aws

개발자 가이드

Amazon Timestream

Amazon Timestream: 개발자 가이드

Table of Contents

용 Amazon Timestream이란 LiveAnalytics무엇입니까?

용 Amazon Timestream LiveAnalytics 은 하루에 수조 개의 시계열 데이터 포인트를 쉽게 저장하고 분석할 수 있는 빠르고 확장 가능하며 완전 관리되고 특수 설계된 시계열 데이터베이스입니다. 의 Timestream을 사용하면 최신 데이터를 메모리에 유지하고 사용자 정의 정책에 따라 기록 데이터를 비용 최적화 스토리지 계층으로 이동하여 시계열 데이터의 수명 주기를 관리하는 데 드는 시간과 비용을 LiveAnalytics 절약할 수 있습니다. LiveAnalytics의 전용 쿼리 엔진에 대한 Timestream을 사용하면 위치를 지정하지 않고도 최근 및 과거 데이터에 액세스하고 분석할 수 있습니다. 용 Amazon Timestream LiveAnalytics 에는 시계열 분석 기능이 내장되어 있어 거의 실시간으로 데이터의 추세와 패턴을 식별할 수 있습니다. 의 Timestream LiveAnalytics 은 서버리스이며 용량 및 성능을 조정하기 위해 자동으로 확장 또는 축소됩니다. 기본 인프라를 관리할 필요가 없으므로 애플리케이션을 최적화하고 구축하는 데 집중할 수 있습니다.

LiveAnalytics 또한 의 Timestream은 데이터 수집, 시각화 및 기계 학습에 일반적으로 사용되는 서비스와 통합됩니다. AWS IoT Core, Amazon Kinesis , Amazon MSK및 오픈 소스 Telegraf를 LiveAnalytics 사용하여 Amazon Timestream으로 데이터를 전송할 수 있습니다. 를 통해 Amazon QuickSight, Grafana 및 비즈니스 인텔리전스 도구를 사용하여 데이터를 시각화할 수 있습니다JDBC. 기계 학습을 위해 Amazon SageMaker 을 LiveAnalytics 용 Timestream과 함께 사용할 수도 있습니다.

LiveAnalytics 주요 이점을 위한 타임스트림

에 대한 Amazon Timestream의 주요 이점은 다음과 LiveAnalytics 같습니다.

- 자동 크기 조정 기능이 있는 서버리스 - 용 Amazon Timestream을 사용하면 관리할 LiveAnalytics서버와 프로비저닝할 용량이 없습니다. 애플리케이션의 요구 사항이 변경되면 의 Timestream이 용량을 조정하기 위해 LiveAnalytics 자동으로 확장됩니다.
- 데이터 수명 주기 관리 - Amazon Timestream for 는 데이터 수명 주기 관리의 복잡한 프로세스를 LiveAnalytics 간소화합니다. 최근 데이터를 위한 메모리 스토어와 과거 데이터를 위한 마그네틱 스토어를 통해 스토리지 계층화를 제공합니다. Amazon Timestream은 사용자 구성 가능 정책에 따라 메모리 스토어에서 마그네틱 스토어로 데이터 전송을 자동화합니다.
- 간소화된 데이터 액세스 - 용 Amazon Timestream을 사용하면 더 이상 서로 다른 도구를 사용하여 최근 및 과거 데이터에 액세스할 필요가 LiveAnalytics없습니다. LiveAnalytics의 전용 쿼리 엔진용 Amazon Timestream은 데이터 위치를 지정할 필요 없이 스토리지 계층 간에 데이터에 투명하게 액세스하고 결합합니다.

- 시계열용으로 특별히 구축 - 를 사용하여 시계열 데이터를 빠르게 분석할 수 있으며SQL, 평활화, 근사화 및 보간을 위한 시계열 함수가 내장되어 있습니다. 의 Timestream은 고급 집계, 창 함수 및 배열 및 행과 같은 복잡한 데이터 유형 LiveAnalytics 도 지원합니다.
- 항상 암호화됨 - Amazon Timestream for 는 저장 중이든 전송 중이든 시계열 데이터가 항상 암호화되도록 LiveAnalytics 합니다. 용 Amazon Timestream을 사용하면 마그네틱 스토어에서 데이터를 암호화하기 위한 AWS KMS 고객 관리형 키(CMK)를 지정할 LiveAnalytics 수도 있습니다.
- 고가용성 - Amazon Timestream은 단일 AWS 리전 내에서 데이터를 자동으로 복제하고 최소 3개의 서로 다른 가용 영역에 리소스를 할당하여 쓰기 및 읽기 요청의 고가용성을 보장합니다. 자세한 내용은 Timestream Service Level Agreement를 참조하세요.
- 내구성 - Amazon Timestream은 단일 AWS 리전 내의 다양한 가용 영역에 메모리 및 마그네틱 스토어 데이터를 자동으로 복제하여 데이터의 내구성을 보장합니다. 쓰기 요청이 완료되었음을 확인하기 전에 모든 데이터가 디스크에 기록됩니다.

LiveAnalytics 사용 사례의 타임스트림

에 대한 Timestream 사용 사례의 목록이 늘어나는 예는 다음과 LiveAnalytics 같습니다.

- 지표를 모니터링하여 애플리케이션의 성능과 가용성을 개선합니다.
- 산업 원격 측정의 저장 및 분석을 통해 장비 관리 및 유지 관리를 간소화합니다.
- 시간 경과에 따라 애플리케이션과의 사용자 상호 작용을 추적합니다.
- IoT 센서 데이터의 저장 및 분석.

에 대한 Timestream 시작하기 LiveAnalytics

다음 단원을 읽고 시작하면 도움이 됩니다.

- 튜토리얼 - 샘플 데이터 세트로 채워진 데이터베이스를 생성하고 샘플 쿼리를 실행합니다.
- LiveAnalytics Amazon Timestream 개념 - LiveAnalytics 개념에 대한 필수 Timestream을 학습합니다.
- 에 대한 Timestream 액세스 LiveAnalytics - 콘솔 AWS CLI또는 를 LiveAnalytics 사용하여 Timestream에 액세스하는 방법을 알아봅니다API.
- 할당량 - 프로비저닝할 수 있는 LiveAnalytics 구성 요소의 Timestream 수에 대한 할당량에 대해 알아봅니다.

용 Timestream용 애플리케이션 개발을 빠르게 시작하는 방법을 알아보려면 다음을 LiveAnalytics참조하세요.

- 사용 AWS SDKs
- 쿼리 언어 참조

작동 방식

다음 섹션에서는 Amazon Timestream for Live Analytics 서비스 구성 요소와 상호 작용하는 방식에 대한 개요를 제공합니다.

이 소개를 읽은 후 에 대한 Timestream 액세스 LiveAnalytics 섹션을 참조하여 콘솔 AWS CLI또는 를 사용하여 Timestream for Live Analytics에 액세스하는 방법을 알아봅니다SDKs.

주제

- LiveAnalytics Amazon Timestream 개념
- 아키텍처
- 쓰기
- 스토리지
- 쿼리
- 예약된 쿼리
- Timestream Compute Unit(TCU)

LiveAnalytics Amazon Timestream 개념

시계열 데이터는 일정 시간 간격에 걸쳐 기록된 일련의 데이터 포인트입니다. 이 유형의 데이터는 시간 경과에 따라 변화하는 이벤트를 측정하는 데 사용됩니다. 예는 다음과 같습니다.

- 시간 경과에 따른 주가
- 시간 경과에 따른 온도 측정
- CPU 시간 경과에 따른 EC2 인스턴스 사용률

시계열 데이터의 경우 각 데이터 포인트는 타임스탬프, 하나 이상의 속성 및 시간이 지남에 따라 변경되는 이벤트로 구성됩니다. 이 데이터를 사용하여 애플리케이션의 성능과 상태에 대한 인사이트를 도출하고, 이상을 감지하고, 최적화 기회를 식별할 수 있습니다. 예를 들어, DevOps 엔지니어는 인프라

성능 지표의 변화를 측정하는 데이터를 보고 싶을 수 있습니다. 제조업체는 시설 전반의 장비 변화를 측정하는 IoT 센서 데이터를 추적하려고 할 수 있습니다. 온라인 마케터는 시간이 지남에 따라 사용자가 웹 사이트를 탐색하는 방법을 캡처하는 클릭스트림 데이터를 분석하려고 할 수 있습니다. 시계열 데이터는 매우 많은 볼륨의 여러 소스에서 생성되므로 거의 실시간으로 비용 효율적으로 수집해야 하므로 데이터를 구성하고 분석하는 데 도움이 되는 효율적인 스토리지가 필요합니다.

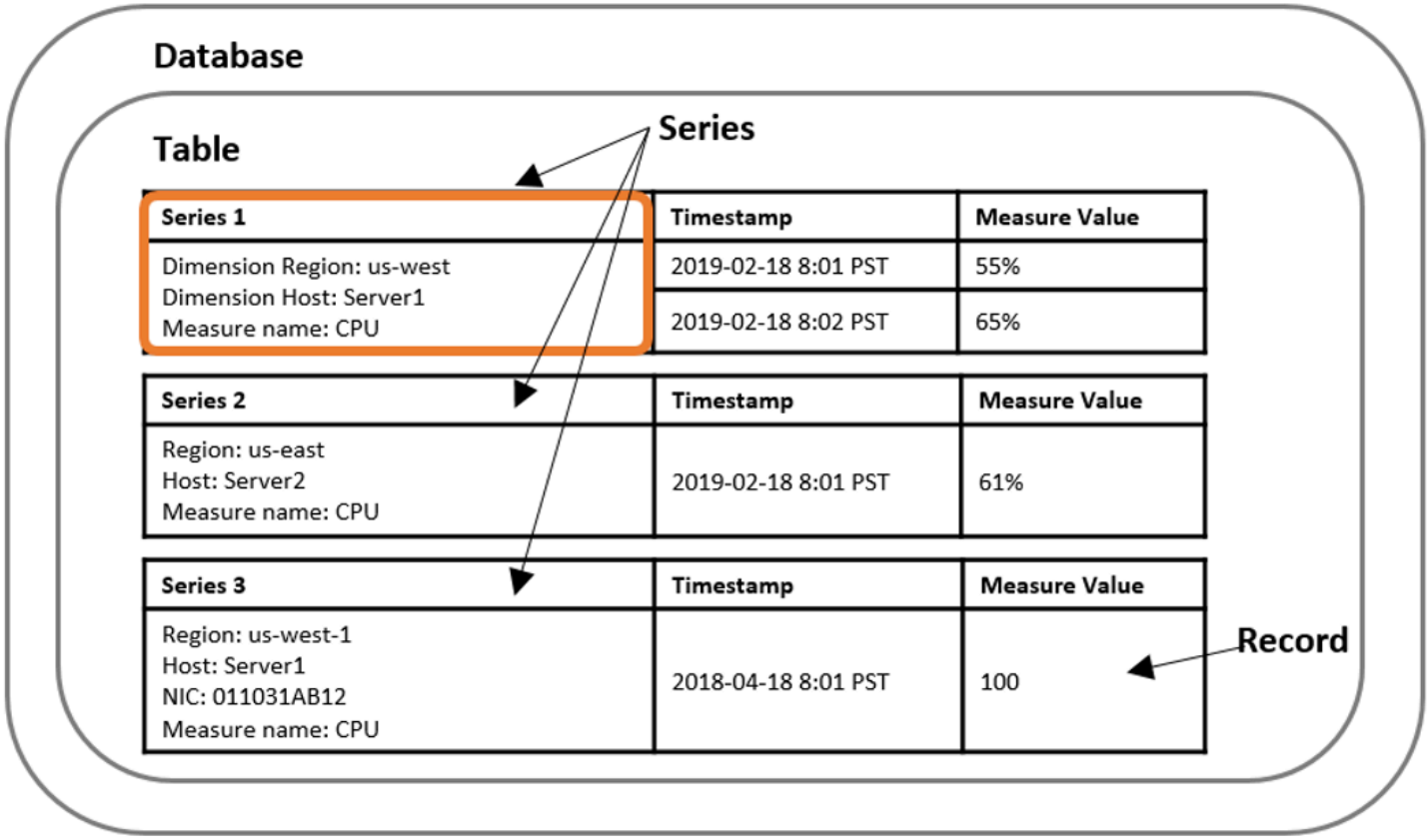
다음은 용 Timestream의 주요 개념입니다 LiveAnalytics.

- 시계열 - 일정 시간 간격 동안 기록된 하나 이상의 데이터 포인트(또는 레코드) 시퀀스입니다. 예를 들어 시간 경과에 따른 주식 가격, 시간 경과에 따른 EC2 인스턴스의 CPU 또는 메모리 사용률, 시간 경과에 따른 IoT 센서의 온도/압력 판독값이 있습니다.
- 레코드 - 시계열의 단일 데이터 포인트입니다.
- 차원 - 시계열의 메타 데이터를 설명하는 속성입니다. 차원은 차원 이름과 차원 값으로 구성됩니다. 다음 예제를 살펴보세요.
  - 증권거래소를 차원으로 고려할 때 차원 이름은 “주식거래소”이고 차원 값은 “NYSE”입니다.
  - AWS 리전을 차원으로 고려할 때 차원 이름은 “ 리전”이고 차원 값은 “us-east-1”입니다.
  - IoT 센서의 경우 차원 이름은 “장치 ID”이고 차원 값은 “12345”입니다.
- 측정 - 레코드에서 측정 중인 실제 값입니다. 예를 들어 주가, CPU 또는 메모리 사용률, 온도 또는 습도 판독값이 있습니다. 측정값은 측정값 이름과 측정값으로 구성됩니다. 다음 예제를 살펴보세요.
  - 주가의 경우 측정값 이름은 '주가'이고 측정값은 특정 시점의 실제 주가입니다.
  - CPU 사용률의 경우 측정값 이름은 “CPU사용률”이고 측정값은 실제 CPU 사용률입니다.

  측정값은 Timestream에서 다중 측정 또는 단일 측정 레코드 LiveAnalytics 로 모델링할 수 있습니다. 자세한 내용은 다중 측정 레코드와 단일 측정 레코드 비교 단원을 참조하십시오.
- 타임스탬프 - 지정된 레코드에 대해 측정값이 수집된 시기를 나타냅니다. 의 Timestream은 나노초 단위로 타임스탬프를 LiveAnalytics 지원합니다.
- 표 - 관련 시계열 세트를 위한 컨테이너입니다.
- 데이터베이스 - 테이블의 최상위 컨테이너입니다.

LiveAnalytics 개념에 대한 Timestream 요약

데이터베이스에는 0개 이상의 테이블이 포함되어 있습니다. 각 테이블에는 0개 이상의 시계열 이 포함되어 있습니다. 각 시계열은 지정된 세분화 에서 지정된 시간 간격에 걸친 일련의 레코드로 구성됩니다. 각 시계열은 메타데이터 또는 차원 , 데이터 또는 측정값 , 타임스탬프 를 사용하여 설명할 수 있습니다.

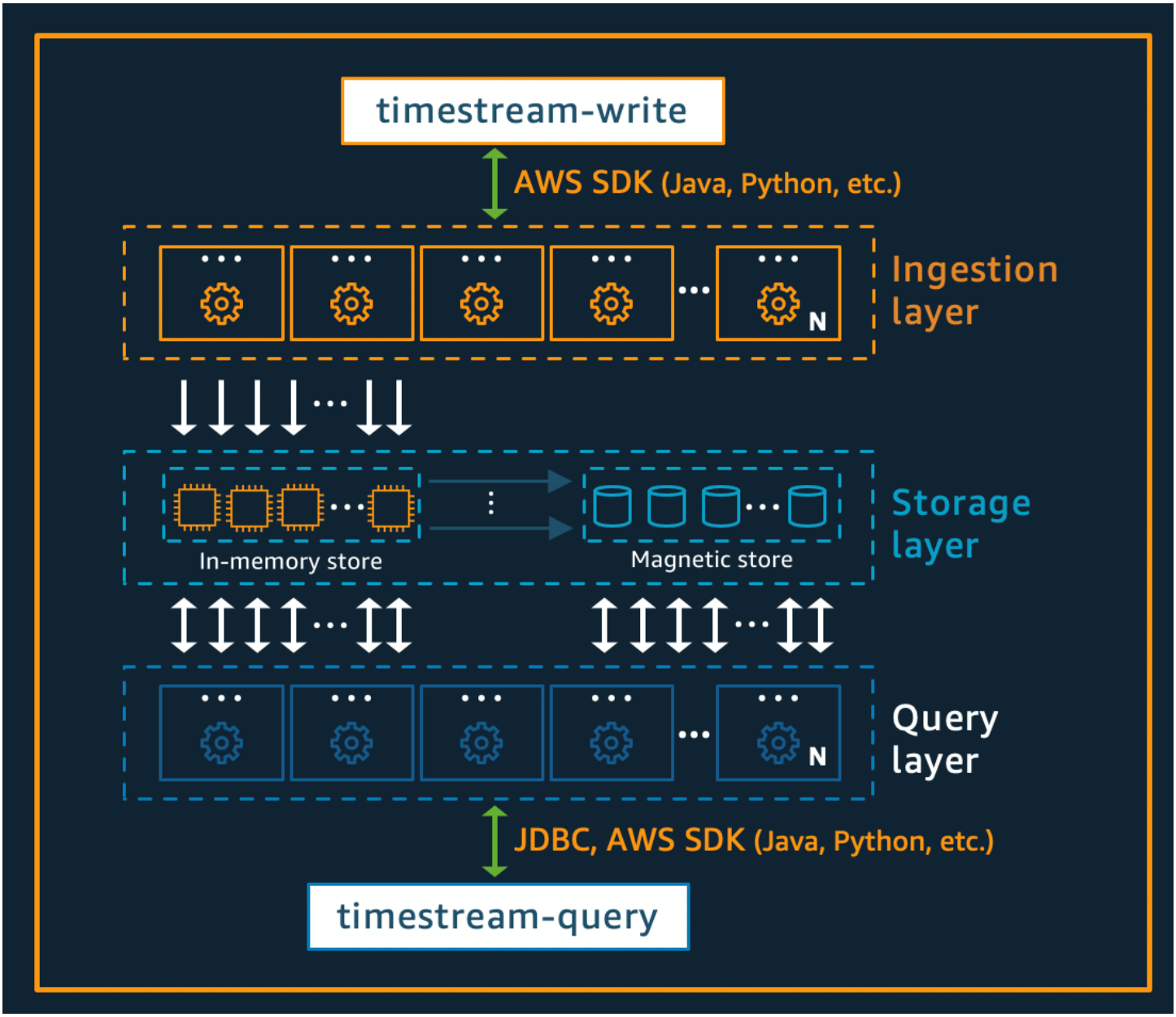


아키텍처

Amazon Timestream for Live Analytics는 처음부터 대규모로 시계열 데이터를 수집, 저장 및 처리하도록 설계되었습니다. 서버리스 아키텍처는 독립적으로 확장할 수 있는 완전히 분리된 데이터 수집, 스토리지 및 쿼리 처리 시스템을 지원합니다. 이 설계는 각 하위 시스템을 간소화하여 확고한 안정성을 달성하고, 병목 현상을 조정하지 않으며, 상관 관계가 있는 시스템 장애 가능성을 줄일 수 있습니다. 이러한 각 요인은 시스템이 확장됨에 따라 더 중요해집니다.

주제

- 쓰기 아키텍처
- 스토리지 아키텍처
- 쿼리 아키텍처
- 셀룰러 아키텍처



쓰기 아키텍처

시계열 데이터를 작성할 때 Amazon Timestream for Live Analytics는 테이블, 파티션에 대한 쓰기를 처리량이 높은 데이터 쓰기를 처리하는 내결함성 메모리 스토어 인스턴스로 라우팅합니다. 그러면 메모리 스토어는 3개의 가용 영역()에서 데이터를 복제하는 별도의 스토리지 시스템에서 내구성을 달성합니다AZs. 복제는 노드 또는 전체 AZ의 손실이 쓰기 가용성을 방해하지 않도록 정족수를 기반으로 합니다. 거의 실시간으로 다른 인 메모리 스토리지 노드가 데이터에 동기화되어 쿼리를 제공합니다. 리더 복제본 노드AZs도 확장되어 높은 읽기 가용성을 보장합니다.

Timestream for Live Analytics는 처리량이 더 낮은 지연 도착 데이터를 생성하는 애플리케이션을 위해 마그네틱 스토어에 직접 데이터를 쓰는 것을 지원합니다. 지연 도착 데이터는 타임스탬프가 현재 시간보다 이전인 데이터입니다. 메모리 스토어의 높은 처리량 쓰기와 마찬가지로 마그네틱 스토어에 기록된 데이터는 3개에 복제AZs되고 복제는 쿼럼 기반입니다.

데이터가 메모리에 기록되든 마그네틱 스토어에 기록되든 관계없이 Timestream for Live Analytics는 스토리지에 기록되기 전에 데이터를 자동으로 인덱싱하고 분할합니다. 실시간 분석을 위한 단일 Timestream 테이블에는 수백, 수천 또는 수백만 개의 파티션이 있을 수 있습니다. 개별 파티션은 서로 직접 통신하지 않으며 데이터를 공유하지 않습니다(공유 없음 아키텍처). 대신 고가용성 파티션 추적 및 인덱싱 서비스를 통해 테이블의 파티셔닝을 추적합니다. 이렇게 하면 시스템 장애의 영향을 최소화하고 상관관계가 있는 장애 가능성을 크게 낮추기 위해 특별히 설계된 또 다른 우려 사항이 분리됩니다.

스토리지 아키텍처

데이터가 Timestream for Live Analytics에 저장되면 데이터는 데이터로 작성된 컨텍스트 속성에 따라 시간 순서 및 시간별로 구성됩니다. 시계열 시스템을 대규모로 조정하려면 시간 외에도 '공간'을 분할하는 파티셔닝 체계가 중요합니다. 이는 대부분의 시계열 데이터가 현재 시간 또는 그 즈음에 작성되기 때문입니다. 따라서 시간별 파티셔닝만으로는 쓰기 트래픽을 분산하거나 쿼리 시간에 데이터를 효과적으로 정리할 수 없습니다. 이는 극한 규모 시계열 처리에 중요하며, Timestream for Live Analytics는 현재 서버리스 방식으로 다른 선도적인 시스템보다 규모를 크게 조정할 수 있었습니다. 결과 파티션은 2차원 공간(유사한 크기로 설계됨)의 분할을 나타내기 때문에 “타일”이라고 합니다. Live Analytics 테이블의 Timestream은 단일 파티션(타일)으로 시작한 다음 처리량에 따라 공간 차원으로 분할됩니다. 타일이 특정 크기에 도달하면 데이터 크기가 증가함에 따라 더 나은 읽기 병렬화를 달성하기 위해 시간 차원으로 분할됩니다.

Timestream for Live Analytics는 시계열 데이터의 수명 주기를 자동으로 관리하도록 설계되었습니다. Timestream for Live Analytics는 인메모리 스토어와 비용 효율적인 마그네틱 스토어라는 두 개의 데이터 스토어를 제공합니다. 또한 스토어 간에 데이터를 자동으로 전송하도록 테이블 수준 정책을 구성하는 것도 지원합니다. 수신되는 처리량이 높은 데이터 쓰기는 메모리 스토어에 도착하며, 여기서 데이터는 쓰기에 최적화되고, 대시보드에 전원을 공급하고 유형 쿼리에 알림을 보내기 위해 현재 시간에 수행된 읽기도 마찬가지입니다. 쓰기, 알림 및 대시보드 요구 사항에 대한 기본 기간이 경과하면 메모리 스토어에서 마그네틱 스토어로 데이터가 자동으로 흐르도록 하여 비용을 최적화합니다. 실시간 분석을 위한 Timestream을 사용하면 이러한 목적으로 메모리 스토어에 데이터 보존 정책을 설정할 수 있습니다. 지연 도착 데이터에 대한 데이터 쓰기는 마그네틱 스토어에 직접 기록됩니다.

마그네틱 스토어에서 데이터를 사용할 수 있게 되면(메모리 스토어 보존 기간이 만료되거나 마그네틱 스토어에 직접 쓰기 때문에) 대용량 데이터 읽기에 매우 최적화된 형식으로 재구성됩니다. 또한 마그네

틱 스토어에는 데이터가 유용성을 벗어날 수 있는 시간 임계값이 있는 경우 구성할 수 있는 데이터 보존 정책이 있습니다. 데이터가 마그네틱 스토어 보존 정책에 정의된 시간 범위를 초과하면 자동으로 제거됩니다. 따라서 일부 구성 이외의 Timestream for Live Analytics를 사용하면 데이터 수명 주기 관리가 장면 뒤에서 원활하게 수행됩니다.

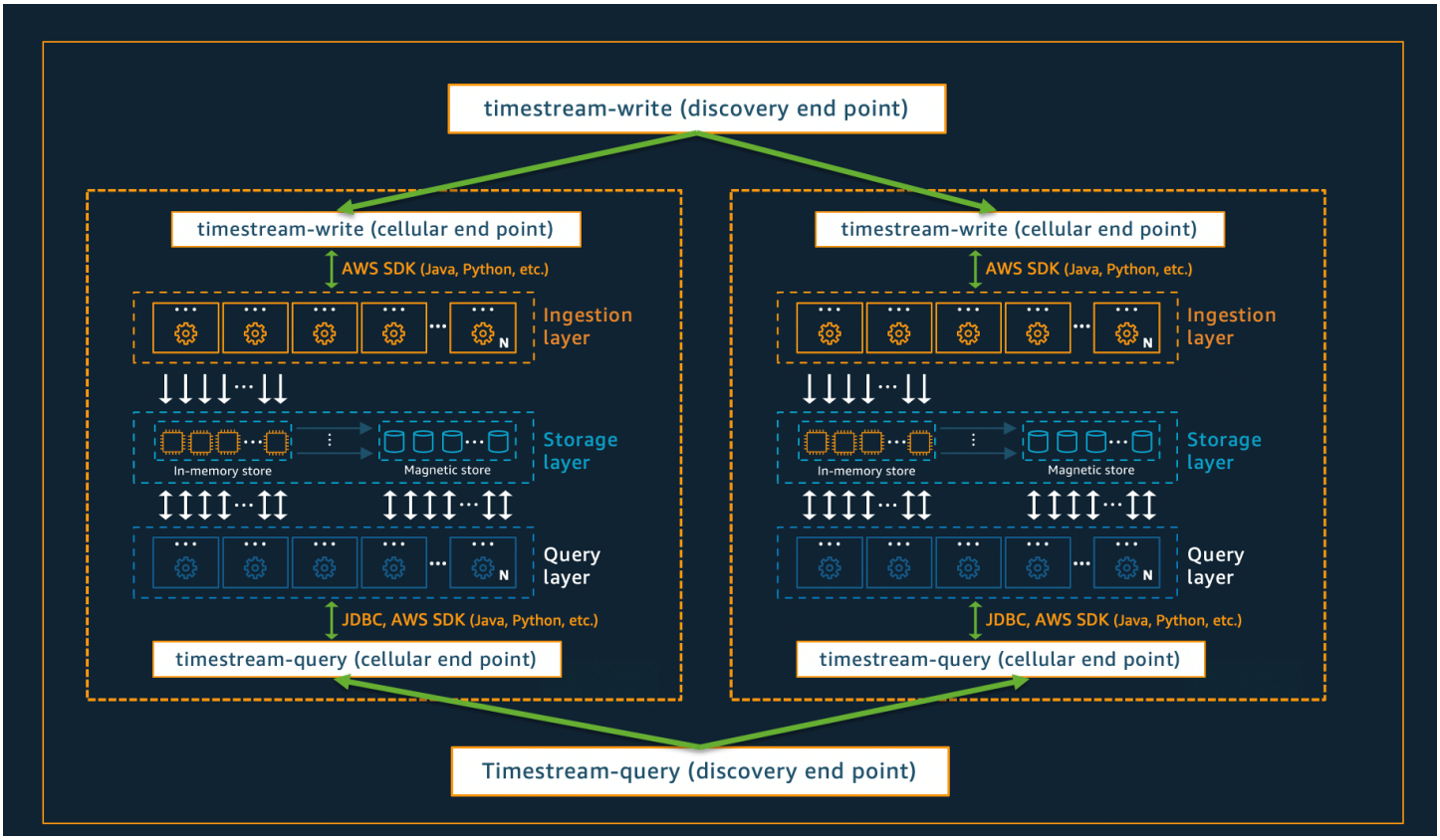
쿼리 아키텍처

Timestream for Live Analytics 쿼리는 시계열별 지원(시계열별 데이터 유형 및 함수)을 위한 확장이 있는 문SQL법으로 표현되므로 이미 에 익숙한 개발자는 학습 곡선을 쉽게 사용할 수 있습니다SQL. 그런 다음 쿼리는 타일 추적 및 인덱싱 서비스의 메타데이터를 사용하여 쿼리가 발행될 때 데이터 스토어 전반의 데이터에 원활하게 액세스하고 결합하는 적응형 분산 쿼리 엔진에 의해 처리됩니다. 이를 통해 Rube Goldberg의 많은 복잡성을 간단하고 익숙한 데이터베이스 추상화로 축소하여 고객과 잘 공감하는 경험을 얻을 수 있습니다.

쿼리는 지정된 쿼리를 실행하도록 등록된 작업자 수가 쿼리 복잡성 및 데이터 크기에 따라 결정되는 전용 작업자 플릿에서 실행됩니다. 대규모 데이터 세트에 대한 복잡한 쿼리의 성능은 쿼리 런타임 플릿과 시스템의 스토리지 플릿 모두에서 대규모 병렬 처리를 통해 달성됩니다. 대량의 데이터를 빠르고 효율적으로 분석하는 기능은 실시간 분석을 위한 Timestream의 가장 큰 강점 중 하나입니다. 테라바이트 또는 페타바이트 이상의 데이터에서 실행되는 단일 쿼리에는 수천 대의 시스템이 동시에 모두 작동할 수 있습니다.

셀룰러 아키텍처

Timestream for Live Analytics가 애플리케이션에 사실상 무한한 규모를 제공하는 동시에 99.99% 가용성을 보장하기 위해 이 시스템은 셀룰러 아키텍처를 사용하여 설계되었습니다. 시스템을 전체적으로 확장하는 대신 Timestream for Live Analytics 세그먼트를 셀 이라고 하는 여러 개의 더 작은 자체 복사본으로 분할합니다. 이렇게 하면 전체 규모로 셀을 테스트할 수 있으며 한 셀의 시스템 문제가 특정 리전의 다른 셀의 활동에 영향을 미치지 않습니다. Timestream for Live Analytics는 리전당 여러 셀을 지원하도록 설계되었지만 리전에 2개의 셀이 있는 다음과 같은 가상 시나리오를 고려하세요.



위에 표시된 시나리오에서 데이터 수집 및 쿼리 요청은 먼저 데이터 수집 및 쿼리에 대한 검색 엔드포인트에서 각각 처리됩니다. 그런 다음 검색 엔드포인트는 고객 데이터가 포함된 셀을 식별하고 해당 셀에 대한 적절한 수집 또는 쿼리 엔드포인트로 요청을 전달합니다. 를 사용하면 SDKs이러한 엔드포인트 관리 작업이 투명하게 처리됩니다.

> **Note**
>
> Timestream for Live Analytics와 함께 VPC 엔드포인트를 사용하거나 Timestream for Live Analytics에 대한 REST API 작업에 직접 액세스하는 경우 셀룰러 엔드포인트와 직접 상호 작용해야 합니다. 이렇게 하는 방법에 대한 지침은 VPC 엔드포인트를 설정하는 방법에 대한 지침을 참조하고VPC, REST API 작업의 직접 호출에 대한 지침은 엔드포인트 검색 패턴을 참조하세요.

쓰기

연결된 디바이스, IT 시스템 및 산업 장비에서 시계열 데이터를 수집하여 실시간 분석을 위한 Timestream에 쓸 수 있습니다. Timestream for Live Analytics를 사용하면 시계열이 동일한 테이블에 속할 때 단일 시계열의 데이터 포인트 및/또는 여러 시리즈의 데이터 포인트를 단일 쓰기 요청으로 작

성할 수 있습니다. 사용자의 편의를 위해 Timestream for Live Analytics는 데이터베이스에 쓰기를 호출할 때 지정한 측정값의 차원 이름과 데이터 유형을 기반으로 Timestream for Live Analytics 테이블의 열 이름과 데이터 유형을 자동으로 감지하는 유연한 스키마를 제공합니다. 실시간 분석을 위한 Timestream에 데이터 배치를 작성할 수도 있습니다.

> **Note**
>
> Timestream for Live Analytics는 읽기에 대한 최종 일관성 의미 체계를 지원합니다. 즉, Timestream for Live Analytics에 데이터 배치를 작성한 직후에 데이터를 쿼리할 때 쿼리 결과에는 최근에 완료된 쓰기 작업의 결과가 반영되지 않을 수 있습니다. 결과에는 일부 오래된 데이터도 포함될 수 있습니다. 마찬가지로, 하나 이상의 새 차원으로 시계열 데이터를 쓰는 동안 쿼리는 짧은 시간 동안 열의 부분 하위 집합을 반환할 수 있습니다. 잠시 후 이러한 쿼리 요청을 반복하면 결과가 최신 데이터를 반환해야 합니다.

AWS SDKs, , , AWS CLI, AWS IoT Core, Amazon Managed Service for Apache Flink, 를 통해 AWS Lambda, 또는 Amazon KinesisAmazon MSK를 사용하여 데이터를 쓸 수 있습니다오픈 소스 Telegraf.

주제

- 데이터 타입
- 사전 스키마 정의 없음
- 데이터 쓰기(삽입 및 업서트)
- 읽기에 대한 최종 일관성
- 로 쓰기 배치 WriteRecords API
- 배치 로드
- 작업과 배치 로드 중에서 WriteRecords API 선택

데이터 타입

Timestream for Live Analytics는 쓰기에 대해 다음 데이터 유형을 지원합니다.

데이터 유형	설명
BIGINT	64비트 부호 있는 정수를 나타냅니다.
BOOLEAN	로직의 두 가지 진실 값, 즉 true와 false를 나타냅니다.

데이터 유형	설명
DOUBLE	바이너리 부동 소수점 산술용 IEEE Standard 754를 구현하는 64비트 변수 정밀도입니다. **Note** 쿼리에 사용할 수 있는 `Infinity` 및 `NaN` 이중 값에 대한 쿼리 언어 함수가 있습니다. 하지만 이러한 값을 Timestream에 쓸 수는 없습니다.
VARCHAR	선택적 최대 길이가 있는 가변 길이 문자 데이터입니다. 최대 제한은 2KB입니다.
MULTI	다중 측정 레코드의 데이터 유형입니다. 이 데이터 유형에는 유형 `BIGINT`, , `BOOLEAN`, 및 의 하나 이상의 측정값`DOUBLE``VARCHAR`이 포함됩니다`TIMESTAMP` .
TIMESTAMP	Unix 시간 이후의 시간을 UTC추적하여 에서 나노초 정밀도 시간을 사용하여 인스턴스를 시간 단위로 나타냅니다. 이 데이터 유형은 현재 다중 측정 레코드(즉, 유형 의 측정값 내`MULTI`)에서만 지원됩니다. *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*sssssssss* 범위의 지원 타임스탬프`1970-01-01 00:00:00.000000000` 를 에 기록합니다`2262-04-11 23:47:16.854775807` .

사전 스키마 정의 없음

Amazon Timestream for Live Analytics로 데이터를 전송하기 전에 , AWS Management Console Timestream for Live Analytics SDKs또는 Timestream for Live Analytics API 작업을 사용하여 데이터베이스와 테이블을 생성해야 합니다. 자세한 내용은 데이터베이스 생성 및 테이블 생성 단원을 참조하세요. 테이블을 생성하는 동안 스키마를 미리 정의할 필요가 없습니다. Amazon Timestream for Live Analytics는 전송되는 데이터 포인트의 측정값과 차원을 기반으로 스키마를 자동으로 감지하므로 더 이상 스키마를 오프라인으로 변경하여 빠르게 변화하는 시계열 데이터에 적용할 필요가 없습니다.

데이터 쓰기(삽입 및 업서트)

Amazon Timestream for Live Analytics의 쓰기 작업을 사용하면 데이터를 삽입하고 업서트할 수 있습니다. 기본적으로 Amazon Timestream for Live Analytics의 쓰기는 첫 번째 라이터 승리 의미 체계를 따르며, 여기서 데이터는 추가용으로만 저장되고 중복 레코드는 거부됩니다. 첫 번째 라이터는 많은 시계열 애플리케이션의 요구 사항을 충족하지만, 애플리케이션이 이상적으로 기존 레코드를 업데이트하고/하거나 마지막 라이터와 함께 데이터를 작성해야 하는 시나리오가 있으며, 이 경우 가장 높은 버전의 레코드가 서비스에 저장됩니다. 이러한 시나리오를 해결하기 위해 Amazon Timestream for Live Analytics는 데이터를 업서트할 수 있는 기능을 제공합니다. Upsert는 레코드가 없을 때 레코드를 시스템에 삽입하거나 레코드가 있을 때 레코드를 업데이트하는 작업입니다. 레코드가 업데이트되면 등전성 방식으로 업데이트됩니다.

삭제할 레코드 수준 작업이 없습니다. 하지만 테이블과 데이터베이스를 삭제할 수 있습니다.

메모리 스토어와 마그네틱 스토어에 데이터 쓰기

Amazon Timestream for Live Analytics는 메모리 스토어와 마그네틱 스토어에 직접 데이터를 쓸 수 있는 기능을 제공합니다. 메모리 스토어는 처리량이 높은 데이터 쓰기에 최적화되어 있고 마그네틱 스토어는 도착 후 지연 데이터의 처리량이 낮은 쓰기에 최적화되어 있습니다.

지연 도착 데이터는 타임스탬프가 현재 시간보다 이전이고 메모리 스토어 보존 기간을 벗어난 데이터입니다. 테이블에 대한 마그네틱 스토어 쓰기를 활성화하여 지연 도착 데이터를 마그네틱 스토어에 쓸 수 있는 기능을 명시적으로 활성화해야 합니다. 또한 테이블`MagneticStoreRejectedDataLocation`이 생성될 때 정의됩니다. 마그네틱 스토어에 쓰려면 의 호출자에게 테이블 생성 `MagneticStoreRejectedDataLocation`중에 에 지정된 S3 버킷에 대한 `S3:PutObject` 권한이 `WriteRecords` 있어야 합니다. 자세한 내용은 , WriteRecords및 CreateTable섹션을 참조하세요PutObject.

단일 측정 레코드 및 다중 측정 레코드로 데이터 쓰기

Amazon Timestream for Live Analytics는 두 가지 유형의 레코드, 즉 단일 측정 레코드와 다중 측정 레코드를 사용하여 데이터를 쓸 수 있는 기능을 제공합니다.

단일 측정 레코드

단일 측정 레코드를 사용하면 레코드당 단일 측정값을 보낼 수 있습니다. 이 형식을 사용하여 데이터를 Timestream for Live Analytics로 전송하면 Timestream for Live Analytics는 레코드당 하나의 테이블 행을 생성합니다. 즉, 디바이스가 4개의 지표를 내보내고 각 지표가 단일 측정 레코드로 전송되는 경우 Timestream for Live Analytics는 테이블에 4개의 행을 생성하여 이 데이터를 저장하고 각 행에 대해 디

바이스 속성이 반복됩니다. 이 형식은 애플리케이션에서 단일 지표를 모니터링하려는 경우 또는 애플리케이션이 여러 지표를 동시에 내보내지 않는 경우에 권장됩니다.

다중 측정 레코드

다중 측정 레코드를 사용하면 테이블 행당 하나의 측정값을 저장하는 대신 단일 테이블 행에 여러 측정값을 저장할 수 있습니다. 따라서 다중 측정 레코드를 사용하면 최소한의 변경으로 관계형 데이터베이스에서 Amazon Timestream for Live Analytics로 기존 데이터를 마이그레이션할 수 있습니다.

단일 측정 레코드보다 단일 쓰기 요청에서 더 많은 데이터를 배치할 수도 있습니다. 이렇게 하면 데이터 쓰기 처리량과 성능이 증가하고 데이터 쓰기 비용도 줄어듭니다. 이는 쓰기 요청에서 더 많은 데이터를 배치화하면 Amazon Timestream for Live Analytics가 단일 쓰기 요청(해당하는 경우)에서 더 반복 가능한 데이터를 식별하고 반복된 데이터에 대해 한 번만 청구할 수 있기 때문입니다.

주제

- 다중 측정 레코드
- 과거 또는 미래에 존재하는 타임스탬프가 있는 데이터 쓰기

다중 측정 레코드

다중 측정 레코드를 사용하면 메모리 및 마그네틱 스토어에 시계열 데이터를 더 컴팩트한 형식으로 저장할 수 있으므로 데이터 스토리지 비용을 절감할 수 있습니다. 또한 컴팩트 데이터 스토리지는 데이터 검색을 위한 더 간단한 쿼리를 작성하는 데 도움이 되고, 쿼리 성능을 개선하고, 쿼리 비용을 절감합니다.

또한 다중 측정 레코드는 시계열 레코드에 둘 이상의 타임스탬프를 저장하기 위한 TIMESTAMP 데이터 유형도 지원합니다. TIMESTAMP 다중 측정 레코드의 속성은 향후 또는 과거의 타임스탬프를 지원합니다. 따라서 다중 측정 레코드는 성능, 비용 및 쿼리 단순성을 개선하고 서로 다른 유형의 상관관계가 있는 측정값을 저장할 수 있는 유연성을 제공합니다.

이점

다음은 다중 측정 레코드 사용의 이점입니다.

- 성능 및 비용 - 다중 측정 레코드를 사용하면 단일 쓰기 요청으로 여러 시계열 측정값을 작성할 수 있습니다. 이렇게 하면 쓰기 처리량이 증가하고 쓰기 비용도 줄어듭니다. 다중 측정 레코드를 사용하면 데이터를 더 컴팩트하게 저장할 수 있으므로 데이터 스토리지 비용을 절감할 수 있습니다. 다중 측정 레코드의 컴팩트한 데이터 스토리지로 인해 쿼리로 처리되는 데이터가 줄어듭니다. 이는 전체 쿼리 성능을 개선하고 쿼리 비용을 낮추는 데 도움이 되도록 설계되었습니다.

- 쿼리 단순성 - 다중 측정 레코드를 사용하면 타임스탬프가 동일한 여러 측정값을 읽기 위해 쿼리에 복잡한 일반 테이블 표현식(CTEs)을 작성할 필요가 없습니다. 이는 측정값이 단일 테이블 행에 열로 저장되기 때문입니다. 따라서 다중 측정 레코드를 사용하면 더 간단한 쿼리를 작성할 수 있습니다.
- 데이터 모델링 유연성 - TIMESTAMP 데이터 유형 및 다중 측정 레코드를 사용하여 Timestream for Live Analytics에 향후 타임스탬프를 작성할 수 있습니다. 다중 측정 레코드에는 레코드의 시간 필드 외에도 TIMESTAMP 데이터 유형의 속성이 여러 개 있을 수 있습니다. TIMESTAMP 다중 측정 레코드의 속성에는 향후 또는 과거의 타임스탬프가 있을 수 있으며 Timestream for Live Analytics가 다중 측정 레코드의 유형 값을 인덱싱하지 않는다는 점을 제외하고 시간 필드처럼 동작TIMESTAMP할 수 있습니다.

사용 사례

지정된 시간에 동일한 디바이스에서 둘 이상의 측정값을 생성하는 모든 시계열 애플리케이션에 다중 측정 레코드를 사용할 수 있습니다. 다음은 몇 가지 예제 애플리케이션입니다.

- 지정된 시간에 수백 개의 지표를 생성하는 비디오 스트리밍 플랫폼입니다.
- 혈중 산소 수치, 심박수, 맥박과 같은 측정치를 생성하는 의료 기기입니다.
- 지표, 온도 및 날씨 센서를 생성하는 오일 리그와 같은 산업용 장비입니다.
- 하나 이상의 마이크로서비스로 설계된 기타 애플리케이션.

예: 비디오 스트리밍 애플리케이션의 성능 및 상태 모니터링

200개의 EC2 인스턴스에서 실행 중인 비디오 스트리밍 애플리케이션을 고려해 보세요. Amazon Timestream for Live Analytics를 사용하여 애플리케이션에서 내보내는 지표를 저장하고 분석하면 애플리케이션의 성능과 상태를 이해하고, 이상을 빠르게 식별하고, 문제를 해결하고, 최적화 기회를 발견할 수 있습니다.

단일 측정 레코드와 다중 측정 레코드를 사용하여 이 시나리오를 모델링한 다음 두 접근 방식을 비교합니다. 각 접근 방식에 대해 다음과 같이 가정합니다.

- 각 EC2 인스턴스는 초당 4개의 측정값(video_startup_time, rebuffering_ratio, video_playback_failures 및 average_frame_rate)과 4개의 차원(device_id, device_type, os_version 및 리전)을 내보냅니다.
- 메모리 스토어에 6시간의 데이터를 저장하고 마그네틱 스토어에 6개월의 데이터를 저장하려고 합니다.
- 이상을 식별하기 위해 지난 몇 분 동안 비정상적인 활동을 식별하기 위해 1분마다 실행되는 쿼리 10개를 설정했습니다. 또한 애플리케이션을 효과적으로 모니터링할 수 있도록 지난 6시간의 데이터를

표시하는 위젯 8개가 포함된 대시보드를 구축했습니다. 이 대시보드는 지정된 시간에 5명의 사용자가 액세스하며 매시간 자동으로 새로 고쳐집니다.

단일 측정 레코드 사용

데이터 모델링 : 단일 측정 레코드를 사용하면 네 가지 측정(비디오 시작 시간, 재버퍼링 비율, 비디오 재생 실패 및 평균 프레임 속도) 각각에 대해 하나의 레코드를 생성합니다. 각 레코드에는 4차원(device_id, device_type, os_version 및 리전)과 타임스탬프가 있습니다.

쓰기: Amazon Timestream for Live Analytics에 데이터를 쓰면 레코드가 다음과 같이 구성됩니다.

```
public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();

    final Dimension device_id = new
 Dimension().withName("device_id").withValue("12345678");
    final Dimension device_type = new
 Dimension().withName("device_type").withValue("iPhone 11");
    final Dimension os_version = new
 Dimension().withName("os_version").withValue("14.8");
    final Dimension region = new Dimension().withName("region").withValue("us-east-1");

    dimensions.add(device_id);
    dimensions.add(device_type);
    dimensions.add(os_version);
    dimensions.add(region);

    Record videoStartupTime = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_startup_time")
        .withMeasureValue("200")
        .withMeasureValueType(MeasureValueType.BIGINT)
        .withTime(String.valueOf(time));
    Record rebufferingRatio = new Record()
        .withDimensions(dimensions)
        .withMeasureName("rebuffering_ratio")
        .withMeasureValue("0.5")
```

```
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));
    Record videoPlaybackFailures = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_playback_failures")
        .withMeasureValue("0")
        .withMeasureValueType(MeasureValueType.BIGINT)
        .withTime(String.valueOf(time));
    Record averageFrameRate = new Record()
        .withDimensions(dimensions)
        .withMeasureName("average_frame_rate")
        .withMeasureValue("0.5")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    records.add(videoStartupTime);
    records.add(rebufferingRatio);
    records.add(videoPlaybackFailures);
    records.add(averageFrameRate);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

단일 측정 레코드를 저장할 때 데이터는 다음과 같이 논리적으로 표시됩니다.

Time	device_id	device_type	os_version	region	measure_name	measure_value::bigint	measure_value::double
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	video_startup_time	200	
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	재버퍼링_비율		0.5
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	비디오_재생_실패	0	
2021-09-07 21:48:44.00	12345678	iPhone 11	14.8	us-east-1	평균 프레임 속도		0.85
2021-09-07 21:53:44.00	12345678	iPhone 11	14.8	us-east-1	video_startup_time	500	
2021-09-07 21:53:44.00	12345678	iPhone 11	14.8	us-east-1	재버퍼링_비율		1.5
2021-09-07 21:53:44.00	12345678	iPhone 11	14.8	us-east-1	비디오_재생_실패	10	

Time	device_id	device_type	os_version	region	measure_name	measure_value::bigint	measure_value::double
2021-09-07 21:53:44.00	12345678	iPhone 11	14.8	us-east-1	평균 프레임 속도		0.2

쿼리: 다음과 같이 지난 15분 동안 수신한 타임스탬프가 동일한 모든 데이터 포인트를 검색하는 쿼리를 작성할 수 있습니다.

```
with cte_video_startup_time as ( SELECT time, device_id, device_type, os_version,
 region, measure_value::bigint as video_startup_time FROM table where time >= ago(15m)
 and measure_name=”video_startup_time”),
cte_rebuffering_ratio as ( SELECT time, device_id, device_type, os_version, region,
 measure_value::double as rebuffering_ratio FROM table where time >= ago(15m) and
 measure_name=”rebuffering_ratio”),
cte_video_playback_failures as ( SELECT time, device_id, device_type, os_version,
 region, measure_value::bigint as video_playback_failures FROM table where time >=
 ago(15m) and measure_name=”video_playback_failures”),
cte_average_frame_rate as ( SELECT time, device_id, device_type, os_version, region,
 measure_value::double as average_frame_rate FROM table where time >= ago(15m) and
 measure_name=”average_frame_rate”)
SELECT a.time, a.device_id, a.os_version, a.region, a.video_startup_time,
 b.rebuffering_ratio, c.video_playback_failures, d.average_frame_rate FROM
 cte_video_startup_time a, cte_buffering_ratio b, cte_video_playback_failures c,
 cte_average_frame_rate d WHERE
a.time = b.time AND a.device_id = b.device_id AND a.os_version = b.os_version AND
 a.region=b.region AND
a.time = c.time AND a.device_id = c.device_id AND a.os_version = c.os_version AND
 a.region=c.region AND
a.time = d.time AND a.device_id = d.device_id AND a.os_version = d.os_version AND
 a.region=d.region
```

워크로드 비용 : 이 워크로드의 비용은 단일 측정 레코드와 함께 매월 373.23달러로 추정됩니다.

다중 측정 레코드 사용

데이터 모델링 : 다중 측정 레코드를 사용하면 4개의 모든 측정값(비디오 시작 시간, 재버퍼링 비율, 비디오 재생 실패 및 평균 프레임 속도), 4개의 모든 차원(device_id, device_type, os_version 및 리전) 및 타임스탬프가 포함된 하나의 레코드를 생성합니다.

쓰기: Amazon Timestream for Live Analytics에 데이터를 쓰면 레코드가 다음과 같이 구성됩니다.

```
public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();

    final Dimension device_id = new
 Dimension().withName("device_id").withValue("12345678");
    final Dimension device_type = new
 Dimension().withName("device_type").withValue("iPhone 11");
    final Dimension os_version = new
 Dimension().withName("os_version").withValue("14.8");
    final Dimension region = new Dimension().withName("region").withValue("us-east-1");

    dimensions.add(device_id);
    dimensions.add(device_type);
    dimensions.add(os_version);
    dimensions.add(region);

    Record videoMetrics = new Record()
        .withDimensions(dimensions)
        .withMeasureName("video_metrics")
        .withTime(String.valueOf(time));
        .withMeasureValueType(MeasureValueType.MULTI)
        .withMeasureValues(
          new MeasureValue()
         .withName("video_startup_time")
         .withValue("0")
         .withValueType(MeasureValueType.BIGINT),
         new MeasureValue()
  .withName("rebuffering_ratio")
         .withValue("0.5")
         .withType(MeasureValueType.DOUBLE),
```

```
            new MeasureValue()
           .withName("video_playback_failures")
           .withValue("0")
           .withValueType(MeasureValueType.BIGINT),
     new MeasureValue()
            .withName("average_frame_rate")
           .withValue("0.5")
           .withValueType(MeasureValueType.DOUBLE))

    records.add(videoMetrics);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

다중 측정 레코드를 저장할 때 데이터는 다음과 같이 논리적으로 표시됩니다.

Time	device_id	device_type	os_version	region	measure_name	video_startup_time	재버퍼링_비율	비디오_재생_실패	평균 프레임 속도
2021-09-7	1234567	iPhone 11	14.8	us-east-1	비디오_메트릭	200	0.5	0	0.85

Time	device_id	device_type	os_version	region	measure_name	video_startup_time	재버퍼링_비율	비디오_재생_실패	평균 프레임 속도
21:48:440									
2021-09-7 21:53:440	1234567	iPhone 11	14.8	us-east-1	비디오_메트릭	500	1.5	10	0.2

쿼리: 다음과 같이 지난 15분 동안 수신한 타임스탬프가 동일한 모든 데이터 포인트를 검색하는 쿼리를 작성할 수 있습니다.

```
SELECT time, device_id, device_type, os_version, region, video_startup_time,
 rebuffering_ratio, video_playback_failures, average_frame_rate FROM table where time
 >= ago(15m)
```

워크로드 비용 : 워크로드 비용은 다중 측정 레코드에서 127.43달러로 추정됩니다.

> **Note**
>
> 이 경우 다중 측정 레코드를 사용하면 전체 월별 예상 지출이 2.5배 줄어들고 데이터 쓰기 비용은 3.3배, 스토리지 비용은 3.3배, 쿼리 비용은 1.2배 줄어듭니다.

과거 또는 미래에 존재하는 타임스탬프가 있는 데이터 쓰기

Timestream for Live Analytics는 몇 가지 메커니즘을 통해 메모리 스토어 보존 기간 외부에 있는 타임스탬프를 사용하여 데이터를 쓸 수 있는 기능을 제공합니다.

- 마그네틱 스토어 쓰기 - 마그네틱 스토어 쓰기를 통해 지연 도착 데이터를 마그네틱 스토어에 직접 쓸 수 있습니다. 마그네틱 스토어 쓰기를 사용하려면 먼저 테이블에 마그네틱 스토어 쓰기를 활성화해야 합니다. 그런 다음 메모리 스토어에 데이터를 쓰는 데 사용된 것과 동일한 메커니즘을 사용하여 테이블에 데이터를 수집할 수 있습니다. Amazon Timestream for Live Analytics는 타임스탬프에 따라 마그네틱 스토어에 데이터를 자동으로 기록합니다.

> **Note**
>
> 지연 시간이 write-to-read 1초 미만 범위인 메모리 스토어에 데이터를 쓰는 것과 달리 마그네틱 스토어의 write-to-read 지연 시간은 최대 6시간일 수 있습니다.

- TIMESTAMP 측정값의 데이터 유형 - TIMESTAMP 데이터 유형을 사용하여 과거, 현재 또는 미래의 데이터를 저장할 수 있습니다. 다중 측정 레코드에는 레코드의 시간 필드 외에도 TIMESTAMP 데이터 유형의 속성이 여러 개 있을 수 있습니다. TIMESTAMP 다중 측정 레코드의 속성에는 향후 또는 과거의 타임스탬프가 있을 수 있으며 Timestream for Live Analytics가 다중 측정 레코드의 유형 값을 인덱싱하지 않는다는 점을 제외하고 시간 필드처럼 동작TIMESTAMP할 수 있습니다.

> **Note**
>
> TIMESTAMP 데이터 유형은 다중 측정 레코드에만 지원됩니다.

읽기에 대한 최종 일관성

Timestream for Live Analytics는 읽기에 대한 최종 일관성 의미 체계를 지원합니다. 즉, Timestream for Live Analytics에 데이터 배치를 작성한 직후에 데이터를 쿼리할 때 쿼리 결과에는 최근에 완료된 쓰기 작업의 결과가 반영되지 않을 수 있습니다. 잠시 후 이러한 쿼리 요청을 반복하면 결과가 최신 데이터를 반환해야 합니다.

로 쓰기 배치 WriteRecords API

Amazon Timestream for Live Analytics를 사용하면 단일 시계열의 데이터 포인트 및/또는 여러 시리즈의 데이터 포인트를 단일 쓰기 요청으로 작성할 수 있습니다. 단일 쓰기 작업에서 여러 데이터 포인트를 배치하는 것은 성능 및 비용 관점에서 유용합니다. 자세한 내용은 측정 및 요금 섹션쓰기의 섹션을 참조하세요.

> **Note**
>
> Timestream for Live Analytics에 대한 쓰기 요청은 애플리케이션의 데이터 수집 요구 사항에 맞게 Timestream for Live Analytics가 조정됨에 따라 제한될 수 있습니다. 애플리케이션에서 제한적인 예외가 발생하는 경우 Live Analytics용 Timestream이 애플리케이션의 필요에 맞게 자동으로 확장할 수 있도록 동일한(또는 더 높은) 처리량으로 데이터를 계속 전송해야 합니다.

배치 로드

용 Amazon Timestream에 대한 배치 로드로 Amazon S3에 저장된 CSV 파일을 Timestream에 배치로 수집할 LiveAnalytics수 있습니다. 이 새로운 기능을 사용하면 다른 도구에 의존하거나 사용자 지정 코드를 작성할 필요 LiveAnalytics 없이 에 대한 데이터를 Timestream에 저장할 수 있습니다. 쿼리 또는 분석에 즉시 필요하지 않은 데이터와 같이 유연한 대기 시간으로 데이터를 채우는 데 배치 로드를 사용할 수 있습니다.

AWS Management Console, 및 를 사용하여 배치 로드 작업을 생성할 수 AWS CLI있습니다 AWS SDKs. 자세한 내용은 콘솔에서 배치 로드 사용, 에서 배치 로드 사용 AWS CLI, 에서 배치 로드 사용 AWS SDKs 단원을 참조하세요.

배치 로드에 대한 자세한 내용은 섹션을 참조하세요에 대해 Timestream에서 배치 로드 사용 LiveAnalytics.

작업과 배치 로드 중에서 WriteRecords API 선택

작업을 사용하면 WriteRecords API 스트리밍 시계열 데이터가 시스템에서 생성 LiveAnalytics 되므로에 대해 스트리밍 시계열 데이터를 에 쓸 수 있습니다. 를 사용하면 단일 데이터 포인트 또는 더 작은 데이터 배치를 실시간으로 지속적으로 수집할 WriteRecords수 있습니다. 의 LiveAnalytics Timestream은 데이터베이스에 쓰기를 호출할 때 지정한 데이터 포인트의 차원 이름 및 데이터 유형을 기반으로 LiveAnalytics 테이블에 대한 Timestream의 열 이름 및 데이터 유형을 자동으로 감지하는 유연한 스키마를 제공합니다.

반대로 배치 로드는 정의한 데이터 모델을 LiveAnalytics사용하여 소스 파일(CSV 파일)에서 의 Timestream으로 배치화된 시계열 데이터를 강력하게 수집할 수 있도록 합니다. 소스 파일과 함께 배치 로드를 사용하는 몇 가지 예는 개념 증명을 LiveAnalytics 통해 의 Timestream 평가를 위해 시계열 데이터를 대량으로 가져오고, 일정 시간 동안 오프라인 상태였던 IoT 디바이스에서 시계열 데이터를 대량으로 가져오고, Amazon S3에서 의 Timestream으로 기록 시계열 데이터를 마이그레이션하는 것입니다 LiveAnalytics. 배치 로드에 대한 자세한 내용은 섹션을 참조하세요에 대해 Timestream에서 배치 로드 사용 LiveAnalytics.

두 솔루션 모두 안전하고 안정적이며 성능이 뛰어납니다.

다음과 같은 WriteRecords 경우에 사용합니다.

- 요청당 더 적은 양의 데이터 스트리밍(10MB 미만).
- 기존 테이블 채우기.
- 로그 스트림에서 데이터 수집.
- 실시간 분석 수행.

- 지연 시간이 더 짧아야 합니다.

다음과 같은 경우 배치 로드를 사용합니다.

- Amazon S3에서 시작된 더 많은 양의 데이터를 CSV 파일에 수집합니다. 제한에 대한 자세한 내용은 할당량 섹션을 참조하세요.
- 데이터 마이그레이션의 경우와 같이 새 테이블 채우기.
- 기록 데이터를 사용하여 데이터베이스 강화(새 테이블로 수집).
- 소스 데이터가 느리게 변경되거나 전혀 변경되지 않습니다.
- 특히 대량의 데이터를 로드하는 경우 리소스가 제공될 때까지 배치 로드 작업이 보류 중 상태일 수 있으므로 대기 시간이 유연합니다. 배치 로드는 쿼리 또는 분석에 쉽게 사용할 수 없어 더 명확하게 하기 위한 데이터에 적합합니다.

스토리지

Timestream for Live Analytics는 시계열 데이터를 저장하고 구성하여 쿼리 처리 시간을 최적화하고 스토리지 비용을 절감합니다. 데이터 스토리지 계층화를 제공하며 메모리 스토어와 마그네틱 스토어의 두 가지 스토리지 계층을 지원합니다. 메모리 스토어는 대용량 데이터 쓰기 및 빠른 point-in-time 쿼리에 최적화되어 있습니다. 마그네틱 스토어는 더 낮은 처리량의 지연 도착 데이터 쓰기, 장기 데이터 스토리지 및 빠른 분석 쿼리에 최적화되어 있습니다.

Timestream for Live Analytics는 단일 내에서 다양한 가용 영역에 메모리 및 마그네틱 스토어 데이터를 자동으로 복제하여 데이터의 내구성을 보장합니다 AWS 리전. 쓰기 요청이 완료되었음을 확인하기 전에 모든 데이터가 디스크에 기록됩니다.

Live Analytics용 Timestream을 사용하면 메모리 스토어에서 마그네틱 스토어로 데이터를 이동하도록 보존 정책을 구성할 수 있습니다. 데이터가 구성된 값에 도달하면 Live Analytics용 Timestream은 자동으로 데이터를 마그네틱 스토어로 이동합니다. 마그네틱 스토어에서 보존 값을 설정할 수도 있습니다. 마그네틱 스토어에서 데이터가 만료되면 영구적으로 삭제됩니다.

예를 들어 일주일 분량의 데이터를 저장하도록 메모리 스토어를 구성하고 1년 분량의 데이터를 저장하도록 자성 스토어를 구성하는 시나리오를 고려해 보겠습니다. 데이터 연식은 데이터 포인트와 연결된 타임스탬프를 사용하여 계산됩니다. 메모리 스토어의 데이터가 1주일이 되면 자동으로 마그네틱 스토어로 이동합니다. 그런 다음 마그네틱 저장소에서 1년 동안 유지됩니다. 데이터가 1년이 되면 실시간 분석을 위한 Timestream에서 삭제됩니다. 메모리 스토어와 마그네틱 스토어의 보존 값은 데이터가 Live Analytics용 Timestream에 저장되는 시간을 누적하여 정의합니다. 즉, 위 시나리오의 경우 데이터 도착 시점부터 데이터는 총 1년 1주일 동안 실시간 분석을 위한 Timestream에 저장됩니다.

> **Note**
>
> 메모리 또는 마그네틱 스토어의 보존 기간을 업그레이드하면 해당 시점 이후부터 보존 변경이 적용됩니다. 예를 들어 메모리 스토어의 보존 기간이 2시간으로 설정된 다음 테이블 보존 정책을 업데이트하여 24시간으로 변경된 경우 메모리 스토어는 24시간의 데이터를 보존할 수 있지만 이 변경 사항이 적용된 후 22시간이 지나면 24시간의 데이터로 채워집니다. Live Analytics의 Timestream은 마그네틱 스토어에서 데이터를 검색하여 메모리 스토어를 채우지 않습니다.

시계열 데이터의 보안을 보장하기 위해 Timestream for Live Analytics의 데이터는 항상 기본적으로 암호화됩니다. 이는 전송 중 및 유휴 데이터에 적용됩니다. 또한 Timestream for Live Analytics를 사용하면 고객 관리형 키를 사용하여 마그네틱 스토어의 데이터를 보호할 수 있습니다. 고객 관리형 키에 대한 자세한 내용은 섹션을 참조하세요AWS KMS keys.

쿼리

실시간 분석을 위한 Timestream을 사용하면 에 대한 지표, IoT 애플리케이션을 위한 DevOps센서 데이터, 장비 유지 관리를 위한 산업용 원격 측정 데이터 및 기타 여러 사용 사례를 쉽게 저장하고 분석할 수 있습니다. Timestream for Live Analytics의 특수 설계된 적응형 쿼리 엔진을 사용하면 단일 SQL 문을 사용하여 스토리지 계층 전반의 데이터에 액세스할 수 있습니다. 데이터 위치를 지정할 필요 없이 스토리지 계층 전반에 걸쳐 데이터에 투명하게 액세스하고 결합합니다. SQL 를 사용하여 Timestream for Live Analytics에서 데이터를 쿼리하여 하나 이상의 테이블에서 시계열 데이터를 검색할 수 있습니다. 데이터베이스 및 테이블에 대한 메타데이터 정보에 액세스할 수 있습니다. Timestream for Live Analytics는 시계열 분석을 위한 내장 함수SQL도 지원합니다. 자세한 내용은 쿼리 언어 참조 참조를 참조하세요.

Timestream for Live Analytics는 각 구성 요소가 다른 구성 요소와 독립적으로 확장될 수 있는 완전 분리 데이터 수집, 스토리지 및 쿼리 아키텍처를 갖도록 설계되었습니다(애플리케이션 요구 사항에 따라 사실상 무한한 규모를 제공할 수 있도록 허용). 즉, 애플리케이션이 하루에 수백 테라바이트의 데이터를 보내거나 소량 또는 대량의 데이터를 처리하는 수백만 개의 쿼리를 실행할 때 Timestream for Live Analytics는 “tip over”하지 않습니다. 시간이 지남에 따라 데이터가 증가함에 따라 Timestream for Live Analytics의 쿼리 지연 시간은 대부분 변경되지 않습니다. 이는 Timestream for Live Analytics 쿼리 아키텍처가 대량의 병렬 처리를 활용하여 더 큰 데이터 볼륨을 처리하고 애플리케이션의 쿼리 처리량 요구 사항에 맞게 자동으로 확장할 수 있기 때문입니다.

데이터 모델

Timestream은 쿼리에 대해 플랫 모델과 시계열 모델의 두 가지 데이터 모델을 지원합니다.

> **Note**
>
> Timestream의 데이터는 플랫 모델을 사용하여 저장되며 데이터 쿼리를 위한 기본 모델입니다. 시계열 모델은 쿼리 시간 개념이며 시계열 분석에 사용됩니다.

플랫 모델

플랫 모델은 쿼리에 대한 Timestream의 기본 데이터 모델입니다. 시계열 데이터를 표 형식으로 나타냅니다. 차원 이름, 시간, 측정값 이름 및 측정값은 열로 표시됩니다. 테이블의 각 행은 시계열 내 특정 시점의 측정치에 해당하는 원자 데이터 포인트입니다. Timestream 데이터베이스, 테이블 및 열에는 몇 가지 이름 지정 제약 조건이 있습니다. 이에 대한 설명은 에 나와 있습니다서비스 한도.

아래 표는 데이터가 단일 측정 레코드로 전송될 때 Timestream이 EC2 인스턴스의 CPU 사용률, 메모리 사용률 및 네트워크 활동을 나타내는 데이터를 저장하는 방법에 대한 예시를 보여줍니다. 이 경우 차원은 리전, 가용 영역, 가상 프라이빗 클라우드 및 EC2 인스턴스 IDs 인스턴스입니다. 측정값은 EC2 인스턴스의 CPU 사용률, 메모리 사용률 및 수신 네트워크 데이터입니다. 열 리전, az, vpc 및 instance_id에는 차원 값이 포함됩니다. 열 시간에는 각 레코드의 타임스탬프가 포함됩니다. measure_name 열에는 cpu-utilization, memory_utilization 및 network_bytes_in으로 표시되는 측정값의 이름이 포함되어 있습니다. measure_value::double 열에는 두 배로 방출된 측정값(예: CPU 사용률 및 메모리 사용률)이 포함됩니다. 열 measure_value::bigint에는 수신 네트워크 데이터와 같은 정수로 방출되는 측정값이 포함됩니다.

Time	region	az	vpc	instance-id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	35.0	null

Time	region	az	vpc	instance-id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	38.2	null
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	45.3	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	54.9	null
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	42.6	null
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	memory_utilization	33.3	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes, 네트워크_바이트	34,400	null
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes, 네트워크_바이트	1,500	null

Time	region	az	vpc	instance-id	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	network_bytes, 네트워크_바이트	6,000	null

아래 표는 데이터가 다중 측정 레코드로 전송될 때 Timestream이 EC2 인스턴스의 CPU 사용률, 메모리 사용률 및 네트워크 활동을 나타내는 데이터를 저장하는 방법에 대한 예시를 보여줍니다.

Time	region	az	vpc	instance-id	measure_name	cpu_utilization	memory_utilization	network_bytes, 네트워크_바이트
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	지표	35.0	54.9	34,400
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	지표	38.2	42.6	1,500
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcde0	지표	45.3	33.3	6,600

시계열 모델

시계열 모델은 시계열 분석에 사용되는 쿼리 시간 구성입니다. 이는 데이터를 (시간, 측정값) 페어의 순서 시퀀스로 나타냅니다. Timestream은 데이터 격차를 메울 수 있도록 보간과 같은 시계열 함수를 지

원합니다. 이러한 함수를 사용하려면 create_time_series와 같은 함수를 사용하여 데이터를 시계열 모델로 변환해야 합니다. 자세한 내용은 섹션을 참조쿼리 언어 참조하세요.

EC2 인스턴스의 이전 예제를 사용하면 다음은 시간 단위로 표현되는 CPU 사용률 데이터입니다.

region	az	vpc	instance-id	cpu_utilization
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	[{time: 2019-12-04 19:00:00.000000000, 값: 35}, {time: 2019-12-04 19:00:01.000000000, 값: 38.2}, {time: 2019-12-04 19:00:02.000000000, 값: 45.3}]

예약된 쿼리

Amazon Timestream for Live Analytics의 예약된 쿼리 기능은 운영 대시보드, 비즈니스 보고서, 임시 분석 및 기타 애플리케이션에 일반적으로 사용되는 집계, 롤업 및 기타 형태의 사전 처리된 데이터를 계산하고 저장하기 위한 완전 관리형, 서버리스 및 확장 가능한 솔루션입니다. 예약된 쿼리를 사용하면 실시간 분석이 더 효과적이고 비용 효율적이므로 데이터에서 추가 인사이트를 도출하고 더 나은 비즈니스 결정을 내릴 수 있습니다.

예약된 쿼리에 대한 자세한 내용은 섹션을 참조하세요에 대해 Timestream에서 예약된 쿼리 사용 LiveAnalytics.

Timestream Compute Unit(TCU)

Amazon Timestream for Live Analytics는 Timestream 컴퓨팅 유닛()에서 쿼리 요구 사항에 할당된 컴퓨팅 용량을 측정합니다TCU. 하나는 4 vCPUs GB와 16GB의 메모리로 TCU 구성됩니다. Timestream for Live Analytics에서 쿼리를 실행하면 서비스는 쿼리의 복잡성과 처리 중인 데이터의 양에 따라 TCUs 온디맨드 방식으로 할당합니다. 쿼리가 소비TCUs하는 수에 따라 관련 비용이 결정됩니다.

> **Note**
>
> 2024년 4월 29일 이후 서비스에 온보딩 AWS 계정 된 모든 항목은 기본적으로 를 TCUs 사용하여 쿼리 요금을 결정합니다.

이 주제의 내용

MaxQuery TCU

이 설정은 서비스가 언제든지 쿼리를 처리하는 데 사용할 최대 컴퓨팅 단위 수를 지정합니다. 쿼리를 실행하려면 최소 용량을 4로 설정해야 합니다TCUs. 의 최대 개수를 4, 예를 들어 4, 8, 16, 32 등의 TCUs 배수로 설정할 수 있습니다. 워크로드에 사용하는 컴퓨팅 리소스에 대해서만 요금이 부과됩니다. 예를 들어 최대값을 128TCUs로 설정했지만 일관되게 8개만 사용하는 경우 TCUs 8 을 사용한 기간 동안에만 요금이 부과됩니다TCUs. 계정`MaxQueryTCU`의 기본값은 200으로 설정됩니다. 또는 에서 AWS Management Console 또는 UpdateAccountSettings API 작업을 사용하여 `4MaxQueryTCU`에서 1000으로 AWS SDK 조정할 수 있습니다 AWS CLI.

계정에 `MaxQueryTCU` 대해 를 설정하는 것이 좋습니다. 최대 TCU 한도를 설정하면 서비스에서 쿼리 워크로드에 사용할 수 있는 컴퓨팅 단위 수를 제한하여 비용을 제어할 수 있습니다. 이를 통해 쿼리 지출을 더 잘 예측하고 관리할 수 있습니다.

TCU에 대한 요금 청구

각 TCU 는 초당 세분성 및 최소 30초 동안 시간 단위로 청구됩니다. 이러한 컴퓨팅 단위의 사용 단위는 TCU시간입니다.

쿼리를 실행하면 쿼리 실행 시간 동안 TCUs 사용된 에 대한 요금이 TCU-시간 단위로 청구됩니다. 예:

- 워크로드는 3시간 TCUs 동안 20을 사용합니다. 60TCU시간(20 TCUs x 3시간)의 요금이 청구됩니다.
- 워크로드는 30분 TCUs 동안 10을 사용한 다음 다음 다음 30분 TCUs 동안 20을 사용합니다. 15TCU시간(10 TCUs x 0.5시간 + 20 TCUs x 0.5시간)의 요금이 청구됩니다.

TCU시간당 요금은 에 따라 다릅니다 AWS 리전. 자세한 내용은 Amazon Timestream 요금을 참조하세요. 워크로드가 증가함에 따라 서비스는 일관된 성능을 유지하기 위해 컴퓨팅 용량을 지정된 최대 TCU 한도(MaxQueryTCU)까지 자동으로 확장합니다. MaxQueryTCU 설정은 서비스가 확장할 수 있는 컴퓨팅 용량의 상한 역할을 합니다. 이 설정을 사용하면 컴퓨팅 리소스 수와 그로 인한 비용을 제어할 수 있습니다.

구성 TCU

서비스를 온보딩할 때 각 서비스의 기본 MaxQueryTCU 제한 AWS 계정 은 200입니다. 또는 를 사용하는 AWS Management Console 또는 UpdateAccountSettings API 작업을 사용하여 언제든지 필요에 따라 이 제한을 업데이트할 수 있습니다 AWS SDK AWS CLI.

구성할 값이 확실하지 않은 경우 계정의 QueryTCU 지표를 모니터링하세요. 이 지표는 AWS Management Console 및 Amazon 에서 사용할 수 있습니다 CloudWatch. 이 지표는 1분 단위로 TCUs 사용되는 최대 수에 대한 인사이트를 제공합니다. 과거 데이터와 미래 성장 추정에 따라 사용량의 급증에 MaxQueryTCU 맞게 를 설정합니다. 사용량이 가장 많은 곳보다 4~16TCUs개 이상의 헤드룸을 사용하는 것이 좋습니다. 예를 들어 지난 30일 QueryTCU 동안의 피크가 128인 경우 132MaxQueryTCU~144로 설정하는 것이 좋습니다.

필요한 컴퓨팅 단위 추정

컴퓨팅 유닛은 쿼리를 동시에 처리할 수 있습니다. 필요한 컴퓨팅 단위 수를 확인하려면 다음 표의 일반 지침을 고려하세요.

동시 쿼리 수	TCUs
7	4
14	8
21	12

Note

- 다음은 일반적인 지침이며 필요한 실제 컴퓨팅 단위 수는 다음과 같은 여러 요인에 따라 달라집니다.
  - 쿼리의 효과적인 동시성입니다.
  - 쿼리 패턴.
  - 스캔한 파티션 수입니다.
  - 기타 워크로드별 특성.
- 이 지침은 지난 몇 분에서 한 시간 동안 데이터를 스캔하고 Timestream 쿼리 모범 사례 및 데이터 모델링 지침 을 준수하는 쿼리와 관련이 있습니다. ???
- 애플리케이션 성능과 `QueryTCU` 지표를 모니터링하여 필요에 따라 컴퓨팅 단위를 조정합니다.

언제 늘려야 하는가 MaxQueryTCU

다음 시나리오`MaxQueryTCU`에서는 를 늘리는 것을 고려해야 합니다.

- 최대 쿼리 소비량이 현재 구성된 최대 쿼리 에 가 가까워지거나 도달하고 있습니다TCU. 최대 쿼리를 최대 사용량보다 TCU 4~16 이상 TCUs높게 설정하는 것이 좋습니다.
- 쿼리에서 메시지가 MaxQueryTCU 초과된 4xx 오류를 반환합니다. 워크로드가 계획대로 증가할 것으로 예상되는 경우 를 다시 방문하여 구성된 최대 쿼리를 TCU 그에 따라 조정합니다.

감소 시기 MaxQueryTCU

다음 시나리오`MaxQueryTCU`에서는 를 줄이는 것을 고려해야 합니다.

- 워크로드는 예측 가능하고 안정적인 사용 패턴을 가지며 컴퓨팅 사용 요구 사항을 잘 이해하고 있습니다. 최대 쿼리를 최대 사용량TCU보다 4~16 이내TCU로 낮추면 의도하지 않은 사용 및 비용을 방지하는 데 도움이 될 수 있습니다. UpdateAccountSettings API 작업을 사용하여 값을 수정할 수 있습니다.
- 애플리케이션 또는 사용자 동작 패턴의 변경으로 인해 시간이 지남에 따라 워크로드의 최대 사용량이 감소했습니다. 최대값을 낮추면 의도하지 않은 비용을 줄이는 데 도움이 될 TCU 수 있습니다.

> **Note**
>
> 현재 사용량에 따라 최대 TCU 한도 변경을 줄이는 데 최대 24시간이 걸릴 수 있습니다. TCUs 쿼리가 실제로 사용하는 에 대해서만 요금이 청구됩니다. 최대 쿼리 TCU 한도가 더 높아도 워크로드에서 TCUs 사용하지 않는 한 비용은 영향을 받지 않습니다.

CloudWatch 지표를 사용하여 사용량 모니터링

TCU 사용량을 모니터링하기 위해 Timestream for Live Analytics는 다음 CloudWatch 지표를 제공합니다`QueryTCU`. 이 지표는 1분에 사용되는 컴퓨팅 단위 수를 지정하고 1분마다 방출됩니다. 1분에 TCUs 사용되는 최대값과 최소값을 모니터링하도록 선택할 수 있습니다. 이 지표에 경보를 설정하여 쿼리 비용을 실시간으로 추적할 수도 있습니다.

컴퓨팅 단위 사용량의 차이 이해

쿼리에 필요한 컴퓨팅 리소스 수는 여러 파라미터에 따라 증가하거나 감소할 수 있습니다. 예를 들어, 실시간 및 분석 쿼리를 사용하는 데이터 볼륨, 데이터 수집 패턴, 쿼리 지연 시간, 쿼리 모양, 쿼리 효율성 및 쿼리 조합이 있습니다. 이러한 파라미터로 인해 워크로드에 더 높거나 낮은 TCU 단위가 필요할 수 있습니다. 이러한 파라미터가 변경되지 않는 정상 상태에서는 워크로드에 필요한 컴퓨팅 단위 수가 감소하는 것을 관찰할 수 있습니다. 따라서 월별 비용이 절감될 수 있습니다.

또한 워크로드 또는 데이터에서 이러한 파라미터가 변경되면 필요한 컴퓨팅 단위 수가 증가할 수 있습니다. Timestream이 쿼리를 수신할 때 쿼리가 액세스하는 데이터 파티션에 따라 Timestream은 쿼리를 효과적으로 처리할 컴퓨팅 리소스 수를 결정합니다.

수집 및 쿼리 액세스 패턴에 따라 정기적으로 Timestream은 데이터 레이아웃을 최적화합니다. Timestream은 액세스가 적은 파티션을 단일 파티션으로 묶거나 성능을 위해 핫 파티션을 여러 파티션으로 분할하여 최적화를 수행합니다. 따라서 동일한 쿼리에서 사용하는 컴퓨팅 용량은 시점마다 약간 다를 수 있습니다.

> **쿼리에 TCU 요금을 사용하도록 옵트인**
>
> 기존 사용자는 일회성 옵트인을 수행하여 쿼리당 측정된 최소 바이트 수를 TCUs 더 잘 관리하고 제거하는 데 사용할 수 있습니다. 또는 에서 AWS Management Console 또는 UpdateAccountSettings API 작업을 사용하여 옵트인할 AWS SDK 수 있습니다 AWS CLI. API 작업에서 `QueryPricingModel` 파라미터를 로 설정합니다`COMPUTE_UNITS`.
> 컴퓨팅 기반 요금 모델을 선택하는 것은 되돌릴 수 없는 변화입니다.

에 대한 Timestream 액세스 LiveAnalytics

콘솔 CLI 또는 를 LiveAnalytics 사용하여 Timestream에 액세스할 수 있습니다API. 의 Timestream 액세스에 대한 자세한 내용은 다음을 LiveAnalytics참조하세요.

주제

- 에 가입 AWS 계정
- 관리자 액세스 권한이 있는 사용자 생성
- LiveAnalytics 액세스를 위한 Timestream 제공
- 프로그래밍 방식 액세스 권한 부여

에 가입 AWS 계정

가 없는 경우 다음 단계를 AWS 계정완료하여 를 생성합니다.

에 가입하려면 AWS 계정

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

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

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

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/로 이동하여 내 계정 을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 를 AWS 계정보호하고, 를 AWS 계정 루트 사용자활성화하고 AWS IAM Identity Center, 관리 사용자를 생성합니다.

보안 AWS 계정 루트 사용자

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

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 루트 사용자로 로그인을 참조하십시오.
2. 루트 사용자에 대해 다중 인증(MFA)을 켭니다.

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

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 자격 증명 소스 IAM Identity Center 디렉터리 로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 기본값으로 사용자 액세스 구성을 IAM Identity Center 디렉터리 참조하세요.

관리 액세스 권한이 있는 사용자로 로그인

- IAM Identity Center 사용자로 로그인하려면 IAM Identity Center 사용자를 생성할 때 이메일 주소로 전송URL된 로그인을 사용합니다.

  IAM Identity Center 사용자를 사용하여 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 AWS 액세스 포털에 로그인을 참조하세요.

추가 사용자에게 액세스 권한 할당

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

   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.

LiveAnalytics 액세스를 위한 Timestream 제공

에 대한 Timestream에 액세스하는 데 필요한 권한이 관리자에게 이미 부여 LiveAnalytics 되었습니다. 다른 사용자의 경우 다음 정책을 사용하여 Timestream에 LiveAnalytics 액세스 권한을 부여해야 합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "timestream:*",
        "kms:DescribeKey",
        "kms:CreateGrant",
        "kms:Decrypt",
        "dbqms:CreateFavoriteQuery",
        "dbqms:DescribeFavoriteQueries",
        "dbqms:UpdateFavoriteQuery",
        "dbqms:DeleteFavoriteQueries",
        "dbqms:GetQueryString",
        "dbqms:CreateQueryHistory",
        "dbqms:UpdateQueryHistory",
        "dbqms:DeleteQueryHistory",
        "dbqms:DescribeQueryHistory",
        "s3:ListAllMyBuckets"
      ],
      "Resource": "*"
    }
  ]
}
```

> **Note**
>
> 에 대한 자세한 내용은 데이터베이스 쿼리 메타데이터 서비스의 작업, 리소스 및 조건 키를 dbqms참조하세요. https://docs.aws.amazon.com/service-authorization/latest/reference/list_databasequerymetadataservice.html 자세한 내용은 AWS Key Management Service 의 작업, 리소스 및 조건 키를 kms 참조하세요.

프로그래밍 방식 액세스 권한 부여

사용자는 AWS 외부에서 와 상호 작용하려는 경우 프로그래밍 방식으로 액세스해야 합니다 AWS Management Console. 프로그래밍 방식 액세스를 부여하는 방법은 에 액세스하는 사용자 유형에 따라 다릅니다 AWS.

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAMID 센터에서 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI AWS SDKs, 또는 에 대한 프로그래밍 요청에 서명합니다 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 의 경우 AWS Command Line Interface 사용 설명서의 AWS CLI 를 사용하도록 구성을 AWS IAM Identity Center AWS CLI참조하세요. • AWS SDKs, 도구 및 의 경우 AWS SDKs 및 도구 참조 가이드의 IAM Identity Center 인증을 AWS APIs참조하세요.
IAM	임시 자격 증명을 사용하여 AWS CLI AWS SDKs, 또는 에 대한 프로그래밍 요청에 서명합니다 AWS APIs.	IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용의 지침을 따릅니다.
IAM	(권장되지 않음) 장기 보안 인증 정보를 사용하여 AWS CLI AWS SDKs, 또는 에 대한 프로그래밍 요청에 서명합니다 AWS APIs.	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. • 의 경우 AWS Command Line Interface 사용 설명서의 IAM 사용자 자격 증명을 사용하여 인증을 AWS CLI참조하세요.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		• 및 도구에 대한 AWS SDKs 자세한 내용은 AWS SDKs 및 도구 참조 가이드의 장기 자격 증명을 사용하여 인증을 참조하세요. • 의 경우 IAM 사용 설명서의 IAM 사용자에 대한 액세스 키 관리를 AWS APIs참조하세요.

콘솔 사용

Timestream Live Analytics용 AWS 관리 콘솔을 사용하여 데이터베이스 및 테이블을 생성, 편집, 삭제, 설명 및 나열할 수 있습니다. 콘솔을 사용하여 쿼리를 실행할 수도 있습니다.

주제

- 튜토리얼
- 데이터베이스 생성
- 테이블 생성
- 쿼리 실행
- 예약된 쿼리 생성
- 예약된 쿼리 삭제
- 테이블 삭제
- 데이터베이스 삭제
- 테이블 편집
- 데이터베이스 편집

튜토리얼

이 자습서에서는 샘플 데이터 세트로 채워진 데이터베이스를 생성하고 샘플 쿼리를 실행하는 방법을 보여줍니다. 이 자습서에서 사용되는 샘플 데이터 세트는 IoT 및 DevOps 시나리오에서 자주 볼 수 있

습니다. IoT 데이터 세트에는 트럭의 속도, 위치 및 부하와 같은 시계열 데이터가 포함되어 있어 플릿 관리를 간소화하고 최적화 기회를 식별할 수 있습니다. 데이터 DevOps 세트에는 애플리케이션 성능과 가용성을 개선하기 위해 , CPU네트워크 및 메모리 사용률과 같은 EC2 인스턴스 지표가 포함되어 있습니다. 다음은 이 섹션에 설명된 지침에 대한 비디오 자습서입니다.

다음 단계에 따라 샘플 데이터 세트로 채워진 데이터베이스를 생성하고 AWS 콘솔을 사용하여 샘플 쿼리를 실행합니다.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 데이터베이스 생성 을 클릭합니다.
4. 데이터베이스 생성 페이지에서 다음을 입력합니다.
   - 구성 선택 - 샘플 데이터베이스 를 선택합니다.
   - 이름 - 선택한 데이터베이스 이름을 입력합니다.
   - 샘플 데이터 세트 선택 - IoT 및 를 선택합니다DevOps.
   - 데이터베이스 생성을 클릭하여 IoT와 샘플 데이터로 DevOps 채워진 두 개의 테이블이 포함된 데이터베이스를 생성합니다.
5. 탐색 창에서 쿼리 편집기를 선택합니다.
6. 상단 메뉴에서 샘플 쿼리를 선택합니다.
7. 샘플 쿼리 중 하나를 클릭합니다. 그러면 샘플 쿼리로 채워진 편집기가 있는 쿼리 편집기로 돌아갑니다.
8. 실행을 클릭하여 쿼리를 실행하고 쿼리 결과를 확인합니다.

데이터베이스 생성

AWS 콘솔을 사용하여 데이터베이스를 생성하는 방법은 다음과 같습니다.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 데이터베이스 생성 을 클릭합니다.
4. 데이터베이스 생성 페이지에서 다음을 입력합니다.
   - 구성 선택 - 표준 데이터베이스 를 선택합니다.
   - 이름 - 선택한 데이터베이스 이름을 입력합니다.

- 암호화 - KMS 키를 선택하거나 기본 옵션을 사용합니다. 여기서 Timestream Live Analytics는 키가 아직 없는 경우 계정에 KMS 키를 생성합니다.

5. 데이터베이스 생성을 클릭하여 데이터베이스를 생성합니다.

테이블 생성

AWS 콘솔을 사용하여 테이블을 생성하는 방법은 다음과 같습니다.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 테이블을 선택합니다.
3. 테이블 생성을 클릭합니다.
4. 테이블 생성 페이지에서 다음을 입력합니다.

   - 데이터베이스 이름 - 에서 생성된 데이터베이스의 이름을 선택합니다 데이터베이스 생성.
   - 테이블 이름 - 선택한 테이블 이름을 입력합니다.
   - 메모리 스토어 보존 - 메모리 스토어에 데이터를 보존할 기간을 지정합니다. 메모리 스토어는 지연 도착 데이터(현재 시간보다 타임스탬프가 빠른 데이터)를 포함하여 들어오는 데이터를 처리하고 빠른 point-in-time 쿼리에 최적화되어 있습니다.
   - 마그네틱 스토어 보존 - 마그네틱 스토어에 데이터를 보존할 기간을 지정합니다. 마그네틱 스토어는 장기 스토리지용이며 빠른 분석 쿼리에 최적화되어 있습니다.

5. 테이블 생성을 클릭합니다.

쿼리 실행

AWS 콘솔을 사용하여 쿼리를 실행하려면 다음 단계를 따르세요.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 쿼리 편집기를 선택합니다.
3. 왼쪽 창에서 에서 생성된 데이터베이스를 선택합니다 데이터베이스 생성.
4. 왼쪽 창에서 에서 생성된 데이터베이스를 선택합니다테이블 생성.
5. 쿼리 편집기에서 쿼리를 실행할 수 있습니다. 테이블의 최신 10개 행을 보려면 다음을 실행합니다.

```
SELECT * FROM <database_name>.<table_name> ORDER BY time DESC LIMIT 10
```

6. (선택 사항) 쿼리의 효율성에 대한 인사이트를 얻으려면 인사이트 활성화를 켭니다.

예약된 쿼리 생성

AWS 콘솔을 사용하여 예약된 쿼리를 생성하는 방법은 다음과 같습니다.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 예약된 쿼리 를 선택합니다.
3. 예약된 쿼리 생성 을 클릭합니다.
4. 쿼리 이름 및 대상 테이블 섹션에 다음을 입력합니다.
   - 이름 - 쿼리 이름을 입력합니다.
   - 데이터베이스 이름 - 에서 생성된 데이터베이스의 이름을 선택합니다 데이터베이스 생성.
   - 테이블 이름 - 에서 생성된 테이블의 이름을 선택합니다테이블 생성.
5. 쿼리 문 섹션에서 유효한 쿼리 문을 입력합니다. 그런 다음 쿼리 검증을 클릭합니다.
6. 대상 테이블 모델 에서 정의되지 않은 속성에 대한 모델을 정의합니다. Visual Builder 또는 를 사용할 수 있습니다JSON.
7. 일정 실행 섹션에서 고정 속도 또는 Chron 표현식을 선택합니다. 시간 표현식의 경우 일정 표현식에 대한 자세한 내용은 예약된 쿼리에 대한 일정 표현식을 참조하세요.
8. SNS 주제 섹션에서 알림에 사용할 SNS 주제를 입력합니다.
9. 오류 로그 보고서 섹션에 오류를 보고하는 데 사용할 S3 위치를 입력합니다.

   암호화 키 유형(Encryption key type)을 선택합니다.
10. AWS KMS 키 의 보안 설정 섹션에서 키 유형을 AWS KMS 선택합니다.

    Timestream에서 예약된 쿼리를 실행하는 데 LiveAnalytics 사용할 IAM 역할을 입력합니다. 역할에 필요한 권한 및 신뢰 관계에 대한 자세한 내용은 IAM 예약된 쿼리의 정책 예제를 참조하세요.
11. 예약된 쿼리 생성 을 클릭합니다.

예약된 쿼리 삭제

콘솔을 사용하여 예약된 쿼리를 삭제하거나 비활성화하려면 다음 단계를 따르세요 AWS .

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 예약된 쿼리를 선택합니다.
3. 에서 생성된 예약된 쿼리를 선택합니다예약된 쿼리 생성.

4. 작업을 선택합니다.
5. 비활성화 또는 삭제를 선택합니다.
6. 삭제를 선택한 경우 작업을 확인하고 삭제를 선택합니다.

테이블 삭제

AWS 콘솔을 사용하여 데이터베이스를 삭제하려면 다음 단계를 따르세요.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 테이블을 선택합니다.
3. 에서 생성한 테이블을 선택합니다테이블 생성.
4. 삭제를 클릭합니다.
5. 확인 상자에 삭제를 입력합니다.

데이터베이스 삭제

AWS 콘솔을 사용하여 데이터베이스를 삭제하려면 다음 단계를 따르세요.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 데이터베이스 생성 에서 생성한 데이터베이스를 선택합니다.
4. 삭제를 클릭합니다.
5. 확인 상자에 삭제를 입력합니다.

테이블 편집

AWS 콘솔을 사용하여 테이블을 편집하려면 다음 단계를 따르세요.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 테이블을 선택합니다.
3. 에서 생성한 테이블을 선택합니다테이블 생성.
4. 편집을 클릭합니다.
5. 테이블 세부 정보를 편집하고 저장합니다.

- 메모리 스토어 보존 - 메모리 스토어에 데이터를 보존할 기간을 지정합니다. 메모리 스토어는 지연 도착 데이터(현재 시간보다 타임스탬프가 빠른 데이터)를 포함하여 들어오는 데이터를 처리하고 빠른 point-in-time 쿼리에 최적화되어 있습니다.
- 마그네틱 스토어 보존 - 마그네틱 스토어에 데이터를 보존할 기간을 지정합니다. 마그네틱 스토어는 장기 스토리지용이며 빠른 분석 쿼리에 최적화되어 있습니다.

데이터베이스 편집

AWS 콘솔을 사용하여 데이터베이스를 편집하려면 다음 단계를 따르세요.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 데이터베이스 생성 에서 생성한 데이터베이스를 선택합니다.
4. 편집을 클릭합니다.
5. 데이터베이스 세부 정보를 편집하고 저장합니다.

를 LiveAnalytics 사용하기 위한 Amazon Timestream 액세스 AWS CLI

AWS Command Line Interface (AWS CLI)를 사용하여 명령줄에서 여러 AWS 서비스를 제어하고 스크립트를 통해 자동화할 수 있습니다. 임시 작업에 AWS CLI 를 사용할 수 있습니다. 유틸리티 스크립트 내에 LiveAnalytics 작업에 Amazon Timestream을 포함할 수도 있습니다.

에 대해 Timestream과 AWS CLI 함께 를 사용하려면 먼저 프로그래밍 액세스를 설정해야 LiveAnalytics합니다. 자세한 내용은 프로그래밍 방식 액세스 권한 부여 단원을 참조하십시오.

의 Timestream for LiveAnalytics QueryAPI에 사용할 수 있는 모든 명령의 전체 목록은 AWS CLI 명령 참조를 AWS CLI참조하세요.

API 에서 Timestream for LiveAnalytics Write에 사용할 수 있는 모든 명령의 전체 목록은 AWS CLI 명령 참조를 AWS CLI참조하세요.

주제

- AWS CLI다운로드 및 구성
- 에 Timestream과 AWS CLI 함께 사용 LiveAnalytics

AWS CLI다운로드 및 구성

는 Windows, macOS 또는 Linux에서 AWS CLI 실행됩니다. 다운로드, 설치 및 구성하려면 다음 단계를 따르세요.

1. http://aws.amazon.com/cli AWS CLI 를 다운로드합니다.
2. AWS Command Line Interface 사용 설명서의 설치 AWS CLI 및 구성 AWS CLI 지침을 따릅니다.

에 Timestream과 AWS CLI 함께 사용 LiveAnalytics

명령줄 형식은 LiveAnalytics 작업 이름에 대한 Amazon Timestream과 해당 작업에 대한 파라미터로 구성됩니다. 는 외에도 파라미터 값에 대한 간략 구문을 AWS CLI 지원합니다JSON.

help 에 대한 Timestream에서 사용 가능한 모든 명령을 나열하는 데 사용합니다 LiveAnalytics. 예:

```
aws timestream-write help
```

```
aws timestream-query help
```

help를 사용하여 특정 명령을 설명하고 그 사용법에 대해 자세히 알아볼 수도 있습니다.

```
aws timestream-write create-database help
```

예를 들어 데이터베이스를 생성하려면:

```
aws timestream-write create-database --database-name myFirstDatabase
```

마그네틱 스토어 쓰기가 활성화된 테이블을 생성하려면:

```
aws timestream-write create-table \
--database-name metricsdb \
--table-name metrics \
--magnetic-store-write-properties "{\"EnableMagneticStoreWrites\": true}"
```

단일 측정 레코드를 사용하여 데이터를 쓰려면:

```
aws timestream-write write-records \
```

```
--database-name metricsdb \
--table-name metrics \
--common-attributes "{\"Dimensions\":[{\"Name\":\"asset_id\", \"Value\":\"100\"}],
 \"Time\":\"1631051324000\",\"TimeUnit\":\"MILLISECONDS\"}" \
--records "[{\"MeasureName\":\"temperature\", \"MeasureValueType\":\"DOUBLE\",
\"MeasureValue\":\"30\"},{\"MeasureName\":\"windspeed\", \"MeasureValueType\":\"DOUBLE
\",\"MeasureValue\":\"7\"},{\"MeasureName\":\"humidity\", \"MeasureValueType\":\"DOUBLE
\",\"MeasureValue\":\"15\"},{\"MeasureName\":\"brightness\", \"MeasureValueType\":
\"DOUBLE\",\"MeasureValue\":\"17\"}]"
```

다중 측정 레코드를 사용하여 데이터를 쓰려면:

```
# wide model helper method to create Multi-measure records
function ingest_multi_measure_records {
  epoch=`date +%s`
  epoch+=$i

  # multi-measure records
  aws timestream-write write-records \
  --database-name $src_db_wide \
  --table-name $src_tbl_wide \
  --common-attributes "{\"Dimensions\":[{\"Name\":\"device_id\", \
              \"Value\":\"12345678\"},\
            {\"Name\":\"device_type\", \"Value\":\"iPhone\"}, \
            {\"Name\":\"os_version\", \"Value\":\"14.8\"}, \
            {\"Name\":\"region\", \"Value\":\"us-east-1\"} ], \
            \"Time\":\"$epoch\",\"TimeUnit\":\"MILLISECONDS\"}" \
--records "[{\"MeasureName\":\"video_metrics\", \"MeasureValueType\":\"MULTI\", \
  \"MeasureValues\": \
  [{\"Name\":\"video_startup_time\",\"Value\":\"0\",\"Type\":\"BIGINT\"}, \
  {\"Name\":\"rebuffering_ratio\",\"Value\":\"0.5\",\"Type\":\"DOUBLE\"}, \
  {\"Name\":\"video_playback_failures\",\"Value\":\"0\",\"Type\":\"BIGINT\"}, \
  {\"Name\":\"average_frame_rate\",\"Value\":\"0.5\",\"Type\":\"DOUBLE\"}]}]" \
--endpoint-url $ingest_endpoint \
  --region  $region
}

# create 5 records
for i in {100..105};
  do ingest_multi_measure_records $i;
done
```

테이블 쿼리:

```
aws timestream-query query \
--query-string "SELECT time, device_id, device_type, os_version,
region, video_startup_time, rebuffering_ratio, video_playback_failures, \
average_frame_rate \
FROM metricsdb.metrics \
where time >= ago (15m)"
```

예약된 쿼리를 생성하려면:

```
aws timestream-query create-scheduled-query \
  --name scheduled_query_name \
  --query-string "select bin(time, 1m) as time, \
          avg(measure_value::double) as avg_cpu, min(measure_value::double) as min_cpu,
 region \
          from $src_db.$src_tbl where measure_name = 'cpu' \
          and time BETWEEN @scheduled_runtime - (interval '5' minute)  AND
 @scheduled_runtime \
          group by region, bin(time, 1m)" \
  --schedule-configuration "{\"ScheduleExpression\":\"$cron_exp\"}" \
  --notification-configuration "{\"SnsConfiguration\":{\"TopicArn\":\"$sns_topic_arn
\"}}" \
  --scheduled-query-execution-role-arn "arn:aws:iam::452360119086:role/
TimestreamSQExecutionRole" \
  --target-configuration "{\"TimestreamConfiguration\":{\
          \"DatabaseName\": \"$dest_db\",\
          \"TableName\": \"$dest_tbl\",\
          \"TimeColumn\":\"time\",\
          \"DimensionMappings\":[{\
            \"Name\": \"region\", \"DimensionValueType\": \"VARCHAR\"
          }],\
          \"MultiMeasureMappings\":{\
            \"TargetMultiMeasureName\": \"mma_name\",
            \"MultiMeasureAttributeMappings\":[{\
              \"SourceColumn\": \"avg_cpu\", \"MeasureValueType\": \"DOUBLE\",
 \"TargetMultiMeasureAttributeName\": \"target_avg_cpu\"
            },\
            { \
              \"SourceColumn\": \"min_cpu\", \"MeasureValueType\": \"DOUBLE\",
 \"TargetMultiMeasureAttributeName\": \"target_min_cpu\"
            }] \
          }\
          }}" \
  --error-report-configuration "{\"S3Configuration\": {\
```

```
        \"BucketName\": \"$s3_err_bucket\",\
        \"ObjectKeyPrefix\": \"scherrors\",\
        \"EncryptionOption\": \"SSE_S3\"\
        }\
      }"
```

사용 API

Amazon SDKsTimestream for LiveAnalytics 는 외에도 엔드포인트 검색 패턴 을 통해 직접 REST API 액세스합니다. 엔드포인트 검색 패턴은 사용 사례와 함께 아래에 설명되어 있습니다.

엔드포인트 검색 패턴

Timestream Live AnalyticsSDKs는 서비스 엔드포인트의 관리 및 매핑을 포함하여 서비스의 아키텍처와 투명하게 작동하도록 설계되었으므로 대부분의 애플리케이션에 SDKs 를 사용하는 것이 좋습니다. 그러나 API 엔드포인트 검색 패턴에 Timestream LiveAnalytics REST을 사용해야 하는 몇 가지 인스턴스가 있습니다.

- 에 VPC 대해 Timestream과 함께 엔드포인트(AWS PrivateLink)를 사용하고 있습니다. LiveAnalytics
- 애플리케이션에서 아직 SDK 지원되지 않는 프로그래밍 언어를 사용함
- 클라이언트 측 구현을 더 잘 제어해야 합니다.

이 섹션에는 엔드포인트 검색 패턴의 작동 방식, 엔드포인트 검색 패턴 구현 방법 및 사용 노트에 대한 정보가 포함되어 있습니다. 자세한 내용을 알아보려면 아래에서 주제를 선택하세요.

주제

- 엔드포인트 검색 패턴의 작동 방식
- 엔드포인트 검색 패턴 구현

엔드포인트 검색 패턴의 작동 방식

Timestream은 셀룰러 아키텍처를 사용하여 구축되어 더 나은 규모 조정과 트래픽 격리 속성을 보장합니다. 각 고객 계정은 리전의 특정 셀에 매핑되므로 애플리케이션은 계정이 매핑된 올바른 셀별 엔드포인트를 사용해야 합니다. 를 사용할 때 SDKs이 매핑은 투명하게 처리되므로 셀별 엔드포인트를 관리할 필요가 없습니다. 그러나 REST 에 직접 액세스할 때는 올바른 엔드포인트를 직접 관리하고 매핑 API해야 합니다. 엔드포인트 검색 패턴 인 이 프로세스는 아래에 설명되어 있습니다.

1. 엔드포인트 검색 패턴은 DescribeEndpoints 작업에 대한 호출로 시작됩니다 (DescribeEndpoints섹션에 설명됨).
2. 반환된 (TTL) 값()에 지정된 시간 동안 엔드포인트를 time-to-live 캐시하고 재사용해야 합니다CachePeriodInMinutes. 그런 다음 의 기간 동안 Timestream Live Analytics를 호출할 API 수 있습니다TTL.
3. 이 TTL 만료되면 엔드포인트를 새로 고치기 위해 에 대한 새 호출을 수행해야 DescribeEndpoints 합니다(즉, 1단계에서 다시 시작).

> **Note**
>
> DescribeEndpoints 작업에 대한 구문, 파라미터 및 기타 사용 정보는 API 참조 에 설명되어 있습니다. DescribeEndpoints 작업은 를 통해 사용할 수 SDKs있으며 각각에 대해 동일합니다.

엔드포인트 검색 패턴의 구현은 섹션을 참조하세요엔드포인트 검색 패턴 구현.

엔드포인트 검색 패턴 구현

엔드포인트 검색 패턴을 구현하려면 API (쓰기 또는 쿼리)를 선택하고 DescribeEndpoints 요청을 생성한 다음 반환된 TTL 값(들) 기간 동안 반환된 엔드포인트(들)를 사용합니다. 구현 절차는 아래에 설명되어 있습니다.

> **Note**
>
> 사용 정보 를 숙지해야 합니다.

구현 절차

1. DescribeEndpoints 요청을 사용하여 (쓰기 또는 쿼리)에 대해 호출API하려는 의 엔드포인트를 획득합니다.

   a. 아래 설명된 두 엔드포인트 중 하나를 사용하여 관심 API 대상(쓰기 또는 쿼리)에 DescribeEndpoints 해당하는 에 대한 요청을 생성합니다. 요청에 대한 입력 파라미터가 없습니다. 아래 참고 사항을 읽어야 합니다.

쓰기SDK:

```
ingest.timestream.<region>.amazonaws.com
```

쿼리SDK:

```
query.timestream.<region>.amazonaws.com
```

리전에 대한 예제 CLI 호출은 us-east-1 다음과 같습니다.

```
REGION_ENDPOINT="https://query.timestream.us-east-1.amazonaws.com"
REGION=us-east-1
aws timestream-write describe-endpoints \
--endpoint-url $REGION_ENDPOINT \
--region $REGION
```

> **Note**
>
> HTTP '호스트' 헤더에는 API 엔드포인트도 포함되어야 합니다. 헤더가 채워지지 않으면 요청이 실패합니다. 이는 모든 HTTP/1.1 요청에 대한 표준 요구 사항입니다. 1.1 이상을 지원하는 HTTP 라이브러리를 사용하는 경우 HTTP 라이브러리가 자동으로 헤더를 채웁니다.

> **Note**
>
> 대체 *<region>* 요청이 수행 중인 리전의 리전 식별자가 있는 경우. 예: us-east-1

b. 응답을 구문 분석하여 엔드포인트(들)와 캐시 TTL 값(들)을 추출합니다. 응답은 하나 이상의 Endpoint 객체 배열입니다. 각 Endpoint 객체에는 엔드포인트 주소(Address)와 해당 엔드포인트()TTL의 가 포함됩니다CachePeriodInMinutes.

2. 지정된 까지 엔드포인트를 캐시합니다TTL.
3. 이 TTL 만료되면 구현의 1단계에서 다시 시작하여 새 엔드포인트를 검색합니다.

엔드포인트 검색 패턴에 대한 사용 정보

- DescribeEndpoints 작업은 Timestream Live Analytics 리전 엔드포인트가 인식하는 유일한 작업입니다.
- 응답에는 Timestream Live Analytics를 API 호출할 엔드포인트 목록이 포함되어 있습니다.
- 응답이 성공하면 목록에 엔드포인트가 하나 이상 있어야 합니다. 목록에 엔드포인트가 두 개 이상 있는 경우 모든 엔드포인트를 API 호출에 동일하게 사용할 수 있으며 호출자는 임의로 사용할 엔드포인트를 선택할 수 있습니다.
- 엔드포인트의 DNS 주소 외에도 목록의 각 엔드포인트는 분 단위로 지정된 엔드포인트를 사용할 수 있는 라이브 시간(TTL)을 지정합니다.
- 엔드포인트는 캐시되고 반환된 TTL 값으로 지정된 시간(분) 동안 재사용되어야 합니다. 가 TTL 만료되면 엔드포인트TTL가 만료된 후에는 더 이상 작동하지 않으므로 에 대한 새 호출을 수행하여 사용할 엔드포인트를 새로 고쳐DescribeEndpoints야 합니다.

사용 AWS SDKs

를 사용하여 Amazon Timestream에 액세스할 수 있습니다 AWS SDKs. Timestream은 언어SDKs 당 2개, 즉 쓰기 SDK 및 쿼리를 지원합니다SDK. 쓰기는 CRUD 작업을 수행하고 시계열 데이터를 Timestream에 삽입하는 데 SDK 사용됩니다. 쿼리SDK는 Timestream에 저장된 기존 시계열 데이터를 쿼리하는 데 사용됩니다.

SDK 선택한 에 필요한 사전 조건을 완료하면 를 시작할 수 있습니다코드 샘플.

주제

- Java
- Java v2
- Go
- Python
- Node.js
- .NET

Java

Java 1.0 SDK 및 Amazon Timestream을 시작하려면 아래에 설명된 사전 조건을 완료하세요.

Java 에 필요한 사전 조건을 완료하면 를 시작할 SDK수 있습니다코드 샘플.

사전 조건

Java를 시작하기 전에 다음을 수행해야 합니다.

1. 의 AWS 설정 지침을 따릅니다에 대한 Timestream 액세스 LiveAnalytics.
2. 다음을 다운로드하고 설치하여 Java 개발 환경을 설정합니다.
   - Java SE 개발 키트 8(예: Amazon Corretto 8).
   - JavaIDE(예: Eclipse 또는 IntelliJ).

   자세한 내용은 시작하기를 참조하세요. AWS SDK for Java
3. 개발을 위해 AWS 자격 증명 및 리전을 구성합니다.
   - 와 함께 사용할 보안 자격 증명을 설정합니다 AWS AWS SDK for Java.
   - LiveAnalytics 엔드포인트의 AWS 기본 Timestream을 결정하도록 리전을 설정합니다.

Apache Maven 사용

Apache Maven을 사용하여 AWS SDK for Java 프로젝트를 구성하고 빌드할 수 있습니다.

> **Note**
>
> Apache Maven을 사용하려면 Java SDK 및 런타임이 1.8 이상이어야 합니다.

Apache Maven과 AWS SDK 함께 사용 에 설명된 대로 를 Maven 종속성으로 구성할 수 있습니다. SDK

다음 명령을 사용하여 컴파일을 실행하고 소스 코드를 실행할 수 있습니다.

```
mvn clean compile
mvn exec:java -Dexec.mainClass=<your source code Main class>
```

> **Note**
>
> `<your source code Main class>` 는 Java 소스 코드의 기본 클래스 경로입니다.

자격 증명 설정 AWS

에서는 런타임 시 애플리케이션에 AWS 보안 인증 정보를 제공해야 AWS SDK for Java 합니다. 이 안내서의 코드 예제에서는 AWS SDK for Java 개발자 안내서의 개발을 위한 자격 증명 및 리전 설정에 설명된 대로 자격 AWS 증명 파일을 사용하고 AWS 있다고 가정합니다.

다음은 라는 AWS 자격 증명 파일의 예입니다. `~/.aws/credentials`여기서 틸드 문자(~)는 홈 디렉터리를 나타냅니다.

```
[default]
aws_access_key_id = AWS access key ID goes here
aws_secret_access_key = Secret key goes here
```

Java v2

Java 2.0 SDK 및 Amazon Timestream을 시작하려면 아래 설명된 사전 조건을 완료하세요.

Java 2.0 에 필요한 사전 조건을 완료SDK하면 를 시작할 수 있습니다코드 샘플.

사전 조건

Java를 시작하기 전에 다음을 수행해야 합니다.

1. 의 AWS 설정 지침을 따릅니다에 대한 Timestream 액세스 LiveAnalytics.
2. Apache Maven과 AWS SDK 함께 사용 에 설명된 대로 를 Maven 종속성으로 구성할 수 있습니다. SDK
3. 다음을 다운로드하고 설치하여 Java 개발 환경을 설정합니다.
   - Java SE 개발 키트 8(예: Amazon Corretto 8).
   - JavaIDE(예: Eclipse 또는 IntelliJ).

     자세한 내용은 시작하기를 참조하세요. AWS SDK for Java

Apache Maven 사용

Apache Maven을 사용하여 AWS SDK for Java 프로젝트를 구성하고 빌드할 수 있습니다.

> **Note**
>
> Apache Maven을 사용하려면 Java SDK 및 런타임이 1.8 이상이어야 합니다.

Apache Maven과 AWS SDK 함께 사용 에 설명된 대로 를 Maven 종속성으로 구성할 수 있습니다. SDK pom.xml 파일에 필요한 변경 사항은 여기에 설명되어 있습니다.

다음 명령을 사용하여 컴파일을 실행하고 소스 코드를 실행할 수 있습니다.

```
mvn clean compile
mvn exec:java -Dexec.mainClass=<your source code Main class>
```

> **Note**
>
> `<your source code Main class>` 는 Java 소스 코드의 기본 클래스 경로입니다.

Go

Go SDK 및 Amazon Timestream을 시작하려면 아래에 설명된 사전 조건을 완료하세요.

Go 에 필요한 사전 조건을 완료SDK하면 를 시작할 수 있습니다코드 샘플.

사전 조건

1. GO SDK 1.14를 다운로드합니다.
2. GO 를 구성합니다SDK.
3. 클라이언트를 구성합니다.

Python

Python SDK 및 Amazon Timestream을 시작하려면 아래에 설명된 사전 조건을 완료합니다.

Python 에 필요한 사전 조건을 완료하면 를 시작할 SDK수 있습니다코드 샘플.

사전 조건

Python을 사용하려면 https://boto3.amazonaws.com/v1/documentation/api/latest/index.html의 지침에 따라 Boto3를 설치하고 구성합니다.

Node.js

Node.js SDK 및 Amazon Timestream을 시작하려면 아래 설명된 사전 조건을 완료하세요.

Node.js 에 필요한 사전 조건을 완료SDK하면 를 시작할 수 있습니다코드 샘플.

사전 조건

Node.js를 시작하기 전에 다음을 수행해야 합니다.

1. Node.js 를 설치합니다.
2. 용 를 AWS SDK 설치합니다 JavaScript.

.NET

. 및 Amazon Timestream을NET SDK 시작하려면 아래에 설명된 사전 조건을 완료하세요.

에 필요한 사전 조건을NET 완료하면 를 시작할 SDK수 있습니다코드 샘플.

사전 조건

를 시작하기 전에 필요한 NuGet 패키지를 NET설치하고 다음 명령을 실행하여 AWSSDK.Core 버전이 3.3.107 이상인지 확인합니다.

```
dotnet add package AWSSDK.Core
dotnet add package AWSSDK.TimestreamWrite
dotnet add package AWSSDK.TimestreamQuery
```

시작하기

이 섹션에는 Amazon Timestream Live Analytics를 시작하기 위한 자습서와 전체 기능 샘플 애플리케이션 설정 지침이 포함되어 있습니다. 아래 링크 중 하나를 선택하여 자습서 또는 샘플 애플리케이션을 시작할 수 있습니다.

주제

- 튜토리얼
- 샘플 애플리케이션

튜토리얼

이 자습서에서는 샘플 데이터 세트로 채워진 데이터베이스를 생성하고 샘플 쿼리를 실행하는 방법을 보여줍니다. 이 자습서에서 사용되는 샘플 데이터 세트는 IoT 및 DevOps 시나리오에서 자주 볼 수 있습니다. IoT 데이터 세트에는 트럭의 속도, 위치 및 부하와 같은 시계열 데이터가 포함되어 있어 플릿

관리를 간소화하고 최적화 기회를 식별할 수 있습니다. DevOps 데이터 세트에는 애플리케이션 성능 및 가용성을 개선하기 위한 , CPU네트워크 및 메모리 사용률과 같은 EC2 인스턴스 지표가 포함되어 있습니다. 다음은 이 섹션에 설명된 지침에 대한 비디오 자습서입니다.

다음 단계에 따라 샘플 데이터 세트로 채워진 데이터베이스를 생성하고 AWS 콘솔을 사용하여 샘플 쿼리를 실행합니다.

콘솔 사용

다음 단계에 따라 샘플 데이터 세트로 채워진 데이터베이스를 생성하고 AWS 콘솔을 사용하여 샘플 쿼리를 실행합니다.

1. AWS 콘솔 을 엽니다.
2. 탐색 창에서 데이터베이스를 선택합니다.
3. 데이터베이스 생성 을 클릭합니다.
4. 데이터베이스 생성 페이지에서 다음을 입력합니다.
   - 구성 선택 - 샘플 데이터베이스 를 선택합니다.
   - 이름 - 선택한 데이터베이스 이름을 입력합니다.

     > **Note**
     >
     > 샘플 데이터 세트가 포함된 데이터베이스를 생성한 후 콘솔에서 사용할 수 있는 샘플 쿼리를 사용하려면 여기에 입력한 데이터베이스 이름과 일치하도록 쿼리에서 참조된 데이터베이스 이름을 조정할 수 있습니다. 샘플 데이터 세트와 시계열 레코드 유형의 각 조합에 대한 샘플 쿼리가 있습니다.

   - 샘플 데이터 세트 선택 - IoT 및 를 선택합니다DevOps.
   - 시계열 레코드 유형 선택 - 다중 측정 레코드를 선택합니다.
   - 데이터베이스 생성을 클릭하여 샘플 데이터로 채워진 두 개의 테이블이 포함된 데이터베이스를 생성합니다. 다중 측정 레코드가 있는 샘플 데이터 세트의 테이블 이름은 `DevOpsMulti` 및 입니다`IoTMulti`. 단일 측정 레코드가 있는 샘플 데이터 세트의 테이블 이름은 `DevOps` 및 입니다`IoT`.
5. 탐색 창에서 쿼리 편집기를 선택합니다.
6. 상단 메뉴에서 샘플 쿼리를 선택합니다.
7. 샘플 데이터베이스를 생성할 때 선택한 데이터 세트에 대한 샘플 쿼리 중 하나를 클릭합니다. 그러면 샘플 쿼리로 채워진 편집기가 있는 쿼리 편집기로 돌아갑니다.

8. 샘플 쿼리의 데이터베이스 이름을 조정합니다.
9. 실행을 클릭하여 쿼리를 실행하고 쿼리 결과를 확인합니다.

사용 SDKs

Timestream Live Analytics는 데이터베이스 및 테이블을 생성하고, 테이블을 샘플 데이터의 ~126K 행으로 채우고, 샘플 쿼리를 실행하는 방법을 보여주는 완전한 기능을 갖춘 샘플 애플리케이션을 제공합니다. 샘플 애플리케이션은 Java, Python, Node.js, Go 및 GitHub에서 사용할 수 있습니다NET.

1. 의 지침에 따라 GitHub 리포지토리 Timestream Live Analytics 샘플 애플리케이션을 복제합니다 GitHub.
2. 에 AWS SDK 설명된 지침에 따라 Amazon Timestream Live Analytics에 연결하도록 사용 AWS SDKs 를 구성합니다.
3. 아래 지침에 따라 샘플 애플리케이션을 컴파일하고 실행합니다.

   - Java 샘플 애플리케이션 에 대한 지침입니다.
   - Java v2 샘플 애플리케이션 에 대한 지침입니다.
   - Go 샘플 애플리케이션 에 대한 지침입니다.
   - Python 샘플 애플리케이션 에 대한 지침입니다.
   - Node.js 샘플 애플리케이션 에 대한 지침입니다.
   - .NET 샘플 애플리케이션 에 대한 지침입니다.

샘플 애플리케이션

Timestream은 데이터베이스 및 테이블을 생성하고, 테이블을 샘플 데이터의 ~126K 행으로 채우고, 샘플 쿼리를 실행하는 방법을 보여주는 완전한 기능의 샘플 애플리케이션과 함께 제공됩니다. 지원되는 언어로 샘플 애플리케이션을 시작하려면 아래 단계를 따르세요.

Java

1. 의 지침에 따라 샘플 애플리케이션의 GitHub 리포지토리 Timestream을 복제합니다GitHub. LiveAnalytics
2. 시작하기에 설명된 지침에 LiveAnalytics 따라 를 Timestream에 연결AWSSDK하도록 구성합니다Java.
3. 여기에 설명된 지침에 따라 Java 샘플 애플리케이션을 실행합니다.

Java v2

1. 의 지침에 따라 샘플 애플리케이션의 GitHub 리포지토리 Timestream을 복제합니다GitHub. LiveAnalytics
2. 시작하기에 설명된 지침에 LiveAnalytics 따라 를 AWS SDK Amazon Timestream에 연결하도록 구성합니다Java v2.
3. 여기에 설명된 지침에 따라 Java 2.0 샘플 애플리케이션을 실행합니다.

Go

1. 의 지침에 따라 샘플 애플리케이션의 GitHub 리포지토리 Timestream을 복제합니다GitHub. LiveAnalytics
2. 시작하기에 설명된 지침에 LiveAnalytics 따라 를 AWS SDK Amazon Timestream에 연결하도록 구성합니다Go.
3. 여기에 설명된 지침에 따라 Go 샘플 애플리케이션을 실행합니다.

Python

1. 의 지침에 따라 샘플 애플리케이션의 GitHub 리포지토리 Timestream을 복제합니다GitHub. LiveAnalytics
2. 에 AWS SDK 설명된 지침에 LiveAnalytics 따라 를 Amazon Timestream에 연결하도록 구성합니다Python.
3. 여기에 설명된 지침에 따라 Python 샘플 애플리케이션을 실행합니다.

Node.js

1. 의 지침에 따라 샘플 애플리케이션의 GitHub 리포지토리 Timestream을 복제합니다GitHub. LiveAnalytics
2. 시작하기에 설명된 지침에 LiveAnalytics 따라 를 AWS SDK Amazon Timestream에 연결하도록 구성합니다Node.js.
3. 여기에 설명된 지침에 따라 Node.js 샘플 애플리케이션을 실행합니다.

.NET

1. 의 지침에 따라 샘플 애플리케이션의 GitHub 리포지토리 Timestream을 복제합니다GitHub. LiveAnalytics
2. 시작하기에 설명된 지침에 LiveAnalytics 따라 를 AWS SDK Amazon Timestream에 연결하도록 구성합니다.NET.
3. 여기에 설명된 지침에 따라 .NET 샘플 애플리케이션을 실행합니다.

코드 샘플

를 사용하여 Amazon Timestream에 액세스할 수 있습니다 AWS SDKs. Timestream은 언어SDKs 당 2개, 즉 쓰기 SDK 및 쿼리를 지원합니다SDK. 쓰기는 CRUD 작업을 수행하고 시계열 데이터를 Timestream에 삽입하는 데 SDK 사용됩니다. 쿼리SDK는 Timestream에 저장된 기존 시계열 데이터를 쿼리하는 데 사용됩니다. 지원되는 각 에 대한 코드 샘플을 포함하여 자세한 내용을 알아보려면 아래 목록에서 주제를 선택합니다SDKs.

주제

- SDK 클라이언트 쓰기
- 쿼리 SDK 클라이언트
- 데이터베이스 생성
- 데이터베이스 설명
- 데이터베이스 업데이트
- 데이터베이스 삭제
- 데이터베이스 나열
- 테이블 생성
- 테이블 설명
- 테이블 업데이트
- 테이블 삭제
- 테이블 나열
- 데이터 쓰기(삽입 및 업서트)
- 쿼리 실행
- UNLOAD 쿼리 실행
- 쿼리 취소

- 배치 로드 작업 생성
- 배치 로드 작업 설명
- 배치 로드 작업 나열
- 배치 로드 작업 재개
- 예약된 쿼리 생성
- 예약된 쿼리 나열
- 예약된 쿼리 설명
- 예약된 쿼리 실행
- 예약된 쿼리 업데이트
- 예약된 쿼리 삭제

SDK 클라이언트 쓰기

다음 코드 조각을 사용하여 쓰기 에 대한 Timestream 클라이언트를 생성할 수 있습니다SDK. 쓰기는 CRUD 작업을 수행하고 시계열 데이터를 Timestream에 삽입하는 데 SDK 사용됩니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
private static AmazonTimestreamWrite buildWriteClient() {
    final ClientConfiguration clientConfiguration = new ClientConfiguration()
            .withMaxConnections(5000)
            .withRequestTimeout(20 * 1000)
            .withMaxErrorRetry(10);

    return AmazonTimestreamWriteClientBuilder
            .standard()
            .withRegion("us-east-1")
            .withClientConfiguration(clientConfiguration)
            .build();
}
```

Java v2

```
    private static TimestreamWriteClient buildWriteClient() {
        ApacheHttpClient.Builder httpClientBuilder =
                ApacheHttpClient.builder();
        httpClientBuilder.maxConnections(5000);

        RetryPolicy.Builder retryPolicy =
                RetryPolicy.builder();
        retryPolicy.numRetries(10);

        ClientOverrideConfiguration.Builder overrideConfig =
                ClientOverrideConfiguration.builder();
        overrideConfig.apiCallAttemptTimeout(Duration.ofSeconds(20));
        overrideConfig.retryPolicy(retryPolicy.build());

        return TimestreamWriteClient.builder()
                .httpClientBuilder(httpClientBuilder)
                .overrideConfiguration(overrideConfig.build())
                .region(Region.US_EAST_1)
                .build();
    }
```

Go

```
tr := &http.Transport{
        ResponseHeaderTimeout: 20 * time.Second,
        // Using DefaultTransport values for other parameters: https://golang.org/
pkg/net/http/#RoundTripper
        Proxy: http.ProxyFromEnvironment,
        DialContext: (&net.Dialer{
            KeepAlive: 30 * time.Second,
            DualStack: true,
            Timeout:   30 * time.Second,
        }).DialContext,
        MaxIdleConns:          100,
        IdleConnTimeout:       90 * time.Second,
        TLSHandshakeTimeout:   10 * time.Second,
        ExpectContinueTimeout: 1 * time.Second,
    }

    // So client makes HTTP/2 requests
    http2.ConfigureTransport(tr)
```

```
    sess, err := session.NewSession(&aws.Config{ Region: aws.String("us-east-1"),
 MaxRetries: aws.Int(10), HTTPClient: &http.Client{ Transport: tr }})
    writeSvc := timestreamwrite.New(sess)
```

Python

```
write_client = session.client('timestream-write', config=Config(read_timeout=20,
 max_pool_connections = 5000, retries={'max_attempts': 10}))
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

여기에 추가 명령 가져오기가 표시됩니다. `CreateDatabaseCommand` 가져오기는 클라이언트를 생성하는 데 필요하지 않습니다.

```
import { TimestreamWriteClient, CreateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
```

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
var https = require('https');
var agent = new https.Agent({
    maxSockets: 5000
});
writeClient = new AWS.TimestreamWrite({
        maxRetries: 10,
        httpOptions: {
            timeout: 20000,
            agent: agent
        }
    });
```

.NET

```
var writeClientConfig = new AmazonTimestreamWriteConfig
{
    RegionEndpoint = RegionEndpoint.USEast1,
```

```
    Timeout = TimeSpan.FromSeconds(20),
    MaxErrorRetry = 10
};

var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
```

다음 구성을 사용하는 것이 좋습니다.

- SDK 재시도 횟수를 로 설정합니다10.
- SDK DEFAULT_BACKOFF_STRATEGY를 사용합니다.
- 20 초RequestTimeout로 설정합니다.
- 최대 연결을 5000 이상으로 설정합니다.

쿼리 SDK 클라이언트

다음 코드 조각을 사용하여 쿼리 에 대한 Timestream 클라이언트를 생성할 수 있습니다SDK. 쿼리SDK는 Timestream에 저장된 기존 시계열 데이터를 쿼리하는 데 사용됩니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
    private static AmazonTimestreamQuery buildQueryClient() {
        AmazonTimestreamQuery client =
AmazonTimestreamQueryClient.builder().withRegion("us-east-1").build();
        return client;
    }
```

Java v2

```
    private static TimestreamQueryClient buildQueryClient() {
        return TimestreamQueryClient.builder()
                .region(Region.US_EAST_1)
                .build();
```

```
    }
```

Go

```
sess, err := session.NewSession(&aws.Config{Region: aws.String("us-east-1")})
```

Python

```
query_client = session.client('timestream-query')
```

Node.js

다음 조각은 JavaScript v3에 를 사용합니다 AWS SDK. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3의 Timestream Query Client - 를AWS SDK 참조하세요.

여기에 추가 명령 가져오기가 표시됩니다. `QueryCommand` 가져오기는 클라이언트를 생성하는 데 필요하지 않습니다.

```
import { TimestreamQueryClient, QueryCommand } from "@aws-sdk/client-timestream-
query";
const queryClient = new TimestreamQueryClient({ region: "us-east-1" });
```

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
queryClient = new AWS.TimestreamQuery();
```

.NET

```
var queryClientConfig = new AmazonTimestreamQueryConfig
{
    RegionEndpoint = RegionEndpoint.USEast1
};

var queryClient = new AmazonTimestreamQueryClient(queryClientConfig);
```

데이터베이스 생성

다음 코드 조각을 사용하여 데이터베이스를 생성할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
  public void createDatabase() {
       System.out.println("Creating database");
       CreateDatabaseRequest request = new CreateDatabaseRequest();
       request.setDatabaseName(DATABASE_NAME);
       try {
           amazonTimestreamWrite.createDatabase(request);
           System.out.println("Database [" + DATABASE_NAME + "] created
successfully");
       } catch (ConflictException e) {
           System.out.println("Database [" + DATABASE_NAME + "] exists. Skipping
database creation");
       }
   }
```

Java v2

```
   public void createDatabase() {
       System.out.println("Creating database");
       CreateDatabaseRequest request =
CreateDatabaseRequest.builder().databaseName(DATABASE_NAME).build();
       try {
           timestreamWriteClient.createDatabase(request);
           System.out.println("Database [" + DATABASE_NAME + "] created
successfully");
       } catch (ConflictException e) {
           System.out.println("Database [" + DATABASE_NAME + "] exists. Skipping
database creation");
       }
   }
```

Go

```
// Create database.
    createDatabaseInput := &timestreamwrite.CreateDatabaseInput{
```

```
        DatabaseName: aws.String(*databaseName),
    }

    _, err = writeSvc.CreateDatabase(createDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Database successfully created")
    }

    fmt.Println("Describing the database, hit enter to continue")
```

Python

```
    def create_database(self):
        print("Creating Database")
        try:
            self.client.create_database(DatabaseName=Constant.DATABASE_NAME)
            print("Database [%s] created successfully." % Constant.DATABASE_NAME)
        except self.client.exceptions.ConflictException:
            print("Database [%s] exists. Skipping database creation" %
Constant.DATABASE_NAME)
        except Exception as err:
            print("Create database failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 CreateDatabaseCommand 및 도 참조하세요CreateDatabase.

```
import { TimestreamWriteClient, CreateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
};

const command = new CreateDatabaseCommand(params);
```

```
try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} created successfully`);
} catch (error) {
    if (error.code === 'ConflictException') {
        console.log(`Database ${params.DatabaseName} already exists. Skipping
 creation.`);
    } else {
        console.log("Error creating database", error);
    }
}
```

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream LiveAnalytics 의 샘플 애플리케이션을 기반으로 합니다 GitHub.

```
async function createDatabase() {
    console.log("Creating Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.createDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} created
 successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Database ${params.DatabaseName} already exists.
 Skipping creation.`);
            } else {
                console.log("Error creating database", err);
            }
        }
    );
}
```

.NET

```
        public async Task CreateDatabase()
```

```
        {
            Console.WriteLine("Creating Database");

            try
            {
                var createDatabaseRequest = new CreateDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                CreateDatabaseResponse response = await
writeClient.CreateDatabaseAsync(createDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Database already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create database failed:" + e.ToString());
            }

        }
```

데이터베이스 설명

다음 코드 조각을 사용하여 새로 생성된 데이터베이스의 속성에 대한 정보를 가져올 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
   public void describeDatabase() {
       System.out.println("Describing database");
       final DescribeDatabaseRequest describeDatabaseRequest = new
DescribeDatabaseRequest();
       describeDatabaseRequest.setDatabaseName(DATABASE_NAME);
```

```
        try {
            DescribeDatabaseResult result =
 amazonTimestreamWrite.describeDatabase(describeDatabaseRequest);
            final Database databaseRecord = result.getDatabase();
            final String databaseId = databaseRecord.getArn();
            System.out.println("Database " + DATABASE_NAME + " has id " +
 databaseId);
        } catch (final Exception e) {
            System.out.println("Database doesn't exist = " + e);
            throw e;
        }
    }
```

Java v2

```
    public void describeDatabase() {
        System.out.println("Describing database");
        final DescribeDatabaseRequest describeDatabaseRequest =
 DescribeDatabaseRequest.builder()
                .databaseName(DATABASE_NAME).build();
        try {
            DescribeDatabaseResponse response =
 timestreamWriteClient.describeDatabase(describeDatabaseRequest);
            final Database databaseRecord = response.database();
            final String databaseId = databaseRecord.arn();
            System.out.println("Database " + DATABASE_NAME + " has id " +
 databaseId);
        } catch (final Exception e) {
            System.out.println("Database doesn't exist = " + e);
            throw e;
        }
    }
```

Go

```
describeDatabaseOutput, err := writeSvc.DescribeDatabase(describeDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Describe database is successful, below is the output:")
        fmt.Println(describeDatabaseOutput)
```

```
    }
```

Python

```
    def describe_database(self):
        print("Describing database")
        try:
            result =
 self.client.describe_database(DatabaseName=Constant.DATABASE_NAME)
            print("Database [%s] has id [%s]" % (Constant.DATABASE_NAME,
 result['Database']['Arn']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Database doesn't exist")
        except Exception as err:
            print("Describe database failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 DescribeDatabaseCommand 및 도 참조하세요DescribeDatabase.

```
import { TimestreamWriteClient, DescribeDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
};

const command = new DescribeDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} has id
 ${data.Database.Arn}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Database doesn't exist.");
    } else {
        console.log("Describe database failed.", error);
        throw error;
```

```
    }
}
```

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream LiveAnalytics 의 샘플 애플리케이션을 기반으로 합니다 GitHub.

```
async function describeDatabase () {
    console.log("Describing Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.describeDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} has id
 ${data.Database.Arn}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Database doesn't exist.");
            } else {
                console.log("Describe database failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DescribeDatabase()
        {
            Console.WriteLine("Describing Database");

            try
            {
                var describeDatabaseRequest = new DescribeDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
```

```
                DescribeDatabaseResponse response = await
 writeClient.DescribeDatabaseAsync(describeDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} has id:
{response.Database.Arn}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Database does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe database failed:" + e.ToString());
            }

        }
```

데이터베이스 업데이트

다음 코드 조각을 사용하여 데이터베이스를 업데이트할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
    public void updateDatabase(String kmsId) {
        System.out.println("Updating kmsId to " + kmsId);
        UpdateDatabaseRequest request = new UpdateDatabaseRequest();
        request.setDatabaseName(DATABASE_NAME);
        request.setKmsKeyId(kmsId);
        try {
            UpdateDatabaseResult result =
 amazonTimestreamWrite.updateDatabase(request);
            System.out.println("Update Database complete");
        } catch (final ValidationException e) {
            System.out.println("Update database failed:");
            e.printStackTrace();
        } catch (final ResourceNotFoundException e) {
```

```
            System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
        } catch (final Exception e) {
            System.out.println("Could not update Database " + DATABASE_NAME + " = "
+ e);
            throw e;
        }
    }
```

Java v2

```
    public void updateDatabase(String kmsKeyId) {

        if (kmsKeyId == null) {
            System.out.println("Skipping UpdateDatabase because KmsKeyId was not
given");
            return;
        }

        System.out.println("Updating database");

        UpdateDatabaseRequest request = UpdateDatabaseRequest.builder()
                .databaseName(DATABASE_NAME)
                .kmsKeyId(kmsKeyId)
                .build();
        try {
            timestreamWriteClient.updateDatabase(request);
            System.out.println("Database [" + DATABASE_NAME + "] updated
successfully with kmsKeyId " + kmsKeyId);
        } catch (ResourceNotFoundException e) {
            System.out.println("Database [" + DATABASE_NAME + "] does not exist.
Skipping UpdateDatabase");
        } catch (Exception e) {
            System.out.println("UpdateDatabase failed: " + e);
        }
    }
```

Go

```
// Update Database.
        updateDatabaseInput := &timestreamwrite.UpdateDatabaseInput {
            DatabaseName: aws.String(*databaseName),
            KmsKeyId: aws.String(*kmsKeyId),
```

```
        }

        updateDatabaseOutput, err := writeSvc.UpdateDatabase(updateDatabaseInput)

        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update database is successful, below is the output:")
            fmt.Println(updateDatabaseOutput)
        }
```

Python

```
    def update_database(self, kms_id):
        print("Updating database")
        try:
            result =
self.client.update_database(DatabaseName=Constant.DATABASE_NAME, KmsKeyId=kms_id)
            print("Database [%s] was updated to use kms [%s] successfully" %
(Constant.DATABASE_NAME,

result['Database']['KmsKeyId']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Database doesn't exist")
        except Exception as err:
            print("Update database failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 UpdateDatabaseCommand 및 도 참조하세요UpdateDatabase.

```
import { TimestreamWriteClient, UpdateDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });
let updatedKmsKeyId = "<updatedKmsKeyId>";

const params = {
    DatabaseName: "testDbFromNode",
    KmsKeyId: updatedKmsKeyId
```

```
};

const command = new UpdateDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Database ${data.Database.DatabaseName} updated kmsKeyId to
 ${updatedKmsKeyId}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Database doesn't exist.");
    } else {
        console.log("Update database failed.", error);
    }
}
```

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function updateDatabase(updatedKmsKeyId) {

    if (updatedKmsKeyId === undefined) {
        console.log("Skipping UpdateDatabase; KmsKeyId was not given");
        return;
    }
    console.log("Updating Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        KmsKeyId: updatedKmsKeyId
    }

    const promise = writeClient.updateDatabase(params).promise();

    await promise.then(
        (data) => {
            console.log(`Database ${data.Database.DatabaseName} updated kmsKeyId to
 ${updatedKmsKeyId}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Database doesn't exist.");
            } else {
                console.log("Update database failed.", err);
```

```
            }
        }
    );
}
```

.NET

```
        public async Task UpdateDatabase(String updatedKmsKeyId)
        {
            Console.WriteLine("Updating Database");

            try
            {
                var updateDatabaseRequest = new UpdateDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    KmsKeyId = updatedKmsKeyId
                };
                UpdateDatabaseResponse response = await
 writeClient.UpdateDatabaseAsync(updateDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} updated with
 KmsKeyId {updatedKmsKeyId}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Database does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Update database failed: " + e.ToString());
            }

        }

        private void PrintDatabases(List<Database> databases)
        {
            foreach (Database database in databases)
                Console.WriteLine($"Database:{database.DatabaseName}");
        }
```

데이터베이스 삭제

다음 코드 조각을 사용하여 데이터베이스를 삭제할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
   public void deleteDatabase() {
       System.out.println("Deleting database");
       final DeleteDatabaseRequest deleteDatabaseRequest = new
DeleteDatabaseRequest();
       deleteDatabaseRequest.setDatabaseName(DATABASE_NAME);
       try {
           DeleteDatabaseResult result =
                   amazonTimestreamWrite.deleteDatabase(deleteDatabaseRequest);
           System.out.println("Delete database status: " +
result.getSdkHttpMetadata().getHttpStatusCode());
       } catch (final ResourceNotFoundException e) {
           System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
e);
           throw e;
       } catch (final Exception e) {
           System.out.println("Could not delete Database " + DATABASE_NAME + " = "
+ e);
           throw e;
       }
   }
```

Java v2

```
   public void deleteDatabase() {
       System.out.println("Deleting database");
       final DeleteDatabaseRequest deleteDatabaseRequest = new
DeleteDatabaseRequest();
       deleteDatabaseRequest.setDatabaseName(DATABASE_NAME);
       try {
```

```
            DeleteDatabaseResult result =
                    amazonTimestreamWrite.deleteDatabase(deleteDatabaseRequest);
            System.out.println("Delete database status: " +
 result.getSdkHttpMetadata().getHttpStatusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Database " + DATABASE_NAME + " doesn't exist = " +
 e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete Database " + DATABASE_NAME + " = "
 + e);
            throw e;
        }
    }
```

Go

```
deleteDatabaseInput := &timestreamwrite.DeleteDatabaseInput{
        DatabaseName:   aws.String(*databaseName),
    }

    _, err = writeSvc.DeleteDatabase(deleteDatabaseInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Database deleted:", *databaseName)
    }
```

Python

```
    def delete_database(self):
        print("Deleting Database")
        try:
            result =
 self.client.delete_database(DatabaseName=Constant.DATABASE_NAME)
            print("Delete database status [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
        except self.client.exceptions.ResourceNotFoundException:
            print("database [%s] doesn't exist" % Constant.DATABASE_NAME)
        except Exception as err:
            print("Delete database failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 DeleteDatabaseCommand 및 도 참조하세요DeleteDatabase.

```
import { TimestreamWriteClient, DeleteDatabaseCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode"
};

const command = new DeleteDatabaseCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Deleted database");
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log(`Database ${params.DatabaseName} doesn't exists.`);
    } else {
        console.log("Delete database failed.", error);
        throw error;
    }
}
```

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function deleteDatabase() {
    console.log("Deleting Database");
    const params = {
        DatabaseName: constants.DATABASE_NAME
    };

    const promise = writeClient.deleteDatabase(params).promise();

    await promise.then(
        function (data) {
            console.log("Deleted database");
```

```
         },
        function(err) {
            if (err.code === 'ResourceNotFoundException') {
                console.log(`Database ${params.DatabaseName} doesn't exists.`);
            } else {
                console.log("Delete database failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DeleteDatabase()
        {
            Console.WriteLine("Deleting database");
            try
            {
                var deleteDatabaseRequest = new DeleteDatabaseRequest
                {
                    DatabaseName = Constants.DATABASE_NAME
                };
                DeleteDatabaseResponse response = await
 writeClient.DeleteDatabaseAsync(deleteDatabaseRequest);
                Console.WriteLine($"Database {Constants.DATABASE_NAME} delete
 request status:{response.HttpStatusCode}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine($"Database {Constants.DATABASE_NAME} does not
 exists");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception while deleting database:" +
 e.ToString());
            }
        }
```

데이터베이스 나열

다음 코드 조각을 사용하여 데이터베이스를 나열할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
    public void listDatabases() {
        System.out.println("Listing databases");
        ListDatabasesRequest request = new ListDatabasesRequest();
        ListDatabasesResult result = amazonTimestreamWrite.listDatabases(request);
        final List<Database> databases = result.getDatabases();
        printDatabases(databases);

        String nextToken = result.getNextToken();
        while (nextToken != null && !nextToken.isEmpty()) {
            request.setNextToken(nextToken);
            ListDatabasesResult nextResult =
amazonTimestreamWrite.listDatabases(request);
            final List<Database> nextDatabases = nextResult.getDatabases();
            printDatabases(nextDatabases);
            nextToken = nextResult.getNextToken();
        }
    }

    private void printDatabases(List<Database> databases) {
        for (Database db : databases) {
            System.out.println(db.getDatabaseName());
        }
    }
```

Java v2

```
    public void listDatabases() {
        System.out.println("Listing databases");
        ListDatabasesRequest request =
ListDatabasesRequest.builder().maxResults(2).build();
```

```
        ListDatabasesIterable listDatabasesIterable =
 timestreamWriteClient.listDatabasesPaginator(request);
        for(ListDatabasesResponse listDatabasesResponse : listDatabasesIterable) {
            final List<Database> databases = listDatabasesResponse.databases();
            databases.forEach(database ->
 System.out.println(database.databaseName()));
        }
    }
```

Go

```
// List databases.
    listDatabasesMaxResult := int64(15)

    listDatabasesInput := &timestreamwrite.ListDatabasesInput{
        MaxResults: &listDatabasesMaxResult,
    }

    listDatabasesOutput, err := writeSvc.ListDatabases(listDatabasesInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("List databases is successful, below is the output:")
        fmt.Println(listDatabasesOutput)
    }
```

Python

```
    def list_databases(self):
        print("Listing databases")
        try:
            result = self.client.list_databases(MaxResults=5)
            self._print_databases(result['Databases'])
            next_token = result.get('NextToken', None)
            while next_token:
                result = self.client.list_databases(NextToken=next_token,
 MaxResults=5)
                self._print_databases(result['Databases'])
                next_token = result.get('NextToken', None)
        except Exception as err:
            print("List databases failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 ListDatabasesCommand 및 도 참조하세요ListDatabases.

```
import { TimestreamWriteClient, ListDatabasesCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    MaxResults: 15
};

const command = new ListDatabasesCommand(params);

getDatabasesList(null);

async function getDatabasesList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.Databases.forEach(function (database) {
            console.log(database.DatabaseName);
        });

        if (data.NextToken) {
            return getDatabasesList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing databases", error);
    }
}
```

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream LiveAnalytics 의 샘플 애플리케이션을 기반으로 합니다 GitHub.

```
async function listDatabases() {
```

```
    console.log("Listing databases:");
    const databases = await getDatabasesList(null);
    databases.forEach(function(database){
        console.log(database.DatabaseName);
    });
}

function getDatabasesList(nextToken, databases = []) {
    var params = {
        MaxResults: 15
    };

    if(nextToken) {
        params.NextToken = nextToken;
    }

    return writeClient.listDatabases(params).promise()
        .then(
            (data) => {
                databases.push.apply(databases, data.Databases);
                if (data.NextToken) {
                    return getDatabasesList(data.NextToken, databases);
                } else {
                    return databases;
                }
            },
            (err) => {
                console.log("Error while listing databases", err);
            });
}
```

.NET

```
        public async Task ListDatabases()
        {
            Console.WriteLine("Listing Databases");

            try
            {
                var listDatabasesRequest = new ListDatabasesRequest
                {
                    MaxResults = 5
                };
```

```
                ListDatabasesResponse response = await
writeClient.ListDatabasesAsync(listDatabasesRequest);
                PrintDatabases(response.Databases);
                var nextToken = response.NextToken;
                while (nextToken != null)
                {
                    listDatabasesRequest.NextToken = nextToken;
                    response = await
writeClient.ListDatabasesAsync(listDatabasesRequest);
                    PrintDatabases(response.Databases);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List database failed:" + e.ToString());
            }

        }
```

테이블 생성

주제

- 메모리 스토어 쓰기
- 마그네틱 스토어 쓰기

메모리 스토어 쓰기

다음 코드 조각을 사용하여 마그네틱 스토어 쓰기가 비활성화된 테이블을 생성할 수 있습니다. 따라서 메모리 스토어 보존 기간에만 데이터를 쓸 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
   public void createTable() {
       System.out.println("Creating table");
       CreateTableRequest createTableRequest = new CreateTableRequest();
       createTableRequest.setDatabaseName(DATABASE_NAME);
       createTableRequest.setTableName(TABLE_NAME);
       final RetentionProperties retentionProperties = new RetentionProperties()
               .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
               .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
       createTableRequest.setRetentionProperties(retentionProperties);

       try {
           amazonTimestreamWrite.createTable(createTableRequest);
           System.out.println("Table [" + TABLE_NAME + "] successfully created.");
       } catch (ConflictException e) {
           System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
       }
   }
```

Java v2

```
   public void createTable() {
       System.out.println("Creating table");

       final RetentionProperties retentionProperties =
RetentionProperties.builder()
               .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
               .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS).build();
       final CreateTableRequest createTableRequest = CreateTableRequest.builder()

.databaseName(DATABASE_NAME).tableName(TABLE_NAME).retentionProperties(retentionProperties)

       try {
           timestreamWriteClient.createTable(createTableRequest);
           System.out.println("Table [" + TABLE_NAME + "] successfully created.");
       } catch (ConflictException e) {
           System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
DATABASE_NAME + "] . Skipping database creation");
       }
   }
```

Go

```
// Create table.
    createTableInput := &timestreamwrite.CreateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    _, err = writeSvc.CreateTable(createTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Create table is successful")
    }
```

Python

```
    def create_table(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        try:
            self.client.create_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties)
            print("Table [%s] successfully created." % Constant.TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                Constant.TABLE_NAME, Constant.DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 CreateTableCommand 및 도 참조하세요CreateTable.

```
import { TimestreamWriteClient, CreateTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode",
    RetentionProperties: {
        MemoryStoreRetentionPeriodInHours: 24,
        MagneticStoreRetentionPeriodInDays: 365
    }
};

const command = new CreateTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Table ${data.Table.TableName} created successfully`);
} catch (error) {
    if (error.code === 'ConflictException') {
        console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
    } else {
        console.log("Error creating table. ", error);
        throw error;
    }
}
```

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream LiveAnalytics 의 샘플 애플리케이션을 기반으로 합니다 GitHub.

```
async function createTable() {
    console.log("Creating Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        }
    };

    const promise = writeClient.createTable(params).promise();
```

```
    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} created successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
            } else {
                console.log("Error creating table. ", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task CreateTable()
        {
            Console.WriteLine("Creating Table");

            try
            {
                var createTableRequest = new CreateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    }
                };
                CreateTableResponse response = await
 writeClient.CreateTableAsync(createTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Table already exists.");
            }
```

```
            catch (Exception e)
            {
                Console.WriteLine("Create table failed:" + e.ToString());
            }

        }
```

마그네틱 스토어 쓰기

다음 코드 조각을 사용하여 마그네틱 스토어 쓰기가 활성화된 테이블을 생성할 수 있습니다. 마그네틱 스토어 쓰기를 사용하면 메모리 스토어 보존 기간과 마그네틱 스토어 보존 기간 모두에 데이터를 쓸 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
   public void createTable(String databaseName, String tableName) {
       System.out.println("Creating table");
       CreateTableRequest createTableRequest = new CreateTableRequest();
       createTableRequest.setDatabaseName(databaseName);
       createTableRequest.setTableName(tableName);
       final RetentionProperties retentionProperties = new RetentionProperties()
               .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
               .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
       createTableRequest.setRetentionProperties(retentionProperties);
       // Enable MagneticStoreWrite
       final MagneticStoreWriteProperties magneticStoreWriteProperties = new
MagneticStoreWriteProperties()
               .withEnableMagneticStoreWrites(true);

createTableRequest.setMagneticStoreWriteProperties(magneticStoreWriteProperties);
       try {
           amazonTimestreamWrite.createTable(createTableRequest);
           System.out.println("Table [" + tableName + "] successfully created.");
       } catch (ConflictException e) {
```

```
            System.out.println("Table [" + tableName + "] exists on database [" +
 databaseName + "] . Skipping table creation");
            //We do not throw exception here, we use the existing table instead
        }
    }
```

Java v2

```
    public void createTable(String databaseName, String tableName) {
        System.out.println("Creating table");

        // Enable MagneticStoreWrite
        final MagneticStoreWriteProperties magneticStoreWriteProperties =
                MagneticStoreWriteProperties.builder()
                        .enableMagneticStoreWrites(true)
                        .build();

        CreateTableRequest createTableRequest =
                CreateTableRequest.builder()
                        .databaseName(databaseName)
                        .tableName(tableName)
                        .retentionProperties(RetentionProperties.builder()
                                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS)
                                .build())
                        .magneticStoreWriteProperties(magneticStoreWriteProperties)
                        .build();
        try {
            timestreamWriteClient.createTable(createTableRequest);
            System.out.println("Table [" + tableName + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + tableName + "] exists in database [" +
 databaseName + "] . Skipping table creation");
        }
    }
```

Go

```
// Create table.
    createTableInput := &timestreamwrite.CreateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    // Enable MagneticStoreWrite
```

```
        MagneticStoreWriteProperties: &timestreamwrite.MagneticStoreWriteProperties{
            EnableMagneticStoreWrites: aws.Bool(true),
             },
      }
    _, err = writeSvc.CreateTable(createTableInput)
```

Python

```
    def create_table(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        magnetic_store_write_properties = {
            'EnableMagneticStoreWrites': True
        }
        try:
            self.client.create_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties,

 MagneticStoreWriteProperties=magnetic_store_write_properties)
            print("Table [%s] successfully created." % Constant.TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                Constant.TABLE_NAME, Constant.DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

Node.js

```
async function createTable() {
    console.log("Creating Table");

    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        },
        MagneticStoreWriteProperties: {
```

```
            EnableMagneticStoreWrites: true
        }
    };

    const promise = writeClient.createTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} created successfully`);
        },
        (err) => {
            if (err.code === 'ConflictException') {
                console.log(`Table ${params.TableName} already exists on db
 ${params.DatabaseName}. Skipping creation.`);
            } else {
                console.log("Error creating table. ", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task CreateTable()
        {
            Console.WriteLine("Creating Table");

            try
            {
                var createTableRequest = new CreateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    },
                    // Enable MagneticStoreWrite
                    MagneticStoreWriteProperties = new MagneticStoreWriteProperties
                    {
                        EnableMagneticStoreWrites = true,
```

```
                    }
                };
                CreateTableResponse response = await
writeClient.CreateTableAsync(createTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} created");
            }
            catch (ConflictException)
            {
                Console.WriteLine("Table already exists.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Create table failed:" + e.ToString());
            }

        }
```

테이블 설명

다음 코드 조각을 사용하여 테이블의 속성에 대한 정보를 가져올 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
   public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest = new
DescribeTableRequest();
        describeTableRequest.setDatabaseName(DATABASE_NAME);
        describeTableRequest.setTableName(TABLE_NAME);
        try {
            DescribeTableResult result =
amazonTimestreamWrite.describeTable(describeTableRequest);
            String tableId = result.getTable().getArn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
        } catch (final Exception e) {
```

```
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

Java v2

```
    public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest =
 DescribeTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DescribeTableResponse response =
 timestreamWriteClient.describeTable(describeTableRequest);
            String tableId = response.table().arn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

Go

```
// Describe table.
    describeTableInput := &timestreamwrite.DescribeTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    describeTableOutput, err := writeSvc.DescribeTable(describeTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Describe table is successful, below is the output:")
        fmt.Println(describeTableOutput)
    }
```

Python

```
    def describe_table(self):
```

```
        print("Describing table")
        try:
            result = self.client.describe_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME)
            print("Table [%s] has id [%s]" % (Constant.TABLE_NAME, result['Table']
['Arn']))
        except self.client.exceptions.ResourceNotFoundException:
            print("Table doesn't exist")
        except Exception as err:
            print("Describe table failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 DescribeTableCommand 및 도 참조하세요DescribeTable.

```
import { TimestreamWriteClient, DescribeTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode"
};

const command = new DescribeTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Table ${data.Table.TableName} has id ${data.Table.Arn}`);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Table or Database doesn't exist.");
    } else {
        console.log("Describe table failed.", error);
        throw error;
    }
}
```

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function describeTable() {
    console.log("Describing Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME
    };

    const promise = writeClient.describeTable(params).promise();

    await promise.then(
        (data) => {
            console.log(`Table ${data.Table.TableName} has id ${data.Table.Arn}`);
        },
        (err) => {
            if (err.code === 'ResourceNotFoundException') {
                console.log("Table or Database doesn't exists.");
            } else {
                console.log("Describe table failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DescribeTable()
        {
            Console.WriteLine("Describing Table");

            try
            {
                var describeTableRequest = new DescribeTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME
                };
                DescribeTableResponse response = await
 writeClient.DescribeTableAsync(describeTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} has id:
{response.Table.Arn}");
            }
            catch (ResourceNotFoundException)
```

```
            {
                Console.WriteLine("Table does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe table failed:" + e.ToString());
            }

        }
```

테이블 업데이트

다음 코드 조각을 사용하여 테이블을 업데이트할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
    public void updateTable() {
        System.out.println("Updating table");
        UpdateTableRequest updateTableRequest = new UpdateTableRequest();
        updateTableRequest.setDatabaseName(DATABASE_NAME);
        updateTableRequest.setTableName(TABLE_NAME);

        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);

        updateTableRequest.setRetentionProperties(retentionProperties);

        amazonTimestreamWrite.updateTable(updateTableRequest);
        System.out.println("Table updated");
    }
```

Java v2

```
    public void updateTable() {
```

```
        System.out.println("Updating table");

        final RetentionProperties retentionProperties =
 RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS).build();
        final UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()

 .databaseName(DATABASE_NAME).tableName(TABLE_NAME).retentionProperties(retentionProperties)

        timestreamWriteClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
    }
```

Go

```
// Update table.
    magneticStoreRetentionPeriodInDays := int64(7 * 365)
    memoryStoreRetentionPeriodInHours := int64(24)

    updateTableInput := &timestreamwrite.UpdateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
        RetentionProperties: &timestreamwrite.RetentionProperties{
            MagneticStoreRetentionPeriodInDays: &magneticStoreRetentionPeriodInDays,
            MemoryStoreRetentionPeriodInHours:  &memoryStoreRetentionPeriodInHours,
        },
    }
    updateTableOutput, err := writeSvc.UpdateTable(updateTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Update table is successful, below is the output:")
        fmt.Println(updateTableOutput)
    }
```

Python

```
    def update_table(self):
        print("Updating table")
        retention_properties = {
```

```
            'MemoryStoreRetentionPeriodInHours': Constant.HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': Constant.CT_TTL_DAYS
        }
        try:
            self.client.update_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                                     RetentionProperties=retention_properties)
            print("Table updated.")
        except Exception as err:
            print("Update table failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 UpdateTableCommand 및 도 참조하세요UpdateTable.

```
import { TimestreamWriteClient, UpdateTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode",
    RetentionProperties: {
        MemoryStoreRetentionPeriodInHours: 24,
        MagneticStoreRetentionPeriodInDays: 180
    }
};

const command = new UpdateTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Table updated")
} catch (error) {
    console.log("Error updating table. ", error);
}
```

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function updateTable() {
    console.log("Updating Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME,
        RetentionProperties: {
            MemoryStoreRetentionPeriodInHours: constants.HT_TTL_HOURS,
            MagneticStoreRetentionPeriodInDays: constants.CT_TTL_DAYS
        }
    };

    const promise = writeClient.updateTable(params).promise();

    await promise.then(
        (data) => {
            console.log("Table updated")
        },
        (err) => {
            console.log("Error updating table. ", err);
            throw err;
        }
    );
}
```

.NET

```
        public async Task UpdateTable()
        {
            Console.WriteLine("Updating Table");

            try
            {
                var updateTableRequest = new UpdateTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME,
                    RetentionProperties = new RetentionProperties
                    {
                        MagneticStoreRetentionPeriodInDays = Constants.CT_TTL_DAYS,
                        MemoryStoreRetentionPeriodInHours = Constants.HT_TTL_HOURS
                    }
                };
```

```
                UpdateTableResponse response = await
writeClient.UpdateTableAsync(updateTableRequest);
                Console.WriteLine($"Table {Constants.TABLE_NAME} updated");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Table does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Update table failed:" + e.ToString());
            }

        }
```

테이블 삭제

다음 코드 조각을 사용하여 테이블을 삭제할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
    public void deleteTable() {
        System.out.println("Deleting table");
        final DeleteTableRequest deleteTableRequest = new DeleteTableRequest();
        deleteTableRequest.setDatabaseName(DATABASE_NAME);
        deleteTableRequest.setTableName(TABLE_NAME);
        try {
            DeleteTableResult result =
                    amazonTimestreamWrite.deleteTable(deleteTableRequest);
            System.out.println("Delete table status: " +
result.getSdkHttpMetadata().getHttpStatusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        } catch (final Exception e) {
```

```
            System.out.println("Could not delete table " + TABLE_NAME + " = " + e);
            throw e;
        }
    }
```

Java v2

```
    public void deleteTable() {
        System.out.println("Deleting table");
        final DeleteTableRequest deleteTableRequest = DeleteTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DeleteTableResponse response =
                    timestreamWriteClient.deleteTable(deleteTableRequest);
            System.out.println("Delete table status: " +
 response.sdkHttpResponse().statusCode());
        } catch (final ResourceNotFoundException e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        } catch (final Exception e) {
            System.out.println("Could not delete table " + TABLE_NAME + " = " + e);
            throw e;
        }
    }
```

Go

```
deleteTableInput := &timestreamwrite.DeleteTableInput{
        DatabaseName:   aws.String(*databaseName),
        TableName:    aws.String(*tableName),
    }
    _, err = writeSvc.DeleteTable(deleteTableInput)

    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    } else {
        fmt.Println("Table deleted", *tableName)
    }
```

Python

```
    def delete_table(self):
```

```
        print("Deleting Table")
        try:
            result = self.client.delete_table(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME)
            print("Delete table status [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
        except self.client.exceptions.ResourceNotFoundException:
            print("Table [%s] doesn't exist" % Constant.TABLE_NAME)
        except Exception as err:
            print("Delete table failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 DeleteTableCommand 및 도 참조하세요DeleteTable.

```
import { TimestreamWriteClient, DeleteTableCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    TableName: "testTableFromNode"
};

const command = new DeleteTableCommand(params);

try {
    const data = await writeClient.send(command);
    console.log("Deleted table");
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log(`Table ${params.TableName} or Database ${params.DatabaseName}
 doesn't exist.`);
    } else {
        console.log("Delete table failed.", error);
        throw error;
    }
}
```

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function deleteTable() {
    console.log("Deleting Table");
    const params = {
        DatabaseName: constants.DATABASE_NAME,
        TableName: constants.TABLE_NAME
    };

    const promise = writeClient.deleteTable(params).promise();

    await promise.then(
        function (data) {
            console.log("Deleted table");
        },
        function(err) {
            if (err.code === 'ResourceNotFoundException') {
                console.log(`Table ${params.TableName} or Database
 ${params.DatabaseName} doesn't exists.`);
            } else {
                console.log("Delete table failed.", err);
                throw err;
            }
        }
    );
}
```

.NET

```
        public async Task DeleteTable()
        {
            Console.WriteLine("Deleting table");
            try
            {
                var deleteTableRequest = new DeleteTableRequest
                {
                    DatabaseName = Constants.DATABASE_NAME,
                    TableName = Constants.TABLE_NAME
                };
                DeleteTableResponse response = await
 writeClient.DeleteTableAsync(deleteTableRequest);
```

```
                Console.WriteLine($"Table {Constants.TABLE_NAME} delete request
status: {response.HttpStatusCode}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine($"Table {Constants.TABLE_NAME} does not exists");
            }
            catch (Exception e)
            {
                Console.WriteLine("Exception while deleting table:" + e.ToString());
            }
        }
```

테이블 나열

다음 코드 조각을 사용하여 테이블을 나열할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
    public void listTables() {
        System.out.println("Listing tables");
        ListTablesRequest request = new ListTablesRequest();
        request.setDatabaseName(DATABASE_NAME);
        ListTablesResult result = amazonTimestreamWrite.listTables(request);
        printTables(result.getTables());

        String nextToken = result.getNextToken();
        while (nextToken != null && !nextToken.isEmpty()) {
            request.setNextToken(nextToken);
            ListTablesResult nextResult = amazonTimestreamWrite.listTables(request);

            printTables(nextResult.getTables());
            nextToken = nextResult.getNextToken();
        }
    }
```

```
    private void printTables(List<Table> tables) {
       for (Table table : tables) {
           System.out.println(table.getTableName());
       }
   }
```

Java v2

```
   public void listTables() {
       System.out.println("Listing tables");
       ListTablesRequest request =
ListTablesRequest.builder().databaseName(DATABASE_NAME).maxResults(2).build();
       ListTablesIterable listTablesIterable =
timestreamWriteClient.listTablesPaginator(request);
       for(ListTablesResponse listTablesResponse : listTablesIterable) {
           final List<Table> tables = listTablesResponse.tables();
           tables.forEach(table -> System.out.println(table.tableName()));
       }
   }
```

Go

```
listTablesMaxResult := int64(15)

   listTablesInput := &timestreamwrite.ListTablesInput{
       DatabaseName: aws.String(*databaseName),
       MaxResults:   &listTablesMaxResult,
   }
   listTablesOutput, err := writeSvc.ListTables(listTablesInput)

   if err != nil {
       fmt.Println("Error:")
       fmt.Println(err)
   } else {
       fmt.Println("List tables is successful, below is the output:")
       fmt.Println(listTablesOutput)
   }
```

Python

```
   def list_tables(self):
```

```
        print("Listing tables")
        try:
            result = self.client.list_tables(DatabaseName=Constant.DATABASE_NAME,
 MaxResults=5)
            self.__print_tables(result['Tables'])
            next_token = result.get('NextToken', None)
            while next_token:
                result =
 self.client.list_tables(DatabaseName=Constant.DATABASE_NAME,
                                                    NextToken=next_token, MaxResults=5)
                self.__print_tables(result['Tables'])
                next_token = result.get('NextToken', None)
        except Exception as err:
            print("List tables failed:", err)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

클래스 ListTablesCommand 및 도 참조하세요ListTables.

```
import { TimestreamWriteClient, ListTablesCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "us-east-1" });

const params = {
    DatabaseName: "testDbFromNode",
    MaxResults: 15
};

const command = new ListTablesCommand(params);

getTablesList(null);

async function getTablesList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.Tables.forEach(function (table) {
```

```
            console.log(table.TableName);
        });

        if (data.NextToken) {
            return getTablesList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing tables", error);
    }
}
```

다음 코드 조각은 JavaScript V2 스타일에 대해 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function listTables() {
    console.log("Listing tables:");
    const tables = await getTablesList(null);
    tables.forEach(function(table){
        console.log(table.TableName);
    });
}

function getTablesList(nextToken, tables = []) {
    var params = {
        DatabaseName: constants.DATABASE_NAME,
        MaxResults: 15
    };

    if(nextToken) {
        params.NextToken = nextToken;
    }

    return writeClient.listTables(params).promise()
        .then(
            (data) => {
                tables.push.apply(tables, data.Tables);
                if (data.NextToken) {
                    return getTablesList(data.NextToken, tables);
                } else {
                    return tables;
                }
            },
            (err) => {
```

```
                console.log("Error while listing databases", err);
            });
}
```

.NET

```
        public async Task ListTables()
        {
            Console.WriteLine("Listing Tables");

            try
            {
                var listTablesRequest = new ListTablesRequest
                {
                    MaxResults = 5,
                    DatabaseName = Constants.DATABASE_NAME
                };
                ListTablesResponse response = await
 writeClient.ListTablesAsync(listTablesRequest);
                PrintTables(response.Tables);
                string nextToken = response.NextToken;
                while (nextToken != null)
                {
                    listTablesRequest.NextToken = nextToken;
                    response = await writeClient.ListTablesAsync(listTablesRequest);
                    PrintTables(response.Tables);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List table failed:" + e.ToString());
            }

        }

        private void PrintTables(List<Table> tables)
        {
            foreach (Table table in tables)
                Console.WriteLine($"Table: {table.TableName}");
        }
```

데이터 쓰기(삽입 및 업서트)

주제

레코드 배치 작성

다음 코드 조각을 사용하여 Amazon Timestream 테이블에 데이터를 쓸 수 있습니다. 배치로 데이터를 작성하면 쓰기 비용을 최적화하는 데 도움이 됩니다. 자세한 내용은 쓰기 수 계산 섹션을 참조하세요.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
  public void writeRecords() {
    System.out.println("Writing records");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);
```

```
    Record cpuUtilization = new Record()
        .withDimensions(dimensions)
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));
    Record memoryUtilization = new Record()
        .withDimensions(dimensions)
        .withMeasureName("memory_utilization")
        .withMeasureValue("40")
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ":
"
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
  public void writeRecords() {
    System.out.println("Writing records");
```

```
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record cpuUtilization = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .measureName("cpu_utilization")
        .measureValue("13.5")
        .time(String.valueOf(time)).build();

    Record memoryUtilization = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .measureName("memory_utilization")
        .measureValue("40")
        .time(String.valueOf(time)).build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME).tableName(TABLE_NAME).records(records).build();

    try {
      WriteRecordsResponse writeRecordsResponse =
 timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println("WriteRecords Status: " +
 writeRecordsResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
```

```
            + rejectedRecord.reason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Go

```
now := time.Now()
currentTimeInSeconds := now.Unix()
writeRecordsInput := &timestreamwrite.WriteRecordsInput{
  DatabaseName: aws.String(*databaseName),
  TableName:  aws.String(*tableName),
  Records: []*timestreamwrite.Record{
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
        &timestreamwrite.Dimension{
          Name:  aws.String("region"),
          Value: aws.String("us-east-1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("az"),
          Value: aws.String("az1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("hostname"),
          Value: aws.String("host1"),
        },
      },
      MeasureName:    aws.String("cpu_utilization"),
      MeasureValue:   aws.String("13.5"),
      MeasureValueType: aws.String("DOUBLE"),
      Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
    },
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
        &timestreamwrite.Dimension{
          Name:  aws.String("region"),
          Value: aws.String("us-east-1"),
        },
```

```
        &timestreamwrite.Dimension{
          Name:  aws.String("az"),
          Value: aws.String("az1"),
        },
        &timestreamwrite.Dimension{
          Name:  aws.String("hostname"),
          Value: aws.String("host1"),
        },
      },
      MeasureName:    aws.String("memory_utilization"),
      MeasureValue:   aws.String("40"),
      MeasureValueType: aws.String("DOUBLE"),
      Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
    },
  },
}

_, err = writeSvc.WriteRecords(writeRecordsInput)

if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
} else {
  fmt.Println("Write records is successful")
}
```

Python

```
  def write_records(self):
    print("Writing records")
    current_time = self._current_milli_time()

    dimensions = [
      {'Name': 'region', 'Value': 'us-east-1'},
      {'Name': 'az', 'Value': 'az1'},
      {'Name': 'hostname', 'Value': 'host1'}
    ]

    cpu_utilization = {
      'Dimensions': dimensions,
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5',
```

```
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }

    memory_utilization = {
      'Dimensions': dimensions,
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40',
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }

    records = [cpu_utilization, memory_utilization]

    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes={})
      print("WriteRecords Status: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
      print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
      if "ExistingVersion" in rr:
        print("Rejected record existing version: ", rr["ExistingVersion"])

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function writeRecords() {
```

```
  console.log("Writing records");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const cpuUtilization = {
    'Dimensions': dimensions,
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5',
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const memoryUtilization = {
    'Dimensions': dimensions,
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40',
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records
  };

  const request = writeClient.writeRecords(params);

  await request.promise().then(
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
JSON.parse(request.response.httpResponse.body.toString());
```

```
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
}
```

.NET

```
  public async Task WriteRecords()
  {
    Console.WriteLine("Writing records");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

    List<Dimension> dimensions = new List<Dimension>{
      new Dimension { Name = "region", Value = "us-east-1" },
      new Dimension { Name = "az", Value = "az1" },
      new Dimension { Name = "hostname", Value = "host1" }
    };

    var cpuUtilization = new Record
    {
      Dimensions = dimensions,
      MeasureName = "cpu_utilization",
      MeasureValue = "13.6",
      MeasureValueType = MeasureValueType.DOUBLE,
      Time = currentTimeString
    };

    var memoryUtilization = new Record
    {
      Dimensions = dimensions,
      MeasureName = "memory_utilization",
      MeasureValue = "40",
      MeasureValueType = MeasureValueType.DOUBLE,
      Time = currentTimeString
    };


    List<Record> records = new List<Record> {
      cpuUtilization,
```

```
      memoryUtilization
    };

    try
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      Console.WriteLine("RejectedRecordsException:" + e.ToString());
      foreach (RejectedRecord rr in e.RejectedRecords) {
        Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
      }
      Console.WriteLine("Other records were written successfully. ");
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
  }
```

공통 속성이 있는 레코드 배치 작성

시계열 데이터에 여러 데이터 포인트에서 일반적인 측정값 및/또는 차원이 있는 경우 다음과 같은 최적화된 버전의 를 writeRecords API 사용하여 의 Timestream에 데이터를 삽입할 수도 있습니다 LiveAnalytics. 일괄 처리와 함께 공통 속성을 사용하면 에 설명된 대로 쓰기 비용을 추가로 최적화할 수 있습니다쓰기 수 계산.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
  public void writeRecordsWithCommonAttributes() {
    System.out.println("Writing records with extracting common attributes");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time));

    Record cpuUtilization = new Record()
        .withMeasureName("cpu_utilization")
        .withMeasureValue("13.5");
    Record memoryUtilization = new Record()
        .withMeasureName("memory_utilization")
        .withMeasureValue("40");

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsRequest.setRecords(records);

    try {
      WriteRecordsResult writeRecordsResult =
 amazonTimestreamWrite.writeRecords(writeRecordsRequest);
```

```
      System.out.println("writeRecordsWithCommonAttributes Status: " +
 writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ":
 "
            + rejectedRecord.getReason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
  public void writeRecordsWithCommonAttributes() {
    System.out.println("Writing records with extracting common attributes");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time)).build();

    Record cpuUtilization = Record.builder()
        .measureName("cpu_utilization")
        .measureValue("13.5").build();
    Record memoryUtilization = Record.builder()
```

```
        .measureName("memory_utilization")
        .measureValue("40").build();

    records.add(cpuUtilization);
    records.add(memoryUtilization);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    try {
      WriteRecordsResponse writeRecordsResponse =
 timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println("writeRecordsWithCommonAttributes Status: " +
 writeRecordsResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
            + rejectedRecord.reason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Go

```
now = time.Now()
currentTimeInSeconds = now.Unix()
writeRecordsCommonAttributesInput := &timestreamwrite.WriteRecordsInput{
 DatabaseName: aws.String(*databaseName),
 TableName:  aws.String(*tableName),
 CommonAttributes: &timestreamwrite.Record{
  Dimensions: []*timestreamwrite.Dimension{
   &timestreamwrite.Dimension{
    Name:  aws.String("region"),
    Value: aws.String("us-east-1"),
   },
   &timestreamwrite.Dimension{
```

```
    Name:  aws.String("az"),
    Value: aws.String("az1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("hostname"),
    Value: aws.String("host1"),
   },
  },
  MeasureValueType: aws.String("DOUBLE"),
  Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
  TimeUnit:     aws.String("SECONDS"),
 },
 Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
   MeasureName:  aws.String("cpu_utilization"),
   MeasureValue: aws.String("13.5"),
  },
  &timestreamwrite.Record{
   MeasureName:  aws.String("memory_utilization"),
   MeasureValue: aws.String("40"),
  },
 },
}

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Ingest records is successful")
}
```

Python

```
  def write_records_with_common_attributes(self):
    print("Writing records extracting common attributes")
    current_time = self._current_milli_time()

    dimensions = [
      {'Name': 'region', 'Value': 'us-east-1'},
      {'Name': 'az', 'Value': 'az1'},
      {'Name': 'hostname', 'Value': 'host1'}
```

```
    ]

    common_attributes = {
      'Dimensions': dimensions,
      'MeasureValueType': 'DOUBLE',
      'Time': current_time
    }

    cpu_utilization = {
      'MeasureName': 'cpu_utilization',
      'MeasureValue': '13.5'
    }

    memory_utilization = {
      'MeasureName': 'memory_utilization',
      'MeasureValue': '40'
    }

    records = [cpu_utilization, memory_utilization]

    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _print_rejected_records_exceptions(err):
    print("RejectedRecords: ", err)
    for rr in err.response["RejectedRecords"]:
      print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
      if "ExistingVersion" in rr:
        print("Rejected record existing version: ", rr["ExistingVersion"])

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function writeRecordsWithCommonAttributes() {
  console.log("Writing records with common attributes");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const commonAttributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString()
  };

  const cpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5'
  };

  const memoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40'
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records,
    CommonAttributes: commonAttributes
  };

  const request = writeClient.writeRecords(params);

  await request.promise().then(
    (data) => {
```

```
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
 JSON.parse(request.response.httpResponse.body.toString());
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
}
```

.NET

```
  public async Task WriteRecordsWithCommonAttributes()
  {
    Console.WriteLine("Writing records with common attributes");

    DateTimeOffset now = DateTimeOffset.UtcNow;
    string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

    List<Dimension> dimensions = new List<Dimension>{
      new Dimension { Name = "region", Value = "us-east-1" },
      new Dimension { Name = "az", Value = "az1" },
      new Dimension { Name = "hostname", Value = "host1" }
    };

    var commonAttributes = new Record
    {
      Dimensions = dimensions,
      MeasureValueType = MeasureValueType.DOUBLE,
      Time = currentTimeString
    };

    var cpuUtilization = new Record
    {
      MeasureName = "cpu_utilization",
      MeasureValue = "13.6"
    };

    var memoryUtilization = new Record
```

```
    {
      MeasureName = "memory_utilization",
      MeasureValue = "40"
    };


    List<Record> records = new List<Record>();
    records.Add(cpuUtilization);
    records.Add(memoryUtilization);

    try
    {
      var writeRecordsRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = records,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
      Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      Console.WriteLine("RejectedRecordsException:" + e.ToString());
      foreach (RejectedRecord rr in e.RejectedRecords) {
        Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
      }
      Console.WriteLine("Other records were written successfully. ");
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
  }
```

레코드 업서스팅

Amazon Timestream의 기본 쓰기는 첫 번째 라이터가 의미학을 얻고, 데이터는 추가로만 저장되고 중복 레코드는 거부되지만, 마지막 라이터가 의미학을 얻는 것을 사용하여 Amazon Timestream에 데이터를 쓸 수 있는 기능이 필요한 애플리케이션이 있으며, 이 경우 버전이 가장 높은 레코드가 시스템에

저장됩니다. 기존 레코드를 업데이트할 수 있는 기능이 필요한 애플리케이션도 있습니다. 이러한 시나리오를 해결하기 위해 Amazon Timestream은 데이터를 업서트할 수 있는 기능을 제공합니다. Upsert는 레코드가 없는 경우 시스템에 레코드를 삽입하거나 레코드가 있는 경우 레코드를 업데이트하는 작업입니다.

