전 테이블의 Amazon Keyspaces Auto Scaling 설정을 관리하는 방법에 대한 예제가 포함되어 있습니다. AWS CLI 일반 Auto Scaling 구성 옵션 및 작동 방식에 대한 자세한 내용은 을 참조하십시오the section called “Auto Scaling을 통한 처리 용량 관리”.

다중 리전 테이블에 프로비저닝된 용량 모드를 사용하는 경우 항상 Amazon Keyspaces API 호출을 사용하여 Auto Scaling을 구성해야 합니다. 이는 기본 Application Auto Scaling API 작업이 지역을 인식하지 않기 때문입니다.

프로비전된 다중 리전 테이블의 쓰기 용량 처리량을 추정하는 방법에 대한 자세한 내용은 을 참조하십시오. the section called “멀티 리전 테이블”

아마존 키스페이스 API에 대한 자세한 내용은 아마존 키스페이스 API 레퍼런스를 참조하십시오.

지원되는 구성 및 다중 지역 복제 기능에 대한 자세한 내용은 을 참조하십시오. the section called “사용 노트”

주제

- 콘솔을 사용하여 다중 지역 테이블 생성 및 관리
- CQL을 사용하여 다중 지역 테이블 생성 및 관리
- 를 AWS CLI 사용하여 다중 지역 테이블 생성 및 관리

콘솔을 사용하여 다중 지역 테이블 생성 및 관리

이 섹션에서는 Amazon Keyspaces (Apache Cassandra용) 콘솔을 사용하여 온디맨드 및 프로비저닝 용량 모드에서 다중 리전 키스페이스와 테이블을 생성하는 방법의 예를 제공합니다. 다중 리전 키스페이스에서 생성하는 모든 테이블은 키스페이스의 다중 지역 설정을 자동으로 상속합니다.

CQL 예제는 을 참조하십시오. the section called “CQL 사용” AWS CLI 예제는 을 참조하십시오the section called “사용 AWS CLI”.

주제

- 다중 리전 키스페이스 생성(콘솔)
- 기본 설정으로 멀티 리전 테이블 생성 (콘솔)
- Auto Scaling이 활성화된 상태에서 프로비저닝 모드에서 멀티 리전 테이블 생성 (콘솔)
- 기존 멀티 리전 테이블에 대한 Auto Scaling 활성화 (콘솔)
- 멀티 리전 테이블의 Auto Scaling 끄기 (콘솔)
- 콘솔에서 Amazon Keyspaces 자동 조정 활동 보기

다중 리전 키스페이스 생성(콘솔)

Amazon Keyspaces 콘솔을 사용하여 새 다중 지역 키스페이스를 생성하려면 다음 단계를 따르십시오.

다중 리전 키스페이스를 생성하려면(콘솔)

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 탐색 창에서 Keyspaces를 선택한 다음 키스페이스 생성을 선택합니다.
3. 키스페이스 이름에 키스페이스의 이름을 입력합니다.
4. 다중 리전 복제 섹션에서 목록에서 사용할 수 있는 리전을 최대 5개까지 추가할 수 있습니다.
5. 완료하려면 키스페이스 생성을 선택합니다.

> **Note**
>
> 다중 리전 키스페이스를 생성할 때 Amazon Keyspaces는 계정의 이름 `AWSServiceRoleForAmazonKeyspacesReplication`을 사용하여 서비스 연결 역할을 생성합니다. 이 역할을 통해 Amazon Keyspaces는 사용자를 대신하여 다중 리전 테이블의 모든 복제본에 쓰기를 복제할 수 있습니다. 자세한 내용은 the section called “다중 리전 복제”를 참조하세요.

기본 설정으로 멀티 리전 테이블 생성 (콘솔)

Amazon Keyspaces 콘솔을 사용하여 다중 리전 테이블을 생성하려면 다음 단계를 따릅니다.

다중 리전 테이블을 생성하려면(콘솔)

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 다중 리전 키스페이스를 선택합니다.
3. 테이블 탭에서 테이블 생성을 선택합니다.
4. 테이블 이름에는 테이블의 이름을 입력합니다. 이 테이블이 복제되는 AWS 리전 는 정보 상자에 표시됩니다.
5. 테이블 스키마를 계속 진행합니다.
6. 테이블 설정에서 기본 설정 옵션을 계속 진행합니다. 다중 리전 테이블에 대한 다음 기본 설정을 참고하세요.

   - 용량 모드 - 기본 용량 모드는 온디맨드입니다. 프로비저닝 모드 구성에 대한 자세한 내용은 을 참조하십시오. the section called “Auto Scaling이 활성화된 상태에서 프로비저닝 모드에서 멀티 리전 테이블 생성 (콘솔)”
   - 암호화 키 관리 - AWS 소유 키 옵션만 지원됩니다.
   - 클라이언트 측 타임스탬프 - 이 기능은 다중 리전 테이블에 필요합니다.

   - 테이블 및 모든 복제본에 대해 TTL(Time To Live)을 켜야 하는 경우 설정 사용자 지정을 선택합니다.

     > **Note**
     >
     > 기존 다중 리전 테이블에서는 TTL 설정을 변경할 수 없습니다.

7. 완료하려면 테이블 생성을 선택합니다.

Auto Scaling이 활성화된 상태에서 프로비저닝 모드에서 멀티 리전 테이블 생성 (콘솔)

> **Note**
>
> Amazon Keyspaces Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 설명은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

자동 크기 조정이 활성화된 새 다중 지역 테이블 생성하기

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 다중 리전 키스페이스를 선택합니다.
3. 테이블 탭에서 테이블 생성을 선택합니다.
4. 테이블 세부 정보 섹션의 테이블 생성 페이지에서 키스페이스를 선택하고 새 테이블의 이름을 입력합니다.
5. 열 섹션에서 테이블의 스키마를 생성합니다.
6. 기본 키 섹션에서 테이블의 기본 키를 정의하고 선택적 클러스터링 열을 선택합니다.
7. 테이블 설정 섹션에서 설정 사용자 지정을 선택합니다.
8. 읽기/쓰기 용량 설정을 계속합니다.
9. 용량 모드에서 프로비저닝됨을 선택합니다.
10. 읽기 용량 섹션에서 자동 규모 조정이 선택되어 있는지 확인합니다.

    테이블이 AWS 리전 복제되는 모든 항목에 대해 동일한 읽기 용량 단위를 구성하도록 선택할 수 있습니다. 또는 확인란의 선택을 취소하고 각 지역의 읽기 용량을 다르게 구성할 수도 있습니다.

    각 지역을 다르게 구성하려면 각 테이블 복제본의 최소 및 최대 읽기 용량 단위와 목표 사용률을 선택합니다.

    - 최소 용량 단위 - 테이블이 항상 지원할 준비가 되어 있어야 하는 최소 처리량 수준 값을 입력합니다. 값은 1에서 계정에 대해 초당 최대 처리량 할당량(기본값 40,000) 사이여야 합니다.
    - 최대 용량 단위 - 테이블에 프로비저닝하려는 최대 처리량을 입력합니다. 값은 1에서 계정에 대해 초당 최대 처리량 할당량(기본값 40,000) 사이여야 합니다.
    - 목표 사용률 — 목표 사용률을 20% ~ 90% 사이로 입력합니다. 트래픽이 정의된 목표 사용률을 초과하면 용량이 자동으로 스케일 업됩니다. 트래픽이 정의된 목표 미만으로 떨어지면 자동으로 다시 스케일 다운됩니다.

    - 테이블의 읽기 용량을 수동으로 프로비저닝하려면 Scale auto (자동 조정) 확인란의 선택을 취소하십시오. 이 설정은 테이블의 모든 복제본에 적용됩니다.

> **Note**
>
> 모든 복제본에 충분한 읽기 용량을 확보하려면 프로비저닝된 다중 리전 테이블에 대해 Amazon Keyspaces 자동 크기 조정을 권장합니다.

> **Note**
>
> 계정의 기본 할당량 및 할당량을 높이는 방법에 대한 자세한 내용은 할당량 섹션을 참조하세요.

11. 쓰기 용량 섹션에서 자동 크기 조정이 선택되었는지 확인합니다. 그런 다음 테이블의 용량 단위를 구성합니다. 쓰기 용량 단위는 지역 전체에 걸쳐 쓰기 이벤트를 복제하기에 충분한 용량이 확보되도록 모든 AWS 리전 영역에서 동기화된 상태를 유지합니다.
    - 테이블의 쓰기 용량을 수동으로 프로비저닝하려면 자동 크기 조정을 지우십시오. 이 설정은 테이블의 모든 복제본에 적용됩니다.

> **Note**
>
> 모든 복제본에 충분한 쓰기 용량을 확보하려면 프로비저닝된 멀티 리전 테이블에 대해 Amazon Keyspaces 자동 크기 조정을 권장합니다.

12. 테이블 생성을 선택합니다. Auto Scaling 파라미터로 테이블이 생성됩니다.

기존 멀티 리전 테이블에 대한 Auto Scaling 활성화 (콘솔)

Amazon Keyspaces 콘솔을 사용하여 프로비저닝 모드에서 멀티 리전 테이블에 대한 Auto Scaling을 활성화하려면 다음 단계를 따르십시오.

> **Note**
>
> Amazon Keyspaces Auto Scaling을 사용하려면 사용자 대신 Auto Scaling 작업을 수행하는 서비스 연결 역할(`AWSServiceRoleForApplicationAutoScaling_CassandraTable`)이 있어야 합니다. 이 역할은 자동으로 생성됩니다. 자세한 설명은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

기존 멀티 리전 테이블에 대해 Amazon Keyspaces 자동 크기 조정을 활성화하려면

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 작업하려는 테이블을 선택하고 용량 탭으로 이동합니다.
3. 용량 설정 섹션에서 편집을 선택합니다.
4. 용량 모드에서 테이블이 프로비저닝된 용량 모드를 사용하고 있는지 확인합니다.
5. 자동 크기 조정을 선택하고 9단계를 Auto Scaling이 활성화된 상태에서 프로비저닝 모드에서 멀티 리전 테이블 생성 (콘솔) 참조하여 읽기 및 쓰기 용량을 편집하십시오.
6. Auto Scaling 설정이 정의되면 저장을 선택합니다.

멀티 리전 테이블의 Auto Scaling 끄기 (콘솔)

Amazon Keyspaces 콘솔을 사용하여 프로비저닝 모드의 다중 지역 테이블에 대한 자동 크기 조정을 끄려면 다음 단계를 따르십시오.

기존 멀티 리전 테이블의 Amazon Keyspaces 자동 크기 조정을 끄려면

1. 에 AWS Management Console 로그인하고 https://console.aws.amazon.com/keyspaces/home 에서 Amazon Keyspaces 콘솔을 엽니다.
2. 작업하려는 테이블을 선택하고 용량 탭을 선택합니다.
3. 용량 설정 섹션에서 편집을 선택합니다.
4. Amazon Keyspaces 자동 크기 조정을 비활성화하려면 자동 크기 조정 확인란의 선택을 취소하십시오. 자동 스케일링을 비활성화하면 테이블이 Application Auto Scaling에서 확장 가능한 타겟으로 등록 취소됩니다. Application Auto Scaling에서 Amazon Keyspaces 테이블에 액세스하는 데 사용하는 서비스 연결 역할을 삭제하려면 의 단계를 따르십시오. the section called “Amazon Keyspaces에 대한 서비스 연결 역할 삭제”

   > **Note**
   >
   > Application Auto Scaling에서 사용하는 서비스 연결 역할을 삭제하려면 계정 내 모든 테이블에서 전체 테이블의 자동 크기 조정을 비활성화해야 합니다. AWS 리전

5. Auto Scaling 설정이 정의되면 저장을 선택합니다.

콘솔에서 Amazon Keyspaces 자동 조정 활동 보기

사용량 및 성능에 대한 지표를 생성하는 Amazon을 사용하여 Amazon CloudWatch Keyspaces 자동 크기 조정이 리소스를 어떻게 사용하는지 모니터링할 수 있습니다. Application Auto Scaling 사용 설명서의 단계에 따라 CloudWatch 대시보드를 생성하십시오.

CQL을 사용하여 다중 지역 테이블 생성 및 관리

카산드라 쿼리 언어 (CQL) 를 사용하여 Amazon Keyspace에서 멀티 리전 키스페이스와 테이블을 생성하고 관리할 수 있습니다.

이 섹션에서는 CQL을 사용하여 다중 지역 테이블을 생성하고 관리하는 방법에 대한 예를 제공합니다. 다중 리전 키스페이스에서 생성하는 모든 테이블은 키스페이스의 다중 지역 설정을 자동으로 상속합니다. CQL에 대한 자세한 내용은 Amazon Keyspaces CQL 언어 참조를 참조하십시오.

지원되는 구성 및 기능에 대한 자세한 내용은 the section called “사용 노트” 섹션을 참조하세요.

주제

- 다중 리전 키스페이스 생성(CQL)
- 기본 설정 (CQL) 이 포함된 다중 지역 테이블 만들기
- 프로비저닝된 용량 모드 및 CQL (Auto Scaling) 을 사용하여 멀티 리전 테이블 생성
- 멀티 리전 테이블 (CQL) 의 프로비저닝된 용량 및 Auto Scaling 설정 업데이트
- 멀티 리전 테이블 (CQL) 의 프로비저닝된 용량 및 Auto Scaling 설정 보기
- 다중 지역 테이블 (CQL) 의 자동 크기 조정 끄기
- 다중 지역 테이블의 프로비저닝 용량 수동 설정 (CQL)

다중 리전 키스페이스 생성(CQL)

다중 리전 키스페이스를 생성하려면 AWS 리전 키스페이스가 복제될 `NetworkTopologyStrategy` 키스페이스를 지정하는 데 사용하십시오. 현재 리전과 하나 이상의 추가 리전을 포함해야 합니다. 다음 CQL 문은 이에 대한 예입니다.

```
CREATE KEYSPACE mykeyspace
WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'us-east-1':'3', 'ap-
southeast-1':'3','eu-west-1':'3' };
```

키스페이스의 모든 테이블은 키스페이스와 동일한 복제 전략을 사용합니다. 테이블 수준에서는 복제 전략을 변경할 수 없습니다.

NetworkTopologyStrategy— Amazon Keyspace는 기본적으로 동일한 AWS 리전 가용 영역 내 세 개의 가용 영역에 데이터를 복제하기 때문에 각 지역의 복제 인자는 3입니다.

> **Note**
>
> 다중 리전 키스페이스를 생성할 때 Amazon Keyspaces는 계정의 이름 AWSServiceRoleForAmazonKeyspacesReplication을 사용하여 서비스 연결 역할을 생성합니다. 이 역할을 통해 Amazon Keyspaces는 사용자를 대신하여 다중 리전 테이블의 모든 복제본에 쓰기를 복제할 수 있습니다. 자세한 내용은 the section called “다중 리전 복제”를 참조하세요.

CQL 문을 사용하여 system_multiregion_info 키스페이스의 tables 테이블을 쿼리하여 지정한 다중 지역 테이블의 지역과 상태를 프로그래밍 방식으로 나열할 수 있습니다. 다음 코드는 이에 대한 예입니다.

```
SELECT * from system_multiregion_info.tables WHERE keyspace_name = 'mykeyspace' AND
 table_name = 'mytable';
```

명령문의 출력은 다음과 같습니다.

```
 keyspace_name  | table_name     | region         | status
----------------+----------------+----------------+--------
 mykeyspace     | mytable        | us-east-1      | ACTIVE
 mykeyspace     | mytable        | ap-southeast-1 | ACTIVE
 mykeyspace     | mytable        | eu-west-1      | ACTIVE
```

기본 설정 (CQL) 이 포함된 다중 지역 테이블 만들기

기본 설정으로 다중 지역 테이블을 만들려면 다음 예제를 사용할 수 있습니다.

```
CREATE TABLE mykeyspace.mytable(pk int, ck int, PRIMARY KEY (pk, ck))
    WITH CUSTOM_PROPERTIES = {
 'capacity_mode':{
  'throughput_mode':'PAY_PER_REQUEST'
 },
 'point_in_time_recovery':{
  'status':'enabled'
 },
 'encryption_specification':{
```

```
  'encryption_type':'AWS_OWNED_KMS_KEY'
 },
 'client_side_timestamps':{
  'status':'enabled'
 }
};
```

프로비저닝된 용량 모드 및 CQL (Auto Scaling) 을 사용하여 멀티 리전 테이블 생성

Auto Scaling을 사용하여 프로비저닝 모드에서 멀티 리전 테이블을 생성하려면 먼저 테이블에 CUSTOM_PROPERTIES 대한 정의를 통해 용량 모드를 지정해야 합니다. 프로비저닝된 용량 모드를 지정한 후 를 사용하여 테이블에 대한 Auto Scaling 설정을 구성할 수 있습니다. AUTOSCALING_SETTINGS

Auto Scaling 설정, 대상 추적 정책, 대상 값 및 선택적 설정에 대한 자세한 내용은 을 참조하십시오the section called “CQL을 사용하여 자동 크기 조정이 가능한 새 테이블을 생성합니다.”.

다중 지역 테이블을 생성할 때 테이블의 각 복제본에 대해 서로 다른 읽기 용량과 읽기 Auto Scaling 설정을 지정할 수도 있습니다. 지정한 설정이 지정된 항목에 대한 테이블의 일반 설정을 덮어씁니다. AWS 리전 하지만 쓰기 용량은 모든 지역에서 쓰기를 복제하기에 충분한 용량을 확보할 수 있도록 모든 복제본 간에 동기화된 상태로 유지됩니다.

특정 지역의 테이블 복제본에 대한 읽기 용량을 정의하려면 다음 파라미터를 테이블의 일부로 구성할 수 있습니다. replica_updates

- 리전
- 프로비저닝된 읽기 용량 단위 (선택 사항)
- 읽기 용량 자동 조정 설정 (선택 사항)

다음 예제는 프로비저닝 모드의 다중 지역 테이블에 대한 CREATE TABLE 명령문을 보여줍니다. 일반적인 쓰기 및 읽기 용량 Auto Scaling 설정은 동일합니다. 하지만 읽기 Auto Scaling 설정에서는 테이블의 읽기 용량을 늘리거나 줄이기 전에 60초의 추가 휴지 기간을 지정합니다. 또한 미국 동부 (버지니아 북부) 지역의 읽기 용량 Auto Scaling 설정은 다른 복제본의 읽기 용량 Auto Scaling 설정보다 높습니다. 또한 목표값은 50% 가 아닌 70% 로 설정되어 있습니다.

```
CREATE TABLE mykeyspace.mytable(pk int, ck int, PRIMARY KEY (pk, ck))
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 5,
```

```
        'write_capacity_units': 5
    }
} AND AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50
            }
        }
    },
    'provisioned_read_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50,
                'scale_in_cooldown': 60,
                'scale_out_cooldown': 60
            }
        }
    },
    'replica_updates': {
        'us-east-1': {
            'provisioned_read_capacity_autoscaling_update': {
                'maximum_units': 20,
                'minimum_units': 5,
                'scaling_policy': {
                    'target_tracking_scaling_policy_configuration': {
                        'target_value': 70
                    }
                }
            }
        }
    }
};
```

멀티 리전 테이블 (CQL) 의 프로비저닝된 용량 및 Auto Scaling 설정 업데이트

를 사용하여 ALTER TABLE 기존 테이블의 용량 모드 및 Auto Scaling 설정을 업데이트할 수 있습니다. 현재 온디맨드 용량 모드인 테이블을 업데이트하려는 경우에는 capacity_mode 필수입니다. 테이블이 이미 프로비저닝된 용량 모드에 있는 경우 이 필드를 생략할 수 있습니다.

Auto Scaling 설정, 대상 추적 정책, 대상 값 및 선택적 설정에 대한 자세한 내용은 을 참조하십시오the section called “CQL을 사용하여 자동 크기 조정이 가능한 새 테이블을 생성합니다.”.

마찬가지로 테이블의 `replica_updates` 속성을 업데이트하여 특정 리전에 있는 테이블 복제본의 읽기 용량 및 Auto Scaling 설정을 업데이트할 수도 있습니다. 다음 문은 이에 대한 예입니다.

```
ALTER TABLE mykeyspace.mytable
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 1,
        'write_capacity_units': 1
    }
} AND AUTOSCALING_SETTINGS = {
    'provisioned_write_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50
            }
        }
    },
    'provisioned_read_capacity_autoscaling_update': {
        'maximum_units': 10,
        'minimum_units': 5,
        'scaling_policy': {
            'target_tracking_scaling_policy_configuration': {
                'target_value': 50,
                'scale_in_cooldown': 60,
                'scale_out_cooldown': 60
            }
        }
    },
    'replica_updates': {
        'us-east-1': {
            'provisioned_read_capacity_autoscaling_update': {
                'maximum_units': 20,
                'minimum_units': 5,
                'scaling_policy': {
                    'target_tracking_scaling_policy_configuration': {
                        'target_value': 70
                    }
```

```
                }
            }
        }
    }
};
```

멀티 리전 테이블 (CQL) 의 프로비저닝된 용량 및 Auto Scaling 설정 보기

다중 지역 테이블의 Auto Scaling 구성을 보려면 다음 명령을 사용하십시오.

```
SELECT * FROM system_multiregion_info.autoscaling WHERE keyspace_name = 'mykeyspace'
 AND table_name = 'mytable';
```

이 명령의 출력은 다음과 같습니다.

```
 keyspace_name  | table_name | region         |
 provisioned_read_capacity_autoscaling_update

                                     | provisioned_write_capacity_autoscaling_update
---------------+------------+---------------
+-----------------------------------------------------------------------------------------------------
+-----------------------------------------------------------------------------------------------------
  mykeyspace    |  mytable   | ap-southeast-1 | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 60, 'disable_scale_in': false, 'target_value':
 50, 'scale_in_cooldown': 60}}} | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 0, 'disable_scale_in': false, 'target_value': 50,
 'scale_in_cooldown': 0}}}
  mykeyspace    |  mytable   | us-east-1      | {'minimum_units': 5, 'maximum_units':
 20, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 60, 'disable_scale_in': false, 'target_value':
 70, 'scale_in_cooldown': 60}}} | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 0, 'disable_scale_in': false, 'target_value': 50,
 'scale_in_cooldown': 0}}}
  mykeyspace    |  mytable   | eu-west-1      | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
 {'scale_out_cooldown': 60, 'disable_scale_in': false, 'target_value':
 50, 'scale_in_cooldown': 60}}} | {'minimum_units': 5, 'maximum_units':
 10, 'scaling_policy': {'target_tracking_scaling_policy_configuration':
```

```
 {'scale_out_cooldown': 0, 'disable_scale_in': false, 'target_value': 50,
 'scale_in_cooldown': 0}}}
```

다중 지역 테이블 (CQL) 의 자동 크기 조정 끄기

를 사용하여 ALTER TABLE 기존 테이블의 Auto Scaling을 끌 수 있습니다. 참고로 개별 테이블 복제본에 대해서는 Auto Scaling을 끌 수 없습니다.

다음 예시에서는 테이블의 읽기 용량에 대해 Auto Scaling이 꺼져 있습니다.

```
ALTER TABLE mykeyspace.mytable
WITH AUTOSCALING_SETTINGS = {
    'provisioned_read_capacity_autoscaling_update': {
        'autoscaling_disabled': true
    }
};
```

> **Note**
>
> Application Auto Scaling에서 사용하는 서비스 연결 역할을 삭제하려면 모든 AWS 리전에 걸쳐 계정 내 모든 테이블에서 Auto Scaling을 비활성화해야 합니다.

다중 지역 테이블의 프로비저닝 용량 수동 설정 (CQL)

다중 지역 테이블에 대한 Auto Scaling을 해제해야 ALTER TABLE 하는 경우, 를 사용하여 복제 테이블에 대한 테이블의 읽기 용량을 수동으로 프로비저닝할 수 있습니다.

```
ALTER TABLE mykeyspace.mytable
WITH CUSTOM_PROPERTIES = {
    'capacity_mode': {
        'throughput_mode': 'PROVISIONED',
        'read_capacity_units': 1,
        'write_capacity_units': 1
    },
    'replica_updates': {
        'us-east-1': {
            'read_capacity_units': 2
         }
    }
```

```
};
```

> **Note**
>
> 프로비저닝된 용량을 사용하는 멀티 리전 테이블에는 Auto Scaling을 사용하는 것이 좋습니다. 자세한 내용은 the section called “멀티 리전 테이블”을(를) 참조하세요.

를 AWS CLI 사용하여 다중 지역 테이블 생성 및 관리

AWS Command Line Interface (AWS CLI) 를 사용하여 Amazon Keyspace에서 다중 지역 키스페이스와 테이블을 생성하고 관리할 수 있습니다.

이 섹션에서는 를 사용하여 다중 지역 테이블을 생성하고 관리하는 방법에 대한 예를 제공합니다. AWS CLI 다중 리전 키스페이스에서 생성하는 모든 테이블은 키스페이스의 다중 지역 설정을 자동으로 상속합니다.

이 주제에 설명된 Amazon Keyspaces AWS CLI 명령에 대한 자세한 내용은 Amazon Keyspaces의 AWS CLI 명령 참조를 참조하십시오.

주제

- 새 다중 리전 키스페이스 생성(CLI)
- 기본 설정이 포함된 새 다중 지역 테이블 생성 (CLI)
- Auto Scaling (CLI) 을 사용하여 프로비저닝 모드에서 새 멀티 리전 테이블 생성
- 다중 지역 테이블 (CLI) 의 프로비저닝된 용량 및 Auto Scaling 설정 업데이트
- 멀티 리전 테이블 (CLI) 의 프로비저닝된 용량 및 Auto Scaling 설정 보기
- 멀티 리전 테이블 (CLI) 의 자동 스케일링 끄기
- 다중 지역 테이블의 프로비저닝 용량을 수동으로 설정 (CLI)

새 다중 리전 키스페이스 생성(CLI)

다중 리전 키스페이스를 생성하려면 다음 CLI 문을 사용합니다. 현재 리전과 하나 이상의 추가 리전을 `regionList`에 지정합니다.

```
aws keyspaces create-keyspace --keyspace-name mykeyspace
            \ --replication-specification
 replicationStrategy=MULTI_REGION,regionList=us-east-1,eu-west-1
```

> **Note**
>
> 다중 리전 키스페이스를 생성할 때 Amazon Keyspaces는 계정의 이름 `AWSServiceRoleForAmazonKeyspacesReplication`을 사용하여 서비스 연결 역할을 생성합니다. 이 역할을 통해 Amazon Keyspaces는 사용자를 대신하여 다중 리전 테이블의 모든 복제본에 쓰기를 복제할 수 있습니다. 자세한 내용은 the section called “다중 리전 복제”를 참조하세요.

기본 설정이 포함된 새 다중 지역 테이블 생성 (CLI)

기본 설정을 사용하여 다중 지역 테이블을 만들려면 스키마만 지정하면 됩니다. 다음 예제를 사용할 수 있습니다.

```
aws keyspaces create-table --keyspace-name mykeyspace --table-name mytable
            \ --schema-definition 'allColumns=[{name=pk,type=int}],partitionKeys={name=
 pk}'
```

명령의 출력은 다음과 같습니다.

```
{
    "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/
table/mytable"
}
```

테이블 설정을 확인하려면 다음 명령문을 사용할 수 있습니다.

```
aws keyspaces get-table --keyspace-name mykeyspace --table-name mytable
```

출력에는 다중 지역 테이블의 모든 기본 설정이 표시됩니다.

```
{
    "keyspaceName": "mykeyspace",
    "tableName": "mytable",
    "resourceArn": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/
table/mytable",
    "creationTimestamp": "2023-12-19T16:50:37.639000+00:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
```

```
            {
                "name": "pk",
                "type": "int"
            }
        ],
        "partitionKeys": [
            {
                "name": "pk"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2023-12-19T16:50:37.639000+00:00"
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    },
    "clientSideTimestamps": {
        "status": "ENABLED"
    },
    "replicaSpecifications": [
        {
            "region": "us-east-1",
            "status": "ACTIVE",
            "capacitySpecification": {
                "throughputMode": "PAY_PER_REQUEST",
                "lastUpdateToPayPerRequestTimestamp": 1702895811.469
            }
        },
        {
            "region": "eu-north-1",
            "status": "ACTIVE",
            "capacitySpecification": {
                "throughputMode": "PAY_PER_REQUEST",
```

```
                "lastUpdateToPayPerRequestTimestamp": 1702895811.121
            }
        }
    ]
}
```

Auto Scaling (CLI) 을 사용하여 프로비저닝 모드에서 새 멀티 리전 테이블 생성

Auto Scaling 구성을 사용하여 프로비저닝 모드에서 다중 지역 테이블을 생성하려면 를 사용할 수 있습니다. AWS CLI 멀티 리전 자동 스케일링 설정을 구성하려면 Amazon Keyspaces CLI `create-table` 명령을 사용해야 한다는 점에 유의하십시오. Amazon Keyspace가 사용자를 대신하여 자동 크기 조정을 수행하는 데 사용하는 서비스인 Application Auto Scaling이 여러 지역을 지원하지 않기 때문입니다.

Auto Scaling 설정, 대상 추적 정책, 대상 값 및 선택적 설정에 대한 자세한 내용은 을 참조하십시오the section called “를 사용하여 자동 크기 조정이 포함된 새 테이블을 생성하십시오. AWS CLI”.

Auto Scaling 설정을 사용하여 프로비저닝 모드에서 새 멀티 리전 테이블을 생성할 때 테이블이 AWS 리전 복제되는 모든 항목에 유효한 테이블의 일반 설정을 지정할 수 있습니다. 그런 다음 각 복제본의 읽기 용량 설정과 읽기 Auto Scaling 설정을 덮어쓸 수 있습니다. 그러나 쓰기 용량은 모든 복제본 간에 동기화된 상태로 유지되므로 모든 지역에서 쓰기를 복제하기에 충분한 용량이 확보됩니다.

특정 지역의 테이블 복제본에 대한 읽기 용량을 정의하려면 다음 파라미터를 테이블의 일부로 구성할 수 있습니다. `replicaSpecifications`

- 리전
- 프로비저닝된 읽기 용량 단위 (선택 사항)
- 읽기 용량 자동 조정 설정 (선택 사항)

복잡한 Auto Scaling 설정과 테이블 복제본의 구성이 다른 프로비저닝된 멀티 리전 테이블을 생성할 때는 JSON 파일에서 테이블의 Auto Scaling 설정과 복제본 구성을 로드하는 것이 좋습니다.

다음 코드 예제를 사용하려면 auto-scaling.zip 에서 예제 JSON 파일을 다운로드하고 및 를 `auto-scaling.json` 추출하면 됩니다. `replication.json` 파일 경로를 기록해 두십시오.

이 예시에서는 JSON 파일이 현재 디렉터리에 있습니다. 다양한 파일 경로 옵션에 대해서는 파일에서 매개변수를 로드하는 방법을 참조하세요.

```
aws keyspaces create-table --keyspace-name mykeyspace --table-name mytable
```

```
          \ --schema-definition 'allColumns=[{name=pk,type=int},
{name=ck,type=int}],partitionKeys=[{name=pk},{name=ck}]'
          \ --capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1
          \ --auto-scaling-specification file://auto-scaling.json
          \ --replica-specifications file://replication.json
```

다중 지역 테이블 (CLI) 의 프로비저닝된 용량 및 Auto Scaling 설정 업데이트

기존 테이블의 프로비저닝 모드 및 Auto Scaling 구성을 업데이트하려면 명령을 사용할 수 있습니다. AWS CLI `update-table`

참고로, Amazon Keyspaces CLI 명령을 사용하여 멀티 리전 자동 스케일링 설정을 생성하거나 수정해야 합니다. Amazon Keyspace가 사용자를 대신하여 테이블 용량의 자동 크기 조정을 수행하는 데 사용하는 서비스인 Application Auto Scaling이 여러 개를 지원하지 않기 때문입니다. AWS 리전

멀티 리전 테이블의 프로비저닝 모드 또는 Auto Scaling 설정을 업데이트할 때 테이블의 각 복제본에 대한 읽기 용량 설정 및 읽기 Auto Scaling 구성을 업데이트할 수 있습니다.

하지만 쓰기 용량은 모든 리전에 걸쳐 쓰기를 복제하기에 충분한 용량이 확보되도록 모든 복제본 간에 동기화된 상태로 유지됩니다. 특정 지역의 테이블 복제본에 대한 읽기 용량을 업데이트하려면 테이블의 다음 선택적 파라미터 중 하나를 변경할 수 있습니다. `replicaSpecifications`

- 프로비저닝된 읽기 용량 단위 (선택 사항)
- 읽기 용량 자동 조정 설정 (선택 사항)

복잡한 Auto Scaling 설정과 테이블 복제본에 대한 다양한 구성을 사용하여 다중 지역 테이블을 업데이트할 때는 JSON 파일에서 테이블의 Auto Scaling 설정과 복제본 구성을 로드하는 것이 좋습니다.

다음 코드 예제를 사용하려면 auto-scaling.zip 에서 예제 JSON 파일을 다운로드하고 및 를 `auto-scaling.json` 추출하면 됩니다. `replication.json` 파일 경로를 기록해 두십시오.

이 예시에서는 JSON 파일이 현재 디렉터리에 있습니다. 다양한 파일 경로 옵션에 대해서는 파일에서 매개변수를 로드하는 방법을 참조하세요.

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
          \ --capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1
          \ --auto-scaling-specification file://auto-scaling.json
          \ --replica-specifications file://replication.json
```

멀티 리전 테이블 (CLI) 의 프로비저닝된 용량 및 Auto Scaling 설정 보기

다중 지역 테이블의 Auto Scaling 구성을 보려면 `get-table-auto-scaling-settings` 작업을 사용할 수 있습니다. 다음 CLI 명령은 이에 대한 예입니다.

```
aws keyspaces get-table-auto-scaling-settings --keyspace-name mykeyspace --table-name
 mytable
```

다음과 같이 출력되어야 합니다.

```
{
    "keyspaceName": "mykeyspace",
    "tableName": "mytable",
    "resourceArn": "arn:aws:cassandra:us-east-1:777788889999:/keyspace/mykeyspace/
table/mytable",
    "autoScalingSpecification": {
        "writeCapacityAutoScaling": {
            "autoScalingDisabled": false,
            "minimumUnits": 5,
            "maximumUnits": 10,
            "scalingPolicy": {
                "targetTrackingScalingPolicyConfiguration": {
                    "disableScaleIn": false,
                    "scaleInCooldown": 0,
                    "scaleOutCooldown": 0,
                    "targetValue": 50.0
                }
            }
        },
        "readCapacityAutoScaling": {
            "autoScalingDisabled": false,
            "minimumUnits": 5,
            "maximumUnits": 20,
            "scalingPolicy": {
                "targetTrackingScalingPolicyConfiguration": {
                    "disableScaleIn": false,
                    "scaleInCooldown": 60,
                    "scaleOutCooldown": 60,
                    "targetValue": 70.0
                }
            }
        }
    },
```

```
    "replicaSpecifications": [
        {
            "region": "us-east-1",
            "autoScalingSpecification": {
                "writeCapacityAutoScaling": {
                    "autoScalingDisabled": false,
                    "minimumUnits": 5,
                    "maximumUnits": 10,
                    "scalingPolicy": {
                        "targetTrackingScalingPolicyConfiguration": {
                            "disableScaleIn": false,
                            "scaleInCooldown": 0,
                            "scaleOutCooldown": 0,
                            "targetValue": 50.0
                        }
                    }
                },
                "readCapacityAutoScaling": {
                    "autoScalingDisabled": false,
                    "minimumUnits": 5,
                    "maximumUnits": 20,
                    "scalingPolicy": {
                        "targetTrackingScalingPolicyConfiguration": {
                            "disableScaleIn": false,
                            "scaleInCooldown": 60,
                            "scaleOutCooldown": 60,
                            "targetValue": 70.0
                        }
                    }
                }
            }
        },
        {
            "region": "eu-north-1",
            "autoScalingSpecification": {
                "writeCapacityAutoScaling": {
                    "autoScalingDisabled": false,
                    "minimumUnits": 5,
                    "maximumUnits": 10,
                    "scalingPolicy": {
                        "targetTrackingScalingPolicyConfiguration": {
                            "disableScaleIn": false,
                            "scaleInCooldown": 0,
                            "scaleOutCooldown": 0,
```

```
                        "targetValue": 50.0
                    }
                }
            },
            "readCapacityAutoScaling": {
                "autoScalingDisabled": false,
                "minimumUnits": 5,
                "maximumUnits": 10,
                "scalingPolicy": {
                    "targetTrackingScalingPolicyConfiguration": {
                        "disableScaleIn": false,
                        "scaleInCooldown": 60,
                        "scaleOutCooldown": 60,
                        "targetValue": 50.0
                    }
                }
            }
        }
    ]
}
```

멀티 리전 테이블 (CLI) 의 자동 스케일링 끄기

AWS CLI `update-table`명령을 사용하여 기존 테이블의 Auto Scaling을 끌 수 있습니다. 개별 테이블 복제본에 대해서는 Auto Scaling을 끌 수 없다는 점에 유의하세요.

다음 예시에서는 테이블의 읽기 용량에 대해 Auto Scaling이 꺼져 있습니다.

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
           \ --auto-scaling-specification
 readCapacityAutoScaling={autoScalingDisabled=true}
```

> **Note**
>
> Application Auto Scaling에서 사용하는 서비스 연결 역할을 삭제하려면 계정 내 모든 테이블에서 전체 테이블의 자동 크기 조정을 비활성화해야 합니다. AWS 리전

다중 지역 테이블의 프로비저닝 용량을 수동으로 설정 (CLI)

다중 지역 테이블에 대한 Auto Scaling을 해제해야 `update-table` 하는 경우, 를 사용하여 복제 테이블에 대한 테이블의 읽기 용량을 수동으로 프로비저닝할 수 있습니다.

```
aws keyspaces update-table --keyspace-name mykeyspace --table-name mytable
           \ --capacity-specification
 throughputMode=PROVISIONED,readCapacityUnits=1,writeCapacityUnits=1
           \ --replica-specifications region="us-east-1",readCapacityUnits=5
```

> **Note**
>
> 프로비저닝된 용량을 사용하는 멀티 리전 테이블에는 Auto Scaling을 사용하는 것이 좋습니다. 자세한 내용은 the section called “멀티 리전 테이블”을(를) 참조하세요.

Amazon Keyspaces(Apache Cassandra용)에 대한 시점 복구

시점 복구(PITR)를 사용하면 테이블 데이터를 지속적으로 백업하여 우발적인 쓰기 또는 삭제 작업으로부터 Amazon Keyspaces 테이블을 보호할 수 있습니다.

예를 들어 테스트 스크립트가 프로덕션 Amazon Keyspaces 테이블에 우발적으로 데이터를 썼다고 가정합니다. 시점 복구를 사용하면 최근 35일 이내에 PITR이 활성화된 이후 원하는 시점으로 테이블 데이터를 복원할 수 있습니다. 시점 복구가 활성화된 상태에서 테이블을 삭제하면 35일 동안(추가 비용 없음) 삭제된 테이블의 데이터를 쿼리하고 삭제 시점 바로 이전 상태로 복원할 수 있습니다.

콘솔, AWS SDK 및 AWS Command Line Interface(AWS CLI) 또는 Cassandra 쿼리 언어(CQL)를 사용하여 Amazon Keyspaces 테이블을 특정 시점으로 복원할 수 있습니다. 자세한 내용은 Amazon Keyspaces 테이블을 특정 시점으로 복원 섹션을 참조하세요.

특정 시점 작업은 기본 테이블에 성능이나 가용성에 영향을 주지 않으며 테이블을 복원해도 처리량이 추가로 소모되지 않습니다.

PITR 할당량에 대한 자세한 내용은 할당량 섹션을 참조하세요.

요금에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 요금을 참조하세요.

주제

- Amazon Keyspaces에서 point-in-time 복구가 작동하는 방식
- Amazon Keyspaces 테이블을 특정 시점으로 복원

Amazon Keyspaces에서 point-in-time 복구가 작동하는 방식

이 섹션에서는 Amazon Keyspaces point-in-time 복구 (PITR) 작동 방식에 대한 개요를 제공합니다. 요금에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 요금을 참조하세요.

주제

- point-in-time 복구 활성화 (PITR)
- 테이블 복원에 필요한 권한
- PITR 연속 백업 기간
- PITR 복원 설정
- 암호화된 테이블의 PITR 복원
- 다중 리전 테이블의 PITR 복원

- PITR을 사용한 테이블 복원 시간
- Amazon Keyspaces PITR 및 AWS 서비스와 통합

point-in-time 복구 활성화 (PITR)

콘솔을 사용하여 PITR을 사용하거나 프로그래밍 방식으로 활성화할 수 있습니다.

콘솔로 PITR 활성화

새 테이블의 PITR 설정은 사용자 지정 설정 옵션에서 관리할 수 있습니다. 기본적으로 PITR은 콘솔을 통해 만든 새 테이블에서 활성화됩니다.

기존 테이블에 PITR을 사용하려면 다음 단계를 완료합니다.

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 테이블을 선택하고 편집하려는 테이블을 선택합니다.
3. 백업 탭에서 편집을 선택합니다.
4. point-in-time 복구 설정 편집 섹션에서 P oint-in-time 복구 활성화를 선택합니다.

다음 단계에 따라 언제든지 테이블에서 PITR을 비활성화할 수 있습니다.

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 테이블을 선택하고 편집하려는 테이블을 선택합니다.
3. 백업 탭에서 편집을 선택합니다.
4. point-in-time 복구 설정 편집 섹션에서 P oint-in-time 복구 활성화 확인란의 선택을 취소합니다.

> **Important**
>
> PITR을 비활성화하면 35일 이내에 테이블에서 PITR을 다시 활성화하더라도 백업 기록이 즉시 삭제됩니다.

콘솔을 사용하여 테이블을 복원하는 방법은 the section called “테이블을 특정 시점으로 복원(콘솔)” 섹션을 참조하세요.

AWS CLI를 사용하여 PITR 활성화

`UpdateTable` API를 사용하여 테이블의 PITR 설정을 관리할 수 있습니다.

AWS CLI를 사용하여 새 테이블을 생성하는 경우 새 테이블을 생성할 때 PITR을 명시적으로 활성화해야 합니다.

새 테이블을 생성할 때 PITR을 활성화하려면 다음 AWS CLI 명령을 예로 사용할 수 있습니다. 가독성을 높이기 위해 명령을 별도의 줄로 나누었습니다.

```
aws keyspaces create-table --keyspace-name 'myKeyspace' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},{name=name,type=text},
{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --point-in-time-recovery 'status=ENABLED'
```

> **Note**
>
> point-in-time 복구 값을 지정하지 않은 경우 기본적으로 point-in-time 복구가 비활성화됩니다.

다음 AWS CLI 명령을 사용하여 테이블의 point-in-time 복구 설정을 확인할 수 있습니다.

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

AWS CLI를 사용하여 기존 테이블에 대해 PITR을 활성화하려면 다음 명령을 실행합니다.

```
aws keyspaces update-table --keyspace-name 'myKeyspace' --table-name 'myTable' --point-
in-time-recovery 'status=ENABLED'
```

기존 테이블에서 PITR을 비활성화하려면 다음 AWS CLI 명령을 실행합니다.

```
aws keyspaces update-table --keyspace-name 'myKeyspace' --table-name 'myTable' --point-
in-time-recovery 'status=DISABLED'
```

> **Important**
>
> PITR을 비활성화하면 35일 이내에 테이블에서 PITR을 다시 활성화하더라도 백업 기록이 즉시 삭제됩니다.

CQL을 사용하여 PITR 활성화

`point_in_time_recovery` 사용자 지정 속성을 사용하여 테이블의 PITR 설정을 관리할 수 있습니다.

CQL을 사용하여 새 테이블을 생성하는 경우 새 테이블을 생성할 때 PITR을 명시적으로 활성화해야 합니다.

새 테이블을 생성할 때 PITR을 활성화하려면 다음 CQL 명령을 예로 사용할 수 있습니다.

```
CREATE TABLE "my_keyspace1"."my_table1"(
 "id" int,
 "name" ascii,
 "date" timestamp,
 PRIMARY KEY("id"))
WITH CUSTOM_PROPERTIES = {
 'capacity_mode':{'throughput_mode':'PAY_PER_REQUEST'},
 'point_in_time_recovery':{'status':'enabled'}
}
```

> **Note**
>
> point-in-time 복구 사용자 지정 속성을 지정하지 않은 경우 기본적으로 point-in-time 복구가 비활성화됩니다.

