aws

Managed Service for Apache Flink 개발자 안내서

Managed Service for Apache Flink

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Managed Service for Apache Flink: Managed Service for Apache Flink 개발자 안내서

Copyright © 2024 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께 사용하여 고객에게 혼란을 초래하거나 Amazon을 폄하 또는 브랜드 이미지에 악영향을 끼치는 목적으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon 계열사, 관련 업체 또는 Amazon의 지원 업체 여부에 상관없이 해당 소유자의 자산입니다.

Table of Contents

Amazon Managed Service for Apache Flink는 이전에 Amazon Kinesis Data Analytics for Apache Flink로 알려졌습니다.

기계 번역으로 제공되는 번역입니다. 제공된 번역과 원본 영어의 내용이 상충하는 경우에는 영어 버전이 우선합니다.

Amazon Managed Service for Apache Flink란 무엇인가?

Apache Flink용 Amazon 관리형 서비스를 사용하면 자바, Scala, Python 또는 SQL을 사용하여 스트리밍 데이터를 처리하고 분석할 수 있습니다. 이 서비스를 사용하면 스트리밍 소스 및 정적 소스를 대상으로 코드를 작성하고 실행하여 시계열 분석을 수행하고 실시간 대시보드 및 지표를 제공할 수 있습니다.

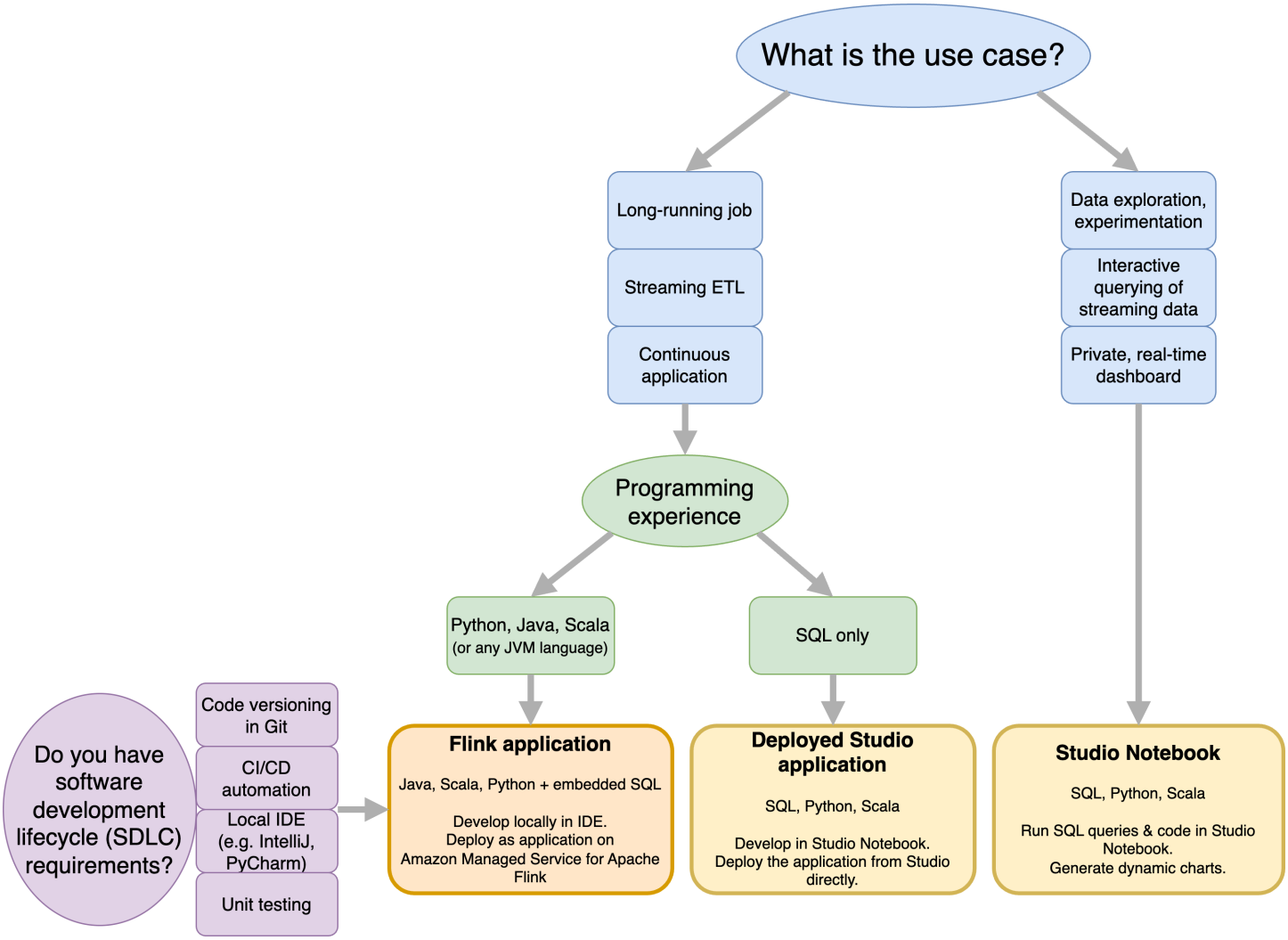
Apache Flink 기반 오픈 소스 라이브러리를 사용하여 Apache Flink용 관리 서비스에서 원하는 언어로 애플리케이션을 빌드할 수 있습니다. Apache Flink는 데이터 스트림을 처리하기 위한 인기 있는 프레임워크 및 엔진입니다.

Managed Service for Apache Flink는 Apache Flink 애플리케이션을 위한 기본 인프라를 제공합니다. 컴퓨팅 리소스 프로비저닝, AZ 페일오버 복원력, 병렬 계산, 자동 확장, 애플리케이션 백업 (체크포인트 및 스냅샷으로 구현) 과 같은 핵심 기능을 처리합니다. Flink 인프라를 직접 호스팅할 때 사용하는 것과 동일한 방식으로 고급 Flink 프로그래밍 기능(예: 연산자, 함수, 소스, 싱크)을 사용할 수 있습니다.

Apache Flink용 관리형 서비스 또는 Apache Flink Studio용 관리형 서비스 선택

Apache Flink용 아마존 매니지드 서비스를 사용하여 Flink 작업을 실행할 수 있는 두 가지 옵션이 있습니다. Apache Flink용 관리형 서비스를 사용하면 선택한 IDE와 Apache Flink 데이터스트림 또는 테이블 API를 사용하여 Java, Scala 또는 Python (및 임베디드 SQL) 으로 Flink 애플리케이션을 빌드할 수 있습니다. Apache Flink Studio용 관리형 서비스를 사용하면 실시간으로 대화형 방식으로 데이터 스트림을 쿼리하고 표준 SQL, Python 및 Scala를 사용하여 스트림 처리 애플리케이션을 쉽게 빌드하고 실행할 수 있습니다.

사용 사례에 가장 적합한 방법을 선택할 수 있습니다. 확실하지 않은 경우 이 섹션에서 자세한 지침을 확인할 수 있습니다.



Apache Flink용 Amazon Managed Service를 사용할지 아니면 Apache Flink Studio용 Amazon Managed Service를 사용할지 결정하기 전에 사용 사례를 고려해야 합니다.

스트리밍 ETL 또는 지속적 애플리케이션과 같은 워크로드를 처리하는 장기 실행 애플리케이션을 운영하려는 경우 Apache Flink용 관리형 서비스를 사용하는 것을 고려해야 합니다. 선택한 IDE에서 직접 Flink API를 사용하여 Flink 애플리케이션을 만들 수 있기 때문입니다. 또한 IDE를 사용하여 로컬에서 개발하면 Git의 코드 버전 관리, CI/CD 자동화 또는 유닛 테스트와 같은 SDLC (소프트웨어 개발 라이프사이클) 공통 프로세스 및 도구를 활용할 수 있습니다.

임시 데이터 탐색에 관심이 있거나, 스트리밍 데이터를 대화형 방식으로 쿼리하거나, 비공개 실시간 대시보드를 생성하려는 경우, Apache Flink Studio의 Managed Service for Apache Flink Studio를 사용하면 클릭 몇 번으로 이러한 목표를 달성할 수 있습니다. SQL에 익숙한 사용자는 장기간 실행되는 애플리케이션을 Studio에서 직접 배포하는 방안을 고려해 볼 수 있습니다.

> **Note**
>
> Studio 노트북을 장기 실행 애플리케이션으로 승격할 수 있습니다. 그러나 Git의 코드 버전 관리 및 CI/CD 자동화와 같은 SDLC 도구 또는 단위 테스트와 같은 기술과 통합하려는 경우 선택한 IDE를 사용하는 Apache Flink용 관리형 서비스를 사용하는 것이 좋습니다.

Apache Flink용 관리 서비스에서 사용할 Apache Flink API 선택

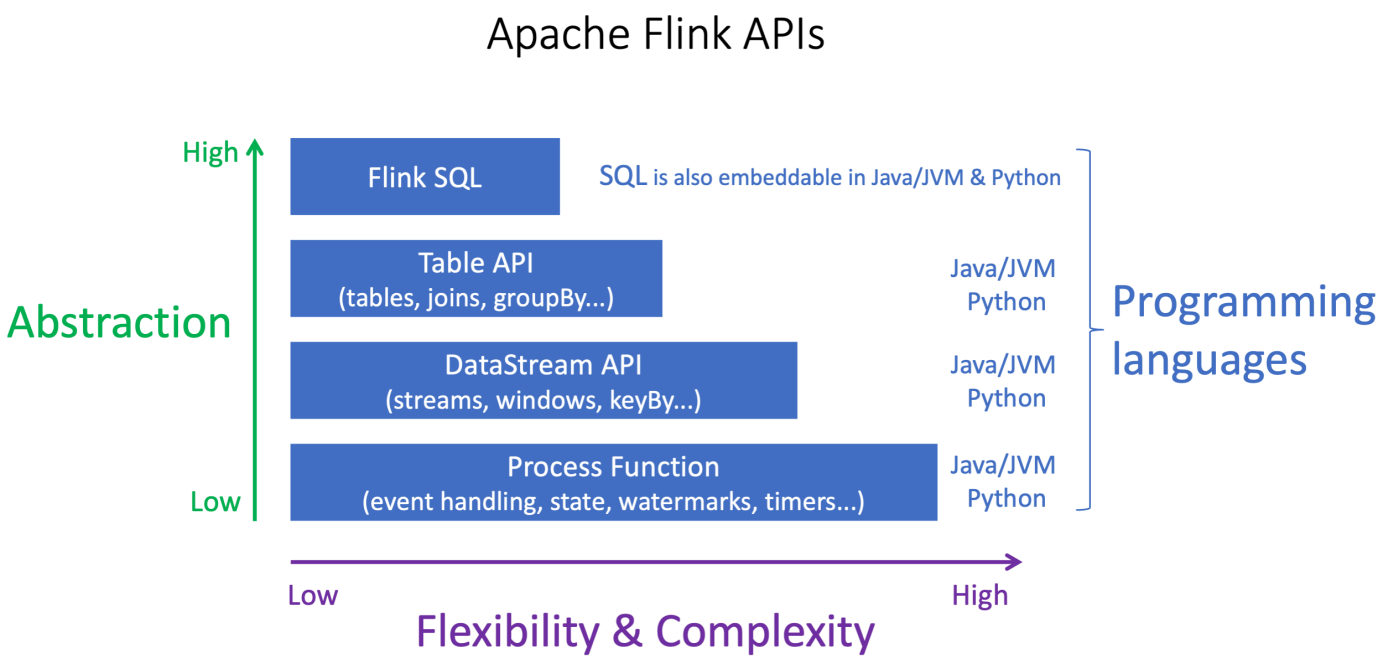
선택한 IDE의 Apache Flink API를 사용하여 Apache Flink용 관리 서비스에서 Java, Python 및 Scala를 사용하여 애플리케이션을 빌드할 수 있습니다. Flink 데이터스트림 및 테이블 API를 사용하여 애플리케이션을 빌드하는 방법에 대한 지침은 설명서에서 찾을 수 있습니다. Flink 애플리케이션을 만드는 데 사용할 언어와 애플리케이션 및 운영의 요구 사항을 가장 잘 충족하는 데 사용할 API를 선택할 수 있습니다. 확실하지 않은 경우 이 섹션에서 자세한 지침을 확인할 수 있습니다.

플링크 API 선택하기

Apache Flink API는 추상화 수준이 다르므로 애플리케이션 구축 결정에 영향을 미칠 수 있습니다. 표현력이 뛰어나고 유연하며 함께 사용하여 애플리케이션을 빌드할 수 있습니다. Flink API를 하나만 사용할 필요는 없습니다. 플링크 API에 대한 자세한 내용은 아파치 플링크 설명서에서 확인할 수 있습니다.

플링크는 플링크 SQL, 테이블 API, API, API와 함께 사용되는 프로세스 함수 등 네 가지 수준의 DataStream API 추상화를 제공합니다. DataStream 이는 모두 Apache Flink용 아마존 매니지드 서비스에서 지원됩니다. 가능하면 더 높은 수준의 추상화로 시작하는 것이 좋지만 일부 Flink 기능은 Java, Python 또는 Scala로 애플리케이션을 만들 수 있는 Datastream API에서만 사용할 수 있습니다. 다음과 같은 경우 데이터스트림 API 사용을 고려해야 합니다.

- 상태를 세밀하게 제어해야 합니다.
- 외부 데이터베이스 또는 엔드포인트를 비동기적으로 호출하는 기능을 활용하려는 경우 (예: 추론)
- 사용자 지정 타이머를 사용하고 싶습니다.



> **Note**
>
> 데이터스트림 API를 사용한 언어 선택:
>
> - 선택한 프로그래밍 언어에 관계없이 모든 Flink 애플리케이션에 SQL을 내장할 수 있습니다.
> - DataStream API를 사용할 계획이라면 Python에서 모든 커넥터가 지원되는 것은 아닙니다.
> - 낮은 지연 시간/높은 처리량이 필요한 경우 API와 상관없이 Java/Scala를 고려해야 합니다.
> - 프로세스 함수 API에서 비동기 IO를 사용하려면 Java를 사용해야 합니다.

시작하기

스트리밍 데이터를 지속적으로 읽고 처리하는 Managed Service for Apache Flink 애플리케이션을 생성하는 것부터 시작할 수 있습니다. 그런 다음 선택한 IDE를 사용하여 코드를 작성하고 라이브 스트리밍 데이터로 테스트하십시오. Managed Service for Apache Flink가 결과를 전송하려는 대상을 구성할 수도 있습니다.

시작하려면 다음 섹션을 읽어보는 것이 좋습니다:

- Managed Service for Apache Flink: 작동 방식
- 아파치 플링크용 아마존 매니지드 서비스 (DataStream API) 시작하기

또는 실시간으로 대화식으로 데이터 스트림을 쿼리하고 표준 SQL, Python 및 Scala를 사용하여 스트림 처리 애플리케이션을 쉽게 빌드하고 실행할 수 있는 Apache Flink Studio용 관리형 서비스 노트북을 만드는 것부터 시작할 수 있습니다. AWS Management Console에서 클릭 몇 번으로 서버리스 노트북을 실행하여 데이터 스트림을 쿼리하고 몇 초 만에 결과를 얻을 수 있습니다. 시작하려면 다음 섹션을 읽어보는 것이 좋습니다:

- Managed Service for Apache Flink와 함께 Studio 노트북 사용
- Studio 노트북 생성

Managed Service for Apache Flink: 작동 방식

Managed Service for Apache Flink Flink는 Apache Flink 애플리케이션을 사용하여 스트리밍 데이터를 처리할 수 있는 완전관리형 Amazon 서비스입니다.

Apache Flink 애플리케이션 프로그래밍

Apache Flink 애플리케이션은 Apache Flink 프레임워크를 사용하여 만든 Java 또는 Scala 애플리케이션입니다. 로컬에서 Apache Flink 애플리케이션을 작성하고 빌드합니다.

애플리케이션은 주로 DataStream API 또는 Table API를 사용합니다. 다른 Apache Flink API도 사용할 수 있지만 스트리밍 애플리케이션을 빌드하는 데는 많이 사용되지 않습니다.

두 API의 특징은 다음과 같습니다.

DataStream API

Apache Flink DataStream API 프로그래밍 모델은 두 가지 구성 요소를 기반으로 합니다.

- 데이터 스트림: 연속적인 데이터 레코드 흐름을 구조적으로 표현한 것입니다.
- 변환 연산자: 하나 이상의 데이터 스트림을 입력으로 받아 하나 이상의 데이터 스트림을 출력으로 생성합니다.

DataStream API로 만든 애플리케이션은 다음을 수행합니다.

- 데이터 소스(예: Kinesis 스트림 또는 Amazon MSK 항목)에서 데이터를 읽습니다.
- 필터링, 집계 또는 보강과 같은 변환을 데이터에 적용합니다.
- 변환된 데이터를 Data Sink에 씁니다.

DataStream API를 사용하는 애플리케이션은 Java 또는 Scala로 작성할 수 있으며 Kinesis 데이터 스트림, Amazon MSK 항목 또는 사용자 지정 소스에서 읽을 수 있습니다.

애플리케이션은 커넥터를 사용하여 데이터를 처리합니다. Apache Flink는 다음 유형의 커넥터를 사용합니다.

- 소스: 외부 데이터를 읽는 데 사용되는 커넥터입니다.
- 싱크: 외부 위치에 쓰는 데 사용되는 커넥터입니다.

- 오퍼레이터: 애플리케이션 내에서 데이터를 처리하는 데 사용되는 커넥터입니다.

일반적인 애플리케이션은 소스가 있는 하나 이상의 데이터 스트림, 하나 이상의 연산자가 있는 데이터 스트림, 하나 이상의 데이터 싱크로 구성됩니다.

DataStream API 사용에 대한 자세한 내용은 DataStream API을(를) 참조하세요.

테이블 API

Apache Flink Table API 프로그래밍 모델은 다음 구성 요소를 기반으로 합니다.

- 테이블 환경: 하나 이상의 테이블을 만들고 호스팅하는 데 사용하는 기본 데이터에 대한 인터페이스입니다.
- 테이블: SQL 테이블 또는 뷰에 대한 액세스를 제공하는 객체입니다.
- 테이블 소스: Amazon MSK 항목과 같은 외부 소스에서 데이터를 읽는 데 사용됩니다.
- 테이블 함수: 데이터를 변환하는 데 사용되는 SQL 쿼리 또는 API 직접 호출입니다.
- 테이블 싱크: Amazon S3 버킷과 같은 외부 위치에 데이터를 쓰는 데 사용됩니다.

Table API로 생성한 애플리케이션은 다음을 수행합니다.

- `Table Source`에 연결하여 `TableEnvironment`을(를) 생성합니다.
- SQL 쿼리 또는 Table API 함수를 사용하여 `TableEnvironment`에 테이블을 생성합니다.
- 테이블 API 또는 SQL을 사용하여 테이블에서 쿼리를 실행
- 테이블 함수 또는 SQL 쿼리를 사용하여 쿼리 결과에 변환을 적용합니다.
- 쿼리 또는 함수 결과를 `Table Sink`에 씁니다.

Table API를 사용하는 애플리케이션은 Java 또는 Scala로 작성할 수 있으며, API 직접 호출 또는 SQL 쿼리를 사용하여 데이터를 쿼리할 수 있습니다.

Table API 사용에 대한 자세한 내용은 표 API을(를) 참조하세요.

Managed Service for Apache Flink 애플리케이션 생성

Managed Service for Apache Flink는 Apache Flink 애플리케이션을 호스팅하기 위한 환경을 만들고 다음과 같은 설정을 제공하는 AWS 서비스입니다.

- 런타임 속성: 애플리케이션에 제공할 수 있는 파라미터. 애플리케이션 코드를 다시 컴파일하지 않고도 이러한 파라미터를 변경할 수 있습니다.
- 내결함성: 애플리케이션이 인터럽트 및 재시작으로부터 복구되는 방법
- 로깅 및 모니터링: 애플리케이션에서 CloudWatch Logs에 이벤트를 로깅하는 방법
- 스케일링: 애플리케이션이 컴퓨팅 리소스를 프로비저닝하는 방법.

콘솔이나 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 생성합니다. Managed Service for Apache Flink 애플리케이션 생성을 시작하려면 시작하기 (API) DataStream 을(를) 참조하세요.

Managed Service for Apache Flink 애플리케이션 생성

이 주제에는 Managed Service for Apache Flink에 대한 자세한 내용이 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 코드 구축
- Managed Service for Apache Flink 애플리케이션 생성
- Managed Service for Apache Flink 애플리케이션 시작하기
- Managed Service for Apache Flink 애플리케이션 확인

Managed Service for Apache Flink 애플리케이션 코드 구축

이 섹션에서는 Managed Service for Apache Flink 애플리케이션의 애플리케이션 코드를 구축하는 데 사용하는 구성 요소에 대해 설명합니다.

지원되는 최신 버전의 Apache Flink를 애플리케이션 코드에 사용하는 것이 좋습니다. Managed Service for Apache Flink가 지원하는 Apache Flink의 최신 버전은 1.15.2입니다. Managed Service for Apache Flink 애플리케이션 업그레이드에 대한 자세한 설명은 애플리케이션 업그레이드을 참조하십시오.

Apache Maven을 사용하여 애플리케이션 코드를 구축합니다. Apache Maven 프로젝트는 `pom.xml` 파일을 사용하여 해당 프로젝트에서 사용하는 구성 요소의 버전을 지정합니다.

> **Note**
>
> Managed Service for Apache Flink는 최대 512MB 크기의 JAR 파일을 지원합니다. 이보다 큰 JAR 파일을 사용하면 애플리케이션이 시작되지 않습니다.

Managed Service for Apache Flink 애플리케이션에 다음 구성 요소 버전을 사용하세요:

구성 요소	버전
Java	11 (권장)
Scala	아래 Scala 디커플링 노트를 참조하십시오.
아파치 플링크 런타임용 매니지드 서비스 () aws-kinesisanalytics-runtime	1.2.0
AWSKinesis 커넥터 () flink-connector-kinesis	1.15.2
Apache Beam (빔 애플리케이션만 해당)	2.33.0(Jackson 버전 2.12.2 포함)

버전 1.15부터 Flink에는 Scala가 없습니다. 이제 애플리케이션은 모든 Scala 버전에서 Java API를 사용할 수 있습니다. 선택한 Scala 표준 라이브러리를 Scala 애플리케이션에 번들로 제공해야 합니다.

Apache Flink 버전 1.15.2를 사용하는 Managed Service for Apache Flink 애플리케이션의 `pom.xml` 파일 예는 Managed Service for Apache Flink 시작하기 애플리케이션을 참조하십시오.

Apache Beam을 사용하는 Managed Service for Apache Flink 애플리케이션을 만드는 방법에 대한 자세한 설명은 Apache Beam 사용을 참조하십시오.

애플리케이션의 Apache Flink 버전 지정

Managed Service for Apache Flink 런타임 버전 1.1.0 이상을 사용하는 경우 애플리케이션을 컴파일할 때 애플리케이션에서 사용하는 Apache Flink 버전을 지정합니다. 다음과 같이 `-Dflink.version` 파라미터와 함께 Apache Flink 버전을 제공합니다.

```
mvn package -Dflink.version=1.15.3
```

이전 버전의 Apache Flink를 사용하여 애플리케이션을 구축하려면 이전 버전을 참조하십시오.

Managed Service for Apache Flink 애플리케이션 생성

애플리케이션 코드를 구축한 후에는 다음을 수행하여 Managed Service for Apache Flink 애플리케이션을 생성합니다.

- 애플리케이션 코드 업로드: Amazon S3 버킷에 애플리케이션 코드를 업로드합니다. 애플리케이션 코드의 S3 버킷 명칭과 객체 명칭은 애플리케이션을 생성할 때 지정합니다. 애플리케이션 코드를 업로드하는 방법을 보여주는 자습서는 시작하기 (API) DataStream 자습서의 the section called “Apache Flink 스트리밍 Java 코드 업로드”를 참조하십시오.
- Managed Service for Apache Flink 애플리케이션 만들기: 다음 방법 중 하나를 사용하여 Managed Service for Apache Flink 애플리케이션을 만드십시오:
  - AWS 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션 만들기: AWS 콘솔을 사용하여 애플리케이션을 만들고 구성할 수 있습니다.

    콘솔을 사용하여 애플리케이션을 생성하면 애플리케이션의 종속 리소스 (예: CloudWatch 로그 스트림, IAM 역할, IAM 정책) 가 자동으로 생성됩니다.

    콘솔을 사용하여 애플리케이션을 생성할 때는 Managed Service for Apache Flink - 애플리케이션 생성 페이지의 풀다운에서 애플리케이션을 선택하여 애플리케이션에서 사용하는 Apache Flink 버전을 지정합니다.

    콘솔을 사용하여 애플리케이션을 생성하는 방법에 대한 자습서는 시작하기 (API) DataStream 자습서의 the section called “애플리케이션 생성 및 실행(콘솔)” 섹션을 참조하세요.
  - AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션 생성: AWS CLI를 사용하여 애플리케이션을 생성하고 구성할 수 있습니다.

    CLI를 사용하여 애플리케이션을 생성할 때는 애플리케이션의 종속 리소스 (예: CloudWatch 로그 스트림, IAM 역할, IAM 정책) 도 수동으로 생성해야 합니다.

    CLI를 사용하여 애플리케이션을 만들 때는 `CreateApplication` 작업의 `RuntimeEnvironment` 파라미터를 사용하여 애플리케이션에서 사용하는 Apache Flink 버전을 지정합니다.

    CLI를 사용하여 애플리케이션을 생성하는 방법에 대한 자습서는 시작하기 (API) DataStream 자습서의 the section called “CLI를 사용하여 애플리케이션 생성 및 실행”을 참조하세요.

> **Note**
>
> 기존 애플리케이션의 `RuntimeEnvironment`은 변경할 수 없습니다. 기존 애플리케이션의 `RuntimeEnvironment`을 변경해야 하는 경우 애플리케이션을 삭제하고 다시 생성해야 합니다.

Managed Service for Apache Flink 애플리케이션 시작하기

애플리케이션 코드를 구축하고, S3에 업로드하고, Managed Service for Apache Flink 애플리케이션을 생성한 후 애플리케이션을 시작합니다. Managed Service for Apache Flink 애플리케이션을 시작하는 데 일반적으로 몇 분이 걸립니다.

애플리케이션을 시작하려면 다음 방법 중 하나를 사용합니다:

- AWS 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션 시작: AWS 콘솔의 애플리케이션 페이지에서 실행을 선택하여 애플리케이션을 실행할 수 있습니다.
- AWSAPI를 사용하여 Apache Flink용 관리형 서비스 애플리케이션 시작: 작업을 사용하여 애플리케이션을 실행할 수 있습니다. StartApplication

Managed Service for Apache Flink 애플리케이션 확인

다음과 같은 방법으로 애플리케이션이 작동하는지 확인할 수 있습니다.

- CloudWatch 로그 사용: 로그 및 CloudWatch CloudWatch 로그 인사이트를 사용하여 애플리케이션이 제대로 실행되고 있는지 확인할 수 있습니다. Apache Flink용 관리형 서비스 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오. 로깅 및 모니터링
- 지표 사용 CloudWatch : CloudWatch 지표를 사용하여 애플리케이션의 활동이나 애플리케이션이 입력 또는 출력에 사용하는 리소스 (예: Kinesis 스트림, Kinesis Data Firehose 스트림 또는 Amazon S3 버킷) 의 활동을 모니터링할 수 있습니다. CloudWatch 지표에 대한 자세한 내용은 Amazon CloudWatch 사용 설명서의 지표 사용을 참조하십시오.
- 출력 위치 모니터링: 애플리케이션이 특정 위치(예: Amazon S3 버킷 또는 데이터베이스)에 출력을 기록하는 경우 해당 위치에서 기록된 데이터를 모니터링할 수 있습니다.

Managed Service for Apache Flink 애플리케이션 실행

이 주제에는 Managed Service for Apache Flink 실행에 대한 자세한 내용이 포함되어 있습니다.

Managed Service for Apache Flink 애플리케이션을 실행하면 서비스가 Apache Flink 작업을 생성합니다. Apache Flink 작업은 Managed Service for Apache Flink 애플리케이션의 실행 라이프사이클입니다. 작업 실행 및 작업에 사용되는 리소스는 Job Manager에서 관리합니다. Job Manager는 애플리케이션 실행을 작업으로 구분합니다. 각 작업은 작업 관리자가 관리합니다. 애플리케이션 성능을 모니터링할 때 각 Task Manager 또는 Job Manager 전체의 성능을 검사할 수 있습니다.

Apache Flink 작업에 대한 자세한 설명은 Apache Flink 설명서의 작업 및 예약을 참조하십시오.

애플리케이션 및 작업 상태

애플리케이션과 애플리케이션 작업 모두 현재 실행 상태입니다.

- 애플리케이션 상태: 애플리케이션에는 실행 단계를 설명하는 현재 상태가 있습니다. 애플리케이션 상태에는 다음이 포함됩니다:
  - 꾸준한 애플리케이션 상태: 상태가 변경될 때까지 애플리케이션은 일반적으로 다음과 같은 상태를 유지합니다.
    - 준비: 새 애플리케이션이나 중지된 애플리케이션은 실행할 때까지 준비 상태입니다.
    - 가동: 성공적으로 시작된 애플리케이션은 가동 상태입니다.
  - 임시 애플리케이션 상태: 이러한 상태의 애플리케이션은 일반적으로 다른 상태로 전환되는 중입니다. 애플리케이션이 일정 시간 동안 임시 상태에 있는 경우 `Force` 파라미터가 `true`로 설정된 StopApplication 작업을 사용하여 애플리케이션을 중지할 수 있습니다. 이러한 상태에는 다음이 포함됩니다:
    - StartApplication 작업 후에 `STARTING:`이 발생합니다. 애플리케이션이 `READY` 상태에서 `RUNNING` 상태로 전환되고 있습니다.
    - StopApplication 작업 후에 `STOPPING:`이 발생합니다. 애플리케이션이 `RUNNING` 상태에서 `READY` 상태로 전환되고 있습니다.
    - DeleteApplication 작업 이후에 `DELETING:`이 발생합니다. 애플리케이션을 삭제하는 중입니다.
    - UpdateApplication 작업 후에 `UPDATING:`이 발생합니다. 애플리케이션이 업데이트 중이며 `RUNNING` 또는 `READY` 상태로 다시 전환됩니다.
    - `AUTOSCALING:` 애플리케이션은 ParallelismConfiguration의 `AutoScalingEnabled` 속성을 `true`으로 설정하고, 서비스는 애플리케이션의 병렬성을 높이고 있습니다. 애플리케이션이 이 상태인 경우 사용할 수 있는 유효한 API 작업은 `Force` 파라미터가 `true`로 설정된

StopApplication 작업뿐입니다. 자동 조정에 대한 자세한 설명은 자동 조정 섹션을 참조하십시오.

- FORCE_STOPPING:Force 파라미터가 true로 설정된 상태에서 StopApplication 작업을 호출한 후에 이 발생합니다. 애플리케이션을 강제 중지하는 중입니다. 애플리케이션이 STARTING, UPDATING, STOPPING 또는 AUTOSCALING 상태에서 READY 상태로 전환됩니다.
- RollbackApplication 작업이 호출된 후에 ROLLING_BACK:이 발생합니다. 애플리케이션이 이전 버전으로 회귀하는 중입니다. 애플리케이션이 UPDATING 또는 AUTOSCALING 상태에서 RUNNING 상태로 전환됩니다.
- ROLLED_BACK: 애플리케이션을 성공적으로 회귀하면 이는 회귀한 소스 버전의 상태가 됩니다. 애플리케이션 회귀에 대한 자세한 설명은 RollbackApplication을 참조하십시오.
- Managed Service for Apache Flink가 애플리케이션에 패치를 적용하는 동안 MAINTENANCE:이 발생합니다. 자세한 설명은 정비 섹션을 참조하세요.

콘솔을 사용하거나 DescribeApplication 작업을 사용하여 애플리케이션 상태를 확인할 수 있습니다.

- 작업 현황: 애플리케이션이 RUNNING 상태에 있으면 작업에는 현재 실행 단계를 설명하는 상태가 있습니다. 작업은 CREATED 상태에서 시작하고, 시작되면 RUNNING 상태로 진행됩니다. 오류 상황이 발생하는 경우 애플리케이션은 다음 상태가 됩니다.
  - Apache Flink 1.11 이상을 사용하는 애플리케이션의 경우 애플리케이션은 RESTARTING 상태가 됩니다.
  - Apache Flink 1.8 및 이전 버전을 사용하는 애플리케이션의 경우 플리케이션은 FAILING 상태가 됩니다.

  그러면 작업이 다시 시작될 수 있는지 여부에 따라 애플리케이션이 RESTARTING 또는 FAILED 상태로 진행됩니다.

  애플리케이션의 CloudWatch 로그에서 상태 변경을 검사하여 작업 상태를 확인할 수 있습니다.

배치 워크로드

Managed Service for Apache Flink는 Apache Flink 배치 워크로드 실행을 지원합니다. 배치 작업에서 Apache Flink 작업이 FINISHED 상태가 되면 Managed Service for Apache Flink 애플리케이션 상태가 준비로 설정됩니다. Flink 작업 상태에 대한 자세한 설명은 작업 및 예약을 참조하십시오.

애플리케이션 리소스

이 섹션에서는 애플리케이션에서 사용하는 시스템 리소스에 대해 설명합니다. Managed Service for Apache Flink가 리소스를 프로비저닝하고 사용하는 방법을 이해하면 성능이 뛰어나고 안정적인 Apache Flink용 관리형 서비스 애플리케이션을 설계, 작성 및 유지 관리하는 데 도움이 됩니다.

Managed Service for Apache Flink 애플리케이션 리소스

Managed Service for Apache Flink는 Apache Flink 애플리케이션을 호스팅하기 위한 환경을 만드는 AWS 서비스입니다. Managed Service for Apache Flink는 Kinesis 처리 단위(KPU)라는 단위를 사용하여 리소스를 제공합니다.

하나의 KPU는 다음과 같은 시스템 리소스를 나타냅니다:

- 하나의 CPU 코어
- 4GB 메모리. 이 중 1GB는 기본 메모리이고 3GB는 힙 메모리입니다.
- 50GB의 여유 디스크 공간

KPU는 작업과 하위 작업이라는 별개의 실행 단위로 애플리케이션을 실행합니다. 하위 작업은 스레드와 같다고 생각할 수 있습니다.

애플리케이션에서 사용할 수 있는 KPU 수는 애플리케이션 `Parallelism` 설정을 애플리케이션 `ParallelismPerKPU` 설정으로 나눈 값과 같습니다.

애플리케이션 병렬성에 대한 자세한 설명은 스케일링 섹션을 참조하십시오.

Apache Flink 애플리케이션 리소스

Apache Flink 환경에서는 작업 슬롯이라는 단위를 사용하여 애플리케이션에 리소스를 할당합니다. Managed Service for Apache Flink는 애플리케이션에 리소스를 할당할 때 하나 이상의 Apache Flink 작업 슬롯을 단일 KPU에 할당합니다. 단일 KPU에 할당된 슬롯 수는 애플리케이션의 `ParallelismPerKPU` 설정과 같습니다. 작업 슬롯에 대한 자세한 설명은 Apache Flink 설명서의 작업 스케줄링을 참조하십시오.

연산자 병렬성

연산자가 사용할 수 있는 최대 하위 작업 수를 설정할 수 있습니다. 이 값을 연산자 병렬성이라고 합니다. 기본적으로 애플리케이션의 각 연산자의 병렬성은 애플리케이션의 병렬성과 동일합니다. 즉, 기본

적으로 애플리케이션의 각 연산자는 필요한 경우 애플리케이션에서 사용 가능한 모든 하위 작업을 사용할 수 있습니다.

`setParallelism` 메서드를 사용하여 애플리케이션의 연산자 병렬성을 설정할 수 있습니다. 이 방법을 사용하면 각 연산자가 한 번에 사용할 수 있는 하위 작업의 수를 제어할 수 있습니다.

연산자 연결에 대한 자세한 설명은 Apache Flink 설명서의 작업 연결 및 리소스 그룹을 참조하십시오.

연산자 연결

일반적으로 각 연산자는 별도의 하위 작업을 사용하여 실행하지만 여러 연산자가 항상 순서대로 실행되는 경우 런타임에서 이들 모두를 동일한 작업에 할당할 수 있습니다. 이 프로세스를 연산자 연결이라고 합니다.

순차 연산자 여러 개가 모두 같은 데이터에 대해 연산을 수행하는 경우 여러 연산자를 단일 작업으로 연결할 수 있습니다. 이를 실현하기 위해 필요한 몇 가지 기준은 다음과 같습니다:

- 운영자는 일대일 단순 전달을 수행합니다.
- 연산자는 모두 동일한 연산자 병렬성을 갖습니다.

애플리케이션이 연산자를 단일 하위 작업으로 연결하면 서비스가 네트워크 작업을 수행하고 연산자별로 하위 작업을 할당할 필요가 없으므로 시스템 리소스가 절약됩니다. 애플리케이션에서 연산자 연결을 사용하고 있는지 확인하려면 Managed Service for Apache Flink 콘솔의 작업 그래프를 살펴보세요. 애플리케이션의 각 꼭짓점은 하나 이상의 연산자를 나타냅니다. 그래프에는 체인으로 연결된 연산자가 단일 꼭지점으로 표시됩니다.

DataStream API

Apache Flink 애플리케이션은 Apache Flink DataStream API를 사용하여 데이터 스트림의 데이터를 변환합니다.

이 섹션은 다음 주제를 포함합니다.

- API를 사용하여 Apache Flink용 관리형 서비스에서 커넥터를 사용하여 데이터 이동 DataStream : 이러한 구성 요소는 애플리케이션과 외부 데이터 소스 및 목적지 간에 데이터를 이동합니다.
- DataStream API를 사용한 Managed Service for Apache Flink의 연산자를 사용하여 데이터 변환: 이러한 구성 요소는 응용 프로그램 내의 데이터 요소를 변환하거나 그룹화합니다.

- DataStream API를 사용하여 Managed Service for Apache Flink에서 이벤트 추적: 이 항목에서는 Managed Service for Apache Flink가 DataStream API를 사용할 때 이벤트를 추적하는 방법에 대해 설명합니다.

API를 사용하여 Apache Flink용 관리형 서비스에서 커넥터를 사용하여 데이터 이동 DataStream

Apache Flink용 Amazon 관리형 서비스 DataStream API에서 커넥터는 Apache Flink용 관리형 서비스 애플리케이션에서 데이터를 내보내고 받는 소프트웨어 구성 요소입니다. 커넥터는 파일과 디렉터리에서 읽을 수 있는 유연한 통합입니다. 커넥터는 Amazon 서비스 및 제3자 시스템과 상호 작용하기 위한 완전한 모듈로 구성됩니다.

커넥터 유형은 다음을 포함합니다:

- 소스: Kinesis 데이터 스트림, 파일 또는 기타 데이터 소스에서 애플리케이션에 데이터를 제공합니다.
- 싱크: 애플리케이션에서 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림 또는 기타 데이터 대상으로 데이터를 전송합니다.
- 비동기식 I/O: 스트림 이벤트를 강화하기 위해 데이터 소스(예: 데이터베이스)에 대한 비동기 액세스를 제공합니다.

사용 가능한 커넥터

Apache Flink 프레임워크에는 다양한 소스의 데이터에 액세스하기 위한 커넥터가 포함되어 있습니다. Apache Flink 프레임워크에서 사용할 수 있는 커넥터에 대한 자세한 설명은 Apache Flink 설명서의 커넥터를 참조하세요.

> **Warning**
>
> Flink 1.6, 1.8, 1.11 또는 1.13에서 실행되는 애플리케이션이 중동(UAE), 아시아 태평양(하이데라바드), 이스라엘(텔아비브), 유럽(취리히), 아시아 태평양(멜버른) 또는 아시아 태평양(자카르타) 지역에서 실행하려는 경우 업데이트된 커넥터로 애플리케이션 아카이브를 재구축하거나 Flink 1.15로 업그레이드해야 할 수 있습니다. 다음은 권장 가이드라인입니다:

커넥터 업그레이드

Fl 버 전	커넥터 사용	해 결 방 법
1. Firehose - 1.		애 플 리 케 이 션 은 최 신 AWS 지 역 을 인 식 하 지 못 하 는 오 래 된 버 전 의 Firehose 커 넥

Fl 버 전	커넥터 사용	해 결 방 법
		터 를 사 용 합 니 다. Firehose 커 넥 터 버 전 2.1.0 으 로 애 플 리 케 이 션 아 카 이 브 를 다 시 구 축 하

Fl 버 전	커넥터 사용	해 결 방 법 세 요. v2.1.0

Fl 버 전	커넥터 사용	해 결 방 법
1.	Kinesis	귀 하 의 애 플 리 케 이 션 은 새 로 운 AWS 지 역 을 인 식 하 지 못 하 는 오 래 된 버 전 의 Flink

Fl 버 전	커넥터 사용	해 결 방 법
		Kinesis 커넥터를 사용하고 있습니다. Flink Kinesis 커넥터 버전 1.6.1을 사용하여 애플리케이션

Fl 버 전	커넥터 사용	해결 방법
		아카이브를 다시 구축하세요. https://github.com/awslabs/amazon-kinesis-connector-flink / 트리/1.6.1

Fl 버 전	커넥터 사용	해 결 방 법
1.	Kinesis	귀하의 애플리케이션은 새로운 AWS 지역을 인식하지 못하는 오래된 버전의 Flink

Fl 버 전	커넥터 사용	해 결 방 법
		Kinesis 커넥터를 사용하고 있습니다. Flink Kinesis 커넥터 버전 2.4.1을 사용하여 애플리케이션

Fl 버 전	커넥터 사용	해결 방법
		아카이브를 다시 구축하세요. https://github.com/awslabs/amazon-kinesis-connector-flink / 트리/2.4.1

Fl 버 전	커넥터 사용	해 결 방 법
1. 및 1.	Kinesis	귀 하 의 애 플 리 케 이 션 은 새 로 운 AWS 지 역 을 인 식 하 지 못 하 는 오 래 된 버 전 의 Flink

Fl 버 전	커넥터 사용	해 결 방 법
		Kinesis 커넥터를 사용하고 있습니다. 안타깝게도 Flink는 더 이상 1.6/1.13 커넥터에 대한 패치

Fl 버 전	커넥터 사용	해결 방법
		나버그 수정을 릴리스하지 않습니다. Flink 1.15로 애플리케이션 아카이브를 다시 구

Fl 버 전	커넥터 사용	해결 방법
		축하여 Flink 1.15로 업데이트하는 것이 좋습니다.

Managed Service for Apache Flink에 스트리밍 데이터 소스 추가

Apache Flink는 파일, 소켓, 컬렉션, 맞춤 소스에서 읽을 수 있는 커넥터를 제공합니다. 애플리케이션 코드에서 Apache Flink 소스를 사용하여 스트림으로부터 데이터를 수신합니다. 이 섹션에서는 Amazon 서비스에 사용할 수 있는 소스를 설명합니다.

Kinesis Data Streams

`FlinkKinesisConsumer` 소스는 Amazon Kinesis 데이터 스트림에서 애플리케이션으로 스트리밍 데이터를 제공합니다.

`FlinkKinesisConsumer` 생성

다음 코드 예는 `FlinkKinesisConsumer`를 생성하는 방법을 보여 줍니다:

```
Properties inputProperties = new Properties();
inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "LATEST");

DataStream<string> input = env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
 new SimpleStringSchema(), inputProperties));
```

`FlinkKinesisConsumer` 사용에 대한 자세한 설명은 Apache Flink 스트리밍 Java 코드 다운로드 및 검사 섹션을 참조하세요.

EFO 컨슈머를 사용하는 **`FlinkKinesisConsumer`** 생성

FlinkKinesisConsumer 이제 향상된 팬아웃 (EFO) 을 지원합니다.

Kinesis 컨슈머가 EFO를 사용하는 경우 Kinesis Data Streams 서비스는 컨슈머가 스트림에서 읽는 다른 컨슈머와 스트림의 고정 대역폭을 공유하지 않고 자체 전용 대역폭을 제공합니다.

Kinesis 소비자와 함께 EFO를 사용하는 방법에 대한 자세한 내용은 FLIP-128: AWS Kinesis 컨슈머를 위한 고급 팬아웃을 참조하세요.

Kinesis 소비자에 다음 파라미터를 설정하여 EFO 소비자를 활성화합니다.

- RECORD_PUBLISHER_TYPE: 애플리케이션이 EFO 소비자를 사용하여 Kinesis Data Stream 데이터에 액세스하도록 이 파라미터를 EFO로 설정하세요.
- EFO_CONSUMER_NAME: 이 파라미터를 이 스트림의 소비자 간에 고유한 문자열 값으로 설정합니다. 동일한 Kinesis Data Stream에서 컨슈머 명칭을 재사용하면 해당 명칭을 사용하던 이전 컨슈머가 종료됩니다.

EFO를 사용하도록 `FlinkKinesisConsumer`를 구성하려면 컨슈머에 다음 파라미터를 추가하세요.

```
consumerConfig.putIfAbsent(RECORD_PUBLISHER_TYPE, "EFO");
consumerConfig.putIfAbsent(EFO_CONSUMER_NAME, "basic-efo-flink-app");
```

EFO 컨슈머를 사용하는 Managed Service for Apache Flink 애플리케이션의 예는 EFO 컨슈머 섹션을 참조하세요.

Amazon MSK

`KafkaSource` 소스는 Amazon MSK 주제에서 스트리밍 데이터를 애플리케이션에 제공합니다.

`KafkaSource` 생성

다음 코드 예는 `KafkaSource`를 생성하는 방법을 보여 줍니다:

```
KafkaSource<String> source = KafkaSource.<String>builder()
    .setBootstrapServers(brokers)
    .setTopics("input-topic")
    .setGroupId("my-group")
    .setStartingOffsets(OffsetsInitializer.earliest())
    .setValueOnlyDeserializer(new SimpleStringSchema())
    .build();

env.fromSource(source, WatermarkStrategy.noWatermarks(), "Kafka Source");
```

`KafkaSource` 사용에 대한 자세한 설명은 MSK 복제 섹션을 참조하세요.

Managed Service for Apache Flink에서 싱크를 사용하여 데이터 쓰기

애플리케이션 코드에서 Apache Flink 싱크를 사용하여 Apache Flink 스트림에서 데이터를 Kinesis Data Streams와 같은 AWS 서비스에 기록합니다.

Apache Flink는 파일, 소켓, 맞춤 싱크를 위한 싱크를 제공합니다. AWS에 사용할 수 있는 싱크는 다음과 같습니다:

Kinesis Data Streams

Apache Flink는 Apache Flink 설명서에서 Kinesis Data Streams 커넥터에 대한 정보를 제공합니다.

입력 및 출력에 Kinesis 데이터 스트림을 사용하는 애플리케이션의 예는 시작하기 (API) DataStream 섹션을 참조하세요.

Amazon S3

Amazon S3 버킷에 객체를 쓰는 데 Apache Flink `StreamingFileSink`를 사용할 수 있습니다.

S3에 객체를 쓰는 방법에 대한 예는 the section called “S3 싱크” 섹션을 참조하세요.

Kinesis Data Firehose

`FlinkKinesisFirehoseProducer`는 Kinesis Data Firehose 서비스를 사용하여 애플리케이션 출력을 저장하기 위한 안정적이고 확장 가능한 Apache Flink 싱크입니다. 이 섹션에서는 Maven 프로젝트를 설정하여 `FlinkKinesisFirehoseProducer`를 생성하고 사용하는 방법을 설명합니다.

주제

`FlinkKinesisFirehoseProducer` 생성

다음 코드 예는 `FlinkKinesisFirehoseProducer`를 생성하는 방법을 보여 줍니다:

```
Properties outputProperties = new Properties();
outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

FlinkKinesisFirehoseProducer<String> sink = new
 FlinkKinesisFirehoseProducer<>(outputStreamName, new SimpleStringSchema(),
 outputProperties);
```

`FlinkKinesisFirehoseProducer` 코드 예

다음 코드 예는 `FlinkKinesisFirehoseProducer`를 생성 및 구성하고 Apache Flink 데이터 스트림에서 Kinesis Data Firehose 서비스로 데이터를 전송하는 방법을 보여줍니다.

```
package com.amazonaws.services.kinesisanalytics;

import
 com.amazonaws.services.kinesisanalytics.flink.connectors.config.ProducerConfigConstants;
import
 com.amazonaws.services.kinesisanalytics.flink.connectors.producer.FlinkKinesisFirehoseProducer
import com.amazonaws.services.kinesisanalytics.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;

import org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;

import java.io.IOException;
import java.util.Map;
import java.util.Properties;

public class StreamingJob {

 private static final String region = "us-east-1";
```

```
private static final String inputStreamName = "ExampleInputStream";
private static final String outputStreamName = "ExampleOutputStream";

private static DataStream<String>
createSourceFromStaticConfig(StreamExecutionEnvironment env) {
 Properties inputProperties = new Properties();
 inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
 inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
"LATEST");

 return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
SimpleStringSchema(), inputProperties));
}

private static DataStream<String>
createSourceFromApplicationProperties(StreamExecutionEnvironment env)
  throws IOException {
 Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
 return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
SimpleStringSchema(),
   applicationProperties.get("ConsumerConfigProperties")));
}

private static FlinkKinesisFirehoseProducer<String>
createFirehoseSinkFromStaticConfig() {
 /*
  * com.amazonaws.services.kinesisanalytics.flink.connectors.config.
  * ProducerConfigConstants
  * lists of all of the properties that firehose sink can be configured with.
  */

 Properties outputProperties = new Properties();
 outputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);

 FlinkKinesisFirehoseProducer<String> sink = new
FlinkKinesisFirehoseProducer<>(outputStreamName,
   new SimpleStringSchema(), outputProperties);
 ProducerConfigConstants config = new ProducerConfigConstants();
 return sink;
}

private static FlinkKinesisFirehoseProducer<String>
createFirehoseSinkFromApplicationProperties() throws IOException {
```

```
  /*
   * com.amazonaws.services.kinesisanalytics.flink.connectors.config.
   * ProducerConfigConstants
   * lists of all of the properties that firehose sink can be configured with.
   */

  Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
  FlinkKinesisFirehoseProducer<String> sink = new
 FlinkKinesisFirehoseProducer<>(outputStreamName,
    new SimpleStringSchema(),
    applicationProperties.get("ProducerConfigProperties"));
  return sink;
 }

 public static void main(String[] args) throws Exception {
  // set up the streaming execution environment
  final StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();

  /*
   * if you would like to use runtime configuration properties, uncomment the
   * lines below
   * DataStream<String> input = createSourceFromApplicationProperties(env);
   */

  DataStream<String> input = createSourceFromStaticConfig(env);

  // Kinesis Firehose sink
  input.addSink(createFirehoseSinkFromStaticConfig());

  // If you would like to use runtime configuration properties, uncomment the
  // lines below
  // input.addSink(createFirehoseSinkFromApplicationProperties());

  env.execute("Flink Streaming Java API Skeleton");
 }
}
```

Kinesis Data Firehose 싱크를 사용하는 방법에 대한 전체 자습서는 the section called “Kinesis Data Firehose Sink” 섹션을 참조하세요.

Managed Service for Apache Flink에서 비동기식 I/O 사용

비동기식 I/O 연산자는 데이터베이스와 같은 외부 데이터 소스를 사용하여 스트림 데이터를 강화합니다. Managed Service for Apache Flink는 스트림 이벤트를 비동기적으로 보강하여 요청을 일괄 처리하여 효율성을 높일 수 있도록 합니다.

자세한 설명은 Apache Flink 설명서의 비동기식 I/O 섹션을 참조하세요.

DataStream API를 사용한 Managed Service for Apache Flink의 연산자를 사용하여 데이터 변환

Managed Service for Apache Flink에서 들어오는 데이터를 변환하려면 Apache Flink 연산자를 사용합니다. Apache Flink 연산자는 하나 이상의 데이터 스트림을 새 데이터 스트림으로 변환합니다. 새 데이터 스트림에는 원래 데이터 스트림에서 수정된 데이터가 포함됩니다. Apache Flink는 25개 이상의 사전 빌드된 스트림 처리 연산자를 제공합니다. 자세한 내용을 알아보려면 Apache Flink 설명서의 연산자를 참조하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 변환 연산자
- 집계 연산자

변환 연산자

다음은 JSON 데이터 스트림의 필드 중 하나에 대한 간단한 텍스트 변환의 예입니다.

이 코드는 변환된 데이터 스트림을 만듭니다. 새 데이터 스트림에는 원래 스트림과 동일한 데이터가 포함되며 TICKER 필드 내용에 " Company" 문자열이 추가됩니다.

```
DataStream<ObjectNode> output = input.map(
    new MapFunction<ObjectNode, ObjectNode>() {
        @Override
        public ObjectNode map(ObjectNode value) throws Exception {
            return value.put("TICKER", value.get("TICKER").asText() + " Company");
        }
    }
);
```

집계 연산자

다음은 집계 연산자의 예시입니다. 코드는 집계된 데이터 스트림을 만듭니다. 연산자는 5초짜리 텀블링 윈도우를 만들고 창에 있는 레코드에 대한 PRICE 값의 합계를 같은 TICKER 값으로 반환합니다.

```
DataStream<ObjectNode> output = input.keyBy(node -> node.get("TICKER").asText())
    .window(TumblingProcessingTimeWindows.of(Time.seconds(5)))
    .reduce((node1, node2) -> {
        double priceTotal = node1.get("PRICE").asDouble() +
 node2.get("PRICE").asDouble();
        node1.replace("PRICE", JsonNodeFactory.instance.numberNode(priceTotal));
    return node1;
});
```

연산자를 사용하는 전체 코드 예제는 시작하기 (API) DataStream 을(를) 참조하세요. 시작하기 애플리케이션의 소스 코드는 Managed Service for Apache Flink Java Examples GitHub 리포지토리의 시작하기에서 확인할 수 있습니다.

DataStream API를 사용하여 Managed Service for Apache Flink에서 이벤트 추적

Managed Service for Apache Flink는 다음 타임스탬프를 사용하여 이벤트를 추적합니다:

- 처리 시간: 각 작업을 실행 중인 시스템의 시스템 시간을 나타냅니다.
- 이벤트 시간: 각 개별 이벤트가 생성 장치에서 발생한 시간을 나타냅니다.
- 수집 시간: 이벤트가 Managed Service for Apache Flink에 입력되는 시간을 나타냅니다.

`setStreamTimeCharacteristic`을 사용하여 스트리밍 환경에서 사용되는 시간을 설정합니다.

```
env.setStreamTimeCharacteristic(TimeCharacteristic.ProcessingTime);
env.setStreamTimeCharacteristic(TimeCharacteristic.IngestionTime);
env.setStreamTimeCharacteristic(TimeCharacteristic.EventTime);
```

타임스탬프에 대한 자세한 설명은 Apache Flink 설명서의 이벤트 시간을 참조하세요.

표 API

Apache Flink 애플리케이션은 Apache Flink 표 API를 사용하여 관계형 모델을 사용하여 스트림의 데이터와 상호 작용합니다. 표 API를 사용하여 표 소스를 사용하여 데이터에 액세스한 다음 표 함수를 사용

하여 표 데이터를 변환하고 필터링합니다. API 함수 또는 SQL 명령을 사용하여 표 형식 데이터를 변환하고 필터링할 수 있습니다.

이 섹션은 다음 주제를 포함합니다:

- 표 API 커넥터: 이러한 구성 요소는 애플리케이션과 외부 데이터 소스 및 목적지 간에 데이터를 이동합니다.
- 표 API 시간 속성: 이 항목에서는 Apache Flink용 관리형 서비스가 표 API를 사용할 때 이벤트를 추적하는 방법에 대해 설명합니다.

표 API 커넥터

Apache Flink 프로그래밍 모델에서 커넥터는 애플리케이션이 다른 AWS 서비스와 같은 외부 소스에서 데이터를 읽거나 쓰는 데 사용하는 구성 요소입니다.

Apache Flink 표 API를 사용하면 다음과 같은 유형의 커넥터를 사용할 수 있습니다.

- 표 API 소스: 표 API 소스 커넥터를 사용하면 API 호출 또는 SQL 쿼리를 사용하여 귀하의 `TableEnvironment` 내에 표를 생성할 수 있습니다.
- 표 API 싱크: SQL 명령을 사용하여 Amazon MSK 주제 또는 Amazon S3 버킷과 같은 외부 소스에 표 데이터를 쓸 수 있습니다.

표 API 소스

데이터 스트림에서 표 소스를 생성합니다. 다음 코드는 Amazon MSK 주제에서 표를 생성합니다.

```
//create the table
    final FlinkKafkaConsumer<StockRecord> consumer = new
 FlinkKafkaConsumer<StockRecord>(kafkaTopic, new KafkaEventDeserializationSchema(),
 kafkaProperties);
    consumer.setStartFromEarliest();
    //Obtain stream
    DataStream<StockRecord> events = env.addSource(consumer);

    Table table = streamTableEnvironment.fromDataStream(events);
```

표 소스에 대한 자세한 설명은 Apache Flink 설명서의 표 및 커넥터를 참조하십시오.

표 API 싱크

표 데이터를 싱크에 쓰려면 SQL로 싱크를 만든 다음 `StreamTableEnvironment` 객체에서 SQL 기반 싱크를 실행합니다.

다음 코드 예는 Amazon S3 싱크에 표 데이터를 쓰는 방법을 보여줍니다:

```
final String s3Sink = "CREATE TABLE sink_table (" +
    "event_time TIMESTAMP," +
    "ticker STRING," +
    "price DOUBLE," +
    "dt STRING," +
    "hr STRING" +
    ")" +
    " PARTITIONED BY (ticker,dt,hr)" +
    " WITH" +
    "(" +
    " 'connector' = 'filesystem'," +
    " 'path' = '" + s3Path + "'," +
    " 'format' = 'json'" +
    ") ";

    //send to s3
    streamTableEnvironment.executeSql(s3Sink);
    filteredTable.executeInsert("sink_table");
```

`format` 파라미터를 사용하여 Apache Flink용 관리형 서비스가 싱크에 출력을 기록하는 데 사용하는 형식을 제어할 수 있습니다. 형식에 대한 자세한 설명은 Apache Flink 설명서의 형식을 참조하십시오.

표 싱크에 대한 자세한 설명은 Apache Flink 설명서의 표 및 커넥터를 참조하십시오.

사용자 정의 소스 및 싱크

기존 Apache Kafka 커넥터를 사용하여 Amazon MSK 및 Amazon S3와 같은 다른 AWS 서비스와 데이터를 주고 받을 수 있습니다. 다른 데이터 소스 및 목적지과 상호 작용하기 위해 자체 소스 및 싱크를 정의할 수 있습니다. 자세한 설명은 Apache Flink 설명서의 사용자 정의 소스 및 싱크를 참조하십시오.

표 API 시간 속성

데이터 스트림의 각 레코드에는 레코드와 관련된 이벤트가 발생한 시기를 정의하는 여러 타임스탬프가 있습니다.

- 이벤트 시간: 레코드를 생성한 이벤트가 발생한 시기를 정의하는 사용자 정의 타임스탬프입니다.
- 통합 시간: 애플리케이션이 데이터 스트림에서 레코드를 검색한 시간입니다.
- 처리 시간: 애플리케이션에서 레코드를 처리한 시간입니다.

Apache Flink 표 API가 레코드 시간을 기반으로 창을 생성할 때, setStreamTimeCatular 메서드를 사용하여 이러한 타임스탬프 중 어떤 타임스탬프를 사용할지 정의합니다.

표 API에서 타임스탬프를 사용하는 방법에 대한 자세한 설명은 Apache Flink 설명서의 시간 속성을 참조하십시오.

Managed Service for Apache Flink와 함께 Python 사용하기

> **Note**
>
> 애플 실리콘 칩이 장착된 새로운 Mac에서 Python Flink 애플리케이션을 개발하는 경우, PyFlink 1.15의 Python 종속성과 관련된 몇 가지 알려진 문제가 발생할 수 있습니다. 이 경우 Docker에서 Python 인터프리터를 실행하는 것이 좋습니다. 단계별 지침은 Apple Silicon Mac에서의 PyFlink 1.15 개발을 참조하세요.

Apache Flink 버전 1.15.2에는 PyFlink 라이브러리를 사용하여 Python 버전 3.8을 사용하여 애플리케이션을 만들 수 있는 지원이 포함되어 있습니다. 다음을 통해 Python을 사용하여 Managed Service for Apache Flink 애플리케이션을 생성합니다:

- `main` 메서드를 사용하여 Python 애플리케이션 코드를 텍스트 파일로 만듭니다.
- 애플리케이션 코드 파일과 Python 또는 Java 종속성을 zip 파일로 번들링한 다음 Amazon S3 버킷에 업로드합니다.
- Amazon S3 코드 위치, 애플리케이션 속성 및 애플리케이션 설정을 지정하여 Managed Service for Apache Flink 애플리케이션을 생성합니다.

상위 수준에서 보면, Python 표 API는 자바 표 API에 대한 래퍼입니다. Python 표 API에 대한 자세한 설명은 Apache Flink 설명서의 Python 표 API 소개를 참조하십시오.

Python 애플리케이션용 Managed Service for Apache Flink 프로그래밍

Apache Flink Python 표 API를 사용하여 Python 애플리케이션용 Managed Service for Apache Flink를 코딩합니다. Apache Flink 엔진은 Python 표 API 명령문(Python가상 머신에서 실행)을 Java 표 API 명령문(Java 가상 머신에서 실행)으로 변환합니다.

Python 표 API를 사용하는 방법은 다음과 같습니다:

- `StreamTableEnvironment`에 대한 참조를 생성합니다.
- `StreamTableEnvironment` 참조에 대해 쿼리를 실행하여 소스 스트리밍 데이터에서 `table` 객체를 생성합니다.
- `table` 객체에 대해 쿼리를 실행하여 출력 표를 생성합니다.
- `StatementSet`을 사용하여 대상에 출력 표를 작성합니다.

Managed Service for Apache Flink에서 Python 표 API를 사용하여 시작하려면 Amazon Managed Service for Apache Flink for Python 시작하기을 참조하십시오.

스트리밍 데이터 읽기 및 쓰기

스트리밍 데이터를 읽고 쓰려면 표 환경에서 SQL 쿼리를 실행합니다.

표 생성

다음 코드 예는 SQL 쿼리를 생성하는 사용자 정의 함수를 보여줍니다. SQL 쿼리는 Kinesis 스트림과 상호 작용하는 표를 생성합니다.

```
def create_table(table_name, stream_name, region, stream_initpos):
   return """ CREATE TABLE {0} (
                `record_id` VARCHAR(64) NOT NULL,
                `event_time` BIGINT NOT NULL,
                `record_number` BIGINT NOT NULL,
                `num_retries` BIGINT NOT NULL,
                `verified` BOOLEAN NOT NULL
              )
              PARTITIONED BY (record_id)
              WITH (
                'connector' = 'kinesis',
                'stream' = '{1}',
                'aws.region' = '{2}',
                'scan.stream.initpos' = '{3}',
```

```
                'sink.partitioner-field-delimiter' = ';',
                'sink.producer.collection-max-count' = '100',
                'format' = 'json',
                'json.timestamp-format.standard' = 'ISO-8601'
              ) """.format(table_name, stream_name, region, stream_initpos)
```

스트리밍 데이터 읽기

다음 코드 예는 표 환경 참조에서 이전 `CreateTable`SQL 쿼리를 사용하여 데이터를 읽는 방법을 보여줍니다:

```
  table_env.execute_sql(create_table(input_table, input_stream, input_region,
stream_initpos))
```

스트리밍 데이터 쓰기

다음 코드 예는 `CreateTable` 예의 SQL 쿼리를 사용하여 출력 표 참조를 생성하는 방법과 `StatementSet`를 사용하여 표과 상호 작용하여 대상 Kinesis 스트림에 데이터를 쓰는 방법을 보여줍니다.

```
  table_result = table_env.execute_sql("INSERT INTO {0} SELECT * FROM {1}"
                      .format(output_table_name, input_table_name))
```

런타임 속성 읽기

애플리케이션 코드를 변경하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.

Java 애플리케이션용 Managed Service for Apache Flink를 사용할 때와 같은 방식으로 애플리케이션의 애플리케이션 속성을 지정합니다. 런타임 속성은 다음과 같은 방법으로 지정할 수 있습니다:

- CreateApplication 작업 사용하기
- UpdateApplication 작업 사용하기
- 콘솔을 사용하여 애플리케이션 구성하기.

Managed Service for Apache Flink 런타임에서 생성하는 `application_properties.json`이라는 json 파일을 읽으면 코드에서 애플리케이션 속성을 검색할 수 있습니다.

다음 코드 예는 `application_properties.json` 파일에서 애플리케이션 속성을 읽는 방법을 보여줍니다:

```
file_path = '/etc/flink/application_properties.json'
   if os.path.isfile(file_path):
       with open(file_path, 'r') as file:
           contents = file.read()
           properties = json.loads(contents)
```

다음 사용자 정의 함수 코드 예는 애플리케이션 속성 객체에서 속성 그룹을 읽는 방법을 보여줍니다: 검색합니다:

```
def property_map(properties, property_group_id):
   for prop in props:
       if prop["PropertyGroupId"] == property_group_id:
           return prop["PropertyMap"]
```

다음 코드 예는 이전 예에서 반환하는 속성 그룹에서 INPUT_STREAM_KEY라는 속성을 읽는 방법을 보여 줍니다.

```
input_stream = input_property_map[INPUT_STREAM_KEY]
```

애플리케이션의 코드 패키지 만들기

Python 애플리케이션을 만든 후에는 코드 파일과 종속성을 zip 파일로 번들로 묶습니다.

zip 파일에는 `main` 메서드가 있는 Python 스크립트가 포함되어야 하며 선택적으로 다음을 포함할 수 있습니다:

- 추가 Python 코드 파일
- JAR 파일의 사용자 정의 Java 코드
- JAR 파일의 Java 라이브러리

> **Note**
>
> 애플리케이션 zip 파일에는 애플리케이션의 모든 종속성이 포함되어야 합니다. 다른 소스의 라이브러리는 애플리케이션에 참조할 수 없습니다.

Managed Service for Apache Flink Python 애플리케이션 생성

코드 파일 지정

애플리케이션의 코드 패키지를 만들었으면 Amazon S3 버킷에 업로드합니다. 그런 다음 콘솔 또는 CreateApplication 작업을 사용하여 애플리케이션을 생성합니다.

CreateApplication 작업을 사용하여 애플리케이션을 만드는 경우 `kinesis.analytics.flink.run.options`라는 특수 애플리케이션 속성 그룹을 사용하여 zip 파일의 코드 파일과 아카이브를 지정합니다. 다음과 같은 유형의 파일을 정의할 수 있습니다:

- python: Python 기본 메서드를 포함하는 텍스트 파일입니다.
- jarfile: Java 사용자 정의 함수를 포함하는 Java JAR 파일입니다.
- pyFiles: 애플리케이션에서 사용할 리소스가 포함된 Python 리소스 파일입니다.
- pyArchives: 애플리케이션의 리소스 파일이 들어 있는 zip 파일입니다.

Apache Flink Python 코드 파일 유형에 대한 자세한 설명은 Apache Flink 설명서의 명령행 사용을 참조하십시오.

> **Note**
>
> Managed Service for Apache Flink는 `pyModule`, `pyExecutable` 또는 `pyRequirements` 파일 유형을 지원하지 않습니다. 코드, 요건 및 종속성은 모두 zip 파일에 있어야 합니다. pip를 사용하여 설치할 종속성을 지정할 수 없습니다.

다음 예 json 스니펫은 애플리케이션의 zip 파일 내에서 파일 위치를 지정하는 방법을 보여줍니다:

```
"ApplicationConfiguration": {
    "EnvironmentProperties": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "kinesis.analytics.flink.run.options",
          "PropertyMap": {
            "python": "MyApplication/main.py",
            "jarfile": "MyApplication/lib/myJarFile.jar",
            "pyFiles": "MyApplication/lib/myDependentFile.py",
            "pyArchives": "MyApplication/lib/myArchive.zip"
          }
```

```
        },
```

Python Managed Service for Apache Flink 애플리케이션 모니터링

애플리케이션의 CloudWatch 로그를 사용하여 Managed Service for Apache Flink Python 애플리케이션을 모니터링할 수 있습니다.

Managed Service for Apache Flink는 Python 애플리케이션을 위해 다음 메시지를 기록합니다:

- 애플리케이션의 `main` 메서드에서 `print()`를 사용하여 콘솔에 기록되는 메시지.
- `logging` 패키지를 사용하여 사용자 정의 함수로 전송된 메시지. 다음 코드 예는 사용자 정의 함수에서 애플리케이션 로그에 쓰는 방법을 보여줍니다.

```
import logging

@udf(input_types=[DataTypes.BIGINT()], result_type=DataTypes.BIGINT())
def doNothingUdf(i):
    logging.info("Got {} in the doNothingUdf".format(str(i)))
    return i
```

- 애플리케이션에서 발생한 오류 메시지.

  애플리케이션에서 `main` 함수에서 예외가 발생하면 애플리케이션 로그에 해당 예외가 나타납니다.

  다음 예는 Python 코드에서 발생한 예외에 대한 로그 항목을 보여줍니다:

```
2021-03-15 16:21:20.000   --------------------------- Python Process Started
 ---------------------------
2021-03-15 16:21:21.000   Traceback (most recent call last):
2021-03-15 16:21:21.000   "  File ""/tmp/flink-
web-6118109b-1cd2-439c-9dcd-218874197fa9/flink-web-upload/4390b233-75cb-4205-
a532-441a2de83db3_code/PythonKinesisSink/PythonUdfUndeclared.py"", line 101, in
 <module>"
2021-03-15 16:21:21.000       main()
2021-03-15 16:21:21.000   "  File ""/tmp/flink-
web-6118109b-1cd2-439c-9dcd-218874197fa9/flink-web-upload/4390b233-75cb-4205-
a532-441a2de83db3_code/PythonKinesisSink/PythonUdfUndeclared.py"", line 54, in main"
2021-03-15 16:21:21.000   "    table_env.register_function(""doNothingUdf"",
 doNothingUdf)"
2021-03-15 16:21:21.000   NameError: name 'doNothingUdf' is not defined
2021-03-15 16:21:21.000   --------------------------- Python Process Exited
 ---------------------------
```

```
2021-03-15 16:21:21.000   Run python process failed
2021-03-15 16:21:21.000   Error occurred when trying to start the job
```

> **Note**
>
> 성능 문제로 인해 애플리케이션 개발 중에는 맞춤 로그 메시지만 사용하는 것이 좋습니다.

CloudWatch 인사이트를 통한 로그 쿼리

다음 CloudWatch 인사이트 쿼리는 애플리케이션의 기본 기능을 실행하는 동안 Python 진입점에서 생성된 로그를 검색합니다:

```
fields @timestamp, message
| sort @timestamp asc
| filter logger like /PythonDriver/
| limit 1000
```

Managed Service for Apache Flink에 대한 런타임 속성

애플리케이션 코드를 다시 컴파일하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- 콘솔에서 런타임 속성 작업
- CLI에서 런타임 속성 작업
- Managed Service for Apache Flink 애플리케이션의 런타임 속성에 액세스

콘솔에서 런타임 속성 작업

콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션에서 런타임 속성을 추가, 업데이트 또는 제거할 수 있습니다.

> **Note**
>
> Managed Service for Apache Flink 콘솔에서 애플리케이션을 만들 때는 런타임 속성을 추가할 수 없습니다.

Managed Service for Apache Flink에 대한 런타임 속성 업데이트

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 애플리케이션을 선택하세요. 애플리케이션 세부 정보를 선택합니다.
3. 애플리케이션 페이지에서 구성을 선택합니다.
4. 속성 섹션을 확장합니다.
5. 속성 섹션의 컨트롤을 사용하여 키-값 쌍으로 속성 그룹을 정의할 수 있습니다. 이러한 컨트롤을 사용하여 속성 그룹 및 런타임 속성을 추가, 업데이트 또는 제거할 수 있습니다.
6. 업데이트를 선택합니다.

CLI에서 런타임 속성 작업

AWS CLI를 사용하여 런타임 속성을 추가, 업데이트 또는 제거할 수 있습니다.

이 섹션에는 애플리케이션의 런타임 속성을 구성하기 위한 API 작업에 대한 예제 요청이 포함되어 있습니다. JSON 파일을 사용하여 API 작업을 입력하는 방법에 대한 자세한 방법은 Managed Service for Apache Flink API 예 코드 섹션을 참조하세요.

> **Note**
>
> 다음 예제의 샘플 계정 ID (*012345678901*)을(를) 계정 ID로 바꾸세요.

애플리케이션 생성 시 런타임 속성 추가

`CreateApplication` 작업에 대한 다음 예제 요청은 애플리케이션을 생성할 때 두 개의 런타임 속성 그룹(`ProducerConfigProperties` 및 `ConsumerConfigProperties`)을 추가합니다.

```
{
```

```
    "ApplicationName": "MyApplication",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

기존 애플리케이션의 런타임 속성 추가 및 업데이트

UpdateApplication 작업에 대한 다음 예제 요청은 기존 응용 프로그램의 런타임 속성을 추가하거나 업데이트합니다.

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 2,
```

```
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": [
        {
          "PropertyGroupId": "ProducerConfigProperties",
          "PropertyMap" : {
            "flink.stream.initpos" : "LATEST",
            "aws.region" : "us-west-2",
            "AggregationEnabled" : "false"
          }
        },
        {
          "PropertyGroupId": "ConsumerConfigProperties",
          "PropertyMap" : {
            "aws.region" : "us-west-2"
          }
        }
      ]
    }
  }
}
```

> **Note**
>
> 속성 그룹에 해당하는 런타임 속성이 없는 키를 사용하는 경우 Managed Service for Apache Flink는 키-값 쌍을 새 속성으로 추가합니다. 속성 그룹의 기존 런타임 속성에 키를 사용하는 경우 Managed Service for Apache Flink에서 속성 값을 업데이트합니다.

런타임 속성 제거

UpdateApplication 작업에 대한 다음 예제 요청은 기존 애플리케이션에서 모든 런타임 속성과 속성 그룹을 제거합니다.

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 3,
  "ApplicationConfigurationUpdate": {
    "EnvironmentPropertyUpdates": {
      "PropertyGroups": []
    }
  }
```

```
}
```

> **Important**
>
> 속성 그룹에서 기존 속성 그룹 또는 기존 속성 키를 생략하면 해당 속성 그룹 또는 속성이 제거됩니다.

Managed Service for Apache Flink 애플리케이션의 런타임 속성에 액세스

`Map<String, Properties>` 객체를 반환하는 정적 `KinesisAnalyticsRuntime.getApplicationProperties()` 메서드를 사용하여 Java 애플리케이션 코드에서 런타임 속성을 검색합니다.

다음은 애플리케이션의 런타임 속성을 검색하는 Java 코드 예제입니다.

```
Map<String, Properties> applicationProperties =
KinesisAnalyticsRuntime.getApplicationProperties();
```

다음과 같이 속성 그룹(`Java.Util.Properties`객체)을 검색합니다.

```
Properties consumerProperties = applicationProperties.get("ConsumerConfigProperties");
```

일반적으로 개별 속성을 검색할 필요 없이 `Properties` 객체를 전달하여 Apache Flink 소스 또는 싱크를 구성합니다. 다음 코드 예제는 런타임 속성에서 검색된 `Properties` 객체를 전달하여 Flink 소스를 만드는 방법을 보여줍니다.

```
private static FlinkKinesisProducer<String> createSinkFromApplicationProperties()
 throws IOException {
  Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
  FlinkKinesisProducer<String> sink = new FlinkKinesisProducer<String>(new
 SimpleStringSchema(),
    applicationProperties.get("ProducerConfigProperties"));

  sink.setDefaultStream(outputStreamName);
  sink.setDefaultPartition("0");
  return sink;
}
```

런타임 속성을 사용하는 전체 코드 예제는 시작하기 (API) DataStream 을(를) 참조하세요. 시작하기 애플리케이션의 소스 코드는 Managed Service for Apache Flink Java Examples GitHub 리포지토리의 시작하기에서 확인할 수 있습니다.

Managed Service for Apache Flink에서 내결함성 구현

체크포인트는 Amazon Managed Service for Apache Flink.에서 내결함성을 구현하는 데 사용하는 방법입니다. 체크포인트는 실행 중인 애플리케이션의 최신 백업으로, 예상치 못한 애플리케이션 중단이나 장애 조치로부터 즉시 복구하는 데 사용됩니다.

Apache Flink 애플리케이션의 체크포인트에 대한 자세한 설명은 Apache Flink 설명서의 체크포인트를 참조하세요.

스냅샷은 수동으로 생성하고 관리되는 애플리케이션 상태 백업입니다. 스냅샷으로 `UpdateApplication` 호출을 통해 애플리케이션을 이전 상태로 복원할 수 있습니다. 자세한 설명은 스냅샷을 사용한 애플리케이션 백업 관리 섹션을 참조하세요.

애플리케이션에 체크포인트가 활성화되어 있는 경우 서비스는 예상치 못한 애플리케이션 재시작 시 애플리케이션 데이터의 백업을 생성하고 로드하여 내결함성을 제공합니다. 이러한 예기치 않은 애플리케이션 재시작은 예상치 못한 작업 재시작, 인스턴스 장애 등으로 인해 발생할 수 있습니다. 이렇게 하면 애플리케이션을 다시 시작하는 동안 장애 없이 실행하는 것과 동일한 의미 체계를 갖게 됩니다.

애플리케이션에 대해 스냅샷을 활성화하고 애플리케이션의 ApplicationRestoreConfiguration을 사용하여 구성한 경우, 서비스는 애플리케이션 업데이트 중이나 서비스 관련 확장 또는 정비 중에 정확히 한 번 처리 의미 체계를 제공합니다.

Managed Service for Apache Flink에서 체크포인트 구성

애플리케이션의 체크포인트 동작을 구성할 수 있습니다. 체크포인트 상태를 유지할지 여부, 상태를 체크포인트에 저장하는 빈도, 한 체크포인트 작업의 종료와 다른 체크포인트 작업의 시작 사이의 최소 간격을 정의할 수 있습니다.

`CreateApplication` 또는 `UpdateApplication` API 작업을 사용하여 다음 설정을 구성합니다.

- `CheckpointingEnabled` — 애플리케이션에서 체크포인트가 활성화되었는지 여부를 나타냅니다.
- `CheckpointInterval` — 체크포인트(지속성) 작업 사이의 시간(밀리초)을 포함합니다.
- `ConfigurationType` — 기본 체크포인트 동작을 사용하려면 이 값을 `DEFAULT`으로 설정합니다. 다른 값을 구성하려면 이 값을 `CUSTOM`으로 설정하세요.

> **Note**
>
> 기본 체크포인트 동작은 다음과 같습니다:
>
> - CheckpointingEnabled: true
> - CheckpointInterval: 60000
> - MinPauseBetweenCheckpoints: 5000
>
> ConfigurationType이 DEFAULT으로 설정된 경우 AWS Command Line Interface을 사용하거나 애플리케이션 코드에서 값을 설정하여 다른 값으로 설정된 경우에도 이전 값을 사용합니다.

> **Note**
>
> Flink 1.15 이후 버전에서는 Managed Service for Apache Flink가 자동 스냅샷 생성, 즉 애플리케이션 업데이트, 크기 조정 또는 중지하는 동안 `stop-with-savepoint`를 사용합니다.

- `MinPauseBetweenCheckpoints` — 한 체크포인트 작업 종료와 다른 체크포인트 작업 시작 사이의 최소 시간(밀리초) 이 값을 설정하면 체크포인트 작업이 `CheckpointInterval` 보다 오래 걸리는 경우에 애플리케이션이 연속으로 체크포인트를 수행하는 것을 방지합니다.

API 예 체크포인트

이 섹션에는 애플리케이션의 체크포인트 구성을 위한 API 작업에 대한 예 요청이 포함되어 있습니다. JSON 파일을 사용하여 API 작업을 입력하는 방법에 대한 자세한 방법은 Managed Service for Apache Flink API 예 코드 섹션을 참조하세요.

새 애플리케이션의 체크포인트 수행 구성

`CreateApplication` 작업을 위한 다음 예 요청은 애플리케이션을 생성할 때 체크포인트 수행을 구성합니다.

```
{
   "ApplicationName": "MyApplication",
   "RuntimeEnvironment":"FLINK-1_15",
   "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration":{
```

```
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "FlinkApplicationConfiguration": {
         "CheckpointConfiguration": {
            "CheckpointingEnabled": "true",
            "CheckpointInterval": 20000,
            "ConfigurationType": "CUSTOM",
            "MinPauseBetweenCheckpoints": 10000
         }
      }
}
```

새 애플리케이션에 대한 체크포인트 수행 비활성화

CreateApplication 작업을 위한 다음 예 요청은 애플리케이션을 생성할 때 체크포인트 수행을 비활성화합니다.

```
{
   "ApplicationName": "MyApplication",
   "RuntimeEnvironment":"FLINK-1_15",
   "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "FlinkApplicationConfiguration": {
         "CheckpointConfiguration": {
            "CheckpointingEnabled": "false"
         }
      }
}
```

기존 애플리케이션의 체크포인트 수행 구성

UpdateApplication 작업을 위한 다음 예 요청은 기존 애플리케이션의 체크포인트 수행을 구성합니다.

```
{
   "ApplicationName": "MyApplication",
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "CheckpointConfigurationUpdate": {
            "CheckpointingEnabledUpdate": true,
            "CheckpointIntervalUpdate": 20000,
            "ConfigurationTypeUpdate": "CUSTOM",
            "MinPauseBetweenCheckpointsUpdate": 10000
         }
      }
   }
}
```

기존 애플리케이션에 대한 체크포인트 수행 비활성화

UpdateApplication 작업을 위한 다음 예 요청은 기존 애플리케이션에 대한 체크포인트 수행을 비활성화합니다.

```
{
   "ApplicationName": "MyApplication",
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "CheckpointConfigurationUpdate": {
            "CheckpointingEnabledUpdate": false,
            "CheckpointIntervalUpdate": 20000,
            "ConfigurationTypeUpdate": "CUSTOM",
            "MinPauseBetweenCheckpointsUpdate": 10000
         }
      }
   }
}
```

스냅샷을 사용한 애플리케이션 백업 관리

스냅샷은 Apache Flink Savepoint를 구현한 Managed Service for Apache Flink입니다. 스냅샷은 사용자 또는 서비스에 의해 트리거, 생성, 관리되는 애플리케이션 상태 백업입니다. Apache Flink 저장점에

대한 자세한 설명은 Apache Flink 설명서의 저장점을 참조하세요. 스냅샷을 사용하면 애플리케이션 상태의 특정 스냅샷에서 애플리케이션을 다시 시작할 수 있습니다.

> **Note**
>
> 애플리케이션이 올바른 상태 데이터로 제대로 다시 시작하려면 하루에 여러 번 스냅샷을 생성하는 것이 좋습니다. 스냅샷의 올바른 주기는 애플리케이션의 비즈니스 로직에 따라 다릅니다. 스냅샷을 자주 생성하면 최신 데이터를 복구할 수 있지만 비용이 증가하고 더 많은 시스템 리소스가 필요합니다.

Managed Service for Apache Flink에서는 다음 API 작업을 사용하여 스냅샷을 관리합니다.

- `CreateApplicationSnapshot`
- `DeleteApplicationSnapshot`
- `DescribeApplicationSnapshot`
- `ListApplicationSnapshots`

스냅샷 수에 대한 애플리케이션별 한도는 할당량 섹션을 참조하세요. 애플리케이션이 스냅샷 한도에 도달한 경우 스냅샷을 수동으로 만들면 `LimitExceededException` 코드와 함께 실패합니다.

Managed Service for Apache Flink는 스냅샷을 절대 삭제하지 않습니다. `DeleteApplicationSnapshot` 작업을 사용하여 스냅샷을 수동으로 삭제해야 합니다.

애플리케이션을 시작할 때 애플리케이션 상태의 저장된 스냅샷을 로드하려면 `StartApplication` 또는 `UpdateApplication` 작업의 `ApplicationRestoreConfiguration` 파라미터를 사용하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 자동 스냅샷 생성
- 호환되지 않는 상태 데이터가 포함된 스냅샷에서 복원
- 스냅샷 API 예

자동 스냅샷 생성

`SnapshotsEnabled`가 애플리케이션의 ApplicationSnapshotConfiguration에서 `true`로 설정된 경우 Managed Service for Apache Flink는 애플리케이션이 업데이트, 확장 또는 중지될 때 스냅샷을 자동으로 생성하고 사용하여 정확히 한 번 처리 의미 체계를 제공합니다.

> **Note**
>
> `ApplicationSnapshotConfiguration::SnapshotsEnabled`를 `false`로 설정하면 애플리케이션 업데이트 중에 데이터가 손실될 수 있습니다.

> **Note**
>
> Managed Service for Apache Flink는 스냅샷 생성 중에 중간 저장점을 트리거합니다. Flink 버전 1.15 이상에서는 중간 저장점이 더 이상 부작용을 일으키지 않습니다. 저장점 트리거를 참조하세요.

자동으로 생성된 스냅샷의 품질은 다음과 같습니다.

- 스냅샷은 서비스에 의해 관리되지만 ListApplicationSnapshots 작업을 사용하여 스냅샷을 볼 수 있습니다. 자동으로 생성된 스냅샷은 스냅샷 한도 계산에 포함됩니다.
- 애플리케이션이 스냅샷 한도를 초과하는 경우 수동으로 생성한 스냅샷은 실패하지만 Managed Service for Apache Flink 서비스는 애플리케이션이 업데이트, 확장 또는 중지될 때 여전히 스냅샷을 성공적으로 생성합니다. 수동으로 더 많은 스냅샷을 생성하기 전에 먼저 DeleteApplicationSnapshot 작업을 사용하여 스냅샷을 수동으로 삭제해야 합니다.

호환되지 않는 상태 데이터가 포함된 스냅샷에서 복원

스냅샷에는 연산자에 대한 정보가 포함되어 있기 때문에 이전 애플리케이션 버전 이후 변경된 연산자의 스냅샷에서 상태 데이터를 복원하면 예상치 못한 결과가 발생할 수 있습니다. 현재 연산자에 해당하지 않는 스냅샷에서 상태 데이터를 복원하려고 시도하면 애플리케이션에 오류가 발생합니다. 오류가 발생한 애플리케이션은 `STOPPING` 또는 `UPDATING` 상태에서 멈춥니다.

애플리케이션이 호환되지 않는 상태 데이터가 포함된 스냅샷에서 복원할 수 있도록 하려면 FlinkRunConfiguration의 `AllowNonRestoredState` 파라미터가 UpdateApplication 작업을 사용하도록 `true`로 설정하세요.

더 이상 사용하지 않는 스냅샷에서 애플리케이션을 복원하면 다음과 같은 동작이 나타납니다.

- 연산자 추가: 새 연산자가 추가된 경우 저장점에는 새 연산자에 대한 상태 데이터가 없습니다. 오류가 발생하지 않으며 `AllowNonRestoredState`를 설정할 필요도 없습니다.
- 연산자 삭제: 기존 연산자가 삭제된 경우 저장점에는 누락된 연산자에 대한 상태 데이터가 있습니다. `AllowNonRestoredState`를 `true`로 설정하지 않으면 오류가 발생합니다.
- 연산자 수정: 파라미터 유형을 호환 가능한 유형으로 변경하는 등 호환되는 변경이 이루어진 경우 애플리케이션은 사용되지 않는 스냅샷에서 복원할 수 있습니다. 스냅샷에서 복원하는 방법에 대한 자세한 설명은 Apache Flink 설명서의 저장점을 참조하세요. Apache Flink 버전 1.8 이상을 사용하는 애플리케이션은 다른 스키마를 사용한 스냅샷에서 복원할 수 있습니다. Apache Flink 버전 1.6을 사용하는 애플리케이션은 복원할 수 없습니다. 2단계 커밋 싱크에서 사용자가 생성한 스냅샷(CreateApplication Snapshot) 대신 시스템 스냅샷(SwS)을 사용하는 것이 좋습니다.

  Flink의 경우 Managed Service for Apache Flink는 스냅샷 생성 중에 중간 저장점을 트리거합니다. Flink 1.15 이상 버전에서는 중간 저장점이 더 이상 부작용을 일으키지 않습니다. 저장점 트리거를 참조하세요.

기존 저장점 데이터와 호환되지 않는 애플리케이션을 재개해야 하는 경우 StartApplication 작업의 `ApplicationRestoreType` 파라미터를 `SKIP_RESTORE_FROM_SNAPSHOT`으로 설정하여 스냅샷에서 복원을 건너뛰는 것이 좋습니다.

Apache Flink가 병립될 수 없는 상태 데이터를 처리하는 방법에 대한 자세한 설명은 Apache Flink 설명서의 상태 스키마 개선을 참조하세요.

스냅샷 API 예

이 섹션에는 애플리케이션에서 스냅샷을 사용하기 위한 API 작업에 대한 예 요청이 포함되어 있습니다. JSON 파일을 사용하여 API 작업을 입력하는 방법에 대한 자세한 방법은 Managed Service for Apache Flink API 예 코드 섹션을 참조하세요.

애플리케이션에 대한 스냅샷 활성화

`UpdateApplication` 작업을 위한 다음 예 요청은 애플리케이션에 대한 스냅샷을 활성화합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "ApplicationSnapshotConfigurationUpdate": {
         "SnapshotsEnabledUpdate": "true"
```

```
        }
    }
}
```

스냅샷 생성

CreateApplicationSnapshot 작업을 위한 다음 예 요청은 현재 애플리케이션 상태의 스냅샷을 생성합니다.

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot"
}
```

애플리케이션에 대한 스냅샷 나열

ListApplicationSnapshots 작업을 위한 다음 예 요청은 현재 애플리케이션 상태의 처음 50개 스냅샷이 나열합니다.

```
{
   "ApplicationName": "MyApplication",
   "Limit": 50
}
```

애플리케이션 스냅샷에 대한 세부 정보 나열

DescribeApplicationSnapshot 작업을 위한 다음 예 요청은 애플리케이션 스냅샷에 대한 세부 정보를 나열합니다.

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot"
}
```

스냅샷 삭제

DeleteApplicationSnapshot 작업을 위한 다음 예 요청은 이전에 저장한 스냅샷을 삭제합니다. ListApplicationSnapshots 또는 DeleteApplicationSnapshot 다음 중 하나를 사용하여 SnapshotCreationTimestamp 값을 가져올 수 있습니다.

```
{
```

```
   "ApplicationName": "MyApplication",
   "SnapshotName": "MyCustomSnapshot",
   "SnapshotCreationTimestamp": 12345678901.0,
}
```

명명된 스냅샷을 사용하여 애플리케이션 재시작

StartApplication 작업을 위한 다음 예 요청은 특정 스냅샷에 저장된 상태를 사용하여 애플리케이션을 시작합니다.

```
{
   "ApplicationName": "MyApplication",
   "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_CUSTOM_SNAPSHOT",
         "SnapshotName": "MyCustomSnapshot"
      }
   }
}
```

최신 스냅샷을 사용하여 애플리케이션 재시작

StartApplication작업을 위한 다음 예 요청은 가장 최근의 스냅샷을 사용하여 애플리케이션을 시작합니다.

```
{
   "ApplicationName": "MyApplication",
   "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
      }
   }
}
```

스냅샷을 사용하지 않고 애플리케이션 재시작

StartApplication작업을 위한 다음 예 요청은 스냅샷이 있더라도 애플리케이션 상태를 로드하지 않고 애플리케이션을 시작합니다.

```
{
   "ApplicationName": "MyApplication",
   "RunConfiguration": {
```

```
      "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "SKIP_RESTORE_FROM_SNAPSHOT"
      }
   }
}
```

Managed Service for Apache Flink의 애플리케이션 조정

Amazon Managed Service for Apache Flink의 작업 병렬 실행 및 리소스 할당을 구성하여 규모 조정을 구현할 수 있습니다. Apache Flink가 작업의 병렬 인스턴스를 예약하는 방법에 대한 자세한 내용을 알아보려면 Apache Flink 설명서의 병렬 실행을 참조하세요.

주제

- 애플리케이션 병렬 처리 및 KPU 구성 ParallelismPer
- Kinesis 처리 단위 할당
- 애플리케이션의 병렬성 업데이트
- 자동 조정

애플리케이션 병렬 처리 및 KPU 구성 ParallelismPer

다음 `ParallelismConfiguration` 속성을 사용하여 Managed Service for Apache Flink 애플리케이션 작업(예: 소스에서 읽기 또는 연산자 실행)에 대한 병렬 실행을 구성합니다:

- `Parallelism` — 이 속성을 사용하여 기본 Apache Flink 애플리케이션 병렬성을 설정합니다. 모든 연산자, 소스 및 싱크는 애플리케이션 코드에서 재정의되지 않는 한 이 병렬성으로 실행됩니다. 기본값은 1이고, 기본 최대값은 256입니다.
- `ParallelismPerKPU` — 이 속성을 사용하여 애플리케이션의 Kinesis 처리 단위(KPU)에 따라 예약할 수 있는 병렬 작업 수를 설정합니다. 기본값은 1이고 최대값은 8입니다. 블로킹 작업(예: I/O)이 있는 애플리케이션의 경우 `ParallelismPerKPU` 값이 높을수록 KPU 리소스가 완전히 활용됩니다.

> **Note**
>
> `Parallelism`의 한도는 KPU 한도(기본 값 64)의 `ParallelismPerKPU`배와 같습니다. 한도 증가를 요청하여 KPU 한도를 늘릴 수 있습니다. 한도 증가를 요청하는 방법에 대한 지침은 Service Quotas의 “한도 증가를 요청하려면”을 참조하세요.

특정 연산자의 작업 병렬성 설정에 대한 자세한 설명은 Apache Flink 설명서의 병렬성 설정: 연산자를 참조하십시오.

Kinesis 처리 단위 할당

Amazon Managed Service for Apache Flink는 용량을 KPU로 프로비저닝합니다. 단일 KPU는 1개의 vCPU 및 4GB의 메모리를 제공합니다. 할당된 모든 KPU에 대해 50GB의 실행 중인 애플리케이션 스토리지도 제공됩니다.

Managed Service for Apache Flink는 다음과 같이 `Parallelism` 및 `ParallelismPerKPU` 속성을 사용하여 애플리케이션을 실행하는 데 필요한 KPU를 계산합니다:

```
Allocated KPUs for the application = Parallelism/ParallelismPerKPU
```

Managed Service for Apache Flink는 처리량 또는 처리 활동의 급증에 대응하여 애플리케이션 리소스를 신속하게 제공합니다. 활동 급증이 지난 후 애플리케이션에서 리소스를 점진적으로 제거합니다. 리소스 자동 할당을 비활성화하려면 나중에 애플리케이션의 병렬성 업데이트에서 설명하는 대로 `AutoScalingEnabled` 값을 `false`로 설정합니다.

애플리케이션에 대한 KPU 기본 한도는 64입니다. 이 한도 증가를 요청하는 방법에 대한 지침은 Service Quotas의 "한도 증가를 요청하려면"을 참조하십시오.

> **Note**
>
> 오케스트레이션을 위해 추가 KPU가 부과됩니다. 자세한 설명은 Managed Service for Apache Flink 요금을 참조하세요.

애플리케이션의 병렬성 업데이트

이 섹션에는 애플리케이션의 병렬성을 설정하는 API 작업에 대한 샘플 요청이 포함되어 있습니다. API 작업과 함께 요청 블록을 사용하는 방법에 대한 추가 예와 지침은 Managed Service for Apache Flink API 예 코드을 참조하십시오.

다음 예 `CreateApplication` 작업 요청은 애플리케이션을 만들 때 병렬성을 설정합니다.

```
{
   "ApplicationName": "string",
   "RuntimeEnvironment":"FLINK-1_15",
   "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
```

```
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration":{
      "CodeContent":{
         "S3ContentLocation":{
            "BucketARN":"arn:aws:s3:::mybucket",
            "FileKey":"myflink.jar",
            "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
            }
         },
      "CodeContentType":"ZIPFILE"
   },
      "FlinkApplicationConfiguration": {
         "ParallelismConfiguration": {
            "AutoScalingEnabled": "true",
            "ConfigurationType": "CUSTOM",
            "Parallelism": 4,
            "ParallelismPerKPU": 4
         }
      }
   }
}
```

UpdateApplication 작업을 위한 다음 예 요청은 기존 애플리케이션에 대한 병렬성을 설정합니다.

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "ParallelismConfigurationUpdate": {
            "AutoScalingEnabledUpdate": "true",
            "ConfigurationTypeUpdate": "CUSTOM",
            "ParallelismPerKPUUpdate": 4,
            "ParallelismUpdate": 4
         }
      }
   }
}
```

UpdateApplication 작업을 위한 다음 예 요청은 기존 애플리케이션에 대한 병렬성을 비활성화합니다.

```
{
```

```
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "ParallelismConfigurationUpdate": {
            "AutoScalingEnabledUpdate": "false"
         }
      }
   }
}
```

자동 조정

Managed Service for Apache Flink는 대부분의 시나리오에서 소스의 데이터 처리량과 연산자의 복잡성을 수용할 수 있도록 애플리케이션의 병렬성을 탄력적으로 조정합니다. Managed Service for Apache Flink는 애플리케이션의 리소스(CPU) 사용량을 모니터링하고 그에 따라 애플리케이션의 병렬성을 탄력적으로 늘리거나 줄입니다.

- CloudWatch 지표가 15분 동안 75% 이상이거나 그 이상이면 애플리케이션이 확장 (병렬 처리 증가) `containerCPUUtilization` 됩니다. 즉, 75% 이상인 1분 기간의 연속 데이터 포인트가 15개 있을 때 `ScaleUp` 작업이 트리거됩니다.
- CPU 사용량이 6시간 동안 10% 미만으로 유지되면 애플리케이션이 축소됩니다 (병렬성 감소). 즉, 10% 미만인 1분 기간의 연속 데이터 포인트가 360개 있을 때 `ScaleDown` 작업이 트리거됩니다.

> **Note**
>
> 1분 이상의 최대 `containerCPUUtilization` 기간을 참조하여 조정 작업에 사용되는 데이터 포인트와의 상관 관계를 찾을 수 있지만, 작업이 트리거된 정확한 순간을 반영할 필요는 없습니다.

Managed Service for Apache Flink는 애플리케이션 `CurrentParallelism` 값을 애플리케이션의 `Parallelism` 설정보다 낮추지 않습니다.

Managed Service for Apache Flink 서비스가 애플리케이션을 확장할 때는 `AUTOSCALING` 상태가 됩니다. DescribeApplication또는 ListApplications작업을 사용하여 현재 애플리케이션 상태를 확인할 수 있습니다. 서비스가 애플리케이션을 확장하는 동안 사용할 수 있는 유효한 API 작업은 `Force` 파라미터를 StopApplication로 설정한 상태에서만 사용할 수 `true` 있습니다.

AutoScalingEnabled 속성(FlinkApplicationConfiguration에 속함)을 사용하여 Auto Scaling 동작을 활성화하거나 비활성화할 수 있습니다. Managed Service for Apache Flink가 애플리케이션 parallelism 및 parallelismPerKPU 설정에 따라 제공하는 KPU에 대한 요금이 사용자 AWS 계정에 청구됩니다. 활동이 급증하면 Managed Service for Apache Flink 비용이 증가합니다.

요금에 대한 자세한 설명은 Amazon Managed Service for Apache Flink 요금을 참조하세요.

애플리케이션 규모 조정에 대해 다음을 유의하십시오:

- 자동 조정은 기본적으로 활성화되어 있습니다.
- 조정 기능은 Studio 노트북에는 적용되지 않습니다. 그러나 Studio Notebook을 내구성 상태의 애플리케이션으로 배포하는 경우 배포된 애플리케이션에 조정 작업이 적용됩니다.
- 애플리케이션의 기본 한도는 64 KPU입니다. 자세한 설명은 할당량 섹션을 참조하세요.
- 자동 크기 조정이 애플리케이션 병렬성을 업데이트하면 애플리케이션 다운타임이 발생합니다. 이러한 다운타임을 방지하려면 다음을 수행하십시오:
  - 자동 조정 사용 중지
  - UpdateApplication액션을 사용하여 애플리케이션의 parallelism parallelismPerKPU AX를 구성하십시오. 애플리케이션의 병렬성 설정 설정에 대한 자세한 설명은 다음 the section called “애플리케이션의 병렬성 업데이트”을 참조하세요.
  - 애플리케이션의 리소스 사용량을 주기적으로 모니터링하여 애플리케이션의 워크로드에 대한 병렬성 설정이 올바른지 확인하십시오. 할당 리소스 사용량 모니터링에 대한 자세한 설명은 the section called “Managed Service for Apache Flink의 지표 및 차원”을 참조하십시오.

maxParallelism 고려 사항

- 자동 크기 조정 로직은 Flink 작업을 병렬성으로 조정하여 작업 및 연산자 maxParallelism에 간섭을 일으키는 것을 방지합니다. 예를 들어 소스만 있고 싱크만 있는 간단한 작업의 경우 소스가 maxParallelism 16이고 sink가 8인 경우 작업이 8을 초과하도록 자동 크기 조정되지 않습니다.
- 어떤 작업에 대해 maxParallelism이 설정되지 않은 경우, Flink는 기본적으로 128로 설정됩니다. 따라서 작업을 128보다 높은 병렬성으로 실행해야 한다고 생각되면 해당 숫자를 애플리케이션에 맞게 설정해야 합니다.
- 작업이 자동 크기 조정될 것으로 예상되지만 확인되지 않는 경우 maxParallelism 값이 이를 허용하는지 확인하세요.

자세한 설명은 Apache Flink의 고급 모니터링 및 자동 크기 조정을 참조하십시오.

예를 들어, 을 참조하십시오 kda-flink-app-autoscaling.

태그 지정 사용

이 섹션에서는 Managed Service for Apache Flink 애플리케이션에 키-값 메타데이터를 추가하는 방법을 설명합니다. 이러한 태그는 다음과 같은 용도로 사용할 수 있습니다:

- 개별 Managed Service for Apache Flink 애플리케이션에 대한 청구 결정 자세한 내용을 알아보려면 청구 및 비용 관리 가이드의 비용 할당 태그 사용을 참조하세요.
- 태그에 근거한 애플리케이션 리소스에 대한 액세스 통제. 자세한 설명은 AWS Identity and Access Management사용자 가이드의 태그를 사용한 액세스 통제를 참조하세요.
- 사용자 정의 용도. 사용자 태그의 존재 유무를 기반으로 애플리케이션 기능을 정의할 수 있습니다.

태그 지정에 대한 다음 정보를 참조하십시오.

- 애플리케이션 태그의 최대 수는 시스템 태그를 포함합니다. 사용자 정의 애플리케이션 태그의 최대 수는 50입니다.
- 작업에 중복된 `Key` 값이 있는 태그 목록이 포함된 경우 서비스에서 `InvalidArgumentException`가 발생합니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- 애플리케이션이 생성될 때 태그 추가
- 기존 애플리케이션에 대한 태그 추가 또는 업데이트
- 애플리케이션에 대한 태그 나열
- 애플리케이션에서 태그 제거

애플리케이션이 생성될 때 태그 추가

CreateApplication 작업의 `tags` 파라미터를 사용하여 애플리케이션을 만들 때 태그를 추가합니다.

다음 예 요청은 `CreateApplication` 요청에 대한 `Tags` 노드를 보여줍니다.