`WriteRecords` 요청을 보내는 동안 레코드 정의`Version`에 를 포함하여 레코드를 업서트할 수 있습니다. Amazon Timestream은 레코드가 가장 높은 와 함께 레코드를 저장합니다`Version`. 아래 코드 샘플은 데이터를 업서트하는 방법을 보여줍니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
  public void writeRecordsWithUpsert() {
    System.out.println("Writing records with upsert");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue("us-
east-1");
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withMeasureValueType(MeasureValueType.DOUBLE)
        .withTime(String.valueOf(time))
        .withVersion(version);
```

```
   Record cpuUtilization = new Record()
       .withMeasureName("cpu_utilization")
       .withMeasureValue("13.5");
   Record memoryUtilization = new Record()
       .withMeasureName("memory_utilization")
       .withMeasureValue("40");

   records.add(cpuUtilization);
   records.add(memoryUtilization);

   WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
       .withDatabaseName(DATABASE_NAME)
       .withTableName(TABLE_NAME)
       .withCommonAttributes(commonAttributes);
   writeRecordsRequest.setRecords(records);

   // write records for first time
   try {
     WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for first time: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
   try {
     WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for retry: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // upsert with lower version, this would fail because a higher version is
required to update the measure value.
```

```
    version -= 1;
    commonAttributes.setVersion(version);

    cpuUtilization.setMeasureValue("14.5");
    memoryUtilization.setMeasureValue("50");

    List<Record> upsertedRecords = new ArrayList<>();
    upsertedRecords.add(cpuUtilization);
    upsertedRecords.add(memoryUtilization);

    WriteRecordsRequest writeRecordsUpsertRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsUpsertRequest.setRecords(upsertedRecords);

    try {
      WriteRecordsResult writeRecordsUpsertResult =
amazonTimestreamWrite.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with lower version: " +
writeRecordsUpsertResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("WriteRecords Status for upsert with lower version: ");
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // upsert with higher version as new data in generated
    version = System.currentTimeMillis();
    commonAttributes.setVersion(version);

    writeRecordsUpsertRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes);
    writeRecordsUpsertRequest.setRecords(upsertedRecords);

    try {
      WriteRecordsResult writeRecordsUpsertResult =
amazonTimestreamWrite.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with higher version: " +
writeRecordsUpsertResult.getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
```

```
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Java v2

```
  public void writeRecordsWithUpsert() {
    System.out.println("Writing records with upsert");
    // Specify repeated values for all records
    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = Dimension.builder().name("region").value("us-
east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
 Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time))
        .version(version)
        .build();

    Record cpuUtilization = Record.builder()
        .measureName("cpu_utilization")
        .measureValue("13.5").build();
    Record memoryUtilization = Record.builder()
        .measureName("memory_utilization")
        .measureValue("40").build();

    records.add(cpuUtilization);
```

```
   records.add(memoryUtilization);

   WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
       .databaseName(DATABASE_NAME)
       .tableName(TABLE_NAME)
       .commonAttributes(commonAttributes)
       .records(records).build();

   // write records for first time
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for first time: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("WriteRecords Status for retry: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
   } catch (RejectedRecordsException e) {
     printRejectedRecordsException(e);
   } catch (Exception e) {
     System.out.println("Error: " + e);
   }

   // upsert with lower version, this would fail because a higher version is
required to update the measure value.
   version -= 1;
   commonAttributes = Record.builder()
       .dimensions(dimensions)
       .measureValueType(MeasureValueType.DOUBLE)
       .time(String.valueOf(time))
       .version(version)
       .build();

   cpuUtilization = Record.builder()
```

```
        .measureName("cpu_utilization")
        .measureValue("14.5").build();
    memoryUtilization = Record.builder()
        .measureName("memory_utilization")
        .measureValue("50").build();

    List<Record> upsertedRecords = new ArrayList<>();
    upsertedRecords.add(cpuUtilization);
    upsertedRecords.add(memoryUtilization);

    WriteRecordsRequest writeRecordsUpsertRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(upsertedRecords).build();

    try {
      WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with lower version: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      System.out.println("WriteRecords Status for upsert with lower version: ");
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }

    // upsert with higher version as new data in generated
    version = System.currentTimeMillis();
    commonAttributes = Record.builder()
        .dimensions(dimensions)
        .measureValueType(MeasureValueType.DOUBLE)
        .time(String.valueOf(time))
        .version(version)
        .build();

    writeRecordsUpsertRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(upsertedRecords).build();

    try {
```

```
      WriteRecordsResponse writeRecordsUpsertResponse =
 timestreamWriteClient.writeRecords(writeRecordsUpsertRequest);
      System.out.println("WriteRecords Status for upsert with higher version: " +
 writeRecordsUpsertResponse.sdkHttpResponse().statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

Go

```
// Below code will ingest and upsert cpu_utilization and memory_utilization metric
 for a host on
// region=us-east-1, az=az1, and hostname=host1
fmt.Println("Ingesting records and set version as currentTimeInMills, hit enter to
 continue")
reader.ReadString('\n')

// Get current time in seconds.
now = time.Now()
currentTimeInSeconds = now.Unix()
// To achieve upsert (last writer wins) semantic, one example is to use current time
 as the version if you are writing directly from the data source
version := time.Now().Round(time.Millisecond).UnixNano() / 1e6   // set version as
 currentTimeInMills

writeRecordsCommonAttributesUpsertInput := &timestreamwrite.WriteRecordsInput{
 DatabaseName: aws.String(*databaseName),
 TableName:  aws.String(*tableName),
 CommonAttributes: &timestreamwrite.Record{
  Dimensions: []*timestreamwrite.Dimension{
   &timestreamwrite.Dimension{
    Name:  aws.String("region"),
    Value: aws.String("us-east-1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("az"),
    Value: aws.String("az1"),
   },
   &timestreamwrite.Dimension{
    Name:  aws.String("hostname"),
```

```
    Value: aws.String("host1"),
   },
  },
  MeasureValueType: aws.String("DOUBLE"),
  Time:       aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
  TimeUnit:  aws.String("SECONDS"),
  Version:      &version,
 },
 Records: []*timestreamwrite.Record{
  &timestreamwrite.Record{
   MeasureName:  aws.String("cpu_utilization"),
   MeasureValue: aws.String("13.5"),
  },
  &timestreamwrite.Record{
   MeasureName:  aws.String("memory_utilization"),
   MeasureValue: aws.String("40"),
  },
 },
}

// write records for first time
_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Frist-time write records is successful")
}

fmt.Println("Retry same writeRecordsRequest with same records and versions. Because
 writeRecords API is idempotent, this will success. hit enter to continue")
reader.ReadString('\n')

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Retry write records for same request is successful")
}
```

```
fmt.Println("Upsert with lower version, this would fail because a higher version is
 required to update the measure value. hit enter to continue")
reader.ReadString('\n')
version -= 1
writeRecordsCommonAttributesUpsertInput.CommonAttributes.Version = &version

updated_cpu_utilization := &timestreamwrite.Record{
 MeasureName:    aws.String("cpu_utilization"),
 MeasureValue:   aws.String("14.5"),
}
updated_memory_utilization := &timestreamwrite.Record{
 MeasureName:    aws.String("memory_utilization"),
 MeasureValue:   aws.String("50"),
}


writeRecordsCommonAttributesUpsertInput.Records = []*timestreamwrite.Record{
 updated_cpu_utilization,
 updated_memory_utilization,
}

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
 fmt.Println("Write records with lower version is successful")
}

fmt.Println("Upsert with higher version as new data in generated, this would
 success. hit enter to continue")
reader.ReadString('\n')

version = time.Now().Round(time.Millisecond).UnixNano() / 1e6  // set version as
 currentTimeInMills
writeRecordsCommonAttributesUpsertInput.CommonAttributes.Version = &version

_, err = writeSvc.WriteRecords(writeRecordsCommonAttributesUpsertInput)

if err != nil {
 fmt.Println("Error:")
 fmt.Println(err)
} else {
```

```
 fmt.Println("Write records with higher version is successful")
}
```

Python

```
 def write_records_with_upsert(self):
   print("Writing records with upsert")
   current_time = self._current_milli_time()
   # To achieve upsert (last writer wins) semantic, one example is to use current
time as the version if you are writing directly from the data source
   version = int(self._current_milli_time())

   dimensions = [
         {'Name': 'region', 'Value': 'us-east-1'},
         {'Name': 'az', 'Value': 'az1'},
         {'Name': 'hostname', 'Value': 'host1'}
       ]

   common_attributes = {
     'Dimensions': dimensions,
     'MeasureValueType': 'DOUBLE',
     'Time': current_time,
     'Version': version
   }

   cpu_utilization = {
     'MeasureName': 'cpu_utilization',
     'MeasureValue': '13.5'
   }

   memory_utilization = {
     'MeasureName': 'memory_utilization',
     'MeasureValue': '40'
   }

   records = [cpu_utilization, memory_utilization]

   # write records for first time
   try:
     result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes=common_attributes)
```

```
      print("WriteRecords Status for first time: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

    # Successfully retry same writeRecordsRequest with same records and versions,
 because writeRecords API is idempotent.
    try:
      result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                         Records=records, CommonAttributes=common_attributes)
      print("WriteRecords Status for retry: [%s]" % result['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

    # upsert with lower version, this would fail because a higher version is
 required to update the measure value.
    version -= 1
    common_attributes["Version"] = version

    cpu_utilization["MeasureValue"] = '14.5'
    memory_utilization["MeasureValue"] = '50'

    upsertedRecords = [cpu_utilization, memory_utilization]

    try:
      upsertedResult =
 self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                            Records=upsertedRecords,
 CommonAttributes=common_attributes)
      print("WriteRecords Status for upsert with lower version: [%s]" %
 upsertedResult['ResponseMetadata']['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)
```

```
    # upsert with higher version as new data is generated
    version = int(self._current_milli_time())
    common_attributes["Version"] = version

    try:
      upsertedResult =
 self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME,
                           Records=upsertedRecords,
 CommonAttributes=common_attributes)
      print("WriteRecords Upsert Status: [%s]" % upsertedResult['ResponseMetadata']
['HTTPStatusCode'])
    except self.client.exceptions.RejectedRecordsException as err:
      self._print_rejected_records_exceptions(err)
    except Exception as err:
      print("Error:", err)

  @staticmethod
  def _current_milli_time():
    return str(int(round(time.time() * 1000)))
```

Node.js

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function writeRecordsWithUpsert() {
  console.log("Writing records with upsert");
  const currentTime = Date.now().toString(); // Unix time in milliseconds
  // To achieve upsert (last writer wins) semantic, one example is to use current
 time as the version if you are writing directly from the data source
  let version = Date.now();

  const dimensions = [
    {'Name': 'region', 'Value': 'us-east-1'},
    {'Name': 'az', 'Value': 'az1'},
    {'Name': 'hostname', 'Value': 'host1'}
  ];

  const commonAttributes = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
```

```
    'Version': version
  };

  const cpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '13.5'
  };

  const memoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '40'
  };

  const records = [cpuUtilization, memoryUtilization];

  const params = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: records,
    CommonAttributes: commonAttributes
  };

  const request = writeClient.writeRecords(params);

  // write records for first time
  await request.promise().then(
    (data) => {
      console.log("Write records successful for first time.");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(request);
      }
    }
  );

  // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
  await request.promise().then(
    (data) => {
      console.log("Write records successful for retry.");
    },
    (err) => {
```

```
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(request);
      }
    }
  );

  // upsert with lower version, this would fail because a higher version is required
to update the measure value.
  version--;

  const commonAttributesWithLowerVersion = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
    'Version': version
  };

  const updatedCpuUtilization = {
    'MeasureName': 'cpu_utilization',
    'MeasureValue': '14.5'
  };

  const updatedMemoryUtilization = {
    'MeasureName': 'memory_utilization',
    'MeasureValue': '50'
  };

  const upsertedRecords = [updatedCpuUtilization, updatedMemoryUtilization];

  const upsertedParamsWithLowerVersion = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: upsertedRecords,
    CommonAttributes: commonAttributesWithLowerVersion
  };

  const upsertRequestWithLowerVersion =
writeClient.writeRecords(upsertedParamsWithLowerVersion);

  await upsertRequestWithLowerVersion.promise().then(
    (data) => {
      console.log("Write records for upsert with lower version successful");
    },
```

```
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(upsertRequestWithLowerVersion);
      }
    }
  );

  // upsert with higher version as new data in generated
  version = Date.now();

  const commonAttributesWithHigherVersion = {
    'Dimensions': dimensions,
    'MeasureValueType': 'DOUBLE',
    'Time': currentTime.toString(),
    'Version': version
  };

  const upsertedParamsWithHigherVerion = {
    DatabaseName: constants.DATABASE_NAME,
    TableName: constants.TABLE_NAME,
    Records: upsertedRecords,
    CommonAttributes: commonAttributesWithHigherVersion
  };

  const upsertRequestWithHigherVersion =
 writeClient.writeRecords(upsertedParamsWithHigherVerion);

  await upsertRequestWithHigherVersion.promise().then(
    (data) => {
      console.log("Write records upsert successful with higher version");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        printRejectedRecordsException(upsertedParamsWithHigherVerion);
      }
    }
  );

}
```

.NET

```
 public async Task WriteRecordsWithUpsert()
 {
   Console.WriteLine("Writing records with upsert");

   DateTimeOffset now = DateTimeOffset.UtcNow;
   string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();
   // To achieve upsert (last writer wins) semantic, one example is to use current
time as the version if you are writing directly from the data source
   long version = now.ToUnixTimeMilliseconds();

   List<Dimension> dimensions = new List<Dimension>{
     new Dimension { Name = "region", Value = "us-east-1" },
     new Dimension { Name = "az", Value = "az1" },
     new Dimension { Name = "hostname", Value = "host1" }
   };

   var commonAttributes = new Record
   {
     Dimensions = dimensions,
     MeasureValueType = MeasureValueType.DOUBLE,
     Time = currentTimeString,
     Version = version
   };

   var cpuUtilization = new Record
   {
     MeasureName = "cpu_utilization",
     MeasureValue = "13.6"
   };

   var memoryUtilization = new Record
   {
     MeasureName = "memory_utilization",
     MeasureValue = "40"
   };


   List<Record> records = new List<Record>();
   records.Add(cpuUtilization);
   records.Add(memoryUtilization);

   // write records for first time
```

```
   try
   {
     var writeRecordsRequest = new WriteRecordsRequest
     {
       DatabaseName = Constants.DATABASE_NAME,
       TableName = Constants.TABLE_NAME,
       Records = records,
       CommonAttributes = commonAttributes
     };
     WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
     Console.WriteLine($"WriteRecords Status for first time:
{response.HttpStatusCode.ToString()}");
   }
   catch (RejectedRecordsException e) {
     PrintRejectedRecordsException(e);
   }
   catch (Exception e)
   {
     Console.WriteLine("Write records failure:" + e.ToString());
   }

   // Successfully retry same writeRecordsRequest with same records and versions,
because writeRecords API is idempotent.
   try
   {
     var writeRecordsRequest = new WriteRecordsRequest
     {
       DatabaseName = Constants.DATABASE_NAME,
       TableName = Constants.TABLE_NAME,
       Records = records,
       CommonAttributes = commonAttributes
     };
     WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
     Console.WriteLine($"WriteRecords Status for retry:
{response.HttpStatusCode.ToString()}");
   }
   catch (RejectedRecordsException e) {
     PrintRejectedRecordsException(e);
   }
   catch (Exception e)
   {
     Console.WriteLine("Write records failure:" + e.ToString());
```

```
    }

    // upsert with lower version, this would fail because a higher version is
 required to update the measure value.
    version--;
    Type recordType = typeof(Record);
    recordType.GetProperty("Version").SetValue(commonAttributes, version);
    recordType.GetProperty("MeasureValue").SetValue(cpuUtilization, "14.6");
    recordType.GetProperty("MeasureValue").SetValue(memoryUtilization, "50");

    List<Record> upsertedRecords = new List<Record> {
      cpuUtilization,
      memoryUtilization
    };

    try
    {
      var writeRecordsUpsertRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = upsertedRecords,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse upsertResponse = await
 writeClient.WriteRecordsAsync(writeRecordsUpsertRequest);
      Console.WriteLine($"WriteRecords Status for upsert with lower version:
 {upsertResponse.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }

    // upsert with higher version as new data in generated
    now = DateTimeOffset.UtcNow;
    version = now.ToUnixTimeMilliseconds();
    recordType.GetProperty("Version").SetValue(commonAttributes, version);

    try
    {
```

```
      var writeRecordsUpsertRequest = new WriteRecordsRequest
      {
        DatabaseName = Constants.DATABASE_NAME,
        TableName = Constants.TABLE_NAME,
        Records = upsertedRecords,
        CommonAttributes = commonAttributes
      };
      WriteRecordsResponse upsertResponse = await
writeClient.WriteRecordsAsync(writeRecordsUpsertRequest);
      Console.WriteLine($"WriteRecords Status for upsert with higher version:
{upsertResponse.HttpStatusCode.ToString()}");
    }
    catch (RejectedRecordsException e) {
      PrintRejectedRecordsException(e);
    }
    catch (Exception e)
    {
      Console.WriteLine("Write records failure:" + e.ToString());
    }
  }
```

다중 측정 속성 예제

이 예제에서는 다중 mearure 속성 작성을 보여줍니다. 추적 중인 디바이스 또는 애플리케이션이 동일한 타임스탬프에서 여러 지표 또는 이벤트를 내보내는 경우 다중 측정 속성이 유용합니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
package com.amazonaws.services.timestream;

import static com.amazonaws.services.timestream.Main.DATABASE_NAME;
import static com.amazonaws.services.timestream.Main.REGION;
import static com.amazonaws.services.timestream.Main.TABLE_NAME;

import java.util.ArrayList;
```

```
import java.util.List;

import com.amazonaws.services.timestreamwrite.AmazonTimestreamWrite;
import com.amazonaws.services.timestreamwrite.model.Dimension;
import com.amazonaws.services.timestreamwrite.model.MeasureValue;
import com.amazonaws.services.timestreamwrite.model.MeasureValueType;
import com.amazonaws.services.timestreamwrite.model.Record;
import com.amazonaws.services.timestreamwrite.model.RejectedRecordsException;
import com.amazonaws.services.timestreamwrite.model.WriteRecordsRequest;
import com.amazonaws.services.timestreamwrite.model.WriteRecordsResult;


public class multimeasureAttributeExample {
  AmazonTimestreamWrite timestreamWriteClient;

  public multimeasureAttributeExample(AmazonTimestreamWrite client) {
    this.timestreamWriteClient = client;
  }

  public void writeRecordsMultiMeasureValueSingleRecord() {
    System.out.println("Writing records with multi value attributes");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue(REGION);
    final Dimension az = new Dimension().withName("az").withValue("az1");
    final Dimension hostname = new
 Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withTime(String.valueOf(time))
        .withVersion(version);

    MeasureValue cpuUtilization = new MeasureValue()
        .withName("cpu_utilization")
        .withType(MeasureValueType.DOUBLE)
```

```
        .withValue("13.5");
    MeasureValue memoryUtilization = new MeasureValue()
        .withName("memory_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("40");
    Record computationalResources = new Record()
        .withMeasureName("cpu_memory")
        .withMeasureValues(cpuUtilization, memoryUtilization)
        .withMeasureValueType(MeasureValueType.MULTI);

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes)
        .withRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordResult =
timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordResult
              .getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  public void writeRecordsMultiMeasureValueMultipleRecords() {
    System.out.println(
        "Writing records with multi value attributes mixture type");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region = new Dimension().withName("region").withValue(REGION);
    final Dimension az = new Dimension().withName("az").withValue("az1");
```

```
    final Dimension hostname = new
Dimension().withName("hostname").withValue("host1");

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = new Record()
        .withDimensions(dimensions)
        .withTime(String.valueOf(time))
        .withVersion(version);

    MeasureValue cpuUtilization = new MeasureValue()
        .withName("cpu_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("13");
    MeasureValue memoryUtilization =new MeasureValue()
        .withName("memory_utilization")
        .withType(MeasureValueType.DOUBLE)
        .withValue("40");
    MeasureValue activeCores = new MeasureValue()
        .withName("active_cores")
        .withType(MeasureValueType.BIGINT)
        .withValue("4");


    Record computationalResources = new Record()
        .withMeasureName("computational_utilization")
        .withMeasureValues(cpuUtilization, memoryUtilization, activeCores)
        .withMeasureValueType(MeasureValueType.MULTI);

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = new WriteRecordsRequest()
        .withDatabaseName(DATABASE_NAME)
        .withTableName(TABLE_NAME)
        .withCommonAttributes(commonAttributes)
        .withRecords(records);

    // write records for first time
    try {
      WriteRecordsResult writeRecordResult =
timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
```

```
          "WriteRecords Status for multi value attributes: " + writeRecordResult
              .getSdkHttpMetadata().getHttpStatusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  private void printRejectedRecordsException(RejectedRecordsException e) {
    System.out.println("RejectedRecords: " + e);
    e.getRejectedRecords().forEach(System.out::println);
  }
}
```

Java v2

```
package com.amazonaws.services.timestream;

import java.util.ArrayList;
import java.util.List;

import software.amazon.awssdk.services.timestreamwrite.TimestreamWriteClient;
import software.amazon.awssdk.services.timestreamwrite.model.Dimension;
import software.amazon.awssdk.services.timestreamwrite.model.MeasureValue;
import software.amazon.awssdk.services.timestreamwrite.model.MeasureValueType;
import software.amazon.awssdk.services.timestreamwrite.model.Record;
import
 software.amazon.awssdk.services.timestreamwrite.model.RejectedRecordsException;
import software.amazon.awssdk.services.timestreamwrite.model.WriteRecordsRequest;
import software.amazon.awssdk.services.timestreamwrite.model.WriteRecordsResponse;

import static com.amazonaws.services.timestream.Main.DATABASE_NAME;
import static com.amazonaws.services.timestream.Main.TABLE_NAME;


public class multimeasureAttributeExample {

  TimestreamWriteClient timestreamWriteClient;

  public multimeasureAttributeExample(TimestreamWriteClient client) {
    this.timestreamWriteClient = client;
  }
```

```
public void writeRecordsMultiMeasureValueSingleRecord() {
  System.out.println("Writing records with multi value attributes");

  List<Record> records = new ArrayList<>();
  final long time = System.currentTimeMillis();
  long version = System.currentTimeMillis();

  List<Dimension> dimensions = new ArrayList<>();
  final Dimension region =
      Dimension.builder().name("region").value("us-east-1").build();
  final Dimension az = Dimension.builder().name("az").value("az1").build();
  final Dimension hostname =
      Dimension.builder().name("hostname").value("host1").build();

  dimensions.add(region);
  dimensions.add(az);
  dimensions.add(hostname);

  Record commonAttributes = Record.builder()
      .dimensions(dimensions)
      .time(String.valueOf(time))
      .version(version)
      .build();

  MeasureValue cpuUtilization = MeasureValue.builder()
      .name("cpu_utilization")
      .type(MeasureValueType.DOUBLE)
      .value("13.5").build();
  MeasureValue memoryUtilization = MeasureValue.builder()
      .name("memory_utilization")
      .type(MeasureValueType.DOUBLE)
      .value("40").build();
  Record computationalResources = Record
      .builder()
      .measureName("cpu_memory")
      .measureValues(cpuUtilization, memoryUtilization)
      .measureValueType(MeasureValueType.MULTI)
      .build();

  records.add(computationalResources);

  WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
      .databaseName(DATABASE_NAME)
```

```
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    // write records for first time
    try {
      WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
      System.out.println(
          "WriteRecords Status for multi value attributes: " + writeRecordsResponse
              .sdkHttpResponse()
              .statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }

  public void writeRecordsMultiMeasureValueMultipleRecords() {
    System.out.println(
        "Writing records with multi value attributes mixture type");

    List<Record> records = new ArrayList<>();
    final long time = System.currentTimeMillis();
    long version = System.currentTimeMillis();

    List<Dimension> dimensions = new ArrayList<>();
    final Dimension region =
        Dimension.builder().name("region").value("us-east-1").build();
    final Dimension az = Dimension.builder().name("az").value("az1").build();
    final Dimension hostname =
        Dimension.builder().name("hostname").value("host1").build();

    dimensions.add(region);
    dimensions.add(az);
    dimensions.add(hostname);

    Record commonAttributes = Record.builder()
        .dimensions(dimensions)
        .time(String.valueOf(time))
        .version(version)
        .build();
```

```
    MeasureValue cpuUtilization = MeasureValue.builder()
        .name("cpu_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("13.5").build();
    MeasureValue memoryUtilization = MeasureValue.builder()
        .name("memory_utilization")
        .type(MeasureValueType.DOUBLE)
        .value("40").build();
    MeasureValue activeCores = MeasureValue.builder()
        .name("active_cores")
        .type(MeasureValueType.BIGINT)
        .value("4").build();


    Record computationalResources = Record
        .builder()
        .measureName("computational_utilization")
        .measureValues(cpuUtilization, memoryUtilization, activeCores)
        .measureValueType(MeasureValueType.MULTI)
        .build();

    records.add(computationalResources);

    WriteRecordsRequest writeRecordsRequest = WriteRecordsRequest.builder()
        .databaseName(DATABASE_NAME)
        .tableName(TABLE_NAME)
        .commonAttributes(commonAttributes)
        .records(records).build();

    // write records for first time
    try {
      WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println(
         "WriteRecords Status for multi value attributes: " + writeRecordsResponse
             .sdkHttpResponse()
             .statusCode());
    } catch (RejectedRecordsException e) {
      printRejectedRecordsException(e);
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
  }
```

```
  private void printRejectedRecordsException(RejectedRecordsException e) {
    System.out.println("RejectedRecords: " + e);
    e.rejectedRecords().forEach(System.out::println);
  }
}
```

Go

```
  now := time.Now()
  currentTimeInSeconds := now.Unix()
  writeRecordsInput := &timestreamwrite.WriteRecordsInput{
    DatabaseName: aws.String(*databaseName),
    TableName:  aws.String(*tableName),
    Records: []*timestreamwrite.Record{
    &timestreamwrite.Record{
      Dimensions: []*timestreamwrite.Dimension{
      &timestreamwrite.Dimension{
        Name:  aws.String("region"),
        Value: aws.String("us-east-1"),
      },
      &timestreamwrite.Dimension{
        Name:  aws.String("az"),
        Value: aws.String("az1"),
      },
      &timestreamwrite.Dimension{
        Name:  aws.String("hostname"),
        Value: aws.String("host1"),
      },
      },
      MeasureName:  aws.String("metrics"),
      MeasureValueType: aws.String("MULTI"),
      Time:     aws.String(strconv.FormatInt(currentTimeInSeconds, 10)),
      TimeUnit:  aws.String("SECONDS"),
      MeasureValues: []*timestreamwrite.MeasureValue{
      &timestreamwrite.MeasureValue{
        Name:  aws.String("cpu_utilization"),
        Value: aws.String("13.5"),
        Type:  aws.String("DOUBLE"),
      },
      &timestreamwrite.MeasureValue{
        Name:  aws.String("memory_utilization"),
        Value: aws.String("40"),
        Type:  aws.String("DOUBLE"),
```

```
      },
      },
    },
    },
  }

  _, err = writeSvc.WriteRecords(writeRecordsInput)

  if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
  } else {
    fmt.Println("Write records is successful")
  }
```

Python

```
import time
import boto3
import psutil
import os

from botocore.config import Config

DATABASE_NAME = os.environ['DATABASE_NAME']
TABLE_NAME = os.environ['TABLE_NAME']

COUNTRY = "UK"
CITY = "London"
HOSTNAME = "MyHostname" # You can make it dynamic using socket.gethostname()

INTERVAL = 1 # Seconds

def prepare_common_attributes():
  common_attributes = {
    'Dimensions': [
      {'Name': 'country', 'Value': COUNTRY},
      {'Name': 'city', 'Value': CITY},
      {'Name': 'hostname', 'Value': HOSTNAME}
    ],
    'MeasureName': 'utilization',
    'MeasureValueType': 'MULTI'
  }
```

```
  return common_attributes


def prepare_record(current_time):
  record = {
    'Time': str(current_time),
    'MeasureValues': []
  }
  return record


def prepare_measure(measure_name, measure_value):
  measure = {
    'Name': measure_name,
    'Value': str(measure_value),
    'Type': 'DOUBLE'
  }
  return measure


def write_records(records, common_attributes):
  try:
    result = write_client.write_records(DatabaseName=DATABASE_NAME,
                                        TableName=TABLE_NAME,
                                        CommonAttributes=common_attributes,
                                        Records=records)
    status = result['ResponseMetadata']['HTTPStatusCode']
    print("Processed %d records. WriteRecords HTTPStatusCode: %s" %
        (len(records), status))
  except Exception as err:
    print("Error:", err)


if __name__ == '__main__':

  print("writing data to database {} table {}".format(
    DATABASE_NAME, TABLE_NAME))

  session = boto3.Session()
  write_client = session.client('timestream-write', config=Config(
    read_timeout=20, max_pool_connections=5000, retries={'max_attempts': 10}))
  query_client = session.client('timestream-query') # Not used

  common_attributes = prepare_common_attributes()
```

```
records = []

while True:

  current_time = int(time.time() * 1000)
  cpu_utilization = psutil.cpu_percent()
  memory_utilization = psutil.virtual_memory().percent
  swap_utilization = psutil.swap_memory().percent
  disk_utilization = psutil.disk_usage('/').percent

  record = prepare_record(current_time)
  record['MeasureValues'].append(prepare_measure('cpu', cpu_utilization))
  record['MeasureValues'].append(prepare_measure('memory', memory_utilization))
  record['MeasureValues'].append(prepare_measure('swap', swap_utilization))
  record['MeasureValues'].append(prepare_measure('disk', disk_utilization))

  records.append(record)

  print("records {} - cpu {} - memory {} - swap {} - disk {}".format(
    len(records), cpu_utilization, memory_utilization,
    swap_utilization, disk_utilization))

  if len(records) == 100:
    write_records(records, common_attributes)
    records = []

  time.sleep(INTERVAL)
```

Node.js

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
async function writeRecords() {
  console.log("Writing records");
  const currentTime = Date.now().toString(); // Unix time in milliseconds

  const dimensions = [
  {'Name': 'region', 'Value': 'us-east-1'},
  {'Name': 'az', 'Value': 'az1'},
  {'Name': 'hostname', 'Value': 'host1'}
  ];
```

```
    const record = {
    'Dimensions': dimensions,
    'MeasureName': 'metrics',
    'MeasureValues': [
      {
        'Name': 'cpu_utilization',
        'Value': '40',
        'Type': 'DOUBLE',
      },
      {
        'Name': 'memory_utilization',
        'Value': '13.5',
        'Type': 'DOUBLE',
      },
      ],
      'MeasureValueType': 'MULTI',
      'Time': currentTime.toString()
    }

    const records = [record];

    const params = {
    DatabaseName: 'DatabaseName',
    TableName: 'TableName',
    Records: records
    };

    const response = await writeClient.writeRecords(params);

    console.log(response);
  }
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
```

```
{
  static class MultiMeasureValueConstants
  {
    public const string MultiMeasureValueSampleDb = "multiMeasureValueSampleDb";
    public const string MultiMeasureValueSampleTable =
 "multiMeasureValueSampleTable";
  }

  public class MultiValueAttributesExample
  {
    private readonly AmazonTimestreamWriteClient writeClient;

    public MultiValueAttributesExample(AmazonTimestreamWriteClient writeClient)
    {
      this.writeClient = writeClient;
    }

    public async Task WriteRecordsMultiMeasureValueSingleRecord()
    {
      Console.WriteLine("Writing records with multi value attributes");

      DateTimeOffset now = DateTimeOffset.UtcNow;
      string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();

      List<Dimension> dimensions = new List<Dimension>{
        new Dimension { Name = "region", Value = "us-east-1" },
        new Dimension { Name = "az", Value = "az1" },
        new Dimension { Name = "hostname", Value = "host1" }
      };

      var commonAttributes = new Record
      {
        Dimensions = dimensions,
        Time = currentTimeString
      };

      var cpuUtilization = new MeasureValue
      {
        Name = "cpu_utilization",
        Value = "13.6",
        Type = "DOUBLE"
      };

      var memoryUtilization = new MeasureValue
```

```
      {
        Name = "memory_utilization",
        Value = "40",
        Type = "DOUBLE"
      };

      var computationalRecord = new Record
      {
        MeasureName = "cpu_memory",
        MeasureValues = new List<MeasureValue> {cpuUtilization, memoryUtilization},
        MeasureValueType = "MULTI"
      };


      List<Record> records = new List<Record>();
      records.Add(computationalRecord);

      try
      {
        var writeRecordsRequest = new WriteRecordsRequest
        {
          DatabaseName = MultiMeasureValueConstants.MultiMeasureValueSampleDb,
          TableName = MultiMeasureValueConstants.MultiMeasureValueSampleTable,
          Records = records,
          CommonAttributes = commonAttributes
        };
        WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
        Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
      }
      catch (Exception e)
      {
        Console.WriteLine("Write records failure:" + e.ToString());
      }
    }

    public async Task WriteRecordsMultiMeasureValueMultipleRecords()
    {
      Console.WriteLine("Writing records with multi value attributes mixture type");

      DateTimeOffset now = DateTimeOffset.UtcNow;
      string currentTimeString = (now.ToUnixTimeMilliseconds()).ToString();
```

```
      List<Dimension> dimensions = new List<Dimension>{
        new Dimension { Name = "region", Value = "us-east-1" },
        new Dimension { Name = "az", Value = "az1" },
        new Dimension { Name = "hostname", Value = "host1" }
      };

      var commonAttributes = new Record
      {
        Dimensions = dimensions,
        Time = currentTimeString
      };

      var cpuUtilization = new MeasureValue
      {
        Name = "cpu_utilization",
        Value = "13.6",
        Type = "DOUBLE"
      };

      var memoryUtilization = new MeasureValue
      {
        Name = "memory_utilization",
        Value = "40",
        Type = "DOUBLE"
      };

      var activeCores = new MeasureValue
      {
        Name = "active_cores",
        Value = "4",
        Type = "BIGINT"
      };

      var computationalRecord = new Record
      {
        MeasureName = "computational_utilization",
        MeasureValues = new List<MeasureValue> {cpuUtilization, memoryUtilization,
activeCores},
        MeasureValueType = "MULTI"
      };

      var aliveRecord = new Record
      {
        MeasureName = "is_healthy",
```

```
        MeasureValue = "true",
        MeasureValueType = "BOOLEAN"
      };

      List<Record> records = new List<Record>();
      records.Add(computationalRecord);
      records.Add(aliveRecord);

      try
      {
        var writeRecordsRequest = new WriteRecordsRequest
        {
          DatabaseName = MultiMeasureValueConstants.MultiMeasureValueSampleDb,
          TableName = MultiMeasureValueConstants.MultiMeasureValueSampleTable,
          Records = records,
          CommonAttributes = commonAttributes
        };
        WriteRecordsResponse response = await
 writeClient.WriteRecordsAsync(writeRecordsRequest);
        Console.WriteLine($"Write records status code:
 {response.HttpStatusCode.ToString()}");
      }
      catch (Exception e)
      {
        Console.WriteLine("Write records failure:" + e.ToString());
      }
    }
  }
}
```

쓰기 실패 처리

Amazon Timestream의 쓰기는 다음 이유 중 하나 이상으로 인해 실패할 수 있습니다.

- 메모리 스토어의 보존 기간을 벗어나는 타임스탬프가 있는 레코드가 있습니다.
- Timestream에서 정의한 한도를 초과하는 차원 및/또는 측정값이 포함된 레코드가 있습니다.
- Amazon Timestream에서 중복 레코드를 감지했습니다. 동일한 차원, 타임스탬프 및 측정값 이름을 가진 레코드가 여러 개 있는 경우 레코드는 중복으로 표시됩니다.
  - 측정값은 다릅니다.

- 요청에 버전이 없거나 새 레코드의 버전 값이 기존 값과 같거나 낮습니다. Amazon Timestream이 이러한 이유로 데이터를 거부하는 경우 의 `ExistingVersion` 필드에`RejectedRecords`는 Amazon Timestream에 저장된 레코드의 현재 버전이 포함됩니다. 업데이트를 강제로 적용하려면 레코드 세트의 버전이 보다 큰 값으로 요청을 재전송할 수 있습니다`ExistingVersion`.

오류 및 거부된 레코드에 대한 자세한 내용은 오류 및 섹션을 참조하세요RejectedRecord.

Timestream에 레코드를 작성하려고 할 `RejectedRecordsException` 때 애플리케이션이 를 수신하는 경우 거부된 레코드를 구문 분석하여 아래와 같이 쓰기 실패에 대해 자세히 알아볼 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
 try {
   WriteRecordsResult writeRecordsResult =
amazonTimestreamWrite.writeRecords(writeRecordsRequest);
   System.out.println("WriteRecords Status: " +
writeRecordsResult.getSdkHttpMetadata().getHttpStatusCode());
 } catch (RejectedRecordsException e) {
   System.out.println("RejectedRecords: " + e);
   for (RejectedRecord rejectedRecord : e.getRejectedRecords()) {
     System.out.println("Rejected Index " + rejectedRecord.getRecordIndex() + ": "
         + rejectedRecord.getReason());
   }
   System.out.println("Other records were written successfully. ");
 } catch (Exception e) {
   System.out.println("Error: " + e);
 }
```

Java v2

```
   try {
     WriteRecordsResponse writeRecordsResponse =
timestreamWriteClient.writeRecords(writeRecordsRequest);
     System.out.println("writeRecordsWithCommonAttributes Status: " +
writeRecordsResponse.sdkHttpResponse().statusCode());
```

```
    } catch (RejectedRecordsException e) {
      System.out.println("RejectedRecords: " + e);
      for (RejectedRecord rejectedRecord : e.rejectedRecords()) {
        System.out.println("Rejected Index " + rejectedRecord.recordIndex() + ": "
            + rejectedRecord.reason());
      }
      System.out.println("Other records were written successfully. ");
    } catch (Exception e) {
      System.out.println("Error: " + e);
    }
```

Go

```
_, err = writeSvc.WriteRecords(writeRecordsInput)

if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
} else {
  fmt.Println("Write records is successful")
}
```

Python

```
try:
  result = self.client.write_records(DatabaseName=Constant.DATABASE_NAME,
 TableName=Constant.TABLE_NAME, Records=records, CommonAttributes=common_attributes)
  print("WriteRecords Status: [%s]" % result['ResponseMetadata']['HTTPStatusCode'])
except self.client.exceptions.RejectedRecordsException as err:
  print("RejectedRecords: ", err)
  for rr in err.response["RejectedRecords"]:
    print("Rejected Index " + str(rr["RecordIndex"]) + ": " + rr["Reason"])
  print("Other records were written successfully. ")
except Exception as err:
  print("Error:", err)
```

Node.js

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
await request.promise().then(
```

```
    (data) => {
      console.log("Write records successful");
    },
    (err) => {
      console.log("Error writing records:", err);
      if (err.code === 'RejectedRecordsException') {
        const responsePayload =
JSON.parse(request.response.httpResponse.body.toString());
        console.log("RejectedRecords: ", responsePayload.RejectedRecords);
        console.log("Other records were written successfully. ");
      }
    }
  );
```

.NET

```
 try
 {
   var writeRecordsRequest = new WriteRecordsRequest
   {
     DatabaseName = Constants.DATABASE_NAME,
     TableName = Constants.TABLE_NAME,
     Records = records,
     CommonAttributes = commonAttributes
   };
   WriteRecordsResponse response = await
writeClient.WriteRecordsAsync(writeRecordsRequest);
   Console.WriteLine($"Write records status code:
{response.HttpStatusCode.ToString()}");
 }
 catch (RejectedRecordsException e) {
   Console.WriteLine("RejectedRecordsException:" + e.ToString());
   foreach (RejectedRecord rr in e.RejectedRecords) {
     Console.WriteLine("RecordIndex " + rr.RecordIndex + " : " + rr.Reason);
   }
   Console.WriteLine("Other records were written successfully. ");
 }
 catch (Exception e)
 {
   Console.WriteLine("Write records failure:" + e.ToString());
 }
```

쿼리 실행

주제

결과 페이지 매김

쿼리를 실행하면 Timestream은 애플리케이션의 응답성을 최적화하기 위해 페이지 매김 방식으로 결과 세트를 반환합니다. 아래 코드 조각은 결과 집합을 파지하는 방법을 보여줍니다. null 값이 표시될 때까지 모든 결과 세트 페이지를 반복해야 합니다. 페이지 매김 토큰은 에 대해 Timestream에서 발급된 후 3시간 후에 만료됩니다 LiveAnalytics.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
    private void runQuery(String queryString) {
        try {
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.setQueryString(queryString);
            QueryResult queryResult = queryClient.query(queryRequest);
            while (true) {
                parseQueryResult(queryResult);
                if (queryResult.getNextToken() == null) {
                    break;
                }
                queryRequest.setNextToken(queryResult.getNextToken());
                queryResult = queryClient.query(queryRequest);
            }
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
has more than 10000 entries
            e.printStackTrace();
        }
```

```
    }
```

Java v2

```
    private void runQuery(String queryString) {
        try {
            QueryRequest queryRequest =
 QueryRequest.builder().queryString(queryString).build();
            final QueryIterable queryResponseIterator =
 timestreamQueryClient.queryPaginator(queryRequest);
            for(QueryResponse queryResponse : queryResponseIterator) {
                parseQueryResult(queryResponse);
            }
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
 has more than 10000 entries
            e.printStackTrace();
        }
    }
```

Go

```
func runQuery(queryPtr *string, querySvc *timestreamquery.TimestreamQuery, f
 *os.File) {
    queryInput := &timestreamquery.QueryInput{
        QueryString: aws.String(*queryPtr),
    }
    fmt.Println("QueryInput:")
    fmt.Println(queryInput)
    // execute the query
    err := querySvc.QueryPages(queryInput,
        func(page *timestreamquery.QueryOutput, lastPage bool) bool {
            // process query response
            queryStatus := page.QueryStatus
            fmt.Println("Current query status:", queryStatus)
            // query response metadata
            // includes column names and types
            metadata := page.ColumnInfo
            // fmt.Println("Metadata:")
            fmt.Println(metadata)
            header := ""
            for i := 0; i < len(metadata); i++ {
                header += *metadata[i].Name
```

```
                if i != len(metadata)-1 {
                    header += ", "
                }
            }
            write(f, header)

            // query response data
            fmt.Println("Data:")
            // process rows
            rows := page.Rows
            for i := 0; i < len(rows); i++ {
                data := rows[i].Data
                value := processRowType(data, metadata)
                fmt.Println(value)
                write(f, value)
            }
            fmt.Println("Number of rows:", len(page.Rows))
            return true
        })
    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    }
}
```

Python

```
    def run_query(self, query_string):
        try:
            page_iterator = self.paginator.paginate(QueryString=query_string)
            for page in page_iterator:
                self._parse_query_result(page)
        except Exception as err:
            print("Exception while running query:", err)
```

Node.js

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
async function getAllRows(query, nextToken) {
    const params = {
        QueryString: query
```

```
    };

    if (nextToken) {
        params.NextToken = nextToken;
    }

    await queryClient.query(params).promise()
        .then(
            (response) => {
                parseQueryResult(response);
                if (response.NextToken) {
                    getAllRows(query, response.NextToken);
                }
            },
            (err) => {
                console.error("Error while querying:", err);
            });
}
```

.NET

```
        private async Task RunQueryAsync(string queryString)
        {
            try
            {
                QueryRequest queryRequest = new QueryRequest();
                queryRequest.QueryString = queryString;
                QueryResponse queryResponse = await
 queryClient.QueryAsync(queryRequest);
                while (true)
                {
                    ParseQueryResult(queryResponse);
                    if (queryResponse.NextToken == null)
                    {
                        break;
                    }
                    queryRequest.NextToken = queryResponse.NextToken;
                    queryResponse = await queryClient.QueryAsync(queryRequest);
                }
            } catch(Exception e)
            {
                // Some queries might fail with 500 if the result of a sequence
 function has more than 10000 entries
```

```
                Console.WriteLine(e.ToString());
            }
        }
```

결과 세트 구문 분석

다음 코드 조각을 사용하여 결과 세트에서 데이터를 추출할 수 있습니다. 쿼리 결과는 쿼리가 완료된 후 최대 24시간 동안 액세스할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
   private static final DateTimeFormatter TIMESTAMP_FORMATTER =
DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss.SSSSSSSSS");
   private static final DateTimeFormatter DATE_FORMATTER =
DateTimeFormatter.ofPattern("yyyy-MM-dd");
   private static final DateTimeFormatter TIME_FORMATTER =
DateTimeFormatter.ofPattern("HH:mm:ss.SSSSSSSSS");

   private static final long ONE_GB_IN_BYTES = 1073741824L;

   private void parseQueryResult(QueryResult response) {
       final QueryStatus currentStatusOfQuery = queryResult.getQueryStatus();

       System.out.println("Query progress so far: " +
currentStatusOfQuery.getProgressPercentage() + "%");

       double bytesScannedSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesScanned() / ONE_GB_IN_BYTES);
       System.out.println("Bytes scanned so far: " + bytesScannedSoFar + " GB");

       double bytesMeteredSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesMetered() / ONE_GB_IN_BYTES);
       System.out.println("Bytes metered so far: " + bytesMeteredSoFar + " GB");

       List<ColumnInfo> columnInfo = response.getColumnInfo();
```

```
        List<Row> rows = response.getRows();

        System.out.println("Metadata: " + columnInfo);
        System.out.println("Data: ");

        // iterate every row
        for (Row row : rows) {
            System.out.println(parseRow(columnInfo, row));
        }
    }

    private String parseRow(List<ColumnInfo> columnInfo, Row row) {
        List<Datum> data = row.getData();
        List<String> rowOutput = new ArrayList<>();
        // iterate every column per row
        for (int j = 0; j < data.size(); j++) {
            ColumnInfo info = columnInfo.get(j);
            Datum datum = data.get(j);
            rowOutput.add(parseDatum(info, datum));
        }
        return String.format("{%s}",
rowOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseDatum(ColumnInfo info, Datum datum) {
        if (datum.isNullValue() != null && datum.isNullValue()) {
            return info.getName() + "=" + "NULL";
        }
        Type columnType = info.getType();
        // If the column is of TimeSeries Type
        if (columnType.getTimeSeriesMeasureValueColumnInfo() != null) {
            return parseTimeSeries(info, datum);
        }
        // If the column is of Array Type
        else if (columnType.getArrayColumnInfo() != null) {
            List<Datum> arrayValues = datum.getArrayValue();
            return info.getName() + "=" +
parseArray(info.getType().getArrayColumnInfo(), arrayValues);
        }
        // If the column is of Row Type
        else if (columnType.getRowColumnInfo() != null) {
            List<ColumnInfo> rowColumnInfo = info.getType().getRowColumnInfo();
            Row rowValues = datum.getRowValue();
            return parseRow(rowColumnInfo, rowValues);
```

```
        }
        // If the column is of Scalar Type
        else {
            return parseScalarType(info, datum);
        }
    }

    private String parseTimeSeries(ColumnInfo info, Datum datum) {
        List<String> timeSeriesOutput = new ArrayList<>();
        for (TimeSeriesDataPoint dataPoint : datum.getTimeSeriesValue()) {
            timeSeriesOutput.add("{time=" + dataPoint.getTime() + ", value=" +
                    parseDatum(info.getType().getTimeSeriesMeasureValueColumnInfo(),
dataPoint.getValue()) + "}");
        }
        return String.format("[%s]",
timeSeriesOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseScalarType(ColumnInfo info, Datum datum) {
        switch (ScalarType.fromValue(info.getType().getScalarType())) {
            case VARCHAR:
                return parseColumnName(info) + datum.getScalarValue();
            case BIGINT:
                Long longValue = Long.valueOf(datum.getScalarValue());
                return parseColumnName(info) + longValue;
            case INTEGER:
                Integer intValue = Integer.valueOf(datum.getScalarValue());
                return parseColumnName(info) + intValue;
            case BOOLEAN:
                Boolean booleanValue = Boolean.valueOf(datum.getScalarValue());
                return parseColumnName(info) + booleanValue;
            case DOUBLE:
                Double doubleValue = Double.valueOf(datum.getScalarValue());
                return parseColumnName(info) + doubleValue;
            case TIMESTAMP:
                return parseColumnName(info) +
LocalDateTime.parse(datum.getScalarValue(), TIMESTAMP_FORMATTER);
            case DATE:
                return parseColumnName(info) +
LocalDate.parse(datum.getScalarValue(), DATE_FORMATTER);
            case TIME:
                return parseColumnName(info) +
LocalTime.parse(datum.getScalarValue(), TIME_FORMATTER);
            case INTERVAL_DAY_TO_SECOND:
```

```
            case INTERVAL_YEAR_TO_MONTH:
                return parseColumnName(info) + datum.getScalarValue();
            case UNKNOWN:
                return parseColumnName(info) + datum.getScalarValue();
            default:
                throw new IllegalArgumentException("Given type is not valid: " +
info.getType().getScalarType());
        }
    }

    private String parseColumnName(ColumnInfo info) {
        return info.getName() == null ? "" : info.getName() + "=";
    }

    private String parseArray(ColumnInfo arrayColumnInfo, List<Datum> arrayValues) {
        List<String> arrayOutput = new ArrayList<>();
        for (Datum datum : arrayValues) {
            arrayOutput.add(parseDatum(arrayColumnInfo, datum));
        }
        return String.format("[%s]",
arrayOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }
```

Java v2

```
    private static final long ONE_GB_IN_BYTES = 1073741824L;

    private void parseQueryResult(QueryResponse response) {
        final QueryStatus currentStatusOfQuery = response.queryStatus();

        System.out.println("Query progress so far: " +
currentStatusOfQuery.progressPercentage() + "%");

        double bytesScannedSoFar = ((double)
currentStatusOfQuery.cumulativeBytesScanned() / ONE_GB_IN_BYTES);
        System.out.println("Bytes scanned so far: " + bytesScannedSoFar + " GB");

        double bytesMeteredSoFar = ((double)
currentStatusOfQuery.cumulativeBytesMetered() / ONE_GB_IN_BYTES);
        System.out.println("Bytes metered so far: " + bytesMeteredSoFar + " GB");

        List<ColumnInfo> columnInfo = response.columnInfo();
        List<Row> rows = response.rows();
```

```
        System.out.println("Metadata: " + columnInfo);
        System.out.println("Data: ");

        // iterate every row
        for (Row row : rows) {
            System.out.println(parseRow(columnInfo, row));
        }
    }

    private String parseRow(List<ColumnInfo> columnInfo, Row row) {
        List<Datum> data = row.data();
        List<String> rowOutput = new ArrayList<>();
        // iterate every column per row
        for (int j = 0; j < data.size(); j++) {
            ColumnInfo info = columnInfo.get(j);
            Datum datum = data.get(j);
            rowOutput.add(parseDatum(info, datum));
        }
        return String.format("{%s}",
rowOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseDatum(ColumnInfo info, Datum datum) {
        if (datum.nullValue() != null && datum.nullValue()) {
            return info.name() + "=" + "NULL";
        }
        Type columnType = info.type();
        // If the column is of TimeSeries Type
        if (columnType.timeSeriesMeasureValueColumnInfo() != null) {
            return parseTimeSeries(info, datum);
        }
        // If the column is of Array Type
        else if (columnType.arrayColumnInfo() != null) {
            List<Datum> arrayValues = datum.arrayValue();
            return info.name() + "=" + parseArray(info.type().arrayColumnInfo(),
arrayValues);
        }
        // If the column is of Row Type
        else if (columnType.rowColumnInfo() != null &&
columnType.rowColumnInfo().size() > 0) {
            List<ColumnInfo> rowColumnInfo = info.type().rowColumnInfo();
            Row rowValues = datum.rowValue();
            return parseRow(rowColumnInfo, rowValues);
```

```
        }
        // If the column is of Scalar Type
        else {
            return parseScalarType(info, datum);
        }
    }

    private String parseTimeSeries(ColumnInfo info, Datum datum) {
        List<String> timeSeriesOutput = new ArrayList<>();
        for (TimeSeriesDataPoint dataPoint : datum.timeSeriesValue()) {
            timeSeriesOutput.add("{time=" + dataPoint.time() + ", value=" +
                    parseDatum(info.type().timeSeriesMeasureValueColumnInfo(),
 dataPoint.value()) + "}");
        }
        return String.format("[%s]",
 timeSeriesOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }

    private String parseScalarType(ColumnInfo info, Datum datum) {
        return parseColumnName(info) + datum.scalarValue();
    }

    private String parseColumnName(ColumnInfo info) {
        return info.name() == null ? "" : info.name() + "=";
    }

    private String parseArray(ColumnInfo arrayColumnInfo, List<Datum> arrayValues) {
        List<String> arrayOutput = new ArrayList<>();
        for (Datum datum : arrayValues) {
            arrayOutput.add(parseDatum(arrayColumnInfo, datum));
        }
        return String.format("[%s]",
 arrayOutput.stream().map(Object::toString).collect(Collectors.joining(",")));
    }
```

Go

```
func processScalarType(data *timestreamquery.Datum) string {
    return *data.ScalarValue
}

func processTimeSeriesType(data []*timestreamquery.TimeSeriesDataPoint, columnInfo
 *timestreamquery.ColumnInfo) string {
```

```
    value := ""
    for k := 0; k < len(data); k++ {
        time := data[k].Time
        value += *time + ":"
        if columnInfo.Type.ScalarType != nil {
            value += processScalarType(data[k].Value)
        } else if columnInfo.Type.ArrayColumnInfo != nil {
            value += processArrayType(data[k].Value.ArrayValue,
 columnInfo.Type.ArrayColumnInfo)
        } else if columnInfo.Type.RowColumnInfo != nil {
            value += processRowType(data[k].Value.RowValue.Data,
 columnInfo.Type.RowColumnInfo)
        } else {
            fail("Bad data type")
        }
        if k != len(data)-1 {
            value += ", "
        }
    }
    return value
}

func processArrayType(datumList []*timestreamquery.Datum, columnInfo
 *timestreamquery.ColumnInfo) string {
    value := ""
    for k := 0; k < len(datumList); k++ {
        if columnInfo.Type.ScalarType != nil {
            value += processScalarType(datumList[k])
        } else if columnInfo.Type.TimeSeriesMeasureValueColumnInfo != nil {
            value += processTimeSeriesType(datumList[k].TimeSeriesValue,
 columnInfo.Type.TimeSeriesMeasureValueColumnInfo)
        } else if columnInfo.Type.ArrayColumnInfo != nil {
            value += "["
            value += processArrayType(datumList[k].ArrayValue,
 columnInfo.Type.ArrayColumnInfo)
            value += "]"
        } else if columnInfo.Type.RowColumnInfo != nil {
            value += "["
            value += processRowType(datumList[k].RowValue.Data,
 columnInfo.Type.RowColumnInfo)
            value += "]"
        } else {
            fail("Bad column type")
        }
```

```
        if k != len(datumList)-1 {
            value += ", "
        }
    }
    return value
}

func processRowType(data []*timestreamquery.Datum, metadata
 []*timestreamquery.ColumnInfo) string {
    value := ""
    for j := 0; j < len(data); j++ {
        if metadata[j].Type.ScalarType != nil {
            // process simple data types
            value += processScalarType(data[j])
        } else if metadata[j].Type.TimeSeriesMeasureValueColumnInfo != nil {
            // fmt.Println("Timeseries measure value column info")
            // fmt.Println(metadata[j].Type.TimeSeriesMeasureValueColumnInfo.Type)
            datapointList := data[j].TimeSeriesValue
            value += "["
            value += processTimeSeriesType(datapointList,
 metadata[j].Type.TimeSeriesMeasureValueColumnInfo)
            value += "]"
        } else if metadata[j].Type.ArrayColumnInfo != nil {
            columnInfo := metadata[j].Type.ArrayColumnInfo
            // fmt.Println("Array column info")
            // fmt.Println(columnInfo)
            datumList := data[j].ArrayValue
            value += "["
            value += processArrayType(datumList, columnInfo)
            value += "]"
        } else if metadata[j].Type.RowColumnInfo != nil {
            columnInfo := metadata[j].Type.RowColumnInfo
            datumList := data[j].RowValue.Data
            value += "["
            value += processRowType(datumList, columnInfo)
            value += "]"
        } else {
            panic("Bad column type")
        }
        // comma seperated column values
        if j != len(data)-1 {
            value += ", "
        }
```

```
    }
    return value
}
```

Python

```
    def _parse_query_result(self, query_result):
        query_status = query_result["QueryStatus"]

        progress_percentage = query_status["ProgressPercentage"]
        print(f"Query progress so far: {progress_percentage}%")

        bytes_scanned = float(query_status["CumulativeBytesScanned"]) /
ONE_GB_IN_BYTES
        print(f"Data scanned so far: {bytes_scanned} GB")

        bytes_metered = float(query_status["CumulativeBytesMetered"]) /
ONE_GB_IN_BYTES
        print(f"Data metered so far: {bytes_metered} GB")

        column_info = query_result['ColumnInfo']

        print("Metadata: %s" % column_info)
        print("Data: ")
        for row in query_result['Rows']:
            print(self._parse_row(column_info, row))

    def _parse_row(self, column_info, row):
        data = row['Data']
        row_output = []
        for j in range(len(data)):
            info = column_info[j]
            datum = data[j]
            row_output.append(self._parse_datum(info, datum))

        return "{%s}" % str(row_output)

    def _parse_datum(self, info, datum):
        if datum.get('NullValue', False):
            return "%s=NULL" % info['Name'],

        column_type = info['Type']
```

```
        # If the column is of TimeSeries Type
        if 'TimeSeriesMeasureValueColumnInfo' in column_type:
            return self._parse_time_series(info, datum)

        # If the column is of Array Type
        elif 'ArrayColumnInfo' in column_type:
            array_values = datum['ArrayValue']
            return "%s=%s" % (info['Name'], self._parse_array(info['Type']
['ArrayColumnInfo'], array_values))

        # If the column is of Row Type
        elif 'RowColumnInfo' in column_type:
            row_column_info = info['Type']['RowColumnInfo']
            row_values = datum['RowValue']
            return self._parse_row(row_column_info, row_values)

        # If the column is of Scalar Type
        else:
            return self._parse_column_name(info) + datum['ScalarValue']

    def _parse_time_series(self, info, datum):
        time_series_output = []
        for data_point in datum['TimeSeriesValue']:
            time_series_output.append("{time=%s, value=%s}"
                                      % (data_point['Time'],
                                         self._parse_datum(info['Type']
['TimeSeriesMeasureValueColumnInfo'],
                                                           data_point['Value'])))
        return "[%s]" % str(time_series_output)

    def _parse_array(self, array_column_info, array_values):
        array_output = []
        for datum in array_values:
            array_output.append(self._parse_datum(array_column_info, datum))

        return "[%s]" % str(array_output)

    @staticmethod
    def _parse_column_name(info):
        if 'Name' in info:
            return info['Name'] + "="
        else:
            return ""
```

Node.js

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
function parseQueryResult(response) {
    const queryStatus = response.QueryStatus;
    console.log("Current query status: " + JSON.stringify(queryStatus));

    const columnInfo = response.ColumnInfo;
    const rows = response.Rows;

    console.log("Metadata: " + JSON.stringify(columnInfo));
    console.log("Data: ");

    rows.forEach(function (row) {
        console.log(parseRow(columnInfo, row));
    });
}

function parseRow(columnInfo, row) {
    const data = row.Data;
    const rowOutput = [];

    var i;
    for ( i = 0; i < data.length; i++ ) {
        info = columnInfo[i];
        datum = data[i];
        rowOutput.push(parseDatum(info, datum));
    }

    return `{${rowOutput.join(", ")}}`
}

function parseDatum(info, datum) {
    if (datum.NullValue != null && datum.NullValue === true) {
        return `${info.Name}=NULL`;
    }

    const columnType = info.Type;

    // If the column is of TimeSeries Type
    if (columnType.TimeSeriesMeasureValueColumnInfo != null) {
        return parseTimeSeries(info, datum);
```

```
    }
    // If the column is of Array Type
    else if (columnType.ArrayColumnInfo != null) {
        const arrayValues = datum.ArrayValue;
        return `${info.Name}=${parseArray(info.Type.ArrayColumnInfo, arrayValues)}`;
    }
    // If the column is of Row Type
    else if (columnType.RowColumnInfo != null) {
        const rowColumnInfo = info.Type.RowColumnInfo;
        const rowValues = datum.RowValue;
        return parseRow(rowColumnInfo, rowValues);
    }
    // If the column is of Scalar Type
    else {
        return parseScalarType(info, datum);
    }
}

function parseTimeSeries(info, datum) {
    const timeSeriesOutput = [];
    datum.TimeSeriesValue.forEach(function (dataPoint) {
        timeSeriesOutput.push(`{time=${dataPoint.Time}, value=
${parseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, dataPoint.Value)}}`)
    });

    return `[${timeSeriesOutput.join(", ")}]`
}

function parseScalarType(info, datum) {
    return parseColumnName(info) + datum.ScalarValue;
}

function parseColumnName(info) {
    return info.Name == null ? "" : `${info.Name}=`;
}

function parseArray(arrayColumnInfo, arrayValues) {
    const arrayOutput = [];
    arrayValues.forEach(function (datum) {
        arrayOutput.push(parseDatum(arrayColumnInfo, datum));
    });
    return `[${arrayOutput.join(", ")}]`
}
```

.NET

```
        private void ParseQueryResult(QueryResponse response)
        {
            List<ColumnInfo> columnInfo = response.ColumnInfo;
            var options = new JsonSerializerOptions
            {
                IgnoreNullValues = true
            };
            List<String> columnInfoStrings = columnInfo.ConvertAll(x =>
JsonSerializer.Serialize(x, options));
            List<Row> rows = response.Rows;

            QueryStatus queryStatus = response.QueryStatus;
            Console.WriteLine("Current Query status:" +
JsonSerializer.Serialize(queryStatus, options));

            Console.WriteLine("Metadata:" + string.Join(",", columnInfoStrings));
            Console.WriteLine("Data:");

            foreach (Row row in rows)
            {
                Console.WriteLine(ParseRow(columnInfo, row));
            }
        }

        private string ParseRow(List<ColumnInfo> columnInfo, Row row)
        {
            List<Datum> data = row.Data;
            List<string> rowOutput = new List<string>();
            for (int j = 0; j < data.Count; j++)
            {
                ColumnInfo info = columnInfo[j];
                Datum datum = data[j];
                rowOutput.Add(ParseDatum(info, datum));
            }
            return $"{{{string.Join(",", rowOutput)}}}";
        }

        private string ParseDatum(ColumnInfo info, Datum datum)
        {
            if (datum.NullValue)
            {
                return $"{info.Name}=NULL";
```

```
            }

            Amazon.TimestreamQuery.Model.Type columnType = info.Type;
            if (columnType.TimeSeriesMeasureValueColumnInfo != null)
            {
                return ParseTimeSeries(info, datum);
            }
            else if (columnType.ArrayColumnInfo != null)
            {
                List<Datum> arrayValues = datum.ArrayValue;
                return $"{info.Name}={ParseArray(info.Type.ArrayColumnInfo,
arrayValues)}";
            }
            else if (columnType.RowColumnInfo != null &&
columnType.RowColumnInfo.Count > 0)
            {
                List<ColumnInfo> rowColumnInfo = info.Type.RowColumnInfo;
                Row rowValue = datum.RowValue;
                return ParseRow(rowColumnInfo, rowValue);
            }
            else
            {
                return ParseScalarType(info, datum);
            }
        }

        private string ParseTimeSeries(ColumnInfo info, Datum datum)
        {
            var timeseriesString = datum.TimeSeriesValue
                .Select(value => $"{{time={value.Time},
value={ParseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, value.Value)}}}")
                .Aggregate((current, next) => current + "," + next);

            return $"[{timeseriesString}]";
        }

        private string ParseScalarType(ColumnInfo info, Datum datum)
        {
            return ParseColumnName(info) + datum.ScalarValue;
        }

        private string ParseColumnName(ColumnInfo info)
        {
            return info.Name == null ? "" : (info.Name + "=");
```

```
        }

        private string ParseArray(ColumnInfo arrayColumnInfo, List<Datum>
arrayValues)
        {
            return $"[{arrayValues.Select(value => ParseDatum(arrayColumnInfo,
value)).Aggregate((current, next) => current + "," + next)}]";
        }
```

쿼리 상태 액세스

를 통해 쿼리 상태에 액세스할 수 있습니다. 여기에는 쿼리 진행 상황QueryResponse, 쿼리로 스캔한 바이트, 쿼리로 측정한 바이트에 대한 정보가 포함되어 있습니다. bytesMetered 및 bytesScanned 값은 누적되며 쿼리 결과를 호출하는 동안 지속적으로 업데이트됩니다. 이 정보를 사용하여 개별 쿼리에서 스캔한 바이트를 이해하고 이를 사용하여 특정 결정을 내릴 수도 있습니다. 예를 들어, 쿼리 가격이 스캔한 GB당 0.01달러라고 가정하면 쿼리당 25달러 또는 X GB를 초과하는 쿼리를 취소할 수 있습니다. 아래 코드 조각은 이 작업을 수행하는 방법을 보여줍니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
   private static final long ONE_GB_IN_BYTES = 1073741824L;
   private static final double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
price of query is $0.01 per GB

   public void cancelQueryBasedOnQueryStatus() {
       System.out.println("Starting query: " + SELECT_ALL_QUERY);
       QueryRequest queryRequest = new QueryRequest();
       queryRequest.setQueryString(SELECT_ALL_QUERY);
       QueryResult queryResult = queryClient.query(queryRequest);

       while (true) {
           final QueryStatus currentStatusOfQuery = queryResult.getQueryStatus();
           System.out.println("Query progress so far: " +
currentStatusOfQuery.getProgressPercentage() + "%");
```

```
            double bytesMeteredSoFar = ((double)
currentStatusOfQuery.getCumulativeBytesMetered() / ONE_GB_IN_BYTES);
            System.out.println("Bytes metered so far: " + bytesMeteredSoFar + "
GB");
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01) {
                cancelQuery(queryResult);
                break;
            }

            if (queryResult.getNextToken() == null) {
                break;
            }
            queryRequest.setNextToken(queryResult.getNextToken());
            queryResult = queryClient.query(queryRequest);
        }
    }
```

Java v2

```
    private static final long ONE_GB_IN_BYTES = 1073741824L;
    private static final double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
price of query is $0.01 per GB

    public void cancelQueryBasedOnQueryStatus() {
        System.out.println("Starting query: " + SELECT_ALL_QUERY);
        QueryRequest queryRequest =
QueryRequest.builder().queryString(SELECT_ALL_QUERY).build();

        final QueryIterable queryResponseIterator =
timestreamQueryClient.queryPaginator(queryRequest);
        for(QueryResponse queryResponse : queryResponseIterator) {
            final QueryStatus currentStatusOfQuery = queryResponse.queryStatus();
            System.out.println("Query progress so far: " +
currentStatusOfQuery.progressPercentage() + "%");
            double bytesMeteredSoFar = ((double)
currentStatusOfQuery.cumulativeBytesMetered() / ONE_GB_IN_BYTES);
            System.out.println("Bytes metered so far: " + bytesMeteredSoFar + "GB");
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01) {
                cancelQuery(queryResponse);
                break;
            }
```

```
        }
    }
```

Go

```
const OneGbInBytes = 1073741824
// Assuming the price of query is $0.01 per GB
const QueryCostPerGbInDollars = 0.01

func cancelQueryBasedOnQueryStatus(queryPtr *string, querySvc
 *timestreamquery.TimestreamQuery, f *os.File) {
    queryInput := &timestreamquery.QueryInput{
        QueryString: aws.String(*queryPtr),
    }
    fmt.Println("QueryInput:")
    fmt.Println(queryInput)
    // execute the query
    err := querySvc.QueryPages(queryInput,
        func(page *timestreamquery.QueryOutput, lastPage bool) bool {
            // process query response
            queryStatus := page.QueryStatus
            fmt.Println("Current query status:", queryStatus)
            bytes_metered := float64(*queryStatus.CumulativeBytesMetered) /
 float64(ONE_GB_IN_BYTES)
            if bytes_metered * QUERY_COST_PER_GB_IN_DOLLARS > 0.01 {
                cancelQuery(page, querySvc)
                return true
            }
            // query response metadata
            // includes column names and types
            metadata := page.ColumnInfo
            // fmt.Println("Metadata:")
            fmt.Println(metadata)
            header := ""
            for i := 0; i < len(metadata); i++ {
                header += *metadata[i].Name
                if i != len(metadata)-1 {
                    header += ", "
                }
            }
            write(f, header)

            // query response data
```

```
            fmt.Println("Data:")
            // process rows
            rows := page.Rows
            for i := 0; i < len(rows); i++ {
                data := rows[i].Data
                value := processRowType(data, metadata)
                fmt.Println(value)
                write(f, value)
            }
            fmt.Println("Number of rows:", len(page.Rows))
            return true
        })
    if err != nil {
        fmt.Println("Error:")
        fmt.Println(err)
    }
}
```

Python

```
ONE_GB_IN_BYTES = 1073741824
# Assuming the price of query is $0.01 per GB
QUERY_COST_PER_GB_IN_DOLLARS = 0.01

    def cancel_query_based_on_query_status(self):
        try:
            print("Starting query: " + self.SELECT_ALL)
            page_iterator = self.paginator.paginate(QueryString=self.SELECT_ALL)
            for page in page_iterator:
                query_status = page["QueryStatus"]
                progress_percentage = query_status["ProgressPercentage"]
                print("Query progress so far: " + str(progress_percentage) + "%")
                bytes_metered = query_status["CumulativeBytesMetered"] /
 self.ONE_GB_IN_BYTES
                print("Bytes Metered so far: " + str(bytes_metered) + " GB")
                if bytes_metered * self.QUERY_COST_PER_GB_IN_DOLLARS > 0.01:
                    self.cancel_query_for(page)
                    break
        except Exception as err:
            print("Exception while running query:", err)
            traceback.print_exc(file=sys.stderr)
```

Node.js

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
function parseQueryResult(response) {
    const queryStatus = response.QueryStatus;
    console.log("Current query status: " + JSON.stringify(queryStatus));

    const columnInfo = response.ColumnInfo;
    const rows = response.Rows;

    console.log("Metadata: " + JSON.stringify(columnInfo));
    console.log("Data: ");

    rows.forEach(function (row) {
        console.log(parseRow(columnInfo, row));
    });
}

function parseRow(columnInfo, row) {
    const data = row.Data;
    const rowOutput = [];

    var i;
    for ( i = 0; i < data.length; i++ ) {
        info = columnInfo[i];
        datum = data[i];
        rowOutput.push(parseDatum(info, datum));
    }

    return `{${rowOutput.join(", ")}}`
}

function parseDatum(info, datum) {
    if (datum.NullValue != null && datum.NullValue === true) {
        return `${info.Name}=NULL`;
    }

    const columnType = info.Type;

    // If the column is of TimeSeries Type
    if (columnType.TimeSeriesMeasureValueColumnInfo != null) {
        return parseTimeSeries(info, datum);
```

```
    }
    // If the column is of Array Type
    else if (columnType.ArrayColumnInfo != null) {
        const arrayValues = datum.ArrayValue;
        return `${info.Name}=${parseArray(info.Type.ArrayColumnInfo, arrayValues)}`;
    }
    // If the column is of Row Type
    else if (columnType.RowColumnInfo != null) {
        const rowColumnInfo = info.Type.RowColumnInfo;
        const rowValues = datum.RowValue;
        return parseRow(rowColumnInfo, rowValues);
    }
    // If the column is of Scalar Type
    else {
        return parseScalarType(info, datum);
    }
}

function parseTimeSeries(info, datum) {
    const timeSeriesOutput = [];
    datum.TimeSeriesValue.forEach(function (dataPoint) {
        timeSeriesOutput.push(`{time=${dataPoint.Time}, value=
${parseDatum(info.Type.TimeSeriesMeasureValueColumnInfo, dataPoint.Value)}}`)
    });

    return `[${timeSeriesOutput.join(", ")}]`
}

function parseScalarType(info, datum) {
    return parseColumnName(info) + datum.ScalarValue;
}

function parseColumnName(info) {
    return info.Name == null ? "" : `${info.Name}=`;
}

function parseArray(arrayColumnInfo, arrayValues) {
    const arrayOutput = [];
    arrayValues.forEach(function (datum) {
        arrayOutput.push(parseDatum(arrayColumnInfo, datum));
    });
    return `[${arrayOutput.join(", ")}]`
}
```

.NET

```
private static readonly long ONE_GB_IN_BYTES = 1073741824L;
private static readonly double QUERY_COST_PER_GB_IN_DOLLARS = 0.01; // Assuming the
 price of query is $0.01 per GB

private async Task CancelQueryBasedOnQueryStatus(string queryString)
{
    try
    {
        QueryRequest queryRequest = new QueryRequest();
        queryRequest.QueryString = queryString;
        QueryResponse queryResponse = await queryClient.QueryAsync(queryRequest);
        while (true)
        {
            QueryStatus queryStatus = queryResponse.QueryStatus;
            double bytesMeteredSoFar = ((double)
 queryStatus.CumulativeBytesMetered / ONE_GB_IN_BYTES);
            // Cancel query if its costing more than 1 cent
            if (bytesMeteredSoFar * QUERY_COST_PER_GB_IN_DOLLARS > 0.01)
            {
                await CancelQuery(queryResponse);
                break;
            }

            ParseQueryResult(queryResponse);
            if (queryResponse.NextToken == null)
            {
                break;
            }
            queryRequest.NextToken = queryResponse.NextToken;
            queryResponse = await queryClient.QueryAsync(queryRequest);
       }
    } catch(Exception e)
    {
        // Some queries might fail with 500 if the result of a sequence function has
 more than 10000 entries
        Console.WriteLine(e.ToString());
    }
}
```

쿼리를 취소하는 방법에 대한 자세한 내용은 섹션을 참조하세요쿼리 취소.

UNLOAD 쿼리 실행

다음 코드 예제에서는 UNLOAD 쿼리를 호출합니다. UNLOAD에 대한 자세한 내용은 UNLOAD 를 사용하여 에 대해 Timestream에서 S3로 쿼리 결과 내보내기 LiveAnalytics 섹션을 참조하십시오. UNLOAD 쿼리의 예는 섹션을 참조하세요에 대한 TimestreamUNLOAD의 에 대한 사용 사례 예 LiveAnalytics.

주제

- UNLOAD 쿼리 빌드 및 실행
- UNLOAD 응답을 구문 분석하고 행 수, 매니페스트 링크 및 메타데이터 링크 가져오기
- 매니페스트 콘텐츠 읽기 및 구문 분석
- 메타데이터 콘텐츠 읽기 및 구문 분석

UNLOAD 쿼리 빌드 및 실행

Java

```
// When you have a SELECT like below

String QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
        + DATABASE_NAME + "." + UNLOAD_TABLE_NAME
        + " WHERE time BETWEEN ago(2d) AND now()";

// You can construct UNLOAD query as follows
UnloadQuery unloadQuery = UnloadQuery.builder()
        .selectQuery(QUERY_1)
        .bucketName("timestream-sample-<region>-<accountId>")
        .resultsPrefix("without_partition")
        .format(CSV)
        .compression(UnloadQuery.Compression.GZIP)
        .build();
QueryResult unloadResult = runQuery(unloadQuery.getUnloadQuery());

// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
    private QueryResult runQuery(String queryString) {
        QueryResult queryResult = null;
        try {
            QueryRequest queryRequest = new QueryRequest();
```

```
            queryRequest.setQueryString(queryString);
            queryResult = queryClient.query(queryRequest);
            while (true) {
                parseQueryResult(queryResult);
                if (queryResult.getNextToken() == null) {
                    break;
                }
                queryRequest.setNextToken(queryResult.getNextToken());
                queryResult = queryClient.query(queryRequest);
            }
        } catch (Exception e) {
            // Some queries might fail with 500 if the result of a sequence function
 has more than 10000 entries
            e.printStackTrace();
        }
        return queryResult;
    }

// Utility that helps to construct UNLOAD query

@Builder
static class UnloadQuery {
    private String selectQuery;
    private String bucketName;
    private String resultsPrefix;
    private Format format;
    private Compression compression;
    private EncryptionType encryptionType;
    private List<String> partitionColumns;
    private String kmsKey;
    private Character csvFieldDelimiter;
    private Character csvEscapeCharacter;

    public String getUnloadQuery() {
        String destination = constructDestination();
        String withClause = constructOptionalParameters();
        return String.format("UNLOAD (%s) TO '%s' %s", selectQuery, destination,
 withClause);
    }

    private String constructDestination() {
        return "s3://" + this.bucketName + "/" + this.resultsPrefix + "/";
    }
```

```
   private String constructOptionalParameters() {
       boolean isOptionalParametersPresent = Objects.nonNull(format)
               || Objects.nonNull(compression)
               || Objects.nonNull(encryptionType)
               || Objects.nonNull(partitionColumns)
               || Objects.nonNull(kmsKey)
               || Objects.nonNull(csvFieldDelimiter)
               || Objects.nonNull(csvEscapeCharacter);

       String withClause = "";
       if (isOptionalParametersPresent) {
           StringJoiner optionalParameters = new StringJoiner(",");
           if (Objects.nonNull(format)) {
               optionalParameters.add("format = '" + format + "'");
           }
           if (Objects.nonNull(compression)) {
               optionalParameters.add("compression = '" + compression + "'");
           }
           if (Objects.nonNull(encryptionType)) {
               optionalParameters.add("encryption = '" + encryptionType + "'");
           }
           if (Objects.nonNull(kmsKey)) {
               optionalParameters.add("kms_key = '" + kmsKey + "'");
           }
           if (Objects.nonNull(csvFieldDelimiter)) {
               optionalParameters.add("field_delimiter = '" + csvFieldDelimiter +
"'");
           }
           if (Objects.nonNull(csvEscapeCharacter)) {
               optionalParameters.add("escaped_by = '" + csvEscapeCharacter + "'");
           }
           if (Objects.nonNull(partitionColumns) && !partitionColumns.isEmpty()) {
               final StringJoiner partitionedByList = new StringJoiner(",");
               partitionColumns.forEach(column -> partitionedByList.add("'" +
column + "'"));
               optionalParameters.add(String.format("partitioned_by = ARRAY[%s]",
partitionedByList));
           }
           withClause = String.format("WITH (%s)", optionalParameters);
       }
       return withClause;
   }

   public enum Format {
```

```
        CSV, PARQUET
    }

    public enum Compression {
        GZIP, NONE
    }

    public enum EncryptionType {
        SSE_S3, SSE_KMS
    }

    @Override
    public String toString() {
        return getUnloadQuery();
    }
}
```

Java v2

```
// When you have a SELECT like below

String QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
        + DATABASE_NAME + "." + UNLOAD_TABLE_NAME
        + " WHERE time BETWEEN ago(2d) AND now()";

//You can construct UNLOAD query as follows
UnloadQuery unloadQuery = UnloadQuery.builder()
        .selectQuery(QUERY_1)
        .bucketName("timestream-sample-<region>-<accountId>")
        .resultsPrefix("without_partition")
        .format(CSV)
        .compression(UnloadQuery.Compression.GZIP)
        .build();

QueryResponse unloadResponse = runQuery(unloadQuery.getUnloadQuery());


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
private QueryResponse runQuery(String queryString) {
   QueryResponse finalResponse = null;
```

```
    try {
        QueryRequest queryRequest =
 QueryRequest.builder().queryString(queryString).build();
        final QueryIterable queryResponseIterator =
 timestreamQueryClient.queryPaginator(queryRequest);
        for(QueryResponse queryResponse : queryResponseIterator) {
            parseQueryResult(queryResponse);
           finalResponse = queryResponse;
        }
    } catch (Exception e) {
        // Some queries might fail with 500 if the result of a sequence function has
 more than 10000 entries
        e.printStackTrace();
    }
   return finalResponse;
}

// Utility that helps to construct UNLOAD query
@Builder
static class UnloadQuery {
    private String selectQuery;
    private String bucketName;
    private String resultsPrefix;
    private Format format;
    private Compression compression;
    private EncryptionType encryptionType;
    private List<String> partitionColumns;
    private String kmsKey;
    private Character csvFieldDelimiter;
    private Character csvEscapeCharacter;

    public String getUnloadQuery() {
        String destination = constructDestination();
        String withClause = constructOptionalParameters();
        return String.format("UNLOAD (%s) TO '%s' %s", selectQuery, destination,
 withClause);
    }

    private String constructDestination() {
        return "s3://" + this.bucketName + "/" + this.resultsPrefix + "/";
    }

    private String constructOptionalParameters() {
        boolean isOptionalParametersPresent = Objects.nonNull(format)
```

```
                || Objects.nonNull(compression)
                || Objects.nonNull(encryptionType)
                || Objects.nonNull(partitionColumns)
                || Objects.nonNull(kmsKey)
                || Objects.nonNull(csvFieldDelimiter)
                || Objects.nonNull(csvEscapeCharacter);

        String withClause = "";
        if (isOptionalParametersPresent) {
            StringJoiner optionalParameters = new StringJoiner(",");
            if (Objects.nonNull(format)) {
                optionalParameters.add("format = '" + format + "'");
            }
            if (Objects.nonNull(compression)) {
                optionalParameters.add("compression = '" + compression + "'");
            }
            if (Objects.nonNull(encryptionType)) {
                optionalParameters.add("encryption = '" + encryptionType + "'");
            }
            if (Objects.nonNull(kmsKey)) {
                optionalParameters.add("kms_key = '" + kmsKey + "'");
            }
            if (Objects.nonNull(csvFieldDelimiter)) {
                optionalParameters.add("field_delimiter = '" + csvFieldDelimiter +
"'");
            }
            if (Objects.nonNull(csvEscapeCharacter)) {
                optionalParameters.add("escaped_by = '" + csvEscapeCharacter + "'");
            }
            if (Objects.nonNull(partitionColumns) && !partitionColumns.isEmpty()) {
                final StringJoiner partitionedByList = new StringJoiner(",");
                partitionColumns.forEach(column -> partitionedByList.add("'" +
column + "'"));
                optionalParameters.add(String.format("partitioned_by = ARRAY[%s]",
partitionedByList));
            }
            withClause = String.format("WITH (%s)", optionalParameters);
        }
        return withClause;
    }

    public enum Format {
        CSV, PARQUET
    }
```

```
    public enum Compression {
        GZIP, NONE
    }

    public enum EncryptionType {
        SSE_S3, SSE_KMS
    }

    @Override
    public String toString() {
        return getUnloadQuery();
    }
}
```

Go

```
// When you have a SELECT like below
var Query = "SELECT user_id, ip_address, event, session_id, measure_name, time,
 query, quantity, product_id, channel FROM "
+ *databaseName + "." + *tableName + " WHERE time BETWEEN ago(2d) AND now()"

// You can construct UNLOAD query as follows
var unloadQuery = UnloadQuery{
    Query: "SELECT user_id, ip_address, session_id, measure_name, time, query,
 quantity, product_id, channel, event FROM " + *databaseName + "." + *tableName +
    " WHERE time BETWEEN ago(2d) AND now()",
    Partitioned_by: []string{},
    Compression: "GZIP",
    Format: "CSV",
    S3Location: bucketName,
    ResultPrefix: "without_partition",
}


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination

queryInput := &timestreamquery.QueryInput{
    QueryString: build_query(unloadQuery),
}
```

```
err := querySvc.QueryPages(queryInput,
    func(page *timestreamquery.QueryOutput, lastPage bool) bool {
        if (lastPage) {
            var response = parseQueryResult(page)
            var unloadFiles = getManifestAndMetadataFiles(s3Svc, response)
            displayColumns(unloadFiles, unloadQuery.Partitioned_by)
            displayResults(s3Svc, unloadFiles)
        }
        return true
    })

if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
}

// Utility that helps to construct UNLOAD query
type UnloadQuery struct {
    Query string
    Partitioned_by []string
    Format string
    S3Location string
    ResultPrefix string
    Compression string
}

func build_query(unload_query UnloadQuery) *string {
    var query_results_s3_path = "'s3://" + unload_query.S3Location + "/" +
 unload_query.ResultPrefix + "/'"
    var query = "UNLOAD(" + unload_query.Query + ") TO " + query_results_s3_path + "
 WITH ( "
    if (len(unload_query.Partitioned_by) > 0) {
        query = query + "partitioned_by=ARRAY["
        for i, column := range unload_query.Partitioned_by {
            if i == 0 {
                query = query + "'" + column + "'"
            } else {
                query = query + ",'" + column + "'"
            }
        }
        query = query + "],"
    }
    query = query + " format='" + unload_query.Format + "', "
    query = query + "  compression='" + unload_query.Compression + "')"
```

```
    fmt.Println(query)
    return aws.String(query)
}
```

Python

```
# When you have a SELECT like below
QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel FROM "
        + database_name + "." + table_name + " WHERE time BETWEEN ago(2d) AND now()"
# You can construct UNLOAD query as follows
UNLOAD_QUERY_1 = UnloadQuery(QUERY_1, "timestream-sample-<region>-<accountId>",
 "without_partition", "CSV", "GZIP", "")

# Run UNLOAD query (Similar to how you run SELECT query)
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination
def run_query(self, query_string):
    try:
        page_iterator = self.paginator.paginate(QueryString=UNLOAD_QUERY_1)
    except Exception as err:
        print("Exception while running query:", err)

# Utility that helps to construct UNLOAD query
class UnloadQuery:
    def __init__(self, query, s3_bucket_location, results_prefix, format,
 compression , partition_by):
        self.query = query
        self.s3_bucket_location = s3_bucket_location
        self.results_prefix = results_prefix
        self.format = format
        self.compression = compression
        self.partition_by = partition_by

    def build_query(self):
        query_results_s3_path = "'s3://" + self.s3_bucket_location + "/" +
 self.results_prefix + "/'"
        unload_query = "UNLOAD("
        unload_query = unload_query + self.query
        unload_query = unload_query + ") "
        unload_query = unload_query + " TO " + query_results_s3_path
        unload_query = unload_query + " WITH ( "
```

```
        if(len(self.partition_by) > 0) :
            unload_query = unload_query + " partitioned_by = ARRAY" +
 str(self.partition_by) + ","

        unload_query = unload_query + " format='" + self.format  + "', "
        unload_query = unload_query + "  compression='" + self.compression + "')"

        return unload_query
```

Node.js

```
// When you have a SELECT like below
QUERY_1 = "SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel FROM "
        + database_name + "." + table_name + " WHERE time BETWEEN ago(2d) AND now()"
// You can construct UNLOAD query as follows
UNLOAD_QUERY_1 = new UnloadQuery(QUERY_1, "timestream-sample-<region>-<accountId>",
 "without_partition", "CSV", "GZIP", "")


// Run UNLOAD query (Similar to how you run SELECT query)
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.pagination

async runQuery(query = UNLOAD_QUERY_1, nextToken) {
    const params = new QueryCommand({
        QueryString: query
    });

    if (nextToken) {
        params.NextToken = nextToken;
    }

    await queryClient.send(params).then(
            (response) => {
                if (response.NextToken) {
                    runQuery(queryClient, query, response.NextToken);
                } else {
                    await parseAndDisplayResults(response);
                }
            },
            (err) => {
                console.error("Error while querying:", err);
```

```
            });
}


class UnloadQuery {
    constructor(query, s3_bucket_location, results_prefix, format, compression ,
 partition_by) {
        this.query = query;
        this.s3_bucket_location = s3_bucket_location
        this.results_prefix = results_prefix
        this.format = format
        this.compression = compression
        this.partition_by = partition_by
    }

    buildQuery() {
        const query_results_s3_path = "'s3://" + this.s3_bucket_location + "/" +
 this.results_prefix + "/'"
        let unload_query = "UNLOAD("
        unload_query = unload_query + this.query
        unload_query = unload_query + ") "
        unload_query = unload_query + " TO " + query_results_s3_path
        unload_query = unload_query + " WITH ( "

        if(this.partition_by.length > 0) {
            let partitionBy = ""
            this.partition_by.forEach((str, i) => {
                partitionBy = partitionBy + (i ? ",'" : "'") + str + "'"
            })
            unload_query = unload_query + " partitioned_by = ARRAY[" + partitionBy +
 "],"
        }
        unload_query = unload_query + " format='" + this.format  + "', "
        unload_query = unload_query + "  compression='" + this.compression + "')"

        return unload_query
    }
}
```

UNLOAD 응답을 구문 분석하고 행 수, 매니페스트 링크 및 메타데이터 링크 가져오기

Java

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

public UnloadResponse parseResult(QueryResult queryResult) {
    Map<String, String> outputMap = new HashMap<>();
    for (int i = 0; i < queryResult.getColumnInfo().size(); i++) {
        outputMap.put(queryResult.getColumnInfo().get(i).getName(),
                queryResult.getRows().get(0).getData().get(i).getScalarValue());

    }
    return new UnloadResponse(outputMap);
}

@Getter
class UnloadResponse {
    private final String metadataFile;
    private final String manifestFile;
    private final int rows;

    public UnloadResponse(Map<String, String> unloadResponse) {
        this.metadataFile = unloadResponse.get("metadataFile");
        this.manifestFile = unloadResponse.get("manifestFile");
        this.rows = Integer.parseInt(unloadResponse.get("rows"));
    }
}
```

Java v2

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)
```

```
public UnloadResponse parseResult(QueryResponse queryResponse) {
    Map<String, String> outputMap = new HashMap<>();
    for (int i = 0; i < queryResponse.columnInfo().size(); i++) {
        outputMap.put(queryResponse.columnInfo().get(i).name(),
                queryResponse.rows().get(0).data().get(i).scalarValue());

    }
    return new UnloadResponse(outputMap);
}

@Getter
class UnloadResponse {
    private final String metadataFile;
    private final String manifestFile;
    private final int rows;

    public UnloadResponse(Map<String, String> unloadResponse) {
        this.metadataFile = unloadResponse.get("metadataFile");
        this.manifestFile = unloadResponse.get("manifestFile");
        this.rows = Integer.parseInt(unloadResponse.get("rows"));
    }
}
```

Go

```
// Parsing UNLOAD query response is similar to how you parse SELECT query response:
// https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

// But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
// (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

func parseQueryResult(queryOutput *timestreamquery.QueryOutput) map[string]string {
    var columnInfo = queryOutput.ColumnInfo;
    fmt.Println("ColumnInfo", columnInfo)
    fmt.Println("QueryId", queryOutput.QueryId)
    fmt.Println("QueryStatus", queryOutput.QueryStatus)
    return parseResponse(columnInfo, queryOutput.Rows[0])
}

func parseResponse(columnInfo []*timestreamquery.ColumnInfo, row
 *timestreamquery.Row) map[string]string {
    var datum = row.Data
```

```
    response := make(map[string]string)
    for i, column := range columnInfo {
        response[*column.Name] = *datum[i].ScalarValue
    }
    return response
}
```

Python

```
# Parsing UNLOAD query response is similar to how you parse SELECT query response:
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing

# But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
# (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

for page in page_iterator:
  last_page = page
response = self._parse_unload_query_result(last_page)

def _parse_unload_query_result(self, query_result):
    column_info = query_result['ColumnInfo']

    print("ColumnInfo: %s" % column_info)
    print("QueryId: %s" % query_result['QueryId'])
    print("QueryStatus:%s" % query_result['QueryStatus'])
    return self.parse_unload_response(column_info, query_result['Rows'][0])

def parse_unload_response(self, column_info, row):
    response = {}
    data = row['Data']
    for i, column in enumerate(column_info):
        response[column['Name']] = data[i]['ScalarValue']
   print("Rows: %s" % response['rows'])
   print("Metadata File location: %s" % response['metadataFile'])
   print("Manifest File location: %s" % response['manifestFile'])
   return response
```

Node.js

```
# Parsing UNLOAD query response is similar to how you parse SELECT query response:
# https://docs.aws.amazon.com/timestream/latest/developerguide/code-samples.run-
query.html#code-samples.run-query.parsing
```

```
# But unlike SELECT, UNLOAD only has 1 row * 3 columns outputed
# (rows, metadataFile, manifestFile) => (BIGINT, VARCHAR, VARCHAR)

async parseAndDisplayResults(data, query) {
    const columnInfo = data['ColumnInfo'];
    console.log("ColumnInfo:", columnInfo)
    console.log("QueryId: %s", data['QueryId'])
    console.log("QueryStatus:", data['QueryStatus'])
    await this.parseResponse(columnInfo, data['Rows'][0], query)
}

async parseResponse(columnInfo, row, query) {
    let response = {}
    const data = row['Data']
    columnInfo.forEach((column, i) => {
        response[column['Name']] = data[i]['ScalarValue']
    })

   console.log("Manifest file", response['manifestFile']);
   console.log("Metadata file", response['metadataFile']);

   return response
}
```

매니페스트 콘텐츠 읽기 및 구문 분석

Java

```
// Read and parse manifest content
public UnloadManifest getUnloadManifest(UnloadResponse unloadResponse) throws
 IOException {
   AmazonS3URI s3URI = new AmazonS3URI(unloadResponse.getManifestFile());
    S3Object s3Object = s3Client.getObject(s3URI.getBucket(), s3URI.getKey());
    String manifestFileContent = new
 String(IOUtils.toByteArray(s3Object.getObjectContent()), StandardCharsets.UTF_8);
    return new Gson().fromJson(manifestFileContent, UnloadManifest.class);
}

class UnloadManifest {
    @Getter
    public class FileMetadata {
```

```
        long content_length_in_bytes;
        long row_count;
    }

    @Getter
    public class ResultFile {
        String url;
        FileMetadata file_metadata;
    }

    @Getter
    public class QueryMetadata {
        long total_content_length_in_bytes;
        long total_row_count;
        String result_format;
        String result_version;
    }

    @Getter
    public class Author {
        String name;
        String manifest_file_version;
    }

    @Getter
    private List<ResultFile> result_files;
    @Getter
    private QueryMetadata query_metadata;
    @Getter
    private Author author;
}
```

Java v2

```
// Read and parse manifest content
public UnloadManifest getUnloadManifest(UnloadResponse unloadResponse) throws
 URISyntaxException {
    // Space needs to encoded to use S3 parseUri function
    S3Uri s3Uri =
 s3Utilities.parseUri(URI.create(unloadResponse.getManifestFile().replace(" ",
 "%20")));
    ResponseBytes<GetObjectResponse> objectBytes =
 s3Client.getObjectAsBytes(GetObjectRequest.builder()
```

```
            .bucket(s3Uri.bucket().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getManifestFile(), "Invalid S3 URI")))
            .key(s3Uri.key().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getManifestFile(), "Invalid S3 URI")))
            .build());
    String manifestFileContent = new String(objectBytes.asByteArray(),
 StandardCharsets.UTF_8);
    return new Gson().fromJson(manifestFileContent, UnloadManifest.class);
}

class UnloadManifest {
    @Getter
    public class FileMetadata {
        long content_length_in_bytes;
        long row_count;
    }

    @Getter
    public class ResultFile {
        String url;
        FileMetadata file_metadata;
    }

    @Getter
    public class QueryMetadata {
        long total_content_length_in_bytes;
        long total_row_count;
        String result_format;
        String result_version;
    }

    @Getter
    public class Author {
        String name;
        String manifest_file_version;
    }

    @Getter
    private List<ResultFile> result_files;
    @Getter
    private QueryMetadata query_metadata;
    @Getter
    private Author author;
```

```
}
```

Go

```
// Read and parse manifest content

func getManifestFile(s3Svc *s3.S3, response map[string]string) Manifest {
    var manifestBuf = getObject(s3Svc, response["manifestFile"])
    var manifest Manifest
    json.Unmarshal(manifestBuf.Bytes(), &manifest)
    return manifest
}

func getObject(s3Svc *s3.S3, s3Uri string) *bytes.Buffer {
    u,_ := url.Parse(s3Uri)
    getObjectInput := &s3.GetObjectInput{
        Key:    aws.String(u.Path),
        Bucket: aws.String(u.Host),
    }
    getObjectOutput, err := s3Svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Println("Error: %s\n", err.Error())
    }
    buf := new(bytes.Buffer)
    buf.ReadFrom(getObjectOutput.Body)
    return buf
}

// Unload's Manifest structure

type Manifest struct {
    Author interface{}
    Query_metadata map[string]any
    Result_files  []struct {
        File_metadata interface{}
        Url string
    }
}}
```

Python

```
def __get_manifest_file(self, response):
    manifest = self.get_object(response['manifestFile']).read().decode('utf-8')
```

```
    parsed_manifest = json.loads(manifest)
    print("Manifest contents: \n%s" % parsed_manifest)

def get_object(self, uri):
    try:
        bucket, key = uri.replace("s3://", "").split("/", 1)
        s3_client = boto3.client('s3', region_name=<region>)
        response = s3_client.get_object(Bucket=bucket, Key=key)
        return response['Body']
    except Exception as err:
        print("Failed to get the object for URI:", uri)
        raise err
```

Node.js

```
// Read and parse manifest content

async getManifestFile(response) {
    let manifest;
    await this.getS3Object(response['manifestFile']).then(
        (data) => {
            manifest = JSON.parse(data);
        }
    );
    return manifest;
}

async getS3Object(uri) {
    const {bucketName, key} = this.getBucketAndKey(uri);
    const params = new GetObjectCommand({
        Bucket: bucketName,
        Key: key
    })
    const response = await this.s3Client.send(params);
    return await response.Body.transformToString();
}

getBucketAndKey(uri) {
    const [bucketName] = uri.replace("s3://", "").split("/", 1);
    const key = uri.replace("s3://", "").split('/').slice(1).join('/');
    return {bucketName, key};
}
```

메타데이터 콘텐츠 읽기 및 구문 분석

Java

```
// Read and parse metadata content
public UnloadMetadata getUnloadMetadata(UnloadResponse unloadResponse) throws
 IOException {
   AmazonS3URI s3URI = new AmazonS3URI(unloadResponse.getMetadataFile());
   S3Object s3Object = s3Client.getObject(s3URI.getBucket(), s3URI.getKey());
    String metadataFileContent = new
 String(IOUtils.toByteArray(s3Object.getObjectContent()), StandardCharsets.UTF_8);
    final Gson gson = new GsonBuilder()
            .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
            .create();
    return gson.fromJson(metadataFileContent, UnloadMetadata.class);
}

class UnloadMetadata {
    @JsonProperty("ColumnInfo")
    List<ColumnInfo> columnInfo;
    @JsonProperty("Author")
    Author author;

    @Data
    public class Author {
        @JsonProperty("Name")
        String name;
        @JsonProperty("MetadataFileVersion")
        String metadataFileVersion;
    }
}
```

Java v2

```
// Read and parse metadata content

public UnloadMetadata getUnloadMetadata(UnloadResponse unloadResponse) throws
 URISyntaxException {
   // Space needs to encoded to use S3 parseUri function
    S3Uri s3Uri =
 s3Utilities.parseUri(URI.create(unloadResponse.getMetadataFile().replace(" ",
 "%20")));
```

```
    ResponseBytes<GetObjectResponse> objectBytes =
 s3Client.getObjectAsBytes(GetObjectRequest.builder()
            .bucket(s3Uri.bucket().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getMetadataFile(), "Invalid S3 URI")))
            .key(s3Uri.key().orElseThrow(() -> new
 URISyntaxException(unloadResponse.getMetadataFile(), "Invalid S3 URI")))
            .build());
    String metadataFileContent = new String(objectBytes.asByteArray(),
 StandardCharsets.UTF_8);
    final Gson gson = new GsonBuilder()
            .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
            .create();
    return gson.fromJson(metadataFileContent, UnloadMetadata.class);
}

class UnloadMetadata {
    @JsonProperty("ColumnInfo")
    List<ColumnInfo> columnInfo;
    @JsonProperty("Author")
    Author author;

    @Data
    public class Author {
        @JsonProperty("Name")
        String name;
        @JsonProperty("MetadataFileVersion")
        String metadataFileVersion;
    }
}
```

Go

```
// Read and parse metadata content

func getMetadataFile(s3Svc *s3.S3, response map[string]string) Metadata {
    var metadataBuf = getObject(s3Svc, response["metadataFile"])
    var metadata Metadata
    json.Unmarshal(metadataBuf.Bytes(), &metadata)
    return metadata
}

func getObject(s3Svc *s3.S3, s3Uri string) *bytes.Buffer {
    u,_ := url.Parse(s3Uri)
```

```
    getObjectInput := &s3.GetObjectInput{
        Key:    aws.String(u.Path),
        Bucket: aws.String(u.Host),
    }
    getObjectOutput, err := s3Svc.GetObject(getObjectInput)
    if err != nil {
        fmt.Println("Error: %s\n", err.Error())
    }
    buf := new(bytes.Buffer)
    buf.ReadFrom(getObjectOutput.Body)
    return buf
}

// Unload's Metadata structure

type Metadata struct {
    Author interface{}
    ColumnInfo []struct {
        Name string
        Type map[string]string
    }
}
```

Python

```
def __get_metadata_file(self, response):
    metadata = self.get_object(response['metadataFile']).read().decode('utf-8')
    parsed_metadata = json.loads(metadata)
   print("Metadata contents: \n%s" % parsed_metadata)

def get_object(self, uri):
    try:
        bucket, key = uri.replace("s3://", "").split("/", 1)
        s3_client = boto3.client('s3', region_name=<region>)
        response = s3_client.get_object(Bucket=bucket, Key=key)
        return response['Body']
    except Exception as err:
        print("Failed to get the object for URI:", uri)
        raise err
```

Node.js

```
// Read and parse metadata content
async getMetadataFile(response) {
    let metadata;
    await this.getS3Object(response['metadataFile']).then(
        (data) => {
            metadata = JSON.parse(data);
        }
    );
    return metadata;
}

async getS3Object(uri) {
    const {bucketName, key} = this.getBucketAndKey(uri);
    const params = new GetObjectCommand({
        Bucket: bucketName,
        Key: key
    })
    const response = await this.s3Client.send(params);
    return await response.Body.transformToString();
}

getBucketAndKey(uri) {
    const [bucketName] = uri.replace("s3://", "").split("/", 1);
    const key = uri.replace("s3://", "").split('/').slice(1).join('/');
    return {bucketName, key};
}
```

쿼리 취소

다음 코드 조각을 사용하여 쿼리를 취소할 수 있습니다.

> **Note**
>
> 이러한 코드 조각은 의 전체 샘플 애플리케이션을 기반으로 합니다GitHub. 샘플 애플리케이션을 시작하는 방법에 대한 자세한 내용은 섹션을 참조하세요샘플 애플리케이션.

Java

```
   public void cancelQuery() {
       System.out.println("Starting query: " + SELECT_ALL_QUERY);
       QueryRequest queryRequest = new QueryRequest();
       queryRequest.setQueryString(SELECT_ALL_QUERY);
       QueryResult queryResult = queryClient.query(queryRequest);

       System.out.println("Cancelling the query: " + SELECT_ALL_QUERY);
       final CancelQueryRequest cancelQueryRequest = new CancelQueryRequest();
       cancelQueryRequest.setQueryId(queryResult.getQueryId());
       try {
           queryClient.cancelQuery(cancelQueryRequest);
           System.out.println("Query has been successfully cancelled");
       } catch (Exception e) {
           System.out.println("Could not cancel the query: " + SELECT_ALL_QUERY + "
= " + e);
       }
   }
```

Java v2

```
   public void cancelQuery() {
       System.out.println("Starting query: " + SELECT_ALL_QUERY);
       QueryRequest queryRequest =
QueryRequest.builder().queryString(SELECT_ALL_QUERY).build();
       QueryResponse queryResponse = timestreamQueryClient.query(queryRequest);

       System.out.println("Cancelling the query: " + SELECT_ALL_QUERY);
       final CancelQueryRequest cancelQueryRequest = CancelQueryRequest.builder()
               .queryId(queryResponse.queryId()).build();
       try {
           timestreamQueryClient.cancelQuery(cancelQueryRequest);
           System.out.println("Query has been successfully cancelled");
       } catch (Exception e) {
           System.out.println("Could not cancel the query: " + SELECT_ALL_QUERY + "
= " + e);
       }
   }
```

Go

```
cancelQueryInput := &timestreamquery.CancelQueryInput{
```

```
    QueryId: aws.String(*queryOutput.QueryId),
}

fmt.Println("Submitting cancellation for the query")
fmt.Println(cancelQueryInput)

// submit the query
cancelQueryOutput, err := querySvc.CancelQuery(cancelQueryInput)

if err != nil {
    fmt.Println("Error:")
    fmt.Println(err)
} else {
  fmt.Println("Query has been cancelled successfully")
  fmt.Println(cancelQueryOutput)
}
```

Python

```
  def cancel_query(self):
      print("Starting query: " + self.SELECT_ALL)
      result = self.client.query(QueryString=self.SELECT_ALL)
      print("Cancelling query: " + self.SELECT_ALL)
      try:
          self.client.cancel_query(QueryId=result['QueryId'])
          print("Query has been successfully cancelled")
      except Exception as err:
          print("Cancelling query failed:", err)
```

Node.js

다음 코드 조각은 JavaScript V2 스타일에 대해 를 AWS SDK 사용합니다. 이는 Node.js 샘플 Amazon Timestream의 샘플 애플리케이션을 기반으로 LiveAnalytics 합니다 GitHub.

```
async function tryCancelQuery() {
    const params = {
        QueryString: SELECT_ALL_QUERY
    };
    console.log(`Running query: ${SELECT_ALL_QUERY}`);

    await queryClient.query(params).promise()
        .then(
            async (response) => {
```

```
                await cancelQuery(response.QueryId);
            },
            (err) => {
                console.error("Error while executing select all query:", err);
            });
}

async function cancelQuery(queryId) {
    const cancelParams = {
        QueryId: queryId
    };
    console.log(`Sending cancellation for query: ${SELECT_ALL_QUERY}`);
    await queryClient.cancelQuery(cancelParams).promise()
        .then(
            (response) => {
                console.log("Query has been cancelled successfully");
            },
            (err) => {
                console.error("Error while cancelling select all:", err);
            });
}
```

.NET

```
        public async Task CancelQuery()
        {
            Console.WriteLine("Starting query: " + SELECT_ALL_QUERY);
            QueryRequest queryRequest = new QueryRequest();
            queryRequest.QueryString = SELECT_ALL_QUERY;
            QueryResponse queryResponse = await
 queryClient.QueryAsync(queryRequest);

            Console.WriteLine("Cancelling query: " + SELECT_ALL_QUERY);
            CancelQueryRequest cancelQueryRequest = new CancelQueryRequest();
            cancelQueryRequest.QueryId = queryResponse.QueryId;

            try
            {
                await queryClient.CancelQueryAsync(cancelQueryRequest);
                Console.WriteLine("Query has been successfully cancelled.");
            } catch(Exception e)
            {
```

```
                Console.WriteLine("Could not cancel the query: " + SELECT_ALL_QUERY
 + " = " + e);
            }
        }
```

배치 로드 작업 생성

다음 코드 조각을 사용하여 배치 로드 작업을 생성할 수 있습니다.

Java

```
package com.example.tryit;

import java.util.Arrays;

import
 software.amazon.awssdk.services.timestreamwrite.model.CreateBatchLoadTaskRequest;
import
 software.amazon.awssdk.services.timestreamwrite.model.CreateBatchLoadTaskResponse;
import software.amazon.awssdk.services.timestreamwrite.model.DataModel;
import software.amazon.awssdk.services.timestreamwrite.model.DataModelConfiguration;
import
 software.amazon.awssdk.services.timestreamwrite.model.DataSourceConfiguration;
import
 software.amazon.awssdk.services.timestreamwrite.model.DataSourceS3Configuration;
import software.amazon.awssdk.services.timestreamwrite.model.DimensionMapping;
import
 software.amazon.awssdk.services.timestreamwrite.model.MultiMeasureAttributeMapping;
import software.amazon.awssdk.services.timestreamwrite.model.MultiMeasureMappings;
import software.amazon.awssdk.services.timestreamwrite.model.ReportConfiguration;
import software.amazon.awssdk.services.timestreamwrite.model.ReportS3Configuration;
import software.amazon.awssdk.services.timestreamwrite.model.ScalarMeasureValueType;
import software.amazon.awssdk.services.timestreamwrite.model.TimeUnit;
import software.amazon.awssdk.services.timestreamwrite.TimestreamWriteClient;

public class BatchLoadExample {
    public static final String DATABASE_NAME = <database name>;
    public static final String TABLE_NAME = <table name>;
    public static final String INPUT_BUCKET = <S3 location>;
    public static final String INPUT_OBJECT_KEY_PREFIX = <CSV filename>;
    public static final String REPORT_BUCKET = <S3 location>;
    public static final long HT_TTL_HOURS = 24L;
```

```
    public static final long CT_TTL_DAYS = 7L;

    TimestreamWriteClient amazonTimestreamWrite;

    public BatchLoadExample(TimestreamWriteClient client) {
        this.amazonTimestreamWrite = client;
    }

    public String createBatchLoadTask() {
        System.out.println("Creating batch load task");

        CreateBatchLoadTaskRequest request = CreateBatchLoadTaskRequest.builder()
                .dataModelConfiguration(DataModelConfiguration.builder()
                        .dataModel(DataModel.builder()
                                .timeColumn("timestamp")
                                .timeUnit(TimeUnit.SECONDS)
                                .dimensionMappings(Arrays.asList(
                                        DimensionMapping.builder()
                                                .sourceColumn("vehicle")
                                                .build(),
                                        DimensionMapping.builder()
                                                .sourceColumn("registration")
                                                .destinationColumn("license")
                                                .build()))
                                .multiMeasureMappings(MultiMeasureMappings.builder()
                                        .targetMultiMeasureName("mva_measure_name")

.multiMeasureAttributeMappings(Arrays.asList(

MultiMeasureAttributeMapping.builder()
                                                        .sourceColumn("wgt")

.targetMultiMeasureAttributeName("weight")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                        .build(),

MultiMeasureAttributeMapping.builder()
                                                        .sourceColumn("spd")

.targetMultiMeasureAttributeName("speed")

.measureValueType(ScalarMeasureValueType.DOUBLE)
                                                        .build(),
```

```
 MultiMeasureAttributeMapping.builder()
                                                        .sourceColumn("fuel")

 .measureValueType(ScalarMeasureValueType.DOUBLE)
                                                        .build(),

 MultiMeasureAttributeMapping.builder()
                                                        .sourceColumn("miles")

 .measureValueType(ScalarMeasureValueType.DOUBLE)
                                                        .build()))
                                        .build())
                                .build())
                        .build())
                .dataSourceConfiguration(DataSourceConfiguration.builder()
                        .dataSourceS3Configuration(
                                DataSourceS3Configuration.builder()
                                        .bucketName(INPUT_BUCKET)
                                        .objectKeyPrefix(INPUT_OBJECT_KEY_PREFIX)
                                        .build())
                        .dataFormat("CSV")
                        .build())
                .reportConfiguration(ReportConfiguration.builder()
                        .reportS3Configuration(ReportS3Configuration.builder()
                                .bucketName(REPORT_BUCKET)
                                .build())
                        .build())
                .targetDatabaseName(DATABASE_NAME)
                .targetTableName(TABLE_NAME)
                .build();
        try {
            final CreateBatchLoadTaskResponse createBatchLoadTaskResponse =
 amazonTimestreamWrite.createBatchLoadTask(request);
            String taskId = createBatchLoadTaskResponse.taskId();
            System.out.println("Successfully created batch load task: " + taskId);
            return taskId;
        } catch (Exception e) {
            System.out.println("Failed to create batch load task: " + e);
            throw e;
        }
    }
}
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite/types"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{})(aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID &&  region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &  aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.CreateBatchLoadTask(context.TODO(), &
 timestreamwrite.CreateBatchLoadTaskInput{
            TargetDatabaseName: aws.String("BatchLoadExampleDatabase"),
            TargetTableName: aws.String("BatchLoadExampleTable"),
            RecordVersion: aws.Int64(1),
            DataModelConfiguration: &  types.DataModelConfiguration{
                DataModel: &  types.DataModel{
```

```
                    TimeColumn: aws.String("timestamp"),
                    TimeUnit: types.TimeUnitMilliseconds,
                    DimensionMappings: []types.DimensionMapping{
                        {
                            SourceColumn: aws.String("registration"),
                            DestinationColumn: aws.String("license"),
                        },
                    },
                    MultiMeasureMappings: &  types.MultiMeasureMappings{
                        TargetMultiMeasureName: aws.String("mva_measure_name"),
                        MultiMeasureAttributeMappings:
[]types.MultiMeasureAttributeMapping{
                            {
                                SourceColumn: aws.String("wgt"),
                                TargetMultiMeasureAttributeName:
aws.String("weight"),
                                MeasureValueType:
types.ScalarMeasureValueTypeDouble,
                            },
                            {
                                SourceColumn: aws.String("spd"),
                                TargetMultiMeasureAttributeName:
aws.String("speed"),
                                MeasureValueType:
types.ScalarMeasureValueTypeDouble,
                            },
                            {
                                SourceColumn: aws.String("fuel_consumption"),
                                TargetMultiMeasureAttributeName: aws.String("fuel"),
                                MeasureValueType:
types.ScalarMeasureValueTypeDouble,
                            },
                        },
                    },
                },
            },
           DataSourceConfiguration: &  types.DataSourceConfiguration{
                DataSourceS3Configuration: &  types.DataSourceS3Configuration{
                    BucketName: aws.String("test-batch-load-west-2"),
                    ObjectKeyPrefix: aws.String("sample.csv"),
                },
                DataFormat: types.BatchLoadDataFormatCsv,
            },
           ReportConfiguration: & types.ReportConfiguration{
```

```
                ReportS3Configuration: & types.ReportS3Configuration{
                    BucketName: aws.String("test-batch-load-report-west-2"),
                    EncryptionOption: types.S3EncryptionOptionSseS3,
                },
            },
 })

 fmt.Println(aws.ToString(response.TaskId))
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT = "<URL>"
REGION = "us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
DATABASE_NAME = "<database name>"
TABLE_NAME = "<table name>"
INPUT_BUCKET_NAME = "<S3 location>"
INPUT_OBJECT_KEY_PREFIX = "<CSV file name>"
REPORT_BUCKET_NAME = "<S3 location>"


def create_batch_load_task(client, database_name, table_name, input_bucket_name,
 input_object_key_prefix, report_bucket_name):
    try:
        result = client.create_batch_load_task(TargetDatabaseName=database_name,
 TargetTableName=table_name,
                                               DataModelConfiguration={"DataModel":
 {
                                                   "TimeColumn": "timestamp",
                                                   "TimeUnit": "SECONDS",
                                                   "DimensionMappings": [
                                                       {
                                                           "SourceColumn": "vehicle"
                                                       },
                                                       {
                                                           "SourceColumn":
 "registration",
```

```
                                                        "DestinationColumn":
"license"
                                                    }
                                                ],
                                                "MultiMeasureMappings": {
                                                    "TargetMultiMeasureName":
"metrics",

"MultiMeasureAttributeMappings": [
                                                        {
                                                            "SourceColumn":
"wgt",
                                                            "MeasureValueType":
"DOUBLE"
                                                        },
                                                        {
                                                            "SourceColumn":
"spd",
                                                            "MeasureValueType":
"DOUBLE"
                                                        },
                                                        {
                                                            "SourceColumn":
"fuel_consumption",

"TargetMultiMeasureAttributeName": "fuel",
                                                            "MeasureValueType":
"DOUBLE"
                                                        },
                                                        {
                                                            "SourceColumn":
"miles",
                                                            "MeasureValueType":
"DOUBLE"
                                                        }
                                                    ]}
                                            }
                                            },
                                            DataSourceConfiguration={
                                                "DataSourceS3Configuration": {
                                                    "BucketName":
input_bucket_name,
                                                    "ObjectKeyPrefix":
input_object_key_prefix
```

```
                                                    },
                                                    "DataFormat": "CSV"
                                                },
                                                ReportConfiguration={
                                                    "ReportS3Configuration": {
                                                        "BucketName":
 report_bucket_name,
                                                        "EncryptionOption": "SSE_S3"
                                                    }
                                                }
                                                )

        task_id = result["TaskId"]
        print("Successfully created batch load task: ", task_id)
        return task_id
    except Exception as err:
        print("Create batch load task job failed:", err)
        return None


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write',
                                  endpoint_url=INGEST_ENDPOINT, region_name=REGION,
                                  config=Config(read_timeout=20,
 max_pool_connections=5000, retries={'max_attempts': 10}))

    task_id = create_batch_load_task(write_client, DATABASE_NAME, TABLE_NAME,
                                     INPUT_BUCKET_NAME, INPUT_OBJECT_KEY_PREFIX,
 REPORT_BUCKET_NAME)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

자세한 API 내용은 클래스 CreateBatchLoadCommand 및 섹션을 참조하세요CreateBatchLoadTask.

```
import { TimestreamWriteClient, CreateBatchLoadTaskCommand } from "@aws-sdk/client-
timestream-write";
```

```
const writeClient = new TimestreamWriteClient({ region: "us-west-2", endpoint:
 "https://gamma-ingest-cell3.timestream.us-west-2.amazonaws.com" });

const params = {
    TargetDatabaseName: "BatchLoadExampleDatabase",
    TargetTableName: "BatchLoadExampleTable",
 RecordVersion: 1,
 DataModelConfiguration: {
  DataModel: {
   TimeColumn: "timestamp",
            TimeUnit: "MILLISECONDS",
            DimensionMappings: [
                {
                    SourceColumn: "registration",
                    DestinationColumn: "license"
                }
            ],
            MultiMeasureMappings: {
                TargetMultiMeasureName: "mva_measure_name",
                MultiMeasureAttributeMappings: [
                    {
                        SourceColumn: "wgt",
                        TargetMultiMeasureAttributeName: "weight",
                        MeasureValueType: "DOUBLE"
                    },
                    {
                        SourceColumn: "spd",
                        TargetMultiMeasureAttributeName: "speed",
                        MeasureValueType: "DOUBLE"
                    },
                    {
                        SourceColumn: "fuel_consumption",
                        TargetMultiMeasureAttributeName: "fuel",
                        MeasureValueType: "DOUBLE"
                    }
                ]
            }
        }
    },
 DataSourceConfiguration: {
        DataSourceS3Configuration: {
            BucketName: "test-batch-load-west-2",
            ObjectKeyPrefix: "sample.csv"
        },
```

```
        DataFormat: "CSV"
    },
    ReportConfiguration: {
        ReportS3Configuration: {
            BucketName: "test-batch-load-report-west-2",
            EncryptionOption: "SSE_S3"
        }
    }
};

const command = new CreateBatchLoadTaskCommand(params);

try {
    const data = await writeClient.send(command);
    console.log(`Created batch load task ` + data.TaskId);
} catch (error) {
 console.log("Error creating table. ", error);
 throw error;
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class CreateBatchLoadTaskExample
    {
        public const string DATABASE_NAME = "<database name>";
        public const string TABLE_NAME = "<table name>";
        public const string INPUT_BUCKET = "<input bucket name>";
        public const string INPUT_OBJECT_KEY_PREFIX = "<CSV file name>";
        public const string REPORT_BUCKET = "<report bucket name>";
        public const long HT_TTL_HOURS = 24L;
        public const long CT_TTL_DAYS = 7L;
        private readonly AmazonTimestreamWriteClient writeClient;

        public CreateBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
```

```
        {
            this.writeClient = writeClient;
        }

        public async Task CreateBatchLoadTask()
        {
            try
            {
                var createBatchLoadTaskRequest = new CreateBatchLoadTaskRequest
                {
                    DataModelConfiguration = new DataModelConfiguration
                    {
                        DataModel = new DataModel
                        {
                            TimeColumn = "timestamp",
                            TimeUnit = TimeUnit.SECONDS,
                            DimensionMappings = new List<DimensionMapping>()
                            {
                                new()
                                {
                                        SourceColumn = "vehicle"
                                },
                                new()
                                {
                                        SourceColumn = "registration",
                                        DestinationColumn = "license"
                                }
                            },
                            MultiMeasureMappings = new MultiMeasureMappings
                            {
                                TargetMultiMeasureName = "mva_measure_name",
                                MultiMeasureAttributeMappings = new
List<MultiMeasureAttributeMapping>()
                                {
                                        new()
                                        {
                                                SourceColumn = "wgt",
                                                TargetMultiMeasureAttributeName =
"weight",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                        },
                                        new()
                                        {
```

```
                                                SourceColumn = "spd",
                                                TargetMultiMeasureAttributeName =
"speed",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                            },
                                            new()
                                            {
                                                SourceColumn = "fuel",
                                                TargetMultiMeasureAttributeName =
"fuel",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                            },
                                            new()
                                            {
                                                SourceColumn = "miles",
                                                TargetMultiMeasureAttributeName =
"miles",
                                                MeasureValueType =
ScalarMeasureValueType.DOUBLE
                                            }
                                    }
                                }
                            }
                        },
                        DataSourceConfiguration = new DataSourceConfiguration
                        {
                            DataSourceS3Configuration = new DataSourceS3Configuration
                            {
                                BucketName = INPUT_BUCKET,
                                ObjectKeyPrefix = INPUT_OBJECT_KEY_PREFIX
                            },
                            DataFormat = "CSV"
                        },
                        ReportConfiguration = new ReportConfiguration
                        {
                            ReportS3Configuration = new ReportS3Configuration
                            {
                                BucketName = REPORT_BUCKET
                            }
                        },
                        TargetDatabaseName = DATABASE_NAME,
                        TargetTableName = TABLE_NAME
```

```
                };

                CreateBatchLoadTaskResponse response = await
 writeClient.CreateBatchLoadTaskAsync(createBatchLoadTaskRequest);
                Console.WriteLine($"Task created: " + response.TaskId);
            }
            catch (Exception e)
            {
                Console.WriteLine("Create batch load task failed:" + e.ToString());
            }
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
```

```
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new CreateBatchLoadTaskExample(writeClient);
            await example.CreateBatchLoadTask();
        }
    }
}
```

배치 로드 작업 설명

다음 코드 조각을 사용하여 배치 로드 작업을 설명할 수 있습니다.

Java

```
   public void describeBatchLoadTask(String taskId) {
           final DescribeBatchLoadTaskResponse batchLoadTaskResponse =
 amazonTimestreamWrite

 .describeBatchLoadTask(DescribeBatchLoadTaskRequest.builder()
                                          .taskId(taskId)
                                          .build());

           System.out.println("Task id: " +
 batchLoadTaskResponse.batchLoadTaskDescription().taskId());
           System.out.println("Status: " +
 batchLoadTaskResponse.batchLoadTaskDescription().taskStatusAsString());
           System.out.println("Records processed: "
                           +
 batchLoadTaskResponse.batchLoadTaskDescription().progressReport().recordsProcessed());
   }
```

Go

```
package main

import (
```

```
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.DescribeBatchLoadTask(context.TODO(),
 &timestreamwrite.DescribeBatchLoadTaskInput{
  TaskId: aws.String("<TaskId>"),
 })

 fmt.Println(aws.ToString(response.BatchLoadTaskDescription.TaskId))
}
```

Python

```
import boto3
from botocore.config import Config
```

```
INGEST_ENDPOINT="<url>"
REGION="us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
TASK_ID = "<task id>"

def describe_batch_load_task(client, task_id):
    try:
        result = client.describe_batch_load_task(TaskId=task_id)
        print("Successfully described batch load task: ", result)
    except Exception as err:
        print("Describe batch load task job failed:", err)


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write', \
        endpoint_url=INGEST_ENDPOINT, region_name=REGION, \
        config=Config(read_timeout=20, max_pool_connections = 5000,
 retries={'max_attempts': 10}))

    describe_batch_load_task(write_client, TASK_ID)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

API 자세한 내용은 클래스 DescribeBatchLoadCommand 및 섹션을 참조하세요DescribeBatchLoadTask.

```
import { TimestreamWriteClient, DescribeBatchLoadTaskCommand } from "@aws-sdk/
client-timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    TaskId: "<TaskId>"
};

const command = new DescribeBatchLoadTaskCommand(params);
```

```
try {
    const data = await writeClient.send(command);
    console.log(`Batch load task has id ` + data.BatchLoadTaskDescription.TaskId);
} catch (error) {
    if (error.code === 'ResourceNotFoundException') {
        console.log("Batch load task doesn't exist.");
    } else {
        console.log("Describe batch load task failed.", error);
        throw error;
    }
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class DescribeBatchLoadTaskExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public DescribeBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task DescribeBatchLoadTask(String taskId)
        {
            try
            {
                var describeBatchLoadTaskRequest = new DescribeBatchLoadTaskRequest
                {
                    TaskId = taskId
                };
                DescribeBatchLoadTaskResponse response = await
 writeClient.DescribeBatchLoadTaskAsync(describeBatchLoadTaskRequest);
```

```
                Console.WriteLine($"Task has id:
{response.BatchLoadTaskDescription.TaskId}");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Batch load task does not exist.");
            }
            catch (Exception e)
            {
                Console.WriteLine("Describe batch load task failed:" +
 e.ToString());
            }
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }
```

```
        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new DescribeBatchLoadTaskExample(writeClient);
            await example.DescribeBatchLoadTask("<batch load task id>");
        }
    }
}
```

배치 로드 작업 나열

다음 코드 조각을 사용하여 배치 로드 작업을 나열할 수 있습니다.

Java

```
    public void listBatchLoadTasks() {
            final ListBatchLoadTasksResponse listBatchLoadTasksResponse =
 amazonTimestreamWrite
                            .listBatchLoadTasks(ListBatchLoadTasksRequest.builder()
                                    .maxResults(15)
                                    .build());

            for (BatchLoadTask batchLoadTask :
 listBatchLoadTasksResponse.batchLoadTasks()) {
                    System.out.println(batchLoadTask.taskId());
            }
    }
```

Go

```
package main

import (
 "fmt"
 "context"
```

```
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)

func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)
 listBatchLoadTasksMaxResult := int32(15)

 response, err := client.ListBatchLoadTasks(context.TODO(),
 &timestreamwrite.ListBatchLoadTasksInput{
  MaxResults: &listBatchLoadTasksMaxResult,
 })

 for i, task := range response.BatchLoadTasks {
  fmt.Println(i, aws.ToString(task.TaskId))
 }
}
```

Python

```
import boto3
```

```
from botocore.config import Config

INGEST_ENDPOINT = "<url>"
REGION = "us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7


def print_batch_load_tasks(batch_load_tasks):
    for batch_load_task in batch_load_tasks:
        print(batch_load_task['TaskId'])


def list_batch_load_tasks(client):
    print("\nListing batch load tasks")
    try:
        response = client.list_batch_load_tasks(MaxResults=10)
        print_batch_load_tasks(response['BatchLoadTasks'])
        next_token = response.get('NextToken', None)
        while next_token:
            response = client.list_batch_load_tasks(
                NextToken=next_token, MaxResults=10)
            print_batch_load_tasks(response['BatchLoadTasks'])
            next_token = response.get('NextToken', None)
    except Exception as err:
        print("List batch load tasks failed:", err)
        raise err


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write',
                                  endpoint_url=INGEST_ENDPOINT, region_name=REGION,
                                  config=Config(read_timeout=20,
 max_pool_connections=5000, retries={'max_attempts': 10}))

    list_batch_load_tasks(write_client)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

API 자세한 내용은 클래스 DescribeBatchLoadCommand 및 섹션을 참조하세요DescribeBatchLoadTask.

```
import { TimestreamWriteClient, ListBatchLoadTasksCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    MaxResults: <15>
};

const command = new ListBatchLoadTasksCommand(params);

getBatchLoadTasksList(null);

async function getBatchLoadTasksList(nextToken) {
    if (nextToken) {
        params.NextToken = nextToken;
    }

    try {
        const data = await writeClient.send(command);

        data.BatchLoadTasks.forEach(function (task) {
            console.log(task.TaskId);
        });

        if (data.NextToken) {
            return getBatchLoadTasksList(data.NextToken);
        }
    } catch (error) {
        console.log("Error while listing batch load tasks", error);
    }
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
```

```
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class ListBatchLoadTasksExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public ListBatchLoadTasksExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task ListBatchLoadTasks()
        {
            Console.WriteLine("Listing batch load tasks");

            try
            {
                var listBatchLoadTasksRequest = new ListBatchLoadTasksRequest
                {
                    MaxResults = 15
                };

                ListBatchLoadTasksResponse response = await
 writeClient.ListBatchLoadTasksAsync(listBatchLoadTasksRequest);

                PrintBatchLoadTasks(response.BatchLoadTasks);
                var nextToken = response.NextToken;

                while (nextToken != null)
                {
                    listBatchLoadTasksRequest.NextToken = nextToken;
                    response = await
 writeClient.ListBatchLoadTasksAsync(listBatchLoadTasksRequest);
                    PrintBatchLoadTasks(response.BatchLoadTasks);
                    nextToken = response.NextToken;
                }
            }
            catch (Exception e)
            {
                Console.WriteLine("List batch load tasks failed:" + e.ToString());
            }
        }
```

```
        private void PrintBatchLoadTasks(List<BatchLoadTask> tasks)
        {
            foreach (BatchLoadTask task in tasks)
                Console.WriteLine($"Task:{task.TaskId}");
        }
    }
}
```

```
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using Amazon;
using Amazon.TimestreamQuery;
using System.Threading.Tasks;
using System;
using CommandLine;
static class Constants
{

}
namespace TimestreamDotNetSample
{
    class MainClass
    {
        public class Options
        {

        }
        public static void Main(string[] args)
        {
            Parser.Default.ParseArguments<Options>(args)
                .WithParsed<Options>(o => {
                    MainAsync().GetAwaiter().GetResult();
                });
        }

        static async Task MainAsync()
        {
            var writeClientConfig = new AmazonTimestreamWriteConfig
            {
                ServiceURL =  "<service URL>",
                Timeout = TimeSpan.FromSeconds(20),
                MaxErrorRetry = 10
```

```
            };

            var writeClient = new AmazonTimestreamWriteClient(writeClientConfig);
            var example = new ListBatchLoadTasksExample(writeClient);
            await example.ListBatchLoadTasks();
        }
    }
}
```

배치 로드 작업 재개

다음 코드 조각을 사용하여 배치 로드 작업을 재개할 수 있습니다.

Java

```
    public void resumeBatchLoadTask(String taskId) {
            try {
                    amazonTimestreamWrite

 .resumeBatchLoadTask(ResumeBatchLoadTaskRequest.builder()
                                                    .taskId(taskId)
                                                    .build());

                    System.out.println("Successfully resumed batch load task.");
            } catch (ValidationException validationException) {
                    System.out.println(validationException.getMessage());
            }
    }
```

Go

```
package main

import (
 "fmt"
 "context"
 "log"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/timestreamwrite"
)
```

```
func main() {
 customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
  if service == timestreamwrite.ServiceID && region == "us-west-2" {
      return aws.Endpoint{
          PartitionID:   "aws",
          URL:           <URL>,
          SigningRegion: "us-west-2",
      }, nil
  }
  return aws.Endpoint{}, &aws.EndpointNotFoundError{}
 })

 cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver), config.WithRegion("us-
west-2"))

 if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
 }

 client := timestreamwrite.NewFromConfig(cfg)

 response, err := client.ResumeBatchLoadTask(context.TODO(),
 &timestreamwrite.ResumeBatchLoadTaskInput{
  TaskId: aws.String("TaskId"),
 })

 if err != nil {
  fmt.Println("Error:")
  fmt.Println(err)
 } else {
  fmt.Println("Resume batch load task is successful")
  fmt.Println(response)
 }
}
```

Python

```
import boto3
from botocore.config import Config

INGEST_ENDPOINT="<url>"
```

```
REGION="us-west-2"
HT_TTL_HOURS = 24
CT_TTL_DAYS = 7
TASK_ID = "<TaskId>"

def resume_batch_load_task(client, task_id):
    try:
        result = client.resume_batch_load_task(TaskId=task_id)
        print("Successfully resumed batch load task: ", result)
    except Exception as err:
        print("Resume batch load task failed:", err)


if __name__ == '__main__':
    session = boto3.Session()

    write_client = session.client('timestream-write', \
        endpoint_url=INGEST_ENDPOINT, region_name=REGION, \
        config=Config(read_timeout=20, max_pool_connections = 5000,
 retries={'max_attempts': 10}))

    resume_batch_load_task(write_client, TASK_ID)
```

Node.js

다음 조각은 JavaScript v3AWSSDK에 를 사용합니다. 클라이언트를 설치하는 방법과 사용량에 대한 자세한 내용은 JavaScript v3AWSSDK용 Timestream Write Client -를 참조하세요.

자세한 API 내용은 클래스 CreateBatchLoadCommand 및 섹션을 참조하세요CreateBatchLoadTask.

```
import { TimestreamWriteClient, ResumeBatchLoadTaskCommand } from "@aws-sdk/client-
timestream-write";
const writeClient = new TimestreamWriteClient({ region: "<region>", endpoint:
 "<endpoint>" });

const params = {
    TaskId: "<TaskId>"
};

const command = new ResumeBatchLoadTaskCommand(params);

try {
```

```
    const data = await writeClient.send(command);
    console.log("Resumed batch load task");
} catch (error) {
    console.log("Resume batch load task failed.", error);
 throw error;
}
```

.NET

```
using System;
using System.IO;
using System.Collections.Generic;
using Amazon.TimestreamWrite;
using Amazon.TimestreamWrite.Model;
using System.Threading.Tasks;

namespace TimestreamDotNetSample
{
    public class ResumeBatchLoadTaskExample
    {
        private readonly AmazonTimestreamWriteClient writeClient;

        public ResumeBatchLoadTaskExample(AmazonTimestreamWriteClient writeClient)
        {
            this.writeClient = writeClient;
        }

        public async Task ResumeBatchLoadTask(String taskId)
        {
            try
            {
                var resumeBatchLoadTaskRequest = new ResumeBatchLoadTaskRequest
                {
                    TaskId = taskId
                };
                ResumeBatchLoadTaskResponse response = await
 writeClient.ResumeBatchLoadTaskAsync(resumeBatchLoadTaskRequest);
                Console.WriteLine("Successfully resumed batch load task.");
            }
            catch (ResourceNotFoundException)
            {
                Console.WriteLine("Batch load task does not exist.");
            }
```

```
            catch (Exception e)
            {
                Console.WriteLine("Resume batch load task failed: " + e.ToString());
            }
        }
    }
}
```

예약된 쿼리 생성

다음 코드 조각을 사용하여 다중 측정 매핑으로 예약된 쿼리를 생성할 수 있습니다.

Java

```
public static String DATABASE_NAME = "devops_sample_application";
public static String TABLE_NAME = "host_metrics_sample_application";
public static String HOSTNAME = "host-24Gju";
public static String SQ_NAME = "daily-sample";
public static String SCHEDULE_EXPRESSION = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public static String QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
"ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
"FROM " +  DATABASE_NAME + "." +  TABLE_NAME + " " +
"WHERE measure_name = 'metrics' " +
"AND hostname = '" + HOSTNAME + "' " +
"AND time > ago(2h) " +
"GROUP BY region, hostname, az, BIN(time, 15s) " +
"ORDER BY binned_timestamp ASC " +
"LIMIT 5";


public String createScheduledQuery(String topic_arn,
    String role_arn,
    String database_name,
    String table_name) {
    System.out.println("Creating Scheduled Query");
```

```
   List<Pair<String, MeasureValueType>> sourceColToMeasureValueTypes =
Arrays.asList(
       Pair.of("avg_cpu_utilization", DOUBLE),
       Pair.of("p90_cpu_utilization", DOUBLE),
       Pair.of("p95_cpu_utilization", DOUBLE),
       Pair.of("p99_cpu_utilization", DOUBLE));

   CreateScheduledQueryRequest createScheduledQueryRequest = new
CreateScheduledQueryRequest()
           .withName(SQ_NAME)
           .withQueryString(QUERY)
           .withScheduleConfiguration(new ScheduleConfiguration()
                   .withScheduleExpression(SCHEDULE_EXPRESSION))
           .withNotificationConfiguration(new NotificationConfiguration()
                   .withSnsConfiguration(new SnsConfiguration()
                           .withTopicArn(topic_arn)))
           .withTargetConfiguration(new
TargetConfiguration().withTimestreamConfiguration(new TimestreamConfiguration()
                   .withDatabaseName(database_name)
                   .withTableName(table_name)
                   .withTimeColumn("binned_timestamp")
                   .withDimensionMappings(Arrays.asList(
                           new DimensionMapping()
                                   .withName("region")
                                   .withDimensionValueType("VARCHAR"),
                           new DimensionMapping()
                                   .withName("az")
                                   .withDimensionValueType("VARCHAR"),
                           new DimensionMapping()
                                   .withName("hostname")
                                   .withDimensionValueType("VARCHAR")
                   ))
                   .withMultiMeasureMappings(new MultiMeasureMappings()
                       .withTargetMultiMeasureName("multi-metrics")
                       .withMultiMeasureAttributeMappings(
                           sourceColToMeasureValueTypes.stream()
                           .map(pair -> new MultiMeasureAttributeMapping()
                               .withMeasureValueType(pair.getValue().name())
                               .withSourceColumn(pair.getKey()))
                           .collect(Collectors.toList()))))
           .withErrorReportConfiguration(new ErrorReportConfiguration()
                   .withS3Configuration(new S3Configuration()
```

```
 .withBucketName(timestreamDependencyHelper.getS3ErrorReportBucketName())))
            .withScheduledQueryExecutionRoleArn(role_arn);

    try {
        final CreateScheduledQueryResult createScheduledQueryResult =
 queryClient.createScheduledQuery(createScheduledQueryRequest);
        final String scheduledQueryArn = createScheduledQueryResult.getArn();
        System.out.println("Successfully created scheduled query : " +
 scheduledQueryArn);
        return scheduledQueryArn;
    }
    catch (Exception e) {
        System.out.println("Scheduled Query creation failed: " + e);
        throw e;
    }
}
```

Java v2

```
public static String DATABASE_NAME = "testJavaV2DB";
public static String TABLE_NAME = "testJavaV2Table";
public static String HOSTNAME = "host-24Gju";
public static String SQ_NAME = "daily-sample";
public static String SCHEDULE_EXPRESSION = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public static String VALID_QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
"ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
"ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
"FROM " +  DATABASE_NAME + "." +  TABLE_NAME + " " +
"WHERE measure_name = 'metrics' " +
"AND hostname = '" + HOSTNAME + "' " +
"AND time > ago(2h) " +
"GROUP BY region, hostname, az, BIN(time, 15s) " +
"ORDER BY binned_timestamp ASC " +
"LIMIT 5";
```

```
private String createScheduledQueryHelper(String topicArn, String roleArn,
        String s3ErrorReportBucketName, String query,
        TargetConfiguration targetConfiguration) {
    System.out.println("Creating Scheduled Query");

    CreateScheduledQueryRequest createScheduledQueryRequest =
 CreateScheduledQueryRequest.builder()
            .name(SQ_NAME)
            .queryString(query)
            .scheduleConfiguration(ScheduleConfiguration.builder()
                    .scheduleExpression(SCHEDULE_EXPRESSION)
                    .build())
            .notificationConfiguration(NotificationConfiguration.builder()
                    .snsConfiguration(SnsConfiguration.builder()
                            .topicArn(topicArn)
                            .build())
                    .build())
            .targetConfiguration(targetConfiguration)
            .errorReportConfiguration(ErrorReportConfiguration.builder()
                    .s3Configuration(S3Configuration.builder()
                            .bucketName(s3ErrorReportBucketName)
                            .objectKeyPrefix(SCHEDULED_QUERY_EXAMPLE)
                            .build())
                    .build())
            .scheduledQueryExecutionRoleArn(roleArn)
            .build();

    try {
        final CreateScheduledQueryResponse response =
 queryClient.createScheduledQuery(createScheduledQueryRequest);
        final String scheduledQueryArn = response.arn();
        System.out.println("Successfully created scheduled query : " +
 scheduledQueryArn);
        return scheduledQueryArn;
    }
    catch (Exception e) {
        System.out.println("Scheduled Query creation failed: " + e);
        throw e;
    }
}

public String createScheduledQuery(String topicArn, String roleArn,
        String databaseName, String tableName, String s3ErrorReportBucketName) {
```

```
   List<Pair<String, MeasureValueType>> sourceColToMeasureValueTypes =
Arrays.asList(
           Pair.of("avg_cpu_utilization", DOUBLE),
           Pair.of("p90_cpu_utilization", DOUBLE),
           Pair.of("p95_cpu_utilization", DOUBLE),
           Pair.of("p99_cpu_utilization", DOUBLE));

   TargetConfiguration targetConfiguration = TargetConfiguration.builder()
           .timestreamConfiguration(TimestreamConfiguration.builder()
           .databaseName(databaseName)
           .tableName(tableName)
           .timeColumn("binned_timestamp")
           .dimensionMappings(Arrays.asList(
                   DimensionMapping.builder()
                           .name("region")
                           .dimensionValueType("VARCHAR")
                           .build(),
                   DimensionMapping.builder()
                           .name("az")
                           .dimensionValueType("VARCHAR")
                           .build(),
                   DimensionMapping.builder()
                           .name("hostname")
                           .dimensionValueType("VARCHAR")
                           .build()
           ))
           .multiMeasureMappings(MultiMeasureMappings.builder()
                   .targetMultiMeasureName("multi-metrics")
                   .multiMeasureAttributeMappings(
                           sourceColToMeasureValueTypes.stream()
                                   .map(pair ->
MultiMeasureAttributeMapping.builder()

.measureValueType(pair.getValue().name())
                                           .sourceColumn(pair.getKey())
                                           .build())
                                   .collect(Collectors.toList()))
                   .build())
           .build())
           .build();

   return createScheduledQueryHelper(topicArn, roleArn, s3ErrorReportBucketName,
VALID_QUERY, targetConfiguration);
```

```
}}
```

Go

```
SQ_ERROR_CONFIGURATION_S3_BUCKET_NAME_PREFIX = "sq-error-configuration-sample-s3-
bucket-"
HOSTNAME            = "host-24Gju"
SQ_NAME             = "daily-sample"
SCHEDULE_EXPRESSION = "cron(0/1 * * * ? *)"
QUERY               = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
    "ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
    "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
    "FROM %s.%s " +
    "WHERE measure_name = 'metrics' " +
    "AND hostname = '" + HOSTNAME + "' " +
    "AND time > ago(2h) " +
    "GROUP BY region, hostname, az, BIN(time, 15s) " +
    "ORDER BY binned_timestamp ASC " +
    "LIMIT 5"
s3BucketName = utils.SQ_ERROR_CONFIGURATION_S3_BUCKET_NAME_PREFIX +
 generateRandomStringWithSize(5)

func generateRandomStringWithSize(size int) string {
     rand.Seed(time.Now().UnixNano())
     alphaNumericList := []rune("abcdefghijklmnopqrstuvwxyz0123456789")
     randomPrefix := make([]rune, size)
     for i := range randomPrefix {
         randomPrefix[i] = alphaNumericList[rand.Intn(len(alphaNumericList))]
     }
     return string(randomPrefix)
 }

func (timestreamBuilder TimestreamBuilder) createScheduledQuery(topicArn string,
 roleArn string, s3ErrorReportBucketName string,
query string, targetConfiguration timestreamquery.TargetConfiguration) (string,
 error) {

createScheduledQueryInput := &timestreamquery.CreateScheduledQueryInput{
    Name:        aws.String(SQ_NAME),
    QueryString: aws.String(query),
```

```
    ScheduleConfiguration: &timestreamquery.ScheduleConfiguration{
        ScheduleExpression: aws.String(SCHEDULE_EXPRESSION),
    },
    NotificationConfiguration: &timestreamquery.NotificationConfiguration{
        SnsConfiguration: &timestreamquery.SnsConfiguration{
            TopicArn: aws.String(topicArn),
        },
    },
    TargetConfiguration: &targetConfiguration,
    ErrorReportConfiguration: &timestreamquery.ErrorReportConfiguration{
        S3Configuration: &timestreamquery.S3Configuration{
            BucketName: aws.String(s3ErrorReportBucketName),
        },
    },
    ScheduledQueryExecutionRoleArn: aws.String(roleArn),
}

createScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.CreateScheduledQuery(createScheduledQueryInput)

if err != nil {
    fmt.Printf("Error: %s", err.Error())
} else {
    fmt.Println("createScheduledQueryResult is successful")
    return *createScheduledQueryOutput.Arn, nil
}
return "", err
}

 func (timestreamBuilder TimestreamBuilder) CreateValidScheduledQuery(topicArn
 string, roleArn string, s3ErrorReportBucketName string,
     sqDatabaseName string, sqTableName string, databaseName string, tableName
 string) (string, error) {

     targetConfiguration := timestreamquery.TargetConfiguration{
         TimestreamConfiguration: &timestreamquery.TimestreamConfiguration{
             DatabaseName: aws.String(sqDatabaseName),
             TableName:    aws.String(sqTableName),
             TimeColumn:   aws.String("binned_timestamp"),
             DimensionMappings: []*timestreamquery.DimensionMapping{
                 {
                     Name:               aws.String("region"),
                     DimensionValueType: aws.String("VARCHAR"),
                 },
```

```
                {
                    Name:               aws.String("az"),
                    DimensionValueType: aws.String("VARCHAR"),
                },
                {
                    Name:               aws.String("hostname"),
                    DimensionValueType: aws.String("VARCHAR"),
                },
            },
            MultiMeasureMappings: &timestreamquery.MultiMeasureMappings{
                TargetMultiMeasureName: aws.String("multi-metrics"),
                MultiMeasureAttributeMappings:
[]*timestreamquery.MultiMeasureAttributeMapping{
                    {
                        SourceColumn:     aws.String("avg_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p90_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p95_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                    {
                        SourceColumn:     aws.String("p99_cpu_utilization"),
                        MeasureValueType:
aws.String(timestreamquery.MeasureValueTypeDouble),
                    },
                },
            },
        },
    }
    return timestreamBuilder.createScheduledQuery(topicArn, roleArn,
s3ErrorReportBucketName,
        fmt.Sprintf(QUERY, databaseName, tableName), targetConfiguration)
}
```

Python

```
HOSTNAME = "host-24Gju"
SQ_NAME = "daily-sample"
ERROR_BUCKET_NAME = "scheduledquerysamplerrorbucket" +
 ''.join([choice(ascii_lowercase) for _ in range(5)])
QUERY = \
    "SELECT region, az, hostname, BIN(time, 15s) AS binned_timestamp, " \
    "    ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, "
 \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization,
 " \
    "    ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization "
 \
    "FROM " + database_name + "." + table_name + " " \
    "WHERE measure_name = 'metrics' " \
    "AND hostname = '" + self.HOSTNAME + "' " \
    "AND time > ago(2h) " \
    "GROUP BY region, hostname, az, BIN(time, 15s) " \
    "ORDER BY binned_timestamp ASC " \
    "LIMIT 5"

def create_scheduled_query_helper(self, topic_arn, role_arn, query,
 target_configuration):
    print("\nCreating Scheduled Query")
    schedule_configuration = {
        'ScheduleExpression': 'cron(0/2 * * * ? *)'
    }
    notification_configuration = {
        'SnsConfiguration': {
            'TopicArn': topic_arn
        }
    }
    error_report_configuration = {
        'S3Configuration': {
            'BucketName': ERROR_BUCKET_NAME
        }
    }

    try:
        create_scheduled_query_response = \
            query_client.create_scheduled_query(Name=self.SQ_NAME,
                 QueryString=query,
```

```
                ScheduleConfiguration=schedule_configuration,
                NotificationConfiguration=notification_configuration,
                TargetConfiguration=target_configuration,
                ScheduledQueryExecutionRoleArn=role_arn,
                ErrorReportConfiguration=error_report_configuration
                )
        print("Successfully created scheduled query : ",
 create_scheduled_query_response['Arn'])
        return create_scheduled_query_response['Arn']
    except Exception as err:
        print("Scheduled Query creation failed:", err)
        raise err

def create_valid_scheduled_query(self, topic_arn, role_arn):
    target_configuration = {
        'TimestreamConfiguration': {
            'DatabaseName': self.sq_database_name,
            'TableName': self.sq_table_name,
            'TimeColumn': 'binned_timestamp',
            'DimensionMappings': [
                {'Name': 'region', 'DimensionValueType': 'VARCHAR'},
                {'Name': 'az', 'DimensionValueType': 'VARCHAR'},
                {'Name': 'hostname', 'DimensionValueType': 'VARCHAR'}
            ],
            'MultiMeasureMappings': {
                'TargetMultiMeasureName': 'target_name',
                'MultiMeasureAttributeMappings': [
                    {'SourceColumn': 'avg_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'avg_cpu_utilization'},
                    {'SourceColumn': 'p90_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p90_cpu_utilization'},
                    {'SourceColumn': 'p95_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p95_cpu_utilization'},
                    {'SourceColumn': 'p99_cpu_utilization', 'MeasureValueType':
 'DOUBLE',
                     'TargetMultiMeasureAttributeName': 'p99_cpu_utilization'},
                ]
            }
        }
    }
```

```
    return self.create_scheduled_query_helper(topic_arn, role_arn, QUERY,
 target_configuration)
```

Node.js

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
const DATABASE_NAME = 'devops_sample_application';
const TABLE_NAME = 'host_metrics_sample_application';
const SQ_DATABASE_NAME = 'sq_result_database';
const SQ_TABLE_NAME = 'sq_result_table';
const HOSTNAME = "host-24Gju";
const SQ_NAME = "daily-sample";
const SCHEDULE_EXPRESSION = "cron(0/1 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
const VALID_QUERY = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
    " ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, " +
    " ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
    "FROM " + DATABASE_NAME + "." + TABLE_NAME + " " +
    "WHERE measure_name = 'metrics' " +
    " AND hostname = '" + HOSTNAME + "' " +
    " AND time > ago(2h) " +
    "GROUP BY region, hostname, az, BIN(time, 15s) " +
    "ORDER BY binned_timestamp ASC " +
    "LIMIT 5";

async function createScheduledQuery(topicArn, roleArn, s3ErrorReportBucketName) {
    console.log("Creating Valid Scheduled Query");
    const DimensionMappingList = [{
            'Name': 'region',
            'DimensionValueType': 'VARCHAR'
        },
        {
            'Name': 'az',
            'DimensionValueType': 'VARCHAR'
        },
        {
```

```
            'Name': 'hostname',
            'DimensionValueType': 'VARCHAR'
        }
    ];

    const MultiMeasureMappings = {
        TargetMultiMeasureName: "multi-metrics",
        MultiMeasureAttributeMappings: [{
                'SourceColumn': 'avg_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p90_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p95_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
            {
                'SourceColumn': 'p99_cpu_utilization',
                'MeasureValueType': 'DOUBLE'
            },
        ]
    }

    const timestreamConfiguration = {
        DatabaseName: SQ_DATABASE_NAME,
        TableName: SQ_TABLE_NAME,
        TimeColumn: "binned_timestamp",
        DimensionMappings: DimensionMappingList,
        MultiMeasureMappings: MultiMeasureMappings
    }

    const createScheduledQueryRequest = {
        Name: SQ_NAME,
        QueryString: VALID_QUERY,
        ScheduleConfiguration: {
            ScheduleExpression: SCHEDULE_EXPRESSION
        },
        NotificationConfiguration: {
            SnsConfiguration: {
                TopicArn: topicArn
            }
```

```
        },
        TargetConfiguration: {
            TimestreamConfiguration: timestreamConfiguration
        },
        ScheduledQueryExecutionRoleArn: roleArn,
        ErrorReportConfiguration: {
            S3Configuration: {
                BucketName: s3ErrorReportBucketName
            }
        }
    };
    try {
        const data = await
 queryClient.createScheduledQuery(createScheduledQueryRequest).promise();
        console.log("Successfully created scheduled query: " + data.Arn);
        return data.Arn;
    } catch (err) {
        console.log("Scheduled Query creation failed: ", err);
        throw err;
    }
}
```

.NET

```
public const string Hostname = "host-24Gju";
public const string SqName = "timestream-sample";
public const string SqDatabaseName = "sq_result_database";
public const string SqTableName = "sq_result_table";

public const string ErrorConfigurationS3BucketNamePrefix = "error-configuration-
sample-s3-bucket-";
public const string ScheduleExpression = "cron(0/2 * * * ? *)";

// Find the average, p90, p95, and p99 CPU utilization for a specific EC2 host over
 the past 2 hours.
public const string ValidQuery = "SELECT region, az, hostname, BIN(time, 15s) AS
 binned_timestamp, " +
      "ROUND(AVG(cpu_utilization), 2) AS avg_cpu_utilization, " +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.9), 2) AS p90_cpu_utilization, " +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.95), 2) AS p95_cpu_utilization, "
 +
      "ROUND(APPROX_PERCENTILE(cpu_utilization, 0.99), 2) AS p99_cpu_utilization " +
      "FROM " + Constants.DATABASE_NAME + "." + Constants.TABLE_NAME + " " +
```

```
      "WHERE measure_name = 'metrics' " +
      "AND hostname = '" + Hostname + "' " +
      "AND time > ago(2h) " +
      "GROUP BY region, hostname, az, BIN(time, 15s) " +
      "ORDER BY binned_timestamp ASC " +
      "LIMIT 5";

private async Task<String> CreateValidScheduledQuery(string topicArn, string
 roleArn,
             string databaseName, string tableName, string s3ErrorReportBucketName)
 {
     List<MultiMeasureAttributeMapping> sourceColToMeasureValueTypes =
         new List<MultiMeasureAttributeMapping>()
         {
             new()
             {
                 SourceColumn = "avg_cpu_utilization",
                 MeasureValueType = MeasureValueType.DOUBLE.Value
             },
             new()
             {
                 SourceColumn = "p90_cpu_utilization",
                 MeasureValueType = MeasureValueType.DOUBLE.Value
             },
             new()
             {
                 SourceColumn = "p95_cpu_utilization",
                 MeasureValueType = MeasureValueType.DOUBLE.Value
             },
             new()
             {
                 SourceColumn = "p99_cpu_utilization",
                 MeasureValueType = MeasureValueType.DOUBLE.Value
             }
         };

     TargetConfiguration targetConfiguration = new TargetConfiguration()
     {
         TimestreamConfiguration = new TimestreamConfiguration()
         {
             DatabaseName = databaseName,
             TableName = tableName,
             TimeColumn = "binned_timestamp",
             DimensionMappings = new List<DimensionMapping>()
```

```
            {
                new()
                {
                    Name = "region",
                    DimensionValueType = "VARCHAR"
                },
                new()
                {
                    Name = "az",
                    DimensionValueType = "VARCHAR"
                },
                new()
                {
                    Name = "hostname",
                    DimensionValueType = "VARCHAR"
                }
            },
            MultiMeasureMappings = new MultiMeasureMappings()
            {
                TargetMultiMeasureName = "multi-metrics",
                MultiMeasureAttributeMappings = sourceColToMeasureValueTypes
            }
        }
    };
    return await CreateScheduledQuery(topicArn, roleArn, s3ErrorReportBucketName,
        ScheduledQueryConstants.ValidQuery, targetConfiguration);
}

private async Task<String> CreateScheduledQuery(string topicArn, string roleArn,
            string s3ErrorReportBucketName, string query, TargetConfiguration
 targetConfiguration)
 {
    try
    {
        Console.WriteLine("Creating Scheduled Query");
        CreateScheduledQueryResponse response = await
 _amazonTimestreamQuery.CreateScheduledQueryAsync(
            new CreateScheduledQueryRequest()
            {
                Name = ScheduledQueryConstants.SqName,
                QueryString = query,
                ScheduleConfiguration = new ScheduleConfiguration()
                {
                    ScheduleExpression = ScheduledQueryConstants.ScheduleExpression
```

```
                },
                NotificationConfiguration = new NotificationConfiguration()
                {
                    SnsConfiguration = new SnsConfiguration()
                    {
                        TopicArn = topicArn
                    }
                },
                TargetConfiguration = targetConfiguration,
                ErrorReportConfiguration = new ErrorReportConfiguration()
                {
                    S3Configuration = new S3Configuration()
                    {
                        BucketName = s3ErrorReportBucketName
                    }
                },
                ScheduledQueryExecutionRoleArn = roleArn
            });
        Console.WriteLine($"Successfully created scheduled query :
{response.Arn}");
        return response.Arn;
    }
    catch (Exception e)
    {
        Console.WriteLine($"Scheduled Query creation failed: {e}");
        throw;
    }
}
```

예약된 쿼리 나열

다음 코드 조각을 사용하여 예약된 쿼리를 나열할 수 있습니다.

Java

```
public void listScheduledQueries() {
    System.out.println("Listing Scheduled Query");
    try {
        String nextToken = null;
        List<String> scheduledQueries = new ArrayList<>();

        do {
```

```
            ListScheduledQueriesResult listScheduledQueriesResult =
                    queryClient.listScheduledQueries(new
 ListScheduledQueriesRequest()
                            .withNextToken(nextToken).withMaxResults(10));
            List<ScheduledQuery> scheduledQueryList =
 listScheduledQueriesResult.getScheduledQueries();

            printScheduledQuery(scheduledQueryList);
            nextToken = listScheduledQueriesResult.getNextToken();
        } while (nextToken != null);
    }
    catch (Exception e) {
        System.out.println("List Scheduled Query failed: " + e);
        throw e;
    }
}

public void printScheduledQuery(List<ScheduledQuery> scheduledQueryList) {
    for (ScheduledQuery scheduledQuery: scheduledQueryList) {
        System.out.println(scheduledQuery.getArn());
    }
}
```

Java v2

```
public void listScheduledQueries() {
    System.out.println("Listing Scheduled Query");
    try {
        String nextToken = null;

        do {
            ListScheduledQueriesResponse listScheduledQueriesResult =

 queryClient.listScheduledQueries(ListScheduledQueriesRequest.builder()
                            .nextToken(nextToken).maxResults(10)
                            .build());
            List<ScheduledQuery> scheduledQueryList =
 listScheduledQueriesResult.scheduledQueries();

            printScheduledQuery(scheduledQueryList);
            nextToken = listScheduledQueriesResult.nextToken();
        } while (nextToken != null);
    }
```

```
    catch (Exception e) {
        System.out.println("List Scheduled Query failed: " + e);
        throw e;
    }
}

public void printScheduledQuery(List<ScheduledQuery> scheduledQueryList) {
    for (ScheduledQuery scheduledQuery: scheduledQueryList) {
        System.out.println(scheduledQuery.arn());
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) ListScheduledQueries()
 ([]*timestreamquery.ScheduledQuery, error) {

     var nextToken *string = nil
     var scheduledQueries []*timestreamquery.ScheduledQuery
     for ok := true; ok; ok = nextToken != nil {
         listScheduledQueriesInput := &timestreamquery.ListScheduledQueriesInput{
             MaxResults: aws.Int64(15),
         }
         if nextToken != nil {
             listScheduledQueriesInput.NextToken = aws.String(*nextToken)
         }

         listScheduledQueriesOutput, err :=
 timestreamBuilder.QuerySvc.ListScheduledQueries(listScheduledQueriesInput)
         if err != nil {
             fmt.Printf("Error: %s", err.Error())
             return nil, err
         }
         scheduledQueries = append(scheduledQueries,
 listScheduledQueriesOutput.ScheduledQueries...)
         nextToken = listScheduledQueriesOutput.NextToken
     }
     return scheduledQueries, nil
 }
```

Python

```
def list_scheduled_queries(self):
```

```
    print("\nListing Scheduled Queries")
    try:
        response = self.query_client.list_scheduled_queries(MaxResults=10)
        self.print_scheduled_queries(response['ScheduledQueries'])
        next_token = response.get('NextToken', None)
        while next_token:
            response =
 self.query_client.list_scheduled_queries(NextToken=next_token, MaxResults=10)
            self.print_scheduled_queries(response['ScheduledQueries'])
            next_token = response.get('NextToken', None)
    except Exception as err:
        print("List scheduled queries failed:", err)
        raise err

@staticmethod
def print_scheduled_queries(scheduled_queries):
    for scheduled_query in scheduled_queries:
        print(scheduled_query['Arn'])
```

Node.js

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
async function listScheduledQueries() {
     console.log("Listing Scheduled Query");
     try {
         var nextToken = null;
         do {
             var params = {
                 MaxResults: 10,
                 NextToken: nextToken
             }
             var data = await queryClient.listScheduledQueries(params).promise();
             var scheduledQueryList = data.ScheduledQueries;
             printScheduledQuery(scheduledQueryList);
             nextToken = data.NextToken;
         }
         while (nextToken != null);
     }  catch (err) {
         console.log("List Scheduled Query failed: ", err);
         throw err;
     }
```

```
}

async function printScheduledQuery(scheduledQueryList) {
    scheduledQueryList.forEach(element => console.log(element.Arn));
}
```

.NET

```
private async Task ListScheduledQueries()
{
    try
    {
        Console.WriteLine("Listing Scheduled Query");
        string nextToken;
        do
        {
            ListScheduledQueriesResponse response =
                await _amazonTimestreamQuery.ListScheduledQueriesAsync(new
 ListScheduledQueriesRequest());
            foreach (var scheduledQuery in response.ScheduledQueries)
            {
                Console.WriteLine($"{scheduledQuery.Arn}");
            }

            nextToken = response.NextToken;
        } while (nextToken != null);
    }
    catch (Exception e)
    {
        Console.WriteLine($"List Scheduled Query failed: {e}");
        throw;
    }
}
```

예약된 쿼리 설명

다음 코드 조각을 사용하여 예약된 쿼리를 설명할 수 있습니다.

Java

```
public void describeScheduledQueries(String scheduledQueryArn) {
    System.out.println("Describing Scheduled Query");
```

```
    try {
        DescribeScheduledQueryResult describeScheduledQueryResult =
 queryClient.describeScheduledQuery(new
 DescribeScheduledQueryRequest().withScheduledQueryArn(scheduledQueryArn));
        System.out.println(describeScheduledQueryResult);
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Describe Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void describeScheduledQueries(String scheduledQueryArn) {
    System.out.println("Describing Scheduled Query");
    try {
        DescribeScheduledQueryResponse describeScheduledQueryResult =

 queryClient.describeScheduledQuery(DescribeScheduledQueryRequest.builder()
                        .scheduledQueryArn(scheduledQueryArn)
                        .build());
        System.out.println(describeScheduledQueryResult);
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Describe Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) DescribeScheduledQuery(scheduledQueryArn
 string) error {
```

```
    describeScheduledQueryInput := &timestreamquery.DescribeScheduledQueryInput{
        ScheduledQueryArn: aws.String(scheduledQueryArn),
    }
    describeScheduledQueryOutput, err :=
timestreamBuilder.QuerySvc.DescribeScheduledQuery(describeScheduledQueryInput)

    if err != nil {
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
aerr.Error())
            default:
                fmt.Printf("Error: %s", err.Error())
            }
        } else {
            fmt.Printf("Error: %s", aerr.Error())
        }
        return err
    } else {
        fmt.Println("DescribeScheduledQuery is successful, below is the output:")
        fmt.Println(describeScheduledQueryOutput.ScheduledQuery)
        return nil
    }
}
```

Python

```
def describe_scheduled_query(self, scheduled_query_arn):
    print("\nDescribing Scheduled Query")
    try:
        response =
self.query_client.describe_scheduled_query(ScheduledQueryArn=scheduled_query_arn)
        if 'ScheduledQuery' in response:
            response = response['ScheduledQuery']
            for key in response:
                print("{} :{}".format(key, response[key]))
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query describe failed:", err)
        raise err
```

Node.js

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
async function describeScheduledQuery(scheduledQueryArn) {
     console.log("Describing Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn
     }
     try {
         const data = await queryClient.describeScheduledQuery(params).promise();
         console.log(data.ScheduledQuery);
     } catch (err) {
         console.log("Describe Scheduled Query failed: ", err);
         throw err;
     }
 }
```

.NET

```
private async Task DescribeScheduledQuery(string scheduledQueryArn)
 {
     try
     {
         Console.WriteLine("Describing Scheduled Query");
         DescribeScheduledQueryResponse response = await
 _amazonTimestreamQuery.DescribeScheduledQueryAsync(
             new DescribeScheduledQueryRequest()
             {
                 ScheduledQueryArn = scheduledQueryArn
             });

 Console.WriteLine($"{JsonConvert.SerializeObject(response.ScheduledQuery)}");
     }
     catch (ResourceNotFoundException e)
     {
         Console.WriteLine($"Scheduled Query doesn't exist: {e}");
         throw;
     }
     catch (Exception e)
     {
         Console.WriteLine($"Describe Scheduled Query failed: {e}");
```

```
            throw;
        }
    }
```

예약된 쿼리 실행

다음 코드 조각을 사용하여 예약된 쿼리를 실행할 수 있습니다.

Java

```
public void executeScheduledQueries(String scheduledQueryArn, Date invocationTime) {
    System.out.println("Executing Scheduled Query");
    try {
        ExecuteScheduledQueryResult executeScheduledQueryResult =
 queryClient.executeScheduledQuery(new ExecuteScheduledQueryRequest()
                .withScheduledQueryArn(scheduledQueryArn)
                .withInvocationTime(invocationTime)
        );

    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void executeScheduledQuery(String scheduledQueryArn) {
    System.out.println("Executing Scheduled Query");
    try {
        ExecuteScheduledQueryResponse executeScheduledQueryResult =
 queryClient.executeScheduledQuery(ExecuteScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn)
                .invocationTime(Instant.now())
                .build()
        );
```

```
        System.out.println("Execute ScheduledQuery response code: " +
 executeScheduledQueryResult.sdkHttpResponse().statusCode());

    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) ExecuteScheduledQuery(scheduledQueryArn
 string, invocationTime time.Time) error {

     executeScheduledQueryInput := &timestreamquery.ExecuteScheduledQueryInput{
         ScheduledQueryArn: aws.String(scheduledQueryArn),
         InvocationTime:    aws.Time(invocationTime),
     }
     executeScheduledQueryOutput, err :=
 timestreamBuilder.QuerySvc.ExecuteScheduledQuery(executeScheduledQueryInput)

     if err != nil {
         if aerr, ok := err.(awserr.Error); ok {
             switch aerr.Code() {
             case timestreamquery.ErrCodeResourceNotFoundException:
                 fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
             default:
                 fmt.Printf("Error: %s", aerr.Error())
             }
         } else {
             fmt.Printf("Error: %s", err.Error())
         }
         return err
     } else {
         fmt.Println("ExecuteScheduledQuery is successful, below is the output:")
         fmt.Println(executeScheduledQueryOutput.GoString())
```

```
            return nil
        }
    }
```

Python

```
def execute_scheduled_query(self, scheduled_query_arn, invocation_time):
    print("\nExecuting Scheduled Query")
    try:

 self.query_client.execute_scheduled_query(ScheduledQueryArn=scheduled_query_arn,
 InvocationTime=invocation_time)
        print("Successfully started executing scheduled query")
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query execution failed:", err)
        raise err
```

Node.js

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
async function executeScheduledQuery(scheduledQueryArn, invocationTime) {
     console.log("Executing Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn,
         InvocationTime: invocationTime
     }
     try {
         await queryClient.executeScheduledQuery(params).promise();
     } catch (err) {
         console.log("Execute Scheduled Query failed: ", err);
         throw err;
     }
 }
```

.NET

```
private async Task ExecuteScheduledQuery(string scheduledQueryArn, DateTime
 invocationTime)
```

```
{
    try
    {
        Console.WriteLine("Running Scheduled Query");
        await _amazonTimestreamQuery.ExecuteScheduledQueryAsync(new
ExecuteScheduledQueryRequest()
        {
            ScheduledQueryArn = scheduledQueryArn,
            InvocationTime = invocationTime
        });
        Console.WriteLine("Successfully started manual run of scheduled query");
    }
    catch (ResourceNotFoundException e)
    {
        Console.WriteLine($"Scheduled Query doesn't exist: {e}");
        throw;
    }
    catch (Exception e)
    {
        Console.WriteLine($"Execute Scheduled Query failed: {e}");
        throw;
    }
}
```

예약된 쿼리 업데이트

다음 코드 조각을 사용하여 예약된 쿼리를 업데이트할 수 있습니다.

Java

```
public void updateScheduledQueries(String scheduledQueryArn) {
    System.out.println("Updating Scheduled Query");
    try {
        queryClient.updateScheduledQuery(new UpdateScheduledQueryRequest()
                .withScheduledQueryArn(scheduledQueryArn)
                .withState(ScheduledQueryState.DISABLED));
        System.out.println("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
```

```
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Java v2

```
public void updateScheduledQuery(String scheduledQueryArn, ScheduledQueryState
 state) {
    System.out.println("Updating Scheduled Query");
    try {
        queryClient.updateScheduledQuery(UpdateScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn)
                .state(state)
                .build());
        System.out.println("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e) {
        System.out.println("Scheduled Query doesn't exist");
        throw e;
    }
    catch (Exception e) {
        System.out.println("Execution Scheduled Query failed: " + e);
        throw e;
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) UpdateScheduledQuery(scheduledQueryArn
 string) error {

     updateScheduledQueryInput := &timestreamquery.UpdateScheduledQueryInput{
         ScheduledQueryArn: aws.String(scheduledQueryArn),
         State:             aws.String(timestreamquery.ScheduledQueryStateDisabled),
     }
     _, err :=
 timestreamBuilder.QuerySvc.UpdateScheduledQuery(updateScheduledQueryInput)

     if err != nil {
         if aerr, ok := err.(awserr.Error); ok {
             switch aerr.Code() {
```

```
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("UpdateScheduledQuery is successful")
        return nil
    }
}
```

Python

```
def update_scheduled_query(self, scheduled_query_arn, state):
    print("\nUpdating Scheduled Query")
    try:

 self.query_client.update_scheduled_query(ScheduledQueryArn=scheduled_query_arn,
                                                  State=state)
        print("Successfully update scheduled query state to", state)
    except self.query_client.exceptions.ResourceNotFoundException as err:
        print("Scheduled Query doesn't exist")
        raise err
    except Exception as err:
        print("Scheduled Query deletion failed:", err)
        raise err
```

Node.js

다음 코드 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
async function updateScheduledQueries(scheduledQueryArn) {
     console.log("Updating Scheduled Query");
     var params = {
         ScheduledQueryArn: scheduledQueryArn,
         State: "DISABLED"
     }
```

```
    try {
        await queryClient.updateScheduledQuery(params).promise();
        console.log("Successfully update scheduled query state");
    } catch (err) {
        console.log("Update Scheduled Query failed: ", err);
        throw err;
    }
}
```

.NET

```
private async Task UpdateScheduledQuery(string scheduledQueryArn,
 ScheduledQueryState state)
{
    try
    {
        Console.WriteLine("Updating Scheduled Query");
        await _amazonTimestreamQuery.UpdateScheduledQueryAsync(new
 UpdateScheduledQueryRequest()
        {
            ScheduledQueryArn = scheduledQueryArn,
            State = state
        });
        Console.WriteLine("Successfully update scheduled query state");
    }
    catch (ResourceNotFoundException e)
    {
        Console.WriteLine($"Scheduled Query doesn't exist: {e}");
        throw;
    }
    catch (Exception e)
    {
        Console.WriteLine($"Update Scheduled Query failed: {e}");
        throw;
    }
}
```

예약된 쿼리 삭제

다음 코드 조각을 사용하여 예약된 쿼리를 삭제할 수 있습니다.

Java

```
public void deleteScheduledQuery(String scheduledQueryArn) {
    System.out.println("Deleting Scheduled Query");

    try {
        queryClient.deleteScheduledQuery(new
 DeleteScheduledQueryRequest().withScheduledQueryArn(scheduledQueryArn));
        System.out.println("Successfully deleted scheduled query");
    }
    catch (Exception e) {
        System.out.println("Scheduled Query deletion failed: " + e);
    }
}
```

Java v2

```
public void deleteScheduledQuery(String scheduledQueryArn) {
    System.out.println("Deleting Scheduled Query");

    try {
        queryClient.deleteScheduledQuery(DeleteScheduledQueryRequest.builder()
                .scheduledQueryArn(scheduledQueryArn).build());
        System.out.println("Successfully deleted scheduled query");
    }
    catch (Exception e) {
        System.out.println("Scheduled Query deletion failed: " + e);
    }
}
```

Go

```
func (timestreamBuilder TimestreamBuilder) DeleteScheduledQuery(scheduledQueryArn
 string) error {

     deleteScheduledQueryInput := &timestreamquery.DeleteScheduledQueryInput{
         ScheduledQueryArn: aws.String(scheduledQueryArn),
     }
     _, err :=
 timestreamBuilder.QuerySvc.DeleteScheduledQuery(deleteScheduledQueryInput)

     if err != nil {
         fmt.Println("Error:")
```

```
        if aerr, ok := err.(awserr.Error); ok {
            switch aerr.Code() {
            case timestreamquery.ErrCodeResourceNotFoundException:
                fmt.Println(timestreamquery.ErrCodeResourceNotFoundException,
 aerr.Error())
            default:
                fmt.Printf("Error: %s", aerr.Error())
            }
        } else {
            fmt.Printf("Error: %s", err.Error())
        }
        return err
    } else {
        fmt.Println("DeleteScheduledQuery is successful")
        return nil
    }
}
```

Python

```
def delete_scheduled_query(self, scheduled_query_arn):
    print("\nDeleting Scheduled Query")
    try:

 self.query_client.delete_scheduled_query(ScheduledQueryArn=scheduled_query_arn)
        print("Successfully deleted scheduled query :", scheduled_query_arn)
    except Exception as err:
        print("Scheduled Query deletion failed:", err)
        raise err
```

Node.js

다음 조각은 JavaScript V2 스타일에 를 AWS SDK 사용합니다. 이는 의 애플리케이션에 대한 Node.js 샘플 Amazon Timestream의 샘플 LiveAnalytics 애플리케이션을 GitHub 기반으로 합니다.

```
async function deleteScheduleQuery(scheduledQueryArn) {
     console.log("Deleting Scheduled Query");
     const params = {
         ScheduledQueryArn: scheduledQueryArn
     }
     try {
         await queryClient.deleteScheduledQuery(params).promise();
         console.log("Successfully deleted scheduled query");
```

```
    } catch (err) {
        console.log("Scheduled Query deletion failed: ", err);
    }
}
```

.NET

```
private async Task DeleteScheduledQuery(string scheduledQueryArn)
{
    try
    {
        Console.WriteLine("Deleting Scheduled Query");
        await _amazonTimestreamQuery.DeleteScheduledQueryAsync(new
DeleteScheduledQueryRequest()
        {
            ScheduledQueryArn = scheduledQueryArn
        });
        Console.WriteLine($"Successfully deleted scheduled query :
{scheduledQueryArn}");
    }
    catch (Exception e)
    {
        Console.WriteLine($"Scheduled Query deletion failed: {e}");
        throw;
    }
}
```

에 대해 Timestream에서 배치 로드 사용 LiveAnalytics

용 Amazon Timestream에 대한 배치 로드로 Amazon S3에 저장된 CSV 파일을 Timestream에 배치로 수집할 LiveAnalytics수 있습니다. 이 새로운 기능을 사용하면 다른 도구에 의존하거나 사용자 지정 코드를 작성할 필요 LiveAnalytics 없이 에 대한 데이터를 Timestream에 저장할 수 있습니다. 쿼리 또는 분석에 즉시 필요하지 않은 데이터와 같이 유연한 대기 시간으로 데이터를 채우는 데 배치 로드를 사용할 수 있습니다.

AWS Management Console, 및 를 사용하여 배치 로드 작업을 생성할 수 AWS CLI있습니다 AWS SDKs. 자세한 내용은 콘솔에서 배치 로드 사용, 에서 배치 로드 사용 AWS CLI, 에서 배치 로드 사용 AWS SDKs 단원을 참조하세요.

배치 로드 외에도 WriteRecords API 작업과 함께 여러 레코드를 동시에 작성할 수 있습니다. 사용할 항목에 대한 지침은 섹션을 참조하세요작업과 배치 로드 중에서 WriteRecords API 선택.

주제

Timestream의 배치 로드 개념

배치 로드 기능을 더 잘 이해하려면 다음 개념을 검토하세요.

배치 로드 작업 - Amazon Timestream에서 소스 데이터 및 대상을 정의하는 작업입니다. 배치 로드 작업을 생성할 때 데이터 모델과 같은 추가 구성을 지정합니다. AWS Management Console, 및 를 통해 배치 로드 작업을 생성할 수 AWS CLI있습니다 AWS SDKs.

대상 가져오기 - Timestream의 대상 데이터베이스 및 테이블입니다. 데이터베이스 및 테이블 생성에 대한 자세한 내용은 데이터베이스 생성 및 섹션을 참조하세요테이블 생성.

데이터 소스 - S3 버킷에 저장된 소스 CSV 파일입니다. 데이터 파일 준비에 대한 자세한 내용은 섹션을 참조하세요배치 로드 데이터 파일 준비. S3 요금에 대한 자세한 내용은 Amazon S3 요금 섹션을 참조하세요.

배치 로드 오류 보고서 - 배치 로드 작업의 오류에 대한 정보를 저장하는 보고서입니다. 배치 로드 오류 보고서의 S3 위치를 배치 로드 작업의 일부로 정의합니다. 보고서의 정보에 대한 자세한 내용은 섹션을 참조하세요배치 로드 오류 보고서 사용.

데이터 모델 매핑 - S3 위치의 데이터 소스에서 LiveAnalytics 테이블의 대상 Timestream으로의 시간, 차원 및 측정값에 대한 배치 로드 매핑입니다. 자세한 내용은 배치 로드에 대한 데이터 모델 매핑 단원을 참조하십시오.

배치 로드 사전 조건

배치 로드를 사용하기 위한 사전 조건 목록입니다. 모범 사례는 배치 로드 모범 사례 단원을 참조하세요.

- 배치 로드 소스 데이터는 헤더와 CSV 함께 Amazon S3 형식으로 저장됩니다.
- 각 Amazon S3 소스 버킷에 대해 연결된 정책에 다음 권한이 있어야 합니다.

```
"s3:GetObject",
"s3:GetBucketAcl"
"s3:ListBucket"
```

마찬가지로 보고서가 작성된 각 Amazon S3 출력 버킷에 대해 연결된 정책에 다음 권한이 있어야 합니다.

```
"s3:PutObject",
"s3:GetBucketAcl"
```

예:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject",
        "s3:GetBucketAcl"
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::inputs-source-bucket-name-A”
        "arn:aws:s3:::inputs-source-bucket-name-B”
      ],
      "Effect": "Allow"
    },
    {
      "Action": [
        "s3:PutObject",
        "s3:GetBucketAcl"
      ],
      "Resource": [
```

```
        "arn:aws:s3:::reports-output-bucket-name”
      ]
      "Effect": "Allow"
    }
  ]
}
```

- 의 타임스트림은 데이터 모델에 제공된 정보를 CSV 헤더에 매핑CSV하여 를 LiveAnalytics 구문 분석합니다. 데이터에는 타임스탬프를 나타내는 열, 하나 이상의 차원 열 및 하나 이상의 측정 열이 있어야 합니다.
- 배치 로드와 함께 사용되는 S3 버킷은 배치 로드에 사용되는 LiveAnalytics 테이블의 Timestream과 동일한 리전 및 동일한 계정의 버킷이어야 합니다.
- `timestamp` 열은 Unix 에포크 이후의 시간을 나타내는 긴 데이터 형식이어야 합니다. 예를 들어 타임스탬프는 로 `2021-03-25T08:45:21Z` 표시됩니다`1616661921`. Timestream은 타임스탬프 정밀도에 초, 밀리초, 마이크로초 및 나노초를 지원합니다. 쿼리 언어를 사용하는 경우 와 같은 함수를 사용하여 형식 간에 변환할 수 있습니다`to_unixtime`. 자세한 내용은 날짜/시간 함수 단원을 참조하십시오.
- Timestream은 차원 값에 대한 문자열 데이터 유형을 지원합니다. 측정 열에 대한 긴 데이터, 이중 데이터, 문자열 데이터 및 부울 데이터 유형을 지원합니다.

배치 로드 한도 및 할당량은 섹션을 참조하세요배치 로드.

배치 로드 모범 사례

배치 로드는 다음 조건 및 권장 사항을 준수할 때 가장 잘 작동합니다(높은 처리량).

1. CSV 수집을 위해 제출된 파일은 특히 파일 크기가 100MB~1GB로 작아 병렬 처리와 수집 속도를 개선합니다.
2. 배치 로드가 진행 중일 때는 동일한 테이블에 데이터를 동시에 수집하지 마세요(예: WriteRecords API 작업 또는 예약된 쿼리 사용). 이로 인해 스로틀이 발생하여 배치 로드 작업이 실패할 수 있습니다.
3. 배치 로드 작업이 실행되는 동안 배치 로드에 사용되는 S3 버킷에서 파일을 추가, 수정 또는 제거하지 마세요.
4. 테이블 또는 소스에서 권한을 삭제하거나 취소하거나 예약되었거나 진행 중인 배치 로드 작업이 있는 S3 버킷을 보고하지 마세요.
5. 카디널리티가 높은 차원 값 집합으로 데이터를 수집할 때는 의 지침을 따르세요다중 측정 레코드 파티셔닝을 위한 권장 사항.

6. 작은 파일을 제출하여 데이터의 정확성을 테스트해야 합니다. 정확성과 관계없이 배치 로드에 제출된 모든 데이터에 대해 요금이 부과됩니다. 요금에 대한 자세한 내용은 Amazon Timestream 요금 섹션을 참조하세요.
7. 가 250 미만`ActiveMagneticStorePartitions`이 아닌 한 배치 로드 작업을 재개하지 마십시오. 작업이 제한되어 실패할 수 있습니다. 동일한 데이터베이스에 대해 여러 작업을 동시에 제출하면 수를 줄일 수 있습니다.

다음은 콘솔 모범 사례입니다.

1. 다중 측정 레코드에는 하나의 측정 이름만 사용하는 더 간단한 데이터 모델링에만 빌더를 사용합니다.
2. 보다 복잡한 데이터 모델링을 위해 를 사용합니다JSON. 예를 들어 다중 측정 레코드를 사용할 JSON 때 여러 측정 이름을 사용할 때 를 사용합니다.

LiveAnalytics 모범 사례에 대한 추가 Timestream은 섹션을 참조하세요모범 사례.

배치 로드 데이터 파일 준비

소스 데이터 파일에는 구분 기호로 구분된 값이 있습니다. 보다 구체적인 용어인 쉼표로 구분된 값(CSV)은 일반적으로 사용됩니다. 유효한 열 구분 기호에는 쉼표와 파이프가 포함됩니다. 레코드는 새 줄로 구분됩니다. 파일은 Amazon S3에 저장해야 합니다. 새 배치 로드 작업을 생성하면 소스 데이터의 위치가 파일에 ARN 대해 에 의해 지정됩니다. 파일에는 헤더가 포함됩니다. 한 열은 타임스탬프를 나타냅니다. 하나 이상의 다른 열은 측정값을 나타냅니다.

배치 로드에 사용되는 S3 버킷은 배치 로드에 사용되는 LiveAnalytics 테이블의 Timestream과 동일한 리전에 있어야 합니다. 배치 로드 작업이 제출된 후에는 배치 로드에 사용된 S3 버킷에서 파일을 추가하거나 제거하지 마세요. S3 버킷 작업에 대한 자세한 내용은 Amazon S3 시작하기를 참조하세요.

> **Note**
>
> CSV Excel과 같은 일부 애플리케이션에서 생성되는 파일에는 예상 인코딩과 충돌하는 바이트 순서 표시(BOM)가 포함될 수 있습니다. 프로그래밍 방식으로 처리될 때 오류가 BOM 발생한 CSV 파일을 참조하는 LiveAnalytics 배치 로드 작업의 시간 흐름입니다. 이를 방지하려면 보이지 않는 문자BOM인 를 제거할 수 있습니다.
> 예를 들어 새 인코딩을 지정할 수 있는 Notepad++와 같은 애플리케이션에서 파일을 저장할 수 있습니다. 첫 번째 줄을 읽고, 줄에서 문자를 제거하고, 파일의 첫 번째 줄 위에 새 값을 쓰는 프로그래밍 옵션을 사용할 수도 있습니다.

Excel에서 저장할 때는 여러 CSV 옵션이 있습니다. 다른 CSV 옵션을 사용하여 저장하면 설명된 문제가 방지될 수 있습니다. 그러나 인코딩 변경으로 인해 일부 문자에 영향을 미칠 수 있으므로 결과를 확인해야 합니다.

CSV 형식 파라미터

형식 파라미터로 예약되는 값을 나타낼 때 이스케이프 문자를 사용합니다. 예를 들어, 따옴표 문자가 큰따옴표인 경우 데이터에서 큰따옴표를 나타내려면 큰따옴표 앞에 이스케이프 문자를 배치합니다.

배치 로드 작업을 생성할 때 이러한 작업을 지정하는 시기에 대한 자세한 내용은 섹션을 참조하세요배치 로드 작업 생성.

파라미터	옵션
열 구분자	(쉼표(',') | 파이프('|') | 세미콜론(';') | 탭('/t') | 공백(' '))
이스케이프 문자	없음
따옴표 문자	콘솔: (더블 따옴표(') | 단일 따옴표('))
Null 값	공백(‘ ')
공백 다듬기	콘솔: (아니요 | 예)

배치 로드에 대한 데이터 모델 매핑

다음은 데이터 모델 매핑 스키마와 의 및 예제를 설명합니다.

데이터 모델 매핑 스키마

`CreateBatchLoadTask` 요청 구문과 배치 로드`DataModel`용 를 포함하는 `BatchLoadTaskDescription` 객체를 `DescribeBatchLoadTask` 포함하도록 호출로 반환되는 `DataModelConfiguration` 객체입니다. 는 S3 위치에 CSV 형식으로 저장된 소스 데이터에서 LiveAnalytics 데이터베이스 및 테이블의 대상 Timestream으로의 매핑을 `DataModel` 정의합니다.

`TimeColumn` 필드는 에 대한 Timestream의 대상 테이블 `time` 열에 매핑될 값에 대한 소스 데이터의 위치를 나타냅니다 LiveAnalytics. 는 의 단위를 `TimeUnit` 지정하며 `TimeColumn`, `MILLISECONDS`,

SECONDS MICROSECONDS또는 중 하나가 될 수 있습니다NANOSECONDS. 차원 및 측정값에 대한 매핑도 있습니다. 차원 매핑은 소스 열과 대상 필드로 구성됩니다.

자세한 내용은 DimensionMapping를 참조하세요. 측정값에 대한 매핑에는 MixedMeasureMappings 및 두 가지 옵션이 있습니다MultiMeasureMappings.

요약하자면 에는 S3 위치의 데이터 소스에서 다음에 대한 LiveAnalytics 테이블에 대한 대상 Timestream으로의 매핑이 DataModel 포함되어 있습니다.

- Time
- 차원
- 치수

가능하면 에 대한 Timestream의 다중 측정 레코드에 측정 데이터를 매핑하는 것이 좋습니다 LiveAnalytics. 다중 측정 레코드의 이점에 대한 자세한 내용은 섹션을 참조하세요다중 측정 레코드.

소스 데이터의 여러 측정값이 한 행에 저장되는 경우 해당 여러 측정값을 를 LiveAnalytics 사용하기 위해 Timestream의 다중 측정 레코드에 매핑할 수 있습니다MultiMeasureMappings. 단일 측정 레코드에 매핑해야 하는 값이 있는 경우 를 사용할 수 있습니다MixedMeasureMappings.

MixedMeasureMappings 및 둘 MultiMeasureMappings 다 를 포함합니다MultiMeasureAttributeMappings. 단일 측정 레코드가 필요한지 여부에 관계없이 다중 측정 레코드가 지원됩니다.

의 Timestream에는 다중 측정 대상 레코드만 필요한 경우 다음 구조에서 측정 매핑을 정의할 LiveAnalytics수 있습니다.

```
CreateBatchLoadTask
    MeasureNameColumn
    MultiMeasureMappings
        TargetMultiMeasureName
        MultiMeasureAttributeMappings array
```

> **Note**
>
> 가능하면 MultiMeasureMappings 를 사용하는 것이 좋습니다.

의 Timestream에 단일 측정 대상 레코드가 필요한 경우 다음 구조에서 측정 매핑을 정의할 LiveAnalytics수 있습니다.

```
CreateBatchLoadTask
    MeasureNameColumn
    MixedMeasureMappings array
        MixedMeasureMapping
            MeasureName
            MeasureValueType
            SourceColumn
            TargetMeasureName
            MultiMeasureAttributeMappings array
```

MultiMeasureMappings를 사용하는 경우 항상 MultiMeasureAttributeMappings 배열이 필요합니다. MixedMeasureMappings 배열을 사용할 때 MeasureValueType가 지정된 에 MULTI 대한 경우 MixedMeasureMappingMultiMeasureAttributeMappings는 해당 에 필요합니다MixedMeasureMapping. 그렇지 않으면 는 단일 측정 레코드의 측정 유형을 MeasureValueType 나타냅니다.

어느 쪽이든 사용 MultiMeasureAttributeMapping 가능한 배열이 있습니다. 다음과 MultiMeasureAttributeMapping 같이 각 에서 다중 측정 레코드에 대한 매핑을 정의합니다.

SourceColumn

Amazon S3에 있는 소스 데이터의 열입니다.

TargetMultiMeasureAttributeName

대상 테이블에서 대상 다중 측정 이름의 이름입니다. 이 입력은 MeasureNameColumn 이 제공되지 않은 경우 필요합니다. MeasureNameColumn 이 제공되면 해당 열의 값이 다중 측정 이름으로 사용됩니다.

MeasureValueType

DOUBLE, BIGINT BOOLEANVARCHAR, 또는 중 하나TIMESTAMP.

MultiMeasureMappings 예제를 사용한 데이터 모델 매핑

이 예제에서는 각 측정값을 전용 열에 저장하는 선호하는 접근 방식인 다중 측정 레코드에 대한 매핑을 보여줍니다. 샘플 CSV에서 샘플을 다운로드할 수 있습니다. CSV 샘플에는 LiveAnalytics 테이블에 대한 Timestream의 대상 열에 매핑할 수 있는 다음 제목이 있습니다.

- time
- measure_name
- region
- location
- hostname
- memory_utilization
- cpu_utilization

CSV 파일에서 time 및 measure_name 열을 식별합니다. 이 경우 이러한 맵은 동일한 이름의 LiveAnalytics 테이블 열에 대해 Timestream에 직접 매핑됩니다.

- time 에 매핑 time
- measure_name measure_name (또는 선택한 값)에 매핑

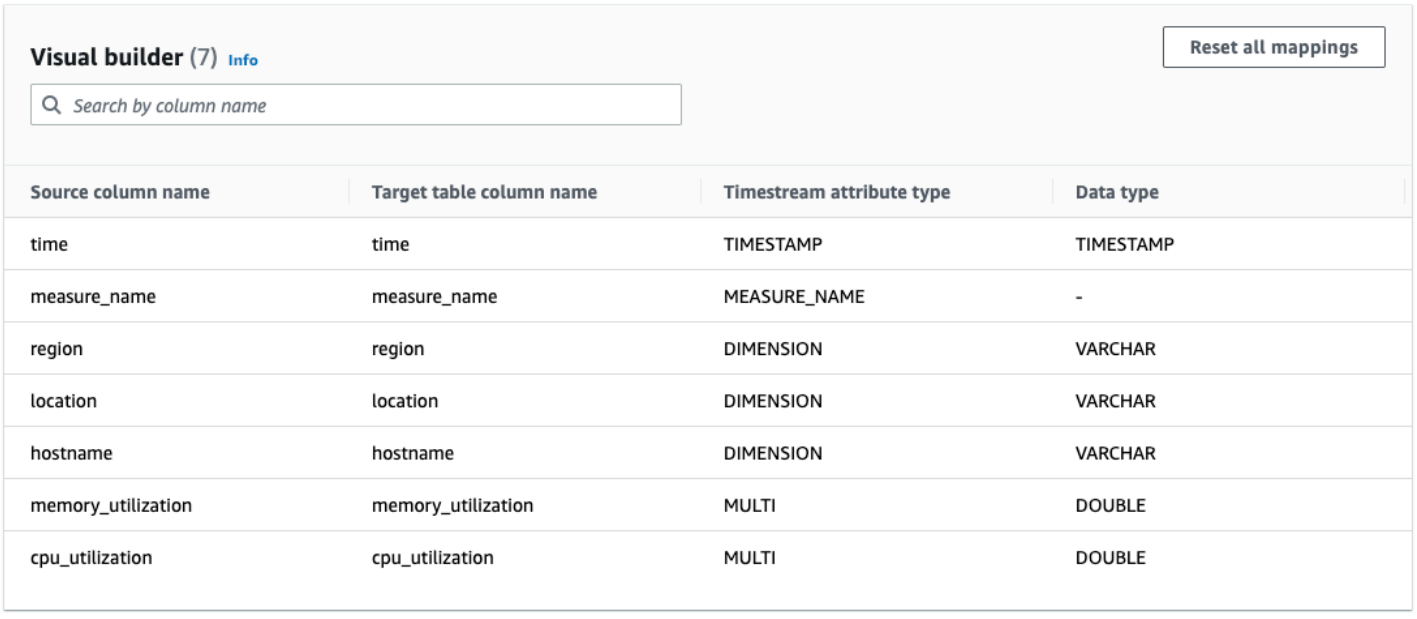
를 사용할 때 TimeColumn 필드에 time를 지정하고 필드와 같은 지원되는 시간 단위 값을 API지정합니다MILLISECONDSTimeUnit. 이는 콘솔의 소스 열 이름 및 타임스탬프 시간 입력에 해당합니다. MeasureNameColumn 키로 정의된 measure_name를 사용하여 레코드를 그룹화하거나 파티션할 수 있습니다.

샘플에서 , location및 regionhostname는 차원입니다. 차원은 DimensionMapping 객체 배열로 매핑됩니다.

측정값의 경우 값은 LiveAnalytics 테이블의 Timestream에서 열이 TargetMultiMeasureAttributeName 됩니다. 이 예제와 같은 이름을 유지할 수 있습니다. 또는 새를 지정할 수 있습니다. MeasureValueType는 DOUBLE, BIGINT, BOOLEAN, VARCHAR또는 중 하나입니다TIMESTAMP.

```
{
  "TimeColumn": "time",
  "TimeUnit": "MILLISECONDS",
  "DimensionMappings": [
    {
      "SourceColumn": "region",
      "DestinationColumn": "region"
    },
    {
      "SourceColumn": "location",
```

```
      "DestinationColumn": "location"
    },
    {
      "SourceColumn": "hostname",
      "DestinationColumn": "hostname"
    }
  ],
  "MeasureNameColumn": "measure_name",
  "MultiMeasureMappings": {
    "MultiMeasureAttributeMappings": [
      {
        "SourceColumn": "memory_utilization",
        "TargetMultiMeasureAttributeName": "memory_utilization",
        "MeasureValueType": "DOUBLE"
      },
      {
        "SourceColumn": "cpu_utilization",
        "TargetMultiMeasureAttributeName": "cpu_utilization",
        "MeasureValueType": "DOUBLE"
      }
    ]
  }
}
```



`MixedMeasureMappings` 예제를 사용한 데이터 모델 매핑

Timestream for 에서 단일 측정 레코드에 매핑해야 하는 경우에만 이 접근 방식을 사용하는 것이 좋습니다 LiveAnalytics.

콘솔에서 배치 로드 사용

다음은 에서 배치 로드를 사용하는 단계입니다 AWS Management Console. 샘플 CSV에서 샘플을 다운로드할 수 있습니다. CSV

주제

- 배치 로드 액세스
- 배치 로드 작업 생성
- 배치 로드 작업 재개
- 시각적 빌더 사용

배치 로드 액세스

다음 단계에 따라 를 사용하여 배치 로드에 액세스합니다 AWS Management Console.

1. Amazon Timestream 콘솔 을 엽니다.
2. 탐색 창에서 관리 도구 를 선택한 다음 작업 배치 로드 를 선택합니다.
3. 여기에서 배치 로드 작업 목록을 보고 자세한 내용을 보려면 지정된 작업을 드릴링할 수 있습니다. 작업을 생성하고 재개할 수도 있습니다.

배치 로드 작업 생성

다음 단계에 따라 를 사용하여 배치 로드 작업을 생성합니다 AWS Management Console.

1. Amazon Timestream 콘솔 을 엽니다.
2. 탐색 창에서 관리 도구 를 선택한 다음 작업 배치 로드 를 선택합니다.
3. 배치 로드 작업 생성을 선택합니다.
4. 가져오기 대상 에서 다음을 선택합니다.
   - 대상 데이터베이스 - 에서 생성된 데이터베이스의 이름을 선택합니다 데이터베이스 생성.

- 대상 테이블 - 에서 생성된 테이블의 이름을 선택합니다테이블 생성.

  필요한 경우 새 테이블 생성 버튼을 사용하여 이 패널에서 테이블을 추가할 수 있습니다.

5. 데이터 소스 의 데이터 소스 S3 위치에서 소스 데이터가 저장되는 S3 버킷을 선택합니다. S3 찾아보기 버튼을 사용하여 활성 AWS 계정이 액세스할 수 있는 S3 리소스를 보거나 S3 위치 를 입력합니다URL. 데이터 소스는 동일한 리전에 있어야 합니다.
6. 파일 형식 설정(확장 가능 섹션)에서 기본 설정을 사용하여 입력 데이터를 구문 분석할 수 있습니다. 고급 설정 을 선택할 수도 있습니다. 여기에서 CSV 형식 파라미터 를 선택하고 입력 데이터를 구문 분석할 파라미터를 선택할 수 있습니다. 이러한 파라미터에 대한 자세한 내용은 섹션을 참조하세요CSV 형식 파라미터.
7. 데이터 모델 매핑 구성 에서 데이터 모델을 구성합니다. 추가 데이터 모델 지침은 섹션을 참조하세요. 배치 로드에 대한 데이터 모델 매핑

   - 데이터 모델 매핑에서 구성 입력 매핑을 선택하고 다음 중 하나를 선택합니다.
     - 시각적 빌더 - 데이터를 시각적으로 매핑하려면 TargetMultiMeasureName 또는 를 선택합니다MeasureNameColumn. 그런 다음 시각적 빌더 에서 열을 매핑합니다.

       시각적 빌더는 단일 파일을 데이터 소스로 선택하면 데이터 소스 CSV 파일에서 소스 열 헤더를 자동으로 감지하고 로드합니다. 매핑을 생성할 속성과 데이터 유형을 선택합니다.

       시각적 빌더 사용에 대한 자세한 내용은 섹션을 참조하세요시각적 빌더 사용.
     - JSON 편집기 - 데이터 모델을 구성하기 위한 자유 형식 JSON 편집기입니다. 에 대한 Timestream에 익숙 LiveAnalytics 하고 고급 데이터 모델 매핑을 빌드하려는 경우 이 옵션을 선택합니다.
     - JSON S3의 파일 - S3에 저장한 JSON 모델 파일을 선택합니다. 데이터 모델을 이미 구성했고 추가 배치 로드에 재사용하려는 경우 이 옵션을 선택합니다.
8. 오류 로그 보고서의 오류 로그 S3 위치 에서 오류를 보고하는 데 사용할 S3 위치를 선택합니다. 이 보고서를 사용하는 방법에 대한 자세한 내용은 섹션을 참조하세요배치 로드 오류 보고서 사용.
9. 암호화 키 유형 에서 다음 중 하나를 선택합니다.

   - Amazon S3-managed 키(SSE-S3) - Amazon S3가 생성, 관리 및 사용하는 암호화 키입니다.
   - AWS KMS key (SSE-KMS) - AWS Key Management Service ()로 보호되는 암호화 키AWS KMS입니다.
10. Next(다음)를 선택합니다.
11. 검토 및 생성 페이지에서 설정을 검토하고 필요에 따라 편집합니다.

> **Note**
>
> 작업이 생성된 후에는 배치 로드 작업 설정을 변경할 수 없습니다. 작업 완료 시간은 가져오는 데이터의 양에 따라 달라집니다.

12. 배치 로드 작업 생성을 선택합니다.

배치 로드 작업 재개

상태가 '진행 중지됨'인 배치 로드 태스크를 선택했는데도 여전히 재개 가능한 경우 태스크를 재개하라는 메시지가 표시됩니다. 해당 작업에 대한 세부 정보를 볼 때 작업 재개 버튼이 있는 배너도 있습니다. 재사용 가능한 작업의 “재개 기한” 날짜는 입니다. 해당 날짜가 만료되면 작업을 재개할 수 없습니다.

시각적 빌더 사용

시각적 빌더를 사용하여 S3 버킷에 저장된 하나 이상의 CSV 파일(들)의 소스 데이터 열을 LiveAnalytics 테이블용 Timestream의 대상 열에 매핑할 수 있습니다.

> **Note**
>
> 역할에는 파일에 대한 `SelectObjectContent` 권한이 필요합니다. 이렇게 하지 않으면 열을 수동으로 추가하고 삭제해야 합니다.

소스 열 모드 자동 로드

의 Timestream LiveAnalytics 은 버킷을 하나만 지정하는 경우 소스 CSV 파일에서 열 이름을 자동으로 스캔할 수 있습니다. 기존 매핑이 없는 경우 소스 열 가져오기를 선택할 수 있습니다.

1. 매핑 구성 입력 설정 에서 시각적 빌더 옵션을 선택한 경우 타임스탬프 시간 입력을 설정합니다. `Milliseconds`는 기본 설정입니다.
2. 소스 열 로드 버튼을 클릭하여 소스 데이터 파일에 있는 열 헤더를 가져옵니다. 테이블은 데이터 소스 파일의 소스 열 헤더 이름으로 채워집니다.
3. 각 소스 열에 대해 대상 테이블 열 이름, Timestream 속성 유형 및 데이터 유형을 선택합니다.

   이러한 열과 가능한 값에 대한 자세한 내용은 섹션을 참조하세요필드 매핑.
4. 기능을 사용하여 drag-to-fill 여러 열의 값을 한 번에 설정합니다.

수동으로 소스 열 추가

단일 가 아닌 버킷 또는 CSV 접두사를 사용하는 경우 열 매핑 추가 및 열 매핑 삭제 버튼을 사용하여 시각적 편집기에서 열 매핑을 추가하고 삭제할 CSV수 있습니다. 매핑을 재설정하는 버튼도 있습니다.

필드 매핑

- 소스 열 이름 - 가져올 측정값을 나타내는 소스 파일의 열 이름입니다. 소스 열 가져오기 를 사용하면 에 대한 Timestream이 이 값을 자동으로 채울 LiveAnalytics 수 있습니다.
- 대상 테이블 열 이름 - 대상 테이블에서 측정값의 열 이름을 나타내는 선택적 입력입니다.
- Timestream 속성 유형 - 와 같이 지정된 소스 열에 있는 데이터의 속성 유형입니다`DIMENSION`.
  - TIMESTAMP - 측정값이 수집되는 시점을 지정합니다.
  - MULTI - 여러 측정값이 표시됩니다.
  - DIMENSION – 시계열 메타데이터.
  - MEASURE_NAME – 단일 측정 레코드의 경우 측정 이름입니다.
- 데이터 유형 - 와 같은 Timestream 열의 유형입니다`BOOLEAN`.
  - BIGINT – 64비트 정수입니다.
  - BOOLEAN – 로직의 두 가지 진실 값인 true와 false입니다.
  - DOUBLE – 64비트 변수-정밀도 번호.
  - TIMESTAMP - 에서 나노초 정밀도 시간을 사용하고 Unix 에포크 이후의 시간을 UTC추적하는 인스턴스입니다.

에서 배치 로드 사용 AWS CLI

설정

배치 로드 사용을 시작하려면 다음 단계를 수행합니다.

1. 의 지침에 AWS CLI 따라 를 설치합니다를 LiveAnalytics 사용하기 위한 Amazon Timestream 액세스 AWS CLI.
2. 다음 명령을 실행하여 Timestream CLI 명령이 업데이트되었는지 확인합니다. 가 목록에 있는지 create-batch-load-task 확인합니다.

   ```
   aws timestream-write help
   ```

3. 의 지침을 사용하여 데이터 소스를 준비합니다배치 로드 데이터 파일 준비.

4. 의 지침을 사용하여 데이터베이스와 테이블을 생성합니다를 LiveAnalytics 사용하기 위한 Amazon Timestream 액세스 AWS CLI.
5. 보고서 출력을 위한 S3 버킷을 생성합니다. 버킷은 동일한 리전에 있어야 합니다. 버킷에 대한 자세한 내용은 Amazon S3 버킷 생성, 구성 및 작업을 참조하세요.
6. 배치 로드 작업을 생성합니다. 단계는 배치 로드 작업 생성를 참조하세요.
7. 작업 상태를 확인합니다. 단계는 배치 로드 작업 설명를 참조하세요.

배치 로드 작업 생성

`create-batch-load-task` 명령을 사용하여 배치 로드 작업을 생성할 수 있습니다. 를 사용하여 배치 로드 작업을 생성할 때 JSON 파라미터를 단일 JSON 조각으로 집계할 CLI수 있는 파라미터 `cli-input-json`를 사용할 수 있습니다. `data-model-configuration`, , , , 를 비롯한 여러 다른 파라미터를 사용하여 이러한 세부 정보를 구분할 수도 있습니다`data-source-configurationreport-configurationtarget-database-nametarget-table-name`.

예제는 배치 로드 작업 생성 예제 단원을 참조하십시오.

배치 로드 작업 설명

다음과 같이 배치 로드 작업 설명을 검색할 수 있습니다.

```
aws timestream-write describe-batch-load-task --task-id <value>
```

다음은 응답의 예입니다.

```
{
    "BatchLoadTaskDescription": {
        "TaskId": "<TaskId>",
        "DataSourceConfiguration": {
            "DataSourceS3Configuration": {
                "BucketName": "test-batch-load-west-2",
                "ObjectKeyPrefix": "sample.csv"
            },
            "CsvConfiguration": {},
            "DataFormat": "CSV"
        },
        "ProgressReport": {
            "RecordsProcessed": 2,
            "RecordsIngested": 0,
            "FileParseFailures": 0,
```

```
            "RecordIngestionFailures": 2,
            "FileFailures": 0,
            "BytesIngested": 119
        },
        "ReportConfiguration": {
            "ReportS3Configuration": {
                "BucketName": "test-batch-load-west-2",
                "ObjectKeyPrefix": "<ObjectKeyPrefix>",
                "EncryptionOption": "SSE_S3"
            }
        },
        "DataModelConfiguration": {
            "DataModel": {
                "TimeColumn": "timestamp",
                "TimeUnit": "SECONDS",
                "DimensionMappings": [
                    {
                        "SourceColumn": "vehicle",
                        "DestinationColumn": "vehicle"
                    },
                    {
                        "SourceColumn": "registration",
                        "DestinationColumn": "license"
                    }
                ],
                "MultiMeasureMappings": {
                    "TargetMultiMeasureName": "test",
                    "MultiMeasureAttributeMappings": [
                        {
                            "SourceColumn": "wgt",
                            "TargetMultiMeasureAttributeName": "weight",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "spd",
                            "TargetMultiMeasureAttributeName": "speed",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "SourceColumn": "fuel",
                            "TargetMultiMeasureAttributeName": "fuel",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
```

```
                            "SourceColumn": "miles",
                            "TargetMultiMeasureAttributeName": "miles",
                            "MeasureValueType": "DOUBLE"
                        }
                    ]
                }
            }
        },
        "TargetDatabaseName": "BatchLoadExampleDatabase",
        "TargetTableName": "BatchLoadExampleTable",
        "TaskStatus": "FAILED",
        "RecordVersion": 1,
        "CreationTime": 1677167593.266,
        "LastUpdatedTime": 1677167602.38
    }
}
```

배치 로드 작업 나열

다음과 같이 배치 로드 작업을 나열할 수 있습니다.

```
aws timestream-write list-batch-load-tasks
```

출력은 다음과 같이 나타납니다.

```
{
    "BatchLoadTasks": [
        {
            "TaskId": "<TaskId>",
            "TaskStatus": "FAILED",
            "DatabaseName": "BatchLoadExampleDatabase",
            "TableName": "BatchLoadExampleTable",
            "CreationTime": 1677167593.266,
            "LastUpdatedTime": 1677167602.38
        }
    ]
}
```

배치 로드 작업 재개

다음과 같이 배치 로드 작업을 재개할 수 있습니다.

```
aws timestream-write resume-batch-load-task --task-id <value>
```

응답은 성공을 나타내거나 오류 정보를 포함할 수 있습니다.