CQL을 사용하여 기존 테이블에 대해 PITR을 활성화하려면 다음 CQL 명령을 실행합니다.

```
ALTER TABLE mykeyspace.mytable
WITH custom_properties = {'point_in_time_recovery': {'status': 'enabled'}}
```

기존 테이블에서 PITR을 비활성화하려면 다음 CQL 명령을 실행합니다.

```
ALTER TABLE mykeyspace.mytable
WITH custom_properties = {'point_in_time_recovery': {'status': 'disabled'}}
```

> **Important**
>
> PITR을 비활성화하면 35일 이내에 테이블에서 PITR을 다시 활성화하더라도 백업 기록이 즉시 삭제됩니다.

CQL 언어 참조에 대한 자세한 내용은 the section called “CREATE TABLE” 및 the section called “ALTER TABLE” 섹션을 참조하세요. CQL을 사용하여 테이블을 복원하는 방법은 the section called “테이블을 CQL을 사용하여 특정 시점으로 복원” 섹션을 참조하세요.

테이블 복원에 필요한 권한

테이블을 성공적으로 복원하려면 IAM 사용자 또는 역할에 다음과 같은 최소 권한이 필요합니다.

- `cassandra:Restore` - 대상 테이블을 복원하려면 복원 작업이 필요합니다.
- `cassandra:Select` - 소스 테이블에서 읽으려면 선택 작업이 필요합니다.
- `cassandra:TagResource` - 태그 작업은 선택 사항이며 복원 작업에서 태그를 추가하는 경우에만 필요합니다.

다음은 keyspace `mykeyspace`에서 테이블을 복원하는 데 필요한 최소 권한을 사용자에게 부여하는 정책의 예입니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Restore",
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/*",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

선택한 다른 기능에 따라 테이블을 복원하기 위한 추가 권한이 필요할 수 있습니다. 예를 들어 소스 테이블을 고객 관리형 키로 암호화한 경우 Amazon Keyspaces가 소스 테이블의 고객 관리형 키에 액세스할 수 있는 권한을 가져야 테이블을 성공적으로 복원할 수 있습니다. 자세한 설명은 the section called “PITR 및 암호화된 테이블” 섹션을 참조하세요.

IAM 정책을 조건 키와 함께 사용하여 들어오는 트래픽을 특정 소스로 제한하는 경우 Amazon Keyspaces가 보안 주체를 대신하여 복원 작업을 수행할 권한이 있는지 확인해야 합니다. 정책이 수신

트래픽을 다음 중 하나로 제한하는 경우 IAM 정책에 `aws:ViaAWSService` 조건 키를 추가해야 합니다.

- `aws:SourceVpce`의 VPC 엔드포인트
- `aws:SourceIp`의 IP 범위
- `aws:SourceVpc`의 VPC

`aws:ViaAWSService` 조건 키는 AWS 서비스가 주체의 보안 인증을 사용하여 요청할 때 액세스를 허용합니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건 키를 참조하세요.

다음은 소스 트래픽을 특정 IP 주소로 제한하고 Amazon Keyspaces가 보안 주체를 대신하여 테이블을 복원하도록 허용하는 정책의 예입니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CassandraAccessForCustomIp",
         "Effect":"Allow",
         "Action":"cassandra:*",
         "Resource":"*",
         "Condition":{
            "Bool":{
               "aws:ViaAWSService":"false"
            },
            "ForAnyValue:IpAddress":{
               "aws:SourceIp":[
                  "123.45.167.89"
               ]
            }
         }
      },
      {
         "Sid":"CassandraAccessForAwsService",
         "Effect":"Allow",
         "Action":"cassandra:*",
         "Resource":"*",
         "Condition":{
            "Bool":{
               "aws:ViaAWSService":"true"
            }
```

```
            }
        }
    ]
}
```

`aws:ViaAWSService` 글로벌 조건 키를 사용하는 예제 정책은 the section called “VPC 엔드포인트 정책 및 아마존 키스페이스 point-in-time 복구 (PITR)” 섹션을 참조하세요.

PITR 연속 백업 기간

Amazon Keyspaces PITR은 두 개의 타임스탬프를 사용하여 테이블에 대해 복원 가능한 백업을 사용할 수 있는 기간을 유지합니다.

- 가장 빠른 복원 가능 시간 - 복원 가능한 가장 빠른 백업 시간을 표시합니다. 복원 가능한 가장 빠른 백업은 최대 35일 또는 PITR이 활성화된 날짜 중 더 최근 날짜로 거슬러 올라갑니다. 최대 백업 기간인 35일은 수정할 수 없습니다.
- 현재 시간 - 복원 가능한 최신 백업의 타임스탬프는 현재 시간입니다. 복원 중에 타임스탬프가 제공되지 않은 경우 현재 시간이 사용됩니다.

PITR을 사용해서 `EarliestRestorableDateTime` 및 `CurrentTime` 사이의 시점으로 복원할 수 있습니다. 테이블 데이터는 PITR이 활성화된 시점으로만 복원할 수 있습니다.

PITR을 비활성화했다가 나중에 다시 활성화하면 사용 가능한 첫 번째 백업의 시작 시간이 PITR이 다시 활성화된 시간으로 재설정됩니다. 즉 PITR을 비활성화하면 백업 기록이 지워집니다.

> **Note**
>
> 테이블에 대한 데이터 정의 언어(DDL) 작업(예: 스키마 변경)은 비동기적으로 수행됩니다. 복원된 테이블 데이터에서는 완료된 작업만 볼 수 있지만 복원 당시 진행 중이었다면 소스 테이블에 추가 작업이 표시될 수 있습니다. DDL 문의 목록은 the section called “DDL 문” 섹션을 참조하세요.

복원하기 위해 테이블을 활성화할 필요는 없습니다. 삭제된 테이블에서 PITR이 활성화되어 있고 백업 기간 내에(또는 지난 35일 이내) 삭제가 발생한 경우 삭제된 테이블을 복원할 수도 있습니다.

> **Note**
>
> 이전에 삭제한 테이블과 동일한 정규화된 이름(예: mykeyspace.mytable)으로 새 테이블을 생성하는 경우 삭제된 테이블을 더 이상 복원할 수 없습니다. 콘솔에서 이를 시도할 경우 경고가 표시됩니다.

PITR 복원 설정

PITR을 사용하여 테이블을 복원하는 경우 Amazon Keyspaces는 소스 테이블의 스키마 및 데이터를 선택한 타임스탬프(`day:hour:minute:second`)를 기반으로 한 상태로 새 테이블에 복원합니다. PITR은 기존 테이블을 덮어쓰지 않습니다.

테이블의 스키마와 데이터 외에도 PITR은 소스 테이블에서 `custom_properties`를 복원합니다. 가장 빠른 복원 시간과 현재 시간 사이의 선택한 타임스탬프를 기준으로 복원되는 테이블의 데이터와 달리 사용자 지정 속성은 항상 현재 시간을 기준으로 테이블의 설정을 기준으로 복원됩니다.

복원된 테이블의 설정은 복원이 시작된 시점의 타임스탬프가 있는 소스 테이블의 설정과 일치합니다. 복원 중 이러한 설정을 덮어쓰려는 경우 `WITH custom_properties`를 사용하여 덮어쓸 수 있습니다. 사용자 지정 속성에는 다음 설정이 포함됩니다.

- 읽기/쓰기 용량 모드
- 프로비저닝된 처리량 용량 설정
- PITR 설정

테이블이 Auto Scaling이 활성화된 프로비저닝된 용량 모드에 있는 경우 복원 작업은 테이블의 Auto Scaling 설정도 복원합니다. CQL의 `autoscaling_settings` 파라미터를 사용하거나 CLI를 사용하여 덮어쓸 수 있습니다. `autoScalingSpecification` Auto Scaling 설정에 대한 자세한 내용은 을 참조하십시오the section called “Auto Scaling을 통한 처리 용량 관리”.

전체 테이블 복원을 수행하면 복원된 테이블의 모든 테이블 설정은 복원 시 원본 테이블의 현재 설정에서 가져옵니다.

예를 들어 테이블의 프로비저닝된 처리량이 최근에 읽기 용량 단위 50 및 쓰기 용량 단위 50으로 낮춰졌다고 가정합니다. 그런 다음 테이블 상태를 3주 전으로 복원합니다. 이 때 프로비저닝된 처리량은 읽기 용량 단위 100 및 쓰기 용량 단위 100으로 설정되었습니다. 이 경우 Amazon Keyspaces는 테이블 데이터를 해당 시점으로 복원하지만 현재 프로비저닝된 처리량 설정(읽기 용량 단위 50 및 쓰기 용량 단위 50)을 사용합니다.

다음 설정은 복원되지 않으므로 새 테이블에 맞게 수동으로 구성해야 합니다.

- AWS Identity and Access Management (IAM) 정책
- 아마존 CloudWatch 메트릭스 및 알람
- 태그(`WITH TAGS`를 사용하여 CQL `RESTORE` 문에 추가 가능)

암호화된 테이블의 PITR 복원

PITR을 사용하여 테이블을 복원할 때 Amazon Keyspaces는 소스 테이블의 암호화 설정을 복원합니다. 테이블이 AWS 소유 키(기본값)로 암호화된 경우 테이블은 동일한 설정으로 자동으로 복원됩니다. 복원하려는 테이블이 고객 관리형 키를 사용하여 암호화된 경우 테이블 데이터를 복원하려면 Amazon Keyspaces에서 동일한 고객 관리형 키에 액세스할 수 있어야 합니다.

복원 시 테이블의 암호화 설정을 변경할 수 있습니다. AWS 소유 키에서 고객 관리형 키로 변경하려면 복원 시 유효하고 액세스 가능한 고객 관리형 키를 제공해야 합니다.

고객 관리형 키에서 AWS 소유 키로 변경하려면 Amazon Keyspaces가 소스 테이블의 고객 관리형 키에 액세스할 수 있는지 확인하여 AWS 소유 키를 사용하여 테이블을 복원합니다. 테이블의 저장 시 암호화 설정에 대한 자세한 내용은 the section called “작동 방식” 섹션을 참조하세요.

> **Note**
>
> Amazon Keyspaces에서 고객 관리형 키에 액세스할 수 없어 테이블이 삭제된 경우 테이블을 복원하기 전에 Amazon Keyspaces에서 고객 관리형 키에 액세스할 수 있는지 확인해야 합니다. 고객 관리형 키로 암호화된 테이블은 Amazon Keyspaces가 해당 키에 액세스할 수 없는 경우 복원할 수 없습니다. 자세한 내용은 AWS Key Management Service 개발자 안내서의 키 액세스 문제 해결을 참조하세요.

다중 리전 테이블의 PITR 복원

PITR을 사용하여 다중 리전 테이블을 복원할 수 있습니다. 복원 작업이 성공하려면 소스 테이블과 대상 테이블을 모두 동일한 AWS 리전로 복제해야 합니다.

Amazon Keyspace는 키스페이스의 일부인 각 복제 지역의 원본 테이블 설정을 복원합니다. 복원 작업 중에 설정을 재정의할 수도 있습니다. 복원 중에 변경할 수 있는 설정에 대한 자세한 내용은 the section called “복원 설정” 섹션을 참조하세요.

다중 리전 복제에 대한 자세한 내용은 the section called “작동 방식” 섹션을 참조하세요.

PITR을 사용한 테이블 복원 시간

테이블을 복원하는 데 걸리는 시간은 여러 요소에 따라 달라지며 테이블의 크기와 항상 직접적인 연관 관계가 있는 것은 아닙니다.

다음은 복원 시간에 대한 몇 가지 고려 사항입니다.

- 백업을 새로운 테이블에 복원합니다. 새로운 테이블 생성과 복원 프로세스 시작에 필요한 모든 작업을 수행하는 데 최대 20분이 걸릴 수 있습니다(테이블이 비어있더라도 마찬가지임).
- 잘 분산된 데이터 모델을 사용하는 대형 테이블의 복원 시간은 몇 시간 이상일 수 있습니다.
- 소스 테이블에 크게 왜곡된 데이터가 포함되어 있으면 복원 시간이 늘어날 수 있습니다. 예를 들어 테이블의 프라이머리 키가 파티션 키로 그해의 월을 사용하고 있고 모든 데이터가 12월에 집중된 경우 편중 데이터가 있는 것입니다.

재해 복구를 계획할 때 가장 좋은 방법은 평균 복원 완료 시간을 정기적으로 기록하고 이러한 시간이 전체 복구 시간 목표에 어떤 영향을 미치는지 설정하는 것입니다.

Amazon Keyspaces PITR 및 AWS 서비스와 통합

지속적인 모니터링 및 감사를 활성화하기 위해 AWS CloudTrail을 사용하여 다음 PITR 작업이 기록됩니다.

- PITR을 활성화하거나 비활성화하여 새 테이블을 생성합니다.
- 기존 테이블에서 PITR을 활성화 또는 비활성화합니다.
- 활성 테이블 또는 삭제된 테이블을 복원합니다.

자세한 설명은 를 사용하여 Amazon Keyspaces API 호출을 로깅합니다 AWS CloudTrail 섹션을 참조하세요.

AWS CloudFormation을 사용하여 다음 PITR 작업을 수행할 수 있습니다.

- PITR을 활성화하거나 비활성화하여 새 테이블을 생성합니다.
- 기존 테이블에서 PITR을 활성화 또는 비활성화합니다.

자세한 내용은 AWS CloudFormation 사용 설명서의 Cassandra 리소스 유형 참조를 참조하세요.

Amazon Keyspaces 테이블을 특정 시점으로 복원

Amazon Keyspaces(Apache Cassandra용) 시점 복구(PITR)를 사용하면 최근 35일 중 원하는 시점으로 Amazon Keyspaces 테이블 데이터를 복원할 수 있습니다. 이 자습서의 첫 부분에서는 Amazon Keyspaces 콘솔, AWS Command Line Interface(AWS CLI) 및 Cassandra 쿼리 언어(CQL)를 사용하여 테이블을 특정 시점으로 복원하는 방법을 보여 줍니다. 두 번째 부분에서는 AWS CLI 및 CQL을 사용하여 삭제된 테이블을 복원하는 방법을 보여 줍니다.

주제

시작하기 전에

아직 구성하지 않았다면 Amazon Keyspaces 테이블을 복원할 수 있는 적절한 권한을 구성해야 합니다. AWS Identity and Access Management(IAM)의 AWS 관리형 `AmazonKeyspacesFullAccess` 정책에는 Amazon Keyspaces 테이블을 복원할 수 있는 권한이 포함됩니다. 필요한 최소 권한으로 정책을 구현하는 자세한 단계는 the section called “권한 복원” 섹션을 참조하세요.

테이블을 특정 시점으로 복원(콘솔)

다음은 Amazon Keyspaces 콘솔을 사용하여 `mytable`이라는 기존 테이블을 특정 시점으로 복원하는 방법을 보여 주는 예입니다.

> **Note**
>
> 이 절차에서는 특정 시점으로 복구를 활성화했다고 가정합니다. `mytable` 테이블에 PITR을 활성화하려면 the section called “콘솔 사용”의 단계를 따릅니다.

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.

2. 콘솔 왼쪽의 탐색 창에서 테이블을 선택합니다.
3. 테이블 목록에서 `mytable` 테이블을 선택합니다.
4. `mytable` 테이블에 있는 백업 탭의 특정 시점 복구 섹션에서 복원을 선택합니다.
5. 새 테이블 이름에 **`mytable_restored`**를 입력합니다.
6. 복원 작업의 시점을 정의하려면 다음 두 옵션 중에서 선택할 수 있습니다.
   - 사전 구성된 가장 이른 시간을 선택합니다.
   - 날짜 및 시간 지정을 선택하고 새 테이블을 복원하려는 날짜와 시간을 입력합니다.

> **Note**
>
> 가장 이른 시간과 현재 시간 내 어느 특정 시점으로든 복원할 수 있습니다. Amazon Keyspaces는 테이블 데이터를 선택한 날짜와 시간(요일:시:분:초)을 기준으로 한 상태로 복원합니다.

7. 복원을 선택하여 복원 프로세스를 시작합니다.

   복원 중인 테이블은 상태가 복원 중으로 표시됩니다. 복원 프로세스가 완료되면 `mytable_restored` 테이블의 상태가 Active로 변경됩니다.

> **Important**
>
> 복원을 진행하는 동안 IAM 엔터티(예: 사용자, 그룹 또는 역할)에 복원 수행 권한을 부여하는 AWS Identity and Access Management(IAM) 정책을 수정하거나 삭제하지 마세요. 그러면 예기치 않은 동작이 발생할 수 있습니다. 예를 들어 테이블 복원 중에 테이블에 대한 쓰기 권한을 삭제했다고 가정해 보겠습니다. 이 경우 기본 `RestoreTableToPointInTime` 작업에서 복원된 데이터를 해당 테이블에 쓸 수 없게 됩니다.
>
> 복원 작업이 완료되어야 권한을 수정하거나 삭제할 수 있습니다.

테이블을 AWS CLI를 사용하여 특정 시점으로 복원

다음은 AWS CLI를 사용하여 `myTable`이라는 기존 테이블을 특정 시점으로 복원하는 방법을 보여 주는 절차입니다.

1. 첫 번째 단계에서는 PITR이 활성화된 myTable이라는 간단한 테이블을 만듭니다. 가독성을 위해 명령을 별도의 줄로 나누었습니다.

```
aws keyspaces create-table --keyspace-name 'myKeyspace' --table-name 'myTable'
            --schema-definition 'allColumns=[{name=id,type=int},
{name=name,type=text},{name=date,type=timestamp}],partitionKeys=[{name=id}]'
            --point-in-time-recovery 'status=ENABLED'
```

2. 새 테이블의 속성을 확인하고 PITR에 대한 earliestRestorableTimestamp를 검토합니다.

```
aws keyspaces get-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

이 명령의 출력은 다음을 반환합니다.

```
{
    "keyspaceName": "myKeyspace",
    "tableName": "myTable",
    "resourceArn": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/myKeyspace/
table/myTable",
    "creationTimestamp": "2022-06-20T14:34:57.049000-07:00",
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
```

```
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": "2022-06-20T14:34:57.049000-07:00"
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "ENABLED",
        "earliestRestorableTimestamp": "2022-06-20T14:35:13.693000-07:00"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    }
}
```

활성 테이블을 1초 간격으로 `earliestRestorableTimestamp`와 현재 시간 사이의 원하는 시점으로 복원할 수 있습니다. 현재 시간이 기본값입니다.

3. 테이블을 특정 시점으로 복원하려면 ISO 8601 형식으로 `restore_timestamp`를 지정합니다. 최근 35일 중 원하는 시점으로 1초 간격으로 선택할 수 있습니다. 예를 들어 다음 명령은 테이블을 `EarliestRestorableDateTime`으로 복원합니다.

```
aws keyspaces restore-table --source-keyspace-name 'myKeyspace' --source-
table-name 'myTable' --target-keyspace-name 'myKeyspace' --target-table-name
 'myTable_restored' --restore-timestamp "2022-06-20 21:35:14.693"
```

이 명령의 출력은 복원된 테이블의 ARN을 반환합니다.

```
{
    "restoredTableARN": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/
myKeyspace/table/myTable_restored"
}
```

테이블을 현재 시간으로 복원하려면 `restore-timestamp`를 생략할 수 있습니다.

```
aws keyspaces restore-table --source-keyspace-name 'myKeyspace' --source-
table-name 'myTable' --target-keyspace-name 'myKeyspace' --target-table-name
 'myTable_restored1'"
```

Important

복원을 진행하는 동안 IAM 엔터티(예: 사용자, 그룹 또는 역할)에 복원 수행 권한을 부여하는 AWS Identity and Access Management(IAM) 정책을 수정하거나 삭제하지 마세요. 그러면 예기치 않은 동작이 발생할 수 있습니다. 예를 들어 테이블 복원 중에 테이블에 대한 쓰기 권한을 삭제했다고 가정해 보겠습니다. 이 경우 기본 `RestoreTableToPointInTime` 작업에서 복원된 데이터를 해당 테이블에 쓸 수 없게 됩니다.
복원 작업이 완료되어야 권한을 수정하거나 삭제할 수 있습니다.

테이블을 CQL을 사용하여 특정 시점으로 복원

다음은 CQL을 사용하여 `mytable`이라는 기존 테이블을 특정 시점으로 복원하는 방법을 보여 주는 절차입니다.

Note

이 절차에서는 특정 시점으로 복구를 활성화했다고 가정합니다. 테이블에 PITR을 활성화하려면 the section called “CQL”의 단계를 따릅니다.

1. 활성 테이블을 `earliest_restorable_timestamp`와 현재 시간 사이의 원하는 시점으로 복원할 수 있습니다. 현재 시간이 기본값입니다.

   `mytable` 테이블에 대해 시점 복구가 활성화되었는지 확인하려면 다음과 같이 `system_schema_mcs.tables`를 쿼리합니다.

   ```
   SELECT custom_properties
   FROM system_schema_mcs.tables
   WHERE keyspace_name = 'mykeyspace' AND table_name = 'mytable';
   ```

   다음 샘플 출력과 같이 시점 복구가 활성화되었습니다.

```
custom_properties
-----------------
{
  ...,
    "point_in_time_recovery": {
    "earliest_restorable_timestamp":"2020-06-30T19:19:21.175Z"
    "status":"enabled"
  }
}
```

2. ISO 8601 형식의 `restore_timestamp`로 지정된 특정 시점으로 테이블을 복원합니다. 이 경우 `mytable` 테이블이 현재 시간으로 복원됩니다. `WITH restore_timestamp = ...` 절을 생략할 수 있습니다. 절이 없으면 현재 타임스탬프가 사용됩니다.

   ```
   RESTORE TABLE mykeyspace.mytable_restored
   FROM TABLE mykeyspace.mytable;
   ```

   특정 시점으로 복원할 수도 있습니다. 최근 35일 중 원하는 시점을 지정할 수 있습니다. 예를 들어 다음 명령은 테이블을 `EarliestRestorableDateTime`으로 복원합니다.

   ```
   RESTORE TABLE mykeyspace.mytable_restored
   FROM TABLE mykeyspace.mytable
   WITH restore_timestamp = '2020-06-30T19:19:21.175Z';
   ```

   전체 구문 설명은 언어 참조의 the section called “RESTORE TABLE” 섹션을 참조하세요.

테이블 복원이 성공했는지 확인하려면 `system_schema_mcs.tables`를 쿼리하여 테이블 상태를 확인합니다.

```
SELECT status
FROM system_schema_mcs.tables
WHERE keyspace_name = 'mykeyspace' AND table_name = 'mytable_restored'
```

쿼리는 다음 출력을 보여 줍니다.

```
status
------
RESTORING
```

복원 중인 테이블은 상태가 복원 중으로 표시됩니다. 복원 프로세스가 완료되면 mytable_restored 테이블의 상태가 Active로 변경됩니다.

> **Important**
>
> 복원을 진행하는 동안 IAM 엔터티(예: 사용자, 그룹 또는 역할)에 복원 수행 권한을 부여하는 AWS Identity and Access Management(IAM) 정책을 수정하거나 삭제하지 마세요. 그러면 예기치 않은 동작이 발생할 수 있습니다. 예를 들어 테이블 복원 중에 테이블에 대한 쓰기 권한을 삭제했다고 가정해 보겠습니다. 이 경우 기본 RestoreTableToPointInTime 작업에서 복원된 데이터를 해당 테이블에 쓸 수 없게 됩니다.
> 복원 작업이 완료되어야 권한을 수정하거나 삭제할 수 있습니다.

AWS CLI를 사용하여 삭제된 테이블 복원

다음은 AWS CLI를 사용하여 myTable이라는 삭제된 테이블을 삭제 시점으로 복원하는 방법을 보여주는 절차입니다.

> **Note**
>
> 이 절차에서는 삭제된 테이블에서 PITR이 활성화되었다고 가정합니다.

1. 이전 자습서에서 생성한 테이블을 삭제합니다.

```
aws keyspaces delete-table --keyspace-name 'myKeyspace' --table-name 'myTable'
```

2. 다음 명령을 사용하여 삭제된 테이블을 삭제 시점으로 복원합니다.

```
aws keyspaces restore-table --source-keyspace-name 'myKeyspace' --source-
table-name 'myTable' --target-keyspace-name 'myKeyspace' --target-table-name
 'myTable_restored2'
```

이 명령의 출력은 복원된 테이블의 ARN을 반환합니다.

```
{
    "restoredTableARN": "arn:aws:cassandra:us-east-1:111222333444:/keyspace/
myKeyspace/table/myTable_restored2"
```

```
}
```

CQL을 사용하여 삭제된 테이블 복원

다음은 CQL을 사용하여 mytable이라는 삭제된 테이블을 삭제 시점으로 복원하는 방법을 보여 주는 절차입니다.

> **Note**
>
> 이 절차에서는 삭제된 테이블에서 PITR이 활성화되었다고 가정합니다.

1. 삭제된 테이블에 대해 특정 시점 복구가 활성화되었는지 확인하려면 시스템 테이블을 쿼리합니다. 시점 복구가 활성화된 테이블만 표시됩니다.

   ```
   SELECT custom_properties
   FROM system_schema_mcs.tables_history
   WHERE keyspace_name = 'mykeyspace' AND table_name = 'my_table';
   ```

   쿼리는 다음 출력을 보여 줍니다.

   ```
   custom_properties
   ------------------
   {
       ...,
      "point_in_time_recovery":{
         "restorable_until_time":"2020-08-04T00:48:58.381Z",
         "status":"enabled"
      }
   }
   ```

2. 다음 샘플 문을 사용하여 테이블을 삭제 시점으로 복원합니다.

   ```
   RESTORE TABLE mykeyspace.mytable_restored
   FROM TABLE mykeyspace.mytable;
   ```

Amazon Keyspaces TTL(Time To Live)을 사용하여 데이터 만료

Amazon Keyspaces(Apache Cassandra용) TTL(Time To Live)을 통해 테이블의 데이터를 자동으로 만료시켜 애플리케이션 로직을 간소화하고 스토리지 가격을 최적화할 수 있습니다. 더 이상 필요하지 않은 데이터는 설정한 TTL 값에 따라 테이블에서 자동으로 삭제됩니다. 따라서 데이터 보존 기간을 정의하거나 데이터 삭제 시기를 지정하는 비즈니스, 업계 또는 규제 요구 사항을 기반으로 하는 데이터 보존 정책을 보다 쉽게 준수할 수 있습니다.

예를 들어 AdTech 애플리케이션에서 TTL을 사용하여 특정 광고의 데이터가 만료되어 고객에게 더 이상 표시되지 않는 시기를 예약할 수 있습니다. 또한 TTL을 사용하여 오래된 데이터를 자동으로 폐기하고 스토리지 비용을 절약할 수 있습니다. 전체 테이블에 기본 TTL 값을 설정하고 개별 행과 열에 해당 값을 덮어쓸 수 있습니다. TTL 작업은 애플리케이션 성능에 영향을 주지 않습니다. 또한 TTL로 만료되도록 표시된 행과 열의 수는 테이블 가용성에 영향을 주지 않습니다.

Amazon Keyspaces는 만료된 데이터를 자동으로 필터링하여 만료된 데이터가 쿼리 결과에 반환되거나 데이터 조작 언어(DML) 문에 사용할 수 없도록 합니다. Amazon Keyspaces는 일반적으로 만료 날짜로부터 10일 이내에 스토리지에서 만료된 데이터를 삭제합니다. 드문 경우이긴 하지만 가용성 보호를 위해 기본 스토리지 파티션에 지속적인 활동이 있는 경우 Amazon Keyspaces가 10일 이내에 데이터를 삭제하지 못할 수도 있습니다. 이러한 경우 Amazon Keyspaces는 파티션의 트래픽이 감소하면 만료된 데이터를 계속 삭제하려고 시도합니다. 데이터를 스토리지에서 영구적으로 삭제한 후에는 스토리지 요금 발생이 중지됩니다. 자세한 내용은 the section called “작동 방식” 섹션을 참조하세요.

콘솔 또는 Cassandra 쿼리 언어(CQL)를 사용하여 새 테이블과 기존 테이블의 기본 TTL 설정을 지정, 수정 또는 비활성화할 수 있습니다. 기본 TTL이 구성된 테이블에서 Cassandra 쿼리 언어(CQL)를 사용하여 기본 TTL 설정을 재정의하고 사용자 지정 TTL 값을 행과 열에 적용할 수 있습니다. 자세한 내용은 the section called “TTL(Time To Live) 사용 방법” 섹션을 참조하세요.

TTL 요금은 TTL(Time To Live)을 사용하여 삭제되거나 업데이트되는 행의 크기를 기준으로 책정됩니다. TTL 연산은 `TTL deletes` 단위로 측정됩니다. 삭제되거나 업데이트된 행당 데이터 KB당 TTL 삭제 1회가 사용됩니다. 예를 들어 2.5KB의 데이터를 저장하는 행을 업데이트하고 행 내에서 하나 이상의 열을 동시에 삭제하려면 3 TTL deletes가 필요합니다. 또는 3.5KB의 데이터가 포함된 행 전체를 삭제하려면 4 TTL deletes가 필요합니다. 행당 삭제된 데이터 KB당 TTL 삭제 1회가 사용됩니다. 요금에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 요금을 참조하세요.

주제

- TTL(Time To Live) 사용 방법

작동 방식: Amazon Keyspaces TTL(Time To Live)

Amazon Keyspaces TTL(Time To Live)은 완전히 관리되는 서비스입니다. 컴팩션 전략과 같은 낮은 수준의 시스템 설정을 관리할 필요가 없습니다. 날짜는 지정한 시간에 만료되며 Amazon Keyspaces는 애플리케이션 성능이나 가용성에 영향을 주지 않고 만료된 데이터를 자동으로(일반적으로 10일 이내) 제거합니다.

만료된 데이터는 삭제 대상으로 표시되며 데이터 조작 언어(DML) 문에는 사용할 수 없습니다. 삭제된 데이터가 포함된 행에 대해 읽기 및 쓰기를 계속 수행하면 만료된 데이터는 스토리지에서 삭제될 때까지 읽기 용량 단위(RCU) 및 쓰기 용량 단위(WCU)에 계속 포함됩니다.

주제

- 테이블의 기본 TTL 값 설정
- 행과 열의 사용자 지정 TTL 값 설정
- 테이블에서 TTL 활성화
- Amazon Keyspaces TTL(Time To Live) 및 AWS 서비스와 통합

테이블의 기본 TTL 값 설정

Amazon Keyspaces에서는 테이블 생성 시 테이블의 모든 행에 대해 기본 TTL 값을 설정할 수 있습니다. 기존 테이블을 편집하여 테이블에 삽입된 새 행의 기본 TTL 값을 설정하거나 변경할 수도 있습니다. 테이블의 기본 TTL 값을 변경해도 테이블에 있는 기존 데이터의 TTL 값은 수정되지 않습니다. 테이블의 기본 TTL 값은 0이며, 이는 데이터가 자동으로 만료되지 않음을 의미합니다. 테이블의 기본 TTL 값이 0보다 크면 각 행에 만료 타임스탬프가 추가됩니다.

Amazon Keyspaces는 데이터가 업데이트될 때마다 새로운 TTL 타임스탬프를 계산합니다. TTL 값은 초 단위로 설정되며 구성 가능한 최댓값은 630,720,000초로 20년에 해당합니다. AWS Management Console 또는 CQL을 사용하여 테이블의 기본 TTL 값을 설정, 수정 및 비활성화하는 방법에 대한 자세한 내용은 the section called “TTL(Time To Live) 사용 방법” 섹션을 참조하세요.

행과 열의 사용자 지정 TTL 값 설정

> **Note**
>
> 행과 열에 사용자 지정 TTL 값을 설정하기 전에 먼저 테이블에서 TTL을 활성화해야 합니다. 자세한 내용은 the section called “사용자 지정 속성을 사용하여 기존 테이블에 대한 TTL(Time To Live)을 활성화하는 방법” 섹션을 참조하세요.

테이블의 기본 TTL 값을 덮어쓰거나 개별 행의 만료 날짜를 설정하려면 다음 CQL 데이터 조작 언어(DML) 문을 사용할 수 있습니다.

- `INSERT` - TTL 값이 설정된 새 데이터 행을 삽입하는 데 사용합니다.
- `UPDATE` - 새 TTL 값으로 기존 데이터 행을 수정하는 데 사용합니다.

행의 TTL 값 설정은 테이블의 기본 TTL 설정보다 우선합니다.

CQL 구문 및 예제는 the section called “`INSERT`를 사용하여 CQL을 사용하여 사용자 지정 TTL(Time To Live) 설정을 편집하려면” 섹션을 참조하세요.

개별 열의 TTL 값을 덮어쓰거나 설정하려면 다음 CQL DML 문을 사용하여 기존 행 내의 열 하위 집합에 대한 TTL 설정을 업데이트할 수 있습니다.

- `UPDATE` - 데이터 열을 업데이트하는 데 사용합니다.

열의 TTL 값 설정은 테이블에 대한 기본 TTL 설정 및 행에 대한 사용자 지정 TTL 설정보다 우선합니다. CQL 구문 및 예제는 the section called “`UPDATE`를 사용하여 CQL을 사용하여 사용자 지정 TTL(Time To Live) 설정을 편집하려면” 섹션을 참조하세요.

테이블에서 TTL 활성화

`CREATE TABLE` 또는 `ALTER TABLE` 문에서 0보다 큰 `default_time_to_live` 값을 지정하면 테이블에 대해 TTL이 자동으로 활성화됩니다. 테이블에 `default_time_to_live`를 지정하지 않고 `INSERT` 또는 `UPDATE` 작업을 사용하여 행 또는 열의 사용자 지정 TTL 값을 지정하려면 먼저 테이블에 대해 TTL을 활성화해야 합니다. `ttl` 사용자 지정 속성을 사용하여 테이블에 TTL을 활성화할 수 있습니다.

테이블에서 TTL을 활성화하면 Amazon Keyspaces가 각 행에 대해 추가 TTL 관련 메타데이터를 저장하기 시작합니다. 또한 TTL은 만료 타임스탬프를 사용하여 행 또는 열이 만료되는 시기를 추적합니다. 타임스탬프는 행 메타데이터로 저장되며 행의 스토리지 비용에 영향을 줍니다.

TTL 기능이 활성화된 후에는 테이블에 대해 이 기능을 비활성화할 수 없습니다. 테이블의 `default_time_to_live`를 0으로 설정하면 새 데이터에 대한 기본 만료 시간이 비활성화되지만 TTL 기능이 비활성화되거나 테이블을 원래 Amazon Keyspaces 스토리지 메타데이터 또는 쓰기 동작으로 되돌리지는 않습니다.

Amazon Keyspaces TTL(Time To Live) 및 AWS 서비스와 통합

Amazon CloudWatch에서는 다음과 같은 TTL 지표를 사용하여 지속적인 모니터링을 지원합니다.

- `TTLDeletes` - TTL(Time To Live)을 사용하여 행의 데이터를 삭제하거나 업데이트하는 데 사용되는 단위입니다.

CloudWatch 지표를 모니터링하는 방법에 대한 자세한 내용은 the section called “를 통한 모니터링 CloudWatch” 섹션을 참조하세요.

AWS CloudFormation을 사용하면 Amazon Keyspaces 테이블을 생성할 때 TTL을 활성화할 수 있습니다. 자세한 내용은 AWS CloudFormation 사용 설명서를 참조하세요.

TTL(Time To Live) 사용 방법

Amazon Keyspaces(Apache Cassandra용) 콘솔 또는 CQL을 사용하여 TTL(Time To Live) 설정을 활성화, 업데이트 및 비활성화할 수 있습니다.

주제

기본 TTL(Time To Live) 설정을 활성화하여 새 테이블을 만들려면(콘솔)

다음 단계에 따라 Amazon Keyspaces 콘솔을 사용하여 TTL(Time To Live) 설정을 활성화하여 새 테이블을 생성합니다.

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 테이블을 선택한 다음 테이블 생성을 선택합니다.
3. 테이블 세부 정보 섹션의 테이블 생성 페이지에서 키스페이스를 선택하고 새 테이블의 이름을 입력합니다.
4. 스키마 섹션에서 테이블의 스키마를 생성합니다.
5. 테이블 설정 섹션에서 설정 사용자 지정을 선택합니다.
6. TTL(Time To Live)을 계속합니다.

   이 단계에서는 테이블의 기본 TTL 설정을 선택합니다.

   기본 TTL 기간의 경우 만료 시간을 입력하고 입력한 시간 단위(예: 초, 일 또는 년)를 선택합니다. Amazon Keyspaces는 초 단위로 값을 저장합니다.
7. 테이블 생성을 선택합니다. 테이블은 지정된 기본 TTL 값으로 생성됩니다.

> **Note**
>
> CQL 편집기에서 데이터 조작 언어(DML)를 사용하여 특정 행 또는 열에 대한 테이블의 기본 TTL 설정을 덮어쓸 수 있습니다.

기존 테이블에 대한 기본 TTL(Time To Live) 설정을 업데이트하려면(콘솔)

다음 단계에 따라 Amazon Keyspaces 콘솔을 사용하여 기존 테이블에 대한 TTL(Time To Live) 설정을 업데이트합니다.

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 업데이트할 테이블을 선택한 다음 추가 설정 탭을 선택합니다.
3. TTL(Time To Live)을 계속하고 편집을 선택합니다.

4. 기본 TTL 기간의 경우 만료 시간을 입력하고 입력한 시간 단위(예: 초, 일 또는 년)를 선택합니다. Amazon Keyspaces는 초 단위로 값을 저장합니다. 이렇게 해도 기존 행의 TTL 값은 변경되지 않습니다.
5. TTL 설정이 정의되면 변경 내용 저장을 선택합니다.

기존 테이블에 대한 기본 TTL(Time To Live) 설정을 비활성화하려면(콘솔)

다음 단계에 따라 Amazon Keyspaces AWS Management Console을 사용하여 기존 테이블에 대한 TTL(Time To Live) 설정을 비활성화합니다.

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 업데이트할 테이블을 선택한 다음 추가 설정 탭을 선택합니다.
3. TTL(Time To Live)을 계속하고 편집을 선택합니다.
4. 기본 TTL 기간을 선택하고 값을 0으로 설정합니다. 이렇게 하면 향후 데이터에 대한 테이블의 TTL이 기본적으로 비활성화됩니다. 이렇게 해도 기존 행의 TTL 값은 변경되지 않습니다.
5. TTL 설정이 정의되면 변경 내용 저장을 선택합니다.

CQL을 사용하여 기본 TTL(Time To Live) 설정을 활성화하여 새 테이블을 만들려면(콘솔)

기본 TTL 값이 35일을 나타내는 3,024,000초로 설정된 새 테이블을 생성할 때 TTL을 활성화합니다.

```
CREATE TABLE my_table (
                userid uuid,
                time timeuuid,
                subject text,
                body text,
                user inet,
                PRIMARY KEY (userid, time)
                ) WITH default_time_to_live = 3024000;
```

새 테이블의 TTL 설정을 확인하려면 다음 예와 같이 `cqlsh describe` 문을 사용합니다. 출력에는 테이블의 기본 TTL 설정이 `default_time_to_live`로 표시됩니다.

```
describe my_table;
```

ALTER TABLE을 사용하여 CQL을 사용하여 기본 TTL(Time To Live) 설정을 편집하려면

기존 테이블의 TTL 설정을 2,592,000초(30일을 나타내는 초)로 업데이트합니다.

```
ALTER TABLE my_table WITH default_time_to_live = 2592000;
```

업데이트된 테이블의 TTL 설정을 확인하려면 다음 예와 같이 cqlsh describe 문을 사용합니다. 출력에는 테이블의 기본 TTL 설정이 default_time_to_live로 표시됩니다.

```
describe my_table;
```

사용자 지정 속성을 사용하여 새 테이블에 대한 TTL(Time To Live)을 활성화하는 방법

전체 테이블에 대해 TTL 기본 설정을 활성화하지 않고 행과 열에 적용할 수 있는 TTL(Time To Live) 사용자 정의 설정을 활성화하려면 다음 CQL 문을 사용할 수 있습니다.

```
CREATE TABLE my_keyspace.my_table (id int primary key) WITH CUSTOM_PROPERTIES={'ttl':
{'status': 'enabled'}};
```

ttl이 활성화된 후에는 테이블에 대해 이 설정을 비활성화할 수 없습니다.

사용자 지정 속성을 사용하여 기존 테이블에 대한 TTL(Time To Live)을 활성화하는 방법

전체 테이블에 대해 TTL 기본 설정을 활성화하지 않고 행과 열에 적용할 수 있는 TTL(Time To Live) 사용자 정의 설정을 활성화하려면 다음 CQL 문을 사용할 수 있습니다.

```
ALTER TABLE my_table WITH CUSTOM_PROPERTIES={'ttl':{'status': 'enabled'}};
```

ttl이 활성화된 후에는 테이블에 대해 이 설정을 비활성화할 수 없습니다.

INSERT를 사용하여 CQL을 사용하여 사용자 지정 TTL(Time To Live) 설정을 편집하려면

다음 CQL 문은 테이블에 데이터 행을 삽입하고 기본 TTL 설정을 259,200초(3일에 해당)로 변경합니다.

```
INSERT INTO my_table (userid, time, subject, body, user)
        VALUES (B79CB3BA-745E-5D9A-8903-4A02327A7E09, 96a29100-5e25-11ec-90d7-
b5d91eceda0a, 'Message', 'Hello','205.212.123.123')
        USING TTL 259200;
```

삽입된 행의 TTL 설정을 확인하려면 다음 문을 사용합니다.

```
SELECT TTL (subject) from my_table;
```

UPDATE를 사용하여 CQL을 사용하여 사용자 지정 TTL(Time To Live) 설정을 편집하려면

이전에 삽입한 '제목' 열의 TTL 설정을 259,200초(3일)에서 86,400초(1일)로 변경하려면 다음 문을 사용합니다.

```
UPDATE my_table USING TTL 86400 set subject = 'Updated Message' WHERE userid =
 B79CB3BA-745E-5D9A-8903-4A02327A7E09 and time = 96a29100-5e25-11ec-90d7-b5d91eceda0a;
```

간단한 선택 쿼리를 실행하여 만료 시간 전에 업데이트된 레코드를 볼 수 있습니다.

```
SELECT * from my_table;
```

쿼리는 다음 출력을 보여 줍니다.

```
userid                                | time                                 | body  |
 subject         | user
--------------------------------------+--------------------------------------+-------
+-----------------+-----------------
b79cb3ba-745e-5d9a-8903-4a02327a7e09  | 96a29100-5e25-11ec-90d7-b5d91eceda0a | Hello |
 Updated Message | 205.212.123.123
50554d6e-29bb-11e5-b345-feff819cdc9f  | cf03fb21-59b5-11ec-b371-dff626ab9620 | Hello |
        Message | 205.212.123.123
```

만료가 성공적으로 완료되었는지 확인하려면 구성된 만료 시간 이후에 동일한 쿼리를 다시 실행합니다.

```
SELECT * from my_table;
```

쿼리는 '제목' 열이 만료된 후 다음과 같은 출력을 표시합니다.

```
userid                                | time                                 | body  |
 subject | user
--------------------------------------+--------------------------------------+-------
+---------+-----------------
b79cb3ba-745e-5d9a-8903-4a02327a7e09  | 96a29100-5e25-11ec-90d7-b5d91eceda0a | Hello |
   null | 205.212.123.123
50554d6e-29bb-11e5-b345-feff819cdc9f  | cf03fb21-59b5-11ec-b371-dff626ab9620 | Hello |
 Message | 205.212.123.123
```

Amazon Keyspaces에서 클라이언트 측 타임스탬프 사용하기

Amazon Keyspaces에서 클라이언트 측 타임스탬프는 테이블의 각 셀에 대해 유지되는 Cassandra 호환 타임스탬프입니다. 클라이언트 애플리케이션이 쓰기 순서를 결정하도록 함으로써 충돌 해결에 클라이언트 측 타임스탬프를 사용할 수 있습니다. 예를 들어 전 세계에 분산된 애플리케이션의 클라이언트가 동일한 데이터를 업데이트하는 경우 클라이언트 측 타임스탬프는 클라이언트에서 업데이트된 순서대로 유지됩니다. Amazon Keyspaces는 이러한 타임스탬프를 사용하여 쓰기를 처리합니다. 자세한 내용은 the section called “작동 방식” 섹션을 참조하세요.