```
"Tags": [
    {
        "Key": "Key1",
```

```
        "Value": "Value1"
    },
    {
        "Key": "Key2",
        "Value": "Value2"
    }
]
```

기존 애플리케이션에 대한 태그 추가 또는 업데이트

TagResource 작업을 사용하여 애플리케이션에 태그를 추가합니다. UpdateApplication 작업을 사용해서는 애플리케이션에 태그를 추가할 수 없습니다.

기존 태그를 업데이트하려면, 기존 태그의 동일한 키로 태그를 추가합니다.

TagResource 작업을 위한 다음 예 요청은 새 태그를 추가하거나 기존 태그를 업데이트합니다.

```
{
   "ResourceARN": "string",
   "Tags": [
      {
         "Key": "NewTagKey",
         "Value": "NewTagValue"
      },
      {
         "Key": "ExistingKeyOfTagToUpdate",
         "Value": "NewValueForExistingTag"
      }
   ]
}
```

애플리케이션에 대한 태그 나열

기존 태그를 나열하려면 ListTagsForResource 작업을 사용합니다.

ListTagsForResource 작업을 위한 다음 예 요청은 애플리케이션에 대한 태그를 나열합니다.

```
{
   "ResourceARN": "arn:aws:kinesisanalyticsus-west-2:012345678901:application/
MyApplication"
}
```

애플리케이션에서 태그 제거

애플리케이션에서 태그를 제거하려면 UntagResource 작업을 사용합니다.

UntagResource 작업을 위한 다음 예 요청은 애플리케이션에서 태그를 제거합니다.

```
{
   "ResourceARN": "arn:aws:kinesisanalyticsus-west-2:012345678901:application/
MyApplication",
   "TagKeys": [ "KeyOfFirstTagToRemove", "KeyOfSecondTagToRemove" ]
}
```

Managed Service for Apache Flink와 함께 CloudFormation 사용하기

다음 연습은 동일한 스택에서 Lambda 함수를 사용하여 AWS CloudFormation을 통해 생성한 Flink 애플리케이션을 시작하는 방법을 보여줍니다.

시작하기 전에

이 연습을 시작하기 전에 AWS::KinesisAnalytics::Application에서 AWS CloudFormation을 사용하여 Flink 애플리케이션을 생성하는 단계를 따르세요.

Lambda 함수 작성

생성 또는 업데이트 후 Flink 애플리케이션을 시작하려면 kinesisanalyticsv2 애플리케이션 시작 API를 사용합니다. Flink 애플리케이션 생성 후 AWS CloudFormation 이벤트에 의해 호출이 트리거됩니다. 이 연습의 후반부에서 Lambda 함수를 트리거하도록 스택을 설정하는 방법에 대해 설명하겠지만, 먼저 Lambda 함수 선언과 해당 코드를 중점적으로 살펴보겠습니다. 이 예에서는 Python3.8 런타임을 사용합니다.

```
StartApplicationLambda:
    Type: AWS::Lambda::Function
    DependsOn: StartApplicationLambdaRole
    Properties:
      Description: Starts an application when invoked.
      Runtime: python3.8
      Role: !GetAtt StartApplicationLambdaRole.Arn
      Handler: index.lambda_handler
      Timeout: 30
```

```
      Code:
        ZipFile: |
          import logging
          import cfnresponse
          import boto3

          logger = logging.getLogger()
          logger.setLevel(logging.INFO)

          def lambda_handler(event, context):
            logger.info('Incoming CFN event {}'.format(event))

            try:
              application_name = event['ResourceProperties']['ApplicationName']

              # filter out events other than Create or Update,
              # you can also omit Update in order to start an application on Create
 only.
              if event['RequestType'] not in ["Create", "Update"]:
                logger.info('No-op for Application {} because CFN RequestType {} is
 filtered'.format(application_name, event['RequestType']))
                cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

                return

              # use kinesisanalyticsv2 API to start an application.
              client_kda = boto3.client('kinesisanalyticsv2',
 region_name=event['ResourceProperties']['Region'])

              # get application status.
              describe_response =
 client_kda.describe_application(ApplicationName=application_name)
              application_status = describe_response['ApplicationDetail']
['ApplicationStatus']

              # an application can be started from 'READY' status only.
              if application_status != 'READY':
                logger.info('No-op for Application {} because ApplicationStatus {} is
 filtered'.format(application_name, application_status))
                cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

                return

              # create RunConfiguration.
```

```
            run_configuration = {
              'ApplicationRestoreConfiguration': {
                'ApplicationRestoreType': 'RESTORE_FROM_LATEST_SNAPSHOT',
              }
            }

            logger.info('RunConfiguration for Application {}:
{}'.format(application_name, run_configuration))

            # this call doesn't wait for an application to transfer to 'RUNNING'
state.
            client_kda.start_application(ApplicationName=application_name,
RunConfiguration=run_configuration)

            logger.info('Started Application: {}'.format(application_name))
            cfnresponse.send(event, context, cfnresponse.SUCCESS, {})
          except Exception as err:
            logger.error(err)
            cfnresponse.send(event,context, cfnresponse.FAILED, {"Data": str(err)})
```

앞의 코드에서 Lambda는 들어오는 AWS CloudFormation 이벤트를 처리하고, `Create` 및 `Update` 외의 모든 이벤트를 필터링하고, 애플리케이션 상태를 가져와서 상태가 `READY`면 시작합니다. 애플리케이션 상태를 가져오려면 다음과 같이 Lambda 역할을 생성해야 합니다:

Lambda 역할 생성

Lambda가 애플리케이션과 성공적으로 “통신”하고 로그를 기록할 수 있는 역할을 생성합니다. 이 역할은 기본 관리형 정책을 사용하지만 맞춤 정책을 사용하여 범위를 좁히고 싶을 수 있습니다.

```
StartApplicationLambdaRole:
    Type: AWS::IAM::Role
    DependsOn: TestFlinkApplication
    Properties:
      Description: A role for lambda to use while interacting with an application.
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
```

```
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/Amazonmanaged-flinkFullAccess
        - arn:aws:iam::aws:policy/CloudWatchLogsFullAccess
      Path: /
```

Lambda 리소스는 해당 스택에 의존하기 때문에 동일한 스택에서 Flink 애플리케이션을 생성한 후에 생성된다는 점에 유의하세요.

Lambda 함수 호출

이제 Lambda 함수 호출만 남았습니다. 이 작업은 맞춤 리소스를 사용하여 수행됩니다.

```
StartApplicationLambdaInvoke:
    Description: Invokes StartApplicationLambda to start an application.
    Type: AWS::CloudFormation::CustomResource
    DependsOn: StartApplicationLambda
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt StartApplicationLambda.Arn
      Region: !Ref AWS::Region
      ApplicationName: !Ref TestFlinkApplication
```

다음은 Lambda를 사용하여 Flink 애플리케이션을 시작하는 데 필요한 모든 사항입니다. 이제 자체 스택을 만들거나 아래 전체 예를 사용하여 이러한 모든 단계가 실제로 어떻게 작동하는지 확인할 준비가 되었습니다.

전체 예

다음 예는 템플릿 파라미터를 통해 추가로 RunConfiguration 조정한 위 단계를 약간 확장한 버전입니다. 시도해 볼 수 있는 작업 스택입니다. 함께 제공된 참고 사항을 반드시 읽어보세요.

stack.yaml

```
Description: 'kinesisanalyticsv2 CloudFormation Test Application'
Parameters:
  ApplicationRestoreType:
    Description: ApplicationRestoreConfiguration option, can
 be SKIP_RESTORE_FROM_SNAPSHOT, RESTORE_FROM_LATEST_SNAPSHOT or
 RESTORE_FROM_CUSTOM_SNAPSHOT.
    Type: String
    Default: SKIP_RESTORE_FROM_SNAPSHOT
```

```
    AllowedValues: [ SKIP_RESTORE_FROM_SNAPSHOT, RESTORE_FROM_LATEST_SNAPSHOT,
 RESTORE_FROM_CUSTOM_SNAPSHOT ]
  SnapshotName:
    Description: ApplicationRestoreConfiguration option, name of a snapshot to restore
 to, used with RESTORE_FROM_CUSTOM_SNAPSHOT ApplicationRestoreType.
    Type: String
    Default: ''
  AllowNonRestoredState:
    Description: FlinkRunConfiguration option, can be true or false.
    Default: true
    Type: String
    AllowedValues: [ true, false ]
  CodeContentBucketArn:
    Description: ARN of a bucket with application code.
    Type: String
  CodeContentFileKey:
    Description: A jar filename with an application code inside a bucket.
    Type: String
Conditions:
  IsSnapshotNameEmpty: !Equals [ !Ref SnapshotName, '' ]
Resources:
  TestServiceExecutionRole:
    Type: AWS::IAM::Role
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - kinesisanlaytics.amazonaws.com
            Action: sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/AmazonKinesisFullAccess
        - arn:aws:iam::aws:policy/AmazonS3FullAccess
      Path: /
  InputKinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
  OutputKinesisStream:
    Type: AWS::Kinesis::Stream
    Properties:
      ShardCount: 1
```

```
  TestFlinkApplication:
    Type: 'AWS::kinesisanalyticsv2::Application'
    Properties:
      ApplicationName: 'CFNTestFlinkApplication'
      ApplicationDescription: 'Test Flink Application'
      RuntimeEnvironment: 'FLINK-1_15'
      ServiceExecutionRole: !GetAtt TestServiceExecutionRole.Arn
      ApplicationConfiguration:
        EnvironmentProperties:
          PropertyGroups:
            - PropertyGroupId: 'KinesisStreams'
              PropertyMap:
                INPUT_STREAM_NAME: !Ref InputKinesisStream
                OUTPUT_STREAM_NAME: !Ref OutputKinesisStream
                AWS_REGION: !Ref AWS::Region
        FlinkApplicationConfiguration:
          CheckpointConfiguration:
            ConfigurationType: 'CUSTOM'
            CheckpointingEnabled: True
            CheckpointInterval: 1500
            MinPauseBetweenCheckpoints: 500
          MonitoringConfiguration:
            ConfigurationType: 'CUSTOM'
            MetricsLevel: 'APPLICATION'
            LogLevel: 'INFO'
          ParallelismConfiguration:
            ConfigurationType: 'CUSTOM'
            Parallelism: 1
            ParallelismPerKPU: 1
            AutoScalingEnabled: True
        ApplicationSnapshotConfiguration:
          SnapshotsEnabled: True
        ApplicationCodeConfiguration:
          CodeContent:
            S3ContentLocation:
              BucketARN: !Ref CodeContentBucketArn
              FileKey: !Ref CodeContentFileKey
          CodeContentType: 'ZIPFILE'
  StartApplicationLambdaRole:
    Type: AWS::IAM::Role
    DependsOn: TestFlinkApplication
    Properties:
      Description: A role for lambda to use while interacting with an application.
      AssumeRolePolicyDocument:
```

```
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service:
                - lambda.amazonaws.com
            Action:
              - sts:AssumeRole
      ManagedPolicyArns:
        - arn:aws:iam::aws:policy/Amazonmanaged-flinkFullAccess
        - arn:aws:iam::aws:policy/CloudWatchLogsFullAccess
      Path: /
  StartApplicationLambda:
    Type: AWS::Lambda::Function
    DependsOn: StartApplicationLambdaRole
    Properties:
      Description: Starts an application when invoked.
      Runtime: python3.8
      Role: !GetAtt StartApplicationLambdaRole.Arn
      Handler: index.lambda_handler
      Timeout: 30
      Code:
        ZipFile: |
          import logging
          import cfnresponse
          import boto3

          logger = logging.getLogger()
          logger.setLevel(logging.INFO)

          def lambda_handler(event, context):
            logger.info('Incoming CFN event {}'.format(event))

            try:
              application_name = event['ResourceProperties']['ApplicationName']

              # filter out events other than Create or Update,
              # you can also omit Update in order to start an application on Create
only.
              if event['RequestType'] not in ["Create", "Update"]:
                logger.info('No-op for Application {} because CFN RequestType {} is
filtered'.format(application_name, event['RequestType']))
                cfnresponse.send(event, context, cfnresponse.SUCCESS, {})
```

```
                return

            # use kinesisanalyticsv2 API to start an application.
            client_kda = boto3.client('kinesisanalyticsv2',
 region_name=event['ResourceProperties']['Region'])

            # get application status.
            describe_response =
 client_kda.describe_application(ApplicationName=application_name)
            application_status = describe_response['ApplicationDetail']
['ApplicationStatus']

            # an application can be started from 'READY' status only.
            if application_status != 'READY':
              logger.info('No-op for Application {} because ApplicationStatus {} is
 filtered'.format(application_name, application_status))
              cfnresponse.send(event, context, cfnresponse.SUCCESS, {})

              return

            # create RunConfiguration from passed parameters.
            run_configuration = {
              'FlinkRunConfiguration': {
                'AllowNonRestoredState': event['ResourceProperties']
['AllowNonRestoredState'] == 'true'
              },
              'ApplicationRestoreConfiguration': {
                'ApplicationRestoreType': event['ResourceProperties']
['ApplicationRestoreType'],
              }
            }

            # add SnapshotName to RunConfiguration if specified.
            if event['ResourceProperties']['SnapshotName'] != '':
              run_configuration['ApplicationRestoreConfiguration']['SnapshotName'] =
 event['ResourceProperties']['SnapshotName']

            logger.info('RunConfiguration for Application {}:
 {}'.format(application_name, run_configuration))

            # this call doesn't wait for an application to transfer to 'RUNNING'
 state.
            client_kda.start_application(ApplicationName=application_name,
 RunConfiguration=run_configuration)
```

```
            logger.info('Started Application: {}'.format(application_name))
            cfnresponse.send(event, context, cfnresponse.SUCCESS, {})
          except Exception as err:
            logger.error(err)
            cfnresponse.send(event,context, cfnresponse.FAILED, {"Data": str(err)})
  StartApplicationLambdaInvoke:
    Description: Invokes StartApplicationLambda to start an application.
    Type: AWS::CloudFormation::CustomResource
    DependsOn: StartApplicationLambda
    Version: "1.0"
    Properties:
      ServiceToken: !GetAtt StartApplicationLambda.Arn
      Region: !Ref AWS::Region
      ApplicationName: !Ref TestFlinkApplication
      ApplicationRestoreType: !Ref ApplicationRestoreType
      SnapshotName: !Ref SnapshotName
      AllowNonRestoredState: !Ref AllowNonRestoredState
```

다시, 애플리케이션 자체뿐만 아니라 Lambda의 역할도 조정하고 싶을 수 있습니다.

위의 스택을 생성하기 전에 파라미터를 지정하는 것을 잊지 마세요.

parameters.json

```
[
  {
    "ParameterKey": "CodeContentBucketArn",
    "ParameterValue": "YOUR_BUCKET_ARN"
  },
  {
    "ParameterKey": "CodeContentFileKey",
    "ParameterValue": "YOUR_JAR"
  },
  {
    "ParameterKey": "ApplicationRestoreType",
    "ParameterValue": "SKIP_RESTORE_FROM_SNAPSHOT"
  },
  {
    "ParameterKey": "AllowNonRestoredState",
    "ParameterValue": "true"
  }
]
```

YOUR_BUCKET_ARN 및 YOUR_JAR을 특정 요건으로 바꾸세요. 이 가이드에 따라 Amazon S3 버킷과 애플리케이션 jar를 생성할 수 있습니다.

이제 스택을 생성합니다(YOUR_REGION을 원하는 지역(예: us-east-1)으로 대체).

```
aws cloudformation create-stack --region YOUR_REGION --template-body "file://
stack.yaml" --parameters "file://parameters.json" --stack-name "TestManaged Service for
 Apache FlinkStack" --capabilities CAPABILITY_NAMED_IAM
```

이제 https://console.aws.amazon.com/cloudformation으로 이동하여 진행 상황을 확인할 수 있습니다. 생성한 후에는 Flink 애플리케이션이 현재 `Starting` 상태인 것을 확인할 수 있습니다. `Running`을 시작하려면 몇 분이 걸릴 수 있습니다.

자세한 정보는 을(를) 참조하세요.

- AWS CloudFormation을 사용하여 AWS 서비스 속성을 검색하는 네 가지 방법(3부 중 제1부)
- 연습: Amazon Machine Image ID 조회

Managed Service for Apache Flink 대시보드 사용

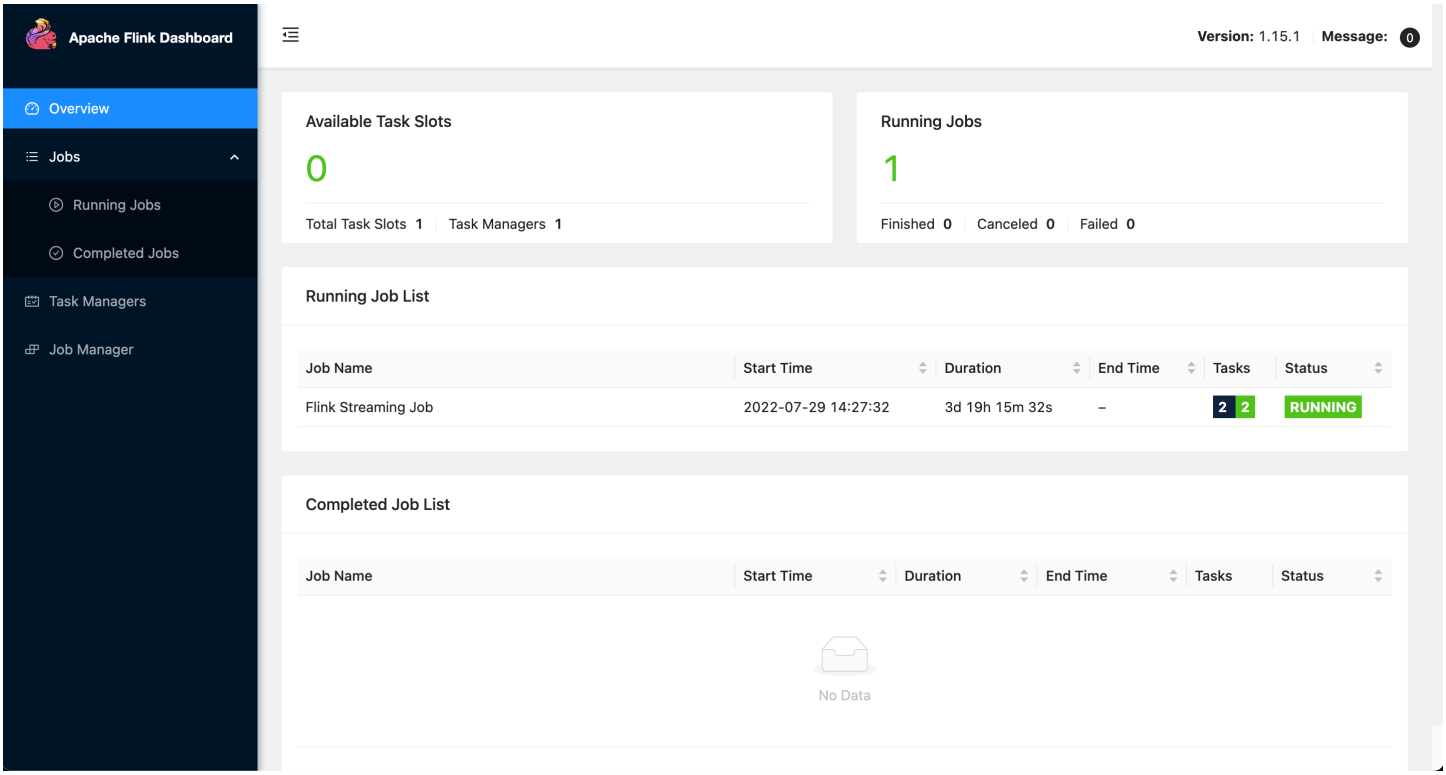
애플리케이션의 Apache Flink 대시보드를 사용하여 Managed Service for Apache Flink 애플리케이션의 상태를 모니터링할 수 있습니다. 애플리케이션 대시보드에 다음 정보가 표시됩니다:

- 사용 중인 리소스(작업 관리자 및 작업 슬롯 포함)
- 실행 중인 작업, 완료한 작업, 취소된 작업, 실패한 작업을 비롯한 작업에 대한 정보

Apache Flink 작업 관리자, 작업 슬롯 및 작업에 대한 자세한 설명은 Apache Flink 웹 사이트의 Apache Flink 아키텍처를 참조하세요.

Managed Service for Apache Flink 애플리케이션과 Apache Flink 대시보드를 사용하는 방법에 대해서는 다음을 참고하세요.

- Managed Service for Apache Flink 애플리케이션용 Apache Flink 대시보드는 읽기 전용입니다. Apache Flink 대시보드를 사용하여 Managed Service for Apache Flink 애플리케이션을 변경할 수 없습니다.
- Apache Flink 대시보드는 Microsoft Internet Explorer와 병립될 수 없습니다.



애플리케이션의 Apache Flink 대시보드에 액세스

Managed Service for Apache Flink 콘솔을 통해 또는 CLI를 사용하여 보안 URL 엔드포인트를 요청하여 애플리케이션의 Apache Flink 대시보드에 액세스할 수 있습니다.

Managed Service for Apache Flink 콘솔을 사용하여 애플리케이션의 Apache Flink 대시보드에 액세스

콘솔에서 애플리케이션의 Apache Flink 대시보드에 액세스하려면 애플리케이션 페이지에서 Apache Flink 대시보드를 선택합니다.

> **Note**
>
> Managed Service for Apache Flink 콘솔에서 대시보드를 열면 콘솔이 생성하는 URL이 12시간 동안 유효합니다.

Managed Service for Apache Flink CLI를 사용하여 애플리케이션의 Apache Flink 대시보드에 액세스

Managed Service for Apache Flink CLI를 사용하여 애플리케이션 대시보드에 액세스하는 URL을 생성할 수 있습니다. 생성한 URL은 지정된 시간 동안 유효합니다.

> **Note**
>
> 생성된 URL에 3분 이내에 액세스하지 않으면 더 이상 유효하지 않습니다.

CreateApplicationPresignedUrl 작업을 사용하여 대시보드 URL을 생성합니다. 작업에 대해 다음 파라미터를 지정할 수 있습니다:

- 애플리케이션 명칭
- URL이 유효한 시간(단위: 초)
- `FLINK_DASHBOARD_URL`을 URL 유형으로 지정합니다.

릴리스 버전

이 항목에는 Managed Service for Apache Flink의 각 릴리스에서 지원되는 기능 및 권장되는 구성 요소 버전에 대한 정보가 포함되어 있습니다.

Amazon Managed Service for Apache Flink 1.15.2 릴리스

아파치 플링크용 관리형 서비스는 Apache 1.15.2의 다음과 같은 새로운 기능을 지원합니다.

특징	설명	Apache Flip 참조
비동기식 싱크	AWS은(는) 개발자가 이전 노력의 절반도 안 되는 노력으로 사용자 지정 AWS 커넥터를 구축할 수 있도록 하는 비동기 대상 구축을 위한 기본 프레임워크입니다. 자세한 내용은 일반 비동기식 베이스 싱크를 참조하세요.	FLIP-171: 비동기 싱크.
Kinesis Data Firehose Sink	AWS은(는) 비동기 프레임워크를 사용하여 새로운 Amazon Kinesis Firehose Sink를 제공했습니다.	Amazon Kinesis Data Firehose Sink
세이브 포인트로 중지	세이브 포인트로 중지는 클린 스탑 오퍼레이션을 보장하며, 가장 중요한 것은 이를 신뢰하는 고객을 위해 정확히 한 번의 시맨틱을 지원한다는 것입니다.	FLIP-34: 세이브 포인트를 사용하여 작업을 종료/일시 중단합니다.
Scala 디커플링	이제 사용자는 Scala 3을 포함한 모든 Scala 버전에서 Java API를 활용할 수 있습니다. 고객은 Scala 애플리케이션에 원	FLIP-28: 스칼라 없는 플링크 테이블을 만드는 장기 목표.

특징	설명	Apache Flip 참조
	하는 Scala 표준 라이브러리를 번들로 제공해야 합니다.	
Scala	위의 Scala 디커플링을 참조하세요.	FLIP-28: 스칼라 없는 플링크 테이블을 만드는 장기 목표.
통합 커넥터 지표	Flink는 작업, 작업 및 연산자에 대한 표준 지표를 정의했습니다. Managed Service for Apache Flink는 싱크 및 소스 지표를 계속 지원할 예정이며, 1.15에서는 `numRestarts` 을(를) 가용성 지표`fullRestarts` 와 병행하여 도입될 예정입니다.	FLIP-33: 커넥터 지표의 표준화 및 FLIP-179: 표준화된 운영자 지표 공개.
완료된 작업 체크포인트	이 기능은 Flink 1.15에서 기본적으로 활성화되어 있으며, 작업 그래프의 일부가 모든 데이터 처리를 완료한 경우에도 체크포인트를 계속 수행할 수 있습니다. 이는 제한된 (일괄 처리) 소스가 포함된 경우 발생할 수 있습니다.	FLIP-147: 작업 완료 후 체크포인트를 지원합니다.

Apache Flink 1.15를 사용하는 Amazon Managed Service for Apache Flink의 변경 사항

Studio 노트북

이제 Managed Service for Apache Flink Studio가 Apache Flink 1.15를 지원합니다. Managed Service for Apache Flink Studio는 Apache Zeppelin 노트북을 활용하여 Apache Flink 스트림 처리 애플리케이션을 개발, 디버깅 및 실행하기 위한 단일 인터페이스 개발 환경을 제공합니다. Managed Service for Apache Flink Studio 및 시작 방법에 대한 자세한 내용은 Managed Service for Apache Flink와 함께 Studio 노트북 사용에서 확인할 수 있습니다.

EFO 커넥터

Managed Service for Apache Flink 버전 1.15로 업그레이드할 때는 최신 EFO 커넥터(모든 버전 1.15.3 이상)를 사용하고 있는지 확인하세요. 이유에 대한 자세한 내용은 FLINK-29324 항목을 참조하세요.

Scala 디커플링

Flink 1.15.2부터 Scala 애플리케이션에 원하는 Scala 표준 라이브러리를 번들로 제공해야 합니다.

Kinesis Data Firehose Sink

Managed Service for Apache Flink 버전 1.15로 업그레이드할 때는 최신 Amazon Kinesis Data Firehose Sink를 사용하고 있는지 확인하세요.

Kafka 커넥터

Apache Flink 버전 1.15용 Amazon Managed Service for Apache Flink로 업그레이드할 때 최신 Kafka 커넥터 API를 사용하고 있는지 확인하세요. Apache Flink는 FlinkKafkaConsumer 및 FlinkKafkaProducer를 더 이상 사용하지 않습니다. Kafka 싱크용 이러한 API는 Flink 1.15용 Kafka를 커밋할 수 없습니다. KafkaSource와 KafkaSink를 사용하고 있는지 확인하세요.

구성 요소

구성 요소	버전
Java	11 (권장)
Scala	2.12
Managed Service for Apache Flink Flink Runtime (aws-kinesisanalytics-runtime)	1.2.0
AWS Kinesis Connector (플링크-커넥터-키네시스)	1.15.4
Apache Beam (빔 애플리케이션만 해당)	2.33.0(Jackson 버전 2.12.2 포함)

Managed Service for Apache Flink와 함께 Studio 노트북 사용

Managed Service for Apache Flink용 Studio 노트북을 사용하면 실시간으로 데이터 스트림을 대화식으로 쿼리하고 표준 SQL, Python 및 Scala를 사용하여 스트림 처리 애플리케이션을 쉽게 빌드하고 실행할 수 있습니다. AWS 관리 콘솔에서 클릭 몇 번으로 서버리스 노트북을 실행하여 데이터 스트림을 쿼리하고 몇 초 만에 결과를 얻을 수 있습니다.

노트북은 웹 기반 개발 환경입니다. 노트북을 사용하면 Apache Flink에서 제공하는 고급 기능과 결합된 간단한 대화형 개발 환경을 얻을 수 있습니다. Studio 노트북은 Apache Zeppelin 기반 노트북을 사용하며 Apache Flink를 스트림 처리 엔진으로 사용합니다. Studio 노트북은 이러한 기술을 완벽하게 결합하여 모든 기술을 갖춘 개발자가 데이터 스트림에 대한 고급 분석을 이용할 수 있도록 합니다.

Apache Zeppelin은 Studio 노트북에 다음을 포함한 완벽한 분석 도구 제품군을 제공합니다.

- 데이터 시각화
- 데이터를 파일로 내보내기
- 보다 쉬운 분석을 위한 출력 형식 제어

Apache Flink 및 Apache Zeppelin용 관리 서비스 사용을 시작하려면 Studio 노트북 자습서 만들기 을 참조하세요. Apache Zeppelin 에 대한 자세한 내용은 Apache Zeppelin 문서를 참조하세요.

노트북을 사용하면 SQL, Python 또는 Scala의 Apache Flink Table API 및 SQL 또는 Scala의 DataStream API를 사용하여 쿼리를 모델링할 수 있습니다. 그런 다음 몇 번의 클릭만으로 Studio 노트북을 프로덕션 워크로드에 맞게 지속적으로 실행되는 비대화형 Managed Service for Apache Flink 스트림 처리 애플리케이션으로 승격할 수 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Studio 노트북 생성
- 스트리밍 데이터의 대화형 분석
- 지속 가능한 상태의 애플리케이션으로 배포
- Studio 노트북의 IAM 권한
- 커넥터 및 종속성
- 사용자 정의 함수
- 체크포인트 활성화

Studio 노트북 생성

Studio 노트북에는 스트리밍 데이터에서 실행되고 분석 결과를 반환하는 SQL, Python 또는 Scala로 작성된 쿼리 또는 프로그램이 포함되어 있습니다. 콘솔이나 CLI를 사용하여 애플리케이션을 만들고 데이터 소스의 데이터를 분석하기 위한 쿼리를 제공합니다.

애플리케이션은 다음과 같은 구성 요소로 이루어집니다.

- Amazon MSK 클러스터, Kinesis 데이터 스트림 또는 Amazon S3 버킷과 같은 데이터 소스.
- AWS Glue 데이터베이스. 이 데이터베이스에는 데이터 소스, 대상 스키마 및 엔드포인트를 저장하는 테이블이 포함되어 있습니다. 자세한 설명은 AWS Glue 작업 섹션을 참조하세요.
- 애플리케이션 코드. 코드는 분석 쿼리 또는 프로그램을 구현합니다.
- 애플리케이션 설정 및 런타임 속성. 애플리케이션 설정과 런타임 속성에 대한 자세한 내용은 Apache Flink 애플리케이션 개발자 안내서에서 다음 항목을 참조하세요.
  - 애플리케이션 병렬성 및 크기 조정: 애플리케이션의 병렬성 설정을 사용하여 애플리케이션이 동시에 실행할 수 있는 쿼리 수를 제어할 수 있습니다. 또한 다음과 같은 상황에서 쿼리에 실행 경로가 여러 개 있는 경우 향상된 병렬 처리 기능을 활용할 수 있습니다.
    - Kinesis 데이터 스트림의 여러 샤드를 처리하는 경우
    - KeyBy 연산자를 사용하여 데이터를 분할하는 경우.
    - 여러 윈도우 연산자를 사용하는 경우

    애플리케이션 스케일링에 대한 자세한 내용은 Apache Flink용 Managed Service for Apache Flink의 애플리케이션 스케일링을 참조하세요.
  - 로깅 및 모니터링: 애플리케이션 로깅 및 모니터링에 대한 자세한 내용은 Apache Flink용 Amazon Managed Service for Apache Flink의 로깅 및 모니터링을 참조하세요.
  - 애플리케이션은 내결함성을 위해 체크포인트와 세이브포인트를 사용합니다. Studio 노트북에서는 체크포인트와 세이브포인트가 기본적으로 활성화되지 않습니다.

AWS Management Console 또는 AWS CLI를 사용하여 Studio 노트북을 만들 수 있습니다.

콘솔에서 애플리케이션을 생성할 때 사용할 수 있는 옵션은 다음과 같습니다.

- Amazon MSK 콘솔에서 클러스터를 선택한 다음 실시간 데이터 처리를 선택합니다.
- Kinesis Data Streams 콘솔에서 데이터 스트림을 선택한 다음 애플리케이션 탭에서 실시간 데이터 처리를 선택합니다.
- Apache Flink용 관리형 서비스 콘솔에서 Studio 탭을 선택한 다음 Studio 노트북 생성을 선택합니다.

자습서는 Event Detection with Managed Service for Apache Flink를 참조하세요.

고급 Studio 노트북 솔루션의 예는 Apache Flink Studio용 Amazon Managed Service for Apache Flink의 Apache Flink를 참조하세요.

스트리밍 데이터의 대화형 분석

Apache Zeppelin으로 구동되는 서버리스 노트북을 사용하여 스트리밍 데이터와 상호 작용합니다. 노트북에는 여러 개의 노트가 있을 수 있으며, 각 노트에는 코드를 작성할 수 있는 하나 이상의 단락이 있을 수 있습니다.

다음 예제 SQL 쿼리는 데이터 소스에서 데이터를 검색하는 방법을 보여줍니다.

```
%flink.ssql(type=update)
select * from stock;
```

Flink Streaming SQL 쿼리의 더 많은 예는 다음 예시 및 자습서 및 Apache Flink 설명서의 쿼리를 참조하세요.

Studio 노트북에서 Flink SQL 쿼리를 사용하여 스트리밍 데이터를 쿼리할 수 있습니다. Python (Table API) 및 Scala (Table and Datastream APIs) 를 사용하여 스트리밍 데이터를 대화형 방식으로 쿼리하는 프로그램을 작성할 수도 있습니다. 쿼리 또는 프로그램의 결과를 보고, 몇 초 만에 업데이트한 다음, 다시 실행하여 업데이트된 결과를 볼 수 있습니다.

플링크 인터프리터

인터프리터를 사용하여 Managed Service for Apache Flink에서 애플리케이션을 실행하는 데 사용할 언어를 지정합니다. Managed Service for Apache Flink와 함께 다음 인터프리터를 사용할 수 있습니다.

명칭	Class	설명
%flink	FlinkInterpreter	Creates ExecutionEnvironment/StreamExecutionEnvironment/BatchTableEnvironment/StreamTableEnvironment and provides a Scala environment
%flink.pyflink	PyFlinkInterpreter	Provides a python environment
%flink.ipyflink	IPyFlinkInterpreter	Provides an ipython environment
%flink.ssql	FlinkStreamSqlInterpreter	Provides a stream sql environment
%flink.bsql	FlinkBatchSqlInterpreter	Provides a batch sql environment

Flink 인터프리터에 대한 자세한 내용은 Apache Zeppelin용 Flink 인터프리터를 참조하세요.

`%flink.pyflink` 또는 `%flink.ipyflink`를 인터프리터로 사용하는 경우 노트북에서 결과를 시각화하려면 `ZeppelinContext`를 사용해야 합니다.

보다 PyFlink 구체적인 예는 Apache Flink Studio 및 Python용 관리형 서비스를 사용하여 대화형 방식으로 데이터 스트림을 쿼리하는 방법을 참조하십시오.

Apache Flink 테이블 환경 변수

Apache Zeppelin은 환경 변수를 사용하여 테이블 환경 리소스에 액세스할 수 있습니다.

다음 변수를 사용하여 Scala 테이블 환경 리소스에 액세스할 수 있습니다.

변수	Resource
`senv`	`StreamExecutionEnvironment`
`stenv`	`StreamTableEnvironment ### ####`

다음 변수를 사용하여 Python 테이블 환경 리소스에 액세스합니다.

변수	Resource
`s_env`	`StreamExecutionEnvironment`
`st_env`	`StreamTableEnvironment ### ####`

테이블 환경 사용에 대한 자세한 내용은 Apache Flink 설명서의 Create TableEnvironment a를 참조하십시오.

지속 가능한 상태의 애플리케이션으로 배포

코드를 빌드하고 Amazon S3로 내보낼 수 있습니다. 노트에 작성한 코드를 지속적으로 실행되는 스트림 처리 애플리케이션으로 승격할 수 있습니다. Managed Service for Apache Flink에서 Apache Flink 애플리케이션을 실행하는 두 가지 모드가 있습니다. Studio 노트북을 사용하면 코드를 대화형 방식으로 개발하고, 코드 결과를 실시간으로 보고, 노트 내에서 시각화할 수 있습니다. 스트리밍 모드에서 실행할 노트를 배포하면 Managed Service for Apache Flink가 지속적으로 실행되는 애플리케이션을 생성하고, 소스에서 데이터를 읽고, 대상에 쓰고, 장기 실행 애플리케이션 상태를 유지하고, 소스 스트림의 처리량에 따라 자동으로 스케일링하는 애플리케이션을 생성합니다.

> **Note**
>
> 애플리케이션 코드를 내보내는 S3 버킷은 Studio 노트북과 동일한 리전에 있어야 합니다.

다음 기준을 충족하는 경우에만 Studio 노트북의 노트를 배포할 수 있습니다.

- 단락은 순차적으로 정렬해야 합니다. 애플리케이션을 배포하면 노트 내의 모든 단락이 노트에 표시된 대로 순차적으로 (left-to-right, top-to-bottom) 실행됩니다. 노트에서 모든 단락 실행을 선택하여 이 순서를 확인할 수 있습니다.
- 코드는 Python과 SQL 또는 Scala와 SQL의 조합입니다. 현재는 Python과 Scala를 함께 지원하지 않습니다. deploy-as-application
- 노트에는`%flink`, `%flink.ssql`, `%flink.pyflink`, `%flink.ipyflink`, `%md`와(과) 같은 인터프리터만 있어야 합니다.

- Zeppelin 컨텍스트 객체 z의 사용은 지원되지 않습니다. 아무것도 반환하지 않는 메서드는 경고를 기록하는 것 외에는 아무것도 수행하지 않습니다. 다른 메서드는 Python 예외를 발생시키거나 Scala에서 컴파일하지 못합니다.
- 노트 하나로 Apache Flink 작업이 한 번 발생해야 합니다.
- 동적 양식이 포함된 노트는 애플리케이션으로 배포할 수 없습니다.
- %md (Markdown) 단락은 애플리케이션의 일부로 실행하기에 부적합한 사람이 읽을 수 있는 문서를 포함할 것으로 예상되어 애플리케이션으로 배포할 때는 생략됩니다.
- Zeppelin 내에서 실행되지 않는 문단은 애플리케이션으로 배포할 때 생략됩니다. 비활성화된 단락이 호환되지 않는 인터프리터를 사용하는 경우, 예를 들어 `%flink and %flink.ssql` 인터프리터가 있는 노트 `%flink.ipyflink`을(를) 사용하더라도 노트를 애플리케이션으로 배포하는 동안에는 해당 단락을 건너뛰고 오류가 발생하지 않습니다.
- 응용 프로그램 배포가 성공하려면 실행할 수 있는 소스 코드 (Flink SQL PyFlink 또는 Flink Scala) 가 포함된 단락이 하나 이상 있어야 합니다.
- 단락 내에서 인터프리터 지시문에 병렬성을 설정하는 경우 (예: `%flink.ssql(parallelism=32)`) 노트에서 배포된 애플리케이션에서는 무시됩니다. 대신AWS Management Console, AWS Command Line Interface 또는 AWS API를 통해 배포된 응용 프로그램을 업데이트하여 응용 프로그램에 필요한 병렬성 수준에 따라 병렬성 및/또는 ParallelismPer KPU 설정을 변경하거나 배포된 응용 프로그램의 자동 크기 조정을 활성화할 수 있습니다.
- 지속 가능한 상태의 애플리케이션으로 배포하는 경우 VPC는 인터넷에 액세스할 수 있어야 합니다. VPC가 인터넷에 액세스할 수 없는 경우 인터넷에 접속할 수 없는 VPC에서 지속 가능한 상태의 애플리케이션으로 배포을(를) 참조하세요.

Scala/Python 기준

- Scala 또는 Python 코드에서는 이전의 “Flink” 플래너(Scala용 `senv`, `stenv`; Python용 `s_env`, `st_env`)가 아닌 Blink 플래너 (Scala의 경우 `stenv_2`, Python의 경우 `st_env_2`)를 사용하세요. Apache Flink 프로젝트에서는 프로덕션 사용 사례에 Blink 플래너를 사용할 것을 권장하는데, 이 플래너는 Zeppelin과 Flink의 기본 플래너입니다.
- Python 단락은 애플리케이션으로 배포하기 위한 노트와 같이 ! 또는 `%timeit`와(과) `%conda`같은 IPython 매직 명령을 사용하여 셸 호출/할당을 사용해서는 안 됩니다.
- `map` 및 `filter` 같은 고차 데이터 흐름 연산자에 전달되는 함수의 파라미터로는 Scala 케이스 클래스를 사용할 수 없습니다. Scala 케이스 클래스에 대한 자세한 내용은 Scala 설명서의 케이스 클래스를 참조하세요

SQL 기준

- 데이터를 전달할 수 있는 단락의 출력 섹션과 같은 곳이 없기 때문에 단순 SELECT 문은 허용되지 않습니다.
- 어떤 단락에서든 DDL 문 (USE, CREATE, ALTER, DROP,SET, RESET) 은 DML(INSERT) 문 앞에 와야 합니다. 이는 한 단락의 DML 명령문을 단일 Flink 작업으로 함께 제출해야 하기 때문입니다.
- DML 명령문이 포함된 단락은 최대 하나여야 합니다. 이 deploy-as-application 기능의 경우 Flink에 단일 작업을 제출하는 것만 지원하기 때문입니다.

자세한 내용과 예제는 Amazon Managed Service for Apache Flink, Amazon Translate 및 Amazon Comprehend에서 SQL 함수를 사용하여 스트리밍 데이터를 번역, 편집 및 분석하기를 참조하세요.

Studio 노트북의 IAM 권한

Apache Flink용 관리형 서비스는 AWS Management Console을(를) 통해 Studio 노트북을 생성할 때 IAM 역할을 생성합니다. 또한 다음 액세스를 허용하는 정책을 해당 역할과 연결합니다.

Service	액세스
CloudWatch 로그	나열
Amazon EC2	나열
AWS Glue	읽기, 쓰기
Managed Service for Apache Flink	읽기
Managed Service for Apache Flink V2	읽기
Amazon S3	읽기, 쓰기

커넥터 및 종속성

커넥터를 사용하면 다양한 기술 전반에서 데이터를 읽고 쓸 수 있습니다. Managed Service for Apache Flink는 Studio 노트북과 함께 세 개의 기본 커넥터를 번들로 제공합니다. 사용자 지정 커넥터를 작성할 수도 있습니다. 커넥터에 대한 자세한 내용은 Apache Flink 설명서의 테이블 및 SQL 커넥터를 참조하세요.

기본 커넥터

AWS Management Console을(를) 사용하여 Studio 노트북을 만드는 경우 Managed Service for Apache Flink에는 기본적으로 `flink-sql-connector-flink`, `flink-connector-kafka_2.12` 및 `aws-msk-iam-auth` 같은 사용자 지정 커넥터가 포함됩니다. 이러한 사용자 지정 커넥터 없이 콘솔을 통해 Studio 노트북을 만들려면 사용자 지정 설정으로 만들기 옵션을 선택합니다. 그런 다음 구성 페이지로 이동하면 두 커넥터 옆에 있는 확인란의 선택을 취소하세요.

CreateApplicationAPI를 사용하여 Studio 노트북을 만드는 경우, `flink-sql-connector-flink` 및 `flink-connector-kafka` 커넥터는 기본적으로 포함되지 않습니다. 추가하려면 다음 예와 같이 `CustomArtifactsConfiguration` 데이터 유형에 `MavenRefernce`(으)로 지정하세요.

`aws-msk-iam-auth` 커넥터는 Amazon MSK에서 사용할 커넥터로, IAM으로 자동 인증하는 기능이 포함되어 있습니다.

> **Note**
>
> 다음 예제에 표시된 커넥터 버전은 지원되는 유일한 버전입니다.

```
For the Kinesis connector:

"CustomArtifactsConfiguration": [{
"ArtifactType": "DEPENDENCY_JAR",
   "MavenReference": {
"GroupId": "org.apache.flink",

      "ArtifactId": "flink-sql-connector-kinesis",
      "Version": "1.15.4"

   }
}]

For authenticating with AWS MSK through AWS IAM:

"CustomArtifactsConfiguration": [{
"ArtifactType": "DEPENDENCY_JAR",
   "MavenReference": {
"GroupId": "software.amazon.msk",
      "ArtifactId": "aws-msk-iam-auth",
      "Version": "1.1.6"
```

```
   }
}]

For the Apache Kafka connector:

"CustomArtifactsConfiguration": [{
"ArtifactType": "DEPENDENCY_JAR",
   "MavenReference": {
"GroupId": "org.apache.flink",

      "ArtifactId": "flink-connector-kafka",
      "Version": "1.15.4"

   }
}]
```

이러한 커넥터를 기존 노트북에 추가하려면 UpdateApplicationAPI 작업을 사용하고 `CustomArtifactsConfigurationUpdate` 데이터 유형에 `MavenReference` a로 지정하십시오.

> **Note**
>
> 테이블 API의 `flink-sql-connector-kinesis` 커넥터에 대해 `failOnError`을(를) true로 설정할 수 있습니다.

종속성 및 사용자 정의 커넥터

AWS Management Console을(를) 사용하여 Studio 노트북에 종속성 또는 사용자 지정 커넥터를 추가하려면 다음 단계를 따르세요.

1. 사용자 정의 커넥터 파일을 Amazon S3에 업로드합니다.
2. AWS Management Console에서 Studio 노트북을 생성하기 위한 사용자 지정 생성 옵션을 선택합니다.
3. 구성 단계에 도달할 때까지 Studio 노트북 제작 워크플로를 따르세요.
4. 사용자 지정 커넥터 섹션에서 사용자 지정 커넥터 추가를 선택합니다.
5. 종속성 또는 사용자 지정 커넥터의 Amazon S3 위치를 지정합니다.
6. 변경 사항 저장을 선택합니다.

CreateApplicationAPI를 사용하여 새 Studio 노트북을 생성할 때 종속성 JAR 또는 사용자 지정 커넥터를 추가하려면 종속성 JAR의 Amazon S3 위치 또는 `CustomArtifactsConfiguration` 데이터 유형에서 사용자 지정 커넥터를 지정하십시오. 기존 Studio 노트북에 종속성 또는 사용자 지정 커넥터를 추가하려면 UpdateApplicationAPI 작업을 호출하고 종속성 JAR의 Amazon S3 위치 또는 데이터 유형에서 사용자 지정 커넥터를 지정합니다. `CustomArtifactsConfigurationUpdate`

> **Note**
>
> 종속성이나 사용자 지정 커넥터를 포함할 때는 종속성 또는 사용자 지정 커넥터에 번들로 제공되지 않은 모든 전이적 종속성도 포함해야 합니다.

사용자 정의 함수

사용자 정의 함수 (UDF) 는 자주 사용되는 로직이나 쿼리에서 달리 표현할 수 없는 사용자 지정 로직을 호출할 수 있는 확장점입니다. Python 또는 Java 또는 Scala와 같은 JVM 언어를 사용하여 Studio 노트북 내에서 단락으로 UDF를 구현할 수 있습니다. JVM 언어로 구현된 UDF가 포함된 외부 JAR 파일을 Studio 노트북에 추가할 수도 있습니다.

서브클래스 `UserDefinedFunction` (또는 자체 추상 클래스)가 있는 추상 클래스를 등록하는 JAR을 구현할 때는 Apache Maven에서 제공된 범위, Gradle에서 `compileOnly` 종속성 선언, SBT에서 제공된 범위 또는 UDF 프로젝트 빌드 구성에서 동등한 디렉티브를 사용하세요. 이렇게 하면 UDF 소스 코드를 Flink API에 대해 컴파일할 수 있지만 Flink API 클래스 자체는 빌드 아티팩트에 포함되지 않습니다. Maven 프로젝트의 이러한 전제 조건을 준수하는 UDF jar 예제의 다음 pom을 참조하세요.

> **Note**
>
> 예제 설정은 기계 학습 블로그 에서 AWS 기계 학습 블로그에서 Amazon Managed Service for Apache Flink, Amazon Translate, 그리고 Amazon Comprehend를 사용하여 SQL 기능으로 스트리밍 데이터를 번역, 수정 및 분석을 참조하세요.

콘솔을 사용하여 UDF JAR 파일을 Studio 노트북에 추가하려면 다음 단계를 따르세요.

1. Amazon S3에 UDF JAR 파일을 업로드합니다.
2. AWS Management Console에서 Studio 노트북을 생성하기 위한 사용자 지정 생성 옵션을 선택합니다.
3. 구성 단계에 도달할 때까지 Studio 노트북 제작 워크플로를 따르세요.

4. 사용자 정의 함수 섹션에서 사용자 정의 함수 추가를 선택합니다.
5. UDF를 구현한 JAR 파일 또는 ZIP 파일의 Amazon S3 위치를 지정하세요.
6. 변경 사항 저장을 선택합니다.

CreateApplicationAPI를 사용하여 새 스튜디오 노트북을 생성할 때 UDF JAR을 추가하려면 데이터 유형에 JAR 위치를 지정하십시오. `CustomArtifactConfiguration` 기존 스튜디오 노트북에 UDF JAR을 추가하려면 UpdateApplicationAPI 작업을 호출하고 데이터 유형에 JAR 위치를 지정합니다. `CustomArtifactsConfigurationUpdate` 또는 AWS Management Console을(를) 사용하여 Studio 노트북에 UDF JAR 파일을 추가할 수 있습니다.

사용자 정의 함수 관련 고려 사항

- Managed Service for Apache Flink Studio는 Apache Zeppelin 용어를 사용합니다. 여기서 노트북은 여러 노트를 포함할 수 있는 Zeppelin 인스턴스입니다. 그러면 각 노트에는 여러 단락이 포함될 수 있습니다. Managed Service for Apache Flink Studio를 사용하면 인터프리터 프로세스가 노트북의 모든 노트에서 공유됩니다. 따라서 한 노트에서 Function을 사용하여 명시적 createTemporarySystem 함수 등록을 수행하면 동일한 노트북의 다른 노트에서도 동일한 함수를 있는 그대로 참조할 수 있습니다.

  하지만 애플리케이션으로 배포 작업은 개별 노트에서만 작동하며 노트북의 모든 노트에 적용할 수는 없습니다. 애플리케이션으로 배포를 수행하는 경우 활성 노트의 내용만 애플리케이션을 생성하는 데 사용됩니다. 다른 노트북에서 수행된 명시적 함수 등록은 생성된 애플리케이션 종속성에 포함되지 않습니다. 또한 애플리케이션으로 배포 옵션을 사용하는 경우 JAR의 기본 클래스 이름을 소문자 문자열로 변환하여 암시적 함수를 등록할 수 있습니다.

  예를 들어 `TextAnalyticsUDF`이(가) UDF JAR의 기본 클래스인 경우 암시적 등록으로 인해 함수 이름은 `textanalyticsudf`(으)로 생성됩니다. 따라서 스튜디오의 노트 1에서 다음과 같은 명시적 함수 등록이 발생하면 공유 인터프리터로 인해 해당 노트북의 다른 모든 노트(예: 노트 2)가 함수를 이름을 `myNewFuncNameForClass`을(를) 참조할 수 있습니다.

  ```
  stenv.createTemporarySystemFunction("myNewFuncNameForClass", new
  TextAnalyticsUDF())
  ```

  하지만 노트 2에서 애플리케이션으로 배포하기 작업 중에는 이러한 명시적 등록이 종속성에 포함되지 않으므로 배포된 애플리케이션이 예상대로 작동하지 않습니다. 암시적 등록으로 인해 기본적으로 이 함수에 대한 모든 참조는 `myNewFuncNameForClass`이(가) 아닌 `textanalyticsudf`와(과) 함께 있어야 합니다.

사용자 정의 함수 이름 등록이 필요한 경우 노트 2 자체에는 다음과 같이 명시적 등록을 다시 수행하는 다른 단락이 포함될 것으로 예상됩니다.

```
%flink(parallelism=1)
import com.amazonaws.kinesis.udf.textanalytics.TextAnalyticsUDF
# re-register the JAR for UDF with custom name
stenv.createTemporarySystemFunction("myNewFuncNameForClass", new TextAnalyticsUDF())
```

```
%flink. ssql(type=update, parallelism=1)
INSERT INTO
    table2
SELECT
    myNewFuncNameForClass(column_name)
FROM
    table1
;
```

- UDF JAR에 Flink SDK가 포함된 경우 UDF 소스 코드가 Flink SDK에 대해 컴파일될 수 있지만 Flink SDK 클래스 자체는 빌드 아티팩트에 포함되지 않도록 JAR과 같이 자바 프로젝트를 구성하세요.

  Apache Maven에서는 `provided` 범위를, Gradle에서는 `compileOnly` 종속성 선언을, SBT에서는 `provided` 범위를, UDF 프로젝트 빌드 구성에서는 이에 상응하는 디렉티브를 사용할 수 있습니다. 메이븐 프로젝트의 이러한 전제 조건을 준수하는 UDF jar 예제에서 이 pom을 참조할 수 있습니다. 전체 step-by-step 자습서를 보려면 이 Apache Flink, Amazon Translate 및 Amazon Comprehend용 Amazon Managed Service의 SQL 함수를 사용하여 스트리밍 데이터를 번역, 편집 및 분석하십시오.

체크포인트 활성화

환경 설정을 사용하여 체크포인트를 활성화할 수 있습니다. 체크포인트에 대한 자세한 내용은 Managed Service for Apache Flink Developer Guide의 Fault Tolerance를 참조하세요.

체크포인트 간격 설정

다음 Scala 코드 예제는 애플리케이션의 체크포인트 간격을 1분으로 설정합니다.

```
// start a checkpoint every 1 minute
stenv.enableCheckpointing(60000)
```

다음 Python 코드 예제는 애플리케이션의 체크포인트 간격을 1분으로 설정합니다.

```
st_env.get_config().get_configuration().set_string(
    "execution.checkpointing.interval", "1min"
)
```

체크포인트 유형 설정

다음 Scala 코드 예제는 애플리케이션의 체크포인트 모드를 EXACTLY_ONCE(기본값)(으)로 설정합니다.

```
// set mode to exactly-once (this is the default)
stenv.getCheckpointConfig.setCheckpointingMode(CheckpointingMode.EXACTLY_ONCE)
```

다음 Python 코드 예제는 애플리케이션의 체크포인트 모드를 EXACTLY_ONCE(기본값)으(로) 설정합니다.

```
st_env.get_config().get_configuration().set_string(
    "execution.checkpointing.mode", "EXACTLY_ONCE"
)
```

AWS Glue 작업

Studio 노트북은 AWS Glue에서 해당 데이터 소스 및 싱크에 대한 정보를 저장하고 가져옵니다. Studio 노트북을 만들 때는 연결 정보가 포함된 AWS Glue 데이터베이스를 지정합니다. 데이터 소스와 싱크에 접근할 때는 데이터베이스에 포함된 AWS Glue 테이블을 지정합니다. AWS Glue 테이블을 통해 데이터 원본 및 대상의 위치, 스키마, 파라미터를 정의하는 AWS Glue 연결에 액세스할 수 있습니다.

Studio 노트북은 테이블 속성을 사용하여 애플리케이션별 데이터를 저장합니다. 자세한 설명은 테이블 속성 섹션을 참조하세요.

Studio 노트북에서 사용할 AWS Glue 연결, 데이터베이스 및 테이블을 설정하는 방법의 예는 Studio 노트북 자습서 만들기 자습서의 AWS Glue 데이터베이스 생성을(를) 참조하세요.

테이블 속성

AWS Glue 테이블은 데이터 필드 외에도 테이블 속성을 사용하여 Studio 노트북에 기타 정보를 제공합니다. Managed Service for Apache Flink는 다음 AWS Glue 테이블 속성을 사용합니다.

- Apache Flink 시간 값 사용: 이러한 속성은 Managed Service for Apache Flink가 Apache Flink의 내부 데이터 처리 시간 값을 내보내는 방법을 정의합니다.

- Flink Connector 및 포맷 속성 사용: 이러한 속성은 데이터 스트림에 대한 정보를 제공합니다.

AWS Glue 테이블에 속성을 추가하려면 다음을 수행합니다.

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 테이블 목록에서 애플리케이션이 데이터 연결 정보를 저장하는 데 사용하는 테이블을 선택합니다. 작업, 테이블 세부 정보 편집을 선택합니다.
3. 테이블 속성에서 **managed-flink.proctime** 키와 **user_action_time** 값을 입력합니다.

Apache Flink 시간 값 사용

Apache Flink는 처리 시간 및 이벤트 시간과 같은 스트림 처리 이벤트가 발생한 시간 값을 제공합니다. 이러한 값을 애플리케이션 출력에 포함하려면 Managed Service for Apache Flink 런타임에 이러한 값을 지정된 필드로 내보내도록 지시하는 속성을 AWS Glue 테이블에 정의합니다.

테이블 속성에서 사용하는 키와 값은 다음과 같습니다.

타임스탬프 유형	키	값
처리 시간	managed-flink.proctime	The column name that AWS Glue will use to expose the value. This column name does not correspond to an existing table column.
이벤트 시간	managed-flink.rowtime	The column name that AWS Glue will use to expose the value. This column name corresponds to an existing table column.
	managed-flink.watermark.*column_name*.milliseconds	The watermark interval in milliseconds

Flink Connector 및 포맷 속성 사용

AWS Glue 테이블 속성을 사용하여 애플리케이션의 Flink 커넥터에 데이터 소스에 대한 정보를 제공합니다. Managed Service for Apache Flink에서 커넥터에 사용하는 속성의 몇 가지 예는 다음과 같습니다.

커넥터 유형	키	값
Kafka	##	The format used to deserialize and serialize Kafka messages, e.g. `json` or `csv`.
	`scan.startup.mode`	The startup mode for the Kafka consumer, e.g. `earliest-offset` or `#####`.
Kinesis	##	The format used to deserialize and serialize Kinesis data stream records, e.g. `json` or `csv`.
	`aws.region`	The AWS region where the stream is defined.
S3 (Filesystem)	format	The format used to deserialize and serialize files, e.g. `json` or `csv`.
	##	The Amazon S3 path, e.g. `s3://mybucket/` .

Kinesis 및 Apache Kafka 이외의 다른 커넥터에 대한 자세한 내용은 커넥터의 설명서를 참조하세요.

예시 및 자습서

주제

- 자습서: Managed Service for Apache Flink에서 Studio 노트북 생성

자습서: Managed Service for Apache Flink에서 Studio 노트북 생성

다음 자습서는 Kinesis Data Stream 또는 Amazon MSK 클러스터에서 데이터를 읽는 Studio 노트북을 생성하는 방법을 보여줍니다.

이 자습서는 다음 섹션을 포함하고 있습니다:

설정

AWS CLI가 버전 2 이상인지 확인합니다. 최신 AWS CLI를 설치하려면 AWS CLI 버전 2 설치, 업데이트 및 제거를 참조하세요.

AWS Glue 데이터베이스 생성

Studio 노트북은 Amazon MSK 데이터 소스에 대한 메타데이터용 AWS Glue 데이터베이스를 사용합니다.

AWS Glue 데이터베이스 생성

1. https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 데이터베이스 추가(Add database)를 선택합니다. 데이터베이스 추가 창에서 데이터베이스 이름을 **`default`**(으)로 입력합니다. 생성을 선택합니다.

다음 단계

이 자습서에서는 Kinesis Data Streams 또는 Amazon MSK를 사용하는 Studio 노트북을 만들 수 있습니다.

- Kinesis Data Streams: Kinesis Data Streams 를 사용하면 Kinesis 데이터 스트림을 소스로 사용하는 애플리케이션을 빠르게 생성할 수 있습니다. Kinesis 데이터 스트림을 종속 리소스로 생성하기만 하면 됩니다.
- Amazon MSK: Amazon MSK를 사용하면 Amazon MSK 클러스터를 소스로 사용하는 애플리케이션을 생성할 수 있습니다. Amazon VPC, Amazon EC2 클라이언트 인스턴스, Amazon MSK 클러스터를 종속 리소스로 생성해야 합니다.

Kinesis Data Streams로 Studio 노트북 생성

이 자습서에서는 Kinesis 데이터 스트림을 소스로 사용하는 Studio 노트북을 생성하는 방법을 설명합니다.

이 자습서는 다음 섹션을 포함하고 있습니다:

- 설치
- AWS Glue 테이블 생성
- Kinesis Data Streams로 Studio 노트북 생성
- Kinesis 데이터 스트림으로 데이터 전송
- Studio 노트북을 테스트

설치

Studio 노트북을 생성하기 전에 Kinesis 데이터 스트림(`ExampleInputStream`)을 생성하세요. 애플리케이션은 이 스트림을 애플리케이션 소스로 사용합니다.

Amazon Kinesis 콘솔 또는 다음 AWS CLI 명령을 사용하여 이러한 스트림을 만들 수 있습니다. 콘솔 지침은 Amazon Kinesis Data Streams 개발자 가이드의 데이터 스트림 생성 및 업데이트를 참조하십시오. 스트림의 이름을 **`ExampleInputStream`**(으)로 지정하고 열린 샤드 수를 **`1`**(으)로 설정합니다.

AWS CLI를 사용하여 스트림(`ExampleInputStream`)을 생성하려면 다음 Amazon Kinesis `create-stream` AWS CLI 명령을 사용하세요.

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-east-1 \
--profile adminuser
```

AWS Glue 테이블 생성

Studio 노트북은 Kinesis Data Streams 데이터 소스에 대한 메타데이터용 AWS Glue 데이터베이스를 사용합니다.

> **Note**
>
> 먼저 데이터베이스를 수동으로 생성하거나 노트북을 만들 때 Managed Service for Apache Flink가 자동으로 데이터베이스를 생성하도록 할 수 있습니다. 마찬가지로 이 섹션에 설명된 대로 테이블을 수동으로 생성하거나 Apache Zeppelin 내에서 노트북의 Managed Service for Apache Flink용 테이블 커넥터 코드 생성을 사용하여 DDL 문을 통해 테이블을 생성할 수 있습니다. 그런 다음 AWS Glue을(를) 체크인하여 테이블이 올바르게 생성되었는지 확인할 수 있습니다.

표 생성

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 아직 AWS Glue 데이터베이스가 없는 경우 왼쪽 탐색 모음에서 데이터베이스를 선택합니다. 데이터베이스 추가를 선택합니다. 데이터베이스 추가 창에서 데이터베이스 이름을 **`default`**(으)로 입력합니다. 생성을 선택하세요.
3. 왼쪽 탐색 모음에서 표을 선택합니다. 표 페이지에서 표 추가, 표 수동 추가를 선택합니다.
4. 표 속성 설정 페이지에서 표 명칭에 **`stock`**을 입력합니다. 이전에 만든 데이터베이스를 선택했는지 확인하세요. 다음을 선택합니다.
5. 데이터 스토어 추가 페이지에서 Kinesis를 선택합니다. 스트림 이름에는 **`ExampleInputStream`**을(를) 입력합니다. Kinesis 소스 URL의 경우 **`https://kinesis.us-east-1.amazonaws.com`** 입력을 선택합니다. Kinesis 소스 URL을 복사하여 붙여넣는 경우 선행 또는 후행 공백을 모두 삭제해야 합니다. 다음을 선택합니다.
6. 분류 페이지에서 JSON을 선택합니다. 다음을 선택합니다.
7. 스키마 정의 페이지에서 열 추가를 선택하여 열을 추가합니다. 다음 속성을 가진 열을 추가합니다:

열 명칭	데이터 유형
`##`	**`###`**
`##`	**`double`**

다음을 선택합니다.

8. 다음 페이지에서 설정을 확인하고 마침을 선택합니다.
9. 테이블 목록에서 새로 생성한 테이블을 선택합니다.
10. 테이블 편집을 선택하고 `managed-flink.proctime` 키와 `proctime` 값이 있는 속성을 추가합니다.
11. 적용을 선택합니다.

Kinesis Data Streams로 Studio 노트북 생성

애플리케이션에서 사용하는 리소스를 생성했으니 이제 Studio 노트북을 생성합니다.

응용 프로그램을 만들려면 AWS Management Console 또는 AWS CLI를 사용할 수 있습니다.

- AWS Management Console을(를) 사용하여 Studio 노트북을 만드세요.
- AWS CLI을(를) 사용하여 Studio 노트북을 만드세요.

AWS Management Console을(를) 사용하여 Studio 노트북을 만드세요.

1. https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 애플리케이션 페이지에서 Studio 탭을 선택합니다. Studio 노트북 생성을 선택합니다.

   > **Note**
   >
   > 입력한 Amazon MSK 클러스터 또는 Kinesis 데이터 스트림을 선택하고 실시간 데이터 처리를 선택하여 Amazon MSK 또는 Kinesis Data Streams 콘솔에서 Studio 노트북을 생성할 수도 있습니다.

3. Studio 노트북 생성 페이지에서 다음 정보를 입력합니다:

   - 노트북 이름을 **`MyNotebook`**(으)로 입력합니다.
   - AWS Glue 데이터베이스의 기본값을 선택합니다.

   Studio 노트북 생성을 선택합니다.

4. MyNotebook페이지에서 실행을 선택합니다. 상태가 실행 중으로 표시될 때까지 기다리세요. 노트북이 실행 중일 때는 요금이 부과됩니다.

AWS CLI을(를) 사용하여 Studio 노트북을 만드세요.

AWS CLI를 사용하여 Studio 노트북을 만들려면 다음과 같이 하세요.

1. 계정 ID를 확인합니다. 애플리케이션을 생성하려면 이 값을 사용합니다.
2. `arn:aws:iam::AccountID:role/ZeppelinRole` 역할을 생성하고 콘솔에서 자동 생성된 역할에 다음 권한을 추가합니다.

   ```
   "kinesis:GetShardIterator",

   "kinesis:GetRecords",

   "kinesis:ListShards"
   ```

3. 다음 콘텐츠를 가진 `create.json`이라는 파일을 생성합니다: 자리 표시자 값을 정보로 바꿉니다.

   ```
   {
       "ApplicationName": "MyNotebook",
       "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
       "ApplicationMode": "INTERACTIVE",
       "ServiceExecutionRole": "arn:aws:iam::AccountID:role/ZeppelinRole",
       "ApplicationConfiguration": {
           "ApplicationSnapshotConfiguration": {
               "SnapshotsEnabled": false
           },
           "ZeppelinApplicationConfiguration": {
               "CatalogConfiguration": {
                   "GlueDataCatalogConfiguration": {
                       "DatabaseARN": "arn:aws:glue:us-east-1:AccountID:database/
   default"
                   }
               }
           }
       }
   }
   ```

4. 애플리케이션을 생성하려면 다음 명령을 실행합니다:

   ```
   aws kinesisanalyticsv2 create-application --cli-input-json file://create.json
   ```

5. 명령이 완료되면 새 Studio 노트북의 세부 정보를 보여주는 출력이 표시됩니다. 다음은 출력의 예제입니다.

```
{
    "ApplicationDetail": {
        "ApplicationARN": "arn:aws:kinesisanalyticsus-
east-1:012345678901:application/MyNotebook",
        "ApplicationName": "MyNotebook",
        "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
        "ApplicationMode": "INTERACTIVE",
        "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZeppelinRole",
...
```

6. 애플리케이션을 시작하려면 다음 명령을 실행합니다. 샘플 값을 계정 ID로 바꿉니다.

```
aws kinesisanalyticsv2 start-application --application-arn
 arn:aws:kinesisanalyticsus-east-1:012345678901:application/MyNotebook\
```

Kinesis 데이터 스트림으로 데이터 전송

Kinesis 데이터 스트림으로 테스트 데이터를 보내려면 다음을 수행하세요.

1. Kinesis 데이터 제너레이터(KDG)를 엽니다.
2. Cognito 사용자 생성을 선택합니다. CloudFormation
3. AWS CloudFormation 콘솔이 Kinesis Data Generator 템플릿과 함께 열립니다. 다음을 선택합니다.
4. 스택 세부 정보 지정 페이지에서 Cognito 사용자의 사용자 이름 및 암호를 입력합니다. 다음을 선택합니다.
5. 스택 옵션 구성 페이지에서 다음을 선택합니다.
6. Kinesis-데이터 생성기-Cognito-User 검토 페이지에서 IAM 리소스를 생성할 수 있는 확인을 선택합니다. AWS CloudFormation 체크박스. 스택 생성을 선택합니다.
7. AWS CloudFormation 스택 생성이 완료될 때까지 기다립니다. 스택이 완료되면 AWS CloudFormation 콘솔에서 Kinesis-Data-Generator-Cognito-User 스택을 열고 출력 탭을 선택합니다. KinesisDataGeneratorUrl출력 값에 대해 나열된 URL을 엽니다.
8. Amazon Kinesis Data Generator 페이지에서 4단계에서 생성한 보안 인증을 사용하여 로그인합니다.
9. 다음 페이지에서 다음 값을 입력합니다.

리전	**us-east-1**
스트림/Kinesis Data Firehose 스트림	**ExampleInputStream**
초당 레코드 수	**1**

레코드 템플릿의 경우 다음 코드를 붙여넣습니다.

```
{
    "ticker": "{{random.arrayElement(
        ["AMZN","MSFT","GOOG"]
    )}}",
    "price": {{random.number(
        {
            "min":10,
            "max":150
        }
    )}}
}
```

10. 데이터 보내기를 선택합니다.
11. 생성기가 데이터를 Kinesis 데이터 스트림으로 전송합니다.

    다음 섹션을 완료하는 동안 생성기를 계속 실행하세요.

Studio 노트북을 테스트

이 섹션에서는 Studio 노트북을 사용하여 Kinesis 데이터 스트림의 데이터를 쿼리합니다.

1. https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 애플리케이션 페이지에서 Studio 노트북 탭을 선택합니다. 선택합니다 MyNotebook.
3. MyNotebook페이지에서 Apache Zeppelin에서 열기를 선택합니다.

   Apache Zeppelin 인터페이스가 새 탭에서 열립니다.
4. 제플린에 오신 것을 환영합니다! 페이지에서 Zeppelin Note를 선택하세요.

5. Zeppelin 노트 페이지에서 새로운 노트에 다음 쿼리를 입력합니다.

```
%flink.ssql(type=update)
select * from stock
```

실행 아이콘을 선택합니다.

잠시 후 노트에 Kinesis 데이터 스트림의 데이터가 표시됩니다.

애플리케이션에서 작동 측면을 볼 수 있도록 Apache Flink 대시보드를 열려면 FLINK JOB을 선택합니다. Flink 대시보드에 대한 자세한 설명은 Managed Service for Apache Flink 개발자 가이드의 Apache Flink 대시보드를 참조하세요.

Flink Streaming SQL 쿼리의 더 많은 예는 Apache Flink 설명서의 쿼리를 참조하세요.

Amazon MSK로 Studio 노트북 만들기

이 자습서에서는 소스로 Amazon MSK 클러스터를 사용하는 Studio 노트북을 생성하는 방법을 설명합니다.

이 자습서는 다음 섹션을 포함하고 있습니다:

- 설정
- VPC에 NAT 게이트웨이 추가
- AWS Glue 연결 및 표 생성
- Amazon MSK로 Studio 노트북 생성
- Amazon MSK 클러스터로 데이터 전송
- Studio 노트북을 테스트

설정

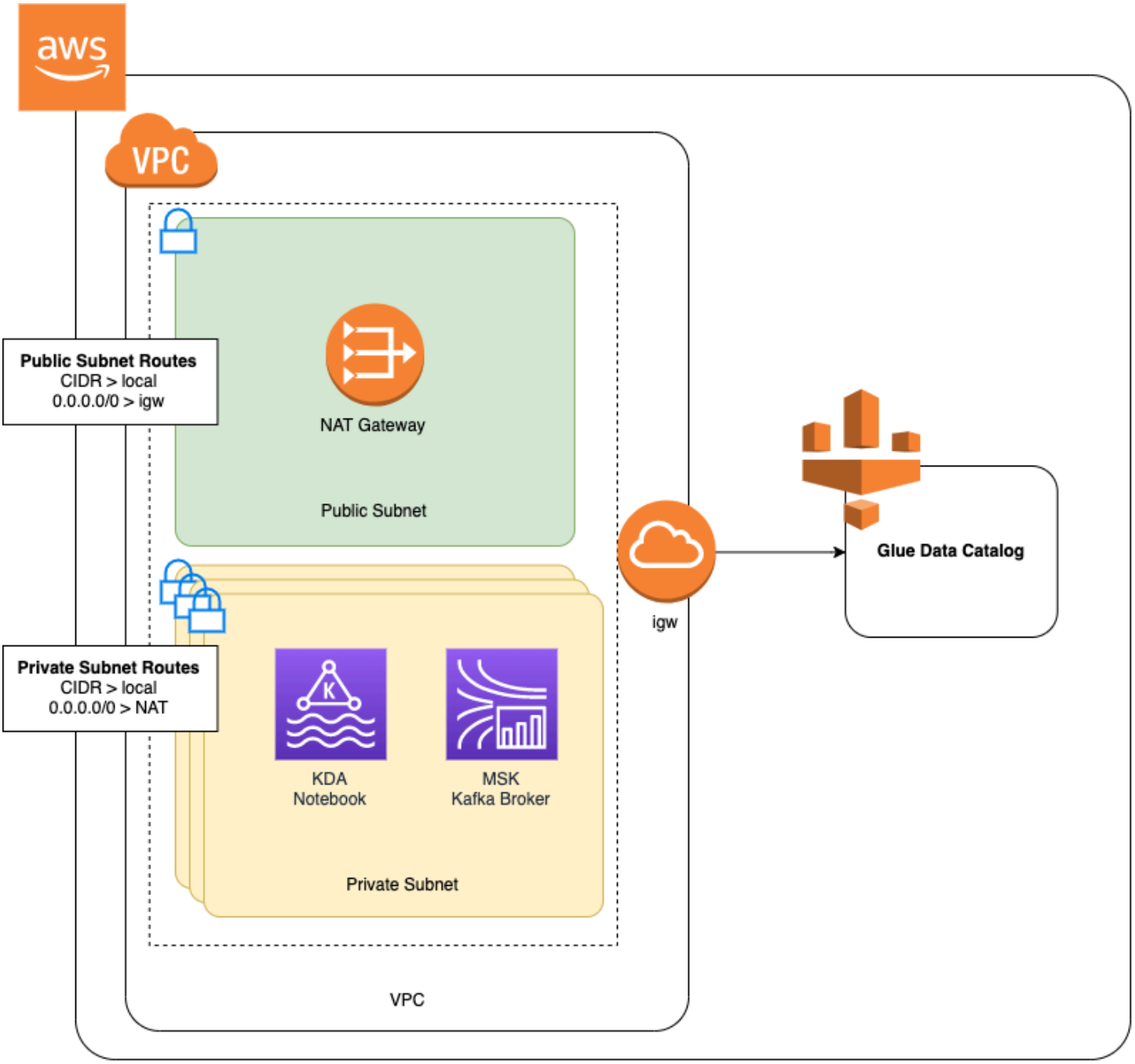
이 자습서에서는 일반 텍스트 액세스를 허용하는 Amazon MSK 클러스터가 필요합니다. Amazon MSK 클러스터를 아직 설정하지 않은 경우, Amazon MSK 사용 시작하기 자습서를 따라 Amazon VPC, Amazon MSK 클러스터, 주제 및 Amazon EC2 클라이언트 인스턴스를 생성하세요.

자습서에 따라 다음을 수행하십시오:

- 3단계: Amazon MSK 클러스터 생성의 4단계에서 `ClientBroker` 값을 `TLS`에서 **`PLAINTEXT`**로 변경합니다.

VPC에 NAT 게이트웨이 추가

Amazon MSK 사용 시작하기 자습서에 따라 Amazon MSK 클러스터를 생성했거나 기존 Amazon VPC에 프라이빗 서브넷용 NAT 게이트웨이가 아직 없는 경우, Amazon VPC에 NAT 게이트웨이를 추가해야 합니다. 다음 다이어그램은 아키텍처입니다.



Amazon VPC용 NAT 게이트웨이를 생성하려면 다음을 수행하세요:

1. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.

2. 왼쪽 탐색 모음에서 NAT 게이트웨이를 선택합니다.
3. NAT 게이트웨이 페이지에서 NAT 게이트웨이 생성을 선택합니다.
4. NAT 게이트웨이 생성 페이지에서 다음 값을 입력합니다:

명칭 - 옵션	**`ZeppelinGateway`**
서브넷	AWSKafkaTutorialSubnet1
탄력적 IP 할당 ID	Choose an available Elastic IP. If there are no Elastic IPs available, choose 탄력적 IP 할당, and then choose the Elasic IP that the console creates.

   NAT 게이트웨이 생성을 선택합니다.
5. 왼쪽 탐색 모음에서 경로 표을 선택합니다.
6. [Create Route Table]을 선택합니다.
7. 경로 표 생성 페이지에서 다음 정보를 제공합니다:

   - Name tag: **`ZeppelinRouteTable`**
   - VPC: VPC(예: AWSKafkaTutorialVPC)를 선택합니다.

   생성을 선택합니다.
8. 경로 표 목록에서 ZeppelinRouteTable을 선택합니다. 경로 탭에서 경로 편집을 선택합니다.
9. 경로 편집 페이지에서 경로 추가를 선택합니다.
10. 에서 대상 주소에 **`0.0.0.0/0`**을 입력합니다. 타겟에서 NAT 게이트웨이, ZeppelinGateway를 선택합니다. 경로 저장을 선택합니다. 닫기를 선택합니다.
11. 경로 표 페이지에서 ZeppelinRouteTable을 선택한 상태에서 서브넷 연결 탭을 선택합니다. 서브넷 연결 편집을 선택합니다.
12. 서브넷 연결 편집 페이지에서 AWSKafkaTutorialSubnet2와 AWSKafkaTutorialSubnet3을 선택합니다. Save를 선택합니다.

AWS Glue 연결 및 표 생성

Studio 노트북은 Amazon MSK 데이터 소스에 대한 메타데이터용 AWS Glue 데이터베이스를 사용합니다. 이 섹션에서는 Amazon MSK 클러스터에 액세스하는 방법을 설명하는 AWS Glue 연결과 데이터 소스의 데이터를 Studio 노트북과 같은 클라이언트에 제공하는 방법을 설명하는 AWS Glue 표를 생성합니다.

연결 생성

1. AWS Management Console에 로그인하여 https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 아직 AWS Glue 데이터베이스가 없는 경우 왼쪽 탐색 모음에서 데이터베이스를 선택합니다. 데이터베이스 추가를 선택합니다. 데이터베이스 추가 창에서 데이터베이스 이름을 **`default`**(으)로 입력합니다. 생성을 선택합니다.
3. 왼쪽 탐색 모음에서 연결을 선택합니다. 연결 추가를 선택합니다.
4. 연결 추가 창에서 다음 값을 입력합니다.

   - 연결 명칭에 **`ZeppelinConnection`**을 입력합니다.
   - 연결 유형에서 Kafka를 선택합니다.
   - Kafka 부트스트랩 서버 URL에 클러스터의 부트스트랩 브로커 문자열을 제공하세요. MSK 콘솔에서 또는 다음 CLI 명령을 입력하여 부트스트랩 브로커를 가져올 수 있습니다.

     ```
     aws kafka get-bootstrap-brokers --region us-east-1 --cluster-arn ClusterArn
     ```

   - SSL 연결 필요 확인란의 선택을 취소하세요.

   다음(Next)을 선택합니다.
5. VPC 페이지에서 다음 값을 입력합니다.

   - VPC에서 VPC의 명칭을 선택합니다(예: AWSKafkaTutorialVPC).
   - 서브넷에서 AWSKafkaTutorialSubnet2를 선택합니다.
   - 보안 그룹의 경우 사용 가능한 모든 그룹을 선택합니다.

   다음(Next)을 선택합니다.
6. 연결 속성 및 연결 액세스 페이지에서 마침을 선택합니다.

표 생성

> **Note**
>
> 다음 단계에 설명된 대로 표을 수동으로 생성하거나 Apache Zeppelin 내의 노트북에서 Managed Service for Apache Flink용 표 커넥터 코드 생성을 사용하여 DDL 문을 통해 표을 생성할 수 있습니다. 그런 다음 AWS Glue을(를) 체크인하여 테이블이 올바르게 생성되었는지 확인할 수 있습니다.

1. 왼쪽 탐색 모음에서 표을 선택합니다. 표 페이지에서 표 추가, 표 수동 추가를 선택합니다.
2. 표 속성 설정 페이지에서 표 명칭에 **stock**을 입력합니다. 이전에 만든 데이터베이스를 선택했는지 확인하세요. 다음(Next)을 선택합니다.
3. 데이터 스토어 추가 페이지에서 Kafka를 선택합니다. 주제 명칭에는 주제 명칭(예: AWSKafkaTutorialTopic)을 입력합니다. 연결에서 Zeppel In Connection을 선택합니다.
4. 분류 페이지에서 JSON을 선택합니다. 다음(Next)을 선택합니다.
5. 스키마 정의 페이지에서 열 추가를 선택하여 열을 추가합니다. 다음 속성을 가진 열을 추가합니다:

열 명칭	데이터 형식
##	**string**
price	**double**

   다음(Next)을 선택합니다.
6. 다음 페이지에서 설정을 확인하고 마침을 선택합니다.
7. 테이블 목록에서 새로 생성한 테이블을 선택합니다.
8. 테이블 편집을 선택하고 `managed-flink.proctime` 키와 `proctime` 값이 있는 속성을 추가합니다.
9. 적용을 선택합니다.

Amazon MSK로 Studio 노트북 생성

애플리케이션에서 사용하는 리소스를 생성했으니 이제 Studio 노트북을 생성합니다.

AWS Management Console 또는 AWS CLI를 사용하여 애플리케이션을 생성할 수 있습니다.

- AWS Management Console을(를) 사용하여 Studio 노트북을 만드세요.
- AWS CLI을(를) 사용하여 Studio 노트북을 만드세요.

> **Note**
>
> Amazon MSK 콘솔에서 기존 클러스터를 선택한 다음 실시간 데이터 처리를 선택하여 Studio 노트북을 생성할 수도 있습니다.

AWS Management Console을(를) 사용하여 Studio 노트북을 만드세요.

1. https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 애플리케이션 페이지에서 Studio 탭을 선택합니다. Studio 노트북 생성을 선택합니다.

   > **Note**
   >
   > Amazon MSK 또는 Kinesis Data Streams 콘솔에서 Studio 노트북을 생성하려면 입력 Amazon MSK 클러스터 또는 Kinesis 데이터 스트림을 선택한 다음 실시간 데이터 처리를 선택합니다.

3. Studio 노트북 생성 페이지에서 다음 정보를 입력합니다:

   - Studio 노트북 명칭 **`MyNotebook`**을 입력합니다.
   - AWS Glue 데이터베이스의 기본값을 선택합니다.

   Studio 노트북 생성을 선택합니다.
4. MyNotebook 페이지에서 구성 탭을 선택합니다. 네트워킹 섹션에서 편집을 선택합니다.
5. MyNotebook의 네트워킹 편집 페이지에서 Amazon MSK 클러스터를 기반으로 하는 VPC 구성을 선택합니다. Amazon MSK 클러스터용 Amazon MSK 클러스터를 선택하세요. Save changes(변경 사항 저장)를 선택합니다.
6. MyNotebook 페이지에서 실행을 선택합니다. 상태가 실행 중으로 표시될 때까지 기다리세요.

AWS CLI을(를) 사용하여 Studio 노트북을 만드세요.

AWS CLI를 사용하여 Studio 노트북을 생성하려면 다음을 수행하세요:

1. 다음 정보가 있는지 확인합니다. 애플리케이션을 생성하려면 이러한 값이 필요합니다.
   - 계정 ID
   - Amazon MSK 클러스터를 포함하는 Amazon VPC의 서브넷 ID 및 보안 그룹 ID
2. 다음 콘텐츠를 가진 `create.json`이라는 파일을 생성합니다: 자리 표시자 값을 정보로 바꿉니다.

```
{
    "ApplicationName": "MyNotebook",
    "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
    "ApplicationMode": "INTERACTIVE",
    "ServiceExecutionRole": "arn:aws:iam::AccountID:role/ZeppelinRole",
    "ApplicationConfiguration": {
        "ApplicationSnapshotConfiguration": {
            "SnapshotsEnabled": false
        },
        "VpcConfigurations": [
            {
                "SubnetIds": [
                    "SubnetID 1",
                    "SubnetID 2",
                    "SubnetID 3"
                ],
                "SecurityGroupIds": [
                    "VPC Security Group ID"
                ]
            }
        ],
        "ZeppelinApplicationConfiguration": {
            "CatalogConfiguration": {
                "GlueDataCatalogConfiguration": {
                    "DatabaseARN": "arn:aws:glue:us-east-1:AccountID:database/
default"
                }
            }
        }
    }
}
```

3. 애플리케이션을 생성하려면 다음 명령을 실행합니다:

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create.json
```

4. 명령이 완료되면 새 Studio 노트북의 세부 정보를 보여 주는 다음과 유사한 출력이 나타나야 합니다.

```
{
    "ApplicationDetail": {
        "ApplicationARN": "arn:aws:kinesisanalyticsus-
east-1:012345678901:application/MyNotebook",
        "ApplicationName": "MyNotebook",
        "RuntimeEnvironment": "ZEPPELIN-FLINK-3_0",
        "ApplicationMode": "INTERACTIVE",
        "ServiceExecutionRole": "arn:aws:iam::012345678901:role/ZeppelinRole",
...
```

5. 애플리케이션을 시작하려면 다음 명령을 실행합니다. 샘플 값을 계정 ID로 바꿉니다.

```
aws kinesisanalyticsv2 start-application --application-arn
 arn:aws:kinesisanalyticsus-east-1:012345678901:application/MyNotebook\
```

Amazon MSK 클러스터로 데이터 전송

이 섹션에서는 Amazon EC2 클라이언트에서 Python 스크립트를 실행하여 Amazon MSK 데이터 소스로 데이터를 전송합니다.

1. Amazon EC2 클라이언트에 연결합니다.
2. 다음 명령을 실행하여 Python 버전 3, Pip 및 Python용 Kafka 패키지를 설치하고 작업을 확인합니다:

```
sudo yum install python37
curl -O https://bootstrap.pypa.io/get-pip.py
python3 get-pip.py --user
pip install kafka-python
```

3. 다음 명령을 입력하여 클라이언트 머신에서 AWS CLI 구성합니다:

```
aws configure
```

region에 계정 자격 증명 및 **us-east-1**을 제공하세요.

4. 다음 콘텐츠를 가진 stock.py이라는 파일을 생성합니다: 샘플 값을 Amazon MSK 클러스터의 부트스트랩 브로커 문자열로 바꾸고, 주제가 AWSKafkaTutorialTopic이 아닌 경우 주제 명칭을 업데이트하세요.

```
from kafka import KafkaProducer
import json
import random
from datetime import datetime

BROKERS = "<<Bootstrap Broker List>>"
producer = KafkaProducer(
    bootstrap_servers=BROKERS,
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    retry_backoff_ms=500,
    request_timeout_ms=20000,
    security_protocol='PLAINTEXT')


def getStock():
    data = {}
    now = datetime.now()
    str_now = now.strftime("%Y-%m-%d %H:%M:%S")
    data['event_time'] = str_now
    data['ticker'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
    price = random.random() * 100
    data['price'] = round(price, 2)
    return data


while True:
    data =getStock()
    # print(data)
    try:
        future = producer.send("AWSKafkaTutorialTopic", value=data)
        producer.flush()
        record_metadata = future.get(timeout=10)
        print("sent event to Kafka! topic {} partition {} offset
 {}".format(record_metadata.topic, record_metadata.partition,
 record_metadata.offset))
    except Exception as e:
        print(e.with_traceback())
```

5. 다음 명령으로 스크립트를 실행합니다:

```
$ python3 stock.py
```

6. 다음 섹션을 완료하는 동안 스크립트는 실행 상태로 두세요.

Studio 노트북을 테스트

이 섹션에서는 Studio 노트북을 사용하여 Amazon MSK 클러스터에서 데이터를 쿼리합니다.

1. https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 애플리케이션 페이지에서 Studio 노트북 탭을 선택합니다. MyNotebook을 선택합니다.
3. MyNotebook 페이지에서 Apache Zeppelin에서 열기를 선택합니다.

   Apache Zeppelin 인터페이스가 새 탭에서 열립니다.
4. 제플린에 오신 것을 환영합니다! 페이지에서 Zeppelin 새로운 노트를 선택하세요.
5. Zeppelin 노트 페이지에서 새로운 노트에 다음 쿼리를 입력합니다.

   ```
   %flink.ssql(type=update)
   select * from stock
   ```

   실행 아이콘을 선택합니다.

   애플리케이션은 Amazon MSK 클러스터의 데이터를 표시합니다.

애플리케이션에서 작동 측면을 볼 수 있도록 Apache Flink 대시보드를 열려면 FLINK JOB을 선택합니다. Flink 대시보드에 대한 자세한 설명은 Managed Service for Apache Flink 개발자 가이드의 Apache Flink 대시보드를 참조하세요.

Flink Streaming SQL 쿼리의 더 많은 예는 Apache Flink 설명서의 쿼리를 참조하세요.

애플리케이션 및 종속 리소스 정리

Studio 노트북 삭제

1. Managed Service for Apache Flink 콘솔을 엽니다.
2. MyNotebook을 선택합니다.

3. 작업을 선택한 다음 삭제를 선택합니다.

AWS Glue 데이터베이스 및 연결 삭제

1. https://console.aws.amazon.com/glue/에서 AWS Glue 콘솔을 엽니다.
2. 왼쪽 탐색 모음에서 데이터베이스를 선택합니다. 기본값 옆의 체크박스를 선택하여 선택합니다. 작업, 데이터베이스 삭제를 선택합니다. 선택 항목을 확인합니다.
3. 왼쪽 탐색 모음에서 연결을 선택합니다. ZeppelInConnection 옆의 체크박스를 선택하여 선택하세요. 작업, 연결 삭제를 선택합니다. 선택 항목을 확인합니다.

IAM 역할 및 정책 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 왼쪽 탐색 메뉴에서 역할을 선택합니다.
3. 검색창을 사용하여 ZeppelInRole 역할을 검색합니다.
4. ZeppelInRole 역할을 선택하세요. 역할 삭제를 선택합니다. 삭제를 확인합니다.

CloudWatch 로그 그룹 삭제

콘솔을 사용하여 애플리케이션을 생성하면 콘솔에서 CloudWatch Logs 그룹과 로그 스트림을 자동으로 생성합니다. AWS CLI를 사용하여 애플리케이션을 생성한 경우에는 로그 그룹과 스트림이 없습니다.

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 왼쪽 탐색 모음에서 로그 그룹을 선택합니다.
3. /AWS/KinesisAnalytics/MyNotebook 로그 그룹을 선택합니다.
4. 작업(Actions), 로그 그룹 삭제(Delete log group(s))를 선택합니다. 삭제를 확인합니다.

Kinesis Data Streams 리소스 정리

Kinesis 스트림을 삭제하려면 Kinesis 데이터 스트림 콘솔을 열고 Kinesis 스트림을 선택한 다음 작업, 삭제를 선택합니다.

MSK 리소스 정리

이 자습서에서 Amazon MSK 클러스터를 생성한 경우 이번 섹션의 단계를 따르세요. 이 섹션에는 Amazon EC2 클라이언트 인스턴스, Amazon VPC 및 Amazon MSK 클러스터를 정리하는 방법이 나와 있습니다.

Amazon MSK 클러스터 삭제

이 자습서에서 Amazon MSK 클러스터를 생성한 경우 다음 단계를 따르세요.

1. https://console.aws.amazon.com/msk/home?region=us-east-1#/home/에서 Amazon MSK 콘솔을 엽니다.
2. AWSKafkaTutorialCluster를 선택합니다. 삭제를 선택합니다. 나타나는 창에 **delete**을(를) 입력하고 선택을 확인합니다.

클라이언트 인스턴스 종료

이 자습서에서 Amazon EC2 클라이언트 인스턴스를 생성한 경우 다음 단계를 따르세요.

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 왼쪽 탐색 모음에서 인스턴스를 선택합니다.
3. ZeppelinClient 옆의 체크박스를 선택하여 선택하세요.
4. 인스턴스 상태, 인스턴스 종료를 차례로 선택합니다.

Amazon VPC 삭제

이 자습서에서 Amazon VPC를 만든 경우 다음 단계를 따르세요.

1. https://console.aws.amazon.com/ec2/에서 Amazon EC2 콘솔을 엽니다.
2. 왼쪽 탐색 표시줄에서 네트워크 인터페이스를 선택합니다.
3. 검색창에 VPC ID를 입력하고 Enter 키를 눌러 검색합니다.
4. 테이블 헤더의 체크박스를 선택하여 표시된 모든 네트워크 인터페이스를 선택합니다.
5. 작업, 분리를 선택합니다. 표시되는 창에서 강제 분리 아래에서 활성화를 선택합니다. 분리를 선택하고 모든 네트워크 인터페이스가 사용 가능 상태에 도달할 때까지 기다립니다.
6. 테이블 헤더의 체크박스를 선택하여 표시된 모든 네트워크 인터페이스를 다시 선택합니다.
7. 작업, 삭제를 선택합니다. 작업을 확인합니다.
8. https://console.aws.amazon.com/vpc/에서 Amazon VPC 콘솔을 엽니다.

9. AWS KafkaTutorialVPC를 선택합니다. 작업, VPC 삭제를 선택합니다. **delete**을(를) 입력하고 삭제를 확인합니다.

자습서: 지속 가능한 상태의 애플리케이션으로 배포

다음 자습서에서는 지속 가능한 상태의 Apache Flink 애플리케이션용 관리 서비스로 Studio 노트북을 배포하는 방법을 보여줍니다.

이 자습서는 다음 섹션을 포함하고 있습니다:

- 설정
- AWS Management Console를 사용하여 지속 가능한 상태의 애플리케이션을 배포하세요.
- AWS CLI를 사용하여 지속 가능한 상태의 애플리케이션을 배포하세요.

설정

Kinesis Data Streams 또는 Amazon MSK를 사용하여 Studio 노트북 자습서 만들기에 따라 새 Studio 노트북을 생성하세요. Studio 노트북의 이름을 `ExampleTestDeploy`(으)로 지정하세요.

AWS Management Console를 사용하여 지속 가능한 상태의 애플리케이션을 배포하세요.

1. 콘솔의 애플리케이션 코드 위치(선택 사항)에 패키지 코드를 저장할 S3 버킷 위치를 추가합니다. 이렇게 하면 단계를 통해 노트북에서 직접 애플리케이션을 배포하고 실행할 수 있습니다.
2. Amazon S3 버킷을 읽고 쓰는 데 사용하는 역할을 활성화하고 Managed Service for Apache Flink 애플리케이션을 시작하는 데 필요한 권한을 애플리케이션 역할에 추가합니다.
   - AmazonS3FullAccess
   - Amazonmanaged-flinkFullAccess
   - 해당하는 경우 소스, 대상 및 VPC에 액세스할 수 있습니다. 자세한 설명은 Studio 노트북의 IAM 권한 섹션을 참조하세요.
3. 다음 예제 코드를 사용하세요.

```
%flink.ssql(type=update)
CREATE TABLE exampleoutput (
  'ticket' VARCHAR,
  'price' DOUBLE
```

```
)
WITH (
  'connector' = 'kinesis',
  'stream' = 'ExampleOutputStream',
  'aws.region' = 'us-east-1',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'json'
);

INSERT INTO exampleoutput SELECT ticker, price FROM exampleinputstream
```

4. 이번 기능 출시와 함께 노트북의 각 노트 오른쪽 상단 모서리에 노트북 이름과 함께 새로운 드롭다운이 표시됩니다. 다음을 수행할 수 있습니다.

   - AWS Management Console에서 Studio 노트북 설정을 볼 수 있습니다.
   - Zeppelin Note를 빌드하고 Amazon S3로 내보내세요. 이때 애플리케이션 이름을 입력하고 빌드 및 내보내기를 선택합니다. 내보내기가 완료되면 알림을 받게 됩니다.
   - 필요한 경우 Amazon S3의 실행 파일에서 추가 테스트를 보고 실행할 수 있습니다.
   - 빌드가 완료되면 지속 가능한 상태 및 자동 크기 조정 기능을 갖춘 Kinesis 스트리밍 애플리케이션으로 코드를 배포할 수 있습니다.
   - 드롭다운을 사용하여 Zeppelin Note를 Kinesis 스트리밍 애플리케이션으로 배포를 선택합니다. 애플리케이션 이름을 검토하고 AWS 콘솔을 통한 배포를 선택합니다.
   - 그러면 Managed Service for Apache Flink 애플리케이션을 만들기 위한 AWS Management Console 페이지로 이동합니다. 참고로 애플리케이션 이름, 병렬 처리, 코드 위치, 기본 Glue DB, VPC (해당하는 경우) 및 IAM 역할이 미리 입력되어 있습니다. IAM 역할에 소스 및 대상에 필요한 권한이 있는지 확인하세요. 안정적인 애플리케이션 상태 관리를 위해 스냅샷은 기본적으로 활성화됩니다.
   - 애플리케이션 생성을 선택합니다.
   - 설정 구성 및 수정을 선택한 다음 Run을 선택하여 스트리밍 애플리케이션을 시작할 수 있습니다.

AWS CLI를 사용하여 지속 가능한 상태의 애플리케이션을 배포하세요.

AWS CLI를 사용하여 애플리케이션을 배포하려면 베타 2 정보와 함께 제공된 서비스 모델을 사용하도록 AWS CLI를 업데이트해야 합니다. 업데이트된 서비스 모델을 사용하는 방법에 대한 자세한 내용은 설정을(를) 참조하세요.

다음 예제 코드에서는 새 Studio 노트북을 생성합니다.

```
aws kinesisanalyticsv2 create-application \
     --application-name <app-name> \
     --runtime-environment ZEPPELIN-FLINK-3_0 \
     --application-mode INTERACTIVE \
     --service-execution-role <iam-role>
     --application-configuration '{
       "ZeppelinApplicationConfiguration": {
         "CatalogConfiguration": {
           "GlueDataCatalogConfiguration": {
             "DatabaseARN": "arn:aws:glue:us-east-1:<account>:database/<glue-database-
name>"
           }
         }
       },
       "FlinkApplicationConfiguration": {
         "ParallelismConfiguration": {
           "ConfigurationType": "CUSTOM",
           "Parallelism": 4,
           "ParallelismPerKPU": 4
         }
       },
       "DeployAsApplicationConfiguration": {
            "S3ContentLocation": {
               "BucketARN": "arn:aws:s3:::<s3bucket>",
               "BasePath": "/something/"
            }
        },
       "VpcConfigurations": [
         {
           "SecurityGroupIds": [
             "<security-group>"
           ],
           "SubnetIds": [
             "<subnet-1>",
             "<subnet-2>"
           ]
         }
       ]
     }' \
     --region us-east-1
```

다음 코드 예제에서는 Studio 노트북을 시작합니다.

