



Managed Service for Apache Flink 개발자 안내서

Managed Service for Apache Flink



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 브랜드 디자인은 Amazon 외 제품 또는 서비스와 함께, 브랜드 이미지를 떨어뜨리거나 고객에게 혼동을 일으킬 수 있는 방식으로 사용할 수 없습니다. Amazon이 소유하지 않은 기타 모든 상표는 Amazon과 제휴 관계이거나 관련이 있거나 후원 관계와 관계없이 해당 소유자의 자산입니다.

Table of Contents

.....	xvi
아파치 플링크용 관리형 서비스란 무엇입니까?	1
Apache Flink용 관리형 서비스 또는 Apache Flink Studio용 관리형 서비스 선택	1
Apache Flink용 관리 서비스에서 사용할 Apache Flink API 선택	3
플링크 API 선택하기	3
스트리밍 데이터 애플리케이션 시작하기	4
작동 방식	6
아파치 플링크 애플리케이션 프로그래밍	6
DataStream API	6
표 API	7
Apache Flink 애플리케이션용 관리형 서비스 만들기	7
애플리케이션 생성	8
Apache Flink용 매니지드 서비스 구축 애플리케이션 코드	8
Apache Flink 애플리케이션용 관리형 서비스 만들기	9
Apache Flink용 매니지드 서비스 애플리케이션 시작	10
Apache Flink용 매니지드 서비스 애플리케이션 확인	11
Apache Flink용 관리형 서비스 애플리케이션의 시스템 롤백 활성화	11
애플리케이션 실행	13
애플리케이션 및 작업 상태	14
배치 워크로드	15
애플리케이션 리소스	15
아파치 Flink 애플리케이션 리소스를 위한 매니지드 서비스	15
아파치 플링크 애플리케이션 리소스	16
DataStream API	17
DataStream API 커넥터	17
DataStream API 연산자	27
DataStream API 타임스탬프	28
표 API	28
테이블 API 커넥터	29
테이블 API 시간 속성	30
Python 사용	31
애플리케이션 프로그래밍	31
애플리케이션 생성	34
모니터링	35

런타임 속성	37
콘솔에서 런타임 속성으로 작업하기	37
CLI에서 런타임 속성 사용	38
Apache Flink용 관리 서비스 애플리케이션의 런타임 속성에 액세스	40
아파치 플링크 커넥터	41
내결함성	43
체크포인트 구성	44
체크포인트 API 예제	45
스냅샷	47
자동 스냅샷 생성	48
호환되지 않는 상태 데이터가 포함된 스냅샷에서 복원	49
스냅샷 API 예제	50
현재 위치 버전 업그레이드	52
Apache Flink의 인플레이스 버전 업그레이드를 사용하여 애플리케이션을 업그레이드합니 다.	53
애플리케이션을 새 Apache Flink 버전으로 업그레이드	54
롤백	59
일반 모범 사례 및 권장 사항	59
예방 조치 및 알려진 문제	60
스케일링	62
애플리케이션 병렬 처리 및 KPU 구성 ParallelismPer	62
Kinesis 처리 단위 할당	63
애플리케이션의 병렬성 업데이트	63
자동 크기 조정	65
태그 지정	67
애플리케이션 생성 시 태그 추가	67
기존 애플리케이션에 태그 추가 또는 u 태그 추가	68
애플리케이션 태그 목록	68
애플리케이션에서 태그 제거	69
아파치 플링크용 매니지드 서비스와 CloudFormation 함께 사용	69
시작하기 전 준비 사항	69
Lambda 함수 작성	69
Lambda 역할 생성	71
Lambda 함수 호출	72
Lambda 함수 호출	72
Apache Flink 대시보드	78

애플리케이션의 Apache Flink 대시보드에 액세스	79
릴리스 버전	80
아파치 플링크용 아마존 매니지드 서비스 1.19	81
지원되는 기능	82
아파치 플링크용 아마존 매니지드 서비스 1.19.1의 변경	84
구성 요소	84
알려진 문제	85
아파치 플링크용 아마존 매니지드 서비스 1.18	85
Apache Flink 1.15를 사용하는 Amazon Managed Service for Apache Flink의 변경 사항	87
구성 요소	87
버그 수정	88
알려진 문제	88
아파치 플링크용 아마존 매니지드 서비스 1.15	88
Apache Flink 1.15를 사용하는 Amazon Managed Service for Apache Flink의 변경 사항	87
구성 요소	87
이전 버전	91
아파치 플링크 Kinesis Streams 커넥터를 이전 아파치 플링크 버전과 함께 사용하기	92
아파치 플링크 1.8.2를 이용한 애플리케이션 구축	93
아파치 플링크를 이용한 애플리케이션 구축 1.6.2	94
애플리케이션 업그레이드	95
아파치 플링크 1.6.2 및 1.8.2에서 사용 가능한 커넥터	95
시작하기: Flink 1.13.2	95
시작하기: Flink 1.11.1	120
시작하기: 플링크 1.8.2 - 지원 중단	145
시작하기: 플링크 1.6.2 - 지원 중단	170
레거시 예제	194
Studio 노트북	361
스튜디오 노트북 버전	362
Studio 노트북 생성	363
스트리밍 데이터의 대화형 분석	364
플링크 인터프리터	364
Apache Flink 테이블 환경 변수	365
지속 가능한 상태의 애플리케이션으로 배포	366
Scala/Python 기준	367
SQL 기준	367
IAM 권한	368

커넥터 및 종속성	368
기본 커넥터	368
종속성 및 사용자 지정 커넥터 추가	370
사용자 정의 함수	371
사용자 정의 함수 관련 고려 사항	372
체크포인트 활성화	373
체크포인트 간격 설정	373
체크포인트 유형 설정	374
Studio 런타임 업그레이드	374
노트북을 새 Studio 런타임으로 업그레이드	374
작업 대상 AWS Glue	379
테이블 속성	379
예시 및 자습서	381
스튜디오 노트북 만들기 튜토리얼	381
내구성이 뛰어난 상태 튜토리얼을 통한 애플리케이션으로 배포	401
예	404
문제 해결	416
중단된 애플리케이션 중지	416
인터넷에 접속할 수 없는 VPC에서 지속 가능한 상태의 애플리케이션으로 배포	416
D eploy-as-app 크기 및 빌드 시간 단축	417
작업 취소	419
Apache Flink 인터프리터 재시작	420
부록: 사용자 지정 IAM 정책 생성	420
AWS Glue	421
CloudWatch 로그	421
Kinesis 스트림	422
Amazon MSK 클러스터	425
시작하기 (API) DataStream	426
애플리케이션 구성 요소	146
사전 조건	427
1단계: 계정 설정	427
가입하세요. AWS 계정	97
관리자 액세스 권한이 있는 사용자 생성	97
프로그래밍 방식 액세스 권한 부여	429
다음 단계	430
2단계: 설정 AWS CLI	430

다음 단계	432
3단계: 애플리케이션 생성	432
두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.	432
입력 스트림에 샘플 레코드를 작성합니다.	433
Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.	434
애플리케이션 코드를 컴파일하십시오.	435
아파치 플링크 스트리밍 자바 코드를 업로드하세요.	436
Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.	437
다음 단계	448
4단계: 정리	449
Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.	449
Kinesis 데이터 스트림을 삭제합니다.	449
Amazon S3 객체 및 버킷을 삭제합니다.	449
IAM 리소스를 삭제합니다.	450
CloudWatch 리소스 삭제하기	450
다음 단계	450
5단계: 리소스를 사용하여 다음 단계 완료	450
시작하기 (표 API)	452
애플리케이션 구성 요소	452
사전 조건	453
애플리케이션 만들기	453
종속 리소스 생성	453
입력 스트림에 샘플러 레코드를 기록하십시오.	455
아파치 플링크 스트리밍 자바 코드를 다운로드하고 검사하세요.	456
애플리케이션 코드를 컴파일하십시오.	458
아파치 플링크 스트리밍 자바 코드를 업로드하세요.	458
Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.	459
다음 단계	463
정리	464
Apache Flink용 관리형 서비스 애플리케이션 삭제	464
Amazon MSK 클러스터를 삭제합니다.	464
VPC 삭제	464
Amazon S3 객체 및 버킷을 삭제합니다.	465
IAM 리소스를 삭제합니다.	465
CloudWatch 리소스 삭제하기	465
다음 단계	466

다음 단계	466
시작하기 (Python)	467
Pyflink 시작하기 - Apache를 위한 Python 인터프리터 Amazon 웹 서비스	467
애플리케이션 구성 요소	467
사전 조건	468
애플리케이션 만들기	468
종속 리소스 생성	469
입력 스트림에 샘플 레코드를 작성합니다.	470
아파치 플링크 스트리밍 Python 코드 생성 및 검토	471
Python 앱에 제3자 종속성 추가	473
아파치 플링크 스트리밍 Python 코드 업로드	474
Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다.	476
다음 단계	480
정리	480
Apache Flink용 관리형 서비스 애플리케이션 삭제	481
Kinesis 데이터 스트림을 삭제합니다.	481
Amazon S3 객체 및 버킷을 삭제합니다.	481
IAM 리소스를 삭제합니다.	481
CloudWatch 리소스 삭제하기	482
시작하기 (스칼라)	483
종속 리소스 생성	483
입력 스트림에 샘플 레코드를 작성합니다.	484
애플리케이션 코드 다운로드 및 검토	486
애플리케이션 코드 컴파일 및 업로드	487
애플리케이션 생성 및 실행 (콘솔)	488
애플리케이션 생성	488
애플리케이션 구성	489
IAM 정책을 편집하십시오.	491
애플리케이션을 실행합니다	492
애플리케이션 중지	493
애플리케이션 생성 및 실행(CLI)	493
권한 정책 생성	493
IAM 정책 생성	495
애플리케이션 생성	496
애플리케이션 시작	497
애플리케이션을 중지하십시오.	356

CloudWatch 로깅 옵션 추가	357
환경 속성 업데이트	357
애플리케이션 코드 업데이트	358
정리	501
Apache Flink용 관리형 서비스 애플리케이션 삭제	501
Kinesis 데이터 스트림을 삭제합니다.	501
Amazon S3 객체 및 버킷을 삭제합니다.	501
IAM 리소스를 삭제합니다.	502
CloudWatch 리소스 삭제하기	502
Apache Beam 사용	503
Managed Service for Apache Flink와 함께 Apache Beam 사용	503
빔 기능	504
Apache Beam을 사용하여 애플리케이션 생성	504
종속 리소스 생성	504
입력 스트림에 샘플 레코드 쓰기	505
애플리케이션 코드 다운로드 및 검토	506
애플리케이션 코드를 컴파일합니다.	507
아파치 플링크 스트리밍 자바 코드를 업로드하세요.	508
Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.	508
정리	512
다음 단계	513
교육 워크숍, 랩 및 솔루션 구현	514
아파치 플링크를 위한 매니지드 서비스 워크숍	514
아파치 플링크용 매니지드 서비스에 배포하기 전에 로컬에서 아파치 플링크 애플리케이션을 개 발하세요.	514
Managed Service for Apache Flink Studio를 사용한 이벤트 감지	515
AWS 스트리밍 데이터 솔루션	515
클릭스트림 랩	515
맞춤 조정	516
CloudWatch 대시보드	516
Amazon MSK	516
에서 Apache Flink용 매니지드 서비스 솔루션에 대해 자세히 알아보십시오. GitHub	516
유틸리티	518
스냅샷 관리자	518
벤치마킹	518
예제	519

Java 예제	519
Python 예제	521
.....	521
.....	522
스칼라 예제	522
보안	523
데이터 보호	523
데이터 암호화	524
자격 증명 및 액세스 관리	525
고객	525
ID를 통한 인증	526
정책을 사용한 액세스 관리	529
Amazon Managed Service for Apache Flink가 IAM과 연동되는 방식	531
자격 증명 기반 정책 예시	538
문제 해결	541
교차 서비스 혼동된 대리인 방지	542
모니터링	544
규정 준수 확인	544
FedRAMP	545
복원력	545
재해 복구	545
버전 관리	546
인프라 보안	546
보안 모범 사례	547
최소 권한 액세스 구현	547
IAM 역할을 사용하여 다른 Amazon 서비스에 액세스	547
종속 리소스에 서버 측 암호화를 구현하십시오.	547
API 호출을 모니터링하는 데 사용합니다. CloudTrail	548
로그 및 모니터링	549
로그	550
로그 인사이트를 통한 로그 쿼리 CloudWatch	550
모니터링	550
로그 설정	551
콘솔을 사용한 CloudWatch 로깅 설정	552
CLI를 사용한 CloudWatch 로깅 설정	553
애플리케이션 모니터링 수준	558

로그 모범 사례	559
로그 문제 해결	559
다음 단계	560
로그 분석	560
샘플 쿼리 실행	560
쿼리 예제	561
Apache Flink용 매니지드 서비스의 메트릭과 디멘션	564
애플리케이션 지표	564
Kinesis Data Streams 커넥터 메트릭	589
아마존 MSK 커넥터 메트릭	589
아파치 제플린 메트릭	591
지표 보기 CloudWatch	592
지표	592
사용자 지정 지표	594
경보	597
커스텀 메시지 작성	607
Log4J를 사용하여 CloudWatch 로그에 기록합니다.	608
SLF4J 사용하여 CloudWatch 로그에 쓰기	609
사용 AWS CloudTrail	610
Apache Flink용 관리형 서비스 정보는 다음을 참조하십시오. CloudTrail	610
Apache Flink용 관리형 서비스 로그 파일 항목의 이해	611
성능	614
성능 문제 해결	614
데이터 경로	614
성능 문제 해결 솔루션	615
성능 모범 사례	617
적절한 크기 조정	617
외부 종속성 리소스 사용량 모니터링	619
Apache Flink 애플리케이션을 로컬에서 실행	619
모니터링 성능	619
메트릭을 사용한 성능 모니터링 CloudWatch	619
CloudWatch 로그 및 경보를 사용한 성능 모니터링	620
할당량	621
정비	623
모든 연산자에 대해 UUID 설정	625
유지보수 인스턴스 식별	625

프로덕션 준비 상태	626
부하 테스트 애플리케이션	626
최대 병렬 처리	626
모든 연산자에 대해 UUID 설정	627
모범 사례	628
내결함성: 체크포인트 및 세이브포인트	628
지원되지 않는 커넥터 버전	629
성능 및 병렬 처리	629
연산자별 병렬 처리 설정	630
로깅	630
코딩	631
보안 인증 관리	631
샤드/파티션이 거의 없는 소스에서 읽기	632
스튜디오 노트북 새로 고침 간격	632
스튜디오 노트북 최적 성능	632
워터마크 전략과 유휴 샤드가 타임윈도우에 미치는 영향	633
요약	634
예	634
모든 연산자에 대해 UUID 설정	643
메이븐 ServiceResourceTransformer 웨이드 플러그인에 추가	644
아파치 플링크 스테이트풀 함수	645
아파치 플링크 애플리케이션 템플릿	645
모듈 구성의 위치입니다.	646
플링크 설정	647
아파치 플링크 구성	647
스테이트 백엔드	647
체크포인트	648
세이브포인트	649
힙 크기	650
버퍼 디블로팅	650
수정 가능한 Flink 구성 속성	650
내결함성	650
체크포인트 및 상태 백엔드	650
체크포인트	650
RocksDB 네이티브 메트릭	651
고급 상태 백엔드 옵션	652

전체 옵션 TaskManager	652
메모리 구성	653
RPC / Akka	653
클라이언트	653
고급 클러스터 옵션	654
파일 시스템 구성	654
고급 내결함성 옵션	654
메모리 구성	653
지표	654
REST 엔드포인트 및 클라이언트를 위한 고급 옵션	654
고급 SSL 보안 옵션	654
고급 예약 옵션	654
플링크 웹 UI의 고급 옵션	655
구성된 Flink 속성 보기	655
Amazon VPC 사용	656
Amazon VPC 개념	656
VPC 애플리케이션 권한	657
Amazon VPC에 액세스하기 위한 권한 정책	657
인터넷 및 서비스 액세스	658
관련 정보	659
VPC API	660
애플리케이션 생성	660
AddApplicationVpcConfiguration	661
DeleteApplicationVpcConfiguration	661
애플리케이션 업데이트	661
예: VPC 사용	662
문제 해결	663
개발 문제 해결	663
FlinkRuntimeException: “허용되지 않은 구성 변경이 감지되었습니다.”	663
시스템 롤백 모범 사례	664
Hudi 구성 모범 사례	666
Apache Flink 프레임 그래프	666
EFO 커넥터 1.15.2의 자격 증명 공급자 문제	666
지원되지 않는 Kinesis 커넥터가 있는 애플리케이션	666
컴파일 오류: “프로젝트에 대한 종속성을 해결할 수 없습니다.”	669
잘못된 선택: “kinesisanalyticsv2”	670

UpdateApplication 액션은 애플리케이션 코드를 다시 로드하는 것이 아닙니다.	670
S3 StreamingFileSink FileNotFoundExceptions	670
FlinkKafkaConsumer 세이브 포인트 사용 중지 관련 문제	672
Flink 1.15 Async Sink 교착 상태	672
리샤딩 중에 Amazon Kinesis 데이터 스트림의 소스 처리 순서가 잘못되었습니다.	682
런타임 문제 해결	683
문제 해결 도구	683
애플리케이션 문제	683
애플리케이션 재시작 중	688
처리량이 너무 느림	690
무한한 국가 성장	691
I/O 바운드 연산자	693
Kinesis 데이터 스트림의 업스트림 또는 소스 조절	693
체크포인트	694
체크포인트 시간 초과	700
체크포인트 실패 (Beam)	702
역압	703
데이터 편중	705
상태 편중	705
여러 지역의 리소스와의 통합	706
사용 설명서 기록	707
API 예제 코드	713
AddApplicationCloudWatchLoggingOption	714
AddApplicationInput	714
AddApplicationInputProcessingConfiguration	715
AddApplicationOutput	716
AddApplicationReferenceDataSource	716
AddApplicationVpcConfiguration	717
CreateApplication	717
CreateApplicationSnapshot	719
DeleteApplication	719
DeleteApplicationCloudWatchLoggingOption	719
DeleteApplicationInputProcessingConfiguration	719
DeleteApplicationOutput	720
DeleteApplicationReferenceDataSource	720
DeleteApplicationSnapshot	720

DeleteApplicationVpcConfiguration	721
DescribeApplication	721
DescribeApplicationSnapshot	721
DiscoverInputSchema	721
ListApplications	722
ListApplicationSnapshots	722
StartApplication	723
StopApplication	723
UpdateApplication	723
API 참조	725
.....	726

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.

아파치 플링크용 아마존 매니지드 서비스란 무엇입니까?

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.

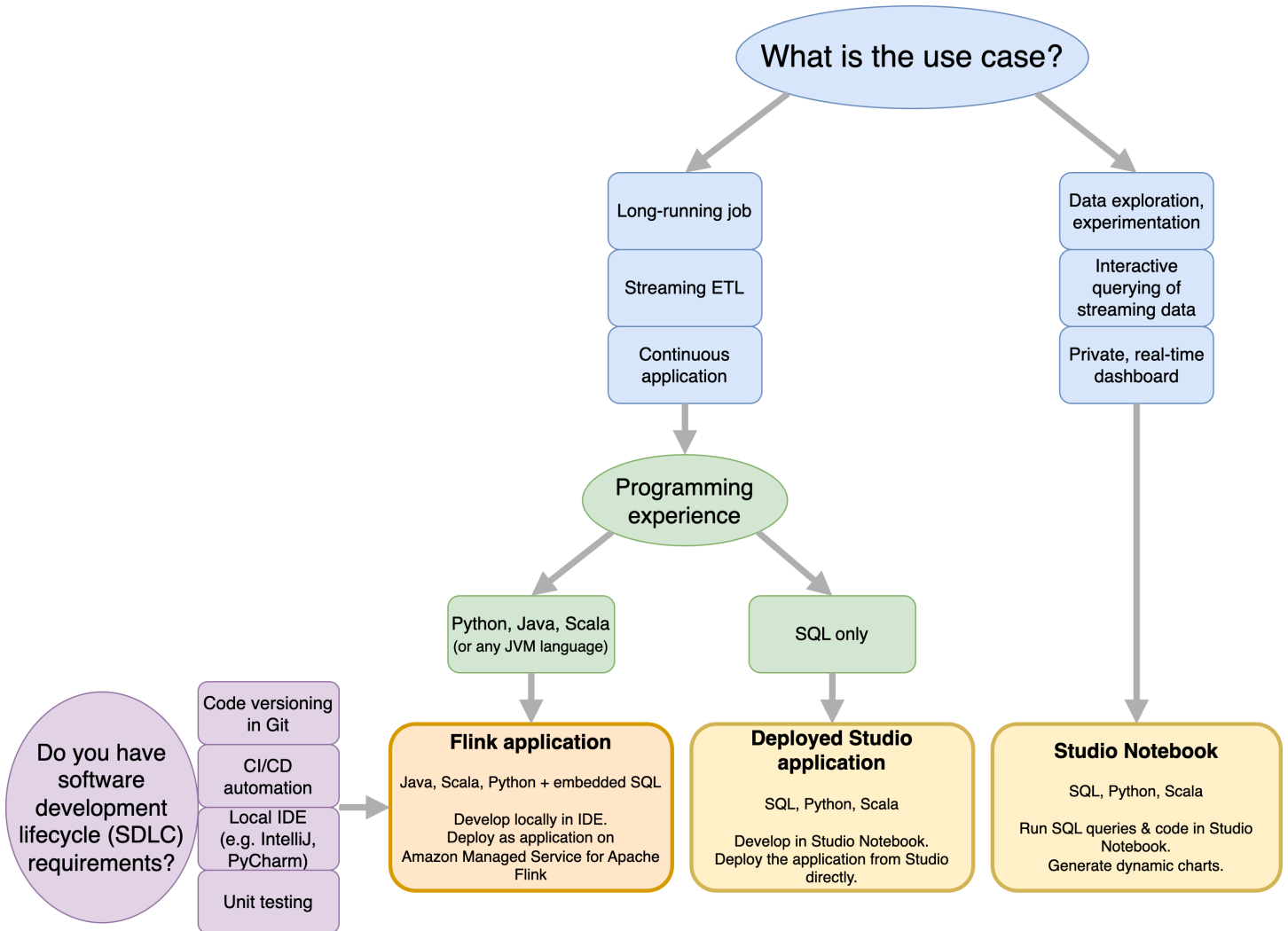
[Apache Flink 기반 오픈 소스 라이브러리를 사용하여 Apache Flink용 관리 서비스에서 원하는 언어로 애플리케이션을 빌드할 수 있습니다.](#) Apache Flink는 데이터 스트림을 처리하기 위한 인기 있는 프레임워크 및 엔진입니다.

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 Studio](#)를 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.

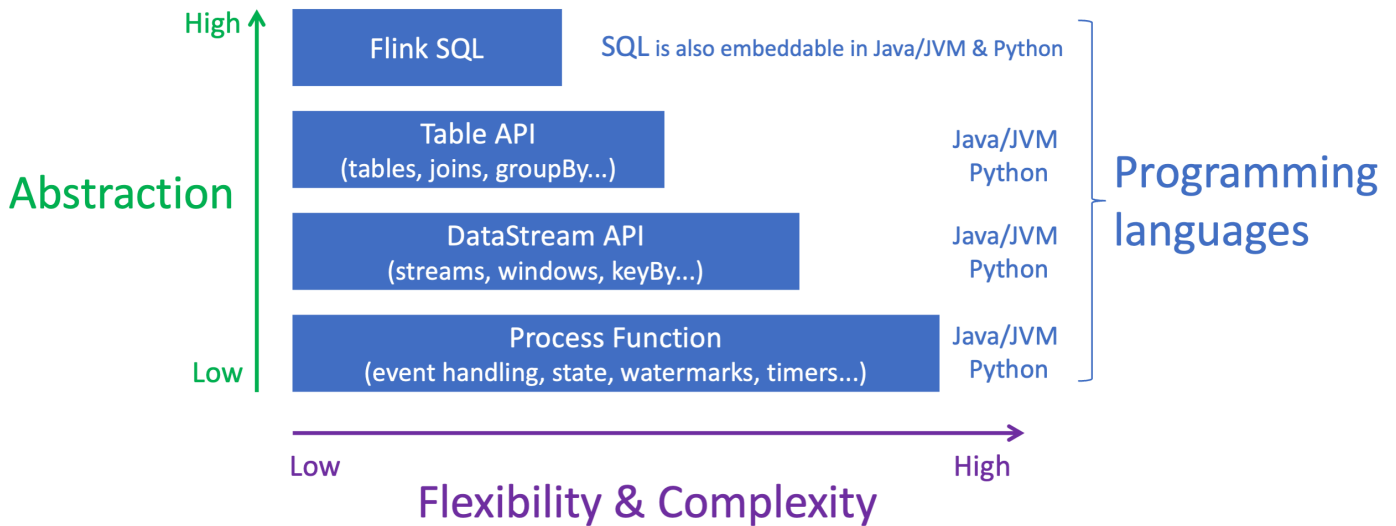
플링크 API 선택하기

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 사용하려는 경우 (예: 사용자 지정 윈도우 설정 또는 지연 이벤트 처리 구현)
- 상태를 재설정하지 않고도 애플리케이션의 흐름을 수정할 수 있기를 원합니다.

Apache Flink APIs



Note

DataStreamAPI로 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.

API 선택은 상태를 재설정하지 않고도 애플리케이션 로직을 발전시키는 능력에도 영향을 미칠 수 있습니다. 이는 Java와 Python 모두에서 DataStream API에서만 사용할 수 있는 연산자에 UID를 설정하는 기능과 같은 특정 기능에 따라 달라집니다. 자세한 내용은 Apache Flink 설명서의 [모든 연산자에 대한 UID 설정을](#) 참조하십시오.

스트리밍 데이터 애플리케이션 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.

- [아파치 플링크용 매니지드 서비스: 작동 방식](#)
- [아파치 플링크용 아마존 매니지드 서비스 \(DataStreamAPI\) 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. 에서 몇 번만 클릭하면 서버리스 노트북을 실행하여 AWS Management Console에서 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](#)

아파치 플링크용 매니지드 서비스: 작동 방식

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

아파치 플링크 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 [테이블 API](#)를 사용합니다. 다른 Apache Flink API도 사용할 수 있지만 스트리밍 애플리케이션을 빌드하는 데는 많이 사용되지 않습니다.

두 API의 특징은 다음과 같습니다.

DataStream API

아파치 플링크 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 다음과 같은 작업을 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 주제 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 [을 참조하십시오. DataStream API](#)

표 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 Source에 연결하여 TableEnvironment을(를) 생성합니다.
- SQL 쿼리 또는 Table API 함수를 사용하여 TableEnvironment에 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 Sink에 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 API](#) 섹션을 참조하세요.

Apache Flink 애플리케이션용 관리형 서비스 만들기

Apache Flink용 관리형 AWS 서비스는 Apache Flink 애플리케이션을 호스팅하기 위한 환경을 만들고 다음과 같은 설정을 제공하는 서비스입니다.

- [런타임 속성](#): 애플리케이션에 제공할 수 있는 파라미터. 애플리케이션 코드를 다시 컴파일하지 않고도 이러한 파라미터를 변경할 수 있습니다.
- [내결합성](#): 애플리케이션이 인터럽트 및 재시작으로부터 복구되는 방법
- [로깅 및 모니터링](#): 애플리케이션이 이벤트를 로그에 기록하는 방법. CloudWatch
- [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 참조하세요.

아파치 플링크용 매니지드 서비스 애플리케이션 만들기

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

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 매니지드 서비스 구축 애플리케이션 코드](#)
- [Apache Flink 애플리케이션용 관리형 서비스 만들기](#)
- [Apache Flink용 매니지드 서비스 애플리케이션 시작](#)
- [Apache Flink용 매니지드 서비스 애플리케이션 확인](#)
- [Apache Flink용 관리형 서비스 애플리케이션의 시스템 롤백 활성화](#)

Apache Flink용 매니지드 서비스 구축 애플리케이션 코드

이 섹션에서는 Apache Flink용 관리 서비스 애플리케이션의 애플리케이션 코드를 빌드하는 데 사용하는 구성 요소에 대해 설명합니다.

지원되는 최신 버전의 Apache Flink를 애플리케이션 코드에 사용하는 것이 좋습니다. Managed Service for Apache Flink 애플리케이션 업그레이드에 대한 자세한 내용을 알아보려면 [???](#) 섹션을 참조하십시오.

[Apache Maven](#)을 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.

이제 애플리케이션은 모든 Scala 버전에서 Java API를 사용할 수 있습니다. 선택한 Scala 표준 라이브러리를 Scala 애플리케이션에 번들로 제공해야 합니다.

Apache Beam을 사용하는 Managed Service for Apache Flink 애플리케이션을 만드는 방법에 대한 자세한 내용을 알아보려면 [Apache Beam 사용](#) 섹션을 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. 파라미터와 함께 아파치 플링크 버전을 제공합니다. `-Dflink.version` 예를 들어 아파치 플링크 1.19.1을 사용하는 경우 다음을 제공합니다.

```
mvn package -Dflink.version=1.19.1
```

이전 버전의 Apache Flink를 사용하여 애플리케이션을 빌드하려면 을 참조하십시오. [이전 버전](#)

Apache Flink 애플리케이션용 관리형 서비스 만들기

애플리케이션 코드를 빌드한 후에는 다음을 수행하여 Apache Flink용 관리형 서비스 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 “아파치 플링크 스트리밍 자바 코드를 업로드하세요.”](#) 섹션을 참조하십시오.
- Managed Service for Apache Flink 애플리케이션 만들기: 다음 방법 중 하나를 사용하여 Managed Service for Apache Flink 애플리케이션을 만드십시오.
 - 콘솔을 사용하여 Apache Flink용 관리 서비스 애플리케이션 만들기: AWS 콘솔을 사용하여 애플리케이션을 만들고 구성할 수 있습니다. AWS

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 폴다운에서 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.

- CLI를 사용하여 Apache Flink용 관리 서비스 애플리케이션 생성: AWS CLI를 사용하여 애플리케이션을 생성하고 구성할 수 있습니다. AWS

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”](#) 섹션을 참조하세요.

Note

기존 애플리케이션을 변경할 수 RuntimeEnvironment 있습니다. 자세한 방법은 [아파치 플링크의 인플레이스 버전 업그레이드\(을\)](#)를 참조하세요.

Apache Flink용 매니지드 서비스 애플리케이션 시작

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

애플리케이션을 시작하려면 다음 방법 중 하나를 사용합니다.

- 콘솔을 사용하여 Apache Flink용 관리 서비스 애플리케이션 시작: AWS 콘솔의 애플리케이션 페이지에서 실행을 선택하여 애플리케이션을 실행할 수 있습니다. AWS
- AWS API를 사용하여 Apache Flink용 관리 서비스 애플리케이션 시작: 작업을 사용하여 애플리케이션을 실행할 수 있습니다. [StartApplication](#)

Apache Flink용 매니지드 서비스 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 [로그 및 모니터링](#)을 참조하십시오.
- 지표 사용 CloudWatch : CloudWatch 지표를 사용하여 애플리케이션의 활동이나 애플리케이션이 입력 또는 출력에 사용하는 리소스 (예: Kinesis 스트림, 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.

Apache Flink용 관리형 서비스 애플리케이션의 시스템 롤백 활성화

시스템 롤백 기능을 사용하면 Apache Flink용 Amazon 관리형 서비스에서 실행 중인 Apache Flink 애플리케이션의 가용성을 높일 수 있습니다. 이 구성을 선택하면 코드나 구성 버그와 같은 작업이 실행될 때 서비스에서 애플리케이션을 이전 실행 버전으로 자동으로 되돌릴 수 있습니다.

UpdateApplication autoscaling

Note

시스템 롤백 기능을 사용하려면 애플리케이션을 업데이트하여 옵트인해야 합니다. 기존 애플리케이션은 기본적으로 시스템 롤백을 자동으로 사용하지 않습니다.

작동 방식

업데이트 또는 조정 작업과 같은 애플리케이션 작업을 시작하면 Apache Flink용 Amazon Managed Service에서 먼저 해당 작업을 실행하려고 시도합니다. 코드 버그나 권한 부족 등 작업 성공을 방해하는 문제를 감지하면 서비스가 자동으로 작업을 시작합니다. RollbackApplication

롤백은 관련 애플리케이션 상태와 함께 애플리케이션을 성공적으로 실행된 이전 버전으로 복원하려고 시도합니다. 롤백이 성공하면 애플리케이션은 이전 버전을 사용하여 가동 중지 시간을 최소화하면서 데이터를 계속 처리합니다. 자동 롤백도 실패할 경우 Amazon Managed Service for Apache Flink는 애플리케이션을 READY 상태로 전환하므로 사용자는 오류 수정 및 작업 재시도를 비롯한 추가 조치를 취할 수 있습니다.

자동 시스템 롤백을 사용하려면 옵트인해야 합니다. 이 시점부터 애플리케이션의 모든 작업에 대해 콘솔 또는 API를 사용하여 활성화할 수 있습니다.

다음 예제 UpdateApplication 작업 요청은 애플리케이션에 대한 시스템 롤백을 활성화합니다.

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationSystemRollbackConfigurationUpdate": {
      "RollbackEnabledUpdate": "true"
    }
  }
}
```

일반적인 시나리오

다음 시나리오는 자동 시스템 롤백이 유용한 경우를 보여줍니다.

- 응용 프로그램 업데이트: 기본 방법을 통해 Flink 작업을 초기화할 때 버그가 있는 새 코드로 응용 프로그램을 업데이트하면 자동 롤백을 통해 이전 작업 버전을 복원할 수 있습니다. 시스템 롤백이 유용한 기타 업데이트 시나리오는 다음과 같습니다.
 - [애플리케이션이 MaxParallelism보다 높은 병렬 처리량으로 실행되도록 업데이트된 경우](#)
 - 애플리케이션이 VPC 애플리케이션에 대해 잘못된 서브넷으로 실행되도록 업데이트되어 Flink 작업 시작 중에 오류가 발생하는 경우
- Flink 버전 업그레이드: 새 Apache Flink 버전으로 업그레이드할 때 업그레이드된 애플리케이션에서 스냅샷 호환성 문제가 발생하는 경우 시스템 롤백을 통해 이전 Flink 버전으로 자동으로 되돌릴 수 있습니다.
- AutoScaling: 스냅샷과 Flink 작업 그래프 간의 운영자 불일치로 인해 애플리케이션이 확장되지만 세이브 포인트에서 복원하는 데 문제가 발생하는 경우.

운영 API

가시성을 높이기 위해 Apache Flink용 Amazon Managed Service에는 장애 및 관련 시스템 롤백을 추적하는 데 도움이 되는 애플리케이션 운영과 관련된 두 개의 API가 있습니다.

ListApplicationOperations

이 API는, UpdateApplication MaintenanceRollbackApplication, 및 기타 작업을 포함하여 애플리케이션에서 수행된 모든 작업을 시간 역순으로 나열합니다. 다음 예제 ListApplicationOperations 작업 요청에는 해당 애플리케이션의 처음 10개 애플리케이션 작업이 나열되어 있습니다.

```
{
  "ApplicationName": "MyApplication",
  "Limit": 10
}
```

다음 예제 ListApplicationOperations 요청은 목록을 애플리케이션의 이전 업데이트로 필터링하는 데 도움이 됩니다.

```
{
  "ApplicationName": "MyApplication",
  "operation": "UpdateApplication"
}
```

DescribeApplicationOperation

이 API는 해당하는 경우 실패 이유를 ListApplicationOperations 포함하여 나열된 특정 작업에 대한 세부 정보를 제공합니다. 다음 예제 DescribeApplicationOperation 작업 요청은 특정 애플리케이션 작업에 대한 세부 정보를 나열합니다.