테이블에 클라이언트 측 타임스탬프가 활성화되면 데이터 조작 언어(DML) CQL 쿼리에서 `USING TIMESTAMP` 절을 사용하여 타임스탬프를 지정할 수 있습니다. CQL 쿼리에 타임스탬프를 지정하지 않는 경우 Amazon Keyspaces는 클라이언트 드라이버가 전달한 타임스탬프를 사용합니다. 클라이언트 드라이버가 타임스탬프를 제공하지 않는 경우 Amazon Keyspaces는 자동으로 셀 수준의 타임스탬프를 할당합니다. 타임스탬프를 쿼리하려면 DML 문에 `WRITETIME` 함수를 사용할 수 있습니다. 자세한 내용은 the section called “클라이언트 측 타임스탬프를 사용하는 방법” 섹션을 참조하세요.

Amazon Keyspaces는 클라이언트 측 타임스탬프를 활성화하는 데 추가 비용을 청구하지 않습니다. 하지만 클라이언트 측 타임스탬프를 사용하면 행의 각 값에 대해 추가 데이터를 저장하고 쓸 수 있습니다. 이로 인해 스토리지 사용량이 늘어나고 경우에 따라 처리량 사용량이 증가할 수 있습니다. 행 크기에 미치는 영향을 추정하는 방법에 대한 자세한 내용은 the section called “작동 방식” 섹션을 참조하세요. Amazon Keyspaces 요금에 대한 자세한 내용은 Amazon Keyspaces(Apache Cassandra용) 요금을 참조하세요.

주제

Amazon Keyspaces에서 클라이언트 측 타임스탬프의 작동 방식

Amazon Keyspaces 클라이언트 측 타임스탬프는 완전한 관리형입니다. 정리 및 컴팩션 전략과 같은 낮은 수준의 시스템 설정을 관리할 필요가 없습니다.

데이터를 삭제하면 행이 삭제 표시와 함께 삭제로 표시됩니다. Amazon Keyspaces는 애플리케이션 성능이나 가용성에 영향을 주지 않고 삭제된 데이터를 자동으로(일반적으로 10일 이내) 제거합니다. 삭제 표시된 데이터는 데이터 조작 언어(DML) 문에 사용할 수 없습니다. 삭제된 데이터가 포함된 행에 대해 읽기 및 쓰기를 계속 수행하면 삭제된 데이터는 스토리지에서 삭제될 때까지 스토리지, 읽기 용량 단위(RCU), 쓰기 용량 단위(WCU)에 계속 포함됩니다.

주제

Amazon Keyspaces에서 클라이언트 측 타임스탬프의 작동 방식

Amazon Keyspaces에서 클라이언트 측 타임스탬프를 활성화하면 모든 행의 모든 열에 타임스탬프가 저장됩니다. 이러한 타임스탬프는 약 20~40바이트(데이터에 따라 다름)를 차지하며 행의 스토리지 및 처리량 비용에 영향을 줍니다. 이러한 메타데이터 바이트는 1MB 행 크기 할당량에도 포함됩니다. 행 크기가 1MB 미만으로 유지되도록 하기 위해 스토리지 공간의 전체 증가량을 확인하려면 테이블의 열 수와 각 행의 수집 요소 수를 확인하세요. 예를 들어 테이블에 20개의 열이 있고 각 열에 40바이트의 데이터가 저장되어 있는 경우 행 크기는 800바이트에서 1,200바이트로 증가합니다. 행 크기를 추정하는 방법에 대한 자세한 내용은 the section called “행 크기 계산” 섹션을 참조하세요. 이 예세에서는 스토리지를 위한 추가 400바이트 외에도 쓰기당 소비되는 쓰기 용량 단위(WCU) 수가 1WCU에서 2WCU로 늘어납니다. 읽기 및 쓰기 용량을 계산하는 방법에 대한 자세한 내용은 the section called “읽기/쓰기 용량 모드” 섹션을 참조하세요.

테이블에 대해 클라이언트 측 타임스탬프가 활성화된 후에는 타임스탬프를 비활성화할 수 없습니다. 또한 타임스탬프는 NULL이(가) 될 수 없으므로 CQL 문 또는 클라이언트 드라이버에서 클라이언트 측 타임스탬프를 제공하지 않는 경우 Amazon Keyspaces에서 생성된 타임스탬프가 자동으로 추가됩니다.

Amazon Keyspaces 클라이언트 측 타임스탬프 및 AWS 서비스와의 통합

Amazon CloudWatch에서는 다음과 같은 클라이언트 측 타임스탬프 지표를 사용하여 지속적인 모니터링을 지원합니다.

- `SystemReconciliationDeletes` — 삭제된 데이터를 제거하는 데 필요한 삭제 작업 횟수.

CloudWatch 지표를 모니터링하는 방법에 대한 자세한 내용은 the section called “를 통한 모니터링 CloudWatch” 섹션을 참조하세요.

Amazon Keyspaces의 클라이언트 측 타임스탬프 사용

Amazon Keyspaces(Apache Cassandra용) 콘솔, Cassandra 쿼리 언어(CQL), AWS SDK, AWS Command Line Interface(AWS CLI)를 사용하여 클라이언트 측 타임스탬프를 활성화할 수 있습니다. 이 섹션에서는 새 테이블과 기존 테이블에서 클라이언트 측 타임스탬프를 활성화하는 방법과 쿼리

에 클라이언트 측 타임스탬프를 사용하는 방법에 대한 예시를 알아봅니다. API에 대한 자세한 내용은 Amazon Keyspaces API 참조를 참조하세요.

> **Important**
>
> 클라이언트 측 타임스탬프는 비활성화할 수 없습니다. 클라이언트 측 타임스탬프를 활성화하고 나면 다시 변경할 수 있습니다. Amazon Keyspaces는 테이블을 삭제하지 않고 타임스탬프를 비활성화하는 옵션을 제공하지 않습니다.

주제

클라이언트 측 타임스탬프가 활성화된 새 테이블 생성(콘솔)

다음 단계에 따라 Amazon Keyspaces 콘솔을 통해 클라이언트 측 타임스탬프가 활성화된 새 테이블을 생성합니다.

클라이언트 측 타임스탬프가 있는 새 테이블 만들기(콘솔)

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 테이블을 선택한 다음 테이블 생성을 선택합니다.
3. 테이블 세부 정보 섹션의 테이블 생성 페이지에서 키스페이스를 선택하고 새 테이블의 이름을 입력합니다.
4. 스키마 섹션에서 테이블의 스키마를 생성합니다.
5. 테이블 설정 섹션에서 설정 사용자 지정을 선택합니다.
6. 클라이언트 측 타임스탬프로 이동합니다.

테이블의 클라이언트 측 타임스탬프를 활성화하려면 클라이언트 측 타임스탬프 활성화를 선택합니다.

7. 테이블 생성을 선택합니다. 클라이언트 측 타임스탬프가 활성화된 상태로 테이블이 생성됩니다.

기존 테이블의 클라이언트 측 타임스탬프 활성화(콘솔)

다음 단계에 따라 Amazon Keyspaces AWS Management Console를 사용하여 기존 테이블의 클라이언트 측 타임스탬프를 활성화합니다.

기존 테이블의 클라이언트 측 타임스탬프 활성화(콘솔)

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 업데이트할 테이블을 선택한 다음 추가 설정 탭을 선택합니다.
3. 추가 설정 탭에서 클라이언트 측 타임스탬프 수정으로 이동한 다음 클라이언트 측 타임스탬프 활성화를 선택합니다.
4. 변경 내용 저장을 선택하여 표 설정을 변경합니다.

클라이언트 측 타임스탬프가 설정된 새 테이블 생성(CQL)

새 테이블을 만들 때 클라이언트 측 타임스탬프를 활성화하려면 다음 CQL 문을 사용할 수 있습니다.

```
CREATE TABLE my_table (
   userid uuid,
   time timeuuid,
   subject text,
   body text,
   user inet,
   PRIMARY KEY (userid, time)
) WITH CUSTOM_PROPERTIES = {'client_side_timestamps': {'status': 'enabled'}};
```

새 테이블의 클라이언트 측 타임스탬프 설정을 확인하려면 다음 예시와 같이 SELECT 문을 사용하여 custom_properties를 검토합니다.

```
SELECT custom_properties from system_schema_mcs.tables where keyspace_name =
 'my_keyspace' and table_name = 'my_table';
```

이 문의 출력에는 클라이언트 측 타임스탬프의 상태가 표시됩니다.

```
'client_side_timestamps': {'status': 'enabled'}
```

ALTER TABLE(CQL)를 사용하여 기존 테이블의 클라이언트 측 타임스탬프 활성화

기존 테이블의 클라이언트 측 타임스탬프를 활성화하려면 다음 CQL 문을 사용할 수 있습니다.

```
ALTER TABLE my_table WITH custom_properties = {'client_side_timestamps': {'status':
 'enabled'}};;
```

새 테이블의 클라이언트 측 타임스탬프 설정을 확인하려면 다음 예시와 같이 SELECT 문을 사용하여 custom_properties를 검토합니다.

```
SELECT custom_properties from system_schema_mcs.tables where keyspace_name =
 'my_keyspace' and table_name = 'my_table';
```

이 문의 출력에는 클라이언트 측 타임스탬프의 상태가 표시됩니다.

```
'client_side_timestamps': {'status': 'enabled'}
```

클라이언트 측 타임스탬프가 활성화된 새 테이블 생성(CLI)

새 테이블을 만들 때 클라이언트 측 타임스탬프를 활성화하려면 다음 CLI 문을 사용할 수 있습니다.

```
./aws keyspaces create-table \
--keyspace-name my_keyspace \
--table-name my_table \
--client-side-timestamps 'status=ENABLED' \
--schema-definition 'allColumns=[{name=id,type=int},{name=date,type=timestamp},
{name=name,type=text}],partitionKeys=[{name=id}]'
```

새 테이블에 클라이언트 측 타임스탬프가 활성화되어 있는지 확인하려면 다음 코드를 실행합니다.

```
./aws keyspaces get-table \
--keyspace-name my_keyspace \
--table-name my_table
```

결과가 다음 예제와 비슷해야 합니다.

```
{
    "keyspaceName": "my_keyspace",
    "tableName": "my_table",
    "resourceArn": "arn:aws:cassandra:us-east-2:555555555555:/keyspace/my_keyspace/
table/my_table",
    "creationTimestamp": 1662681206.032,
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": 1662681206.032
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "clientSideTimestamps": {
        "status": "ENABLED"
```

```
    },
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    }
}
```

기존 테이블의 클라이언트 측 타임스탬프 활성화(CLI)

CLI를 사용하는 기존 테이블의 클라이언트 측 타임스탬프를 활성화하려면 다음 코드를 사용할 수 있습니다.

```
./aws keyspaces update-table \
--keyspace-name my_keyspace \
--table-name my_table \
--client-side-timestamps 'status=ENABLED'
```

테이블에 대해 클라이언트 측 타임스탬프가 활성화되어 있는지 확인하려면 다음 코드를 실행합니다.

```
./aws keyspaces get-table \
--keyspace-name my_keyspace \
--table-name my_table
```

결과가 다음 예제와 비슷해야 합니다.

```
{
    "keyspaceName": "my_keyspace",
    "tableName": "my_table",
    "resourceArn": "arn:aws:cassandra:us-east-2:555555555555:/keyspace/my_keyspace/
table/my_table",
    "creationTimestamp": 1662681312.906,
    "status": "ACTIVE",
    "schemaDefinition": {
        "allColumns": [
            {
                "name": "id",
                "type": "int"
            },
            {
```

```
                "name": "date",
                "type": "timestamp"
            },
            {
                "name": "name",
                "type": "text"
            }
        ],
        "partitionKeys": [
            {
                "name": "id"
            }
        ],
        "clusteringKeys": [],
        "staticColumns": []
    },
    "capacitySpecification": {
        "throughputMode": "PAY_PER_REQUEST",
        "lastUpdateToPayPerRequestTimestamp": 1662681312.906
    },
    "encryptionSpecification": {
        "type": "AWS_OWNED_KMS_KEY"
    },
    "pointInTimeRecovery": {
        "status": "DISABLED"
    },
    "clientSideTimestamps": {
        "status": "ENABLED"
    },
    "ttl": {
        "status": "ENABLED"
    },
    "defaultTimeToLive": 0,
    "comment": {
        "message": ""
    }
}
```

DML(데이터 조작 언어) 문에서 클라이언트 측 타임스탬프 사용

클라이언트 측 타임스탬프를 활성화하면 INSERT, UPDATE, DELETE 문에 타임스탬프를 USING TIMESTAMP 절과 함께 전달할 수 있습니다. 타임스탬프 값은 epoch로 알려진 표준 기준 시간인 1970년 1월 1일 00:00:00(그리니치 표준시 기준) 이후 경과된 시간을 마이크로초 단위로 나타내는 bigint

값입니다. 클라이언트가 제공하는 타임스탬프는 현재 벽시계 시간으로부터 과거 2일과 미래의 5분 사이 범위에 속해야 합니다. Amazon Keyspaces는 데이터 수명 기간 동안 타임스탬프 메타데이터를 보관합니다. `WRITETIME` 함수를 사용하여 지난 몇 년 동안 발생한 타임스탬프를 조회할 수 있습니다. CQL 구문에 대한 자세한 내용은 the section called “DML 문” 섹션을 참조하세요.

다음 CQL 문은 타임스탬프를 `update_parameter`로 사용하는 방법의 예제입니다.

```
INSERT INTO catalog.book_awards (year, award, rank, category, book_title, author,
 publisher)
   VALUES (2022, 'Wolf', 4, 'Non-Fiction', 'Science Update', 'Ana Carolina Silva',
 'SomePublisher')
   USING TIMESTAMP 1669069624;
```

CQL 쿼리에 타임스탬프를 지정하지 않는 경우 Amazon Keyspaces는 클라이언트 드라이버가 전달한 타임스탬프를 사용합니다. 클라이언트 드라이버가 타임스탬프를 제공하지 않는 경우 Amazon Keyspaces는 쓰기 작업에 서버 측 타임스탬프를 할당합니다.

특정 열에 저장된 타임스탬프 값을 보려면 다음 예시와 같이 `SELECT` 문에 `WRITETIME` 함수를 사용하면 됩니다.

```
SELECT year, award, rank, category, book_title, author, publisher, WRITETIME(year),
 WRITETIME(award), WRITETIME(rank),
  WRITETIME(category), WRITETIME(book_title), WRITETIME(author), WRITETIME(publisher)
 from catalog.book_awards;
```

AWS CloudFormation를 사용하여 Amazon Keyspaces 리소스 생성

Amazon Keyspaces(Apache Cassandra용)는 리소스 및 인프라를 생성하고 관리하는 데 소요되는 시간을 줄일 수 있도록 AWS 리소스를 모델링하고 설정하는 데 도움이 되는 서비스인 AWS CloudFormation과 통합됩니다. 필요한 모든 AWS 리소스(예: 키스페이스 및 테이블)를 설명하는 템플릿을 생성하면 AWS CloudFormation이 해당 리소스의 프로비저닝과 구성을 담당합니다.

AWS CloudFormation을 사용할 때 템플릿을 재사용하여 Amazon Keyspaces 리소스를 일관되고 반복적으로 설정할 수 있습니다. 리소스를 한 번 설명한 다음 여러 AWS 계정 및 리전에서 동일한 리소스를 반복적으로 프로비저닝할 수 있습니다.

Amazon Keyspaces 및 AWS CloudFormation 템플릿

Amazon Keyspaces에 대한 리소스를 프로비저닝하고 구성하려면 AWS CloudFormation 템플릿을 이해해야 합니다. 템플릿은 JSON 또는 YAML로 서식 지정된 텍스트 파일입니다. 이 템플릿은 AWS CloudFormation 스택에서 프로비저닝할 리소스에 대해 설명합니다. JSON 또는 YAML에 익숙하지 않은 경우 AWS CloudFormation Designer를 사용하면 AWS CloudFormation 템플릿을 시작하는 데 도움이 됩니다. 자세한 내용은 AWS CloudFormation 사용 설명서에서 AWS CloudFormation Designer이란 무엇입니까?를 참조하세요.

Amazon Keyspaces는 AWS CloudFormation에서 키스페이스 및 테이블 생성을 지원합니다. AWS CloudFormation 템플릿을 사용하여 생성하는 테이블의 경우 스키마, 읽기/쓰기 모드, 프로비저닝된 처리량 설정을 지정할 수 있습니다. 키스페이스와 테이블에 대한 JSON 및 YAML 템플릿 예제를 비롯한 자세한 내용은 AWS CloudFormation 사용 설명서의 Cassandra 리소스 유형 참조에서 확인할 수 있습니다.

AWS CloudFormation에 대해 자세히 알아보기

AWS CloudFormation에 대한 자세한 내용은 다음 리소스를 참조하세요.

- AWS CloudFormation
- AWS CloudFormation 사용 설명서
- AWS CloudFormation 명령줄 인터페이스 사용 설명서

Amazon Keyspaces(Apache Cassandra용) 모니터링

Amazon Keyspaces 및 다른 AWS 솔루션의 신뢰성, 가용성 및 성능을 유지하려면 모니터링이 중요합니다. AWS는 Amazon Keyspaces를 모니터링하고, 이상이 있을 때 이를 보고하고, 필요한 경우 자동 조치를 취할 수 있도록 다음과 같은 모니터링 도구를 제공합니다.

- Amazon Keyspaces는 계정의 모든 테이블에서 집계된 지연 시간 및 오류를 보여 주는 AWS Management Console에서 사전 구성된 대시보드를 제공합니다.
- Amazon CloudWatch는 AWS에서 실행하는 AWS 리소스와 애플리케이션을 실시간으로 모니터링합니다. 사용자 지정된 대시보드를 사용하여 지표를 수집하고 추적할 수 있습니다. 예를 들어 다양한 시간과 다양한 부하 조건에서 성능을 측정하여 환경에서 일반 Amazon Keyspaces 성능의 기준을 만들 수 있습니다. Amazon Keyspaces가 과거 모니터링 데이터를 저장하는 것을 모니터링하면서 현재 성능 데이터를 이 과거 데이터와 비교하면 일반적인 성능 패턴과 성능 이상을 식별하고 이를 해결할 방법을 고안할 수 있습니다. 기준을 설정하려면 최소한 시스템 오류를 모니터링해야 합니다. 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하세요.
- Amazon CloudWatch 경보는 지정한 기간 동안 단일 메트릭을 모니터링하고 여러 기간에 대해 지정된 임계값과 관련하여 지표 값을 기준으로 하나 이상의 작업을 수행합니다. 예를 들어 Application Auto Scaling으로 프로비저닝된 모드에서 Amazon Keyspaces를 사용하는 경우 작업은 Amazon Simple Notification Service(SNS)에서 Application Auto Scaling 정책을 평가하기 위해 전송되는 알림입니다.

  CloudWatch 경보는 단순히 특정 상태에 있다고 해서 작업을 호출하지 않습니다. 상태가 변경되어 지정된 기간 수 동안 유지되어야 합니다. 자세한 내용은 아마존을 통한 아마존 키스페이스 모니터링 CloudWatch 섹션을 참조하세요.
- Amazon CloudWatch Logs로 Amazon Keyspaces 테이블, CloudTrail, 기타 소스의 로그 파일을 모니터링, 저장 및 액세스할 수 있습니다. CloudWatch Logs는 로그 파일의 정보를 모니터링하고 특정 임계값에 도달하면 사용자에게 알릴 수 있습니다. 또한 매우 내구력 있는 스토리지에 로그 데이터를 저장할 수 있습니다. 자세한 내용은 Amazon CloudWatch Logs 사용 설명서를 참조하십시오.
- AWS CloudTrail은 직접 수행하거나 AWS 계정를 대신하여 수행한 API 호출 및 관련 이벤트를 캡처하고 지정한 Amazon S3 버킷에 로그 파일을 전송합니다. 어떤 사용자 및 계정이 AWS를 호출했는지 어떤 소스 IP 주소에 호출이 이루어졌는지 언제 호출이 발생했는지 확인할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서를 참조하세요.

Amazon EventBridge: 애플리케이션을 다양한 소스의 데이터와 쉽게 연결할 수 있는 서버리스 이벤트 버스 서비스입니다. EventBridge는 자체 애플리케이션, 서비스형 소프트웨어(SaaS) 애플리케이션 및

AWS 서비스의 실시간 데이터 스트림을 제공한 다음, 해당 데이터를 Lambda 등의 대상으로 라우팅합니다. 이를 통해 서비스에서 발생하는 이벤트를 모니터링하고 이벤트 기반 아키텍처를 구축할 수 있습니다. 자세한 내용은 Amazon EventBridge 사용 설명서를 참조하세요.

주제

- 아마존을 통한 아마존 키스페이스 모니터링 CloudWatch
- 를 사용하여 Amazon Keyspaces API 호출을 로깅합니다 AWS CloudTrail

아마존을 통한 아마존 키스페이스 모니터링 CloudWatch

원시 데이터를 수집하여 읽기 쉬운 거의 실시간 지표로 처리하는 Amazon을 사용하여 Amazon CloudWatch Keyspaces를 모니터링할 수 있습니다. 이러한 통계는 15개월간 보관되므로 기록 정보에 액세스하고 웹 애플리케이션 또는 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다.

특정 임계값을 주시하다가 해당 임계값이 충족될 때 알림을 전송하거나 조치를 취하도록 경보를 설정할 수도 있습니다. 자세한 내용은 Amazon CloudWatch 사용 설명서를 참조하십시오.

> **Note**
>
> Amazon Keyspace의 공통 지표를 보여주는 사전 구성된 CloudWatch 대시보드로 빠르게 시작하려면 에서 제공되는 템플릿을 사용할 AWS CloudFormation 수 있습니다. https://github.com/aws-samples/amazon-keyspaces-cloudwatch-cloudformation-templates

주제

- Amazon Keyspaces 지표를 사용하려면 어떻게 해야 하나요?
- Amazon Keyspaces 지표 및 차원
- Amazon CloudWatch Keyspace를 모니터링하기 위한 경보 생성

Amazon Keyspaces 지표를 사용하려면 어떻게 해야 하나요?

Amazon Keyspaces에서 보고하는 지표는 다양한 방법으로 분석이 기능한 정보를 제공합니다. 다음 목록은 몇 가지 일반적인 지표 사용 사례를 보여 줍니다. 모든 사용 사례를 망라한 것은 아니지만 시작하는 데 참고가 될 것입니다. 지표 및 보존 기간에 대한 자세한 내용은 지표를 참조하세요.

사용 방법	관련 지표
시스템 오류가 발생했는지 어떻게 확인할 수 있나요?	SystemErrors 를 모니터링하여 서버 오류 코드가 발생한 요청이 있는지 확인할 수 있습니다. 일반적으로 이 지표는 0이어야 합니다. 그렇지 않다면 조사가 필요합니다.
프로비저닝된 평균 읽기와 사용된 읽기 용량을 비교하려면 어떻게 해야 하나요?	프로비저닝된 평균 읽기 용량과 사용된 읽기 용량을 모니터링하려면 1. ConsumedReadCapacityUnits 및 ProvisionedReadCapacityUnits 의 기간을 모니터링하려는 간격으로 설정합니다. 2. ConsumedReadCapacityUnits 의 통계를 Average에서 Sum으로 변경합니다. 3. 빈 수학 표현식을 새로 생성합니다. 4. 새 수학 표현식의 세부 정보 섹션에서 지표의 ID를 입력하고 측정치를 지표의 **ConsumedReadCapacityUnits** CloudWatch 기간 함수 (metric_id/ (PERIOD (metric_id))) 로 나눕니다. 5. ConsumedReadCapacityUnits 의 선택을 취소합니다. 이제 평균 소비 읽기 용량을 프로비저닝된 용량과 비교할 수 있습니다. 기본 산술 함수 및 시계열 생성 방법에 대한 자세한 내용은 지표 수학 사용을 참조하세요.
프로비저닝된 평균 쓰기와 사용된 쓰기 용량을 비교하려면 어떻게 해야 하나요?	프로비저닝된 평균 쓰기 용량과 사용된 쓰기 용량을 모니터링하려면 1. ConsumedWriteCapacityUnits 및 ProvisionedWriteCapacityUnits 의 기간을 모니터링하려는 간격으로 설정합니다. 2. ConsumedWriteCapacityUnits 의 통계를 Average에서 Sum으로 변경합니다. 3. 빈 수학 표현식을 새로 생성합니다. 4. 새 수학 표현식의 세부 정보 섹션에 지표를 입력하고 지표를 지표의 CloudWatch 기간 함수 (metric_id/ (PERIOD

사용 방법	관련 지표
	(metric_id)) 로 나눕니다. **ConsumedWriteCapacityUnits** 5. ConsumedWriteCapacityUnits 의 선택을 취소합니다. 이제 평균 소비 쓰기 용량을 프로비저닝된 용량과 비교할 수 있습니다. 기본 산술 함수 및 시계열 생성 방법에 대한 자세한 내용은 지표 수학 사용을 참조하세요.

Amazon Keyspaces 지표 및 차원

Amazon Keyspace와 상호 작용하면 다음 지표와 차원이 Amazon으로 전송됩니다. CloudWatch 모든 지표는 집계되며 1분마다 보고됩니다. 다음 절차를 사용하여 Amazon Keyspaces에 대한 지표를 볼 수 있습니다.

콘솔을 CloudWatch 사용하여 지표를 보려면

지표는 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, 지역을 변경합니다. 탐색 모음에서 AWS 리소스가 상주하는 리전을 선택합니다. 자세한 내용은 AWS 서비스 엔드포인트를 참조하세요.
3. 탐색 창에서 지표(Metrics)를 선택합니다.
4. 모든 지표 탭에서 AWS/Cassandra.를 선택합니다.

AWS CLI를 사용하여 지표를 보려면

- 명령 프롬프트에서 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace "AWS/Cassandra"
```

Amazon Keyspaces 지표 및 차원

Amazon Keyspace가 Amazon에 전송하는 지표 및 측정기준은 CloudWatch 다음과 같습니다.

Amazon Keyspaces 지표

아마존은 CloudWatch 1분 간격으로 아마존 키스페이스 지표를 집계합니다.

Average 또는 Sum과 같은 모든 지표에 적용되지 않는 통계도 있습니다. 하지만 이러한 모든 값은 Amazon Keyspaces 콘솔을 통해, 또는 모든 지표에 대한 콘솔 또는 AWS SDK를 사용하여 사용할 수 있습니다. CloudWatch AWS CLI다음 테이블에는 각 지표마다 적용되는 유효한 통계 목록이 있습니다.

지표	설명
AccountMaxTableLevelReads	계정의 테이블에서 사용할 수 있는 읽기 용량 단위의 최대 수입니다. 온디맨드 테이블의 경우 이 제한이 테이블에서 사용할 수 있는 읽기 요청 단위의 최대 수에 영향을 미칩니다. 단위: Count 유효한 통계: • Maximum - 계정의 테이블에서 사용할 수 있는 읽기 용량 단위의 최대 수입니다.
AccountMaxTableLevelWrites	계정의 테이블에서 사용할 수 있는 쓰기 용량 단위의 최대 수입니다. 온디맨드 테이블의 경우 이 제한이 테이블에서 사용할 수 있는 쓰기 요청 단위의 최대 수에 영향을 미칩니다. 단위: Count 유효한 통계: • Maximum - 계정의 테이블에서 사용할 수 있는 쓰기 용량 단위의 최대 수입니다.
AccountProvisionedReadCapacityUtilization	계정에서 사용하는 프로비저닝된 읽기 용량 단위의 백분율입니다. 단위: Percent 유효한 통계:

지표	설명
	• Maximum - 계정에서 사용하는 프로비저닝된 읽기 용량 단위의 최대 백분율입니다. • Minimum - 계정에서 사용하는 프로비저닝된 읽기 용량 단위의 최소 백분율입니다. • Average - 계정에서 사용하는 프로비저닝된 읽기 용량 단위의 평균 백분율입니다. 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 읽기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.
AccountProvisionedWriteCapacityUtilization	계정에서 사용하는 프로비저닝된 쓰기 용량 단위의 백분율입니다. 단위: Percent 유효한 통계: • Maximum - 계정에서 사용하는 프로비저닝된 쓰기 용량 단위의 최대 백분율입니다. • Minimum - 계정에서 사용하는 프로비저닝된 쓰기 용량 단위의 최소 백분율입니다. • Average - 계정에서 사용하는 프로비저닝된 쓰기 용량 단위의 평균 백분율입니다. 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 쓰기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

지표	설명
BillableTableSizeInBytes	테이블의 청구 가능 크기(바이트)입니다. 테이블에 있는 모든 행의 인코딩된 크기의 합계입니다. 이 지표를 통해 시간 경과에 따른 테이블 스토리지 비용을 추적할 수 있습니다. 단위: Bytes 차원: Keyspace, TableName 유효한 통계: • Maximum - 테이블의 최대 스토리지 크기입니다. • Minimum - 테이블의 최소 스토리지 크기입니다. • Average - 테이블의 평균 스토리지 크기입니다. 이 지표는 4~6시간 간격으로 계산됩니다.
ConditionalCheckFailedRequests	실패한 간단한 트랜잭션(LWT) 쓰기 요청 수입니다. INSERT, UPDATE 및 DELETE 작업에서는 해당 작업이 진행하려면 먼저 true로 평가되어야 하는 논리적 조건을 사용자가 제공하도록 합니다. 이 조건에서 false로 평가되면 ConditionalCheckFailedRequests 가 1씩 증분됩니다. 허위로 평가되는 조건 검사는 행 크기를 기준으로 쓰기 용량 단위를 소비합니다. 자세한 정보는 the section called “간단한 트랜잭션”을 참조하세요. 단위: Count 차원: Keyspace, TableName 유효한 통계: • Minimum • Maximum • Average • SampleCount • Sum

지표	설명
ConsumedReadCapacityUnits	지정된 기간 동안 사용된 읽기 용량 단위의 수입니다. 자세한 내용은 읽기/쓰기 용량 모드를 참조하세요. **Note** 초당 평균 처리량 사용률을 이해하려면 Sum 통계를 사용하여 1분 동안 소비된 처리량을 계산합니다. 그런 다음 합계를 분당 초 수(60)로 나누어 초당 평균 ConsumedReadCapacityUnits 를 계산합니다(이 평균은 해당 분 동안 발생한 읽기 활동의 크고 짧은 급증을 강조하지 않는다는 점을 인식). 평균 소비 읽기 용량과 프로비저닝된 읽기 용량을 비교하는 방법에 대한 자세한 내용은 the section called “지표 사용” 섹션을 참조하세요. 단위: Count 차원: Keyspace, TableName 유효한 통계: • Minimum - 테이블에 대한 개별 요청에 의해 사용된 읽기 용량 단위의 최소 수입니다. • Maximum - 테이블에 대한 개별 요청에 의해 사용된 읽기 용량 단위의 최대 수입니다. • Average - 사용된 요청당 읽기 용량 평균입니다. **Note** Average 값은 샘플 값이 0이 될 비활동 기간의 영향을 받습니다. • Sum - 사용된 총 읽기 용량 단위입니다. ConsumedReadCapacityUnits 지표에 가장 유용한 통계입니다.

지표	설명
	• `SampleCount` - Amazon Keyspaces에 대한 요청 수입니다(읽기 용량이 사용되지 않은 경우도 해당). **Note** `SampleCount` 값은 샘플 값이 0이 될 비활동 기간의 영향을 받습니다.

지표	설명
ConsumedWriteCapacityUnits	지정된 기간 동안 사용된 쓰기 용량 단위의 수입니다. 테이블에 대해 소비된 총 쓰기 용량을 검색할 수 있습니다. 자세한 내용은 읽기/쓰기 용량 모드를 참조하세요. **Note** 초당 평균 처리량 사용률을 이해하려면 Sum 통계를 사용하여 1분 동안 소비된 처리량을 계산합니다. 그런 다음 합계를 분당 초 수(60)로 나누어 초당 평균 ConsumedWriteCapacityUnits 를 계산합니다(이 평균은 해당 분 동안 발생한 쓰기 활동의 크고 짧은 급증을 강조하지 않는다는 점을 인식). 평균 소비 쓰기 용량과 프로비저닝된 쓰기 용량을 비교하는 방법에 대한 자세한 내용은 the section called “지표 사용” 섹션을 참조하세요. 단위: Count 차원: Keyspace, TableName 유효한 통계: • Minimum - 테이블에 대한 개별 요청에 의해 사용된 쓰기 용량 단위의 최소 수입니다. • Maximum - 테이블에 대한 개별 요청에 의해 사용된 쓰기 용량 단위의 최대 수입니다. • Average - 사용된 요청당 쓰기 용량 평균입니다. **Note** Average 값은 샘플 값이 0이 될 비활동 기간의 영향을 받습니다.

지표	설명
	• Sum - 사용된 총 쓰기 용량 단위입니다. ConsumedWriteCapacityUnits 지표에 가장 유용한 통계입니다. • SampleCount - Amazon Keyspaces에 대한 요청 수입니다(쓰기 용량이 사용되지 않은 경우도 해당). Note SampleCount 값은 샘플 값이 0이 될 비활동 기간의 영향을 받습니다.
MaxProvisionedTableReadCapacityUtilization	계정의 가장 높은 프로비저닝된 읽기 테이블에서 사용하는 프로비저닝된 읽기 용량 단위의 백분율입니다. 단위: Percent 유효한 통계: • Maximum - 계정의 가장 높은 프로비저닝된 읽기 테이블에서 사용하는 프로비저닝된 읽기 용량 단위의 최대 백분율입니다. • Minimum - 계정의 가장 높은 프로비저닝된 읽기 테이블에서 사용하는 프로비저닝된 읽기 용량 단위의 최소 백분율입니다. • Average - 계정의 가장 높은 프로비저닝된 읽기 테이블에서 사용하는 프로비저닝된 읽기 용량 유닛의 평균 백분율입니다. 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 읽기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

지표	설명
MaxProvisionedTableWriteCapacityUtilization	계정의 가장 높은 프로비저닝된 쓰기 테이블에서 사용하는 프로비저닝된 쓰기 용량의 백분율입니다. 단위: Percent 유효한 통계: • Maximum - 계정의 가장 높은 프로비저닝된 쓰기 테이블에서 사용하는 프로비저닝된 쓰기 용량 단위의 최대 백분율입니다. • Minimum - 계정의 가장 높은 프로비저닝된 쓰기 테이블에서 사용하는 프로비저닝된 쓰기 용량 단위의 최소 백분율입니다. • Average - 계정의 가장 높은 프로비저닝된 쓰기 테이블에서 사용하는 프로비저닝된 쓰기 용량 단위의 평균 백분율입니다. 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 쓰기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

지표	설명
PerConnectionRequestRateExceeded	연결당 요청 속도 할당량을 초과하는 Amazon Keyspaces에 대한 요청입니다. Amazon Keyspaces에 대한 각 클라이언트 연결은 초당 최대 3000개의 CQL 요청을 지원할 수 있습니다. 클라이언트는 여러 연결을 생성하여 처리량을 늘릴 수 있습니다. 다중 리전 복제를 사용하는 경우 복제된 각 쓰기 작업도 이 할당량에 포함됩니다. PerConnectionRequestRateExceeded 오류가 발생하지 않도록 테이블에 대한 연결 수를 늘리는 것이 가장 좋습니다. Amazon Keyspaces에서 사용할 수 있는 연결 수에는 제한이 없습니다. 단위: Count 차원: Keyspace, TableName, Operation 유효한 통계: • SampleCount • Sum

지표	설명
ProvisionedReadCapacityUnits	테이블에 대해 프로비저닝된 읽기 용량 단위의 수입니다. TableName 차원은 테이블에 대한 ProvisionedReadCapacityUnits 를 반환합니다. 단위: Count 차원: Keyspace, TableName 유효한 통계: • Minimum - 프로비저닝된 읽기 용량에 대한 가장 낮은 설정입니다. ALTER TABLE을 사용하여 읽기 용량을 늘리는 경우 이 지표는 해당 기간 동안 프로비저닝된 ReadCapacityUnits 의 최저값을 보여 줍니다. • Maximum - 프로비저닝된 읽기 용량에 대한 가장 높은 설정입니다. ALTER TABLE을 사용하여 읽기 용량을 줄이는 경우 이 지표는 해당 기간 동안 프로비저닝된 ReadCapacityUnits 의 최고값을 보여 줍니다. • Average - 프로비저닝된 읽기 용량 평균입니다. ProvisionedReadCapacityUnits 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 읽기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.

지표	설명
ProvisionedWriteCapacityUnits	테이블에 대해 프로비저닝된 쓰기 용량 단위의 수입니다. TableName 차원은 테이블에 대한 ProvisionedWriteCapacityUnits 를 반환합니다. 단위: Count 차원: Keyspace, TableName 유효한 통계: • Minimum - 프로비저닝된 쓰기 용량에 대한 가장 낮은 설정입니다. ALTER TABLE을 사용하여 쓰기 용량을 늘리는 경우 이 지표는 해당 기간 동안 프로비저닝된 WriteCapacityUnits 의 최저값을 보여 줍니다. • Maximum - 프로비저닝된 쓰기 용량에 대한 가장 높은 설정입니다. ALTER TABLE을 사용하여 쓰기 용량을 줄이는 경우 이 지표는 해당 기간 동안 프로비저닝된 WriteCapacityUnits 의 최고값을 보여 줍니다. • Average - 프로비저닝된 쓰기 용량 평균입니다. ProvisionedWriteCapacityUnits 지표는 5분 간격으로 게시됩니다. 따라서 프로비저닝된 쓰기 용량 단위를 빠르게 조정하는 경우 이 통계는 정확한 평균값을 나타내지 않을 수도 있습니다.
ReadThrottleEvents	테이블에 대해 프로비저닝된 읽기 용량을 초과하는 Amazon Keyspace에 대한 요청, 계정 수준 할당량, 연결당 요청 할당량 또는 파티션 수준 할당량 단위: Count 차원: Keyspace, TableName, Operation 유효한 통계: • SampleCount • Sum

지표	설명
ReplicationLatency	이 지표는 다중 리전 키스페이스에만 적용되며 다중 리전 키스페이스의 한 복제 테이블에서 다른 복제 테이블로 updates, inserts 또는 deletes를 복제하는 데 걸린 시간을 측정합니다. 단위: Millisecond 차원: TableName, ReceivingRegion 유효한 통계: • Average • Maximum • Minimum

지표	설명
ReturnedItemCountBySelect	지정된 기간 동안 다중 행 SELECT 쿼리에 의해 반환되는 행 수입니다. 다중 행 SELECT 쿼리는 전체 테이블 스캔 및 범위 쿼리와 같이 정규화된 프라이머리 키를 포함하지 않는 쿼리입니다. 반환되는 행 수가 평가된 행 수와 반드시 일치하지는 않습니다. 예를 들어 100개의 행이 있는 테이블에 대해 ALLOW FILTERING 으로 SELECT *를 요청했지만 15개의 행만 반환되도록 결과를 좁히는 WHERE 절을 지정했다고 가정해 보겠습니다. 이 경우 SELECT의 응답에 100개 ScanCount 및 15개 Count의 반환된 행이 포함됩니다. 단위: Count 차원: Keyspace, TableName, Operation 유효한 통계: • Minimum • Maximum • Average • SampleCount • Sum

지표	설명
StoragePartitionThroughputCapacityExceeded	파티션의 처리량 용량을 초과하는 Amazon Keyspaces 스토리지 파티션에 대한 요청입니다. Amazon Keyspaces 스토리지 파티션은 초당 최대 1000 WCU/WRU 및 초당 3000 RCU/RRU를 지원할 수 있습니다. 이러한 예외를 완화하려면 데이터 모델을 검토하여 더 많은 파티션에 읽기/쓰기 트래픽을 분산하는 것이 좋습니다. **Note** 논리적 Amazon Keyspaces 파티션은 여러 스토리지 파티션에 걸쳐 있을 수 있으며 사실상 크기에 제한이 없습니다. 단위: Count 차원: Keyspace, TableName, Operation 유효한 통계: • SampleCount • Sum
SuccessfulRequestCount	지정한 시간 동안 성공적으로 처리된 요청 수입니다. 단위: Count 차원: Keyspace, TableName, Operation 유효한 통계: • SampleCount

지표	설명
SuccessfulRequestLatency	지정된 기간 동안 Amazon Keyspaces에 대한 성공한 요청입니다. SuccessfulRequestLatency 는 다음과 같이 두 가지 종류의 정보를 제공할 수 있습니다. • 성공한 요청의 경과 시간(Minimum, Maximum, Sum 또는 Average) • 성공한 요청의 수(SampleCount) SuccessfulRequestLatency 는 Amazon Keyspaces 내의 활동만 반영하며 네트워크 지연 시간이나 클라이언트 측 활동은 고려하지 않습니다. 단위: Milliseconds 차원: Keyspace, TableName, Operation 유효한 통계: • Minimum • Maximum • Average • SampleCount
SystemErrors	지정된 기간 동안 ServerError 를 생성하는 Amazon Keyspace에 대한 요청입니다. ServerError 는 일반적으로 내부 서비스 오류를 나타냅니다. 단위: Count 차원: Keyspace, TableName, Operation 유효한 통계: • Sum • SampleCount

지표	설명
SystemReconciliationDeletes	클라이언트 측 타임스탬프가 활성화된 경우 삭제된 데이터를 삭제하는 데 사용된 단위입니다. 각 SystemReconciliationDelete 는 행당 최대 1KB의 데이터를 삭제하거나 업데이트할 수 있는 충분한 용량을 제공합니다. 예를 들어 2.5KB의 데이터를 저장하는 행을 업데이트하고 행 내에서 하나 이상의 열을 동시에 삭제하려면 3 SystemReconciliationDeletes 가 필요합니다. 또는 3.5KB의 데이터가 포함된 행 전체를 삭제하려면 4 SystemReconciliationDeletes 가 필요합니다. 단위: Count 차원: Keyspace, TableName 유효한 통계: • Sum - 특정 기간 동안 소비된 SystemReconciliationDeletes 의 총 수입니다.
TTLDeletes	TTL(Time To Live)을 사용하여 행의 데이터를 삭제하거나 업데이트하는 데 사용되는 단위입니다. 각 TTLDelete 는 행당 최대 1KB의 데이터를 삭제하거나 업데이트할 수 있는 충분한 용량을 제공합니다. 예를 들어 2.5KB의 데이터를 저장하는 행을 업데이트하고 행 내에서 하나 이상의 열을 동시에 삭제하려면 3 TTL deletes가 필요합니다. 또는 3.5KB의 데이터가 포함된 행 전체를 삭제하려면 4 TTL deletes가 필요합니다. 단위: Count 차원: Keyspace, TableName 유효한 통계: • Sum - 특정 기간 동안 소비된 TTLDeletes 의 총 수입니다.

지표	설명
UserErrors	지정된 기간 동안 InvalidRequest 오류를 생성하는 Amazon Keyspace에 대한 요청입니다. InvalidRequest 는 유효하지 않은 파라미터 조합, 존재하지 않는 테이블을 업데이트하려는 시도, 잘못된 요청 서명 등 일반적으로 클라이언트 측 오류를 나타냅니다. UserErrors 현재 AWS 리전 요청과 현재 요청에 대한 유효하지 않은 요청의 총계를 나타냅니다. AWS 계정 단위: Count 차원: Keyspace, TableName, Operation 유효한 통계: • Sum • SampleCount
WriteThrottleEvents	테이블에 대해 프로비저닝된 쓰기 용량을 초과하는 Amazon Keyspace에 대한 요청, 계정 수준 할당량, 연결당 요청 할당량 또는 파티션 수준 할당량 단위: Count 차원: Keyspace, TableName, Operation 유효한 통계: • SampleCount • Sum

Amazon Keyspaces 지표에 대한 차원

Amazon Keyspaces에 대한 지표는 계정, 테이블 이름 또는 작업의 값으로 한정됩니다. CloudWatch 콘솔을 사용하여 다음 표의 모든 차원을 따라 Amazon Keyspaces 데이터를 검색할 수 있습니다.

측정기준	설명
Keyspace	이 차원은 특정 키스페이스에 대한 데이터를 제한합니다. 이 값은 현재 리전 및 현재 AWS 계정의 키스페이스일 수 있습니다.
Operation	이 차원은 INSERT 또는 SELECT 작업과 같은 Amazon Keyspaces CQL 작업 중 하나에 대한 데이터를 제한합니다.
TableName	이 차원은 특정 테이블에 대한 데이터를 제한합니다. 이 값은 현재 리전 및 현재 AWS 계정의 테이블 이름일 수 있습니다. 테이블 이름이 계정 내에서 고유하지 않은 경우 Keyspace도 지정해야 합니다.

Amazon CloudWatch Keyspace를 모니터링하기 위한 경보 생성

CloudWatch 알람 상태가 변경될 때 Amazon Simple Notification Service (Amazon SNS) 메시지를 보내는 Amazon Keyspaces용 아마존 경보를 생성할 수 있습니다. 경보는 지정한 기간 동안 단일 지표를 감시합니다. 기간 수에 대한 주어진 임계값과 지표 값을 비교하여 하나 이상의 작업을 수행합니다. 이 작업은 Amazon SNS 주제 또는 Auto Scaling 정책에 전송되는 알림입니다.

Application Auto Scaling과 함께 프로비저닝 모드에서 Amazon Keyspace를 사용하면 서비스가 사용자를 대신하여 두 쌍의 CloudWatch 경보를 생성합니다. 각 쌍은 프로비저닝되고 사용된 처리량 설정의 상한값과 하한값을 나타냅니다. 이러한 CloudWatch 경보는 테이블의 실제 사용률이 일정 기간 동안 목표 사용률에서 벗어날 때 트리거됩니다. Application Auto Scaling에서 생성된 CloudWatch 경보에 대한 자세한 내용은 을 참조하십시오the section called “Amazon Keyspaces Auto Scaling 작동 방식”.

경보는 지속적인 상태 변경에 대한 조치만 호출합니다. CloudWatch 경보는 단순히 특정 상태에 있다는 이유만으로 조치를 호출하지 않습니다. 상태가 변경되어 지정된 기간 수 동안 유지되어야 합니다.

CloudWatch 경보 생성에 대한 자세한 내용은 Amazon 사용 CloudWatch 설명서의 Amazon CloudWatch 경보 사용을 참조하십시오.

를 사용하여 Amazon Keyspaces API 호출을 로깅합니다 AWS CloudTrail

Amazon Keyspaces는 Amazon Keyspaces에서 사용자, 역할 또는 서비스가 수행한 작업의 기록을 제공하는 AWS 서비스와 통합되어 있습니다. AWS CloudTrail CloudTrail Amazon Keyspaces에 대한 데

이터 정의 언어 (DDL) API 호출 및 데이터 조작 언어 (DML) API 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 Amazon Keyspaces 콘솔에서 수행한 직접 호출과 Amazon Keyspaces API 작업에 대한 프로그램 방식의 직접 호출이 포함됩니다.

트레일을 생성하면 Amazon Keyspace에 대한 CloudTrail 이벤트를 포함하여 Amazon Simple Storage Service (Amazon S3) 버킷으로 이벤트를 지속적으로 전송할 수 있습니다.

트레일을 구성하지 않아도 CloudTrail 콘솔의 이벤트 기록에서 가장 최근에 지원되는 이벤트를 계속 볼 수 있습니다. 에서 수집한 CloudTrail 정보를 사용하여 Amazon Keyspaces에 이루어진 요청, 요청이 이루어진 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

자세한 CloudTrail 내용은 AWS CloudTrail 사용 설명서를 참조하십시오.

주제

- 에서 Amazon Keyspaces 로그 파일 항목 구성 CloudTrail
- Amazon Keyspace의 DDL (데이터 정의 언어) 정보 CloudTrail
- Amazon Keyspace의 데이터 조작 언어 (DML) 정보 CloudTrail
- Amazon Keyspaces 로그 파일 항목 이해

에서 Amazon Keyspaces 로그 파일 항목 구성 CloudTrail

로그인된 각 Amazon Keyspaces API 작업에는 CQL 쿼리 언어로 표현되는 요청 파라미터가 CloudTrail 포함되어 있습니다. 자세한 내용은 CQL 언어 참조을 참조하세요.

AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 이벤트 기록으로 CloudTrail 이벤트 보기를 참조하십시오.

Amazon Keyspace의 이벤트를 AWS 계정포함하여 귀하의 이벤트에 대한 지속적인 기록을 보려면 트레일을 생성하십시오. 트레일을 사용하면 CloudTrail Amazon S3 버킷에 로그 파일을 전송할 수 있습니다. 기본적으로 콘솔에서 트레일을 생성하면 트레일이 모든 AWS 지역에 적용됩니다. 트레일은 AWS 파티션에 있는 모든 지역의 이벤트를 기록하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다.

자세한 내용은AWS CloudTrail 사용 설명서에서 다음 주제를 참조하세요.

- 추적 생성 개요
- CloudTrail 지원되는 서비스 및 통합

- 에 대한 Amazon SNS 알림 구성 CloudTrail
- 여러 지역에서 CloudTrail 로그 파일 수신
- 여러 계정에서 CloudTrail 로그 파일 받기

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 신원 정보를 이용하면 다음을 쉽게 알아볼 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부
- 역할 또는 연동 사용자를 위한 임시 보안 인증으로 요청을 생성하였는지.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 CloudTrail UserIdentity 요소를 참조하십시오.

Amazon Keyspace의 DDL (데이터 정의 언어) 정보 CloudTrail

CloudTrail 계정을 만들 AWS 계정 때 활성화됩니다. Amazon Keyspace에서 DDL 활동이 발생하면 해당 활동은 이벤트 기록에 AWS 다른 서비스 이벤트와 함께 이벤트로 자동 기록됩니다. CloudTrail 다음 표는 Amazon Keyspaces에 대해 기록된 DDL 문을 보여 줍니다.

CloudTrail **eventName**	문	CQL 작업	AWS SDK 작업
CreateKeyspace	DDL	`CREATE KEYSPACE`	`CreateKeyspace`
DropKeyspace	DDL	`DROP KEYSPACE`	`DeleteKeyspace`
CreateTable	DDL	`CREATE TABLE`	`CreateTable`
DropTable	DDL	`DROP TABLE`	`DeleteTable`
AlterTable	DDL	`ALTER TABLE`	`UpdateTable` , `TagResource` , `UntagResource`

Amazon Keyspace의 데이터 조작 언어 (DML) 정보 CloudTrail

에서 Amazon Keyspaces DML 문의 로깅을 활성화하려면 먼저 데이터 플레인 API 활동 로깅을 활성화해야 합니다. CloudTrail CloudTrail CloudTrail 콘솔을 사용하여 데이터 이벤트 유형 Cassandra 테이블에 대한 활동을 기록하도록 선택하거나 CLI 또는 API 작업을 사용하여 값을 설정함으로써 신규 또는 기존 트레일에서 Amazon Keyspaces DML 이벤트 `resources.type` 로깅을 시작할 수 있습니다. `AWS::Cassandra::Table` AWS CloudTrail 자세한 내용은 데이터 이벤트 로깅 섹션을 참조하세요.

다음 표에는 에 의해 기록된 데이터 이벤트가 나와 있습니다. CloudTrail `Cassandra table`

CloudTrail **`eventName`**	문	CQL 작업	AWS SDK 작업
Select	DML	`SELECT`	`GetKeyspace` , `GetTable`, `ListKeyspaces` , `ListTables` `ListTagsForResource`
Insert	DML	`INSERT`	AWS SDK 액션을 사용할 수 없음
업데이트	DML	`UPDATE`	AWS SDK 액션을 사용할 수 없음
삭제	DML	`DELETE`	AWS SDK 액션을 사용할 수 없음

Amazon Keyspaces 로그 파일 항목 이해

CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함되어 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜 및 시간, 요청 매개 변수 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는`CreateKeyspace`, `DropKeyspaceCreateTable`, 및 `DropTable` 작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