```
aws kinesisanalyticsv2 start-application \
    --application-name <app-name> \
    --region us-east-1 \
    --no-verify-ssl
```

다음 코드는 애플리케이션의 Apache Zeppelin 노트북 페이지의 URL을 반환합니다.

```
aws kinesisanalyticsv2 create-application-presigned-url \
    --application-name <app-name> \
    --url-type ZEPPELIN_UI_URL \

    --region us-east-1 \
    --no-verify-ssl
```

예제

다음 예제 쿼리는 Studio 노트북에서 창 쿼리를 사용하여 데이터를 분석하는 방법을 보여줍니다.

- Amazon MSK/Apache Kafka로 테이블 생성
- Kinesis를 사용하여 테이블 생성
- 텀블링 윈도우
- 슬라이딩 윈도우
- 대화형 SQL
- BlackHole SQL 커넥터
- 데이터 생성기
- 대화형 Scala
- 대화형 Python
- 대화형 Python, SQL, 스칼라
- 계정 간 Kinesis 데이터 스트림

Apache Flink SQL 쿼리 설정에 대한 자세한 내용은 대화형 데이터 분석을 위한 Zeppelin 노트북의 Flink를 참조하세요.

Apache Flink 대시보드에서 애플리케이션을 보려면 애플리케이션의 Zeppelin Note 페이지에서 FLINK JOB을 선택하세요.

윈도우 쿼리에 대한 자세한 내용은 Apache Flink 설명서의 윈도우를 참조하세요.

Apache Flink Streaming SQL 쿼리의 더 많은 예는 Apache Flink 설명서의 쿼리를 참조하세요.

Amazon MSK/Apache Kafka로 테이블 생성

Managed Service for Apache Flink Studio와 Amazon MSK Flink 커넥터를 사용하여 일반 텍스트, SSL 또는 IAM 인증과의 연결을 인증할 수 있습니다. 요구 사항에 따라 특정 속성을 사용하여 테이블을 생성하세요.

```
-- Plaintext connection

CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
  'connector' = 'kafka',
  'topic' = 'your_topic',
  'properties.bootstrap.servers' = '<bootstrap servers>',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);

-- SSL connection

CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
  'connector' = 'kafka',
   'topic' = 'your_topic',
  'properties.bootstrap.servers' = '<bootstrap servers>',
  'properties.security.protocol' = 'SSL',
  'properties.ssl.truststore.location' = '/usr/lib/jvm/java-11-amazon-corretto/lib/
security/cacerts',
  'properties.ssl.truststore.password' = 'changeit',
  'properties.group.id' = 'myGroup',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);

-- IAM connection (or for MSK Serverless)
```

```
CREATE TABLE your_table (
  `column1` STRING,
  `column2` BIGINT
) WITH (
  'connector' = 'kafka',
  'topic' = 'your_topic',
  'properties.bootstrap.servers' = '<bootstrap servers>',
  'properties.security.protocol' = 'SASL_SSL',
  'properties.sasl.mechanism' = 'AWS_MSK_IAM',
  'properties.sasl.jaas.config' = 'software.amazon.msk.auth.iam.IAMLoginModule
 required;',
  'properties.sasl.client.callback.handler.class' =
 'software.amazon.msk.auth.iam.IAMClientCallbackHandler',
  'properties.group.id' = 'myGroup',
  'scan.startup.mode' = 'earliest-offset',
  'format' = 'json'
);
```

Apache Kafka SQL Connector에서 이러한 속성을 다른 속성과 결합할 수 있습니다.

Kinesis를 사용하여 테이블 생성

다음 예제에서는 Kinesis를 사용하여 테이블을 생성합니다.

```
CREATE TABLE KinesisTable (
  `column1` BIGINT,
  `column2` BIGINT,
  `column3` BIGINT,
  `column4` STRING,
  `ts` TIMESTAMP(3)
)
PARTITIONED BY (column1, column2)
WITH (
  'connector' = 'kinesis',
  'stream' = 'test_stream',
  'aws.region' = '<region>',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'csv'
);
```

사용할 수 있는 다른 속성에 대한 자세한 내용은 Amazon Kinesis Data Streams SQL Connector를 참조하세요.

텀블링 윈도우

다음 Flink Streaming SQL 쿼리는 ZeppelinTopic 테이블에서 각 5초의 텀블링 윈도우 내에서 가장 높은 가격을 선택합니다.

```
%flink.ssql(type=update)
SELECT TUMBLE_END(event_time, INTERVAL '5' SECOND) as winend, MAX(price) as
 five_second_high, ticker
FROM ZeppelinTopic
GROUP BY ticker, TUMBLE(event_time, INTERVAL '5' SECOND)
```

슬라이딩 윈도우

다음 Apache Flink Streaming SQL 쿼리는 ZeppelinTopic 표에서 각 5초 슬라이딩 윈도우에서 가장 높은 가격을 선택합니다.

```
%flink.ssql(type=update)
SELECT HOP_END(event_time, INTERVAL '3' SECOND, INTERVAL '5' SECOND) AS winend,
 MAX(price) AS sliding_five_second_max
FROM ZeppelinTopic//or your table name in AWS Glue
GROUP BY HOP(event_time, INTERVAL '3' SECOND, INTERVAL '5' SECOND)
```

대화형 SQL

이 예제는 최대 이벤트 시간 및 처리 시간과 키 값 테이블의 값의 합계를 인쇄합니다. the section called "데이터 생성기" 샘플 데이터 생성 스크립트가 실행 중인지 확인하세요. Studio 노트북에서 필터링 및 조인과 같은 다른 SQL 쿼리를 시도하려면 Apache Flink 설명서: Apache Flink 설명서의 쿼리를 참조하세요.

```
%flink.ssql(type=single, parallelism=4, refreshInterval=1000, template=<h1>{2}</h1>
 records seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints how many records from the `key-value-stream` we have
 seen so far, along with the current processing and event time.
SELECT
  MAX(`et`) as `et`,
  MAX(`pt`) as `pt`,
  SUM(`value`) as `sum`
FROM
  `key-values`
```

```
%flink.ssql(type=update, parallelism=4, refreshInterval=1000)

-- An interactive tumbling window query that displays the number of records observed
 per (event time) second.
-- Browse through the chart views to see different visualizations of the streaming
 result.
SELECT
  TUMBLE_START(`et`, INTERVAL '1' SECONDS) as `window`,
  `key`,
  SUM(`value`) as `sum`
FROM
  `key-values`
GROUP BY
  TUMBLE(`et`, INTERVAL '1' SECONDS),
  `key`;
```

BlackHole SQL 커넥터

BlackHole SQL 커넥터는 쿼리를 테스트하기 위해 Kinesis 데이터 스트림 또는 Amazon MSK 클러스터를 생성할 필요가 없습니다. BlackHoleSQL 커넥터에 대한 자세한 내용은 Apache BlackHole Flink 설명서의 SQL 커넥터를 참조하십시오. 이 예제에서 기본 카탈로그는 인메모리 카탈로그입니다.

```
%flink.ssql

CREATE TABLE default_catalog.default_database.blackhole_table (
 `key` BIGINT,
 `value` BIGINT,
 `et` TIMESTAMP(3)
) WITH (
 'connector' = 'blackhole'
)
```

```
%flink.ssql(parallelism=1)

INSERT INTO `test-target`
SELECT
  `key`,
  `value`,
  `et`
FROM
  `test-source`
WHERE
```

```
  `key` > 3
```

```
%flink.ssql(parallelism=2)

INSERT INTO `default_catalog`.`default_database`.`blackhole_table`
SELECT
  `key`,
  `value`,
  `et`
FROM
  `test-target`
WHERE
  `key` > 7
```

데이터 생성기

이 예제에서는 Scala를 사용하여 샘플 데이터를 생성합니다. 이 샘플 데이터를 사용하여 다양한 쿼리를 테스트할 수 있습니다. 테이블 생성 명령문을 사용하여 키 값 테이블을 생성합니다.

```
import org.apache.flink.streaming.api.functions.source.datagen.DataGeneratorSource
import org.apache.flink.streaming.api.functions.source.datagen.RandomGenerator
import org.apache.flink.streaming.api.scala.DataStream

import java.sql.Timestamp

// ad-hoc convenience methods to be defined on Table
implicit class TableOps[T](table: DataStream[T]) {
    def asView(name: String): DataStream[T] = {
      if (stenv.listTemporaryViews.contains(name)) {
        stenv.dropTemporaryView("`" + name + "`")
      }
      stenv.createTemporaryView("`" + name + "`", table)
      return table;
    }
}
```

```
%flink(parallelism=4)
val stream = senv
 .addSource(new DataGeneratorSource(RandomGenerator.intGenerator(1, 10), 1000))
 .map(key => (key, 1, new Timestamp(System.currentTimeMillis)))
 .asView("key-values-data-generator")
```

```
%flink.ssql(parallelism=4)
-- no need to define the paragraph type with explicit parallelism (such as
 "%flink.ssql(parallelism=2)")
-- in this case the INSERT query will inherit the parallelism of the of the above
 paragraph
INSERT INTO `key-values`
SELECT
 `_1` as `key`,
 `_2` as `value`,
 `_3` as `et`
FROM
 `key-values-data-generator`
```

대화형 Scala

the section called “대화형 SQL”의 스칼라 번역본입니다. 더 많은 스칼라 예제를 보려면 Apache Flink 설명서의 테이블 API를 참조하세요.

```
%flink
import org.apache.flink.api.scala._
import org.apache.flink.table.api._
import org.apache.flink.table.api.bridge.scala._

// ad-hoc convenience methods to be defined on Table
implicit class TableOps(table: Table) {
    def asView(name: String): Table = {
      if (stenv.listTemporaryViews.contains(name)) {
        stenv.dropTemporaryView(name)
      }
      stenv.createTemporaryView(name, table)
      return table;
    }
}
```

```
%flink(parallelism=4)

// A view that computes many records from the `key-values` we have seen so far, along
 with the current processing and event time.
val query01 = stenv
  .from("`key-values`")
  .select(
    $"et".max().as("et"),
```

```
    $"pt".max().as("pt"),
    $"value".sum().as("sum")
  ).asView("query01")
```

```
%flink.ssql(type=single, parallelism=16, refreshInterval=1000, template=<h1>{2}</h1>
 records seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints the query01 output.
SELECT * FROM query01
```

```
%flink(parallelism=4)

// An tumbling window view that displays the number of records observed per (event
 time) second.
val query02 = stenv
  .from("`key-values`")
  .window(Tumble over 1.seconds on $"et" as $"w")
  .groupBy($"w", $"key")
  .select(
    $"w".start.as("window"),
    $"key",
    $"value".sum().as("sum")
  ).asView("query02")
```

```
%flink.ssql(type=update, parallelism=4, refreshInterval=1000)

-- An interactive query prints the query02 output.
-- Browse through the chart views to see different visualizations of the streaming
 result.
SELECT * FROM `query02`
```

대화형 Python

the section called “대화형 SQL”의 Python 번역은 다음과 같습니다 더 많은 Python 예제를 보려면 Apache Flink 문서의 테이블 API를 참조하세요.

```
%flink.pyflink
from pyflink.table.table import Table

def as_view(table, name):
  if (name in st_env.list_temporary_views()):
```

```
    st_env.drop_temporary_view(name)
  st_env.create_temporary_view(name, table)
  return table

Table.as_view = as_view
```

```
%flink.pyflink(parallelism=16)

# A view that computes many records from the `key-values` we have seen so far, along
 with the current processing and event time
st_env \
  .from_path("`keyvalues`") \
  .select(", ".join([
    "max(et) as et",
    "max(pt) as pt",
    "sum(value) as sum"
  ])) \
  .as_view("query01")
```

```
%flink.ssql(type=single, parallelism=16, refreshInterval=1000, template=<h1>{2}</h1>
 records seen until <h1>Processing Time: {1}</h1> and <h1>Event Time: {0}</h1>)

-- An interactive query prints the query01 output.
SELECT * FROM query01
```

```
%flink.pyflink(parallelism=16)

# A view that computes many records from the `key-values` we have seen so far, along
 with the current processing and event time
st_env \
  .from_path("`key-values`") \
  .window(Tumble.over("1.seconds").on("et").alias("w")) \
  .group_by("w, key") \
  .select(", ".join([
    "w.start as window",
    "key",
    "sum(value) as sum"
  ])) \
  .as_view("query02")
```

```
%flink.ssql(type=update, parallelism=16, refreshInterval=1000)
```

```
-- An interactive query prints the query02 output.
-- Browse through the chart views to see different visualizations of the streaming
 result.
SELECT * FROM `query02`
```

대화형 Python, SQL, 스칼라

대화형 분석을 위해 노트북에서 SQL, Python 및 Scala의 모든 조합을 사용할 수 있습니다. 지속 가능한 상태의 애플리케이션으로 배포할 Studio 노트북에서는 SQL과 Scala를 함께 사용할 수 있습니다. 이 예제에서는 무시되는 섹션과 지속 가능한 상태의 애플리케이션에 배포되는 섹션을 보여줍니다.

```
%flink.ssql
CREATE TABLE `default_catalog`.`default_database`.`my-test-source` (
  `key` BIGINT NOT NULL,
  `value` BIGINT NOT NULL,
  `et` TIMESTAMP(3) NOT NULL,
  `pt` AS PROCTIME(),
  WATERMARK FOR `et` AS `et` - INTERVAL '5' SECOND
)
WITH (
  'connector' = 'kinesis',
  'stream' = 'kda-notebook-example-test-source-stream',
  'aws.region' = 'eu-west-1',
  'scan.stream.initpos' = 'LATEST',
  'format' = 'json',
  'json.timestamp-format.standard' = 'ISO-8601'
)
```

```
%flink.ssql
CREATE TABLE `default_catalog`.`default_database`.`my-test-target` (
  `key` BIGINT NOT NULL,
  `value` BIGINT NOT NULL,
  `et` TIMESTAMP(3) NOT NULL,
  `pt` AS PROCTIME(),
  WATERMARK FOR `et` AS `et` - INTERVAL '5' SECOND
)
WITH (
  'connector' = 'kinesis',
  'stream' = 'kda-notebook-example-test-target-stream',
  'aws.region' = 'eu-west-1',
  'scan.stream.initpos' = 'LATEST',
```

```
  'format' = 'json',
  'json.timestamp-format.standard' = 'ISO-8601'
)
```

```
%flink()

// ad-hoc convenience methods to be defined on Table
implicit class TableOps(table: Table) {
  def asView(name: String): Table = {
    if (stenv.listTemporaryViews.contains(name)) {
      stenv.dropTemporaryView(name)
    }
    stenv.createTemporaryView(name, table)
    return table;
  }
}
```

```
%flink(parallelism=1)
val table = stenv
  .from("`default_catalog`.`default_database`.`my-test-source`")
  .select($"key", $"value", $"et")
  .filter($"key" > 10)
  .asView("query01")
```

```
%flink.ssql(parallelism=1)

-- forward data
INSERT INTO `default_catalog`.`default_database`.`my-test-target`
SELECT * FROM `query01`
```

```
%flink.ssql(type=update, parallelism=1, refreshInterval=1000)

-- forward data to local stream (ignored when deployed as application)
SELECT * FROM `query01`
```

```
%flink

// tell me the meaning of life (ignored when deployed as application!)
print("42!")
```

계정 간 Kinesis 데이터 스트림

Studio 노트북이 있는 계정이 아닌 다른 계정에 있는 Kinesis 데이터 스트림을 사용하려면 Studio 노트북이 실행되는 계정에서 서비스 실행 역할을 생성하고 데이터 스트림이 있는 계정에서 역할 신뢰 정책을 생성하세요. 테이블 생성 DDL 명령문의 Kinesis 커넥터에서 `aws.credentials.provider`, `aws.credentials.role.arn`, 및 `aws.credentials.role.sessionName`을(를) 사용하여 데이터 스트림에 대한 테이블을 생성합니다.

Studio 노트북 계정에 다음 서비스 실행 역할을 사용하세요.

```
{
 "Sid": "AllowNotebookToAssumeRole",
 "Effect": "Allow",
 "Action": "sts:AssumeRole"
 "Resource": "*"
}
```

데이터 스트림 계정에 `AmazonKinesisFullAccess` 정책과 다음 역할 신뢰 정책을 사용하세요.

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

테이블 생성 명령문에는 다음 단락을 사용하세요.

```
%flink.ssql
CREATE TABLE test1 (
name VARCHAR,
age BIGINT
) WITH (
'connector' = 'kinesis',
```

```
'stream' = 'stream-assume-role-test',
'aws.region' = 'us-east-1',
'aws.credentials.provider' = 'ASSUME_ROLE',
'aws.credentials.role.arn' = 'arn:aws:iam::<accountID>:role/stream-assume-role-test-
role',
'aws.credentials.role.sessionName' = 'stream-assume-role-test-session',
'scan.stream.initpos' = 'TRIM_HORIZON',
'format' = 'json'
)
```

문제 해결

이 섹션에는 Studio 노트북의 문제 해결 정보가 포함되어 있습니다.

중단된 애플리케이션 중지

일시적 상태에서 멈춘 응용 프로그램을 중지하려면 `Force` 매개 변수를 로 설정한 상태로 StopApplication작업을 호출하십시오. `true` 자세한 내용은 Managed Service for Apache Flink 개발자 안내서의 Running Applications을 참조하세요.

인터넷에 접속할 수 없는 VPC에서 지속 가능한 상태의 애플리케이션으로 배포

Apache Flink Studio용 관리형 서비스 deploy-as-application 기능은 인터넷 액세스가 없는 VPC 애플리케이션을 지원하지 않습니다. 스튜디오에서 애플리케이션을 빌드한 다음 Managed Service for Apache Flink를 사용하여 Flink 애플리케이션을 수동으로 만들고 노트북에 빌드한 zip 파일을 선택하는 것이 좋습니다.

다음 단계에서는 이 접근 방식을 요약합니다.

1. 스튜디오 애플리케이션을 빌드하고 Amazon S3로 내보냅니다. 이 파일은 zip 파일이어야 합니다.
2. Amazon S3의 zip 파일 위치를 참조하는 코드 경로를 사용하여 Managed Service for Apache Flink 애플리케이션을 수동으로 생성합니다. 또한 다음 `env` 변수(총 2 `groupID`, 3 `var`)를 사용하여 애플리케이션을 구성해야 합니다.
3. kinesis.analytics.flink.run.options
    a. python: source/note.py
    b. jarfile: lib/ .jar PythonApplicationDependencies

4. managed.deploy_as_app.options

   - DatabaseARN: *<glue database ARN (Amazon Resource Name)>*

5. 애플리케이션에서 사용하는 서비스에 대한 Managed Service for Apache Flink Studio 및 Managed Service for Apache Flink IAM 역할에 권한을 부여해야 할 수 있습니다. 두 앱에 동일한 IAM 역할을 사용할 수 있습니다.

D eploy-as-app 크기 및 빌드 시간 단축

필요한 라이브러리를 결정할 수 없기 때문에 Studio deploy-as-app for Python 애플리케이션은 Python 환경에서 사용할 수 있는 모든 것을 패키징합니다. 이로 인해 크기가 필요 이상으로 커질 수 있습니다. deploy-as-app 다음 절차는 종속성을 제거하여 deploy-as-app Python 응용 프로그램 크기를 줄이는 방법을 보여줍니다.

Studio의 deploy-as-app 기능을 사용하여 Python 응용 프로그램을 빌드하는 경우 응용 프로그램이 의존하지 않는 경우 시스템에서 사전 설치된 Python 패키지를 제거하는 것을 고려할 수 있습니다. 이렇게 하면 최종 아티팩트 크기를 줄여 애플리케이션 크기에 대한 서비스 제한을 위반하지 않을 뿐만 아니라 이 기능을 통해 애플리케이션의 빌드 시간을 단축할 수 있습니다. deploy-as-app

다음 명령을 실행하여 설치된 모든 Python 패키지를 각각의 설치된 크기와 함께 나열하고 상당한 크기의 패키지를 선택적으로 제거할 수 있습니다.

```
%flink.pyflink

!pip list --format freeze | awk -F = {'print $1'} | xargs pip show | grep -E
 'Location:|Name:' | cut -d ' ' -f 2 | paste -d ' ' - - | awk '{gsub("-","_",$1); print
 $2 "/" tolower($1)}' | xargs du -sh 2> /dev/null | sort -hr
```

> **Note**
>
> `apache-beam`은 Flink Python이 작동하는 데 필요합니다. 이 패키지와 해당 종속성을 제거하면 안 됩니다.

다음은 제거를 고려할 수 있는 Studio V2의 사전 설치 Python 패키지 목록입니다.

```
scipy
statsmodels
```

```
plotnine
seaborn
llvmlite
bokeh
pandas
matplotlib
botocore
boto3
numba
```

Zeppelin 노트북에서 Python 패키지 제거

1. 애플리케이션을 제거하기 전에 애플리케이션이 패키지 또는 해당 패키지를 사용하는 패키지에 종속되어 있는지 확인합니다. pipdeptree를 사용하여 패키지의 종속 항목을 식별할 수 있습니다.
2. 다음 명령을 실행하여 패키지를 제거합니다.

   ```
   %flink.pyflink
   !pip uninstall -y <package-to-remove>
   ```

3. 실수로 제거한 패키지를 검색해야 하는 경우 다음 명령을 실행합니다.

   ```
   %flink.pyflink
   !pip install <package-to-install>
   ```

Example 예: deploy-as-app 기능이 있는 Python 애플리케이션을 배포하기 전에 **scipy** 패키지를 제거하십시오.

1. pipdeptree를 사용하여 모든 scipy 소비자를 검색하고 scipy를 안전하게 제거할 수 있는지 확인합니다.

   - 노트북을 통해 도구를 설치합니다.

     ```
     %flink.pyflink
     !pip install pipdeptree
     ```

   - 다음을 실행하여 scipy의 역방향 종속성 트리를 가져옵니다.

     ```
     %flink.pyflink
     !pip -r -p scipy
     ```

다음과 유사한 출력 화면이 표시되어야 합니다(간결하게 나타내기 위해 요약됨).

```
...
------------------------------------------------------------------------
scipy==1.8.0
### plotnine==0.5.1 [requires: scipy>=1.0.0]
### seaborn==0.9.0 [requires: scipy>=0.14.0]
### statsmodels==0.12.2 [requires: scipy>=1.1]
    ### plotnine==0.5.1 [requires: statsmodels>=0.8.0]
```

2. 애플리케이션에서 seaborn, statsmodels 및 plotnine의 사용법을 주의 깊게 살펴봅니다. 애플리케이션이 scipy, seaborn, statemodels 또는 plotnine에 종속되지 않은 경우 이러한 패키지를 모두 제거하거나 애플리케이션에 필요하지 않은 패키지만 제거할 수 있습니다.
3. 다음을 실행하여 패키지를 제거합니다.

```
!pip uninstall -y scipy plotnine seaborn statemodels
```

작업 취소

이 섹션에서는 Apache Zeppelin에서 가져올 수 없는 Apache Flink 작업을 취소하는 방법을 보여줍니다. 이러한 작업을 취소하려면 Apache Flink 대시보드로 이동하여 작업 ID를 복사한 다음 다음 예제 중 하나에서 이 작업을 사용하세요

단일 작업을 취소하려면:

```
%flink.pyflink
import requests

requests.patch("https://zeppelin-flink:8082/jobs/[job_id]", verify=False)
```

실행 중인 모든 작업을 취소하려면:

```
%flink.pyflink
import requests

r = requests.get("https://zeppelin-flink:8082/jobs", verify=False)
jobs = r.json()['jobs']

for job in jobs:
```

```
    if (job["status"] == "RUNNING"):
        print(requests.patch("https://zeppelin-flink:8082/jobs/{}".format(job["id"]),
 verify=False))
```

모든 작업을 취소하려면:

```
%flink.pyflink
import requests

r = requests.get("https://zeppelin-flink:8082/jobs", verify=False)
jobs = r.json()['jobs']

for job in jobs:
    requests.patch("https://zeppelin-flink:8082/jobs/{}".format(job["id"]),
 verify=False)
```

Apache Flink 인터프리터 재시작

Studio 노트북에서 Apache Flink 인터프리터를 다시 시작하려면

1. 화면 상단 오른쪽 모서리 부분에 있는 구성을 선택합니다.
2. 인터프리터를 선택합니다.
3. 재시작을 선택한 다음 확인을 선택합니다.

부록: 사용자 지정 IAM 정책 생성

일반적으로 관리형 IAM 정책을 사용하여 애플리케이션이 종속 리소스에 액세스할 수 있도록 허용합니다. 애플리케이션의 권한을 더 세밀하게 제어해야 하는 경우 사용자 지정 IAM 정책을 사용할 수 있습니다. 이 섹션에는 사용자 지정 IAM 정책의 예가 포함되어 있습니다.

> **Note**
>
> 다음 정책 예시에서는 자리 표시자 텍스트를 애플리케이션 값으로 대체합니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- AWS Glue
- CloudWatch 로그

- Kinesis 스트림
- Amazon MSK 클러스터

AWS Glue

다음과 같은 정책 예시는 AWS Glue 데이터베이스에 액세스할 수 있는 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "GlueTable",
            "Effect": "Allow",
            "Action": [
                "glue:GetConnection",
                "glue:GetTable",
                "glue:GetTables",
                "glue:GetDatabase",
                "glue:CreateTable",
                "glue:UpdateTable"
            ],
            "Resource": [
                "arn:aws:glue:<region>:<accountId>:connection/*",
                "arn:aws:glue:<region>:<accountId>:table/<database-name>/*",
                "arn:aws:glue:<region>:<accountId>:database/<database-name>",
                "arn:aws:glue:<region>:<accountId>:database/hive",
                "arn:aws:glue:<region>:<accountId>:catalog"
            ]
        },
        {
            "Sid": "GlueDatabase",
            "Effect": "Allow",
            "Action": "glue:GetDatabases",
            "Resource": "*"
        }
    ]
}
```

CloudWatch 로그

다음 정책은 CloudWatch 로그에 액세스할 수 있는 권한을 부여합니다.

```
{
      "Sid": "ListCloudwatchLogGroups",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:<region>:<accountId>:log-group:*"
      ]
    },
    {
      "Sid": "ListCloudwatchLogStreams",
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogStreams"
      ],
      "Resource": [
        "<logGroupArn>:log-stream:*"
      ]
    },
    {
      "Sid": "PutCloudwatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents"
      ],
      "Resource": [
        "<logStreamArn>"
      ]
    }
```

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 생성하는 경우 콘솔은 CloudWatch 로그에 액세스하는 데 필요한 정책을 애플리케이션 역할에 추가합니다.

Kinesis 스트림

애플리케이션은 소스 또는 대상에 Kinesis Stream을 사용할 수 있습니다. 애플리케이션에는 소스 스트림에서 읽을 수 있는 읽기 권한과 대상 스트림에 쓸 수 있는 쓰기 권한이 필요합니다.

다음 정책은 소스로 사용되는 Kinesis Stream에서 읽을 수 있는 권한을 부여합니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "KinesisShardDiscovery",
      "Effect": "Allow",
      "Action": "kinesis:ListShards",
      "Resource": "*"
    },
    {
      "Sid": "KinesisShardConsumption",
      "Effect": "Allow",
      "Action": [
        "kinesis:GetShardIterator",
        "kinesis:GetRecords",
        "kinesis:DescribeStream",
        "kinesis:DescribeStreamSummary",
        "kinesis:RegisterStreamConsumer",
        "kinesis:DeregisterStreamConsumer"
      ],
      "Resource": "arn:aws:kinesis:<region>:<accountId>:stream/<stream-name>"
    },
    {
      "Sid": "KinesisEfoConsumer",
      "Effect": "Allow",
      "Action": [
        "kinesis:DescribeStreamConsumer",
        "kinesis:SubscribeToShard"
      ],
      "Resource": "arn:aws:kinesis:<region>:<account>:stream/<stream-name>/consumer/*"
    }
  ]
}
```

다음 정책은 대상으로 사용되는 Kinesis Stream에 쓰기 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
         {
            "Sid": "KinesisStreamSink",
```

```
            "Effect": "Allow",
            "Action": [
                "kinesis:PutRecord",
                "kinesis:PutRecords",
                "kinesis:DescribeStreamSummary",
                "kinesis:DescribeStream"
            ],
            "Resource": "arn:aws:kinesis:<region>:<accountId>:stream/<stream-name>"
        }
    ]
}
```

애플리케이션이 암호화된 Kinesis 스트림에 액세스하는 경우 스트림과 스트림의 암호화 키에 액세스할 수 있는 추가 권한을 부여해야 합니다.

다음 정책은 암호화된 소스 스트림과 스트림의 암호화 키에 액세스할 수 있는 권한을 부여합니다.

```
{
      "Sid": "ReadEncryptedKinesisStreamSource",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
        "<inputStreamKeyArn>"
      ]
    }
    ,
```

다음 정책은 암호화된 대상 스트림과 스트림의 암호화 키에 액세스할 수 있는 권한을 부여합니다.

```
{
      "Sid": "WriteEncryptedKinesisStreamSink",
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey"
      ],
      "Resource": [
        "<outputStreamKeyArn>"
      ]
    }
```

Amazon MSK 클러스터

Amazon MSK 클러스터에 대한 액세스 권한을 부여하려면 클러스터의 VPC에 대한 액세스 권한을 부여해야 합니다. Amazon VPC에 액세스하기 위한 정책 예제는 VPC 애플리케이션 권한을 참조하세요.

아파치 플링크용 아마존 매니지드 서비스 (DataStream API) 시작하기

이 섹션에서는 Apache Flink용 관리형 서비스 및 API의 기본 개념을 소개합니다. DataStream 애플리케이션 생성 및 테스트에 사용할 수 있는 옵션에 대해 설명합니다. 또한 이 가이드의 자습서를 완료하고 첫 번째 애플리케이션을 만드는 데 필요한 도구를 설치하는 방법에 대한 지침도 제공합니다.

주제

Managed Service for Apache Flink 애플리케이션 구성 요소

Managed Service for Apache Flink 애플리케이션은 데이터를 처리하기 위해 Apache Flink 런타임을 사용하여 입력을 처리하고 출력을 생성하는 Java/Apache Maven 또는 Scala 애플리케이션을 사용합니다.

Managed Service for Apache Flink 애플리케이션에는 다음과 같은 구성 요소가 있습니다:

- 런타임 속성: 애플리케이션 코드를 다시 컴파일하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.
- 소스: 애플리케이션은 소스를 사용하여 데이터를 소비합니다. 소스 커넥터는 Kinesis 데이터 스트림, Amazon S3 버킷 등에서 데이터를 읽습니다. 자세한 설명은 소스 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 설명은 DataStream API 연산자 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 설명은 싱크 섹션을 참조하세요.

애플리케이션 코드를 생성, 컴파일 및 패키징한 후 Amazon Simple Storage Service (Amazon S3) 버킷에 코드 패키지를 업로드합니다. 그런 다음 Managed Service for Apache Flink 애플리케이션을 생성합니다. 코드 패키지 위치, Kinesis 데이터 스트림을 스트리밍 데이터 소스로 전달하고, 일반적으로 애플리케이션의 처리된 데이터를 수신하는 스트리밍 또는 파일 위치를 전달합니다.

연습 완료를 위한 필수 조건

이 가이드의 단계를 완료하려면 다음이 필요합니다.

- Java Development Kit(JDK) 버전 11. JAVA_HOME 환경 변수가 JDK 설치 위치를 가리키도록 설정합니다.
- 애플리케이션을 개발하고 컴파일하려면 개발 환경(예: Eclipse Java Neon 또는 IntelliJ Idea)을 사용하는 것이 좋습니다.
- Git 클라이언트. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다.
- Apache Maven 컴파일러 플러그인. Maven이 해당 작업 경로에 있어야 합니다. Apache Maven 설치를 테스트하려면 다음을 입력하십시오.

```
$ mvn -version
```

시작하려면 1단계: AWS 계정 설정 및 관리자 사용자 생성하기 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성하기

Managed Service for Apache Flink를 처음 사용하기 전에 다음 작업을 완료해야 합니다:

AWS 계정에 등록

AWS 계정 항목이 없으면 다음 절차에 따라 생성하십시오.

AWS 계정에 가입하려면

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

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

   AWS 계정에 가입하면 AWS 계정 루트 사용자이(가) 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스및 리소스에 액세스하는 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자

액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

가입 프로세스가 완료되면 AWS가 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자를 보안하며 AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

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

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자에 대해 다중 인증(MFA)을 활성화합니다.

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

관리 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서에서 Enabling AWS IAM Identity Center를 참조하세요.
2. IAM Identity Center에서 관리 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 Configure user access with the default IAM Identity Center 디렉터리를 참조하세요.

관리 사용자로 로그인

- IAM 자격 증명 센터 사용자로 로그인하려면 IAM 자격 증명 센터 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

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

프로그래밍 방식 액세스 권한 부여

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

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

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

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

다음 단계

2단계: AWS Command Line Interface(AWS CLI) 설정

2단계: AWS Command Line Interface(AWS CLI) 설정

이 단계에서는 Managed Service for Apache Flink와 함께 사용하도록 AWS CLI을 다운로드하고 구성합니다.

> **Note**
>
> 이 가이드의 시작하기 연습에서는 해당 계정에서 관리자 자격 증명(`adminuser`)을 사용하여 작업을 수행한다고 가정합니다.

> **Note**
>
> 이미 AWS CLI가 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수도 있습니다. 자세한 설명은 AWS Command Line Interface 사용자 가이드에서 AWS Command Line Interface 설치를 참조하세요. AWS CLI의 버전을 확인하려면 다음 명령을 실행합니다.
>
> ```
> aws --version
> ```
>
> 이 자습서의 연습에서는 다음 버전 이상의 AWS CLI가 필요합니다.
>
> ```
> aws-cli/1.16.63
> ```

AWS CLI를 설정하려면

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 AWS Command Line Interface 사용 설명서에서 다음 토픽을 참조하세요.
   - AWS Command Line Interface 설치
   - AWS CLI 구성
2. AWS CLI `config` 파일에 관리자 사용자를 위해 명명된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 명명된 프로필에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드의 명명된 프로필을 참조하세요.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   사용할 수 있는 AWS 지역 목록은 https://docs.aws.amazon.com/general/latest/gr/rande.html의 Amazon Web Services 일반 참조지역 및 엔드포인트를 참조하십시오.

   > **Note**
   >
   > 이 자습서의 예 코드 및 명령은 미국 서부(오레곤) 지역을 사용합니다. 다른 지역을 사용하려면 이 자습서의 코드 및 명령에서 지역을 사용하려는 지역으로 변경하십시오.

3. 명령 프롬프트에서 다음 help 명령을 입력하여 설정을 확인하십시오:

```
aws help
```

AWS계정을 설정하고 나면 샘플 애플리케이션을 구성하고 설정을 테스트하는 다음 연습을 시도할 수 있습니다. AWS CLI end-to-end

다음 단계

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

이 연습에서는 데이터 스트림을 소스 및 싱크로 사용하여 Managed Service for Apache Flink 애플리케이션을 만듭니다.

이 섹션은 다음 주제를 포함합니다:

- 2개의 Amazon Kinesis Data Streams 생성
- 샘플 레코드를 입력 스트림에 쓰기
- Apache Flink 스트리밍 Java 코드 다운로드 및 검사
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- 다음 단계

2개의 Amazon Kinesis Data Streams 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(`ExampleInputStream` 및 `ExampleOutputStream`)을 생성하십시오. 이 애플리케이션은 애플리케이션 소스 및 대상 스트림에 대해 이러한 스트림을 사용합니다.

Amazon Kinesis 콘솔 또는 다음 AWS CLI 명령을 사용하여 이러한 스트림을 만들 수 있습니다. 콘솔 지침은 Amazon Kinesis Data Streams 개발자 가이드의 데이터 스트림 생성 및 업데이트를 참조하십시오.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림(ExampleInputStream)을 생성하려면 다음 Amazon Kinesis create-stream AWS CLI 명령을 사용합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 애플리케이션에서 출력을 쓰는 데 사용하는 두 번째 스트림을 생성하려면 동일한 명령을 실행하여 스트림 명칭을 ExampleOutputStream으로 변경합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 stock.py이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"
```

```
    def get_data():
        return {
            'event_time': datetime.datetime.now().isoformat(),
            'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
            'price': round(random.random() * 100, 2)}


    def generate(stream_name, kinesis_client):
        while True:
            data = get_data()
            print(data)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey")


    if __name__ == '__main__':
        generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. 이 자습서의 뒷부분에서 `stock.py` 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드 다운로드 및 검사

이 예제의 Java 응용 프로그램 코드는 에서 제공됩니다 GitHub. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. `amazon-kinesis-data-analytics-java-examples/GettingStarted` 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- 프로젝트 객체 모델(pom.xml) 파일에는 Managed Service for Apache Flink 라이브러리를 비롯한 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- `BasicStreamingJob.java` 파일에는 애플리케이션의 기능을 정의하는 `main` 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 애플리케이션은 `StreamExecutionEnvironment` 객체를 사용하여 외부 리소스에 액세스하기 위한 소스 및 싱크 커넥터를 생성합니다.
- 애플리케이션은 정적 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 동적 애플리케이션 속성을 사용하려면 `createSourceFromApplicationProperties` 및 `createSinkFromApplicationProperties` 메서드를 사용하여 커넥터를 생성합니다. 이 메서드는 애플리케이션의 속성을 읽어 커넥터를 구성합니다.

  이러한 런타임 속성에 대한 자세한 설명은 런타임 속성 섹션을 참조하십시오.

애플리케이션 코드 컴파일

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 설명은 연습 완료를 위한 필수 조건 섹션을 참조하십시오.

애플리케이션 코드를 컴파일하려면

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. 다음 두 가지 방법 중 하나로 코드를 컴파일하고 패키징할 수 있습니다:

   - 명령행 Maven 도구 사용. `pom.xml` 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

     ```
     mvn package -Dflink.version=1.15.3
     ```

   - 귀하의 개발 환경 사용. 자세한 설명은 해당 개발 환경 설명서를 참조하십시오.

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. AWS CLI를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형(JAR 또는 ZIP)을 지정합니다.

2. 컴파일하는 동안 오류가 발생하면 `JAVA_HOME` 환경 변수가 올바르게 설정되어 있는지 확인하십시오.

애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 Amazon Simple Storage Service(Amazon S3) 버킷을 만들고 애플리케이션 코드를 업로드합니다.

애플리케이션 코드 업로드하기

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 **`ka-app-code-<username>`**을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 생성을 선택합니다.
7. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다. 다음을 선택합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔이나 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 생성하고 실행할 수 있습니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 생성하면 사용자 AWS Identity and Access Management (IAM) 및 Amazon CloudWatch Logs 리소스가 자동으로 생성됩니다. AWS CLI를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 만듭니다.

주제

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 설명에 **`My java test app`**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
```

```
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
```

```
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 ID에 **ProducerConfigProperties**를 입력합니다.
5. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

6. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
7. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
8. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`

- 로그 스트림: `kinesis-analytics-log-stream`

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

MyApplication페이지에서 [Stop] 을 선택합니다. 작업을 확인합니다.

애플리케이션 업데이트

콘솔을 사용하여 애플리케이션 속성, 모니터링 설정, 애플리케이션 JAR의 위치 또는 파일 명칭과 같은 애플리케이션 설정을 업데이트할 수 있습니다. 애플리케이션 코드를 업데이트해야 하는 경우 Amazon S3 버킷에서 애플리케이션 JAR을 다시 로드할 수도 있습니다.

MyApplication페이지에서 구성을 선택합니다. 애플리케이션 설정을 업데이트하고 업데이트를 선택합니다.

애플리케이션 생성 및 실행 (AWS CLI)

이 섹션에서는 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. Managed Service for Apache Flink는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Managed Service for Apache Flink를 생성하고 Managed Service for Apache Flink와 상호 작용합니다.

권한 정책 생성

> **Note**
>
> 애플리케이션에 대한 권한 정책 및 역할을 생성해야 합니다. 이러한 IAM 리소스를 생성하지 않으면 애플리케이션은 해당 데이터 및 로그 스트림에 액세스할 수 없습니다.

먼저 소스 스트림에서 두 개의 명령문, 즉 `read` 작업에 대한 권한을 부여하는 명령문과 싱크 스트림에서 `write` 작업에 대한 권한을 부여하는 명령문이 있는 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 AKReadSourceStreamWriteSinkStream 권한 정책을 생성합니다. *username*을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(*012345678901*)를 사용자의 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

> **Note**
>
> AWS SDK for Java를 사용하여 다른 Amazon 서비스에 액세스할 수 있습니다. Managed Service for Apache Flink는 SDK에 필요한 자격 증명을 애플리케이션과 연결된 서비스 실행 IAM 역할의 자격 증명으로 자동 설정합니다. 추가 단계는 필요 없습니다.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할, 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다. 사용 사례 선택에서 Kinesis Analytics를 선택합니다.

   다음: 권한을 선택합니다.
4. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
5. 역할 생성 페이지에서 역할 이름으로 **`MF-stream-rw-role`**을 입력합니다. Create role(역할 생성)을 선택합니다.

   `MF-stream-rw-role`이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.
6. 역할에 권한 정책을 연결합니다.

> **Note**
>
> 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계 the section called “권한 정책 생성”에서 생성한 정책을 연결합니다.

a. 요약 페이지에서 권한 탭을 선택합니다.

b. 정책 연결을 선택합니다.

c. 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.

d. AK ReadSourceStreamWriteSinkStream 정책을 선택하고 Attach policy (Attach policy) 를 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

Managed Service for Apache Flink 애플리케이션 생성

1. 다음 JSON 코드를 `create_request.json`이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(*username*)를 이전 섹션에서 선택한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(*012345678901*)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "aws-kinesis-analytics-java-apps-1.0.jar"
                }
```

```
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

2. 위의 요청과 함께 CreateApplication 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "test",
```

```
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```
{
   "ApplicationName": "test"
}
```

2. 다음 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “로깅 설정”.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 UpdateApplication 작업을 사용하여 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
   }
}
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationAWS CLI 작업을 사용합니다.

> **Note**
>
> 파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하십시오.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 the section called “2개의 Amazon Kinesis Data Streams 생성” 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

다음 단계

4단계: AWS 리소스 정리

4단계: AWS 리소스 정리

이 섹션에는 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

다음 단계

5단계: 다음 절차

5단계: 다음 절차

이제 Managed Service for Apache Flink 애플리케이션을 만들고 실행했으므로 고급 Managed Service for Apache Flink 솔루션에 대해서는 다음 리소스를 참조하십시오.

- AWS Amazon Kinesis용 스트리밍 데이터 솔루션: AWS Amazon Kinesis용 스트리밍 데이터 솔루션은 스트리밍 데이터를 쉽게 캡처, 저장, 처리 및 전송하는 데 필요한 AWS 서비스를 자동으로 구성합니다. 이 솔루션은 스트리밍 데이터 사용 사례를 해결하기 위한 다양한 옵션을 제공합니다. Apache Flink용 관리형 서비스 옵션은 시뮬레이션된 뉴욕 택시 데이터에 대한 분석 작업을 실행하는 실제 애플리케이션을 보여주는 end-to-end 스트리밍 ETL 예제를 제공합니다. 이 솔루션은 IAM 역할 및 정책, 대시보드, 경보 등 필요한 모든 AWS 리소스를 설정합니다. CloudWatch CloudWatch

- AWSAmazon MSK용 스트리밍 데이터 솔루션: Amazon MSK용 AWS 스트리밍 데이터 솔루션은 생산자, 스트리밍 스토리지, 소비자 및 목적지를 통해 데이터가 흐르는 AWS CloudFormation 템플릿을 제공합니다.
- Apache Flink 및 Apache Kafka를 사용한 클릭스트림 랩: 스트리밍 스토리지로는 Apache Kafka용 Amazon Managed Streaming을 사용하고 스트림 처리에는 Managed Service for Apache Flink 애플리케이션을 사용하는 클릭스트림 사용 사례를 위한 종합적인 실습입니다.
- Apache Flink용 Amazon Managed Service 워크숍: 이 워크숍에서는 end-to-end 스트리밍 데이터를 거의 실시간으로 수집, 분석 및 시각화하는 스트리밍 아키텍처를 구축합니다. 여러분은 뉴욕시에 있는 택시 회사의 운영을 개선하기 위해 나섰습니다. 뉴욕시에 있는 택시의 원격 측정 데이터를 거의 실시간으로 분석하여 차량 운영을 최적화합니다.
- Managed Service for Apache Flink: 예제: 이 개발자 가이드의 이 섹션에서는 Managed Service for Apache Flink에서 애플리케이션을 만들고 사용하는 예를 제공합니다. 여기에는 Apache Flink 애플리케이션용 관리 서비스를 생성하고 결과를 테스트하는 데 도움이 되는 예제 코드와 step-by-step 지침이 포함되어 있습니다.
- Flink 알아보기: 실습 교육: 확장 가능한 스트리밍 ETL, 분석 및 이벤트 기반 애플리케이션 작성을 시작하는 데 도움이 되는 공식 Apache Flink 입문 교육입니다.

> **Note**
>
> Managed Service for Apache Flink는 이 교육에 사용된 Apache Flink 버전(1.12)을 지원하지 않는다는 점에 유의하십시오. 아파치 플링크용 플링크 매니지드 서비스에서 Flink 1.15.2를 사용할 수 있습니다.

Amazon Managed Service for Apache Flink (표 API) 시작하기

이 섹션에서는 Managed Service for Apache Flink와 표 API의 기본 개념을 소개합니다. 애플리케이션 생성 및 테스트에 사용할 수 있는 옵션에 대해 설명합니다. 또한 이 가이드의 자습서를 완료하고 첫 번째 애플리케이션을 만드는 데 필요한 도구를 설치하는 방법에 대한 지침도 제공합니다.

주제

Managed Service for Apache Flink 애플리케이션 구성 요소

Managed Service for Apache Flink 애플리케이션은 데이터를 처리하기 위해 Apache Flink 런타임을 사용하여 입력을 처리하고 출력을 생성하는 Java/Apache Maven 또는 Scala 애플리케이션을 사용합니다.

Managed Service for Apache Flink 애플리케이션에는 다음 구성 요소가 있습니다:

- 런타임 속성: 애플리케이션 코드를 다시 컴파일하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.
- 표 소스: 애플리케이션은 소스를 사용하여 데이터를 소비합니다. 소스 커넥터는 Kinesis 데이터 스트림, Amazon MSK 주제 등으로부터 데이터를 읽습니다. 자세한 설명은 표 API 소스 섹션을 참조하세요.
- 함수: 애플리케이션은 하나 이상의 함수를 사용하여 데이터를 처리합니다. 함수는 데이터를 변환, 강화 또는 집계할 수 있습니다.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림, Amazon MSK 주제, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 설명은 표 API 싱크 섹션을 참조하세요.

애플리케이션 코드를 생성, 컴파일 및 패키징한 후 Amazon Simple Storage Service (Amazon S3) 버킷에 코드 패키지를 업로드합니다. 그런 다음 Managed Service for Apache Flink 애플리케이션을 생성합니다. 코드 패키지 위치, Amazon MSK 주제를 스트리밍 데이터 소스로 전달하고, 일반적으로 애플리케이션의 처리된 데이터를 수신하는 스트리밍 또는 파일 위치를 전달합니다.

사전 조건

이 자습서를 시작하기 전에 아파치 플링크용 아마존 매니지드 서비스 (DataStream API) 시작하기의 첫 두 단계를 완료하십시오:

- 1단계: AWS 계정 설정 및 관리자 사용자 생성하기
- 2단계: AWS Command Line Interface(AWS CLI) 설정

시작하려면 애플리케이션 만들기섹션을 참조하세요.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

이 연습에서는 Amazon MSK 주제를 소스로 사용하고 Amazon S3 버킷을 싱크로 사용하여 Managed Service for Apache Flink 애플리케이션을 생성합니다.

이 섹션은 다음 주제를 포함합니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- Apache Flink 스트리밍 Java 코드 다운로드 및 검사
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- 다음 단계

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink를 생성하기 전에 먼저 다음과 같은 종속 리소스를 생성해야 합니다:

- Amazon VPC와 Amazon MSK 클러스터 기반 Virtual Private Cloud (VPC)

- 애플리케이션 코드와 출력을 저장할 Amazon S3 버킷(ka-app-code-*<username>*)

VPC 및 Amazon MSK 클러스터 생성

Managed Service for Apache Flink 애플리케이션에서 액세스할 VPC 및 Amazon MSK 클러스터를 생성하려면 Amazon MSK 사용 시작하기 자습서를 따르십시오.

자습서를 완료할 때는 다음 사항에 유의하십시오:

- 클러스터의 부트스트랩 서버 목록을 기록합니다. *ClusterArn*을 MSK 클러스터의 Amazon 리소스 이름(ARN)으로 대체하는 다음 명령을 사용하여 부트스트랩 서버 목록을 가져올 수 있습니다:

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
{...
    "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094"
}
```

- 자습서의 단계를 따를 때는 코드, 명령 및 콘솔 항목에서 선택한 AWS 리전을 사용해야 합니다.

Amazon S3 버킷 생성

콘솔을 사용하여 Amazon S3 버킷을 생성할 수 있습니다. 리소스 생성에 대한 지침은 다음 주제를 참조하세요:

- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

기타 리소스

애플리케이션을 생성할 때 Apache Flink용 관리형 서비스는 다음과 같은 Amazon CloudWatch 리소스를 생성합니다 (아직 존재하지 않는 경우).

- /AWS/KinesisAnalytics-java/MyApplication라는 로그 그룹.
- kinesis-analytics-log-stream라는 로그 스트림.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 Amazon MSK 주제에 씁니다.

1. Amazon MSK 사용 시작하기 자습서의 4단계: 클라이언트 머신 생성에서 생성한 클라이언트 인스턴스에 연결합니다.
2. Python3, Pip 그리고 Kafka Python 라이브러리를 설치하세요:

```
$ sudo yum install python37
$ curl -O https://bootstrap.pypa.io/get-pip.py
$ python3 get-pip.py --user
$ pip install kafka-python
```

3. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다: `BROKERS` 값을 이전에 기록한 부트스트랩 브로커 목록으로 바꾸십시오.

```
import datetime
    import json
    import random
    import boto3

    STREAM_NAME = "ExampleInputStream"


    def get_data():
        return {
            'event_time': datetime.datetime.now().isoformat(),
            'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
            'price': round(random.random() * 100, 2)}


    def generate(stream_name, kinesis_client):
        while True:
            data = get_data()
            print(data)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey")
```

```
    if __name__ == '__main__':
        generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

4. 이 자습서의 뒷부분에서 stock.py 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python3 stock.py
```

Apache Flink 스트리밍 Java 코드 다운로드 및 검사

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub

Java 애플리케이션 코드를 다운로드하려면

1. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/GettingStartedTable 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- 프로젝트 객체 모델(pom.xml) 파일에는 Managed Service for Apache Flink 라이브러리를 비롯한 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- StreamingJob.java 파일에는 애플리케이션의 기능을 정의하는 main 메서드가 들어 있습니다.
- 애플리케이션은 FlinkKafkaConsumer를 사용하여 Amazon MSK 주제에서 읽습니다. 다음 코드 조각은 FlinkKafkaConsumer 객체를 생성합니다:

```
final FlinkKafkaConsumer<StockRecord> consumer = new
 FlinkKafkaConsumer<StockRecord>(kafkaTopic, new KafkaEventDeserializationSchema(),
 kafkaProps);
```

- 애플리케이션은 StreamExecutionEnvironment 및 TableEnvironment 객체를 사용하여 외부 리소스에 액세스하기 위한 소스 및 싱크 커넥터를 생성합니다.
- 애플리케이션은 동적 애플리케이션 속성을 사용하여 소스 및 싱크 커넥터를 생성하므로 코드를 재컴파일하지 않고도 애플리케이션 파라미터(예: S3 버킷)를 지정할 수 있습니다.

```
//read the parameters from the Managed Service for Apache Flink environment
Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
Properties flinkProperties = null;

String kafkaTopic = parameter.get("kafka-topic", "AWSKafkaTutorialTopic");
String brokers = parameter.get("brokers", "");
String s3Path = parameter.get("s3Path", "");

if (applicationProperties != null) {
    flinkProperties = applicationProperties.get("FlinkApplicationProperties");
}

if (flinkProperties != null) {
    kafkaTopic = flinkProperties.get("kafka-topic").toString();
    brokers = flinkProperties.get("brokers").toString();
    s3Path = flinkProperties.get("s3Path").toString();
}
```

이러한 런타임 속성에 대한 자세한 설명은 런타임 속성 섹션을 참조하십시오.

> **Note**
>
> 애플리케이션을 구축할 때는 Amazon MSK 클러스터와 동일한 지역에서 Managed Service for Apache Flink 애플리케이션을 만들고 실행하는 것이 좋습니다. 이는 Flink Kafka 커넥터가 기본적으로 지연 시간이 짧은 환경에 최적화되어 있기 때문입니다. 지역 간 Kafka 클러스터를 사용해야 하는 경우 2097152와 같이 `receive.buffer.byte`의 구성 값을 높이는 것이 좋습니다. 자세한 설명은 맞춤 MSK 구성을 참조하십시오.

애플리케이션 코드 컴파일

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 설명은 연습 완료를 위한 필수 조건 섹션을 참조하십시오.

애플리케이션 코드를 컴파일하려면

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. 다음 두 가지 방법 중 하나로 코드를 컴파일하고 패키징할 수 있습니다:

   - 명령행 Maven 도구 사용. `pom.xml` 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

     ```
     mvn package -Dflink.version=1.15.3
     ```

   - 귀하의 개발 환경 사용. 자세한 설명은 해당 개발 환경 설명서를 참조하십시오.

     > **Note**
     >
     > 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

   패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. AWS CLI를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형(JAR 또는 ZIP)을 지정합니다.
2. 컴파일하는 동안 오류가 발생하면 `JAVA_HOME` 환경 변수가 올바르게 설정되어 있는지 확인하십시오.

애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 Amazon S3 버킷을 만들고 애플리케이션 코드를 업로드합니다.

애플리케이션 코드 업로드하기

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 **`ka-app-code-<username>`**을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.

6. 버킷 생성을 선택합니다.
7. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다. 다음을 선택합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 설명에 **`My java test app`**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두십시오.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*

- 역할: kinesisanalytics-*MyApplication*-*us-west-2*

IAM 정책 편집

IAM 정책을 편집하여 Amazon S3 버킷에 액세스할 수 있는 권한을 추가합니다.

IAM 정책을 편집하여 S3 버킷 권한을 추가하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Sid": "S3",
        "Effect": "Allow",
        "Action": [
            "s3:Abort*",
            "s3:DeleteObject*",
            "s3:GetObject*",
            "s3:GetBucket*",
            "s3:List*",
            "s3:ListBucket",
            "s3:PutObject"
        ],
        "Resource": [
            "arn:aws:s3:::ka-app-code-<username>",
            "arn:aws:s3:::ka-app-code-<username>/*"
        ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
```

```
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        }
    ]
}
```

애플리케이션 구성

애플리케이션을 구성하려면 다음 절차를 사용합니다.

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.

3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 생성을 선택합니다.
5. 다음을 입력합니다:

그룹 ID	키	값
FlinkApplicationProperties	**kafka-topic**	**AWSKafkaTutorialTopic**
FlinkApplicationProperties	**brokers**	***Your Amazon MSK cluster's Bootstrap Brokers list***
FlinkApplicationProperties	**s3Path**	**ka-app-code-** ***<username>***
FlinkApplicationProperties	**security.protocol**	**SSL**
FlinkApplicationProperties	**ssl.truststore.location**	**/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts**
FlinkApplicationProperties	**ssl.truststore.password**	**changeit**

6. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
7. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
8. Virtual Private Cloud(VPC) 섹션에서 Amazon MSK 클러스터 기반 VPC 구성을 선택합니다. 선택합니다 AWSKafkaTutorialCluster.
9. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

애플리케이션 실행

애플리케이션을 실행하려면 다음 절차를 따르세요.

애플리케이션을 실행하려면

1. MyApplication페이지에서 [Run] 을 선택합니다. 작업을 확인합니다.
2. 애플리케이션이 실행 중이면 페이지를 새로 고칩니다. 콘솔에 애플리케이션 그래프가 표시됩니다.
3. Amazon EC2 클라이언트에서 이전에 만든 Python 스크립트를 실행하여 애플리케이션이 처리할 수 있도록 Amazon MSK 클러스터에 레코드를 기록합니다.

```
$ python3 stock.py
```

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 [Stop] 을 선택합니다. 작업을 확인합니다.

다음 단계

AWS 리소스 정리

AWS 리소스 정리

이 섹션에는 시작하기(표 API) 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Amazon MSK 클러스터 삭제

Managed Service for Apache Flink 애플리케이션 삭제

애플리케이션을 삭제하려면 다음 절차를 따르세요.

애플리케이션을 삭제하려면

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확정합니다.

Amazon MSK 클러스터 삭제

Amazon MSK 클러스터를 삭제하려면 Apache Kafka용 Amazon Managed Streaming 개발자 가이드의 8단계: Amazon MSK 클러스터 삭제를 따르십시오.

VPC 삭제

Amazon VPC를 삭제하려면 다음을 수행합니다:

- Amazon VPC 콘솔을 엽니다.
- VPC를 선택합니다.
- 작업에서 VPC 삭제를 선택합니다.

Amazon S3 객체 및 버킷 삭제

다음 절차에 따라 S3 객체 및 버킷을 삭제합니다.

S3 객체 및 버킷을 삭제하려면

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.

2. ka-app-code- <username>버킷을 선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

다음 절차에 따라 IAM 리소스를 삭제합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

CloudWatch 리소스를 삭제하려면 다음 절차를 따르십시오.

CloudWatch 리소스를 삭제하려면

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

다음 단계

다음 단계

다음 단계

이제 표 API를 사용하는 Managed Service for Apache Flink 애플리케이션을 만들고 실행했으므로 아파치 플링크용 아마존 매니지드 서비스 (DataStream API) 시작하기의 5단계: 다음 절차을 참조하십시오.

Amazon Managed Service for Apache Flink for Python 시작하기

이 섹션에서는 Python과 Table API를 사용하는 Amazon Managed Service for Apache Flink의 기본 개념을 소개합니다. 애플리케이션 생성 및 테스트에 사용할 수 있는 옵션에 대해 설명합니다. 또한 이 가이드의 자습서를 완료하고 첫 번째 애플리케이션을 만드는 데 필요한 도구를 설치하는 방법에 대한 지침도 제공합니다.

주제

> **Note**
>
> Apple Silicon 칩이 탑재된 새 Mac에서 Python Flink 응용 프로그램을 개발하는 경우 1.15의 PyFlink Python 종속성과 관련된 몇 가지 알려진 문제가 발생할 수 있습니다. 이 경우 Docker에서 Python 인터프리터를 실행하는 것이 좋습니다. 자세한 step-by-step 지침은 Apple Silicon Mac에서의 PyFlink 1.15 개발을 참조하십시오.

Pyflink 시작하기 - Apache를 위한 Python 인터프리터 | Amazon 웹 서비스

시작하기 전에 다음 동영상을 시청하는 것이 좋습니다:

Pyflink 시작하기 - Apache를 위한 Python 인터프리터 | Amazon 웹 서비스

Managed Service for Apache Flink 애플리케이션 구성 요소

Managed Service for Apache Flink 애플리케이션은 데이터를 처리하기 위해 Apache Flink 런타임을 사용하여 입력을 처리하고 출력을 생성하는 Python 애플리케이션을 사용합니다.

Managed Service for Apache Flink 애플리케이션에는 다음 구성 요소가 있습니다:

- 런타임 속성: 애플리케이션 코드를 다시 컴파일하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.
- 표 소스: 애플리케이션은 소스를 사용하여 데이터를 소비합니다. 소스 커넥터는 Kinesis 데이터 스트림, Amazon MSK 주제 등으로부터 데이터를 읽습니다. 자세한 설명은 표 API 소스 섹션을 참조하세요.
- 함수: 애플리케이션은 하나 이상의 함수를 사용하여 데이터를 처리합니다. 함수는 데이터를 변환, 강화 또는 집계할 수 있습니다.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림, Amazon MSK 주제, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 설명은 표 API 싱크 섹션을 참조하세요.

애플리케이션 코드를 생성 및 패키징한 후 Amazon S3 버킷에 코드 패키지를 업로드합니다. 그런 다음 Managed Service for Apache Flink 애플리케이션을 생성합니다. 코드 패키지 위치, 스트리밍 데이터 소스, 그리고 일반적으로 애플리케이션의 처리된 데이터를 수신하는 스트리밍 또는 파일 위치를 전달합니다.

사전 조건

이 자습서를 시작하기 전에 아파치 플링크용 아마존 매니지드 서비스 (DataStream API) 시작하기의 첫 두 단계를 완료하십시오:

- 1단계: AWS 계정 설정 및 관리자 사용자 생성하기
- 2단계: AWS Command Line Interface(AWS CLI) 설정

시작하려면 애플리케이션 만들기 섹션을 참조하세요.

Managed Service for Apache Flink Python 애플리케이션의 생성 및 실행

이 예에서는 Kinesis 스트림을 소스 및 싱크로 사용하여 Managed Service for Apache Flink Python 애플리케이션을 생성합니다.

이 섹션은 다음 주제를 포함합니다:

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink를 생성하기 전에 먼저 다음과 같은 종속 리소스를 생성해야 합니다:

- 입력 및 출력을 위한 Kinesis 스트림 2개.
- 애플리케이션 코드와 출력을 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

2개의 Kinesis 스트림 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(`ExampleInputStream` 및 `ExampleOutputStream`)을 생성하십시오. 이 애플리케이션은 애플리케이션 소스 및 대상 스트림에 대해 이러한 스트림을 사용합니다.

Amazon Kinesis 콘솔 또는 다음 AWS CLI 명령을 사용하여 이러한 스트림을 만들 수 있습니다. 콘솔 지침은 Amazon Kinesis Data Streams 개발자 가이드의 데이터 스트림 생성 및 업데이트를 참조하십시오.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림(`ExampleInputStream`)을 생성하려면 다음 Amazon Kinesis `create-stream` AWS CLI 명령을 사용합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 애플리케이션에서 출력을 쓰는 데 사용하는 두 번째 스트림을 생성하려면 동일한 명령을 실행하여 스트림 명칭을 ExampleOutputStream으로 변경합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

Amazon S3 버킷 생성

콘솔을 사용하여 Amazon S3 버킷을 생성할 수 있습니다. 리소스 생성에 대한 지침은 다음 주제를 참조하세요:

- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

기타 리소스

애플리케이션을 생성할 때 Apache Flink용 관리형 서비스는 다음과 같은 Amazon CloudWatch 리소스를 생성합니다 (아직 존재하지 않는 경우).

- /AWS/KinesisAnalytics-java/MyApplication라는 로그 그룹.
- kinesis-analytics-log-stream라는 로그 스트림.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

> **Note**
>
> 이 섹션의 Python 스크립트는 AWS CLI를 사용합니다. 사용자의 계정 보안 인증과 기본 리전을 사용하도록 사용자의 AWS CLI을(를) 구성해야 합니다. 다음 명령을 입력하여 사용자의 AWS CLI를 구성합니다.
>
> ```
> aws configure
> ```

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        'event_time': datetime.datetime.now().isoformat(),
        'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
        'price': round(random.random() * 100, 2)}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(data),
            PartitionKey="partitionkey")


if __name__ == '__main__':
    generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. `stock.py` 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

Apache Flink 스트리밍 Python 코드 생성 및 검사

이 예제의 Python 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. `amazon-kinesis-data-analytics-java-examples/python/GettingStarted` 디렉터리로 이동합니다.

애플리케이션 코드는 `getting_started.py` 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- 애플리케이션은 Kinesis 테이블 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 `create_table` 함수를 호출하여 Kinesis 테이블 소스를 생성합니다.

```
table_env.execute_sql(
        create_table(output_table_name, output_stream, output_region)
```

이 `create_table` 함수는 SQL 명령을 사용하여 스트리밍 소스가 지원하는 테이블을 생성합니다.

```
def create_table(table_name, stream_name, region, stream_initpos = None):
    init_pos = "\n'scan.stream.initpos' = '{0}',".format(stream_initpos) if
 stream_initpos is not None else ''

    return """ CREATE TABLE {0} (
                ticker VARCHAR(6),
                price DOUBLE,
                event_time TIMESTAMP(3),
                WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
              )
```

```
            PARTITIONED BY (ticker)
            WITH (
              'connector' = 'kinesis',
              'stream' = '{1}',
              'aws.region' = '{2}',{3}
              'format' = 'json',
              'json.timestamp-format.standard' = 'ISO-8601'
            ) """.format(table_name, stream_name, region, init_pos)
}
```

- 애플리케이션은 2개의 표를 만든 다음 한 표의 내용을 다른 표에 씁니다.

```
    # 2. Creates a source table from a Kinesis Data Stream
    table_env.execute_sql(
        create_table(input_table_name, input_stream, input_region)
    )

    # 3. Creates a sink table writing to a Kinesis Data Stream
    table_env.execute_sql(
        create_table(output_table_name, output_stream, output_region, stream_initpos)
    )

    # 4. Inserts the source table data into the sink table
    table_result = table_env.execute_sql("INSERT INTO {0} SELECT * FROM {1}"
                         .format(output_table_name, input_table_name))
```

- 응용 프로그램은 flink - sql-connector-kinesis _2.12/1.15.2 파일의 플링크 커넥터를 사용합니다.

Python 앱에 제3자 종속성 추가

제3자 python 패키지(예: boto3)를 사용할 때는 해당 패키지의 전이 종속성과 이러한 종속성을 목적지으로 하는 데 필요한 속성을 추가해야 합니다. 상위 수준에서 PyPi 종속성의 경우 python 환경 `site-packages` 폴더 내에 있는 파일 및 폴더를 복사하여 다음과 같은 디렉터리 구조를 만들 수 있습니다.

```
PythonPackages
#   README.md
#   python-packages.py
#
####my_deps
       ####boto3
```

```
        #   #   session.py
        #   #   utils.py
        #   #   ...
        #
        ####botocore
        #   #   args.py
        #   #   auth.py
        #    ...
        ####mynonpypimodule
        #   #   mymodulefile1.py
        #   #   mymodulefile2.py
             ...
####lib
#    # flink-sql-connector-kinesis-1.15.2.jar
#    # ...
...
```

boto3를 타사 종속 항목으로 추가하려면:

1. 필요한 종속성을 사용하여 로컬 시스템에 독립형 Python 환경(예: conda)을 생성합니다.
2. 해당 환경의 `site_packages` 폴더에 있는 초기 패키지 목록을 확인하십시오.
3. `pip-install` 앱에 필요한 모든 종속성.
4. 위의 3단계 이후에 `site_packages` 폴더에 추가된 패키지를 주목하십시오. 귀하의 패키지(`my_deps` 폴더 아래의)에 포함해야 하는 폴더는 위와 같이 구성됩니다. 이렇게 하면 2단계와 3단계 사이의 패키지 차이를 캡처하여 애플리케이션에 적합한 패키지 종속성을 식별할 수 있습니다.
5. 속성에 대한 아래 설명과 같이 `pyFiles` `kinesis.analytics.flink.run.options` 속성 그룹의 속성에 `jarfiles` 대한 인수(argument)로 `my_deps/`을 제공하십시오. Flink에서는 add_python_file 함수를 사용하여 Python 종속성을 지정할 수도 있지만, 둘 중 하나만 지정하면 되고 둘 다 지정해야 하는 것은 아니란 점을 명심해야 합니다.

> **Note**
>
> 폴더 `my_deps`을 명명할 필요는 없습니다. 중요한 부분은 `pyFiles` 또는 `add_python_file` 중 하나를 사용하여 종속성을 등록하는 것입니다. 예는 pyFlink 내에서 boto3를 사용하는 방법에서 찾을 수 있습니다.

Apache Flink 스트리밍 Python 코드 업로드

이 섹션에서는 Amazon S3 버킷을 만들고 애플리케이션 코드를 업로드합니다.

콘솔을 사용하여 애플리케이션 코드를 업로드하려면:

1. 선호하는 압축 응용 프로그램을 사용하여 `getting-started.py` 및 https://mvnrepository.com/artifact/org.apache.flink/flink - sql-connector-kinesis _2.12/1.15.2 파일을 압축합니다. 아카이브 이름은 `myapp.zip`라고 짓습니다. 아카이브에 외부 폴더를 포함시키는 경우, 경로에 구성 파일의 코드와 함께 이 외부 폴더를 포함시켜야 합니다: `GettingStarted/getting-started.py`.
2. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
3. 버킷 만들기를 선택합니다.
4. 버킷 명칭 필드에 **`ka-app-code-<username>`**을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
5. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
7. 버킷 생성을 선택합니다.
8. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
9. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `myapp.zip` 파일로 이동합니다. 다음을 선택합니다.
10. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

AWS CLI을 사용하여 애플리케이션 코드를 업로드

> **Note**
>
> `myapp.zip` 아카이브를 만들 때는 파인더 (macOS) 또는 Windows Explorer(Windows)의 압축 기능을 사용하지 마십시오. 이로 인해 애플리케이션 코드가 유효하지 않게 될 수 있습니다.

1. 선호하는 압축 애플리케이션을 사용하여 `streaming-file-sink.py` 및 https://mvnrepository.com/artifact/org.apache.flink/flink - sql-connector-kinesis _2.12/1.15.2 파일을 압축합니다.

> **Note**
>
> Finder(macOS) 또는 Windows Exploer(Windows)의 압축 기능을 사용하여 myapp.zip 아카이브를 만들지 마십시오. 이로 인해 애플리케이션 코드가 유효하지 않게 될 수 있습니다.

2. 선호하는 압축 응용 프로그램을 사용하여 및 https://mvnrepository.com/artifact/org.apache.flink/ /1.15.2 파일을 압축합니다. `getting-started.py` flink-sql-connector-kinesis 아카이브 이름은 `myapp.zip`라고 짓습니다. 아카이브에 외부 폴더를 포함시키는 경우, 경로에 구성 파일의 코드와 함께 이 외부 폴더를 포함시켜야 합니다: `GettingStarted/getting-started.py`.
3. 다음 명령을 실행합니다:

```
$ aws s3 --region aws region cp myapp.zip s3://ka-app-code-<username>
```

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 설명에 **`My java test app`**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두십시오.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

애플리케이션을 구성하려면 다음 절차를 사용합니다.

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **myapp.zip**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 추가를 선택합니다.
5. 다음을 입력합니다:

그룹 ID	키	값
consumer.config.0	**input.stream.name**	**ExampleInputStream**
consumer.config.0	**aws.region**	**us-west-2**
consumer.config.0	**scan.stream.initpos**	**LATEST**

   저장을 선택합니다.
6. 속성에서 그룹 추가를 다시 선택합니다.

7. 다음을 입력합니다:

그룹 ID	키	값
producer.config.0	**output.stream.name**	**ExampleOutputStream**
producer.config.0	**aws.region**	**us-west-2**
producer.config.0	**shard.count**	**1**

8. 속성에서 그룹 추가를 다시 선택합니다. 역할 이름에 **kinesis.analytics.flink.run.options**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 설명은 코드 파일 지정 섹션을 참조하세요.
9. 다음을 입력합니다:

그룹 ID	키	값
kinesis.analytics.flink.run.options	**python**	**getting-started.py**
kinesis.analytics.flink.run.options	**jarfile**	**flink-sql-connector-kinesis-1.15.2.jar**

10. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
11. CloudWatch 로깅하려면 활성화 확인란을 선택합니다.
12. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

IAM 정책 편집

IAM 정책을 편집하여 Amazon S3 버킷에 액세스할 수 있는 권한을 추가합니다.

IAM 정책을 편집하여 S3 버킷 권한을 추가하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/myapp.zip"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
```

```
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 [Stop] 을 선택합니다. 작업을 확인합니다.

다음 단계

AWS 리소스 정리

AWS 리소스 정리

이 섹션에는 시작하기 (Python) 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

애플리케이션을 삭제하려면 다음 절차를 따르세요.

애플리케이션을 삭제하려면

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확정합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

다음 절차에 따라 S3 객체 및 버킷을 삭제합니다.

S3 객체 및 버킷을 삭제하려면

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- <username>버킷을 선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

다음 절차에 따라 IAM 리소스를 삭제합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

CloudWatch 리소스를 삭제하려면 다음 절차를 따르십시오.

CloudWatch 리소스를 삭제하려면

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

시작하기 (Scala)

> **Note**
>
> Flink 버전 1.15부터 Scala는 무료입니다. 이제 애플리케이션은 모든 Scala 버전에서 Java API를 사용할 수 있습니다. Flink는 여전히 내부적으로 몇 가지 주요 구성 요소에서 Scala를 사용하지만 사용자 코드 클래스 로더에 Scala를 노출하지는 않습니다. 따라서 사용자는 자신의 jar-acrchive에 Scala 종속성을 추가해야 합니다.
> Flink 1.15의 Scala 변경 사항에 대한 자세한 내용은 Scala Free in One Fifteen을 참조하세요.

이 예에서는 Kinesis 스트림을 소스 및 싱크로 사용하여 Scala를 위한 Managed Service for Apache Flink 애플리케이션을 생성합니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- 애플리케이션 코드 다운로드 및 검토
- 애플리케이션 코드 컴파일 및 업로드
- 애플리케이션 생성 및 실행(콘솔)
- 애플리케이션 생성 및 실행(CLI)
- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 다음과 같은 종속 리소스를 생성해야 합니다.

- 입력 및 출력을 위한 Kinesis 스트림 2개.
- 애플리케이션 코드를 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **ExampleInputStream** 및 **ExampleOutputStream**에 명칭을 지정합니다.

  데이터 스트림 (AWS CLI)을 생성하려면

  - 첫 번째 스트림(ExampleInputStream)을 생성하려면 다음 Amazon Kinesis AWS CLI 명령을 사용합니다.

    ```
    aws kinesis create-stream \
        --stream-name ExampleInputStream \
        --shard-count 1 \
        --region us-west-2 \
        --profile adminuser
    ```

  - 애플리케이션에서 출력을 쓰는 데 사용하는 두 번째 스트림을 생성하려면 동일한 명령을 실행하여 스트림 명칭을 ExampleOutputStream으로 변경합니다.

    ```
    aws kinesis create-stream \
        --stream-name ExampleOutputStream \
        --shard-count 1 \
        --region us-west-2 \
        --profile adminuser
    ```

- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

기타 리소스

애플리케이션을 생성할 때 Managed Service for Apache Flink는 다음과 같은 Amazon CloudWatch 리소스를 생성합니다(아직 존재하지 않는 경우라면):

- /AWS/KinesisAnalytics-java/MyApplication라는 로그 그룹.
- kinesis-analytics-log-stream라는 로그 스트림.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

> **Note**
>
> 이 섹션의 Python 스크립트는 AWS CLI를 사용합니다. 사용자의 계정 보안 인증과 기본 리전을 사용하도록 사용자의 AWS CLI을(를) 구성해야 합니다. 다음 명령을 입력하여 사용자의 AWS CLI를 구성합니다.
>
> ```
> aws configure
> ```

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        'event_time': datetime.datetime.now().isoformat(),
        'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
        'price': round(random.random() * 100, 2)}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(data),
            PartitionKey="partitionkey")
```

```
if __name__ == '__main__':
    generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. stock.py 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예제에 대한 Python 애플리케이션 코드는 GitHub에서 받을 수 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다.

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/scala/GettingStarted 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- build.sbt 파일에는 Managed Service for Apache Flink 라이브러리를 포함하여 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- BasicStreamingJob.scala 파일에는 애플리케이션의 기능을 정의하는 주요 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
 defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
```

```
}
```

또한 애플리케이션은 Kinesis 싱크를 사용하여 결과 스트림에 기록합니다. 다음 조각은 Kinesis 싱크를 생성합니다.

```
private def createSink: KinesisStreamsSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val outputProperties = applicationProperties.get("ProducerConfigProperties")

  KinesisStreamsSink.builder[String]
    .setKinesisClientProperties(outputProperties)
    .setSerializationSchema(new SimpleStringSchema)
    .setStreamName(outputProperties.getProperty(streamNameKey,
 defaultOutputStreamName))
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))
    .build
}
```

- 애플리케이션은 StreamExecutionEnvironment 객체를 사용하여 외부 리소스에 액세스하기 위한 소스 및 싱크 커넥터를 생성합니다.
- 애플리케이션은 동적 애플리케이션 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 런타임 애플리케이션의 속성을 읽어 커넥터를 구성합니다. 런타임 속성에 대한 자세한 내용은 런타임 속성을 참조하세요.

애플리케이션 코드 컴파일 및 업로드

이 섹션에서는 귀하가 종속 리소스 생성 섹션에서 생성한 Amazon S3 버킷에 귀하의 애플리케이션 코드를컴파일 및 업로드합니다.

애플리케이션 코드 컴파일

이 섹션에서는 SBT 구축 도구를 사용하여 애플리케이션용 Scala 코드를 구축하는 도구를 구축합니다. SBT를 설치하려면 cs 설정으로 sbt 설치를 참조하십시오. 또한 Java Devopment Kit(JDK)를 설치해야 합니다. 연습 완료를 위한 사전 조건 참조

1. 애플리케이션 코드를 사용하려면 컴파일하고 JAR 파일로 패키징합니다. SBT를 사용하여 코드를 컴파일하고 패키징할 수 있습니다.

   ```
   sbt assembly
   ```

2. 애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/scala-3.2.0/getting-started-scala-1.0.jar
```

Apache Flink 스트리밍 Scala 코드 업로드

이 섹션에서는 Amazon S3 버킷을 만들고 애플리케이션 코드를 업로드합니다.

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 ka-app-code-<username>을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 만들기를 선택합니다.
7. ka-app-code-<username> 버킷을 선택한 다음 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 getting-started-scala-1.0.jar 파일로 이동합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하십시오.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **MyApplication**을 입력합니다.

- 설명(Description)에 **My scala test app**를 입력합니다.
- 런타임에서 Apache Flink를 선택합니다.
- 버전은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두십시오.

4. Access permissions(액세스 권한)에서 Create / update IAM role **kinesis-analytics-MyApplication-us-west-2**(IAM 역할 kinesis-analytics-MyApplication-us-west-2 생성/업데이트)를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

애플리케이션을 구성하려면 다음 절차를 사용합니다.

애플리케이션 구성

1. MyApplication 페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷에 **ka-app-code-*<username>***을 입력합니다.
   - Amazon S3 객체 경로에는 **getting-started-scala-1.0.jar.**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 추가를 선택합니다.
5. 다음을 입력합니다:

그룹 ID	키	값
ConsumerConfigProperties	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

저장을 선택합니다.

6. 속성에서 그룹 추가를 다시 선택합니다.
7. 다음을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**output.stream.name**	**ExampleOutputStream**
ProducerConfigProperties	**aws.region**	**us-west-2**

8. 모니터링에서 Monitoring metrics level(지표 수준 모니터링)이 애플리케이션으로 설정되어 있는지 확인합니다.
9. CloudWatch 로깅에서 활성화 확인란을 선택합니다.
10. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Managed Service for Apache Flink에서 로그 그룹 및 로그 스트림을 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`

- 로그 스트림: `kinesis-analytics-log-stream`

IAM 정책 편집

IAM 정책을 편집하여 Amazon S3 버킷에 액세스할 수 있는 권한을 추가합니다.

IAM 정책을 편집하여 S3 버킷 권한을 추가하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 정책을 선택합니다. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 해당 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/getting-started-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
```

```
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication 페이지에서 중지를 선택합니다. 작업을 확인합니다.

애플리케이션 생성 및 실행(CLI)

이 섹션에서는 AWS Command Line Interface를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. kinesisanalyticsv2 AWS CLI 명령을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 이 애플리케이션과 상호 작용할 수 있습니다.

권한 정책 생성

> **Note**
>
> 애플리케이션에 대한 권한 정책 및 역할을 생성해야 합니다. 이러한 IAM 리소스를 생성하지 않으면 애플리케이션은 해당 데이터 및 로그 스트림에 액세스할 수 없습니다.

먼저 소스 스트림의 읽기 작업에 대한 권한을 부여하는 문과 싱크 스트림의 쓰기 작업에 대한 권한을 부여하는 문 두 개를 사용하여 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 AKReadSourceStreamWriteSinkStream 권한 정책을 생성합니다. **username**을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(**012345678901**)를 사용자의 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/getting-started-scala-1.0.jar"
```

```
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
```

```
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

권한 정책을 생성하는 단계별 지침은 IAM 사용자 가이드의 IAM 자습서: 첫 번째 고객 관리형 정책 만들기 및 연결을 참조하세요.

IAM 정책을 생성합니다.

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다.
4. Choose the service that will use this role(이 역할을 사용할 서비스 선택)에서 Kinesis를 선택합니다.
5. 사용 사례 선택에서 Managed Service for Apache Flink를 선택합니다.
6. 다음: 권한을 선택합니다.
7. Attach permissions policies(권한 정책 연결) 페이지에서 Next: Review(다음: 검토)를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
8. 역할 생성 페이지에서 역할 이름으로 **MF-stream-rw-role**을 입력합니다. 역할 생성을 선택합니다.

   MF-stream-rw-role이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.

9. 역할에 권한 정책을 연결합니다.

> **Note**
>
> 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계인 권한 정책 생성에서 생성한 정책을 연결합니다.

   a. 요약 페이지에서 권한 탭을 선택합니다.
   b. Attach Policies(정책 연결)를 선택합니다.
   c. 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.
   d. AKReadSourceStreamWriteSinkStream 정책을 선택한 후 정책 연결을 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 단계별 지침은 IAM 사용 설명서의 IAM 역할 생성(콘솔)을 참조하세요.

애플리케이션 생성

다음 JSON 코드를 create_request.json이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(사용자 이름)를 이전 섹션에서 선택한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(012345678901)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "getting_started",
    "ApplicationDescription": "Scala getting started application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "getting-started-scala-1.0.jar"
                }
            },
```

```
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

다음 요청과 함께 CreateApplication을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "getting_started",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 `StartApplication` 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Managed Service for Apache Flink 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```
{
   "ApplicationName": "s3_sink"
}
```

2. 위의 요청과 함께 `StopApplication` 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

AWS CLI를 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch Logs를 사용하는 방법에 대한 자세한 내용은 애플리케이션 로깅 설정을 참조하세요.

환경 속성 업데이트

이 섹션에서는 UpdateApplication 작업을 사용하여 애플리케이션 코드를 다시 컴파일하지 않고도 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{
      "ApplicationName": "getting_started",
       "CurrentApplicationVersionId": 1,
       "ApplicationConfigurationUpdate": {
        "EnvironmentPropertyUpdates": {
           "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
           ]
        }
    }
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplication CLI 작업을 사용합니다.

> **Note**
>
> 파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하십시오.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (<username>)를 종속 리소스 생성 섹션에서 선택한 접미사로 업데이트하십시오.

```
{{
    "ApplicationName": "getting_started",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-<username>",
                    "FileKeyUpdate": "getting-started-scala-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

AWS 리소스 정리

이 섹션에는 시작하기 텀블링 윈도우 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 패널에서 MyApplication을 선택합니다.
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 ExampleInputStream을 선택합니다.
3. ExampleInputStream 페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 ExampleOutputStream을 선택하고, 작업을 선택하고, 삭제를 선택한 다음 삭제를 확인합니다.

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code-***<username>*** 버킷을 선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service-MyApplication-us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. kinesis-analytics-MyApplication-us-west-2 역할을 선택합니다.
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. /aws/kinesis-analytics/MyApplication 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

Apache Beam을 사용하여 Managed Service for Apache Flink 애플리케이션 생성

Apache Beam 프레임워크를 Managed Service for Apache Flink 애플리케이션과 함께 사용하여 스트리밍 데이터를 처리할 수 있습니다. Apache Beam을 사용하는 Managed Service for Apache Flink 애플리케이션은 Apache Flink 러너를 사용하여 Beam 파이프라인을 실행합니다.

Managed Service for Apache Flink 애플리케이션에서 Apache Beam을 사용하는 방법에 대한 자습서는 Managed Service for Apache Flink와 함께 CloudFormation 사용하기 섹션을 참조하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink와 함께 Apache Beam 사용
- Beam 기능
- Apache Beam을 사용하여 애플리케이션 생성

Managed Service for Apache Flink와 함께 Apache Beam 사용

Managed Service for Apache Flink와 함께 Apache Flink 러너를 사용하는 방법에 대해서는 다음을 참고하세요.

- Apache Beam 지표는 Managed Service for Apache Flink 콘솔에서 볼 수 없습니다.
- Apache Beam은 Apache Flink 버전 1.8 이상을 사용하는 Managed Service for Apache Flink 애플리케이션에서만 지원됩니다. Apache Beam은 Apache Flink 버전 1.6을 사용하는 Managed Service for Apache Flink 애플리케이션에서 지원되지 않습니다.

Beam 기능

Managed Service for Apache Flink는 Apache Flink 러너와 동일한 Apache Beam 기능을 지원합니다. Apache Flink 러너에서 지원되는 기능에 대한 자세한 설명은 Beam 병립성 매트릭스를 참조하세요.

Managed Service for Apache Flink 서비스에서 Apache Flink 애플리케이션을 테스트하여 애플리케이션에 필요한 모든 기능을 지원하는지 확인하는 것이 좋습니다.

Apache Beam을 사용하여 애플리케이션 생성

이 연습에서는 Apache Beam을 사용하여 데이터를 변환하는 Managed Service for Apache Flink 애플리케이션을 생성합니다. Apache Beam은 스트리밍 데이터를 처리하기 위한 프로그래밍 모델입니다. Managed Service for Apache Flink로 Apache Beam을 사용하는 방법에 대한 자세한 설명은 Apache Beam 사용 섹션을 참조하세요.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (API) DataStream 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- 애플리케이션 코드 다운로드 및 검토
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- AWS 리소스 정리
- 다음 단계

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 다음과 같은 종속 리소스를 생성해야 합니다.

- 두 개의 Kinesis Data Streams(`ExampleInputStream` 및 `ExampleOutputStream`)
- 애플리케이션 코드를 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **`ExampleInputStream`** 및 **`ExampleOutputStream`**에 명칭을 지정합니다.

- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 임의의 문자열을 스트림에 씁니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `ping.py`이라는 파일을 생성합니다:

```
import json
import boto3
import random

kinesis = boto3.client('kinesis')

while True:
        data = random.choice(['ping', 'telnet', 'ftp', 'tracert', 'netstat'])
        print(data)
        kinesis.put_record(
                StreamName="ExampleInputStream",
                Data=data,
                PartitionKey="partitionkey")
```

2. `ping.py` 스크립트를 실행합니다.

```
$ python ping.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예에 대한 Java 애플리케이션 코드는 GitHub에서 사용할 수 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. `amazon-kinesis-data-analytics-java-examples/Beam` 디렉터리로 이동합니다.

애플리케이션 코드는 `BasicBeamStreamingJob.java` 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오.

- 애플리케이션은 Apache Beam ParDo를 사용하여 `PingPongFn`라는 맞춤 변환 함수를 호출하여 들어오는 레코드를 처리합니다.

  `PingPongFn` 함수를 호출하는 코드는 다음과 같습니다:

```
.apply("Pong transform",
    ParDo.of(new PingPongFn())
```

- Apache Beam을 사용하는 Managed Service for Apache Flink 애플리케이션에는 다음과 같은 구성 요소가 필요합니다. 이러한 구성 요소와 버전을 `pom.xml`에 포함시키지 않으면 애플리케이션이 환경 종속성에서 잘못된 버전을 로드하고 버전이 일치하지 않으므로 애플리케이션이 런타임에 충돌합니다.

```
<jackson.version>2.10.2</jackson.version>
...
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
    <artifactId>jackson-module-jaxb-annotations</artifactId>
    <version>2.10.2</version>
</dependency>
```

- `PingPongFn` 변환 함수는 입력 데이터가 ping이 아닌 경우 입력 데이터를 출력 스트림으로 전달합니다. 이 경우 pong\n 문자열을 출력 스트림으로 내보냅니다.

  변환 함수의 코드는 다음과 같습니다:

```
    private static class PingPongFn extends DoFn<KinesisRecord, byte[]> {
    private static final Logger LOG = LoggerFactory.getLogger(PingPongFn.class);

    @ProcessElement
    public void processElement(ProcessContext c) {
        String content = new String(c.element().getDataAsBytes(),
 StandardCharsets.UTF_8);
        if (content.trim().equalsIgnoreCase("ping")) {
            LOG.info("Ponged!");
            c.output("pong\n".getBytes(StandardCharsets.UTF_8));
        } else {
            LOG.info("No action for: " + content);
            c.output(c.element().getDataAsBytes());
        }
    }
}
```

애플리케이션 코드 컴파일

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 시작하기 (API) DataStream 자습서의 사전 조건을(를) 참조하세요.
2. 다음 명령을 사용하여 애플리케이션을 컴파일합니다:

```
mvn package -Dflink.version=1.15.3 -Dflink.version.minor=1.8
```

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

애플리케이션을 컴파일하면 애플리케이션 JAR 파일(`target/basic-beam-app-1.0.jar`)이 생성됩니다.

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 종속 리소스 생성 섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

1. Amazon S3 콘솔에서 ka-app-code-***<username>*** 버킷을 선택하고 업로드를 선택합니다.
2. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `basic-beam-app-1.0.jar` 파일로 이동합니다.
3. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하십시오.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.

     > **Note**
     >
     > Managed Service for Apache Flink는 Apache Flink 버전 1.15.2를 사용합니다.

   - 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.
4. Access permissions(액세스 권한)에서 Create / update IAM role **`kinesis-analytics-MyApplication-us-west-2`**(IAM 역할 kinesis-analytics-MyApplication-us-west-2 생성/업데이트)를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesis-analytics-MyApplication-us-west-2`

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 정책을 선택합니다. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 해당 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/basic-beam-app-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
```

```
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 구성

1. MyApplication 페이지에서 구성을 선택합니다.

2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷에 **ka-app-code-*<username>***을 입력합니다.
   - Amazon S3 객체 경로에 **basic-beam-app-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 다음을 입력합니다.

그룹 ID	키	값
BeamApplicationProperties	**InputStreamName**	**ExampleInputStream**
BeamApplicationProperties	**OutputStreamName**	**ExampleOutputStream**
BeamApplicationProperties	**AwsRegion**	**us-west-2**

5. 모니터링에서 Monitoring metrics level(지표 수준 모니터링)이 애플리케이션으로 설정되어 있는지 확인합니다.
6. CloudWatch logging(CloudWatch 로깅)에서 활성화 확인란을 선택합니다.
7. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Managed Service for Apache Flink에서 로그 그룹 및 로그 스트림을 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다. 이 로그 스트림은 애플리케이션이 결과를 전송하는 데 사용하는 로그 스트림과 다릅니다.

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

CloudWatch 콘솔에서 Managed Service for Apache Flink 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

AWS 리소스 정리

이 섹션에는 시작하기 텀블링 윈도우 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 패널에서 MyApplication을 선택합니다.
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 ExampleInputStream을 선택합니다.
3. ExampleInputStream 페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 ExampleOutputStream을 선택하고, 작업을 선택하고, 삭제를 선택한 다음 삭제를 확인합니다.

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code-***<username>*** 버킷을 선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service-MyApplication-us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. kinesis-analytics-MyApplication-us-west-2 역할을 선택합니다.
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제

1. https://console.aws.amazon.com/cloudwatch/에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. /aws/kinesis-analytics/MyApplication 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

다음 단계

이제 Apache Beam을 사용하여 데이터를 변환하는 기본 Managed Service for Apache Flink 애플리케이션을 생성하고 실행했으니, 고급 Managed Service for Apache Flink 솔루션의 예에 대한 자세한 설명은 다음의 애플리케이션을 참조하세요.

- Managed Service for Apache Flink 스트리밍 워크숍의 Beam: 이 워크숍에서는 하나의 균일한 Apache Beam 파이프라인에서 배치와 스트리밍 측면을 결합한 종합 예를 탐색합니다.

교육 워크숍, 실습 및 솔루션 구현

다음 end-to-end 예는 Apache Flink용 고급 관리 서비스 솔루션을 보여줍니다.

주제

- Apache Flink 애플리케이션을 로컬에서 개발하여 Apache Flink용 Managed Service에 배포하기
- Managed Service for Apache Flink Studio를 사용한 이벤트 감지
- Amazon Kinesis용 AWS 스트리밍 데이터 솔루션
- Apache Flink와 Apache Kafka를 사용한 클릭스트림 랩
- 애플리케이션 Auto Scaling 을 사용한 맞춤 조정
- 아마존 CloudWatch 대시보드
- Amazon MSK용 AWS 스트리밍 데이터 솔루션
- Apache Flink 솔루션을 위한 추가 관리 서비스: GitHub

Apache Flink 애플리케이션을 로컬에서 개발하여 Apache Flink용 Managed Service에 배포하기

이 워크숍에서는 Apache Flink용 Managed Service에 배포한다는 장기적인 목표를 가지고 로컬에서 Apache Flink 애플리케이션을 시작하고 개발하는 데 필요한 기본 사항을 보여줍니다.

솔루션은 다음에서 찾을 수 있습니다: Apache Flink를 사용한 로컬 개발을 위한 스타터 가이드

Managed Service for Apache Flink Studio를 사용한 이벤트 감지

이 워크샵에서는 Managed Service for Apache Flink Studio를 사용하여 이벤트를 감지하고 이를 Managed Service for Apache Flink 애플리케이션으로 배포하는 방법을 설명합니다.

해결 방법은 다음에서 찾을 수 있습니다: Apache Flink용 Managed Service를 사용한 이벤트 감지

Amazon Kinesis용 AWS 스트리밍 데이터 솔루션

AWS Streaming Data Solution for Amazon Kinesis는 스트리밍 데이터를 쉽게 캡처, 저장, 처리 및 전송하는 데 필요한 AWS 서비스를 자동으로 구성합니다. 이 솔루션은 스트리밍 데이터 사용 사례를 해결하기 위한 다양한 옵션을 제공합니다. Apache Flink용 관리형 서비스 옵션은 시뮬레이션된 뉴욕 택

시 데이터에 대한 분석 작업을 실행하는 실제 애플리케이션을 보여주는 end-to-end 스트리밍 ETL 예제를 제공합니다.

각 솔루션은 다음 구성 요소를 포함합니다.

- 전체 예를 배포하기 위한 AWS CloudFormation 패키지입니다.
- 애플리케이션 메트릭을 표시하기 위한 대시보드입니다. CloudWatch
- CloudWatch 가장 관련성이 높은 애플리케이션 지표에 대한 경보.
- 필요한 모든 IAM 역할 및 정책

솔루션은 Amazon Kinesis용 스트리밍 데이터 솔루션에서 찾을 수 있습니다.

Apache Flink와 Apache Kafka를 사용한 클릭스트림 랩

스트리밍 스토리지로는 Apache Kafka용 Amazon Managed Streaming을 사용하고 스트림 처리에는 Apache Flink용 Managed Service를 사용하는 클릭스트림 사용 사례를 위한 포괄적인 실습입니다.

솔루션은 클릭스트림 랩에서 찾을 수 있습니다.

애플리케이션 Auto Scaling 을 사용한 맞춤 조정

사용자가 Application Auto Scaling을 사용하여 Apache Flink 애플리케이션용 관리 서비스를 자동으로 확장하는 데 도움이 되는 샘플입니다. 이를 통해 사용자는 맞춤 조정 정책 및 맞춤 조정 속성을 설정할 수 있습니다.

솔루션은 다음에서 찾을 수 있습니다:

- Apache Flink 앱 자동 크기 조정을 위한 관리형 서비스
- 예약된 조정

사용자 지정 규모 조정을 수행하는 방법에 대한 자세한 내용은 Apache Flink용 Amazon Managed Service의 지표 기반 및 예약 조정 활성화를 참조하십시오.

아마존 CloudWatch 대시보드

Apache Flink 애플리케이션용 관리 서비스를 모니터링하기 위한 샘플 CloudWatch 대시보드입니다. 샘플 대시보드에는 대시보드의 기능을 시연하는 데 도움이 되는 데모 애플리케이션도 포함되어 있습니다.

솔루션은 Managed Service for Apache Flink 지표 대시보드에서 찾을 수 있습니다.

Amazon MSK용 AWS 스트리밍 데이터 솔루션

Amazon MSK용 AWS 스트리밍 데이터 솔루션은 생산자, 스트리밍 스토리지, 소비자 및 목적지를 통해 데이터가 흐르는 AWS CloudFormation 템플릿을 제공합니다.

솔루션은 Amazon MSK용 AWS 스트리밍 데이터 솔루션에서 찾을 수 있습니다.

Apache Flink 솔루션을 위한 추가 관리 서비스: GitHub

다음 end-to-end 예는 Apache Flink용 고급 관리 서비스 솔루션을 보여 주며 다음에서 사용할 수 있습니다. GitHub

- Amazon Managed Service for Apache Flink – 벤치마킹 유틸리티
- 스냅샷 관리자 – Apache Flink용 Amazon Managed Service for Apache Flink
- Apache Flink와 Amazon Managed Service for Apache Flink를 사용한 스트리밍 ETL
- 고객 피드백에 대한 실시간 감정 분석

유틸리티

다음 유틸리티를 사용하면 Managed Service for Apache Flink를 보다 쉽게 사용할 수 있습니다:

주제

스냅샷 관리자

Flink 애플리케이션이 저장점/스냅샷을 정기적으로 트리거하여 보다 원활한 장애 복구가 가능하도록 하는 것이 가장 좋습니다. 스냅샷 관리자는 이 작업을 자동화하며 다음과 같은 이점을 제공합니다:

- Managed Service for Apache Flink 애플리케이션 실행의 새 스냅샷을 찍습니다.
- 애플리케이션 스냅샷 수를 가져옵니다.
- 개수가 필요한 스냅샷 수보다 많은지 확인합니다.
- 필요한 수보다 오래된 스냅샷을 삭제합니다.

예제는 스냅샷 관리자를 참조하십시오.

벤치마킹

Managed Service for Apache Flink 벤치마킹 유틸리티는 Managed Service for Apache Flink 애플리케이션의 용량 계획, 통합 테스트 및 벤치마킹을 지원합니다.

예를 위해 벤치마킹을 참조하십시오.

Managed Service for Apache Flink: 예제

이 섹션은 Managed Service for Apache Flink에서 애플리케이션을 만들고 사용하는 방법의 예제들을 제공합니다. 여기에는 Apache Flink용 관리 서비스 애플리케이션을 만들고 결과를 테스트하는 데 도움이 되는 예제 코드와 step-by-step 지침이 포함되어 있습니다.

이 예제들을 살펴보기 전에 먼저 다음 사항을 검토하는 것이 좋습니다.

- 작동 방식
- 시작하기 (API) DataStream

> **Note**
>
> 이 예제들은 사용자가 미국 서부(오레곤) 리전(`us-west-2`)을 사용하고 있다고 가정합니다. 다른 리전을 사용하는 경우 애플리케이션 코드, 명령, IAM 역할을 적절하게 업데이트하세요.

주제

- DataStream API 예제
- Python 예제
- Scala 예제

DataStream API 예제

다음 예제는 Apache Flink DataStream API를 사용하여 애플리케이션을 만드는 방법을 보여줍니다.

주제

- 예: 텀블링 윈도우
- 예: 슬라이딩 윈도우
- 예: Amazon S3 버킷에 쓰기
- 자습서: Managed Service for Apache Flink 애플리케이션을 사용하여 MSK 클러스터의 한 주제에서 VPC의 다른 주제로 데이터 복제
- 예: Kinesis Data Stream과 함께 EFO 소비자 사용
- 예제: Kinesis Data Firehose에 쓰기

- 예: 다른 계정의 Kinesis 스트림에서 읽기를 참조하세요.
- 자습서: Amazon MSK에서 사용자 지정 트러스트 스토어 사용

예: 텀블링 윈도우

이 연습에서는 텀블링 윈도우를 사용하여 데이터를 집계하는 Managed Service for Apache Flink 애플리케이션을 만들어 봅니다. Flink에서 집계(aggregration)는 기본적으로 활성화되어 있습니다. 이것을 비활성화하려면 다음 명령을 사용합니다.

```
sink.producer.aggregation-enabled' = 'false'
```

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (API) DataStream 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- 애플리케이션 코드 다운로드 및 검토
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 다음과 같은 종속 리소스를 생성해야 합니다.

- 두 개의 Kinesis Data Streams(`ExampleInputStream` 및 `ExampleOutputStream`)
- 애플리케이션 코드를 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **ExampleInputStream** 및 **ExampleOutputStream**의 이름을 지정합니다.
- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        'event_time': datetime.datetime.now().isoformat(),
        'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
        'price': round(random.random() * 100, 2)}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(data),
```

```
                PartitionKey="partitionkey")


    if __name__ == '__main__':
        generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. stock.py 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/TumblingWindow 디렉터리로 이동합니다.