```
{
  "ApplicationName": "MyApplication",
  "OperationId": "xyzoperation"
}
```

문제 해결 정보는 [시스템 롤백 모범 사례](#)를 참조하세요.

아파치 플링크용 매니지드 서비스 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

션 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 내용은 Apache Flink 설명서의 [작업 및 예약](#)을 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. 응용 프로그램이 일정 시간 동안 일시적인 상태로 유지되는 경우 Force 매개 변수가 로 설정된 [StopApplication](#) 작업을 사용하여 응용 프로그램을 중지할 수 있습니다. true 이러한 상태에는 다음이 포함됩니다.
 - STARTING: [StartApplication](#) 작업 후에 발생합니다. 애플리케이션이 READY 상태에서 RUNNING 상태로 전환되고 있습니다.
 - STOPPING: [StopApplication](#) 작업 후에 발생합니다. 애플리케이션이 RUNNING 상태에서 READY 상태로 전환되고 있습니다.
 - DELETING: [DeleteApplication](#) 작업 후에 발생합니다. 애플리케이션을 삭제하는 중입니다.
 - UPDATING: [UpdateApplication](#) 작업 후에 발생합니다. 애플리케이션이 업데이트 중이며 RUNNING 또는 READY 상태로 다시 전환됩니다.
 - AUTOSCALING: 응용 프로그램의 AutoScalingEnabled 속성이 로 [ParallelismConfiguration](#) true 설정되어 있고 서비스가 응용 프로그램의 병렬성을 높이고 있습니다. 애플리케이션이 이 상태인 경우 사용할 수 있는 유효한 API 작업은 Force 매개 변수가 로 설정된 [StopApplication](#) 작업뿐입니다. true 자동 조정에 대한 자세한 내용을 알아보려면 [자동 크기 조정](#) 섹션을 참조하세요.
 - FORCE_STOPPING: Force 파라미터가 로 설정된 상태에서 [StopApplication](#) 액션을 호출한 후에 발생합니다 true. 애플리케이션을 강제 중지하는 중입니다. 애플리케이션이 STARTING, UPDATING, STOPPING 또는 AUTOSCALING 상태에서 READY 상태로 전환됩니다.

- **ROLLING_BACK**: [RollbackApplication](#) 액션이 호출된 후 발생합니다. 애플리케이션이 이전 버전으로 회귀하는 중입니다. 애플리케이션이 UPDATING 또는 AUTOSCALING 상태에서 RUNNING 상태로 전환됩니다.
- Managed Service for Apache Flink가 애플리케이션에 패치를 적용하는 동안 **MAINTENANCE**: 이 발생합니다. 자세한 정보는 [정비](#)을 참조하세요.

콘솔이나 [DescribeApplication](#) 액션을 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 애플리케이션은 FAILING 상태가 됩니다.

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

애플리케이션 CloudWatch 로그에서 상태 변경을 검토하여 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 내용을 알아보려면 [작업 및 예약](#)을 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.

아파치 Flink 애플리케이션 리소스를 위한 매니지드 서비스

Apache Flink용 관리형 AWS 서비스는 Apache Flink 애플리케이션을 호스팅하기 위한 환경을 만드는 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 내용을 알아보려면 [스케일링](#) 섹션을 참조하세요.

아파치 플링크 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 내용은 Apache Flink 설명서의 [작업 스케줄링](#)을 참조하십시오.

운영자 병렬 처리

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, 기본적으로 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.

연산자에 대한 자세한 내용은 Apache Flink [설명서의 연산자를](#) 참조하십시오.

오퍼레이터 체이닝

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 꼭짓점으로 표시됩니다.

DataStream API

아파치 플링크 애플리케이션은 [아파치 플링크 DataStream API](#)를 사용하여 데이터 스트림의 데이터를 변환합니다.

이 섹션은 다음 주제를 포함합니다:

- [API를 사용하여 Apache Flink용 관리 서비스에서 커넥터를 사용하여 데이터 이동 DataStream](#): 이러한 구성 요소는 애플리케이션과 외부 데이터 소스 및 목적지 간에 데이터를 이동합니다.
 - [API와 함께 Apache Flink용 관리 서비스에서 연산자를 사용하여 데이터 변환 DataStream](#): 이러한 구성 요소는 응용 프로그램 내의 데이터 요소를 변환하거나 그룹화합니다.
 - [API를 사용하여 Apache Flink용 매니지드 서비스에서 이벤트 추적하기 DataStream](#): 이 항목에서는 Apache Flink용 관리형 서비스가 API를 사용할 때 이벤트를 추적하는 방법에 대해 설명합니다.
- DataStream

API를 사용하여 Apache Flink용 관리 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**: 애플리케이션의 데이터를 Kinesis 데이터 스트림, Firehose 스트림 또는 기타 데이터 대상으로 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 내용을 알아보려면 [Apache Flink 설명서의 커넥터](#)를 참조하세요.

⚠ Warning

Flink 1.6, 1.8, 1.11 또는 1.13에서 실행되는 애플리케이션이 중동 (UAE), 아시아 태평양 (하이데라바드), 이스라엘 (텔아비브), 유럽 (취리히), 중동 (UAE), 아시아 태평양 (멜버른) 또는 아시아 태평양 (자카르타) 지역에서 실행하려는 경우 업데이트된 커넥터로 애플리케이션 아카이브를 재구축하거나 Flink 1.18로 업그레이드해야 할 수 있습니다.

Apache Flink 커넥터는 자체 오픈 소스 리포지토리에 저장됩니다. 버전 1.18 이상으로 업그레이드하는 경우 종속성을 업데이트해야 합니다. Apache Flink AWS 커넥터의 저장소에 액세스하려면 [flink-connector-aws](#)을 참조하십시오. 다음은 권장 가이드라인입니다.

커넥터 업그레이드

커넥터 사용	해결 방법
1 EFO	Apache Flink용 Amazon 매니지드 서비스 버전 1.15로 업그레이드할 때는 최신 EFO 커넥터를 사용하고 있는지 확인하십시오. 버전 1.15.3 이상이어야 합니다.

F 커넥터 사용	해결 방법
	<p>자세한 내용은 다음을 참조하세요.</p> <p>FLINK-29324.</p>
1 아마존 데이터 파이어호스 싱크	<p>Apache Flink용 Amazon 매니지드 서비스 버전 1.15로 업그레이드할 때는 최신 Amazon Data Firehose Sink를 사용하고 있는지 확인하십시오.</p> <p>아마존 데이터 파이어호스 싱크</p>
1 카프카 커넥터	<p>Apache Flink용 Amazon 매니지드 서비스 버전 1.15로 업그레이드할 때는 최신 Kafka 커넥터 API를 사용하고 있는지 확인하십시오. 아파치 플링크는 더 이상 사용되지 않으며. FlinkKafkaConsumer FlinkKafkaProducer Kafka 싱크용 이러한 API는 플링크 1.15용 Kafka에 커밋할 수 없습니다. FlinkKafkaSource를 사용하고 있는지 확인하세요. KafkaSourceKafkaSink</p>

F 커넥터 사용 버 지	해결 방법
1 Firehose	<p>애플리케이션은 최신 지역을 인식하지 못하는 오래된 버전의 Firehose 커넥터를 사용합니다. AWS Firehose 커넥터 버전 2.1.0으로 애플리케이션 아카이브를 다시 구축하세요.</p> <p>v2.1.0</p>
1 Kinesis	<p>애플리케이션은 최신 지역을 인식하지 못하는 오래된 버전의 Flink Kinesis 커넥터를 사용합니다. AWS Flink Kinesis 커넥터 버전 1.6.1을 사용하여 애플리케이션 아카이브를 다시 구축하세요.</p> <p>amazon-kinesis-connector-flink https://github.com/aws-labs/amazon-kinesis-connector-flink/tree/1.6.1</p>
1 Kinesis	<p>애플리케이션은 최신 지역을 인식하지 못하는 오래된 버전의 Flink Kinesis 커넥터를 사용합니다. AWS Flink Kinesis 커넥터 버전 2.4.1을 사용하여 애플리케이션 아카이브를 다시 구축하세요.</p> <p>https://github.com/aws-labs/amazon-kinesis-connector-flink/tree/2.4.1</p>

F 커넥터 사용

해결 방법

1 Kinesis

:
1

애플리케이션은 최신 지역을 인식하지 못하는 오래된 버전의 Flink Kinesis 커넥터를 사용합니다. AWS 안타깝게도 Flink는 더 이상 1.6/1.13 커넥터에 대한 패치나 버그 수정을 릴리스하지 않습니다. Flink 1.15로 애플리케이션 아카이브를 다시 구축하여 Flink 1.15로 업데이트하는 것이 좋습니다.

Apache Flink용 매니지드 서비스에 스트리밍 데이터 소스 추가

Apache Flink는 파일, 소켓, 컬렉션, 맞춤 소스에서 읽을 수 있는 커넥터를 제공합니다. 애플리케이션 코드에서 [Apache Flink 소스](#)를 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 FlinkKinesisConsumer를 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 내용을 알아보려면 [Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오](#) 섹션을 참조하세요.

EFO 컨슈머를 사용하는 FlinkKinesisConsumer 생성

는 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: Kinesis 소비자를 위한 향상된 팬아웃을 AWS 참조하십시오](#).

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 FlinkKinesisConsumer를 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 KafkaSource를 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 내용을 알아보려면 [MSK 복제](#) 섹션을 참조하세요.

Apache Flink용 매니지드 서비스에서 싱크를 사용하여 데이터 쓰기

애플리케이션 코드에서 모든 [Apache Flink 싱크](#) 커넥터를 사용하여 Kinesis Data Streams 및 DynamoDB와 같은 AWS 서비스를 포함한 외부 시스템에 쓸 수 있습니다.

또한 Apache Flink는 파일 및 소켓용 싱크를 제공하며, 사용자 지정 싱크를 구현할 수 있습니다. 지원되는 여러 싱크 중에서 자주 사용되는 싱크는 다음과 같습니다.

Kinesis Data Streams

Apache Flink는 Apache Flink 설명서에서 [Kinesis Data Streams 커넥터](#)에 대한 정보를 제공합니다.

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

아파치 카프카 및 아마존 매니지드 스트리밍 (MSK) 용 아파치 카프카

[Apache Flink Kafka 커넥터](#)는 [Apache Kafka](#) 및 Amazon MSK에 데이터를 게시하기 위한 광범위한 지원을 제공하며, 여기에는 정확히 한 번만 적용되는 보장이 포함됩니다. [Kafka에 쓰는 방법을 알아보려면 Apache Flink 설명서에서 Kafka 커넥터 예제를 참조하십시오.](#)

Amazon S3

Amazon S3 버킷에 객체를 쓰는 데 Apache Flink StreamingFileSink를 사용할 수 있습니다.

S3에 객체를 쓰는 방법에 대한 예는 [the section called “S3 싱크”](#) 섹션을 참조하세요.

Firehose

[Firehose 서비스를 사용하여 애플리케이션 출력을 저장하기 위한 안정적이고 확장 가능한 Apache Flink FlinkKinesisFirehoseProducer 싱크입니다.](#) 이 섹션에서는 Maven 프로젝트를 설정하여 FlinkKinesisFirehoseProducer를 생성하고 사용하는 방법을 설명합니다.

주제

- [FlinkKinesisFirehoseProducer 생성](#)

- [FlinkKinesisFirehoseProducer 코드 예](#)

FlinkKinesisFirehoseProducer 생성

다음 코드 예는 FlinkKinesisFirehoseProducer를 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 예제는 Apache Flink 데이터 스트림을 만들고 구성된 FlinkKinesisFirehoseProducer 다음 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");  
}  
}
```

Firehose 싱크 사용 방법에 대한 전체 자습서는 [을 참조하십시오. the section called “Firehose 싱크”](#)

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 내용은 Apache Flink [설명서의 비동기 I/O](#)를 참조하십시오.

API와 함께 Apache Flink용 관리 서비스에서 연산자를 사용하여 데이터 변환 DataStream

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 내용은 Apache Flink [설명서의 연산자](#)를 참조하십시오.

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;
    });
```

코드 예제에 대한 자세한 내용은 [을 참조하십시오](#) [예제](#).

API를 사용하여 Apache Flink용 매니지드 서비스에서 이벤트 추적하기 DataStream

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.

를 사용하여 스트리밍 환경에서 사용하는 시간을 설정합니다. `setStreamTimeCharacteristic`

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

타임스탬프에 대한 자세한 내용은 Apache Flink 설명서의 [워터마크 생성](#)을 참조하십시오.

표 API

Apache Flink 애플리케이션은 [Apache Flink 표 API](#)를 사용하여 관계형 모델을 사용하여 스트림의 데이터와 상호 작용합니다. 표 API를 사용하여 표 소스를 사용하여 데이터에 액세스한 다음 표 함수를 사용하여 표 데이터를 변환하고 필터링합니다. API 함수 또는 SQL 명령을 사용하여 표 형식 데이터를 변환하고 필터링할 수 있습니다.

이 섹션은 다음 주제를 포함합니다:

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

테이블 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.

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

테이블 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);
```

테이블 소스에 대한 자세한 내용은 Apache Flink 설명서의 [테이블 및 SQL 커넥터를](#) 참조하십시오.

테이블 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 내용은 Apache [Flink 설명서에서 지원되는 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 목적지와 상호 작용하기 위해 자체 소스 및 싱크를 정의할 수 있습니다. 자세한 내용은 Apache Flink [설명서의 사용자 정의 소스 및 싱크](#)를 참조하십시오.

테이블 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 Table API가 레코드 시간을 기반으로 창을 생성할 때 메서드를 사용하여 이러한 타임스탬프 중 어떤 타임스탬프를 사용할지 정의합니다. `setStreamTimeCharacteristic`

테이블 API에서 타임스탬프를 사용하는 방법에 대한 자세한 내용은 Apache Flink 설명서의 [시간 속성 및 적시 스트림 처리](#)를 참조하십시오.

Managed Service for Apache Flink와 함께 Python 사용하기

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 개발](#)을 참조하십시오.

아파치 플링크 버전 1.18.1에는 Python 버전 3.10을 사용하여 응용 프로그램을 만들 수 있는 지원이 포함되어 있습니다. 자세한 내용은 [Flink Python 문서](#)를 참조하십시오. 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 테이블 API에 대한 자세한 내용은 아파치 플링크 문서의 [테이블 API 튜토리얼](#)을 참조하십시오.

Python용 아파치 플링크용 매니지드 서비스 프로그래밍 애플리케이션

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.
- StatementSet을 사용하여 대상에 출력 표를 작성합니다.

Managed Service for Apache Flink에서 Python 표 API를 사용하여 시작하려면 [Python용 Apache Flink용 아마존 매니지드 서비스 시작하기](#) 섹션을 참조하십시오.

스트리밍 데이터 읽기 및 쓰기

스트리밍 데이터를 읽고 쓰려면 표 환경에서 SQL 쿼리를 실행합니다.

테이블 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 CreateTableSQL 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 StatementSet를 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](#) 액션 사용.
- [UpdateApplication](#) 액션 사용.
- 콘솔을 사용하여 애플리케이션 구성하기.

Managed Service for Apache Flink 런타임에서 생성하는 application_properties.json이라는 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.

아파치 플링크용 매니지드 서비스 Python 애플리케이션 만들기

코드 파일 지정

애플리케이션의 코드 패키지를 만들었으면 Amazon S3 버킷에 업로드합니다. 그런 다음 콘솔이나 [CreateApplication](#) 액션을 사용하여 애플리케이션을 생성합니다.

[CreateApplication](#) 액션을 사용하여 응용 프로그램을 만들 때는 라는 특수 응용 프로그램 속성 그룹을 사용하여 zip 파일의 코드 파일과 아카이브를 지정합니다. `datakinesis.analytics.flink.run.options`. 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 내용은 Apache [Flink 설명서의 명령줄 인터페이스](#)를 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"
        }
      }
    ],
  },
}
```

아파치 플링크용 Python 매니지드 서비스 애플리케이션 모니터링

애플리케이션의 CloudWatch 로그를 사용하여 Apache Flink Python용 관리형 서비스 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.

Insights를 사용하여 로그를 쿼리합니다. CloudWatch

다음 CloudWatch Insights 쿼리는 애플리케이션의 기본 기능을 실행하는 동안 Python 진입점에서 생성된 로그를 검색합니다.

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

Apache Flink용 관리 서비스의 런타임 속성

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

이 주제는 다음 섹션을 포함하고 있습니다:

- [콘솔에서 런타임 속성으로 작업하기](#)
- [CLI에서 런타임 속성 사용](#)
- [Apache Flink용 관리 서비스 애플리케이션의 런타임 속성에 액세스](#)

콘솔에서 런타임 속성으로 작업하기

를 사용하여 Apache Flink용 관리 서비스 애플리케이션에서 런타임 속성을 추가, 업데이트 또는 제거할 수 있습니다. AWS Management Console

Note

지원되는 이전 버전의 Apache Flink를 사용 중이고 기존 애플리케이션을 Apache Flink 1.19.1로 업그레이드하려면 인플레이스 Apache Flink 버전 업그레이드를 사용하면 됩니다. 인플레이스 버전 업그레이드를 사용하면 스냅샷, 로그, 메트릭, 태그, Flink 구성 등을 포함하여 Apache Flink 버전 전반에서 단일 ARN에 대해 애플리케이션을 추적할 수 있습니다. 이 기능은 모든 상태에서 사용할 수 있습니다. RUNNING READY 자세한 정보는 [아파치 플링크의 인플레이스 버전 업그레이드](#)을 참조하세요.

Managed Service for Apache Flink에 대한 런타임 속성 업데이트

1. <https://console.aws.amazon.com/flink>에서 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 카값 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 사용

[AWS CLI](#)를 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 [아파치 플링크용 매니지드 서비스 API 예제 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_19",
  "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.

Apache Flink용 관리 서비스 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;
}
```

코드 예제는 [Managed Service for Apache Flink: 예제](#) 섹션을 참조하세요.

아파치 플링크 커넥터 사용

Apache Flink 커넥터는 Apache Flink용 Amazon 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, Apache Kafka 주제, 파일 또는 기타 데이터 소스에서 애플리케이션에 데이터를 제공합니다.
- 싱크: 애플리케이션의 데이터를 Kinesis 데이터 스트림, Firehose 스트림, Apache Kafka 주제 또는 기타 데이터 대상으로 전송합니다.

- 비동기 I/O: 데이터베이스와 같은 데이터 소스에 대한 비동기 액세스를 제공하여 스트림을 보강합니다.

Apache Flink 커넥터는 자체 소스 리포지토리에 저장됩니다. Apache Flink 커넥터의 버전과 아티팩트는 사용 중인 Apache Flink 버전과 사용 중인 API, 테이블 또는 SQL API 중 어떤 것을 사용하는지에 따라 달라집니다. [DataStream](#)

아파치 플링크용 아마존 매니지드 서비스는 40개 이상의 사전 구축된 아파치 플링크 소스 및 싱크 커넥터를 지원합니다. 다음 표에는 가장 많이 사용되는 커넥터와 관련 버전이 요약되어 있습니다. Async-Sink 프레임워크를 사용하여 사용자 지정 싱크를 만들 수도 있습니다. 자세한 내용은 [Apache Flink 설명서의 일반 비동기 베이스 싱크](#)를 참조하십시오.

Apache Flink 커넥터의 저장소에 액세스하려면 [AWS flink-connector-aws](#)을 참조하십시오.

플링크 버전용 커넥터

커넥터	플링크 버전 1.15	플링크 버전 1.18	플링크 버전 1.19
Kinesis 데이터 스트림 - 소스 - DataStream 및 테이블 API	flink-connector-kinesis, 1.15.4	flink-connector-kinesis, 4.3.0-1.18	flink-connector-kinesis, 4.3.0-1.19
Kinesis 데이터 스트림 - 싱크 - DataStream 및 테이블 API	flink-connector-aws-kinesis-스트림, 1.15.4	flink-connector-aws-kinesis-스트림, 4.3.0-1.18	flink-connector-aws-kinesis-스트림, 4.3.0-1.19
Kinesis Data Streams - 소스/싱크 - SQL	flink-sql-connector-kinesis, 1.15.4	flink-sql-connector-kinesis, 4.3.0-1.18	flink-sql-connector-kinesis, 4.3.0-1.19
카프카 - 및 테이블 API DataStream	flink-connector-kafka, 1.15.4	flink-connector-kafka, 3.2.0-1.18	flink-connector-kafka, 3.2.0-1.19
카프카 - SQL	flink-sql-connector-kafka, 1.15.4	flink-sql-connector-kafka, 3.2.0-1.18	flink-sql-connector-kafka, 3.2.0-1.19
Firehose - DataStream 및 테이블 API	flink-connector-aws-kinesis-파이어호스, 1.15.4	flink-connector-aws-firehose, 4.3.0-1.18	flink-connector-aws-firehose, 4.3.0-1.19

커넥터	플링크 버전 1.15	플링크 버전 1.18	플링크 버전 1.19
Firehose - SQL	flink-sql-connector-aws- 중국어 - 파이어 호스, 1.15.4	flink-sql-connector-aws- 파이어 호스, 4.3.0-1.18	flink-sql-connector-aws- 파이어호스, 4.3.0-1.19
DynamoDB DataStream - 및 테이블 API	flink-connector-dynamodb, 3.0.0-1.15	flink-connector-dynamodb, 4.3.0-1.18	flink-connector-dynamodb, 4.3.0-1.19
DynamoDB - SQL	flink-sql-connector-dynamodb, 3.0.0-1.15	flink-sql-connector-dynamodb, 4.3.0-1.18	flink-sql-connector-dynamodb, 4.3.0-1.19
OpenSearch - 및 테이블 API DataStream	-	flink-connector-opensearch, 1.2.0-1.18	flink-connector-opensearch, 1.2.0-1.19
OpenSearch - SQL	-	flink-sql-connector-opensearch, 1.2.0-1.18	flink-sql-connector-opensearch, 1.2.0-1.19

Apache Flink용 Amazon 매니지드 서비스의 커넥터에 대한 자세한 내용은 다음을 참조하십시오.

- [DataStream API 커넥터](#)
- [테이블 API 커넥터](#)

Apache Flink용 관리 서비스의 내결함성 구현

체크포인트는 Amazon Managed Service for Apache Flink.에서 내결함성을 구현하는 데 사용하는 방법입니다. 체크포인트는 예상치 못한 애플리케이션 중단이나 페일오버로부터 즉시 복구하는 데 사용되는 실행 중인 애플리케이션의 up-to-date 백업입니다.

[Apache Flink 애플리케이션의 체크포인트에 대한 자세한 내용은 Apache Flink 설명서의 체크포인트를 참조하십시오.](#)

스냅샷은 수동으로 생성하고 관리되는 애플리케이션 상태 백업입니다. 스냅샷으로 [UpdateApplication](#) 호출을 통해 애플리케이션을 이전 상태로 복원할 수 있습니다. 자세한 정보는 [스냅샷을 사용한 애플리케이션 백업 관리](#)을 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 애플리케이션을 사용하여 구성된 경우, 서비스는 애플리케이션 업데이트 중이나 서비스 관련 확장 또는 유지 관리 중에 정확히 한 번의 처리 시맨틱을 제공합니다. [ApplicationRestoreConfiguration](#)

Apache Flink용 관리 서비스의 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`: 6만
- `MinPauseBetweenCheckpoints`: 5000

로 `ConfigurationTypeDEFAULT` 설정된 경우 를 사용하거나 애플리케이션 코드에서 값을 설정하여 다른 값으로 설정하더라도 이전 값이 사용됩니다. AWS Command Line Interface

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.

체크포인트 API 예제

이 섹션에는 애플리케이션의 체크포인트 구성을 위한 API 작업에 대한 예 요청이 포함되어 있습니다. JSON 파일을 사용하여 API 작업을 입력하는 방법에 대한 자세한 방법은 [아파치 플링크용 매니저드 서비스 API 예제 코드](#) 섹션을 참조하세요.

새 애플리케이션을 위한 체크포인트 구성

[CreateApplication](#) 작업을 위한 다음 예 요청은 애플리케이션을 생성할 때 체크포인트 수행을 구성합니다.

```
{
  "ApplicationName": "MyApplication",
  "RuntimeEnvironment": "FLINK-1_19",
  "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 애플리케이션의 체크포인트 비활성화

[CreateApplication](#) 작업을 위한 다음 예 요청은 애플리케이션을 생성할 때 체크포인트 수행을 비활성화합니다.

```
{
  "ApplicationName": "MyApplication",
  "RuntimeEnvironment": "FLINK-1_19",
  "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 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 애플리케이션의 체크포인트 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 세이브포인트에 대한 자세한 내용은 Apache Flink 설명서의 세이브포인트를 참조하십시오.](#) 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 예제](#)

자동 스냅샷 생성

애플리케이션에 대해 로 `true` 설정된 경우 `SnapshotsEnabled` Managed Service [ApplicationSnapshotConfiguration](#) for Apache Flink는 애플리케이션이 업데이트, 확장 또는 중지될 때 스냅샷을 자동으로 생성하고 사용하여 정확히 한 번의 처리 시맨틱을 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. [세이브포인트](#) 트리거링을 참조하십시오.

자동으로 생성된 스냅샷의 품질은 다음과 같습니다.

- 스냅샷은 서비스에서 관리하지만 작업을 사용하여 스냅샷을 볼 수 있습니다.
[ListApplicationSnapshots](#) 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 [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.

응용 프로그램이 호환되지 않는 상태 데이터를 포함하는 스냅샷에서 복원할 수 있도록 [FlinkRunConfiguration](#) 하려면 작업을 true 사용하여 `AllowNonRestoredState` 매개 변수를 로 설정합니다. [UpdateApplication](#)

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 true로 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 내용은 Apache Flink [설명서의 세이브포인트를](#) 참조하십시오. 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. two-phase-commit 싱크의 경우 사용자가 만든 스냅샷 () 대신 시스템 스냅샷 (SW) 을 사용하는 것이 좋습니다. `CreateApplicationSnapshot`

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 트리거](#)를 참조하세요.

기존 세이브포인트 데이터와 호환되지 않는 애플리케이션을 재개해야 하는 경우, 작업 파라미터를 로 설정하여 스냅샷에서의 복원을 건너뛰는 것이 좋습니다. [ApplicationRestoreType StartApplication](#) SKIP_RESTORE_FROM_SNAPSHOT

Apache Flink가 병립될 수 없는 상태 데이터를 처리하는 방법에 대한 자세한 내용을 알아보려면 Apache Flink 설명서의 [상태 스키마 개선](#)을 참조하세요.

스냅샷 API 예제

이 섹션에는 애플리케이션에서 스냅샷을 사용하기 위한 API 작업에 대한 예 요청이 포함되어 있습니다. JSON 파일을 사용하여 API 작업을 입력하는 방법에 대한 자세한 방법은 [아파치 플링크용 매니저 서비스 API 예제 코드](#) 섹션을 참조하세요.

애플리케이션용 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"
}
```

애플리케이션의 스냅샷 목록

[ListApplicationSnapshots](#) 작업을 위한 다음 예 요청은 현재 애플리케이션 상태의 처음 50개 스냅샷이 나열합니다.

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

애플리케이션 스냅샷의 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,
}
```

이름이 지정된 스냅샷을 사용하여 애플리케이션을 다시 시작합니다.

[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

[StartApplication](#) 작업을 위한 다음 예 요청은 가장 최근의 스냅샷을 사용하여 애플리케이션을 시작합니다.

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

스냅샷을 사용하지 않고 애플리케이션 재시작

[StartApplication](#) 작업을 위한 다음 예 요청은 스냅샷이 있더라도 애플리케이션 상태를 로드하지 않고 애플리케이션을 시작합니다.

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

아파치 플링크의 인플레이스 버전 업그레이드

Apache Flink의 인플레이스 버전 업그레이드를 사용하면 Apache Flink 버전 전체에서 단일 ARN에 대해 애플리케이션을 추적할 수 있습니다. 여기에는 스냅샷, 로그, 메트릭, 태그, Flink 구성, 리소스 제한 증가, VPC 등이 포함됩니다. Apache Flink용 Amazon 관리 서비스에서 Apache Flink의 인플레이스 버전 업그레이드를 수행하여 기존 애플리케이션을 새 Flink 버전으로 업그레이드할 수 있습니다. 이 작업을 수행하려면, SDK 또는 를 사용할 수 있습니다. AWS CLI AWS CloudFormation AWS AWS Management Console

Note

Apache Flink Studio용 Amazon 매니지드 서비스에서는 Apache Flink의 인플레이스 버전 업그레이드를 사용할 수 없습니다.

이 주제는 다음 섹션을 포함하고 있습니다.

- [Apache Flink의 인플레이스 버전 업그레이드를 사용하여 애플리케이션을 업그레이드합니다.](#)
- [애플리케이션을 새 Apache Flink 버전으로 업그레이드](#)
- [롤백](#)
- [일반 모범 사례 및 권장 사항](#)
- [예방 조치 및 알려진 문제](#)

Apache Flink의 인플레이스 버전 업그레이드를 사용하여 애플리케이션을 업그레이드합니다.

시작하기 전에 다음 동영상을 시청하는 것이 좋습니다. [인플레이스 버전 업그레이드](#).

Apache Flink의 전체 버전 업그레이드를 수행하려면, AWS CLI, AWS CloudFormation, AWS SDK 또는 `awscli`를 사용할 수 있습니다. AWS Management Console 또는 상태의 Apache Flink용 관리 서비스와 함께 사용하는 모든 기존 애플리케이션에서 이 기능을 사용할 수 있습니다. `READY RUNNING` `UpdateApplication` API를 사용하여 Flink 런타임을 변경하는 기능을 추가합니다.

업그레이드 전: Apache Flink 애플리케이션 업데이트

Flink 애플리케이션을 작성할 때는 종속 항목과 함께 애플리케이션 JAR에 번들로 묶고 JAR을 Amazon S3 버킷에 업로드합니다. 그러면 Apache Flink용 Amazon 관리형 서비스가 사용자가 선택한 새 Flink 런타임에서 작업을 실행합니다. 업그레이드하려는 Flink 런타임과 호환되도록 애플리케이션을 업데이트해야 할 수도 있습니다. Flink 버전 간에 불일치가 발생하여 버전 업그레이드가 실패할 수 있습니다. 소스 (인그레스) 또는 대상 (싱크, 이그레스) 용 커넥터와 Scala 종속성을 사용하는 것이 가장 일반적입니다. Apache Flink용 관리 서비스의 Flink 1.15 이상 버전은 Scala에 구애받지 않으므로 JAR에는 사용하지 않는 Scala 버전이 포함되어 있어야 합니다.

애플리케이션을 업데이트하려면

1. 상태를 통한 애플리케이션 업그레이드에 대한 Flink 커뮤니티의 조언을 읽어보세요. [애플리케이션 및 Flink 버전 업그레이드를 참조하십시오.](#)

2. 알려진 문제 및 제한 사항 목록을 읽어보세요. [예방 조치 및 알려진 문제](#) 섹션을 참조하십시오.
3. 종속성을 업데이트하고 로컬에서 애플리케이션을 테스트하세요. 이러한 종속성은 일반적으로 다음과 같습니다.
 1. 플링크 런타임 및 API.
 2. 새 Flink 런타임에 권장되는 커넥터. 업데이트하려는 특정 런타임의 [릴리스 버전에서](#) 이러한 기능을 찾을 수 있습니다.
 3. 스칼라 — 아파치 플링크는 Flink 1.15부터 Scala에 구애받지 않는다. 애플리케이션 JAR에 사용하려는 스칼라 종속성을 포함해야 합니다.
4. zip 파일에 새 애플리케이션 JAR을 빌드하고 Amazon S3에 업로드합니다. 이전 JAR/ZIP 파일과 다른 이름을 사용하는 것이 좋습니다. 롤백해야 하는 경우 이 정보를 사용하세요.
5. 스테이트풀 애플리케이션을 실행 중인 경우 현재 애플리케이션의 스냅샷을 생성하는 것이 좋습니다. 이렇게 하면 업그레이드 도중이나 업그레이드 후에 문제가 발생할 경우 스테이트풀하게 롤백할 수 있습니다.

애플리케이션을 새 Apache Flink 버전으로 업그레이드

액션을 사용하여 Flink 애플리케이션을 업그레이드할 수 있습니다. [UpdateApplication](#)

다음과 같은 여러 가지 방법으로 UpdateApplication API를 호출할 수 있습니다.

- 에서 기존 구성 워크플로를 사용하십시오 AWS Management Console.
 - 의 앱 페이지로 이동합니다 AWS Management Console.
 - 구성을 선택합니다.
 - 새 런타임과 시작하려는 스냅샷 (구성 복원이라고도 함) 을 선택합니다. 최신 설정을 복원 구성으로 사용하여 최신 스냅샷에서 앱을 시작합니다. Amazon S3에서 새로 업그레이드된 애플리케이션 JAR/Zip을 가리키십시오.
- 애플리케이션 AWS CLI [업데이트](#) 작업을 사용하십시오.
- AWS CloudFormation (CFN) 을 사용하십시오.
 - 필드를 업데이트하세요. [RuntimeEnvironment](#) 이전에는 애플리케이션을 AWS CloudFormation 삭제하고 새 애플리케이션을 생성하여 스냅샷과 기타 앱 기록이 손실되었습니다. 이제 애플리케이션이 삭제되지 않고 현재 RuntimeEnvironment 위치에서 AWS CloudFormation 업데이트됩니다.
- AWS SDK를 사용하세요.
 - 선택한 프로그래밍 언어는 SDK 설명서를 참조하십시오. [UpdateApplication](#) 섹션을 참조하세요.

애플리케이션이 RUNNING 상태에 있거나 애플리케이션이 중지된 상태에서 업그레이드를 수행할 수 있습니다. READY Apache Flink용 Amazon 관리형 서비스는 원본 런타임 버전과 대상 런타임 버전 간의 호환성을 검증하여 확인합니다. 이 호환성 검사는 주 내에서 수행할 [UpdateApplication](#) 때 실행되고, RUNNING 주 내에서 업그레이드하는 [StartApplication](#) 경우 다음 번에 실행됩니다. READY

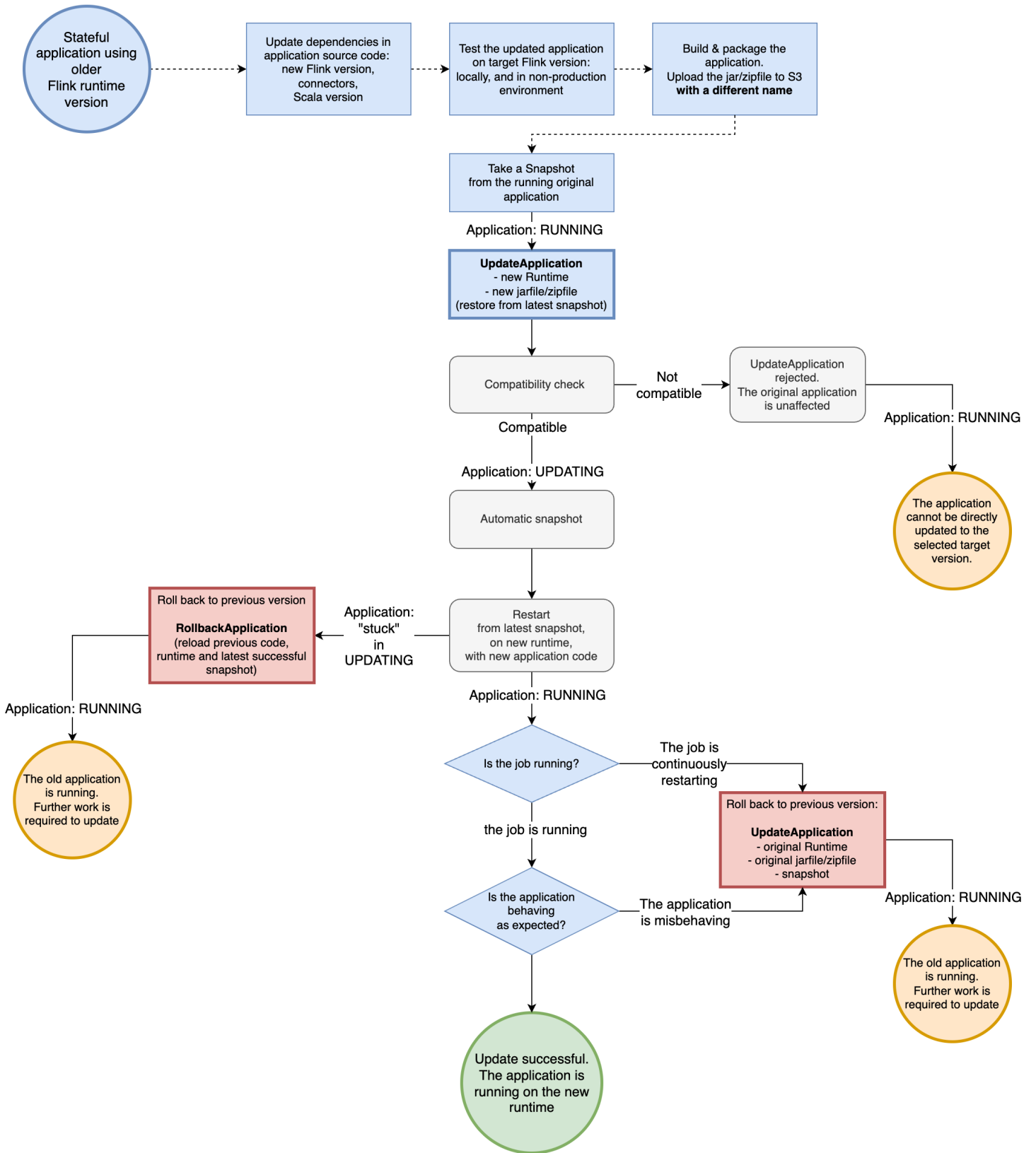
RUNNING 상태의 애플리케이션 업그레이드

다음 예제는 `aws` 를 사용하여 미국 동부 (버지니아 북부) 에서 Flink 1.18이라는 RUNNING UpgradeTest 주의 앱을 AWS CLI 업그레이드하고 최신 스냅샷에서 업그레이드된 앱을 시작하는 방법을 보여줍니다.

```
aws --region us-east-1 kinesisanalyticstv2 update-application \
--application-name UpgradeTest --runtime-environment-update "FLINK-1_18" \
--application-configuration-update '{"ApplicationCodeConfigurationUpdate": '\
'{"CodeContentUpdate": {"S3ContentLocationUpdate": '\
'{"FileKeyUpdate": "flink_1_18_app.jar"}}}' \
--run-configuration-update '{"ApplicationRestoreConfiguration": '\
'{"ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"}}' \
--current-application-version-id ${current_application_version}
```

- 서비스 스냅샷을 활성화하고 최신 스냅샷에서 애플리케이션을 계속 진행하려는 경우, Amazon Managed Service for Apache Flink는 현재 RUNNING 애플리케이션의 런타임이 선택한 대상 런타임과 호환되는지 확인합니다.
- 대상 런타임을 계속할 스냅샷을 지정한 경우 Apache Flink용 Amazon Managed Service는 대상 런타임이 지정된 스냅샷과 호환되는지 확인합니다. 호환성 검사에 실패하면 업데이트 요청이 거부되고 애플리케이션은 변경되지 않은 상태로 유지됩니다. RUNNING
- 스냅샷 없이 애플리케이션을 시작하기로 선택한 경우 Apache Flink용 Amazon Managed Service는 호환성 검사를 실행하지 않습니다.
- 업그레이드된 애플리케이션이 실패하거나 전이 상태에서 멈추는 경우 [롤백](#) 섹션의 지침에 따라 정상 UPDATING 상태로 돌아가십시오.

실행 중인 상태 응용 프로그램의 프로세스 흐름



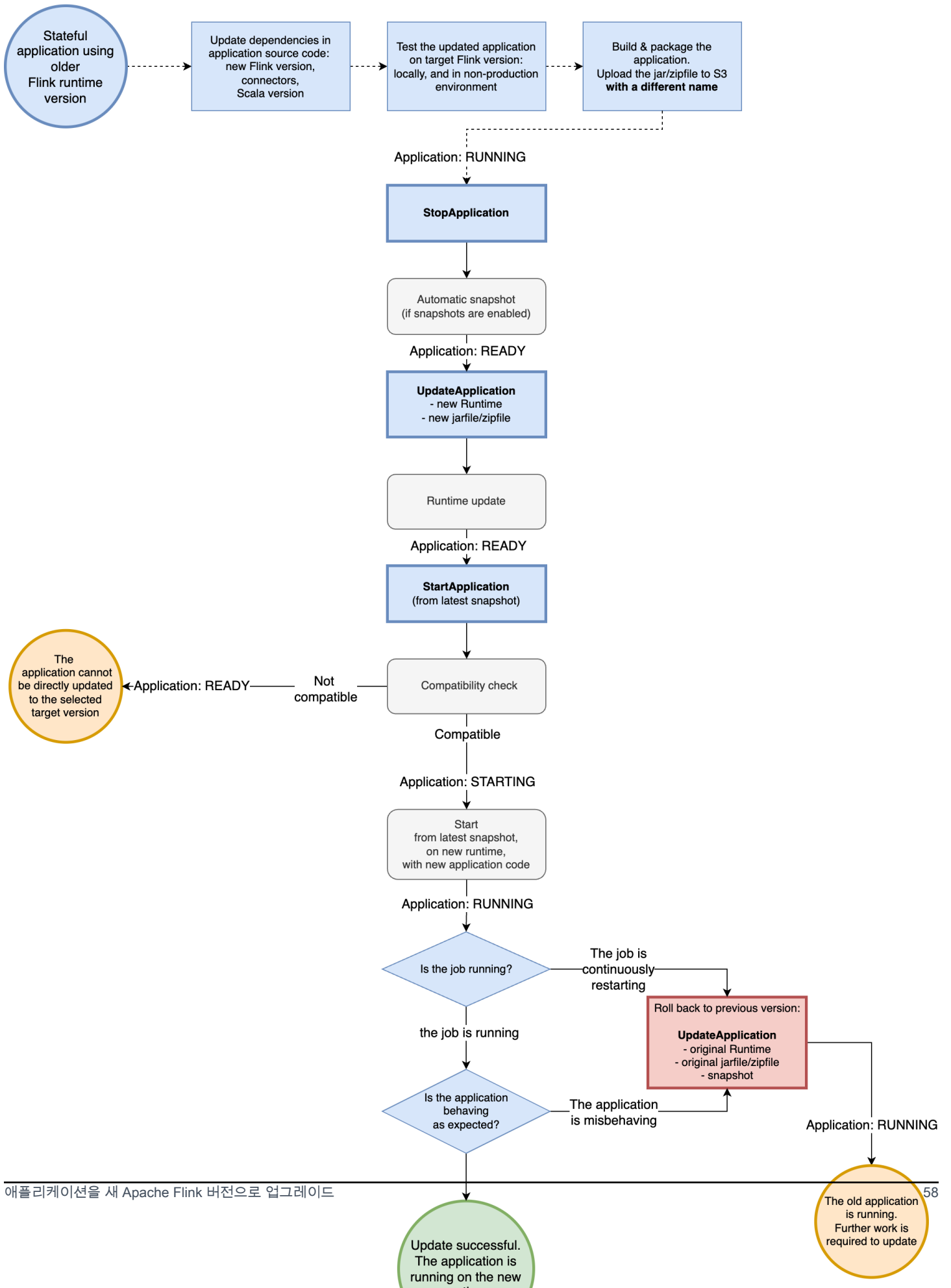
READY 상태의 애플리케이션 업그레이드

다음 예제는 를 사용하여 미국 동부 (버지니아 북부) 에서 Flink 1.18이라는 READY 주의 앱을 UpgradeTest 업그레이드하는 방법을 보여줍니다. AWS CLI 애플리케이션이 실행되고 있지 않기 때문에 앱을 시작하기 위한 스냅샷이 지정되어 있지 않습니다. 애플리케이션 시작 요청을 실행할 때 스냅샷을 지정할 수 있습니다.

```
aws --region us-east-1 kinesisanalyticstv2 update-application \
--application-name UpgradeTest --runtime-environment-update "FLINK-1_18" \
--application-configuration-update '{"ApplicationCodeConfigurationUpdate": '\
'{"CodeContentUpdate": {"S3ContentLocationUpdate": '\
'{"FileKeyUpdate": "flink_1_18_app.jar"}}}' \
--current-application-version-id ${current_application_version}
```

- READY 상태에 있는 애플리케이션의 런타임을 모든 Flink 버전으로 업데이트할 수 있습니다. Apache Flink용 Amazon 관리형 서비스는 애플리케이션을 시작할 때까지 검사를 실행하지 않습니다.
- Apache Flink용 Amazon 관리형 서비스는 앱을 시작하기 위해 선택한 스냅샷에 대해서만 호환성 검사를 실행합니다. 이는 [Flink](#) 호환성 표에 따른 기본 호환성 검사입니다. 스냅샷을 촬영하는 데 사용한 Flink 버전과 대상 Flink 버전만 확인합니다. 선택한 스냅샷의 Flink 런타임이 앱의 새 런타임과 호환되지 않는 경우 시작 요청이 거부될 수 있습니다.