배치 로드 작업 생성 예제

Example

1. 이름이 인 LiveAnalytics 데이터베이스BatchLoad와 이름이 인 테이블의 Timestream을 생성합니다BatchLoadTest. MemoryStoreRetentionPeriodInHours 및 의 값을 확인하고 필요한 경우 조정합니다MagneticStoreRetentionPeriodInDays.

   ```
   aws timestream-write create-database --database-name BatchLoad \

   aws timestream-write create-table --database-name BatchLoad \
   --table-name BatchLoadTest \
   --retention-properties "{\"MemoryStoreRetentionPeriodInHours\": 12,
    \"MagneticStoreRetentionPeriodInDays\": 100}"
   ```

2. 콘솔을 사용하여 S3 버킷을 생성하고 sample.csv 파일을 해당 위치에 복사합니다. 샘플 CSV에서 샘플을 다운로드할 수 있습니다. CSV

3. 콘솔을 사용하여 의 Timestream용 S3 버킷을 생성 LiveAnalytics 하여 배치 로드 작업이 오류와 함께 완료된 경우 보고서를 작성합니다.

4. 배치 로드 작업을 생성합니다. 교체해야 합니다.*$INPUT_BUCKET* 그리고 *$REPORT_BUCKET* 이전 단계에서 생성한 버킷을 사용합니다.

   ```
   aws timestream-write create-batch-load-task \
   --data-model-configuration "{\
               \"DataModel\": {\
                 \"TimeColumn\": \"timestamp\",\
                 \"TimeUnit\": \"SECONDS\",\
                 \"DimensionMappings\": [\
                   {\
                     \"SourceColumn\": \"vehicle\"\
                   },\
                   {\
                     \"SourceColumn\": \"registration\",\
                     \"DestinationColumn\": \"license\"\
                   }\
                 ],
                 \"MultiMeasureMappings\": {\
   ```

```
                \"TargetMultiMeasureName\": \"mva_measure_name\",\
                \"MultiMeasureAttributeMappings\": [\
                  {\
                    \"SourceColumn\": \"wgt\",\
                    \"TargetMultiMeasureAttributeName\": \"weight\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"spd\",\
                    \"TargetMultiMeasureAttributeName\": \"speed\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"fuel_consumption\",\
                    \"TargetMultiMeasureAttributeName\": \"fuel\",\
                    \"MeasureValueType\": \"DOUBLE\"\
                  },\
                  {\
                    \"SourceColumn\": \"miles\",\
                    \"MeasureValueType\": \"BIGINT\"\
                  }\
                ]\
              }\
            }\
          }" \
--data-source-configuration "{
            \"DataSourceS3Configuration\": {\
              \"BucketName\": \"$INPUT_BUCKET\",\
              \"ObjectKeyPrefix\": \"$INPUT_OBJECT_KEY_PREFIX\"
            },\
            \"DataFormat\": \"CSV\"\
          }" \
--report-configuration "{\
            \"ReportS3Configuration\": {\
              \"BucketName\": \"$REPORT_BUCKET\",\
              \"EncryptionOption\": \"SSE_S3\"\
            }\
          }" \
--target-database-name BatchLoad \
--target-table-name BatchLoadTest
```

이전 명령은 다음 출력을 반환합니다.

```
{
    "TaskId": "TaskId "
}
```

5. 작업 진행 상황을 확인합니다. 바꾸어야 합니다.*$TASK_ID* 이전 단계에서 반환된 태스크 ID를 사용합니다.

```
aws timestream-write describe-batch-load-task --task-id $TASK_ID
```

출력 예시

```
{
    "BatchLoadTaskDescription": {
        "ProgressReport": {
            "BytesIngested": 1024,
            "RecordsIngested": 2,
            "FileFailures": 0,
            "RecordIngestionFailures": 0,
            "RecordsProcessed": 2,
            "FileParseFailures": 0
        },
        "DataModelConfiguration": {
            "DataModel": {
                "DimensionMappings": [
                    {
                        "SourceColumn": "vehicle",
                        "DestinationColumn": "vehicle"
                    },
                    {
                        "SourceColumn": "registration",
                        "DestinationColumn": "license"
                    }
                ],
                "TimeUnit": "SECONDS",
                "TimeColumn": "timestamp",
                "MultiMeasureMappings": {
                    "MultiMeasureAttributeMappings": [
                        {
                            "TargetMultiMeasureAttributeName": "weight",
                            "SourceColumn": "wgt",
                            "MeasureValueType": "DOUBLE"
```

```
                    },
                    {
                        "TargetMultiMeasureAttributeName": "speed",
                        "SourceColumn": "spd",
                        "MeasureValueType": "DOUBLE"
                    },
                    {
                        "TargetMultiMeasureAttributeName": "fuel",
                        "SourceColumn": "fuel_consumption",
                        "MeasureValueType": "DOUBLE"
                    },
                    {
                        "TargetMultiMeasureAttributeName": "miles",
                        "SourceColumn": "miles",
                        "MeasureValueType": "DOUBLE"
                    }
                ],
                "TargetMultiMeasureName": "mva_measure_name"
            }
        }
    },
    "TargetDatabaseName": "BatchLoad",
    "CreationTime": 1672960381.735,
    "TaskStatus": "SUCCEEDED",
    "RecordVersion": 1,
    "TaskId": "TaskId ",
    "TargetTableName": "BatchLoadTest",
    "ReportConfiguration": {
        "ReportS3Configuration": {
            "EncryptionOption": "SSE_S3",
            "ObjectKeyPrefix": "ObjectKeyPrefix ",
            "BucketName": "test-report-bucket"
        }
    },
    "DataSourceConfiguration": {
        "DataSourceS3Configuration": {
            "ObjectKeyPrefix": "sample.csv",
            "BucketName": "test-input-bucket"
        },
        "DataFormat": "CSV",
        "CsvConfiguration": {}
    },
    "LastUpdatedTime": 1672960387.334
}
```

```
}
```

에서 배치 로드 사용 AWS SDKs

를 사용하여 배치 로드 태스크를 AWS 생성, 설명 및 나열하는 방법에 대한 예는 , 배치 로드 작업 생성, 배치 로드 작업 설명및 배치 로드 작업 나열섹션을 SDKs참조하세요배치 로드 작업 재개.

배치 로드 오류 보고서 사용

배치 로드 작업에는 다음 상태 값 중 하나가 있습니다.

- CREATED (생성됨) - 작업이 생성됩니다.
- IN_PROGRESS (진행 중) - 작업이 진행 중입니다.
- FAILED (실패) - 작업이 완료되었습니다. 하지만 하나 이상의 오류가 감지되었습니다.
- SUCCEEDED (완료됨) - 태스크가 오류 없이 완료되었습니다.
- PROGRESS_STOPPED (진행이 중지됨) - 작업이 중지되었지만 완료되지 않았습니다. 작업을 재개할 수 있습니다.
- PENDING_RESUME (재개 보류 중) - 태스크가 재개 보류 중입니다.

오류가 있는 경우 해당 에 대해 정의된 S3 버킷에 오류 로그 보고서가 생성됩니다. 오류는 별도의 배열 fileErrors 에서 taskErrors 또는 로 분류됩니다. 다음은 오류 보고서의 예입니다.

```
{
    "taskId": "9367BE28418C5EF902676482220B631C",
    "taskErrors": [],
    "fileErrors": [
        {
            "fileName": "example.csv",
            "errors": [
                {
                    "reason": "The record timestamp is outside the time range of the
 data ingestion window.",
                    "lineRanges": [
                        [
                            2,
                            3
                        ]
                    ]
```

```
                }
            ]
        }
    ]
}
```

에 대해 Timestream에서 예약된 쿼리 사용 LiveAnalytics

용 Amazon Timestream의 예약된 쿼리 기능은 운영 대시보드, 비즈니스 보고서, 임시 분석 및 기타 애플리케이션에 일반적으로 사용되는 집계, 롤업 및 기타 형태의 사전 처리된 데이터를 계산하고 저장하기 위한 LiveAnalytics 완전 관리형, 서버리스 및 확장 가능한 솔루션입니다. 예약된 쿼리를 사용하면 실시간 분석이 더 효과적이고 비용 효율적이므로 데이터에서 추가 인사이트를 도출하고 더 나은 비즈니스 결정을 내릴 수 있습니다.

예약된 쿼리를 사용하면 데이터에 대한 집계, 롤업 및 기타 작업을 계산하는 실시간 분석 쿼리를 정의하고 에 대한 Amazon Timestream은 이러한 쿼리를 LiveAnalytics 주기적으로 자동으로 실행하고 쿼리 결과를 별도의 테이블에 안정적으로 기록합니다. 데이터는 일반적으로 몇 분 내에 계산되어 이러한 테이블로 업데이트됩니다.

그런 다음 대시보드와 보고서를 지시하여 상당히 큰 소스 테이블을 쿼리하는 대신 집계된 데이터가 포함된 테이블을 쿼리할 수 있습니다. 이로 인해 성능 및 비용 이득이 100%를 초과할 수 있습니다. 이는 집계된 데이터가 있는 테이블이 소스 테이블보다 훨씬 적은 데이터를 포함하기 때문에 쿼리가 더 빨라지고 데이터 스토리지가 더 저렴하기 때문입니다.

또한 예약된 쿼리가 있는 테이블은 LiveAnalytics 테이블용 Timestream의 모든 기존 기능을 제공합니다. 예를 들어 를 사용하여 테이블을 쿼리할 수 있습니다SQL. Grafana를 사용하여 테이블에 저장된 데이터를 시각화할 수 있습니다. Amazon Kinesis , Amazon ,MSK AWS IoT Core 및 Telegraf를 사용하여 테이블에 데이터를 수집할 수도 있습니다. 자동 데이터 수명 주기 관리를 위해 이러한 테이블에서 데이터 보존 정책을 구성할 수 있습니다.

집계된 데이터가 포함된 테이블의 데이터 보존은 소스 테이블의 데이터 보존과 완전히 분리되므로 데이터 스토리지 비용의 일부만으로 소스 테이블의 데이터 보존을 줄이고 집계 데이터를 훨씬 더 오래 보존하도록 선택할 수도 있습니다. 예약된 쿼리는 실시간 분석을 더 빠르고 저렴하게 만들므로 더 많은 고객이 더 쉽게 액세스할 수 있으므로 애플리케이션을 모니터링하고 더 나은 데이터 기반 비즈니스 결정을 내릴 수 있습니다.

주제

- 예약된 쿼리 혜택

- 예약된 쿼리 사용 사례
- 예: 실시간 분석을 사용하여 사기 결제를 감지하고 더 나은 비즈니스 의사 결정
- 예약된 쿼리 개념
- 예약된 쿼리에 대한 표현식 예약
- 예약된 쿼리에 대한 데이터 모델 매핑
- 예약된 쿼리 알림 메시지
- 예약된 쿼리 오류 보고서
- 예약된 쿼리 패턴 및 예제

예약된 쿼리 혜택

다음은 예약된 쿼리의 이점입니다.

- 운영 용이성 - 예약된 쿼리는 서버가 없으며 완전히 관리됩니다.
- 성능 및 비용 - 예약된 쿼리는 데이터에 대한 집계, 롤업 또는 기타 실시간 분석 작업을 미리 계산하고 결과를 테이블에 저장하기 때문에 예약된 쿼리로 채워진 액세스 테이블에 소스 테이블보다 적은 데이터가 포함된 쿼리입니다. 따라서 이러한 테이블에서 실행되는 쿼리는 더 빠르고 저렴합니다. 예약된 계산으로 채워진 테이블에는 소스 테이블보다 적은 데이터가 포함되어 있으므로 스토리지 비용을 줄이는 데 도움이 됩니다. 또한 메모리 스토어에 소스 데이터를 보존하는 비용의 일부만으로 메모리 스토어에 이 데이터를 더 오래 보존할 수 있습니다.
- 상호 운용성 - 예약된 쿼리로 채워진 테이블은 LiveAnalytics 테이블용 Timestream의 모든 기존 기능을 제공하며 용 Timestream으로 작동하는 모든 서비스 및 도구와 함께 사용할 수 있습니다 LiveAnalytics. 자세한 내용은 다른 서비스 작업을 참조하세요.

예약된 쿼리 사용 사례

애플리케이션의 최종 사용자 활동을 요약하는 비즈니스 보고서에 예약된 쿼리를 사용할 수 있으므로 개인화를 위해 기계 학습 모델을 훈련할 수 있습니다. 또한 이상, 네트워크 침입 또는 사기 활동을 감지하는 경보에 대해 예약된 쿼리를 사용할 수 있으므로 즉각적인 수정 조치를 취할 수 있습니다.

또한 보다 효과적인 데이터 거버넌스를 위해 예약된 쿼리를 사용할 수 있습니다. 이렇게 하려면 예약된 쿼리에만 소스 테이블 액세스 권한을 부여하고 개발자에게 예약된 쿼리로 채워진 테이블에 대한 액세스 권한만 부여하면 됩니다. 이렇게 하면 의도하지 않고 오래 실행되는 쿼리의 영향을 최소화할 수 있습니다.

예: 실시간 분석을 사용하여 사기 결제를 감지하고 더 나은 비즈니스 의사 결정

미국의 주요 대도시 전역에 분산된 여러 point-of-sale 터미널에서 전송된 트랜잭션을 처리하는 결제 시스템을 고려해 보세요. 용 Amazon Timestream LiveAnalytics 을 사용하여 트랜잭션 데이터를 저장하고 분석하면 사기 트랜잭션을 감지하고 실시간 분석 쿼리를 실행할 수 있습니다. 이러한 쿼리는 시간당 가장 사용량이 많고 가장 적게 사용되는 point-of-sale 터미널, 각 도시에서 하루 중 가장 사용량이 많은 시간, 시간당 대부분의 트랜잭션이 있는 도시를 식별하는 등 비즈니스 질문에 답변하는 데 도움이 될 수 있습니다.

시스템은 분당 약 10만 개의 트랜잭션을 처리합니다. 에 대해 LiveAnalytics Amazon Timestream에 저장된 각 트랜잭션은 100바이트입니다. 1분마다 실행되는 쿼리 10개를 구성하여 다양한 종류의 사기 결제를 감지했습니다. 또한 다양한 차원에 따라 데이터를 집계하고 조각화/다이싱하는 25개의 쿼리를 생성하여 비즈니스 질문에 답변하는 데 도움이 됩니다. 이러한 각 쿼리는 지난 1시간의 데이터를 처리합니다.

이러한 쿼리에서 생성된 데이터를 표시하는 대시보드를 생성했습니다. 대시보드에는 25개의 위젯이 포함되어 있으며, 이는 매시간 새로 고쳐지고, 일반적으로 지정된 시간에 10명의 사용자가 액세스합니다. 마지막으로 메모리 스토어는 2시간의 데이터 보존 기간으로 구성되고 마그네틱 스토어는 6개월의 데이터 보존 기간을 갖도록 구성됩니다.

이 경우 대시보드에 액세스하고 새로 고칠 때마다 데이터를 다시 계산하는 실시간 분석 쿼리를 사용하거나 대시보드에 파생 테이블을 사용할 수 있습니다. 실시간 분석 쿼리를 기반으로 하는 대시보드의 쿼리 비용은 매월 120.70달러입니다. 반면 파생 테이블로 구동되는 쿼리를 대시보드화하는 비용은 매월 12.27달러입니다(가격은 Amazon Timestream LiveAnalytics 참조). 이 경우 파생 테이블을 사용하면 쿼리 비용이 약 10배 줄어듭니다.

예약된 쿼리 개념

쿼리 문자열 - LiveAnalytics 테이블의 다른 Timestream에 사전 계산하고 저장하는 결과를 가진 쿼리입니다. Timestream for 의 전체 SQL 표면적을 사용하여 예약된 쿼리를 정의할 수 있습니다. LiveAnalytics이 영역은 일반적인 테이블 표현식, 중첩된 쿼리, 창 함수 또는 쿼리 언어 에 대해 Timestream에서 지원하는 모든 종류의 집계 및 스칼라 함수를 사용하여 LiveAnalytics 쿼리를 작성할 수 있는 유연성을 제공합니다.

스케줄 표현식 - 예약된 쿼리 인스턴스가 실행되는 시기를 지정할 수 있습니다. cron 표현식(예: UTC 매일 오전 8시에 실행) 또는 속도 표현식(예: 10분마다 실행)을 사용하여 표현식을 지정할 수 있습니다.

대상 구성 - 예약된 쿼리의 결과를 이 예약된 쿼리의 결과가 저장될 대상 테이블에 매핑하는 방법을 지정할 수 있습니다.

알림 구성 - 의 Timestream은 일정 표현식을 기반으로 예약된 쿼리의 인스턴스를 LiveAnalytics 자동으로 실행합니다. 예약된 쿼리를 생성할 때 구성한 SNS 주제에 대해 실행되는 모든 쿼리에 대한 알림을 받습니다. 이 알림은 인스턴스가 성공적으로 실행되었는지 또는 오류가 발생했는지 여부를 지정합니다. 또한 측정된 바이트, 대상 테이블에 기록된 데이터, 다음 호출 시간 등과 같은 정보를 제공합니다.

다음은 이러한 종류의 알림 메시지의 예입니다.

```
{
    "type":"AUTO_TRIGGER_SUCCESS",
    "arn":"arn:aws:timestream:us-east-1:123456789012:scheduled-query/
 PT1mPerMinutePerRegionMeasureCount-9376096f7309",
    "nextInvocationEpochSecond":1637302500,
    "scheduledQueryRunSummary":
    {
        "invocationEpochSecond":1637302440,
        "triggerTimeMillis":1637302445697,
        "runStatus":"AUTO_TRIGGER_SUCCESS",
        "executionStats":
        {
            "executionTimeInMillis":21669,
            "dataWrites":36864,
            "bytesMetered":13547036820,
            "recordsIngested":1200,
            "queryResultRows":1200
        }
    }
}
```

이 알림 메시지에서 bytesMetered는 쿼리가 소스 테이블에서 스캔한 바이트이고 dataWrites 는 대상 테이블에 기록된 바이트입니다.

> **Note**
>
> 이러한 알림을 프로그래밍 방식으로 사용하는 경우 나중에 알림 메시지에 새 필드를 추가할 수 있습니다.

오류 보고서 위치 - 예약된 쿼리가 비동기적으로 실행되어 대상 테이블에 데이터를 저장합니다. 인스턴스에 오류가 발생하는 경우(예: 저장할 수 없는 잘못된 데이터) 오류가 발생한 레코드는 예약된 쿼리를 생성할 때 지정한 오류 보고서 위치의 오류 보고서에 기록됩니다. 위치에 대한 S3 버킷 및 접두사를 지정합니다. 의 Timestream은 예약된 쿼리의 특정 인스턴스와 관련된 오류를 식별하는 데 도움이 되도록 이 접두사에 예약된 쿼리 이름과 호출 시간을 LiveAnalytics 추가합니다.

태그 지정 - 선택적으로 예약된 쿼리와 연결할 수 있는 태그를 지정할 수 있습니다. 자세한 내용은 리소스 에 대한 LiveAnalytics Timestream 태그 지정을 참조하세요.

예

다음 예제에서는 예약된 쿼리를 사용하여 단순 집계를 계산합니다.

```
SELECT region, bin(time, 1m) as minute,
    SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m
GROUP BY bin(time, 1m), region
```

`@scheduled_runtime parameter` - 이 예제에서는 라는 특별한 파라미터 를 수락하는 쿼리를 볼 수 있습니다`@scheduled_runtime`. 예약된 쿼리의 특정 인스턴스를 호출할 때 서비스가 설정하는 특수 파라미터(타임스탬프 유형)로, 예약된 쿼리의 특정 인스턴스가 소스 테이블의 데이터를 분석하는 시간 범위를 결정적으로 제어할 수 있습니다. 타임스탬프 유형이 예상되는 모든 위치에서 쿼리에 를 사용할 수 `@scheduled_runtime` 있습니다.

예약 쿼리가 매시간 0분, 5분, 10분, 15분, 20분, 25분, 30분, 35분, 40분, 45분, 50분, 55분에 실행되는 cron(0/5 * * * ? *)이라는 예약 표현식을 설정하는 예를 생각해 보세요. 2021-12-01 00:05:00에 트리거되는 인스턴스의 경우 @scheduled_runtime 파라미터가 이 값으로 초기화되므로 현재 인스턴스는 2021-11-30 23:55:00~2021-12-01 00:06:00 범위의 데이터에 대해 작동합니다.

중복 시간 범위가 있는 인스턴스 - 이 예제에서 볼 수 있듯이 예약된 쿼리의 두 후속 인스턴스가 해당 시간 범위에서 중복될 수 있습니다. 이는 요구 사항, 지정한 시간, 일정 표현식에 따라 제어할 수 있습니다. 이 경우 이 중복을 통해 이러한 계산은 도착이 약간 지연된 모든 데이터를 기반으로 집계를 업데이트할 수 있으며, 이 예제에서는 최대 10분까지 가능합니다. 2021-12-01 00:00:00에 트리거된 쿼리 실행은 2021-11-30 23:50:00부터 2021-12-30 00:01:00까지의 시간 범위를 포함하며 2021-12-01 00:05:00에 트리거된 쿼리 실행은 2021-11-30 23:55:00부터 2021-12-01 00:06:00까지의 범위를 포함합니다.

정확성을 보장하고 대상 테이블에 저장된 집계가 소스 테이블에서 계산된 집계와 일치하는지 확인하기 위해 Timestream for 는 2021-12-01 00:05:00의 계산이 완료된 후에만 2021-12-01 00:00:00의

계산이 수행되도록 LiveAnalytics 합니다. 후자의 계산 결과는 새 값이 생성되는 경우 이전에 구체화된 집계를 업데이트할 수 있습니다. 내부적으로 Timestream for 는 예약된 쿼리의 후자 인스턴스에서 생성된 레코드에 더 높은 버전 번호가 할당되는 레코드 버전을 LiveAnalytics 사용합니다. 따라서 2021-12-01 00:05:00에 호출로 계산된 집계는 소스 테이블에서 새 데이터를 사용할 수 있다고 가정하여 2021-12-01 00:00:00에 호출로 계산된 집계를 업데이트할 수 있습니다.

자동 트리거와 수동 트리거 비교 - 예약된 쿼리가 생성된 후 의 Timestream은 지정된 일정에 따라 인스턴스를 LiveAnalytics 자동으로 실행합니다. 이러한 자동 트리거는 전적으로 서비스에 의해 관리됩니다.

그러나 예약된 쿼리의 일부 인스턴스를 수동으로 시작하려는 시나리오가 있을 수 있습니다. 예를 들어, 특정 인스턴스가 쿼리 실행에서 실패한 경우, 자동 일정 실행 후 소스 테이블에 늦게 도착한 데이터 또는 업데이트가 있는 경우 또는 자동 쿼리 실행에서 다루지 않는 시간 범위(예: 예약된 쿼리 생성 전 시간 범위)에 대해 대상 테이블을 업데이트하려는 경우가 있습니다.

를 ExecuteScheduledQuery API 사용하여 @scheduled_runtime InvocationTime 파라미터에 사용되는 값인 파라미터를 전달하여 예약된 쿼리의 특정 인스턴스를 수동으로 시작할 수 있습니다. 다음은 를 사용할 때 고려해야 할 몇 가지 중요한 사항입니다 ExecuteScheduledQuery API.

- 이러한 호출을 여러 번 트리거하는 경우 이러한 호출이 중복 시간 범위를 생성하지 않도록 해야 합니다. 중복되지 않는 시간 범위를 보장할 수 없는 경우 이러한 쿼리 실행을 순차적으로 하나씩 시작해야 합니다. 시간 범위 내에서 중복되는 여러 쿼리 실행을 동시에 시작하는 경우 이러한 쿼리 실행에 대한 오류 보고서에서 버전 충돌이 발생할 수 있는 트리거 실패를 볼 수 있습니다.
- @scheduled_runtime의 타임스탬프 값으로 호출을 시작할 수 있습니다. 따라서 소스 테이블에서 데이터가 업데이트된 범위에 해당하는 대상 테이블에서 적절한 시간 범위가 업데이트되도록 값을 적절하게 설정하는 것은 사용자의 책임입니다.

예약된 쿼리에 대한 표현식 예약

cron 또는 rate 표현식을 사용하는 예약된 쿼리에 Amazon Timestream을 사용하여 자동화된 일정에 LiveAnalytics 예약된 쿼리를 생성할 수 있습니다. 예약된 모든 쿼리는 UTC 시간대를 사용하며 일정에 대해 가능한 최소 정밀도는 1분입니다.

예약 표현식을 지정하는 두 가지 방법은 cron과 rate입니다. Cron 표현식은 더 세분화된 일정 제어를 제공하는 반면, 속도 표현식은 표현하기가 더 간단하지만 세분화된 제어가 부족합니다.

예를 들어 cron 표현식을 사용하면 매주 또는 매월 특정 요일에 지정된 시간에 트리거되는 예약된 쿼리를 정의하거나 월요일 - 금요일에만 매시간 지정된 분 등을 정의할 수 있습니다. 반면 속도 표현식은 예

약된 쿼리가 생성된 정확한 시간부터 시작하여 1분, 시간 또는 1일 1회와 같은 정기적인 속도로 예약된 쿼리를 시작합니다.

cron 표현식

- 구문

```
cron(fields)
```

cron 표현식에는 각각 공백으로 구분되는 필수 필드 6개가 있습니다.

필드	값	와일드카드
Minutes	0~59	, - * /
시간	0~23	, - * /
Day-of-month	1~31	, - * ? / L W
월	1~12 또는 JAN-DEC	, - * /
Day-of-week	1~7 또는 SUN-SAT	, - * ? L #
연도	1970~2199	, - * /

와일드카드 문자

- *,*(쉼표) 와일드카드에는 추가 값이 포함됩니다. 월 필드에는 JANFEB1MAR월, 2월, 3월이 포함됩니다.
- *-*(대시) 와일드카드는 범위를 지정합니다. 예컨대, Day 필드에서 1-15는 지정된 달의 1일에서 15일까지 포함한다는 의미입니다.
- ***(별표) 와일드카드에는 필드의 모든 값이 포함됩니다. 시간 필드에는 ***에 매시간이 포함됩니다. 및 Day-of-week 필드 모두에서 ***를 Day-of-month 사용할 수 없습니다. 한 에서 사용하는 경우 다른 에서 *?*를 사용해야 합니다.
- */*(전진 슬래시) 와일드카드는 증분을 지정합니다. 분 필드에 1/10을 입력하여 매 10분마다 지정할 수 있습니다. 이 시간은 1시간의 첫 분부터 시작합니다(예: 11분, 21분, 31분 등).
- *?*(질문 표시) 와일드카드는 둘 중 하나를 지정합니다. Day-of-month 필드에 *7*을 입력할 수 있으며, 7번째 요일이 몇 요일인지 신경 쓰지 않았다면 필드에 *?*를 Day-of-week 입력할 수 있습니다.

- 또는 Day-of-week 필드의 Day-of-month *L* 와일드카드는 월 또는 주의 마지막 날짜를 지정합니다.
- 필드의 Day-of-month W 와일드카드는 평일을 지정합니다. Day-of-month 필드에서 3W는 해당 월의 셋째 날에 가장 가까운 평일을 지정합니다.
- 필드의 Day-of-week *#* 와일드카드는 한 달 내에 지정된 요일의 특정 인스턴스를 지정합니다. 예를 들어, 3#2는 그 달의 두 번째 화요일입니다. 3은 각 주의 셋째 날이므로 화요일을 나타내고 2는 그 달의 두 번째 해당 요일입니다.

> **Note**
>
> '#' 문자를 사용하는 경우 day-of-week 필드에 하나의 표현식만 정의할 수 있습니다. 예를 들어 '3#1,6#3'은 두 개의 표현식으로 해석되기 때문에 유효하지 않습니다.

제한 사항

- 동일한 cron 표현식에서 및 Day-of-week 필드를 지정할 Day-of-month 수 없습니다. 필드 중 하나에 값(또는 *)을 지정하는 경우 다른 필드에 *?*(질문 표시)를 사용해야 합니다.
- 1분보다 빠른 속도로 이어지는 cron 표현식은 지원되지 않습니다.

예제

분	시간	일	월	요일	연도	의미
0	10	*	*	?	*	매일 오전 10:00(UTC)에 실행합니다.
15	12	*	*	?	*	매일 오후 12:15(UTC)에 실행합니다.

분	시간	일	월	요일	연도	의미
0	18	?	*	MON-FRI	*	매주 월요일부터 금요일까지 오후 6시(UTC)에 실행합니다.
0	8	1	*	?	*	매월 첫째 날 오전 8시(UTC)에 실행합니다.
0/15	*	*	*	?	*	15분마다 실행합니다.
0/10	*	*	*	MON-FRI	*	월요일부터 금요일까지 10분마다 실행합니다.
0/5	8~17	?	*	MON-FRI	*	월요일부터 금요일까지 오전 8시부터 오후 5시 55분()까지 5분마다 실행합니다 UTC.

rate 표현식

- rate 표현식은 예약된 이벤트 규칙을 생성할 때 시작되며, 정의된 예약 일정에 따라 실행됩니다. rate 표현식에는 필수 필드가 2개 있습니다. 각 필드는 공백으로 구분됩니다.

  구문

  ```
  rate(value unit)
  ```

  - `value`: 양수입니다.
  - `unit`: 시간 단위입니다. 값 1(예: 분)과 값 1(예: 분)에는 서로 다른 단위가 필요합니다. 유효값: 분 | 분 | 시간 | 시간 | 일 | 일

예약된 쿼리에 대한 데이터 모델 매핑

의 Timestream은 테이블의 유연한 데이터 모델링을 LiveAnalytics 지원하며, 테이블의 다른 Timestream으로 구체화된 예약된 쿼리의 결과에도 동일한 유연성이 적용됩니다 LiveAnalytics . 예약된 쿼리를 사용하면 다중 측정 레코드 또는 단일 측정 레코드에 데이터가 있는지 여부에 관계없이 모든 테이블을 쿼리하고 다중 측정 또는 단일 측정 레코드를 사용하여 쿼리 결과를 쓸 수 있습니다.

예약된 쿼리 TargetConfiguration 사양의 를 사용하여 쿼리 결과를 대상 파생 테이블의 적절한 열에 매핑합니다. 다음 섹션에서는 파생 테이블에서 다양한 데이터 모델을 달성하기 TargetConfiguration 위해 이를 지정하는 다양한 방법을 설명합니다. 특히 다음을 확인할 수 있습니다.

- 쿼리 결과에 측정값 이름이 없고 에서 대상 측정값 이름을 지정하는 경우 다중 측정 레코드에 쓰는 방법 TargetConfiguration.
- 쿼리 결과에서 측정값 이름을 사용하여 다중 측정 레코드를 작성하는 방법.
- 모델을 정의하여 다양한 다중 측정 속성으로 여러 레코드를 작성하는 방법.
- 파생 테이블의 단일 측정 레코드에 쓸 모델을 정의하는 방법.
- 예약된 쿼리에서 단일 측정 레코드 및/또는 다중 측정 레코드를 쿼리하고 결과를 단일 측정 레코드 또는 다중 측정 레코드로 구체화하여 데이터 모델의 유연성을 선택할 수 있는 방법.

예: 다중 측정 레코드의 대상 측정 이름

이 예제에서는 쿼리가 다중 측정 데이터가 있는 테이블에서 데이터를 읽고 다중 측정 레코드를 사용하여 결과를 다른 테이블에 쓰는 것을 볼 수 있습니다. 예약된 쿼리 결과에는 자연 측정 이름 열이 없습니

다. 여기서 의 TargetMultiMeasureName 속성을 사용하여 파생 테이블에서 측정값 이름을 지정합니다 TargetConfigurationTimestreamConfiguration.

```
{
    "Name" : "CustomMultiMeasureName",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(memory_cached)
 as avg_mem_cached_1h, MIN(memory_free) as min_mem_free_1h, MAX(memory_used) as
 max_mem_used_1h, SUM(disk_io_writes) as sum_1h, AVG(disk_used) as avg_disk_used_1h,
 AVG(disk_free) as avg_disk_free_1h, MAX(cpu_user) as max_cpu_user_1h, MIN(cpu_idle) as
 min_cpu_idle_1h, MAX(cpu_system) as max_cpu_system_1h FROM raw_data.devops_multi WHERE
 time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h
 AND measure_name = 'metrics' GROUP BY region, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_1",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MultiMeasureMappings" : {
                "TargetMultiMeasureName": "dashboard-metrics",
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "avg_mem_cached_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "avgMemCached"
                    },
                    {
                        "SourceColumn" : "min_mem_free_1h",
                        "MeasureValueType" : "DOUBLE"
```

```
                    },
                    {
                        "SourceColumn" : "max_mem_used_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "sum_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "totalDiskWrites"
                    },
                    {
                        "SourceColumn" : "avg_disk_used_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "avg_disk_free_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_user_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "CpuUserP100"
                    },
                    {
                        "SourceColumn" : "min_cpu_idle_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_cpu_system_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName" : "CpuSystemP100"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
```

```
}
```

이 예제의 매핑은 측정 이름 대시보드 지표 및 속성 이름 avgMemCached, min_mem_free_1h, max_mem_used_1h, totalDiskWritesavg_disk_used_1h, avg_disk_free_1h, CpuUserP100, min_cpu_idle_1h, CpuSystemP100을 사용하여 하나의 다중 측정 레코드를 생성합니다. 를 선택적으로 사용하여 쿼리 출력 열의 이름을 결과 구체화에 사용되는 다른 속성 이름으로 TargetMultiMeasureAttributeName 바꿉니다.

다음은 이 예약된 쿼리가 구체화되면 대상 테이블의 스키마입니다. 다음 결과의 LiveAnalytics 속성 유형에 대한 Timestream에서 볼 수 있듯이 측정 스키마와 `dashboard-metrics`같이 결과는 단일 측정 이름 의 다중 측정 레코드로 구체화됩니다.

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
시간	타임스탬프	TIMESTAMP
CpuSystemP100	double	MULTI
avgMemCached	double	MULTI
min_cpu_idle_1h	double	MULTI
평균_디스크_무료_1시간	double	MULTI
avg_disk_used_1h	double	MULTI
totalDiskWrites	double	MULTI
max_mem_used_1h	double	MULTI
min_mem_free_1h	double	MULTI
CpuUserP100	double	MULTI

다음은 SHOW MEASURES 쿼리로 얻은 해당 측정값입니다.

measure_name	data_type	차원
대시보드 지표	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]

예: 다중 측정 레코드에서 예약된 쿼리의 측정 이름 사용

이 예제에서는 단일 측정 레코드가 있는 테이블에서 쿼리 판독값을 보고 결과를 다중 측정 레코드로 구체화합니다. 이 경우 예약된 쿼리 결과에는 예약된 쿼리의 결과가 구체화된 대상 테이블에서 값을 측정값 이름으로 사용할 수 있는 열이 있습니다. 그런 다음 의 MeasureNameColumn 속성을 사용하여 파생 테이블에서 다중 측정 레코드의 측정 이름을 지정할 수 있습니다 TargetConfigurationTimestreamConfiguration.

```
{
    "Name" : "UsingMeasureNameFromQueryResult",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, measure_name, AVG(CASE WHEN
 measure_name IN ('memory_cached', 'disk_used', 'disk_free') THEN measure_value::double
 ELSE NULL END) as avg_1h, MIN(CASE WHEN measure_name IN ('memory_free', 'cpu_idle')
 THEN measure_value::double ELSE NULL END) as min_1h, SUM(CASE WHEN measure_name
 IN ('disk_io_writes') THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name IN ('memory_used', 'cpu_user', 'cpu_system') THEN
 measure_value::double ELSE NULL END) as max_1h FROM raw_data.devops WHERE time
 BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND
 measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 measure_name, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_2",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
```

```
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MeasureNameColumn" : "measure_name",
            "MultiMeasureMappings" : {
                "MultiMeasureAttributeMappings" : [
                    {
                        "SourceColumn" : "avg_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "min_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "p0_1h"
                    },
                    {
                        "SourceColumn" : "sum_1h",
                        "MeasureValueType" : "DOUBLE"
                    },
                    {
                        "SourceColumn" : "max_1h",
                        "MeasureValueType" : "DOUBLE",
                        "TargetMultiMeasureAttributeName": "p100_1h"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

이 예제의 매핑은 속성 avg_1h, p0_1h, sum_1h, p100_1h로 다중 측정 레코드를 생성하고 쿼리 결과의 measure_name 열 값을 대상 테이블의 다중 측정 레코드의 측정 이름으로 사용합니다. 또한 이전 예제에서는 선택적으로 매핑의 하위 집합과 TargetMultiMeasureAttributeName 함께 를 사용하여 속성의

이름을 바꿉니다. 예를 들어 min_1h의 이름이 p0_1h로 변경되고 max_1h의 이름이 p100_1h로 변경됩니다.

다음은 이 예약된 쿼리가 구체화되면 대상 테이블의 스키마입니다. 다음 결과의 LiveAnalytics 속성 유형에 대한 Timestream에서 볼 수 있듯이 결과는 다중 측정 레코드로 구체화됩니다. 측정 스키마를 보면 쿼리 결과에 표시된 값에 해당하는 9개의 다른 측정 이름이 수집되었습니다.

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
시간	타임스탬프	TIMESTAMP
sum_1h	double	MULTI
p100_1h	double	MULTI
p0_1h	double	MULTI
평균_1시간	double	MULTI

다음은 SHOW MEASURES 쿼리로 얻은 해당 측정값입니다.

measure_name	data_type	차원
cpu_idle	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_system	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_user	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_free	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]

measure_name	data_type	차원
disk_io_writes	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_used	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
메모리_캐치	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
memory_free	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
memory_free	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]

예: 속성이 서로 다른 다양한 다중 측정 레코드에 결과 매핑

다음 예제에서는 쿼리 결과의 다양한 열을 측정값 이름이 다른 다양한 다중 측정 레코드에 매핑하는 방법을 보여줍니다. 다음과 같은 예약된 쿼리 정의가 표시되는 경우 쿼리의 결과에는 리전, 시간, avg_mem_cached_1h, min_mem_free_1h, max_mem_used_1h, total_disk_io_writes_1h, avg_disk_used_1h, avg_disk_free_1h, max_cpu_user_1h, max_cpu_system_1h, min_cpu_system_1h 열`region`이 차원에 매핑되고 시간 열에 매핑`hour`됩니다.

의 MixedMeasureMappings 속성입니다 TargetConfiguration.TimestreamConfiguration 는 측정값을 파생 테이블의 다중 측정 레코드에 매핑하는 방법을 지정합니다.

이 특정 예에서 avg_mem_cached_1h, min_mem_free_1h, max_mem_used_1h는 측정 이름이 mem_aggregates인 하나의 다중 측정 레코드에 사용되고, total_disk_io_writes_1h, avg_disk_used_1h, avg_disk_free_1h는 측정 이름이 disk_aggregates인 다른 다중 측정 레코드에 사용되며, 마지막으로 max_cpu_user_1h, max_cpu_system_1h, min_cpu_system_1h는 측정 이름이 cpu_aggregates인 다른 다중 측정 레코드에 사용됩니다.

이러한 매핑에서는 선택적으로 TargetMultiMeasureAttributeName 를 사용하여 쿼리 결과 열의 이름을 대상 테이블에 다른 속성 이름으로 바꿀 수도 있습니다. 예를 들어 결과 열 avg_mem_cached_1h의 이름이 로, avgMemCachedtotal_disk_io_writes_1h의 이름이 totalIOWrites로 변경됩니다.

다중 측정 레코드에 대한 매핑을 정의할 때 Timestream for 는 쿼리 결과의 모든 행을 LiveAnalytics 검사하고 NULL 값이 있는 열 값을 자동으로 무시합니다. 따라서 측정값 이름이 여러 개인 매핑의 경우 매핑의 해당 그룹에 대한 모든 열 값이 지정된 행NULL에 해당하는 경우 해당 측정값 이름에 대한 값이 해당 행에 수집되지 않습니다.

예를 들어 다음 매핑에서 avg_mem_cached_1h, min_mem_free_1h 및 max_mem_used_1h는 mem_aggregates라는 이름을 측정하도록 매핑됩니다. 쿼리 결과의 지정된 행에 대해 이러한 모든 열 값이 인 경우 NULL의 Timestream LiveAnalytics 은 해당 행의 측정값 mem_aggregates를 수집하지 않습니다. 지정된 행에 대한 9개의 열이 모두 NULL인 경우 오류 보고서에 사용자 오류가 보고됩니다.

```
{
    "Name" : "AggsInDifferentMultiMeasureRecords",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(CASE WHEN measure_name
 = 'memory_cached' THEN measure_value::double ELSE NULL END) as avg_mem_cached_1h,
 MIN(CASE WHEN measure_name = 'memory_free' THEN measure_value::double ELSE
 NULL END) as min_mem_free_1h, MAX(CASE WHEN measure_name = 'memory_used' THEN
 measure_value::double ELSE NULL END) as max_mem_used_1h, SUM(CASE WHEN measure_name =
 'disk_io_writes' THEN measure_value::double ELSE NULL END) as total_disk_io_writes_1h,
 AVG(CASE WHEN measure_name = 'disk_used' THEN measure_value::double ELSE NULL END) as
 avg_disk_used_1h, AVG(CASE WHEN measure_name = 'disk_free' THEN measure_value::double
 ELSE NULL END) as avg_disk_free_1h, MAX(CASE WHEN measure_name = 'cpu_user' THEN
 measure_value::double ELSE NULL END) as max_cpu_user_1h, MAX(CASE WHEN measure_name
 = 'cpu_system' THEN measure_value::double ELSE NULL END) as max_cpu_system_1h,
 MIN(CASE WHEN measure_name = 'cpu_idle' THEN measure_value::double ELSE NULL END)
 as min_cpu_system_1h FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime,
 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND measure_name IN ('memory_cached',
 'memory_free', 'memory_used', 'disk_io_writes', 'disk_used', 'disk_free', 'cpu_user',
 'cpu_system', 'cpu_idle') GROUP BY region, bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_3",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
```

```
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MixedMeasureMappings" : [
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "mem_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "avg_mem_cached_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "avgMemCached"
                        },
                        {
                            "SourceColumn" : "min_mem_free_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "max_mem_used_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "maxMemUsed"
                        }
                    ]
                },
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "disk_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "total_disk_io_writes_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "totalIOWrites"
                        },
                        {
                            "SourceColumn" : "avg_disk_used_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "avg_disk_free_1h",
                            "MeasureValueType" : "DOUBLE"
                        }
                    ]
```

```
                },
                {
                    "MeasureValueType" : "MULTI",
                    "TargetMeasureName" : "cpu_aggregates",
                    "MultiMeasureAttributeMappings" : [
                        {
                            "SourceColumn" : "max_cpu_user_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "max_cpu_system_1h",
                            "MeasureValueType" : "DOUBLE"
                        },
                        {
                            "SourceColumn" : "min_cpu_idle_1h",
                            "MeasureValueType" : "DOUBLE",
                            "TargetMultiMeasureAttributeName": "minCpuIdle"
                        }
                    ]
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

다음은 이 예약된 쿼리가 구체화되면 대상 테이블의 스키마입니다.

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
시간	타임스탬프	TIMESTAMP

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
minCpuIdle	double	MULTI
max_cpu_system_1h	double	MULTI
max_cpu_user_1h	double	MULTI
avgMemCached	double	MULTI
maxMemUsed	double	MULTI
min_mem_free_1h	double	MULTI
평균_디스크_무료_1시간	double	MULTI
avg_disk_used_1h	double	MULTI
totalIOWrites	double	MULTI

다음은 SHOW MEASURES 쿼리로 얻은 해당 측정값입니다.

measure_name	data_type	차원
cpu_aggregates	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_aggregates	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]
mem_aggregates	다중	[{'dimension_name': 'region', 'data_type': 'varchar'}]

예: 쿼리 결과의 측정 이름을 사용하여 결과를 단일 측정 레코드에 매핑

다음은 결과가 단일 측정 레코드로 구체화된 예약된 쿼리의 예입니다. 이 예제에서 쿼리 결과에는 대상 테이블에서 값이 측정 이름으로 사용되는 measure_name 열이 있습니다. 의

MixedMeasureMappings 속성을 사용하여 쿼리 결과 열을 대상 테이블의 스칼라 측정값에 매핑하도록 TargetConfigurationTimestreamConfiguration 지정합니다.

다음 예제 정의에서 쿼리 결과는 9개의 고유한 measure_name 값으로 예상됩니다. 매핑에 이러한 모든 측정값 이름을 나열하고 해당 측정값 이름의 단일 측정값에 사용할 열을 지정합니다. 예를 들어 이 매핑에서 지정된 결과 행에 대해 memory_cached의 측정 이름이 표시되는 경우 avg_1h 열의 값은 데이터가 대상 테이블에 기록될 때 측정값의 값으로 사용됩니다. 선택적으로 TargetMeasureName 를 사용하여 이 값에 대한 새 측정 이름을 제공할 수 있습니다.

```
{
    "Name" : "UsingMeasureNameColumnForSingleMeasureMapping",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, measure_name, AVG(CASE WHEN
 measure_name IN ('memory_cached', 'disk_used', 'disk_free') THEN measure_value::double
 ELSE NULL END) as avg_1h, MIN(CASE WHEN measure_name IN ('memory_free', 'cpu_idle')
 THEN measure_value::double ELSE NULL END) as min_1h, SUM(CASE WHEN measure_name
 IN ('disk_io_writes') THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name IN ('memory_used', 'cpu_user', 'cpu_system') THEN
 measure_value::double ELSE NULL END) as max_1h FROM raw_data.devops WHERE time
 BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h AND
 measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 bin(time, 1h), measure_name",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_4",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MeasureNameColumn" : "measure_name",
```

```
            "MixedMeasureMappings" : [
                {
                    "MeasureName" : "memory_cached",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h",
                    "TargetMeasureName" : "AvgMemCached"
                },
                {
                    "MeasureName" : "disk_used",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h"
                },
                {
                    "MeasureName" : "disk_free",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_1h"
                },
                {
                    "MeasureName" : "memory_free",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_1h",
                    "TargetMeasureName" : "MinMemFree"
                },
                {
                    "MeasureName" : "cpu_idle",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_1h"
                },
                {
                    "MeasureName" : "disk_io_writes",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "sum_1h",
                    "TargetMeasureName" : "total-disk-io-writes"
                },
                {
                    "MeasureName" : "memory_used",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h",
                    "TargetMeasureName" : "maxMemUsed"
                },
                {
                    "MeasureName" : "cpu_user",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h"
```

```
                },
                {
                    "MeasureName" : "cpu_system",
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_1h"
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

다음은 이 예약된 쿼리가 구체화되면 대상 테이블의 스키마입니다. 스키마에서 볼 수 있듯이 테이블은 단일 측정 레코드를 사용합니다. 테이블에 대한 측정 스키마를 나열하면 사양에 제공된 매핑을 기반으로 에 작성된 9개의 측정값이 표시됩니다.

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
시간	타임스탬프	TIMESTAMP
measure_value::double	double	MEASURE_VALUE

다음은 SHOW MEASURES 쿼리로 얻은 해당 측정값입니다.

measure_name	data_type	차원
AvgMemCached	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]

measure_name	data_type	차원
MinMemFree	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_idle	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_system	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
cpu_user	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_free	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
disk_used	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
maxMemUsed	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
total-disk-io-writes	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]

예: 쿼리 결과 열을 측정 이름으로 사용하여 결과를 단일 측정 레코드에 매핑

이 예제에서는 결과에 측정값 이름 열이 없는 쿼리가 있습니다. 대신 출력을 단일 측정 레코드에 매핑할 때 쿼리 결과 열 이름을 측정 이름으로 사용하려고 합니다. 이전에는 유사한 결과가 다중 측정 레코드에 기록된 예가 있었습니다. 이 예제에서는 애플리케이션 시나리오에 맞는 경우 단일 측정 레코드에 매핑하는 방법을 알아봅니다.

다시 한 번, 의 MixedMeasureMappings 속성을 사용하여 이 매핑을 지정합니다 TargetConfigurationTimestreamConfiguration. 다음 예제에서는 쿼리 결과에 9개의 열이 있음을 볼 수 있습니다. 결과 열을 측정 이름으로 사용하고 값을 단일 측정 값으로 사용합니다.

예를 들어 쿼리 결과의 지정된 행에 대해 열 이름 avg_mem_cached_1h는 열과 연결된 열 이름 및 값으로 사용되고 avg_mem_cached_1h는 단일 측정 레코드의 측정값으로 사용됩니다.

TargetMeasureName 를 사용하여 대상 테이블에서 다른 측정 이름을 사용할 수도 있습니다. 예를 들어, 열 sum_1h의 값의 경우 매핑은 total_disk_io_writes_1h를 대상 테이블의 측정 이름으로 사용하도록 지정합니다. 열의 값이 이면 NULL해당 측정값은 무시됩니다.

```
{
    "Name" : "SingleMeasureMappingWithoutMeasureNameColumnInQueryResult",
    "QueryString" : "SELECT region, bin(time, 1h) as hour, AVG(CASE WHEN measure_name
 = 'memory_cached' THEN measure_value::double ELSE NULL END) as avg_mem_cached_1h,
 AVG(CASE WHEN measure_name = 'disk_used' THEN measure_value::double ELSE NULL END) as
 avg_disk_used_1h, AVG(CASE WHEN measure_name = 'disk_free' THEN measure_value::double
 ELSE NULL END) as avg_disk_free_1h, MIN(CASE WHEN measure_name = 'memory_free' THEN
 measure_value::double ELSE NULL END) as min_mem_free_1h, MIN(CASE WHEN measure_name =
 'cpu_idle' THEN measure_value::double ELSE NULL END) as min_cpu_idle_1h, SUM(CASE WHEN
 measure_name = 'disk_io_writes' THEN measure_value::double ELSE NULL END) as sum_1h,
 MAX(CASE WHEN measure_name = 'memory_used' THEN measure_value::double ELSE NULL END)
 as max_mem_used_1h, MAX(CASE WHEN measure_name = 'cpu_user' THEN measure_value::double
 ELSE NULL END) as max_cpu_user_1h, MAX(CASE WHEN measure_name = 'cpu_system' THEN
 measure_value::double ELSE NULL END) as max_cpu_system_1h FROM raw_data.devops WHERE
 time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND bin(@scheduled_runtime, 1h) - 2h
 AND measure_name IN ('memory_free', 'memory_used', 'memory_cached', 'disk_io_writes',
 'disk_used', 'disk_free', 'cpu_user', 'cpu_system', 'cpu_idle') GROUP BY region,
 bin(time, 1h)",
    "ScheduleConfiguration" : {
        "ScheduleExpression" : "cron(0 0/1 * * ? *)"
    },
    "NotificationConfiguration" : {
        "SnsConfiguration" : {
            "TopicArn" : "******"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******",
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName" : "derived",
            "TableName" : "dashboard_metrics_1h_agg_5",
            "TimeColumn" : "hour",
            "DimensionMappings" : [
                {
                    "Name": "region",
                    "DimensionValueType" : "VARCHAR"
                }
            ],
            "MixedMeasureMappings" : [
```

```
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_mem_cached_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_disk_used_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "avg_disk_free_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_mem_free_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "min_cpu_idle_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "sum_1h",
                    "TargetMeasureName" : "total_disk_io_writes_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_mem_used_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_cpu_user_1h"
                },
                {
                    "MeasureValueType" : "DOUBLE",
                    "SourceColumn" : "max_cpu_system_1h"
                }
            ]
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
```

```
            "EncryptionOption": "SSE_S3"
        }
    }
}
```

다음은 이 예약된 쿼리가 구체화되면 대상 테이블의 스키마입니다. 보시다시피 대상 테이블은 단일 측정 값이 이중인 레코드를 저장합니다. 마찬가지로 테이블의 측정 스키마에는 9개의 측정 이름이 표시됩니다. 또한 매핑 이름이 sum_1h에서 total_disk_io_writes_1h로 변경된 이후로 측정 이름 total_disk_io_writes_1h가 존재한다는 점에 유의하세요.

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
region	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
시간	타임스탬프	TIMESTAMP
measure_value::double	double	MEASURE_VALUE

다음은 SHOW MEASURES 쿼리로 얻은 해당 측정값입니다.

measure_name	data_type	차원
평균_디스크_무료_1시간	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
avg_disk_used_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
평균_mem_cached_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
max_cpu_system_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
max_cpu_user_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]

measure_name	data_type	차원
max_mem_used_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
min_cpu_idle_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
min_mem_free_1h	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]
total-disk-io-writes	double	[{'dimension_name': 'region', 'data_type': 'varchar'}]

예약된 쿼리 알림 메시지

이 섹션에서는 예약된 쿼리의 상태를 생성, 삭제, 실행 또는 업데이트할 LiveAnalytics 때 Timestream에서 에 보낸 메시지를 설명합니다.

알림 메시지 이름	구조	설명
CreatingNotificationMessage	`CreatingNotificationMessage { String arn; NotificationType type; }`	이 알림 메시지는 에 대한 응답을 보내기 전에 전송됩니다 `CreateScheduledQuery` . 이 알림을 전송한 후 예약된 쿼리가 활성화됩니다. arn - 생성 중인 예약된 쿼리 ARN의 입니다. 유형 - SCHEDULED_QUERY_CREATING
UpdateNotificationMessage	`UpdateNotificationMessage { String arn; NotificationType type;`	이 알림 메시지는 예약된 쿼리가 업데이트될 때 전송됩니다. 의 Timestream은 다음과 같이 복구할 LiveAnalytics 수 없는 오류가 발생하는 경우 예약된

알림 메시지 이름	구조	설명
	`QueryState state;` `}`	쿼리를 자동으로 비활성화할 수 있습니다. • AssumeRole 실패 • 고객 관리형 KMS 키가 지정될 때 와 통신KMS할 때 발생하는 4xx 오류입니다. • 예약된 쿼리를 실행하는 동안 발생하는 4xx 오류입니다. • 쿼리 결과를 수집하는 동안 발생하는 모든 4xx 오류 arn - 업데이트 중인 예약된 쿼리ARN의 입니다. 유형 - SCHEDULED_QUERY_UPDATE 상태 - ENABLED 또는 DISABLED
DeleteNotificationMessage	`DeletionNotificationMessage {` `String arn;` `NotificationType type;` `}`	이 알림 메시지는 예약된 쿼리가 삭제될 때 전송됩니다. arn - 생성 중인 예약된 쿼리 ARN의 입니다. 유형 - SCHEDULED_QUERY_DELETED

알림 메시지 이름	구조	설명
SuccessNotificationMessage	SuccessNotificationMessage { NotificationType type; String arn; Date nextInvocationEpochSecond; ScheduledQueryRunSummary runSummary; } ScheduledQueryRunSummary { Date invocationTime; Date triggerTime; String runStatus; ExecutionStats executionstats; ErrorReportLocation errorReportLocation; String failureReason; } ExecutionStats { Long bytesMetered; Long dataWrites; Long queryResultRows; Long recordsIngested; Long executionTimeInMillis; } ErrorReportLocation {	이 알림 메시지는 예약된 쿼리가 실행되고 결과가 성공적으로 수집되면 전송됩니다. ARN - 삭제 중인 예약된 쿼리 ARN의 입니다. NotificationType - AUTO_TRIGGER_SUCCESS 또는 MANUAL_TRIGGER_SUCCESS. nextInvocationEpoch둘째 - 다음에 의 LiveAnalytics Timestream이 예약된 쿼리를 실행합니다. runSummary - 예약된 쿼리 실행에 대한 정보입니다.

알림 메시지 이름	구조	설명
	S3ReportLocation s3ReportLocation; } S3ReportLocation { String bucketName; String objectKey; }	

알림 메시지 이름	구조	설명
FailureNotificationMessage	FailureNotificationMessage { NotificationType type; String arn; ScheduledQueryRunSummary runSummary; } ScheduledQueryRunSummary { Date invocationTime; Date triggerTime; String runStatus; ExecutionStats executionstats; ErrorReportLocation errorReportLocation; String failureReason; } ExecutionStats { Long bytesMetered; Long dataWrites; Long queryResultRows; Long recordsIngested; Long executionTimeInMillis; } ErrorReportLocation { S3ReportLocation s3ReportLocation; }	이 알림 메시지는 예약된 쿼리 실행 중에 실패가 발생하거나 쿼리 결과를 수집할 때 전송됩니다. arn - 실행 중인 예약된 쿼리 ARN의 입니다. 유형 - AUTO_TRIGGER_FAILURE 또는 MANUAL_TRIGGER_FAILURE. runSummary - 예약된 쿼리 실행에 대한 정보입니다.

알림 메시지 이름	구조	설명
	`S3ReportLocation { String bucketName; String objectKey; }`	

예약된 쿼리 오류 보고서

이 섹션에서는 예약된 쿼리를 실행하여 오류가 발생할 LiveAnalytics 때 Timestream에서 생성된 오류 보고서의 위치, 형식 및 이유를 설명합니다.

주제

- 예약된 쿼리 오류는 이유를 보고합니다.
- 예약된 쿼리 오류 보고서 위치
- 예약된 쿼리 오류 보고서 형식
- 예약된 쿼리 오류 유형
- 예약된 쿼리 오류 보고서 예제

예약된 쿼리 오류는 이유를 보고합니다.

복구 가능한 오류에 대한 오류 보고서가 생성됩니다. 복구할 수 없는 오류에 대해서는 오류 보고서가 생성되지 않습니다. 의 Timestream은 복구할 LiveAnalytics 수 없는 오류가 발생할 때 예약된 쿼리를 자동으로 비활성화할 수 있습니다. 다음이 포함됩니다.

- `AssumeRole` 실패
- 고객 관리형 KMS 키가 지정될 때 와 통신KMS할 때 발생하는 모든 4xx 오류
- 예약된 쿼리가 실행될 때 발생하는 모든 4xx 오류
- 쿼리 결과를 수집하는 동안 발생하는 모든 4xx 오류

복구할 수 없는 오류의 경우 용 Timestream은 복구할 수 없는 오류 메시지와 함께 실패 알림을 LiveAnalytics 보냅니다. 예약된 쿼리가 비활성화되었음을 나타내는 업데이트 알림도 전송됩니다.

예약된 쿼리 오류 보고서 위치

예약된 쿼리 오류 보고서 위치에는 다음과 같은 명명 규칙이 있습니다.

```
s3://customer-bucket/customer-prefix/
```

다음은 예약된 쿼리의 예입니다ARN.

```
arn:aws:timestream:us-east-1:000000000000:scheduled-query/test-query-hd734tegrgfd
```

```
s3://customer-bucket/customer-prefix/test-query-hd734tegrgfd/<InvocationTime>/<Auto or
 Manual>/<Actual Trigger Time>
```

Auto 는 LiveAnalytics 및 에 대해 Timestream에서 자동으로 예약된 예약된 쿼리를 나타냅니다.*Manual* 는 Amazon Timestream for Query의 ExecuteScheduledQuery API 작업을 통해 사용자가 수동으로 트리거한 예약된 LiveAnalytics 쿼리를 나타냅니다. 에 대한 자세한 내용은 섹션을 ExecuteScheduledQuery참조하세요ExecuteScheduledQuery.

예약된 쿼리 오류 보고서 형식

오류 보고서의 JSON 형식은 다음과 같습니다.

```
{
    "reportId": <String>,            // A unique string ID for all error reports
 belonging to a particular scheduled query run
    "errors": [ <Error>, ... ],      // One or more errors
}
```

예약된 쿼리 오류 유형

Error 객체는 다음 세 가지 유형 중 하나일 수 있습니다.

- 레코드 수집 오류

```
{
    "reason": <String>,              // The error message String
    "records": [ <Record>, ... ],    // One or more rejected records )
}
```

- 행 구문 분석 및 검증 오류

```
{
    "reason": <String>,        // The error message String
    "rawLine": <String>,       // [Optional] The raw line String that is being parsed
 into record(s) to be ingested. This line has encountered the above-mentioned parse
 error.
}
```

- 일반 오류

```
{
    "reason": <String>,        // The error message
}
```

예약된 쿼리 오류 보고서 예제

다음은 수집 오류로 인해 생성된 오류 보고서의 예입니다.

```
{
    "reportId": "C9494AABE012D1FBC162A67EA2C18255",
    "errors": [
        {
            "reason": "The record timestamp is outside the time range
 [2021-11-12T14:18:13.354Z, 2021-11-12T16:58:13.354Z) of the memory store.",
            "records": [
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_1",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_1",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "3.141592653589793",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
```

```
                    "time": "1637166175635000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                },
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_2",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_2",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "6.283185307179586",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175636000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                },
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_3",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_3",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "9.42477796076938",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175637000000",
                    "timeUnit": "NANOSECONDS",
```

```
                    "version": null
                },
                {
                    "dimensions": [
                        {
                            "name": "dim0",
                            "value": "d0_4",
                            "dimensionValueType": null
                        },
                        {
                            "name": "dim1",
                            "value": "d1_4",
                            "dimensionValueType": null
                        }
                    ],
                    "measureName": "random_measure_value",
                    "measureValue": "12.566370614359172",
                    "measureValues": null,
                    "measureValueType": "DOUBLE",
                    "time": "1637166175638000000",
                    "timeUnit": "NANOSECONDS",
                    "version": null
                }
            ]
        }
    ]
}
```

예약된 쿼리 패턴 및 예제

이 섹션에서는 예약된 쿼리 end-to-end의 사용 패턴과 예제를 설명합니다.

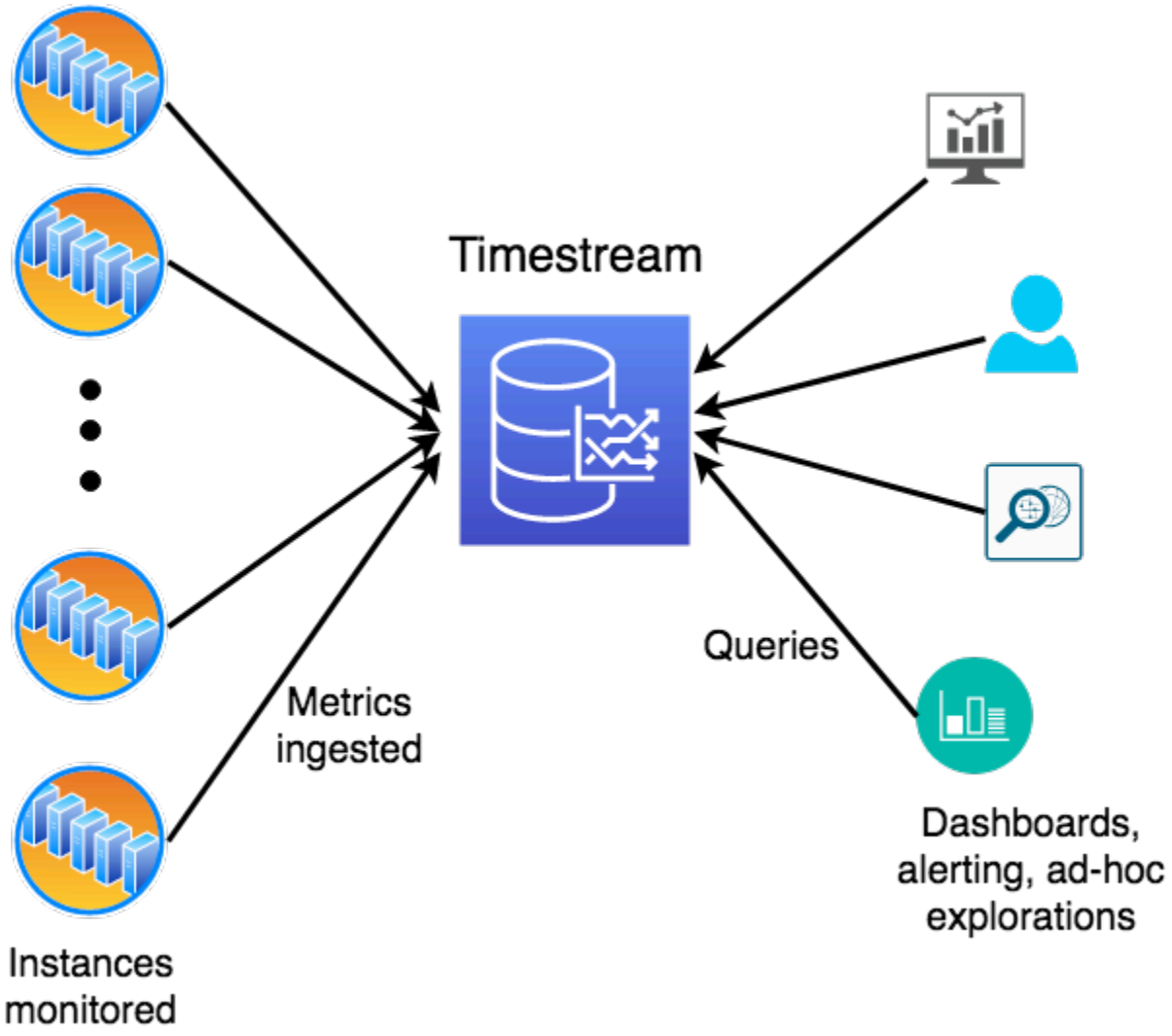
주제

- 예약된 쿼리 샘플 스키마
- 예약된 쿼리 패턴
- 예약된 쿼리 예제

예약된 쿼리 샘플 스키마

이 예제에서는 대규모 서버 플릿의 DevOps 시나리오 모니터링 지표를 모방한 샘플 애플리케이션을 사용합니다. 사용자는 비정상적인 리소스 사용량에 대해 알리고, 집계 플릿 동작 및 사용률에 대한 대시

보드를 생성하고, 최근 및 과거 데이터에 대한 정교한 분석을 수행하여 상관관계를 찾고자 합니다. 다음 다이어그램은 모니터링된 인스턴스 세트가 의 Timestream으로 지표를 내보내는 설정을 보여줍니다 LiveAnalytics. 또 다른 동시 사용자 집합은 쿼리와 수집이 병렬로 실행되는 알림, 대시보드 또는 임시 분석에 대한 쿼리를 발행합니다.



모니터링 중인 애플리케이션은 전 세계 여러 리전에 배포되는 대규모 서비스로 모델링됩니다. 각 리전은 리전 내 인프라 측면에서 격리 수준이 있는 셀이라는 여러 조정 단위로 추가로 세분화됩니다. 각 셀은 소프트웨어 격리 수준을 나타내는 사일로로 추가로 세분화됩니다. 각 사일로에는 서비스의 격리된 인스턴스 하나를 구성하는 5개의 마이크로서비스가 있습니다. 각 마이크로서비스에는 인스턴스 유형과 OS 버전이 서로 다른 여러 서버가 있으며, 이 서버는 3개의 가용 영역에 배포됩니다. 지표를 내보내는 서버를 식별하는 이러한 속성은 의 Timestream에서 차원으로 모델링됩니다 LiveAnalytics. 이 아키텍처에는 계층 구조(예: 리전, 셀, 사일로, microservice_name)와 계층 구조(예: instance_type 및 availability_zone)를 가로지르는 기타 차원의 계층 구조가 있습니다.

애플리케이션은 다양한 지표(예: cpu_user 및 memory_free)와 이벤트(예: task_completed 및 gc_reclaimed)를 내보냅니다. 각 지표 또는 이벤트는 이를 내보내는 서버를 고유하게 식별하는 8개 차원(예: 리전 또는 셀)과 연결됩니다. 데이터는 측정 이름 지표가 있는 다중 측정 레코드에 함께 저장된 20개의 지표로 작성되며 5개의 모든 이벤트는 측정 이름 이벤트가 있는 다른 다중 측정 레코드에 함께 저장됩니다. 데이터 모델, 스키마 및 데이터 생성은 오픈 소스 데이터 생성기 에서 찾을 수 있습니다. 스키마 및 데이터 배포 외에도 데이터 생성기는 여러 라이터를 사용하여 데이터를 병렬로 수집하며, 에 대한 Timestream의 수집 조정을 사용하여 초당 수백만 개의 측정값 LiveAnalytics 을 수집하는 예를 제공합니다. 아래에는 스키마(표 및 측정 스키마)와 데이터 세트의 일부 샘플 데이터가 나와 있습니다.

주제

다중 측정 레코드

테이블 스키마

다음은 다중 측정 레코드를 사용하여 데이터를 수집한 후의 테이블 스키마입니다. DESCRIBE 쿼리의 출력입니다. 데이터가 데이터베이스 raw_data 및 테이블 데보로 수집된다고 가정하면 아래는 쿼리입니다.

```
DESCRIBE "raw_data"."devops"
```

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
availability_zone	varchar	DIMENSION
microservice_name	varchar	DIMENSION
instance_name	varchar	DIMENSION
process_name	varchar	DIMENSION
os_version	varchar	DIMENSION
jdk_version	varchar	DIMENSION

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
셀	varchar	DIMENSION
region	varchar	DIMENSION
사일로	varchar	DIMENSION
instance_type	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
시간	타임스탬프	TIMESTAMP
memory_free	double	MULTI
cpu_steal	double	MULTI
cpu_iowait	double	MULTI
cpu_user	double	MULTI
메모리_캐치	double	MULTI
disk_io_reads	double	MULTI
cpu_hi	double	MULTI
읽기당 지연 시간	double	MULTI
network_bytes_out	double	MULTI
cpu_idle	double	MULTI
disk_free	double	MULTI
메모리_사용	double	MULTI
cpu_system	double	MULTI
file_descriptors_in_use	double	MULTI

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
disk_used	double	MULTI
cpu_nice	double	MULTI
disk_io_writes	double	MULTI
cpu_si	double	MULTI
쓰기당 지연 시간	double	MULTI
network_bytes_in	double	MULTI
task_end_state	varchar	MULTI
gc_pause	double	MULTI
작업_완료	bigint	MULTI
gc_reclaimed	double	MULTI

스키마 측정

다음은 SHOW MEASURES 쿼리에서 반환되는 측정 스키마입니다.

```
SHOW MEASURES FROM "raw_data"."devops"
```

measure_name	data_type	차원
이벤트	다중	[{'data_type':'varchar','dimension_name':'availability_zone'},{'data_type':'varchar','dimension_name':'microservice_name'},{'data_type':'varchar','dimension_name':'instance_name'},{'data_type','dimension_name',{'da

measure_name	data_type	차원
		ta_type':'varchar','dimensi on_name":'jdk_version'},{' data_type':'varchar','dimen sion_name':'cell'},{'data_t ype':'varchar","varchar", 'dimension_name"":'
지표	다중	[{'data_type':'varchar','di mension_name':'availability _zone'},{'data_type':'varch ar','dimension_name',{'data _type':'varchar','dimension _name':'instance_name'},{'d ata_type','dimension_name', "data_type':'varchar','dim ension_name':'cell'},{'data _type':'varchar','dimension _name":'region'},{'data_ty pe':'varchar','dimension_na me','silo"

예제 데이터

re	설	사 을 토	a ity	m ic	in n	in ty	o n	p a	jo o	m a	T	c	c m	c	c	c t	c	c	c	c	마 도 르 _ 사 울	마 도 르 _ 키 츠	d e	을 가 등 지 안 시 간	d ri	스 가 등 지 안 시 간	d	d	n y	n y	fil ri n	m e	ta st	g	직 을 _ 온 토	gs med	recla
u e	u e - c	u e - c	u e -1	a	i- za k- a	m e	A			지 꼬	1 1 1	6	0	3	0	0	0	0	0	5	8	5	9	3	2	6	2	8	4	3	5					

re	설	사용도	ar	mi	in	in	op	pa	jo	ma	Ti	cr	cr	cr	cr	cr	cr	cr	cr	cr	마도르_사용	마도르_키	de	일가능지인사건	dr	스가능지인사건	de	de	ny	ny	fil	me	ta	ge	즉일_온도	ge	recla
		sil			us																																	med
					e-																																	
					cs																																	
					il																																	
					0																																	
					zo																																	
					m																																	
us	us	us	us	at	i-	me	AK			지포	11	5	0	3	0	0	0	0	0	9	3	5	3	2	9	5	3	7	5	6	2					
	e-	e-	e-	-1	k-						1																											
	co	co	cs		au																																	
			il		e-																																	
					cs																																	
					il																																	
					0																																	
					zo																																	
					m																																	

re	설	시 울 토	a ity	m ic	in n	ir ty	o n	p a	jo o	m a	T	c	c m	c	c	c t	c	c	c	마 도 리 _ 시 옹	마 도 리 _ 키 치	d e	일 기 등 자 인 시 간	d ri	스 기 등 자 인 시 간	d	d	n y	n y	fil ri n	m e	ta st	g	즈 일 _ 온 토	gs_e med ses_recla
u e	u e - c	u e - c s il	u e -1	a	i- za k a u e - c s il 0 z m	m e	A			ㅈ 표	1 1 1.	4	0	4	0	0	0	0	0	9	4	5	3.	8	3	7.	6	8	5	4	8				
u e	u e - c	u e - c s il	u e -1	a	i- za k a u e - c s il 0 z m	m e	A			ㅈ 표	1 1 1.	3	0	5	0	0	0	0	0	4	7.	2	4	4	3	8	6	2	1	2	1				

re	설	사용도	ac ity	mic	in n	in ty	on	pa	jo o	ma	T	c	c m	c	c	c t	c	c	c	c	마도르_사용	마도르_키	de	일기등지인시간	dri	스기등지인시간	d	d	ny	ny	fil rin	me	ta st	g	즉일_온도	gs_reclaimed
us e	us e- co	us e- co sil	us e- -1	at	i- za k- at us e- - co sil 0 zo m	me	AI			지포	11 11	5	0	3	0	0	0	0	0	0	5	3	8	5	7	1	8	6	7	6	5	3				
us e	us e- co	us e- co sil	us e- -1	at	i- za k- at us e- - co sil 0 zo m			호스트_관리자	JI	이분트	11 11																					7	SIIS	8	3	64	3.8 WRE

re	설	사용도	ac ity	mic	in n	in ty	op on	pa	jo o	ma	Th	co	co m	co	co	co t	co	co	co	마도르_사용	마도르_키 츠	de	일기등지인사건	dri	스기등지인사건	d	d	ny	ny yt	fil rin	me	ta st	ga	즉일_온도	gs_reclamed
us e	us e- co	us e- co sil	us e- -1	at	i- za k- at us e- co sil 0 zo m			호스트_관리자	JI	0분트	11 11																				7	SIlT	2	4	73	345_WJL
us e	us e- co	us e- co sil	us e- -1	at	i- za k- at us e- co sil 0 zo m			호스트_관리자	JI	0분트	11 11																				6	SIlT	6	2	73	2_3WJL

re	설	사용도	ac ity	mic	in n	in ty	op on	pa	jo o	ma	TI	cl	cl m	cl	cl	cl t	cl	cl	cl	마도르_사용	마도르_키	de	일기등지인시간	di ri	스기등지인시간	di	di	ny	ny yt	fil rin	me	ta st	ga	즉일_온도	gs_ med	recla
us e	us e- co	us e- co sil	us e- -1	ac	i- za k- ac us e- co sil 0 zo m			호스트_관리자	JI	0분트	11 11																				2	SI T	5	1	99	9_W JL
us e	us e- co	us e- co sil	us e- -1	ac	i- za k- ac us e- co sil 0 zo m			호스트_관리자	JI	0분트	11 11																				3	SI S	7	2	82	2_W RE

단일 측정 레코드

LiveAnalytics 또한 의 Timestream을 사용하면 시계열 레코드당 하나의 측정값으로 데이터를 수집할 수 있습니다. 다음은 단일 측정 레코드를 사용하여 수집할 때의 스키마 세부 정보입니다.

테이블 스키마

다음은 다중 측정 레코드를 사용하여 데이터를 수집한 후의 테이블 스키마입니다. DESCRIBE 쿼리의 출력입니다. 데이터가 데이터베이스 raw_data 및 테이블 데보로 수집된다고 가정하면 아래는 쿼리입니다.

```
DESCRIBE "raw_data"."devops_single"
```

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
availability_zone	varchar	DIMENSION
microservice_name	varchar	DIMENSION
인스턴스_이름	varchar	DIMENSION
process_name	varchar	DIMENSION
os_version	varchar	DIMENSION
jdk_version	varchar	DIMENSION
셀	varchar	DIMENSION
region	varchar	DIMENSION
사일로	varchar	DIMENSION
instance_type	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
시간	타임스탬프	TIMESTAMP
measure_value::double	double	MEASURE_VALUE

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
measure_value::bigint	bigint	MEASURE_VALUE
measure_value::varchar	varchar	MEASURE_VALUE

스키마 측정

다음은 SHOW MEASURES 쿼리에서 반환되는 측정 스키마입니다.

```
SHOW MEASURES FROM "raw_data"."devops_single"
```

measure_name	data_type	차원
cpu_hi	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
cpu_idle	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar'

measure_name	data_type	차원
		}, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
cpu_iowait	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
cpu_nice	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'

measure_name	data_type	차원
cpu_si	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
cpu_steal	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'

measure_name	data_type	차원
cpu_system	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
cpu_user	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'

measure_name	data_type	차원
disk_free	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
disk_io_reads	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'

measure_name	data_type	차원
disk_io_writes	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
disk_used	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'

measure_name	data_type	차원
file_descriptors_in_use	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
gc_pause	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'jdk_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar'

measure_name	data_type	차원
gc_reclaimed	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'jdk_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar'
읽기당 지연 시간	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'

measure_name	data_type	차원
쓰기당 지연 시간	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
메모리_캐치	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'

measure_name	data_type	차원
memory_free	double	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name': 'process_name', 'data_type': 'varchar'}, {'dimension_name': 'os_version', 'data_type': 'varchar'}, {'dimension_name': 'jdk_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': ' 리전', 'data_type': 'varchar'}, {'dimension_name': '사일로', 'data_type': 'varchar'}, {'dimension_name': 'instance_type', 'data_type': 'varchar'}]

measure_name	data_type	차원
메모리_사용	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
network_bytes_in	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'

measure_name	data_type	차원
network_bytes_out	double	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'varchar' }, {'dimension_name': 'cell', 'data_type': 'varchar'}, {'dimension_name': 'region', 'data_type': 'varchar'}, {'dimension_name'
작업_완료됨	bigint	[{'dimension_name': 'availabi lity_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'data_typ e': 'varchar'}, {'dimensi on_name': 'jdk_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar'

measure_name	data_type	차원
태스크_종료_상태	varchar	[{'dimension_name': 'availability_zone', 'data_type': 'varchar'}, {'dimension_name': 'microservice_name', 'data_type': 'varchar'}, {'dimension_name': 'instance_name', 'data_type': 'varchar'}, {'dimension_name', 'data_type': 'varchar'}, {'dimension_name': 'jdk_version', 'data_type': 'varchar'}, {'dimension_name': 'cell', 'data_type': 'varchar'

예제 데이터

availa ity_zc	micro ice_n	인스턴스_이름	proce ame	os_ve n	jdk_v on	셀	regio	사일로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu-west--1	약어	i-zaZsk-hercus-eu-west-1-cell-9-silo-2		AL20		eu-west--cell-9	eu-west-	eu-west--cell-9silo-2	r5.4xle	cpu_l	34:57	0.871		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
		0000 mazo com												
eu- west- -1	약 어	i- zaZs k- hercu s- eu- west -1- cell-9 - silo-2 0000 mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_i	34:57	3.462		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	약 어	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_i t	34:57	0.102		
eu- west- -1	약 어	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_r	34:57	0.630		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	약 어	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s	34:57	0.164		
eu- west- -1	약 어	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s	34:57	0.107		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	약 어	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_s m	34:57	0.457		
eu- west- -1	약 어	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	cpu_u	34:57	2044 년 94 월		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_f	34:57	72.51		
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_i eads	34:57	81.73		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_ rites	34:57	77.11		
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	disk_	34:57	89.42		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	file_d riptor n_use	34:57	30.08		
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com	서 버		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		gc_pa	34:57	60.28		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com	서 버		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		gc_re med	34:57	75.28		
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	읽 기 당 지 연 시 간	34:57	8.076		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	쓰 기 당 지 연 시 간	34:57	58.11		
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	mem(ched	34:57	87.56		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com	서 버		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		mem(ee	34:57	18.95		
eu- west- -1	헤 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	mem(ee	34:57	2052 년 97 월		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 르 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	메 모 리 _사 용	34:57	12.37		
eu- west- -1	헤 르 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	netwo ytes_	34:57	2065 년 31 월		

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 르 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com		AL20		eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2	r5.4xl e	netwo ytes_	34:57	0.514		
eu- west- -1	헤 르 클	i- zaZsv k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com	서 버		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		작 업 _완 료	34:57		69	

availa ity_zo	micro ice_n	인 스 턴 스_ 이 름	proce ame	os_ve n	jdk_v on	셀	regio	사 일 로	instar type	meas ame	Time	meas alue:: ble	meas alue:: int	measure_v alue::var char
eu- west- -1	헤 르 클	i- zaZs\ k- hercu s- eu- west -1- cell-9 - silo-2 0000(mazo com	서 버		JDK_	eu- west- - cell-9	eu- west-	eu- west- - cell-9 s ilo-2		task_ state	34:57			SUCCESS_ ITH_RESUL T

예약된 쿼리 패턴

이 섹션에서는 Amazon Timestream for LiveAnalytics Scheduled Queries를 사용하여 대시보드를 최적화하여 더 빠르고 저렴한 비용으로 로드할 수 있는 몇 가지 일반적인 패턴을 찾을 수 있습니다. 아래 예제에서는 DevOps 애플리케이션 시나리오를 사용하여 애플리케이션 시나리오와 관계없이 일반적으로 예약된 쿼리에 적용되는 주요 개념을 보여줍니다.

에 대한 Timestream의 예약된 쿼리를 LiveAnalytics 사용하면 에 대한 Timestream의 전체 SQL 표면적을 사용하여 쿼리를 표현할 수 있습니다 LiveAnalytics. 쿼리에는 소스 테이블이 하나 이상 포함되고, LiveAnalytics의 SQL 언어에 대해 Timestream에서 허용하는 집계 또는 기타 쿼리를 수행한 다음 의 Timestream의 다른 대상 테이블에 쿼리 결과를 구체화할 수 있습니다 LiveAnalytics. 쉽게 엑스포지션할 수 있도록 이 섹션에서는 예약된 쿼리의 이 대상 테이블을 파생 테이블 로 참조합니다.

다음은 이 섹션에서 다루는 핵심 사항입니다.

- 간단한 플릿 수준 집계를 사용하여 예약된 쿼리를 정의하고 몇 가지 기본 개념을 이해하는 방법을 설명합니다.
- 예약된 쿼리의 대상(파생된 테이블)의 결과를 소스 테이블의 결과와 결합하여 예약된 쿼리의 비용 및 성능 이점을 얻는 방법.
- 예약된 쿼리의 새로 고침 기간을 구성할 때의 장단점은 무엇입니까?
- 일부 일반적인 시나리오에 대해 예약된 쿼리 사용.
  - 특정 날짜 이전의 모든 인스턴스에서 마지막 데이터 포인트를 추적합니다.
  - 대시보드에 변수를 채우는 데 사용할 차원의 고유한 값입니다.
- 예약된 쿼리의 맥락에서 지연 도착 데이터를 처리하는 방법.
- 일회성 수동 실행을 사용하여 예약된 쿼리에 대한 자동 트리거에서 직접 다루지 않는 다양한 시나리오를 처리하는 방법.

주제

시나리오

다음 예제에서는 에 설명된 DevOps 모니터링 시나리오를 사용합니다예약된 쿼리 샘플 스키마.

이 예제에서는 예약된 쿼리에 대한 실행 상태 알림을 수신할 위치, 예약된 쿼리 실행 중에 발생한 오류에 대한 보고서를 수신할 위치, 예약된 쿼리가 작업을 수행하는 데 사용하는 IAM 역할에 대한 적절한 구성을 연결할 수 있는 예약된 쿼리 정의를 제공합니다.

앞의 옵션을 채우고, 대상(또는 파생) 테이블을 생성하고, 를 통해 AWS 를 실행한 후 이러한 예약된 쿼리를 생성할 수 있습니다CLI. 예를 들어 예약된 쿼리 정의가 파일 에 저장된다고 가정합니다`scheduled_query_example.json`. CLI 명령을 사용하여 쿼리를 생성할 수 있습니다.

```
aws timestream-query create-scheduled-query --cli-input-json  file://
scheduled_query_example.json --profile aws_profile --region us-east-1
```

앞의 명령에서 --profile 옵션을 사용하여 전달된 프로필에는 예약된 쿼리를 생성할 수 있는 적절한 권한이 있어야 합니다. 정책 및 권한에 대한 자세한 지침은 예약된 쿼리에 대한 자격 증명 기반 정책을 참조하세요.

간단한 플릿 수준 집계

이 첫 번째 예제에서는 간단한 예제 컴퓨팅 플릿 수준 집계를 사용하여 예약된 쿼리를 작업할 때 몇 가지 기본 개념을 안내합니다. 이 예제를 통해 다음을 배울 수 있습니다.

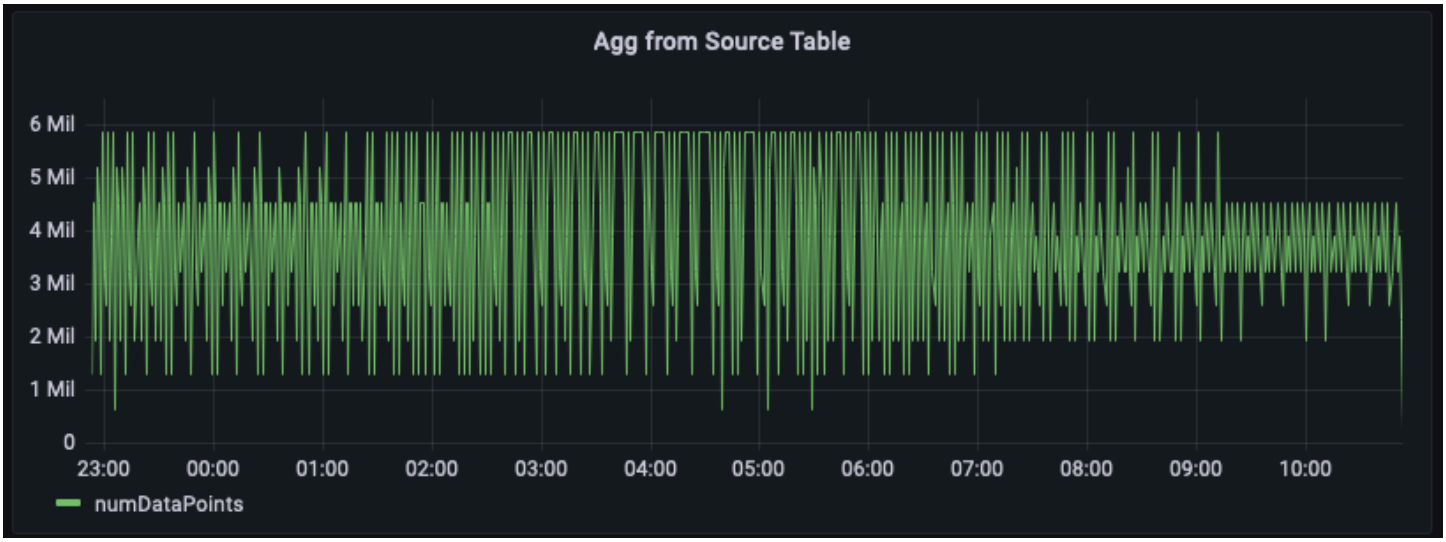
- 집계 통계를 얻고 예약된 쿼리에 매핑하는 데 사용되는 대시보드 쿼리를 가져오는 방법.
- 의 Timestream이 예약된 쿼리의 다양한 인스턴스 실행을 LiveAnalytics 관리하는 방법.
- 예약된 쿼리의 다양한 인스턴스가 시간 범위에서 겹치도록 할 수 있는 방법 및 예약된 쿼리의 결과를 사용하여 대시보드가 원시 데이터에 대해 계산된 동일한 집계와 일치하는 결과를 제공하도록 하기 위해 대상 테이블에서 데이터의 정확성을 유지하는 방법.
- 예약된 쿼리의 시간 범위 및 새로 고침 주기를 설정하는 방법.
- 예약된 쿼리의 결과를 셀프 서비스 방식으로 추적하여 쿼리 인스턴스의 실행 지연 시간이 대시보드 새로 고침의 허용 가능한 지연 시간 내에 있도록 조정하는 방법.

주제

- 소스 테이블에서 집계
- 집계를 사전 계산하기 위한 예약된 쿼리
- 파생 테이블에서 집계
- 소스 테이블과 파생 테이블의 결합 집계
- 자주 새로 고쳐지는 예약된 계산에서 집계

소스 테이블에서 집계

이 예제에서는 1분마다 지정된 리전 내의 서버에서 내보내는 지표 수를 추적합니다. 아래 그래프는 us-east-1 리전에 대해 이 시계열을 도표화한 예제입니다.



다음은 원시 데이터에서 이 집계를 계산하기 위한 예제 쿼리입니다. us-east-1 리전의 행을 필터링한 다음 20개의 지표(measure_name이 지표인 경우) 또는 5개의 이벤트(measure_name이 이벤트인 경우)를 고려하여 분당 합계를 계산합니다. 이 예제에서 그래프 그림은 방출되는 지표 수가 분당 150만 ~600만 개로 다양함을 보여줍니다. 이 시계열을 몇 시간(이 그림에서 지난 12시간) 동안 그래프로 표시할 때 원시 데이터에 대한 이 쿼리는 수억 개의 행을 분석합니다.

```
WITH grouped_data AS (
    SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name = 'metrics' THEN
 20 ELSE 5 END) as numDataPoints
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636699996445) AND
 from_milliseconds(1636743196445)
        AND region = 'us-east-1'
    GROUP BY region, measure_name, bin(time, 1m)
)
SELECT minute, SUM(numDataPoints) AS numDataPoints
FROM grouped_data
GROUP BY minute
ORDER BY 1 desc, 2 desc
```

집계를 사전 계산하기 위한 예약된 쿼리

더 적은 데이터를 스캔하여 대시보드를 더 빠르게 로드하고 비용을 절감하도록 최적화하려는 경우 예약된 쿼리를 사용하여 이러한 집계를 미리 계산할 수 있습니다. 의 Timestream에서 예약된 쿼리를 LiveAnalytics 사용하면 LiveAnalytics 테이블의 다른 Timestream에서 이러한 사전 계산을 구체화할 수 있으며, 나중에 대시보드에 사용할 수 있습니다.

예약된 쿼리를 생성하는 첫 번째 단계는 사전 계산하려는 쿼리를 식별하는 것입니다. 이전 대시보드는 us-east-1 리전에 대해 그려졌습니다. 그러나 다른 사용자는 us-west-2 또는 eu-west-1과 같은 다른 리전에 대해 동일한 집계를 원할 수 있습니다. 이러한 각 쿼리에 대해 예약된 쿼리를 생성하지 않으려면 각 리전의 집계를 미리 계산하고 LiveAnalytics 테이블의 다른 Timestream에서 리전별 집계를 구체화할 수 있습니다.

아래 쿼리는 해당 사전 계산의 예를 제공합니다. 보시다시피 원시 데이터의 쿼리에 사용되는 일반적인 테이블 표현식 grouped_data와 유사합니다. 단, 1) 하나의 쿼리를 사용하여 모든 리전에 대해 사전 계산할 수 있도록 리전 예측을 사용하지 않으며, 2) 아래 세부 정보에서 설명하는 특수 파라미터 @scheduled_runtime이 포함된 파라미터화된 시간 예측을 사용합니다.

```
SELECT region, bin(time, 1m) as minute,
    SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m
GROUP BY bin(time, 1m), region
```

다음 사양을 사용하여 이전 쿼리를 예약된 쿼리로 변환할 수 있습니다. 예약된 쿼리에는 사용자 친화적인 니모닉인 이름이 할당됩니다. 그런 다음 cron 표현식인 QueryString ScheduleConfiguration, 를 포함합니다. 쿼리 결과를 Timestream for 의 대상 테이블에 매핑 TargetConfiguration 하는 를 지정합니다 LiveAnalytics. 마지막으로 쿼리의 개별 실행에 대한 알림이 전송되는 , 쿼리에 오류가 발생할 경우를 대비 ErrorReportConfiguration 하여 보고서가 작성 NotificationConfiguration되는 , 예약된 쿼리에 대한 작업을 수행하는 데 사용되는 역할 ScheduledQueryExecutionRoleArn인 와 같은 여러 다른 구성을 지정합니다.

```
{
    "Name": "MultiPT5mPerMinutePerRegionMeasureCount",
    "QueryString": "SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name
 = 'metrics' THEN 20 ELSE 5 END) as numDataPoints FROM raw_data.devops WHERE time
 BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m GROUP BY bin(time, 1m),
 region",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/5 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
```

```
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_minute_aggs_pt5m",
            "TimeColumn": "minute",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

이 예제에서 ScheduleExpression cron(0/5 * * * ? *)은 쿼리가 매일 매시간 5, 10, 15, ..분에 5분마다 한 번씩 실행됨을 나타냅니다. 이 쿼리의 특정 인스턴스가 트리거될 때 이러한 타임스탬프는 쿼리에 사용되는 @scheduled_runtime 파라미터로 변환됩니다. 예를 들어 2021-12-01 00:00:00에 실행 중인 이 예약된 쿼리의 인스턴스를 고려해 보세요. 이 경우 쿼리를 호출할 때 @scheduled_runtime 파라미터가 타임스탬프 2021-12-01 00:00:00으로 초기화됩니다. 따라서 이 특정 인스턴스는 타임스탬프 2021-12-01 00:00:00에 실행되며 2021-11-30 23:50:00부터 2021-12-0100:01:00까지의 시간 범위에서 분당 집계를 계산합니다. 마찬가지로 이 쿼리의 다음 인스턴스는 타임스탬프 2021-12-01 00:05:00에 트리거되며, 이 경우 쿼리는 2021-11-30 23:55:00부터 2021-12-01 00:06:00까지의 시간 범위에서 분당 집계를 계산합니다. 따라서 @scheduled_runtime 파라미터는 쿼리에 대한 호출 시간을 사용하여 구성된 시간 범위에 대한 집계를 사전 계산하는 예약된 쿼리를 제공합니다.

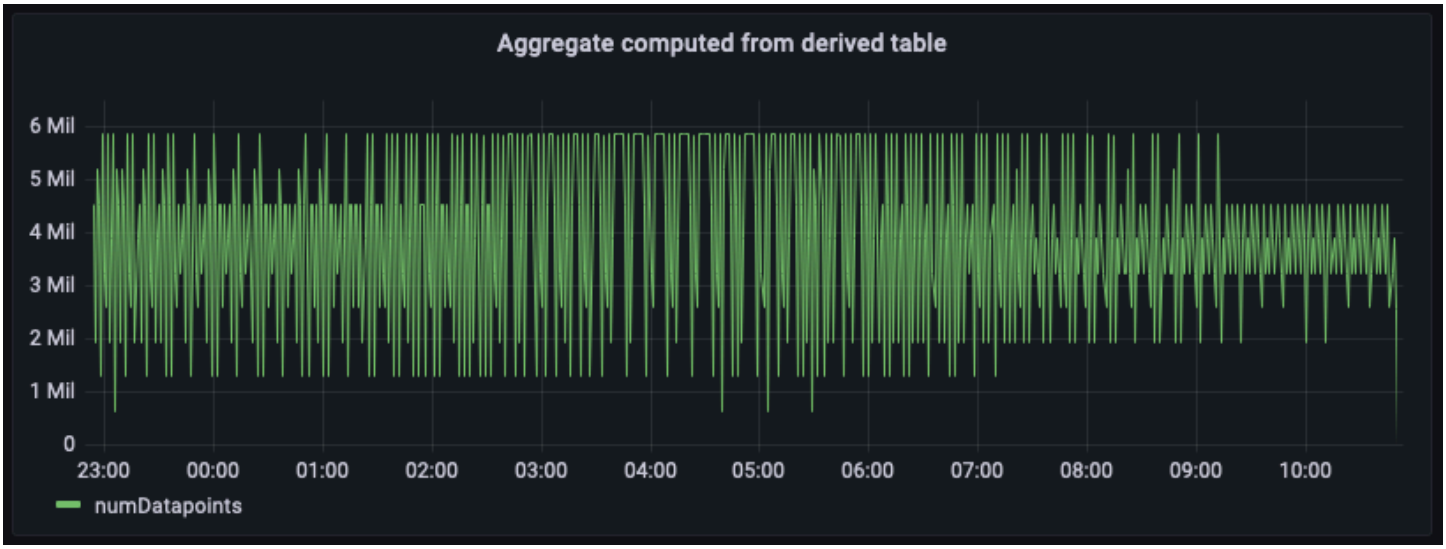
쿼리의 후속 인스턴스 두 개가 시간 범위에서 겹칩니다. 이는 요구 사항에 따라 제어할 수 있는 사항입니다. 이 경우 이러한 중복을 통해 이 쿼리는 이 예제에서 최대 5분까지 도착이 약간 지연된 모든 데이터를 기반으로 집계를 업데이트할 수 있습니다. 구체화된 쿼리의 정확성을 보장하기 위해 Timestream for 는 2021-12-01 00:05:00의 쿼리가 00:00:00의 쿼리가 완료된 후에만 2021-12-01 00:05:00의 쿼리가 수행되도록 LiveAnalytics 하고, 이후 쿼리의 결과는 새 값이 생성되는 경우 를 사용하여 이전에 구체화된 집계를 업데이트할 수 있습니다. 예를 들어 타임스탬프 2021-11-30 23:59:00의 일부 데이터가 2021-12-01 00:00:00에 대한 쿼리가 실행된 후 2021-12-01 00:05:00에 대한 쿼리 이전에 도착한 경우 2021-12-01 00:05:00의 실행은 2021-11-30 23:59:00분에 대한 집계를 다시 계산하고 이로 인해 이전 집계가 새로 계산된 값으로 업데이트됩니다. 예약된 쿼리의 이러한 의미 체계를 사용하여 사전 계산을 얼마나 빠르게 업데이트하는지와 도착이 지연된 일부 데이터를 어떻게 정상적으로 처리할 수 있는지 간에 균형을 맞출 수 있습니다. 데이터의 신선도로 이 새로 고침 주기를 상쇄하는 방법과 훨씬 더 지연된 상태로 도착하는 데이터의 집계 업데이트를 처리하는 방법 또는 예약된 계산의 소스에 집계를 다시 계산해야 하는 업데이트된 값이 있는지 여부에 대한 추가 고려 사항은 아래에서 설명합니다.

예약된 모든 계산에는 Timestream이 예약된 구성의 모든 실행에 대한 알림을 LiveAnalytics 보내는 알림 구성이 있습니다. 각 호출에 대한 알림을 수신하도록 에 대한 SNS 주제를 구성할 수 있습니다. 특정 인스턴스의 성공 또는 실패 상태 외에도 이 계산이 실행되는 데 걸린 시간, 스캔한 계산 바이트 수, 계산이 대상 테이블에 기록한 바이트 수와 같은 여러 통계가 있습니다. 이러한 통계를 사용하여 쿼리를 추가로 조정하거나, 구성을 예약하거나, 예약된 쿼리에 대한 지출을 추적할 수 있습니다. 주목할 만한 한 가지 측면은 인스턴스의 실행 시간입니다. 이 예제에서는 예약된 계산이 5분마다 실행되도록 구성됩니다. 실행 시간에 따라 사전 계산을 사용할 수 있는 지연 시간이 결정되며, 이는 대시보드에서 사전 계산 데이터를 사용할 때 대시보드의 지연 시간도 정의합니다. 또한 이 지연이 새로 고침 간격보다 일관되게 더 높은 경우, 예를 들어 실행 시간이 5분마다 새로 고침하도록 구성된 계산에 대해 5분 이상인 경우 대시보드에서 추가 지연을 방지하기 위해 계산을 더 빠르게 실행하도록 조정하는 것이 중요합니다.

파생 테이블에서 집계

이제 예약된 쿼리를 설정하고 예약된 계산의 대상 구성에 지정된 LiveAnalytics 테이블에 대해 집계가 미리 계산되고 다른 Timestream으로 구체화되었으므로 해당 테이블의 데이터를 사용하여 대시보드에 전원을 공급하는 SQL 쿼리를 작성할 수 있습니다. 다음은 구체화된 사전 집계를 사용하여 us-east-1에 대한 분당 데이터 포인트 수 집계를 생성하는 쿼리와 동일합니다.

```
SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
FROM "derived"."per_minute_aggs_pt5m"
WHERE time BETWEEN from_milliseconds(1636699996445) AND
 from_milliseconds(1636743196445)
    AND region = 'us-east-1'
GROUP BY bin(time, 1m)
ORDER BY 1 desc
```



이전 그림은 집계 테이블에서 계산된 집계를 표시합니다. 이 패널을 원시 소스 데이터에서 계산된 패널과 비교하면 정확히 일치하지만 이러한 집계는 예약된 계산에 대해 구성한 새로 고침 간격과 실행 시간에 따라 몇 분 정도 지연됩니다.

사전 계산된 데이터에 대한 이 쿼리는 원시 소스 데이터를 통해 계산된 집계에 비해 몇 배 더 적은 데이터를 스캔합니다. 집계의 세분성에 따라 이러한 감소로 인해 비용 및 쿼리 지연 시간이 100X 더 낮아질 수 있습니다. 이 예약된 계산을 실행하는 데 비용이 발생합니다. 그러나 이러한 대시보드를 새로 고치는 빈도와 이러한 대시보드를 로드하는 동시 사용자 수에 따라 이러한 사전 계산을 사용하여 전체 비용을 크게 절감할 수 있습니다. 또한 대시보드의 로드 시간이 10-100X 빨라집니다.

소스 테이블과 파생 테이블의 결합 집계

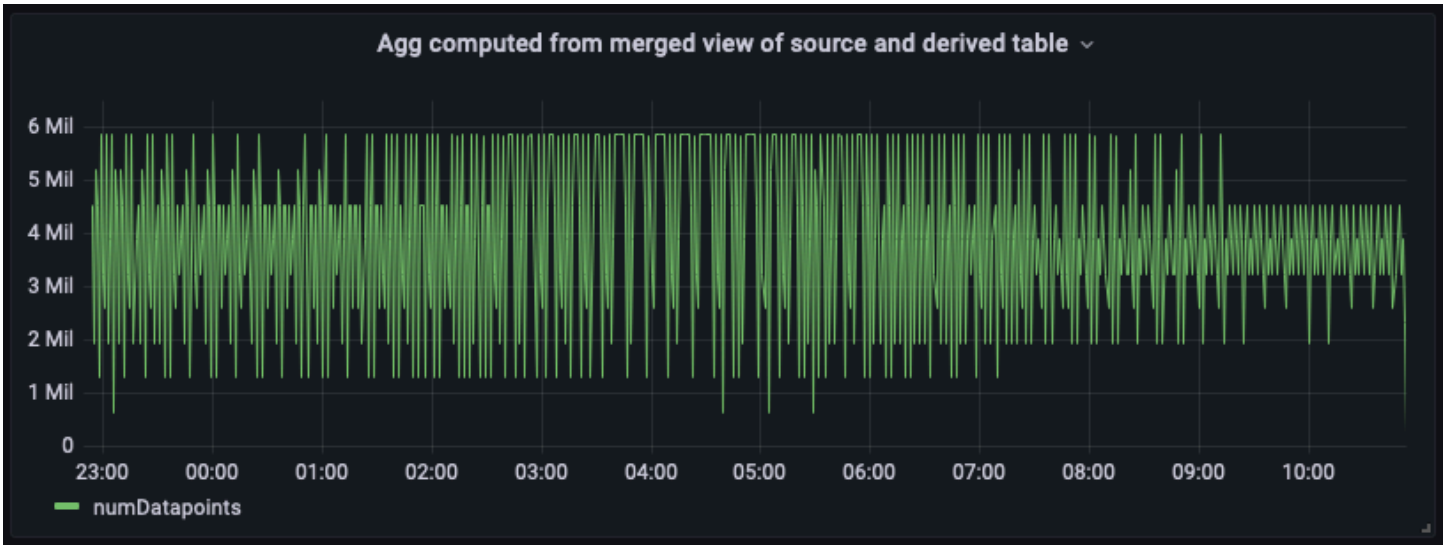
파생 테이블을 사용하여 생성된 대시보드에는 지연이 있을 수 있습니다. 애플리케이션 시나리오에서 대시보드에 최신 데이터가 있어야 하는 경우 LiveAnalytics의 SQL 지원을 위해 Timestream의 성능과 유연성을 사용하여 소스 테이블의 최신 데이터를 파생 테이블의 기록 집계와 결합하여 병합된 뷰를 구성할 수 있습니다. 이 병합된 보기는 소스 SQL 및 파생 테이블의 및 겹치지 않는 시간 범위의 유니온 의미 체계를 사용합니다. 아래 예제에서는 'derived'.'per_minute_aggs_pt5m' 파생 테이블을 사용합니다. 파생된 테이블에 대한 예약된 계산이 5분마다 한 번씩 새로 고쳐지기 때문에(스케줄 표현식 사양에 따라) 아래의 이 쿼리는 소스 테이블의 최근 15분 데이터를 사용합니다. 및 파생된 테이블에서 15분 이상 지난 모든 데이터는 결과를 결합하여 두 월드 모두의 장점을 갖춘 병합된 뷰를 생성합니다. 파생된 테이블에서 미리 계산된 이전 집계를 읽고 소스 테이블에서 집계의 신선도를 확인하여 실시간 분석 사용 사례를 강화함으로써 경제성과 지연 시간을 줄입니다.

이 유니온 접근 방식은 파생 테이블만 쿼리하는 것에 비해 쿼리 지연 시간이 약간 더 높고, 가장 최근 시간 간격을 채우기 위해 원시 데이터를 실시간으로 집계하기 때문에 스캔된 데이터도 약간 더 높다는 점에 유의하세요. 그러나 이 병합된 뷰는 특히 며칠 또는 몇 주간의 데이터를 렌더링하는 대시보드의 경

우 소스 테이블에서 즉시 집계하는 것보다 훨씬 빠르고 저렴합니다. 이 예제의 시간 범위를 조정하여 애플리케이션의 새로 고침 요구 사항과 지연 허용 오차를 맞출 수 있습니다.

```
WITH aggregated_source_data AS (
    SELECT bin(time, 1m) as minute, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE
 5 END) as numDatapoints
    FROM "raw_data"."devops"
    WHERE time BETWEEN bin(from_milliseconds(1636743196439), 1m) - 15m AND
 from_milliseconds(1636743196439)
        AND region = 'us-east-1'
    GROUP BY bin(time, 1m)
), aggregated_derived_data AS (
    SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
    FROM "derived"."per_minute_aggs_pt5m"
    WHERE time BETWEEN from_milliseconds(1636699996439) AND
 bin(from_milliseconds(1636743196439), 1m) - 15m
        AND region = 'us-east-1'
    GROUP BY bin(time, 1m)
)
SELECT minute, numDatapoints
FROM (
    (
    SELECT *
    FROM aggregated_derived_data
    )
    UNION
    (
    SELECT *
    FROM aggregated_source_data
    )
)
ORDER BY 1 desc
```

다음은 이 통합 병합 보기가 있는 대시보드 패널입니다. 보시다시피 대시보드는 가장 오른쪽 팁에 집계가 가장 많 up-to-date다는 점을 제외하면 파생 테이블에서 계산된 보기와 거의 동일하게 보입니다.



자주 새로 고쳐지는 예약된 계산에서 집계

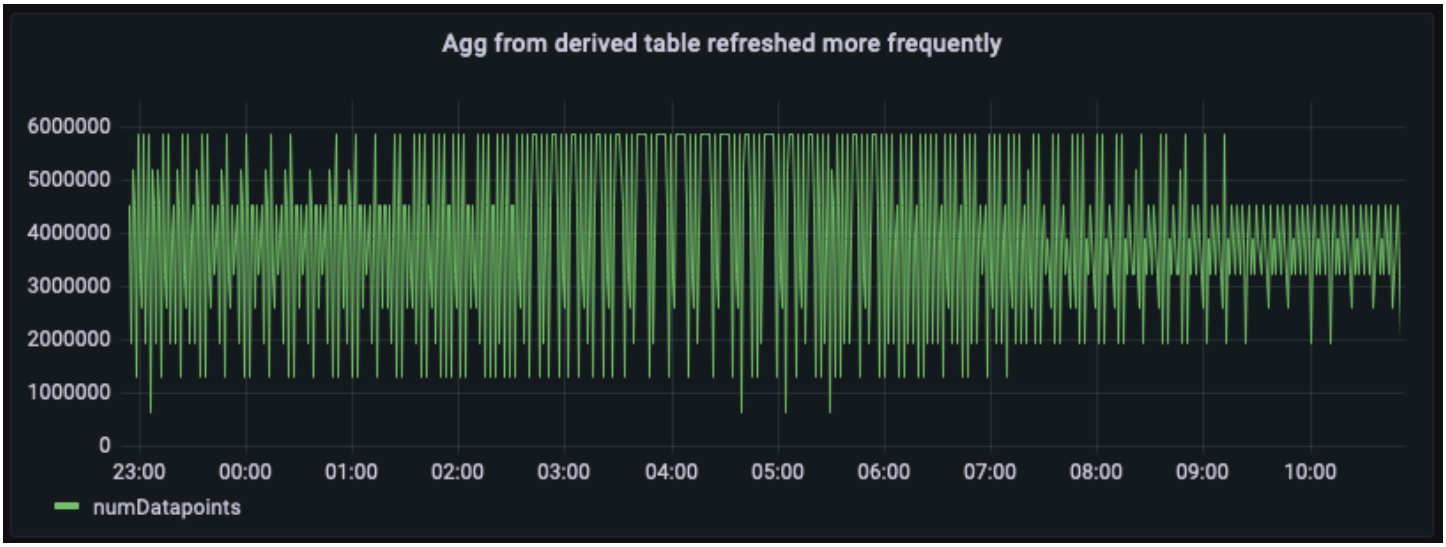
대시보드가 로드되는 빈도와 대시보드에 원하는 지연 시간에 따라 대시보드에서 더 새로운 결과를 얻기 위한 또 다른 접근 방식이 있습니다. 예약된 계산으로 집계를 더 자주 새로 고칩니다. 예를 들어, 다음은 1분에 한 번 새로 고쳐지는 경우를 제외하고 동일한 예약된 계산의 구성입니다(스케줄 Express cron(0/1 * * * * ? *)). 이 설정을 사용하면 연산에서 5분마다 1회의 새로 고침 일정을 지정한 시나리오에 비해 파생 테이블 per_minute_aggs_pt1m의 최신 집계가 훨씬 더 많아집니다.

```
{
    "Name": "MultiPT1mPerMinutePerRegionMeasureCount",
    "QueryString": "SELECT region, bin(time, 1m) as minute, SUM(CASE WHEN measure_name
 = 'metrics' THEN 20 ELSE 5 END) as numDataPoints FROM raw_data.devops WHERE time
 BETWEEN @scheduled_runtime - 10m AND @scheduled_runtime + 1m GROUP BY bin(time, 1m),
 region",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/1 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_minute_aggs_pt1m",
            "TimeColumn": "minute",
            "DimensionMappings": [
```

```
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

```
SELECT bin(time, 1m) as minute, SUM(numDataPoints) as numDatapoints
FROM "derived"."per_minute_aggs_pt1m"
WHERE time BETWEEN from_milliseconds(1636699996446) AND
 from_milliseconds(1636743196446)
    AND region = 'us-east-1'
GROUP BY bin(time, 1m), region
ORDER BY 1 desc
```

파생 테이블에는 최신 집계가 있으므로 이제 이전 쿼리 및 아래 대시보드 스냅샷에서 볼 수 있듯이 파생 테이블 per_minute_aggs_pt1m을 직접 쿼리하여 더 새로운 집계를 가져올 수 있습니다.



예약된 계산을 더 빠른 일정(예: 5분 대비 1분)으로 새로 고치면 예약된 계산의 유지 관리 비용이 증가합니다. 모든 계산 실행에 대한 알림 메시지는 스캔된 데이터의 양과 파생 테이블에 기록된 데이터의 양에 대한 통계를 제공합니다. 마찬가지로 병합된 뷰를 사용하여 파생 테이블을 결합하면 병합된 뷰에서 비용을 쿼리하면 파생 테이블만 쿼리하는 것보다 대시보드 로드 지연 시간이 더 높아집니다. 따라서 선택하는 접근 방식은 대시보드가 새로 고쳐지는 빈도와 예약된 쿼리의 유지 관리 비용에 따라 달라집니다. 1분에 한 번 정도 대시보드를 새로 고치는 사용자가 수십 명인 경우 파생 테이블을 더 자주 새로 고치면 전반적으로 비용이 절감될 수 있습니다.

각 디바이스의 마지막 포인트

애플리케이션에서 디바이스에서 내보낸 마지막 측정값을 읽어야 할 수 있습니다. 지정된 이전의 디바이스에 대한 마지막 측정치를 얻기 위한 일반적인 사용 사례가 있을 수 있습니다date/time or the first measurement for a device after a given date/time. 수백만 개의 디바이스와 수년간의 데이터가 있는 경우 이 검색을 수행하려면 대량의 데이터를 스캔해야 할 수 있습니다.

아래에는 예약된 쿼리를 사용하여 디바이스에서 내보내는 마지막 포인트에 대한 검색을 최적화하는 방법의 예가 나와 있습니다. 애플리케이션에 필요한 경우에도 동일한 패턴을 사용하여 첫 번째 포인트 쿼리를 최적화할 수 있습니다.

주제

- 소스 테이블에서 계산됨
- 일일 세분화로 사전 계산할 파생 테이블
- 파생 테이블에서 계산됨
- 소스 및 파생 테이블에서 결합

소스 테이블에서 계산됨

다음은 특정 배포의 서비스(예: 지정된 리전, 셀, 사일로 및 availability_zone 내의 지정된 마이크로 서비스에 대한 서버)에서 내보내는 마지막 측정값을 찾기 위한 쿼리 예제입니다. 예제 애플리케이션에서 이 쿼리는 수백 개의 서버에 대한 마지막 측정값을 반환합니다. 또한 이 쿼리는 시간 제한값이 미결합되어 있으며 지정된 타임스탬프보다 오래된 데이터를 찾습니다.

> **Note**
>
> `max` 및 `max_by` 함수에 대한 자세한 내용은 섹션을 참조하세요집계 함수.

```
SELECT instance_name, MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
FROM "raw_data"."devops"
WHERE time < from_milliseconds(1636685271872)
    AND measure_name = 'events'
    AND region = 'us-east-1'
    AND cell = 'us-east-1-cell-10'
    AND silo = 'us-east-1-cell-10-silo-3'
    AND availability_zone = 'us-east-1-1'
    AND microservice_name = 'hercules'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
ORDER BY instance_name, time DESC
```

일일 세분화로 사전 계산할 파생 테이블

이전 사용 사례를 예약된 계산으로 변환할 수 있습니다. 애플리케이션 요구 사항이 여러 리전, 셀, 사일로, 가용 영역 및 마이크로서비스의 전체 플릿에 대해 이러한 값을 얻어야 하는 경우 하나의 일정 계산을 사용하여 전체 플릿의 값을 사전 계산할 수 있습니다. 이는 LiveAnalytics의 서버리스 예약 쿼리에 대한 Timestream의 성능으로, 이러한 쿼리를 애플리케이션의 크기 조정 요구 사항에 따라 확장할 수 있습니다.

아래는 지정된 날짜의 모든 서버에서 마지막 점을 사전 계산하는 쿼리입니다. 쿼리에는 시간 기호만 있으며 차원에 대한 기호는 없습니다. 시간 예측은 지정된 일정 표현식을 기반으로 계산이 트리거된 시점부터 쿼리를 전날로 제한합니다.

```
SELECT region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version,
    MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
FROM raw_data.devops
```

```
WHERE time BETWEEN bin(@scheduled_runtime, 1d) - 1d AND bin(@scheduled_runtime, 1d)
    AND measure_name = 'events'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
```

아래는 매일 01:00시에 쿼리를 실행UTC하여 지난 날의 집계를 계산하는 이전 쿼리를 사용하여 예약된 계산에 대한 구성입니다. 일정 표현식 cron(0 1 * * ? *)은 이 동작을 제어하고 하루가 종료된 후 1시간을 실행하여 최대 하루 늦게 도착하는 모든 데이터를 고려합니다.

```
{
    "Name": "PT1DPerInstanceLastpoint",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 instance_name, process_name, jdk_version, MAX(time) AS time, MAX_BY(gc_pause, time)
 AS last_measure FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1d) -
 1d AND bin(@scheduled_runtime, 1d) AND measure_name = 'events' GROUP BY region, cell,
 silo, availability_zone, microservice_name, instance_name, process_name, jdk_version",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0 1 * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "per_timeseries_lastpoint_pt1d",
            "TimeColumn": "time",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
```

```
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "last_measure",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "last_measure",
                        "MeasureValueType": "DOUBLE"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

파생 테이블에서 계산됨

이전 구성을 사용하여 파생 테이블을 정의하고 예약된 쿼리의 인스턴스 중 하나 이상이 파생 테이블로 데이터를 구체화한 경우 이제 파생 테이블을 쿼리하여 최신 측정값을 얻을 수 있습니다. 다음은 파생 테이블의 예제 쿼리입니다.

```
SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
FROM "derived"."per_timeseries_lastpoint_pt1d"
WHERE time < from_milliseconds(1636746715649)
    AND measure_name = 'last_measure'
    AND region = 'us-east-1'
    AND cell = 'us-east-1-cell-10'
    AND silo = 'us-east-1-cell-10-silo-3'
    AND availability_zone = 'us-east-1-1'
    AND microservice_name = 'hercules'
GROUP BY region, cell, silo, availability_zone, microservice_name,
    instance_name, process_name, jdk_version
ORDER BY instance_name, time DESC
```

소스 및 파생 테이블에서 결합

이전 예제와 마찬가지로 파생 테이블의 모든 데이터는 가장 최근의 쓰기를 갖지 않습니다. 따라서 이전과 유사한 패턴을 다시 사용하여 이전 데이터에 대해 파생 테이블의 데이터를 병합하고 나머지 팁에 대해 소스 데이터를 사용할 수 있습니다. 다음은 유사한 UNION 접근 방식을 사용하는 이러한 쿼리의 예입니다. 애플리케이션 요구 사항은 특정 기간 전에 최신 측정값을 찾는 것이고 이 시작 시간은 과거일 수 있으므로 이 쿼리를 작성하는 방법은 제공된 시간을 사용하고, 지정된 시간으로부터 최대 하루 동안 소스 데이터를 사용한 다음, 이전 데이터에 파생 테이블을 사용하는 것입니다. 아래 쿼리 예제에서 볼 수 있듯이 소스 데이터의 예측 시간은 경계가 지정됩니다. 이렇게 하면 데이터 볼륨이 상당히 높은 소스 테이블에서 효율적으로 처리할 수 있으며, 그런 다음 비결합 시간 예측이 파생 테이블에 표시됩니다.

```
WITH last_point_derived AS (
    SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
    FROM "derived"."per_timeseries_lastpoint_pt1d"
    WHERE time < from_milliseconds(1636746715649)
        AND measure_name = 'last_measure'
        AND region = 'us-east-1'
        AND cell = 'us-east-1-cell-10'
        AND silo = 'us-east-1-cell-10-silo-3'
        AND availability_zone = 'us-east-1-1'
        AND microservice_name = 'hercules'
```

```
    GROUP BY region, cell, silo, availability_zone, microservice_name,
        instance_name, process_name, jdk_version
), last_point_source AS (
    SELECT instance_name, MAX(time) AS time, MAX_BY(gc_pause, time) AS last_measure
    FROM "raw_data"."devops"
    WHERE time < from_milliseconds(1636746715649) AND time >
 from_milliseconds(1636746715649) - 26h
        AND measure_name = 'events'
        AND region = 'us-east-1'
        AND cell = 'us-east-1-cell-10'
        AND silo = 'us-east-1-cell-10-silo-3'
        AND availability_zone = 'us-east-1-1'
        AND microservice_name = 'hercules'
    GROUP BY region, cell, silo, availability_zone, microservice_name,
        instance_name, process_name, jdk_version
)
SELECT instance_name, MAX(time) AS time, MAX_BY(last_measure, time) AS last_measure
FROM (
    SELECT * FROM last_point_derived
    UNION
    SELECT * FROM last_point_source
)
GROUP BY instance_name
ORDER BY instance_name, time DESC
```

앞의 그림은 파생 테이블을 구성하는 방법을 보여줍니다. 수년 간의 데이터가 있는 경우 더 많은 수준의 집계를 사용할 수 있습니다. 예를 들어 일일 집계 외에 월별 집계를 사용할 수 있으며, 일별 집계 전에 시간별 집계를 사용할 수 있습니다. 따라서 가장 최근의 를 병합하여 지난 1시간을 채우고, 매시간은 마지막 날을 채우고, 일별은 지난 달을 채우고, 매월은 이전 을 채울 수 있습니다. 설정한 수준 수와 새로 고침 일정은 이러한 쿼리가 문제가 되는 빈도와 이러한 쿼리를 동시에 발행하는 사용자 수에 대한 요구 사항에 따라 달라집니다.

고유한 차원 값

고유한 차원 값을 변수로 사용하여 특정 데이터 조각에 해당하는 지표를 드릴다운하려는 대시보드가 있는 사용 사례가 있을 수 있습니다. 아래 스냅샷은 대시보드가 리전, 셀, 사일로, 마이크로서비스 및 availability_zone과 같은 여러 차원의 고유한 값을 미리 채우는 예제입니다. 여기에서는 예약된 쿼리를 사용하여 추적 중인 지표에서 이러한 변수의 고유한 값을 계산하는 속도를 크게 높일 수 있는 방법의 예를 보여줍니다.

주제

- 원시 데이터

- 고유 차원 값 사전 계산
- 파생 테이블에서 변수 계산

원시 데이터

SELECT DISTINCT 를 사용하여 데이터에서 보이는 고유한 값을 계산할 수 있습니다. 예를 들어 리전의 고유한 값을 얻으려면 이 양식의 쿼리를 사용할 수 있습니다.

```
SELECT DISTINCT region
FROM "raw_data"."devops"
WHERE time > ago(1h)
ORDER BY 1
```

수백만 개의 디바이스와 수십억 개의 시계열을 추적하고 있을 수 있습니다. 그러나 대부분의 경우 이러한 흥미로운 변수는 몇 개에서 수십 개의 값이 있는 낮은 카디널리티 차원에 사용됩니다. 원시 데이터 DISTINCT에서 계산하려면 대량의 데이터를 스캔해야 할 수 있습니다.

고유 차원 값 사전 계산

대시보드가 대화형이 되도록 이러한 변수를 빠르게 로드해야 합니다. 또한 이러한 변수는 모든 대시보드 로드에 대해 계산되는 경우가 많으므로 비용 효율성도 높아야 합니다. 예약된 쿼리를 사용하고 파생된 테이블에서 구체화하여 이러한 변수를 찾는 것을 최적화할 수 있습니다.

먼저 값을 계산할 때 예측에 사용할 DISTINCT 값 또는 열을 계산해야 하는 차원을 식별해야 합니다 DISTINCT.

이 예제에서는 대시보드가 차원 리전, 셀, 사일로, availability_zone 및 마이크로서비스에 대한 고유한 값을 채우고 있음을 확인할 수 있습니다. 따라서 아래 쿼리를 사용하여 이러한 고유한 값을 미리 계산할 수 있습니다.

```
SELECT region, cell, silo, availability_zone, microservice_name,
    min(@scheduled_runtime) AS time, COUNT(*) as numDataPoints
FROM raw_data.devops
WHERE time BETWEEN @scheduled_runtime - 15m AND @scheduled_runtime
GROUP BY region, cell, silo, availability_zone, microservice_name
```

여기에 유의해야 할 몇 가지 중요한 사항이 있습니다.

- 하나의 예약된 계산을 사용하여 여러 쿼리에 대한 값을 사전 계산할 수 있습니다. 예를 들어 이전 쿼리를 사용하여 5개의 다른 변수에 대한 값을 미리 계산합니다. 따라서 각 변수에 대해 하나씩 필요하

지 않습니다. 이 동일한 패턴을 사용하여 여러 패널에서 공유 컴퓨팅을 식별하여 유지 관리해야 하는 예약된 쿼리 수를 최적화할 수 있습니다.

- 차원의 고유 값은 본질적으로 시계열 데이터가 아닙니다. 따라서 @scheduled_runtime을 사용하여 시계열로 변환할 수 있습니다. 이 데이터를 @scheduled_runtime 파라미터와 연결하면 지정된 시점에 어떤 고유 값이 나타나는지 추적하여 시계열 데이터를 생성할 수도 있습니다.
- 이전 예에서는 지표 값이 추적되는 것을 볼 수 있습니다. 이 예제에서는 COUNT(*)를 사용합니다. 대시보드에 대해 추적하려는 경우 다른 의미 있는 집계를 계산할 수 있습니다.

다음은 이전 쿼리를 사용하여 예약된 계산에 대한 구성입니다. 이 예제에서는 일정 표현식 cron(0/15 * * * ?을 사용하여 15분마다 새로 고치도록 구성되어 있습니다. *).

```
{
    "Name": "PT15mHighCardPerUniqueDimensions",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 min(@scheduled_runtime) AS time, COUNT(*) as numDataPoints FROM raw_data.devops WHERE
 time BETWEEN @scheduled_runtime - 15m AND @scheduled_runtime GROUP BY region, cell,
 silo, availability_zone, microservice_name",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/15 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "hc_unique_dimensions_pt15m",
            "TimeColumn": "time",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
```

```
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "count_multi",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

파생 테이블에서 변수 계산

예약된 계산이 파생 테이블 hc_unique_dimensions_pt15m의 고유 값을 미리 구체화하면 파생 테이블을 사용하여 차원의 고유 값을 효율적으로 계산할 수 있습니다. 다음은 고유 값을 계산하는 방법과 이러한 고유 값 쿼리에서 다른 변수를 예측 변수로 사용하는 방법에 대한 쿼리 예제입니다.

리전

```
SELECT DISTINCT region
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
```

```
ORDER BY 1
```

셀

```
SELECT DISTINCT cell
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
  AND region = '${region}'
ORDER BY 1
```

사일로

```
SELECT DISTINCT silo
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}'
ORDER BY 1
```

마이크로서비스

```
SELECT DISTINCT microservice_name
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}'
ORDER BY 1
```

가용 영역

```
SELECT DISTINCT availability_zone
FROM "derived"."hc_unique_dimensions_pt15m"
WHERE time > ago(1h)
   AND region = '${region}' AND cell = '${cell}' AND silo = '${silo}'
ORDER BY 1
```

지연 도착 데이터 처리

데이터가 상당히 늦게 도착할 수 있는 시나리오가 있을 수 있습니다. 예를 들어, 데이터가 Timestream에 수집된 시간이 수집된 행과 연결된 타임스탬프에 비해 상당히 지연되는 경우가 LiveAnalytics 있습니다. 이전 예제에서는 @scheduled_runtime 파라미터에 정의된 시간 범위를 사용하여 지연된 데이터를 설명하는 방법을 살펴보았습니다. 그러나 데이터가 몇 시간 또는 며칠 지연될 수 있는 사용 사례가 있는 경우 파생 테이블의 사전 계산이 이러한 지연 도착 데이터를 반영하도록 적절하게 업데이트되도

록 다른 패턴이 필요할 수 있습니다. 지연 도착 데이터에 대한 일반 정보는 섹션을 참조하세요데이터 쓰기(삽입 및 업서트).

다음에서는 이 지연 도착 데이터를 처리하는 두 가지 방법을 볼 수 있습니다.

- 데이터 도착에 예측 가능한 지연이 있는 경우 다른 '캐치업' 예약 계산을 사용하여 도착 지연 데이터에 대한 집계를 업데이트할 수 있습니다.
- 예측할 수 없는 지연 또는 간헐적인 지연 도착 데이터가 있는 경우 수동 실행을 사용하여 파생 테이블을 업데이트할 수 있습니다.

이 논의에서는 데이터 도착 지연 시나리오를 다룹니다. 그러나 소스 테이블의 데이터를 수정하고 파생 테이블의 집계를 업데이트하려는 데이터 수정에도 동일한 원칙이 적용됩니다.

주제

- 예약된 캐치업 쿼리
- 예측할 수 없는 지연 도착 데이터에 대한 수동 실행

예약된 캐치업 쿼리

정시에 도착한 데이터 집계 쿼리

다음은 데이터 도착에 예측 가능한 지연이 있는 경우 자동화된 방법을 사용하여 집계를 업데이트하는 방법을 보여주는 패턴입니다. 아래 실시간 데이터에 대한 예약된 계산의 이전 예제 중 하나를 고려해 보세요. 이 예약된 계산은 30분마다 파생 테이블을 새로 고치고 최대 1시간 지연된 데이터를 이미 설명합니다.

```
{
    "Name": "MultiPT30mPerHrPerTimeseriesDPCount",
    "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
 instance_type, os_version, instance_name, process_name, jdk_version, bin(time,
 1h) as hour, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as
 numDataPoints FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1h)
 - 1h AND @scheduled_runtime + 1h GROUP BY region, cell, silo, availability_zone,
 microservice_name, instance_type, os_version, instance_name, process_name,
 jdk_version, bin(time, 1h)",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
```

```
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "dp_per_timeseries_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "availability_zone",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "os_version",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
```

```
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

늦게 도착한 데이터에 대한 집계를 업데이트하는 캐치업 쿼리

이제 데이터가 약 12시간 지연될 수 있다고 가정해 보겠습니다. 다음은 동일한 쿼리의 변형입니다. 그러나 차이점은 예약된 계산이 트리거될 때와 비교하여 최대 12시간 지연된 데이터에 대한 집계를 계산한다는 것입니다. 예를 들어 아래 예제에서 쿼리를 볼 수 있습니다. 이 쿼리가 대상으로 하는 시간 범위는 쿼리가 트리거되기 전 2시간에서 14시간 사이입니다. 또한 일정 표현식 cron(0 0,12 * * ? *)을 발견하면 UTC 매일 00:00 UTC 및 12:00에 계산이 트리거됩니다. 따라서 2021-12-01 00:00:00에 쿼리가 트리거되면 쿼리는 2021-11-30 10:00:00에서 2021-11-30 22:00:00까지의 시간 범위에서 집계를 업데이트합니다. 예약된 쿼리는 LiveAnalytics의 쓰기에 대해 Timestream과 유사한 업서트 의미 체계를 사용합니다. 이 캐치업 쿼리는 창에 늦게 도착하는 데이터가 있거나 새 집계가 발견되면(예: 원래 예약된 계산이 트리거될 때 존재하지 않았던 이 집계에 새 그룹화가 표시됨) 집계 값을 더 새로운 값으로 업데이트합니다. 마찬가지로 다음 인스턴스가 2021-12-01 12:00:00에 트리거되면 해당 인스턴스는 2021-11-30 22:00:00~ 2021-12-0110:00:00 범위의 집계를 업데이트합니다.

```
      {
   "Name": "MultiPT12HPerHrPerTimeseriesDPCountCatchUp",
   "QueryString": "SELECT region, cell, silo, availability_zone, microservice_name,
instance_type, os_version, instance_name, process_name, jdk_version, bin(time, 1h)
as hour, SUM(CASE WHEN measure_name = 'metrics' THEN 20 ELSE 5 END) as numDataPoints
FROM raw_data.devops WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 14h AND
bin(@scheduled_runtime, 1h) - 2h GROUP BY region, cell, silo, availability_zone,
microservice_name, instance_type, os_version, instance_name, process_name,
jdk_version, bin(time, 1h)",
   "ScheduleConfiguration": {
       "ScheduleExpression": "cron(0 0,12 * * ? *)"
   },
   "NotificationConfiguration": {
       "SnsConfiguration": {
           "TopicArn": "******"
       }
   },
   "TargetConfiguration": {
       "TimestreamConfiguration": {
           "DatabaseName": "derived",
           "TableName": "dp_per_timeseries_per_hr",
           "TimeColumn": "hour",
           "DimensionMappings": [
               {
                   "Name": "region",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "cell",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "silo",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "availability_zone",
                   "DimensionValueType": "VARCHAR"
               },
               {
                   "Name": "microservice_name",
                   "DimensionValueType": "VARCHAR"
               },
               {
```

```
                    "Name": "instance_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "os_version",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "instance_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "process_name",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "jdk_version",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "numDataPoints",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "numDataPoints",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

이 이전 예제는 늦은 도착이 12시간으로 제한되어 있고 실시간 기간보다 늦게 도착하는 데이터에 대해 12시간마다 한 번씩 파생 테이블을 업데이트해도 된다고 가정한 예시입니다. 이 패턴을 조정하여 파생

테이블이 지연된 데이터를 더 빨리 반영하도록 1시간에 한 번 파생 테이블을 업데이트할 수 있습니다. 마찬가지로 예측 가능한 지연 도착 데이터를 처리하기 위해 시간 범위를 하루 또는 일주일 이상 등 12시간보다 오래도록 조정할 수 있습니다.

예측할 수 없는 지연 도착 데이터에 대한 수동 실행

예측할 수 없는 지연 데이터가 있거나 소스 데이터를 변경하고 사후에 일부 값을 업데이트한 인스턴스가 있을 수 있습니다. 이러한 모든 경우 예약된 쿼리를 수동으로 트리거하여 파생 테이블을 업데이트할 수 있습니다. 다음은 이를 달성하는 방법에 대한 예입니다.

파생 테이블 dp_per_timeseries_per_hr에 작성된 계산이 있는 사용 사례가 있다고 가정합니다. 테이블 데보프의 기본 데이터가 2021-11-30 23:00:00~2021-12-01 00:00:00의 시간 범위로 업데이트되었습니다. 이 파생된 테이블을 업데이트하는 데 사용할 수 있는 두 가지 예약 쿼리는 다중 PT30mPerHrPerTimeseriesDPCount과 다중 입니다PT12HPerHrPerTimeseriesDPCountCatchUp. Timestream에서 생성하는 각 예약된 계산 LiveAnalytics 에는 계산을 만들거나 목록 작업을 수행할 때 얻ARN는 고유한 가 있습니다. 를 계산ARN에 사용하고 쿼리에서 가져온 파라미터 @scheduled_runtime의 값을 사용하여 이 작업을 수행할 수 있습니다.

다중 PT30mPerHrPerTimeseriesDPCount에 대한 계산에 ARN arn_1이 있고 이 계산을 사용하여 파생 테이블을 업데이트하려는 것으로 가정합니다. 앞의 예약된 계산은 @scheduled_runtime 값 1시간 전과 1시간 후에 집계를 업데이트하므로 @scheduled_runtime 파라미터에 대해 2021-12-01 00:00:00 값을 사용하여 업데이트(2021-11-30 23:00:00 - 2021-12-01 00:00:00)의 시간 범위를 포함할 수 있습니다. 를 ExecuteScheduledQuery API 사용하여 이 계산ARN의 를 전달하고 시간 파라미터 값을 에폭 초() 단위로 전달UTC하여 이를 달성할 수 있습니다. 다음은 를 사용하는 예제AWSCLI이며 Timestream for 에서 SDKs 지원하는 를 사용하여 동일한 패턴을 따를 수 있습니다 LiveAnalytics.

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638316800 --profile profile --region us-east-1
```

이전 예제에서 프로필은 이 API 전화를 걸 수 있는 적절한 권한이 있는 AWS 프로필이며 1638316800은 2021-12-01 00:00:00의 에폭 초에 해당합니다. 이 수동 트리거는 시스템이 원하는 시간에 이 호출을 트리거했다고 가정할 때 자동 트리거와 거의 비슷하게 작동합니다.

더 긴 기간 동안 업데이트가 있었다면 기본 데이터가 2021-11-30 23:00:00~2021-12-01 11:00:00에 대해 업데이트되었다고 가정해 보겠습니다. 그러면 이전 쿼리를 여러 번 트리거하여 이 전체 시간 범위를 처리할 수 있습니다. 예를 들어 다음과 같이 6개의 다른 실행을 수행할 수 있습니다.

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638316800 --profile profile --region us-east-1
```

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638324000 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638331200 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638338400 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638345600 --profile profile --region us-east-1

aws timestream-query execute-scheduled-query --scheduled-query-arn arn_1 --invocation-
time 1638352800 --profile profile --region us-east-1
```

이전 6개의 명령은 2021-12-01 00:00:00, 2021-12-01 02:00:00, 2021-12-01 04:0:00, 2021-12-01 06:00:00, 2021-12-01 08:00:00 및 2021-12-01 10:00에 호출된 예약된 계산에 해당합니다.

또는 2021-12-01 13:00:00에 트리거된 계산 멀티PT12HPerHrPerTimeseriesDPCountCatchUp를 사용하여 한 번의 실행으로 전체 12시간 범위에 대한 집계를 업데이트할 수 있습니다. 예를 들어 arn_2가 해당 계산ARN의 인 경우 에서 다음 명령을 실행할 수 있습니다CLI.

```
aws timestream-query execute-scheduled-query --scheduled-query-arn arn_2 --invocation-
time 1638363600 --profile profile --region us-east-1
```

수동 트리거의 경우 자동 트리거 타임스탬프와 정렬할 필요가 없는 호출 시간 파라미터에 타임스탬프를 사용할 수 있습니다. 예를 들어 이전 예제에서는 자동 일정이 타임스탬프 2021-12-01 10:00:00, 12021-12-012:00:00 및 00:00:00에서만 트리거되더라도 2021-12-02 2021-12-0113:00:00에 계산을 트리거했습니다. 용 Timestream LiveAnalytics 은 수동 작업에 필요한 적절한 값으로 트리거할 수 있는 유연성을 제공합니다.

다음은 를 사용할 때 고려해야 할 몇 가지 중요한 사항입니다 ExecuteScheduledQuery API.

- 이러한 호출을 여러 번 트리거하는 경우 이러한 호출이 중복 시간 범위를 생성하지 않도록 해야 합니다. 예를 들어 이전 예제에서는 6회의 호출이 있었습니다. 각 호출은 2시간의 시간 범위를 포함하므로 호출 타임스탬프는 업데이트의 중복을 방지하기 위해 각각 2시간씩 분산되었습니다. 이렇게 하면 파생 테이블의 데이터가 일치하는 상태로 끝나고 소스 테이블의 집계가 됩니다. 중복되지 않는 시간 범위를 보장할 수 없는 경우 이러한 실행이 순차적으로 하나씩 트리거되는지 확인합니다. 시간 범위에서 겹치는 여러 실행을 동시에 트리거하는 경우 이러한 실행에 대한 오류 보고서에서 버전 충돌이 발생할 수 있는 트리거 실패를 볼 수 있습니다. 예약된 쿼리 호출에 의해 생성된 결과에는 호출이 트리거된 시기에 따라 버전이 할당됩니다. 따라서 최신 호출에서 생성된 행의 버전은 더 높습니

다. 상위 버전 레코드는 하위 버전 레코드를 덮어쓸 수 있습니다. 자동 트리거된 예약 쿼리의 경우 용 LiveAnalytics Timestream은 일정을 자동으로 관리하므로 후속 호출에 중복된 시간 범위가 있더라도 이러한 문제가 표시되지 않습니다.

- 앞서 언급한 대로 @scheduled_runtime의 타임스탬프 값으로 호출을 트리거할 수 있습니다. 따라서 원본 테이블에서 데이터가 업데이트된 범위에 해당하는 파생 테이블에서 적절한 시간 범위가 업데이트되도록 값을 적절하게 설정하는 것은 사용자의 책임입니다.
- DISABLED 상태에 있는 예약된 쿼리에 이러한 수동 트리거를 사용할 수도 있습니다. 이렇게 하면 자동화된 일정에 실행되지 않는 특수 쿼리가 DISABLED 상태에 있으므로 해당 쿼리를 정의할 수 있습니다. 대신 수동 트리거를 사용하여 데이터 수정 또는 지연 도착 사용 사례를 관리할 수 있습니다.

과거 사전 계산 채우기

예약된 계산을 생성할 때 Timestream for 는 사용자가 제공하는 일정 표현식에 따라 새로 고침이 관리되는 쿼리의 실행을 LiveAnalytics 관리합니다. 소스 테이블의 과거 데이터 양에 따라 과거 데이터에 해당하는 집계로 파생 테이블을 업데이트할 수 있습니다. 수동 트리거에 앞의 로직을 사용하여 과거 집계를 채울 수 있습니다.

예를 들어, per_timeseries_lastpoint_pt1d의 파생 테이블을 고려하면 예약된 계산은 지난 날 하루에 한 번 업데이트됩니다. 소스 테이블에 1년의 데이터가 있는 경우 이 예약된 계산ARN에 를 사용하고 최대 1년 동안 매일 수동으로 트리거하여 파생 테이블에 모든 기록 쿼리가 채워지도록 할 수 있습니다. 수동 트리거에 대한 모든 주의 사항이 여기에 적용됩니다. 또한 과거 수집이 파생 테이블의 마그네틱 스토어에 쓸 수 있도록 파생 테이블이 설정된 경우 마그네틱 스토어에 쓸 수 있는 모범 사례와 제한에 유의하세요. https://docs.aws.amazon.com/timestream/latest/developerguide/ts-limits.html

예약된 쿼리 예제

이 섹션에는 LiveAnalytics플릿 전체 통계를 시각화할 때 디바이스 플릿을 효과적으로 모니터링할 때의 예약된 쿼리에 Timestream을 사용하여 비용 및 대시보드 로드 시간을 최적화하는 방법의 예가 포함되어 있습니다. 에 대한 Timestream의 예약된 쿼리를 LiveAnalytics 사용하면 에 대한 Timestream의 전체 SQL 표면적을 사용하여 쿼리를 표현할 수 있습니다 LiveAnalytics. 쿼리에는 하나 이상의 소스 테이블이 포함되며, LiveAnalytics의 SQL 언어에 대해 Timestream에서 허용하는 집계 또는 기타 쿼리를 수행한 다음 쿼리 결과를 의 Timestream의 다른 대상 테이블에 저장할 수 있습니다 LiveAnalytics.

이 섹션에서는 예약된 쿼리의 대상 테이블을 파생 테이블 로 참조합니다.

예를 들어 여러 배포(예: 리전, 셀, 사일로), 여러 마이크로서비스에 배포된 대규모 서버 플릿을 모니터링하고 용 Timestream을 사용하여 플릿 전체 통계를 추적하는 DevOps 애플리케이션을 사용합니다 LiveAnalytics. 사용할 예제 스키마는 예약된 쿼리 샘플 스키마 에 설명되어 있습니다.

다음 시나리오가 설명됩니다.

- 대시보드를 변환하고 수집한 원시 데이터의 집계된 통계를 Timestream으로 예약된 쿼리 LiveAnalytics 로 도표화한 다음 미리 계산된 집계를 사용하여 집계 통계를 보여주는 새 대시보드를 생성하는 방법.
- 예약된 쿼리를 결합하여 집계 보기와 원시 세분화된 데이터를 가져와 세부 정보로 드릴다운하는 방법. 이렇게 하면 예약된 쿼리를 사용하여 일반적인 플릿 전체 작업을 최적화하면서 원시 데이터를 저장하고 분석할 수 있습니다.
- 여러 대시보드에서 사용되는 집계를 찾고 동일한 예약된 쿼리가 동일한 대시보드 또는 여러 대시보드의 여러 패널을 채우도록 하여 예약된 쿼리를 사용하여 비용을 최적화하는 방법.

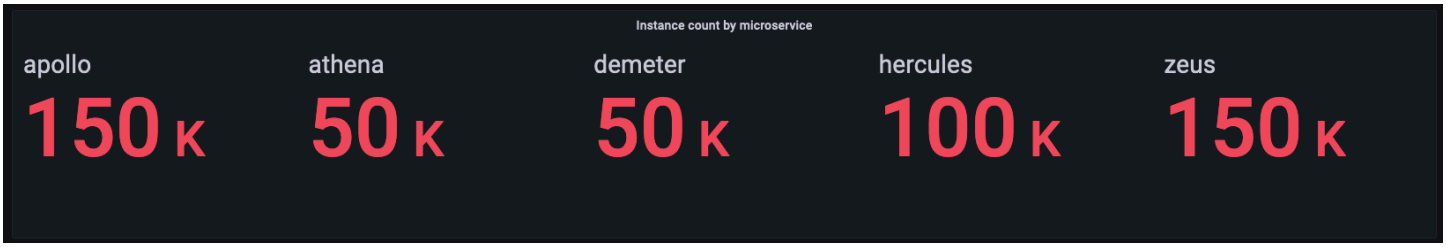
주제

- 집계 대시보드를 예약된 쿼리로 변환
- 드릴다운에 예약된 쿼리 및 원시 데이터 사용
- 대시보드 간에 예약된 쿼리를 공유하여 비용 최적화
- 기본 테이블의 쿼리와 예약된 쿼리 결과의 쿼리 비교

집계 대시보드를 예약된 쿼리로 변환

플릿의 호스트 수와 같은 플릿 전체 통계를 5개의 마이크로서비스와 서비스가 배포된 6개의 리전별로 계산한다고 가정해 보겠습니다. 아래 스냅샷에서 지표를 내보내는 서버가 500K개이고, 더 큰 리전(예: us-east-1) 중 일부는 >200,000개의 서버를 가지고 있습니다.

수백 기가바이트 이상의 데이터를 통해 고유한 인스턴스 이름을 계산하는 이러한 집계를 계산하면 데이터 스캔 비용 외에도 수십 초의 쿼리 지연 시간이 발생할 수 있습니다.



원본 대시보드 쿼리

대시보드 패널에 표시된 집계는 아래 쿼리를 사용하여 원시 데이터에서 계산됩니다. 쿼리는 고유 수 및 여러 집계 함수와 같은 여러 SQL 구성 요소를 사용합니다.

```
SELECT CASE WHEN microservice_name = 'apollo' THEN num_instances ELSE NULL END AS
 apollo,
    CASE WHEN microservice_name = 'athena' THEN num_instances ELSE NULL END AS athena,
    CASE WHEN microservice_name = 'demeter' THEN num_instances ELSE NULL END AS
 demeter,
    CASE WHEN microservice_name = 'hercules' THEN num_instances ELSE NULL END AS
 hercules,
    CASE WHEN microservice_name = 'zeus' THEN num_instances ELSE NULL END AS zeus
FROM (
    SELECT microservice_name, SUM(num_instances) AS num_instances
    FROM (
        SELECT microservice_name, COUNT(DISTINCT instance_name) as num_instances
        FROM "raw_data"."devops"
        WHERE time BETWEEN from_milliseconds(1636526171043) AND
 from_milliseconds(1636612571043)
            AND measure_name = 'metrics'
        GROUP BY region, cell, silo, availability_zone, microservice_name
    )
    GROUP BY microservice_name
)
```

예약된 쿼리로 변환

이전 쿼리는 다음과 같이 예약된 쿼리로 변환할 수 있습니다. 먼저 리전, 셀, 사일로, 가용 영역 및 마이크로서비스의 지정된 배포 내에서 고유한 호스트 이름을 계산합니다. 그런 다음 호스트를 추가하여 마이크로서비스 호스트 수당 시간당 을 계산합니다. 예약된 쿼리에서 지원하는 `@scheduled_runtime` 파라미터를 사용하여 쿼리가 호출된 지난 한 시간 동안 다시 계산할 수 있습니다. 내부 쿼리의 `WHERE` 절`bin(@scheduled_runtime, 1h)`에서 는 쿼리가 한 시간 중 한 시간에 예약되어 있더라도 1시간 동안 데이터를 얻을 수 있도록 합니다.

쿼리가 시간당 집계를 계산하더라도 예약된 계산 구성에서 볼 수 있듯이 파생 테이블에서 업데이트를 더 빨리 받을 수 있도록 30분마다 새로 고치도록 설정되어 있습니다. 신선도 요구 사항에 따라 이를 조정할 수 있습니다. 예를 들어 15분마다 집계를 다시 계산하거나 시간 경계에서 다시 계산할 수 있습니다.

```
SELECT microservice_name, hour, SUM(num_instances) AS num_instances
FROM (
        SELECT microservice_name, bin(time, 1h) AS hour,
            COUNT(DISTINCT instance_name) as num_instances
       FROM raw_data.devops
       WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND @scheduled_runtime
```

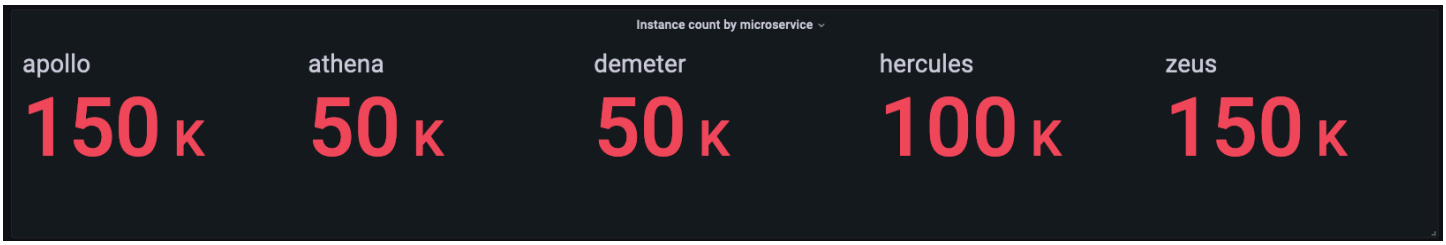
```
            AND measure_name = 'metrics'
        GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)

     )
GROUP BY microservice_name, hour
```

```
{
    "Name": "MultiPT30mHostCountMicroservicePerHr",
    "QueryString": "SELECT microservice_name, hour, SUM(num_instances) AS num_instances
    FROM (        SELECT microservice_name, bin(time, 1h) AS hour, COUNT(DISTINCT
 instance_name) as num_instances        FROM raw_data.devops        WHERE time BETWEEN
 bin(@scheduled_runtime, 1h) - 1h AND @scheduled_runtime            AND measure_name
 = 'metrics'        GROUP BY region, cell, silo, availability_zone, microservice_name,
 bin(time, 1h)    )    GROUP BY microservice_name, hour",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "host_count_pt1h",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "num_instances",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "num_instances",
                        "MeasureValueType": "BIGINT"
                    }
                ]
            }
        }
```

```
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

새 대시보드에서 미리 계산된 결과 사용

이제 생성한 예약된 쿼리에서 파생된 테이블을 사용하여 집계 보기 대시보드를 생성하는 방법을 확인할 수 있습니다. 대시보드 스냅샷에서 파생 테이블과 기본 테이블에서 계산된 집계도 일치하는지 확인할 수도 있습니다. 파생 테이블을 사용하여 대시보드를 생성하면 원시 데이터에서 이러한 집계를 계산하는 것보다 훨씬 빠른 로드 시간과 파생 테이블 사용 비용이 절감됩니다. 다음은 사전 계산된 데이터를 사용하는 대시보드의 스냅샷이며, 테이블 'derived'.'host_count_pt1h'에 저장된 사전 계산된 데이터를 사용하여 이 패널을 렌더링하는 데 사용되는 쿼리입니다. 쿼리의 구조는 원시 데이터에 대해 대시보드에서 사용된 쿼리와 매우 유사하지만, 예외적으로 이 쿼리가 집계하는 고유 수를 이미 계산하는 파생 테이블을 사용합니다.




```
SELECT CASE WHEN microservice_name = 'apollo' THEN num_instances ELSE NULL END AS
 apollo,
    CASE WHEN microservice_name = 'athena' THEN num_instances ELSE NULL END AS athena,
    CASE WHEN microservice_name = 'demeter' THEN num_instances ELSE NULL END AS
 demeter,
    CASE WHEN microservice_name = 'hercules' THEN num_instances ELSE NULL END AS
 hercules,
    CASE WHEN microservice_name = 'zeus' THEN num_instances ELSE NULL END AS zeus
FROM (
    SELECT microservice_name, AVG(num_instances) AS num_instances
    FROM (
        SELECT microservice_name, bin(time, 1h), SUM(num_instances) as num_instances
        FROM "derived"."host_count_pt1h"
```

```
        WHERE time BETWEEN from_milliseconds(1636567785421) AND
 from_milliseconds(1636654185421)
            AND measure_name = 'num_instances'
        GROUP BY microservice_name, bin(time, 1h)
    )
    GROUP BY microservice_name
)
```

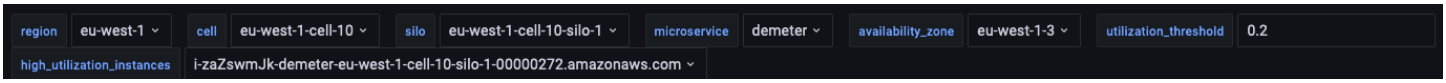
드릴다운에 예약된 쿼리 및 원시 데이터 사용

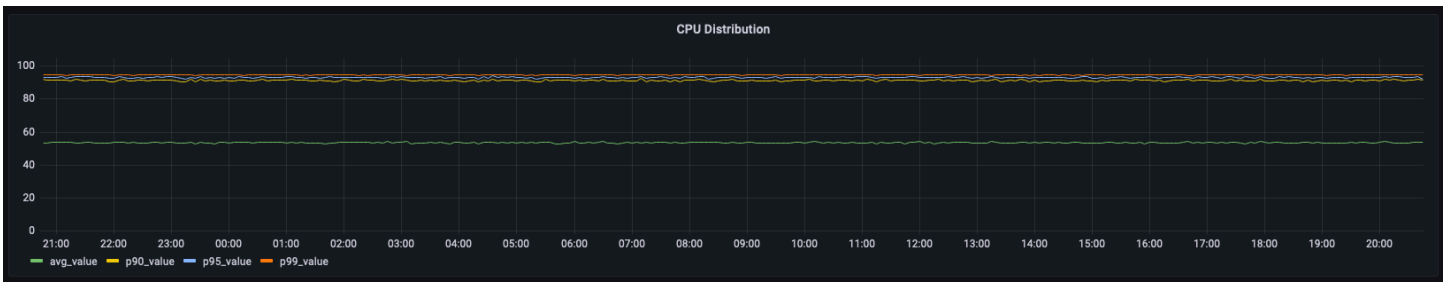
플릿 전체에서 집계된 통계를 사용하여 드릴다운이 필요한 영역을 식별한 다음 원시 데이터를 사용하여 세분화된 데이터로 드릴다운하여 심층적인 인사이트를 얻을 수 있습니다.

이 예제에서는 집계 대시보드를 사용하여 다른 배포에 비해 CPU 사용률이 더 높은 것으로 보이는 배포(배포는 지정된 리전, 셀, 사일로 및 가용 영역 내의 특정 마이크로서비스용)를 식별하는 방법을 알아봅니다. 그런 다음 원시 데이터를 사용하여 드릴다운하여 더 잘 이해할 수 있습니다. 이러한 드릴다운은 빈도가 낮고 배포와 관련된 데이터에만 액세스할 수 있으므로 이 분석에 원시 데이터를 사용할 수 있으며 예약된 쿼리를 사용할 필요가 없습니다.

배포 드릴다운당

아래 대시보드는 지정된 배포 내에서 보다 세분화된 서버 수준 통계를 심층적으로 보여줍니다. 플릿의 다양한 부분을 심층 분석할 수 있도록 이 대시보드는 리전, 셀, 사일로, 마이크로서비스 및 availability_zone과 같은 변수를 사용합니다. 그런 다음 해당 배포에 대한 몇 가지 집계 통계를 표시합니다.





아래 쿼리에서 변수 드롭다운에서 선택한 값이 쿼리 WHERE 절의 예측 변수로 사용되므로 배포를 위한 데이터에만 집중할 수 있습니다. 그런 다음 패널은 해당 배포의 인스턴스에 대한 집계 CPU 지표를 표시합니다. 원시 데이터를 사용하여 대화형 쿼리 지연 시간으로 이 드릴다운을 수행하여 더 깊은 인사이트를 도출할 수 있습니다.

```
SELECT bin(time, 5m) as minute,
```

```
    ROUND(AVG(cpu_user), 2) AS avg_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.9), 2) AS p90_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.95), 2) AS p95_value,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.99), 2) AS p99_value
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099476) AND
 from_milliseconds(1636613499476)
    AND region = 'eu-west-1'
    AND cell = 'eu-west-1-cell-10'
    AND silo = 'eu-west-1-cell-10-silo-1'
    AND microservice_name = 'demeter'
    AND availability_zone = 'eu-west-1-3'
    AND measure_name = 'metrics'
GROUP BY bin(time, 5m)
ORDER BY 1
```

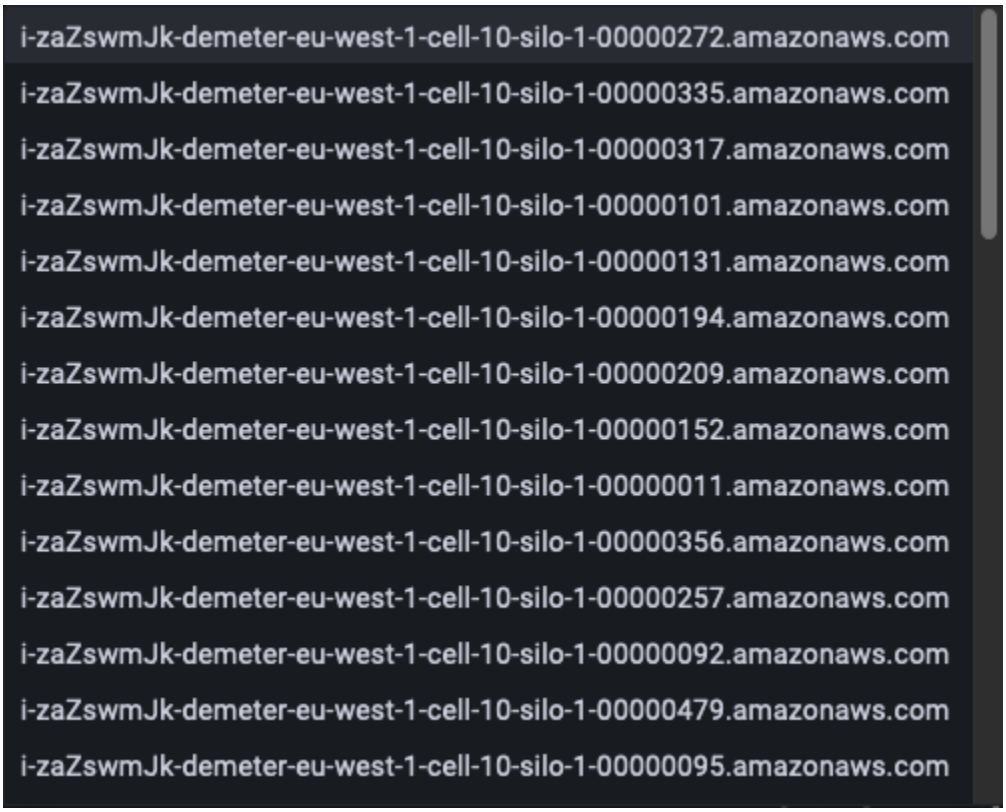
인스턴스 수준 통계

또한 이 대시보드는 사용CPU률이 높은 서버/인스턴스를 나열하는 다른 변수를 사용률 내림차순으로 정렬하여 계산합니다. 이 변수를 계산하는 데 사용되는 쿼리가 아래에 표시됩니다.

```
WITH microservice_cell_avg AS (
    SELECT AVG(cpu_user) AS microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE $__timeFilter
        AND measure_name = 'metrics'
        AND region = '${region}'
        AND cell = '${cell}'
        AND silo = '${silo}'
        AND availability_zone = '${availability_zone}'
        AND microservice_name = '${microservice}'
), instance_avg AS (
    SELECT instance_name,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE $__timeFilter
        AND measure_name = 'metrics'
        AND region = '${region}'
        AND cell = '${cell}'
        AND silo = '${silo}'
        AND microservice_name = '${microservice}'
        AND availability_zone = '${availability_zone}'
    GROUP BY availability_zone, instance_name
```

```
)
SELECT i.instance_name
FROM instance_avg i CROSS JOIN microservice_cell_avg m
WHERE i.instance_avg_metric > (1 + ${utilization_threshold}) *
 m.microservice_avg_metric
ORDER BY i.instance_avg_metric DESC
```

이전 쿼리에서 변수는 다른 변수에 대해 선택한 값에 따라 동적으로 다시 계산됩니다. 배포를 위해 변수가 채워지면 목록에서 개별 인스턴스를 선택하여 해당 인스턴스의 지표를 추가로 시각화할 수 있습니다. 아래 스냅샷과 같이 인스턴스 이름의 드롭다운에서 다른 인스턴스를 선택할 수 있습니다.



앞의 패널은 선택한 인스턴스의 통계를 표시하며, 아래는 이러한 통계를 가져오는 데 사용되는 쿼리입니다.

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(cpu_user) AS avg_cpu,
    ROUND(APPROX_PERCENTILE(cpu_user, 0.99), 2) as p99_cpu
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499477)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(memory_used) AS avg_memory,
    ROUND(APPROX_PERCENTILE(memory_used, 0.99), 2) as p99_memory
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499477)
    AND measure_name = 'metrics'
```

```
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

```
SELECT COUNT(gc_pause)
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099477) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'events'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
SELECT avg(gc_pause) as avg, round(approx_percentile(gc_pause, 0.99), 2) as p99
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099478) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'events'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
SELECT BIN(time, 30m) AS time_bin,
    AVG(disk_io_reads) AS avg,
    ROUND(APPROX_PERCENTILE(disk_io_reads, 0.99), 2) as p99
FROM "raw_data"."devops"
WHERE time BETWEEN from_milliseconds(1636527099478) AND
 from_milliseconds(1636613499478)
    AND measure_name = 'metrics'
    AND region = 'eu-west-1' AND cell = 'eu-west-1-cell-10' AND silo = 'eu-west-1-
cell-10-silo-1'
    AND availability_zone = 'eu-west-1-3' AND microservice_name = 'demeter'
    AND instance_name = 'i-zaZswmJk-demeter-eu-west-1-cell-10-
silo-1-00000272.amazonaws.com'
```

```
GROUP BY BIN(time, 30m)
ORDER BY time_bin desc
```

대시보드 간에 예약된 쿼리를 공유하여 비용 최적화

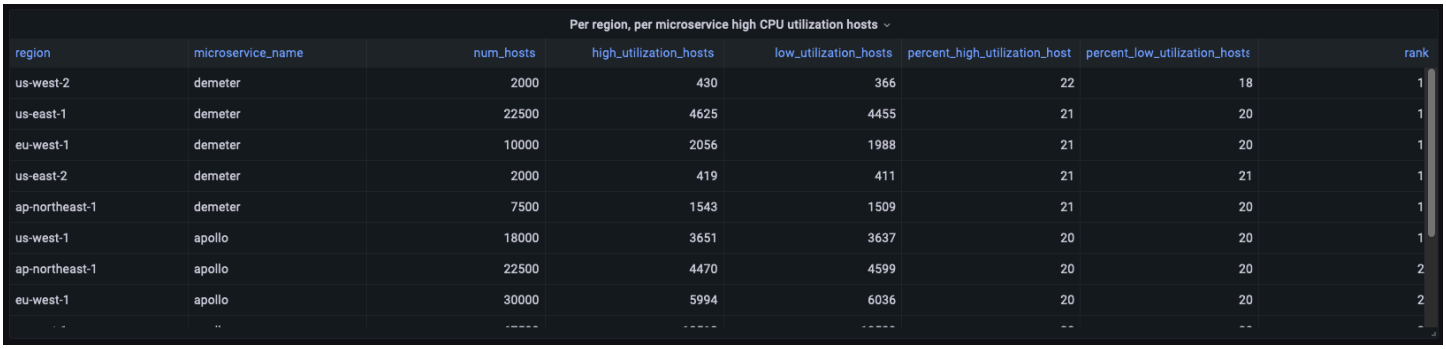
이 예제에서는 여러 대시보드 패널이 유사한 정보의 변형(높은 CPU 호스트 및 CPU 사용률이 높은 플릿 비율 찾기)을 표시하는 시나리오와 동일한 예약된 쿼리를 사용하여 결과를 미리 계산한 다음 여러 패널을 채우는 방법을 알아봅니다. 이렇게 재사용하면 각 패널에 대해 하나씩 서로 다른 예약된 쿼리를 사용하는 대신 소유자만 사용하는 비용을 추가로 최적화할 수 있습니다.

원시 데이터가 포함된 대시보드 패널

CPU 마이크로서비스당 리전별 사용률

첫 번째 패널은 평균 CPU 사용률이 리전, 셀, 사일로, 가용 영역 및 마이크로서비스 내에서 지정된 배포에 대한 위 CPU 사용률보다 낮거나 높은 임계값인 인스턴스를 계산합니다. 그런 다음 사용률이 높은 호스트 비율이 가장 높은 리전과 마이크로서비스를 정렬합니다. 특정 배포의 서버가 얼마나 더운지 식별한 다음 드릴다운하여 문제를 더 잘 이해하는 데 도움이 됩니다.

패널에 대한 쿼리는 LiveAnalytics가 일반적인 테이블 표현식, 창 함수, 조인 등을 사용하여 복잡한 분석 작업을 수행할 수 있도록 SQL 지원하는 Timestream의 유연성을 보여줍니다.



Per region, per microservice high CPU utilization hosts

region	microservice_name	num_hosts	high_utilization_hosts	low_utilization_hosts	percent_high_utilization_host	percent_low_utilization_hosts	rank
us-west-2	demeter	2000	430	366	22	18	1
us-east-1	demeter	22500	4625	4455	21	20	1
eu-west-1	demeter	10000	2056	1988	21	20	1
us-east-2	demeter	2000	419	411	21	21	1
ap-northeast-1	demeter	7500	1543	1509	21	20	1
us-west-1	apollo	18000	3651	3637	20	20	1
ap-northeast-1	apollo	22500	4470	4599	20	20	2
eu-west-1	apollo	30000	5994	6036	20	20	2

쿼리:

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, AVG(cpu_user) AS
 microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526593876) AND
 from_milliseconds(1636612993876)
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name
```

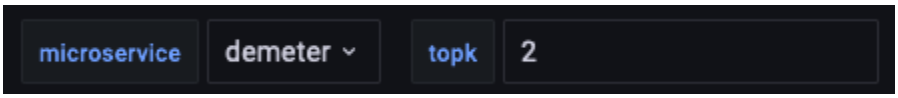
```
), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526593876) AND
 from_milliseconds(1636612993876)
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name
), instances_above_threshold AS (
  SELECT i.*,
    CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS high_utilization,
    CASE WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS low_utilization
  FROM instance_avg i INNER JOIN microservice_cell_avg m
    ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
      AND m.microservice_name = i.microservice_name
), per_deployment_high AS (
SELECT region, microservice_name, COUNT(*) AS num_hosts, SUM(high_utilization) AS
 high_utilization_hosts, SUM(low_utilization) AS low_utilization_hosts,
    ROUND(SUM(high_utilization) * 100.0 / COUNT(*), 0) AS
 percent_high_utilization_hosts,
    ROUND(SUM(low_utilization) * 100.0 / COUNT(*), 0) AS percent_low_utilization_hosts
FROM instances_above_threshold
GROUP BY region, microservice_name
), per_region_ranked AS (
    SELECT *,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY percent_high_utilization_hosts
 DESC, high_utilization_hosts DESC) AS rank
    FROM per_deployment_high
)
SELECT *
FROM per_region_ranked
WHERE rank <= 2
ORDER BY percent_high_utilization_hosts desc, rank asc
```

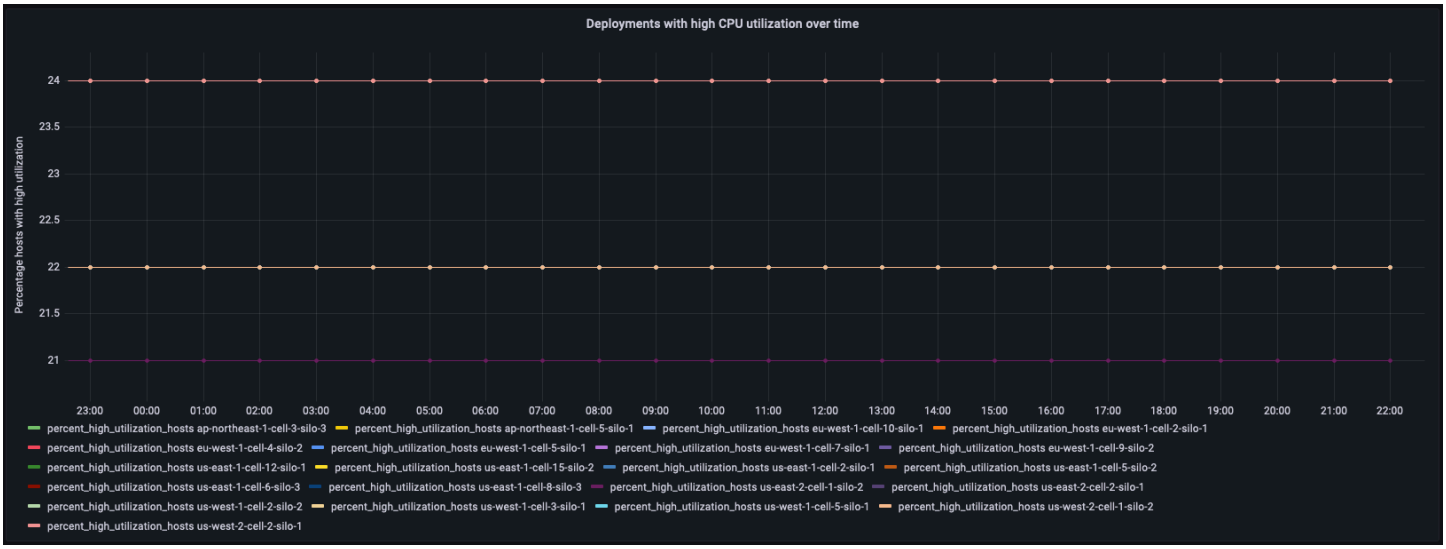
마이크로서비스로 드릴다운하여 핫 스팟 찾기

다음 대시보드를 사용하면 마이크로서비스 중 하나를 심층적으로 분석하여 해당 마이크로서비스의 특정 리전, 셀 및 사일로가 더 높은 CPU 사용률로 플릿의 몇 분이나 실행되고 있는지 확인할 수 있습니다. 예를 들어 플릿 전체 대시보드에서 마이크로서비스 지표가 상위 몇 개의 순위 위치에 표시되는 것을 확인했으므로 이 대시보드에서는 해당 마이크로서비스를 더 자세히 분석하려고 합니다.

이 대시보드는 변수를 사용하여 마이크로서비스를 선택하여 드릴다운하고 변수의 값은 차원의 고유한 값을 사용하여 채워집니다. 마이크로서비스를 선택하면 나머지 대시보드가 새로 고쳐집니다.

아래 보이는 것처럼 첫 번째 패널은 시간 경과에 따른 배포(마이크로서비스의 리전, 셀 및 사일로)의 호스트 백분율과 대시보드를 구성하는 데 사용되는 해당 쿼리를 표시합니다. 이 플롯 자체는 가 높은 호스트의 비율이 높은 특정 배포를 식별합니다CPU.





쿼리:

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, bin(time, 1h) as
 hour, AVG(cpu_user) AS microservice_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526898831) AND
 from_milliseconds(1636613298831)
        AND measure_name = 'metrics'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)
), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h) as hour,
        AVG(cpu_user) AS instance_avg_metric
    FROM "raw_data"."devops"
    WHERE time BETWEEN from_milliseconds(1636526898831) AND
 from_milliseconds(1636613298831)
        AND measure_name = 'metrics'
```

```
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h)
), instances_above_threshold AS (
  SELECT i.*,
    CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE
 0 END AS high_utilization
  FROM instance_avg i INNER JOIN microservice_cell_avg m
    ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
      AND m.microservice_name = i.microservice_name AND m.hour = i.hour
), high_utilization_percent AS (
    SELECT region, cell, silo, microservice_name, hour, COUNT(*) AS num_hosts,
 SUM(high_utilization) AS high_utilization_hosts,
        ROUND(SUM(high_utilization) * 100.0 / COUNT(*), 0) AS
 percent_high_utilization_hosts
    FROM instances_above_threshold
    GROUP BY region, cell, silo, microservice_name, hour
), high_utilization_ranked AS (
    SELECT region, cell, silo, microservice_name,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY
 AVG(percent_high_utilization_hosts) desc, AVG(high_utilization_hosts) desc) AS rank
    FROM high_utilization_percent
    GROUP BY region, cell, silo, microservice_name
)
SELECT hup.silo, CREATE_TIME_SERIES(hour, hup.percent_high_utilization_hosts) AS
 percent_high_utilization_hosts
FROM high_utilization_percent hup INNER JOIN high_utilization_ranked hur
    ON hup.region = hur.region AND hup.cell = hur.cell AND hup.silo = hur.silo AND
 hup.microservice_name = hur.microservice_name
WHERE rank <= 2
GROUP BY hup.region, hup.cell, hup.silo
ORDER BY hup.silo
```

재사용을 가능하게 하는 단일 예약 쿼리로 변환

두 대시보드의 여러 패널에서 유사한 계산이 수행됩니다. 각 패널에 대해 별도의 예약된 쿼리를 정의할 수 있습니다. 여기에서는 3개의 패널을 모두 렌더링하는 데 결과를 사용할 수 있는 하나의 예약된 쿼리를 정의하여 비용을 추가로 최적화하는 방법을 확인할 수 있습니다.

다음은 서로 다른 모든 패널에 대해 계산되고 사용되는 집계를 캡처하는 쿼리입니다. 이 예약된 쿼리의 정의에서 몇 가지 중요한 측면을 관찰할 수 있습니다.

- 일반적인 테이블 표현식, 조인, 대/소문자 설명 등을 사용할 수 있는 예약된 쿼리에서 지원하는 SQL 표면 영역의 유연성과 성능입니다.
- 예약된 쿼리 하나를 사용하여 특정 대시보드에 필요할 수 있는 것보다 세분화된 통계와 대시보드가 다른 변수에 사용할 수 있는 모든 값을 계산할 수 있습니다. 예를 들어 집계는 리전, 셀, 사일로 및 마이크로서비스에서 계산됩니다. 따라서 이를 결합하여 리전 수준 또는 리전 및 마이크로서비스 수준 집계를 생성할 수 있습니다. 마찬가지로 동일한 쿼리는 모든 리전, 셀, 사일로 및 마이크로서비스의 집계를 계산합니다. 이를 통해 이러한 열에 필터를 적용하여 값의 하위 집합에 대한 집계를 가져올 수 있습니다. 예를 들어 us-east-1과 같은 하나의 리전 또는 하나의 마이크로서비스에서 데미터를 말하거나 리전, 셀, 사일로 및 마이크로서비스 내의 특정 배포로 드릴다운하는 하나의 리전에 대한 집계를 계산할 수 있습니다. 이 접근 방식은 사전 계산된 집계를 유지하는 데 드는 비용을 더욱 최적화합니다.

```
WITH microservice_cell_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, bin(time, 1h) as
 hour, AVG(cpu_user) AS microservice_avg_metric
    FROM raw_data.devops
    WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h)
 + 1h
        AND measure_name = 'metrics'
    GROUP BY region, cell, silo, availability_zone, microservice_name, bin(time, 1h)

), instance_avg AS (
    SELECT region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h) as hour,
        AVG(cpu_user) AS instance_avg_metric
  FROM raw_data.devops
  WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h)
 + 1h
      AND measure_name = 'metrics'
  GROUP BY region, cell, silo, availability_zone, microservice_name, instance_name,
 bin(time, 1h)
), instances_above_threshold AS (
    SELECT i.*,
        CASE WHEN i.instance_avg_metric > (1 + 0.2) * m.microservice_avg_metric THEN 1
 ELSE 0 END AS high_utilization,
        CASE WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1
 ELSE 0 END AS low_utilization
  FROM instance_avg i INNER JOIN microservice_cell_avg m
      ON i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND
 i.availability_zone = m.availability_zone
```

```
           AND m.microservice_name = i.microservice_name AND m.hour = i.hour
)
SELECT region, cell, silo, microservice_name, hour,
     COUNT(*) AS num_hosts, SUM(high_utilization) AS high_utilization_hosts,
 SUM(low_utilization) AS low_utilization_hosts
FROM instances_above_threshold GROUP BY region, cell, silo, microservice_name, hour
```

다음은 이전 쿼리에 대해 예약된 쿼리 정의입니다. 일정 표현식은 30분마다 새로 고치도록 구성되며 최대 1시간 동안 데이터를 새로 고칩니다. 다시 bin(@scheduled_runtime, 1시간) 구성 요소를 사용하여 전체 시간의 이벤트를 가져옵니다. 애플리케이션의 신선도 요구 사항에 따라 새로 고침 빈도를 늘리거나 줄이도록 구성할 수 있습니다. WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h) + 1h를 사용하면 15분마다 한 번씩 새로 고치더라도 현재 시간과 이전 시간에 대한 전체 시간 데이터를 얻을 수 있습니다.

나중에 세 패널이 테이블 deployment_cpu_stats_per_hr에 작성된 이러한 집계를 사용하여 패널과 관련된 지표를 시각화하는 방법을 알아봅니다.

```
{
    "Name": "MultiPT30mHighCpuDeploymentsPerHr",
    "QueryString": "WITH microservice_cell_avg AS (    SELECT region, cell,
 silo, availability_zone, microservice_name, bin(time, 1h) as hour, AVG(cpu_user)
 AS microservice_avg_metric    FROM raw_data.devops    WHERE time BETWEEN
 bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime, 1h) + 1h    AND
 measure_name = 'metrics'    GROUP BY region, cell, silo, availability_zone,
 microservice_name, bin(time, 1h)    ), instance_avg AS (    SELECT region,
 cell, silo, availability_zone, microservice_name, instance_name, bin(time, 1h)
 as hour,    AVG(cpu_user) AS instance_avg_metric    FROM raw_data.devops
 WHERE time BETWEEN bin(@scheduled_runtime, 1h) - 1h AND bin(@scheduled_runtime,
 1h) + 1h    AND measure_name = 'metrics'    GROUP BY region, cell, silo,
 availability_zone, microservice_name, instance_name, bin(time, 1h)    ),
 instances_above_threshold AS (    SELECT i.*,    CASE WHEN i.instance_avg_metric >
 (1 + 0.2) * m.microservice_avg_metric THEN 1 ELSE 0 END AS high_utilization,    CASE
 WHEN i.instance_avg_metric < (1 - 0.2) * m.microservice_avg_metric THEN 1 ELSE 0 END
 AS low_utilization    FROM instance_avg i INNER JOIN microservice_cell_avg m    ON
 i.region = m.region AND i.cell = m.cell AND i.silo = m.silo AND i.availability_zone
 = m.availability_zone    AND m.microservice_name = i.microservice_name AND m.hour =
 i.hour    )     SELECT region, cell, silo, microservice_name, hour,         COUNT(*)
 AS num_hosts, SUM(high_utilization) AS high_utilization_hosts, SUM(low_utilization) AS
 low_utilization_hosts    FROM instances_above_threshold GROUP BY region, cell, silo,
 microservice_name, hour",
    "ScheduleConfiguration": {
        "ScheduleExpression": "cron(0/30 * * * ? *)"
```

```
    },
    "NotificationConfiguration": {
        "SnsConfiguration": {
            "TopicArn": "******"
        }
    },
    "TargetConfiguration": {
        "TimestreamConfiguration": {
            "DatabaseName": "derived",
            "TableName": "deployment_cpu_stats_per_hr",
            "TimeColumn": "hour",
            "DimensionMappings": [
                {
                    "Name": "region",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "cell",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "silo",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "Name": "microservice_name",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": {
                "TargetMultiMeasureName": "cpu_user",
                "MultiMeasureAttributeMappings": [
                    {
                        "SourceColumn": "num_hosts",
                        "MeasureValueType": "BIGINT"
                    },
                    {
                        "SourceColumn": "high_utilization_hosts",
                        "MeasureValueType": "BIGINT"
                    },
                    {
                        "SourceColumn": "low_utilization_hosts",
                        "MeasureValueType": "BIGINT"
                    }
```

```
                ]
            }
        }
    },
    "ErrorReportConfiguration": {
        "S3Configuration" : {
            "BucketName" : "******",
            "ObjectKeyPrefix": "errors",
            "EncryptionOption": "SSE_S3"
        }
    },
    "ScheduledQueryExecutionRoleArn": "******"
}
```

사전 계산된 결과의 대시보드

높은 CPU 사용률의 호스트

사용률이 높은 호스트의 경우 여러 패널이 deployment_cpu_stats_per_hr의 데이터를 사용하여 패널에 필요한 다양한 집계를 계산하는 방법을 확인할 수 있습니다. 예를 들어 이 패널은 리전 수준 정보를 제공하므로 리전 또는 마이크로서비스를 필터링하지 않고 리전 및 마이크로서비스별로 그룹화된 집계를 보고합니다.

Per region, per microservice high utilization hosts

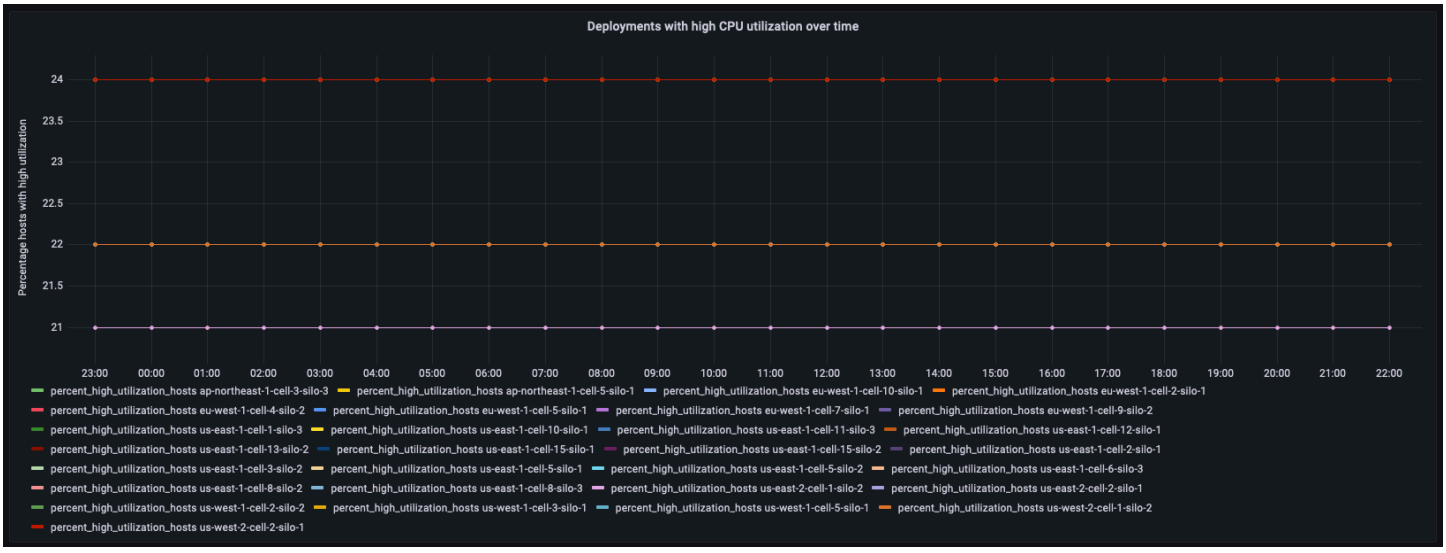
region	microservice_name	num_hosts	high_utilization_hosts	low_utilization_hosts	percent_high_utilization_host	percent_low_utilization_hosts	rank
us-west-2	demeter	1962	423	359	22	18	1
us-east-2	demeter	2000	419	411	21	21	1
us-east-1	demeter	22500	4628	4455	21	20	1
ap-northeast-1	demeter	7500	1544	1509	21	20	1
eu-west-1	demeter	9983	2056	1984	21	20	1
us-west-1	apollo	18000	3657	3643	20	20	1
ap-northeast-1	apollo	22500	4470	4599	20	20	2
us-east-2	hercules	4000	813	752	20	19	2

```
WITH per_deployment_hosts AS (
    SELECT region, cell, silo, microservice_name,
        AVG(num_hosts) AS num_hosts,
        AVG(high_utilization_hosts) AS high_utilization_hosts,
        AVG(low_utilization_hosts) AS low_utilization_hosts
    FROM "derived"."deployment_cpu_stats_per_hr"
    WHERE time BETWEEN from_milliseconds(1636567785437) AND
 from_milliseconds(1636654185437)
        AND measure_name = 'cpu_user'
    GROUP BY region, cell, silo, microservice_name
), per_deployment_high AS (
```

```
    SELECT region, microservice_name,
        SUM(num_hosts) AS num_hosts,
        ROUND(SUM(high_utilization_hosts), 0) AS high_utilization_hosts,
        ROUND(SUM(low_utilization_hosts),0) AS low_utilization_hosts,
        ROUND(SUM(high_utilization_hosts) * 100.0 / SUM(num_hosts)) AS
 percent_high_utilization_hosts,
        ROUND(SUM(low_utilization_hosts) * 100.0 / SUM(num_hosts)) AS
 percent_low_utilization_hosts
    FROM per_deployment_hosts
    GROUP BY region, microservice_name
),
per_region_ranked AS (
    SELECT *,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY percent_high_utilization_hosts
 DESC, high_utilization_hosts DESC) AS rank
    FROM per_deployment_high
)
SELECT *
FROM per_region_ranked
WHERE rank <= 2
ORDER BY percent_high_utilization_hosts desc, rank asc
```

마이크로서비스로 드릴다운하여 CPU 사용량이 많은 배포 찾기

다음 예제에서는 deployment_cpu_stats_per_hr 파생 테이블을 다시 사용하지만 이제 특정 마이크로서비스에 필터를 적용합니다(이 예제에서는 집계 대시보드에서 사용률이 높은 호스트를 보고했기 때문에 경계값). 이 패널은 시간 경과에 따른 사용CPU률이 높은 호스트의 백분율을 추적합니다.




```
WITH high_utilization_percent AS (
```

```
    SELECT region, cell, silo, microservice_name, bin(time, 1h) AS hour, MAX(num_hosts)
 AS num_hosts,
        MAX(high_utilization_hosts) AS high_utilization_hosts,
        ROUND(MAX(high_utilization_hosts) * 100.0 / MAX(num_hosts)) AS
 percent_high_utilization_hosts
    FROM "derived"."deployment_cpu_stats_per_hr"
    WHERE time BETWEEN from_milliseconds(1636525800000) AND
 from_milliseconds(1636612200000)
        AND measure_name = 'cpu_user'
        AND microservice_name = 'demeter'
    GROUP BY region, cell, silo, microservice_name, bin(time, 1h)
), high_utilization_ranked AS (
    SELECT region, cell, silo, microservice_name,
        DENSE_RANK() OVER (PARTITION BY region ORDER BY
 AVG(percent_high_utilization_hosts) desc, AVG(high_utilization_hosts) desc) AS rank
    FROM high_utilization_percent
    GROUP BY region, cell, silo, microservice_name
)
SELECT hup.silo, CREATE_TIME_SERIES(hour, hup.percent_high_utilization_hosts) AS
 percent_high_utilization_hosts
FROM high_utilization_percent hup INNER JOIN high_utilization_ranked hur
    ON hup.region = hur.region AND hup.cell = hur.cell AND hup.silo = hur.silo AND
 hup.microservice_name = hur.microservice_name
WHERE rank <= 2
GROUP BY hup.region, hup.cell, hup.silo
ORDER BY hup.silo
```

기본 테이블의 쿼리와 예약된 쿼리 결과의 쿼리 비교

이 Timestream 쿼리 예제에서는 다음 스키마, 예제 쿼리 및 출력을 사용하여 기본 테이블의 쿼리를 예약된 쿼리 결과의 파생 테이블의 쿼리와 비교합니다. 잘 계획된 예약 쿼리를 사용하면 원래 기본 테이블에서 가능한 것보다 더 빠른 쿼리로 이어질 수 있는 행 및 기타 특성이 더 적은 파생 테이블을 얻을 수 있습니다.

이 시나리오를 설명하는 동영상은 Amazon Timestream for 에서 예약된 쿼리를 사용하여 쿼리 성능 개선 및 비용 절감을 참조하세요 LiveAnalytics.

이 예제에서는 다음 시나리오를 사용합니다.

- 리전 - us-east-1
- 기본 테이블 - `"clickstream"."shopping"`
- 파생 테이블 - `"clickstream"."aggregate"`

기본 테이블

다음은 기본 테이블의 스키마를 설명합니다.

열	유형	LiveAnalytics 속성 유형에 대한 타임스트림
채널	varchar	MULTI
설명	varchar	MULTI
이벤트	varchar	DIMENSION
ip_address	varchar	DIMENSION
measure_name	varchar	MEASURE_NAME
product	varchar	MULTI
product_id	varchar	MULTI
양	double	MULTI
쿼리	varchar	MULTI
session_id	varchar	DIMENSION
user_group	varchar	DIMENSION
user_id	varchar	DIMENSION

다음은 기본 테이블의 측정값을 설명합니다. 기본 테이블은 예약된 쿼리가 실행되는 Timestream의 테이블을 나타냅니다.

- measure_name – `metrics`
- 데이터 - 다중
- 차원:

```
[ ( user_group, varchar ),( user_id, varchar ),( session_id, varchar ),( ip_address,
 varchar ),( event, varchar ) ]
```

기본 테이블의 쿼리

다음은 지정된 시간 범위에서 5분 집계로 수를 수집하는 임시 쿼리입니다.

```
SELECT BIN(time, 5m) as time,
channel,
product_id,
SUM(quantity) as product_quantity
FROM "clickstream"."shopping"
WHERE BIN(time, 5m) BETWEEN '2023-05-11 10:10:00.000000000' AND '2023-05-11
 10:30:00.000000000'
AND channel = 'Social media'
and product_id = '431412'
GROUP BY BIN(time, 5m),channel,product_id
```

출력:

```
duration:1.745 sec
Bytes scanned: 29.89 MB
Query Id: AEBQEANMHG7MHHBHCKJ3BSOE3QUGIDBGWCCP5I6J6YUW5CVJZ2M3JCJ27QRMM7A
Row count:5
```

예약된 쿼리

다음은 5분마다 실행되는 예약된 쿼리입니다.

```
SELECT BIN(time, 5m) as time, channel as measure_name, product_id, product,
SUM(quantity) as product_quantity
FROM "clickstream"."shopping"
WHERE time BETWEEN BIN(@scheduled_runtime, 5m) - 10m AND BIN(@scheduled_runtime, 5m) -
 5m
AND channel = 'Social media'
GROUP BY BIN(time, 5m), channel, product_id, product
```

파생 테이블의 쿼리

다음은 파생 테이블의 임시 쿼리입니다. 파생 테이블은 예약된 쿼리의 결과가 포함된 Timestream 테이블을 나타냅니다.