애플리케이션 코드는 TumblingWindowStreamingJob.java 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 다음 가져오기 문을 추가합니다.

```
import
 org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows; //
flink 1.13 onward
```

- 이 애플리케이션은 `timeWindow` 연산자를 사용하여 5초 동안의 텀블링 윈도우에서 각 주식 기호에 대한 값의 개수를 찾습니다. 다음 코드는 연산자를 생성하고 집계된 데이터를 새로운 Kinesis Data Streams 싱크로 전송합니다.

```
input.flatMap(new Tokenizer()) // Tokenizer for generating words
                    .keyBy(0) // Logically partition the stream for each word

                    .window(TumblingProcessingTimeWindows.of(Time.seconds(5))) //
Flink 1.13 onward
                    .sum(1) // Sum the number of words per partition
                    .map(value -> value.f0 + "," + value.f1.toString() + "\n")
                    .addSink(createSinkFromStaticConfig());
```

애플리케이션 코드 컴파일

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 시작하기 (API) DataStream 자습서의 사전 조건을(를) 참조하세요.
2. 다음 명령을 사용하여 애플리케이션을 컴파일합니다:

```
mvn package -Dflink.version=1.15.3
```

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

애플리케이션을 컴파일하면 애플리케이션 JAR 파일(`target/aws-kinesis-analytics-java-apps-1.0.jar`)이 생성됩니다.

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 종속 리소스 생성 섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

1. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.

2. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다.
3. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.

     > **Note**
     >
     > Managed Service for Apache Flink는 Apache Flink 버전 1.15.2를 사용합니다.

   - 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*

- 역할: kinesisanalytics-*MyApplication*-*us-west-2*

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
```

```
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.

5. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
6. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다. 이 로그 스트림은 애플리케이션이 결과를 전송하는 데 사용하는 로그 스트림과 다릅니다.

애플리케이션 실행

1. MyApplication페이지에서 [Run] 을 선택합니다. 스냅샷 없이 실행 옵션을 선택한 상태로 두고 작업을 확인합니다.
2. 애플리케이션이 실행 중이면 페이지를 새로 고칩니다. 콘솔에 애플리케이션 그래프가 표시됩니다.

CloudWatch 콘솔에서 Apache Flink용 관리 서비스 메트릭을 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

AWS 리소스 정리

이 섹션에는 시작하기 텀블링 윈도우 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

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

2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: 슬라이딩 윈도우

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (API) DataStream 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- 애플리케이션 코드 다운로드 및 검토
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 다음과 같은 종속 리소스를 생성해야 합니다.

- 두 개의 Kinesis Data Streams(`ExampleInputStream` 및 `ExampleOutputStream`).
- 애플리케이션 코드를 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **`ExampleInputStream`** 및 **`ExampleOutputStream`**에 명칭을 지정합니다.

- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )
```

```
if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

2. stock.py 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/SlidingWindow 디렉터리로 이동합니다.

애플리케이션 코드는 SlidingWindowStreamingJobWithParallelism.java 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 이 애플리케이션은 timeWindow 연산자를 사용하여 5초씩 슬라이드되는 10초 윈도우에서 각 주식 종목의 최소값을 찾습니다. 다음 코드는 연산자를 생성하고 집계된 데이터를 새로운 Kinesis Data Streams 싱크로 전송합니다.
- 다음 가져오기 문을 추가합니다.

```
import
 org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows; //
flink 1.13 onward
```

- 이 애플리케이션은 timeWindow 연산자를 사용하여 5초 동안의 텀블링 윈도우에서 각 주식 기호에 대한 값의 개수를 찾습니다. 다음 코드는 연산자를 생성하고 집계된 데이터를 새로운 Kinesis Data Streams 싱크로 전송합니다.

```
input.flatMap(new Tokenizer()) // Tokenizer for generating words
                .keyBy(0) // Logically partition the stream for each word

  .window(TumblingProcessingTimeWindows.of(Time.seconds(5))) //Flink 1.13 onward
                .sum(1) // Sum the number of words per partition
                .map(value -> value.f0 + "," + value.f1.toString() + "\n")
                .addSink(createSinkFromStaticConfig());
```

애플리케이션 코드 컴파일

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 시작하기 (API) DataStream 자습서의 사전 조건을(를) 참조하세요.
2. 다음 명령을 사용하여 애플리케이션을 컴파일합니다:

```
mvn package -Dflink.version=1.15.3
```

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

애플리케이션을 컴파일하면 애플리케이션 JAR 파일(target/aws-kinesis-analytics-java-apps-1.0.jar)이 생성됩니다.

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 종속 리소스 생성섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

1. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택한 다음 업로드를 선택합니다.
2. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다.
3. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - 역할: `kinesisanalytics-`*`MyApplication`*`-`*`us-west-2`*

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
```

```
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
5. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
6. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다. 이 로그 스트림은 애플리케이션이 결과를 전송하는 데 사용하는 로그 스트림과 다릅니다.

애플리케이션 병렬 처리 구성

이 애플리케이션 예제에서는 작업의 병렬 실행을 사용합니다. 다음 애플리케이션 코드는 `min` 연산자의 병렬 처리를 설정합니다.

```
.setParallelism(3) // Set parallelism for the min operator
```

애플리케이션 병렬 처리는 기본값이 1인 프로비저닝된 병렬 처리보다 클 수 없습니다. 애플리케이션의 병렬 처리를 높이려면 다음 AWS CLI 작업을 사용하세요.

```
aws kinesisanalyticsv2 update-application
      --application-name MyApplication
      --current-application-version-id <VersionId>
      --application-configuration-update "{\"FlinkApplicationConfigurationUpdate
\": { \"ParallelismConfigurationUpdate\": {\"ParallelismUpdate\": 5,
 \"ConfigurationTypeUpdate\": \"CUSTOM\" }}}"
```

DescribeApplication또는 ListApplications작업을 사용하여 현재 애플리케이션 버전 ID를 검색할 수 있습니다.

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 메트릭을 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

AWS 리소스 정리

이 섹션에는 슬라이딩 윈도우 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.

2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: Amazon S3 버킷에 쓰기

이 연습에서는 Kinesis Data Stream을 소스로 사용하고 Amazon S3 버킷을 싱크로 사용하는 Managed Service for Apache Flink를 생성합니다. 싱크를 사용하여 Amazon S3 콘솔의 애플리케이션 출력을 확인할 수 있습니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (API) DataStream 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- 애플리케이션 코드 다운로드 및 검토
- 애플리케이션 코드 수정
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행

- 애플리케이션 출력 확인
- 선택 사항: 소스 및 싱크 사용자 지정
- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink를 생성하기 전에 먼저 다음과 같은 종속 리소스를 생성해야 합니다:

- Kinesis Data Streams(`ExampleInputStream`).
- 애플리케이션 코드와 출력을 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

> **Note**
>
> Managed Service for Apache Flink는 Managed Service for Apache Flink에서 서버 측 암호화가 활성화된 상태에서는 Amazon S3에 데이터를 쓸 수 없습니다.

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **`ExampleInputStream`**의 이름을 지정합니다.
- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **`ka-app-code-<username>`**)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다. Amazon S3 버킷에 두 개의 폴더(**`code`**및**`data`**)를 생성합니다.

애플리케이션은 다음 리소스가 아직 없는 경우 해당 리소스를 생성합니다. CloudWatch

- `/AWS/KinesisAnalytics-java/MyApplication`라는 로그 그룹.
- `kinesis-analytics-log-stream`라는 로그 스트림.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        'event_time': datetime.datetime.now().isoformat(),
        'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
        'price': round(random.random() * 100, 2)}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(data),
            PartitionKey="partitionkey")


if __name__ == '__main__':
    generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. `stock.py` 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다 GitHub. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. `amazon-kinesis-data-analytics-java-examples/S3Sink` 디렉터리로 이동합니다.

애플리케이션 코드는 `S3StreamingSinkJob.java` 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 가져오기 문을 다음과 같이 추가합니다.

```
import
 org.apache.flink.streaming.api.windowing.assigners.TumblingProcessingTimeWindows;
```

- 이 애플리케이션은 Apache Flink S3 싱크를 사용하여 Amazon S3에 쓰기 합니다.

  싱크는 텀블링 윈도우에서 메시지를 읽고, 메시지를 S3 버킷 객체로 인코딩하고, 인코딩된 객체를 S3 싱크로 보냅니다. 다음 코드는 Amazon S3로 전송할 객체를 인코딩합니다.

```
input.map(value -> { // Parse the JSON
                JsonNode jsonNode = jsonParser.readValue(value, JsonNode.class);
                return new Tuple2<>(jsonNode.get("ticker").toString(), 1);
            }).returns(Types.TUPLE(Types.STRING, Types.INT))
            .keyBy(v -> v.f0) // Logically partition the stream for each word
            .window(TumblingProcessingTimeWindows.of(Time.minutes(1)))
            .sum(1) // Count the appearances by ticker per partition
            .map(value -> value.f0 + " count: " + value.f1.toString() + "\n")
            .addSink(createS3SinkFromStaticConfig());
```

> **Note**
>
> 이 애플리케이션은 Flink `StreamingFileSink` 객체를 사용하여 Amazon S3에 쓰기 합니다. 에 대한 자세한 내용은 Apache Flink 설명서를 참조하십시오 StreamingFileSink. `StreamingFileSink`

애플리케이션 코드 수정

이 섹션에서는 Amazon S3 버킷에 출력을 쓰기 하도록 애플리케이션 코드를 수정합니다.

다음 줄을 사용자 이름으로 업데이트하여 애플리케이션의 출력 위치를 지정하세요.

```
private static final String s3SinkPath = "s3a://ka-app-code-<username>/data";
```

애플리케이션 코드 컴파일

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 시작하기 (API) DataStream 자습서의 사전 조건을(를) 참조하세요.
2. 다음 명령을 사용하여 애플리케이션을 컴파일합니다:

   ```
   mvn package -Dflink.version=1.15.3
   ```

애플리케이션을 컴파일하면 애플리케이션 JAR 파일(`target/aws-kinesis-analytics-java-apps-1.0.jar`)이 생성됩니다.

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 종속 리소스 생성 섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

1. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 코드 폴더로 이동한 다음 Upload를 선택합니다.
2. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다.
3. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
> - 런타임에서 Apache Flink를 선택합니다.
> - 버전은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두십시오.

6. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
7. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink를 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Stream에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다. <username>을 내 사용자 이름으로 바꿉니다.

```
{
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:Abort*",
                "s3:DeleteObject*",
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
```

```
                "arn:aws:s3:::ka-app-code-<username>",
                "arn:aws:s3:::ka-app-code-<username>/*"
            ]
          },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:%LOG_GROUP_PLACEHOLDER
%:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:region:account-id:log-group:%LOG_GROUP_PLACEHOLDER
%:log-stream:%LOG_STREAM_PLACEHOLDER%"
            ]
        }
        ,
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
```

```
    ]
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **code/aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
5. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
6. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다. 이 로그 스트림은 애플리케이션이 결과를 전송하는 데 사용하는 로그 스트림과 다릅니다.

애플리케이션 실행

1. MyApplication페이지에서 [Run] 을 선택합니다. 스냅샷 없이 실행 옵션을 선택한 상태로 두고 작업을 확인합니다.

2. 애플리케이션이 실행 중이면 페이지를 새로 고칩니다. 콘솔에 애플리케이션 그래프가 표시됩니다.

애플리케이션 출력 확인

Amazon S3 콘솔에서 S3 버킷의 데이터 폴더를 엽니다.

몇 분 후 애플리케이션에서 집계된 데이터를 포함하는 객체가 나타납니다.

> **Note**
>
> Flink에서 집계(aggregration)는 기본적으로 활성화되어 있습니다. 이것을 비활성화하려면 다음 명령을 사용합니다.
>
> ```
> sink.producer.aggregation-enabled' = 'false'
> ```

선택 사항: 소스 및 싱크 사용자 지정

이 섹션에서는 소스 및 싱크 객체의 설정을 사용자 지정합니다.

> **Note**
>
> 다음 섹션에 설명된 코드 섹션을 변경한 후 다음을 수행하여 애플리케이션 코드를 다시 로드하세요.
>
> - the section called “애플리케이션 코드 컴파일” 섹션의 단계를 반복하여 업데이트된 애플리케이션 코드를 컴파일합니다.
> - the section called “Apache Flink 스트리밍 Java 코드 업로드” 섹션의 단계를 반복하여 업데이트된 애플리케이션 코드를 업로드합니다.
> - 콘솔의 애플리케이션 페이지에서 구성을 선택한 다음 업데이트를 선택하여 업데이트된 애플리케이션 코드를 애플리케이션에 다시 로드합니다.

이 섹션에는 다음 섹션이 포함됩니다.

- 데이터 파티셔닝 구성
- 읽기 빈도 구성

- 쓰기 버퍼링 구성

데이터 파티셔닝 구성

이 섹션에서는 스트리밍 파일 싱크가 S3 버킷에 생성하는 폴더의 이름을 구성합니다. 스트리밍 파일 싱크에 버킷 할당자를 추가하여 이 작업을 수행합니다.

S3 버킷에 생성된 폴더 이름을 사용자 지정하려면 다음을 수행합니다.

1. S3StreamingSinkJob.java 파일 시작 부분에 다음 가져오기 문을 추가합니다.

```
import
 org.apache.flink.streaming.api.functions.sink.filesystem.rollingpolicies.DefaultRollingPol
import
 org.apache.flink.streaming.api.functions.sink.filesystem.bucketassigners.DateTimeBucketAss
```

2. 코드의 createS3SinkFromStaticConfig() 메서드를 다음과 같이 업데이트하세요.

```
private static StreamingFileSink<String> createS3SinkFromStaticConfig() {

    final StreamingFileSink<String> sink = StreamingFileSink
        .forRowFormat(new Path(s3SinkPath), new
 SimpleStringEncoder<String>("UTF-8"))
        .withBucketAssigner(new DateTimeBucketAssigner("yyyy-MM-dd--HH"))
        .withRollingPolicy(DefaultRollingPolicy.create().build())
        .build();
    return sink;
}
```

앞의 코드 예제에서는 사용자 지정 날짜 형식과 함께 DateTimeBucketAssigner를 사용하여 S3 버킷에 폴더를 만듭니다. DateTimeBucketAssigner에서는 현재 시스템 시간을 사용하여 버킷 이름을 생성합니다. 사용자 지정 버킷 할당자를 만들어 생성된 폴더 이름을 추가로 사용자 지정하려는 경우 이를 구현하는 클래스를 만들 수 있습니다. BucketAssigner getBucketId 메서드를 사용하여 사용자 지정 로직을 구현합니다.

BucketAssigner의 사용자 지정 구현에서는 콘텍스트 파라미터를 사용하여 레코드에 대한 추가 정보를 가져와 대상 폴더를 결정할 수 있습니다.

읽기 빈도 구성

이 섹션에서는 소스 스트림의 읽기 빈도를 구성합니다.

Kinesis Streams 소비자는 기본적으로 소스 스트림에서 초당 5회 읽습니다. 스트림에서 읽는 클라이언트가 두 명 이상이거나 애플리케이션이 레코드 읽기를 재시도해야 하는 경우 이 빈도로 인해 문제가 발생합니다. 소비자의 읽기 빈도를 설정하면 이러한 문제를 피할 수 있습니다.

Kinesis 소비자의 읽기 빈도를 설정하려면 SHARD_GETRECORDS_INTERVAL_MILLIS 설정을 지정합니다.

다음 코드 예제는 SHARD_GETRECORDS_INTERVAL_MILLIS 설정을 1초로 지정합니다.

```
kinesisConsumerConfig.setProperty(ConsumerConfigConstants.SHARD_GETRECORDS_INTERVAL_MILLIS,
 "1000");
```

쓰기 버퍼링 구성

이 섹션에서는 싱크의 쓰기 빈도와 기타 설정을 구성합니다.

기본적으로 애플리케이션은 대상 버킷에 1분마다 데이터를 씁니다. DefaultRollingPolicy 객체를 구성하여 이 간격 및 기타 설정을 변경할 수 있습니다.

> **Note**
>
> Apache Flink 스트리밍 파일 싱크는 애플리케이션이 체크포인트를 생성할 때마다 출력 버킷에 기록합니다. 애플리케이션은 기본적으로 1분마다 체크포인트를 생성합니다. S3 싱크의 쓰기 간격을 늘리려면 체크포인트 간격도 늘려야 합니다.

DefaultRollingPolicy 객체를 구성하려면 다음을 수행합니다.

1. 애플리케이션의 CheckpointInterval 설정을 늘립니다. UpdateApplication작업에 대한 다음 입력은 체크포인트 간격을 10분으로 설정합니다.

```
{
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "CheckpointConfigurationUpdate": {
            "ConfigurationTypeUpdate" : "CUSTOM",
            "CheckpointIntervalUpdate": 600000
         }
```

```
      }
   },
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 5
}
```

위의 코드를 사용하려면 현재 애플리케이션 버전을 지정합니다. ListApplications작업을 사용하여 애플리케이션 버전을 검색할 수 있습니다.

2. S3StreamingSinkJob.java 파일 시작 부분에 다음 가져오기 문을 추가합니다.

```
import java.util.concurrent.TimeUnit;
```

3. S3StreamingSinkJob.java 파일에서 createS3SinkFromStaticConfig 메서드를 다음과 같이 업데이트합니다.

```
private static StreamingFileSink<String> createS3SinkFromStaticConfig() {

        final StreamingFileSink<String> sink = StreamingFileSink
                .forRowFormat(new Path(s3SinkPath), new
 SimpleStringEncoder<String>("UTF-8"))
                .withBucketAssigner(new DateTimeBucketAssigner("yyyy-MM-dd--HH"))
                .withRollingPolicy(
                        DefaultRollingPolicy.create()
                            .withRolloverInterval(TimeUnit.MINUTES.toMillis(8))
                            .withInactivityInterval(TimeUnit.MINUTES.toMillis(5))
                            .withMaxPartSize(1024 * 1024 * 1024)
                            .build())
                .build();
        return sink;
    }
```

위의 코드 예제는 Amazon S3 버킷에 대한 쓰기 빈도를 8분으로 설정합니다.

Apache Flink 스트리밍 파일 싱크를 구성하는 방법에 대한 자세한 내용은 Apache Flink 설명서의 행 인코딩 형식을 참조하세요.

AWS 리소스 정리

이 섹션에는 Amazon S3 tutorial 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Stream 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Stream 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색바에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication

8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

자습서: Managed Service for Apache Flink 애플리케이션을 사용하여 MSK 클러스터의 한 주제에서 VPC의 다른 주제로 데이터 복제

다음 자습서는 Amazon MSK 클러스터와 두 가지 주제를 사용하여 Amazon VPC를 생성하는 방법과 하나의 Amazon MSK 주제에서 읽고 다른 주제에 쓰는 Managed Service for Apache Flink 애플리케이션을 생성하는 방법을 보여줍니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (API) DataStream 연습을 완료하세요.

이 자습서는 다음 섹션을 포함하고 있습니다:

- Amazon MSK 클러스터로 Amazon VPC 생성
- 애플리케이션 코드 생성
- Apache Flink 스트리밍 Java 코드 업로드
- 애플리케이션 생성
- 애플리케이션 구성
- 애플리케이션 실행
- 애플리케이션 테스트

Amazon MSK 클러스터로 Amazon VPC 생성

Managed Service for Apache Flink 애플리케이션에서 액세스할 샘플 VPC 및 Amazon MSK 클러스터를 생성하려면 Amazon MSK 사용 시작하기 자습서를 따르세요.

자습서를 완료할 때는 다음 사항에 유의하십시오:

- 3단계: 주제 만들기에서 `kafka-topics.sh --create` 명령을 반복하여 `AWSKafkaTutorialTopicDestination`라는 이름의 대상 주제를 생성합니다.

```
bin/kafka-topics.sh --create --zookeeper ZooKeeperConnectionString --replication-
factor 3 --partitions 1 --topic AWSKafkaTutorialTopicDestination
```

- 클러스터의 부트스트랩 서버 목록을 기록합니다. 다음 명령을 사용하여 부트스트랩 서버 목록을 가져올 수 있습니다 (MSK 클러스터의 *ClusterArn*ARN으로 대체).

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
{...
    "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094"
}
```

- 자습서의 단계를 따를 때는 코드, 명령 및 콘솔 항목에서 선택한 AWS 리전을 사용해야 합니다.

애플리케이션 코드 생성

이 섹션에서는 애플리케이션 JAR 파일을 다운로드하고 컴파일합니다. Java 11을 사용하는 것이 좋습니다.

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. 애플리케이션 코드는 `amazon-kinesis-data-analytics-java-examples/KafkaConnectors/KafkaGettingStartedJob.java` 파일에 있습니다. 코드를 검토하여 Managed Service for Apache Flink 애플리케이션 코드의 구조를 익힐 수 있습니다.
4. 명령줄 Maven 도구 또는 원하는 개발 환경을 사용하여 JAR 파일을 만드세요. 명령줄 Maven 도구를 사용하여 JAR 파일을 컴파일하려면 다음을 입력합니다.

```
mvn package -Dflink.version=1.15.3
```

빌드가 성공하면 다음 파일이 생성됩니다.

```
target/KafkaGettingStartedJob-1.0.jar
```

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다. 개발 환경을 사용하는 경우

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 시작하기 (API) DataStream 자습서에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

> **Note**
>
> 시작 자습서에서 Amazon S3 버킷을 삭제한 경우 the section called “Apache Flink 스트리밍 Java 코드 업로드” 단계를 다시 수행하세요.

1. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
2. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `KafkaGettingStartedJob-1.0.jar` 파일로 이동합니다.
3. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:

- 애플리케이션 명칭에 **MyApplication**을 입력합니다.
- 런타임에서 Apache Flink 버전 1.15.2를 선택합니다.

4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **KafkaGettingStartedJob-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.

> **Note**
>
> 콘솔 (예: CloudWatch Logs 또는 Amazon VPC) 을 사용하여 애플리케이션 리소스를 지정하면 콘솔은 애플리케이션 실행 역할을 수정하여 해당 리소스에 액세스할 권한을 부여합니다.

4. 속성에서 그룹 추가를 선택합니다. 다음 속성을 입력합니다.

그룹 ID	키	값
KafkaSource	주제	AWSKafkaTutorialTopic
KafkaSource	bootstrap.servers	### ### ##### ## ##
KafkaSource	security.protocol	SSL
KafkaSource	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSource	ssl.truststore.password	changeit

> **Note**
>
> 기본 인증서의 ssl.truststore.password는 “changeit”입니다. 기본 인증서를 사용하는 경우에는 이 값을 변경할 필요가 없습니다.

그룹 추가를 다시 선택합니다. 다음 속성을 입력합니다.

그룹 ID	키	값
KafkaSink	주제	AWSKafkaTutorialTopicDestination
KafkaSink	bootstrap.servers	### ### ##### ## ##
KafkaSink	security.protocol	SSL
KafkaSink	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSink	ssl.truststore.password	changeit
KafkaSink	transaction.timeout.ms	1000

애플리케이션 코드는 위의 애플리케이션 속성을 읽고 VPC 및 Amazon MSK 클러스터와 상호 작용하는 데 사용되는 소스 및 싱크를 구성합니다. 클러스터 속성 사용에 대한 자세한 내용은 런타임 속성을(를) 참조하세요.

5. 스냅샷에서 비활성화를 선택합니다. 이렇게 하면 잘못된 애플리케이션 상태 데이터를 로드하지 않고도 애플리케이션을 더 쉽게 업데이트할 수 있습니다.
6. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
7. CloudWatch 로깅하려면 활성화 확인란을 선택합니다.
8. Virtual Private Cloud(VPC) 섹션에서 애플리케이션과 연결할 VPC를 선택합니다. 애플리케이션이 VPC 리소스에 액세스하는 데 사용할 VPC와 연결된 서브넷 및 보안 그룹을 선택합니다.
9. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다.

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 테스트

이 섹션에서는 소스 주제에 레코드를 쓰기 합니다. 애플리케이션은 소스 주제에서 레코드를 읽고 대상 주제에 쓰기 합니다. 소스 주제에 레코드를 쓰고 대상 주제에서 레코드를 읽어 애플리케이션이 작동 중인지 확인합니다.

주제에서 레코드를 쓰고 읽으려면 Amazon MSK 사용 시작하기 자습서의 6단계: 데이터 생성 및 소비의 단계를 따르세요.

대상 주제에서 읽으려면 클러스터에 대한 두 번째 연결에서 소스 주제 대신 대상 주제 이름을 사용하세요.

```
bin/kafka-console-consumer.sh --bootstrap-server BootstrapBrokerString --
consumer.config client.properties --topic AWSKafkaTutorialTopicDestination --from-
beginning
```

대상 주제에 레코드가 표시되지 않는 경우 문제 해결 주제의 VPC에서 리소스에 액세스 불가능 섹션을 참조하세요.

예: Kinesis Data Stream과 함께 EFO 소비자 사용

이 연습에서는 향상된 팬아웃(EFO) 소비자를 사용하여 Kinesis Data Stream에서 읽는 Managed Service for Apache Flink 애플리케이션을 생성합니다. Kinesis 컨슈머가 EFO를 사용하는 경우 Kinesis Data Streams 서비스는 컨슈머가 스트림에서 읽는 다른 컨슈머와 스트림의 고정 대역폭을 공유하지 않고 자체 전용 대역폭을 제공합니다.

Kinesis 소비자와 함께 EFO를 사용하는 방법에 대한 자세한 내용은 FLIP-128: Kinesis 컨슈머를 위한 고급 팬아웃을 참조하세요.

이 예제에서 만든 애플리케이션은 AWS Kinesis 커넥터 (flink-connector-kinesis) 1.15.3을 사용합니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (API) DataStream 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- 애플리케이션 코드 다운로드 및 검토
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 다음과 같은 종속 리소스를 생성해야 합니다.

- 두 개의 Kinesis Data Streams(`ExampleInputStream` 및 `ExampleOutputStream`)
- 애플리케이션 코드를 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **`ExampleInputStream`** 및 **`ExampleOutputStream`**의 이름을 지정합니다.
- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **`ka-app-code-<username>`**)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
```

```
        return {
            'event_time': datetime.datetime.now().isoformat(),
            'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
            'price': round(random.random() * 100, 2)}


    def generate(stream_name, kinesis_client):
        while True:
            data = get_data()
            print(data)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey")


    if __name__ == '__main__':
        generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. `stock.py` 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. `amazon-kinesis-data-analytics-java-examples/EfoConsumer` 디렉터리로 이동합니다.

애플리케이션 코드는 `EfoApplication.java` 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- Kinesis 소비자에 다음 파라미터를 설정하여 EFO 소비자를 활성화합니다.
  - RECORD_PUBLISHER_TYPE: 애플리케이션이 EFO 소비자를 사용하여 Kinesis Data Stream 데이터에 액세스하도록 이 파라미터를 EFO로 설정하세요.
  - EFO_CONSUMER_NAME: 이 파라미터를 이 스트림의 소비자 간에 고유한 문자열 값으로 설정합니다. 동일한 Kinesis Data Stream에서 컨슈머 명칭을 재사용하면 해당 명칭을 사용하던 이전 컨슈머가 종료됩니다.
- 다음 코드 예제는 EFO 소비자가 소스 스트림에서 읽을 수 있도록 소비자 구성 속성에 값을 할당하는 방법을 보여줍니다.

```
consumerConfig.putIfAbsent(RECORD_PUBLISHER_TYPE, "EFO");
consumerConfig.putIfAbsent(EFO_CONSUMER_NAME, "basic-efo-flink-app");
```

애플리케이션 코드 컴파일

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 시작하기 (API) DataStream 자습서의 사전 조건을(를) 참조하세요.
2. 다음 명령을 사용하여 애플리케이션을 컴파일합니다:

```
mvn package -Dflink.version=1.15.3
```

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

애플리케이션을 컴파일하면 애플리케이션 JAR 파일(`target/aws-kinesis-analytics-java-apps-1.0.jar`)이 생성됩니다.

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 종속 리소스 생성 섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

1. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.

2. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다.
3. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.

     > **Note**
     >
     > Managed Service for Apache Flink는 Apache Flink 버전 1.15.2를 사용합니다.

   - 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*

- 역할: kinesisanalytics-*MyApplication*-*us-west-2*

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

> **Note**
>
> 이러한 권한은 애플리케이션에 EFO 소비자에 액세스할 수 있는 권한을 부여합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
```

```
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "AllStreams",
            "Effect": "Allow",
            "Action": [
                "kinesis:ListShards",
                "kinesis:ListStreamConsumers",
                "kinesis:DescribeStreamSummary"
            ],
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/*"
        },
        {
            "Sid": "Stream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:RegisterStreamConsumer",
                "kinesis:DeregisterStreamConsumer"
            ],
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
```

```
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        },
        {
            "Sid": "Consumer",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStreamConsumer",
                "kinesis:SubscribeToShard"
            ],
            "Resource": [
                "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream/
consumer/my-efo-flink-app",
                "arn:aws:kinesis:us-west-2:012345678901:stream/ExampleInputStream/
consumer/my-efo-flink-app:*"
            ]
        }
    ]
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 생성을 선택합니다.
5. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
ConsumerConfigProperties	**flink.stream.recordpublisher**	**EFO**

그룹 ID	키	값
ConsumerConfigProperties	**flink.stream.efo.consumername**	**basic-efo-flink-app**
ConsumerConfigProperties	**INPUT_STREAM**	**ExampleInputStream**
ConsumerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ConsumerConfigProperties	**AWS_REGION**	**us-west-2**

6. 속성에서 그룹 생성을 선택합니다.
7. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**OUTPUT_STREAM**	**ExampleOutputStream**
ProducerConfigProperties	**AWS_REGION**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

8. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
9. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
10. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`

- 로그 스트림: `kinesis-analytics-log-stream`

이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다. 이 로그 스트림은 애플리케이션이 결과를 전송하는 데 사용하는 로그 스트림과 다릅니다.

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

CloudWatch 콘솔에서 Apache Flink용 관리 서비스 메트릭을 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

데이터 스트림의 향상된 팬아웃 탭에 있는 Kinesis Data Streams 콘솔에서 소비자 이름 () 을 확인할 수도 있습니다. basic-efo-flink-app

AWS 리소스 정리

이 섹션에는 efo Window 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.

2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예제: Kinesis Data Firehose에 쓰기

이 연습에서는 Kinesis 데이터 스트림을 소스로 사용하고 Kinesis Data Firehose 스트림을 싱크로 사용하는 Managed Service for Apache Flink 애플리케이션을 생성합니다. 싱크를 사용하여 Amazon S3 버킷의 애플리케이션 출력을 확인할 수 있습니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (API) DataStream 연습을 완료하세요.

이 섹션은 다음 주제를 포함합니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- Apache Flink 스트리밍 Java 코드 다운로드 및 검사
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink를 생성하기 전에 먼저 다음과 같은 종속 리소스를 생성해야 합니다:

- Kinesis Data Streams(`ExampleInputStream`).
- 애플리케이션이 출력을 (`ExampleDeliveryStream`)에 기록하는 Kinesis Data Firehose 스트림
- 애플리케이션 코드를 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

콘솔을 사용하여 Kinesis 스트림, Amazon S3 버킷 및 Kinesis Data Firehose 스트림을 생성할 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **`ExampleInputStream`**의 이름을 지정합니다.

- 자세한 정보는 Amazon Kinesis Data Firehose 개발자 안내서의 Amazon Kinesis Data Firehose 전송 스트림 생성을 참조하세요. Kinesis Data Firehose 스트림의 이름을 **ExampleDeliveryStream**으로 지정합니다. Kinesis Data Firehose 스트림을 생성할 때는 Kinesis Data Firehose 스트림의 S3 대상과 IAM 역할도 생성합니다.
- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
   import json
   import random
   import boto3

   STREAM_NAME = "ExampleInputStream"


   def get_data():
       return {
           'event_time': datetime.datetime.now().isoformat(),
           'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
           'price': round(random.random() * 100, 2)}


   def generate(stream_name, kinesis_client):
       while True:
           data = get_data()
           print(data)
           kinesis_client.put_record(
```

```
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey")


    if __name__ == '__main__':
        generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. stock.py 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

Apache Flink 스트리밍 Java 코드 다운로드 및 검사

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/FirehoseSink 디렉터리로 이동합니다.

애플리케이션 코드는 FirehoseSinkStreamingJob.java 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 애플리케이션은 Kinesis Data Firehose 싱크를 사용하여 Kinesis Data Firehose 스트림에 데이터를 씁니다. 다음 스니펫은 Kinesis Data Firehose 싱크를 생성합니다.

```
private static KinesisFirehoseSink<String> createFirehoseSinkFromStaticConfig() {
        Properties sinkProperties = new Properties();
```

```
        sinkProperties.setProperty(AWS_REGION, region);

        return KinesisFirehoseSink.<String>builder()
                .setFirehoseClientProperties(sinkProperties)
                .setSerializationSchema(new SimpleStringSchema())
                .setDeliveryStreamName(outputDeliveryStreamName)
                .build();
    }
```

애플리케이션 코드 컴파일

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 시작하기 (API) DataStream 자습서의 사전 조건을(를) 참조하세요.
2. 다음 애플리케이션에 Kinesis 커넥터를 사용하려면 Apache Maven을 다운로드, 빌드 및 설치해야 합니다. 자세한 내용은 the section called “Apache Flink Kinesis 스트림 커넥터를 이전 Apache Flink 버전과 함께 사용” 단원을 참조하세요.
3. 다음 명령을 사용하여 애플리케이션을 컴파일합니다:

```
mvn package -Dflink.version=1.15.3
```

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

애플리케이션을 컴파일하면 애플리케이션 JAR 파일(`target/aws-kinesis-analytics-java-apps-1.0.jar`)이 생성됩니다.

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 종속 리소스 생성섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

애플리케이션 코드 업로드하기

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.

2. 콘솔에서 ka-app-code- <username>버킷을 선택한 다음 Upload를 선택합니다.
3. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `java-getting-started-1.0.jar` 파일로 이동합니다.
4. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔이나 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 생성하고 실행할 수 있습니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 생성하면 사용자 AWS Identity and Access Management (IAM) 및 Amazon CloudWatch Logs 리소스가 자동으로 생성됩니다. AWS CLI를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 만듭니다.

주제

- 애플리케이션 생성 및 실행(콘솔)
- 애플리케이션 생성 및 실행 (AWS CLI)

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 설명에 **`My java test app`**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.

> **Note**
>
> Managed Service for Apache Flink는 Apache Flink 버전 1.15.2를 사용합니다.

- 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.

4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

IAM 정책 편집

IAM 정책을 편집하여 Kinesis 데이터 스트림과 Kinesis Data Firehose 스트림에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)의 모든 인스턴스를 내 계정 아이디로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
           ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
```

```
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteDeliveryStream",
            "Effect": "Allow",
            "Action": "firehose:*",
            "Resource": "arn:aws:firehose:us-west-2:012345678901:deliverystream/
ExampleDeliveryStream"
        }
    ]
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **java-getting-started-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
5. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
6. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

MyApplication페이지에서 [중지] 를 선택합니다. 작업을 확인합니다.

애플리케이션 업데이트

콘솔을 사용하여 애플리케이션 속성, 모니터링 설정, 애플리케이션 JAR의 위치 또는 파일 명칭과 같은 애플리케이션 설정을 업데이트할 수 있습니다.

MyApplication페이지에서 구성을 선택합니다. 애플리케이션 설정을 업데이트하고 업데이트를 선택합니다.

> **Note**
>
> 콘솔에서 애플리케이션 코드를 업데이트하려면 JAR의 객체 이름을 변경하거나, 다른 S3 버킷을 사용하거나, the section called “애플리케이션 코드 업데이트”섹션에 설명된 대로 AWS CLI를 사용해야 합니다. 파일 이름이나 버킷이 변경되지 않으면 구성 페이지에서 업데이트를 선택할 때 애플리케이션 코드가 다시 로드되지 않습니다.

애플리케이션 생성 및 실행 (AWS CLI)

이 섹션에서는 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다.

권한 정책 생성

먼저 소스 스트림에서 두 개의 명령문, 즉 `read` 작업에 대한 권한을 부여하는 명령문과 싱크 스트림에서 `write` 작업에 대한 권한을 부여하는 명령문이 있는 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 `AKReadSourceStreamWriteSinkStream`권한 정책을 생성합니다. `### ##`을 애플리케이션 코드를 저장하기 위한 Amazon S3 버킷을 생성하는 데 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(*012345678901*)를 사용자의 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteDeliveryStream",
            "Effect": "Allow",
            "Action": "firehose:*",
            "Resource": "arn:aws:firehose:us-west-2:012345678901:deliverystream/
ExampleDeliveryStream"
        }
    ]
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

> **Note**
>
> AWS SDK for Java를 사용하여 다른 Amazon 서비스에 액세스할 수 있습니다. Managed Service for Apache Flink는 SDK에 필요한 자격 증명을 애플리케이션과 연결된 서비스 실행 IAM 역할의 자격 증명으로 자동 설정합니다. 추가 단계는 필요 없습니다.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한이 없는 경우 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Apache Flink용 Managed Service for Apache Flink 역할을 맡을 권한을 부여합니다. 권한 정책은 Managed Service for Apache Flink가 역할을 맡은 후 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할, 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다. 사용 사례 선택에서 Kinesis Analytics를 선택합니다.

   다음: 권한을 선택합니다.
4. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
5. 역할 생성 페이지에서 역할 이름으로 **`MF-stream-rw-role`**을 입력합니다. Create role(역할 생성)을 선택합니다.

   `MF-stream-rw-role`이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.
6. 역할에 권한 정책을 연결합니다.

   > **Note**
   >
   > 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계 the section called “권한 정책 생성”에서 생성한 정책을 연결합니다.

   a. 요약 페이지에서 권한 탭을 선택합니다.

   b. 정책 연결을 선택합니다.

c. 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.

d. AK ReadSourceStreamWriteSinkStream 정책을 선택하고 Attach policy (Attach policy) 를 선택합니다.

이제 애플리케이션에서 리소스에 액세스하는 데 사용할 서비스 실행 역할을 만들었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

Managed Service for Apache Flink 애플리케이션 생성

1. 다음 JSON 코드를 `create_request.json`이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사를 the section called “종속 리소스 생성”섹션에서 선택한 접미사(`ka-app-code-<username>`)로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(*012345678901*)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        }
      }
    }
}
```

2. 위의 요청과 함께 `CreateApplication` 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "test"
```

```
}
```

2. 다음 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “로깅 설정”.

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationAWS CLI 작업을 사용합니다.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 the section called “종속 리소스 생성” 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "java-getting-started-1.0.jar"
                }
            }
        }
    }
```

```
}
```

AWS 리소스 정리

이 섹션에는 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Stream 삭제
- Kinesis Data Firehose 스트림 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 구성을 선택합니다.
4. 스냅샷 섹션에서 비활성화를 선택한 다음 업데이트를 선택합니다.
5. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Stream 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Firehose 스트림 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Firehose 패널에서 을 선택합니다. ExampleDeliveryStream
3. ExampleDeliveryStream페이지에서 Kinesis Data Firehose 스트림 삭제를 선택한 다음 삭제를 확인합니다.

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- <username>버킷을 선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.
4. Kinesis Data Firehose 스트림의 대상에 대한 Amazon S3 버킷을 생성한 경우 해당 버킷도 삭제합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. Kinesis Data Firehose 스트림에 대한 새 정책을 생성한 경우 해당 정책도 삭제합니다.
7. 탐색 모음에서 역할을 선택합니다.
8. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
9. 역할 삭제를 선택하고 삭제를 확인합니다.
10. Kinesis Data Firehose 스트림에 대한 새 역할을 생성한 경우 해당 역할도 삭제합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: 다른 계정의 Kinesis 스트림에서 읽기를 참조하세요.

이 예제에서는 다른 계정의 Kinesis 스트림에서 데이터를 읽는 Managed Service for Apache Flink 애플리케이션을 만드는 방법을 보여줍니다. 이 예제에서는 소스 Kinesis 스트림에 계정 하나를 사용하고, Managed Service for Apache Flink 애플리케이션 및 싱크 Kinesis 스트림에 두 번째 계정을 사용합니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- 사전 조건
- 설치
- 소스 Kinesis 스트림 생성
- IAM 역할 및 정책 생성 및 업데이트
- Python 스크립트 업데이트
- JAVA 애플리케이션 업데이트
- 애플리케이션 빌드, 업로드 및 실행

사전 조건

- 이 자습서에서는 다른 계정의 Kinesis 스트림에서 데이터를 읽도록 시작하기 예제를 수정합니다. 계속하기 전에 시작하기 (API) DataStream 자습서를 완료하세요.
- 이 자습서를 완료하려면 두 개의 AWS계정이 필요합니다. 하나는 소스 스트림용이고 다른 하나는 애플리케이션과 싱크 스트림용입니다. 시작하기 자습서에서 사용한 AWS계정을 애플리케이션 및 싱크 스트림에 사용하세요. 소스 스트림에는 다른 AWS계정을 사용하세요.

설치

이름이 지정된 프로필을 사용하여 두 AWS계정에 액세스할 수 있습니다. 두 계정의 리전 및 연결 정보가 포함된 프로필 두 개를 포함하도록 AWS 보안 인증 및 구성 파일을 수정하세요.

다음 예제 보안 인증 파일에는 두 개의 명명된 프로필, `ka-source-stream-account-profile` 및 `ka-sink-stream-account-profile`이(가) 포함되어 있습니다. 싱크 스트림 계정에는 시작하기 자습서에서 사용한 계정을 사용합니다.

```
[ka-source-stream-account-profile]
aws_access_key_id=AKIAIOSFODNN7EXAMPLE
aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

[ka-sink-stream-account-profile]
aws_access_key_id=AKIAI44QH8DHBEXAMPLE
aws_secret_access_key=je7MtGbClwBF/2Zp9Utk/h3yCo8nvbEXAMPLEKEY
```

다음 예제 구성 파일에는 리전 및 출력 형식 정보와 함께 이름이 지정된 동일한 프로필이 포함되어 있습니다.

```
[profile ka-source-stream-account-profile]
region=us-west-2
output=json

[profile ka-sink-stream-account-profile]
region=us-west-2
output=json
```

> **Note**
>
> 이 자습서에서는 `ka-sink-stream-account-profile`를 사용하지 않습니다. 프로필을 사용하여 서로 다른 두 AWS 계정에 액세스하는 방법의 예로 들어 있습니다.

AWS CLI에서 명명된 프로필을 사용하는 방법에 대한 자세한 내용은 AWS Command Line Interface 설명서의 명명된 프로필을 참조하세요.

소스 Kinesis 스트림 생성

이 섹션에서는 소스 계정에 Kinesis 스트림을 생성합니다.

애플리케이션에서 입력에 사용할 Kinesis 스트림을 생성하려면 다음 명령을 입력합니다. 참고로 `--profile` 파라미터는 사용할 계정 프로필을 지정합니다.

```
$ aws kinesis create-stream \
--stream-name SourceAccountExampleInputStream \
--shard-count 1 \
--profile ka-source-stream-account-profile
```

IAM 역할 및 정책 생성 및 업데이트

AWS 계정 간 객체 액세스를 허용하려면 소스 계정에서 IAM 역할 및 정책을 생성합니다. 그런 다음 싱크 계정에서 IAM 정책을 수정합니다. IAM 역할과 정책을 생성하고 관리하는 방법에 대한 자세한 내용은 AWS Identity and Access Management 사용 설명서에서 다음 주제를 참조하세요.

- IAM 역할 생성
- IAM 정책 생성

싱크 계정 역할 및 정책

1. 시작하기 자습서에서 kinesis-analytics-service-MyApplication-us-west-2정책을 편집하세요. 이 정책을 사용하면 소스 계정의 역할을 맡아 소스 스트림을 읽을 수 있습니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 만들면 콘솔에서 kinesis-analytics-service-*<application name>*-*<application region>*라는 정책과 kinesisanalytics-*<application name>*-*<application region>*이라는 역할을 생성합니다.

아래에 강조 표시된 섹션을 정책에 추가하세요. 샘플 계정 ID(*SOURCE01234567*)를 소스 스트림에 사용할 계정 ID로 바꾸세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "AssumeRoleInSourceAccount",
            "Effect": "Allow",
            "Action": "sts:AssumeRole",
            "Resource": "arn:aws:iam::SOURCE01234567:role/KA-Source-Stream-Role"
        },
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
```

```
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:SINK012345678:log-group:*"
            ]
        },
        {
            "Sid": "ListCloudwatchLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:SINK012345678:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutCloudwatchLogs",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:SINK012345678:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        }
    ]
}
```

2. kinesis-analytics-MyApplication-us-west-2 역할을 열고 Amazon 리소스 이름(ARN)을 기록해 둡니다. 다음 섹션에서 이 값을 사용하게 됩니다. 역할 ARN 번호는 다음과 같습니다.

```
arn:aws:iam::SINK012345678:role/service-role/kinesis-analytics-MyApplication-us-
west-2
```

소스 계정 역할 및 정책

1. KA-Source-Stream-Policy라는 소스 계정에서 정책을 생성합니다. 정책에는 다음 JSON을 사용합니다. 샘플 계정 번호를 소스 계정의 계정 번호로 바꾸세요.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": [
                "kinesis:DescribeStream",
                "kinesis:GetRecords",
                "kinesis:GetShardIterator",
                "kinesis:ListShards"
            ],
            "Resource":
               "arn:aws:kinesis:us-west-2:SOURCE123456784:stream/
SourceAccountExampleInputStream"
        }
    ]
}
```

2. MF-Source-Stream-Role라는 소스 계정에서 역할을 생성합니다. Managed Flink 사용 사례를 사용하여 역할을 만들려면 다음과 같이 하세요.

   1. IAM Managed Console에서 역할 생성을 선택합니다.
   2. 역할 생성 페이지에서 AWS서비스를 선택합니다. 서비스 목록에서 Kinesis를 선택합니다.
   3. 사용 사례 선택 섹션에서 Managed Service for Apache Flink를 선택합니다.
   4. 다음: 권한을 선택합니다.
   5. 이전 단계에서 만든 KA-Source-Stream-Policy 권한 정책을 추가합니다. 다음: 태그를 선택합니다.
   6. 다음: 검토를 선택합니다.
   7. 역할 이름을 KA-Source-Stream-Role로 지정합니다. 애플리케이션은 이 역할을 사용하여 소스 스트림에 액세스합니다.

3. 싱크 계정의 kinesis-analytics-MyApplication-us-west-2 ARN을 소스 계정 내 KA-Source-Stream-Role 역할의 신뢰 관계에 추가합니다.

   1. IAM 콘솔에서 KA-Source-Stream-Role를 엽니다.
   2. 신뢰 관계(Trust Relationships) 탭을 선택합니다.
   3. 신뢰 관계 편집(Edit trust relationship)을 선택합니다.

4. 신뢰 관계에 대해 다음 코드를 사용합니다. 샘플 계정 ID(*SINK012345678*)를 싱크 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
      {
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::SINK012345678:role/service-role/kinesis-analytics-
MyApplication-us-west-2"
        },
        "Action": "sts:AssumeRole"
      }
    ]
}
```

Python 스크립트 업데이트

이 섹션에서는 소스 계정 프로필을 사용하도록 샘플 데이터를 생성하는 Python 스크립트를 업데이트합니다.

다음과 같이 강조 표시된 변경 사항으로 `stock.py` 스크립트를 업데이트하세요.

```
import json
import boto3
import random
import datetime
import os

os.environ['AWS_PROFILE'] ='ka-source-stream-account-profile'
os.environ['AWS_DEFAULT_REGION'] = 'us-west-2'


kinesis = boto3.client('kinesis')
def getReferrer():
    data = {}
    now = datetime.datetime.now()
    str_now = now.isoformat()
    data['event_time'] = str_now
    data['ticker'] = random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV'])
```

```
    price = random.random() * 100
    data['price'] = round(price, 2)
    return data

while True:
        data = json.dumps(getReferrer())
        print(data)
        kinesis.put_record(
                StreamName="SourceAccountExampleInputStream",
                Data=data,
                PartitionKey="partitionkey")
```

JAVA 애플리케이션 업데이트

이 섹션에서는 소스 스트림에서 읽을 때 소스 계정 역할을 맡도록 Java 애플리케이션 코드를 업데이트합니다.

`BasicStreamingJob.java` 파일을 다음과 같이 변경합니다. 예제 소스 계정 번호(*SOURCE01234567*)를 소스 계정 번호로 바꾸세요.

```
package com.amazonaws.services.managed-flink;

import com.amazonaws.services.managed-flink.runtime.KinesisAnalyticsRuntime;
import org.apache.flink.api.common.serialization.SimpleStringSchema;
import org.apache.flink.streaming.api.datastream.DataStream;
import org.apache.flink.streaming.api.environment.StreamExecutionEnvironment;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer;
import org.apache.flink.streaming.connectors.kinesis.FlinkKinesisProducer;
import org.apache.flink.streaming.connectors.kinesis.config.ConsumerConfigConstants;
import org.apache.flink.streaming.connectors.kinesis.config.AWSConfigConstants;

import java.io.IOException;
import java.util.Map;
import java.util.Properties;

 /**
 * A basic Managed Service for Apache Flink for Java application with Kinesis data
 streams
 * as source and sink.
 */
public class BasicStreamingJob {
    private static final String region = "us-west-2";
    private static final String inputStreamName = "SourceAccountExampleInputStream";
```

```
    private static final String outputStreamName = ExampleOutputStream;
    private static final String roleArn = "arn:aws:iam::SOURCE01234567:role/KA-Source-
Stream-Role";
    private static final String roleSessionName = "ksassumedrolesession";

    private static DataStream<String>
 createSourceFromStaticConfig(StreamExecutionEnvironment env) {
        Properties inputProperties = new Properties();
        inputProperties.setProperty(AWSConfigConstants.AWS_CREDENTIALS_PROVIDER,
 "ASSUME_ROLE");
        inputProperties.setProperty(AWSConfigConstants.AWS_ROLE_ARN, roleArn);
        inputProperties.setProperty(AWSConfigConstants.AWS_ROLE_SESSION_NAME,
 roleSessionName);
        inputProperties.setProperty(ConsumerConfigConstants.AWS_REGION, region);
        inputProperties.setProperty(ConsumerConfigConstants.STREAM_INITIAL_POSITION,
 "LATEST");

        return env.addSource(new FlinkKinesisConsumer<>(inputStreamName, new
 SimpleStringSchema(), inputProperties));
    }

    private static KinesisStreamsSink<String> createSinkFromStaticConfig() {
        Properties outputProperties = new Properties();
        outputProperties.setProperty(AWSConfigConstants.AWS_REGION, region);

        return KinesisStreamsSink.<String>builder()
                .setKinesisClientProperties(outputProperties)
                .setSerializationSchema(new SimpleStringSchema())
                .setStreamName(outputProperties.getProperty("OUTPUT_STREAM",
 "ExampleOutputStream"))
                .setPartitionKeyGenerator(element ->
 String.valueOf(element.hashCode()))
                .build();
    }

    public static void main(String[] args) throws Exception {
        // set up the streaming execution environment
        final StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();

        DataStream<String> input = createSourceFromStaticConfig(env);

        input.addSink(createSinkFromStaticConfig());
```

```
        env.execute("Flink Streaming Java API Skeleton");
    }
}
```

애플리케이션 빌드, 업로드 및 실행

애플리케이션을 업데이트하고 실행하려면 다음을 수행합니다.

1. pom.xml 파일이 있는 디렉토리에서 다음 명령을 실행하여 애플리케이션을 다시 빌드합니다.

   ```
   mvn package -Dflink.version=1.15.3
   ```

2. Amazon Simple Storage Service(S3) 버킷에서 이전 JAR 파일을 삭제한 후 새 aws-kinesis-analytics-java-apps-1.0.jar파일을 S3 버킷으로 업로드합니다.
3. Managed Service for Apache Flink 콘솔의 애플리케이션 페이지에서 구성, 업데이트를 선택하여 애플리케이션 JAR 파일을 다시 로드합니다.
4. stock.py 스크립트를 실행하여 소스 스트림으로 데이터를 전송합니다.

   ```
   python stock.py
   ```

이제 애플리케이션은 다른 계정의 Kinesis 스트림에서 데이터를 읽습니다.

애플리케이션이 작동하는지 확인하려면 ExampleOutputStream 스트림의 PutRecords.Bytes 지표를 확인하면 됩니다. 출력 스트림에 활동이 있는 경우 애플리케이션이 제대로 작동하는 것입니다.

자습서: Amazon MSK에서 사용자 지정 트러스트 스토어 사용

현재 데이터 소스 API

현재 데이터 소스 API를 사용하는 경우 애플리케이션은 여기에 설명된 MSK Config Providers 유틸리티를 활용할 수 있습니다. 이렇게 하면 KafkaSource 함수가 Amazon S3의 상호 TLS를 위해 키스토어와 신뢰 저장소에 액세스할 수 있습니다.

```
...
// define names of config providers:
builder.setProperty("config.providers", "secretsmanager,s3import");

// provide implementation classes for each provider:
```

```
builder.setProperty("config.providers.secretsmanager.class",
 "com.amazonaws.kafka.config.providers.SecretsManagerConfigProvider");
builder.setProperty("config.providers.s3import.class",
 "com.amazonaws.kafka.config.providers.S3ImportConfigProvider");

String region = appProperties.get(Helpers.S3_BUCKET_REGION_KEY).toString();
String keystoreS3Bucket = appProperties.get(Helpers.KEYSTORE_S3_BUCKET_KEY).toString();
String keystoreS3Path = appProperties.get(Helpers.KEYSTORE_S3_PATH_KEY).toString();
String truststoreS3Bucket =
 appProperties.get(Helpers.TRUSTSTORE_S3_BUCKET_KEY).toString();
String truststoreS3Path = appProperties.get(Helpers.TRUSTSTORE_S3_PATH_KEY).toString();
String keystorePassSecret =
 appProperties.get(Helpers.KEYSTORE_PASS_SECRET_KEY).toString();
String keystorePassSecretField =
 appProperties.get(Helpers.KEYSTORE_PASS_SECRET_FIELD_KEY).toString();

// region, etc..
builder.setProperty("config.providers.s3import.param.region", region);

// properties
builder.setProperty("ssl.truststore.location", "${s3import:" + region + ":" +
 truststoreS3Bucket + "/" + truststoreS3Path + "}");
builder.setProperty("ssl.keystore.type", "PKCS12");
builder.setProperty("ssl.keystore.location", "${s3import:" + region + ":" +
 keystoreS3Bucket + "/" + keystoreS3Path + "}");
builder.setProperty("ssl.keystore.password", "${secretsmanager:" + keystorePassSecret +
 ":" + keystorePassSecretField + "}");
builder.setProperty("ssl.key.password", "${secretsmanager:" + keystorePassSecret + ":"
 + keystorePassSecretField + "}");
...
```

자세한 내용 및 안내는 여기에서 확인할 수 있습니다.

레거시 API SourceFunction

레거시 SourceFunction API를 사용하는 경우 애플리케이션은 메서드를 재정의하는 사용자 지정 직렬화 및 역직렬화 스키마를 사용하여 사용자 지정 신뢰 저장소를 로드합니다open. 이렇게 하면 애플리케이션이 다시 시작되거나 스레드를 교체한 후에 애플리케이션에서 트러스트 스토어를 사용할 수 있습니다.

사용자 지정 트러스트 스토어는 다음 코드를 사용하여 검색 및 저장됩니다.

```
public static void initializeKafkaTruststore() {
```

```
    ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
    URL inputUrl = classLoader.getResource("kafka.client.truststore.jks");
    File dest = new File("/tmp/kafka.client.truststore.jks");

    try {
        FileUtils.copyURLToFile(inputUrl, dest);
    } catch (Exception ex) {
        throw new FlinkRuntimeException("Failed to initialize Kakfa truststore", ex);
    }
}
```

> **Note**
>
> Apache Flink를 사용하려면 트러스트 스토어가 JKS 형식이어야 합니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (API) DataStream 연습을 완료하세요.

다음 자습서에서는 사용자 지정, 사설 또는 자체 호스팅 CA(인증 기관)에서 발급한 서버 인증서를 사용하는 Kafka 클러스터에 안전하게 연결(전송 중 암호화)하는 방법을 설명합니다.

Kafka 클라이언트 (예: Flink 애플리케이션) 는 TLS를 통해 Kafka 클러스터에 안전하게 연결하기 위해 Kafka 클러스터 서버 인증서(예: 발급 CA에서 루트 수준 CA까지)가 제공하는 완전한 신뢰 체인을 신뢰해야 합니다. 사용자 지정 트러스트 스토어의 예로 상호 TLS (MTLS) 인증이 활성화된 Amazon MSK 클러스터를 사용하겠습니다. 이는 MSK 클러스터 노드가 AWS Certificate Manager 사설 인증 기관 (ACM 사설 CA) 에서 발급한 서버 인증서를 사용한다는 것을 의미합니다. 이 인증서는 사용자 계정 및 리전 전용이므로 Flink 애플리케이션을 실행하는 JVM (Java Virtual Machine)의 기본 트러스트 스토어에서 신뢰하지 않습니다.

> **Note**
>
> - 키스토어는 애플리케이션이 검증을 위해 서버나 클라이언트 모두에 제공해야 하는 개인 키와 ID 인증서를 저장하는 데 사용됩니다.
> - 트러스트 스토어는 SSL 연결을 통해 서버가 제공한 인증서를 확인하는 인증 기관(CA)의 인증서를 저장하는 데 사용됩니다.

이 자습서의 기법은 다음과 같이 Managed Service for Apache Flink 애플리케이션과 다른 Apache Kafka 소스 간의 상호 작용에도 사용할 수 있습니다.

- AWS(Amazon EC2 또는 Amazon EKS)에서 호스팅되는 사용자 정의 Apache Kafka 클러스터
- AWS에서 호스팅되는 Confluent Kafka 클러스터
- AWS Direct Connect 또는 VPN을 통해 액세스하는 온프레미스 Kafka 클러스터

이 자습서는 다음 섹션을 포함하고 있습니다:

- Amazon MSK 클러스터로 VPC 생성
- 사용자 지정 트러스트 스토어를 생성하여 클러스터에 적용
- 애플리케이션 코드 생성
- Apache Flink 스트리밍 Java 코드 업로드
- 애플리케이션 생성
- 애플리케이션 구성
- 애플리케이션 실행
- 애플리케이션 테스트

Amazon MSK 클러스터로 VPC 생성

Managed Service for Apache Flink 애플리케이션에서 액세스할 샘플 VPC 및 Amazon MSK 클러스터를 생성하려면 Amazon MSK 사용 시작하기 자습서를 따르세요.

자습서를 완료할 때는 다음 작업도 수행하세요.

- 3단계: 주제 만들기에서 `kafka-topics.sh --create` 명령을 반복하여 `AWSKafkaTutorialTopicDestination`라는 이름의 대상 주제를 생성합니다.

```
bin/kafka-topics.sh --create --bootstrap-server ZooKeeperConnectionString --
replication-factor 3 --partitions 1 --topic AWSKafkaTutorialTopicDestination
```

> **Note**
>
> `kafka-topics.sh` 명령이 `ZooKeeperClientTimeoutException`를 반환하는 경우 Kafka 클러스터의 보안 그룹에 클라이언트 인스턴스의 프라이빗 IP 주소에서 들어오는 모든 트래픽을 허용하는 인바운드 규칙이 있는지 확인하세요.

- 클러스터의 부트스트랩 서버 목록을 기록합니다. 다음 명령을 사용하여 부트스트랩 서버 목록을 가져올 수 있습니다 (MSK 클러스터의 *ClusterArn*ARN으로 대체).

```
aws kafka get-bootstrap-brokers --region us-west-2 --cluster-arn ClusterArn
{...
    "BootstrapBrokerStringTls": "b-2.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-1.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094,b-3.awskafkatutorialcluste.t79r6y.c4.kafka.us-
west-2.amazonaws.com:9094"
}
```

- 이 자습서와 사전 조건 자습서의 단계를 따를 때는 코드, 명령 및 콘솔 항목에서 선택한 AWS 리전을 사용해야 합니다.

사용자 지정 트러스트 스토어를 생성하여 클러스터에 적용

이 섹션에서는 사용자 지정 CA(인증 기관)를 만들고, 이를 사용하여 사용자 지정 트러스트 스토어를 생성하고, 이를 MSK 클러스터에 적용합니다.

사용자 지정 트러스트 스토어를 생성하고 적용하려면 Amazon Managed Streaming for Apache Kafka 개발자 안내서의 클라이언트 인증 자습서를 따르세요.

애플리케이션 코드 생성

이 섹션에서는 애플리케이션 JAR 파일을 다운로드하고 컴파일합니다.

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. 애플리케이션 코드는 `amazon-kinesis-data-analytics-java-examples/CustomKeystore`에 있습니다. 코드를 검토하여 Managed Service for Apache Flink 코드의 구조를 익힐 수 있습니다.
4. 명령줄 Maven 도구 또는 원하는 개발 환경을 사용하여 JAR 파일을 만드세요. 명령줄 Maven 도구를 사용하여 JAR 파일을 컴파일하려면 다음을 입력합니다.

```
mvn package -Dflink.version=1.15.3
```

빌드가 성공하면 다음 파일이 생성됩니다.

```
target/flink-app-1.0-SNAPSHOT.jar
```

> **Note**
>
> 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 시작하기 (API) DataStream 자습서에서 생성한 Amazon S3 버킷으로 애플리케이션 코드를 업로드합니다.

> **Note**
>
> 시작 자습서에서 Amazon S3 버킷을 삭제한 경우 the section called “Apache Flink 스트리밍 Java 코드 업로드” 단계를 다시 수행하세요.

1. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
2. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `flink-app-1.0-SNAPSHOT.jar` 파일로 이동합니다.
3. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:

- 애플리케이션 명칭에 **MyApplication**을 입력합니다.
- 런타임에서 Apache Flink 버전 1.15.2를 선택합니다.

4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink를 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **flink-app-1.0-SNAPSHOT.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션 리소스(예: 로그 또는 VPC)를 지정하면 콘솔은 애플리케이션 실행 역할을 수정하여 해당 리소스에 액세스할 권한을 부여합니다.

4. 속성에서 그룹 추가를 선택합니다. 다음 속성을 입력합니다.

그룹 ID	키	값
KafkaSource	주제	AWSKafkaTutorialTopic
KafkaSource	bootstrap.servers	### ### ##### ## ##
KafkaSource	security.protocol	SSL
KafkaSource	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSource	ssl.truststore.password	changeit

> **Note**
>
> 기본 인증서의 ssl.truststore.password는 “changeit”입니다. 기본 인증서를 사용하는 경우에는 이 값을 변경할 필요가 없습니다.

그룹 추가를 다시 선택합니다. 다음 속성을 입력합니다.

그룹 ID	키	값
KafkaSink	주제	AWSKafkaTutorialTopicDestination
KafkaSink	bootstrap.servers	### ### ##### ## ##
KafkaSink	security.protocol	SSL
KafkaSink	ssl.truststore.location	/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
KafkaSink	ssl.truststore.password	changeit
KafkaSink	transaction.timeout.ms	1000

애플리케이션 코드는 위의 애플리케이션 속성을 읽고 VPC 및 Amazon MSK 클러스터와 상호 작용하는 데 사용되는 소스 및 싱크를 구성합니다. 클러스터 속성 사용에 대한 자세한 내용은 런타임 속성을(를) 참조하세요.

5. 스냅샷에서 비활성화를 선택합니다. 이렇게 하면 잘못된 애플리케이션 상태 데이터를 로드하지 않고도 애플리케이션을 더 쉽게 업데이트할 수 있습니다.
6. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
7. CloudWatch 로깅하려면 활성화 확인란을 선택합니다.
8. Virtual Private Cloud(VPC) 섹션에서 애플리케이션과 연결할 VPC를 선택합니다. 애플리케이션이 VPC 리소스에 액세스하는 데 사용할 VPC와 연결된 서브넷 및 보안 그룹을 선택합니다.
9. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다.

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 테스트

이 섹션에서는 소스 주제에 레코드를 쓰기 합니다. 애플리케이션은 소스 주제에서 레코드를 읽고 대상 주제에 쓰기 합니다. 소스 주제에 레코드를 쓰고 대상 주제에서 레코드를 읽어 애플리케이션이 작동 중인지 확인합니다.

주제에서 레코드를 쓰고 읽으려면 Amazon MSK 사용 시작하기 자습서의 6단계: 데이터 생성 및 소비의 단계를 따르세요.

대상 주제에서 읽으려면 클러스터에 대한 두 번째 연결에서 소스 주제 대신 대상 주제 이름을 사용하세요.

```
bin/kafka-console-consumer.sh --bootstrap-server BootstrapBrokerString --
consumer.config client.properties --topic AWSKafkaTutorialTopicDestination --from-
beginning
```

대상 주제에 레코드가 표시되지 않는 경우 문제 해결 주제의 VPC에서 리소스에 액세스 불가능 섹션을 참조하세요.

Python 예제

다음 예제는 Apache Flink Table API와 함께 Python을 사용하여 애플리케이션을 만드는 방법을 보여줍니다.

주제

- 예: Python에서 텀블링 윈도우 만들기
- 예: Python에서 슬라이딩 윈도우 만들기
- 예: Python에서 Amazon S3로 스트리밍 데이터 보내기

예: Python에서 텀블링 윈도우 만들기

이 연습에서는 텀블링 창을 사용하여 데이터를 집계하는 Python Managed Service for Apache Flink 애플리케이션을 생성합니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (Python) 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- 애플리케이션 코드 다운로드 및 검토
- Apache Flink 스트리밍 Python 코드 압축 및 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행

- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 다음과 같은 종속 리소스를 생성해야 합니다.

- 두 개의 Kinesis Data Streams(`ExampleInputStream` 및 `ExampleOutputStream`)
- 애플리케이션 코드를 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **`ExampleInputStream`** 및 **`ExampleOutputStream`**에 명칭을 지정합니다.
- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **`ka-app-code-<username>`**)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

> **Note**
>
> 이 섹션의 Python 스크립트는 AWS CLI를 사용합니다. 사용자의 계정 보안 인증과 기본 리전을 사용하도록 사용자의 AWS CLI을(를) 구성해야 합니다. 다음 명령을 입력하여 사용자의 AWS CLI를 구성합니다.
>
> ```
> aws configure
> ```

1. 다음 콘텐츠를 가진 stock.py이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        'event_time': datetime.datetime.now().isoformat(),
        'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
        'price': round(random.random() * 100, 2)}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(data),
            PartitionKey="partitionkey")


if __name__ == '__main__':
    generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. stock.py 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Python 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. amazon-kinesis-data-analytics-java-examples/python/TumblingWindow 디렉터리로 이동합니다.

애플리케이션 코드는 tumbling-windows.py 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- 애플리케이션은 Kinesis 테이블 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 create_table 함수를 호출하여 Kinesis 테이블 소스를 생성합니다.

```
table_env.execute_sql(
        create_input_table(input_table_name, input_stream, input_region,
 stream_initpos)
    )
```

  이 create_table 함수는 SQL 명령을 사용하여 스트리밍 소스가 지원하는 테이블을 생성합니다.

```
def create_input_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
                ticker VARCHAR(6),
                price DOUBLE,
                event_time TIMESTAMP(3),
                WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
              )
              PARTITIONED BY (ticker)
              WITH (
                'connector' = 'kinesis',
                'stream' = '{1}',
                'aws.region' = '{2}',
                'scan.stream.initpos' = '{3}',
                'format' = 'json',
                'json.timestamp-format.standard' = 'ISO-8601'
              ) """.format(table_name, stream_name, region, stream_initpos)
```

- 애플리케이션은 Tumble 연산자를 사용하여 지정된 텀블링 윈도우 내에서 레코드를 집계하고 집계된 레코드를 테이블 객체로 반환합니다.

```
tumbling_window_table = (
        input_table.window(
            Tumble.over("10.seconds").on("event_time").alias("ten_second_window")
        )
        .group_by("ticker, ten_second_window")
        .select("ticker, price.min as price, to_string(ten_second_window.end) as
 event_time")
```

- 애플리케이션은 flink-sql-connector-kinesis-1.15.2.jar의 Kinesis Flink 커넥터를 사용합니다.

Apache Flink 스트리밍 Python 코드 압축 및 업로드

이 섹션에서는 종속 리소스 생성 섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

1. 선호하는 압축 애플리케이션을 사용하여 `tumbling-windows.py` 및 `flink-sql-connector-kinesis-1.15.2.jar` 파일을 압축합니다. 아카이브 이름은 `myapp.zip`라고 짓습니다.
2. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
3. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `myapp.zip` 파일로 이동합니다.
4. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.

- 런타임에서 Apache Flink를 선택합니다.

> **Note**
>
> Managed Service for Apache Flink는 Apache Flink 버전 1.15.2를 사용합니다.

- 버전 폴다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.

4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **myapp.zip**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 추가를 선택합니다.
5. 다음을 입력합니다:

그룹 ID	키	값
consumer.config.0	**input.stream.name**	**ExampleInputStream**
consumer.config.0	**aws.region**	**us-west-2**
consumer.config.0	**scan.stream.initpos**	**LATEST**

저장을 선택합니다.

6. 속성에서 그룹 추가를 다시 선택합니다.
7. 다음을 입력합니다:

그룹 ID	키	값
producer.config.0	**output.stream.name**	**ExampleOutputStream**
producer.config.0	**aws.region**	**us-west-2**
producer.config.0	**shard.count**	**1**

8. 속성에서 그룹 추가를 다시 선택합니다. 역할 이름에 **kinesis.analytics.flink.run.options**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 설명은 코드 파일 지정 섹션을 참조하세요.
9. 다음을 입력합니다:

그룹 ID	키	값
kinesis.analytics.flink.run.options	**python**	**tumbling-windows.py**
kinesis.analytics.flink.run.options	**jarfile**	**flink-sql-connector-kinesis-1.15.2.jar**

10. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.

11. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
12. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다. 이 로그 스트림은 애플리케이션이 결과를 전송하는 데 사용하는 로그 스트림과 다릅니다.

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
```

```
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
```

```
}
```

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

CloudWatch 콘솔에서 Apache Flink용 관리 서비스 메트릭을 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

AWS 리소스 정리

이 섹션에는 시작하기 텀블링 윈도우 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: Python에서 슬라이딩 윈도우 만들기

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (Python) 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기

- 애플리케이션 코드 다운로드 및 검토
- Apache Flink 스트리밍 Python 코드 압축 및 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 다음과 같은 종속 리소스를 생성해야 합니다.

- 두 개의 Kinesis Data Streams(`ExampleInputStream` 및 `ExampleOutputStream`)
- 애플리케이션 코드를 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **ExampleInputStream** 및 **ExampleOutputStream**에 명칭을 지정합니다.
- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

> **Note**
>
> 이 섹션의 Python 스크립트는 AWS CLI를 사용합니다. 사용자의 계정 보안 인증과 기본 리전을 사용하도록 사용자의 AWS CLI을(를) 구성해야 합니다. 다음 명령을 입력하여 사용자의 AWS CLI를 구성합니다.
>
> ```
> aws configure
> ```

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        'event_time': datetime.datetime.now().isoformat(),
        'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
        'price': round(random.random() * 100, 2)}


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(data),
            PartitionKey="partitionkey")


if __name__ == '__main__':
    generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. `stock.py` 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Python 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/>amazon-kinesis-data-analytics-java-
examples
```

3. `amazon-kinesis-data-analytics-java-examples/python/SlidingWindow` 디렉터리로 이동합니다.

애플리케이션 코드는 `sliding-windows.py` 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- 애플리케이션은 Kinesis 테이블 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 `create_input_table` 함수를 호출하여 Kinesis 테이블 소스를 생성합니다.

```
table_env.execute_sql(
        create_input_table(input_table_name, input_stream, input_region,
 stream_initpos)
    )
```

이 `create_input_table` 함수는 SQL 명령을 사용하여 스트리밍 소스가 지원하는 테이블을 생성합니다.

```
def create_input_table(table_name, stream_name, region, stream_initpos):
    return """ CREATE TABLE {0} (
                ticker VARCHAR(6),
                price DOUBLE,
                event_time TIMESTAMP(3),
                WATERMARK FOR event_time AS event_time - INTERVAL '5' SECOND
```

```
            )
            PARTITIONED BY (ticker)
            WITH (
              'connector' = 'kinesis',
              'stream' = '{1}',
              'aws.region' = '{2}',
              'scan.stream.initpos' = '{3}',
              'format' = 'json',
              'json.timestamp-format.standard' = 'ISO-8601'
            ) """.format(table_name, stream_name, region, stream_initpos)
}
```

- 애플리케이션은 `Slide` 연산자를 사용하여 지정된 슬라이딩 윈도우 내에서 레코드를 집계하고 집계된 레코드를 테이블 객체로 반환합니다.

```
sliding_window_table = (
        input_table
            .window(
                Slide.over("10.seconds")
                .every("5.seconds")
                .on("event_time")
                .alias("ten_second_window")
            )
            .group_by("ticker, ten_second_window")
            .select("ticker, price.min as price, to_string(ten_second_window.end) as
 event_time")
    )
```

- 애플리케이션은 -1.15.2.jar 파일의 Kinesis Flink 커넥터를 사용합니다. flink-sql-connector-kinesis

Apache Flink 스트리밍 Python 코드 압축 및 업로드

이 섹션에서는 종속 리소스 생성 섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

이 섹션에서는 Python 애플리케이션을 패키징하는 방법을 설명합니다.

1. 선호하는 압축 애플리케이션을 사용하여 `sliding-windows.py` 및 `flink-sql-connector-kinesis-1.15.2.jar`파일을 압축합니다. 아카이브 이름은 `myapp.zip`라고 짓습니다.
2. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.

3. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `myapp.zip` 파일로 이동합니다.
4. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.

     > **Note**
     >
     > Managed Service for Apache Flink는 Apache Flink 버전 1.15.2를 사용합니다.

   - 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*

- 역할: kinesisanalytics-*MyApplication*-*us-west-2*

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **myapp.zip**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 추가를 선택합니다.
5. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
consumer.config.0	**input.stream.name**	**ExampleInputStream**
consumer.config.0	**aws.region**	**us-west-2**
consumer.config.0	**scan.stream.initpos**	**LATEST**

   저장을 선택합니다.
6. 속성에서 그룹 추가를 다시 선택합니다.
7. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
producer.config.0	**output.stream.name**	**ExampleOutputStream**
producer.config.0	**aws.region**	**us-west-2**
producer.config.0	**shard.count**	**1**

8. 속성에서 그룹 추가를 다시 선택합니다. 역할 이름에 **kinesis.analytics.flink.run.options**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 설명은 코드 파일 지정 섹션을 참조하세요.
9. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
kinesis.analytics.flink.run.options	**python**	**sliding-windows.py**
kinesis.analytics.flink.run.options	**jarfile**	**flink-sql-connector-kinesis_1.15.2.jar**

10. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
11. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
12. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다. 이 로그 스트림은 애플리케이션이 결과를 전송하는 데 사용하는 로그 스트림과 다릅니다.

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.

2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
```

```
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

CloudWatch 콘솔에서 Apache Flink용 관리 서비스 메트릭을 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

AWS 리소스 정리

이 섹션에는 슬라이딩 윈도우 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

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

2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: Python에서 Amazon S3로 스트리밍 데이터 보내기

이 연습에서는 Amazon Simple Storage Service 싱크로 데이터를 스트리밍하는 Python Managed Service for Apache Flink 애플리케이션을 생성합니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기 (Python) 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 종속 리소스 생성
- 샘플 레코드를 입력 스트림에 쓰기
- 애플리케이션 코드 다운로드 및 검토
- Apache Flink 스트리밍 Python 코드 압축 및 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행
- AWS 리소스 정리

종속 리소스 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 다음과 같은 종속 리소스를 생성해야 합니다.

- Kinesis Data Streams(`ExampleInputStream`).
- 애플리케이션 코드와 출력을 저장할 Amazon S3 버킷(`ka-app-code-<username>`)

> **Note**
>
> Managed Service for Apache Flink는 Managed Service for Apache Flink에서 서버 측 암호화가 활성화된 상태에서는 Amazon S3에 데이터를 쓸 수 없습니다.

콘솔을 사용하여 Kinesis 스트림과 Amazon S3 버킷을 만들 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하세요.

- Amazon Kinesis Data Streams 개발자 안내서의 데이터 스트림 만들기 및 업데이트. 데이터 스트림 **ExampleInputStream**의 이름을 지정합니다.
- Amazon Simple Storage Service 사용 설명서의 S3 버킷을 생성하려면 어떻게 해야 합니까?를 참조하세요. 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

> **Note**
>
> 이 섹션의 Python 스크립트는 AWS CLI를 사용합니다. 사용자의 계정 보안 인증과 기본 리전을 사용하도록 사용자의 AWS CLI을(를) 구성해야 합니다. 다음 명령을 입력하여 사용자의 AWS CLI를 구성합니다.
>
> ```
> aws configure
> ```

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"
```

```
    def get_data():
        return {
            'event_time': datetime.datetime.now().isoformat(),
            'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
            'price': round(random.random() * 100, 2)}


    def generate(stream_name, kinesis_client):
        while True:
            data = get_data()
            print(data)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey")


    if __name__ == '__main__':
        generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. `stock.py` 스크립트를 실행합니다.

```
$ python stock.py
```

자습서의 나머지 부분을 완료하는 동안 스크립트가 계속 돌아가게 둡니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Python 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. `amazon-kinesis-data-analytics-java-examples/python/S3Sink` 디렉터리로 이동합니다.

애플리케이션 코드는 `streaming-file-sink.py` 파일에 있습니다. 애플리케이션 코드에 대해 다음을 유의하십시오:

- 애플리케이션은 Kinesis 테이블 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 `create_source_table` 함수를 호출하여 Kinesis 테이블 소스를 생성합니다.

```
table_env.execute_sql(
        create_source_table(input_table_name, input_stream, input_region,
 stream_initpos)
    )
```

이 `create_source_table` 함수는 SQL 명령을 사용하여 스트리밍 소스가 지원하는 테이블을 생성합니다.

```
import datetime
    import json
    import random
    import boto3

    STREAM_NAME = "ExampleInputStream"


    def get_data():
        return {
            'event_time': datetime.datetime.now().isoformat(),
            'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
            'price': round(random.random() * 100, 2)}


    def generate(stream_name, kinesis_client):
        while True:
            data = get_data()
            print(data)
            kinesis_client.put_record(
                StreamName=stream_name,
                Data=json.dumps(data),
                PartitionKey="partitionkey")


    if __name__ == '__main__':
        generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

- 애플리케이션은 `filesystem` 커넥터를 사용하여 레코드를 Amazon S3 버킷으로 전송합니다.

```
def create_sink_table(table_name, bucket_name):
    return """ CREATE TABLE {0} (
                ticker VARCHAR(6),
                price DOUBLE,
                event_time VARCHAR(64)
              )
              PARTITIONED BY (ticker)
              WITH (
                  'connector'='filesystem',
                  'path'='s3a://{1}/',
                  'format'='json',
                  'sink.partition-commit.policy.kind'='success-file',
                  'sink.partition-commit.delay' = '1 min'
              ) """.format(table_name, bucket_name)
```

- 애플리케이션은 -1.15.2.jar 파일의 Kinesis Flink 커넥터를 사용합니다. flink-sql-connector-kinesis

Apache Flink 스트리밍 Python 코드 압축 및 업로드

이 섹션에서는 종속 리소스 생성 섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

1. 선호하는 압축 애플리케이션을 사용하여 및 -1.15.2.jar 파일을 압축합니다. `streaming-file-sink.py` flink-sql-connector-kinesis 아카이브 이름은 `myapp.zip`라고 짓습니다.
2. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
3. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `myapp.zip` 파일로 이동합니다.
4. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.

2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **MyApplication**을 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.

   > **Note**
   > Managed Service for Apache Flink는 Apache Flink 버전 1.15.2를 사용합니다.

   - 버전 풀다운은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두세요.
4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **myapp.zip**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.

4. 속성에서 그룹 추가를 선택합니다.
5. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
consumer.config.0	**input.stream.name**	**ExampleInputStream**
consumer.config.0	**aws.region**	**us-west-2**
consumer.config.0	**scan.stream.initpos**	**LATEST**

저장을 선택합니다.

6. 속성에서 그룹 추가를 다시 선택합니다. 역할 이름에 **kinesis.analytics.flink.run.options**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 설명은 코드 파일 지정 섹션을 참조하세요.
7. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
kinesis.analytics.flink.run.options	**python**	**streaming-file-sink.py**
kinesis.analytics.flink.run.options	**jarfile**	**S3Sink/lib/flink-sql-connector-kinesis-1.15.2.jar**

8. 속성에서 그룹 추가를 다시 선택합니다. 역할 이름에 **sink.config.0**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 설명은 코드 파일 지정 섹션을 참조하세요.
9. 다음 애플리케이션 속성 및 값을 입력합니다(*bucket-name*을 실제 Amazon S3 버킷의 이름으로 바꿉니다).

그룹 ID	키	값
sink.config.0	**output.bucket.name**	***bucket-name***

10. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
11. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
12. 업데이트를 선택합니다.

> **Note**
>
> CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`
>
> 이 로그 스트림은 애플리케이션을 모니터링하는 데 사용됩니다. 이 로그 스트림은 애플리케이션이 결과를 전송하는 데 사용하는 로그 스트림과 다릅니다.

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **`kinesis-analytics-service-MyApplication-us-west-2`** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "logs:DescribeLogGroups",
                "s3:GetObjectVersion"
            ],
```

```
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*",
                "arn:aws:s3:::ka-app-code-<username>/myapp.zip"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": "logs:DescribeLogStreams",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:*"
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": "logs:PutLogEvents",
            "Resource": "arn:aws:logs:us-west-2:012345678901:log-group:/aws/
kinesis-analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
        },
        {
            "Sid": "ListCloudwatchLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteObjects",
            "Effect": "Allow",
            "Action": [
                "s3:Abort*",
                "s3:DeleteObject*",
                "s3:GetObject*",
                "s3:GetBucket*",
```

```
                "s3:List*",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-<username>",
                "arn:aws:s3:::ka-app-code-<username>/*"
            ]
        }
    ]
}
```

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

CloudWatch 콘솔에서 Apache Flink용 관리 서비스 메트릭을 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

AWS 리소스 정리

이 섹션에는 슬라이딩 윈도우 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Stream 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Stream 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

Scala 예제

다음 예제는 Apache Flink와 함께 Scala를 사용하여 애플리케이션을 만드는 방법을 보여줍니다.

주제

예: Scala에서 텀블링 윈도우 만들기

> **Note**
>
> Flink 버전 1.15부터 Scala는 무료입니다. 이제 애플리케이션은 모든 Scala 버전에서 Java API를 사용할 수 있습니다. Flink는 여전히 내부적으로 몇 가지 주요 구성 요소에서 Scala를 사용하지만 사용자 코드 클래스 로더에 Scala를 노출하지는 않습니다. 따라서 사용자는 자신의 jar-acrchive에 Scala 종속성을 추가해야 합니다.
> Flink 1.15의 Scala 변경 사항에 대한 자세한 내용은 Scala Free in One Fifteen을 참조하세요.

이 연습에서는 Scala 3.2.0과 Flink의 Java API를 사용하는 간단한 스트리밍 애플리케이션을 만들어 보겠습니다. DataStream 애플리케이션은 Kinesis 스트림에서 데이터를 읽고, 슬라이딩 윈도우를 사용하여 데이터를 집계하고, 결과를 Kinesis 스트림에 기록합니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기(Scala) 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

애플리케이션 코드 다운로드 및 검토

이 예제의 Python 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. `amazon-kinesis-data-analytics-java-examples/scala/TumblingWindow` 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- `build.sbt` 파일에는 Managed Service for Apache Flink 라이브러리를 포함하여 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- `BasicStreamingJob.scala` 파일에는 애플리케이션의 기능을 정의하는 주요 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
 defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

또한 애플리케이션은 Kinesis 싱크를 사용하여 결과 스트림에 기록합니다. 다음 조각은 Kinesis 싱크를 생성합니다.

```
private def createSink: KinesisStreamsSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val outputProperties = applicationProperties.get("ProducerConfigProperties")

  KinesisStreamsSink.builder[String]
```

```
    .setKinesisClientProperties(outputProperties)
    .setSerializationSchema(new SimpleStringSchema)
    .setStreamName(outputProperties.getProperty(streamNameKey,
 defaultOutputStreamName))
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))
    .build
}
```

- 이 애플리케이션은 윈도우 연산자를 사용하여 5초 동안의 텀블링 윈도우에서 각 주식 기호에 대한 값의 개수를 찾습니다. 다음 코드는 연산자를 생성하고 집계된 데이터를 새로운 Kinesis Data Streams 싱크로 전송합니다.

```
environment.addSource(createSource)
  .map { value =>
    val jsonNode = jsonParser.readValue(value, classOf[JsonNode])
    new Tuple2[String, Int](jsonNode.get("ticker").toString, 1)
  }
  .returns(Types.TUPLE(Types.STRING, Types.INT))
  .keyBy(v => v.f0) // Logically partition the stream for each ticker
  .window(TumblingProcessingTimeWindows.of(Time.seconds(10)))
  .sum(1) // Sum the number of tickers per partition
  .map { value => value.f0 + "," + value.f1.toString + "\n" }
  .sinkTo(createSink)
```

- 응용 프로그램은 StreamExecutionEnvironment 객체를 사용하여 외부 리소스에 액세스할 수 있는 소스 및 싱크 커넥터를 만듭니다.
- 애플리케이션은 동적 애플리케이션 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 런타임 애플리케이션의 속성을 읽어 커넥터를 구성합니다. 런타임 속성에 대한 자세한 내용은 런타임 속성을 참조하세요.

애플리케이션 코드 컴파일 및 업로드

이 섹션에서는 애플리케이션 코드를 컴파일하여 Amazon S3 버킷에 업로드합니다.

애플리케이션 코드 컴파일

SBT 빌드 도구를 사용하여 애플리케이션용 Scala 코드를 빌드합니다. SBT를 설치하려면 cs 설정으로 sbt 설치를 참조하십시오. 또한 Java Devopment Kit(JDK)를 설치해야 합니다. 연습 완료를 위한 사전 조건 참조

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. SBT를 사용하여 코드를 컴파일하고 패키징할 수 있습니다.

```
sbt assembly
```

2. 애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/scala-3.2.0/tumbling-window-scala-1.0.jar
```

Apache Flink 스트리밍 Scala 코드 업로드

이 섹션에서는 Amazon S3 버킷을 만들고 애플리케이션 코드를 업로드합니다.

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 ka-app-code-<username>을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 생성을 선택합니다.
7. ka-app-code-<username> 버킷을 선택한 다음 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 tumbling-window-scala-1.0.jar 파일로 이동합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.

3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **MyApplication**을 입력합니다.
   - 설명에 **My Scala test app**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두십시오.
4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

애플리케이션을 구성하려면 다음 절차를 사용합니다.

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **tumbling-window-scala-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 추가를 선택합니다.

5. 다음을 입력합니다:

그룹 ID	키	값
ConsumerConfigProperties	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

저장을 선택합니다.

6. 속성에서 그룹 추가를 다시 선택합니다.
7. 다음을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**output.stream.name**	**ExampleOutputStream**
ProducerConfigProperties	**aws.region**	**us-west-2**

8. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
9. CloudWatch 로깅하려면 활성화 확인란을 선택합니다.
10. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`

- 로그 스트림: `kinesis-analytics-log-stream`

IAM 정책 편집

IAM 정책을 편집하여 Amazon S3 버킷에 액세스할 수 있는 권한을 추가합니다.

IAM 정책을 편집하여 S3 버킷 권한을 추가하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/tumbling-window-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
```

```
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 [Stop] 을 선택합니다. 작업을 확인합니다.

애플리케이션 생성 및 실행(CLI)

이 섹션에서는 AWS Command Line Interface를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. kinesisanalyticsv2 AWS CLI 명령을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 이 애플리케이션과 상호 작용할 수 있습니다.

권한 정책 생성

> **Note**
>
> 애플리케이션에 대한 권한 정책 및 역할을 생성해야 합니다. 이러한 IAM 리소스를 생성하지 않으면 애플리케이션은 해당 데이터 및 로그 스트림에 액세스할 수 없습니다.

먼저 소스 스트림의 읽기 작업에 대한 권한을 부여하는 문과 싱크 스트림의 쓰기 작업에 대한 권한을 부여하는 문 두 개를 사용하여 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 AKReadSourceStreamWriteSinkStream 권한 정책을 생성합니다. **username**을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(**012345678901**)를 사용자의 계정 ID로 바꿉니다. **MF-stream-rw-role** 서비스 실행 역할은 고객별 역할에 맞게 조정되어야 합니다.

```
{
    "ApplicationName": "tumbling_window",
    "ApplicationDescription": "Scala tumbling window application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "tumbling-window-scala-1.0.jar"
                }
```

```
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다.
4. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다.
5. 사용 사례 선택에서 Managed Service for Apache Flink를 선택합니다.
6. 다음: 권한을 선택합니다.
7. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
8. 역할 생성 페이지에서 역할 이름으로 **MF-stream-rw-role**을 입력합니다. Create role(역할 생성)을 선택합니다.

   MF-stream-rw-role이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.
9. 역할에 권한 정책을 연결합니다.

   > **Note**
   >
   > 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계인 권한 정책 생성에서 생성한 정책을 연결합니다.

   a. 요약 페이지에서 권한 탭을 선택합니다.
   b. 정책 연결을 선택합니다.
   c. 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.
   d. AKReadSourceStreamWriteSinkStream 정책을 선택한 후 정책 연결을 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

애플리케이션 생성

다음 JSON 코드를 create_request.json이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(사용자 이름)를 이전 섹션에서 선택한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(012345678901)를 해당 계정 ID로 바꿉니다. ServiceExecutionRole에는 이전 섹션에서 만든 IAM 사용자 역할이 포함되어야 합니다.

```
"ApplicationName": "tumbling_window",
    "ApplicationDescription": "Scala getting started application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "tumbling-window-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
```

```
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

다음 CreateApplication요청과 함께 를 실행하여 애플리케이션을 생성하십시오.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "tumbling_window",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```
{
   "ApplicationName": "tumbling_window"
}
```

2. 위의 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 애플리케이션 로깅 설정을 참조하십시오.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 UpdateApplication 작업을 사용하여 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{"ApplicationName": "tumbling_window",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
```

```
                }
            },
            {
                "PropertyGroupId": "ProducerConfigProperties",
                "PropertyMap" : {
                     "aws.region" : "us-west-2",
                     "stream.name" : "ExampleOutputStream"
                }
            }
         ]
      }
   }
}
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationCLI 작업을 사용합니다.

> **Note**
>
> 파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하십시오.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (<username>)를 종속 리소스 생성 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "tumbling_window",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "tumbling-window-scala-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

AWS 리소스 정리

이 섹션에는 시작하기 텀블링 윈도우 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream

3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: Scala에서 슬라이딩 윈도우 만들기

> **Note**
>
> Flink 버전 1.15부터 Scala는 무료입니다. 이제 애플리케이션은 모든 Scala 버전에서 Java API를 사용할 수 있습니다. Flink는 여전히 내부적으로 몇 가지 주요 구성 요소에서 Scala를 사용

하지만 사용자 코드 클래스 로더에 Scala를 노출하지는 않습니다. 따라서 사용자는 자신의 jar-acrchive에 Scala 종속성을 추가해야 합니다.
Flink 1.15의 Scala 변경 사항에 대한 자세한 내용은 Scala Free in One Fifteen을 참조하세요.

이 연습에서는 Scala 3.2.0과 Flink의 Java API를 사용하는 간단한 스트리밍 애플리케이션을 만들어 보겠습니다. DataStream 애플리케이션은 Kinesis 스트림에서 데이터를 읽고, 슬라이딩 윈도우를 사용하여 데이터를 집계하고, 결과를 Kinesis 스트림에 기록합니다.

> **Note**
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기(Scala) 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- 애플리케이션 코드 다운로드 및 검토
- 애플리케이션 코드 컴파일 및 업로드
- 애플리케이션 생성 및 실행(콘솔)
- 애플리케이션 생성 및 실행(CLI)
- 애플리케이션 코드 업데이트
- AWS 리소스 정리

애플리케이션 코드 다운로드 및 검토

이 예제의 Python 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. `amazon-kinesis-data-analytics-java-examples/scala/SlidingWindow` 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- `build.sbt` 파일에는 Managed Service for Apache Flink 라이브러리를 포함하여 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- `BasicStreamingJob.scala` 파일에는 애플리케이션의 기능을 정의하는 주요 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
 defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

또한 애플리케이션은 Kinesis 싱크를 사용하여 결과 스트림에 기록합니다. 다음 조각은 Kinesis 싱크를 생성합니다.

```
private def createSink: KinesisStreamsSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val outputProperties = applicationProperties.get("ProducerConfigProperties")

  KinesisStreamsSink.builder[String]
    .setKinesisClientProperties(outputProperties)
    .setSerializationSchema(new SimpleStringSchema)
    .setStreamName(outputProperties.getProperty(streamNameKey,
 defaultOutputStreamName))
    .setPartitionKeyGenerator((element: String) => String.valueOf(element.hashCode))
    .build
}
```

- 애플리케이션은 창 연산자를 사용하여 5초씩 미끄러지는 10초 동안 각 주식 종목의 값 수를 찾습니다. 다음 코드는 연산자를 생성하고 집계된 데이터를 새로운 Kinesis Data Streams 싱크로 전송합니다.

```
environment.addSource(createSource)
    .map { value =>
      val jsonNode = jsonParser.readValue(value, classOf[JsonNode])
```

```
      new Tuple2[String, Double](jsonNode.get("ticker").toString,
jsonNode.get("price").asDouble)
    }
    .returns(Types.TUPLE(Types.STRING, Types.DOUBLE))
    .keyBy(v => v.f0) // Logically partition the stream for each word
    .window(SlidingProcessingTimeWindows.of(Time.seconds(10), Time.seconds(5)))
    .min(1) // Calculate minimum price per ticker over the window
    .map { value => value.f0 + String.format(",%.2f", value.f1) + "\n" }
    .sinkTo(createSink)
```

- 응용 프로그램은 StreamExecutionEnvironment 객체를 사용하여 외부 리소스에 액세스할 수 있는 소스 및 싱크 커넥터를 만듭니다.
- 애플리케이션은 동적 애플리케이션 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 런타임 애플리케이션의 속성을 읽어 커넥터를 구성합니다. 런타임 속성에 대한 자세한 내용은 런타임 속성을 참조하세요.

애플리케이션 코드 컴파일 및 업로드

이 섹션에서는 애플리케이션 코드를 컴파일하여 Amazon S3 버킷에 업로드합니다.

애플리케이션 코드 컴파일

SBT 빌드 도구를 사용하여 애플리케이션용 Scala 코드를 빌드합니다. SBT를 설치하려면 cs 설정으로 sbt 설치를 참조하십시오. 또한 Java Devopment Kit(JDK)를 설치해야 합니다. 연습 완료를 위한 사전 조건 참조

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. SBT를 사용하여 코드를 컴파일하고 패키징할 수 있습니다.

   ```
   sbt assembly
   ```

2. 애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

   ```
   target/scala-3.2.0/sliding-window-scala-1.0.jar
   ```

Apache Flink 스트리밍 Scala 코드 업로드

이 섹션에서는 Amazon S3 버킷을 만들고 애플리케이션 코드를 업로드합니다.

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.

2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 ka-app-code-<username>을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 생성을 선택합니다.
7. ka-app-code-<username> 버킷을 선택한 다음 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 sliding-window-scala-1.0.jar 파일로 이동합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **MyApplication**을 입력합니다.
   - 설명에 **My Scala test app**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두십시오.
4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

애플리케이션을 구성하려면 다음 절차를 사용합니다.

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **sliding-window-scala-1.0.jar.**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 추가를 선택합니다.
5. 다음을 입력합니다:

그룹 ID	키	값
ConsumerConfigProperties	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

저장을 선택합니다.

6. 속성에서 그룹 추가를 다시 선택합니다.
7. 다음을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**output.stream.name**	**ExampleOutputStream**
ProducerConfigProperties	**aws.region**	**us-west-2**

8. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
9. CloudWatch 로깅하려면 활성화 확인란을 선택합니다.
10. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

IAM 정책 편집

IAM 정책을 편집하여 Amazon S3 버킷에 액세스할 수 있는 권한을 추가합니다.

IAM 정책을 편집하여 S3 버킷 권한을 추가하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.

4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/sliding-window-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
```

```
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 [Stop] 을 선택합니다. 작업을 확인합니다.

애플리케이션 생성 및 실행(CLI)

이 섹션에서는 AWS Command Line Interface를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. kinesisanalyticsv2 AWS CLI 명령을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 이 애플리케이션과 상호 작용할 수 있습니다.

권한 정책 생성

> **Note**
>
> 애플리케이션에 대한 권한 정책 및 역할을 생성해야 합니다. 이러한 IAM 리소스를 생성하지 않으면 애플리케이션은 해당 데이터 및 로그 스트림에 액세스할 수 없습니다.

먼저 소스 스트림의 읽기 작업에 대한 권한을 부여하는 문과 싱크 스트림의 쓰기 작업에 대한 권한을 부여하는 문 두 개를 사용하여 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 AKReadSourceStreamWriteSinkStream 권한 정책을 생성합니다. **username**을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(**012345678901**)를 사용자의 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "sliding_window",
    "ApplicationDescription": "Scala sliding window application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "sliding-window-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
```

```
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다.
4. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다.
5. 사용 사례 선택에서 Managed Service for Apache Flink를 선택합니다.

6. 다음: 권한을 선택합니다.
7. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
8. 역할 생성 페이지에서 역할 이름으로 **MF-stream-rw-role**을 입력합니다. Create role(역할 생성)을 선택합니다.

   MF-stream-rw-role이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.
9. 역할에 권한 정책을 연결합니다.

   > **Note**
   >
   > 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계인 권한 정책 생성에서 생성한 정책을 연결합니다.

   a. 요약 페이지에서 권한 탭을 선택합니다.
   b. 정책 연결을 선택합니다.
   c. 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.
   d. AKReadSourceStreamWriteSinkStream 정책을 선택한 후 정책 연결을 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

애플리케이션 생성

다음 JSON 코드를 create_request.json이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(사용자 이름)를 이전 섹션에서 선택한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(012345678901)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "sliding_window",
    "ApplicationDescription": "Scala sliding_window application",
```

```
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "sliding-window-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

다음 CreateApplication요청과 함께 를 실행하여 애플리케이션을 생성하십시오.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "sliding_window",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```
{
   "ApplicationName": "sliding_window"
}
```

2. 위의 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 애플리케이션 로깅 설정을 참조하십시오.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 UpdateApplication 작업을 사용하여 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{"ApplicationName": "sliding_window",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleOutputStream"
               }
            }
         ]
      }
   }
}
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationCLI 작업을 사용합니다.

> **Note**
>
> 파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하십시오.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (<username>)를 종속 리소스 생성 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "sliding_window",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
```

```
        }
    }
}
```

AWS 리소스 정리

이 섹션에는 슬라이딩 윈도우 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.

3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: 스트리밍 데이터를 Scala의 Amazon S3로 전송

> **Note**
>
> Flink 버전 1.15부터 Scala는 무료입니다. 이제 애플리케이션은 모든 Scala 버전에서 Java API를 사용할 수 있습니다. Flink는 여전히 내부적으로 몇 가지 주요 구성 요소에서 Scala를 사용하지만 사용자 코드 클래스 로더에 Scala를 노출하지는 않습니다. 따라서 사용자는 자신의 jar-acrchive에 Scala 종속성을 추가해야 합니다.
> Flink 1.15의 Scala 변경 사항에 대한 자세한 내용은 Scala Free in One Fifteen을 참조하세요.

이 연습에서는 Scala 3.2.0과 Flink의 Java API를 사용하는 간단한 스트리밍 애플리케이션을 만들어 보겠습니다. DataStream 애플리케이션은 Kinesis 스트림에서 데이터를 읽고, 슬라이딩 윈도우를 사용하여 데이터를 집계하고, 결과를 S3에 기록합니다.

> **Note**
>
> 이 연습에 필수 사전 조건을 설정하려면 먼저 시작하기(Scala) 연습을 완료하세요. Amazon S3 ***data/*** ka-app-code 버킷에 추가 폴더를 생성하기만 하면 됩니다. <username>

이 주제는 다음 섹션을 포함하고 있습니다.

- 애플리케이션 코드 다운로드 및 검토
- 애플리케이션 코드 컴파일 및 업로드
- 애플리케이션 생성 및 실행(콘솔)
- 애플리케이션 생성 및 실행(CLI)
- 애플리케이션 코드 업데이트
- AWS 리소스 정리

애플리케이션 코드 다운로드 및 검토

이 예제의 Python 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 Git 설치를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

3. `amazon-kinesis-data-analytics-java-examples/scala/S3Sink` 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- `build.sbt` 파일에는 Managed Service for Apache Flink 라이브러리를 포함하여 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- `BasicStreamingJob.scala` 파일에는 애플리케이션의 기능을 정의하는 주요 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
private def createSource: FlinkKinesisConsumer[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val inputProperties = applicationProperties.get("ConsumerConfigProperties")

  new FlinkKinesisConsumer[String](inputProperties.getProperty(streamNameKey,
 defaultInputStreamName),
    new SimpleStringSchema, inputProperties)
}
```

또한 애플리케이션은 StreamingFileSink a를 사용하여 Amazon S3 버킷에 씁니다.