준비 상태 애플리케이션의 프로세스 흐름



롤백

애플리케이션에 문제가 있거나 Flink 버전 간에 애플리케이션 코드가 일치하지 않는 경우,, SDK 또는 를 사용하여 롤백할 수 있습니다. AWS CLI AWS CloudFormation AWS AWS Management Console다 음 예제는 다양한 실패 시나리오에서 롤백이 어떻게 보이는지 보여줍니다.

런타임 업그레이드에 성공했고 응용 프로그램은 **RUNNING** 상태에 있지만 작업이 실패 하고 계속 다시 시작됩니다.

미국 동부 (버지니아 북부) TestApplication 에서 Flink 1.15라는 이름의 스테이트풀 애플리케이션 을 Flink 1.18로 업그레이드하려고 한다고 가정해 보겠습니다. 그러나 업그레이드된 Flink 1.18 응용 프 로그램은 상태에 있음에도 불구하고 시작되지 않거나 계속 다시 시작됩니다. RUNNING 이는 일반적인 실패 시나리오입니다. 추가 다운타임을 방지하려면 애플리케이션을 이전 실행 버전 (Flink 1.15) 으로 즉시 롤백하고 나중에 문제를 진단하는 것이 좋습니다.

애플리케이션을 이전 실행 버전으로 롤백하려면 [rollback-application](#) AWS CLI 명령 또는 API 작업을 사용합니다. [RollbackApplication](#) 이 API 작업은 변경한 내용을 롤백하여 최신 버전으로 업데이트합니 다. 그런 다음 최근에 성공한 스냅샷을 사용하여 애플리케이션을 다시 시작합니다.

업그레이드를 시도하기 전에 기존 앱으로 스냅샷을 찍는 것이 좋습니다. 이렇게 하면 데이터 손실이나 데이터를 재처리해야 하는 상황을 방지하는 데 도움이 됩니다.

이 실패 AWS CloudFormation 시나리오에서는 응용 프로그램을 롤백하지 않습니다. 이전 런타임을 가 리키도록 CloudFormation 템플릿을 업데이트하고 응용 프로그램을 강제로 CloudFormation 업데이트 하려면 이전 코드를 업데이트해야 합니다. 그렇지 않으면 응용 프로그램이 상태로 전환될 때 업데이트 된 것으로 CloudFormation 가정합니다. RUNNING

중단된 애플리케이션 롤백 **UPDATING**

업그레이드 시도 후 애플리케이션이 UPDATING 또는 AUTOSCALING 상태에서 멈추는 경 우 Apache Flink용 Amazon Managed Service는 [rollback-applications](#) AWS CLI 명령 또는 [RollbackApplications](#) API 작업을 제공하여 애플리케이션을 중단 또는 상태 이전의 버전으로 롤백할 수 있습니다. UPDATING AUTOSCALING 이 API는 애플리케이션이 멈추거나 과도기 상태가 되는 원인이 된 변경 사항을 롤백합니다. UPDATING AUTOSCALING

일반 모범 사례 및 권장 사항

- 프로덕션 업그레이드를 시도하기 전에 비프로덕션 환경에서 상태 없이 새 작업/런타임을 테스트하 십시오.

- 먼저 비프로덕션 애플리케이션으로 스테이트풀 업그레이드를 테스트해 보세요.
- 새 작업 그래프가 업그레이드된 애플리케이션을 시작하는 데 사용할 스냅샷과 호환되는 상태인지 확인하십시오.
- 운영자 상태에 저장된 유형이 동일하게 유지되는지 확인하십시오. 유형이 변경된 경우 Apache Flink는 운영자 상태를 복원할 수 없습니다.
- uid메서드를 사용하여 설정한 운영자 ID가 동일하게 유지되는지 확인하십시오. Apache Flink는 운영자에게 고유한 ID를 할당할 것을 강력히 권장합니다. 자세한 내용은 Apache Flink 설명서의 [운영자 ID 할당](#)을 참조하십시오.

운영자에게 ID를 할당하지 않으면 Flink에서 자동으로 ID를 생성합니다. 이 경우 프로그램 구조에 따라 달라질 수 있으며 변경되면 호환성 문제가 발생할 수 있습니다. Flink는 운영자 ID를 사용하여 스냅샷의 상태를 운영자와 일치시킵니다. 운영자 ID를 변경하면 애플리케이션이 시작되지 않거나 스냅샷에 저장된 상태가 삭제되고 새 운영자가 상태 없이 시작합니다.

- 키 상태를 저장하는 데 사용되는 키는 변경하지 마십시오.
- 윈도우 또는 조인과 같은 상태 저장 연산자의 입력 유형을 수정하지 마세요. 이렇게 하면 연산자의 내부 상태 유형이 암시적으로 변경되어 상태 비호환성이 발생합니다.

예방 조치 및 알려진 문제

Flink 1.19 이상에서 구성을 변경할 수 없습니다.

- Flink 1.18 이전 버전에서 Flink 1.19 이상으로 런타임을 업데이트하는 경우 Flink 작업 코드를 사용하여 Flink 작업 구성을 변경하는 것은 더 이상 허용되지 않습니다. 따라서 응용 프로그램은 작업을 제출하지 못합니다. 오류 로그는 런타임 시 수정된 허용되지 않은 구성을 나타냅니다. 자세한 정보는 [FlinkRuntimeException: “허용되지 않은 구성 변경이 감지되었습니다.”](#)을 참조하세요.

상태 호환성에 대한 알려진 제한

- 테이블 API를 사용하는 경우 Apache Flink는 Flink 버전 간의 상태 호환성을 보장하지 않습니다. 자세한 내용은 Apache Flink 설명서의 [스테이트풀 업그레이드 및 에볼루션](#)을 참조하십시오.
- 플링크 1.6 상태는 플링크 1.18과 호환되지 않습니다. 1.6에서 1.18 이상으로 상태를 사용하여 업그레이드하려고 하면 API가 요청을 거부합니다. 1.8, 1.11, 1.13, 1.15로 업그레이드하고 스냅샷을 만든 다음 1.18 이상으로 업그레이드할 수 있습니다. 자세한 내용은 Apache Flink [설명서의 애플리케이션 및 Flink 버전 업그레이드](#)를 참조하십시오.

Flink Kinesis 커넥터의 알려진 문제점

- Flink 1.11 이전 버전을 사용하고 Enhanced-fan-out (EFO) 지원용 `amazon-kinesis-connector-flink` 커넥터를 사용하는 경우 Flink 1.13 이상으로 스테이트풀 업그레이드를 하려면 추가 단계를 거쳐야 합니다. 이는 커넥터의 패키지 이름이 변경되었기 때문입니다. 자세한 내용은 [amazon-kinesis-connector-flink](#)를 참조하세요.

Flink 1.11 및 이전 버전용 `amazon-kinesis-connector-flink` 커넥터는 패키지를 `software.amazon.kinesis` 사용하지만 Flink 1.13 이상의 Kinesis 커넥터는 패키지를 사용합니다. [org.apache.flink.streaming.connectors.kinesis 이 도구를 사용하여 마이그레이션을 지원하십시오: -state-migrator. amazon-kinesis-connector-flink](#)

- Flink 1.13 이전 버전을 사용하고 Flink 1.15 이상으로 업그레이드하는 경우, 스테이트풀 업그레이드를 위해서는 최신 버전이 아닌 Flink 1.15 `FlinkKinesisProducer` 이상에서 계속 사용해야 합니다. `FlinkKinesisProducer KinesisStreamsSink` 하지만 싱크대에 이미 커스텀 uid 세트가 있다면 상태를 유지하지 않기 때문에 전환할 수 있을 것입니다. `KinesisStreamsSink FlinkKinesisProducer` 사용자 지정이 설정되어 있기 때문에 uid Flink는 동일한 연산자로 취급합니다.

스칼라로 작성된 Flink 애플리케이션은

- 플링크 1.15부터 아파치 플링크는 런타임에 스칼라를 포함하지 않습니다. Flink 1.15 이상으로 업그레이드할 때는 사용하려는 스칼라 버전과 기타 스칼라 종속 항목을 코드 JAR/ZIP에 포함해야 합니다. 자세한 내용은 아파치 플링크용 [아마존 매니지드 서비스 \(Apache Flink\) 1.15.2 릴리스](#)를 참조하십시오.
- 애플리케이션에서 스칼라를 사용하고 플링크 1.11 이전 버전 (스칼라 2.11) 에서 플링크 1.13 (스칼라 2.12) 으로 업그레이드하는 경우, 코드에 스칼라 2.12가 사용되는지 확인하십시오. 그렇지 않으면 Flink 1.13 애플리케이션이 Flink 1.13 런타임에서 Scala 2.11 클래스를 찾지 못할 수 있습니다.

Flink 애플리케이션을 다운그레이드할 때 고려할 사항

- Flink 응용 프로그램을 다운그레이드할 수 있지만 이전에 응용 프로그램을 이전 Flink 버전에서 실행했던 경우로 제한됩니다. 스테이트풀 업그레이드의 경우 Apache Flink용 Managed Service for Apache Flink에서는 다운그레이드에 해당 버전 또는 이전 버전으로 찍은 스냅샷을 사용해야 합니다.
- 런타임을 Flink 1.13 이상에서 Flink 1.11 또는 이전 버전으로 업데이트하고 앱에서 HashMap 상태 백엔드를 사용하는 경우 애플리케이션이 계속 실패합니다.

Apache Flink용 매니지드 서비스의 애플리케이션 스케일링

Amazon Managed Service for Apache Flink의 작업 병렬 실행 및 리소스 할당을 구성하여 규모 조정을 구현할 수 있습니다. Apache Flink가 작업의 병렬 인스턴스를 스케줄링하는 방법에 대한 자세한 내용은 Apache Flink 설명서의 [병렬 실행](#)을 참조하십시오.

주제

- [애플리케이션 병렬 처리 및 KPU 구성 ParallelismPer](#)
- [Kinesis 처리 단위 할당](#)
- [애플리케이션의 병렬성 업데이트](#)
- [자동 크기 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. 기본 값은 1이고, 기본 최대값은 256입니다.
- **ParallelismPerKPU** — 이 속성을 사용하여 애플리케이션의 Kinesis 처리 단위(KPU)에 따라 예약할 수 있는 병렬 작업 수를 설정합니다. 기본 값은 1이고 최대 값은 8입니다. 블로킹 작업(예: I/O)이 있는 애플리케이션의 경우 ParallelismPerKPU 값이 높을수록 KPU 리소스가 완전히 활용됩니다.

Note

Parallelism의 한도는 KPU 한도(기본 값 64)의 ParallelismPerKPU배와 같습니다. 한도 증가를 요청하여 KPU 한도를 늘릴 수 있습니다. 한도 증가를 요청하는 방법에 대한 지침은 [Service Quotas](#)의 “한도 증가를 요청하려면”을 참조하세요.

특정 연산자의 작업 병렬성 설정에 대한 자세한 내용은 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 업데이트](#)에서 설명하는 대로 AutoScalingEnabled 값을 false로 설정합니다.

애플리케이션에 대한 KPU 기본 한도는 64입니다. 이 한도 증가를 요청하는 방법에 대한 지침은 [Service Quotas](#)의 “한도 증가를 요청하려면”을 참조하십시오.

Note

오케스트레이션을 위해 추가 KPU가 부과됩니다. 자세한 내용을 알아보려면 [Managed Service for Apache Flink 요금](#)을 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 [아파치 플링크용 매니지드 서비스 API 예제 코드](#) 섹션을 참조하십시오.

다음 예 [CreateApplication](#) 작업 요청은 애플리케이션을 만들 때 병렬성을 설정합니다.

```
{
  "ApplicationName": "string",
  "RuntimeEnvironment": "FLINK-1_18",
  "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 크기 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 지표 최대값이 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. [DescribeApplication](#) 또는 [ListApplications](#) 작업을 사용하여 현재 애플리케이션 상태를 확인할 수 있습니다. 서비스가 애플리케이션을 확장하는 동안 사용할 수 있는 유효한 API 작업은 `Force` 파라미터를 [StopApplication](#)로 설정한 상태에서만 사용할 수 `true` 있습니다.

`AutoScalingEnabled` 속성([FlinkApplicationConfiguration](#)에 속함)을 사용하여 `Auto Scaling` 동작을 활성화하거나 비활성화할 수 있습니다. Apache Flink용 관리형 서비스가 애플리케이션

이션 및 설정에 따라 제공하는 KPU에 대한 요금이 사용자 AWS 계정에 부과됩니다. `parallelismPerKPU` 활동이 급증하면 Managed Service for Apache Flink 비용이 증가합니다.

요금에 대한 자세한 내용을 알아보려면 [Amazon Managed Service for Apache Flink 요금](#)을 참조하세요.

애플리케이션 규모 조정에 대해 다음을 유의하세요.

- 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 내용을 알아보려면 [할당량](#) 섹션을 참조하세요.
- 자동 크기 조정이 애플리케이션 병렬성을 업데이트하면 애플리케이션 다운타임이 발생합니다. 이러한 다운타임을 방지하려면 다음을 수행하십시오.
 - 자동 조정 사용 중지
 - 작업을 사용하여 애플리케이션의 AND를 `parallelism` 구성하십시오 `parallelismPerKPU`. [UpdateApplication](#) 애플리케이션의 병렬성 설정 설정에 대한 자세한 내용을 알아보려면 다음 [the section called “애플리케이션의 병렬성 업데이트”](#) 섹션을 참조하세요.
 - 애플리케이션의 리소스 사용량을 주기적으로 모니터링하여 애플리케이션의 워크로드에 대한 병렬성 설정이 올바른지 확인하십시오. 할당 리소스 사용량 모니터링에 대한 자세한 내용을 알아보려면 [the section called “Apache Flink용 매니지드 서비스의 메트릭과 디멘션”](#) 섹션을 참조하십시오.

maxParallelism 고려 사항

- 자동 크기 조정 로직은 Flink 작업을 병렬성으로 조정하여 작업 및 연산자 `maxParallelism`에 간섭을 일으키는 것을 방지합니다. 예를 들어 소스만 있고 싱크만 있는 간단한 작업의 경우 소스가 `maxParallelism` 16이고 싱크가 8인 경우 작업이 8을 초과하도록 자동 크기 조정되지 않습니다.
- 어떤 작업에 대해 `maxParallelism`이 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 내용을 알아보려면 [Apache Flink의 고급 모니터링 및 자동 크기 조정](#)을 참조하십시오.

예를 들어, 을 참조하십시오 [kda-flink-app-autoscaling](#).

태깅 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 사용](#)을 참조하세요.
- 태그를 기반으로 애플리케이션 리소스에 대한 액세스 통제. 자세한 설명은 AWS Identity and Access Management 사용자 가이드의 [태그를 사용한 액세스 통제](#)를 참조하십시오.
- 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.

- [애플리케이션 생성 시 태그 추가](#)
- [기존 애플리케이션에 태그 추가 또는 u 태그 추가](#)
- [애플리케이션 태그 목록](#)
- [애플리케이션에서 태그 제거](#)

애플리케이션 생성 시 태그 추가

[CreateApplication](#)액션의 tags 매개 변수를 사용하여 응용 프로그램을 만들 때 태그를 추가합니다.

다음 예 요청은 CreateApplication 요청에 대한 Tags 노드를 보여줍니다.

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

```

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

```

기존 애플리케이션에 태그 추가 또는 u 태그 추가

[TagResource](#) 액션을 사용하여 애플리케이션에 태그를 추가합니다. [UpdateApplication](#) 액션을 사용하여 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"
    }
  ]
}

```

애플리케이션 태그 목록

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" ]
}
```

아파치 플링크용 매니지드 서비스와 CloudFormation 함께 사용

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

시작하기 전에

이 연습을 시작하기 전에 `at`를 사용하여 Flink 애플리케이션을 생성하는 단계를 따르십시오. AWS CloudFormation [AWS::KinesisAnalytics::Application](#)

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 cfntools
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']))
            cfntools.send(event, context, cfntools.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))
            cfntools.send(event, context, cfntools.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))
cfresponse.send(event, context, cfresponse.SUCCESS, {})
except Exception as err:
    logger.error(err)
    cfresponse.send(event, context, cfresponse.FAILED, {"Data": str(err)})

```

위 코드에서 Lambda는 AWS CloudFormation 들어오는 이벤트를 처리하고, Create 그 외의 모든 이벤트를 Update 필터링하고, 애플리케이션 상태를 가져와서 상태가 다음과 같으면 시작합니다. READY 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 사용자 지정 정책을 사용하는 것으로 범위를 좁히는 것이 좋습니다.

```

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. [사용자 지정 리소스를](#) 사용하여 이 작업을 수행할 수 있습니다.

```
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 예제는 [템플릿 매개변수를](#) 통해 추가 RunConfiguration 조정을 수행한 이전 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:
- kinesisanalytics.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_18'
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 [가이드](#)에 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/cloudformation>으로 이동하여 진행 상황을 확인할 수 있습니다. 생성한 후에는 Flink 애플리케이션이 현재 Starting 상태인 것을 확인할 수 있습니다. Running을 시작하려면 몇 분이 걸릴 수 있습니다.

자세한 정보는 [을\(를\) 참조하세요](#).

- [AWS CloudFormation \(3부 중 1부\) 를 사용하여 AWS 서비스 속성을 검색하는 네 가지 방법](#).
- [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 내용을 알아보려면 Apache Flink 웹 사이트의 [Apache Flink 아키텍처](#)를 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.

Apache Flink용 관리형 서비스 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.

작업을 사용하여 대시보드 URL을 생성합니다. [CreateApplicationPresignedUrl](#) 작업에 대해 다음 파라미터를 지정할 수 있습니다.

- 애플리케이션 명칭
- URL이 유효한 시간(단위: 초)
- FLINK_DASHBOARD_URL을 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.

Note

더 이상 사용되지 않는 Apache Flink 버전을 사용하는 경우 Apache Flink용 관리 서비스의 [아파치 플링크의 인플레이스 버전 업그레이드](#) 기능을 사용하여 지원되는 최신 Flink 버전으로 애플리케이션을 업그레이드하는 것이 좋습니다.

아파치 플링크 버전	상태 - 아파치 플링크용 아마존 매니지드 서비스	상태 - 아파치 플링크 커뮤니티	링크
1.19.1	지원	지원	아파치 플링크용 아마존 매니지드 서비스 1.19
1.18.1	지원	지원	아파치 플링크용 아마존 매니지드 서비스 1.18
1.15.2	지원	지원되지 않음	아파치 플링크용 아마존 매니지드 서비스 1.15
1.13.1	지원	지원되지 않음	시작하기: 플링크 1.13.2
1.11.1	비하하는 행위	지원되지 않음	Apache Flink용 관리 서비스의 이전 버전 정보(2024년 11월 5일 지원 중단)
1.8.2	비하	지원되지 않음	Apache Flink용 관리 서비스의 이전 버전 정

아파치 플링크 버전	상태 - 아파치 플링크용 아마존 매니지드 서비스	상태 - 아파치 플링크 커뮤니티	링크
			보 (2024년 11월 5일 지원 중단)
1.6.2	비하	지원되지 않음	Apache Flink용 관리 서비스의 이전 버전 정보 (2024년 11월 5일 지원 중단)

주제

- [아파치 플링크용 아마존 매니지드 서비스 1.19](#)
- [아파치 플링크용 아마존 매니지드 서비스 1.18](#)
- [아파치 플링크용 아마존 매니지드 서비스 1.15](#)
- [Apache Flink용 관리 서비스의 이전 버전 정보](#)

아파치 플링크용 아마존 매니지드 서비스 1.19

아파치 플링크용 관리형 서비스는 이제 아파치 플링크 버전 1.19.1을 지원합니다. 이 섹션에서는 Apache Flink 1.19.1의 Apache Flink용 관리 서비스 지원에 도입된 주요 새 기능 및 변경 사항을 소개합니다.

Note

지원되는 이전 버전의 Apache Flink를 사용 중이고 기존 애플리케이션을 Apache Flink 1.19.1로 업그레이드하려는 경우, 현재 Apache Flink 버전 업그레이드를 사용하여 업그레이드할 수 있습니다. 자세한 정보는 [아파치 플링크의 인플레이스 버전 업그레이드](#)을 참조하세요. 인플레이스 버전 업그레이드를 사용하면 스냅샷, 로그, 메트릭, 태그, Flink 구성 등을 포함하여 Apache Flink 버전 전반에서 단일 ARN에 대해 애플리케이션을 추적할 수 있습니다.


지원되는 기능

Apache Flink 1.19.1에서는 명명된 매개변수, 사용자 지정 소스 병렬화, 다양한 Flink 연산자에 대한 다양한 상태 TTL과 같은 SQL API의 개선 사항을 도입했습니다.

지원되는 기능 및 관련 문서

지원되는 기능	설명	아파치 플링크 설명서 참조
SQL API: SQL 힌트를 사용하여 다양한 상태 TTL 구성 지원	이제 사용자는 스트림 일반 조인 및 그룹 애그리게이트에서 상태 TTL을 구성할 수 있습니다.	FLIP-373: SQL 힌트를 사용하여 다양한 상태 TTL 구성
SQL API: 함수 및 호출 프로시저에 대한 명명된 매개 변수 지원	이제 사용자는 매개 변수 순서에 의존하지 않고 함수에서 명명된 매개 변수를 사용할 수 있습니다.	FLIP-378: 함수 및 호출 프로시저에 대한 명명된 매개 변수 지원
SQL API: SQL 소스에 대한 병렬 처리 설정	이제 사용자는 SQL 소스의 병렬성을 지정할 수 있습니다.	FLIP-367: 테이블/SQL 소스에 대한 병렬 처리 설정 지원
SQL API: Support 세션 윈도우 TVF 지원	이제 사용자는 세션 창 테이블 반환 함수를 사용할 수 있습니다.	FLINK-24024: Support 세션 윈도우 TVF
SQL API: 윈도우 TVF 어그리게이션은 변경 로그 입력을 지원합니다.	이제 사용자는 변경 로그 입력에 대해 창 집계를 수행할 수 있습니다.	FLINK-20281: 창 집계는 변경 로그 스트림 입력을 지원합니다.
Support Python 3.11 지원	플링크는 이제 파이썬 3.11을 지원합니다. 파이썬 3.11은 파이썬 3.10에 비해 10~60% 더 빠릅니다. 자세한 내용은 내용은 Python 3.11의 새로운 기능을 참조하십시오.	FLINK-33030: 파이썬 3.11 지원 추가

지원되는 기능	설명	아파치 플링크 설명서 참조
싱크에 대한 메트릭을 TwoPhaseCommitting 제공하세요	사용자는 2단계 커밋 싱크의 커미터 상태에 대한 통계를 볼 수 있습니다.	FLIP-371: 커미터 생성을 위한 초기화 컨텍스트를 제공합니다. TwoPhaseCommittingSink
작업 재시작 및 체크포인트를 위한 추적 리포터	이제 사용자는 체크포인트 기간 및 복구 추세에 대한 추적을 모니터링할 수 있습니다. Apache Flink용 Amazon Managed Service에서는 기본적으로 SLF4j 추적 보고기를 활성화하여 사용자가 애플리케이션 로그를 통해 체크포인트 및 작업 추적을 모니터링할 수 있습니다. CloudWatch	FLIP-384: 체크포인트 TraceReporter 및 복구 추적 생성에 도입 및 사용

 Note

[지원](#) 사례를 제출하여 다음 기능을 선택할 수 있습니다.

옵트인 기능 및 관련 문서

옵트인 기능	설명	아파치 플링크 문서 참조
소스에서 백로그를 처리 중일 때 더 큰 체크포인트 간격 사용 지원	사용자가 특정 작업 요구 사항에 맞게 구성을 조정해야 하므로 이는 옵트인 기능입니다.	FLIP-309: 소스에서 백로그를 처리 중일 때 더 큰 체크포인트 간격 사용 지원
System.out과 System.err를 자바 로그로 리디렉션합니다.	이 기능은 옵트인 기능입니다. Apache Flink용 Amazon Managed Service의 기본 동작은 System.out 및 System.err의 출력을 무시하는 것입니다. 프로덕션 환경에서는 네이티브	FLIP-390: Support 시스템 종료 및 오류가 LOG로 리디렉션되거나 삭제됨

옵트인 기능	설명	아파치 플링크 문서 참조
	자바 로거를 사용하는 것이 가장 좋기 때문입니다.	

[아파치 플링크 1.19.1 릴리스 설명서는 아파치 플링크 설명서 v1.19.1을 참조하십시오.](#)

아파치 플링크용 아마존 매니지드 서비스 1.19.1의 변경

로깅 트레이스 리포터는 기본적으로 활성화되어 있습니다.

Apache Flink 1.19.1에는 체크포인트 및 복구 트레이스가 도입되어 사용자가 체크포인트 및 작업 복구 문제를 더 잘 디버깅할 수 있게 되었습니다. Amazon Managed Service for Apache Flink에서는 이러한 추적이 CloudWatch 로그 스트림에 기록되므로 사용자는 작업 초기화에 소요된 시간을 세분화하고 체크포인트의 과거 크기를 기록할 수 있습니다.

기본 재시작 전략은 이제 지수 지연입니다.

Apache Flink 1.19.1에서는 지수 지연 재시작 전략이 크게 개선되었습니다. Flink 1.19.1 이후 버전의 Apache Flink용 Amazon 매니지드 서비스에서 Flink 작업은 기본적으로 지수 지연 재시작 전략을 사용합니다. 즉, 사용자 작업은 일시적 오류로부터 더 빨리 복구되지만 작업이 계속 재시작되더라도 외부 시스템에 과부하가 걸리지는 않습니다.

Flink 작업 코드를 사용한 특정 Flink 작업 구성은 사용할 수 없습니다.

Apache Flink 1.18 및 이전 버전에서는 Apache Flink용 Amazon Managed Service에서 모든 프로그래밍 방식의 구성 변경을 허용했으며 일부 구성은 자동으로 재정의되었습니다. Apache Flink 1.19 이상부터 Flink 작업 코드를 사용하여 Flink 작업 구성 변경을 비활성화했습니다. 사용자가 이 구성을 지정하면 작업 목록이 표시됩니다. ProgramInvocationException 자세한 정보는 [FlinkRuntimeException: “허용되지 않은 구성 변경이 감지되었습니다.”](#)을 참조하세요.

구성 요소

구성 요소	버전
Java	11 (권장)
Python	3.11

구성 요소	버전
Kinesis Data Analytics 플링크 런타임 () aws-kinesisanalytics-runtime	1.2.0
커넥터	사용 가능한 커넥터에 대한 자세한 내용은 Apache Flink 커넥터 를 참조하십시오.
Apache Beam (빔 애플리케이션만 해당)	플링크 1.19와 호환되는 아파치 플링크 러너는 없습니다. 자세한 내용은 Flink 버전 호환성을 참조하십시오.

알려진 문제

아파치 빔

현재 아파치 빔에는 플링크 1.19와 호환되는 아파치 플링크 러너가 없습니다. [자세한 내용은 Flink 버전 호환성을 참조하십시오.](#)

아파치 플링크 스튜디오용 아마존 매니지드 서비스

스튜디오는 Apache Zeppelin 노트북을 사용하여 Apache Flink 스트림 처리 애플리케이션을 개발, 디버깅 및 실행하기 위한 단일 인터페이스 개발 환경을 제공합니다. Flink 1.19를 지원하려면 제플린의 플링크 인터프리터를 업그레이드해야 합니다. 이 작업은 Zeppelin 커뮤니티와 함께 진행될 예정이며, 작업이 완료되면 이 노트를 업데이트할 예정입니다. 아파치 플링크 스튜디오용 아마존 매니지드 서비스와 함께 Flink 1.15를 계속 사용할 수 있습니다. 자세한 내용은 스튜디오 노트북 [만들기를](#) 참조하십시오.

아파치 플링크용 아마존 매니지드 서비스 1.18

아파치 플링크용 관리형 서비스는 이제 아파치 플링크 버전 1.18.1을 지원합니다. Apache Flink 1.18.1의 Apache Flink용 관리 서비스 지원에 도입된 주요 새 기능 및 변경 사항에 대해 알아보십시오.

Note

지원되는 이전 버전의 Apache Flink를 사용 중이고 기존 애플리케이션을 Apache Flink 1.18.1로 업그레이드하려는 경우, 현재 Apache Flink 버전 업그레이드를 사용하여 업그레이드할 수 있습니다. 인플레이스 버전 업그레이드를 사용하면 스냅샷, 로그, 메트릭, 태그, Flink 구성 등을 포함하여 Apache Flink 버전 전반에서 단일 ARN에 대해 애플리케이션을 추적할 수 있습니다.

다. 이 기능은 모든 상태에서 사용할 수 있습니다. RUNNING READY 자세한 정보는 [아파치 플링크의 인플레이스 버전 업그레이드](#)을 참조하세요.

지원되는 기능	설명	아파치 플링크 설명서 참조
오픈서치 커넥터	이 커넥터에는 at-least-once 보증을 제공하는 싱크가 포함되어 있습니다.	github: 오픈서치 커넥터
아마존 DynamoDB 커넥터	이 커넥터에는 보증을 제공하는 at-least-once 싱크가 포함되어 있습니다.	아마존 다이내모DB 싱크
몽고DB 커넥터	이 커넥터에는 at-least-once 보증을 제공하는 소스 및 싱크가 포함되어 있습니다.	몽고DB 커넥터
플링크 플래너를 사용하여 하이브를 분리하십시오.	별도의 JAR 교체 없이 Hive 사투리를 직접 사용할 수 있습니다.	FLINK-26603: 플링크 플래너와 하이브를 분리하세요
RockSDB에서 WAL을 기본적으로 비활성화합니다. WriteBatchWrapper	이렇게 하면 복구 시간이 더 빨라집니다.	FLINK-32326: 기본적으로 WriteBatchWrapper RockSDB에서 WAL을 비활성화합니다.
워터마크 정렬을 활성화할 때 워터마크 집계 성능을 개선하세요.	워터마크 정렬을 활성화할 때 워터마크 집계 성능을 개선하고 관련 벤치마크를 추가합니다.	FLINK-32524: 워터마크 집계 성능
워터마크 정렬을 프로덕션 환경에서 사용할 수 있도록 준비하세요	대규모 작업 과부하 위험을 제거합니다. JobManager	FLINK-32548: 워터마크 정렬 준비 완료
비동기 RateLimitingStrategy 싱크용으로 구성 가능	RateLimitingStrategy 규모 조정 대상, 규모 조정 시기, 규모	FLIP-242: 비동기 싱크를 RateLimitingStrategy 위한 구성 가능 소개

지원되는 기능	설명	아파치 플링크 설명서 참조
	조정 등에 대한 결정을 구성할 수 있습니다.	
테이블 및 열 통계를 일괄 가져 오기	쿼리 성능이 개선되었습니다.	FLIP-247: 지정된 파티션에 대한 테이블 및 열 통계를 대량으로 가져옵니다.

[아파치 플링크 1.18.1 릴리스 설명서는 아파치 플링크 1.18.1 릴리스 발표를 참조하십시오.](#)

아파치 플링크를 사용한 아파치 플링크용 Amazon 매니지드 서비스의 변경 사항 1.18

Akka는 Pekko로 대체되었습니다.

아파치 플링크 1.18에서 아파치 플링크가 아카를 페코로 대체했습니다. 이 변경 사항은 아파치 플링크 1.18.1 이상의 아파치 플링크용 매니지드 서비스에서 완벽하게 지원됩니다. 이번 변경으로 인해 애플리케이션을 수정할 필요가 없습니다. 자세한 내용은 [FLINK-32468: Akka를 Pekko로 바꾸기를 참조하십시오.](#)

스레드 모드에서 Support PyFlink 런타임 실행 지원

이번 Apache Flink 변경은 Pyflink 런타임 프레임워크를 위한 새로운 실행 모드인 프로세스 모드를 도입합니다. Process Mode는 이제 별도의 프로세스 대신 동일한 스레드에서 Python 사용자 정의 함수를 실행할 수 있습니다.

구성 요소

구성 요소	버전
Java	11 (권장)
Scala	버전 1.15부터 Flink는 Scala에 구애받지 않습니다. 참고로 MSF 플링크 1.18은 스칼라 3.3 (LTS)에 대해 검증되었습니다.
아파치 플링크 플링크 런타임을 위한 매니지드 서비스 () aws-kinesisanalytics-runtime	1.2.0

구성 요소	버전
AWS Kinesis 커넥터 (flink-connector-kinesis) [출처]	4.2.0-1.18
AWS Kinesis 커넥터 (flink-connector-kinesis) [싱크]	4.2.0-1.18
Apache Beam (빔 애플리케이션만 해당)	2.75.0 이전 버전 및 최대 버전입니다. 자세한 내용은 Flink 버전 호환성을 참조하십시오.

버그 수정

아파치 플링크 1.18.1의 상태 압축

Apache Flink는 모든 체크포인트와 세이브포인트에 대해 선택적 압축(기본값: 끄기)을 제공합니다. Apache Flink는 Flink 1.18.1에서 스냅샷 압축을 활성화했을 때 운영자 상태를 제대로 복원할 수 없는 버그를 식별했습니다. 이로 인해 데이터가 손실되거나 체크포인트에서 복원할 수 없게 될 수 있습니다. 자세한 내용은 [FLINK-34063: 스냅샷 압축이 활성화된 경우 소스 운영자를 재조정하면 일부 분할이 손실되는 현상을](#) 참조하십시오.

이 문제를 해결하기 위해 Apache Flink용 Amazon 관리형 서비스는 Apache Flink의 향후 버전에 포함될 수정 사항을 백포트했습니다. 자세한 내용은 [github: 항상 압축 버퍼 플러시를](#) 참조하십시오.

알려진 문제

아파치 플링크 스튜디오용 아마존 매니지드 서비스

스튜디오는 Apache Zeppelin 노트북을 사용하여 Apache Flink 스트림 처리 애플리케이션을 개발, 디버깅 및 실행하기 위한 단일 인터페이스 개발 환경을 제공합니다. Flink 1.18을 지원하려면 제플린의 플링크 인터프리터를 업그레이드해야 합니다. 이 작업은 Zeppelin 커뮤니티와 함께 진행될 예정이며, 작업이 완료되면 이 노트를 업데이트할 예정입니다. 아파치 플링크 스튜디오용 아마존 매니지드 서비스와 함께 Flink 1.15를 계속 사용할 수 있습니다. 자세한 내용은 스튜디오 노트북 [만들기를](#) 참조하십시오.

아파치 플링크용 아마존 매니지드 서비스 1.15

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

기능	설명	Apache Flip 참조
비동기식 싱크	개발자가 이전 AWS 노력의 절반도 안 되는 노력으로 사용자 지정 AWS 커넥터를 구축할 수 있도록 하는 비동기 대상 구축을 위한 기본 프레임워크입니다. 자세한 내용은 일반 비동기식 베이스 싱크 를 참조하세요.	FLIP-171: 비동기 싱크.
Kinesis Data Firehose Sink	AWS 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 원하는 Scala 표준 라이브러리를 번들로 제공해야 합니다.	FLIP-28: 스칼라 없는 플링크 테이블을 만드는 장기 목표.
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 정의 했습니다. Managed Service for Apache Flink는 싱크 및 소스 지표를 계속 지원할 예정이며, 1.15에서는 numRestarts 을	FLIP-33: 커넥터 지표의 표준화 및 FLIP-179: 표준화된 운영자 지표 공개.

기능	설명	Apache Flip 참조
완료된 작업 체크포인트	(를) 가용성 지표fullRestarts 와 병행하여 도입될 예정입니다. 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 사용](#)에서 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 Sink](#)를 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 더 이상 사용되지 [FlinkKafkaConsumer](#) 않으며 Kafka 싱크용 [FlinkKafkaProducer](#)이러한 API는 Flink 1.15용 Kafka를 커밋할 수 없습니다. [KafkaSource](#) 및 [KafkaSink](#)를 사용하고 있는지 확인하세요.

구성 요소

구성 요소	버전
Java	11 (권장)
Scala	2.12
아파치 플링크 플링크 런타임을 위한 매니지드 서비스 () aws-kinesisanalytics-runtime	1.2.0
AWS Kinesis 커넥터 () flink-connector-kinesis	1.15.4
Apache Beam (빔 애플리케이션만 해당)	2.33.0(Jackson 버전 2.12.2 포함)

Apache Flink용 관리 서비스의 이전 버전 정보

Note

아파치 플링크 커뮤니티는 아파치 플링크 버전 1.6, 1.8, 1.11을 3년 넘게 지원하지 않았습니 다. 2024년 11월 5일에 Apache Flink용 아마존 매니지드 서비스에서 이러한 버전을 더 이상 사용하지 않을 계획입니다. 이 날짜부터 이러한 Flink 버전에 대한 새 애플리케이션을 만들 수 없습니다. 지금은 기존 애플리케이션을 계속 실행할 수 있습니다. Apache Flink용 Amazon Managed Service의 인플레이스 버전 업그레이드 기능을 사용하여 애플리케이션을 안전하게 업그레이드할 수 있습니다. 자세한 내용은 [아파치 플링크의 인플레이스 버전 업그레이드](#)

아파치 플링크 버전 1.15.2 및 1.13.2는 아파치 플링크용 관리형 서비스에서 지원되지만 아파치 플링크 커뮤니티에서는 더 이상 지원하지 않습니다.

이 주제는 다음 섹션을 포함하고 있습니다.

- [아파치 플링크 Kinesis Streams 커넥터를 이전 아파치 플링크 버전과 함께 사용하기](#)
- [아파치 플링크 1.8.2를 이용한 애플리케이션 구축](#)
- [아파치 플링크를 이용한 애플리케이션 구축 1.6.2](#)
- [애플리케이션 업그레이드](#)
- [아파치 플링크 1.6.2 및 1.8.2에서 사용 가능한 커넥터](#)
- [시작하기: 플링크 1.13.2](#)
- [시작하기: 플링크 1.11.1 - 지원 중단](#)
- [시작하기: 플링크 1.8.2 - 지원 중단](#)
- [시작하기: 플링크 1.6.2 - 지원 중단](#)
- [Apache Flink용 관리형 서비스의 이전 버전 \(레거시\) 예제](#)

아파치 플링크 Kinesis Streams 커넥터를 이전 아파치 플링크 버전과 함께 사용하기

Apache Flink Kinesis Streams 커넥터는 버전 1.11 이전의 Apache Flink에 포함되지 않았습니다. 애플리케이션에서 Apache Flink Kinesis 커넥터를 이전 버전의 Apache Flink와 함께 사용하려면 애플리케이션에서 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 Maven](#)이 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.

아파치 플링크 1.8.2를 이용한 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
플링크 런타임을 위한 아파치 플링크용 매니저 서비스 () <code>aws-kinesisanalytics-runtime</code>	1.0.1
아파치 플링크 플링크 커넥터 관리 서비스 () <code>aws-kinesisanalytics-flink</code>	1.0.1

구성 요소	버전
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 애플리케이션](#)을 참조하세요.

Managed Service for Apache Flink 애플리케이션의 애플리케이션 코드를 구축하고 사용하는 방법에 대한 자세한 내용을 알아보려면 [애플리케이션 생성](#) 섹션을 참조하세요.

아파치 플링크를 이용한 애플리케이션 구축 1.6.2

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)
AWS 자바 SDK	1.11.379
Apache Flink	1.6.2
플링크 런타임을 위한 아파치 플링크용 매니저 서비스 () aws-kinesisanalytics-runtime	1.0.1
아파치 플링크 플링크 커넥터 관리 서비스 () 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 애플리케이션](#)을 참조하세요.

Managed Service for Apache Flink 애플리케이션의 애플리케이션 코드를 구축하고 사용하는 방법에 대한 자세한 내용을 알아보려면 [애플리케이션 생성](#) 섹션을 참조하세요.

애플리케이션 업그레이드

Apache Flink용 Amazon 관리형 서비스 애플리케이션의 Apache Flink 버전을 업그레이드하려면, SDK 또는 CLI를 사용하여 인플레이스 Apache Flink 버전 업그레이드 기능을 사용하십시오. AWS CLI, AWS CloudFormation, AWS Management Console. 자세한 정보는 [아파치 플링크의 인플레이스 버전 업그레이드](#)을 참조하세요.

이 기능은 Apache Flink용 Amazon Managed Service for Apache Flink와 함께 사용하는 모든 기존 애플리케이션에서 사용할 수 있습니다.