```
SELECT time, measure_name, product_id,product_quantity
FROM "clickstream"."aggregate"
```

```
WHERE time BETWEEN '2023-05-11 10:10:00.000000000' AND '2023-05-11 10:30:00.000000000'
AND measure_name = 'Social media'
and product_id = '431412'
```

출력:

```
duration: 0.2960 sec
Bytes scanned: 235.00 B
QueryID: AEBQEANMHHAAQU4FFTT6CFM6UYXTL4SMLZV22MFP4KV2Z7IRVOPLOMLDD6BR33Q
Row count: 5
```

비교

다음은 기본 테이블의 쿼리 결과를 파생 테이블의 쿼리와 비교한 것입니다. 예약된 쿼리를 통해 결과를 집계한 파생 테이블의 동일한 쿼리는 스캔된 바이트 수를 줄이면 더 빠르게 완료됩니다.

이러한 결과는 예약된 쿼리를 사용하여 더 빠른 쿼리를 위한 데이터를 집계하는 값을 보여줍니다.

	기본 테이블의 쿼리	파생 테이블의 쿼리
지속 시간	1.745초	0.2960초
스캔된 바이트 수	29.89MB	235바이트
행 수	5	5

UNLOAD 를 사용하여 에 대해 Timestream에서 S3로 쿼리 결과 내보내기 LiveAnalytics

LiveAnalytics 이제 Amazon Timestream을 사용하면 UNLOAD 명령문을 사용하여 비용 효율적이고 안전한 방식으로 쿼리 결과를 Amazon S3로 내보낼 수 있습니다. UNLOAD 문을 사용하여 이제 Apache Parquet 또는 쉼표로 구분된 값(CSV) 형식으로 선택한 S3 버킷으로 시계열 데이터를 내보낼 수 있습니다. 이를 통해 시계열 데이터를 다른 서비스와 저장, 결합 및 분석할 수 있습니다. UNLOAD 문을 사용하면 데이터를 압축된 방식으로 내보낼 수 있으므로 전송된 데이터와 필요한 스토리지 공간이 줄어듭니다. UNLOAD 또한 는 데이터를 내보낼 때 선택한 속성을 기반으로 파티셔닝을 지원하여 성능을 개선하고 데이터에 액세스하는 다운스트림 서비스의 처리 시간을 줄입니다. 또한 Amazon S3 관리형 키(SSE-S3) 또는 AWS Key Management Service(AWS KMS) 관리형 키(SSE-KMS)를 사용하여 내보낸 데이터를 암호화할 수 있습니다.

UNLOAD 에 대한 Timestream의 이점 LiveAnalytics

UNLOAD 문 사용의 주요 이점은 다음과 같습니다.

- 운영 편의성 - UNLOAD 문을 사용하면 Apache Parquet 또는 CSV 형식으로 단일 쿼리 요청으로 기가바이트의 데이터를 내보낼 수 있으므로 다운스트림 처리 요구 사항에 가장 적합한 형식을 선택할 수 있는 유연성을 제공하고 데이터 레이크를 더 쉽게 구축할 수 있습니다.
- 보안 및 비용 효율성 SSE- UNLOAD KMS 문은 데이터를 S3 버킷으로 압축 방식으로 내보내고 고객 관리형 키를 사용하여 데이터를 암호화(- 또는 SSE_S3)하여 데이터 스토리지 비용을 절감하고 무단 액세스로부터 보호하는 기능을 제공합니다.
- 성능 - UNLOAD 문을 사용하여 S3 버킷으로 내보낼 때 데이터를 분할할 수 있습니다. 데이터를 분할하면 다운스트림 서비스가 데이터를 병렬로 처리하여 처리 시간을 줄일 수 있습니다. 또한 다운스트림 서비스는 필요한 데이터만 처리할 수 있으므로 필요한 처리 리소스가 줄어들고 관련 비용이 절감됩니다.

에 대한 TimestreamUNLOAD의 에 대한 사용 사례 LiveAnalytics

UNLOAD 문을 사용하여 S3 버킷에 데이터를 다음과 같이 쓸 수 있습니다.

- 빌드 데이터 웨어하우스 - 기가바이트의 쿼리 결과를 S3 버킷으로 내보내고 시계열 데이터를 데이터 레이크에 더 쉽게 추가할 수 있습니다. Amazon Athena 및 Amazon Redshift와 같은 서비스를 사용하여 시계열 데이터를 다른 관련 데이터와 결합하여 복잡한 비즈니스 인사이트를 도출할 수 있습니다.
- AI 및 ML 데이터 파이프라인 구축 - 이 UNLOAD 문을 사용하면 시계열 데이터에 액세스하는 기계 학습 모델의 데이터 파이프라인을 쉽게 구축할 수 있으므로 Amazon SageMaker 및 Amazon과 같은 서비스와 시계열 데이터를 더 쉽게 사용할 수 있습니다EMR.
- ETL 처리 간소화 - 데이터를 S3 버킷으로 내보내면 데이터에 대한 추출, 변환, 로드(ETL) 작업을 수행하는 프로세스를 간소화하여 AWS Glue와 같은 타사 도구 또는 AWS 서비스를 원활하게 사용하여 데이터를 처리하고 변환할 수 있습니다.

UNLOAD 개념

구문

```
UNLOAD (SELECT statement)
 TO 's3://bucket-name/folder'
```

```
 WITH ( option = expression [, ...] )
```

option 는 입니다.

```
{ partitioned_by = ARRAY[ col_name[,…] ]
 | format = [ '{ CSV | PARQUET }' ]
 | compression = [ '{ GZIP | NONE }' ]
 | encryption = [ '{ SSE_KMS | SSE_S3 }' ]
 | kms_key = '<string>'
 | field_delimiter ='<character>'
 | escaped_by = '<character>'
 | include_header = ['{true, false}']
 | max_file_size = '<value>'
 | }
```

파라미터

SELECT 문

LiveAnalytics 테이블에 대한 하나 이상의 Timestream에서 데이터를 선택하고 검색하는 데 사용되는 쿼리 문입니다.

```
(SELECT column 1, column 2, column 3 from database.table
      where measure_name = "ABC" and timestamp between ago (1d) and now() )
```

TO 절

```
TO 's3://bucket-name/folder'
```

또는

```
TO 's3://access-point-alias/folder'
```

UNLOAD 문에 있는 TO 절은 쿼리 결과의 출력 대상을 지정합니다. Timestream for LiveAnalytics 가 출력 파일 객체를 쓰는 Amazon S3 버킷 이름 또는 Amazon S3 access-point-alias에 폴더 위치가 있는 Amazon S3를 포함한 전체 경로를 제공해야 합니다. S3 버킷은 동일한 계정과 동일한 리전에서 소유해야 합니다. 쿼리 결과 세트 외에도 Timestream for 는 매니페스트 및 메타데이터 파일을 지정된 대상 폴더에 LiveAnalytics 씁니다.

PARTITIONED_BY 절

```
partitioned_by = ARRAY [col_name[,…] , (default: none)
```

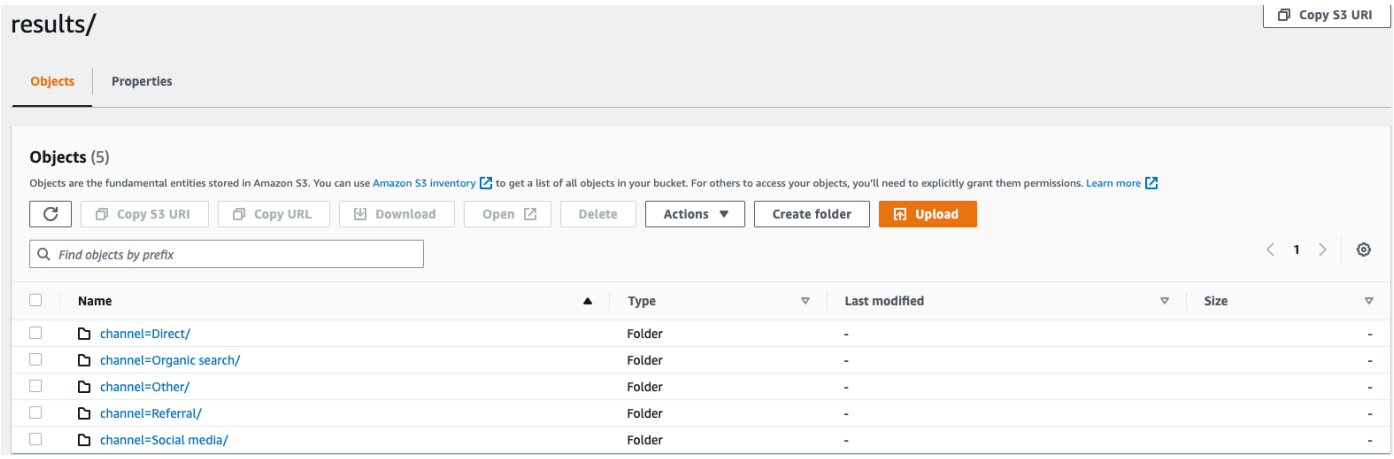
이 partitioned_by 절은 쿼리에서 세분화된 수준에서 데이터를 그룹화하고 분석하는 데 사용됩니다. 쿼리 결과를 S3 버킷으로 내보낼 때 선택 쿼리의 하나 이상의 열을 기반으로 데이터를 분할하도록 선택할 수 있습니다. 데이터를 분할할 때 내보낸 데이터는 파티션 열을 기반으로 하위 집합으로 분할되고 각 하위 집합은 별도의 폴더에 저장됩니다. 내보낸 데이터가 포함된 결과 폴더 내에 하위 폴더folder/results/partition column = partition value/가 자동으로 생성됩니다. 하지만 파티션된 열은 출력 파일에 포함되지 않습니다.

partitioned_by 는 구문의 필수 절이 아닙니다. 파티셔닝 없이 데이터를 내보내도록 선택한 경우 구문에서 절을 제외할 수 있습니다.

Example

웹 사이트의 클릭스트림 데이터를 모니터링하고 direct, , , , 라는 5개의 트래픽 채널이 있다고 가정합니다Social MediaOrganic SearchOtherReferral. 데이터를 내보낼 때 열을 사용하여 데이터를 분할하도록 선택할 수 있습니다Channel. 데이터 폴더인 s3://bucketname/results에는 각각 해당 채널 이름이 있는 폴더가 5개 있습니다. 예를 들어 이 폴더s3://bucketname/results/channel=Social Media/.에서는 Social Media 채널을 통해 웹 사이트에 도착한 모든 고객의 데이터를 찾을 수 있습니다. 마찬가지로 나머지 채널에 대한 다른 폴더도 있습니다.

채널 열로 분할된 내보낸 데이터



FORMAT

```
format = [ '{ CSV | PARQUET }' , default: CSV
```

S3 버킷에 기록된 쿼리 결과의 형식을 지정하는 키워드입니다. 쉼표(,CSV)를 기본 구분 기호로 사용하거나 분석을 위한 효율적인 오픈 열 스토리지 형식인 Apache Parquet 형식으로 데이터를 쉼표로 구분된 값()으로 내보낼 수 있습니다.

COMPRESSION

```
compression = [ '{ GZIP | NONE }' ], default: GZIP
```

압축 알고리즘을 사용하여 내보낸 데이터를 압축GZIP하거나 NONE 옵션을 지정하여 압축을 해제할 수 있습니다.

ENCRYPTION

```
encryption = [ '{ SSE_KMS | SSE_S3 }' ], default: SSE_S3
```

Amazon S3의 출력 파일은 선택한 암호화 옵션을 사용하여 암호화됩니다. 데이터 외에도 매니페스트 및 메타데이터 파일도 선택한 암호화 옵션에 따라 암호화됩니다. 현재 SSE_S3 및 SSE_KMS 암호화를 지원합니다. SSE_S3는 Amazon S3가 256비트 고급 암호화 표준(AES) 암호화를 사용하여 데이터를 암호화하는 서버 측 암호화입니다. SSE_KMS는 고객 관리형 키를 사용하여 데이터를 암호화하는 서버 측 암호화입니다.

KMS_KEY

```
kms_key = '<string>'
```

KMS 키는 내보낸 쿼리 결과를 암호화하는 고객 정의 키입니다. KMS 키는 AWS Key Management Service(AWS KMS)에서 안전하게 관리되며 Amazon S3의 데이터 파일을 암호화하는 데 사용됩니다.

FIELD_DELIMITER

```
field_delimiter ='<character>' , default: (,)
```

데이터를 CSV 형식으로 내보낼 때 이 필드는 파이프 ASCII 문자(|), 쉼표(,) 또는 탭(/t)과 같이 출력 파일의 필드를 구분하는 데 사용되는 단일 문자를 지정합니다. CSV 파일의 기본 구분 기호는 쉼표입니다. 데이터의 값에 선택한 구분 기호가 포함된 경우 구분 기호는 따옴표 문자로 따옴표로 표시됩니다. 예를 들어 데이터의 값에 가 포함된 경우 Time,stream이 값은 내보낸 데이터"Time,stream"에서와 같이 인용됩니다. 에 대해 Timestream에서 사용하는 따옴표 문자 LiveAnalytics 는 큰따옴표(')입니다.

에 헤더를 포함하려는 FIELD_DELIMITER 경우 캐리지 리턴 문자(ASCII 13, 16진0D수, 텍스트 '\r') 또는 줄 바꿈 문자(ASCII 10, 16진수 0A, 텍스트 '\n')를 로 지정하지 마세요. 이렇게 하면 많은 파서가 결과 CSV 출력에 헤더를 올바르게 구문 분석할 수 없기 CSV때문입니다.

ESCAPED_BY

```
escaped_by = '<character>', default: (\)
```

데이터를 CSV 형식으로 내보낼 때 이 필드는 S3 버킷에 기록된 데이터 파일에서 이스케이프 문자로 처리해야 하는 문자를 지정합니다. Escaping은 다음 시나리오에서 발생합니다.

1. 값 자체에 따옴표 문자(')가 포함된 경우 이스케이프 문자를 사용하여 이스케이프됩니다. 예를 들어 값이 이고 Time"stream(\)가 구성된 이스케이프 문자인 경우 값이 로 이스케이프됩니다Time\"stream.
2. 값에 구성된 이스케이프 문자가 포함된 경우 이스케이프됩니다. 예를 들어 값이 인 경우 값이 로 이스케이프Time\stream됩니다Time\\stream.

> **Note**
>
> 내보낸 출력에 배열, 행 또는 시계열과 같은 복잡한 데이터 유형이 포함된 경우 JSON 문자열로 직렬화됩니다. 다음은 한 예입니다.

데이터 유형	실제 값	값이 [직렬화된 JSON 문자열] CSV 형식으로 이스케이프되는 방법
배열	`[ 23,24,25 ]`	`"[23,24,25]"`
열	`( x=23.0, y=hello )`	`"{\"x\":23.0,\"y\":\"hello\"}"`
시계열	`[ ( time=1970-01-01 00:00:00.000000010, value=100.0 ), ( time=1970-01-01 00:00:00.000000012, value=120.0 ) ]`	`"[{\"time\":\"1970-01-01 00:00:00.000000010Z\",\"value\":100.0},{\"time\":\"1970-01-01 00:00:00.000000012`

데이터 유형	실제 값	값이 [직렬화된 JSON 문자열] CSV 형식으로 이스케이프되는 방법
		Z\",\"value\":120.0}]"

INCLUDE_HEADER

```
include_header = 'true' , default: 'false'
```

데이터를 CSV 형식으로 내보낼 때 이 필드를 사용하면 내보낸 CSV 데이터 파일의 첫 번째 행에 열 이름을 포함할 수 있습니다.

허용되는 값은 'true'와 'false'이고 기본값은 'false'입니다. escaped_by 및 와 같은 텍스트 변환 옵션은 헤더에도 field_delimiter 적용됩니다.

> **Note**
>
> 헤더를 포함할 때는 캐리지 리턴 문자(ASCII 13, 16진수 0D, 텍스트 '\r') 또는 줄 바꿈 문자(ASCII 10, 16진수 0A, 텍스트 '\n')를 로 선택하지 않는 것이 중요합니다. 이렇게 하면 많은 파서가 결과 CSV 출력에서 헤더를 올바르게 구문 분석할 수 없기 FIELD_DELIMITER때문입니다.

MAX_FILE_SIZE

```
max_file_size = 'X[MB|GB]' , default: '78GB'
```

이 필드는 UNLOAD 명령문이 Amazon S3에서 생성하는 파일의 최대 크기를 지정합니다. UNLOAD 문은 여러 파일을 생성할 수 있지만 Amazon S3에 작성된 각 파일의 최대 크기는 이 필드에 지정된 것과 비슷합니다.

필드 값은 16MB~78GB여야 합니다. 와 같은 정수로 지정12GB하거나 또는 와 같은 소수로 지정할 수 있습니다0.5GB24.7MB. 기본값은 78GB입니다.

실제 파일 크기는 파일을 작성할 때 근사치이므로 실제 최대 크기는 지정한 수와 정확히 같지 않을 수 있습니다.

S3 버킷에는 무엇이 기록되나요?

성공적으로 실행된 모든 UNLOAD 쿼리에 대해 Timestream for 는 쿼리 결과, 메타데이터 파일 및 매니페스트 파일을 S3 버킷에 LiveAnalytics 기록합니다. 데이터를 파티셔닝한 경우 결과 폴더에 모든 파티션 폴더가 있습니다. 매니페스트 파일에는 UNLOAD 명령에 의해 작성된 파일 목록이 포함되어 있습니다. 메타데이터 파일에는 작성된 데이터의 특성, 속성 및 속성을 설명하는 정보가 포함되어 있습니다.

내보낸 파일 이름은 무엇입니까?

내보낸 파일 이름에는 두 가지 구성 요소가 포함되어 있습니다. 첫 번째 구성 요소는 queryID이고 두 번째 구성 요소는 고유 식별자입니다.

CSV 파일

```
S3://bucket_name/results/<queryid>_<UUID>.csv
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.csv
```

압축 CSV 파일

```
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.gz
```

Parquet 파일

```
S3://bucket_name/results/<partitioncolumn>=<partitionvalue>/<queryid>_<UUID>.parquet
```

메타데이터 및 매니페스트 파일

```
S3://bucket_name/<queryid>_<UUID>_manifest.json
S3://bucket_name/<queryid>_<UUID>_metadata.json
```

CSV 형식의 데이터는 파일 수준에서 저장되므로 S3로 내보낼 때 데이터를 압축하면 파일에 “.gz” 확장자가 있습니다. 그러나 Parquet의 데이터는 열 수준에서 압축되므로 내보내는 동안 데이터를 압축하더라도 파일에는 여전히 .parquet 확장자가 있습니다.

각 파일에 포함된 정보는 무엇입니까?

매니페스트 파일

매니페스트 파일은 UNLOAD 실행으로 내보내는 파일 목록에 대한 정보를 제공합니다. 매니페스트 파일은 파일 이름이 인 제공된 S3 버킷에서 사용할 수 있습니다`s3://<bucket_name>/`

<queryid>_<UUID>_manifest.json. 매니페스트 파일에는 결과 폴더의 파일 URL, 레코드 수 및 각 파일의 크기, 쿼리 메타데이터(쿼리에 대해 S3로 내보낸 총 바이트 및 총 행)가 포함됩니다.

```
{
  "result_files": [
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 32295,
                "row_count": 10
            }
    },
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 62295,
                "row_count": 20
            }
    },
  ],
  "query_metadata":
    {
      "content_length_in_bytes": 94590,
      "total_row_count": 30,
      "result_format": "CSV",
      "result_version": "Amazon Timestream version 1.0.0"
    },
  "author": {
        "name": "Amazon Timestream",
        "manifest_file_version": "1.0"
  }
}
```

메타데이터

메타데이터 파일은 열 이름, 열 유형 및 스키마와 같은 데이터 세트에 대한 추가 정보를 제공합니다. 메타데이터 파일은 파일 이름이 S3://bucket_name/<queryid>_<UUID>_metadata.json인 제공된 S3 버킷에서 사용할 수 있습니다.

다음은 메타데이터 파일의 예입니다.

```
{
    "ColumnInfo": [
        {
            "Name": "hostname",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "region",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "measure_name",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "cpu_utilization",
            "Type": {
                "TimeSeriesMeasureValueColumnInfo": {
                    "Type": {
                        "ScalarType": "DOUBLE"
                    }
                }
            }
        }
  ],
  "Author": {
        "Name": "Amazon Timestream",
        "MetadataFileVersion": "1.0"
  }
}
```

메타데이터 파일에 공유된 열 정보는 SELECT 쿼리에 대한 쿼리 API 응답에서 ColumnInfo 전송된 것과 동일한 구조를 갖습니다.

결과

결과 폴더에는 내보낸 데이터가 Apache Parquet 또는 CSV 형식으로 포함되어 있습니다.

예

UNLOAD 쿼리 를 통해 아래와 같은 쿼리를 제출하면 API

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time, query,
 quantity, product_id, channel
                    FROM sample_clickstream.sample_shopping WHERE time BETWEEN ago(2d)
 AND now())
                TO 's3://my_timestream_unloads/withoutpartition/' WITH ( format='CSV',
 compression='GZIP')
```

UNLOAD 쿼리 응답에는 행 1개 * 열 3개가 있습니다. 이 3개의 열은 다음과 같습니다.

- 유형의 행 BIGINT - 내보낸 행 수를 나타냄
- metadataFile 의 유형 VARCHAR - 내보낸 메타데이터 파일의 S3입니다URI.
- manifestFile 유형 VARCHAR - 내보낸 URI 매니페스트 파일의 S3

쿼리 에서 다음 응답을 받게 됩니다API.

```
{
    "Rows": [
        {
            "Data": [
                {
                    "ScalarValue": "20" # No of rows in output across all files
                },
                {
                    "ScalarValue": "s3://my_timestream_unloads/withoutpartition/
AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY_<UUID>_metadata.json"
 #Metadata file
                },
                {
                    "ScalarValue": "s3://my_timestream_unloads/withoutpartition/
AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY_<UUID>_manifest.json"
 #Manifest file
                }
            ]
```

```
        }
    ],
    "ColumnInfo": [
        {
            "Name": "rows",
            "Type": {
                "ScalarType": "BIGINT"
            }
        },
        {
            "Name": "metadataFile",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "manifestFile",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        }
    ],
    "QueryId": "AEDAAANGH3D7FYHOBQGQQMEAISCJ45B42OWWJMOT4N6RRJICZUA7R25VYVOHJIY",
    "QueryStatus": {
        "ProgressPercentage": 100.0,
        "CumulativeBytesScanned": 1000,
        "CumulativeBytesMetered": 10000000
    }
}
```

데이터 타입

UNLOAD 문은 `time` 및 를 지원되는 데이터 유형 제외하고 에 설명된 LiveAnalytics의 쿼리 언어에 대한 Timestream의 모든 데이터 유형을 지원합니다unknown.

에 대한 TimestreamUNLOAD의 에 대한 사전 조건 LiveAnalytics

다음은 용 TimestreamUNLOAD에서 를 사용하여 S3에 데이터를 쓰기 위한 사전 조건입니다 LiveAnalytics.

- UNLOAD 명령에 LiveAnalytics 테이블(들)을 사용하려면 Timestream에서 데이터를 읽을 권한이 있어야 합니다.

- 리소스에 대한 LiveAnalytics Timestream과 동일한 AWS 리전에 Amazon S3 버킷이 있어야 합니다.
- 선택한 S3 버킷의 경우 S3 버킷 정책에 의 Timestream이 데이터를 내보낼 수 LiveAnalytics 있는 권한도 있는지 확인합니다.
- UNLOAD 쿼리를 실행하는 데 사용되는 자격 증명에는 Timestream이 S3 LiveAnalytics 에 데이터를 쓸 수 있도록 허용하는 필요한 AWS 자격 증명 및 액세스 관리(IAM) 권한이 있어야 합니다. 예제 정책은 다음과 같습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [{
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:ListMeasures",
                "timestream:WriteRecords",
                "timestream:Unload"
            ],
            "Resource": "arn:aws:timestream:<region>:<account_id>:database/
<database_name>/table/<table_name>"
        },
        {
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketAcl",
                "s3:PutObject",
                "s3:GetObjectMetadata",
                "s3:AbortMultipartUpload"
            ],
            "Resource": [
                "arn:aws:s3:::<S3_Bucket_Created>",
                "arn:aws:s3:::<S3_Bucket_Created>/*"
            ]
        }
    ]
}
```

이러한 S3 쓰기 권한에 대한 추가 컨텍스트는 Amazon Simple Storage Service 안내서를 참조하세요. 내보낸 데이터를 암호화하는 데 KMS 키를 사용하는 경우 필요한 추가 IAM 정책은 다음을 참조하세요.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Action": [
            "kms:DescribeKey",
            "kms:Decrypt",
            "kms:GenerateDataKey*"
        ],
        "Resource": "<account_id>-arn:aws:kms:<region>:<account_id>:key/*",
        "Condition": {
            "ForAnyValue:StringLike": {
                "kms:ResourceAliases": "alias/<Alias_For_Generated_Key>"
            }
        }
    }, {
        "Effect": "Allow",
        "Action": [
            "kms:CreateGrant"
        ],
        "Resource": "<account_id>-arn:aws:kms:<region>:<account_id>:key/*",
        "Condition": {
            "ForAnyValue:StringEquals": {
                "kms:EncryptionContextKeys": "aws:timestream:<database_name>"
            },
            "Bool": {
                "kms:GrantIsForAWSResource": true
            },
            "StringLike": {
                "kms:ViaService": "timestream.<region>.amazonaws.com"
            },
            "ForAnyValue:StringLike": {
                "kms:ResourceAliases": "alias/<Alias_For_Generated_Key>"
            }
        }
    }
]
}
```

에 대한 TimestreamUNLOAD의 모범 사례 LiveAnalytics

다음은 UNLOAD 명령과 관련된 모범 사례입니다.

- UNLOAD 명령을 사용하여 S3 버킷으로 내보낼 수 있는 데이터의 양은 제한되지 않습니다. 그러나 쿼리는 60분 후에 시간이 초과되므로 단일 쿼리로 60GB 이하의 데이터를 내보내는 것이 좋습니다. 60GB 이상의 데이터를 내보내야 하는 경우 작업을 여러 쿼리로 분할합니다.
- 데이터를 업로드하기 위해 S3에 수천 개의 요청을 보낼 수 있지만 쓰기 작업을 여러 S3 접두사에 병렬화하는 것이 좋습니다. 여기에서 설명서를 참조하세요. 여러 리더/라이터가 동일한 폴더에 액세스할 때 S3 API 통화 속도가 제한될 수 있습니다.
- 접두사를 정의하기 위한 S3 키 길이 제한을 고려할 때 특히 `partitioned_by` 절을 사용할 때는 버킷 및 폴더 이름을 10~15자 이내로 사용하는 것이 좋습니다.
- UNLOAD 문이 포함된 쿼리에 대해 4XX 또는 5XX를 받으면 S3 버킷에 부분 결과가 기록될 수 있습니다. 의 Timestream LiveAnalytics 은 버킷에서 데이터를 삭제하지 않습니다. S3 대상이 동일한 다른 UNLOAD 쿼리를 실행하기 전에 실패한 쿼리로 생성된 파일을 수동으로 삭제하는 것이 좋습니다. 해당 를 사용하여 실패한 쿼리로 작성된 파일을 식별할 수 있습니다`QueryExecutionId`. 실패한 쿼리의 경우 Timestream for LiveAnalytics 는 매니페스트 파일을 S3 버킷으로 내보내지 않습니다.
- 의 Timestream LiveAnalytics 은 다중 파트 업로드를 사용하여 쿼리 결과를 S3로 내보냅니다. UNLOAD 문이 포함된 쿼리에 LiveAnalytics 대해 Timestream에서 4XX 또는 5XX를 받으면 Timestream for LiveAnalytics 는 다중 파트 업로드를 최대한 중단하지만 일부 불완전한 부분이 남아 있을 수 있습니다. 따라서 여기 지침에 따라 S3 버킷에서 불완전한 다중 파트 업로드의 자동 정리를 설정하는 것이 좋습니다.

CSV 파서를 사용하여 CSV 형식의 데이터에 액세스하는 권장 사항

- CSV 파서는 구분 기호, 이스케이프 및 따옴표 문자가 동일한 문자를 갖도록 허용하지 않습니다.
- 일부 CSV 파서는 배열과 같은 복잡한 데이터 유형을 해석할 수 없습니다. JSON 디시리얼라이저를 통해 해석하는 것이 좋습니다.

Parquet 형식의 데이터에 액세스하기 위한 권장 사항

1. 사용 사례에서 스키마를 열 이름으로 UTF8자 지원해야 하는 경우 Parquet-mr 라이브러리를 사용하는 것이 좋습니다.
2. 결과의 타임스탬프는 12바이트 정수(INT96)로 표시됩니다.
3. 시계열은 로 표시되며`array<row<time, value>>`, 다른 중첩 구조는 Parquet 형식으로 지원되는 해당 데이터 유형을 사용합니다.

partition_by 절 사용

- partitioned_by 필드에 사용되는 열은 선택 쿼리의 마지막 열이어야 합니다. partitioned_by 필드에 두 개 이상의 열을 사용하는 경우 열은 선택 쿼리의 마지막 열이어야 하며 partition_by 필드에 사용된 것과 동일한 순서로 열이 되어야 합니다.
- 데이터(partitioned_by 필드)를 분할하는 데 사용되는 열 값에는 ASCII 문자만 포함할 수 있습니다. 에 대한 Timestream은 값에서 UTF-8자를 LiveAnalytics 허용하지만 S3는 객체 키로 ASCII 문자만 지원합니다.

에 대한 TimestreamUNLOAD의 에 대한 사용 사례 예 LiveAnalytics

전자 상거래 웹 사이트의 사용자 세션 지표, 트래픽 소스 및 제품 구매를 모니터링하고 있다고 가정합니다. 용 Timestream LiveAnalytics 을 사용하여 사용자 행동, 제품 판매에 대한 실시간 인사이트를 도출하고 고객을 웹 사이트로 유도하는 트래픽 채널(유기 검색, 소셜 미디어, 직접 트래픽, 유료 캠페인 등)에 대한 마케팅 분석을 수행합니다.

주제

파티션 없이 데이터 내보내기

데이터의 마지막 이틀을 CSV 형식으로 내보내려고 합니다.

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time,
query, quantity, product_id, channel
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/withoutpartition'
WITH (  format='CSV',
compression='GZIP')
```

채널별 데이터 분할

지난 2일 동안의 데이터를 CSV 형식으로 내보내고 싶지만 각 트래픽 채널의 데이터를 별도의 폴더에 저장하려고 합니다. 이렇게 하려면 다음과 같이 channel 열을 사용하여 데이터를 분할해야 합니다.

```
UNLOAD(SELECT user_id, ip_address, event, session_id, measure_name, time,
query, quantity, product_id, channel
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbychannel/'
WITH (
partitioned_by = ARRAY ['channel'],
format='CSV',
compression='GZIP')
```

이벤트별 데이터 분할

지난 2일 동안의 데이터를 CSV 형식으로 내보내고 싶지만 각 이벤트의 데이터를 별도의 폴더에 저장하려고 합니다. 이렇게 하려면 다음과 같이 event 열을 사용하여 데이터를 분할해야 합니다.

```
UNLOAD(SELECT user_id, ip_address, channel, session_id, measure_name, time,
query, quantity, product_id, event
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbyevent/'
WITH (
partitioned_by = ARRAY ['event'],
format='CSV',
compression='GZIP')
```

채널 및 이벤트별 데이터 분할

지난 2일간의 데이터를 CSV 형식으로 내보내려고 하지만 각 채널 및 채널 내 각 이벤트에 대한 데이터를 별도의 폴더에 저장하려고 합니다. 이렇게 하려면 다음 그림과 같이 channel 및 event 열을 모두 사용하여 데이터를 분할해야 합니다.

```
UNLOAD(SELECT user_id, ip_address, session_id, measure_name, time,
query, quantity, product_id, channel,event
FROM sample_clickstream.sample_shopping
WHERE time BETWEEN ago(2d) AND now())
TO 's3://<bucket_name>/partitionbychannelevent/'
```

```
WITH (
partitioned_by = ARRAY ['channel','event'],
format='CSV',
compression='GZIP')
```

매니페스트 및 메타데이터 파일

매니페스트 파일

매니페스트 파일은 UNLOAD 실행으로 내보내는 파일 목록에 대한 정보를 제공합니다. 매니페스트 파일은 파일 이름이 인 제공된 S3 버킷에서 사용할 수 있습니다`S3://bucket_name/<queryid>_<UUID>_manifest.json`. 매니페스트 파일에는 결과 폴더의 파일 URL, 레코드 수 및 각 파일의 크기, 쿼리 메타데이터(쿼리에 대해 S3로 내보낸 총 바이트 및 총 행)가 포함됩니다.

```
{
  "result_files": [
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 32295,
                "row_count": 10
            }
    },
    {
        "url":"s3://my_timestream_unloads/ec2_metrics/
AEDAGANLHLBH4OLISD3CVOZZRWPX5GV2XCXRBKCVD554N6GWPWWXBP7LSG74V2Q_1448466917_szCL4YgVYzGXj2lS.gz"
        "file_metadata":
            {
                "content_length_in_bytes": 62295,
                "row_count": 20
            }
    },
  ],
  "query_metadata":
    {
      "content_length_in_bytes": 94590,
      "total_row_count": 30,
      "result_format": "CSV",
      "result_version": "Amazon Timestream version 1.0.0"
    },
  "author": {
```

```
        "name": "Amazon Timestream",
        "manifest_file_version": "1.0"
  }
}
```

메타데이터

메타데이터 파일은 열 이름, 열 유형 및 스키마와 같은 데이터 세트에 대한 추가 정보를 제공합니다. 메타데이터 파일은 파일 이름이 S3://bucket_name/<queryid>_<UUID>_metadata.json인 제공된 S3 버킷에서 사용할 수 있습니다.

다음은 메타데이터 파일의 예입니다.

```
{
    "ColumnInfo": [
        {
            "Name": "hostname",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "region",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "measure_name",
            "Type": {
                "ScalarType": "VARCHAR"
            }
        },
        {
            "Name": "cpu_utilization",
            "Type": {
                "TimeSeriesMeasureValueColumnInfo": {
                    "Type": {
                        "ScalarType": "DOUBLE"
                    }
                }
            }
        }
```

```
  ],
  "Author": {
        "Name": "Amazon Timestream",
        "MetadataFileVersion": "1.0"
  }
}
```

메타데이터 파일에 공유된 열 정보는 SELECT 쿼리에 대한 쿼리 API 응답에서 ColumnInfo 전송된 것과 동일한 구조를 갖습니다.

Glue 크롤러를 사용하여 Glue 데이터 카탈로그 구축

1. 다음 검증을 위해 관리자 자격 증명으로 계정에 로그인합니다.
2. 여기에 제공된 지침을 사용하여 Glue 데이터베이스용 크롤러를 생성합니다. 데이터 소스에 제공할 S3 폴더는 와 같은 UNLOAD 결과 폴더여야 합니다s3://my_timestream_unloads/results.
3. 여기 지침에 따라 https://docs.aws.amazon.com/glue/latest/ug/tutorial-add-crawler.html#tutorial-add-crawler-step2크롤러를 실행합니다.
4. Glue 테이블을 봅니다.
   - AWS Glue → Tables로 이동합니다.
   - 크롤러를 생성하는 동안 제공된 테이블 접두사로 생성된 새 테이블이 표시됩니다.
   - 테이블 세부 정보 보기를 클릭하여 스키마 및 파티션 정보를 볼 수 있습니다.

다음은 Glue 데이터 카탈로그를 AWS 사용하는 다른 AWS 서비스 및 오픈 소스 프로젝트입니다.

- Amazon Athena - 자세한 내용은 Amazon Athena 사용 설명서의 테이블, 데이터베이스 및 데이터 카탈로그 이해를 참조하세요.
- Amazon Redshift Spectrum - 자세한 내용은 Amazon Redshift 데이터베이스 개발자 안내서의 Amazon Redshift Spectrum을 사용하여 외부 데이터 쿼리를 참조하세요.
- Amazon EMR - 자세한 내용은 Amazon 관리 안내서의 AWS Glue 데이터 카탈로그에 대한 Amazon EMR 액세스에 리소스 기반 정책 사용을 참조하세요. EMR
- AWS Apache Hive 메타스토어용 Glue Data Catalog 클라이언트 - 이 GitHub 프로젝트에 대한 자세한 내용은 AWS Apache Hive 메타스토어용 Glue Data Catalog 클라이언트를 참조하세요.

의 TimestreamUNLOAD에서 에 대한 제한 LiveAnalytics

다음은 UNLOAD 명령과 관련된 제한입니다.

- UNLOAD 문을 사용한 쿼리의 동시성은 초당 쿼리 1개입니다(QPS). 쿼리 속도를 초과하면 제한이 발생할 수 있습니다.
- UNLOAD 문이 포함된 쿼리는 쿼리당 최대 100개의 파티션을 내보낼 수 있습니다. 내보낸 데이터를 분할하는 데 사용하기 전에 선택한 열의 고유 개수를 확인하는 것이 좋습니다.
- 60분 후 UNLOAD 문 제한 시간이 포함된 쿼리입니다.
- UNLOAD 명령문이 Amazon S3에서 생성하는 파일의 최대 크기는 78GB입니다.

의 Timestream에 대한 기타 제한은 섹션을 LiveAnalytics참조하세요. 할당량

Amazon Timestream에서 쿼리 인사이트를 사용하여 쿼리 최적화

쿼리 인사이트는 쿼리를 최적화하고, 성능을 개선하고, 비용을 절감하는 데 도움이 되는 성능 튜닝 기능입니다. 쿼리 인사이트를 통해 쿼리의 시간, 시간 및 공간 파티션 키 기반 정리 효율성을 평가할 수 있습니다. 쿼리 인사이트를 사용하여 개선이 필요한 영역을 식별하여 쿼리 성능을 향상시킬 수도 있습니다. 또한 쿼리 인사이트를 통해 쿼리가 시간 기반 및 파티션 키 기반 인덱싱을 얼마나 효과적으로 사용하여 데이터 검색을 최적화하는지 평가할 수 있습니다. 쿼리 성능을 최적화하려면 쿼리 실행을 제어하는 시간 파라미터와 공간 파라미터를 모두 미세 조정해야 합니다.

주제

- 쿼리 인사이트의 이점
- Amazon Timestream에서 데이터 액세스 최적화
- Amazon Timestream에서 쿼리 인사이트 활성화
- 쿼리 인사이트 응답을 사용하여 쿼리 최적화

쿼리 인사이트의 이점

다음은 쿼리 인사이트를 사용할 때 얻을 수 있는 주요 이점입니다.

- 비효율적인 쿼리 식별 - 쿼리 인사이트는 쿼리에서 액세스하는 테이블의 시간 기반 및 속성 기반 정리에 대한 정보를 제공합니다. 이 정보는 최적이 아닌 방식으로 액세스되는 테이블을 식별하는 데 도움이 됩니다.
- 데이터 모델 및 파티셔닝 최적화 - 쿼리 인사이트 정보를 사용하여 데이터 모델 및 파티셔닝 전략에 액세스하고 미세 조정할 수 있습니다.
- 쿼리 조정 - 쿼리 인사이트는 인덱스를 더 효과적으로 사용할 수 있는 기회를 강조합니다.

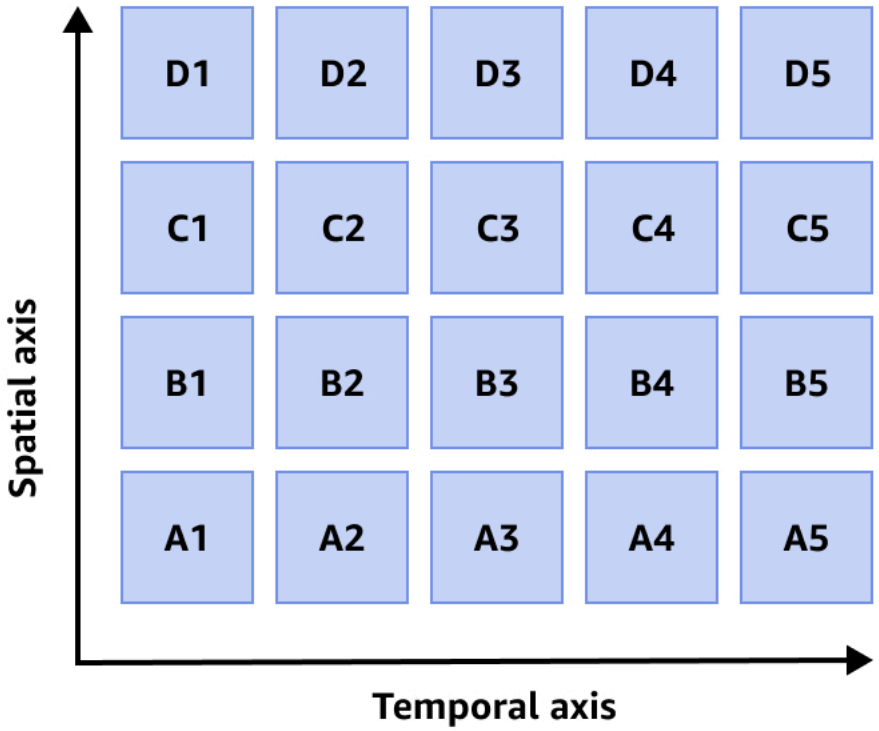
Amazon Timestream에서 데이터 액세스 최적화

Timestream 파티셔닝 체계 또는 데이터 조직 기법을 사용하여 Amazon Timestream의 데이터 액세스 패턴을 최적화할 수 있습니다.

주제

타임스트림 파티셔닝 체계

Amazon Timestream은 각 Timestream 테이블에 수백, 수천 또는 수백만 개의 독립 파티션이 있을 수 있는 확장성이 뛰어난 파티셔닝 체계를 사용합니다. 고가용성 파티션 추적 및 인덱싱 서비스는 파티셔닝을 관리하여 장애의 영향을 최소화하고 시스템의 복원력을 높입니다.



데이터 조직

Timestream은 수집한 각 데이터 포인트를 단일 파티션에 저장합니다. Timestream 테이블에 데이터를 수집하면 Timestream은 데이터의 타임스탬프, 파티션 키 및 기타 컨텍스트 속성을 기반으로 파티션을 자동으로 생성합니다. Timestream은 정시에 데이터를 파티셔닝(임시 파티셔닝)하는 것 외에도 선택한 파티셔닝 키 및 기타 차원(공간 파티셔닝)을 기반으로 데이터를 파티셔닝합니다. 이 접근 방식은 쓰기 트래픽을 분산하고 쿼리를 위한 데이터를 효과적으로 정리할 수 있도록 설계되었습니다.

쿼리 인사이트 기능은 쿼리 공간 적용 범위 및 쿼리 시간 적용 범위를 포함하여 쿼리의 정리 효율성에 대한 귀중한 인사이트를 제공합니다.

주제

- QuerySpatialCoverage
- QueryTemporalCoverage

QuerySpatialCoverage

QuerySpatialCoverage 지표는 실행된 쿼리의 공간 범위와 가장 비효율적인 공간 정리가 있는 테이블에 대한 인사이트를 제공합니다. 이 정보는 파티셔닝 전략의 개선 영역을 식별하여 공간 정리를 개선하는 데 도움이 될 수 있습니다. 지표의 값은 `QuerySpatialCoverage` 0에서 1 사이입니다. 지표 값이 낮을수록 공간 축에서 쿼리 정리가 더 최적화됩니다. 예를 들어 값이 0.1이면 쿼리가 공간 축의 10%를 스캔함을 나타냅니다. 값이 1이면 쿼리가 공간 축의 100%를 스캔함을 나타냅니다.

Example 쿼리 인사이트를 사용하여 쿼리의 공간 범위 분석

날씨 데이터를 저장하는 Timestream 데이터베이스가 있다고 가정해 보겠습니다. 미국 내 여러 주에 위치한 기상대에서 1시간마다 온도가 기록된다고 가정합니다. 상태별로 데이터를 분할하기 위한 고객 정의 파티셔닝 키(CDPK)`State`로 를 선택한다고 가정해 보겠습니다.

쿼리를 실행하여 특정 요일 오후 2시에서 오후 4시 사이에 캘리포니아의 모든 기상 스테이션의 평균 온도를 검색한다고 가정해 보겠습니다. 다음 예제에서는 이 시나리오에 대한 쿼리를 보여줍니다.

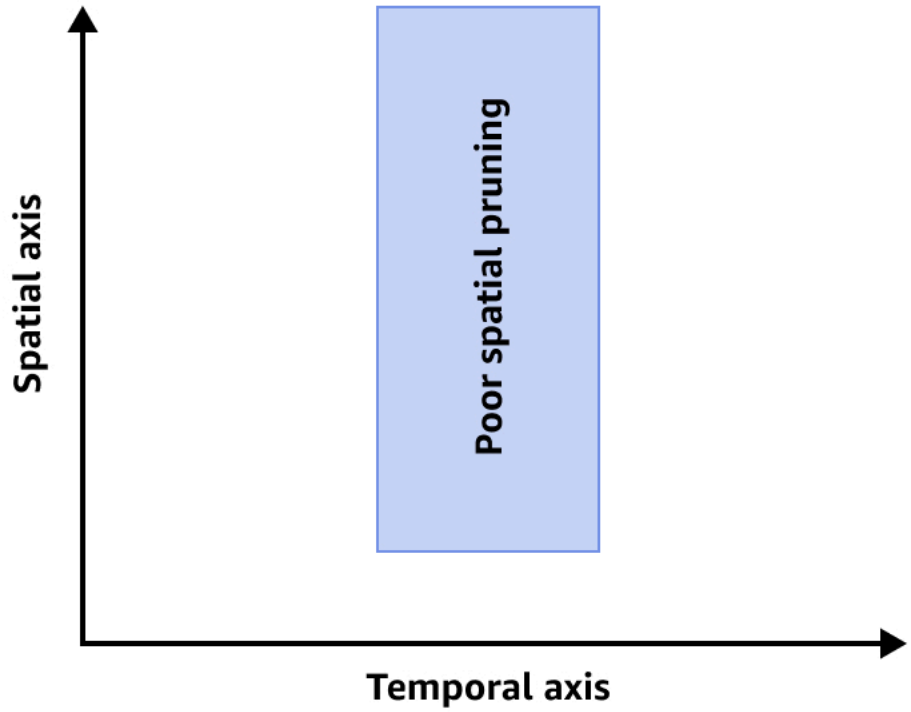
```
SELECT AVG(temperature)
FROM "weather_data"."hourly_weather"
WHERE time >= '2024-10-01 14:00:00' AND time < '2024-10-01 16:00:00'
  AND state = 'CA';
```

쿼리 인사이트 기능을 사용하여 쿼리의 공간 범위를 분석할 수 있습니다. 지표가 `QuerySpatialCoverage` 0.02의 값을 반환한다고 가정해 보겠습니다. 즉, 쿼리는 공간 축의 2%만 스

캔하므로 효율적입니다. 이 경우 쿼리는 캘리포니아에서 데이터를 검색하고 다른 주에서 데이터를 무시하기만 하여 공간 범위를 효과적으로 정리할 수 있었습니다.

반대로 지표가 QuerySpatialCoverage 0.8의 값을 반환하면 쿼리가 공간 축의 80%를 스캔하여 효율성이 떨어졌음을 나타냅니다. 이는 공간 정리를 개선하기 위해 파티셔닝 전략을 개선해야 함을 암시할 수 있습니다. 예를 들어 파티션 키를 상태 대신 도시 또는 리전으로 선택할 수 있습니다. QuerySpatialCoverage 지표를 분석하여 파티셔닝 전략을 최적화하고 쿼리 성능을 개선할 기회를 식별할 수 있습니다.

다음 이미지는 공간 정리 불량을 보여줍니다.



공간 정리 효율성을 개선하기 위해 다음 중 하나 또는 둘 다를 수행할 수 있습니다.

- 기본 구문 분석 키measure_name인 를 추가하거나 쿼리에서 CDPK 어휘를 사용합니다.
- 이전 포인트에서 언급한 속성을 이미 추가한 경우 와 같은 속성 또는 절과 관련된 함수를 제거합니다LIKE.

QueryTemporalCoverage

QueryTemporalCoverage 지표는 스캔된 시간 범위가 가장 큰 테이블을 포함하여 실행된 쿼리로 스캔된 시간 범위에 대한 인사이트를 제공합니다. QueryTemporalCoverage 지표의 값은 나노초로 표시되는 시간 범위입니다. 이 지표의 값이 낮을수록 시간 범위에서 쿼리 정리가 더 최적화됩니다. 예를 들어, 지난 몇 분 동안의 데이터 쿼리 스캔은 테이블의 전체 시간 범위를 스캔하는 쿼리보다 성능이 뛰어납니다.

Example

제조 공장에 있는 디바이스에서 매분 측정한 측정값과 함께 IoT 센서 데이터를 저장하는 Timestream 데이터베이스가 있다고 가정해 보겠습니다. 에서 데이터를 분할했다고 가정합니다device_ID.

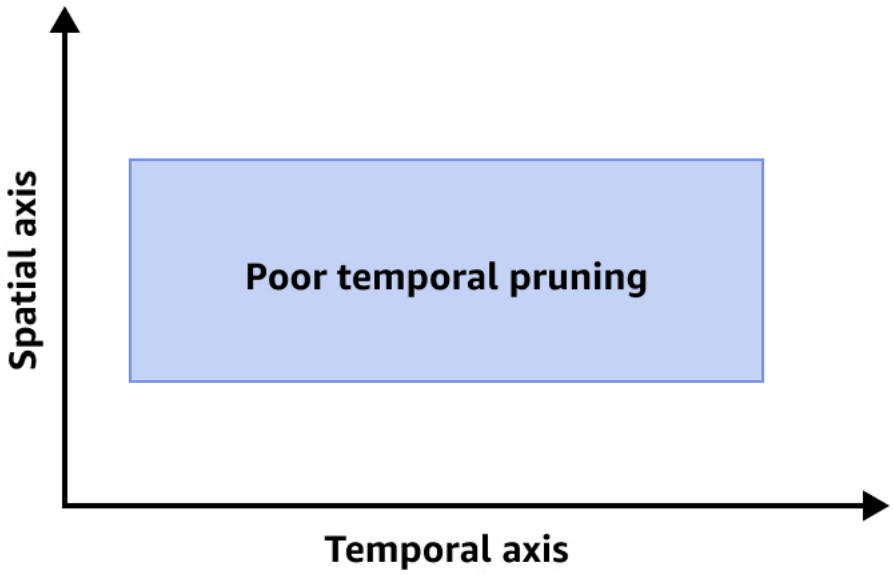
쿼리를 실행하여 지난 30분 동안 특정 디바이스의 평균 센서 판독값을 검색한다고 가정해 보겠습니다. 다음 예제에서는 이 시나리오에 대한 쿼리를 보여줍니다.

```
SELECT AVG(sensor_reading)
FROM "sensor_data"."factory_1"
WHERE device_id = 'DEV_123'
  AND time >= NOW() - INTERVAL 30 MINUTE and time < NOW();
```

쿼리 인사이트 기능을 사용하여 쿼리에서 스캔한 시간 범위를 분석할 수 있습니다. QueryTemporalCoverage 지표가 1800000000000나노초(30분)의 값을 반환한다고 가정해 보겠습니다. 즉, 쿼리는 최근 30분 동안의 데이터만 스캔했으며, 이는 상대적으로 좁은 시간 범위입니다. 이는 쿼리가 시간 파티셔닝을 효과적으로 조정하고 요청된 데이터만 검색했음을 나타내므로 좋은 신호입니다.

반대로 지표가 QueryTemporalCoverage 1년을 나노초 단위로 반환한 경우 쿼리가 테이블에서 1년 시간 범위를 스캔했음을 나타내며, 이는 효율성이 떨어집니다. 이는 쿼리가 시간 정리에 최적화되지 않았으며 시간 필터를 추가하여 쿼리를 개선할 수 있음을 암시할 수 있습니다.

다음 이미지는 시간 정리 불량을 보여줍니다.



시간 정리를 개선하려면 다음 중 하나 또는 모두를 수행하는 것이 좋습니다.

- 쿼리에 누락된 시간 예측을 추가하고 시간 예측이 원하는 시간 기간을 정리하고 있는지 확인합니다.
- 가 예측하는 시간 `MAX()`즈음에 와 같은 함수를 제거합니다.
- 모든 하위 쿼리에 시간을 추가합니다. 이는 하위 쿼리가 큰 테이블을 조인하거나 복잡한 작업을 수행하는 경우 중요합니다.

Amazon Timestream에서 쿼리 인사이트 활성화

쿼리 응답을 통해 직접 전달되는 인사이트를 사용하여 쿼리에 대한 쿼리 인사이트를 활성화할 수 있습니다. 쿼리 인사이트를 활성화해도 추가 인프라가 필요하지 않으며 추가 비용이 발생하지 않습니다. 쿼리 인사이트를 활성화하면 쿼리 응답의 일부로 쿼리 결과 외에도 쿼리 성능 관련 메타데이터 필드를 반환합니다. 이 정보를 사용하여 쿼리를 조정하여 쿼리 성능을 개선하고 쿼리 비용을 줄일 수 있습니다.

쿼리 인사이트 활성화에 대한 자세한 내용은 섹션을 참조하세요쿼리 실행.

쿼리 인사이트를 활성화하여 반환된 응답의 예를 보려면 예약된 쿼리의 예제를 참조하세요.

> **Note**
>
> - 쿼리 인사이트를 활성화하면 쿼리 속도가 초당 1개의 쿼리로 제한됩니다(QPS). 성능 영향을 방지하려면 쿼리를 프로덕션에 배포하기 전에 쿼리 평가 단계에서만 쿼리 인사이트를 활성화하는 것이 좋습니다.

- 쿼리 인사이트에 제공된 인사이트는 결국 일관적이므로 새 데이터가 테이블에 지속적으로 수집되면 변경될 수 있습니다.

쿼리 인사이트 응답을 사용하여 쿼리 최적화

에 Amazon Timestream을 사용하여 다양한 위치의 에너지 소비 LiveAnalytics 를 모니터링하고 있다고 가정해 보겠습니다. 데이터베이스에 `raw-metrics` 및 라는 두 개의 테이블이 있다고 가정해 보겠습니다`aggregate-metrics`.

이 `raw-metrics` 표는 디바이스 수준에서 세부 에너지 데이터를 저장하며 다음 열을 포함합니다.

- Timestamp
- 워싱턴과 같은 상태
- 디바이스 ID
- 에너지 소비

이 테이블의 데이터는 세분화된 minute-by-minute 상태로 수집 및 저장됩니다. 테이블은 를 `State`로 사용합니다CDPK.

이 `aggregate-metrics` 표는 예약된 쿼리의 결과를 저장하여 모든 디바이스의 에너지 소비 데이터를 매시간 집계합니다. 이 테이블에는 다음 열이 포함되어 있습니다.

- Timestamp
- 워싱턴과 같은 상태
- 총 에너지 소비

`aggregate-metrics` 테이블은 이 데이터를 시간별 세분화로 저장합니다. 테이블은 를 `State`로 사용합니다CDPK.

주제

- 지난 24시간 동안의 에너지 소비 쿼리
- 시간 범위에 대한 쿼리 최적화
- 공간 범위에 대한 쿼리 최적화
- 쿼리 성능 개선

지난 24시간 동안의 에너지 소비 쿼리

지난 24시간 동안 워싱턴에서 소비한 총 에너지를 추출하려고 한다고 가정해 보겠습니다. 이 데이터를 찾으려면 및 테이블의 강점을 모두 활용할 수 있습니다raw-metricsaggregate-metrics. 이 aggregate-metrics 표는 지난 23시간 동안의 시간별 에너지 소비 데이터를 제공하는 반면, raw-metrics 이 표는 지난 1시간 동안의 분 단위 데이터를 제공합니다. 두 테이블을 모두 쿼리하면 지난 24시간 동안 워싱턴의 에너지 소비를 완전하고 정확하게 파악할 수 있습니다.

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
 "metrics"."aggregate-metrics" am
 LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE rm.time >= ago(1h) and rm.time < now()
```

이 예제 쿼리는 설명 목적으로만 제공되며 그대로 작동하지 않을 수 있습니다. 개념을 시연하기 위한 것이지만 특정 사용 사례 또는 환경에 맞게 수정해야 할 수 있습니다.

이 쿼리를 실행한 후 쿼리 응답 시간이 예상보다 느려질 수 있습니다. 이 성능 문제의 근본 원인을 식별하려면 쿼리 인사이트 기능을 사용하여 쿼리의 성능을 분석하고 실행을 최적화할 수 있습니다.

다음 예제에서는 쿼리 인사이트 응답을 보여줍니다.

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 1.0,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/raw-metrics,
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
                    Max: {
                        Value:31540000000000000 //365 days,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

쿼리 인사이트 응답은 다음 정보를 제공합니다.

- 시간 범위 : 쿼리가 aggregate-metrics 테이블에 대해 과도한 365일 시간 범위를 스캔했습니다. 이는 시간 필터링을 비효율적으로 사용함을 나타냅니다.
- 공간 범위 : 쿼리가 raw-metrics 테이블의 전체 공간 범위(100%)를 스캔했습니다. 즉, 공간 필터링이 효과적으로 활용되지 않습니다.

쿼리가 둘 이상의 테이블에 액세스하는 경우 쿼리 인사이트는 가장 최적화되지 않은 액세스 패턴을 가진 테이블에 대한 지표를 제공합니다.

시간 범위에 대한 쿼리 최적화

쿼리 인사이트 응답을 기반으로 다음 예제와 같이 쿼리를 시간 범위에 맞게 최적화할 수 있습니다.

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
  "metrics"."aggregate-metrics" am
  LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE
  am.time >=  ago(23h) and am.time < now()
  AND rm.time >=  ago(1h) and rm.time < now()
  AND rm.state = 'Washington'
```

QueryInsights 명령을 다시 실행하면 다음 응답이 반환됩니다.

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 1.0,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics,
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
                    Max: {
                        Value: 82800000000000 //23 hours,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
```

```
                },
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

이 응답은 aggregate-metrics 테이블의 공간 범위가 여전히 100%이며 비효율적임을 보여줍니다. 다음 섹션에서는 공간 범위에 대한 쿼리를 최적화하는 방법을 보여줍니다.

공간 범위에 대한 쿼리 최적화

쿼리 인사이트 응답을 기반으로 다음 예제와 같이 공간 적용 범위에 대한 쿼리를 최적화할 수 있습니다.

```
SELECT am.time, am.state, am.total_energy_consumption,
rm.time, rm.state, rm.device_id, rm.energy_consumption
FROM
  "metrics"."aggregate-metrics" am
  LEFT JOIN "metrics"."raw-metrics" rm ON am.state = rm.state
WHERE
  am.time >=  ago(23h) and am.time < now()
  AND am.state ='Washington'
  AND rm.time >=  ago(1h) and rm.time < now()
  AND rm.state = 'Washington'
```

QueryInsights 명령을 다시 실행하면 다음 응답이 반환됩니다.

```
queryInsightsResponse={
                QuerySpatialCoverage: {
                    Max: {
                        Value: 0.02,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics,
                        PartitionKey: [State]
                    }
                },
                QueryTemporalRange: {
                    Max: {
                        Value: 82800000000000 //23 hours,
                        TableArn: arn:aws:timestream:us-east-1:123456789012:database/
metrics/table/aggregate-metrics
                    }
                },
```

```
                QueryTableCount: 2,
                OutputRows: 83,
                OutputBytes: 590
```

쿼리 성능 개선

쿼리를 최적화한 후 쿼리 인사이트는 다음 정보를 제공합니다.

- `aggregate-metrics` 테이블의 임시 정리는 23시간입니다. 이는 시간 범위의 23시간만 스캔됨을 나타냅니다.
- `aggregate-metrics` 테이블의 공간 정리는 0.02입니다. 이는 테이블의 공간 범위 데이터의 2%만 스캔되고 있음을 나타냅니다. 쿼리는 테이블의 매우 작은 부분을 스캔하여 빠른 성능과 리소스 사용률 감소를 가져옵니다. 정리 효율성이 개선되면 쿼리가 이제 성능에 최적화되었음을 나타냅니다.

작업 AWS Backup

용 Amazon Timestream의 데이터 보호 기능은 규정 준수 및 비즈니스 연속성 요구 사항을 충족하는 데 도움이 되는 LiveAnalytics 완전 관리형 솔루션입니다. 이 기능은 백업의 생성, 마이그레이션, 복원 및 삭제를 간소화하는 동시에 향상된 보고 및 감사를 제공하도록 설계된 통합 백업 서비스 AWS Backup 인 와의 기본 통합을 통해 활성화됩니다. 와의 통합을 통해 완전 관리형 정책 기반 중앙 집중식 데이터 보호 솔루션을 사용하여 변경할 AWS Backup수 없는 백업을 생성하고 Timestream 및 에서 지원하는 기타 AWS 서비스에 걸친 애플리케이션 데이터의 데이터 보호를 중앙에서 관리할 수 있습니다 AWS Backup.

기능을 사용하려면 가 Timestream 리소스를 AWS Backup 보호하도록 허용하도록 옵트인해야 합니다. 옵트인 선택 사항은 특정 계정 및 AWS 리전에 적용되므로 동일한 계정을 사용하여 여러 리전에 옵트인해야 할 수 있습니다. 백업에 대한 AWS 자세한 내용은 AWS Backup 개발자 안내서를 참조하세요.

에서 사용할 수 있는 데이터 보호 기능에는 다음이 AWS Backup 포함됩니다.

예약된 백업 - 백업 계획을 사용하여 LiveAnalytics 테이블에 대해 정기적으로 예약된 Timestream 백업을 설정할 수 있습니다.

교차 계정 및 교차 리전 복사 - 백업을 다른 AWS 리전 또는 계정의 다른 백업 저장소에 자동으로 복사할 수 있으므로 데이터 보호 요구 사항을 지원할 수 있습니다.

콜드 스토리지 계층화 - 백업을 구성하여 수명 주기 규칙을 구현하여 백업을 삭제하거나 콜드 스토리지로 전환할 수 있습니다. 이렇게 하면 백업 비용을 최적화할 수 있습니다.

태그 - 청구 및 비용 할당 목적으로 백업에 태그를 자동으로 지정할 수 있습니다.

암호화 - 백업 데이터가 볼트에 AWS Backup 저장됩니다. 이렇게 하면 LiveAnalytics 테이블 암호화 키에 대해 Timestream과 독립적인 키를 사용하여 백업을 AWS KMS 암호화하고 보호할 수 있습니다.

WORM 모델을 사용한 보안 백업 - Vault Lock을 사용하여 AWS Backup 백업에 write-once-read-many 대해 (WORM) 설정을 활성화할 수 있습니다. AWS Backup Vault Lock을 사용하면 우발적 또는 악의적인 삭제 작업, 백업 보존 기간 변경, 수명 주기 설정 업데이트로부터 백업을 보호하는 추가 방어 계층을 추가할 수 있습니다. 자세한 내용은 AWS Backup 볼트 잠금 섹션을 참조하세요.

모든 리전에서 데이터 보호 기능을 사용할 수 있습니다. 기능에 대한 자세한 내용은 AWS Backup 개발자 안내서를 참조하세요.

Timestream 테이블 백업 및 복원: 작동 방식

Amazon Timestream 테이블의 백업을 생성할 수 있습니다. 이 섹션에서는 백업 및 복원 프로세스를 수행하는 중에 발생하는 상황에 대한 개요를 제공합니다.

주제

- 백업
- 복원

백업

온디맨드 백업 기능을 사용하여 LiveAnalytics 테이블용 Amazon Timestream의 전체 백업을 생성할 수 있습니다. 이 섹션에서는 백업 및 복원 프로세스를 수행하는 중에 발생하는 상황에 대한 개요를 제공합니다.

테이블 단위로 Timestream 데이터의 백업을 생성할 수 있습니다. Timestream 콘솔 또는 AWS Backup 콘솔, 또는 를 사용하여 선택한 테이블의 백업을 시작할 수 있습니다SDKCLI. 백업은 비동기적으로 생성되며 백업 시작 시간이 백업에 포함될 때까지 테이블의 모든 데이터가 생성됩니다. 그러나 백업이 진행되는 동안 테이블에 수집된 일부 데이터가 백업에 포함될 수도 있습니다. 데이터를 보호하기 위해 일회성 온디맨드 백업을 생성하거나 테이블의 반복 백업을 예약할 수 있습니다.

백업이 진행 중인 동안에는 다음을 수행할 수 없습니다.

- 백업 작업을 일시 중지하거나 취소합니다.
- 백업의 원본 테이블을 삭제합니다.
- 테이블에 대한 백업이 진행 중인 경우 해당 테이블에서 백업을 비활성화합니다.

구성되면 자동 백업 일정, 보존 관리 및 수명 주기 관리를 AWS Backup 제공하므로 사용자 지정 스크립트 및 수동 프로세스가 필요하지 않습니다. 자세한 내용은 AWS Backup 개발자 안내서를 참조하세요.

LiveAnalytics 백업을 위한 모든 Timestream은 본질적으로 증분적이므로 테이블의 첫 번째 백업은 전체 백업이고 동일한 테이블의 모든 후속 백업은 증분 백업으로, 마지막 백업 이후의 데이터 변경 사항만 복사합니다. 에 대한 Timestream의 데이터가 파티션 컬렉션에 LiveAnalytics 저장되므로 마지막 백업 이후 새 데이터를 수집하거나 기존 데이터를 업데이트하여 변경된 모든 파티션은 후속 백업 중에 복사됩니다.

LiveAnalytics 콘솔용 Timestream을 사용하는 경우 계정의 모든 리소스에 대해 생성된 백업이 백업 탭에 나열됩니다. 또한 백업은 테이블 세부 정보에도 나열됩니다.

복원

콘솔용 Timestream 또는 LiveAnalytics 콘솔, 또는 AWS Backup 에서 테이블을 복원할 수 있습니다 SDK AWS CLI. 백업에서 전체 데이터를 복원하거나 선택한 데이터를 복원하도록 테이블 보존 설정을 구성할 수 있습니다. 복원을 시작할 때 다음 테이블 설정을 구성할 수 있습니다.

- 데이터베이스 이름(Database Name)
- 테이블 이름
- 메모리 스토어 보존
- 마그네틱 스토어 보존
- 마그네틱 스토리지 쓰기 활성화
- S3 오류 로그 위치(선택 사항)
- IAM 백업을 복원할 때 수임 AWS Backup 할 역할

이전 구성은 소스 테이블과 독립적입니다. 백업의 모든 데이터를 복원하려면 메모리 스토어 보존 기간과 마그네틱 스토어 보존 기간의 합계가 가장 오래된 타임스탬프와 현재 사이의 차이보다 크도록 새 테이블 설정을 구성하는 것이 좋습니다. 복원할 증분 백업을 선택하면 모든 데이터(증분 + 기본 전체 데이터)가 복원됩니다. 복원에 성공하면 테이블이 활성 상태이며 복원된 테이블에서 수집 및/또는 쿼리 작업을 수행할 수 있습니다. 하지만 복원이 진행 중인 동안에는 이러한 작업을 수행할 수 없습니다. 복원되면 테이블은 계정의 다른 테이블과 유사합니다.

Example 백업에서 모든 데이터 복원

이 예제에서는 다음과 같은 가정을 사용합니다.

가장 오래된 타임스탬프 -August 1, 2021 0:00:00

- 지금 -November 9, 2022 0:00:00

백업에서 모든 데이터를 복원하려면 다음과 같이 값을 입력하고 비교합니다.

1. 메모리 스토어 보존 및 마그네틱 스토어 보존 을 입력합니다. 예를 들어 이러한 값을 가정합니다.
   - 메모리 스토어 보존 - 12시간
   - 마그네틱 스토어 보존 기간 - 500일
2. 메모리 스토어 보존과 마그네틱 스토어 보존의 합계를 찾습니다.

```
12 hours + (500 * 24 hours) =
12 hours + 12,000 hours =
12,012 hours
```

3. 가장 오래된 타임스탬프와 지금의 차이점을 찾아보세요.

```
November 9, 2022 0:00:00 - August 1, 2021 0:00:00 =
465 days =
465 * 24 hours =
11,160 hours
```

4. 두 번째 단계의 보존 값 합계가 세 번째 단계의 시간 차이보다 큰지 확인합니다. 필요한 경우 보존 시간을 조정합니다.

```
12,012 > 11,160
true
```

Example 백업에서 선택한 데이터 복원

이 예제에서는 다음과 같은 가정을 사용합니다.

- 지금 -November 9, 2022 0:00:00

백업에서 선택한 데이터만 복원하려면 다음과 같이 값을 입력하고 비교합니다.

1. 필요한 가장 빠른 타임스탬프를 결정합니다. 예를 들어 를 가정합니다December 4, 2021 0:00:00.

2. 필요한 가장 빠른 타임스탬프와 현재 타임스탬프의 차이점을 찾습니다.

```
November 9, 2022 0:00:00 - December 4, 2021 0:00:00 =
340 days =
340 * 24 hours =
8,160 hours
```

3. 메모리 스토어 보존 에 원하는 값을 입력합니다. 예를 들어 12시간을 입력합니다.
4. 두 번째 단계의 차이에서 값을 뺍니다.

```
8,160 hours - 12 hours =
8148 hours
```

5. 마그네틱 스토어 보존에 대한 값을 입력합니다.

LiveAnalytics 테이블 데이터에 대한 Timestream의 백업을 다른 AWS 리전에 복사한 다음 해당 새 리전에서 복원할 수 있습니다. AWS 상용 리전과 AWS GovCloud (미국) 리전 간에 백업을 복사한 다음 복원할 수 있습니다. 소스 리전에서 복사한 데이터와 대상 리전의 새 테이블에 복원한 데이터에 대해서만 요금을 지불합니다.

테이블이 복원되면 복원된 테이블에 다음을 수동으로 설정해야 합니다.

- AWS 자격 증명 및 액세스 관리(IAM) 정책
- Tags
- 예약된 쿼리

복원 시간은 테이블 구성과 직접 관련이 있습니다. 여기에는 테이블 크기, 기본 파티션 수, 메모리 스토어에 복원된 데이터 양 및 기타 변수가 포함됩니다. 재해 복구를 계획할 때 가장 좋은 방법은 평균 복원 완료 시간을 정기적으로 문서화하고 이러한 시간이 전체 복구 시간 목표()에 미치는 영향을 설정하는 것입니다RTO.

모든 백업 및 복원 콘솔과 API 작업은 로깅, 지속적 모니터링 및 감사를 위해 에 AWS CloudTrail 캡처 및 기록됩니다.

Amazon Timestream 테이블의 백업 생성

이 섹션에서는 Amazon Timestream에 대한 온디맨드 AWS Backup 및 예약 백업을 활성화하고 생성하는 방법을 설명합니다.

주제

- LiveAnalytics 데이터에 대한 Timestream AWS Backup 보호 활성화
- 온디맨드 백업 생성
- 예약한 백업

LiveAnalytics 데이터에 대한 Timestream AWS Backup 보호 활성화

AWS Backup 에 대해 Timestream과 함께 사용할 수 있도록 를 활성화해야 합니다 LiveAnalytics.

LiveAnalytics 콘솔용 Timestream AWS Backup 에서 를 활성화하려면 다음 단계를 수행합니다.

1. AWS 관리 콘솔에 로그인합니다.
2. 가 LiveAnalytics 데이터에 대해 Timestream을 지원할 수 AWS Backup 있도록 LiveAnalytics 대시보드용 Timestream 페이지 상단에 팝업 배너가 나타납니다. 그렇지 않으면 탐색 창에서 백업을 선택합니다.
3. 백업 창에 를 활성화할 배너가 표시됩니다 AWS Backup. 활성화를 선택합니다.

   AWS Backup 이제 LiveAnalytics 테이블용 Timestream에서 를 통한 데이터 보호를 사용할 수 있습니다.

를 통해 활성화하려면 콘솔을 통해 프로그래밍 방식으로 활성화하는 AWS Backup 설명서를 AWS Backup참조하세요.

데이터가 활성화된 후 Timestream AWS Backup 의 LiveAnalytics 데이터 보호를 비활성화하려면 AWS Backup 콘솔을 통해 로그인하고 토글을 왼쪽으로 이동합니다.

AWS Backup 기능을 활성화하거나 비활성화할 수 없는 경우 AWS 관리자가 이러한 작업을 수행해야 할 수 있습니다.

온디맨드 백업 생성

LiveAnalytics 테이블에 대한 Timestream의 온디맨드 백업을 생성하려면 다음 단계를 따르세요.

1. AWS 관리 콘솔에 로그인합니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 온디맨드 백업 생성을 선택합니다.

4. 백업 창에서 설정을 계속 선택합니다.
5. 지금 백업을 생성하거나, 백업을 즉시 시작하거나, 백업 기간을 선택하여 백업을 시작할 수 있습니다.
6. 백업의 수명 주기 관리 정책을 선택합니다. 백업 데이터를 콜드 스토리지로 전환하여 백업을 최소 90일 동안 보존할 수 있습니다. 백업에 필요한 보존 기간을 설정할 수 있습니다. 기존 볼트를 선택하거나 새 백업 볼트 생성을 선택하여 AWS Backup 콘솔로 이동하여 새 백업 볼트를 생성할 수 있습니다. <여기서 새 백업 볼트 생성에 대한 문서 링크>
7. 적절한 IAM 역할을 선택합니다.
8. 온디맨드 백업에 하나 이상의 태그를 할당하려면 키 및 값(선택 사항)을 입력하고 태그 추가를 선택합니다.
9. 온디맨드 백업을 생성하려면 선택합니다. 그러면 작업 목록이 표시되는 백업 페이지로 이동합니다.
10. 백업하도록 선택한 리소스의 백업 작업 ID를 선택하여 해당 작업의 세부 정보를 확인합니다.

예약한 백업

백업을 예약하려면 예약된 백업 생성 을 참조하세요.

Amazon Timestream 테이블의 백업 복원

이 섹션에서는 Amazon Timestream 테이블의 백업을 복원하는 방법을 설명합니다.

주제

- 에서 LiveAnalytics 테이블에 대한 Timestream 복원 AWS Backup
- LiveAnalytics 테이블의 Timestream을 다른 리전 또는 계정으로 복원

에서 LiveAnalytics 테이블에 대한 Timestream 복원 AWS Backup

LiveAnalytics 콘솔용 Timestream을 AWS Backup 사용하여 LiveAnalytics 테이블의 Timestream을 복원하려면 다음 단계를 따르세요.

1. AWS 관리 콘솔에 로그인합니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 리소스를 복원하려면 리소스의 복구 시점 ID 옆에 있는 라디오 버튼을 선택합니다. 창의 오른쪽 위에서 복원을 선택합니다.

4. 테이블 구성 설정, 즉 데이터베이스 이름 및 테이블 이름을 입력합니다. 복원된 테이블 이름은 원래 소스 테이블 이름과 달라야 합니다.
5. 메모리 및 마그네틱 스토어 보존 설정을 구성합니다.
6. 복원 역할 에서 이 복원을 수임 AWS Backup 할 IAM 역할을 선택합니다.
7. 백업 복원을 선택합니다. 페이지 상단에 복원 작업에 대한 정보를 제공하는 메시지가 나타납니다.

> **Note**
>
> 구성된 메모리 및 마그네틱 스토어 보존 기간에 관계없이 전체 백업을 복원하는 데 요금이 부과됩니다. 하지만 복원이 완료되면 복원된 테이블에는 구성된 보존 기간 내의 데이터만 포함됩니다.

LiveAnalytics 테이블의 Timestream을 다른 리전 또는 계정으로 복원

LiveAnalytics 테이블의 Timestream을 다른 리전 또는 계정으로 복원하려면 먼저 백업을 해당 새 리전 또는 계정으로 복사해야 합니다. 다른 계정으로 복사하려면 먼저 해당 계정에서 권한을 부여해야 합니다. LiveAnalytics 백업을 위해 Timestream을 새 리전 또는 계정으로 복사한 후 이전 섹션의 프로세스를 사용하여 복원할 수 있습니다.

Amazon Timestream 테이블의 백업 복사

현재 백업의 복사본을 만들 수 있습니다. 요청 시 또는 예약된 백업 계획의 일부로 자동으로 여러 AWS 계정 또는 AWS 리전에 백업을 복사할 수 있습니다. 교차 리전 복제는 프로덕션 데이터로부터 최소 거리에 백업을 저장해야 하는 비즈니스 연속성 또는 규정 준수 요구 사항이 있는 경우 특히 유용합니다.

교차 계정 백업은 백업을 운영 또는 보안상의 이유로 조직의 하나 이상의 AWS 계정에 안전하게 복사하는 데 유용합니다. 원본 백업이 실수로 삭제된 경우 대상 계정에서 소스 계정으로 백업을 복사한 다음 복원을 시작할 수 있습니다. 이렇게 하려면 먼저 Organizations 서비스에서 동일한 조직에 속한 두 개의 계정과 계정에 필요한 권한이 있어야 합니다. 증분 백업을 다른 계정 또는 리전에 복사하면 연결된 전체 백업도 복사됩니다.

달리 지정하지 않는 한 복사는 소스 백업의 구성을 상속합니다. 한 가지 예외가 있습니다. 새 복사본을 “전혀 만료하지 않음”으로 지정하는 경우. 이 설정을 사용하면 새 복사본은 여전히 원본 만료 날짜를 상속합니다. 새 백업 복사본을 영구적으로 사용하려면 소스 백업이 만료되지 않도록 설정하거나 새 복사본을 만든 후 100년 후에 만료되도록 지정합니다.

Timestream 콘솔에서 백업을 복사하려면 다음 단계를 따르세요.

1. AWS 관리 콘솔에 로그인합니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 리소스의 복구 시점 ID 옆에 있는 라디오 버튼을 선택합니다. 창의 오른쪽 상단에서 작업을 선택하고 복사를 선택합니다.
4. AWS 백업으로 계속을 선택하고 크로스 계정 백업 단계를 따릅니다.

계정 및 리전 간에 온디맨드 백업 및 예약된 백업을 복사하는 것은 현재 LiveAnalytics 콘솔용 Timestream에서 기본적으로 지원되지 않으며 작업을 수행 AWS Backup 하려면 로 이동해야 합니다.

백업 삭제

이 섹션에서는 LiveAnalytics 테이블에 대한 Timestream의 백업을 삭제하는 방법을 설명합니다.

Timestream 콘솔에서 백업을 삭제하려면 다음 단계를 따르세요.

1. AWS 관리 콘솔에 로그인합니다.
2. 콘솔 왼쪽의 탐색 창에서 [Backups]를 선택합니다.
3. 리소스의 복구 시점 ID 옆에 있는 라디오 버튼을 선택합니다. 창의 오른쪽 상단에서 작업을 선택하고 삭제를 선택합니다.
4. AWS 백업으로 계속을 선택하고 백업 삭제 에서 백업을 삭제하는 단계를 따릅니다.

> **Note**
>
> 증분 백업을 삭제하면 증분 백업만 삭제되고 기본 전체 백업은 삭제되지 않습니다.

할당량 및 한도

AWS Backup 는 백업을 리소스당 하나의 동시 백업으로 제한합니다. 따라서 리소스에 대한 추가 예약 또는 온디맨드 백업 요청이 대기열에 추가되며 기존 백업 작업이 완료된 후에만 시작됩니다. 백업 기간 내에 백업 작업이 시작되거나 완료되지 않으면 요청이 실패합니다. AWS Backup 제한에 대한 자세한 내용은 AWS 백업 개발자 안내서의 백업 제한을 참조하세요. AWS

백업을 생성할 때 계정당 최대 4개의 동시 백업을 실행할 수 있습니다. 마찬가지로 계정당 동시 복원 하나를 실행할 수 있습니다. 4개 이상의 백업 작업을 동시에 시작하면 4개의 백업 작업만 시작되고 나머지 작업은 주기적으로 재시도됩니다. 일단 시작되면 구성된 백업 기간 내에 백업 작업이 완료되지 않으

면 백업 작업이 실패합니다. 실패한 백업 작업이 온디맨드 백업인 경우 백업을 다시 시도할 수 있으며 예약된 백업의 경우 다음 일정으로 작업이 시도됩니다.

고객 정의 파티션 키

Amazon Timestream for LiveAnalytics customer-defined partition key는 고객이 테이블에 대해 자체 파티션 키를 정의할 수 LiveAnalytics 있도록 지원하는 의 Timestream 기능입니다. 파티셔닝은 여러 물리적 스토리지 유닛에 데이터를 분산하는 데 사용되는 기법으로, 더 빠르고 효율적인 데이터 검색을 가능하게 합니다. 고객 정의 파티션 키를 사용하면 쿼리 패턴 및 사용 사례에 더 잘 맞는 파티셔닝 스키마를 생성할 수 있습니다.

LiveAnalytics 고객 정의 파티션 키의 Timestream을 사용하면 고객은 테이블의 파티션 키로 하나의 차원 이름을 선택할 수 있습니다. 이를 통해 데이터에 대한 파티셔닝 스키마를 보다 유연하게 정의할 수 있습니다. 올바른 파티션 키를 선택하면 고객은 데이터 모델을 최적화하여 쿼리 성능을 개선하고 쿼리 지연 시간을 줄일 수 있습니다.

주제

고객 정의 파티션 키 사용

카디널리티 차원이 높고 쿼리 지연 시간이 짧은 잘 정의된 쿼리 패턴이 있는 경우 LiveAnalytics 고객 정의 파티션 키를 위한 Timestream이 데이터 모델을 개선하는 데 유용한 도구가 될 수 있습니다. 예를 들어 웹 사이트에서 고객 상호 작용을 추적하는 소매 회사인 경우 주요 액세스 패턴은 고객 ID 및 타임스탬프일 수 있습니다. 고객 ID를 파티션 키로 정의하면 데이터를 균등하게 분산하여 지연 시간을 줄이고 궁극적으로 사용자 환경을 개선할 수 있습니다.

또 다른 예는 웨어러블 디바이스가 환자의 활력 징후를 추적하기 위해 센서 데이터를 수집하는 의료 산업입니다. 기본 액세스 패턴은 두 차원 모두에서 카디널리티가 높은 디바이스 ID 및 타임스탬프입니다. 디바이스 ID를 파티션 키로 정의하면 는 쿼리 실행을 최적화하고 장기 쿼리 성능을 유지할 수 있습니다.

요약하면 LiveAnalytics 고객 정의 파티션 키의 Timestream은 명확한 쿼리 패턴, 높은 카디널리티 차원이 있고 쿼리에 짧은 지연 시간이 필요한 경우에 가장 유용합니다. 쿼리 패턴과 일치하는 파티션 키를 정의하면 쿼리 실행을 최적화하고 장기 성능 쿼리 성능을 유지할 수 있습니다.

고객 정의 파티션 키 시작하기

콘솔에서 테이블을 선택하고 새 테이블을 생성합니다. SDK 를 사용하여 `CreateTable` 작업에 액세스하여 고객 정의 파티션 키를 포함할 수 있는 새 테이블을 생성할 수도 있습니다.

차원 유형 파티션 키를 사용하여 테이블 생성

다음 코드 조각을 사용하여 차원 유형 파티션 키가 있는 테이블을 생성할 수 있습니다.

Java

```
public void createTableWithDimensionTypePartitionKeyExample() {
        System.out.println("Creating table");
        CreateTableRequest createTableRequest = new CreateTableRequest();
        createTableRequest.setDatabaseName(DATABASE_NAME);
        createTableRequest.setTableName(TABLE_NAME);
        final RetentionProperties retentionProperties = new RetentionProperties()
                .withMemoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .withMagneticStoreRetentionPeriodInDays(CT_TTL_DAYS);
        createTableRequest.setRetentionProperties(retentionProperties);

        // Can specify enforcement level with OPTIONAL or REQUIRED
        final List<PartitionKey> partitionKeyWithDimensionAndOptionalEnforcement =
 Collections.singletonList(new PartitionKey()
            .withName(COMPOSITE_PARTITION_KEY_DIM_NAME)
            .withType(PartitionKeyType.DIMENSION)
            .withEnforcementInRecord(PartitionKeyEnforcementLevel.OPTIONAL));
        Schema schema = new Schema();

 schema.setCompositePartitionKey(partitionKeyWithDimensionAndOptionalEnforcement);
        createTableRequest.setSchema(schema);

        try {
```

```
            writeClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
 DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Java v2

```
public void createTableWithDimensionTypePartitionKeyExample() {
        System.out.println("Creating table");
        final RetentionProperties retentionProperties =
 RetentionProperties.builder()
                .memoryStoreRetentionPeriodInHours(HT_TTL_HOURS)
                .magneticStoreRetentionPeriodInDays(CT_TTL_DAYS)
                .build();
        // Can specify enforcement level with OPTIONAL or REQUIRED
        final List<PartitionKey> partitionKeyWithDimensionAndOptionalEnforcement =
 Collections.singletonList(PartitionKey
                .builder()
                .name(COMPOSITE_PARTITION_KEY_DIM_NAME)
                .type(PartitionKeyType.DIMENSION)
                .enforcementInRecord(PartitionKeyEnforcementLevel.OPTIONAL)
                .build());
        final Schema schema = Schema.builder()

 .compositePartitionKey(partitionKeyWithDimensionAndOptionalEnforcement).build();
        final CreateTableRequest createTableRequest = CreateTableRequest.builder()
                .databaseName(DATABASE_NAME)
                .tableName(TABLE_NAME)
                .retentionProperties(retentionProperties)
                .schema(schema)
                .build();

        try {
            writeClient.createTable(createTableRequest);
            System.out.println("Table [" + TABLE_NAME + "] successfully created.");
        } catch (ConflictException e) {
            System.out.println("Table [" + TABLE_NAME + "] exists on database [" +
 DATABASE_NAME + "] . Skipping database creation");
        }
    }
```

Go v1

```
func createTableWithDimensionTypePartitionKeyExample(){
        // Can specify enforcement level with OPTIONAL or REQUIRED
        partitionKeyWithDimensionAndOptionalEnforcement :=
 []*timestreamwrite.PartitionKey{
                {
                    Name:                aws.String(CompositePartitionKeyDimName),
                    EnforcementInRecord: aws.String("OPTIONAL"),
                    Type:                aws.String("DIMENSION"),
                },
        }
        createTableInput := &timestreamwrite.CreateTableInput{
             DatabaseName: aws.String(*databaseName),
             TableName:    aws.String(*tableName),
             // Enable MagneticStoreWrite for Table
             MagneticStoreWriteProperties:
 &timestreamwrite.MagneticStoreWriteProperties{
                 EnableMagneticStoreWrites: aws.Bool(true),
                 // Persist MagneticStoreWrite rejected records in S3
                 MagneticStoreRejectedDataLocation:
 &timestreamwrite.MagneticStoreRejectedDataLocation{
                     S3Configuration: &timestreamwrite.S3Configuration{
                         BucketName:       aws.String("timestream-sample-bucket"),
                         ObjectKeyPrefix:  aws.String("TimeStreamCustomerSampleGo"),
                         EncryptionOption: aws.String("SSE_S3"),
                     },
                 },
             },
             Schema: &timestreamwrite.Schema{
                 CompositePartitionKey:
 partitionKeyWithDimensionAndOptionalEnforcement,
             }
         }
         _, err := writeSvc.CreateTable(createTableInput)
    }
```

Go v2

```
 func (timestreamBuilder TimestreamBuilder)
 CreateTableWithDimensionTypePartitionKeyExample() error {
         partitionKeyWithDimensionAndOptionalEnforcement := []types.PartitionKey{
             {
```

```
                Name:                aws.String(CompositePartitionKeyDimName),
                EnforcementInRecord: types.PartitionKeyEnforcementLevelOptional,
                Type:                types.PartitionKeyTypeDimension,
            },
        }
        _, err := timestreamBuilder.WriteSvc.CreateTable(context.TODO(),
&timestreamwrite.CreateTableInput{
            DatabaseName: aws.String(databaseName),
            TableName:    aws.String(tableName),
            MagneticStoreWriteProperties: &types.MagneticStoreWriteProperties{
                EnableMagneticStoreWrites: aws.Bool(true),
                // Persist MagneticStoreWrite rejected records in S3
                MagneticStoreRejectedDataLocation:
&types.MagneticStoreRejectedDataLocation{
                    S3Configuration: &types.S3Configuration{
                        BucketName:       aws.String(s3BucketName),
                        EncryptionOption: "SSE_S3",
                    },
                },
            },
            Schema: &types.Schema{
                CompositePartitionKey:
partitionKeyWithDimensionAndOptionalEnforcement,
            },
        })

        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Create table is successful")
        }
        return err
    }
```

Python

```
def create_table_with_measure_name_type_partition_key(self):
        print("Creating table")
        retention_properties = {
            'MemoryStoreRetentionPeriodInHours': HT_TTL_HOURS,
            'MagneticStoreRetentionPeriodInDays': CT_TTL_DAYS
        }
```

```
        partitionKey_with_measure_name = [
            {'Type': 'MEASURE'}
        ]
        schema = {
            'CompositePartitionKey': partitionKey_with_measure_name
        }
        try:
            self.client.create_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME,
                                     RetentionProperties=retention_properties,
Schema=schema)
            print("Table [%s] successfully created." % TABLE_NAME)
        except self.client.exceptions.ConflictException:
            print("Table [%s] exists on database [%s]. Skipping table creation" % (
                TABLE_NAME, DATABASE_NAME))
        except Exception as err:
            print("Create table failed:", err)
```

파티셔닝 스키마 구성 확인

몇 가지 방법으로 스키마 파티셔닝을 위한 테이블 구성을 확인할 수 있습니다. 콘솔에서 데이터베이스를 선택하고 확인할 테이블을 선택합니다. SDK 를 사용하여 `DescribeTable` 작업에 액세스할 수도 있습니다.

파티션 키가 있는 테이블 설명

다음 코드 조각을 사용하여 파티션 키가 있는 테이블을 설명할 수 있습니다.

Java

```
   public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest = new
DescribeTableRequest();
        describeTableRequest.setDatabaseName(DATABASE_NAME);
        describeTableRequest.setTableName(TABLE_NAME);
        try {
            DescribeTableResult result =
amazonTimestreamWrite.describeTable(describeTableRequest);
            String tableId = result.getTable().getArn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
```

```
            // If table is created with composite partition key, it can be described
with
            //
System.out.println(result.getTable().getSchema().getCompositePartitionKey());
        } catch (final Exception e) {
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

다음은 예시 출력입니다.

1. 테이블에 차원 유형 파티션 키 있음

```
[{Type: DIMENSION,Name: hostId,EnforcementInRecord: OPTIONAL}]
```

2. 테이블에 측정 이름 유형 파티션 키가 있음

```
[{Type: MEASURE,}]
```

3. 복합 파티션 키를 지정하지 않고 생성된 테이블에서 복합 파티션 키 가져오기

```
[{Type: MEASURE,}]
```

Java v2

```
 public void describeTable() {
        System.out.println("Describing table");
        final DescribeTableRequest describeTableRequest =
DescribeTableRequest.builder()
                .databaseName(DATABASE_NAME).tableName(TABLE_NAME).build();
        try {
            DescribeTableResponse response =
writeClient.describeTable(describeTableRequest);
            String tableId = response.table().arn();
            System.out.println("Table " + TABLE_NAME + " has id " + tableId);
            // If table is created with composite partition key, it can be described
with
            //
System.out.println(response.table().schema().compositePartitionKey());
        } catch (final Exception e) {
```

```
            System.out.println("Table " + TABLE_NAME + " doesn't exist = " + e);
            throw e;
        }
    }
```

다음은 예시 출력입니다.

1. 테이블에 차원 유형 파티션 키 있음

```
[PartitionKey(Type=DIMENSION, Name=hostId, EnforcementInRecord=OPTIONAL)]
```

2. 테이블에 측정 이름 유형 파티션 키가 있음

```
[PartitionKey(Type=MEASURE)]
```

3. 복합 파티션 키를 지정하지 않고 생성된 테이블에서 복합 파티션 키를 가져오면

```
[PartitionKey(Type=MEASURE)]
```

Go v1

```
    <tablistentry>
     <tabname> Go </tabname>
     <tabcontent>
      <programlisting language="go"></programlisting>
     </tabcontent>
    </tablistentry>
```

다음은 예제 출력입니다.

```
{
  Table: {
    Arn: "arn:aws:timestream:us-west-2:533139590831:database/devops/table/
host_metrics_dim_pk_1",
    CreationTime: 2023-05-31 01:52:00.511 +0000 UTC,
    DatabaseName: "devops",
    LastUpdatedTime: 2023-05-31 01:52:00.511 +0000 UTC,
    MagneticStoreWriteProperties: {
      EnableMagneticStoreWrites: true,
      MagneticStoreRejectedDataLocation: {
        S3Configuration: {
```

```
          BucketName: "timestream-sample-bucket-west",
          EncryptionOption: "SSE_S3",
          ObjectKeyPrefix: "TimeStreamCustomerSampleGo"
        }
      }
    },
    RetentionProperties: {
      MagneticStoreRetentionPeriodInDays: 73000,
      MemoryStoreRetentionPeriodInHours: 6
    },
    Schema: {
      CompositePartitionKey: [{
          EnforcementInRecord: "OPTIONAL",
          Name: "hostId",
          Type: "DIMENSION"
        }]
    },
    TableName: "host_metrics_dim_pk_1",
    TableStatus: "ACTIVE"
  }
}
```

Go v2

```
func (timestreamBuilder TimestreamBuilder) DescribeTable()
(*timestreamwrite.DescribeTableOutput, error) {
        describeTableInput := &timestreamwrite.DescribeTableInput{
            DatabaseName: aws.String(databaseName),
            TableName:    aws.String(tableName),
        }
       describeTableOutput, err :=
timestreamBuilder.WriteSvc.DescribeTable(context.TODO(), describeTableInput)

       if err != nil {
           fmt.Printf("Failed to describe table with Error: %s", err.Error())
       } else {
           fmt.Printf("Describe table is successful : %s\n",
JsonMarshalIgnoreError(*describeTableOutput))
           // If table is created with composite partition key, it will be included
in the output
       }

       return describeTableOutput, err
```

```
    }
```

다음은 예제 출력입니다.

```
{
  "Table": {
    "Arn":"arn:aws:timestream:us-east-1:351861611069:database/cdpk-wr-db/table/
host_metrics_dim_pk",
    "CreationTime":"2023-05-31T22:36:10.66Z",
    "DatabaseName":"cdpk-wr-db",
    "LastUpdatedTime":"2023-05-31T22:36:10.66Z",
    "MagneticStoreWriteProperties":{
      "EnableMagneticStoreWrites":true,
      "MagneticStoreRejectedDataLocation":{
        "S3Configuration":{
          "BucketName":"error-configuration-sample-s3-bucket-cq8my",
          "EncryptionOption":"SSE_S3",
          "KmsKeyId":null,"ObjectKeyPrefix":null
        }
      }
    },
    "RetentionProperties":{
      "MagneticStoreRetentionPeriodInDays":73000,
      "MemoryStoreRetentionPeriodInHours":6
    },
    "Schema":{
      "CompositePartitionKey":[{
        "Type":"DIMENSION",
        "EnforcementInRecord":"OPTIONAL",
        "Name":"hostId"
      }]
    },
    "TableName":"host_metrics_dim_pk",
    "TableStatus":"ACTIVE"
  },
  "ResultMetadata":{}
}
```

Python

```
  def describe_table(self):
        print('Describing table')
        try:
```

```
            result = self.client.describe_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME)
            print("Table [%s] has id [%s]" % (TABLE_NAME, result['Table']['Arn']))
            # If table is created with composite partition key, it can be described
with
            # print(result['Table']['Schema'])
        except self.client.exceptions.ResourceNotFoundException:
            print("Table doesn't exist")
        except Exception as err:
            print("Describe table failed:", err)
```

다음은 예시 출력입니다.

1. 테이블에 차원 유형 파티션 키 있음

```
[{'CompositePartitionKey': [{'Type': 'DIMENSION', 'Name': 'hostId',
 'EnforcementInRecord': 'OPTIONAL'}]}]
```

2. 테이블에 측정 이름 유형 파티션 키가 있음

```
[{'CompositePartitionKey': [{'Type': 'MEASURE'}]}]
```

3. 복합 파티션 키를 지정하지 않고 생성된 테이블에서 복합 파티션 키 가져오기

```
[{'CompositePartitionKey': [{'Type': 'MEASURE'}]}]
```

파티셔닝 스키마 구성 업데이트

UpdateTable 작업에 액세스할 수 SDK 있는 를 사용하여 분할 스키마에 대한 테이블 구성을 업데이트할 수 있습니다.

파티션 키로 테이블 업데이트

다음 코드 조각을 사용하여 파티션 키로 테이블을 업데이트할 수 있습니다.

Java

```
    public void updateTableCompositePartitionKeyEnforcement() {
        System.out.println("Updating table");
```

```
        UpdateTableRequest updateTableRequest = new UpdateTableRequest();
        updateTableRequest.setDatabaseName(DATABASE_NAME);
        updateTableRequest.setTableName(TABLE_NAME);

        // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
        final List<PartitionKey> partitionKeyWithDimensionAndRequiredEnforcement =
Collections.singletonList(new PartitionKey()
            .withName(COMPOSITE_PARTITION_KEY_DIM_NAME)
            .withType(PartitionKeyType.DIMENSION)
            .withEnforcementInRecord(PartitionKeyEnforcementLevel.REQUIRED));
        Schema schema = new Schema();

schema.setCompositePartitionKey(partitionKeyWithDimensionAndRequiredEnforcement);
        updateTableRequest.withSchema(schema);

        writeClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
```

Java v2

```
   public void updateTableCompositePartitionKeyEnforcement() {
        System.out.println("Updating table");
        // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
        final List<PartitionKey> partitionKeyWithDimensionAndRequiredEnforcement =
Collections.singletonList(PartitionKey
            .builder()
            .name(COMPOSITE_PARTITION_KEY_DIM_NAME)
            .type(PartitionKeyType.DIMENSION)
            .enforcementInRecord(PartitionKeyEnforcementLevel.REQUIRED)
            .build());
        final Schema schema = Schema.builder()

.compositePartitionKey(partitionKeyWithDimensionAndRequiredEnforcement).build();
        final UpdateTableRequest updateTableRequest = UpdateTableRequest.builder()

.databaseName(DATABASE_NAME).tableName(TABLE_NAME).schema(schema).build();

        writeClient.updateTable(updateTableRequest);
        System.out.println("Table updated");
```

Go v1

```
// Update table partition key enforcement attribute
   updateTableInput := &timestreamwrite.UpdateTableInput{
        DatabaseName: aws.String(*databaseName),
        TableName:    aws.String(*tableName),
        // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
        Schema: &timestreamwrite.Schema{
            CompositePartitionKey: []*timestreamwrite.PartitionKey{
                {
                        Name:
aws.String(CompositePartitionKeyDimName),
                        EnforcementInRecord: aws.String("REQUIRED"),
                        Type:                aws.String("DIMENSION"),
                },
            }},
    }
    updateTableOutput, err := writeSvc.UpdateTable(updateTableInput)
        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update table is successful, below is the output:")
            fmt.Println(updateTableOutput)
        }
```

Go v2

```
// Update table partition key enforcement attribute
        updateTableInput := &timestreamwrite.UpdateTableInput{
            DatabaseName: aws.String(*databaseName),
            TableName:    aws.String(*tableName),
            // Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
            Schema: &types.Schema{
                CompositePartitionKey: []types.PartitionKey{
                    {
                        Name:
aws.String(CompositePartitionKeyDimName),
                        EnforcementInRecord:
types.PartitionKeyEnforcementLevelRequired,
                        Type:                types.PartitionKeyTypeDimension,
```

```
                    },
                }},
        }
        updateTableOutput, err :=
timestreamBuilder.WriteSvc.UpdateTable(context.TODO(), updateTableInput)
        if err != nil {
            fmt.Println("Error:")
            fmt.Println(err)
        } else {
            fmt.Println("Update table is successful, below is the output:")
            fmt.Println(updateTableOutput)
        }
```

Python

```
   def update_table(self):
       print('Updating table')
       try:
           # Can update enforcement level for dimension type partition key with
OPTIONAL or REQUIRED enforcement
           partition_key_with_dimension_and_required_enforcement = [
               {
                   'Type': 'DIMENSION',
                   'Name': COMPOSITE_PARTITION_KEY_DIM_NAME,
                   'EnforcementInRecord': 'REQUIRED'
               }
           ]
           schema = {
               'CompositePartitionKey':
partition_key_with_dimension_and_required_enforcement
           }
           self.client.update_table(DatabaseName=DATABASE_NAME,
TableName=TABLE_NAME,
                                    Schema=schema)
           print('Table updated.')
       except Exception as err:
           print('Update table failed:', err)
```

고객 정의 파티션 키의 이점

향상된 쿼리 성능: 고객 정의 파티션 키를 사용하면 쿼리 실행을 최적화하고 전체 쿼리 성능을 개선할 수 있습니다. 쿼리 패턴과 일치하는 파티션 키를 정의하면 데이터 스캔을 최소화하고 데이터 정리를 최적화하여 쿼리 지연 시간을 줄일 수 있습니다.

장기 성능 예측 가능성 개선: 고객 정의 파티션 키를 사용하면 고객이 파티션 간에 데이터를 균등하게 분산하여 데이터 관리의 효율성을 개선할 수 있습니다. 이렇게 하면 저장된 데이터가 시간이 지남에 따라 확장됨에 따라 쿼리 성능이 안정적으로 유지됩니다.

고객 정의 파티션 키의 제한 사항

LiveAnalytics 사용자를 위한 Timestream으로서 고객 파티션 키와 관련된 제한 사항을 염두에 두는 것이 중요합니다. 먼저 워크로드와 쿼리 패턴을 잘 이해해야 합니다. 즉, 쿼리에서 기본 필터링 조건으로 가장 자주 사용되는 차원을 명확하게 파악하고 파티션 키를 가장 효과적으로 사용할 수 있는 카디널리티가 높아야 합니다.

둘째, 테이블 생성 시 파티션 키를 정의해야 하며 기존 테이블에 추가할 수 없습니다. 즉, 테이블을 생성하기 전에 파티셔닝 전략을 신중하게 고려하여 비즈니스 요구 사항에 맞게 조정해야 합니다.

마지막으로 테이블이 생성되면 나중에 파티션 키를 변경할 수 없다는 점에 유의해야 합니다. 즉, 파티셔닝 전략을 실행하기 전에 이를 철저히 테스트하고 평가해야 합니다. 이러한 제한을 염두에 두고 Timestream의 고객 정의 파티션 키는 쿼리 성능과 장기 만족도를 크게 개선할 수 있습니다.

고객 정의 파티션 키 및 낮은 카디널리티 차원

특정 리전 또는 상태와 같이 카디널리티가 매우 낮은 파티션 키를 사용하기로 결정한 경우, , `customerID` `ProductCategory`및 기타와 같은 다른 엔터티에 대한 의 데이터가 데이터가 거의 없거나 전혀 없는 너무 많은 파티션에 분산될 수 있다는 점에 유의해야 합니다. 이로 인해 쿼리 실행이 비효율적이고 성능이 저하될 수 있습니다.

이를 방지하려면 키 필터링 조건의 일부일 뿐만 아니라 카디널리티가 더 높은 차원을 선택하는 것이 좋습니다. 이렇게 하면 데이터가 파티션에 고르게 분산되고 쿼리 성능이 향상됩니다.

기존 테이블에 대한 파티션 키 생성

에 대한 테이블이 이미 Timestream에 LiveAnalytics 있고 고객 정의 파티션 키를 사용하려는 경우 원하는 파티셔닝 스키마 정의가 있는 새 테이블로 데이터를 마이그레이션해야 합니다. 이렇게 하려면 S3로 내보내기와 배치 로드를 함께 사용하면 됩니다. 여기에는 기존 테이블에서 S3로 데이터를 내보내고, 파티션 키를 포함하도록 데이터를 수정하고(필요한 경우), CSV 파일에 헤더를 추가한 다음 원하는 파

티셔닝 스키마가 정의된 새 테이블로 데이터를 가져오는 작업이 포함됩니다. 이 방법은 특히 대형 테이블의 경우 시간이 많이 걸리고 비용이 많이 들 수 있다는 점에 유의하세요.

또는 예약된 쿼리를 사용하여 원하는 파티셔닝 스키마가 있는 새 테이블로 데이터를 마이그레이션할 수 있습니다. 이 방법에는 기존 테이블에서 읽고 새 테이블에 쓰는 예약된 쿼리를 생성하는 작업이 포함됩니다. 예약된 쿼리는 모든 데이터가 마이그레이션될 때까지 정기적으로 실행되도록 설정할 수 있습니다. 마이그레이션 프로세스 중에 데이터를 읽고 쓰는 데 대한 요금이 청구된다는 점에 유의하세요.

사용자 지정 복합 파티션 키를 사용한 LiveAnalytics 스키마 검증을 위한 타임스트림

에 대한 Timestream의 스키마 검증은 데이터베이스에 수집된 데이터가 지정된 스키마를 준수하도록 하여 수집 오류를 최소화하고 데이터 품질을 개선하는 LiveAnalytics 데 도움이 됩니다. 특히 스키마 검증은 쿼리 성능을 최적화하기 위해 고객 정의 파티션 키를 채택할 때 특히 유용합니다.

고객 정의 파티션 키를 사용한 LiveAnalytics 스키마 검증을 위한 Timestream이란 무엇입니까?

LiveAnalytics 스키마 검증을 위한 Timestream은 사전 정의된 스키마를 기반으로 테이블의 LiveAnalytics Timestream에 수집되는 데이터를 검증하는 기능입니다. 이 스키마는 삽입되는 레코드에 대한 파티션 키, 데이터 유형 및 제약 조건을 포함한 데이터 모델을 정의합니다.

고객 정의 파티션 키를 사용하면 스키마 검증이 훨씬 더 중요해집니다. 파티션 키를 사용하면 의 Timestream에 데이터가 저장되는 방법을 결정하는 파티션 키를 지정할 수 있습니다 LiveAnalytics. 사용자 지정 파티션 키를 사용하여 스키마에 대해 수신 데이터를 검증하면 데이터 일관성을 적용하고, 오류를 조기에 감지하고, 의 Timestream에 저장된 데이터의 전반적인 품질을 개선할 수 있습니다 LiveAnalytics.

사용자 지정 복합 파티션 키를 사용하여 LiveAnalytics 스키마 검증에 Timestream을 사용하는 방법

사용자 지정 복합 파티션 키를 사용하여 LiveAnalytics 스키마 검증에 Timestream을 사용하려면 다음 단계를 따르세요.

쿼리 패턴이 어떻게 보일지 생각해 보세요. LiveAnalytics 테이블용 Timestream의 스키마를 올바르게 선택하고 정의하려면 쿼리 요구 사항으로 시작해야 합니다.

사용자 지정 복합 파티션 키 지정: 테이블을 생성할 때 사용자 지정 파티션 키를 지정합니다. 이 키는 테이블 데이터를 분할하는 데 사용할 속성을 결정합니다. 파티셔닝을 위한 차원 키와 측정 키 중에서 선

택할 수 있습니다. 차원 키는 차원 이름을 기반으로 데이터를 분할하는 반면, 측정 키는 측정 이름을 기반으로 데이터를 분할합니다.

적용 수준 설정: 적절한 데이터 파티셔닝과 함께 제공되는 이점을 보장하기 위해 용 Amazon Timestream을 LiveAnalytics 사용하면 스키마의 각 파티션 키에 대한 적용 수준을 설정할 수 있습니다. 적용 수준은 레코드를 수집할 때 파티션 키 차원이 필요한지 아니면 선택 사항인지 결정합니다. 두 가지 옵션 중에서 선택할 수 있습니다. 즉`REQUIRED`, 수집 레코드에 파티션 키가 있어야 하고, `OPTIONAL`는 파티션 키가 없어야 합니다. 고객 정의 파티션을 사용할 때는 `REQUIRED` 적용 수준을 사용하여 데이터가 올바르게 분할되고 이 기능의 모든 이점을 얻을 수 있도록 하는 것이 좋습니다. 또한 스키마 생성 후 언제든지 적용 수준 구성을 변경하여 데이터 수집 요구 사항에 맞게 조정할 수 있습니다.

데이터 수집: 테이블의 LiveAnalytics Timestream에 데이터를 수집할 때 스키마 검증 프로세스는 사용자 지정 복합 파티션 키를 사용하여 정의된 스키마와 비교하여 레코드를 확인합니다. 레코드가 스키마를 준수하지 않으면 에 대한 Timestream LiveAnalytics 은 검증 오류를 반환합니다.

검증 오류 처리: 검증 오류가 발생하는 경우 의 Timestream LiveAnalytics 은 오류 유형에 `RejectedRecordsException`따라 `ValidationException` 또는 를 반환합니다. 애플리케이션에서 이러한 예외를 처리하고 잘못된 레코드 수정 및 수집 재시도와 같은 적절한 조치를 취해야 합니다.

적용 수준 업데이트: 필요한 경우 `UpdateTable` 작업을 사용하여 테이블 생성 후 파티션 키의 적용 수준을 업데이트할 수 있습니다. 하지만 테이블 생성 후에는 이름, 유형과 같은 파티션 키 구성의 일부 측면을 변경할 수 없다는 점에 유의해야 합니다. 적용 수준을 에서 `REQUIRED`로 변경하면 고객 정의 파티션 키로 선택된 속성의 존재 여부에 관계없이 `OPTIONAL`모든 레코드가 수락됩니다. 반대로 적용 수준을 에서 `OPTIONAL`로 변경하면 이 조건을 충족하지 않는 레코드에 대해 4xx 쓰기 오류가 나타나`REQUIRED`기 시작할 수 있습니다. 따라서 데이터 파티셔닝 요구 사항에 따라 테이블을 생성할 때 사용 사례에 적합한 적용 수준을 선택해야 합니다.

사용자 지정 복합 파티션 키를 사용하여 LiveAnalytics 스키마 검증에 Timestream을 사용해야 하는 경우

데이터 일관성, 품질 및 최적화된 파티셔닝이 중요한 시나리오에서는 사용자 지정 복합 파티션 키를 사용한 LiveAnalytics 스키마 검증을 위한 타임스트림을 사용해야 합니다. 데이터 수집 중에 스키마를 적용하면 잘못된 분석 또는 귀중한 인사이트 손실로 이어질 수 있는 오류와 불일치를 방지할 수 있습니다.

배치 로드 작업과의 상호 작용

고객 정의 파티션 키를 사용하여 테이블로 데이터를 가져오도록 배치 로드 작업을 설정할 때 프로세스에 영향을 미칠 수 있는 몇 가지 시나리오가 있습니다.

1. 적용 수준이 로 설정된 경우 작업 구성 중에 파티션 키가 매핑되지 않은 경우 생성 흐름 중에 콘솔에 OPTIONAL알림이 표시됩니다. API 또는 를 사용할 때는 이 알림이 표시되지 않습니다CLI.
2. 적용 수준이 로 설정된 경우 파티션 키REQUIRED가 소스 데이터 열에 매핑되지 않는 한 작업 생성이 거부됩니다.
3. 작업이 생성된 REQUIRED 후 적용 수준이 로 변경되면 작업은 계속 실행되지만 파티션 키에 대한 적절한 매핑이 없는 레코드는 4xx 오류와 함께 거부됩니다.

예약된 쿼리와의 상호 작용

집계, 롤업 및 기타 형태의 사전 처리된 데이터를 계산하고 고객 정의 파티션 키를 사용하여 테이블에 저장하기 위해 예약된 쿼리 작업을 설정할 때 프로세스에 영향을 미칠 수 있는 몇 가지 시나리오가 있습니다.

1. 적용 수준이 로 설정된 경우 작업 구성 중에 파티션 키가 매핑되지 않으면 OPTIONAL알림이 표시됩니다. API 또는 를 사용할 때는 이 알림이 표시되지 않습니다CLI.
2. 적용 수준이 로 설정된 경우 파티션 키REQUIRED가 소스 데이터 열에 매핑되지 않는 한 작업 생성이 거부됩니다.
3. 작업이 생성된 REQUIRED 후 적용 수준이 로 변경되고 예약된 쿼리 결과에 파티션 키 차원이 포함되지 않은 경우 작업의 다음 반복이 모두 실패합니다.

리소스에 태그 및 레이블 추가

태그를 사용하여 LiveAnalytics 리소스에 대한 Amazon Timestream에 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.
- 태그로 분류된 AWS 청구서를 참조하세요.

태깅은 Amazon Elastic Compute Cloud(Amazon EC2), Amazon Simple Storage Service(Amazon S3), Timestream for LiveAnalytics와 같은 AWS 서비스에서 지원됩니다. 효율적으로 태그를 지정하면 특정 태그와 연결하여 서비스 전체에 대해 생성된 보고서를 통해 비용을 분석할 수 있습니다.

태그 지정을 시작하려면 다음을 수행합니다.

1. 태그 지정 제한 사항 을 이해합니다.

2. 태그 지정 작업 을 사용하여 태그를 생성합니다.

끝으로, 최적화된 태깅 전략을 따르는 것이 좋습니다. 자세한 내용은 AWS 태깅 전략을 참조하세요.

태그 지정 제한

각 태그는 사용자가 정의하는 키와 값으로 구성됩니다. 다음과 같은 제한 사항이 있습니다.

- LiveAnalytics 테이블의 각 Timestream에는 동일한 키가 있는 태그가 하나만 있을 수 있습니다. 기존 태그를 추가하려고 하면 기존 태그 값이 새 값으로 업데이트됩니다.
- 값은 태그 범주 내에서 설명자 역할을 합니다. Timestream에서 LiveAnalytics 값의 값은 비어 있거나 null일 수 없습니다.
- 태그 키와 값은 대소문자를 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-할당된 태그 이름과 값에는 `aws:` 자동으로 접두사가 할당되며, 이 접두사는 할당할 수 없습니다. AWS-할당된 태그 이름은 태그 제한인 50에 포함되지 않습니다. 비용 할당 보고서에는 사용자가 지정한 태그 이름인 `user:` 접두사가 포함됩니다.
- 태그를 소급해서 적용할 수 없습니다.

태그 지정 작업

콘솔용 Amazon Timestream, 쿼리 언어 또는 AWS Command Line Interface ()를 사용하여 데이터베이스 및 테이블에 대한 LiveAnalytics 태그를 추가, 나열, 편집 또는 삭제할 수 있습니다AWS CLI.

주제

- 콘솔을 사용하여 신규 또는 기존 데이터베이스 및 테이블에 태그 추가

콘솔을 사용하여 신규 또는 기존 데이터베이스 및 테이블에 태그 추가

LiveAnalytics 콘솔용 Timestream을 사용하여 새 데이터베이스, 테이블 및 예약된 쿼리를 생성할 때 태그를 추가할 수 있습니다. 기존 테이블의 태그를 추가, 편집 또는 삭제할 수도 있습니다.

데이터베이스를 생성할 때 태그를 지정하려면(콘솔)

1. https://console.aws.amazon.com/timestream에서 Timestream 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스를 선택하고 데이터베이스 생성을 선택합니다.
3. 데이터베이스 생성 페이지에서 데이터베이스의 이름을 입력합니다. 태그에 대한 키 및 값을 입력하려면 새 태그 추가를 선택합니다.
4. 데이터베이스 생성를 선택합니다.

테이블을 생성할 때 테이블에 태그를 지정하려면(콘솔)

1. https://console.aws.amazon.com/timestream에서 Timestream 콘솔을 엽니다.
2. 탐색 창에서 테이블을 선택한 다음 테이블 생성을 선택합니다.
3. LiveAnalytics 테이블에 대한 Timestream 생성 페이지에서 테이블의 이름을 입력합니다. 태그의 키와 값을 입력하고 새 태그 추가를 선택합니다.
4. 테이블 생성을 선택합니다.

예약 쿼리를 생성할 때 태그를 지정하려면(콘솔)

1. https://console.aws.amazon.com/timestream에서 Timestream 콘솔을 엽니다.
2. 탐색 창에서 예약된 쿼리 를 선택한 다음 예약된 쿼리 생성 을 선택합니다.
3. 3단계. 쿼리 설정 페이지를 구성하고 새 태그 추가를 선택합니다. 해당 태그의 키와 값을 입력합니다. 태그를 추가하려면 새 태그 추가를 선택합니다.
4. Next(다음)를 선택합니다.

기존 리소스에 태그를 지정하려면(콘솔)

1. https://console.aws.amazon.com/timestream에서 Timestream 콘솔을 엽니다.
2. 탐색 창에서 데이터베이스 , 테이블 또는 예약된 쿼리 를 선택합니다.
3. 목록에서 데이터베이스 또는 테이블을 선택합니다. 그런 다음 태그 관리를 선택하여 태그를 추가, 편집 또는 삭제합니다.

태그 구조에 대한 자세한 내용은 태그 지정 제한 섹션을 참조하세요.

에 대한 Timestream의 보안 LiveAnalytics

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 가장 보안에 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. 는 안전하게 사용할 수 있는 서비스 AWS 도 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. 의 Timestream에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 준수 프로그램 의 범위 내 서비스를 LiveAnalytics참조하세요. AWS
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Timestream for 를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다 LiveAnalytics. 다음 주제에서는 보안 및 규정 준수 목표에 맞게 에 대한 Timestream LiveAnalytics을 구성하는 방법을 보여줍니다. 또한 Timestream에서 LiveAnalytics 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법도 알아봅니다.

주제

- 에 대한 Timestream의 데이터 보호 LiveAnalytics
- 용 Amazon Timestream의 자격 증명 및 액세스 관리 LiveAnalytics
- 에 대한 Timestream의 로깅 및 모니터링 LiveAnalytics
- Amazon Timestream Live Analytics의 복원력
- Amazon Timestream Live Analytics의 인프라 보안
- Timestream의 구성 및 취약성 분석
- 에 대한 Timestream의 인시던트 응답 LiveAnalytics
- VPC 엔드포인트(AWS PrivateLink)
- 용 Amazon Timestream의 보안 모범 사례 LiveAnalytics

에 대한 Timestream의 데이터 보호 LiveAnalytics

AWS 공동 책임 모델 Amazon Timestream Live Analytics의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든 를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 섹션을 FAQ참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 책임 공유 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management ()를 사용하여 개별 사용자를 설정하는 것이 좋습니다IAM. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다단계 인증(MFA)을 사용합니다.
- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2가 필요하며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- AWS 암호화 솔루션과 내의 모든 기본 보안 제어를 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 FIPS 를 AWS 통해 에 액세스할 때 140-3개의 검증된 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 API사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 콘솔, 또는 를 사용하여 Timestream Live Analytics 또는 기타 AWS 서비스 로 작업하는 경우가 포함됩니다API AWS CLI AWS SDKs. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL를 제공하는 경우 해당 서버에 대한 요청을 검증URL하기 위해 에 보안 인증 정보를 포함하지 않는 것이 좋습니다.

저장 시 암호화 및 키 관리와 같은 LiveAnalytics 데이터 보호 주제에 대한 Timestream에 대한 자세한 내용을 알아보려면 아래에서 사용 가능한 주제를 선택합니다.

주제

- 저장 중 암호화
- 전송 중 암호화
- 키 관리

저장 중 암호화

저장 시 LiveAnalytics 암호화를 위한 타임스트림은 AWS Key Management Service (AWS KMS)에 저장된 암호화 키를 사용하여 저장 중인 모든 데이터를 암호화하여 보안을 강화합니다. 이 기능을 사용하면 중요한 데이터 보호와 관련된 운영 부담 및 복잡성을 줄일 수 있습니다. 저장 시 암호화를 사용하면 엄격한 암호화 규정 준수 및 규제 요구 사항이 필요한, 보안에 민감한 애플리케이션을 구축할 수 있습니다.

- 암호화는 LiveAnalytics 데이터베이스용 Timestream에서 기본적으로 켜져 있으며 해제할 수 없습니다. 업계 표준 AES-256 암호화 알고리즘은 사용되는 기본 암호화 알고리즘입니다.
- AWS KMS 는 에 대한 Timestream의 저장 시 암호화에 필요합니다 LiveAnalytics.
- 테이블에서 항목의 하위 집합만 암호화할 수 없습니다.
- 암호화를 사용하도록 데이터베이스 클라이언트 애플리케이션을 수정하지 않아도 됩니다.

키를 제공하지 않으면 Timestream for 는 `alias/aws/timestream` 계정에 이름이 지정된 AWS KMS 키를 LiveAnalytics 생성하고 사용합니다.

에서 자체 고객 관리형 키를 사용하여 LiveAnalytics 데이터에 대한 TimestreamKMS을 암호화할 수 있습니다. 용 Timestream의 키에 대한 자세한 내용은 섹션을 LiveAnalytics참조하세요키 관리.

의 타임스트림은 메모리 스토어와 마그네틱 스토어의 두 스토리지 계층에 데이터를 LiveAnalytics 저장합니다. 메모리 스토어 데이터는 LiveAnalytics 서비스 키용 Timestream을 사용하여 암호화됩니다. 마그네틱 스토어 데이터는 키를 사용하여 AWS KMS 암호화됩니다.

Timestream Query 서비스는 데이터에 액세스하려면 자격 증명이 필요합니다. 이러한 자격 증명은 KMS 키를 사용하여 암호화됩니다.

> **Note**
>
> 의 Timestream LiveAnalytics 은 AWS KMS 모든 복호화 작업을 호출하지 않습니다. 대신 활성 트래픽으로 5분 동안 키의 로컬 캐시를 유지합니다. 모든 권한 변경 사항은 최대 5분 이내에 최종적으로 일관성을 유지하면서 LiveAnalytics 시스템의 Timestream을 통해 전파됩니다.

전송 중 암호화

모든 Timestream Live Analytics 데이터는 전송 중에 암호화됩니다. 기본적으로 에 대한 Timestream 간 모든 통신 LiveAnalytics 은 전송 계층 보안(TLS) 암호화를 사용하여 보호됩니다.

키 관리

Key AWS Management Service(AWS KMS)를 사용하여 Amazon Timestream Live Analytics의 키를 관리할 수 있습니다. Timestream Live Analytics는 를 사용하여 데이터를 암호화KMS해야 합니다. 키에 필요한 제어 정도에 따라 키 관리를 위한 다음 옵션이 있습니다.

데이터베이스 및 테이블 리소스

- Timestream Live Analytics 관리형 키: 키를 제공하지 않으면 Timestream Live Analytics가 를 사용하여 `alias/aws/timestream` 키를 생성합니다KMS.
- 고객 관리형 키: KMS 고객 관리형 키가 지원됩니다. 매년 자동으로 교체하는 기능을 포함하여 키의 권한 및 수명 주기에 대한 추가 제어가 필요한 경우 이 옵션을 선택합니다.

예약된 쿼리 리소스

- Timestream Live Analytics 소유 키: 키를 제공하지 않으면 Timestream Live Analytics는 자체 KMS 키를 사용하여 쿼리 리소스를 암호화하며 이 키는 Timestream 계정에 있습니다. 자세한 내용은 KMS 개발자 안내서의 AWS 소유 키를 참조하세요.
- 고객 관리형 키: KMS 고객 관리형 키가 지원됩니다. 매년 자동으로 교체하는 기능을 포함하여 키의 권한 및 수명 주기에 대한 추가 제어가 필요한 경우 이 옵션을 선택합니다.

KMS 외부 키 스토어(XKS)의 키는 지원되지 않습니다.

용 Amazon Timestream의 자격 증명 및 액세스 관리 LiveAnalytics

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 LiveAnalytics 리소스에 Timestream을 사용할 수 있는 인증(로그인) 및 권한 부여(권한 보유) 대상을 제어합니다. IAM 는 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- 고객
- ID를 통한 인증

- 정책을 사용한 액세스 관리
- 용 Amazon Timestream의 LiveAnalytics 작동 방식 IAM
- AWS Amazon Timestream Live Analytics에 대한 관리형 정책
- LiveAnalytics 자격 증명 기반 정책 예제를 위한 Amazon Timestream
- LiveAnalytics 자격 증명 및 액세스를 위한 Amazon Timestream 문제 해결

고객

AWS Identity and Access Management (IAM) 사용 방법은 에 대한 Timestream에서 수행하는 작업에 따라 다릅니다 LiveAnalytics.

서비스 사용자 - LiveAnalytics 서비스에 Timestream을 사용하여 작업을 수행하는 경우 관리자는 필요한 자격 증명과 권한을 제공합니다. LiveAnalytics 기능을 사용하여 작업을 수행하는 데 더 많은 Timestream을 사용하므로 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. 에 대한 Timestream의 기능에 액세스할 수 없는 경우 섹션을 LiveAnalytics참조하세요 LiveAnalytics 자격 증명 및 액세스를 위한 Amazon Timestream 문제 해결.

서비스 관리자 - 회사의 LiveAnalytics 리소스에 대해 Timestream을 담당하는 경우 의 Timestream에 대한 전체 액세스 권한이 있을 수 있습니다 LiveAnalytics. 서비스 사용자가 액세스해야 하는 LiveAnalytics 기능 및 리소스에 대한 Timestream을 결정하는 것은 사용자의 작업입니다. 그런 다음 IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여 의 기본 개념을 이해합니다IAM. 회사에서 용 TimestreamIAM을 사용하는 방법에 대한 자세한 내용은 섹션을 LiveAnalytics참조하세요용 Amazon Timestream의 LiveAnalytics 작동 방식 IAM.

IAM 관리자 - IAM 관리자인 경우 정책을 작성하여 의 Timestream에 대한 액세스를 관리하는 방법에 대한 세부 정보를 알고 싶을 수 있습니다 LiveAnalytics. 에서 사용할 수 있는 LiveAnalytics 자격 증명 기반 정책에 대한 Timestream 예제를 보려면 섹션을 IAM참조하세요 LiveAnalytics 자격 증명 기반 정책 예제를 위한 Amazon Timestream.

ID를 통한 인증

인증은 자격 증명 AWS 으로 에 로그인하는 방법입니다. 로 AWS 계정 루트 사용자, IAM 사용자로 또는 IAM 역할을 수임하여 인증(에 로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 에 페더레이션 자격 증명 AWS 으로 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션 자격 증명으로 로

그인하면 관리자가 이전에 IAM 역할을 사용하여 자격 증명 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 에 로그인하는 방법을 AWS 계정 AWS참조하세요.

AWS 프로그래밍 방식으로 에 액세스하는 경우 는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공하여 자격 증명을 사용하여 요청에 암호화 방식으로 서명합니다. AWS 도구를 사용하지 않는 경우 직접 요청에 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청에 대한 서명 버전 4를 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것이 AWS 좋습니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다단계 인증 및 사용 설명서의 AWS 다단계 인증을 IAM 참조하세요IAM.

IAM 사용자 및 그룹

IAM 사용자는 한 사람 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내 자격 증명입니다. 가능한 경우 암호 및 액세스 키와 같은 장기 보안 인증 정보가 있는 IAM 사용자를 생성하는 대신 임시 보안 인증 정보를 사용하는 것이 좋습니다. 그러나 IAM 사용자와 장기 보안 인증이 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례에 대한 액세스 키 정기적으로 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어 라는 이름의 그룹을 지정IAMAdmins하고 해당 그룹에 IAM 리소스를 관리할 수 있는 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 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내 자격 증명입니다. IAM 사용자와 비슷하지만 특정 사람과는 연결되지 않습니다. 에서 IAM 역할을 일시적으로 수임하려면 사용자에서 역할(콘솔)로 전환할 AWS Management Console수 있습니다. IAM 또는 AWS API 작업을 호출 AWS CLI 하거나 사용자 지정 를

사용하여 역할을 수임할 수 있습니다URL. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

IAM 임시 자격 증명이 있는 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 타사 자격 증명 공급자에 대한 역할 생성을 참조하세요. IAM Identity Center를 사용하는 경우 권한 세트를 구성합니다. 인증 후 자격 증명이 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 의 역할과 상호 연관시킵니다IAM. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 특정 작업에 대해 일시적으로 다른 권한을 맡을 IAM 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 누군가(신뢰할 수 있는 보안 주체)가 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 에서는 정책을 리소스에 직접 연결할 AWS 서비스수 있습니다(역할을 프록시로 사용하는 대신). 크로스 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 에서 크로스 계정 리소스 액세스를 IAM 참조하세요.
- 교차 서비스 액세스 - 일부 는 다른 에서 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어 서비스에서 호출할 때 해당 서비스가 Amazon에서 애플리케이션을 실행EC2하거나 Amazon S3에 객체를 저장하는 것이 일반적입니다. 서비스는직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여 에서 작업을 수행하면 보안 주체로 AWS간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS 는 를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 다른 AWS 서비스 또는 리소스와의 상호 작용을 완료해야 하는 요청을 수신할 때만 수행됩니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 액세스 세션 전달을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 수임하는 IAM 역할입니다. IAM 관리자는 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다IAM. 자세한 내용은 IAM 사용 설명서의 에 권한을 위임할 역할 생성을 AWS 서비스 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할에 대한 권한을 볼 수 있지만 편집할 수는 없습니다.

- Amazon에서 실행되는 애플리케이션 EC2 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 보안 인증을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장하는 것보다 좋습니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행 중인 프로그램이 임시 자격 증명을 가져올 수 있습니다. 자세한 내용은 IAM 사용 설명서EC2의 IAM 역할 사용을 참조하세요.

정책을 사용한 액세스 관리

정책을 AWS 생성하고 AWS 자격 증명 또는 리소스에 연결하여 의 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결된 AWS 경우 권한을 정의하는 의 객체입니다. 는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조 및 내용에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 정책을 사용하여 AWS JSON 대상에 액세스할 수 있는 사용자를 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 필요한 리소스에 대한 작업을 수행할 수 있는 권한을 부여하기 위해 IAM 관리자는 IAM 정책을 생성할 수 있습니다. 그런 다음 관리자는 IAM 정책을 역할에 추가하고 사용자는 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하는 데 사용하는 방법에 관계없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI또는 에서 역할 정보를 가져올 수 있습니다 AWS API.

보안 인증 기반 정책

자격 증명 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 고객 관리형 정책을 사용하여 사용자 지정 IAM 권한 정의를 참조하세요.

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 의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책 중에서 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 문서입니다. 리소스 기반 정책의 예로는 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 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 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 가 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책IAM에서는 의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACLs)

액세스 제어 목록(ACLs)은 리소스에 액세스할 수 있는 권한이 있는 보안 주체(계정 멤버, 사용자 또는 역할)를 제어합니다. ACLs 는 리소스 기반 정책과 유사하지만 JSON 정책 문서 형식을 사용하지는 않습니다.

Amazon S3 AWS WAF및 AmazonVPC은 를 지원하는 서비스의 예입니다ACLs. 에 대한 자세한 내용은 Amazon Simple Storage Service 개발자 안내서의 액세스 제어 목록(ACL) 개요를 ACLs참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 - 권한 경계는 자격 증명 기반 정책이 IAM엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 자격 증명 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 IAM엔터티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - 의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 JSON 정책 SCPs입니다 AWS Organizations. AWS Organizations 는 비즈니스가 소유 AWS 계정 한 여러 을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직의 모든 기능을 활성화하면 서비스 제어 정책(SCPs)을 모든 계정에 적용할 수 있습니다. 는 각 를 포함하여 멤버 계정의 엔터티에 대한 권한을 SCP 제한합니다 AWS 계정 루트 사용자. 조직 및 에 대한 자세한 내용은 AWS Organizations 사용 설명서의 서비스 제어 정책을 SCPs참조하세요.
- 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 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. AWS 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

용 Amazon Timestream의 LiveAnalytics 작동 방식 IAM

IAM 를 사용하여 용 Timestream에 대한 액세스를 관리하기 전에 용 Timestream과 함께 사용할 수 있는 IAM 기능을 이해해야 LiveAnalytics합니다 LiveAnalytics. LiveAnalytics 및 기타 AWS 서비스에 대한 Timestream이 에서 작동하는 방식을 개괄적으로 알아보려면 IAM 사용 설명서의 AWS 에서 작업하는 서비스를 IAM IAM참조하세요.

주제

- 자격 증명 기반 정책의 타임스트림 LiveAnalytics
- 리소스 기반 정책의 시간 흐름 LiveAnalytics
- LiveAnalytics 태그에 대한 Timestream 기반 권한 부여
- 역할의 타임스트림 LiveAnalytics IAM

자격 증명 기반 정책의 타임스트림 LiveAnalytics

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부된 작업 및 리소스와 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. 의 Timestream은 특정 작업 및 리소스와 조건 키를 LiveAnalytics 지원합니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

작업

관리자는 정책을 사용하여 AWS JSON 대상에 액세스할 수 있는 사용자를 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 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없

는 권한 전용 작업과 같은 몇 가지 예외가 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.

IAM 정책문의 작업 요소에 다음 작업을 지정할 수 있습니다. 정책을 사용하여 에서 작업을 수행할 수 있는 권한을 부여합니다AWS. 정책에서 작업을 사용하는 경우 일반적으로 동일한 이름의 API 작업, CLI 명령 또는 SQL 명령에 대한 액세스를 허용하거나 거부합니다.

경우에 따라 단일 작업은 API 작업 및 SQL 명령에 대한 액세스를 제어합니다. 또는 일부 작업을 수행하려면 다양한 작업이 필요합니다.

에서 지원되는 Timestream 목록은 아래 표를 참조 LiveAnalytics `Action`하세요.

> **Note**
>
> 모든 데이터베이스별 에 대해 작업을 특정 데이터베이스로 제한ARN하도록 데이터베이스를 지정할 `Actions`수 있습니다.

작업	설명	액세스 레벨	리소스 유형(*필수)
DescribeEndpoints	후속 요청을 수행해야 하는 Timestream 엔드포인트를 반환합니다.	모두	*
Select	하나 이상의 테이블에서 데이터를 선택하는 Timestream에서 쿼리를 실행합니다. 자세한 설명은 이 참고를 참조하세요.	읽기	테이블*
CancelQuery	쿼리를 취소합니다.	읽기	*
ListTables	테이블 목록을 가져옵니다.	나열	데이터베이스*

작업	설명	액세스 레벨	리소스 유형(*필수)
ListDatabases	데이터베이스 목록을 가져옵니다.	나열	*
ListMeasures	측정값 목록을 가져옵니다.	읽기	테이블*
DescribeTable	테이블 설명을 가져옵니다.	읽기	테이블*
DescribeDatabase	데이터베이스 설명을 가져옵니다.	읽기	데이터베이스*
SelectValues	특정 리소스를 지정할 필요가 없는 쿼리를 실행합니다. 자세한 설명은 이 참고를 참조하세요.	읽기	*
WriteRecords	Timestream에 데이터를 삽입합니다.	쓰기	테이블*
CreateTable	테이블을 생성합니다.	쓰기	데이터베이스*
CreateDatabase	데이터베이스를 생성합니다.	쓰기	*
DeleteDatabase	데이터베이스를 삭제합니다.	쓰기	*
UpdateDatabase	데이터베이스를 업데이트합니다.	쓰기	*
DeleteTable	테이블을 삭제합니다.	쓰기	데이터베이스*
UpdateTable	테이블을 업데이트합니다.	쓰기	데이터베이스*

SelectValues vs. 선택:

`SelectValues` 는 리소스가 필요하지 `Action` 않은 쿼리에 사용되는 입니다. 리소스가 필요하지 않은 쿼리의 예는 다음과 같습니다.

```
SELECT 1
```

이 쿼리는 LiveAnalytics 리소스에 대한 특정 Timestream을 참조하지 않습니다. 다른 예를 생각해 보세요.

```
SELECT now()
```

이 쿼리는 `now()` 함수를 사용하여 현재 타임스탬프를 반환하지만 리소스를 지정할 필요는 없습니다. `SelectValues` 는 에 대한 Timestream이 리소스 없이 쿼리를 실행할 LiveAnalytics 수 있도록 테스트에 자주 사용됩니다. 이제 `Select` 쿼리를 고려해 보겠습니다.

```
SELECT * FROM database.table
```

이 유형의 쿼리에는 지정된 데이터를 테이블에서 가져올 수 있도록 리소스, 특히 의 LiveAnalytics `table` Timestream이 필요합니다.

리소스

관리자는 정책을 사용하여 AWS JSON 대상에 액세스할 수 있는 사용자를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

`Resource` JSON 정책 요소는 작업이 적용되는 객체를 지정합니다. 문장에는 `Resource`또는 `NotResource`요소가 반드시 추가되어야 합니다. 가장 좋은 방법은 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정하는 것입니다. 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": "*"
```

데이터베이스 LiveAnalytics 및 테이블의 Timestream에서 IAM 권한 `Resource` 요소에 사용할 수 있습니다.

LiveAnalytics 데이터베이스 리소스의 Timestream에는 다음과 같은 가 있습니다ARN.

```
arn:${Partition}:timestream:${Region}:${Account}:database/${DatabaseName}
```

LiveAnalytics 테이블 리소스의 Timestream에는 다음과 같은 가 있습니다ARN.

```
arn:${Partition}:timestream:${Region}:${Account}:database/${DatabaseName}/table/
${TableName}
```

형식에 대한 자세한 내용은 Amazon 리소스 이름() 및 서비스 네임스페이스를 ARNs참조하세요. ARNs AWS

예를 들어 문에서 database 키스페이스를 지정하려면 다음 를 사용합니다ARN.

```
"Resource": "arn:aws:timestream:us-east-1:123456789012:database/mydatabase"
```

특정 계정에 속하는 모든 데이터베이스를 지정하려면 와일드카드(*)를 사용합니다.

```
"Resource": "arn:aws:timestream:us-east-1:123456789012:database/*"
```

리소스 생성과 같은 LiveAnalytics 일부 작업의 Timestream은 특정 리소스에서 수행할 수 없습니다. 이러한 경우, 와일드카드(*)를 사용해야 합니다.

```
"Resource": "*"
```

조건 키

에 대한 Timestream LiveAnalytics 은 서비스별 조건 키를 제공하지 않지만 일부 전역 조건 키 사용을 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

예시

LiveAnalytics 자격 증명 기반 정책에 대한 Timestream의 예를 보려면 섹션을 참조하세요 LiveAnalytics 자격 증명 기반 정책 예제를 위한 Amazon Timestream.

리소스 기반 정책의 시간 흐름 LiveAnalytics

에 대한 Timestream LiveAnalytics 은 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 단원을 참조하십시오.

LiveAnalytics 태그에 대한 Timestream 기반 권한 부여

태그를 사용하여 LiveAnalytics 리소스에 대한 Timestream 액세스를 관리할 수 있습니다. 태그를 기반으로 리소스 액세스를 관리하려면 `timestream:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다. LiveAnalytics 리소스의 Timestream 태그 지정에 대한 자세한 내용은 섹션을 참조하세요the section called “리소스에 태그 지정”.

해당 리소스의 태그를 기반으로 리소스에 대한 액세스를 제한하는 ID 기반 정책의 예를 보려면 태그를 기반으로 LiveAnalytics 리소스 액세스를 위한 타임스트림 섹션을 참조하세요.

역할의 타임스트림 LiveAnalytics IAM

IAM 역할은 AWS 계정 내에서 특정 권한이 있는 엔터티입니다.

용 Timestream에서 임시 자격 증명 사용 LiveAnalytics

임시 자격 증명을 사용하여 페더레이션으로 로그인하거나, IAM 역할을 맡거나, 교차 계정 역할을 맡을 수 있습니다. AssumeRole 또는 와 같은 작업을 호출 AWS STS API하여 임시 보안 자격 증명을 얻을 수 있습니다GetFederationToken.

서비스 연결 역할

에 대한 Timestream LiveAnalytics 은 서비스 연결 역할을 지원하지 않습니다.

서비스 역할

에 대한 Timestream LiveAnalytics 은 서비스 역할을 지원하지 않습니다.

AWS Amazon Timestream Live Analytics에 대한 관리형 정책

AWS 관리형 정책은 에서 생성하고 관리하는 독립 실행형 정책입니다 AWS. AWS 관리형 정책은 사용자, 그룹 및 역할에 권한 할당을 시작할 수 있도록 많은 일반적인 사용 사례에 대한 권한을 제공하도록 설계되었습니다.

AWS 관리형 정책은 모든 AWS 고객이 사용할 수 있으므로 특정 사용 사례에 대해 최소 권한 권한을 부여하지 않을 수 있다는 점에 유의하세요. 사용 사례에 고유한 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다.

AWS 관리형 정책에 정의된 권한은 변경할 수 없습니다. 가 AWS 관리형 정책에 정의된 권한을 AWS 업데이트하면 정책이 연결된 모든 보안 주체 자격 증명(사용자, 그룹 및 역할)에 영향을 미칩니다.

AWS 는 새 AWS 서비스 가 시작되거나 기존 서비스에 새 API 작업을 사용할 수 있게 되면 AWS 관리형 정책을 업데이트할 가능성이 높습니다.

자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

주제

- AWS 관리형 정책: AmazonTimestreamReadOnlyAccess
- AWS 관리형 정책: AmazonTimestreamConsoleFullAccess
- AWS 관리형 정책: AmazonTimestreamFullAccess
- AWS 관리형 정책에 대한 Timestream Live Analytics 업데이트

AWS 관리형 정책: AmazonTimestreamReadOnlyAccess

사용자, 그룹 및 역할에 `AmazonTimestreamReadOnlyAccess`를 연결할 수 있습니다. 이 정책은 Amazon Timestream에 대한 읽기 전용 액세스를 제공합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `Amazon Timestream` – Amazon Timestream에 대한 읽기 전용 액세스를 제공합니다. 또한 이 정책은 실행 중인 쿼리를 취소할 수 있는 권한을 부여합니다.

이 정책을 JSON 형식으로 검토하려면 섹션을 참조하세요AmazonTimestreamReadOnlyAccess.

AWS 관리형 정책: AmazonTimestreamConsoleFullAccess

사용자, 그룹 및 역할에 `AmazonTimestreamConsoleFullAccess`를 연결할 수 있습니다.

정책은 를 사용하여 Amazon Timestream을 관리할 수 있는 전체 액세스 권한을 제공합니다 AWS Management Console. 또한 이 정책은 저장된 쿼리를 관리할 수 있는 특정 AWS KMS 작업 및 작업에 대한 권한을 부여합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `Amazon Timestream` – 보안 주체에게 Amazon Timestream에 대한 전체 액세스 권한을 부여합니다.
- `AWS KMS` - 보안 주체가 별칭을 나열하고 키를 설명할 수 있습니다.
- `Amazon S3` - 보안 주체가 모든 Amazon S3 버킷을 나열할 수 있도록 허용합니다.
- `Amazon SNS` - 보안 주체가 Amazon SNS 주제를 나열할 수 있도록 허용합니다.
- `IAM` - 보안 주체가 IAM 역할을 나열할 수 있도록 허용합니다.
- `DBQMS` - 보안 주체에게 쿼리를 액세스, 생성, 삭제, 설명, 업데이트를 할 수 있도록 허용합니다. 데이터베이스 쿼리 메타데이터 서비스(dbqms)는 내부 전용 서비스입니다. Amazon Timestream을 AWS 서비스포함하여 여러 에 대한 의 쿼리 편집기에 AWS Management Console 대한 최근 쿼리와 저장된 쿼리를 제공합니다.

형식으로 이 정책을 검토하려면 섹션을 참조JSON하세요AmazonTimestreamConsoleFullAccess.

AWS 관리형 정책: AmazonTimestreamFullAccess

사용자, 그룹 및 역할에 `AmazonTimestreamFullAccess`를 연결할 수 있습니다.

정책은 Amazon Timestream에 대한 전체 액세스를 제공합니다. 이 정책은 특정 AWS KMS 작업에 대한 권한도 부여합니다.

권한 세부 정보

이 정책에는 다음 권한이 포함되어 있습니다.

- `Amazon Timestream` – 보안 주체에게 Amazon Timestream에 대한 전체 액세스 권한을 부여합니다.
- `AWS KMS` - 보안 주체가 별칭을 나열하고 키를 설명할 수 있습니다.
- `Amazon S3` - 보안 주체가 모든 Amazon S3 버킷을 나열할 수 있도록 허용합니다.

형식으로 이 정책을 검토하려면 섹션을 참조JSON하세요AmazonTimestreamFullAccess.

AWS 관리형 정책에 대한 Timestream Live Analytics 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 Timestream Live Analytics의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 Timestream Live Analytics 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
AmazonTimestreamReadOnlyAccess - 기존 정책에 대한 업데이트	기존 AmazonTimestreamReadOnlyAccess 관리형 정책에 timestream:DescribeAccountSettings 작업을 추가했습니다. 이 작업은 AWS 계정 설정을 설명하는 데 사용됩니다. 또한 Timestream Live Analytics는 Sid 필드를 추가하여 이 관리형 정책을 업데이트했습니다. 정책 업데이트는 AmazonTimestreamReadOnlyAccess 관리형 정책의 사용에 영향을 주지 않습니다.	2024년 6월 3일
AmazonTimestreamReadOnlyAccess -기존 정책 업데이트	기존 AmazonTimestreamReadOnlyAccess 관리형 정책에 timestream:DescribeBatchLoadTask 및 timestream:ListBatchLoadTasks 작업이 추가되었습니다. 이러한 작업은 배치 로드 작업을 나열하고 설명할 때 사용됩니다. 정책 업데이트는 AmazonTimestreamReadOnlyAccess 관리형 정책의 사용에 영향을 주지 않습니다.	2023년 2월 24일
AmazonTimestreamReadOnlyAccess -기존 정책 업데이트	기존 AmazonTimestreamReadOnlyAccess 관리형 정책에 timestream:Describ	2021년 11월 29일

변경 사항	설명	날짜
	eScheduledQuery 및 timestream:ListSch eduledQueries 작업이 추가되었습니다. 이러한 작업은 기존 예약 쿼리를 나열하고 설명할 때 사용됩니다. 정책 업데이트는 AmazonTim estreamReadOnlyAcc ess 관리형 정책의 사용에 영향을 주지 않습니다.	
AmazonTimestreamCo nsoleFullAccess -기존 정책 업데이트	기존 AmazonTimestreamCo nsoleFullAccess 관리형 정책에 s3:ListAl lMyBuckets 작업을 추가했습니다. 이 작업은 Timestream에 대한 Amazon S3 버킷을 지정하여 마그네틱 스토어 쓰기 오류를 기록할 때 사용됩니다. 정책 업데이트는 AmazonTim estreamConsoleFull Access 관리형 정책의 사용에 영향을 주지 않습니다.	2021년 11월 29일

변경 사항	설명	날짜
AmazonTimestreamFullAccess -기존 정책 업데이트	기존 `AmazonTimestreamFullAccess` 관리형 정책에 `s3:ListAllMyBuckets` 작업을 추가했습니다. 이 작업은 Timestream에 대한 Amazon S3 버킷을 지정하여 마그네틱 스토어 쓰기 오류를 기록할 때 사용됩니다. 정책 업데이트는 `AmazonTimestreamFullAccess` 관리형 정책의 사용에 영향을 주지 않습니다.	2021년 11월 29일
AmazonTimestreamConsoleFullAccess -기존 정책 업데이트	기존 `AmazonTimestreamConsoleFullAccess` 관리형 정책에서 중복 작업을 제거했습니다. 이전에는 이 정책에 중복 작업 이 포함되었습니다`dbqms:DescribeQueryHistory` . 업데이트된 정책은 중복 작업을 제거합니다. 정책 업데이트는 `AmazonTimestreamConsoleFullAccess` 관리형 정책의 사용에 영향을 주지 않습니다.	2021년 4월 23일
Timestream Live Analytics에서 변경 사항 추적 시작	Timestream Live Analytics는 AWS 관리형 정책에 대한 변경 사항을 추적하기 시작했습니다.	2021년 4월 21일

LiveAnalytics 자격 증명 기반 정책 예제를 위한 Amazon Timestream

기본적으로 IAM 사용자 및 역할에는 LiveAnalytics 리소스에 대한 Timestream을 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, CQLSH, AWS CLI또는 를 사용하여 작업을 수행할 수 없습니다 AWS API. IAM 관리자는 사용자 및 역할에 필요한 지정된 리소스에 대해 특정 API 작업을 수행할 수 있는 권한을 부여하는 IAM 정책을 생성해야 합니다. 그런 다음 관리자는 해당 권한을 필요로 하는 IAM 사용자 또는 그룹에 해당 정책을 연결해야 합니다.

이러한 예제 정책 문서를 사용하여 IAM 자격 증명 기반 JSON 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 JSON 탭에서 정책 생성을 참조하세요.

주제

- 정책 모범 사례
- LiveAnalytics 콘솔용 Timestream 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용
- 에 대한 Timestream의 공통 작업 LiveAnalytics
- 태그를 기반으로 LiveAnalytics 리소스 액세스를 위한 타임스트림
- 예약된 쿼리

정책 모범 사례

자격 증명 기반 정책은 계정의 LiveAnalytics 리소스에 대한 Timestream을 생성, 액세스 또는 삭제할 수 있는지 여부를 결정합니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 관리AWS 형 정책 또는 AWS 작업 함수에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 - IAM 정책으로 권한을 설정할 때 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. 를 사용하여 권한을 적용하는 IAM 방법에 대한 자세한 내용은 IAM 사용 설명서의 에서 정책 및 권한을 IAM 참조하세요.
- IAM 정책의 조건을 사용하여 액세스를 추가로 제한 - 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어 정책 조건을 작성하여 를 사용하여 모든 요청을 전송하도

록 지정할 수 있습니다SSL. AWS 서비스와 같은 특정 를 통해 사용되는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 AWS CloudFormation. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

- IAM Access Analyzer를 사용하여 IAM 정책을 검증하여 안전하고 기능적인 권한을 보장합니다. IAM Access Analyzer는 정책이 정책 언어(JSON) 및 IAM 모범 사례를 준수하도록 새 정책 및 기존 IAM 정책을 검증합니다. IAM Access Analyzer는 안전하고 기능적인 정책을 작성하는 데 도움이 되는 100개 이상의 정책 확인 및 실행 가능한 권장 사항을 제공합니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer를 사용한 정책 검증을 참조하세요.
- 다중 인증 필요(MFA) - 에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 MFA 위해 를 AWS 계정켭니다. API 작업을 호출할 MFA 때 를 요구하려면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 를 사용한 보안 API 액세스를 MFA 참조하세요.

의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 의 보안 모범 사례를 IAM IAM참조하세요.

LiveAnalytics 콘솔용 Timestream 사용

의 Timestream LiveAnalytics 은 LiveAnalytics 콘솔용 Amazon Timestream에 액세스하는 데 특정 권한이 필요하지 않습니다. AWS 계정의 LiveAnalytics 리소스에 대한 Timestream에 대한 세부 정보를 나열하고 보려면 읽기 전용 권한이 필요합니다. 최소 필수 권한보다 더 제한적인 자격 증명 기반 정책을 생성하는 경우 콘솔은 해당 정책이 있는 엔터티(IAM 사용자 또는 역할)에 대해 의도한 대로 작동하지 않습니다.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제에서는 IAM 사용자가 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함되어 있습니다 AWS API.

```
{
    "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": "*"
        }
    ]
}
```

에 대한 Timestream의 공통 작업 LiveAnalytics

다음은 Timestream for LiveAnalytics Service에서 일반적인 작업을 허용하는 샘플 IAM 정책입니다.

주제

- 모든 작업 허용
- SELECT 작업 허용
- 여러 리소스에서 SELECT 작업 허용
- 메타데이터 작업 허용
- INSERT 작업 허용
- CRUD 작업 허용
- 리소스를 지정하지 않고 쿼리 취소 및 데이터 선택
- 데이터베이스 생성, 설명, 삭제 및 설명
- 태그별 나열된 데이터베이스 제한{"Owner": "${username}"}
- 데이터베이스의 모든 테이블 나열
- 테이블에서 생성, 설명, 삭제, 업데이트 및 선택
- 테이블별 쿼리 제한

모든 작업 허용

다음은 에 대한 Timestream의 모든 작업을 허용하는 샘플 정책입니다 LiveAnalytics.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:*"
            ],
            "Resource": "*"
        }
    ]
}
```

SELECT 작업 허용

다음 샘플 정책은 특정 리소스에 대한 SELECT스타일 쿼리를 허용합니다.

> **Note**
>
> 를 Amazon 계정 ID<account_ID>로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:DescribeTable",
                "timestream:ListMeasures"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        },
        {
            "Effect": "Allow",
            "Action": [
```

```
                "timestream:DescribeEndpoints",
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

여러 리소스에서 SELECT 작업 허용

다음 샘플 정책은 여러 리소스에 대한 SELECT스타일 쿼리를 허용합니다.

> **Note**
>
> 를 Amazon 계정 ID<account_ID>로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select",
                "timestream:DescribeTable",
                "timestream:ListMeasures"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps",
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps1",
                "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/table/
DevOps2"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:SelectValues",
```

```
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

메타데이터 작업 허용

다음 샘플 정책은 사용자가 메타데이터 쿼리를 수행할 수 있도록 허용하지만 사용자가 의 Timestream 에서 실제 데이터를 읽거나 쓰는 작업을 수행할 수는 없습니다 LiveAnalytics.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:DescribeTable",
                "timestream:ListMeasures",
                "timestream:SelectValues",
                "timestream:ListTables",
                "timestream:ListDatabases",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

INSERT 작업 허용

다음 샘플 정책을 통해 사용자는 계정 에서 에 대한 `INSERT` 작업을 수행할 `database/sampleDB/table/DevOps` 수 있습니다`<account_id>`.

> **Note**
>
> 를 Amazon 계정 ID`<account_ID>`로 바꿉니다.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "timestream:WriteRecords"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:<account_id>:database/sampleDB/table/
DevOps"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

CRUD 작업 허용

다음 샘플 정책을 통해 사용자는 에 대한 Timestream에서 CRUD 작업을 수행할 수 있습니다 LiveAnalytics.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints",
                "timestream:CreateTable",
                "timestream:DescribeTable",
                "timestream:CreateDatabase",
                "timestream:DescribeDatabase",
                "timestream:ListTables",
                "timestream:ListDatabases",
                "timestream:DeleteTable",
                "timestream:DeleteDatabase",
                "timestream:UpdateTable",
```

```
                "timestream:UpdateDatabase"
            ],
            "Resource": "*"
        }
    ]
}
```

리소스를 지정하지 않고 쿼리 취소 및 데이터 선택

다음 샘플 정책을 통해 사용자는 리소스 사양이 필요하지 않은 데이터에 대해 쿼리를 취소하고 `Select` 쿼리를 수행할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:SelectValues",
                "timestream:CancelQuery"
            ],
            "Resource": "*"
        }
    ]
}
```

데이터베이스 생성, 설명, 삭제 및 설명

다음 샘플 정책을 통해 사용자는 데이터베이스를 생성, 설명, 삭제 및 설명할 수 있습니다`sampleDB`.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:CreateDatabase",
                "timestream:DescribeDatabase",
                "timestream:DeleteDatabase",
                "timestream:UpdateDatabase"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB"
        }
```

```
    ]
}
```

태그별 나열된 데이터베이스 제한**{"Owner": "${username}"}**

다음 샘플 정책을 통해 사용자는 키 값 페어 로 태그가 지정된 모든 데이터베이스를 나열할 수 있습니다{"Owner": "${username}"}.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:ListDatabases"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

데이터베이스의 모든 테이블 나열

데이터베이스 의 모든 테이블을 나열하는 다음 샘플 정책입니다sampleDB.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:ListTables"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/"
        }
    ]
}
```

테이블에서 생성, 설명, 삭제, 업데이트 및 선택

다음 샘플 정책을 통해 사용자는 테이블을 생성하고, 테이블을 설명하고, 테이블을 삭제하고, 테이블을 업데이트하고, 데이터베이스의 테이블에 대한 Select 쿼리를 수행할 DevOps 수 있습니다sampleDB.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:CreateTable",
                "timestream:DescribeTable",
                "timestream:DeleteTable",
                "timestream:UpdateTable",
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        }
    ]
}
```

테이블별 쿼리 제한

다음 샘플 정책을 통해 사용자는 데이터베이스 를 제외한 모든 테이블을 쿼리할 DevOps 수 있습니다sampleDB.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/*"
        },
        {
            "Effect": "Deny",
            "Action": [
```

```
                "timestream:Select"
            ],
            "Resource": "arn:aws:timestream:us-east-1:<account_ID>:database/sampleDB/
table/DevOps"
        }
    ]
}
```

태그를 기반으로 LiveAnalytics 리소스 액세스를 위한 타임스트림

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 LiveAnalytics 리소스에 대한 Timestream 액세스를 제어할 수 있습니다. 이 단원에서는 몇 가지 예를 들어보겠습니다.

다음 예제는 테이블의 Owner에 사용자의 사용자 이름 값이 포함된 경우 사용자에게 테이블을 볼 수 있는 권한을 부여하는 정책을 생성하는 방법을 보여 줍니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadOnlyAccessTaggedTables",
            "Effect": "Allow",
            "Action": "timestream:Select",
            "Resource": "arn:aws:timestream:us-east-2:111122223333:database/mydatabase/
table/*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

이 정책을 계정의 IAM 사용자에게 연결할 수 있습니다. 이름이 인 사용자가 LiveAnalytics 테이블의 Timestream을 보richard-roe려고 하면 테이블에 Owner=richard-roe 또는 태그를 지정해야 합니다owner=richard-roe. 그렇지 않으면 액세스가 거부됩니다. 조건 키 이름은 대소문자를 구분하지 않기 때문에 조건 태그 키 Owner는 Owner 및 owner 모두와 일치합니다. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.

다음 정책은 요청에서 전달된 태그에 키Owner와 값이 있는 경우 사용자에게 태그가 포함된 테이블을 생성할 수 있는 권한을 부여합니다username.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "CreateTagTableUser",
            "Effect": "Allow",
            "Action": [
                "timestream:Create",
                "timestream:TagResource"
            ],
            "Resource": "arn:aws:timestream:us-east-2:111122223333:database/mydatabase/
table/*",
            "Condition": {
                "ForAnyValue:StringEquals": {
                    "aws:RequestTag/Owner": "${aws:username}"
                }
            }
        }
    ]
}
```

아래 정책은 env 태그가 dev 또는 로 설정된 모든 데이터베이스DescribeDatabaseAPI에서 를 사용할 수 있도록 허용합니다test.

```
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowDescribeEndpoints",
      "Effect": "Allow",
      "Action": [
        "timestream:DescribeEndpoints"
      ],
      "Resource": "*"
    },
    {
      "Sid": "AllowDevTestAccess",
      "Effect": "Allow",
      "Action": [
        "timestream:DescribeDatabase"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
```

```
          "timestream:tag/env": [
              "dev",
              "test"
          ]
        }
      }
    }
  ]
}
{ "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTagAccessForDevResources",
      "Effect": "Allow",
      "Action": [
        "timestream:TagResource"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "aws:RequestTag/env": [
            "test",
            "dev"
          ]
        }
      }
    }
  ]
}
```

이 정책은 Condition 키를 사용하여 키env와 값이 test, qa또는인 태그를 리소스에 추가할 dev 수 있도록 허용합니다.

예약된 쿼리

목록, 삭제, 업데이트, 실행 ScheduledQuery

다음 샘플 정책을 통해 사용자는 예약된 쿼리를 나열, 삭제, 업데이트 및 실행할 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Effect": "Allow",
            "Action": [
                "timestream:DeleteScheduledQuery",
                "timestream:ExecuteScheduledQuery",
                "timestream:UpdateScheduledQuery",
                "timestream:ListScheduledQueries",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*"
        }
    ]
}
```

CreateScheduledQuery 고객 관리형 KMS 키 사용

다음 샘플 정책을 통해 사용자는 고객 관리형 KMS 키를 사용하여 암호화된 예약된 쿼리를 생성할 수 있습니다.*<keyid for ScheduledQuery>*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws:iam::123456789012:role/ScheduledQueryExecutionRole"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:CreateScheduledQuery",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:DescribeKey",
                "kms:GenerateDataKey"
            ],
```

```
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        }
    ]
}
```

DescribeScheduledQuery 고객 관리형 KMS 키 사용

다음 샘플 정책을 통해 사용자는 고객 관리형 KMS 키를 사용하여 생성된 예약된 쿼리를 설명할 수 있습니다.*<keyid for ScheduledQuery>*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "timestream:DescribeScheduledQuery",
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        }
    ]
}
```

실행 역할 권한(예약된 쿼리에 고객 관리형 KMS 키 사용 및 오류 보고서에 SSE-KMS 사용)

다음 샘플 정책을 예약된 쿼리 암호화에 고객 관리형 KMS 키를 사용하고 오류 보고서에 SSE-KMS 암호화를 CreateScheduledQuery API 사용하는 의 ScheduledQueryExecutionRoleArn 파라미터에 지정된 IAM 역할에 연결합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Action": [
                "kms:GenerateDataKey",
            ],
            "Resource": "arn:aws:kms:us-west-2:123456789012:key/<keyid for
 ScheduledQuery>",
            "Effect": "Allow"
        },
        {
            "Action": [
                "kms:Decrypt"
            ],
            "Resource": [
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for database-1>",
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for database-n>",
                "arn:aws:kms:us-west-2:123456789012:key/<keyid for ScheduledQuery>"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "sns:Publish"
            ],
            "Resource": [
                "arn:aws:sns:us-west-2:123456789012:scheduled-query-notification-topic-
*"
            ],
            "Effect": "Allow"
        },
        {
            "Action": [
                "timestream:Select",
                "timestream:SelectValues",
                "timestream:WriteRecords"
            ],
            "Resource": "*",
            "Effect": "Allow"
        },
        {
            "Action": [
                "s3:PutObject",
                "s3:GetBucketAcl"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::scheduled-query-error-bucket",
                "arn:aws:s3:::scheduled-query-error-bucket/*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

실행 역할 신뢰 관계

다음은 의 ScheduledQueryExecutionRoleArn 파라미터에 지정된 IAM 역할에 대한 신뢰 관계입니다CreateScheduledQueryAPI.

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

계정 내에서 생성된 모든 예약된 쿼리에 대한 액세스 허용

계정 내에서 생성된 모든 예약된 쿼리에 대한 액세스를 허용CreateScheduledQueryAPI하려면 의 ScheduledQueryExecutionRoleArn 파라미터에 지정된 IAM 역할에 다음 샘플 정책을 연결합니다.*Account_ID*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream.amazonaws.com"
            },
```

```
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "Account_ID"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:timestream:us-
west-2:Account_ID:scheduled-query/*"
                }
            }
        }
    ]
}
```

특정 이름으로 예약된 모든 쿼리에 대한 액세스 허용

다음 샘플 정책을 CreateScheduledQuery 의 ScheduledQueryExecutionRoleArn 파라미터에 지정된 IAM 역할에 연결하여 로 시작하는 이름으로 예약된 모든 쿼리에 대한 액세스를 API허용합니다.*Scheduled_Query_Name*, 계정 내 *Account_ID*.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "Account_ID"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:timestream:us-
west-2:Account_ID:scheduled-query/Scheduled_Query_Name*"
                }
            }
        }
    ]
}
```

LiveAnalytics 자격 증명 및 액세스를 위한 Amazon Timestream 문제 해결

다음 정보를 사용하여 및 용 Timestream 작업 시 발생할 수 있는 일반적인 문제를 진단 LiveAnalytics 하고 해결할 수 있습니다IAM.

주제

- 에 대해 Timestream에서 작업을 수행할 권한이 없습니다. LiveAnalytics
- iam을 수행할 권한이 없습니다.PassRole
- 내 AWS 계정 외부의 사람들이 LiveAnalytics 리소스를 위해 Timestream에 액세스하도록 허용하고 싶습니다.

에 대해 Timestream에서 작업을 수행할 권한이 없습니다. LiveAnalytics

에 작업을 수행할 권한이 없다고 AWS Management Console 표시되면 관리자에게 문의하여 지원을 받아야 합니다. 관리자는 로그인 보안 인증 정보를 제공한 사람입니다.

다음 예제 오류는 `mateojackson` IAM 사용자가 콘솔을 사용하여 에 대한 세부 정보를 보려고 할 때 발생합니다.*`table`* 에는 테이블에 대한 `timestream:`*`Select`* 권한이 없습니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 timestream:Select on resource: mytable
```

이 경우 Mateo는 *`mytable`* 작업을 사용하여 `timestream:`*`Select`* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

iam을 수행할 권한이 없습니다.PassRole

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 발생하면 에 대한 Timestream에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다 LiveAnalytics.

일부는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 기존 역할을 해당 서비스에 전달할 수 있도록 AWS 서비스 허용합니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 이름이 인 IAM 사용자가 콘솔을 사용하여 의 Timestream에서 작업을 수행하려고 `marymajor` 할 때 발생합니다 LiveAnalytics. 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 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 AWS 계정 외부의 사람들이 LiveAnalytics 리소스를 위해 Timestream에 액세스하도록 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(ACLs)을 지원하는 서비스의 경우 이러한 정책을 사용하여 사용자에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- 용 Timestream이 이러한 기능을 LiveAnalytics 지원하는지 알아보려면 섹션을 참조하세요용 Amazon Timestream의 LiveAnalytics 작동 방식 IAM.
- 소유 AWS 계정 한 의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유 AWS 계정 한 다른 의 IAM 사용자에게 액세스 권한 제공을 참조하세요.
- 타사에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정 소유 에 대한 액세스 권한 제공을 AWS 계정참조하세요.
- 자격 증명 페더레이션을 통해 액세스를 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부 인증 사용자(자격 증명 페더레이션)에 대한 액세스 제공을 참조하세요.
- 교차 계정 액세스를 위한 역할 및 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 에서 교차 계정 리소스 액세스를 IAM 참조하세요.

에 대한 Timestream의 로깅 및 모니터링 LiveAnalytics

모니터링은 LiveAnalytics 및 AWS 솔루션을 위한 Timestream의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. 다중 포인트 장애가 발생할 경우 더 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집해야 합니다. 그러나 에 대한 Timestream 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 생성 LiveAnalytics해야 합니다.

- 모니터링의 목표
- 모니터링할 리소스

- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

다음 단계는 다양한 시간과 다양한 로드 조건에서 LiveAnalytics 성능을 측정하여 환경의 성능을 위한 정상 Timestream의 기준을 설정하는 것입니다. 에 대한 Timestream을 모니터링할 때 과거 모니터링 데이터를 LiveAnalytics저장하여 현재 성능 데이터와 비교하고, 정상적인 성능 패턴과 성능 이상을 식별하고, 문제를 해결하기 위한 방법을 고안할 수 있습니다.

기준선을 설정하려면 최소한 다음 항목을 모니터링해야 합니다.

- 시스템 오류 - 요청으로 인해 오류가 발생했는지 확인할 수 있습니다.

주제

- 모니터링 도구
- 를 사용한 호출에 대한 LiveAnalytics API Timestream 로깅 AWS CloudTrail

모니터링 도구

AWS 는 에 대한 Timestream을 모니터링하는 데 사용할 수 있는 다양한 도구를 제공합니다 LiveAnalytics. 이들 도구 중에는 모니터링을 자동으로 수행하도록 구성할 수 있는 도구도 있지만, 수동 작업이 필요한 도구도 있습니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

주제

- 자동 모니터링 도구
- 수동 모니터링 도구

자동 모니터링 도구

다음과 같은 자동 모니터링 도구를 사용하여 Timestream을 모니터링하고 문제가 발생할 경우 LiveAnalytics 보고할 수 있습니다.

- Amazon CloudWatch 경보 - 지정한 기간 동안 단일 지표를 관찰하고 여러 기간 동안 지정된 임계값을 기준으로 지표 값을 기반으로 하나 이상의 작업을 수행합니다. 작업은 Amazon Simple

Notification Service(Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책으로 전송되는 알림입니다. CloudWatch 알람은 특정 상태라는 이유만으로 작업을 호출하지 않습니다. 상태가 변경되어 지정된 기간 동안 유지되어야 합니다. 자세한 내용은 Amazon을 사용한 모니터링 CloudWatch 단원을 참조하십시오.

수동 모니터링 도구

에 대한 Timestream 모니터링의 또 다른 중요한 부분은 CloudWatch 경보 LiveAnalytics 가 다루지 않는 항목을 수동으로 모니터링하는 것입니다. LiveAnalytics, CloudWatch Trusted Advisor및 기타 AWS Management Console 대시보드 at-a-glance의 Timestream은 AWS 환경의 상태를 보여줍니다.

- CloudWatch 홈 페이지에는 다음이 표시됩니다.
  - 현재 경보 및 상태
  - 경보 및 리소스 그래프
  - 서비스 상태

  또한 CloudWatch 를 사용하여 다음을 수행할 수 있습니다.
  - 맞춤 대시보드를 생성하여 관심 있는 서비스 모니터링
  - 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악
  - 모든 AWS 리소스 지표 검색 및 찾아보기
  - 문제에 대해 알려주는 경보 생성 및 편집

를 사용한 호출에 대한 LiveAnalytics API Timestream 로깅 AWS CloudTrail

에 대한 Timestream LiveAnalytics 은 Timestream for . LiveAnalytics CloudTrail captures Data Definition Language(DDL)가 에 대한 Timestream을 이벤트 LiveAnalytics 로 API 요청하는 사용자, 역할 또는 AWS 서비스의 작업 레코드를 AWS CloudTrail제공하는 서비스인 와 통합됩니다. 캡처되는 호출에는 LiveAnalytics 콘솔의 경우 Timestream의 호출, 작업을 위한 Timestream의 LiveAnalytics API 코드 호출이 포함됩니다. 추적을 생성하는 경우 Timestream for 에 CloudTrail 대한 이벤트를 포함하여 Amazon Simple Storage Service(Amazon S3) 버킷에 이벤트를 지속적으로 전송할 수 있습니다 LiveAnalytics. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록 에서 최신 이벤트를 볼 수 있습니다. 에서 수집한 정보를 사용하여 에 대해 Timestream에 수행된 요청 LiveAnalytics, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 CloudTrail수 있습니다.

에 대한 자세한 내용은 AWS CloudTrail 사용 설명서 섹션을 CloudTrail참조하세요.

의 LiveAnalytics 정보에 대한 시간 흐름 CloudTrail

CloudTrail 는 AWS 계정을 생성할 때 계정에서 활성화됩니다. 에 대한 Timestream에서 활동이 발생하면 LiveAnalytics해당 활동은 CloudTrail 이벤트 기록 의 다른 AWS 서비스 이벤트와 함께 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 이벤트 기록을 사용하여 CloudTrail 이벤트 보기를 참조하세요.

> **Warning**
>
> 현재 의 Timestream은 모든 관리 및 `Query` API 작업에 대한 CloudTrail 이벤트를 LiveAnalytics 생성하지만 `WriteRecords` 및 `DescribeEndpoints` 에 대한 이벤트는 생성하지 않습니다APIs.

에 대한 Timestream 이벤트를 포함하여 AWS 계정의 이벤트에 대한 지속적인 레코드를 위해 추적을 LiveAnalytics생성합니다. 추적을 사용하면 CloudTrail 가 Amazon S3 버킷에 로그 파일을 전달할 수 있습니다. 기본적으로 콘솔에서 추적을 생성하면 추적이 모든 AWS 리전에 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 기록하고 지정한 Amazon S3 버킷에 로그 파일을 전달합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 작업하도록 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 리전에서 CloudTrail 로그 파일 수신
- 여러 계정에서 CloudTrail 로그 파일 수신
- 데이터 이벤트 로깅

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 요청이 이루어졌는지 여부
- 역할 또는 페더레이션 사용자에 대한 임시 보안 자격 증명을 사용하여 요청이 생성되었는지 여부
- 다른 AWS 서비스에서 요청을 했는지 여부

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

Query API 이벤트의 경우:

- 모든 이벤트를 수신하는 추적을 생성하거나 LiveAnalytics 리소스 유형 AWS::Timestream::Database 또는 에 대해 Timestream으로 이벤트를 선택합니다AWS::Timestream::Table.
- Query API 데이터베이스 또는 테이블에 액세스하지 않거나 형식이 잘못된 쿼리 문자열로 인해 검증 예외가 발생하는 요청은 CloudTrail 리소스 유형 AWS::Timestream::Database 및 ARN 값으로 에 기록됩니다.

  ```
  arn:aws:timestream:(region):(accountId):database/NO_RESOURCE_ACCESSED
  ```

  이러한 이벤트는 리소스 유형이 인 이벤트를 수신하는 추적에만 전달됩니다AWS::Timestream::Database.

Amazon Timestream Live Analytics의 복원력

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 기반으로 구축됩니다. AWS 리전은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결된 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라 섹션을 참조하세요.

를 통해 사용할 수 있는 Timestream의 데이터 보호 기능에 대한 자세한 내용은 섹션을 AWS Backup참조하세요작업 AWS Backup.

Amazon Timestream Live Analytics의 인프라 보안

관리형 서비스인 Amazon Timestream Live Analytics는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안 절차에 의해 보호됩니다. https://d0.awsstatic.com/whitepapers/Security/AWS_Security_Whitepaper.pdf

AWS 게시된 API 호출을 사용하여 네트워크를 통해 Timestream Live Analytics에 액세스합니다. 클라이언트는 전송 계층 보안(TLS) 1.0 이상을 지원해야 합니다. TLS 1.2 이상을 사용하는 것이 좋습니다. 또한 클라이언트는 Ephemeral Diffie-Hellman(PFS) 또는 Elliptic Curve Ephemeral Diffie-Hellman()과

같은 완벽한 순방향 보안(DHE)을 갖춘 암호 제품군을 지원해야 합니다ECDHE. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 액세스 키 ID와 IAM 보안 주체와 연결된 보안 액세스 키를 사용하여 요청에 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 인증을 생성하여 요청에 서명할 수 있습니다.

Timestream Live Analytics는 트래픽이 Timestream Live Analytics 인스턴스가 있는 특정 AWS 리전으로 격리되도록 설계되었습니다.

Timestream의 구성 및 취약성 분석

구성 및 IT 제어는 고객과 AWS 고객 간의 공동 책임입니다. 자세한 내용은 AWS 공동 책임 모델 을 참조하세요. 책임 공유 모델 외에도 Timestream for LiveAnalytics users는 다음 사항에 유의해야 합니다.

- 관련 클라이언트 측 종속성을 사용하여 클라이언트 애플리케이션을 패치하는 것은 고객의 책임입니다.
- 고객은 적절한 경우 침투 테스트를 고려해야 합니다(https://aws.amazon.com/security/침투 테스트/ 참조).

에 대한 Timestream의 인시던트 응답 LiveAnalytics

LiveAnalytics 서비스 인시던트에 대한 Amazon Timestream은 Personal Health Dashboard 에 보고됩니다. 대시보드 및 에 대해 자세히 알아볼 수 AWS Health 있습니다.

의 Timestream은 를 사용한 보고를 LiveAnalytics 지원합니다 AWS CloudTrail. 자세한 내용은 를 사용한 호출에 대한 LiveAnalytics API Timestream 로깅 AWS CloudTrail 단원을 참조하십시오.

VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트 를 생성 LiveAnalytics 하여 VPC 와 Amazon Timestream 간에 에 대한 프라이빗 연결을 설정할 수 있습니다. 인터페이스 엔드포인트는 인터넷 게이트웨이AWS PrivateLink, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 의 Timestream에 LiveAnalytics APIs 비공개로 액세스할 수 있는 기술인 로 구동됩니다. 의 인스턴스는 의 Timestream과 통신하는 데 퍼블릭 IP 주소가 필요하지 VPC 않습니다 LiveAnalytics APIs. 에 대한 VPC LiveAnalytics 와 Timestream 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 탄력적 네트워크 인터페이스로 표현됩니다. 인터페이스 VPC 엔드포인트에 대한 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.

LiveAnalytics 및 VPC 엔드포인트에 대한 Timestream을 시작하기 위해 VPC 엔드포인트 LiveAnalytics에 대한 Timestream의 특정 고려 사항, 에 대한 Timestream의 인터페이스 VPC 엔드포인트 생성 LiveAnalytics, 에 대한 Timestream의 VPC 엔드포인트 정책 생성 LiveAnalytics, VPC 엔드포인트에 대한 Timestream 클라이언트 사용(쓰기 또는 쿼리용SDK)에 대한 정보를 제공했습니다.

주제

- VPC 엔드포인트가 Timestream과 작동하는 방식
- 에 대한 Timestream용 인터페이스 VPC 엔드포인트 생성 LiveAnalytics
- 에 대한 Timestream의 VPC 엔드포인트 정책 생성 LiveAnalytics

VPC 엔드포인트가 Timestream과 작동하는 방식

Timestream 쓰기 또는 Timestream 쿼리 에 액세스하기 위해 VPC 엔드포인트를 생성하면 SDK모든 요청이 Amazon 네트워크 내의 엔드포인트로 라우팅되고 퍼블릭 인터넷에 액세스하지 않습니다. 보다 구체적으로, 요청은 지정된 리전에 대해 계정이 매핑된 셀의 쓰기 및 쿼리 엔드포인트로 라우팅됩니다. Timestream의 셀룰러 아키텍처 및 셀별 엔드포인트에 대한 자세한 내용은 섹션을 참조하세요셀룰러 아키텍처. 예를 들어 계정이 `cell1`에서 에 매핑되었고 쓰기(`ingest-cell1.timestream.us-west-2.amazonaws.com`) `us-west-2`및 쿼리()에 대한 VPC 인터페이스 엔드포인트를 설정했다고 가정해 보겠습니다`query-cell1.timestream.us-west-2.amazonaws.com`. 이 경우 이러한 엔드포인트를 사용하여 전송된 모든 쓰기 요청은 Amazon 네트워크 내에 완전히 남게 되며 퍼블릭 인터넷에 액세스하지 않습니다.

Timestream VPC 엔드포인트에 대한 고려 사항

Timestream용 VPC 엔드포인트를 생성할 때 다음 사항을 고려하세요.

- 용 Timestream에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 속성 및 제한을 검토해야 LiveAnalytics합니다.
- 의 Timestream은 에서 모든 API 작업에 대한 호출을 LiveAnalytics 지원합니다VPC.
- VPC 엔드포인트 정책은 의 Timestream에 지원됩니다 LiveAnalytics. 기본적으로 엔드포인트를 통해 에 대한 Timestream에 대한 전체 액세스 LiveAnalytics 가 허용됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 사용하여 서비스에 대한 액세스 제어를 참조하세요.

- Timestream의 아키텍처로 인해 쓰기 및 쿼리 작업에 모두 액세스하려면 각 에 대해 하나씩 두 개의 VPC 인터페이스 엔드포인트를 생성해야 합니다SDK. 또한 셀 엔드포인트를 지정해야 합니다(매핑된 Timestream 셀에 대한 엔드포인트만 생성할 수 있음). 자세한 정보는 이 가이드의 Timestream용 인터페이스 VPC 엔드포인트 생성 LiveAnalytics 섹션에서 확인할 수 있습니다.

이제 Timestream for 가 VPC 엔드포인트에서 LiveAnalytics 작동하는 방법을 이해했으므로 Timestream for 에 대한 인터페이스 VPC 엔드포인트를 생성합니다 LiveAnalytics.

에 대한 Timestream용 인터페이스 VPC 엔드포인트 생성 LiveAnalytics

Amazon VPC 콘솔 또는 AWS Command Line Interface ()를 사용하여 Timestream for LiveAnalytics service에 대한 인터페이스 VPC 엔드포인트를 생성할 수 있습니다AWS CLI. Timestream에 대한 VPC 엔드포인트를 생성하려면 아래 설명된 Timestream별 단계를 완료합니다.

> **Note**
>
> 아래 단계를 완료하기 전에 Timestream VPC 엔드포인트에 대한 특정 고려 사항을 이해해야 합니다.

Timestream 셀을 사용하여 VPC 엔드포인트 서비스 이름 구성

Timestream의 고유한 아키텍처로 인해 각 SDK (쓰기 및 쿼리)에 대해 별도의 VPC 인터페이스 엔드포인트를 생성해야 합니다. 또한 Timestream 셀 엔드포인트를 지정해야 합니다(매핑된 Timestream 셀에 대한 엔드포인트만 생성할 수 있음). 인터페이스 VPC 엔드포인트를 사용하여 내 에서 Timestream에 직접 연결하려면 아래 단계를 VPC완료하세요.

1. 먼저 사용 가능한 Timestream 셀 엔드포인트를 찾습니다. 사용 가능한 셀 엔드포인트를 찾으려면 `DescribeEndpoints` 작업(쓰기 및 쿼리를 통해 사용 가능APIs)을 사용하여 Timestream 계정에서 사용 가능한 셀 엔드포인트를 나열합니다. 자세한 내용은 예제를 참조하세요.
2. 사용할 셀 엔드포인트를 선택한 후 Timestream 쓰기 또는 쿼리 에 대한 VPC 인터페이스 엔드포인트 문자열을 생성합니다API.
   - 쓰기의 경우API:

     ```
     com.amazonaws.<region>.timestream.ingest-<cell>
     ```

   - 쿼리 의 경우API:

```
com.amazonaws.<region>.timestream.query-<cell>
```

여기서 각 항목은 다음과 같습니다.*<region>* 는 유효한 AWS 리전 코드이며 *<cell>* 는 DescribeEndpoints 작업 에서 엔드포인트 객체에 반환된 셀 엔드포인트 주소(예: `cell1` 또는 `cell2`) 중 하나입니다. 자세한 내용은 예제를 참조하세요.

3. 이제 VPC 엔드포인트 서비스 이름을 구성했으므로 인터페이스 엔드포인트 를 생성합니다. VPC 엔드포인트 서비스 이름을 제공하라는 메시지가 표시되면 2단계에서 구성한 VPC 엔드포인트 서비스 이름을 사용합니다.

예: VPC 엔드포인트 서비스 이름 구성

다음 예제에서는 `us-west-2` 리전에서 AWS CLI 쓰기를 사용하여 API에서 `DescribeEndpoints` 작업이 실행됩니다.

```
aws timestream-write describe-endpoints --region us-west-2
```

이 명령은 다음 출력을 반환합니다.

```
{
    "Endpoints": [
        {
            "Address": "ingest-cell1.timestream.us-west-2.amazonaws.com",
            "CachePeriodInMinutes": 1440
        }
    ]
}
```

이 경우 *cell1* 는 *<cell>* , 및 *us-west-2* 는 *<region>*. 따라서 결과 VPC 엔드포인트 서비스 이름은 다음과 같습니다.

```
com.amazonaws.us-west-2.timestream.ingest-cell1
```

이제 용 Timestream에 대한 인터페이스 VPC 엔드포인트를 생성했으므로 용 Timestream에 대한 엔드포인트 정책을 LiveAnalytics생성합니다. VPC LiveAnalytics

에 대한 Timestream의 VPC 엔드포인트 정책 생성 LiveAnalytics

의 Timestream에 대한 액세스를 제어하는 엔드포인트 정책을 VPC 엔드포인트에 연결할 수 있습니다 LiveAnalytics. 이 정책은 다음 정보를 지정합니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 사용하여 서비스에 대한 액세스 제어를 참조하세요.

예: LiveAnalytics 작업에 대한 Timestream의 VPC 엔드포인트 정책

다음은 의 Timestream에 대한 엔드포인트 정책의 예입니다 LiveAnalytics. 엔드포인트에 연결되면 이 정책은 모든 리소스의 모든 보안 주체에 대해 나열된 LiveAnalytics 작업에 대한 Timestream(이 경우 `ListDatabases`)에 대한 액세스 권한을 부여합니다.

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "timestream:ListDatabases"
         ],
         "Resource":"*"
      }
   ]
}
```

용 Amazon Timestream의 보안 모범 사례 LiveAnalytics

용 Amazon Timestream LiveAnalytics 은 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 사용하십시오.

주제

- LiveAnalytics 예방 보안 모범 사례를 위한 타임스트림

LiveAnalytics 예방 보안 모범 사례를 위한 타임스트림

다음 모범 사례는 의 Timestream에서 보안 인시던트를 예측하고 방지하는 데 도움이 될 수 있습니다 LiveAnalytics.

저장 중 암호화

휴지 상태의 LiveAnalytics 에 대한 타임스트림은 AWS Key Management Service (AWS KMS)에 저장된 암호화 키를 사용하여 테이블에 저장된 모든 사용자 데이터를 암호화합니다. 이를 통해 기본 스토리지에 대한 무단 액세스로부터 데이터를 보호하여 데이터 보호 계층을 추가로 제공합니다.

의 Timestream LiveAnalytics 은 단일 서비스 기본 키(AWS 소유CMK)를 사용하여 모든 테이블을 암호화합니다. 이 키가 없으면 자동으로 생성됩니다. 서비스 기본 키는 비활성화할 수 없습니다. 자세한 내용은 저장 시 LiveAnalytics 암호화를 위한 Timestream을 참조하세요.

IAM 역할을 사용하여 에 대한 Timestream 액세스를 인증합니다. LiveAnalytics

사용자, 애플리케이션 및 기타 AWS 서비스가 의 Timestream에 액세스하려면 요청에 유효한 AWS 보안 인증 AWS API 정보를 포함해야 LiveAnalytics합니다. 자격 AWS 증명을 애플리케이션 또는 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 서비스 및 리소스에 액세스하는 데 사용할 수 있는 임시 액세스 AWS 키를 얻을 수 있습니다.

자세한 내용은 IAM 역할을 참조하십시오.

LiveAnalytics 기본 권한 부여에 Timestream IAM 정책 사용

권한을 부여할 때 권한을 부여할 사람, 권한을 APIs 부여할 Timestream LiveAnalytics 및 해당 리소스에 대해 허용하려는 특정 작업을 결정합니다. 최소 권한을 구현하는 것이 오류 또는 악의적인 의도로 인해 발생할 수 있는 보안 위험과 영향을 줄일 수 있는 비결입니다.

권한 정책을 IAM 자격 증명(즉, 사용자, 그룹 및 역할)에 연결하여 리소스에 대해 LiveAnalytics Timestream에서 작업을 수행할 수 있는 권한을 부여합니다.

이를 위해 다음 정책을 사용할 수 있습니다.

- AWS 관리형(미리 정의된) 정책
- 고객 관리형 정책
- 태그 기반 권한 부여

클라이언트 측 암호화 참조

의 Timestream에 민감한 데이터 또는 기밀 데이터를 저장하는 경우 LiveAnalytics수명 주기 동안 데이터가 보호되도록 해당 데이터를 오리진에 최대한 가깝게 암호화해야 할 수 있습니다. 전송 중 및 저장 중 민감한 데이터를 암호화하면 타사에서 일반 텍스트 데이터를 사용할 수 없습니다.

다른 서비스와 함께 사용

용 Amazon Timestream은 다양한 AWS 서비스 및 인기 있는 타사 도구와 LiveAnalytics 통합됩니다. 현재 용 Timestream은 다음과의 통합을 LiveAnalytics 지원합니다.

주제

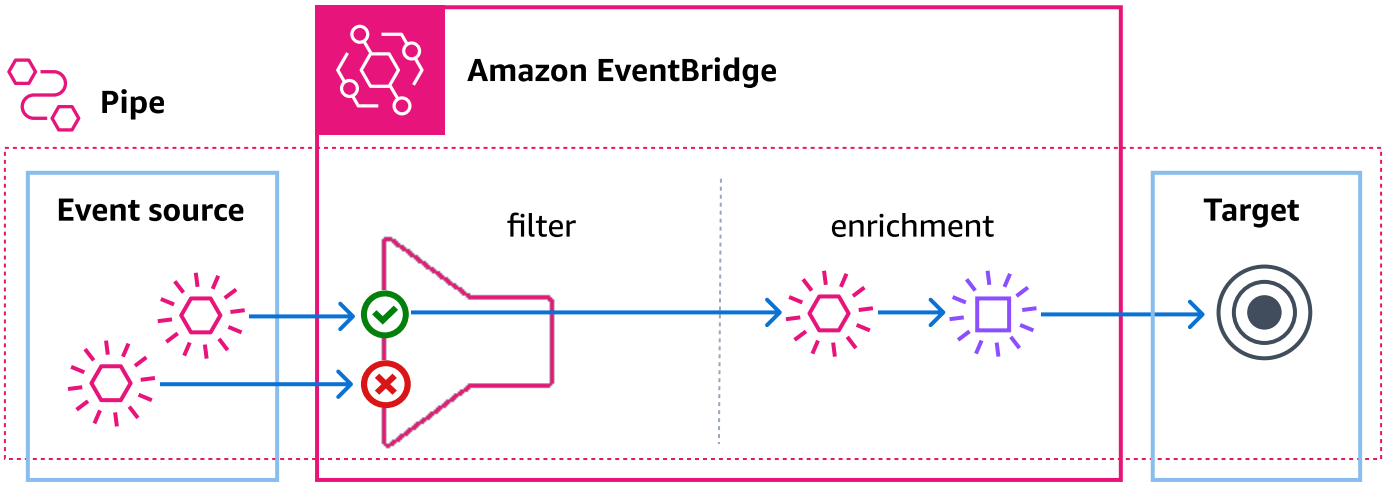
- Amazon DynamoDB
- AWS Lambda
- AWS IoT Core
- Amazon Managed Service for Apache Flink
- Amazon Kinesis
- Amazon MQ
- Amazon MSK
- Amazon QuickSight
- Amazon SageMaker
- Amazon SQS
- DBeaver 를 사용하여 Amazon Timestream 작업
- Grafana
- SquaredUp 를 사용하여 Amazon Timestream 작업
- 오픈 소스 Telegraf
- JDBC
- ODBC
- VPC 엔드포인트(AWS PrivateLink)

Amazon DynamoDB

EventBridge 파이프를 사용하여 DynamoDB 데이터 전송 Timestream

EventBridge 파이프를 사용하여 DynamoDB 스트림에서 LiveAnalytics 테이블용 Amazon Timestream으로 데이터를 전송할 수 있습니다.

파이프는 고급 변환 및 보강을 지원하여 지원되는 소스와 대상 간의 통합을 위한 point-to-point 것입니다. 파이프는 이벤트 기반 아키텍처를 개발할 때 전문 지식 및 통합 코드의 필요성을 줄입니다. 파이프를 설정하려면 소스를 선택하고, 선택적 필터링을 추가하고, 선택적 보강을 정의하고, 이벤트 데이터의 대상을 선택합니다.



EventBridge 파이프에 대한 자세한 내용은 EventBridge 사용 설명서의 EventBridge 파이프를 참조하세요. LiveAnalytics 테이블용 Amazon Timestream에 이벤트를 전달하도록 파이프를 구성하는 방법에 대한 자세한 내용은 EventBridge 파이프 대상 세부 정보 섹션을 참조하세요.

AWS Lambda

의 Timestream과 상호 작용하는 Lambda 함수를 생성할 수 있습니다 LiveAnalytics. 예를 들어 정기적으로 실행되는 Lambda 함수를 생성하여 Timestream에서 쿼리를 실행하고 하나 이상의 기준을 충족하는 쿼리 결과에 따라 SNS 알림을 보낼 수 있습니다. Lambda에 대한 자세한 내용은 AWS Lambda 설명서 섹션을 참조하세요.

주제

- Python LiveAnalytics 에서 용 Amazon Timestream을 사용하여 AWS Lambda 함수 빌드
- LiveAnalytics 에서 에 대한 Amazon Timestream을 사용하여 AWS Lambda 함수 빌드 JavaScript

- Go LiveAnalytics 와 함께 용 Amazon Timestream을 사용하여 AWS Lambda 함수 빌드
- C# LiveAnalytics 에서 에 대한 Amazon Timestream을 사용하여 AWS Lambda 함수 빌드

Python LiveAnalytics 에서 용 Amazon Timestream을 사용하여 AWS Lambda 함수 빌드

Python LiveAnalytics 을 사용하여 용 Amazon Timestream을 사용하여 AWS Lambda 함수를 빌드하려면 아래 단계를 따릅니다.

1. 에서 에 설명된 대로 가 Timestream Service에 액세스하는 데 필요한 권한을 부여한다고 가정할 Lambda에 대한 IAM 역할을 생성합니다 LiveAnalytics 액세스를 위한 Timestream 제공.
2. IAM 역할의 신뢰 관계를 편집하여 Lambda 서비스를 추가합니다. AWS Lambda가 수임할 수 있도록 아래 명령을 사용하여 기존 역할을 업데이트할 수 있습니다.

   a. 신뢰 정책 문서 생성:

```
cat > Lambda-Role-Trust-Policy.json << EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": [
          "lambda.amazonaws.com"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

   b. 신뢰 문서를 사용하여 이전 단계에서 역할 업데이트

```
aws iam update-assume-role-policy --role-name <name_of_the_role_from_step_1> --
policy-document file://Lambda-Role-Trust-Policy.json
```

관련 참조는 TimestreamWrite 및 에 있습니다TimestreamQuery.

LiveAnalytics 에서 에 대한 Amazon Timestream을 사용하여 AWS Lambda 함수 빌드 JavaScript

에서 용 Amazon Timestream을 사용하여 AWS Lambda 함수 LiveAnalytics 를 빌드하려면 여기에 설명된 지침을 JavaScript따르세요.

관련 참조는 v3의 경우 Timestream Write Client에 AWS SDK, JavaScript v3의 경우 Timestream Query Client AWS SDK에 있습니다 JavaScript .

Go LiveAnalytics 와 함께 용 Amazon Timestream을 사용하여 AWS Lambda 함수 빌드

Go와 LiveAnalytics 함께 용 Amazon Timestream을 사용하여 AWS Lambda 함수를 빌드하려면 여기에 설명된 지침을 따르세요.

관련 참조는 timestreamwrite 및 timestreamquery 에 있습니다.

C# LiveAnalytics 에서 에 대한 Amazon Timestream을 사용하여 AWS Lambda 함수 빌드

Amazon Timestream for with C#을 사용하여 AWS Lambda 함수 LiveAnalytics 를 빌드하려면 여기에 설명된 지침을 따르세요.

관련 참조는 Amazon.TimestreamWrite 및 Amazon.에 있습니다.TimestreamQuery

AWS IoT Core

IoT AWS Core를 사용하여 IoT 디바이스에서 데이터를 수집하고 IoT Core 규칙 작업을 통해 Amazon Timestream으로 데이터를 라우팅할 수 있습니다. AWS IoT 규칙 작업은 규칙이 트리거될 때 수행할 작업을 지정합니다. Amazon Timestream 테이블, Amazon DynamoDB 데이터베이스로 데이터를 전송하고 AWS Lambda 함수를 호출하는 작업을 정의할 수 있습니다.