```
def createSink: StreamingFileSink[String] = {
  val applicationProperties = KinesisAnalyticsRuntime.getApplicationProperties
  val s3SinkPath =
 applicationProperties.get("ProducerConfigProperties").getProperty("s3.sink.path")

  StreamingFileSink
    .forRowFormat(new Path(s3SinkPath), new SimpleStringEncoder[String]("UTF-8"))
    .build()
}
```

- 애플리케이션은 StreamExecutionEnvironment 객체를 사용하여 외부 리소스에 액세스할 수 있는 소스 및 싱크 커넥터를 생성합니다.
- 애플리케이션은 동적 애플리케이션 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 런타임 애플리케이션의 속성을 읽어 커넥터를 구성합니다. 런타임 속성에 대한 자세한 내용은 런타임 속성을 참조하세요.

애플리케이션 코드 컴파일 및 업로드

이 섹션에서는 애플리케이션 코드를 컴파일하여 Amazon S3 버킷에 업로드합니다.

애플리케이션 코드 컴파일

SBT 빌드 도구를 사용하여 애플리케이션용 Scala 코드를 빌드합니다. SBT를 설치하려면 cs 설정으로 sbt 설치를 참조하십시오. 또한 Java Devopment Kit(JDK)를 설치해야 합니다. 연습 완료를 위한 사전 조건 참조

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. SBT를 사용하여 코드를 컴파일하고 패키징할 수 있습니다.

```
sbt assembly
```

2. 애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/scala-3.2.0/s3-sink-scala-1.0.jar
```

Apache Flink 스트리밍 Scala 코드 업로드

이 섹션에서는 Amazon S3 버킷을 만들고 애플리케이션 코드를 업로드합니다.

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 ka-app-code-<username>을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 생성을 선택합니다.
7. ka-app-code-<username> 버킷을 선택한 다음 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 s3-sink-scala-1.0.jar 파일로 이동합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:

- 애플리케이션 명칭에 **MyApplication**을 입력합니다.
- 설명에 **My java test app**를 입력합니다.
- 런타임에서 Apache Flink를 선택합니다.
- 버전은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 두십시오.

4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

애플리케이션을 구성하려면 다음 절차를 사용합니다.

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **s3-sink-scala-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 추가를 선택합니다.
5. 다음을 입력합니다:

그룹 ID	키	값
ConsumerConfigProperties	**input.stream.name**	**ExampleInputStream**
ConsumerConfigProperties	**aws.region**	**us-west-2**
ConsumerConfigProperties	**flink.stream.initpos**	**LATEST**

저장을 선택합니다.

6. 속성에서 그룹 추가를 선택합니다.
7. 다음을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**s3.sink.path**	**s3a://ka-app-code-*<user-name>* /data**

8. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
9. CloudWatch 로깅하려면 활성화 확인란을 선택합니다.
10. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

IAM 정책 편집

IAM 정책을 편집하여 Amazon S3 버킷에 액세스할 수 있는 권한을 추가합니다.

IAM 정책을 편집하여 S3 버킷 권한을 추가하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:Abort*",
                "s3:DeleteObject*",
                "s3:GetObject*",
                "s3:GetBucket*",
                "s3:List*",
                "s3:ListBucket",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-<username>",
                "arn:aws:s3:::ka-app-code-<username>/*"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
```

```
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        }
    ]
}
```

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 [Stop] 을 선택합니다. 작업을 확인합니다.

애플리케이션 생성 및 실행(CLI)

이 섹션에서는 AWS Command Line Interface를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. kinesisanalyticsv2 AWS CLI 명령을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 이 애플리케이션과 상호 작용할 수 있습니다.

권한 정책 생성

> **Note**
>
> 애플리케이션에 대한 권한 정책 및 역할을 생성해야 합니다. 이러한 IAM 리소스를 생성하지 않으면 애플리케이션은 해당 데이터 및 로그 스트림에 액세스할 수 없습니다.

먼저 소스 스트림의 읽기 작업에 대한 권한을 부여하는 문과 싱크 스트림의 쓰기 작업에 대한 권한을 부여하는 문 두 개를 사용하여 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 `AKReadSourceStreamWriteSinkStream` 권한 정책을 생성합니다. **`username`**을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(**`012345678901`**)를 사용자의 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/getting-started-scala-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
```

```
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-analytics/
MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
```

```
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할을 선택한 후 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다.
4. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다.
5. 사용 사례 선택에서 Managed Service for Apache Flink를 선택합니다.
6. 다음: 권한을 선택합니다.
7. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
8. 역할 생성 페이지에서 역할 이름으로 **`MF-stream-rw-role`**을 입력합니다. Create role(역할 생성)을 선택합니다.

   `MF-stream-rw-role`이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.
9. 역할에 권한 정책을 연결합니다.

> **Note**
>
> 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계인 권한 정책 생성에서 생성한 정책을 연결합니다.

a. 요약 페이지에서 권한 탭을 선택합니다.
b. 정책 연결을 선택합니다.
c. 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.
d. AKReadSourceStreamWriteSinkStream 정책을 선택한 후 정책 연결을 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

애플리케이션 생성

다음 JSON 코드를 create_request.json이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(사용자 이름)를 이전 섹션에서 선택한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(012345678901)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "s3_sink",
    "ApplicationDescription": "Scala tumbling window application",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "s3-sink-scala-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
```

```
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "s3.sink.path" : "s3a://ka-app-code-<username>/data"
               }
            }
         ]
      }
    },
    "CloudWatchLoggingOptions": [
      {
         "LogStreamARN": "arn:aws:logs:us-west-2:012345678901:log-
group:MyApplication:log-stream:kinesis-analytics-log-stream"
      }
   ]
}
```

다음 CreateApplication요청과 함께 를 실행하여 애플리케이션을 생성하십시오.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{{
    "ApplicationName": "s3_sink",
```

```
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 `StartApplication` 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```
{
   "ApplicationName": "s3_sink"
}
```

2. 위의 요청과 함께 `StopApplication` 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 애플리케이션 로깅 설정을 참조하십시오.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 UpdateApplication 작업을 사용하여 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{"ApplicationName": "s3_sink",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2",
                    "stream.name" : "ExampleInputStream",
                    "flink.stream.initpos" : "LATEST"
               }
            },
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "s3.sink.path" : "s3a://ka-app-code-<username>/data"
               }
            }
         ]
      }
   }
}
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationCLI 작업을 사용합니다.

> **Note**
>
> 파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하십시오.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 `UpdateApplication`를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 `UpdateApplication` 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. `CurrentApplicationVersionId`를 현재 애플리케이션 버전으로 업데이트하십시오. `ListApplications` 또는 `DescribeApplication` 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (<username>)를 종속 리소스 생성 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "s3_sink",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "s3-sink-scala-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

AWS 리소스 정리

이 섹션에는 시작하기 텀블링 윈도우 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.

2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

Amazon Managed Service for Apache Flink의 보안

AWS에서는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객은 보안에 가장 보안에 민감한 조직의 요건에 부합하도록 구축된 데이터 센터 및 네트워크 아키텍처의 혜택을 누릴 수 있습니다.

보안은 AWS와 사용자의 공동 책임입니다. 공동 책임 모델은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안 - AWS는 AWS 클라우드 내에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 AWS 규정 준수 프로그램의 일환으로 보안 효과를 테스트하고 검증합니다. Managed Service for Apache Flink에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 AWS 규정 준수 프로그램 제공 범위 내 서비스를 참조하세요.
- 클라우드 내 보안 – 사용자의 책임은 사용자가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 데이터의 민감도, 조직의 요건 및 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Managed Service for Apache Flink를 사용할 때 공유 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 다음 주제에서는 보안 및 규정 준수 목적에 맞게 Managed Service for Apache Flink를 구성하는 방법을 보여줍니다. 또한 Managed Service for Apache Flink 리소스를 모니터링하고 보호하는 데 도움이 될 수 있는 다른 Amazon 서비스를 사용하는 방법도 배웁니다.

주제

Amazon Managed Service for Apache Flink의 데이터 보호

AWS에서 제공하는 도구를 사용하여 데이터를 보호할 수 있습니다. Managed Service for Apache Flink는 Kinesis Data Firehose 및 Amazon S3를 포함하여 데이터 암호화를 지원하는 서비스와 함께 사용할 수 있습니다.

Managed Service for Apache Flink의 데이터 암호화

유휴 상태에서의 암호화

다음은 Managed Service for Apache Flink를 사용한 저장 데이터 암호화에 대한 유의 사항입니다.

- 를 사용하여 수신 Kinesis 데이터 스트림의 데이터를 암호화할 수 있습니다. StartStreamEncryption 자세한 정보는 Kinesis Data Streams용 서버 측 암호화란 무엇입니까?를 참조하세요.
- 출력 데이터는 Kinesis Data Firehose를 사용하여 유휴 시 암호화하여 암호화된 Amazon S3 버킷에 데이터를 저장할 수 있습니다. Amazon S3 버킷이 사용하는 암호화 키를 사용자가 지정할 수 있습니다. 자세한 정보는 KMS 관리형 키(SSE-KMS)를 사용하는 서버 측 암호화로 데이터 보호를 참조하세요.
- Managed Service for Apache Flink는 모든 스트리밍 소스에서 읽고 모든 스트리밍 또는 데이터베이스 대상에 쓸 수 있습니다. 소스와 대상이 전송 중인 모든 데이터와 저장된 데이터를 암호화하는지 확인하세요.
- 애플리케이션의 코드는 암호화된 상태로 저장됩니다.
- 내구성이 뛰어난 애플리케이션 스토리지는 유휴 시 암호화됩니다.
- 실행 중인 애플리케이션 스토리지는 유휴 시 암호화됩니다.

전송 중 데이터 암호화

Managed Service for Apache Flink는 전송 중인 모든 데이터를 암호화합니다. 전송 중 데이터 암호화는 모든 Managed Service for Apache Flink 애플리케이션에서 활성화되어 있으며 비활성화할 수 없습니다.

Managed Service for Apache Flink는 다음 시나리오에서 전송 중인 데이터를 암호화합니다:

- Kinesis Data Streams에서 Managed Service for Apache Flink로 전송 중인 데이터.
- Managed Service for Apache Flink내의 내부 구성 요소 간에 전송 중인 데이터.
- Managed Service for Apache Flink와 Kinesis Data Firehose 간에 전송 중인 데이터.

키 관리

Managed Service for Apache Flink의 데이터 암호화는 서비스 관리 키를 사용합니다. 고객 관리형 키는 지원되지 않습니다.

Amazon Managed Service for Apache Flink의 자격 증명 및 액세스 관리

AWS Identity and Access Management(IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 누가 인증(로그인) 및 권한(권한 있음)을 받아 Managed Service for Apache Flink 리소스를 사용할 수 있는지를 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

주제

고객

AWS Identity and Access Management(IAM)를 사용하는 방법은 Managed Service for Apache Flink에서 수행하는 작업에 따라 달라집니다.

서비스 사용자 – Managed Service for Apache Flink 서비스를 사용하여 작업을 수행하는 경우 필요한 보안 인증과 권한을 관리자가 제공합니다. 더 많은 Managed Service for Apache Flink 기능을 사용하여 작업을 수행하게 되면 추가 권한이 필요할 수 있습니다. 액세스 권한 관리 방식을 이해하면 적절한 권한을 관리자에게 요청할 수 있습니다. Managed Service for Apache Flink의 기능에 액세스할 수 없는 경우 Amazon Managed Service for Apache Flink의 자격 증명 및 액세스 문제 해결 섹션을 참조하세요.

서비스 관리자 - 회사에서 Managed Service for Apache Flink 리소스를 책임지고 있는 경우 Managed Service for Apache Flink의 전체 액세스 권한을 가지고 있을 것입니다. 서비스 관리자는 서비스 사용자가 액세스해야 하는 Managed Service for Apache Flink 기능과 리소스를 결정합니다. 그런 다음, IAM 관리자에게 요청을 제출하여 서비스 사용자의 권한을 변경해야 합니다. 이 페이지의 정보를 검토하여

IAM의 기본 개념을 이해하십시오. 회사가 Managed Service for Apache Flink에서 IAM을 사용하는 방법에 대해 자세히 알아보려면 Amazon Managed Service for Apache Flink가 IAM과 연동되는 방식 섹션을 참조하세요.

IAM 관리자 - IAM 관리자라면 Managed Service for Apache Flink의 액세스 권한 관리 정책 작성 방법을 자세히 알고 싶을 것입니다. IAM에서 사용할 수 있는 Managed Service for Apache Flink 자격 증명 기반 정책 예제를 보려면 Amazon Managed Service for Apache Flink의 자격 증명 기반 정책 예제 섹션을 참조하세요.

보안 인증 정보를 통한 인증

인증은 ID 보안 인증 정보를 사용하여 AWS에 로그인하는 방식입니다. AWS 계정 루트 사용자이나 IAM 사용자로 또는 IAM 역할을 수임하여 인증(AWS에 로그인)되어야 합니다.

보안 인증 정보 소스를 통해 제공된 보안 인증 정보를 사용하여 페더레이션형 ID로 AWS에 로그인할 수 있습니다. AWS IAM Identity Center (IAM Identity Center) 사용자, 회사의 Single Sign-On 인증, Google 또는 Facebook 보안 인증 정보가 페더레이션형 ID의 예입니다. 페더레이션형 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 AWS에 액세스하면 간접적으로 역할을 수임합니다.

사용자 유형에 따라 AWS Management Console 또는AWS 액세스 포털에 로그인할 수 있습니다. AWS에 로그인하는 방법에 대한 자세한 내용은 AWS 로그인사용 설명서의 AWS 계정에 로그인하는 방법을 참조하세요.

AWS에 프로그래밍 방식으로 액세스하는 경우, AWS에서는 보안 인증 정보를 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트(SDK) 및 명령줄 인터페이스(CLI)를 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 요청에 직접 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS API 요청에 서명을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS는 다중 인증(MFA)을 사용하여 계정의 보안을 강화하는 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center사용 설명서의 다중 인증 및 IAM 사용 설명서의 AWS에서 다중 인증(MFA) 사용을 참조하세요.

AWS 계정 루트 사용자

AWS 계정을 생성할 때는 해당 계정의 모든 AWS 서비스및 리소스에 대한 완전한 액세스 권한이 있는 단일 로그인 ID로 시작합니다. 이 자격 증명은 AWS 계정루트 사용자라고 하며, 계정을 생성할 때 사용한 이메일 주소와 암호로 로그인하여 액세스합니다. 일상적인 작업에는 루트 사용자를 가급적 사용하지 않는 것이 좋습니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있는 작업을

수행하는 데 사용합니다. 루트 사용자로 로그인해야 하는 태스크의 전체 목록은 IAM 사용자 안내서의 루트 사용자 보안 인증이 필요한 태스크를 참조하세요.

페더레이션 ID

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 포함한 사용자가 자격 증명 공급자와의 페더레이션을 통해 임시 보안 인증을 사용하여 AWS 서비스에 액세스하도록 요구하는 것입니다.

연동 자격 증명은 엔터프라이즈 사용자 디렉터리, 웹 자격 증명 공급자, AWS Directory Service, Identity Center 디렉터리의 사용자 또는 자격 증명 소스를 통해 제공된 자격 증명을 사용하여 AWS 서비스에 액세스하는 모든 사용자입니다. 페더레이션 보안 인증 정보는 AWS 계정에 액세스할 때 역할을 수임하고 역할은 임시 보안 인증 정보를 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center을 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 모든 AWS 계정및 애플리케이션에서 사용하기 위해 고유한 자격 증명 소스의 사용자 및 그룹 집합에 연결하고 동기화할 수 있습니다. IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center사용 설명서에서 IAM Identity Center란 무엇입니까?를 참조하세요.

IAM 사용자 및 그룹

IAM 사용자는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가지고 있는 AWS 계정내 자격 증명입니다. 가능하면 암호 및 액세스 키와 같은 장기 보안 인증이 있는 IAM 사용자를 생성하는 대신 임시 보안 인증을 사용하는 것이 좋습니다. 하지만 IAM 사용자의 장기 자격 증명이 필요한 특정 사용 사례가 있는 경우 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체를 참조하십시오.

IAM 그룹은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 귀하는 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, IAMAdmins(이)라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수임할 수 있습니다. 사용자는 영구적인 보안 인증을 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 정보는 IAM 사용 설명서의 IAM 사용자를 만들어야 하는 경우(역할이 아님)를 참조하십시오.

IAM 역할

IAM 역할은 특정 권한을 가지고 있는 AWS 계정계정 내 ID입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할 전환하여 AWS Management Console에서 IAM 역할을 임시로 수임할

수 있습니다. AWS CLI 또는 AWSAPI 태스크를 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 정보는 IAM 사용 설명서의 IAM 역할 사용을 참조하십시오.

임시 보안 인증 정보가 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 자격 증명에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 자격 증명이 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 Creating a role for a third-party Identity Provider(서드 파티 자격 증명 공급자의 역할 만들기) 부분을 참조하세요. IAM Identity Center를 사용하는 경우 권한 세트를 구성합니다. 인증 후 보안 인증 정보에서 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 집합을 IAM의 역할과 연결합니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center사용 설명서의 권한 세트를 참조하세요.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 작업에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스: IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스를 사용하면 역할을(프록시로 사용하는 대신) 리소스에 정책을 직접 연결할 수 있습니다. 크로스 계정 액세스를 위한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하십시오.
- 교차 서비스 액세스 - 일부 AWS 서비스는 다른 AWS 서비스의 기능을 사용합니다. 예를 들어 서비스에서 직접적으로 호출하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 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 AWS에서 작업을 수행하는 사람은 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 보안 주체의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하십시오.
  - 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 AWS 서비스에 대한 권한을 위임할 역할 생성을 참조하십시오.
  - 서비스 연결 역할 - 서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 유형입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은

AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수는 없습니다.

- Amazon EC2에서 실행 중인 애플리케이션 – IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 AWS CLI또는 AWSAPI 요청을 수행하는 애플리케이션의 임시 보안 인증 정보를 관리할 수 있습니다. 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS역할을 할당하고 해당 역할을 모든 애플리케이션에서 사용할 수 있도록 하려면 인스턴스에 연결된 인스턴스 프로파일을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여를 참조하십시오.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 IAM 사용 설명서의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하십시오.

정책을 사용한 액세스 관리

정책을 생성하고 AWSID 또는 리소스에 연결하여 AWS내 액세스를 제어합니다. 정책은 자격 증명 또는 리소스와 연결될 때 해당 권한을 정의하는 AWS의 객체입니다. AWS는 보안 주체(사용자, 루트 사용자 또는 역할 세션)가 요청을 보낼 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되는지 또는 거부되는지를 결정합니다. 대부분의 정책은 AWS에 JSON 설명서로서 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 정보는 IAM 사용 설명서의 JSON 정책 개요를 참조하십시오.

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, `iam:GetRole`태스크를 허용하는 정책이 있다고 가정합니다. 해당 정책이 있는 사용자는 AWS Management Console, AWS CLI또는 AWSAPI에서 역할 정보를 가져올 수 있습니다.

ID 기반 정책

ID 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 IAM 정책 생성을 참조하십시오.

자격 증명 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 AWS 계정에 속한 다수의 사용자, 그룹 및 역할에 독립적으로 추가할 수 있는 정책입니다. 관리형 정책에는 AWS관리형 정책과 고객 관리형 정책이 포함되어 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 관리형 정책과 인라인 정책의 선택을 참조하세요.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS관리형 정책을 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Amazon S3, AWS WAF및 Amazon VPC는 ACL을 지원하는 대표적인 서비스입니다. ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 안내서의 액세스 제어 목록(ACL) 개요를 참조하십시오.

기타 정책 유형

AWS은(는) 비교적 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 유형은 더 일반적인 정책 유형에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 ID 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 엔터티의 ID 기반 정책 및 해당 권한 경계의 교집합입니다. `Principal`필드에서 사용자나 역할을 보안 주체로 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 권한 경계에 대한 자세한 정보는 IAM 사용 설명서의 IAM 엔터티에 대한 권한 경계를 참조하십시오.

- 서비스 제어 정책(SCP) – SCP는 AWS Organizations에서 조직 또는 조직 단위(OU)에 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations은 기업이 소유하는 여러 개의 AWS 계정를 그룹화하고 중앙에서 관리하기 위한 서비스입니다. 조직에서 모든 기능을 활성화할 경우 서비스 제어 정책(SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 각 AWS 계정 루트 사용자을 비롯하여 멤버 계정의 엔터티에 대한 권한을 제한합니다. 조직 및 SCP에 대한 자세한 정보는 AWS Organizations사용 설명서의 SCP 작동 방식을 참조하세요.
- 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 ID 기반 정책 및 세션 정책의 교집합입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 정보는 IAM 사용 설명서의 세션 정책을 참조하십시오.

여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지를 결정하는 방법을 알아보려면 IAM 사용 설명서의 정책 평가 로직을 참조하십시오.

Amazon Managed Service for Apache Flink가 IAM과 연동되는 방식

IAM을 사용하여 Managed Service for Apache Flink의 액세스를 관리하기 전에 Managed Service for Apache Flink와 함께 사용할 수 있는 IAM 기능을 알아보세요.

Amazon Managed Service for Apache Flink에서 사용할 수 있는 IAM 기능

IAM 특성	Managed Service for Apache Flink 지원
ID 기반 정책	예
리소스 기반 정책	아니요
정책 작업	예
정책 리소스	예
정책 조건 키	아니오
ACL	아니요

IAM 특성	Managed Service for Apache Flink 지원
ABAC(정책의 태그)	예
임시 보안 인증	예
보안 주체 권한	예
서비스 역할	아니요
서비스 연결 역할	아니오

Managed Service for Apache Flink 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

Managed Service for Apache Flink를 위한 자격 증명 기반 정책

ID 기반 정책 지원	예

ID 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 IAM 정책 생성을 참조하세요.

IAM 자격 증명 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. ID 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 참조 섹션을 참조하십시오.

Managed Service for Apache Flink를 위한 자격 증명 기반 정책 예제

Managed Service for Apache Flink 자격 증명 기반 정책 예제를 보려면 Amazon Managed Service for Apache Flink의 자격 증명 기반 정책 예제 섹션을 참조하세요.

Managed Service for Apache Flink를 위한 리소스 기반 정책

리소스 기반 정책 지원	예

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 보안 주체를 지정해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스이(가) 포함될 수 있습니다.

교차 계정 액세스를 활성화하려는 경우 전체 계정이나 다른 계정의 IAM 엔터티를 리소스 기반 정책의 보안 주체로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념합니다. 보안 주체와 리소스가 서로 다른 AWS 계정에 있는 경우 신뢰할 수 있는 계정의 IAM 관리자는 보안 주체 개체(사용자 또는 역할)에도 리소스 액세스 권한을 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 보안 주체에 액세스를 부여하는 경우 추가 ID 기반 정책이 필요하지 않습니다. 자세한 정보는 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이 섹션을 참조하십시오.

Managed Service for Apache Flink의 정책 작업

정책 작업 지원	예

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 `Action`요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 태스크를 설명합니다. 일반적으로 정책 작업의 이름은 연결된 AWSAPI 작업의 이름과 동일합니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 종속 작업이라고 합니다.

연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함합니다.

Managed Service for Apache Flink 작업 목록을 보려면 서비스 승인 참조의 Amazon Managed Service for Apache Flink에서 정의한 작업을 참조하세요.

Managed Service for Apache Flink의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
Kinesis Analytics
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
      "Kinesis Analytics:action1",
      "Kinesis Analytics:action2"
         ]
```

와일드카드(*)를 사용하여 여러 작업을 지정할 수 있습니다. 예를 들어, Describe라는 단어로 시작하는 모든 작업을 지정하려면 다음 작업을 포함합니다.

```
"Action": "Kinesis Analytics:Describe*"
```

Managed Service for Apache Flink 자격 증명 기반 정책 예제를 보려면 Amazon Managed Service for Apache Flink의 자격 증명 기반 정책 예제 섹션을 참조하세요.

Managed Service for Apache Flink의 정책 리소스

정책 리소스 지원	예

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 개체를 지정합니다. 문에는 Resource또는 NotResource요소가 반드시 추가되어야 합니다. 모범 사례에 따라 Amazon 리소스 이름(ARN)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이 태스크를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"
```

Managed Service for Apache Flink 리소스 유형 및 해당 ARN 목록을 보려면 서비스 승인 참조의 Amazon Managed Service for Apache Flink에서 정의한 리소스를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 Amazon Managed Service for Apache Flink에서 정의한 작업을 참조하세요.

Managed Service for Apache Flink 자격 증명 기반 정책 예제를 보려면 Amazon Managed Service for Apache Flink의 자격 증명 기반 정책 예제 섹션을 참조하세요.

Managed Service for Apache Flink의 정책 조건 키

서비스별 정책 조건 키 지원	예

관리자는 AWSJSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지 지정할 수 있습니다.

`Condition` 요소(또는 `Condition`블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. `Condition` 요소는 옵션입니다. 같거나 작음과 같은 조건 연산자를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다.

한 문에서 여러 `Condition`요소를 지정하거나 단일 `Condition`요소에서 여러 키를 지정하는 경우 AWS는 논리적 `AND`태스크를 사용하여 평가합니다. 단일 조건 키의 여러 값을 지정하는 경우 AWS는 논리적 `OR`태스크를 사용하여 조건을 평가합니다. 명령문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리표시자 변수를 사용할 수도 있습니다. 예를 들어, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM 정책 요소: 변수 및 태그를 참조하세요.

AWS는 전역 조건 키와 서비스별 조건 키를 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 설명서의 AWS글로벌 조건 컨텍스트 키를 참조하세요.

Managed Service for Apache Flink 조건 키 목록을 보려면 서비스 승인 참조의 Amazon Managed Service for Apache Flink에 사용되는 조건 키를 참조하세요. 조건 키를 사용할 수 있는 작업과 리소스를 알아보려면 Amazon Managed Service for Apache Flink에서 정의한 작업을 참조하세요.

Managed Service for Apache Flink 자격 증명 기반 정책 예제를 보려면 Amazon Managed Service for Apache Flink의 자격 증명 기반 정책 예제 섹션을 참조하세요.

Managed Service for Apache Flink의 액세스 제어 목록 (ACL)

ACL 지원	아니요

액세스 제어 목록(ACLs)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는 지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

Managed Service for Apache Flink의 ABAC(속성 기반 액세스 제어)

ABAC 지원(정책의 태그)	예

속성 기반 액세스 제어(ABAC)는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. AWS에서는 이러한 속성을 태그라고 합니다. IAM 엔터티(사용자 또는 역할) 및 많은 AWS리소스에 태그를 연결할 수 있습니다. ABAC의 첫 번째 단계로 개체 및 리소스에 태그를 지정합니다. 그런 다음 보안 주체의 태그가 액세스하려는 리소스의 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계합니다.

ABAC는 빠르게 성장하는 환경에서 유용하며 정책 관리가 번거로운 상황에 도움이 됩니다.

태그를 기반으로 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 부분입니다.

ABAC에 대한 자세한 정보는 IAM 사용 설명서의 ABAC란 무엇인가요?를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 ABAC(속성 기반 액세스 제어) 사용을 참조하세요.

Managed Service for Apache Flink에서 임시 보안 인증 사용하기

임시 보안 인증 지원	예

일부 AWS 서비스는 임시 보안 인증 정보를 사용하여 로그인할 때 작동하지 않습니다. 임시 자격 증명으로 작동하는 AWS 서비스를 비롯한 추가 정보는 IAM 사용 설명서의 IAM으로 작업하는 AWS 서비스를 참조하세요.

사용자 이름과 암호를 제외한 다른 방법을 사용하여 AWS Management Console에 로그인하면 임시 보안 인증 정보를 사용하는 것입니다. 예를 들어 회사의 Single Sign-On(SSO) 링크를 사용하여 AWS에 액세스하면 해당 프로세스에서 자동으로 임시 보안 인증 정보를 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 보안 인증 정보를 자동으로 생성합니다. 역할 전환에 대한 자세한 정보는 IAM 사용 설명서의 역할로 전환(콘솔)을 참조하세요.

AWS CLI 또는 AWSAPI를 사용하여 임시 보안 인증 정보를 수동으로 만들 수 있습니다 그런 다음 이러한 임시 보안 인증 정보를 사용하여 AWS에 액세스할 수 있습니다. AWS에서는 장기 액세스 키를 사용하는 대신 임시 보안 인증 정보를 동적으로 생성할 것을 권장합니다. 자세한 정보는 IAM의 임시 보안 인증 섹션을 참조하세요.

Managed Service for Apache Flink의 서비스 간 보안 주체 권한

전달 액세스 세션(FAS) 지원	예

IAM 사용자 또는 역할을 사용하여 AWS에서 작업을 수행하는 사람은 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 AWS 서비스를 직접 호출하는 보안 주체의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청은 서비스에서 완료를 위해 다른 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 받은 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 전달 액세스 세션을 참조하십시오.

Managed Service for Apache Flink의 서비스 역할

서비스 역할 지원	예

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 IAM 역할입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 역할을 생성하여 AWS 서비스에 권한 위임을 참조하세요.

> **Warning**
>
> 서비스 역할에 대한 권한을 변경하면 Managed Service for Apache Flink 기능이 중단될 수 있습니다. Managed Service for Apache Flink에서 관련 지침을 제공하는 경우에만 서비스 역할을 편집하세요.

Managed Service for Apache Flink의 서비스 연결 역할

서비스 연결 역할 지원	예

서비스 연결 역할은 AWS 서비스에 연결된 서비스 역할의 한 타입입니다. 서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은 AWS 계정에 나타나고, 서비스가 소유합니다. IAM 관리자는 서비스 연결 역할의 권한을 볼 수 있지만 편집할 수는 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 IAM으로 작업하는 AWS서비스 단원을 참조하세요. 서비스 연결 역할 열에서 Yes이(가) 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 Yes(네) 링크를 선택합니다.

Amazon Managed Service for Apache Flink의 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할은 Managed Service for Apache Flink 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 또한 AWS Management Console, AWS Command Line Interface(AWS CLI) 또는 AWS API를 사용해 작업을 수행할 수 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수임할 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 IAM 정책 생성을 참조하십시오.

각 리소스 유형에 대한 ARN 형식을 포함하여 Managed Service for Apache Flink에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 승인 참조의 Amazon Managed Service for Apache Flink에 사용되는 작업, 리소스 및 조건 키를 참조하세요.

주제

- 정책 모범 사례
- Managed Service for Apache Flink의 콘솔 사용
- 사용자가 자신의 고유한 권한을 볼 수 있도록 허용

정책 모범 사례

ID 기반 정책에 따라 계정에서 사용자가 Managed Service for Apache Flink 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS관리형 정책을 사용합니다. 관리형 정책은 AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS고객 관리형 정책을 정의하여 권한

을 줄이는 것이 좋습니다. 자세한 정보는 IAM 사용 설명서의 AWSmanaged policies(관리형 정책) 또는 AWSmanaged policies for job functions(직무에 대한 관리형 정책)를 참조하세요.

- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 Policies and permissions in IAM(IAM의 정책 및 권한)을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한: 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어 SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. 특정 AWS 서비스(예: AWS CloudFormation)을(를) 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 정보는 IAM 사용 설명서의 IAM JSON 정책 요소: 조건을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 검증하여 안전하고 기능적인 권한 보장 – IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 IAM 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 검증합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 권장 사항을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 정보는 IAM 사용 설명서의 IAM Access Analyzer 정책 검증을 참조하십시오.
- 다중 인증(MFA) 필요 – AWS 계정계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 MFA를 설정합니다. API 작업을 직접적으로 호출할 때 MFA가 필요하면 정책에 MFA 조건을 추가합니다. 자세한 정보는 IAM 사용 설명서의 Configuring MFA-protected API access(MFA 보호 API 액세스 구성)를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 IAM의 보안 모범 사례 섹션을 참조하십시오.

Managed Service for Apache Flink의 콘솔 사용

Amazon Managed Service for Apache Flink 콘솔에 액세스하려면 최소한의 권한 집합이 있어야 합니다. 이러한 권한은 AWS 계정에서 Managed Service for Apache Flink 리소스에 대한 세부 정보를 나열하고 볼 수 있도록 허용해야 합니다. 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 AWSAPI만 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요가 없습니다. 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 계속 Managed Service for Apache Flink 콘솔을 사용할 수 있도록 하려면, Managed Service for Apache Flink `ConsoleAccess`또는 `ReadOnly`AWS 관리형 정책을 엔터티에 추가합니다. 자세한 정보는 IAM 사용 설명서의 사용자에게 권한 추가를 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예시는 IAM 사용자가 자신의 사용자 자격 증명에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 AWS CLI나 AWS API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Amazon Managed Service for Apache Flink의 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 Managed Service for Apache Flink 및 IAM에서 발생할 수 있는 공통적인 문제를 진단하고 수정할 수 있습니다.

주제

Managed Service for Apache Flink에서 작업을 수행할 권한이 없음

AWS Management Console에서 작업을 수행할 권한이 없다는 메시지가 나타나는 경우 관리자에게 문의하여 도움을 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 `Kinesis Analytics:`*`GetWidget`* 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: Kinesis
 Analytics:GetWidget on resource: my-example-widget
```

이 경우 Mateo는 *`my-example-widget`* 작업을 사용하여 `Kinesis Analytics:`*`GetWidget`* 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

IAM을 수행할 권한이 없습니다. PassRole

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Managed Service for Apache Flink에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 `marymajor`라는 IAM 사용자가 콘솔을 사용하여 Managed Service for Apache Flink에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

이 경우 Mary가 `iam:PassRole`작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS관리자에게 문의하세요. 관리자는 로그인 보안 인증을 제공한 사용자입니다.

내 AWS 계정 외부의 사람이 내 Managed Service for Apache Flink 리소스에 액세스하도록 허용하려고 함

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- Managed Service for Apache Flink가 이러한 기능을 지원하는지 여부를 알아보려면 Amazon Managed Service for Apache Flink가 IAM과 연동되는 방식 섹션을 참조하세요.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 자신이 소유한 다른 AWS 계정의 IAM 사용자에 대한 액세스 권한 제공을 참조하십시오.
- 리소스에 대한 액세스 권한을 서드 파티 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 서드 파티가 소유한 AWS 계정에 대한 액세스 제공을 참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 외부에서 인증된 사용자에게 액세스 권한 제공(ID 페더레이션)을 참조하십시오.
- 크로스 계정 액세스를 위한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 IAM 역할과 리소스 기반 정책의 차이를 참조하세요.

교차 서비스 혼동된 대리자 예방

AWS에서 교차 서비스 가장은 한 서비스(호출하는 서비스)가 다른 서비스(호출받는 서비스)를 호출할 때 발생할 수 있습니다. 호출하는 서비스는 적절한 권한이 없는 경우에도 다른 고객의 리소스에 대해 작동하게 권한을 사용하도록 조작될 수 있으며, 이로 인해 혼동된 대리자 문제가 발생할 수 있습니다.

혼동된 대리자를 방지하기 위해 AWS에서는 계정의 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하여 모든 서비스에 대한 데이터를 보호하는 데 도움이 되는 도구를 제공합니다. 이 섹션에서는 Apache Flink용 Managed Service에 특정한 서비스 간 혼동된 대리자 방지에 중점을 두지만, 이 주제에 대한 자세한 내용은 IAM 사용 설명서의 혼동된 대리자 문제 섹션에서 확인할 수 있습니다.

Apache Flink용 관리형 서비스의 경우 역할 신뢰 정책에 aws: SourceArn 및 aws: SourceAccount 글로벌 조건 컨텍스트 키를 사용하여 예상 리소스에서 생성된 요청으로만 역할에 대한 액세스를 제한하는 것이 좋습니다.

하나의 리소스만 교차 서비스 액세스와 연결되도록 허용하려는 경우 `aws:SourceArn`을 사용하십시오. 해당 계정의 모든 리소스가 교차 서비스 사용과 연결되도록 허용하려는 경우 `aws:SourceAccount`을 사용하세요.

의 값은 Managed Service for Apache Flink에서 사용하는 리소스의 `aws:SourceArn` ARN이어야 하며, `arn:aws:kinesisanalytics:region:account:resource` 형식으로 지정됩니다.

혼동된 대리자 문제에 대한 권장 접근 방식은 전체 리소스 ARN이 포함된 `aws:SourceArn` 글로벌 조건 컨텍스트 키를 사용하는 것입니다.

리소스의 전체 ARN을 모르거나 여러 리소스를 지정하는 경우, ARN의 알 수 없는 부분에 대해 와일드카드 문자(*)를 포함한 `aws:SourceArn` 키를 사용합니다. 예를 들면 `arn:aws:kinesisanalytics::111122223333:*`입니다.

Managed Service for Apache Flink에 제공하는 역할 정책과 사용자를 위해 생성된 역할의 신뢰 정책은 이러한 키를 사용할 수 있습니다.

혼동된 대리자 문제를 방지하려면 다음 단계를 수행하세요.

혼동된 대리자 문제로부터 보호

1. AWS 관리 콘솔에 로그인하여 https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 역할을 선택한 다음 수정하려는 역할을 선택합니다.
3. 신뢰 정책 편집을 선택합니다.
4. 신뢰 정책 편집 페이지에서 기본 JSON 정책을 및 글로벌 조건 컨텍스트 키 중 하나 또는 `aws:SourceArn` 및 `aws:SourceAccount` 둘 다를 사용하는 정책으로 바꿉니다. 다음 예를 참조하세요.
5. 정책 업데이트를 선택합니다.

```
{
   "Version":"2012-10-17",
   "Statement":[
      {
         "Effect":"Allow",
         "Principal":{
            "Service":"kinesisanalytics.amazonaws.com"
```

```
         },
         "Action":"sts:AssumeRole",
         "Condition":{
            "StringEquals":{
               "aws:SourceAccount":"Account ID"
            },
            "ArnEquals":{
               "aws:SourceArn":"arn:aws:kinesisanalytics:us-
east-1:123456789012:application/my-app"
            }
         }
      }
   ]
}
```

Managed Service for Apache Flink 모니터링

Managed Service for Apache Flink 애플리케이션에 대한 모니터링 기능을 제공합니다. 자세한 설명은 로깅 및 모니터링 섹션을 참조하세요.

Amazon Managed Service for Apache Flink의 규정 준수 확인

타사 감사자는 여러 AWS 규정 준수 프로그램의 일환으로 Amazon Managed Service for Apache Flink의 보안 및 규정 준수를 평가합니다. 여기에는 SOC, PCI, HIPAA 등이 포함됩니다.

특정 규정 준수 프로그램의 범위에 속하는 AWS 서비스 목록은 다음을 참조하세요. 일반 정보는 AWS 규정 준수 프로그램을 참조하세요.

AWS Artifact를 사용하여 제3자 감사 보고서를 다운로드할 수 있습니다. 자세한 정보는 AWS Artifact의 보고서 다운로드를 참조하세요.

Managed Service for Apache Flink 사용 시 규정 준수 책임은 데이터의 민감도, 회사의 규정 준수 목표 및 관련 법률 및 규정에 따라 결정됩니다. Managed Service for Apache Flink 사용 시 HIPAA 또는 PCI와 같은 표준을 준수해야 하는 경우 다음과 같이 AWS에서 제공하는 도움말 리소스를 활용하세요.

- 보안 및 규정 준수 빠른 시작 안내서 – 이 배포 안내서에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 AWS 환경을 배포하기 위한 단계를 제공합니다.
- Architecting for HIPAA Security and Compliance on Amazon Web Services 이 백서는 기업에서 AWS를 사용하여 HIPAA를 준수하는 애플리케이션을 만드는 방법을 설명합니다.

- AWS 규정 준수 리소스 – 사용자의 업계와 위치에 해당할 수 있는 워크북 및 안내서 모음입니다.
- AWS Config – 이 AWS 서비스로 리소스 구성이 내부 관행, 업계 지침 및 규정을 준수하는 정도를 평가할 수 있습니다.
- AWS Security Hub - 이 AWS 서비스는 보안 산업 표준 및 모범 사례 규정 준수 여부를 확인하는 데 도움이 되도록 AWS 내 보안 상태를 종합적으로 보여줍니다.

FedRAMP

AWS FedRAMP 규정 준수 프로그램에는 Managed Service for Apache Flink가 FedRAMP 권한 부여 서비스로 포함되어 있습니다. 연방 또는 기업 고객인 경우 이 서비스를 사용하여 영향력이 높은 수준의 데이터를 포함하는 AWS GovCloud (미국) 지역의 인증 경계에서 민감한 워크로드를 처리하고, 최대 중간 수준의 데이터를 포함하는 미국 동부 (버지니아 북부), 미국 동부 (오하이오), 미국 서부 (오하이오), 미국 서부 (오레곤) 지역에서 민감한 워크로드를 처리하고 저장할 수 있습니다.

AWS FedRAMP 보안 패키지에 대한 액세스 권한은 FedRAMP PMO 또는 사용자의 AWS 영업 계정 관리자를 통해 요청하거나 AWS 아티팩트의 AWS 아티팩트를 통해 다운로드할 수 있습니다.

자세한 정보는 FedRAMP 섹션을 참조하세요.

Amazon Managed Service for Apache Flink의 복원성

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 지역은 물리적으로 분리되고 격리된 다수의 가용 영역을 제공하며 이러한 가용 영역은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 다중 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 AWS 글로벌 인프라를 참조하세요.

AWS 글로벌 인프라뿐만 아니라 Managed Service for Apache Flink도 데이터 복원력과 백업 요구 사항을 지원하는 다양한 기능을 제공합니다.

재해 복구

Managed Service for Apache Flink는 서버리스 모드에서 실행되며, 자동 마이그레이션을 수행하여 호스트 저하, 가용 영역 가용성 및 기타 인프라 관련 문제를 처리합니다. Managed Service for Apache Flink는 여러 중복 메커니즘을 통해 이를 달성합니다. 각 Apache Flink 관리 서비스 애플리케이션은 단일 테넌트 Apache Flink 클러스터에서 실행됩니다. Apache Flink 클러스터는 여러 가용 영역에서

Zookeeper를 사용하여 고가용성 모드로 실행됩니다. JobMananger Managed Service for Apache Flink는 Amazon EKS를 사용하여 Apache Flink를 배포합니다. Amazon EKS에서는 가용 영역 전반의 각 AWS 리전에 대해 여러 개의 Kubernetes 포드가 사용됩니다. 장애가 발생할 경우 Managed Service for Apache Flink는 먼저 애플리케이션의 체크포인트(사용 가능한 경우)를 사용하여 실행 중인 Apache Flink 클러스터 내에서 애플리케이션을 복구하려고 시도합니다.

Managed Service for Apache Flink는 체크포인트와 스냅샷을 사용하여 애플리케이션 상태를 백업합니다.

- 체크포인트는 Apache Flink용 Managed Service에서 주기적으로 자동 생성하여 장애를 복구하는 데 사용하는 애플리케이션 상태의 백업입니다.
- 스냅샷은 수동으로 만들고 복원한 애플리케이션 상태의 백업입니다.

검사에 대한 자세한 내용은 내결함성 섹션을 참조하세요.

버전 관리

저장된 버전의 애플리케이션 상태는 다음과 같이 버전이 관리됩니다.

- 체크포인트는 서비스에서 자동으로 버전을 관리합니다. 서비스가 체크포인트를 사용하여 애플리케이션을 다시 시작하는 경우 최신 체크포인트가 사용됩니다.
- 세이브포인트는 작업 파라미터를 사용하여 버전이 지정됩니다. SnapshotNameCreateApplicationSnapshot

Managed Service for Apache Flink 는 체크포인트와 세이브포인트에 저장된 데이터를 암호화합니다.

Managed Service for Apache Flink의 인프라 보안

관리형 서비스인 Managed Service for Apache Flink는 Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS글로벌 네트워크 보안 절차로 보호됩니다.

AWS에서 게시한 API 호출을 사용하여 네트워크를 통해 Managed Service for Apache Flink에 액세스합니다. Managed Service for Apache Flink의 모든 API 호출은 전송 계층 보안(TLS)을 통해 보호되며 IAM을 통해 인증됩니다. 클라이언트는 TLS 1.2 이상을 지원해야 합니다. 클라이언트는 Ephemeral Diffie-Hellman(DHE) 또는 Elliptic Curve Ephemeral Diffie-Hellman(ECDHE)과 같은 PFS(전달 완전 보안, Perfect Forward Secrecy)가 포함된 암호 제품군도 지원해야 합니다. Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

또한 요청은 액세스 키 자격 증명 및 IAM 보안 주체와 관련된 비밀 액세스 키를 사용하여 서명해야 합니다. 또는 AWS Security Token Service(AWS STS)을(를) 사용하여 임시 보안 인증을 생성하여 요청에 서명할 수 있습니다.

Managed Service for Apache Flink의 보안 모범 사례

Amazon Managed Service for Apache Flink는 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 사용자는 누가 어떤 Managed Service for Apache Flink 리소스에 대해 어떤 권한을 갖는지 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 역할을 사용하여 다른 Amazon 서비스에 액세스

Managed Service for Apache Flink 애플리케이션에 유효한 보안 인증이 있어야 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림 또는 Amazon S3 버킷과 같은 다른 서비스의 리소스에 액세스할 수 있습니다. AWS 보안 인증을 애플리케이션이나 Amazon S3 버킷에 직접 저장하면 안 됩니다. 이러한 보안 인증은 자동으로 교체되지 않으며 손상된 경우 비즈니스에 큰 영향을 줄 수 있는 장기 보안 인증입니다.

대신 IAM 역할을 사용하여 애플리케이션이 다른 리소스에 액세스하기 위한 임시 보안 인증을 관리해야 합니다. 역할을 사용하면 장기 보안 인증을 사용하여 다른 리소스에 액세스할 필요가 없습니다.

자세한 정보는 IAM 사용 설명서에서 다음 주제를 참조하세요.

- IAM 역할
- 역할에 대한 일반적인 시나리오: 사용자, 애플리케이션 및 서비스

종속 리소스에서 서버 측 암호화 구현

저장 데이터와 전송 중인 데이터는 Managed Service for Apache Flink에서 암호화되며, 이 암호화는 비활성화할 수 없습니다. Kinesis 데이터 스트림, Kinesis Data Firehose 시스템, Amazon S3 버킷 등

종속 리소스에서 서버 측 암호화를 구현해야 합니다. 종속 리소스에서 서버 측 암호화 구현에 대한 자세한 내용은 데이터 보호 섹션을 참조하세요.

API 호출을 모니터링하는 데 사용합니다 CloudTrail .

Managed Service for Apache Flink는 Managed Service for Apache Flink에서 사용자, 역할 또는 Amazon 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다.

에서 수집한 CloudTrail 정보를 사용하여 Apache Flink용 관리형 서비스에 이루어진 요청, 요청이 이루어진 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

자세한 설명은 the section called “AWS CloudTrail 사용하기” 섹션을 참조하세요.

Amazon Managed Service for Apache Flink의 로깅 및 모니터링

모니터링은 Managed Service for Apache Flink 애플리케이션의 신뢰성, 가용성 및 성능을 유지하는 중요한 역할을 합니다. 발생하는 다중 지점 실패를 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분으로부터 모니터링 데이터를 수집해야 합니다.

Managed Service for Apache Flink에 대한 모니터링을 시작하기 전에 다음 질문에 대한 답변을 포함하는 모니터링 계획을 작성해야 합니다

- 모니터링의 목표
- 모니터링할 리소스
- 이러한 리소스를 모니터링하는 빈도
- 사용할 모니터링 도구
- 모니터링 작업을 수행할 사람
- 문제 발생 시 알려야 할 대상

다음 단계는 환경의 정상 Managed Service for Apache Flink 성능에 대한 기준을 설정하는 것입니다. 이렇게 하려면 다양한 시간과 다양한 부하 조건에서 성능을 측정해야 합니다. Managed Service for Apache Flink를 모니터링하면서 모니터링 데이터를 저장할 수 있습니다. 이러면 현재 성능 데이터와 비교하고, 일반적인 성능 패턴과 성능 이상을 식별하고, 문제를 해결할 방법을 강구할 수 있습니다.

주제

로깅

로깅은 프로덕션 애플리케이션에서 오류와 장애를 파악하는 데 중요합니다. 그러나 로깅 하위 시스템은 로그 항목을 수집하여 CloudWatch Logs에 전달해야 합니다. 일부 로깅은 괜찮고 바람직하지만 광범위한 로깅은 서비스에 과부하가 걸리고 Flink 애플리케이션이 지연될 수 있습니다. 예외 및 경고를 기록하는 것은 분명 좋은 생각입니다. 하지만 Flink 애플리케이션에서 처리하는 모든 메시지에 대해 로그 메시지를 생성할 수는 없습니다. Flink는 높은 처리량과 짧은 지연 시간에 최적화되어 있지만 로깅 하위 시스템은 그렇지 않습니다. 처리된 모든 메시지에 대해 로그 출력을 생성해야 하는 경우 Flink 애플리케이션 DataStream 내부의 추가 파일과 적절한 싱크를 사용하여 데이터를 Amazon S3 또는 CloudWatch 로 전송하십시오. 이러한 용도로는 Java 로깅 시스템을 사용하지 마세요. 게다가 Managed Service for Apache Flink' `Debug Monitoring Log Level` 설정은 대량의 트래픽을 생성하므로 배압(backpressure)이 발생할 수 있습니다. 애플리케이션 관련 문제를 적극적으로 조사할 때만 사용해야 합니다.

로그 인사이트를 통한 로그 쿼리 CloudWatch

CloudWatch Logs Insights는 로그를 대규모로 쿼리할 수 있는 강력한 서비스입니다. 고객은 이 기능을 활용하여 로그를 신속하게 검색하여 운영 이벤트 중 오류를 식별하고 완화할 수 있습니다.

다음 쿼리는 모든 작업 관리자 로그에서 예외를 찾아 발생 시간에 따라 예외를 정렬합니다.

```
fields @timestamp, @message
| filter isPresent(throwableInformation.0) or isPresent(throwableInformation) or
 @message like /(Error|Exception)/
| sort @timestamp desc
```

기타 유용한 쿼리는 쿼리 예를 참조하세요.

모니터링

프로덕션 환경에서 스트리밍 애플리케이션을 실행할 때는 애플리케이션을 지속적으로 그리고 무기한으로 실행하도록 설정해야 합니다. Flink 애플리케이션뿐만 아니라 모든 구성 요소에 대한 모니터링과 적절한 경보를 구현하는 것이 중요합니다. 그렇지 않으면 새로운 문제를 조기에 발견하지 못하고 문제가 완전히 전개되어 완화하기 훨씬 더 어려워진 후에야 운영상의 이벤트를 인지하게 될 수 있습니다. 모니터링해야 할 일반적인 사항은 다음과 같습니다.

- 소스가 데이터를 수집하고 있나요?
- 소스에서 데이터를 읽나요(소스의 관점에서 볼때)?

- Flink 애플리케이션에서 데이터를 수신하고 있나요?
- Flink 애플리케이션이 이를 따라잡을 수 있나요, 아니면 뒤처지고 있나요?
- Flink 애플리케이션이 데이터를 싱크에 보관하고 있나요(애플리케이션 관점에서 볼 때)?
- 싱크가 데이터를 수신하고 있나요?

그런 다음 Flink 애플리케이션에 대해 좀 더 구체적인 지표를 고려해야 합니다. 이 CloudWatch 대시보드는 좋은 출발점을 제공합니다. 프로덕션 애플리케이션에 대해 모니터링할 지표에 대한 자세한 내용은 Apache Flink용 Amazon 매니지드 서비스에서 CloudWatch 알람 사용하기을 참조하세요. 이러한 지표에는 다음이 포함됩니다.

- records_lag_max 및 millisBehindLatest — 애플리케이션이 Kinesis 또는 Kafka를 사용하고 있는 경우, 이러한 지표는 애플리케이션이 지연되고 있는지 여부를 나타내며 현재 로드를 따라잡으려면 규모를 조정(스케일 인)해야 합니다. 이는 모든 종류의 애플리케이션에서 쉽게 추적할 수 있는 유용한 일반 지표입니다. 하지만 애플리케이션이 이미 뒤쳐진 경우 등 반응형 스케일링에만 사용할 수 있습니다.
- CPU사용률 heapMemoryUtilization및 — 이러한 지표는 애플리케이션의 전체 리소스 사용률을 잘 나타내며, 애플리케이션이 I/O 바인딩되지 않는 한 사전 예방적 확장에 사용할 수 있습니다.
- 다운타임 - 다운타임이 0보다 크면 애플리케이션에 장애가 발생한 것입니다. 값이 0보다 크면 애플리케이션에서 데이터를 처리하지 않는 것입니다.
- lastCheckpointSize그리고 lastCheckpointDuration— 이러한 지표는 상태에 저장된 데이터의 양과 체크포인트를 생성하는 데 걸리는 시간을 모니터링합니다. 체크포인트가 늘어나거나 시간이 오래 걸리면 애플리케이션은 체크포인트에 지속적으로 시간을 소비하게 되고 실제 처리에 소요되는 주기도 줄어듭니다. 어떤 지점에서는 체크포인트가 너무 커지거나 너무 오래 걸려 실패할 수 있습니다. 절대값을 모니터링하는 것 외에도 고객은 `RATE(lastCheckpointSize)` 및 `RATE(lastCheckpointDuration)`를 사용하여 변경률을 모니터링하는 것도 고려해야 합니다.
- numberOfFailed체크포인트 — 이 지표는 애플리케이션이 시작된 이후 실패한 체크포인트 수를 계산합니다. 애플리케이션에 따라 가끔 체크포인트에 장애가 발생해도 괜찮을 수 있습니다. 그러나 체크포인트에 정기적으로 장애가 발생하면 애플리케이션이 비정상일 가능성이 높으므로 추가 주의가 필요합니다. 절대값이 아닌 경사에 대한 경보를 위해 `RATE(numberOfFailedCheckpoints)`를 모니터링하는 것이 좋습니다.

애플리케이션 로깅 설정

Apache Flink용 관리형 서비스 애플리케이션에 Amazon CloudWatch 로깅 옵션을 추가하면 애플리케이션 이벤트 또는 구성 문제를 모니터링할 수 있습니다.

이 주제에서는 애플리케이션 이벤트를 CloudWatch 로그 스트림에 기록하도록 애플리케이션을 구성하는 방법을 설명합니다. CloudWatch 로깅 옵션은 애플리케이션이 애플리케이션 이벤트를 CloudWatch 로그에 기록하는 방식을 구성하는 데 사용하는 애플리케이션 설정 및 권한 모음입니다. AWS Management Console또는 AWS Command Line Interface (AWS CLI) 를 사용하여 CloudWatch 로깅 옵션을 추가하고 구성할 수 있습니다.

애플리케이션에 CloudWatch 로깅 옵션을 추가하는 방법에 대한 다음 내용을 참고하십시오.

- 콘솔을 사용하여 CloudWatch 로깅 옵션을 추가하면 Managed Service for Apache Flink가 자동으로 CloudWatch 로그 그룹과 로그 스트림을 생성하고 애플리케이션이 로그 스트림에 기록하는 데 필요한 권한을 추가합니다.
- API를 사용하여 CloudWatch 로깅 옵션을 추가할 때는 애플리케이션의 로그 그룹과 로그 스트림을 생성하고 애플리케이션이 로그 스트림에 기록하는 데 필요한 권한을 추가해야 합니다.

이 주제는 다음 섹션을 포함하고 있습니다.

콘솔을 사용한 CloudWatch 로깅 설정

콘솔에서 애플리케이션 CloudWatch 로깅을 활성화하면 CloudWatch 로그 그룹과 로그 스트림이 자동으로 생성됩니다. 또한 스트림에 쓸 수 있는 권한으로 애플리케이션의 권한 정책이 업데이트됩니다.

Apache Flink용 관리형 서비스는 다음 규칙에 따라 이름이 지정된 로그 그룹을 생성합니다. 여기서 *ApplicationName*은 애플리케이션 이름입니다.

```
/AWS/KinesisAnalytics/ApplicationName
```

Managed Service for Apache Flink는 새 로그 그룹에 다음과 같은 이름으로 로그 스트림을 생성합니다.

```
kinesis-analytics-log-stream
```

애플리케이션 구성 페이지의 모니터링 로그 수준 섹션을 사용하여 애플리케이션 모니터링 지표 수준 및 모니터링 로그 수준을 설정합니다. 애플리케이션 로그 수준에 대한 자세한 내용은 the section called “애플리케이션 모니터링 수준”을 참조하세요.

CLI를 사용한 CloudWatch 로깅 설정

를 사용하여 CloudWatch 로깅 옵션을 추가하려면 다음과 AWS CLI 같이 하십시오.

- CloudWatch 로그 그룹과 로그 스트림을 생성합니다.
- 작업을 사용하여 응용 프로그램을 만들 때 로깅 옵션을 추가하거나 CreateApplication작업을 사용하여 기존 응용 프로그램에 로깅 옵션을 추가합니다. AddApplicationCloudWatchLoggingOption
- 애플리케이션 정책에 로그에 쓸 수 있는 권한을 추가합니다.

이 섹션은 다음 주제를 포함합니다:

- CloudWatch 로그 그룹 및 로그 스트림 생성
- 응용 프로그램 CloudWatch 로깅 옵션 사용
- CloudWatch 로그 스트림에 쓰기 권한 추가

CloudWatch 로그 그룹 및 로그 스트림 생성

CloudWatch 로그 콘솔 또는 API를 사용하여 로그 그룹과 스트리밍을 생성합니다. CloudWatch CloudWatch 로그 그룹 및 로그 스트림을 만드는 방법에 대한 자세한 내용은 로그 그룹 및 로그 스트림 작업을 참조하십시오.

응용 프로그램 CloudWatch 로깅 옵션 사용

다음 API 작업을 사용하여 새 애플리케이션 또는 기존 애플리케이션에 CloudWatch 로그 옵션을 추가하거나 기존 애플리케이션의 로그 옵션을 변경할 수 있습니다. JSON 파일을 사용하여 API 작업을 입력하는 방법에 대한 자세한 방법은 Managed Service for Apache Flink API 예 코드 섹션을 참조하세요.

애플리케이션 생성 시 CloudWatch 로그 옵션 추가

다음 예제는 애플리케이션을 만들 때 `CreateApplication` 작업을 사용하여 CloudWatch 로그 옵션을 추가하는 방법을 보여줍니다. 이 예시에서는 *`# ####### ### CloudWatch ## # ### Amazon ### ## (ARN) #`* 자체 정보로 대체합니다. 이러한 작업에 대한 자세한 내용은 `CreateApplication` 섹션을 참조하세요.

```
{
```

```
    "ApplicationName": "test",
    "ApplicationDescription": "test-application-description",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation":{
                            "BucketARN": "arn:aws:s3:::mybucket",
                            "FileKey": "myflink.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        }
    },
    "CloudWatchLoggingOptions": [{
      "LogStreamARN": "<Amazon Resource Name (ARN) of the CloudWatch log stream to add
 to the new application>"
 }]
}
```

기존 CloudWatch 애플리케이션에 로그 옵션 추가

다음 예제는 AddApplicationCloudWatchLoggingOption 작업을 사용하여 기존 애플리케이션에 CloudWatch 로그 옵션을 추가하는 방법을 보여줍니다. 예제에서 각 *### ## ## ###*를 자신의 정보로 바꿉니다. 이러한 작업에 대한 자세한 내용은 AddApplicationCloudWatchLoggingOption 섹션을 참조하세요.

```
{
   "ApplicationName": "<Name of the application to add the log option to>",
   "CloudWatchLoggingOption": {
      "LogStreamARN": "<ARN of the log stream to add to the application>"
   },
   "CurrentApplicationVersionId": <Version of the application to add the log to>
}
```

기존 CloudWatch 로그 옵션 업데이트

다음 예제는 UpdateApplication 작업을 사용하여 기존 CloudWatch 로그 옵션을 수정하는 방법을 보여줍니다. 예제에서 각 *### ## ## ###*를 자신의 정보로 바꿉니다. 이러한 작업에 대한 자세한 내용은 UpdateApplication 섹션을 참조하세요.

```
{
   "ApplicationName": "<Name of the application to update the log option for>",
   "CloudWatchLoggingOptionUpdates": [
         {
            "CloudWatchLoggingOptionId": "<ID of the logging option to modify>",
            "LogStreamARNUpdate": "<ARN of the new log stream to use>"
         }
      ],
   "CurrentApplicationVersionId": <ID of the application version to modify>
}
```

애플리케이션에서 CloudWatch 로그 옵션 삭제

다음 예제는 DeleteApplicationCloudWatchLoggingOption 작업을 사용하여 기존 CloudWatch 로그 옵션을 삭제하는 방법을 보여줍니다. 예제에서 각 *# ## ## ## ###*를 자신의 정보로 바꿉니다. 이러한 작업에 대한 자세한 내용은 DeleteApplicationCloudWatchLoggingOption 섹션을 참조하세요.

```
{
   "ApplicationName": "<Name of application to delete log option from>",
   "CloudWatchLoggingOptionId": "<ID of the application log option to delete>",
   "CurrentApplicationVersionId": <Version of the application to delete the log option
 from>
}
```

애플리케이션 로깅 수준 설정

애플리케이션 로깅 수준을 설정하려면 CreateApplication 작업의 MonitoringConfiguration 파라미터 또는 UpdateApplication 작업의 MonitoringConfigurationUpdate 파라미터를 사용합니다.

애플리케이션 로그 수준에 대한 자세한 내용은 the section called “애플리케이션 모니터링 수준”을 참조하세요.

애플리케이션을 생성할 때 애플리케이션 로깅 수준을 설정합니다.

다음 예제의 CreateApplication요청은 애플리케이션 로그 수준을 INFO로 설정합니다.

```
{
   "ApplicationName": "MyApplication",
   "ApplicationDescription": "My Application Description",
   "ApplicationConfiguration": {
      "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "CodeContentType":"ZIPFILE"
      },
      "FlinkApplicationConfiguration":
         "MonitoringConfiguration": {
            "ConfigurationType": "CUSTOM",
            "LogLevel": "INFO"
         }
      },
   "RuntimeEnvironment": "FLINK-1_15",
   "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole"
}
```

애플리케이션 로깅 수준 업데이트

다음 예제의 UpdateApplication요청은 애플리케이션 로그 수준을 INFO로 설정합니다.

```
{
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "MonitoringConfigurationUpdate": {
            "ConfigurationTypeUpdate": "CUSTOM",
            "LogLevelUpdate": "INFO"
         }
      }
   }
}
```

CloudWatch 로그 스트림에 쓰기 권한 추가

Apache Flink용 관리형 서비스에는 잘못된 구성 오류를 기록할 수 있는 권한이 필요합니다. CloudWatch Managed Service for Apache Flink가 맡는 AWS Identity and Access Management(IAM) 역할에 이러한 권한을 추가할 수 있습니다.

Managed Service for Apache Flink에 IAM 역할을 사용하는 방법에 대한 자세한 내용은 Amazon Managed Service for Apache Flink의 자격 증명 및 액세스 관리을 참조하세요.

신뢰 정책

Managed Service for Apache Flink에 권한을 부여하여 IAM 역할을 맡게하려면 다음의 신뢰 정책을 해당 역할에 연결합니다.

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

권한 정책

Apache Flink용 관리형 서비스 CloudWatch 리소스에서 로그 이벤트를 기록할 수 있는 권한을 애플리케이션에 부여하려면 다음 IAM 권한 정책을 사용할 수 있습니다. 로그 그룹과 스트림에 대한 올바른 Amazon 리소스 이름(ARN)을 입력합니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt0123456789000",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents",
                "logs:DescribeLogGroups",
```

```
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:log-
stream:my-log-stream*",
                "arn:aws:logs:us-east-1:123456789012:log-group:my-log-group:*",
                "arn:aws:logs:us-east-1:123456789012:log-group:*",
            ]
        }
    ]
}
```

애플리케이션 모니터링 수준

애플리케이션의 모니터링 지표 수준 및 모니터링 로그 수준을 사용하여 애플리케이션 로그 메시지 생성을 제어할 수 있습니다.

애플리케이션의 모니터링 지표 수준은 로그 메시지의 세분성을 제어합니다. 모니터링 지표 수준은 다음과 같이 정의됩니다.

- 애플리케이션: 지표의 범위는 전체 애플리케이션으로 지정됩니다.
- 작업: 지표의 범위는 각 작업별로 지정됩니다. 작업에 대한 자세한 내용은 the section called “스케일링”을 참조하세요.
- 연산자: 지표의 범위는 각 연산자별로 지정됩니다. 연산자에 대한 자세한 내용은 the section called “DataStream API 연산자”을 참조하세요.
- 병렬 처리: 지표의 범위는 애플리케이션 병렬 처리로 지정됩니다. API의 MonitoringConfigurationUpdate파라미터를 사용해서만 이 지표 수준을 설정할 수 있습니다. UpdateApplication 콘솔을 사용하여 이 지표 수준을 설정할 수는 없습니다. 병렬 처리에 대한 자세한 내용은 the section called “스케일링”을 참조하세요

애플리케이션의 모니터링 로그 수준은 애플리케이션 로그의 상세 정보를 제어합니다. 모니터링 로그 수준은 다음과 같이 정의됩니다.

- 오류: 애플리케이션의 잠재적 재해 이벤트.
- 경고: 애플리케이션의 잠재적으로 위험한 상황입니다.
- 정보: 애플리케이션의 정보 및 일시적 오류 이벤트. 이 로그 수준을 사용하는 것이 좋습니다.
- 디버그: 애플리케이션을 디버깅하는 데 가장 유용한 세분화된 정보 이벤트입니다. 참고: 이 수준은 임시 디버깅 용도로만 사용하세요.

로깅 모범 사례

애플리케이션에서는 정보 로깅 수준을 사용하는 것이 좋습니다. 오류 수준이 아닌 정보 수준에서 기록되는 Apache Flink 오류가 표시되도록 하려면 이 수준을 사용하는 것이 좋습니다.

애플리케이션 문제를 조사하는 동안에는 디버그 수준을 일시적으로만 사용하는 것이 좋습니다. 문제가 해결되면 정보 수준으로 다시 전환하세요. 디버그 로깅 수준을 사용하면 애플리케이션 성능이 크게 영향을 받습니다.

과도한 로깅은 애플리케이션 성능에도 상당한 영향을 미칠 수 있습니다. 예를 들어 처리된 모든 레코드에 대해 로그 항목을 작성하지 않는 것이 좋습니다. 과도한 로깅은 데이터 처리에 심각한 병목 현상을 일으킬 수 있으며, 소스에서 데이터를 읽는 데 배압이 발생할 수 있습니다.

로깅 문제 해결

애플리케이션 로그가 로그 스트림에 기록되지 않는 경우 다음을 확인하세요.

- 애플리케이션의 IAM 역할 및 정책이 올바른지 확인하세요. 로그 스트림에 액세스하려면 애플리케이션 정책에 다음과 같은 권한이 필요합니다.
  - `logs:PutLogEvents`
  - `logs:DescribeLogGroups`
  - `logs:DescribeLogStreams`

  자세한 설명은 the section called “ CloudWatch 로그 스트림에 쓰기 권한 추가” 섹션을 참조하세요.
- 애플리케이션이 실행 중인지 확인합니다. 애플리케이션 상태를 확인하려면 콘솔에서 애플리케이션 페이지를 보거나 DescribeApplication또는 ListApplications작업을 사용하십시오.
- 기타 애플리케이션 문제를 `downtime` 진단하는 등의 CloudWatch 지표를 모니터링하세요. CloudWatch 지표 읽기에 대한 자세한 내용은 을 참조하십시오Managed Service for Apache Flink의 지표 및 차원.

다음 단계

애플리케이션에서 CloudWatch 로깅을 활성화한 후에는 CloudWatch Logs Insights를 사용하여 애플리케이션 로그를 분석할 수 있습니다. 자세한 설명은 the section called “로그 분석” 섹션을 참조하세요.

로그 인사이트를 통한 CloudWatch 로그 분석

이전 섹션에서 설명한 대로 애플리케이션에 CloudWatch 로깅 옵션을 추가한 후에는 CloudWatch Logs Insights를 사용하여 로그 스트림에서 특정 이벤트 또는 오류를 쿼리할 수 있습니다.

CloudWatch Logs Insights를 사용하면 Logs의 로그 데이터를 대화형 방식으로 검색하고 분석할 수 있습니다. CloudWatch

로그 인사이트를 시작하는 방법에 대한 자세한 내용은 CloudWatch 로그 인사이트를 통한 CloudWatch 로그 데이터 분석을 참조하십시오.

샘플 쿼리 실행

이 섹션에서는 샘플 CloudWatch Logs Insights 쿼리를 실행하는 방법을 설명합니다.

사전 조건

- Logs에 설정된 기존 로그 그룹 및 CloudWatch 로그 스트림.
- 로그에 저장된 기존 CloudWatch 로그.

Amazon Route 53 또는 Amazon VPC와 AWS CloudTrail 같은 서비스를 사용하는 경우 해당 서비스에서 로그로 이동하도록 CloudWatch 이미 로그를 설정했을 것입니다. 로그로 CloudWatch 로그를 보내는 방법에 대한 자세한 내용은 로그 시작하기를 참조하십시오. CloudWatch

CloudWatch Logs Insights의 쿼리는 로그 이벤트의 필드 집합이나 로그 이벤트에 대해 수행된 수학적 집계 또는 기타 작업의 결과를 반환합니다. 이 섹션에서는 로그 이벤트 목록을 반환하는 쿼리를 보여줍니다.

CloudWatch Logs Insights 샘플 쿼리를 실행하려면

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 Insights를 선택합니다.
3. 화면 상단의 쿼리 편집기에는 가장 최근의 로그 이벤트 20개를 반환하는 기본 쿼리가 포함되어 있습니다. 쿼리 편집기 위에서 쿼리할 로그 그룹을 선택합니다.

   로그 그룹을 선택하면 CloudWatch Logs Insights는 로그 그룹의 데이터에서 필드를 자동으로 탐지하여 오른쪽 창의 검색된 필드에 표시합니다. 또한 이 로그 그룹의 로그 이벤트를 시간의 흐름에

따라 보여주는 막대 그래프도 표시합니다. 이 막대 그래프는 단순히 테이블에 표시된 이벤트가 아니라 쿼리 및 시간 범위와 일치하는 로그 그룹 내 이벤트의 분포를 보여줍니다.

4. 쿼리 실행을 선택합니다.

   쿼리의 결과가 표시됩니다. 이 예제에서 결과는 모든 유형의 최신 로그 이벤트 20개입니다.

5. 반환된 로그 이벤트 중 하나에 대한 모든 필드를 보려면 해당 로그 이벤트 왼쪽에 있는 화살표를 선택합니다.

CloudWatch Logs Insights 쿼리를 실행하고 수정하는 방법에 대한 자세한 내용은 샘플 쿼리 실행 및 수정을 참조하십시오.

쿼리 예제

이 섹션에는 Apache Flink용 관리 서비스 응용 프로그램 CloudWatch 로그를 분석하기 위한 Logs Insights 예제 쿼리가 포함되어 있습니다. 이러한 쿼리는 몇 가지 예제 오류 조건을 검색하고 다른 오류 조건을 찾는 쿼리를 작성하기 위한 템플릿 역할을 합니다.

> **Note**
>
> 다음 쿼리 예제의 지역 (*us-west-2*), 계정 ID (*012345678901*) 및 애플리케이션 이름 (*YourApplication*) 을 애플리케이션의 지역 및 계정 ID로 바꾸십시오.

이 주제는 다음 섹션을 포함하고 있습니다.

- 작업 분석: 작업 분배
- 작업 분석: 병렬 변경
- 오류 분석: 액세스 거부됨
- 오류 분석: 소스 또는 싱크를 찾을 수 없음
- 오류 분석: 애플리케이션 작업 관련 실패

작업 분석: 작업 분배

다음 CloudWatch 로그 인사이트 쿼리는 Apache Flink Job Manager가 작업 관리자 간에 배포하는 작업 수를 반환합니다. 쿼리가 이전 작업의 작업을 반환하지 않도록 쿼리 시간을 한 작업 실행과 일치하도록 설정해야 합니다. 병렬에 관한 자세한 내용은 스케일링을 참조하세요.

```
fields @timestamp, message
| filter message like /Deploying/
| parse message " to flink-taskmanager-*" as @tmid
| stats count(*) by @tmid
| sort @timestamp desc
| limit 2000
```

다음 CloudWatch Logs Insights 쿼리는 각 작업 관리자에 할당된 하위 작업을 반환합니다. 총 하위 작업 수는 모든 작업의 병렬 처리 수를 합한 값입니다. 작업 병렬은 연산자 병렬에서 파생되며 코드에서 `setParallelism`을 지정하여 변경하지 않는 한 기본적으로 애플리케이션의 병렬과 동일합니다. 연산자 병렬 처리 설정에 대한 자세한 내용은 Apache Flink 설명서의 병렬 처리 설정: 연산자 수준을 참조하세요.

```
fields @timestamp, @tmid, @subtask
| filter message like /Deploying/
| parse message "Deploying * to flink-taskmanager-*" as @subtask, @tmid
| sort @timestamp desc
| limit 2000
```

작업 예약에 대한 자세한 내용은 Apache Flink 설명서의 작업 및 예약을 참조하세요.

작업 분석: 병렬 변경

다음 CloudWatch Logs Insights 쿼리는 애플리케이션의 병렬 처리 (예: 자동 크기 조정) 에 대한 변경 사항을 반환합니다. 또한 이 쿼리는 애플리케이션 병렬 처리에 대한 수동 변경 내용을 반환합니다. 자동 스케일링에 대한 자세한 내용은 the section called “자동 조정”을 참조하세요.

```
fields @timestamp, @parallelism
| filter message like /property: parallelism.default, /
| parse message "default, *" as @parallelism
| sort @timestamp asc
```

오류 분석: 액세스 거부됨

다음 CloudWatch 로그 인사이트 쿼리는 로그를 반환합니다`Access Denied`.

```
fields @timestamp, @message, @messageType
| filter applicationARN like /arn:aws:kinesisanalytics us-
west-2:012345678901:application\/YourApplication/
| filter @message like /AccessDenied/
```

```
| sort @timestamp desc
```

오류 분석: 소스 또는 싱크를 찾을 수 없음

다음 CloudWatch 로그 인사이트 쿼리는 ResourceNotFound 로그를 반환합니다. ResourceNotFoundKinesis 소스 또는 싱크를 찾을 수 없는 경우 결과를 기록합니다.

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalyticsus-
west-2:012345678901:application\/YourApplication/
| filter @message like /ResourceNotFoundException/
| sort @timestamp desc
```

오류 분석: 애플리케이션 작업 관련 실패

다음 CloudWatch Logs Insights 쿼리는 애플리케이션의 작업 관련 실패 로그를 반환합니다. 이러한 로그는 애플리케이션 상태가 RUNNING에서 RESTARTING로 전환될 때 발생합니다.

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalyticsus-
west-2:012345678901:application\/YourApplication/
| filter @message like /switched from RUNNING to RESTARTING/
| sort @timestamp desc
```

Apache Flink 버전 1.8.2 이하를 사용하는 애플리케이션의 경우 작업 관련 오류가 발생하면 애플리케이션 상태가 RUNNING에서 FAILED로 전환되는 결과를 낳습니다. Apache Flink 1.8.2 이전 버전을 사용하는 경우 다음 쿼리를 사용하여 애플리케이션 작업 관련 실패를 검색하세요.

```
fields @timestamp,@message
| filter applicationARN like /arn:aws:kinesisanalyticsus-
west-2:012345678901:application\/YourApplication/
| filter @message like /switched from RUNNING to FAILED/
| sort @timestamp desc
```

Managed Service for Apache Flink에서 지표 및 차원 보기

이 주제는 다음 섹션을 포함하고 있습니다.

- 애플리케이션 지표
- Kinesis Data Streams 커넥터 지표

- Amazon MSK 커넥터 지표
- Apache Zeppelin 지표
- 지표 보기 CloudWatch
- CloudWatch 지표 보고 수준 설정
- Amazon Managed Service for Apache Flink에서 사용자 지정 지표 사용하기
- Apache Flink용 Amazon 매니지드 서비스에서 CloudWatch 알람 사용하기

Apache Flink용 관리형 서비스가 데이터 소스를 처리할 때 Apache Flink용 관리형 서비스는 다음 지표와 차원을 Amazon에 보고합니다. CloudWatch

애플리케이션 지표

지표	단위	설명	수준	사용 관련 참고 사항
`backPressuredTimeMsPerSecond*`	밀리초	이 작업 또는 연산자가 초당 배압을 받는 시간(밀리초)입니다.	작업, 연산자, 병렬	*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다. 이러한 지표는 애플리케이션의 병목 현상을 식별하는 데 유용할 수 있습니다.
`busyTimeMsPerSecond*`	밀리초	이 작업 또는 연산자가 초당 사용 중인(유휴 상태도 배압 상태도 아닌) 시간(밀리초)입니	작업, 연산자, 병렬	*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에

지표	단위	설명	수준	사용 관련 참고 사항	
		다. 값을 계산할 수 없는 경우 NaN이 될 수 있습니다.		서만 사용할 수 있습니다. 이러한 지표는 애플리케이션의 병목 현상을 식별하는 데 유용할 수 있습니다.	
`cpuUtilization`	백분율	작업 관리자 전체에서 CPU 사용률의 전체 비율입니다. 예를 들어 작업 관리자가 5명인 경우 Managed Service for Apache Flink는 보고 간격당 이 지표의 샘플 5개를 게시합니다.	애플리케이션	이 지표를 사용하여 애플리케이션의 최소, 평균 및 최대 CPU 사용률을 모니터링할 수 있습니다. 이 `CPUUtilization` 지표는 컨테이너 내에서 실행 중인 TaskManager JVM 프로세스의 CPU 사용량만 고려합니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
`containerCPUUtilization`	백분율	Flink 애플리케이션 클러스터의 작업 관리자 컨테이너 전체 CPU 사용률 비율입니다. 예를 들어 작업 관리자가 5명이고 그에 따라 TaskManager 컨테이너가 다섯 개이고 Managed Service for Apache Flink는 1분 보고 간격당 이 지표의 샘플을 2개* 5개 게시합니다.	애플리케이션	컨테이너별로 다음과 같이 계산됩니다. 컨테이너가 사용한 총 CPU 시간(초) * 100 / 컨테이너 CPU 제한(CPU/초) `CPUUtilization` 지표에는 컨테이너 내에서 실행되는 TaskManager JVM 프로세스의 CPU 사용량만 고려됩니다. 동일한 컨테이너 내에서 JVM 외부에서 실행되는 다른 구성 요소도 있습니다. 이 `containerCPUUtilization` 지표는 컨테이너의 CPU 소모 및 이로 인한 장애 측면에서 모든 프로세스를 포함하여 보다 완	

지표	단위	설명	수준	사용 관련 참고 사항	
				전한 그림을 제공합니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
containerMemoryUtilization	백분율	Flink 애플리케이션 클러스터의 작업 관리자 컨테이너 전체 메모리 사용률입니다. 예를 들어 작업 관리자가 5명이고 그에 따라 TaskManager 컨테이너가 다섯 개이고 Managed Service for Apache Flink는 1분 보고 간격당 이 지표의 샘플을 2개* 5개 게시합니다.	애플리케이션	컨테이너별로 다음과 같이 계산됩니다. 컨테이너 메모리 사용량(바이트) * 100/ 포드 배포 사양에 따른 컨테이너 메모리 제한(바이트) HeapMemoryUtilization 및 ManagedMemoryUtilzations 지표는 JVM의 힙 메모리 사용량 또는 관리 메모리 (RocksDB State Backend와 같은 네이티브 프로세스의 TaskManager JVM 외부 메모리 사용량) 와 같은 특정 메모리 지표만 고려합니다. 이 containerMemoryUtilization 지	

지표	단위	설명	수준	사용 관련 참고 사항	
				표는 작업 세트 메모리를 포함함으로써 전체 메모리 소모를 더 잘 추적할 수 있는 작업 세트 메모리를 포함하여 더 완전한 그림을 제공합니다. 용량이 고갈되면 파드에 문제가 생길 수 있습니다. `Out of Memory Error` TaskManager	

지표	단위	설명	수준	사용 관련 참고 사항	
containerDiskUtilization	백분율	Flink 애플리케이션 클러스터의 작업 관리자 컨테이너 전체 디스크 사용률입니다. 예를 들어 작업 관리자가 5명이고 그에 따라 TaskManager 컨테이너가 다섯 개이고 Apache Flink용 Managed Service for Apache Flink는 1분 보고 간격당 이 지표의 샘플을 2* 5개 게시합니다.	애플리케이션	컨테이너별로 다음과 같이 계산됩니다. 디스크 사용량(바이트) * 100 / 컨테이너의 디스크 제한(바이트) 컨테이너의 경우 컨테이너의 루트 볼륨이 설정된 파일 시스템의 사용률을 나타냅니다.	
currentInputWatermark	밀리초	이 애플리케이션/연산자/작업/스레드가 받은 마지막 워터마크	애플리케이션, 연산자, 작업, 병렬 처리	이 레코드는 입력이 두 개 있는 차원에 대해서만 생성됩니다. 이는 마지막으로 수신한 워터마크의 최소값입니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
currentOutputWatermark	밀리초	이 애플리케이션/연산자/작업/스레드가 생성한 마지막 워터마크	애플리케이션, 연산자, 작업, 병렬 처리		
downtime	밀리초	현재 실패/복구 중인 작업의 경우 이 중단이 발생한 동안 경과된 시간입니다.	애플리케이션	이 지표는 작업이 실패하거나 복구되는 동안 경과된 시간을 측정합니다. 이 지표는 실행 중인 작업의 경우 0을 반환하고 완료된 작업에 대해 -1을 반환합니다. 이 지표가 0 또는 -1이 아닌 경우 애플리케이션의 Apache Flink 작업이 실행되지 않았음을 나타냅니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
fullRestarts	개수	제출된 이후 이 작업이 완전히 다시 시작된 총 횟수입니다. 이 지표는 세분화된 재시작을 측정하지 않습니다.	애플리케이션	이 지표를 사용하여 일반적인 애플리케이션 상태를 평가할 수 있습니다. 내부 유지 관리 중에 Managed Service for Apache Flink에 의해 재시작이 발생할 수 있습니다. 재시작 빈도가 정상보다 높으면 애플리케이션에 문제가 있음을 나타냅니다.	
heapMemoryUtilization	백분율	작업 관리자 전반의 전체 힙 메모리 사용률입니다. 예를 들어 작업 관리자가 5명인 경우 Managed Service for Apache Flink는 보고 간격당 이 지표의 샘플 5개를 게시합니다.	애플리케이션	이 지표를 사용하여 애플리케이션의 최소, 평균 및 최대 힙 메모리 사용률을 모니터링할 수 있습니다. JVM의 힙 메모리 사용량과 같은 특정 메모리 HeapMemoryUtilization 지표만 설명합니다. TaskManager	

지표	단위	설명	수준	사용 관련 참고 사항	
idleTimeMsPerSecond*	밀리초	이 작업 또는 연산자가 유휴 상태(처리할 데이터가 없음)인 초당 시간(밀리초)입니다. 유휴 시간에는 배압이 가해진 시간이 제외되므로 작업에 배압이 가해져도 작업이 유휴 상태가 아닙니다.	작업, 연산자, 병렬	*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다. 이러한 지표는 애플리케이션의 병목 현상을 식별하는 데 유용할 수 있습니다.	
lastCheckpointSize	바이트	마지막 체크포인트의 총 크기	애플리케이션	이 지표를 사용하여 실행 중인 애플리케이션 스토리지 사용률을 확인할 수 있습니다. 이 지표의 값이 증가하면 메모리 누수 또는 병목 현상 등 애플리케이션에 문제가 있음을 의미할 수 있습니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
lastCheckpointDuration	밀리초	마지막 체크포인트를 완료하는 데 걸린 시간	애플리케이션	이 지표는 가장 최근의 체크포인트를 완료하는 데 걸린 시간을 측정합니다. 이 지표의 값이 증가하면 메모리 누수 또는 병목 현상 등 애플리케이션에 문제가 있음을 의미할 수 있습니다. 경우에 따라 체크포인트를 사용하지 않도록 설정하여 이 문제를 해결할 수 있습니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
managedMemoryUsed*	바이트	현재 사용 중인 관리형 메모리의 양입니다.	애플리케이션, 연산자, 작업, 병렬 처리	*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다. 이는 Java 힙 외부에서 Flink가 관리하는 메모리와 관련이 있습니다. RocksDB 상태 백엔드에 사용되며 애플리케이션에서도 사용할 수 있습니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
managedMemoryTotal *	바이트	관리형 메모리의 총량입니다.	애플리케이션, 연산자, 작업, 병렬 처리	*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다. 이는 Java 힙 외부에서 Flink가 관리하는 메모리와 관련이 있습니다. RocksDB 상태 백엔드에 사용되며 애플리케이션에서도 사용할 수 있습니다. managedMemoryUtilizations 지표는 관리형 메모리(RocksDB State Backend 등 네이티브 프로세스의 JVM 외부 메모리 사용량)와 같은 특정 메모리 지표만 설명합니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
managedMemoryUtilization*	백분율	/에서 파생됨 managedMemoryUsed managedMemoryTotal	애플리케이션, 연산자, 작업, 병렬 처리	*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다. 이는 Java 힙 외부에서 Flink가 관리하는 메모리와 관련이 있습니다. RocksDB 상태 백엔드에 사용되며 애플리케이션에서도 사용할 수 있습니다.	
numberOfFailedCheckpoints	개수	체크포인트가 실패한 횟수입니다.	애플리케이션	이 지표를 사용하여 애플리케이션 상태 및 진행 상황을 모니터링할 수 있습니다. 처리량 또는 권한 문제와 같은 애플리케이션 문제로 인해 체크포인트가 실패할 수 있습니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
numRecordsIn*	개수	해당 애플리케이션, 연산자 또는 작업이 수신한 총 레코드 수입니다.	애플리케이션, 연산자, 작업, 병렬 처리	*일정 기간(초/분)에 대한 SUM 통계를 적용하려면: • 올바른 수준에서 지표를 선택합니다. 연산자의 지표를 추적하는 경우 해당 연산자 지표를 선택해야 합니다. • Managed Service for Apache Flink는 분당 4개의 지표 스냅샷을 생성하므로 다음과 같은 지표 수식을 사용해야 합니다. m1/4, 여기서 m1은 기간(초/분)에 대한 SUM 통계입니다. 지표의 수준은 이 지표가 전체 애플리케이션,	

지표	단위	설명	수준	사용 관련 참고 사항	
				특정 운영자 또는 특정 작업에서 받은 총 레코드 수를 측정할지 여부를 지정합니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
numRecordsInPerSecond*	개수/초	이 애플리케이션, 연산자 또는 작업이 초당 수신한 총 레코드 수입니다.	애플리케이션, 연산자, 작업, 병렬 처리	*일정 기간(초/분)에 대한 SUM 통계를 적용하려면: • 올바른 수준에서 지표를 선택합니다. 연산자의 지표를 추적하는 경우 해당 연산자 지표를 선택해야 합니다. • Managed Service for Apache Flink는 분당 4개의 지표 스냅샷을 생성하므로 다음과 같은 지표 수식을 사용해야 합니다. m1/4, 여기서 m1은 기간(초/분)에 대한 SUM 통계입니다. 지표의 수준은 이 지표가 전체 애플리케이션,	

지표	단위	설명	수준	사용 관련 참고 사항	
				특정 연산자 또는 특정 작업이 초당 수신한 총 레코드 수를 측정하는지 여부를 지정합니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
numRecordsOut*	개수	해당 애플리케이션, 연산자 또는 작업이 생성한 총 레코드 수입니다.	애플리케이션, 연산자, 작업, 병렬 처리	*일정 기간(초/분)에 대한 SUM 통계를 적용하려면: • 올바른 수준에서 지표를 선택합니다. 연산자의 지표를 추적하는 경우 해당 연산자 지표를 선택해야 합니다. • Managed Service for Apache Flink는 분당 4개의 지표 스냅샷을 생성하므로 다음과 같은 지표 수식을 사용해야 합니다. m1/4, 여기서 m1은 기간(초/분)에 대한 SUM 통계입니다. 지표의 수준은 이 지표가 전체 애플리케이션,	

지표	단위	설명	수준	사용 관련 참고 사항	
				특정 운영자 또는 특정 작업에서 내보낸 총 레코드 수를 측정할지 여부를 지정합니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
numLateRecordsDropped*	개수	애플리케이션, 연산자, 작업, 병렬 처리		*일정 기간(초/분)에 대한 SUM 통계를 적용하려면: • 올바른 수준에서 지표를 선택합니다. 연산자의 지표를 추적하는 경우 해당 연산자 지표를 선택해야 합니다. • Managed Service for Apache Flink는 분당 4개의 지표 스냅샷을 생성하므로 다음과 같은 지표 수식을 사용해야 합니다. m1/4, 여기서 m1은 기간(초/분)에 대한 SUM 통계입니다. 이 연산자 또는 작업이 늦게 도착하여 삭제한	

지표	단위	설명	수준	사용 관련 참고 사항	
				레코드 수입니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
numRecordsOutPerSecond*	개수/초	해당 애플리케이션, 연산자 또는 작업이 초당 생성한 총 레코드 수입니다.	애플리케이션, 연산자, 작업, 병렬 처리	*일정 기간(초/분)에 대한 SUM 통계를 적용하려면: • 올바른 수준에서 지표를 선택합니다. 연산자의 지표를 추적하는 경우 해당 연산자 지표를 선택해야 합니다. • Managed Service for Apache Flink는 분당 4개의 지표 스냅샷을 생성하므로 다음과 같은 지표 수식을 사용해야 합니다. m1/4, 여기서 m1은 기간(초/분)에 대한 SUM 통계입니다. 지표의 수준은 이 지표가 전체 애플리케이션,	

지표	단위	설명	수준	사용 관련 참고 사항	
				특정 연산자 또는 특정 작업이 초당 생성한 총 레코드 수를 측정하는지 여부를 지정합니다.	
oldGenerationGCCount	개수	모든 작업 관리자에서 발생한 이전 가비지 수집 작업의 총 수입니다.	애플리케이션		
oldGenerationGCTime	밀리초	이전 가비지 수집 작업을 수행하는 데 소요된 총 시간입니다.	애플리케이션	이 지표를 사용하여 가비지 수집 시간 합계, 평균 및 최대 시간을 모니터링할 수 있습니다.	
threadCount	개수	애플리케이션에서 사용한 총 라이브 스레드 수입니다.	애플리케이션	이 지표는 애플리케이션 코드에서 사용하는 스레드 수를 측정합니다. 이는 애플리케이션 병렬 처리와는 다릅니다.	

지표	단위	설명	수준	사용 관련 참고 사항	
uptime	밀리초	작업이 중단 없이 실행된 시간.	애플리케이션	이 지표를 사용하여 작업이 성공적으로 실행되고 있는지 확인할 수 있습니다. 이 지표는 완료된 작업에 대해 -1을 반환합니다.	

Kinesis Data Streams 커넥터 지표

AWS는 다음 외에도 Kinesis Data Streams에 대한 모든 레코드를 내보냅니다.

지표	단위	설명	수준	사용 관련 참고 사항
millisbehindLatest	밀리초	소비자가 스트림 헤드보다 뒤처진 시간(밀리초)으로, 소비자가 현재 시간보다 얼마나 뒤처져 있는지를 나타냅니다.	애플리케이션 (스트림용), 병렬성 (용) ShardId	• 값이 0이면 레코드 처리를 따라잡았으며 이 시점에서 처리할 새 레코드가 없음을 나타냅니다. 스트림 이름과 샤드 ID로 특정 샤드의 지표를 지정할 수 있습니다. • 값이 -1이면 서비스가 해당 지표의 값을 아직 보고하지 않았

지표	단위	설명	수준	사용 관련 참고 사항
				음을 나타냅니다.
bytesRequestedPerFetch	바이트	getRecords 에 대한 단일 호출에서 요청된 바이트 수입니다.	애플리케이션 (스트림용), 병렬성 (용) ShardId	

Amazon MSK 커넥터 지표

AWS는 다음 외에도 Amazon MSK에 대한 모든 레코드를 내보냅니다.

지표	단위	설명	수준	사용 관련 참고 사항
currentoffsets	N/A	각 파티션에 대한 소비자의 현재 읽기 오프셋입니다. 특정 파티션의 지표는 주제 이름 및 파티션 ID로 지정할 수 있습니다.	애플리케이션 (주제용), 병렬성 (대상) PartitionId	
commitsFailed	N/A	오프셋 커밋과 체크포인트가 활성화된 경우, Kafka에 대한 총 오프셋 커밋 실패 횟수입니다.	애플리케이션, 연산자, 작업, 병렬 처리	오프셋을 카프카에 다시 커밋하는 것은 소비자 진행 상황을 노출하기 위한 수단일 뿐이므로, 커밋 실패는 Flink의 체크포인트 파티션 오프셋의 무결성에 영

지표	단위	설명	수준	사용 관련 참고 사항
				향을 미치지 않습니다.
commitsSucceeded	N/A	오프셋 커밋과 체크포인팅이 활성화된 경우, 카프카에 성공적으로 커밋한 총 오프셋 횟수입니다.	애플리케이션, 연산자, 작업, 병렬 처리	
committed offsets	N/A	각 파티션에 대해 Kafka에 마지막으로 성공적으로 커밋된 오프셋 특정 파티션의 지표는 주제 이름 및 파티션 ID로 지정할 수 있습니다.	애플리케이션 (주제용), 병렬성 (대상) PartitionId	
records_lag_max	개수	이 창에 있는 파티션의 레코드 수 기준 최대 지연 시간	애플리케이션, 연산자, 작업, 병렬 처리	
bytes_consumed_rate	바이트	초당 사용된 주제의 평균 바이트 수	애플리케이션, 연산자, 작업, 병렬 처리	

Apache Zeppelin 지표

Studio 노트북의 경우 AWS는 애플리케이션 수준에서 KPUs, cpuUtilization, heapMemoryUtilization, oldGenerationGCTime, oldGenerationGCCount, threadCount 지표를 내보냅니다. 또한 애플리케이션 수준에서도 다음 테이블에 표시된 지표를 내보냅니다.

지표	단위	설명	Prometheus 이름
zeppelinCpuUtilization	백분율	Apache Zeppelin 서버의 전체 CPU 사용률입니다.	process_cpu_usage
zeppelinHeapMemoryUtilization	백분율	Apache Zeppelin 서버의 전체 힙 메모리 사용률입니다.	jvm_memory_used_bytes
zeppelinThreadCount	개수	Apache Zeppelin 서버에서 사용한 총 라이브 스레드 수입니다.	jvm_threads_live_threads
zeppelinWaitingJobs	개수	스레드를 기다리는 대기 중인 Apache Zeppelin 작업 수입니다.	jetty_threads_jobs
zeppelinServerUptime	초	서버가 가동되어 실행된 총 시간.	process_uptime_seconds

지표 보기 CloudWatch

Amazon CloudWatch 콘솔 또는 를 사용하여 애플리케이션의 CloudWatch 지표를 볼 수 AWS CLI 있습니다.

CloudWatch 콘솔을 사용하여 지표를 보려면

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 창에서 지표(Metrics)를 선택합니다.
3. Apache Flink용 관리 서비스의 범주별 CloudWatch 지표 창에서 지표 범주를 선택합니다.
4. 위쪽 창에서 지표의 전체 목록이 보일 때까지 아래로 스크롤합니다.

AWS CLI을(를) 사용하여 지표를 보려면

- 명령 프롬프트에서 다음 명령을 사용합니다.

```
aws cloudwatch list-metrics --namespace "AWS/KinesisAnalytics" --region region
```

CloudWatch 지표 보고 수준 설정

애플리케이션이 생성하는 애플리케이션 지표의 수준을 제어할 수 있습니다. Managed Service for Apache Flink는 다음 지표 수준을 지원합니다.

- 애플리케이션: 애플리케이션은 각 애플리케이션에 대해 최고 수준의 지표만 보고합니다. Managed Service for Apache Flink 지표는 기본적으로 애플리케이션 수준에서 게시됩니다.
- 작업: 애플리케이션은 초당 애플리케이션에서 들어오고 나가는 레코드 수와 같이 작업 지표 보고 수준으로 정의된 지표에 대한 작업별 지표 차원을 보고합니다.
- 연산자: 애플리케이션은 각 필터 또는 맵 작업에 대한 지표와 같이 연산자 지표 보고 수준으로 정의된 지표에 대한 연산자별 지표 차원을 보고합니다.
- 병렬 처리: 각 실행 스레드에 대한 애플리케이션 보고서 `Task` 및 `Operator` 수준 지표입니다. 이 보고 수준은 과도한 비용으로 인해 병렬 처리 설정이 64 이상인 애플리케이션에는 권장되지 않습니다.

> **Note**
>
> 서비스에서 생성하는 지표 데이터의 양이 많으므로 이 지표 수준은 문제 해결에만 사용해야 합니다. 이 지표 수준은 CLI를 통해서만 설정할 수 있습니다. 콘솔에서는 이 지표 수준을 사용할 수 없습니다.

기본 수준은 애플리케이션입니다. 애플리케이션은 현재 수준 및 모든 상위 수준에서 지표를 보고합니다. 예를 들어 보고 수준이 연산자로 설정된 경우 애플리케이션은 애플리케이션, 작업 및 연산자 지표를 보고합니다.

작업 매개 변수 또는 `CreateApplication`작업 `MonitoringConfiguration` 매개 변수를 사용하여 CloudWatch 지표 보고 수준을 설정합니다. `MonitoringConfigurationUpdate` `UpdateApplication` 다음 예제 `UpdateApplication`작업 요청은 CloudWatch 지표 보고 수준을 Task로 설정합니다.

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
```

```
   "ApplicationConfigurationUpdate": {
      "FlinkApplicationConfigurationUpdate": {
         "MonitoringConfigurationUpdate": {
            "ConfigurationTypeUpdate": "CUSTOM",
            "MetricsLevelUpdate": "TASK"
         }
      }
   }
}
```

CreateApplication 작업의 LogLevel 파라미터 또는 UpdateApplication 작업의 LogLevelUpdate 파라미터를 사용하여 로깅 수준을 구성할 수도 있습니다. 다음 로그 수준을 사용할 수 있습니다.

- ERROR: 잠재적으로 복구 가능한 오류 이벤트를 기록합니다.
- WARN: 오류로 이어질 수 있는 경고 이벤트를 기록합니다.
- INFO: 정보 이벤트를 기록합니다.
- DEBUG: 일반 디버깅 이벤트를 기록합니다.

Log4j 로깅 수준에 대한 자세한 내용은 Apache Log4j 설명서의 사용자 지정 로그 수준을 참조하세요.

Amazon Managed Service for Apache Flink에서 사용자 지정 지표 사용하기

Apache Flink용 관리형 서비스는 리소스 사용량 및 처리량에 대한 메트릭을 포함하여 19개의 메트릭을 CloudWatch 노출합니다. 또한 자체 지표를 만들어 이벤트 처리 또는 외부 리소스 액세스 등 애플리케이션별 데이터를 추적할 수 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- 작동 방식
- 예제
- 사용자 지정 지표 보기

작동 방식

Managed Service for Apache Flink의 사용자 지정 지표는 Apache Flink 지표 시스템을 사용합니다. Amazon Flink 지표에는 다음과 같은 속성이 있습니다.

- 유형: 지표 유형은 데이터를 측정하고 보고하는 방법을 설명합니다. 사용 가능한 Apache Flink 지표 유형에는 개수, 게이지, 히스토그램, 미터 등이 있습니다. Apache Flink 지표 유형에 대한 자세한 내용은 지표 유형을 참조하세요.

> **Note**
>
> AWS CloudWatch 메트릭은 히스토그램 Apache Flink 메트릭 유형을 지원하지 않습니다. CloudWatch 카운트, 게이지 및 미터 유형의 Apache Flink 메트릭만 표시할 수 있습니다.

- 범위: 메트릭의 범위는 해당 식별자와 메트릭이 보고되는 방식을 나타내는 키-값 쌍 세트로 구성됩니다. CloudWatch 지표 식별자는 다음과 같은 요소로 구성됩니다.
  - 지표가 보고되는 수준을 나타내는 시스템 범위(예: 연산자).
  - 사용자 변수 또는 지표 그룹 이름과 같은 속성을 정의하는 사용자 범위입니다. 이러한 속성은 `MetricGroup.addGroup(key, value)` 또는 `MetricGroup.addGroup(name)`를 사용하여 정의됩니다.

  범위에 대한 자세한 내용은 범위를 참조하세요.

Apache Flink 지표에 대한 자세한 내용은 Apache Flink 설명서의 지표를 참조하세요.