아파치 플링크 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 내용을 알아보려면 [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 내용을 알아보려면 [Apache Flink 설명서\(1.8.2\)의 커넥터\(1.8.2\)](#)를 참조하세요.

시작하기: 플링크 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

- [Apache Flink 애플리케이션용 매니지드 서비스의 구성 요소](#)
- [연습을 완료하기 위한 사전 요구 사항](#)
- [1단계: AWS 계정 설정 및 관리자 사용자 생성](#)
- [다음 단계](#)
- [2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)
- [3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)
- [4단계: AWS 리소스 정리](#)
- [5단계: 다음 절차](#)

Apache Flink 애플리케이션용 매니지드 서비스의 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 내용을 알아보려면 [소스](#) 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 내용을 알아보려면 [DataStream API 연산자](#) 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 정보는 [싱크](#)를 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.

연습을 완료하기 위한 사전 요구 사항

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 생성](#) 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성

가입하세요. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

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

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

에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업](#)을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 <https://aws.amazon.com/>으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

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

보안을 유지하세요 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 [AWS Management Console](#) 소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

루트 사용자를 사용하여 로그인하는 데 도움이 필요하다면 AWS 로그인 User Guide의 [루트 사용자 로 로그인](#)을 참조하십시오.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

지침은 IAM [사용 설명서의 AWS 계정 루트 사용자 \(콘솔\)에 대한 가상 MFA 디바이스 활성화를 참조](#)하십시오.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

지침은 AWS IAM Identity Center 사용 설명서의 [AWS IAM Identity Center 설정](#)을 참조하십시오.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

를 ID 소스로 사용하는 방법에 대한 자습서는 사용 [설명서의 기본값으로 IAM Identity Center 디렉터리 사용자 액세스 구성](#)을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 액세스 권한이 있는 사용자로 로그인

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

IAM Identity Center 사용자를 사용하여 [로그인하는 데 도움이 필요하다면 사용 설명서의 AWS 액세스 포털에 로그인](#)을 참조하십시오. AWS 로그인

추가 사용자에게 액세스 권한 할당

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

지침은 AWS IAM Identity Center 사용 설명서의 [Create a permission set](#)를 참조하십시오.

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

지침은 AWS IAM Identity Center 사용 설명서의 [Add groups](#)를 참조하십시오.

프로그래밍 방식 액세스 권한 부여

사용자가 AWS 외부 사용자와 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다. AWS Management Console 프로그래밍 방식의 액세스 권한을 부여하는 방법은 액세스하는 사용자 유형에 따라 다릅니다. AWS

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명할 수 있습니다. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. <ul style="list-style-type: none"> • AWS CLI에 대한 내용은 사용 설명서의 AWS CLI 사용을 AWS IAM Identity Center위한 구성을 참조하십시오. AWS Command Line Interface • AWS SDK, 도구 및 AWS API의 경우 AWS SDK 및 도구 참조 안내서의 IAM ID 센터 인증을 참조하십시오.
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 방식 요청에 서명할 수 있습니다. AWS	IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용의 지침을 따르십시오.
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명하십시오. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. <ul style="list-style-type: none"> • 에 대한 내용은 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하십시오. AWS CLI AWS Command Line Interface

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		<ul style="list-style-type: none"> • AWS SDK 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용한 인증을 참조하십시오. AWS • AWS API의 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

다음 단계

[2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)

다음 단계

[2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)

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

이 단계에서는 Apache Flink용 관리 서비스와 함께 사용하도록 AWS CLI 를 다운로드하고 구성합니다.

Note

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

Note

이미 AWS CLI 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수 있습니다. 자세한 내용을 알아보려면 AWS Command Line Interface 사용자 가이드에서 [AWS Command Line Interface 설치](#)를 참조하세요. 의 버전을 확인하려면 다음 명령을 실행합니다. AWS CLI

```
aws --version
```

이 자습서의 연습에는 다음 AWS CLI 버전 이상이 필요합니다.

```
aws-cli/1.16.63
```

설정하려면 AWS CLI

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 [AWS Command Line Interface 사용 설명서](#)에서 다음 토픽을 참조하세요.
 - [AWS Command Line Interface설치](#)
 - [AWS CLI구성](#)
2. AWS CLI config파일에 관리자 사용자의 이름이 지정된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용됩니다. 프로파일 명명에 대한 자세한 설명은 [AWS Command Line Interface 사용자 가이드의 프로파일 명명](#)을 참조하십시오.

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

사용 가능한 AWS 지역 목록은 [의 지역 및 엔드포인트를](#) 참조하십시오. Amazon Web Services 일반 참조

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
```

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

다음 단계

[3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)

3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행

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

이 섹션은 다음 주제를 포함합니다:

- [두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.](#)
- [입력 스트림에 샘플 레코드를 작성합니다.](#)
- [Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [다음 단계](#)

두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(ExampleInputStream 및 ExampleOutputStream)을 생성하세요. 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 업데이트](#)를 참조하세요.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림 (ExampleInputStream) 을 생성하려면 다음 Amazon Kinesis 명령을 create-stream AWS CLI 사용하십시오.

```
$ 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
```

입력 스트림에 샘플 레코드를 작성합니다.

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기합니다.

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'))
```

- 이 자습서의 뒷부분에서 `stock.py` 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

- 다음 명령을 사용하여 원격 리포지토리를 복제합니다.

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

- `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 Kinesis 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 내용을 알아보려면 [런타임 속성](#) 섹션을 참조하세요.

애플리케이션 코드를 컴파일하십시오.

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 내용을 알아보려면 [연습을 완료하기 위한 사전 요구 사항을 충족하십시오](#). 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 도구를 사용합니다. pom.xml 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

```
mvn package -Dflink.version=1.13.2
```

- 귀하의 개발 환경 사용. 자세한 내용을 알아보려면 해당 개발 환경 설명서를 참조하십시오.

Note

제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. 를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형 (JAR 또는 ZIP) 을 지정합니다. AWS CLI

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
```

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

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. 를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 생성합니다. AWS CLI

주제

- [애플리케이션 \(콘솔\) 생성 및 실행](#)
- [애플리케이션 생성 및 실행 \(AWS CLI\)](#)

애플리케이션 (콘솔) 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 들 수 있는 옵션이 있습니다. 귀하의 애플리케이션은 이 역할 및 정책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 리전을 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. 다음을 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

애플리케이션을 실행합니다

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션을 중지하십시오.

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`를 사용하여 Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다. AWS CLI Apache Flink용 관리 서비스는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 이와 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 연결](#)을 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. 역할 생성을 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.

Apache Flink용 관리형 서비스 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 kinesisanalyticstv2 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.json`이라는 파일에 저장합니다.

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

- 위의 요청과 함께 [StartApplication](#) 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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

- 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```
{
  "ApplicationName": "test"
}
```

- 다음 요청과 함께 [StopApplication](#) 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticstv2 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 [the 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.json`이라는 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 kinesisanalyticstv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 [UpdateApplication](#) AWS CLI 작업을 사용합니다.

Note

파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 참조하세요.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름, 새 객체 버전을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 [the section called “두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.”](#) 섹션에서 선택한 접미사로 업데이트합니다.

```
{
  "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": "SAMPLEUehYngP87ex1nzYIGYgfhyvpvDU"
        }
      }
    }
  }
}
```

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:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)
- [다음 단계](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/flink>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 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 참조하세요.

- Amazon [Kinesis용 AWS 스트리밍 데이터 솔루션: Amazon Kinesis용 AWS 스트리밍 데이터 솔루션](#)은 스트리밍 데이터를 쉽게 캡처, 저장, 처리 및 전송하는 데 필요한 서비스를 자동으로 구성합니다 AWS . 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
- AWS Amazon [MSK용 스트리밍 데이터 솔루션: Amazon MSK용 AWS 스트리밍 데이터 솔루션](#)은 생산자, 스트리밍 스토리지, 소비자 및 목적지를 통해 데이터가 흐르는 AWS CloudFormation 템플릿을 제공합니다.
- [Apache Flink 및 Apache Kafka를 사용한 클릭스트림 랩](#): 스트리밍 스토리지는 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.
- [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: 플링크 1.11.1 - 지원 중단

Note

아파치 플링크 커뮤니티는 아파치 플링크 버전 1.6, 1.8, 1.11을 3년 넘게 지원하지 않고 있다. 2024년 11월 5일에 Apache Flink용 아마존 매니지드 서비스에서 이러한 버전을 더 이상 사용하지 않을 계획입니다. 이 날짜부터 이러한 Flink 버전에 대한 새 애플리케이션을 만들 수 없습니다. 지금은 기존 애플리케이션을 계속 실행할 수 있습니다. Apache Flink용 Amazon Managed Service의 인플레이스 버전 업그레이드 기능을 사용하여 애플리케이션을 안전하게 업그레이드할 수 있습니다. 자세한 내용은 [아파치 플링크의 인플레이스 버전 업그레이드](#)를 참조하십시오.

이 주제에는 Apache Flink 1.11.1을 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

- [Apache Flink 애플리케이션용 매니지드 서비스의 구성 요소](#)
- [연습을 완료하기 위한 사전 요구 사항](#)

- [1단계: AWS 계정 설정 및 관리자 사용자 생성](#)
- [2단계: AWS Command Line Interface \(AWS CLI\) 설정](#)
- [3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)
- [4단계: AWS 리소스 정리](#)
- [5단계: 다음 절차](#)

Apache Flink 애플리케이션용 매니지드 서비스의 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 내용을 알아보려면 [소스](#) 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 내용을 알아보려면 [DataStream API 연산자](#) 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 정보는 [싱크](#)을 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.

연습을 완료하기 위한 사전 요구 사항

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 생성](#) 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성

가입하세요. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

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

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

에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업](#)을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 <https://aws.amazon.com/>으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

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

보안을 유지하세요 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 [AWS Management Console](#)로 유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

루트 사용자를 사용하여 로그인하는 데 도움이 필요하다면 AWS 로그인 User Guide의 [루트 사용자 로 로그인](#)을 참조하십시오.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

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

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

지침은 AWS IAM Identity Center 사용 설명서의 [AWS IAM Identity Center 설정](#)을 참조하십시오.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

를 ID 소스로 사용하는 방법에 대한 자습서는 사용 [설명서의 기본값으로 IAM Identity Center 디렉터리 사용자 액세스 구성](#)을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 액세스 권한이 있는 사용자로 로그인

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

IAM Identity Center 사용자를 사용하여 [로그인하는 데 도움이 필요하다면 사용 설명서의 AWS 액세스 포털에 로그인](#)을 참조하십시오. AWS 로그인

추가 사용자에게 액세스 권한 할당

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

지침은 AWS IAM Identity Center 사용 설명서의 [Create a permission set](#)를 참조하십시오.

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

지침은 AWS IAM Identity Center 사용 설명서의 [Add groups](#)를 참조하십시오.

프로그래밍 방식 액세스 권한 부여

사용자가 AWS 외부 사용자와 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다. AWS Management Console 프로그래밍 방식의 액세스 권한을 부여하는 방법은 액세스하는 사용자 유형에 따라 다릅니다. AWS

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명할 수 있습니다. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. <ul style="list-style-type: none"> • AWS CLI에 대한 내용은 사용 설명서의 AWS CLI 사용을 AWS IAM Identity Center위한 구성을 참조하십시오. AWS Command Line Interface • AWS SDK, 도구 및 AWS API의 경우 AWS SDK 및 도구 참조 안내서의 IAM ID 센터 인증을 참조하십시오.
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 방식 요청에 서명할 수 있습니다. AWS	IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용 의 지침을 따르십시오.
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명하십시오. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. <ul style="list-style-type: none"> • 에 대한 내용은 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하십시오. AWS CLI AWS Command Line Interface

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
		<ul style="list-style-type: none"> • AWS SDK 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용한 인증을 참조하십시오. AWS • AWS API의 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

다음 단계

[2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)

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

이 단계에서는 Apache Flink용 관리 서비스와 함께 사용하도록 AWS CLI 를 다운로드하고 구성합니다.

Note

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

Note

이미 AWS CLI 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수 있습니다. 자세한 내용을 알아보려면 AWS Command Line Interface 사용자 가이드에서 [AWS Command Line Interface 설치](#)를 참조하세요. 의 버전을 확인하려면 다음 명령을 실행합니다. AWS CLI

```
aws --version
```

이 자습서의 연습에는 다음 AWS CLI 버전 이상이 필요합니다.

```
aws-cli/1.16.63
```

설정하려면 AWS CLI

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 [AWS Command Line Interface 사용 설명서](#)에서 다음 토픽을 참조하세요.
 - [AWS Command Line Interface설치](#)
 - [AWS CLI구성](#)
2. AWS CLI config파일에 관리자 사용자의 이름이 지정된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용됩니다. 프로파일 명명에 대한 자세한 설명은 [AWS Command Line Interface 사용자 가이드의 프로파일 명명](#)을 참조하십시오.

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

사용 가능한 AWS 지역 목록은 [의 지역 및 엔드포인트를](#) 참조하십시오. Amazon Web Services 일반 참조

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
```

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

다음 단계

[3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)

3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행

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

이 섹션은 다음 주제를 포함합니다:

- [두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.](#)
- [입력 스트림에 샘플 레코드를 작성합니다.](#)
- [Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [다음 단계](#)

두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(ExampleInputStream 및 ExampleOutputStream)을 생성하세요. 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 업데이트](#)를 참조하세요.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림 (ExampleInputStream) 을 생성하려면 다음 Amazon Kinesis 명령을 create-stream AWS CLI 사용하십시오.

```
$ 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
```

입력 스트림에 샘플 레코드를 작성합니다.

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"))
```


- 이 자습서의 뒷부분에서 `stock.py` 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

- 다음 명령을 사용하여 원격 리포지토리를 복제합니다.

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

- `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 Kinesis 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 내용을 알아보려면 [런타임 속성](#) 섹션을 참조하세요.

애플리케이션 코드를 컴파일하십시오.

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 내용을 알아보려면 [연습을 완료하기 위한 사전 요구 사항을 충족하십시오](#). 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 도구를 사용합니다. pom.xml 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

```
mvn package -Dflink.version=1.11.3
```

- 귀하의 개발 환경 사용. 자세한 내용을 알아보려면 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. 를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형 (JAR 또는 ZIP) 을 지정합니다. AWS CLI

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
```

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

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. 를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 생성합니다. AWS CLI

주제

- [애플리케이션 \(콘솔\) 생성 및 실행](#)
- [애플리케이션 생성 및 실행 \(AWS CLI\)](#)

애플리케이션 (콘솔) 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 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

그룹 ID	키	값
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

애플리케이션을 실행합니다

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션을 중지하십시오.

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` 를 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 실행합니다. Apache Flink용 관리 서비스는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Apache Flink 응용 프로그램용 관리 서비스 응용 프로그램을 만들고 상호 작용합니다. AWS CLI

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 관리 형 정책 생성 및 연결](#)을 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. 역할 생성을 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.

Apache Flink용 관리형 서비스 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_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"
          }
        }
      ]
    }
  }
}
```

- 위의 요청과 함께 [CreateApplication](#) 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticstv2 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.json`이라는 파일에 저장합니다.

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

2. 위의 요청과 함께 [StartApplication](#) 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```
{
  "ApplicationName": "test"
}
```

2. 다음 요청과 함께 [StopApplication](#) 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticstv2 stop-application --cli-input-json file://stop_request.json
```

애플리케이션이 중지됩니다.

CloudWatch로깅 옵션 추가

를 AWS CLI 사용하여 Amazon CloudWatch 로그 스트림을 애플리케이션에 추가할 수 있습니다. 애플리케이션에서 CloudWatch 로그를 사용하는 방법에 대한 자세한 내용은 [the 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.json`이라는 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"
          }
        }
      ]
    }
  }
}
```

```
}
}
```

- 이전 요청과 함께 [UpdateApplication](#) 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticstv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 [UpdateApplication](#) AWS CLI 작업을 사용합니다.

Note

파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 참조하세요.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름, 새 객체 버전을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 [the section called “두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.”](#) 섹션에서 선택한 접미사로 업데이트합니다.

```
{
  "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": "SAMPLEUehYngP87ex1nzYIGYgfhyvDU"  
  }  
} }  
}
```

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:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)
- [다음 단계](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/flink>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 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 참조하세요.

- Amazon [Kinesis용 AWS 스트리밍 데이터 솔루션: Amazon Kinesis용 AWS 스트리밍 데이터 솔루션](#)은 스트리밍 데이터를 쉽게 캡처, 저장, 처리 및 전송하는 데 필요한 서비스를 자동으로 구성합니다 AWS . 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

- AWS Amazon [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.
- [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: 플링크 1.8.2 - 지원 중단

Note

아파치 플링크 커뮤니티는 아파치 플링크 버전 1.6, 1.8, 1.11을 3년 넘게 지원하지 않고 있다. 2024년 11월 5일에 Apache Flink용 아마존 매니지드 서비스에서 이러한 버전을 더 이상 사용하지 않을 계획입니다. 이 날짜부터 이러한 Flink 버전에 대한 새 애플리케이션을 만들 수 없습니다. 지금은 기존 애플리케이션을 계속 실행할 수 있습니다. Apache Flink용 Amazon Managed Service의 인플레이스 버전 업그레이드 기능을 사용하여 애플리케이션을 안전하게 업그레이드할 수 있습니다. 자세한 내용은 [아파치 플링크의 인플레이스 버전 업그레이드](#)

이 주제에는 Apache Flink 1.8.2를 사용하는 [시작하기 \(API\) DataStream](#) 자습서 버전이 포함되어 있습니다.

주제

- [아파치 플링크 애플리케이션을 위한 매니지드 서비스의 구성 요소](#)
- [연습을 완료하기 위한 사전 요구 사항](#)
- [1단계: AWS 계정 설정 및 관리자 사용자 생성](#)
- [2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)
- [3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)
- [4단계: AWS 리소스 정리](#)

아파치 플링크 애플리케이션을 위한 매니지드 서비스의 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 내용을 알아보려면 [소스](#) 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 내용을 알아보려면 [DataStream API 연산자](#) 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 정보는 [싱크](#)을 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.

연습을 완료하기 위한 사전 요구 사항

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 내용을 알아보려면 [아파치 플링크 Kinesis Streams 커넥터를 이전 아파치 플링크 버전과 함께 사용하기](#) 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 생성](#) 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성

가입하세요. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

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

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

에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업](#)을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 <https://aws.amazon.com/>으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

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

보안을 유지하세요 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 [AWS Management Console](#)소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

루트 사용자를 사용하여 로그인하는 데 도움이 필요하다면 AWS 로그인 User Guide의 [루트 사용자 로 로그인](#)을 참조하십시오.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

지침은 IAM [사용 설명서의 AWS 계정 루트 사용자 \(콘솔\)에 대한 가상 MFA 디바이스 활성화](#)를 참조하십시오.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

지침은 AWS IAM Identity Center 사용 설명서의 [AWS IAM Identity Center](#)설정을 참조하세요.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

를 ID 소스로 사용하는 방법에 대한 자습서는 사용 [설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성](#)을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 액세스 권한이 있는 사용자로 로그인

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

IAM Identity Center 사용자를 사용하여 [로그인하는 데 도움이 필요하다면 사용 설명서의 AWS 액세스 포털에 로그인](#)을 참조하십시오.AWS 로그인

추가 사용자에게 액세스 권한 할당

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

지침은AWS IAM Identity Center 사용 설명서의 [Create a permission set](#)를 참조하세요.

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

지침은AWS IAM Identity Center 사용 설명서의 [Add groups](#)를 참조하세요.

프로그래밍 방식 액세스 권한 부여

사용자가 AWS 외부 사용자와 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다. AWS Management Console프로그래밍 방식의 액세스 권한을 부여하는 방법은 액세스하는 사용자 유형에 따라 다릅니다. AWS

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에서 서명할 수 있습니다. AWS	<p>사용하고자 하는 인터페이스에 대한 지침을 따릅니다.</p> <ul style="list-style-type: none"> • AWS CLI에 대한 내용은 사용 설명서의 AWS CLI사용을 AWS IAM Identity Center위한 구성을 참조하십시오. AWS Command Line Interface • AWS SDK, 도구 및 AWS API의 경우 AWS SDK 및 도구 참조 안내서의 IAM ID 센터 인증을 참조하십시오.
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 방식 요청에서 서명할 수 있습니다. AWS	IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용의 지침을 따르십시오.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명하십시오. AWS	사용하고자 하는 인터페이스에 대한 지침을 따릅니다. <ul style="list-style-type: none"> 에 대한 내용은 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하십시오. AWS CLI AWS Command Line Interface AWS SDK 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용한 인증을 참조하십시오. AWS AWS API의 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

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

이 단계에서는 Apache Flink용 관리 서비스와 함께 사용하도록 AWS CLI 를 다운로드하고 구성합니다.

Note

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

Note

이미 AWS CLI 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수 있습니다. 자세한 내용을 알아보려면 AWS Command Line Interface 사용자 가이드에서 [AWS Command Line Interface설치](#)를 참조하세요. 의 버전을 확인하려면 다음 명령을 실행합니다. AWS CLI

```
aws --version
```

이 자습서의 연습에는 다음 AWS CLI 버전 이상이 필요합니다.

```
aws-cli/1.16.63
```

설정하려면 AWS CLI

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 [AWS Command Line Interface 사용 설명서](#)에서 다음 토픽을 참조하세요.
 - [AWS Command Line Interface 설치](#)
 - [AWS CLI 구성](#)
2. AWS CLI config 파일에 관리자 사용자의 이름이 지정된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 명명된 프로필에 대한 자세한 내용을 알아보려면 [AWS Command Line Interface 사용자 가이드의 명명된 프로필](#)을 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 리전 목록은 [Amazon Web Services 일반 참조의 리전 및 엔드포인트](#)를 참조하세요.

Note

이 자습서의 예 코드 및 명령은 미국 서부(오레곤) 지역을 사용합니다. 다른 AWS 지역을 사용하려면 이 자습서의 코드 및 명령에 있는 지역을 사용하려는 지역으로 변경하십시오.

3. 명령 프롬프트에서 다음 help 명령을 입력하여 설정을 확인하십시오:

```
aws help
```

AWS 계정을 설정하고 나면 샘플 애플리케이션을 구성하고 end-to-end 설정을 테스트하는 다음 연습을 시도할 수 있습니다. [AWS CLI](#)

다음 단계

[3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)

3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행

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

이 섹션은 다음 주제를 포함합니다:

- [두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.](#)
- [입력 스트림에 샘플 레코드를 작성합니다.](#)
- [Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [다음 단계](#)

두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(ExampleInputStream 및 ExampleOutputStream)을 생성하세요. 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 업데이트](#)를 참조하세요.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림 (ExampleInputStream) 을 생성하려면 다음 Amazon Kinesis 명령을 create-stream AWS CLI 사용하십시오.

```
$ 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 kineses create-stream \
--stream-name ExampleOutputStream \
--shard-count 1 \
--region us-west-2 \
--profile adminuser
```

입력 스트림에 샘플 레코드를 작성합니다.

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰기합니다.

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"))

```

- 이 자습서의 뒷부분에서 stock.py 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

- 다음 명령을 사용하여 원격 리포지토리를 복제합니다.

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

- 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 Kinesis 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 내용을 알아보려면 [런타임 속성](#) 섹션을 참조하세요.

애플리케이션 코드를 컴파일하십시오.

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 내용을 알아보려면 [연습을 완료하기 위한 사전 요구 사항](#) 섹션을 참조하십시오.

Note

Kinesis 커넥터를 Apache Flink 1.11 이전 버전에서 사용하려면 Apache Maven을 다운로드, 빌드, 설치해야 합니다. 자세한 내용은 [the section called “아파치 플링크 Kinesis Streams 커넥터를 이전 아파치 플링크 버전과 함께 사용하기”](#) 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 도구를 사용합니다. `pom.xml` 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

```
mvn package -Dflink.version=1.8.2
```

- 귀하의 개발 환경 사용. 자세한 내용을 알아보려면 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. 를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형 (JAR 또는 ZIP) 을 지정합니다. AWS CLI
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
```

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

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. 를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 생성합니다. AWS CLI

주제

- [애플리케이션 \(콘솔\) 생성 및 실행](#)
- [애플리케이션 생성 및 실행 \(AWS CLI\)](#)

애플리케이션 (콘솔) 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 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

애플리케이션을 실행합니다

1. MyApplication페이지에서 [Run] 을 선택합니다. 작업을 확인합니다.
2. 애플리케이션이 실행 중이면 페이지를 새로 고칩니다. 콘솔에 애플리케이션 그래프가 표시됩니다.

애플리케이션을 중지합니다.

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 를 사용하여 Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다. AWS CLI Apache Flink용 관리 서비스는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 이와 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 연결](#)을 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. 역할 생성을 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.

Apache Flink용 관리형 서비스 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"
      }
    }
  ]
}
}
}
}

```

- 위의 요청과 함께 [CreateApplication](#) 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticstv2 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

- 다음 JSON 코드를 `start_request.json`이라는 파일에 저장합니다.

```

{
  "ApplicationName": "test",
  "RunConfiguration": {
    "ApplicationRestoreConfiguration": {
      "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
    }
  }
}

```

- 위의 요청과 함께 [StartApplication](#) 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```
{
  "ApplicationName": "test"
}
```

2. 다음 요청과 함께 [StopApplication](#) 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticstv2 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 [the 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.json`이라는 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"
      }
    }
  ]
}
}
}

```

- 이전 요청과 함께 [UpdateApplication](#) 작업을 실행하여 환경 속성을 업데이트하십시오.

```

aws kinesisanalyticstv2 update-application --cli-input-json file://
update_properties_request.json

```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 [UpdateApplication](#) AWS CLI 작업을 사용합니다.

Note

파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 참조하세요.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름, 새 객체 버전을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (<username>)를 [the section called “두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.”](#) 섹션에서 선택한 접미사로 업데이트합니다.

```
{
  "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": "SAMPLEUehYngP87ex1nzYIGYgfhyvDU"
        }
      }
    }
  }
}
```

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:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/kinesis>에서 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 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/flink>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

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

2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

시작하기: 플링크 1.6.2 - 지원 중단

Note

아파치 플링크 커뮤니티는 아파치 플링크 버전 1.6, 1.8, 1.11을 3년 넘게 지원하지 않고 있다. 2024년 11월 5일에 Apache Flink용 아마존 매니지드 서비스에서 이러한 버전을 더 이상 사용하지 않을 계획입니다. 이 날짜부터 이러한 Flink 버전에 대한 새 애플리케이션을 만들 수 없습니다. 지금은 기존 애플리케이션을 계속 실행할 수 있습니다. Apache Flink용 Amazon Managed Service의 인플레이스 버전 업그레이드 기능을 사용하여 애플리케이션을 안전하게 업그레이드할 수 있습니다. 자세한 내용은 [아파치 플링크의 인플레이스 버전 업그레이드](#)

이 주제에는 Apache Flink 1.6.2를 사용하는 [시작하기 \(API\) DataStream](#) 자습서 버전이 포함되어 있습니다.

주제

- [Apache Flink 애플리케이션용 관리형 서비스의 구성 요소](#)
- [연습을 완료하기 위한 사전 요구 사항](#)
- [1단계: AWS 계정 설정 및 관리자 사용자 생성](#)
- [2단계: \(\) 설정 AWS Command Line InterfaceAWS CLI](#)
- [3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)
- [4단계: AWS 리소스 정리](#)

Apache Flink 애플리케이션용 관리형 서비스의 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 내용을 알아보려면 [소스](#) 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 내용을 알아보려면 [DataStream API 연산자](#) 섹션을 참조하세요.
- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 정보는 [싱크](#)를 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.

연습을 완료하기 위한 사전 요구 사항

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 생성](#) 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성

가입하세요. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

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

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

에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업](#)을 수행하는 것입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 <https://aws.amazon.com/>으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

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

보안을 유지하세요 AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 [AWS Management Console](#)소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 [루트 사용자 로 로그인](#)을 참조하십시오.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

지침은 IAM [사용 설명서의 AWS 계정 루트 사용자 \(콘솔\)에 대한 가상 MFA 디바이스 활성화를 참조](#)하십시오.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

지침은 AWS IAM Identity Center 사용 설명서의 [AWS IAM Identity Center설정](#)을 참조하세요.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

를 ID 소스로 사용하는 방법에 대한 자습서는 [사용 설명서의 기본값으로 IAM Identity Center 디렉터리 사용자 액세스 구성](#)을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 액세스 권한이 있는 사용자로 로그인

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

IAM Identity Center 사용자를 사용하여 [로그인하는 데 도움이 필요하다면 사용 설명서의 AWS 액세스 포털 로그인](#)을 참조하십시오. AWS 로그인

추가 사용자에게 액세스 권한 할당

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

지침은 AWS IAM Identity Center 사용 설명서의 [Create a permission set](#)를 참조하세요.

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

지침은 AWS IAM Identity Center 사용 설명서의 [Add groups](#)를 참조하세요.

프로그래밍 방식 액세스 권한 부여

사용자가 AWS 외부 사용자와 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다. AWS Management Console 프로그래밍 방식의 액세스 권한을 부여하는 방법은 액세스하는 사용자 유형에 따라 다릅니다. AWS

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에서 명할 수 있습니다. AWS	<p>사용하고자 하는 인터페이스에 대한 지침을 따릅니다.</p> <ul style="list-style-type: none"> • AWS CLI에 대한 내용은 사용 설명서의 AWS CLI 사용을 AWS IAM Identity Center위

<p>프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?</p>	<p>To</p>	<p>액세스 권한을 부여하는 사용자</p>
		<p>한 구성을 참조하십시오. AWS Command Line Interface</p> <ul style="list-style-type: none"> • AWS SDK, 도구 및 AWS API의 경우 AWS SDK 및 도구 참조 안내서의 IAM ID 센터 인증을 참조하십시오.
<p>IAM</p>	<p>임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 방식 요청에 서명할 수 있습니다. AWS</p>	<p>IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용의 지침을 따르십시오.</p>
<p>IAM</p>	<p>(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명하십시오. AWS</p>	<p>사용하고자 하는 인터페이스에 대한 지침을 따릅니다.</p> <ul style="list-style-type: none"> • 에 대한 내용은 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하십시오. AWS CLI AWS Command Line Interface • AWS SDK 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용한 인증을 참조하십시오. AWS • AWS API의 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

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

이 단계에서는 Apache Flink용 관리형 서비스와 함께 사용하도록 AWS CLI 를 다운로드하고 구성합니다.

Note

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

Note

이미 AWS CLI 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수 있습니다. 자세한 내용을 알아보려면 AWS Command Line Interface 사용자 가이드에서 [AWS Command Line Interface설치](#)를 참조하세요. 이 버전을 확인하려면 다음 명령을 실행합니다. AWS CLI

```
aws --version
```

이 자습서의 연습에는 다음 AWS CLI 버전 이상이 필요합니다.

```
aws-cli/1.16.63
```

설정하려면 AWS CLI

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 AWS Command Line Interface 사용 설명서에서 다음 토픽을 참조하세요.
 - [AWS Command Line Interface설치](#)
 - [AWS CLI구성](#)
2. AWS CLI config파일에 관리자 사용자의 이름이 지정된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 프로파일 명명에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드의 [프로파일 명명](#)을 참조하십시오.

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

사용 가능한 AWS 지역 목록은 의 [지역 및 엔드포인트](#)를 참조하십시오. Amazon Web Services 일반 참조

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
```

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

다음 단계

3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행

3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행

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

이 섹션은 다음 주제를 포함합니다:

- [두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.](#)
- [입력 스트림에 샘플 레코드를 작성합니다.](#)
- [Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)

두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(ExampleInputStream 및 ExampleOutputStream)을 생성하세요. 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 업데이트](#)를 참조하세요.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림 (ExampleInputStream) 을 생성하려면 다음 Amazon Kinesis 명령을 create-stream AWS CLI 사용하십시오.

```
$ 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
```

입력 스트림에 샘플 레코드를 작성합니다.

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 스트림에 쓰겠습니다.

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"))

```

2. 이 자습서의 뒷부분에서 `stock.py` 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python stock.py
```

Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.

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/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 Kinesis 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 내용을 알아보려면 [런타임 속성](#) 섹션을 참조하세요.

애플리케이션 코드를 컴파일하십시오.

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 내용을 알아보려면 [연습을 완료하기 위한 사전 요구 사항](#) 섹션을 참조하십시오.

Note

1.11 이전 버전의 Apache Flink에서 Kinesis 커넥터를 사용하려면 커넥터의 소스 코드를 다운로드하고 [Apache Flink 설명서](#)에 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 도구를 사용합니다. 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 내용을 알아보려면 [the section called “애플리케이션의 Apache Flink 버전 지정”](#) 섹션을 참조하세요.

- 귀하의 개발 환경 사용. 자세한 내용을 알아보려면 해당 개발 환경 설명서를 참조하십시오.

패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. 를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형 (JAR 또는 ZIP) 을 지정합니다. AWS CLI

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
```

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

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. 를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 생성합니다. AWS CLI

주제

- [애플리케이션 \(콘솔\) 생성 및 실행](#)
- [애플리케이션 생성 및 실행 \(AWS CLI\)](#)

애플리케이션 (콘솔) 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 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-analytcs-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

애플리케이션을 실행합니다

1. MyApplication페이지에서 [Run] 을 선택합니다. 작업을 확인합니다.
2. 애플리케이션이 실행 중이면 페이지를 새로 고칩니다. 콘솔에 애플리케이션 그래프가 표시됩니다.

애플리케이션을 중지합니다.

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` 를 사용하여 Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다. AWS CLI Apache Flink용 관리 서비스는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 이와 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 연결](#)을 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. 역할 생성을 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.

- 요약 페이지에서 권한 탭을 선택합니다.
- 정책 연결을 선택합니다.
- 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.
- 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.

Apache Flink용 관리형 서비스 애플리케이션 생성

- 다음 JSON 코드를 `create_request.json`이라는 파일에 저장합니다. 샘플 역할을 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"
        }
      }
    ]
  }
}

```

- 위의 요청과 함께 [CreateApplication](#) 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticstv2 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

- 다음 JSON 코드를 `start_request.json`이라는 파일에 저장합니다.

```
{
  "ApplicationName": "test",
```

```

    "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
        "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
      }
    }
  }
}

```

- 위의 요청과 함께 [StartApplication](#) 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

- 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```

{
  "ApplicationName": "test"
}

```

- 다음 요청과 함께 [StopApplication](#) 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticstv2 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 [the 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.json`이라는 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 kinesisanalyticstv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 [UpdateApplication](#) AWS CLI 작업을 사용합니다.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션 이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 [the section called “두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.”](#) 섹션에서 선택한 접미사로 업데이트합니다.

```
{
  "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:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)

- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/kinesis>에서 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 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/flink>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication

8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

Apache Flink용 관리형 서비스의 이전 버전 (레거시) 예제

Note

현재 예제는 을 참조하십시오. [예제](#)

이 섹션은 Managed Service for Apache Flink에서 애플리케이션을 만들고 사용하는 방법의 예제들을 제공합니다. 여기에는 Apache Flink 애플리케이션용 관리 서비스를 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](#)
- [스칼라 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: Apache Flink용 관리형 서비스 애플리케이션을 사용하여 MSK 클러스터의 한 주제에서 VPC의 다른 주제로 데이터 복제](#)
- [예: Kinesis 데이터 스트림과 함께 EFO 소비자 사용](#)
- [예: Firehose에 쓰기](#)
- [예: 다른 계정의 Kinesis 스트림에서 읽기](#)
- [자습서: Amazon MSK에서 사용자 지정 트러스트 스토어 사용](#)

예: 텀블링 윈도우

Note

현재 예제는 을 참조하십시오. [예제](#)

이 연습에서는 텀블링 윈도우를 사용하여 데이터를 집계하는 Managed Service for Apache Flink 애플리케이션을 만들어 봅니다. Flink에서 집계(aggregation)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](#)

- [입력 스트림에 샘플 레코드 쓰기](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [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 하나요?](#) 로그인 명칭(예: `ka-app-code-<username>`)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

입력 스트림에 샘플 레코드 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 됩니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 [Git 설치](#)를 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 Kinesis 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다.

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

- 다음 가져오기 설명을 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 코드를 컴파일하십시오.

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 [시작하기 \(API\)](#) [DataStream](#) 자습서의 [사전 조건](#) 섹션을 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.

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.


Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성


1. <https://console.aws.amazon.com/flink>에서 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 리전을 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.

애플리케이션을 실행합니다

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 텀블링 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: 슬라이딩 윈도우

Note

현재 예제는 [을 참조하십시오](#) [예제](#).

Note

이 연습에 필수 사전 조건을 설정하려면 먼저 [시작하기 \(API\) DataStream](#) 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- [종속 리소스 생성](#)
- [입력 스트림에 샘플 레코드 쓰기](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [리소스를 정리하세요. AWS](#)

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 하나요?](#) 로그인 명칭(예: `ka-app-code-<username>`)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

입력 스트림에 샘플 레코드 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 됩니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 응용 프로그램 코드는 에서 사용할 수 [GitHub](#) 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 [Git 설치](#)를 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 Kinesis 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 설명을 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 코드를 컴파일하십시오.

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 [시작하기 \(API\) DataStream](#) 자습서의 [사전 조건](#) 섹션을 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.

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

이 섹션에서는 [종속 리소스 생성](#) 섹션에서 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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 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 병렬성을 구성합니다.

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 병렬성을 높이려면 다음 작업을 사용하십시오. AWS CLI

```
aws kinesisanalyticstv2 update-application
  --application-name MyApplication
  --current-application-version-id <VersionId>
  --application-configuration-update "{\"FlinkApplicationConfigurationUpdate\": { \"ParallelismConfigurationUpdate\": { \"ParallelismUpdate\": 5, \"ConfigurationTypeUpdate\": \"CUSTOM\" } } }"
```

[DescribeApplication](#) 또는 [ListApplications](#) 작업을 사용하여 현재 애플리케이션 버전 ID를 검색할 수 있습니다.

애플리케이션을 실행합니다

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.

리소스를 정리하세요. AWS

이 섹션에는 슬라이딩 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

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

2. 탐색 바에서 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](#)
- [입력 스트림에 샘플 레코드 쓰기](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [애플리케이션 코드 수정](#)
- [애플리케이션 코드 컴파일](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [애플리케이션 출력을 확인합니다.](#)
- [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 하나요?](#) 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 CloudWatch 리소스가 아직 없는 경우 해당 리소스를 생성합니다.

- /AWS/KinesisAnalytics-java/MyApplication라는 로그 그룹.
- kinesis-analytics-log-stream라는 로그 스트림.

입력 스트림에 샘플 레코드 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 됩니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 [Git 설치](#)를 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 Kinesis 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다.

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

- 가져오기 설명을 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](#).

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 조건` 섹션을 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.

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 사용하여 다음과 같이 지정됩니다.

- 애플리케이션 명칭에 **MyApplication**을 입력합니다.
- 런타임에서 Apache Flink를 선택합니다.
- 버전은 Apache Flink 버전 1.15.2(권장 버전)로 그대로 둡니다.

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 리전을 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.

애플리케이션을 실행합니다

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 출력을 확인합니다.

Amazon S3 콘솔에서 S3 버킷의 데이터 폴더를 엽니다.

몇 분 후 애플리케이션에서 집계된 데이터를 포함하는 객체가 나타납니다.

Note

Flink에서 집계(aggregation)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 “아파치 플링크 스트리밍 자바 코드를 업로드하세요.”](#) 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 빈도를 설정합니다.](#)
- [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
SimpleStringEncodingEncoder<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 `DateTimeBucketAssigner`를 사용하여 S3 버킷에 폴더를 만듭니다. `DateTimeBucketAssigner`에서는 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.

`BucketAssigner`의 사용자 지정 구현에서는 [컨텍스트](#) 파라미터를 사용하여 레코드에 대한 추가 정보를 가져와 대상 폴더를 결정할 수 있습니다.

읽기 빈도를 설정합니다.

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. [UpdateApplication](#) 작업에 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. [ListApplications](#) 작업을 사용하여 애플리케이션 버전을 검색할 수 있습니다.

2. `S3StreamingSinkJob.java` 파일 시작 부분에 다음 가져오기 설명을 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 인 코딩 형식](#)을 참조하세요.

AWS 리소스 정리

이 섹션에는 Amazon S3 자습서에서 생성한 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션을 삭제하십시오.](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션을 삭제하십시오.

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication

3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.

Amazon S3 객체 및 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 바에서 역할을 선택합니다.
7. 키네시스-애널리틱스- -US-West-2 역할을 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

자습서: Apache Flink용 관리형 서비스 애플리케이션을 사용하여 MSK 클러스터의 한 주제에서 VPC의 다른 주제로 데이터 복제

Note

현재 예제는 [을 참조하십시오. 예제](#)

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](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [애플리케이션 생성](#)
- [애플리케이션을 구성합니다.](#)
- [애플리케이션을 실행합니다](#)
- [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 만들기](#)에서 `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 AWS KafkaTutorialTopicDestination
```

- 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 지역을 사용해야 합니다.