IoT 규칙의 Timestream 작업은 수신 메시지의 데이터를 Timestream에 직접 삽입하는 데 사용됩니다. 작업은 IoT Core SQL 문 결과를 구문 분석하고 데이터를 Timestream에 저장합니다. 반환된 SQL 결과 세트의 필드 이름은 measure::name으로 사용되며 필드 값은 measure::value입니다.

예를 들어 SQL 문과 샘플 메시지 페이로드를 고려해 보겠습니다.

```
SELECT temperature, humidity from 'iot/topic'
```

```
{
  "dataFormat": 5,
  "rssi": -88,
  "temperature": 24.04,
  "humidity": 43.605,
  "pressure": 101082,
  "accelerationX": 40,
  "accelerationY": -20,
  "accelerationZ": 1016,
  "battery": 3007,
  "txPower": 4,
  "movementCounter": 219,
  "device_id": 46216,
  "device_firmware_sku": 46216
}
```

위 SQL 문으로 Timestream에 대한 IoT Core 규칙 작업이 생성되면 측정 이름 온도 및 습도와 각각 24.04 및 43.605의 측정 값을 사용하여 두 개의 레코드가 Timestream에 추가됩니다.

SELECT 문에서 AS 연산자를 사용하여 Timestream에 추가되는 레코드의 측정값 이름을 수정할 수 있습니다. 아래 SQL 문은 온도 대신 메시지 이름 임시로 레코드를 생성합니다.

측정값의 데이터 유형은 메시지 페이로드 값의 데이터 유형에서 추론됩니다. JSON 정수, 이중, 부울 및 문자열과 같은 데이터 유형은 VARCHAR 각각 , BIGINT, BOOLEAN, DOUBLE의 Timestream 데이터 유형에 매핑됩니다. 또한 cast() 함수를 사용하여 특정 데이터 유형에 데이터를 강제로 적용할 수도 있습니다. 측정값의 타임스탬프를 지정할 수 있습니다. 타임스탬프를 비워 두면 항목이 처리된 시간이 사용됩니다.

자세한 내용은 Timestream 규칙 작업 설명서를 참조하세요.

Timestream에 데이터를 저장하는 IoT Core 규칙 작업을 생성하려면 다음 단계를 따르세요.

주제

- 사전 조건
- 콘솔 사용
- 사용 CLI
- 샘플 애플리케이션
- 비디오 자습서

사전 조건

1. 에 설명된 지침을 사용하여 Amazon Timestream에서 데이터베이스를 생성합니다 데이터베이스 생성.
2. 에 설명된 지침을 사용하여 Amazon Timestream에서 테이블을 생성합니다테이블 생성.

콘솔 사용

1. AWS IoT Core용 AWS 관리 콘솔을 사용하여 관리 > 메시지 라우팅 > 규칙을 클릭한 다음 규칙 생성을 클릭하여 규칙을 생성합니다.
2. 규칙 이름을 원하는 이름으로 설정하고 를 아래 표시된 텍스트SQL로 설정합니다.

```
SELECT temperature as temp, humidity from 'iot/topic'
```

3. 작업 목록에서 Timestream을 선택합니다.
4. Timestream에 데이터를 쓸 역할과 함께 Timestream 데이터베이스, 테이블 및 차원 이름을 지정합니다. 역할이 없는 경우 역할 생성을 클릭하여 생성할 수 있습니다.
5. 규칙을 테스트하려면 여기에 표시된 지침을 따르세요.

사용 CLI

AWS 명령줄 인터페이스(AWS CLI)를 설치하지 않은 경우 여기에서 설치합니다.

1. 다음 규칙 페이로드를 timestream_rule.json이라는 JSON 파일에 저장합니다. Replace *arn:aws:iam::123456789012:role/TimestreamRole* Amazon Timestream에 데이터를 저장할 수 있는 AWS IoT 액세스 권한을 부여하는 역할 arn

```
{
    "actions": [
            {
                "timestream": {
                    "roleArn": "arn:aws:iam::123456789012:role/TimestreamRole",
                    "tableName": "devices_metrics",
                    "dimensions": [
                        {
                            "name": "device_id",
                            "value": "${clientId()}"
                        },
```

```
                        {
                            "name": "device_firmware_sku",
                            "value": "My Static Metadata"
                        }
                    ],
                    "databaseName": "record_devices"
                }
            }
    ],
    "sql": "select * from 'iot/topic'",
    "awsIotSqlVersion": "2016-03-23",
    "ruleDisabled": false
}
```

2. 다음 명령을 사용하여 주제 규칙 생성

```
aws iot create-topic-rule --rule-name timestream_test --topic-rule-payload file://
<path/to/timestream_rule.json> --region us-east-1
```

3. 다음 명령을 사용하여 주제 규칙의 세부 정보 검색

```
aws iot get-topic-rule --rule-name timestream_test
```

4. timestream_msg.json이라는 파일에 다음 메시지 페이로드를 저장합니다.

```
{
  "dataFormat": 5,
  "rssi": -88,
  "temperature": 24.04,
  "humidity": 43.605,
  "pressure": 101082,
  "accelerationX": 40,
  "accelerationY": -20,
  "accelerationZ": 1016,
  "battery": 3007,
  "txPower": 4,
  "movementCounter": 219,
  "device_id": 46216,
  "device_firmware_sku": 46216
}
```

5. 다음 명령을 사용하여 규칙을 테스트합니다.

```
aws iot-data publish --topic 'iot/topic' --payload file://<path/to/
timestream_msg.json>
```

샘플 애플리케이션

AWS IoT Core와 함께 Timestream 사용을 시작하는 데 도움이 되도록 주제 규칙을 생성하는 데 필요한 아티팩트를 AWS IoT Core 및 Timestream에 생성하는 완전한 기능의 샘플 애플리케이션과 주제에 데이터를 게시하는 샘플 애플리케이션을 만들었습니다.

1. 의 지침에 따라 AWS IoT Core 통합을 위한 샘플 애플리케이션의 GitHub 리포지토리 복제 GitHub
2. 의 지침에 따라 AWS CloudFormation 템플릿을 README 사용하여 Amazon Timestream 및 AWS IoT Core에서 필요한 아티팩트를 생성하고 샘플 메시지를 주제에 게시합니다.

비디오 자습서

이 동영상에서는 IoT Core가 Timestream과 작동하는 방법을 설명합니다.

Amazon Managed Service for Apache Flink

Apache Flink를 사용하여 Amazon Managed Service for Apache Flink, Amazon , MSKApache Kafka 및 기타 스트리밍 기술에서 의 Amazon Timestream으로 시계열 데이터를 직접 전송할 수 있습니다 LiveAnalytics. Timestream용 Apache Flink 샘플 데이터 커넥터를 생성했습니다. 또한 데이터가 Amazon Kinesis에서 Managed Service for Apache Flink로, 마지막으로 Amazon Timestream으로 흐를 수 있도록 Amazon Kinesis로 데이터를 보내기 위한 샘플 애플리케이션을 만들었습니다. 이러한 모든 아티팩트는 에서 사용할 수 있습니다 GitHub. 이 동영상 자습서에서는 설정에 대해 설명합니다.

> **Note**
>
> Java 11은 Managed Service for Apache Flink 애플리케이션을 사용하는 데 권장되는 버전입니다. Java 버전이 여러 개인 경우 Java 11을 JAVA_HOME 환경 변수로 내보내야 합니다.

주제

- 샘플 애플리케이션
- 비디오 자습서

샘플 애플리케이션

시작하려면 아래 절차를 따르세요.

1. 에 설명된 지침에 `kdaflink` 따라 이름이 인 데이터베이스를 Timestream에서 생성합니다. 데이터베이스 생성
2. 에 설명된 지침에 `kinesisdata1` 따라 이름이 인 테이블을 Timestream에서 생성합니다. 테이블 생성
3. 스트림 생성에 설명된 지침에 `TimestreamTestStream` 따라 이름으로 Amazon Kinesis 데이터 스트림을 생성합니다.
4. 의 지침에 따라 Timestream용 Apache Flink 데이터 커넥터의 GitHub 리포지토리를 복제합니다. GitHub
5. 샘플 애플리케이션을 컴파일, 실행 및 사용하려면 Apache Flink 샘플 데이터 커넥터의 지침을 따릅니다. README
6. 애플리케이션 코드 컴파일 지침에 따라 Managed Service for Apache Flink 애플리케이션을 컴파일합니다.
7. Apache Flink 스트리밍 코드 업로드 지침에 따라 Managed Service for Apache Flink 애플리케이션 바이너리 업로드

   a. 애플리케이션 생성을 클릭한 후 애플리케이션의 IAM 역할 링크를 클릭합니다.

   b. AmazonKinesisReadOnlyAccess 및 에 대한 IAM 정책을 연결합니다AmazonTimestreamFullAccess.

   > **Note**
   >
   > 위 IAM 정책은 특정 리소스로 제한되지 않으며 프로덕션 사용에 적합하지 않습니다. 프로덕션 시스템의 경우 특정 리소스에 대한 액세스를 제한하는 정책을 사용하는 것이 좋습니다.

8. 의 지침에 따라 Kinesis에 데이터를 쓰는 샘플 애플리케이션의 GitHub 리포지토리를 복제합니다. GitHub
9. 의 지침에 따라 Kinesis에 데이터를 쓰기 위한 샘플 애플리케이션을 README 실행합니다.
10. 지침에 따라 Timestream에서 하나 이상의 쿼리를 실행하여 Kinesis에서 Managed Service for Apache Flink로 데이터가 전송되고 있는지 확인합니다. 테이블 생성

비디오 자습서

이 동영상에서는 Managed Service for Apache Flink와 함께 Timestream을 사용하는 방법을 설명합니다.

Amazon Kinesis

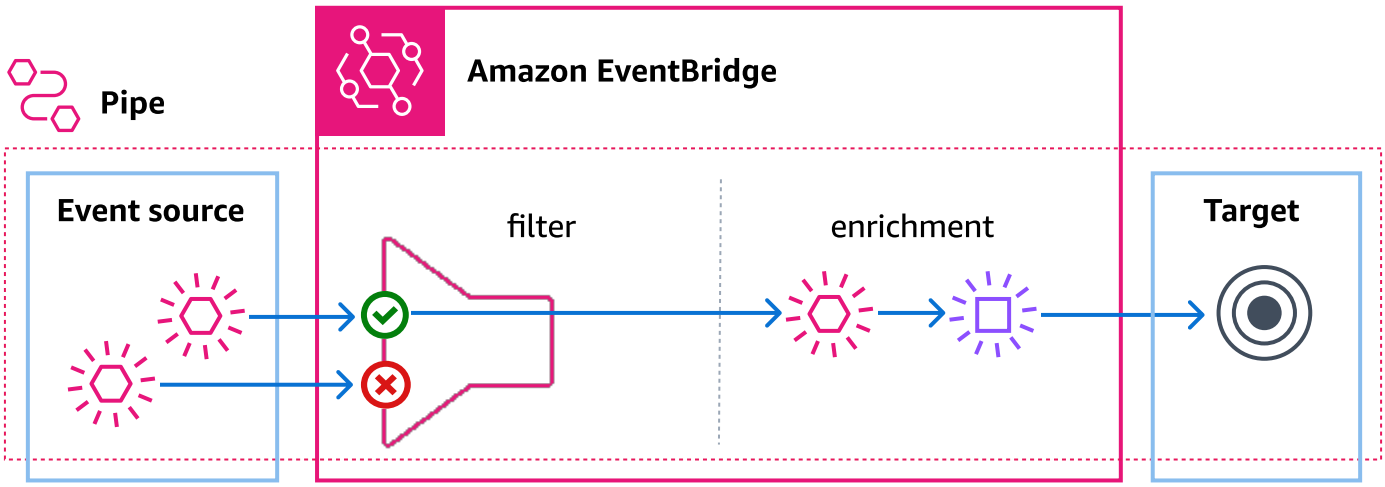
사용 Amazon Managed Service for Apache Flink

Managed Service for Apache Flink용 샘플 데이터 커넥터를 LiveAnalytics 사용하여 Kinesis Data Streams에서 Timestream으로 Timestream 데이터를 전송할 수 있습니다. 자세한 내용은 Amazon Managed Service for Apache Flink Apache Flink를 참조하세요.

EventBridge 파이프를 사용하여 Kinesis 데이터 전송 Timestream

EventBridge 파이프를 사용하여 Kinesis 스트림에서 LiveAnalytics 테이블용 Amazon Timestream으로 데이터를 전송할 수 있습니다.

파이프는 고급 변환 및 보강을 지원하여 지원되는 소스와 대상 간의 통합을 위한 point-to-point 것입니다. 파이프는 이벤트 기반 아키텍처를 개발할 때 전문 지식 및 통합 코드의 필요성을 줄입니다. 파이프를 설정하려면 소스를 선택하고, 선택적 필터링을 추가하고, 선택적 보강을 정의하고, 이벤트 데이터의 대상을 선택합니다.



이 통합을 통해 의 Timestream시계열 데이터 분석 기능의 성능을 활용하는 동시에 데이터 수집 파이프라인을 간소화할 수 있습니다.

에서 EventBridge 파이프를 사용하면 다음과 같은 이점이 Timestream 있습니다.

- 실시간 데이터 수집: Kinesis에서 용 Timestream으로 직접 데이터를 스트리밍하여 실시간 분석 및 모니터링을 LiveAnalytics가능하게 합니다.
- 원활한 통합: EventBridge 파이프를 활용하여 복잡한 사용자 지정 통합 없이 데이터 흐름을 관리합니다.
- 향상된 필터링 및 변환: 특정 데이터 처리 요구 사항을 충족하기 Timestream 위해 Kinesis 레코드를에 저장하기 전에 필터링하거나 변환합니다.
- 확장성: 처리량이 많은 데이터 스트림을 처리하고 병렬 처리 및 배치 기능을 내장하여 효율적인 데이터 처리를 보장합니다.

구성

Kinesis에서 로 데이터를 스트리밍하도록 EventBridge 파이프를 설정하려면 다음 단계를 Timestream 따르세요.

1. Kinesis 스트림 생성

   데이터를 수집할 활성 Kinesis 데이터 스트림이 있는지 확인합니다.

2. Timestream 데이터베이스 및 테이블 생성

   데이터가 저장될 Timestream 데이터베이스와 테이블을 설정합니다.

3. EventBridge 파이프 구성:
   - 소스: Kinesis 스트림을 소스로 선택합니다.
   - 대상: Timestream 를 대상으로 선택합니다.
   - 배치 설정: 배치 창과 배치 크기를 정의하여 데이터 처리를 최적화하고 지연 시간을 줄입니다.

> **Important**
>
> 파이프를 설정할 때는 몇 개의 레코드를 수집하여 모든 구성의 정확성을 테스트하는 것이 좋습니다. 파이프를 성공적으로 생성한다고 해서 파이프라인이 올바르고 데이터가 오류 없이 흐른다는 보장은 없습니다. 매핑을 적용한 후 잘못된 테이블, 잘못된 동적 경로 파라미터 또는 잘못된 Timestream 레코드와 같은 런타임 오류가 있을 수 있으며, 이는 실제 데이터가 파이프를 통해 흐를 때 발견됩니다.

다음 구성은 데이터가 수집되는 속도를 결정합니다.

- BatchSize: 에 대해 Timestream으로 전송할 배치의 최대 크기입니다 LiveAnalytics. 범위: 0~100. 최대 처리량을 얻으려면 이 값을 100으로 유지하는 것이 좋습니다.
- MaximumBatchingWindowInSeconds: 배치가 LiveAnalytics 대상에 대해 Timestream으로 전송되기 batchSize 전에 를 채우기 위해 대기하는 최대 시간입니다. 수신 이벤트 속도에 따라 이 구성은 수집 지연을 결정하므로 데이터를 Timestream 거의 실시간으로 계속 전송하려면 이 값을 < 10초로 유지하는 것이 좋습니다.
- ParallelizationFactor: 각 샤드에서 동시에 처리할 배치 수입니다. 최대 처리량과 거의 실시간 수집을 얻으려면 최대 값 10을 사용하는 것이 좋습니다.

  스트림을 여러 대상으로 읽는 경우 향상된 팬아웃을 사용하여 파이프에 전용 소비자를 제공하여 처리량을 높입니다. 자세한 내용은 Kinesis Data Streams 사용 설명서의 를 사용하여 향상된 팬아웃 소비자 개발을 Kinesis Data Streams API 참조하세요.

> **Note**
>
> 달성할 수 있는 최대 처리량은 계정당 동시 파이프 실행으로 제한됩니다.

다음 구성은 데이터 손실을 방지합니다.

- DeadLetterConfig: 사용자 오류 DeadLetterConfig 로 LiveAnalytics 인해 이벤트를 Timestream에 수집할 수 없는 경우 데이터 손실을 방지하려면 항상 를 구성하는 것이 좋습니다.

다음과 같은 구성 설정을 사용하여 파이프의 성능을 최적화하면 레코드가 속도 저하 또는 차단되는 것을 방지할 수 있습니다.

- MaximumRecordAgeInSeconds: 이보다 오래된 레코드는 처리되지 않으며 로 직접 이동됩니다DLQ. 이 값을 대상 Timestream 테이블의 구성된 메모리 스토어 보존 기간 이하로 설정하는 것이 좋습니다.
- MaximumRetryAttempts: 레코드가 로 전송되기 전에 레코드에 대한 재시도 횟수입니다 DeadLetterQueue. 이를 10으로 구성하는 것이 좋습니다. 이렇게 하면 일시적인 문제를 해결하는 데 도움이 될 수 있으며, 지속적인 문제의 경우 레코드가 로 이동 DeadLetterQueue 되고 나머지 스트림의 차단이 해제됩니다.
- OnPartialBatchItemFailure: 부분 배치 처리를 지원하는 소스의 경우 에 삭제/전송하기 전에 실패한 레코드를 추가로 재시도하려면 이를 활성화하고 AUTOMATIC_BISECT로 구성하는 것이 좋습니다 DLQ.

구성 예

다음은 Kinesis 스트림에서 Timestream 테이블로 데이터를 스트리밍하도록 EventBridge 파이프를 구성하는 방법의 예입니다.

Example IAM 에 대한 정책 업데이트 Timestream

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream:WriteRecords"
            ],
            "Resource": [
                "arn:aws:timestream:us-east-1:123456789012:database/my-database/table/
my-table"
            ]
        },
        {
            "Effect": "Allow",
            "Action": [
                "timestream:DescribeEndpoints"
            ],
            "Resource": "*"
        }
    ]
}
```

Example Kinesis 스트림 구성

```
{
  "Source": "arn:aws:kinesis:us-east-1:123456789012:stream/my-kinesis-stream",
  "SourceParameters": {
    "KinesisStreamParameters": {
        "BatchSize": 100,
        "DeadLetterConfig": {
            "Arn": "arn:aws:sqs:us-east-1:123456789012:my-sqs-queue"
        },
       "MaximumBatchingWindowInSeconds": 5,
        "MaximumRecordAgeInSeconds": 1800,
        "MaximumRetryAttempts": 10,
```

```
        "StartingPosition": "LATEST",
       "OnPartialBatchItemFailure": "AUTOMATIC_BISECT"
    }
  }
}
```

Example Timestream 대상 구성

```
{
    "Target": "arn:aws:timestream:us-east-1:123456789012:database/my-database/table/my-
table",
    "TargetParameters": {
        "TimestreamParameters": {
            "DimensionMappings": [
                {
                    "DimensionName": "sensor_id",
                    "DimensionValue": "$.data.device_id",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "DimensionName": "sensor_type",
                    "DimensionValue": "$.data.sensor_type",
                    "DimensionValueType": "VARCHAR"
                },
                {
                    "DimensionName": "sensor_location",
                    "DimensionValue": "$.data.sensor_loc",
                    "DimensionValueType": "VARCHAR"
                }
            ],
            "MultiMeasureMappings": [
                {
                    "MultiMeasureName": "readings",
                    "MultiMeasureAttributeMappings": [
                        {
                            "MultiMeasureAttributeName": "temperature",
                            "MeasureValue": "$.data.temperature",
                            "MeasureValueType": "DOUBLE"
                        },
                        {
                            "MultiMeasureAttributeName": "humidity",
                            "MeasureValue": "$.data.humidity",
                            "MeasureValueType": "DOUBLE"
```

```
                        },
                        {
                            "MultiMeasureAttributeName": "pressure",
                            "MeasureValue": "$.data.pressure",
                            "MeasureValueType": "DOUBLE"
                        }
                    ]
                }
            ],
            "SingleMeasureMappings": [],
            "TimeFieldType": "TIMESTAMP_FORMAT",
            "TimestampFormat": "yyyy-MM-dd HH:mm:ss.SSS",
            "TimeValue": "$.data.time",
            "VersionValue": "$.approximateArrivalTimestamp"
        }
    }
}
```

이벤트 변환

EventBridge 파이프를 사용하면 에 도달하기 전에 데이터를 변환할 수 있습니다 Timestream. 변환 규칙을 정의하여 필드 이름 변경과 같은 수신 Kinesis 레코드를 수정할 수 있습니다.

Kinesis 스트림에 온도 및 습도 데이터가 포함되어 있다고 가정해 보겠습니다. EventBridge 변환을 사용하여 에 삽입하기 전에 이러한 필드의 이름을 바꿀 수 있습니다 Timestream.

모범 사례

배치 및 버퍼링

- 쓰기 지연 시간과 처리 효율성 간의 균형을 맞추도록 배치 창과 크기를 구성합니다.
- 배치 창을 사용하여 처리 전에 충분한 데이터를 축적하여 자주 발생하는 소규모 배치의 오버헤드를 줄입니다.

병렬 처리

특히 처리량이 많은 스트림의 경우 ParallelizationFactor 설정을 활용하여 동시성을 높입니다. 이렇게 하면 각 샤드의 여러 배치를 동시에 처리할 수 있습니다.

데이터 변환

EventBridge Pipes의 변환 기능을 활용하여 에 저장하기 전에 레코드를 필터링하고 개선합니다 Timestream. 이렇게 하면 데이터를 분석 요구 사항에 맞추는 데 도움이 될 수 있습니다.

보안

- EventBridge 파이프에 사용되는 IAM 역할에 에서 읽고 에 Kinesis 쓸 수 있는 필요한 권한이 있는지 확인합니다 Timestream.
- 암호화 및 액세스 제어 조치를 사용하여 전송 중 및 저장 중인 데이터를 보호합니다.

디버깅 실패

- 파이프 자동 비활성화

  대상이 없거나 권한 문제가 있는 경우 약 2시간 내에 파이프가 자동으로 비활성화됩니다.
- 제한

  파이프에는 스로틀이 줄어들 때까지 자동으로 백오프하고 재시도할 수 있는 기능이 있습니다.
- 로그 활성화

  ERROR 레벨에서 로그를 활성화하고 실행 데이터를 포함하여 실패에 대한 더 많은 인사이트를 얻는 것이 좋습니다. 장애가 발생하면 이러한 로그에는 request/response sent/received의 이 포함됩니다 Timestream. 이렇게 하면 연결된 오류를 이해하고 필요한 경우 레코드를 수정한 후 다시 처리할 수 있습니다.

모니터링

데이터 흐름 문제를 감지하려면 다음에 대한 경보를 설정하는 것이 좋습니다.

- 소스에서 레코드의 최대 기간
  - `GetRecords.IteratorAgeMilliseconds`
- 파이프의 실패 지표
  - `ExecutionFailed`
  - `TargetStageFailed`
- Timestream 쓰기 API 오류
  - `UserErrors`

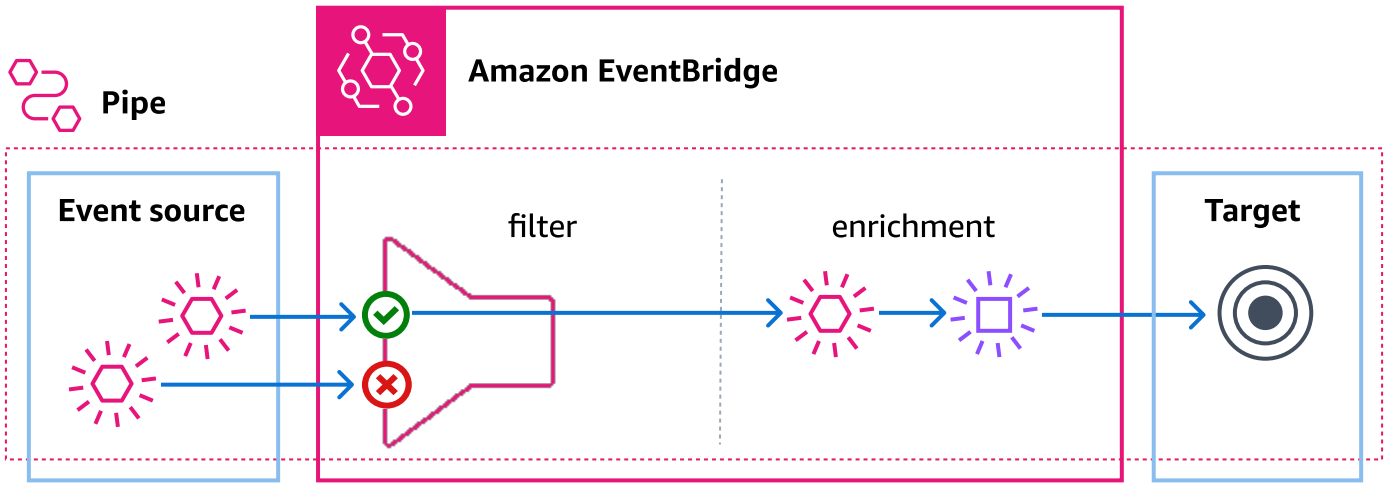
추가 모니터링 지표는 EventBridge 사용 설명서의 모니터링을 EventBridge 참조하세요.

Amazon MQ

EventBridge 파이프를 사용하여 Amazon MQ 데이터 전송 Timestream

파이프 EventBridge 를 사용하여 Amazon MQ 브로커에서 LiveAnalytics 테이블용 Amazon Timestream으로 데이터를 전송할 수 있습니다.

파이프는 고급 변환 및 보강을 지원하여 지원되는 소스와 대상 간의 통합을 위한 point-to-point 것입니다. 파이프는 이벤트 기반 아키텍처를 개발할 때 전문 지식 및 통합 코드의 필요성을 줄입니다. 파이프를 설정하려면 소스를 선택하고, 선택적 필터링을 추가하고, 선택적 보강을 정의하고, 이벤트 데이터의 대상을 선택합니다.



EventBridge 파이프에 대한 자세한 내용은 EventBridge 사용 설명서의 EventBridge 파이프를 참조하세요. LiveAnalytics 테이블용 Amazon Timestream에 이벤트를 전달하도록 파이프를 구성하는 방법에 대한 자세한 내용은 EventBridge 파이프 대상 세부 정보 단원을 참조하세요.

Amazon MSK

Managed Service for Apache Flink를 사용하여 의 Timestream으로 Amazon MSK 데이터 전송 LiveAnalytics

Managed Service for Apache Flink의 샘플 데이터 커넥터와 유사한 데이터 커넥터를 구축 Timestream 하여 에서 Amazon MSK 로 Timestream 데이터를 전송할 수 있습니다. 자세한 내용은 Amazon Managed Service for Apache Flink 섹션을 참조하세요.

Kafka Connect를 사용하여 에 대한 Amazon MSK 데이터를 Timestream으로 전송 LiveAnalytics

Kafka Connect를 사용하여 시계열 데이터를 의 Timestream으로 Amazon MSK 직접 수집할 수 있습니다 LiveAnalytics.

용 Kafka Sink Connector 샘플을 생성했습니다 Timestream. 또한 데이터를 Kafka 주제에 게시하기 위한 샘플 Apache jMeter 테스트 계획을 생성하여 데이터가 주제에서 Timestream Kafka Sink Connector를 통해 테이블의 LiveAnalytics Timestream으로 흐를 수 있도록 했습니다. 이러한 모든 아티팩트는 에서 사용할 수 있습니다 GitHub.

> **Note**
>
> Java 11은 Timestream Kafka Sink Connector를 사용하는 데 권장되는 버전입니다. Java 버전이 여러 개인 경우 Java 11을 JAVA_HOME 환경 변수로 내보내야 합니다.

샘플 애플리케이션 생성

시작하려면 아래 절차를 따르세요.

1. 의 Timestream에서 이름이 인 데이터베이스를 LiveAnalytics생성합니다`kafkastream`.

   자세한 지침은 절차를 ??? 참조하세요.
2. 의 Timestream에서 이름이 인 테이블을 LiveAnalytics생성합니다`purchase_history`.

   자세한 지침은 절차를 ??? 참조하세요.
3. 에서 공유한 지침에 따라 , 및 를 생성합니다.

   - Amazon MSK 클러스터
   - Kafka 생산자 클라이언트 시스템으로 구성된 Amazon EC2 인스턴스
   - Kafka 주제

   자세한 지침은 kafka_ingestor 프로젝트의 사전 조건을 참조하세요.
4. Timestream Kafka Sink Connector 리포지토리를 복제합니다.

   자세한 지침은 의 리포지토리 복제 GitHub 를 참조하세요.
5. 플러그인 코드를 컴파일합니다.

자세한 지침은 커넥터 - 의 소스에서 빌드 GitHub 를 참조하세요.

6. 에 설명된 지침에 따라 S3 버킷에 다음 파일을 업로드합니다.

   - `/target` 디렉터리의 jar 파일(kafka-connector-timestream->VERSION<-jar-with-dependencies.jar)
   - 샘플 json 스키마 파일, `purchase_history.json`.

   자세한 지침은 Amazon S3 사용 설명서의 객체 업로드를 참조하세요.

7. 두 개의 VPC 엔드포인트를 생성합니다. 이러한 엔드포인트는 MSK 커넥터에서 를 사용하여 리소스에 액세스하는 데 사용됩니다 AWS PrivateLink.

   - Amazon S3 버킷에 액세스하는 방법
   - LiveAnalytics 테이블의 Timestream에 액세스하는 방법.

   자세한 지침은 VPC 엔드포인트를 참조하세요.

8. 업로드된 jar 파일로 사용자 지정 플러그인을 생성합니다.

   자세한 지침은 Amazon MSK 개발자 안내서의 플러그인을 참조하세요.

9. 에 설명된 지침에 따라 작업자 구성 파라미터 에 설명된 JSON 내용을 사용하여 사용자 지정 작업자 구성을 생성합니다. https://github.com/awslabs/amazon-timestream-tools/tree/mainline/integrations/kafka_connector#worker-configuration-parameters

   자세한 지침은 Amazon MSK 개발자 안내서의 사용자 지정 작업자 구성 생성을 참조하세요.

10. 서비스 실행 IAM 역할을 생성합니다.

    자세한 지침은 IAM 서비스 역할을 참조하세요.

11. 이전 단계에서 생성된 Amazon MSK 사용자 지정 플러그인, 사용자 지정 작업자 구성 및 서비스 실행 IAM 역할과 샘플 커넥터 구성 을 사용하여 커넥터를 생성합니다.

    자세한 지침은 Amazon MSK 개발자 안내서의 커넥터 생성을 참조하세요.

    아래 구성 파라미터의 값을 해당 값으로 업데이트해야 합니다. 자세한 내용은 커넥터 구성 파라미터를 참조하세요.

    - `aws.region`
    - `timestream.schema.s3.bucket.name`

- `timestream.ingestion.endpoint`

  커넥터 생성을 완료하는 데 5~10분이 걸립니다. 파이프라인의 상태가 로 변경되면 파이프라인이 준비됩니다`Running`.

12. 생성된 Kafka 주제에 데이터를 작성하기 위한 메시지의 연속 스트림을 게시합니다.

    자세한 지침은 사용 방법을 참조하세요.

13. 하나 이상의 쿼리를 실행하여 데이터가 에서 MSK Connect Amazon MSK 로 LiveAnalytics 테이블의 Timestream으로 전송되고 있는지 확인합니다.

    자세한 지침은 절차를 ??? 참조하세요.

추가 리소스

Kafka Connect를 LiveAnalytics 사용하기 위해 Kafka 클러스터에서 Timestream으로 실시간 서버리스 데이터를 수집하는 블로그에서는 LiveAnalytics Kafka Sink Connector용 Timestream을 사용하여 파이프라인을 end-to-end 설정하는 방법을 설명합니다. 이 파이프라인은 Apache jMeter 테스트 계획을 사용하여 Kafka 주제에 수천 개의 샘플 메시지를 게시하여 LiveAnalytics 테이블용 Timestream에서 수집된 레코드를 확인하는 Kafka 생산자 클라이언트 머신부터 시작합니다.

Amazon QuickSight

Amazon QuickSight 을 사용하여 Amazon Timestream 데이터가 포함된 데이터 대시보드를 분석하고 게시할 수 있습니다. 이 섹션에서는 새 QuickSight 데이터 소스 연결을 생성하고, 권한을 수정하고, 새 데이터 세트를 생성하고, 분석을 수행하는 방법을 설명합니다. 이 동영상 자습서에서는 Timestream 및 Amazon 를 사용하는 방법을 설명합니다 QuickSight.

> **Note**
>
> Amazon의 모든 데이터 세트 QuickSight 는 읽기 전용입니다. Amazon을 사용하여 데이터 소스, 데이터 세트 또는 필드를 QuickSight 제거하면 Timestream에서 실제 데이터를 변경할 수 없습니다.

주제

- 에서 Amazon Timestream에 액세스 QuickSight
- Timestream에 대한 새 QuickSight 데이터 소스 연결 생성

- Timestream의 QuickSight 데이터 소스 연결에 대한 권한 편집
- Timestream에 대한 새 QuickSight 데이터 세트 생성
- Timestream에 대한 새 분석 생성
- 비디오 자습서

에서 Amazon Timestream에 액세스 QuickSight

계속하기 전에 Amazon Timestream에 연결할 수 있는 권한을 Amazon에 QuickSight 부여해야 합니다. 연결이 활성화되지 않은 경우 연결을 시도할 때 오류가 발생합니다. QuickSight 관리자는 AWS 리소스에 대한 연결을 승인할 수 있습니다. 에서 Timestream QuickSight 으로의 연결을 승인하려면 다른 AWS 서비스 사용: 액세스 범위 축소 의 절차를 따르고 5단계에서 Amazon Timestream을 선택합니다.

Timestream에 대한 새 QuickSight 데이터 소스 연결 생성

> **Note**
>
> Amazon QuickSight 과 Amazon Timestream 간의 연결은 SSL (TLS 1.2)를 사용하여 전송 중에 암호화됩니다. 암호화되지 않은 연결은 생성할 수 없습니다.

1. 에 설명된 대로 Amazon QuickSight이 Amazon Timestream에 액세스할 수 있는 적절한 권한을 구성했는지 확인합니다 에서 Amazon Timestream에 액세스 QuickSight.
2. 먼저 새 데이터 세트를 생성합니다. 탐색 창에서 데이터 세트를 선택한 다음 새 데이터 세트 를 선택합니다.
3. Timestream 데이터 소스 카드를 선택합니다.
4. 데이터 소스 이름 에 Timestream 데이터 소스 연결의 이름을 입력합니다. 예: `US Timestream Data`.

   > **Note**
   >
   > Timestream에 연결하여 많은 데이터 세트를 생성할 수 있으므로 이름은 단순하게 유지하는 것이 좋습니다.

5. 연결 검증을 선택하여 Timestream에 성공적으로 연결할 수 있는지 확인합니다.

> **Note**
>
> 연결 검증은 연결할 수 있는지만 검증합니다. 그러나 특정 테이블 또는 쿼리를 검증하지는 않습니다.

6. 계속하려면 데이터 소스 생성을 선택합니다.
7. 데이터베이스 에서 선택...을 선택하여 사용 가능한 옵션 목록을 봅니다. 사용할 항목을 선택합니다.
8. 계속하려면 선택을 선택합니다.
9. 다음 중 하나를 선택합니다.
   - 데이터를 QuickSight의 인 메모리 엔진(라고 함SPICE)으로 가져오려면 로 가져오기SPICE를 선택하여 더 빠른 분석을 수행합니다.
   - 데이터 세트를 새로 고치거나 분석 또는 대시보드를 사용할 때마다 가 데이터에 대한 쿼리를 실행 QuickSight 하도록 허용하려면 데이터 직접 쿼리를 선택합니다.
10. 편집/미리보기를 선택한 다음 저장을 선택하여 데이터 세트를 저장하고 닫습니다.

Timestream의 QuickSight 데이터 소스 연결에 대한 권한 편집

다음 절차에서는 다른 사용자가 동일한 Timestream 데이터 소스에 액세스할 수 있도록 다른 QuickSight 사용자의 권한을 확인, 추가 및 취소하는 방법을 설명합니다. 사용자를 QuickSight 추가하려면 먼저 의 활성 사용자여야 합니다.

> **Note**
>
> 에서 QuickSight데이터 소스에는 사용자와 소유자의 두 가지 권한 수준이 있습니다.
>
> - 읽기 액세스를 허용할 사용자를 선택합니다.
> - 소유자를 선택하여 해당 사용자가 이 QuickSight 데이터 소스를 편집, 공유 또는 삭제할 수 있도록 허용합니다.

1. 에 설명된 대로 Amazon QuickSight이 Amazon Timestream에 액세스할 수 있는 적절한 권한을 구성했는지 확인합니다 에서 Amazon Timestream에 액세스 QuickSight.
2. 왼쪽에서 데이터 세트를 선택한 다음 아래로 스크롤하여 Timestream 연결을 위한 데이터 소스 카드를 찾습니다. 예: `US Timestream Data`.

3. Timestream 데이터 소스 카드를 선택합니다.
4. Share data source를 선택합니다. 현재 권한 목록이 표시됩니다.
5. (선택 사항) 권한을 편집하려면 user 또는 를 선택할 수 있습니다owner.
6. (선택 사항) 권한을 취소하려면 를 선택합니다Revoke access. 취소한 사용자는 이 데이터 소스에서 새 데이터 세트를 생성할 수 없습니다. 그러나 기존 데이터 세트는 여전히 이 데이터 소스에 액세스할 수 있습니다.
7. 권한을 추가하려면 를 선택한 Invite users다음 다음 다음 단계에 따라 사용자를 추가합니다.
   a. 동일한 데이터 소스를 사용할 수 있도록 사용자를 추가합니다.
   b. 각각에 대해 적용할 Permission 를 선택합니다.
8. 마쳤으면 Close를 선택합니다.

Timestream에 대한 새 QuickSight 데이터 세트 생성

1. 에 설명된 대로 Amazon QuickSight이 Amazon Timestream에 액세스할 수 있는 적절한 권한을 구성했는지 확인합니다 에서 Amazon Timestream에 액세스 QuickSight.
2. 왼쪽에서 데이터 세트를 선택한 다음 아래로 스크롤하여 Timestream 연결을 위한 데이터 소스 카드를 찾습니다. 데이터 소스가 많은 경우 페이지 상단의 검색 창을 사용하여 이름에 부분 일치하는 데이터 소스를 찾을 수 있습니다.
3. Timestream 데이터 소스 카드를 선택합니다. 그런 다음 데이터 세트 생성을 선택합니다.
4. 데이터베이스의 경우 선택을 선택하여 사용 가능한 옵션 목록을 확인합니다. 사용할 데이터베이스를 선택합니다.
5. 테이블의 경우 사용할 테이블을 선택합니다.
6. 편집/미리 보기를 선택합니다.
7. (선택 사항) 데이터를 더 추가하려면 오른쪽 상단의 데이터 추가를 선택합니다.
   a. 데이터 소스 전환을 선택하고 다른 데이터 소스를 선택합니다.
   b. UI 프롬프트에 따라 데이터 추가를 완료합니다.
   c. 동일한 데이터 세트에 새 데이터를 추가한 후 이 조인 구성(빨간색 점 2개)을 선택합니다. 각 추가 테이블에 대해 조인을 설정합니다.
   d. 계산된 필드를 추가하고자 하는 경우 계산된 필드 추가를 선택합니다.
   e. Sagemaker를 사용하려면 를 사용하여 Augment를 SageMaker 선택합니다. 이 옵션은 QuickSight 엔터프라이즈 에디션에서만 사용할 수 있습니다.

f. 생략하려는 필드를 선택 취소합니다.

g. 변경하려는 데이터 유형을 업데이트합니다.

8. 마치면 저장을 선택하여 데이터 세트를 저장하고 닫습니다.

Timestream에 대한 새 분석 생성

1. 에 설명된 대로 Amazon QuickSight이 Amazon Timestream에 액세스할 수 있는 적절한 권한을 구성했는지 확인합니다 에서 Amazon Timestream에 액세스 QuickSight.
2. 왼쪽에서 분석을 선택합니다.
3. 다음 중 하나를 선택합니다.

   - 새 분석을 만들려면 오른쪽에서 새 분석을 선택합니다.
   - 기존 분석에 Timestream 데이터 세트를 추가하려면 편집하려는 분석을 엽니다. 왼쪽 상단에 있는 연필 아이콘을 선택한 다음 데이터 세트 추가 를 선택합니다.

4. 왼쪽의 필드를 선택하여 첫 번째 데이터 시각화를 시작합니다.
5. 자세한 내용은 분석 작업 - Amazon을 참조하세요. QuickSight

비디오 자습서

이 동영상에서는 Amazon이 Timestream에서 QuickSight 작동하는 방법을 설명합니다.

Amazon SageMaker

Amazon SageMaker Notebooks를 사용하여 기계 학습 모델을 Amazon Timestream과 통합할 수 있습니다. 시작하는 데 도움이 되도록 Timestream에서 데이터를 처리하는 샘플 SageMaker 노트북을 만들었습니다. 데이터는 데이터를 지속적으로 전송하는 다중 스레드 Python 애플리케이션에서 Timestream에 삽입됩니다. 샘플 SageMaker 노트북 및 샘플 Python 애플리케이션의 소스 코드는 에서 사용할 수 있습니다 GitHub.

1. 데이터베이스 생성 및 에 설명된 지침에 따라 데이터베이스 및 테이블 생성 테이블 생성
2. 의 지침에 따라 다중 스레드 Python 샘플 애플리케이션의 GitHub 리포지토리 복제 GitHub
3. 의 지침에 따라 샘플 Timestream SageMaker Notebook의 GitHub 리포지토리를 복제합니다 GitHub.
4. 애플리케이션을 실행하여 의 지침에 따라 Timestream으로 데이터를 지속적으로 수집합니다. README

5. 지침에 따라 여기에 설명된 SageMaker 대로 Amazon용 Amazon S3 버킷을 생성합니다.
6. 최신 boto3가 설치된 Amazon SageMaker 인스턴스를 생성합니다. 여기에 설명된 지침 외에도 아래 단계를 따릅니다.

   a. 노트북 인스턴스 생성 페이지에서 추가 구성을 클릭합니다.
   b. 수명 주기 구성 - 선택 사항을 클릭하고 새 수명 주기 구성 생성을 선택합니다.
   c. 수명 주기 구성 생성 마법사 상자에서 다음을 수행합니다.

      i. 구성에 원하는 이름을 입력합니다. 예: `on-start`
      ii. 노트북 스크립트 시작에서 Github의 스크립트 콘텐츠를 복사-붙여넣기합니다.
      iii. 붙여넣은 스크립트`PACKAGE=boto3`에서 를 `PACKAGE=scipy`로 바꿉니다.
7. 구성 생성을 클릭합니다.
8. AWS 관리 콘솔에서 IAM 서비스로 이동하여 노트북 인스턴스에 대해 새로 생성된 SageMaker 실행 역할을 찾습니다.
9. 에 대한 IAM 정책을 `AmazonTimestreamFullAccess` 실행 역할에 연결합니다.

> **Note**
>
> 이 `AmazonTimestreamFullAccess` IAM 정책은 특정 리소스로 제한되지 않으며 프로덕션 사용에 적합하지 않습니다. 프로덕션 시스템의 경우 특정 리소스에 대한 액세스를 제한하는 정책을 사용하는 것이 좋습니다.

10. 노트북 인스턴스의 상태가 이면 Open Jupyter를 InService선택하여 인스턴스에 대한 SageMaker 노트북을 시작합니다.
11. 업로드 버튼을 선택하여 노트북`Timestream_SageMaker_Demo.ipynb`에 `timestreamquery.py` 및 파일을 업로드합니다.
12. `Timestream_SageMaker_Demo.ipynb`

> **Note**
>
> 커널을 찾을 수 없는 팝업이 표시되면 conda_python3을 선택하고 커널 설정을 클릭합니다.

13. 훈련 모델의 데이터베이스 이름`DB_NAME`, 테이블 이름, S3 버킷 이름 및 리전과 `ENDPOINT` 일치하도록 `bucket`, `TABLE_NAME`, , 를 수정합니다.
14. 재생 아이콘을 선택하여 개별 셀을 실행합니다.

15. 셀 에 도달하면 출력이 최소 2개의 호스트 이름을 반환하는지 `Leverage Timestream to find hosts with average CPU utilization across the fleet`확인합니다.

> **Note**
>
> 출력에 호스트 이름이 2개 미만인 경우 더 많은 수의 스레드와 호스트 규모로 데이터를 Timestream으로 수집하는 샘플 Python 애플리케이션을 다시 실행해야 할 수 있습니다.

16. 셀 에 도달하면 훈련 작업에 대한 리소스 요구 사항에 `train_instance_type` 따라 를 `Train a Random Cut Forest (RCF) model using the CPU utilization history`변경합니다.
17. 셀 에 도달하면 추론 작업에 대한 리소스 요구 사항에 `instance_type` 따라 를 `Deploy the model for inference`변경합니다.

> **Note**
>
> 모델을 훈련하는 데 몇 분 정도 걸릴 수 있습니다. 훈련이 완료되면 셀 출력에 완료됨 - 훈련 작업 완료 메시지가 표시됩니다.

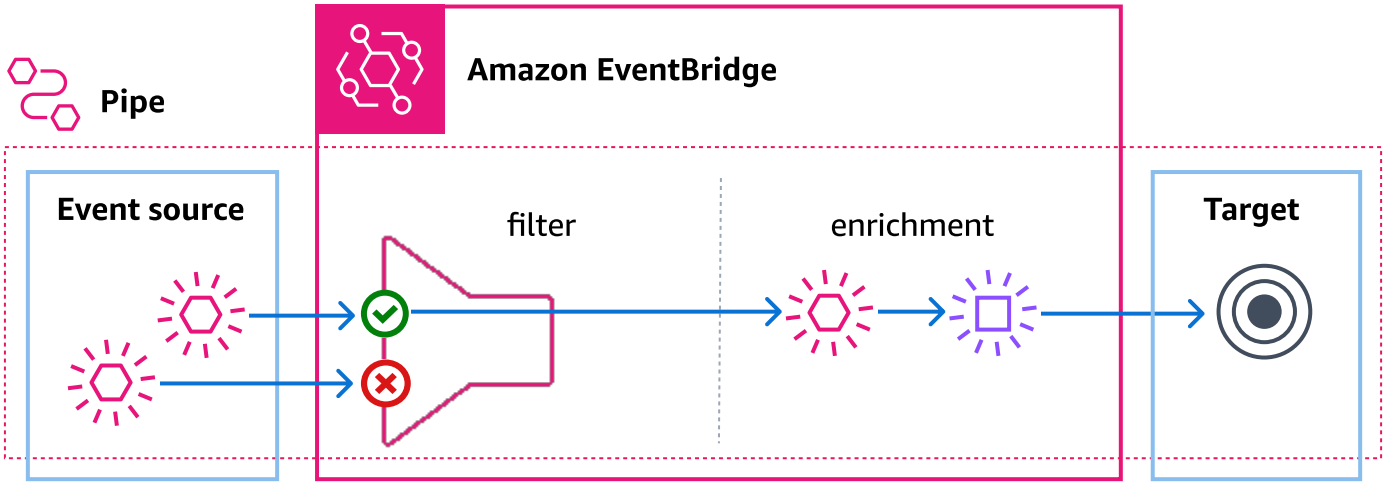
18. 셀을 실행`Stop and delete the endpoint`하여 리소스를 정리합니다. SageMaker 콘솔에서 인스턴스를 중지하고 삭제할 수도 있습니다.

Amazon SQS

EventBridge 파이프를 사용하여 Amazon SQS 데이터 전송 Timestream

EventBridge 파이프를 사용하여 Amazon SQS 대기열에서 LiveAnalytics 테이블용 Amazon Timestream으로 데이터를 전송할 수 있습니다.

파이프는 고급 변환 및 보강을 지원하여 지원되는 소스와 대상 간의 통합을 위한 point-to-point 것입니다. 파이프는 이벤트 기반 아키텍처를 개발할 때 전문 지식 및 통합 코드의 필요성을 줄입니다. 파이프를 설정하려면 소스를 선택하고, 선택적 필터링을 추가하고, 선택적 보강을 정의하고, 이벤트 데이터의 대상을 선택합니다.



EventBridge 파이프에 대한 자세한 내용은 EventBridge 사용 설명서의 EventBridge 파이프를 참조하세요. LiveAnalytics 테이블용 Amazon Timestream에 이벤트를 전달하도록 파이프를 구성하는 방법에 대한 자세한 내용은 EventBridge 파이프 대상 세부 정보 섹션을 참조하세요.

DBeaver 를 사용하여 Amazon Timestream 작업

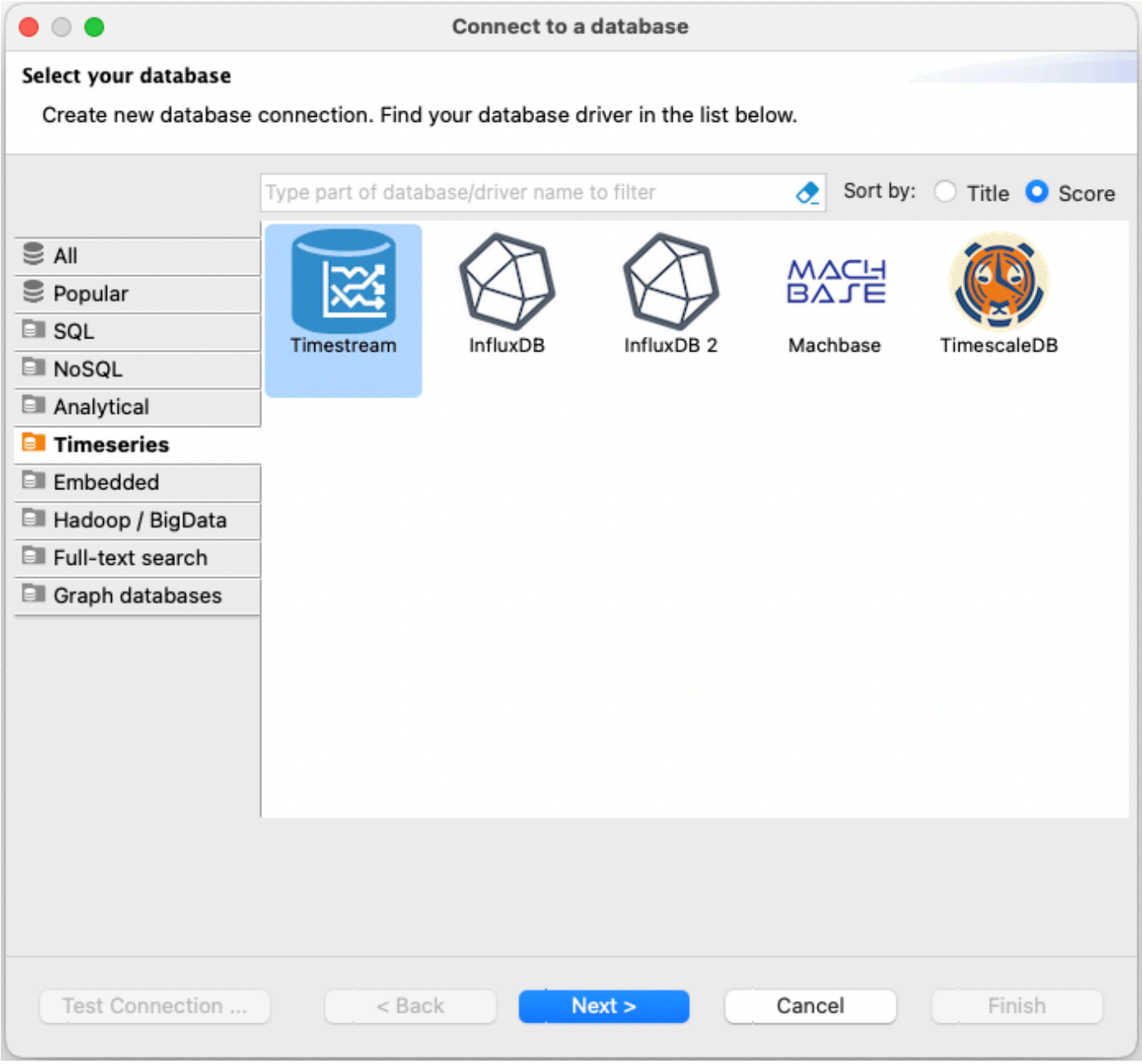
DBeaver 는 JDBC 드라이버가 있는 모든 데이터베이스를 관리하는 데 사용할 수 있는 무료 범용 SQL 클라이언트입니다. 강력한 데이터 보기, 편집 및 관리 기능으로 인해 개발자와 데이터베이스 관리자에게 널리 사용됩니다.

DBeaver의 클라우드 연결 옵션을 사용하여 기본적으로 Amazon TimestreamDBeaver에 연결할 수 있습니다. DBeaver 는 DBeaver 애플리케이션 내에서 직접 시계열 데이터를 사용할 수 있는 포괄적이고 직관적인 인터페이스를 제공합니다. 자격 증명을 사용하면 다른 쿼리 인터페이스에서 실행할 수 있는 모든 쿼리에 대한 전체 액세스 권한도 얻을 수 있습니다. 쿼리 결과를 더 잘 이해하고 시각화할 수 있도록 그래프를 생성할 수도 있습니다.

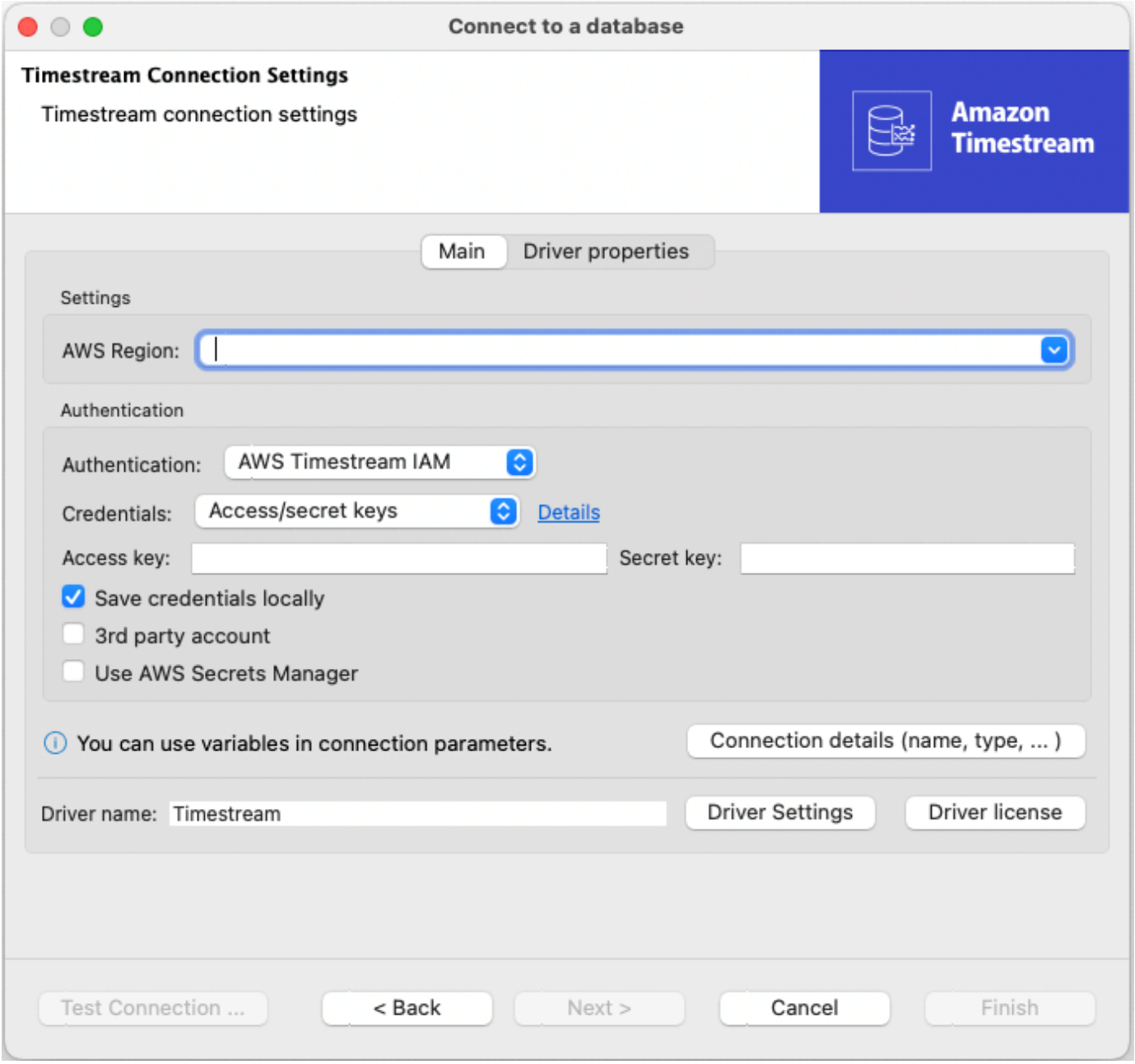
Timestream에서 작동DBeaver하도록 설정

Timestream에서 작동DBeaver하도록 설정하려면 다음 단계를 수행합니다.

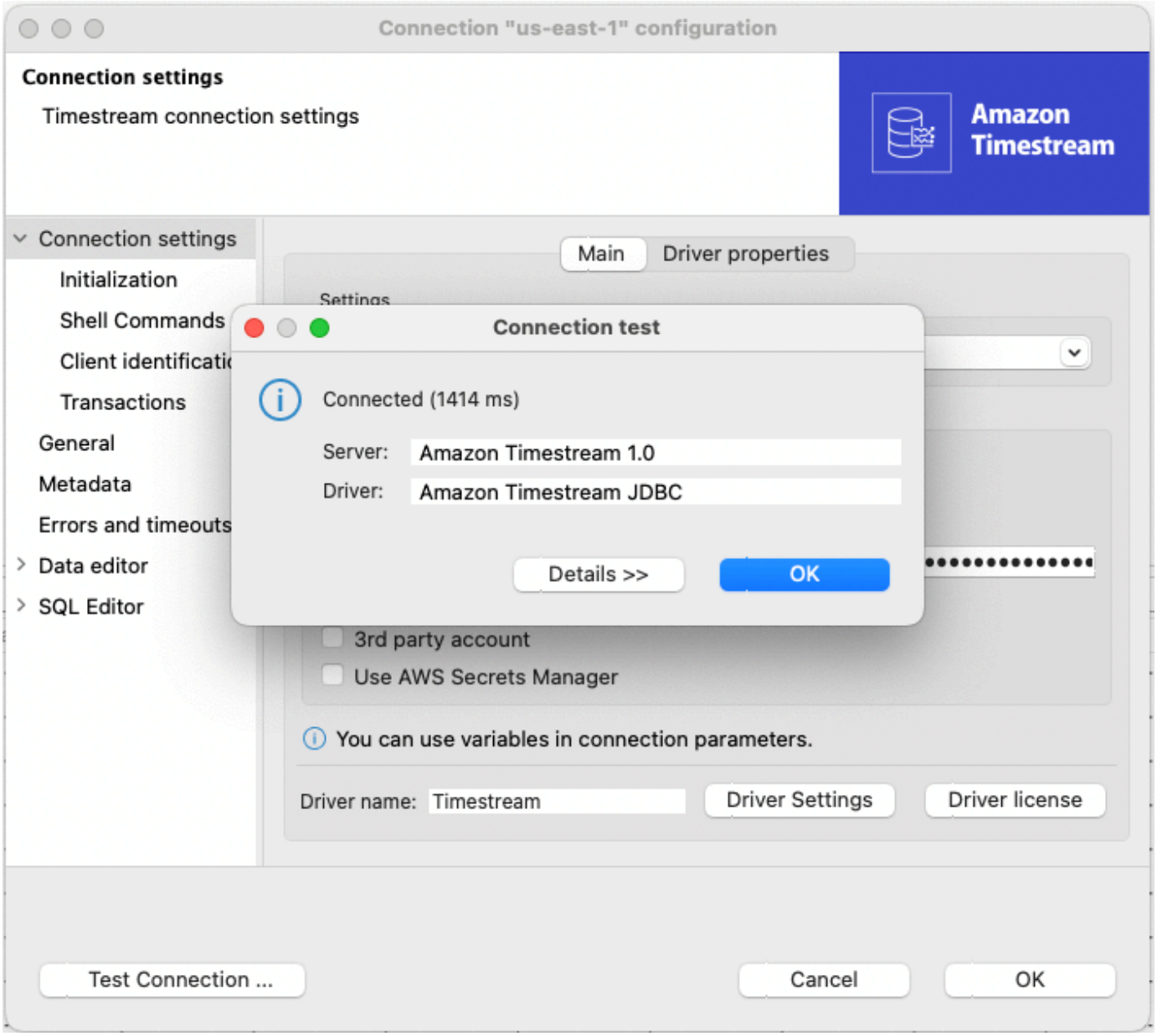
1. 로컬 시스템에 다운로드하고 설치합니다DBeaver.
2. 를 시작하고 데이터베이스 선택 영역으로 DBeaver이동하여 왼쪽 창에서 시계열을 선택한 다음 오른쪽 창에서 시계열 아이콘을 선택합니다.



3. Timestream 연결 설정 창에서 Amazon Timestream 데이터베이스에 연결하는 데 필요한 모든 정보를 입력합니다. 입력한 사용자 키에 Timestream 데이터베이스에 액세스하는 데 필요한 권한이 있는지 확인하세요. 또한 민감한 정보와 마찬가지로 입력한 정보와 키를 DBeaver 안전하고 비공개로 유지해야 합니다.



4. 연결을 테스트하여 모든 항목이 올바르게 설정되었는지 확인합니다.



5. 연결 테스트가 성공하면 이제 의 다른 데이터베이스와 마찬가지로 Amazon Timestream 데이터베이스와 상호 작용할 수 있습니다DBeaver. 예를 들어 SQL 편집기 또는 ER 다이어그램 보기로 이동하여 쿼리를 실행할 수 있습니다.

6. DBeaver 는 강력한 데이터 시각화 도구도 제공합니다. 이를 사용하려면 쿼리를 실행한 다음 그래프 아이콘을 선택하여 결과 세트를 시각화합니다. 그래프 도구는 시간이 지남에 따른 데이터 추세를 더 잘 이해하는 데 도움이 될 수 있습니다.

Amazon Timestream을 와 페어링하면 시계열 데이터를 관리하기 위한 효과적인 환경이 DBeaver 생성됩니다. 기존 워크플로에 원활하게 통합하여 생산성과 효율성을 높일 수 있습니다.

Grafana

Grafana를 사용하여 시계열 데이터를 시각화하고 알림을 생성할 수 있습니다. 데이터 시각화를 시작하는 데 도움이 되도록 Python 애플리케이션에서 Timestream으로 전송된 데이터를 시각화하는 샘플 대시보드와 설정을 설명하는 비디오 자습서를 Grafana에 만들었습니다.

주제

- 샘플 애플리케이션
- 비디오 자습서

샘플 애플리케이션

1. 자세한 내용은 에 설명된 지침에 따라 Timestream에서 데이터베이스와 테이블 데이터베이스 생성을 생성합니다.

   > **Note**
   >
   > Grafana 대시보드의 기본 데이터베이스 이름과 테이블 이름은 grafanaTable 각각 grafanaDB 및 로 설정됩니다. 이러한 이름을 사용하여 설정을 최소화합니다.

2. Python 3.7 이상 설치
3. Timestream Python 설치 및 구성 SDK
4. 의 지침에 따라 데이터를 Timestream으로 지속적으로 수집하는 멀티 스레드 Python 애플리케이션의 GitHub 리포지토리 복제 GitHub
5. 의 지침에 따라 Timestream으로 데이터를 지속적으로 수집하기 위해 애플리케이션을 실행합니다. README
6. Amazon Managed Grafana 시작하기를 완료하거나 Grafana 설치를 완료합니다.
7. Amazon Managed Grafana를 사용하는 대신 Grafana를 설치하는 경우 Grafana용 Timestream 플러그인 설치를 완료합니다.
8. 원하는 브라우저를 사용하여 Grafana 대시보드를 엽니다. Grafana를 로컬에 설치한 경우 Grafana 설명서에 설명된 지침에 따라 로그인할 수 있습니다.
9. Grafana를 시작한 후 Datasources로 이동하여 데이터 소스 추가를 클릭하고 Timestream을 검색한 다음 Timestream 데이터 소스를 선택합니다.
10. 인증 공급자 및 리전을 구성하고 저장 및 테스트를 클릭합니다.
11. 기본 매크로 설정

a. $__database를 Timestream 데이터베이스의 이름으로 설정(예: grafanaDB)
b. $__table을 Timestream 테이블의 이름으로 설정(예: grafanaTable)
c. $__measure를 탭에서 가장 일반적으로 사용되는 측정값으로 설정

12. 저장 및 테스트를 클릭합니다.
13. 대시보드 탭을 클릭합니다.
14. 가져오기를 클릭하여 대시보드를 가져옵니다.
15. 샘플 애플리케이션 대시보드를 두 번 클릭합니다.
16. 대시보드 설정을 클릭합니다.
17. 변수 선택
18. Timestream 데이터베이스 및 테이블의 이름과 tableName 일치하도록 dbName 및 변경
19. 저장을 클릭합니다.
20. 대시보드 새로 고침
21. 알림을 생성하려면 Grafana 설명서에 설명된 지침에 따라 Grafana 관리형 알림 규칙 생성
22. 경보 문제를 해결하려면 Grafana 문제 해결 설명서에 설명된 지침을 따르세요.
23. 자세한 내용은 Grafana 설명서를 참조하세요.

비디오 자습서

이 동영상에서는 Grafana가 Timestream에서 작동하는 방법을 설명합니다.

SquaredUp 를 사용하여 Amazon Timestream 작업

SquaredUp 는 Amazon Timestream과 통합되는 관찰 플랫폼입니다. SquaredUp의 직관적인 대시보드 디자이너를 사용하여 시계열 데이터를 시각화, 분석 및 모니터링할 수 있습니다. 대시보드는 공개적으로 또는 비공개로 공유할 수 있으며, 알림 채널을 생성하여 모니터의 상태가 변경될 때 알림을 보낼 수 있습니다.

Amazon Timestream SquaredUp 과 함께 사용

1. 에 가입SquaredUp하고 무료로 시작하세요.
2. AWS 데이터 소스 를 추가합니다.
3. Timestream Query 데이터 스트림을 사용하는 대시보드 타일을 생성합니다.
4. 필요에 따라 타일에 대한 모니터링을 활성화하거나, 알림 채널을 생성하거나, 대시보드를 공개적으로 또는 비공개로 공유할 수 있습니다.

5. 선택적으로 다른 타일을 생성하여 다른 모니터링 및 관찰 도구의 데이터와 함께 Timestream 데이터를 볼 수 있습니다.

오픈 소스 Telegraf

Telegraf용 LiveAnalytics 출력 플러그인에 Timestream을 사용하여 오픈 소스 Telegraf에서 직접 에 대한 LiveAnalytics 지표를 Timestream에 쓸 수 있습니다.

이 섹션에서는 출력 플러그인용 Timestream과 함께 Telegraf를 설치하는 방법, LiveAnalytics 출력 플러그인용 Timestream과 함께 Telegraf를 실행하는 방법 LiveAnalytics , 오픈 소스 Telegraf가 용 Timestream과 작동하는 방법에 대해 설명합니다 LiveAnalytics.

주제

- 출력 플러그인용 Timestream을 LiveAnalytics 사용하여 Telegraf 설치
- LiveAnalytics 출력 플러그인용 Timestream으로 Telegraf 실행
- Telegraf/InfluxDB 지표를 모델의 Timestream에 LiveAnalytics 매핑

출력 플러그인용 Timestream을 LiveAnalytics 사용하여 Telegraf 설치

버전 1.16부터 LiveAnalytics 출력 플러그인의 Timestream은 공식 Telegraf 릴리스에서 사용할 수 있습니다. 대부분의 주요 운영 체제에 출력 플러그인을 설치하려면 InfluxData Telegraf 설명서 에 설명된 단계를 따릅니다. Amazon Linux 2 OS에 를 설치하려면 아래 지침을 따르세요.

Amazon Linux 2에서 LiveAnalytics 출력 플러그인용 Timestream으로 Telegraf 설치

Amazon Linux 2에 Timestream 출력 플러그인이 있는 Telegraf를 설치하려면 다음 단계를 수행합니다.

1. yum 패키지 관리자를 사용하여 Telegraf를 설치합니다.

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdb.repo
[influxdb]
name = InfluxDB Repository - RHEL \$releasever
baseurl = https://repos.influxdata.com/rhel/\$releasever/\$basearch/stable
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdb.key
EOF
```

2. 다음 명령을 실행합니다.

```
sudo sed -i "s/\$releasever/$(rpm -E %{rhel})/g" /etc/yum.repos.d/influxdb.repo
```

3. Telegraf를 설치하고 시작합니다.

```
sudo yum install telegraf
sudo service telegraf start
```

LiveAnalytics 출력 플러그인용 Timestream으로 Telegraf 실행

아래 지침에 따라 LiveAnalytics 플러그인용 Timestream을 사용하여 Telegraf를 실행할 수 있습니다.

1. Telegraf를 사용하여 예제 구성을 생성합니다.

```
telegraf --section-filter agent:inputs:outputs --input-filter cpu:mem --output-
filter timestream config > example.config
```

2. 관리 콘솔 , 또는 를 사용하여 Timestream에서 데이터베이스를 생성합니다SDKs. CLI
3. `example.config` 파일에서 `[[outputs.timestream]]` 섹션 아래의 다음 키를 편집하여 데이터베이스 이름을 추가합니다.

```
database_name = "yourDatabaseNameHere"
```

4. 기본적으로 Telegraf는 테이블을 생성합니다. 테이블을 수동으로 생성하려면 `false` `create_table_if_not_exists`로 설정하고 지침에 따라 관리 콘솔 , 또는 를 사용하여 테이블을 생성합니다SDKs. CLI
5. example.config 파일의 `[[outputs.timestream]]` 섹션에서 자격 증명을 구성합니다. 자격 증명은 다음 작업을 허용해야 합니다.

```
timestream:DescribeEndpoints
timestream:WriteRecords
```

> **Note**
>
> 를 로 `create_table_if_not_exists` 설정한 상태로 두면 다음을 `true`포함합니다.
>
> ```
> timestream:CreateTable
> ```

> **Note**
>
> 를 `describe_database_on_start`로 설정한 경우 다음을 `true`포함합니다.
>
> ```
> timestream:DescribeDatabase
> ```

6. 기본 설정에 따라 나머지 구성을 편집할 수 있습니다.
7. 구성 파일 편집이 완료되면 다음을 사용하여 Telegraf를 실행합니다.

```
./telegraf --config example.config
```

8. 지표는 에이전트 구성에 따라 몇 초 내에 표시되어야 합니다. 또한 Timestream 콘솔에 새 테이블인 cpu 및 mem 도 표시됩니다.

Telegraf/InfluxDB 지표를 모델의 Timestream에 LiveAnalytics 매핑

LiveAnalytics에 대해 Telegraf에서 Timestream으로 데이터를 작성할 때 데이터는 다음과 같이 매핑됩니다.

- 타임스탬프는 시간 필드로 기록됩니다.
- 태그는 차원으로 작성됩니다.
- 필드는 측정값으로 기록됩니다.
- 측정값은 대부분 테이블 이름으로 작성됩니다(자세한 내용은 아래 참조).

Telegraf용 LiveAnalytics 출력 플러그인의 Timestream은 용 Timestream에 데이터를 구성하고 저장하기 위한 여러 옵션을 제공합니다 LiveAnalytics. 이는 일렬 프로토콜 형식의 데이터로 시작하는 예제로 설명할 수 있습니다.

```
weather,location=us-midwest,season=summer temperature=82,humidity=71
1465839830100400200 airquality,location=us-west no2=5,pm25=16
1465839830100400200
```

다음은 데이터를 설명합니다.

- 측정 이름은 `weather` 및 입니다`airquality`.

- 태그는 `location` 및 입니다`season`.
- 필드는 `temperature`, `no2`, `humidity`및 입니다`pm25`.

주제

- 여러 테이블에 데이터 저장
- 데이터를 단일 테이블에 저장

여러 테이블에 데이터 저장

측정당 별도의 테이블을 생성하고 각 필드를 테이블당 별도의 행에 저장하도록 선택할 수 있습니다.

구성은 입니다`mapping_mode = "multi-table"`.

- LiveAnalytics 어댑터의 Timestream은 `weather` 및 라는 두 개의 테이블을 생성합니다`airquality`.
- 각 테이블 행에는 단일 필드만 포함됩니다.

LiveAnalytics 테이블 `weather` 및 에 대한 결과 Timestream`airquality`은 다음과 같습니다.

`weather`

시간	location	시즌	measure_name	measure_value::bigint
2016-06-13 17:43:50	us-미드웨스트	여름	temperature	82
2016-06-13 17:43:50	us-미드웨스트	여름	습도	71

`airquality`

시간	location	measure_name	measure_value::bigint
2016-06-13 17:43:50	us-미드웨스트	no2	5
2016-06-13 17:43:50	us-미드웨스트	pm25	16

데이터를 단일 테이블에 저장

모든 측정값을 단일 테이블에 저장하고 각 필드를 별도의 테이블 행에 저장하도록 선택할 수 있습니다.

구성은 입니다`mapping_mode = "single-table"`. `single-table`, `single_table_name` 및 를 사용할 때는 두 가지 추가 구성이 있습니다`single_table_dimension_name_for_telegraf_measurement_name`.

- LiveAnalytics 출력 플러그인의 Timestream은 이름이 인 단일 테이블을 생성합니다.*<single_table_name>* 여기에는 *<single_table_dimension_name_for_telegraf_measurement_name>* 열.
- 테이블에는 단일 테이블 행에 여러 필드가 포함될 수 있습니다.

LiveAnalytics 테이블의 결과 Timestream은 다음과 같습니다.

weather

시간	location	시즌	*<single_table_dimension_name_for_telegraf_measurement_name>*	measure_name	measure_value::bigint
2016-06-13 17:43:50	us-미드웨스트	여름	날씨	temperature	82
2016-06-13 17:43:50	us-미드웨스트	여름	날씨	습도	71
2016-06-13 17:43:50	us-미드웨스트	여름	공기질	no2	5
2016-06-13 17:43:50	us-미드웨스트	여름	날씨	pm25	16

JDBC

Java Database Connectivity(JDBC) 연결을 사용하여 용 Timestream LiveAnalytics 을 비즈니스 인텔리전스 도구 및 SQL Workbench와 같은 기타 애플리케이션에 연결할 수 있습니다. 드라이버용 LiveAnalytics JDBC Timestream은 현재 Okta 및 Microsoft Azure ADSSO에서 를 지원합니다.

주제

- 에 대한 Timestream용 JDBC 드라이버 구성 LiveAnalytics
- 연결 속성
- JDBC URL 예제
- Okta를 사용한 Single Sign-On 인증을 위한 LiveAnalytics JDBC Timestream 설정
- Microsoft Azure AD를 사용한 Single Sign-On 인증을 위한 LiveAnalytics JDBC Timestream 설정

에 대한 Timestream용 JDBC 드라이버 구성 LiveAnalytics

아래 단계에 따라 JDBC 드라이버를 구성합니다.

주제

- 드라이버의 LiveAnalytics JDBC 타임스트림 JARs
- 드라이버 클래스 및 URL 형식의 LiveAnalytics JDBC Timestream
- 샘플 애플리케이션

드라이버의 LiveAnalytics JDBC 타임스트림 JARs

드라이버를 Maven 종속성으로 추가하거나 직접 다운로드를 통해 드라이버의 LiveAnalytics JDBC Timestream을 가져올 수 있습니다.

- 직접 다운로드:. JDBC 드라이버용 LiveAnalytics Timestream을 직접 다운로드하려면 다음 단계를 완료합니다.

  1. https://github.com/awslabs/amazon-timestream-driver-jdbc/릴리스로 이동
  2. 비즈니스 인텔리전스 도구 및 애플리케이션과 함께 `amazon-timestream-jdbc-1.0.1-shaded.jar` 직접 사용할 수 있습니다.
  3. 원하는 디렉터리`amazon-timestream-jdbc-1.0.1-javadoc.jar`로 다운로드합니다.
  4. 를 다운로드한 디렉터리에서 다음 명령을 `amazon-timestream-jdbc-1.0.1-javadoc.jar`실행하여 Javadoc HTML 파일을 추출합니다.

```
jar -xvf amazon-timestream-jdbc-1.0.1-javadoc.jar
```

- Maven 종속성: 드라이버의 LiveAnalytics JDBC Timestream을 Maven 종속성으로 추가하려면 다음 단계를 완료합니다.

  1. 선택한 편집기에서 애플리케이션 `pom.xml` 파일로 이동하여 엽니다.
  2. JDBC 드라이버를 애플리케이션 `pom.xml` 파일에 종속성으로 추가합니다.

```
<!-- https://mvnrepository.com/artifact/software.amazon.timestream/amazon-
timestream-jdbc -->
<dependency>
    <groupId>software.amazon.timestream</groupId>
    <artifactId>amazon-timestream-jdbc</artifactId>
    <version>1.0.1</version>
</dependency>
```

드라이버 클래스 및 URL 형식의 LiveAnalytics JDBC Timestream

드라이버용 Timestream의 LiveAnalytics JDBC 드라이버 클래스는 다음과 같습니다.

```
software.amazon.timestream.jdbc.TimestreamDriver
```

Timestream JDBC 드라이버에는 다음 JDBC URL 형식이 필요합니다.

```
jdbc:timestream:
```

를 통해 데이터베이스 속성을 지정하려면 다음 URL 형식을 JDBC URL사용합니다.

```
jdbc:timestream://
```

샘플 애플리케이션

에서 용 Timestream 사용을 시작하는 데 도움이 되도록 LiveAnalytics 에서 전체 기능을 갖춘 샘플 애플리케이션을 JDBC만들었습니다 GitHub.

1. 여기에 설명된 지침에 따라 샘플 데이터가 포함된 데이터베이스를 생성합니다.
2. 의 지침에 따라 에 대한 샘플 애플리케이션의 JDBC GitHub 리포지토리를 복제합니다GitHub.

3. 의 지침에 따라 샘플 애플리케이션을 README 시작합니다.

연결 속성

드라이버용 LiveAnalytics JDBC Timestream은 다음 옵션을 지원합니다.

주제

- 기본 인증 옵션
- 표준 클라이언트 정보 옵션
- 드라이버 구성 옵션
- SDK 옵션
- 엔드포인트 구성 옵션
- 자격 증명 공급자 옵션
- SAMLOkta에 대한 기반 인증 옵션
- SAMLAzure AD에 대한 기반 인증 옵션

> **Note**
>
> 속성이 제공되지 않으면 드라이버용 LiveAnalytics JDBC Timestream은 기본 보안 인증 체인을 사용하여 보안 인증 정보를 로드합니다.

> **Note**
>
> 모든 속성 키는 대/소문자를 구분합니다.

기본 인증 옵션

다음 표에서는 사용 가능한 기본 인증 옵션을 설명합니다.

옵션	설명	기본값
AccessKeyId	AWS 사용자 액세스 키 ID입니다.	NONE

옵션	설명	기본값
SecretAccessKey	AWS 사용자 보안 액세스 키입니다.	NONE
SessionToken	다중 인증(MFA)이 활성화된 데이터베이스에 액세스하는 데 필요한 임시 세션 토큰입니다.	NONE

표준 클라이언트 정보 옵션

다음 표에서는 표준 클라이언트 정보 옵션을 설명합니다.

옵션	설명	기본값
ApplicationName	현재 연결을 사용하는 애플리케이션의 이름입니다. `ApplicationName` 는 디버깅 목적으로 사용되며 Timestream for LiveAnalytics Service에 전달되지 않습니다.	드라이버에서 감지한 애플리케이션 이름입니다.

드라이버 구성 옵션

다음 표에서는 드라이버 구성 옵션을 설명합니다.

옵션	설명	기본값
EnableMetaDataPreparedStatement	드라이버가 에 대한 LiveAnalytics JDBC 메타데이터를 반환하도록 Timestream을 활성화하지만 `PreparedStatements` 메타데이터를 검색할 LiveAnalytics 때 Timestream에 추가 비용이 발생합니다.	FALSE

옵션	설명	기본값
리전	데이터베이스의 리전입니다.	us-east-1

SDK 옵션

다음 표에서는 SDK 옵션에 대해 설명합니다.

옵션	설명	기본값
RequestTimeout	가 시간 초과 전에 쿼리 요청을 기다리는 밀리초 AWS SDK 단위의 시간입니다. 긍정이 아닌 값은 요청 제한 시간을 비활성화합니다.	0
SocketTimeout	시간이 초과되기 전에 가 AWS SDK 열린 연결을 통해 데이터가 전송될 때까지 밀리초 단위로 대기하는 시간입니다. 값은 음수가 아니어야 합니다. 값은 소켓 제한 시간을 `0` 비활성화합니다.	50000
MaxRetryCountClient	에서 5XX 오류 코드를 사용하여 재시도 가능한 오류에 대한 최대 재시도 횟수입니다SDK. 값은 음수가 아니어야 합니다.	NONE
MaxConnections	Timestream for LiveAnalytics Service에 대해 동시에 열린 최대 허용 HTTP 연결 수입니다. 값은 양수여야 합니다.	50

엔드포인트 구성 옵션

다음 표에서는 엔드포인트 구성 옵션을 설명합니다.

옵션	설명	기본값
엔드포인트	Timestream for LiveAnalytics Service의 엔드포인트입니다.	NONE

자격 증명 공급자 옵션

다음 표에서는 사용 가능한 자격 증명 공급자 옵션을 설명합니다.

옵션	설명	기본값
AwsCredentialsProviderClass	인증`InstanceProfileCredentialsProvider` 에 사용할 `PropertiesFileCredentialsProvider` 또는 중 하나입니다.	NONE
CustomCredentialsFilePath	AWS 보안 자격 증명 `accessKey` 및 가 포함된 속성 파일의 경로입니다`secretKey` . 이 `PropertiesFileCredentialsProvider` 로 `AwsCredentialsProviderClass` 지정된 경우에만 필요합니다.	NONE

SAMLOkta에 대한 기반 인증 옵션

다음 표에서는 Okta에 사용할 수 있는 SAML기반 인증 옵션을 설명합니다.

옵션	설명	기본값
IdpName	SAML기반 인증에 사용할 Identity Provider(Idp) 이름	NONE

옵션	설명	기본값
	입니다. `Okta` 또는 중 하나`AzureAD`.	
IdpHost	지정된 Idp의 호스트 이름입니다.	NONE
IdpUserName	지정된 Idp 계정의 사용자 이름입니다.	NONE
IdpPassword	지정된 Idp 계정의 암호입니다.	NONE
OktaApplicationID	Timestream for LiveAnalytics Application과 연결된 고유한 Okta 제공 ID입니다. 애플리케이션 메타데이터에 제공된 `entityID` 필드에서 찾을 `AppId` 수 있습니다. 다음 예를 고려해 보세요. `entityID = http://www.okta.com//IdpAppID`	NONE
역할ARN	호출자가 수임하는 역할의 Amazon 리소스 이름(ARN)입니다.	NONE
IdpARN	Idp를 IAM 설명하는 의 SAML 공급자의 Amazon 리소스 이름(ARN)입니다.	NONE

SAMLAzure AD에 대한 기반 인증 옵션

다음 표에서는 Azure AD에 사용할 수 있는 SAML기반 인증 옵션을 설명합니다.

옵션	설명	기본값
IdpName	SAML기반 인증에 사용할 Identity Provider(Idp) 이름입니다. `Okta` 또는 중 하나`AzureAD`.	NONE
IdpHost	지정된 Idp의 호스트 이름입니다.	NONE
IdpUserName	지정된 Idp 계정의 사용자 이름입니다.	NONE
IdpPassword	지정된 Idp 계정의 암호입니다.	NONE
AADApplicationID	Azure AD에 등록된 애플리케이션의 고유 ID입니다.	NONE
AADClientSecret	가져오기 토큰을 승인하는 데 사용되는 Azure AD의 등록된 애플리케이션과 연결된 클라이언트 보안 암호입니다.	NONE
AADTenant	Azure AD 테넌트 ID입니다.	NONE
IdpARN	Idp를 IAM 설명하는 의 SAML 공급자의 Amazon 리소스 이름(ARN)입니다.	NONE

JDBC URL 예제

이 섹션에서는 JDBC 연결을 생성하는 방법을 설명하고 예제를 URL제공합니다. 선택적 연결 속성 을 지정하려면 다음 URL 형식을 사용합니다.

```
jdbc:timestream://PropertyName1=value1;PropertyName2=value2...
```

> **Note**
>
> 모든 연결 속성은 선택 사항입니다. 모든 속성 키는 대/소문자를 구분합니다.

다음은 JDBC 연결 의 몇 가지 예입니다URLs.

기본 인증 옵션 및 리전이 있는 예제:

```
jdbc:timestream://
AccessKeyId=<myAccessKeyId>;SecretAccessKey=<mySecretAccessKey>;SessionToken=<mySessionToken
east-1
```

클라이언트 정보, 리전 및 SDK 옵션이 포함된 예제:

```
jdbc:timestream://ApplicationName=MyApp;Region=us-
east-1;MaxRetryCountClient=10;MaxConnections=5000;RequestTimeout=20000
```

환경 변수에 자격 증명이 설정된 기본 AWS 자격 증명 공급자 체인을 사용하여 연결합니다.

```
jdbc:timestream
```

연결 에서 자격 증명이 설정된 기본 AWS 자격 증명 공급자 체인을 사용하여 연결합니다URL.

```
jdbc:timestream://
AccessKeyId=<myAccessKeyId>;SecretAccessKey=<mySecretAccessKey>;SessionToken=<mySessionToken
```

를 PropertiesFileCredentialsProvider 인증 방법으로 사용하여 연결합니다.

```
jdbc:timestream://
AwsCredentialsProviderClass=PropertiesFileCredentialsProvider;CustomCredentialsFilePath=<pat
 to properties file>
```

를 InstanceProfileCredentialsProvider 인증 방법으로 사용하여 연결합니다.

```
jdbc:timestream://AwsCredentialsProviderClass=InstanceProfileCredentialsProvider
```

Okta 자격 증명을 인증 방법으로 사용하여 연결합니다.

```
jdbc:timestream://
IdpName=Okta;IdpHost=<host>;IdpUserName=<name>;IdpPassword=<password>;OktaApplicationID=<id>
```

Azure AD 자격 증명을 인증 방법으로 사용하여 연결합니다.

```
jdbc:timestream://
IdpName=AzureAD;IdpUserName=<name>;IdpPassword=<password>;AADApplicationID=<id>;AADClientSec
```

특정 엔드포인트와 연결:

```
jdbc:timestream://Endpoint=abc.us-east-1.amazonaws.com;Region=us-east-1
```

Okta를 사용한 Single Sign-On 인증을 위한 LiveAnalytics JDBC Timestream 설정

용 Timestream은 Okta를 사용한 Single Sign-On 인증을 위해 LiveAnalytics JDBC Timestream을 LiveAnalytics 지원합니다. Okta를 사용한 Single Sign-On 인증에 LiveAnalytics JDBC Timestream을 사용하려면 아래 나열된 각 섹션을 완료하세요.

주제

- 사전 조건
- AWS Okta의 계정 페더레이션
- 에 대한 Okta 설정 SAML

사전 조건

Okta를 사용한 JDBC Single Sign-On 인증에 LiveAnalytics Timestream을 사용하기 전에 다음 사전 조건을 충족했는지 확인합니다.

- 에서 자격 증명 공급자 및 역할을 AWS 생성할 수 있는 관리자 권한입니다.
- Okta 계정(계정 생성https://www.okta.com/login/으로 이동).
- 용 Amazon Timestream에 대한 액세스입니다 LiveAnalytics.

이제 사전 조건을 완료했으므로 로 진행할 수 있습니다AWS Okta의 계정 페더레이션.

AWS Okta의 계정 페더레이션

드라이버의 LiveAnalytics JDBC Timestream은 Okta의 AWS 계정 페더레이션을 지원합니다. Okta에서 AWS 계정 페더레이션을 설정하려면 다음 단계를 완료하세요.

1. 다음 를 사용하여 Okta Admin 대시보드에 로그인합니다URL.

```
https://<company-domain-name>-admin.okta.com/admin/apps/active
```

> **Note**
>
> <company-domain-name>를 도메인 이름으로 바꿉니다.

2. 로그인에 성공하면 애플리케이션 추가를 선택하고 AWS 계정 통합을 검색합니다.
3. 추가를 선택합니다.
4. 로그인을 적절한 URL로 변경합니다URL.
5. 다음을 선택합니다.
6. SAML 2.0을 로그인 방법으로 선택
7. Identity Provider 메타데이터를 선택하여 메타데이터 XML 파일을 엽니다. 파일을 로컬에 저장합니다.
8. 다른 모든 구성 옵션은 비워 둡니다.
9. 완료를 선택합니다.

이제 Okta에서 AWS 계정 페더레이션을 완료했으므로 로 진행할 수 있습니다에 대한 Okta 설정 SAML.

에 대한 Okta 설정 SAML

1. 로그인(Sign On) 탭을 선택합니다. 보기 를 선택합니다.
2. 설정 섹션에서 설정 지침 버튼을 선택합니다.

Okta 메타데이터 문서 찾기

1. 문서를 찾으려면 다음으로 이동합니다.

```
https://<domain>-admin.okta.com/admin/apps/active
```

> **Note**
>
> <domain>은 Okta 계정의 고유한 도메인 이름입니다.

2. AWS Account Federation 애플리케이션 선택
3. 로그인 탭을 선택합니다.

Microsoft Azure AD를 사용한 Single Sign-On 인증을 위한 LiveAnalytics JDBC Timestream 설정

의 Timestream은 Microsoft Azure AD를 사용한 Single Sign-On 인증을 위한 LiveAnalytics JDBC Timestream을 LiveAnalytics 지원합니다. Microsoft Azure AD를 사용한 Single Sign-On 인증에 LiveAnalytics JDBC Timestream을 사용하려면 아래 나열된 각 섹션을 완료하세요.

주제

- 사전 조건
- Azure AD 설정
- 에서 IAM Identity Provider 및 역할 설정 AWS

사전 조건

Microsoft Azure AD를 사용한 JDBC Single Sign-On 인증에 LiveAnalytics Timestream을 사용하기 전에 다음 사전 조건을 충족했는지 확인합니다.

- 에서 자격 증명 공급자 및 역할을 AWS 생성할 수 있는 관리자 권한입니다.
- Azure Active Directory 계정(https://azure.microsoft.com/en-ca/Services/active-directory/로 이동하여 계정 생성)
- 용 Amazon Timestream에 대한 액세스입니다 LiveAnalytics.

Azure AD 설정

1. Azure Portal에 로그인
2. Azure 서비스 목록에서 Azure Active Directory를 선택합니다. 그러면 기본 디렉터리 페이지로 리디렉션됩니다.
3. 사이드바의 관리 섹션에서 엔터프라이즈 애플리케이션을 선택합니다.

4. + 새 애플리케이션 을 선택합니다.
5. Amazon Web Services 를 찾아 선택합니다.
6. 사이드바의 관리 섹션에서 Single Sign-On을 선택합니다.
7. Single Sign-On 메SAML서드로 선택
8. 기본 SAML 구성 섹션에서 식별자와 회신 모두에 URL 대해 다음을 입력합니다URL.

```
https://signin.aws.amazon.com/saml
```

9. 저장을 선택합니다.
10. 인증서 SAML 서명 섹션에서 페더레이션 메타데이터XML를 다운로드합니다. 이는 나중에 IAM Identity Provider를 생성할 때 사용됩니다.
11. 기본 디렉터리 페이지로 돌아가서 관리에서 앱 등록을 선택합니다.
12. 모든 애플리케이션 섹션에서 에 대한 Timestream LiveAnalytics을 선택합니다. 페이지가 애플리케이션의 개요 페이지로 리디렉션됩니다.

> **Note**
>
> 애플리케이션(클라이언트) ID와 디렉터리(테넌트) ID를 기록해 둡니다. 이러한 값은 연결을 생성할 때 필요합니다.

13. 인증서 및 보안 암호 선택
14. 클라이언트 보안 암호 에서 + 새 클라이언트 보안 암호 를 사용하여 새 클라이언트 보안 암호를 생성합니다.

> **Note**
>
> 생성된 클라이언트 보안 암호는 에 대한 Timestream 연결을 생성할 때 필요하므로 기록해 둡니다 LiveAnalytics.

15. 사이드바의 관리에서 API 권한을 선택합니다.
16. 구성된 권한 에서 권한 추가를 사용하여 Azure AD에 에 대한 Timestream에 로그인할 수 있는 권한을 부여합니다 LiveAnalytics. API 권한 요청 페이지에서 Microsoft Graph를 선택합니다.
17. 위임된 권한을 선택하고 User.Read 권한을 선택합니다.
18. 권한 추가를 선택합니다.
19. 기본 디렉터리에 대한 관리자 동의 부여를 선택합니다.

에서 IAM Identity Provider 및 역할 설정 AWS

Microsoft Azure AD를 사용한 Single Sign-On 인증을 IAM 위한 Timestream을 LiveAnalytics JDBC 설정하려면 아래의 각 섹션을 완료하세요.

주제

- SAML 자격 증명 공급자 생성
- IAM 역할 생성
- IAM 정책 생성
- 프로비저닝

SAML 자격 증명 공급자 생성

Microsoft Azure AD를 사용한 Single Sign-On 인증을 위한 LiveAnalytics JDBC Timestream용 SAML Identity Provider를 생성하려면 다음 단계를 완료합니다.

1. AWS 관리 콘솔에 로그인
2. 서비스를 선택하고 보안, 자격 증명 및 규정 준수IAM에서 를 선택합니다.
3. 액세스 관리에서 자격 증명 공급자 선택
4. 공급자 생성을 선택하고 공급자 유형SAML으로 를 선택합니다. 공급자 이름 을 입력합니다. 이 예제에서는 A 를 사용합니다zureADProvider.
5. 이전에 다운로드한 페더레이션 메타데이터 XML 파일 업로드
6. 다음을 선택한 다음 생성을 선택합니다.
7. 완료되면 페이지가 자격 증명 공급자 페이지로 다시 리디렉션됩니다.

IAM 역할 생성

Microsoft Azure AD를 사용한 Single Sign-On 인증을 위한 LiveAnalytics JDBC Timestream의 IAM 역할을 생성하려면 다음 단계를 완료합니다.

1. 사이드바에서 액세스 관리에서 역할을 선택합니다.
2. 역할 생성을 선택합니다.
3. 신뢰할 수 있는 엔터티로 SAML 2.0 페더레이션 선택
4. Azure AD 공급자 선택
5. 프로그래밍 방식 및 AWS 관리 콘솔 액세스 허용을 선택합니다.

6. 다음: 권한 선택
7. 권한 정책을 연결하거나 다음:Tags로 계속 진행합니다.
8. 선택적 태그를 추가하거나 다음:검토로 계속 진행합니다.
9. 역할 이름을 입력합니다. 이 예제에서는 A를 사용합니다.zureSAMLRole
10. 역할 설명 제공
11. 완료할 역할 생성을 선택합니다.

IAM 정책 생성

Microsoft Azure AD를 사용한 Single Sign-On 인증을 위한 LiveAnalytics JDBC Timestream IAM 정책을 생성하려면 다음 단계를 완료하세요.

1. 사이드바에서 액세스 관리에서 정책을 선택합니다.
2. 정책 생성을 선택하고 JSON 탭을 선택합니다.
3. 다음 정책 추가

```
{
"Version": "2012-10-17",
"Statement": [
    {
          "Effect": "Allow",
          "Action": [
                "iam:ListRoles",
                "iam:ListAccountAliases"
           ],
           "Resource": "*"
      }
]
}
```

4. 정책 생성(Create policy)을 선택합니다.
5. 정책 이름을 입력합니다. 이 예제에서는 를 사용합니다 TimestreamAccessPolicy.
6. 정책 생성을 선택합니다.
7. 사이드바에서 액세스 관리에서 역할을 선택합니다.
8. 이전에 생성한 Azure AD 역할을 선택하고 권한에서 정책 연결을 선택합니다.
9. 이전에 생성한 액세스 정책을 선택합니다.

프로비저닝

Microsoft Azure AD를 사용한 Single Sign-On 인증을 위해 Timestream용 LiveAnalytics JDBC 자격 증명 공급자를 프로비저닝하려면 다음 단계를 완료합니다.

1. Azure 포털로 돌아가기
2. Azure 서비스 목록에서 Azure Active Directory를 선택합니다. 그러면 기본 디렉터리 페이지로 리디렉션됩니다.
3. 사이드바의 관리 섹션에서 엔터프라이즈 애플리케이션을 선택합니다.
4. 프로비저닝 선택
5. 프로비저닝 방법의 자동 모드 선택
6. 관리자 보안 인증 정보에서 클라이언트 보안 암호의 ID와 보안 암호 토큰SecretAccessKey의 AwsAccessKeyID를 입력합니다.
7. 프로비저닝 상태를 켜기로 설정
8. 저장을 선택합니다. 이렇게 하면 Azure AD가 필요한 IAM 역할을 로드할 수 있습니다.
9. 현재 주기 상태가 완료되면 사이드바에서 사용자 및 그룹을 선택합니다.
10. 선택 + 사용자 추가
11. 에 대한 Timestream 액세스를 제공할 Azure AD 사용자 선택 LiveAnalytics
12. 에서 생성된 IAM Azure AD 역할 및 해당 Azure Identity Provider를 선택합니다. AWS
13. 할당 선택

ODBC

용 Amazon Timestream용 오픈 소스 ODBC 드라이버는 개발자에게 Timestream에 LiveAnalytics 대한 SQL관계형 인터페이스를 LiveAnalytics 제공하고 Power BI 데스크톱 및 Microsoft Excel과 같은 비즈니스 인텔리전스(BI) 도구에서 연결할 수 있도록 합니다. 드라이버에 대한 LiveAnalytics ODBC Timestream은 현재 Windows, macOS 및 Linux 에서 사용할 수 있으며 Okta 및 Microsoft Azure Active Directory(AD)를 SSO 지원합니다.

자세한 내용은 Amazon Timestream에서 LiveAnalytics 의 ODBC 드라이버 설명서를 GitHub 참조하세요.

주제

- 드라이버의 Timestream LiveAnalytics ODBC 설정
- ODBC 드라이버에 대한 연결 문자열 구문 및 옵션

- 드라이버용 Timestream LiveAnalytics ODBC의 연결 문자열 예제
- ODBC 드라이버와의 연결 문제 해결

드라이버의 Timestream LiveAnalytics ODBC 설정

AWS 계정 LiveAnalytics 에서 의 Timestream에 대한 액세스 설정

에 대한 Timestream을 사용하도록 AWS 계정을 아직 설정하지 않은 경우 의 축소를 LiveAnalytics따르세요에 대한 Timestream 액세스 LiveAnalytics.

시스템에 ODBC 드라이버 설치

ODBC GitHub 리포지토리 에서 시스템에 적합한 Timestream ODBC 드라이버 설치 관리자를 다운로드하고 시스템에 적용되는 설치 지침을 따릅니다.

- Windows 설치 가이드
- MacOS 설치 가이드
- Linux 설치 가이드

ODBC 드라이버의 데이터 소스 이름(DSN) 설정

시스템 DSN 구성 가이드의 지침을 따릅니다.

- Windows DSN 구성
- MacOS DSN 구성
- Linux DSN 구성

ODBC 드라이버와 함께 작동하도록 비즈니스 인텔리전스(BI) 애플리케이션 설정

다음은 ODBC 드라이버와 함께 작동하도록 몇 가지 일반적인 BI 애플리케이션을 설정하는 지침입니다.

- Microsoft Power BI 설정.
- Microsoft Excel 설정
- Tableau 설정

다른 애플리케이션의 경우

ODBC 드라이버에 대한 연결 문자열 구문 및 옵션

ODBC 드라이버에 대한 연결 문자열 옵션을 지정하는 구문은 다음과 같습니다.

```
DRIVER={Amazon Timestream ODBC Driver};(option)=(value);
```

사용 가능한 옵션은 다음과 같습니다.

드라이버 연결 옵션

- **Driver** (필수) - 에서 사용 중인 드라이버입니다ODBC.

  기본값은 Amazon Timestream입니다.
- **DSN** - 연결을 구성하는 데 사용할 데이터 소스 이름(DSN)입니다.

  기본값은 NONE입니다.
- **Auth** - 인증 모드입니다. 이 값은 다음 중 하나여야 합니다.
  - AWS_PROFILE - 기본 자격 증명 체인을 사용합니다.
  - IAM – 보안 인증 정보를 사용합니다 AWS IAM.
  - AAD – Azure Active Directory(AD) 자격 증명 공급자를 사용합니다.
  - OKTA – Okta 자격 증명 공급자를 사용합니다.

  기본값은 AWS_PROFILE입니다.

엔드포인트 구성 옵션

- **EndpointOverride** - Timestream for LiveAnalytics Service의 엔드포인트 재정의입니다. 이는 리전을 재정의하는 고급 옵션입니다. 예:

  ```
  query-cell2.timestream.us-east-1.amazonaws.com
  ```
- **Region** - Timestream for LiveAnalytics Service 엔드포인트의 서명 리전입니다.

  기본값은 us-east-1입니다.

자격 증명 공급자 옵션

- **ProfileName** - AWS 구성 파일의 프로필 이름입니다.

기본값은 NONE입니다.

AWS IAM 인증 옵션

- **UID** 또는 **AccessKeyId**- AWS 사용자 액세스 키 ID입니다. 연결 문자열에 UID 및 AccessKeyId 가 모두 제공되는 경우 값이 비어 있지 않은 한 UID 값이 사용됩니다.

  기본값은 NONE입니다.
- **PWD** 또는 **SecretKey** - AWS 사용자 보안 액세스 키입니다. 연결 문자열에 PWD 및 SecretKey 가 모두 제공되는 경우 값이 비어 있지 않은 한 가 있는 PWD 값이 사용됩니다.

  기본값은 NONE입니다.
- **SessionToken** - 다중 인증(MFA)이 활성화된 데이터베이스에 액세스하는 데 필요한 임시 세션 토큰입니다. 입력 = 에 후행을 포함하지 마십시오.

  기본값은 NONE입니다.

SAMLOkta에 대한 기반 인증 옵션

- **IdPHost** - 지정된 IdP 의 호스트 이름입니다.

  기본값은 NONE입니다.
- **UID** 또는 **IdPUserName** - 지정된 IdP 계정의 사용자 이름입니다. 연결 문자열에 UID 및 IdPUserName 가 모두 제공되는 경우 값이 비어 있지 않은 한 UID 값이 사용됩니다.

  기본값은 NONE입니다.
- **PWD** 또는 **IdPPassword** - 지정된 IdP 계정의 암호입니다. 연결 문자열에 PWD 및 IdPPassword 가 모두 제공되는 경우 값이 비어 있지 않은 한 PWD 값이 사용됩니다.

  기본값은 NONE입니다.
- **OktaApplicationID** - LiveAnalytics 애플리케이션용 Timestream과 연결된 고유한 Okta 제공 ID입니다. 애플리케이션 ID(AppId)를 찾을 수 있는 위치는 애플리케이션 메타데이터에 제공된 entityID 필드에 있습니다. 예는 다음과 같습니다.

  ```
  entityID="http://www.okta.com//(IdPAppID)
  ```

  기본값은 NONE입니다.

- **RoleARN** - 호출자가 수임하는 역할의 Amazon 리소스 이름(ARN)입니다.

  기본값은 NONE입니다.
- **IdPARN** - IdP 를 IAM 설명하는 의 SAML 공급자의 Amazon 리소스 이름(ARN)입니다.

  기본값은 NONE입니다.

SAMLAzure Active Directory에 대한 기반 인증 옵션

- **UID** 또는 **IdPUserName** - 지정된 IdP 계정의 사용자 이름입니다.

  기본값은 NONE입니다.
- **PWD** 또는 **IdPPassword** - 지정된 IdP 계정의 암호입니다.

  기본값은 NONE입니다.
- **AADApplicationID** - Azure AD에 등록된 애플리케이션의 고유 ID입니다.

  기본값은 NONE입니다.
- **AADClientSecret** - 가져오기 토큰을 승인하는 데 사용되는 Azure AD의 등록된 애플리케이션과 연결된 클라이언트 보안 암호입니다.

  기본값은 NONE입니다.
- **AADTenant** - Azure AD 테넌트 ID입니다.

  기본값은 NONE입니다.
- **RoleARN** - 호출자가 수임하는 역할의 Amazon 리소스 이름(ARN)입니다.

  기본값은 NONE입니다.
- **IdPARN** - IdP 를 IAM 설명하는 의 SAML 공급자의 Amazon 리소스 이름(ARN)입니다.

  기본값은 NONE입니다.

AWS SDK (고급) 옵션

- **RequestTimeout** - 가 AWS SDK 제한 시간 전에 쿼리 요청을 기다리는 밀리초 단위의 시간입니다. 긍정이 아닌 값은 요청 제한 시간을 비활성화합니다.

  기본값은 3000입니다.

- **ConnectionTimeout** - 가 시간 초과 전에 열린 연결을 통해 데이터가 전송될 때까지 대기하는 AWS SDK 밀리초 단위 시간입니다. 값이 0이면 연결 제한 시간이 비활성화됩니다. 이 값은 음수가 아니어야 합니다.

  기본값은 1000입니다.
- **MaxRetryCountClient** - 에서 5xx 오류 코드를 사용하여 재시도 가능한 오류에 대한 최대 재시도 횟수입니다SDK. 값은 음수가 아니어야 합니다.

  기본값은 0입니다.
- **MaxConnections** - Timestream 서비스에 대해 동시에 열린 최대 허용 HTTP 연결 수입니다. 값은 양수여야 합니다.

  기본값은 25입니다.

ODBC 드라이버 로깅 옵션

- **LogLevel** – 드라이버 로깅의 로그 수준입니다. 다음 중 하나여야 합니다.
  - 0(OFF).
  - 1(ERROR).
  - 2(WARNING).
  - 3(INFO).
  - 4(DEBUG).

  기본값은 1 ()입니다ERROR.

  경고: 로깅 모드를 사용할 때 드라이버가 개인 정보를 DEBUG 기록할 수 있습니다.
- **LogOutput** – 로그 파일을 저장할 폴더입니다.

  기본값은 다음과 같습니다.
  - Windows: %USERPROFILE%, 또는 사용할 수 없는 경우 %HOMEDRIVE%%HOMEPATH%.
  - macOS 및 Linux: pw_dir 함수 getpwuid(getuid()) 반환 값의 필드 $HOME또는 사용할 수 없는 경우 .

SDK 로깅 옵션

AWS SDK 로그 수준은 드라이버 로그 수준에 대한 LiveAnalytics ODBC Timestream과는 별개입니다. 하나를 설정해도 다른 설정은 영향을 받지 않습니다.

SDK 로그 수준은 환경 변수 를 사용하여 설정됩니다TS_AWS_LOG_LEVEL. 유효한 값은 다음과 같습니다.

- OFF
- ERROR
- WARN
- INFO
- DEBUG
- TRACE
- FATAL

TS_AWS_LOG_LEVEL 이 설정되지 않으면 SDK 로그 수준이 기본값인 로 설정됩니다WARN.

프록시를 통한 연결

ODBC 드라이버는 프록시를 LiveAnalytics 통해 를 Amazon Timestream에 연결할 수 있도록 지원합니다. 이 기능을 사용하려면 프록시 설정에 따라 다음 환경 변수를 구성합니다.

- **TS_PROXY_HOST** - 프록시 호스트입니다.
- **TS_PROXY_PORT** - 프록시 포트 번호입니다.
- **TS_PROXY_SCHEME** - http 또는 의 프록시 체계입니다https.
- **TS_PROXY_USER** - 프록시 인증을 위한 사용자 이름입니다.
- **TS_PROXY_PASSWORD** - 프록시 인증을 위한 사용자 암호입니다.
- **TS_PROXY_SSL_CERT_PATH** - HTTPS 프록시에 연결하는 데 사용할 SSL 인증서 파일입니다.
- **TS_PROXY_SSL_CERT_TYPE** - 프록시 클라이언트 SSL 인증서의 유형입니다.
- **TS_PROXY_SSL_KEY_PATH** - HTTPS 프록시에 연결하는 데 사용할 프라이빗 키 파일입니다.
- **TS_PROXY_SSL_KEY_TYPE** - HTTPS 프록시에 연결하는 데 사용되는 프라이빗 키 파일의 유형입니다.
- **TS_PROXY_SSL_KEY_PASSWORD** - HTTPS 프록시에 연결하는 데 사용되는 프라이빗 키 파일에 대한 암호입니다.

드라이버용 Timestream LiveAnalytics ODBC의 연결 문자열 예제

IAM 보안 인증 정보로 ODBC 드라이버에 연결하는 예

```
Driver={Amazon Timestream ODBC Driver};Auth=IAM;AccessKeyId=(your access key
 ID);secretKey=(your secret key);SessionToken=(your session token);Region=us-east-2;
```

프로파일을 사용하여 ODBC 드라이버에 연결하는 예

```
Driver={Amazon Timestream ODBC Driver};ProfileName=(the profile name);region=us-west-2;
```

드라이버는 에 제공된 자격 증명을 사용하여 연결을 시도하거나~/.aws/credentials, 환경 변수 에 파일이 지정된 경우 해당 파일의 자격 증명을 AWS_SHARED_CREDENTIALS_FILE사용하여 연결을 시도합니다.

Okta를 사용하여 ODBC 드라이버에 연결하는 예

```
driver={Amazon Timestream ODBC Driver};auth=okta;region=us-west-2;idPHost=(your host at
 Okta);idPUsername=(your user name);idPPassword=(your password);OktaApplicationID=(your
 Okta AppId);roleARN=(your role ARN);idPARN=(your Idp ARN);
```

Azure Active Directory를 사용하여 ODBC 드라이버에 연결하는 예(AAD)

```
driver={Amazon Timestream ODBC Driver};auth=aad;region=us-west-2;idPUsername=(your
 user name);idPPassword=(your password);aadApplicationID=(your AAD
 AppId);aadClientSecret=(your AAD client secret);aadTenant=(your AAD
 tenant);roleARN=(your role ARN);idPARN=(your idP ARN);
```

지정된 엔드포인트와 로그 수준 2(WARNING)를 사용하여 ODBC 드라이버에 연결하는 예제

```
Driver={Amazon Timestream ODBC Driver};Auth=IAM;AccessKeyId=(your access
 key ID);secretKey=(your secret key);EndpointOverride=ingest.timestream.us-
west-2.amazonaws.com;Region=us-east-2;LogLevel=2;
```

ODBC 드라이버와의 연결 문제 해결

> **Note**
>
> 사용자 이름과 암호가 에 이미 지정된 경우 ODBC 드라이버 관리자가 요청할 때 다시 지정할 필요가 DSN없습니다.

메시지가 `01S02` 포함된 오류 코드는 연결 문자열 옵션이 연결 문자열에서 두 번 이상 전달될 때 `Re-writing (connection string option) (have you specified it several times?` 발생합니다. 옵션을 두 번 이상 지정하면 오류가 발생합니다. DSN 및 연결 문자열로 연결할 때 연결 옵션이 에 이미 지정된 경우 연결 문자열에 다시 지정하지 DSN마세요.

VPC 엔드포인트(AWS PrivateLink)

인터페이스 VPC 엔드포인트 를 생성 LiveAnalytics 하여 VPC 와 의 Amazon Timestream 간에 프라이빗 연결을 설정할 수 있습니다. 자세한 내용은 VPC 엔드포인트(AWS PrivateLink) 단원을 참조하십시오.

모범 사례

용 Amazon Timestream의 이점을 완전히 실현하려면 아래 설명된 모범 사례를 LiveAnalytics따르세요.

> **Note**
>
> 애플리케이션을 실행할 proof-of-concept 때는 에 대한 Timestream의 성능과 규모를 평가하면서 애플리케이션이 몇 개월 또는 몇 년에 걸쳐 누적되는 데이터의 양을 고려합니다 LiveAnalytics. 시간이 지남에 따라 데이터가 증가함에 따라 서버리스 아키텍처는 대량의 병렬 처리를 활용하여 더 큰 데이터 볼륨을 처리하고 애플리케이션의 요구 사항에 맞게 자동으로 확장할 수 있으므로 에 대한 Timestream의 성능 LiveAnalytics 은 대부분 변경되지 않습니다.

주제

- 데이터 모델링
- 보안
- 에 대한 Amazon Timestream 구성 LiveAnalytics
- 쓰기
- 쿼리
- 예약된 쿼리
- 클라이언트 애플리케이션 및 지원되는 통합
- 일반

데이터 모델링

용 Amazon Timestream LiveAnalytics 은 타임스탬프가 있는 데이터 시퀀스를 내보내는 애플리케이션 및 디바이스에서 시계열 데이터를 수집, 저장 및 분석하도록 설계되었습니다. 최적의 성능을 위해 에 대해 Timestream으로 전송되는 데이터는 시간적 특성을 LiveAnalytics 가져야 하며 시간은 데이터의 필수 구성 요소여야 합니다.

의 Timestream LiveAnalytics 은 애플리케이션의 요구 사항에 맞게 다양한 방식으로 데이터를 모델링할 수 있는 유연성을 제공합니다. 이 섹션에서는 이러한 패턴 중 몇 가지를 다루고 비용과 성능을 최적화하기 위한 지침을 제공합니다. 차원 및 측정치와 같은 LiveAnalytics 개념에 대한 주요 Timestream을 숙지합니다. 이 섹션에서는 다음에 대해 자세히 알아봅니다.

데이터를 저장하기 위해 단일 테이블을 생성할지 아니면 여러 테이블을 생성할지 결정할 때 다음을 고려하세요.

- 동일한 테이블에 넣을 데이터와 여러 테이블 및 데이터베이스에서 데이터를 분리하려는 경우의 데이터 비교.
- LiveAnalytics 다중 측정 레코드의 Timestream과 단일 측정 레코드 중에서 선택하는 방법과 애플리케이션이 여러 측정값을 동시에 추적하는 경우 다중 측정 레코드를 사용한 모델링의 이점.
- 차원 또는 측정값으로 모델링할 속성입니다.
- 측정 이름 속성을 효과적으로 사용하여 쿼리 지연 시간을 최적화하는 방법.

주제

- 단일 테이블과 여러 테이블 비교
- 다중 측정 레코드와 단일 측정 레코드 비교
- 차원 및 치수
- 다중 측정 레코드에서 측정 이름 사용
- 다중 측정 레코드 파티셔닝을 위한 권장 사항

단일 테이블과 여러 테이블 비교

애플리케이션에서 데이터를 모델링할 때 또 다른 중요한 측면은 데이터를 테이블과 데이터베이스로 모델링하는 방법입니다. 에 대한 Timestream의 데이터베이스 및 테이블 LiveAnalytics 은 액세스 제어, KMS 키 지정, 보존 기간 등을 위한 추상화입니다. 의 LiveAnalytics Timestream은 데이터를 자동으로 분할하며 애플리케이션의 수집, 스토리지 및 쿼리 로드와 요구 사항에 맞게 리소스를 확장하도록 설계되었습니다.

에 대한 Timestream의 테이블 LiveAnalytics 은 페타바이트의 저장된 데이터, 수십 기가바이트/초의 데이터 쓰기로 확장할 수 있으며 쿼리는 시간TBs당 수백 개를 처리할 수 있습니다. 에 대한 Timestream의 쿼리 LiveAnalytics 는 여러 테이블 및 데이터베이스에 걸쳐 조인 및 유니온을 제공하여 여러 테이블 및 데이터베이스의 데이터에 원활하게 액세스할 수 있습니다. 따라서 의 Timestream에서 데이터를 구성하는 방법을 결정할 때 데이터 또는 요청 볼륨의 규모는 일반적으로 주요 관심사가 아닙니다 LiveAnalytics. 다음은 동일한 테이블과 다른 테이블 또는 다른 데이터베이스의 테이블에 어떤 데이터를 공동 배치할지 결정할 때 고려해야 할 몇 가지 중요한 사항입니다.

- 데이터 보존 정책(메모리 스토어 보존, 마그네틱 스토어 보존 등)은 테이블의 세분성에서 지원됩니다. 따라서 서로 다른 보존 정책이 필요한 데이터는 서로 다른 테이블에 있어야 합니다.
- AWS KMS 데이터를 암호화하는 데 사용되는 키는 데이터베이스 수준에서 구성됩니다. 따라서 암호화 키 요구 사항이 다르기 때문에 데이터가 서로 다른 데이터베이스에 있어야 합니다.
- 의 Timestream은 테이블 및 데이터베이스의 세분화된 수준에서 리소스 기반 액세스 제어를 LiveAnalytics 지원합니다. 동일한 테이블과 다른 테이블에 쓸 데이터를 결정할 때는 액세스 제어 요구 사항을 고려하세요.
- 어떤 데이터가 어떤 테이블에 저장되는지 결정할 때는 차원, 측정 이름 및 다중 측정 속성 이름의 수에 대한 제한에 유의하세요.
- 쿼리 지연 시간과 쿼리 작성의 용이성은 데이터에 따라 달라지므로 데이터를 구성하는 방법을 결정할 때는 쿼리 워크로드와 액세스 패턴을 고려하세요.
  - 동일한 테이블에 자주 쿼리하는 데이터를 저장하는 경우 일반적으로 쿼리를 작성하는 방식이 쉬워지므로 조인, 유니온 또는 일반 테이블 표현식을 작성할 필요가 없습니다. 또한 일반적으로 쿼리 지연 시간이 줄어듭니다. 차원 및 측정 이름에 대한 표현을 사용하여 쿼리와 관련된 데이터를 필터링할 수 있습니다.

    예를 들어, 6개 대륙에 있는 디바이스의 데이터를 저장하는 경우를 고려해 보세요. 쿼리가 여러 대륙의 데이터에 자주 액세스하여 글로벌 집계 뷰를 가져오는 경우 이러한 대륙의 데이터를 동일한 테이블에 저장하면 쿼리를 더 쉽게 작성할 수 있습니다. 반면 다른 테이블에 데이터를 저장하는 경우 동일한 쿼리의 데이터를 결합할 수 있지만 테이블 간에 데이터를 결합하려면 쿼리를 작성해야 합니다.
  - 의 Timestream LiveAnalytics 은 데이터에 적응형 파티셔닝 및 인덱싱을 사용합니다. 따라서 쿼리는 쿼리와 관련된 데이터에 대해서만 요금이 부과됩니다. 예를 들어, 6개 대륙에 있는 백만 개의 디바이스에서 데이터를 저장하는 테이블이 있는 경우 쿼리에 양식 `WHERE device_id = 'abcdef'` 또는 의 예측자가 있는 경우 `WHERE continent = 'North America'`쿼리는 디바이스 또는 대륙의 데이터에 대해서만 요금이 부과됩니다.
  - 가능하면 측정 이름을 사용하여 동시에 내보내지 않거나 자주 쿼리되지 않는 동일한 테이블의 데이터를 분리한 다음 쿼리`WHERE measure_name = 'cpu'`에서와 같은 예측을 사용하는 경우 측

정 이점을 얻을 수 있을 뿐만 아니라 Timestream for는 쿼리 예측에 사용되는 측정 이름이 없는 파티션도 효과적으로 제거할 LiveAnalytics 수 있습니다. 이렇게 하면 쿼리 지연 시간이나 비용에 영향을 주지 않고 동일한 테이블에 다른 측정값 이름을 가진 관련 데이터를 저장할 수 있으며 데이터를 여러 테이블로 분산하지 않아도 됩니다. 측정 이름은 기본적으로 쿼리와 관련이 없는 데이터 및 정리 파티션을 분할하는 데 사용됩니다.

다중 측정 레코드와 단일 측정 레코드 비교

의 Timestream을 LiveAnalytics 사용하면 레코드당 여러 측정값(다중 측정) 또는 레코드당 단일 측정값(단일 측정)으로 데이터를 작성할 수 있습니다.

다중 측정 레코드

많은 사용 사례에서 추적 중인 디바이스 또는 애플리케이션이 동일한 타임스탬프에서 여러 지표 또는 이벤트를 생성할 수 있습니다. 이러한 경우 동일한 타임스탬프에서 방출된 모든 지표를 동일한 다중 측정 레코드에 저장할 수 있습니다. 즉, 동일한 다중 측정 레코드에 저장된 모든 측정값은 동일한 데이터 행에 서로 다른 열로 표시됩니다.

예를 들어 애플리케이션이 cpu, 메모리, disk_iops와 같은 지표를 동시에 측정된 디바이스에서 즉시 내보내고 있다고 가정해 보겠습니다. 다음은 인스턴스에서 동시에 내보내는 여러 지표가 동일한 행에 저장되는 테이블의 예입니다. 두 호스트가 매초마다 지표를 내보내는 것을 볼 수 있습니다.

Hostname	measure_name	Time	cpu	메모리	disk_iops
host-24Gju	지표	2021-12-01 19:00:00	35	54.9	38.2
host-24Gju	지표	2021-12-01 19:00:01	36	58	39
host-28Gju	지표	2021-12-01 19:00:00	15	55	92
host-28Gju	지표	2021-12-01 19:00:01	16	50	40

단일 측정 레코드

단일 측정 레코드는 디바이스가 서로 다른 기간에 서로 다른 지표를 내보내거나 metrics/events at different time periods (for instance, when a device's reading/state 변경 사항을 내보내는 사용자 지정 처리 로직을 사용하는 경우에 적합합니다.) 모든 측정값에는 고유한 타임스탬프가 있기 때문에 측정값을 의 Timestream에 있는 자체 레코드에 저장할 수 있습니다 LiveAnalytics. 예를 들어, 토양 온도와 습기를 추적하는 IoT 센서는 이전에 보고된 항목으로부터의 변화를 감지할 때만 레코드를 내보냅니다. 다음 예제에서는 단일 측정 레코드를 사용하여 내보내는 이러한 데이터의 예를 제공합니다.

device_id	measure_name	Time	measure_value::double	measure_value::bigint
sensor-sea478	temperature	2021-12-01 19:22:32	35	NULL
sensor-sea478	temperature	2021-12-01 18:07:51	36	NULL
sensor-sea478	수분	2021-12-01 19:05:30	NULL	21
sensor-sea478	수분	2021-12-01 19:00:01	NULL	23

단일 측정 및 다중 측정 레코드 비교

의 Timestream LiveAnalytics 은 애플리케이션의 요구 사항 및 특성에 따라 데이터를 단일 측정 또는 다중 측정 레코드로 모델링할 수 있는 유연성을 제공합니다. 애플리케이션 요구 사항이 필요한 경우 단일 테이블에서 단일 측정 및 다중 측정 레코드를 모두 저장할 수 있습니다. 일반적으로 애플리케이션이 여러 측정값/이벤트를 동시에 즉시 내보내는 경우 성능 있는 데이터 액세스 및 비용 효율적인 데이터 스토리지에는 일반적으로 데이터를 다중 측정 레코드로 모델링하는 것이 좋습니다.

예를 들어 수십만 개의 서버에서 DevOps 사용 사례 추적 지표와 이벤트를 고려하면 각 서버는 주기적으로 20개의 지표와 5개의 이벤트를 내보내고, 이벤트와 지표는 동시에 즉시 내보내집니다. 이 데이터는 단일 측정 레코드 또는 다중 측정 레코드를 사용하여 모델링할 수 있습니다(결과 스키마는 오픈 소스 데이터 생성기 참조). 이 사용 사례의 경우 단일 측정 레코드와 비교하여 다중 측정 레코드를 사용하여 데이터를 모델링하면 다음과 같은 결과가 발생합니다.

- 섭취 측정 - 다중 측정 레코드는 약 40% 더 낮은 수집 바이트를 기록합니다.

- 수집 일괄 처리 - 다중 측정 레코드는 더 많은 데이터 일괄을 전송하므로 클라이언트가 수집을 처리하는 CPU 데 더 적은 스레드와 더 적은 스레드가 필요함을 의미합니다.
- 스토리지 측정 - 다중 측정 레코드를 사용하면 스토리지가 약 8X 낮아지므로 메모리 및 마그네틱 스토어 모두에서 스토리지 비용을 크게 절감할 수 있습니다.
- 쿼리 지연 시간 - 다중 측정 레코드는 단일 측정 레코드와 비교할 때 대부분의 쿼리 유형에 대해 쿼리 지연 시간을 줄입니다.
- 쿼리 측정 바이트 - 10MB 미만의 데이터를 스캔하는 쿼리의 경우 단일 측정 레코드와 다중 측정 레코드가 모두 비슷합니다. 단일 측정값에 액세스하고 > 10MB 데이터를 스캔하는 쿼리의 경우 단일 측정값 레코드는 일반적으로 더 낮은 바이트를 측정하게 됩니다. 3개 이상의 측정값을 참조하는 쿼리의 경우 다중 측정 레코드로 인해 측정되는 바이트 수가 줄어듭니다.
- 다중 측정 쿼리 표현의 용이성 - 쿼리가 다중 측정을 참조할 때 다중 측정 레코드로 데이터를 모델링하면 더 작은 쿼리를 더 쉽게 작성할 수 있습니다.

이전 요인은 추적 중인 지표 수, 데이터 차원 수 등에 따라 달라집니다. 앞의 예에서는 한 가지 예에 대한 몇 가지 구체적인 데이터를 제공하지만 많은 애플리케이션 시나리오와 사용 사례에서 애플리케이션이 동시에 여러 측정값을 내보내는 경우 데이터를 다중 측정 레코드로 저장하는 것이 더 효과적입니다. 또한 다중 측정 레코드는 데이터 형식의 유연성을 제공하고 여러 다른 값을 컨텍스트로 저장합니다(예: 요청 저장 IDs및 나중에 설명하는 추가 타임스탬프).

다중 측정 레코드는 단일 측정 레코드에 대한 이전 예제와 같은 희소 측정값을 모델링할 수도 있습니다. measure_name을 사용하여 측정값의 이름을 저장하고 value_double과 같은 일반 다중 측정 속성 이름, value_bigint를 사용하여 DOUBLE 측정값을 저장BIGINT, value_timestamp를 사용하여 추가 TIMESTAMP 값 저장 등을 수행할 수 있습니다.

차원 및 치수

의 Timestream 테이블을 LiveAnalytics 사용하면 차원(저장 중인 디바이스/데이터의 속성 식별) 및 측정값(추적 중인 지표/값)을 저장할 수 있습니다. 자세한 내용은 Timestream의 LiveAnalytics 개념을 참조하세요. 용 Timestream에서 애플리케이션을 모델링할 때 데이터를 차원 및 측정값으로 매핑하는 LiveAnalytics방법은 수집 및 쿼리 지연 시간에 영향을 미칩니다. 다음은 데이터를 사용 사례에 적용할 수 있는 차원 및 측정값으로 모델링하는 방법에 대한 지침입니다.

차원 선택

시계열 데이터를 전송하는 소스를 식별하는 데이터는 시간이 지남에 따라 변경되지 않는 속성인 차원에 자연스럽게 적합합니다. 예를 들어 지표를 내보내는 서버가 있는 경우 호스트 이름, 리전, 랙, 가용

영역과 같이 서버를 식별하는 속성이 차원의 후보가 됩니다. 마찬가지로 시계열 데이터를 보고하는 센서가 여러 개 있는 IoT 디바이스의 경우 디바이스 ID, 센서 ID 등이 차원에 적합합니다.

데이터를 다중 측정 레코드로 작성하는 경우 테이블에서 SELECT 문을 실행DESCRIBE하거나 실행할 때 차원 및 다중 측정 속성이 테이블의 열로 표시됩니다. 따라서 쿼리를 작성할 때 동일한 쿼리에서 차원과 측정값을 자유롭게 사용할 수 있습니다. 그러나 데이터를 수집하기 위해 쓰기 레코드를 구성할 때 어떤 속성을 차원으로 지정하고 어떤 속성을 값을 측정하는지 선택할 때 다음 사항에 유의하세요.

- 차원 이름, 값, 측정값 이름 및 타임스탬프는 시계열 데이터를 고유하게 식별합니다. 의 Timestream LiveAnalytics 은 이 고유 식별자를 사용하여 데이터를 자동으로 중복 제거합니다. 즉, Timestream for 가 차원 이름, 차원 값, 측정 이름 및 타임스탬프 값이 동일한 두 개의 데이터 포인트를 LiveAnalytics 수신하는 경우 값이 버전 번호가 동일한 경우 중복 제거를 위한 LiveAnalytics Timestream입니다. 새 쓰기 요청의 버전이 에 대한 Timestream에 이미 있는 데이터보다 낮으면 LiveAnalytics쓰기 요청이 거부됩니다. 새 쓰기 요청의 버전이 더 높으면 새 값이 이전 값을 덮어씁니다. 따라서 차원 값을 선택하는 방법은 이 중복 제거 동작에 영향을 미칩니다.
- 차원 이름 및 값을 업데이트할 수 없으며 측정값을 업데이트할 수 있습니다. 따라서 업데이트가 필요할 수 있는 모든 데이터는 측정값으로 더 잘 모델링됩니다. 예를 들어, 색상이 변경될 수 있는 기계가 공장 바닥에 있는 경우 색상을 측정 값으로 모델링할 수 있습니다. 단, 색상도 중복 제거에 필요한 식별 속성으로 사용하려는 경우는 예외입니다. 즉, 측정값을 사용하여 시간이 지남에 따라 천천히 변화하는 속성을 저장할 수 있습니다.

에 대한 Timestream의 테이블 LiveAnalytics 은 차원 이름과 값의 고유한 조합 수를 제한하지 않습니다. 예를 들어 테이블에 수십억 개의 고유한 값 조합이 저장될 수 있습니다. 그러나 다음 예제에서 볼 수 있듯이 차원과 측정값을 신중하게 선택하면 특히 쿼리의 경우 요청 지연 시간을 크게 최적화할 수 있습니다.

차원IDs이 고유함

애플리케이션 시나리오에서 모든 데이터 포인트(예: 요청 ID, 트랜잭션 ID 또는 상관 관계 ID)에 대한 고유 식별자를 저장해야 하는 경우 ID 속성을 측정값으로 모델링하면 쿼리 지연 시간이 크게 개선됩니다. 다중 측정 레코드로 데이터를 모델링할 때 ID는 다른 차원 및 시계열 데이터와 동일한 행에 표시되므로 쿼리가 계속해서 효과적으로 사용할 수 있습니다. 예를 들어, 서버에서 방출되는 모든 데이터 포인트에 고유한 요청 ID 속성이 있는 DevOps 사용 사례를 고려하면, 요청 ID를 측정값으로 모델링하면 고유한 요청 ID를 차원으로 모델링하는 대신 서로 다른 쿼리 유형에서 쿼리 지연 시간이 최대 4배 줄어듭니다.

모든 데이터 포인트에 대해 완전히 고유하지는 않지만 수십만 또는 수백만 개의 고유한 값을 가진 속성에 유사한 비유를 사용할 수 있습니다. 이러한 속성을 차원 또는 측정값으로 모델링할 수 있습니다. 앞

에서 설명한 대로 쓰기 경로에서 중복 제거에 값이 필요한 경우 또는 쿼리에서 값을 조건(예: 애플리케이션이 수백만 개의 디바이스를 추적device_id = 'abcde'하는 경우와 같은 속성 값에 대해 균등한 조건의 WHERE 절)으로 사용하는 경우 차원으로 모델링해야 합니다.

다중 측정 레코드를 통한 데이터 유형의 풍부성

다중 측정 레코드는 데이터를 효과적으로 모델링할 수 있는 유연성을 제공합니다. 다중 측정 레코드에 저장하는 데이터는 차원과 유사한 테이블의 열로 표시되므로 차원 및 측정값에 대해 쿼리하기가 쉽습니다. 앞서 설명한 예제에서 이러한 패턴 중 일부를 보았습니다. 아래에서 다중 측정 레코드를 효과적으로 사용하여 애플리케이션의 사용 사례를 충족하는 추가 패턴을 확인할 수 있습니다.

다중 측정 레코드는 데이터 유형 DOUBLE, , BIGINT, 및 VARCHAR의 속성을 지원합니다BOOLEANTIMESTAMP. 따라서 다양한 유형의 속성에 자연스럽게 맞습니다.

- 위치 정보 : 예를 들어 위치(위도 및 경도로 표현)를 추적하려는 경우 다중 측정 속성으로 모델링하면 특히 위도 및 경도에 대한 기준이 있는 경우 VARCHAR 차원으로 저장하는 것보다 쿼리 지연 시간이 줄어듭니다.
- 레코드의 여러 타임스탬프: 애플리케이션 시나리오에서 시계열 레코드에 대해 여러 타임스탬프를 추적해야 하는 경우 다중 측정 레코드에서 추가 속성으로 모델링할 수 있습니다. 이 패턴은 향후 타임스탬프 또는 과거 타임스탬프와 함께 데이터를 저장하는 데 사용할 수 있습니다. 모든 레코드는 여전히 시간 열의 타임스탬프를 사용하여 레코드를 분할, 인덱싱 및 고유하게 식별합니다.

특히 쿼리에 예측자가 있는 숫자 데이터 또는 타임스탬프가 있는 경우 이러한 속성을 차원이 아닌 다중 측정 속성으로 모델링하면 쿼리 지연 시간이 줄어듭니다. 이는 다중 측정 레코드에서 지원되는 풍부한 데이터 유형을 사용하여 이러한 데이터를 모델링할 때 이러한 데이터를 차원으로 모델링한 경우 에서 다른 데이터 유형으로 값을 캐스팅하는 대신 기본 데이터 유형을 사용하여 예측을 표현VARCHAR할 수 있기 때문입니다.

다중 측정 레코드에서 측정 이름 사용

에 대한 Timestream의 테이블은 측정 이름이라는 특수 속성(또는 열)을 LiveAnalytics 지원합니다. Timestream for 에 기록하는 모든 레코드에 대해 이 속성의 값을 지정합니다 LiveAnalytics. 단일 측정 레코드의 경우 지표의 이름(예: cpu, 서버 지표의 메모리 또는 온도, 센서 지표의 압력)을 사용하는 것은 자연스러운 일입니다. 다중 측정 레코드를 사용하는 경우 다중 측정 레코드의 속성 이름이 지정되고(이 이름은 테이블의 열 이름이 됨), cpu, 메모리 또는 온도이므로 압력은 다중 측정 속성 이름이 될 수 있습니다. 따라서 자연스러운 질문은 측정값 이름을 효과적으로 사용하는 방법입니다.

의 Timestream LiveAnalytics 은 측정 이름 속성의 값을 사용하여 데이터를 분할하고 인덱싱합니다. 따라서 테이블의 측정값 이름이 여러 개이고 쿼리가 해당 값을 쿼리 예측으로 사용하는 경우 의

Timestream은 사용자 지정 파티셔닝 및 인덱싱을 사용하여 쿼리와 관련이 없는 데이터를 정리할 LiveAnalytics 수 있습니다. 예를 들어 테이블에 cpu 및 메모리 측정 이름이 있고 쿼리에 이라는 어휘가 있는 경우 `WHERE measure_name = 'cpu'`Timestream for 는 이 예제에서 측정 이름 메모리가 있는 행과 같이 쿼리와 관련이 없는 측정 이름에 대한 데이터를 효과적으로 정리할 LiveAnalytics 수 있습니다. 이 정리는 다중 측정 레코드와 함께 측정 이름을 사용하는 경우에도 적용됩니다. 측정 이름 속성을 테이블의 파티셔닝 속성으로 효과적으로 사용할 수 있습니다. 측정 이름과 차원 이름 및 값, 시간은 LiveAnalytics 테이블의 Timestream에서 데이터를 분할하는 데 사용됩니다. LiveAnalytics 테이블에 대한 Timestream에서 허용되는 고유 측정 이름 수의 제한에 유의하세요. 또한 측정값 이름은 측정값 데이터 유형과도 연결됩니다. 예를 들어 단일 측정값 이름은 하나의 측정값 유형과만 연결할 수 있습니다. 이 유형은 `DOUBLE`, , `BIGINT`, `BOOLEAN``VARCHAR`, 중 하나일 수 있습니다`MULTI`. 측정 이름으로 저장된 다중 측정 레코드의 데이터 유형은 입니다`MULTI`. 단일 다중 측정 레코드는 다양한 데이터 유형(`DOUBLE`, , `BIGINT`, `VARCHAR` `BOOLEAN`및 `TIMESTAMP`)으로 여러 지표를 저장할 수 있으므로 다중 측정 레코드에서 다양한 유형의 데이터를 연결할 수 있습니다.

다음 섹션에서는 측정값 이름 속성을 효과적으로 사용하여 동일한 테이블에서 다양한 유형의 데이터를 그룹화하는 방법에 대한 몇 가지 예를 설명합니다.

품질 및 가치를 보고하는 IoT 센서

IoT 센서의 애플리케이션 모니터링 데이터가 있다고 가정해 보겠습니다. 각 센서는 온도, 압력과 같은 다양한 측정값을 추적합니다. 실제 값 외에도 센서는 측정값의 품질도 보고합니다. 이는 측정값의 정확도와 측정 단위의 측정치입니다. 품질, 단위 및 값은 함께 방출되므로 아래 예제 데이터에 표시된 것처럼 다중 측정 레코드로 모델링할 수 있습니다. 여기서 device_id는 차원, 품질, 값 및 단위는 다중 측정 속성입니다.

device_id	measure_name	Time	화질	값	단위
sensor-sea478	temperature	2021-12-01 19:22:32	92	35	c
sensor-sea478	temperature	2021-12-01 18:07:51	93	34	c
sensor-sea478	pressure	2021-12-01 19:05:30	98	31	psi
sensor-sea478	pressure	2021-12-01 19:00:01	24	132	psi

이 접근 방식을 사용하면 측정값 이름 값을 사용하여 데이터 분할 및 정리와 함께 다중 측정 레코드의 이점을 결합할 수 있습니다. 쿼리가 온도와 같은 단일 측정값을 참조하는 경우 쿼리에 측정값 이름 기호를 포함할 수 있습니다. 다음은 품질이 90을 초과하는 측정을 위해 단위를 투영하는 이러한 쿼리의 예입니다.

```
SELECT device_id, time, value AS temperature, unit
FROM db.table
WHERE time > ago(1h)
    AND measure_name = 'temperature'
    AND quality > 90
```

쿼리에 measure_name 예측을 사용하면 LiveAnalytics 가 쿼리와 관련이 없는 파티션 및 데이터를 효과적으로 정리하여 쿼리 지연 시간을 개선할 수 있습니다.

또한 모든 지표가 동일한 타임스탬프로 방출되고/되거나 동일한 쿼리에서 여러 지표가 함께 쿼리되는 경우 모든 지표를 동일한 다중 측정 레코드에 저장할 수도 있습니다. 예를 들어 temperature_quality, temperature_value, temperature_unit, pressure_quality, pressure_value, pressure_unit 등의 속성을 사용하여 다중 측정 레코드로 구성할 수 있습니다. 단일 측정 대 다중 측정 레코드를 사용하여 데이터를 모델링하는 방법에 대해 앞서 설명한 많은 요점이 데이터 모델링 방법에 대한 결정에 적용됩니다. 쿼리 액세스 패턴과 데이터 생성 방법을 고려하여 비용, 수집 및 쿼리 지연 시간, 쿼리 작성 편의성을 최적화하는 모델을 선택합니다.

동일한 테이블의 다양한 유형의 지표

다중 측정 레코드와 측정 이름 값을 결합할 수 있는 또 다른 사용 사례는 동일한 디바이스에서 독립적으로 내보내는 다양한 유형의 데이터를 모델링하는 것입니다. DevOps 모니터링 사용 사례 서버가 두 가지 유형의 데이터, 즉 정기적으로 방출되는 지표와 불규칙한 이벤트를 내보내고 있음을 고려합니다. 이 접근 방식의 예로는 데이터 생성기 모델링 DevOps 사용 사례 에서 설명하는 스키마가 있습니다. 이 경우 다른 측정 이름을 사용하여 동일한 서버에서 내보내는 다양한 유형의 데이터를 동일한 테이블에 저장할 수 있습니다. 예를 들어 동시에 생성되는 모든 지표는 측정 이름 지표와 함께 저장됩니다. 지표에서 다른 시간에 내보내는 모든 이벤트는 측정 이름 이벤트와 함께 저장됩니다. 테이블의 측정 스키마(예: SHOW MEASURES 쿼리 출력)는 다음과 같습니다.

measure_name	data_type	차원
이벤트	다중	[{'data_type':'varchar','dimension_name':'availability_zone'},{'data_type':'varch

measure_name	data_type	차원
		ar','dimension_name'',{'dat a_type':'varchar','dimensio n_name':'instance_name'}, {'data_type':'varchar','dimen sion_name',{'data_type':'va rchar','dimension_name'':' jdk_version'},{'data_type'': 'varchar','dimension_name': 'cell'},{'data_type':'varch ar','varchar'','dimension_n ame"""":'
지표	다중	[{'data_type':'varchar','di mension_name':'availability _zone'},{'data_type':'varch ar','dimension_name'',{'dat a_type':'varchar','dimensio n_name':'instance_name'},{' data_type','dimension_name' ,''varchar','dimension_vers ion'},{'data_type':'varchar ','dimension_name':'cell'}, {'data_type':'varchar','dim ension_name'':'region'},'da ta_type':'varchar','dimensi on_name'""

이 경우 이벤트와 지표의 차원 세트도 서로 다르며, 여기서 이벤트의 차원은 jdk_version과 process_name이고 지표의 차원은 instance_type과 os_version입니다.

다른 측정 이름을 사용하면 와 같은 표현으로 쿼리`WHERE measure_name = 'metrics'`를 작성하여 지표만 가져올 수 있습니다. 또한 동일한 테이블의 동일한 인스턴스에서 내보내는 모든 데이터를 보유한다는 것은 instance_name predicate로 더 간단한 쿼리를 작성하여 해당 인스턴스의 모든 데이터를 가져올 수 있음을 의미합니다. 예를 들어 measure_name predicate가 `WHERE instance_name = ‘instance-1234’` 없는 양식의 predicate는 특정 서버 인스턴스에 대한 모든 데이터를 반환합니다.

다중 측정 레코드 파티셔닝을 위한 권장 사항

> **Important**
>
> 이 섹션은 더 이상 사용되지 않습니다!
> 이러한 권장 사항은 최신 버전이 아닙니다. 이제 고객 정의 파티션 키 를 사용하여 파티셔닝을 더 잘 제어할 수 있습니다.

시계열 에코시스템에서 방대한 양의 데이터를 수집 및 저장해야 하는 워크로드 수가 증가하고 있으며, 동시에 카디널리티가 높은 차원 값 집합으로 데이터에 액세스할 때 지연 시간이 짧은 쿼리 응답이 필요하다는 것을 알았습니다.

이러한 특성으로 인해 이 섹션의 권장 사항은 다음과 같은 고객 워크로드에 유용합니다.

- 다중 측정 레코드를 채택했거나 채택하려고 합니다.
- 장기간 저장될 대량의 데이터가 시스템에 유입될 것으로 예상됩니다.
- 기본 액세스(쿼리) 패턴에 짧은 지연 시간 응답 시간이 필요합니다.
- 가장 중요한 쿼리 패턴에는 조건의 필터링 조건이 포함됩니다. 이 필터링 조건은 높은 카디널리티 차원을 기반으로 합니다. 예를 들어 , UserId, ServerID DeviceId, 호스트 이름 등을 기준으로 이벤트 또는 집계를 고려합니다.

이러한 경우 엔진은 다중 측정 이름을 사용하여 데이터를 분할하고 단일 값을 가지면 얻을 수 있는 파티션 이점이 제한되므로 모든 다중 측정 측정값에 대한 단일 이름이 도움이 되지 않습니다. 이러한 레코드의 파티셔닝은 주로 두 가지 차원을 기반으로 합니다. 시간은 x축에 있고 차원 이름의 해시와 y축`measure_name`에 있다고 가정해 보겠습니다. `measure_name` 이러한 경우 는 파티셔닝 키와 거의 비슷하게 작동합니다.

권장 사항은 다음과 같습니다.

- 앞서 언급한 것과 같은 사용 사례에 대한 데이터를 모델링할 때는 기본 쿼리 액세스 패턴의 직접 파생물`measure_name`인 를 사용합니다. 예:
  - 사용 사례에서는 최종 사용자 관점에서 애플리케이션 성능 및 QoE를 추적해야 합니다. 이는 단일 서버 또는 IoT 디바이스에 대한 측정값을 추적하는 것일 수도 있습니다.
  - 를 기준으로 쿼리하고 필터링 UserId하는 경우 수집 시간에 에 연결할 수 `measure_name` 있는 가장 좋은 방법을 찾아야 합니다 UserId.

  - 다중 측정 테이블에는 8,192개의 서로 다른 측정 이름만 포함될 수 있으므로 어떤 공식을 채택하든 8,192개의 서로 다른 값을 더 많이 생성해서는 안 됩니다.
- 문자열 값에 성공적으로 적용한 한 가지 접근 방식은 문자열 값에 해싱 알고리즘을 적용하는 것입니다. 그런 다음 해시 결과 및 8192의 절대 값을 사용하여 모듈로 작업을 수행합니다.

```
measure_name = getMeasureName(UserId)
int getMeasureName(value) {
    hash_value =  abs(hash(value))
    return hash_value % 8192
}
```

- 또한 값이 -8192~8192 범위일 가능성을 없애`abs()`기 위해 기호를 제거했습니다. 이는 모듈로 작업 전에 수행해야 합니다.
- 이 메서드를 사용하면 파티셔닝되지 않은 데이터 모델에서 쿼리를 실행하는 데 걸리는 시간의 일부만에 쿼리를 실행할 수 있습니다.
- 데이터를 쿼리할 때 measure_name의 새로 파생된 값을 사용하는 예측에 필터링 조건을 포함해야 합니다. 예:
  -
```
SELECT * FROM your_database.your_table
WHERE host_name = 'Host-1235' time BETWEEN '2022-09-01'
    AND '2022-09-18'
    AND measure_name = (SELECT
 cast(abs(from_big_endian_64(xxhash64(CAST('HOST-1235' AS varbinary))))%8192 AS
 varchar))
```

  - 이렇게 하면 스캔된 총 파티션 수를 최소화하여 시간이 지남에 따라 더 빠른 쿼리로 변환되는 데이터를 얻을 수 있습니다.

이 파티션 스키마의 이점을 얻으려면 클라이언트 측에서 해시를 계산하여 LiveAnalytics 에 대한 Timestream에 쿼리 엔진의 정적 값으로 전달해야 합니다. 앞의 예에서는 필요한 경우 엔진에서 생성된 해시를 확인할 수 있는지 검증하는 방법을 제공합니다.

시간	host_name	location	server_type, 서버_유형	cpu_usage	가용_기억	cpu_temp
2022-09-07 21:48:44 .000 0	호스트-1235	us-east1	5.8xl	55	16.2	78
R2022-09-07 21:48:44 .000 0	호스트-3587	us-west1	5.8xl	62	18.1	81
2022-09-07 21:48:45. 000000000	호스트-258743	eu-central	5.8xl	88	9.4	91
2022-09-07 21:48:45 .000 0	호스트-35654	us-east2	5.8xl	29	24	54
R2022-09-07 21:48:45 .000 0	호스트-254	us-west1	5.8xl	44	32	48

권장 사항에 `measure_name` 따라 연결된 를 생성하려면 수집 패턴에 따라 두 가지 경로가 있습니다.

1. 과거 데이터의 배치 수집의 경우 - 배치 프로세스에 자체 코드를 사용할 경우 쓰기 코드에 변환을 추가할 수 있습니다.

   이전 예제를 기반으로 를 구축합니다.

```
        List<String> hosts = new ArrayList<>();

        hosts.add("host-1235");
        hosts.add("host-3587");
        hosts.add("host-258743");
```

```
        hosts.add("host-35654");
        hosts.add("host-254");

        for (String h: hosts){
            ByteBuffer buf2 = ByteBuffer.wrap(h.getBytes());
            partition = abs(hasher.hash(buf2, 0L)) % 8192;
            System.out.println(h + " - " + partition);

        }
```

출력

```
host-1235 - 6445
host-3587 - 6399
host-258743 - 640
host-35654 - 2093
host-254 - 7051
```

결과 데이터 세트

시간	host_name	location	measure_name	server_type, 서버_유형	cpu_usage	가용_기억	cpu_temp
2022-09-07 21:48:44 .00	호스트-1235	us-east1	6445	5.8xl	55	16.2	78
R2022-09-07 21:48:44 .00	호스트-3587	us-west1	6399	5.8xl	62	18.1	81
2022-09-07 21:48:45. 000000000	호스트-258743	eu-central	640	5.8xl	88	9.4	91

시간	host_name	location	measure_name	server_type, 서버_유형	cpu_usage	가용_기억	cpu_temp
2022-09-07 21:48:45 .00	호스트-35654	us-east2	2093	5.8xl	29	24	54
R2022-09-07 21:48:45 .00	호스트-254	us-west1	7051	5.8xl	44	32	48

2. 실시간 수집의 경우 - 데이터가 들어오는 동안 전송 `measure_name` 중인 를 생성해야 합니다.

두 경우 모두 양쪽 끝(수집 및 쿼리)에서 해시 생성 알고리즘을 테스트하여 동일한 결과를 얻고 있는지 확인하는 것이 좋습니다.

다음은 를 기반으로 해시 값을 생성하는 몇 가지 코드 예제입니다`host_name`.

Example Python

```
>>> import xxhash
>>> from bitstring import BitArray
>>> b=xxhash.xxh64('HOST-ID-1235').digest()
>>> BitArray(b).int % 8192
### 3195
```

Example Go

```
package main

import (
    "bytes"
    "fmt"
    "github.com/cespare/xxhash"
)

func main() {
```

```
    buf := bytes.NewBufferString("HOST-ID-1235")
    x := xxhash.New()
    x.Write(buf.Bytes())
    // convert unsigned integer to signed integer before taking mod
    fmt.Printf("%f\n", abs(int64(x.Sum64())) % 8192)
}

func abs(x int64) int64 {
    if (x < 0) {
        return -x
    }
    return x
}
```

Example Java

```
import java.nio.ByteBuffer;

import net.jpountz.xxhash.XXHash64;

public class test {
    public static void main(String[] args) {
        XXHash64 hasher = net.jpountz.xxhash.XXHashFactory.fastestInstance().hash64();

        String host = "HOST-ID-1235";
        ByteBuffer buf = ByteBuffer.wrap(host.getBytes());

        Long result = Math.abs(hasher.hash(buf, 0L));
        Long partition = result % 8192;

        System.out.println(result);
        System.out.println(partition);
    }
}
```

Example Maven의 종속성

```
        <dependency>
            <groupId>net.jpountz.lz4</groupId>
            <artifactId>lz4</artifactId>
            <version>1.3.0</version>
        </dependency>
```

보안

- 용 Timestream에 지속적으로 액세스하려면 암호화 키가 보호되어 있고 취소되거나 액세스할 수 없게 되지 않았는지 LiveAnalytics확인하세요.
- 의 API 액세스 로그를 모니터링합니다 AWS CloudTrail. 권한이 없는 사용자로부터 비정상적인 액세스 패턴을 감사하고 취소합니다.
- 에 설명된 추가 지침을 따릅니다용 Amazon Timestream의 보안 모범 사례 LiveAnalytics.

에 대한 Amazon Timestream 구성 LiveAnalytics

데이터 처리, 스토리지, 쿼리 성능 및 비용 요구 사항과 일치하도록 메모리 스토어 및 마그네틱 스토어의 데이터 보존 기간을 구성합니다.

- 지연 도착 데이터 처리에 대한 애플리케이션의 요구 사항에 맞게 메모리 스토어의 데이터 보존을 설정합니다. 지연 도착 데이터는 타임스탬프가 현재 시간보다 이전인 수신 데이터입니다. 용 Timestream으로 데이터를 전송하기 전에 일정 기간 동안 이벤트를 배치하는 리소스 LiveAnalytics와 간헐적으로 온라인 상태인 IoT 센서와 같은 간헐적 연결이 있는 리소스에서 방출됩니다.
- 지연 도착 데이터가 메모리 스토어 보존보다 빠른 타임스탬프와 함께 가끔 도착할 것으로 예상되는 경우 테이블에 대해 마그네틱 스토어 쓰기를 활성화해야 합니다. 테이블에 MagneticStoreWritesProperties 대해 EnableMagneticStoreWrites 를 설정하면 테이블은 메모리 스토어 보존 기간보다 이전이지만 마그네틱 스토어 보존 기간 내에 타임스탬프가 있는 데이터를 수락합니다.
- 쿼리 유형, 빈도, 시간 범위 및 성능 요구 사항 LiveAnalytics 과 같이 Timestream에서 실행하려는 쿼리의 특성을 고려합니다. 메모리 스토어와 마그네틱 스토어가 다양한 시나리오에 최적화되어 있기 때문입니다. 메모리 스토어는 에 대해 Timestream으로 전송된 소량의 최근 데이터를 처리하는 빠른 point-in-time 쿼리에 최적화되어 있습니다 LiveAnalytics. 마그네틱 스토어는 에 대해 Timestream으로 전송된 중간 볼륨에서 대량의 데이터를 처리하는 빠른 분석 쿼리에 최적화되어 있습니다 LiveAnalytics.
- 또한 데이터 보존 기간은 시스템의 비용 요구 사항에 영향을 받아야 합니다.

  예를 들어 애플리케이션의 지연 도착 데이터 임계값이 2시간이고 애플리케이션이 1일치, 1주치 또는 1개월치 데이터를 처리하는 많은 쿼리를 전송하는 시나리오를 고려해 보겠습니다. 이 경우 메모리 스토어의 보존 기간(2~3시간)을 더 적게 구성하고 자성 스토어가 빠른 분석 쿼리에 최적화되어 있는 경우 더 많은 데이터가 자성 스토어로 흐르도록 허용할 수 있습니다.

메모리 스토어와 기존 테이블의 마그네틱 스토어의 데이터 보존 기간을 늘리거나 줄일 때 미치는 영향을 이해합니다.

- 메모리 스토어의 보존 기간을 줄이면 데이터가 메모리 스토어에서 마그네틱 스토어로 이동되고 이 데이터 전송은 영구적입니다. 에 대한 Timestream LiveAnalytics 은 마그네틱 스토어에서 데이터를 검색하여 메모리 스토어를 채우지 않습니다. 마그네틱 스토어의 보존 기간을 줄이면 데이터가 시스템에서 삭제되고 데이터 삭제는 영구적입니다.
- 메모리 스토어 또는 마그네틱 스토어의 보존 기간을 늘리면 LiveAnalytics 해당 시점부터 Timestream으로 전송되는 데이터에 변경 사항이 적용됩니다. 에 대한 Timestream LiveAnalytics 은 마그네틱 스토어에서 데이터를 검색하여 메모리 스토어를 채우지 않습니다. 예를 들어 메모리 스토어의 보존 기간이 처음에 2시간으로 설정된 다음 24시간으로 증가한 경우 메모리 스토어에 24시간 분량의 데이터가 포함되려면 22시간이 걸립니다.

쓰기

- 수신 데이터의 타임스탬프가 메모리 스토어에 대해 구성된 데이터 보존보다 빠르지 않고 에 정의된 향후 수집 기간보다 늦지 않은지 확인합니다[할당량](). 타임스탬프가 이러한 경계를 벗어나는 데이터를 전송하면 테이블에 대해 마그네틱 스토어 쓰기를 활성화하지 LiveAnalytics 않는 한 에 대해 Timestream에서 데이터가 거부됩니다. 마그네틱 스토어 쓰기를 활성화하는 경우 수신 데이터의 타임스탬프가 마그네틱 스토어에 구성된 데이터 보존보다 빠르지 않은지 확인합니다.
- 도착 지연 데이터가 예상되는 경우 테이블에 대한 마그네틱 스토어 쓰기를 켭니다. 이렇게 하면 메모리 스토어 보존 기간을 벗어났지만 자성 스토어 보존 기간 내에 있는 타임스탬프가 있는 데이터를 수집할 수 있습니다. 테이블의 에서 `EnableMagneticStoreWrites` 플래그`MagneticStoreWritesProperties`를 업데이트하여 이를 설정할 수 있습니다. 이 속성은 기본적으로 false입니다. 마그네틱 스토어에 대한 쓰기는 즉시 쿼리할 수 없습니다. 6시간 이내에 사용할 수 있습니다.
- 수집된 데이터의 타임스탬프가 메모리 스토어 보존 경계 내에 속하도록 하여 메모리 스토어에 대한 고처리량 워크로드를 대상으로 지정합니다. 마그네틱 스토어에 대한 쓰기는 데이터베이스에 대한 동시 수집을 수신할 수 있는 최대 활성 마그네틱 스토어 파티션 수로 제한됩니다. 이 지표`ActiveMagneticStorePartitions`는 에서 확인할 수 있습니다 CloudWatch. 활성 마그네틱 스토어 파티션을 줄이려면 마그네틱 스토어 수집을 위해 동시에 에 수집되는 시리즈 수와 기간을 줄이는 것을 목표로 하세요.
- 의 Timestream으로 데이터를 전송하는 동안 데이터 수집 성능을 최적화하기 위해 단일 요청으로 여러 레코드를 LiveAnalytics일괄 처리합니다.
  - 동일한 시계열의 레코드와 측정값 이름이 동일한 레코드를 일괄 배치하는 것이 좋습니다.

  - 요청이 에 정의된 서비스 한도 내에 있는 한 단일 요청에서 가능한 한 많은 레코드를 일괄 처리합니다할당량.
  - 가능한 경우 공통 속성을 사용하여 데이터 전송 및 수집 비용을 줄입니다. 자세한 내용은 섹션을 WriteRecords API참조하세요.
- 에 대한 데이터를 Timestream에 쓰는 동안 클라이언트 측에서 부분 장애가 발생하면 거부 원인을 해결한 후 수집에 실패한 레코드 배치를 다시 보낼 LiveAnalytics수 있습니다.
- 타임스탬프로 정렬된 데이터는 쓰기 성능이 더 좋습니다.
- 용 Amazon Timestream LiveAnalytics 은 애플리케이션의 필요에 따라 자동으로 확장되도록 설계되었습니다. LiveAnalytics 알림의 Timestream이 애플리케이션의 쓰기 요청이 급증하면 애플리케이션에 일정 수준의 초기 메모리 스토어 제한이 발생할 수 있습니다. 애플리케이션에 메모리 스토어 제한이 발생하는 경우 가 애플리케이션의 요구 사항을 충족하도록 자동으로 확장할 수 있도록 LiveAnalytics 동일한(또는 증가된) 속도로 LiveAnalytics 에 대한 데이터를 Timestream으로 계속 전송합니다. 마그네틱 스토어 제한 사항이 표시되면 `ActiveMagneticStorePartitions` 넘어질 때까지 마그네틱 스토어 섭취 속도를 줄여야 합니다.

배치 로드

배치 로드 모범 사례는 에 설명되어 있습니다배치 로드 모범 사례.

쿼리

다음은 Amazon Timestream for 를 사용하는 쿼리에 권장되는 모범 사례입니다 LiveAnalytics.

- 쿼리에 필수적인 측정값 및 차원 이름만 포함합니다. 외부 열을 추가하면 데이터 스캔이 증가하여 쿼리 성능에 영향을 미칩니다.
- 프로덕션에 쿼리를 배포하기 전에 쿼리 인사이트를 검토하여 공간 및 시간 정리가 최적인지 확인하는 것이 좋습니다. 자세한 내용은 Amazon Timestream에서 쿼리 인사이트를 사용하여 쿼리 최적화 단원을 참조하십시오.
- 가능한 경우, 쿼리 성능을 개선하고 비용을 절감하기 위해 WHERE 절 및 SELECT 절의 기본 제공 집계 및 스칼라 함수를 LiveAnalytics 사용하기 위해 데이터 계산을 Timestream으로 푸시합니다. SELECT 및 집계 함수 단원을 참조하세요.
- 가능하면 대략적인 함수를 사용합니다. 예: COUNT(DISTINCT columnAPPROX_DISTINCTname) 대신 _를 사용하여 쿼리 성능을 최적화하고 쿼리 비용을 절감합니다. 집계 함수을 참조하세요.
- CASE 표현식을 사용하여 동일한 테이블에서 여러 번 선택하는 대신 복잡한 집계를 수행합니다. CASE 문을 참조하세요.

- 가능한 경우 쿼리의 WHERE 절에 시간 범위를 포함합니다. 이렇게 하면 쿼리 성능과 비용이 최적화됩니다. 예를 들어 데이터 세트에 마지막 1시간의 데이터만 필요한 경우 시간 > ago(1h)와 같은 시간 기호를 포함합니다. SELECT 및 간격 및 기간 단원을 참조하세요.
- 쿼리가 테이블의 측정값 하위 집합에 액세스할 때는 항상 쿼리의 WHERE 절에서 측정값 이름을 포함합니다.
- 가능한 경우 쿼리 WHERE 절의 차원과 측정값을 비교할 때 평등 연산자를 사용합니다. 차원 및 측정 이름에 대한 평등 예측을 사용하면 쿼리 성능을 개선하고 쿼리 비용을 줄일 수 있습니다.
- 가능하면 WHERE 절의 함수를 사용하여 비용을 최적화하지 마세요.
- LIKE 절을 여러 번 사용하지 마세요. 대신 문자열 열에서 여러 값을 필터링할 때 정규식을 사용합니다. 정규식 함수을 참조하세요.
- 쿼리의 GROUP BY 절에는 필요한 열만 사용합니다.
- 쿼리 결과가 특정 순서로 되어 있어야 하는 경우 가장 바깥쪽 쿼리의 ORDER BY 절에서 해당 순서를 명시적으로 지정합니다. 쿼리 결과에 주문이 필요하지 않은 경우 ORDER BY 절을 사용하여 쿼리 성능을 개선하지 마세요.
- 쿼리에 첫 번째 N행만 필요한 경우 LIMIT 절을 사용합니다.
- ORDER BY 절을 사용하여 최상위 또는 최하위 N 값을 보는 경우 LIMIT 절을 사용하여 쿼리 비용을 줄입니다.
- 반환된 응답의 페이지 매김 토큰을 사용하여 쿼리 결과를 검색합니다. 자세한 내용은 쿼리를 참조하세요.
- 쿼리 실행을 시작했는데 쿼리가 찾고 있는 결과를 반환하지 않는다는 것을 알게 되면 쿼리를 취소하여 비용을 절감하세요. 자세한 내용은 CancelQuery를 참조하세요.
- 애플리케이션에 제한이 발생하는 경우 동일한 속도로 에 대한 LiveAnalytics 데이터를 Amazon Timestream으로 계속 전송하여 가 애플리케이션의 쿼리 처리량 요구 사항을 충족하도록 LiveAnalytics 에 대한 Amazon Timestream을 자동 확장할 수 있도록 합니다.
- 애플리케이션의 쿼리 동시성 요구 사항이 에 대한 Timestream의 기본 한도를 초과하는 경우 AWS Support 에 LiveAnalytics문의하여 한도 증가를 문의하세요.

예약된 쿼리

예약된 쿼리는 일부 플릿 전체 집계 통계를 사전 계산하여 대시보드를 최적화하는 데 도움이 됩니다. 따라서 물어봐야 할 자연스러운 질문은 사용 사례를 어떻게 받아들이고 어떤 결과를 사전 계산할지, 파생 테이블에 저장된 이러한 결과를 사용하여 대시보드를 생성하는 방법을 식별하는 것입니다. 이 프로세스의 첫 번째 단계는 사전 계산할 패널을 식별하는 것입니다. 다음은 몇 가지 상위 수준 지침입니다.

- 패널을 채우는 데 사용되는 쿼리로 스캔한 바이트, 대시보드 재로드 빈도 및 이러한 대시보드를 로드할 동시 사용자 수를 고려합니다. 가장 자주 로드되는 대시보드로 시작하고 상당한 양의 데이터를 스캔해야 합니다. 집계 대시보드 예제의 처음 두 대시보드와 드릴다운 예제의 집계 대시보드는 이러한 대시보드의 좋은 예입니다.
- 어떤 계산이 반복적으로 사용되고 있는지 고려합니다. 모든 패널과 패널에 사용되는 모든 변수 값에 대해 예약된 쿼리를 생성할 수 있지만 하나의 계산을 사용하여 여러 패널에 필요한 데이터를 사전 계산할 방법을 찾아 비용과 예약된 쿼리 수를 크게 최적화할 수 있습니다.
- 예약된 쿼리의 빈도를 고려하여 파생 테이블에서 구체화된 결과를 새로 고칩니다. 대시보드가 새로 고쳐지는 빈도와 대시보드에서 쿼리되는 시간 범위 대 사전 계산 및 대시보드의 패널에 사용되는 시간 비닝을 분석해야 합니다. 예를 들어, 지난 며칠 동안 시간당 집계를 표시하는 대시보드가 몇 시간에 한 번만 새로 고쳐지면 30분 또는 한 시간에 한 번만 새로 고치도록 예약된 쿼리를 구성해야 할 수 있습니다. 반면, 분당 집계를 표시하고 1분마다 새로 고치는 대시보드가 있는 경우 예약된 쿼리가 1분 또는 몇 분마다 결과를 새로 고치도록 해야 합니다.
- 예약된 쿼리를 사용하여 추가로 최적화할 수 있는 쿼리 패턴(쿼리 비용 및 쿼리 지연 시간 관점에서 모두)을 고려합니다. 예를 들어 대시보드의 변수로 자주 사용되는 고유한 차원 값을 계산하거나 센서에서 내보낸 마지막 데이터 포인트 또는 특정 날짜 이후에 센서에서 내보낸 첫 번째 데이터 포인트 등을 반환하는 경우입니다. 이러한 예제 패턴 중 일부는 이 안내서에서 설명합니다.

앞의 고려 사항은 파생 테이블을 쿼리하기 위해 대시보드를 이동할 때 절감에 상당한 영향을 미치며, 대시보드의 데이터의 신선도와 예약된 쿼리로 인해 발생하는 비용을 포함합니다.

클라이언트 애플리케이션 및 지원되는 통합

의 Timestream과 동일한 리전에서 클라이언트 애플리케이션을 실행 LiveAnalytics 하여 네트워크 지연 시간과 데이터 전송 비용을 줄입니다. 다른 서비스 작업에 대한 자세한 내용은 섹션을 참조하세요다른 서비스와 함께 사용. 다음은 몇 가지 유용한 링크입니다.

- 를 사용한 AWS 개발 모범 사례 AWS SDK for Java
- AWS Lambda 함수 작업 모범 사례
- Amazon Managed Service for Apache Flink 모범 사례
- Grafana에서 대시보드를 생성하는 모범 사례

일반

- 에 Timestream을 사용할 때는 AWS Well-Architected 프레임워크를 따라야 합니다 LiveAnalytics. 이 백서는 운영 우수성, 보안, 신뢰성, 성능 효율성 및 비용 최적화의 모범 사례에 대한 지침을 제공합니다.

측정 및 비용 최적화

Amazon Timestream for 를 사용하면 사용한 만큼만 LiveAnalytics비용을 지불합니다. 쓰기, 데이터 저장 및 쿼리로 스캔한 데이터에 대해 별도로 LiveAnalytics 측정기를 위한 타임스트림입니다. 각 측정 차원의 가격은 요금 페이지에 지정되어 있습니다. Amazon Timestream for LiveAnalytics Pricing Calculator를 사용하여 월별 청구서를 추정할 수 있습니다.

이 섹션에서는 Timestream for 의 쓰기, 스토리지 및 쿼리에 대한 측정 작동 방식을 설명합니다 LiveAnalytics. 예제 시나리오 및 계산도 제공됩니다. 또한 비용 최적화를 위한 모범 사례 목록이 포함되어 있습니다. 아래에서 주제를 선택할 수 있습니다.

주제

- 쓰기
- 스토리지
- 쿼리
- 비용 최적화
- Amazon을 사용한 모니터링 CloudWatch

쓰기

각 시계열 이벤트의 쓰기 크기는 타임스탬프 크기와 하나 이상의 차원 이름, 차원 값, 측정값 이름 및 측정값의 합계로 계산됩니다. 타임스탬프의 크기는 8바이트입니다. 차원 이름, 차원 값 및 측정값 이름의 크기는 각 차원 이름, 차원 값 및 측정값 이름을 나타내는 문자열의 UTF-8 인코딩 바이트 길이입니다. 측정값의 크기는 데이터 유형에 따라 달라집니다. 부울 데이터 형식의 경우 1바이트, 빅틴 및 더블의 경우 8바이트, 문자열의 경우 UTF-8 인코딩 바이트 길이입니다. 각 쓰기는 1KiB 단위로 계산됩니다.

아래에는 두 가지 계산 예시가 나와 있습니다.

주제

- 시계열 이벤트의 쓰기 크기 계산

- 쓰기 수 계산

시계열 이벤트의 쓰기 크기 계산

아래와 같이 EC2 인스턴스의 CPU 사용률을 나타내는 시계열 이벤트를 고려해 보세요.

Time	region	az	vpc	Hostname	measure_name	measure_value::double
1602983435238563000	us-east-1	1일	vpc-1a2b3c4d	host-24Gju	cpu_utilization	35.0

시계열 이벤트의 쓰기 크기는 다음과 같이 계산할 수 있습니다.

- 시간 = 8바이트
- 첫 번째 차원 = 15바이트(region+us-east-1)
- 두 번째 차원 = 4바이트(az+1d)
- 3차원 = 15바이트(vpc+vpc-1a2b3c4d)
- 네 번째 차원 = 18바이트(hostname+host-24Gju)
- 측정값 이름 = 15바이트(cpu_utilization)
- 측정값 값 = 8바이트

시계열 이벤트의 쓰기 크기 = 83바이트

쓰기 수 계산

이제 에 설명된 EC2 인스턴스와 유사한 100개의 인스턴스를 고려하여 5초마다 지표시계열 이벤트의 쓰기 크기 계산를 내보냅니다. EC2 인스턴스의 총 월별 쓰기 수는 쓰기당 존재하는 시계열 이벤트 수와 시계열 이벤트를 배치화하는 동안 공통 속성이 사용되는지 여부에 따라 달라집니다. 다음 각 시나리오에 대해 총 월별 쓰기를 계산하는 예제가 제공됩니다.

주제

- 쓰기당 시계열 이벤트 1개

- 쓰기에서 시계열 이벤트 배치
- 시계열 이벤트 배치 및 쓰기에서 공통 속성 사용

쓰기당 시계열 이벤트 1개

각 쓰기에 시계열 이벤트가 하나만 포함된 경우 총 월별 쓰기는 다음과 같이 계산됩니다.

- 시계열 이벤트 100개 = 5초마다 쓰기 100개
- x 12 쓰기/분 = 1,200 쓰기
- x 60분/시간 = 72,000개 쓰기
- x 24시간/일 = 1,728,000개 쓰기
- x 30일/월 = 51,840,000개 쓰기

총 월별 쓰기 수 = 51,840,000

쓰기에서 시계열 이벤트 배치

각 쓰기는 1KB 단위로 측정되므로 쓰기에는 12개의 시계열 이벤트(998바이트) 배치가 포함될 수 있으며 총 월별 쓰기는 다음과 같이 계산됩니다.

- 시계열 이벤트 100개 = 5초마다 쓰기 9개(쓰기당 시계열 이벤트 12개)
- x 12 쓰기/분 = 108 쓰기
- x 60분/시간 = 6,480개 쓰기
- x 24시간/일 = 155,520개의 쓰기
- x 30일/월 = 4,665,600개 쓰기

총 월별 쓰기 수 = 4,665,600

시계열 이벤트 배치 및 쓰기에서 공통 속성 사용

리전, az, vpc 및 측정값 이름이 100개의 EC2 인스턴스에서 공통인 경우, 공통 값은 쓰기당 한 번만 지정할 수 있으며 공통 속성이라고 합니다. 이 경우 공통 속성의 크기는 52바이트이고 시계열 이벤트의 크기는 27바이트입니다. 각 쓰기는 1KiB 단위로 측정되므로 쓰기에는 36개의 시계열 이벤트와 공통 속성이 포함될 수 있으며 총 월별 쓰기는 다음과 같이 계산됩니다.

- 시계열 이벤트 100개 = 5초마다 쓰기 3개(쓰기당 시계열 이벤트 36개)

- x 12 쓰기/분 = 36 쓰기
- x 60분/시간 = 2,160개 쓰기
- x 24시간/일 = 51,840개 쓰기
- x 30일/월 = 1,555,200개 쓰기

총 월별 쓰기 수 = 1,555,200

> **Note**
>
> 배치화 사용, 공통 속성 및 1KB 단위로 쓰기 반올림으로 인해 시계열 이벤트의 스토리지 크기는 쓰기 크기와 다를 수 있습니다.

스토리지

메모리 스토어와 마그네틱 스토어의 각 시계열 이벤트의 스토리지 크기는 타임스탬프 크기, 차원 이름, 차원 값, 측정값 이름 및 측정값의 합계로 계산됩니다. 타임스탬프의 크기는 8바이트입니다. 차원 이름, 차원 값 및 측정값 이름의 크기는 차원 이름, 차원 값 및 측정값 이름을 나타내는 각 문자열의 UTF-8 인코딩 바이트 길이입니다. 측정값의 크기는 데이터 유형에 따라 달라집니다. 부울 데이터 유형의 경우 1바이트, 빅틴 및 더블의 경우 8바이트, 문자열의 경우 UTF-8 인코딩 바이트 길이입니다. 각 측정값은 에 대해 Amazon Timestream에 별도의 레코드로 저장됩니다. 즉 LiveAnalytics, 시계열 이벤트에 4개의 측정값이 있는 경우 스토리지에 해당 시계열 이벤트에 대한 레코드가 4개 있습니다.

EC2 인스턴스의 CPU 사용률을 나타내는 시계열 이벤트의 예제(참조시계열 이벤트의 쓰기 크기 계산)를 고려하면 시계열 이벤트의 스토리지 크기는 다음과 같이 계산됩니다.

- 시간 = 8바이트
- 첫 번째 차원 = 15바이트(`region+us-east-1`)
- 두 번째 차원 = 4바이트(`az+1d`)
- 3차원 = 15바이트(`vpc+vpc-1a2b3c4d`)
- 네 번째 차원 = 18바이트(`hostname+host-24Gju`)
- 측정값 이름 = 15바이트(`cpu_utilization`)
- 측정값 값 = 8바이트

시계열 이벤트의 스토리지 크기 = 83바이트

> **Note**
>
> 메모리 스토어는 GB-시간 단위로 측정되고 마그네틱 스토어는 GB-월 단위로 측정됩니다.

쿼리

쿼리는 Amazon Timestream 요금 페이지에 지정된 대로 애플리케이션에서 사용하는 Timestream 컴퓨팅 단위(TCUs)의 기간을 기준으로 TCU-시간 단위로 요금이 부과됩니다. LiveAnalytics의 Amazon Timestream 쿼리 엔진은 쿼리를 처리하는 동안 관련 없는 데이터를 정리합니다. 시간 범위, 측정 이름 및/또는 차원 이름을 포함한 예측 및 예측이 있는 쿼리를 사용하면 쿼리 처리 엔진이 상당한 양의 데이터를 정리하고 쿼리 비용을 낮추는 데 도움이 됩니다.

비용 최적화

쓰기, 스토리지 및 쿼리 비용을 최적화하려면 Amazon Timestream for 에서 다음 모범 사례를 사용하세요 LiveAnalytics.

- 쓰기 요청 수를 줄이기 위해 쓰기당 여러 시계열 이벤트를 일괄 처리합니다.
- 다중 측정 레코드를 사용하는 것이 좋습니다. 이렇게 하면 단일 쓰기 요청으로 여러 시계열 측정값을 작성하고 데이터를 더 컴팩트하게 저장할 수 있습니다. 이렇게 하면 쓰기 요청 수와 데이터 스토리지 비용 및 쿼리 비용이 줄어듭니다.
- 일괄 처리와 함께 공통 속성을 사용하여 쓰기당 더 많은 시계열 이벤트를 일괄 처리하여 쓰기 요청 수를 추가로 줄입니다.
- 지연 도착 데이터 처리에 대한 애플리케이션의 요구 사항에 맞게 메모리 스토어의 데이터 보존을 설정합니다. 지연 도착 데이터는 타임스탬프가 현재 시간보다 이전이고 메모리 스토어 보존 기간을 벗어난 수신 데이터입니다.
- 장기 데이터 스토리지 요구 사항에 맞게 마그네틱 스토어의 데이터 보존을 설정합니다.
- 쿼리를 작성할 때는 쿼리에 필수적인 측정값 및 차원 이름만 포함합니다. 외부 열을 추가하면 데이터 스캔이 증가하므로 쿼리 비용도 증가합니다. 쿼리 인사이트를 검토하여 포함된 차원 및 측정값의 정리 효율성을 평가하는 것이 좋습니다.
- 가능한 경우 쿼리의 WHERE 절에 시간 범위를 포함합니다. 예를 들어 데이터 세트에 마지막 1시간의 데이터만 필요한 경우 와 같은 시간 기호를 포함합니다`time > ago(1h)`.
- 쿼리가 테이블의 측정값 하위 집합에 액세스할 때는 항상 쿼리의 WHERE 절에 측정값 이름을 포함시킵니다.

- 쿼리 실행을 시작했는데 쿼리가 찾고 있는 결과를 반환하지 않는다는 것을 알게 되면 쿼리를 취소하여 비용을 절감하세요.

Amazon을 사용한 모니터링 CloudWatch

Timestream에서 원시 데이터를 수집하여 읽을 near-real-time 수 CloudWatch있는 지표로 처리하는 Amazon 를 LiveAnalytics 사용하여 Timestream을 LiveAnalytics 모니터링할 수 있습니다. 이러한 통계는 2주간 기록되므로 기록 정보에 액세스하고 웹 애플리케이션 또는 서비스가 어떻게 실행되고 있는지 전체적으로 더 잘 파악할 수 있습니다. 기본적으로 LiveAnalytics 지표 데이터에 대한 Timestream은 1분 또는 15분 간격으로 CloudWatch 로 자동 전송됩니다. 자세한 내용은 Amazon 사용 설명서의 Amazon이란 무엇입니까 CloudWatch?를 참조하세요. CloudWatch

주제

- LiveAnalytics 지표에 Timestream을 사용하려면 어떻게 해야 하나요?
- LiveAnalytics 지표 및 차원에 대한 시간 흐름
- 에 대한 Timestream을 모니터링하는 CloudWatch 경보 생성 LiveAnalytics

LiveAnalytics 지표에 Timestream을 사용하려면 어떻게 해야 하나요?

에 대해 Timestream에서 보고하는 지표는 다양한 방식으로 분석할 수 있는 정보를 LiveAnalytics 제공합니다. 다음 목록은 몇 가지 일반적인 지표 사용 사례를 보여 줍니다. 모든 사용 사례를 망라한 것은 아니지만 시작하는 데 참고가 될 것입니다.

사용 방법	관련 지표
`How can I determine if any system errors occurred?`	`SystemErrors` 를 모니터링하여 서버 오류 코드가 발생한 요청이 있는지 확인할 수 있습니다. 일반적으로 이 지표는 0이어야 합니다. 그렇지 않다면 조사가 필요합니다.
`How can I monitor the amount of data in the memory store?`	지정된 기간 `MemoryCumulativeBytesMetered` 동안 를 모니터링하여 메모리 스토어에 저장된 데이터의 양을 바이트 단위로 모니터링할 수 있습니다. 이 지표는 매시간 방출되며 계정 및 데이터베이스 세분화에 저장된 바이트를 추적할 수 있습니다. 메모리 스토어는 GB 시간(1시간 동안 1GB의 데이터를 저장하는 비용) 단위로 측정됩니다. 따라서 의 시간당 값을 해당 리전의 GB

사용 방법	관련 지표
	시간 요금MemoryCumulativeBytesMetered 과 곱하면 시간당 발생하는 비용이 발생합니다. 차원: 작업(스토리지), DatabaseName, 지표 이름
How can I monitor the amount of data in the magnetic store?	지정된 기간 MagneticCumulativeBytesMetered 동안를 모니터링하여 마그네틱 스토어에 저장된 데이터의 양을 바이트 단위로 모니터링할 수 있습니다. 이 지표는 매시간 방출되며 계정 및 데이터베이스 세분화에 저장된 바이트를 추적할 수 있습니다. 메모리 스토어는 GB-월(1개월 동안 1GB의 데이터를 저장하는 비용) 단위로 측정됩니다. 따라서 의 시간당 값을 해당 리전의 GB-월 요금MagneticCumulativeBytesMetered 과 곱하면 시간당 발생하는 비용이 발생합니다. 예를 들어 값이 MagneticCumulativeBytesMetered 107374182400바이트(100GB)인 경우 마그네틱 스토어의 시간당 데이터 1GB 요금은 (0.03) (us-east-1 요금) / (30.4*24)입니다. 이 값을 GB MagneticCumulativeBytesMetered 단위로 곱하면 해당 시간에 대해 ~$0.004가 됩니다. 차원: 작업(스토리지), DatabaseName, 지표 이름
How can I monitor the data scanned by queries?	지정된 기간 CumulativeBytesMetered 동안 를 모니터링하여 에 대해 Timestream으로 전송된 쿼리(바이트)로 스캔된 데이터를 모니터링할 수 있습니다 LiveAnalytics. 이 지표는 쿼리 실행 후 생성되며 계정 및 데이터베이스 세분화에서 스캔된 데이터를 추적할 수 있습니다. 지표 값에 리전의 GB당 스캔 가격을 곱하여 특정 기간의 쿼리 비용을 계산할 수 있습니다. 예약된 쿼리로 스캔한 바이트는 이 지표에서 설명됩니다. 차원: 작업(쿼리), DatabaseName, 지표 이름

사용 방법	관련 지표
How can I monitor the data scanned by scheduled queries?	지정된 기간 CumulativeBytesMetered 동안 를 모니터링하여 에 대해 Timestream에서 실행한 예약된 쿼리(바이트)로 스캔한 데이터를 모니터링할 수 있습니다 LiveAnalytics. 이 지표는 쿼리 실행 후 생성되며 계정 및 데이터베이스 세분화에서 스캔된 데이터를 추적할 수 있습니다. 지표 값에 리전의 GB당 스캔 가격을 곱하여 특정 기간의 쿼리 비용을 계산할 수 있습니다. **Note** 측정된 바이트는 쿼리 에서도 고려됩니다CumulativeBytesMetered . 차원: 작업(TriggeredScheduledQuery), DatabaseName, 지표 이름
How can I monitor the number of records ingested?	지정된 기간 NumberOfRecords 동안 모니터링하여 수집된 레코드 수를 모니터링할 수 있습니다. 계정 및 데이터베이스 세분화에 저장된 바이트를 추적할 수 있습니다. 또한 이 지표를 사용하여 쿼리 결과가 별도의 테이블에 기록될 때 예약된 쿼리가 작성한 쓰기를 모니터링할 수 있습니다. 를 사용하면 각 WriteRecords 요청에 대해 지표WriteRecords API가 방출되며 CloudWatch 작업 차원은 입니다WriteRecords . BatchLoad 또는 ScheduledQuery 를 사용하면 작업이 완료될 때까지 서비스에서 결정한 간격으로 APIs지표가 방출됩니다. 이 지표의 CloudWatch 작업 차원은 가 사용되는 항목에 ScheduledQuery 따라 BatchLoad 또는 API입니다. 차원: 작업(WriteRecords BatchLoad, 또는 ScheduledQuery), DatabaseName지표 이름

사용 방법	관련 지표
How can I monitor the cost of records ingested?	를 모니터링CumulativeBytesMetered 하여 비용이 발생하는 수집된 바이트 수를 모니터링할 수 있습니다. 계정 및 데이터베이스 세분화에 저장된 바이트를 추적할 수 있습니다. 수집된 레코드는 누적 바이트 단위로 측정됩니다. 의 값에 리전의 CumulativeBytesMetered Writes 요금을 곱하면 발생한 수집 비용이 발생합니다. 를 사용할 때 WriteRecords API이 지표는 각 WriteRecords 요청에 CloudWatch 대해 방출되며 작업 차원은 입니다WriteRecords . BatchLoad 또는 ScheduledQuery 를 사용하면 작업이 완료될 때까지 서비스에서 결정한 간격으로 API 지표가 방출됩니다. 이 지표의 CloudWatch 작업 차원은 가 사용되는 항목에 ScheduledQuery 따라 BatchLoad 또는 API입니다. 차원: 작업(WriteRecords BatchLoad, 또는 ScheduledQuery), DatabaseName지표 이름
How can I monitor the Timestream Compute Units (TCUs) used in my account?	지정된 기간 QueryTCU 동안 를 모니터링하여 계정의 쿼리 워크로드에 사용된 컴퓨팅 단위를 모니터링할 수 있습니다. 이 지표는 계정의 활성 쿼리 워크로드 중에 1분당 최대 및 최소 컴퓨팅 단위로 생성됩니다. 단위: Count 유효한 통계: 최소, 최대 지표: ResourceCount 크기: Service: Timestream , Resource: QueryTCU, Type: Resource, Class: OnDemand

LiveAnalytics 지표 및 차원에 대한 시간 흐름

의 Timestream과 상호 작용하면 다음 지표와 차원을 Amazon 에 LiveAnalytics전송합니다 CloudWatch. 모든 지표는 집계되며 1분마다 보고됩니다. 다음 절차를 사용하여 에 대한 Timestream 지표를 볼 수 있습니다 LiveAnalytics.

CloudWatch 콘솔을 사용하여 지표를 보려면

지표는 먼저 서비스 네임스페이스별로 그룹화된 다음 각 네임스페이스 내에서 다양한 차원 조합별로 그룹화됩니다.

1. 에서 CloudWatch 콘솔을 엽니다https://console.aws.amazon.com/cloudwatch/.
2. 필요한 경우, 지역을 변경합니다. 탐색 모음에서 AWS 리소스가 있는 리전을 선택합니다. 자세한 내용은 AWS 서비스 엔드포인트를 참조하십시오.
3. 탐색 창에서 지표(Metrics)를 선택합니다.
4. 모든 지표 탭에서 `AWS/Timestream for LiveAnalytics.`를 선택합니다.