Managed Service for Apache Flink에서 사용자 정의 지표를 생성하려면, `RichFunction`를 확장하는 모든 사용자 함수에서 `GetMetricGroup`를 호출하여 Apache Flink 지표 시스템에 액세스할 수 있습니다. 이 메서드는 사용자 지정 지표를 만들고 등록하는 데 사용할 수 있는 MetricGroup개체를 반환합니다. Apache Flink용 관리형 서비스는 그룹 키를 `KinesisAnalytics` 사용하여 생성된 모든 메트릭을 보고합니다. CloudWatch 내가 정의하는 사용자 지정 지표에는 다음과 같은 특성이 있습니다.

- 내 사용자 지정 지표에는 지표 이름과 그룹 이름이 있습니다. 이름은 116자의 영숫자로 구성되어야 합니다.
- 사용자 범위에서 정의한 속성 (`KinesisAnalytics`지표 그룹 제외) 은 차원으로 CloudWatch 게시됩니다.
- 사용자 지정 지표는 기본적으로 해당 `Application` 수준에 게시됩니다.
- 차원(작업/연산자/병렬 처리)은 애플리케이션의 모니터링 수준에 따라 지표에 추가됩니다. 작업의 매개 변수나 CreateApplication작업의 또는 MonitoringConfiguration매개 변수를 사용하여 응용 프로그램의 모니터링 수준을 설정합니다. MonitoringConfigurationUpdateUpdateApplication

예제

다음 코드 예제에서는 사용자 지정 지표를 생성하고 증가시키는 매핑 클래스를 만드는 방법과 매핑 클래스를 DataStream 객체에 추가하여 애플리케이션에서 구현하는 방법을 보여 줍니다.

레코드 수 사용자 지정 지표

다음 코드 예제는 데이터 스트림의 레코드 수를 세는 지표를 생성하는 매핑 클래스를 만드는 방법을 보여줍니다(numRecordsIn 지표와 동일한 기능).

```
    private static class NoOpMapperFunction extends RichMapFunction<String, String> {
        private transient int valueToExpose = 0;
        private final String customMetricName;

        public NoOpMapperFunction(final String customMetricName) {
            this.customMetricName = customMetricName;
        }

        @Override
        public void open(Configuration config) {
            getRuntimeContext().getMetricGroup()
                    .addGroup("KinesisAnalytics")
                    .addGroup("Program", "RecordCountApplication")
                    .addGroup("NoOpMapperFunction")
                    .gauge(customMetricName, (Gauge<Integer>) () -> valueToExpose);
        }

        @Override
        public String map(String value) throws Exception {
            valueToExpose++;
            return value;
        }
    }
```

위 예제에서는 애플리케이션이 처리하는 각 레코드에 대해 valueToExpose 변수가 증가합니다.

매핑 클래스를 정의한 후 맵을 구현하는 인애플리케이션 스트림을 생성합니다.

```
DataStream<String> noopMapperFunctionAfterFilter =
    kinesisProcessed.map(new NoOpMapperFunction("FilteredRecords"));
```

이 애플리케이션의 전체 코드는 레코드 수 사용자 지정 지표 애플리케이션을 참조하세요.

단어 수 사용자 지정 지표

다음 코드 예제는 데이터 스트림에서 단어 수를 계산하는 지표를 생성하는 매핑 클래스를 만드는 방법을 보여 줍니다:

```
private static final class Tokenizer extends RichFlatMapFunction<String, Tuple2<String,
 Integer>> {

            private transient Counter counter;

            @Override
            public void open(Configuration config) {
                this.counter = getRuntimeContext().getMetricGroup()
                        .addGroup("KinesisAnalytics")
                        .addGroup("Service", "WordCountApplication")
                        .addGroup("Tokenizer")
                        .counter("TotalWords");
            }

            @Override
            public void flatMap(String value, Collector<Tuple2<String, Integer>>out) {
                // normalize and split the line
                String[] tokens = value.toLowerCase().split("\\W+");

                // emit the pairs
                for (String token : tokens) {
                    if (token.length() > 0) {
                        counter.inc();
                        out.collect(new Tuple2<>(token, 1));
                    }
                }
            }
        }
```

위 예제에서는 애플리케이션이 처리하는 각 단어마다 `counter` 변수가 증가합니다.

매핑 클래스를 정의한 후 맵을 구현하는 인애플리케이션 스트림을 생성합니다.

```
// Split up the lines in pairs (2-tuples) containing: (word,1), and
// group by the tuple field "0" and sum up tuple field "1"
DataStream<Tuple2<String, Integer>> wordCountStream = input.flatMap(new
 Tokenizer()).keyBy(0).sum(1);
```

```
// Serialize the tuple to string format, and publish the output to kinesis sink
wordCountStream.map(tuple -> tuple.toString()).addSink(createSinkFromStaticConfig());
```

이 애플리케이션의 전체 코드는 단어 수 사용자 지정 지표 애플리케이션을 참조하세요.

사용자 지정 지표 보기

애플리케이션의 사용자 지정 지표는 AWS/KinesisAnalytics대시보드의 CloudWatch Metrics 콘솔의 애플리케이션 지표 그룹 아래에 표시됩니다.

Apache Flink용 Amazon 매니지드 서비스에서 CloudWatch 알람 사용하기

Amazon CloudWatch 지표 경보를 사용하면 지정한 기간 동안 CloudWatch 지표를 관찰할 수 있습니다. 이러한 경보는 여러 기간에 대해 지정된 임계값과 지표 또는 표현식의 값을 비교하여 하나 이상의 작업을 수행합니다. 작업의 한 예로는 Amazon Simple Notification Service(SNS) 주제에 알림을 보내는 것이 있습니다.

경보에 대한 자세한 내용은 Amazon CloudWatch CloudWatch 경보 사용을 참조하십시오.

권장되는 경보

이 섹션은 Managed Service for Apache Flink 애플리케이션을 모니터링할 때 권장되는 경보를 설명합니다.

이 표에는 권장 경보가 설명되어 있으며 다음과 같은 열이 있습니다.

- 지표 표현식: 임계값에 비춰 테스트할 지표나 지표 표현식입니다.
- 통계: 지표를 확인하는 데 사용되는 통계(예: 평균)입니다.
- 임계값: 이 경보를 사용하려면 예상 애플리케이션 성능의 한계를 정의하는 임계값을 결정해야 합니다. 이 임계값은 정상적인 조건에서 애플리케이션을 모니터링하여 결정해야 합니다.
- 설명: 이 경보를 트리거할 수 있는 원인과 해당 조건에 대한 가능한 해결 방법.

지표 표현식	통계	Threshold	설명
#### > 0	Average	0	A downtime greater than zero indicates that the application has failed. If the value is larger than 0, the

지표 표현식	통계	Threshold	설명
			application is not processing any data. Recommended for all applications. The ### ## metric measures the duration of an outage. A downtime greater than zero indicates that the application has failed. For troubleshooting, see 애플리케이션 다시 시작 중.

지표 표현식	통계	Threshold	설명
## (#####) numberOfFailed > 0	Average	0	This metric counts the number of failed checkpoints since the application started. Depending on the application, it can be tolerable if checkpoints fail occasionally. But if checkpoints are regularly failing, the application is likely unhealthy and needs further attention. We recommend monitoring RATE(numberOfFailedCheckpoints) to alarm on the gradient and not on absolute values. Recommended for all applications. Use this metric to monitor application health and checkpointing progress. The application saves state data to checkpoints when it's healthy. Checkpointing can fail due to timeouts if the application isn't making progress in processing the input

지표 표현식	통계	Threshold	설명
			data. For troublesh ooting, see 체크포인팅 시간이 초과되었습니다..
###. numRecords0utPerSecond < threshold	Average	The minimum number of records emitted from the applicati on during normal conditions.	Recommended for all applications. Falling below this threshold can indicate that the application isn't making expected progress on the input data. For troublesh ooting, see 처리량이 너무 느림.

지표 표현식	통계	Threshold	설명
`records_lag_max|millisbehindLatest` > threshold	Maximum	The maximum expected latency during normal conditions.	If the application is consuming from Kinesis or Kafka, these metrics indicate if the application is falling behind and needs to be scaled in order to keep up with the current load. This is a good generic metric that is easy to track for all kinds of applications. But it can only be used for reactive scaling, i.e., when the application has already fallen behind. Recommended for all applications. Use the `records_lag_max` metric for a Kafka source, or the `millisbehindLatest` for a Kinesis stream source. Rising above this threshold can indicate that the application isn't making expected progress on the input data. For troubleshooting, see 처리량이 너무 느림.

지표 표현식	통계	Threshold	설명
`lastCheckpointDuration` > threshold	Maximum	The maximum expected checkpoint duration during normal conditions.	Monitors how much data is stored in state and how long it takes to take a checkpoint. If checkpoints grow or take long, the application is continuously spending time on checkpointing and has less cycles for actual processing. At some points, checkpoints may grow too large or take so long that they fail. In addition to monitoring absolute values, customers should also considering monitoring the change rate with `## (lastCheckpointSize)` and `## (lastCheckpointDuration)` . If the `lastCheckpointDuration` continuously increases, rising above this threshold can indicate that the application isn't making expected progress on the input

지표 표현식	통계	Threshold	설명
			data, or that there are problems with application health such as backpressure. For troubleshooting, see 영구 지속적 성장.

지표 표현식	통계	Threshold	설명
`lastCheckpointSize` > threshold	Maximum	The maximum expected checkpoint size during normal conditions.	Monitors how much data is stored in state and how long it takes to take a checkpoint. If checkpoints grow or take long, the application is continuously spending time on checkpointing and has less cycles for actual processing. At some points, checkpoints may grow too large or take so long that they fail. In addition to monitoring absolute values, customers should also considering monitoring the change rate with `## (lastCheckpointSize)` and `## (lastCheckpointDuration)` . If the `lastCheckpointSize` continuously increases, rising above this threshold can indicate that the application is accumulating state data. If the state data

지표 표현식	통계	Threshold	설명
			becomes too large, the application can run out of memory when recovering from a checkpoint, or recovering from a checkpoint might take too long. For troublesh ooting, see 영구 지속적 성장.
`heapMemor yUtilization` > threshold	Maximum	This gives a good indication of the overall resource utilization of the application and can be used for proactive scaling unless the application is I/O bound. The maximum expected `heapMemor yUtilization` size during normal conditions, with a recommended value of 90 percent.	You can use this metric to monitor the maximum memory utilization of task managers across the application. If the application reaches this threshold, you need to provision more resources . You do this by enabling automatic scaling or increasin g the application parallelism. For more information about increasing resources, see 스케일링.

지표 표현식	통계	Threshold	설명
`cpuUtilization` > threshold	Maximum	This gives a good indication of the overall resource utilization of the application and can be used for proactive scaling unless the application is I/O bound. The maximum expected `cpuUtilization` size during normal conditions, with a recommended value of 80 percent.	You can use this metric to monitor the maximum CPU utilization of task managers across the application. If the application reaches this threshold, you need to provision more resources You do this by enabling automatic scaling or increasing the application parallelism. For more information about increasing resources, see 스케일링.
`threadsCount` > threshold	Maximum	The maximum expected `threadsCount` size during normal conditions.	You can use this metric to watch for thread leaks in task managers across the application. If this metric reaches this threshold, check your application code for threads being created without being closed.

지표 표현식	통계	Threshold	설명
(oldGarbageCollection##* 100) /1### 60_000') > threshold	Maximum	The maximum expected oldGarbageCollection### ## duration. We recommend setting a threshold such that typical garbage collection time is 60 percent of the specified threshold, but the correct threshold for your application will vary.	If this metric is continually increasing, this can indicate that there is a memory leak in task managers across the application.
## (oldGarbageCollection###) > threshold	Maximum	The maximum expected oldGarbageCollection ## under normal conditions. The correct threshold for your application will vary.	If this metric is continually increasing, this can indicate that there is a memory leak in task managers across the application.
###. currentOutputWatermark - ###. currentInputWatermark > threshold	Minimum	The minimum expected watermark increment under normal conditions. The correct threshold for your application will vary.	If this metric is continually increasing, this can indicate that either the application is processing increasingly older events, or that an upstream subtask has not sent a watermark in an increasingly long time.

CloudWatch 로그에 사용자 지정 메시지 작성

Apache Flink용 관리 서비스 애플리케이션의 로그에 사용자 지정 메시지를 작성할 수 있습니다. CloudWatch Apache log4j 라이브러리 또는 Simple Logging Facade for Java (SLF4J) 라이브러리를 사용하여 이 작업을 수행할 수 있습니다.

주제

- Log4J를 사용하여 CloudWatch 로그에 쓰기
- [SLF4J] 를 사용하여 CloudWatch 로그에 쓰기

Log4J를 사용하여 CloudWatch 로그에 쓰기

1. 애플리케이션의 pom.xml 파일에서 다음 종속 항목을 추가합니다.

```
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-api</artifactId>
    <version>2.6.1</version>
</dependency>
<dependency>
    <groupId>org.apache.logging.log4j</groupId>
    <artifactId>log4j-core</artifactId>
    <version>2.6.1</version>
</dependency>
```

2. 라이브러리의 객체를 포함시키세요.

```
import org.apache.logging.log4j.Logger;
```

3. 애플리케이션 클래스를 전달하여 Logger 객체를 인스턴스화합니다.

```
private static final Logger log =
 LogManager.getLogger.getLogger(YourApplicationClass.class);
```

4. log.info를 사용하여 로그에 기록합니다. 많은 수의 메시지가 애플리케이션 로그에 기록됩니다. 사용자 지정 메시지를 더 쉽게 필터링하려면 INFO 애플리케이션 로그 수준을 사용하세요.

```
log.info("This message will be written to the application's CloudWatch log");
```

애플리케이션은 다음과 유사한 메시지와 함께 레코드를 로그에 기록합니다.

```
{
  "locationInformation": "com.amazonaws.services.managed-
flink.StreamingJob.main(StreamingJob.java:95)",
  "logger": "com.amazonaws.services.managed-flink.StreamingJob",
  "message": "This message will be written to the application's CloudWatch log",
  "threadName": "Flink-DispatcherRestEndpoint-thread-2",
  "applicationARN": "arn:aws:kinesisanalyticsus-east-1:123456789012:application/test",
  "applicationVersionId": "1", "messageSchemaVersion": "1",
  "messageType": "INFO"
}
```

[SLF4J] 를 사용하여 CloudWatch 로그에 쓰기

1. 애플리케이션의 pom.xml 파일에서 다음 종속 항목을 추가합니다.

   ```
   <dependency>
       <groupId>org.slf4j</groupId>
       <artifactId>slf4j-log4j12</artifactId>
       <version>1.7.7</version>
       <scope>runtime</scope>
   </dependency>
   ```

2. 라이브러리의 객체를 포함시키세요.

   ```
   import org.slf4j.Logger;
   import org.slf4j.LoggerFactory;
   ```

3. 애플리케이션 클래스를 전달하여 Logger 객체를 인스턴스화합니다.

   ```
   private static final Logger log =
    LoggerFactory.getLogger(YourApplicationClass.class);
   ```

4. log.info를 사용하여 로그에 기록합니다. 많은 수의 메시지가 애플리케이션 로그에 기록됩니다. 사용자 지정 메시지를 더 쉽게 필터링하려면 INFO 애플리케이션 로그 수준을 사용하세요.

   ```
   log.info("This message will be written to the application's CloudWatch log");
   ```

애플리케이션은 다음과 유사한 메시지와 함께 레코드를 로그에 기록합니다.

```
{
  "locationInformation": "com.amazonaws.services.managed-
flink.StreamingJob.main(StreamingJob.java:95)",
  "logger": "com.amazonaws.services.managed-flink.StreamingJob",
  "message": "This message will be written to the application's CloudWatch log",
  "threadName": "Flink-DispatcherRestEndpoint-thread-2",
  "applicationARN": "arn:aws:kinesisanalyticsus-east-1:123456789012:application/test",
  "applicationVersionId": "1", "messageSchemaVersion": "1",
  "messageType": "INFO"
}
```

AWS CloudTrail을 사용한 Managed Service for Apache Flink API 호출 로깅

Apache Flink용 관리 서비스는 Apache Flink용 관리 서비스에서 사용자, 역할 또는 AWS 서비스가 수행한 작업의 기록을 제공하는 서비스와 통합되어 있습니다. AWS CloudTrail CloudTrail Apache Flink용 관리 서비스에 대한 모든 API 호출을 이벤트로 캡처합니다. 캡처된 호출에는 Managed Service for Apache Flink 콘솔에서 수행한 호출과 Managed Service for Apache Flink API 작업에 대한 코드 호출이 포함됩니다. 트레일을 생성하면 Apache Flink용 관리 서비스에 대한 CloudTrail 이벤트를 포함하여 Amazon S3 버킷으로 이벤트를 지속적으로 전송할 수 있습니다. 트레일을 구성하지 않아도 CloudTrail 콘솔의 이벤트 기록에서 가장 최근 이벤트를 계속 볼 수 있습니다. 에서 수집한 CloudTrail 정보를 사용하여 Apache Flink용 관리형 서비스에 이루어진 요청, 요청이 이루어진 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

자세한 CloudTrail 내용은 AWS CloudTrail사용 설명서를 참조하십시오.

Apache Flink용 관리형 서비스 정보: CloudTrail

CloudTrail 계정을 만들면 AWS 계정에서 활성화됩니다. Apache Flink용 관리 서비스에서 활동이 발생하면 해당 활동이 CloudTrail 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 이벤트 기록으로 CloudTrail 이벤트 보기를 참조하십시오.

Managed Service for Apache Flink의 이벤트를 포함한 AWS 계정에서의 이벤트를 지속적으로 기록하려면 추적을 생성합니다. 트레일을 사용하면 CloudTrail Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 지역에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 지역의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다.

또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- 추적 생성 개요
- CloudTrail 지원되는 서비스 및 통합
- 에 대한 Amazon SNS 알림 구성 CloudTrail
- 여러 지역에서 CloudTrail 로그 파일 수신 및 여러 계정으로부터 CloudTrail 로그 파일 수신

모든 Apache Flink용 관리 서비스 작업은 Apache Flink용 관리 서비스 API 참조에 의해 CloudTrail 기록되고 문서화됩니다. 예를 들어, `CreateApplication` 및 `UpdateApplication` 작업에 대한 호출은 로그 파일에 항목을 생성합니다. CloudTrail

모든 이벤트 및 로그 항목에는 요청을 생성한 사용자에 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 AWS Identity and Access Management(IAM) 사용자 보안 인증 정보로 했는지
- 역할 또는 페더레이션 사용자에 대한 임시 보안 인증 정보를 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

자세한 내용은 CloudTrail userIdentity 요소를 참조하세요.

Managed Service for Apache Flink 로그 파일 항목 업데이트

트레일은 지정한 Amazon S3 버킷에 이벤트를 로그 파일로 전송할 수 있는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함되어 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요청된 작업, 작업 날짜 및 시간, 요청 매개 변수 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 공개 API 호출의 정렬된 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음 예제는 AddApplicationCloudWatchLoggingOption및 DescribeApplication작업을 보여주는 CloudTrail 로그 항목을 보여줍니다.

```
{
    "Records": [
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
```

```
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2019-03-07T01:19:47Z",
            "eventSource": "kinesisanlaytics.amazonaws.com",
            "eventName": "AddApplicationCloudWatchLoggingOption",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "applicationName": "cloudtrail-test",
                "currentApplicationVersionId": 1,
                "cloudWatchLoggingOption": {
                    "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:flink-cloudwatch"
                }
            },
            "responseElements": {
                "cloudWatchLoggingOptionDescriptions": [
                    {
                        "cloudWatchLoggingOptionId": "2.1",
                        "logStreamARN": "arn:aws:logs:us-east-1:012345678910:log-
group:cloudtrail-test:log-stream:flink-cloudwatch"
                    }
                ],
                "applicationVersionId": 2,
                "applicationARN": "arn:aws:kinesisanalyticsus-
east-1:012345678910:application/cloudtrail-test"
            },
            "requestID": "18dfb315-4077-11e9-afd3-67f7af21e34f",
            "eventID": "d3c9e467-db1d-4cab-a628-c21258385124",
            "eventType": "AwsApiCall",
            "apiVersion": "2018-05-23",
            "recipientAccountId": "012345678910"
        },
        {
            "eventVersion": "1.05",
            "userIdentity": {
                "type": "IAMUser",
                "principalId": "EX_PRINCIPAL_ID",
                "arn": "arn:aws:iam::012345678910:user/Alice",
```

```
                "accountId": "012345678910",
                "accessKeyId": "EXAMPLE_KEY_ID",
                "userName": "Alice"
            },
            "eventTime": "2019-03-12T02:40:48Z",
            "eventSource": "kinesisanlaytics.amazonaws.com",
            "eventName": "DescribeApplication",
            "awsRegion": "us-east-1",
            "sourceIPAddress": "127.0.0.1",
            "userAgent": "aws-sdk-java/unknown-version Linux/x.xx",
            "requestParameters": {
                "applicationName": "sample-app"
            },
            "responseElements": null,
            "requestID": "3e82dc3e-4470-11e9-9d01-e789c4e9a3ca",
            "eventID": "90ffe8e4-9e47-48c9-84e1-4f2d427d98a5",
            "eventType": "AwsApiCall",
            "apiVersion": "2018-05-23",
            "recipientAccountId": "012345678910"
        }
    ]
}
```

Amazon Managed Service for Apache Flink의 튜닝 성능

이 항목에서는 Managed Service for Apache Flink 애플리케이션의 성능을 모니터링하고 개선하는 기술에 대해 설명합니다.

주제

성능 문제 해결

이 섹션에는 성능 문제를 진단하고 해결하기 위해 확인할 수 있는 증상 목록이 포함되어 있습니다.

데이터 소스가 Kinesis 스트림인 경우 성능 문제는 일반적으로 `millisbehindLatest` 지표가 높거나 증가하는 것으로 나타납니다. 다른 소스의 경우 소스의 읽기 지연을 나타내는 유사한 지표를 확인할 수 있습니다.

데이터 경로

애플리케이션의 성능 문제를 조사할 때는 데이터가 취하는 전체 경로를 고려하세요. 다음과 같은 애플리케이션 구성 요소는 제대로 설계되거나 프로비저닝되지 않을 경우 성능 병목 현상이 발생하고 역압을 초래할 수 있습니다.

- 데이터 소스 및 대상: 애플리케이션과 상호 작용하는 외부 리소스가 애플리케이션이 경험할 처리량에 맞게 속성이 프로비저닝되었는지 확인하세요.
- 상태 데이터: 애플리케이션이 상태 저장소와 너무 자주 상호 작용하지 않도록 하세요.

  애플리케이션이 사용하는 시리얼라이저를 최적화할 수 있습니다. 기본 Kryo 시리얼라이저는 모든 직렬화 가능 유형을 처리할 수 있지만 애플리케이션이 데이터를 POJO 유형으로만 저장하는 경우 더 성능이 좋은 시리얼라이저를 사용할 수 있습니다. Apache Flink 시리얼라이저에 대한 자세한 내용은 Apache Flink 설명서의 데이터 유형 및 직렬화를 참조하세요.
- 오퍼레이터: 오퍼레이터가 구현하는 비즈니스 로직이 너무 복잡하지 않은지, 모든 레코드가 처리될 때마다 리소스를 생성하거나 사용하지 않는지 확인하세요. 또한 애플리케이션에서 슬라이딩 윈도우나 텀블링 윈도우가 너무 자주 생성되지 않도록 하세요.

성능 문제 해결 솔루션

이 섹션에는 성능 문제에 대한 잠재적 해결책이 수록되어 있습니다.

주제

CloudWatch 모니터링 수준

CloudWatch 모니터링 수준이 너무 과도하게 설정되어 있지 않은지 확인하세요.

Debug 모니터링 로그 수준 설정은 대량의 트래픽을 생성하여 역압을 유발할 수 있습니다. 애플리케이션 관련 문제를 적극적으로 조사할 때만 사용해야 합니다.

애플리케이션의 Parallelism 설정이 높은 경우 Parallelism Monitoring Metrics Level을 사용하면 마찬가지로 많은 양의 트래픽이 발생하여 역압으로 이어질 수 있습니다. 애플리케이션의 Parallelism이(가) 낮거나 애플리케이션 관련 문제를 조사할 때만 이 지표 수준을 사용하세요.

자세한 설명은 애플리케이션 모니터링 수준 섹션을 참조하세요.

애플리케이션 CPU 지표

애플리케이션의 CPU 지표를 확인하세요. 이 지표가 75%를 초과하는 경우 Auto Scaling을 활성화하여 애플리케이션이 자체적으로 더 많은 리소스를 할당하도록 허용할 수 있습니다.

Auto Scaling이 활성화된 경우 15분 동안 CPU 사용량이 75% 를 초과하면 애플리케이션에서 더 많은 리소스를 할당합니다. 스케일링에 대한 자세한 내용은 다음 적절한 크기 조정 섹션이나 스케일링을(를) 참조하세요.

> **Note**
>
> 애플리케이션은 CPU 사용량에 따라서만 자동으로 규모가 조정됩니다. 애플리케이션은 `heapMemoryUtilization`와(과) 같은 다른 시스템 지표에 따라 Auto Scaling되지 않습니다. 애플리케이션의 다른 지표 사용량이 높은 경우 애플리케이션의 병렬성을 수동으로 높이세요.

애플리케이션 병렬성

애플리케이션의 병렬성을 높이세요. UpdateApplication 작업의 `ParallelismConfigurationUpdate` 파라미터를 사용하여 애플리케이션의 병렬 처리를 업데이트합니다.

애플리케이션의 최대 KPU는 기본적으로 64이며, 한도 증가를 요청하여 늘릴 수 있습니다.

또한 애플리케이션 병렬성만 높이는 것보다는 워크로드를 기반으로 각 운영자에게 병렬성을 할당하는 것도 중요합니다. 다음을 연산자 병렬성을(를) 참조하세요.

애플리케이션 로깅

애플리케이션이 처리 중인 모든 레코드에 대한 항목을 로깅하고 있는지 확인하세요. 애플리케이션 처리량이 높은 시간에 각 레코드에 대한 로그 항목을 작성하면 데이터 처리에 심각한 병목 현상이 발생할 수 있습니다. 이 상태를 확인하려면 애플리케이션이 처리하는 모든 레코드와 함께 기록하는 로그 항목이 있는지 로그를 쿼리하세요. 애플리케이션 로그 읽기에 대한 자세한 내용은 the section called “로그 분석”을(를) 참조하세요.

연산자 병렬성

애플리케이션의 워크로드가 작업자 프로세스 간에 균등하게 분산되어 있는지 확인하세요.

애플리케이션 운영자의 워크로드 조정에 대한 자세한 내용은 연산자 스케일링을(를) 참조하세요.

애플리케이션 로직

애플리케이션 로직을 검사하여 외부 종속성 (예: 데이터베이스 또는 웹 서비스) 액세스, 애플리케이션 상태 액세스 등과 같은 비효율적이거나 성능이 떨어지는 작업이 있는지 확인합니다. 또한 외부 종속성은 성능이 좋지 않거나 안정적으로 액세스할 수 없는 경우 성능을 저해할 수 있으며, 이로 인해 외부 종속성이 `HTTP 500` 오류를 반환할 수 있습니다.

애플리케이션에서 외부 종속성을 사용하여 들어오는 데이터를 보강하거나 처리하는 경우에는 비동기 IO를 대신 사용하는 것이 좋습니다. 자세한 내용을 알아보려면 Apache Flink 설명서의 Async I/O를 참조하세요.

애플리케이션 메모리

애플리케이션에 리소스 누수가 있는지 확인하세요. 애플리케이션이 스레드나 메모리를 제대로 처리하지 않는 경우 `millisbehindLatest`, `CheckpointSize`, `CheckpointDuration`, 지표가 급증하거나 점차 증가할 수 있습니다. 이 경우 작업 관리자 또는 작업 관리자 오류가 발생할 수도 있습니다.

성능 모범 사례

이 섹션은 성능을 위한 애플리케이션을 설계할 때 특별히 고려해야 할 사항에 대해 설명합니다.

적절한 크기 조정

이 섹션은 애플리케이션 수준 및 운영자 수준 조정 관리에 대한 정보가 포함되어 있습니다.

이 섹션은 다음 주제를 포함합니다.

- 애플리케이션 스케일링을 적절하게 관리
- 연산자 스케일링을 적절하게 관리

애플리케이션 스케일링을 적절하게 관리

자동 크기 조정을 사용하여 애플리케이션 활동의 예상치 못한 급증을 처리할 수 있습니다. 다음 기준이 충족되면 애플리케이션의 KPU가 자동으로 증가합니다.

- 애플리케이션에 자동 크기 조정이 활성화되어 있습니다.
- 15분 동안 CPU 사용량이 75% 이상을 유지합니다.

자동 크기 조정이 활성화되어 있지만 CPU 사용량이 이 임계값을 유지하지 않는 경우 애플리케이션은 KPU를 스케일 업하지 않습니다. CPU 사용량이 이 임계값을 충족하지 못하거나 `heapMemoryUtilization`와(과) 같은 다른 사용량 지표에서 급증하는 경우 애플리케이션이 활동 급증을 처리할 수 있도록 수동으로 스케일링을 늘리세요.

> **Note**
>
> 애플리케이션이 Auto Scaling을 통해 더 많은 리소스를 자동으로 추가한 경우 일정 기간 동안 사용하지 않으면 새 리소스를 릴리스합니다. 리소스 다운스케일링은 일시적으로 성능에 영향을 미칩니다.

스케일링에 대한 자세한 내용은 스케일링을(를) 참조하세요.

연산자 스케일링을 적절하게 관리

애플리케이션의 워크로드가 작업자 프로세스에 균등하게 분배되고 애플리케이션의 운영자가 안정적이고 성능을 유지하는 데 필요한 시스템 리소스를 보유하고 있는지 확인하여 애플리케이션의 성능을 개선할 수 있습니다.

`parallelism` 설정을 사용하여 애플리케이션 코드의 각 연산자에 대한 병렬성을 설정할 수 있습니다. 연산자의 병렬성을 설정하지 않으면 애플리케이션 수준의 병렬성 설정이 사용됩니다. 애플리케이션 수준의 병렬성 처리 설정을 사용하는 연산자는 애플리케이션에 사용할 수 있는 모든 시스템 리소스를 사용할 수 있으므로 애플리케이션이 불안정해질 수 있습니다.

각 연산자의 병렬성을 가장 잘 결정하려면 애플리케이션의 다른 연산자와 비교하여 상대적인 리소스 요구 사항을 고려하세요. 리소스를 많이 사용하는 연산자를 리소스 사용량이 적은 연산자보다 높은 연산자 병렬성 설정으로 설정하세요.

애플리케이션의 전체 연산자 병렬성은 애플리케이션의 모든 연산자에 대한 병렬성의 합계입니다. 애플리케이션에 사용할 수 있는 전체 작업 슬롯과 가장 적합한 비율을 결정하여 애플리케이션의 전체 연산자 병렬성을 조정합니다. 작업 슬롯에 대한 전체 연산자 병렬성의 일반적인 안정적인 비율은 4:1 입니다. 즉, 애플리케이션에는 사용 가능한 연산자 하위 작업 4개당 작업 슬롯 1개가 있습니다. 리소스를 많이 사용하는 연산자가 있는 애플리케이션에는 3:1 또는 2:1 의 비율이 필요할 수 있지만, 리소스를 덜 사용하는 애플리케이션에서는 비율이 10:1 이면 안정적일 수 있습니다.

런타임 속성을(를) 사용하여 연산자의 비율을 설정할 수 있으므로 애플리케이션 코드를 컴파일하고 업로드하지 않고도 연산자의 병렬성을 조정할 수 있습니다.

다음 코드 예제는 연산자 병렬성을 현재 애플리케이션 병렬성의 조정 가능한 비율로 설정하는 방법을 보여줍니다.

```
Map<String, Properties> applicationProperties =
 KinesisAnalyticsRuntime.getApplicationProperties();
```

```
operatorParallelism =
    StreamExecutionEnvironment.getParallelism() /
    Integer.getInteger(

 applicationProperties.get("OperatorProperties").getProperty("MyOperatorParallelismRatio")
    );
```

하위 작업, 작업 슬롯 및 기타 애플리케이션 리소스에 대한 자세한 내용은 애플리케이션 리소스을(를) 참조하세요.

애플리케이션의 작업자 프로세스 전반에 걸친 워크로드 분배를 제어하려면 `Parallelism` 설정과 `KeyBy` 파티션 방법을 사용하세요. 자세한 내용은 Apache Flink 설명서에서 다음 주제를 참조하세요.

- 병렬 실행
- 데이터스트림 변환

외부 종속성 리소스 사용량 모니터링

대상(예: Kinesis Streams, Kinesis Data Firehose, DynamoDB 또는 OpenSearch Service) 에 성능 병목 현상이 있는 경우 애플리케이션에 역압이 발생합니다. 외부 종속성이 애플리케이션 처리량에 맞게 적절하게 프로비저닝되었는지 확인하세요.

> **Note**
>
> 다른 서비스에 장애가 발생하면 애플리케이션에 장애가 발생할 수 있습니다. 애플리케이션에서 장애가 발생하는 경우 대상 서비스의 CloudWatch 로그에서 장애가 있는지 확인하세요.

Apache Flink 애플리케이션을 로컬에서 실행

메모리 문제를 해결하려면 로컬 Flink 설치에서 애플리케이션을 실행할 수 있습니다. 이렇게 하면 Managed Service for Apache Flink에서 애플리케이션을 실행할 때 사용할 수 없는 스택 트레이스 및 힙 덤프와 같은 디버깅 도구에 액세스할 수 있습니다.

로컬 Flink 설치를 만드는 방법에 대한 자세한 내용은 Apache Flink 설명서의 로컬 설치 자습서를 참조하세요.

성능 모니터링

이 섹션에서는 애플리케이션 성능을 모니터링하는 도구에 대해 설명합니다.

CloudWatch 지표를 사용한 성능 모니터링

CloudWatch 지표를 사용하여 애플리케이션의 리소스 사용량, 처리량, 체크포인트, 다운타임을 모니터링합니다. Managed Service for Apache Flink 애플리케이션에서 CloudWatch 지표를 사용하는 방법은 Managed Service for Apache Flink의 지표 및 차원을(를) 참조하세요.

CloudWatch 로그 및 경보를 사용한 성능 모니터링

CloudWatch Logs를 사용하여 잠재적으로 성능 문제를 일으킬 수 있는 오류 상태를 모니터링합니다.

Apache Flink 작업 상태가 `RUNNING` 상태에서 `FAILED` 상태로 변경될 때 로그 항목에 오류 상태가 나타납니다.

CloudWatch 경보를 사용하여 안전한 임계값을 초과하는 리소스 사용 또는 체크포인트 지표, 예상치 못한 애플리케이션 상태 변경 등과 같은 성능 문제에 대한 알림을 생성할 수 있습니다.

Apache Flink용 관리형 서비스 애플리케이션을 위한 CloudWatch 경보를 생성하는 방법에 대한 자세한 내용은 경보을(를) 참조하세요.

Managed Service for Apache Flink 및 Studio 노트북 할당량

Amazon Managed Service for Apache Flink로 작업할 때는 다음 할당량을 참고하세요.

- 계정에서 지역당 최대 50개의 Managed Service for Apache Flink 애플리케이션을 생성할 수 있습니다. 서비스 할당량 상향 양식을 통해 추가 애플리케이션 요청을 할 수 있습니다. 자세한 설명은 AWS Support 센터 섹션을 참조하세요.

  Managed Service for Apache Flink를 지원하는 지역 목록은 Managed Service for Apache Flink 지역 및 엔드포인트를 참조하세요.

- Kinesis 처리 단위(KPU)의 수는 기본적으로 64로 제한됩니다. 이 할당량 증가를 요청하는 방법에 대한 지침은 Service Quotas의 할당량 증가를 요청하려면을 참조하세요. 새 KPU 한도를 적용해야 하는 애플리케이션 접두사를 지정했는지 확인하세요.

  Managed Service for Apache Flink를 사용하면 애플리케이션에서 사용하는 리소스 대신 할당된 리소스에 대한 요금이 AWS 계정에 청구됩니다. 스트림 처리 애플리케이션을 실행하는 데 사용되는 KPU의 최대 개수에 따라 시간당 비용이 청구됩니다. 단일 KPU는 1개의 vCPU 및 4GiB의 메모리를 제공합니다. 각 KPU에서 이 서비스는 50GiB의 실행 중인 애플리케이션 스토리지도 프로비저닝합니다.

- Managed Service for Apache Flink 스냅샷는 애플리케이션당 최대 1,000개까지 생성할 수 있습니다.

- 애플리케이션당 최대 50개의 태그를 지정할 수 있습니다.

- 애플리케이션 JAR 파일의 최대 크기는 512MiB입니다. 이 할당량을 초과하면 애플리케이션이 시작되지 않습니다.

Studio 노트북에서 다음 할당량이 적용됩니다. 할당량을 더 높여 달라고 요청하려면 지원 사례 생성합니다.

- `websocketMessageSize` = 5MiB
- `noteSize` = 5MiB
- `noteCount` = 1,000
- `Max cumulative UDF size` = 100MiB
- `Max cumulative dependency jar size` = 300MiB

Managed Service for Apache Flink 정비

Managed Service for Apache Flink는 작동 시스템 및 컨테이너 이미지 보안 업데이트로 애플리케이션을 정기적으로 패치하여 규정 준수를 유지하고 AWS 보안 목표를 충족합니다. 다음 표는 Managed Service for Apache Flink가 이러한 유형의 정비를 수행하는 기본 기간을 나열합니다. 애플리케이션에 대한 정비는 해당 지역에 해당하는 기간 중 언제든지 이루어질 수 있습니다. 이 정비 프로세스 중에 애플리케이션 가동 중지가 10초~30초 발생할 수 있습니다. 하지만 실제 가동 중지 기간은 애플리케이션 상태에 따라 달라집니다. 이 가동 중지의 영향을 최소화하는 방법에 대한 자세한 설명은 the section called “내결함성: 체크포인트 및 세이브포인트” 섹션을 참조하세요.

Managed Service for Apache Flink가 애플리케이션에 대한 정비를 수행하는 기간을 변경하려면 UpdateApplicationMaintenanceConfiguration API를 사용하세요.

지역	정비 창
AWS GovCloud(미국 서부)	06:00~14:00 UTC
AWS GovCloud(미국 동부)	03:00~11:00 UTC
미국 동부(버지니아 북부)	03:00~11:00 UTC
미국 동부(오하이오)	03:00~11:00 UTC
미국 서부(캘리포니아 북부 지역)	06:00~14:00 UTC
미국 서부(오레곤)	06:00~14:00 UTC
아시아 태평양(홍콩)	13:00~21:00 UTC
아시아 태평양(뭄바이)	16:30~00:30 UTC
아시아 태평양(하이데라바드)	16:30~00:30 UTC
아시아 태평양(서울)	13:00~21:00 UTC
아시아 태평양(싱가포르)	14:00~22:00 UTC
아시아 태평양(시드니)	12:00~20:00 UTC
아시아 태평양(자카르타)	15:00~23:00 UTC

지역	정비 창
아시아 태평양(도쿄)	13:00~21:00 UTC
캐나다(중부)	03:00~11:00 UTC
중국(베이징)	13:00~21:00 UTC
중국(닝샤)	13:00~21:00 UTC
유럽(프랑크푸르트)	06:00~14:00 UTC
유럽(취리히)	20:00~04:00 UTC
유럽(아일랜드)	22:00~06:00 UTC
유럽(런던)	22:00~06:00 UTC
유럽(스톡홀름)	23:00~07:00 UTC
유럽(밀라노)	21:00–05:00 UTC
유럽(스페인)	21:00–05:00 UTC
아프리카(케이프타운)	20:00~04:00 UTC
유럽(아일랜드)	22:00~06:00 UTC
유럽(런던)	23:00~07:00 UTC
유럽(파리)	23:00~07:00 UTC
유럽(스톡홀름)	23:00~07:00 UTC
중동(바레인)	13:00~21:00 UTC
중동(UAE)	18:00~02:00 UTC
남아메리카(상파울루)	19:00~03:00 UTC
이스라엘(텔아비브)	20:00~04:00 UTC

모든 연산자에 대해 UUID 설정

Managed Service for Apache Flink가 스냅샷이 있는 애플리케이션에 대한 Flink 작업을 시작하는 경우 특정 문제로 인해 Flink 작업이 시작되지 않을 수 있습니다. 그 중 하나는 연산자 ID 불일치입니다. Flink는 Flink 작업 그래프 연산자에 대해 명시적이고 일관된 연산자 ID를 예상합니다. 명시적으로 설정하지 않으면 Flink는 연산자의 ID를 자동 생성합니다. 이는 Flink가 이러한 연산자 ID를 사용하여 작업 그래프에서 연산자를 고유하게 식별하고 이를 사용하여 각 연산자의 상태를 저장점에 저장하기 때문입니다.

연산자 ID 불일치 문제는 Flink가 작업 그래프의 연산자 ID와 저장점에 정의된 연산자 ID 간의 1:1 매핑을 찾지 못할 때 발생합니다. 이는 명시적으로 일관된 연산자 ID가 설정되지 않은 상태에서 Flink가 연산자 ID를 자동으로 생성하는 경우 발생하며, 이 연산자 ID는 모든 작업 그래프 생성과 일치하지 않을 수 있습니다. 유지보수 실행 중에는 애플리케이션에서 이 문제가 발생할 가능성이 높습니다. 이를 방지하려면 고객이 모든 연산자의 UUID를 Flink 코드로 설정하는 것이 좋습니다. 자세한 정보는 프로덕션 준비 주제의 모든 연산자의 UUID 설정 내용을 참조하세요.

프로덕션 준비 상태

Managed Service for Apache Flink에서 프로덕션 애플리케이션을 실행하는 데 있어 중요한 측면을 모아 놓은 것입니다. 전체 목록은 아니지만 애플리케이션을 프로덕션에 적용하기 전에 주의해야 할 최소한의 사항만 나열한 것입니다.

부하 테스트 애플리케이션

애플리케이션 관련 일부 문제는 부하가 심한 경우에만 나타납니다. 애플리케이션이 정상처럼 보이고 작동 이벤트로 인해 애플리케이션의 부하가 크게 증폭되는 사례가 있었습니다. 이는 애플리케이션 자체와 완전히 독립적으로 발생할 수 있습니다. 데이터 소스 또는 데이터 싱크를 몇 시간 동안 사용할 수 없으면 Flink 애플리케이션을 진행할 수 없습니다. 이 문제가 해결되면 처리되지 않은 데이터가 쌓여 가용 리소스가 완전히 고갈될 수 있습니다. 그러면 부하로 인해 이전에 발생하지 않았던 버그나 성능 문제가 증폭될 수 있습니다.

따라서 프로덕션 애플리케이션을 위한 적절한 부하 테스트를 실행하는 것이 중요합니다. 이러한 부하 테스트 중에 답변해야 하는 질문은 다음과 같습니다:

- 지속적인 고부하 상태에서도 애플리케이션이 안정적입니까?
- 최대 부하 상태에서도 애플리케이션이 저장점을 가져갈 수 있나요?
- 1시간의 백로그를 처리하는 데 얼마나 걸리나요? 그리고 24시간은 얼마나 걸리나요(스트림에 있는 데이터의 최대 보존 기간에 따라 다름)?
- 애플리케이션을 확장하면 애플리케이션 처리량이 증가하나요?

데이터 스트림에서 소비하는 경우 일정 시간 동안 스트림으로 생성하여 이러한 시나리오를 시뮬레이션할 수 있습니다. 그런 다음 애플리케이션을 시작하고 처음부터 데이터를 소비하도록 합니다. 예를 들어 Kinesis Data Stream의 경우 `TRIM_HORIZON`의 시작 위치를 사용합니다.

최대 병렬 처리

최대 병렬 처리는 상태 기반 애플리케이션이 스케일링할 수 있는 최대 병렬 처리를 정의합니다. 이는 상태가 처음 생성될 때 정의되며 상태를 삭제하지 않고 이 최대값 이상으로 연산자를 확장할 수 있는 방법은 없습니다.

최대 병렬 처리는 상태가 처음 생성될 때 설정됩니다.

기본적인 최대 병렬 처리 설정은 다음과 같습니다:

- 병렬 처리가 128 미만인 경우 128
- `MIN(nextPowerOfTwo(parallelism + (parallelism / 2)), 2^15)`: 병렬 처리가 128을 초과하는 경우

애플리케이션을 128 병렬 처리 이상으로 확장하려는 경우 최대 병렬 처리를 명시적으로 정의해야 합니다.

최대 병렬 처리는 `env.setMaxParallelism(x)` 연산자를 사용하거나 단일 연산자를 사용하여 애플리케이션 수준에서 정의할 수 있습니다. 다르게 지정하지 않는 한 모든 연산자는 애플리케이션의 최대 병렬 처리를 상속합니다.

자세한 설명은 Flink 설명서의 명시적 최대 병렬 처리 설정을 참조하세요.

모든 연산자에 대해 UUID 설정

UUID는 Flink가 저장점을 개별 연산자에 다시 매핑하는 작업에 사용됩니다. 각 연산자에 대해 특정 UUID를 설정하면 복원할 저장점 프로세스를 안정적으로 매핑할 수 있습니다.

```
.map(...).uid("my-map-function")
```

자세한 설명은 프로덕션 준비 체크리스트를 참조하세요.

Managed Service for Apache Flink 모범 사례

이 섹션에는 안정적이고 성능이 뛰어난 Managed Service for Apache Flink 애플리케이션을 개발하기 위한 정보와 권장 사항이 포함되어 있습니다.

주제

내결함성: 체크포인트 및 세이브포인트

체크포인트와 세이브포인트를 사용하여 Managed Service for Apache Flink 애플리케이션에서 내결함성을 구현하세요. 애플리케이션을 개발하고 유지 관리할 때는 다음 사항에 유의하세요.

- 애플리케이션에 체크포인팅을 활성화시켜 두는 것이 좋습니다. 체크포인트는 예정된 유지 관리 동안뿐만 아니라 서비스 문제, 애플리케이션 종속성 장애 및 기타 문제로 인해 예상치 못한 장애가 발생하는 경우에도 애플리케이션에 내결함성을 제공합니다. 유지 관리에 대한 자세한 내용은 정비 섹션을 참조하세요.
- 애플리케이션 개발 또는 문제 해결 `false` 중에 ApplicationSnapshotConfiguration: SnapshotsEnabled :를 로 설정합니다. 애플리케이션이 중지될 때마다 스냅샷이 생성되므로 애플리케이션이 비정상 상태이거나 성능이 좋지 않을 경우 문제가 발생할 수 있습니다. 애플리케이션이 프로덕션 단계에 들어가고 안정된 이후에 `SnapshotsEnabled`을 `true`으로 설정합니다.

> **Note**
>
> 애플리케이션이 올바른 상태 데이터로 제대로 다시 시작하려면 하루에 여러 번 스냅샷을 생성하는 것이 좋습니다. 스냅샷의 올바른 주기는 애플리케이션의 비즈니스 로직에 따라 다릅니다. 스냅샷을 자주 생성하면 최신 데이터를 복구할 수 있지만 비용이 증가하고 더 많은 시스템 리소스가 필요합니다.

애플리케이션 다운타임 모니터링에 대한 자세한 내용은 Managed Service for Apache Flink의 지표 및 차원 섹션을 참조하세요.

내결함성에 대한 자세한 내용은 내결함성 섹션을 확인하세요.

지원되지 않는 커넥터 버전

Managed Service for Apache Flink 버전 1.15는 지원되지 않는 Kinesis Connector 버전(애플리케이션 JAR에 번들로 제공)을 사용하는 경우 애플리케이션이 시작되거나 업데이트되지 않도록 자동으로 차단합니다. Managed Service for Apache Flink 버전 1.15로 업그레이드할 때는 최신 Kinesis 커넥터를 사용하고 있는지 확인하세요. 이 버전은 버전 1.15.2와 같거나 더 최근에 나온 버전입니다. 다른 모든 버전은 Apache Flink용 Managed Service for Apache Flink에서 지원되지 않습니다. 이 경우 세이브포인트로 중지 기능에 일관성 문제가 발생하거나 오류가 발생하여 클린 중지/업데이트 작업이 불가능할 수 있기 때문입니다.

성능 및 병렬 처리

애플리케이션 병렬 처리를 조정하고 성능 저하를 방지함으로써 애플리케이션을 모든 처리량 수준에 맞게 확장할 수 있습니다. 애플리케이션을 개발하고 유지 관리할 때는 다음 사항에 유의하세요.

- 모든 애플리케이션 소스 및 싱크가 충분히 프로비저닝되고 병목 현상이 발생하지 않는지 확인하세요. 소스와 싱크가 다른 AWS 서비스인 경우 를 사용하여 CloudWatch해당 서비스를 모니터링하세요.
- 병렬 처리가 매우 높은 애플리케이션의 경우 애플리케이션의 모든 연산자에게 높은 수준의 병렬 처리가 적용되는지 확인하세요. 기본적으로 Apache Flink는 애플리케이션 그래프의 모든 연산자에 대해 동일한 애플리케이션 병렬 처리를 적용합니다. 이로 인해 소스 또는 싱크의 프로비저닝 문제가 발생하거나 연산자 데이터 처리에 병목 현상이 발생할 수 있습니다. SetParallelism을 사용하여 코드에서 각 연산자의 병렬 처리를 변경할 수 있습니다.

- 애플리케이션의 연산자에 대한 병렬 처리 설정의 의미를 이해하세요. 연산자의 병렬 처리를 변경하면 연산자의 병렬 처리가 현재 설정과 호환되지 않을 때 생성된 스냅샷에서 애플리케이션을 복원하지 못할 수 있습니다. 연산자 병렬 처리 설정에 대한 자세한 내용은 연산자의 최대 병렬 처리를 명시적으로 설정하기를 참조하세요.

단순 조정에 대한 자세한 내용은 스케일링 섹션을 참조하세요.

연산자별 병렬 처리 설정

기본적으로 모든 연산자는 애플리케이션 수준으로 설정된 병렬 처리를 갖습니다. 를 사용하여 API를 사용하여 단일 연산자의 병렬 처리를 재정의할 수 있습니다. DataStream `.setParallelism(x)` 연산자 병렬 처리를 애플리케이션 병렬 처리 수와 같거나 낮은 모든 병렬 처리로 설정할 수 있습니다.

가능하면 연산자 병렬 처리를 애플리케이션 병렬 처리의 함수로 정의하세요. 이렇게 하면 애플리케이션 병렬 처리에 따라 연산자 병렬 처리가 달라집니다. 예를 들어 자동 크기 조정을 사용하는 경우 모든 연산자의 병렬 처리가 같은 비율로 달라집니다.

```
int appParallelism = env.getParallelism();
...
...ops.setParalleism(appParallelism/2);
```

경우에 따라 연산자 병렬 처리를 상수로 설정해야 할 수 있습니다. 예를 들어 Kinesis Stream 소스의 병렬 처리를 샤드 수로 설정합니다. 이러한 경우, 예를 들어 소스 스트림을 리샤딩해야 하는 경우, 코드를 변경하지 않고 변경하려면 연산자 병렬 처리를 애플리케이션 구성 파라미터로 전달하는 것을 고려해야 합니다.

로깅

로그를 사용하여 애플리케이션의 성능과 오류 상태를 모니터링할 수 있습니다. CloudWatch 애플리케이션에 대한 로깅을 구성할 때는 다음 사항에 유의하세요.

- 모든 런타임 문제를 CloudWatch 디버깅할 수 있도록 애플리케이션에 대한 로깅을 활성화하십시오.
- 애플리케이션에서 처리 중인 모든 레코드에 대해 로그 항목을 생성하지 마세요. 이로 인해 처리 중에 심각한 병목 현상이 발생하고 데이터 처리 시 배압이 발생할 수 있습니다.
- 애플리케이션이 제대로 실행되지 않을 때 알려주는 CloudWatch 경보를 생성하십시오. 자세한 정보는 경보을(를) 참조하세요.

에서 로깅을 구현하는 방법에 대한 자세한 내용은 로깅 및 모니터링 섹션을 참조하세요.

코딩

권장 프로그래밍 관행을 사용하여 애플리케이션의 성능과 안정성을 높일 수 있습니다. 애플리케이션 코드를 작성할 때 다음 사항을 유의하세요.

- 애플리케이션 코드, 애플리케이션의 `main` 메서드 또는 사용자 정의 함수에는 `system.exit()`를 사용하지 마세요. 코드 내에서 애플리케이션을 종료하려면 `Exception` 또는 `RuntimeException`에서 파생된 예외를 발생시키세요. 이 예외에는 애플리케이션에 어떤 문제가 발생했는지에 대한 메시지가 포함되어 있습니다.

  서비스에서 이 예외를 처리하는 방법에 대한 다음 내용을 참고하세요.

  - 애플리케이션의 `main` 메서드에서 예외가 발생하는 경우 애플리케이션이 `RUNNING` 상태로 전환될 때 서비스가 이를 `ProgramInvocationException`으로 래핑하고 작업 관리자가 작업을 제출하지 못합니다.
  - 사용자 정의 함수에서 예외가 발생한 경우 작업 관리자는 작업을 실패하고 작업을 다시 시작하며 예외 세부 정보가 예외 로그에 기록됩니다.
- 애플리케이션 JAR 파일과 포함된 종속성을 셰이딩하는 것을 고려해 보세요. 애플리케이션과 Apache Flink 런타임 간에 패키지 이름이 충돌할 가능성이 있는 경우에는 셰이딩을 사용하는 것이 좋습니다. 충돌이 발생하는 경우 애플리케이션 로그에 `java.util.concurrent.ExecutionException` 유형 예외가 포함될 수 있습니다. 애플리케이션 JAR 파일 셰이딩에 대한 자세한 내용은 Apache Maven Shade 플러그인을 참조하세요.

보안 인증 관리

장기 보안 인증을 프로덕션(또는 기타) 애플리케이션에 적용해서는 안 됩니다. 장기 보안 인증은 버전 관리 시스템에 체크인되어 쉽게 분실될 수 있습니다. 대신 Managed Service for Apache Flink 애플리케이션에 역할을 연결하고 해당 역할에 권한을 부여할 수 있습니다. 그러면 실행 중인 Flink 애플리케이션이 환경에서 해당 권한이 있는 임시 보안 인증을 가져올 수 있습니다. 인증을 위해 사용자 이름과 암호가 필요한 데이터베이스 등 기본적으로 IAM과 통합되지 않은 서비스에 대한 인증이 필요한 경우, AWS Secrets Manager에 암호를 저장하는 것을 고려해야 합니다.

많은 AWS 네이티브 서비스가 인증을 지원합니다.

- Kinesis Data Streams — .java ProcessTaxiStream

- 아마존 MSK — https://github.com/aws/ aws-msk-iam-auth using-the-amazon-msk /# - library-for-iam-authentication
- 아마존 엘라스틱서치 서비스 — .java AmazonElasticsearchSink
- Amazon S3 — Managed Service for Apache Flink에서 즉시 사용 가능

샤드/파티션이 거의 없는 소스에서 읽기

Apache Kafka 또는 Kinesis Data Stream에서 읽을 때 스트림의 병렬처리(예: Kafka의 파티션 수와 Kinesis의 샤드 수)와 애플리케이션의 병렬 처리 간에 불일치가 있을 수 있습니다. 네이티브 설계에서는 애플리케이션의 병렬 처리를 스트림의 병렬 처리 이상으로 확장할 수 없습니다. 소스 연산자의 각 하위 작업은 1개 이상의 샤드/파티션에서만 읽을 수 있습니다. 즉, 샤드가 2개뿐인 스트림과 병렬 처리 수가 8인 애플리케이션의 경우 스트림에서 실제로 소비되는 하위 작업은 두 개뿐이고 하위 작업 6개는 유휴 상태로 유지됩니다. 이로 인해 애플리케이션의 처리량이 크게 제한될 수 있습니다. 특히 역직렬화를 소스에서 수행하는 경우(기본값) 더욱 그렇습니다.

이러한 영향을 줄이려면 스트림을 확장하거나 둘 중 하나를 선택할 수 있습니다. 하지만 이것이 항상 바람직하거나 가능한 것은 아닙니다. 또는 소스를 재구성하여 직렬화를 수행하지 않고 `byte[]`를 그냥 전달하도록 할 수도 있습니다. 그런 다음 데이터를 리밸런싱하여 모든 작업에 균등하게 분배한 다음 데이터를 역직렬화할 수 있습니다. 이렇게 하면 역직렬화에 모든 하위 작업을 활용할 수 있으며 비용이 많이 들 수 있는 이 작업은 더 이상 스트림의 샤드/파티션 수에 얽매이지 않아도 됩니다.

스튜디오 노트북 새로 고침 간격

단락 결과 새로 고침 간격을 변경하는 경우 최소 1000밀리초 이상의 값으로 설정합니다.

스튜디오 노트북 최적 성능

다음 문으로 테스트한 결과, `events-per-second`에 `number-of-keys`를 곱한 값이 25,000,000 미만일 때 가장 좋은 성능을 보였습니다. 이는 15만 미만 `events-per-second`에 대한 것이었습니다.

```
SELECT key, sum(value) FROM key-values GROUP BY key
```

워터마크 전략과 유휴 샤드가 타임윈도우에 미치는 영향

Apache Kafka 및 Kinesis Data Streams에서 이벤트를 읽을 때 소스는 스트림의 속성을 기반으로 이벤트 시간을 설정할 수 있습니다. Kinesis의 경우 이벤트 시간은 대략적인 이벤트 도착 시간과 같습니다.

하지만 이벤트 소스에서 이벤트 시간을 설정하는 것만으로는 플링크 애플리케이션이 이벤트 시간을 사용하기에 충분하지 않습니다. 또한 소스는 소스에서 다른 모든 연산자에게 이벤트 시간에 대한 정보를 전파하는 워터마크를 생성해야 합니다. Flink 설명서에는 해당 프로세스의 작동 방식에 대한 좋은 개요가 나와 있습니다.

기본적으로 Kinesis에서 읽은 이벤트의 타임스탬프는 Kinesis에서 결정한 대략적인 도착 시간으로 설정됩니다. 애플리케이션에서 이벤트 시간이 제대로 작동하기 위한 추가 전제 조건은 워터마크 전략입니다.

```
WatermarkStrategy<String> s = WatermarkStrategy
    .<String>forMonotonousTimestamps()
    .withIdleness(Duration.ofSeconds(...));
```

그런 다음 `assignTimestampsAndWatermarks` 메서드를 사용하여 워터마크 전략을 `DataStream`에 적용합니다. 다음과 같은 몇 가지 유용한 기본 전략이 있습니다.

- `forMonotonousTimestamps()`는 이벤트 시간(대략적인 도착 시간)만 사용하고 주기적으로 최대값을 워터마크로 표시합니다(각 특정 하위 작업에 대해).
- `forBoundedOutOfOrderness(Duration.ofSeconds(...))`은 이전 전략과 비슷하지만 이벤트 시간인 워터마크 생성 기간을 사용합니다.

이 방법은 작동하지만 몇 가지 주의해야 할 사항이 있습니다. 워터마크는 하위 작업 수준에서 생성되며 오퍼레이터 그래프를 통해 흐릅니다.

Flink 설명서에서 발췌:

소스 함수의 각 병렬 하위 작업은 일반적으로 워터마크를 독립적으로 생성합니다. 이러한 워터마크는 특정 병렬 소스의 이벤트 시간을 정의합니다.

워터마크가 스트리밍 프로그램을 통해 흐르면서 워터마크가 도착하는 연산자에게도 이벤트 시간이 앞당겨집니다. 연산자가 이벤트 시간을 앞당길 때마다 후속 연산자를 위해 다운스트림에 새 워터마크가 생성됩니다.

예를 들어 유니온을 사용하는 연산자나 KeyBy(...) 또는 파티션(...) 함수 뒤에 오는 연산자 등 여러 입력 스트림을 소비하는 연산자도 있습니다. 이러한 연산자의 현재 이벤트 시간은 해당 입력 스트림의 이벤트 시간 중 최소값입니다. 입력 스트림이 이벤트 시간을 업데이트하면 연산자도 업데이트됩니다.

즉, 소스 하위 작업이 유휴 샤드에서 소비되는 경우, 다운스트림 연산자는 해당 하위 작업에서 새 워터마크를 받지 못하므로 시간 창을 사용하는 모든 다운스트림 연산자의 처리가 중단됩니다. 이를 방지하

기 위해 고객은 워터마크 전략에 withIdleness 옵션을 추가할 수 있습니다. 이 옵션을 사용하면 연산자의 이벤트 시간을 계산할 때 유휴 업스팀 하위 작업에서 워터마크가 제외됩니다. 따라서 유휴 하위 작업으로 인해 다운스트림 연산자의 이벤트 시간 단축이 더 이상 방해되지 않습니다.

하지만 워터마크 전략이 내장된 유휴 옵션을 사용하면 이벤트를 읽는 하위 작업이 없는 경우(예: 스트림에 이벤트가 없는 경우) 이벤트 시간을 앞당길 수 없습니다. 이는 스트림에서 한정된 이벤트 세트를 읽는 테스트 사례에서 특히 두드러집니다. 마지막 이벤트를 읽은 후 이벤트 시간이 늘어나지 않으므로 마지막 이벤트가 포함된 마지막 창은 절대 닫히지 않습니다.

요약

- 샤드가 유휴 상태인 경우 이 withIdleness 설정은 새 워터마크를 생성하지 않으며, 유휴 하위 작업이 보낸 마지막 워터마크는 다운스트림 연산자의 최소 워터마크 계산에서 제외됨
- 내장된 워터마크 전략을 사용하면 마지막으로 열린 창은 절대 닫히지 않습니다(워터마크를 앞당기는 새 이벤트가 전송되지 않는 한). 단, 새 창이 생성되고 그 이후에는 열린 상태로 유지됨
- Kinesis 스트림에서 시간을 설정하더라도 하나의 샤드가 다른 샤드보다 빨리 소비되는 경우(앱 초기화 중 또는 기존의 모든 샤드가 부모/자식 관계를 무시하고 병렬로 소비되는 TRIM_HORIZON을 사용할 때) 지연 이벤트가 계속 발생할 수 있음
- 워터마크 전략의 withIdleness 설정이 유휴 샤드(ConsumerConfigConstants.SHARD_IDLE_INTERVAL_MILLIS에 대한 키네시스 소스별 설정을 더 이상 지원하지 않는 듯함

예

다음 애플리케이션은 스트림에서 데이터를 읽고 이벤트 시간에 따라 세션 창을 생성합니다.

```
Properties consumerConfig = new Properties();
consumerConfig.put(AWSConfigConstants.AWS_REGION, "eu-west-1");
consumerConfig.put(ConsumerConfigConstants.STREAM_INITIAL_POSITION, "TRIM_HORIZON");

FlinkKinesisConsumer<String> consumer = new FlinkKinesisConsumer<>("...", new
 SimpleStringSchema(), consumerConfig);

WatermarkStrategy<String> s = WatermarkStrategy
    .<String>forMonotonousTimestamps()
    .withIdleness(Duration.ofSeconds(15));

env.addSource(consumer)
    .assignTimestampsAndWatermarks(s)
```

```
    .map(new MapFunction<String, Long>() {
        @Override
        public Long map(String s) throws Exception {
            return Long.parseLong(s);
        }
    })
    .keyBy(l -> 0l)
    .window(EventTimeSessionWindows.withGap(Time.seconds(10)))
    .process(new ProcessWindowFunction<Long, Object, Long, TimeWindow>() {
        @Override
        public void process(Long aLong, ProcessWindowFunction<Long, Object, Long,
 TimeWindow>.Context context, Iterable<Long>iterable, Collector<Object> collector)
 throws Exception {
            long count = StreamSupport.stream(iterable.spliterator(), false).count();
            long timestamp = context.currentWatermark();

            System.out.print("XXXXXXXXXXXXXX Window with " + count + " events");
            System.out.println("; Watermark: " + timestamp + ", " +
 Instant.ofEpochMilli(timestamp));


            for (Long l : iterable) {
                System.out.println(l);
            }
        }
    });
```

다음 예제에서는 8개의 이벤트가 16개의 샤드 스트림에 기록됩니다(처음 2개와 마지막 이벤트는 같은 샤드에 포함됨).

```
$ aws kinesis put-record --stream-name hp-16 --partition-key 1 --data MQ==
$ aws kinesis put-record --stream-name hp-16 --partition-key 2 --data Mg==
$ aws kinesis put-record --stream-name hp-16 --partition-key 3 --data Mw==
$ date

{
    "ShardId": "shardId-000000000012",
    "SequenceNumber": "49627894338614655560500811028721934184977530127978070210"
}
{
    "ShardId": "shardId-000000000012",
    "SequenceNumber": "49627894338614655560500811028795678659974022576354623682"
}
```

```
{
    "ShardId": "shardId-000000000014",
    "SequenceNumber": "49627894338659257050897872275134360684221592378842022114"
}
Wed Mar 23 11:19:57 CET 2022

$ sleep 10
$ aws kinesis put-record --stream-name hp-16 --partition-key 4 --data NA==
$ aws kinesis put-record --stream-name hp-16 --partition-key 5 --data NQ==
$ date

{
    "ShardId": "shardId-000000000010",
    "SequenceNumber": "49627894338570054070103749783042116732419934393936642210"
}
{
    "ShardId": "shardId-000000000014",
    "SequenceNumber": "49627894338659257050897872275659034489934342334017700066"
}
Wed Mar 23 11:20:10 CET 2022

$ sleep 10
$ aws kinesis put-record --stream-name hp-16 --partition-key 6 --data Ng==
$ date

{
    "ShardId": "shardId-000000000001",
    "SequenceNumber": "49627894338369347363316974173886988345467035365375213586"
}
Wed Mar 23 11:20:22 CET 2022

$ sleep 10
$ aws kinesis put-record --stream-name hp-16 --partition-key 7 --data Nw==
$ date

{
    "ShardId": "shardId-000000000008",
    "SequenceNumber": "49627894338525452579706688535878947299195189349725503618"
}
Wed Mar 23 11:20:34 CET 2022

$ sleep 60
$ aws kinesis put-record --stream-name hp-16 --partition-key 8 --data OA==
$ date
```

```
{
    "ShardId": "shardId-000000000012",
    "SequenceNumber": "49627894338614655560500811029600823255837371928900796610"
}
Wed Mar 23 11:21:27 CET 2022
```

이렇게 입력하면 5개의 세션 창(이벤트 1,2,3, 이벤트 4,5, 이벤트 6, 이벤트 7, 이벤트 8)이 생성될 것입니다. 하지만 프로그램은 처음 4개의 창만 출력합니다.

```
11:59:21,529 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 5 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000006,HashKeyRange: {StartingHashKey:
 127605887595351923798765477786913079296,EndingHashKey:
 148873535527910577765226390751398592511},SequenceNumberRange: {StartingSequenceNumber:
 49627894338480851089309627289524549239292625588395704418,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 5 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000006,HashKeyRange: {StartingHashKey:
 127605887595351923798765477786913079296,EndingHashKey:
 148873535527910577765226390751398592511},SequenceNumberRange: {StartingSequenceNumber:
 49627894338480851089309627289524549239292625588395704418,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,530 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 6 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000007,HashKeyRange: {StartingHashKey:
 148873535527910577765226390751398592512,EndingHashKey:
 170141183460469231731687303715884105727},SequenceNumberRange: {StartingSequenceNumber:
 49627894338503151834508157912666084957565273949901684850,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 6 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000010,HashKeyRange: {StartingHashKey:
 212676479325586539664609129644855132160,EndingHashKey:
 233944127258145193631070042609340645375},SequenceNumberRange: {StartingSequenceNumber:
 49627894338570054070103749782090692112383219034419626146,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,530 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 6 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000007,HashKeyRange: {StartingHashKey:
```

```
 148873535527910577765226390751398592512,EndingHashKey:
 170141183460469231731687303715884105727},SequenceNumberRange: {StartingSequenceNumber:
 496278943385031518345081579126660849575652739499016848 50,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,531 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 4 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000005,HashKeyRange: {StartingHashKey:
 106338239662793269832304564822427566080,EndingHashKey:
 127605887595351923798765477786913079295},SequenceNumberRange: {StartingSequenceNumber:
 496278943384585503441110966663830135210199772268897239 86,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 4 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000005,HashKeyRange: {StartingHashKey:
 106338239662793269832304564822427566080,EndingHashKey:
 127605887595351923798765477786913079295},SequenceNumberRange: {StartingSequenceNumber:
 496278943384585503441110966663830135210199772268897239 86,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 3 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000004,HashKeyRange: {StartingHashKey:
 850705917302346158658436518579420528 64,EndingHashKey:
 106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
 496278943384362495989125660432414778027473288653837435 54,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 2 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000003,HashKeyRange: {StartingHashKey:
 638029437976759618993827388934565396 48,EndingHashKey:
 850705917302346158658436518579420528 63},SequenceNumberRange: {StartingSequenceNumber:
 496278943384139488537140354200999420844746805038777631 22,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 3 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000015,HashKeyRange: {StartingHashKey:
 319014718988379809496913694467282698240,EndingHashKey:
 340282366920938463463374607431768211455},SequenceNumberRange: {StartingSequenceNumber:
 496278943386815577960964028977983707037464608419495283 06,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 2 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000014,HashKeyRange: {StartingHashKey:
 297747071055821155530452781502797185024,EndingHashKey:
```

```
 319014718988379809496913694467282698239},SequenceNumberRange: {StartingSequenceNumber:
 496278943386592570508978722746568349854738124804435478 74,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 3 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000004,HashKeyRange: {StartingHashKey:
 850705917302346158658436518579420528 64,EndingHashKey:
 106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
 496278943384362495989125660432414778027473288653837435 54,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 2 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000003,HashKeyRange: {StartingHashKey:
 638029437976759618993827388934565396 48,EndingHashKey:
 850705917302346158658436518579420528 63},SequenceNumberRange: {StartingSequenceNumber:
 496278943384139488537140354200999420844746805038777631 22,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000001,HashKeyRange: {StartingHashKey:
 212676479325586539664609129644855132 16,EndingHashKey:
 425352958651173079329218259289710264 31},SequenceNumberRange: {StartingSequenceNumber:
 496278943383693473633169741738168706479293837808658022 58,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000009,HashKeyRange: {StartingHashKey:
 191408831393027885698148216680369618 944,EndingHashKey:
 212676479325586539664609129644855132 159},SequenceNumberRange: {StartingSequenceNumber:
 496278943385477533249052191589491563941105706729136457 14,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,532 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000000,HashKeyRange: {StartingHashKey: 0,EndingHashKey:
 212676479325586539664609129644855132 15},SequenceNumberRange: {StartingSequenceNumber:
 496278943383470466181184435506753349296567354193598218 26,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 0 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000012,HashKeyRange: {StartingHashKey:
 255211775190703847597530955573826158 592,EndingHashKey:
 276479423123262501563991868538311671807},SequenceNumberRange: {StartingSequenceNumber:
```

```
 49627894338614655560500811028373763548928515757431587010,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000008,HashKeyRange: {StartingHashKey:
 170141183460469231731687303715884105728,EndingHashKey:
 191408831393027885698148216680369618943},SequenceNumberRange: {StartingSequenceNumber:
 49627894338525452579706688535807620675837922311407665282,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000001,HashKeyRange: {StartingHashKey:
 212676479325586539664609129644855132 16,EndingHashKey:
 425352958651173079329218259289710264 31},SequenceNumberRange: {StartingSequenceNumber:
 49627894338369347363316974173816870647929383780865802258,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,533 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 7 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000011,HashKeyRange: {StartingHashKey:
 233944127258145193631070042609340645376,EndingHashKey:
 255211775190703847597530955573826158591},SequenceNumberRange: {StartingSequenceNumber:
 49627894338592354815302280405232227830655867395925606578,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,533 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000000,HashKeyRange: {StartingHashKey: 0,EndingHashKey:
 212676479325586539664609129644855132 15},SequenceNumberRange: {StartingSequenceNumber:
 49627894338347046618118443550675334929656735419359821826,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:21,568 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 1 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000002,HashKeyRange: {StartingHashKey:
 425352958651173079329218259289710264 32,EndingHashKey:
 638029437976759618993827388934565396 47},SequenceNumberRange: {StartingSequenceNumber:
 49627894338391648108515504796958406366202032142371782690,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,568 INFO  org.apache.flink.streaming.connectors.kinesis.FlinkKinesisConsumer
 [] - Subtask 1 will be seeded with initial shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000013,HashKeyRange: {StartingHashKey:
 276479423123262501563991868538311671808,EndingHashKey:
 297747071055821155530452781502797185023},SequenceNumberRange: {StartingSequenceNumber:
```

```
 496278943386369563056993416515152992672011641189375674 42,}}'}, starting state set as
 sequence number EARLIEST_SEQUENCE_NUM
11:59:21,568 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 1 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000002,HashKeyRange: {StartingHashKey:
 42535295865117307932921825928971026432,EndingHashKey:
 63802943797675961899382738893456539647},SequenceNumberRange: {StartingSequenceNumber:
 496278943383916481085155047969584063662020321423717826 90,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 0
11:59:23,209 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000009,HashKeyRange: {StartingHashKey:
 19140883139302788569814821668036961894 4,EndingHashKey:
 21267647932558653966460912964485513215 9},SequenceNumberRange: {StartingSequenceNumber:
 496278943385477533249052191589491563941105706729136457 14,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,244 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 6 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000010,HashKeyRange: {StartingHashKey:
 21267647932558653966460912964485513216 0,EndingHashKey:
 23394412725814519363107004260934064537 5},SequenceNumberRange: {StartingSequenceNumber:
 496278943385700540701037497820906921123832190344196261 46,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
event: 6; timestamp: 1648030822428, 2022-03-23T10:20:22.428Z
11:59:23,377 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 3 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000015,HashKeyRange: {StartingHashKey:
 31901471898837980949691369446728269824 0,EndingHashKey:
 34028236692093846346337460743176821145 5},SequenceNumberRange: {StartingSequenceNumber:
 496278943386815577960964028977983707037464608419495283 06,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,405 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 2 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000014,HashKeyRange: {StartingHashKey:
 29774707105582115553045278150279718502 4,EndingHashKey:
 31901471898837980949691369446728269823 9},SequenceNumberRange: {StartingSequenceNumber:
 496278943386592570508978722746568349854738124804435478 74,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
```

```
11:59:23,581 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000008,HashKeyRange: {StartingHashKey:
 170141183460469231731687303715884105728,EndingHashKey:
 191408831393027885698148216680369618943},SequenceNumberRange: {StartingSequenceNumber:
 496278943385254525797066885358076206758379223114076652 82,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:23,586 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 1 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000013,HashKeyRange: {StartingHashKey:
 276479423123262501563991868538311671808,EndingHashKey:
 297747071055821155530452781502797185023},SequenceNumberRange: {StartingSequenceNumber:
 496278943386369563056993416515152992672011641189375674 42,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 1
11:59:24,790 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 0 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000012,HashKeyRange: {StartingHashKey:
 255211775190703847597530955573826158592,EndingHashKey:
 276479423123262501563991868538311671807},SequenceNumberRange: {StartingSequenceNumber:
 496278943386146555605008110283737635489285157574315870 10,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 2
event: 4; timestamp: 1648030809282, 2022-03-23T10:20:09.282Z
event: 3; timestamp: 1648030797697, 2022-03-23T10:19:57.697Z
event: 5; timestamp: 1648030810871, 2022-03-23T10:20:10.871Z
11:59:24,907 INFO
 org.apache.flink.streaming.connectors.kinesis.internals.KinesisDataFetcher [] -
 Subtask 7 will start consuming seeded shard StreamShardHandle{streamName='hp-16',
 shard='{ShardId: shardId-000000000011,HashKeyRange: {StartingHashKey:
 233944127258145193631070042609340645376,EndingHashKey:
 255211775190703847597530955573826158591},SequenceNumberRange: {StartingSequenceNumber:
 496278943385923548153022804052322278306558673959256065 78,}}'} from sequence number
 EARLIEST_SEQUENCE_NUM with ShardConsumer 2
event: 7; timestamp: 1648030834105, 2022-03-23T10:20:34.105Z
event: 1; timestamp: 1648030794441, 2022-03-23T10:19:54.441Z
event: 2; timestamp: 1648030796122, 2022-03-23T10:19:56.122Z
event: 8; timestamp: 1648030887171, 2022-03-23T10:21:27.171Z
XXXXXXXXXXXXXX Window with 3 events; Watermark: 1648030809281, 2022-03-23T10:20:09.281Z
3
1
2
XXXXXXXXXXXXXX Window with 2 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
```

```
4
5
XXXXXXXXXXXXXX Window with 1 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
6
XXXXXXXXXXXXXX Window with 1 events; Watermark: 1648030887170, 2022-03-23T10:21:27.170Z
7
```

출력에는 4개의 창만 표시됩니다(이벤트 8이 포함된 마지막 창은 빠짐). 이는 이벤트 시간과 워터마크 전략 때문입니다. 빌드별 워터마크 전략을 사용하면 스트림에서 읽은 마지막 이벤트 시간을 넘어서는 시간이 절대 진행되지 않기 때문에 마지막 창을 닫을 수 없습니다. 하지만 윈도우를 닫으려면 마지막 이벤트 이후 시간이 10초 이상 경과해야 합니다. 이 경우 마지막 워터마크는 2022-03-23T10:21:27.170Z이지만 세션 창을 닫으려면 10초 및 1ms 후의 워터마크가 필요합니다.

워터마크 전략에서 이 `withIdleness` 옵션을 제거하면 창 연산자의 "글로벌 워터마크"를 진행할 수 없으므로 세션 창이 닫히지 않습니다.

참고로 Flink 애플리케이션을 시작할 때(또는 데이터 왜곡이 있는 경우) 일부 샤드는 다른 샤드보다 빨리 소모될 수 있습니다. 이로 인해 일부 워터마크가 하위 작업에서 너무 일찍 생성될 수 있습니다. 하위 작업은 구독한 다른 샤드의 워터마크를 소비하지 않고 한 샤드의 콘텐츠를 기반으로 워터마크를 내보낼 수 있습니다. 이를 완화하는 방법은 안전 버퍼(`forBoundedOutOfOrderness(Duration.ofSeconds(30))`를 추가하거나 늦게 도착하는 이벤트(`allowedLateness(Time.minutes(5))`를 명시적으로 허용하는 다양한 워터마크 전략을 사용하는 것입니다.

모든 연산자에 대해 UUID 설정

Managed Service for Apache Flink가 스냅샷이 있는 애플리케이션에 대한 Flink 작업을 시작하는 경우 특정 문제로 인해 Flink 작업이 시작되지 않을 수 있습니다. 그 중 하나는 연산자 ID 불일치입니다. Flink는 Flink 작업 그래프 연산자에 대해 명시적이고 일관된 연산자 ID를 예상합니다. 명시적으로 설정하지 않으면 Flink는 연산자의 ID를 자동 생성합니다. 이는 Flink가 이러한 연산자 ID를 사용하여 작업 그래프에서 연산자를 고유하게 식별하고 이를 사용하여 각 연산자의 상태를 저장점에 저장하기 때문입니다.

연산자 ID 불일치 문제는 Flink가 작업 그래프의 연산자 ID와 저장점에 정의된 연산자 ID 간의 1:1 매핑을 찾지 못할 때 발생합니다. 이는 명시적으로 일관된 연산자 ID가 설정되지 않은 상태에서 Flink가 연산자 ID를 자동으로 생성하는 경우 발생하며, 이 연산자 ID는 모든 작업 그래프 생성과 일치하지 않을 수 있습니다. 유지보수 실행 중에는 애플리케이션에서 이 문제가 발생할 가능성이 높습니다. 이를 방지하려면 고객이 모든 연산자의 UUID를 Flink 코드로 설정하는 것이 좋습니다. 자세한 정보는 프로덕션 준비 주제의 모든 연산자의 UUID 설정 내용을 참조하세요.

메이븐 ServiceResourceTransformer 쉐이드 플러그인에 추가

Flink는 Java의 서비스 공급자 인터페이스(SPI)를 사용하여 커넥터 및 형식과 같은 구성 요소를 로드합니다. SPI를 사용하는 Flink 종속성이 여러 개 있으면 uber-jar에서 충돌이 발생할 수 있고 예상치 못한 애플리케이션 동작이 발생할 수 있습니다. pom.xml 에 ServiceResourceTransformer정의된 메이븐 쉐이드 플러그인을 추가하는 것이 좋습니다.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
                <executions>
                    <execution>
                        <id>shade</id>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <transformers combine.children="append">
                                <!-- The service transformer is needed to merge META-
INF/services files -->
                                <transformer
 implementation="org.apache.maven.plugins.shade.resource.ServicesResourceTransformer"/>
                                <!-- ... -->
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
        </plugin>
```

Apache Flink 상태 저장 함수

상태 저장 함수는 분산된 상태 저장 애플리케이션 빌드를 간소화하는 API입니다. 강력한 일관성을 보장하면서 동적으로 상호 작용할 수 있는 지속적 상태의 함수를 기반으로 합니다.

상태 저장 함수 애플리케이션은 기본적으로 Apache Flink 애플리케이션일 뿐이므로 Managed Service for Apache Flink에 배포할 수 있습니다. 하지만 Kubernetes 클러스터용 상태 저장 함수 패키징과 Managed Service for Apache Flink상태 저장 함수 패키징에는 몇 가지 차이점이 있습니다. 상태 저장 함수 애플리케이션에서 가장 중요한 점은 상태 저장 함수 런타임을 구성하는 데 필요한 모든 런타임 정보가 모듈 구성에 포함되어 있다는 것입니다. 이 구성은 일반적으로 상태 저장 함수 전용 컨테이너에 패키징되어 Kubernetes에 배포됩니다. 하지만 Managed Service for Apache Flink에서는 불가능합니다.

다음은 Managed Service for Apache Flink에 대한 StateFun Python 예를 수정한 것입니다:

Apache Flink 애플리케이션 템플릿

고객은 상태 저장 함수 런타임에 고객 컨테이너를 사용하는 대신 상태 저장 함수 런타임을 호출하고 필수 종속성을 포함하는 Flink 애플리케이션 jar를 컴파일할 수 있습니다. Flink 1.13의 경우 필수 종속성은 다음과 비슷합니다:

```
<dependency>
 <groupId>org.apache.flink</groupId>
 <artifactId>statefun-flink-distribution</artifactId>
 <version>3.1.0</version>
 <exclusions>
  <exclusion>
   <groupId>org.slf4j</groupId>
   <artifactId>slf4j-log4j12</artifactId>
  </exclusion>
  <exclusion>
   <groupId>log4j</groupId>
   <artifactId>log4j</artifactId>
  </exclusion>
 </exclusions>
</dependency>
```

그리고 Flink 애플리케이션이 상태 저장 함수 런타임을 호출하는 주요 메서드는 다음과 같습니다.

```
public static void main(String[] args) throws Exception {
```

```
 final StreamExecutionEnvironment env =
 StreamExecutionEnvironment.getExecutionEnvironment();

 StatefulFunctionsConfig stateFunConfig = StatefulFunctionsConfig.fromEnvironment(env);

 stateFunConfig.setProvider((StatefulFunctionsUniverseProvider) (classLoader,
 statefulFunctionsConfig) -> {
  Modules modules = Modules.loadFromClassPath();
  return modules.createStatefulFunctionsUniverse(stateFunConfig);
 });

 StatefulFunctionsJob.main(env, stateFunConfig);
}
```

참고로 이들 구성 요소는 일반적이며 상태 저장 함수에 구현된 논리와 독립적입니다.

모듈 구성의 위치입니다.

상태 저장 함수 런타임에서 검색할 수 있으려면 상태 저장 함수 모듈 구성을 클래스 경로에 포함해야 합니다. Flink 애플리케이션의 리소스 폴더에 포함시키고 jar 파일로 패키징하는 것이 가장 좋습니다.

일반적인 Apache Flink 애플리케이션과 마찬가지로 Maven을 사용하여 uber jar 파일을 만들고 Managed Service for Apache Flink에 배포할 수 있습니다.

Managed Service for Apache Flink에 대한 이전 버전 정보

이 주제는 이전 버전의 Apache Flink와 Managed Service for Apache Flink를 사용하는 방법에 대한 내용을 포함하고 있습니다. Managed Service for Apache Flink가 지원하는 Apache Flink 버전은 1.15.2(권장), 1.13.2, 1.11.1, 1.8.2, 1.6.2입니다.

Apache Flink application은 지원되는 최신 버전의 Apache Flink를 Managed Service for Apache Flink와 함께 사용하는 것이 좋습니다. Apache Flink 버전 1.15.2에는 다음과 같은 특징이 있습니다:

- Apache Flink 표 API 및 SQL 지원
- Python 애플리케이션에 대한 지원
- Java 버전 11 및 모든 Scala 버전에 대한 지원
- 개선된 메모리 모델
- 애플리케이션 안정성 향상을 위한 RocksDB 최적화
- Apache Flink 대시보드에서 작업 관리자 및 스택 트레이스에 대한 지원

이 주제는 다음 섹션을 포함하고 있습니다:

- Apache Flink Kinesis 스트림 커넥터를 이전 Apache Flink 버전과 함께 사용
- Apache Flink 1.8.2를 사용한 애플리케이션 구축
- Apache Flink 1.6.2를 사용한 애플리케이션 구축
- 애플리케이션 업그레이드
- Apache Flink 1.6.2 및 1.8.2에서 사용 가능한 커넥터
- 시작하기: Flink 1.13.2
- 시작하기: Flink 1.11.1
- 시작하기: Flink 1.8.2
- 시작하기: Flink 1.6.2

Apache Flink Kinesis 스트림 커넥터를 이전 Apache Flink 버전과 함께 사용

Apache Flink Kinesis Streams 커넥터는 버전 1.11 이전의 Apache Flink에 포함되지 않았습니다. 애플리케이션에서 Apache Flink Kinesis 커넥터를 이전 버전의 Apache Flink와 함께 사용하려면 애플리케

이션에서 사용하는 Apache Flink 버전을 다운로드, 컴파일 및 설치해야 합니다. 이 커넥터는 애플리케이션 소스로 사용되는 Kinesis 스트림의 데이터를 사용하거나 애플리케이션 출력에 사용되는 Kinesis 스트림에 데이터를 쓰는 데 사용됩니다.

> **Note**
>
> KPL 버전 0.14.0 이상으로 커넥터를 구축하고 있는지 확인하세요.

Apache Flink 버전 1.8.2 소스 코드를 다운로드하고 설치하려면 다음을 수행하세요.

1. Apache Maven이 설치되어 있고 JAVA_HOME 환경 변수가 JRE가 아닌 JDK인지 확인하세요. 다음 명령을 사용하여 Apache Maven 설치를 테스트할 수 있습니다.

   ```
   mvn -version
   ```

2. Apache Flink 버전 1.8.2 소스 코드를 다운로드하세요.

   ```
   wget https://archive.apache.org/dist/flink/flink-1.8.2/flink-1.8.2-src.tgz
   ```

3. Apache Flink 소스 코드 압축 해제:

   ```
   tar -xvf flink-1.8.2-src.tgz
   ```

4. Apache Flink 소스 코드 디렉토리로 변경:

   ```
   cd flink-1.8.2
   ```

5. Apache Flink 컴파일 및 설치:

   ```
   mvn clean install -Pinclude-kinesis -DskipTests
   ```

   > **Note**
   >
   > Microsoft Windows에서 Flink를 컴파일하는 경우 `-Drat.skip=true` 파라미터를 추가해야 합니다.

Apache Flink 1.8.2를 사용한 애플리케이션 구축

이 섹션에는 Apache Flink 1.8.2와 작동되는 Managed Service for Apache Flink 애플리케이션을 구축하는 데 사용하는 구성 요소에 대한 정보가 포함되어 있습니다.

Managed Service for Apache Flink 애플리케이션에 다음 구성 요소 버전을 사용하세요:

구성 요소	버전
Java	1.8 (권장)
Apache Flink	1.8.2
Flink 런타임용 Managed Service for Apache Flink(aws-kinesisanalytics-runtime)	1.0.1
Managed Service for Apache Flink Flink 커넥터 (aws-kinesisanalytics-flink)	1.0.1
Apache Maven	3.1

Apache Flink 1.8.2를 사용하여 애플리케이션을 컴파일하려면 다음 파라미터를 사용하여 Maven을 실행하세요.

```
mvn package -Dflink.version=1.8.2
```

Apache Flink 버전 1.8.2를 사용하는 Managed Service for Apache Flink 애플리케이션의 `pom.xml` 파일 예는 Managed Service for Apache Flink 1.8.2 시작하기 애플리케이션을 참조하세요.

Managed Service for Apache Flink 애플리케이션의 애플리케이션 코드를 구축하고 사용하는 방법에 대한 자세한 설명은 애플리케이션 생성 섹션을 참조하세요.

Apache Flink 1.6.2를 사용한 애플리케이션 구축

이 섹션에는 Apache Flink 1.6.2와 작동하는 Managed Service for Apache Flink 애플리케이션을 구축하는 데 사용하는 구성 요소에 대한 정보가 포함되어 있습니다.

Managed Service for Apache Flink 애플리케이션에 다음 구성 요소 버전을 사용하세요:

구성 요소	버전
Java	1.8 (권장)
AWSJava SDK	1.11.379
Apache Flink	1.6.2
Flink 런타임용 Managed Service for Apache Flink(aws-kinesisanalytics-runtime)	1.0.1
Managed Service for Apache Flink Flink 커넥터 (aws-kinesisanalytics-flink)	1.0.1
Apache Maven	3.1
Apache Beam	Apache Flink 1.6.2에서는 지원되지 않습니다.

> **Note**
>
> Managed Service for Apache Flink 런타임 버전 1.0.1을 사용하는 경우 애플리케이션 코드를 컴파일할 때 `-Dflink.version` 파라미터를 사용하지 않고 `pom.xml` 파일에 Apache Flink 버전을 지정합니다.

Flink 버전 1.6.2를 사용하는 Managed Service for Apache Flink 애플리케이션의 `pom.xml` 파일 예는 Managed Service for Apache Flink 1.6.2 시작하기 애플리케이션을 참조하세요.

Managed Service for Apache Flink 애플리케이션의 애플리케이션 코드를 구축하고 사용하는 방법에 대한 자세한 설명은 애플리케이션 생성 섹션을 참조하세요.

애플리케이션 업그레이드

Managed Service for Apache Flink 애플리케이션 버전을 업그레이드하려면 애플리케이션 코드를 업데이트하고 이전 애플리케이션을 삭제한 다음 업데이트된 코드로 새 애플리케이션을 생성해야 합니다. 이렇게 하려면 다음을 수행하세요:

- 애플리케이션 `pom.xml` 파일에서 Managed Service for Apache Flink Runtime 및 Managed Service for Apache Flink 커넥터(aws-kinesisanalytics-flink)의 버전을 1.1.0으로 변경합니다.
- 애플리케이션 `pom.xml` 파일에서 `flink.version` 속성을 제거합니다. 다음 단계에서 애플리케이션 코드를 컴파일할 때 이 파라미터를 제공합니다.
- 다음 명령을 사용하여 애플리케이션 코드를 다시 컴파일합니다:

```
mvn package -Dflink.version=1.15.3
```

- 기존 애플리케이션을 삭제합니다. 애플리케이션을 다시 만들고 애플리케이션 런타임으로 Apache Flink 버전 1.15.2(권장 버전)를 선택합니다.

> **Note**
>
> 이전 애플리케이션 버전의 스냅샷은 사용할 수 없습니다.

Apache Flink 1.6.2 및 1.8.2에서 사용 가능한 커넥터

Apache Flink 프레임워크에는 다양한 소스의 데이터에 액세스하기 위한 커넥터가 포함되어 있습니다.

- Apache Flink 1.6.2 프레임워크에서 사용할 수 있는 커넥터에 대한 자세한 설명은 Apache Flink 설명서(1.6.2)의 커넥터(1.6.2)를 참조하세요.
- Apache Flink 1.8.2 프레임워크에서 사용할 수 있는 커넥터에 대한 자세한 설명은 Apache Flink 설명서(1.8.2)의 커넥터(1.8.2)를 참조하세요.

시작하기: Flink 1.13.2

이 섹션에서는 Apache Flink용 관리형 서비스 및 API의 기본 개념을 소개합니다. DataStream 애플리케이션 생성 및 테스트에 사용할 수 있는 옵션에 대해 설명합니다. 또한 이 가이드의 자습서를 완료하고 첫 번째 애플리케이션을 만드는 데 필요한 도구를 설치하는 방법에 대한 지침도 제공합니다.

주제

- Managed Service for Apache Flink 애플리케이션 구성 요소
- 연습 완료를 위한 필수 조건
- 1단계: AWS 계정 설정 및 관리자 사용자 생성하기
- 다음 단계

- 2단계: AWS Command Line Interface(AWS CLI) 설정
- 3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행
- 4단계: AWS 리소스 정리
- 5단계: 다음 절차

Managed Service for Apache Flink 애플리케이션 구성 요소

Managed Service for Apache Flink 애플리케이션은 데이터를 처리하기 위해 Apache Flink 런타임을 사용하여 입력을 처리하고 출력을 생성하는 Java/Apache Maven 또는 Scala 애플리케이션을 사용합니다.

Managed Service for Apache Flink 애플리케이션에는 다음 구성 요소가 있습니다:

- 런타임 속성: 애플리케이션 코드를 다시 컴파일하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.
- 소스: 애플리케이션은 소스를 사용하여 데이터를 소비합니다. 소스 커넥터는 Kinesis 데이터 스트림, Amazon S3 버킷 등에서 데이터를 읽습니다. 자세한 설명은 소스 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 설명은 DataStream API 연산자 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 설명은 싱크 섹션을 참조하세요.

애플리케이션 코드를 생성, 컴파일 및 패키징한 후 Amazon Simple Storage Service (Amazon S3) 버킷에 코드 패키지를 업로드합니다. 그런 다음 Managed Service for Apache Flink 애플리케이션을 생성합니다. 코드 패키지 위치, Kinesis 데이터 스트림을 스트리밍 데이터 소스로 전달하고, 일반적으로 애플리케이션의 처리된 데이터를 수신하는 스트리밍 또는 파일 위치를 전달합니다.

연습 완료를 위한 필수 조건

이 가이드의 단계를 완료하려면 다음이 필요합니다.

- Java Development Kit(JDK) 버전 11. `JAVA_HOME` 환경 변수가 JDK 설치 위치를 가리키도록 설정합니다.
- 애플리케이션을 개발하고 컴파일하려면 개발 환경(예: Eclipse Java Neon 또는 IntelliJ Idea)을 사용하는 것이 좋습니다.

- Git 클라이언트. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다.
- Apache Maven 컴파일러 플러그인. Maven이 해당 작업 경로에 있어야 합니다. Apache Maven 설치를 테스트하려면 다음을 입력하십시오.

```
$ mvn -version
```

시작하려면 1단계: AWS 계정 설정 및 관리자 사용자 생성하기 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성하기

AWS 계정에 등록

AWS 계정 항목이 없으면 다음 절차에 따라 생성하십시오.

AWS 계정에 가입하려면

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

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

   AWS 계정에 가입하면 AWS 계정 루트 사용자이(가) 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스하는 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

가입 프로세스가 완료되면 AWS가 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자를 보안하며 AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

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

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

2. 루트 사용자에 대해 다중 인증(MFA)을 활성화합니다.

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

관리 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서에서 Enabling AWS IAM Identity Center를 참조하세요.

2. IAM Identity Center에서 관리 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 Configure user access with the default IAM Identity Center 디렉터리를 참조하세요.

관리 사용자로 로그인

- IAM 자격 증명 센터 사용자로 로그인하려면 IAM 자격 증명 센터 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

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

프로그래밍 방식 액세스 권한 부여

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

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

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		• AWS API에 대해서는 IAM 사용 설명서에서 IAM 사용자의 액세스 키 관리를 참조하세요.

다음 단계

2단계: AWS Command Line Interface(AWS CLI) 설정

다음 단계

2단계: AWS Command Line Interface(AWS CLI) 설정

2단계: AWS Command Line Interface(AWS CLI) 설정

이 단계에서는 Managed Service for Apache Flink와 함께 사용하도록 AWS CLI을 다운로드하고 구성합니다.

> **Note**
>
> 이 가이드의 시작하기 연습에서는 해당 계정에서 관리자 자격 증명(`adminuser`)을 사용하여 작업을 수행한다고 가정합니다.

> **Note**
>
> 이미 AWS CLI가 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수도 있습니다. 자세한 설명은 AWS Command Line Interface 사용자 가이드에서 AWS Command Line Interface 설치를 참조하세요. AWS CLI의 버전을 확인하려면 다음 명령을 실행합니다.
>
> ```
> aws --version
> ```
>
> 이 자습서의 연습에서는 다음 버전 이상의 AWS CLI가 필요합니다.

```
aws-cli/1.16.63
```

AWS CLI를 설정하려면

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 AWS Command Line Interface 사용 설명서에서 다음 토픽을 참조하세요.
   - AWS Command Line Interface 설치
   - AWS CLI 구성
2. AWS CLI `config` 파일에 관리자 사용자를 위해 명명된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 명명된 프로필에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드의 명명된 프로필을 참조하세요.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   사용할 수 있는 AWS 지역 목록은 https://docs.aws.amazon.com/general/latest/gr/rande.html의 Amazon Web Services 일반 참조지역 및 엔드포인트를 참조하십시오.

   > **Note**
   >
   > 이 자습서의 예 코드 및 명령은 미국 서부(오레곤) 지역을 사용합니다. 다른 지역을 사용하려면 이 자습서의 코드 및 명령에서 지역을 사용하려는 지역으로 변경하십시오.

3. 명령 프롬프트에서 다음 help 명령을 입력하여 설정을 확인하십시오:

   ```
   aws help
   ```

AWS계정을 설정하고 나면 샘플 애플리케이션을 구성하고 설정을 테스트하는 다음 연습을 시도할 수 있습니다. AWS CLI end-to-end

다음 단계

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

이 연습에서는 데이터 스트림을 소스 및 싱크로 사용하여 Managed Service for Apache Flink 애플리케이션을 만듭니다.

이 섹션은 다음 주제를 포함합니다:

2개의 Amazon Kinesis Data Streams 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(`ExampleInputStream` 및 `ExampleOutputStream`)을 생성하십시오. 이 애플리케이션은 애플리케이션 소스 및 대상 스트림에 대해 이러한 스트림을 사용합니다.

Amazon Kinesis 콘솔 또는 다음 AWS CLI 명령을 사용하여 이러한 스트림을 만들 수 있습니다. 콘솔 지침은 Amazon Kinesis Data Streams 개발자 가이드의 데이터 스트림 생성 및 업데이트를 참조하십시오.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림(`ExampleInputStream`)을 생성하려면 다음 Amazon Kinesis `create-stream` AWS CLI 명령을 사용합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 애플리케이션에서 출력을 쓰는 데 사용하는 두 번째 스트림을 생성하려면 동일한 명령을 실행하여 스트림 명칭을 `ExampleOutputStream`으로 변경합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3
STREAM_NAME = "ExampleInputStream"
def get_data():
    return {
        'event_time': datetime.datetime.now().isoformat(),
        'ticker': random.choice(['AAPL', 'AMZN', 'MSFT', 'INTC', 'TBV']),
        'price': round(random.random() * 100, 2)}
def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name,
            Data=json.dumps(data),
            PartitionKey="partitionkey")
if __name__ == '__main__':
    generate(STREAM_NAME, boto3.client('kinesis', region_name='us-west-2'))
```

2. 이 자습서의 뒷부분에서 stock.py 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드 다운로드 및 검사

이 예제의 Java 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/GettingStarted 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- 프로젝트 객체 모델(pom.xml) 파일에는 Managed Service for Apache Flink 라이브러리를 비롯한 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- BasicStreamingJob.java 파일에는 애플리케이션의 기능을 정의하는 main 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 애플리케이션은 StreamExecutionEnvironment 객체를 사용하여 외부 리소스에 액세스하기 위한 소스 및 싱크 커넥터를 생성합니다.
- 애플리케이션은 정적 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 동적 애플리케이션 속성을 사용하려면 createSourceFromApplicationProperties 및 createSinkFromApplicationProperties 메서드를 사용하여 커넥터를 생성합니다. 이 메서드는 애플리케이션의 속성을 읽어 커넥터를 구성합니다.

  이러한 런타임 속성에 대한 자세한 설명은 런타임 속성 섹션을 참조하십시오.

애플리케이션 코드 컴파일

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 설명은 연습 완료를 위한 필수 조건 섹션을 참조하십시오.

애플리케이션 코드를 컴파일하려면

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. 다음 두 가지 방법 중 하나로 코드를 컴파일하고 패키징할 수 있습니다:

   - 명령행 Maven 도구 사용. `pom.xml` 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

     ```
     mvn package -Dflink.version=1.13.2
     ```

   - 귀하의 개발 환경 사용. 자세한 설명은 해당 개발 환경 설명서를 참조하십시오.