애플리케이션 코드 생성

이 섹션에서는 애플리케이션 JAR 파일을 다운로드하고 컴파일합니다. Java 11을 사용하는 것이 좋습니다.

이 예제의 Java 애플리케이션 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

- 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 [Git 설치](#)를 참조하세요.
- 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

- 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.
- 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

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

이 섹션에서는 [시작하기 \(API\) DataStream](#) 자습서에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

Note

시작 자습서에서 Amazon S3 버킷을 삭제한 경우 [the section called “아파치 플링크 스트리밍 자바 코드를 업로드하세요.”](#) 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/flink> 에서 아파치 플링크용 관리형 서비스 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 리전을 사용하여 다음과 같이 지정됩니다.

- 정책: `kinesis-analytics-service-MyApplication-us-west-2`
- 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션을 구성합니다.

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	AWS KafkaTutorialTopic
KafkaSource	bootstrap.servers	<i>### ### ##### ## ##</i>

그룹 ID	키	값
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	AWS KafkaTutorialTopic Destination
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 속성](#) 섹션을 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.

애플리케이션을 실행합니다

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 소비](#)의 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 AWS KafkaTutorialTopicDestination --from-
beginning
```

대상 주제에 레코드가 표시되지 않는 경우 [문제 해결](#) 주제의 [VPC의 리소스에 액세스할 수 없음](#) 섹션을 참조하세요.

예: Kinesis 데이터 스트림과 함께 EFO 소비자 사용

Note

현재 예는 을 참조하십시오. [예제](#)

이 연습에서는 EFO ([항상된 팬아웃](#)) 소비자를 사용하여 Kinesis 데이터 스트림에서 읽는 Apache Flink [용 관리형](#) 서비스 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 팬아웃](#)을 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](#)
- [입력 스트림에 샘플 레코드 쓰기](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [리소스 정리 AWS](#)

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 하나요?](#) 로그인 명칭(예: `ka-app-code-<username>`)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

입력 스트림에 샘플 레코드 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 됩니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 응용 프로그램 코드는 에서 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 [Git 설치](#)를 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 코드를 컴파일하십시오.

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 [시작하기 \(API\) DataStream](#) 자습서의 [사전 조건](#) 섹션을 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.

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 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.recorderpublisher	EF0

그룹 ID	키	값
ConsumerConfigProperties	flink.stream.efs.consumername	basic-efs-flink-app
ConsumerConfigProperties	INPUT_STREAM	ExampleInputStream
ConsumerConfigProperties	flink.inputstream.initpos	LATEST
ConsumerConfigProperties	AWS_REGION	us-west-2

- 속성에서 그룹 생성을 선택합니다.
- 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	OUTPUT_STREAM	ExampleOutputStream
ProducerConfigProperties	AWS_REGION	us-west-2
ProducerConfigProperties	AggregationEnabled	false

- 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
- CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
- 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.

애플리케이션을 실행합니다

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`

리소스 정리 AWS

이 섹션에는 efo Window 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷 삭제](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 <username>선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: Firehose에 쓰기

Note

현재 예제는 을 참조하십시오. [예제](#)

이 연습에서는 Kinesis 데이터 스트림을 소스로 사용하고 Firehose 스트림을 싱크로 사용하는 Apache Flink용 관리형 서비스 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 리소스를 생성하십시오.](#)
- [입력 스트림에 샘플 레코드를 작성합니다.](#)
- [Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.](#)
- [애플리케이션 코드를 컴파일하세요.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다.](#)
- [AWS 리소스 정리](#)

종속 리소스를 생성하십시오.

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 Firehose 스트림입니다.
- 애플리케이션 코드를 저장할 Amazon S3 버킷(ka-app-code-*<username>*)

콘솔을 사용하여 Kinesis 스트림, Amazon S3 버킷 및 Firehose 스트림을 생성할 수 있습니다. 이러한 리소스를 만드는 방법 설명은 다음 주제를 참조하십시오.

- Amazon Kinesis Data Streams 개발자 안내서의 [데이터 스트림 만들기 및 업데이트](#). 데이터 스트림 **ExampleInputStream**의 이름을 지정합니다.
- [Amazon Data Firehose 개발자 안내서에서 Amazon Kinesis Data Firehose 전송 스트림을 생성하십시오](#). Firehose 스트림의 이름을 지정하세요. **ExampleDeliveryStream** Firehose 스트림을 생성할 때는 Firehose 스트림의 S3 대상 및 IAM 역할도 생성해야 합니다.
- Amazon Simple Storage Service 사용 설명서의 [S3 버킷을 생성하려면 어떻게 해야 하나요?](#) 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

입력 스트림에 샘플 레코드를 작성합니다.

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 됩니다.

Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.

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 Kinesis 소스를 사용하여 소스 스트림에서 읽습니다. 다음 스니펫은 Kinesis 소스를 생성합니다.

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

- 애플리케이션은 Firehose 싱크를 사용하여 Firehose 스트림에 데이터를 씁니다. 다음 스니펫은 Firehose 싱크를 만듭니다.

```
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 코드를 컴파일하세요.

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 [시작하기 \(API\) DataStream](#) 자습서의 [사전 조건](#) 섹션을 참조하세요.
2. 다음 애플리케이션에 Kinesis 커넥터를 사용하려면 Apache Maven을 다운로드, 빌드 및 설치해야 합니다. 자세한 내용은 [the section called “아파치 플링크 Kinesis Streams 커넥터를 이전 아파치 플링크 버전과 함께 사용하기”](#) 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.

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

이 섹션에서는 [종속 리소스를 생성하십시오](#). 섹션에서 생성한 Amazon S3 버킷에 애플리케이션 코드를 업로드합니다.

애플리케이션 코드 업로드하기

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.

2. 콘솔에서 ka-app-code- 버킷을 선택한 다음 업로드를 선택합니다. <username>
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.

Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다.

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. 를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 생성합니다. AWS CLI

주제

- [애플리케이션 \(콘솔\) 생성 및 실행](#)
- [애플리케이션 생성 및 실행 \(AWS CLI\)](#)

애플리케이션 (콘솔) 생성 및 실행

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 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 스트림 및 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"
    }
]
}

```

애플리케이션을 구성합니다.

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-analytcs-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

애플리케이션을 실행합니다

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션을 중지하세요.

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.

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 를 사용하여 Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다. AWS CLI

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 AKReadStreamWriteSinkStream권한 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 관리 형 정책 생성 및 연결](#)을 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. 역할 생성을 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 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.

Apache Flink용 관리형 서비스 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 리소스를 생성하십시오.”](#) 섹션에서 선택한 접미사(`ka-app-code-<username>`)로 바꿉니다. 서비스 실행 역할의 샘플 계정 ID(`012345678901`)를 사용자 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 kinesisanalyticstv2 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.json`이라는 파일에 저장합니다.

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

2. 위의 요청과 함께 [StartApplication](#) 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```
{
```

```
"ApplicationName": "test"
}
```

- 다음 요청과 함께 [StopApplication](#) 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticstv2 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 [the section called “로깅 설정”](#).

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 [UpdateApplication](#) AWS CLI 작업을 사용합니다.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름을 지정하여 UpdateApplication 호출합니다. AWS CLI

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 [the section called “종속 리소스를 생성하십시오.”](#) 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 시작 자습서에서 생성한 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Firehose 스트림을 삭제하세요](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제하세요.](#)
- [리소스 삭제 CloudWatch](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 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 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.

Firehose 스트림을 삭제하세요

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Firehose 패널에서 을 선택합니다. ExampleDeliveryStream
3. ExampleDeliveryStream페이지에서 Firehose 스트림 삭제를 선택한 다음 삭제를 확인합니다.

Amazon S3 객체 및 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.
4. Firehose 스트림의 대상으로 Amazon S3 버킷을 만든 경우 해당 버킷도 삭제하십시오.

IAM 리소스를 삭제하세요.

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. Firehose 스트림에 대한 새 정책을 만든 경우 해당 정책도 삭제하세요.
7. 탐색 모음에서 역할을 선택합니다.
8. 키네시스-애널리틱스- -US-West-2 역할을 선택합니다. MyApplication
9. 역할 삭제를 선택하고 삭제를 확인합니다.
10. Firehose 스트림용 새 역할을 만든 경우 해당 역할도 삭제하세요.

리소스 삭제 CloudWatch

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: 다른 계정의 Kinesis 스트림에서 읽기

Note

현재 예는 을 참조하십시오 [예제](#).

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, 실행](#)

사전 조건

- 이 자습서에서는 다른 계정의 Kinesis 스트림에서 데이터를 읽도록 시작하기 예제를 수정합니다. 계속하기 전에 [시작하기 \(API\) DataStream](#) 자습서를 완료하세요.
- 이 가이드를 완료하려면 두 개의 AWS 계정이 필요합니다. 하나는 소스 스트림용이고 다른 하나는 애플리케이션과 싱크 스트림용입니다. 시작하기 튜토리얼에서 사용한 AWS 계정을 애플리케이션 및 싱크 스트림에 사용하세요. 소스 스트림에는 다른 AWS 계정을 사용하세요.

설치

이름이 지정된 프로필을 사용하여 두 AWS 계정에 접속할 수 있습니다. 두 계정의 지역 및 연결 정보가 포함된 프로필 두 개를 포함하도록 AWS 자격 증명 및 구성 파일을 수정하십시오.

다음 예제 보안 인증 파일에는 두 개의 명명된 프로필, ka-source-stream-account-profile 및 ka-sink-stream-account-profile이(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.

에서 명명된 프로필을 사용하는 방법에 대한 자세한 내용은 [AWS Command Line Interface 설명서의 명명된 프로필을](#) 참조하십시오. AWS CLI

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 원본 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 설명서](#)에서 다음 주제를 참조하세요.

- [IAM 역할 생성](#)
- [IAM 정책 생성](#)

싱크 계정 역할 및 정책

1. 시작하기 자습서에서 `kinesis-analytics-service-MyApplication-us-west-2` 정책을 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 AWS 서비스를 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 업데이트 합니다.

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, 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.jar 파일을 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

Note

현재 예는 을 참조하십시오. [예제](#)

현재 데이터 소스 API

[최신 데이터 소스 API를 사용하는 경우, 애플리케이션은 여기에 설명된 Amazon 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 [여기](#)에서 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 로드합니다. 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 Kafka truststore", ex);
    }
}
```

Note

Apache Flink를 사용하려면 트러스트 스토어가 [JKS 형식](#)이어야 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 신뢰 저장소의 예로 상호 TLS (MTLS) 인증이 활성화된 Amazon MSK 클러스터를 사용할 것입니다. 이는 MSK 클러스터 노드가 AWS Certificate Manager 사설 인증 기관 (ACM Private CA)에서 발급한 서버 인증서를 사용한다는 것을 의미합니다. 이 인증서는 사용자 계정

및 지역 전용이므로 Flink 응용 프로그램을 실행하는 JVM (Java Virtual Machine) 의 기본 신뢰 저장소에서 신뢰하지 않습니다.

Note

- 키스토어는 애플리케이션이 확인을 위해 서버나 클라이언트 모두에 제공해야 하는 개인 키와 ID 인증서를 저장하는 데 사용됩니다.
- 신뢰 저장소는 SSL 연결을 통해 서버가 제공하는 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.

- [\(아마존 AWSEC2 또는 아마존 EKS\) 에서 호스팅되는 사용자 지정 아파치 카프카 클러스터](#)
- [에서 호스팅되는 컨플루언트 Kafka 클러스터](#) AWS
- [AWS Direct Connect](#) 또는 VPN을 통해 액세스하는 온프레미스 Kafka 클러스터

이 자습서는 다음 섹션을 포함하고 있습니다.

- [Amazon MSK 클러스터를 사용하여 VPC를 생성합니다.](#)
- [사용자 지정 신뢰 저장소를 만들어 클러스터에 적용하세요.](#)
- [애플리케이션 코드 생성](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [애플리케이션 생성](#)
- [애플리케이션을 구성합니다.](#)
- [애플리케이션을 실행합니다](#)
- [애플리케이션 테스트](#)

Amazon MSK 클러스터를 사용하여 VPC를 생성합니다.

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 만들기**에서 `kafka-topics.sh --create` 명령을 반복하여 AWS KafkaTutorialTopicDestination라는 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 자습서의 단계와 사전 필수 자습서를 따를 때는 코드, 명령 및 콘솔 항목에서 선택한 AWS 지역을 사용해야 합니다.

사용자 지정 신뢰 저장소를 만들어 클러스터에 적용하세요.

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 사용할 수 GitHub 있습니다. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 [Git 설치](#)를 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/ CustomKeystore에 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.

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

이 섹션에서는 [시작하기 \(API\) DataStream](#) 자습서에서 생성한 Amazon S3 버킷으로 애플리케이션 코드를 업로드합니다.

Note

시작 자습서에서 Amazon S3 버킷을 삭제한 경우 [the section called “아파치 플링크 스트리밍 자바 코드를 업로드하세요.”](#) 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/flink>에서 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 리전을 사용하여 다음과 같이 지정됩니다.

- 정책: `kinesis-analytics-service-MyApplication-us-west-2`
- 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션을 구성합니다.

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	AWS KafkaTutorialTopic
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	AWS KafkaTutorialTopic Destination
KafkaSink	bootstrap.servers	### ### ##### ## ##
KafkaSink	security.protocol	SSL

그룹 ID	키	값
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 속성](#) 섹션을 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.

애플리케이션을 실행합니다

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 소비](#)의 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 AWS KafkaTutorialTopicDestination --from-
beginning
```

대상 주제에 레코드가 표시되지 않는 경우 [문제 해결](#) 주제의 [VPC의 리소스에 액세스할 수 없음](#) 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: Python으로 Amazon S3에 스트리밍 데이터 전송](#)

예: Python에서 텀블링 윈도우 만들기

Note

현재 예제는 을 참조하십시오. [예제](#)

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](#)
- [입력 스트림에 샘플 레코드 쓰기](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [아파치 플링크 스트리밍 Python 코드 압축 및 업로드](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [리소스를 정리하세요. AWS](#)

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?](#) 로그인 명칭(예: `ka-app-code-<username>`)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

입력 스트림에 샘플 레코드 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. 계정 자격 증명과 기본 지역을 사용하도록 구성해야 합니다. AWS CLI 를 AWS CLI 구성하려면 다음을 입력합니다.

```
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 됩니다.

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 설치](#)를 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.jar](#)의 Kinesis Flink 커넥터를 사용합니다.

아파치 플링크 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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 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. 그룹 ID에 **kinesis.analytics.flink.run.options**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 내용을 알아보려면 [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

- 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
- CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
- 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 정책을 편집합니다.

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"
    }
  ]
}

```

애플리케이션을 실행합니다

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.

리소스를 정리하세요. AWS

이 섹션에는 텀블링 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.

2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.

4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: Python에서 슬라이딩 윈도우 만들기

Note

현재 예제는 [을 참조하십시오](#)[예제](#).

Note

이 연습에 필수 사전 조건을 설정하려면 먼저 [시작하기 \(Python\)](#) 연습을 완료하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- [종속 리소스 생성](#)
- [입력 스트림에 샘플 레코드 쓰기](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [아파치 플링크 스트리밍 Python 코드 압축 및 업로드](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [리소스를 정리하세요. AWS](#)

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 하나요?](#) 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

입력 스트림에 샘플 레코드 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. 계정 자격 증명과 기본 지역을 사용하도록 구성해야 합니다. AWS CLI 를 AWS CLI 구성하려면 다음을 입력합니다.

```
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 됩니다.

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 설치](#)를 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](#)

아파치 플링크 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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 사용하여 다음과 같이 지정됩니다.

- 정책: `kinesis-analytics-service-MyApplication-us-west-2`
- 역할: `kinesisanalytics-MyApplication-us-west-2`

애플리케이션 구성

1. MyApplication 페이지에서 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
<code>consumer.config.0</code>	<code>input.stream.name</code>	<code>ExampleInputStream</code>
<code>consumer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>
<code>consumer.config.0</code>	<code>scan.stream.initpos</code>	<code>LATEST</code>

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

- 속성에서 그룹 추가를 다시 선택합니다. 그룹 ID에 **kinesis.analytics.flink.run.options**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 내용을 알아보려면 [코드 파일 지정](#) 섹션을 참조하세요.
- 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

- 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
- CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
- 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 정책을 편집합니다.

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"
  }
]
}

```

애플리케이션을 실행합니다

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.

리소스를 정리하세요. AWS

이 섹션에는 슬라이딩 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: Python으로 Amazon S3에 스트리밍 데이터 전송

Note

현재 예제는 [을 참조하십시오](#) [예제](#).

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](#)
- [입력 스트림에 샘플 레코드 쓰기](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [아파치 플링크 스트리밍 Python 코드 압축 및 업로드](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [리소스를 정리하세요. AWS](#)

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 하나요?](#) 로그인 명칭(예: `ka-app-code-<username>`)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

입력 스트림에 샘플 레코드 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. 계정 자격 증명과 기본 지역을 사용하여 도록 구성해야 합니다. AWS CLI 를 AWS CLI 구성하려면 다음을 입력합니다.

```
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 됩니다.

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 설치](#)를 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](#)

아파치 플링크 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 압축합니다streaming-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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 리전을 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-analytcs-MyApplication-us-west-2` 생성/업데이트를 선택합니다.
4. 속성에서 그룹 추가를 선택합니다.
5. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
<code>consumer.config.0</code>	<code>input.stream.name</code>	<code>ExampleInputStream</code>
<code>consumer.config.0</code>	<code>aws.region</code>	<code>us-west-2</code>
<code>consumer.config.0</code>	<code>scan.stream.initpos</code>	<code>LATEST</code>

저장을 선택합니다.

6. 속성에서 그룹 추가를 다시 선택합니다. 그룹 ID에 `kinesis.analytics.flink.run.options`를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 내용을 알아보려면 [코드 파일 지정](#) 섹션을 참조하세요.
7. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
<code>kinesis.analytics.flink.run.options</code>	<code>python</code>	<code>streaming-file-sink.py</code>

그룹 ID	키	값
kinesis.analytics.flink.run.options	jarfile	S3Sink/lib/flink-sql-connector-kinesis-1.15.2.jar

- 속성에서 그룹 추가를 다시 선택합니다. 그룹 ID에 **sink.config.0**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 내용을 알아보려면 [코드 파일 지정](#) 섹션을 참조하세요.
- 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	<i>bucket-name</i>

- 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
- CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
- 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 정책을 편집합니다.

IAM 정책을 편집하여 Kinesis Data Streams에 액세스할 수 있는 권한을 추가합니다.

- <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>/*"
    ]
  }
]
}

```

애플리케이션을 실행합니다

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.

리소스를 정리하세요. AWS

이 섹션에는 슬라이딩 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream
3. ExampleInputStream페이지에서 Kinesis 스트림 삭제를 선택한 다음 삭제를 확인합니다.

Amazon S3 객체 및 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

스칼라 예제

다음 예제는 Apache Flink와 함께 Scala를 사용하여 애플리케이션을 만드는 방법을 보여줍니다.

주제

- [예: Scala에서 텀블링 윈도우 만들기](#)
- [예: Scala에서 슬라이딩 윈도우 만들기](#)
- [예: 스트리밍 데이터를 스칼라의 Amazon S3로 전송](#)

예: Scala에서 텀블링 윈도우 만들기

Note

현재 예제는 을 참조하십시오. [예제](#)

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-arcrchive에 Scala 종속성을 추가해야 합니다.

Flink 1.15의 Scala 변경 사항에 대한 자세한 내용은 [Scala Free in One Fifteen](#)을 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 코드를 다운로드하고 검토하세요.](#)
- [애플리케이션 코드 컴파일 및 업로드](#)
- [애플리케이션 생성 및 실행 \(콘솔\)](#)
- [애플리케이션 생성 및 실행\(CLI\)](#)
- [애플리케이션 코드 업데이트](#)
- [AWS 리소스를 정리하세요.](#)

애플리케이션 코드를 다운로드하고 검토하세요.

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 설치](#)를 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 Kinesis 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 참조](#)하세요.

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 설치](#)를 참조하십시오. 또한 Java Development Kit(JDK)를 설치해야 합니다. [연습 완료를 위한 사전 조건](#)을 참조하세요.

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 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/flink>에서 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 둡니다.
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 리전을 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.

애플리케이션을 구성하려면

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

그룹 ID	키	값
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 정책을 편집하십시오.

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"
    }
]
}

```

애플리케이션을 실행합니다

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 중지를 선택합니다. 작업을 확인합니다.

애플리케이션 생성 및 실행(CLI)

이 섹션에서는 를 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 실행합니다. AWS Command Line Interface kinesisanalyticsv2 AWS CLI 명령을 사용하여 아파치 플링크용 관리 서비스 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 설명과 싱크 스트림의 쓰기 작업에 대한 권한을 부여하는 설명 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.

username을 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 연결](#)을 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. 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 생성](#)에서 만든 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.json`이라는 파일에 저장합니다. 샘플 역할을 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 사용자 계정 ID로 바꿉니다. `ServiceExecutionRole`에는 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 [CreateApplication](#) 요청과 함께 를 실행하여 애플리케이션을 생성하십시오.

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

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션을 시작합니다.

이 섹션에서는 [StartApplication](#) 액션을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 `start_request.json`이라는 파일에 저장합니다.

```
{
```

```

    "ApplicationName": "tumbling_window",
    "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
        "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
      }
    }
  }
}

```

- 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 액션을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

- 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```

{
  "ApplicationName": "tumbling_window"
}

```

- 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 참조](#)하십시오.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 [UpdateApplication](#) 액션을 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.json`이라는 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 kinesisanalyticstv2 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 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 참조하세요.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름, 새 객체 버전을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사(<username>)를 [종속 리소스 생성](#) 섹션에서 선택한 접미사로 업데이트합니다.

```
{
  "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 리소스를 정리하세요.

이 섹션에는 텀블링 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: Scala에서 슬라이딩 윈도우 만들기

Note

현재 예제는 [을 참조하십시오](#) [예제](#).

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-arcrchive에 Scala 종속성을 추가해야 합니다.

Flink 1.15의 Scala 변경 사항에 대한 자세한 내용은 [Scala Free in One Fifteen](#)을 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 코드를 다운로드하고 검토하세요.](#)
- [애플리케이션 코드 컴파일 및 업로드](#)
- [애플리케이션 생성 및 실행 \(콘솔\)](#)
- [애플리케이션 생성 및 실행\(CLI\)](#)
- [애플리케이션 코드 업데이트](#)
- [AWS 리소스를 정리하세요.](#)

애플리케이션 코드를 다운로드하고 검토하세요.

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 설치](#)를 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 Kinesis 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 속성](#)을 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 설치](#)를 참조하십시오. 또한 Java Development Kit(JDK)를 설치해야 합니다. [연습 완료를 위한 사전 조건](#)을 참조하세요.

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 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/flink>에서 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 둡니다.
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 리전을 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.

애플리케이션을 구성하려면

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-analytcs-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.initializers	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 정책을 편집하십시오.

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"
}
]

```

```
}

```

애플리케이션을 실행합니다

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 중지를 선택합니다. 작업을 확인합니다.

애플리케이션 생성 및 실행(CLI)

이 섹션에서는 `aws` 를 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 실행합니다. AWS Command Line Interface `kinesisanalyticsv2` AWS CLI 명령을 사용하여 아파치 플링크용 관리 서비스 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 설명과 싱크 스트림의 쓰기 작업에 대한 권한을 부여하는 설명 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.

username을 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 연결](#)을 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. 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 생성](#)에서 만든 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.json`이라는 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 사용자 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 [CreateApplication](#) 요청과 함께 `aws kinesisanalyticsv2 create-application` 를 실행하여 애플리케이션을 생성하십시오.

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

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션을 시작합니다.

이 섹션에서는 [StartApplication](#) 액션을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 `start_request.json`이라는 파일에 저장합니다.

```

{
  "ApplicationName": "sliding_window",
  "RunConfiguration": {
    "ApplicationRestoreConfiguration": {
      "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
    }
  }
}

```

2. 위의 요청과 함께 `StartApplication` 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 액션을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

1. 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```
{
  "ApplicationName": "sliding_window"
}
```

2. 위의 요청과 함께 `StopApplication` 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticstv2 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](#) 액션을 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.json`이라는 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"
          }
        }
      ]
    }
  }
}

```

- 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```

aws kinesisanalyticstv2 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 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 참조하세요.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름, 새 객체 버전을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사(<username>)를 [중속 리소스 생성](#) 섹션에서 선택한 접미사로 업데이트합니다.

```
{
  "ApplicationName": "sliding_window",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
        "S3ContentLocationUpdate": {
          "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
          "FileKeyUpdate": "-1.0.jar",
          "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhypvDU"
        }
      }
    }
  }
}
```

AWS 리소스를 정리하세요.

이 섹션에는 슬라이딩 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

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

2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

예: 스트리밍 데이터를 스칼라의 Amazon S3로 전송

Note

현재 예제는 [을 참조하십시오](#).

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-archiv에 Scala 종속성을 추가해야 합니다.

Flink 1.15의 Scala 변경 사항에 대한 자세한 내용은 [Scala Free in One Fifteen](#)을 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 코드를 다운로드하여 검토하십시오.](#)
- [애플리케이션 코드 컴파일 및 업로드](#)
- [애플리케이션 생성 및 실행 \(콘솔\)](#)
- [애플리케이션 생성 및 실행\(CLI\)](#)

- [애플리케이션 코드 업데이트](#)
- [AWS 리소스를 정리하세요.](#)

애플리케이션 코드를 다운로드하여 검토하십시오.

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 설치](#)를 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 Kinesis 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 속성](#)을 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 설치](#)를 참조하십시오. 또한 Java Development Kit(JDK)를 설치해야 합니다. [연습 완료를 위한 사전 조건](#)을 참조하세요.

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 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/flink>에서 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 둡니다.
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 정

책을 사용하여 종속 리소스에 액세스합니다. 이러한 IAM 리소스의 이름은 애플리케이션 명칭과 리전을 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.

애플리케이션을 구성하려면

1. MyApplication 페이지에서 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	<code>input.stream.name</code>	<code>ExampleInputStream</code>
ConsumerConfigProperties	<code>aws.region</code>	<code>us-west-2</code>
ConsumerConfigProperties	<code>flink.stream.initpos</code>	<code>LATEST</code>

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 정책을 편집하십시오.

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"
  }
]
}

```

애플리케이션을 실행합니다

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 중지를 선택합니다. 작업을 확인합니다.

애플리케이션 생성 및 실행(CLI)

이 섹션에서는 `aws`를 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 실행합니다. AWS Command Line Interface `kinesisanalyticsv2` AWS CLI 명령을 사용하여 아파치 플링크용 관리 서비스 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 설명과 싱크 스트림의 쓰기 작업에 대한 권한을 부여하는 설명 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.

username을 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 연결](#)을 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. 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 생성](#)에서 만든 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.json`이라는 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 사용자 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 [CreateApplication](#) 요청과 함께 `aws` 를 실행하여 애플리케이션을 생성하십시오.

```
aws kinesisanalyticstv2 create-application --cli-input-json file://create_request.json
```

애플리케이션이 생성되었습니다. 다음 단계에서는 애플리케이션을 시작합니다.

애플리케이션을 시작합니다.

이 섹션에서는 [StartApplication](#) 액션을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 `start_request.json`이라는 파일에 저장합니다.

```

{{
  "ApplicationName": "s3_sink",
  "RunConfiguration": {
    "ApplicationRestoreConfiguration": {
      "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
    }
  }
}

```

2. 위의 요청과 함께 `StartApplication` 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 액션을 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 kinesisanalyticstv2 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 참조](#)하십시오.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 [UpdateApplication](#) 액션을 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.json`이라는 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"
        }
      }
    ]
  }
}

```

- 이전 요청과 함께 UpdateApplication 작업을 실행하여 환경 속성을 업데이트하십시오.

```
aws kinesisanalyticstv2 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 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 참조하세요.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름, 새 객체 버전을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사(<username>)를 [종속 리소스 생성](#) 섹션에서 선택한 접미사로 업데이트합니다.

```
{
  "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": "SAMPLEUehYngP87ex1nzYIGYgfhyvpvDU"
        }
      }
    }
  }
}
```

AWS 리소스를 정리하세요.

이 섹션에는 텀블링 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Kinesis Data Streams 패널에서 을 선택합니다. ExampleInputStream

3. ExampleInputStream 페이지에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

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 Flink](#)를 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 [스튜디오 노트북 만들기 튜토리얼](#) 섹션을 참조하세요. Apache Zeppelin에 대한 자세한 내용은 [Apache Zeppelin 문서](#)를 참조하세요.

노트북을 사용하면 SQL, Python 또는 Scala의 Apache Flink [Table API 및 SQL](#) 또는 Scala의 [DataStream API](#)를 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.

- [스튜디오 노트북 런타임 버전](#)
- [Studio 노트북 생성](#)
- [스트리밍 데이터의 대화형 분석](#)
- [지속 가능한 상태의 애플리케이션으로 배포](#)
- [Studio 노트북의 IAM 권한](#)
- [커넥터 및 종속성 사용](#)

- [사용자 정의 함수를 구현하십시오.](#)
- [체크포인트 활성화](#)
- [Studio 런타임 업그레이드](#)
- [작업 대상 AWS Glue](#)
- [예시 및 자습서](#)
- [문제 해결](#)
- [부록: 사용자 지정 IAM 정책 생성](#)

스튜디오 노트북 런타임 버전

Apache Flink Studio용 Amazon Managed Service를 사용하면 대화형 노트북에서 표준 SQL, Python 및 Scala를 사용하여 실시간으로 데이터 스트림을 쿼리하고 스트림 처리 애플리케이션을 구축 및 실행할 수 있습니다. [스튜디오 노트북은 Apache Zeppelin으로 구동되며 Apache Flink를 스트림 처리 엔진으로 사용합니다.](#)

Note

2024년 11월 5일부터 아파치 플링크 버전 1.11이 포함된 스튜디오 런타임을 더 이상 사용하지 않을 예정입니다. 이 날짜부터 이 버전을 사용하여 새 노트북을 실행하거나 새 애플리케이션을 만들 수 없습니다. 그 전에 최신 런타임(아파치 플링크 1.15 및 아파치 제플린 0.10)으로 업그레이드하는 것이 좋습니다. 노트북 업그레이드 방법에 대한 지침은 [여기](#)를 참조하십시오. [Studio 런타임 업그레이드](#)

스튜디오 런타임

아파치 플링크 버전	아파치 제플린 버전	Python 버전	
1.15	0.10	3.8	권장
1.13	0.9	3.8	지원
1.11	0.9	3.7	2024년 11월 5일에 사용 중단됨

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.
- 데이터베이스. AWS Glue 이 데이터베이스에는 데이터 소스, 대상 스키마 및 엔드포인트를 저장하는 테이블이 포함되어 있습니다. 자세한 내용을 알아보려면 [작업 대상 AWS Glue](#) 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 안내서](#)에서 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 스케일링](#)을 참조하세요.

- 로깅 및 모니터링: 애플리케이션 로깅 및 모니터링에 대한 자세한 내용은 [Apache Flink용 Amazon Managed Service for Apache Flink의 로깅 및 모니터링](#)을 참조하세요.
- 애플리케이션은 내결함성을 위해 체크포인트와 세이브포인트를 사용합니다. Studio 노트북에서는 체크포인트와 세이브포인트가 기본적으로 활성화되지 않습니다.

AWS Management Console 또는 를 사용하여 Studio 노트북을 만들 수 AWS CLI있습니다.

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 Flink](#)를 참조하세요.

고급 Studio 노트북 솔루션의 예는 [Apache Flink Studio용 Amazon Managed Service for Apache Flink의 Apache Flink](#)를 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 스트리밍 SQL 쿼리의 더 많은 예는 [예시 및 자습서](#) 다음 및 Apache Flink 설명서의 [쿼리를](#) 참조하십시오.

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	ExecutionEnvironmentStreamExecutionEnvironmentBatchTableEnvironmentStreamTableEnvironment ///를 생성하고 Scala 환경을 제공합니다.

명칭	Class	설명
<code>%flink.pyflink</code>	<code>PyFlinkInterpreter</code>	파이썬 환경을 제공합니다.
<code>%flink.ipyflink</code>	<code>IPyFlinkInterpreter</code>	ipython 환경을 제공합니다.
<code>%flink.ssql</code>	<code>FlinkStreamSqlInterpreter</code>	스트림 SQL 환경을 제공합니다.
<code>%flink.bsql</code>	<code>FlinkBatchSqlInterpreter</code>	일괄 SQL 환경을 제공합니다.

Flink 인터프리터에 대한 자세한 내용은 [Apache Zeppelin용 Flink 인터프리터](#)를 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
<code>senv</code>	<code>StreamExecutionEnvironment</code>
<code>stenv</code>	<code>StreamTableEnvironment for blink planner</code>

다음 변수를 사용하여 Python 테이블 환경 리소스에 액세스합니다.

변수	Resource
<code>s_env</code>	<code>StreamExecutionEnvironment</code>

변수	Resource
st_env	StreamTableEnvironment for blink planner

테이블 환경 사용에 대한 자세한 내용은 Apache Flink [설명서의 개념 및 일반 API](#)를 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.ipynk, %md와(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 양식](#)이 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.sql 인터프리터가 있는 노트 %flink.ipynk을(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.sql(parallelism=32)) 노트에서 배포된 애플리케이션에서는 무시됩니다. 대신 AWS 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 배포](#) 섹션을 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 %timeit와(과) %conda같은 [IPython 매직 명령](#)을 사용하여 [셀 호출/할당](#)을 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 클래스](#)를 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 분석하기](#)를 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 사용

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 사용할 수도 있습니다. 커넥터에 대한 자세한 내용은 Apache Flink 설명서의 [테이블 및 SQL 커넥터](#)를 참조하세요.

기본 커넥터

를 사용하여 Studio AWS Management Console 노트북을 만드는 경우 Apache Flink용 관리 서비스에 는 기본적으로 다음과 `flink-sql-connector-kinesis` 같은 사용자 지정 커넥터가 포함됩니다.

`flink-connector-kafka_2.12 aws-msk-iam-auth` 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.

[CreateApplication](#) API를 사용하여 스튜디오 노트북을 만드는 경우, `flink-sql-connector-flink` 및 `flink-connector-kafka` 커넥터는 기본적으로 포함되지 않습니다. 추가하려면 다음 예와 같이 `CustomArtifactsConfiguration` 데이터 유형에 `MavenReference`(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 [UpdateApplication](#) API 작업을 사용하고 CustomArtifactsConfigurationUpdate 데이터 유형에 MavenReference a로 지정하십시오.

Note

테이블 API의 `flink-sql-connector-kinesis` 커넥터에 대해 `failOnError`을(를) `true`로 설정할 수 있습니다.

종속성 및 사용자 지정 커넥터 추가

를 사용하여 Studio 노트북에 종속성 또는 사용자 지정 커넥터를 AWS Management Console 추가하려면 다음 단계를 따르십시오.

1. 사용자 정의 커넥터 파일을 Amazon S3에 업로드합니다.
2. 에서 스튜디오 노트북을 만들기 위한 사용자 지정 만들기 옵션을 선택합니다. AWS Management Console
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.

[CreateApplication](#) API를 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 [UpdateApplication](#) API 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 함수를 구현하십시오.

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 [pom](#)을 참조하세요.

Note

예제 설정은 기계 학습 블로그 에서 AWS 기계 학습 블로그에서 [Amazon Managed Service for Apache Flink, Amazon Translate, 그리고 Amazon Comprehend](#)를 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. 에서 AWS Management Console Studio 노트북을 만들기 위한 사용자 지정 만들기 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.

[CreateApplication](#) API를 사용하여 새 스튜디오 노트북을 만들 때 UDF JAR을 추가하려면 [CustomArtifactConfiguration](#) 데이터 유형에 JAR 위치를 지정하십시오. 기존 스튜디오 노트북에 UDF JAR을 추가하려면 [UpdateApplication](#) API 작업을 호출하고 데이터 유형에 JAR 위치를 지정합니다. [CustomArtifactsConfigurationUpdate](#) 또는 [를 사용하여](#) 스튜디오 AWS Management Console 노트북에 UDF JAR 파일을 추가할 수 있습니다.

사용자 정의 함수 관련 고려 사항

- Managed Service for Apache Flink Studio는 [Apache Zeppelin 용어](#)를 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 함수를 사용하여 명시적 함수 등록을 수행하면 동일한 노트북의 다른 노트에서도 동일한 [createTemporarySystem함수](#)를 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 함수 이름을 `myNewFuncNameForClass`(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. ssl(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 [pom](#)을 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 Guide](#)의 [Fault Tolerance](#)를 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"
)
```

Studio 런타임 업그레이드

이 섹션에는 Studio 노트북 런타임을 업그레이드하는 방법에 대한 정보가 포함되어 있습니다. 항상 지원되는 최신 Studio 런타임으로 업그레이드하는 것이 좋습니다.

노트북을 새 Studio 런타임으로 업그레이드

Studio를 사용하는 방식에 따라 런타임을 업그레이드하는 단계가 달라집니다. 사용 사례에 맞는 옵션을 선택하세요.

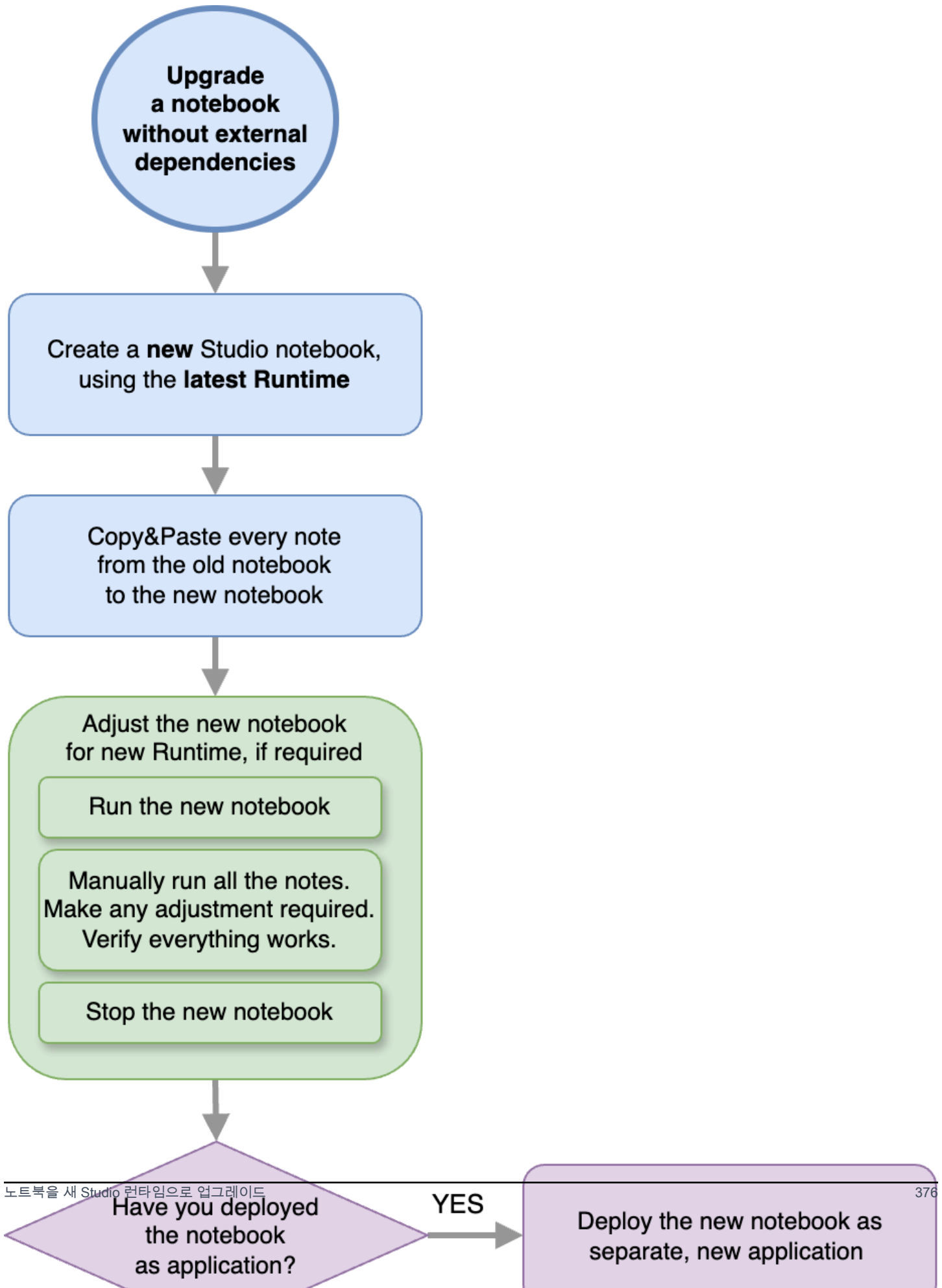
외부 종속성이 없는 SQL 쿼리 또는 Python 코드

외부 종속성 없이 SQL 또는 Python을 사용하는 경우 다음 런타임 업그레이드 프로세스를 사용하십시오. 최신 런타임 버전으로 업그레이드하는 것이 좋습니다. 업그레이드 프로세스는 업그레이드 중인 런타임 버전과 상관없이 동일합니다.