```
{
  "Records": [
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
        "accountId": "111122223333",
        "sessionContext": {
          "sessionIssuer": {
            "type": "Role",
            "principalId": "AKIAIOSFODNN7EXAMPLE",
            "arn": "arn:aws:iam::111122223333:role/Admin",
            "accountId": "111122223333",
            "userName": "Admin"
          },
          "webIdFederationData": {},
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2020-01-15T18:47:56Z"
          }
        }
      },
      "eventTime": "2020-01-15T18:53:04Z",
      "eventSource": "cassandra.amazonaws.com",
      "eventName": "CreateKeyspace",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "10.24.34.01",
      "userAgent": "Cassandra Client/ProtocolV4",
      "requestParameters": {
        "rawQuery": "\n\tCREATE KEYSPACE \"mykeyspace\"\n\tWITH\n\t\tREPLICATION =
 {'class': 'SingleRegionStrategy'}\n\t\t",
        "keyspaceName": "mykeyspace"
      },
      "responseElements": null,
      "requestID": "bfa3e75d-bf4d-4fc0-be5e-89d15850eb41",
      "eventID": "d25beae8-f611-4229-877a-921557a07bb9",
      "readOnly": false,
      "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Keyspace",
```

```
        "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/"
      }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "3.4.4",
    "recipientAccountId": "111122223333",
    "managementEvent": true,
    "eventCategory": "Management",
    "tlsDetails": {
        "tlsVersion": "TLSv1.2",
        "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
  },
  {
    "eventVersion": "1.05",
    "userIdentity": {
      "type": "AssumedRole",
      "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
      "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
      "accountId": "111122223333",
      "sessionContext": {
        "sessionIssuer": {
          "type": "Role",
          "principalId": "AKIAIOSFODNN7EXAMPLE",
          "arn": "arn:aws:iam::111122223333:role/Admin",
          "accountId": "111122223333",
          "userName": "Admin"
        },
        "webIdFederationData": {},
        "attributes": {
          "mfaAuthenticated": "false",
          "creationDate": "2020-01-15T18:47:56Z"
        }
      }
    },
    "eventTime": "2020-01-15T19:28:39Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "DropKeyspace",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
      "rawQuery": "DROP KEYSPACE \"mykeyspace\"",
      "keyspaceName": "mykeyspace"
```

```
      },
      "responseElements": null,
      "requestID": "66f3d86a-56ae-4c29-b46f-abcd489ed86b",
      "eventID": "e5aebeac-e1dd-41e3-a515-84fe6aaabd7b",
      "readOnly": false,
      "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Keyspace",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/"
        }
      ],
      "eventType": "AwsApiCall",
      "apiVersion": "3.4.4",
      "recipientAccountId": "111122223333",
      "managementEvent": true,
      "eventCategory": "Management",
      "tlsDetails": {
          "tlsVersion": "TLSv1.2",
          "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
          "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    },
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
        "accountId": "111122223333",
        "sessionContext": {
          "sessionIssuer": {
            "type": "Role",
            "principalId": "AKIAIOSFODNN7EXAMPLE",
            "arn": "arn:aws:iam::111122223333:role/Admin",
            "accountId": "111122223333",
            "userName": "Admin"
          },
          "webIdFederationData": {},
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2020-01-15T18:47:56Z"
          }
        }
      },
```

```
      "eventTime": "2020-01-15T18:55:24Z",
      "eventSource": "cassandra.amazonaws.com",
      "eventName": "CreateTable",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "10.24.34.01",
      "userAgent": "Cassandra Client/ProtocolV4",
      "requestParameters": {
        "rawQuery": "\n\tCREATE TABLE \"mykeyspace\".\"mytable\"(\n\t\t\"ID\" int,
\n\t\t\"username\" text,\n\t\t\"email\" text,\n\t\t\"post_type\" text,\n\t\tPRIMARY
 KEY((\"ID\", \"username\", \"email\")))",
        "keyspaceName": "mykeyspace",
        "tableName": "mytable"
      },
      "responseElements": null,
      "requestID": "5f845963-70ea-4988-8a7a-2e66d061aacb",
      "eventID": "fe0dbd2b-7b34-4675-a30c-740f9d8d73f9",
      "readOnly": false,
      "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Table",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable"
        }
      ],
      "eventType": "AwsApiCall",
      "apiVersion": "3.4.4",
      "recipientAccountId": "111122223333",
      "managementEvent": true,
      "eventCategory": "Management",
      "tlsDetails": {
          "tlsVersion": "TLSv1.2",
          "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
          "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    },
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AKIAIOSFODNN7EXAMPLE:alice",
        "arn": "arn:aws:sts::111122223333:assumed-role/users/alice",
        "accountId": "111122223333",
        "sessionContext": {
          "sessionIssuer": {
```

```
            "type": "Role",
            "principalId": "AKIAIOSFODNN7EXAMPLE",
            "arn": "arn:aws:iam::111122223333:role/Admin",
            "accountId": "111122223333",
            "userName": "Admin"
          },
          "webIdFederationData": {},
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2020-01-15T18:47:56Z"
          }
        }
      },
      "eventTime": "2020-01-15T19:27:59Z",
      "eventSource": "cassandra.amazonaws.com",
      "eventName": "DropTable",
      "awsRegion": "us-east-1",
      "sourceIPAddress": "10.24.34.01",
      "userAgent": "Cassandra Client/ProtocolV4",
      "requestParameters": {
        "rawQuery": "DROP TABLE \"mykeyspace\".\"mytable\"",
        "keyspaceName": "mykeyspace",
        "tableName": "mytable"
      },
      "responseElements": null,
      "requestID": "025501b0-3582-437e-9d18-8939e9ef262f",
      "eventID": "1a5cbedc-4e38-4889-8475-3eab98de0ffd",
      "readOnly": false,
      "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Table",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable"
        }
      ],
      "eventType": "AwsApiCall",
      "apiVersion": "3.4.4",
      "recipientAccountId": "111122223333",
      "managementEvent": true,
      "eventCategory": "Management",
      "tlsDetails": {
          "tlsVersion": "TLSv1.2",
          "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
```

```
            "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
        }
    ]
}
```

다음 로그 파일은 SELECT 문의 예를 보여줍니다.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/alice",
        "accountId": "111122223333",
        "userName": "alice"
    },
    "eventTime": "2023-11-17T10:38:04Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "Select",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
          "keyspaceName": "my_keyspace",
          "tableName": "my_table",
          "conditions": [
              "pk = **(Redacted)",
              "ck < 3**(Redacted)0",
              "region = 't**(Redacted)t'"
          ],
          "select": [
              "pk",
              "ck",
              "region"
          ],
          "allowFiltering": true
    },
    "responseElements": null,
    "requestID": "6d83bbf0-a3d0-4d49-b1d9-e31779a28628",
    "eventID": "e00552d3-34e9-4092-931a-912c4e08ba17",
    "readOnly": true,
    "resources": [
        {
```

```
            "accountId": "111122223333",
            "type": "AWS::Cassandra::Table",
            "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/my_keyspace/
table/my_table"
        }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "3.4.4",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    }
}
```

다음 로그 파일은 INSERT 문의 예를 보여줍니다.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/alice",
        "accountId": "111122223333",
        "userName": "alice"
    },
    "eventTime": "2023-12-01T22:11:43Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "Insert",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
          "keyspaceName": "my_keyspace",
          "tableName": "my_table",
          "primaryKeys": {
              "pk": "**(Redacted)",
              "ck": "1**(Redacted)8"
          },
          "columnNames": [
```

```
                "pk",
                "ck",
                "region"
            ],
            "updateParameters": {
                "TTL": "2**(Redacted)0"
            }
        }
    },
    "responseElements": null,
    "requestID": "edf8af47-2f87-4432-864d-a960ac35e471",
    "eventID": "81b56a1c-9bdd-4c92-bb8e-92776b5a3bf1",
    "readOnly": false,
    "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Table",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/my_keyspace/table/
my_table"
        }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "3.4.4",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    }
}
```

다음 로그 파일은 UPDATE 문의 예를 보여줍니다.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
          "type": "IAMUser",
          "principalId": "AKIAIOSFODNN7EXAMPLE",
          "arn": "arn:aws:iam::111122223333:user/alice",
          "accountId": "111122223333",
          "userName": "alice"
```

```
    },
    "eventTime": "2023-12-01T22:11:43Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "Update",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
          "keyspaceName": "my_keyspace",
          "tableName": "my_table",
          "primaryKeys": {
              "pk": "'t**(Redacted)t'",
              "ck": "'s**(Redacted)g'"
          },
          "assignmentColumnNames": [
             "nonkey"
          ],
          "conditions": [
              "nonkey < 1**(Redacted)7"
          ]
    },
    "responseElements": null,
    "requestID": "edf8af47-2f87-4432-864d-a960ac35e471",
    "eventID": "81b56a1c-9bdd-4c92-bb8e-92776b5a3bf1",
    "readOnly": false,
    "resources": [
        {
          "accountId": "111122223333",
          "type": "AWS::Cassandra::Table",
          "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/my_keyspace/table/
my_table"
        }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "3.4.4",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
          "tlsVersion": "TLSv1.3",
          "cipherSuite": "TLS_AES_128_GCM_SHA256",
          "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    }
```

```
}
```

다음 로그 파일은 DELETE 문의 예를 보여줍니다.

```
{
    "eventVersion": "1.09",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::111122223333:user/alice",
        "accountId": "111122223333",
        "userName": "alice",
    },
    "eventTime": "2023-10-23T13:59:05Z",
    "eventSource": "cassandra.amazonaws.com",
    "eventName": "Delete",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "10.24.34.01",
    "userAgent": "Cassandra Client/ProtocolV4",
    "requestParameters": {
        "keyspaceName": "my_keyspace",
        "tableName": "my_table",
        "primaryKeys": {
            "pk": "**(Redacted)",
            "ck": "**(Redacted)"
        },
        "conditions": [],
        "deleteColumnNames": [
            "m",
            "s"
        ],
        "updateParameters": {}
    },
    "responseElements": null,
    "requestID": "3d45e63b-c0c8-48e2-bc64-31afc5b4f49d",
    "eventID": "499da055-c642-4762-8775-d91757f06512",
    "readOnly": false,
    "resources": [
      {
        "accountId": "111122223333",
        "type": "AWS::Cassandra::Table",
        "ARN": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/my_keyspace/table/
my_table"
```

```
      }
    ],
    "eventType": "AwsApiCall",
    "apiVersion": "3.4.4",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data",
    "tlsDetails": {
        "tlsVersion": "TLSv1.3",
        "cipherSuite": "TLS_AES_128_GCM_SHA256",
        "clientProvidedHostHeader": "cassandra.us-east-1.amazonaws.com"
    }
}
```

Amazon Keyspaces(Apache Cassandra용)의 보안

AWS에서는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 매우 민감한 조직의 요구 사항에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 귀하의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드의 보안 및 클라우드 내 보안으로 설명합니다.

- 클라우드의 보안 – AWS는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. 또한, AWS는 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon Keyspaces에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 제공 범위 내 AWS 서비스를 참조하세요.
- 클라우드 내 보안 – 사용자의 책임은 사용자가 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 Keyspaces 사용 시 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목적에 맞게 Amazon Keyspaces를 구성하는 방법을 보여 줍니다. 또한 Amazon Keyspaces 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스의 사용 방법을 배우게 됩니다.

주제

Amazon Keyspaces의 데이터 보호

AWS 공동 책임 모델은 Amazon Keyspaces(Apache Cassandra용)의 데이터 보호에 적용됩니다. 이 모델이 설명하는 것처럼 AWS는 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습

니다. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 Data Privacy FAQ(데이터 프라이버시 FAQ)를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS Shared Responsibility Model and GDPR 블로그 게시물을 참조하세요.

데이터를 보호하려면 AWS 계정보안 인증 정보를 보호하고 AWS IAM Identity Center또는 AWS Identity and Access Management(IAM)를 통해 개별 사용자 계정을 설정하는 것이 좋습니다. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 멀티 팩터 인증 설정(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신하세요. TLS 1.2는 필수이며 TLS 1.3를 권장합니다.
- AWS CloudTrail로 API 및 사용자 활동 로깅을 설정하세요.
- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 컨트롤과 함께 사용하세요.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-2 인증 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 사용하세요. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 FIPS(Federal Information Processing Standard) 140-2를 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 Name 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 Amazon Keyspaces 또는 기타 AWS 서비스에서 콘솔, API, AWS CLI 또는 AWS SDK를 사용하여 작업하는 경우가 포함됩니다. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버로 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명 정보를 URL에 포함해서는 안 됩니다.

주제

- Amazon Keyspaces에서 저장 시 암호화
- Amazon Keyspaces에서 전송 중 암호화
- Amazon Keyspaces의 인터네트워크 트래픽 개인 정보

Amazon Keyspaces에서 저장 시 암호화

Amazon Keyspaces(Apache Cassandra용) 저장 시 암호화는 AWS Key Management Service(AWS KMS)에 저장된 암호화 키를 사용하여 모든 저장 데이터를 암호화하여 향상된 보안을 제공합니다. 이 기능을 사용하면 중요한 데이터 보호와 관련된 운영 부담 및 복잡성을 줄일 수 있습니다. 저장 시 암호화를 사용하면 엄격한 규정 준수 및 데이터 보호에 대한 규제 요구 사항이 필요한, 보안에 민감한 애플리케이션을 구축할 수 있습니다.

Amazon Keyspaces 저장 시 암호화는 256비트 고급 암호화 표준(AES-256)을 사용하여 데이터를 암호화합니다. 이는 기본 스토리지에 대한 무단 액세스로부터 데이터의 보안을 유지하는 데 도움을 줍니다.

Amazon Keyspaces는 테이블 데이터를 투명하게 암호화하고 해독합니다. Amazon Keyspaces는 봉투 암호화와 키 계층 구조를 사용하여 데이터 암호화 키를 보호합니다. AWS KMS와 통합되어 루트 암호화 키를 저장 및 관리합니다. 암호화 키 계층 구조에 대한 자세한 내용은 the section called “작동 방식” 섹션을 참조하세요. 봉투 암호화와 같은 AWS KMS 개념에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서의 AWS KMS 관리 서비스 개념을 참조하세요.

새 테이블을 만들 때 다음 AWS KMS 키(KMS 키) 중 하나를 선택할 수 있습니다.

- AWS 소유 키 - 기본 암호화 유형입니다. 키는 Amazon Keyspaces가 소유합니다(추가 비용 없음).
- 고객 관리형 키 - 이 키는 사용자의 계정에 저장되며 사용자가 생성, 소유 및 관리합니다. 고객 관리형 키에 대해 사용자가 모든 것을 제어합니다(AWS KMS 비용 적용).

언제든지, AWS 소유 키 및 고객 관리형 키 간에 전환할 수 있습니다. 콘솔을 사용하거나 프로그래밍 방식으로 CQL 문을 사용하여 새 테이블을 생성하거나 기존 테이블의 KMS 키를 변경할 때 고객 관리형 키를 지정할 수 있습니다. 자세한 방법은 저장 시 암호화: 고객 관리형 키를 사용하여 Amazon Keyspaces에서 테이블을 암호화하는 방법 섹션을 참조하세요.

AWS 소유 키의 기본 옵션을 사용한 저장 데이터 암호화는 추가 비용 없이 제공됩니다. 그러나 고객 관리형 키에 대해서는 AWS KMS 비용이 부과됩니다. 요금에 대한 자세한 내용은 AWS KMS 요금을 참조하세요.

AWS 중국(베이징) 및 AWS 중국(닝샤) 리전을 포함한 모든 AWS 리전에서 Amazon Keyspaces 저장 시 암호화를 사용할 수 있습니다. 자세한 내용은 저장 시 암호화: Amazon Keyspaces 작동 방식 섹션을 참조하세요.

주제

- 저장 시 암호화: Amazon Keyspaces 작동 방식
- 저장 시 암호화: 고객 관리형 키를 사용하여 Amazon Keyspaces에서 테이블을 암호화하는 방법

저장 시 암호화: Amazon Keyspaces 작동 방식

Amazon Keyspaces(Apache Cassandra용) 저장 시 암호화는 256비트 고급 암호화 표준(AES-256)을 사용하여 데이터를 암호화합니다. 이는 기본 스토리지에 대한 무단 액세스로부터 데이터의 보안을 유지하는 데 도움을 줍니다. Amazon Keyspaces 테이블의 모든 고객 데이터는 기본적으로 저장 시 암호화되며 서버 측 암호화는 투명하므로 애플리케이션을 변경할 필요가 없습니다.

저장 시 암호화는 테이블을 암호화하는 데 사용되는 암호화 키를 관리하기 위해 AWS Key Management Service(AWS KMS)와(과) 통합됩니다. 새 테이블을 생성하거나 기존 테이블을 업데이트할 때 다음 AWS KMS 키 옵션 중 하나를 선택할 수 있습니다.

- AWS 소유 키 - 기본 암호화 유형입니다. 키는 Amazon Keyspaces가 소유합니다(추가 비용 없음).
- 고객 관리형 키 - 이 키는 사용자의 계정에 저장되며 사용자가 생성, 소유 및 관리합니다. 고객 관리형 키에 대해 사용자가 모든 것을 제어합니다(AWS KMS 비용 적용).

AWS KMS 키(KMS 키)

저장 시 암호화는 AWS KMS 키를 사용하여 모든 Amazon Keyspaces 데이터를 보호합니다. 기본적으로 Amazon Keyspaces는 Amazon Keyspaces 서비스 계정에서 생성 및 관리되는 다중 테넌트 암호화 키인 AWS 소유 키를 사용합니다.

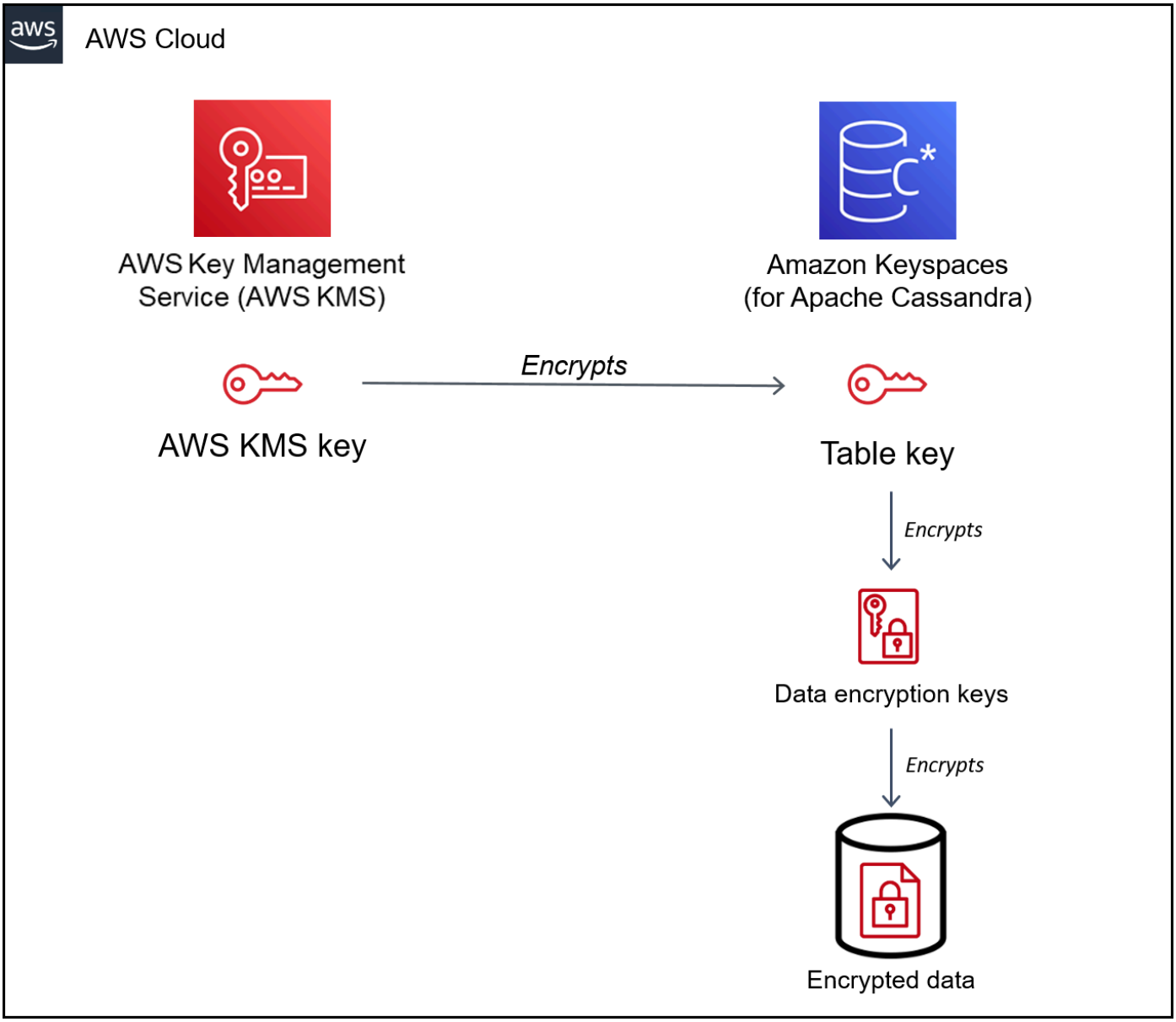
하지만 AWS 계정에서 고객 관리형 키를 사용하여 Amazon Keyspaces 테이블을 암호화할 수 있습니다. 키스페이스의 각 테이블마다 다른 KMS 키를 선택할 수 있습니다. 테이블에 대해 선택한 KMS 키는 모든 메타데이터 및 복원 가능한 백업을 암호화하는 데에도 사용됩니다.

테이블을 생성 또는 업데이트할 때 테이블에 대해 KMS 키를 선택합니다. 언제든지 Amazon Keyspaces 콘솔에서 또는 ALTER TABLE 문을 사용하여 테이블에 대한 KMS 키를 변경할 수 있습니다. KMS 키 전환 프로세스는 원활하며 가동 중지 시간 또는 서비스 저하가 발생하지 않습니다.

키 계층 구조

Amazon Keyspaces는 키 계층 구조를 사용하여 데이터를 암호화합니다. 이 키 계층 구조에서 KMS 키는 루트 키입니다. Amazon Keyspaces 테이블 암호화 키를 암호화 및 해독하는 데 사용됩니다. 테이블 암호화 키는 Amazon Keyspaces가 읽기 및 쓰기 작업을 수행할 때 데이터를 암호화하고 해독하기 위해 내부적으로 사용하는 암호화 키를 암호화하는 데 사용됩니다.

암호화 키 계층 구조를 사용하면 데이터를 다시 암호화하거나 애플리케이션 및 진행 중인 데이터 작업에 영향을 주지 않고도 KMS 키를 변경할 수 있습니다.



테이블 키

Amazon Keyspaces 테이블 키는 키 암호화 키로 사용됩니다. Amazon Keyspaces는 테이블 키를 사용하여 테이블, 로그 파일 및 복원 가능한 백업에 저장된 데이터를 암호화하는 데 사용되는 내부 데이터 암호화 키를 보호합니다. Amazon Keyspaces는 테이블에 있는 각 기본 구조의 고유한 데이터 암호화 키를 생성합니다. 하지만 여러 테이블 행이 동일한 데이터 암호화 키로 보호될 수 있습니다.

KMS 키를 고객 관리형 키로 처음 설정하면 AWS KMS에서 데이터 키를 생성합니다. AWS KMS 데이터 키는 Amazon Keyspaces의 테이블 키를 가리킵니다.

암호화된 테이블에 액세스하면 Amazon Keyspaces가 KMS 키를 사용하여 테이블 키를 해독하도록 AWS KMS에 요청합니다. 그런 다음 일반 텍스트 테이블 키를 사용하여 Amazon Keyspaces 데이터 암호화 키를 해독하고, 일반 텍스트 데이터 암호화 키를 사용하여 테이블 데이터를 해독합니다.

Amazon Keyspaces는 AWS KMS 외부에서 테이블 키와 데이터 암호화 키를 사용하고 저장합니다. 고급 암호화 표준(AES) 암호화 및 256비트 암호화 키로 모든 키를 보호합니다. 그런 다음 필요에 따라 테이블 데이터를 해독할 때 사용할 수 있도록 암호화된 데이터로 암호화된 키를 저장합니다.

테이블 키 캐싱

각 Amazon Keyspaces 작업마다 AWS KMS를 호출하지 않도록 Amazon Keyspaces는 메모리에 있는 각 연결의 일반 텍스트 테이블 키를 캐싱합니다. 5분의 비활성 시간이 경과하여 Amazon Keyspaces가 캐싱된 테이블 키를 요청할 경우 테이블 키를 해독하라는 새로운 요청을 AWS KMS에 보냅니다. 이 호출은 마지막 테이블 키 해독 요청 이후 KMS 키의 액세스 정책에 적용된 모든 변경 사항을 AWS KMS 또는 AWS Identity and Access Management(IAM)에 캡처합니다.

봉투 암호화

테이블의 고객 관리형 키를 변경하면 Amazon Keyspaces가 새 테이블 키를 생성합니다. 그런 다음 새 테이블 키를 사용하여 데이터 암호화 키를 다시 암호화합니다. 또한 새 테이블 키를 사용하여 복원 가능한 백업을 보호하는 데 사용되는 이전 테이블 키를 암호화합니다. 이 프로세스를 봉투 암호화라고 합니다. 이렇게 하면 고객 관리형 키를 교체해도 복원 가능한 백업에 액세스할 수 있습니다. 봉투 암호화에 대한 자세한 내용은 AWS Key Management Service 개발자 안내서에서 봉투 암호화를 참조하세요.

주제

- AWS 소유 키
- 고객 관리형 키
- 저장 시 암호화 사용 노트

AWS 소유 키

AWS 소유 키는 AWS 계정에는 저장되지 않습니다. 이들은 AWS가 여러 AWS 계정 계정에서 사용하기 위해 소유 및 관리하는 KMS 키 모음의 일부입니다. AWS 서비스는 AWS 소유 키를 사용하여 데이터를 보호할 수 있습니다.

사용자는 AWS 소유 키를 확인, 관리 또는 사용하거나 사용을 감사할 수 없습니다. 하지만 사용자는 데이터를 암호화하는 키를 보호하기 위해 어떤 작업을 수행하거나 어떤 프로그램을 변경할 필요가 없습니다.

월별 요금 또는 AWS 소유 키의 사용량에 따른 요금이 부과되지 않으며 계정의 AWS KMS 할당량에 포함되지 않습니다.

고객 관리형 키

고객 관리형 키는 사용자가 생성, 소유 및 관리하는 AWS 계정의 키입니다. 이러한 KMS 키를 완전히 제어할 수 있습니다.

고객 관리형 키를 사용하여 다음과 같은 기능을 얻을 수 있습니다.

- 고객 관리형 키에 대한 액세스를 제어하기 위한 키 정책, IAM 정책 및 권한 부여의 설정 및 유지 관리를 포함하여 고객 관리형 키를 생성하고 관리합니다. 고객 관리형 키를 활성화 및 비활성화하고, 자동 키 교체를 활성화 및 비활성화하고, 고객 관리형 키가 더 이상 사용되지 않을 때 고객 관리형 키를 삭제하도록 예약할 수 있습니다. 관리하는 고객 관리형 키의 태그와 별칭을 생성할 수 있습니다.
- 가져온 키 구성 요소가 있는 고객 관리형 키를 사용하거나 고객이 소유하고 관리하는 사용자 지정 키 스토어에서 고객 관리형 키를 사용할 수 있습니다.
- AWS CloudTrail 및 Amazon CloudWatch Logs를 사용하여 Amazon Keyspaces가 사용자 대신 AWS KMS로 전송하는 요청을 추적할 수 있습니다. 자세한 내용은 the section called “6단계: AWS CloudTrail을 사용하여 모니터링 구성” 섹션을 참조하세요.

고객 관리형 키는 각 API 직접 호출에 대해 요금이 부과되며 이러한 KMS 키에 AWS KMS 할당량이 적용됩니다. 자세한 내용은 AWS KMS 리소스 또는 요청 할당량을 참조하세요.

고객 관리형 키를 테이블에 대한 루트 암호화 키로 지정하면 복원 가능한 백업은 백업이 생성될 때 테이블에 지정된 것과 동일한 암호화 키로 암호화됩니다. 테이블의 KMS 키를 교체하면 키 엔벨로핑을 통해 최신 KMS 키가 복원 가능한 모든 백업에 액세스할 수 있습니다.

Amazon Keyspaces는 테이블 데이터에 대한 액세스를 제공하려면 고객 관리형 키에 액세스할 수 있어야 합니다. 암호화 키 상태가 비활성화됨으로 설정되거나 삭제가 예정된 경우 Amazon Keyspaces는 데이터를 암호화하거나 해독할 수 없습니다. 따라서 테이블에서 읽기 및 쓰기 작업을 수행할 수 없습니다. 서비스에서 사용자의 암호화 키가 접근 불가 상태인 것을 탐지하면 바로 Amazon Keyspaces는 사용자에게 이를 알리기 위해 이메일 알림을 보냅니다.

7일 이내에 암호화 키에 대한 액세스를 복원해야 합니다. 그렇지 않으면 Amazon Keyspaces가 테이블을 자동으로 삭제합니다. 예방 차원에서 Amazon Keyspaces는 테이블을 삭제하기 전에 테이블 데이터의 복원 가능한 백업을 생성합니다. Amazon Keyspaces는 복원 가능한 백업을 35일 동안 유지합니다. 35일이 지나면 더 이상 테이블 데이터를 복원할 수 없습니다. 복원 가능한 백업에 대해서는 요금이 청구되지 않지만 표준 복원 요금이 적용됩니다.

이 복원 가능한 백업을 새로운 테이블에 데이터를 복원할 때 사용할 수 있습니다. 복원을 시작하려면 테이블에 대한 최종 고객 관리형 키가 활성화되어야 하며 Amazon Keyspaces는 이에 액세스할 수 있어야 합니다.

> **Note**
>
> 액세스할 수 없거나 생성 프로세스가 완료되기 전에 삭제되도록 예약된 고객 관리형 키를 사용하여 암호화된 테이블을 생성할 때 오류가 발생합니다. 테이블 생성 작업이 실패하고 이메일 알림이 전송됩니다.

저장 시 암호화 사용 노트

Amazon Keyspaces에서 저장 시 암호화를 사용할 때 다음을 고려하세요.

- 서버 측 저장 시 암호화는 모든 Amazon Keyspaces 테이블에서 활성화되며 비활성화할 수 없습니다. 전체 테이블은 저장 시 암호화되므로 암호화할 특정 열이나 행을 선택할 수 없습니다.
- 기본적으로 Amazon Keyspaces는 단일 서비스 기본 키(AWS 소유 키)를 사용하여 모든 테이블을 암호화합니다. 이 키가 없는 경우 자동으로 생성됩니다. 서비스 기본 키는 비활성화할 수 없습니다.
- 저장 시 암호화는 영구 스토리지 미디어에서 정적(유휴) 상태인 데이터만 암호화합니다. 전송 중인 데이터 또는 사용 중인 데이터에 대한 데이터 보안 문제가 있는 경우 추가 조치를 취해야 합니다.
  - 전송 중 데이터: Amazon Keyspaces의 모든 데이터는 전송 중에 암호화됩니다. 기본적으로 Amazon Keyspaces와 수신 통신은 Secure Sockets Layer(SSL) 및 전송 계층 보안(TLS) 암호화를 사용하여 보호됩니다.
  - 사용 중인 데이터: Amazon Keyspaces에 보내기 전에 클라이언트 측 암호화를 사용하여 데이터를 보호합니다.
  - 고객 관리형 키: 테이블의 저장 데이터는 고객 관리형 키를 사용하여 항상 암호화됩니다. 그러나 여러 행의 원자적 업데이트를 수행하는 작업은 처리 중에 AWS 소유 키를 사용하여 데이터를 임시로 암호화합니다. 여기에는 범위 삭제 작업과 정적 및 비정적 데이터에 동시에 액세스하는 작업이 포함됩니다.
- 단일 고객 관리형 키에 최대 50,000건의 권한 부여를 받을 수 있습니다. 고객 관리형 키와 관련된 모든 Amazon Keyspaces 테이블은 2개의 권한 부여를 소비합니다. 테이블이 삭제되면 하나의 권한 부여가 릴리스됩니다. 두 번째 권한 부여는 Amazon Keyspaces에서 실수로 고객 관리형 키에 대한 액세스 권한을 잃은 경우 데이터 손실을 방지하기 위해 테이블의 자동 스냅샷을 생성하는 데 사용됩니다. 이 권한 부여는 테이블 삭제 후 42일 후에 릴리스됩니다.

저장 시 암호화: 고객 관리형 키를 사용하여 Amazon Keyspaces에서 테이블을 암호화하는 방법

콘솔 또는 CQL 문을 사용하여 새 테이블의 AWS KMS key을 지정하고 Amazon Keyspaces에 있는 기존 테이블의 암호화 키를 업데이트할 수 있습니다. 다음 주제에서는 새 테이블과 기존 테이블에 대한 고객 관리형 키를 구현하는 방법을 간략하게 설명합니다.

주제

전제 조건: AWS KMS를 사용하여 고객 관리형 키 생성 및 Amazon Keyspaces에 권한 부여

고객 관리형 키로 Amazon Keyspaces 테이블을 보호하려면 먼저 AWS Key Management Service(AWS KMS)에서 키를 생성한 다음 Amazon Keyspaces가 해당 키를 사용하도록 승인해야 합니다.

1단계: AWS KMS를 사용하여 고객 관리형 키 생성

Amazon Keyspaces 테이블을 보호하는 데 사용할 고객 관리형 키를 생성하려면 콘솔 또는 AWS API를 사용하여 대칭 암호화 KMS 키 생성의 단계를 따를 수 있습니다.

2단계: 고객 관리형 키 사용 승인

Amazon Keyspaces 테이블을 보호할 고객 관리형 키를 선택하려면 먼저 해당 고객 관리형 키의 정책이 Amazon Keyspaces에 사용자를 대신하여 키를 사용할 수 있는 권한을 부여해야 합니다. 고객은 고객 관리형 키에 대한 정책 및 권한 부여를 완벽하게 제어할 수 있습니다. 키 정책, IAM 정책 또는 권한 부여에 이러한 권한을 제공할 수 있습니다.

Amazon Keyspaces는 기본 AWS 소유 키를 사용하여 AWS 계정의 Amazon Keyspaces 테이블을 보호하기 위해 추가 승인이 필요하지 않습니다.

다음 주제에서는 Amazon Keyspaces 테이블에서 고객 관리형 키를 사용할 수 있도록 허용하는 IAM 정책 및 권한 부여를 사용하여 필요한 권한을 구성하는 방법을 보여 줍니다.

주제

- 고객 관리형 키에 대한 키 정책
- 예제 키 정책
- 권한 부여를 사용하여 Amazon Keyspaces 승인

고객 관리형 키에 대한 키 정책

Amazon Keyspaces 테이블을 보호하기 위해 고객 관리형 키를 선택하면 Amazon Keyspaces는 선택하는 보안 주체를 대신하여 고객 관리형 키를 사용할 수 있는 권한을 얻습니다. 해당 보안 주체, 즉 사용자 또는 역할은 Amazon Keyspaces가 필요한 고객 관리형 키에 대한 권한이 있어야 합니다.

Amazon Keyspaces는 고객 관리형 키에 대해 최소한 다음 권한이 있어야 합니다.

- kms:Encrypt
- kms:Decrypt
- kms:ReEncrypt*(kms:ReEncryptFrom and kms:ReEncryptTo용)
- kms:GenerateDataKey*(kms:GenerateDataKey 및 kms:GenerateDataKeyWithoutPlaintext용)
- kms:DescribeKey
- kms:CreateGrant

예제 키 정책

예를 들어 다음 예제 키 정책은 필요한 권한만 제공합니다. 이 정책에는 다음과 같은 효과가 있습니다.

- Amazon Keyspaces가 암호화 작업에 고객 관리형 키를 사용하고 권한 부여를 생성하도록 허용합니다. 단, Amazon Keyspaces를 사용할 권한이 있는 계정의 보안 주체를 대신해 동작하는 경우에 한합니다. 정책 문에 지정된 보안 주체가 Amazon Keyspaces를 사용할 권한이 없는 경우 Amazon Keyspaces 서비스에서 오는 경우에도 호출이 실패합니다.
- kms:ViaService 조건 키는 정책 문에 나열된 보안 주체를 대신하여 Amazon Keyspaces로부터 요청이 오는 경우에만 사용 권한을 허용합니다. 이러한 보안 주체는 이러한 작업을 직접 호출 할 수 없습니다. 참고: `kms:ViaService` 값(`cassandra.*.amazonaws.com`)은 리전 위치에 별표(*)가 있습니다. Amazon Keyspaces는 특정 AWS 리전과 독립적일 수 있는 권한이 필요합니다.
- 고객 관리형 키 관리자(`db-team` 역할을 수임할 수 있는 사용자)에게 고객 관리형 키에 대한 읽기 전용 액세스와 테이블을 보호하기 위해 Amazon Keyspaces가 필요로 하는 권한 부여를 포함하여 권한 부여를 취소할 수 있는 권한을 제공합니다.