     > **Note**
     >
     > 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

   패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. AWS CLI를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형(JAR 또는 ZIP)을 지정합니다.
2. 컴파일하는 동안 오류가 발생하면 `JAVA_HOME` 환경 변수가 올바르게 설정되어 있는지 확인하십시오.

애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

`target/aws-kinesis-analytics-java-apps-1.0.jar`

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 Amazon Simple Storage Service(Amazon S3) 버킷을 만들고 애플리케이션 코드를 업로드합니다.

애플리케이션 코드 업로드하기

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.

2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 **ka-app-code-*<username>***을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 생성을 선택합니다.
7. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다. 다음을 선택합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔이나 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 생성하고 실행할 수 있습니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 생성하면 사용자 AWS Identity and Access Management (IAM) 및 Amazon CloudWatch Logs 리소스가 자동으로 생성됩니다. AWS CLI를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 만듭니다.

주제

- 애플리케이션 생성 및 실행(콘솔)
- 애플리케이션 생성 및 실행 (AWS CLI)

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.

2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **MyApplication**을 입력합니다.
   - 설명에 **My java test app**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전 풀다운은 Apache Flink 버전 1.13으로 그대로 두십시오.
4. 액세스 권한에서 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
```

```
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 구성

1. MyApplication페이지에서 [구성] 을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 다음을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**

그룹 ID	키	값
ProducerConfigProperties	**AggregationEnabled**	**false**

5. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
6. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
7. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

MyApplication페이지에서 [Stop] 을 선택합니다. 작업을 확인합니다.

애플리케이션 업데이트

콘솔을 사용하여 애플리케이션 속성, 모니터링 설정, 애플리케이션 JAR의 위치 또는 파일 명칭과 같은 애플리케이션 설정을 업데이트할 수 있습니다. 애플리케이션 코드를 업데이트해야 하는 경우 Amazon S3 버킷에서 애플리케이션 JAR을 다시 로드할 수도 있습니다.

MyApplication페이지에서 구성을 선택합니다. 애플리케이션 설정을 업데이트하고 업데이트를 선택합니다.

애플리케이션 생성 및 실행 (AWS CLI)

이 섹션에서는 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. Managed Service for Apache Flink는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Managed Service for Apache Flink를 생성하고 Managed Service for Apache Flink와 상호 작용합니다.

권한 정책 생성

> **Note**
>
> 애플리케이션에 대한 권한 정책 및 역할을 생성해야 합니다. 이러한 IAM 리소스를 생성하지 않으면 애플리케이션은 해당 데이터 및 로그 스트림에 액세스할 수 없습니다.

먼저 소스 스트림에서 두 개의 명령문, 즉 `read` 작업에 대한 권한을 부여하는 명령문과 싱크 스트림에서 `write` 작업에 대한 권한을 부여하는 명령문이 있는 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 `AKReadSourceStreamWriteSinkStream` 권한 정책을 생성합니다. *`username`*을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(*`012345678901`*)를 사용자의 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
```

```
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

> **Note**
>
> AWS SDK for Java를 사용하여 다른 Amazon 서비스에 액세스할 수 있습니다. Managed Service for Apache Flink는 SDK에 필요한 자격 증명을 애플리케이션과 연결된 서비스 실행 IAM 역할의 자격 증명으로 자동 설정합니다. 추가 단계는 필요 없습니다.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할, 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다. 사용 사례 선택에서 Kinesis Analytics를 선택합니다.

다음: 권한을 선택합니다.

4. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.

5. 역할 생성 페이지에서 역할 이름으로 **MF-stream-rw-role**을 입력합니다. Create role(역할 생성)을 선택합니다.

   MF-stream-rw-role이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.

6. 역할에 권한 정책을 연결합니다.

   > **Note**
   >
   > 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계 the section called “권한 정책 생성”에서 생성한 정책을 연결합니다.

   a. 요약 페이지에서 권한 탭을 선택합니다.

   b. 정책 연결을 선택합니다.

   c. 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.

   d. AK ReadSourceStreamWriteSinkStream 정책을 선택하고 Attach policy (Attach policy) 를 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

Managed Service for Apache Flink 애플리케이션 생성

1. 다음 JSON 코드를 create_request.json이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(*username*)를 이전 섹션에서 선택한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(*012345678901*)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_15",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "aws-kinesis-analytics-java-apps-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

2. 위의 요청과 함께 CreateApplication 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```
{
   "ApplicationName": "test"
}
```

2. 다음 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “로깅 설정”.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 UpdateApplication 작업을 사용하여 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
   }
}
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationAWS CLI 작업을 사용합니다.

> **Note**
>
> 파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하십시오.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 the section called “2개의 Amazon Kinesis Data Streams 생성” 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
```

```
}
```

다음 단계

4단계: AWS 리소스 정리

4단계: AWS 리소스 정리

이 섹션에는 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기
- 다음 단계

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.

2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

다음 단계

5단계: 다음 절차

5단계: 다음 절차

이제 Managed Service for Apache Flink 애플리케이션을 만들고 실행했으므로 고급 Managed Service for Apache Flink 솔루션에 대해서는 다음 리소스를 참조하십시오.

- AWS Amazon Kinesis용 스트리밍 데이터 솔루션: AWS Amazon Kinesis용 스트리밍 데이터 솔루션은 스트리밍 데이터를 쉽게 캡처, 저장, 처리 및 전송하는 데 필요한 AWS 서비스를 자동으로 구성합니다. 이 솔루션은 스트리밍 데이터 사용 사례를 해결하기 위한 다양한 옵션을 제공합니다. Apache Flink용 관리형 서비스 옵션은 시뮬레이션된 뉴욕 택시 데이터에 대한 분석 작업을 실행하는 실제 애

플리케이션을 보여주는 end-to-end 스트리밍 ETL 예제를 제공합니다. 이 솔루션은 IAM 역할 및 정책, 대시보드, 경보 등 필요한 모든 AWS 리소스를 설정합니다. CloudWatch CloudWatch

- AWSAmazon MSK용 스트리밍 데이터 솔루션: Amazon MSK용 AWS 스트리밍 데이터 솔루션은 생산자, 스트리밍 스토리지, 소비자 및 목적지를 통해 데이터가 흐르는 AWS CloudFormation 템플릿을 제공합니다.
- Apache Flink 및 Apache Kafka를 사용한 클릭스트림 랩: 스트리밍 스토리지로는 Apache Kafka용 Amazon Managed Streaming을 사용하고 스트림 처리에는 Managed Service for Apache Flink 애플리케이션을 사용하는 클릭스트림 사용 사례를 위한 종합적인 실습입니다.
- Apache Flink용 Amazon Managed Service 워크숍: 이 워크숍에서는 end-to-end 스트리밍 데이터를 거의 실시간으로 수집, 분석 및 시각화하는 스트리밍 아키텍처를 구축합니다. 여러분은 뉴욕시에 있는 택시 회사의 운영을 개선하기 위해 나섰습니다. 뉴욕시에 있는 택시의 원격 측정 데이터를 거의 실시간으로 분석하여 차량 운영을 최적화합니다.
- Managed Service for Apache Flink: 예제: 이 개발자 가이드의 이 섹션에서는 Managed Service for Apache Flink에서 애플리케이션을 만들고 사용하는 예를 제공합니다. 여기에는 Apache Flink 애플리케이션용 관리 서비스를 생성하고 결과를 테스트하는 데 도움이 되는 예제 코드와 step-by-step 지침이 포함되어 있습니다.
- Flink 알아보기: 실습 교육: 확장 가능한 스트리밍 ETL, 분석 및 이벤트 기반 애플리케이션 작성을 시작하는 데 도움이 되는 공식 Apache Flink 입문 교육입니다.

> **Note**
>
> Managed Service for Apache Flink는 이 교육에 사용된 Apache Flink 버전(1.12)을 지원하지 않는다는 점에 유의하십시오. 아파치 플링크용 플링크 매니지드 서비스에서 Flink 1.15.2를 사용할 수 있습니다.

시작하기: Flink 1.11.1

이 항목에는 Apache Flink 1.11.1를 사용하는 시작하기 (API) DataStream 자습서 버전이 포함되어 있습니다.

이 섹션에서는 Apache Flink용 관리 서비스 및 API의 기본 개념을 소개합니다. DataStream 애플리케이션 생성 및 테스트에 사용할 수 있는 옵션에 대해 설명합니다. 또한 이 가이드의 자습서를 완료하고 첫 번째 애플리케이션을 만드는 데 필요한 도구를 설치하는 방법에 대한 지침도 제공합니다.

주제

Managed Service for Apache Flink 애플리케이션 구성 요소

Managed Service for Apache Flink 애플리케이션은 데이터를 처리하기 위해 Apache Flink 런타임을 사용하여 입력을 처리하고 출력을 생성하는 Java/Apache Maven 또는 Scala 애플리케이션을 사용합니다.

Managed Service for Apache Flink 애플리케이션에는 다음과 같은 구성 요소가 있습니다:

- 런타임 속성: 애플리케이션 코드를 다시 컴파일하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.
- 소스: 애플리케이션은 소스를 사용하여 데이터를 소비합니다. 소스 커넥터는 Kinesis 데이터 스트림, Amazon S3 버킷 등에서 데이터를 읽습니다. 자세한 설명은 소스 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 설명은 DataStream API 연산자 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 설명은 싱크 섹션을 참조하세요.

애플리케이션 코드를 생성, 컴파일 및 패키징한 후 Amazon Simple Storage Service (Amazon S3) 버킷에 코드 패키지를 업로드합니다. 그런 다음 Managed Service for Apache Flink 애플리케이션을 생성합니다. 코드 패키지 위치, Kinesis 데이터 스트림을 스트리밍 데이터 소스로 전달하고, 일반적으로 애플리케이션의 처리된 데이터를 수신하는 스트리밍 또는 파일 위치를 전달합니다.

연습 완료를 위한 필수 조건

이 가이드의 단계를 완료하려면 다음이 필요합니다.

- Java Development Kit(JDK) 버전 11. JAVA_HOME 환경 변수가 JDK 설치 위치를 가리키도록 설정합니다.
- 애플리케이션을 개발하고 컴파일하려면 개발 환경(예: Eclipse Java Neon 또는 IntelliJ Idea)을 사용하는 것이 좋습니다.
- Git 클라이언트. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다.
- Apache Maven 컴파일러 플러그인. Maven이 해당 작업 경로에 있어야 합니다. Apache Maven 설치를 테스트하려면 다음을 입력하십시오.

```
$ mvn -version
```

시작하려면 1단계: AWS 계정 설정 및 관리자 사용자 생성하기 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성하기

AWS 계정에 등록

AWS 계정 항목이 없으면 다음 절차에 따라 생성하십시오.

AWS 계정에 가입하려면

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

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

   AWS 계정에 가입하면 AWS 계정 루트 사용자이(가) 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스및 리소스에 액세스하는 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

가입 프로세스가 완료되면 AWS가 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자를 보안하며 AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

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

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자에 대해 다중 인증(MFA)을 활성화합니다.

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

관리 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서에서 Enabling AWS IAM Identity Center를 참조하세요.
2. IAM Identity Center에서 관리 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 Configure user access with the default IAM Identity Center 디렉터리를 참조하세요.

관리 사용자로 로그인

- IAM 자격 증명 센터 사용자로 로그인하려면 IAM 자격 증명 센터 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

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

프로그래밍 방식 액세스 권한 부여

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

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

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		• AWS API에 대해서는 IAM 사용 설명서에서 IAM 사용자의 액세스 키 관리를 참조하세요.

다음 단계

2단계: AWS Command Line Interface(AWS CLI) 설정

2단계: AWS Command Line Interface(AWS CLI) 설정

이 단계에서는 Managed Service for Apache Flink와 함께 사용하도록 AWS CLI을 다운로드하고 구성합니다.

> **Note**
>
> 이 가이드의 시작하기 연습에서는 해당 계정에서 관리자 자격 증명(`adminuser`)을 사용하여 작업을 수행한다고 가정합니다.

> **Note**
>
> 이미 AWS CLI가 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수도 있습니다. 자세한 설명은 AWS Command Line Interface 사용자 가이드에서 AWS Command Line Interface 설치를 참조하세요. AWS CLI의 버전을 확인하려면 다음 명령을 실행합니다.
>
> ```
> aws --version
> ```
>
> 이 자습서의 연습에서는 다음 버전 이상의 AWS CLI가 필요합니다.
>
> ```
> aws-cli/1.16.63
> ```

AWS CLI를 설정하려면

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 AWS Command Line Interface 사용 설명서에서 다음 토픽을 참조하세요.

   - AWS Command Line Interface 설치
   - AWS CLI 구성

2. AWS CLI `config` 파일에 관리자 사용자를 위해 명명된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 명명된 프로필에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드의 명명된 프로필을 참조하세요.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   사용할 수 있는 AWS 지역 목록은 https://docs.aws.amazon.com/general/latest/gr/rande.html의 Amazon Web Services 일반 참조지역 및 엔드포인트를 참조하십시오.

   > **Note**
   >
   > 이 자습서의 예 코드 및 명령은 미국 서부(오레곤) 지역을 사용합니다. 다른 지역을 사용하려면 이 자습서의 코드 및 명령에서 지역을 사용하려는 지역으로 변경하십시오.

3. 명령 프롬프트에서 다음 help 명령을 입력하여 설정을 확인하십시오:

   ```
   aws help
   ```

AWS계정을 설정하고 나면 샘플 애플리케이션을 구성하고 설정을 테스트하는 다음 연습을 시도할 수 있습니다. AWS CLI end-to-end

다음 단계

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

이 연습에서는 데이터 스트림을 소스 및 싱크로 사용하여 Managed Service for Apache Flink 애플리케이션을 만듭니다.

이 섹션은 다음 주제를 포함합니다:

2개의 Amazon Kinesis Data Streams 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(`ExampleInputStream` 및 `ExampleOutputStream`)을 생성하십시오. 이 애플리케이션은 애플리케이션 소스 및 대상 스트림에 대해 이러한 스트림을 사용합니다.

Amazon Kinesis 콘솔 또는 다음 AWS CLI 명령을 사용하여 이러한 스트림을 만들 수 있습니다. 콘솔 지침은 Amazon Kinesis Data Streams 개발자 가이드의 데이터 스트림 생성 및 업데이트를 참조하십시오.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림(`ExampleInputStream`)을 생성하려면 다음 Amazon Kinesis `create-stream` AWS CLI 명령을 사용합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 애플리케이션에서 출력을 쓰는 데 사용하는 두 번째 스트림을 생성하려면 동일한 명령을 실행하여 스트림 명칭을 `ExampleOutputStream`으로 변경합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
```

```
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

2. 이 자습서의 뒷부분에서 `stock.py` 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드 다운로드 및 검사

이 예제의 Java 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. `amazon-kinesis-data-analytics-java-examples/GettingStarted` 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- 프로젝트 객체 모델(pom.xml) 파일에는 Managed Service for Apache Flink 라이브러리를 비롯한 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- `BasicStreamingJob.java` 파일에는 애플리케이션의 기능을 정의하는 `main` 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 애플리케이션은 `StreamExecutionEnvironment` 객체를 사용하여 외부 리소스에 액세스하기 위한 소스 및 싱크 커넥터를 생성합니다.

- 애플리케이션은 정적 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 동적 애플리케이션 속성을 사용하려면 `createSourceFromApplicationProperties` 및 `createSinkFromApplicationProperties` 메서드를 사용하여 커넥터를 생성합니다. 이 메서드는 애플리케이션의 속성을 읽어 커넥터를 구성합니다.

  이러한 런타임 속성에 대한 자세한 설명은 런타임 속성 섹션을 참조하십시오.

애플리케이션 코드 컴파일

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 설명은 연습 완료를 위한 필수 조건 섹션을 참조하십시오.

애플리케이션 코드를 컴파일하려면

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. 다음 두 가지 방법 중 하나로 코드를 컴파일하고 패키징할 수 있습니다:

   - 명령행 Maven 도구 사용. `pom.xml` 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

     ```
     mvn package -Dflink.version=1.11.3
     ```

   - 귀하의 개발 환경 사용. 자세한 설명은 해당 개발 환경 설명서를 참조하십시오.

     > **Note**
     >
     > 제공된 소스 코드는 Java 11의 라이브러리를 사용합니다. 프로젝트의 Java 버전이 11인지 확인하십시오.

   패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. AWS CLI를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형(JAR 또는 ZIP)을 지정합니다.

2. 컴파일하는 동안 오류가 발생하면 `JAVA_HOME` 환경 변수가 올바르게 설정되어 있는지 확인하십시오.

애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 Amazon Simple Storage Service(Amazon S3) 버킷을 만들고 애플리케이션 코드를 업로드합니다.

애플리케이션 코드 업로드하기

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 **ka-app-code-*<username>***을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 생성을 선택합니다.
7. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다. 다음을 선택합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔이나 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 생성하고 실행할 수 있습니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 생성하면 사용자 AWS Identity and Access Management (IAM) 및 Amazon CloudWatch Logs 리소스가 자동으로 생성됩니다. AWS CLI를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 만듭니다.

주제

- 애플리케이션 생성 및 실행(콘솔)
- 애플리케이션 생성 및 실행 (AWS CLI)

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 설명에 **`My java test app`**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전 풀다운은 Apache Flink 버전 1.11(권장 버전)로 그대로 두십시오.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-`*`MyApplication`*`-`*`us-west-2`*
> - 역할: `kinesisanalytics-`*`MyApplication`*`-`*`us-west-2`*

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
```

```
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 속성에서 그룹 ID에 **ProducerConfigProperties**를 입력합니다.
5. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

6. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
7. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
8. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

애플리케이션 실행

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

MyApplication페이지에서 [Stop] 을 선택합니다. 작업을 확인합니다.

애플리케이션 업데이트

콘솔을 사용하여 애플리케이션 속성, 모니터링 설정, 애플리케이션 JAR의 위치 또는 파일 명칭과 같은 애플리케이션 설정을 업데이트할 수 있습니다. 애플리케이션 코드를 업데이트해야 하는 경우 Amazon S3 버킷에서 애플리케이션 JAR을 다시 로드할 수도 있습니다.

MyApplication페이지에서 구성을 선택합니다. 애플리케이션 설정을 업데이트하고 업데이트를 선택합니다.

애플리케이션 생성 및 실행 (AWS CLI)

이 섹션에서는 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. Managed Service for Apache Flink는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 상호 작용합니다.

권한 정책 생성

> **Note**
>
> 애플리케이션에 대한 권한 정책 및 역할을 생성해야 합니다. 이러한 IAM 리소스를 생성하지 않으면 애플리케이션은 해당 데이터 및 로그 스트림에 액세스할 수 없습니다.

먼저 소스 스트림에서 두 개의 명령문, 즉 `read` 작업에 대한 권한을 부여하는 명령문과 싱크 스트림에서 `write` 작업에 대한 권한을 부여하는 명령문이 있는 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 `AKReadSourceStreamWriteSinkStream` 권한 정책을 생성합니다. *`username`*을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(*`012345678901`*)를 사용자의 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
```

```
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

> **Note**
>
> AWS SDK for Java를 사용하여 다른 Amazon 서비스에 액세스할 수 있습니다. Managed Service for Apache Flink는 SDK에 필요한 자격 증명을 애플리케이션과 연결된 서비스 실행 IAM 역할의 자격 증명으로 자동 설정합니다. 추가 단계는 필요 없습니다.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.

2. 탐색 창에서 역할, 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다. 사용 사례 선택에서 Kinesis Analytics를 선택합니다.

   다음: 권한을 선택합니다.
4. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
5. 역할 생성 페이지에서 역할 이름으로 **`MF-stream-rw-role`**을 입력합니다. Create role(역할 생성)을 선택합니다.

   `MF-stream-rw-role`이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.
6. 역할에 권한 정책을 연결합니다.

   > **Note**
   >
   > 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계 the section called “권한 정책 생성”에서 생성한 정책을 연결합니다.

   a. 요약 페이지에서 권한 탭을 선택합니다.
   b. 정책 연결을 선택합니다.
   c. 검색 상자에 **`AKReadSourceStreamWriteSinkStream`**(이전 섹션에서 생성한 정책)을 입력합니다.
   d. AK ReadSourceStreamWriteSinkStream 정책을 선택하고 Attach policy (Attach policy) 를 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

Managed Service for Apache Flink 애플리케이션 생성

1. 다음 JSON 코드를 `create_request.json`이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(*`username`*)를 이전 섹션에서 선택

한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(*012345678901*)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_11",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "aws-kinesis-analytics-java-apps-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
         ]
      }
    }
}
```

2. 위의 요청과 함께 [CreateApplication](#) 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```
{
```

```
    "ApplicationName": "test"
}
```

2. 다음 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “로깅 설정”.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 UpdateApplication 작업을 사용하여 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
```

```
                }
            }
        ]
    }
  }
}
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationAWS CLI 작업을 사용합니다.

> **Note**
>
> 파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하십시오.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 the section called “2개의 Amazon Kinesis Data Streams 생성” 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
```

```
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

다음 단계

4단계: AWS 리소스 정리

4단계: AWS 리소스 정리

이 섹션에는 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기
- 다음 단계

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.

2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

다음 단계

5단계: 다음 절차

5단계: 다음 절차

이제 Managed Service for Apache Flink 애플리케이션을 만들고 실행했으므로 고급 Managed Service for Apache Flink 솔루션에 대해서는 다음 리소스를 참조하십시오.

- AWS Amazon Kinesis용 스트리밍 데이터 솔루션: AWS Amazon Kinesis용 스트리밍 데이터 솔루션은 스트리밍 데이터를 쉽게 캡처, 저장, 처리 및 전송하는 데 필요한 AWS 서비스를 자동으로 구성합니다. 이 솔루션은 스트리밍 데이터 사용 사례를 해결하기 위한 다양한 옵션을 제공합니다. Apache Flink용 관리형 서비스 옵션은 시뮬레이션된 뉴욕 택시 데이터에 대한 분석 작업을 실행하는 실제 애플리케이션을 보여주는 end-to-end 스트리밍 ETL 예제를 제공합니다. 이 솔루션은 IAM 역할 및 정책, 대시보드, 경보 등 필요한 모든 AWS 리소스를 설정합니다. CloudWatch CloudWatch
- AWSAmazon MSK용 스트리밍 데이터 솔루션: Amazon MSK용 AWS 스트리밍 데이터 솔루션은 생산자, 스트리밍 스토리지, 소비자 및 목적지를 통해 데이터가 흐르는 AWS CloudFormation 템플릿을 제공합니다.
- Apache Flink 및 Apache Kafka를 사용한 클릭스트림 랩: 스트리밍 스토리지로는 Apache Kafka용 Amazon Managed Streaming을 사용하고 스트림 처리에는 Managed Service for Apache Flink 애플리케이션을 사용하는 클릭스트림 사용 사례를 위한 종합적인 실습입니다.
- Apache Flink용 Amazon Managed Service 워크숍: 이 워크숍에서는 end-to-end 스트리밍 데이터를 거의 실시간으로 수집, 분석 및 시각화하는 스트리밍 아키텍처를 구축합니다. 여러분은 뉴욕시에 있는 택시 회사의 운영을 개선하기 위해 나섰습니다. 뉴욕시에 있는 택시의 원격 측정 데이터를 거의 실시간으로 분석하여 차량 운영을 최적화합니다.
- Managed Service for Apache Flink: 예제: 이 개발자 가이드의 이 섹션에서는 Managed Service for Apache Flink에서 애플리케이션을 만들고 사용하는 예를 제공합니다. 여기에는 Apache Flink 애플리케이션용 관리 서비스를 생성하고 결과를 테스트하는 데 도움이 되는 예제 코드와 step-by-step 지침이 포함되어 있습니다.
- Flink 알아보기: 실습 교육: 확장 가능한 스트리밍 ETL, 분석 및 이벤트 기반 애플리케이션 작성을 시작하는 데 도움이 되는 공식 Apache Flink 입문 교육입니다.

> **Note**
>
> Managed Service for Apache Flink는 이 교육에 사용된 Apache Flink 버전(1.12)을 지원하지 않는다는 점에 유의하십시오. 아파치 플링크용 플링크 매니지드 서비스에서 Flink 1.15.2를 사용할 수 있습니다.

- 아파치 플링크 코드 예제: 다양한 아파치 플링크 애플리케이션 예제의 GitHub 저장소입니다.

시작하기: Flink 1.8.2

이 항목에는 Apache Flink 1.8.2를 사용하는 시작하기 (API) DataStream 자습서 버전이 포함되어 있습니다.

주제

Managed Service for Apache Flink 애플리케이션 구성 요소

Managed Service for Apache Flink 애플리케이션은 데이터를 처리하기 위해 Apache Flink 런타임을 사용하여 입력을 처리하고 출력을 생성하는 Java/Apache Maven 또는 Scala 애플리케이션을 사용합니다.

Managed Service for Apache Flink 애플리케이션에는 다음과 같은 구성 요소가 있습니다:

- 런타임 속성: 애플리케이션 코드를 다시 컴파일하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.
- 소스: 애플리케이션은 소스를 사용하여 데이터를 소비합니다. 소스 커넥터는 Kinesis 데이터 스트림, Amazon S3 버킷 등에서 데이터를 읽습니다. 자세한 설명은 소스 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 설명은 DataStream API 연산자 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 설명은 싱크 섹션을 참조하세요.

애플리케이션 코드를 생성, 컴파일 및 패키징한 후 Amazon Simple Storage Service (Amazon S3) 버킷에 코드 패키지를 업로드합니다. 그런 다음 Managed Service for Apache Flink 애플리케이션을 생성합니다. 코드 패키지 위치, Kinesis 데이터 스트림을 스트리밍 데이터 소스로 전달하고, 일반적으로 애플리케이션의 처리된 데이터를 수신하는 스트리밍 또는 파일 위치를 전달합니다.

연습 완료를 위한 필수 조건

이 가이드의 단계를 완료하려면 다음이 필요합니다.

- Java Development Kit(JDK) 버전 8. JAVA_HOME 환경 변수가 JDK 설치 위치를 가리키도록 설정합니다.
- 이 자습서에서 Apache Flink Kinesis 커넥터를 사용하려면 Apache Flink를 다운로드하여 설치해야 합니다. 자세한 설명은 Apache Flink Kinesis 스트림 커넥터를 이전 Apache Flink 버전과 함께 사용 섹션을 참조하세요.
- 애플리케이션을 개발하고 컴파일하려면 개발 환경(예: Eclipse Java Neon 또는 IntelliJ Idea)을 사용하는 것이 좋습니다.
- Git 클라이언트. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다.
- Apache Maven 컴파일러 플러그인. Maven이 해당 작업 경로에 있어야 합니다. Apache Maven 설치를 테스트하려면 다음을 입력하십시오.

```
$ mvn -version
```

시작하려면 1단계: AWS 계정 설정 및 관리자 사용자 생성하기 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성하기

AWS 계정에 등록

AWS 계정 항목이 없으면 다음 절차에 따라 생성하십시오.

AWS 계정에 가입하려면

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

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

   AWS 계정에 가입하면 AWS 계정 루트 사용자이(가) 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스및 리소스에 액세스하는 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

가입 프로세스가 완료되면 AWS가 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자를 보안하며 AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

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

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자에 대해 다중 인증(MFA)을 활성화합니다.

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

관리 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서에서 Enabling AWS IAM Identity Center를 참조하세요.
2. IAM Identity Center에서 관리 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 Configure user access with the default IAM Identity Center 디렉터리를 참조하세요.

관리 사용자로 로그인

- IAM 자격 증명 센터 사용자로 로그인하려면 IAM 자격 증명 센터 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

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

프로그래밍 방식 액세스 권한 부여

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

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

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		보안 인증 정보를 사용한 인증을 참조하세요. • AWS SDK와 도구에 대해서는 AWS SDK 및 도구 참조 가이드에서 장기 보안 인증 정보를 사용한 인증을 참조하세요. • AWS API에 대해서는 IAM 사용 설명서에서 IAM 사용자의 액세스 키 관리를 참조하세요.

2단계: AWS Command Line Interface(AWS CLI) 설정

이 단계에서는 Managed Service for Apache Flink와 함께 사용하도록 AWS CLI을 다운로드하고 구성합니다.

> **Note**
>
> 이 가이드의 시작하기 연습에서는 해당 계정에서 관리자 자격 증명(`adminuser`)을 사용하여 작업을 수행한다고 가정합니다.

> **Note**
>
> 이미 AWS CLI가 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수도 있습니다. 자세한 설명은 AWS Command Line Interface 사용자 가이드에서 AWS Command Line Interface 설치를 참조하세요. AWS CLI의 버전을 확인하려면 다음 명령을 실행합니다.
>
> ```
> aws --version
> ```
>
> 이 자습서의 연습에서는 다음 버전 이상의 AWS CLI가 필요합니다.

```
aws-cli/1.16.63
```

AWS CLI를 설정하려면

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 AWS Command Line Interface 사용 설명서에서 다음 토픽을 참조하세요.
   - AWS Command Line Interface 설치
   - AWS CLI 구성
2. AWS CLI `config` 파일에 관리자 사용자를 위해 명명된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 명명된 프로필에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드의 명명된 프로필을 참조하세요.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   사용할 수 있는 지역 목록은 Amazon Web Services 일반 참조의 지역 및 엔드포인트를 참조하십시오.

   > **Note**
   >
   > 이 자습서의 예 코드 및 명령은 미국 서부(오레곤) 지역을 사용합니다. 다른 AWS 지역을 사용하려면 이 자습서의 코드 및 명령에서 지역을 사용하려는 지역으로 변경하십시오.

3. 명령 프롬프트에서 다음 help 명령을 입력하여 설정을 확인하십시오:

   ```
   aws help
   ```

AWS계정을 설정하고 나면 샘플 애플리케이션을 구성하고 end-to-end 설정을 테스트하는 다음 연습을 시도할 수 있습니다. AWS CLI

다음 단계

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

이 연습에서는 데이터 스트림을 소스 및 싱크로 사용하여 Managed Service for Apache Flink 애플리케이션을 만듭니다.

이 섹션은 다음 주제를 포함합니다:

2개의 Amazon Kinesis Data Streams 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(`ExampleInputStream` 및 `ExampleOutputStream`)을 생성하십시오. 이 애플리케이션은 애플리케이션 소스 및 대상 스트림에 대해 이러한 스트림을 사용합니다.

Amazon Kinesis 콘솔 또는 다음 AWS CLI 명령을 사용하여 이러한 스트림을 만들 수 있습니다. 콘솔 지침은 Amazon Kinesis Data Streams 개발자 가이드의 데이터 스트림 생성 및 업데이트를 참조하십시오.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림(`ExampleInputStream`)을 생성하려면 다음 Amazon Kinesis `create-stream` AWS CLI 명령을 사용합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 애플리케이션에서 출력을 쓰는 데 사용하는 두 번째 스트림을 생성하려면 동일한 명령을 실행하여 스트림 명칭을 `ExampleOutputStream`으로 변경합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
```

```
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

2. 이 자습서의 뒷부분에서 `stock.py` 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드 다운로드 및 검사

이 예제의 Java 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. `amazon-kinesis-data-analytics-java-examples/GettingStarted_1_8` 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- 프로젝트 객체 모델(pom.xml) 파일에는 Managed Service for Apache Flink 라이브러리를 비롯한 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- `BasicStreamingJob.java` 파일에는 애플리케이션의 기능을 정의하는 `main` 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 애플리케이션은 `StreamExecutionEnvironment` 객체를 사용하여 외부 리소스에 액세스하기 위한 소스 및 싱크 커넥터를 생성합니다.

- 애플리케이션은 정적 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 동적 애플리케이션 속성을 사용하려면 `createSourceFromApplicationProperties` 및 `createSinkFromApplicationProperties` 메서드를 사용하여 커넥터를 생성합니다. 이 메서드는 애플리케이션의 속성을 읽어 커넥터를 구성합니다.

  이러한 런타임 속성에 대한 자세한 설명은 런타임 속성 섹션을 참조하십시오.

애플리케이션 코드 컴파일

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 설명은 연습 완료를 위한 필수 조건 섹션을 참조하십시오.

> **Note**
>
> Kinesis 커넥터를 Apache Flink 1.11 이전 버전에서 사용하려면 Apache Maven을 다운로드, 빌드, 설치해야 합니다. 자세한 내용은 the section called “Apache Flink Kinesis 스트림 커넥터를 이전 Apache Flink 버전과 함께 사용” 단원을 참조하세요.

애플리케이션 코드를 컴파일하려면

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. 다음 두 가지 방법 중 하나로 코드를 컴파일하고 패키징할 수 있습니다:

   - 명령행 Maven 도구 사용. `pom.xml` 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

     ```
     mvn package -Dflink.version=1.8.2
     ```

   - 귀하의 개발 환경 사용. 자세한 설명은 해당 개발 환경 설명서를 참조하십시오.

     > **Note**
     >
     > 제공된 소스 코드는 Java 1.8의 라이브러리를 사용합니다. 프로젝트의 Java 버전이 1.8인지 확인하십시오.

패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. AWS CLI를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형(JAR 또는 ZIP)을 지정합니다.

2. 컴파일하는 동안 오류가 발생하면 `JAVA_HOME` 환경 변수가 올바르게 설정되어 있는지 확인하십시오.

애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 Amazon Simple Storage Service(Amazon S3) 버킷을 만들고 애플리케이션 코드를 업로드합니다.

애플리케이션 코드 업로드하기

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 **`ka-app-code-<username>`**을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 생성을 선택합니다.
7. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다. 다음을 선택합니다.
9. 개체 정보에 대한 설정은 변경할 필요가 없으므로 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔이나 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 생성하고 실행할 수 있습니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 생성하면 사용자 AWS Identity and Access Management (IAM) 및 Amazon CloudWatch Logs 리소스가 자동으로 생성됩니다. AWS CLI를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 만듭니다.

주제

- 애플리케이션 생성 및 실행(콘솔)
- 애플리케이션 생성 및 실행 (AWS CLI)

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 설명에 **`My java test app`**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.
   - 버전 풀다운은 Apache Flink 1.8(권장 버전)으로 그대로 두십시오.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:

- 정책: kinesis-analytics-service-*MyApplication*-*us-west-2*
- 역할: kinesisanalytics-*MyApplication*-*us-west-2*

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/aws-kinesis-analytics-java-
apps-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
```

```
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.

2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **aws-kinesis-analytics-java-apps-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

5. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
6. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
7. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

애플리케이션 실행

1. MyApplication페이지에서 [Run] 을 선택합니다. 작업을 확인합니다.
2. 애플리케이션이 실행 중이면 페이지를 새로 고칩니다. 콘솔에 애플리케이션 그래프가 표시됩니다.

애플리케이션 중지

MyApplication페이지에서 중지를 선택합니다. 작업을 확인합니다.

애플리케이션 업데이트

콘솔을 사용하여 애플리케이션 속성, 모니터링 설정, 애플리케이션 JAR의 위치 또는 파일 명칭과 같은 애플리케이션 설정을 업데이트할 수 있습니다. 애플리케이션 코드를 업데이트해야 하는 경우 Amazon S3 버킷에서 애플리케이션 JAR을 다시 로드할 수도 있습니다.

MyApplication페이지에서 구성을 선택합니다. 애플리케이션 설정을 업데이트하고 업데이트를 선택합니다.

애플리케이션 생성 및 실행 (AWS CLI)

이 섹션에서는 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. Managed Service for Apache Flink는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Managed Service for Apache Flink를 생성하고 Managed Service for Apache Flink와 상호 작용합니다.

권한 정책 생성

> **Note**
>
> 애플리케이션에 대한 권한 정책 및 역할을 생성해야 합니다. 이러한 IAM 리소스를 생성하지 않으면 애플리케이션은 해당 데이터 및 로그 스트림에 액세스할 수 없습니다.

먼저 소스 스트림에서 두 개의 명령문, 즉 `read` 작업에 대한 권한을 부여하는 명령문과 싱크 스트림에서 `write` 작업에 대한 권한을 부여하는 명령문이 있는 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 `AKReadSourceStreamWriteSinkStream` 권한 정책을 생성합니다. *`username`*을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(*`012345678901`*)를 사용자의 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

> **Note**
>
> AWS SDK for Java를 사용하여 다른 Amazon 서비스에 액세스할 수 있습니다. Managed Service for Apache Flink는 SDK에 필요한 자격 증명을 애플리케이션과 연결된 서비스 실행 IAM 역할의 자격 증명으로 자동 설정합니다. 추가 단계는 필요 없습니다.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할, 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다. 사용 사례 선택에서 Kinesis Analytics를 선택합니다.

   다음: 권한을 선택합니다.
4. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
5. 역할 생성 페이지에서 역할 이름으로 **`MF-stream-rw-role`**을 입력합니다. Create role(역할 생성)을 선택합니다.

   `MF-stream-rw-role`이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.
6. 역할에 권한 정책을 연결합니다.

   > **Note**
   >
   > 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계 the section called “권한 정책 생성”에서 생성한 정책을 연결합니다.

   a. 요약 페이지에서 권한 탭을 선택합니다.

   b. 정책 연결을 선택합니다.

   c. 검색 상자에 **`AKReadSourceStreamWriteSinkStream`**(이전 섹션에서 생성한 정책)을 입력합니다.

   d. AK ReadSourceStreamWriteSinkStream 정책을 선택하고 Attach policy (Attach policy) 를 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

Managed Service for Apache Flink 애플리케이션 생성

1. 다음 JSON 코드를 `create_request.json`이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(*username*)를 이전 섹션에서 선택한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(*012345678901*)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_8",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "aws-kinesis-analytics-java-apps-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
```

```
            ]
        }
    }
}
```

2. 위의 요청과 함께 CreateApplication 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```
{
   "ApplicationName": "test"
}
```

2. 다음 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “로깅 설정”.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 UpdateApplication 작업을 사용하여 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
```

```
                    "AggregationEnabled" : "false"
                }
            },
            {
                "PropertyGroupId": "ConsumerConfigProperties",
                "PropertyMap" : {
                    "aws.region" : "us-west-2"
                }
            }
        ]
      }
   }
}
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationAWS CLI 작업을 사용합니다.

> **Note**
>
> 파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하십시오.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 the section called “2개의 Amazon Kinesis Data Streams 생성” 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
                    "FileKeyUpdate": "aws-kinesis-analytics-java-apps-1.0.jar",
                    "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
                }
            }
        }
    }
}
```

다음 단계

4단계: AWS 리소스 정리

4단계: AWS 리소스 정리

이 섹션에는 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 구성을 선택합니다.
4. 스냅샷 섹션에서 비활성화를 선택한 다음 업데이트를 선택합니다.

5. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

시작하기: Flink 1.6.2

이 항목에는 Apache Flink 1.6.2를 사용하는 시작하기 (API) DataStream 자습서 버전이 포함되어 있습니다.

주제

Managed Service for Apache Flink 구성 요소

Managed Service for Apache Flink 애플리케이션은 데이터를 처리하기 위해 Apache Flink 런타임을 사용하여 입력을 처리하고 출력을 생성하는 Java/Apache Maven 또는 Scala 애플리케이션을 사용합니다.

Managed Service for Apache Flink는 다음과 같은 구성 요소를 갖습니다:

- 런타임 속성: 애플리케이션 코드를 다시 컴파일하지 않고도 런타임 속성을 사용하여 애플리케이션을 구성할 수 있습니다.
- 소스: 애플리케이션은 소스를 사용하여 데이터를 소비합니다. 소스 커넥터는 Kinesis 데이터 스트림, Amazon S3 버킷 등에서 데이터를 읽습니다. 자세한 설명은 소스 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 설명은 DataStream API 연산자 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Kinesis Data Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 설명은 싱크 섹션을 참조하세요.

애플리케이션 코드를 생성, 컴파일 및 패키징한 후 Amazon Simple Storage Service (Amazon S3) 버킷에 코드 패키지를 업로드합니다. 그런 다음 Managed Service for Apache Flink 애플리케이션을 생성합니다. 코드 패키지 위치, Kinesis 데이터 스트림을 스트리밍 데이터 소스로 전달하고, 일반적으로 애플리케이션의 처리된 데이터를 수신하는 스트리밍 또는 파일 위치를 전달합니다.

연습 완료를 위한 필수 조건

이 가이드의 단계를 완료하려면 다음이 필요합니다.

- Java Development Kit(JDK) 버전 8. JAVA_HOME 환경 변수가 JDK 설치 위치를 가리키도록 설정합니다.
- 애플리케이션을 개발하고 컴파일하려면 개발 환경(예: Eclipse Java Neon 또는 IntelliJ Idea)을 사용하는 것이 좋습니다.
- Git 클라이언트. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다.
- Apache Maven 컴파일러 플러그인. Maven이 해당 작업 경로에 있어야 합니다. Apache Maven 설치를 테스트하려면 다음을 입력하십시오.

```
$ mvn -version
```

시작하려면 1단계: AWS 계정 설정 및 관리자 사용자 생성하기 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성하기

AWS 계정에 등록

AWS 계정 항목이 없으면 다음 절차에 따라 생성하십시오.

AWS 계정에 가입하려면

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

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

   AWS 계정에 가입하면 AWS 계정 루트 사용자이(가) 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스및 리소스에 액세스하는 권한이 있습니다. 보안 모범 사례는 관리 사용자에게 관리자 액세스 권한을 할당하고, 루트 사용자만 루트 사용자 액세스 권한이 필요한 작업을 수행하는 것입니다.

가입 프로세스가 완료되면 AWS가 확인 이메일을 전송합니다. 언제든지 https://aws.amazon.com/으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자를 보안하며 AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

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

   루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 사용 설명서의 루트 사용자로 로그인을 참조하세요.
2. 루트 사용자에 대해 다중 인증(MFA)을 활성화합니다.

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

관리 사용자 생성

1. IAM Identity Center를 활성화합니다.

   지침은 AWS IAM Identity Center 사용 설명서에서 Enabling AWS IAM Identity Center를 참조하세요.
2. IAM Identity Center에서 관리 사용자에게 관리 액세스 권한을 부여합니다.

   IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 Configure user access with the default IAM Identity Center 디렉터리를 참조하세요.

관리 사용자로 로그인

- IAM 자격 증명 센터 사용자로 로그인하려면 IAM 자격 증명 센터 사용자를 생성할 때 이메일 주소로 전송된 로그인 URL을 사용합니다.

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

프로그래밍 방식 액세스 권한 부여

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

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

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		보안 인증 정보를 사용한 인증을 참조하세요. • AWS SDK와 도구에 대해서는 AWS SDK 및 도구 참조 가이드에서 장기 보안 인증 정보를 사용한 인증을 참조하세요. • AWS API에 대해서는 IAM 사용 설명서에서 IAM 사용자의 액세스 키 관리를 참조하세요.

2단계: AWS Command Line Interface(AWS CLI) 설정

이 단계에서는 Managed Service for Apache Flink와 함께 사용하도록 AWS CLI을 다운로드하고 구성합니다.

> **Note**
>
> 이 가이드의 시작하기 연습에서는 해당 계정에서 관리자 자격 증명(`adminuser`)을 사용하여 작업을 수행한다고 가정합니다.

> **Note**
>
> 이미 AWS CLI가 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수도 있습니다. 자세한 설명은 AWS Command Line Interface 사용자 가이드에서 AWS Command Line Interface 설치를 참조하세요. AWS CLI의 버전을 확인하려면 다음 명령을 실행합니다.
>
> ```
> aws --version
> ```
>
> 이 자습서의 연습에서는 다음 버전 이상의 AWS CLI가 필요합니다.

```
aws-cli/1.16.63
```

AWS CLI를 설정하려면

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 AWS Command Line Interface 사용 설명서에서 다음 토픽을 참조하세요.
   - AWS Command Line Interface 설치
   - AWS CLI 구성
2. AWS CLI `config` 파일에 관리자 사용자를 위해 명명된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 명명된 프로필에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드의 명명된 프로필을 참조하세요.

   ```
   [profile adminuser]
   aws_access_key_id = adminuser access key ID
   aws_secret_access_key = adminuser secret access key
   region = aws-region
   ```

   사용할 수 있는 AWS 지역 목록은 https://docs.aws.amazon.com/general/latest/gr/rande.html의 Amazon Web Services 일반 참조지역 및 엔드포인트를 참조하십시오.

   > **Note**
   >
   > 이 자습서의 예 코드 및 명령은 미국 서부(오레곤) 지역을 사용합니다. 다른 지역을 사용하려면 이 자습서의 코드 및 명령에서 지역을 사용하려는 지역으로 변경하십시오.

3. 명령 프롬프트에서 다음 help 명령을 입력하여 설정을 확인하십시오:

   ```
   aws help
   ```

AWS계정을 설정하고 나면 샘플 애플리케이션을 구성하고 end-to-end 설정을 테스트하는 다음 연습을 시도할 수 있습니다. AWS CLI

다음 단계

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

3단계: Managed Service for Apache Flink 애플리케이션의 생성 및 실행

이 연습에서는 데이터 스트림을 소스 및 싱크로 사용하여 Managed Service for Apache Flink 애플리케이션을 만듭니다.

이 섹션은 다음 주제를 포함합니다:

- 2개의 Amazon Kinesis Data Streams 생성
- 샘플 레코드를 입력 스트림에 쓰기
- Apache Flink 스트리밍 Java 코드 다운로드 및 검사
- 애플리케이션 코드 컴파일
- Apache Flink 스트리밍 Java 코드 업로드
- Managed Service for Apache Flink 애플리케이션 생성 및 실행

2개의 Amazon Kinesis Data Streams 생성

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(`ExampleInputStream` 및 `ExampleOutputStream`)을 생성하십시오. 이 애플리케이션은 애플리케이션 소스 및 대상 스트림에 대해 이러한 스트림을 사용합니다.

Amazon Kinesis 콘솔 또는 다음 AWS CLI 명령을 사용하여 이러한 스트림을 만들 수 있습니다. 콘솔 지침은 Amazon Kinesis Data Streams 개발자 가이드의 데이터 스트림 생성 및 업데이트를 참조하십시오.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림(`ExampleInputStream`)을 생성하려면 다음 Amazon Kinesis `create-stream` AWS CLI 명령을 사용합니다.

```
$ aws kinesis create-stream \
--stream-name ExampleInputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

2. 애플리케이션에서 출력을 쓰는 데 사용하는 두 번째 스트림을 생성하려면 동일한 명령을 실행하여 스트림 명칭을 `ExampleOutputStream`으로 변경합니다.

```
$ aws kinesis create-stream \
```

```
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

샘플 레코드를 입력 스트림에 쓰기

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기 합니다.

> **Note**
>
> 이 섹션에서는 AWS SDK for Python (Boto)이 필요합니다.

1. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다:

```
import datetime
import json
import random
import boto3

STREAM_NAME = "ExampleInputStream"


def get_data():
    return {
        "EVENT_TIME": datetime.datetime.now().isoformat(),
        "TICKER": random.choice(["AAPL", "AMZN", "MSFT", "INTC", "TBV"]),
        "PRICE": round(random.random() * 100, 2),
    }


def generate(stream_name, kinesis_client):
    while True:
        data = get_data()
        print(data)
        kinesis_client.put_record(
            StreamName=stream_name, Data=json.dumps(data),
 PartitionKey="partitionkey"
```

```
        )


if __name__ == "__main__":
    generate(STREAM_NAME, boto3.client("kinesis"))
```

2. 이 자습서의 뒷부분에서 stock.py 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드 다운로드 및 검사

이 예제의 Java 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

2. amazon-kinesis-data-analytics-java-examples/GettingStarted_1_6 디렉터리로 이동합니다.

애플리케이션 코드에 대해 다음을 유의하십시오:

- 프로젝트 객체 모델(pom.xml) 파일에는 Managed Service for Apache Flink 라이브러리를 비롯한 애플리케이션의 구성 및 종속성에 대한 정보가 들어 있습니다.
- BasicStreamingJob.java 파일에는 애플리케이션의 기능을 정의하는 main 메서드가 들어 있습니다.
- 애플리케이션은 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다:

```
return env.addSource(new FlinkKinesisConsumer<>(inputStreamName,
                new SimpleStringSchema(), inputProperties));
```

- 애플리케이션은 StreamExecutionEnvironment 객체를 사용하여 외부 리소스에 액세스하기 위한 소스 및 싱크 커넥터를 생성합니다.

- 애플리케이션은 정적 속성을 사용하여 소스 및 싱크 커넥터를 만듭니다. 동적 애플리케이션 속성을 사용하려면 `createSourceFromApplicationProperties` 및 `createSinkFromApplicationProperties` 메서드를 사용하여 커넥터를 생성합니다. 이 메서드는 애플리케이션의 속성을 읽어 커넥터를 구성합니다.

  이러한 런타임 속성에 대한 자세한 설명은 런타임 속성 섹션을 참조하십시오.

애플리케이션 코드 컴파일

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 설명은 연습 완료를 위한 필수 조건 섹션을 참조하십시오.

> **Note**
>
> 다음 애플리케이션에 대해 Kinesis 커넥터를 사용하려면 커넥터의 소스 코드를 다운로드하고 Apache Flink 설명서에 설명된 대로 구축해야 합니다.

애플리케이션 코드를 컴파일하려면

1. 애플리케이션 코드를 사용하려면 이를 컴파일하고 JAR 파일로 패키징합니다. 다음 두 가지 방법 중 하나로 코드를 컴파일하고 패키징할 수 있습니다:

   - 명령행 Maven 도구 사용. `pom.xml` 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

     ```
     mvn package
     ```

     > **Note**
     >
     > -Dflink.version 파라미터는 Managed Service for Apache Flink 런타임 버전 1.0.1에는 필요하지 않으며 버전 1.1.0 이상에만 필요합니다. 자세한 설명은 the section called “애플리케이션의 Apache Flink 버전 지정” 섹션을 참조하세요.

   - 귀하의 개발 환경 사용. 자세한 설명은 해당 개발 환경 설명서를 참조하십시오.

패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. AWS CLI를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형(JAR 또는 ZIP)을 지정합니다.

2. 컴파일하는 동안 오류가 발생하면 `JAVA_HOME` 환경 변수가 올바르게 설정되어 있는지 확인하십시오.

애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/aws-kinesis-analytics-java-apps-1.0.jar
```

Apache Flink 스트리밍 Java 코드 업로드

이 섹션에서는 Amazon Simple Storage Service(Amazon S3) 버킷을 만들고 애플리케이션 코드를 업로드합니다.

애플리케이션 코드 업로드하기

1. https://console.aws.amazon.com/s3/에서 Amazon S3 콘솔을 엽니다.
2. 버킷 만들기를 선택합니다.
3. 버킷 명칭 필드에 **ka-app-code-*<username>***을 입력합니다. 버킷 명칭에 사용자 이름 등의 접미사를 추가하여 전역적으로 고유하게 만듭니다. 다음을 선택합니다.
4. 옵션 구성 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
5. 권한 설정 단계에서 설정을 기본값 그대로 두고 다음을 선택합니다.
6. 버킷 생성을 선택합니다.
7. Amazon S3 콘솔에서 ka-app-code- <username>버킷을 선택하고 업로드를 선택합니다.
8. 파일 선택 단계에서 파일 추가를 선택합니다. 이전 단계에서 생성한 `aws-kinesis-analytics-java-apps-1.0.jar` 파일로 이동합니다. 다음을 선택합니다.
9. 권한 설정 단계에서 설정을 기본값 그대로 유지합니다. 다음을 선택합니다.
10. 속성 설정 단계에서 설정을 기본값 그대로 유지합니다. 업로드를 선택합니다.

이제 애플리케이션 코드가 애플리케이션에서 액세스할 수 있는 Amazon S3 버킷에 저장됩니다.

Managed Service for Apache Flink 애플리케이션 생성 및 실행

콘솔이나 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 생성하고 실행할 수 있습니다.

> **Note**
>
> 콘솔을 사용하여 애플리케이션을 생성하면 사용자 AWS Identity and Access Management (IAM) 및 Amazon CloudWatch Logs 리소스가 자동으로 생성됩니다. AWS CLI를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 만듭니다.

주제

애플리케이션 생성 및 실행(콘솔)

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 대시보드에서 분석 애플리케이션 생성을 선택합니다.
3. Managed Service for Apache Flink - 애플리케이션 생성 페이지에서 다음과 같이 애플리케이션 세부 정보를 제공합니다:
   - 애플리케이션 명칭에 **`MyApplication`**을 입력합니다.
   - 설명에 **`My java test app`**를 입력합니다.
   - 런타임에서 Apache Flink를 선택합니다.

     > **Note**
     >
     > Managed Service for Apache Flink는 Apache Flink 버전 1.8.2 또는 1.6.2를 사용합니다.

   - 버전 풀다운을 Apache Flink 1.6으로 변경합니다.
4. 액세스 권한에서 IAM 역할 **`kinesis-analytics-MyApplication-us-west-2`** 생성/업데이트를 선택합니다.
5. 애플리케이션 생성을 선택합니다.

> **Note**
>
> 콘솔을 사용하여 Managed Service for Apache Flink 애플리케이션을 만들 때 내 애플리케이션에 대한 IAM 역할 및 정책을 둘 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 지역을 사용하여 다음과 같이 지정됩니다:
>
> - 정책: `kinesis-analytics-service-MyApplication-us-west-2`
> - 역할: `kinesisanalytics-MyApplication-us-west-2`

IAM 정책 편집

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 여세요.
2. 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 **kinesis-analytics-service-MyApplication-us-west-2** 정책을 선택합니다.
3. 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
4. 다음 정책 예제의 강조 표시된 부분을 정책에 추가하세요. 샘플 계정 ID(*012345678901*)를 내 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ReadCode",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": [
                "arn:aws:s3:::ka-app-code-username/java-getting-started-1.0.jar"
            ]
        },
        {
            "Sid": "DescribeLogGroups",
            "Effect": "Allow",
            "Action": [
```

```
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:*"
            ]
        },
        {
            "Sid": "DescribeLogStreams",
            "Effect": "Allow",
            "Action": [
                "logs:DescribeLogStreams"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:*"
            ]
        },
        {
            "Sid": "PutLogEvents",
            "Effect": "Allow",
            "Action": [
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws:logs:us-west-2:012345678901:log-group:/aws/kinesis-
analytics/MyApplication:log-stream:kinesis-analytics-log-stream"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
```

```
}
```

애플리케이션 구성

1. MyApplication페이지에서 구성을 선택합니다.
2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
   - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
   - Amazon S3 객체 경로에는 **java-getting-started-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytics-MyApplication-us-west-2** 생성/업데이트를 선택합니다.
4. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	**flink.inputstream.initpos**	**LATEST**
ProducerConfigProperties	**aws.region**	**us-west-2**
ProducerConfigProperties	**AggregationEnabled**	**false**

5. 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
6. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
7. 업데이트를 선택합니다.

> **Note**
>
> Amazon CloudWatch 로깅을 활성화하도록 선택하면 Apache Flink용 관리형 서비스에서 로그 그룹과 로그 스트림을 자동으로 생성합니다. 이러한 리소스의 이름은 다음과 같습니다.
>
> - 로그 그룹: `/aws/kinesis-analytics/MyApplication`
> - 로그 스트림: `kinesis-analytics-log-stream`

애플리케이션 실행

1. MyApplication페이지에서 [Run] 을 선택합니다. 작업을 확인합니다.
2. 애플리케이션이 실행 중이면 페이지를 새로 고칩니다. 콘솔에 애플리케이션 그래프가 표시됩니다.

애플리케이션 중지

MyApplication페이지에서 중지를 선택합니다. 작업을 확인합니다.

애플리케이션 업데이트

콘솔을 사용하여 애플리케이션 속성, 모니터링 설정, 애플리케이션 JAR의 위치 또는 파일 명칭과 같은 애플리케이션 설정을 업데이트할 수 있습니다. 애플리케이션 코드를 업데이트해야 하는 경우 Amazon S3 버킷에서 애플리케이션 JAR을 다시 로드할 수도 있습니다.

MyApplication페이지에서 구성을 선택합니다. 애플리케이션 설정을 업데이트하고 업데이트를 선택합니다.

애플리케이션 생성 및 실행 (AWS CLI)

이 섹션에서는 AWS CLI를 사용하여 Managed Service for Apache Flink 애플리케이션을 만들고 실행합니다. Managed Service for Apache Flink는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Managed Service for Apache Flink를 생성하고 Managed Service for Apache Flink와 상호 작용합니다.

권한 정책 생성

먼저 소스 스트림에서 두 개의 명령문, 즉 `read` 작업에 대한 권한을 부여하는 명령문과 싱크 스트림에서 `write` 작업에 대한 권한을 부여하는 명령문이 있는 권한 정책을 만듭니다. 그런 다음 정책을 IAM 역할(다음 섹션에서 생성)에 연결합니다. 따라서 Managed Service for Apache Flink가 역할을 맡을 때 서비스는 소스 스트림에서 읽고 싱크 스트림에 쓸 수 있는 권한이 있습니다.

다음 코드를 사용하여 `AKReadSourceStreamWriteSinkStream` 권한 정책을 생성합니다. *`username`*을 애플리케이션 코드를 저장하기 위해 Amazon S3 버킷을 만들 때 사용한 사용자 이름으로 바꿉니다. Amazon 리소스 이름(ARN)의 계정 ID(*`012345678901`*)를 사용자의 계정 ID로 바꿉니다.

```
{
    "Version": "2012-10-17",
    "Statement": [
```

```
        {
            "Sid": "S3",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": ["arn:aws:s3:::ka-app-code-username",
                "arn:aws:s3:::ka-app-code-username/*"
            ]
        },
        {
            "Sid": "ReadInputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleInputStream"
        },
        {
            "Sid": "WriteOutputStream",
            "Effect": "Allow",
            "Action": "kinesis:*",
            "Resource": "arn:aws:kinesis:us-west-2:012345678901:stream/
ExampleOutputStream"
        }
    ]
}
```

권한 정책을 생성하는 step-by-step 방법에 대한 지침은 IAM 사용 설명서의 자습서: 첫 번째 고객 관리형 정책 생성 및 연결을 참조하십시오.

> **Note**
>
> AWS SDK for Java를 사용하여 다른 Amazon 서비스에 액세스할 수 있습니다. Managed Service for Apache Flink는 SDK에 필요한 자격 증명을 애플리케이션과 연결된 서비스 실행 IAM 역할의 자격 증명으로 자동 설정합니다. 추가 단계는 필요 없습니다.

IAM 역할 생성

이 섹션에서는 Managed Service for Apache Flink 애플리케이션이 소스 스트림을 읽고 싱크 스트림에 쓰기 위해 맡을 수 있는 IAM 역할을 생성합니다.

Managed Service for Apache Flink는 권한 없이 스트림에 액세스할 수 없습니다. IAM 역할을 통해 이러한 권한을 부여합니다. 각 IAM 역할에는 두 가지 정책이 연결됩니다. 신뢰 정책은 Managed Service for Apache Flink가 역할을 취할 수 있는 권한을 부여하고, 권한 정책은 역할을 취한 후 Managed Service for Apache Flink에서 수행할 수 있는 작업을 결정합니다.

이전 섹션에서 생성한 권한 정책을 이 역할에 연결합니다.

IAM 역할을 생성하려면

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 창에서 역할, 역할 생성을 선택합니다.
3. 신뢰할 수 있는 유형의 자격 증명 선택에서 AWS 서비스를 선택합니다. 이 역할을 사용할 서비스 선택에서 Kinesis를 선택합니다. 사용 사례 선택에서 Kinesis Analytics를 선택합니다.

   다음: 권한을 선택합니다.
4. 권한 정책 연결 페이지에서 다음: 검토를 선택합니다. 역할을 생성한 후에 권한 정책을 연결합니다.
5. 역할 생성 페이지에서 역할 이름으로 **`MF-stream-rw-role`**을 입력합니다. Create role(역할 생성)을 선택합니다.

   `MF-stream-rw-role`이라는 새 IAM 역할이 생성되었습니다. 그런 다음 역할의 신뢰 정책 및 권한 정책을 업데이트합니다.
6. 역할에 권한 정책을 연결합니다.

   > **Note**
   >
   > 이 연습에서는 Managed Service for Apache Flink가 이 역할을 취하여 Kinesis 데이터 스트림(소스)에서 데이터를 읽고 출력을 다른 Kinesis 데이터 스트림에 씁니다. 따라서 이전 단계 the section called “권한 정책 생성”에서 생성한 정책을 연결합니다.

   a. 요약 페이지에서 권한 탭을 선택합니다.

   b. 정책 연결을 선택합니다.

   c. 검색 상자에 **`AKReadSourceStreamWriteSinkStream`**(이전 섹션에서 생성한 정책)을 입력합니다.

   d. AK ReadSourceStreamWriteSinkStream 정책을 선택하고 Attach policy (Attach policy) 를 선택합니다.

이제 애플리케이션이 리소스에 액세스하는 데 사용하는 서비스 실행 역할이 생성되었습니다. 새 역할의 ARN을 기록합니다.

역할 생성에 대한 step-by-step 지침은 IAM 사용 설명서의 IAM 역할 생성 (콘솔) 을 참조하십시오.

Managed Service for Apache Flink 애플리케이션 생성

1. 다음 JSON 코드를 `create_request.json`이라는 파일에 저장합니다. 샘플 역할 ARN을 이전에 생성한 역할을 위한 ARN으로 바꿉니다. 버킷 ARN 접미사(*username*)를 이전 섹션에서 선택한 접미사로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(*012345678901*)를 해당 계정 ID로 바꿉니다.

```
{
    "ApplicationName": "test",
    "ApplicationDescription": "my java test app",
    "RuntimeEnvironment": "FLINK-1_6",
    "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
    "ApplicationConfiguration": {
        "ApplicationCodeConfiguration": {
            "CodeContent": {
                "S3ContentLocation": {
                    "BucketARN": "arn:aws:s3:::ka-app-code-username",
                    "FileKey": "java-getting-started-1.0.jar"
                }
            },
            "CodeContentType": "ZIPFILE"
        },
        "EnvironmentProperties":  {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
               }
            },
            {
               "PropertyGroupId": "ConsumerConfigProperties",
               "PropertyMap" : {
                    "aws.region" : "us-west-2"
               }
            }
```

```
            ]
        }
    }
}
```

2. 위의 요청과 함께 CreateApplication 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticsv2 create-application --cli-input-json file://
create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션 시작

이 섹션에서는 StartApplication 작업을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 start_request.json이라는 파일에 저장합니다.

```
{
    "ApplicationName": "test",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

2. 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticsv2 start-application --cli-input-json file://start_request.json
```

애플리케이션이 실행됩니다. Amazon CloudWatch 콘솔에서 Apache Flink용 관리형 서비스 지표를 확인하여 애플리케이션이 작동하는지 확인할 수 있습니다.

애플리케이션 중지

이 섹션에서는 StopApplication 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```
{
   "ApplicationName": "test"
}
```

2. 다음 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticsv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch 로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 을 참조하십시오the section called “로깅 설정”.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 UpdateApplication 작업을 사용하여 애플리케이션의 환경 속성을 변경할 수 있습니다. 이 예제에서는 원본 스트림과 대상 스트림의 리전을 변경합니다.

애플리케이션의 환경 속성 업데이트

1. 다음 JSON 코드를 update_properties_request.json이라는 파일에 저장합니다.

```
{"ApplicationName": "test",
   "CurrentApplicationVersionId": 1,
   "ApplicationConfigurationUpdate": {
      "EnvironmentPropertyUpdates": {
         "PropertyGroups": [
            {
               "PropertyGroupId": "ProducerConfigProperties",
               "PropertyMap" : {
                    "flink.stream.initpos" : "LATEST",
                    "aws.region" : "us-west-2",
                    "AggregationEnabled" : "false"
```

```
                }
            },
            {
                "PropertyGroupId": "ConsumerConfigProperties",
                "PropertyMap" : {
                     "aws.region" : "us-west-2"
                }
            }
         ]
      }
   }
}
```

2. 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticsv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 UpdateApplicationAWS CLI 작업을 사용합니다.

AWS CLI를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 명칭, 새 객체 버전을 지정하여 UpdateApplication를 호출합니다. 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하십시오. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 the section called “2개의 Amazon Kinesis Data Streams 생성” 섹션에서 선택한 접미사로 업데이트하십시오.

```
{
    "ApplicationName": "test",
    "CurrentApplicationVersionId": 1,
    "ApplicationConfigurationUpdate": {
        "ApplicationCodeConfigurationUpdate": {
            "CodeContentUpdate": {
                "S3ContentLocationUpdate": {
                    "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
```

```
                    "FileKeyUpdate": "java-getting-started-1.0.jar"
                }
            }
        }
    }
}
```

4단계: AWS 리소스 정리

이 섹션에는 시작하기 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- Managed Service for Apache Flink 애플리케이션 삭제
- Kinesis Data Streams 삭제
- Amazon S3 객체 및 버킷 삭제
- IAM 리소스 삭제
- CloudWatch 리소스 삭제하기

Managed Service for Apache Flink 애플리케이션 삭제

1. https://console.aws.amazon.com/kinesis에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 구성을 선택합니다.
4. 스냅샷 섹션에서 비활성화를 선택한 다음 업데이트를 선택합니다.
5. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis Data Streams 삭제

1. https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.
4. Kinesis 스트림 페이지에서 를 선택하고 작업을 선택하고 삭제를 선택한 다음 삭제를 확인합니다. ExampleOutputStream

Amazon S3 객체 및 버킷 삭제

1. https://console.aws.amazon.com/s3/ 에서 Amazon S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스 삭제

1. https://console.aws.amazon.com/iam/에서 IAM 콘솔을 엽니다.
2. 탐색 바에서 정책을 선택합니다.
3. 필터 컨트롤에서 kinesis를 입력합니다.
4. kinesis-analytics-service- MyApplication -us-west-2 정책을 선택합니다.
5. 정책 작업을 선택한 후 삭제를 선택합니다.
6. 탐색 모음에서 역할을 선택합니다.
7. 키네시스-애널리틱스 - -US-West-2 역할을 선택하세요. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. https://console.aws.amazon.com/cloudwatch/ 에서 CloudWatch 콘솔을 엽니다.
2. 탐색바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

Apache Flink 설정

Managed Service for Apache Flink는 Apache Flink 프레임워크를 구현한 것입니다. Managed Service for Apache Flink는 이 섹션에 설명된 기본값을 사용합니다. 이러한 값 중 일부는 Managed Service for Apache Flink 애플리케이션에서 코드로 설정할 수 있고 다른 값은 변경할 수 없습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

Apache Flink 구성.

Managed Service for Apache Flink는 대부분의 속성에 대한 Apache Flink 권장 값과 일반적인 응용 프로그램 프로필을 기반으로 하는 몇 가지 값으로 구성된 기본 Flink 구성을 제공합니다. Flink 구성에 대한 자세한 내용은 구성을 참조하세요. 서비스 제공 기본 구성은 대부분의 애플리케이션에서 작동합니다. 하지만 Flink 구성 속성을 조정하여 병렬성이 높고 메모리 및 상태 사용량이 많은 특정 응용 프로그램의 성능을 개선하거나 Apache Flink에서 새로운 디버깅 기능을 활성화해야 하는 경우 지원 사례를 요청하여 특정 속성을 변경할 수 있습니다. 자세한 내용은 AWS 지원 센터를 참조하세요. Apache Flink 대시보드를 사용하여 애플리케이션의 현재 구성을 확인할 수 있습니다.

상태 백엔드

Managed Service for Apache Flink 는 임시 데이터를 상태 백엔드에 저장합니다. Managed Service for Apache Flink는 RockSDBStateBackEnd를 사용합니다. 다른 백엔드를 설정하기 위해 `setStateBackend`을(를) 호출해도 효과가 없습니다.

상태 백엔드에서 다음 기능을 활성화합니다.

- 증분 상태 백엔드 스냅샷
- 비동기 상태 백엔드 스냅샷
- 체크포인트 로컬 복구

Managed Service for Apache Flink에서는 `state.backend.rocksdb.ttl.compaction.filter.enabled` 구성이 기본적으로 활성화되어 있습니다. 이 필터를 사용하면 애플리케이션 코드를 업데이트하여 컴팩션 클린업 전략을 활성화할 수 있습니다. 자세한 내용을 알아보려면 Apache Flink의 설명서에서 Flink 1.8.0의 State TTL을 참조하세요.

상태 백엔드에 대한 자세한 내용은 Apache Flink 설명서의 상태 백엔드를 참조하세요.

체크포인트

Managed Service for Apache Flink는 다음 값이 포함된 기본 체크포인트 구성을 사용합니다. 이 값 중 일부는 변경할 수 있습니다. Managed Service for Apache Flink에서 수정된 체크포인트 값을 사용하려면 CheckpointConfiguration.ConfigurationType을 `CUSTOM`(으)로 설정해야 합니다.

설정	수정할 수 있나요?	수정할 수 있다면 방법이 무엇입니까?	기본값
CheckpointingEnabled	수정 가능	애플리케이션 생성 애플리케이션 업데이트 AWS CloudFormation	True
CheckpointInterval	수정 가능	애플리케이션 생성 애플리케이션 업데이트 AWS CloudFormation	60000
MinPauseBetweenCheckpoints	수정 가능	애플리케이션 생성 애플리케이션 업데이트	5000

설정	수정할 수 있나요?	수정할 수 있다면 방법이 무엇입니까?	기본값
		AWS CloudFormation	
정렬되지 않은 체크포인트	수정 가능	지원 사례	False
동시 체크포인트 수	수정 불가능	해당 사항 없음	1
체크포인트 모드	수정 불가능	해당 사항 없음	딱 한 번
체크포인트 보존 정책	수정 불가능	해당 사항 없음	실패 시
체크포인트 타임아웃	수정 불가능	해당 사항 없음	60분
유지된 최대 체크포인트	수정 불가능	해당 사항 없음	1
재시작 전략	수정 불가능	해당 사항 없음	10초마다 무한 재시도가 가능한 딜레이를 수정했습니다.
체크포인트 및 세이브포인트 위치	수정 불가능	해당 사항 없음	내구성이 뛰어난 체크포인트와 세이브포인트 데이터를 서비스 소유의 S3 버킷에 저장합니다.
상태 백엔드 메모리 임계값	수정 불가능	해당 사항 없음	1048576

세이브포인팅

기본적으로 저장점에서 복원할 때 재개 작업은 저장점의 모든 상태를 복원 중인 프로그램에 다시 매핑하려고 시도합니다. 연산자를 삭제한 경우 기본적으로 누락된 연산자에 해당하는 데이터가 있는 저장점에서의 복원은 실패합니다. 응용 프로그램 FlinkRunConfiguration의 AllowNonRestoredState 파라미터를 `true`(으)로 설정하여 작업이 성공하도록 허용할 수 있습니다. 이렇게 하면 재개 작업이 새 프로그램에 매핑할 수 없는 상태를 건너뛰도록 할 수 있습니다.

자세한 내용을 알아보려면 Apache Flink의 설명서에서 복원되지 않은 상태 허용을 참조하세요.

힙 크기

Managed Service for Apache Flink는 JVM 힙의 각 KPU 3 GiB를 할당하고 네이티브 코드 할당을 위해 1 GiB를 예약합니다. 애플리케이션 용량 증가에 대한 자세한 내용은 the section called “스케일링”을(를) 참조하세요.

JVM 힙 크기에 대한 자세한 내용은 Apache Flink 설명서의 구성을 참조하세요.

버퍼 디블로팅

버퍼 디블로팅은 배압이 높은 어플리케이션에 도움이 될 수 있습니다. 애플리케이션에 체크포인트/세이브포인트에 장애가 발생하는 경우 이 기능을 활성화하면 유용할 수 있습니다. 이를 위해서는 지원 사례를 요청하세요.

자세한 내용은 Apache Flink 설명서의 버퍼 디블로팅 메커니즘을 참조하세요.

수정 가능한 Flink 구성 속성

다음은 지원 사례를 사용하여 수정할 수 있는 Flink 구성 설정입니다. 애플리케이션 접두사를 지정하여 한 번에 둘 이상의 속성을 수정할 수 있으며 여러 애플리케이션의 속성을 동시에 수정할 수 있습니다. 이 목록 외에 수정하려는 다른 Flink 구성 속성이 있는 경우 케이스에 맞는 정확한 속성을 지정하세요.

내결함성

```
restart-strategy:

restart-strategy.fixed-delay.delay:
```

체크포인트 및 상태 백엔드

```
state.backend:

state.backend.fs.memory-threshold:

state.backend.incremental:
```

체크포인트

```
execution.checkpointing.unaligned:
```

RocksDB 네이티브 지표

RocksDB 네이티브 지표는 CloudWatch에 제공되지 않습니다. 활성화되면 Flink 대시보드 또는 사용자 지정 도구를 사용하여 Flink REST API에서 이러한 지표에 액세스할 수 있습니다.

Managed Service for Apache Flink를 통해 고객은 CreateApplicationPresignedURL API를 사용하여 읽기 전용 모드에서 최신 Flink REST API (또는 사용 중인 지원 버전) 에 액세스할 수 있습니다. 이 API는 Flink의 자체 대시보드에서 사용되지만 사용자 지정 모니터링 도구에서도 사용할 수 있습니다.

```
state.backend.rocksdb.compaction.style:

state.backend.rocksdb.memory.partitioned-index-filters:

state.backend.rocksdb.metrics.actual-delayed-write-rate:

state.backend.rocksdb.metrics.background-errors:

state.backend.rocksdb.metrics.block-cache-capacity:

state.backend.rocksdb.metrics.block-cache-pinned-usage:

state.backend.rocksdb.metrics.block-cache-usage:

state.backend.rocksdb.metrics.column-family-as-variable:

state.backend.rocksdb.metrics.compaction-pending:

state.backend.rocksdb.metrics.cur-size-active-mem-table:

state.backend.rocksdb.metrics.cur-size-all-mem-tables:

state.backend.rocksdb.metrics.estimate-live-data-size:

state.backend.rocksdb.metrics.estimate-num-keys:

state.backend.rocksdb.metrics.estimate-pending-compaction-bytes:

state.backend.rocksdb.metrics.estimate-table-readers-mem:

state.backend.rocksdb.metrics.is-write-stopped:

state.backend.rocksdb.metrics.mem-table-flush-pending:
```

state.backend.rocksdb.metrics.num-deletes-active-mem-table:

state.backend.rocksdb.metrics.num-deletes-imm-mem-tables:

state.backend.rocksdb.metrics.num-entries-active-mem-table:

state.backend.rocksdb.metrics.num-entries-imm-mem-tables:

state.backend.rocksdb.metrics.num-immutable-mem-table:

state.backend.rocksdb.metrics.num-live-versions:

state.backend.rocksdb.metrics.num-running-compactions:

state.backend.rocksdb.metrics.num-running-flushes:

state.backend.rocksdb.metrics.num-snapshots:

state.backend.rocksdb.metrics.size-all-mem-tables:

state.backend.rocksdb.thread.num:

고급 상태 백엔드 옵션

state.storage.fs.memory-threshold:

전체 작업 관리자 옵션

task.cancellation.timeout:

taskmanager.jvm-exit-on-oom:

taskmanager.numberOfTaskSlots:

taskmanager.slot.timeout:

taskmanager.network.memory.fraction:

taskmanager.network.memory.max:

taskmanager.network.request-backoff.initial:

taskmanager.network.request-backoff.max:

taskmanager.network.memory.buffer-debloat.enabled:

taskmanager.network.memory.buffer-debloat.period:

taskmanager.network.memory.buffer-debloat.samples:

taskmanager.network.memory.buffer-debloat.threshold-percentages:

메모리 구성

taskmanager.memory.jvm-metaspace.size:

taskmanager.memory.jvm-overhead.fraction:

taskmanager.memory.jvm-overhead.max:

taskmanager.memory.managed.consumer-weights:

taskmanager.memory.managed.fraction:

taskmanager.memory.network.fraction:

taskmanager.memory.network.max:

taskmanager.memory.segment-size:

taskmanager.memory.task.off-heap.size:

RPC / Akka

akka.ask.timeout:

akka.client.timeout:

akka.framesize:

akka.lookup.timeout:

akka.tcp.timeout:

클라이언트

client.timeout:

고급 클러스터 옵션

cluster.intercept-user-system-exit:

cluster.processes.halt-on-fatal-error:

파일 시스템 구성

fs.s3.connection.maximum:

fs.s3a.connection.maximum:

fs.s3a.threads.max:

s3.upload.max.concurrent.uploads:

고급 내결함성 옵션

heartbeat.timeout:

jobmanager.execution.failover-strategy:

메모리 구성

jobmanager.memory.heap.size:

지표

metrics.latency.interval:

REST 엔드포인트 및 클라이언트를 위한 고급 옵션

rest.flamegraph.enabled:

rest.server.numThreads:

고급 SSL 보안 옵션

security.ssl.internal.handshake-timeout:

고급 스케줄링 옵션

slot.request.timeout:

플링크 웹 UI의 고급 옵션

`web.timeout:`

구성된 Flink 속성 보기

Apache Flink Dashboard를 통해 다음 단계에 따라 지원 사례를 통해 직접 구성했거나 수정을 요청한 Apache Flink 속성을 볼 수 있습니다.

1. Flink Dashboard로 이동
2. 왼쪽 탐색 창에서 Job Manager를 선택합니다.
3. 구성을 선택하여 Flink 속성 목록을 확인합니다.

Amazon VPC의 리소스에 액세스하도록 Managed Service for Apache Flink 구성

계정에서 VPC(Virtual Private Cloud)의 프라이빗 서브넷에 연결하도록 Managed Service for Apache Flink를 구성할 수 있습니다. Amazon Virtual Private Cloud(Amazon VPC)를 사용하여 데이터베이스, 캐시 인스턴스, 내부 서비스 등과 같은 리소스에 대해 프라이빗 네트워크를 생성합니다. 애플리케이션을 VPC에 연결하여 실행 중 프라이빗 리소스에 액세스합니다.

이 주제는 다음 섹션을 포함하고 있습니다.

- Amazon VPC 개념
- VPC 애플리케이션 권한
- VPC 연결 Managed Service for Apache Flink 애플리케이션을 위한 인터넷 및 서비스 액세스
- Managed Service for Apache Flink VPC API
- 예: VPC를 사용하여 Amazon MSK 클러스터의 데이터에 액세스

Amazon VPC 개념

Amazon VPC는 Amazon EC2의 네트워킹 계층입니다. Amazon EC2를 처음 사용한다면 Linux 인스턴스용 Amazon EC2 사용자 가이드의 Amazon EC2란 무엇인가?를 읽어 대략적인 내용을 확인하십시오.

다음은 VPC의 핵심 개념입니다:

- Virtual Private Cloud(VPC)는 사용자의 AWS 계정 전용 가상 네트워크입니다.
- 서브넷은 VPC의 IP 주소 범위입니다.
- 라우팅 표에는 네트워크 트래픽을 전달할 위치를 결정하는 데 사용되는 라우팅이라는 규칙 집합이 포함되어 있습니다.
- 인터넷 게이트웨이는 수평 확장되고 가용성이 높은 중복 VPC 구성 요소로, VPC의 인스턴스와 인터넷 간에 통신할 수 있게 해줍니다. 따라서 네트워크 트래픽에 가용성 위험이나 대역폭 제약 조건이 발생하지 않습니다.
- VPC 엔드포인트를 통해 인터넷 게이트웨이, NAT 기기, VPN 연결 또는 AWS Direct Connect 연결을 필요로 하지 않고 PrivateLink 구동 지원 AWS 서비스 및 VPC 엔드포인트 서비스에 비공개로 연결할 수 있습니다. VPC의 인스턴스는 서비스의 리소스와 통신하는 데 퍼블릭 IP 주소를 필요로 하지 않습니다. VPC와 기타 서비스 간의 트래픽은 Amazon 네트워크를 벗어나지 않습니다.

Amazon VPC 서비스에 대한 자세한 설명은 Amazon Virtual Private Cloud 사용자 가이드를 참조하세요.

Managed Service for Apache Flink는 애플리케이션의 VPC 구성에 제공된 서브넷 중 하나에서 탄력적 네트워크 인터페이스를 생성합니다. VPC 서브넷에서 생성되는 탄력적 네트워크 인터페이스의 수는 애플리케이션의 KPU당 병렬 처리 및 병렬성에 따라 달라질 수 있습니다. 애플리케이션 조정에 대한 자세한 설명은 스케일링을 참조하세요.

> **Note**
>
> VPC 구성은 SQL 애플리케이션에 지원되지 않습니다.

> **Note**
>
> Managed Service for Apache Flink는 VPC 구성이 있는 애플리케이션의 체크포인트 및 스냅샷 상태를 관리합니다.

VPC 애플리케이션 권한

이 섹션에서는 애플리케이션이 VPC와 연동하는 데 필요한 권한 정책을 설명합니다. 권한 정책 사용에 대한 자세한 설명은 Amazon Managed Service for Apache Flink의 자격 증명 및 액세스 관리 섹션을 참조하십시오.

다음 권한 정책은 애플리케이션에 VPC와 상호 작용하는 데 필요한 권한을 부여합니다. 이 권한 정책을 사용하려면 애플리케이션의 실행 역할에 추가하십시오.

Amazon VPC에 액세스하기 위한 권한 정책

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "VPCReadOnlyPermissions",
      "Effect": "Allow",
      "Action": [
        "ec2:DescribeVpcs",
        "ec2:DescribeSubnets",
        "ec2:DescribeSecurityGroups",
```

```
        "ec2:DescribeDhcpOptions"
      ],
      "Resource": "*"
    },
    {
      "Sid": "ENIReadWritePermissions",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:CreateNetworkInterfacePermission",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DeleteNetworkInterface"
      ],
      "Resource": "*"
    }

    ]
}
```

> **Note**
>
> 콘솔을 사용하여 애플리케이션 리소스(예: CloudWatch Logs 또는 Amazon VPC)를 지정하면 콘솔은 애플리케이션 실행 역할을 수정하여 해당 리소스에 액세스할 권한을 부여합니다. 콘솔을 사용하지 않고 애플리케이션을 생성하는 경우에는 애플리케이션의 실행 역할을 수동으로 수정만 하면 됩니다.

VPC 연결 Managed Service for Apache Flink 애플리케이션을 위한 인터넷 및 서비스 액세스

계정에서 VPC에 Managed Service for Apache Flink 애플리케이션을 연결할 때 VPC가 액세스 권한을 제공하지 않는 경우 인터넷에 액세스할 수 없습니다. 애플리케이션에 인터넷 액세스가 필요한 경우 다음 조건을 충족해야 합니다:

- Managed Service for Apache Flink 애플리케이션은 프라이빗 서브넷으로만 구성해야 합니다.
- VPC는 퍼블릭 서브넷의 NAT 게이트웨이 또는 인스턴스를 포함해야 합니다.
- 프라이빗 서브넷에서 퍼블릭 서브넷의 NAT 게이트웨이로 향하는 아웃바운드 트래픽을 위한 경로가 있어야 합니다.

> **Note**
>
> 일부 서비스에서는 VPC 엔드포인트를 제공합니다. VPC 엔드포인트를 사용하면 VPC 내에서 인터넷 액세스 권한 없이 Amazon 서비스에 연결할 수 있습니다.

서브넷이 퍼블릭 또는 프라이빗 서브넷인지는 라우팅 표에 의존합니다. 모든 라우팅 표에는 퍼블릭 대상이 있는 패킷의 그 다음 홉을 결정하는 기본 경로가 있습니다.

- 프라이빗 서브넷의 경우: 기본 경로는 NAT 게이트웨이(nat-...) 또는 NAT 인스턴스(eni-...)를 가리킵니다.
- 퍼블릭 서브넷의 경우: 기본 경로는 인터넷 게이트웨이(igw-...)를 가리킵니다.

퍼블릭 서브넷(NAT 포함)과 하나 이상의 프라이빗 서브넷으로 VPC를 구성한 후에는 다음을 수행하여 프라이빗 서브넷과 퍼블릭 서브넷을 식별하십시오.

- 탐색 창의 VPC 콘솔에서 서브넷을 선택합니다.
- 서브넷을 선택한 다음 라우팅 표 탭을 선택합니다. 기본 경로 확인:
  - 퍼블릭 서브넷: 목적지: 0.0.0.0/0, 타겟: igw-...
  - 프라이빗 서브넷: 목적지: 0.0.0.0/0, 타겟: nat-... 또는 eni-...

Managed Service for Apache Flink 애플리케이션을 프라이빗 서브넷과 연결하려면

- https://console.aws.amazon.com/flink에서 Managed Service for Apache Flink콘솔을 엽니다.
- Managed Service for Apache Flink 애플리케이션 페이지에서 애플리케이션을 선택하고 애플리케이션 세부 정보를 선택합니다.
- 애플리케이션 페이지에서 구성을 선택합니다.
- VPC 연결 섹션에서 애플리케이션과 연결할 VPC를 선택합니다. 애플리케이션이 VPC 리소스에 액세스하는 데 사용할 VPC와 연결된 서브넷 및 보안 그룹을 선택합니다.
- 업데이트를 선택합니다.

관련 정보

퍼블릭 및 프라이빗 서브넷이 있는 VPC 생성

NAT 게이트웨이 기본 사항

Managed Service for Apache Flink VPC API

다음과 같은 Managed Service for Apache Flink API 연산을 사용하여 애플리케이션의 VPC를 관리할 수 있습니다. Managed Service for Apache Flink API 사용에 대한 자세한 설명은 API 예 코드을 참조하십시오.

CreateApplication

CreateApplication 작업을 사용하면 생성 중에 VPC 구성을 애플리케이션에 추가할 수 있습니다.

`CreateApplication` 작업에 대한 다음 예 요청 코드에는 애플리케이션 생성 시 VPC 구성이 포함되어 있습니다.

```
{
  "ApplicationName":"MyApplication",
  "ApplicationDescription":"My-Application-Description",
  "RuntimeEnvironment":"FLINK-1_15",
  "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "CodeContentType":"ZIPFILE"
    },
      "FlinkApplicationConfiguration":{
      "ParallelismConfiguration":{
        "ConfigurationType":"CUSTOM",
        "Parallelism":2,
        "ParallelismPerKPU":1,
        "AutoScalingEnabled":true
      }
    },
  "VpcConfigurations": [
        {
           "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
           "SubnetIds": [ "subnet-0123456789abcdef0" ]
        }
```

```
      ]
  }
}
```

AddApplicationVpcConfiguration

애플리케이션이 생성된 후 애플리케이션에 VPC 구성을 추가하려면 AddApplicationVPCConfiguration 작업을 사용하십시오.

`AddApplicationVpcConfiguration` 작업을 위한 다음 예 요청 코드는 기존 애플리케이션에 VPC 구성을 추가합니다.

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfiguration": {
      "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
      "SubnetIds": [ "subnet-0123456789abcdef0" ]
   }
}
```

DeleteApplicationVpcConfiguration

DeleteApplicationVpcConfiguration 작업을 사용하면 애플리케이션에서 VPC 구성을 제거할 수 있습니다.

`AddApplicationVpcConfiguration` 작업을 위한 다음 예 요청은 애플리케이션에서 기존 VPC 구성을 제거합니다.

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfigurationId": "1.1"
}
```

UpdateApplication

UpdateApplication 작업을 사용하면 애플리케이션의 모든 VPC 구성을 한 번에 업데이트할 수 있습니다.

`UpdateApplication` 작업을 위한 다음 예 요청 코드는 애플리케이션의 모든 VPC 구성을 업데이트 합니다.

```
{
   "ApplicationConfigurationUpdate": {
      "VpcConfigurationUpdates": [
         {
            "SecurityGroupIdUpdates": [ "sg-0123456789abcdef0" ],
            "SubnetIdUpdates": [ "subnet-0123456789abcdef0" ],
            "VpcConfigurationId": "2.1"
         }
      ]
   },
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9
}
```

예: VPC를 사용하여 Amazon MSK 클러스터의 데이터에 액세스

VPC에서 Amazon MSK 클러스터의 데이터에 액세스하는 방법에 대한 전체 자습서는 MSK 복제을 참조하십시오.

Managed Service for Apache Flink

다음은 Amazon Managed Service for Apache Flink와 관련하여 발생할 수 있는 문제를 해결하는 데 도움이 되는 내용입니다.

주제

개발 문제 해결

주제

Apache Flink 플레임 그래프

플레임 그래프는 플레임 그래프를 지원하는 Managed Service for Apache Flink버전의 애플리케이션에서 기본적으로 활성화됩니다. Flink 설명서에 나와 있는 것처럼 그래프를 열어 두면 플레임 그래프가 애플리케이션 성능에 영향을 미칠 수 있습니다.

애플리케이션에 대해 플레임 그래프를 비활성화하려면 애플리케이션 ARN에서 플레임 그래프를 비활성화하도록 요청하는 케이스를 생성하십시오. 자세한 설명은 AWS 지원 센터를 참조하십시오.

EFO 커넥터 1.15.2의 자격 증명 공급자 문제

Kinesis 데이터 스트림 EFO 커넥터 버전 최대 1.15.2에는 `FlinkKinesisConsumer`가 `Credential Provider` 구성을 준수하지 않는 알려진 문제가 있습니다. 이 문제로 인해 유효한 구성이 무시되고 있으며, 이로 인해 AUTO 자격 증명 공급자가 사용됩니다. 이로 인해 EFO 커넥터를 사용하여 Kinesis에 대한 교차 계정 액세스를 사용할 때 문제가 발생할 수 있습니다.

이 오류를 해결하려면 EFO 커넥터 버전 1.15.3 이상을 사용하십시오.

지원되지 않는 Kinesis 커넥터가 있는 애플리케이션

Managed Service for Apache Flink버전 1.15는 애플리케이션 JAR 또는 아카이브(ZIP)에 번들로 제공되는 지원되지 않는 Kinesis Connector 버전(1.15.2 이전 버전)을 사용하는 경우 애플리케이션의 시작 또는 업데이트를 자동으로 거부합니다.

거부 오류

다음을 통해 애플리케이션 생성/업데이트 호출을 제출할 때 다음 오류가 표시됩니다:

```
An error occurred (InvalidArgumentException) when calling the CreateApplication
 operation: An unsupported Kinesis connector version has been detected in the
 application. Please update flink-connector-kinesis to any version equal to or newer
 than 1.15.2.
For more information refer to connector fix: https://issues.apache.org/jira/browse/
FLINK-23528
```

해결 단계

- `flink-connector-kinesis`에 대한 애플리케이션 종속성을 업데이트하십시오. Maven을 프로젝트의 빌드 도구로 사용하고 있다면 Maven 종속성 업데이트 을 따르세요. Gradle을 사용하고 있다면 Gradle 종속성 업데이트 을 따르세요.
- 애플리케이션 재패키징
- Amazon S3 버킷에 업로드
- Amazon S3 버킷으로 업로드한 수정된 애플리케이션으로 애플리케이션 생성/업데이트 요청을 다시 제출합니다.
- 동일한 오류 메시지가 계속 표시되면 애플리케이션 종속성을 다시 확인하십시오. 문제가 지속되면 지원 티켓을 생성하세요.

Maven 종속성 업데이트

1. 프로젝트의 pom.xml를 엽니다.
2. 프로젝트의 종속성을 찾아보세요. 모양은 다음과 같습니다.

```
<project>

    ...

    <dependencies>

        ...

        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kinesis</artifactId>
        </dependency>

        ...

    </dependencies>

    ...

</project>
```

3. 1.15.2 이상의 버전으로 flink-connector-kinesis을 업데이트하세요. 예를 들면 다음과 같습니다.

```
<project>

    ...

    <dependencies>

        ...

        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kinesis</artifactId>
            <version>1.15.2</version>
        </dependency>
```

```
        ...

    </dependencies>

    ...

</project>
```

Gradle 종속성 업데이트

1. 프로젝트 `build.gradle`(또는 Kotlin 애플리케이션의 `build.gradle.kts`)를 엽니다.
2. 프로젝트의 종속성을 찾아보세요. 모양은 다음과 같습니다.

```
...

dependencies {

    ...

    implementation("org.apache.flink:flink-connector-kinesis")

    ...

}

...
```

3. 1.15.2 이상의 버전으로 `flink-connector-kinesis`을 업데이트하세요. 예를 들면 다음과 같습니다.

```
...

dependencies {

    ...

    implementation("org.apache.flink:flink-connector-kinesis:1.15.2")

    ...

}
```

```
...
```

컴파일 오류: "프로젝트의 종속성을 해결할 수 없습니다."

Managed Service for Apache Flink 샘플 애플리케이션을 컴파일하려면 먼저 Apache Flink Kinesis 커넥터를 다운로드하여 컴파일한 다음 로컬 Maven 저장소에 추가해야 합니다. 커넥터가 저장소에 추가되지 않은 경우 다음과 비슷한 컴파일 오류가 나타납니다:

```
Could not resolve dependencies for project your project name: Failure to
 find org.apache.flink:flink-connector-kinesis_2.11:jar:1.8.2 in https://
repo.maven.apache.org/maven2 was cached in the local repository, resolution will not be
 reattempted until the update interval of central has elapsed or updates are forced
```

이 오류를 해결하려면 커넥터용 Apache Flink 소스 코드(https://flink.apache.org/downloads.html의 버전 1.8.2)를 다운로드해야 합니다. Apache Flink 소스 코드를 다운로드, 컴파일 및 설치하는 방법에 대한 지침은 the section called "Apache Flink Kinesis 스트림 커넥터를 이전 Apache Flink 버전과 함께 사용"을 참조하십시오.

잘못된 선택: "kinesisanalyticsv2"

Managed Service for Apache FlinkAPI의 v2를 사용하려면 최신 버전 AWS Command Line Interface(AWS CLI)이 필요합니다.

AWS CLI 업그레이드에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드에서 AWS Command Line Interface 설치를 참조하세요.

UpdateApplication Action이 애플리케이션 코드를 다시 로드하지 않음

지정된 S3 객체 버전이 없는 경우 UpdateApplication작업은 동일한 파일 이름을 가진 애플리케이션 코드를 다시 로드하지 않습니다. 애플리케이션 코드를 동일한 파일 명칭으로 다시 로드하려면 S3 버킷의 버전 관리를 활성화하고 `ObjectVersionUpdate` 파라미터를 사용하여 새 객체 버전을 지정하십시오. S3 버킷에서 객체 버전 관리 활성화에 대한 자세한 설명은 버전 관리 활성화 또는 비활성화를 참조하세요.

S3. StreamingFileSink FileNotFoundExceptions

Managed Service for Apache Flink 애플리케이션은 저장 포인트에서 참조하는 진행 중인 부분 파일이 없는 경우 스냅샷에서 시작할 때 진행 중인 부분 파일 `FileNotFoundException`로 실행될 수 있습니

다. 이 오류 모드가 발생하면 Managed Service for Apache Flink 애플리케이션의 연산자 상태는 일반적으로 복구할 수 없으므로 SKIP_RESTORE_FROM_SNAPSHOT을 사용하는 스냅샷 없이 다시 시작해야 합니다. 다음 예 스택트레이스를 참조하십시오:

```
java.io.FileNotFoundException: No such file or directory: s3://your-s3-bucket/pathj/
INSERT/2023/4/19/7/_part-2-1234_tmp_12345678-1234-1234-1234-123456789012
        at
 org.apache.hadoop.fs.s3a.S3AFileSystem.s3GetFileStatus(S3AFileSystem.java:2231)
        at
 org.apache.hadoop.fs.s3a.S3AFileSystem.innerGetFileStatus(S3AFileSystem.java:2149)
        at
 org.apache.hadoop.fs.s3a.S3AFileSystem.getFileStatus(S3AFileSystem.java:2088)
        at org.apache.hadoop.fs.s3a.S3AFileSystem.open(S3AFileSystem.java:699)
        at org.apache.hadoop.fs.FileSystem.open(FileSystem.java:950)
        at
 org.apache.flink.fs.s3hadoop.HadoopS3AccessHelper.getObject(HadoopS3AccessHelper.java:98)
        at
 org.apache.flink.fs.s3.common.writer.S3RecoverableMultipartUploadFactory.recoverInProgressPart
...
```

StreamingFileSinkFlink는 Flink 추상화가 지원하는 파일 시스템에 레코드를 기록합니다. FileSystem 들어오는 스트림을 제한하지 않을 수 있다는 점을 고려하면 데이터는 데이터가 기록될 때 새 파일이 추가되고 한정된 크기의 부분 파일로 구성됩니다. 부품 라이프사이클 및 롤오버 정책에 따라 부분 파일의 타이밍, 크기 및 명칭이 결정됩니다.

> **Note**
>
> 자세한 설명은 부분 파일 라이프사이클를 참조하십시오.

체크포인트 지정 및 저장점 지정(스냅샷) 중에는 보류 중인 모든 파일의 명칭이 바뀌고 구속됩니다. 그러나 진행 중인 부분 파일은 구속되지 않고 명칭이 변경되며 해당 참조는 작업을 복원할 때 사용할 체크포인트 또는 저장점 메타데이터에 보관됩니다. 진행 중인 이러한 부분 파일은 결국 미결로 넘겨지고 후속 체크포인트 또는 저장점에서 명칭을 바꾸고 구속됩니다.

진행 중인 부분 파일이 누락되는 근본 원인과 해결 방법은 다음과 같습니다:

- Apache Flink용 관리 서비스 애플리케이션을 시작하는 데 사용되는 오래된 스냅샷 — 애플리케이션이 중지되거나 업데이트될 때 생성된 최신 시스템 스냅샷만 Amazon S3에서 Apache Flink용 관리 서

비스 애플리케이션을 시작하는 데 사용할 수 있습니다. StreamingFileSink 이러한 장애 클래스를 방지하려면 최신 시스템 스냅샷을 사용하십시오.

  - 예를 들어 중지 또는 업데이트 중에 시스템 트리거 스냅샷 대신 `CreateSnapshot`를 사용하여 만든 스냅샷을 선택할 때 이런 현상이 발생합니다. 이전 스냅샷의 저장 지점에는 후속 체크포인트 또는 저장점에 의해 이름이 변경되고 커밋된 진행 중인 부분 파일에 out-of-date 대한 참조가 보관됩니다.
  - 이는 최신이 아닌 Stop/Update 이벤트에서 시스템이 트리거한 스냅샷을 선택한 경우에도 발생할 수 있습니다. 시스템 스냅샷이 비활성화되었지만 `RESTORE_FROM_LATEST_SNAPSHOT`을 이미 구성한 애플리케이션을 예로 들 수 있습니다. 일반적으로 Amazon S3를 사용하는 Apache Flink 애플리케이션용 관리형 서비스에서는 항상 시스템 스냅샷을 활성화하고 StreamingFileSink `RESTORE_FROM_LATEST_SNAPSHOT` 구성해야 합니다.
- 진행 중인 부분 파일 제거 – 진행 중인 부분 파일은 S3 버킷에 있으므로 해당 버킷에 액세스할 수 있는 다른 구성 요소나 행위자에 의해 제거될 수 있습니다.
  - 이는 앱을 너무 오랫동안 중지하고 앱의 저장 지점에서 참조하는 진행 중인 부분 파일이 S3 버킷 수명 주기 정책에 의해 제거된 경우 발생할 수 있습니다. MultiPartUpload 이러한 장애 클래스를 방지하려면 S3 Bucket MPU 라이프사이클 정책이 사용 사례에 비해 충분히 긴 기간 동안 적용되는지 확인하십시오.
  - 이는 진행 중인 부분 파일을 수동으로 제거하거나 시스템의 다른 구성 요소에 의해 제거된 경우에도 발생할 수 있습니다. 이러한 오류 클래스를 방지하려면 진행 중인 부분 파일이 다른 행위자나 구성 요소에 의해 제거되지 않도록 하십시오.
- 저장점 이후 자동 체크포인트가 트리거되는 경쟁 조건 – 이는 Managed Service for Apache Flink 버전(최대 1.13 포함)에 영향을 미칩니다. 이 문제는 Managed Service for Apache Flink 버전 1.15에서 수정되었습니다. 재발을 방지하려면 Managed Service for Apache Flink 버전 1.15로 애플리케이션을 마이그레이션하십시오. 또한 에서 로 마이그레이션하는 것이 좋습니다. StreamingFileSink FileSink
  - 애플리케이션이 중지되거나 업데이트되면 Managed Service for Apache Flink는 저장점을 트리거하고 두 단계에 걸쳐 애플리케이션을 중지합니다. 두 단계 사이에 자동 체크포인트가 트리거되면 진행 중인 부분 파일의 명칭이 바뀌고 잠재적으로 구속될 수 있으므로 저장점을 사용할 수 없게 됩니다.

FlinkKafkaConsumer 세이브 포인트 사용 중지 관련 문제

FlinkKafkaConsumer 레거시를 사용할 때 시스템 스냅샷을 활성화한 경우 애플리케이션이 업데이트, 중지 또는 스케일링에서 멈출 수 있습니다. 이 문제에 대해 게시된 수정 사항이 없으므로 이 문제를 KafkaSource완화하려면 새 버전으로 업그레이드하는 것이 좋습니다.

스냅샷이 활성화된 상태로 `FlinkKafkaConsumer`를 사용하는 경우 Flink 작업에서 저장 점 API로 중지 요청을 처리할 때 `ClosedException`를 보고하는 런타임 오류가 발생하여 `FlinkKafkaConsumer`이 실패할 수 있습니다. 이러한 상황에서는 Flink 애플리케이션이 중단되고 실패한 체크포인트로 나타납니다.

Flink 1.15 Async Sink 교착 상태

Apache Flink 구현 인터페이스용 AWS 커넥터와 관련된 알려진 문제가 있습니다. AsyncSink 이는 다음 커넥터와 함께 Flink 1.15를 사용하는 애플리케이션에 영향을 줍니다:

- Java 애플리케이션의 경우:
  - KinesisStreamsSink – `org.apache.flink:flink-connector-kinesis`
  - KinesisStreamsSink – `org.apache.flink:flink-connector-aws-kinesis-streams`
  - KinesisFirehoseSink – `org.apache.flink:flink-connector-aws-kinesis-firehose`
  - DynamoDbSink – `org.apache.flink:flink-connector-dynamodb`

- Flink SQL/TableAPI/Python 애플리케이션:
  - kinesis – `org.apache.flink:flink-sql-connector-kinesis`
  - kinesis – `org.apache.flink:flink-sql-connector-aws-kinesis-streams`
  - firehose – `org.apache.flink:flink-sql-connector-aws-kinesis-firehose`
  - dynamodb – `org.apache.flink:flink-sql-connector-dynamodb`

영향을 받는 애플리케이션에서는 다음과 같은 증상이 나타납니다:

- Flink 작업이 `RUNNING` 상태이지만 데이터를 처리하지 못하고 있다;
- 작업이 다시 시작되지 않는다;
- 체크포인트 시간이 초과되었다.

이 문제는 비동기 HTTP 클라이언트를 사용할 때 호출자에게 특정 오류가 표시되지 않는 AWS SDK의 버그로 인해 발생합니다. 이로 인해 싱크는 체크포인트 플러시 작업 중에 “진행 중인 요청”이 완료될 때까지 무기한 대기하게 됩니다.

이 문제는 버전 2.20.144부터 AWS SDK에서 수정되었습니다.

다음은 애플리케이션에서 새 버전의 AWS SDK를 사용하도록 영향을 받는 커넥터를 업데이트하는 방법에 대한 지침입니다:

주제

Java 애플리케이션 업데이트

아래 절차에 따라 Java 애플리케이션을 업데이트하십시오.

flink-connector-kinesis

애플리케이션은 `flink-connector-kinesis`을 사용합니다.

Kinesis 커넥터는 음영 처리를 사용하여 AWS SDK를 비롯한 일부 종속성을 커넥터 jar에 패키징합니다. AWS SDK 버전을 업데이트하려면 다음 절차를 사용하여 이러한 음영 처리된 클래스를 바꾸십시오.

Maven

1. Kinesis 커넥터 및 필수 AWS SDK 모듈을 프로젝트 종속성으로 추가합니다.
2. 구성 `maven-shade-plugin`:

   a. Kinesis 커넥터 jar의 콘텐츠를 복사할 때 음영 처리된 AWS SDK 클래스를 제외하도록 필터를 추가합니다.

   b. Kinesis 커넥터에서 예상하는 재배치 규칙을 추가하여 업데이트된 AWS SDK 클래스를 패키지로 이동합니다.

   pom.xml

```
<project>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.apache.flink</groupId>
            <artifactId>flink-connector-kinesis</artifactId>
            <version>1.15.4</version>
        </dependency>

        <dependency>
            <groupId>software.amazon.awssdk</groupId>
```

```
            <artifactId>kinesis</artifactId>
            <version>2.20.144</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>netty-nio-client</artifactId>
            <version>2.20.144</version>
        </dependency>
        <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>sts</artifactId>
            <version>2.20.144</version>
        </dependency>
        ...
    </dependencies>
    ...
    <build>
        ...
        <plugins>
            ...
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.1.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            ...
                            <filters>
                                ...
                                <filter>
                                    <artifact>org.apache.flink:flink-connector-
kinesis</artifact>
                                    <excludes>
                                        <exclude>org/apache/flink/kinesis/
shaded/software/amazon/awssdk/**</exclude>
                                        <exclude>org/apache/flink/kinesis/
shaded/org/reactivestreams/**</exclude>
                                        <exclude>org/apache/flink/kinesis/
shaded/io/netty/**</exclude>
```

```
                                        <exclude>org/apache/flink/kinesis/
shaded/com/typesafe/netty/**</exclude>
                                    </excludes>
                                </filter>
                                ...
                            </filters>
                            <relocations>
                                ...
                                <relocation>
                                    <pattern>software.amazon.awssdk</pattern>

 <shadedPattern>org.apache.flink.kinesis.shaded.software.amazon.awssdk</
shadedPattern>
                                </relocation>
                                <relocation>
                                    <pattern>org.reactivestreams</pattern>

 <shadedPattern>org.apache.flink.kinesis.shaded.org.reactivestreams</
shadedPattern>
                                </relocation>
                                <relocation>
                                    <pattern>io.netty</pattern>

 <shadedPattern>org.apache.flink.kinesis.shaded.io.netty</shadedPattern>
                                </relocation>
                                <relocation>
                                    <pattern>com.typesafe.netty</pattern>

 <shadedPattern>org.apache.flink.kinesis.shaded.com.typesafe.netty</
shadedPattern>
                                </relocation>
                                ...
                            </relocations>
                        ...
                    </configuration>
                </execution>
            </executions>
        </plugin>
        ...
    </plugins>
    ...
  </build>
</project>
```

Gradle

1. Kinesis 커넥터 및 필수 AWS SDK 모듈을 프로젝트 종속성으로 추가합니다.
2. ShadowJar 구성 조정:

   a. Kinesis 커넥터 jar의 콘텐츠를 복사할 때 음영 처리된 AWS SDK 클래스를 제외하십시오.

   b. 업데이트된 AWS SDK 클래스를 Kinesis 커넥터가 예상하는 패키지로 재배치합니다.

   build.gradle

```
...
dependencies {
    ...
    flinkShadowJar("org.apache.flink:flink-connector-kinesis:1.15.4")

    flinkShadowJar("software.amazon.awssdk:kinesis:2.20.144")
    flinkShadowJar("software.amazon.awssdk:sts:2.20.144")
    flinkShadowJar("software.amazon.awssdk:netty-nio-client:2.20.144")
    ...
}
...
shadowJar {
    configurations = [project.configurations.flinkShadowJar]

    exclude("org/apache/flink/kinesis/shaded/software/amazon/awssdk/**/*.class")
    exclude("org/apache/flink/kinesis/shaded/org/reactivestreams/**/*.class")
    exclude("org/apache/flink/kinesis/shaded/io/netty/**/*.class")
    exclude("org/apache/flink/kinesis/shaded/com/typesafe/netty/**/*.class")

    relocate("software.amazon.awssdk",
 "org.apache.flink.kinesis.shaded.software.amazon.awssdk")
    relocate("org.reactivestreams",
 "org.apache.flink.kinesis.shaded.org.reactivestreams")
    relocate("io.netty", "org.apache.flink.kinesis.shaded.io.netty")
    relocate("com.typesafe.netty",
 "org.apache.flink.kinesis.shaded.com.typesafe.netty")
}
...
```

영향을 받는 기타 커넥터

애플리케이션이 영향을 받는 다른 커넥터를 사용하는 경우:

AWS SDK 버전을 업데이트하려면 프로젝트 빌드 구성에 SDK 버전을 적용해야 합니다.

Maven

pom.xml 파일의 종속성 관리 섹션에 AWS SDK 재료 명세서(BOM)를 추가하여 프로젝트에 SDK 버전을 적용하세요.

pom.xml

```
<project>
    ...
    <dependencyManagement>
        <dependencies>
            ...
            <dependency>
                <groupId>software.amazon.awssdk</groupId>
                <artifactId>bom</artifactId>
                <version>2.20.144</version>
                <scope>import</scope>
                <type>pom</type>
            </dependency>
            ...
        </dependencies>
    </dependencyManagement>
    ...
</project>
```

Gradle

AWS SDK 재료 명세서(BOM)에 플랫폼 종속성을 추가하여 프로젝트에 SDK 버전을 적용하세요. 이를 위해서는 Gradle 5.0 이상이 필요합니다.

build.gradle

```
...
dependencies {
    ...
    flinkShadowJar(platform("software.amazon.awssdk:bom:2.20.144"))
    ...
```

```
}
...
```

Python 애플리케이션 업데이트

Python 애플리케이션은 커넥터를 두 가지 방식으로 사용할 수 있습니다. 커넥터 및 기타 Java 종속성을 단일 uber-jar의 일부로 패키징하거나 커넥터 jar를 직접 사용하는 것입니다. Async Sink 교착 상태의 영향을 받는 애플리케이션을 수정하려면,

- 애플리케이션이 uber jar를 사용하는 경우 Java 애플리케이션 업데이트 의 지침을 따르세요.
- 소스에서 커넥터 jar를 다시 빌드하려면 다음 단계를 사용하십시오:

소스에서 커넥터 만들기:

Flink 빌드 요건과 유사한 사전 요건:

- Java 11
- Maven 3.2.5

flink-sql-connector-kinesis

1. Flink 1.15.4의 소스 코드 다운로드:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. 소스 코드 압축 해제:

```
tar -xvf flink-1.15.4-src.tgz
```

3. kinesis 커넥터 디렉터리로 이동합니다.