1. 최신 런타임을 사용하여 새 Studio 노트북을 만드십시오.
2. 기존 노트북의 모든 노트 코드를 복사하여 새 노트북에 붙여넣습니다.
3. 새 노트북에서 이전 버전에서 변경된 모든 Apache Flink 기능과 호환되도록 코드를 조정하십시오.
 - 새 노트북을 실행합니다. 노트북을 열고 메모별로 순서대로 실행한 다음 작동하는지 테스트합니다.
 - 코드를 필요에 따라 변경하십시오.

- 새 노트북을 중지하세요.
4. 이전 노트북을 애플리케이션으로 배포한 경우:
 - 새 노트북을 별도의 새 애플리케이션으로 배포하십시오.
 - 기존 애플리케이션을 중지합니다.
 - 스냅샷 없이 새 애플리케이션을 실행합니다.
 5. 이전 노트북이 실행 중이면 중지하십시오. 필요에 따라 대화식 사용을 위해 새 노트북을 시작합니다.

외부 종속성 없이 업그레이드할 수 있는 프로세스 흐름

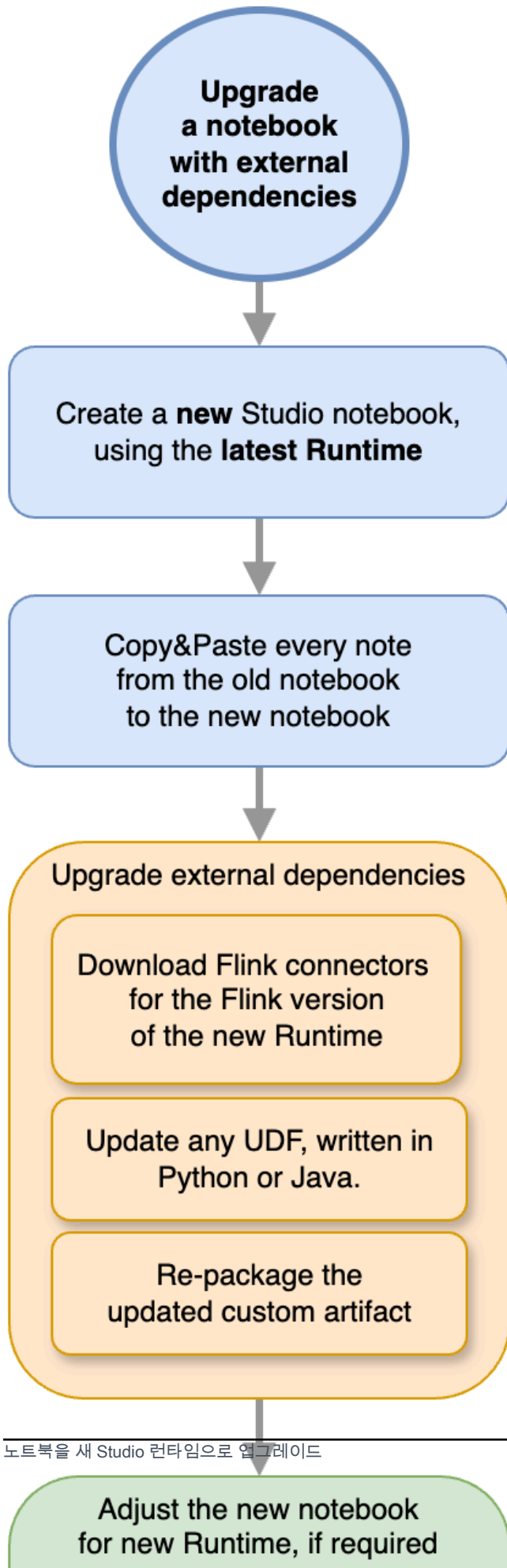


외부 종속성이 있는 SQL 쿼리 또는 Python 코드

SQL 또는 Python을 사용하고 Python 또는 Java로 구현된 사용자 정의 함수와 같은 커넥터나 사용자 정의 아티팩트와 같은 외부 종속성을 사용하는 경우 이 프로세스를 따르십시오. 최신 런타임으로 업그레이드하는 것이 좋습니다. 업그레이드 중인 런타임 버전에 관계없이 프로세스는 동일합니다.

1. 최신 런타임을 사용하여 새 Studio 노트북을 만드십시오.
2. 기존 노트북의 모든 노트 코드를 복사하여 새 노트북에 붙여넣습니다.
3. 외부 종속성 및 사용자 지정 아티팩트를 업데이트하세요.
 - 새 런타임의 Apache Flink 버전과 호환되는 새 커넥터를 찾아보세요. Flink 버전에 맞는 [커넥터를 찾으려면 Apache Flink 설명서의 테이블 및 SQL](#) 커넥터를 참조하십시오.
 - Apache Flink API의 변경 사항 및 사용자 정의 함수에서 사용하는 Python 또는 JAR 종속성과 일치하도록 사용자 정의 함수의 코드를 업데이트합니다. 업데이트된 커스텀 아티팩트를 다시 패키징하세요.
 - 새 노트북에 새 커넥터와 아티팩트를 추가하십시오.
4. 새 노트북에서 코드를 조정하여 이전 버전에서 변경된 모든 Apache Flink 기능과 호환되도록 하십시오.
 - 새 노트북을 실행합니다. 노트북을 열고 메모별로 순서대로 실행한 다음 작동하는지 테스트합니다.
 - 코드를 필요에 따라 변경하십시오.
 - 새 노트북을 중지하세요.
5. 이전 노트북을 애플리케이션으로 배포한 경우:
 - 새 노트북을 별도의 새 애플리케이션으로 배포하십시오.
 - 기존 애플리케이션을 중지합니다.
 - 스냅샷 없이 새 애플리케이션을 실행합니다.
6. 이전 노트북이 실행 중이면 중지하십시오. 필요에 따라 대화식 사용을 위해 새 노트북을 시작합니다.

외부 종속성을 이용한 업그레이드 프로세스 흐름



작업 대상 AWS Glue

Studio 노트북은 데이터 소스 및 싱크에 대한 정보를 저장하고 가져옵니다. AWS Glue 스튜디오 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, 매개변수를 정의하는 AWS Glue 연결에 액세스할 수 있습니다.

Studio 노트북은 테이블 속성을 사용하여 애플리케이션별 데이터를 저장합니다. 자세한 정보는 [테이블 속성](#)을 참조하세요.

Studio [데이터베이스 생성 AWS Glue](#) Notebook에서 사용할 AWS Glue 연결, 데이터베이스 및 테이블을 설정하는 방법에 대한 예는 자습서를 참조하십시오. [스튜디오 노트북 만들기 튜토리얼](#)

테이블 속성

AWS Glue 테이블은 데이터 필드 외에도 테이블 속성을 사용하여 Studio Notebook에 기타 정보를 제공합니다. Apache Flink용 관리형 서비스는 다음과 같은 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 커넥터 및 형식 속성 사용](#): 이러한 속성은 데이터 스트림에 대한 정보를 제공합니다.

AWS Glue 테이블에 속성을 추가하려면 다음과 같이 하세요.

1. <https://console.aws.amazon.com/glue/> 에서 AWS Management Console 로그인하고 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 시간](#)과 같은 스트림 처리 이벤트가 발생한 시간 값을 제공합니다. 애플리케이션 출력에 이러한 값을 포함하려면 AWS Glue 테이블에서 속성을 정의하여 Apache Flink용 관리 서비스 런타임에 이러한 값을 지정된 필드로 내보내도록 지시합니다.

테이블 속성에서 사용하는 키와 값은 다음과 같습니다.

타임스탬프 유형	키	값
처리 시간	매니지드 플링크.proctime	값을 노출하는 AWS Glue 데 사용할 열 이름입니다. 이 열 이름은 기존 테이블 열과 일치하지 않습니다.
이벤트 시간	매니지드 플링크.로우타임	값을 노출하는 데 사용할 열 이름. AWS Glue 이 열 이름은 기존 테이블 열에 해당합니다.
	managed-flink.watermark. <i>column_name</i> .milliseconds	워터마크 간격 (밀리초)

Flink 커넥터 및 형식 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
카프카	format	Kafka 메시지를 역직렬화하고 직렬화하는 데 사용되는 형식 (예: 또는 json csv)
	scan.startup.mode	Kafka 소비자의 시작 모드 (예: 또는 earliest-offset timestamp)
Kinesis	format	Kinesis 데이터 스트림 레코드를 역직렬화하고 직렬화하는 데 사용되는 형식 (예: 또는 json csv)
	aws.region	스트림이 AWS 정의된 지역.

커넥터 유형	키	값
S3 (파일 시스템)	형식	파일을 역직렬화 및 직렬화하는 데 사용되는 형식 (예: 또는 json csv)
	path	Amazon S3 경로, 예s3://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: 지속 가능한 상태의 애플리케이션으로 배포](#)
- [예](#)

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.

- [설치](#)
- [데이터베이스 생성 AWS Glue](#)
- [다음 단계](#)
- [Kinesis Data Streams로 Studio 노트북 생성](#)
- [Amazon MSK로 Studio 노트북 만들기](#)
- [애플리케이션 및 종속 리소스 정리](#)

설치

버전 2 AWS CLI 이상인지 확인하세요. 최신 AWS CLI 버전을 설치하려면 [AWS CLI 버전 2 설치, 업데이트 및 제거](#)를 참조하십시오.

데이터베이스 생성 AWS Glue

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](#)
- [테이블을 AWS Glue 생성하십시오.](#)
- [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 업데이트](#)를 참조하세요. 스트림의 이름을 **ExampleInputStream**(으)로 지정하고 열린 샤드 수를 **1**(으)로 설정합니다.

를 사용하여 스트림 (ExampleInputStream) 을 생성하려면 다음 Amazon Kinesis 명령을 create-stream AWS CLI 사용하십시오. AWS CLI

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

테이블을 AWS Glue 생성하십시오.

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 Glue 체크인해 테이블이 올바르게 생성되었는지 확인할 수 있습니다.

표 생성

1. AWS Management Console 로그인하고 <https://console.aws.amazon.com/glue/> 에서 AWS Glue 콘솔을 엽니다.

2. 아직 AWS Glue 데이터베이스가 없는 경우 왼쪽 탐색 표시줄에서 데이터베이스를 선택합니다. 데이터베이스 추가를 선택합니다. 데이터베이스 추가 창에서 데이터베이스 이름을 **default**(으)로 입력합니다. 생성을 선택합니다.
3. 왼쪽 탐색 모음에서 표를 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
ticker	string
price	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 를 사용할 수 AWS CLI 있습니다.

- [를 사용하여 Studio 노트북을 만드십시오. AWS Management Console](#)
- [를 사용하여 Studio 노트북을 생성합니다. AWS CLI](#)

를 사용하여 Studio 노트북을 만드십시오. AWS Management Console

1. <https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard>에서 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 [Run] 을 선택합니다. 상태가 실행 중으로 표시될 때까지 기다리세요. 노트북이 실행 중일 때는 요금이 부과됩니다.

를 사용하여 Studio 노트북을 생성합니다. AWS CLI

를 사용하여 스튜디오 노트북을 AWS CLI만들려면 다음과 같이 하십시오.

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 kinesisanalyticstv2 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:kinesisanalyticstv2:us-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 kinesisanalyticstv2 start-application --application-arn
arn:aws:kinesisanalyticstv2:us-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 데이터 생성기 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 기다리세요. 스택이 완료되면 콘솔에서 Kinesis-Data-Generator-Cognito-User 스택을 열고 출력 탭을 선택합니다. AWS CloudFormation 출력 값에 대해 나열된 URL을 KinesisDataGeneratorUrl입니다.
8. Amazon Kinesis Data Generator 페이지에서 4단계에서 생성한 보안 인증을 사용하여 로그인합니다.
9. 다음 페이지에서 다음 값을 입력합니다.

리전	us-east-1
스트림/파이퍼호스 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/dashboard>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 애플리케이션 페이지에서 Studio 노트북 탭을 선택합니다. 선택하세요 MyNotebook.
3. MyNotebook 페이지에서 [Apache Zeppelin에서 열기] 를 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 내용을 알아보려면 [Managed Service for Apache Flink 개발자 가이드](#)의 [Apache Flink 대시보드](#)를 참조하세요.

Flink Streaming SQL 쿼리의 더 많은 예는 [Apache Flink 설명서](#)의 [쿼리](#)를 참조하세요.

Amazon MSK로 Studio 노트북 만들기

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

이 자습서는 다음 섹션을 포함하고 있습니다.

- [설치](#)
- [VPC에 NAT 게이트웨이 추가](#)
- [AWS Glue 연결 및 테이블 만들기](#)
- [Amazon MSK로 Studio 노트북 생성](#)
- [Amazon MSK 클러스터로 데이터 전송](#)
- [Studio 노트북을 테스트](#)

설치

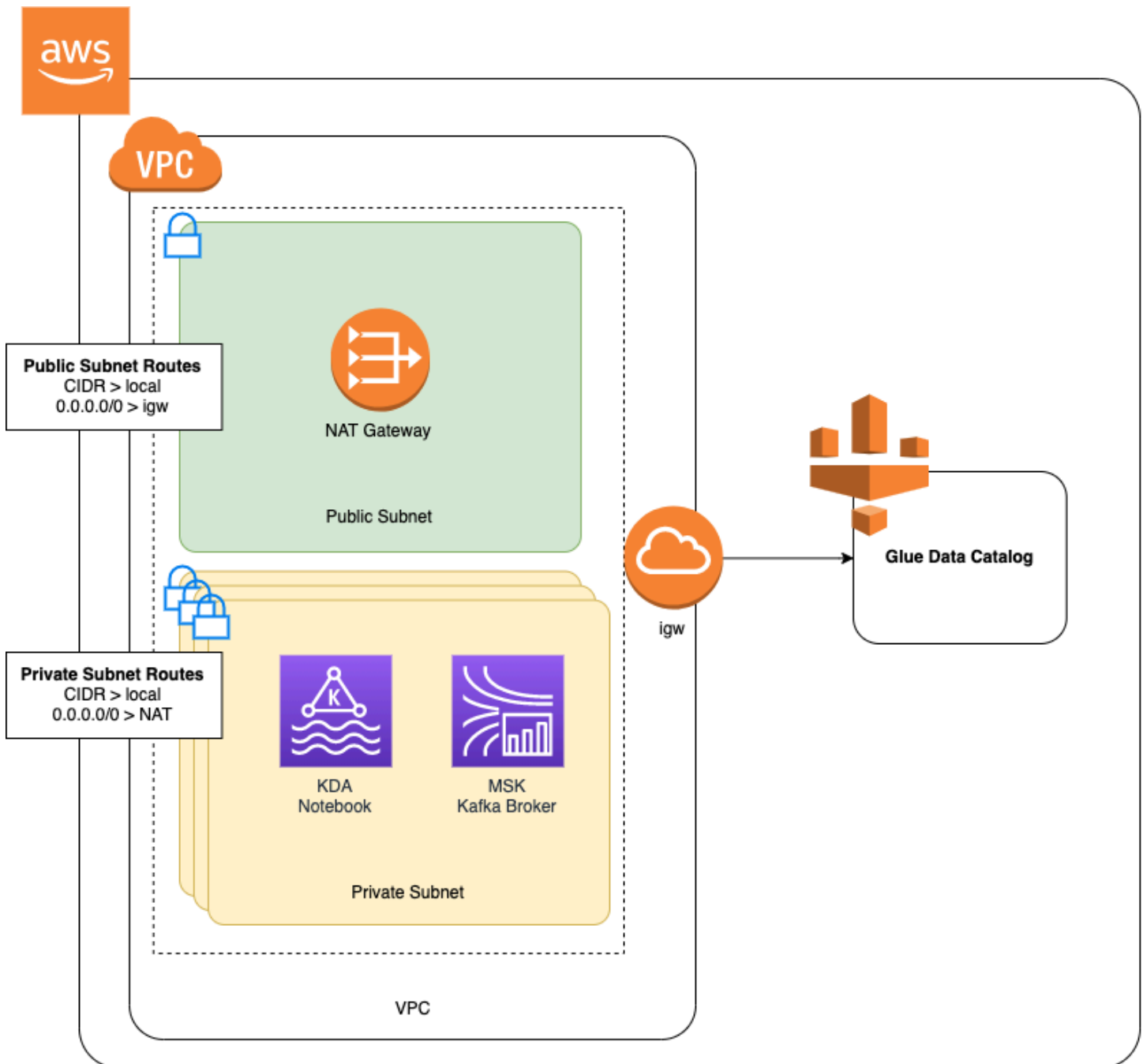
이 자습서에서는 일반 텍스트 액세스를 허용하는 Amazon MSK 클러스터가 필요합니다. Amazon MSK 클러스터를 아직 설정하지 않은 경우, [Amazon MSK 사용 시작하기](#) 자습서를 따라 Amazon VPC, Amazon MSK 클러스터, 주제 및 Amazon EC2 클라이언트 인스턴스를 생성하세요.

자습서에 따라 다음을 수행하십시오:

- [3단계: Amazon MSK 클러스터 생성](#)의 4단계에서 ClientBroker 값을 TLS에서 **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 다음과 같이 하십시오.

1. <https://console.aws.amazon.com/vpc/>에서 Amazon VPC 콘솔을 여세요.
2. 왼쪽 탐색 모음에서 NAT 게이트웨이를 선택합니다.
3. NAT 게이트웨이 페이지에서 NAT 게이트웨이 생성을 선택합니다.
4. NAT 게이트웨이 생성 페이지에서 다음 값을 입력합니다.

이름 - 선택 사항

ZeppelinGateway

서브넷

AWS KafkaTutorialSubnet1

엘라스틱 IP 할당 ID

사용 가능한 엘라스틱 IP를 선택하세요. 사용할 수 있는 엘라스틱 IP가 없는 경우 [Elastic IP 할당] 을 선택한 다음 콘솔에서 생성하는 엘라스틱 IP를 선택합니다.

NAT 게이트웨이 생성을 선택합니다.

5. 왼쪽 탐색 모음에서 경로 표를 선택합니다.
6. [Create Route Table]을 선택합니다.
7. 경로 표 생성 페이지에서 다음 정보를 제공합니다.

- Name tag: **ZeppelinRouteTable**
- VPC: VPC (예: VPC) 를 선택합니다.AWS KafkaTutorial

생성을 선택합니다.

8. 라우팅 테이블 목록에서 선택합니다. ZeppelinRouteTable 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 선택하세요.
11. 라우팅 테이블 페이지에서 ZeppelinRouteTable선택한 상태에서 서브넷 연결 탭을 선택합니다. 서브넷 연결 편집을 선택합니다.
12. 서브넷 연결 편집 페이지에서 AWS KafkaTutorialSubnet2와AWS KafkaTutorialSubnet 3을 선택합니다. 저장을 선택합니다.

AWS Glue 연결 및 테이블 만들기

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. <https://console.aws.amazon.com/glue/> 에서 AWS Management Console 로그인하고 AWS Glue 콘솔을 엽니다.
2. 아직 AWS Glue 데이터베이스가 없는 경우 왼쪽 탐색 표시줄에서 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.

다음을 선택합니다.

5. VPC 페이지에서 다음 값을 입력합니다.
 - VPC의 경우 VPC의 이름 (예: VPC) 을 선택합니다. AWS KafkaTutorial
 - 서브넷의 경우 2를 선택합니다.AWS KafkaTutorialSubnet
 - 보안 그룹의 경우 사용 가능한 모든 그룹을 선택합니다.

다음을 선택합니다.

6. 연결 속성 및 연결 액세스 페이지에서 마침을 선택합니다.

표 생성

Note

다음 단계에 설명된 대로 표를 수동으로 생성하거나 Apache Zeppelin 내의 노트북에서 Managed Service for Apache Flink용 표 커넥터 코드 생성을 사용하여 DDL 문을 통해 표를 생

성할 수 있습니다. 그런 다음 AWS Glue 체크인해 테이블이 올바르게 생성되었는지 확인할 수 있습니다.

1. 왼쪽 탐색 모음에서 표를 선택합니다. 표 페이지에서 표 추가, 표 수동 추가를 선택합니다.
2. 표 속성 설정 페이지에서 표 명칭에 **stock**을 입력합니다. 이전에 만든 데이터베이스를 선택했는지 확인하세요. 다음을 선택합니다.
3. 데이터 스토어 추가 페이지에서 Kafka를 선택합니다. 주제 이름에는 주제 이름을 입력합니다 (예: AWS KafkaTutorialTopic). 연결에서 선택합니다 ZeppelinConnection.
4. 분류 페이지에서 JSON을 선택합니다. 다음을 선택합니다.
5. 스키마 정의 페이지에서 열 추가를 선택하여 열을 추가합니다. 다음 속성을 가진 열을 추가합니다.

열 명칭	데이터 유형
ticker	string
price	double

다음을 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 를 사용하여 애플리케이션을 생성할 수 AWS CLI있습니다.

- [를 사용하여 스튜디오 노트북을 만드십시오. AWS Management Console](#)
- [를 사용하여 Studio 노트북을 생성합니다. AWS CLI](#)

Note

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

를 사용하여 스튜디오 노트북을 만드십시오. AWS Management Console

1. <https://console.aws.amazon.com/managed-flink/home?region=us-east-1#/applications/dashboard>에서 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. MyNotebook페이지에서 구성 탭을 선택합니다. 네트워킹 섹션에서 편집을 선택합니다.
5. 네트워킹 편집 MyNotebook 페이지에서 Amazon MSK 클러스터를 기반으로 하는 VPC 구성을 선택합니다. Amazon MSK 클러스터용 Amazon MSK 클러스터를 선택하세요. 변경 사항 저장을 선택합니다.
6. MyNotebook페이지에서 [Run] 을 선택합니다. 상태가 실행 중으로 표시될 때까지 기다리세요.

를 사용하여 Studio 노트북을 생성합니다. AWS CLI

를 사용하여 스튜디오 노트북을 만들려면 다음과 같이 하십시오. AWS CLI

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 kinesisanalyticstv2 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:kinesisanalyticssus-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 kinesisanalyticssv2 start-application --application-arn
arn:aws:kinesisanalyticssus-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 AWS CLI 컴퓨터에서 를 구성합니다.

```
aws configure
```

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

4. 다음 콘텐츠를 가진 `stock.py`이라는 파일을 생성합니다: 샘플 값을 Amazon MSK 클러스터의 Bootstrap Brokers 문자열로 바꾸고, 주제가 다음과 같지 않은 경우 주제 이름을 업데이트하십시오. `AWSKafkaTutorialTopic`

```
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/dashboard>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Managed Service for Apache Flink 애플리케이션 페이지에서 Studio 노트북 탭을 선택합니다. 선택하세요. MyNotebook
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.sql(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 내용을 알아보려면 [Managed Service for Apache Flink 개발자 가이드](#)의 [Apache Flink 대시보드](#)를 참조하세요.

Flink Streaming SQL 쿼리의 더 많은 예는 [Apache Flink 설명서](#)의 [쿼리](#)를 참조하세요.

애플리케이션 및 종속 리소스 정리

Studio 노트북 삭제

1. Managed Service for Apache Flink 콘솔을 엽니다.
2. 선택하세요 MyNotebook.

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. 옆의 체크박스를 ZeppelinConnection체크하여 선택합니다. 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 모음에서 역할을 선택합니다.
3. 검색 창을 사용하여 ZeppelinRole역할을 검색합니다.
4. ZeppelinRole역할을 선택합니다. 역할 삭제를 선택합니다. 삭제를 확인합니다.

CloudWatch 로그 그룹 삭제

콘솔을 사용하여 애플리케이션을 생성하면 콘솔에서 자동으로 CloudWatch 로그 그룹과 로그 스트림을 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/MyNotebook로그 그룹을 선택합니다.
4. 작업, 로그 그룹 삭제를 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 클러스터를 삭제합니다.

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. 선택하세요 AWS KafkaTutorialCluster. 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. 옆에 있는 체크박스를 ZeppelinClient 선택하여 선택합니다.
4. 인스턴스 상태, 인스턴스 종료를 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 여세요.

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.

- [설치](#)
- [AWS Management Console를 사용하여 지속 가능한 상태의 애플리케이션을 배포하세요.](#)
- [AWS CLI를 사용하여 지속 가능한 상태의 애플리케이션을 배포하세요.](#)

설치

Kinesis Data Streams 또는 Amazon MSK를 사용하여 [스튜디오 노트북 만들기 튜토리얼](#)에 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.

- 아마존 S3 FullAccess
- 아마존 매니지드- 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 콘솔을 통한 배포를 AWS 선택합니다.
 - 그러면 Apache Flink용 관리 서비스 애플리케이션을 만드는 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.

를 사용하여 애플리케이션을 배포하려면 베타 2 정보와 함께 제공된 서비스 모델을 사용하도록 업데이트해야 합니다. AWS CLI AWS CLI 업데이트된 서비스 모델을 사용하는 방법에 대한 자세한 내용은 [설치](#) 섹션을 참조하세요.

다음 예제 코드에서는 새 Studio 노트북을 생성합니다.

```

aws kinesisanalyticv2 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 kinesisanalyticstv2 start-application \  
  --application-name <app-name> \  
  --region us-east-1 \  
  --no-verify-ssl
```

다음 코드는 애플리케이션의 Apache Zeppelin 노트북 페이지의 URL을 반환합니다.

```
aws kinesisanalyticstv2 create-application-presigned-url \  
  --application-name <app-name> \  
  --url-type ZEPPELIN_UI_URL \  
  
  --region us-east-1 \  
  --no-verify-ssl
```

예

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 Flink](#)를 참조하세요.

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

윈도우 쿼리에 대한 자세한 내용은 [Apache Flink 설명서의 윈도우](#)를 참조하세요.

Apache Flink Streaming SQL 쿼리의 더 많은 예는 [Apache Flink 설명서의 쿼리](#)를 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 Connector](#)에서 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 Connector](#)를 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 [쿼리](#)를 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.sql(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.sql

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

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

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



```
`key` > 3
```

```
%flink.sql(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.sql(parallelism=4)
-- no need to define the paragraph type with explicit parallelism (such as
"%flink.sql(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 API](#)를 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 API](#)를 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.sql
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 [StopApplication](#) 작업을 호출하십시오. true 자세한 내용은 [Managed Service for Apache Flink 개발자 안내서의 애플리케이션 실행](#)을 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 deploy-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-beam은 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. [pipdeptree](#)를 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 오른쪽 상단에 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

다음 예제 정책은 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 권한](#)을 참조하세요.

아파치 플링크용 아마존 매니지드 서비스 (DataStreamAPI) 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

- [Apache Flink용 관리 서비스 응용 프로그램의 구성 요소를 검토하십시오.](#)
- [연습을 완료하기 위한 사전 요구 사항을 충족하십시오.](#)
- [1단계: AWS 계정 설정 및 관리자 사용자 생성](#)
- [2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)
- [3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)
- [4단계: AWS 리소스 정리](#)
- [5단계: 리소스를 사용하여 다음 단계 완료](#)

Apache Flink용 관리 서비스 응용 프로그램의 구성 요소를 검토하십시오.

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 내용을 알아보려면 [소스](#) 섹션을 참조하세요.
- 연산자: 애플리케이션은 하나 이상의 연산자를 사용하여 데이터를 처리합니다. 연산자는 데이터를 변환, 강화 또는 집계할 수 있습니다. 자세한 내용을 알아보려면 [DataStream API 연산자](#) 섹션을 참조하세요.

- 싱크: 애플리케이션은 싱크를 사용하여 외부 소스에 데이터를 생성합니다. 싱크 커넥터는 Kinesis 데이터 스트림, Firehose 스트림, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 정보는 [싱크](#)을 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.

연습을 완료하기 위한 사전 요구 사항을 충족하십시오.

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 생성](#) 섹션으로 이동하십시오.

1단계: AWS 계정 설정 및 관리자 사용자 생성

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

가입하세요. AWS 계정

계정이 없는 경우 다음 단계를 완료하여 계정을 만드세요. AWS 계정

가입하려면 AWS 계정

1. <https://portal.aws.amazon.com/billing/signup>을 엽니다.

2. 온라인 지시 사항을 따릅니다.

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

에 AWS 계정가입하면 AWS 계정 루트 사용자a가 생성됩니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업을 수행하는 것](#)입니다.

AWS 가입 절차가 완료된 후 확인 이메일을 보냅니다. 언제든지 <https://aws.amazon.com/>으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

관리자 액세스 권한이 있는 사용자 생성

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

보안을 유지하세요. AWS 계정 루트 사용자

1. 루트 사용자를 선택하고 AWS 계정 이메일 주소를 입력하여 계정 [AWS Management Console](#)소유자로 로그인합니다. 다음 페이지에서 비밀번호를 입력합니다.

루트 사용자를 사용하여 로그인하는 데 도움이 필요하면 AWS 로그인 User Guide의 [루트 사용자 로 로그인](#)을 참조하십시오.

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

지침은 IAM [사용 설명서의 AWS 계정 루트 사용자 \(콘솔\)에 대한 가상 MFA 디바이스 활성화를 참조](#)하십시오.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

지침은 AWS IAM Identity Center 사용 설명서의 [AWS IAM Identity Center설정을 참조](#)하세요.

2. IAM Identity Center에서 사용자에게 관리 액세스 권한을 부여합니다.

를 ID 소스로 사용하는 방법에 대한 자습서는 사용 [설명서의 기본값으로 IAM Identity Center 디렉터리사용자 액세스 구성](#)을 참조하십시오. IAM Identity Center 디렉터리 AWS IAM Identity Center

관리 액세스 권한이 있는 사용자로 로그인

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

IAM IDentity Center 사용자를 사용하여 [로그인하는 데 도움이 필요하다면 사용 설명서의 AWS 액세스 포털에 로그인](#)을 참조하십시오.AWS 로그인

추가 사용자에게 액세스 권한 할당

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

지침은AWS IAM IDentity Center 사용 설명서의 [Create a permission set](#)를 참조하세요.

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

지침은AWS IAM IDentity Center 사용 설명서의 [Add groups](#)를 참조하세요.

프로그래밍 방식 액세스 권한 부여

사용자가 AWS 외부 사용자와 상호 작용하려는 경우 프로그래밍 방식의 액세스가 필요합니다. AWS Management Console프로그래밍 방식의 액세스 권한을 부여하는 방법은 액세스하는 사용자 유형에 따라 다릅니다. AWS

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID (IAM IDentity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에서 명할 수 있습니다. AWS	<p>사용하고자 하는 인터페이스에 대한 지침을 따릅니다.</p> <ul style="list-style-type: none"> • AWS CLI에 대한 내용은 사용 설명서의 AWS CLI사용을 AWS IAM IDentity Center위 한 구성을 참조하십시오.AWS S Command Line Interface • AWS SDK, 도구 및 AWS API의 경우 AWS SDK 및 도

<p>프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?</p>	<p>To</p>	<p>액세스 권한을 부여하는 사용자</p>
		<p>구 참조 안내서의 IAM ID 센터 인증을 참조하십시오.</p>
<p>IAM</p>	<p>임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 방식 요청에 서명할 수 있습니다. AWS</p>	<p>IAM 사용 설명서의 AWS 리소스와 함께 임시 자격 증명 사용의 지침을 따르십시오.</p>
<p>IAM</p>	<p>(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 API에 대한 프로그래밍 요청에 서명할 수 있습니다. AWS</p>	<p>사용하고자 하는 인터페이스에 대한 지침을 따릅니다.</p> <ul style="list-style-type: none"> • 에 대한 내용은 사용 설명서의 IAM 사용자 자격 증명을 사용한 인증을 참조하십시오. AWS CLI AWS Command Line Interface • AWS SDK 및 도구의 경우 SDK 및 도구 참조 안내서의 장기 자격 증명을 사용한 인증을 참조하십시오. AWS • AWS API의 경우 IAM 사용 설명서의 IAM 사용자의 액세스 키 관리를 참조하십시오.

다음 단계

[2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)

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

이 단계에서는 Apache Flink용 관리 서비스와 함께 사용하도록 AWS CLI 를 다운로드하고 구성합니다.

Note

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

Note

이미 AWS CLI 설치되어 있는 경우 최신 기능을 사용하려면 업그레이드해야 할 수 있습니다. 자세한 내용을 알아보려면 AWS Command Line Interface 사용자 가이드에서 [AWS Command Line Interface설치](#)를 참조하세요. 이 버전을 확인하려면 다음 명령을 실행합니다. AWS CLI

```
aws --version
```

이 자습서의 연습에는 다음 AWS CLI 버전 이상이 필요합니다.

```
aws-cli/1.16.63
```

설정하려면 AWS CLI

1. AWS CLI를 다운로드하고 구성합니다. 관련 지침은 AWS Command Line Interface 사용 설명서에서 다음 토픽을 참조하세요.
 - [AWS Command Line Interface설치](#)
 - [AWS CLI구성](#)
2. AWS CLI config파일에 관리자 사용자의 이름이 지정된 프로필을 추가합니다. 이 프로필은 AWS CLI 명령을 실행할 때 사용합니다. 프로파일 명명에 대한 자세한 설명은 AWS Command Line Interface 사용자 가이드의 [프로파일 명명](#)을 참조하십시오.

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

사용 가능한 AWS 지역 목록은 의 [지역 및 엔드포인트](#)를 참조하십시오. Amazon Web Services 일반 참조

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
```

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

다음 단계

[3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행](#)

3단계: Apache Flink용 관리 서비스 응용 프로그램 만들기 및 실행

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

이 섹션은 다음 주제를 포함합니다:

- [두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.](#)
- [입력 스트림에 샘플 레코드를 작성합니다.](#)
- [Apache Flink 스트리밍 Java 코드를 다운로드하여 검토하십시오.](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [다음 단계](#)

두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(ExampleInputStream 및 ExampleOutputStream)을 생성하세요. 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 업데이트](#)를 참조하세요.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림 (ExampleInputStream) 을 생성하려면 다음 Amazon Kinesis 명령을 create-stream AWS CLI 사용하십시오.

```
$ aws kinesys 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 kinesys create-stream \  
--stream-name ExampleOutputStream \  
--shard-count 1 \  
--region us-west-2 \  
--profile adminuser
```

입력 스트림에 샘플 레코드를 작성합니다.

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 코드를 다운로드하여 검토하십시오.

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-managed-service-for-apache-flink-examples.git
```

2. amazon-managed-service-for-apache-flink-examples/tree/main/java/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 Kinesis 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 내용을 알아보려면 [런타임 속성](#) 섹션을 참조하세요.

애플리케이션 코드를 컴파일하십시오.

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 내용을 알아보려면 [연습을 완료하기 위한 사전 요구 사항을 충족하십시오](#) 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 도구를 사용합니다. `pom.xml` 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

```
mvn package -Dflink.version=1.19.1
```

- 귀하의 개발 환경 사용. 자세한 내용을 알아보려면 해당 개발 환경 설명서를 참조하십시오.

Note

제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. 를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형 (JAR 또는 ZIP) 을 지정합니다. AWS CLI

2. 컴파일하는 동안 오류가 발생하면 JAVA_HOME 환경 변수가 올바르게 설정되어 있는지 확인하십시오.

애플리케이션이 성공적으로 컴파일되면 다음 파일이 생성됩니다:

```
target/amazon-msf-java-stream-app-1.0.jar
```

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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 amazon-msf-java-stream-app-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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

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. 를 사용하여 애플리케이션을 생성할 때는 이러한 리소스를 별도로 생성합니다. AWS CLI

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/flink>에서 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 아파치 플링크 버전 1.19로 그대로 둡니다.
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 리전을 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/amazon-msf-java-stream-app-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 **amazon-msf-java-stream-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. 속성에서 그룹 ID에 **ProducerConfigProperties**를 입력합니다.
5. 다음 애플리케이션 속성 및 값을 입력합니다:

그룹 ID	키	값
FlinkApplicationPr operties	flink.inputstream. initpos	LATEST
FlinkApplicationPr operties	aws.region	us-west-2
FlinkApplicationPr operties	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 를 사용하여 Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다. AWS CLI Apache Flink용 관리 서비스는 `kinesisanalyticsv2` AWS CLI 명령을 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 이와 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 연결](#)을 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. 역할 생성을 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.

- 요약 페이지에서 권한 탭을 선택합니다.
- 정책 연결을 선택합니다.
- 검색 상자에 **AKReadSourceStreamWriteSinkStream**(이전 섹션에서 생성한 정책)을 입력합니다.
- 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.

Apache Flink용 관리형 서비스 애플리케이션 생성

- 다음 JSON 코드를 `create_request.json`이라는 파일에 저장합니다. 샘플 역할을 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_19",
  "ServiceExecutionRole": "arn:aws:iam::012345678901:role/MF-stream-rw-role",
  "ApplicationConfiguration": {
    "ApplicationCodeConfiguration": {
      "CodeContent": {
        "S3ContentLocation": {
          "BucketARN": "arn:aws:s3:::ka-app-code-username",
          "FileKey": "amazon-msf-java-stream-app-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"
        }
      }
    ]
  }
}

```

- 위의 요청과 함께 [CreateApplication](#) 작업을 실행하여 애플리케이션을 생성합니다.

```
aws kinesisanalyticstv2 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

- 다음 JSON 코드를 `start_request.json`이라는 파일에 저장합니다.

```
{
  "ApplicationName": "test",
```

```

    "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
        "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
      }
    }
  }
}

```

- 위의 요청과 함께 [StartApplication](#) 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

- 다음 JSON 코드를 `stop_request.json`이라는 파일에 저장합니다.

```

{
  "ApplicationName": "test"
}

```

- 다음 요청과 함께 [StopApplication](#) 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticstv2 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 [the 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.json`이라는 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 kinesisanalyticstv2 update-application --cli-input-json file://
update_properties_request.json
```

애플리케이션 코드 업데이트

새 버전의 코드 패키지로 애플리케이션 코드를 업데이트해야 하는 경우 [UpdateApplication](#) AWS CLI 작업을 사용합니다.

Note

파일 이름이 같은 새 버전의 애플리케이션 코드를 로드하려면 새 객체 버전을 지정해야 합니다. Amazon S3 객체 버전 사용에 대한 자세한 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 참조하세요.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름, 새 객체 버전을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사 (*<username>*)를 [the section called “두 개의 Amazon Kinesis 데이터 스트림을 생성합니다.”](#) 섹션에서 선택한 접미사로 업데이트합니다.