- Amazon Keyspaces는 고객 관리형 키에 대한 읽기 전용 액세스 권한을 부여합니다. 이 경우 Amazon Keyspaces는 이러한 작업을 직접 호출할 수 있습니다. 계정 보안 주체를 대신하여 동작할 필요는 없습니다.

예제 키 정책을 사용하기 전에 예제 보안 주체를 AWS 계정의 실제 보안 주체로 바꿉니다.

```
{
  "Id": "key-policy-cassandra",
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid" : "Allow access through Amazon Keyspaces for all principals in the account
 that are authorized to use Amazon Keyspaces",
      "Effect": "Allow",
      "Principal": {"AWS": "arn:aws:iam::111122223333:user/db-lead"},
      "Action": [
        "kms:Encrypt",
        "kms:Decrypt",
        "kms:ReEncrypt*",
        "kms:GenerateDataKey*",
        "kms:DescribeKey",
        "kms:CreateGrant"
      ],
      "Resource": "*",
      "Condition": {
         "StringLike": {
           "kms:ViaService" : "cassandra.*.amazonaws.com"
         }
      }
    },
    {
      "Sid":  "Allow administrators to view the customer managed key and revoke
 grants",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::111122223333:role/db-team"
       },
      "Action": [
        "kms:Describe*",
        "kms:Get*",
        "kms:List*",
        "kms:RevokeGrant"
```

```
      ],
      "Resource": "*"
    }
  ]
}
```

권한 부여를 사용하여 Amazon Keyspaces 승인

키 정책 이외에 Amazon Keyspaces는 권한 부여를 사용하여 고객 관리형 키에 대한 권한을 설정합니다. 계정에서 고객 관리형 키에 대한 권한 부여를 보려면 ListGrants 작업을 사용합니다. Amazon Keyspaces는 AWS 소유 키를 사용하여 테이블을 보호하는 데 권한 부여 또는 추가 권한이 필요하지 않습니다.

Amazon Keyspaces는 배경 시스템 유지 관리 및 연속 데이터 보호 태스크를 수행할 때 권한 부여 권한을 사용합니다. 권한 부여를 사용하여 테이블 키도 생성합니다.

각 권한 부여는 테이블마다 다릅니다. 계정에 동일한 고객 관리형 키로 암호화되는 여러 테이블이 있는 경우 각 테이블 유형별 권한 부여가 있습니다. 권한 부여는 테이블 이름 및 AWS 계정 ID를 포함하는 Amazon Keyspaces 암호화 컨텍스트에 의해 제한됩니다. 권한 부여에는 더 이상 필요하지 않은 경우 권한 부여를 사용 중지할 수 있는 권한이 포함됩니다.

권한 부여를 생성하려면 Amazon Keyspaces가 암호화된 테이블을 생성한 사용자를 대신하여 `CreateGrant`를 호출할 수 있는 권한이 있어야 합니다.

이 키 정책은 계정이 고객 관리형 키에 대한 권한 부여를 취소하도록 허용할 수도 있습니다. 단, 활성 암호화 테이블에서 권한 부여를 취소할 경우에는 Amazon Keyspaces가 테이블을 보호하고 유지하지 못합니다.

3단계: 새 테이블에 대한 고객 관리형 키 지정

Amazon Keyspaces 콘솔 또는 CQL을 사용하여 새로운 테이블에 고객 관리형 키를 지정하기 위해 다음 단계를 따르세요.

고객 관리형 키(콘솔)를 사용하여 암호화된 테이블 생성

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 테이블을 선택한 다음 테이블 생성을 선택합니다.
3. 테이블 세부 정보 섹션의 테이블 생성 페이지에서 키스페이스를 선택하고 새 테이블의 이름을 입력합니다.

4. 스키마 섹션에서 테이블의 스키마를 생성합니다.
5. 테이블 설정 섹션에서 설정 사용자 지정을 선택합니다.
6. 암호화 설정으로 계속 진행합니다.

   이 단계에서는 테이블의 암호화 설정을 선택합니다.

   AWS KMS key 선택 아래의 저장 시 암호화 섹션에서 다른 KMS 키(고급) 선택 옵션을 선택하고 검색 필드에서 AWS KMS key를 선택하거나 Amazon 리소스 이름(ARN)을 입력합니다.

   > **Note**
   >
   > 선택한 키에 액세스할 수 없거나 필요한 권한이 없는 경우 AWS Key Management Service 개발자 안내서의 키 액세스 문제 해결을 참조하세요.

7. 생성을 선택하여 암호화된 테이블을 생성합니다.

저장 시 암호화(CQL)에 대한 고객 관리형 키를 사용하여 새 테이블 생성

저장 시 암호화에 고객 관리형 키를 사용하는 새 테이블을 생성하려면 다음 예제와 같이 CREATE TABLE 문을 사용할 수 있습니다. Amazon Keyspaces에 권한이 부여된 유효한 키에 대해 키 ARN을 ARN으로 교체해야 합니다.

```
CREATE TABLE my_keyspace.my_table(id bigint, name text, place text STATIC, PRIMARY
 KEY(id, name)) WITH CUSTOM_PROPERTIES = {
        'encryption_specification':{
                'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
                'kms_key_identifier':'arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111'
            }
    };
```

Invalid Request Exception을 수신한 경우 고객 관리형 키가 유효하고 Amazon Keyspaces에 필요한 권한이 있는지 확인해야 합니다. 키가 올바르게 구성되었는지 확인하려면 AWS Key Management Service 개발자 안내서의 키 액세스 문제 해결을 참조하세요.

4단계: 기존 테이블의 암호화 키 업데이트

또한 언제든지 Amazon Keyspaces 콘솔 또는 CQL을 사용하여 AWS 소유 키 및 고객 관리형 KMS 키 간 기존 테이블의 암호화 키를 변경할 수 있습니다.

새 고객 관리형 키로 기존 테이블 업데이트(콘솔)

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 탐색 창에서 테이블을 선택합니다.
3. 업데이트할 테이블을 선택한 다음 추가 설정 탭을 선택합니다.
4. 저장 시 암호화 섹션에서 암호화 관리를 선택하여 테이블의 암호화 설정을 편집합니다.

   AWS KMS key 선택에서 다른 KMS 키(고급) 선택 옵션을 선택하고 검색 필드에서 AWS KMS key를 선택하거나 Amazon 리소스 이름(ARN)을 입력합니다.

   > **Note**
   >
   > 선택한 키가 유효하지 않은 경우 AWS Key Management Service 개발자 안내서의 키 액세스 문제 해결을 참조하세요.

   또는 고객 관리형 키로 암호화된 테이블에 대해 AWS 소유 키를 선택할 수 있습니다.
5. 변경 사항 저장을 선택하여 테이블 변경 사항을 저장합니다.

기존 테이블에 사용된 암호화 키 업데이트

기존 테이블의 암호화 키를 변경하려면 `ALTER TABLE` 문을 사용하여 저장 시 암호화에 대한 고객 관리형 키를 지정합니다. Amazon Keyspaces에 권한이 부여된 유효한 키에 대해 키 ARN을 ARN으로 교체해야 합니다.

```
ALTER TABLE my_keyspace.my_table WITH CUSTOM_PROPERTIES = {
              'encryption_specification':{
                     'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
                     'kms_key_identifier':'arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111'
                  }
         };
```

`Invalid Request Exception`을 수신한 경우 고객 관리형 키가 유효하고 Amazon Keyspaces에 필요한 권한이 있는지 확인해야 합니다. 키가 올바르게 구성되었는지 확인하려면 AWS Key Management Service 개발자 안내서의 키 액세스 문제 해결을 참조하세요.

AWS 소유 키에서 암호화 키를 기본 저장 시 암호화 옵션으로 다시 변경하려면 다음 예와 같이 ALTER TABLE 문을 사용할 수 있습니다.

```
ALTER TABLE my_keyspace.my_table WITH CUSTOM_PROPERTIES = {
                'encryption_specification':{
                      'encryption_type' : 'AWS_OWNED_KMS_KEY'
                    }
         };
```

5단계: 로그에서 Amazon Keyspaces 암호화 컨텍스트 사용

암호화 컨텍스트는 보안되지 않은 임의의 데이터를 포함하는 키-값 페어 세트입니다. 데이터 암호화 요청에 암호화 컨텍스트를 포함하는 경우 AWS KMS는 암호화된 데이터에 암호화 컨텍스트를 암호 방식으로 바인딩합니다. 따라서 동일한 암호화 컨텍스트로 전달해야 이 데이터를 해독할 수 있습니다.

Amazon Keyspaces는 모든 AWS KMS 암호화 작업에서 동일한 암호화 컨텍스트를 사용합니다. 고객 관리형 키를 사용하여 Amazon Keyspaces 테이블을 보호할 경우 암호화 컨텍스트를 사용하여 감사 레코드 및 로그에서 고객 관리형 키의 사용을 식별할 수 있습니다. AWS CloudTrail의 로그 및 Amazon CloudWatch Logs와 같은 로그에서 일반 텍스트에 나타나기도 합니다.

AWS KMS에 요청할 때 Amazon Keyspaces는 세 개의 키-값 쌍이 있는 암호화 컨텍스트를 사용합니다.

```
"encryptionContextSubset": {
    "aws:cassandra:keyspaceName": "my_keyspace",
    "aws:cassandra:tableName": "mytable"
    "aws:cassandra:subscriberId": "111122223333"
}
```

- 키스페이스 – 첫 번째 키-값 쌍은 Amazon Keyspaces가 암호화 중인 테이블을 포함하는 키스페이스를 식별합니다. 키는 aws:cassandra:keyspaceName입니다. 값은 키스페이스의 이름입니다.

  ```
  "aws:cassandra:keyspaceName": "<keyspace-name>"
  ```

  예:

  ```
  "aws:cassandra:keyspaceName": "my_keyspace"
  ```

- 테이블 – 두 번째 키-값 쌍은 Amazon Keyspaces가 암호화 중인 테이블을 식별합니다. 키는 aws:cassandra:tableName입니다. 값은 테이블의 이름입니다.

```
"aws:cassandra:tableName": "<table-name>"
```

예:

```
"aws:cassandra:tableName": "my_table"
```

- 계정 – 세 번째 키-값 쌍은 AWS 계정를 식별합니다. 키는 `aws:cassandra:subscriberId`입니다. 값은 계정 ID입니다.

```
"aws:cassandra:subscriberId": "<account-id>"
```

예:

```
"aws:cassandra:subscriberId": "111122223333"
```

6단계: AWS CloudTrail을 사용하여 모니터링 구성

고객 관리형 키를 사용하여 Amazon Keyspaces 테이블을 보호할 경우 AWS CloudTrail 로그를 사용하여 Amazon Keyspaces가 사용자 대신 AWS KMS로 전송하는 요청을 추적할 수 있습니다.

`GenerateDataKey`, `DescribeKey`, `Decrypt` 및 `CreateGrant` 요청은 이 섹션에서 다룹니다. 또한 Amazon Keyspaces는 RetireGrant 작업을 사용하여 사용자가 테이블을 삭제할 때 권한 부여를 제거합니다.

GenerateDataKey

Amazon Keyspaces는 고유한 테이블 키를 생성하여 저장 데이터를 암호화합니다. 테이블에 대한 KMS 키를 지정하는 GenerateDataKey 요청을 AWS KMS로 보냅니다.

`GenerateDataKey` 작업을 기록하는 이벤트는 다음 예제 이벤트와 유사합니다. 사용자는 Amazon Keyspaces 서비스 계정입니다. 파라미터에는 고객 관리형 키의 Amazon 리소스 이름(ARN), 256비트 키가 필요한 키 지정자, 키스페이스, 테이블 및 AWS 계정를 식별하는 암호화 컨텍스트가 포함됩니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
```

```
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-04-16T04:56:05Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "GenerateDataKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keySpec": "AES_256",
        "encryptionContext": {
            "aws:cassandra:keyspaceName": "my_keyspace",
            "aws:cassandra:tableName": "my_table",
            "aws:cassandra:subscriberId": "123SAMPLE012"
        },
        "keyId": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
    },
    "responseElements": null,
    "requestID": "5e8e9cb5-9194-4334-aacc-9dd7d50fe246",
    "eventID": "49fccab9-2448-4b97-a89d-7d5c39318d6f",
    "readOnly": true,
    "resources": [
        {
            "accountId": "123SAMPLE012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123SAMPLE012",
    "sharedEventID": "84fbaaf0-9641-4e32-9147-57d2cb08792e"
}
```


DescribeKey

Amazon Keyspaces는 DescribeKey 작업을 사용하여 사용자가 선택한 KMS 키가 계정과 리전에 있는지 확인합니다.

`DescribeKey` 작업을 기록하는 이벤트는 다음 예제 이벤트와 유사합니다. 사용자는 Amazon Keyspaces 서비스 계정입니다. 파라미터에는 고객 관리형 키의 ARN과 256비트 키가 필요한 키 지정자가 포함됩니다.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAZ3FNIIVIZZ6H7CFQG",
        "arn": "arn:aws:iam::123SAMPLE012:user/admin",
        "accountId": "123SAMPLE012",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "admin",
        "sessionContext": {
            "sessionIssuer": {},
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-04-16T04:55:42Z"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-04-16T04:55:58Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "DescribeKey",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
    },
    "responseElements": null,
    "requestID": "c25a8105-050b-4f52-8358-6e872fb03a6c",
    "eventID": "0d96420e-707e-41b9-9118-56585a669658",
    "readOnly": true,
    "resources": [
        {
            "accountId": "123SAMPLE012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
```

```
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123SAMPLE012"
}
```

Decrypt

Amazon Keyspaces 테이블에 액세스하면 Amazon Keyspaces가 테이블 키를 해독해야만 계층에서 그 아래에 있는 키를 해독할 수 있습니다. 이후 테이블에 있는 데이터를 해독합니다. 테이블 키를 해독하기 위해 Amazon Keyspaces는 테이블에 대한 KMS 키를 지정하는 AWS KMS로 Decrypt 요청을 보냅니다.

`Decrypt` 작업을 기록하는 이벤트는 다음 예제 이벤트와 유사합니다. 사용자는 테이블에 액세스 중인 AWS 계정에 있는 보안 주체입니다. 파라미터에는 암호화된 테이블 키(사이퍼텍스트 Blob)와, 테이블 및 AWS 계정를 식별하는 암호화 컨텍스트가 포함됩니다. AWS KMS는 사이퍼텍스트에서 고객 관리형 키의 ID를 파생합니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AWSService",
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-04-16T05:29:44Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "Decrypt",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "encryptionContext": {
            "aws:cassandra:keyspaceName": "my_keyspace",
            "aws:cassandra:tableName": "my_table",
            "aws:cassandra:subscriberId": "123SAMPLE012"
        },
        "encryptionAlgorithm": "SYMMETRIC_DEFAULT"
    },
    "responseElements": null,
    "requestID": "50e80373-83c9-4034-8226-5439e1c9b259",
    "eventID": "8db9788f-04a5-4ae2-90c9-15c79c411b6b",
```

```
    "readOnly": true,
    "resources": [
        {
            "accountId": "123SAMPLE012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123SAMPLE012",
    "sharedEventID": "7ed99e2d-910a-4708-a4e3-0180d8dbb68e"
}
```

CreateGrant

고객 관리형 키를 사용하여 Amazon Keyspaces 테이블을 보호하면 Amazon Keyspaces는 권한 부여를 사용하여 서비스가 지속적 데이터 보호와 유지 관리 및 내구성 태스크를 수행하도록 허용합니다. 이러한 권한 부여가 AWS 소유 키에서는 필요하지 않습니다.

Amazon Keyspaces가 생성하는 권한 부여는 테이블마다 다릅니다. CreateGrant 요청에 있는 주체는 테이블을 생성한 사용자입니다.

`CreateGrant` 작업을 기록하는 이벤트는 다음 예제 이벤트와 유사합니다. 파라미터에는 테이블에 대한 고객 관리형 키의 ARN, 피부여자 주체 및 삭제 보안 주체(Amazon Keyspaces 서비스), 권한 부여가 적용되는 작업이 포함됩니다. 모든 암호화 작업에서 지정된 암호화 컨텍스트를 사용할 것을 요구하는 제한도 포함됩니다.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAZ3FNIIVIZZ6H7CFQG",
        "arn": "arn:aws:iam::arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111:user/admin",
        "accountId": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111",
        "accessKeyId": "AKIAI44QH8DHBEXAMPLE",
        "userName": "admin",
        "sessionContext": {
            "sessionIssuer": {},
```

```
            "webIdFederationData": {},
            "attributes": {
                "mfaAuthenticated": "false",
                "creationDate": "2021-04-16T04:55:42Z"
            }
        },
        "invokedBy": "AWS Internal"
    },
    "eventTime": "2021-04-16T05:11:10Z",
    "eventSource": "kms.amazonaws.com",
    "eventName": "CreateGrant",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "AWS Internal",
    "userAgent": "AWS Internal",
    "requestParameters": {
        "keyId": "a7d328af-215e-4661-9a69-88c858909f20",
        "operations": [
            "DescribeKey",
            "GenerateDataKey",
            "Decrypt",
            "Encrypt",
            "ReEncryptFrom",
            "ReEncryptTo",
            "RetireGrant"
        ],
        "constraints": {
            "encryptionContextSubset": {
                "aws:cassandra:keyspaceName": "my_keyspace",
                "aws:cassandra:tableName": "my_table",
                "aws:cassandra:subscriberId": "123SAMPLE012"
            }
        },
        "retiringPrincipal": "cassandratest.us-east-1.amazonaws.com",
        "granteePrincipal": "cassandratest.us-east-1.amazonaws.com"
    },
    "responseElements": {
        "grantId":
"18e4235f1b07f289762a31a1886cb5efd225f069280d4f76cd83b9b9b5501013"
    },
    "requestID": "b379a767-1f9b-48c3-b731-fb23e865e7f7",
    "eventID": "29ee1fd4-28f2-416f-a419-551910d20291",
    "readOnly": false,
    "resources": [
        {
```

```
            "accountId": "123SAMPLE012",
            "type": "AWS::KMS::Key",
            "ARN": "arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "eventCategory": "Management",
    "recipientAccountId": "123SAMPLE012"
}
```

Amazon Keyspaces에서 전송 중 암호화

Amazon Keyspaces는 전송 계층 보안(TLS)을 사용한 보안 연결만 허용합니다. 전송 중 암호화는 Amazon Keyspaces와 주고 받는 데이터를 암호화하여 추가 데이터 보호 계층을 제공합니다. 조직의 정책, 업계나 정부 규범 및 규정 준수 요건에 따라 네트워크를 통해 데이터를 전송할 때 애플리케이션의 데이터 보안을 강화하기 위해 전송 중 암호화를 사용해야 하는 경우가 많습니다.

TLS를 사용하여 Amazon Keyspaces에 대한 `cqlsh` 연결을 암호화하는 방법을 알아보려면 the section called “TLS에 대한 `cqlsh` 연결을 수동으로 구성하는 방법” 섹션을 참조하세요. 클라이언트 드라이버에서 TLS 암호화를 사용하는 방법을 알아보려면 the section called “Cassandra 클라이언트 드라이버 사용” 섹션을 참조하세요.

Amazon Keyspaces의 인터네트워크 트래픽 개인 정보

이 주제에서는 Amazon Keyspaces(Apache Cassandra용)가 온프레미스 애플리케이션에서 Amazon Keyspace로의 연결과 Amazon Keyspace와 동일한 AWS 리전 내의 다른 AWS 리소스 간의 연결을 보호하는 방법을 설명합니다.

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 VPN 사용 설명서의 AWS Site-to-Site VPN란 무엇입니까?를 참조하세요.
- AWS Direct Connect 연결 자세한 내용은 AWS Direct Connect 사용 설명서의 AWS Direct Connect란 무엇입니까?를 참조하세요.

관리형 서비스인 Amazon Keyspaces(Apache Cassandra용)는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon Keyspaces에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- 전송 계층 보안(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

Amazon Keyspaces는 클라이언트 요청을 인증하는 두 가지 방법을 지원합니다. 첫 번째 방법은 특정 IAM 사용자에 대해 생성된 암호 기반 보안 인증인 서비스별 보안 인증을 사용합니다. IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 암호를 생성하고 관리할 수 있습니다. 자세한 내용은 Amazon Keyspaces로 IAM 사용을 참조하세요.

두 번째 방법은 Cassandra용 오픈 소스 DataStax Java 드라이버에 대한 인증 플러그인을 사용하는 것입니다. 이 플러그인을 사용하면 IAM 사용자, 역할 및 페더레이션 ID가 AWS 서명 버전 4 프로세스(SigV4)를 사용하여 Amazon Keyspaces(Apache Cassandra용) API 요청에 인증 정보를 추가할 수 있습니다. 자세한 내용은 the section called “인증을 위한 IAM 자격 증명 AWS ” 섹션을 참조하세요.

같은 리전에 있는 AWS 리소스 사이의 트래픽

인터페이스 VPC 엔드포인트를 사용하면 Amazon VPC에서 실행되는 Virtual Private Cloud(VPC)와 Amazon Keyspaces 간에 프라이빗 통신이 가능해집니다. 인터페이스 VPC 엔드포인트는 VPC와 AWS 서비스 간에 프라이빗 통신을 가능하게 하는 AWS 서비스인 AWS PrivateLink에 의해 구동됩니다. AWS PrivateLink는 VPC에 프라이빗 IP가 있는 탄력적 네트워크 인터페이스를 사용하여 네트워크 트래픽이 Amazon 네트워크를 벗어나지 않도록 함으로써 이를 가능하게 합니다. 인터페이스 VPC 엔드포인트에는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결이 필요하지 않습니다. 자세한 내용은 Amazon Virtual Private Cloud 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요. 예제 정책은 the section called “Amazon Keyspaces에 인터페이스 VPC 엔드포인트 사용” 섹션을 참조하세요.

AWS Identity and Access Management 아마존 키스페이스의 경우

AWS Identity and Access Management (IAM) 은 관리자가 리소스에 대한 액세스를 안전하게 제어할 수 AWS 서비스 있도록 AWS 도와줍니다. IAM 관리자는 어떤 사용자가 Amazon Keyspaces 리소스를 사용할 수 있는 인증(로그인) 및 권한(권한 있음)을 받을 수 있는지 제어합니다. IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

주제

고객

Amazon Keyspace에서 수행하는 작업에 따라 사용 방법 AWS Identity and Access Management (IAM) 이 다릅니다.

서비스 사용자 – Amazon Keyspaces 서비스를 사용하여 작업을 수행하는 경우 필요한 보안 인증과 권한을 관리자가 제공합니다. 더 많은 Amazon Keyspaces 기능을 사용하여 작업을 수행한다면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. Amazon Keyspaces의 기능에 액세스할 수 없다면 Amazon Keyspaces ID 및 액세스 문제 해결 섹션을 참조하세요.

서비스 관리자 – 회사에서 Amazon Keyspaces 리소스를 책임지고 있다면 Amazon Keyspaces에 대한 완전한 액세스 권한이 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 Amazon Keyspaces 기능과 리소스를 결정합니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 IAM의 기본 개념을 이해하십시오. 회사가 Amazon Keyspaces에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon Keyspaces에서 IAM을 사용하는 방법 섹션을 참조하세요.

IAM 관리자 - IAM 관리자라면 Amazon Keyspaces에 대한 액세스 관리 정책 작성 방법을 자세히 알고 싶을 수도 있습니다. IAM에서 사용할 수 있는 Amazon Keyspaces ID 기반 정책 예제를 보려면 Amazon Keyspaces ID 기반 정책 예제 섹션을 참조하세요.

ID를 통한 인증

인증은 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페더레이션 ID의 예입니다. 페더레이션 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 내 로그인 방법을 참조하십시오. AWS AWS 계정

AWS 프로그래밍 방식으로 액세스하는 경우 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트 (SDK) 와 명령줄 인터페이스 (CLI) 를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청 서명을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다중 인증 및 IAM 사용 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하십시오.

AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 태스크를 수행하는 데 사용하세요. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 Tasks that require root user credentials를 참조하십시오.

IAM 사용자 및 그룹

IAM 사용자는 단일 사용자 또는 애플리케이션에 대한 특정 권한을 가진 사용자 내의 자격 증명입니다. AWS 계정 가능하면 암호 및 액세스 키와 같은 장기 보안 인증이 있는 IAM 사용자를 생성하는 대신 임시 보안 인증을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 보안 인증이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하십시오.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서의 IAM 사용자를 만들어야 하는 경우(역할이 아님)를 참조하십시오.

IAM 역할

IAM 역할은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console 전환하여 에서 일시적으로 IAM 역할을 맡을 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 사용을 참조하십시오.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 서드 파티 ID 공급자의 역할 생성 단원을 참조하십시오. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하십시오.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 태스크에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입

니다. 그러나 일부 AWS 서비스경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 계정 간 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하십시오.

- 서비스 간 액세스 — 일부는 다른 기능을 사용합니다. AWS 서비스 AWS 서비스예를 들어 서비스에서 직접적 호출을 수행하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 태스크를 수행할 수 있습니다.
  - 순방향 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 보안 AWS 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 AWS 서비스서비스에 AWS 서비스 요청하기 위한 요청과 결합하여 사용합니다. FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하십시오.
  - 서비스 연결 역할 — 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.
- Amazon EC2에서 실행되는 애플리케이션 — IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하십시오.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하십시오.

정책을 사용한 액세스 관리

정책을 생성하고 이를 AWS ID 또는 리소스에 AWS 연결하여 액세스를 제어할 수 있습니다. 정책은 ID 또는 리소스와 연결될 때 AWS 해당 권한을 정의하는 객체입니다. AWS 주도자 (사용자, 루트 사용자

또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 JSON 문서로 AWS 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요를 참조하십시오.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI, 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

보안 인증 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하십시오.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 내 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책과 인라인 정책의 선택을 참조하십시오.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우, 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. IAM의 AWS 관리형 정책은 리소스 기반 정책에 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ACL을 지원하는 서비스의 예로는 아마존 S3와 아마존 VPC가 있습니다. AWS WAF ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 가이드의 ACL(액세스 제어 목록) 개요를 참조하십시오.

기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 타입에 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. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 보안 인증 기반 정책과 그 권한 경계의 교집합입니다. `Principal` 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 내용은 IAM 사용 설명서의 IAM 엔터티에 대한 권한 경계를 참조하십시오.
- 서비스 제어 정책 (SCP) - SCP는 조직 또는 조직 단위 (OU) 에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations AWS Organizations 사업체가 소유한 여러 AWS 계정 개를 그룹화하고 중앙에서 관리하는 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 구성원 계정의 엔티티 (각 엔티티 포함) 에 대한 권한을 제한합니다. AWS 계정 루트 사용자조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 SCP 작동 방식을 참조하십시오.
- 세션 정책 - 세션 정책은 역할 또는 페더레이션 사용자에 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 보안 인증 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 정책을 참조하십시오.

여러 정책 타입

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련되어 있을 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하십시오.

Amazon Keyspaces에서 IAM을 사용하는 방법

IAM을 사용하여 Amazon Keyspaces에 대한 액세스를 관리하기 전에 Amazon Keyspaces에서 사용할 수 있는 IAM 기능을 이해해야 합니다. Amazon Keyspaces 및 AWS 기타 서비스가 IAM과 어떻게 연동되는지 자세히 알아보려면 IAM 사용 설명서에서 IAM과 연동되는 서비스를AWS 참조하십시오.

주제

- Amazon Keyspaces ID 기반 정책
- Amazon Keyspaces 리소스 기반 정책
- Amazon Keyspaces 태그 기반 권한 부여
- Amazon Keyspaces IAM 역할

Amazon Keyspaces ID 기반 정책

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스 및 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. Amazon Keyspaces는 특정 작업, 리소스 및 조건 키를 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

Amazon Keyspaces 서비스별 리소스 및 작업과 IAM 권한 정책에 사용할 수 있는 조건 컨텍스트 키를 보려면 서비스 권한 부여 참조의 Amazon Keyspaces(Apache Cassandra용)의 작업, 리소스 및 조건 키를 참조하세요.

작업

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 태스크를 설명합니다. 정책 작업은 일반적으로 관련 AWS API 작업과 이름이 같습니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하십시오.

Amazon Keyspaces의 정책 작업은 작업 앞에 `cassandra:` 접두사를 사용합니다. 예를 들어 Amazon Keyspaces `CREATE` CQL 문으로 Amazon Keyspaces 키스페이스를 생성할 수 있는 권한을 부여하려면 해당 정책에 `cassandra:Create` 작업을 포함합니다. 정책 문에는 `Action` 또는 `NotAction` 요소

가 포함되어야 합니다. Amazon Keyspaces는 이 서비스로 수행할 수 있는 태스크를 설명하는 고유한 작업 세트를 정의합니다.

명령문 하나에 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "cassandra:CREATE",
      "cassandra:MODIFY"
          ]
```

Amazon Keyspaces 작업 목록을 보려면 서비스 권한 부여 참조의 Amazon Keyspaces(Apache Cassandra용)에서 정의한 작업을 참조하세요.

리소스

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 하나 이상의 개체를 지정합니다. 문장에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이 태스크를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

Amazon Keyspaces에서는 키스페이스와 테이블을 IAM 권한의 `Resource` 요소로 사용할 수 있습니다.

Amazon Keyspaces 키스페이스 리소스에는 다음 ARN이 있습니다.

```
arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/${KeyspaceName}/
```

Amazon Keyspaces 테이블 리소스에는 다음 ARN이 있습니다.

```
arn:${Partition}:cassandra:${Region}:${Account}:/keyspace/${KeyspaceName}/table/
${tableName}
```

ARN 형식에 대한 자세한 내용은 Amazon 리소스 이름 (ARN) 및 AWS 서비스 네임스페이스를 참조하십시오.

예를 들어 명령문에 mykeyspace 키스페이스를 지정하려면 다음 ARN을 사용합니다.

```
"Resource": "arn:aws:cassandra:us-east-1:123456789012:/keyspace/mykeyspace/"
```

특정 계정에 속하는 모든 키스페이스를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:cassandra:us-east-1:123456789012:/keyspace/*"
```

리소스 생성 작업과 같은 일부 Amazon Keyspaces 작업은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

```
"Resource": "*"
```

표준 드라이버를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 연결하려면 보안 주체가 시스템 테이블에 대한 SELECT 액세스 권한을 가지고 있어야 합니다. 이는 대부분의 드라이버가 연결 시 시스템 키스페이스/테이블을 읽기 때문입니다. 예를 들어 IAM 사용자에게 mykeyspace의 mytable에 대한 SELECT 권한을 부여하려면 보안 주체는 mytable 및 system keyspace 모두 읽을 수 있는 권한이 있어야 합니다. 단일 문에서 여러 리소스를 지정하려면 ARN을 쉼표로 구분합니다.

```
"Resource": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
```

Amazon Keyspaces 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 권한 부여 참조의 Amazon Keyspaces(Apache Cassandra용)에서 정의한 작업을 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon Keyspaces(Apache Cassandra용)에서 정의한 작업을 참조하세요.

조건 키

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 Condition 요소를 지정하거나 단일 Condition 요소에서 여러 키를 지정하는 경우, AWS 는 논리적 AND 태스크를 사용하여 평가합니다. 단일 조건 키에 여러 값을 지정하는 경우 는 논리적 OR 연산을 사용하여 조건을 AWS 평가합니다. 명문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예컨대, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하십시오.

AWS 글로벌 조건 키 및 서비스별 조건 키를 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 AWS 설명서의 글로벌 조건 컨텍스트 키를 참조하십시오.

Amazon Keyspaces는 자체 조건 키 집합을 정의하며 일부 전역 조건 키 사용도 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 설명서의AWS 글로벌 조건 컨텍스트 키를 참조하십시오.

모든 Amazon Keyspaces 작업은 aws:RequestTag/${TagKey}, aws:ResourceTag/${TagKey} 및 aws:TagKeys 조건 키를 지원합니다. 자세한 정보는 the section called “ 태그를 기반으로 한 Amazon Keyspaces 리소스 액세스”을 참조하세요.

Amazon Keyspaces 조건 키 목록을 보려면 서비스 권한 부여 참조의 Amazon Keyspaces(Apache Cassandra용)에 사용되는 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon Keyspaces(Apache Cassandra용)에서 정의한 작업을 참조하세요.

예제

Amazon Keyspaces ID 기반 정책 예제를 보려면 Amazon Keyspaces ID 기반 정책 예제 섹션을 참조하세요.

Amazon Keyspaces 리소스 기반 정책

Amazon Keyspaces는 리소스 기반 정책을 지원하지 않습니다. 자세한 리소스 기반 정책 페이지의 예를 보려면 https://docs.aws.amazon.com/lambda/latest/dg/access-control-resource-based.html 섹션을 참조하세요.

Amazon Keyspaces 태그 기반 권한 부여

태그를 사용하여 Amazon Keyspaces 리소스에 대한 액세스를 관리할 수 있습니다. 태그를 기반으로 리소스 액세스를 관리하려면 cassandra:ResourceTag/*key-name*, aws:RequestTag/*key-name* 또는 aws:TagKeys 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. Amazon Keyspaces 리소스 태그 지정에 대한 자세한 내용은 the section called “태그 사용하기” 섹션을 참조하세요.

해당 리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 ID 기반 정책의 예를 보려면 태그를 기반으로 한 Amazon Keyspaces 리소스 액세스 섹션을 참조하세요.

Amazon Keyspaces IAM 역할

IAM 역할은 특정 권한을 AWS 계정 가진 사용자 내의 엔티티입니다.

Amazon Keyspaces에서 임시 보안 인증 사용

임시 자격 증명을 사용하여 페더레이션을 통해 로그인하거나, IAM 역할을 수임하거나, 교차 계정 역할을 수임할 수 있습니다. AssumeRole또는 GetFederation토큰과 같은 AWS STS API 작업을 호출하여 임시 보안 자격 증명을 얻습니다.

Amazon Keyspace는 다음 언어의 Github 리포지토리에서 제공되는 AWS 서명 버전 4 (SigV4) 인증 플러그인을 통해 임시 자격 증명을 사용할 수 있도록 지원합니다.

- Java: https://github.com/aws/aws-sigv4-auth-cassandra-java-driver-plugin.
- Node.js: https://github.com/aws/aws-sigv4-auth-cassandra-nodejs-driver-plugin.
- Python: https://github.com/aws/aws-sigv4-auth-cassandra-python-driver-plugin.
- Go: https://github.com/aws/aws-sigv4-auth-cassandra-gocql-driver-plugin.

Amazon Keyspaces에 프로그래밍 방식으로 액세스하기 위한 인증 플러그인을 구현하는 예제와 자습서는 을 참조하십시오. the section called “Cassandra 클라이언트 드라이버 사용”

서비스 연결 역할

서비스 연결 역할을 사용하면 AWS 서비스가 다른 서비스의 리소스에 액세스하여 사용자를 대신하여 작업을 완료할 수 있습니다. 서비스 연결 역할은 IAM 계정에 나타나고 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수 없습니다.

Amazon Keyspaces 서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.

서비스 역할

Amazon Keyspaces는 서비스 역할을 지원하지 않습니다.

Amazon Keyspaces ID 기반 정책 예제

기본적으로 IAM 사용자 및 역할은 Amazon Keyspaces 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 콘솔, CQLSH 또는 API를 사용하여 작업을 수행할 수 없습니다. AWS CLI AWS IAM 관리

자는 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 ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

- 정책 모범 사례
- Amazon Keyspaces 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용
- Amazon Keyspaces 테이블에 액세스
- 태그를 기반으로 한 Amazon Keyspaces 리소스 액세스

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Amazon Keyspaces 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하고 최소 권한 권한으로 이동 — 사용자와 워크로드에 권한을 부여하려면 여러 일반 사용 사례에 권한을 부여하는 AWS 관리형 정책을 사용하세요. 해당 내용은 에서 사용할 수 있습니다. AWS 계정사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 더 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책 또는 직무에 대한AWS 관리형 정책을 참조하십시오.
- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 태스크를 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서의 IAM의 정책 및 권한을 참조하십시오.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어 SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. 예를 AWS 서비스들어 특정 작업을 통해 서비스 작업을 사용하는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 AWS CloudFormation있습니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하십시오.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 확

인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer 정책 검증을 참조하십시오.

- 멀티 팩터 인증 (MFA) 필요 - IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 AWS 계정 MFA를 활성화하십시오. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 MFA 보호 API 액세스 구성을 참조하십시오.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례를 참조하십시오.

Amazon Keyspaces 콘솔 사용

Amazon Keyspaces는 Amazon Keyspaces 콘솔에 액세스하는 데 특정 권한이 필요하지 않습니다. Amazon Keyspaces 리소스의 세부 정보를 나열하고 보려면 최소한 읽기 전용 권한이 있어야 합니다. AWS 계정최소 필수 권한보다 더 제한적인 보안 인증 기반 정책을 만들면 콘솔이 해당 정책에 연결된 개체(IAM 사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

Amazon Keyspaces 콘솔 액세스를 위해 엔티티가 사용할 수 있는 AWS 관리형 정책은 두 가지입니다.

- AmazonKeyspacesReadOnlyAccess_v2 — 이 정책은 Amazon Keyspace에 대한 읽기 전용 액세스 권한을 부여합니다.
- AmazonKeyspacesFullAccess— 이 정책은 모든 기능에 대한 전체 액세스 권한을 가진 Amazon Keyspace를 사용할 수 있는 권한을 부여합니다.

Amazon Keyspaces 관리형 정책에 대한 자세한 내용은 the section called “AWS 관리형 정책” 섹션을 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 AWS CLI 권한이 포함됩니다. AWS

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
```

```
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Keyspaces 테이블에 액세스

다음은 Amazon Keyspaces 시스템 테이블에 대한 읽기 전용 (SELECT) 액세스 권한을 부여하는 샘플 정책입니다. 모든 샘플의 경우 Amazon 리소스 이름 (ARN) 의 지역 및 계정 ID를 자체 것으로 바꾸십시오.

> **Note**
>
> 표준 드라이버를 사용하여 연결하려면 사용자는 최소한 시스템 테이블에 대한 SELECT 액세스 권한을 가지고 있어야 합니다. 이는 대부분의 드라이버가 연결 시 시스템 키스페이스/테이블을 읽기 때문입니다.

```
{
   "Version":"2012-10-17",
```

```
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

다음 샘플 정책은 키스페이스의 사용자 테이블에 `mytable` 대한 읽기 전용 액세스를 추가합니다. `mykeyspace`

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

다음 샘플 정책은 사용자 테이블에 대한 읽기/쓰기 액세스 권한과 시스템 테이블에 대한 읽기 액세스 권한을 할당합니다.

> **Note**
>
> 시스템 테이블은 항상 읽기 전용입니다.