```
cd flink-1.15.4/flink-connectors/flink-connector-kinesis/
```

4. 필요한 AWS SDK 버전을 지정하여 커넥터 jar를 컴파일하고 설치합니다. 빌드 속도를 높이려면 테스트 실행을 건너뛰는 `-DskipTests` 방법과 추가 소스 코드 검사를 건너뛰는 `-Dfast` 방법을 사용하십시오.

```
mvn clean install -DskipTests -Dfast -Daws.sdkv2.version=2.20.144
```

5. kinesis 커넥터 디렉터리로 이동합니다.

```
cd ../flink-sql-connector-kinesis
```

6. sql 커넥터 jar 컴파일 및 설치:

```
mvn clean install -DskipTests -Dfast
```

7. 결과 jar는 다음에서 확인할 수 있습니다:

```
target/flink-sql-connector-kinesis-1.15.4.jar
```

flink-sql-connector-aws-키네시스-스트림

1. Flink 1.15.4의 소스 코드 다운로드:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. 소스 코드 압축 해제:

```
tar -xvf flink-1.15.4-src.tgz
```

3. kinesis 커넥터 디렉터리로 이동합니다.

```
cd flink-1.15.4/flink-connectors/flink-connector-aws-kinesis-streams/
```

4. 필요한 AWS SDK 버전을 지정하여 커넥터 jar를 컴파일하고 설치합니다. 빌드 속도를 높이려면 테스트 실행을 건너뛰는 `-DskipTests` 방법과 추가 소스 코드 검사를 건너뛰는 `-Dfast` 방법을 사용하십시오.

```
mvn clean install -DskipTests -Dfast -Daws.sdk.version=2.20.144
```

5. kinesis 커넥터 디렉터리로 이동합니다.

```
cd ../flink-sql-connector-aws-kinesis-streams
```

6. sql 커넥터 jar 컴파일 및 설치:

```
mvn clean install -DskipTests -Dfast
```

7. 결과 jar는 다음에서 확인할 수 있습니다:

```
target/flink-sql-connector-aws-kinesis-streams-1.15.4.jar
```

flink-sql-connector-aws- 중국어 - 파이어 호스

1. Flink 1.15.4의 소스 코드 다운로드:

```
wget https://archive.apache.org/dist/flink/flink-1.15.4/flink-1.15.4-src.tgz
```

2. 소스 코드 압축 해제:

```
tar -xvf flink-1.15.4-src.tgz
```

3. 커넥터 디렉토리로 이동합니다.

```
cd flink-1.15.4/flink-connectors/flink-connector-aws-kinesis-firehose/
```

4. 필요한 AWS SDK 버전을 지정하여 커넥터 jar를 컴파일하고 설치합니다. 빌드 속도를 높이려면 테스트 실행을 건너뛰는 `-DskipTests` 방법과 추가 소스 코드 검사를 건너뛰는 `-Dfast` 방법을 사용하십시오.

```
mvn clean install -DskipTests -Dfast -Daws.sdk.version=2.20.144
```

5. sql 커넥터 디렉터리로 이동합니다.

```
cd ../flink-sql-connector-aws-kinesis-firehose
```

6. sql 커넥터 jar 컴파일 및 설치:

```
mvn clean install -DskipTests -Dfast
```

7. 결과 jar는 다음에서 확인할 수 있습니다:

```
target/flink-sql-connector-aws-kinesis-firehose-1.15.4.jar
```

flink-sql-connector-dynamodb

1. Flink 1.15.4의 소스 코드 다운로드:

```
wget https://archive.apache.org/dist/flink/flink-connector-aws-3.0.0/flink-
connector-aws-3.0.0-src.tgz
```

2. 소스 코드 압축 해제:

```
tar -xvf flink-connector-aws-3.0.0-src.tgz
```

3. 커넥터 디렉토리로 이동합니다.

```
cd flink-connector-aws-3.0.0
```

4. 필요한 AWS SDK 버전을 지정하여 커넥터 jar를 컴파일하고 설치합니다. 빌드 속도를 높이려면 테스트 실행을 건너뛰는 `-DskipTests` 방법과 추가 소스 코드 검사를 건너뛰는 `-Dfast` 방법을 사용하십시오.

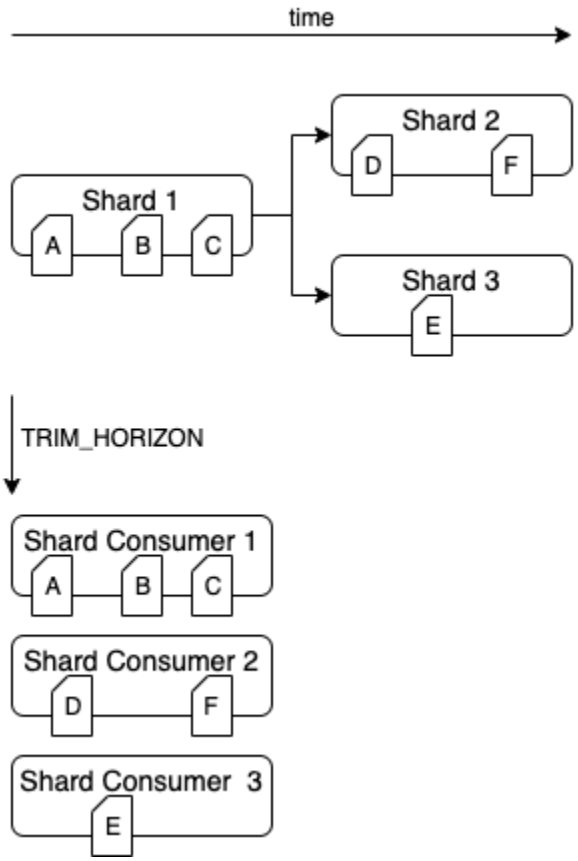
```
mvn clean install -DskipTests -Dfast -Dflink.version=1.15.4 -
Daws.sdk.version=2.20.144
```

5. 결과 jar는 다음에서 확인할 수 있습니다:

```
flink-sql-connector-dynamodb/target/flink-sql-connector-dynamodb-3.0.0.jar
```

리샤딩 중 Amazon Kinesis Data Streams 소스 처리 순서가 잘못됨

현재 FlinkKinesisConsumer 구현에서는 Kinesis 샤드 간에 강력한 주문 보장을 제공하지 않습니다. 이로 인해 특히 out-of-order 처리 지연이 발생하는 Flink 애플리케이션의 경우 Kinesis Stream을 다시 샤딩하는 동안 처리가 발생할 수 있습니다. 이벤트 시간을 기반으로 하는 Windows 연산자와 같은 일부 상황에서는 결과적으로 지연이 발생하여 이벤트가 삭제될 수 있습니다.



이는 오픈 소스 Flink의 알려진 문제입니다. 커넥터 수정 사항이 제공될 때까지 다시 파티셔닝하는 동안 Flink 애플리케이션이 Kinesis Data Streams에 뒤쳐지지 않도록 합니다. Flink 앱이 처리 지연을 허용하도록 하면 처리의 out-of-order 영향과 데이터 손실 위험을 최소화할 수 있습니다.

런타임 문제 해결

이 섹션에는 Managed Service for Apache Flink 애플리케이션의 런타임 문제를 진단하고 수정하는 방법에 대한 정보가 포함되어 있습니다.

주제

- 문제 해결 도구
- 애플리케이션 문제
- 애플리케이션 다시 시작 중
- 처리량이 너무 느림
- 영구 지속적 성장
- I/O 바운드 연산자
- Kinesis 데이터 스트림의 업스트림 또는 소스 조절

- 체크포인트
- 체크포인팅 시간이 초과되었습니다.
- Apache Beam 애플리케이션의 체크포인트 실패
- 역압
- 데이터 편중
- 상태 편중
- 여러 지역의 리소스와의 통합

문제 해결 도구

애플리케이션 문제를 탐지하는 기본 도구는 CloudWatch 경보입니다. CloudWatch 경보를 사용하면 애플리케이션의 오류 또는 병목 상태를 나타내는 CloudWatch 지표에 대한 임계값을 설정할 수 있습니다. CloudWatch 경보에 대한 일반적인 정보는 Apache Flink용 Amazon 매니지드 서비스에서 CloudWatch 알람 사용하기 섹션을 참조하세요.

애플리케이션 문제

이 섹션에는 Managed Service for Apache Flink 애플리케이션에서 발생할 수 있는 오류 조건에 대한 솔루션이 포함되어 있습니다.

주제

- 애플리케이션이 일시적 상태에서 정체
- 스냅샷 생성 실패
- VPC에서 리소스에 액세스 불가능
- Amazon S3 버킷에 작성할 때 데이터의 손실
- 애플리케이션이 가동 상태이지만 데이터를 처리하지 않고 있습니다.
- 스냅샷, 애플리케이션 업데이트 또는 애플리케이션 중지 오류: 잘못된 애플리케이션 구성 예외
- java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/security/cacerts

애플리케이션이 일시적 상태에서 정체

애플리케이션이 일시적 상태 (`STARTING`, `UPDATING``STOPPING`, 또는`AUTOSCALING`)에 정체되어 있는 경우 `Force` 파라미터가 `true`로 설정된 상태에서 StopApplication 작업을 사용하여 애플리케이션을 중지할 수 있습니다. `DELETING` 상태에서는 애플리케이션을 강제로 중지할 수 없습니다. 또는 애플리케이션이 `UPDATING` 또는 `AUTOSCALING` 상태인 경우 이전 실행 버전으로 회귀할 수 있습니다. 애

플리케이션을 회귀시키면 마지막으로 성공한 스냅샷의 상태 데이터가 로드됩니다. 애플리케이션에 스냅샷이 없는 경우 Managed Service for Apache Flink는 회귀 요청을 거부합니다. 애플리케이션 회귀에 대한 자세한 설명은 RollbackApplication을 참조하십시오.

> **Note**
>
> 애플리케이션을 강제로 중지하면 데이터가 손실되거나 중복될 수 있습니다. 애플리케이션 재시작 시 데이터 손실이나 데이터 중복 처리를 방지하려면 애플리케이션의 스냅샷을 자주 생성하는 것이 좋습니다.

애플리케이션이 정체되는 사유에는 다음이 포함됩니다:

- 애플리케이션 상태가 너무 큼: 애플리케이션 상태가 너무 크거나 너무 지속적이면 체크포인트 또는 스냅샷 작업 중에 애플리케이션이 정체될 수 있습니다. 애플리케이션의 지표 `lastCheckpointDuration` 및 `lastCheckpointSize`의 값이 꾸준히 증가하거나 비정상적으로 높은지 확인하십시오.
- 애플리케이션 코드가 너무 큼: 애플리케이션 JAR 파일이 512MB보다 작은지 확인하십시오. 512MB보다 큰 JAR 파일은 지원되지 않습니다.
- 애플리케이션 스냅샷 생성 실패: Managed Service for Apache Flink는 `UpdateApplication` 또는 `StopApplication` 요청 중에 애플리케이션의 스냅샷을 찍습니다. 그런 다음 이 서비스는 이 스냅샷 상태를 사용하고 업데이트된 애플리케이션 구성을 사용하여 애플리케이션을 복원하여 정확히 1회에 한하여 시맨틱의 처리를 제공합니다. 자동 스냅샷 생성이 실패할 경우, 다음의 스냅샷 생성 실패을 참조하십시오.
- 스냅샷 실패로부터 복구: 애플리케이션 업데이트에서 연산자를 제거하거나 변경한 후 스냅샷에서 복구하려고 시도하면 스냅샷에 누락된 연산자에 대한 상태 데이터가 포함되어 있으면 기본적으로 복구가 실패합니다. 또한 이 애플리케이션은 `STOPPED` 또는 `UPDATING` 상태 중 하나로 고정됩니다. 이 동작을 변경하여 복구에 성공하려면 애플리케이션 FlinkRun Configuration의 AllowNonRestoredState 파라미터를 `true`으로 변경하십시오. 이렇게 하면 새 프로그램에 매핑할 수 없는 상태 데이터를 재개 작업에서 건너뛰도록 할 수 있습니다.
- 애플리케이션 초기화 시간이 더 오래 걸림: Managed Service for Apache Flink는 Flink 작업이 시작되기를 기다리는 동안 내부 제한 시간 5분(소프트 설정)을 사용합니다. 이 제한 시간 내에 작업을 시작하지 못하면 다음과 같은 CloudWatch 로그가 표시됩니다:

```
Flink job did not start within a total timeout of 5 minutes for application: %s under
 account: %s
```

위와 같은 오류가 발생하면 Flink 작업의 `main` 메서드에 정의된 작업이 5분 이상 걸리고, 이로 인해 Managed Service for Apache Flink 끝에서 Flink 작업 생성 시간이 초과되었음을 의미합니다. Flink JobManager 로그와 애플리케이션 코드를 확인하여 `main` 메서드에서 이러한 지연이 예상되는지 확인하는 것이 좋습니다. 그렇지 않은 경우 5분 이내에 완료되도록 문제를 해결하기 위한 조치를 취해야 합니다.

`ListApplications` 또는 `DescribeApplication` 조치를 사용하여 귀하의 신청 상태를 확인할 수 있습니다.

스냅샷 생성 실패

Managed Service for Apache Flink는 다음과 같은 상황에서는 스냅샷을 찍을 수 없습니다:

- 애플리케이션이 스냅샷 한도를 초과했습니다. 스냅샷 한도는 1,000입니다. 자세한 설명은 스냅샷 섹션을 참조하세요.
- 애플리케이션에는 소스 또는 싱크에 액세스할 수 있는 권한이 없습니다.
- 애플리케이션 코드가 제대로 작동하지 않습니다.
- 애플리케이션에 다른 구성 문제가 발생했습니다.

애플리케이션 업데이트 중에 스냅샷을 찍는 동안 또는 애플리케이션을 중지하는 중에 예외가 발생하는 경우 애플리케이션의 `ApplicationSnapshotConfiguration`의 `SnapshotsEnabled` 속성을 `false`로 설정하고 요청을 재시도하십시오.

애플리케이션 연산자가 제대로 프로비저닝되지 않은 경우 스냅샷이 실패할 수 있습니다. 연산자 성능 조정에 대한 자세한 설명은 연산자 스케일링을 참조하십시오.

애플리케이션이 정상 상태로 돌아오면 애플리케이션 `SnapshotsEnabled` 속성을 `true`로 설정하는 것이 좋습니다.

VPC에서 리소스에 액세스 불가능

애플리케이션이 Amazon VPC에서 실행되는 VPC를 사용하는 경우, 다음을 수행하여 애플리케이션이 해당 리소스에 액세스할 수 있는지 확인하십시오:

- CloudWatch 로그에서 다음 오류를 확인하십시오. 이 오류는 귀하의 애플리케이션이 VPC의 리소스에 액세스할 수 없음을 나타냅니다.

```
org.apache.kafka.common.errors.TimeoutException: Failed to update metadata after
 60000 ms.
```

이 오류가 표시되면 경로 표가 올바르게 설정되어 있고 커넥터에 올바른 연결 설정이 있는지 확인하십시오.

CloudWatch 로그 설정 및 분석에 대한 자세한 설명은 로깅 및 모니터링 섹션을 참조하십시오.

Amazon S3 버킷에 작성할 때 데이터의 손실

Apache Flink 버전 1.6.2를 사용하여 Amazon S3 버킷에 출력을 쓸 때 일부 데이터 손실이 발생할 수 있습니다. Amazon S3를 사용하여 직접 출력할 때는 지원되는 최신 버전의 Apache Flink를 사용하는 것이 좋습니다. Apache Flink 1.6.2를 사용하여 Amazon S3 버킷에 쓰려면 Kinesis Data Firehose를 사용하는 것이 좋습니다. Managed Service for Apache Flink와 함께 Kinesis Data Firehose를 사용하는 방법에 대한 자세한 설명은 Kinesis Data Firehose Sink을 참조하십시오.

애플리케이션이 가동 상태이지만 데이터를 처리하지 않고 있습니다.

`ListApplications` 또는 `DescribeApplication` 작업을 사용하여 애플리케이션 상태를 확인할 수 있습니다. 애플리케이션이 `RUNNING` 상태에 들어가지만 싱크에 데이터를 쓰지 않는 경우 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가하여 문제를 해결할 수 있습니다. 자세한 설명은 응용 프로그램 CloudWatch 로깅 옵션 사용 섹션을 참조하세요. 로그 스트림에는 애플리케이션 문제를 해결하는 데 사용할 수 있는 메시지가 포함됩니다.

스냅샷, 애플리케이션 업데이트 또는 애플리케이션 중지 오류: 잘못된 애플리케이션 구성 예외

스냅샷 작업 중에 또는 스냅샷을 생성하는 작업 (예: 애플리케이션 업데이트 또는 중지) 중에 다음과 유사한 오류가 발생할 수 있습니다.

```
An error occurred (InvalidApplicationConfigurationException) when calling the
 UpdateApplication operation:

Failed to take snapshot for the application xxxx at this moment. The application is
 currently experiencing downtime.
Please check the application's CloudWatch metrics or CloudWatch logs for any possible
 errors and retry the request.
You can also retry the request after disabling the snapshots in the Managed Service for
 Apache Flink console or by updating
```

```
the ApplicationSnapshotConfiguration through the AWS SDK
```

이 오류는 애플리케이션에서 스냅샷을 생성할 수 없을 때 발생합니다.

스냅샷 작업 또는 스냅샷을 생성하는 작업 중에 이 오류가 발생하는 경우 다음을 수행하십시오.

- 애플리케이션의 스냅샷을 비활성화하십시오. Managed Service for Apache Flink 콘솔에서 또는 UpdateApplication 작업의 `SnapshotsEnabledUpdate` 파라미터를 사용하여 이 작업을 수행할 수 있습니다.
- 스냅샷을 생성할 수 없는 이유를 조사하세요. 자세한 설명은 애플리케이션이 일시적 상태에서 정체 섹션을 참조하세요.
- 애플리케이션이 정상 상태로 돌아오면 스냅샷을 다시 활성화하십시오.

java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/security/cacerts

SSL 신뢰 저장소의 위치는 이전 배포에서 업데이트되었습니다. `ssl.truststore.location` 파라미터에 대해 다음 값을 대신 사용하십시오:

```
/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
```

애플리케이션 다시 시작 중

사용하시는 애플리케이션이 정상이 아닌 경우 Apache Flink 작업이 계속 실패하고 다시 시작됩니다. 이 섹션에서는 이러한 상태에 대한 증상과 문제 해결 조치들을 설명합니다.

증상

이 상태는 다음과 같은 증상이 있을 수 있습니다:

- `FullRestarts` 지표가 0이 아님. 이 지표는 측정 단위는 귀하가 애플리케이션을 시작한 이후 해당 애플리케이션의 작업이 다시 시작된 횟수를 나타냅니다.
- `Downtime` 지표가 0이 아님. 이 지표는 애플리케이션이 `FAILING` 또는 `RESTARTING` 상태에 있는 밀리초의 수치를 나타냅니다.
- 애플리케이션 로그는 `RESTARTING` 또는 `FAILED`로의 상태 변경을 포함합니다. 다음과 같은 CloudWatch Logs Insights 쿼리를 사용하여 애플리케이션 로그에서 이러한 상태 변경을 검색할 수 있습니다: 오류 분석: 애플리케이션 작업 관련 실패.

원인 및 해결 방법

다음과 같은 상태 하에서 애플리케이션은 불안정해지고 반복적으로 다시 시작될 수 있습니다:

- 연산자가 예외를 발생시키는 경우: 귀하의 애플리케이션에서 연산자의 예외가 처리되지 않으면 연산자가 오류를 처리할 수 없다고 해석하여 애플리케이션이 페일오버됩니다. “정확히 한 번” 처리하라는 의미를 유지하기 위해 애플리케이션이 최근의 체크포인트에서 다시 시작됩니다. 따라서 이러한 재시작 기간 동안에는 `Downtime`의 값이 0이 아닙니다. 이러한 일이 발생하지 않도록 하려면 애플리케이션 코드에서 재시도 가능한 예외를 모두 처리하는 것이 좋습니다.

  애플리케이션 로그를 검색하여 애플리케이션 상태를 `RUNNING`에서 `FAILED`로 변경할 것을 요청함으로써 이 상태의 원인을 조사할 수 있습니다. 자세한 설명은 the section called “오류 분석: 애플리케이션 작업 관련 실패” 섹션을 참조하세요.
- Kinesis Data Streams가 제대로 프로비저닝되지 않음: 애플리케이션의 소스 또는 싱크가 Kinesis Data Streams인 경우, 스트림의 지표에 `ReadProvisionedThroughputExceeded` 또는 `WriteProvisionedThroughputExceeded` 오류가 있는지 확인하십시오.

  이러한 오류가 표시되면 스트림의 샤드 수를 늘려 Kinesis 스트림의 가용 처리량을 늘릴 수 있습니다. 자세한 설명은 Kinesis Data Streams에서 열린 샤드 수를 변경하려면 어떻게 해야 하는가?를 참조하십시오.
- 다른 소스 또는 싱크가 제대로 프로비저닝되거나 사용 가능하지 않음: 애플리케이션이 소스 및 싱크를 올바르게 프로비저닝하고 있는지 확인하세요. 애플리케이션에서 사용되는 소스 또는 싱크(예: 다른 AWS 서비스, 외부 소스 또는 목적지)가 제대로 프로비저닝되었는지, 읽기 또는 쓰기 제한이 발생하지 않는지, 또는 주기적으로 사용 불능상태인지 확인하세요.

  종속 서비스에서 처리량 관련 문제가 발생하는 경우 해당 서비스에 사용할 수 있는 리소스를 늘리거나 오류 또는 사용 불능의 원인을 조사하십시오.
- 연산자가 제대로 프로비저닝되지 않은 경우: 애플리케이션의 연산자 중 하나에 대한 스레드의 워크로드가 제대로 분배되지 않으면 연산자에 과부하가 걸리고 애플리케이션이 와해될 수 있습니다. 연산자 병렬성 조정에 대한 자세한 설명은 연산자 스케일링을 적절하게 관리을 참조하십시오.
- DaemonException에서의 애플리케이션 실패: 이 오류는 Apache Flink 1.11 이전 버전을 사용하는 경우 귀하의 애플리케이션 로그에 나타납니다. KPL 0.14 또는 그 이후 버전을 사용하려면 Apache Flink를 이후 버전으로 업그레이드해야 할 수 있습니다.
- TimeoutException, FlinkException 또는 RemoteTransportException에서 애플리케이션이 실패함: 작업 관리자가 충돌하는 경우 이러한 오류가 애플리케이션 로그에 나타날 것입니다. 애플리케이션에 과부하가 걸리면 작업 관리자가 CPU 또는 메모리 리소스 부족을 경험하여 실패가 발생할 수 있습니다.

이러한 오류는 아마 다음과 같을 것입니다:

- `java.util.concurrent.TimeoutException: The heartbeat of JobManager with id xxx timed out`
- `org.apache.flink.util.FlinkException: The assigned slot xxx was removed`
- `org.apache.flink.runtime.io.network.netty.exception.RemoteTransportException: Connection unexpectedly closed by remote task manager`

이 문제를 해결하려면 다음과 같이 하세요:

- CloudWatch 측정치를 확인하여 CPU 또는 메모리 사용량이 비정상적으로 급증했는지 확인하십시오.
- 애플리케이션에서 처리량 문제를 확인하십시오. 자세한 설명은 성능 문제 해결 섹션을 참조하세요.
- 애플리케이션 로그에서 애플리케이션 코드에서 발생하는 처리되지 않은 예외가 있는지 확인하십시오.

- JaxBannotationModule에서 찾을 수 없음 오류로 인해 애플리케이션 실패. 이 오류는 애플리케이션에서 Apache Beam을 사용하지만 의존성 또는 의존성 버전이 올바르지 않은 경우 발생합니다. Apache Beam을 사용하는 Managed Service for Apache Flink 애플리케이션은 다음 버전의 의존성을 사용해야 합니다:

```
<jackson.version>2.10.2</jackson.version>
...
<dependency>
    <groupId>com.fasterxml.jackson.module</groupId>
    <artifactId>jackson-module-jaxb-annotations</artifactId>
    <version>2.10.2</version>
</dependency>
```

정확한 버전의 `jackson-module-jaxb-annotations`를 명시적 의존성으로 제공하지 않으면 애플리케이션이 환경 의존성에서 해당 버전을 로드하고, 버전이 일치하지 않으므로 애플리케이션이 런타임에 충돌합니다.

Managed Service for Apache Flink와 Apache Beam을 사용하는 방법에 대한 자세한 설명은 Managed Service for Apache Flink와 함께 CloudFormation 사용하기 섹션을 참조하세요.

- Java.io.IOException에서 애플리케이션 실패: 네트워크 버퍼 수 부족.

이는 애플리케이션에 네트워크 버퍼에 할당된 메모리가 충분하지 않을 때 발생합니다. 네트워크 버퍼는 하위 작업 간의 통신을 용이하게 합니다. 또한 네트워크를 통해 전송하기 전에 레코드를 저장하고, 수신 데이터를 분해하여 기록하고 이를 하위 작업에 전달하기 전에 저장하는 데 사용됩니다. 필요한 네트워크 버퍼 수는 귀하의 작업 그래프의 병렬성과 복잡성에 따라 조정됩니다. 이 문제를 완화할 수 있는 여러 가지 방법이 있습니다:

- 귀하는 하위 작업 및 네트워크 버퍼별로 더 많은 메모리가 할당되도록 더 낮은 값의`parallelismPerKpu` 을 구성할 수 있습니다. `parallelismPerKpu`을 낮추면 KPU가 증가하여 비용 또한 증가한다는 점에 유의하세요. 이를 방지하려면 병렬성을 같은 배수로 낮춰 KPU의 양을 동일하게 유지할 수 있습니다.
- 연산자 수를 줄이거나 서로 연결하여 필요한 버퍼 수를 줄이면 귀하의 작업 그래프를 단순화할 수 있습니다.
- 또는 https://aws.amazon.com/premiumsupport/ 에 문의하여 고객의 맞춤 네트워크 버퍼 구성에 대해 문의할 수 있습니다.

처리량이 너무 느림

애플리케이션이 들어오는 스트리밍 데이터를 충분히 빠르게 처리하지 않으면 성능이 저하되고 불안정해집니다. 이 섹션에서는 이러한 상태에 대한 증상과 문제 해결 조치들을 설명합니다.

증상

이 상태는 다음과 같은 증상이 있을 수 있습니다:

- 애플리케이션의 데이터 소스가 Kinesis 스트림인 경우 스트림의 `millisbehindLatest` 지표는 계속 증가합니다.
- 애플리케이션의 데이터 소스가 Amazon MSK 클러스터인 경우 클러스터의 소비자 지연 지표는 계속 증가합니다. 자세한 설명은 Amazon MSK 개발자 가이드의 소비자 지연 모니터링을 참조하십시오.
- 애플리케이션의 데이터 소스가 다른 서비스 또는 소스인 경우 사용 가능한 소비자 지연 지표 또는 사용 가능한 데이터를 확인하십시오.

원인 및 해결 방법

애플리케이션 처리 속도가 느려지는 원인은 여러 가지가 있을 수 있습니다. 애플리케이션이 입력을 따라가지 못하는 경우 다음을 확인하세요:

- 처리량 지연이 급증하다가 점점 줄어드는 경우 애플리케이션이 다시 시작되고 있는지 확인하세요. 애플리케이션이 다시 시작되는 동안 입력 처리가 중지되어 지연이 급증합니다. 애플리케이션 보고서에 대한 자세한 설명은 애플리케이션 다시 시작 중 섹션을 참조하십시오.
- 처리량 지연이 일정하다면 애플리케이션이 성능을 위해 최적화되었는지 확인하세요. 애플리케이션 성능 최적화에 대한 자세한 설명은 성능 문제 해결을 참조하십시오.
- 처리량 지연이 급증하지 않고 계속 증가하고 있으며 애플리케이션이 성능에 맞게 최적화되어 있다면 애플리케이션 리소스를 늘려야 합니다. 애플리케이션 리소스 증가에 대한 자세한 설명은 스케일링을 참조하십시오.
- 애플리케이션이 다른 지역의 Kafka 클러스터에서 읽고, 높은 소비자 지체에도 불구하고 `FlinkKafkaConsumer` 또는 `KafkaSource`가 대부분 유휴 상태(`idleTimeMsPerSecond` 높음 또는 `CPUUtilization` 낮음)인 경우, `receive.buffer.byte`의 값을 높일 수 있습니다(예: 2097152). 자세한 설명은 맞춤 MSK 구성에서 높은 잠복 환경 섹션을 참조하십시오.

애플리케이션 소스의 느린 처리량 또는 소비자 지연 증가에 대한 문제 해결 단계는 성능 문제 해결을 참조하십시오.

영구 지속적 성장

애플리케이션이 이미 지난 정보를 제대로 처리하지 않으면 낡은 정보가 계속 누적되어 애플리케이션 성능이나 안정성 문제가 발생할 수 있습니다. 이 섹션에서는 이러한 상태에 대한 증상과 문제 해결 조치들을 설명합니다.

증상

이 상태는 다음과 같은 증상이 있을 수 있습니다:

- `lastCheckpointDuration` 지표가 점차 증가하거나 급증함.
- `lastCheckpointSize` 지표가 점차 증가하거나 급증함.

원인 및 해결 방법

다음과 같은 조건에서는 애플리케이션이 상태 데이터를 축적할 수 있습니다:

- 애플리케이션이 상태 데이터를 필요 기간보다 더 오래 보존하고 있음.
- 애플리케이션이 윈도우 쿼리를 너무 오랜 시간 동안 사용함.
- 상태 데이터에 TTL을 설정하지 않았음. 자세한 설명은 Apache Flink 설명서의 상태 존속 시간(TTL)을 참조하십시오.

- 귀하는 지금 Apache Beam 버전 2.25.0 이상을 사용하는 애플리케이션을 실행하고 있습니다. 주요 실험 및 값 `use_deprecated_read`로 BeamApplicationProperties를 확장함으로써 새 버전의 읽기 변환을 옵트아웃할 수 있습니다. 자세한 설명은 Apache Beam 설명서를 참조하십시오.

간혹 애플리케이션은 상태 크기가 계속 커지는 상황에 직면하고 있는데, 이는 장기적으로 지속가능하지 않습니다(결국 Flink 애플리케이션은 무한정 실행됩니다). 때로는 데이터를 상태 그대로 저장하고 오래된 정보를 폐기하지 않는 애플리케이션이 원인일 수 있습니다. 하지만 가끔은 Flink가 제공할 수 있는 기능에 대해 지나치게 기대하는 경우가 있습니다. 애플리케이션은 며칠 또는 몇 주에 걸친 긴 단위 시간에 걸쳐 집계를 사용할 수 있습니다. 증분 집계를 허용하는 AggregateFunctions를 사용하지 않는 한 Flink는 전체 창의 사건들을 상태 내에 유지해야 합니다.

또한 프로세스 함수를 사용하여 맞춤 연산자를 구현하는 경우 애플리케이션은 더 이상 비즈니스 로직에 필요하지 않은 데이터를 상태로부터 제거해야 합니다. 이 경우, 상태 존속 시간을 사용하여 처리 시간에 따라 오래된 데이터를 자동으로 퇴출할 수 있습니다. Managed Service for Apache Flink는 증분 체크포인트를 사용하므로 상태 존속 시간은 RocksDB 압축에 근거합니다. 압축 작업이 수행된 후에야 실제 상태 크기 감소(체크포인트 크기로 표시)를 관찰할 수 있습니다. 특히 체크포인트 크기가 200MB 미만인 경우 상태 만료로 인해 체크포인트 크기가 줄어들 가능성은 거의 없습니다. 하지만 세이브 포인트는 이전 데이터가 포함되지 않은 상태의 깨끗한 사본에 근거하므로 Managed Service for Apache Flink에서 스냅샷을 트리거하여 이미 지난 상태를 강제로 제거할 수 있습니다.

디버깅을 위해서는 증분 체크포인트를 비활성화하여 체크포인트 크기가 실제로 줄어들거나 안정화되는지 더 빨리 확인하고 RockSBS에서 압축으로 인한 영향을 방지하는 것이 좋습니다. 하지만 이를 위해서는 서비스 팀에 대한 티켓이 필요합니다.

I/O 바운드 연산자

데이터 경로에서 외부 시스템에 대한 종속성을 피하는 것이 가장 좋습니다. 개별 이벤트를 보강하기 위해 외부 시스템을 쿼리하는 것보다 참조 데이터 세트를 상태로 유지하는 것이 훨씬 더 효과적인 경우가 많습니다. 그러나 Amazon Sagemaker에서 호스팅되는 기계 학습 모델로 이벤트를 강화하려는 경우처럼의 상태로 쉽게 이동할 수 없는 종속성이 있는 경우가 있습니다.

네트워크를 통해 외부 시스템과 상호 작용하는 연산자는 병목 현상이 발생하여 역압을 유발할 수 있습니다. 개별 호출의 대기 시간을 줄이고 전체 애플리케이션의 속도 저하를 방지하려면 AsyncIO를 사용하여 기능을 구현하는 것이 좋습니다.

또한 I/O 바운드 연산자가 있는 애플리케이션의 경우 Managed Service for Apache Flink 애플리케이션의 ParallelismperKPU 설정을 높이는 것도 합리적일 수 있습니다. 이 구성은 애플리케이션이 Kinesis 처리 단위(KPU)에 따라 수행할 수 있는 병렬 하위 작업 수를 설명합니다. 값을 기본값 1에서 4로 늘리

면 애플리케이션은 동일한 리소스를 활용하고 비용은 같지만 병렬 처리 수를 4배까지 확장할 수 있습니다. 이는 I/O 바인딩된 애플리케이션에서는 잘 작동하지만 I/O 바인딩되지 않은 애플리케이션에는 추가 오버헤드가 발생합니다.

Kinesis 데이터 스트림의 업스트림 또는 소스 조절

증상: 애플리케이션이 업스트림 소스 Kinesis 데이터 스트림에서 `LimitExceededExceptions`을 발견합니다.

잠재적 원인: Apache Flink 라이브러리 Kinesis 커넥터의 기본 설정이 Kinesis 데이터 스트림 소스에서 읽도록 설정되어 있으며, `GetRecords` 호출당 인출되는 최대 레코드 수에 대해 매우 엄격한 기본 설정이 사용됩니다. 샤드당 레코드 한도는 레코드 1,000개에 불과하지만 Apache Flink는 기본적으로 GetRecords 호출당 10,000개의 레코드를 가져오도록 구성되어 있습니다(이 호출은 기본적으로 200ms마다 수행됨).

이 기본 동작으로 인해 Kinesis 데이터 스트림을 사용하려고 할 때 병목 현상이 발생하여 애플리케이션 성능 및 안정성에 영향을 미칠 수 있습니다.

이는 CloudWatch `ReadProvisionedThroughputExceeded` 지표를 확인하고 이 지표가 0보다 큰 장기간 또는 지속적인 기간을 확인함으로써 확인할 수 있습니다.

또한 고객은 Managed Service for Apache Flink 애플리케이션에 대한 CloudWatch 로그에서 계속되는 `LimitExceededException` 오류를 확인하여 이를 확인할 수 있습니다.

해결 방법: 고객은 다음 두 가지 중 하나를 수행하여 이 시나리오를 해결할 수 있습니다:

- GetRecords 호출당 가져오는 레코드 수의 기본 한도를 낮춥니다.
- 고객은 Managed Service for Apache Flink 애플리케이션에서 적응형 읽기를 활성화할 수 있습니다. 적응형 읽기 기능에 대한 자세한 설명은 SHARD_USE_ADAPTIVE_READS를 참조하십시오.

체크포인트

체크포인트는 애플리케이션의 상태가 내결함성을 갖는지 확인하는 Flink의 메커니즘입니다. 이 메커니즘을 통해 Flink는 작업이 실패할 경우 연산자의 상태를 복구할 수 있으며 애플리케이션에는 오류 없는 실행과 동일한 의미를 부여합니다. Managed Service for Apache Flink를 사용하면 애플리케이션의 상태가 디스크에 작동 상태를 유지하는 내장형 키/값 저장소인 RocksDB에 저장됩니다. 체크포인트를 가져오면 상태가 Amazon S3에도 업로드되므로 디스크가 손실되더라도 체크포인트를 사용하여 애플리케이션 상태를 복원할 수 있습니다.

자세한 설명은 상태 스냅샷은 어떻게 작동하는가? 섹션을 참조하세요.

체크포인트 단계

Flink의 체크포인트 연산자 하위 태스크에는 5대 단계가 있습니다:

- 대기 [시작 지연] – Flink는 스트림에 삽입되는 체크포인트 장벽을 사용하므로 이 단계의 시간은 연산자가 체크포인트 장벽에 도달할 때까지 기다리는 시간입니다.
- 정렬 [정렬 지속 시간] – 이 단계에서 하위 작업은 하나의 장벽에 도달했지만 다른 입력 스트림으로 들어오는 장벽을 기다리고 있습니다.
- 동기화 체크포인트 [동기화 지속 시간] – 이 단계는 하위 작업이 실제로 연산자의 상태를 스냅샷하고 하위 작업의 다른 모든 활동을 차단하는 단계입니다.
- 비동기 체크포인트 [비동기 지속 시간] – 이 단계의 대부분은 상태를 Amazon S3에 업로드하는 하위 작업입니다. 이 단계에서는 하위 작업이 더 이상 차단되지 않고 레코드를 처리할 수 있습니다.
- 승인 – 일반적으로 짧은 단계이며 단순히 하위 작업이 JobManager에 승인을 보내고 위탁 메시지(예컨대, Kafka 싱크와 함께)를 수행하는 단계입니다.

이러한 각 단계(승인 단계 제외)는 Flink WebUI에서 사용할 수 있는 체크포인트의 기간 지표에 매핑되므로 긴 체크포인트의 원인을 파악하는 데 도움이 될 수 있습니다.

체크포인트에서 사용할 수 있는 각 지표의 정확한 정의를 보려면 이력 탭으로 이동하십시오.

조사 중

긴 체크포인트 기간을 조사할 때 가장 중요한 것은 체크포인트의 병목 현상, 즉 어떤 연산자와 하위 작업이 체크포인트에 가장 오래 걸리고 해당 하위 작업의 어느 단계에 오랜 시간이 걸리는지입니다. 이는 작업 체크포인트 태스크에서 Flink WebUI를 사용하여 확인할 수 있습니다. Flink의 웹 인터페이스는 체크포인트 문제를 조사하는 데 도움이 되는 데이터와 정보를 제공합니다. 자세한 설명은 체크포인트 모니터링을 참조하십시오.

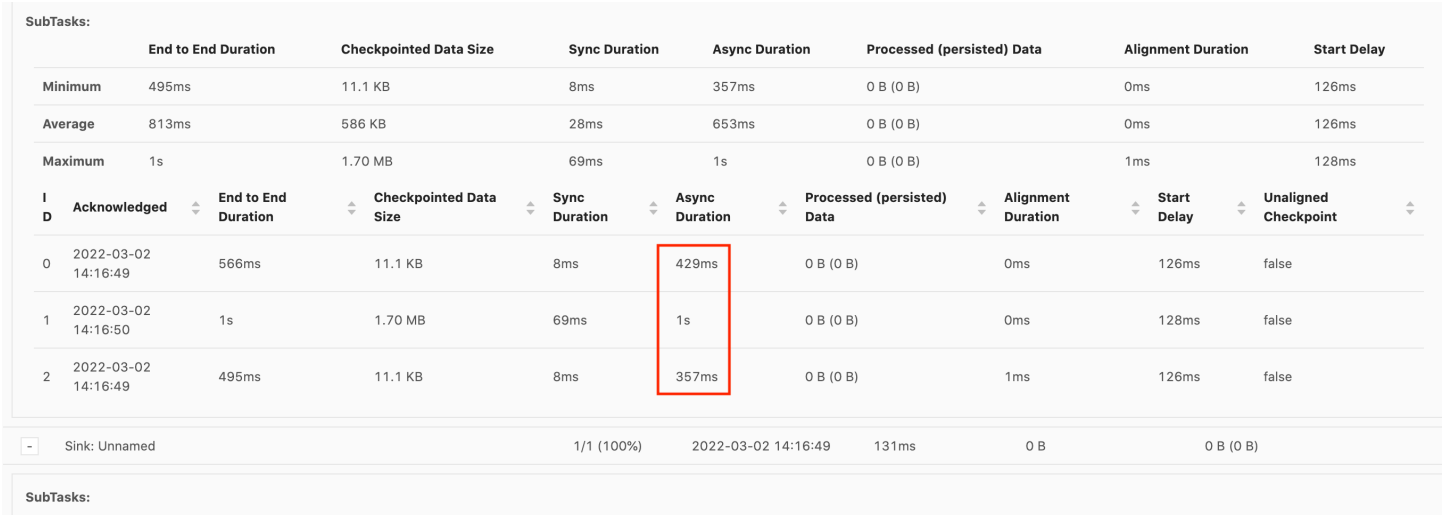
가장 먼저 살펴볼 것은 Job 그래프에서 각 연산자의 단대단 지속시간입니다. 이를 통해 어떤 연산자가 체크포인트에 걸리는 시간이 소요되며 추가 조사가 필요한지 확인할 수 있습니다. Flink 설명서에 따르면 지속시간의 정의는 다음과 같습니다:

트리거 타임스탬프부터 가장 최근 승인까지 남은 시간(또는 아직 승인이 수신되지 않은 경우 해당 없음). 전체 체크포인트의 단대단 지속 시간은 체크포인트를 승인하는 마지막 하위 작업에 의해 결정됩니다. 이 시간은 일반적으로 단일 하위 작업에서 상태를 실제로 체크포인팅하는 데 필요한 시간보다 큽니다.

체크포인트를 위한 다른 지속시간은 또한 그 시간이 어디에 사용되는지에 대한 보다 세밀한 정보를 제공합니다.

동기화 지속 시간이 길면 스냅샷 중에 문제가 발생한 것입니다. 이 단계에서는 SnapshotState 인터페이스를 구현하는 클래스를 얻기 위해 `snapshotState()`을 호출합니다. 이것은 사용자 코드일 수 있으므로 스레드 덤프가 이 문제를 조사하는 데 유용할 수 있습니다.

비동기 지속 시간이 길면 상태를 Amazon S3에 업로드하는 데 많은 시간이 소요되고 있음을 알 수 있습니다. 이는 상태가 크거나 업로드되는 상태 파일이 많은 경우 발생할 수 있습니다. 이런 경우에는 애플리케이션이 상태를 어떻게 사용하는지 조사하고 가능한 경우 Flink 네이티브 데이터 구조가 사용되고 있는지 확인하는 것이 좋습니다 (관건 상태 사용). Managed Service for Apache Flink는 Amazon S3 호출 횟수를 최소화하는 방식으로 Flink를 구성하여 호출이 너무 오래 걸리지 않도록 합니다. 다음은 연산자의 체크포인트 통계 예입니다. 이전 연산자 체크포인트 통계에 비해 비동기 지속 시간이 상대적으로 길다는 것을 알 수 있습니다.



SubTasks:

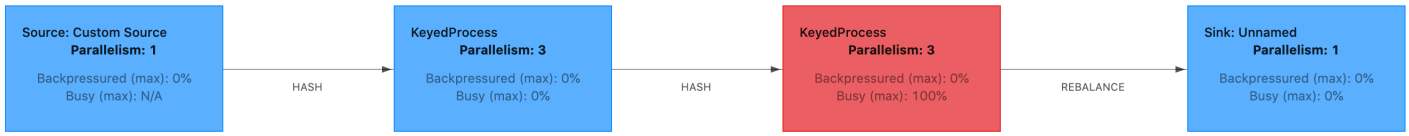
	End to End Duration	Checkpointed Data Size	Sync Duration	Async Duration	Processed (persisted) Data	Alignment Duration	Start Delay
Minimum	495ms	11.1 KB	8ms	357ms	0 B (0 B)	0ms	126ms
Average	813ms	586 KB	28ms	653ms	0 B (0 B)	0ms	126ms
Maximum	1s	1.70 MB	69ms	1s	0 B (0 B)	1ms	128ms

ID	Acknowledged	End to End Duration	Checkpointed Data Size	Sync Duration	Async Duration	Processed (persisted) Data	Alignment Duration	Start Delay	Unaligned Checkpoint
0	2022-03-02 14:16:49	566ms	11.1 KB	8ms	429ms	0 B (0 B)	0ms	126ms	false
1	2022-03-02 14:16:50	1s	1.70 MB	69ms	1s	0 B (0 B)	0ms	128ms	false
2	2022-03-02 14:16:49	495ms	11.1 KB	8ms	357ms	0 B (0 B)	1ms	126ms	false

Sink: Unnamed 1/1 (100%) 2022-03-02 14:16:49 131ms 0 B 0 B (0 B)

SubTasks:

개시 지연이 높으면 대부분의 시간이 체크포인트 장벽이 연산자에게 도달하기를 기다리는 데 소비되고 있다는 것을 알 수 있습니다. 이는 애플리케이션에서 기록을 처리하는 데 시간이 오래 걸리고 있다는 것을 의미하며, 이는 작업 그래프를 통해 장벽이 느리게 흐르고 있음을 의미합니다. 이는 일반적으로 Job이 역압을 받거나 연산자가 계속 바쁠 경우에 해당합니다. 다음은 두 번째 KeyedProcess 연산자가 사용 중인 JobGraph의 예입니다.



Flink Graphs 또는 TaskManager 스레드 덤프를 사용하여 무엇이 그렇게 오래 걸리는지 조사할 수 있습니다. 병목 현상이 확인되면 Flame-graph 또는 스레드 덤프를 사용하여 더 자세히 조사할 수 있습니다.

스레드 덤프

스레드 덤프는 플레임 그래프보다 약간 낮은 수준에 있는 또 다른 디버깅 도구입니다. 스레드 덤프는 특정 시점의 모든 스레드의 실행 상태를 출력합니다. Flink는 Flink 프로세스 내 모든 스레드의 실행 상태인 JVM 스레드 덤프를 사용합니다. 스레드 상태는 스레드의 스택 트레이스와 몇 가지 추가 정보로 표시됩니다. 실제로 플레임 그래프는 여러 개의 스택 트레이스를 연속으로 빠르게 캡처하여 작성합니다. 그래프는 이러한 트레이스를 바탕으로 만든 가시화이므로 일반적인 코드 경로를 쉽게 식별할 수 있습니다.

```
"KeyedProcess (1/3)#0" prio=5 Id=1423 RUNNABLE
    at app//scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:154)
    at $line33.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:19)
    at $line33.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:14)
    at app//
org.apache.flink.streaming.api.operators.KeyedProcessOperator.processElement(KeyedProcessOperat
    at app//org.apache.flink.streaming.runtime.tasks.OneInputStreamTask
$StreamTaskNetworkOutput.emitRecord(OneInputStreamTask.java:205)
    at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(AbstractStr
    at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(AbstractStreamTas
    at app//
org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOneInputProces
```

```
    ...
```

위는 Flink UI에서 단일 스레드에 대해 가져온 스레드 덤프의 스니펫입니다. 첫 번째 줄에는 다음을 포함하여 이 스레드에 대한 몇 가지 일반 정보가 포함되어 있습니다:

- 스레드 명칭 KeyedProcess (1/3) #0
- 스레드의 우선순위 prio=5
- 고유 스레드 ID ID=1423
- 스레드 상태 실행 가능

스레드 명칭은 일반적으로 스레드의 일반적인 용도에 대한 정보를 제공합니다. 연산자 스레드는 연산자와 명칭이 같기 때문에 연산자 스레드는 그 명칭으로 식별할 수 있으며 스레드가 어떤 하위 작업과 관련되어 있는지를 나타냅니다. 예컨대, KeyedProcess (1/3)#0 스레드는 KeyedProcess 연산자 출신으로서 (3개 중) 첫 번째 하위 태스크에서 온 것입니다.

스레드는 몇 가지 상태 중 하나일 수 있습니다:

- 신규 – 스레드가 생성되었지만 아직 처리되지는 않았음.
- 실행 가능 – 스레드가 CPU에서 실행 중임.
- 차단됨 – 스레드가 다른 스레드에 의한 잠금 해제를 기다리고 있음.
- 대기 – `wait()`, `join()`, 또는 `park()` 메서드를 사용함으로써 스레드가 대기 중임.
- TIMED_WAITING – 스레드가 sleep, wait, join 또는 park 메서드를 사용하여 대기 중이지만 대기 시간은 최대로 설정되어 있음.

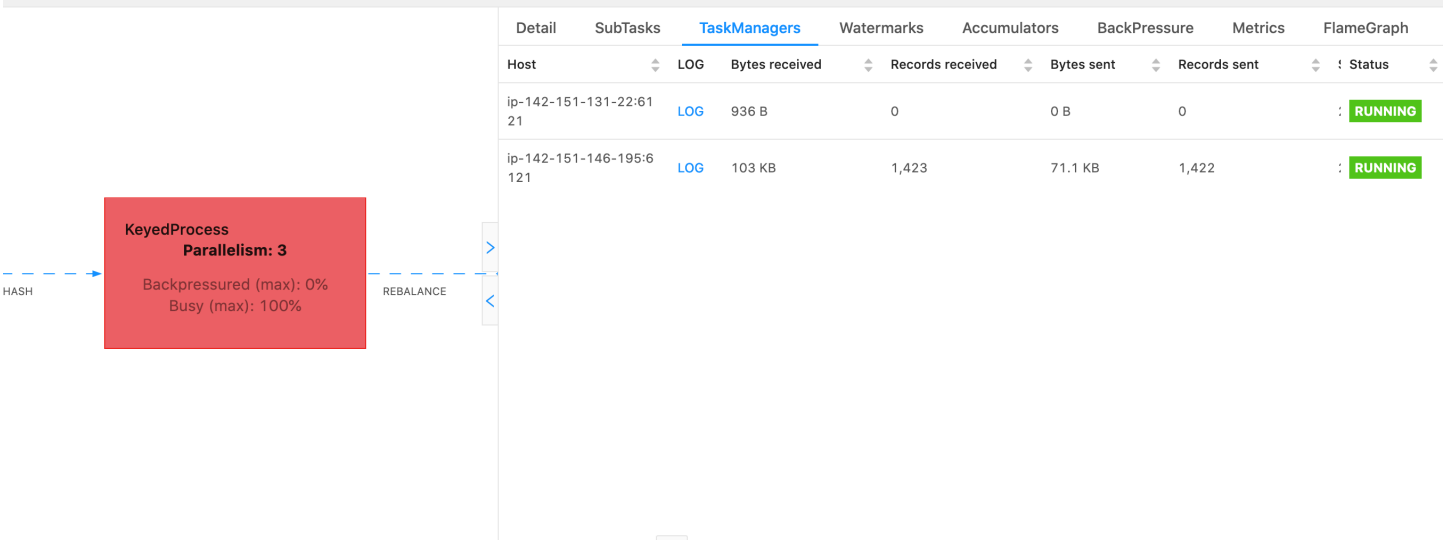
> **Note**
>
> Flink 1.13에서는 스레드 덤프에 있는 단일 스택 트레이스의 최대 깊이가 8로 제한됩니다.

> **Note**
>
> 스레드 덤프는 읽기가 어렵고 여러 샘플을 채취하여 수동으로 분석해야 하므로 Flink 애플리케이션에서 성능 문제를 디버깅할 때 최후의 수단이어야 합니다. 가능하면 플레임 그래프를 사용하는 것이 좋습니다.

Flink의 스레드 덤프

Flink에서는 Flink UI의 왼쪽 탐색 표시줄에서 작업 관리자 옵션을 선택하고 특정 작업 관리자를 선택한 다음 스레드 덤프 탭으로 이동하여 스레드 덤프를 가져올 수 있습니다. 스레드 덤프는 다운로드하거나 즐겨 사용하는 텍스트 편집기 (또는 스레드 덤프 분석기) 에 복사하거나 Flink 웹 UI의 텍스트 보기 내에서 직접 분석할 수 있습니다 (하지만 이 마지막 옵션은 약간 복잡할 수 있습니다).

스레드 덤프를 수행할 작업 관리자를 결정하려면 특정 연산자를 선택한 경우 TaskManager 탭의 스레드 덤프를 사용할 수 있습니다. 이는 연산자가 연산자의 여러 하위 작업에서 실행되고 있으며 다른 작업 관리자에서 실행될 수 있음을 나타냅니다.



덤프는 여러 스택 트레이스로 구성됩니다. 하지만 덤프를 조사할 때는 연산자와 관련된 것이 가장 중요합니다. 연산자 스레드는 연산자와 이름이 같을 뿐만 아니라 해당 스레드가 어떤 하위 작업과 관련되어 있는지 알 수 있기 때문에 쉽게 찾을 수 있습니다. 예를 들어 다음 스택 트레이스는 KeyedProcess 연산자에서 가져온 것이며 첫 번째 하위 태스크입니다.

```
"KeyedProcess (1/3)#0" prio=5 Id=595 RUNNABLE
    at app//scala.collection.immutable.Range.foreach$mVc$sp(Range.scala:155)
    at $line360.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:19)
    at $line360.$read$$iw$$iw$ExpensiveFunction.processElement(<console>:14)
    at app//
org.apache.flink.streaming.api.operators.KeyedProcessOperator.processElement(KeyedProcessOperat
    at app//org.apache.flink.streaming.runtime.tasks.OneInputStreamTask
$StreamTaskNetworkOutput.emitRecord(OneInputStreamTask.java:205)
    at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(AbstractStr
    at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(AbstractStreamTas
```

```
    at app//
org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOneInputProces
    ...
```

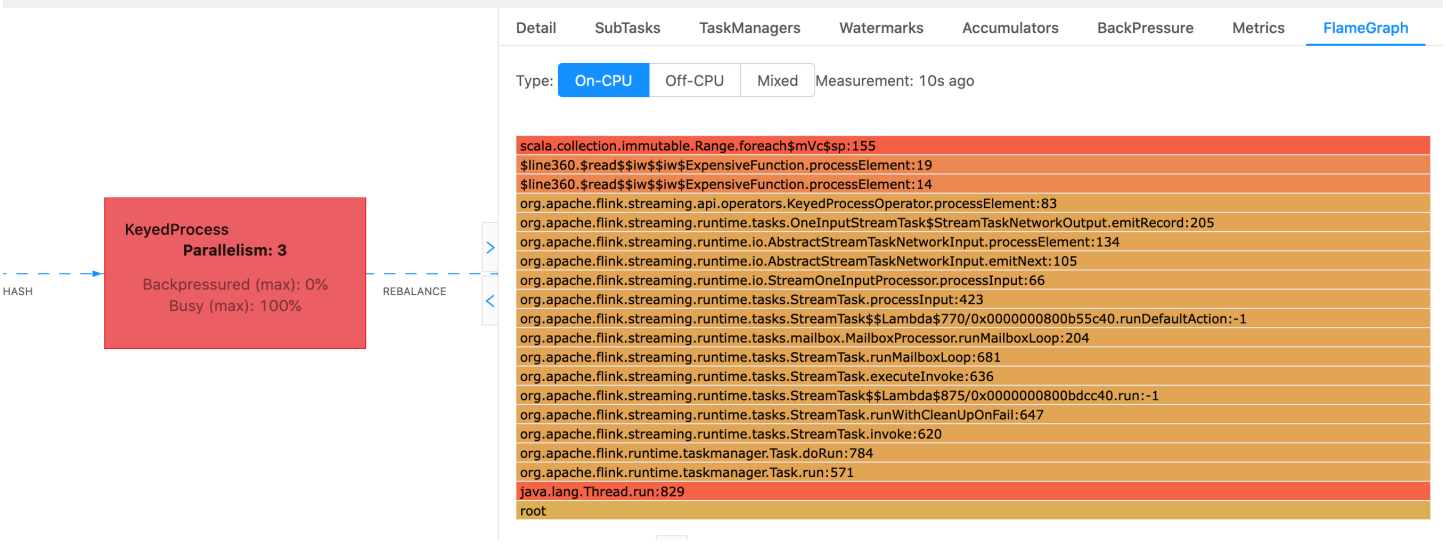
이름이 같은 연산자가 여러 개 있는 경우 혼란스러울 수 있지만 연산자의 이름을 지정하여 이러한 문제를 피할 수 있습니다. 예:

```
....
.process(new ExpensiveFunction).name("Expensive function")
```

플레임 그래프

플레임 그래프는 목적지 코드의 스택 트레이스를 가시화하는 유용한 디버깅 도구로, 이를 통해 가장 빈번한 코드 경로를 식별할 수 있습니다. 스택 트레이스를 여러 번 샘플링하여 생성합니다. 플레임 그래프의 x 축은 다양한 스택 프로필을 보여주고, y 축은 스택 깊이와 스택 트레이스에서의 호출을 보여줍니다. 플레임 그래프에서 단일 직사각형은 스택 프레임을 나타내며, 프레임 너비는 스택에서 해당 사각형이 나타나는 빈도를 나타냅니다. 플레임 그래프 및 사용 방법에 대한 자세한 설명은 플레임 그래프를 참조하십시오.

Flink에서는 연산자를 선택한 다음 FlameGraph 탭을 선택하여 웹 UI를 통해 연산자에 대한 플레임 그래프에 액세스할 수 있습니다. 샘플이 충분히 수집되면 플레임 그래프가 표시됩니다. 다음은 체크포인트에 많은 시간이 걸렸던 ProcessFunction의 플레임 그래프입니다.



이것은 매우 간단한 플레임 그래프로, 모든 CPU 시간이 ExpensiveFunction 연산자의 `processElement`내의 foreach 룩에 소비되고 있음을 보여줍니다. 또한 라인 번호도 확인할 수 있어 코드 실행이 이루어지는 위치를 파악하는 데 도움이 됩니다.

체크포인팅 시간이 초과되었습니다.

애플리케이션이 최적화되지 않았거나 제대로 프로비저닝되지 않은 경우 체크포인트가 실패할 수 있습니다. 이 섹션에서는 이러한 상태에 대한 증상과 문제 해결 조치들을 설명합니다.

증상

애플리케이션의 체크포인트에 장애가 발생하는 경우 `numberOfFailedCheckpoints` 값은 0보다 커집니다.

애플리케이션 오류와 같은 직접적인 장애나 애플리케이션 리소스 부족과 같은 일시적인 장애로 인해 체크포인트가 실패할 수 있습니다. 애플리케이션 로그와 지표에서 다음과 같은 증상이 있는지 확인하십시오.

- 코드에 오류가 있습니다.
- 애플리케이션의 종속 서비스에 액세스하는 중 오류가 발생했습니다.
- 데이터를 직렬화하는 중 오류가 발생했습니다. 기본 시리얼라이저가 애플리케이션 데이터를 직렬화할 수 없는 경우 애플리케이션이 실패합니다. 애플리케이션에서 맞춤 직렬 변환기를 사용하는 방법에 대한 자세한 설명은 Apache Flink 설명서의 맞춤 직렬 변환기를 참조하십시오.
- 메모리 부족 오류
- 다음 지표의 스큐 또는 꾸준한 증가
  - `heapMemoryUtilization`
  - `oldGenerationGCTime`
  - `oldGenerationGCCount`
  - `lastCheckpointSize`
  - `lastCheckpointDuration`

자세한 내용을 알아보려면 Apache Flink 설명서의 체크포인트 모니터링을 참조하세요.

원인 및 해결 방법

애플리케이션 로그 오류 메시지에는 직접 실패의 원인이 표시됩니다. 일시적 오류에는 다음과 같은 원인이 있을 수 있습니다:

- 애플리케이션에 KPU 프로비저닝이 충분하지 않음. 애플리케이션 프로비저닝 향상에 대한 자세한 설명은 스케일링을 참조하십시오.

- 애플리케이션 상태 크기가 너무 큼. `lastCheckpointSize`지표를 사용하여 애플리케이션 상태 크기를 모니터링할 수 있습니다.
- 애플리케이션의 상태 데이터는 여러 키 간에 불균등하게 분산됩니다. 애플리케이션에서 KeyBy 연산자를 사용하는 경우 들어오는 데이터가 여러 키 간에 균등하게 분배되는지 확인하십시오. 대부분의 데이터가 단일 키에 할당되면 병목 현상이 발생하여 오류가 발생합니다.
- 애플리케이션에서 메모리 또는 가비지 컬렉션 역압을 경험하고 있음. 애플리케이션의 값이 급증하거나 꾸준히 증가하는 값이 있는지 애플리케이션의 `heapMemoryUtilization`, `oldGenerationGCTime` 및 `oldGenerationGCCount`을 모니터링하세요.

Apache Beam 애플리케이션의 체크포인트 실패

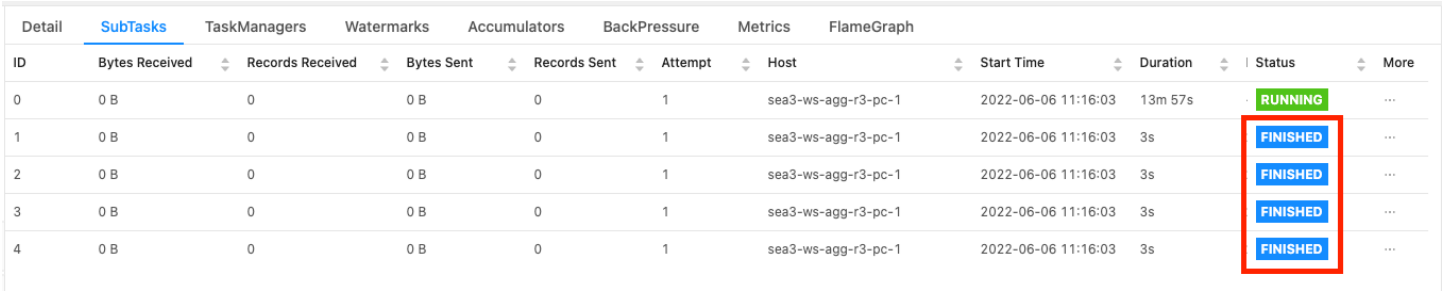
Beam 애플리케이션이 shutdownSourcesAfterIdleMs를 0ms로 설정하도록 구성된 경우 작업이 "완료" 상태이기 때문에 체크포인트가 트리거되지 않을 수 있습니다. 이 섹션에서는 그러한 상황의 증상과 해결 방법을 설명합니다.

증상

Managed Service for Apache Flink 애플리케이션 CloudWatch 로그로 이동하여 다음 로그 메시지가 기록되었는지 확인합니다. 다음 로그 메시지는 일부 작업이 완료되어 체크포인트가 트리거되지 않았음을 나타냅니다.

```
                {
                "locationInformation":
"org.apache.flink.runtime.checkpoint.CheckpointCoordinator.onTriggerFailure(CheckpointCoordina
                "logger": "org.apache.flink.runtime.checkpoint.CheckpointCoordinator",
                "message": "Failed to trigger checkpoint for job your job ID since some
tasks of job your job ID has been finished, abort the checkpoint Failure reason: Not
all required tasks are currently running.",
                "threadName": "Checkpoint Timer",
                "applicationARN": your application ARN,
                "applicationVersionId": "5",
                "messageSchemaVersion": "1",
                "messageType": "INFO"
                }
```

일부 작업이 "완료" 상태로 전환되어 더 이상 체크포인트를 지정할 수 없는 Flink 대시보드에서도 이 문제를 확인할 수 있습니다.



원인

ShutdownSourcesAfterIdleMs는 구성된 시간(밀리초) 동안 유휴 상태였던 소스를 종료하는 Beam 구성 변수입니다. 일단 소스가 종료되면 체크포인트를 더 이상 사용할 수 없습니다. 이로 인해 체크포인트 오류가 발생할 수 있습니다.

작업이 "완료" 상태로 전환되는 원인 중 하나는 ShutdownSourcesAfterIdleMs가 0ms로 설정된 경우입니다. 즉, 유휴 상태인 작업이 즉시 종료됩니다.

솔루션

작업이 즉시 "완료" 상태로 전환되지 않도록 하려면 shutdownSourcesAfterIdleMs를 Long.MAX_VALUE를 설정하십시오. 이것은 두 가지 방법으로 수행될 수 있습니다:

- 옵션 1: Managed Service for Apache Flink 애플리케이션 구성 페이지에서 빔 구성을 설정한 경우 다음과 같이 새 키 값 쌍을 추가하여 shutdpwnSourcesAfteridleMs를 설정할 수 있습니다.

- 옵션 2: JAR 파일에 빔 컨피그레이션이 설정된 경우 다음과 같이 shutdpwnSourcesAfteridleMs를 설정할 수 있습니다:

```
                    FlinkPipelineOptions options =
PipelineOptionsFactory.create().as(FlinkPipelineOptions.class); // Initialize Beam
Options object

                    options.setShutdownSourcesAfterIdleMs(Long.MAX_VALUE); // set
shutdownSourcesAfterIdleMs to Long.MAX_VALUE
                    options.setRunner(FlinkRunner.class);
```

```
                        Pipeline p = Pipeline.create(options); // attach specified
options to Beam pipeline
```

역압

Flink는 역압을 사용하여 개별 연산자의 처리 속도를 조정합니다.

연산자는 여러 가지 이유로 수신한 메시지 양을 계속 처리하는 데 어려움을 겪을 수 있습니다. 작업에는 연산자가 사용할 수 있는 것보다 많은 CPU 리소스가 필요할 수 있습니다. 연산자는 I/O 작업이 완료될 때까지 기다릴 수 있습니다. 연산자가 이벤트를 충분히 빠르게 처리하지 못하면 업스트림 연산자에 역압이 가중되어 속도가 느린 연산자에게 전달됩니다. 이로 인해 업스트림 연산자의 속도가 느려지고, 이로 인해 역압이 소스로 더 전파되어 소스가 애플리케이션의 전체 처리량에 맞게 조정되어 속도가 느려질 수 있습니다. Apache Flink™가 역압을 처리하는 방법에서 역압과 그 작동 방식에 대한 자세한 설명을 확인할 수 있습니다.

애플리케이션에서 속도가 느린 연산자를 알면 애플리케이션 성능 문제의 근본 원인을 이해하는 데 중요한 정보를 얻을 수 있습니다. 역압 정보는 Flink 대시보드를 통해 노출됩니다. 속도가 느린 연산자를 식별하려면 싱크에 가장 가까운 높은 역압 값을 가진 연산자를 찾으십시오(다음 예에서는 연산자 B). 그러면 속도 저하를 초래한 연산자가 다운스트림 연산자 중 한 명입니다(해당 예에서는 연산자 C). B는 이벤트를 더 빠르게 처리할 수 있지만 출력을 실제 느린 연산자 C에게 전달할 수 없으므로 역압을 받습니다.

```
A (backpressured 93%) -> B (backpressured 85%) -> C (backpressured 11%) -> D
 (backpressured 0%)
```

느린 연산자를 식별한 후에는 속도가 느린 이유를 이해해 보십시오. 이유는 무수히 많을 수 있으며, 무엇이 잘못되었는지 명확하지 않을 수 있으며 문제를 해결하는 데 며칠의 디버깅과 프로필링이 필요할 수 있습니다. 다음은 명확하고 일반적인 몇 가지 이유이며, 그 중 일부는 아래에 자세히 설명되어 있습니다:

- 연산자가 느린 I/O(예: 네트워크 호출)를 하고 있습니다(대신 AsyncIO를 사용할 것을 고려하세요).
- 데이터에 편중이 있고 Flink 대시보드에서 한 연산자가 다른 연산자보다 더 많은 이벤트를 수신하고 있습니다(개별 하위 작업(예: 동일한 연산자의 인스턴스)에서 들어오고 나가는 메시지 수를 보아서 확인하십시오.
- 이는 리소스 집약적인 작업입니다(데이터 왜곡이 없다면 CPU/메모리 바운드 작업을 위해 규모를 확장하거나 I/O 바운드 작업을 위해 `ParallelismPerKPU`을 늘리는 것을 고려해 보세요).

- 연산자의 광범위한 로깅(프로덕션 애플리케이션의 경우 로깅을 최소한으로 줄이거나 대신 디버그 출력을 데이터 스트림으로 보내는 방안을 고려해 보세요).

폐기 싱크를 사용한 처리량 테스트

폐기 싱크는 단순히 애플리케이션을 실행하는 동안 수신되는 모든 이벤트를 무시합니다. 즉, 싱크가 없는 애플리케이션은 실행되지 않습니다. 이는 처리량 테스트, 프로필링, 애플리케이션이 적절하게 확장되고 있는지 확인하는 데 매우 유용합니다. 또한 싱크로 인해 역압이나 애플리케이션이 발생하는지 확인하는 것도 매우 실용적인 온전성 검사입니다(하지만 많은 경우 역압 지표를 확인하는 것만으로도 더 쉽고 간단합니다).

애플리케이션의 모든 싱크를 폐기 싱크로 바꾸고 프로덕션 데이터와 유사한 데이터를 생성하는 모의 소스를 만들면 특정 병렬 처리 설정에서 애플리케이션의 최대 처리량을 측정할 수 있습니다. 그런 다음 병렬 처리를 늘려 애플리케이션이 적절하게 규모 조정되고 처리량이 높아질 때만 발생하는 병목 현상이 없는지 (예: 데이터 편중) 확인할 수도 있습니다.

데이터 편중

Flink 애플리케이션은 클러스터에서 분산 방식으로 실행됩니다. Flink는 여러 노드로 확장하기 위해 키 스트림이라는 개념을 사용합니다. 즉, 고객 ID와 같은 특정 키에 따라 스트림의 이벤트가 분할되고 Flink는 여러 노드의 여러 파티션을 처리할 수 있습니다. 그런 다음 키 윈도우, 프로세스 함수, 비동기 I/O 등과 같은 여러 Flink 연산자를 이러한 파티션을 기반으로 평가합니다.

파티션 키 선택은 대개 비즈니스 로직에 따라 달라집니다. 동시에 다음과 같은 DynamoDB 및 Spark의 여러 모범 사례가 Flink에도 동일하게 적용됩니다.

- 파티션 키의 높은 농도 보장
- 파티션 간 이벤트 볼륨 왜곡 방지

Flink 대시보드에서 하위 작업(예: 동일한 연산자의 인스턴스)의 수신/전송 기록을 비교하여 파티션의 편차를 식별할 수 있습니다. 또한 Managed Service for Apache Flink 모니터링을 `numRecordsIn/Out` 및 `numRecordsInPerSecond/OutPerSecond`의 지표들이 하위 작업 수준에도 노출되도록 구성할 수 있습니다.

상태 편중

상태 저장 연산자, 즉 창과 같은 비즈니스 로직의 상태를 유지하는 연산자의 경우 데이터 편중은 항상 상태 편중으로 이어집니다. 일부 하위 작업은 데이터의 편중으로 인해 다른 작업보다 더 많은 이벤트를

수신하므로 더 많은 데이터가 상태 그대로 유지됩니다. 그러나 파티션이 균등하게 분산된 애플리케이션의 경우에도 상태에 유지되는 데이터 양에 차이가 있을 수 있습니다. 예를 들어 세션 창의 경우 일부 사용자와 세션은 각각 다른 사용자보다 훨씬 더 길 수 있습니다. 긴 세션이 동일한 파티션에 속하게 되면 동일한 연산자의 여러 하위 태스크가 유지하는 상태 크기가 불균형해질 수 있습니다.

상태 편중은 개별 하위 작업에 필요한 메모리 및 디스크 리소스를 증가시킬 뿐만 아니라 애플리케이션의 전반적인 성능을 저하시킬 수도 있습니다. 애플리케이션이 체크포인트 또는 세이브포인트를 가져오는 경우 연산자 상태가 Amazon S3에 유지되어 노드 또는 클러스터 장애로부터 상태를 보호합니다. 이 프로세스 진행 중에는 (특히 Managed Service for Apache Flink에서 정확히 1회의 시맨틱이 기본으로 활성화되어 있는 경우) 체크포인트/세이브포인트가 완료될 때까지 외부 관점에서 처리가 중단됩니다. 데이터 편중이 있는 경우 특히 많은 양의 상태가 누적된 단일 하위 작업으로 인해 작업을 완료하는 데 걸리는 시간이 제한될 수 있습니다. 극단적인 경우에는 단일 하위 작업이 상태를 유지할 수 없어 체크포인트/세이브포인트 가져오기가 실패할 수 있습니다.

따라서 상태 편중은 데이터 편중과 마찬가지로 애플리케이션 속도를 크게 저하시킬 수 있습니다.

상태 편중을 식별하기 위해 Flink 대시보드를 활용할 수 있습니다. 최근 체크포인트 또는 세이브포인트를 찾아 세부 정보에서 개별 하위 작업에 대해 저장된 데이터의 양을 비교해 보세요.

여러 지역의 리소스와의 통합

Flink 구성에서 지역 간 복제에 필요한 설정을 통해 Managed Service for Apache Flink 애플리케이션과 다른 지역에 있는 Amazon S3 버킷에 쓸 수 있도록 `StreamingFileSink`의 사용을 활성화할 수 있습니다. 이를 위해서는 AWS Support 센터에 지원 티켓을 제출하십시오.

Amazon Managed Service for Apache Flink에 대한 문서 이력

다음 표에서는 Managed Service for Apache Flink의 최신 릴리스 이후 이 설명서에서 변경된 중요 사항에 대해 설명합니다.

- API 버전: 2018-05-23
- 최종 설명서 업데이트: 2023년 8월 30일

변경 사항	설명	날짜
Kinesis Data Analytics에서 Managed Service for Apache Flink로 변경	서비스 엔드포인트, API, 명령줄 인터페이스, IAM 액세스 정책, CloudWatch 지표 또는 결제 대시보드에는 변경 사항이 없습니다. AWS 기존 애플리케이션은 이전과 같이 계속 작동합니다. 자세한 설명은 Managed Service for Apache Flink란 무엇인가?를 참조하세요.	2023년 8월 30일
Apache Flink 버전 1.15.2에 대한 지원	Managed Service for Apache Flink는 이제 Apache Flink 버전 1.15.2를 사용하는 애플리케이션을 지원합니다. Apache Flink 표 API를 사용하여 Kinesis Data Analytics 애플리케이션을 생성합니다. 자세한 설명은 애플리케이션 생성 섹션을 참조하세요.	2022년 11월 22일
Apache Flink 버전 1.13.2에 대한 지원	Managed Service for Apache Flink는 이제 Apache Flink 버전 1.13.2를 사용하는 애	2021년 10월 13일

변경 사항	설명	날짜
	플리케이션을 지원합니다. Apache Flink 표 API를 사용하여 Kinesis Data Analytics 애플리케이션을 생성합니다. 자세한 설명은 시작하기: Flink 1.13.2 섹션을 참조하세요.	
Python에 대한 지원	Managed Service for Apache Flink는 이제 Apache Flink 표 API 및 SQL과 Python을 사용하는 애플리케이션을 지원합니다. 자세한 설명은 Python 사용하기 섹션을 참조하세요.	2021년 3월 25일
Apache Flink 1.11.1에 대한 지원	Managed Service for Apache Flink는 이제 Apache Flink 1.11.1을 사용하는 애플리케이션을 지원합니다. Apache Flink 표 API를 사용하여 Kinesis Data Analytics 애플리케이션을 생성합니다. 자세한 설명은 애플리케이션 생성 섹션을 참조하세요.	2020년 11월 19일
Apache Flink 대시보드	Apache Flink 대시보드를 사용하여 애플리케이션 상태 및 성능을 모니터링할 수 있습니다. 자세한 설명은 Apache Flink 대시보드 섹션을 참조하세요.	2020년 11월 19일
EFO 컨슈머	고급 팬아웃(EFO) 컨슈머를 사용하여 Kinesis Data Stream에서 읽는 애플리케이션을 생성할 수 있습니다. 자세한 설명은 EFO 컨슈머 섹션을 참조하세요.	2020년 10월 6일

변경 사항	설명	날짜
Apache Beam	Apache Beam을 사용하여 스트리밍 데이터를 처리하는 애플리케이션을 생성합니다. 자세한 설명은 Managed Service for Apache Flink와 함께 CloudFormation 사용하기 섹션을 참조하세요.	2020년 9월 15일
성능	애플리케이션 성능 문제를 해결하는 방법 및 성능이 우수한 애플리케이션을 생성하는 방법 자세한 설명은 성능 섹션을 참조하세요.	2020년 7월 21일
맞춤 키스토어	전송 중 암호화를 위해 맞춤 키스토어를 사용하는 Amazon MSK 클러스터에 액세스하는 방법 자세한 설명은 사용자 지정 트러스트 스토어 섹션을 참조하세요.	2020년 6월 10일
CloudWatch 알람	Apache Flink용 관리형 서비스를 사용하여 CloudWatch 경보를 생성하기 위한 권장 사항 자세한 설명은 경보 섹션을 참조하세요.	2020년 6월 5일
CloudWatch 새 지표	아파치 플링크용 관리형 서비스는 이제 Amazon Metrics에 22개의 지표를 내보냅니다. CloudWatch 자세한 설명은 Managed Service for Apache Flink의 지표 및 차원 섹션을 참조하세요.	2020년 5월 12일

변경 사항	설명	날짜
CloudWatch 사용자 지정 지표	애플리케이션별 지표를 정의하고 이를 Amazon Metrics로 내보냅니다. CloudWatch 자세한 설명은 사용자 지정 지표 섹션을 참조하세요.	2020년 5월 12일
예: 다른 계정의 Kinesis 스트림에서 읽기를 참조하세요.	Managed Service for Apache Flink 애플리케이션에서 다른 AWS 계정으로 Kinesis 스트림에 액세스하는 방법을 알아봅니다. 자세한 설명은 크로스 계정 섹션을 참조하세요.	2020년 3월 30일
Apache Flink 1.8.2에 대한 지원	Managed Service for Apache Flink는 이제 Apache Flink 1.8.2를 사용하는 애플리케이션을 지원합니다. Flink StreamingFileSink 커넥터를 사용하여 출력을 S3에 직접 기록할 수 있습니다. 자세한 설명은 애플리케이션 생성 섹션을 참조하세요.	2019년 12월 17일
Managed Service for Apache Flink VPC	가상 프라이빗 클라우드에 연결되도록 Managed Service for Apache Flink 애플리케이션을 구성합니다. 자세한 설명은 Amazon VPC 사용 섹션을 참조하세요.	2019년 11월 25일
Managed Service for Apache Flink 모범 사례	Managed Service for Apache Flink 애플리케이션을 만들고 관리하는 모범 사례입니다. 자세한 설명은 모범 사례 섹션을 참조하세요.	2019년 10월 14일

변경 사항	설명	날짜
Managed Service for Apache Flink 애플리케이션 로그 분석	CloudWatch 로그 인사이트를 사용하여 Apache Flink용 관리형 서비스 애플리케이션을 모니터링할 수 있습니다. 자세한 설명은 로그 분석 섹션을 참조하세요.	2019년 6월 26일
Managed Service for Apache Flink 애플리케이션 런타임 속성	Managed Service for Apache Flink에서 런타임 속성으로 작업합니다. 자세한 설명은 런타임 속성 섹션을 참조하세요.	2019년 6월 24일
Managed Service for Apache Flink 애플리케이션 태그 지정	애플리케이션 태그 지정을 사용하여 애플리케이션별 비용, 액세스 통제 또는 사용자 정의 용도를 결정합니다. 자세한 설명은 태그 지정 사용 섹션을 참조하세요.	2019년 5월 8일
Managed Service for Apache Flink 예 애플리케이션	창 연산자를 보여주고 Logs에 출력을 쓰는 방법을 보여주는 Apache Flink용 관리형 서비스의 예제 애플리케이션 CloudWatch 자세한 설명은 예제 섹션을 참조하세요.	2019년 5월 1일
AWS CloudTrail을 사용한 Managed Service for Apache Flink API 호출 로깅	Managed Service for Apache Flink는 Managed Service for Apache Flink에서 사용자, 역할 또는 AWS 서비스가 수행한 작업에 대한 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. 자세한 설명은 AWS CloudTrail 사용하기 섹션을 참조하세요.	2019년 3월 22일

변경 사항	설명	날짜
애플리케이션(Kinesis Data Firehose Sink) 생성	Amazon Kinesis 데이터 스트림을 소스로 사용하고 Amazon Kinesis Data Firehose 스트림을 싱크로 사용하여 Managed Service for Apache Flink 생성을 연습해 봅니다. 자세한 설명은 Kinesis Data Firehose Sink 섹션을 참조하세요.	2018년 12월 13일
공개 릴리스	Java 애플리케이션용 Managed Service for Apache Flink 개발자 가이드가 처음으로 릴리스되었습니다.	2018년 11월 27일

Managed Service for Apache Flink API 예 코드

이 항목에는 Managed Service for Apache Flink 작업을 위한 예 요청 블록이 포함되어 있습니다.

AWS Command Line Interface(AWS CLI)와 함께 작업을 위한 입력으로 JSON을 사용하려면 요청을 JSON 파일에 저장하십시오. 그런 다음 `--cli-input-json` 파라미터를 사용하여 파일 명칭을 작업에 전달합니다.

다음 예에서는 작업과 함께 JSON 파일을 사용하는 방법을 보여줍니다.

```
$ aws kinesisanalyticsv2 start-application --cli-input-json file://start.json
```

AWS CLI과 함께 JSON을 사용하는 방법에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드의 CLI Skeleton 및 CLI 입력 JSON 파라미터 생성을 참조하십시오.

주제

- AddApplicationCloudWatchLoggingOption
- AddApplicationInput
- AddApplicationInputProcessingConfiguration
- AddApplicationOutput
- AddApplicationReferenceDataSource
- AddApplicationVpcConfiguration
- CreateApplication
- CreateApplicationSnapshot
- DeleteApplication
- DeleteApplicationCloudWatchLoggingOption
- DeleteApplicationInputProcessingConfiguration
- DeleteApplicationOutput
- DeleteApplicationReferenceDataSource
- DeleteApplicationSnapshot
- DeleteApplicationVpcConfiguration
- DescribeApplication

- DescribeApplicationSnapshot
- DiscoverInputSchema
- ListApplications
- ListApplicationSnapshots
- StartApplication
- StopApplication
- UpdateApplication

AddApplicationCloudWatchLoggingOption

AddApplicationCloudWatchLoggingOption 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션에 Amazon CloudWatch 로깅 옵션을 추가합니다:

```
{
    "ApplicationName": "MyApplication",
    "CloudWatchLoggingOption": {
        "LogStreamARN": "arn:aws:logs:us-east-1:123456789123:log-group:my-log-
group:log-stream:My-LogStream"
    },
    "CurrentApplicationVersionId": 2
}
```

AddApplicationInput

AddApplicationInput 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink에 애플리케이션 입력을 추가합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 2,
   "Input": {
      "InputParallelism": {
         "Count": 2
      },
      "InputSchema": {
         "RecordColumns": [
            {
               "Mapping": "$.TICKER",
```

```
                "Name": "TICKER_SYMBOL",
                "SqlType": "VARCHAR(50)"
            },
            {
                "SqlType": "REAL",
                "Name": "PRICE",
                "Mapping": "$.PRICE"
            }
         ],
         "RecordEncoding": "UTF-8",
         "RecordFormat": {
            "MappingParameters": {
               "JSONMappingParameters": {
                  "RecordRowPath": "$"
               }
            },
            "RecordFormatType": "JSON"
         }
      },
      "KinesisStreamsInput": {
         "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/
ExampleInputStream"
      }
   }
}
```

AddApplicationInputProcessingConfiguration

ApplicationInputProcessingConfiguration 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션에 애플리케이션 입력 처리 구성을 추가합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 2,
   "InputId": "2.1",
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
         "ResourceARN": "arn:aws:lambda:us-
east-1:012345678901:function:MyLambdaFunction"
      }
   }
}
```

AddApplicationOutput

AddApplicationOutput 작업을 위한 다음 예 요청 코드는 Kinesis 데이터 스트림을 Managed Service for Apache Flink 애플리케이션에 애플리케이션 출력으로 추가합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 2,
   "Output": {
      "DestinationSchema": {
         "RecordFormatType": "JSON"
      },
      "KinesisStreamsOutput": {
         "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/
ExampleOutputStream"
      },
      "Name": "DESTINATION_SQL_STREAM"
   }
}
```

AddApplicationReferenceDataSource

AddApplicationReferenceDataSource 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션에 CSV 애플리케이션 참조 데이터 소스를 추가합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 5,
   "ReferenceDataSource": {
      "ReferenceSchema": {
         "RecordColumns": [
            {
               "Mapping": "$.TICKER",
               "Name": "TICKER",
               "SqlType": "VARCHAR(4)"
            },
            {
               "Mapping": "$.COMPANYNAME",
               "Name": "COMPANY_NAME",
               "SqlType": "VARCHAR(40)"
            },
         ],
```

```
         "RecordEncoding": "UTF-8",
         "RecordFormat": {
            "MappingParameters": {
               "CSVMappingParameters": {
                  "RecordColumnDelimiter": " ",
                  "RecordRowDelimiter": "\r\n"
               }
            },
            "RecordFormatType": "CSV"
         }
      },
      "S3ReferenceDataSource": {
         "BucketARN": "arn:aws:s3:::MyS3Bucket",
         "FileKey": "TickerReference.csv"
      },
      "TableName": "string"
   }
}
```

AddApplicationVpcConfiguration

AddApplicationVPCConfiguration 작업을 위한 다음 예 요청 코드는 기존 애플리케이션에 VPC 구성을 추가합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfiguration": {
      "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
      "SubnetIds": [ "subnet-0123456789abcdef0" ]
   }
}
```

CreateApplication

CreateApplication 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션을 생성합니다:

```
{
  "ApplicationName":"MyApplication",
```

```
  "ApplicationDescription":"My-Application-Description",
  "RuntimeEnvironment":"FLINK-1_15",
  "ServiceExecutionRole":"arn:aws:iam::123456789123:role/myrole",
  "CloudWatchLoggingOptions":[
    {
      "LogStreamARN":"arn:aws:logs:us-east-1:123456789123:log-group:my-log-group:log-
stream:My-LogStream"
    }
  ],
  "ApplicationConfiguration": {
    "EnvironmentProperties":
      {"PropertyGroups":
        [
          {"PropertyGroupId": "ConsumerConfigProperties",
            "PropertyMap":
              {"aws.region": "us-east-1",
              "flink.stream.initpos": "LATEST"}
          },
          {"PropertyGroupId": "ProducerConfigProperties",
            "PropertyMap":
              {"aws.region": "us-east-1"}
          },
        ]
      },
    "ApplicationCodeConfiguration":{
      "CodeContent":{
        "S3ContentLocation":{
          "BucketARN":"arn:aws:s3:::mybucket",
          "FileKey":"myflink.jar",
          "ObjectVersion":"AbCdEfGhIjKlMnOpQrStUvWxYz12345"
        }
      },
      "CodeContentType":"ZIPFILE"
    },
      "FlinkApplicationConfiguration":{
      "ParallelismConfiguration":{
        "ConfigurationType":"CUSTOM",
        "Parallelism":2,
        "ParallelismPerKPU":1,
        "AutoScalingEnabled":true
      }
    }
  }
```

```
}
```

CreateApplicationSnapshot

CreateApplicationSnapshot 작업을 위한 다음 예 요청 코드는 애플리케이션 상태의 스냅샷을 생성합니다:

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MySnapshot"
}
```

DeleteApplication

DeleteApplication 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션을 삭제합니다:

```
{"ApplicationName": "MyApplication",
"CreateTimestamp": 12345678912}
```

DeleteApplicationCloudWatchLoggingOption

DeleteApplicationCloudWatchLoggingOption 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션에서 Amazon CloudWatch 로깅 옵션을 삭제합니다:

```
{
    "ApplicationName": "MyApplication",
    "CloudWatchLoggingOptionId": "3.1"
    "CurrentApplicationVersionId": 3
}
```

DeleteApplicationInputProcessingConfiguration

DeleteApplicationInputProcessingConfiguration 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink에서 입력 처리 구성을 제거합니다:

```
{
```

```
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "InputId": "2.1"
}
```

DeleteApplicationOutput

DeleteApplicationOutput 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션에서 애플리케이션 출력을 제거합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 4,
   "OutputId": "4.1"
}
```

DeleteApplicationReferenceDataSource

DeleteApplicationReferenceDataSource 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션에서 애플리케이션 참조 데이터 소스를 제거합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 5,
   "ReferenceId": "5.1"
}
```

DeleteApplicationSnapshot

DeleteApplicationSnapshot 작업을 위한 다음 예 요청 코드는 애플리케이션 상태의 스냅샷을 삭제합니다:

```
{
   "ApplicationName": "MyApplication",
   "SnapshotCreationTimestamp": 12345678912,
   "SnapshotName": "MySnapshot"
}
```

DeleteApplicationVpcConfiguration

DeleteApplicationVpcConfiguration 삭제 작업을 위한 다음 예 요청 코드는 애플리케이션에서 기존 VPC 구성을 제거합니다:

```
{
   "ApplicationName": "MyApplication",
   "CurrentApplicationVersionId": 9,
   "VpcConfigurationId": "1.1"
}
```

DescribeApplication

DescribeApplication 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink에 대한 세부 정보를 반환합니다:

```
{"ApplicationName": "MyApplication"}
```

DescribeApplicationSnapshot

DescribeApplicationSnapshot 작업을 위한 다음 예 요청 코드는 애플리케이션 상태의 스냅샷에 대한 세부 정보를 반환합니다:

```
{
   "ApplicationName": "MyApplication",
   "SnapshotName": "MySnapshot"
}
```

DiscoverInputSchema

DiscoverInputSchema 작업을 위한 다음 예 요청 코드는 스트리밍 소스에서 스키마를 생성합니다:

```
{
   "InputProcessingConfiguration": {
      "InputLambdaProcessor": {
         "ResourceARN": "arn:aws:lambda:us-
east-1:012345678901:function:MyLambdaFunction"
```

```
      }
   },
   "InputStartingPositionConfiguration": {
      "InputStartingPosition": "NOW"
   },
   "ResourceARN": "arn:aws:kinesis:us-east-1:012345678901:stream/ExampleInputStream",
   "S3Configuration": {
      "BucketARN": "string",
      "FileKey": "string"
   },
   "ServiceExecutionRole": "string"
}
```

DiscoverInputSchema 작업을 위한 다음 예 요청 코드는 참조 소스에서 스키마를 생성합니다:

```
{
   "S3Configuration": {
      "BucketARN": "arn:aws:s3:::mybucket",
      "FileKey": "TickerReference.csv"
   },
   "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole"
}
```

ListApplications

ListApplications 작업을 위한 다음 예 요청 코드는 계정의 Managed Service for Apache Flink 애플리케이션 목록을 반환합니다:

```
{
   "ExclusiveStartApplicationName": "MyApplication",
   "Limit": 50
}
```

ListApplicationSnapshots

ListApplication Snapshots 작업을 위한 다음 예 요청 코드는 애플리케이션 상태의 스냅샷 목록을 반환합니다:

```
{"ApplicationName": "MyApplication",
   "Limit": 50,
```

```
   "NextToken": "aBcDeFgHiJkLmNoPqRsTuVwXyZ0123"
}
```

StartApplication

StartApplication 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션을 시작하고 최신 스냅샷(있는 경우)에서 애플리케이션 상태를 로드합니다:

```
{
    "ApplicationName": "MyApplication",
    "RunConfiguration": {
        "ApplicationRestoreConfiguration": {
         "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
         }
    }
}
```

StopApplication

API_StopApplication 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink를 중지합니다:

```
{"ApplicationName": "MyApplication"}
```

UpdateApplication

UpdateApplication 작업을 위한 다음 예 요청 코드는 Managed Service for Apache Flink 애플리케이션을 업데이트하여 애플리케이션 코드의 위치를 변경합니다:

```
{"ApplicationName": "MyApplication",
"CurrentApplicationVersionId": 1,
"ApplicationConfigurationUpdate": {
      "ApplicationCodeConfigurationUpdate": {
         "CodeContentTypeUpdate": "ZIPFILE",
         "CodeContentUpdate": {
            "S3ContentLocationUpdate": {
               "BucketARNUpdate": "arn:aws:s3:::my_new_bucket",
               "FileKeyUpdate": "my_new_code.zip",
```

```
                "ObjectVersionUpdate": "2"
            }
        }
    }
}
```

Managed Service for Apache Flink API 참조

Managed Service for Apache Flink에서 제공하는 API에 대한 자세한 설명은 Managed Service for Apache FlinkAPI 참조를 참조하십시오.