를 사용하여 지표를 보려면 AWS CLI

- 명령 프롬프트에서 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace "AWS/Timestream"
```

LiveAnalytics 지표에 대한 Timestream의 차원

에 대한 Timestream 지표는 계정, 테이블 이름 또는 작업의 값에 따라 검증 LiveAnalytics 됩니다. CloudWatch 콘솔을 사용하여 다음 표의 모든 차원에 따라 LiveAnalytics 데이터에 대한 Timestream을 검색할 수 있습니다.

측정기준	설명
`DatabaseName`	이 차원은 데이터를 LiveAnalytics 데이터베이스의 특정 Timestream으로 제한합니다. 이 값은 현재 리전 및 현재 AWS 계정의 모든 데이터베이스일 수 있습니다.
`Operation`	이 차원은 데이터를 , `Storage`, `WriteRecords` `BatchLoad` 또는 와 같은 LiveAnalytics 작업에 대한 Timestream 중 하

측정기준	설명
	나로 제한합니다ScheduledQuery . 사용 가능한 값 목록은 LiveAnalytics 쿼리 API 참조를 위한 Timestream을 참조하세요.
TableName	이 차원은 LiveAnalyticss 데이터베이스용 Timestream의 특정 테이블로 데이터를 제한합니다.

Important

CumulativeBytesMetered, UserErrors 및 SystemErrors 지표에는 Operation차원만 있습니다. SuccessfulRequestLatency 지표에는 항상 Operation 차원이 있지만 값에 따라 DatabaseName 및 TableName차원도 있을 수 있습니다Operation. 테이블 수준 작업의 LiveAnalytics Timestream에는 DatabaseName 및 가 차원TableName으로 있지만 계정 수준 작업은 그렇지 않기 때문입니다.

LiveAnalytics 지표의 타임스트림

Note

Amazon은 1분 간격으로 LiveAnalytics 지표에 대해 다음 모든 Timestream을 CloudWatch 집계합니다.

일반 지표

지표	설명
SuccessfulRequestLatency	지정된 기간 LiveAnalytics 동안 에 대한 Timestream에 대한 성공적인 요청입니다. SuccessfulRequestLatency 는 두 가지 종류의 정보를 제공할 수 있습니다. • 성공한 요청의 경과 시간(최소, 최대, 합계 또는 평균). • 성공한 요청의 수(SampleCount)

지표	설명
	SuccessfulRequestLatency 는 에 대한 Timestream 내의 활동만 반영 LiveAnalytics 하며 네트워크 지연 시간 또는 클라이언트 측 활동은 고려하지 않습니다. 단위: `Milliseconds` 차원 • `DatabaseName` • `TableName` • `Operation` 유효한 통계: • `Minimum` • `Maximum` • `Average` • `SampleCount` • `P10` • `p50` • `p90` • `p95` • `p99`

쓰기 및 스토리지 지표

지표	설명
`MagneticStoreRejectedRecordCount`	비동기적으로 거부된 마그네틱 스토어 쓰기 레코드 수입니다. 새 레코드의 버전이 현재 버전보다 작거나 새 레코드의 버전이 현재 버전과 같지만 데이터가 다른 경우 이 문제가 발생할 수 있습니다.

지표	설명
	단위: Count 차원 • DatabaseName • TableName • Operation 유효한 통계: • Sum • SampleCount
MagneticStoreRejectedUploadUserFailures	사용자 오류로 인해 업로드되지 않은 마그네틱 스토어 거부 레코드 보고서의 수입니다. 이는 IAM 권한이 올바르게 구성되지 않았거나 S3 버킷이 삭제되었기 때문일 수 있습니다. 단위: Count 차원 • DatabaseName • TableName • Operation 유효한 통계: • Sum • SampleCount

지표	설명
MagneticStoreRejectedUpload SystemFailures	시스템 오류로 인해 업로드되지 않은 마그네틱 스토어 거부 레코드 보고서의 수입니다. 단위: Count 차원 • DatabaseName • TableName • Operation 유효한 통계: • Sum • SampleCount
ActiveMagneticStorePartitions	지정된 시간에 데이터를 적극적으로 수집하는 마그네틱 스토어 파티션의 수입니다. 단위: Count 차원 • DatabaseName • Operation 유효한 통계: • Sum • SampleCount

지표	설명
MagneticStorePendingRecordsLatency	쿼리에 사용할 수 없는 마그네틱 스토어에 대한 가장 오래된 쓰기입니다. 마그네틱 스토어에 기록된 레코드는 6시간 이내에 쿼리할 수 있습니다. 단위: Milliseconds 차원 • DatabaseName • TableName • Operation 유효한 통계: • Minimum • Maximum • Average • SampleCount • P10 • p50 • p90 • p95 • p99
MemoryCumulativeBytesMetered	메모리 스토어에 저장된 데이터의 양, 바이트 단위: Bytes 차원: Operation 유효한 통계: • Average

지표	설명
MagneticCumulativeBytesMetered	마그네틱 스토어에 저장된 데이터의 양, 바이트 단위: Bytes 차원: Operation 유효한 통계: • Average
CumulativeBytesMetered	에 대해 Timestream으로 수집하여 측정되는 데이터의 양으로 LiveAnalytics, 바이트 단위로 표시됩니다. 단위: Bytes 차원: Operation 유효한 통계: Sum
NumberOfRecords	에 대해 Timestream에 수집된 레코드 수입니다 LiveAnalytics. 단위: Count 차원: Operation 유효한 통계: Sum

쿼리 지표

지표	설명
CumulativeBytesMetered	에 대해 Timestream으로 전송된 쿼리로 스캔한 데이터의 양으로 LiveAnalytics, 바이트 단위입니다. 단위: Bytes

지표	설명
	차원: `Operation` 유효한 통계: • `Sum`
`ResourceCount`	계정의 쿼리 워크로드에 사용되는 Timestream Compute Units(TCUs)입니다. 이 지표는 계정의 활성 쿼리 워크로드 중에 1분당 최대 및 최소 컴퓨팅 단위로 생성됩니다. 단위: `Count` 유효한 통계: 최소, 최대 크기: `Service: Timestream` , `Resource: QueryTCU`, `Type: Resource`, `Class: OnDemand`

오류 지표

지표	설명
`SystemErrors`	지정된 기간 SystemError 동안 를 LiveAnalytics 생성하는 에 대한 Timestream에 대한 요청입니다. 는 SystemError 일반적으로 내부 서비스 오류를 나타냅니다. 단위: `Count` 차원: `Operation` 유효한 통계: • `Sum` • `SampleCount`

지표	설명
UserErrors	지정된 기간 동안 InvalidRequest 오류를 LiveAnalytics 생성하는 에 대한 Timestream에 대한 요청입니다. 는 InvalidRequest 일반적으로 잘못된 파라미터 조합, 존재하지 않는 테이블 업데이트 시도 또는 잘못된 요청 서명과 같은 클라이언트 측 오류를 나타냅니다. UserErrors 는 현재 AWS 리전 및 현재 AWS 계정에 대한 잘못된 요청의 집계를 나타냅니다. 단위: Count 차원: Operation 유효한 통계: • Sum • SampleCount

Important

Average 또는 Sum과 같은 모든 지표에 적용되지 않는 통계도 있습니다. 그러나 이러한 모든 값은 LiveAnalytics 콘솔용 Timestream을 통해, 콘솔을 사용하여 CloudWatch AWS CLI또는 AWS SDKs 모든 지표에 사용할 수 있습니다.

에 대한 Timestream을 모니터링하는 CloudWatch 경보 생성 LiveAnalytics

CloudWatch 경보 상태가 변경될 때 Amazon Simple Notification Service(Amazon SNS) 메시지를 LiveAnalytics 보내는 에 대한 Timestream용 Amazon 경보를 생성할 수 있습니다. 경보는 지정한 기간 동안 단일 지표를 감시합니다. 기간 수에 대한 주어진 임계값과 지표 값을 비교하여 하나 이상의 작업을 수행합니다. 작업은 Amazon SNS 주제 또는 Auto Scaling 정책으로 전송되는 알림입니다.

경보는 지속적인 상태 변경에 대해서만 작업을 호출합니다. CloudWatch 경보는 특정 상태라는 이유만으로 작업을 호출하지 않습니다. 상태가 변경되어 지정된 기간 수 동안 유지되어야 합니다.

CloudWatch 경보 생성에 대한 자세한 내용은 Amazon 사용 설명서의 Amazon CloudWatch 경보 사용을 참조하세요. CloudWatch

문제 해결

이 섹션에는 의 Timestream 문제 해결에 대한 정보가 포함되어 있습니다 LiveAnalytics.

주제

- WriteRecords 스로틀 처리
- 거부된 레코드 처리
- 에 대한 TimestreamUNLOAD의 문제 해결 LiveAnalytics
- LiveAnalytics 특정 오류 코드의 시간대

WriteRecords 스로틀 처리

Timestream에 대한 메모리 스토어 쓰기 요청은 애플리케이션의 데이터 수집 요구 사항에 맞게 Timestream 규모가 조정됨에 따라 제한될 수 있습니다. 애플리케이션에서 제한적인 예외가 발생하는 경우 Timestream이 애플리케이션의 요구 사항에 맞게 자동으로 확장할 수 있도록 동일한(또는 더 높은) 처리량으로 데이터를 계속 전송해야 합니다.

수집을 수신하는 마그네틱 스토어 파티션의 최대 한도가 있는 경우 Timestream에 대한 마그네틱 스토어 쓰기 요청이 제한될 수 있습니다. 이 데이터베이스에 대한 `ActiveMagneticStorePartitions` Cloudwatch 지표를 확인하라는 스로틀 메시지가 표시됩니다. 이 제한은 해결하는 데 최대 6시간이 걸릴 수 있습니다. 이 제한 사항을 방지하려면 처리량이 높은 수집 워크로드에 메모리 스토어를 사용해야 합니다. 마그네틱 스토어 수집의 경우 수집되는 시리즈 수와 기간을 제한하여 더 적은 파티션으로 수집을 대상으로 지정할 수 있습니다.

데이터 수집 모범 사례에 대한 자세한 내용은 섹션을 참조하세요쓰기.

거부된 레코드 처리

Timestream이 레코드를 거부하면 거부에 대한 세부 정보가 `RejectedRecordsException` 포함된 를 받게 됩니다. WriteRecords 응답에서 이 정보를 추출하는 방법에 대한 자세한 내용은 쓰기 실패 처리를 참조하세요.

새 레코드의 버전이 기존 레코드의 버전 이하인 마그네틱 스토어에 대한 업데이트를 제외하고 모든 거부가 이 응답에 포함됩니다. 이 경우 Timestream은 버전이 더 높은 기존 레코드를 업데

이트하지 않습니다. Timestream은 더 낮거나 동일한 버전의 새 레코드를 거부하고 이러한 오류를 S3 버킷에 비동기적으로 기록합니다. 이러한 비동기 오류 보고서를 수신하려면 테이블에 `MagneticStoreRejectedDataLocation` 속성을 `MagneticStoreWriteProperties`로 설정해야 합니다.

에 대한 TimestreamUNLOAD의 문제 해결 LiveAnalytics

다음은 UNLOAD 명령과 관련된 문제 해결 지침입니다.

범주	오류 메시지	문제 해결 방법
S3 키 길이	UNLOAD 대상에 제공된 S3 접두사 [%s]를 사용할 때 결과 파일 키는 S3 허용 키 길이를 초과합니다. 자세한 내용은 설명서를 참조하세요.	UNLOAD 문을 사용하여 쿼리 결과를 내보낼 때 S3 버킷 이름과 접두사 길이의 합계로 구성된 S3 키 길이가 지원되는 최대 S3 키 길이를 초과합니다. 접두사 또는 버킷 이름 길이를 줄이는 것이 좋습니다.
	UNLOAD partitioned_by [%s]를 사용할 때 결과 파일 키는 S3 허용 키 길이를 초과합니다. 자세한 내용은 설명서를 참조하세요.	UNLOAD 문을 사용하여 쿼리 결과를 내보낼 때 partitioned_by 열을 사용하는 S3 키 길이가 지원되는 최대 S3 키 길이를 초과합니다. 대체 열로 분할하거나 partitioned_column(가능한 경우)의 길이를 줄이는 것이 좋습니다.
	UNLOAD S3 접두사 [%s]를 파티션된_x [%s]와 함께 사용할 때 결과 파일 키는 S3 허용 키 길이를 초과합니다. 자세한 내용은 설명서를 참조하세요.	UNLOAD 문을 사용하여 쿼리 결과를 내보낼 때 S3 버킷 이름, 접두사 및 파티션된_바이 열 이름의 길이의 합계로 구성된 S3 키 길이가 지원되는 최대 S3 키 길이를 초과합니다. 접두사, 버킷 이름 길이를 줄이거나 대체 열을 사용하여 데이터를 분할하는 것이 좋습니다.

범주	오류 메시지	문제 해결 방법
	생성된 S3 객체 키: %s가 너무 깁니다. 자세한 내용은 설명서를 참조하세요.	UNLOAD 문을 사용하여 쿼리를 처리하는 동안 분할된 열의 값 중 하나가 지원되는 최대 S3 키 길이를 초과합니다. 파티션 열과 값은 생성된 객체 키에서 찾을 수 있습니다.
S3 스로틀	Amazon S3가 UNLOAD 명령에서 쓰기를 제한하는 것을 감지했습니다. 자세한 내용은 Amazon Timestream 설명서를 참조하세요.	여기에서 S3 설명서를 참조하세요. 여러 리더/라이터가 동일한 폴더에 액세스할 때 S3 API 통화 속도가 제한될 수 있습니다. 제공된 버킷에 대한 통화 볼륨을 감사하십시오. 여러 동시 UNLOAD 쿼리에 동일한 버킷을 사용하는 경우 동일한 버킷에 대해 다른 버킷을 사용해 보세요. 에 대한 Timestream 이외의 여러 작업에 동일한 버킷을 사용하는 경우 UNLOAD 결과를 별도의 버킷으로 이동하는 것이 LiveAnalytics UNLOAD좋습니다.

LiveAnalytics 특정 오류 코드의 시간대

이 섹션에는 의 Timestream에 대한 특정 오류 코드가 포함되어 있습니다 LiveAnalytics.

LiveAnalytics 쓰기 API 오류의 시간 흐름

InternalServerException

HTTP 상태 코드: 500

ThrottlingException

HTTP 상태 코드: 429

ValidationException

HTTP 상태 코드: 400

ConflictException

HTTP 상태 코드: 409

AccessDeniedException

이 작업을 수행할 수 있는 충분한 액세스 권한이 없습니다.

HTTP 상태 코드: 403

ServiceQuotaExceededException

HTTP 상태 코드: 402

ResourceNotFoundException

HTTP 상태 코드: 404

RejectedRecordsException

HTTP 상태 코드: 419

InvalidEndpointException

HTTP 상태 코드: 421

LiveAnalytics 쿼리 API 오류에 대한 시간 흐름

ValidationException

HTTP 상태 코드: 400

QueryExecutionException

HTTP 상태 코드: 400

ConflictException

HTTP 상태 코드: 409

ThrottlingException

HTTP 상태 코드: 429

InternalServerException

HTTP 상태 코드: 500

InvalidEndpointException

HTTP 상태 코드: 421

할당량

이 주제에서는 Amazon Timestream for 에서 제한이라고도 하는 현재 할당량을 설명합니다 LiveAnalytics. 각 할당량은 다르게 지정되지 않는 한 리전별로 적용됩니다.

주제

기본 할당량

다음 테이블에는 LiveAnalytics 할당량에 대한 Timestream과 기본값이 포함되어 있습니다.

displayName	설명	defaultValue
계정당 데이터베이스	별로 생성할 수 있는 최대 데이터베이스 수입니다 AWS 계정.	500
계정당 표	별로 생성할 수 있는 최대 테이블 수입니다 AWS 계정.	50000
에 대한 제한 속도 CRUD APIs	현재 리전에서 계정당 초당 허용되는 최대 Create/Update/	1

displayName	설명	defaultValue
	List/Describe/Delete database/table/scheduled 쿼리 API 요청 수입니다.	
계정당 예약 쿼리	별로 생성할 수 있는 예약 쿼리의 최대 수입니다 AWS 계정.	10000
활성 마그네틱 스토어 파티션의 최대 수	데이터베이스당 최대 활성 마그네틱 스토어 파티션 수입니다. 파티션은 수집을 받은 후 최대 6시간 동안 활성 상태로 유지될 수 있습니다.	250
maxQueryTCU	계정에 대해 설정할 수 TCUs 있는 최대 쿼리입니다.	1000

서비스 한도

다음 테이블에는 LiveAnalytics 서비스 제한에 대한 Timestream과 기본값이 포함되어 있습니다. 콘솔에서 테이블에 대한 데이터 보존을 편집하려면 테이블 편집을 참조하세요.

displayName	설명	defaultValue
향후 수집 기간(분)	현재 시스템 시간과 비교한 시계열 데이터의 최대 리드 타임(분) 예를 들어 향후 수집 기간이 15분인 경우 LiveAnalytics의 Timestream은 현재 시스템 시간보다 최대 15분 앞선 데이터를 수락합니다.	15
메모리 저장소의 최소 보존 기간(시간)	테이블당 메모리 저장소에 유지해야 할 최소 기간(시간)	1

displayName	설명	defaultValue
메모리 저장소의 최대 보존 기간(시간)	표당 데이터를 메모리 저장소에 보관할 수 있는 최대 기간(시간)	8766
마그네틱 스토어의 최소 보존 기간(일)	표당 마그네틱 저장소에 데이터가 보관되어야 하는 최소 기간(일)	1
마그네틱 저장소의 최대 보존 기간(일)	마그네틱 스토어에 데이터를 보관할 수 있는 최대 기간(일) 이 값은 200년과 같습니다.	73000
마그네틱 스토어의 기본 보존 기간 일 수	테이블당 마그네틱 스토어에 데이터가 유지되는 기본값(일) 입니다. 이 값은 200년과 같습니다.	73000
시간 단위 메모리 스토어의 기본 보존 기간	메모리 스토어에 데이터가 보존되는 기본 기간(시간)입니다.	6
표당 차원	표당 최대 차원 수	128
테이블당 측정 이름	테이블당 고유 측정 이름의 최대 수입니다.	8192
차원 이름, 차원, 값, 시리즈별, 쌍, 크기	시리즈별 차원 이름 및 차원 값 쌍의 최대 크기	2KB
최대 레코드 크기	레코드의 최대 크기입니다.	2KB
요청당 WriteRecords API 레코드 수	요청의 최대 레코드 수입니다 WriteRecords API.	100
차원 이름 길이	차원 이름의 최대 바이트 수입니다.	60바이트

displayName	설명	defaultValue
이름 길이 측정값	측정값 이름의 최대 바이트 수입니다.	256 bytes
데이터베이스 이름 길이	데이터베이스 이름의 최대 바이트 수입니다.	256 bytes
표 이름 길이	테이블 이름의 최대 바이트 수입니다.	256 bytes
QueryString KiB의 길이	쿼리 문자열의 최대 길이(KiB 단위)로, 쿼리에 대해 UTF-8개의 인코딩된 문자입니다.	256
쿼리 실행 기간(시간)	쿼리의 최대 실행 기간(시간) 쿼리가 오래 걸리면 제한 시간이 초과됩니다.	1
쿼리 인사이트	현재 리전에서 계정당 초당 쿼리 인사이트가 활성화된 상태로 허용되는 최대 쿼리 API 요청 수입니다.	1
쿼리 결과의 메타데이터 크기	쿼리 결과의 최대 메타데이터 크기	100킬로바이트
쿼리 결과의 데이터 크기	쿼리 결과의 최대 데이터 크기	5기가바이트
다중 측정 기록당 측정값	다중 측정값 기록당 최대 측정값 수	256
다중 측정값 기록당 측정값 크기	다중 측정 기록당 최대 측정값 크기	2048
표당 다중 측정값 기록 전반의 고유한 측정값	단일 표에 정의된 모든 다중 측정값 기록의 고유한 측정값	1024

displayName	설명	defaultValue
계정당 Timestream Compute Units(TCUs)	계정TCUs당 기본 최대값입니다.	200

지원되는 데이터 유형

다음 표에서는 측정값 및 차원 값에 대해 지원되는 데이터 유형을 설명합니다.

설명	LiveAnalytics 값에 대한 시간 흐름
측정값에 지원되는 데이터 유형입니다.	빅 int, 더블, 문자열, 부울, MULTI, 타임스탬프
차원 값에 지원되는 데이터 유형입니다.	String

배치 로드

배치 로드 내에서 제한이라고도 하는 현재 할당량은 다음과 같습니다.

설명	LiveAnalytics 값에 대한 시간 흐름
최대 배치 로드 작업 크기	최대 배치 로드 태스크 크기는 100GB를 초과할 수 없습니다.
파일 수량	배치 로드 태스크에는 100개 이상의 파일을 포함할 수 없습니다.
최대 파일 크기	배치 로드 작업의 최대 파일 크기는 5GB를 초과할 수 없습니다.
CSV 파일 행 크기	CSV 파일의 행은 16MB를 초과할 수 없습니다. 이는 늘릴 수 없는 하드 한도입니다.
활성 배치 로드 작업	테이블에는 5개 이상의 활성 배치 로드 작업이 있을 수 없으며 계정에는 10개 이상의 활성 배치 로드 작업이 있을 수 없습니다. 의 Timestream LiveAnalytics 은 더 많은 리소스를 사용할 수 있을 때까지 새 배치 로드 작업을 제한합니다.

명명 제약 조건

다음 표에서는 이름 지정 제약 조건에 대해 설명합니다.

설명	LiveAnalytics 값에 대한 시간 흐름
차원 이름의 최대 길이입니다.	60바이트
측정값 이름의 최대 길이입니다.	256 bytes
테이블 이름 또는 데이터베이스 이름의 최대 길이입니다.	256 bytes
테이블 및 데이터베이스 이름	• 를 사용하지 않는 것이 좋습니다시스템 식별자. • a-z A-Z 0-9 _ (밑줄) - (대시) . (점)을 포함할 수 있습니다. • 모든 이름은 UTF-8로 인코딩되어야 하며 대/소문자를 구분합니다. Note 테이블 및 데이터베이스 이름은 UTF-8 바이너리 표현을 사용하여 비교됩니다. 즉, ASCII 문자에 대한 비교는 대/소문자를 구분합니다.
측정 이름	• ':'를 포함시스템 식별자하거나 콜론할 수 없습니다. • 예약된 접두사(`ts_`,)로 시작해서는 안 됩니다`measure_value` . Note 테이블 및 데이터베이스 이름은 UTF-8 바이너리 표현을 사용하여 비교됩니다. 즉, ASCII 문자에 대한 비교는 대/소문자를 구분합니다.
차원 이름	• 시스템 식별자, 콜론 ':' 또는 큰따옴표(')를 포함할 수 없습니다.

설명	LiveAnalytics 값에 대한 시간 흐름
	• 예약된 접두사(ts_,)로 시작해서는 안 됩니다measure_value . • 여기에 나열된 유니코드 문자 [0,31] 또는 “\u2028” 또는 “\u2029”를 포함할 수 없습니다. **Note** 차원 및 측정값 이름은 UTF-8 바이너리 표현을 사용하여 비교됩니다. 즉, ASCII 문자에 대한 비교는 대/소문자를 구분합니다.
모든 열 이름	열 이름은 복제할 수 없습니다. 다중 측정 레코드는 차원 및 측정값을 열로 나타내므로 차원의 이름은 측정값의 이름과 같을 수 없습니다. 이름은 대/소문자를 구분합니다.

예약어

다음은 모두 예약된 키워드입니다.

- ALTER
- AND
- AS
- BETWEEN
- BY
- CASE
- CAST
- CONSTRAINT
- CREATE
- CROSS
- CUBE
- CURRENT_DATE
- CURRENT_TIME

- CURRENT_TIMESTAMP
- CURRENT_USER
- DEALLOCATE
- DELETE
- DESCRIBE
- DISTINCT
- DROP
- ELSE
- END
- ESCAPE
- EXCEPT
- EXECUTE
- EXISTS
- EXTRACT
- FALSE
- FOR
- FROM
- FULL
- GROUP
- GROUPING
- HAVING
- IN
- INNER
- INSERT
- INTERSECT
- INTO
- IS
- JOIN
- LEFT
- LIKE

- LOCALTIME
- LOCALTIMESTAMP
- NATURAL
- NORMALIZE
- NOT
- NULL
- ON
- OR
- ORDER
- OUTER
- PREPARE
- RECURSIVE
- RIGHT
- ROLLUP
- SELECT
- TABLE
- THEN
- TRUE
- UESCAPE
- UNION
- UNNEST
- USING
- VALUES
- WHEN
- WHERE
- WITH

시스템 식별자

열 이름 “measure_value”, “ts_non_existent_col” 및 “time”은 시스템 식별자의 LiveAnalytics Timestream으로 예약됩니다. 또한 열 이름은 “ts_” 또는 “measure_name”으로 시작할 수 없습니다. 시

스템 식별자는 대/소문자를 구분합니다. 식별자는 UTF-8 바이너리 표현을 사용하여 비교되었습니다. 즉, 식별자에 대한 비교는 대/소문자를 구분합니다.

> **Note**
>
> 시스템 식별자는 차원 또는 측정 이름에 사용할 수 없습니다. 데이터베이스 또는 테이블 이름에는 시스템 식별자를 사용하지 않는 것이 좋습니다.

UNLOAD

UNLOAD 명령과 관련된 제한은 UNLOAD를 사용하여 쿼리 결과를 Timestream 에서 S3로 내보내기를 참조하세요.

쿼리 언어 참조

> **Note**
>
> 이 쿼리 언어 참조에는 Apache License, 버전 2.0에 따라 라이선스가 부여된 Trino Software Foundation(이전 Presto Software Foundation)의 다음과 같은 타사 설명서가 포함되어 있습니다. 이 라이선스를 준수하는 경우를 제외하고 이 파일을 사용할 수 없습니다. Apache 라이선스 버전 2.0의 사본을 가져오려면 Apache 웹 사이트 를 참조하세요.

의 Timestream은 데이터 작업을 위한 풍부한 쿼리 언어를 LiveAnalytics 지원합니다. 아래에서 사용 가능한 데이터 유형, 연산자, 함수 및 구성을 확인할 수 있습니다.

샘플 쿼리 섹션에서 Timestream의 쿼리 언어를 즉시 시작할 수도 있습니다.

주제

- 지원되는 데이터 유형
- 기본 제공 시계열 기능
- SQL 지원
- 논리 연산자
- 비교 연산자
- 비교 함수
- 조건식

- 변환 함수
- 수학적 연산
- 수학 함수
- 문자열 연산자
- 문자열 함수
- 배열 연산자
- 배열 함수
- 비트 함수
- 정규식 함수
- 날짜/시간 연산자
- 날짜/시간 함수
- 집계 함수
- 윈도 함수
- 샘플 쿼리

지원되는 데이터 유형

LiveAnalytics의 쿼리 언어에 대한 Timestream은 다음 데이터 유형을 지원합니다.

> **Note**
>
> 쓰기에 지원되는 데이터 유형은 데이터 유형 에 설명되어 있습니다.

데이터 유형	설명
`int`	32비트 정수를 나타냅니다.
`bigint`	64비트 부호 있는 정수를 나타냅니다.
`boolean`	로직 및 의 두 가지 진실 값 중 하나입니다`TrueFalse`.
`double`	64비트 변수 정밀도 데이터 유형을 나타냅니다. IEEE 바이너리 부동 소수점 산술용 표준 754를 구현합니다.

데이터 유형	설명
	Note 쿼리 언어는 데이터를 읽기 위한 것입니다. 쿼리에 사용할 수 있는 `Infinity` 및 `NaN` 이중 값에 대한 함수가 있습니다. 하지만 이러한 값을 Timestream에 쓸 수는 없습니다.
`varchar`	최대 크기가 2KB 가변 길이 문자 데이터입니다.
`array[`*`T`*`,...]`	지정된 데이터 유형의 하나 이상의 요소를 포함합니다.*`T`*, 여기서 *`T`* 는 Timestream에서 지원되는 모든 데이터 유형일 수 있습니다.
`row(`*`T`*`,...)`	데이터 유형의 이름이 지정된 필드를 하나 이상 포함합니다.*`T`*. 필드는 Timestream에서 지원하는 모든 데이터 유형일 수 있으며 점 필드 참조 연산자를 사용하여 액세스할 수 있습니다. `.`
`date`	는 형식으로 날짜를 나타냅니다*`YYYY`*`-`*`MM`*`-`*`DD`*. 여기서 *`YYYY`* 는 연도이고 *`MM`* 는 월이며,*`DD`* 는 각각 일입니다. 지원되는 범위는 `1970-01-01` ~입니다`2262-04-11` . 예: `1971-02-03`
`time`	의 시간을 나타냅니다UTC. `time` 데이터 유형은 나노초 정밀도 *`HH`*`.`*`MM`*`.`*`SS`*`.`*`sssssssss`* . 지원 형식으로 표시됩니다. 예: `17:02:07.496000000`

데이터 유형	설명
`timestamp`	에서 나노초 정밀도 시간을 사용하여 인스턴스를 시간 단위로 나타냅니다UTC. *YYYY*-*MM*-*DD* *hh*:*mm*:*ss*.*sssssssss* 쿼리는 ~ 범위의 타임스탬프`1677-09-21 00:12:44.000000000` 를 지원합니다`2262-04-11 23:47:16.854775807` .

데이터 유형	설명
`interval`	두 부분으로 *X* *t*구성된 문자열 리터럴 로 시간 간격을 나타냅니다.*X* 그리고 *t*. *X* 는 보다 크거나 같은 숫자 값이며0,*t* 는 초 또는 시간과 같은 시간 단위입니다. 단위는 복수화되지 않습니다. 시간 단위 *t* 는 다음 문자열 리터럴 중 하나여야 합니다. • `nanosecond` • `microsecond` • `millisecond` • `second` • `minute` • `hour` • `day` • `ns` (와 동일`nanosecond`) • `us` (와 동일`microsecond`) • `ms` (와 동일`millisecond`) • `s` (와 동일`second`) • `m` (와 동일`minute`) • `h` (와 동일`hour`) • `d` (와 동일`day`) 예제: `17s` `12second` `21hour`

데이터 유형	설명
	`2d`
`timeseries[row(timestamp, T,...)]`	일정 시간 동안 기록된 측정값의 값을 `row` 객체로 `array` 구성된 값으로 나타냅니다. 각 에는 데이터 유형의 `timestamp` 및 하나 이상의 측정값이 `row` 포함되어 있습니다.*T*, 여기서 *T* 는 `bigint`, `boolean`, `double`또는 중 하나일 수 있습니다`varchar`. 행은 에 따라 오름차순으로 정렬됩니다`timestamp` . 시간 데이터 형식은 시간 경과에 따른 측정값의 값을 나타냅니다.
unknown	null 데이터를 나타냅니다.

기본 제공 시계열 기능

용 Timestream LiveAnalytics 은 시계열 데이터를 1급 개념으로 취급하는 기본 제공 시계열 기능을 제공합니다.

내장된 시계열 기능은 보기와 함수의 두 범주로 나눌 수 있습니다.

아래에서 각 구성에 대해 읽을 수 있습니다.

주제

- 시계열 보기
- 시계열 함수

시계열 보기

의 Timestream은 데이터를 `timeseries` 데이터 유형으로 변환하기 위해 다음 함수를 LiveAnalytics 지원합니다.

주제

- CREATE_TIME_SERIES
- UNNEST

CREATE_TIME_SERIES

CREATE_TIME_SERIES는 시계열의 모든 원시 측정값(시간 및 측정값 값)을 가져와서 시간 데이터 유형을 반환하는 집계 함수입니다. 이 함수의 구문은 다음과 같습니다.

```
CREATE_TIME_SERIES(time, measure_value::<data_type>)
```

여기서 <data_type>는 측정값의 데이터 유형이며 빅인트, 부울, 더블 또는 varchar 중 하나일 수 있습니다. 두 번째 파라미터는 null일 수 없습니다.

아래와 같이 지표라는 테이블에 저장된 EC2 인스턴스의 CPU 사용률을 고려합니다.

Time	region	az	vpc	instance-id	measure_name	measure_value::double
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	35.0
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	38.2
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	cpu_utilization	45.3
2019-12-04 19:00:00.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	54.1
2019-12-04 19:00:01.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	42.5
2019-12-04 19:00:02.000000000	us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	cpu_utilization	33.7

쿼리 실행:

```
SELECT region, az, vpc, instance_id, CREATE_TIME_SERIES(time, measure_value::double) as
 cpu_utilization FROM metrics
    WHERE measure_name='cpu_utilization'
    GROUP BY region, az, vpc, instance_id
```

는 측정값cpu_utilization으로 가 있는 모든 시리즈를 반환합니다. 이 경우 두 가지 시리즈가 있습니다.

region	az	vpc	instance-id	cpu_utilization
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef0	[{time: 2019-12-04 19:00:00.000000000, measure_value::double: 35.0}, {time: 2019-12-04 19:00:01.000000000, measure_value::double: 38.2}, {time: 2019-12-04 19:00:02.000000000, measure_value::double: 45.3}]
us-east-1	us-east-1d	vpc-1a2b3c4d	i-1234567890abcdef1	[{time: 2019-12-04 19:00:00.000000000, measure_value::double: 35.1}, {time: 2019-12-0

region	az	vpc	instance-id	cpu_utilization
				4 19:00:01.000000000, measure_value::double: 38.5}, {time: 2019-12-04 19:00:02.000000000, measure_value::double: 45.7}]

UNNEST

UNNEST 는 timeseries 데이터를 플랫 모델로 변환할 수 있는 테이블 함수입니다. 구문은 다음과 같습니다.

UNNEST 는 를 두 열, 즉 time 및 timeseries로 변환합니다value. 다음과 UNNEST 같이 에서 별칭을 사용할 수도 있습니다.

```
UNNEST(timeseries) AS <alias_name> (time_alias, value_alias)
```

여기서 <alias_name> 는 플랫 테이블의 별칭이고 time_alias는 time 열의 별칭이며 value_alias는 value 열의 별칭입니다.

예를 들어, 플릿의 일부 EC2 인스턴스가 5초 간격으로 지표를 내보내도록 구성되고, 다른 인스턴스는 15초 간격으로 지표를 내보내며, 지난 6시간 동안 10초 단위로 모든 인스턴스의 평균 지표가 필요한 시나리오를 고려해 보세요. 이 데이터를 가져오려면 CREATE_TIME_SERIES를 사용하여 지표를 시계열 모델로 변환합니다. 그런 다음 INTERPOLATE_LINEAR를 사용하여 10초 단위로 누락된 값을 가져올 수 있습니다. 그런 다음 를 사용하여 데이터를 플랫 모델로 다시 변환UNNEST한 다음 AVG를 사용하여 모든 인스턴스의 평균 지표를 가져옵니다.

```
WITH interpolated_timeseries AS (
    SELECT region, az, vpc, instance_id,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(time, measure_value::double),
```

```
                SEQUENCE(ago(6h), now(), 10s)) AS interpolated_cpu_utilization
    FROM timestreamdb.metrics
    WHERE measure_name= 'cpu_utilization' AND time >= ago(6h)
    GROUP BY region, az, vpc, instance_id
)
SELECT region, az, vpc, instance_id, avg(t.cpu_util)
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_cpu_utilization) AS t (time, cpu_util)
GROUP BY region, az, vpc, instance_id
```

위의 쿼리는 별칭과 UNNEST 함께 를 사용하는 방법을 보여줍니다. 다음은 에 대한 별칭을 사용하지 않고 동일한 쿼리의 예입니다UNNEST.

```
WITH interpolated_timeseries AS (
    SELECT region, az, vpc, instance_id,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(time, measure_value::double),
                SEQUENCE(ago(6h), now(), 10s)) AS interpolated_cpu_utilization
    FROM timestreamdb.metrics
    WHERE measure_name= 'cpu_utilization' AND time >= ago(6h)
    GROUP BY region, az, vpc, instance_id
)
SELECT region, az, vpc, instance_id, avg(value)
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_cpu_utilization)
GROUP BY region, az, vpc, instance_id
```

시계열 함수

용 Amazon Timestream LiveAnalytics 은 파생, 통합 및 상관 관계와 같은 시간 함수뿐만 아니라 시계열 데이터에서 더 깊은 인사이트를 도출할 수 있도록 지원합니다. 이 섹션에서는 이러한 각 함수에 대한 사용 정보와 샘플 쿼리를 제공합니다. 자세한 내용을 알아보려면 아래에서 주제를 선택하세요.

주제

- 보간 함수
- 파생상품 함수
- 통합 함수
- 상관 함수
- 함수 필터링 및 축소

보간 함수

특정 시점의 이벤트에 대한 시계열 데이터가 누락된 경우 보간을 사용하여 누락된 이벤트의 값을 추정할 수 있습니다. Amazon Timestream은 선형 보간, 입방 스플라인 보간, 마지막 관측값 이월(locf) 보간, 지속적 보간이라는 네 가지 보간 변형을 지원합니다. 이 섹션에서는 보간 함수의 Timestream에 대한 LiveAnalytics 사용 정보와 샘플 쿼리를 제공합니다.

사용 정보

함수	출력 데이터 유형	설명
`interpolate_linear (timeseries, array[timestamp])`	시간 기록	선형 보간을 사용하여 누락된 데이터를 채웁니다.
`interpolate_linear (timeseries, timestamp)`	double	선형 보간을 사용하여 누락된 데이터를 채웁니다.
`interpolate_spline _cubic(timeseries, array[timestamp])`	시간 기록	입방 스플라인 보간을 사용하여 누락된 데이터를 채웁니다.
`interpolate_spline _cubic(timeseries, timestamp)`	double	입방 스플라인 보간을 사용하여 누락된 데이터를 채웁니다.
`interpolate_locf(t imeseries, array[tim estamp])`	시간 기록	마지막 샘플링된 값을 사용하여 누락된 데이터를 채웁니다.
`interpolate_locf(t imeseries, timestamp )`	double	마지막 샘플링된 값을 사용하여 누락된 데이터를 채웁니다.
`interpolate_fill(t imeseries, array[tim estamp], double)`	시간 기록	상수 값을 사용하여 누락된 데이터를 채웁니다.

함수	출력 데이터 유형	설명
`interpolate_fill(timeseries, timestamp, double)`	double	상수 값을 사용하여 누락된 데이터를 채웁니다.

쿼리 예제

Example

지난 2시간 동안 특정 EC2 호스트에 대해 30초 간격으로 binned된 평균 CPU 사용률을 찾고 선형 보간을 사용하여 누락된 값을 채웁니다.

```
WITH binned_timeseries AS (
SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double),
 2) AS avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
SELECT hostname,
    INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
            SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
FROM binned_timeseries
GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

Example

지난 2시간 동안 특정 EC2 호스트에 대해 30초 간격으로 비닝된 평균 CPU 사용률을 찾고 마지막 관측값으로 이월된 값을 기준으로 보간을 사용하여 누락된 값을 채웁니다.

```
WITH binned_timeseries AS (
```

```
SELECT hostname, BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double),
 2) AS avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
SELECT hostname,
    INTERPOLATE_LOCF(
        CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
            SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
FROM binned_timeseries
GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

파생상품 함수

파생상품은 지정된 지표에 대한 변경 속도를 계산하고 이벤트에 선제적으로 응답하는 데 사용할 수 있습니다. 예를 들어, 지난 5분 동안 EC2 인스턴스 CPU 사용률의 파생물을 계산하고 상당한 양의 파생물을 발견했다고 가정해 보겠습니다. 이는 워크로드에 대한 수요 증가를 나타낼 수 있으므로 워크로드를 더 잘 처리하기 위해 더 많은 EC2 인스턴스를 스핀업할 수 있습니다.

Amazon Timestream은 두 가지 파생 함수 변형을 지원합니다. 이 섹션에서는 LiveAnalytics 파생 함수의 Timestream에 대한 사용 정보와 샘플 쿼리를 제공합니다.

사용 정보

함수	출력 데이터 유형	설명
`derivative_linear(timeseries, interval)`	시간 기록	지정된 에 `timeseries` 대한의 각 포인트의 도함수를 계산합니다`interval`.
`non_negative_derivative_linear(timeseries, interval)`	시간 기록	와 동일`derivative_linear(timeserie`

함수	출력 데이터 유형	설명
		s, interval) 하지만 양수 값만 반환합니다.

쿼리 예제

Example

지난 1시간 동안 5분마다 CPU 사용률의 변화율을 찾습니다.

```
SELECT DERIVATIVE_LINEAR(CREATE_TIME_SERIES(time, measure_value::double), 5m) AS
 result
FROM “sampleDB”.DevOps
WHERE measure_name = 'cpu_utilization'
AND hostname = 'host-Hovjv' and time > ago(1h)
GROUP BY hostname, measure_name
```

Example

하나 이상의 마이크로서비스에서 생성된 오류 증가율을 계산합니다.

```
WITH binned_view as (
    SELECT bin(time, 5m) as binned_timestamp, ROUND(AVG(measure_value::double), 2) as
 value
    FROM “sampleDB”.DevOps
    WHERE micro_service = 'jwt'
    AND time > ago(1h)
    AND measure_name = 'service_error'
    GROUP BY bin(time, 5m)
)
SELECT non_negative_derivative_linear(CREATE_TIME_SERIES(binned_timestamp, value), 1m)
 as rateOfErrorIncrease
FROM binned_view
```

통합 함수

적분을 사용하여 시계열 이벤트의 시간 단위당 곡선 아래 영역을 찾을 수 있습니다. 예를 들어, 애플리케이션이 시간 단위당 수신한 요청의 양을 추적한다고 가정해 보겠습니다. 이 시나리오에서는 통합 함수를 사용하여 특정 기간 동안 지정된 간격당 제공된 총 요청량을 확인할 수 있습니다.

Amazon Timestream은 하나의 내장 함수 변형을 지원합니다. 이 섹션에서는 LiveAnalytics 통합 함수를 위한 Timestream의 사용량 정보와 샘플 쿼리를 제공합니다.

사용 정보

함수	출력 데이터 유형	설명
`integral_trapezoidal(timeseries(double))` `integral_trapezoidal(timeseries(double), interval day to second)` `integral_trapezoidal(timeseries(bigint))` `integral_trapezoidal(timeseries(bigint), interval day to second)` `integral_trapezoidal(timeseries(integer), interval day to second)` `integral_trapezoidal(timeseries(integer))`	double	사다리꼴 규칙 을 사용하여 `timeseries` 제공된 `interval day to second`에 대해 지정된 에 따라 적분 를 추정합니다. 요일부터 두 번째까지의 간격 파라미터는 선택 사항이며 기본값은 입니다1s. 간격에 대한 자세한 내용은 섹션을 참조하세요간격 및 기간.

쿼리 예제

Example

특정 호스트가 지난 1시간 동안 5분당 제공한 총 요청 수를 계산합니다.

```
SELECT INTEGRAL_TRAPEZOIDAL(CREATE_TIME_SERIES(time, measure_value::double), 5m) AS
 result FROM sample.DevOps
WHERE measure_name = 'request'
AND hostname = 'host-Hovjv'
AND time > ago (1h)
GROUP BY hostname, measure_name
```

상관 함수

두 개의 유사한 길이 시계열을 고려할 때 상관 함수는 두 시계열이 시간 경과에 따라 어떻게 추세를 보이는지 설명하는 상관 계수를 제공합니다. 상관 계수 범위는 `-1.0` ~입니다`1.0`. `-1.0`는 두 시계열이 동일한 속도로 반대 방향으로 추세를 나타내고, `1.0`는 두 시계열이 동일한 속도로 동일한 방향으로 추세를 나타냅니다. 의 값은 두 시계열 간에 상관관계가 없음을 `0` 나타냅니다. 예를 들어, 오일 가격이 상승하고 석유 회사의 주가가 상승하면 석유 가격 상승 추세와 석유 회사의 가격 상승 추세는 양의 상관계수를 갖습니다. 높은 양의 상관계수는 두 가격이 비슷한 속도로 추세임을 나타냅니다. 마찬가지로 채권 가격과 채권 수익률 간의 상관 계수는 음수이며, 이는 이 두 값이 시간이 지남에 따라 반대 방향으로 추세를 보인다는 것을 나타냅니다.

Amazon Timestream은 두 가지 상관 함수 변형을 지원합니다. 이 섹션에서는 LiveAnalytics 상관관계 함수의 Timestream에 대한 사용량 정보와 샘플 쿼리를 제공합니다.

사용 정보

함수	출력 데이터 유형	설명
`correlate_pearson( timeseries, timeseries)`	double	두 에 대한 Pearson의 상관 계수를 계산합니다`timeseries` . 시간 기록에는 동일한 타임스탬프가 있어야 합니다.
`correlate_spearman (timeseries, timeseries)`	double	두 에 대한 Spearman의 상관 계수를 계산합니다`timeserie`

함수	출력 데이터 유형	설명
		s . 시간 기록에는 동일한 타임스탬프가 있어야 합니다.

쿼리 예제

Example

```
WITH cte_1 AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, measure_value::double),
        SEQUENCE(min(time), max(time), 10m)) AS result
    FROM sample.DevOps
    WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv' AND time > ago(1h)
    GROUP BY hostname, measure_name
),
cte_2 AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, measure_value::double),
        SEQUENCE(min(time), max(time), 10m)) AS result
    FROM sample.DevOps
    WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv' AND time > ago(1h)
    GROUP BY hostname, measure_name
)
SELECT correlate_pearson(cte_1.result, cte_2.result) AS result
FROM cte_1, cte_2
```

함수 필터링 및 축소

Amazon Timestream은 시계열 데이터에 대한 필터를 수행하고 작업을 줄이기 위한 함수를 지원합니다. 이 섹션에서는 LiveAnalytics 필터 및 축소 함수와 샘플 쿼리에 대한 Timestream의 사용 정보를 제공합니다.

사용 정보

함수	출력 데이터 유형	설명
`filter(timeseries(T), function(T, Boolean))`	timeeries(T)	전달된 가 를 function 반환하는 값을 사용하여 입력 시계열에서 시계열을 구성합니다true.
`reduce(timeseries(T), initialState S, inputFunction(S, T, S), outputFunction(S, R))`	R	시계열에서 줄어든 단일 값을 반환합니다. inputFunction 는 각 요소에 대해 시간 순서대로 호출됩니다. 현재 요소를 가져오는 것 외에도 는 현재 상태(최초 initialState)를 inputFunction 가져와서 새 상태를 반환합니다. outputFunction 가 호출되어 최종 상태를 결과 값으로 바꿉니다. 는 자격 증명 함수일 outputFunction 수 있습니다.

쿼리 예제

Example

측정값이 70보다 큰 호스트 및 필터 포인트의 시계열 CPU 사용률을 구성합니다.

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT FILTER(cpu_user, x -> x.value > 70.0) AS cpu_above_threshold
```

```
from time_series_view
```

Example

호스트의 일련의 CPU 사용률을 구성하고 측정값의 제곱 합계를 결정합니다.

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT REDUCE(cpu_user,
    DOUBLE '0.0',
    (s, x) -> x.value * x.value + s,
    s -> s)
from time_series_view
```

Example

호스트의 일련의 CPU 사용률을 구성하고 CPU 임계값을 초과하는 샘플의 비율을 결정합니다.

```
WITH time_series_view AS (
    SELECT INTERPOLATE_LINEAR(
        CREATE_TIME_SERIES(time, ROUND(measure_value::double,2)),
            SEQUENCE(ago(15m), ago(1m), 10s)) AS cpu_user
    FROM sample.DevOps
    WHERE hostname = 'host-Hovjv' and measure_name = 'cpu_utilization'
        AND time > ago(30m)
    GROUP BY hostname
)
SELECT ROUND(
    REDUCE(cpu_user,
      -- initial state
      CAST(ROW(0, 0) AS ROW(count_high BIGINT, count_total BIGINT)),
      -- function to count the total points and points above a certain threshold
      (s, x) -> CAST(ROW(s.count_high + IF(x.value > 70.0, 1, 0), s.count_total + 1) AS
 ROW(count_high BIGINT, count_total BIGINT)),
      -- output function converting the counts to fraction above threshold
      s -> IF(s.count_total = 0, NULL, CAST(s.count_high AS DOUBLE) / s.count_total)),
```

```
    4) AS fraction_cpu_above_threshold
from time_series_view
```

SQL 지원

의 타임스트림은 몇 가지 일반적인 SQL 구성을 LiveAnalytics 지원합니다. 아래에서 자세한 내용을 확인할 수 있습니다.

주제

- SELECT
- 하위 쿼리 지원
- SHOW 문
- DESCRIBE 문
- UNLOAD

SELECT

SELECT 문을 사용하여 하나 이상의 테이블에서 데이터를 검색할 수 있습니다. Timestream의 쿼리 언어는 SELECT 문에 대해 다음 구문을 지원합니다.

```
[ WITH with_query [, ...] ]
            SELECT [ ALL | DISTINCT ] select_expr [, ...]
            [ function (expression) OVER (
            [ PARTITION BY partition_expr_list ]
            [ ORDER BY order_list ]
            [ frame_clause ] )
            [ FROM from_item [, ...] ]
            [ WHERE condition ]
            [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
            [ HAVING condition]
            [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
            [ ORDER BY order_list ]
            [ LIMIT [ count | ALL ] ]
```

여기서 각 항목은 다음과 같습니다.

- `function (expression)` 는 지원되는 윈도우 함수 중 하나입니다.
- `partition_expr_list`는

```
expression | column_name [, expr_list ]
```

- order_list는

```
expression | column_name [ ASC | DESC ]
[ NULLS FIRST | NULLS LAST ]
[, order_list ]
```

- frame_clause는

```
ROWS | RANGE
{ UNBOUNDED PRECEDING | expression PRECEDING | CURRENT ROW } |
{BETWEEN
{ UNBOUNDED PRECEDING | expression { PRECEDING | FOLLOWING } |
CURRENT ROW}
AND
{ UNBOUNDED FOLLOWING | expression { PRECEDING | FOLLOWING } |
CURRENT ROW }}
```

- from_item 는 다음 중 하나입니다.

```
table_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
from_item join_type from_item [ ON join_condition | USING ( join_column [, ...] ) ]
```

- join_type 는 다음 중 하나입니다.

```
[ INNER ] JOIN
LEFT [ OUTER ] JOIN
RIGHT [ OUTER ] JOIN
FULL [ OUTER ] JOIN
```

- grouping_element 는 다음 중 하나입니다.

```
()
expression
```

하위 쿼리 지원

Timestream은 EXISTS 및 IN 조건의 하위 쿼리를 지원합니다. EXISTS 조건부로 하위 쿼리가 행을 반환할지 여부가 결정됩니다. IN 조건부는 하위 쿼리에서 생성된 값이 IN 절의 값 또는 표현식과 일치하는지 여부를 결정합니다. Timestream 쿼리 언어는 상관된 하위 쿼리 및 기타 하위 쿼리를 지원합니다.

```
SELECT t.c1
FROM (VALUES 1, 2, 3, 4, 5) AS t(c1)
WHERE EXISTS
(SELECT t.c2
 FROM (VALUES 1, 2, 3) AS t(c2)
 WHERE t.c1= t.c2
)
ORDER BY t.c1
```

c1
1
2
3

```
SELECT t.c1
FROM (VALUES 1, 2, 3, 4, 5) AS t(c1)
WHERE t.c1 IN
(SELECT t.c2
 FROM (VALUES 2, 3, 4) AS t(c2)
)
ORDER BY t.c1
```

c1
2
3
4

SHOW 문

SHOW DATABASES 문을 사용하여 계정의 모든 데이터베이스를 볼 수 있습니다. 구문은 다음과 같습니다.

```
SHOW DATABASES [LIKE pattern]
```

여기서 LIKE 절을 사용하여 데이터베이스 이름을 필터링할 수 있습니다.

SHOW TABLES 문을 사용하여 계정의 모든 테이블을 볼 수 있습니다. 구문은 다음과 같습니다.

```
SHOW TABLES [FROM database] [LIKE pattern]
```

여기서 FROM 절은 데이터베이스 이름을 필터링하는 데 사용할 수 있고 LIKE 절은 테이블 이름을 필터링하는 데 사용할 수 있습니다.

SHOW MEASURES 문을 사용하여 테이블에 대한 모든 측정값을 볼 수 있습니다. 구문은 다음과 같습니다.

```
SHOW MEASURES FROM database.table [LIKE pattern]
```

여기서 FROM 절은 데이터베이스 및 테이블 이름을 지정하는 데 사용되고 LIKE 절은 측정값 이름을 필터링하는 데 사용될 수 있습니다.

DESCRIBE 문

DESCRIBE 문을 사용하여 테이블의 메타데이터를 볼 수 있습니다. 구문은 다음과 같습니다.

```
DESCRIBE database.table
```

여기서 는 테이블 이름을 table 포함합니다. 설명 문은 테이블의 열 이름과 데이터 유형을 반환합니다.

UNLOAD

용 Timestream LiveAnalytics 은 해당 지원의 확장으로 UNLOAD 명령을 SQL 지원합니다. 에서 지원하는 데이터 유형은 에 UNLOAD 설명되어 있습니다지원되는 데이터 유형. time 및 unknown 유형은 에 적용되지 않습니다UNLOAD.

```
UNLOAD (SELECT statement)
 TO 's3://bucket-name/folder'
```

```
 WITH ( option = expression [, ...] )
```

여기서 옵션은 입니다.

```
{ partitioned_by = ARRAY[ col_name[,…] ]
 | format = [ '{ CSV | PARQUET }' ]
 | compression = [ '{ GZIP | NONE }' ]
 | encryption = [ '{ SSE_KMS | SSE_S3 }' ]
 | kms_key = '<string>'
 | field_delimiter ='<character>'
 | escaped_by = '<character>'
 | include_header = ['{true, false}']
 | max_file_size = '<value>'
}
```

SELECT 문

LiveAnalytics 테이블에 대한 하나 이상의 Timestream에서 데이터를 선택하고 검색하는 데 사용되는 쿼리 문입니다.

```
(SELECT column 1, column 2, column 3 from database.table
      where measure_name = "ABC" and timestamp between ago (1d) and now() )
```

TO 절

```
TO 's3://bucket-name/folder'
```

또는

```
TO 's3://access-point-alias/folder'
```

UNLOAD 문에 있는 TO 절은 쿼리 결과의 출력 대상을 지정합니다. Timestream for LiveAnalytics 가 출력 파일 객체를 쓰는 Amazon S3 버킷 이름 또는 Amazon S3 access-point-alias에 폴더 위치가 있는 Amazon S3를 포함한 전체 경로를 제공해야 합니다. S3 버킷은 동일한 계정과 동일한 리전에서 소유해야 합니다. 쿼리 결과 세트 외에도 Timestream for 는 매니페스트 및 메타데이터 파일을 지정된 대상 폴더에 LiveAnalytics 씁니다.

PARTITIONED_BY 절

```
partitioned_by = ARRAY [col_name[,…] , (default: none)
```

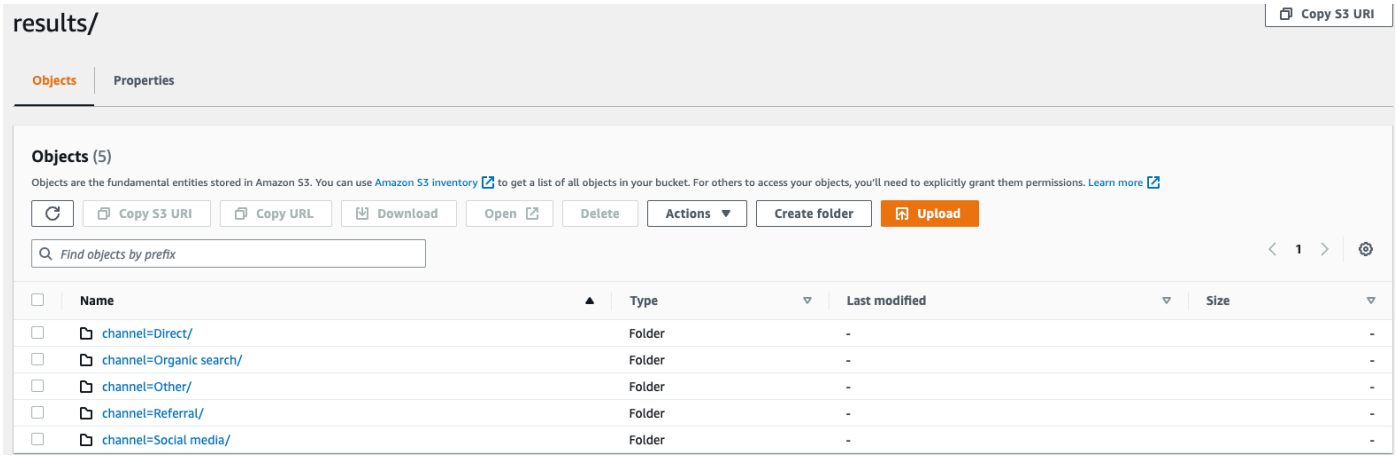
이 partitioned_by 절은 쿼리에서 세분화된 수준에서 데이터를 그룹화하고 분석하는 데 사용됩니다. 쿼리 결과를 S3 버킷으로 내보낼 때 선택 쿼리의 하나 이상의 열을 기반으로 데이터를 분할하도록 선택할 수 있습니다. 데이터를 분할할 때 내보낸 데이터는 파티션 열을 기반으로 하위 집합으로 분할되고 각 하위 집합은 별도의 폴더에 저장됩니다. 내보낸 데이터가 포함된 결과 폴더 내에 하위 폴더folder/results/partition column = partition value/가 자동으로 생성됩니다. 하지만 파티션된 열은 출력 파일에 포함되지 않습니다.

partitioned_by 는 구문의 필수 절이 아닙니다. 파티셔닝 없이 데이터를 내보내도록 선택한 경우 구문에서 절을 제외할 수 있습니다.

Example

웹 사이트의 클릭스트림 데이터를 모니터링하고 direct, , , , 라는 5개의 트래픽 채널이 있다고 가정합니다Social MediaOrganic SearchOtherReferral. 데이터를 내보낼 때 열을 사용하여 데이터를 분할하도록 선택할 수 있습니다Channel. 데이터 폴더인 s3://bucketname/results에는 각각 해당 채널 이름이 있는 폴더가 5개 있습니다. 예를 들어 이 폴더s3://bucketname/results/channel=Social Media/.에서는 Social Media 채널을 통해 웹 사이트에 도착한 모든 고객의 데이터를 찾을 수 있습니다. 마찬가지로 나머지 채널에 대한 다른 폴더도 있습니다.

채널 열로 분할된 내보낸 데이터



FORMAT

```
format = [ '{ CSV | PARQUET }' , default: CSV
```

S3 버킷에 기록된 쿼리 결과의 형식을 지정하는 키워드입니다. 쉼표(,CSV)를 기본 구분 기호로 사용하거나 분석을 위한 효율적인 오픈 열 스토리지 형식인 Apache Parquet 형식으로 데이터를 쉼표로 구분된 값()으로 내보낼 수 있습니다.

COMPRESSION

```
compression = [ '{ GZIP | NONE }' ], default: GZIP
```

압축 알고리즘을 사용하여 내보낸 데이터를 압축GZIP하거나 NONE 옵션을 지정하여 압축을 해제할 수 있습니다.

ENCRYPTION

```
encryption = [ '{ SSE_KMS | SSE_S3 }' ], default: SSE_S3
```

Amazon S3의 출력 파일은 선택한 암호화 옵션을 사용하여 암호화됩니다. 데이터 외에도 매니페스트 및 메타데이터 파일도 선택한 암호화 옵션에 따라 암호화됩니다. 현재 SSE_S3 및 SSE_KMS 암호화를 지원합니다. SSE_S3는 Amazon S3가 256비트 고급 암호화 표준(AES) 암호화를 사용하여 데이터를 암호화하는 서버 측 암호화입니다. SSE_KMS는 고객 관리형 키를 사용하여 데이터를 암호화하는 서버 측 암호화입니다.

KMS_KEY

```
kms_key = '<string>'
```

KMS 키는 내보낸 쿼리 결과를 암호화하는 고객 정의 키입니다. KMS 키는 AWS Key Management Service(AWS KMS)에서 안전하게 관리되며 Amazon S3의 데이터 파일을 암호화하는 데 사용됩니다.

FIELD_DELIMITER

```
field_delimiter ='<character>' , default: (,)
```

CSV 형식으로 데이터를 내보낼 때 이 필드는 파이프 ASCII 문자(|), 쉼표(,) 또는 탭(/t)과 같이 출력 파일의 필드를 구분하는 데 사용되는 단일 문자를 지정합니다. CSV 파일의 기본 구분 기호는 쉼표입니다. 데이터의 값에 선택한 구분 기호가 포함된 경우 구분 기호는 따옴표 문자로 따옴표로 표시됩니다. 예를 들어 데이터의 값에 가 포함된 경우 `Time,stream`이 값은 내보낸 데이터`"Time,stream"`에서와 같이 인용됩니다. 에 대해 Timestream에서 사용하는 따옴표 문자 LiveAnalytics 는 큰따옴표(')입니다.

에 헤더를 포함하려는 FIELD_DELIMITER 경우 캐리지 리턴 문자(ASCII 13, 16진수`0D`, 텍스트 '\r') 또는 줄 바꿈 문자(ASCII 10, 16진수 0A, 텍스트 '\n')를 로 지정하지 마세요. 이렇게 하면 많은 파서가 결과 CSV 출력에 헤더를 올바르게 구문 분석할 수 없기 CSV때문입니다.

ESCAPED_BY

```
escaped_by = '<character>', default: (\)
```

데이터를 CSV 형식으로 내보낼 때 이 필드는 S3 버킷에 기록된 데이터 파일에서 이스케이프 문자로 처리해야 하는 문자를 지정합니다. Escaping은 다음 시나리오에서 발생합니다.

1. 값 자체에 따옴표 문자(')가 포함된 경우 이스케이프 문자를 사용하여 이스케이프됩니다. 예를 들어 값이 이고 Time"stream(\)가 구성된 이스케이프 문자인 경우 값이 로 이스케이프됩니다Time\"stream.
2. 값에 구성된 이스케이프 문자가 포함된 경우 이스케이프됩니다. 예를 들어 값이 인 경우 값이 로 이스케이프Time\stream됩니다Time\\stream.

> **Note**
>
> 내보낸 출력에 배열, 행 또는 시계열과 같은 복잡한 데이터 유형이 포함된 경우 JSON 문자열로 직렬화됩니다. 다음은 한 예입니다.

데이터 유형	실제 값	값이 [직렬화된 JSON 문자열] CSV 형식으로 이스케이프되는 방법
배열	`[ 23,24,25 ]`	`"[23,24,25]"`
열	`( x=23.0, y=hello )`	`"{\"x\":23.0,\"y\":\"hello\"}"`
시계열	`[ ( time=1970-01-01 00:00:00.000000010 , value=100.0 ), ( time=1970-01-01 00:00:00.000000012, value=120.0 ) ]`	`"[{\"time\":\"1970-01-01 00:00:00.000000010Z\",\"value\":100.0},{\"time\":\"1970-01-01 00:00:00.000000012Z\",\"value\":120.0}]"`

INCLUDE_HEADER

```
include_header = 'true' , default: 'false'
```

데이터를 CSV 형식으로 내보낼 때 이 필드를 사용하면 내보낸 CSV 데이터 파일의 첫 번째 행에 열 이름을 포함할 수 있습니다.

허용되는 값은 'true'와 'false'이고 기본값은 'false'입니다. `escaped_by` 및 와 같은 텍스트 변환 옵션은 헤더에도 `field_delimiter` 적용됩니다.

> **Note**
>
> 헤더를 포함할 때는 캐리지 리턴 문자(ASCII 13, 16진수 0D , 텍스트 '\r') 또는 줄 바꿈 문자(ASCII 10, 16진수 0A , 텍스트 '\n')를 로 선택하지 않는 것이 중요합니다. 이렇게 하면 많은 파서가 결과 CSV 출력에서 헤더를 올바르게 구문 분석할 수 없기 `FIELD_DELIMITER`때문입니다.

MAX_FILE_SIZE

```
max_file_size = 'X[MB|GB]' , default: '78GB'
```

이 필드는 UNLOAD 명령문이 Amazon S3에서 생성하는 파일의 최대 크기를 지정합니다. UNLOAD 문은 여러 파일을 생성할 수 있지만 Amazon S3에 기록된 각 파일의 최대 크기는 이 필드에 지정된 것과 비슷합니다.

필드 값은 16MB~78GB여야 합니다. 와 같은 정수로 지정`12GB`하거나 또는 와 같은 소수로 지정할 수 있습니다`0.5GB24.7MB`. 기본값은 78GB입니다.

실제 파일 크기는 파일을 작성할 때 근사치이므로 실제 최대 크기는 지정한 수와 정확히 같지 않을 수 있습니다.

논리 연산자

에 대한 Timestream은 다음과 같은 논리 연산자를 LiveAnalytics 지원합니다.

연산자	설명	예
AND	두 값이 모두 true인 경우 True	a AND b

연산자	설명	예
OR	두 값 중 하나가 true인 경우 True	a 또는 b
NOT	값이 false인 경우 True	NOT a

- 표현식의 한쪽 또는 양쪽이 인 NULL 경우 AND 비교 결과가 나올 수 있습니다NULL.
- AND 연산자의 한 면 이상이 인 경우 표현식FALSE은 로 평가됩니다FALSE.
- 표현식의 한쪽 또는 양쪽이 인 NULL 경우 OR 비교 결과가 나올 수 있습니다NULL.
- OR 연산자의 한 면 이상이 인 경우 표현TRUE식은 로 평가합니다TRUE.
- 의 논리적 보완은 NULL입니다NULL.

다음 진실 표는 AND 및 NULL에서 의 처리를 보여줍니다OR.

A	B	A 및 b	A 또는 b
null	null	null	null
false	null	false	null
null	false	false	null
true	null	null	true
null	true	null	true
false	false	false	false
true	false	false	true
false	true	false	true
true	true	true	true

다음 진실 표는 NULL에서 의 처리를 보여줍니다NOT.

A	가 아님
null	null
true	false
false	true

비교 연산자

의 Timestream은 다음 비교 연산자를 LiveAnalytics 지원합니다.

연산자	설명
<	보다 작음
>	보다 큼
<=	작거나 같음
>=	크거나 같음
=	같음
<>	같지 않음
!=	같지 않음

Note

- BETWEEN 연산자는 값이 지정된 범위 내에 있는지 테스트합니다. 구문은 다음과 같습니다.

  ```
  BETWEEN min AND max
  ```

  BETWEEN 또는 NOT BETWEEN 문NULL에 가 있으면 문이 로 평가됩니다NULL.
- IS NULL 및 IS NOT NULL 연산자는 값이 null(정의되지 않음)인지 테스트합니다. 와 NULL 함께 를 사용하면 가 true로 IS NULL 평가됩니다.

- 에서 SQL NULL 값은 알 수 없는 값을 나타냅니다.

비교 함수

의 Timestream은 다음 비교 함수를 LiveAnalytics 지원합니다.

주제

가장 큼()

가장 큼() 함수는 제공된 값 중 가장 큰 값을 반환합니다. 제공된 값 중 하나가 NULL이면 반환됩니다NULL. 구문은 다음과 같습니다.

```
greatest(value1, value2, ..., valueN)
```

최소()

최소() 함수는 제공된 값 중 가장 작은 값을 반환합니다. 제공된 값 중 하나가 NULL이면 반환됩니다NULL. 구문은 다음과 같습니다.

```
least(value1, value2, ..., valueN)
```

ALL(), ANY() 및 SOME()

ALL, ANY 및 SOME 퀀타이즈는 다음과 같은 방법으로 비교 연산자와 함께 사용할 수 있습니다.

표현식	의미
A = ALL(...)	A가 모든 값과 같을 때 true로 평가됩니다.
A <> ALL(...)	A가 값과 일치하지 않을 때 true로 평가됩니다.
A < ALL(...)	A가 최소 값보다 작을 때 true로 평가됩니다.

표현식	의미
A = ANY(...)	A가 값 중 하나와 같을 때 true로 평가됩니다.
A <> ANY(...)	A가 하나 이상의 값과 일치하지 않을 때 true로 평가됩니다.
A < ANY(...)	A가 가장 큰 값보다 작을 때 true로 평가됩니다.

예제 및 사용 정보

> **Note**
>
> ALL, ANY 또는 SOME를 사용할 때는 비교 값이 리터럴 목록인 경우 키워드를 사용해야 VALUES 합니다.

예시: **ANY()**

다음과 같은 쿼리 문ANY()의 예입니다.

```
SELECT 11.7 = ANY (VALUES 12.0, 13.5, 11.7)
```

동일한 작업에 대한 대체 구문은 다음과 같습니다.

```
SELECT 11.7 = ANY (SELECT 12.0 UNION ALL SELECT 13.5 UNION ALL SELECT 11.7)
```

이 경우 는 를 ANY() 평가합니다True.

예시: **ALL()**

다음과 같은 쿼리 문ALL()의 예입니다.

```
SELECT 17 < ALL (VALUES 19, 20, 15);
```

동일한 작업에 대한 대체 구문은 다음과 같습니다.

```
SELECT 17 < ALL (SELECT 19 UNION ALL SELECT 20 UNION ALL SELECT 15);
```

이 경우 는 를 ALL() 평가합니다False.

예시: **SOME()**

다음과 같은 쿼리 문SOME()의 예입니다.

```
SELECT 50 >= SOME (VALUES 53, 77, 27);
```

동일한 작업에 대한 대체 구문은 다음과 같습니다.

```
SELECT 50 >= SOME (SELECT 53 UNION ALL SELECT 77 UNION ALL SELECT 27);
```

이 경우 는 를 SOME() 평가합니다True.

조건식

에 대한 Timestream은 다음과 같은 조건식 표현식을 LiveAnalytics 지원합니다.

주제

- CASE 문
- IF 문
- COALESCE 문
- NULLIF 문
- TRY 문

CASE 문

CASE 문은 와 같은 값을 찾을 때까지 왼쪽에서 오른쪽으로 각 값 표현식을 검색합니다expression. 일치하는 항목을 찾으면 일치하는 값에 대한 결과가 반환됩니다. 일치하는 항목이 없으면 ELSE 절의 결과가 있는 경우 반환되고, 그렇지 않으면 null가 반환됩니다. 구문은 다음과 같습니다.

```
CASE expression
    WHEN value THEN result
    [ WHEN ... ]
    [ ELSE result ]
END
```

또한 Timestream은 CASE 문에 대해 다음 구문을 지원합니다. 이 구문에서 “searched” 양식은 왼쪽에서 오른쪽으로 각 부울 조건을 평가하고 일치하는 결과를 true 반환합니다. 조건이 없는 경우 ELSE 절true의 결과가 있는 경우 반환되고 그렇지 않은 경우 반환null됩니다. 대체 구문은 아래를 참조하세요.

```
CASE
    WHEN condition THEN result
    [ WHEN ... ]
    [ ELSE result ]
END
```

IF 문

IF 문은 조건을 true 또는 false로 평가하고 적절한 값을 반환합니다. Timestream은 IF 에 대해 다음 두 가지 구문 표현을 지원합니다.

```
if(condition, true_value)
```

이 구문은 조건이 인 true_value 경우 평가하고 반환합니다. true그렇지 않으면 null 가 반환되고 평가true_value되지 않습니다.

```
if(condition, true_value, false_value)
```

이 구문은 조건이 인 true_value 경우 를 평가하고 반환하며true, 그렇지 않으면 를 평가하고 반환합니다false_value.

예시

```
SELECT
  if(true, 'example 1'),
  if(false, 'example 2'),
  if(true, 'example 3 true', 'example 3 false'),
  if(false, 'example 4 true', 'example 4 false')
```

_col0	_col1	_col2	_col3
example 1	- null	example 3 true	example 4 false

COALESCE 문

COALESCE 는 인수 목록의 첫 번째 null이 아닌 값을 반환합니다. 구문은 다음과 같습니다.

```
coalesce(value1, value2[,...])
```

NULLIF 문

IF 문은 조건을 true 또는 false로 평가하고 적절한 값을 반환합니다. Timestream은 IF 에 대해 다음 두 가지 구문 표현을 지원합니다.

NULLIF `value1`가 이면 는 null을 반환하고`value2`, 그렇지 않으면 를 반환합니다`value1`. 구문은 다음과 같습니다.

```
nullif(value1, value2)
```

TRY 문

TRY 함수는 표현식을 평가하고 를 반환하여 특정 유형의 오류를 처리합니다`null`. 구문은 다음과 같습니다.

```
try(expression)
```

변환 함수

의 Timestream은 다음과 같은 변환 함수를 LiveAnalytics 지원합니다.

주제

- cast()
- try_cast()

cast()

값을 유형으로 명시적으로 캐스팅하는 캐스트 함수의 구문은 다음과 같습니다.

```
cast(value AS type)
```

try_cast()

LiveAnalytics 또한 에 대한 Timestream은 캐스트와 유사한 try_cast 함수를 지원하지만 캐스트가 실패하면 null을 반환합니다. 구문은 다음과 같습니다.

```
try_cast(value AS type)
```

수학적 연산

의 Timestream은 다음과 같은 수학 연산자를 LiveAnalytics 지원합니다.

연산자	설명
+	Addition
-	뺄셈
*	곱셈
/	디비전(정수 디비전이 잘림 수행)
%	모듈러스(나머지)

수학 함수

의 Timestream은 다음과 같은 수학적 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
abs(x)	[입력과 동일]	x의 절대값을 반환합니다.
cbrt(x)	double	x의 큐브 루트를 반환합니다.
천장(x) 또는 ceil(x)	[입력과 동일]	x를 가장 가까운 정수로 반올림하여 반환합니다.
도(x)	double	각도 x를 라디안 단위로 각도로 변환합니다.

함수	출력 데이터 유형	설명
e()	double	상수 Euler의 번호를 반환합니다.
exp(x)	double	제기된 Euler의 번호를 x의 힘으로 반환합니다.
층(x)	[입력과 동일]	x를 가장 가까운 정수로 반올림하여 반환합니다.
from_base(문자열,radix)	bigint	기본 라픽스 숫자로 해석된 문자열 값을 반환합니다.
ln(x)	double	x의 자연 로그를 반환합니다.
log2(x)	double	x의 기본 2 로그를 반환합니다.
log10(x)	double	x의 기본 10 로그를 반환합니다.
모드(n,m)	[입력과 동일]	n의 모듈러스(나머지)를 m으로 나눈 값으로 반환합니다.
pi()	double	상수 Pi를 반환합니다.
pow(x, p) 또는 power(x, p)	double	p의 출력으로 x를 반환합니다.
라디안(x)	double	각도 x를 라디안으로 변환합니다.
rand() 또는 random()	double	0.0 1.0 범위의 의사 무작위 값을 반환합니다.
random(n)	[입력과 동일]	0에서 n(독점) 사이의 의사 무작위 번호를 반환합니다.
round(x)	[입력과 동일]	x를 가장 가까운 정수로 반올림하여 반환합니다.

함수	출력 데이터 유형	설명
round(x,d)	[입력과 동일]	x를 소수점 이하 자릿수로 반올림하여 반환합니다.
sign(x)	[입력과 동일]	x의 서명 함수를 반환합니다. 즉, • 인수가 0인 경우 0 • 인수가 0보다 큰 경우 1 • 인수가 0 미만인 경우 -1입니다. 이중 인수의 경우 함수는 다음을 추가로 반환합니다. • 인수가 NaN인 경우 NaN • 인수가 +Infinity인 경우 1 • 인수가 -Infinity인 경우 -1입니다.
sqrt(x)	double	x의 제곱근을 반환합니다.
to_base(x, radix)	varchar	x의 기본-반사 표현을 반환합니다.
잘라내기(x)	double	소수점 뒤에 숫자를 삭제하여 x를 정수로 반올림하여 반환합니다.
acos(x)	double	x의 아크 코사인을 반환합니다.
asin(x)	double	x의 아크 사인을 반환합니다.
atan(x)	double	x의 아크 탄젠트를 반환합니다.
atan2(y, x)	double	y/x의 아크 탄젠트를 반환합니다.

함수	출력 데이터 유형	설명
cos(x)	double	x의 코사인을 반환합니다.
cosh(x)	double	x의 하이퍼볼릭 코사인을 반환합니다.
sin(x)	double	x의 사인을 반환합니다.
tan(x)	double	x의 탄젠트를 반환합니다.
tanh(x)	double	x의 하이퍼볼릭 탄젠트를 반환합니다.
무한대()	double	양의 무한성을 나타내는 상수를 반환합니다.
is_finite(x)	boolean	x가 유한한지 확인합니다.
is_infinite(x)	boolean	x가 무한인지 확인합니다.
is_nan(x)	boolean	x가 인지 확인합니다 not-a-number.
nan()	double	를 나타내는 상수를 반환합니다 not-a-number.

문자열 연산자

의 Timestream은 하나 이상의 문자열을 연결하기 위해 \|\| 연산자를 LiveAnalytics 지원합니다.

문자열 함수

> **Note**
>
> 달리 지정하지 않는 한 이러한 함수의 입력 데이터 유형은 varchar로 간주됩니다.

함수	출력 데이터 유형	설명
chr(n)	varchar	유니코드 코드 포인트 n을 varchar로 반환합니다.
codepoint(x)	정수	str의 유일한 문자의 유니코드 코드 포인트를 반환합니다.
concat(x1, ..., xN)	varchar	x1, x2, ..., xN의 연결을 반환합니다.
hamming_distance(x1,x2)	bigint	x1 및 x2의 해밍 거리, 즉 해당 문자가 다른 위치 수를 반환합니다. 두 개의 varchar 입력의 길이는 같아야 합니다.
길이(x)	bigint	x의 길이를 문자로 반환합니다.
levenshtein_distance(x1, x2)	bigint	x1 및 x2의 Levenshtein 편집 거리, 즉 x1을 x2로 변경하는 데 필요한 단일 문자 편집(삽입, 삭제 또는 대체)의 최소 수를 반환합니다.
lower(x)	varchar	x를 소문자로 변환합니다.
lpad(x1, bigint 크기, x2)	varchar	왼쪽 패드 x1을 사용하여 x2로 문자 크기를 조정합니다. 크기가 x1 길이보다 작으면 결과는 문자 크기로 잘립니다. 크기는 음수일 수 없으며 x2는 비워둘 수 없습니다.
ltrim(x)	varchar	x에서 선행 공백을 제거합니다.
교체(x1, x2)	varchar	x1에서 x2의 모든 인스턴스를 제거합니다.

함수	출력 데이터 유형	설명
교체(x1, x2, x3)	varchar	x2의 모든 인스턴스를 x1의 x3으로 바꿉니다.
역방향(x)	varchar	문자와 함께 x를 역순으로 반환합니다.
rpad(x1, bigint 크기, x2)	varchar	x2로 문자 크기를 조정하려면 x1을 오른쪽 패드로 설정합니다. 크기가 x1 길이보다 작으면 결과는 문자 크기로 잘립니다. 크기는 음수일 수 없으며 x2는 비워둘 수 없습니다.
rtrim(x)	varchar	x에서 후행 공백을 제거합니다.
split(x1, x2)	array(varchar)	구분자 x2의 x1을 분할하고 배열을 반환합니다.
split(x1, x2, bigint 제한)	array(varchar)	구분자 x2의 x1을 분할하고 배열을 반환합니다. 배열의 마지막 요소에는 항상 x1에 남아 있는 모든 항목이 포함됩니다. 제한은 양수여야 합니다.
split_part(x1, x2, bigint pos)	varchar	구분자 x2의 x1을 분할하고 위치에 있는 varchar 필드를 반환합니다. 필드 인덱스는 1로 시작합니다. pos가 필드 수보다 크면 null이 반환됩니다.
strpos(x1, x2)	bigint	x2의 첫 번째 인스턴스의 시작 위치를 x1로 반환합니다. 위치는 1로 시작합니다. 찾을 수 없는 경우 0이 반환됩니다.

함수	출력 데이터 유형	설명
strpos(x1, x2,bigint 인스턴스)	bigint	x2의 N번째 인스턴스 위치를 x1 단위로 반환합니다. 인스턴스는 양수여야 합니다. 위치는 1로 시작합니다. 찾을 수 없는 경우 0이 반환됩니다.
strrpos(x1, x2)	bigint	x2의 마지막 인스턴스의 시작 위치를 x1로 반환합니다. 위치는 1로 시작합니다. 찾을 수 없는 경우 0이 반환됩니다.
strrpos(x1, x2, bigint 인스턴스)	bigint	x1 끝부터 x2의 N번째 인스턴스 위치를 x1로 반환합니다. 인스턴스는 양수여야 합니다. 위치는 1로 시작합니다. 찾을 수 없는 경우 0이 반환됩니다.
위치(x2 IN x1)	bigint	x2의 첫 번째 인스턴스의 시작 위치를 x1로 반환합니다. 위치는 1로 시작합니다. 찾을 수 없는 경우 0이 반환됩니다.
substr(x, bigint 시작)	varchar	시작 위치 시작에서 나머지 x를 반환합니다. 위치는 1로 시작합니다. 음의 시작 위치는 x의 끝을 기준으로 해석됩니다.
substr(x, bigint start, bigint len)	varchar	시작 위치 시작에서 길이 길이 길이 x의 하위 문자열을 반환합니다. 위치는 1로 시작합니다. 음의 시작 위치는 x의 끝을 기준으로 해석됩니다.
트림(x)	varchar	x에서 선행 및 후행 공백을 제거합니다.

함수	출력 데이터 유형	설명
upper(x)	varchar	x를 대문자로 변환합니다.

배열 연산자

의 Timestream은 다음 배열 연산자를 LiveAnalytics 지원합니다.

연산자	설명
[]	첫 번째 인덱스가 1에서 시작하는 배열의 요소에 액세스합니다.
||	배열을 동일한 유형의 다른 배열 또는 요소와 연결합니다.

배열 함수

의 Timestream은 다음 배열 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
array_distinct(x)	array	배열 x에서 중복 값을 제거합니다. `SELECT array_distinct(ARRAY[1,2,2,3])` 예제 결과: `[ 1,2,3 ]`
array_intersect(x, y)	array	중복 없이 x와 y의 교차점에 있는 요소 배열을 반환합니다. `SELECT array_intersect(ARRAY[1,2,3], ARRAY[3,4,5])`

함수	출력 데이터 유형	설명
		예제 결과: `[ 3 ]`
array_union(x, y)	array	중복 없이 x와 y의 조합에 있는 요소 배열을 반환합니다. `SELECT array_union(ARRAY[1,2,3], ARRAY[3,4,5])` 예제 결과: `[ 1,2,3,4,5 ]`
array_except(x, y)	array	중복 없이 x로 요소 배열을 반환하지만 y로 반환하지 않습니다. `SELECT array_except(ARRAY[1,2,3], ARRAY[3,4,5])` 예제 결과: `[ 1,2 ]`
array_join(x, 구분 기호, null_replacement)	varchar	null을 대체하는 구분 기호와 선택적 문자열을 사용하여 지정된 배열의 요소를 연결합니다. `SELECT array_join(ARRAY[1,2,3], ';', '')` 예제 결과: `1;2;3`

함수	출력 데이터 유형	설명
array_max(x)	배열 요소와 동일	입력 배열의 최대값을 반환합니다. `SELECT array_max (ARRAY[1,2,3])` 예제 결과: 3
array_min(x)	배열 요소와 동일	입력 배열의 최소값을 반환합니다. `SELECT array_min (ARRAY[1,2,3])` 예제 결과: 1
array_position(x, 요소)	bigint	배열 x(또는 찾을 수 없는 경우 0)에서 요소가 처음 발생한 위치를 반환합니다. `SELECT array_pos ition(ARRAY[3,4,5,9], 5)` 예제 결과: 3
array_remove(x, 요소)	array	배열 x에서 동일한 요소를 모두 제거합니다. `SELECT array_rem ove(ARRAY[3,4,5,9], 4)` 예제 결과: `[ 3,5,9 ]`

함수	출력 데이터 유형	설명
array_sort(x)	array	배열 x를 정렬하고 반환합니다. x의 요소는 주문 가능해야 합니다. Null 요소는 반환된 배열의 끝에 배치됩니다. `SELECT array_sort(ARRAY[6,8,2,9,3])` 예제 결과: `[ 2,3,6,8,9 ]`
arrays_overlap(x, y)	boolean	x 및 y 배열에 공통된 null이 아닌 요소가 있는지 테스트합니다. 공통적으로 null이 아닌 요소가 없지만 배열 중 하나에 null이 포함된 경우 null을 반환합니다. `SELECT arrays_overlap(ARRAY[6,8,2,9,3], ARRAY[6,8])` 예제 결과: `true`
카디널리티(x)	bigint	배열 x의 크기를 반환합니다. `SELECT cardinality(ARRAY[6,8,2,9,3])` 예제 결과: 5

함수	출력 데이터 유형	설명
concat(array1, array2, ..., arrayN)	array	배열 array1, array2, ..., arrayN을 연결합니다. `SELECT concat(ARRAY[6,8,2,9,3], ARRAY[11,32], ARRAY[6,8,2,0,14])` 예제 결과: `[ 6,8,2,9,3,11,32,6,8,2,0,14 ]`
element_at(array(E), 인덱스)	E	지정된 인덱스에서 배열 요소를 반환합니다. 인덱스가 < 0인 경우 element_at는 마지막에서 첫 번째로 요소에 액세스합니다. `SELECT element_at(ARRAY[6,8,2,9,3], 1)` 예제 결과: 6
repeat(요소, 개수)	array	계수 시간에 대해 요소를 반복합니다. `SELECT repeat(1, 3)` 예제 결과: `[ 1,1,1 ]`

함수	출력 데이터 유형	설명
reverse(x)	array	배열 x의 역순으로 배열을 반환합니다. `SELECT reverse(ARRAY[6,8,2,9,3])` 예제 결과: `[ 3,9,2,8,6 ]`
시퀀스(시작, 중지)	array(bigint)	시작부터 중지까지 정수 시퀀스를 생성하여 시작이 중지보다 작거나 같으면 1씩 증가시키고, 그렇지 않으면 -1씩 증가시킵니다. `SELECT sequence(3, 8)` 예제 결과: `[ 3,4,5,6,7,8 ]`
시퀀스(시작, 중지, 단계)	array(bigint)	시작부터 중지까지 단계적으로 증가하는 정수 시퀀스를 생성합니다. `SELECT sequence(3, 15, 2)` 예제 결과: `[ 3,5,7,9,11,13,15 ]`

함수	출력 데이터 유형	설명
시퀀스(시작, 중지)	array(타임스탬프)	시작 날짜부터 중지 날짜까지 1일씩 증가하는 일련의 타임스탬프를 생성합니다. `SELECT sequence( '2023-04-02 19:26:12.941000000', '2023-04-06 19:26:12.941000000', 1d)` 예제 결과: `[ 2023-04-02 19:26:12.941000000 ,2023-04-03 19:26:12.941000000 ,2023-04-04 19:26:12.941000000 ,2023-04-05 19:26:12.941000000 ,2023-04-06 19:26:12.941000000 ]`

함수	출력 데이터 유형	설명
시퀀스(시작, 중지, 단계)	array(타임스탬프)	시작부터 중지까지 단계별로 타임스탬프 시퀀스를 생성합니다. 단계의 데이터 유형은 간격입니다. `SELECT sequence( '2023-04-02 19:26:12.941000000', '2023-04-10 19:26:12.941000000', 2d)` 예제 결과: `[ 2023-04-02 19:26:12.941000000 ,2023-04-04 19:26:12.941000000 ,2023-04-06 19:26:12.941000000 ,2023-04-08 19:26:12.941000000 ,2023-04-10 19:26:12.941000000 ]`
셔플(x)	array	지정된 배열 x의 임의 순열을 생성합니다. `SELECT shuffle(ARRAY[6,8,2,9,3])` 예제 결과: `[ 6,3,2,9,8 ]`

함수	출력 데이터 유형	설명
slice(x, 시작, 길이)	array	길이가 긴 인덱스 시작부터(또는 시작이 음수인 경우 종료부터) 하위 집합 배열 x. `SELECT slice(ARRAY[6,8,2,9,3], 1, 3)` 예제 결과: `[ 6,8,2 ]`
zip(array1, array2[, ...])	배열(행)	주어진 배열을 요소별로 단일 행 배열로 병합합니다. 인수의 길이가 고르지 않은 경우 누락된 값은 로 채워집니다NULL. `SELECT zip(ARRAY[6,8,2,9,3], ARRAY[15, 24])` 예제 결과: `[ ( 6, 15 ), ( 8, 24 ),( 2, - ), ( 9, - ),( 3, - ) ]`

비트 함수

의 Timestream은 다음과 같은 비트 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
bit_count(bigint, bigint)	bigint(두 개의 보완)	두 번째 파라미터가 8 또는 64와 같은 비트 부호 있는 정수인 첫 번째 Bigint 파라미터의 비트 수를 반환합니다. `SELECT bit_count(19, 8)`

함수	출력 데이터 유형	설명
		예제 결과: 3 `SELECT bit_count(19, 2)` 예제 결과: `Number must be representable with the bits specified . 19 can not be represented with 2 bits`
bitwise_and(bigint, bigint)	bigint(두 개의 보완)	bigint 파라미터AND의 비트 단위를 반환합니다. `SELECT bitwise_and(12, 7)` 예제 결과: 4
bitwise_not(bigint)	bigint(두 개의 보완)	bigint 파라미터NOT의 비트를 반환합니다. `SELECT bitwise_not(12)` 예제 결과: -13
bitwise_or(bigint, bigint)	bigint(두 개의 보완)	bigint 파라미터의 비트 OR을 반환합니다. `SELECT bitwise_or(12, 7)` 예제 결과: 15

함수	출력 데이터 유형	설명
bitwise_xor(bigint, bigint)	bigint(두 개의 보완)	bigint 파라미터XOR의 비트를 반환합니다. `SELECT bitwise_xor(12, 7)` 예제 결과: 11

정규식 함수

에 대한 Timestream의 정규식 함수는 Java 패턴 구문을 LiveAnalytics 지원합니다. 의 Timestream은 다음과 같은 정규식 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
regexp_extract_all(문자열, 패턴)	array(varchar)	문자열의 정규식 패턴과 일치하는 하위 문자열(들)을 반환합니다. `SELECT regexp_extract_all('example expect complex', 'ex\w')` 예제 결과: `[ exa,exp ]`
regexp_extract_all(문자열, 패턴, 그룹)	array(varchar)	문자열에서 정규식 패턴의 모든 발생을 찾고 캡처 그룹 번호 그룹을 반환합니다. `SELECT regexp_extract_all('example expect complex', '(ex)(\w)', 2)`

함수	출력 데이터 유형	설명
		예제 결과: `[ a,p ]`
regexp_extract(문자열, 패턴)	varchar	문자열의 정규식 패턴과 일치하는 첫 번째 하위 문자열을 반환합니다. `SELECT regexp_extract('example expect', 'ex\w')` 예제 결과: `exa`
regexp_extract(문자열, 패턴, 그룹)	varchar	문자열에서 정규식 패턴의 첫 번째 발생을 찾아 캡처 그룹 번호 그룹을 반환합니다. `SELECT regexp_extract('example expect', '(ex)(\w)', 2)` 예제 결과: `a`

함수	출력 데이터 유형	설명
regexp_like(문자열, 패턴)	boolean	정규식 패턴을 평가하고 문자열 내에 포함되는지 확인합니다. 이 함수는 모든 문자열을 일치시킬 필요 없이 패턴만 문자열 내에 포함하면 된다는 점을 제외하면 LIKE 연산자와 유사합니다. 즉, 일치 작업 대신 포함 작업을 수행합니다. ^ 및 $를 사용하여 패턴을 고정하여 전체 문자열을 일치시킬 수 있습니다. `SELECT regexp_like('example', 'ex')` 예제 결과: `true`
regexp_replace(문자열, 패턴)	varchar	문자열에서 정규식 패턴과 일치하는 하위 문자열의 모든 인스턴스를 제거합니다. `SELECT regexp_replace('example expect', 'expect')` 예제 결과: `example`

함수	출력 데이터 유형	설명
regexp_replace(문자열, 패턴, 교체)	varchar	문자열의 정규식 패턴과 일치하는 하위 문자열의 모든 인스턴스를 대체로 바꿉니다. 번호가 매겨진 그룹의 경우 $g를, 명명된 그룹의 경우 ${name}을 사용하여 그룹 캡처를 대체하여 참조할 수 있습니다. 달러 기호($)는 백슬래시(\$)로 대체에 포함할 수 있습니다. `SELECT regexp_replace('example expect', 'expect', 'surprise')` 예제 결과: `example surprise`
regexp_replace(문자열, 패턴, 함수)	varchar	함수를 사용하여 문자열의 정규식 패턴과 일치하는 하위 문자열의 모든 인스턴스를 바꿉니다. lambda 표현식 함수는 배열로 전달된 캡처 그룹과 함께 각 일치에 대해 호출됩니다. 그룹 번호 캡처는 한 곳에서 시작합니다. 전체 일치에 대한 그룹은 없습니다(필요한 경우 전체 표현식을 괄호로 묶습니다). `SELECT regexp_replace('example', '(\w)', x -> upper(x[1]))` 예제 결과: `EXAMPLE`

함수	출력 데이터 유형	설명
regexp_split(문자열, 패턴)	array(varchar)	정규식 패턴을 사용하여 문자열을 분할하고 배열을 반환합니다. 후행 빈 문자열은 보존됩니다. `SELECT regexp_split('example', 'x')` 예제 결과: `[ e,ample ]`

날짜/시간 연산자

> **Note**
>
> 에 대한 Timestream LiveAnalytics 은 음수 시간 값을 지원하지 않습니다. 음수 시간을 초래하는 모든 작업은 오류가 발생합니다.

에 대한 Timestream은 `timestamps`, `dates`및 에서 다음 작업을 LiveAnalytics 지원합니다`intervals`.

연산자	설명
+	Addition
-	뺄셈

주제

- 운영
- Addition
- 뺄셈

운영

작업의 결과 유형은 피연산자를 기반으로 합니다. 1day 및 와 같은 간격 리터럴을 사용할 3s 수 있습니다.

```
SELECT date '2022-05-21' + interval '2' day
```

```
SELECT date '2022-05-21' + 2d
```

```
SELECT date '2022-05-21' + 2day
```

각 에 대한 예제 결과: 2022-05-23

간격 단위에는 second, minute, hour, day, week, 및 month가 포함됩니다year. 그러나 일부 경우에는 모두 해당되지 않습니다. 예를 들어 초, 분 및 시간은 날짜에 추가하거나 뺄 수 없습니다.

```
SELECT interval '4' year + interval '2' month
```

예제 결과: 4-2

```
SELECT typeof(interval '4' year + interval '2' month)
```

예제 결과: interval year to month

간격 작업의 결과 유형은 피연산자에 'interval day to second' 따라 'interval year to month' 또는 가 될 수 있습니다. 간격은 dates 및 에 추가하거나 에서 뺄 수 있습니다timestamps. 그러나 date 또는 는 date 또는 에 추가하거나 에서 뺄 수 timestamp 없습니다timestamp. 날짜 또는 타임스탬프와 관련된 간격 또는 기간을 찾으려면 의 date_diff 및 관련 함수를 참조하세요간격 및 기간.

Addition

Example

```
SELECT date '2022-05-21' + interval '2' day
```

예제 결과: 2022-05-23

Example

```
SELECT typeof(date '2022-05-21' + interval '2' day)
```

예제 결과: `date`

Example

```
SELECT interval '2' year + interval '4' month
```

예제 결과: `2-4`

Example

```
SELECT typeof(interval '2' year + interval '4' month)
```

예제 결과: `interval year to month`

뺄셈

Example

```
SELECT timestamp '2022-06-17 01:00' - interval '7' hour
```

예제 결과: `2022-06-16 18:00:00.000000000`

Example

```
SELECT typeof(timestamp '2022-06-17 01:00' - interval '7' hour)
```

예제 결과: `timestamp`

Example

```
SELECT interval '6' day - interval '4' hour
```

예제 결과: `5 20:00:00.000000000`

Example

```
SELECT typeof(interval '6' day - interval '4' hour)
```

예제 결과: `interval day to second`

날짜/시간 함수

> **Note**
>
> 에 대한 Timestream LiveAnalytics 은 음수 시간 값을 지원하지 않습니다. 음수 시간을 초래하는 모든 작업은 오류가 발생합니다.

의 시간대는 날짜 및 시간에 UTC 시간대를 LiveAnalytics 사용합니다. Timestream은 날짜 및 시간에 대해 다음 함수를 지원합니다.

주제

- 일반 및 변환
- 간격 및 기간
- 서식 지정 및 구문 분석
- 추출

일반 및 변환

의 Timestream은 날짜 및 시간에 대해 다음과 같은 일반 및 변환 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
현재_날짜	date	에서 현재 날짜를 반환합니다 UTC. 사용된 괄호가 없습니다. `SELECT current_date` 예제 결과: `2022-07-07` > **Note** > 이는 예약된 키워드이기도 합니다. 예약된 키

함수	출력 데이터 유형	설명
		워드 목록은 섹션을 참조하세요예약어.
현재_시간	시간	에서 현재 시간을 반환합니다 UTC. 사용된 괄호가 없습니다. `SELECT current_time` 예제 결과: `17:41:52.827000000` Note 이는 예약된 키워드이기도 합니다. 예약된 키워드 목록은 섹션을 참조하세요예약어.
current_timestamp 또는 now()	타임스탬프	에서 현재 타임스탬프를 반환합니다UTC. `SELECT current_timestamp` 예제 결과: `2022-07-07 17:42:32.939000000` Note 이는 예약된 키워드이기도 합니다. 예약된 키워드 목록은 섹션을 참조하세요예약어.

함수	출력 데이터 유형	설명
current_timezone()	varchar 값은 '.'입니다UTC.	Timestream은 날짜 및 시간에 UTC 시간대를 사용합니다. `SELECT current_timezone()` 예제 결과: UTC
date(varchar(x)), date(timestamp)	date	`SELECT date(TIMESTAMP '2022-07-07 17:44:43.771000000')` 예제 결과: 2022-07-07
마지막_월_일(타임스탬프), 마지막_월_일(날짜)	date	`SELECT last_day_of_month(TIMESTAMP '2022-07-07 17:44:43.771000000')` 예제 결과: 2022-07-31
from_iso8601_timestamp(문자열)	타임스탬프	ISO 8601 타임스탬프를 내부 타임스탬프 형식으로 구문 분석합니다. `SELECT from_iso8601_timestamp('2022-06-17T08:04:05.000000000+05:00')` 예제 결과: 2022-06-17 03:04:05.000000000

함수	출력 데이터 유형	설명
from_iso8601_date(문자열)	date	8601 날짜 문자열을 지정된 날짜의 ISO UTC00:00:00에 대한 내부 타임스탬프 형식으로 구문 분석합니다. `SELECT from_iso8601_date('2022-07-17')` 예제 결과: `2022-07-17`
to_iso8601(타임스탬프), to_iso8601(날짜)	varchar	입력에 대한 ISO 8601 형식의 문자열을 반환합니다. `SELECT to_iso8601(from_iso8601_date('2022-06-17'))` 예제 결과: `2022-06-17`
from_milliseconds(bigint)	타임스탬프	`SELECT from_milliseconds(1)` 예제 결과: `1970-01-01 00:00:00.001000000`
from_nanoseconds(bigint)	타임스탬프	`select from_nanoseconds(300000001)` 예제 결과: `1970-01-01 00:00:00.300000001`

함수	출력 데이터 유형	설명
from_unixtime(이중)	타임스탬프	제공된 unixtime에 해당하는 타임스탬프를 반환합니다. `SELECT from_unixtime(1)` 예제 결과: `1970-01-01 00:00:01.000000000`
로컬 시간	시간	에서 현재 시간을 반환합니다 UTC. 사용된 괄호가 없습니다. `SELECT localtime` 예제 결과: `17:58:22.654000000` **Note** 이는 예약된 키워드이기도 합니다. 예약된 키워드 목록은 섹션을 참조하세요예약어.

함수	출력 데이터 유형	설명
localtimestamp	타임스탬프	에서 현재 타임스탬프를 반환합니다UTC. 사용된 괄호가 없습니다. `SELECT localtimestamp` 예제 결과: `2022-07-07 17:59:04.368000000` **Note** 이는 예약된 키워드이기도 합니다. 예약된 키워드 목록은 섹션을 참조하세요예약어.
to_milliseconds(일에서 초 간격), to_milliseconds(timestamp)	bigint	`SELECT to_milliseconds(INTERVAL '2' DAY + INTERVAL '3' HOUR)` 예제 결과: `183600000` `SELECT to_milliseconds(TIMESTAMP '2022-06-17 17:44:43.771000000')` 예제 결과: `1655487883771`

함수	출력 데이터 유형	설명
~_nanoseconds(일에서 초 간격), ~_nanoseconds(timestamp)	bigint	`SELECT to_nanoseconds(INTERVAL '2' DAY + INTERVAL '3' HOUR)` 예제 결과: 183600000000000 `SELECT to_nanoseconds(TIMESTAMP '2022-06-17 17:44:43.771000678')` 예제 결과: 1655487883771000678
to_unixtime(타임스탬프)	double	제공된 타임스탬프에 대한 unixtime을 반환합니다. `SELECT to_unixtime('2022-06-17 17:44:43.771000000')` 예제 결과: 1.655487883771E9

함수	출력 데이터 유형	설명
date_trunc(단위, 타임스탬프)	타임스탬프	단위로 잘린 타임스탬프를 반환합니다. 여기서 단위는 [초, 분, 시간, 일, 주, 월, 분기 또는 연도] 중 하나입니다. `SELECT date_trunc('minute', TIMESTAMP '2022-06-17 17:44:43.771000000')` 예제 결과: `2022-06-17 17:44:00.000000000`

간격 및 기간

의 Timestream은 날짜 및 시간에 대해 다음과 같은 간격 및 기간 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
date_add(단위, bigint, date), date_add(단위, bigint, time), date_add(varchar(x), bigint, timestamp)	타임스탬프	단위가 [초, 분, 시간, 일, 주, 월, 분기 또는 연도] 중 하나인 단위의 대인트를 추가합니다. `SELECT date_add('hour', 9, TIMESTAMP '2022-06-17 00:00:00')` 예제 결과: `2022-06-17 09:00:00.000000000`
date_diff(단위, 날짜, 날짜) , date_diff(단위, 시간, 시간) , date_diff(단위, 타임스탬프, 타임스탬프)	bigint	단위가 [초, 분, 시간, 일, 주, 월, 분기 또는 연도] 중 하나인 차이를 반환합니다.

함수	출력 데이터 유형	설명
		`SELECT date_diff('day', DATE '2020-03-01', DATE '2020-03-02')` 예제 결과: 1
parse_duration(문자열)	interval	입력 문자열을 구문 분석하여 interval 동등한 문자열을 반환합니다. `SELECT parse_duration('42.8ms')` 예제 결과: `0 00:00:00.042800000` `SELECT typeof(parse_duration('42.8ms'))` 예제 결과: `interval day to second`

함수	출력 데이터 유형	설명
bin(타임스탬프, 간격)	타임스탬프	`timestamp` 파라미터의 정수 값을 파라미터 정수 `interval` 값의 가장 가까운 배수로 반올림합니다. 이 반환 값의 의미는 명확하지 않을 수 있습니다. 먼저 타임스탬프 정수를 간격 정수로 나눈 다음 결과에 간격 정수를 곱하여 정수 산술을 사용하여 계산합니다. 타임스탬프는 UTC 시점을 POSIX 에포크(1970년 1월 1일) 이후 경과된 초 단위의 수로 지정하므로 반환 값은 달력 단위와 거의 정렬되지 않습니다. 예를 들어 간격을 30일로 지정하면 에폭이 30일 단위로 분할되고 가장 최근 30일 증분의 시작이 반환되며, 이는 역월과 관련이 없습니다. 여기 몇 가지 예가 있습니다: `bin(TIMESTAMP '2022-06-17 10:15:20', 5m) ==> 2022-06-17 10:15:00.000000000` `bin(TIMESTAMP '2022-06-17 10:15:20', 1d) ==> 2022-06-17 00:00:00.000000000` `bin(TIMESTAMP '2022-06-17 10:15:20', 10day) ==> 2022-06-17 00:00:00.000000000`

함수	출력 데이터 유형	설명
		`bin(TIMESTAMP '2022-06-17 10:15:20', 30day) ==> 2022-05-28 00:00:00.000000000`
전(간격)	타임스탬프	current_timestamp 에 해당하는 값을 반환합니다`interval`. `SELECT ago(1d)` 예제 결과: `2022-07-06 21:08:53.245000000`
1시간, 1일, 30분과 같은 간격 리터럴	interval	간격 리터럴은 parse_duration(문자열)의 편리함입니다. 예를 들어 `1d`는 `parse_duration('1d')` 와 동일합니다. 이렇게 하면 간격이 사용되는 모든 곳에서 리터럴을 사용할 수 있습니다. 예: `ago(1d)` 및 `bin(`*`<timestamp>`* `, 1m)`.

일부 간격 리터럴은 parse_duration의 약어 역할을 합니다. 예를 들어 , `parse_duration('1d')`, 및 `1d` 각 `parse_duration('1day')` `1day`는 유형이 `1 00:00:00.000000000`인 를 반환합니다`interval day to second`. 에 제공된 형식으로 공백이 허용됩니다`parse_duration`. 예를 들어 `parse_duration('1day')`도 반환합니다`00:00:00.000000000`. 하지만 간격 리터럴`1 day`은 아닙니다.

와 관련된 단위는 ns, 나노초, us, 마이크로초, ms, 밀리초, s, 초, m, 분, h, 시간, d, 일`interval day to second`입니다.

또한 도 있습니다`interval year to month`. 연도 간 간격과 관련된 단위는 y, 연도 및 월입니다. 예를 들어 는 를 `SELECT 1year` 반환`1-0`하고 `SELECT 12month`도 를 반환`1-0`합니다. 는 를 `SELECT 8month` 반환합니다`0-8`.

의 단위quarter는 date_trunc 및 와 같은 일부 함수에서도 사용할 수 있지만 date_addquarter는 간격 리터럴의 일부로 사용할 수 없습니다.

서식 지정 및 구문 분석

의 Timestream은 날짜 및 시간에 대해 다음과 같은 형식 지정 및 구문 분석 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
date_format(timestamp, varchar(x))	varchar	이 함수에서 사용하는 형식 지정자에 대한 자세한 내용은 https://trino.io/docs/current/functions/datetime.html#mysql-date-functions을 참조하세요. `SELECT date_format(TIMESTAMP '2019-10-20 10:20:20', '%Y-%m-%d %H:%i:%s')` 예제 결과: 2019-10-20 10:20:20
date_parse(varchar(x), varchar(y))	타임스탬프	이 함수에서 사용하는 형식 지정자에 대한 자세한 내용은 https://trino.io/docs/current/functions/datetime.html#mysql-date-functions을 참조하세요. `SELECT date_parse('2019-10-20 10:20:20', '%Y-%m-%d %H:%i:%s')` 예제 결과: 2019-10-20 10:20:20.000000000

함수	출력 데이터 유형	설명
format_datetime(timestamp, varchar(x))	varchar	이 함수에서 사용하는 형식 문자열에 대한 자세한 내용은 http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html을 참조하세요. `SELECT format_datetime(parse_datetime('1968-01-13 12', 'yyyy-MM-dd HH'), 'yyyy-MM-dd HH')` 예제 결과: `1968-01-13 12`
parse_datetime(varchar(x), varchar(y))	타임스탬프	이 함수에서 사용하는 형식 문자열에 대한 자세한 내용은 http://joda-time.sourceforge.net/apidocs/org/joda/time/format/DateTimeFormat.html을 참조하세요. `SELECT parse_datetime('2019-12-29 10:10 PST', 'uuuu-LL-dd HH:mm z')` 예제 결과: `2019-12-29 18:10:00.000000000`

추출

의 Timestream은 날짜 및 시간에 대해 다음과 같은 추출 함수를 LiveAnalytics 지원합니다. 추출 함수는 나머지 편의 함수의 기본입니다.

함수	출력 데이터 유형	설명
extract	bigint	타임스탬프에서 필드를 추출합니다. 여기서 필드는 [YEAR, , QUARTER, MONTH, WEEK, DAYDAY_OF_MONTH, DAY_OF_WEEK, DOW, DAY_OF_YEAR, DOY, YEAR_OF_WEEK, YOW, HOURMINUTE, 또는 SECOND] 중 하나입니다. `SELECT extract(YEAR FROM '2019-10-12 23:10:34.000000000')` 예제 결과: 2019
day(timestamp), day(date), day(interval day to second)	bigint	`SELECT day('2019-10-12 23:10:34.000000000')` 예제 결과: 12
day_of_month(타임스탬프), day_of_month(date), day_of_month(간격일에서 초)	bigint	`SELECT day_of_month('2019-10-12 23:10:34.000000000')` 예제 결과: 12
day_of_week(타임스탬프), day_of_week(date)	bigint	`SELECT day_of_week('2019-10-12 23:10:34.000000000')` 예제 결과: 6

함수	출력 데이터 유형	설명
day_of_year(타임스탬프), day_of_year(date)	bigint	`SELECT day_of_year('2019-10-12 23:10:34.000000000')` 예제 결과: 285
dow(타임스탬프), dow(날짜)	bigint	day_of_week의 별칭
doy(타임스탬프), doy(날짜)	bigint	연도_일 별칭
hour(timestamp), hour(time), hour(interval day to second)	bigint	`SELECT hour('2019-10-12 23:10:34.000000000')` 예제 결과: 23
밀리초(타임스탬프), 밀리초(시간), 밀리초(일-초 간격)	bigint	`SELECT millisecond('2019-10-12 23:10:34.000000000')` 예제 결과: 0
minute(timestamp), minute(time), minute(interval day to second)	bigint	`SELECT minute('2019-10-12 23:10:34.000000000')` 예제 결과: 10
month(timestamp), month(date), month(interval year to month)	bigint	`SELECT month('2019-10-12 23:10:34.000000000')` 예제 결과: 10

함수	출력 데이터 유형	설명
nanosecond(타임스탬프), nanosecond(시간), nanosecond(일에서 초 간격)	bigint	`SELECT nanosecond(current_timestamp)` 예제 결과: `162000000`
quarter(timestamp), quarter(date)	bigint	`SELECT quarter('2019-10-12 23:10:34.000000000')` 예제 결과: `4`
second(timestamp), second(time), second(간격 요일에서 초)	bigint	`SELECT second('2019-10-12 23:10:34.000000000')` 예제 결과: `34`
week(타임스탬프), week(date)	bigint	`SELECT week('2019-10-12 23:10:34.000000000')` 예제 결과: `41`
week_of_year(타임스탬프), week_of_year(date)	bigint	주별 별칭
연도(타임스탬프), 연도(날짜), 연도(연간-월)	bigint	`SELECT year('2019-10-12 23:10:34.000000000')` 예제 결과: `2019`
year_of_week(타임스탬프), year_of_week(날짜)	bigint	`SELECT year_of_week('2019-10-12 23:10:34.000000000')` 예제 결과: `2019`

함수	출력 데이터 유형	설명
yow(타임스탬프), yow(날짜)	bigint	year_of_week의 별칭

집계 함수

의 Timestream은 다음과 같은 집계 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
임의(x)	[입력과 동일]	임의의 null이 아닌 값이 있는 경우 x 값을 반환합니다. `SELECT arbitrary(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: 1
array_agg(x)	배열<[입력과 동일]	입력 x 요소에서 생성된 배열을 반환합니다. `SELECT array_agg(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: `[ 1,2,3,4 ]`
평균(x)	double	모든 입력 값의 평균(산술 평균)을 반환합니다. `SELECT avg(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: `2.5`

함수	출력 데이터 유형	설명
bool_and(boolean) every(boolean)	boolean	모든 입력 값이 인 TRUE 경우를 반환하고TRUE, 그렇지 않은 경우 를 반환합니다FALSE. `SELECT bool_and(t.c) FROM (VALUES true, true, false, true) AS t(c)` 예제 결과: `false`
bool_or(부울)	boolean	입력 값이 이면 를 반환하고 TRUE, 그렇지 TRUE 않으면 를 반환합니다FALSE. `SELECT bool_or(t.c) FROM (VALUES true, true, false, true) AS t(c)` 예제 결과: `true`
count(*) count(x)	bigint	count(*)는 입력 행 수를 반환합니다. count(x)는 null이 아닌 입력 값의 수를 반환합니다. `SELECT count(t.c) FROM (VALUES true, true, false, true) AS t(c)` 예제 결과: `4`

함수	출력 데이터 유형	설명
count_if(x)	bigint	TRUE 입력 값의 수를 반환합니다. `SELECT count_if(t.c) FROM (VALUES true, true, false, true) AS t(c)` 예제 결과: 3
geometric_mean(x)	double	모든 입력 값의 기하 평균을 반환합니다. `SELECT geometric_mean(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: 2.213363839400643
max_by(x, y)	[x와 동일]	모든 입력 값에 대해 y의 최대 값과 연결된 x의 값을 반환합니다. `SELECT max_by(t.c1, t.c2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 예제 결과: d

함수	출력 데이터 유형	설명
max_by(x, y, n)	배열<[x와 동일]>	y의 모든 입력 값 중 가장 큰 n과 연결된 x의 n 값을 y의 내림차순으로 반환합니다. `SELECT max_by(t.c1, t.c2, 2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 예제 결과: `[ d,c ]`
min_by(x, y)	[x와 동일]	모든 입력 값에 대해 y의 최소 값과 연결된 x의 값을 반환합니다. `SELECT min_by(t.c1, t.c2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 예제 결과: a
min_by(x, y, n)	배열<[x와 동일]>	y의 오름차순으로 y의 모든 입력 값 중 가장 작은 n과 연결된 x의 n 값을 반환합니다. `SELECT min_by(t.c1, t.c2, 2) FROM (VALUES (('a', 1)), (('b', 2)), (('c', 3)), (('d', 4))) AS t(c1, c2)` 예제 결과: `[ a,b ]`

함수	출력 데이터 유형	설명
max(x)	[입력과 동일]	모든 입력 값의 최대값을 반환합니다. `SELECT max(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: 4
최대(x, n)	배열<[x와 동일]>	x의 모든 입력 값 중 가장 큰 값 n개를 반환합니다. `SELECT max(t.c, 2) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: [4,3]
min(x)	[입력과 동일]	모든 입력 값의 최소값을 반환합니다. `SELECT min(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: 1
min(x, n)	배열<[x와 동일]>	x의 모든 입력 값 중 최소 n개의 값을 반환합니다. `SELECT min(t.c, 2) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: [1,2]

함수	출력 데이터 유형	설명
sum(x)	[입력과 동일]	모든 입력 값의 합계를 반환합니다. `SELECT sum(t.c) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: `10`
bitwise_and_agg(x)	bigint	AND 모든 입력 값의 비트 단위를 2's 보완 표현으로 반환합니다. `SELECT bitwise_and_agg(t.c) FROM (VALUES 1, -3) AS t(c)` 예제 결과: `1`
bitwise_or_agg(x)	bigint	모든 입력 값의 비트 OR을 2's 보완 표현으로 반환합니다. `SELECT bitwise_or_agg(t.c) FROM (VALUES 1, -3) AS t(c)` 예제 결과: `-3`

함수	출력 데이터 유형	설명
약_구분(x)	bigint	고유한 입력 값의 대략적인 수를 반환합니다. 이 함수는 수(DISTINCT x)의 근사치를 제공합니다. 모든 입력 값이 null이면 0이 반환됩니다. 이 함수는 가능한 모든 세트에 대한 (대략 정상) 오류 분포의 표준 편차인 2.3%의 표준 오차를 생성해야 합니다. 특정 입력 세트에 대한 오류 상한을 보장하지는 않습니다. `SELECT approx_distinct(t.c) FROM (VALUES 1, 2, 3, 4, 8) AS t(c)` 예제 결과: 5

함수	출력 데이터 유형	설명
approx_distinct(x, e)	bigint	고유한 입력 값의 대략적인 수를 반환합니다. 이 함수는 수(DISTINCT x)의 근사치를 제공합니다. 모든 입력 값이 null이면 0이 반환됩니다. 이 함수는 가능한 모든 세트에 대한 (대략 정상) 오류 분포의 표준 편차인 e 이하의 표준 오차를 생성해야 합니다. 특정 입력 세트에 대한 오류 상한을 보장하지는 않습니다. 이 함수를 현재 구현하려면 e가 [0.0040625, 0.26000] 범위에 있어야 합니다. `SELECT approx_distinct(t.c, 0.2) FROM (VALUES 1, 2, 3, 4, 8) AS t(c)` 예제 결과: 5
약_백분위수(x, 백분율)	[x와 동일]	지정된 백분율에서 x의 모든 입력 값에 대한 대략적인 백분위수를 반환합니다. 백분율 값은 0에서 1 사이여야 하며 모든 입력 행에 대해 일정해야 합니다. `SELECT approx_percentile(t.c, 0.4) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: 2

함수	출력 데이터 유형	설명
약_백분위수(x, 백분율)	배열<[x와 동일]>	지정된 각 백분율에서 x의 모든 입력 값에 대한 대략적인 백분위수를 반환합니다. 백분율 배열의 각 요소는 0과 1 사이여야 하며 배열은 모든 입력 행에 대해 일정해야 합니다. `SELECT approx_percentile(t.c, ARRAY[0.1, 0.8, 0.8]) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: `[ 1,4,4 ]`
약_백분위수(x, w, 백분율)	[x와 동일]	p 백분율에서 항목별 가중치 w를 사용하여 x의 모든 입력 값에 대한 대략적인 가중치 백분위수를 반환합니다. 가중치는 1 이상의 정수 값이어야 합니다. 백분위수 세트의 값 x에 대한 복제 수입니다. p 값은 0과 1 사이여야 하며 모든 입력 행에 대해 일정해야 합니다. `SELECT approx_percentile(t.c, 1, 0.1) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: 1

함수	출력 데이터 유형	설명
약_백분위수(x, w, 백분율)	배열<[x와 동일]>	배열에 지정된 지정된 각 백분율에서 항목별 가중치 w를 사용하여 x의 모든 입력 값에 대한 대략적인 가중치 백분위수를 반환합니다. 가중치는 1 이상의 정수 값이어야 합니다. 백분위수 세트의 값 x에 대한 복제 수입니다. 배열의 각 요소는 0과 1 사이여야 하며 배열은 모든 입력 행에 대해 일정해야 합니다. `SELECT approx_pe rcentile(t.c, 1, ARRAY[0.1, 0.8, 0.8]) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: `[ 1,4,4 ]`

함수	출력 데이터 유형	설명
approx_percentile(x, w, 백분율, 정확도)	[x와 동일]	p 백분율에서 항목별 가중치 w를 사용하여 x의 모든 입력 값에 대한 대략적인 가중치 백분위수를 반환하고 최대 순위 오류는 정확도입니다. 가중치는 1 이상의 정수 값이어야 합니다. 백분위수 세트의 값 x에 대한 복제 수입니다. p 값은 0과 1 사이여야 하며 모든 입력 행에 대해 일정해야 합니다. 정확도는 0보다 크고 1보다 작아야 하며 모든 입력 행에 대해 일정해야 합니다. `SELECT approx_percentile(t.c, 1, 0.1, 0.5) FROM (VALUES 1, 2, 3, 4) AS t(c)` 예제 결과: `1`
corr(y, x)	double	입력 값의 상관 계수를 반환합니다. `SELECT corr(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 예제 결과: `1.0`

함수	출력 데이터 유형	설명
covar_pop(y, x)	double	입력 값의 모집단 공분산을 반환합니다. `SELECT covar_pop(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 예제 결과: `1.25`
covar_samp(y, x)	double	입력 값의 샘플 공분산을 반환합니다. `SELECT covar_samp(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 예제 결과: `1.6666666666666667`
regr_intercept(y, x)	double	입력 값의 선형 회귀 절편을 반환합니다. y는 종속 값이고 x는 독립 값입니다. `SELECT regr_intercept(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 예제 결과: `0.0`

함수	출력 데이터 유형	설명
regr_slope(y, x)	double	입력 값의 선형 회귀 기울기를 반환합니다. y는 종속 값이고 x는 독립 값입니다. `SELECT regr_slope(t.c1, t.c2) FROM (VALUES ((1, 1)), ((2, 2)), ((3, 3)), ((4, 4))) AS t(c1, c2)` 예제 결과: `1.0`
비뚤어짐(x)	double	모든 입력 값의 왜곡을 반환합니다. `SELECT skewness(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 예제 결과: `0.8978957037987335`
stddev_pop(x)	double	모든 입력 값의 모집단 표준 편차를 반환합니다. `SELECT stddev_pop(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 예제 결과: `2.4166091947189146`

함수	출력 데이터 유형	설명
stddev_samp(x) stddev(x)	double	모든 입력 값의 샘플 표준 편차를 반환합니다. `SELECT stddev_samp(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 예제 결과: 2.701851217221259
var_pop(x)	double	모든 입력 값의 모집단 분산을 반환합니다. `SELECT var_pop(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 예제 결과: 5.840000000000001
var_samp(x) 분산(x)	double	모든 입력 값의 샘플 분산을 반환합니다. `SELECT var_samp(t.c1) FROM (VALUES 1, 2, 3, 4, 8) AS t(c1)` 예제 결과: 7.300000000000001

윈도 함수

창 함수는 쿼리 결과의 행에 걸쳐 계산을 수행합니다. 절 뒤에는 실행HAVING되지만 ORDER BY 절 앞에는 실행됩니다. 창 함수를 호출하려면 OVER 절을 사용하여 창을 지정하는 특수 구문이 필요합니다. 창은 세 가지 구성 요소로 구성됩니다.

- 입력 행을 다른 파티션으로 구분하는 파티션 사양입니다. 이는 GROUP BY 절이 집계 함수를 위해 행을 서로 다른 그룹으로 구분하는 방식과 유사합니다.
- 윈도우 함수에서 입력 행을 처리하는 순서를 결정하는 주문 사양입니다.
- 지정된 행의 함수에서 처리할 행의 슬라이딩 창을 지정하는 창 프레임입니다. 프레임이 지정되지 않은 경우 기본적으로 RANGE UNBOUNDED 로 설정되며PRECEDING, RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT 이는 와 동일합니다ROW. 이 프레임에는 파티션 시작부터 현재 행의 마지막 피어까지 모든 행이 포함됩니다.

OVER 절을 추가하여 모든 집계 함수를 창 함수로 사용할 수 있습니다. 집계 함수는 현재 행의 창 프레임 내에 있는 행의 각 행에 대해 계산됩니다. 집계 함수 외에도 Timestream for는 다음과 같은 순위 및 값 함수를 LiveAnalytics 지원합니다.

함수	출력 데이터 유형	설명
cume_dist()	bigint	값 그룹에 있는 값의 누적 분포를 반환합니다. 그 결과 창 파티션의 창 순서에 행이 있는 선행 행 또는 피어 행 수를 창 파티션의 총 행 수로 나눈 값이 됩니다. 따라서 순서의 모든 연결 값은 동일한 분포 값으로 평가됩니다.
dense_rank()	bigint	값 그룹의 값 순위를 반환합니다. 이는 타이 값이 시퀀스에 공백을 생성하지 않는다는 점을 제외하면 rank()와 유사합니다.
ntile(n)	bigint	각 윈도우 파티션의 행을 1~최대 n 범위의 버킷 n개로 나눕니다. 버킷 값은 최대 1까지 달라집니다. 파티션의 행 수가 버킷 수로 균등하게 분할되지 않으면 나머지 값은 첫 번째 버킷부터 버킷당 하나씩 배포됩니다.

함수	출력 데이터 유형	설명
percent_rank()	double	값 그룹의 값의 백분율 순위를 반환합니다. 결과는 (r - 1) / (n - 1)입니다. 여기서 r은 행의 순위()이고 n은 창 파티션의 총 행 수입니다.
순위()	bigint	값 그룹의 값 순위를 반환합니다. 순위는 1에 행과 피어링되지 않은 행 앞의 행 수를 더한 값입니다. 따라서 순서에 값을 연결하면 시퀀스에 공백이 생성됩니다. 순위는 각 창 파티션에 대해 수행됩니다.
row_number()	bigint	창 파티션 내의 행 순서에 따라 각 행에 대해 하나씩 시작하는 고유한 순차적 번호를 반환합니다.
first_value(x)	[입력과 동일]	창의 첫 번째 값을 반환합니다. 이 함수는 창 프레임으로 범위가 지정됩니다. 함수는 표현식 또는 대상을 파라미터로 사용합니다.
last_value(x)	[입력과 동일]	창의 마지막 값을 반환합니다. 이 함수는 창 프레임으로 범위가 지정됩니다. 함수는 표현식 또는 대상을 파라미터로 사용합니다.

함수	출력 데이터 유형	설명
nth_value(x, 오프셋)	[입력과 동일]	지정된 오프셋의 값을 창 시작부터 반환합니다. 오프셋은 1부터 시작합니다. 오프셋은 모든 스칼라 표현식일 수 있습니다. 오프셋이 null이거나 창의 값 수보다 크면 null이 반환됩니다. 오프셋이 0 또는 음수인 경우 오류가 발생합니다. 함수는 표현식 또는 대상을 첫 번째 파라미터로 사용합니다.
lead(x[, offset[, default_value]])	[입력과 동일]	창의 현재 행 뒤에 있는 오프셋 행의 값을 반환합니다. 오프셋은 현재 행인 0에서 시작합니다. 오프셋은 모든 스칼라 표현식일 수 있습니다. 기본 오프셋은 1입니다. 오프셋이 null이거나 창보다 크면 default_value가 반환되고, 지정되지 않은 경우에는 null이 반환됩니다. 함수는 표현식 또는 대상을 첫 번째 파라미터로 사용합니다.
lag(x[, offset[, default_value]])	[입력과 동일]	오프셋 창의 현재 행 앞에 있는 오프셋 행의 값을 반환합니다. 오프셋은 현재 행인 0에서 시작합니다. 오프셋은 모든 스칼라 표현식일 수 있습니다. 기본 오프셋은 1입니다. 오프셋이 null이거나 창보다 크면 default_value가 반환되고, 지정되지 않은 경우에는 null이 반환됩니다. 함수는 표현식 또는 대상을 첫 번째 파라미터로 사용합니다.

샘플 쿼리

이 섹션에는 의 쿼리 언어에 대한 Timestream 사용 사례의 예 LiveAnalytics가 포함되어 있습니다.

주제

- 간단한 쿼리
- 시계열 함수를 사용한 쿼리
- 집계 함수가 있는 쿼리

간단한 쿼리

다음은 테이블에 대해 가장 최근에 추가된 10개의 데이터 포인트를 가져옵니다.

```
SELECT * FROM <database_name>.<table_name>
ORDER BY time DESC
LIMIT 10
```

다음은 특정 측정값에 대해 가장 오래된 데이터 포인트 5개를 가져옵니다.

```
SELECT * FROM <database_name>.<table_name>
WHERE measure_name = '<measure_name>'
ORDER BY time ASC
LIMIT 5
```

다음은 나노초 단위 타임스탬프와 함께 작동합니다.

```
SELECT now() AS time_now
, now() - (INTERVAL '12' HOUR) AS twelve_hour_earlier -- Compatibility with ANSI SQL
, now() - 12h AS also_twelve_hour_earlier -- Convenient time interval literals
, ago(12h) AS twelve_hours_ago -- More convenience with time functionality
, bin(now(), 10m) AS time_binned -- Convenient time binning support
, ago(50ns) AS fifty_ns_ago -- Nanosecond support
, now() + (1h + 50ns) AS hour_fifty_ns_future
```

다중 측정 레코드의 측정값은 열 이름으로 식별됩니다. 단일 측정 레코드의 측정값은 로 식별되며measure_value::*<data_type>*, 여기서 *<data_type>*는 double에 설명된 varchar 대로 bigint, boolean, 또는 중 하나입니다지원되는 데이터 유형. 측정값을 모델링하는 방법에 대한 자세한 내용은 단일 테이블과 여러 테이블 비교를 참조하세요.

다음은 가 인 다중 측정 레코드speed에서 호출된 측정값measure_name의 값을 검색합니다IoTMulti-stats.

```
SELECT speed FROM <database_name>.<table_name> where measure_name = 'IoTMulti-stats'
```

다음은 measure_name가 인 단일 측정 레코드에서 double 값을 검색합니다load.

```
SELECT measure_value::double FROM <database_name>.<table_name> WHERE measure_name =
 'load'
```

시계열 함수를 사용한 쿼리

주제

- 데이터 세트 및 쿼리 예제

데이터 세트 및 쿼리 예제

용 Timestream LiveAnalytics 을 사용하여 서비스 및 애플리케이션의 성능과 가용성을 이해하고 개선할 수 있습니다. 다음은 해당 테이블에서 실행되는 예제 테이블 및 샘플 쿼리입니다.

테이블에는 EC2 인스턴스의 CPU 사용률 및 기타 지표와 같은 원격 측정 데이터가 ec2_metrics 저장됩니다. 아래 표를 볼 수 있습니다.

Time	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000000000	us-east-1	us-east-1a	프런트엔드01	cpu_utilization	35.1	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1a	프런트엔드01	memory_utilization	55.3	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1a	프런트엔드01	network_bytes_in	null	1,500

Time	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:00.000000000	us-east-1	us-east-1a	프런트엔드01	network_bytes_out	null	6,700
2019-12-04 19:00:00.000000000	us-east-1	us-east-1b	프런트엔드02	cpu_utilization	38.5	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1b	프런트엔드02	memory_utilization	58.4	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1b	프런트엔드02	network_bytes_in	null	23,000
2019-12-04 19:00:00.000000000	us-east-1	us-east-1b	프런트엔드02	network_bytes_out	null	12,000
2019-12-04 19:00:00.000000000	us-east-1	us-east-1c	프런트엔드03	cpu_utilization	45.0	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1c	프런트엔드03	memory_utilization	65.8	null
2019-12-04 19:00:00.000000000	us-east-1	us-east-1c	프런트엔드03	network_bytes_in	null	15,000
2019-12-04 19:00:00.000000000	us-east-1	us-east-1c	프런트엔드03	network_bytes_out	null	836,000

Time	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:05.000000000	us-east-1	us-east-1a	프런트엔드01	cpu_utilization	55.2	null
2019-12-04 19:00:05.000000000	us-east-1	us-east-1a	프런트엔드01	memory_utilization	75.0	null
2019-12-04 19:00:05.000000000	us-east-1	us-east-1a	프런트엔드01	network_bytes_in	null	1,245
2019-12-04 19:00:05.000000000	us-east-1	us-east-1a	프런트엔드01	network_bytes_out	null	68,432
2019-12-04 19:00:08.000000000	us-east-1	us-east-1b	프런트엔드02	cpu_utilization	65.6	null
2019-12-04 19:00:08.000000000	us-east-1	us-east-1b	프런트엔드02	memory_utilization	85.3	null
2019-12-04 19:00:08.000000000	us-east-1	us-east-1b	프런트엔드02	network_bytes_in	null	1,245
2019-12-04 19:00:08.000000000	us-east-1	us-east-1b	프런트엔드02	network_bytes_out	null	68,432
2019-12-04 19:00:20.000000000	us-east-1	us-east-1c	프런트엔드03	cpu_utilization	12.1	null

Time	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:20.000000000	us-east-1	us-east-1c	프런트엔드03	memory_utilization	32.0	null
2019-12-04 19:00:20.000000000	us-east-1	us-east-1c	프런트엔드03	network_bytes_in	null	1,400
2019-12-04 19:00:20.000000000	us-east-1	us-east-1c	프런트엔드03	network_bytes_out	null	345
2019-12-04 19:00:10.000000000	us-east-1	us-east-1a	프런트엔드01	cpu_utilization	15.3	null
2019-12-04 19:00:10.000000000	us-east-1	us-east-1a	프런트엔드01	memory_utilization	35.4	null
2019-12-04 19:00:10.000000000	us-east-1	us-east-1a	프런트엔드01	network_bytes_in	null	23
2019-12-04 19:00:10.000000000	us-east-1	us-east-1a	프런트엔드01	network_bytes_out	null	0
2019-12-04 19:00:16.000000000	us-east-1	us-east-1b	프런트엔드02	cpu_utilization	44.0	null
2019-12-04 19:00:16.000000000	us-east-1	us-east-1b	프런트엔드02	memory_utilization	64.2	null

Time	region	az	Hostname	measure_name	measure_value::double	measure_value::bigint
2019-12-04 19:00:16.000000000	us-east-1	us-east-1b	프런트엔드02	network_bytes_in	null	1,450
2019-12-04 19:00:16.000000000	us-east-1	us-east-1b	프런트엔드02	network_bytes_out	null	200
2019-12-04 19:00:40.000000000	us-east-1	us-east-1c	프런트엔드03	cpu_utilization	66.4	null
2019-12-04 19:00:40.000000000	us-east-1	us-east-1c	프런트엔드03	memory_utilization	86.3	null
2019-12-04 19:00:40.000000000	us-east-1	us-east-1c	프런트엔드03	network_bytes_in	null	300
2019-12-04 19:00:40.000000000	us-east-1	us-east-1c	프런트엔드03	network_bytes_out	null	423

지난 2시간 동안 특정 EC2 호스트의 평균, p90, p95 및 p99 CPU 사용률을 찾습니다.

```
SELECT region, az, hostname, BIN(time, 15s) AS binned_timestamp,
    ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.9), 2) AS p90_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.95), 2) AS p95_cpu_utilization,
    ROUND(APPROX_PERCENTILE(measure_value::double, 0.99), 2) AS p99_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
```

```
GROUP BY region, hostname, az, BIN(time, 15s)
ORDER BY binned_timestamp ASC
```

지난 2시간 동안 전체 플릿의 평균 CPU 사용률에 비해 CPU 사용률이 10% 이상 높은 EC2 호스트를 식별합니다.

```
WITH avg_fleet_utilization AS (
    SELECT COUNT(DISTINCT hostname) AS total_host_count, AVG(measure_value::double) AS
 fleet_avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND time > ago(2h)
), avg_per_host_cpu AS (
    SELECT region, az, hostname, AVG(measure_value::double) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND time > ago(2h)
    GROUP BY region, az, hostname
)
SELECT region, az, hostname, avg_cpu_utilization, fleet_avg_cpu_utilization
FROM avg_fleet_utilization, avg_per_host_cpu
WHERE avg_cpu_utilization > 1.1 * fleet_avg_cpu_utilization
ORDER BY avg_cpu_utilization DESC
```

지난 2시간 동안 특정 EC2 호스트에 대해 30초 간격으로 금지되는 평균 CPU 사용률을 찾습니다.

```
SELECT BIN(time, 30s) AS binned_timestamp, ROUND(AVG(measure_value::double), 2) AS
 avg_cpu_utilization
FROM "sampleDB".DevOps
WHERE measure_name = 'cpu_utilization'
    AND hostname = 'host-Hovjv'
    AND time > ago(2h)
GROUP BY hostname, BIN(time, 30s)
ORDER BY binned_timestamp ASC
```

지난 2시간 동안 특정 EC2 호스트에 대해 30초 간격으로 binned된 평균 CPU 사용률을 찾고 선형 보간을 사용하여 누락된 값을 채웁니다.

```
WITH binned_timeseries AS (
    SELECT hostname, BIN(time, 30s) AS binned_timestamp,
 ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization
```

```
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND hostname = 'host-Hovjv'
        AND time > ago(2h)
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_LINEAR(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

지난 2시간 동안 특정 EC2 호스트에 대해 30초 간격으로 비닝된 평균 CPU 사용률을 찾고 마지막 관측값으로 이월된 값을 기준으로 보간을 사용하여 누락된 값을 채웁니다.

```
WITH binned_timeseries AS (
    SELECT hostname, BIN(time, 30s) AS binned_timestamp,
 ROUND(AVG(measure_value::double), 2) AS avg_cpu_utilization
    FROM "sampleDB".DevOps
    WHERE measure_name = 'cpu_utilization'
        AND hostname = 'host-Hovjv'
        AND time > ago(2h)
    GROUP BY hostname, BIN(time, 30s)
), interpolated_timeseries AS (
    SELECT hostname,
        INTERPOLATE_LOCF(
            CREATE_TIME_SERIES(binned_timestamp, avg_cpu_utilization),
                SEQUENCE(min(binned_timestamp), max(binned_timestamp), 15s)) AS
 interpolated_avg_cpu_utilization
    FROM binned_timeseries
    GROUP BY hostname
)
SELECT time, ROUND(value, 2) AS interpolated_cpu
FROM interpolated_timeseries
CROSS JOIN UNNEST(interpolated_avg_cpu_utilization)
```

집계 함수가 있는 쿼리

다음은 집계 함수를 사용하여 쿼리를 설명하는 IoT 시나리오 예제 데이터 세트의 예입니다.

주제

예시 데이터

Timestream을 사용하면 하나 이상의 트럭 플릿의 위치, 연료 소비, 속도 및 부하 용량과 같은 IoT 센서 데이터를 저장하고 분석하여 효과적인 플릿 관리를 가능하게 할 수 있습니다. 다음은 트럭의 위치, 연료 소비, 속도 및 적재 용량과 같은 원격 측정을 저장하는 테이블 iot_trucks의 스키마 및 일부 데이터입니다.

Time	트럭_id	Make	모델	플릿	fuel_capacity, 연료_용량	load_capacity	measure_name	measure_value::double	measure_value::varchar
2019-12-4 19:00:00.0000000	1234567	GMC	아스트로	Alpha(알파)	100	500	fuel_reading	65.2	null
2019-12-4 19:00:00.0000000	1234567	GMC	아스트로	Alpha(알파)	100	500	로드	400.0	null
2019-12-4 19:00:00.0000000	1234567	GMC	아스트로	Alpha(알파)	100	500	속도	90.2	null
2019-12-4 19:00:00	1234567	GMC	아스트로	Alpha(알파)	100	500	location	null	47.6062 N,

Time	트럭_id	Make	모델	플릿	fuel_capacity, 연료_용량	load_capacity	measure_name	measure_value::double	measure_value::varchar
0000000									122.3321 W
2019-12-4 19:00:00 0000000	1234567	Kenworth	W900	Alpha(알파)	150	1000	fuel_reading	10.1	null
2019-12-4 19:00:00 0000000	1234567	Kenworth	W900	Alpha(알파)	150	1000	로드	950.3	null
2019-12-4 19:00:00 0000000	1234567	Kenworth	W900	Alpha(알파)	150	1000	속도	50.8	null
2019-12-4 19:00:00 0000000	1234567	Kenworth	W900	Alpha(알파)	150	1000	location	null	40.7128 도 N, 74.0060 도 W

쿼리 예제

플릿의 각 트럭에 대해 모니터링되는 모든 센서 속성 및 값의 목록을 가져옵니다.

```
SELECT
    truck_id,
    fleet,
    fuel_capacity,
    model,
    load_capacity,
    make,
```

```
    measure_name
FROM "sampleDB".IoT
GROUP BY truck_id, fleet, fuel_capacity, model, load_capacity, make, measure_name
```

지난 24시간 동안 플릿의 각 트럭에 대한 최신 연료 판독값을 얻습니다.

```
WITH latest_recorded_time AS (
    SELECT
        truck_id,
        max(time) as latest_time
    FROM "sampleDB".IoT
    WHERE measure_name = 'fuel-reading'
    AND time >= ago(24h)
    GROUP BY truck_id
)
SELECT
    b.truck_id,
    b.fleet,
    b.make,
    b.model,
    b.time,
    b.measure_value::double as last_reported_fuel_reading
FROM
latest_recorded_time a INNER JOIN "sampleDB".IoT b
ON a.truck_id = b.truck_id AND b.time = a.latest_time
WHERE b.measure_name = 'fuel-reading'
AND b.time > ago(24h)
ORDER BY b.truck_id
```

지난 48시간 동안 연료 부족(10% 미만)으로 실행된 트럭을 식별합니다.

```
WITH low_fuel_trucks AS (
    SELECT time, truck_id, fleet, make, model, (measure_value::double/
cast(fuel_capacity as double)*100) AS fuel_pct
    FROM "sampleDB".IoT
    WHERE time >= ago(48h)
    AND (measure_value::double/cast(fuel_capacity as double)*100) < 10
    AND measure_name = 'fuel-reading'
),
other_trucks AS (
SELECT time, truck_id, (measure_value::double/cast(fuel_capacity as double)*100) as
 remaining_fuel
    FROM "sampleDB".IoT
```

```
    WHERE time >= ago(48h)
    AND truck_id IN (SELECT truck_id FROM low_fuel_trucks)
    AND (measure_value::double/cast(fuel_capacity as double)*100) >= 10
    AND measure_name = 'fuel-reading'
),
trucks_that_refuelled AS (
    SELECT a.truck_id
    FROM low_fuel_trucks a JOIN other_trucks b
    ON a.truck_id = b.truck_id AND b.time >= a.time
)
SELECT DISTINCT truck_id, fleet, make, model, fuel_pct
FROM low_fuel_trucks
WHERE truck_id NOT IN (
    SELECT truck_id FROM trucks_that_refuelled
)
```

지난 주에 각 트럭의 평균 부하 및 최대 속도를 찾습니다.

```
SELECT
    bin(time, 1d) as binned_time,
    fleet,
    truck_id,
    make,
    model,
    AVG(
        CASE WHEN measure_name = 'load' THEN measure_value::double ELSE NULL END
    ) AS avg_load_tons,
    MAX(
        CASE WHEN measure_name = 'speed' THEN measure_value::double ELSE NULL END
    ) AS max_speed_mph
FROM "sampleDB".IoT
WHERE time >= ago(7d)
AND measure_name IN ('load', 'speed')
GROUP BY fleet, truck_id, make, model, bin(time, 1d)
ORDER BY truck_id
```

지난 한 주 동안 각 트럭에 대한 로드 효율성을 얻습니다.

```
WITH average_load_per_truck AS (
    SELECT
        truck_id,
        avg(measure_value::double)  AS avg_load
    FROM "sampleDB".IoT
```

```
    WHERE measure_name = 'load'
    AND time >= ago(7d)
    GROUP BY truck_id, fleet, load_capacity, make, model
),
truck_load_efficiency AS (
    SELECT
        a.truck_id,
        fleet,
        load_capacity,
        make,
        model,
        avg_load,
        measure_value::double,
        time,
        (measure_value::double*100)/avg_load as load_efficiency -- ,
 approx_percentile(avg_load_pct, DOUBLE '0.9')
    FROM "sampleDB".IoT a JOIN average_load_per_truck b
    ON a.truck_id = b.truck_id
    WHERE a.measure_name = 'load'
)
SELECT
    truck_id,
    time,
    load_efficiency
FROM truck_load_efficiency
ORDER BY truck_id, time
```

API 참조

이 섹션에는 Amazon Timestream에 대한 API 참조 설명서가 포함되어 있습니다.

Timestream에는 APIs쿼리와 쓰기라는 두 가지 가 있습니다.

- 쓰기API를 사용하면 테이블 생성, 리소스 태그 지정 및 Timestream에 레코드 쓰기와 같은 작업을 수행할 수 있습니다.
- 쿼리API를 사용하면 쿼리 작업을 수행할 수 있습니다.

> **Note**
>
> 둘 다 DescribeEndpoints 작업을 APIs 포함합니다. 쿼리와 쓰기 모두 DescribeEndpoints 작업은 동일합니다.

데이터 유형, 일반적인 오류 및 파라미터와 함께 API 아래에서 각각에 대해 자세히 알아볼 수 있습니다.

> **Note**
>
> 모든 AWS 서비스에 공통적인 오류 코드는 AWS 지원 섹션을 참조하세요.

주제

작업

Amazon Timestream Write에서 지원하는 작업은 다음과 같습니다.

- CreateBatchLoadTask
- CreateDatabase
- CreateTable
- DeleteDatabase
- DeleteTable
- DescribeBatchLoadTask
- DescribeDatabase
- DescribeEndpoints
- DescribeTable
- ListBatchLoadTasks

- ListDatabases
- ListTables
- ListTagsForResource
- ResumeBatchLoadTask
- TagResource
- UntagResource
- UpdateDatabase
- UpdateTable
- WriteRecords

Amazon Timestream Query에서 지원하는 작업은 다음과 같습니다.

- CancelQuery
- CreateScheduledQuery
- DeleteScheduledQuery
- DescribeAccountSettings
- DescribeEndpoints
- DescribeScheduledQuery
- ExecuteScheduledQuery
- ListScheduledQueries
- ListTagsForResource
- PrepareQuery
- Query
- TagResource
- UntagResource
- UpdateAccountSettings
- UpdateScheduledQuery

Amazon Timestream 쓰기

Amazon Timestream Write에서 지원하는 작업은 다음과 같습니다.

- CreateBatchLoadTask
- CreateDatabase
- CreateTable
- DeleteDatabase
- DeleteTable
- DescribeBatchLoadTask
- DescribeDatabase
- DescribeEndpoints
- DescribeTable
- ListBatchLoadTasks
- ListDatabases
- ListTables
- ListTagsForResource
- ResumeBatchLoadTask
- TagResource
- UntagResource
- UpdateDatabase
- UpdateTable
- WriteRecords

CreateBatchLoadTask
서비스: Amazon Timestream Write

새 Timestream 배치 로드 작업을 생성합니다. 배치 로드 태스크는 S3 위치의 CSV 소스에서 데이터를 처리하고 Timestream 테이블에 씁니다. 소스에서 대상으로의 매핑은 배치 로드 태스크에 정의됩니다. 오류 및 이벤트는 S3 위치의 보고서에 기록됩니다. 보고서의 경우 AWS KMS 키가 지정되지 않은 경우 SSE_S3가 옵션일 때 보고서가 S3 관리형 키로 암호화됩니다. 그렇지 않으면 오류가 발생합니다. 자세한 내용은 AWS 관리형 키를 참조하세요. 서비스 할당량이 적용됩니다. 자세한 내용은 코드 샘플 섹션을 참조하세요.

구문 요청

```
{
   "ClientToken": "string",
   "DataModelConfiguration": {
      "DataModel": {
         "DimensionMappings": [
            {
               "DestinationColumn": "string",
               "SourceColumn": "string"
            }
         ],
         "MeasureNameColumn": "string",
         "MixedMeasureMappings": [
            {
               "MeasureName": "string",
               "MeasureValueType": "string",
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "SourceColumn": "string",
               "TargetMeasureName": "string"
            }
         ],
         "MultiMeasureMappings": {
            "MultiMeasureAttributeMappings": [
               {
                  "MeasureValueType": "string",
                  "SourceColumn": "string",
```

```
                    "TargetMultiMeasureAttributeName": "string"
                }
            ],
            "TargetMultiMeasureName": "string"
        },
        "TimeColumn": "string",
        "TimeUnit": "string"
    },
    "DataModelS3Configuration": {
        "BucketName": "string",
        "ObjectKey": "string"
    }
},
"DataSourceConfiguration": {
    "CsvConfiguration": {
        "ColumnSeparator": "string",
        "EscapeChar": "string",
        "NullValue": "string",
        "QuoteChar": "string",
        "TrimWhiteSpace": boolean
    },
    "DataFormat": "string",
    "DataSourceS3Configuration": {
        "BucketName": "string",
        "ObjectKeyPrefix": "string"
    }
},
"RecordVersion": number,
"ReportConfiguration": {
    "ReportS3Configuration": {
        "BucketName": "string",
        "EncryptionOption": "string",
        "KmsKeyId": "string",
        "ObjectKeyPrefix": "string"
    }
},
"TargetDatabaseName": "string",
"TargetTableName": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ClientToken

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

필수 여부: 아니요

DataModelConfiguration

유형: DataModelConfiguration 객체

필수 여부: 아니요

DataSourceConfiguration

배치 로드 작업의 데이터 소스에 대한 구성 세부 정보를 정의합니다.

유형: DataSourceConfiguration 객체

필수 여부: 예

RecordVersion

유형: Long

필수 여부: 아니요

ReportConfiguration

배치 로드 태스크에 대한 구성 보고 여기에는 오류 보고서가 저장되는 위치에 대한 세부 정보가 포함되어 있습니다.

유형: ReportConfiguration 객체

필수 여부: 예

TargetDatabaseName

배치 로드 작업의 대상 Timestream 데이터베이스입니다.

유형: String

Pattern: `[a-zA-Z0-9_.-]+`

필수 여부: 예

TargetTableName

배치 로드 작업에 대한 대상 Timestream 테이블입니다.

유형: String

Pattern: [a-zA-Z0-9_.-]+

필수 여부: 예

응답 구문

```
{
   "TaskId": "string"
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

TaskId

배치 로드 작업의 ID입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이 32.

패턴: [A-Z0-9]+

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

ConflictException

Timestream에 이미 존재하는 리소스가 포함되어 있어 이 요청을 처리할 수 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ServiceQuotaExceededException

이 계정에 대한 리소스의 인스턴스 할당량을 초과했습니다.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

CreateDatabase
서비스: Amazon Timestream Write

새 Timestream 데이터베이스를 생성합니다. AWS KMS 키가 지정되지 않은 경우 데이터베이스는 계정에 있는 Timestream 관리형 AWS KMS 키로 암호화됩니다. 자세한 내용은 AWS 관리형 키를 참조하세요. 서비스 할당량이 적용됩니다. 자세한 내용은 코드 샘플 섹션을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string",
   "KmsKeyId": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256입니다.

Pattern: `[a-zA-Z0-9_.-]+`

필수 여부: 예

KmsKeyId

데이터베이스의 AWS KMS 키입니다. AWS KMS 키가 지정되지 않은 경우 데이터베이스는 계정에 있는 Timestream 관리형 AWS KMS 키로 암호화됩니다. 자세한 내용은 AWS 관리형 키를 참조하세요.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

Tags

테이블에 레이블을 지정할 키-값 페어 목록입니다.

유형: Tag객체 어레이

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 여부: 아니요

응답 구문

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Database

새로 생성된 Timestream 데이터베이스입니다.

유형: Database 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

ConflictException

Timestream에 이미 존재하는 리소스가 포함되어 있어 이 요청을 처리할 수 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ServiceQuotaExceededException

이 계정에 대한 리소스의 인스턴스 할당량을 초과했습니다.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

CreateTable
서비스: Amazon Timestream Write

계정의 기존 데이터베이스에 새 테이블을 추가합니다. 에서 AWS 계정테이블 이름은 동일한 데이터베이스에 있는 경우 각 리전 내에서 최소한 고유해야 합니다. 테이블이 별도의 데이터베이스에 있는 경우 동일한 리전에 동일한 테이블 이름이 있을 수 있습니다. 테이블을 생성하는 동안 테이블 이름, 데이터베이스 이름 및 보존 속성을 지정해야 합니다. 서비스 할당량이 적용됩니다. 자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string",
   "MagneticStoreWriteProperties": {
      "EnableMagneticStoreWrites": boolean,
      "MagneticStoreRejectedDataLocation": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      }
   },
   "RetentionProperties": {
      "MagneticStoreRetentionPeriodInDays": number,
      "MemoryStoreRetentionPeriodInHours": number
   },
   "Schema": {
      "CompositePartitionKey": [
         {
            "EnforcementInRecord": "string",
            "Name": "string",
            "Type": "string"
         }
      ]
   },
   "TableName": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
```

```
    ]
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256입니다.

Pattern: `[a-zA-Z0-9_.-]+`

필수 여부: 예

MagneticStoreWriteProperties

마그네틱 스토어 쓰기를 활성화할 때 테이블에 설정할 속성이 들어 있습니다.

유형: MagneticStoreWriteProperties 객체

필수 여부: 아니요

RetentionProperties

시계열 데이터를 메모리 스토어와 마그네틱 스토어에 저장해야 하는 기간입니다.

유형: RetentionProperties 객체

필수 여부: 아니요

Schema

테이블의 스키마입니다.

유형: Schema 객체

필수 여부: 아니요

TableName

Timestream 테이블의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256입니다.

Pattern: `[a-zA-Z0-9_.-]+`

필수 여부: 예

Tags

테이블에 레이블을 지정할 키-값 페어 목록입니다.

유형: Tag객체 어레이

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 여부: 아니요

응답 구문

```
{
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
```

```
            "MemoryStoreRetentionPeriodInHours": number
         },
         "Schema": {
            "CompositePartitionKey": [
               {
                  "EnforcementInRecord": "string",
                  "Name": "string",
                  "Type": "string"
               }
            ]
         },
         "TableName": "string",
         "TableStatus": "string"
      }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Table

새로 생성된 Timestream 테이블입니다.

유형: Table 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

ConflictException

Timestream에 이미 존재하는 리소스가 포함되어 있어 이 요청을 처리할 수 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ServiceQuotaExceededException

이 계정에 대한 리소스의 인스턴스 할당량을 초과했습니다.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DeleteDatabase
서비스: Amazon Timestream Write

지정된 Timestream 데이터베이스를 삭제합니다. 이는 되돌릴 수 없는 작업입니다. 데이터베이스를 삭제한 후에는 테이블에서 시계열 데이터를 복구할 수 없습니다.

> **Note**
>
> 데이터베이스의 모든 테이블을 먼저 삭제해야 합니다. 그렇지 않으면 ValidationException 오류가 발생합니다.
> 분산 재시도의 특성으로 인해 작업은 성공 또는 를 반환할 수 있습니다 ResourceNotFoundException. 고객은 이를 동등한 것으로 간주해야 합니다.

자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

삭제할 Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

Response Elements

작업이 성공하면 서비스는 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DeleteTable
서비스: Amazon Timestream Write

지정된 Timestream 테이블을 삭제합니다. 이는 되돌릴 수 없는 작업입니다. Timestream 데이터베이스 테이블이 삭제된 후에는 테이블에 저장된 시계열 데이터를 복구할 수 없습니다.

> **Note**
>
> 분산 재시도의 특성으로 인해 작업은 성공 또는 를 반환할 수 있습니다 ResourceNotFoundException. 고객은 이를 동등한 것으로 간주해야 합니다.

자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string",
   "TableName": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

Timestream 데이터베이스를 삭제할 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

TableName

삭제할 Timestream 테이블의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

Response Elements

작업이 성공하면 서비스가 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DescribeBatchLoadTask
서비스: Amazon Timestream Write

구성, 매핑, 진행 상황 및 기타 세부 정보를 포함하여 배치 로드 작업에 대한 정보를 반환합니다. 서비스 할당량이 적용됩니다. 자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "TaskId": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

TaskId

배치 로드 작업의 ID입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이 32.

Pattern: [A-Z0-9]+

필수 여부: 예

응답 구문

```
{
   "BatchLoadTaskDescription": {
      "CreationTime": number,
      "DataModelConfiguration": {
         "DataModel": {
            "DimensionMappings": [
               {
                  "DestinationColumn": "string",
                  "SourceColumn": "string"
               }
            ],
```

```
            "MeasureNameColumn": "string",
            "MixedMeasureMappings": [
               {
                  "MeasureName": "string",
                  "MeasureValueType": "string",
                  "MultiMeasureAttributeMappings": [
                     {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                     }
                  ],
                  "SourceColumn": "string",
                  "TargetMeasureName": "string"
               }
            ],
            "MultiMeasureMappings": {
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "TargetMultiMeasureName": "string"
            },
            "TimeColumn": "string",
            "TimeUnit": "string"
         },
         "DataModelS3Configuration": {
            "BucketName": "string",
            "ObjectKey": "string"
         }
      },
      "DataSourceConfiguration": {
         "CsvConfiguration": {
            "ColumnSeparator": "string",
            "EscapeChar": "string",
            "NullValue": "string",
            "QuoteChar": "string",
            "TrimWhiteSpace": boolean
         },
         "DataFormat": "string",
         "DataSourceS3Configuration": {
```

```
            "BucketName": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "ErrorMessage": "string",
      "LastUpdatedTime": number,
      "ProgressReport": {
         "BytesMetered": number,
         "FileFailures": number,
         "ParseFailures": number,
         "RecordIngestionFailures": number,
         "RecordsIngested": number,
         "RecordsProcessed": number
      },
      "RecordVersion": number,
      "ReportConfiguration": {
         "ReportS3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "ResumableUntil": number,
      "TargetDatabaseName": "string",
      "TargetTableName": "string",
      "TaskId": "string",
      "TaskStatus": "string"
   }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

BatchLoadTaskDescription

배치 로드 작업에 대한 설명입니다.

유형: BatchLoadTaskDescription 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용

- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DescribeDatabase
서비스: Amazon Timestream Write

데이터베이스 이름, 데이터베이스가 생성된 시간, 데이터베이스 내에서 발견된 총 테이블 수를 포함하여 데이터베이스에 대한 정보를 반환합니다. 서비스 할당량이 적용됩니다. 자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

응답 구문

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Database

Timestream 테이블의 이름입니다.

유형: Database 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DescribeEndpoints
서비스: Amazon Timestream Write

Timestream에서 API 호출할 수 있는 사용 가능한 엔드포인트 목록을 반환합니다. 이 API 작업은 쓰기 및 쿼리 를 통해 사용할 수 있습니다APIs.

TimestreamSDKs은 서비스 엔드포인트의 관리 및 매핑을 포함하여 서비스의 아키텍처와 투명하게 작동하도록 설계되었으므로 가 아닌 한 이 API 작업을 사용하지 않는 것이 좋습니다.

- VPC Timestream에서 엔드포인트(AWS PrivateLink)를 사용하고 있습니다.
- 애플리케이션에서 아직 SDK 지원되지 않는 프로그래밍 언어를 사용합니다.
- 클라이언트 측 구현을 더 잘 제어해야 합니다.

를 사용하고 구현하는 방법과 시기에 대한 자세한 내용은 엔드포인트 검색 패턴 섹션을 DescribeEndpoints참조하세요. https://docs.aws.amazon.com/timestream/latest/developerguide/Using.API.html#Using-API.endpoint-discovery

Response Syntax

```
{
   "Endpoints": [
      {
         "Address": "string",
         "CachePeriodInMinutes": number
      }
   ]
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Endpoints

`DescribeEndpoints` 요청이 이루어지면 `Endpoints` 객체가 반환됩니다.

타입: Endpoint 객체 배열

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DescribeTable
서비스: Amazon Timestream Write

테이블 이름, 데이터베이스 이름, 메모리 스토어 및 마그네틱 스토어의 보존 기간을 포함하여 테이블에 대한 정보를 반환합니다. 서비스 할당량이 적용됩니다. 자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string",
   "TableName": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

TableName

Timestream 테이블의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

응답 구문

```
{
   "Table": {
```

```
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Table

Timestream 테이블입니다.

유형: Table 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

ListBatchLoadTasks
서비스: Amazon Timestream Write

배치 로드 작업 목록과 함께 이름, 상태, 작업을 다시 사용할 수 있는 시기 및 기타 세부 정보를 제공합니다. 자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "MaxResults": number,
   "NextToken": "string",
   "TaskStatus": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

MaxResults

출력에서 반환할 총 항목 수입니다. 사용 가능한 총 항목 수가 지정된 값보다 많은 경우 출력에 NextToken 가 제공됩니다. 페이지 매김을 재개하려면 NextToken 값을 후속 API 호출의 인수로 제공합니다.

타입: 정수

유효 범위: 최소값 1. 최댓값은 100입니다.

필수 여부: 아니요

NextToken

페이지 매김을 시작할 위치를 지정하기 위한 토큰입니다. 이는 이전에 잘린 응답 NextToken 의 입니다.

유형: 문자열

필수 항목 여부: 아니요

TaskStatus

배치 로드 작업의 상태입니다.

타입: 문자열

유효 값: CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME

필수 항목 여부: 아니요

응답 구문

```
{
   "BatchLoadTasks": [
      {
         "CreationTime": number,
         "DatabaseName": "string",
         "LastUpdatedTime": number,
         "ResumableUntil": number,
         "TableName": "string",
         "TaskId": "string",
         "TaskStatus": "string"
      }
   ],
   "NextToken": "string"
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

BatchLoadTasks

배치 로드 작업 세부 정보 목록입니다.

유형: BatchLoadTask객체 어레이

NextToken

페이지 매김을 시작할 위치를 지정하기 위한 토큰입니다. 다음 를 제공합니다 ListBatchLoadTasksRequest.

유형: 문자열

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용

- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

ListDatabases
서비스: Amazon Timestream Write

Timestream 데이터베이스 목록을 반환합니다. 서비스 할당량이 적용됩니다. 자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

MaxResults

출력에서 반환할 총 항목 수입니다. 사용 가능한 총 항목 수가 지정된 값보다 많은 경우 출력에 NextToken 가 제공됩니다. 페이지 매김을 재개하려면 NextToken 값을 후속 API 호출의 인수로 제공합니다.

타입: 정수

유효한 범위: 최소값은 1. 최대값은 20.

필수 여부: 아니요

NextToken

페이지 매김 토큰입니다. 페이지 매김을 재개하려면 NextToken 값을 후속 API 호출의 인수로 제공합니다.

유형: 문자열

필수사항: 아니요

응답 구문

```
{
```

```
   "Databases": [
      {
         "Arn": "string",
         "CreationTime": number,
         "DatabaseName": "string",
         "KmsKeyId": "string",
         "LastUpdatedTime": number,
         "TableCount": number
      }
   ],
   "NextToken": "string"
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Databases

데이터베이스 이름 목록입니다.

유형: Database객체 어레이

NextToken

페이지 매김 토큰입니다. 응답이 잘리면 이 파라미터가 반환됩니다.

유형: 문자열

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

ListTables
서비스: Amazon Timestream Write

각 테이블의 이름, 상태 및 보존 속성과 함께 테이블 목록을 제공합니다. 자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string",
   "MaxResults": number,
   "NextToken": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 아니요

MaxResults

출력에서 반환할 총 항목 수입니다. 사용 가능한 총 항목 수가 지정된 값보다 많은 경우 출력에 NextToken 가 제공됩니다. 페이지 매김을 재개하려면 NextToken 값을 후속 API 호출의 인수로 제공합니다.

타입: 정수

유효한 범위: 최소값은 1. 최대값은 20.

필수 여부: 아니요

NextToken

페이지 매김 토큰입니다. 페이지 매김을 재개하려면 NextToken 값을 후속 API 호출의 인수로 제공합니다.

유형: 문자열

필수사항: 아니요

응답 구문

```
{
   "NextToken": "string",
   "Tables": [
      {
         "Arn": "string",
         "CreationTime": number,
         "DatabaseName": "string",
         "LastUpdatedTime": number,
         "MagneticStoreWriteProperties": {
            "EnableMagneticStoreWrites": boolean,
            "MagneticStoreRejectedDataLocation": {
               "S3Configuration": {
                  "BucketName": "string",
                  "EncryptionOption": "string",
                  "KmsKeyId": "string",
                  "ObjectKeyPrefix": "string"
               }
            }
         },
         "RetentionProperties": {
            "MagneticStoreRetentionPeriodInDays": number,
            "MemoryStoreRetentionPeriodInHours": number
         },
         "Schema": {
            "CompositePartitionKey": [
               {
                  "EnforcementInRecord": "string",
                  "Name": "string",
                  "Type": "string"
               }
            ]
         },
```

```
            "TableName": "string",
            "TableStatus": "string"
        }
    ]
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

NextToken

페이지 매김을 시작할 위치를 지정하기 위한 토큰입니다. 이는 이전에 잘린 응답 NextToken 의 입니다.

유형: 문자열

Tables

테이블 목록입니다.

타입: Table 객체 배열

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

ListTagsForResource
서비스: Amazon Timestream Write

Timestream 리소스의 모든 태그를 나열합니다.

구문 요청

```
{
   "ResourceARN": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ResourceARN

나열할 태그가 있는 Timestream 리소스입니다. 이 값은 Amazon 리소스 이름()입니다ARN.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 1011입니다.

필수 항목 여부: 예

응답 구문

```
{
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Tags

현재 Timestream 리소스와 연결된 태그입니다.

유형: Tag객체 어레이

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

ResumeBatchLoadTask
서비스: Amazon Timestream Write

구문 요청

```
{
   "TaskId": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

TaskId

재개할 배치 로드 작업의 ID입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이 32.

Pattern: `[A-Z0-9]+`

필수 여부: 예

Response Elements

작업이 성공하면 서비스가 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용

- AWS SDK Python용
- AWS SDK Ruby V3용

TagResource
서비스: Amazon Timestream Write

태그 세트를 Timestream 리소스와 연결합니다. 그런 다음 이러한 사용자 정의 태그를 활성화하여 비용 할당 추적을 위해 Billing and Cost Management 콘솔에 표시되도록 할 수 있습니다.

구문 요청

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ResourceARN

태그를 추가해야 하는 Timestream 리소스를 식별합니다. 이 값은 Amazon 리소스 이름()입니다 ARN.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 1011입니다.

필수 여부: 예

Tags

Timestream 리소스에 할당할 태그입니다.

유형: Tag객체 어레이

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 여부: 예

Response Elements

작업이 성공하면 서비스가 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ServiceQuotaExceededException

이 계정에 대한 리소스의 인스턴스 할당량을 초과했습니다.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface

- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

UntagResource
서비스: Amazon Timestream Write

Timestream 리소스에서 태그 연결을 제거합니다.

구문 요청

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ResourceARN

태그를 제거할 Timestream 리소스입니다. 이 값은 Amazon 리소스 이름()입니다ARN.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 1011입니다.

필수 여부: 예

TagKeys

태그 키 목록입니다. 키가 이 목록의 멤버인 리소스의 기존 태그는 Timestream 리소스에서 제거됩니다.

타입: 문자열 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

길이 제약: 최소 길이 1. 최대 길이는 128.

필수 여부: 예

Response Elements

작업이 성공하면 서비스는 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ServiceQuotaExceededException

이 계정에 대한 리소스의 인스턴스 할당량을 초과했습니다.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용

- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

UpdateDatabase
서비스: Amazon Timestream Write

기존 데이터베이스의 AWS KMS 키를 수정합니다. 데이터베이스를 업데이트하는 동안 사용할 새 AWS KMS 키의 데이터베이스 이름과 식별자()를 지정해야 합니다KmsKeyId. 동시 UpdateDatabase 요청이 있는 경우 첫 번째 라이터가 승리합니다.

자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string",
   "KmsKeyId": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

KmsKeyId

데이터베이스에 저장된 데이터를 암호화하는 데 사용할 새 AWS KMS 키(KmsKeyId)의 식별자입니다. KmsKeyId 현재 데이터베이스에 등록된 이 요청KmsKeyId의 와 동일한 경우 업데이트가 발생하지 않습니다.

다음 중 하나를 KmsKeyId 사용하여 를 지정할 수 있습니다.

- 키 ID: 1234abcd-12ab-34cd-56ef-1234567890ab
- 키ARN: arn:aws:kms:us-east-1:111122223333:key/1234abcd-12ab-34cd-56ef-1234567890ab

- 별칭 이름: `alias/ExampleAlias`
- 별칭ARN: `arn:aws:kms:us-east-1:111122223333:alias/ExampleAlias`

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 항목 여부: 예

응답 구문

```
{
   "Database": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "KmsKeyId": "string",
      "LastUpdatedTime": number,
      "TableCount": number
   }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Database

테이블의 최상위 컨테이너입니다. 데이터베이스와 테이블은 Amazon Timestream의 기본 관리 개념입니다. 데이터베이스의 모든 테이블은 동일한 AWS KMS 키로 암호화됩니다.

유형: Database 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ServiceQuotaExceededException

이 계정에 대한 리소스의 인스턴스 할당량을 초과했습니다.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET

- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

UpdateTable
서비스: Amazon Timestream Write

Timestream 테이블에 대한 메모리 스토어 및 마그네틱 스토어의 보존 기간을 수정합니다. 보존 기간의 변경 사항은 즉시 적용됩니다. 예를 들어 메모리 스토어의 보존 기간이 처음에 2시간으로 설정된 다음 24시간으로 변경된 경우 메모리 스토어는 24시간의 데이터를 보관할 수 있지만 이 변경 사항이 적용된 후 22시간이 지나면 24시간의 데이터로 채워집니다. Timestream은 마그네틱 스토어에서 데이터를 검색하여 메모리 스토어를 채우지 않습니다.

자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "DatabaseName": "string",
   "MagneticStoreWriteProperties": {
      "EnableMagneticStoreWrites": boolean,
      "MagneticStoreRejectedDataLocation": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "KmsKeyId": "string",
            "ObjectKeyPrefix": "string"
         }
      }
   },
   "RetentionProperties": {
      "MagneticStoreRetentionPeriodInDays": number,
      "MemoryStoreRetentionPeriodInHours": number
   },
   "Schema": {
      "CompositePartitionKey": [
         {
            "EnforcementInRecord": "string",
            "Name": "string",
            "Type": "string"
         }
      ]
   },
   "TableName": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

DatabaseName

Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

MagneticStoreWriteProperties

마그네틱 스토어 쓰기를 활성화할 때 테이블에 설정할 속성이 들어 있습니다.

유형: MagneticStoreWriteProperties 객체

필수 여부: 아니요

RetentionProperties

메모리 스토어와 마그네틱 스토어의 보존 기간입니다.

유형: RetentionProperties 객체

필수 여부: 아니요

Schema

테이블의 스키마입니다.

유형: Schema 객체

필수 여부: 아니요

TableName

Timestream 테이블의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

응답 구문

```
{
   "Table": {
      "Arn": "string",
      "CreationTime": number,
      "DatabaseName": "string",
      "LastUpdatedTime": number,
      "MagneticStoreWriteProperties": {
         "EnableMagneticStoreWrites": boolean,
         "MagneticStoreRejectedDataLocation": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "KmsKeyId": "string",
               "ObjectKeyPrefix": "string"
            }
         }
      },
      "RetentionProperties": {
         "MagneticStoreRetentionPeriodInDays": number,
         "MemoryStoreRetentionPeriodInHours": number
      },
      "Schema": {
         "CompositePartitionKey": [
            {
               "EnforcementInRecord": "string",
               "Name": "string",
               "Type": "string"
            }
         ]
      },
      "TableName": "string",
      "TableStatus": "string"
   }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Table

업데이트된 Timestream 테이블입니다.

유형: Table 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

WriteRecords
서비스: Amazon Timestream Write

시계열 데이터를 Timestream에 쓸 수 있습니다. 단일 데이터 포인트 또는 시스템에 삽입할 데이터 포인트 배치를 지정할 수 있습니다. Timestream은 데이터베이스에 쓰기를 호출할 때 지정한 데이터 포인트의 차원 이름 및 데이터 유형을 기반으로 Timestream 테이블의 열 이름 및 데이터 유형을 자동으로 감지하는 유연한 스키마를 제공합니다.

Timestream은 최종 일관성 읽기 의미 체계를 지원합니다. 즉, Timestream에 데이터 배치를 작성한 직후 데이터를 쿼리할 때 쿼리 결과에는 최근에 완료된 쓰기 작업의 결과가 반영되지 않을 수 있습니다. 결과에는 일부 오래된 데이터도 포함될 수 있습니다. 잠시 후 쿼리 요청을 반복하면 결과가 최신 데이터를 반환해야 합니다. 서비스 할당량이 적용됩니다.

자세한 내용은 코드 샘플을 참조하세요.

업서트

`WriteRecords` 요청에 `Version` 파라미터를 사용하여 데이터 포인트를 업데이트할 수 있습니다. Timestream은 각 레코드와 함께 버전 번호를 추적합니다. 는 요청의 레코드에 지정되지 않은 `1` 경우 `Version` 기본적으로 로 설정됩니다. Timestream은 기존 레코드의 측정값을 해당 레코드에 대해 더 높은 `Version` 수의 쓰기 요청을 수신할 `Version` 때와 함께 업데이트합니다. 측정값이 기존 레코드의 값과 동일한 업데이트 요청을 수신하면 Timestream은 기존 값보다 큰 `Version`경우 를 계속 업데이트합니다`Version`. 값이 `Version` 지속적으로 증가하는 한 데이터 포인트를 원하는 만큼 업데이트할 수 있습니다.

예를 들어 요청에 표시하지 않고 새 레코드`Version`를 작성한다고 가정해 보겠습니다. Timestream은 이 레코드를 저장하고 `Version`로 설정합니다`1`. 이제 측정값이 다른 동일한 레코드의 `WriteRecords` 요청으로 이 레코드를 업데이트하려고 하지만 이전과 마찬가지로 를 제공하지 않는다고 가정해 보겠습니다`Version`. 이 경우 업데이트된 레코드의 버전이 기존 버전 값보다 크지 `RejectedRecordsException` 않으므로 Timestream은 이 업데이트를 로 거부합니다.

그러나 를 로 `Version` 설정하여 업데이트 요청을 재전송하는 경우 `2`Timestream은 레코드 값을 성공적으로 업데이트하고 를 로 `Version` 설정합니다`2`. 다음으로, 이 레코드와 측정값은 동일하지만 로 `Version` 설정된 `WriteRecords` 요청을 전송했다고 가정해 보겠습니다`3`. 이 경우 Timestream은 `Version`로만 업데이트합니다`3`. 추가 업데이트 시 보다 큰 버전 번호를 전송해야 `3`하거나 업데이트 요청에 가 수신됩니다`RejectedRecordsException`.

구문 요청

```
{
   "CommonAttributes": {
```

```
      "Dimensions": [
         {
            "DimensionValueType": "string",
            "Name": "string",
            "Value": "string"
         }
      ],
      "MeasureName": "string",
      "MeasureValue": "string",
      "MeasureValues": [
         {
            "Name": "string",
            "Type": "string",
            "Value": "string"
         }
      ],
      "MeasureValueType": "string",
      "Time": "string",
      "TimeUnit": "string",
      "Version": number
   },
   "DatabaseName": "string",
   "Records": [
      {
         "Dimensions": [
            {
               "DimensionValueType": "string",
               "Name": "string",
               "Value": "string"
            }
         ],
         "MeasureName": "string",
         "MeasureValue": "string",
         "MeasureValues": [
            {
               "Name": "string",
               "Type": "string",
               "Value": "string"
            }
         ],
         "MeasureValueType": "string",
         "Time": "string",
         "TimeUnit": "string",
         "Version": number
```

```
        }
    ],
    "TableName": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

CommonAttributes

요청의 모든 레코드에서 공유된 공통 측정값, 차원, 시간 및 버전 속성을 포함하는 레코드입니다. 지정된 측정값 및 차원 속성은 데이터가 Timestream에 기록될 때 레코드 객체의 측정값 및 차원 속성과 병합됩니다. 차원이 겹치지 않거나 이 발생ValidationException될 수 있습니다. 즉, 레코드에는 고유한 이름이 있는 차원이 포함되어야 합니다.

유형: Record 객체

필수 여부: 아니요

DatabaseName

Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

Records

각 시계열 데이터 포인트에 대한 고유한 측정값, 차원, 시간 및 버전 속성을 포함하는 레코드 배열입니다.

유형: Record객체 어레이

어레이 멤버: 최소 항목 수 1개. 최대수는 100개입니다.

필수 여부: 예

TableName

Timestream 테이블의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 예

응답 구문

```
{
   "RecordsIngested": {
      "MagneticStore": number,
      "MemoryStore": number,
      "Total": number
   }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

RecordsIngested

이 요청에 의해 수집된 레코드에 대한 정보입니다.

유형: RecordsIngested 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 Timestream에서 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 500

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

RejectedRecordsException

WriteRecords 다음과 같은 경우 이 예외가 발생합니다.

- 동일한 차원, 타임스탬프 및 측정값 이름을 가진 레코드가 여러 개 있지만 다음과 같은 중복 데이터가 있는 레코드:
  - 측정값이 다릅니다.
  - 요청에 버전이 없거나 새 레코드의 버전 값이 기존 값보다 작거나 같습니다.

  이 경우 Timestream이 데이터를 거부하면 `RejectedRecords` 응답의 `ExistingVersion` 필드에 현재 레코드의 버전이 표시됩니다. 업데이트를 강제로 적용하려면 레코드 세트의 버전이 보다 큰 값으로 요청을 재전송할 수 있습니다`ExistingVersion`.
- 메모리 스토어의 보존 기간을 벗어나는 타임스탬프가 있는 레코드입니다.
- Timestream에서 정의한 한도를 초과하는 차원 또는 측정값이 있는 레코드입니다.

자세한 내용은 Amazon Timestream 개발자 안내서의 할당량을 참조하세요.

HTTP 상태 코드: 400

ResourceNotFoundException

작업이 존재하지 않는 리소스에 액세스하려고 했습니다. 리소스가 올바르게 지정되지 않았거나 상태가 가 아닐 수 있습니다ACTIVE.

HTTP 상태 코드: 400

ThrottlingException

사용자가 너무 많은 요청을 했으며 서비스 할당량을 초과했습니다. 요청에 병목 현상이 발생했습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

Amazon Timestream 쿼리

Amazon Timestream Query에서 지원하는 작업은 다음과 같습니다.

- CancelQuery
- CreateScheduledQuery
- DeleteScheduledQuery
- DescribeAccountSettings
- DescribeEndpoints
- DescribeScheduledQuery
- ExecuteScheduledQuery
- ListScheduledQueries
- ListTagsForResource
- PrepareQuery
- Query
- TagResource
- UntagResource
- UpdateAccountSettings
- UpdateScheduledQuery

CancelQuery
서비스: Amazon Timestream Query

발급된 쿼리를 취소합니다. 취소 요청은 취소 요청이 발행되기 전에 쿼리 실행이 완료되지 않은 경우에만 제공됩니다. 취소는 이기적인 작업이므로 후속 취소 요청은 쿼리가 이미 취소되었음을 `CancellationMessage`나타내는 를 반환합니다. 자세한 내용은 코드 샘플을 참조하세요.

구문 요청

```
{
   "QueryId": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

QueryId

취소해야 하는 쿼리의 ID입니다. `QueryID`는 쿼리 결과의 일부로 반환됩니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

Pattern: `[a-zA-Z0-9]+`

필수 여부: 예

응답 구문

```
{
   "CancellationMessage": "string"
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

CancellationMessage

에서 지정한 쿼리에 대한 CancelQuery 요청이 이미 발급되면 QueryId 가 반환CancellationMessage됩니다.

유형: 문자열

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

CreateScheduledQuery
서비스: Amazon Timestream Query

구성된 일정에 따라 사용자를 대신하여 실행될 예약된 쿼리를 생성합니다. Timestream은 쿼리를 실행하기 위해ScheduledQueryExecutionRoleArn 파라미터의 일부로 제공된 실행 역할을 수임합니다. NotificationConfiguration 파라미터를 사용하여 예약된 쿼리 작업에 대한 알림을 구성할 수 있습니다.

구문 요청

```
{
   "ClientToken": "string",
   "ErrorReportConfiguration": {
      "S3Configuration": {
         "BucketName": "string",
         "EncryptionOption": "string",
         "ObjectKeyPrefix": "string"
      }
   },
   "KmsKeyId": "string",
   "Name": "string",
   "NotificationConfiguration": {
      "SnsConfiguration": {
         "TopicArn": "string"
      }
   },
   "QueryString": "string",
   "ScheduleConfiguration": {
      "ScheduleExpression": "string"
   },
   "ScheduledQueryExecutionRoleArn": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ],
   "TargetConfiguration": {
      "TimestreamConfiguration": {
         "DatabaseName": "string",
         "DimensionMappings": [
            {
               "DimensionValueType": "string",
```

```
                "Name": "string"
            }
        ],
        "MeasureNameColumn": "string",
        "MixedMeasureMappings": [
            {
                "MeasureName": "string",
                "MeasureValueType": "string",
                "MultiMeasureAttributeMappings": [
                    {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                    }
                ],
                "SourceColumn": "string",
                "TargetMeasureName": "string"
            }
        ],
        "MultiMeasureMappings": {
            "MultiMeasureAttributeMappings": [
                {
                    "MeasureValueType": "string",
                    "SourceColumn": "string",
                    "TargetMultiMeasureAttributeName": "string"
                }
            ],
            "TargetMultiMeasureName": "string"
        },
        "TableName": "string",
        "TimeColumn": "string"
    }
  }
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ClientToken

를 사용하면 ClientToken 가 CreateScheduledQuery idempotent로 호출합니다. 즉, 동일한 요청을 반복적으로 수행하면 동일한 결과가 생성됩니다. 동일한 CreateScheduledQuery 요청을 여러 번 수행하면 단일 요청을 수행하는 것과 동일한 효과가 있습니다.

- 이 없이 CreateScheduledQuery 호출되면 `ClientToken`쿼리가 사용자를 대신하여 `ClientToken`를 SDK 생성합니다.
- 8시간 후 동일한 `ClientToken`이 있는 요청은 새 요청으로 처리됩니다.

유형: 문자열

길이 제한: 최소 길이는 32입니다. 최대 길이 128.

필수 여부: 아니요

ErrorReportConfiguration

오류 보고를 위한 구성입니다. 쿼리 결과를 작성할 때 문제가 발생하면 오류 보고서가 생성됩니다.

유형: ErrorReportConfiguration 객체

필수 여부: 예

KmsKeyId

예약된 쿼리 리소스인 유휴를 암호화하는 데 사용되는 Amazon KMS 키입니다. Amazon KMS 키를 지정하지 않으면 예약된 쿼리 리소스가 Timestream 소유 Amazon KMS 키로 암호화됩니다. KMS 키를 지정하려면 키 ID, 키 ARN, 별칭 이름 또는 별칭 을 사용합니다ARN. 별칭 이름을 사용할 때 이름 앞에 alias/를 붙입니다.

를 암호화 유형`SSE_KMS`으로 ErrorReportConfiguration 사용하는 경우 저장 중 오류 보고서를 암호화하는 데도 동일한 KmsKeyId 가 사용됩니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

Name

예약된 쿼리의 이름입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

Pattern: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

필수 여부: 예

NotificationConfiguration

예약된 쿼리에 대한 알림 구성입니다. 쿼리 실행이 완료되거나 상태가 업데이트되거나 삭제할 때 Timestream에서 알림을 보냅니다.

유형: NotificationConfiguration 객체

필수 여부: 예

QueryString

실행할 쿼리 문자열입니다. 쿼리 문자열 @ 문자와 식별자로 파라미터 이름을 지정할 수 있습니다. 명명된 파라미터 `@scheduled_runtime`은 예약되어 있으며 쿼리 실행이 예약된 시간을 가져오기 위해 쿼리에서 사용할 수 있습니다.

ScheduleConfiguration 파라미터에 따라 계산된 타임스탬프는 각 쿼리 실행에 대한 `@scheduled_runtime` 파라미터 값이 됩니다. 2021-12-01 00:00:00에 실행되는 예약된 쿼리의 인스턴스를 예로 들 수 있습니다. 이 인스턴스에 대해 쿼리를 호출할 때 `@scheduled_runtime` 파라미터는 타임스탬프 2021-12-01 00:00:00으로 초기화됩니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 262144자입니다.

필수 여부: 예

ScheduleConfiguration

쿼리의 일정 구성입니다.

유형: ScheduleConfiguration 객체

필수 여부: 예

ScheduledQueryExecutionRoleArn

예약된 쿼리ARN를 실행할 때 Timestream이 수임할 IAM 역할의 입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

Tags

예약된 쿼리에 레이블을 지정할 키-값 페어의 목록입니다.

유형: Tag객체 어레이

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 여부: 아니요

TargetConfiguration

쿼리 결과를 작성하는 데 사용되는 구성입니다.

유형: TargetConfiguration 객체

필수 항목 여부: 아니요

응답 구문

```
{
   "Arn": "string"
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Arn

ARN 생성된 예약 쿼리에 대해 .

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

ConflictException

취소된 쿼리에 대한 결과를 폴링할 수 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ServiceQuotaExceededException

서비스 할당량을 초과했습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface

- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DeleteScheduledQuery
서비스: Amazon Timestream Query

지정된 예약 쿼리를 삭제합니다. 이는 되돌릴 수 없는 작업입니다.

구문 요청

```
{
   "ScheduledQueryArn": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ScheduledQueryArn

예약된 쿼리의 ARN입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

Response Elements

작업이 성공하면 서비스는 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

요청한 리소스를 찾을 수 없습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DescribeAccountSettings
서비스: Amazon Timestream Query

쿼리 요금 모델과 서비스가 쿼리 워크로드에 사용할 수 있는 구성된 최대값TCUs을 포함하는 계정 설정을 설명합니다.

워크로드에 사용되는 컴퓨팅 단위의 기간 동안에만 요금이 부과됩니다.

Response Syntax

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

MaxQueryTCU

서비스가 언제든지 쿼리를 처리하는 데 사용할 Timestream 컴퓨팅 단위(TCUs)의 최대 수입니다.

유형: 정수

QueryPricingModel

계정의 쿼리에 대한 요금 모델입니다.

타입: 문자열

유효 값: `BYTES_SCANNED | COMPUTE_UNITS`

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DescribeEndpoints
서비스: Amazon Timestream Query

DescribeEndpoints 는 Timestream API 호출에 사용할 수 있는 엔드포인트 목록을 반환합니다. API 이는 쓰기와 쿼리를 통해 사용할 수 있습니다.

TimestreamSDKs은 서비스 엔드포인트의 관리 및 매핑을 포함하여 서비스의 아키텍처와 투명하게 작동하도록 설계되었으므로 가 API 아닌 한 이를 사용하지 않는 것이 좋습니다.

- VPC Timestream에서 엔드포인트(AWS PrivateLink)를 사용하고 있습니다.
- 애플리케이션에서 아직 SDK 지원되지 않는 프로그래밍 언어를 사용합니다.
- 클라이언트 측 구현을 더 잘 제어해야 합니다.

를 사용하고 구현하는 방법과 시기에 대한 자세한 내용은 엔드포인트 검색 패턴 섹션을 DescribeEndpoints참조하세요. https://docs.aws.amazon.com/timestream/latest/developerguide/Using.API.html#Using-API.endpoint-discovery

Response Syntax

```
{
   "Endpoints": [
      {
         "Address": "string",
         "CachePeriodInMinutes": number
      }
   ]
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Endpoints

DescribeEndpoints 요청이 이루어지면 Endpoints 객체가 반환됩니다.

타입: Endpoint 객체 배열

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

DescribeScheduledQuery
서비스: Amazon Timestream Query

예약된 쿼리에 대한 자세한 정보를 제공합니다.

구문 요청

```
{
   "ScheduledQueryArn": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ScheduledQueryArn

예약된 쿼리의 ARN입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 항목 여부: 예

응답 구문

```
{
   "ScheduledQuery": {
      "Arn": "string",
      "CreationTime": number,
      "ErrorReportConfiguration": {
         "S3Configuration": {
            "BucketName": "string",
            "EncryptionOption": "string",
            "ObjectKeyPrefix": "string"
         }
      },
      "KmsKeyId": "string",
      "LastRunSummary": {
         "ErrorReportLocation": {
```

```
            "S3ReportLocation": {
               "BucketName": "string",
               "ObjectKey": "string"
            }
         },
         "ExecutionStats": {
            "BytesMetered": number,
            "CumulativeBytesScanned": number,
            "DataWrites": number,
            "ExecutionTimeInMillis": number,
            "QueryResultRows": number,
            "RecordsIngested": number
         },
         "FailureReason": "string",
         "InvocationTime": number,
         "QueryInsightsResponse": {
            "OutputBytes": number,
            "OutputRows": number,
            "QuerySpatialCoverage": {
               "Max": {
                  "PartitionKey": [ "string" ],
                  "TableArn": "string",
                  "Value": number
               }
            },
            "QueryTableCount": number,
            "QueryTemporalRange": {
               "Max": {
                  "TableArn": "string",
                  "Value": number
               }
            }
         },
         "RunStatus": "string",
         "TriggerTime": number
      },
      "Name": "string",
      "NextInvocationTime": number,
      "NotificationConfiguration": {
         "SnsConfiguration": {
            "TopicArn": "string"
         }
      },
      "PreviousInvocationTime": number,
```

```
      "QueryString": "string",
      "RecentlyFailedRuns": [
         {
            "ErrorReportLocation": {
               "S3ReportLocation": {
                  "BucketName": "string",
                  "ObjectKey": "string"
               }
            },
            "ExecutionStats": {
               "BytesMetered": number,
               "CumulativeBytesScanned": number,
               "DataWrites": number,
               "ExecutionTimeInMillis": number,
               "QueryResultRows": number,
               "RecordsIngested": number
            },
            "FailureReason": "string",
            "InvocationTime": number,
            "QueryInsightsResponse": {
               "OutputBytes": number,
               "OutputRows": number,
               "QuerySpatialCoverage": {
                  "Max": {
                     "PartitionKey": [ "string" ],
                     "TableArn": "string",
                     "Value": number
                  }
               },
               "QueryTableCount": number,
               "QueryTemporalRange": {
                  "Max": {
                     "TableArn": "string",
                     "Value": number
                  }
               }
            },
            "RunStatus": "string",
            "TriggerTime": number
         }
      ],
      "ScheduleConfiguration": {
         "ScheduleExpression": "string"
      },
```

```
      "ScheduledQueryExecutionRoleArn": "string",
      "State": "string",
      "TargetConfiguration": {
         "TimestreamConfiguration": {
            "DatabaseName": "string",
            "DimensionMappings": [
               {
                  "DimensionValueType": "string",
                  "Name": "string"
               }
            ],
            "MeasureNameColumn": "string",
            "MixedMeasureMappings": [
               {
                  "MeasureName": "string",
                  "MeasureValueType": "string",
                  "MultiMeasureAttributeMappings": [
                     {
                        "MeasureValueType": "string",
                        "SourceColumn": "string",
                        "TargetMultiMeasureAttributeName": "string"
                     }
                  ],
                  "SourceColumn": "string",
                  "TargetMeasureName": "string"
               }
            ],
            "MultiMeasureMappings": {
               "MultiMeasureAttributeMappings": [
                  {
                     "MeasureValueType": "string",
                     "SourceColumn": "string",
                     "TargetMultiMeasureAttributeName": "string"
                  }
               ],
               "TargetMultiMeasureName": "string"
            },
            "TableName": "string",
            "TimeColumn": "string"
         }
      }
   }
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

ScheduledQuery

예약된 쿼리입니다.

유형: ScheduledQueryDescription 객체

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

요청한 리소스를 찾을 수 없습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

ExecuteScheduledQuery
서비스: Amazon Timestream Query

이를 사용하여 예약된 쿼리를 수동으로 API 실행할 수 있습니다.

를 활성화하면 Amazon SNS 알림의 일부로 실행한 쿼리와 관련된 인사이트와 지표QueryInsightsAPI도 반환됩니다. 는 쿼리의 성능 조정을 QueryInsights 지원합니다. 에 대한 자세한 내용은 Amazon Timestream 에서 쿼리 인사이트를 사용하여 쿼리 최적화를 QueryInsights참조하세요.

구문 요청

```
{
   "ClientToken": "string",
   "InvocationTime": number,
   "QueryInsights": {
      "Mode": "string"
   },
   "ScheduledQueryArn": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ClientToken

사용하지 않음

유형: 문자열

길이 제한: 최소 길이는 32입니다. 최대 길이 128.

필수 여부: 아니요

InvocationTime

의 타임스탬프입니다UTC. 쿼리는 이 타임스탬프에서 호출된 것처럼 실행됩니다.

유형: 타임스탬프

필수 여부: 예

QueryInsights

활성화에 대한 설정을 캡슐화합니다QueryInsights.

활성화하면 실행한 쿼리에 대한 Amazon SNS 알림의 일부로 인사이트와 지표가 QueryInsights 반환됩니다. QueryInsights 를 사용하여 쿼리 성능 및 비용을 조정할 수 있습니다.

유형: ScheduledQueryInsights 객체

필수 여부: 아니요

ScheduledQueryArn

ARN 예약된 쿼리의 .

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

Response Elements

작업이 성공하면 서비스가 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

요청한 리소스를 찾을 수 없습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

예시

ENABLED_WITH_RATE_CONTROL 모드에 대한 예약된 쿼리 알림 메시지

다음 예제에서는 QueryInsights 파라미터 ENABLED_WITH_RATE_CONTROL 모드에 대해 성공적으로 예약된 쿼리 알림 메시지를 보여줍니다.

```
"SuccessNotificationMessage": {
    "type": "MANUAL_TRIGGER_SUCCESS",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-49c6ed55-
c2e7-4cc2-9956-4a0ecea13420-80e05b035236a4c3",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1723710546,
        "triggerTimeMillis": 1723710547490,
        "runStatus": "MANUAL_TRIGGER_SUCCESS",
        "executionStats": {
            "executionTimeInMillis": 17343,
            "dataWrites": 1024,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 600,
            "recordsIngested": 1,
            "queryResultRows": 1
        },
        "queryInsightsResponse": {
            "querySpatialCoverage": {
                "max": {
```

```
                    "value": 1.0,
                    "tableArn": "arn:aws:timestream:<Region>:<Account>:database/BaseDb/
table/BaseTable",
                    "partitionKey": [
                        "measure_name"
                    ]
                }
            },
            "queryTemporalRange": {
                "max": {
                    "value": 2399999999999,
                    "tableArn": "arn:aws:timestream:<Region>:<Account>:database/BaseDb/
table/BaseTable"
                }
            },
            "queryTableCount": 1,
            "outputRows": 1,
            "outputBytes": 59
        }
    }
}
```

DISABLED 모드에 대해 예약된 쿼리 알림 메시지

다음 예제에서는 QueryInsights 파라미터 DISABLED 모드에 대한 성공적인 예약 쿼리 알림 메시지를 보여줍니다.

```
"SuccessNotificationMessage": {
    "type": "MANUAL_TRIGGER_SUCCESS",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
fa109d9e-6528-4a0d-ac40-482fa05e657f-140faaeecdc5b2a7",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1723711401,
        "triggerTimeMillis": 1723711402144,
        "runStatus": "MANUAL_TRIGGER_SUCCESS",
        "executionStats": {
            "executionTimeInMillis": 17992,
            "dataWrites": 1024,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 600,
            "recordsIngested": 1,
            "queryResultRows": 1
        }
```

```
    }
}
```

ENABLED_WITH_RATE_CONTROL 모드에 대한 실패 알림 메시지

다음 예제에서는 QueryInsights 파라미터 ENABLED_WITH_RATE_CONTROL 모드에 대한 실패한 예약 쿼리 알림 메시지를 보여줍니다.

```
"FailureNotificationMessage": {
    "type": "MANUAL_TRIGGER_FAILURE",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
b261670d-790c-4116-9db5-0798071b18b1-b7e27a1d79be226d",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1727915513,
        "triggerTimeMillis": 1727915513894,
        "runStatus": "MANUAL_TRIGGER_FAILURE",
        "executionStats": {
            "executionTimeInMillis": 10777,
            "dataWrites": 0,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 0,
            "recordsIngested": 0,
            "queryResultRows": 4
        },
        "errorReportLocation": {
            "s3ReportLocation": {
                "bucketName": "my-amzn-s3-demo-bucket",
                "objectKey": "4my-organization-f7a3c5d065a1a95e/1727915513/
MANUAL/1727915513894/5e14b3df-b147-49f4-9331-784f749b68ae"
            }
        },
        "failureReason": "Schedule encountered some errors and is incomplete. Please
 take a look at error report for further details"
    }
}
```

DISABLED 모드에 대한 실패 알림 메시지

다음 예제에서는 QueryInsights 파라미터 DISABLED 모드에 대한 실패한 예약 쿼리 알림 메시지를 보여줍니다.

```
"FailureNotificationMessage": {
```

```
    "type": "MANUAL_TRIGGER_FAILURE",
    "arn": "arn:aws:timestream:<Region>:<Account>:scheduled-query/sq-test-
b261670d-790c-4116-9db5-0798071b18b1-b7e27a1d79be226d",
    "scheduledQueryRunSummary": {
        "invocationEpochSecond": 1727915194,
        "triggerTimeMillis": 1727915195119,
        "runStatus": "MANUAL_TRIGGER_FAILURE",
        "executionStats": {
            "executionTimeInMillis": 10777,
            "dataWrites": 0,
            "bytesMetered": 0,
            "cumulativeBytesScanned": 0,
            "recordsIngested": 0,
            "queryResultRows": 4
        },
        "errorReportLocation": {
            "s3ReportLocation": {
                "bucketName": "my-amzn-s3-demo-bucket",
                "objectKey": "4my-organization-b7e27a1d79be226d/1727915194/
MANUAL/1727915195119/08dea9f5-9a0a-4e63-a5f7-ded23247bb98"
            }
        },
        "failureReason": "Schedule encountered some errors and is incomplete. Please
 take a look at error report for further details"
    }
}
```

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

ListScheduledQueries
서비스: Amazon Timestream Query

호출자의 Amazon 계정 및 리전에서 예약된 모든 쿼리 목록을 가져옵니다.
`ListScheduledQueries`는 최종적으로 일관됩니다.

구문 요청

```
{
   "MaxResults": number,
   "NextToken": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

MaxResults

출력에서 반환할 최대 항목 수입니다. 사용 가능한 총 항목 수가 지정된 값보다 많은 경우 출력에 `NextToken` 가 제공됩니다. 페이지 매김을 재개하려면 `NextToken` 값을 에 대한 후속 호출에 대한 인수로 제공합니다`ListScheduledQueriesRequest`.

타입: 정수

유효한 범위: 최소값은 1입니다. 최대값은 1000입니다.

필수 여부: 아니요

NextToken

페이지 매김을 재개하기 위한 페이지 매김 토큰입니다.

유형: 문자열

필수사항: 아니요

응답 구문

```
{
   "NextToken": "string",
```

```
   "ScheduledQueries": [
      {
         "Arn": "string",
         "CreationTime": number,
         "ErrorReportConfiguration": {
            "S3Configuration": {
               "BucketName": "string",
               "EncryptionOption": "string",
               "ObjectKeyPrefix": "string"
            }
         },
         "LastRunStatus": "string",
         "Name": "string",
         "NextInvocationTime": number,
         "PreviousInvocationTime": number,
         "State": "string",
         "TargetDestination": {
            "TimestreamDestination": {
               "DatabaseName": "string",
               "TableName": "string"
            }
         }
      }
   ]
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

NextToken

페이지 매김을 시작할 위치를 지정하기 위한 토큰입니다. 이는 이전에 잘린 응답 NextToken 의 입니다.

유형: 문자열

ScheduledQueries

예약된 쿼리 목록입니다.

타입: ScheduledQuery 객체 배열

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용

- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

ListTagsForResource
서비스: Amazon Timestream Query

Timestream 쿼리 리소스의 모든 태그를 나열합니다.

구문 요청

```
{
   "MaxResults": number,
   "NextToken": "string",
   "ResourceARN": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

MaxResults

반환할 최대 태그 수입니다.

타입: 정수

유효 범위: 최소값 1. 최대값은 200입니다.

필수 여부: 아니요

NextToken

페이지 매김을 재개하기 위한 페이지 매김 토큰입니다.