```
{
  "ApplicationName": "test",
  "CurrentApplicationVersionId": 1,
  "ApplicationConfigurationUpdate": {
    "ApplicationCodeConfigurationUpdate": {
      "CodeContentUpdate": {
        "S3ContentLocationUpdate": {
          "BucketARNUpdate": "arn:aws:s3:::ka-app-code-username",
          "FileKeyUpdate": "amazon-msf-java-stream-app-1.0.jar",
          "ObjectVersionUpdate": "SAMPLEUehYngP87ex1nzYIGYgfhyvpvDU"
        }
      }
    }
  }
}
```

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:

- [Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)
- [다음 단계](#)

Apache Flink용 관리 서비스 애플리케이션을 삭제하십시오.

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/flink>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

다음 단계

[5단계: 리소스를 사용하여 다음 단계 완료](#)

5단계: 리소스를 사용하여 다음 단계 완료

이제 Managed Service for Apache Flink 애플리케이션을 만들고 실행했으므로 고급 Managed Service for Apache Flink 솔루션에 대해서는 다음 리소스를 참조하세요.

- Amazon [Kinesis용 AWS 스트리밍 데이터 솔루션: Amazon Kinesis용 AWS 스트리밍 데이터 솔루션](#)은 스트리밍 데이터를 쉽게 캡처, 저장, 처리 및 전송하는 데 필요한 서비스를 자동으로 구성합니다 AWS . 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

- [AWS Amazon 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.

아파치 플링크용 아마존 매니지드 서비스 시작하기 (테이블 API)

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

- [아파치 플링크용 매니지드 서비스 애플리케이션의 구성 요소](#)
- [사전 조건](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [AWS 리소스 정리](#)
- [다음 단계](#)

아파치 플링크용 매니지드 서비스 애플리케이션의 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 내용을 알아보려면 [테이블 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, Firehose 스트림, Amazon MSK 주제, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 정보는 [테이블 API 싱크](#)를 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 \(DataStreamAPI\) 시작하기](#)의 첫 두 단계를 완료하세요.

- [1단계: AWS 계정 설정 및 관리자 사용자 생성](#)
- [2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)

시작하려면 [애플리케이션 만들기](#) 섹션을 참조하세요.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

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](#)
- [입력 스트림에 샘플러 레코드를 기록하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 다운로드하고 검사하세요.](#)
- [애플리케이션 코드를 컴파일하십시오.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [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 아마존 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.aws-kafka-tutorial-cluster.t79r6y.c4.kafka.us-west-2.amazonaws.com:9094,b-1.aws-kafka-tutorial-cluster.t79r6y.c4.kafka.us-west-2.amazonaws.com:9094,b-3.aws-kafka-tutorial-cluster.t79r6y.c4.kafka.us-west-2.amazonaws.com:9094"
}
```

- 자습서의 단계를 따를 때는 코드, 명령 및 콘솔 항목에서 선택한 AWS 지역을 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?](#) 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.

입력 스트림에 샘플러 레코드를 기록하십시오.

이 섹션에서는 Python 스크립트를 사용하여 애플리케이션에서 처리할 샘플 레코드를 Amazon MSK 주제에 씁니다.

1. [Amazon MSK 사용 시작하기](#) 자습서의 [4단계: 클라이언트 머신 생성](#)에서 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.

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

# BROKERS = "b-1.stocks.8e6izk.c12.kafka.us-east-1.amazonaws.com:9092,b-2.stocks.8e6izk.c12.kafka.us-east-1.amazonaws.com:9092"
BROKERS = "localhost:9092"
producer = KafkaProducer(
    bootstrap_servers=BROKERS,
    value_serializer=lambda v: json.dumps(v).encode('utf-8'),
    key_serializer=str.encode,
    retry_backoff_ms=500,
    request_timeout_ms=20000,
    security_protocol='PLAINTEXT')

def getReferrer():
    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 =getReferrer()
    # print(data)
    try:
        future = producer.send("stocktopic", value=data,key=data['ticker'])
        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())

```

- 이 자습서의 뒷부분에서 stock.py 스크립트를 실행하여 애플리케이션으로 데이터를 전송합니다.

```
$ python3 stock.py
```

아파치 플링크 스트리밍 자바 코드를 다운로드하고 검사하세요.

이 예제의 Java 애플리케이션 코드는 에서 제공됩니다. GitHub

Java 애플리케이션 코드를 다운로드하려면

- 다음 명령을 사용하여 원격 리포지토리를 복제합니다.

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

- 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 `FlinkKafkaConsumer`를 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 재 컴파일하지 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 내용을 알아보려면 [런타임 속성](#) 섹션을 참조하세요.

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 내용을 알아보려면 [맞춤 MSK 구성](#)을 참조하세요.

애플리케이션 코드를 컴파일하십시오.

이 섹션에서는 Apache Maven 컴파일러를 사용하여 애플리케이션용 Java 코드를 생성합니다. Apache Maven 및 Java Development Kit(JDK) 설치에 대한 자세한 내용을 알아보려면 [연습을 완료하기 위한 사전 요구 사항을 충족하십시오](#). 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 도구를 사용합니다. pom.xml 파일이 있는 디렉터리에서 다음 명령을 실행하여 JAR 파일을 생성합니다:

```
mvn package -Dflink.version=1.19.1
```

- 귀하의 개발 환경 사용. 자세한 내용을 알아보려면 해당 개발 환경 설명서를 참조하십시오.

Note

제공된 소스 코드는 Java 11의 라이브러리를 사용합니다.

패키지를 JAR 파일로 업로드하거나 패키지를 압축하여 ZIP 파일로 업로드할 수 있습니다. 를 사용하여 애플리케이션을 생성하는 경우 코드 콘텐츠 유형 (JAR 또는 ZIP) 을 지정합니다. AWS CLI

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
```

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink/>에서 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.
 - 버전을 아파치 플링크 버전 1.19.1로 유지하세요.
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 리전을 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.

애플리케이션을 구성하려면

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	<i>Your Amazon MSK cluster's Bootstrap Brokers List</i>
FlinkApplicationProperties	s3Path	ka-app-code- <i><username></i>
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

애플리케이션을 실행합니다

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
```

애플리케이션을 중지합니다.

애플리케이션을 중지하려면 MyApplication페이지에서 중지를 선택합니다. 작업을 확인합니다.

다음 단계

[AWS 리소스 정리](#)

AWS 리소스 정리

이 섹션에는 시작하기 (테이블 API) 자습서에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다.

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Amazon MSK 클러스터를 삭제합니다.](#)
- [VPC 삭제](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)
- [다음 단계](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

애플리케이션을 삭제하려면 다음 절차를 따르세요.

애플리케이션을 삭제하려면

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확정합니다.

Amazon MSK 클러스터를 삭제합니다.

Amazon MSK 클러스터를 삭제하려면 [Apache Kafka용 Amazon Managed Streaming 개발자 가이드의 8단계: Amazon MSK 클러스터 삭제](#)를 따르십시오.

VPC 삭제

Amazon VPC를 삭제하려면 다음을 수행합니다.

- Amazon VPC 콘솔을 엽니다.
- VPC를 선택합니다.

- 작업에서 VPC 삭제를 선택합니다.

Amazon S3 객체 및 버킷을 삭제합니다.

다음 절차에 따라 S3 객체 및 버킷을 삭제합니다.

S3 객체 및 버킷을 삭제하려면

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- <username>버킷을 선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. 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. 탐색 바에서 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 \(DataStreamAPI\) 시작하기의 5단계: 리소스를 사용하여 다음 단계 완료](#) 섹션을 참조하십시오.

Python용 Apache Flink용 아마존 매니지드 서비스 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

- [Pyflink 시작하기 - Apache를 위한 Python 인터프리터 | Amazon 웹 서비스](#)
- [Apache Flink 애플리케이션용 관리형 서비스의 구성 요소](#)
- [사전 조건](#)
- [Python용 Apache Flink용 매니지드 서비스 생성 및 실행 애플리케이션](#)
- [AWS 리소스 정리](#)

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](#)

Apache Flink 애플리케이션용 관리형 서비스의 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 내용을 알아보려면 [테이블 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, Firehose Firehose 스트림, Amazon MSK 주제, Amazon S3 버킷 등에 데이터를 씁니다. 자세한 정보는 [테이블 API 싱크](#)을 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 \(DataStreamAPI\) 시작하기](#)의 첫 두 단계를 완료하세요.

- [1단계: AWS 계정 설정 및 관리자 사용자 생성](#)
- [2단계: \(\) 설정 AWS Command Line Interface AWS CLI](#)

시작하려면 [애플리케이션 만들기](#) 섹션을 참조하세요.

Python용 Apache Flink용 매니지드 서비스 생성 및 실행 애플리케이션

이 예에서는 Kinesis 스트림을 소스 및 싱크로 사용하여 Managed Service for Apache Flink Python 애플리케이션을 생성합니다.

이 섹션은 다음 주제를 포함합니다.

- [종속 리소스 생성](#)
- [입력 스트림에 샘플 레코드를 작성합니다.](#)
- [아파치 플링크 스트리밍 Python 코드 생성 및 검토](#)
- [Python 앱에 제3자 종속성 추가](#)
- [아파치 플링크 스트리밍 Python 코드 업로드](#)
- [Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다.](#)
- [다음 단계](#)

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>*)

두 개의 Kinesis 스트림을 생성합니다.

이 연습을 위해 Managed Service for Apache Flink 애플리케이션을 생성하기 전에 두 개의 Kinesis 데이터 스트림(ExampleInputStream 및 ExampleOutputStream)을 생성하세요. 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 업데이트](#)를 참조하세요.

데이터 스트림 (AWS CLI)을 생성하려면

1. 첫 번째 스트림 (ExampleInputStream) 을 생성하려면 다음 Amazon Kinesis 명령을 create-stream AWS CLI 사용하십시오.

```
$ 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?](#) 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.

입력 스트림에 샘플 레코드를 작성합니다.

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. 계정 자격 증명과 기본 지역을 사용하도록 구성해야 합니다. AWS CLI 를 AWS CLI구성하려면 다음을 입력합니다.

```
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 됩니다.

아파치 플링크 스트리밍 Python 코드 생성 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 설치](#)를 참조하세요.
2. 다음 명령을 사용하여 원격 리포지토리를 복제합니다:

```
git clone https://github.com/aws-samples/amazon-kinesis-data-analytics-examples.git
```

- 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 폴링크](#) 커넥터를 사용합니다.

Python 앱에 제3자 종속성 추가

제3자 python 패키지(예: [boto3](#))를 사용할 때는 해당 패키지의 전이 종속성과 이러한 종속성을 목적지로 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-4.2.0-1.18.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 방법](#)에서 찾을 수 있습니다.

아파치 플링크 스트리밍 Python 코드 업로드

이 섹션에서는 Amazon S3 버킷을 만들고 애플리케이션 코드를 업로드합니다.

콘솔을 사용하여 애플리케이션 코드를 업로드하려면:

1. 선호하는 압축 애플리케이션을 사용하여 `getting-started.py` 및 [Flink](#) SQL 커넥터 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 [Flink SQL 커넥터](#) 파일을 압축합니다.

Note

Finder(macOS) 또는 Windows Explorer(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 getting-started.py 및 [Flink SQL 커넥터](#) 파일을 압축하십시오. 아카이브 이름은 myapp.zip라고 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.

Apache Flink용 관리 서비스 애플리케이션을 만들고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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 아파치 플링크 버전 1.18로 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 리전을 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.

애플리케이션을 구성하려면

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-analytcs-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. 그룹 ID에 **flink.sql.connector.kinesis.options**를 입력합니다. 이 특수 속성 그룹은 애플리케이션에 코드 리소스를 찾을 수 있는 위치를 알려줍니다. 자세한 내용을 알아보려면 [코드 파일 지정](#) 섹션을 참조하세요.
9. 다음을 입력합니다:

그룹 ID	키	값
<code>kinesis.analytics.flink.run.options</code>	<code>python</code>	<code>getting-started.py</code>
<code>kinesis.analytics.flink.run.options</code>	<code>jarfile</code>	<code>flink-sql-connector-kinesis-4.2.0-1.18.jar</code>

- 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
- CloudWatch 로깅하려면 활성화 확인란을 선택합니다.
- 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 정책을 편집하십시오.

IAM 정책을 편집하여 Amazon S3 버킷에 액세스할 수 있는 권한을 추가합니다.

IAM 정책을 편집하여 S3 버킷 권한을 추가하려면

- <https://console.aws.amazon.com/iam/>에서 IAM 콘솔을 여세요.
- 정책을 선택하세요. 이전 섹션에서 콘솔이 생성한 `kinesis-analytics-service-MyApplication-us-west-2` 정책을 선택합니다.
- 요약 페이지에서 정책 편집을 선택합니다. JSON 탭을 선택합니다.
- 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"
    }
  ]
}

```

애플리케이션을 실행합니다

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 중지를 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.

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

애플리케이션을 삭제하려면 다음 절차를 따르세요.

애플리케이션을 삭제하려면

1. <https://console.aws.amazon.com/kinesis>에서 Kinesis 콘솔을 엽니다.
2. Apache Flink용 관리 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확정합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/flink>에서 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 버킷을 삭제합니다.

다음 절차에 따라 S3 객체 및 버킷을 삭제합니다.

S3 객체 및 버킷을 삭제하려면

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- <username>버킷을 선택합니다.
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. 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. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

시작하기 (스칼라)

Note

버전 1.15부터 플링크는 스칼라가 없습니다. 이제 애플리케이션은 모든 Scala 버전에서 Java API를 사용할 수 있습니다. Flink는 여전히 내부적으로 몇 가지 주요 구성 요소에서 Scala를 사용하고 있지만 사용자 코드 클래스 로더에 Scala를 노출하지는 않습니다. 따라서 JAR 아카이브에 Scala 종속 항목을 추가해야 합니다.

Flink 1.15의 Scala 변경 사항에 대한 자세한 내용은 [Scala Free in One Fifteen](#)을 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](#)
- [입력 스트림에 샘플 레코드를 작성합니다.](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [애플리케이션 코드 컴파일 및 업로드](#)
- [애플리케이션 생성 및 실행 \(콘솔\)](#)
- [애플리케이션 생성 및 실행\(CLI\)](#)
- [AWS 리소스를 정리하세요.](#)

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 create-stream 명령을 사용하십시오.

```
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?](#) 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.

입력 스트림에 샘플 레코드를 작성합니다.

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. 계정 자격 증명과 기본 지역을 사용하여 구성해야 합니다. AWS CLI 를 AWS CLI 구성하려면 다음을 입력합니다.

```
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 됩니다.

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 설치](#)를 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 Kinesis 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
}
```

- 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 속성](#)을 참조하세요.

애플리케이션 코드 컴파일 및 업로드

이 섹션에서는 사용자가 [종속 리소스 생성](#) 섹션에서 생성한 Amazon S3 버킷에 사용자의 애플리케이션 코드를 컴파일 및 업로드합니다.

애플리케이션 코드 컴파일

이 섹션에서는 [SBT](#) 구축 도구를 사용하여 애플리케이션용 Scala 코드를 구축하는 도구를 구축합니다. SBT를 설치하려면 [cs 설정으로 sbt 설치](#)를 참조하십시오. 또한 Java Development Kit(JDK)를 설치해야 합니다. [연습 완료를 위한 사전 조건](#)을 참조하세요.

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/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 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 `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 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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.
 - 버전을 아파치 플링크 버전 1.19.1로 유지합니다.
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 리전을 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.

애플리케이션을 구성하려면

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 **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.

- 속성에서 그룹 추가를 다시 선택합니다.
- 다음을 입력합니다:

그룹 ID	키	값
ProducerConfigProperties	output.stream.name	ExampleOutputStream
ProducerConfigProperties	aws.region	us-west-2

- 모니터링에서 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
- CloudWatch 로깅하려면 활성화 확인란을 선택합니다.
- 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 정책을 편집하십시오.

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/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"
}
]
}

```

애플리케이션을 실행합니다

애플리케이션을 실행하고 Apache Flink 대시보드를 연 다음 원하는 Flink 작업을 선택하면 Flink 작업 그래프를 볼 수 있습니다.

애플리케이션 중지

애플리케이션을 중지하려면 MyApplication페이지에서 중지를 선택합니다. 작업을 확인합니다.

애플리케이션 생성 및 실행(CLI)

이 섹션에서는 `aws`를 사용하여 Apache Flink용 관리 서비스 응용 프로그램을 만들고 실행합니다. AWS Command Line Interface `kinesisanalyticsv2` AWS CLI 명령을 사용하여 아파치 플링크용 관리 서비스 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 설명과 싱크 스트림의 쓰기 작업에 대한 권한을 부여하는 설명 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.

username을 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 연결](#)을 참조하십시오.

IAM 정책 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. 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 생성](#)에서 만든 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.json`이라는 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 사용자 계정 ID로 바꿉니다.

```
{
  "ApplicationName": "getting_started",
  "ApplicationDescription": "Scala getting started application",
  "RuntimeEnvironment": "FLINK-1_19",
  "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 [CreateApplication](#) 요청과 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](#) 액션을 사용하여 애플리케이션을 시작합니다.

애플리케이션을 시작하려면

1. 다음 JSON 코드를 `start_request.json`이라는 파일에 저장합니다.

```
{
```

```

    "ApplicationName": "getting_started",
    "RunConfiguration": {
      "ApplicationRestoreConfiguration": {
        "ApplicationRestoreType": "RESTORE_FROM_LATEST_SNAPSHOT"
      }
    }
  }
}

```

- 위의 요청과 함께 StartApplication 작업을 실행하여 애플리케이션을 시작합니다.

```
aws kinesisanalyticstv2 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.

애플리케이션을 중지하십시오.

이 섹션에서는 [StopApplication](#) 작업을 사용하여 애플리케이션을 중지합니다.

애플리케이션을 중지하려면

- 다음 JSON 코드를 stop_request.json이라는 파일에 저장합니다.

```

{
  "ApplicationName": "s3_sink"
}

```

- 위의 요청과 함께 StopApplication 작업을 실행하여 애플리케이션을 중지합니다.

```
aws kinesisanalyticstv2 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 참조](#)하십시오.

환경 속성 업데이트

이 섹션에서는 애플리케이션 코드를 다시 컴파일하지 않고도 [UpdateApplication](#) 액션을 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.json`이라는 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 kinesisanalyticstv2 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 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 참조하세요.

를 사용하려면 Amazon S3 버킷에서 이전 코드 패키지를 삭제하고 새 버전을 업로드한 다음 동일한 Amazon S3 버킷과 객체 이름, 새 객체 버전을 지정하여 UpdateApplication 호출합니다. AWS CLI 애플리케이션이 새 코드 패키지로 다시 시작됩니다.

다음 예 UpdateApplication 작업 요청은 애플리케이션 코드를 다시 로드하고 애플리케이션을 다시 시작합니다. CurrentApplicationVersionId를 현재 애플리케이션 버전으로 업데이트하세요. ListApplications 또는 DescribeApplication 작업을 사용하여 현재 애플리케이션 버전을 확인할 수 있습니다. 버킷 명칭 접미사(<username>)를 [종속 리소스 생성](#) 섹션에서 선택한 접미사로 업데이트합니다.

```

{{
  "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 리소스를 정리하세요.

이 섹션에는 텀블링 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션 삭제](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션 삭제

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3/>에서 S3 콘솔을 엽니다.
2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 로그 그룹을 선택합니다.
4. 로그 그룹 삭제를 선택한 다음 삭제를 확인합니다.

Apache Beam을 사용하여 Managed Service for Apache Flink 애플리케이션 생성

Note

아파치 플링크 버전 1.19에서는 아파치 빔이 지원되지 않습니다. 2024년 6월 27일 현재, 플링크 1.18과 호환되는 아파치 플링크 러너는 없습니다. 자세한 내용은 Apache Beam 설명서의 [Flink 버전 호환성](#)을 참조하십시오. >

[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 러너](#)를 사용하여 Beam 파이프라인을 실행합니다.

Managed Service for Apache Flink 애플리케이션에서 Apache Beam을 사용하는 방법에 대한 자습서는 [아파치 플링크용 매니지드 서비스와 CloudFormation 함께 사용](#) 섹션을 참조하세요.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Managed Service for Apache Flink와 함께 Apache Beam 사용](#)
- [빔 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.

범 기능

Managed Service for Apache Flink는 Apache Flink 러너와 동일한 Apache Beam 기능을 지원합니다. Apache Flink 러너에서 지원되는 기능에 대한 자세한 내용을 알아보려면 [Beam 병렬성 매트릭스](#)를 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 Beam](#)을 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 내용을 알아보려면 [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](#)
- [입력 스트림에 샘플 레코드 쓰기](#)
- [애플리케이션 코드 다운로드 및 검토](#)
- [애플리케이션 코드를 컴파일합니다.](#)
- [아파치 플링크 스트리밍 자바 코드를 업로드하세요.](#)
- [Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.](#)
- [리소스를 정리하세요. AWS](#)
- [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 하나요?](#) 로그인 명칭(예: **ka-app-code-*<username>***)을 추가하여 Amazon S3 버킷에 전역적으로 고유한 명칭을 지정합니다.

입력 스트림에 샘플 레코드 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.py이라는 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 됩니다.

애플리케이션 코드 다운로드 및 검토

이 예제의 Java 응용 프로그램 코드는 에서 제공됩니다 GitHub. 애플리케이션 코드를 다운로드하려면 다음을 수행하세요.

1. 아직 설치하지 않았다면 Git 클라이언트를 설치합니다. 자세한 정보는 [Git 설치](#)를 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 [ParDo](#)Beam을 사용하여 라는 사용자 지정 변환 함수를 호출하여 들어오는 레코드를 처리합니다. PingPongFn

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.xml에 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 코드를 컴파일합니다.

애플리케이션을 컴파일하려면 다음을 수행하세요.

1. 아직 Java 및 Maven을 설치하지 않았으면 설치합니다. 자세한 정보는 [시작하기 \(API\) `DataStream`](#) 자습서의 [사전 조건](#) 섹션을 참조하세요.
2. 다음 명령을 사용하여 애플리케이션을 컴파일합니다.

```
mvn package -Dflink.version=1.15.2 -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.

아파치 플링크 스트리밍 자바 코드를 업로드하세요.

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.

Apache Flink용 관리형 서비스 애플리케이션을 생성하고 실행합니다.

콘솔을 사용하여 애플리케이션을 생성, 구성, 업데이트 및 실행하려면 다음 단계를 수행하세요.

애플리케이션 생성

1. <https://console.aws.amazon.com/flink>에서 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

아파치 빔은 현재 아파치 플링크 버전 1.19 이상과 호환되지 않습니다.

- 버전 풀다운에서 아파치 플링크 버전 1.15를 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 리전을 사용하여 다음과 같이 지정됩니다.

- 정책: `kinesis-analytics-service-MyApplication-us-west-2`
- 역할: `kinesis-analytics-MyApplication-us-west-2`

IAM 정책을 편집합니다.

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/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. MyApplication페이지에서 구성을 선택합니다.

2. 애플리케이션 구성 페이지에서 코드 위치를 입력합니다.
 - Amazon S3 버킷의 경우 **ka-app-code-*<username>***를 입력합니다.
 - Amazon S3 객체 경로에는 **basic-beam-app-1.0.jar**를 입력합니다.
3. 애플리케이션 리소스에 대한 액세스 아래에서 액세스 권한의 경우 IAM 역할 **kinesis-analytcs-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 지표 수준 모니터링이 애플리케이션으로 설정되어 있는지 확인합니다.
6. CloudWatch 로깅의 경우 활성화 확인란을 선택합니다.
7. 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.

애플리케이션을 실행합니다

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.

리소스를 정리하세요. AWS

이 섹션에는 텀블링 윈도우 튜토리얼에서 만든 AWS 리소스를 정리하는 절차가 포함되어 있습니다.

이 주제는 다음 섹션을 포함하고 있습니다:

- [Apache Flink용 관리형 서비스 애플리케이션을 삭제하십시오.](#)
- [Kinesis 데이터 스트림을 삭제합니다.](#)
- [Amazon S3 객체 및 버킷을 삭제합니다.](#)
- [IAM 리소스를 삭제합니다.](#)
- [CloudWatch 리소스 삭제하기](#)

Apache Flink용 관리형 서비스 애플리케이션을 삭제하십시오.

1. <https://console.aws.amazon.com/flink>에서 Managed Service for Apache Flink 콘솔을 엽니다.
2. Apache Flink용 관리형 서비스 패널에서 선택합니다. MyApplication
3. 애플리케이션 페이지에서 삭제를 선택한 다음 삭제를 확인합니다.

Kinesis 데이터 스트림을 삭제합니다.

1. <https://console.aws.amazon.com/kinesis>에서 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 버킷을 삭제합니다.

1. <https://console.aws.amazon.com/s3>에서 S3 콘솔을 엽니다.

2. ka-app-code- 버킷을 선택합니다. <username>
3. 삭제를 선택한 후 버킷 이름을 입력하여 삭제를 확인합니다.

IAM 리소스를 삭제합니다.

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 선택합니다. MyApplication
8. 역할 삭제를 선택하고 삭제를 확인합니다.

CloudWatch 리소스 삭제하기

1. <https://console.aws.amazon.com/cloudwatch/> 에서 CloudWatch 콘솔을 엽니다.
2. 탐색 바에서 로그를 선택합니다.
3. MyApplication/aws/kinesis-analytics/ 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 내용을 알아보려면 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, 랩 및 솔루션 구현

다음 end-to-end 예는 Apache Flink용 고급 관리형 서비스 솔루션을 보여줍니다.

주제

- [Apache Flink용 Amazon 매니지드 서비스를 사용하여 애플리케이션을 배포, 운영 및 확장합니다.](#)
- [아파치 플링크용 매니지드 서비스에 배포하기 전에 로컬에서 아파치 플링크 애플리케이션을 개발하세요.](#)
- [Apache Flink Studio용 매니지드 서비스를 통해 이벤트 감지 기능을 사용하세요.](#)
- [Amazon Kinesis용 AWS 스트리밍 데이터 솔루션 사용](#)
- [Apache Flink 및 Apache Kafka와 함께 클릭스트림 랩을 사용해 연습해 보세요.](#)
- [Application Auto Scaling을 사용하여 사용자 지정 크기 조정 설정](#)
- [샘플 Amazon CloudWatch 대시보드 보기](#)
- [Amazon MSK용 AWS 스트리밍 데이터 솔루션용 템플릿 사용](#)
- [에서 Apache Flink용 매니지드 서비스 솔루션에 대해 자세히 알아보십시오. GitHub](#)

Apache Flink용 Amazon 매니지드 서비스를 사용하여 애플리케이션을 배포, 운영 및 확장합니다.

이 워크숍에서는 Java로 Apache Flink 애플리케이션을 개발하는 방법, IDE에서 실행 및 디버깅하는 방법, Apache Flink용 Amazon 관리 서비스에서 패키징, 배포 및 실행하는 방법을 다룹니다. 또한 애플리케이션을 확장하고 모니터링하고 문제를 해결하는 방법도 배우게 됩니다.

워크숍은 [Apache Flink용 아마존 매니지드 서비스](#) 워크숍에서 확인할 수 있습니다.

아파치 플링크용 매니지드 서비스에 배포하기 전에 로컬에서 아파치 플링크 애플리케이션을 개발하세요.

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](#)

Apache Flink Studio용 매니지드 서비스를 통해 이벤트 감지 기능을 사용하세요.

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 사용

Amazon Kinesis용 AWS 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.

각 솔루션은 다음 구성 요소를 포함합니다.

- 전체 예제를 배포하기 위한 AWS CloudFormation 패키지입니다.
- 애플리케이션 지표를 표시하기 위한 CloudWatch 대시보드.
- CloudWatch 가장 관련성이 높은 애플리케이션 지표에 대한 경보.
- 필요한 모든 IAM 역할 및 정책

솔루션은 [Amazon Kinesis용 스트리밍 데이터 솔루션](#)에서 찾을 수 있습니다.

Apache Flink 및 Apache Kafka와 함께 클릭스트림 랩을 사용해 연습해 보세요.

스트리밍 스토리지는 Apache Kafka용 Amazon Managed Streaming을 사용하고 스트림 처리에는 Apache Flink용 Managed Service를 사용하는 클릭스트림 사용 사례를 위한 포괄적인 실습입니다.

솔루션은 [클릭스트림 랩](#)에서 찾을 수 있습니다.

Application Auto Scaling을 사용하여 사용자 지정 크기 조정 설정

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 활성화](#)를 참조하십시오.

샘플 Amazon CloudWatch 대시보드 보기

Apache Flink 애플리케이션용 관리 서비스를 모니터링하기 위한 샘플 CloudWatch 대시보드입니다. 샘플 대시보드에는 대시보드의 기능을 시연하는 데 도움이 되는 [데모 애플리케이션](#)도 포함되어 있습니다.

솔루션은 [Managed Service for Apache Flink 지표 대시보드](#)에서 찾을 수 있습니다.

Amazon MSK용 AWS 스트리밍 데이터 솔루션용 템플릿 사용

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 솔루션](#)에서 찾을 수 있습니다.

에서 Apache Flink용 매니지드 서비스 솔루션에 대해 자세히 알아보십시오. 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](#)
- [벤치마킹](#)

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 [벤치마킹](#)을 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

- [Java 예제](#)
- [Python 예제](#)
- [스칼라 예제](#)

Java 예제

다음 예제는 Java로 작성된 애플리케이션을 만드는 방법을 보여줍니다.

Note

대부분의 예제는 개발 시스템과 선택한 IDE, 그리고 Apache Flink용 Amazon Managed Service에서 로컬로 실행되도록 설계되었습니다. 애플리케이션 매개 변수를 전달하는 데 사용할 수 있는 메커니즘과 두 환경 모두에서 변경 없이 애플리케이션을 실행하도록 종속성을 올바르게 설정하는 방법을 보여줍니다.

API 시작하기 DataStream

이 예제에서는 API를 사용하여 Kinesis 데이터 스트림에서 읽고 다른 Kinesis 데이터 스트림에 쓰는 간단한 애플리케이션을 보여줍니다. DataStream 이 예제는 올바른 종속성을 사용하여 파일을 설정하고, Uber-JAR을 빌드한 다음 구성 파라미터를 파싱하여 로컬, IDE 및 Apache Flink용 Amazon Managed Service 모두에서 애플리케이션을 실행할 수 있도록 하는 방법을 보여줍니다.

코드 예제: [GettingStarted](#)

테이블 API 및 SQL 시작하기

이 예제는 Table API와 SQL을 사용하는 간단한 애플리케이션을 보여줍니다. 동일한 Java 애플리케이션에서 DataStream API를 Table API 또는 SQL과 통합하는 방법을 보여줍니다. 또한 외부 데이터 생성기가 필요 없이 DataGen 커넥터를 사용하여 Flink 애플리케이션 자체 내에서 임의의 테스트 데이터를 생성하는 방법도 보여줍니다.

전체 예제: [GettingStartedTable](#)

S3 싱크 (DataStream API) 사용

이 예제는 DataStream API를 사용하여 S3 버킷에 JSON 파일을 쓰는 FileSink 방법을 보여줍니다.

코드 예제: [S3Sink](#)

Kinesis 소스, 표준 또는 EFO 소비자, 싱크 (API) 사용 DataStream

이 예제에서는 표준 소비자 또는 EFO를 사용하여 Kinesis 데이터 스트림의 소스 소비를 구성하는 방법과 Kinesis 데이터 스트림에 대한 싱크를 설정하는 방법을 보여줍니다.

코드 예제: [KinesisConnectors](#)

Amazon 데이터 파이어호스 싱크 (DataStream API) 사용

이 예제는 Amazon Data Firehose (이전에는 Kinesis Data Firehose로 알려짐) 로 데이터를 보내는 방법을 보여줍니다.

코드 예제: [KinesisFirehoseSink](#)

슬라이딩 및 텀블링 윈도우 사용 (API) DataStream

이 두 예제는 API를 사용하여 처리 시간 창 (슬라이딩 또는 텀블링) 에 대한 집계를 구현하는 방법을 보여줍니다. DataStream

코드 예제:

- [WindowingSliding](#)
- [WindowingTumbling](#)

사용자 정의 지표 사용

이 두 개의 개별 예제는 DataStream API에서 사용자 지정 지표를 구현하고 이를 지표로 CloudWatch 보내는 방법을 WordCount 보여줍니다. RecordCount

코드 예제: [CustomMetrics](#)

Python 예제

다음 예제는 Python으로 작성된 응용 프로그램을 만드는 방법을 보여줍니다.

Note

대부분의 예제는 개발 시스템과 선택한 IDE, 그리고 Apache Flink용 Amazon Managed Service에서 로컬로 실행되도록 설계되었습니다. 애플리케이션 매개 변수를 전달하는 데 사용할 수 있는 간단한 메커니즘과 두 환경 모두에서 변경 없이 애플리케이션을 실행하도록 종속성을 올바르게 설정하는 방법을 보여줍니다.

프로젝트 종속성

대부분의 PyFlink 예제에는 Flink 커넥터 또는 Python 타사 라이브러리와 같은 JAR 파일 형식의 종속성이 하나 이상 필요합니다. 이러한 종속성은 예제 저장소에 포함되지 않습니다. 애플리케이션을 패키징하기 전에 컴퓨터에 다운로드해야 합니다. 종속성에 대한 자세한 내용은 [README: 패키징](#)을 참조하십시오.

필요한 커넥터의 올바른 버전을 다운로드할 수 있는 링크를 찾으려면 Flink 설명서를 참조하십시오. 항상 사용 중인 Flink 버전에 대한 종속 항목을 사용하십시오. 다음은 Flink 1.18에서 자주 사용되는 커넥터 종속성입니다.

- [Kinesis SQL 커넥터](#)
- [카프카 SQL 커넥터](#)
- [아마존 데이터 파이어호스 SQL 커넥터](#)

다음 예제에는 종속성을 설정하는 데 필요한 코드가 이미 포함되어 있습니다.

예제

시작하기 PyFlink

이 예제는 Table API와 SQL을 사용하는 PyFlink 애플리케이션의 기본 구조를 보여줍니다. 또한 애플리케이션에 단일 JAR 종속성을 포함하는 방법도 보여줍니다. PyFlink 이 경우에는 Kinesis SQL 커넥터가 포함되어 있습니다.

코드 예제: [GettingStarted](#)

텀블링 및 슬라이딩 윈도우 사용

이 두 예제는 API를 사용하여 이벤트 타임에 텀블링 및 슬라이딩 윈도우를 구현하는 방법을 보여줍니다. Table 또한 예제는 이벤트 시간 윈도우에 사용되는 소스 테이블에 워터마크를 정의하는 방법을 보여줍니다. 두 예제 모두 Kinesis SQL 커넥터의 JAR 종속성도 포함합니다.

코드 예제:

- [TumblingWindows](#)
- [SlidingWindows](#)

S3 싱크 사용

이 예제는 출력을 Amazon S3에 JSON 파일로 쓰는 방법을 보여줍니다. Amazon S3에 파일을 쓰고 교체하려면 S3 싱크의 체크포인트 지정을 활성화해야 합니다.

코드 예제: [StreamingFileSink](#)

스칼라 예제

다음 예제는 Apache Flink와 함께 Scala를 사용하여 애플리케이션을 만드는 방법을 보여줍니다.

다단계 애플리케이션

이 예제는 Scala에서 Flink 애플리케이션을 설정하는 방법을 보여줍니다. 종속성을 포함하도록 SBT 프로젝트 구성하고 Uber-JAR를 빌드하는 방법을 보여줍니다.

코드 예제: [GettingStarted](#)

Amazon Managed Service for Apache Flink의 보안

클라우드 AWS 보안이 최우선 과제입니다. AWS 고객은 가장 보안에 민감한 조직의 요구 사항을 충족 하도록 구축된 데이터 센터 및 네트워크 아키텍처를 활용할 수 있습니다.

보안은 두 사람 사이의 AWS 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드 보안 - AWS 클라우드에서 AWS 서비스를 실행하는 인프라를 보호하는 역할을 합니다. AWS 또한 안전하게 사용할 수 있는 서비스를 제공합니다. 서드 파티 감사자는 정기적으로 [AWS 규정 준수 프로그램](#)의 일환으로 보안 효과를 테스트하고 검증합니다. Managed Service for Apache Flink에 적용되는 규정 준수 프로그램에 대한 자세한 내용을 알아보려면 [규정 준수 프로그램 제공 범위 내 AWS 서비스](#)를 참조하세요.
- 클라우드에서의 보안 — 귀하의 책임은 사용하는 AWS 서비스에 따라 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 이해 하는 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

- [아파치 플링크용 아마존 매니지드 서비스의 데이터 보호](#)
- [Amazon Managed Service for Apache Flink의 자격 증명 및 액세스 관리](#)
- [Managed Service for Apache Flink 모니터링](#)
- [아파치 플링크용 아마존 매니지드 서비스에 대한 규정 준수 검증](#)
- [Amazon Managed Service for Apache Flink의 복원성](#)
- [Apache Flink용 매니지드 서비스의 인프라 보안](#)
- [Apache Flink용 관리형 서비스의 보안 모범 사례](#)

아파치 플링크용 아마존 매니지드 서비스의 데이터 보호

에서 제공하는 도구를 사용하여 데이터를 보호할 수 있습니다. AWS Apache Flink용 관리형 서비스는 Firehose 및 Amazon S3를 비롯하여 데이터 암호화를 지원하는 서비스와 함께 사용할 수 있습니다.

아파치 플링크용 매니지드 서비스의 데이터 암호화

저장 중 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 설명은 [Kinesis 데이터 스트림용 서버 측 암호화란 무엇인가?](#)를 참조하십시오.
- Firehose를 사용하여 출력 데이터를 유휴 상태로 암호화하여 암호화된 Amazon S3 버킷에 데이터를 저장할 수 있습니다. Amazon S3 버킷이 사용하는 암호화 키를 지정할 수 있습니다. 자세한 정보는 [KMS 관리형 키\(SSE-KMS\)를 사용하는 서버 측 암호화로 데이터 보호](#)를 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 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 Flink 내의 내부 구성 요소 간에 전송 중인 데이터.
- Apache Flink용 관리형 서비스와 Firehose용 관리 서비스 간에 전송되는 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) 은 관리자가 리소스에 대한 액세스를 안전하게 제어할 수 있도록 도와줍니다. AWS IAM 관리자는 누가 인증(로그인) 및 권한(권한 있음)을 받아 Managed Service for Apache Flink 리소스를 사용할 수 있는지를 제어합니다. IAM은 추가 AWS 서비스 비용 없이 사용할 수 있습니다.

주제

- [고객](#)
- [ID를 통한 인증](#)
- [정책을 사용한 액세스 관리](#)
- [Amazon Managed Service for Apache Flink가 IAM과 연동되는 방식](#)
- [Amazon Managed Service for Apache Flink의 자격 증명 기반 정책 예제](#)
- [Amazon Managed Service for Apache Flink의 자격 증명 및 액세스 문제 해결](#)
- [교차 서비스 혼동된 대리인 방지](#)

고객

Apache Flink용 관리형 서비스에서 수행하는 작업에 따라 AWS Identity and Access Management (IAM) 사용 방식이 다릅니다.

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.

ID를 통한 인증

인증은 ID 자격 증명을 AWS 사용하여 로그인하는 방법입니다. IAM 사용자로 인증 (로그인 AWS) 하거나 IAM 역할을 맡아 인증 (로그인) 해야 합니다. AWS 계정 루트 사용자

ID 소스를 통해 제공된 자격 증명을 사용하여 페더레이션 ID로 로그인할 수 있습니다. AWS IAM Identity Center (IAM ID 센터) 사용자, 회사의 싱글 사인온 인증, Google 또는 Facebook 자격 증명이 페더레이션 ID의 예입니다. 페더레이션 ID로 로그인할 때 관리자가 이전에 IAM 역할을 사용하여 ID 페더레이션을 설정했습니다. 페더레이션을 사용하여 액세스하는 경우 AWS 간접적으로 역할을 맡게 됩니다.

사용자 유형에 따라 AWS Management Console 또는 AWS 액세스 포털에 로그인할 수 있습니다. 로그인에 대한 자세한 내용은 AWS 로그인 사용 설명서의 [내 로그인 방법](#)을 참조하십시오. AWS 계정

AWS 프로그래밍 방식으로 액세스하는 경우 자격 증명을 사용하여 요청에 암호화 방식으로 서명할 수 있는 소프트웨어 개발 키트 (SDK) 와 명령줄 인터페이스 (CLI) 를 AWS 제공합니다. AWS 도구를 사용하지 않는 경우 요청에 직접 서명해야 합니다. 권장 방법을 사용하여 직접 요청에 서명하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 AWS [API 요청 서명](#)을 참조하십시오.

사용하는 인증 방법에 상관없이 추가 보안 정보를 제공해야 할 수도 있습니다. 예를 들어, AWS 계정의 보안을 강화하기 위해 다단계 인증 (MFA) 을 사용할 것을 권장합니다. 자세한 내용은 AWS IAM Identity Center 사용 설명서의 [다중 인증](#) 및 IAM 사용 설명서의 [AWS에서 다중 인증\(MFA\) 사용](#)을 참조하십시오.

AWS 계정 루트 사용자

계정을 AWS 계정만들 때는 먼저 계정의 모든 AWS 서비스 리소스에 대한 완전한 액세스 권한을 가진 하나의 로그인 ID로 시작합니다. 이 ID를 AWS 계정 루트 사용자라고 하며, 계정을 만들 때 사용한 이메일 주소와 비밀번호로 로그인하여 액세스할 수 있습니다. 일상적인 태스크에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 보안 인증 정보를 보호하고 루트 사용자만 수행할 수 있

는 태스크를 수행하는 데 사용하세요. 루트 사용자로 로그인해야 하는 전체 작업 목록은 IAM 사용 설명서의 [루트 사용자 보안 인증이 필요한 작업](#)을 참조하십시오.

페더레이션 자격 증명

가장 좋은 방법은 관리자 액세스가 필요한 사용자를 비롯한 수동 AWS 서비스 사용자가 ID 공급자와의 페더레이션을 사용하여 임시 자격 증명을 사용하여 액세스하도록 하는 것입니다.

페더레이션 ID는 기업 사용자 디렉토리, 웹 ID 공급자, Identity Center 디렉터리의 사용자 또는 ID 소스를 통해 제공된 자격 증명을 사용하여 액세스하는 AWS 서비스 모든 사용자를 말합니다. AWS Directory Service 페더레이션 ID에 AWS 계정 액세스하면 이들이 역할을 맡고 역할은 임시 자격 증명을 제공합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center(을)를 사용하는 것이 좋습니다. IAM Identity Center에서 사용자 및 그룹을 생성하거나 자체 ID 소스의 사용자 및 그룹 집합에 연결하고 동기화하여 모든 사용자 및 애플리케이션에서 사용할 수 있습니다. AWS 계정 IAM Identity Center에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서에서 [IAM Identity Center란 무엇입니까?](#)를 참조하십시오.

IAM 사용자 및 그룹

[IAM 사용자는 단일 사용자](#) 또는 애플리케이션에 대한 특정 권한을 AWS 계정 가진 사용자 내 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 보안 인증이 필요한 특정 사용 사례가 있는 경우, 액세스 키를 교체하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [장기 보안 인증이 필요한 사용 사례의 경우 정기적으로 액세스 키 교체](#)를 참조하십시오.

[IAM 그룹](#)은 IAM 사용자 컬렉션을 지정하는 자격 증명입니다. 사용자는 그룹으로 로그인할 수 없습니다. 그룹을 사용하여 여러 사용자의 권한을 한 번에 지정할 수 있습니다. 그룹을 사용하면 대규모 사용자 집합의 권한을 더 쉽게 관리할 수 있습니다. 예를 들어, IAMAdmins라는 그룹이 있고 이 그룹에 IAM 리소스를 관리할 권한을 부여할 수 있습니다.

사용자는 역할과 다릅니다. 사용자는 한 사람 또는 애플리케이션과 고유하게 연결되지만, 역할은 해당 역할이 필요한 사람이라면 누구나 수입할 수 있습니다. 사용자는 영구적인 장기 보안 인증 정보를 가지고 있지만, 역할은 임시 보안 인증만 제공합니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자를 만들어야 하는 경우\(역할이 아님\)](#)를 참조하십시오.

IAM 역할

[IAM 역할](#)은 특정 권한을 가진 사용자 AWS 계정 내의 자격 증명입니다. IAM 사용자와 유사하지만, 특정 개인과 연결되지 않습니다. 역할을 AWS Management Console [전환하여](#) 에서 일시적으로 IAM 역

할을 맡을 수 있습니다. AWS CLI 또는 AWS API 작업을 호출하거나 사용자 지정 URL을 사용하여 역할을 수임할 수 있습니다. 역할 사용 방법에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 역할 사용](#)을 참조하십시오.

임시 보안 인증이 있는 IAM 역할은 다음과 같은 상황에서 유용합니다.

- 페더레이션 사용자 액세스 - 페더레이션 ID에 권한을 부여하려면 역할을 생성하고 해당 역할의 권한을 정의합니다. 페더레이션 ID가 인증되면 역할이 연결되고 역할에 정의된 권한이 부여됩니다. 페더레이션 역할에 대한 자세한 내용은 IAM 사용 설명서의 [서드 파티 ID 공급자의 역할 생성](#) 단원을 참조하십시오. IAM Identity Center를 사용하는 경우, 권한 집합을 구성합니다. 인증 후 ID가 액세스할 수 있는 항목을 제어하기 위해 IAM Identity Center는 권한 세트를 IAM의 역할과 연관짓습니다. 권한 세트에 대한 자세한 내용은 AWS IAM Identity Center 사용 설명서의 [권한 세트](#)를 참조하십시오.
- 임시 IAM 사용자 권한 - IAM 사용자 또는 역할은 IAM 역할을 수임하여 특정 태스크에 대한 다양한 권한을 임시로 받을 수 있습니다.
- 크로스 계정 액세스 - IAM 역할을 사용하여 다른 계정의 사용자(신뢰할 수 있는 보안 주체)가 내 계정의 리소스에 액세스하도록 허용할 수 있습니다. 역할은 계정 간 액세스를 부여하는 기본적인 방법입니다. 그러나 일부 AWS 서비스 경우에는 역할을 프록시로 사용하는 대신 정책을 리소스에 직접 연결할 수 있습니다. 크로스 계정 액세스에 대한 역할과 리소스 기반 정책의 차이점을 알아보려면 IAM 사용 설명서의 [IAM의 크로스 계정 리소스 액세스](#)를 참조하세요.
- 서비스 간 액세스 — 일부는 다른 AWS 서비스서비스의 기능을 AWS 서비스 사용합니다. 예를 들어 서비스에서 직접 호출을 수행하면 일반적으로 해당 서비스는 Amazon EC2에서 애플리케이션을 실행하거나 Amazon S3에 객체를 저장합니다. 서비스는 직접적으로 호출하는 보안 주체의 권한을 사용하거나, 서비스 역할을 사용하거나, 또는 서비스 연결 역할을 사용하여 이 태스크를 수행할 수 있습니다.
- 순방향 액세스 세션 (FAS) — IAM 사용자 또는 역할을 사용하여 작업을 수행하는 경우 보안 AWS 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 AWS 서비스서비스에 AWS 서비스 요청하기 위한 요청과 결합하여 사용합니다. FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 [전달 액세스 세션](#)을 참조하세요.
- 서비스 역할 - 서비스 역할은 서비스가 사용자를 대신하여 태스크를 수행하기 위해 맡는 [IAM 역할](#)입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 서비스에 대한 권한을 위임할 역할 생성](#)을 참조하십시오.
- 서비스 연결 역할 — 서비스 연결 역할은 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수임할 수 있습니다. 서비스 연결 역할은

사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

- Amazon EC2에서 실행되는 애플리케이션 — IAM 역할을 사용하여 EC2 인스턴스에서 실행되고 API 요청을 AWS CLI 하는 애플리케이션의 임시 자격 증명을 관리할 수 있습니다. AWS 이는 EC2 인스턴스 내에 액세스 키를 저장할 때 권장되는 방법입니다. EC2 인스턴스에 AWS 역할을 할당하고 모든 애플리케이션에서 사용할 수 있게 하려면 인스턴스에 연결된 인스턴스 프로필을 생성합니다. 인스턴스 프로파일에는 역할이 포함되어 있으며 EC2 인스턴스에서 실행되는 프로그램이 임시 보안 인증을 얻을 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 역할을 사용하여 Amazon EC2 인스턴스에서 실행되는 애플리케이션에 권한 부여](#)를 참조하십시오.

IAM 역할을 사용할지 또는 IAM 사용자를 사용할지를 알아보려면 [IAM 사용 설명서](#)의 IAM 역할(사용자 대신)을 생성하는 경우를 참조하십시오.

정책을 사용한 액세스 관리

정책을 생성하고 이를 AWS ID 또는 리소스에 AWS 연결하여 액세스를 제어할 수 있습니다. 정책은 ID 또는 리소스와 연결될 때 AWS 해당 권한을 정의하는 객체입니다. AWS 주도자 (사용자, 루트 사용자 또는 역할 세션) 가 요청할 때 이러한 정책을 평가합니다. 정책에서 권한은 요청이 허용되거나 거부되는 지를 결정합니다. 대부분의 정책은 JSON 문서로 AWS 저장됩니다. JSON 정책 문서의 구조와 콘텐츠에 대한 자세한 내용은 IAM 사용 설명서의 [JSON 정책 개요](#)를 참조하십시오.

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

기본적으로, 사용자와 역할에는 어떠한 권한도 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다. 그런 다음 관리자가 IAM 정책을 역할에 추가하고, 사용자가 역할을 수입할 수 있습니다.

IAM 정책은 작업을 수행하기 위해 사용하는 방법과 상관없이 작업에 대한 권한을 정의합니다. 예를 들어, iam:GetRole 작업을 허용하는 정책이 있다고 가정합니다. 해당 정책을 사용하는 사용자는 AWS Management Console, AWS CLI, 또는 AWS API에서 역할 정보를 가져올 수 있습니다.

보안 인증 기반 정책

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성](#)을 참조하십시오.

보안 인증 기반 정책은 인라인 정책 또는 관리형 정책으로 한층 더 분류할 수 있습니다. 인라인 정책은 단일 사용자, 그룹 또는 역할에 직접 포함됩니다. 관리형 정책은 내 여러 사용자, 그룹 및 역할에 연결할 수 있는 독립형 정책입니다. AWS 계정관리형 정책에는 AWS 관리형 정책과 고객 관리형 정책이 포함됩니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책과 인라인 정책의 선택](#)을 참조하십시오.

리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예는 IAM 역할 신뢰 정책과 Amazon S3 버킷 정책입니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 정책이 연결된 리소스의 경우, 정책은 지정된 보안 주체가 해당 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. IAM의 AWS 관리형 정책은 리소스 기반 정책에 사용할 수 없습니다.

액세스 제어 목록(ACL)

액세스 제어 목록(ACL)은 어떤 보안 주체(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACLs는 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

ACL을 지원하는 서비스의 예로는 아마존 S3와 아마존 VPC가 있습니다. AWS WAF ACL에 대해 자세히 알아보려면 Amazon Simple Storage Service 개발자 가이드의 [ACL\(액세스 제어 목록\) 개요](#)를 참조하십시오.

기타 정책 타입

AWS 일반적이지 않은 추가 정책 유형을 지원합니다. 이러한 정책 타입은 더 일반적인 정책 타입에 따라 사용자에게 부여되는 최대 권한을 설정할 수 있습니다.

- 권한 경계 – 권한 경계는 자격 증명 기반 정책에 따라 IAM 엔터티(IAM 사용자 또는 역할)에 부여할 수 있는 최대 권한을 설정하는 고급 기능입니다. 개체에 대한 권한 경계를 설정할 수 있습니다. 그 결과로 얻는 권한은 개체의 보안 인증 기반 정책과 그 권한 경계의 교집합입니다. Principal 필드에서 사용자나 역할을 지정하는 리소스 기반 정책은 권한 경계를 통해 제한되지 않습니다. 이러한 정책 중 하나에 포함된 명시적 거부 허용을 재정의합니다. 권한 경계에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 엔터티에 대한 권한 경계](#)를 참조하십시오.

- 서비스 제어 정책 (SCP) - SCP는 조직 또는 조직 단위 (OU) 에 대한 최대 권한을 지정하는 JSON 정책입니다. AWS Organizations AWS Organizations 사업체가 소유한 여러 AWS 계정 개를 그룹화하고 중앙에서 관리하는 서비스입니다. 조직에서 모든 기능을 활성화할 경우, 서비스 제어 정책 (SCP)을 임의의 또는 모든 계정에 적용할 수 있습니다. SCP는 구성원 계정의 엔티티 (각 엔티티 포함) 에 대한 권한을 제한합니다. AWS 계정 루트 사용자조직 및 SCP에 대한 자세한 내용은 AWS Organizations 사용 설명서의 [SCP 작동 방식](#)을 참조하십시오.
- 세션 정책 - 세션 정책은 역할 또는 페더레이션 사용자에게 대해 임시 세션을 프로그래밍 방식으로 생성할 때 파라미터로 전달하는 고급 정책입니다. 결과적으로 얻는 세션의 권한은 사용자 또는 역할의 보안 인증 기반 정책의 교차와 세션 정책입니다. 또한 권한을 리소스 기반 정책에서 가져올 수도 있습니다. 이러한 정책 중 하나에 포함된 명시적 거부는 허용을 재정의합니다. 자세한 내용은 IAM 사용 설명서의 [세션 정책](#)을 참조하십시오.

여러 정책 타입

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련되어 있을 때 요청을 허용할지 여부를 AWS 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 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	아니요
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	아니요

Apache Flink용 관리 서비스 및 기타 AWS 서비스가 대부분의 IAM 기능과 어떻게 작동하는지 자세히 알아보려면 IAM 사용 설명서에서 [IAM과 함께 작동하는AWS 서비스를](#) 참조하십시오.

Managed Service for Apache Flink를 위한 자격 증명 기반 정책

보안 인증 기반 정책 지원	예
----------------	---

자격 증명 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 자격 증명에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자와 역할이 어떤 리소스와 어떤 조건에서 어떤 태스크를 수행할 수 있는지를 제어합니다. ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성](#)을 참조하십시오.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. 보안 인증 기반 정책에서는 보안 주체가 연결된 사용자 또는 역할에 적용되므로 보안 주체를 지정할 수 없습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 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 태스크를 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [보안 주체를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 연동 사용자 등이 포함될 수 있습니다. AWS 서비스

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 보안 주체로 지정할 수 있습니다. 리소스 기반 정책에 크로스 계정 보안 주체를 추가하는 것은 트러스트 관계 설정의 절반밖에 되지 않는다는 것을 유념하십시오. 보안 주체와 리소스가 다른 AWS 계정경우 신뢰할 수 있는 계정의 IAM 관리자는 보안 주체 개체 (사용자 또는 역할)에게 리소스에 액세스할 수 있는 권한도 부여해야 합니다. 엔터티에 ID 기반 정책을 연결하여 권한을 부여합니다. 하지만 리소스 기반 정책이 동일 계정의 보안 주체에 액세스를 부여하는 경우, 추가 자격 증명 기반 정책이 필요하지 않습니다. 자세한 내용은 IAM 사용 설명서의 [IAM의 교차 계정 리소스 액세스](#)를 참조하십시오.

Managed Service for Apache Flink의 정책 작업

정책 작업 지원	예
<p>관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.</p> <p>JSON 정책의 Action요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 태스크를 설명합니다. 정책 작업은 일반적으로 관련 AWS API 작업과 이름이 같습니다. 일치하는 API 작업이 없는 권한 전용 작업 같은 몇 가지 예외도 있습니다. 정책에서 여러 작업이 필요한 몇 가지 작업도 있습니다. 이러한 추가 작업을 일컬어 종속 작업이라고 합니다.</p> <p>연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하십시오.</p>	
<p>Managed Service for Apache Flink 작업 목록을 보려면 서비스 승인 참조의 Amazon Managed Service for Apache Flink에서 정의한 작업을 참조하세요.</p>	
<p>Managed Service for Apache Flink의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.</p> <div data-bbox="90 1722 1536 1801" style="border: 1px solid #ccc; border-radius: 10px; padding: 10px; margin: 10px 0;">Kinesis Analytics</div>	

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 태스크를 지정하려면 다음 태스크를 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 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 개체를 지정합니다. 문장에는 Resource 또는 NotResource 요소가 반드시 추가되어야 합니다. 모범 사례에 따라 [Amazon 리소스 이름\(ARN\)](#)을 사용하여 리소스를 지정합니다. 리소스 수준 권한이라고 하는 특정 리소스 유형을 지원하는 작업에 대해 이 태스크를 수행할 수 있습니다.

작업 나열과 같이 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"