```
{
```

```
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Select",
            "cassandra:Modify"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/
mytable",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

다음 샘플 정책은 사용자가 키스페이스 mykeyspace에서 테이블을 생성할 수 있도록 허용합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Action":[
            "cassandra:Create",
            "cassandra:Select"
         ],
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/*",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ]
      }
   ]
}
```

태그를 기반으로 한 Amazon Keyspaces 리소스 액세스

ID 기반 정책의 조건을 사용하여 태그를 기반으로 한 Amazon Keyspaces 리소스에 대한 액세스를 제어할 수 있습니다. 이러한 정책은 계정 내 키스페이스 및 테이블의 가시성을 제어합니다. 단, 시스템 테이블에 대한 태그 기반 권한은 AWS SDK를 사용하여 요청하는 경우와 Cassandra 드라이버 및 개발자 도구를 통한 CQL (Cassandra Query Language) API 호출과 다르게 동작합니다.

- 태그 기반 액세스를 사용할 때 AWS SDK로 `List` 및 `Get` 리소스를 요청하려면 호출자에게 시스템 테이블에 대한 읽기 액세스 권한이 있어야 합니다. 예를 들어 `GetTable` 작업을 통해 시스템 테이블에서 데이터를 읽으려면 `Select` 작업 권한이 필요합니다. 호출자에게 특정 테이블에 대한 태그 기반 액세스 권한만 있는 경우 시스템 테이블에 대한 추가 액세스가 필요한 작업은 실패합니다.
- 설정된 Cassandra 드라이버 동작과의 호환성을 위해 Cassandra 드라이버 및 개발자 도구를 통해 Cassandra 쿼리 언어(CQL) API 호출을 사용하여 시스템 테이블에서 작업을 수행할 때는 태그 기반 권한 부여 정책이 적용되지 않습니다.

다음 예제는 테이블의 `Owner`에 사용자의 사용자 이름 값이 포함된 경우 사용자에게 테이블을 볼 수 있는 권한을 부여하는 정책을 생성하는 방법을 보여 줍니다. 이 예에서는 시스템 테이블에 대한 읽기 액세스 권한도 부여합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ReadOnlyAccessTaggedTables",
         "Effect":"Allow",
         "Action":"cassandra:Select",
         "Resource":[
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/table/*",
            "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
         ],
         "Condition":{
            "StringEquals":{
               "aws:ResourceTag/Owner":"${aws:username}"
            }
         }
      }
   ]
}
```

이 정책을 계정의 IAM 사용자에게 연결할 수 있습니다. `richard-roe`라는 사용자가 Amazon Keyspaces 테이블을 보려고 시도하는 경우 테이블에는 `Owner=richard-roe` 또는 `owner=richard-roe`라는 태그를 지정해야 합니다. 그렇지 않으면 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 `Owner`는 `Owner` 및 `owner` 모두와 일치합니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

다음 정책은 테이블의 `Owner`에 사용자의 사용자 이름 값이 포함된 경우 사용자에게 태그를 사용하여 테이블을 생성할 수 있는 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
       {
          "Sid": "CreateTagTableUser",
          "Effect": "Allow",
          "Action": [
              "cassandra:Create",
              "cassandra:TagResource"
          ],
          "Resource": "arn:aws:cassandra:us-east-1:111122223333:/keyspace/mykeyspace/
table/*",
          "Condition":{
             "StringEquals":{
                "aws:RequestTag/Owner":"${aws:username}"
            }
         }
      }
   ]
}
```

Amazon Keyspaces의 AWS 관리형 정책

AWS 관리형 정책은 AWS에 의해 생성되고 관리되는 독립 실행형 정책입니다. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있기 때문에 특정 사용 사례에 대해 최소 권한을 부여하지 않을 수 있습니다. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에서 정의한 권한은 변경할 수 없습니다. AWS에서 AWS 관리형 정책에 정의된 권한을 업데이트할 경우 정책이 연결되어 있는 모든 보안 주체 엔터티(사용자, 그룹 및 역할)에도 업데이트가 적용됩니다. 새로운 AWS 서비스를 시작하거나 새로운 API 작업을 기존 서비스에 이용하는 경우 AWS가 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 관리형 정책: AmazonKeyspacesReadOnlyAccess_v2

AmazonKeyspacesReadOnlyAccess_v2 정책을 IAM 자격 증명에 연결할 수 있습니다.

이 정책은 Amazon Keyspaces에 대한 읽기 전용 액세스 권한을 부여하며 이 정책은 프라이빗 VPC 엔드포인트를 통해 연결할 때 필요한 권한을 포함합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- Amazon Keyspaces - Amazon Keyspaces에 대한 읽기 전용 액세스 권한을 제공합니다.
- Application Auto Scaling - 보안 주체가 Application Auto Scaling의 구성을 볼 수 있도록 허용합니다. 이는 사용자가 테이블에 첨부된 자동 조정 정책을 볼 수 있도록 하기 위해 필요합니다.
- CloudWatch – 보안 주체가 CloudWatch에서 구성된 지표 데이터와 경보를 볼 수 있도록 허용합니다. 이는 사용자가 청구 가능한 테이블 크기 및 테이블에 구성된 CloudWatch 경보를 볼 수 있도록 하기 위해 필요합니다.
- AWS KMS - 보안 주체가 AWS KMS에 구성된 키를 볼 수 있도록 허용합니다. 이는 사용자가 자신의 계정에서 생성하고 관리하는 AWS KMS 키를 보고 Amazon Keyspaces에 할당된 키가 활성화된 대칭 암호화 키인지 확인할 수 있도록 하기 위해 필요합니다.
- Amazon EC2 - VPC 엔드포인트를 통해 Amazon Keyspaces에 연결하는 보안 주체가 Amazon EC2 인스턴스의 VPC에 엔드포인트 및 네트워크 인터페이스 정보를 쿼리할 수 있도록 허용합니다. Amazon Keyspaces가 연결 로드 밸런싱에 사용되는 system.peers 테이블에서 사용 가능한 인터페이스 VPC 엔드포인트를 조회하고 저장할 수 있으려면 Amazon EC2 인스턴스에 대한 이러한 읽기 전용 액세스가 필요합니다.

정책을 JSON 형식으로 검토하려면 AmazonKeyspacesReadOnlyAccess_v2를 참조하세요.

AWS 관리형 정책: AmazonKeyspacesReadOnlyAccess

AmazonKeyspacesReadOnlyAccess 정책을 IAM 자격 증명에 연결할 수 있습니다.

이 정책은 Amazon Keyspaces에 대한 읽기 전용 액세스 권한을 부여합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- Amazon Keyspaces - Amazon Keyspaces에 대한 읽기 전용 액세스 권한을 제공합니다.
- Application Auto Scaling - 보안 주체가 Application Auto Scaling의 구성을 볼 수 있도록 허용합니다. 이는 사용자가 테이블에 첨부된 자동 조정 정책을 볼 수 있도록 하기 위해 필요합니다.
- CloudWatch – 보안 주체가 CloudWatch에서 구성된 지표 데이터와 경보를 볼 수 있도록 허용합니다. 이는 사용자가 청구 가능한 테이블 크기 및 테이블에 구성된 CloudWatch 경보를 볼 수 있도록 하기 위해 필요합니다.
- AWS KMS - 보안 주체가 AWS KMS에 구성된 키를 볼 수 있도록 허용합니다. 이는 사용자가 자신의 계정에서 생성하고 관리하는 AWS KMS 키를 보고 Amazon Keyspaces에 할당된 키가 활성화된 대칭 암호화 키인지 확인할 수 있도록 하기 위해 필요합니다.

정책을 JSON 형식으로 검토하려면 AmazonKeyspacesReadOnlyAccess를 참조하세요.

AWS 관리형 정책: AmazonKeyspacesFullAccess

AmazonKeyspacesFullAccess 정책을 IAM 자격 증명에 연결할 수 있습니다.

이 정책은 관리자에게 Amazon Keyspaces에 대한 무제한 액세스를 허용하는 관리 권한을 부여합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `Amazon Keyspaces` - 보안 주체가 모든 Amazon Keyspaces 리소스에 액세스하고 모든 작업을 수행할 수 있도록 허용합니다.
- `Application Auto Scaling` - 보안 주체가 Amazon Keyspaces 테이블에 대한 자동 조정 정책을 생성, 확인 및 삭제할 수 있도록 허용합니다. 이는 관리자가 Amazon Keyspaces 테이블의 자동 조정 정책을 관리할 수 있도록 하기 위해 필요합니다.
- `CloudWatch` - 보안 주체가 청구 가능한 테이블 크기를 확인하고 Amazon Keyspaces 자동 조정 정책에 대한 CloudWatch 경보를 생성, 확인 및 삭제할 수 있도록 허용합니다. 이는 관리자가 청구 가능한 테이블 크기를 확인하고 CloudWatch 대시보드를 생성할 수 있도록 하기 위해 필요합니다.
- `IAM` - 다음 기능이 켜져 있을 때 Amazon Keyspaces가 IAM을 사용하여 서비스 연결 역할을 자동으로 생성할 수 있도록 허용합니다.
  - `Application Auto Scaling` - 관리자가 테이블에 대해 Application Auto Scaling을 활성화하면 Amazon Keyspaces는 사용자를 대신하여 자동 조정 작업을 수행하는 서비스 연결 역할을 생성합니다.
  - `Amazon Keyspaces Multi-Region Replication` - 관리자가 다중 리전 키스페이스를 생성하면 사용자를 대신하여 선택한 AWS 리전로 데이터 복제를 수행하도록 서비스 연결 역할이 자동으로 생성됩니다.

  서비스 연결 역할에 대한 자세한 내용은 the section called “서비스 연결 역할 사용” 섹션을 참조하세요.
- `AWS KMS` - 보안 주체가 AWS KMS에 구성된 키를 볼 수 있도록 허용합니다. 이는 사용자가 자신의 계정에서 생성하고 관리하는 AWS KMS 키를 보고 Amazon Keyspaces에 할당된 키가 활성화된 대칭 암호화 키인지 확인할 수 있도록 하기 위해 필요합니다.
- `Amazon EC2` - VPC 엔드포인트를 통해 Amazon Keyspaces에 연결하는 보안 주체가 Amazon EC2 인스턴스의 VPC에 엔드포인트 및 네트워크 인터페이스 정보를 쿼리할 수 있도록 허용합니다. Amazon Keyspaces가 연결 로드 밸런싱에 사용되는 `system.peers` 테이블에서 사용 가능한 인터페이스 VPC 엔드포인트를 조회하고 저장할 수 있으려면 Amazon EC2 인스턴스에 대한 이러한 읽기 전용 액세스가 필요합니다.

정책을 `JSON` 형식으로 검토하려면 AmazonKeyspacesFullAccess를 참조하세요.

AWS 관리형 정책에 대한 Amazon Keyspaces 업데이트

이 서비스가 이러한 변경 내용을 추적하기 시작한 이후부터 Amazon Keyspaces의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 사용 설명서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
AmazonKeyspacesFullAccess - 기존 정책에 대한 업데이트	Amazon Keyspaces는 인터페이스 VPC 엔드포인트를 통해 Amazon Keyspaces에 연결하는 클라이언트가 Amazon EC2 인스턴스에 액세스하여 네트워크 정보를 조회할 수 있는 새로운 읽기 전용 권한을 추가했습니다. Amazon Keyspaces는 연결 로드 밸런싱을 위해 사용 가능한 인터페이스 VPC 엔드포인트를 `system.peers` 테이블에 저장합니다. 자세한 내용은 the section called “인터페이스 VPC 엔드포인트 사용” 섹션을 참조하세요.	2023년 10월 3일
AmazonKeyspacesReadOnlyAccess_v2 – 새 정책	Amazon Keyspaces는 인터페이스 VPC 엔드포인트를 통해 Amazon Keyspaces에 연결하는 클라이언트가 Amazon EC2 인스턴스에 액세스하여 네트워크 정보를 조회할 수 있는 새로운 읽기 전용 권한을 추가하는 새 정책을 생성했습니다. Amazon Keyspaces는 연결 로드 밸런싱을 위해 사용 가능한 인터페이스 VPC 엔드포인트를 `system.peers` 테이블	2023년 9월 12일

변경 사항	설명	날짜
	에 저장합니다. 자세한 내용은 the section called “인터페이스 VPC 엔드포인트 사용” 섹션을 참조하세요.	
AmazonKeyspacesFullAccess - 기존 정책에 대한 업데이트	Amazon Keyspaces는 관리자가 다중 리전 키스페이스를 생성할 때 Amazon Keyspaces가 서비스 연결 역할을 생성할 수 있도록 새로운 권한을 추가했습니다. Amazon Keyspaces는 서비스 연결 역할을 사용하여 사용자를 대신하여 데이터 복제 작업을 수행합니다. 자세한 내용은 the section called “다중 리전 복제” 섹션을 참조하세요.	2023년 6월 5일
AmazonKeyspacesReadOnlyAccess – 기존 정책에 대한 업데이트	Amazon Keyspaces는 사용자가 CloudWatch를 사용하여 청구 가능한 테이블 크기를 볼 수 있는 새로운 권한을 추가했습니다. Amazon Keyspaces는 Amazon CloudWatch와 통합되어 청구 가능한 테이블 크기를 모니터링할 수 있습니다. 자세한 내용은 the section called “Amazon Keyspaces 지표 및 차원” 섹션을 참조하세요.	2022년 7월 7일

변경 사항	설명	날짜
AmazonKeyspacesFullAccess - 기존 정책에 대한 업데이트	Amazon Keyspaces는 사용자가 CloudWatch를 사용하여 청구 가능한 테이블 크기를 볼 수 있는 새로운 권한을 추가했습니다. Amazon Keyspaces는 Amazon CloudWatch와 통합되어 청구 가능한 테이블 크기를 모니터링할 수 있습니다. 자세한 내용은 the section called “Amazon Keyspaces 지표 및 차원” 섹션을 참조하세요.	2022년 7월 7일
AmazonKeyspacesReadOnlyAccess – 기존 정책에 대한 업데이트	Amazon Keyspaces는 Amazon Keyspaces 저장 시 암호화에 대해 구성된 AWS KMS 키를 사용자가 볼 수 있도록 하는 새로운 권한을 추가했습니다. Amazon Keyspaces 저장 시 암호화는 데이터를 저장 시 암호화하는 데 사용되는 암호화 키를 보호 및 관리하기 위해 AWS KMS와 통합됩니다. Amazon Keyspaces에 구성된 AWS KMS 키를 보기 위해 읽기 전용 권한이 추가되었습니다.	2021년 6월 1일

변경 사항	설명	날짜
AmazonKeyspacesFullAccess - 기존 정책에 대한 업데이트	Amazon Keyspaces는 Amazon Keyspaces 저장 시 암호화에 대해 구성된 AWS KMS 키를 사용자가 볼 수 있도록 하는 새로운 권한을 추가했습니다. Amazon Keyspaces 저장 시 암호화는 데이터를 저장 시 암호화하는 데 사용되는 암호화 키를 보호 및 관리하기 위해 AWS KMS와 통합됩니다. Amazon Keyspaces에 구성된 AWS KMS 키를 보기 위해 읽기 전용 권한이 추가되었습니다.	2021년 6월 1일
Amazon Keyspaces에서 변경 사항 추적 시작	Amazon Keyspaces가 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2021년 6월 1일

Amazon Keyspaces ID 및 액세스 문제 해결

다음 정보를 사용하여 Amazon Keyspaces 및 IAM으로 작업할 때 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

주제

- Amazon Keyspaces에서 작업을 수행할 권한이 없음
- IAM 사용자 또는 역할을 수정했는데 변경 사항이 즉시 적용되지 않음
- Amazon Keyspaces point-in-time 복구 (PITR) 를 사용하여 테이블을 복원할 수 없습니다.
- 저는 iam을 수행할 권한이 없습니다. PassRole
- 관리자인데, 다른 사용자가 Amazon Keyspaces에 액세스할 수 있게 허용하기를 원함
- 외부 사용자가 내 Amazon Keyspaces AWS 계정 리소스에 액세스할 수 있도록 허용하고 싶습니다.

Amazon Keyspaces에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 도움을 요청해야 합니다. 관리자는 사용자 이름과 비밀번호를 제공한 사람입니다.

다음 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 *###*에 대한 세부 정보를 보려고 하지만 테이블에 대한 `cassandra:`*`Select`* 권한이 없는 경우에 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 cassandra:Select on resource: mytable
```

이 경우 Mateo는 *`mytable`* 작업을 사용하여 `cassandra:`*`Select`* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

IAM 사용자 또는 역할을 수정했는데 변경 사항이 즉시 적용되지 않음

Amazon Keyspaces에 대한 기존 연결이 설정된 애플리케이션에 IAM 정책 변경 사항이 적용되는 데 최대 10분이 걸릴 수 있습니다. IAM 정책 변경 사항은 애플리케이션이 새 연결을 설정하는 즉시 적용됩니다. 기존 IAM 사용자 또는 역할을 수정했지만 즉시 적용되지 않는 경우 10분 동안 기다리거나 연결을 끊었다가 Amazon Keyspaces에 다시 연결합니다.

Amazon Keyspaces point-in-time 복구 (PITR) 를 사용하여 테이블을 복원할 수 없습니다.

Amazon Keyspaces 테이블을 PITR (point-in-time 복구 포함) 으로 복원하려고 할 때 복원 프로세스가 시작되었지만 성공적으로 완료되지 않은 경우 복원 프로세스에 필요한 모든 필수 권한을 구성하지 않았을 수 있습니다. 관리자에게 문의하여 Amazon Keyspaces 테이블 복원을 허용하는 정책을 업데이트하라고 관리자에게 요청해야 합니다.

사용자 권한 외에도 Amazon Keyspaces는 보안 주체를 대신하여 복원 프로세스 중에 작업을 수행할 수 있는 권한을 요구할 수 있습니다. 테이블이 고객 관리형 키로 암호화되거나 들어오는 트래픽을 제한하는 IAM 정책을 사용하는 경우가 이에 해당합니다. 예를 들어 IAM 정책에서 조건 키를 사용하여 소스 트래픽을 특정 엔드포인트 또는 IP 범위로 제한하는 경우 복원 작업이 실패합니다. Amazon Keyspaces가 보안 주체를 대신하여 테이블 복원 작업을 수행하도록 허용하려면 IAM 정책에 `aws:ViaAWSService` 글로벌 조건 키를 추가해야 합니다.

테이블 복원 권한에 대한 자세한 내용은 the section called “권한 복원” 섹션을 참조하세요.

저는 iam을 수행할 권한이 없습니다. PassRole

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon Keyspaces에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

새 서비스 역할 또는 서비스 연결 역할을 만드는 대신 기존 역할을 해당 서비스에 전달할 AWS 서비스 수 있는 기능도 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 Amazon Keyspaces에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 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, Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요하면 관리자에게 문의하세요. AWS 관리자는 로그인 보안 인증을 제공한 사람입니다.

관리자인데, 다른 사용자가 Amazon Keyspaces에 액세스할 수 있게 허용하기를 원함

다른 사용자가 Amazon Keyspaces에 액세스하도록 허용하려면 액세스 권한이 필요한 사용자 또는 애플리케이션에 대한 IAM 엔터티(사용자 또는 역할)를 생성해야 합니다. 다른 사용자들은 해당 엔터티에 대한 보안 인증을 사용해 AWS에 액세스합니다. 그런 다음 Amazon Keyspaces에서 올바른 권한을 부여하는 정책을 엔터티에 연결해야 합니다.

바로 시작하려면 IAM 사용 설명서의 첫 번째 IAM 위임 사용자 및 그룹 생성을 참조하십시오.

외부 사용자가 내 Amazon Keyspaces AWS 계정 리소스에 액세스할 수 있도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- Amazon Keyspaces에서 이러한 기능을 지원하는지 여부를 알아보려면 Amazon Keyspaces에서 IAM을 사용하는 방법 섹션을 참조하세요.

- 소유한 리소스에 대한 액세스 권한을 AWS 계정 부여하는 방법을 알아보려면 IAM 사용 설명서의 다른 AWS 계정 IAM 사용자에게 액세스 권한 제공을 참조하십시오.
- 제3자에게 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정AWS 계정 소유에 대한 액세스 제공을 참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(자격 증명 페더레이션)을 참조하십시오.
- 교차 계정 액세스에 대한 역할 사용과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스를 참조하십시오.

Amazon Keyspaces에 대해 서비스 연결 역할 사용

Amazon Keyspaces(Apache Cassandra용)는 AWS Identity and Access Management(IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon Keyspaces에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon Keyspaces에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

주제

- Amazon Keyspaces 애플리케이션 Auto Scaling에 역할 사용
- Amazon Keyspaces 다중 리전 복제에 역할 사용

Amazon Keyspaces 애플리케이션 Auto Scaling에 역할 사용

Amazon Keyspaces(Apache Cassandra용)는 AWS Identity and Access Management(IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon Keyspaces에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon Keyspaces에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon Keyspaces를 더 쉽게 설정할 수 있습니다. Amazon Keyspaces에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon Keyspaces만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제해야만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon Keyspaces 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는 AWS 서비스를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

Amazon Keyspaces에 대한 서비스 연결 역할 권한

Amazon Keyspaces는 지정된 서비스 연결 역할을 사용하여 Application AWSServiceRoleForApplicationAutoScaling_CassandraTableAuto Scaling이 사용자를 대신하여 Amazon Keyspace와 Amazon을 호출할 수 있도록 합니다. CloudWatch

AWSServiceRoleForApplicationAutoScaling_CassandraTable 서비스 연결 역할은 다음 서비스가 역할을 맡을 것으로 신뢰합니다.

- `cassandra.application-autoscaling.amazonaws.com`

역할 권한 정책은 Application Auto Scaling이 지정된 Amazon Keyspaces 리소스에서 다음 작업을 완료하도록 허용합니다.

- 작업: `arn:*:cassandra:*:*:/keyspace/system/table/*`에 대한 `cassandra:Select`
- 작업: `arn:*:cassandra:*:*:/keyspace/system_schema/table/*` 리소스에 대한 `cassandra:Select`
- 작업: `arn:*:cassandra:*:*:/keyspace/system_schema_mcs/table/*` 리소스에 대한 `cassandra:Select`
- 작업: `arn:*:cassandra:*:*:"*"` 리소스에 대한 `cassandra:Alter`

Amazon Keyspaces에 대한 서비스 연결 역할 생성

Amazon Keyspaces 자동 크기 조정을 위해 서비스 연결 역할을 수동으로 생성할 필요가 없습니다. AWS Management Console, CQLAWS CLI, 또는 AWS API를 사용하여 테이블에서 Amazon Keyspaces 자동 크기 조정을 활성화하면 Application Auto Scaling이 서비스 연결 역할을 자동으로 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 테이블에 대해 Amazon Keyspaces 자동 크기 조정을 활성화하면 Application Auto Scaling에서 서비스 연결 역할을 다시 생성합니다.

> **Important**
>
> 이러한 서비스 연결 역할은 해당 역할이 지원하는 기능을 사용하는 다른 서비스에서 작업을 완료했을 경우 계정에 나타날 수 있습니다. 자세한 내용은 내 AWS 계정에 표시되는 새 역할을 참조하십시오.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 테이블에 대해 Amazon Keyspaces 자동 애플리케이션 크기 조정을 활성화하면 Application Auto Scaling이 서비스 연결 역할을 다시 생성합니다.

Amazon Keyspaces에 대한 서비스 연결 역할 편집

Amazon Keyspaces에서는 AWSServiceRoleForApplicationAutoScaling_CassandraTable 서비스 연결 역할을 편집할 수 없습니다. 서비스 연결 역할을 생성한 후에는 다양한 객체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하십시오.

Amazon Keyspaces에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 이렇게 하면 적극적으로 모니터링하거나 유지 관리하지 않는 미사용 엔터티가 없게 됩니다. 그러나 서비스 연결 역할을 수동으로 삭제하려면 먼저 모든 AWS 리전에 걸쳐 계정의 모든 테이블에서 자동 크기 조정을 비활성화해야 합니다. Amazon Keyspaces 테이블에서 자동 크기 조정을 비활성화하려면 Amazon Keyspaces 자동 크기 조정 설정 수정 또는 비활성화를 참조하세요.

> **Note**
>
> 리소스를 수정하려고 할 때 Amazon Keyspaces 자동 크기 조정이 역할을 사용 중이면 등록 취소가 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하려면

IAM 콘솔AWS CLI, 또는 AWS API를 사용하여 서비스 연결 역할을 삭제하십시오. AWSServiceRoleForApplicationAutoScaling_CassandraTable 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하세요.

> **Note**
>
> Amazon Keyspaces 자동 크기 조정에서 사용하는 서비스 연결 역할을 삭제하려면 계정의 모든 테이블에서 자동 크기 조정을 비활성화해야 합니다.

Amazon Keyspaces 서비스 연결 역할이 지원되는 리전

Amazon Keyspaces는 서비스가 제공되는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은 Amazon Keyspaces의 서비스 엔드포인트를 참조하세요.

Amazon Keyspaces 다중 리전 복제에 역할 사용

Amazon Keyspaces(Apache Cassandra용)는 AWS Identity and Access Management(IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon Keyspaces에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 Amazon Keyspaces에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 Amazon Keyspaces를 더 쉽게 설정할 수 있습니다. Amazon Keyspaces에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의되지 않은 한, Amazon Keyspaces만 해당 역할을 수임할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔터티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제해야만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 실수로 삭제할 수 없기 때문에 Amazon Keyspaces 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용은 IAM으로 작업하는 AWS 서비스를 참조하고 서비스 연결 역할 열에 예가 있는 서비스를 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

Amazon Keyspaces에 대한 서비스 연결 역할 권한

Amazon Keyspaces는 이름이 지정된 서비스 연결 역할을 사용하여 AWSServiceRoleForAmazonKeyspacesReplicationAmazon Keyspaces가 사용자를 대신하여 다중 리전 테이블의 모든 복제본에 쓰기를 복제할 수 있도록 합니다.

AWSServiceRoleForAmazonKeyspacesReplication 서비스 연결 역할은 다음 서비스를 신뢰하여 역할을 수임합니다.

- `replication.cassandra.amazonaws.com`

이름이 지정된 역할 권한 정책을 KeyspacesReplicationServiceRolePolicy 통해 Amazon Keyspaces는 다음 작업을 완료할 수 있습니다.

- 작업: `cassandra:Select`
- 작업: `cassandra:SelectMultiRegionResource`
- 작업: `cassandra:Modify`
- 작업: `cassandra:ModifyMultiRegionResource`

Amazon Keyspaces 서비스 연결 역할은 정책의 지정된 Amazon 리소스 이름 (ARN) “arn: *”에 대해 “작업:” 권한을 제공하지만 Amazon Keyspace는 사용자 계정의 ARN을 AWSServiceRoleForAmazonKeyspacesReplication 제공합니다.

사용자, 그룹 또는 역할이 서비스 연결 역할을 생성, 편집 또는 삭제할 수 있도록 사용 권한을 구성해야 합니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 권한을 참조하십시오.

Amazon Keyspaces에 대한 서비스 연결 역할 생성

서비스 연결 역할을 수동으로 생성할 수 없습니다. AWS Management Console, AWS CLI 또는 AWS API에서 다중 리전 키스페이스를 생성하면 Amazon Keyspaces에서 서비스 연결 역할이 자동으로 생성됩니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 다중 리전 키스페이스를 생성하면 Amazon Keyspaces에서 서비스 연결 역할이 다시 생성됩니다.

Amazon Keyspaces에 대한 서비스 연결 역할 편집

Amazon Keyspaces에서는 AWSServiceRoleForAmazonKeyspacesReplication 서비스 연결 역할을 편집할 수 없습니다. 서비스 연결 역할을 생성한 후에는 다양한 객체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 편집을 참조하십시오.

Amazon Keyspaces에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 따라서 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 그러나 서비스 연결 역할을 수동으로 삭제하려면 먼저 모든 AWS 리전에 걸쳐 계정의 모든 다중 리전 키스케이스를 삭제해야 합니다.

서비스 연결 역할을 정리

IAM을 사용하여 서비스 연결 역할을 삭제하기 전에 먼저 역할에서 사용되는 다중 리전 키스페이스 및 테이블을 삭제해야 합니다.

> **Note**
>
> 리소스를 삭제하려고 할 때 Amazon Keyspaces 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하십시오.

AWSServiceRoleForAmazonKeyspacesReplication (콘솔) 에서 사용하는 Amazon Keyspaces 리소스를 삭제하려면

1. AWS Management Console에 로그인하고 Amazon Keyspaces 콘솔(https://console.aws.amazon.com/msk/home)을 엽니다.
2. 왼쪽 패널에서 Keyspaces를 선택합니다.
3. 목록에서 모든 다중 리전 키스페이스를 선택합니다.
4. 삭제를 선택하고 삭제를 확인한 다음 키스페이스 삭제를 선택합니다.

다음 방법 중 하나를 사용하여 프로그래밍 방식으로 다중 리전 키스페이스를 삭제할 수도 있습니다.

- Cassandra 쿼리 언어(CQL) ??? 문
- AWS CLI의 delete-keyspace 작업
- 아마존 키스페이스 API의 DeleteKeyspace작동.

수동으로 서비스 연결 역할 삭제

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 AWSServiceRoleForAmazonKeyspacesReplication 서비스 연결 역할을 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 서비스 연결 역할 삭제를 참조하세요.

Amazon Keyspaces 서비스 연결 역할이 지원되는 리전

Amazon Keyspaces에서는 서비스가 제공되는 모든 리전에서 서비스 연결 역할을 사용하도록 지원하지 않습니다. 다음 지역에서 AWSServiceRoleForAmazonKeyspacesReplication 역할을 사용할 수 있습니다.

지역명	리전 자격 증명	Amazon Keyspaces 지원
미국 동부(버지니아 북부)	us-east-1	예
미국 동부(오하이오)	us-east-2	예
미국 서부(캘리포니아 북부)	us-west-1	예
미국 서부(오레곤)	us-west-2	예
아시아 태평양(뭄바이)	ap-south-1	예
아시아 태평양(오사카)	ap-northeast-3	예
아시아 태평양(서울)	ap-northeast-2	예
아시아 태평양(싱가포르)	ap-southeast-1	예
아시아 태평양(시드니)	ap-southeast-2	예
아시아 태평양(도쿄)	ap-northeast-1	예
캐나다(중부)	ca-central-1	예
유럽(프랑크푸르트)	eu-central-1	예
유럽(아일랜드)	eu-west-1	예
유럽(런던)	eu-west-2	예
유럽(파리)	eu-west-3	예
남아메리카(상파울루)	sa-east-1	예
AWS GovCloud (미국 동부)	us-gov-east-1	아니요
AWS GovCloud (미국 서부)	us-gov-west-1	아니요

Amazon Keyspaces(Apache Cassandra용)의 규정 준수 검증

타사 감사자는 여러 규정 준수 프로그램의 일환으로 Amazon Keyspaces (Apache Cassandra용) 의 보안 및 규정 준수를 평가합니다. AWS 다음이 포함됩니다.

- ISO/IEC 27001:2013, 27017:2015, 27018:2019, 및 ISO/IEC 9001:2015. 자세한 내용은 AWS ISO 및 CSA STAR 인증 및 서비스를 참조하십시오.
- SOC(시스템 및 조직 제어)
- 신용카드 업계(PCI)
- 연방정부의 위험 및 인증 관리 프로그램(FedRAMP) 높음
- HIPAA(미국 건강 보험 양도 및 책임에 관한 법)

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면AWS 서비스 규정 준수 프로그램 범위 내 규정 준수 프로그램별 규정 준수 프로그램을 준수 프로그램을 선택하십시오. 일반 정보는 AWS 규정 준수 프로그램AWS 보증 프로그램 규정AWS 참조하십시오.

를 사용하여 AWS Artifact타사 감사 보고서를 다운로드할 수 있습니다. 자세한 내용은 의 보고서 https://docs.aws.amazon.com/artifact/latest/ug/downloading-documents.html 참조하십시오 AWS Artifact.

사용 시 규정 준수 AWS 서비스 책임은 데이터의 민감도, 회사의 규정 준수 목표, 관련 법률 및 규정에 따라 결정됩니다. AWS 규정 준수에 도움이 되는 다음 리소스를 제공합니다.

- 보안 및 규정 준수 퀵 스타트 가이드 - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 AWS 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다.
- Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 설계 — 이 백서에서는 기업이 HIPAA 적격 애플리케이션을 만드는 AWS 데 사용할 수 있는 방법을 설명합니다.

  > **Note**
  >
  > 모든 AWS 서비스 사람이 HIPAA 자격을 갖춘 것은 아닙니다. 자세한 내용은 HIPAA 적격 서비스 참조를 참조하십시오.

- AWS 규정 준수 리소스AWS — 이 워크북 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 — 규정 준수의 관점에서 공동 책임 모델을 이해하십시오. 이 가이드에서는 보안을 유지하기 위한 모범 사례를 AWS 서비스 요약하고 여러 프레임워크 (미국 표준 기술 연

구소 (NIST), 결제 카드 산업 보안 표준 위원회 (PCI), 국제 표준화기구 (ISO) 등) 에서 보안 제어에 대한 지침을 매핑합니다.

- AWS Config 개발자 안내서의 규칙을 사용하여 리소스 평가 — 이 AWS Config 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- AWS Security Hub— 이를 AWS 서비스 통해 내부 AWS보안 상태를 포괄적으로 파악할 수 있습니다. Security Hub는 보안 제어를 사용하여 AWS 리소스를 평가하고 보안 업계 표준 및 모범 사례에 대한 규정 준수를 확인합니다. 지원되는 서비스 및 제어 목록은 Security Hub 제어 참조를 참조하십시오.
- Amazon GuardDuty — 환경에 의심스럽고 악의적인 활동이 있는지 AWS 계정모니터링하여 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. GuardDuty 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족하여 PCI DSS와 같은 다양한 규정 준수 요구 사항을 해결하는 데 도움이 될 수 있습니다.
- AWS Audit Manager— 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 단순화할 수 있습니다.

Amazon Keyspaces의 복원성 및 재해 복구

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전에서는 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 대기 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

Amazon Keyspaces는 내구성과 고가용성을 위해 동일한 AWS 내 여러 AWS 리전 가용 영역에 데이터를 자동으로 세 번 복제합니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라 외에도 Amazon Keyspaces는 데이터 복원성과 백업 요구 사항을 지원하는 다양한 기능을 제공합니다.

다중 리전 복제

더 먼 지역적 거리를 두고 데이터 또는 애플리케이션을 복제해야 하는 경우 Amazon Keyspaces는 다중 리전 복제 기능을 제공합니다. Amazon Keyspaces 테이블을 원하는 AWS 리전로 최대 6개까지 복제할 수 있습니다. 자세한 내용은 다중 리전 복제 섹션을 참조하세요.

시점 복구(PITR)

PITR을 사용하면 테이블 데이터를 지속적으로 백업하여 우발적인 쓰기 또는 삭제 작업으로부터 Amazon Keyspaces 테이블을 보호할 수 있습니다. 자세한 내용은 Amazon Keyspaces의 시점 복구를 참조하세요.

Amazon Keyspaces의 인프라 보안

관리형 서비스인 Amazon Keyspaces(Apache Cassandra용)는 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 AWS 클라우드 보안을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 인프라 보호를 참조하세요.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Amazon Keyspaces에 액세스합니다. 클라이언트는 다음을 지원해야 합니다.

- 전송 계층 보안(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 ID 및 IAM 주체와 관련된 보안 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 자격 증명을 생성하여 요청에 서명할 수 있습니다.

Amazon Keyspaces는 클라이언트 요청을 인증하는 두 가지 방법을 지원합니다. 첫 번째 방법은 특정 IAM 사용자에 대해 생성된 암호 기반 보안 인증인 서비스별 보안 인증을 사용합니다. IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 암호를 생성하고 관리할 수 있습니다. 자세한 내용은 Amazon Keyspaces로 IAM 사용을 참조하세요.

두 번째 방법은 Cassandra용 오픈 소스 DataStax Java 드라이버에 대한 인증 플러그인을 사용하는 것입니다. 이 플러그인을 사용하면 IAM 사용자, 역할 및 페더레이션 ID가 AWS 서명 버전 4 프로세스(SigV4)를 사용하여 Amazon Keyspaces(Apache Cassandra용) API 요청에 인증 정보를 추가할 수 있습니다. 자세한 내용은 the section called “인증을 위한 IAM 자격 증명 AWS ” 섹션을 참조하세요.

인터페이스 VPC 엔드포인트를 사용하여 Amazon 네트워크에서 Amazon VPC 및 Amazon Keyspaces 간의 트래픽을 유지할 수 있습니다. 인터페이스 VPC 엔드포인트는 Amazon VPC에서 프라이빗 IP 주

소가 있는 탄력적 네트워크 인터페이스를 사용하여 AWS 서비스 간 프라이빗 통신을 사용할 수 있는 AWS 기술인 AWS PrivateLink에 의해 구동됩니다. 자세한 내용은 the section called “인터페이스 VPC 엔드포인트 사용” 섹션을 참조하세요.

인터페이스 VPC 엔드포인트에서 Amazon Keyspaces 사용

인터페이스 VPC 엔드포인트를 사용하면 Amazon VPC에서 실행되는 Virtual Private Cloud(VPC)와 Amazon Keyspaces 간에 프라이빗 통신이 가능해집니다. 인터페이스 VPC 엔드포인트는 VPC와 서비스 간의 개인 통신을 가능하게 하는 AWS 서비스인 에 의해 AWS PrivateLink구동됩니다. AWS

AWS PrivateLink VPC에 프라이빗 IP 주소가 있는 Elastic Network 인터페이스를 사용하여 네트워크 트래픽이 Amazon 네트워크를 벗어나지 않도록 함으로써 이를 가능하게 합니다. 인터페이스 VPC 엔드포인트에는 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결이 필요하지 않습니다. 자세한 내용은 Amazon Virtual Private Cloud 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

주제

- Amazon Keyspaces에 인터페이스 VPC 엔드포인트 사용
- 인터페이스 VPC 엔드포인트 정보로 system.peers 테이블 항목 채우기
- Amazon Keyspaces의 인터페이스 VPC 엔드포인트에 대한 액세스 제어
- 가용성
- VPC 엔드포인트 정책 및 아마존 키스페이스 point-in-time 복구 (PITR)
- 일반적인 오류 및 경고

Amazon Keyspaces에 인터페이스 VPC 엔드포인트 사용

Amazon Keyspaces와 Amazon VPC 리소스 간의 트래픽이 인터페이스 VPC 엔드포인트를 통해 흐르기 시작하도록 인터페이스 VPC 엔드포인트를 생성할 수 있습니다. 시작하려면 단계에 따라 인터페이스 엔드포인트를 생성합니다. 다음으로 이전 단계에서 생성한 엔드포인트와 연결된 보안 그룹을 편집하고 포트 9142에 대한 인바운드 규칙을 구성합니다. 자세한 내용은 규칙 추가, 제거 및 업데이트를 참조하세요.

VPC 엔드포인트를 통해 Amazon Keyspace에 대한 연결을 구성하는 방법에 대한 step-by-step 자습서는 을 참조하십시오. the section called “VPC 엔드포인트와 연결” VPC에 있는 AWS 계정 서로 다른 애플리케이션과 분리된 Amazon Keyspaces 리소스에 대한 계정 간 액세스를 구성하는 방법을 알아보려면 을 참조하십시오. the section called “크로스 계정 액세스”

인터페이스 VPC 엔드포인트 정보로 **system.peers** 테이블 항목 채우기

Apache Cassandra 드라이버는 `system.peers` 테이블을 사용하여 클러스터에 대한 노드 정보를 쿼리합니다. Cassandra 드라이버는 노드 정보를 사용하여 연결 부하를 분산하고 작업을 재시도합니다. Amazon Keyspaces는 퍼블릭 엔드포인트를 통해 연결하는 클라이언트에 대해 `system.peers` 테이블에 9개 항목을 자동으로 채웁니다.

인터페이스 VPC 엔드포인트를 통해 연결하는 클라이언트에게 유사한 기능을 제공하기 위해 Amazon Keyspaces는 VPC 엔드포인트를 사용할 수 있는 각 가용 영역에 대한 항목으로 계정의 `system.peers` 테이블을 채웁니다. `system.peers` 테이블에서 사용 가능한 인터페이스 VPC 엔드포인트를 조회하고 저장하려면 Amazon Keyspaces에서는 Amazon Keyspaces에 연결하는 데 사용되는 IAM 엔터티에 엔드포인트 및 네트워크 인터페이스 정보에 대해 VPC를 쿼리할 수 있는 액세스 권한을 부여해야 합니다.

> **Important**
>
> 사용 가능한 인터페이스 VPC 엔드포인트로 `system.peers` 테이블을 채우면 로드 밸런싱이 개선되고 읽기/쓰기 처리량이 증가합니다. 인터페이스 VPC 엔드포인트를 사용하여 Amazon Keyspaces에 액세스하는 모든 클라이언트에게 권장되며 Apache Spark의 경우 필수입니다.

Amazon Keyspaces에 연결하는 데 사용되는 IAM 엔터티에 필요한 인터페이스 VPC 엔드포인트 정보를 조회할 수 있는 권한을 부여하려면 다음 예와 같이 기존 IAM 역할 또는 사용자 정책을 업데이트하거나 새 IAM 정책을 생성할 수 있습니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListVPCEndpoints",
         "Effect":"Allow",
         "Action":[
            "ec2:DescribeNetworkInterfaces",
            "ec2:DescribeVpcEndpoints"
         ],
         "Resource":"*"
      }
   ]
}
```

> **Note**
>
> 관리형 정책 `AmazonKeyspacesReadOnlyAccess_v2` 및 `AmazonKeyspacesFullAccess`에는 Amazon Keyspaces가 Amazon EC2 인스턴스에 액세스하여 사용 가능한 인터페이스 VPC 엔드포인트에 대한 정보를 읽을 수 있도록 하는 데 필요한 권한이 포함되어 있습니다.

정책이 올바르게 설정되었는지 확인하려면 `system.peers` 테이블을 쿼리하여 네트워킹 정보를 확인합니다. `system.peers` 테이블이 비어 있으면 정책이 성공적으로 구성되지 않았거나 `DescribeNetworkInterfaces` 및 `DescribeVPCEndpoints` API 작업에 대한 요청 속도 할당량을 초과했음을 의미할 수 있습니다. `DescribeVPCEndpoints`는 `Describe*` 범주에 속하며 변경 불가 작업으로 간주됩니다. `DescribeNetworkInterfaces`는 필터링되지 않은 작업과 페이지가 지정되지 않은 변경 불가 작업의 하위 집합에 속하며 다른 할당량이 적용됩니다. 자세한 내용은 Amazon EC2 API 참조의 요청 토큰 버킷 크기 및 리필 속도를 참조하세요.

빈 테이블이 보이면 몇 분 후에 다시 시도하여 요청 속도 할당량 문제를 배제합니다. VPC 엔드포인트를 올바르게 구성했는지 확인하려면 the section called “VPC 엔드포인트 연결 오류” 섹션을 참조하세요. 쿼리가 테이블의 결과를 반환하면 정책이 올바르게 구성된 것입니다.

Amazon Keyspaces의 인터페이스 VPC 엔드포인트에 대한 액세스 제어

VPC 엔드포인트 정책을 사용하면 다음 두 가지 방법으로 리소스에 대한 액세스를 제어할 수 있습니다.

- IAM 정책 - 사용자는 특정 VPC 엔드포인트를 통해 Amazon Keyspaces에 액세스하는 데 허용되는 요청, 사용자 또는 그룹을 제어할 수 있습니다. IAM 사용자, 그룹 또는 역할에 연결된 정책의 조건 키를 사용하여 이를 수행할 수 있습니다.
- VPC 정책 - 정책을 연결하여 Amazon Keyspaces 리소스에 액세스할 수 있는 VPC 엔드포인트를 제어할 수 있습니다. 특정 VPC 엔드포인트를 통해 들어오는 트래픽만 허용하도록 특정 키스페이스 또는 테이블에 대한 액세스를 제한하려면 리소스 액세스를 제한하는 기존 IAM 정책을 편집하고 해당 VPC 엔드포인트를 추가합니다.

다음은 Amazon Keyspaces 리소스에 액세스하기 위한 엔드포인트 정책의 예입니다.

- IAM 정책 예: 트래픽이 지정된 VPC 엔드포인트에서 들어오지 않는 한 특정 Amazon Keyspaces 테이블에 대한 모든 액세스 제한 - 이 샘플 정책은 IAM 사용자, 역할 또는 그룹에 연결할 수 있습니다.

들어오는 트래픽이 지정된 VPC 엔드포인트에서 시작되지 않는 한 지정된 Amazon Keyspaces 테이블에 대한 액세스를 제한합니다.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "UserOrRolePolicyToDenyAccess",
         "Action": "cassandra:*",
         "Effect": "Deny",
         "Resource": [
                        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
mykeyspace/table/mytable",
                        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
           ],
         "Condition": { "StringNotEquals" : { "aws:sourceVpce": "vpce-abc123" } }
      }
   ]
}
```

> **Note**
>
> 특정 테이블에 대한 액세스를 제한하려면 시스템 테이블에 대한 액세스 권한도 포함해야 합니다. 시스템 테이블은 읽기 전용입니다.

- VPC 정책 예제: 읽기 전용 액세스 - 이 샘플 정책은 VPC 엔드포인트에 연결할 수 있습니다. (자세한 내용은 Amazon VPC 리소스에 대한 액세스 제어를 참조하세요.) 연결된 VPC 엔드포인트를 통해 Amazon Keyspaces 리소스에 대한 읽기 전용 액세스로 작업을 제한합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadOnly",
      "Principal": "*",
      "Action": [
        "cassandra:Select"
      ],
      "Effect": "Allow",
      "Resource": "*"
    }
```

```
  ]
}
```

- VPC 정책 예: 특정 Amazon Keyspaces 테이블에 대한 액세스 제한 - 이 샘플 정책은 VPC 엔드포인트에 연결할 수 있습니다. 연결된 VPC 엔드포인트를 통한 특정 테이블에 대한 액세스를 제한합니다.

```
{
   "Version": "2012-10-17",
   "Statement": [
        {
            "Sid": "RestrictAccessToTable",
            "Principal": "*",
            "Action": "cassandra:*",
            "Effect": "Allow",
            "Resource": [
                        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/
mykeyspace/table/mytable",
                        "arn:aws:cassandra:us-east-1:111122223333:/keyspace/system*"
           ]
        }
   ]
}
```

> **Note**
>
> 특정 테이블에 대한 액세스를 제한하려면 시스템 테이블에 대한 액세스 권한도 포함해야 합니다. 시스템 테이블은 읽기 전용입니다.

가용성

Amazon Keyspace는 서비스가 제공되는 모든 AWS 리전 곳에서 인터페이스 VPC 엔드포인트를 사용할 수 있도록 지원합니다. 자세한 정보는 ???을 참조하세요.

VPC 엔드포인트 정책 및 아마존 키스페이스 point-in-time 복구 (PITR)

IAM 정책을 조건 키와 함께 사용하여 들어오는 트래픽을 제한하는 경우 테이블 복원 작업이 실패할 수 있습니다. 예를 들어 `aws:SourceVpce` 조건 키를 사용하여 소스 트래픽을 특정 VPC 엔드포인트로 제한하는 경우 테이블 복원 작업이 실패합니다. Amazon Keyspaces가 보안 주체를 대신하여 테이블 복원 작업을 수행하도록 허용하려면 IAM 정책에 `aws:ViaAWSService` 조건 키를 추가해야 합니다. `aws:ViaAWSService`조건 키는 모든 AWS 서비스가 보안 주체의 자격 증명을 사용하여 요청할 때 액

세스를 허용합니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건 키를 참조하세요. 다음은 이에 대한 정책 예제입니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"CassandraAccessForVPCE",
         "Effect":"Allow",
         "Action":"cassandra:*",
         "Resource":"*",
         "Condition":{
            "Bool":{
               "aws:ViaAWSService":"false"
            },
            "StringEquals":{
               "aws:SourceVpce":[
                  "vpce-12345678901234567"
               ]
            }
         }
      },
      {
         "Sid":"CassandraAccessForAwsService",
         "Effect":"Allow",
         "Action":"cassandra:*",
         "Resource":"*",
         "Condition":{
            "Bool":{
               "aws:ViaAWSService":"true"
            }
         }
      }
   ]
}
```

일반적인 오류 및 경고

Amazon Virtual Private Cloud를 사용하고 있고 Amazon Keyspaces에 연결하는 경우 다음 경고가 표시될 수 있습니다.

```
Control node cassandra.us-east-1.amazonaws.com/1.111.111.111:9142 has an entry
 for itself in system.peers: this entry will be ignored. This is likely due to a
 misconfiguration;
please verify your rpc_address configuration in cassandra.yaml on all nodes in your
 cluster.
```

이 경고는 연결된 Amazon VPC 엔드포인트를 포함하여 Amazon Keyspaces가 볼 수 있는 권한을 가진 모든 Amazon VPC 엔드포인트에 대한 항목이 `system.peers` 테이블에 포함되어 있기 때문에 발생합니다. 이 경고는 무시해도 됩니다.

기타 오류는 the section called “VPC 엔드포인트 연결 오류” 섹션을 참조하세요.

Amazon Keyspaces의 구성 및 취약성 분석

AWS가 게스트 운영 체제(OS), 데이터베이스 패치, 방화벽 구성, 재해 복구 등의 기본 보안 작업을 처리합니다. 적합한 제3자가 이 절차를 검토하고 인증하였습니다. 자세한 내용은 다음 리소스를 참조하세요.

- 공동 책임 모델
- Amazon Web Services: 보안 프로세스의 개요(백서)

Amazon Keyspaces의 보안 모범 사례

Amazon Keyspaces(Apache Cassandra용)는 자체 보안 정책을 개발하고 구현할 때 고려해야 할 여러 보안 기능을 제공합니다. 다음 모범 사례는 일반적인 지침이며 완벽한 보안 솔루션을 나타내지는 않습니다. 이러한 모범 사례는 환경에 적절하지 않거나 충분하지 않을 수 있으므로 참고용으로만 사용해 주세요.