유형: 문자열

필수 항목 여부: 아니요

ResourceARN

나열할 태그가 있는 Timestream 리소스입니다. 이 값은 Amazon 리소스 이름()입니다ARN.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 항목 여부: 예

응답 구문

```
{
   "NextToken": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

NextToken

에 대한 후속 호출과 함께 페이지 매김을 재개하는 페이지 매김 토큰입니다ListTagsForResourceResponse.

유형: 문자열

Tags

현재 Timestream 리소스와 연결된 태그입니다.

유형: Tag객체 어레이

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

요청한 리소스를 찾을 수 없습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

PrepareQuery
서비스: Amazon Timestream Query

나중에 실행할 수 있도록 Timestream에 저장할 파라미터가 있는 쿼리를 제출할 수 있는 동기식 작업입니다. Timestream은 로 `ValidateOnly` 설정된 이 작업만 사용할 수 있습니다`true`.

구문 요청

```
{
   "QueryString": "string",
   "ValidateOnly": boolean
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

QueryString

준비된 문으로 사용할 Timestream 쿼리 문자열입니다. 쿼리 문자열 @ 문자와 식별자로 파라미터 이름을 지정할 수 있습니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 262144자입니다.

필수 여부: 예

ValidateOnly

이 값을 로 설정하면 `true`Timestream은 쿼리 문자열이 유효한 Timestream 쿼리인지만 검증하고 나중에 사용할 수 있도록 준비된 쿼리를 저장하지 않습니다.

타입: 부울

필수 항목 여부: 아니요

응답 구문

```
{
   "Columns": [
      {
```

```
         "Aliased": boolean,
         "DatabaseName": "string",
         "Name": "string",
         "TableName": "string",
         "Type": {
            "ArrayColumnInfo": {
               "Name": "string",
               "Type": "Type"
            },
            "RowColumnInfo": [
               {
                  "Name": "string",
                  "Type": "Type"
               }
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": {
               "Name": "string",
               "Type": "Type"
            }
         }
      }
   ],
   "Parameters": [
      {
         "Name": "string",
         "Type": {
            "ArrayColumnInfo": {
               "Name": "string",
               "Type": "Type"
            },
            "RowColumnInfo": [
               {
                  "Name": "string",
                  "Type": "Type"
               }
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": {
               "Name": "string",
               "Type": "Type"
            }
         }
      }
```

```
    ],
    "QueryString": "string"
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

Columns

제출된 쿼리 문자열의 SELECT 절 열 목록입니다.

유형: SelectColumn객체 어레이

Parameters

제출된 쿼리 문자열에 사용되는 파라미터 목록입니다.

유형: ParameterMapping객체 어레이

QueryString

준비하려는 쿼리 문자열입니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 262144자입니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

Query
서비스: Amazon Timestream Query

Query 는 Amazon Timestream 데이터에 대해 쿼리를 실행할 수 있는 동기식 작업입니다.

를 활성화하면 실행한 쿼리와 관련된 인사이트와 지표QueryInsightsAPI도 반환됩니다. 는 쿼리의 성능 조정을 QueryInsights 지원합니다. 에 대한 자세한 내용은 Amazon Timestream 에서 쿼리 인사이트를 사용하여 쿼리 최적화를 QueryInsights참조하세요.

> **Note**
>
> QueryInsights 활성화된 상태로 수행할 수 있는 최대 Query API 요청 수는 초당 쿼리 1개() 입니다QPS. 이 쿼리 속도를 초과하면 제한이 발생할 수 있습니다.

Query 는 60초 후에 시간 초과됩니다. 에서 기본 제한 시간을 업데이트SDK하여 60초의 제한 시간을 지원해야 합니다. 자세한 내용은 코드 샘플을 참조하세요.

다음과 같은 경우 쿼리 요청이 실패합니다.

- 5분 idempotency 기간을 벗어나 동일한 클라이언트 토큰으로 Query 요청을 제출하는 경우.
- 동일한 클라이언트 토큰으로 Query 요청을 제출하지만 다른 파라미터를 변경하는 경우 5분 idempotency 기간 내에.
- 행의 크기(쿼리 메타데이터 포함)가 1MB를 초과하는 경우 다음 오류 메시지와 함께 쿼리가 실패합니다.

  ```
  Query aborted as max page response size has been exceeded by the output result row
  ```
- 쿼리 이니시에이터와 결과 리더의 IAM 보안 주체가 같지 않고/않거나 쿼리 이니시에이터와 결과 리더의 쿼리 요청에 동일한 쿼리 문자열이 없는 경우 쿼리가 실패하고 Invalid pagination token 오류가 발생합니다.

구문 요청

```
{
   "ClientToken": "string",
   "MaxRows": number,
   "NextToken": "string",
   "QueryInsights": {
```

```
        "Mode": "string"
    },
    "QueryString": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ClientToken

Query 요청 시 지정된 최대 64ASCII자의 대/소문자를 구분하는 고유한 문자열입니다. 를 제공ClientToken하면 Query idempotent 에 대한 호출이 이루어집니다. 즉, 동일한 쿼리를 반복적으로 실행하면 동일한 결과가 생성됩니다. 즉, 동일한 Query 요청을 여러 번 수행하면 단일 요청을 수행하는 것과 동일한 효과가 있습니다. 쿼리ClientToken에서 를 사용할 때는 다음 사항에 유의하세요.

- 쿼리API가 없이 인스턴스화되면 ClientToken쿼리는 ClientToken 사용자를 대신하여 를 SDK 생성합니다.
- Query 호출에 만 포함되지ClientToken만 가 포함되지 않은 경우 NextToken의 호출Query은 새 쿼리 실행으로 간주됩니다.
- 호출에 가 포함된 경우 NextToken해당 특정 호출은 쿼리 에 대한 이전 호출의 후속 호출로 간주 API되고 결과 세트가 반환됩니다.
- 4시간이 지나면 동일한 요청이 새 요청으로 ClientToken 처리됩니다.

유형: 문자열

길이 제한: 최소 길이는 32입니다. 최대 길이 128.

필수 여부: 아니요

MaxRows

Query 출력에서 반환할 총 행 수입니다. Query MaxRows 값이 지정된 의 초기 실행은 두 가지 경우에 쿼리의 결과 세트를 반환합니다.

- 결과의 크기는 보다 작습니다1MB.
- 결과 세트의 행 수가 값보다 작습니다maxRows.

그렇지 않으면 의 초기 호출은 Query만 반환하며NextToken, 이후 호출에서 결과 세트를 가져오는 데 사용할 수 있습니다. 페이지 매김을 재개하려면 다음 명령에 NextToken 값을 입력합니다.

행 크기가 큰 경우(예: 행에 열이 많음) Timestream은 응답 크기가 1MB 제한을 초과하지 않도록 더 적은 수의 행을 반환할 수 있습니다. MaxRows 이 제공되지 않으면 Timestream은 1MB 제한을 충족하는 데 필요한 행 수를 전송합니다.

타입: 정수

유효한 범위: 최소값은 1입니다. 최대값은 1000입니다.

필수 여부: 아니요

NextToken

결과 세트를 반환하는 데 사용되는 페이지 매김 토큰입니다. Query API 를 사용하여 를 호출하면 NextToken해당 특정 호출은 에 대한 이전 호출의 후속 호출로 간주Query되고 결과 세트가 반환됩니다. 그러나 Query 호출에 만 포함된 경우 ClientToken의 호출Query은 새 쿼리 실행으로 간주됩니다.

쿼리 NextToken 에서 를 사용할 때 다음 사항에 유의하세요.

- 페이지 매김 토큰은 최대 5회의 Query 호출 또는 최대 1시간의 기간 중 먼저 도래하는 시점에 사용할 수 있습니다.
- 동일한 를 사용하면 동일한 레코드 세트가 반환NextToken됩니다. 결과 세트를 계속 페이지 매김하려면 최신 를 사용해야 합니다nextToken.
- Query 호출이 두 개의 NextToken 값인 TokenA 및 를 반환한다고 가정해 보겠습니다TokenB. 후속 Query 호출에서 TokenB 를 사용하는 경우 TokenA는 무효화되며 재사용할 수 없습니다.
- 페이지 매김이 시작된 후 쿼리에서 이전 결과 세트를 요청하려면 쿼리를 다시 호출해야 합니다 API.
- 가 null 반환NextToken될 때까지 최신 를 사용하여 페이지 매김해야 하며, 반환 시 새 를 사용해야 NextToken 합니다.
- 쿼리 이니시에이터와 결과 리더의 IAM 보안 주체가 같지 않고/않거나 쿼리 이니시에이터와 결과 리더의 쿼리 요청에 동일한 쿼리 문자열이 없는 경우 쿼리가 실패하고 Invalid pagination token 오류가 발생합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

QueryInsights

활성화에 대한 설정을 캡슐화합니다QueryInsights.

를 활성화하면 실행한 쿼리에 대한 쿼리 결과 외에도 인사이트와 지표가 QueryInsights 반환됩니다. QueryInsights 를 사용하여 쿼리 성능을 조정할 수 있습니다.

유형: QueryInsights 객체

필수 여부: 아니요

QueryString

Timestream에서 실행할 쿼리입니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 262144자입니다.

필수 항목 여부: 예

응답 구문

```
{
   "ColumnInfo": [
      {
         "Name": "string",
         "Type": {
            "ArrayColumnInfo": "ColumnInfo",
            "RowColumnInfo": [
               "ColumnInfo"
            ],
            "ScalarType": "string",
            "TimeSeriesMeasureValueColumnInfo": "ColumnInfo"
         }
      }
   ],
   "NextToken": "string",
   "QueryId": "string",
   "QueryInsightsResponse": {
      "OutputBytes": number,
      "OutputRows": number,
      "QuerySpatialCoverage": {
         "Max": {
            "PartitionKey": [ "string" ],
            "TableArn": "string",
            "Value": number
         }
```

```
      },
      "QueryTableCount": number,
      "QueryTemporalRange": {
         "Max": {
            "TableArn": "string",
            "Value": number
         }
      },
      "UnloadPartitionCount": number,
      "UnloadWrittenBytes": number,
      "UnloadWrittenRows": number
   },
   "QueryStatus": {
      "CumulativeBytesMetered": number,
      "CumulativeBytesScanned": number,
      "ProgressPercentage": number
   },
   "Rows": [
      {
         "Data": [
            {
               "ArrayValue": [
                  "Datum"
               ],
               "NullValue": boolean,
               "RowValue": "Row",
               "ScalarValue": "string",
               "TimeSeriesValue": [
                  {
                     "Time": "string",
                     "Value": "Datum"
                  }
               ]
            }
         ]
      }
   ]
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

ColumnInfo

반환된 결과 세트의 열 데이터 유형입니다.

유형: ColumnInfo객체 어레이

NextToken

Query 호출 시 다시 사용하여 다음 결과 세트를 가져올 수 있는 페이지 매김 토큰입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

QueryId

지정된 쿼리의 고유 ID입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

패턴: [a-zA-Z0-9]+

QueryInsightsResponse

실행한 쿼리와 관련된 인사이트 및 지표가 QueryInsights 포함된 캡슐화입니다.

유형: QueryInsightsResponse 객체

QueryStatus

진행 상황 및 스캔된 바이트를 포함하여 쿼리 상태에 대한 정보입니다.

유형: QueryStatus 객체

Rows

쿼리에서 반환된 결과 세트 행입니다.

타입: Row 객체 배열

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

ConflictException

취소된 쿼리에 대한 결과를 폴링할 수 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

QueryExecutionException

Timestream이 쿼리를 성공적으로 실행할 수 없습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface

- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

TagResource
서비스: Amazon Timestream Query

태그 세트를 Timestream 리소스와 연결합니다. 그런 다음 이러한 사용자 정의 태그를 활성화하여 비용 할당 추적을 위해 Billing and Cost Management 콘솔에 표시되도록 할 수 있습니다.

구문 요청

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "string",
         "Value": "string"
      }
   ]
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ResourceARN

태그를 추가해야 하는 Timestream 리소스를 식별합니다. 이 값은 Amazon 리소스 이름()입니다 ARN.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

Tags

Timestream 리소스에 할당할 태그입니다.

유형: Tag객체 어레이

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

필수 여부: 예

Response Elements

작업이 성공하면 서비스는 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

요청한 리소스를 찾을 수 없습니다.

HTTP 상태 코드: 400

ServiceQuotaExceededException

서비스 할당량을 초과했습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용

- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

UntagResource
서비스: Amazon Timestream Query

Timestream 쿼리 리소스에서 태그 연결을 제거합니다.

구문 요청

```
{
   "ResourceARN": "string",
   "TagKeys": [ "string" ]
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ResourceARN

태그를 제거할 Timestream 리소스입니다. 이 값은 Amazon 리소스 이름()입니다ARN.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

TagKeys

태그 키 목록입니다. 키가 이 목록의 멤버인 리소스의 기존 태그는 Timestream 리소스에서 제거됩니다.

타입: 문자열 배열

어레이 멤버: 최소 항목 수 0개. 최대 항목 수 200개.

길이 제약: 최소 길이 1. 최대 길이는 128.

필수 여부: 예

Response Elements

작업이 성공하면 서비스가 빈 HTTP 본문과 함께 HTTP 200 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

요청한 리소스를 찾을 수 없습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

UpdateAccountSettings
서비스: Amazon Timestream Query

쿼리 요금TCUs에 사용할 계정을 전환하고 구성한 최대 쿼리 컴퓨팅 단위를 수정합니다. 값을 원하는 구성MaxQueryTCU으로 줄이면 새 값이 적용되려면 최대 24시간이 걸릴 수 있습니다.

> **Note**
>
> 쿼리 요금TCUs에 사용할 계정을 전환한 후에는 쿼리 요금에 스캔된 바이트 사용으로 전환할 수 없습니다.

구문 요청

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

MaxQueryTCU

서비스가 쿼리를 처리하는 데 언제든지 사용할 최대 컴퓨팅 단위 수입니다. 쿼리를 실행하려면 최소 용량을 4로 설정해야 합니다TCU. 의 최대 개수를 4, 예를 들어 4, 8, 16, 32 등의 TCU 배수로 설정할 수 있습니다.

에 지원되는 최대값은 1000MaxQueryTCU입니다. 이 소프트 한도에 대한 증가를 요청하려면 AWS 지원 부서에 문의하세요. 의 기본 할당량에 대한 자세한 내용은 기본 할당량 을 maxQueryTCU참조하세요. https://docs.aws.amazon.com/timestream/latest/developerguide/ts-limits.html#limits.default

유형: 정수

필수 항목 여부: 아니요

QueryPricingModel

계정의 쿼리에 대한 요금 모델입니다.

> **Note**
>
> QueryPricingModel 파라미터는 여러 Timestream 작업에서 사용되지만 UpdateAccountSettings API 이 작업은 이외의 값을 인식하지 못합니다.COMPUTE_UNITS.

타입: 문자열

유효 값: BYTES_SCANNED | COMPUTE_UNITS

필수 항목 여부: 아니요

응답 구문

```
{
   "MaxQueryTCU": number,
   "QueryPricingModel": "string"
}
```

Response Elements

작업이 성공하면 서비스가 HTTP 200개의 응답을 다시 보냅니다.

다음 데이터는 서비스에서 JSON 형식으로 반환됩니다.

MaxQueryTCU

서비스가 쿼리를 처리하는 데 언제든지 사용할 구성된 최대 컴퓨팅 단위 수입니다.

유형: 정수

QueryPricingModel

계정의 요금 모델입니다.

타입: 문자열

유효 값: BYTES_SCANNED | COMPUTE_UNITS

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용
- AWS SDK Java V2용

- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

UpdateScheduledQuery
서비스: Amazon Timestream Query

예약된 쿼리를 업데이트합니다.

구문 요청

```
{
   "ScheduledQueryArn": "string",
   "State": "string"
}
```

요청 파라미터

모든 작업에 공통되는 파라미터에 대한 내용은 공통 파라미터를 참조하십시오.

요청은 JSON 형식의 다음 데이터를 수락합니다.

ScheduledQueryArn

ARN 예약 쿼리의 입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

State

예약된 쿼리의 상태입니다.

타입: 문자열

유효 값: `ENABLED | DISABLED`

필수 여부: 예

Response Elements

작업이 성공하면 서비스가 빈 HTTP 본문과 함께 HTTP 200개의 응답을 다시 보냅니다.

Errors

모든 작업에 공통되는 오류에 대한 내용은 일반적인 오류 섹션을 참조하십시오.

AccessDeniedException

이 작업을 수행할 권한이 없습니다.

HTTP 상태 코드: 400

InternalServerException

내부 서버 오류로 인해 서비스가 이 요청을 완전히 처리할 수 없습니다.

HTTP 상태 코드: 400

InvalidEndpointException

요청된 엔드포인트가 유효하지 않습니다.

HTTP 상태 코드: 400

ResourceNotFoundException

요청한 리소스를 찾을 수 없습니다.

HTTP 상태 코드: 400

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationException

유효하지 않거나 잘못된 형식의 요청입니다.

HTTP 상태 코드: 400

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS Command Line Interface
- AWS SDK 용.NET
- AWS SDK C++용
- AWS SDK Go v2용

- AWS SDK Java V2용
- AWS SDK for JavaScript V3
- AWS SDK PHP V3용
- AWS SDK Python용
- AWS SDK Ruby V3용

데이터 타입

Amazon Timestream Write에서 지원하는 데이터 유형은 다음과 같습니다.

- BatchLoadProgressReport
- BatchLoadTask
- BatchLoadTaskDescription
- CsvConfiguration
- Database
- DataModel
- DataModelConfiguration
- DataModelS3Configuration
- DataSourceConfiguration
- DataSourceS3Configuration
- Dimension
- DimensionMapping
- Endpoint
- MagneticStoreRejectedDataLocation
- MagneticStoreWriteProperties
- MeasureValue
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- PartitionKey
- Record
- RecordsIngested

- RejectedRecord
- ReportConfiguration
- ReportS3Configuration
- RetentionProperties
- S3Configuration
- Schema
- Table
- Tag

Amazon Timestream Query에서 지원하는 데이터 유형은 다음과 같습니다.

- ColumnInfo
- Datum
- DimensionMapping
- Endpoint
- ErrorReportConfiguration
- ErrorReportLocation
- ExecutionStats
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- NotificationConfiguration
- ParameterMapping
- QueryInsights
- QueryInsightsResponse
- QuerySpatialCoverage
- QuerySpatialCoverageMax
- QueryStatus
- QueryTemporalRange
- QueryTemporalRangeMax
- Row

- S3Configuration
- S3ReportLocation
- ScheduleConfiguration
- ScheduledQuery
- ScheduledQueryDescription
- ScheduledQueryInsights
- ScheduledQueryInsightsResponse
- ScheduledQueryRunSummary
- SelectColumn
- SnsConfiguration
- Tag
- TargetConfiguration
- TargetDestination
- TimeSeriesDataPoint
- TimestreamConfiguration
- TimestreamDestination
- Type

Amazon Timestream 쓰기

Amazon Timestream Write에서 지원하는 데이터 유형은 다음과 같습니다.

- BatchLoadProgressReport
- BatchLoadTask
- BatchLoadTaskDescription
- CsvConfiguration
- Database
- DataModel
- DataModelConfiguration
- DataModelS3Configuration
- DataSourceConfiguration
- DataSourceS3Configuration

- Dimension
- DimensionMapping
- Endpoint
- MagneticStoreRejectedDataLocation
- MagneticStoreWriteProperties
- MeasureValue
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- PartitionKey
- Record
- RecordsIngested
- RejectedRecord
- ReportConfiguration
- ReportS3Configuration
- RetentionProperties
- S3Configuration
- Schema
- Table
- Tag

BatchLoadProgressReport
서비스: Amazon Timestream Write

배치 로드 작업의 진행 상황에 대한 세부 정보입니다.

내용

BytesMetered

유형: Long

필수 여부: 아니요

FileFailures

유형: Long

필수 여부: 아니요

ParseFailures

유형: Long

필수 여부: 아니요

RecordIngestionFailures

유형: Long

필수 여부: 아니요

RecordsIngested

유형: Long

필수 여부: 아니요

RecordsProcessed

유형: Long

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

BatchLoadTask
서비스: Amazon Timestream Write

배치 로드 작업에 대한 세부 정보입니다.

내용

CreationTime

Timestream 배치 로드 작업이 생성된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

DatabaseName

배치 로드 태스크가 데이터를 로드하는 데이터베이스의 데이터베이스 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

LastUpdatedTime

Timestream 배치 로드 작업이 마지막으로 업데이트된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

ResumableUntil

유형: 타임스탬프

필수 여부: 아니요

TableName

배치 로드 태스크가 데이터를 로드하는 테이블의 테이블 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

TaskId

배치 로드 작업의 ID입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이 32.

패턴: `[A-Z0-9]+`

Required: No

TaskStatus

배치 로드 작업의 상태입니다.

타입: 문자열

유효 값: `CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME`

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

BatchLoadTaskDescription
서비스: Amazon Timestream Write

배치 로드 작업에 대한 세부 정보입니다.

내용

CreationTime

Timestream 배치 로드 작업이 생성된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

DataModelConfiguration

배치 로드 작업에 대한 데이터 모델 구성입니다. 여기에는 배치 로드 작업의 데이터 모델이 저장되는 위치에 대한 세부 정보가 포함되어 있습니다.

유형: DataModelConfiguration 객체

필수 여부: 아니요

DataSourceConfiguration

배치 로드 작업의 데이터 소스에 대한 구성 세부 정보입니다.

유형: DataSourceConfiguration 객체

필수 여부: 아니요

ErrorMessage

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

LastUpdatedTime

Timestream 배치 로드 작업이 마지막으로 업데이트된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

ProgressReport

유형: BatchLoadProgressReport 객체

필수 여부: 아니요

RecordVersion

유형: Long

필수 여부: 아니요

ReportConfiguration

배치 로드 태스크에 대한 구성 보고 여기에는 오류 보고서가 저장되는 위치에 대한 세부 정보가 포함되어 있습니다.

유형: ReportConfiguration 객체

필수 여부: 아니요

ResumableUntil

유형: 타임스탬프

필수 여부: 아니요

TargetDatabaseName

유형: 문자열

필수 항목 여부: 아니요

TargetTableName

유형: 문자열

필수 항목 여부: 아니요

TaskId

배치 로드 작업의 ID입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이 32.

패턴: `[A-Z0-9]+`

Required: No

TaskStatus

배치 로드 작업의 상태입니다.

타입: 문자열

유효 값: `CREATED | IN_PROGRESS | FAILED | SUCCEEDED | PROGRESS_STOPPED | PENDING_RESUME`

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

CsvConfiguration
서비스: Amazon Timestream Write

열 구분 기호가 쉼표일 수 있고 레코드 구분 기호가 새 줄 문자인 구분된 데이터 형식입니다.

내용

ColumnSeparator

열 구분 기호는 쉼표(','), 파이프('|), 세미콜론(';'), 탭('/t') 또는 공백(' ') 중 하나일 수 있습니다.

유형: 문자열

길이 제약 조건: 1의 길이가 수정되었습니다.

필수 여부: 아니요

EscapeChar

이스케이프 문자는 다음 중 하나일 수 있습니다.

유형: 문자열

길이 제약 조건: 1의 길이가 수정되었습니다.

필수 여부: 아니요

NullValue

공백(' ')일 수 있습니다.

유형: 문자열

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 256.

필수 여부: 아니요

QuoteChar

작은따옴표(') 또는 큰따옴표(')일 수 있습니다.

유형: 문자열

길이 제약 조건: 1의 길이가 수정되었습니다.

필수 여부: 아니요

TrimWhiteSpace

선행 및 후행 공백을 정리하도록 지정합니다.

타입: 부울

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Database
서비스: Amazon Timestream Write

테이블의 최상위 컨테이너입니다. 데이터베이스와 테이블은 Amazon Timestream의 기본 관리 개념입니다. 데이터베이스의 모든 테이블은 동일한 AWS KMS 키로 암호화됩니다.

내용

Arn

이 데이터베이스를 고유하게 식별하는 Amazon 리소스 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

CreationTime

데이터베이스가 생성된 시간으로, Unix 에포크 시간에서 계산됩니다.

유형: 타임스탬프

필수 여부: 아니요

DatabaseName

Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 아니요

KmsKeyId

데이터베이스에 저장된 데이터를 암호화하는 데 사용되는 AWS KMS 키의 식별자입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

LastUpdatedTime

이 데이터베이스가 마지막으로 업데이트된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

TableCount

Timestream 데이터베이스 내에서 발견된 총 테이블 수입니다.

유형: Long

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

DataModel
서비스: Amazon Timestream Write

배치 로드 작업의 데이터 모델입니다.

내용

DimensionMappings

차원에 대한 소스-대상 매핑.

유형: DimensionMapping객체 어레이

어레이 멤버: 최소 항목 수 1개.

필수 항목 여부: 예

MeasureNameColumn

유형: String

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 256.

필수 여부: 아니요

MixedMeasureMappings

측정값에 대한 소스-대상 매핑입니다.

유형: MixedMeasureMapping객체 어레이

배열 구성원: 최소수는 1개입니다.

필수 여부: 아니요

MultiMeasureMappings

다중 측정 레코드의 소스-대상 매핑.

유형: MultiMeasureMappings 객체

필수 여부: 아니요

TimeColumn

시간에 매핑할 소스 열입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 256.

필수 여부: 아니요

TimeUnit

타임스탬프 단위의 세분화입니다. 시간 값이 초, 밀리초, 나노초 또는 기타 지원되는 값인지 나타냅니다. 기본값은 `MILLISECONDS`입니다.

타입: 문자열

유효 값: `MILLISECONDS | SECONDS | MICROSECONDS | NANOSECONDS`

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

DataModelConfiguration
서비스: Amazon Timestream Write

내용

DataModel

유형: DataModel 객체

필수 여부: 아니요

DataModelS3Configuration

유형: DataModelS3Configuration 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

DataModelS3Configuration
서비스: Amazon Timestream Write

내용

BucketName

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 63입니다.

패턴: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

Required: No

ObjectKey

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 1024입니다.

패턴: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

DataSourceConfiguration
서비스: Amazon Timestream Write

데이터 소스에 대한 구성 세부 정보를 정의합니다.

내용

DataFormat

현재 입니다CSV.

타입: 문자열

유효 값: CSV

필수 사항 여부: 예

DataSourceS3Configuration

로드할 데이터가 포함된 파일의 S3 위치 구성입니다.

유형: DataSourceS3Configuration 객체

필수 여부: 예

CsvConfiguration

열 구분 기호가 쉼표일 수 있고 레코드 구분 기호가 새 줄 문자인 구분된 데이터 형식입니다.

유형: CsvConfiguration 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

DataSourceS3Configuration
서비스: Amazon Timestream Write

내용

BucketName

고객 S3 버킷의 버킷 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 63입니다.

Pattern: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

필수 여부: 예

ObjectKeyPrefix

유형: String

길이 제약: 최소 길이 1. 최대 길이는 1024입니다.

패턴: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Dimension
서비스: Amazon Timestream Write

시계열의 메타데이터 속성을 나타냅니다. 예를 들어 EC2 인스턴스의 이름과 가용 영역 또는 풍력 터빈 제조업체 이름은 차원입니다.

내용

Name

Dimension은 시계열의 메타데이터 속성을 나타냅니다. 예를 들어 EC2 인스턴스의 이름과 가용 영역 또는 풍력 터빈 제조업체 이름은 차원입니다.

차원 이름에 대한 제약 조건은 이름 지정 제약 조건을 참조하세요.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 60입니다.

필수 여부: 예

Value

차원의 값입니다.

유형: 문자열

필수 항목 여부: 예

DimensionValueType

시계열 데이터 포인트에 대한 차원의 데이터 유형입니다.

타입: 문자열

유효 값: VARCHAR

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용

- AWS SDK Java V2용
- AWS SDK Ruby V3용

DimensionMapping
서비스: Amazon Timestream Write

내용

DestinationColumn

유형: 문자열

길이 제약: 최소 길이 1.

필수 여부: 아니요

SourceColumn

유형: 문자열

길이 제약: 최소 길이 1.

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Endpoint
서비스: Amazon Timestream Write

API 전화를 걸 수 있는 사용 가능한 엔드포인트와 해당 엔드포인트에 TTL 대한 를 나타냅니다.

내용

Address

엔드포인트 주소입니다.

유형: 문자열

필수 항목 여부: 예

CachePeriodInMinutes

엔드포인트TTL의 를 분 단위로 표시합니다.

타입: Long

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MagneticStoreRejectedDataLocation
서비스: Amazon Timestream Write

마그네틱 스토어 쓰기 중 비동기적으로 거부된 레코드에 대한 오류 보고서를 작성할 위치입니다.

내용

S3Configuration

마그네틱 스토어 쓰기 중 비동기적으로 거부된 레코드에 대한 오류 보고서를 작성할 S3 위치의 구성입니다.

유형: S3Configuration 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MagneticStoreWriteProperties
서비스: Amazon Timestream Write

마그네틱 스토어 쓰기를 구성하기 위한 테이블의 속성 세트입니다.

내용

EnableMagneticStoreWrites

마그네틱 스토어 쓰기를 활성화하는 플래그입니다.

타입: 부울

필수 여부: 예

MagneticStoreRejectedDataLocation

마그네틱 스토어 쓰기 중 비동기적으로 거부된 레코드에 대한 오류 보고서를 작성할 위치입니다.

유형: MagneticStoreRejectedDataLocation 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MeasureValue
서비스: Amazon Timestream Write

시계열의 데이터 속성을 나타냅니다. 예를 들어 EC2 인스턴스 또는 RPM 풍력 터빈의 CPU 사용률은 measures. MeasureValue has 이름과 값입니다.

MeasureValue 는 유형에만 허용됩니다`MULTI`. `MULTI` 유형을 사용하면 단일 레코드에서 동일한 시계열과 연결된 여러 데이터 속성을 전달할 수 있습니다.

내용

Name

의 이름입니다 MeasureValue.

MeasureValue 이름에 대한 제약 조건은 Amazon Timestream 개발자 안내서의 이름 지정 제약 조건을 참조하세요.

유형: 문자열

길이 제약: 최소 길이 1.

필수 여부: 예

Type

시계열 데이터 포인트에 MeasureValue 대한 의 데이터 유형을 포함합니다.

타입: 문자열

유효 값: `DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI`

필수 사항 여부: 예

Value

의 값입니다 MeasureValue. 자세한 내용은 데이터 유형 섹션을 참조하세요.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MixedMeasureMapping
서비스: Amazon Timestream Write

내용

MeasureValueType

타입: 문자열

유효 값: DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI

필수 사항 여부: 예

MeasureName

유형: String

길이 제약: 최소 길이 1.

필수 여부: 아니요

MultiMeasureAttributeMappings

유형: MultiMeasureAttributeMapping객체 어레이

배열 구성원: 최소수는 1개입니다.

필수 여부: 아니요

SourceColumn

유형: 문자열

길이 제약: 최소 길이 1.

필수 여부: 아니요

TargetMeasureName

유형: 문자열

길이 제약: 최소 길이 1.

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MultiMeasureAttributeMapping
서비스: Amazon Timestream Write

내용

SourceColumn

유형: 문자열

길이 제약: 최소 길이 1.

필수 항목 여부: 예

MeasureValueType

타입: 문자열

유효 값: `DOUBLE | BIGINT | BOOLEAN | VARCHAR | TIMESTAMP`

필수 여부: 아니요

TargetMultiMeasureAttributeName

유형: 문자열

길이 제약: 최소 길이 1.

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MultiMeasureMappings
서비스: Amazon Timestream Write

내용

MultiMeasureAttributeMappings

유형: MultiMeasureAttributeMapping객체 어레이

어레이 멤버: 최소 항목 수 1개.

필수 항목 여부: 예

TargetMultiMeasureName

유형: String

길이 제약: 최소 길이 1.

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

PartitionKey
서비스: Amazon Timestream Write

테이블에서 데이터를 파티셔닝하는 데 사용되는 속성입니다. 차원 키는 차원 이름에 의해 파티션 키로 지정된 차원 값을 사용하여 데이터를 분할하는 반면, 측정 키는 측정 이름('measure_name' 열의 값)을 사용하여 데이터를 분할합니다.

내용

Type

파티션 키의 유형입니다. 옵션은 DIMENSION (치수 키) 및 MEASURE (측정 키)입니다.

타입: 문자열

유효 값: `DIMENSION | MEASURE`

필수 사항 여부: 예

EnforcementInRecord

수집된 레코드의 차원 키 사양에 대한 적용 수준입니다. 옵션은 REQUIRED (치수 키를 지정해야 함) 및 OPTIONAL (치수 키를 지정할 필요가 없음)입니다.

타입: 문자열

유효 값: `REQUIRED | OPTIONAL`

필수 여부: 아니요

Name

차원 키에 사용되는 속성의 이름입니다.

유형: 문자열

길이 제약: 최소 길이 1.

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Record
서비스: Amazon Timestream Write

Timestream에 기록되는 시계열 데이터 포인트를 나타냅니다. 각 레코드에는 일련의 차원이 포함됩니다. 차원은 인스턴스 이름 또는 인스턴스의 가용 영역과 같은 시계열 데이터 포인트의 메타데이터 속성을 나타냅니다EC2. 레코드에는 수집되는 측정값의 이름인 측정값 이름도 포함됩니다(예: EC2 인스턴스 CPU 사용률). 또한 레코드에는 측정값과 측정값의 데이터 유형인 값 유형이 포함됩니다. 또한 레코드에는 측정값이 수집된 시점의 타임스탬프와 타임스탬프의 세분성을 나타내는 타임스탬프 단위가 포함됩니다.

레코드에는 데이터 포인트를 업데이트하는 데 사용할 수 `long` 있는 64비트인 `Version` 필드가 있습니다. 동일한 차원, 타임스탬프 및 측정 이름을 가진 중복 레코드를 작성하지만 다른 측정 값은 쓰기 요청의 레코드 `Version` 속성이 기존 레코드 속성보다 높은 경우에만 성공합니다. Timestream은 `Version` 필드가 없는 1 레코드의 경우 기본적으로 `Version` 의 로 설정됩니다.

내용

Dimensions

시계열 데이터 포인트의 차원 목록을 포함합니다.

유형: Dimension객체 어레이

배열 멤버: 최대 항목 수는 128개입니다.

필수 여부: 아니요

MeasureName

측정값은 시계열의 데이터 속성을 나타냅니다. 예를 들어 EC2 인스턴스 또는 풍력 터빈RPM의 CPU 사용률은 측정값입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 256.

필수 여부: 아니요

MeasureValue

시계열 데이터 포인트의 측정값을 포함합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

MeasureValues

시계열 데이터 포인트에 MeasureValue 대한 목록을 포함합니다.

유형 에만 허용됩니다`MULTI`. 스칼라 값의 경우 레코드의 `MeasureValue` 속성을 직접 사용합니다.

유형: MeasureValue 객체 배열

필수 여부: 아니요

MeasureValueType

시계열 데이터 포인트에 대한 측정값의 데이터 유형을 포함합니다. 기본 유형은 입니다`DOUBLE`. 자세한 내용은 데이터 유형 섹션을 참조하세요.

타입: 문자열

유효 값: `DOUBLE | BIGINT | VARCHAR | BOOLEAN | TIMESTAMP | MULTI`

필수 여부: 아니요

Time

데이터 포인트의 측정값이 수집된 시간을 포함합니다. 시간 값과 단위는 에폭 이후 경과된 시간을 제공합니다. 예를 들어 시간 값이 `12345` 이고 단위가 인 경우 `ms12345 ms`가 에폭 이후 경과한 것입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 256.

필수 여부: 아니요

TimeUnit

타임스탬프 단위의 세분화입니다. 시간 값이 초, 밀리초, 나노초 또는 기타 지원되는 값인지 나타냅니다. 기본값은 `MILLISECONDS`입니다.

타입: 문자열

유효 값: `MILLISECONDS | SECONDS | MICROSECONDS | NANOSECONDS`

필수 여부: 아니요

Version

레코드 업데이트에 사용되는 64비트 속성입니다. 버전 번호가 더 높은 중복 데이터에 대한 쓰기 요청은 기존 측정값 및 버전을 업데이트합니다. 측정값이 동일한 경우에도 `Version` 는 계속 업데이트됩니다. 기본값은 1입니다.

> **Note**
>
> `Version` 가 1 이상이어야 합니다. 그렇지 않으면 `ValidationException` 오류가 발생합니다.

유형: Long

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

RecordsIngested
서비스: Amazon Timestream Write

이 요청에 의해 수집된 레코드에 대한 정보입니다.

내용

MagneticStore

마그네틱 스토어에 수집된 레코드 수입니다.

유형: 정수

필수 항목 여부: 아니요

MemoryStore

메모리 스토어에 수집된 레코드 수입니다.

유형: 정수

필수 항목 여부: 아니요

Total

성공적으로 수집된 레코드의 총 수입니다.

유형: 정수

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

RejectedRecord
서비스: Amazon Timestream Write

시계열 데이터를 시스템에 다시 삽입하기 전에 해결해야 하는 데이터 검증 문제로 인해 Timestream에 성공적으로 삽입되지 않은 레코드를 나타냅니다.

내용

ExistingVersion

레코드의 기존 버전입니다. 이 값은 쓰기 요청의 버전보다 버전이 높은 동일한 레코드가 존재하는 시나리오에 채워집니다.

유형: Long

필수 여부: 아니요

Reason

레코드가 Timestream에 성공적으로 삽입되지 않은 이유입니다. 가능한 실패 원인은 다음과 같습니다.

- 동일한 차원, 타임스탬프 및 측정값 이름을 가진 레코드가 여러 개 있지만 다음과 같은 중복 데이터가 있는 레코드:
  - 측정값이 다릅니다.
  - 요청에 버전이 없거나 새 레코드의 버전 값이 기존 값보다 작거나 같습니다.

  Timestream이 이 사례에 대한 데이터를 거부하면 `RejectedRecords` 응답의 `ExistingVersion` 필드에 현재 레코드의 버전이 표시됩니다. 업데이트를 강제로 적용하려면 레코드 세트의 버전이 보다 큰 값으로 요청을 재전송할 수 있습니다`ExistingVersion`.
- 메모리 스토어의 보존 기간을 벗어나는 타임스탬프가 있는 레코드입니다.

  > **Note**
  >
  > 보존 기간이 업데이트되면 새 기간 내에 데이터를 즉시 수집하려고 하면 `RejectedRecords` 예외가 발생합니다. `RejectedRecords` 예외를 방지하려면 새 창의 기간이 끝날 때까지 기다렸다가 새 데이터를 수집합니다. 자세한 내용은 Timestream 구성을 위한 모범 사례 및 Timestream 에서 스토리지가 작동하는 방식에 대한 설명을 참조하세요.
- Timestream에서 정의한 한도를 초과하는 차원 또는 측정값이 있는 레코드입니다.

자세한 내용은 Timestream 개발자 안내서의 액세스 관리를 참조하세요.

유형: 문자열

필수 항목 여부: 아니요

RecordIndex

에 대한 입력 요청의 레코드 인덱스입니다 WriteRecords. 인덱스는 0으로 시작합니다.

유형: 정수

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ReportConfiguration
서비스: Amazon Timestream Write

배치 로드 태스크에 대한 구성 보고 여기에는 오류 보고서가 저장되는 위치에 대한 세부 정보가 포함되어 있습니다.

내용

ReportS3Configuration

배치 로드에 대한 오류 보고서 및 이벤트를 작성하기 위한 S3 위치 구성.

유형: ReportS3Configuration 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ReportS3Configuration
서비스: Amazon Timestream Write

내용

BucketName

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 63입니다.

Pattern: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

필수 여부: 예

EncryptionOption

타입: 문자열

유효 값: `SSE_S3 | SSE_KMS`

필수 여부: 아니요

KmsKeyId

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

ObjectKeyPrefix

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 928입니다.

패턴: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

RetentionProperties
서비스: Amazon Timestream Write

보존 속성에는 마그네틱 스토어와 메모리 스토어에 시계열 데이터를 저장해야 하는 기간이 포함됩니다.

내용

MagneticStoreRetentionPeriodInDays

마그네틱 스토어에 데이터가 저장되어야 하는 기간입니다.

타입: Long

유효 범위: 최소값 1. 최대값은 73,000입니다.

필수 여부: 예

MemoryStoreRetentionPeriodInHours

메모리 스토어에 데이터가 저장되어야 하는 기간입니다.

타입: Long

유효 범위: 최소값 1. 최대값은 8766입니다.

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

S3Configuration
서비스: Amazon Timestream Write

S3 위치를 지정하는 구성입니다.

내용

BucketName

고객 S3 버킷의 버킷 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 63입니다.

패턴: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

Required: No

EncryptionOption

고객 S3 위치에 대한 암호화 옵션입니다. 옵션은 S3 관리형 키 또는 관리형 AWS 키를 사용한 S3 서버 측 암호화입니다.

타입: 문자열

유효 값: `SSE_S3 | SSE_KMS`

필수 여부: 아니요

KmsKeyId

AWS 관리형 AWS KMS 키로 암호화할 때 고객 S3 위치의 키 ID입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

ObjectKeyPrefix

고객 S3 위치의 객체 키 미리 보기입니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 928입니다.

패턴: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Schema

서비스: Amazon Timestream Write

스키마는 테이블의 예상 데이터 모델을 지정합니다.

내용

CompositePartitionKey

테이블 데이터를 분할하는 데 사용되는 속성을 정의하는 빈 파티션 키 목록입니다. 목록의 순서에 따라 파티션 계층 구조가 결정됩니다. 각 파티션 키의 이름과 유형 및 파티션 키 순서는 테이블이 생성된 후에는 변경할 수 없습니다. 그러나 각 파티션 키의 적용 수준을 변경할 수 있습니다.

유형: PartitionKey객체 어레이

배열 구성원: 최소수는 1개입니다.

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Table

서비스: Amazon Timestream Write

Timestream의 데이터베이스 테이블을 나타냅니다. 테이블에는 하나 이상의 관련 시계열이 포함됩니다. 테이블의 메모리 스토어와 마그네틱 스토어의 보존 기간을 수정할 수 있습니다.

내용

Arn

이 테이블을 고유하게 식별하는 Amazon 리소스 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

CreationTime

Timestream 테이블이 생성된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

DatabaseName

이 테이블을 포함하는 Timestream 데이터베이스의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 아니요

LastUpdatedTime

Timestream 테이블이 마지막으로 업데이트된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

MagneticStoreWriteProperties

마그네틱 스토어 쓰기를 활성화할 때 테이블에 설정할 속성이 들어 있습니다.

유형: MagneticStoreWriteProperties 객체

필수 여부: 아니요

RetentionProperties

메모리 저장소 및 마그네틱 저장소에 대한 보존 기간입니다.

유형: RetentionProperties 객체

필수 여부: 아니요

Schema

테이블의 스키마입니다.

유형: Schema 객체

필수 여부: 아니요

TableName

Timestream 테이블의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 256.

필수 여부: 아니요

TableStatus

테이블의 현재 상태:

- DELETING - 테이블이 삭제되고 있습니다.
- ACTIVE - 테이블을 사용할 준비가 되었습니다.

타입: 문자열

유효 값: ACTIVE | DELETING | RESTORING

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Tag
서비스: Amazon Timestream Write

태그는 목적, 소유자 또는 환경별로 Timestream 데이터베이스 and/or table. Each tag consists of a key and an optional value, both of which you define. With tags, you can categorize databases and/or 테이블에 할당하는 레이블입니다.

내용

Key

태그의 키입니다. 태그 키는 대소문자를 구별합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 128.

필수 여부: 예

Value

태그의 값입니다. 태그 값은 대/소문자를 구분하며 null일 수 있습니다.

유형: 문자열

길이 제약: 최소 길이는 0. 최대 길이는 256.

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Amazon Timestream 쿼리

Amazon Timestream Query에서 지원하는 데이터 유형은 다음과 같습니다.

- ColumnInfo

- Datum
- DimensionMapping
- Endpoint
- ErrorReportConfiguration
- ErrorReportLocation
- ExecutionStats
- MixedMeasureMapping
- MultiMeasureAttributeMapping
- MultiMeasureMappings
- NotificationConfiguration
- ParameterMapping
- QueryInsights
- QueryInsightsResponse
- QuerySpatialCoverage
- QuerySpatialCoverageMax
- QueryStatus
- QueryTemporalRange
- QueryTemporalRangeMax
- Row
- S3Configuration
- S3ReportLocation
- ScheduleConfiguration
- ScheduledQuery
- ScheduledQueryDescription
- ScheduledQueryInsights
- ScheduledQueryInsightsResponse
- ScheduledQueryRunSummary
- SelectColumn
- SnsConfiguration
- Tag

- TargetConfiguration
- TargetDestination
- TimeSeriesDataPoint
- TimestreamConfiguration
- TimestreamDestination
- Type

ColumnInfo
서비스: Amazon Timestream Query

열 이름, 데이터 유형 및 기타 속성과 같은 쿼리 결과에 대한 메타데이터를 포함합니다.

내용

Type

결과 세트 열의 데이터 유형입니다. 데이터 유형은 스칼라 또는 복합 데이터일 수 있습니다. Scalar 데이터 유형은 정수, 문자열, 더블, 부울 등입니다. 복잡한 데이터 유형은 배열, 행 등과 같은 유형입니다.

유형: Type 객체

필수 여부: 예

Name

결과 세트 열의 이름입니다. 결과 세트의 이름은 배열을 제외한 모든 데이터 유형의 열에 사용할 수 있습니다.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Datum
서비스: Amazon Timestream Query

기준점은 쿼리 결과의 단일 데이터 포인트를 나타냅니다.

내용

ArrayValue

데이터 포인트가 배열인지 여부를 나타냅니다.

유형: Datum 객체 배열

필수 여부: 아니요

NullValue

데이터 포인트가 null인지 여부를 나타냅니다.

타입: 부울

필수 항목 여부: 아니요

RowValue

데이터 포인트가 행인지 여부를 나타냅니다.

유형: Row 객체

필수 여부: 아니요

ScalarValue

데이터 포인트가 정수, 문자열, 이중 또는 부울과 같은 스칼라 값인지 여부를 나타냅니다.

유형: 문자열

필수 항목 여부: 아니요

TimeSeriesValue

데이터 포인트가 시간 데이터 유형인지 여부를 나타냅니다.

유형: TimeSeriesDataPoint 객체 배열

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

DimensionMapping
서비스: Amazon Timestream Query

이 유형은 쿼리 결과의 열을 대상 테이블의 차원에 매핑하는 데 사용됩니다.

내용

DimensionValueType

차원의 유형입니다.

타입: 문자열

유효 값: VARCHAR

필수 사항 여부: 예

Name

쿼리 결과의 열 이름입니다.

유형: 문자열

필수 항목 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Endpoint
서비스: Amazon Timestream Query

API 전화를 걸 수 있는 사용 가능한 엔드포인트와 해당 엔드포인트에 TTL 대한 를 나타냅니다.

내용

Address

엔드포인트 주소입니다.

유형: 문자열

필수 항목 여부: 예

CachePeriodInMinutes

엔드포인트TTL의 를 분 단위로 표시합니다.

타입: Long

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ErrorReportConfiguration
서비스: Amazon Timestream Query

오류 보고에 필요한 구성입니다.

내용

S3Configuration

오류 보고서에 대한 S3 구성입니다.

유형: S3Configuration 객체

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ErrorReportLocation
서비스: Amazon Timestream Query

여기에는 단일 예약 쿼리 호출에 대한 오류 보고서의 위치가 포함됩니다.

내용

S3ReportLocation

오류 보고서가 작성되는 S3 위치입니다.

유형: S3ReportLocation 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ExecutionStats
서비스: Amazon Timestream Query

단일 예약 쿼리 실행에 대한 통계입니다.

내용

BytesMetered

단일 예약 쿼리 실행에 대해 측정되는 바이트입니다.

유형: Long

필수 여부: 아니요

CumulativeBytesScanned

단일 예약 쿼리 실행에 대해 스캔된 바이트입니다.

유형: Long

필수 여부: 아니요

DataWrites

단일 예약 쿼리 실행에서 수집된 레코드에 대해 측정되는 데이터 쓰기입니다.

유형: Long

필수 여부: 아니요

ExecutionTimeInMillis

예약된 쿼리 실행을 완료하는 데 필요한 밀리초 단위로 측정된 총 시간입니다.

유형: Long

필수 여부: 아니요

QueryResultRows

대상 데이터 소스로 수집하기 전에 쿼리를 실행하여 출력에 있는 행 수입니다.

유형: Long

필수 여부: 아니요

RecordsIngested

단일 예약 쿼리 실행에 대해 수집된 레코드 수입니다.

유형: Long

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MixedMeasureMapping
서비스: Amazon Timestream Query

MixedMeasureMappings 는 파생 테이블의 좁은 측정값과 여러 측정값의 혼합으로 데이터를 수집하는 데 사용할 수 있는 매핑입니다.

내용

MeasureValueType

에서 읽을 값의 유형입니다sourceColumn. 매핑이 용인 경우 를 MULTI사용합니다 MeasureValueTypeMULTI.

타입: 문자열

유효 값: `BIGINT | BOOLEAN | DOUBLE | VARCHAR | MULTI`

필수 사항 여부: 예

MeasureName

결과 행의 measure_name 값을 참조합니다. MeasureNameColumn 이 제공된 경우 이 필드는 필수입니다.

유형: 문자열

필수 항목 여부: 아니요

MultiMeasureAttributeMappings

measureValueType 가 인 경우 필수입니다MULTI. MULTI 값 측정값에 대한 속성 매핑입니다.

유형: MultiMeasureAttributeMapping객체 어레이

배열 구성원: 최소수는 1개입니다.

필수 여부: 아니요

SourceColumn

이 필드는 결과 구체화를 위해 측정값을 읽을 소스 열을 나타냅니다.

유형: 문자열

필수 항목 여부: 아니요

TargetMeasureName

사용할 대상 측정 이름입니다. 제공되지 않는 경우 기본적으로 대상 측정 이름은 제공된 경우 측정 이름이 되고 그렇지 sourceColumn 않은 경우 측정 이름이 됩니다.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MultiMeasureAttributeMapping
서비스: Amazon Timestream Query

MULTI 값 측정값에 대한 속성 매핑입니다.

내용

MeasureValueType

소스 열에서 읽을 속성의 유형입니다.

타입: 문자열

유효 값: `BIGINT | BOOLEAN | DOUBLE | VARCHAR | TIMESTAMP`

필수 사항 여부: 예

SourceColumn

속성 값을 읽을 소스 열입니다.

유형: 문자열

필수 항목 여부: 예

TargetMultiMeasureAttributeName

파생 테이블의 속성 이름에 사용할 사용자 지정 이름입니다. 제공되지 않으면 소스 열 이름이 사용됩니다.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

MultiMeasureMappings
서비스: Amazon Timestream Query

MixedMeasureMappings 또는 중 하나만 MultiMeasureMappings 제공됩니다. MultiMeasureMappings 는 파생 테이블에서 데이터를 다중 측정값으로 수집하는 데 사용할 수 있습니다.

내용

MultiMeasureAttributeMappings

필수 사항입니다. 다중 측정 속성에 대한 데이터를 모으기 위해 쿼리 결과를 매핑하는 데 사용할 속성 매핑입니다.

유형: MultiMeasureAttributeMapping객체 어레이

어레이 멤버: 최소 항목 수 1개.

필수 여부: 예

TargetMultiMeasureName

파생 테이블에 있는 대상 다중 측정 이름입니다. 이 입력은 measureNameColumn 이 제공되지 않은 경우 필요합니다. MeasureNameColumn 가 제공되면 해당 열의 값이 다중 측정 이름으로 사용됩니다.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

NotificationConfiguration
서비스: Amazon Timestream Query

예약된 쿼리에 대한 알림 구성입니다. 예약된 쿼리가 생성되거나 상태가 업데이트 또는 삭제될 때 Timestream에서 알림을 보냅니다.

내용

SnsConfiguration

SNS 구성에 대한 세부 정보입니다.

유형: SnsConfiguration 객체

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ParameterMapping
서비스: Amazon Timestream Query

명명된 파라미터에 대한 매핑.

내용

Name

파라미터 이름입니다.

유형: 문자열

필수 항목 여부: 예

Type

쿼리 결과 세트에 있는 열의 데이터 유형을 포함합니다. 데이터 유형은 스칼라 또는 복합 데이터일 수 있습니다. 지원되는 스칼라 데이터 유형은 정수, 부울, 문자열, 이중, 타임스탬프, 날짜, 시간 및 간격입니다. 지원되는 복잡한 데이터 유형은 배열, 행 및 시간입니다.

유형: Type 객체

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

QueryInsights
서비스: Amazon Timestream Query

QueryInsights 는 쿼리를 최적화하고 비용을 절감하며 성능을 개선하는 데 도움이 되는 성능 튜닝 기능입니다. 를 사용하면 쿼리의 정리 효율성을 평가하고 개선 영역을 식별하여 쿼리 성능을 향상시킬 QueryInsights수 있습니다. 를 사용하면 시간 및 공간 정리 측면에서 쿼리의 효과를 분석하고 성능을 개선할 기회를 식별할 QueryInsights수도 있습니다. 특히 쿼리가 시간 기반 및 파티션 키 기반 인덱싱 전략을 얼마나 잘 사용하여 데이터 검색을 최적화하는지 평가할 수 있습니다. 쿼리 성능을 최적화하려면 쿼리 실행을 제어하는 시간 파라미터와 공간 파라미터를 모두 미세 조정해야 합니다.

에서 제공하는 주요 지표는 QuerySpatialCoverage 및 QueryInsights입니다QueryTemporalRange. 는 쿼리가 스캔하는 공간 축의 양을 QuerySpatialCoverage 나타내며, 값이 낮을수록 더 효율적입니다. QueryTemporalRange는 스캔한 시간 범위를 나타내며, 범위가 좁을수록 더 성능이 뛰어납니다.

의 이점 QueryInsights

다음은 를 사용할 때 얻을 수 있는 주요 이점입니다QueryInsights.

- 비효율적인 쿼리 식별 - 쿼리에서 액세스하는 테이블의 시간 기반 및 속성 기반 정리에 대한 정보를 QueryInsights 제공합니다. 이 정보는 최적이 아닌 방식으로 액세스되는 테이블을 식별하는 데 도움이 됩니다.
- 데이터 모델 및 파티셔닝 최적화 - QueryInsights 정보를 사용하여 데이터 모델 및 파티셔닝 전략에 액세스하고 미세 조정할 수 있습니다.
- 쿼리 조정 - 인덱스를 더 효과적으로 사용할 수 있는 기회를 QueryInsights 강조합니다.

> **Note**
>
> QueryInsights 활성화된 상태로 수행할 수 있는 최대 Query API 요청 수는 초당 쿼리 1개() 입니다QPS. 이 쿼리 속도를 초과하면 제한이 발생할 수 있습니다.

내용

Mode

를 활성화하는 다음 모드를 제공합니다QueryInsights.

- ENABLED_WITH_RATE_CONTROL - 처리 중인 쿼리QueryInsights를 활성화합니다. 이 모드에는 초당 쿼리 1개()로 QueryInsights 기능을 제한하는 속도 제어 메커니즘도 포함되어 있습니다QPS.
- DISABLED – 를 비활성화합니다QueryInsights.

타입: 문자열

유효 값: ENABLED_WITH_RATE_CONTROL | DISABLED

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

QueryInsightsResponse
서비스: Amazon Timestream Query

실행한 쿼리와 관련된 다양한 인사이트 및 지표를 제공합니다.

내용

OutputBytes

쿼리 결과 세트의 크기를 바이트 단위로 나타냅니다. 이 데이터를 사용하여 쿼리 튜닝 연습의 일환으로 결과 세트가 변경되었는지 확인할 수 있습니다.

유형: Long

필수 여부: 아니요

OutputRows

쿼리 결과 세트의 일부로 반환된 총 행 수를 나타냅니다. 이 데이터를 사용하여 쿼리 튜닝 연습의 일환으로 결과 세트의 행 수가 변경되었는지 확인할 수 있습니다.

유형: Long

필수 여부: 아니요

QuerySpatialCoverage

최적이 아닌(최대) 공간 정리가 있는 테이블을 포함하여 쿼리의 공간 범위에 대한 인사이트를 제공합니다. 이 정보는 파티셔닝 전략에서 개선해야 할 영역을 식별하여 공간 정리를 개선하는 데 도움이 될 수 있습니다.

유형: QuerySpatialCoverage 객체

필수 여부: 아니요

QueryTableCount

쿼리의 테이블 수를 나타냅니다.

유형: Long

필수 여부: 아니요

QueryTemporalRange

시간 범위가 가장 큰(최대) 테이블을 포함하여 쿼리의 시간 범위에 대한 인사이트를 제공합니다. 다음은 시간 기반 정리를 최적화하기 위한 몇 가지 잠재적 옵션입니다.

- 누락된 시간 예측을 추가합니다.
- 시간 표시 주변의 함수를 제거합니다.
- 모든 하위 쿼리에 시간 예측을 추가합니다.

유형: QueryTemporalRange 객체

필수 여부: 아니요

UnloadPartitionCount

`Unload` 작업에 의해 생성된 파티션을 나타냅니다.

유형: Long

필수 여부: 아니요

UnloadWrittenBytes

`Unload` 작업이 작성한 크기를 바이트 단위로 나타냅니다.

유형: Long

필수 여부: 아니요

UnloadWrittenRows

`Unload` 쿼리에서 쓰는 행을 나타냅니다.

유형: Long

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

QuerySpatialCoverage
서비스: Amazon Timestream Query

최적이 아닌(최대) 공간 정리가 있는 테이블을 포함하여 쿼리의 공간 범위에 대한 인사이트를 제공합니다. 이 정보는 파티셔닝 전략에서 개선이 필요한 영역을 식별하여 공간 정리를 개선하는 데 도움이 될 수 있습니다.

예를 들어 `QuerySpatialCoverage` 정보를 사용하여 다음을 수행할 수 있습니다.

- measure_name을 추가하거나 고객 정의 파티션 키(CDPK) 표현자를 사용합니다.
- 이전 작업을 이미 수행한 경우 주변 함수 또는 와 같은 절을 제거합니다`LIKE`.

내용

Max

실행된 쿼리의 공간 범위와 가장 비효율적인 공간 정리를 사용하는 테이블에 대한 인사이트를 제공합니다.

- `Value` – 공간 적용 범위의 최대 비율입니다.
- `TableArn` – 최적이 아닌 공간 정리가 있는 테이블의 Amazon 리소스 이름(ARN)입니다.
- `PartitionKey` – 파티셔닝에 사용되는 파티션 키로, 기본값 `measure_name` 또는 가 될 수 있습니다CDPK.

유형: QuerySpatialCoverageMax 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

QuerySpatialCoverageMax
서비스: Amazon Timestream Query

쿼리로 스캔한 가장 최적이 아닌 공간 범위를 가진 테이블에 대한 인사이트를 제공합니다.

내용

PartitionKey

파티셔닝에 사용되는 파티션 키로, 기본값 `measure_name` 또는 고객 정의 파티션 키일 수 있습니다.

유형: String 배열

필수 여부: 아니요

TableArn

가장 최적이 아닌 공간 정리가 있는 테이블의 Amazon 리소스 이름(ARN)입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

Value

공간 적용 범위의 최대 비율입니다.

유형: 더블

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

QueryStatus
서비스: Amazon Timestream Query

진행 상황 및 스캔된 바이트를 포함하여 쿼리 상태에 대한 정보입니다.

내용

CumulativeBytesMetered

쿼리에서 스캔한 데이터의 양은 요금이 청구되는 바이트입니다. 누적 합계이며 쿼리가 시작된 이후 청구될 총 데이터 양을 나타냅니다. 요금은 한 번만 적용되며 쿼리 실행이 완료될 때 또는 쿼리가 취소될 때 적용됩니다.

유형: Long

필수 여부: 아니요

CumulativeBytesScanned

쿼리에서 스캔한 데이터의 양은 바이트입니다. 누적 합계이며 쿼리가 시작된 이후 스캔된 총 바이트 수를 나타냅니다.

유형: Long

필수 여부: 아니요

ProgressPercentage

백분율로 표시되는 쿼리 진행률입니다.

유형: 더블

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

QueryTemporalRange
서비스: Amazon Timestream Query

시간 범위가 가장 큰(최대) 테이블을 포함하여 쿼리의 시간 범위에 대한 인사이트를 제공합니다.

내용

Max

시간 축에서 가장 최적화되지 않은 성능 테이블에 대한 인사이트를 제공하는 다음 속성을 캡슐화합니다.

- `Value` - 쿼리의 시작과 종료 사이의 나노초 단위 최대 기간입니다.
- `TableArn` - 가장 긴 시간 범위로 쿼리되는 테이블의 Amazon 리소스 이름(ARN)입니다.

유형: QueryTemporalRangeMax 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

QueryTemporalRangeMax
서비스: Amazon Timestream Query

쿼리로 스캔한 시간적 정리가 가장 최적화되지 않은 테이블에 대한 인사이트를 제공합니다.

내용

TableArn

가장 긴 시간 범위로 쿼리되는 테이블의 Amazon 리소스 이름(ARN)입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

Value

쿼리의 시작과 종료 사이의 나노초 단위 최대 기간입니다.

유형: Long

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Row
서비스: Amazon Timestream Query

쿼리 결과의 단일 행을 나타냅니다.

내용

Data

결과 세트의 단일 행에 있는 데이터 포인트 목록입니다.

유형: Datum객체 어레이

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

S3Configuration
서비스: Amazon Timestream Query

쿼리 실행으로 인한 오류 보고서의 S3 위치에 대한 세부 정보입니다.

내용

BucketName

오류 보고서가 생성될 S3 버킷의 이름입니다.

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 63입니다.

Pattern: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

필수 여부: 예

EncryptionOption

오류 보고서에 대한 저장 시 암호화 옵션입니다. 암호화 옵션이 지정되지 않은 경우 Timestream은 SSE_S3를 기본값으로 선택합니다.

타입: 문자열

유효 값: `SSE_S3 | SSE_KMS`

필수 여부: 아니요

ObjectKeyPrefix

오류 보고서 키의 접두사입니다. Timestream은 기본적으로 오류 보고서 경로에 다음 접두사를 추가합니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 896입니다.

패턴: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

S3ReportLocation
서비스: Amazon Timestream Query

예약된 쿼리 실행의 S3 보고서 위치입니다.

내용

BucketName

S3 버킷 이름

유형: 문자열

길이 제약 조건: 최소 길이는 3입니다. 최대 길이는 63입니다.

패턴: `[a-z0-9][\.\-a-z0-9]{1,61}[a-z0-9]`

Required: No

ObjectKey

S3 키.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ScheduleConfiguration
서비스: Amazon Timestream Query

쿼리 일정의 구성입니다.

내용

ScheduleExpression

예약된 쿼리 실행을 트리거할 시기를 나타내는 표현식입니다. cron 표현식 또는 rate 표현식일 수 있습니다.

유형: 문자열

길이 제약 조건: 최소 길이는 1입니다. 최대 길이는 256.

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ScheduledQuery
서비스: Amazon Timestream Query

예약된 쿼리

내용

Arn

Amazon 리소스 이름입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

Name

예약된 쿼리의 이름입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

Pattern: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

필수 여부: 예

State

예약된 쿼리의 상태입니다.

타입: 문자열

유효 값: `ENABLED | DISABLED`

필수 사항 여부: 예

CreationTime

예약된 쿼리의 생성 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

ErrorReportConfiguration

예약된 쿼리 오류 보고를 위한 구성입니다.

유형: ErrorReportConfiguration 객체

필수 여부: 아니요

LastRunStatus

마지막으로 예약된 쿼리 실행의 상태입니다.

타입: 문자열

유효 값: `AUTO_TRIGGER_SUCCESS | AUTO_TRIGGER_FAILURE | MANUAL_TRIGGER_SUCCESS | MANUAL_TRIGGER_FAILURE`

필수 여부: 아니요

NextInvocationTime

다음에 예약된 쿼리를 실행할 때입니다.

유형: 타임스탬프

필수 여부: 아니요

PreviousInvocationTime

예약된 쿼리가 마지막으로 실행된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

TargetDestination

최종 예약 쿼리 결과를 작성할 대상 데이터 소스입니다.

유형: TargetDestination 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ScheduledQueryDescription
서비스: Amazon Timestream Query

예약된 쿼리를 설명하는 구조입니다.

내용

Arn

예약된 쿼리 ARN.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

Name

예약된 쿼리의 이름입니다.

유형: 문자열

길이 제한: 최소 길이는 1. 최대 길이는 64.

Pattern: `[a-zA-Z0-9|!\-_*'\(\)]([a-zA-Z0-9]|[!\-_*'\(\)\/.])+`

필수 여부: 예

NotificationConfiguration

알림 구성.

유형: NotificationConfiguration 객체

필수 여부: 예

QueryString

실행할 쿼리입니다.

유형: 문자열

길이 제약: 최소 길이 1. 최대 길이는 262144자입니다.

필수 여부: 예

ScheduleConfiguration

일정 구성입니다.

유형: ScheduleConfiguration 객체

필수 여부: 예

State

예약된 쿼리의 상태입니다.

타입: 문자열

유효 값: `ENABLED | DISABLED`

필수 사항 여부: 예

CreationTime

예약된 쿼리의 생성 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

ErrorReportConfiguration

예약된 쿼리에 대한 오류 보고 구성입니다.

유형: ErrorReportConfiguration 객체

필수 여부: 아니요

KmsKeyId

예약된 쿼리 리소스를 암호화하는 데 사용되는 고객이 제공한 KMS 키입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

LastRunSummary

마지막으로 예약된 쿼리 실행에 대한 런타임 요약입니다.

유형: ScheduledQueryRunSummary 객체

필수 여부: 아니요

NextInvocationTime

다음에 예약된 쿼리가 실행되도록 예약된 경우.

유형: 타임스탬프

필수 여부: 아니요

PreviousInvocationTime

쿼리가 마지막으로 실행된 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

RecentlyFailedRuns

마지막으로 실패한 예약된 쿼리 실행 5회에 대한 런타임 요약입니다.

유형: ScheduledQueryRunSummary 객체 배열

필수 여부: 아니요

ScheduledQueryExecutionRoleArn

IAM Timestream이 일정 쿼리를 실행하는 데 사용하는 역할입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 아니요

TargetConfiguration

예약된 쿼리 대상 스토어 구성입니다.

유형: TargetConfiguration 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ScheduledQueryInsights
서비스: Amazon Timestream Query

`QueryInsights` 에서 활성화하기 위한 설정을 캡슐화합니다`ExecuteScheduledQueryRequest`.

내용

Mode

를 활성화하는 다음 모드를 제공합니다`ScheduledQueryInsights`.

- `ENABLED_WITH_RATE_CONTROL` - 처리 중인 쿼리`ScheduledQueryInsights`를 활성화합니다. 이 모드에는 초당 쿼리 1개()로 `QueryInsights` 기능을 제한하는 속도 제어 메커니즘도 포함되어 있습니다QPS.
- `DISABLED` – 를 비활성화합니다`ScheduledQueryInsights`.

타입: 문자열

유효 값: `ENABLED_WITH_RATE_CONTROL | DISABLED`

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ScheduledQueryInsightsResponse
서비스: Amazon Timestream Query

실행ExecuteScheduledQueryRequest된 와 관련된 다양한 인사이트 및 지표를 제공합니다.

내용

OutputBytes

쿼리 결과 세트의 크기를 바이트 단위로 나타냅니다. 이 데이터를 사용하여 쿼리 튜닝 연습의 일환으로 결과 세트가 변경되었는지 확인할 수 있습니다.

유형: Long

필수 여부: 아니요

OutputRows

쿼리 결과 세트의 일부로 반환된 총 행 수를 나타냅니다. 이 데이터를 사용하여 쿼리 튜닝 연습의 일환으로 결과 세트의 행 수가 변경되었는지 확인할 수 있습니다.

유형: Long

필수 여부: 아니요

QuerySpatialCoverage

최적이 아닌(최대) 공간 정리가 있는 테이블을 포함하여 쿼리의 공간 범위에 대한 인사이트를 제공합니다. 이 정보는 파티셔닝 전략에서 개선이 필요한 영역을 식별하여 공간 정리를 개선하는 데 도움이 될 수 있습니다.

유형: QuerySpatialCoverage 객체

필수 여부: 아니요

QueryTableCount

쿼리의 테이블 수를 나타냅니다.

유형: Long

필수 여부: 아니요

QueryTemporalRange

시간 범위가 가장 큰(최대) 테이블을 포함하여 쿼리의 시간 범위에 대한 인사이트를 제공합니다. 다음은 시간 기반 정리를 최적화하기 위한 몇 가지 잠재적 옵션입니다.

- 누락된 시간 예측을 추가합니다.
- 시간 표시 주변의 함수를 제거합니다.
- 모든 하위 쿼리에 시간 기호를 추가합니다.

유형: QueryTemporalRange 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

ScheduledQueryRunSummary
서비스: Amazon Timestream Query

예약된 쿼리에 대한 요약 실행

내용

ErrorReportLocation

오류 보고서의 S3 위치입니다.

유형: ErrorReportLocation 객체

필수 여부: 아니요

ExecutionStats

예약된 실행에 대한 런타임 통계입니다.

유형: ExecutionStats 객체

필수 여부: 아니요

FailureReason

실패 시 예약된 쿼리에 대한 오류 메시지입니다. 자세한 오류 이유를 확인하려면 오류 보고서를 확인해야 할 수 있습니다.

유형: 문자열

필수 항목 여부: 아니요

InvocationTime

InvocationTime 이 실행을 위해. 쿼리가 실행되도록 예약된 시간입니다. 파라미터를 쿼리에 사용하여 값을 가져올 `@scheduled_runtime` 수 있습니다.

유형: 타임스탬프

필수 여부: 아니요

QueryInsightsResponse

예약된 쿼리의 실행 요약과 관련된 다양한 인사이트 및 지표를 제공합니다.

유형: ScheduledQueryInsightsResponse 객체

필수 여부: 아니요

RunStatus

예약된 쿼리 실행의 상태입니다.

타입: 문자열

유효 값: `AUTO_TRIGGER_SUCCESS | AUTO_TRIGGER_FAILURE | MANUAL_TRIGGER_SUCCESS | MANUAL_TRIGGER_FAILURE`

필수 여부: 아니요

TriggerTime

쿼리가 실행된 실제 시간입니다.

유형: 타임스탬프

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

SelectColumn
서비스: Amazon Timestream Query

쿼리에서 반환되는 열의 세부 정보입니다.

내용

Aliased

쿼리에서 열 이름을 별칭으로 지정한 경우 True입니다. 그렇지 않으면 거짓입니다.

타입: 부울

필수 항목 여부: 아니요

DatabaseName

이 열이 있는 데이터베이스입니다.

유형: 문자열

필수 항목 여부: 아니요

Name

열의 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

TableName

이 열이 있는 데이터베이스 내의 테이블입니다.

유형: 문자열

필수 항목 여부: 아니요

Type

쿼리 결과 세트에 있는 열의 데이터 유형을 포함합니다. 데이터 유형은 스칼라 또는 복합 데이터일 수 있습니다. 지원되는 스칼라 데이터 유형은 정수, 부울, 문자열, 이중, 타임스탬프, 날짜, 시간 및 간격입니다. 지원되는 복잡한 데이터 유형은 배열, 행 및 시간입니다.

유형: Type 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

SnsConfiguration
서비스: Amazon Timestream Query

알림을 보내는 데 필요한 에 대한 세부 정보SNS입니다.

내용

TopicArn

SNS ARN 예약된 쿼리 상태 알림을 보낼 주제입니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 2,048.

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Tag
서비스: Amazon Timestream Query

태그는 목적, 소유자 또는 환경별로 Timestream 데이터베이스 and/or table. Each tag consists of a key and an optional value, both of which you define. Tags enable you to categorize databases and/or 테이블에 할당하는 레이블입니다.

내용

Key

태그의 키입니다. 태그 키는 대소문자를 구별합니다.

유형: 문자열

길이 제약: 최소 길이는 1. 최대 길이는 128.

필수 여부: 예

Value

태그의 값입니다. 태그 값은 대소문자를 구분하며 null일 수 있습니다.

유형: 문자열

길이 제약: 최소 길이는 0. 최대 길이는 256.

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

TargetConfiguration
서비스: Amazon Timestream Query

쿼리의 출력을 쓰는 데 사용되는 구성입니다.

내용

TimestreamConfiguration

Timestream 데이터베이스 및 테이블에 데이터를 쓰는 데 필요한 구성입니다.

유형: TimestreamConfiguration 객체

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

TargetDestination
서비스: Amazon Timestream Query

대상 데이터 소스에 대한 데이터를 쓰기 위한 대상 세부 정보입니다. 현재 지원되는 데이터 소스는 Timestream입니다.

내용

TimestreamDestination

Timestream 데이터 소스에 대한 쿼리 결과 대상 세부 정보입니다.

유형: TimestreamDestination 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

TimeSeriesDataPoint
서비스: Amazon Timestream Query

시간 데이터 유형은 시간 경과에 따른 측정값의 값을 나타냅니다. 시계열은 타임스탬프 및 측정값의 행 배열이며 행은 오름차순으로 정렬됩니다. TimeSeriesDataPoint 는 시계열의 단일 데이터 포인트입니다. 시계열의 (시간, 측정값) 튜플을 나타냅니다.

내용

Time

측정값이 수집될 때의 타임스탬프입니다.

유형: 문자열

필수 항목 여부: 예

Value

데이터 포인트의 측정값입니다.

유형: Datum 객체

필수 여부: 예

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

TimestreamConfiguration
서비스: Amazon Timestream Query

Timestream 데이터베이스 및 테이블에 데이터를 쓰기 위한 구성입니다. 이 구성을 통해 사용자는 쿼리 결과 선택 열을 대상 테이블 열에 매핑할 수 있습니다.

내용

DatabaseName

쿼리 결과를 쓸 Timeststream 데이터베이스의 이름입니다.

유형: 문자열

필수 항목 여부: 예

DimensionMappings

이는 쿼리 결과의 열을 대상 테이블의 차원으로 매핑할 수 있도록 하기 위한 것입니다.

유형: DimensionMapping객체 어레이

필수 여부: 예

TableName

쿼리 결과를 쓸 Timeststream 테이블의 이름입니다. 테이블은 Timestream 구성에서 제공되는 동일한 데이터베이스 내에 있어야 합니다.

유형: 문자열

필수 항목 여부: 예

TimeColumn

대상 테이블에서 시간 열로 사용해야 하는 쿼리 결과의 열입니다. 이에 대한 열 유형은 이어야 합니다TIMESTAMP.

유형: 문자열

필수 항목 여부: 예

MeasureNameColumn

측정 열의 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

MixedMeasureMappings

다중 측정 레코드에 측정값을 매핑하는 방법을 지정합니다.

유형: MixedMeasureMapping객체 어레이

배열 구성원: 최소수는 1개입니다.

필수 여부: 아니요

MultiMeasureMappings

다중 측정 매핑입니다.

유형: MultiMeasureMappings 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

TimestreamDestination
서비스: Amazon Timestream Query

예약된 쿼리의 대상입니다.

내용

DatabaseName

Timestream 데이터베이스 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

TableName

Timestream 테이블 이름입니다.

유형: 문자열

필수 항목 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

Type
서비스: Amazon Timestream Query

쿼리 결과 세트에 있는 열의 데이터 유형을 포함합니다. 데이터 유형은 스칼라 또는 복합 데이터일 수 있습니다. 지원되는 스칼라 데이터 유형은 정수, 부울, 문자열, 이중, 타임스탬프, 날짜, 시간 및 간격입니다. 지원되는 복잡한 데이터 유형은 배열, 행 및 시간입니다.

내용

ArrayColumnInfo

열이 배열인지 여부를 나타냅니다.

유형: ColumnInfo 객체

필수 여부: 아니요

RowColumnInfo

열이 행인지 여부를 나타냅니다.

유형: ColumnInfo 객체 배열

필수 여부: 아니요

ScalarType

열의 유형 문자열, 정수, 부울, 이중, 타임스탬프, 날짜, 시간을 나타냅니다. 자세한 내용은 지원되는 데이터 유형 섹션을 참조하세요.

타입: 문자열

유효 값: `VARCHAR | BOOLEAN | BIGINT | DOUBLE | TIMESTAMP | DATE | TIME | INTERVAL_DAY_TO_SECOND | INTERVAL_YEAR_TO_MONTH | UNKNOWN | INTEGER`

필수 여부: 아니요

TimeSeriesMeasureValueColumnInfo

열이 시간 데이터 유형인지 여부를 나타냅니다.

유형: ColumnInfo 객체

필수 여부: 아니요

참고

언어별 중 하나API에서 이를 사용하는 방법에 대한 자세한 내용은 다음을 AWS SDKs참조하세요.

- AWS SDK C++용
- AWS SDK Java V2용
- AWS SDK Ruby V3용

일반적인 오류

이 섹션에서는 모든 AWS 서비스의 API 작업에 공통적인 오류를 나열합니다. 이 서비스의 API 작업과 관련된 오류는 해당 API 작업의 주제를 참조하세요.

AccessDeniedException

이 작업을 수행할 수 있는 충분한 액세스 권한이 없습니다.

HTTP 상태 코드: 400

IncompleteSignature

요청 서명이 AWS 표준을 준수하지 않습니다.

HTTP 상태 코드: 400

InternalFailure

알 수 없는 오류, 예외 또는 장애 때문에 요청 처리가 실패했습니다.

HTTP 상태 코드: 500

InvalidAction

요청된 동작 또는 작업이 유효하지 않습니다. 작업을 올바로 입력했는지 확인합니다.

HTTP 상태 코드: 400

InvalidClientTokenId

제공된 X.509 인증서 또는 AWS 액세스 키 ID가 레코드에 없습니다.

HTTP 상태 코드: 403

NotAuthorized

이 작업을 수행하려면 권한이 있어야 합니다.

HTTP 상태 코드: 400

OptInRequired

AWS 액세스 키 ID에는 서비스에 대한 구독이 필요합니다.

HTTP 상태 코드: 403

RequestExpired

요청이 요청의 날짜 스탬프 이후 15분 이상 또는 요청 만료 날짜(예: 미리 서명된 의 경우URLs) 이후 15분 이상 지나 서비스에 도달했거나 요청의 날짜 스탬프가 향후 15분 이상 남았습니다.

HTTP 상태 코드: 400

ServiceUnavailable

서버의 일시적 장애로 인해 요청이 실패했습니다.

HTTP 상태 코드: 503

ThrottlingException

요청 제한 때문에 요청이 거부되었습니다.

HTTP 상태 코드: 400

ValidationError

입력이 AWS 서비스에서 지정한 제약 조건을 충족하지 못합니다.

HTTP 상태 코드: 400

공통 파라미터

다음 목록에는 모든 작업이 쿼리 문자열을 사용하여 Signature Version 4 요청에 서명하는 데 사용하는 파라미터가 포함되어 있습니다. 작업별 파라미터는 그 작업에 대한 항목에 나열되어 있습니다. 서명 버전 4에 대한 자세한 내용은 IAM 사용 설명서의 요청 서명을 AWS API 참조하세요.

Action

수행할 작업입니다.

타입: 문자열

필수 항목 여부: 예

Version

요청이 작성되는 API 버전으로, 형식으로 표현됩니다 YYYY-MM-DD.

유형: 문자열

필수 항목 여부: 예

X-Amz-Algorithm

요청 서명을 생성하는 데 사용된 해시 알고리즘입니다.

조건: HTTP 권한 부여 헤더 대신 쿼리 문자열에 인증 정보를 포함할 때 이 파라미터를 지정합니다.

유형: 문자열

유효 값: `AWS4-HMAC-SHA256`

필수 항목 여부: 조건부

X-Amz-Credential

자격 증명 범위 값이며 액세스 키, 날짜, 대상으로 하는 리전, 요청하는 서비스 및 종료 문자열("aws4_request")이 포함된 문자열입니다. 값은 access_key //region YYYYMMDD/service / aws4_request 형식으로 표현됩니다.

자세한 내용은 IAM 사용 설명서의 서명된 AWS API 요청 생성을 참조하세요.

조건: HTTP 권한 부여 헤더 대신 쿼리 문자열에 인증 정보를 포함할 때 이 파라미터를 지정합니다.

유형: 문자열

필수 항목 여부: 조건부

X-Amz-Date

서명을 만드는 데 사용되는 날짜입니다. 형식은 ISO 8601 기본 형식(YYYYMMDD'T'HHMMSS'Z') 이어야 합니다. 예를 들어 다음 날짜 시간은 유효한 X-Amz-Date 값입니다`20120325T120000Z`.

조건: X-Amz-Date 모든 요청에 대해 선택 사항이며 요청에 서명하는 데 사용되는 날짜를 재정의하는 데 사용할 수 있습니다. 날짜 헤더가 ISO 8601 기본 형식으로 지정된 경우 X-Amz-Date는 필요하지 않습니다. X-Amz-Date 를 사용하면 항상 날짜 헤더의 값을 재정의합니다. 자세한 내용은 IAM 사용 설명서의 AWS API 요청 서명 요소를 참조하세요.

유형: 문자열

필수 항목 여부: 조건부

X-Amz-Security-Token

AWS Security Token Service ()에 대한 호출을 통해 얻은 임시 보안 토큰입니다AWS STS. 에서 임시 보안 자격 증명을 지원하는 서비스 목록은 IAM 사용 설명서의 AWS 서비스 에서 작업하는 IAM 를 AWS STS참조하세요.

조건: 에서 임시 보안 자격 증명을 사용하는 경우 보안 토큰을 포함해야 AWS STS합니다.

유형: 문자열

필수 항목 여부: 조건부

X-Amz-Signature

서명할 문자열과 파생된 서명 키에서 계산된 16진수로 인코딩된 서명을 지정합니다.

조건: HTTP 권한 부여 헤더 대신 쿼리 문자열에 인증 정보를 포함할 때 이 파라미터를 지정합니다.

유형: 문자열

필수 항목 여부: 조건부

X-Amz-SignedHeaders

표준 요청의 일부로 포함된 모든 HTTP 헤더를 지정합니다. 서명된 헤더 지정에 대한 자세한 내용은 IAM 사용 설명서의 서명된 AWS API 요청 생성을 참조하세요.

조건: HTTP 권한 부여 헤더 대신 쿼리 문자열에 인증 정보를 포함할 때 이 파라미터를 지정합니다.

유형: 문자열

필수 항목 여부: 조건부

문서 이력

변경 사항	설명	날짜
설명서 전용 업데이트	기본 할당량과 시스템 제한을 분리하도록 할당량 주제를 업데이트했습니다.	2024년 10월 22일
Amazon Timestream은 이제 쿼리 인사이트를 지원합니다.	이제 Timestream에는 쿼리를 최적화하고, 성능을 개선하고,	2024년 10월 22일

	비용을 절감하는 데 도움이 되는 쿼리 인사이트 기능에 대한 지원이 포함되어 있습니다.	
기존 정책에 대한 Amazon Timestream for InfluxDB 업데이트.	Amazon Timestream for InfluxDB가 라우팅 테이블을 설명하기 위해 기존 `AmazonTimestreamInfluxDBFullAccess` 관리형 정책에 `ec2:DescribeRouteTables` 작업을 추가했습니다.	2024년 10월 8일
`AmazonTimestreamReadOnlyAccess` - 기존 정책 업데이트	에 대한 Timestream이 AWS 계정 설정을 설명하기 위한 `AmazonTimestreamReadOnlyAccess` 관리형 정책에 `DescribeAccountSettings` 권한을 추가 LiveAnalytics 했습니다.	2024년 6월 3일
LiveAnalytics 이제 에 대한 Amazon Timestream 이 Timestream Compute Units(TCUs)를 지원합니다.	LiveAnalytics 이제 Amazon Timestream에 Timestream Compute Units(TCUs)에 대한 지원이 포함되어 쿼리 요구 사항에 할당된 컴퓨팅 용량을 측정할 수 있습니다.	2024년 4월 29일

새 정책 추가	Amazon Timestream for InfluxDB에는 두 가지 새로운 정책이 추가되었습니다. 하나는 서비스가 계정의 네트워크 인터페이스와 보안 그룹을 관리할 수 있도록 허용하는 정책입니다. 자세한 내용은 섹션을 참조하세요AmazonTimestreamInfluxDBServiceRolePolicy. Amazon Timestream InfluxDB 인스턴스를 생성, 업데이트, 삭제 및 나열하고 파라미터 그룹을 생성 및 나열할 수 있는 전체 관리 액세스 권한을 제공하는 또 다른 기능입니다. 자세한 내용은 섹션을 참조하세요AmazonTimestreamInfluxDBFullAccess.	2024년 3월 14일
이제 Amazon Timestream for InfluxDB를 일반적으로 사용할 수 있습니다.	이 설명서에서는 Amazon Timestream for InfluxDB 의 초기 릴리스를 다룹니다.	2024년 3월 14일
Amazon Timestream for LiveAnalytics Query 이벤트는 에서 사용할 수 있습니다. AWS CloudTrail	Amazon Timestream은 LiveAnalytics 이제 쿼리 API 데이터 이벤트를 에 게시합니다 AWS CloudTrail. 고객은 AWS 계정에서 수행된 모든 쿼리 API 요청을 감사하고 어떤 IAM 사용자/역할이 요청을 했는지, 요청이 수행된 시기, 쿼리된 데이터베이스 및 테이블, 요청의 쿼리 ID와 같은 정보를 볼 수 있습니다.	2023년 9월 12일

에 대한 Amazon Timestream LiveAnalytics UNLOAD	Amazon Timestream은 LiveAnalytics 이제 UNLOAD에서 쿼리 결과를 S3로 내보낼 수 있도록 지원합니다.	2023년 5월 12일
기존 정책에 대한 LiveAnalytics 업데이트를 위한 Amazon Timestream입니다.	관리형 정책에 추가된 배치 로드 권한입니다.	2023년 2월 24일
LiveAnalytics 배치 로드에 대한 Amazon Timestream.	Amazon Timestream은 LiveAnalytics 이제 배치 로드 기능을 지원합니다.	2023년 2월 24일
Amazon Timestream은 LiveAnalytics 이제 를 지원합니다 AWS Backup.	Amazon Timestream은 LiveAnalytics 이제 를 지원합니다 AWS Backup.	2022년 12월 14일
AWS 관리형 정책에 대한 LiveAnalytics 업데이트를 위한 Amazon Timestream	기존 AWS 관리형 정책에 대한 업데이트를 LiveAnalytics포함하여 관리형 정책 및 Amazon Timestream for 에 대한 새로운 정보입니다.	2021년 11월 29일
용 Amazon Timestream은 예약된 쿼리 LiveAnalytics 를 지원합니다.	LiveAnalytics 이제 용 Amazon Timestream은 일정에 따라 사용자를 대신하여 쿼리 실행을 지원합니다.	2021년 11월 29일
용 Amazon Timestream은 마그네틱 스토어를 LiveAnalytics 지원합니다.	LiveAnalytics 이제 용 Amazon Timestream은 테이블 쓰기에 마그네틱 스토리지 사용을 지원합니다.	2021년 11월 29일
LiveAnalytics 다중 측정 레코드에 대한 Amazon Timestream입니다.	LiveAnalytics 이제 용 Amazon Timestream은 시계열 데이터를 저장하기 위한 더 컴팩트한 형식을 지원합니다.	2021년 11월 29일

AWS 관리형 정책에 대한 LiveAnalytics 업데이트를 위한 Amazon Timestream	기존 AWS 관리형 정책에 대한 업데이트를 LiveAnalytics포함하여 관리형 정책 및 Amazon Timestream for 에 대한 새로운 정보입니다.	2021년 5월 24일
LiveAnalytics 이제 유럽(프랑크푸르트) 리전에서 에 대한 Amazon Timestream을 사용할 수 있습니다.	LiveAnalytics 이제 유럽(프랑크푸르트) 리전()에서 용 Amazon Timestream을 사용할 수 있습니다`eu-central-1` .	2021년 4월 23일
에 대한 Amazon Timestream LiveAnalytics 은 이제 VPC 엔드포인트()를 지원합니다AWS PrivateLink.	LiveAnalytics 이제 용 Amazon Timestream은 VPC 엔드포인트() 사용을 지원합니다AWS PrivateLink.	2021년 3월 23일
Amazon Timestream은 이제 크로스 테이블 쿼리를 지원합니다.	에 Amazon Timestream을 사용하여 테이블 간 쿼리 LiveAnalytics 를 실행할 수 있습니다.	2021년 2월 10일
Amazon Timestream은 LiveAnalytics 이제 향상된 쿼리 실행 통계를 지원합니다.	LiveAnalytics 이제 용 Amazon Timestream은 스캔된 데이터의 양과 같은 향상된 쿼리 실행 통계를 지원합니다.	2021년 2월 10일
Amazon Timestream은 LiveAnalytics 이제 고급 시계열 함수를 지원합니다.	에 Amazon Timestream LiveAnalytics 을 사용하여 파생, 적분 및 상관 관계와 같은 고급 시계열 함수를 사용하여 SQL 쿼리를 실행할 수 있습니다.	2021년 2월 10일
에 대한 Amazon Timestream LiveAnalytics 은 이제 HIPAA, ISO및 PCI 규정을 준수합니다.	이제 HIPAA, ISO및 PCI규정 준수 인프라가 필요한 워크로드에 Amazon Timestream LiveAnalytics for 를 사용할 수 있습니다.	2021년 1월 27일

Amazon Timestream은 LiveAnalytics 이제 오픈 소스 텔레그래프 및 그래파나를 지원합니다.	이제 Amazon Timestream for 에서 오픈 소스 플러그인 기반 서버 에이전트인 Telegraf를 사용하여 지표를 수집하고 보고할 수 있으며, 데이터베이스의 오픈 소스 분석 및 모니터링 플랫폼인 Grafana를 사용할 수 있습니다 LiveAnalytics.	2020년 11월 25일
이제 에 대한 Amazon Timestream LiveAnalytics 을 일반적으로 사용할 수 있습니다.	이 설명서에서는 용 Amazon Timestream의 초기 릴리스를 다룹니다 LiveAnalytics.	2020년 9월 30일

InfluxDB의 Timestream이란 무엇입니까?

Amazon Timestream for InfluxDB는 관리형 시계열 데이터베이스 엔진으로, 애플리케이션 개발자와 DevOps 팀이 오픈 소스를 사용하여 실시간 시계열 애플리케이션에 AWS 대해 에서 InfluxDB 데이터베이스를 쉽게 실행할 수 있습니다APIs. Amazon Timestream for InfluxDB 사용하면 한 자리 수 밀리초 쿼리 응답 시간으로 쿼리에 응답할 수 있는 시계열 워크로드를 쉽게 설정, 운영 및 확장할 수 있습니다.

Amazon Timestream for InfluxDB를 사용하면 2.x 브랜치에서 InfluxDB의 친숙한 오픈 소스 버전의 기능에 액세스할 수 있습니다. 즉, 현재 기존 InfluxDB 오픈 소스 데이터베이스에서 이미 사용 중인 코드, 애플리케이션 및 도구가 Amazon Timestream for InfluxDB 와 원활하게 작동해야 합니다. Amazon Timestream for InfluxDB는 데이터베이스를 자동으로 백업하고 데이터베이스 소프트웨어를 최신 버전으로 최신 상태로 유지할 수 있습니다. 또한 Amazon Timestream for InfluxDB를 사용하면 복제를 쉽게 사용하여 데이터베이스 가용성을 높이고 데이터 내구성을 개선할 수 있습니다. 모든 AWS 서비스와 마찬가지로 선결제 투자는 필요하지 않으며 사용하는 리소스에 대해서만 비용을 지불합니다.

DB 인스턴스

DB 인스턴스는 클라우드에서 실행하는 격리된 데이터베이스 환경입니다. 이는 Amazon Timestream for InfluxDB 의 기본 구성 요소입니다. DB 인스턴스에는 사용자가 생성한 여러 데이터베이스(또는 InfluxDb 2.x 데이터베이스의 경우 조직 및 버킷)가 포함될 수 있으며, 독립형 자체 관리형 InfluxDB 인스턴스에 액세스하는 데 사용할 수 있는 것과 동일한 클라이언트 도구 및 애플리케이션을 사용하여 액세스할 수 있습니다. DB 인스턴스는 명령줄 도구, Amazon Timestream InfluxDB API 작업 또는 를 AWS 사용하여 간단하게 생성하고 수정할 수 있습니다 AWS Management Console.

> **Note**
>
> Amazon Timestream for InfluxDB는 Influx API 작업 및 Influx UI를 사용하여 데이터베이스에 대한 액세스를 지원합니다. Amazon Timestream for InfluxDB는 직접 호스트 액세스를 허용하지 않습니다.

최대 40개의 Amazon Timestream for InfluxDB 인스턴스를 보유할 수 있습니다.

각 DB 인스턴스에는 DB 인스턴스 이름이 있습니다. 이 고객 제공 이름은 Amazon Timestream for InfluxDB API 및 AWS CLI 명령과 상호 작용할 때 DB 인스턴스를 고유하게 식별합니다. DB 인스턴스 이름은 리전의 AWS 해당 고객에 대해 고유해야 합니다.

DB 인스턴스 이름은 InfluxDB용 Timestream 에서 인스턴스에 할당된 DNS 호스트 이름의 일부를 구성합니다. 예를 들어 influxdb1을 DB 인스턴스 이름으로 지정하는 경우 Timestream은 인스턴스에 DNS 엔드포인트를 자동으로 할당합니다. 예제 엔드포인트는 이며`influxdb1-3ksj4dla5nfjhi.us-east-1.timestream-influxdb.amazonaws.com`, 여기서 `influxdb1`는 인스턴스 이름입니다.

예제 엔드포인트 에서 `influxdb1-3ksj4dla5nfjhi.us-east-1.timestream-influxdb.amazonaws.com`문자열`3ksj4dla5nfjhi`은 에서 생성된 고유한 계정 식별자입니다 AWS. 예제`3ksj4dla5nfjhi`의 식별자는 특정 리전의 지정된 계정에 대해 변경되지 않습니다. 따라서 이 계정에서 생성한 모든 DB 인스턴스는 동일한 고정 식별자를 공유합니다. 고정 식별자의 다음 기능을 고려해 보세요.

- 현재 InfluxDB의 Timestream은 DB 인스턴스 이름 변경을 지원하지 않습니다.
- 동일한 DB 인스턴스 식별자로 DB 인스턴스를 삭제하고 다시 만들면 엔드포인트는 동일합니다.
- 동일한 계정을 사용하여 다른 리전에 DB 인스턴스를 만들 경우, `influxdb2.4a3j5du5ks7md2.us-west-1.timestream-influxdb.amazonaws.com`처럼 리전이 다르기 때문에 내부적으로 생성되는 식별자도 달라집니다.

각 DB 인스턴스는 InfluxDB 데이터베이스 엔진에 대해 하나의 Timestream만 지원합니다.

DB 인스턴스를 생성할 때 InfluxDB는 조직 이름을 지정해야 합니다. DB 인스턴스는 여러 조직과 각 조직과 연결된 여러 버킷을 호스팅할 수 있습니다.

Amazon Timestream for InfluxDB를 사용하면 생성 프로세스의 일부로 DB 인스턴스의 마스터 사용자 계정과 암호를 생성할 수 있습니다. 이 마스터 사용자는 조직, 버킷을 생성하고 데이터에 대한 읽기, 쓰기, 삭제 및 업서트 작업을 수행할 수 있는 권한이 있습니다. 또한 InfluxUI에 액세스하여 사용자 토큰을 검색할 수 있습니다. 첫 번째 로그인입니다. 여기에서 모든 액세스 토큰도 관리할 수 있습니다. DB 인스턴스를 생성할 때 마스터 사용자 암호를 설정해야 하지만 Influx , Influx CLI또는 APIInfluxUI 를 사용하여 언제든지 변경할 수 있습니다.

DB 인스턴스 클래스

DB 인스턴스 클래스는 Amazon Timestream `fi UbfkyxDB` DB 인스턴스의 계산 및 메모리 용량을 결정합니다. 필요한 DB 인스턴스 클래스는 DB 인스턴스의 처리력 및 메모리 요구 사항에 따라 다릅니다.

DB 인스턴스 클래스는 DB 인스턴스 클래스 유형과 크기로 구성됩니다. 예를 들어 `db.influx`는 InfluxDb 워크로드 실행과 관련된 고성능 메모리 요구 사항에 적합한 메모리 최적화 DB 인스턴스 클래스 유형입니다. `db.influx` 인스턴스 클래스 유형 내에서 `db.influx.2xlarge`는 DB 인스턴스 클래스입니다. 이 클래스의 크기는 2xlarge 입니다.

인스턴스 클래스 요금에 대한 자세한 내용은 Amazon Timestream for InfluxDB 요금 섹션을 참조하세요.

DB 인스턴스 클래스 유형

Amazon Timestream for InfluxDB는 InfluxDB 사용 사례에 최적화된 다음 사용 사례에 대해 DB 인스턴스 클래스를 지원합니다.

- **`db.influx`**- 이러한 인스턴스 클래스는 오픈 소스 InfluxDB 데이터베이스에서 메모리 집약적 워크로드를 실행하는 데 적합합니다.

DB 인스턴스 클래스의 하드웨어 사양

다음 용어는 DB 인스턴스 클래스의 하드웨어 사양을 설명합니다.

- vCPU

  가상 중앙 처리 단위 수(CPUs). 가상CPU은 DB 인스턴스 클래스를 비교하는 데 사용할 수 있는 용량 단위입니다.
- 메모리(GiB)

  DB 인스턴스RAM에 할당된 기가바이트 단위의 . 메모리와 v 간에는 일관된 비율이 있는 경우가 많습니다CPU. 예를 들어 r7g 인스턴스 클래스와 유사한 메모리 대 vCPU 비율을 가진 EC2 db.influx 인스턴스 클래스를 예로 들어 보겠습니다.
- Influx 최적화

  DB 인스턴스는 최적화된 구성 스택을 사용하며 I/O에 대한 전용 용량을 추가로 제공합니다. 이 최적화는 I/O와 인스턴스 간의 경합을 최소화하여 최상의 성능을 제공합니다.
- 네트워크 대역폭

  다른 DB 인스턴스 클래스 대비 네트워크 속도입니다. 다음 표에서는 Amazon Timestream for InfluxDB 인스턴스 클래스에 대한 하드웨어 세부 정보를 찾을 수 있습니다.

인스턴스 클래스	vCPU	메모리(GiB)	스토리지 유형	네트워크 대역폭 (Gbps)
db.influx.medium	1	8	유입 IOPS 포함	10

인스턴스 클래스	vCPU	메모리(GiB)	스토리지 유형	네트워크 대역폭 (Gbps)
db.influx.large	2	16	유입 IOPS 포함	10
db.influx.xlarge	4	32	유입 IOPS 포함	10
db.influx.2xlarge	8	64	유입 IOPS 포함	10
db.influx.4xlarge	16	128	유입 IOPS 포함	10
db.influx.8xlarge	32	256	유입 IOPS 포함	12
db.influx .12xlarge	48	384	유입 IOPS 포함	20
db.influx .16xlarge	64	512	유입 IOPS 포함	25

InfluxDB 인스턴스 스토리지

Amazon Timestream for InfluxDB용 DB 인스턴스는 데이터베이스 및 로그 스토리지에 Influx IOPS 포함 볼륨을 사용합니다.

경우에 따라 데이터베이스 워크로드가 프로비저닝한 의 100%를 달성하지 못할 수 IOPS 있습니다. 자세한 내용은 스토리지 성능에 영향을 주는 요인을 참조하십시오. Timestream for InfluxDB 스토리지 요금에 대한 자세한 내용은 Amazon Timestream 요금 섹션을 참조하세요.

Amazon Timestream for InfluxDB 스토리지 유형

Amazon Timestream for InfluxDB는 한 가지 스토리지 유형인 Influx IOPS 포함을 지원합니다. 최대 16테비바이트(TiB)의 스토리지로 InfluxDB 인스턴스에 대한 Timestream을 생성할 수 있습니다.

다음은 사용 가능한 스토리지 유형에 대한 간략한 설명입니다.

- Influx IO 포함 스토리지: 스토리지 성능은 초당 I/O 작업(IOPS)과 스토리지 볼륨이 읽기 및 쓰기(스토리지 처리량)를 수행하는 속도의 조합입니다. Influx IOPS 포함 스토리지 볼륨에서 Amazon Timestream for InfluxDB는 다양한 유형의 워크로드에 필요한 최적의 IOPS 처리량과 처리량으로 사전 구성된 3개의 스토리지 계층을 제공합니다.

InfluxDB 인스턴스 크기 조정

InfluxDB 인스턴스용 Timestream의 최적 구성은 수집 속도, 배치 크기, 시계열 카디널리티, 동시 쿼리 및 쿼리 유형을 포함하는 많은 요인에 따라 달라집니다. 크기 조정 권장 사항을 제공하기 위해 다음과 같은 특성을 가진 모범적인 워크로드에 중점을 두고 있습니다.

- 데이터는 데이터 센터에서 시스템, CPU, 메모리, 디스크, IO 등을 수집하는 Telegraf 에이전트 플릿에서 수집하고 작성합니다.

  각 쓰기 요청에는 5,000개의 줄이 포함됩니다.
- 시스템에서 실행되는 쿼리 유형은 “중등도 복잡성” 쿼리로 분류됩니다. 이 쿼리 범주는 다음과 같은 특성을 제공합니다.
  - 함수가 여러 개 있고 정규식이 하나 또는 두 개 있음
  - 절별로 그룹화하거나 여러 주의 시간 범위를 샘플링할 수도 있습니다.
  - 일반적으로 실행하는 데 수백 밀리초에서 수만 밀리초가 걸립니다.
  - CPU 는 주로 쿼리 성능을 선호합니다.

인스턴스 클래스	스토리지 유형	쓰기(초당 줄 수)	읽기(초당 쿼리 수)
db.influx.large	Influx IO 포함 3K	~50,000	<10
db.influx.2xlarge	Influx IO 포함 3K	~150,000	<25
db.influx.4xlarge	Influx IO 포함 3K	~200,000	~25
db.influx.4xlarge	Influx IO 포함 12K	~250,000	~35
db.influx.8xlarge	Influx IO 포함 12K	~500,000	~50
db.influx.12xlarge	Influx IO 포함 12K	<750,000	<100

AWS 리전 및 가용 영역

Amazon 클라우드 컴퓨팅 리소스는 세계 각지의 여러 곳에서 호스팅됩니다. 이러한 위치는 AWS 리전과 가용 영역으로 구성됩니다. 각 AWS 리전은 별도의 지리적 영역입니다. 각 AWS 리전에는 가용 영역이라고 하는 격리된 위치가 여러 개 있습니다.

> **Note**
>
> AWS 리전의 가용 영역을 찾는 방법에 대한 자세한 내용은 Amazon EC2 사용 설명서의 리전 및 영역을 참조하세요.

Amazon Timestream for InfluxDB를 사용하면 DB 인스턴스 및 데이터와 같은 리소스를 여러 위치에 배치할 수 있습니다.

Amazon은 가용성이 높은 state-of-the-art를 운영합니다. 드물기는 하지만 동일한 위치에 있는 DB 인스턴스의 가용성에 영향을 미치는 장애가 발생할 수도 있습니다. 그런 장애의 영향을 받는 하나의 위치에서 모든 DB 인스턴스를 호스팅하는 경우에는 모든 DB 인스턴스가 사용이 불가능해질 수 있습니다.

각 AWS 리전은 완전히 독립적이라는 점을 기억하는 것이 중요합니다. 시작하는 모든 Amazon Timestream for InfluxDB 활동(예: 데이터베이스 인스턴스 생성 또는 사용 가능한 데이터베이스 인스턴스 나열)은 현재 기본 AWS 리전에서만 실행됩니다. 기본 AWS 리전은 콘솔에서 변경하거나 `AWS_DEFAULT_REGION` 환경 변수를 설정하여 변경할 수 있습니다. 또는 AWS Command Line

Interface ()와 함께 `--region` 파라미터를 사용하여 재정의할 수 있습니다AWS CLI. 자세한 내용은 구성 AWS Command Line Interface, 특히 환경 변수 및 명령줄 옵션에 대한 섹션을 참조하세요.

특정 AWS 리전에서 Amazon Timestream for InfluxDB DB 인스턴스를 생성하거나 작업하려면 해당 리전 서비스 엔드포인트를 사용합니다.

AWS 리전 가용성

Amazon Timestream for InfluxDB를 현재 사용할 수 있는 AWS 리전과 각 리전의 엔드포인트에 대한 자세한 내용은 Amazon Timestream 엔드포인트 및 할당량 섹션을 참조하세요.

AWS 리전 설계

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

리소스를 볼 때 지정한 AWS 리전에 연결된 리소스만 표시됩니다. 이는 AWS 리전이 서로 격리되고 AWS 리전 간에 리소스를 자동으로 복제하지 않기 때문입니다.

AWS 가용 영역

DB 인스턴스를 생성할 때 Amazon Timestream for InfluxDB는 서브넷 구성에 따라 무작위로 하나를 선택합니다. 가용 영역은 AWS 리전 코드와 문자 식별자(예:)로 표시됩니다`us-east-1a`.

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

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

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

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

다중 AZ DB 배포 에서 기본 및 보조 DB 인스턴스의 가용 영역을 선택할 수 없습니다. Amazon Timesteram for InfluxDB가 무작위로 선택합니다. 다중 AZ 배포에 대한 자세한 내용은 섹션을 참조하세요다중 AZ 배포 구성 및 관리.

Amazon Timestream for InfluxDB에 대한 DB 인스턴스 결제

Amazon Timestream for InfluxDB 인스턴스는 다음 구성 요소를 기준으로 요금이 청구됩니다.

- DB 인스턴스 시간(시간당) - db.influx.large와 같은 DB 인스턴스의 DB 인스턴스 클래스를 기반으로 합니다. 요금은 시간 단위로 고시되지만, 청구서는 초 단위로 계산되고 시간을 10진수 형식으로 표시합니다. Amazon Timestream for InfluxDB 사용량은 1초 단위로 청구되며 최소 10분이 소요됩니다. 자세한 내용은 DB 인스턴스 클래스DB 인스턴스 클래스를 참조하세요.
- 스토리지(GiB당 월별) - DB 인스턴스에 프로비저닝한 스토리지 용량입니다. 자세한 내용은 InfluxDB 인스턴스 스토리지 단원을 참조하십시오.
- 데이터 전송(GB당) - 인터넷 및 기타 AWS 리전에서 또는 인터넷 및 기타 리전으로 DB 인스턴스 내외로 데이터를 전송합니다.

Amazon Timestream for InfluxDB 요금 정보는 Amazon Timestream for InfluxDB 요금 페이지를 참조하세요.

InfluxDB용 Amazon Timestream 설정

Amazon Timestream for InfluxDB를 처음 사용하기 전에 다음 작업을 완료합니다.

이미 AWS 계정이 있는 경우 Amazon Timestream for InfluxDB 요구 사항을 파악하고 및 Amazon Timestream for InfluxDB IAM VPC Timestream for InfluxDB 시작하기시작하기 에 대한 기본값을 사용하는 것이 좋습니다.

AWS 계정 가입

AWS 계정이 없는 경우 다음 단계를 완료하여 계정을 생성합니다.

계정에 가입 AWS 하려면

- AWS 로그인 페이지로 이동합니다.
- 새 계정 생성을 선택하고 지침을 따릅니다.

> **Note**
>
> 등록 절차 중 전화를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

AWS 계정에 가입하면 AWS 계정 루트 사용자가 생성됩니다. 루트 사용자는 계정의 모든 AWS 서비스 및 리소스에 액세스할 수 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 태스크를 수행하는 것입니다.

AWS 는 가입 프로세스가 완료된 후 확인 이메일을 보냅니다. 언제든지 https://aws.amazon.com/로 이동하여 내 계정 을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

사용자 관리

관리 사용자 생성

관리 사용자 생성

AWS 계정에 가입한 후 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

AWS 계정 루트 사용자 보안

루트 사용자를 선택하고 계정 이메일 주소를 입력하여 AWS 관리 콘솔에 AWS 계정 소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다. 루트 사용자를 사용하여 로그인하는 방법에 대한 자세한 내용은 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요. AWS

루트 사용자에 대해 다중 인증(MFA)을 켭니다. 지침은 IAM 사용 설명서의 AWS 계정 루트 사용자(콘솔)에 대한 가상 MFA 디바이스 활성화를 참조하세요.

프로그래밍 방식 액세스 권한 부여

사용자는 AWS 외부에서 와 상호 작용하려는 경우 프로그래밍 방식으로 액세스해야 합니다 AWS Management Console. 프로그래밍 방식으로 액세스를 부여하는 방법은 AWS에 액세스하는 사용자 유형에 따라 다릅니다.

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	By
인력 자격 증명(자IAM격 증명 센터에서 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI AWS SDKs, 또는 에 대한 프로그래밍 요청에 서명합니다 AWS APIs.	사용하려는 인터페이스에 대한 지침을 따릅니다.* 의 경우 AWS CLI를 참조하세요. Identity Center를 AWS CLI 사용하도록 AWS IAM 구성

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	By
		의 AWS 명령줄 인터페이스 사용 설명서 .* AWS SDKs, 도구 및 는 섹션을 AWS APIs참조하세요. IAM Identity Center 인증 의 AWS SDKs 및 도구 참조 가이드 .
IAM	임시 자격 증명을 사용하여 AWS CLI, SDKs및 에 대한 프로그래밍 요청에 서명합니다 APIs.	IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용의 지침을 따릅니다.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	By
IAM	(권장하지 않음) 장기 보안 인증 정보를 사용하여 CLI, SDKs 및 에 AWS 대한 프로그래밍 요청에 서명합니다APIs.	사용하려는 인터페이스에 대한 지침에 따라 를 AWS CLI참조하세요. IAM 사용자 자격 증명을 사용하여 인증 의 AWS 명령줄 인터페이스 사용 설명서 . 및 도구에 대한 자세한 AWS SDKs 내용은 섹션을 참조하세요. 장기 자격 증명을 사용하여 인증 의 AWS SDKs 및 도구 참조 가이드 .의 경우 AWS APIs를 참조하세요. IAM 사용자를 위한 액세스 키 관리 의 IAM 사용 설명서

요구 사항 결정

Amazon Timestream for Influx의 기본 구성 요소는 DB 인스턴스입니다. DB 인스턴스에서는 버킷을 생성합니다. DB 인스턴스는 엔드포인트라고 하는 네트워크 주소를 할당합니다. 애플리케이션은 이 엔드포인트를 사용하여 DB 인스턴스에 연결합니다. 또한 브라우저에서 동일한 엔드포인트를 사용하여 InfluxUI에 액세스할 수 있습니다. DB 인스턴스를 생성할 때 스토리지, 메모리, 데이터베이스 엔진 및

버전, 네트워크 구성 및 보안과 같은 세부 정보를 지정합니다. 보안 그룹을 통해 DB 인스턴스에 대한 네트워크 액세스를 제어할 수 있습니다.

DB 인스턴스와 보안 그룹을 생성하기 전에 DB 인스턴스 및 네트워크 필요를 알아야 합니다. 고려해야 할 몇 가지 중요 사항은 다음과 같습니다:

- 리소스 요구 사항 - 애플리케이션 또는 서비스의 메모리 및 프로세서 요구 사항은 무엇입니까? 이러한 설정을 사용하면 어떤 DB 인스턴스 클래스를 사용할지를 결정하는 데 도움이 됩니다. DB 인스턴스 클래스에 대한 사양은 DB 인스턴스 클래스 섹션을 참조하세요.
- VPC 및 보안 그룹 - DB 인스턴스가 가상 프라이빗 클라우드(VPC)에 있을 가능성이 높습니다. DB 인스턴스에 연결하려면 보안 그룹 규칙을 설정해야 합니다. 이러한 규칙은 VPC 사용하는 유형과 사용 방법에 따라 다르게 설정됩니다. 예를 들어 기본 VPC 또는 사용자 정의 를 사용할 수 있습니다 VPC.

  다음 목록은 각 VPC 옵션의 규칙을 설명합니다.

  - 기본VPC값 - AWS 계정에 현재 AWS 리전VPC의 기본값이 있는 경우 DB 인스턴스를 지원하도록 VPC 구성됩니다. DB 인스턴스를 생성할 VPC 때 기본값을 지정하는 경우 애플리케이션 또는 서비스에서 Amazon Timestream for InfluxDB DB 인스턴스로의 연결을 승인하는 VPC 보안 그룹을 생성해야 합니다. VPC 콘솔 또는 AWS CLI의 보안 그룹 옵션을 사용하여 VPC 보안 그룹을 생성합니다. 자세한 내용은 3단계: VPC 보안 그룹 생성 단원을 참조하세요.
- 사용자 정의 VPC - DB 인스턴스를 생성할 VPC 때 사용자 정의를 지정하려면 다음 사항에 유의하세요.
  - 애플리케이션 또는 서비스에서 Amazon Timestream for InfluxDB DB 인스턴스로의 연결을 승인하는 VPC 보안 그룹을 생성해야 합니다. VPC 콘솔 또는 AWS CLI의 보안 그룹 옵션을 사용하여 VPC 보안 그룹을 생성합니다. 자세한 내용은 3단계: VPC 보안 그룹 생성 단원을 참조하세요.
  - 는 DB 인스턴스를 호스팅하기 위해 각각 별도의 가용 영역에 있는 서브넷이 두 개 이상 있는 등 특정 요구 사항을 충족해야 VPC 합니다. 자세한 내용은 Amazon VPC VPCs 및 Amazon Timestream for InfluxDB를 참조하세요.
- 고가용성 — 장애 조치 지원이 필요합니까? Amazon Timestream for InfluxDB 에서 다중 AZ 배포는 장애 조치 지원을 위해 다른 가용 영역에 기본 DB 인스턴스와 보조 대기 DB 인스턴스를 생성합니다. 고가용성을 유지하기 위해 프로덕션 워크로드에는 다중 AZ 배포를 권장합니다. 개발 및 테스트 목적으로는 비 다중 AZ 배포를 사용할 수 있습니다. 자세한 내용은 다중 AZ DB 인스턴스 배포 단원을 참조하십시오.
- IAM 정책 - AWS 계정에 Amazon Timestream for InfluxDB 작업을 수행하는 데 필요한 권한을 부여하는 정책이 있습니까? IAM 자격 증명을 AWS 사용하여 에 연결하는 경우 IAM 계정에는 Amazon Timestream for InfluxDB 제어 영역 작업을 수행하는 데 필요한 권한을 부여하는 IAM 정책이 있어야

합니다. 자세한 내용은 Amazon Timestream for InfluxDB의 자격 증명 및 액세스 관리 단원을 참조하십시오.

- 오픈 포트 - 데이터베이스가 수신 대기하는 TCP/IP 포트는 무엇입니까? 일부 기업에서는 방화벽이 데이터베이스 엔진의 기본 포트 연결을 차단합니다. InfluxDB용 Timestream의 기본값은 8086입니다.
- AWS 리전 - 데이터베이스가 어느 AWS 리전에 들어가길 원합니까? 애플리케이션이나 웹 서비스에 가깝게 데이터베이스를 구성하면 네트워크 지연 시간을 줄일 수 있습니다. 자세한 내용은 AWS 리전 및 가용 영역 단원을 참조하십시오.
- DB 디스크 하위 시스템 - 스토리지 요구 사항은 무엇입니까? Amazon Timestream for InfluxDB는 Influx IOPS 포함 스토리지 유형에 대한 세 가지 구성을 제공합니다.
  - Influx Io 포함 3kIOPS(SSD)
  - Influx Io 포함 12kIOPS(SSD)
  - Influx Io 포함 25kIOPS(SSD)

  Amazon Timestream for InfluxDB 스토리지에 대한 자세한 내용은 Amazon Timestream for InfluxDB DB 인스턴스 스토리지를 참조하세요. 보안 그룹과 DB 인스턴스 생성에 필요한 정보를 확인하였으면 다음 단계로 진행합니다.

보안 그룹을 생성VPC하여 의 DB 인스턴스에 대한 액세스 제공

VPC 보안 그룹은 의 DB 인스턴스에 대한 액세스를 제공합니다VPC. 이들은 연결된 DB 인스턴스에 대한 방화벽 역할을 하여 DB 인스턴스 수준에서 인바운드 트래픽과 아웃바운드 트래픽을 모두 제어합니다. 기본적으로 DB 인스턴스는 DB 인스턴스를 보호하는 방화벽 및 기본 보안 그룹과 함께 생성됩니다.

DB 인스턴스에 연결하려면 먼저 연결하는 데 사용할 수 있는 규칙을 보안 그룹에 추가해야 합니다. 네트워크 및 구성 정보를 사용하여 DB 인스턴스에 액세스하는 데 사용할 수 있는 규칙을 생성합니다.

예를 들어 의 DB 인스턴스에 있는 데이터베이스에 액세스하는 애플리케이션이 있다고 가정해 보겠습니다VPC. 이 경우 애플리케이션이 데이터베이스에 액세스하는 데 사용하는 포트 범위와 IP 주소를 지정하는 사용자 지정 TCP 규칙을 추가해야 합니다. Amazon EC2 인스턴스에 애플리케이션이 있는 경우 Amazon EC2 인스턴스에 대해 설정한 보안 그룹을 사용할 수 있습니다.

VPC 액세스를 위한 보안 그룹 생성

VPC 보안 그룹을 생성하려면 에 로그인 AWS Management Console 하고 를 선택합니다VPC.

> **Note**
>
> Amazon Timesteam for InfluxDB VPC 콘솔이 아닌 콘솔에 있는지 확인합니다.

- 의 오른쪽 상단에서 VPC 보안 그룹 및 DB 인스턴스를 생성할 AWS 리전을 AWS Management Console선택합니다. 해당 AWS 리전의 Amazon VPC 리소스 목록에 하나 이상의 서브넷VPC이 표시됩니다. 그렇지 않으면 해당 AWS 리전VPC에 기본값이 없습니다.
- 탐색 창에서 보안 그룹을 선택합니다.
- 보안 그룹 생성을 선택합니다.
- 보안 그룹 페이지의 기본 세부 정보 섹션에 보안 그룹 이름과 설명을 입력합니다. 에서 DB 인스턴스 VPC를 생성할 를 VPC선택합니다.
- [인바운드 규칙(Inbound rules)]에서 [규칙 추가(Add rule)]를 선택합니다.
  - 유형 에서 사용자 지정 을 TCP선택합니다.
  - 소스 에서 보안 그룹 이름을 선택하거나 DB 인스턴스에 액세스하는 IP 주소 범위(CIDR 값)를 입력합니다. 내 IP를 선택하면 브라우저에서 감지된 IP 주소에서 DB 인스턴스에 액세스할 수 있습니다.

  소스에서 보안 그룹 이름을 선택하거나 DB 인스턴스에 액세스하는 IP 주소 범위(CIDR 값)를 입력합니다. 내 IP를 선택하면 브라우저에서 감지된 IP 주소에서 DB 인스턴스에 액세스할 수 있습니다.
- (선택 사항) [아웃바운드 규칙(Outbound rules)]에서 아웃바운드 트래픽에 대한 규칙을 추가합니다. 기본적으로 모든 아웃바운드 트래픽이 허용됩니다.
- 보안 그룹 생성을 선택합니다.

이 VPC 보안 그룹을 생성할 때 DB 인스턴스의 보안 그룹으로 사용할 수 있습니다.

> **Note**
>
> 기본 VPC를 사용하는 경우 모든 VPC의 서브넷에 걸쳐 있는 기본 서브넷 그룹이 생성됩니다. DB 인스턴스를 생성할 때 기본 eiifccntfVPC를 선택하고 DB 서브넷 그룹의 기본값을 선택할 수 있습니다.

설정 요구 사항을 완료한 후에는 요구 사항과 보안 그룹을 사용하여 DB 인스턴스를 생성할 수 있습니다. 그러려면 DB 인스턴스 생성의 지침을 따르세요.

Timestream for InfluxDB 시작하기

다음 예제에서는 Amazon Timestream for InfluxDB Service를 사용하여 DB 인스턴스를 생성하고 연결하는 방법을 확인할 수 있습니다.

> **Note**
>
> DB 인스턴스를 생성하거나 DB 인스턴스에 연결하려면 먼저 InfluxDB용 Amazon Timestream 설정 섹션의 태스크를 완료해야 합니다.

주제

- InfluxDB 인스턴스용 Timestream 생성 및 연결
- InfluxDB 인스턴스에 대한 새 운영자 토큰 생성

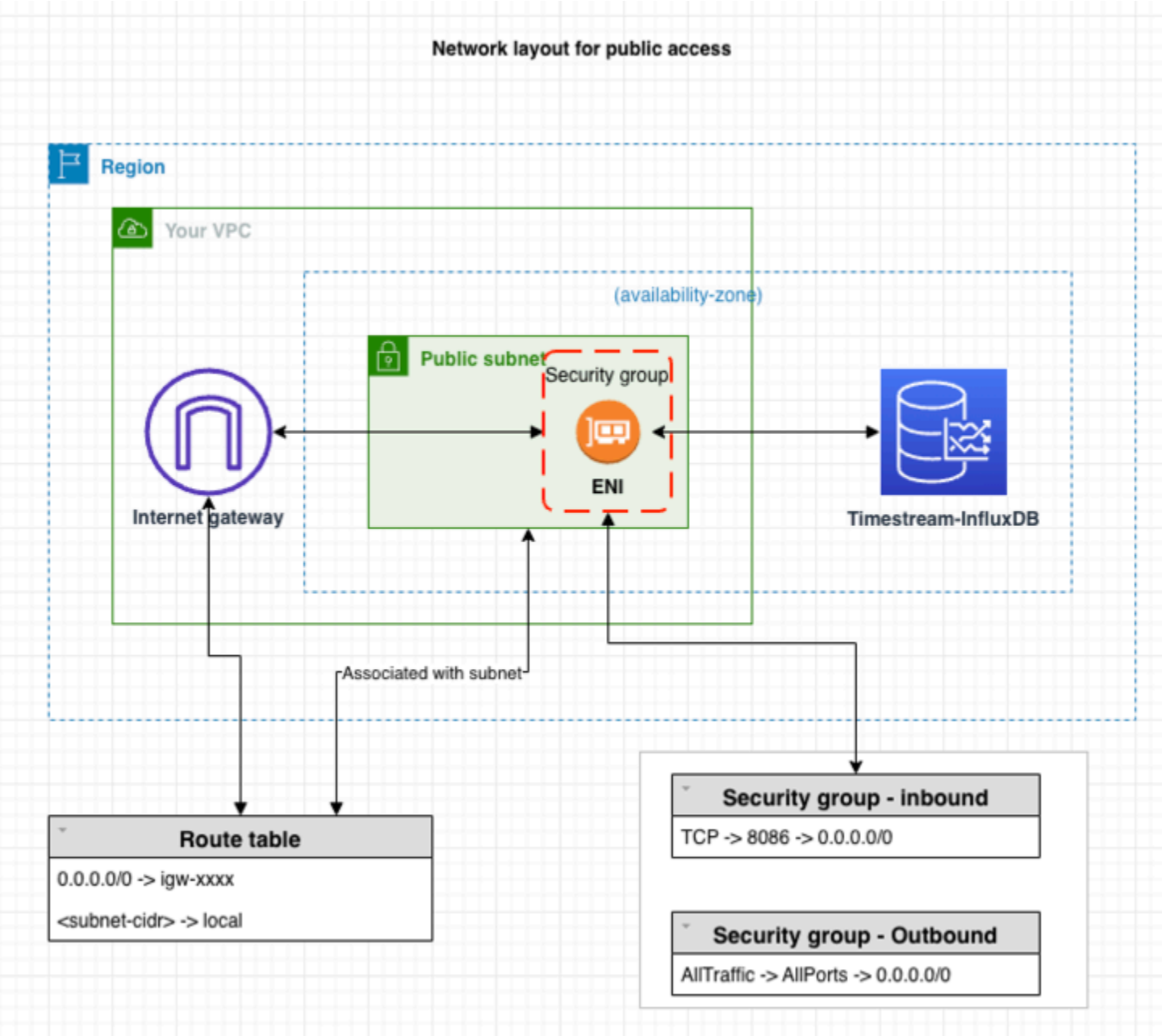
InfluxDB 인스턴스용 Timestream 생성 및 연결

이 자습서에서는 Amazon EC2 인스턴스와 Amazon Timestream for InfluxDB DB 인스턴스를 생성합니다. 자습서에서는 Telegraf 클라이언트를 사용하여 인스턴스에서 DB EC2 인스턴스에 데이터를 쓰는 방법을 보여줍니다. 모범 사례로 이 자습서는 가상 프라이빗 클라우드()에 프라이빗 DB 인스턴스를 생성합니다VPC. 대부분의 경우 EC2 인스턴스VPC와 같은 동일한 의 다른 리소스는 DB 인스턴스에 액세스할 수 있지만 외부의 리소스는 액세스할 VPC 수 없습니다.

자습서를 완료하면 의 각 가용 영역에 퍼블릭 및 프라이빗 서브넷이 있습니다VPC. 하나의 가용 영역에서 EC2 인스턴스는 퍼블릭 서브넷에 있고 DB 인스턴스는 프라이빗 서브넷에 있습니다.

> **Note**
>
> AWS 계정을 생성하는 데는 요금이 부과되지 않습니다. 그러나 이 자습서를 완료하면 사용하는 AWS 리소스에 비용이 발생할 수 있습니다. 자습서가 더 이상 필요하지 않은 경우 자습서를 완료한 후에 이러한 리소스를 삭제할 수 있습니다.

다음 다이어그램은 접근성이 공개된 구성을 보여줍니다.



Warning

모든 IP 주소가 를 통해 퍼블릭 InfluxDB 인스턴스에 HTTP 액세스할 수 있도록 하기 때문에 액세스에는 0.0.0.0/0을 사용하지 않는 것이 좋습니다HTTP. 이 접근 방식은 테스트 환경에서 단기간 동안도 허용되지 않습니다. WebUI 또는 액세스HTTP용 를 사용하여 InfluxDB 인스턴스에 액세스할 수 있도록 특정 IP 주소 또는 주소 범위만 승인합니다API.

이 자습서에서는 를 사용하여 InfluxDB를 실행하는 DB 인스턴스를 생성합니다 AWS Management Console. DB 인스턴스 크기와 DB 인스턴스 식별자에만 초점을 맞출 것입니다. 다른 구성 옵션에 기본 설정을 사용합니다. 이 예제에서 생성한 DB 인스턴스는 비공개입니다.

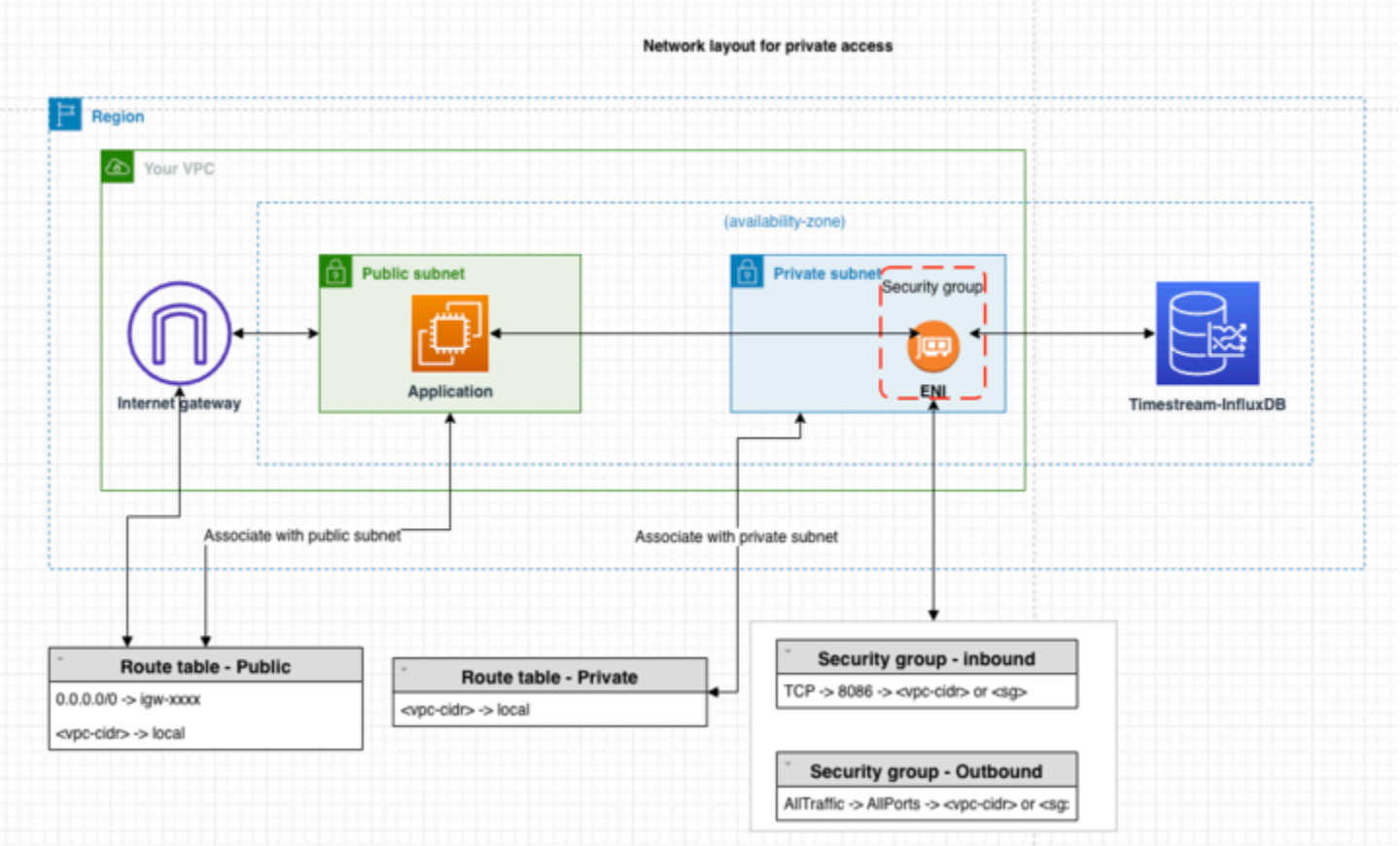
구성할 수 있는 다른 설정에는 가용성, 보안 및 로깅이 포함됩니다. 퍼블릭 DB 인스턴스를 생성하려면 연결 구성 섹션에서 인스턴스를 “공개 액세스 가능”으로 설정해야 합니다. DB 인스턴스 생성에 대한 자세한 내용은 섹션을 참조하세요DB 인스턴스 생성.

인스턴스에 공개적으로 액세스할 수 없는 경우 다음을 수행합니다.

- 트래픽을 터널링할 수 있는 인스턴스VPC의 에 호스트를 생성합니다.
- 인스턴스에 대한 ssh 터널링을 설정합니다. 자세한 내용은 AWS Systems Manager를 사용한 Amazon EC2 인스턴스 포트 전달을 참조하세요.
- 인증서가 작동하려면 클라이언트 시스템의 `/etc/hosts` 파일에 다음 줄을 추가합니다`127.0.0.1`. 인스턴스의 DNS 주소입니다.
- https://<DNS>:8086과 같은 정규화된 도메인 이름을 사용하여 인스턴스에 연결합니다.

> **Note**
>
> localhost는 인증서의 일부가 아니므로 Localhost가 인증서를 검증할 수 없습니다SAN.

다음 다이어그램은 접근성이 비공개일 때의 구성을 보여줍니다.



사전 조건

시작하기 전에 다음 섹션에서 다음 단계를 완료하세요.

- AWS 계정에 가입합니다.
- 관리 사용자를 생성합니다.

1단계: Amazon EC2 인스턴스 생성

데이터베이스에 연결하는 데 사용할 Amazon EC2 인스턴스를 생성합니다.

1. 에 로그인 AWS Management Console 하고 에서 Amazon EC2 콘솔을 엽니다https://console.aws.amazon.com/ec2/.
2. 의 오른쪽 상단에서 EC2 인스턴스를 생성할 AWS 리전을 AWS Management Console선택합니다.
3. EC2 대시보드 를 선택한 다음 인스턴스 시작 을 선택합니다.
4. 인스턴스 시작 페이지가 열리면 인스턴스 시작 페이지에서 다음 설정을 선택합니다.

a. 이름 및 태그의 이름에 ec2-database-connect를 입력합니다.

b. 애플리케이션 및 OS 이미지(Amazon Machine Image)에서 Amazon Linux를 선택한 다음 Amazon Linux 2023 을 선택합니다AMI. 다른 선택 항목에 대해서는 기본값을 그대로 유지합니다.

c. 인스턴스 유형에서 t2.micro를 선택합니다.

d. 키 페어(로그인)에서 기존 키 페어를 사용할 키 페어 이름을 선택합니다. Amazon EC2 인스턴스에 대한 새 키 페어를 생성하려면 새 키 페어 생성을 선택한 다음 키 페어 생성 창을 사용하여 생성합니다. 새 키 페어 생성에 대한 자세한 내용은 Linux 인스턴스용 Amazon 사용 설명서의 키 페어 생성을 참조하세요. EC2

e. 네트워크 설정에서 SSH 트래픽 허용에서 EC2 인스턴스에 대한 SSH 연결 소스를 선택합니다. 표시된 IP 주소가 SSH 연결에 적합한 경우 내 IP를 선택할 수 있습니다. 그렇지 않으면 Secure Shell()을 VPC 사용하여 의 EC2 인스턴스에 연결하는 데 사용할 IP 주소를 결정할 수 있습니다SSH. 퍼블릭 IP 주소를 확인하려면 다른 브라우저 창 또는 탭에서 의 서비스를 사용할 수 있습니다 https://checkip.amazonaws.com. IP 주소의 예는 192.0.2.1/32입니다. 많은 경우 인터넷 서비스 공급자(ISP)를 통해 연결하거나 정적 IP 주소 없이 방화벽 뒤에서 연결할 수 있습니다. 그렇다면 클라이언트 컴퓨터에서 사용하는 IP 주소 범위를 결정합니다.

> **Warning**
>
> 모든 IP 주소가 를 사용하여 퍼블릭 EC2 인스턴스에 SSH 액세스할 수 있도록 하기 때문에 액세스에 0.0.0.0/0을 사용하지 않는 것이 좋습니다SSH. 이 접근 방식은 테스트 환경에서 단기간 동안도 허용되지 않으며, 특정 IP 주소 또는 주소 범위만 를 사용하여 EC2 인스턴스에 액세스할 수 있도록 승인합니다SSH.

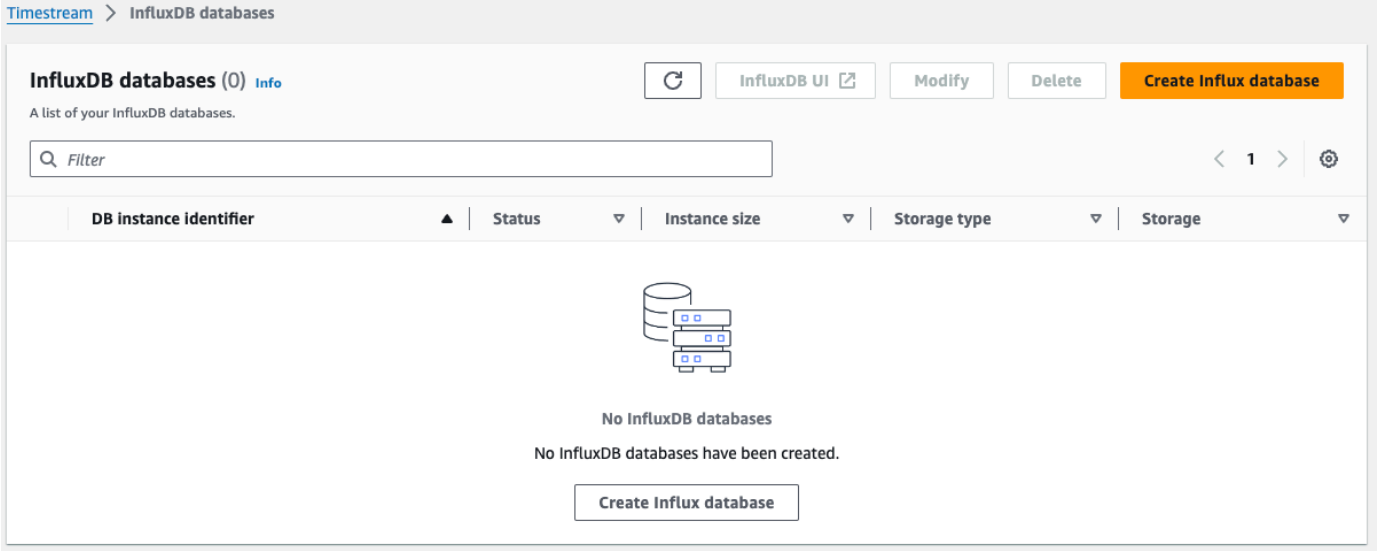
2단계: InfluxDB DB 인스턴스 생성

Amazon Timestream for InfluxDB의 기본 구성 요소는 DB 인스턴스입니다. 이 환경에서는 InfluxDB 데이터베이스를 실행합니다.

이 예제에서는 db.influx.large DB 인스턴스 클래스로 InfluxDB 데이터베이스 엔진을 실행하는 DB 인스턴스를 생성합니다.

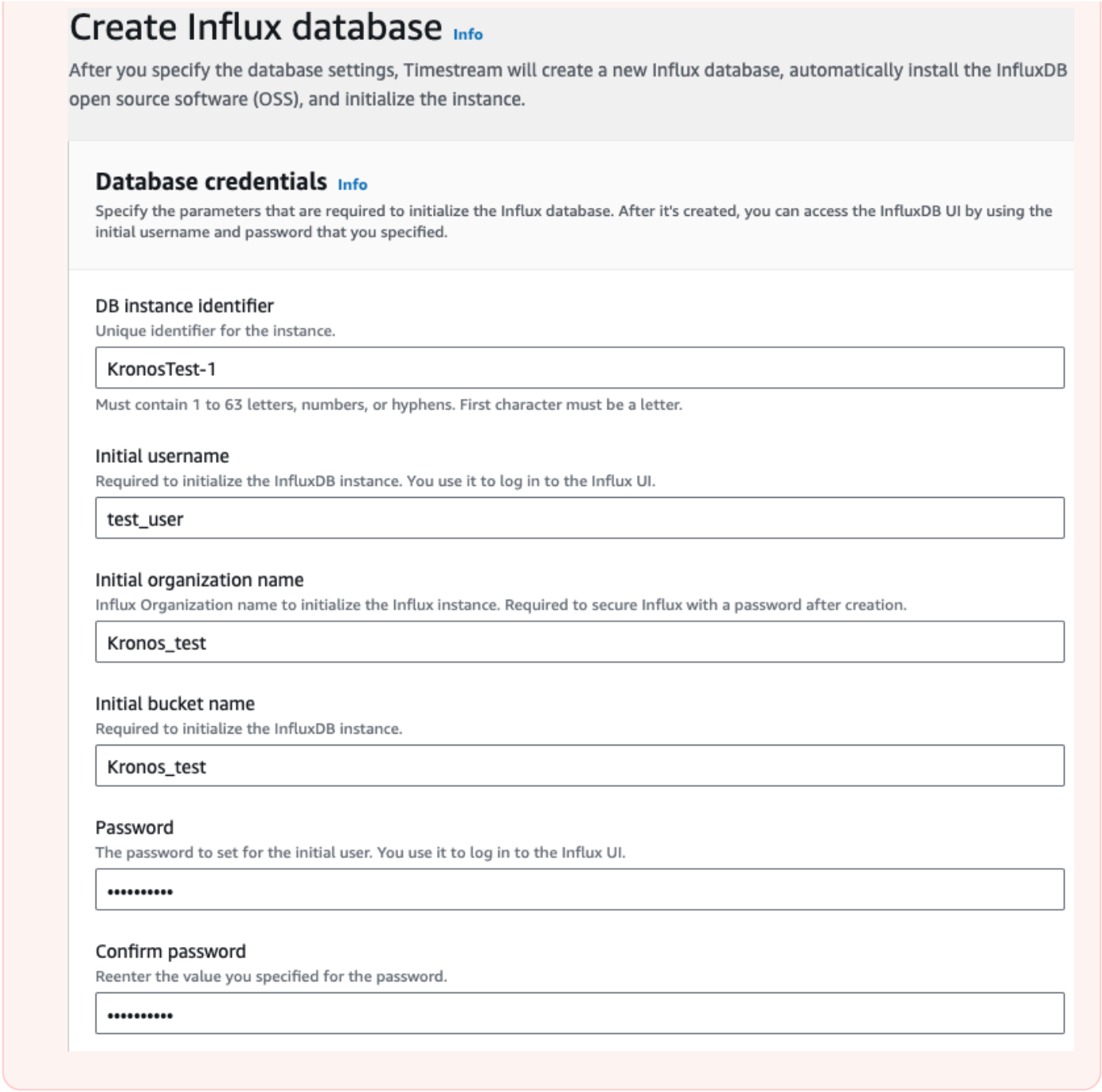
1. 에 로그인 AWS Management Console 하고 에서 Amazon Timestream for InfluxDB 콘솔을 엽니다https://console.aws.amazon.com/timestream/.

2. Amazon Timestream for InfluxDB 콘솔의 오른쪽 상단에서 DB 인스턴스를 생성할 AWS 리전을 선택합니다.
3. 탐색 창에서 InfluxDB 데이터베이스 를 선택합니다.
4. Influx 데이터베이스 생성을 선택합니다.

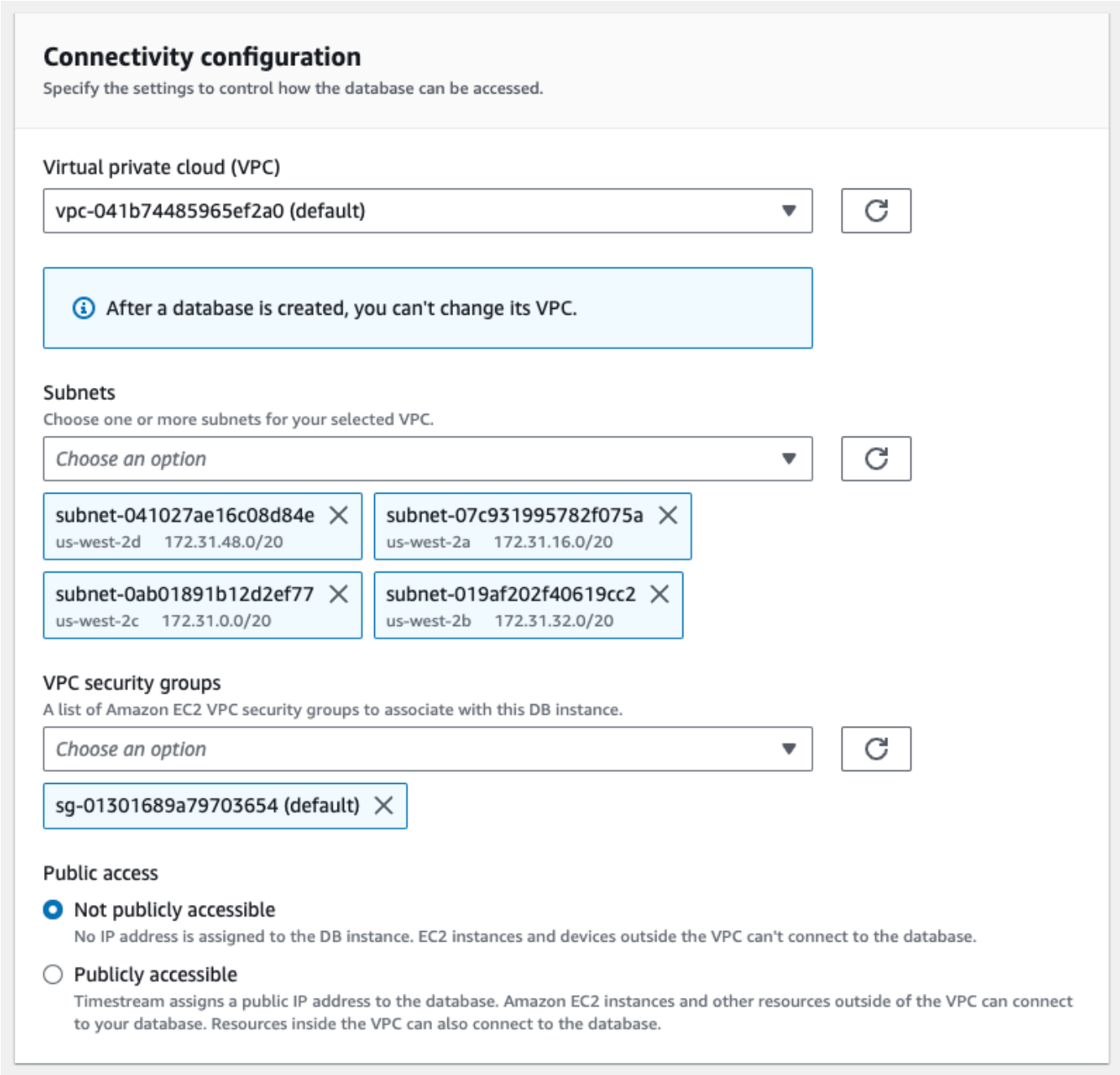


5. DB 인스턴스 식별자 에 KronosTest-1을 입력합니다.
6. InfluxDB 기본 구성 파라미터인 사용자 이름, 조직, 버킷 이름 및 암호 를 제공합니다.

> **Important**
>
> 사용자 암호를 다시 볼 수 없습니다. 암호 없이는 인스턴스에 액세스하고 운영자 토큰을 가져올 수 없습니다. 따라서 기록을 해두지 않으면 이를 변경해야 합니다. InfluxDB 인스턴스에 대한 새 운영자 토큰 생성을 참조하세요.
> DB 인스턴스를 사용할 수 있게 된 후 사용자 암호를 변경해야 하는 경우 DB 인스턴스를 수정하여 변경할 수 있습니다. DB 인스턴스 수정에 대한 자세한 내용은 DB 인스턴스 업데이트 단원을 참조하세요.



7. DB 인스턴스 클래스 에서 db.influx.large 를 선택합니다.
8. DB 스토리지 클래스 에서 IOPS 포함된 3K 유입 을 선택합니다.
9. 로그를 구성합니다. 자세한 내용은 Timestream Influxdb 인스턴스에서 InfluxDB 로그를 볼 수 있도록 설정 단원을 참조하십시오.
10. 연결 구성 섹션에서 InfluxDB 인스턴스가 새로 생성된 EC2 인스턴스와 동일한 서브넷에 있는지 확인합니다.



11. Influx 데이터베이스 생성을 선택합니다.
12. 데이터베이스 목록에서 새 InfluxDB 인스턴스의 이름을 선택하여 세부 정보를 표시합니다. DB 인스턴스는 사용할 준비가 될 때까지 생성 상태가 됩니다.

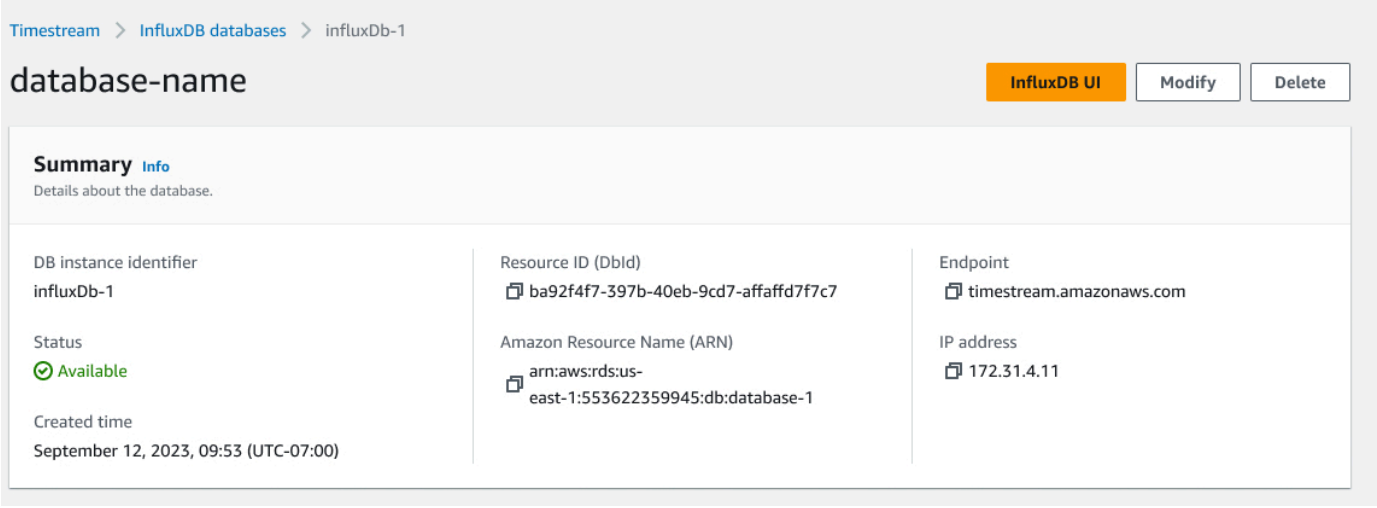
상태가 사용 가능으로 변경될 때 DB 인스턴스에 연결할 수 있습니다. DB 인스턴스 클래스와 스토리지의 양에 따라 새 인스턴스를 사용할 수 있을 때까지 최대 20분이 걸릴 수 있습니다.

> **Important**
>
> 현재 기존 인스턴스의 컴퓨팅(인스턴스 유형) 및 스토리지(스토리지 유형) 구성을 수정할 수 없습니다.

3단계: InfluxDB 인스턴스로 Telegraf 데이터 전송

이제 Telegraf 에이전트를 사용하여 InfluxDB DB 인스턴스로 원격 측정 데이터 전송을 시작할 수 있습니다. 이 예제에서는 InfluxDB DB 인스턴스에 성능 지표를 보내도록 Telegraf 에이전트를 설치하고 구성합니다.

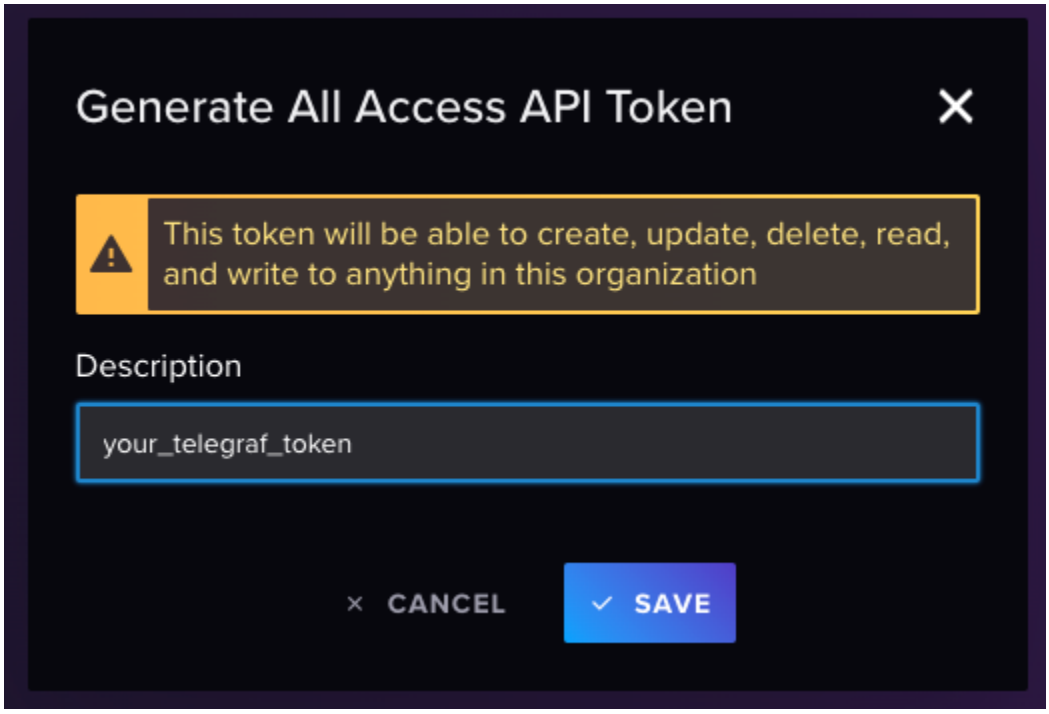
1. DB 인스턴스의 엔드포인트(DNS 이름) 및 포트 번호를 찾습니다.
   a. AWS 관리 콘솔에 로그인하고 에서 Amazon Timestream 콘솔을 엽니다https://console.aws.amazon.com/timestream/.
   b. Amazon Timestream 콘솔의 오른쪽 상단에서 DB 인스턴스의 AWS 리전을 선택합니다.
   c. 탐색 창에서 InfluxDB 데이터베이스 를 선택합니다.
   d. InfluxDB DB 인스턴스 이름을 선택하여 세부 정보를 표시합니다.
   e. 요약 섹션에서 엔드포인트를 복사합니다. 또한 포트 번호를 적어 둡니다. DB 인스턴스에 연결하려면 엔드포인트와 포트 번호가 모두 필요합니다(InfluxDB의 기본 포트 번호는 8086).
2. 그런 다음 InfluxDB UI 를 선택합니다.



3. 그러면 로그인 프롬프트가 표시되는 새 브라우저 창이 열립니다. 이전에 InfluxDB Db 인스턴스를 생성하는 데 사용한 자격 증명을 입력합니다.
4. 탐색 창에서 화살표를 클릭하고 API 토큰 을 선택합니다.

5. 이 테스트를 위해 모든 액세스 토큰을 생성합니다.

> **Note**
>
> 프로덕션 시나리오의 경우 특정 Telegraf 요구 사항에 맞게 구축된 필수 버킷에 대한 특정 액세스 권한이 있는 토큰을 생성하는 것이 좋습니다.



6. 토큰이 화면에 나타납니다.

> **Important**
>
> 토큰을 다시 표시할 수 없으므로 토큰을 복사하고 저장해야 합니다.

7. Linux EC2 인스턴스용 Amazon 사용 설명서 의 Linux 인스턴스에 연결의 단계에 따라 이전에 생성한 인스턴스에 연결합니다. EC2

   를 사용하여 EC2 인스턴스에 연결하는 것이 좋습니다SSH. SSH 클라이언트 유틸리티가 Windows, Linux 또는 Mac에 설치된 경우 다음 명령 형식을 사용하여 인스턴스에 연결할 수 있습니다.

```
ssh -i location_of_pem_file ec2-user@ec2-instance-public-dns-name
```

예를 들어 ec2-database-connect-key-pair.pem가 Linux/dir1에 저장되어 있고 EC2 인스턴스의 퍼블릭이 라고 가정IPv4DNS합니다ec2-12-345-678-90.compute-1.amazonaws.com. SSH 명령은 다음과 같습니다.

```
ssh -i /dir1/ec2-database-connect-key-pair.pem ec2-
user@ec2-12-345-678-90.compute-1.amazonaws.com
```

8. 인스턴스에 최신 버전의 텔레그래프를 설치합니다. 이렇게 하려면 다음 명령을 사용합니다.

```
cat <<EOF | sudo tee /etc/yum.repos.d/influxdata.repo
[influxdata]
name = InfluxData Repository - Stable
baseurl = https://repos.influxdata.com/stable/\$basearch/main
enabled = 1
gpgcheck = 1
gpgkey = https://repos.influxdata.com/influxdata-archive_compat.key
EOF

sudo yum install telegraf
```

9. Telegraf 인스턴스를 구성합니다.

> **Note**
>
> telegraf.conf가 존재하지 않거나 timestream 섹션이 포함된 경우 다음을 사용하여 섹션을 생성할 수 있습니다.
>
> ```
> telegraf –section-filter agent:inputs:outputs –input-filter cpu:mem –output-
> filter timestream config > telegraf.conf
> ```

a. 일반적으로 에 있는 구성 파일을 편집합니다/etc/telegraf.

```
sudo nano /etc/telegraf/telegraf.conf
```

b. CPU, MEM 및 에 대한 기본 입력을 구성합니다DISK.

```
[[inputs.cpu]]
  percpu = true
  totalcpu = true
```

```
  collect_cpu_time = false
  report_active = false

[[inputs.mem]]

[[inputs.disk]]
  ignore_fs = ["tmpfs", "devtmpfs", "devfs"]
```

c. InfluxDB DB 인스턴스로 데이터를 전송하고 변경 사항을 저장하도록 출력 플러그를 구성합니다.

```
[[outputs.influxdb_v2]]
   urls = ["https://us-west-2-1.aws.cloud2.influxdata.com"]
   token = "<your_telegraf_token"
   organization = "your_org"
   bucket = "your_bucket"
   timeout = "5s"
```

d. Timestream 대상을 구성합니다.

```
# Configuration for sending metrics to Amazon Timestream.
[[outputs.timestream]]

  ## Amazon Region and credentials
  region = "us-east-1"
  access_key = "<AWS key here>"
  secret_key = "<AWS secret key here>"
  database_name = "<timestream database name>" # needs to exist

  ## Specifies if the plugin should describe t start.
  describe_database_on_start = false
  mapping_mode = "multi-table" # allows multible tables for each input metrics

  create_table_if_not_exists = true
  create_table_magnetic_store_retention_period_in_days = 365
  create_table_memory_store_retention_period_in_hours = 24

  use_multi_measure_records = true # Important to use multi-measure records
  measure_name_for_multi_measure_records = "telegraf_measure"
  max_write_go_routines = 25
```

10. Telegraf 서비스를 활성화하고 시작합니다.

```
$ sudo systemctl enable telegraf
$ sudo systemctl start telegraf
```

4단계: Amazon EC2 인스턴스 및 InfluxDB DB 인스턴스 삭제

InfluxDB DB 인스턴스와 InfluxUI 를 사용하여 Telegraf에서 생성한 데이터를 탐색한 후 더 이상 요금이 부과되지 않도록 EC2 및 InfluxDB DB 인스턴스를 모두 삭제합니다.

EC2 인스턴스를 삭제하려면:

1. 에 로그인 AWS Management Console 하고 에서 Amazon EC2 콘솔을 엽니다https://console.aws.amazon.com/ec2/.
2. 탐색 창에서 Instances(인스턴스)를 선택합니다.
3. EC2 인스턴스를 선택하고 인스턴스 상태 를 선택한 다음 인스턴스 종료 를 선택합니다.
4. 확인 메시지가 나타나면 종료를 선택합니다.

EC2 인스턴스 삭제에 대한 자세한 내용은 Amazon EC2 사용 설명서의 인스턴스 종료를 참조하세요.

최종 DB 스냅샷 없이 DB 인스턴스를 삭제하려면:

1. 에 로그인 AWS Management Console 하고 에서 Amazon Timestream for InfluxDB 콘솔을 엽니다https://console.aws.amazon.com/timestream/.
2. 탐색 창에서 InfluxDB 데이터베이스 를 선택합니다.
3. 삭제할 DB 인스턴스를 선택합니다.
4. [Actions]에 대해 [Delete]를 선택합니다.
5. 확인을 완료하고 삭제를 선택합니다.

(선택 사항) Amazon Managed Grafana를 사용하여 DB 인스턴스에 연결

Amazon Managed Grafana를 사용하여 대시보드를 생성하고 Amazon Timestream for InfluxDB 를 사용하여 EC2 인스턴스의 성능을 모니터링할 수 있습니다. Amazon Managed Grafana는 지표, 로그 및 트레이스를 쿼리, 시각화 및 알릴 수 있는 인기 있는 오픈 소스 분석 플랫폼인 Grafana를 위한 완전 관리형 서비스입니다.

InfluxDB 인스턴스에 대한 새 운영자 토큰 생성

새 InfluxDB 인스턴스에 대한 Operator Token을 가져와야 하는 경우 다음 단계를 수행합니다.

1. 운영자 토큰을 변경하려면 Influx를 사용하는 것이 좋습니다CLI. 지침은 를 참조하세요. 유입 설치 및 사용 CLI
2. 연산자를 생성할 `--username-password` 수 CLI 있도록 를 사용하도록 구성합니다.

```
influx config create --config-name CONFIG_NAME1  --host-url "https://
yourinstanceid.eu-central-1.timestream-influxdb.amazonaws.com:8086" --org [YOURORG]
  --username-password [YOURUSERNAME] --active
```

3. 새 운영자 토큰을 생성합니다. 이 단계를 확인하기 위해 암호를 묻는 메시지가 표시됩니다.

```
influx auth create --org [YOURORG] --operator
```

> **Important**
>
> 새 운영자 토큰이 생성되면 현재 이전 토큰을 사용하고 있는 모든 클라이언트를 업데이트해야 합니다.

자체 관리형 InfluxDB에서 InfluxDB용 Timestream으로 데이터 마이그레이션

Influx 마이그레이션 스크립트는 InfluxDB OSS 인스턴스가 관리 AWS 되는지 여부에 관계없이 InfluxDB 인스턴스 간에 데이터를 마이그레이션하는 Python 스크립트입니다.

InfluxDB는 시계열 데이터베이스입니다. InfluxDB에는 여러 키-값 페어와 타임스탬프가 포함된 점 가 포함되어 있습니다. 포인트가 키-값 페어별로 그룹화되면 시리즈를 형성합니다. 시리즈는 측정이라는 문자열 식별자별로 그룹화됩니다. InfluxDB는 작업 모니터링, IOT 데이터 및 분석에 자주 사용됩니다. 버킷은 데이터를 저장하기 위한 InfluxDB 내의 일종의 컨테이너입니다. AWS관리형 InfluxDB는 AWS 에코시스템 내의 InfluxDB입니다. InfluxDB는 데이터에 액세스하고 데이터베이스를 변경하기 API 위한 InfluxDB v2를 제공합니다. InfluxDB v2API는 Influx 마이그레이션 스크립트가 데이터를 마이그레이션하는 데 사용하는 기능입니다.

- Influx 마이그레이션 스크립트는 버킷과 메타데이터를 마이그레이션하거나, 모든 조직의 모든 버킷을 마이그레이션하거나, 대상 인스턴스의 모든 데이터를 대체하는 전체 마이그레이션을 수행할 수 있습니다.
- 스크립트는 소스 인스턴스에서 데이터를 로컬로 백업하고, 어떤 시스템에서 스크립트를 실행하든 간에 데이터를 대상 인스턴스로 복원합니다. 데이터는 마이그레이션당 하나씩 code>influxdb-backup-<timestamp></timestamp> 디렉터리에 보관됩니다.
- 스크립트는 마이그레이션 중에 로컬 스토리지 사용을 제한하기 위해 S3 버킷을 탑재하고 마이그레이션 중에 사용할 조직을 선택하는 등 다양한 옵션과 구성을 제공합니다.

주제

준비

InfluxDB에 대한 데이터 마이그레이션은 InfluxDB CLI 기능과 InfluxDB InfluxDB v2를 활용하는 Python 스크립트를 사용하여 수행됩니다API. 마이그레이션 스크립트를 실행하려면 다음 환경 구성이 필요합니다.

- 지원되는 버전: InfluxDB 및 Influx의 최소 버전 2.3CLI이 지원됩니다.
- 토큰 환경 변수
  - 소스 InfluxDB 인스턴스의 토큰이 `INFLUX_SRC_TOKEN` 포함된 환경 변수를 생성합니다.
  - 대상 InfluxDB 인스턴스의 토큰이 `INFLUX_DEST_TOKEN` 포함된 환경 변수를 생성합니다.
- Python 3
  - 설치 확인: `python3 --version`.
  - 설치되지 않은 경우 Python 웹 사이트에서 를 설치합니다. 최소 버전 3.7이 필요합니다. Windows에서 기본 Python 3 별칭은 단순히 python입니다.
  - Python 모듈 요청이 필요합니다. 다음을 사용하여 설치합니다. `shell python3 -m pip install requests`
  - TThe Python 모듈 influxdb_client가 필요합니다. 다음을 사용하여 설치합니다. `shell python3 -m pip install influxdb_client`
- InfluxDB CLI

  - 설치 확인: `influx version`.
  - 설치되지 않은 경우 InfluxDB 설명서 의 설치 안내서를 따릅니다.

    $에 유입을 추가합니다PATH.
- S3 탑재 도구(선택 사항)

  S3 탑재를 사용하면 모든 백업 파일이 사용자 정의 S3 버킷에 저장됩니다. S3 탑재는 실행 중인 시스템의 공간을 절약하거나 백업 파일을 공유해야 할 때 유용할 수 있습니다. S3 탑재를 사용하지 않는 경우 `--s3-bucket` 옵션을 생략하면 로컬 `influxdb-backup-<millisecond timestamp>` 디렉터리가 생성되어 스크립트가 실행된 디렉터리와 동일한 디렉터리에 백업 파일을 저장합니다.

  Linux의 경우: mountpoint-s3.

  Windows의 경우: rclone(이전 rclone 구성 필요).
- 디스크 공간
  - 마이그레이션 프로세스는 백업 파일 세트를 저장하는 고유한 디렉터리를 자동으로 생성하고 제공된 프로그램 인수에 따라 이러한 백업 디렉터리를 S3 또는 로컬 파일 시스템에 유지합니다.
  - 데이터베이스 백업을 위한 디스크 공간이 충분한지 확인합니다. `--s3-bucket` 옵션을 생략하고 백업 및 복원에 로컬 스토리지를 사용하는 경우 기존 InfluxDB 데이터베이스의 크기를 두 배로 늘리는 것이 좋습니다.
  - Windows에서 드라이브 속성을 확인하여 `df -h (UNIX/Linux)` 또는 를 사용하여 공간을 확인합니다.
- 직접 연결

  마이그레이션 스크립트를 실행하는 시스템과 소스 및 대상 시스템 간에 직접 네트워크 연결이 있는지 확인합니다. `influx ping --host <host>`는 직접 연결을 확인하는 한 가지 방법입니다.

스크립트 사용 방법

스크립트 실행의 간단한 예는 명령입니다.

```
python3 influx_migration.py --src-host <source host> --src-bucket <source bucket> --
dest-host <destination host>
```

단일 버킷을 마이그레이션합니다.

다음을 실행하여 모든 옵션을 볼 수 있습니다.

```
python3 influx_migration.py -h
```

사용량

```
shell   influx_migration.py [-h] [--src-bucket SRC_BUCKET] [--dest-bucket DEST_BUCKET]
 [--src-host SRC_HOST] --dest-host DEST_HOST [--full] [--confirm-full] [--src-org
 SRC_ORG] [--dest-org DEST_ORG] [--csv] [--retry-restore-dir RETRY_RESTORE_DIR] [--dir-
name DIR_NAME] [--log-level LOG_LEVEL] [--skip-verify] [--s3-bucket S3_BUCKET]
```

옵션

- -confirm-full(선택 사항): `--csv` 를 `--full` 사용하지 않으면 대상 데이터베이스의 모든 토큰, 사용자, 버킷, 대시보드 및 기타 키-값 데이터가 소스 데이터베이스의 토큰, 사용자, 버킷, 대시보드 및 기타 키-값 데이터로 바뀝니다. `--full` 는 버킷 조직을 포함한 모든 버킷 및 버킷 메타데이터`--csv`만 마이그레이션합니다. 이 옵션(`--confirm-full`)은 전체 마이그레이션을 확인하고 사용자 입력 없이 진행합니다. 이 옵션이 제공되지 않고 제공 `--full` 및 제공되지 `--csv` 않은 경우 스크립트는 실행을 일시 중지하고 사용자 확인을 기다립니다. 이는 중요한 작업이므로 주의하여 진행하세요. 기본값은 false입니다.
- -csv(선택 사항): 백업 및 복원에 csv 파일을 사용할지 여부. 도 `--full` 전달하면 시스템 버킷, 사용자, 토큰 또는 대시보드가 아닌 모든 조직의 모든 사용자 정의 버킷이 마이그레이션됩니다. 기존 소스 조직 대신 대상 서버의 모든 버킷에 대해 단일 조직을 원하는 경우 를 사용합니다`--dest-org`.
- -dest-bucket DEST_BUCKET(선택 사항): 대상 서버의 InfluxDB 버킷 이름은 이미 존재하는 버킷이 아니어야 합니다. 기본값은 제공되지 `--src-bucket` 않은 None 경우 `--src-bucket` 또는 입니다.
- -dest-host DEST_HOST: 대상 서버의 호스트입니다. 예: http://localhost:8086.
- -dest-org DEST_ORG(선택 사항): 대상 서버에서 버킷을 복원할 조직의 이름입니다. 이를 생략하면 소스 서버에서 마이그레이션된 모든 버킷은 원래 조직을 유지하며 조직을 생성하고 전환하지 않으면 마이그레이션된 버킷이 대상 서버에서 표시되지 않을 수 있습니다. 이 값은 단일 버킷, 전체 마이그레이션 또는 백업 및 복원을 위해 csv 파일을 사용하는 마이그레이션 등 모든 형태의 복원에 사용됩니다.
- -dir-name DIR_NAME(선택 사항): 생성할 백업 디렉터리의 이름입니다. 기본값은 `influxdb-backup-<timestamp>`입니다. 아직 존재하지 않아야 합니다.
- -전체(선택 사항): 전체 복원을 수행할지 여부, 대상 서버의 모든 데이터를 토큰, 대시보드, 사용자 등과 같은 모든 키-값 데이터를 포함하여 모든 조직의 소스 서버의 모든 데이터로 바꿉니다. `--src-bucket` 및 를 재정의합니다`--dest-bucket`. 와 함께 사용하는 경우 `--csv`는 버킷의 데이터 및 메타데이터만 마이그레이션합니다. 기본값은 false입니다.

- h, --help : 도움말 메시지를 표시하고 종료합니다.
- -log-level LOG_LEVEL(선택 사항): 실행 중에 사용할 로그 수준입니다. 옵션은 디버그, 오류 및 정보입니다. 기본값은 정보입니다.
- -retry-restore-dir RETRY_RESTORE_DIR(선택 사항): 이전 복원에 실패했을 때 복원에 사용할 디렉터리는 백업 및 디렉터리 생성을 건너뛰고 디렉터리가 없으면 실패하며 S3 버킷 내의 디렉터리가 될 수 있습니다. 복원에 실패하면 복원에 사용할 수 있는 백업 디렉터리 경로가 현재 디렉터리를 기준으로 표시됩니다. S3 버킷은 형식입니다`influxdb-backups/<s3 bucket>/<backup directory>`. 기본 백업 디렉터리 이름은 입니다`influxdb-backup-<timestamp>`.
- -s3-bucket S3_BUCKET(선택 사항): 백업 파일을 저장하는 데 사용할 S3 버킷의 이름입니다. Linux에서는 , `my-bucket`지정된 `AWS_ACCESS_KEY_ID` 및 `AWS_SECRET_ACCESS_KEY` 환경 변수가 설정되었거나 `${HOME}/.aws/credentials` 존재하는 등의 S3 버킷의 이름입니다. Windows에서는 와 같이 `rclone` 구성된 원격 및 버킷 이름입니다`my-remote:my-bucket`. 모든 백업 파일은 생성된 `influxdb-backups-<timestamp>` 디렉터리에서 마이그레이션한 후 S3 버킷에 남습니다. 라는 임시 탑재 디렉터리가 이 스크립트가 실행되는 디렉터리에 `influx-backups` 생성됩니다. 제공되지 않으면 모든 백업 파일이 이 스크립트가 실행되는 생성된 `influxdb-backups-<timestamp>` 디렉터리에 로컬로 저장됩니다.
- -skip-verify(선택 사항): TLS 인증서 확인을 건너뜁니다.
- -src-bucket SRC_BUCKET(선택 사항): 소스 서버의 InfluxDB 버킷 이름입니다. 제공되지 않은 경우를 제공해야 `--full` 합니다.
- -src-host SRC_HOST(선택 사항): 소스 서버의 호스트입니다. 기본값은 http://localhost:8086.

앞서 언급한 대로 `--s3-bucket`를 사용해야 하는 경우 `mountpoint-s3` 및 `rclone` 가 필요하지만 사용자가 에 대한 값을 제공하지 않으면 무시할 수 `--s3-bucket`있습니다. 이 경우 백업 파일은 로컬에 고유한 디렉터리에 저장됩니다.

마이그레이션 개요

사전 조건을 충족한 후:

1. 마이그레이션 스크립트 실행: 선택한 터미널 앱을 사용하여 Python 스크립트를 실행하여 소스 InfluxDB 인스턴스에서 대상 InfluxDB 인스턴스로 데이터를 전송합니다.
2. 자격 증명 제공: 호스트 주소와 포트를 CLI 옵션으로 제공합니다.
3. 데이터 확인: 다음을 통해 데이터가 올바르게 전송되었는지 확인합니다.

    a. InfluxDB UI 사용 및 버킷 검사.

b. 로 버킷 나열. `influx bucket list -t <destination token> --host <destination host address> --skip-verify`

c. `influx v1 shell -t <destination token> --host <destination host address> --skip-verify` 및 `SELECT * FROM <migrated bucket>.<retention period>.<measurement name> LIMIT 100 to view contents of a bucket or SELECT COUNT(*) FROM <migrated bucket>.<retention period>.<measurment name>`를 사용하여 올바른 레코드 수가 마이그레이션되었는지 확인합니다.

Example 실행 예제

1. 원하는 터미널 앱을 열고 필요한 사전 요구 사항이 제대로 설치되어 있는지 확인합니다.

```
~ > python3 --version
Python 3.11.5
~ > influx version
Influx CLI 2.7.3 (git: 8b962c7e75) build_date: 2023-04-28T14:22:49Z
~ > s3fs --version
Amazon Simple Storage Service File System V1.92 (commit:unknown) with GnuTLS(gcrypt)
Copyright (C) 2010 Randy Rizun <rrizun@gmail.com>
License GPL2: GNU GPL version 2 <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
~ >
```

2. 마이그레이션 스크립트로 이동합니다.

```
~ > cd sample-code/influxdb-sample/migration/influxdb
~/sample-code/influxdb-sample/migration/influxdb > ls *.py
influx_migration.py
~/sample-code/influxdb-sample/migration/influxdb >
```

3. 다음 정보를 준비합니다.

a. 마이그레이션할 소스 버킷의 이름입니다.

b. (선택 사항) 대상 서버에서 마이그레이션된 버킷의 새 버킷 이름을 선택합니다.

c. 소스 및 대상 유입 인스턴스의 루트 토큰입니다.

d. 소스 및 대상 유입 인스턴스의 호스트 주소입니다.

e. (선택 사항) S3 버킷 이름 및 자격 증명, AWS Command Line Interface 자격 증명은 OS 환경 변수에 설정해야 합니다.

```
# AWS credentials (for timestream testing)
    export AWS_ACCESS_KEY_ID="xxx"
    export AWS_SECRET_ACCESS_KEY="xxx"
```

f. 명령을 다음과 같이 구성합니다.

```
python3 influx_migration.py --src-bucket [source-bucket-name]  --dest-bucket
 [dest-bucket-name] --src-host [source host] --dest-host [dest host] --s3-
bucket [s3 bucket name](optional) --log-level debug
```

g. 스크립트를 실행합니다.

```
~/sample-code/influxdb-sample/migration/influxdb > python3 influx_migration.py --src-bucket primary-bucket --src-host $INFLUXDB_1_HOST --dest-host $KRO
NOS_HOST --dest-bucket new-bucket-name
```

h. 스크립트 실행이 완료될 때까지 기다립니다.

i. 새로 마이그레이션된 버킷의 데이터 무결성을 확인합니다`performance.txt`. 스크립트가 실행된 디렉터리 아래에 있는 이 파일에는 각 단계가 얼마나 오래 걸렸는지에 대한 몇 가지 기본 정보가 포함되어 있습니다.

마이그레이션 시나리오

Example 예제 1: 로컬 스토리지를 사용한 간단한 마이그레이션

소스 서버에서 대상 서버 로 단일 버킷인 기본 버킷을 마이그레이션하려고 (`http://localhost:8086`) 합니다(`http://dest-server-address:8086`).

포트 8086에서 InfluxDB 인스턴스를 호스팅하는 두 시스템에 TCP 대한 액세스 권한(HTTP액세스용)이 있고 소스 토큰과 대상 토큰이 모두 있고 보안을 강화하기 위해 `INFLUX_DEST_TOKEN`각각 환경 변수 `INFLUX_SRC_TOKEN` 및 로 저장했는지 확인한 후:

```
python3 influx_migration.py --src-bucket primary-bucket --src-host http://
localhost:8086 --dest-host http://dest-server-address:8086
```

다음과 같이 출력됩니다

```
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB CLI
2023/10/26 10:47:15 INFO: Downloading metadata snapshot
2023/10/26 10:47:15 INFO: Backing up TSM for shard 1
2023/10/26 10:47:15 INFO: Backing up TSM for shard 8245
2023/10/26 10:47:15 INFO: Backing up TSM for shard 8263
```

```
[More shard backups . . .]
2023/10/26 10:47:20 INFO: Backing up TSM for shard 8240
2023/10/26 10:47:20 INFO: Backing up TSM for shard 8268
2023/10/26 10:47:20 INFO: Backing up TSM for shard 2
INFO: influx_migration.py: Restoring bucket data and metadata using the InfluxDB CLI
2023/10/26 10:47:20 INFO: Restoring bucket "96c11c8876b3c016" as "primary-bucket"
2023/10/26 10:47:21 INFO: Restoring TSM snapshot for shard 12772
2023/10/26 10:47:22 INFO: Restoring TSM snapshot for shard 12773
[More shard restores . . .]
2023/10/26 10:47:28 INFO: Restoring TSM snapshot for shard 12825
2023/10/26 10:47:28 INFO: Restoring TSM snapshot for shard 12826
INFO: influx_migration.py: Migration complete
```

디렉터리가 생성`influxdb-backup-<timestamp>`되어 백업 파일이 포함된 스크립트가 실행된 디렉터리에 저장됩니다.

Example 예제 2: 로컬 스토리지 및 디버그 로깅을 사용한 전체 마이그레이션

모든 버킷, 토큰, 사용자 및 대시보드를 마이그레이션하고, 대상 서버에서 버킷을 삭제하고, `--confirm-full` 옵션을 사용하여 전체 데이터베이스 마이그레이션을 사용자가 확인하지 않고 진행하려는 경우를 제외하고 위와 동일합니다. 디버그 로깅을 활성화하려면 성능 측정값이 무엇인지도 확인해야 합니다.

```
python3 influx_migration.py --full --confirm-full --src-host http://localhost:8086 --
dest-host http://dest-server-address:8086 --log-level debug
```

다음과 같이 출력됩니다

```
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB CLI
2023/10/26 10:55:27 INFO: Downloading metadata snapshot
2023/10/26 10:55:27 INFO: Backing up TSM for shard 6952
2023/10/26 10:55:27 INFO: Backing up TSM for shard 6953
[More shard backups . . .]
2023/10/26 10:55:36 INFO: Backing up TSM for shard 8268
2023/10/26 10:55:36 INFO: Backing up TSM for shard 2
DEBUG: influx_migration.py: backup started at 2023-10-26 10:55:27 and took 9.41 seconds
 to run.
INFO: influx_migration.py: Restoring bucket data and metadata using the InfluxDB CLI
2023/10/26 10:55:36 INFO: Restoring KV snapshot
2023/10/26 10:55:38 WARN: Restoring KV snapshot overwrote the operator token, ensure
 following commands use the correct token
```

```
2023/10/26 10:55:38 INFO: Restoring SQL snapshot
2023/10/26 10:55:39 INFO: Restoring TSM snapshot for shard 6952
2023/10/26 10:55:39 INFO: Restoring TSM snapshot for shard 6953
[More shard restores . . .]
2023/10/26 10:55:49 INFO: Restoring TSM snapshot for shard 8268
2023/10/26 10:55:49 INFO: Restoring TSM snapshot for shard 2
DEBUG: influx_migration.py: restore started at 2023-10-26 10:55:36 and took 13.51
 seconds to run.
INFO: influx_migration.py: Migration complete
```

Example 예제 3: CSV, 대상 조직 및 S3 버킷을 사용한 전체 마이그레이션

이전 예제와 동일하지만 Linux 또는 Mac을 사용하고 S3 버킷에 파일을 저장하는 경우 `my-s3-bucket`. 이렇게 하면 백업 파일이 로컬 스토리지 용량에 과부하를 일으키지 않습니다.

```
python3 influx_migration.py --full --src-host http://localhost:8086 --dest-host http://
dest-server-address:8086 --csv --dest-org MyOrg --s3-bucket my-s3-bucket
```

다음과 같이 출력됩니다

```
INFO: influx_migration.py: Creating directory influxdb-backups
INFO: influx_migration.py: Mounting influxdb-migration-bucket
INFO: influx_migration.py: Creating directory influxdb-backups/my-s3-bucket/influxdb-
backup-1698352128323
INFO: influx_migration.py: Backing up bucket data and metadata using the InfluxDB v2
 API
INFO: influx_migration.py: Restoring bucket data and metadata from csv
INFO: influx_migration.py: Restoring bucket some-bucket
INFO: influx_migration.py: Restoring bucket another-bucket
INFO: influx_migration.py: Restoring bucket primary-bucket
INFO: influx_migration.py: Migration complete
INFO: influx_migration.py: Unmounting influxdb-backups
INFO: influx_migration.py: Removing temporary mount directory
```

DB 인스턴스 구성

이 섹션에서는 Amazon Timestream for InfluxDB DB 인스턴스를 설정하는 방법을 보여줍니다. DB 인스턴스를 생성하기 전에 DB 인스턴스를 실행할 DB 인스턴스 클래스를 결정합니다. 또한 AWS 리전을 선택하여 DB 인스턴스가 실행되는 위치를 결정합니다. 그런 다음 DB 인스턴스를 생성합니다.

DB 파라미터 그룹으로 DB 인스턴스를 구성할 수 있습니다.DB 파라미터 그룹은 하나 이상의 DB 인스턴스에 적용되는 엔진 구성 값의 컨테이너 역할을 합니다.

사용 가능한 파라미터는 DB 엔진 및 DB 엔진 버전에 따라 다릅니다. DB 인스턴스를 생성할 때 DB 파라미터 그룹을 지정할 수 있습니다. DB 인스턴스를 수정하여 지정할 수도 있습니다.

> **Important**
>
> 현재 기존 인스턴스의 컴퓨팅(인스턴스 유형) 및 스토리지(스토리지 유형) 구성을 수정할 수 없습니다.

DB 인스턴스 생성

콘솔 사용

1. 에 로그인 AWS Management Console 하고 Amazon Timestream for InfluxDB 를 엽니다.
2. Amazon Timestream for InfluxDB 콘솔의 오른쪽 상단에서 DB 인스턴스를 생성할 AWS 리전을 선택합니다.
3. 탐색 창에서 InfluxDB 데이터베이스 를 선택합니다.
4. Influx 데이터베이스 생성을 선택합니다.
5. DB 인스턴스 식별자 에 인스턴스를 식별할 이름을 입력합니다.
6. InfluxDB 기본 구성 파라미터 사용자 이름, 조직, 버킷 이름 및 암호 를 제공합니다.

   > **Important**
   >
   > 사용자 이름, 조직, 버킷 이름 및 암호는 AWS Secrets Manager에 보안 암호로 저장되며, 보안 암호는 계정에 대해 생성됩니다.

   DB 인스턴스를 사용할 수 있게 된 후 사용자 암호를 변경해야 하는 경우 Influx CLI를 사용하여 수정할 수 있습니다.
7.
8. DB 인스턴스 클래스 에서 워크로드 요구 사항에 더 적합한 인스턴스 크기를 선택합니다.
9. DB 스토리지 클래스 에서 필요에 맞는 스토리지 클래스를 선택합니다. 모든 경우에 할당된 스토리지만 구성하면 됩니다.

10. 연결 구성 섹션에서 InfluxDB용 Timestream DB 인스턴스에 연결해야 하는 새 클라이언트와 동일한 서브넷에 InfluxDB 인스턴스가 있는지 확인합니다. DB 인스턴스를 공개적으로 사용할 수 있도록 선택할 수도 있습니다.
11. Influx 데이터베이스 생성을 선택합니다.
12. 데이터베이스 목록에서 새 InfluxDB 인스턴스의 이름을 선택하여 세부 정보를 표시합니다. DB 인스턴스의 상태는 가 사용할 준비가 될 때까지 생성입니다.
13. 상태가 Available(사용 가능)로 변경되면 DB 인스턴스에 연결할 수 있습니다. DB 인스턴스 클래스와 스토리지의 양에 따라 새 인스턴스를 사용할 수 있을 때까지 최대 20분이 걸릴 수 있습니다.

사용 CLI

를 사용하여 DB 인스턴스를 생성하려면 다음 파라미터로 `create-db-instance` 명령을 AWS Command Line Interface호출합니다.

```
--name
--vpc-subnet-ids
--vpc-security-group-ids
--db-instance-type
--db-storage-type
--username
--organization
--password
--allocated-storage
```

각 설정에 대한 자세한 내용은 DB 인스턴스에 대한 설정 단원을 참조하세요.

Example 예: 기본 엔진 구성 사용

Linux, macOS, Unix의 경우:

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

Windows의 경우:

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

사용 API

를 사용하여 DB 인스턴스를 생성하려면 다음 파라미터를 사용하여 `CreateDBInstance` 명령을 AWS Command Line Interface호출합니다.

각 설정에 대한 자세한 내용은 DB 인스턴스에 대한 설정 단원을 참조하세요.

> **Important**
>
> 를 수신하는 DBInstance 응답 객체의 일부입니다 influxAuthParametersSecretArn. 이렇게 하면 계정의 SecretsManager 보안 암호ARN에 가 저장됩니다. InfluxDB DB 인스턴스를 사용할 수 있는 경우에만 채워집니다. 보안 암호에는 `CreateDbInstance` 프로세스 중에 제공된 유입 인증 파라미터가 포함되어 있습니다. 생성된 DB 인스턴스updates/modifications/deletions에는 영향을 주지 않으므로 이 보안 암호의 READONLY 복사본입니다. 이 보안 암호를 삭제해도 API 응답은 삭제된 보안 암호 를 계속 참조합니다ARN.

Timestream for InfluxDB DB 인스턴스 생성을 완료한 후에는 Influx 를 다운로드, 설치 및 구성하는 것이 좋습니다CLI.

유입은 명령줄에서 InfluxDB와 상호 작용하는 간단한 방법을 CLI 제공합니다. 자세한 설치 및 설정 지침은 Influx 사용을 참조하세요CLI.

DB 인스턴스에 대한 설정

콘솔, `create-db-instance` CLI 명령 또는 `CreateDBInstance` Timestream for InfluxDB API 작업을 사용하여 DB 인스턴스를 생성할 수 있습니다.

다음 표에서는 DB 인스턴스를 생성할 때 선택하는 설정에 대한 세부 정보를 제공합니다.

콘솔 설정	설명	CLI 옵션 및 Timestream API 파라미터
할당된 스토리지	DB 인스턴스에 할당할 스토리지 양(기가바이트 단위)입니다. 경우에 따라 DB 인스턴스에 대해 데이터베이스의 크기보다 많은 양의 스토리지를 할당하면 I/O 성능을 개선할 수 있습니다. 자세한 내용은 InfluxDB 인스턴스 스토리지 단원을 참조하십시오.	CLI: `allocated-storage` API: `allocatedstorage`
버킷 이름	InfluxDb 인스턴스를 초기화할 버킷의 이름	CLI: `bucket` API: `bucket`
DB 인스턴스 유형	DB 인스턴스에 대한 구성입니다. 예를 들어 db.influx.large DB 인스턴스 클래스에는 16개의 GiB 메모리, 2개의 vCPUs메모리가 최적화되어 있습니다. 가능하면 일반적인 쿼리 작업 세트를 메모리에 보관할 수 있을 만큼 큰 DB 인스턴스 유형을 선택합니다. 작업 세트가 메모리에 상주할 경우 시스템의 디스크 쓰기가 불필요하여 성능이 향상됩니다. 자세한 내용은 DB 인스턴스 클래스 유형 단원을 참조하십시오.	CLI: `db-instance-type` API: `Dbinstancetype`
DB 인스턴스 식별자	DB 인스턴스의 이름입니다. 온프레미스 서버와 동일한 방식으로 DB 인스턴스의 이름을 지정합니다. DB 인스턴스 식별자는 최대 63자의 영숫자를 포함할 수 있으며 선택한 AWS 리전의 계정에 대해 고유해야 합니다.	CLI: `db-instance-identifier` API: `Dbinstanceidentifier`
DB 파라미터 그룹	DB 인스턴스의 파라미터 그룹입니다. 기본 파라미터 그룹을 사용하거나 사용자 지정 파라미터 그룹을 생성할 수 있습니다. 자세한 내용은 섹션을 참조하세요DB 파라미터 그룹 작업.	CLI: `db-parameter-group-name` API: `DBParameterGroupName`

콘솔 설정	설명	CLI 옵션 및 Timestream API 파라미터
로그 전송 설정	InfluxDB 로그가 저장될 S3 버킷의 이름입니다.	CLI: `LogDeliveryConfiguration` API: `log-delivery-configuration`
다중 AZ 배포	장애 조치를 위해 다른 가용 영역에 DB 인스턴스의 수동 보조 복제본을 생성하려면 Create a standby instance(대기 인스턴스를 생성)를 선택합니다. 이때 고가용성을 유지하려면 프로덕션 워크로드를 위한 다중 AZ를 권장합니다. 개발 및 테스트의 경우 Do not create a standby instance(대기 인스턴스를 생성하지 않음)를 선택할 수 있습니다. 자세한 내용은 다중 AZ 배포 구성 및 관리 단원을 참조하십시오.	CLI: `MultiAz` API: `multi-az`
암호	이는 InfluxDB Db 인스턴스를 초기화하는 데 사용되는 마스터 사용 암호입니다. 이 암호를 사용하여 InfluxUI에 로그인하여 운영자 토큰을 가져옵니다.	CLI: `password` API: `password`

콘솔 설정	설명	CLI 옵션 및 Timestream API 파라미터
퍼블릭 액세스	DB 인스턴스에 퍼블릭 IP 주소를 부여하려면 예. 즉, 외부에서 액세스할 수 있습니다VPC. 공개적으로 액세스하려면 DB 인스턴스도 의 퍼블릭 서브넷에 있어야 합니다VPC. 아니요. 내부에서만 DB 인스턴스에 액세스할 수 있도록 하려면 아니요VPC. 외부에서 DB 인스턴스에 연결하려면 DB 인스턴스 VPC에 공개적으로 액세스할 수 있어야 합니다. 또한 DB 인스턴스 보안 그룹의 인바운드 규칙을 사용하여 액세스 권한을 부여해야 하며, 다른 요구 사항도 충족해야 합니다.	CLI: `publicly-accessible` API: `PubliclyAccessible`
스토리지 유형	DB 인스턴스의 스토리지 유형 워크로드 요구 사항에 따라 프로비저닝된 유입 IOPS 포함 스토리지의 3가지 유형 중에서 선택할 수 있습니다. * Influx IOPS 포함 3000 IOPS * Influx IOPS 포함 12000 IOPS * 16000 INflux IOPS 포함 IOPS 자세한 내용은 InfluxDB 인스턴스 스토리지 단원을 참조하십시오.	CLI: `db-storage-type` API: `DbStorageType`
초기 사용자 이름	이 사용자는 InfluxDB DB 인스턴스를 초기화할 마스터 사용자입니다. 이 사용자 이름을 사용하여 InfluxUI에 로그인하여 운영자 토큰을 가져옵니다.	CLI: `username` API: `Username`

콘솔 설정	설명	CLI 옵션 및 Timestream API 파라미터
서브넷	이 DB 인스턴스와 연결할 vpc 서브넷입니다.	CLI: `vpc-subnet-ids` API: `VPCSubnetIds`
VPC 보안 그룹(방화벽)	DB 인스턴스에 연결할 보안 그룹입니다.	CLI: `vpc-security-group-ids` API: `VPCSecurityGroupIds`

Amazon Timestream for InfluxDB DB 인스턴스에 연결

DB 인스턴스에 연결하려면 먼저 DB 인스턴스를 생성해야 합니다. 자세한 내용은 DB 인스턴스 생성을 참조하세요. Amazon Timestream이 DB 인스턴스를 프로비저닝한 후 influxDB API, Influx CLI 또는 InfluxDB에 호환되는 클라이언트 또는 유틸리티를 사용하여 DB 인스턴스에 연결합니다.

주제

- Amazon Timestream for InfluxDB DB 인스턴스의 연결 정보 찾기
- 데이터베이스 인증 옵션
- 파라미터 그룹 작업

Amazon Timestream for InfluxDB DB 인스턴스의 연결 정보 찾기

DB 인스턴스의 연결 정보에는 엔드포인트, 포트, 사용자 이름, 암호 및 연산자 또는 모든 액세스 토큰과 같은 유효한 액세스 토큰이 포함됩니다. 예를 들어 InfluxDB DB 인스턴스의 경우 엔드포인트 값이 라고 가정합니다`influxdb1-123456789.us-east-1.timestream-influxdb.amazonaws.com`. 이 경우 포트 값은 8086이고 데이터베이스 사용자는 admin 입니다. 이 정보를 바탕으로 연결 문자열에 다음 값을 지정합니다.