주제

- Amazon Keyspaces의 예방적 보안 모범 사례
- Amazon Keyspaces의 탐지 보안 모범 사례

Amazon Keyspaces의 예방적 보안 모범 사례

다음 보안 모범 사례는 Amazon Keyspaces의 보안 사고를 예측하고 방지하는 데 도움이 되기 때문에 예방적인 것으로 간주됩니다.

저장 시 암호화 사용

Amazon Keyspaces는 AWS Key Management Service(AWS KMS)에 저장된 암호화 키를 사용하여 테이블에 저장된 모든 사용자 데이터를 저장 시 암호화합니다. 이를 통해 기본 스토리지에 대한 무단 액세스로부터 데이터를 보호하여 데이터 보호 계층을 추가로 제공합니다.

기본적으로 Amazon Keyspaces는 AWS 소유 키를 사용하여 모든 테이블을 암호화합니다. 이 키가 없는 경우 자동으로 생성됩니다. 서비스 기본 키는 비활성화할 수 없습니다.

또는 저장 시 암호화에 고객 관리형 키를 사용할 수 있습니다. 자세한 내용은 Amazon Keyspaces 저장 시 암호화를 참조하세요.

IAM 역할을 사용하여 Amazon Keyspaces에 대한 액세스 인증

사용자, 애플리케이션, 기타 AWS 서비스는 Amazon Keyspaces에 액세스하려면 AWS API 요청에 유효한 AWS 보안 인증이 있어야 합니다. AWS 보안 인증을 애플리케이션이나 EC2 인스턴스에 직접 저장해서는 안 됩니다. 이러한 보안 인증은 자동으로 교체되지 않기 때문에 손상된 경우 비즈니스에 큰 영향을 줄 수 있는 장기 보안 인증입니다. IAM 역할을 사용하면 AWS 서비스 및 리소스에 액세스하는 데 사용할 수 있는 임시 액세스 키를 얻을 수 있습니다.

자세한 내용은 IAM 역할을 참조하세요.

IAM 정책을 사용한 Amazon Keyspaces 기본 인증

권한을 부여하려면 권한을 부여 받을 사용자, 권한을 행사할 수 있는 대상이 되는 Amazon Keyspaces API, 해당 리소스에 허용하고자 하는 특정 작업을 결정합니다. 최소 권한을 구현하는 것이 오류 또는 악의적인 의도로 인해 발생할 수 있는 보안 위험과 영향을 줄일 수 있는 비결입니다.

IAM 자격 증명(즉 사용자, 그룹 및 역할)에 권한 정책을 연결함으로써 Amazon Keyspaces 리소스에서 작업을 수행할 수 있는 권한을 부여합니다.

이를 위해 다음 정책을 사용할 수 있습니다.

- AWS 관리형(미리 정의된) 정책
- 고객 관리형 정책

IAM 정책 조건을 사용하여 세분화된 액세스 제어 구현

Amazon Keyspaces에서 권한을 부여할 때 권한 정책이 적용되는 방식을 결정하는 조건을 지정할 수 있습니다. 최소 권한을 구현하는 것이 오류 또는 악의적인 의도로 인해 발생할 수 있는 보안 위험과 영향을 줄일 수 있는 비결입니다.

IAM 정책을 사용해 권한을 부여할 때 조건을 지정할 수 있습니다. 예를 들어 다음을 수행할 수 있습니다.

- 사용자에게 특정 키스페이스 또는 테이블에 대한 읽기 전용 액세스를 허용하는 권한을 부여합니다.
- 사용자에게 해당 사용자의 ID를 기반으로 특정 테이블에 대한 쓰기 액세스를 허용하는 권한을 부여합니다.

자세한 내용은 ID 기반 정책 예제를 참조하세요.

클라이언트 측 암호화 참조

Amazon Keyspaces에 중요 데이터나 기밀 데이터를 저장하는 경우 해당 데이터가 수명 주기 내내 보호되도록 최대한 원본에 가깝게 암호화할 수 있습니다. 전송 중 및 유휴 상태의 중요 데이터를 암호화하면 일반 텍스트 데이터를 제3자가 사용할 수 없게 하는 데 도움이 됩니다.

Amazon Keyspaces의 탐지 보안 모범 사례

다음 보안 모범 사례는 잠재적인 보안 약점과 사고를 탐지하는 데 도움이 되기 때문에 탐지적인 것으로 간주됩니다.

AWS CloudTrail을 사용하여 AWS Key Management Service(AWS KMS) AWS KMS 키 사용 모니터링

저장 시 암호화를 위해 고객 관리형 AWS KMS 키를 사용하고 있는 경우 이 키의 사용량은 AWS CloudTrail로 로깅됩니다. CloudTrail은 계정에서 수행한 작업을 기록함으로써 사용자 활동에 대한 가시성을 제공합니다. CloudTrail은 요청한 사용자, 사용한 서비스, 수행한 작업, 작업에 대한 파라미터, AWS 서비스가 반환하는 응답 요소 등 각 작업에 대한 중요 정보를 기록합니다. 이러한 정보는 AWS 리소스의 변경 사항을 추적하고 운영 관련 문제를 해결하는 데 도움이 됩니다. CloudTrail을 이용하면 내부 정책 및 규제 표준 준수를 더 쉽게 보장할 수 있습니다.

CloudTrail을 사용해 키 사용을 감사할 수 있습니다. CloudTrail은 계정의 AWS API 호출 및 관련 이벤트 내역이 포함된 로그 파일을 생성합니다. 이 로그 파일은 콘솔, AWS SDK 및 명령줄 도구를 사용하여 이루어진 모든 AWS KMS API 요청 외에도 통합된 AWS 서비스를 통해 이루어진 요청도 포함합니다. 이러한 로그 파일을 사용하여 AWS KMS 키가 사용된 시간, 요청되었던 작업, 요청자의 ID, 요청이 시작된 IP 주소 등에 대한 정보를 얻을 수 있습니다. 자세한 내용은 AWS CloudTrail을 사용하여 AWS Key Management Service API 호출 로깅 및 AWS CloudTrail 사용 설명서를 참조하세요.

CloudTrail을 사용하여 Amazon Keyspaces의 데이터 정의 언어(DDL) 작업을 모니터링합니다.

CloudTrail은 계정에서 수행한 작업을 기록함으로써 사용자 활동에 대한 가시성을 제공합니다. CloudTrail은 요청한 사용자, 사용한 서비스, 수행한 작업, 작업에 대한 파라미터, AWS 서비스가 반환하는 응답 요소 등 각 작업에 대한 중요 정보를 기록합니다. 이러한 정보는 AWS 리소스의 변경

사항을 추적하고 운영 관련 문제를 해결하는 데 도움이 됩니다. CloudTrail을 이용하면 내부 정책 및 규제 표준 준수를 더 쉽게 보장할 수 있습니다.

모든 Amazon Keyspaces DDL 작업은 CloudTrail에 자동으로 로깅됩니다. DDL 작업은 Amazon Keyspaces 키스페이스와 테이블을 생성하고 관리하도록 해 줍니다.

Amazon Keyspaces에서 활동이 수행되면 해당 활동은 이벤트 기록에 있는 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. 자세한 내용은 AWS CloudTrail을 사용하여 Amazon Keyspaces 작업 로깅을 참조하세요. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 AWS CloudTrail 사용 설명서에서 CloudTrail 이벤트 기록을 사용하여 이벤트 보기를 참조하세요.

Amazon Keyspaces의 이벤트를 포함하여 AWS 계정에 이벤트를 지속적으로 기록하려면 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon Simple Storage Service(S3) 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 S3 버킷으로 로그 파일을 전송합니다. 또는 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 처리하도록 다른 AWS 서비스를 구성할 수 있습니다.

식별 및 자동화를 위해 Amazon Keyspaces 리소스에 태그 지정하기

AWS 리소스에 태그 형태로 메타데이터를 지정할 수 있습니다. 각 태그는 리소스 관리, 검색 및 필터링을 더 수월하게 해줄 수 있는 옵션 값과 고객이 정의한 키로 구성된 간단한 레이블입니다.

태그를 지정하면 그룹화 제어를 구현할 수 있습니다. 고유한 태그 유형은 없지만 목적, 소유자, 환경 또는 기타 기준에 따라 리소스를 분류할 수 있습니다. 다음은 몇 가지 예입니다.

- 액세스 - 태그를 기반으로 Amazon Keyspaces 리소스에 대한 액세스를 제어하는 데 사용됩니다. 자세한 내용은 the section called “Amazon Keyspaces 태그 기반 권한 부여” 섹션을 참조하세요.
- 보안 - 데이터 보호 설정과 같은 요구 사항을 결정하는 데 사용됩니다.
- 기밀성 – 리소스가 지원하는 특정 데이터 기밀성 수준에 대한 식별자입니다.
- 환경 - 개발, 테스트 및 프로덕션 인프라 간 구별에 사용됩니다.

자세한 내용은 AWS 태깅 전략 및 리소스에 태그 및 레이블 추가를 참조하세요.

Amazon Keyspaces(Apache Cassandra용)에 대한 CQL 언어 참조

Amazon Keyspaces(Apache Cassandra용) 엔드포인트에 연결한 후에는 Cassandra 쿼리 언어(CQL)를 사용하여 데이터베이스 작업을 합니다. CQL은 여러 면에서 구조화 쿼리 언어(SQL)와 비슷합니다.

주제

Amazon Keyspaces의 Cassandra 쿼리 언어(CQL) 요소

식별자, 상수, 용어, 데이터 유형을 포함하여 Amazon Keyspaces에서 지원하는 Cassandra 쿼리 언어(CQL) 요소에 대해 알아보세요.

주제

식별자

식별자(또는 이름)는 테이블, 열, 기타 객체를 식별하는 데 사용됩니다. 식별자를 인용하거나 인용하지 않을 수 있습니다. 다음이 적용됩니다.

```
identifier          ::=  unquoted_identifier | quoted_identifier
unquoted_identifier ::=  re('[a-zA-Z][a-zA-Z0-9_]*')
quoted_identifier   ::=  '"' (any character where " can appear if doubled)+ '"'
```

상수

다음 상수가 정의됩니다.

```
constant ::=  string | integer | float | boolean | uuid | blob | NULL
string   ::=  '\'' (any character where ' can appear if doubled)+ '\''
              '$$' (any character other than '$$') '$$'
integer  ::=  re('-?[0-9]+')
float    ::=  re('-?[0-9]+(\.[0-9]*)?([eE][+-]?[0-9+])?') | NAN | INFINITY
boolean  ::=  TRUE | FALSE
uuid     ::=  hex{8}-hex{4}-hex{4}-hex{4}-hex{12}
hex      ::=  re("[0-9a-fA-F]")
blob     ::=  '0' ('x' | 'X') hex+
```

용어

용어는 지원되는 값의 종류를 나타냅니다. 용는 다음에 따라 정의됩니다.

```
term                ::=  constant | literal | function_call | arithmetic_operation |
 type_hint | bind_marker
literal             ::=  collection_literal | tuple_literal
function_call       ::=  identifier '(' [ term (',' term)* ] ')'
arithmetic_operation ::=  '-' term | term ('+' | '-' | '*' | '/' | '%') term
```

데이터 타입

Amazon Keyspaces는 다음 데이터 형식을 지원합니다.

문자열 형식

데이터 유형	설명
`ascii`	ASCII 문자열을 나타냅니다.
`text`	UTF-8 인코딩 문자열을 나타냅니다.
`varchar`	UTF-8 인코딩 문자열(`varchar`은 `text`의 별칭)을 나타냅니다.

숫자형

데이터 유형	설명
bigint	64비트 서명 길이를 나타냅니다.
counter	64비트 서명 정수 카운터를 나타냅니다. 자세한 정보는 the section called “카운터”을 참조하세요.
decimal	가변 정밀도 십진수를 나타냅니다.
double	64비트 IEEE 754 부동 소수점을 나타냅니다.
float	32비트 IEEE 754 부동 소수점을 나타냅니다.
int	32비트 서명 정수를 나타냅니다.
varint	임의 정밀도의 정수를 나타냅니다.

카운터

counter 열에는 64비트 서명 정수가 포함됩니다. 카운터 값은 the section called “UPDATE” 문을 사용하여 높이거나 내릴 수 있으며 직접적으로 설정할 수 없습니다. 따라서 counter 열은 카운트를 추적하는 데 유용합니다. 예를 들어 카운터를 사용하여 로그 파일의 항목 수 또는 소셜 네트워크에서 게시물 조회수를 추적할 수 있습니다. counter 열에 적용되는 제한은 다음과 같습니다.

- counter 유형의 열은 테이블 primary key의 일부가 될 수 없습니다.
- counter 유형 열이 하나 이상 포함된 테이블의 경우 해당 테이블의 모든 열이 counter 유형이어야 합니다.

카운터 업데이트가 실패하는 경우(예: 시간 초과 또는 Amazon Keyspace와의 연결 끊김) 클라이언트는 카운터 값이 업데이트되었는지 여부를 알 수 없습니다. 업데이트가 다시 시도되면 카운터 값에 대한 업데이트가 다시 적용될 수 있습니다.

Blob 유형

데이터 유형	설명
`blob`	임의의 바이트를 나타냅니다.

부울 유형

데이터 유형	설명
`boolean`	`true` 또는 `false`를 나타냅니다.

시간 관련 유형

데이터 유형	설명
`timestamp`	epoch(1970년 1월 1일 00:00:00 GMT) 이후의 날짜 및 시간을 밀리초 단위로 나타내는 64비트 서명 정수.
`timeuuid`	버전 1 UUID를 나타냅니다.

컬렉션 유형

데이터 유형	설명
`list`	정렬된 리터럴 요소 모음을 나타냅니다.
`map`	정렬되지 않은 키-값 페어 모음을 나타냅니다.
`set`	하나 이상의 리터럴 요소의 순서가 지정되지 않은 모음을 나타냅니다.

모음 유형 다음에 다른 데이터 유형(예: `TEXT` 또는 `INT`)을 각괄호로 묶어 집합 열을 선언합니다. 다음 예제와 같이 `TEXT`의 `SET`이 있는 열을 만들거나 `TEXT`의 `MAP`과 `INT` 키-값 쌍을 만들 수 있습니다.

```
SET <TEXT>
MAP <TEXT, INT>
```

비프로즌 수집을 사용하면 각 개별 수집 요소를 업데이트할 수 있습니다. 개별 요소에 대해 클라이언트 측 타임스탬프 및 Time to Live(TTL) 설정이 저장됩니다.

수집 유형에 FROZEN 키워드를 사용하면 수집의 값이 변경할 수 없는 단일 값으로 직렬화되며 Amazon Keyspaces는 이를 BLOB처럼 취급합니다. 이는 프로즌 수집입니다. INSERT 또는 UPDATE 문은 프로즌 수집 전체를 덮어씁니다. 프로즌 수집 내의 개별 요소는 업데이트할 수 없습니다.

클라이언트 측 타임스탬프와 Time to Live(TTL) 설정은 개별 요소가 아닌 프로즌 수집 전체에 적용됩니다. Frozen 수집 열은 PRIMARY KEY 테이블의 일부일 수 있습니다.

프로즌 수집을 중첩할 수 있습니다. 예를 들어 다음 예제와 같이 MAP이 FROZEN 키워드를 사용하는 경우 SET에서 MAP를 정의할 수 있습니다.

```
SET <FROZEN> <MAP <TEXT, INT>>>
```

Amazon Keyspaces는 기본적으로 최대 5개 수준의 프로즌 수집 중첩을 지원합니다. 자세한 정보는 the section called “Amazon Keyspaces 서비스 할당량”을 참조하세요. Apache Cassandra와의 기능적 차이에 대한 자세한 내용은 the section called “FROZEN 컬렉션” 섹션을 참조하십시오. CQL 구문에 대한 자세한 내용은 the section called “CREATE TABLE” 및 the section called “ALTER TABLE” 섹션을 참조하세요.

튜플 유형

tuple 데이터 유형은 한정된 리터럴 요소 그룹을 나타냅니다. user defined type 대신 튜플을 사용할 수 있습니다. 튜플에는 FROZEN 키워드를 사용할 필요가 없습니다. 튜플은 항상 프로즌 상태이고 요소를 개별적으로 업데이트할 수 없기 때문입니다.

기타 유형

데이터 유형	설명
inet	IP 주소를 IPv4 또는 IPv6 형식으로 나타내는 문자열입니다.

정적

클러스터링 열이 있는 Amazon Keyspaces 테이블에서 STATIC 키워드를 사용하여 모든 유형의 정적 열을 생성할 수 있습니다.

다음 문은 이에 대한 예입니다.

```
my_column INT STATIC
```

정적 열 사용에 대한 자세한 내용은 을 참조하십시오. the section called “정적 열”

Amazon Keyspaces 데이터 유형의 JSON 인코딩

Amazon Keyspaces는 Apache Cassandra와 동일한 JSON 데이터 유형 매핑을 제공합니다. 다음 표는 Amazon Keyspaces가 INSERT JSON 문에서 허용하는 데이터 유형과 Amazon Keyspaces가 SELECT JSON 문과 함께 데이터를 반환할 때 사용하는 데이터 유형을 설명합니다.

float, int, UUID, date 등의 단일 필드 데이터 유형의 경우 데이터를 string로 삽입할 수도 있습니다. tuple, map, list 등의 복합 데이터 유형 및 수집의 경우 데이터를 JSON이나 인코딩된 JSON string 형식으로 삽입할 수도 있습니다.

JSON 데이터 유형	**INSERT JSON** 문에서 허용되는 데이터 유형	**SELECT JSON** 문에서 반환되는 데이터 유형	참고
ascii	string	string	JSON 문자 이스케이프 \u을(를) 사용합니다.
bigint	integer, string	integer	문자열은 유효한 64비트 정수여야 합니다.
blob	string	string	문자열은 0x 다음에 16진수의 짝수로 시작해야 합니다.
boolean	boolean, string	boolean	문자열은 true 또는 false여야 합니다.
date	string	string	날짜 형식 YYYY-MM-DD , 시간대 UTC.

JSON 데이터 유형	**INSERT JSON** 문에서 허용되는 데이터 유형	**SELECT JSON** 문에서 반환되는 데이터 유형	참고
decimal	integer, float, string	float	클라이언트 측 디코더에서 32비트 또는 64비트 IEEE-754 부동 소수점 정밀도를 초과할 수 있습니다.
double	integer, float, string	float	문자열은 유효한 정수 또는 부동 소수점이어야 합니다.
float	integer, float, string	float	문자열은 유효한 정수 또는 부동 소수점이어야 합니다.
inet	string	string	IPv4 주소 또는 IPv6 주소.
int	integer, string	integer	문자열은 유효한 32비트 정수여야 합니다.
list	list, string	list	네이티브 JSON 목록 표현을 사용합니다.
map	map, string	map	네이티브 JSON 맵 표현을 사용합니다.
smallint	integer, string	integer	문자열은 유효한 16비트 정수여야 합니다.
set	list, string	list	네이티브 JSON 목록 표현을 사용합니다.
text	string	string	JSON 문자 이스케이프 \u을(를) 사용합니다.

JSON 데이터 유형	**INSERT JSON** 문에서 허용되는 데이터 유형	**SELECT JSON** 문에서 반환되는 데이터 유형	참고
time	string	string	시간대 형식HH-MM-SS[.fffffffff] .
timestamp	integer, string	string	타임스탬프입니다. 문자열 상수를 사용하면 타임스탬프를 날짜로 저장할 수 있습니다. YYYY-MM-DD HH:MM:SS.SSS 형식의 날짜 스탬프가 반환됩니다.
timeuuid	string	string	1 UUID를 입력합니다. UUID 형식에 대한 내용은 constants을 참조하세요.
tinyint	integer, string	integer	문자열은 유효한 8비트 정수여야 합니다.
tuple	list, string	list	네이티브 JSON 목록 표현을 사용합니다.
uuid	string	string	UUID 형식에 대한 내용은 constants을 참조하세요.
varchar	string	string	JSON 문자 이스케이프 \u을(를) 사용합니다.
varint	integer, string	integer	가변 길이. 클라이언트 측 디코더에서 32비트 또는 64비트 정수가 오버플로될 수 있습니다.

Amazon Keyspaces의 DDL 문(데이터 정의 언어)

데이터 정의 언어(DDL)는 Amazon Keyspaces(Apache Cassandra용)에서 키스페이스 및 테이블과 같은 데이터 구조를 관리하는 데 사용하는 Cassandra 쿼리 언어(CQL) 문 집합입니다. DDL을 사용하여 이러한 데이터 구조를 생성하고, 생성 후 수정하고, 더 이상 사용하지 않을 때는 제거할 수 있습니다. Amazon Keyspaces는 DDL 작업을 비동기적으로 수행합니다. 비동기 작업이 완료되었는지 확인하는 방법에 대한 자세한 내용은 the section called “키스페이스 및 테이블의 비동기 생성 및 삭제” 섹션을 참조하세요.

다음 DDL 문이 지원됩니다.

- CREATE KEYSPACE
- ALTER KEYSPACE
- DROP KEYSPACE
- CREATE TABLE
- ALTER TABLE
- RESTORE TABLE
- DROP TABLE

주제

- Keyspaces
- 표

Keyspaces

키스페이스는 하나 이상의 애플리케이션과 관련된 관련 테이블을 그룹화합니다. 관계형 데이터베이스 관리 시스템(RDBMS)의 경우, 키스페이스는 데이터베이스, 테이블스페이스, 또는 유사한 구조와 거의 유사합니다.

> **Note**
>
> Apache Cassandra에서 키스페이스는 여러 스토리지 노드 간에 데이터가 복제되는 방식을 결정합니다. 하지만 Amazon Keyspaces는 완전 관리형 서비스이므로 스토리지 계층의 세부 정보를 자동으로 관리합니다. 따라서 Amazon Keyspaces의 키스페이스는 논리적 구조일 뿐이며 기본적인 물리적 스토리지와는 관련이 없습니다.

Amazon Keyspaces 키스페이스의 할당량 한도 및 제약 조건에 대한 자세한 내용은 할당량을 참조하세요.

키스페이스에 대한 설명

- CREATE KEYSPACE
- ALTER KEYSPACE
- DROP KEYSPACE

CREATE KEYSPACE

CREATE KEYSPACE 문을 사용하여 새 키스페이스를 생성합니다.

구문

```
create_keyspace_statement ::=
    CREATE KEYSPACE [ IF NOT EXISTS ] keyspace_name
    WITH options
```

위치:

- *keyspace_name*은 생성할 키스페이스 이름입니다.
- 옵션은 다음 중 하나 이상입니다.
  - REPLICATION — 키스페이스의 복제 전략을 나타내는 맵:
    - SingleRegionStrategy — 단일 리전 키스페이스용. (필수)
    - NetworkTopologyStrategy— 최소 2개에서 최대 AWS 리전 6개까지 지정합니다. 각 리전의 복제 계수는 3입니다. (선택 사항)
  - DURABLE_WRITES — Amazon Keyspace에 대한 쓰기는 항상 내구성이 뛰어나므로 이 옵션은 필수가 아닙니다. 하지만 지정하는 경우 값은 true이어야 합니다.
  - TAGS – 리소스를 생성할 때 연결할 키-값 페어 태그의 목록입니다. (선택 사항)

예

다음과 같이 키스페이스를 생성합니다.

```
CREATE KEYSPACE my_keyspace
    WITH REPLICATION = {'class': 'SingleRegionStrategy'} and TAGS ={'key1':'val1',
 'key2':'val2'} ;
```

다중 리전 키스페이스를 만들려면 NetworkTopologyStrategy을 지정하고 AWS 리전를 최소 2개에서 최대 6개까지 포함합니다. 각 리전의 복제 계수는 3입니다.

```
CREATE KEYSPACE my_keyspace
    WITH REPLICATION = {'class':'NetworkTopologyStrategy', 'us-east-1':'3', 'ap-southeast-1':'3','eu-west-1':'3'};
```

ALTER KEYSPACE

ALTER KEYSPACE를 사용하여 키스페이스에 태그를 추가하거나 키스페이스에서 태그를 제거합니다.

구문

```
alter_keyspace_statement ::=
    ALTER KEYSPACE keyspace_name
    [[ADD | DROP] TAGS
```

위치:

- *keyspace_name*는 변경할 키스페이스의 이름입니다.
- TAGS — 키스페이스에서 추가하거나 제거할 키-값 페어 태그의 목록입니다.

예

다음과 같이 키스페이스를 변경합니다.

```
ALTER KEYSPACE "myGSGKeyspace" ADD TAGS {'key1':'val1', 'key2':'val2'};
```

DROP KEYSPACE

DROP KEYSPACE 문을 사용하여 테이블과 같은 모든 콘텐츠를 포함한 키스페이스를 제거합니다.

구문

```
drop_keyspace_statement ::=
    DROP KEYSPACE [ IF EXISTS ] keyspace_name
```

위치:

- keyspace_name은 삭제할 키스페이스의 이름입니다.

예

```
DROP KEYSPACE "myGSGKeyspace";
```

표

테이블은 Amazon Keyspaces의 기본 데이터 구조입니다. 테이블의 데이터는 행과 열로 구성됩니다. 이러한 열의 하위 집합은 파티션 키 지정을 통해 파티셔닝(및 데이터 배치)을 결정하는 데 사용됩니다.

또 다른 열 집합을 클러스터링 열로 정의할 수 있습니다. 즉, 이러한 열 집합을 쿼리 실행 시 조건자로 사용할 수 있습니다.

기본적으로 온디맨드 처리량 용량을 갖춘 새 테이블이 생성됩니다. 새 테이블과 기존 테이블의 용량 모드를 변경할 수 있습니다. 읽기/쓰기 용량 처리량 모드에 대한 자세한 내용은 the section called “읽기/쓰기 용량 모드”를 참조하세요.

프로비저닝 모드의 테이블의 경우 선택 사항을 구성할 수 있습니다. AUTOSCALING_SETTINGS Amazon Keyspaces 자동 크기 조정 및 사용 가능한 옵션에 대한 자세한 내용은 을 참조하십시오. the section called “CQL 사용”

Amazon Keyspaces 테이블의 할당량 한도 및 제약 조건에 대한 자세한 내용은 할당량을 참조하세요.

테이블에 대한 설명

- CREATE TABLE
- ALTER TABLE
- RESTORE TABLE
- DROP TABLE

CREATE TABLE

CREATE TABLE 문을 사용하여 새 테이블을 만듭니다.

구문

```
create_table_statement ::=  CREATE TABLE [ IF NOT EXISTS ] table_name
    '('
        column_definition
```

```
        ( ',' column_definition )*
        [ ',' PRIMARY KEY '(' primary_key ')' ]
    ')' [ WITH table_options ]

column_definition      ::=  column_name cql_type [ FROZEN ][ STATIC ][ PRIMARY KEY]

primary_key            ::=  partition_key [ ',' clustering_columns ]

partition_key          ::=  column_name
                              | '(' column_name ( ',' column_name )* ')'

clustering_columns     ::=  column_name ( ',' column_name )*

table_options          ::=  [table_options]
                              | CLUSTERING ORDER BY '(' clustering_order
 ')' [ AND table_options ]
                              | options
                              | CUSTOM_PROPERTIES
                              | AUTOSCALING_SETTINGS
                              | default_time_to_live
                              | TAGS

clustering_order       ::=  column_name (ASC | DESC) ( ',' column_name (ASC | DESC) )*
```

위치:

- *table_name*은 생성할 테이블의 이름입니다.
- *column_definition*은 다음 항목으로 구성됩니다.
  - *column_name* – 열의 이름입니다.
  - *cql_type* — An Amazon Keyspaces 데이터 유형(데이터 타입 참조)
  - *FROZEN* — collection 유형의 열(예: LIST, SET, MAP)을 프로즌으로 지정합니다. 프로즌 수집은 변경할 수 없는 단일 값으로 직렬화되고 BLOB처럼 취급됩니다. 자세한 정보는 the section called “컬렉션 유형”을 참조하세요.
  - *STATIC* — 이 열을 정적 열로 지정합니다. 정적 열에는 동일한 파티션의 모든 행이 공유하는 값이 저장됩니다.
  - *PRIMARY KEY* — 이 열을 테이블의 프라이머리 키로 지정합니다.
- *primary_key*은 다음 항목으로 구성됩니다.
  - partition_key
  - clustering_columns

- *partition_key*:
  - 파티션 키는 단일 열이거나 둘 이상의 열로 구성된 복합 값일 수 있습니다. 프라이머리 키의 파티션 키 부분은 필수 입력 항목이며 이에 따라 Amazon Keyspaces가 데이터를 저장하는 방식이 결정됩니다.
- *clustering_columns*:
  - 프라이머리 키의 선택적 클러스터링 열 부분에 따라 각 파티션 내에서 데이터가 클러스터링되고 정렬되는 방식이 결정됩니다.
- *table_options*은 다음 항목으로 구성됩니다.
  - *CLUSTERING ORDER BY* — 테이블의 기본 클러스터링 순서는 ASC(오름차순) 정렬 방향의 클러스터링 키로 구성됩니다. 기본 정렬 동작을 재정의하도록 지정합니다.
  - *CUSTOM_PROPERTIES* — Amazon Keyspace에만 적용되는 설정 맵.
    - capacity_mode: 테이블에 읽기/쓰기 처리량 용량 모드를 지정합니다. 옵션은 throughput_mode:PAY_PER_REQUEST 및 throughput_mode:PROVISIONED입니다. 프로비저닝된 용량 모드에는 read_capacity_units 및 write_capacity_units가 입력으로 필요합니다. 기본값은 throughput_mode:PAY_PER_REQUEST입니다.
    - client_side_timestamps: 테이블에 대해 클라이언트 측 타임스탬프를 활성화할지 여부를 지정합니다. 옵션은 {'status': 'enabled'} 및 {'status': 'disabled'}입니다. 지정하지 않으면 기본값으로 status:disabled이 지정됩니다. 테이블에 대해 클라이언트 측 타임스탬프를 활성화한 후에는 이 설정을 비활성화할 수 없습니다.
    - encryption_specification: 저장된 암호화에 대한 암호화 옵션을 지정합니다. 지정하지 않으면 기본값으로 encryption_type:AWS_OWNED_KMS_KEY이 지정됩니다. 고객 관리 키 암호화 옵션을 사용하려면 Amazon 리소스 이름 (ARN) 형식의 AWS KMS 키를 입력해야 합니다. kms_key_identifier:ARN kms_key_identifier:ARN
    - point_in_time_recovery: 테이블에 대한 point-in-time 복원 활성화 또는 비활성화 여부를 지정합니다. 옵션은 status:enabled 및 status:disabled입니다. 지정하지 않으면 기본값으로 status:disabled이 지정됩니다.
    - replica_updates: 특정 다중 지역 테이블의 설정을 지정합니다. AWS 리전다중 지역 테이블의 경우 테이블의 읽기 용량을 각각 다르게 구성할 수 있습니다. AWS 리전다음 파라미터를 구성하여 이 작업을 수행할 수 있습니다. 자세한 정보와 지침은 the section called “프로비저닝된 용량 모드 및 CQL (Auto Scaling) 을 사용하여 멀티 리전 테이블 생성” 섹션을 참조하세요.
      - region— 다음과 같은 설정을 가진 테이블 복제본의: AWS 리전
        - read_capacity_units

- TTL: 테이블에 대한 Time to Live 사용자 정의 설정을 활성화합니다. 활성화하려면 `status:enabled`을 사용합니다. 기본값은 `status:disabled`입니다. TTL이 활성화된 후에는 테이블에 대해 이 설정을 비활성화할 수 없습니다.

- *AUTOSCALING_SETTINGS*프로비저닝 모드의 테이블에 대한 다음과 같은 선택적 설정을 포함합니다. 자세한 정보와 지침은 the section called “CQL을 사용하여 자동 크기 조정이 가능한 새 테이블을 생성합니다.” 섹션을 참조하세요.
  - `provisioned_write_capacity_autoscaling_update`:
    - `autoscaling_disabled`— 쓰기 용량에 대한 Auto Scaling을 활성화하려면 값을 로 설정합니다`false`. 기본값은 `true`입니다. (선택 사항)
    - `minimum_units`— 테이블이 항상 지원할 준비가 되어 있어야 하는 최소 쓰기 처리량 수준입니다. 값은 1에서 계정의 초당 최대 처리량 할당량 (기본값 40,000) 사이여야 합니다.
    - `maximum_units`— 테이블이 항상 지원할 준비가 되어 있어야 하는 쓰기 처리량의 최대 수준입니다. 값은 1에서 계정의 초당 최대 처리량 할당량 (기본값 40,000) 사이여야 합니다.
    - `scaling_policy`— Amazon Keyspace는 대상 추적 정책을 지원합니다. Auto Scaling 타겟은 테이블의 프로비저닝된 쓰기 용량입니다.
      - `target_tracking_scaling_policy_configuration`— 대상 추적 정책을 정의하려면 대상 값을 정의해야 합니다. 대상 추적 및 휴지 기간에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 대상 추적 조정 정책을 참조하십시오.
        - `target_value`— 표의 목표 사용률. Amazon Keyspaces Auto Scaling은 프로비저닝된 용량에 대한 소비 용량의 비율이 이 값 이하로 유지되도록 합니다. `target_value`를 백분율로 지정합니다. 20에서 90 사이의 두 배입니다. (필수)
        - `scale_in_cooldown`— 스케일링 활동 사이의 휴지 기간 (초) 으로, 다른 규모의 활동이 시작되기 전에 테이블을 안정화할 수 있습니다. 값을 제공하지 않는 경우 기본값은 0입니다. (선택 사항)
        - `scale_out_cooldown`— 스케일링 활동 사이의 휴지 기간 (초) 으로, 다른 스케일 아웃 활동이 시작되기 전에 테이블을 안정화할 수 있습니다. 값을 제공하지 않는 경우 기본값은 0입니다. (선택 사항)
        - `disable_scale_in`: A: 테이블에 대한 비활성화 또는 활성화 여부를 `boolean` `scale-in` 지정하는 A. 이 매개변수는 기본적으로 비활성화되어 있습니다. `scale-in`켜려면 `boolean` 값을 로 설정합니다`FALSE`. 즉, 사용자 대신 테이블의 용량이 자동으로 축소됩니다. (선택 사항)
  - `provisioned_read_capacity_autoscaling_update`:
    - `autoscaling_disabled`— 읽기 용량에 대해 Auto Scaling을 활성화하려면 값을 로 설정합니다`false`. 기본값은 `true`입니다. (선택 사항)

- `minimum_units`— 테이블이 항상 지원할 준비가 되어 있어야 하는 최소 처리량 수준입니다. 값은 1에서 계정의 초당 최대 처리량 할당량 (기본값 40,000) 사이여야 합니다.
- `maximum_units`— 테이블이 항상 지원할 준비가 되어 있어야 하는 최대 처리량 수준입니다. 값은 1에서 계정의 초당 최대 처리량 할당량 (기본값 40,000) 사이여야 합니다.
- `scaling_policy`— Amazon Keyspace는 대상 추적 정책을 지원합니다. Auto Scaling 목표는 테이블의 프로비저닝된 읽기 용량입니다.
  - `target_tracking_scaling_policy_configuration`— 대상 추적 정책을 정의하려면 목표 값을 정의해야 합니다. 대상 추적 및 휴지 기간에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 대상 추적 조정 정책을 참조하십시오.
    - `target_value`— 표의 목표 사용률. Amazon Keyspaces Auto Scaling은 프로비저닝된 용량에 대한 소비 용량의 비율이 이 값 이하로 유지되도록 합니다. `target_value`를 백분율로 지정합니다. 20에서 90 사이의 두 배입니다. (필수)
    - `scale_in_cooldown`— 스케일링 활동 사이의 휴지 기간 (초) 으로, 다른 규모의 활동이 시작되기 전에 테이블을 안정화할 수 있습니다. 값을 제공하지 않는 경우 기본값은 0입니다. (선택 사항)
    - `scale_out_cooldown`— 스케일링 활동 사이의 휴지 기간 (초) 으로, 다른 스케일 아웃 활동이 시작되기 전에 테이블을 안정화할 수 있습니다. 값을 제공하지 않는 경우 기본값은 0입니다. (선택 사항)
    - `disable_scale_in`: A: 테이블에 대한 비활성화 또는 활성화 여부를 `boolean` `scale-in` 지정하는 A. 이 매개변수는 기본적으로 비활성화되어 있습니다. `scale-in`켜려면 `boolean` 값을 로 설정합니다`FALSE`. 즉, 사용자 대신 테이블의 용량이 자동으로 축소됩니다. (선택 사항)
- `replica_updates`: 다중 지역 테이블의 AWS 리전 특정 Auto Scaling 설정을 지정합니다. 다중 지역 테이블의 경우 테이블의 읽기 용량을 각각 다르게 구성할 수 있습니다. AWS 리전다음 파라미터를 구성하여 이 작업을 수행할 수 있습니다. 자세한 정보와 지침은 the section called “프로비저닝된 용량 모드 및 CQL (Auto Scaling) 을 사용하여 멀티 리전 테이블 생성” 섹션을 참조하세요.
  - `region`— 다음과 같은 설정을 가진 테이블 복제본의: AWS 리전
    - `provisioned_read_capacity_autoscaling_update`
      - `autoscaling_disabled`— 테이블의 읽기 용량에 대해 Auto Scaling을 활성화하려면 값을 로 설정합니다`false`. 기본값은 `true`입니다. (선택 사항)

> **Note**
>
> 다중 지역 테이블의 자동 크기 조정은 테이블의 모든 복제본에 대해 활성화하거나 비활성화해야 합니다.

- `minimum_units`— 테이블이 항상 지원할 준비가 되어 있어야 하는 최소 읽기 처리량 수준. 값은 1에서 계정의 초당 최대 처리량 할당량 (기본값 40,000) 사이여야 합니다.
- `maximum_units`— 테이블이 항상 지원할 준비가 되어 있어야 하는 읽기 처리량의 최대 수준입니다. 값은 1에서 계정의 초당 최대 처리량 할당량 (기본값 40,000) 사이여야 합니다.
- `scaling_policy`— Amazon Keyspace는 대상 추적 정책을 지원합니다. Auto Scaling 목표는 테이블의 프로비저닝된 읽기 용량입니다.
  - `target_tracking_scaling_policy_configuration`— 대상 추적 정책을 정의하려면 목표 값을 정의해야 합니다. 대상 추적 및 휴지 기간에 대한 자세한 내용은 Application Auto Scaling 사용 설명서의 대상 추적 조정 정책을 참조하십시오.
    - `target_value`— 표의 목표 사용률. Amazon Keyspaces Auto Scaling은 사용된 읽기 용량과 프로비저닝된 읽기 용량의 비율이 이 값 이하로 유지되도록 합니다. `target_value`를 백분율로 지정합니다. 20에서 90 사이의 두 배입니다. (필수)
    - `scale_in_cooldown`— 스케일링 활동 사이의 휴지 기간 (초) 으로, 다른 규모의 활동이 시작되기 전에 테이블을 안정화할 수 있습니다. 값을 제공하지 않는 경우 기본값은 0입니다. (선택 사항)
    - `scale_out_cooldown`— 스케일링 활동 사이의 휴지 기간 (초) 으로, 다른 스케일 아웃 활동이 시작되기 전에 테이블을 안정화할 수 있습니다. 값을 제공하지 않는 경우 기본값은 0입니다. (선택 사항)
    - `disable_scale_in`: A: 테이블에 대한 비활성화 또는 활성화 여부를 `boolean` `scale-in` 지정하는 A. 이 매개변수는 기본적으로 비활성화되어 있습니다. `scale-in`켜려면 `boolean` 값을 로 설정합니다`FALSE`. 즉, 사용자 대신 테이블의 읽기 용량이 자동으로 축소됩니다. (선택 사항)
- `default_time_to_live` — 테이블의 기본 Time to Live 설정(초 단위)입니다.
- `TAGS` – 리소스를 생성할 때 연결할 키-값 페어 태그의 목록입니다.

- *`clustering_order`*은 다음 항목으로 구성됩니다.
  - *`column_name`* – 열의 이름입니다.

- *ASC | DESC* — 오름차순(ASC) 또는 내림차순(DESC) 순서 수정자를 설정합니다. 지정하지 않으면 기본 순서로 ASC가 지정됩니다.

예

```
CREATE TABLE IF NOT EXISTS "my_keyspace".my_table (
                                            id text,
                                            name text,
                                            region text,
                                            division text,
                                            project text,
                                            role text,
                                            pay_scale int,
                                            vacation_hrs float,
                                            manager_id text,
                                            PRIMARY KEY (id,division))
                                            WITH CUSTOM_PROPERTIES={
                                                'capacity_mode':{
                                                        'throughput_mode':
 'PROVISIONED', 'read_capacity_units': 10, 'write_capacity_units': 20
                                                    },
                                                'point_in_time_recovery':{'status':
 'enabled'},
                                                'encryption_specification':{
                                                        'encryption_type':
 'CUSTOMER_MANAGED_KMS_KEY',

 'kms_key_identifier':'arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111'
                                                    }
                                            }
                                            AND CLUSTERING ORDER BY (division ASC)
                                            AND TAGS={'key1':'val1', 'key2':'val2'}
                                            AND default_time_to_live = 3024000;
```

클러스터링 열을 사용하는 테이블의 경우 테이블 정의에서 클러스터링이 아닌 열을 고정으로 선언할 수 있습니다. 정적 열에 대한 자세한 내용은 the section called “정적 열”을 참조하세요.

예

```
CREATE TABLE "my_keyspace".my_table (
                                            id int,
```

```
                                        name text,
                                        region text,
                                        division text,
                                        project text STATIC,
                                        PRIMARY KEY (id,division));
```

ALTER TABLE

ALTER TABLE 문을 사용하여 새 열을 추가하거나, 태그를 추가하거나, 테이블의 사용자 지정 속성을 변경합니다.

구문

```
alter_table_statement ::=  ALTER TABLE table_name

        [ ADD ( column_definition | column_definition_list)  ]
        [[ADD | DROP] TAGS {'key1':'val1', 'key2':'val2'}]
        [ WITH table_options [ , ... ] ] ;

column_definition      ::=  column_name cql_type
```

위치:

- *table_name*는 변경할 테이블의 이름입니다.
- *column_definition*는 추가할 열 이름 및 데이터 유형입니다.
- *column_definition_list*는 괄호 안에 있는 쉼표로 구분된 열 목록입니다.
- *table_options*은 다음 항목으로 구성됩니다.
  - *CUSTOM_PROPERTIES*— Amazon Keyspace와 관련된 설정 맵입니다.
    - capacity_mode: 테이블에 읽기/쓰기 처리량 용량 모드를 지정합니다. 옵션은 throughput_mode:PAY_PER_REQUEST 및 throughput_mode:PROVISIONED입니다. 프로비저닝된 용량 모드에는 read_capacity_units 및 write_capacity_units가 입력으로 필요합니다. 기본값은 throughput_mode:PAY_PER_REQUEST입니다.
    - client_side_timestamps: 테이블에 대해 클라이언트 측 타임스탬프를 활성화할지 여부를 지정합니다. 옵션은 {'status': 'enabled'} 및 {'status': 'disabled'}입니다. 지정하지 않으면 기본값으로 status:disabled이 지정됩니다. 테이블에 대해 클라이언트 측 타임스탬프를 활성화한 후에는 이 설정을 비활성화할 수 없습니다.

- `encryption_specification`: 저장된 암호화에 대한 암호화 옵션을 지정합니다. 옵션은 `encryption_type:AWS_OWNED_KMS_KEY` 및 `encryption_type:CUSTOMER_MANAGED_KMS_KEY`입니다. 암호화 옵션 고객 관리형 키를 사용하려면 Amazon 리소스 이름(ARN) 형식의 AWS KMS 키(`kms_key_identifier:ARN`)를 입력해야 합니다.
- `point_in_time_recovery`: 테이블에 대한 point-in-time 복원 활성화 또는 비활성화 여부를 지정합니다. 옵션은 `status:enabled` 및 `status:disabled`입니다. 기본값은 `status:disabled`입니다.
- `replica_updates`: 다중 지역 테이블의 AWS 리전 특정 설정을 지정합니다. 다중 지역 테이블의 경우 테이블의 읽기 용량을 각각 다르게 구성할 수 있습니다. AWS 리전다음 파라미터를 구성하여 이 작업을 수행할 수 있습니다. 자세한 정보와 지침은 the section called “멀티 리전 테이블 (CQL) 의 프로비저닝된 용량 및 Auto Scaling 설정 업데이트” 섹션을 참조하세요.
  - `region`— 다음과 같은 설정을 가진 테이블 복제본의: AWS 리전
    - `read_capacity_units`
- `ttl`: 테이블에 대한 Time to Live 사용자 정의 설정을 활성화합니다. 활성화하려면 `status:enabled`을 사용합니다. 기본값은 `status:disabled`입니다. `ttl`이 활성화된 후에는 테이블에 대해 이 설정을 비활성화할 수 없습니다.

- *`AUTOSCALING_SETTINGS`*프로비저닝된 테이블에 대한 선택적 Auto Scaling 설정을 포함합니다. 구문 및 자세한 설명은 을 참조하십시오. the section called “CREATE TABLE” 예를 보려면 the section called “CQL을 사용하여 기존 테이블의 자동 크기 조정을 활성화합니다.”을 참조하세요.

- `default_time_to_live`: 테이블의 기본 Time to Live 설정(초 단위)입니다.
- *`TAGS`*는 리소스에 연결할 키-값 페어 태그의 목록입니다.

> **Note**
>
> ALTER TABLE을 사용하면 단일 사용자 지정 속성만 변경할 수 있습니다. 동일한 문에 ALTER TABLE 명령을 두 개 이상 조합할 수 없습니다.

예제

다음 문은 기존 테이블에 열을 추가하는 방법을 보여줍니다.

```
ALTER TABLE mykeyspace.mytable ADD (ID int);
```

이 문은 기존 테이블에 두 개의 수집 열을 추가하는 방법을 보여줍니다.

- 중첩된 프로즌 수집이 포함된 프로즌 수집 열 `col_frozen_list`
- 중첩된 프로즌 수집이 포함된 비프로즌 수집 열 `col_map`

```
ALTER TABLE my_Table ADD(col_frozen_list FROZEN<LIST<FROZEN<SET<TEXT>>>>, col_map
 MAP<INT, FROZEN<SET<INT>>>);
```

테이블의 용량 모드를 변경하고 읽기 및 쓰기 용량 단위를 지정하려면 다음 문을 사용할 수 있습니다.

```
ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={'capacity_mode':
{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 10, 'write_capacity_units':
 20}};
```

다음 문은 테이블의 고객 관리형 KMS 키를 지정합니다.

```
ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={
              'encryption_specification':{
                     'encryption_type': 'CUSTOMER_MANAGED_KMS_KEY',
                     'kms_key_identifier':'arn:aws:kms:eu-
west-1:5555555555555:key/11111111-1111-111-1111-111111111111'
                   }
         };
```

테이블 point-in-time 복원을 활성화하려면 다음 명령문을 사용할 수 있습니다.

```
ALTER TABLE mykeyspace.mytable WITH CUSTOM_PROPERTIES={'point_in_time_recovery':
 {'status': 'enabled'}};
```

테이블의 기본 Time to Live 값을 초 단위로 설정하려면 다음 문을 사용할 수 있습니다.

```
ALTER TABLE my_table WITH default_time_to_live = 2592000;
```

이 문을 사용하면 테이블의 Time to Live 설정을 사용자 지정할 수 있습니다.

```
ALTER TABLE mytable WITH CUSTOM_PROPERTIES={'ttl':{'status': 'enabled'}};
```

RESTORE TABLE

RESTORE TABLE 문을 사용하여 테이블을 특정 시점으로 복원합니다. 이 명령문을 사용하려면 테이블에서 point-in-time 복구를 활성화해야 합니다. 자세한 정보는 시점 복구을 참조하세요.

구문

```
restore_table_statement ::=
    RESTORE TABLE restored_table_name FROM TABLE source_table_name
                    [ WITH table_options [ , ... ] ];
```

위치:

- *restored_table_name*은 복원된 테이블의 이름입니다.
- *source_table_name*은 소스 테이블의 이름입니다.
- *table_options*은 다음 항목으로 구성됩니다.
  - *restore_timestamp*는 ISO 8601 형식의 복원 시점입니다. 지정하지 않을 경우 현재 타임스탬프가 사용됩니다.
  - *CUSTOM_PROPERTIES*— Amazon Keyspace와 관련된 설정 맵입니다.
    - capacity_mode: 테이블에 읽기/쓰기 처리량 용량 모드를 지정합니다. 옵션은 throughput_mode:PAY_PER_REQUEST 및 throughput_mode:PROVISIONED입니다. 프로비저닝된 용량 모드에는 read_capacity_units 및 write_capacity_units가 입력으로 필요합니다. 기본값은 소스 테이블의 현재 설정입니다.
    - encryption_specification: 저장된 암호화에 대한 암호화 옵션을 지정합니다. 옵션은 encryption_type:AWS_OWNED_KMS_KEY 및 encryption_type:CUSTOMER_MANAGED_KMS_KEY입니다. 고객 관리 키 암호화 옵션을 사용하려면 Amazon 리소스 이름 (ARN) 형식의 AWS KMS 키를 입력해야 합니다. kms_key_identifier:ARN 고객 관리 키로 암호화된 테이블을 로 암호화된 테이블로 복원하려면 Amazon Keyspaces에서 원본 테이블의 키에 액세스할 수 있어야 합니다. AWS 소유 키 AWS KMS
    - point_in_time_recovery: 테이블에 대한 point-in-time 복원 활성화 또는 비활성화 여부를 지정합니다. 옵션은 status:enabled 및 status:disabled입니다. 새 테이블을 생성할 때와 달리 복원된 테이블의 기본 상태는 설정이 소스 테이블에서 상속되기 때문에 status:enabled입니다. 복원된 테이블의 PITR을 비활성화하려면 status:disabled을 명시적으로 설정해야 합니다.

  - `replica_updates`: 다중 지역 테이블의 AWS 리전 특정 설정을 지정합니다. 다중 지역 테이블의 경우 테이블의 읽기 용량을 각각 다르게 구성할 수 있습니다. AWS 리전다음 파라미터를 구성하여 이 작업을 수행할 수 있습니다.
    - `region`— 다음과 같은 설정을 가진 테이블 복제본의: AWS 리전
      - `read_capacity_units`
- *`AUTOSCALING_SETTINGS`*프로비저닝된 테이블에 대한 선택적 Auto Scaling 설정을 포함합니다. 자세한 구문 및 설명은 을 참조하십시오. the section called “CREATE TABLE”
- *`TAGS`*는 리소스에 연결할 키-값 페어 태그의 목록입니다.

> **Note**
>
> 삭제된 테이블은 삭제 시점까지만 복원할 수 있습니다.

예

```
RESTORE TABLE mykeyspace.mytable_restored from table mykeyspace.my_table
WITH restore_timestamp = '2020-06-30T04:05:00+0000'
AND custom_properties = {'point_in_time_recovery':{'status':'disabled'},
 'capacity_mode':{'throughput_mode': 'PROVISIONED', 'read_capacity_units': 10,
 'write_capacity_units': 20}}
AND TAGS={'key1':'val1', 'key2':'val2'};
```

DROP TABLE

`DROP TABLE` 문을 사용하여 키스페이스에서 테이블을 제거합니다.

구문

```
drop_table_statement ::=
    DROP TABLE [ IF EXISTS ] table_name
```

위치:

- `IF EXISTS`은 테이블이 존재하지 않는 경우 `DROP TABLE`의 실패를 방지합니다. (선택 사항)
- *`table_name`*는 삭제할 테이블의 이름입니다.

예

```
DROP TABLE "myGSGKeyspace".employees_tbl;
```

Amazon Keyspaces의 DML 문(데이터 조작 언어)

데이터 조작 언어(DML)는 Amazon Keyspaces(Apache Cassandra용) 테이블의 데이터를 관리하는 데 사용하는 Cassandra 쿼리 언어(CQL) 문 집합입니다. DML 문을 사용하여 테이블의 데이터를 추가, 수정 또는 삭제할 수 있습니다.

또한 DML 문을 사용하여 테이블의 데이터를 쿼리합니다. (참고로 CQL은 조인 또는 하위 쿼리를 지원하지 않습니다.)

주제

- SELECT
- INSERT
- UPDATE
- DELETE

SELECT

SELECT 문을 사용하여 데이터를 쿼리합니다.

구문

```
select_statement ::=  SELECT  [ JSON ] ( select_clause | '*' )
                      FROM table_name
                      [ WHERE 'where_clause' ]
                      [ ORDER BY 'ordering_clause' ]
                      [ LIMIT (integer | bind_marker) ]
                      [ ALLOW FILTERING ]
select_clause    ::=  selector [ AS identifier ] ( ',' selector [ AS identifier ] )
selector         ::=  column_name
                      | term
                      | CAST '(' selector AS cql_type ')'
                      | function_name '(' [ selector ( ',' selector )* ] ')'
where_clause     ::=  relation ( AND relation )*
relation         ::=  column_name operator term
                      TOKEN
```

```
operator          ::=  '=' | '<' | '>' | '<=' | '>=' | IN | CONTAINS | CONTAINS KEY
ordering_clause   ::=  column_name [ ASC | DESC ] ( ',' column_name [ ASC | DESC ] )*
```

예제

```
SELECT name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;

SELECT JSON name, id, manager_id FROM "myGSGKeyspace".employees_tbl ;
```

JSON 인코딩 데이터 유형을 Amazon Keyspaces 데이터 유형에 매핑하는 테이블은 the section called "Amazon Keyspaces 데이터 유형의 JSON 인코딩"을 참조하세요.

IN 키워드 사용

IN 키워드는 하나 이상의 값이 같도록 지정합니다. 파티션 키와 클러스터링 열에 적용할 수 있습니다. SELECT 문에 키가 표시된 순서대로 결과가 반환됩니다.

예제

```
SELECT * from mykeyspace.mytable WHERE primary.key1 IN (1,2) and clustering.key1 = 2;
SELECT * from mykeyspace.mytable WHERE primary.key1 IN (1,2) and clustering.key1 <= 2;
SELECT * from mykeyspace.mytable WHERE primary.key1 = 1 and clustering.key1 IN (1, 2);
SELECT * from mykeyspace.mytable WHERE primary.key1 <= 2 and clustering.key1 IN (1, 2)
 ALLOW FILTERING;
```

IN 키워드 및 Amazon Keyspaces가 문을 처리하는 방식에 대한 자세한 내용은 the section called "IN SELECT 문"섹션을 참조하세요.

결과 순서 지정

ORDER BY 절은 반환된 결과의 정렬 순서를 지정합니다. 각 열의 정렬 순서와 함께 열 이름 목록을 인수로 사용합니다. 정렬 절에는 클러스터링 열만 지정할 수 있습니다. 비클러스터링 열은 허용되지 않습니다. 정렬 순서 옵션은 ASC(오름차순)과 DESC(내림차순) 정렬 순서입니다. 정렬 순서를 생략하면 클러스터링 열의 기본 순서가 사용됩니다. 가능한 정렬 순서는 the section called "결과 순서 지정"을 참조하세요.

예

```
SELECT name, id, division, manager_id FROM "myGSGKeyspace".employees_tbl WHERE id =
 '012-34-5678' ORDER BY division;
```

IN 키워드와 함께 ORDER BY을 사용하면 페이지 내에서 결과가 정렬됩니다. 페이지 매김이 비활성화된 상태에서는 전체 재정렬이 지원되지 않습니다.

토큰

TOKEN 함수를 SELECT 및 WHERE 절의 PARTITION KEY 열에 적용할 수 있습니다. TOKEN 함수를 사용하면 Amazon Keyspaces는 PARTITION KEY의 값이 아닌 PARTITION_KEY의 매핑된 토큰 값을 기준으로 행을 반환합니다.

IN 키워드에서는 TOKEN 관계가 지원되지 않습니다.

예제

```
SELECT TOKEN(id) from my_table;

SELECT TOKEN(id) from my_table WHERE TOKEN(id) > 100 and TOKEN(id) < 10000;
```

TTL 함수

TTL 함수를 SELECT 문과 함께 사용하여 열에 저장된 만료 시간(초)을 검색할 수 있습니다. TTL인 값이 설정되지 않으면 없으면 함수가 null를 반환합니다.

예

```
SELECT TTL(my_column) from my_table;
```

수집과 같은 다중 셀 열에는 TTL 함수를 사용할 수 없습니다.

WRITETIME 함수

테이블에서 클라이언트 측 타임스탬프를 사용하는 경우에만 WRITETIME 함수를 SELECT 문과 함께 사용하여 열 값의 메타데이터로 저장된 타임스탬프를 검색할 수 있습니다. 자세한 정보는 클라이언트 측 타임스탬프을 참조하세요.

```
SELECT WRITETIME(my_column) from my_table;
```

수집과 같은 다중 셀 열에는 WRITETIME 함수를 사용할 수 없습니다.

> **Note**
>
> 설정된 Cassandra 드라이버 동작과의 호환성을 위해 Cassandra 드라이버 및 개발자 도구를 통해 Cassandra 쿼리 언어(CQL) API 호출을 사용하여 시스템 테이블에서 작업을 수행할 때는

태그 기반 권한 부여 정책이 적용되지 않습니다. 자세한 정보는 the section called “ 태그를 기반으로 한 Amazon Keyspaces 리소스 액세스”을 참조하세요.

INSERT

INSERT 문을 사용하여 테이블에 행을 추가합니다.

구문

```
insert_statement ::=  INSERT INTO table_name ( names_values | json_clause )
                      [ IF NOT EXISTS ]
                      [ USING update_parameter ( AND update_parameter )* ]
names_values     ::=  names VALUES tuple_literal
json_clause      ::=  JSON string [ DEFAULT ( NULL | UNSET ) ]
names            ::=  '(' column_name ( ',' column_name )* ')'
```

예

```
INSERT INTO "myGSGKeyspace".employees_tbl (id, name, project, region, division, role,
 pay_scale, vacation_hrs, manager_id)
VALUES ('012-34-5678','Russ','NightFlight','US','Engineering','IC',3,12.5,
 '234-56-7890') ;
```

파라미터 업데이트

INSERT에서 update_parameter로 지원하는 값은 다음과 같습니다.

- TTL — 시간 값(초)입니다. 구성 가능한 최댓값은 630,720,000초로 20년에 해당합니다.
- TIMESTAMP — epoch로 알려진 표준 기준 시간인 1970년 1월 1일 00:00:00(그리니치 표준시 기준) 이후 경과된 시간을 마이크로초 단위로 나타내는 bigint 값입니다. Amazon Keyspaces의 타임스탬프는 과거 2일에서 미래 5분 사이의 범위여야 합니다.

예

```
INSERT INTO my_table (userid, time, subject, body, user)
        VALUES (B79CB3BA-745E-5D9A-8903-4A02327A7E09, 96a29100-5e25-11ec-90d7-
b5d91eceda0a, 'Message', 'Hello','205.212.123.123')
```

```
        USING TTL 259200;
```

JSON 지원

JSON 인코딩 데이터 유형을 Amazon Keyspaces 데이터 유형에 매핑하는 테이블은 the section called "Amazon Keyspaces 데이터 유형의 JSON 인코딩"을 참조하세요.

JSON 키워드를 사용하여 JSON 인코딩 맵을 단일 행으로 삽입할 수 있습니다. 테이블에는 있지만 JSON insert 문에서 생략된 열의 경우 DEFAULT UNSET을(를) 사용하여 기존 값을 보존합니다. DEFAULT NULL을 사용하여 생략된 열의 각 행에 NULL 값을 쓰고 기존 값을 덮어씁니다(표준 쓰기 요금 적용). DEFAULT NULL가 기본 옵션입니다.

예

```
INSERT INTO "myGSGKeyspace".employees_tbl JSON '{"id":"012-34-5678",
                                                 "name": "Russ",
                                                 "project": "NightFlight",
                                                 "region": "US",
                                                 "division": "Engineering",
                                                 "role": "IC",
                                                 "pay_scale": 3,
                                                 "vacation_hrs": 12.5,
                                                 "manager_id": "234-56-7890"}';
```

JSON 데이터에 중복 키가 포함된 경우 Amazon Keyspaces는 키의 마지막 값을 저장합니다(Apache Cassandra와 유사). 중복 키가 id인 다음 예제에서는 234-56-7890 값이 사용됩니다.

예

```
INSERT INTO "myGSGKeyspace".employees_tbl JSON '{"id":"012-34-5678",
                                                 "name": "Russ",
                                                 "project": "NightFlight",
                                                 "region": "US",
                                                 "division": "Engineering",
                                                 "role": "IC",
                                                 "pay_scale": 3,
                                                 "vacation_hrs": 12.5,
                                                 "id": "234-56-7890"}';
```

UPDATE

UPDATE 문을 사용하여 테이블의 행을 수정합니다.

구문

```
update_statement ::=  UPDATE table_name
                      [ USING update_parameter ( AND update_parameter )* ]
                      SET assignment ( ',' assignment )*
                      WHERE where_clause
                      [ IF ( EXISTS | condition ( AND condition )*) ]
update_parameter ::=  ( integer | bind_marker )
assignment       ::=  simple_selection '=' term
                     | column_name '=' column_name ( '+' | '-' ) term
                     | column_name '=' list_literal '+' column_name
simple_selection ::=  column_name
                     | column_name '[' term ']'
                     | column_name '.' `field_name
condition        ::=  simple_selection operator term
```

예

```
UPDATE "myGSGKeyspace".employees_tbl SET pay_scale = 5 WHERE id = '567-89-0123' AND
 division = 'Marketing' ;
```

counter를 증가하려면 다음 구문을 사용합니다. 자세한 정보는 the section called “카운터”을 참조하세요.

```
UPDATE ActiveUsers SET counter = counter + 1  WHERE user = A70FE1C0-5408-4AE3-
BE34-8733E5K09F14 AND action = 'click';
```

파라미터 업데이트

UPDATE에서 update_parameter로 지원하는 값은 다음과 같습니다.

- TTL — 시간 값(초)입니다. 구성 가능한 최댓값은 630,720,000초로 20년에 해당합니다.
- TIMESTAMP — epoch로 알려진 표준 기준 시간인 1970년 1월 1일 00:00:00(그리니치 표준시 기준) 이후 경과된 시간을 마이크로초 단위로 나타내는 bigint 값입니다. Amazon Keyspaces의 타임스탬프는 과거 2일에서 미래 5분 사이의 범위여야 합니다.

예

```
UPDATE my_table (userid, time, subject, body, user)
```

```
        VALUES (B79CB3BA-745E-5D9A-8903-4A02327A7E09, 96a29100-5e25-11ec-90d7-
b5d91eceda0a, 'Message', 'Hello again','205.212.123.123')
        USING TIMESTAMP '2022-11-03 13:30:54+0400';
```

DELETE

DELETE 문을 사용하여 테이블에서 행을 제거합니다.

구문

```
delete_statement ::=  DELETE [ simple_selection ( ',' simple_selection ) ]
                      FROM table_name
                      [ USING update_parameter ( AND update_parameter )* ]
                      WHERE where_clause
                      [ IF ( EXISTS | condition ( AND condition )*) ]

simple_selection ::=  column_name
                     | column_name '[' term ']'
                     | column_name '.' `field_name

condition        ::=  simple_selection operator term
```

위치:

- *table_name*는 삭제하려는 행이 포함된 테이블입니다.

예

```
DELETE manager_id FROM "myGSGKeyspace".employees_tbl WHERE id='789-01-2345' AND
 division='Executive' ;
```

DELETE은 다음 값을 update_parameter로 지원합니다.

- TIMESTAMP — epoch로 알려진 표준 기준 시간인 1970년 1월 1일 00:00:00(그리니치 표준시 기준) 이후 경과된 시간을 마이크로초 단위로 나타내는 bigint 값입니다.

Amazon Keyspaces의 기본 제공 함수

Amazon Keyspaces(Apache Cassandra용)는 Cassandra 쿼리 언어(CQL) 문에서 사용할 수 있는 다양한 기본 제공 함수를 지원합니다.

주제

스칼라 함수

스칼라 함수는 단일 값에 대해 계산을 수행하고 결과를 단일 값으로 반환합니다. Amazon Keyspaces는 다음과 같은 스칼라 함수를 지원합니다.

함수	설명
`blobAsType`	지정된 데이터 유형의 값을 반환합니다.
`cast`	하나의 네이티브 데이터 유형을 다른 네이티브 데이터 유형으로 변환합니다.
`currentDate`	현재 날짜/시간을 날짜로 반환합니다.
`currentTime`	현재 날짜/시간을 시간으로 반환합니다.
`currentTimestamp`	현재 날짜/시간을 타임스탬프로 반환합니다.
`currentTimeUUID`	현재 날짜/시간을 `timeuuid`으로 반환합니다.
`fromJson`	JSON 문자열을 선택한 열의 데이터 유형으로 변환합니다.
`maxTimeuuid`	타임스탬프 또는 날짜 문자열에 대해 가능한 가장 큰 `timeuuid` 값을 반환합니다.
`minTimeuuid`	타임스탬프 또는 날짜 문자열에 대해 가능한 가장 작은 `timeuuid` 값을 반환합니다.
`now`	새 고유한 `timeuuid` 항목을 반환합니다. `INSERT`, `UPDATE`, `DELETE` 문에 대해 `SELECT` 문 `WHERE` 절의 일부로 지원됩니다.
`toDate`	`timeuuid` 또는 타임스탬프를 날짜 유형으로 변환합니다.

함수	설명
toJson	선택한 열의 열 값을 JSON 형식으로 반환합니다.
token	파티션 키의 해시 값을 반환합니다.
toTimestamp	timeuuid 또는 날짜를 타임스탬프로 변환합니다.
TTL	열의 만료 시간을 초 단위로 반환합니다.
typeAsBlob	지정된 데이터 유형을 blob로 변환합니다.
toUnixTimestamp	timeuuid 또는 타임스탬프를 bigInt로 변환합니다.
uuid	무작위 버전 4 UUID를 반환합니다. INSERT, UPDATE, DELETE 문에 대해 SELECT 문 WHERE 절의 일부로 지원됩니다.
writetime	지정된 열 값의 타임스탬프를 반환합니다.
dateOf	(사용 중단) timeuuid의 타임스탬프를 추출하여 값을 날짜로 반환합니다.
unixTimestampOf	(사용 중단) timeuuid의 타임스탬프를 추출하여 값을 원시 64비트 정수 타임스탬프로 반환합니다.

Amazon Keyspaces(Apache Cassandra용)에 대한 할당량

이 섹션에서는 Amazon Keyspaces(Apache Cassandra용)의 현재 할당량 및 기본값에 대해 설명합니다.

주제

Amazon Keyspaces 서비스 할당량

다음 테이블에는 Amazon Keyspaces(Apache Cassandra용) 할당량과 기본값이 나와 있습니다. 조정할 수 있는 할당량에 대한 정보는 Service Quotas 콘솔에서 확인할 수 있으며 여기서 할당량 증가를 요청할 수도 있습니다. 할당량에 대한 자세한 내용은 문의하세요. AWS Support

할당량	설명	Amazon Keyspaces 기본값
1개당 최대 키스페이스 AWS 리전	리전당 이 구독자의 최대 키스페이스 수입니다. Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	256
개당 최대 테이블 수 AWS 리전	리전당 이 구독자의 모든 키스페이스에 대한 최대 테이블 수입니다. Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	256
최대 테이블 스키마 크기	테이블 스키마의 최대 크기입니다.	350KB
최대 동시 DDL 작업 수	리전당 이 구독자에게 허용되는 최대 동시 DDL 작업 수입니다.	50

할당량	설명	Amazon Keyspaces 기본값
연결당 최대 쿼리 수	초당 단일 클라이언트 TCP 연결로 처리할 수 있는 최대 CQL 쿼리 수입니다.	3000
최대 행 크기	정적 열 데이터를 제외한 행의 최대 크기입니다. 자세한 내용은 the section called “행 크기 계산” 섹션을 참조하세요.	1MB
INSERT 및 UPDATE 문의 최대 열 수	CQL INSERT 또는 UPDATE 문에 허용되는 최대 열 수입니다. TTL(Time To Live)이 꺼져 있는 경우 INSERT 또는 UPDATE 문은 최대 225개의 일반 열을 지원합니다. TTL이 켜져 있는 경우 한 번의 작업으로 최대 166개의 일반 열을 수정할 수 있습니다.	225/166
논리적 파티션당 최대 정적 데이터	논리적 파티션에 있는 정적 데이터의 최대 총 크기입니다. 자세한 내용은 the section called “논리적 파티션당 정적 열 크기 계산” 섹션을 참조하세요.	1MB
IN SELECT 문당 최대 하위 쿼리 수	SELECT 문에서 IN 키워드에 사용할 수 있는 하위 쿼리의 최대 수입니다. Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	100

할당량	설명	Amazon Keyspaces 기본값
중첩된 프로즌 컬렉션의 최대 개수당 AWS 리전	컬렉션 데이터 유형의 열에 FROZEN 키워드를 사용할 때 지원되는 중첩된 컬렉션의 최대 수입니다. 프로즌 수집에 대한 자세한 내용은 the section called “컬렉션 유형” 섹션을 참조하세요. 중첩 수준을 높이려면 다음 연락처로 문의하십시오. AWS Support	5
초당 최대 읽기 처리량	리전당 테이블에 할당할 수 있는 초당 최대 읽기 처리량(읽기 요청 단위(RRU) 또는 읽기 용량 단위(RCU)). Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	40,000
초당 최대 쓰기 처리량	리전당 테이블에 할당할 수 있는 초당 최대 쓰기 처리량(쓰기 요청 단위(WRU) 또는 쓰기 용량 단위(WCU)). Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	40,000
계정 수준 읽기 처리량(프로비저닝됨)	지역별 계정에 할당된 최대 총 읽기 용량 단위 (RCU) 수입니다. 이는 프로비저닝된 읽기/쓰기 용량 모드의 테이블에만 적용됩니다. Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	80,000

할당량	설명	Amazon Keyspaces 기본값
계정 수준 쓰기 처리량(프로비저닝됨)	지역별 계정에 할당된 최대 총 쓰기 용량 단위 (WCU) 수입니다. 이는 프로비저닝된 읽기/쓰기 용량 모드의 테이블에만 적용됩니다. Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	80,000
계정당 지역별 확장 가능한 대상의 최대 수	지역별 계정의 확장 가능한 대상의 최대 수 Amazon Keyspaces 테이블은 읽기 용량에 대해 Auto Scaling이 활성화된 경우 하나의 확장 가능한 대상으로 간주되고, 쓰기 용량에 대해 Auto Scaling이 활성화된 경우 또 다른 확장 가능한 대상으로 간주됩니다. Amazon Keyspaces의 확장 가능한 대상을 선택하여 Application Auto Scaling용 Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	1,500
최대 파티션 키 크기	복합 파티션 키의 최대 크기입니다. 메타데이터용 파티션 키에 포함된 각 열의 원시 크기에는 최대 3바이트의 추가 스토리지가 추가됩니다.	2048바이트
최대 클러스터링 키 크기	모든 클러스터링 열의 최대 합산 크기입니다. 메타데이터용 각 클러스터링 열의 원시 크기에는 최대 4바이트의 추가 스토리지가 추가됩니다.	850바이트

할당량	설명	Amazon Keyspaces 기본값
P 복구 (PITR) 를 사용한 최대 동시 테이블 복원 oint-in-time	구독자당 PITR을 사용한 최대 동시 테이블 복원 수는 4입니다. Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	4
point-in-time 복구 (PITR) 를 사용하여 복원한 최대 데이터 양	PITR을 사용하여 24시간 이내에 복원할 수 있는 최대 데이터 크기입니다. Service Quotas 콘솔에서 이 기본값을 조정할 수 있습니다.	5TB

처리량 늘리기 또는 줄이기(프로비저닝된 테이블의 경우)

프로비저닝된 처리량 늘리기

`ReadCapacityUnits`콘솔이나 명령문을 사용하여 필요한 `WriteCapacityUnits` 만큼 늘리거나 늘릴 수 있습니다. `ALTER TABLE` 새 설정은 `ALTER TABLE` 작업이 완료된 후에 적용됩니다.

프로비저닝된 용량을 추가할 때는 계정당 할당량을 초과할 수 없습니다. 또한 테이블의 프로비저닝 용량을 필요한 만큼 늘릴 수 있습니다. 계정당 할당량에 대한 자세한 내용은 앞의 the section called "Amazon Keyspaces 서비스 할당량" 섹션을 참조하세요.

프로비저닝된 처리량 줄이기

`ALTER TABLE` 문의 모든 테이블에 대해 `ReadCapacityUnits` 또는 `WriteCapacityUnits`(또는 둘 다)를 줄일 수 있습니다. 새 설정은 `ALTER TABLE` 작업이 완료된 후에 적용됩니다.

하루 중 원하는 시간에 최대 네 번까지 줄일 수 있습니다. 하루는 협정 세계시(UTC)에 따라 정의됩니다. 또한 지난 한 시간 동안 처리량을 줄이지 않은 경우 추가로 줄일 수 있습니다. 이에 따라 하루에 줄일 수 있는 최대 횟수를 27로 설정할 수 있습니다(처음 1시간 동안 4번 줄이기, 이후 1시간마다 1회 줄이기(당일 기준)).

Amazon Keyspaces 저장 시 암호화

AWS 소유 AWS KMS 키와 고객 관리 AWS KMS 키 간의 암호화 옵션을 테이블이 생성된 시점부터 24시간 동안 테이블별로 최대 4회까지 변경할 수 있습니다. 지난 6시간 내 변경 사항이 없는 경우 추가 변경이 허용됩니다. 이에 따라 하루에 변경할 수 있는 최대 횟수를 8회로 설정할 수 있습니다(처음 6시간 동안 4회 변경, 이후 6시간마다 1회 변경(당일 기준)).

이전 할당량을 모두 사용했더라도 필요한 만큼 자주 AWS 소유 AWS KMS 키를 사용하도록 암호화 옵션을 변경할 수 있습니다.

이러한 할당량은 용량 증가를 요청하지 않은 경우에 적용됩니다. 서비스 할당량 증가를 요청하려면 을 참조하십시오 AWS Support.

Amazon Keyspaces(Apache Cassandra용) 문서 기록

다음 표에서는 Amazon Keyspaces(Apache Cassandra용)의 최신 릴리스 이후 이 설명서에서 변경된 중요 사항에 대해 설명합니다. 이 설명서에 대한 업데이트 알림을 받으려면 RSS 피드를 구독하면 됩니다.

- 최신 설명서 업데이트: 2024년 2월 7일

변경 사항	설명	날짜
아마존 엘라스틱 쿠버네티스 서비스에서 아마존 키스페이스에 연결	이제 step-by-step 자습서를 따라 Amazon EKS에서 Amazon Keyspace에 연결할 수 있습니다.	2024년 2월 7일
프로비저닝된 테이블을 위한 Amazon Keyspaces 자동 스케일링 API	Amazon Keyspace는 CQL 이제 프로비저닝된 용량 모드로 자동 스케일링을 설정하는 데 필요한 AWS API 지원을 제공합니다.	2024년 1월 23일
프로비저닝된 테이블에 대한 Amazon Keyspaces 다중 지역 복제 지원	Amazon Keyspace는 이제 다중 리전 테이블에 프로비저닝된 용량 모드를 지원합니다.	2024년 1월 23일
Amazon Keyspaces DML 활동이 로그에 포함됨 CloudTrail	이제 AWS CloudTrail에서 Amazon Keyspaces 데이터 조작 언어(DML) API 직접 호출을 감사할 수 있습니다.	2023년 12월 20일
FROZEN 키워드에 대한 Amazon Keyspaces 지원	Amazon Keyspaces가 이제 수집 데이터 유형에 대한 FROZEN 키워드를 지원합니다.	2023년 11월 15일
Amazon Keyspaces 관리형 정책 업데이트	Amazon Keyspaces는 Amazon EC2 인스턴스에 대한 인터페이스 VPC 엔드포	2023년 10월 3일

	인트 액세스를 통해 Amazon Keyspaces에 접속하는 클라이언트가 VPC의 네트워크 정보로 Amazon Keyspaces `system.peers` 테이블을 업데이트할 수 있도록 하는 새로운 권한을 `AmazonKeyspacesFullAccess` 관리형 정책에 추가했습니다.	
Amazon Keyspaces 관리형 정책 업데이트	Amazon Keyspaces는 Amazon EC2 인스턴스에 대한 인터페이스 VPC 엔드포인트 액세스를 통해 Amazon Keyspaces에 접속하는 클라이언트가 VPC의 네트워크 정보로 Amazon Keyspaces `system.peers` 테이블을 업데이트할 수 있도록 하는 새로운 `AmazonKeyspacesReadOnlyAccess_v2` 관리형 정책을 만들었습니다.	2023년 9월 12일
Amazon Keyspaces에서의 연결 생성 모범 사례	Amazon Keyspaces에서 클라이언트 드라이버 구성을 개선하고 최적화하는 방법을 알아보세요.	2023년 6월 30일
Amazon Keyspaces의 시스템 키스페이스 문서화	시스템 키스페이스에 저장된 내용과 Amazon Keyspaces에서 유용한 정보를 찾기 위해 시스템 키스페이스에 저장된 내용을 쿼리하는 방법을 알아보세요.	2023년 6월 21일

Amazon Keyspaces의 다중 리전 복제 지원	Amazon Keyspaces 다중 리전 복제를 통해 향상된 내결함성, 안정성, 복원성을 제공하여 전 세계에 분산된 애플리케이션을 유지 관리할 수 있습니다.	2023년 6월 5일
Amazon Keyspaces 관리형 정책 업데이트	Amazon Keyspaces는 관리자가 다중 리전 키스페이스를 생성할 때 Amazon Keyspaces가 서비스 연결 역할을 생성할 수 있도록 `AmazonKeyspacesFullAccess` 관리형 정책에 새로운 권한을 추가했습니다.	2023년 6월 5일
`IN` 키워드에 대한 Amazon Keyspaces 지원	Amazon Keyspaces가 `SELECT` 명령문에서 `IN` 키워드를 지원합니다.	2023년 4월 25일
Amazon Keyspace와 인터페이스 VPC 엔드포인트에 대한 크로스 계정 액세스	VPC 엔드포인트를 사용하여 Amazon Keyspaces에 대한 크로스 계정 액세스를 구현하는 방법을 알아보세요.	2023년 4월 20일
Amazon Keyspaces의 클라이언트 측 타임스탬프 지원	Amazon Keyspaces 클라이언트 측 타임스탬프는 Cassandra와 호환되는 셀 수준 타임스탬프로, 여러 클라이언트가 동일한 데이터를 변경할 때 분산된 애플리케이션이 쓰기 작업 순서를 결정하는 데 도움이 됩니다.	2023년 3월 14일
Amazon Keyspaces 및 인터페이스 VPC 엔드포인트 시작하기	이 step-by-step 자습서에서는 VPC에서 Amazon Keyspace에 연결하는 방법을 알아봅니다.	2023년 3월 1일

Amazon Keyspaces 테이블 비용 최적화	기존 Amazon Keyspaces 테이블의 비용을 최적화하기 위한 전략을 식별하는 데 도움이 되는 모범 사례 및 지침이 제공됩니다.	2023년 2월 17일
`Murmur3Partitioner` 은(는) 이제 기본	`Murmur3Partitioner` 은(는) 이제 Amazon Keyspaces의 기본 파티셔너입니다.	2022년 11월 17일
Amazon Keyspaces의 `Murmur3Partitioner` 지원	이제 Amazon Keyspaces에서 `Murmur3Partitioner` 을(를) 사용할 수 있습니다.	2022년 11월 9일
빈 문자열 및 Blob 값에 대한 지원 업데이트	Amazon Keyspaces는 클러스터링 열 값으로 빈 문자열과 Blob 값도 지원합니다.	2022년 10월 19일
Amazon Keyspaces는 이제 다음에서 사용할 수 있습니다. AWS GovCloud (US)	Amazon Keyspaces는 이제 에서 사용할 수 AWS GovCloud (US) Region 있으며 FedRAMP의 엄격한 규정 준수의 적용 범위에 포함됩니다. 사용 가능한 엔드포인트에 대한 자세한 내용은 AWS GovCloud (US) Region FIPS 엔드포인트를 참조하세요.	2022년 8월 4일
아마존과 함께 아마존 키스페이스 테이블 스토리지 비용을 모니터링하세요 CloudWatch	Amazon Keyspaces는 이제 지표를 통해 시간 경과에 따른 테이블 스토리지 비용을 모니터링하고 추적할 수 있도록 지원합니다. `BillableTableSizeInBytes` CloudWatch	2022년 6월 14일

Amazon Keyspaces의 Terraform 지원	이제 Amazon Keyspaces에서 Terraform을 사용하여 데이터 정의 언어(DDL) 작업을 수행할 수 있습니다.	2022년 6월 9일
Amazon Keyspaces `token` 함수 지원	Amazon Keyspaces를 통해 `token` 함수를 사용하여 애플리케이션 쿼리를 최적화할 수 있습니다.	2022년 4월 19일
Amazon Keyspaces와 Apache Spark의 통합	Amazon Keyspaces를 통해 오픈 소스 Spark Cassandra 커넥터를 사용하여 Apache Spark에서 데이터를 더 쉽게 읽고 쓸 수 있습니다.	2022년 4월 19일
Amazon Keyspaces API 참조	Amazon Keyspace는 SDK 및 를 사용하여 키스페이스와 테이블을 관리하는 컨트롤 플레인 작업을 지원합니다. AWS AWS CLI API 참조 가이드는 지원되는 컨트롤 플레인 작업을 자세히 설명합니다.	2022년 3월 2일
Amazon Keyspaces를 사용할 때 흔히 발생하는 구성 문제를 해결하는 방법.	Amazon Keyspaces를 사용할 때 발생할 수 있는 일반적인 구성 문제를 해결하는 방법에 대해 자세히 알아보세요.	2021년 11월 22일
Amazon Keyspaces는 TTL(Time to Live)을 지원합니다.	Amazon Keyspaces TTL을 통해 테이블의 데이터를 자동으로 만료시켜 애플리케이션 로직을 간소화하고 스토리지 가격을 최적화할 수 있습니다.	2021년 10월 18일

DSBulk를 사용하여 데이터를 Amazon Keyspaces로 마이그레이션	벌크 로더 (DSBulk) 를 사용하여 DataStax Apache Cassandra에서 Amazon Keyspaces로 데이터를 마이그레이션하는 방법에 대한 tep-by-step 자습서입니다.	2021년 8월 9일
Amazon Keyspaces는 `system.peers` 테이블의 VPC 엔드포인트 항목을 지원합니다.	Amazon Keyspaces를 통해 `system.peers` 테이블에 사용 가능한 인터페이스 VPC 엔드포인트 정보를 입력하여 로드 밸런싱을 개선하고 읽기/쓰기 처리량을 늘릴 수 있습니다.	2021년 7월 29일
고객 관리형 키를 지원하도록 IAM 관리형 정책을 업데이트하십시오. AWS KMS	Amazon Keyspaces의 IAM 관리형 정책에는 이제 저장된 사용 가능한 고객 AWS KMS 관리 키를 나열하고 볼 수 있는 권한이 포함됩니다. AWS KMS	2021년 6월 1일
Amazon Keyspace는 고객 관리형 AWS KMS 키를 지원합니다.	Amazon Keyspaces를 사용하면 저장 중 암호화를 AWS KMS 위해 저장된 고객 관리 AWS KMS 키를 제어할 수 있습니다.	2021년 6월 1일
Amazon Keyspaces의 JSON 구문 지원	Amazon Keyspaces는 삽입 및 선택 작업을 위한 JSON 구문을 지원하므로 JSON 문서를 더 쉽게 읽고 쓸 수 있습니다.	2021년 1월 21일
Amazon Keyspaces의 정적 열 지원	Amazon Keyspaces를 통해 정적 열을 사용하여 여러 행 간의 공통 데이터를 효율적으로 업데이트하고 저장할 수 있습니다.	2020년 11월 9일

Amazon Keyspaces를 위한 NoSQL Workbench 지원 GA 릴리스	NoSQL Workbench는 Amazon Keyspaces의 비관계형 데이터 모델을 보다 쉽게 설계하고 시각화할 수 있는 클라이언트 측 애플리케이션입니다. NoSQL Workbench 클라이언트는 Windows, macOS, Linux에서 사용할 수 있습니다.	2020년 10월 28일
Amazon Keyspaces를 위한 NoSQL Workbench 지원 평가판 릴리스	NoSQL Workbench는 Amazon Keyspaces의 비관계형 데이터 모델을 보다 쉽게 설계하고 시각화할 수 있는 클라이언트 측 애플리케이션입니다. NoSQL Workbench 클라이언트는 Windows, macOS, Linux에서 사용할 수 있습니다.	2020년 10월 5일
Amazon Keyspaces에 프로그래밍 방식으로 액세스하기 위한 새로운 코드 예제	Amazon Keyspaces에 프로그래밍 방식으로 액세스하기 위한 코드 예제를 계속해서 추가하고 있습니다. 이제 Apache Cassandra 버전 3.11.2를 지원하는 Java, Python, Go, C#, Perl Cassandra 드라이버의 샘플을 사용할 수 있습니다.	2020년 7월 17일
아마존 키스페이스 point-in-time 복구 (PITR)	Amazon Keyspaces는 이제 테이블 데이터의 연속 백업을 제공하여 실수로 인한 쓰기 또는 삭제 작업으로부터 테이블을 보호하는 데 도움이 되는 point-in-time 복구 (PITR) 를 제공합니다.	2020년 7월 9일

Amazon Keyspaces 정식 출시	평가판에서 이전에 Amazon Managed Apache Cassandra Service(MCS)로 알려졌던 Amazon Keyspaces를 사용하면 현재 이미 사용하고 있는 Cassandra 쿼리 언어(CQL) 코드, Apache 2.0 라이선스 Cassandra 드라이버, 개발자 도구를 사용할 수 있습니다.	2020년 4월 23일
Amazon Keyspaces 자동 크기 조정	Amazon Keyspaces(Apache Cassandra용)는 Application Auto Scaling과 통합되어 처리량 용량을 자동으로 조정하여 실제 애플리케이션 트래픽에 따라 가변 워크로드에 대한 처리량 용량을 효율적으로 프로비저닝할 수 있도록 지원합니다.	2020년 4월 23일
Amazon Keyspaces의 인터페이스 Virtual Private Cloud(VPC) 엔드포인트	Amazon Keyspaces는 서비스와 VPC 간의 프라이빗 통신을 제공하여 네트워크 트래픽이 Amazon 네트워크를 벗어나지 않도록 합니다.	2020년 4월 16일
태그 기반 액세스 정책	IAM 정책의 리소스 태그를 사용하여 Amazon Keyspaces에 대한 액세스를 관리할 수 있습니다.	2020년 4월 8일
카운터 데이터 유형	Amazon Keyspaces를 통해 카운터를 사용하여 열 값의 증가 단위와 감소 단위를 조정할 수 있습니다.	2020년 4월 7일

리소스에 태그 지정	Amazon Keyspace를 통해 태그를 사용하여 리소스에 레이블을 지정하고 분류할 수 있습니다.	2020년 3월 31일
AWS CloudFormation 지원	Amazon Keyspaces를 통해 AWS CloudFormation을 사용하여 리소스의 생성 및 관리를 자동화할 수 있습니다.	2020년 3월 25일
IAM 역할 및 정책과 SigV4 인증에 대한 지원	IAM AWS Identity and Access Management (IAM) 을 사용하여 액세스 권한을 관리하고 Amazon Keyspace의 보안 정책을 구현하는 방법 및 Cassandra용 Java Driver용 인증 플러그인을 사용하여 IAM 역할 및 페더레이션 ID를 사용하여 프로그래밍 방식으로 Amazon Keyspaces에 액세스하는 방법에 대한 정보가 추가되었습니다. DataStax	2020년 3월 17일
읽기/쓰기 용량 모드	Amazon Keyspaces는 두 가지 읽기/쓰기 처리량 용량 모드를 지원합니다. 읽기/쓰기 용량 모드는 읽기 및 쓰기 처리량에 대한 청구 방식과 테이블 처리량 용량 관리 방식을 제어합니다.	2020년 2월 20일
최초 릴리스	이 설명서는 Amazon Keyspaces(Apache Cassandra용)의 초기 릴리스에 대한 내용입니다.	2019년 12월 3일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.