- 호스트 또는 호스트 이름 또는 DNS 이름에 를 지정합니다`influxdb1-123456789.us-east-1.timestream-influxdb.amazonaws.com`.
- 포트의 경우 8086을 지정합니다.
- 사용자의 경우 관리자를 지정합니다.
- 암호의 경우 DB 인스턴스를 생성할 때 제공한 암호를 지정합니다.

> **Important**
>
> Timestream for InfluxDB Db 인스턴스를 생성하면 DBInstance 응답 객체의 일부로 가 수신됩니다`influxAuthParametersSecretArn`. 이렇게 하면 계정의 SecretsManager 보안 암호가 유지됩니다. InfluxDB DB 인스턴스를 사용할 수 있는 경우에만 채워집니다. 보안 암호에는 `CreateDbInstance` 프로세스 중에 제공된 유입 인증 파라미터가 포함되어 있습니다. 이 보안 암호updates/modifications/deletions에 대한 READONLY 복사본은 생성된 DB 인스턴스에 영향을 주지 않습니다. 이 보안 암호를 삭제해도 API 삭제된 보안 암호는 계속 참조됩니다.

엔드포인트는 DB 인스턴스마다 고유하며 포트 및 사용자 값이 다를 수 있습니다. DB 인스턴스에 연결하려면 Influx CLI, Influx API또는 InfluxDB 와 호환되는 모든 클라이언트를 사용할 수 있습니다.

DB 인스턴스의 연결 정보를 찾으려면 AWS 관리 콘솔을 사용합니다. AWS 명령줄 인터페이스(AWS CLI) `describe-db-instances` 명령 또는 Timestream-influxdb API `GetDBInstance` 작업을 사용할 수도 있습니다.

사용 AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon Timestream 콘솔을 엽니다.
2. 탐색 창에서 InfluxDB 데이터베이스를 선택하여 DB 인스턴스 목록을 표시합니다.
3. DB 인스턴스의 이름을 선택하여 세부 정보를 표시합니다.
4. 요약 섹션 에서 엔드포인트를 복사합니다. 또한 포트 번호를 적어 둡니다. DB 인스턴스에 연결하려면 엔드포인트와 포트 번호가 모두 필요합니다.

사용자 이름과 암호 정보를 찾아야 하는 경우 구성 세부 정보 탭을 선택하고 를 선택하여 Secrets Manager`influxAuthParametersSecretArn`에 액세스합니다.

사용 CLI

- 를 사용하여 InfluxDB DB 인스턴스의 연결 정보를 찾으려면 명령을 AWS CLI호출합니다 get-db-instance. 호출에서 DB 인스턴스 ID, 엔드포인트, 포트 및 influxAuthParameters보안 암호를 쿼리합니다.

  Linux, macOS, Unix의 경우:

  ```
  aws timestream-influxdb get-db-instance --identifier id \
   --query "[name,endpoint,influxAuthParametersSecretArn]"
  ```

  Windows의 경우:

  ```
  aws timestream-influxdb get-db-instance --identifier id \
   --query "[name,endpoint,influxAuthParametersSecretArn]"
  ```

  다음과 유사하게 출력되어야 합니다. 사용자 이름 정보에 액세스하려면 를 확인해야 합니다`InfluxAuthParameterSecret`.

  ```
  [
      [
          "mydb",
          "mydb-123456789012.us-east-1.timestream-influxdb.amazonaws.com",
          8086,
      ]
  ]
  ```

액세스 토큰 생성

이 정보를 사용하면 인스턴스에 연결하여 액세스 토큰을 검색하거나 생성할 수 있습니다. 이를 달성하는 방법에는 여러 가지가 있습니다.

사용 CLI

1. 아직 다운로드하지 않았다면 influx CLI를 다운로드, 설치 및 구성합니다.
2. Influx 구성을 구성할 때 CLI 를 사용하여 인증`--username-password`합니다.

```
influx config create --config-name YOUR_CONFIG_NAME  --host-url "https://
yourinstance.timestream-influxdb.amazonaws.com:8086"  --org yourorg --username-
password admin --active
```

3. influx 인증 생성 명령을 사용하여 운영자 토큰을 다시 생성합니다. 이 프로세스를 수행하면 이전 운영자 토큰이 무효화된다는 점을 고려하세요.

```
influx auth create --org kronos --operator
```

4. 운영자 토큰이 있으면 influx auth list 명령을 사용하여 모든 토큰을 볼 수 있습니다. influx 인증 생성 명령을 사용하여 모든 액세스 토큰을 생성할 수 있습니다.

> **Important**
>
> 이 단계를 수행하여 먼저 운영자 토큰을 얻은 다음 InfluxDB API 또는 를 사용하여 새 토큰을 생성할 수 있어야 합니다CLI.

Influx UI 사용

1. 생성된 엔드포인트를 사용하여 Timestream for InfluxDB 인스턴스로 이동하여 InfluxDB UI에 로그인하고 액세스합니다. InfluxDB DB 인스턴스를 생성하는 데 사용되는 사용자 이름과 암호를 사용해야 합니다. 의 응답 객체에 `influxAuthParametersSecretArn` 지정된 에서 이 정보를 검색할 수 있습니다`CreateDbInstance`.

   또는 InfluxDB용 Timestream 관리 콘솔에서 InfluxUI를 열 수 있습니다. InfluxDB

   a. 에 로그인 AWS Management Console 하고 에서 Amazon Timestream for InfluxDB 콘솔을 엽니다https://console.aws.amazon.com/timestream/.
   b. Amazon Timestream for InfluxDB 콘솔의 오른쪽 상단에서 DB 인스턴스를 생성한 AWS 리전을 선택합니다.
   c. 데이터베이스 목록에서 InfluxDB 인스턴스의 이름을 선택하여 세부 정보를 표시합니다. 오른쪽 상단 모서리에서 Open Influx UI 를 선택합니다.
2. InfluxUI 로그인한 후 왼쪽 탐색 모음을 사용하여 데이터 로드로 이동한 다음 API 토큰으로 이동합니다.
3. +GENERATEAPITOKEN를 선택하고 모든 액세스 토큰 API을 선택합니다.
4. API 토큰에 대한 설명을 입력하고 를 선택합니다SAVE.

5. 생성된 토큰을 복사하고 안전한 보관을 위해 저장합니다.

> **Important**
>
> InfluxUI 에서 토큰을 생성할 때 새로 생성된 토큰은 한 번만 표시됩니다. 복사하지 않으면 다시 생성해야 하므로 복사해야 합니다.

InfluxDB 사용 API

- 요청 메서드를 사용하여 InfluxDB API `/api/v2/authorizations` 엔드포인트에 POST 요청을 보냅니다.

  요청에 다음을 포함합니다.

  a. 헤더:

     i. 권한 부여: 토큰 <INFLUX_OPERATOR_TOKEN>

     ii. 콘텐츠 유형: application/json

  b. 요청 본문: 다음 속성을 가진 JSON 본문:

     i. 상태: “활성”

     ii. 설명: API 토큰 설명

     iii. orgID : InfluxDB 조직 ID

     iv. 권한: 각 객체가 InfluxDB 리소스 유형 또는 특정 리소스에 대한 권한을 나타내는 객체 배열입니다. 각 권한에는 다음 속성이 포함됩니다.

        A. 작업: “읽기” 또는 “쓰기”

        B. 리소스: 권한을 부여할 InfluxDB 리소스를 나타내는 JSON 객체입니다. 각 리소스에는 최소한 다음과 같은 속성이 포함됩니다. orgID : InfluxDB 조직 ID

        C. 유형: 리소스 유형. 존재하는 InfluxDB 리소스 유형에 대한 자세한 내용은 the /api/v2/resources 엔드포인트를 사용합니다.

다음 예제에서는 `curl` 및 InfluxDB를 사용하여 모든 액세스 토큰을 API 생성합니다.

```
export INFLUX_HOST=https://influxdb1-123456789.us-east-1.timestream-
influxdb.amazonaws.com
```

```
export INFLUX_ORG_ID=<YOUR_INFLUXDB_ORG_ID>
export INFLUX_TOKEN=<YOUR_INFLUXDB_OPERATOR_TOKEN>

curl --request POST \
"$INFLUX_HOST/api/v2/authorizations" \
  --header "Authorization: Token $INFLUX_TOKEN" \
  --header "Content-Type: text/plain; charset=utf-8" \
  --data '{
    "status": "active",
    "description": "All access token for get started tutorial",
    "orgID": "'"$INFLUX_ORG_ID"'",
    "permissions": [
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "authorizations"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "authorizations"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "buckets"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "buckets"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "dashboards"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "dashboards"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "orgs"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "orgs"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "sources"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "sources"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "tasks"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "tasks"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "telegrafs"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "telegrafs"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "users"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "users"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "variables"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "variables"}},
```

```
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"scrapers"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"scrapers"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"secrets"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"secrets"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"labels"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"labels"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "views"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"views"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"documents"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"documents"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationRules"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationRules"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationEndpoints"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notificationEndpoints"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"checks"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"checks"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "dbrp"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type": "dbrp"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notebooks"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"notebooks"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"annotations"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"annotations"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
"remotes"}},
```

```
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "remotes"}},
      {"action": "read", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "replications"}},
      {"action": "write", "resource": {"orgID": "'"$INFLUX_ORG_ID"'", "type":
 "replications"}}
    ]
  }
'
```

데이터베이스 인증 옵션

Amazon Timestream for InfluxDB는 데이터베이스 사용자를 인증하는 다음과 같은 방법을 지원합니다.

- 암호 인증 – DB 인스턴스가 모든 사용자 계정 관리 작업을 수행합니다. InfluxUI, Influx CLI 또는 influx 를 사용하여 사용자를 생성하고, 암호를 지정하고, 토큰을 관리합니다API.
- 토큰 인증 - DB 인스턴스가 모든 사용자 계정 관리를 수행합니다. Influx 및 Influx 를 사용하여 사용자 생성, 암호 지정 CLI및 운영자 토큰을 관리할 수 있습니다API.

암호화된 연결

애플리케이션에서 Secure Socket Layer(SSL) 또는 Transport Layer Security(TLS)를 사용하여 DB 인스턴스에 대한 연결을 암호화할 수 있습니다. InfluxDB와 Kronos 서비스에서 생성 및 관리하는 애플리케이션 간의 TLS 핸드셰이크에 필요한 인증서입니다. 인증서가 갱신되면 사용자 개입 없이 인스턴스가 최신 버전으로 자동으로 업데이트됩니다.

파라미터 그룹 작업

[데이터베이스 파라미터(Database parameters)]에서 데이터베이스 구성 방법을 지정합니다. 예를 들어 데이터베이스 파라미터는 메모리를 비롯하여 데이터베이스에 할당할 리소스의 양을 지정할 수 있습니다.

DB 인스턴스를 파라미터 그룹과 연결하여 데이터베이스 구성을 관리합니다. Amazon Timestream for InfluxDB는 기본 설정으로 파라미터 그룹을 정의합니다. 맞춤형 설정으로 자신만의 파라미터 그룹을 정의할 수 있습니다.

파라미터 그룹 개요

DB 파라미터 그룹은 하나 이상의 DB 인스턴스에 적용되는 엔진 구성 값의 컨테이너 역할을 합니다.

주제

기본 및 사용자 지정 파라미터 그룹

DB 인스턴스는 DB 파라미터 그룹을 사용합니다. 다음 섹션에서는 DB 인스턴스 파라미터 그룹 구성 및 관리에 대해 설명합니다.

DB 파라미터 그룹 생성

AWS Management Console, AWS Command Line Interface또는 Timestream 을 사용하여 새 DB 파라미터 그룹을 생성할 수 있습니다API.

DB 파라미터 그룹 이름에는 다음과 같은 제한이 적용됩니다.

- 이름은 1~255자의 문자, 숫자 또는 하이픈으로 구성되어야 합니다.
- 기본 파라미터 그룹 이름에는 마침표(예: `default.InfluxDB.2.7`)가 포함될 수 있습니다. 하지만 사용자 지정 파라미터 그룹 이름에는 마침표를 포함할 수 없습니다.
- 첫 번째 자리는 문자여야 합니다.
- 이름은 “dbpg-”로 시작할 수 없습니다.
- 이름은 하이픈으로 끝나거나 2개 연속 하이픈을 포함할 수 없습니다.
- DB 파라미터 그룹을 지정하지 않고 DB 인스턴스를 생성하는 경우 DB 인스턴스는 influxDB 엔진 기본값을 사용합니다.

기본 DB 파라미터 그룹의 파라미터 설정은 수정할 수 없습니다. 대신에 다음 작업을 할 수 있습니다.

1. 새 파라미터 그룹을 생성해야 합니다.
2. 원하는 파라미터의 설정을 변경합니다. 파라미터 그룹에서 모든 DB 엔진 파라미터를 수정할 수 있는 것은 아닙니다.
3. 사용자 지정 파라미터 그룹을 사용하도록 DB 인스턴스를 업데이트합니다. DB 인스턴스 업데이트에 대한 자세한 내용은 섹션을 참조하세요DB 인스턴스 업데이트.

> **Note**
>
> 사용자 지정 파라미터 그룹을 사용하도록 DB 인스턴스를 수정하고 DB 인스턴스를 시작하는 경우 Amazon Timestream for InfluxDB는 시작 프로세스의 일부로 DB 인스턴스를 자동으로 재부팅합니다.
> 현재 사용자 지정 파라미터 그룹을 생성한 후에는 수정할 수 없습니다. 파라미터를 변경해야 하는 경우 새 사용자 지정 파라미터 그룹을 생성하고 이 구성 변경이 필요한 인스턴스에 할당해야 합니다. 기존 DB 인스턴스를 업데이트하여 새 파라미터 그룹을 할당하면 항상 즉시 적용되고 인스턴스가 재부팅됩니다.

정적 및 동적 DB 인스턴스 파라미터

InfluxDB DB 인스턴스 파라미터는 항상 정적입니다. 다음과 같이 작동합니다.

정적 파라미터를 변경하고 DB 파라미터 그룹을 저장한 다음 인스턴스에 할당하면 인스턴스가 재부팅된 후 파라미터 변경이 자동으로 적용됩니다.

새 DB 파라미터 그룹을 DB 인스턴스와 연결하면 Timestream은 DB 인스턴스가 재부팅된 후에만 수정된 정적 파라미터를 적용합니다. 현재 유일한 옵션은 즉시 적용됩니다.

DB 파라미터 그룹 변경에 대한 자세한 내용은 [g26][/g26][g25][/g25] 단원을 참조하세요.

지원되는 파라미터 및 파라미터 값

DB 인스턴스에 지원되는 파라미터를 확인하려면 DB 인스턴스에서 사용하는 DB 파라미터 그룹의 파라미터를 확인합니다. 자세한 내용은 DB 파라미터 그룹의 파라미터 값 보기를 참조하세요.

InfluxDB의 오픈 소스 버전에서 지원되는 모든 파라미터에 대한 자세한 내용은 InfluxDB 구성 옵션 섹션을 참조하세요. 현재 다음 InfluxDB 파라미터만 수정할 수 있습니다.

파라미터	설명	기본값	값	유효 범위	참고
flux-log-enabled	Flux 쿼리에 대한 세부 로그를 표시하는 옵션 포함	FALSE	true, false	N/A	
로그 수준	로그 출력 수준. InfluxDB	info	디버그, 정보, 오류	N/A	

파라미터	설명	기본값	값	유효 범위	참고
	는 심각도 수준이 지정된 수준 이상인 로그 항목을 출력합니다.				
작업 없음	동시에 실행할 수 있는 쿼리 수입니다. 0으로 설정하면 동시 쿼리 수에 제한이 없습니다.	FALSE	true, false	N/A	
쿼리 동시성	작업 스케줄러를 비활성화합니다. 문제가 있는 태스크로 인해 InfluxDB가 시작되지 않는 경우 이 옵션을 사용하여 태스크를 예약하거나 실행하지 않고 InfluxDB를 시작합니다.	1024		N/A	

파라미터	설명	기본값	값	유효 범위	참고
query-queue-size	실행 대기열에 허용되는 최대 쿼리 수입니다. 대기열 한도에 도달하면 새 쿼리가 거부됩니다. 0으로 설정하면 대기열의 쿼리 수에 제한이 없습니다.	1024		N/A	
추적 유형	InfluxDB에서 추적을 활성화하고 추적 유형을 지정합니다. 추적은 기본적으로 비활성화되어 있습니다.	""	로그, jaeger	N/A	
지표 비활성화됨	내부 InfluxDB 지표를 노출하는 HTTP / 지표 엔드포인트를 비활성화합니다. InfluxDB	FALSE		N/A	

파라미터	설명	기본값	값	유효 범위	참고
http-idle-timeout	새 요청을 기다리는 동안 서버가 설정된 연결을 유지하는 최대 기간입니다. 제한 시간 없이 `0` 를 로 설정합니다.	3m0s	단위 `hours`, `minutes`, `seconds`, 의 기간입니다`milliseconds` . 예시: `durationType=minutes,value=10`	시간: -최소: 0 -최대: 256205 분: -최소: 0 -최대: 15372286 초: -최소: 0 -최대: 922337203 밀리초: -최소: 0 -최대: 922337203685	

파라미터	설명	기본값	값	유효 범위	참고
http-read-header-timeout	서버가 새 요청에 대한 HTTP 헤더를 읽으려고 시도하는 최대 기간입니다. 제한 시간 없이 `0` 를 로 설정합니다.	10초	단위 `hours`, `minutes`, `seconds`, 의 기간입니다`milliseconds` . 예시: `durationType=minutes,value=10`	시간: -최소: 0 -최대: 256205 분: -최소: 0 -최대: 15372286 초: -최소: 0 -최대: 922337203 밀리초: -최소: 0 -최대: 922337203685	

파라미터	설명	기본값	값	유효 범위	참고
http-read-timeout	서버가 새 요청 전체를 읽으려고 시도해야 하는 최대 기간입니다. 제한 시간 없이 `0` 를 로 설정합니다.	0	단위 `hours`, `minutes`, `seconds`, 의 기간입니다`milliseconds` . 예시: `durationType=minutes,value=10`	시간: -최소: 0 -최대: 256205 분: -최소: 0 -최대: 15372286 초: -최소: 0 -최대: 922337203 밀리초: -최소: 0 -최대: 922337203685	

파라미터	설명	기본값	값	유효 범위	참고
http-write-timeout	서버가 쓰기 요청을 처리하고 응답하는 데 소비해야 하는 최대 기간입니다. 제한 시간 없이 `0` 를 로 설정합니다.	0	단위 `hours`, `minutes`, `seconds`, 의 기간입니다`milliseconds` . 예시: `durationType=minutes,value=10`	시간: -최소: 0 -최대: 256205 분: -최소: 0 -최대: 15372286 초: -최소: 0 -최대: 922337203 밀리초: -최소: 0 -최대: 922337203685	
influxql-max-select-buckets	`SELECT` 문이 생성할 수 있는 시간 버킷별 최대 그룹 수입니다. 는 버킷 수를 무제한으로 `0` 허용합니다.	0	Long	최솟값: 0 최대: 9,223,372,036,854,775,807	

파라미터	설명	기본값	값	유효 범위	참고
influxql-max-select-point	SELECT 문이 처리할 수 있는 최대 포인트 수입니다. 는 무제한 포인트 수를 0 허용합니다. InfluxDB는 매초마다 포인트 수를 확인합니다(최대를 초과하는 쿼리는 즉시 중단되지 않음).	0	Long	최솟값: 0 최대: 9,223,372,036,854,775,807	
influxql-max-select-series	SELECT 문이 반환할 수 있는 최대 시리즈 수입니다. 는 무제한의 시리즈 수를 0 허용합니다.	0	Long	최솟값: 0 최대: 9,223,372,036,854,775,807	

파라미터	설명	기본값	값	유효 범위	참고
pprof 비활성화됨	/debug/pprof HTTP 엔드포인트를 비활성화합니다. 이 엔드포인트는 런타임 프로파일링 데이터를 제공하며 디버깅 시 유용할 수 있습니다.	FALSE	불	N/A	
query-initial-memory-bytes	쿼리에 할당된 초기 메모리 바이트입니다.	0	Long	최솟값: 0 최대: query-memory-bytes	
query-max-memory-bytes	쿼리에 허용되는 최대 총 메모리 바이트 수입니다.	0	Long	최솟값: 0 최대: 9,223,372,036,854,775,807	
query-memory-bytes	새로 생성된 사용자 세션에 대한 라이브 시간(TTL)을 분 단위로 지정합니다.	0	Long	최솟값: 0 최대: 2,147,483,647	보다 크거나 같아야 합니다 query-initial-memory-bytes.

파라미터	설명	기본값	값	유효 범위	참고
세션 길이	새로 생성된 사용자 세션에 대한 라이브 시간(TTL)을 분 단위로 지정합니다.	60	Integer	최솟값: 0 최대: 2880	
session-renew-disabled	각 요청TTL에서 사용자 세션의 자동 확장을 비활성화합니다. 기본적으로 모든 요청은 세션의 만료 시간을 지금부터 5분으로 설정합니다. 비활성화하면 세션은 지정된 세션 길이 후에 만료되고 최근에 활성화되었더라도 사용자가 로그인 페이지로 리디렉션됩니다.	FALSE	불	N/A	

파라미터	설명	기본값	값	유효 범위	참고
storage-cache-max-memory-크기	쓰기 거부를 시작하기 전에 샤드의 캐시가 도달할 수 있는 최대 크기(바이트)입니다.	1073741824	Long	최솟값: 0 최대: 549755813888	인스턴스의 총 메모리 용량보다 작아야 합니다. 총 메모리 용량의 15% 미만으로 설정하는 것이 좋습니다.
storage-cache-snapshot-memory-크기	스토리지 엔진이 캐시를 스냅샷하고 TSM 파일에 기록하여 더 많은 메모리를 사용할 수 있도록 하는 크기(바이트)입니다.	26214400	Long	최솟값: 0 최대: 549755813888	storage-cache-max-memory-크기보다 작아야 합니다.

파라미터	설명	기본값	값	유효 범위	참고
storage-cache-snapshot-write-cold-duration	샤드가 쓰기 또는 삭제를 수신하지 못한 경우 스토리지 엔진이 캐시를 스냅샷 처리하고 새 TSM 파일에 쓰는 기간입니다.	1,000만	단위 hours, minutes, seconds, 의 기간입니다milliseconds . 예시: durationType=minutes,value=10	시간: -최소: 0 -최대: 256205 분: -최소: 0 -최대: 15372286 초: -최소: 0 -최대: 922337203 밀리초: -최소: 0 -최대: 922337203685	

파라미터	설명	기본값	값	유효 범위	참고
storage-compact-full-write-cold-duration	스토리지 엔진이 쓰기 또는 삭제를 수신하지 못한 경우 샤드에 있는 모든 TSM 파일을 압축하는 기간입니다.	4시간 0분0초	단위 hours, minutes, seconds, 의 기간입니다milliseconds . 예시: durationType=minutes,value=10	시간: -최소: 0 -최대: 256205 분: -최소: 0 -최대: 15372286 초: -최소: 0 -최대: 922337203 밀리초: -최소: 0 -최대: 922337203685	
storage-compact-throughput-burst	TSM 압축이 디스크에 쓸 수 있는 속도 제한(초당 바이트)입니다.	50331648	Long	최솟값: 0 최대: 9,223,372,036,854,775,807	

파라미터	설명	기본값	값	유효 범위	참고
storage-max-concurrent-compactions	동시에 실행할 수 있는 전체 및 수준 압축의 최대 수입니다. 의 값은 런타임에 `runtime.GOMAXPROCS(0)` 사용된 의 50%를 `0` 생성합니다. 0보다 큰 숫자는 압축을 해당 값으로 제한합니다. 이 설정은 캐시 스냅샷 생성에는 적용되지 않습니다.	0	Integer	최솟값: 0 최대: 64	

파라미터	설명	기본값	값	유효 범위	참고
storage-max-index-log-file-size, 파일 크기	인덱스 미리 쓰기 로그() 파일이 인덱스 파일로 압축되는 크기(바이트WAL) 입니다. 크기가 작으면 로그 파일이 더 빠르게 압축되어 쓰기 처리량을 희생하여 힙 사용량이 줄어듭니다.	1048576	Long	최솟값: 0 최대: 9,223,372,036,854,775,807	
storage-no-validate-field-크기	수신 쓰기 요청에 대한 필드 크기 검증을 건너뜁니다.	FALSE	불	N/A	

파라미터	설명	기본값	값	유효 범위	참고
storage-retention-check-interval	보존 정책 적용 검사 간격.	3,000만	단위 hours, minutes, seconds, 의 기간입니다milliseconds . 예시: durationType=minutes,value=10	N/A	시간: -최소: 0 -최대: 256205 분: -최소: 0 -최대: 15372286 초: -최소: 0 -최대: 922337203 밀리초: -최소: 0 -최대: 9223372036685
storage-series-file-max-concurrent-snapshot-compactions	데이터베이스의 모든 시리즈 파티션에서 동시에 실행할 수 있는 스냅샷 압축의 최대 수입니다.	0	Integer	최솟값: 0 최대: 64	

파라미터	설명	기본값	값	유효 범위	참고
storage-series-id-set-cache-size	이전에 계산된 시리즈 결과를 저장하는 데 TSI 인덱스에 사용되는 내부 캐시의 크기입니다. 동일한 태그 키/값 예측 변수가 있는 후속 쿼리가 실행될 때 다시 계산할 필요 없이 캐시된 결과가 빠르게 반환됩니다. 이 값을 로 설정0하면 캐시가 비활성화되고 쿼리 성능이 저하될 수 있습니다.	100	Long	최솟값: 0 최대: 9,223,372,036,854,775,807	
storage-wal-max-concurrent-쓰기	동시에 시도하기 위해 WAL 디렉터리에 쓰는 최대 수입니다.	0	Integer	최솟값: 0 최대: 256	

파라미터	설명	기본값	값	유효 범위	참고
storage-wal-max-write-지연	WAL 디렉터리에 대한 동시 활성 쓰기의 최대 수가 충족될 때 WAL 디렉터리에 대한 쓰기 요청이 대기하는 최대 시간입니다. 를 로 설정0하여 제한 시간을 비활성화합니다.	10m	단위 hours, minutes, seconds, 의 기간입니다milliseconds . 예시: durationType=minutes,value=10	시간: -최소: 0 -최대: 256205 분: -최소: 0 -최대: 15372286 초: -최소: 0 -최대: 922337203 밀리초: -최소: 0 -최대: 922337203685	
ui 비활성화됨	InfluxDB 사용자 인터페이스(UI)를 비활성화합니다. UI는 기본적으로 활성화됩니다.	FALSE	불	N/A	

파라미터 그룹에 파라미터를 잘못 설정하면 성능 저하나 시스템 불안정 등의 의도하지 않은 부작용이 있을 수 있습니다. 데이터베이스 파라미터를 수정할 때는 항상 주의해야 합니다. 테스트 DB 인스턴스에서 파라미터 그룹 설정 변경 사항을 프로덕션 DB 인스턴스에 적용하기 전에 이러한 파라미터 그룹 변경 사항을 시도해 보십시오.

DB 파라미터 그룹 작업

DB 인스턴스는 DB 파라미터 그룹을 사용합니다. 다음 섹션에서는 DB 인스턴스 파라미터 그룹 구성 및 관리에 대해 설명합니다.

주제

- DB 파라미터 그룹 생성
- DB 파라미터 그룹과 DB 인스턴스 연결
- DB 파라미터 그룹 나열
- DB 파라미터 그룹의 파라미터 값 보기

DB 파라미터 그룹 생성

사용 AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon Timestream for InfluxDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. [Create parameter group]을 선택합니다.
4. 그룹 이름 상자에 새 DB 파라미터 그룹의 이름을 입력합니다.
5. 설명 상자에 새 DB 파라미터 그룹에 대한 설명을 입력합니다.
6. 원하는 값을 수정하고 적용할 파라미터를 선택합니다. 지원되는 파라미터에 대한 자세한 내용은 섹션을 참조하세요지원되는 파라미터 및 파라미터 값.
7. 저장(Save)을 선택합니다.

사용 AWS Command Line Interface

- 를 사용하여 DB 파라미터 그룹을 생성하려면 다음 파라미터로 `create-db-parameter-group` 명령을 AWS CLI호출합니다.

```
--db-parameter-group-name <value>
--description <value>
```

```
--endpoint_url <value>
--region <value>
--parameters (list) (string)
```

Example 예

각 설정에 대한 자세한 내용은 DB 인스턴스에 대한 설정 단원을 참조하세요. 이 예제에서는 기본 엔진 구성을 사용합니다.

```
aws timestream-influxdb create-db-parameter-group
    --db-parameter-group-name YOUR_PARAM_GROUP_NAME\
    --endpoint-url YOUR_ENDPOINT
    --region YOUR_REGION \
    --parameters
 "InfluxDBv2={logLevel=debug,queryConcurrency=10,metricsDisabled=true}" \" \
    --debug
```

DB 파라미터 그룹과 DB 인스턴스 연결

사용자 지정 설정을 사용하여 사용자의 DB 파라미터 그룹을 생성할 수 있습니다. AWS Management Console, AWS Command Line Interface또는 Timestream-InfluxDB 를 사용하여 DB 파라미터 그룹을 DB 인스턴스와 연결할 수 있습니다API. DB 인스턴스를 생성하거나 수정할 때 이 작업을 수행할 수 있습니다.

DB 파라미터 그룹 생성에 대한 자세한 내용은 DB 파라미터 그룹 생성 단원을 참조하세요. DB 인스턴스 생성에 대한 자세한 내용은 DB 인스턴스 생성 단원을 참조하십시오. DB 인스턴스 수정에 대한 자세한 내용은 섹션을 참조하세요DB 인스턴스 업데이트.

> **Note**
>
> 새 DB 파라미터 그룹을 DB 인스턴스와 연결하면 DB 인스턴스가 재부팅된 후에만 수정된 정적 파라미터가 적용됩니다. 현재는 즉시 적용만 지원됩니다. InfluxDB의 Timestream은 정적 파라미터만 지원합니다.

사용 AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon Timestream for InfluxDB 콘솔을 엽니다.
2. 탐색 창에서 InfluxDB 데이터베이스 를 선택한 다음 수정하려는 DB 인스턴스를 선택합니다.

3. 업데이트를 선택합니다. DB 인스턴스 업데이트 페이지가 나타납니다.
4. DB 파라미터 그룹 설정을 변경합니다.
5. [Continue]를 수정 사항을 요약한 내용을 확인합니다.
6. 현재 즉시 적용만 지원됩니다. 이 옵션은 DB 인스턴스를 재부팅하기 때문에 경우에 따라 중단이 발생할 수 있습니다.
7. 확인 페이지에서 변경 내용을 검토합니다. 올바른 경우 DB 인스턴스 업데이트를 선택하여 변경 사항을 저장하고 적용합니다. 또는 뒤로를 선택하여 변경 내용을 편집하거나 취소를 선택하여 변경 내용을 취소합니다.

사용 AWS Command Line Interface

Linux, macOS, Unix의 경우:

```
aws timestream-influxdb update-db-instance
--identifier YOUR_DB_INSTANCE_ID \
--region YOUR_REGION \
--db-parameter-group-identifier YOUR_PARAM_GROUP_ID \
--log-delivery-configuration "{\"s3Configuration\": {\"bucketName\":
 \"${LOGGING_BUCKET}\", \"enabled\": false }}"
```

Windows의 경우:

```
aws timestream-influxdb update-db-instance
--identifier YOUR_DB_INSTANCE_ID ^
--region YOUR_REGION ^
--db-parameter-group-identifier YOUR_PARAM_GROUP_ID ^
--log-delivery-configuration "{\"s3Configuration\": {\"bucketName\":
 \"${LOGGING_BUCKET}\", \"enabled\": false }}"
```

DB 파라미터 그룹 나열

AWS 계정에 대해 생성한 DB 파라미터 그룹을 나열할 수 있습니다.

사용 AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon Timestream for InfluxDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.

3. DB 파라미터 그룹이 목록에 나타납니다.

사용 AWS Command Line Interface

AWS 계정의 모든 DB 파라미터 그룹을 나열하려면 `list-db-parameter-groups` 명령을 사용합니다 AWS Command Line Interface .

```
aws timestream-influxdb list-db-parameter-groups --region region
```

AWS 계정의 특정 DB 파라미터 그룹을 반환하려면 `get-db-parameter-group` 명령을 AWS Command Line Interface 사용합니다.

```
aws timestream-influxdb get-db-parameter-group --region region --identifier identifier
```

DB 파라미터 그룹의 파라미터 값 보기

DB 파라미터 그룹의 모든 파라미터와 해당 값 목록을 가져올 수 있습니다.

사용 AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon Timestream for InfluxDB 콘솔을 엽니다.
2. 탐색 창에서 파라미터 그룹을 선택합니다.
3. DB 파라미터 그룹이 목록에 나타납니다.
4. 파라미터 그룹의 이름을 선택하여 파라미터 목록을 봅니다.

사용 AWS Command Line Interface

DB 파라미터 그룹의 파라미터 값을 보려면 다음 필수 파라미터와 AWS Command Line Interface `get-db-parameters` 함께 명령을 사용합니다.

```
--db-parameter-group-name
```

사용 API

DB 파라미터 그룹의 파라미터 값을 보려면 다음 필수 파라미터와 함께 Timestream API `GetDBParameters` 명령을 사용합니다.

```
DBParameterGroupName
```

DB 인스턴스 관리

이 섹션에서는 최적의 성능, 가용성 및 모니터링 기능을 보장하기 위해 Timestream for InfluxDB 인스턴스 관리의 다양한 측면을 다룹니다. 데이터베이스 인스턴스의 구성 업데이트, 다중 AZ 배포 및 장애 조치 프로세스 처리에 대한 지침을 제공합니다. 또한 데이터베이스 인스턴스를 삭제하고 InfluxDB 인스턴스에 대한 로그 보기를 설정하는 방법을 설명합니다.

주제

DB 인스턴스 업데이트

InfluxDB 인스턴스용 Timestream의 다음 구성 파라미터를 업데이트할 수 있습니다.

- 인스턴스 클래스
- 배포 유형
- Parameter Group
- 로그 전송 구성

> **Important**
>
> 특히 데이터베이스 버전을 업그레이드할 때 프로덕션 인스턴스를 수정하기 전에 테스트 인스턴스의 모든 변경 사항을 테스트하여 영향을 이해하는 것이 좋습니다. 설정을 업데이트하기 전에 데이터베이스 및 애플리케이션에 미치는 영향을 검토합니다. 일부 수정을 수행하려면 DB 인스턴스 재부팅이 필요하므로 가동 중지가 발생합니다.

사용 AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon Timestream for InfluxDB 콘솔을 엽니다.
2. 탐색 창에서 InfluxDB 데이터베이스 를 선택한 다음 수정하려는 DB 인스턴스를 선택합니다.

3. 수정을 선택합니다.
4. DB 인스턴스 수정 페이지에서 원하는 내용을 변경합니다.
5. 계속해서 수정 사항을 요약한 내용을 확인합니다.
6. Next(다음)를 선택합니다.
7. 변경 사항을 검토합니다.
8. 인스턴스 수정을 선택하여 변경 사항을 적용합니다.

> **Note**
>
> 이러한 수정을 수행하려면 Influx DB 인스턴스를 재부팅해야 하며 경우에 따라 중단이 발생할 수 있습니다.

사용 AWS Command Line Interface

를 사용하여 DB 인스턴스를 업데이트하려면 `update-db-instance` 명령을 AWS Command Line Interface호출합니다. DB 인스턴스 식별자와 수정하려는 옵션 값을 지정합니다. 각 옵션에 대한 자세한 내용은 DB 인스턴스에 대한 설정 단원을 참조하십시오.

Example 예

다음 코드는 다른 dbparameter 그룹을 설정하여 mydbinstance를 수정합니다. 변경이 바로 적용됩니다.

Linux, macOS 또는 Unix의 경우는 다음과 같습니다.

```
aws timestream-influxdb update-db-instance \
    --identifier mydbinstance \
    --db-instance-type desired-instance-type \
    --deployment-type desired-deployment-type \
    --db-parameter-group-name newparamgroup \
    --port 8086
```

Windows의 경우:

```
aws timestream-influxdb update-db-instance ^
    --identifier mydbinstance ^
```

```
    --db-instance-type desired-instance-type ^
    --deployment-type desired-deployment-type ^
    --db-parameter-group-name newparamgroup
    --port 8086
```

DB 인스턴스 유지 관리

주기적으로 Amazon Timestream for InfluxDB는 Amazon Timestream for InfluxDB 리소스에 대한 유지 관리를 수행합니다. 유지 관리에는 DB 인스턴스의 다음 리소스에 대한 업데이트가 가장 자주 포함됩니다.

- 기본 하드웨어
- 기본 운영 체제(OS)
- 데이터베이스 엔진 버전

운영 체제 업데이트는 보안상 가장 빈번하게 발생하며

일부 유지 관리 항목에서는 Amazon Timestream for InfluxDB가 잠시 동안 DB 인스턴스를 오프라인으로 전환해야 합니다. 리소스가 오프라인 상태에 있어야 하는 유지 관리 항목에는 필수 운영 체제 또는 데이터베이스 패칭이 포함됩니다. 이때 보안 및 인스턴스 안정성과 관련된 패치에 한해 필수 패치 작업으로 자동 예약됩니다. 이러한 패치 작업은 드물게 발생하며 일반적인 빈도는 몇 개월에 한 번입니다. 대부분 유지 관리 기간의 일부만 필요합니다.

- 유지 관리 기간은 인스턴스가 호스팅되는 리전의 현지 시간으로 매일 오전 12시에서 오전 4시 사이에 이루어지도록 구성됩니다.
- 고객 리소스는 일주일에 한 번 7개의 유지 관리 기간 중 하나에 패치될 수 있습니다.

DB 인스턴스 삭제

DB 인스턴스를 삭제하면 인스턴스 복구 가능성 및 스냅샷 가용성에 영향을 미칩니다. 다음 문제를 고려하세요.

- InfluxDB 리소스에 대한 모든 Timestream을 삭제하려면 DB 인스턴스 리소스에 청구 요금이 발생합니다.
- DB 인스턴스의 상태가 삭제되는 경우 해당 CA 인증서 값은 InfluxDB 콘솔용 Timestream 또는 AWS Command Line Interface 명령 또는 Timestream API 작업용 출력에 표시되지 않습니다.

- DB 인스턴스를 삭제하는 데 필요한 시간은 삭제되는 데이터의 양과 최종 스냅샷을 찍는지 여부에 따라 달라집니다.

AWS Management Console, AWS Command Line Interface또는 Timestream 을 사용하여 DB 인스턴스를 삭제할 수 있습니다API. DB 인스턴스의 이름을 제공해야 합니다.

사용 AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon Timestream for InfluxDB 콘솔을 엽니다.
2. 탐색 창에서 InfluxDB 데이터베이스 를 선택한 다음 삭제할 DB 인스턴스를 선택합니다.
3. Delete(삭제)를 선택합니다.
4. 상자에 확인을 입력합니다.
5. Delete(삭제)를 선택합니다.

사용 AWS Command Line Interface

계정에서 DB 인스턴스IDs의 인스턴스를 찾으려면 `list-db-instances` 명령을 호출합니다.

```
aws timestream-influxdb list-db-instances \
--endpoint-url YOUR_ENDPOINT \
--region YOUR_REGION
```

를 사용하여 DB 인스턴스를 삭제하려면 다음 옵션을 사용하여 `delete-db-instance` 명령을 AWS CLI호출합니다.

```
aws timestream-influxdb list-db-instances \
--identifier YOUR_DB_INSTANCE \
```

Example 예

Linux, macOS, Unix의 경우:

```
aws timestream-influxdb delete-db-instance \
    --identifier mydbinstance
```

Windows의 경우:

```
aws timestream-influxdb delete-db-instance ^
```

```
    --identifier mydbinstance
```

다중 AZ DB 인스턴스 배포

Amazon Timestream for InfluxDB는 단일 대기 DB 인스턴스와 함께 다중 AZ 배포를 사용하는 DB 인스턴스에 대한 고가용성 및 장애 조치 지원을 제공합니다. 이러한 유형의 배포를 다중 AZ DB 인스턴스 배포라고 합니다. Amazon Timestream for InfluxDB는 Amazon 장애 조치 기술을 사용합니다.

다중 AZ DB 인스턴스 배포에서 Amazon Timestream은 다른 가용 영역에 동기 대기 복제본을 자동으로 프로비저닝하고 유지합니다. 기본 DB 인스턴스는 가용 영역 전체에서 대기 복제본으로 동기식으로 복제되어 데이터 중복성을 제공합니다. 고가용성으로 DB 인스턴스를 실행하면 DB 인스턴스 장애 및 가용 영역 중단 시 가용성이 향상될 수 있습니다. 가용 영역에 대한 자세한 내용은 AWS 리전 및 가용 영역 단원을 참조하세요.

> **Note**
>
> 고가용성 옵션은 읽기 전용 시나리오에서는 확장 솔루션이 아닙니다. 대기 복제본을 사용하여 읽기 트래픽을 처리할 수 없습니다.

Amazon Timestream 콘솔을 사용하면 DB 인스턴스를 생성할 때 가용성 및 내구성 구성 섹션에서 대기 인스턴스 생성 옵션을 지정하기만 하면 다중 AZ DB 인스턴스 배포를 생성할 수 있습니다. AWS Command Line Interface 또는 Amazon Timestream 를 사용하여 다중 AZ DB 인스턴스 배포를 지정할 수도 있습니다API. `create-db-instance` 또는 CLI 명령 또는 `CreateDBInstance` API 작업을 사용합니다.

다중 AZ DB 인스턴스 배포를 사용하는 DB 인스턴스는 단일 AZ 배포에 비해 쓰기 및 커밋 대기 시간이 길어질 수 있습니다. 이러한 현상은 동기식 데이터 복제가 발생하기 때문에 일어날 수 있습니다. AWS가 가용 영역 간의 지연 시간이 짧은 네트워크 연결로 설계되었지만 배포가 대기 복제본으로 장애 조치되는 경우 지연 시간이 변경될 수 있습니다. 프로덕션 워크로드의 경우 빠르고 일관된 성능을 위해 IOPS 포함된 스토리지 12K 또는 16KIOPS를 사용하는 것이 좋습니다. DB 인스턴스 클래스에 대한 자세한 내용은 DB 인스턴스 클래스 섹션을 참조하세요.

다중 AZ 배포 구성 및 관리

InfluxDB 다중 AZ 배포의 Timestream에는 하나의 대기만 있을 수 있습니다. 배포에 하나의 대기 DB 인스턴스가 있는 경우를 다중 AZ DB 인스턴스 배포라고 합니다. 다중 AZ DB 인스턴스 배포에는 장애 조치 지원을 제공하지만, 읽기 트래픽은 처리하지 않는 대기 DB 인스턴스가 하나 있습니다.

> **Important**
>
> 단일 AZ에서 다중 AZ로의 업데이트를 실행하려면 인스턴스에 연결된 서브넷이 두 개 이상 있어야 합니다. 인스턴스가 생성되면 단일 AZ에서 다중 AZ 로 배포 모드를 수정할 수 없습니다.

AWS Management Console 를 사용하여 DB 인스턴스가 단일 AZ 배포인지 다중 AZ 배포인지 확인할 수 있습니다.

사용 AWS Management Console

1. 에 로그인 AWS Management Console 하고 Amazon Timestream for InfluxDB 콘솔을 엽니다.
2. 탐색 창에서 InfluxDB 데이터베이스 를 선택한 다음 DB 식별자 를 선택합니다.

다중 AZ DB 인스턴스 배포의 특징은 다음과 같습니다.

- DB 인스턴스에는 행이 하나만 있습니다.
- 역할(Role) 값은 인스턴스(Instance) 또는 기본(Primary)입니다.
- 다중 AZ(Multi-AZ) 값은 예(Yes)입니다.

Amazon Timestream에 대한 장애 조치 프로세스

계획되거나 계획되지 않은 DB 인스턴스 중단이 인프라 결함으로 인해 발생하는 경우 Amazon Timestream for InfluxDB는 다중 AZ를 켠 경우 다른 가용 영역의 대기 복제본으로 자동으로 전환됩니다. 장애 조치가 완료되는 데 소요되는 시간은 프라이머리 DB 인스턴스를 사용할 수 없게 된 시점의 데이터베이스 활동 및 기타 조건에 따라 달라집니다. 장애 조치에 소요되는 시간은 일반적으로 60–120초입니다. 그러나 트랜잭션의 규모가 크거나 복구 프로세스가 복잡한 경우 장애 조치에 소요되는 시간이 증가할 수 있습니다. 장애 조치가 완료되면 Timestream 콘솔이 새 가용 영역을 반영하는 데 추가 시간이 걸릴 수 있습니다.

> **Note**
>
> Amazon Timestream은 장애 조치를 자동으로 처리하므로 관리 개입 없이 최대한 빨리 데이터베이스 작업을 재개할 수 있습니다. 다음 표에 설명된 조건 중 하나가 발생하면 기본 DB 인스턴스는 자동으로 예비 복제본으로 전환됩니다.

장애 조치 이유	설명
Timestream 데이터베이스 인스턴스의 기반이 되는 운영 체제가 오프라인 작업에서 패치되고 있습니다.	OS 패치 또는 보안 업데이트를 위한 유지 관리 기간 동안 장애 조치가 트리거되었습니다.
Timestream 다중 AZ 인스턴스의 기본 호스트가 비정상입니다.	다중 AZ DB 인스턴스 배포에서 손상된 프라이머리 DB 인스턴스를 감지하여 장애 조치를 수행했습니다.
네트워크 연결 손실로 인해 Timestream 다중 AZ 인스턴스의 기본 호스트에 연결할 수 없습니다.	타임스트림 모니터링에서 기본 DB 인스턴스에 대한 네트워크 연결 실패를 감지하고 장애 조치를 트리거했습니다.
고객이 Timestream 인스턴스를 수정했습니다.	InfluxDB DB 인스턴스 수정을 위한 Timesteam이 장애 조치를 트리거했습니다. 자세한 내용은 DB 인스턴스 업데이트 단원을 참조하십시오.
Timestream 다중 AZ 기본 인스턴스가 사용 중이며 응답하지 않습니다.	기본 DB 인스턴스가 응답하지 않습니다. * 이벤트에서 과도한 CPU, 메모리 또는 스왑 공간 사용량을 검사하는 것이 좋습니다. * 워크로드를 평가하여 적절한 DB 인스턴스 클래스를 사용하고 있는지 확인합니다. 자세한 내용은 DB 인스턴스 클래스를 참조하세요.
Timestream 다중 AZ 인스턴스의 기본 호스트의 기본 스토리지 볼륨에 장애가 발생했습니다.	다중 AZ DB 인스턴스 배포가 프라이머리 DB 인스턴스에서 스토리지 문제를 감지하여 장애 조치를 수행했습니다.

DNS 이름 조회를 JVM TTL 위한 설정

장애 조치 메커니즘은 DB 인스턴스의 도메인 이름 시스템(DNS) 레코드를 대기 DB 인스턴스를 가리키도록 자동으로 변경합니다. 그 결과 DB 인스턴스의 기존 연결을 모두 재설정해야 합니다. Java 가상 머신(JVM) 환경에서는 Java DNS 캐싱 메커니즘이 작동하는 방식으로 인해 JVM 설정을 재구성해야 할 수 있습니다.

JVM 캐시 DNS 이름 조회입니다. 에서 호스트 이름을 IP 주소로 JVM 확인하면 time-to-live ()라고 하는 지정된 기간 동안 IP 주소를 캐싱합니다TTL.

AWS 리소스는 가끔 변경되는 DNS 이름 항목을 사용하기 때문에 를 60초 이하의 JVM TTL 값으로 구성하는 것이 좋습니다. 이렇게 하면 리소스의 IP 주소가 변경되면 애플리케이션이 를 다시 쿼리하여 리소스의 새 IP 주소를 수신하고 사용할 수 있습니다DNS.

일부 Java 구성에서는 TTL가 다시 시작될 때까지 DNS 항목을 새로 고치지 않도록 JVM 기본값JVM이 설정됩니다. 따라서 애플리케이션이 실행 중인 동안 AWS 리소스의 IP 주소가 변경되면 를 수동으로 다시 JVM 시작하고 캐시된 IP 정보가 새로 고쳐질 때까지 해당 리소스를 사용할 수 없습니다. 이 경우 캐시된 IP 정보를 주기적으로 새로 고치JVMTTL도록 를 설정하는 것이 중요합니다.

`networkaddress.cache.ttl` 속성 값을 검색TTL하여 JVM 기본값을 가져올 수 있습니다.

```
String ttl = java.security.Security.getProperty("networkaddress.cache.ttl");
```

> **Note**
>
> 기본값은 의 버전JVM과 보안 관리자 설치 여부에 따라 달라질 TTL 수 있습니다. 많은 사용자가 기본값을 60초 TTL 미만으로 JVMs 제공합니다. 이러한 를 사용하고 보안 관리자를 사용하지 JVM 않는 경우 이 주제의 나머지 부분을 무시할 수 있습니다.
> JVM의 를 수정하려면 networkaddress.cache.ttl 속성 값을 TTL설정합니다. 필요에 따라 다음 방법 중 하나를 사용합니다.
>
> - 를 사용하는 모든 애플리케이션에 대해 속성 값을 전역적으로 설정하려면 `$JAVA_HOME/jre/lib/security/java.security` 파일에 `networkaddress.cache.ttl`를 JVM설정합니다.
>
>   ```
>   networkaddress.cache.ttl=60
>   ```
>
> - 애플리케이션에만 로컬로 속성을 설정하려면 네트워크 연결이 설정되기 전에 애플리케이션의 초기화 코드에서 `networkaddress.cache.ttl`을 설정합니다.
>
>   ```
>   java.security.Security.setProperty("networkaddress.cache.ttl" , "60");
>   ```

Timestream Influxdb 인스턴스에서 InfluxDB 로그를 볼 수 있도록 설정

기본적으로 InfluxDB는 stdout으로 이동하는 로그를 생성합니다. 자세한 내용은 InfluxDB 로그 관리를 참조하세요.

Timestream InfluxDB를 통해 생성한 인스턴스에서 생성된 InfluxDB 로그를 보려면 시간당 로그를 제공할 수 있는 기회를 제공합니다. 이러한 로그는 인스턴스를 생성하기 전에 생성해야 하는 지정된 S3 버킷으로 이동합니다.

- 인스턴스를 생성하기 전에 제공된 Amazon S3 버킷은 다음과 같이 Timestream InfluxDB InfluxDB Service Principal(교체 *{BUCKET_NAME}* Amazon S3 버킷의 실제 이름:

```
{
    "Version": "2012-10-17",
    "Id": "PolicyForInfluxLogs",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "timestream-influxdb.amazonaws.com"
            },
            "Action": "s3:PutObject",
            "Resource": "arn:aws:s3:::{BUCKET_NAME}/InfluxLogs/*"
        }
    ]
}
```

- 제공된 버킷은 생성된 Timestream InfluxDB 인스턴스의 동일한 계정과 동일한 리전에 있어야 합니다.

  다음은 유입 로그를 수신할 인스턴스를 만들기 위해 호출할 수 있는 명령입니다.

```
aws timestream-influxdb create-db-instance \
    --name myinfluxDbinstance \
    --allocated-storage 400 \
    --db-instance-type db.influx.4xlarge \
    --vpc-subnet-ids subnetid1 subnetid2
    --vpc-security-group-ids mysecuritygroup \
    --username masterawsuser \
    --password \
    --db-storage-type InfluxIOIncludedT2
```

  다음은 이 파라미터의 형식입니다.

```
-- log-delivery-configuration
{
    "S3Configuration": {
```

```
      "BucketName": "string",
      "Enabled": true|false
    }
}
```

- 이 필드는 필수가 아니며 로깅은 기본적으로 활성화되어 있지 않습니다.
- 이 필드를 설정하지 않는 것은 로그를 활성화하지 않는 것과 동일합니다.
- 로그는 접두사가 인 지정된 버킷으로 전송됩니다`InfluxLogs/`.
- 인스턴스를 생성한 후 `update-db-instance` API 명령을 사용하여 로그 전송 구성을 수정할 수 있습니다.

InfluxDB는 다양한 유형의 로그를 제공합니다. InfluxDB 파라미터를 설정하여 구성할 수 있습니다. 및 로그 수준 파라미터를 사용하여 flux-log-enabled 인스턴스에서 내보내는 로그 유형을 구성합니다. 자세한 내용은 지원되는 파라미터 및 파라미터 값 단원을 참조하십시오.

리소스에 태그 및 레이블 추가

태그를 사용하여 Amazon Timestream for InfluxDB 리소스에 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.
- 태그로 분류된 AWS 청구서를 참조하세요.

태깅은 Amazon Elastic Compute Cloud(Amazon EC2), Amazon Simple Storage Service(Amazon S3), Timestream for InfluxDB 등과 같은 AWS 서비스에서 지원됩니다. 효율적으로 태그를 지정하면 특정 태그와 연결하여 서비스 전체에 대해 생성된 보고서를 통해 비용을 분석할 수 있습니다.

끝으로, 최적화된 태깅 전략을 따르는 것이 좋습니다. 자세한 내용은 AWS 태깅 전략을 참조하세요.

태그 지정 제한

각 태그는 사용자가 정의하는 키와 값으로 구성됩니다. 다음과 같은 제한 사항이 있습니다.

- InfluxDB DB 인스턴스의 각 Timestream에는 동일한 키가 있는 태그가 하나만 있을 수 있습니다. 기존 태그를 추가하려고 하면 기존 태그 값이 새 값으로 업데이트됩니다.

- 값은 태그 범주 내에서 설명자 역할을 합니다. InfluxDB의 Timestream에서 값은 비어 있거나 null일 수 없습니다.
- 태그 키와 값은 대소문자를 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-할당된 태그 이름과 값은 `aws:` 자동으로 접두사가 할당되며, 이 접두사는 할당할 수 없습니다. AWS-할당된 태그 이름은 태그 한도 50에 포함되지 않습니다. 비용 할당 보고서에는 사용자가 지정한 태그 이름인 `user:` 접두사가 포함됩니다.
- 태그를 소급해서 적용할 수 없습니다.

Timestream for InfluxDB의 보안 모범 사례

InfluxDB에 쓰기 최적화

다른 시계열 데이터베이스와 마찬가지로 InfluxDB는 데이터를 실시간으로 수집 및 처리할 수 있도록 구축되었습니다. 시스템 성능을 최상으로 유지하려면 InfluxDB에 데이터를 쓸 때 다음 최적화를 수행하는 것이 좋습니다.

- 배치 쓰기: InfluxDB 데이터를 쓸 때 모든 쓰기 요청과 관련된 네트워크 오버헤드를 최소화하기 위해 데이터를 배치로 씁니다. 최적의 배치 크기는 쓰기 요청당 5,000줄의 라인 프로토콜입니다. 한 요청에 여러 줄을 쓰려면 각 줄의 프로토콜이 새 줄(\n)로 구분되어야 합니다.
- 키별 태그 정렬: InfluxDB에 데이터 포인트를 작성하기 전에 어휘 순서로 키별 태그를 정렬합니다.

```
measurement,tagC=therefore,tagE=am,tagA=i,tagD=i,tagB=think fieldKey=fieldValue
 1562020262

# Optimized line protocol example with tags sorted by key
measurement,tagA=i,tagB=think,tagC=therefore,tagD=i,tagE=am fieldKey=fieldValue
 1562020262
```

- 가능한 가장 거친 시간 정밀도를 사용합니다. – InfluxDB는 나노초 정밀도로 데이터를 쓰지만, 데이터가 나노초 단위로 수집되지 않으면 해당 정밀도로 쓸 필요가 없습니다. 더 나은 성능을 위해 타임스탬프에 가능한 가장 거친 정밀도를 사용하세요. 다음과 같은 경우 쓰기 정밀도를 지정할 수 있습니다.

  - 를 사용할 때 포인트의 시간 속성을 설정할 WritePrecision 때 를 지정할 SDK 수 있습니다. InfluxDB 클라이언트 라이브러리에 대한 자세한 내용은 InfluxDB 설명서 섹션을 참조하세요.
  - Telegraf를 사용하는 경우 Telegraf 에이전트 구성에서 시간 정밀도를 구성합니다. 정밀도는 정수 + 단위(예: 0s,10ms,2us,4s)가 포함된 간격으로 지정됩니다. 유효한 시간 단위는 “ns”, “us”, “ms”, “s”입니다.

```
[agent]
 interval ="10s"
 metric_batch_size="5000"
 precision = "0s"
```

- gzip 압축 사용: - gzip 압축을 사용하여 InfluxDB에 대한 쓰기 속도를 높이고 네트워크 대역폭을 줄입니다. 벤치마크는 데이터가 압축될 때 최대 5배의 속도 개선을 보여주었습니다.
  - Telegraf를 사용하는 경우 telegraf.conf의 Influxdb_v2 출력 플러그인 구성에서 content_encoding 옵션을 gzip으로 설정합니다.

```
[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  # ...
  content_encoding = "gzip"
```

  - 클라이언트 라이브러리를 사용할 때 각 InfluxDB 클라이언트 라이브러리는 쓰기 요청을 압축하거나 기본적으로 압축을 적용하는 옵션을 제공합니다. 압축을 활성화하는 방법은 라이브러리마다 다릅니다. 특정 지침은 InfluxDB 설명서를 참조하세요.
  - InfluxDB API `/api/v2/write` 엔드포인트를 사용하여 데이터를 쓸 때는 gzip으로 데이터를 압축하고 콘텐츠 인코딩 헤더를 gzip으로 설정합니다.

성능을 위한 설계

더 간단하고 더 많은 성능 쿼리를 위해 스키마를 설계합니다. 다음 지침은 스키마를 쉽게 쿼리하고 쿼리 성능을 극대화할 수 있도록 합니다.

- 쿼리할 설계: 쿼리하기 쉬운 측정값, 태그 키 및 필드 키를 선택합니다. 이 목표를 달성하려면 다음 원칙을 따르세요.
  - 이름이 간단한 측정값을 사용하고 스키마를 정확하게 설명합니다.
  - 동일한 스키마 내에서 태그 키와 필드 키에 동일한 이름을 사용하지 마세요.
  - 태그 및 필드 키에 예약 Flux 키워드와 특수 문자를 사용하지 마세요.

  - 태그는 필드를 설명하는 메타데이터를 저장하며 많은 데이터 포인트에서 공통적입니다.
  - 필드는 고유하거나 매우 가변적인 데이터, 일반적으로 숫자 데이터 포인트를 저장합니다.
  - 측정 및 키에는 데이터가 포함되어서는 안 되며 데이터를 집계하거나 설명하는 데 사용해야 합니다. 데이터는 태그 및 필드 값에 저장됩니다.
- 시계열 카디널리티를 제어 상태로 유지합니다. 높은 시리즈 카디널리티는 InfluxDB에서 쓰기 및 읽기 성능이 저하되는 주요 원인 중 하나입니다. InfluxDB의 맥락에서 높은 카디널리티는 매우 많은 수의 고유 태그 값이 존재하는 것을 의미합니다. 태그 값은 InfluxDB에서 인덱싱됩니다. 즉, 고유 값의 수가 매우 많으면 더 큰 인덱스가 생성되어 데이터 수집 및 쿼리 성능이 느려질 수 있습니다.

  잠재적인 높은 카디널리티 관련 문제를 더 잘 이해하고 해결하려면 다음 단계를 따르세요.

  - 카디널리티가 높은 원인 이해
  - 버킷의 카디널리티 측정
  - 높은 카디널리티를 해결하기 위한 작업 수행
- 높은 시리즈 카디널리티 InfluxDB의 원인은 측정 및 태그를 기반으로 데이터를 인덱싱하여 데이터 읽기 속도를 높입니다. InfluxDB 각 인덱싱된 데이터 요소 세트는 시리즈 키 를 구성합니다. 고유한 IDs, 해시 및 무작위 문자열과 같은 매우 가변적인 정보가 포함된 태그는 높은 시리즈 카디널리티 라고도 하는 많은 수의 시리즈 를 생성합니다. 높은 시리즈 카디널리티는 InfluxDB.
- 시리즈 카디널리티 측정 성능 저하가 발생하거나 Timestream for InfluxDB 인스턴스에서 메모리 사용량이 계속 증가하는 경우 버킷의 시리즈 카디널리티를 측정하는 것이 좋습니다.

  InfluxDB는 Flux 및 InfluxQL 모두에서 시리즈 카디널리티를 측정할 수 있는 함수를 제공합니다.

  - Flux에서 함수 사용 `influxdb.cardinality()`
  - FluxQL에서 `SHOW SERIES CARDINALITY` 명령을 사용합니다.

  두 경우 모두 엔진은 데이터의 고유한 시리즈 키 수를 반환합니다. Timestream for InfluxDB 인스턴스에는 1,000만 개 이상의 시리즈 키를 두지 않는 것이 좋습니다.
- 일련의 카디널리티가 높은 원인 버킷 중 하나라도 카디널리티가 높은 경우 이를 수정하기 위해 취할 수 있는 몇 가지 수정 단계가 있습니다.
  - 태그 검토: 워크로드가 케이스를 생성하지 않는지 확인합니다. 대부분의 항목에 대해 태그에 고유한 값이 있는지 확인합니다. 이는 시간이 지남에 따라 고유 태그 값의 수가 항상 증가하는 경우 또는 모든 메시지에 타임스탬프, 태그 등의 고유한 조합이 있는 로그 유형 메시지가 데이터베이스에 기록되는 경우에 발생할 수 있습니다. 다음 Flux 코드를 사용하여 카디널리티가 높은 문제에 가장 많이 기여하는 태그를 파악할 수 있습니다.

```
// Count unique values for each tag in a bucketimport "influxdata/influxdb/schema"
```

```
cardinalityByTag = (bucket) => schema.tagKeys(bucket: bucket)
    |> map(
        fn: (r) => ({
            tag: r._value,
            _value: if contains(set: ["_stop", "_start"], value: r._value) then
                0
            else
                (schema.tagValues(bucket: bucket, tag: r._value)
                    |> count()
                    |> findRecord(fn: (key) => true, idx: 0))._value,
        }),
    )
    |> group(columns: ["tag"])
    |> sum()

cardinalityByTag(bucket: "example-bucket")
```

카디널리티가 매우 높은 경우 위의 쿼리가 시간 초과될 수 있습니다. 제한 시간이 발생하면 아래 쿼리를 한 번에 하나씩 실행합니다.

태그 목록 생성:

```
// Generate a list of tagsimport "influxdata/influxdb/schema"

schema.tagKeys(bucket: "example-bucket")
```

각 태그의 고유 태그 값을 계산합니다.

```
// Run the following for each tag to count the number of unique tag valuesimport
 "influxdata/influxdb/schema"

tag = "example-tag-key"

schema.tagValues(bucket: "my-bucket", tag: tag)
    |> count()
```

어떤 태그가 더 빠르게 성장하고 있는지 식별하려면 다른 시점에 실행하는 것이 좋습니다.

- 스키마 개선: 에서 설명한 모델링 권장 사항을 따릅니다Timestream for InfluxDB의 보안 모범 사례.

- 카디널리티를 줄이기 위해 이전 데이터를 제거하거나 집계합니다. 사용 사례에 카디널리티 문제가 높은 모든 데이터가 필요한지 여부를 고려합니다. 이 데이터가 더 이상 필요하지 않거나 자주 액세스하지 않는 경우 집계하거나 삭제하거나 Timestream for Live Analytics와 같은 다른 엔진으로 내보내 장기 스토리지 및 분석을 수행할 수 있습니다.

문제 해결

“dev” 버전에 대한 경고가 인식되지 않음

마이그레이션 중에 'WARN: 최신 백업/복원이 지원된다고 가정하여 서버에서 보고한 'dev' 버전을 구문 분석할 수 없음' 경고APIs가 표시될 수 있습니다. 이 경고는 무시할 수 있습니다.

복원 단계 중 마이그레이션 실패

복원 단계 중에 마이그레이션이 실패한 경우 사용자는 `--retry-restore-dir` 플래그를 사용하여 복원을 다시 시도할 수 있습니다. 이전에 백업된 디렉터리에 대한 경로가 있는 `--retry-restore-dir` 플래그를 사용하여 백업 단계를 건너뛰고 복원 단계를 다시 시도합니다. 복원 중에 마이그레이션이 실패하면 마이그레이션에 사용된 생성된 백업 디렉터리가 표시됩니다.

복원 실패의 가능한 이유는 다음과 같습니다.

- 잘못된 InfluxDB 대상 토큰 - 소스 인스턴스와 이름이 동일한 대상 인스턴스에 존재하는 버킷입니다. 개별 버킷 마이그레이션의 경우 `--dest-bucket` 옵션을 사용하여 마이그레이션된 버킷의 고유한 이름을 설정합니다.
- 소스 또는 대상 호스트 또는 선택적 S3 버킷을 통한 연결 실패.

Amazon Timestream for InfluxDB 기본 운영 지침

다음은 Amazon Timestream for InfluxDB 를 사용할 때 모든 사람이 따라야 하는 기본 운영 지침입니다. Amazon Timestream for InfluxDB 서비스 수준 계약에 따라 다음 지침을 따라야 합니다.

- 지표를 사용하여 메모리, CPU및 스토리지 사용량을 모니터링합니다. 사용 패턴이 변경되거나 배포 용량에 가까워질 때 알리 CloudWatch 도록 Amazon을 설정할 수 있습니다. 이렇게 하면 시스템 성능 및 가용성을 유지할 수 있습니다.
- 스토리지 용량 한도에 도달할 경우 DB 인스턴스를 확장합니다. 스토리지 및 메모리에 어느 정도 버퍼가 있어야만 애플리케이션에서 수요가 예기치 않게 늘어날 경우 이를 수용할 수 있습니다. 이때 이를 위해서는 새 인스턴스를 생성하고 데이터를 마이그레이션해야 합니다.

- 데이터베이스 작업량으로 인해 프로비저닝한 I/O보다 많이 필요할 경우 장애 조치 또는 데이터베이스 오류가 발생한 후에 복구 속도가 느려집니다. DB 인스턴스의 I/O 용량을 늘리려면 다음 중 일부 항목이나 모든 항목을 수행하십시오.
  - I/O 용량이 더 높은 다른 DB 인스턴스로 마이그레이션합니다.
  - Influx IOPS 포함 스토리지를 이미 사용 중인 경우 더 높은 스토리지 유형을 IOPS 프로비저닝합니다.
- 클라이언트 애플리케이션이 DB 인스턴스의 도메인 이름 서비스(DNS) 데이터를 캐싱하는 경우 (TTL) 값을 30초 미만으로 설정합니다 time-to-live. 장애 조치 후에 DB 인스턴스의 기본 IP 주소가 변경될 수 있습니다. 따라서 장기간 DNS 데이터를 캐싱하면 연결 실패가 발생할 수 있습니다. 애플리케이션이 더 이상 사용되지 않는 IP 주소에 연결을 시도할 수 있습니다.

DB 인스턴스 RAM 권장 사항

Amazon Timestream for InfluxDB 성능 모범 사례는 작업 세트가 메모리에 거의 완전히 상주할 RAM 수 있도록 충분히 할당하는 것입니다. 작업 집합은 인스턴스에서 자주 사용되는 데이터 및 인덱스입니다. DB 인스턴스를 많이 사용할수록 작업 집합이 커집니다.

InfluxDB의 Timestream 보안

의 클라우드 보안 AWS 이 최우선 순위입니다. AWS 고객은 가장 보안에 민감한 조직의 요구 사항을 충족하도록 구축된 데이터 센터 및 네트워크 아키텍처의 이점을 누릴 수 있습니다.

보안은 AWS 와 사용자 간의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 - AWS 는 AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한 는 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Timestream for InfluxDB에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 규정 AWS 준수 프로그램 의 범위 내 서비스를 참조하세요.
- 클라우드의 보안 - 사용자의 책임은 사용하는 AWS 서비스에 따라 결정됩니다. 또한 데이터의 민감도, 조직의 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Timestream for InfluxDB를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표에 맞게 InfluxDB용 Timestream을 구성하는 방

법을 보여줍니다. 또한 Timestream for InfluxDB 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법도 알아봅니다.

주제

개요

이 설명서는 Amazon Timestream for InfluxDB를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목표에 맞게 InfluxDB용 Amazon Timestream을 구성하는 방법을 보여줍니다. 또한 Amazon Timestream for InfluxDB 리소스를 모니터링하고 보호하는 데 도움이 되는 다른 AWS 서비스를 사용하는 방법도 알아봅니다.

DB 인스턴스에서 Amazon Timestream for InfluxDB 리소스 및 데이터베이스에 대한 액세스를 관리할 수 있습니다. 액세스를 관리하는 데 사용하는 방법은 사용자가 Amazon Timestream for InfluxDB로 수행해야 하는 작업 유형에 따라 달라집니다.

- 네트워크 액세스 제어를 위한 Amazon VPC 서비스를 기반으로 Virtual Private Cloud(VPC)에서 DB 인스턴스를 실행합니다.
- AWS Identity and Access Management(IAM) 정책을 사용하여 InfluxDB 리소스용 Amazon Timestream을 관리할 수 있는 사용자를 결정하는 권한을 할당합니다. 예를 들어 IAM를 사용하여 DB 인스턴스를 생성, 설명, 수정 및 삭제하거나 리소스에 태그를 지정하거나 보안 그룹을 수정할 수 있는 사용자를 결정할 수 있습니다.

- 보안 그룹을 사용하여 DB EC2 인스턴스의 데이터베이스에 연결할 수 있는 IP 주소 또는 Amazon 인스턴스를 제어합니다. DB 인스턴스를 처음 생성할 때는 연결된 보안 그룹에서 지정한 규칙을 통해서만 액세스할 수 있습니다.
- DB 인스턴스에 Secure Socket Layer(SSL) 또는 Transport Layer Security(TLS) 연결을 사용합니다.
- InfluxDB 엔진의 보안 기능을 사용하여 DB 인스턴스의 데이터베이스에 로그인할 수 있는 사용자를 제어합니다. 이러한 보안 기능은 데이터베이스가 마치 로컬 네트워크에 있는 것처럼 실행됩니다. 자세한 내용은 InfluxDB의 Timestream 보안 단원을 참조하십시오.

> **Note**
>
> 사용 사례에 따라 보안을 구성해야 합니다. Amazon Timestream for InfluxDB가 관리하는 프로세스에 대한 보안 액세스를 구성할 필요가 없습니다. 이러한 프로세스로는 백업 생성, 기본 DB 인스턴스와 읽기 전용 복제본 간 데이터 복제 등이 있습니다.

주제

일반 보안

주제

권한

InfluxDB 사용자에게는 최소 권한 권한이 부여되어야 합니다. 마이그레이션 중에는 운영자 토큰 대신 특정 사용자에게 부여된 토큰만 사용해야 합니다.

InfluxDB의 Timestream은 IAM 권한을 사용하여 사용자 권한을 제어합니다. 사용자에게 필요한 특정 작업 및 리소스에 대한 액세스 권한을 부여하는 것이 좋습니다. 자세한 내용은 최소 권한 액세스 권한 부여를 참조하세요.

네트워크 액세스

Influx 마이그레이션 스크립트는 로컬에서 작동하여 동일한 시스템의 두 InfluxDB 인스턴스 간에 데이터를 마이그레이션할 수 있지만 마이그레이션의 기본 사용 사례는 로컬 또는 퍼블릭 네트워크인 네트워크에서 데이터를 마이그레이션하는 것으로 가정합니다. 이를 통해 보안 고려 사항이 제공됩니다. Influx 마이그레이션 스크립트는 기본적으로 TLS 활성화된 인스턴스의 TLS 인증서를 확인합니다. 사용자가 InfluxDB 인스턴스TLS에서 를 활성화하고 스크립트`--skip-verify`에 옵션을 사용하지 않는 것이 좋습니다.

허용 목록을 사용하여 네트워크 트래픽을 예상하는 소스에서 발생하는 것으로 제한하는 것이 좋습니다. 알려진 에서만 네트워크 트래픽을 InfluxDB 인스턴스로 제한하여 이 작업을 수행할 수 있습니다 IPs.

의존성

Influx , InfluxDB CLI, Python, Requests 모듈 및 `mountpoint-s3` 및 와 같은 선택적 종속성을 포함하여 모든 종속성의 최신 메이저 버전을 사용해야 합니다`rclone`.

S3 버킷

S3 버킷을 마이그레이션을 위한 임시 스토리지로 사용하는 경우 TLS, 버전 관리 및 퍼블릭 액세스 비활성화를 활성화하는 것이 좋습니다.

마이그레이션에 S3 버킷 사용

1. 를 열고 Amazon Simple Storage Service로 AWS Management Console이동한 다음 버킷을 선택합니다.
2. 사용할 버킷을 선택합니다.
3. 권한 탭을 선택합니다.
4. 퍼블릭 액세스 차단(버킷 설정)(Block public access (bucket settings))에서 편집(Edit)을 선택합니다.
5. 모든 퍼블릭 액세스 차단을 선택합니다.
6. Save changes(변경 사항 저장)를 선택합니다.
7. 버킷 정책에서 편집을 선택합니다.
8. <example-bucket>을 버킷 이름으로 바꾸고 다음을 입력하여 연결에 TLS 버전 1.2 이상을 사용하도록 적용합니다.

```
{
```

```
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "EnforceTLSv12orHigher",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "s3:*"
            ],
            "Effect": "Deny",
            "Resource": [
                "arn:aws:s3:::<example bucket>/*",
                "arn:aws:s3:::<example bucket>"
            ],
            "Condition": {
                "NumericLessThan": {
                    "s3:TlsVersion": 1.2
                }
            }
        }
    ]
}
```

9. Save changes(변경 사항 저장)를 선택합니다.
10. 속성(Properties) 탭을 선택합니다.
11. 버킷 버전 관리(Bucket Versioning)에서 편집을 선택합니다.
12. 활성화를 선택합니다.
13. Save changes(변경 사항 저장)를 선택합니다.

Amazon S3 버킷 모범 보안 사례에 대한 자세한 내용은 Amazon Simple Storage Service의 보안 모범 사례를 참조하세요.

Amazon Timestream for InfluxDB를 사용한 데이터베이스 인증

Amazon Timestream for InfluxDB는 데이터베이스 사용자를 인증하는 두 가지 방법을 지원합니다.

암호 및 액세스 토큰 데이터베이스 인증은 데이터베이스에 대한 다양한 인증 방법을 사용합니다. 따라서 특정 사용자는 하나의 인증 방법만 사용하여 데이터베이스에 로그인할 수 있습니다. 두 경우 모두 InfluxDB는 사용자 계정 및 API 토큰의 모든 관리를 수행합니다.

암호 인증

InfluxDB DB 인스턴스 생성 프로세스 중에 조직, 사용자 및 암호를 생성했습니다. 사용자는 Timestream for InfluxDB DB 인스턴스의 모든 것을 관리할 수 있는 권한이 있습니다. 이 사용자 이름과 암호 조합을 사용하면 InfluxUI를 사용하여 인스턴스 LogIn 로 들어가고 InfluxCLI를 사용하여 운영자 토큰을 생성할 수도 있습니다.

사용자를 생성하고 버킷, 조직 등을 삭제하려면 운영자 토큰이 필요합니다. 자세한 내용은 데이터베이스 인증 옵션 단원을 참조하십시오.

API 토큰

InfluxDB API 토큰은 InfluxDB와 클라이언트 또는 애플리케이션과 같은 외부 도구 간의 안전한 상호 작용을 보장합니다. API 토큰은 특정 사용자에게 속하며 사용자 조직 내의 InfluxDB 권한을 식별합니다.

InfluxDB에는 세 가지 유형의 API 토큰이 있습니다. InfluxDB

- 운영자 토큰: InfluxDB OSS2.x의 모든 조직 및 모든 조직 리소스에 대한 전체 읽기 및 쓰기 액세스 권한을 부여합니다. 예를 들어 서버 구성을 검색하는 일부 작업에는 운영자 권한이 필요합니다. 설정 프로세스가 완료된 CLI 후 InfluxDB UI, `api/v2` API또는 Influx를 사용하여 운영자 토큰을 수동으로 생성하려면 기존 운영자 토큰 또는 사용자 이름과 암호를 사용해야 합니다. 기존 토큰을 사용하지 않고 새 운영자 토큰을 생성하려면 유입된 복구 인증 을 참조하세요CLI.

  > **Important**
  >
  > 운영자 토큰은 데이터베이스의 모든 조직에 대한 전체 읽기 및 쓰기 액세스 권한이 있으므로 각 조직에 대한 All-Access 토큰을 생성하고 이를 사용하여 InfluxDB 관리하는 것이 좋습니다. 이렇게 하면 조직 간 우발적 상호 작용을 방지하는 데 도움이 됩니다.

- 모든 액세스 API 토큰: 조직의 모든 리소스에 대한 전체 읽기 및 쓰기 액세스 권한을 부여합니다.
- 읽기/쓰기 토큰: 조직의 특정 버킷에 대한 읽기 액세스, 쓰기 액세스 또는 둘 다를 부여합니다.

모든 InfluxDb 토큰은 만료 날짜가 설정되지 않은 긴 유효 토큰이므로 사용자 또는 모든 액세스 토큰을 사용하여 클라이언트 또는 Telegraf 에이전트에서 모니터링 데이터를 전송하여 대시보드 애플리케이션에 포함시키지 않는 것이 좋습니다. 이러한 애플리케이션의 경우 작업을 완료하는 데 필요한 권한만 있는 읽기/쓰기 토큰을 생성합니다. influxDB 토큰을 생성하는 방법에 대한 자세한 내용은 토큰 생성을 참조하세요.

암호

InfluxDB 운영자 토큰은 인스턴스 설정 시 생성됩니다. 모든 액세스 및 읽기/쓰기 토큰과 같은 다른 종류의 토큰은 Influx CLI, Influx v2 API또는 InfluxDB 다중 사용자 교체 함수를 위한 Timestream을 사용하여 생성할 수 있습니다. API 토큰을 생성, 보기, 할당 및 삭제하는 방법은 토큰 관리를 참조하세요.

환경 변수를 통해 토큰을 저장 AWS Secrets Manager 하고 자주 를 사용하여 InfluxDB 토큰의 Timestream을 교체하는 것이 좋습니다. 환경 변수의 토큰 사용에 토큰 사용 및 InfluxDB 사용자 및 토큰에 대한 Timestream을 교체하는 보안 암호 교체 방법을 참조하세요.

다음 사항도 참조하세요.

- Amazon Timestream for InfluxDB의 인프라 보안
- Timestream for InfluxDB의 보안 모범 사례

Amazon Timestream for InfluxDB에서 보안 암호를 사용하는 방법

InfluxDB의 Timestream은 사용자 인터페이스를 통한 사용자 이름 및 암호 인증과 최소 권한 클라이언트 및 애플리케이션 연결을 위한 토큰 자격 증명을 지원합니다. InfluxDB 사용자의 Timestream은 조직 내에서 `allAccess` 권한을 갖는 반면 토큰은 모든 권한 집합을 가질 수 있습니다. 보안 API 토큰 관리를 위한 모범 사례에 따라 조직 내 세분화된 액세스를 위한 토큰을 관리하도록 사용자를 생성해야 합니다. Timestream for InfluxDB의 관리자 모범 사례에 대한 자세한 내용은 Influxdata 설명서 에서 확인할 수 있습니다.

AWS Secrets Manager 는 데이터베이스 보안 인증 정보, API 키 및 기타 보안 정보를 보호하는 데 사용할 수 있는 보안 암호 스토리지 서비스입니다. 그런 다음 코드에서 하드코딩된 보안 인증 정보를 Secrets Manager에 대한 API 호출로 바꿀 수 있습니다. 이렇게 하면 보안 암호가 존재하지 않기 때문에 코드를 검사하는 사람이 보안 암호를 손상시키지 않도록 할 수 있습니다. Secrets Manager에 대한 개요는 AWS Secrets Manager란 무엇입니까?를 참조하세요.

데이터베이스 인스턴스를 생성할 때 Timestream for InfluxDB는 다중 사용자 교체 AWS Lambda 함수에 사용할 관리자 암호를 자동으로 생성합니다. InfluxDB 사용자 및 토큰에 대한 Timestream을 교체하려면 교체하려는 각 사용자 또는 토큰에 대해 새 암호를 직접 생성해야 합니다. Lambda 함수를 사용하여 일정에 따라 각 보안 암호를 교체하도록 구성할 수 있습니다. 새 교체 보안 암호를 설정하는 프로세스는 Lambda 함수 코드 업로드, Lambda 역할 구성, 새 보안 암호 정의, 보안 암호 교체 일정 구성으로 구성됩니다.

보안 암호의 의미

Timestream for InfluxDB 사용자 보안 인증을 보안 암호에 저장할 때는 다음 형식을 사용합니다.

단일 사용자:

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbIdentifier": "<required: DB identifier>"
}
```

InfluxDB 인스턴스용 Timestream을 생성하면 관리자 보안 암호가 보안 인증 정보와 함께 Secrets Manager에 자동으로 저장되어 다중 사용자 Lambda 함수와 함께 사용됩니다. `adminSecretArn` 를 DB 인스턴스 요약 페이지에 있는 `Authentication Properties Secret Manager ARN` 값 또는 관리자 보안 암호ARN의 로 설정합니다. 새 관리자 암호를 생성하려면 연결된 보안 인증 정보가 이미 있어야 하며 보안 인증 정보에는 관리자 권한이 있어야 합니다.

Timestream for InfluxDB 토큰 보안 인증 정보를 보안 암호에 저장할 때는 다음 형식을 사용합니다.

다중 사용자:

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "org": "<required: organization to associate token with>",
  "adminSecretArn": "<required: ARN of the admin secret>",
  "type": "<required: allAccess or operator or custom>",
  "dbIdentifier": "<required: DB identifier>",
  "token": "<required unless generating a new token: token being rotated>",
  "writeBuckets": "<optional: list of bucketIDs for custom type token, must be input
 within plaintext panel, for example ['id1','id2']>",
  "readBuckets": "<optional: list of bucketIDs for custom type token, must be input
 within plaintext panel, for example ['id1','id2']>",
  "permissions": "<optional: list of permissions for custom type token, must be input
 within plaintext panel, for example ['write-tasks','read-tasks']>"
}
```

Timestream for InfluxDB 관리자 보안 인증을 보안 암호에 저장할 때는 다음 형식을 사용합니다.

관리자 보안 암호:

```
{
  "engine": "<required: must be set to 'timestream-influxdb'>",
  "username": "<required: username>",
  "password": "<required: password>",
  "dbIdentifier": "<required: DB identifier>",
  "organization": "<optional: initial organization>",
  "bucket": "<optional: initial bucket>"
}
```

보안 암호의 자동 교체를 활성화하려면 보안 암호가 올바른 JSON 구조에 있어야 합니다. InfluxDB 암호의 Timestream을 교체하는 방법은 보안 암호 교체 섹션을 참조하세요.

보안 암호 수정

Timestream for InfluxDB 인스턴스 생성 프로세스 중에 생성된 보안 인증 정보는 계정의 Secrets Manager 보안 암호에 저장됩니다. GetDbInstance 응답 객체에는 Amazon 리소스 이름(ARN)`influxAuthParametersSecretArn`을 해당 보안 암호에 유지하는 이 포함되어 있습니다. 암호는 Timestream for InfluxDB 인스턴스를 사용할 수 있는 후에만 채워집니다. 생성된 DB 인스턴스 updates/modifications/deletions에는 영향을 주지 않으므로 이 보안 암호의 READONLY 복사본입니다. 이 보안 암호를 삭제해도 API 응답은 삭제된 보안 암호 를 계속 참조합니다ARN.

기존 토큰 자격 증명을 저장하지 않고 InfluxDB 인스턴스용 Timestream에서 새 토큰을 생성하려면 보안 암호에 `token` 값을 비워 두고 `AUTHENTICATION_CREATION_ENABLED` Lambda 환경 변수가 로 설정된 다중 사용자 교체 함수를 사용하여 비작업자 토큰을 생성할 수 있습니다`true`. 새 토큰을 생성하면 보안 암호에 정의된 권한이 토큰에 할당되며 처음 교체가 성공한 후에는 변경할 수 없습니다. 보안 암호 교체에 대한 자세한 내용은 AWS 보안 암호 관리자 보안 암호 교체를 참조하세요.

보안 암호가 삭제되면 Timestream for InfluxDB 인스턴스의 연결된 사용자 또는 토큰이 삭제되지 않습니다.

보안 암호 교체

InfluxDB용 Timestream 단일 및 다중 사용자 교체 Lambda 함수를 사용하여 InfluxDB 사용자 및 토큰 자격 증명에 대한 Timestream을 교체하면 됩니다. 단일 사용자 Lambda 함수를 사용하여 Timestream for InfluxDB 인스턴스의 사용자 자격 증명을 교체하고, 다중 사용자 Lambda 함수를 사용하여 Timestream for InfluxDB 인스턴스의 토큰 자격 증명을 교체하세요.

단일 및 다중 사용자 Lambda 함수를 사용하여 사용자 및 토큰을 교체하는 것은 선택 사항입니다. InfluxDB 자격 증명의 Timestream은 만료되지 않으며 노출된 자격 증명은 DB 인스턴스에 대한 악의적인 작업을 일으킬 위험이 있습니다. Secrets Manager를 사용하여 Timestream for InfluxDB 보안 인증

정보를 교체하는 이점은 노출된 보안 인증 정보의 공격 벡터를 다음 교체 주기까지 기간으로 제한하는 보안 계층이 추가되었습니다. DB 인스턴스에 대한 교체 메커니즘이 없는 경우 노출된 자격 증명은 수동으로 삭제될 때까지 유효합니다.

사용자가 지정한 일정에 따라 Secrets Manager가 자동으로 보안 암호를 교체하도록 구성할 수 있습니다. 따라서 단기 보안 암호로 장기 보안 암호를 교체할 수 있어 손상 위험이 크게 줄어듭니다. Secrets Manager를 사용한 보안 암호 교체에 대한 자세한 내용은 AWS Secrets Manager 보안 암호 교체를 참조하세요.

사용자 교체

단일 사용자 Lambda 함수로 사용자를 교체하면 교체 후 매번 사용자에게 새 무작위 암호가 할당됩니다. 자동 교체를 활성화하는 방법에 대한 자세한 내용은 데이터베이스가 아닌 AWS Secrets Manager 보안 암호의 자동 교체 설정을 참조하세요.

관리자 보안 암호 교체

관리자 암호를 교체하려면 단일 사용자 교체 함수를 사용합니다. DB 초기화 시 `dbIdentifier` 값이 자동으로 채워지지 않으므로 보안 암호에 `engine` 및 값을 추가해야 합니다. 전체 보안 암호 템플릿은 섹션을 참조보안 암호의 의미하세요.

Timestream for InfluxDB 인스턴스의 관리자 암호를 찾으려면 ARN Timestream for InfluxDB 인스턴스 요약 페이지의 관리자 암호를 사용합니다. 관리자 사용자는 InfluxDB 인스턴스용 Timestream에 대한 권한이 높으므로 InfluxDB 관리자 보안 암호에 대한 모든 Timestream을 교체하는 것이 좋습니다.

Lambda 교체 함수

새 보안 암호와 함께 를 사용하고 Timestream for InfluxDB 사용자의 필수 필드를 추가하여 단일 사용자 교체 함수보안 암호의 의미로 InfluxDB 사용자의 Timestream을 교체할 수 있습니다. 보안 암호 교체 Lambda 함수에 대한 자세한 내용은 Lambda 함수에 의한 교체를 참조하세요.

단일 사용자 교체 함수는 보안 암호에 정의된 자격 증명을 사용하여 Timestream for InfluxDB DB 인스턴스로 인증한 다음 새 무작위 암호를 생성하고 사용자의 새 암호를 설정합니다. 보안 암호 교체 Lambda 함수에 대한 자세한 내용은 Lambda 함수에 의한 교체를 참조하세요.

Lambda 함수 실행 역할 권한

다음 IAM 정책을 단일 사용자 Lambda 함수의 역할로 사용합니다. 이 정책은 Lambda 함수에 InfluxDB 사용자의 Timestream에 대한 보안 암호 교체를 수행하는 데 필요한 권한을 부여합니다.

IAM 정책에 나열된 모든 항목을 AWS 계정의 값으로 바꿉니다.

- {rotating_secret_arn} - 교체 중인 보안 암호ARN의 는 Secrets Manager 보안 암호 세부 정보에서 찾을 수 있습니다.
- {db_instance_arn} - InfluxDB 인스턴스의 Timestream은 InfluxDB 인스턴스의 Timestream 요약 페이지에서 찾을 ARN 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
            ],
     "Resource": "{rotating_secret_arn}"
        },
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:GetRandomPassword"
            ],
            "Resource": "*"
        },
        {
            "Action": [
                "timestream-influxdb:GetDbInstance"
            ],
     "Resource": "{db_instance_arn}",
            "Effect": "Allow"
        }
    ]
}
```

토큰 교체

새 보안 암호와 함께 를 사용하고 Timestream for InfluxDB 토큰의 필수 필드를 추가하여 다중 사용자 교체 함수보안 암호의 의미로 Timestream for InfluxDB 토큰을 교체할 수 있습니다. 보안 암호 교체 Lambda 함수에 대한 자세한 내용은 Lambda 함수에 의한 교체를 참조하세요.

InfluxDB용 Timestream 다중 사용자 Lambda 함수를 사용하여 InfluxDB용 Timestream 토큰을 교체할 수 있습니다. Lambda 구성`true`에서 `AUTHENTICATION_CREATION_ENABLED` 환경 변수를 로 설정하여 토큰 생성을 활성화합니다. 새 토큰을 생성하려면 보안 암호 값에 보안 암호의 의미 를 사용합니다. 새 보안 암호에서 `token` 키-값 페어를 생략하고 를 `type`로 설정`allAccess`하거나 특정 권한을 정의하고 유형을 로 설정합니다`custom`. 교체 함수는 첫 번째 교체 주기 동안 새 토큰을 생성합니다. 교체 후에는 보안 암호를 편집하여 토큰 권한을 변경할 수 없으며 이후 교체 시 DB 인스턴스에 설정된 권한이 사용됩니다.

Lambda 교체 함수

다중 사용자 교체 함수는 관리자 보안 암호의 관리자 보안 인증 정보를 사용하여 동일한 토큰에 대한 새 권한을 생성하여 토큰 보안 인증을 교체합니다. Lambda 함수는 대체 토큰을 생성하고 새 토큰 값을 보안 암호에 저장하고 이전 토큰을 삭제하기 전에 보안 암호의 토큰 값을 검증합니다. Lambda 함수가 새 토큰을 생성하는 경우 먼저 `AUTHENTICATION_CREATION_ENABLED` 환경 변수가 로 설정되었는지`true`, 보안 암호에 토큰 값이 없는지, 토큰 유형이 유형 연산자가 아닌지 확인합니다.

Lambda 함수 실행 역할 권한

다음 IAM 정책을 다중 사용자 Lambda 함수의 역할로 사용합니다. 이 정책은 Lambda 함수에 InfluxDB 토큰용 Timestream에 대한 보안 암호 교체를 수행하는 데 필요한 권한을 부여합니다.

IAM 정책에 나열된 모든 항목을 AWS 계정의 값으로 바꿉니다.

- {rotating_secret_arn} - 교체 중인 보안 암호ARN의 는 Secrets Manager 보안 암호 세부 정보에서 찾을 수 있습니다.
- {authentication_properties_admin_secret_arn } - InfluxDB 관리자 암호의 Timestream은 InfluxDB 인스턴스의 Timestream 요약 페이지에서 찾을 ARN 수 있습니다.
- {db_instance_arn} - InfluxDB 인스턴스의 Timestream은 InfluxDB 인스턴스의 Timestream 요약 페이지에서 찾을 ARN 수 있습니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "secretsmanager:DescribeSecret",
                "secretsmanager:GetSecretValue",
                "secretsmanager:PutSecretValue",
                "secretsmanager:UpdateSecretVersionStage"
```

```
            ],
      "Resource": "{rotating_secret_arn}"
         },
         {
             "Effect": "Allow",
             "Action": [
                 "secretsmanager:GetSecretValue"
             ],
      "Resource": "{authentication_properties_admin_secret_arn}"
         },
         {
             "Effect": "Allow",
             "Action": [
                 "secretsmanager:GetRandomPassword"
             ],
             "Resource": "*"
         },
         {
             "Action": [
                 "timestream-influxdb:GetDbInstance"
             ],
      "Resource": "{db_instance_arn}",
             "Effect": "Allow"
         }
     ]
}
```

InfluxDB용 Timestream의 데이터 보호

AWS 공동 책임 모델 Amazon Timestream for InfluxDB의 데이터 보호에 적용됩니다. 이 모델에 설명된 대로 AWS 는 모든 를 실행하는 글로벌 인프라를 보호할 책임이 있습니다 AWS 클라우드. 사용자는 인프라에서 호스팅되는 콘텐츠를 관리해야 합니다. 사용하는 AWS 서비스 의 보안 구성과 관리 작업에 대한 책임도 사용자에게 있습니다. 데이터 프라이버시에 대한 자세한 내용은 데이터 프라이버시 섹션을 FAQ참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그의 AWS 책임 공유 모델 및 GDPR 블로그 게시물을 참조하세요.

데이터 보호를 위해 자격 증명을 보호하고 AWS 계정 AWS IAM Identity Center 또는 AWS Identity and Access Management ()를 사용하여 개별 사용자를 설정하는 것이 좋습니다IAM. 이렇게 하면 개별 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다단계 인증(MFA)을 사용합니다.

- SSL/TLS를 사용하여 AWS 리소스와 통신합니다. TLS 1.2가 필요하며 TLS 1.3을 권장합니다.
- 를 사용하여 API 및 사용자 활동 로깅을 설정합니다 AWS CloudTrail. CloudTrail 추적을 사용하여 AWS 활동을 캡처하는 방법에 대한 자세한 내용은 AWS CloudTrail 사용 설명서의 CloudTrail 추적 작업을 참조하세요.
- AWS 암호화 솔루션과 내의 모든 기본 보안 제어를 사용합니다 AWS 서비스.
- Amazon S3에 저장된 민감한 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용하세요.
- 명령줄 인터페이스 또는 FIPS 를 AWS 통해 에 액세스할 때 140-3개의 검증된 암호화 모듈이 필요한 경우 FIPS 엔드포인트를 API사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 연방 정보 처리 표준(FIPS) 140-3을 참조하세요.

고객의 이메일 주소와 같은 기밀 정보나 중요한 정보는 태그나 이름 필드와 같은 자유 양식 필드에 입력하지 않는 것이 좋습니다. 여기에는 콘솔, 또는 를 사용하여 Timestream for InfluxDB 또는 기타 AWS 서비스 를 사용하는 경우가 포함됩니다API AWS CLI AWS SDKs. 이름에 사용되는 태그 또는 자유 형식 텍스트 필드에 입력하는 모든 데이터는 청구 또는 진단 로그에 사용될 수 있습니다. 외부 서버에 URL를 제공하는 경우 해당 서버에 대한 요청을 검증URL하기 위해 에 보안 인증 정보를 포함하지 않는 것이 좋습니다.

저장 시 암호화 및 키 관리와 같은 Timestream for InfluxDB 데이터 보호 주제에 대한 자세한 내용을 알아보려면 아래에서 사용 가능한 주제를 선택합니다.

주제

- 저장 중 암호화
- 전송 중 암호화

저장 중 암호화

저장 시 InfluxDB 암호화를 위한 Timestream은 AWS Key Management Service (AWS KMS)에 저장된 암호화 키를 사용하여 저장 중인 모든 데이터를 암호화하여 보안을 강화합니다. 이 기능을 사용하면 중요한 데이터 보호와 관련된 운영 부담 및 복잡성을 줄일 수 있습니다. 저장 시 암호화를 사용하면 엄격한 암호화 규정 준수 및 규제 요구 사항이 필요한, 보안에 민감한 애플리케이션을 구축할 수 있습니다.

- 암호화는 기본적으로 Timestream for InfluxDB DB 인스턴스에서 활성화되며 끌 수 없습니다. 업계 표준 AES-256 암호화 알고리즘은 사용되는 기본 암호화 알고리즘입니다.
- AWS KMS 는 InfluxDB용 Timestream의 유휴 시 암호화에 사용됩니다.
- 암호화를 사용하기 위해 DB 인스턴스 클라이언트 애플리케이션을 수정할 필요가 없습니다.

전송 중 암호화

모든 Timestream for InfluxDB 데이터는 전송 중에 암호화됩니다. 기본적으로 Timestream for InfluxDB와의 모든 통신은 전송 계층 보안(TLS) 암호화를 사용하여 보호됩니다.

Amazon Timestream for InfluxDB를 오가는 트래픽은 지원되는 TLS 버전 1.2 또는 1.3을 사용하여 보호됩니다.

Amazon Timestream for InfluxDB의 자격 증명 및 액세스 관리

AWS Identity and Access Management (IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어하는 데 도움이 AWS 서비스 되는 입니다. IAM 관리자는 InfluxDB 리소스용 Timestream을 사용하도록 인증(로그인) 및 권한 부여(권한 부여)를 받을 수 있는 사용자를 제어합니다. IAM 는 추가 비용 없이 사용할 수 AWS 서비스 있는 입니다.

주제

- ID를 통한 인증
- 정책을 사용한 액세스 관리
- Amazon Timestream for InfluxDB의 작동 방식 IAM
- Amazon Timestream for InfluxDB의 자격 증명 기반 정책 예제
- InfluxDB 자격 증명 및 액세스를 위한 Amazon Timestream 문제 해결
- 에서 DB 인스턴스에 대한 액세스 제어 VPC
- Amazon Timestream for InfluxDB에 서비스 연결 역할 사용
- AWS Amazon Timestream for InfluxDB에 대한 관리형 정책
- VPC 엔드포인트를 통해 InfluxDB의 Timestream에 연결

ID를 통한 인증

인증은 자격 증명 AWS 으로 에 로그인하는 방법입니다. 로 AWS 계정 루트 사용자, IAM 사용자로 또는 IAM 역할을 수임하여 인증(에 로그인 AWS)되어야 합니다.

자격 증명 소스를 통해 제공된 자격 증명을 사용하여 에 페더레이션 자격 증명 AWS 으로 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증 및 Google 또는 Facebook 자격 증명은 페더레이션 자격 증명의 예입니다. 페더레이션 자격 증명으로 로

그인하면 관리자가 이전에 IAM 역할을 사용하여 자격 증명 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS 에 액세스하면 간접적으로 역할을 수임하게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 에 로그인하는 방법을 AWS 계정 AWS참조하세요.

AWS 프로그래밍 방식으로 에 액세스하는 경우 는 소프트웨어 개발 키트(SDK)와 명령줄 인터페이스(CLI)를 AWS 제공하여 자격 증명을 사용하여 요청에 암호화 방식으로 서명합니다. AWS 도구를 사용하지 않는 경우 직접 요청에 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS 서명 버전 4에서 API 요청을 참조하세요.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것이 AWS 좋습니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 다단계 인증 및 사용 설명서의 AWS 다단계 인증을 IAM 참조하세요IAM.

IAM 사용자 및 그룹

IAM 사용자는 한 사람 또는 애플리케이션에 대한 특정 권한이 AWS 계정 있는 내 자격 증명입니다. 가능한 경우 암호 및 액세스 키와 같은 장기 보안 인증 정보가 있는 IAM 사용자를 생성하는 대신 임시 보안 인증 정보를 사용하는 것이 좋습니다. 그러나 IAM 사용자와 장기 보안 인증이 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례에 대한 액세스 키 정기적으로 교체를 참조하세요.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어 라는 이름의 그룹을 지정IAMAdmins하고 해당 그룹에 IAM 리소스를 관리할 수 있는 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 사용자 사용 사례를 참조하세요.

IAM 역할

IAM 역할은 특정 권한이 AWS 계정 있는 내 자격 증명입니다. IAM 사용자와 비슷하지만 특정 사람과는 연결되지 않습니다. 에서 IAM 역할을 일시적으로 수임하려면 사용자에서 역할(콘솔)로 전환할 AWS Management Console수 있습니다. IAM 또는 AWS API 작업을 호출 AWS CLI 하거나 사용자 지정 를

사용하여 역할을 수임할 수 있습니다URL. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 역할 수임 방법을 참조하세요.

IAM 임시 자격 증명이 있는 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 타사 자격 증명 공급자에 대한 역할 생성을 참조하세요. IAM Identity Center를 사용하는 경우 권한 세트를 구성합니다. 인증 후 자격 증명이 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 의 역할과 상호 연관시킵니다IAM. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 권한 세트를 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 특정 작업에 대해 일시적으로 다른 권한을 맡을 IAM 수 있습니다.
- 교차 계정 액세스 - IAM 역할을 사용하여 다른 계정의 누군가(신뢰할 수 있는 보안 주체)가 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 에서는 정책을 리소스에 직접 연결할 AWS 서비스수 있습니다(역할을 프록시로 사용하는 대신). 크로스 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 에서 크로스 계정 리소스 액세스를 IAM 참조하세요.
- 교차 서비스 액세스 - 일부 는 다른 에서 기능을 AWS 서비스 사용합니다 AWS 서비스. 예를 들어 서비스에서 호출할 때 해당 서비스가 Amazon에서 애플리케이션을 실행EC2하거나 Amazon S3에 객체를 저장하는 것이 일반적입니다. 서비스는직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 작업을 수행할 수 있습니다.
  - 전달 액세스 세션(FAS) - IAM 사용자 또는 역할을 사용하여 에서 작업을 수행하면 보안 주체로 AWS간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS 는 를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 다른 AWS 서비스 또는 리소스와의 상호 작용을 완료해야 하는 요청을 수신할 때만 수행됩니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 액세스 세션 전달을 참조하세요.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 수임하는 IAM 역할입니다. IAM 관리자는 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다IAM. 자세한 내용은 IAM 사용 설명서의 에 권한을 위임할 역할 생성을 AWS 서비스 참조하세요.
  - 서비스 연결 역할 - 서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할에 대한 권한을 볼 수 있지만 편집할 수는 없습니다.

- Amazon에서 실행되는 애플리케이션 EC2 - IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI 또는 AWS API 요청을 수행하는 애플리케이션의 임시 보안 인증을 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장하는 것보다 좋습니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행 중인 프로그램이 임시 자격 증명을 가져올 수 있습니다. 자세한 내용은 IAM 사용 설명서EC2의 IAM 역할 사용을 참조하세요.

정책을 사용한 액세스 관리

정책을 AWS 생성하고 AWS 자격 증명 또는 리소스에 연결하여 의 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결된 AWS 경우 권한을 정의하는 의 객체입니다. 는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청할 때 이러한 정책을 AWS 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 에 JSON 문서 AWS 로 저장됩니다. JSON 정책 문서의 구조 및 내용에 대한 자세한 내용은 IAM 사용 설명서의 JSON 정책 개요를 참조하세요.

관리자는 정책을 사용하여 AWS JSON 대상에 액세스할 수 있는 사용자를 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 필요한 리소스에 대한 작업을 수행할 수 있는 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성할 수 있습니다. 그런 다음 관리자는 IAM 정책을 역할에 추가하고 사용자는 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하는 데 사용하는 방법에 관계없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole` 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI또는 에서 역할 정보를 가져올 수 있습니다 AWS API.

보안 인증 기반 정책

자격 증명 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 고객 관리형 정책을 사용하여 사용자 지정 IAM 권한 정의를 참조하세요.

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 의 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립 실행형 정책입니다 AWS 계정. 관리형 정책에는 AWS 관리형 정책 및 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책 중에서 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책 및 인라인 정책 중에서 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 문서입니다. 리소스 기반 정책의 예로는 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 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 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 이 포함될 수 있습니다 AWS 서비스.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책IAM에서는 의 AWS 관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACLs)

액세스 제어 목록(ACLs)은 리소스에 액세스할 수 있는 권한이 있는 보안 주체(계정 멤버, 사용자 또는 역할)를 제어합니다. ACLs 는 리소스 기반 정책과 유사하지만 JSON 정책 문서 형식을 사용하지는 않습니다.

Amazon S3 AWS WAF및 AmazonVPC은 를 지원하는 서비스의 예입니다ACLs. 에 대한 자세한 내용은 Amazon Simple Storage Service 개발자 안내서의 액세스 제어 목록(ACL) 개요를 ACLs참조하세요.

기타 정책 타입

AWS 는 덜 일반적인 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 - 권한 경계는 자격 증명 기반 정책이 IAM엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 객체의 자격 증명 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 IAM엔터티에 대한 권한 경계를 참조하세요.
- 서비스 제어 정책(SCPs) - 의 조직 또는 조직 단위(OU)에 대한 최대 권한을 지정하는 SCPs JSON 정책입니다 AWS Organizations. AWS Organizations 는 비즈니스가 소유 AWS 계정 한 여러 을 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직의 모든 기능을 활성화하면 서비스 제어 정책(SCPs)을 계정의 일부 또는 전체에 적용할 수 있습니다. 는 각 를 포함하여 멤버 계정의 엔터티에 대한 권한을 SCP 제한합니다 AWS 계정 루트 사용자. 조직 및 에 대한 자세한 내용은 AWS Organizations 사용 설명서의 서비스 제어 정책을 SCPs참조하세요.
- 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 참조하세요.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. AWS 에서 여러 정책 유형이 관련될 때 요청을 허용할지 여부를 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하세요.

Amazon Timestream for InfluxDB의 작동 방식 IAM

IAM Amazon Timestream for InfluxDB와 함께 사용할 수 있는 기능

IAM 기능	InfluxDB 지원을 위한 Timestream
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키	아니요
ACLs	아니요
ABAC (정책의 태그)	예
임시 보안 인증	예
보안 주체 권한	예
서비스 역할	아니요
서비스 링크 역할	예

Timestream for InfluxDB 및 기타 AWS 서비스가 대부분의 IAM 기능에서 작동하는 방식을 자세히 알아보려면 IAM 사용 설명서의 AWS 에서 로 작업하는 서비스를 IAM 참조하세요.

InfluxDB용 Timestream에 대한 자격 증명 기반 정책

ID 기반 정책 지원: 예

자격 증명 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 고객 관리형 정책을 사용하여 사용자 지정 IAM 권한 정의를 참조하세요.

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부된 작업 및 리소스와 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. 보안 인증 기반 정책에서는 보안 주체가 연결된 사용자 또는 역할에 적용되므로 보안 주체를 지정할 수 없습니다. JSON 정책에서 사용할 수 있는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 IAM JSON 정책 요소 참조를 참조하세요.

InfluxDB용 Timestream의 자격 증명 기반 정책 예제

InfluxDB 자격 증명 기반 정책의 Timestream 예제를 보려면 섹션을 참조하세요Amazon Timestream for InfluxDB의 자격 증명 기반 정책 예제.

InfluxDB용 Timestream 내의 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 문서입니다. 리소스 기반 정책의 예로는 IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 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 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 가 포함될 수 있습니다 AWS 서비스.

교차 계정 액세스를 활성화하려면 리소스 기반 정책의 보안 주체로 전체 계정 또는 다른 계정의 IAM엔터티를 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하십시오. 보안 주체와 리소스가 다른 에 있는 경우 신뢰할 수 AWS 계정있는 계정의 IAM 관리자는 보안 주체 엔터티(사용자 또는 역할)에게 리소스에 액세스할 수 있는 권한도 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 보안 주체에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 에서 크로스 계정 리소스 액세스를 IAM 참조하세요.

InfluxDB용 Timestream에 대한 정책 작업

정책 작업 지원: 예

관리자는 정책을 사용하여 AWS JSON 대상에 액세스할 수 있는 사용자를 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 작업을 설명합니다. 정책 작업은 일반적으로 연결된 AWS API 작업과 이름이 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업과 같은 몇 가지 예외가 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.

InfluxDB 작업에 대한 Timestream 목록을 보려면 서비스 권한 부여 참조의 Amazon Timestream for InfluxDB에서 정의한 작업을 참조하세요.

Timestream for InfluxDB의 정책 작업은 작업 전에 다음 접두사를 사용합니다.

```
timestream-influxdb
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "timestream-influxdb:action1",
      "timestream-influxdb:action2"
         ]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, `Describe`라는 단어로 시작하는 모든 태스크를 지정하려면 다음 태스크를 포함합니다.

```
"Action": "timestream-influxdb:Describe*"
```

InfluxDB용 Timestream에 대한 정책 리소스

정책 리소스 지원: 예

관리자는 정책을 사용하여 AWS JSON 대상에 액세스할 수 있는 사용자를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 객체를 지정합니다. 문장에는 Resource또는 NotResource요소가 반드시 추가되어야 합니다. 가장 좋은 방법은 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정하는 것입니다. 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": "*"
```

InfluxDB 리소스 유형 및 해당 의 Timestream 목록을 보려면 서비스 승인 참조의 Amazon Timestream for InfluxDB에서 정의한 리소스를 ARNs참조하세요. 각 리소스ARN의 를 지정할 수 있는 작업을 알아보려면 Amazon Timestream for InfluxDB에서 정의한 작업을 참조하세요.

InfluxDB용 Timestream의 정책 조건 키

서비스별 정책 조건 키 지원: 아니요

관리자는 정책을 사용하여 AWS JSON 대상에 액세스할 수 있는 사용자를 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 작업을 사용하여 조건을 AWS 평가합니다. 명문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예를 들어 IAM 리소스에 사용자 IAM 이름으로 태그가 지정된 경우에만 사용자에게 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS 는 전역 조건 키 및 서비스별 조건 키를 지원합니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 AWS 전역 조건 컨텍스트 키를 참조하세요.

InfluxDB용 Timestream의 액세스 제어 목록(ACLs)

지원 ACLs: 아니요

액세스 제어 목록(ACLs)은 리소스에 액세스할 수 있는 권한이 있는 보안 주체(계정 멤버, 사용자 또는 역할)를 제어합니다. ACLs 는 리소스 기반 정책과 유사하지만 JSON 정책 문서 형식을 사용하지는 않습니다.

InfluxDB용 Timestream을 사용한 속성 기반 액세스 제어(ABAC)

지원ABAC(정책의 태그): 예

속성 기반 액세스 제어(ABAC)는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. 에서는 AWS이러한 속성을 태그 라고 합니다. IAM 엔터티(사용자 또는 역할) 및 여러 AWS 리소스에 태그를 연결할 수 있습니다. 엔터티 및 리소스에 태그를 지정하는 것은 의 첫 번째 단계입니다ABAC. 그런 다음 보안 주체의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하는 ABAC 정책을 설계합니다.

ABAC 는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로워지는 상황에 도움이 됩니다.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 조건 요소에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

에 대한 자세한 내용은 IAM 사용 설명서의 ABAC 권한 부여를 사용하여 권한 정의를 ABAC참조하세요. 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 속성 기반 액세스 제어 사용(ABAC)을 ABAC참조하세요.

InfluxDB용 Timestream에서 임시 자격 증명 사용

임시 자격 증명 지원: 예

임시 자격 증명을 사용하여 로그인할 때 작동하지 AWS 서비스 않는 경우도 있습니다. 임시 자격 증명으로 AWS 서비스 작업하는 경우를 비롯한 자세한 내용은 IAM 사용 설명서의 AWS 서비스 에서 작업하는 IAM 섹션을 참조하세요.

사용자 이름 및 암호를 제외한 방법을 AWS Management Console 사용하여 에 로그인하는 경우 임시 보안 인증 정보를 사용합니다. 예를 들어 회사의 Single Sign-On(SSO) 링크를 AWS 사용하여 에 액세스하면 해당 프로세스가 임시 자격 증명을 자동으로 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 보안 인증을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 사용자에서 IAM 역할(콘솔)로 전환을 참조하세요.

AWS CLI 또는 를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다 AWS API. 그런 다음 이러한 임시 자격 증명을 사용하여 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생성하는

AWS. AWS recommends에 액세스할 수 있습니다. 자세한 내용은 의 임시 보안 자격 증명을 IAM참조하세요.

InfluxDB용 Timestream에 대한 교차 서비스 보안 주체 권한

전달 액세스 세션 지원(FAS): 예

IAM 사용자 또는 역할을 사용하여 에서 작업을 수행하면 보안 주체로 AWS간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS 는 를 호출하는 보안 주체의 권한을 다운스트림 서비스에 AWS 서비스 대한 요청과 AWS 서비스함께 사용합니다. FAS 요청은 서비스가 다른 AWS 서비스 또는 리소스와의 상호 작용을 완료해야 하는 요청을 수신할 때만 수행됩니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 액세스 세션 전달을 참조하세요.

InfluxDB용 Timestream의 서비스 역할

서비스 역할 지원: 아니요

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하기 위해 수임하는 IAM 역할입니다. IAM 관리자는 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다IAM. 자세한 내용은 IAM 사용 설명서의 에 권한을 위임할 역할 생성을 AWS 서비스 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 InfluxDB 기능의 Timestream이 중단될 수 있습니다. Timestream for InfluxDB가 지침을 제공하는 경우에만 서비스 역할을 편집합니다.

InfluxDB용 Timestream의 서비스 연결 역할

서비스 링크 역할 지원: 예

서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다 AWS 서비스. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 에 표시 AWS 계정 되며 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할에 대한 권한을 볼 수 있지만 편집할 수는 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 AWS 에서 작동하는 서비스를 참조하세요IAM. 서비스 연결 역할 열에서 Yes(이)가 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 Yes(네) 링크를 선택합니다.

Amazon Timestream for InfluxDB의 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할에는 InfluxDB 리소스용 Timestream을 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface (AWS CLI) 또는 를 사용하여 작업을 수행할 수 없습니다 AWS API. 사용자에게 필요한 리소스에 대한 작업을 수행할 수 있는 권한을 부여하기 위해 IAM 관리자는 IAM 정책을 생성할 수 있습니다. 그런 다음 관리자는 IAM 정책을 역할에 추가하고 사용자는 역할을 수임할 수 있습니다.

이러한 예제 정책 문서를 사용하여 IAM 자격 증명 기반 JSON 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성(콘솔)을 참조하세요.

각 리소스 유형에 대한 형식 등 Timestream for InfluxDB 정의한 작업 및 리소스 유형에 ARNs 대한 자세한 내용은 서비스 권한 부여 참조의 Amazon Timestream for InfluxDB에 대한 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- InfluxDB 콘솔용 Timestream 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용
- 하나의 Amazon S3 버킷에 액세스
- 모든 작업 허용
- DB 인스턴스 생성, 설명, 삭제 및 업데이트

정책 모범 사례

자격 증명 기반 정책은 계정에서 누군가가 InfluxDB 리소스용 Timestream을 생성, 액세스 또는 삭제할 수 있는지 여부를 결정합니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책을 시작하고 최소 권한 권한으로 전환 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반적인 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. 에서 사용할 수 있습니다 AWS 계정. 사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 추가로 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 관리AWS 형 정책 또는 AWS 작업 함수에 대한 관리형 정책을 참조하세요.
- 최소 권한 적용 - IAM 정책으로 권한을 설정할 때 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합

니다. 를 사용하여 권한을 적용하는 IAM 방법에 대한 자세한 내용은 IAM 사용 설명서의 에서 정책 및 권한을 IAM 참조하세요.

- IAM 정책의 조건을 사용하여 액세스 추가 제한 - 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어 정책 조건을 작성하여 를 사용하여 모든 요청을 전송하도록 지정할 수 있습니다SSL. AWS 서비스와 같은 특정 를 통해 서비스 작업을 사용하는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다 AWS CloudFormation. 자세한 내용은 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 사용하여 IAM 정책을 검증하여 안전하고 기능적인 권한을 보장합니다. IAM Access Analyzer는 정책이 정책 언어(JSON) 및 IAM 모범 사례를 준수하도록 새 정책 및 기존 IAM 정책을 검증합니다. IAM Access Analyzer는 안전하고 기능적인 정책을 작성하는 데 도움이 되는 100개 이상의 정책 확인 및 실행 가능한 권장 사항을 제공합니다. 자세한 내용은 IAM 사용 설명서의 IAM Access Analyzer를 사용한 정책 검증을 참조하세요.
- 다중 인증 필요(MFA) - 에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 MFA 위해 를 AWS 계정켭니다. API 작업을 호출할 MFA 때 를 요구하려면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 를 사용한 보안 API 액세스를 MFA 참조하세요.

의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 의 보안 모범 사례를 IAM IAM참조하세요.

InfluxDB 콘솔용 Timestream 사용

Amazon Timestream for InfluxDB 콘솔에 액세스하려면 최소 권한 집합이 있어야 합니다. 이러한 권한을 통해 에서 InfluxDB 리소스의 Timestream에 대한 세부 정보를 나열하고 볼 수 있어야 합니다 AWS 계정. 최소 필수 권한보다 더 제한적인 자격 증명 기반 정책을 만들면 콘솔이 해당 정책에 연결된 엔터티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

AWS CLI 또는 에만 전화를 거는 사용자에 대해 최소 콘솔 권한을 허용할 필요는 없습니다 AWS API. 대신 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 허용합니다.

사용자와 역할이 여전히 Timestream for InfluxDB 콘솔을 사용할 수 있도록 하려면 Timestream for InfluxDB `ConsoleAccess` 또는 `ReadOnly` AWS 관리형 정책을 엔터티에 연결합니다. 자세한 내용은 IAM 사용 설명서의 사용자에게 권한 추가를 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제에서는 IAM 사용자가 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI 또는 를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함되어 있습니다 AWS API.

```
{
    "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": "*"
        }
    ]
}
```

하나의 Amazon S3 버킷에 액세스

이 예제에서는 AWS 계정의 IAM 사용자에게 Amazon S3 버킷 중 하나인 에 대한 액세스 권한을 부여하려고 합니다`examplebucket`. 또한 사용자가 객체를 추가, 업데이트 및 삭제하도록 허용하려고 합니다.

이 정책에서는 `s3:PutObject`, `s3:GetObject` 및 `s3:DeleteObject` 권한을 사용자에게 부여할 뿐만 아니라 `s3:ListAllMyBuckets`, `s3:GetBucketLocation` 및 `s3:ListBucket` 권한도 부여합니다. 이러한 권한은 콘솔에 필요한 추가 권한입니다. 또한 콘솔에서 객체를 복사, 자르기 및 붙여넣

기를 할 수 있으려면 s3:PutObjectAcl 및 s3:GetObjectAcl 작업이 필요합니다. 사용자에게 권한을 부여하고 콘솔을 사용하여 권한을 테스트하는 예제 연습은 예제 연습: 사용자 정책을 사용하여 버킷에 대한 액세스 제어를 참조하세요.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Sid":"ListBucketsInConsole",
         "Effect":"Allow",
         "Action":[
            "s3:ListAllMyBuckets"
         ],
         "Resource":"arn:aws:s3:::*"
      },
      {
         "Sid":"ViewSpecificBucketInfo",
         "Effect":"Allow",
         "Action":[
            "s3:ListBucket",
            "s3:GetBucketLocation"
         ],
         "Resource":"arn:aws:s3:::examplebucket"
      },
      {
         "Sid":"ManageBucketContents",
         "Effect":"Allow",
         "Action":[
            "s3:PutObject",
            "s3:PutObjectAcl",
            "s3:GetObject",
            "s3:GetObjectAcl",
            "s3:DeleteObject"
         ],
         "Resource":"arn:aws:s3:::examplebucket/*"
      }
   ]
}
```

모든 작업 허용

다음은 InfluxDB용 Timestream의 모든 작업을 허용하는 샘플 정책입니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream-influxdb:*"
            ],
            "Resource": "*"
        }
    ]
}
```

DB 인스턴스 생성, 설명, 삭제 및 업데이트

다음 샘플 정책을 통해 사용자는 DB 인스턴스를 생성, 설명, 삭제 및 업데이트할 수 있습니다sampleDB.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "timestream-influxdb:CreateDbInstance",
                "timestream-influxdb:GetDbInstance",
                "timestream-influxdb:DeleteDbInstance",
                "timestream-influxdb:UpdateDbInstance"
            ],
            "Resource": "arn:aws:timestream-influxdb:us-east-1:<account_ID>:dbinstance/
sampleDB"
        }
    ]
}
```

InfluxDB 자격 증명 및 액세스를 위한 Amazon Timestream 문제 해결

다음 정보를 사용하여 Timestream for InfluxDB 및 작업 시 발생할 수 있는 일반적인 문제를 진단하고 해결할 수 있습니다IAM.

주제

- InfluxDB용 Timestream에서 작업을 수행할 권한이 없습니다.
- 내 AWS 계정 외부의 사람들이 내 Timestream for InfluxDB 리소스에 액세스하도록 허용하고 싶습니다.

InfluxDB용 Timestream에서 작업을 수행할 권한이 없습니다.

에 작업을 수행할 권한이 없다고 AWS Management Console 표시되면 관리자에게 문의하여 지원을 받아야 합니다. 관리자는 사용자 이름과 비밀번호를 제공한 사람입니다.

다음 예제 오류는 `mateojackson` 사용자가 콘솔을 사용하여 가상 *`my-example-widget`* 리소스에 대한 세부 정보를 보려고 하지만 가상 `timestream-influxdb:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 timestream-influxdb:GetWidget on resource: my-example-widget
```

이 경우 Mateo는 *`my-example-widget`* 작업을 사용하여 `timestream-influxdb:`*`GetWidget`* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

내 AWS 계정 외부의 사람들이 내 Timestream for InfluxDB 리소스에 액세스하도록 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(ACLs)을 지원하는 서비스의 경우 이러한 정책을 사용하여 사용자에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하세요.

- 에서 DB 인스턴스에 대한 액세스 제어 VPC
- Timestream for InfluxDB가 이러한 기능을 지원하는지 알아보려면 Amazon Timestream for InfluxDB가 에서 작동하는 방법을 IAM참조하세요.
- 소유한 AWS 계정에서 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 소유한 다른 AWS 계정의 IAM 사용자에게 액세스 권한 제공을 참조하세요.
- 타사 AWS 계정에 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 소유 AWS 계정에 대한 액세스 권한 제공을 참조하세요.

- 자격 증명 페더레이션을 통해 액세스를 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부 인증 사용자에게 액세스 제공(자격 증명 페더레이션)을 참조하세요.
- 교차 계정 액세스에 역할 및 리소스 기반 정책을 사용하는 차이점을 알아보려면 IAM 사용 설명서의 리소스 기반 정책과 IAM 역할이 어떻게 다른지 참조하세요.

에서 DB 인스턴스에 대한 액세스 제어 VPC

Amazon Virtual Private Cloud(Amazon VPC)를 사용하면 Amazon Timestream for InfluxDB DB 인스턴스와 같은 AWS 리소스를 가상 프라이빗 클라우드()로 시작할 수 있습니다VPC. Amazon 를 사용하면 가상 네트워킹 환경을 VPC제어할 수 있습니다. 자기만의 IP 주소 범위를 선택하고, 서브넷을 생성하고, 라우팅 및 액세스 제어 목록을 구성할 수 있습니다.

VPC 보안 그룹은 내부의 DB 인스턴스에 대한 액세스를 제어합니다VPC. 각 VPC 보안 그룹 규칙을 사용하면 특정 소스가 해당 VPC 보안 그룹과 연결된 의 DB 인스턴스에 액세스할 VPC 수 있습니다. 소스는 주소 범위(예: 203.0.113.0/24) 또는 다른 VPC 보안 그룹일 수 있습니다. VPC 보안 그룹을 소스로 지정하면 소스 VPC 보안 그룹을 사용하는 모든 인스턴스(일반적으로 애플리케이션 서버)에서 들어오는 트래픽을 허용합니다. DB 인스턴스에 연결을 시도하기 전에 사용 사례에 VPC 맞게 를 구성합니다. 다음은 에서 DB 인스턴스에 액세스하는 일반적인 시나리오입니다VPC.

동일한 의 Amazon 인스턴스에서 VPC 액세스하는 의 DB EC2 인스턴스 VPC

에서 DB 인스턴스의 일반적인 사용은 동일한 의 인스턴스에서 실행 중인 애플리케이션 서버와 데이터를 VPC 공유하는 EC2 것입니다VPC. 인스턴스는 DB EC2 인스턴스와 상호 작용하는 애플리케이션으로 웹 서버를 실행할 수 있습니다.

다른 의 인스턴스에서 VPC 액세스하는 의 DB EC2 인스턴스 VPC

경우에 따라 DB 인스턴스가 액세스에 사용하는 EC2 인스턴스와 VPC 다른 인스턴스에 있는 경우도 있습니다. 그렇다면 VPC 피어링을 사용하여 DB 인스턴스에 액세스할 수 있습니다.

인터넷을 통해 클라이언트 애플리케이션에서 VPC 액세스하는 의 DB 인스턴스

인터넷을 통해 VPC 클라이언트 애플리케이션에서 의 DB 인스턴스에 액세스하려면 단일 퍼블릭 서브넷VPC으로 를 구성하고 퍼블릭 서브넷을 사용하여 DB 인스턴스를 생성합니다. 또한 인터넷을 통한 통신을 활성화VPC하도록 에서 인터넷 게이트웨이를 구성합니다. 외부에서 DB 인스턴스에 연결하려면 DB 인스턴스VPC에 공개적으로 액세스할 수 있어야 합니다. 또한 DB 인스턴스 보안 그룹의 인바운드 규칙을 사용하여 액세스 권한을 부여해야 하며, 기타 요구 사항을 충족해야 합니다.

VPC 보안 그룹에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 보안 그룹을 참조하세요.

InfluxDB DB 인스턴스용 Timestream에 연결하는 방법에 대한 자세한 내용은 섹션을 참조하세요Amazon Timestream for InfluxDB DB 인스턴스에 연결.

보안 그룹 시나리오

에서 DB 인스턴스의 일반적인 사용은 동일한 의 Amazon EC2 인스턴스에서 실행되는 애플리케이션 서버와 데이터를 VPC 공유하는 것이며VPC, 이는 외부의 클라이언트 애플리케이션에서 액세스합니다 VPC. 이 시나리오에서는 InfluxDB용 Timestream 및 VPC 페이지의 AWS Management Console 또는 InfluxDB 및 EC2 API 작업용 Timestream을 사용하여 필요한 인스턴스 및 보안 그룹을 생성합니다.

1. VPC 보안 그룹(예: `sg-0123ec2example`)을 생성하고 클라이언트 애플리케이션의 IP 주소를 소스로 사용하는 인바운드 규칙을 정의합니다. 이 보안 그룹을 사용하면 클라이언트 애플리케이션이 이 보안 그룹을 사용하는 의 EC2 인스턴스에 연결할 VPC 수 있습니다.
2. 애플리케이션의 EC2 인스턴스를 생성하고 이전 단계에서 생성한 VPC 보안 그룹(`sg-0123ec2example`)에 EC2 인스턴스를 추가합니다.
3. 두 번째 VPC 보안 그룹(예: `sg-6789rdsexample`)을 생성하고 1단계(`sg-0123ec2example`)에서 생성한 VPC 보안 그룹을 소스로 지정하여 새 규칙을 생성합니다.
4. 새 DB 인스턴스를 생성하고 이전 단계에서 생성한 VPC 보안 그룹(`sg-6789rdsexample`)에 DB 인스턴스를 추가합니다. DB를 생성할 때 3단계에서 생성한 VPC 보안 그룹(`sg-6789rdsexample`) 규칙에 지정된 것과 동일한 포트 번호를 사용합니다.

VPC 보안 그룹 생성

VPC 콘솔을 사용하여 DB 인스턴스의 VPC 보안 그룹을 생성할 수 있습니다. 보안 그룹 생성에 대한 자세한 내용은 Amazon Virtual Private Cloud 사용 설명서의 보안 그룹을 참조하세요.

보안 그룹과 DB 인스턴스의 연결

InfluxDB 콘솔용 Timestream, InfluxDB용 `UpdateDBInstance` Timestream InfluxDB API또는 `update-db-instance` AWS CLI 명령에서 업데이트를 사용하여 보안 그룹을 DB 인스턴스와 연결할 수 있습니다.

다음 CLI 예제는 특정 VPC 보안 그룹을 연결하고 DB 인스턴스에서 DB 보안 그룹을 제거합니다.

```
aws timestream-influxdb update-db-instance --identifier dbName --vpc-security-group-
ids sg-ID
```

DB 인스턴스 수정에 대한 자세한 내용은 DB 인스턴스 업데이트 단원을 참조하세요.

Amazon Timestream for InfluxDB에 서비스 연결 역할 사용

Amazon Timestream for InfluxDB는 AWS Identity and Access Management (IAM) 서비스 연결 역할을 사용합니다. 서비스 연결 역할은 Amazon Timestream for InfluxDB와 같은 AWS 서비스에 직접 연결되는 고유한 유형의 IAM 역할입니다. Amazon Timestream for InfluxDB 서비스 연결 역할은 Amazon Timestream for InfluxDB 에서 사전 정의합니다. 여기에는 서비스가 dbinstances를 대신하여 AWS 서비스를 호출하는 데 필요한 모든 권한이 포함됩니다.

서비스 연결 역할을 사용하면 필요한 권한을 수동으로 추가할 필요가 없으므로 InfluxDB용 Amazon Timestream을 더 쉽게 설정할 수 있습니다. 역할은 이미 AWS 계정 내에 있지만 Amazon Timestream for InfluxDB 사용 사례에 연결되어 있으며 사전 정의된 권한이 있습니다. Amazon Timestream for InfluxDB만 이러한 역할을 수임할 수 있으며 이러한 역할만 사전 정의된 권한 정책을 사용할 수 있습니다. 먼저 역할의 관련 리소스를 삭제해야만 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 액세스하는 데 필요한 권한을 실수로 제거할 수 없으므로 Amazon Timestream for InfluxDB 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 다른 서비스에 대한 자세한 내용은 AWS 를 IAM 참조하고 서비스 연결 역할 열에서 예인 서비스를 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예 링크를 선택합니다.

목차

Amazon Timestream for InfluxDB에 대한 서비스 연결 역할 권한

Amazon Timestream for InfluxDB는 AmazonTimestreamInfluxDBServiceRolePolicy 라는 서비스 연결 역할을 사용합니다. 이 정책은 Timestream for InfluxDB가 클러스터 관리에 필요한 경우 사용자를 대신하여 AWS 리소스를 관리할 수 있도록 허용합니다.

AmazonTimestreamInfluxDBServiceRolePolicy 서비스 연결 역할 권한 정책은 Amazon Timestream for InfluxDB가 지정된 리소스에 대해 다음 작업을 완료할 수 있도록 허용합니다.

```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "DescribeNetworkStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:DescribeSubnets",
    "ec2:DescribeVpcs",
    "ec2:DescribeNetworkInterfaces"
   ],
   "Resource": "*"
  },
  {
   "Sid": "CreateEniInSubnetStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterface"
   ],
   "Resource": [
    "arn:aws:ec2:*:*:subnet/*",
    "arn:aws:ec2:*:*:security-group/*"
   ]
  },
  {
   "Sid": "CreateEniStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterface"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:RequestTag/AmazonTimestreamInfluxDBManaged": "false"
```

```
    }
   }
  },
  {
   "Sid": "CreateTagWithEniStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateTags"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:RequestTag/AmazonTimestreamInfluxDBManaged": "false"
    },
    "StringEquals": {
     "ec2:CreateAction": [
      "CreateNetworkInterface"
     ]
    }
   }
  },
  {
   "Sid": "ManageEniStatement",
   "Effect": "Allow",
   "Action": [
    "ec2:CreateNetworkInterfacePermission",
    "ec2:DeleteNetworkInterface"
   ],
   "Resource": "arn:aws:ec2:*:*:network-interface/*",
   "Condition": {
    "Null": {
     "aws:ResourceTag/AmazonTimestreamInfluxDBManaged": "false"
    }
   }
  },
  {
   "Sid": "PutCloudWatchMetricsStatement",
   "Effect": "Allow",
   "Action": [
    "cloudwatch:PutMetricData"
   ],
   "Condition": {
    "StringEquals": {
     "cloudwatch:namespace": [
```

```
      "AWS/Timestream/InfluxDB",
      "AWS/Usage"
     ]
    }
   },
   "Resource": [
    "*"
   ]
  },
  {
   "Sid": "ManageSecretStatement",
   "Effect": "Allow",
   "Action": [
    "secretsmanager:CreateSecret",
    "secretsmanager:DeleteSecret"
   ],
   "Resource": [
    "arn:aws:secretsmanager:*:*:secret:READONLY-InfluxDB-auth-parameters-*"
   ],
   "Condition": {
    "StringEquals": {
     "aws:ResourceAccount": "${aws:PrincipalAccount}"
    }
   }
  }
 ]
}
```

IAM엔터티가 서비스 연결 역할을 생성 AmazonTimestreamInfluxDBServiceRolePolicy하도록 허용하려면

해당 IAM엔터티에 대한 권한에 다음 정책 문을 추가합니다.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:CreateServiceLinkedRole",
        "iam:PutRolePolicy"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/
timestreamforinfluxdb.amazonaws.com/AmazonTimestreamInfluxDBServiceRolePolicy*",
    "Condition": {"StringLike": {"iam:AWS ServiceName":
 "timestreamforinfluxdb.amazonaws.com"}}
```

```
}
```

IAM엔터티가 서비스 연결 역할을 삭제 AmazonTimestreamInfluxDBServiceRolePolicy하도록 허용하려면

해당 IAM엔터티에 대한 권한에 다음 정책 문을 추가합니다.

```
{
    "Effect": "Allow",
    "Action": [
        "iam:DeleteServiceLinkedRole",
        "iam:GetServiceLinkedRoleDeletionStatus"
    ],
    "Resource": "arn:aws:iam::*:role/aws-service-role/
timestreamforinfluxdb.amazonaws.com/AmazonTimestreamInfluxDBServiceRolePolicy*",
    "Condition": {"StringLike": {"iam:AWS ServiceName":
 "timestreamforinfluxdb.amazonaws.com"}}
}
```

또는 AWS 관리형 정책을 사용하여 Amazon Timestream for InfluxDB 에 대한 전체 액세스를 제공할 수 있습니다.

서비스 연결 역할 생성(IAM)

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. DB 인스턴스를 생성하면 Amazon Timestream for InfluxDB가 서비스 연결 역할을 생성합니다.

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. DB 인스턴스를 생성하면 Amazon Timestream for InfluxDB가 서비스 연결 역할을 다시 생성합니다.

Amazon Timestream for InfluxDB에 대한 서비스 연결 역할 설명 편집

Amazon Timestream for InfluxDB에서는 서비스 연결 역할을 편집할 AmazonTimestreamInfluxDBServiceRolePolicy 수 없습니다. 서비스 링크 역할을 생성한 후에는 다양한 개체가 역할을 참조할 수 있기 때문에 역할 이름을 변경할 수 없습니다. 그러나 를 사용하여 역할에 대한 설명을 편집할 수 있습니다IAM.

서비스 연결 역할 설명 편집(IAM 콘솔)

IAM 콘솔을 사용하여 서비스 연결 역할 설명을 편집할 수 있습니다.

서비스 연결 역할의 설명을 편집하려면(콘솔 사용)

1. IAM 콘솔의 왼쪽 탐색 창에서 역할 을 선택합니다.
2. 변경할 역할 이름을 선택합니다.
3. 역할 설명의 맨 오른쪽에서 편집을 선택합니다.
4. 상자에 새 설명을 입력하고 저장을 선택합니다.

서비스 연결 역할 설명 편집(IAM CLI)

의 IAM 작업을 사용하여 서비스 연결 역할 설명을 AWS Command Line Interface 편집할 수 있습니다.

서비스 연결 역할에 대한 설명을 변경하려면(CLI)

1. (선택 사항) 역할에 대한 현재 설명을 보려면 IAM 작업에 AWS CLI 를 사용합니다get-role.

   Example

   ```
   $ aws iam get-role --role-name AmazonTimestreamInfluxDBServiceRolePolicy
   ```

   가 아닌 역할 이름을 사용하여 CLI 작업이 있는 역할을 ARN참조합니다. 예를 들어 역할에 가 있는 경우 역할을 로 ARN `arn:aws:iam::123456789012:role/myrole`참조하세요**myrole**.
2. 서비스 연결 역할의 설명을 업데이트하려면 IAM 작업 AWS CLI 용 를 사용합니다update-role-description.

   Linux 및 MacOS

   ```
   $ aws iam update-role-description \
       --role-name AmazonTimestreamInfluxDBServiceRolePolicy \
       --description "new description"
   ```

   Windows

   ```
   $ aws iam update-role-description ^
       --role-name AmazonTimestreamInfluxDBServiceRolePolicy ^
       --description "new description"
   ```

서비스 연결 역할 설명 편집(IAM API)

IAM API 를 사용하여 서비스 연결 역할 설명을 편집할 수 있습니다.

서비스 연결 역할에 대한 설명을 변경하려면(API)

1. (선택 사항) 역할에 대한 현재 설명을 보려면 IAM API 작업을 사용합니다.GetRole.

   Example

   ```
   https://iam.amazonaws.com/
      ?Action=GetRole
      &RoleName=AmazonTimestreamInfluxDBServiceRolePolicy
      &Version=2010-05-08
      &AUTHPARAMS
   ```

2. 역할의 설명을 업데이트하려면 IAM API 작업을 사용합니다.UpdateRoleDescription.

   Example

   ```
   https://iam.amazonaws.com/
      ?Action=UpdateRoleDescription
      &RoleName=AmazonTimestreamInfluxDBServiceRolePolicy
      &Version=2010-05-08
      &Description="New description"
   ```

Amazon Timestream for InfluxDB에 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. 단, 삭제 전에 서비스 연결 역할을 정리해야 합니다.

Amazon Timestream for InfluxDB는 서비스 연결 역할을 삭제하지 않습니다.

서비스 연결 역할 정리

IAM 를 사용하여 서비스 연결 역할을 삭제하려면 먼저 역할에 연결된 리소스(클러스터)가 없는지 확인합니다.

서비스 연결 역할에 IAM 콘솔에 활성 세션이 있는지 확인하려면

1. 에 로그인 AWS Management Console 하고 에서 IAM 콘솔을 엽니다https://console.aws.amazon.com/iam/.
2. IAM 콘솔의 왼쪽 탐색 창에서 역할 을 선택합니다. 그런 다음 역할의 이름(확인란 아님)을 AmazonTimestreamInfluxDBServiceRolePolicy 선택합니다.
3. 선택한 역할의 요약 페이지에서 Access Advisor 탭을 선택합니다.
4. 액세스 관리자(Access Advisor) 탭에서 서비스 연결 역할의 최근 활동을 검토합니다.

서비스 연결 역할 삭제(IAM 콘솔)

IAM 콘솔을 사용하여 서비스 연결 역할을 삭제할 수 있습니다.

서비스 연결 역할을 삭제하는 방법(콘솔)

1. 에 로그인 AWS Management Console 하고 에서 IAM 콘솔을 엽니다https://console.aws.amazon.com/iam/.
2. IAM 콘솔의 왼쪽 탐색 창에서 역할 을 선택합니다. 그런 다음 삭제할 역할의 이름이나 행이 아닌 이름 옆에 있는 확인란을 선택합니다.
3. 페이지 상단의 역할 작업에서 역할 삭제를 선택합니다.
4. 확인 페이지에서 선택한 각 역할이 서비스에 마지막으로 액세스한 시점을 보여주는 AWS 서비스 마지막 액세스 데이터를 검토합니다. 이를 통해 역할이 현재 활동 중인지를 확인할 수 있습니다. 계속 진행하려면 예, 삭제합니다(Yes, Delete)를 선택하여 삭제할 서비스 연결 역할을 제출합니다.
5. IAM 콘솔 알림을 보고 서비스 연결 역할 삭제 진행 상황을 모니터링합니다. IAM 서비스 연결 역할 삭제는 비동기식이므로 삭제를 위해 역할을 제출한 후 삭제 작업이 성공하거나 실패할 수 있습니다. 태스크에 실패할 경우 알림의 세부 정보 보기 또는 리소스 보기를 선택하면 삭제 실패 이유를 확인할 수 있습니다.

서비스 연결 역할 삭제(IAM CLI)

의 IAM 작업을 사용하여 서비스 연결 역할을 AWS Command Line Interface 삭제할 수 있습니다.

서비스 연결 역할을 삭제하려면(CLI)

1. 삭제할 서비스 연결 역할의 이름을 모르는 경우 다음 명령을 입력합니다. 이 명령은 계정의 역할과 해당 Amazon 리소스 이름(ARNs)을 나열합니다.

```
$ aws iam get-role --role-name role-name
```

가 아닌 역할 이름을 사용하여 CLI 작업이 있는 역할을 ARN참조합니다. 예를 들어 역할에 ARN 이 있는 경우 역할을 라고 `arn:aws:iam::123456789012:role/myrole`합니다**myrole**.

2. 서비스 연결 역할이 사용되지 않거나 연결된 리소스가 없는 경우에는 서비스 연결 역할을 삭제할 수 없으므로 삭제 요청을 제출해야 합니다. 이러한 조건이 충족되지 않으면 요청이 거부될 수 있습니다. 삭제 태스크 상태를 확인하려면 응답의 `deletion-task-id`(을)를 캡처해야 합니다. 다음을 입력하여 서비스 연결 역할 삭제 요청을 제출합니다.

```
$ aws iam delete-service-linked-role --role-name role-name
```

3. 다음을 입력하여 삭제 작업의 상태를 확인합니다.

```
$ aws iam get-service-linked-role-deletion-status --deletion-task-id deletion-task-id
```

삭제 태스크는 `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED` 또는 `FAILED` 상태일 수 있습니다. 삭제에 실패할 경우 문제를 해결할 수 있도록 실패 이유가 호출에 반환됩니다.

서비스 연결 역할 삭제(IAM API)

IAM API 를 사용하여 서비스 연결 역할을 삭제할 수 있습니다.

서비스 연결 역할을 삭제하려면(API)

1. 서비스 연결 롤에 대한 삭제 요청을 제출하려면 DeleteServiceLinkedRole. 요청에서 역할 이름을 지정합니다.

   서비스 연결 역할이 사용되지 않거나 연결된 리소스가 없는 경우에는 서비스 연결 역할을 삭제할 수 없으므로 삭제 요청을 제출해야 합니다. 이러한 조건이 충족되지 않으면 요청이 거부될 수 있습니다. 삭제 태스크 상태를 확인하려면 응답의 `DeletionTaskId`(을)를 캡처해야 합니다.

2. 삭제 상태를 확인하려면 GetServiceLinkedRoleDeletionStatus. 요청에서 를 지정합니다`DeletionTaskId`.

   삭제 태스크는 `NOT_STARTED`, `IN_PROGRESS`, `SUCCEEDED` 또는 `FAILED` 상태일 수 있습니다. 삭제에 실패할 경우 문제를 해결할 수 있도록 실패 이유가 호출에 반환됩니다.

InfluxDB Service-Linked Roles용 Amazon Timestream에 지원되는 리전

Amazon Timestream for InfluxDB는 서비스를 사용할 수 있는 모든 리전에서 서비스 연결 역할 사용을 지원합니다. 자세한 내용은 AWS 서비스 엔드포인트를 참조하세요.

AWS Amazon Timestream for InfluxDB에 대한 관리형 정책

사용자, 그룹 및 역할에 권한을 추가하려면 직접 정책을 작성하는 것보다 AWS 관리형 정책을 사용하는 것이 더 쉽습니다. 팀에 필요한 권한만 제공하는 IAM 고객 관리형 정책을 생성하는 데는 시간과 전문성이 필요합니다. 빠르게 시작하려면 AWS 관리형 정책을 사용할 수 있습니다. 이러한 정책은 일반적인 사용 사례를 다루며 AWS 계정에서 사용할 수 있습니다. AWS 관리형 정책에 대한 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

AWS 서비스는 AWS 관리형 정책을 유지 관리하고 업데이트합니다. AWS 관리형 정책의 권한은 변경할 수 없습니다. 서비스에서 때때로 추가 권한을 AWS 관리형 정책에 추가하여 새로운 기능을 지원합니다. 이 유형의 업데이트는 정책이 연결된 모든 ID(사용자, 그룹 및 역할)에 적용됩니다. 서비스는 새로운 기능이 시작되거나 새 작업을 사용할 수 있을 때 AWS 관리형 정책에 업데이트됩니다. 서비스는 AWS 관리형 정책에서 권한을 제거하지 않으므로 정책 업데이트가 기존 권한을 손상시키지 않습니다.

또한 여러 서비스에 걸쳐 있는 작업 기능에 대한 관리형 정책을 AWS 지원합니다. 예를 들어 ReadOnlyAccess AWS 관리형 정책은 모든 AWS 서비스 및 리소스에 대한 읽기 전용 액세스를 제공합니다. 서비스가 새 기능을 시작하면 는 새 작업 및 리소스에 대한 읽기 전용 권한을 AWS 추가합니다. 작업 함수 정책의 목록 및 설명은 IAM 사용 설명서의 AWS 작업 함수에 대한 관리형 정책을 참조하세요.

AWS 관리형 정책: AmazonTimestreamInfluxDBServiceRolePolicy

AmazonTimestreamInfluxDBServiceRolePolicy AWS 관리형 정책을 계정의 자격 증명에 연결할 수 없습니다. 이 정책은 AWS TimestreamforInfluxDB 서비스 연결 역할의 일부입니다. 이 역할을 통해 서비스는 계정의 네트워크 인터페이스와 보안 그룹을 관리할 수 있습니다.

InfluxDB용 Timestream은 이 정책의 권한을 사용하여 EC2 보안 그룹 및 네트워크 인터페이스를 관리합니다. 이는 InfluxDB DB 인스턴스의 Timestream을 관리하는 데 필요합니다.

형식으로 이 정책을 검토하려면 섹션을 참조JSON하세요AmazonTimestreamInfluxDBServiceRolePolicy.

AWS-Amazon Timestream for InfluxDB에 대한 관리형 정책

AWS 는 에서 생성 및 관리하는 독립 실행형 IAM 정책을 제공하여 많은 일반적인 사용 사례를 처리합니다 AWS. 관리형 정책은 사용자가 필요한 권한을 조사할 필요가 없도록 일반 사용 사례에 필요한 권한을 부여합니다. 자세한 내용은 IAM 사용 설명서의 AWS 관리형 정책을 참조하세요.

계정의 사용자에게 연결할 수 있는 다음 AWS 관리형 정책은 Timestream for InfluxDB 에 따라 다릅니다.

AmazonTimestreamInfluxDBFullAccess

`AmazonTimestreamInfluxDBFullAccess` 정책을 IAM 자격 증명에 연결할 수 있습니다. 이 정책은 InfluxDB 리소스에 대한 모든 Timestream에 대한 전체 액세스를 허용하는 관리 권한을 부여합니다.

또한 사용자 지정 IAM 정책을 생성하여 Amazon Timestream for InfluxDB API 작업에 대한 권한을 허용할 수 있습니다. 이러한 권한이 필요한 IAM 사용자 또는 그룹에 이러한 사용자 지정 정책을 연결할 수 있습니다.

이 정책을 JSON 형식으로 검토하려면 섹션을 참조하세요AmazonTimestreamInfluxDBFullAccess.

AWS 관리형 정책에 대한 InfluxDB 업데이트의 타임스트림

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후 InfluxDB용 Timestream의 AWS 관리형 정책 업데이트에 대한 세부 정보를 봅니다. 이 페이지의 변경 사항에 대한 자동 알림을 받으려면 Timestream for InfluxDB 문서 기록 페이지에서 RSS 피드를 구독하세요.

변경 사항	설명	날짜
AmazonTimestreamInfluxDBFullAccess - 기존 정책에 대한 업데이트	기존 `AmazonTimestreamInfluxDBFullAccess` 관	10/08/2024

변경 사항	설명	날짜
	리형 정책에 `ec2:DescribeRouteTables` 작업을 추가했습니다. 이 작업은 라우팅 테이블을 설명하는 데 사용됩니다.	
AWS 관리형 정책: AmazonTimestreamInfluxDBServiceRolePolicy - 새 정책	Amazon Timestream for InfluxDB는 서비스가 계정의 네트워크 인터페이스 및 보안 그룹을 관리할 수 있도록 허용하는 새 정책을 추가했습니다.	03/14/2024
AmazonTimestreamInfluxDBFullAccess - 새 정책	Amazon Timestream for InfluxDB는 Amazon Timestream InfluxDB 인스턴스를 생성, 업데이트, 삭제 및 나열하고 파라미터 그룹을 생성 및 나열할 수 있는 전체 관리 액세스 권한을 제공하는 새 정책을 추가했습니다.	03/14/2024

VPC 엔드포인트를 통해 InfluxDB의 Timestream에 연결

가상 프라이빗 클라우드()의 프라이빗 인터페이스 엔드포인트를 통해 Timestream for InfluxDB에 직접 연결할 수 있습니다VPC. 인터페이스 VPC 엔드포인트를 사용하면 VPC와 Timestream for InfluxDB 간의 통신이 전적으로 AWS 네트워크 내에서 수행됩니다.

InfluxDB의 Timestream은 에서 구동되는 Amazon Virtual Private Cloud(AmazonVPC) 엔드포인트를 지원합니다AWS PrivateLink. 각 VPC 엔드포인트는 VPC 서브넷에 프라이빗 IP 주소가 있는 하나 이상의 탄력적 네트워크 인터페이스(ENIs)로 표시됩니다.

인터페이스 VPC 엔드포인트는 인터넷 게이트웨이, NAT 디바이스, 연결 또는 VPN 연결 없이 를 Timestream for InfluxDB에 VPC 직접 AWS Direct Connect 연결합니다. 의 인스턴스는 Timestream for InfluxDB 와 통신하는 데 퍼블릭 IP 주소가 필요하지 VPC 않습니다.

리전

InfluxDB용 Timestream은 InfluxDB용 Timestream이 지원되는 모든 AWS 리전 에서 VPC 엔드포인트 및 VPC 엔드포인트 정책을 지원합니다.

주제

- InfluxDB VPC 엔드포인트의 Timestream 고려 사항
- InfluxDB용 Timestream에 대한 VPC 엔드포인트 생성
- InfluxDB VPC 엔드포인트용 Timestream에 연결
- VPC 엔드포인트에 대한 액세스 제어
- 정책 문에서 VPC 엔드포인트 사용
- VPC 엔드포인트 로깅

InfluxDB VPC 엔드포인트의 Timestream 고려 사항

InfluxDB용 Timestream에 대한 인터페이스 VPC 엔드포인트를 설정하기 전에 AWS PrivateLink 가이드의 인터페이스 엔드포인트 속성 및 제한 주제를 검토합니다.

VPC 엔드포인트에 대한 InfluxDB 지원을 위한 Timestream에는 다음이 포함됩니다.

- VPC 엔드포인트를 사용하여 에서 InfluxDB API 작업을 위한 모든 Timestream을 호출할 수 있습니다VPC.
- AWS CloudTrail 로그를 사용하여 VPC 엔드포인트를 통해 Timestream for InfluxDB 리소스 사용을 감사할 수 있습니다. 세부 정보는 VPC 엔드포인트 로깅을 참조하세요.

InfluxDB용 Timestream에 대한 VPC 엔드포인트 생성

Amazon VPC 콘솔 또는 Amazon 를 사용하여 Timestream for InfluxDB용 VPC 엔드포인트를 생성할 수 있습니다VPCAPI. 자세한 내용은 AWS PrivateLink 설명서의 인터페이스 엔드포인트 생성을 참조하십시오.

- Timestream for InfluxDB 에 대한 VPC 엔드포인트를 생성하려면 다음 서비스 이름을 사용합니다.

```
com.amazonaws.region.timestream-influxdb
```

예를 들어 미국 서부(오레곤) 리전(us-west-2)에서 서비스 이름은 다음과 같습니다.

```
com.amazonaws.us-west-2.timestream-influxdb
```

VPC 엔드포인트를 더 쉽게 사용할 수 있도록 VPC 엔드포인트의 프라이빗 DNS 이름을 활성화할 수 있습니다. DNS 이름 활성화 옵션을 선택하면 InfluxDB DNS 호스트 이름의 표준 Timestream 이 VPC 엔드포인트로 확인됩니다. 예를 들어 는 서비스 이름 에 연결된 VPC 엔드포인트로 해석`https://timestream-influxdb.us-west-2.amazonaws.com`합니다`com.amazonaws.us-west-2.timestream-influxdb`.

이 옵션을 사용하면 VPC 엔드포인트를 더 쉽게 사용할 수 있습니다. 및 는 AWS SDKs 기본적으로 표준 Timestream for InfluxDB DNS 호스트 이름을 AWS CLI 사용하므로 애플리케이션 및 명령URL에서 VPC 엔드포인트를 지정할 필요가 없습니다.

자세한 내용은 AWS PrivateLink 가이드의 인터페이스 엔드포인트를 통해 서비스 액세스를 참조하세요.

InfluxDB VPC 엔드포인트용 Timestream에 연결

AWS SDK, AWS CLI 또는 를 사용하여 VPC 엔드포인트를 통해 Timestream for InfluxDB에 연결할 수 있습니다 AWS Tools for PowerShell. VPC 엔드포인트를 지정하려면 해당 DNS 이름을 사용합니다.

VPC 엔드포인트를 생성할 때 프라이빗 호스트 이름을 활성화한 경우 CLI 명령 또는 애플리케이션 구성URL에서 VPC 엔드포인트를 지정할 필요가 없습니다. InfluxDB DNS 호스트 이름의 표준 Timestream은 VPC 엔드포인트로 확인됩니다. AWS CLI 및 는 기본적으로 이 호스트 이름을 SDKs 사용하므로 스크립트 및 애플리케이션의 아무 것도 변경하지 않고 VPC 엔드포인트를 사용하여 Timestream for InfluxDB 리전 엔드포인트에 연결할 수 있습니다.

프라이빗 호스트 이름을 사용하려면 의 `enableDnsHostnames` 및 `enableDnsSupport` 속성을 로 설정해야 VPC 합니다`true`. 이러한 속성을 설정하려면 ModifyVpcAttribute 작업을 사용합니다. 자세한 내용은 Amazon VPC 사용 설명서의 에 대한 DNS 속성 보기 및 업데이트를 VPC 참조하세요.

VPC 엔드포인트에 대한 액세스 제어

InfluxDB용 Timestream 의 VPC 엔드포인트에 대한 액세스를 제어하려면 VPC 엔드포인트 정책을 VPC 엔드포인트에 연결합니다. 엔드포인트 정책은 보안 주체가 VPC 엔드포인트를 사용하여 InfluxDB 리소스의 Timestream for InfluxDB 작업을 호출할 수 있는지 여부를 결정합니다 InfluxDB.

VPC 엔드포인트를 생성할 때 엔드포인트 정책을 생성할 수 있으며 언제든지 VPC 엔드포인트 정책을 변경할 수 있습니다. VPC 관리 콘솔 또는 CreateVpcEndpoint 또는 ModifyVpcEndpoint 작업을 사용합니다. AWS CloudFormation 템플릿 을 사용하여 VPC 엔드포인트 정책을 생성하고 변경할 수도 있습니다. VPC 관리 콘솔 사용에 대한 도움말은 AWS PrivateLink 가이드의 인터페이스 엔드포인트 생성 및 인터페이스 엔드포인트 수정을 참조하세요.

> **Note**
>
> InfluxDB의 Timestream은 2020년 7월부터 VPC 엔드포인트 정책을 지원합니다. VPC 해당 날짜 이전에 생성된 Timestream for InfluxDB의 엔드포인트에는 기본 VPC 엔드포인트 정책이 있지만 언제든지 변경할 수 있습니다.

주제

VPC 엔드포인트 정책 정보

VPC 엔드포인트를 사용하는 Timestream for InfluxDB 요청이 성공하려면 보안 주체에 두 소스의 권한이 필요합니다.

- IAM 정책은 보안 주체에게 리소스에 대한 작업을 호출할 수 있는 권한을 부여해야 합니다.
- VPC 엔드포인트 정책은 보안 주체에게 엔드포인트를 사용하여 요청할 수 있는 권한을 부여해야 합니다.

기본 VPC 엔드포인트 정책

모든 VPC 엔드포인트에는 VPC 엔드포인트 정책이 있지만 정책을 지정할 필요는 없습니다. 정책을 지정하지 않으면 기본 엔드포인트 정책은 엔드포인트의 모든 리소스에 대한 모든 보안 주체의 모든 작업을 허용합니다.

그러나 Timestream for InfluxDB 리소스의 경우 보안 주체는 IAM 정책에서 작업을 호출할 수 있는 권한도 있어야 합니다. 따라서 실제로 기본 정책에는 보안 주체가 리소스에서 작업을 호출할 수 있는 권한이 있는 경우 엔드포인트를 사용하여 호출할 수도 있다고 명시되어 있습니다.

```
{
  "Statement": [
    {
      "Action": "*",
```

```
      "Effect": "Allow",
      "Principal": "*",
      "Resource": "*"
    }
  ]
}
```

보안 주체가 허용된 작업의 하위 집합에만 VPC 엔드포인트를 사용하도록 허용하려면 VPC 엔드포인트 정책 을 생성하거나 업데이트합니다.

VPC 엔드포인트 정책 생성

VPC 엔드포인트 정책은 보안 주체가 VPC 엔드포인트를 사용하여 리소스에 대한 작업을 수행할 수 있는 권한이 있는지 여부를 결정합니다. Timestream for InfluxDB 리소스의 경우 보안 주체는 IAM 정책에서 작업을 수행할 수 있는 권한도 있어야 합니다.

각 VPC 엔드포인트 정책 문에는 다음 요소가 필요합니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업
- 작업을 수행할 수 있는 리소스

정책 문은 VPC 엔드포인트를 지정하지 않습니다. 대신 정책이 연결된 모든 VPC 엔드포인트에 적용됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 사용하여 서비스에 대한 액세스 제어를 참조하세요.

AWS CloudTrail 는 VPC 엔드포인트를 사용하는 모든 작업을 기록합니다.

VPC 엔드포인트 정책 보기

VPC 엔드포인트에 대한 엔드포인트 정책을 보려면 VPC 관리 콘솔 또는 DescribeVpcEndpoints 작업을 사용합니다.

다음 AWS CLI 명령은 지정된 엔드포인트 ID로 VPC 엔드포인트에 대한 정책을 가져옵니다.

이 명령을 사용하기 앞서 예제 엔드포인트 ID를 계정의 유효한 ID로 바꿉니다.

```
$ aws ec2 describe-vpc-endpoints \
--query 'VpcEndpoints[?VpcEndpointId==`vpc-endpoint-id`].[PolicyDocument]'
```

```
--output text
```

정책 문에서 VPC 엔드포인트 사용

요청이 VPC 엔드포인트에서 오VPC거나 엔드포인트를 사용할 때 Timestream for InfluxDB 리소스 및 작업에 대한 액세스를 제어할 수 있습니다. 이렇게 하려면 IAM 정책 에서 다음 전역 조건 키 중 하나를 사용합니다.

- `aws:sourceVpce` 조건 키를 사용하여 VPC 엔드포인트를 기반으로 액세스를 부여하거나 제한합니다.
- `aws:sourceVpc` 조건 키를 사용하여 프라이빗 엔드포인트를 호스팅VPC하는 를 기반으로 액세스를 부여하거나 제한합니다.

> **Note**
>
> VPC 엔드포인트를 기반으로 키 정책 및 IAM 정책을 생성할 때 주의하십시오. 정책 설명에 특정 VPC 또는 VPC 엔드포인트에서 요청을 해야 하는 경우 사용자를 대신하여 Timestream for InfluxDB 리소스를 사용하는 통합 AWS 서비스의 요청이 실패할 수 있습니다.
> 또한 요청이 Amazon VPC 엔드포인트 에서 오면 `aws:sourceIP` 조건 키가 유효하지 않습니다. 요청을 VPC 엔드포인트로 제한하려면 `aws:sourceVpce` 또는 `aws:sourceVpc` 조건 키를 사용합니다. 자세한 내용은 AWS PrivateLink 가이드의 VPC 엔드포인트 및 VPC 엔드포인트 서비스에 대한 자격 증명 및 액세스 관리를 참조하세요.

이러한 전역 조건 키를 사용하여 특정 리소스에 의존하지 않는 CreateDbInstance 와 같은 작업에 대한 액세스를 제어할 수 있습니다.

VPC 엔드포인트 로깅

AWS CloudTrail 는 VPC 엔드포인트를 사용하는 모든 작업을 기록합니다. Timestream for InfluxDB에 대한 요청이 VPC 엔드포인트를 사용하는 경우 요청을 기록하는 AWS CloudTrail 로그 항목에 VPC 엔드포인트 ID가 나타납니다. 엔드포인트 ID를 사용하여 Timestream for InfluxDB VPC 엔드포인트의 사용을 감사할 수 있습니다.

그러나 CloudTrail 로그에는 다른 계정의 보안 주체가 요청한 작업 또는 다른 계정의 Timestream for InfluxDB 리소스 및 별칭에 대한 Timestream for InfluxDB 작업에 대한 요청이 포함되지 않습니다. 또한 를 보호하기 위해 VPC 엔드포인트 정책 에 의해 거부되지만 그렇지 않으면 허용되었을 VPC요청은 에 기록되지 않습니다AWS CloudTrail.

InfluxDB용 Timestream에서 로깅 및 모니터링

모니터링은 Timestream for InfluxDB 및 AWS 솔루션의 안정성, 가용성 및 성능을 유지하는 데 중요한 부분입니다. AWS 솔루션의 모든 부분에서 모니터링 데이터를 수집하여 멀티 포인트 장애가 발생할 경우 더 쉽게 디버깅할 수 있도록 해야 합니다. 그러나 InfluxDB용 Timestream 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 생성해야 합니다.

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

다음 단계는 다양한 시간과 다양한 부하 조건에서 성능을 측정하여 환경에서 InfluxDB 성능에 대한 정상 Timestream의 기준을 설정하는 것입니다. InfluxDB용 Timestream을 모니터링할 때 과거 모니터링 데이터를 저장하여 현재 성능 데이터와 비교하고, 정상적인 성능 패턴과 성능 이상을 식별하고, 문제를 해결하기 위한 방법을 고안할 수 있습니다.

기준선을 설정하려면 최소한 다음 항목을 모니터링해야 합니다.

- 시스템 오류 - 요청으로 인해 오류가 발생했는지 확인할 수 있습니다.

주제

- 모니터링 도구
- 를 사용하여 InfluxDB API 호출에 대한 Timestream 로깅 AWS CloudTrail

모니터링 도구

AWS 는 InfluxDB용 Timestream을 모니터링하는 데 사용할 수 있는 다양한 도구를 제공합니다. 이들 도구 중에는 모니터링을 자동으로 수행하도록 구성할 수 있는 도구도 있지만, 수동 작업이 필요한 도구도 있습니다. 모니터링 작업은 최대한 자동화하는 것이 좋습니다.

주제

- 자동 모니터링 도구

- 수동 모니터링 도구

자동 모니터링 도구

다음 자동 모니터링 도구를 사용하여 InfluxDB용 Timestream을 모니터링하고 문제가 발생할 때 보고할 수 있습니다.

- Amazon CloudWatch 경보 - 지정한 기간 동안 단일 지표를 관찰하고 여러 기간 동안 지정된 임계값을 기준으로 지표 값을 기반으로 하나 이상의 작업을 수행합니다. 작업은 Amazon Simple Notification Service(Amazon SNS) 주제 또는 Amazon EC2 Auto Scaling 정책으로 전송되는 알림입니다. CloudWatch 알람은 특정 상태라는 이유만으로 작업을 호출하지 않습니다. 상태는 변경되어 지정된 기간 동안 유지되어야 합니다. 자세한 내용은 Amazon을 사용한 모니터링 CloudWatch 단원을 참조하십시오.

수동 모니터링 도구

Timestream for InfluxDB 모니터링의 또 다른 중요한 부분은 CloudWatch 경보가 다루지 않는 항목을 수동으로 모니터링하는 것입니다. InfluxDB, CloudWatch Trusted Advisor및 기타 AWS Management Console 대시보드는 at-a-glance AWS 환경의 상태를 보여줍니다.

- CloudWatch 홈 페이지에는 다음이 표시됩니다.
  - 현재 경보 및 상태
  - 경보 및 리소스 그래프
  - 서비스 상태

  또한 CloudWatch 를 사용하여 다음을 수행할 수 있습니다.
  - 맞춤 대시보드를 생성하여 관심 있는 서비스 모니터링
  - 지표 데이터를 그래프로 작성하여 문제를 해결하고 추세 파악
  - 모든 AWS 리소스 지표 검색 및 찾아보기
  - 문제에 대해 알려주는 경보 생성 및 편집

를 사용하여 InfluxDB API 호출에 대한 Timestream 로깅 AWS CloudTrail

Timestream for InfluxDB는 Timestream for InfluxDB . CloudTrail captures Data Definition Language(DDL)가 이벤트로 Timestream for InfluxDB를 API 호출 AWS CloudTrail하는 사용자, 역할 또는 AWS 서비스의 작업 레코드를 제공하는 서비스인 와 통합됩니다. 캡처되는 호출에는 Timestream

for InfluxDB 콘솔의 호출과 Timestream for InfluxDB API 작업에 대한 코드 호출이 포함됩니다. 추적을 생성하는 경우 Timestream for InfluxDB 에 CloudTrail 대한 이벤트를 포함하여 Amazon Simple Storage Service(Amazon S3) 버킷에 이벤트를 지속적으로 전송할 수 있습니다. 추적을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록 에서 최신 이벤트를 볼 수 있습니다. 에서 수집한 정보를 사용하여 Timestream for InfluxDB 수행된 요청 CloudTrail, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

에 대한 자세한 내용은 AWS CloudTrail 사용 설명서 섹션을 CloudTrail참조하세요.

의 InfluxDB 정보에 대한 Timestream CloudTrail

CloudTrail 는 AWS 계정을 생성할 때 계정에서 활성화됩니다. Timestream for InfluxDB 에서 활동이 발생하면 해당 활동은 CloudTrail 이벤트 기록 의 다른 AWS 서비스 이벤트와 함께 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 이벤트 기록을 사용하여 CloudTrail 이벤트 보기를 참조하세요.

Timestream for InfluxDB 에 대한 이벤트를 포함하여 AWS 계정의 이벤트에 대한 지속적인 기록을 위해 추적을 생성합니다. 추적을 사용하면 CloudTrail 가 Amazon S3 버킷에 로그 파일을 전달할 수 있습니다. 기본적으로 콘솔에서 추적을 생성하면 추적이 모든 AWS 리전에 적용됩니다. 추적은 AWS 파티션의 모든 리전에서 이벤트를 기록하고 지정한 Amazon S3 버킷에 로그 파일을 전달합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 작업하도록 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 리전에서 CloudTrail 로그 파일 수신
- 여러 계정에서 CloudTrail 로그 파일 수신
- 데이터 이벤트 로깅

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부
- 역할 또는 페더레이션 사용자에 대한 임시 보안 자격 증명을 사용하여 요청이 생성되었는지 여부

- 다른 AWS 서비스에서 요청을 했는지 여부

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

Amazon Timestream for InfluxDB에 대한 규정 준수 검증

타사 감사자는 여러 규정 준수 프로그램의 일환으로 Amazon Timestream for InfluxDB의 보안 및 AWS 규정 준수를 평가합니다. 여기에는 다음이 포함됩니다.

- GDPR
- HIPAA
- PCI
- SOC

AWS 서비스 가 특정 규정 준수 프로그램의 범위 내에 있는지 알아보려면 AWS 서비스 규정 준수 프로그램 범위 참조하고 관심 있는 규정 준수 프로그램을 선택합니다. 일반 정보는 AWS 규정 준수 프로그램 참조하세요.

를 사용하여 타사 감사 보고서를 다운로드할 수 있습니다 AWS Artifact. 자세한 내용은 의 보고서 다운로드 AWS Artifact.

사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률 및 규정에 따라 AWS 서비스 결정됩니다. 는 규정 준수를 지원하는 다음 리소스를 AWS 제공합니다.

- 보안 및 규정 준수 빠른 시작 안내서 - 이 배포 안내서에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 에 기준 환경을 배포 AWS 하기 위한 단계를 제공합니다.
- Amazon Web Services의 HIPAA 보안 및 규정 준수를 위한 아키텍처 HIPAA- 이 백서에서는 기업이 AWS 를 사용하여 적격 애플리케이션을 생성하는 방법을 설명합니다.

> **Note**
>
> 모두 HIPAA 적합한 AWS 서비스 것은 아닙니다. 자세한 내용은 HIPAA 적격 서비스 참조를 참조하세요.

- AWS 규정 준수 리소스 - 이 워크북 및 가이드 모음은 해당 산업 및 위치에 적용될 수 있습니다.
- AWS 고객 규정 준수 가이드 - 규정 준수의 관점에서 공동 책임 모델을 이해합니다. 이 가이드는 여러 프레임워크(미국 국립표준기술연구소(), 결제카드 산업보안표준위원회(NIST), 국제표준화기구

(ISO) 포함)의 보안 제어에 대한 지침을 보호하고 AWS 서비스 매핑하는 모범 사례를 요약PCI합니다.

- 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 - AWS 서비스 에서 보안 상태를 포괄적으로 볼 수 있습니다 AWS. 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 참조하세요.
- Amazon GuardDuty – 의심스러운 활동 및 악의적인 활동이 있는지 환경을 모니터링하여 AWS 계정, 워크로드, 컨테이너 및 데이터에 대한 잠재적 위협을 AWS 서비스 탐지합니다. 는 특정 규정 준수 프레임워크에서 요구하는 침입 탐지 요구 사항을 충족DSS하여 PCI 와 같은 다양한 규정 준수 요구 사항을 해결하는 데 도움이 될 GuardDuty 수 있습니다.
- AWS Audit Manager - 이를 AWS 서비스 통해 AWS 사용량을 지속적으로 감사하여 규정 및 업계 표준 준수 및 위험을 관리하는 방법을 간소화할 수 있습니다.

Amazon Timestream for InfluxDB의 복원력

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 기반으로 구축됩니다. AWS 리전은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹과 연결된 물리적으로 분리되고 격리된 여러 가용 영역을 제공합니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라 섹션을 참조하세요.

Amazon Timestream for InfluxDB는 정기적으로 내부 백업을 가져와 24시간 동안 보관하여 가용성과 내구성을 지원합니다. 스냅샷은 삭제 중에 캡처되며 복원을 지원하기 위해 30일 동안 보존됩니다. 이러한 에 액세스하거나 사용하려면 AWS 지원 에서 티켓을 제출하세요.

다중 AZ 복구 기능을 사용하여 인스턴스를 생성할 수 있습니다. 자세한 내용은 다중 AZ DB 인스턴스 배포를 참조하세요.

Amazon Timestream for InfluxDB의 인프라 보안

관리형 서비스인 Amazon Timestream for InfluxDB는 Amazon Web Services: Overview of Security Processes 백서에 설명된 AWS 글로벌 네트워크 보안 절차에 의해 보호됩니다.

AWS 게시된 제어 영역 API 호출을 사용하여 네트워크를 통해 InfluxDB용 Timestream에 액세스합니다. 자세한 내용은 제어 영역 및 데이터 영역 섹션을 참조하세요. 클라이언트는 전송 계층 보안(TLS)

1.2 이상을 지원해야 합니다. TLS 1.2 또는 1.3을 권장합니다. 또한 클라이언트는 Ephemeral Diffie-Hellman(PFS) 또는 Elliptic Curve Ephemeral Diffie-Hellman()과 같은 완벽한 순방향 보안(DHE)을 갖춘 암호 제품군을 지원해야 합니다ECDHE. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 액세스 키 ID와 IAM 보안 주체와 연결된 보안 액세스 키를 사용하여 요청에 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)를 사용하여 임시 보안 인증을 생성하여 요청에 서명할 수 있습니다.

InfluxDB용 Timestream은 트래픽이 InfluxDB용 Timestream 인스턴스가 있는 특정 AWS 리전으로 격리되도록 설계되었습니다.

보안 그룹

보안 그룹은 DB 인스턴스에서 송수신되는 트래픽에 대한 액세스를 제어합니다. 기본적으로 DB 인스턴스에 대한 네트워크 액세스는 해제되어 있습니다. IP 주소 범위, 포트 또는 보안 그룹에서 액세스를 허용하는 보안 그룹의 규칙을 지정할 수 있습니다. 수신 규칙이 설정되면 동일한 규칙이 해당 보안 그룹과 연결된 모든 DB 인스턴스에 적용됩니다.

자세한 내용은 에서 DB 인스턴스에 대한 액세스 제어 VPC 단원을 참조하십시오.

InfluxDB용 Timestream의 구성 및 취약성 분석

구성 및 IT 제어는 고객과 AWS 고객 간의 공동 책임입니다. 자세한 내용은 AWS 공동 책임 모델 을 참조하세요. 공동 책임 모델 외에도 Timestream for InfluxDB 사용자는 다음 사항에 유의해야 합니다.

- 관련 클라이언트 측 종속성을 사용하여 클라이언트 애플리케이션을 패치하는 것은 고객의 책임입니다.
- 고객은 적절한 경우 침투 테스트를 고려해야 합니다(https://aws.amazon.com/security/침투 테스트/ 참조).

InfluxDB Timestream의 인시던트 응답

Amazon Timestream for InfluxDB 서비스 인시던트는 Personal Health Dashboard 에 보고됩니다. 대시보드 및 에 대해 자세히 알아볼 수 AWS Health 있습니다.

InfluxDB의 Timestream은 를 사용한 보고를 지원합니다 AWS CloudTrail. 자세한 내용은 를 사용하여 InfluxDB API 호출에 대한 Timestream 로깅 AWS CloudTrail 단원을 참조하십시오.

Amazon Timestream for InfluxDB API 및 인터페이스 VPC 엔드포인트(AWS PrivateLink)

인터페이스 API 엔드포인트를 생성하여 VPC와 Amazon Timestream for InfluxDB 제어 영역 엔드포인트 간에 프라이빗 연결을 설정할 수 있습니다. VPC 인터페이스 엔드포인트는 로 구동됩니다AWS PrivateLink. AWS PrivateLink 를 사용하면 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 AWS Direct Connect 연결 없이 Amazon Timestream for InfluxDB API 작업에 비공개로 액세스할 수 있습니다.

의 인스턴스는 Amazon Timestream for InfluxDB API 엔드포인트와 통신하는 데 퍼블릭 IP 주소가 필요하지 VPC 않습니다. 또한 인스턴스에서 사용 가능한 InfluxDB용 Timestream API 작업을 사용하는 데 퍼블릭 IP 주소가 필요하지 않습니다. VPC 와 Amazon Timestream for InfluxDB 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다. 각 인터페이스 엔드포인트는 서브넷에서 하나 이상의 탄력적 네트워크 인터페이스로 표현됩니다. 탄력적 네트워크 인터페이스에 대한 자세한 내용은 Amazon EC2 사용 설명서의 탄력적 네트워크 인터페이스를 참조하세요.

- VPC 엔드포인트에 대한 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요.
- InfluxDB API 작업의 Timestream에 대한 자세한 내용은 InfluxDB API 작업의 Timestream을 참조하세요.

인터페이스 VPC 엔드포인트를 생성한 후 엔드포인트에 대한 프라이빗 DNS 호스트 이름을 활성화하면 InfluxDB 엔드포인트의 기본 Timestream(https://timestream-influxb.*Region*.amazonaws.com)가 VPC 엔드포인트로 확인됩니다. 프라이빗 DNS 호스트 이름을 활성화하지 않으면 Amazon은 다음 형식으로 사용할 수 있는 DNS 엔드포인트 이름을 VPC 제공합니다.

```
VPC_Endpoint_ID.timestream-influxb.Region.vpce.amazonaws.com
```

자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 VPC 엔드포인트(AWS PrivateLink)를 참조하세요. InfluxDB용 Timestream은 내 모든 API 작업에 대한 호출을 지원합니다VPC.

> **Note**
>
> 프라이빗 DNS 호스트 이름은 의 VPC 엔드포인트 하나에 대해서만 활성화할 수 있습니다 VPC. 추가 VPC 엔드포인트를 생성하려면 프라이빗 DNS 호스트 이름을 비활성화해야 합니다.

VPC 엔드포인트 고려 사항

Amazon Timestream for InfluxDB VPC 엔드포인트에 대한 인터페이스 API 엔드포인트를 설정하기 전에 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 속성 및 제한 사항을 검토해야 합니다. Amazon Timestream for InfluxDB 리소스 관리와 관련된 모든 Timestream for InfluxDB API 작업은 를 VPC 사용하여 에서 사용할 수 있습니다 AWS PrivateLink. VPC 엔드포인트 정책은 InfluxDB API 엔드포인트용 Timestream에 지원됩니다. 기본적으로 엔드포인트를 통해 Timestream for InfluxDB API 작업에 대한 전체 액세스가 허용됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 사용하여 서비스에 대한 액세스 제어를 참조하세요.

InfluxDB용 Timestream에 대한 인터페이스 VPC 엔드포인트 생성 API

Amazon VPC 콘솔 또는 를 API 사용하여 Amazon Timestream for InfluxDB에 대한 VPC 엔드포인트를 생성할 수 있습니다 AWS CLI. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트 생성을 참조하세요.

인터페이스 VPC 엔드포인트를 생성한 후 엔드포인트의 프라이빗 DNS 호스트 이름을 활성화할 수 있습니다. 이 경우 기본 Amazon Timestream for InfluxDB 엔드포인트(https://timestream-influxb.*Region*.amazonaws.com)가 VPC 엔드포인트로 확인됩니다. 자세한 내용은 Amazon VPC 사용 설명서의 인터페이스 엔드포인트를 통한 서비스 액세스를 참조하세요.

Amazon Timestream for InfluxDB에 대한 VPC 엔드포인트 정책 생성 API

Timestream for InfluxDB 에 대한 액세스를 제어하는 엔드포인트 정책을 VPC 엔드포인트에 연결할 수 있습니다API. 이 정책은 다음을 지정합니다.

- 작업을 수행할 수 있는 보안 주체.
- 수행할 수 있는 작업.
- 작업을 수행할 수 있는 리소스.

자세한 내용은 Amazon VPC 사용 설명서의 VPC 엔드포인트를 사용하여 서비스에 대한 액세스 제어를 참조하세요.

Example VPC InfluxDB API 작업의 Timestream에 대한 엔드포인트 정책

다음은 InfluxDB용 Timestream 에 대한 엔드포인트 정책의 예입니다API. 엔드포인트에 연결되면 이 정책은 모든 리소스의 모든 보안 주체에 대해 나열된 InfluxDB용 Timestream API 작업에 대한 액세스 권한을 부여합니다.

```
{
 "Statement": [{
  "Principal": "*",
  "Effect": "Allow",
  "Action": [
   "timestream-influxb:CreateDbInstance",
   "timestream-influxb:UpdateDbInstance"
  ],
  "Resource": "*"
 }]
}
```

Example VPC 지정된 AWS 계정의 모든 액세스를 거부하는 엔드포인트 정책

다음 VPC 엔드포인트 정책이 AWS 계정을 거부합니다.*123456789012* 엔드포인트를 사용하여 리소스에 대한 모든 액세스. 이 정책은 다른 계정의 모든 작업을 허용합니다.

```
{
 "Statement": [{
   "Action": "*",
   "Effect": "Allow",
   "Resource": "*",
   "Principal": "*"
  },
  {
   "Action": "*",
   "Effect": "Deny",
   "Resource": "*",
   "Principal": {
    "AWS": [
     "123456789012"
    ]
   }
  }
 ]
}
```

Timestream for InfluxDB의 보안 모범 사례

Amazon Timestream for InfluxDB는 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.

최소 권한 액세스 구현

권한을 부여할 때 InfluxDB 리소스에 대한 Timestream 권한을 얻는 사람은 누구인지 결정합니다. 해당 리소스에서 허용할 작업을 사용 설정합니다. 따라서 작업을 수행하는 데 필요한 권한만 부여해야 합니다. 최소 권한 액세스를 구현하는 것이 오류 또는 악의적인 의도로 인해 발생할 수 있는 보안 위험과 영향을 최소화할 수 있는 근본적인 방법입니다.

IAM 역할 사용

생산자 및 클라이언트 애플리케이션에는 InfluxDB DB 인스턴스용 Timestream에 액세스하려면 유효한 보안 인증 정보가 있어야 합니다. 보안 AWS 인증 정보를 클라이언트 애플리케이션 또는 Amazon S3 버킷에 직접 저장해서는 안 됩니다. 이러한 보안 인증은 자동으로 교체되지 않으며 손상된 경우 비즈니스에 큰 영향을 줄 수 있는 장기 보안 인증입니다.

대신 IAM 역할을 사용하여 생산자와 클라이언트 애플리케이션이 InfluxDB DB 인스턴스용 Timestream에 액세스할 수 있는 임시 보안 인증을 관리해야 합니다. 역할을 사용하면 장기 자격 증명(예: 사용자 이름과 암호 또는 액세스 키)을 사용하여 다른 리소스에 액세스할 필요가 없습니다.

자세한 내용은 IAM 사용 설명서의 다음 주제를 참조하세요.

- IAM 역할
- 역할에 대한 일반적인 시나리오: 사용자, 애플리케이션 및 서비스

AWS Identity and Access Management (IAM) 계정을 사용하여 Amazon Timestream for InfluxDB API 작업, 특히 Amazon Timestream for InfluxDB 리소스를 생성, 수정 또는 삭제하는 작업에 대한 액세스를 제어합니다. 이러한 리소스에는 DB 인스턴스, 보안 그룹 및 파라미터 그룹이 포함됩니다.

- 사용자를 포함하여 Amazon Timestream for InfluxDB 리소스를 관리하는 각 사용자에 대해 개별 사용자를 생성합니다. AWS 루트 자격 증명을 사용하여 Amazon Timestream for InfluxDB 리소스를 관리하지 마세요.
- 각 사용자에게 각자의 임무를 수행하는 데 필요한 최소 권한 집합을 부여합니다.
- IAM 그룹을 사용하여 여러 사용자의 권한을 효과적으로 관리합니다.
- IAM 자격 증명을 정기적으로 교체합니다.
- AWS Amazon Timestream for InfluxDB 의 보안 암호를 자동으로 교체하도록 Secrets Manager를 구성합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 Secrets Manager 암호 교체를 참조하세요. AWS AWS Secrets Manager에서 프로그래밍 방식으로 보안 인증을 검색할 수도 있습니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 보안 암호 값 검색을 참조하세요.

- 를 사용하여 Timestream for InfluxDB 인플럭스 API 토큰을 보호합니다API 토큰.

종속 리소스에서 서버 측 암호화 구현

저장 데이터 및 전송 중인 데이터는 InfluxDB용 Timestream 에서 암호화할 수 있습니다. 자세한 내용은 전송 중 암호화 단원을 참조하십시오.

CloudTrail 를 사용하여 API 통화 모니터링

InfluxDB용 Timestream은 InfluxDB용 Timestream 에서 사용자, 역할 또는 서비스가 수행한 작업의 레코드를 AWS CloudTrail제공하는 AWS 서비스인 와 통합됩니다. InfluxDB

에서 수집한 정보를 사용하여 Timestream for InfluxDB 수행된 요청 CloudTrail, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

자세한 내용은 the section called “를 사용한 호출에 대한 LiveAnalytics API Timestream 로깅 AWS CloudTrail” 단원을 참조하십시오.

Amazon Timestream for InfluxDB는 제어 영역 CloudTrail 이벤트를 지원하지만 데이터 영역은 지원하지 않습니다. 자세한 내용은 제어 영역 및 데이터 영역 섹션을 참조하세요.

퍼블릭 액세스 가능성

Amazon VPC 서비스를 기반으로 가상 프라이빗 클라우드(VPC) 내에서 DB 인스턴스를 시작하면 해당 DB 인스턴스에 대한 퍼블릭 액세스 가능성을 켜거나 끌 수 있습니다. 생성한 DB 인스턴스의 DNS 이름이 퍼블릭 IP 주소로 확인되는지 여부를 지정하려면 퍼블릭 접근성 파라미터를 사용합니다. 이 파라미터를 사용하면 DB 인스턴스에 대한 퍼블릭 액세스가 있는지 여부를 지정할 수 있습니다.

DB 인스턴스가 에 VPC 있지만 공개적으로 액세스할 수 없는 경우 연결 또는 AWS Direct Connect 연결을 사용하여 AWS Site-to-Site VPN 프라이빗 네트워크에서 액세스할 수도 있습니다.

DB 인스턴스에 공개적으로 액세스할 수 있는 경우 서비스 관련 위협의 거부를 방지하거나 완화하는 데 도움이 되는 조치를 취해야 합니다. 자세한 내용은 서비스 거부 공격 소개 및 네트워크 보호를 참조하세요.

API 참조

Amazon Timestream for InfluxDB 의 전체 목록 및 세부 정보는 Amazon Timestream for InfluxDB 를 APIs APIs참조하세요.

모든 AWS 서비스에 공통적인 오류 코드는 AWS 지원 단원을 참조하세요.

문서 이력

변경 사항	설명	날짜
설명서 전용 업데이트	기본 할당량과 시스템 제한을 분리하도록 할당량 주제를 업데이트했습니다.	2024년 10월 22일
Amazon Timestream은 이제 쿼리 인사이트를 지원합니다.	이제 Timestream에는 쿼리를 최적화하고, 성능을 개선하고, 비용을 절감하는 데 도움이 되는 쿼리 인사이트 기능에 대한 지원이 포함되어 있습니다.	2024년 10월 22일
기존 정책에 대한 Amazon Timestream for InfluxDB 업데이트.	Amazon Timestream for InfluxDB가 라우팅 테이블을 설명하기 위해 기존 `AmazonTimestreamInfluxDBFullAccess` 관리형 정책에 `ec2:DescribeRouteTables` 작업을 추가했습니다.	2024년 10월 8일
`AmazonTimestreamReadOnlyAccess` - 기존 정책 업데이트	에 대한 Timestream이 AWS 계정 설정을 설명하기 위한 `AmazonTimestreamReadOnlyAccess` 관리형 정책에 `DescribeAccountSettings` 권한을 추가 LiveAnalytics 했습니다.	2024년 6월 3일
LiveAnalytics 이제 에 대한 Amazon Timestream에서 Timestream Compute Units(TCUs)를 지원합니다.	LiveAnalytics 이제 Amazon Timestream for 에는 쿼리 요구 사항에 할당된 컴퓨팅 용량을 측정하기 위한 Timestream	2024년 4월 29일

	Compute Units(TCUs)에 대한 지원이 포함되어 있습니다.	
새 정책 추가	Amazon Timestream for InfluxDB는 두 가지 새로운 정책을 추가했습니다. 하나는 서비스가 계정의 네트워크 인터페이스와 보안 그룹을 관리할 수 있도록 허용하는 정책입니다. 자세한 내용은 섹션을 참조하세요AmazonTimestreamInfluxDBServiceRolePolicy. Amazon Timestream InfluxDB 인스턴스를 생성, 업데이트, 삭제 및 나열하고 파라미터 그룹을 생성 및 나열할 수 있는 전체 관리 액세스 권한을 제공하는 또 다른 기능입니다. 자세한 내용은 섹션을 참조하세요AmazonTimestreamInfluxDBFullAccess.	2024년 3월 14일
이제 Amazon Timestream for InfluxDB를 일반적으로 사용할 수 있습니다.	이 설명서에서는 Amazon Timestream for InfluxDB 의 초기 릴리스를 다룹니다.	2024년 3월 14일
Amazon Timestream for LiveAnalytics Query 이벤트는 에서 사용할 수 있습니다. AWS CloudTrail	Amazon Timestream은 LiveAnalytics 이제 에 API 데이터 쿼리 이벤트를 게시합니다 AWS CloudTrail. 고객은 자신의 AWS 계정에서 수행된 모든 쿼리 API 요청을 감사하고 어떤 IAM 사용자/역할이 요청을 수행했는지, 요청이 수행된 시기, 쿼리된 데이터베이스 및 테이블, 요청의 쿼리 ID와 같은 정보를 볼 수 있습니다.	2023년 9월 12일

에 대한 Amazon Timestream LiveAnalytics UNLOAD	Amazon Timestream은 LiveAnalytics 이제 UNLOAD에서 쿼리 결과를 S3로 내보낼 수 있도록 지원합니다.	2023년 5월 12일
기존 정책 LiveAnalytics 업데이트를 위한 Amazon Timestream.	관리형 정책에 배치 로드 권한이 추가되었습니다.	2023년 2월 24일
LiveAnalytics 배치 로드에 대한 Amazon Timestream.	Amazon Timestream은 LiveAnalytics 이제 배치 로드 기능을 지원합니다.	2023년 2월 24일
LiveAnalytics 이제 용 Amazon Timestream에서 를 지원합니다 AWS Backup.	LiveAnalytics 이제 용 Amazon Timestream에서 를 지원합니다 AWS Backup.	2022년 12월 14일
AWS 관리형 정책에 대한 LiveAnalytics 업데이트를 위한 Amazon Timestream	기존 AWS 관리형 정책에 대한 업데이트를 LiveAnalytics포함하여 관리형 정책 및 Amazon Timestream for 에 대한 새로운 정보입니다.	2021년 11월 29일
용 Amazon Timestream은 예약된 쿼리 LiveAnalytics 를 지원합니다.	LiveAnalytics 이제 용 Amazon Timestream은 일정에 따라 사용자를 대신하여 쿼리 실행을 지원합니다.	2021년 11월 29일
용 Amazon Timestream은 마그네틱 스토어를 LiveAnalytics 지원합니다.	LiveAnalytics 이제 용 Amazon Timestream은 테이블 쓰기에 마그네틱 스토리지 사용을 지원합니다.	2021년 11월 29일
LiveAnalytics 다중 측정 레코드에 대한 Amazon Timestream.	LiveAnalytics 이제 용 Amazon Timestream은 시계열 데이터를 저장하기 위한 더 컴팩트한 형식을 지원합니다.	2021년 11월 29일

AWS 관리형 정책에 대한 LiveAnalytics 업데이트를 위한 Amazon Timestream	기존 AWS 관리형 정책에 대한 업데이트를 LiveAnalytics포함하여 관리형 정책 및 Amazon Timestream for 에 대한 새로운 정보입니다.	2021년 5월 24일
LiveAnalytics 이제 유럽(프랑크푸르트) 리전에서 에 대한 Amazon Timestream을 사용할 수 있습니다.	이제 용 Amazon Timestream LiveAnalytics 을 유럽(프랑크푸르트) 리전()에서 일반적으로 사용할 수 있습니다`eu-central-1` .	2021년 4월 23일
에 대한 Amazon Timestream LiveAnalytics 은 이제 VPC 엔드포인트()를 지원합니다AWS PrivateLink.	LiveAnalytics 이제 용 Amazon Timestream은 VPC 엔드포인트() 사용을 지원합니다AWS PrivateLink.	2021년 3월 23일
Amazon Timestream은 이제 크로스 테이블 쿼리를 지원합니다.	에 Amazon Timestream을 사용하여 테이블 간 쿼리 LiveAnalytics 를 실행할 수 있습니다.	2021년 2월 10일
Amazon Timestream은 LiveAnalytics 이제 향상된 쿼리 실행 통계를 지원합니다.	LiveAnalytics 이제 용 Amazon Timestream은 스캔된 데이터의 양과 같은 향상된 쿼리 실행 통계를 지원합니다.	2021년 2월 10일
Amazon Timestream은 LiveAnalytics 이제 고급 시계열 함수를 지원합니다.	용 Amazon Timestream LiveAnalytics 을 사용하여 파생, 적분 및 상관 관계와 같은 고급 시계열 함수를 사용하여 SQL 쿼리를 실행할 수 있습니다.	2021년 2월 10일

에 대한 Amazon Timestream LiveAnalytics 은 이제 HIPAA, ISO및 PCI 규정을 준수합니다.	이제 HIPAA, ISO및 PCI규정 준수 인프라가 필요한 워크로드에 Amazon Timestream LiveAnalytics 을 사용할 수 있습니다.	2021년 1월 27일
Amazon Timestream은 LiveAnalytics 이제 오픈 소스 텔레그래프 및 grafana를 지원합니다.	이제 Amazon Timestream for 에서 오픈 소스 플러그인 기반 서버 에이전트인 Telegraf를 사용하여 지표를 수집하고 보고할 수 있으며, 데이터베이스의 오픈 소스 분석 및 모니터링 플랫폼인 Grafana를 사용할 수 있습니다 LiveAnalytics.	2020년 11월 25일
이제 용 Amazon Timestream LiveAnalytics 을 일반적으로 사용할 수 있습니다.	이 설명서에서는 용 Amazon Timestream의 초기 릴리스를 다룹니다 LiveAnalytics.	2020년 9월 30일

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.