```

Managed Service for Apache Flink 리소스 유형 및 해당 ARN 목록을 보려면 서비스 승인 참조의 [Amazon Managed Service for Apache Flink에서 정의한 리소스](#)를 참조하세요. 각 리소스의 ARN을 지정할 수 있는 작업을 알아보려면 [Amazon Managed Service for Apache Flink에서 정의한 작업](#)을 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 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지 지정할 수 있습니다. 즉, 어떤 보안 주체가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소(또는 Condition 블록)를 사용하면 정책이 발효되는 조건을 지정할 수 있습니다. Condition 요소는 옵션입니다. 같거나 작음과 같은 [조건 연산자](#)를 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 키에 여러 값을 지정하는 경우는 논리적 OR 연산을 사용하여 조건을 AWS 평가합니다. 명문의 권한을 부여하기 전에 모든 조건을 충족해야 합니다.

조건을 지정할 때 자리 표시자 변수를 사용할 수도 있습니다. 예컨대, IAM 사용자에게 IAM 사용자 이름으로 태그가 지정된 경우에만 리소스에 액세스할 수 있는 권한을 부여할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM 정책 요소: 변수 및 태그](#)를 참조하십시오.

AWS 글로벌 조건 키 및 서비스별 조건 키를 지원합니다. 모든 AWS 글로벌 조건 키를 보려면 IAM 사용 [AWS 설명서의 글로벌 조건 컨텍스트 키](#)를 참조하십시오.

Managed Service for Apache Flink 조건 키 목록을 보려면 서비스 승인 참조의 [Amazon Managed Service for Apache Flink에 사용되는 조건 키](#)를 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
--------	-----

ACL(액세스 통제 목록)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.

Managed Service for Apache Flink의 ABAC(속성 기반 액세스 제어)

ABAC 지원(정책의 태그)

예

ABAC(속성 기반 액세스 제어)는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. AWS에서는 이러한 속성을 태그라고 합니다. IAM 개체 (사용자 또는 역할) 및 여러 AWS 리소스에 태그를 첨부할 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.

태그에 근거하여 액세스를 제어하려면 `aws:ResourceTag/key-name`, `aws:RequestTag/key-name` 또는 `aws:TagKeys` 조건 키를 사용하여 정책의 [조건 요소](#)에 태그 정보를 제공합니다.

서비스가 모든 리소스 유형에 대해 세 가지 조건 키를 모두 지원하는 경우, 값은 서비스에 대해 예입니다. 서비스가 일부 리소스 유형에 대해서만 세 가지 조건 키를 모두 지원하는 경우, 값은 부분적입니다.

ABAC에 대한 자세한 정보는 IAM 사용 설명서의 [ABAC란 무엇입니까?](#)를 참조하십시오. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 [ABAC\(속성 기반 액세스 제어\) 사용](#)을 참조하세요.

Managed Service for Apache Flink에서 임시 보안 인증 사용하기

임시 보안 인증 지원

예

임시 자격 증명을 사용하여 로그인하면 작동하지 AWS 서비스 않는 것도 있습니다. 임시 자격 증명을 사용하는 방법을 AWS 서비스 비롯한 추가 정보는 [IAM 사용 설명서의 IAM과AWS 서비스 연동되는 내용](#)을 참조하십시오.

사용자 이름과 암호를 제외한 다른 방법을 AWS Management Console 사용하여 로그인하면 임시 자격 증명을 사용하는 것입니다. 예를 들어 회사의 SSO (Single Sign-On) 링크를 AWS 사용하여 액세스하는 경우 이 프로세스에서 자동으로 임시 자격 증명을 생성합니다. 또한 콘솔에 사용자로 로그인한 다음 역할을 전환할 때 임시 보안 인증을 자동으로 생성합니다. 역할 전환에 대한 자세한 내용은 IAM 사용 설명서의 [역할로 전환\(콘솔\)](#)을 참조하십시오.

또는 API를 사용하여 임시 자격 증명을 수동으로 생성할 수 있습니다 AWS CLI . AWS 그런 다음 해당 임시 자격 증명을 사용하여 액세스할 수 AWS있습니다. AWS 장기 액세스 키를 사용하는 대신 임시 자격 증명을 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 작업을 수행하는 AWS경우 보안 주체로 간주됩니다. 일부 서비스를 사용하는 경우 다른 서비스에서 다른 작업을 시작하는 작업을 수행할 수 있습니다. FAS는 전화를 거는 주체의 권한을 다운스트림 서비스에 AWS 서비스 요청하라는 요청과 결합하여 사용합니다. AWS 서비스 FAS 요청은 다른 서비스 AWS 서비스 또는 리소스와의 상호 작용이 필요한 요청을 서비스가 수신한 경우에만 이루어집니다. 이 경우 두 작업을 모두 수행할 수 있는 권한이 있어야 합니다. FAS 요청 시 정책 세부 정보는 [전달 액세스 세션](#)을 참조하세요.

Managed Service for Apache Flink의 서비스 역할

서비스 역할 지원	예
-----------	---

서비스 역할은 서비스가 사용자를 대신하여 작업을 수행하는 것으로 가정하는 [IAM 역할](#)입니다. IAM 관리자는 IAM 내에서 서비스 역할을 생성, 수정 및 삭제할 수 있습니다. 자세한 정보는 IAM 사용 설명서의 [역할을 생성하여 AWS 서비스에 권한 위임](#)을 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 링크 역할 지원	예
--------------	---

서비스 연결 역할은 에 연결된 서비스 역할의 한 유형입니다. AWS 서비스서비스는 사용자를 대신하여 작업을 수행하기 위해 역할을 수입할 수 있습니다. 서비스 연결 역할은 사용자에게 AWS 계정 표시되며 해당 서비스가 소유합니다. IAM 관리자는 서비스 링크 역할의 권한을 볼 수 있지만 편집은 할 수 없습니다.

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 [IAM으로 작업하는AWS 서비스](#)를 참조하십시오. 서비스 연결 역할 열에서 Yes(이)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 사용하여 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 맡을 수 있습니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성](#)을 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 키](#)를 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 권한 권한으로 이동 — 사용자와 워크로드에 권한을 부여하려면 여러 일반적인 사용 사례에 권한을 부여하는 AWS 관리형 정책을 사용하세요. 해당 내용은 에서 사용할 수 있습니다. AWS 계정사용 사례에 맞는 AWS 고객 관리형 정책을 정의하여 권한을 더 줄이

는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 관리형 정책](#) 또는 [직무에 대한 AWS 관리형 정책](#)을 참조하십시오.

- 최소 권한 적용 – IAM 정책을 사용하여 권한을 설정하는 경우, 태스크를 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서의 [IAM의 정책 및 권한](#)을 참조하십시오.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 – 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어 SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. 예를 들어 AWS 서비스들에서 특정 작업을 통해 서비스 작업을 사용하는 경우 조건을 사용하여 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하십시오.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 신규 및 기존 정책을 확인합니다. IAM Access Analyzer는 100개 이상의 정책 확인 항목과 실행 가능한 추천을 제공하여 안전하고 기능적인 정책을 작성하도록 돕습니다. 자세한 내용은 IAM 사용 설명서의 [IAM Access Analyzer 정책 검증](#)을 참조하십시오.
- 멀티 팩터 인증 (MFA) 필요 - IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우 추가 보안을 위해 AWS 계정 MFA를 활성화하십시오. API 작업을 직접적으로 호출할 때 MFA가 필요하다면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 [MFA 보호 API 액세스 구성](#)을 참조하십시오.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 [IAM의 보안 모범 사례](#)를 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 엔티티(사용자 또는 역할)에 대해 의도대로 작동하지 않습니다.

또는 API만 호출하는 사용자에게 최소 콘솔 권한을 허용할 필요는 없습니다. AWS CLI AWS 그 대신, 수행하려는 API 작업과 일치하는 작업에만 액세스할 수 있도록 합니다.

사용자와 역할이 Apache Flink용 관리 서비스 콘솔을 계속 사용할 수 있도록 하려면 Apache Flink용 관리 서비스 ConsoleAccess 또는 ReadOnly AWS 관리형 정책도 엔티티에 연결하십시오. 자세한 정보는 IAM 사용 설명서의 [사용자에게 권한 추가](#)를 참조하세요.

사용자가 자신의 고유한 권한을 볼 수 있도록 허용

이 예제는 IAM 사용자가 자신의 사용자 ID에 연결된 인라인 및 관리형 정책을 볼 수 있도록 허용하는 정책을 생성하는 방법을 보여줍니다. 이 정책에는 콘솔에서 또는 API를 사용하여 프로그래밍 방식으로 이 작업을 완료할 수 있는 권한이 포함됩니다. AWS CLI AWS

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsWithUser",
        "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](#)
- [저는 IAM을 수행할 권한이 없습니다. PassRole](#)
- [내 AWS 계정 외부의 사용자가 Apache Flink용 관리형 서비스 리소스에 액세스할 수 있도록 허용하고 싶습니다.](#)

Managed Service for Apache Flink에서 작업을 수행할 권한이 없음

작업을 수행할 권한이 없다는 AWS Management Console 메시지가 표시되면 관리자에게 도움을 요청해야 합니다. 관리자는 사용자 이름과 비밀번호를 제공한 사람입니다.

다음 예제 오류는 mateojackson 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.

저는 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.

새 서비스 역할 또는 서비스 연결 역할을 만드는 대신 기존 역할을 해당 서비스에 전달할 AWS 서비스 수 있는 기능도 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 marymajor라는 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 필요하면 관리자에게 문의하세요. AWS 관리자는 로그인 자격 증명을 제공한 사람입니다.

내 AWS 계정 외부의 사용자가 Apache Flink용 관리형 서비스 리소스에 액세스할 수 있도록 허용하고 싶습니다.

다른 계정의 사용자 또는 조직 외부의 사람이 리소스에 액세스할 때 사용할 수 있는 역할을 생성할 수 있습니다. 역할을 수임할 신뢰할 수 있는 사람을 지정할 수 있습니다. 리소스 기반 정책 또는 액세스 제어 목록(ACL)을 지원하는 서비스의 경우 이러한 정책을 사용하여 다른 사람에게 리소스에 대한 액세스 권한을 부여할 수 있습니다.

자세히 알아보려면 다음을 참조하십시오.

- Managed Service for Apache Flink가 이러한 기능을 지원하는지 여부를 알아보려면 [Amazon Managed Service for Apache Flink가 IAM과 연동되는 방식](#) 섹션을 참조하세요.
- 소유하고 있는 리소스에 대한 액세스 권한을 부여하는 방법을 알아보려면 IAM 사용 [설명서에서 소유하고 AWS 계정 있는 다른 IAM 사용자에게 액세스 권한 제공](#)을 참조하십시오. AWS 계정
- [제3자에게 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 타사 AWS 계정AWS 계정 소유에 대한 액세스 제공](#)을 참조하십시오.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [외부에서 인증된 사용자에게 액세스 권한 제공\(자격 증명 페더레이션\)](#)을 참조하십시오.
- 교차 계정 액세스에 대한 역할 사용과 리소스 기반 정책의 차이점을 알아보려면 [IAM 사용 설명서의 IAM의 교차 계정 리소스 액세스](#)를 참조하십시오.

교차 서비스 혼동된 대리인 방지

에서 AWS서비스 간 사칭은 한 서비스 (호출 서비스)가 다른 서비스 (호출 서비스)를 호출할 때 발생할 수 있습니다. 호출 서비스는 다른 고객의 리소스에 대해 적절한 권한이 없는 방식으로 작동하게 권한을 사용하도록 조작될 수 있으며, 이로 인해 대리인 문제가 발생할 수 있습니다.

대리인이 혼동하지 않도록 계정 내 리소스에 대한 액세스 권한이 부여된 서비스 보안 주체를 사용하는 모든 서비스에 대한 사용자 데이터를 보호하는 데 도움이 되는 도구를 AWS 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`를 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 지정한 경우, ARN의 알 수 없는 부분에 대해 와일드카드 문자 (*)를 포함한 `aws:SourceArn` 키를 사용하십시오. 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 대리인 문제를 방지하는 방법

1. AWS [관리 콘솔에 로그인하고 https://console.aws.amazon.com/iam/](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 `aws:SourceArn` 및 `aws:SourceAccount` 글로벌 조건 컨텍스트 키 중 하나 또는 둘 다를 사용하는 정책으로 대체합니다. 다음 정책 예를 참조하십시오:
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 내용을 알아보려면 [로깅 및 모니터링](#) 섹션을 참조하세요.

아파치 플링크용 아마존 매니지드 서비스에 대한 규정 준수 검증

타사 감사자는 여러 AWS 규정 준수 프로그램의 일환으로 Apache Flink용 Amazon 관리형 서비스의 보안 및 규정 준수를 평가합니다. 여기에는 SOC, PCI, HIPAA 등이 포함됩니다.

특정 규정 준수 프로그램 범위에 속하는 AWS 서비스 목록은 [여기](#)를 참조하십시오. 일반 정보는 [AWS 규정 준수 프로그램](#)을 참조하십시오.

[여기](#)를 사용하여 타사 감사 보고서를 다운로드할 수 있는 AWS Artifact가 있습니다. 자세한 내용은 [AWS Artifact에서 보고서 다운로드](#)를 참조하십시오.

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 킷스타트 가이드](#) - 이 배포 가이드에서는 아키텍처 고려 사항을 설명하고 보안 및 규정 준수에 중점을 둔 기본 환경을 배포하기 위한 단계를 제공합니다. AWS
- [Architecting for HIPAA Security and Compliance on Amazon Web Services](#) 이 백서에서는 기업이 HIPAA 준수 애플리케이션을 개발하는 데 사용할 수 있는 방법을 설명합니다. AWS

- [AWS 규정 준수 리소스](#) — 이 통합 문서 및 가이드 모음은 해당 산업 및 지역에 적용될 수 있습니다.
- [AWS Config](#) — 이 AWS 서비스는 리소스 구성이 내부 관행, 업계 지침 및 규정을 얼마나 잘 준수하는지 평가합니다.
- [AWS Security Hub](#) — 이 AWS 서비스는 보안 업계 표준 및 모범 사례를 준수하는지 확인하는 데 도움이 되는 보안 상태를 종합적으로 보여줍니다.

FedRAMP

AWS FedRAMP 규정 준수 프로그램에는 FedRAMP 공인 서비스로 아파치 플링크용 매니지드 서비스가 포함되어 있습니다. 연방 또는 기업 고객의 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.

[영업 AWS 계정 관리자인 AWS FedRAMP PMO를 통해 FedRAMP 보안 패키지에 대한 액세스를 요청하거나 Artifact at Artifact를 통해 다운로드할 수 있습니다. AWSAWS](#)

자세한 정보는 [FedRAMP](#) 섹션을 참조하세요.

Amazon Managed Service for Apache Flink의 복원성

AWS 글로벌 인프라는 지역 및 가용 영역을 중심으로 AWS 구축됩니다. AWS 지역은 물리적으로 분리되고 격리된 여러 가용 영역을 제공하며, 이러한 가용 영역은 지연 시간이 짧고 처리량이 높으며 중복성이 높은 네트워킹으로 연결됩니다. 가용 영역을 사용하면 중단 없이 가용 영역 간에 자동으로 장애 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS [지역 및 가용 영역에 대한 자세한 내용은 글로벌 인프라를 참조하십시오AWS](#).

Apache Flink용 관리형 서비스는 AWS 글로벌 인프라 외에도 데이터 복원력 및 백업 요구 사항을 지원하는 데 도움이 되는 여러 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

JobManager Zookeeper를 사용하여 고가용성 모드로 실행됩니다. Managed Service for Apache Flink는 Amazon EKS를 사용하여 Apache Flink를 배포합니다. Amazon EKS에서는 가용 영역 전반의 각 AWS 지역에 대해 여러 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 작업의 파라미터를 사용하여 버전이 관리됩니다.

SnapshotName [CreateApplicationSnapshot](#)

Managed Service for Apache Flink 는 체크포인트와 세이브포인트에 저장된 데이터를 암호화합니다.

Apache Flink용 매니지드 서비스의 인프라 보안

관리형 서비스인 Apache Flink용 관리형 서비스는 [Amazon Web Services: 보안 프로세스 개요 백서에 설명된 AWS 글로벌 네트워크 보안](#) 절차에 따라 보호됩니다.

AWS 게시된 API 호출을 사용하여 네트워크를 통해 Apache Flink용 관리형 서비스에 액세스할 수 있습니다. 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 ID 및 IAM 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.

Apache Flink용 관리형 서비스의 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

Apache Flink용 관리형 서비스 애플리케이션에 Kinesis 데이터 스트림, Firehose 스트림 또는 Amazon S3 버킷과 같은 다른 서비스의 리소스에 액세스하려면 유효한 자격 증명이 있어야 합니다. AWS 자격 증명을 애플리케이션이나 Amazon S3 버킷에 직접 저장해서는 안 됩니다. 이러한 보안 인증은 자동으로 교체되지 않으며 손상된 경우 비즈니스에 큰 영향을 줄 수 있는 장기 보안 인증입니다.

대신 IAM 역할을 사용하여 애플리케이션이 다른 리소스에 액세스 할 수 있도록 임시 자격 증명을 관리해야 합니다. 역할을 사용하면 다른 리소스에 액세스하기 위해 장기 자격 증명을 사용할 필요가 없습니다.

자세한 설명은 IAM 사용자 가이드에서 다음 주제를 참조하십시오:

- [IAM 역할](#)
- [역할에 대한 일반적인 시나리오: 사용자, 애플리케이션 및 서비스](#)

종속 리소스에 서버 측 암호화를 구현하십시오.

저장 데이터와 전송 중인 데이터는 Managed Service for Apache Flink에서 암호화되며, 이 암호화는 비활성화할 수 없습니다. Kinesis 데이터 스트림, Firehose 스트림, Amazon S3 버킷과 같은 종속 리소

스에서 서버 측 암호화를 구현해야 합니다. 종속 리소스에서 서버 측 암호화 구현에 대한 자세한 설명은 [데이터 보호](#) 섹션을 참조하십시오.

API 호출을 모니터링하는 데 사용합니다. CloudTrail

Apache Flink용 관리형 서비스는 Apache Flink용 관리형 서비스의 사용자, 역할 또는 Amazon 서비스가 수행한 작업의 기록을 제공하는 서비스와 통합되어 있습니다. AWS CloudTrail

에서 수집한 정보를 사용하여 Apache Flink용 관리형 서비스에 이루어진 요청 CloudTrail, 요청이 이루어진 IP 주소, 요청한 사람, 요청 시기 및 추가 세부 정보를 확인할 수 있습니다.

자세한 정보는 [the section called “사용 AWS CloudTrail”](#)을 참조하세요.

아파치 플링크용 Amazon 매니지드 서비스의 로깅 및 모니터링

모니터링은 Managed Service for Apache Flink 애플리케이션의 신뢰성, 가용성 및 성능을 유지하는 중요한 역할을 합니다. 멀티포인트 장애가 발생할 경우 이를 보다 쉽게 디버깅할 수 있도록 AWS 솔루션의 모든 부분에서 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. 그러면 현재 성능 데이터와 비교하고, 일반적인 성능 패턴과 성능 이상을 식별하고, 문제를 해결할 방법을 강구할 수 있습니다.

주제

- [로깅](#)
- [모니터링](#)
- [애플리케이션 로깅 설정](#)
- [Logs Insights를 통한 CloudWatch 로그 분석](#)
- [Apache Flink용 관리 서비스에서 지표 및 측정기준 보기](#)
- [로그에 사용자 지정 메시지 작성 CloudWatch](#)
- [다음을 사용하여 Apache Flink API 호출을 위한 관리 서비스를 로깅합니다. AWS CloudTrail](#)

로깅

로깅은 프로덕션 애플리케이션에서 오류와 장애를 파악하는 데 중요합니다. 하지만 로깅 하위 시스템은 로그 항목을 수집하여 CloudWatch Logs에 전달해야 합니다. 일부 로깅은 괜찮고 바람직하지만, 광범위한 로깅은 서비스에 과부하가 걸리고 Flink 응용 프로그램이 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.

Logs Insights를 사용하여 로그를 쿼리합니다. 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 예](#)를 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 볼 때)?

- 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 대시보드](#)는 좋은 출발점을 제공합니다. 프로덕션 애플리케이션에 대해 모니터링할 지표에 대한 자세한 내용은 [아마존 매니지드 서비스에서 아파치 플링크용 CloudWatch 알람 사용하기](#) 섹션을 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 `heapMemoryUtilization` 및 — 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.
- `lastCheckpointSize` 그리고 `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 늘어듭니다. 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.
- `numberOfFailedCheckpoint` — 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 Console 또는 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](#)
- [CLI를 사용한 CloudWatch 로깅 설정](#)
- [애플리케이션 모니터링 수준](#)
- [로깅 모범 사례](#)
- [로깅 문제 해결](#)
- [다음 단계](#)

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”](#) 섹션을 참조하세요.

CLI를 사용한 CloudWatch 로깅 설정

를 사용하여 CloudWatch 로깅 옵션을 추가하려면 다음과 AWS CLI같이 하십시오.

- CloudWatch 로그 그룹과 로그 스트림을 생성합니다.
- 작업을 사용하여 응용 프로그램을 만들 때 로깅 옵션을 추가하거나 [CreateApplication](#) 작업을 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](#)
- [애플리케이션 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](#).

애플리케이션 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 [아파치 플링크용 매니지드 서비스 API 예제 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”](#) 섹션을 참조하세요.

애플리케이션을 생성할 때 애플리케이션 로깅 수준을 설정합니다.

다음 예제의 [CreateApplication](#)요청은 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 [UpdateApplication](#)요청은 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 Apache Flink용 관리형 서비스가 위임하는 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 관리](#) 섹션을 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": "kinesisanalytics.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”](#) 섹션을 참조하세요.
- 연산자: 지표의 범위는 각 연산자별로 지정됩니다. 연산자에 대한 자세한 내용은 [the section called “DataStream API 연산자”](#) 섹션을 참조하세요.
- 병렬 처리: 지표의 범위는 애플리케이션 병렬 처리로 지정됩니다. [UpdateApplicationAPI의 MonitoringConfigurationUpdate](#) 파라미터를 사용해서만 이 지표 수준을 설정할 수 있습니다. 콘솔을 사용하여 이 지표 수준을 설정할 수는 없습니다. 병렬 처리에 대한 자세한 내용은 [the section called “스케일링”](#) 섹션을 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 내용을 알아보려면 [the section called “ CloudWatch 로그 스트림에 쓰기 권한 추가”](#) 섹션을 참조하세요.

- 애플리케이션이 실행 중인지 확인합니다. 애플리케이션 상태를 확인하려면 콘솔에서 애플리케이션 페이지를 보거나 [DescribeApplication](#) 또는 [ListApplications](#) 작업을 사용하십시오.
- 기타 애플리케이션 문제를 downtime 진단하는 등의 CloudWatch 지표를 모니터링하세요. CloudWatch 지표 읽기에 대한 자세한 내용은 [을 참조하십시오 Apache Flink용 매니지드 서비스의 메트릭과 디멘션](#).

다음 단계

애플리케이션에서 CloudWatch 로깅을 활성화한 후에는 CloudWatch Logs Insights를 사용하여 애플리케이션 로그를 분석할 수 있습니다. 자세한 정보는 [the section called “로그 분석”](#)을 참조하세요.

Logs Insights를 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 분석](#)을 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 CloudTrail같은 서비스를 사용하는 경우 해당 서비스에서 로그로 이동하도록 CloudWatch 이미 로그를 설정했을 것입니다. 로그로 CloudWatch 로그를 보내는 방법에 대한 자세한 내용은 로그 [시작하기](#)를 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 애플리케이션 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 예제에서 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: 병렬성 변경](#)
- [오류 분석: 액세스가 거부되었습니다.](#)
- [오류 분석: 소스 또는 싱크를 찾을 수 없음](#)
- [오류 분석: 애플리케이션 작업 관련 장애](#)

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 [스케일링](#) 섹션을 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 수준](#)을 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 예약](#)을 참조하세요.

작업 분석: 병렬성 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 크기 조정”](#) 섹션을 참조하세요.

```
fields @timestamp, @parallelism
| filter message like /property: parallelism.default, /
| parse message "default, *" as @parallelism
| sort @timestamp asc
```

오류 분석: 액세스가 거부되었습니다.

다음 CloudWatch 로그 인사이트 쿼리는 Access Denied 로그를 반환합니다.

```
fields @timestamp, @message, @messageType
| filter applicationARN like /arn:aws:kinesisanalyticsus-
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 장애

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
```

Apache Flink용 관리 서비스에서 지표 및 측정기준 보기

이 주제는 다음 섹션을 포함하고 있습니다.

- [애플리케이션 지표](#)
- [Kinesis Data Streams 커넥터 메트릭](#)
- [아마존 MSK 커넥터 메트릭](#)
- [아파치 제플린 메트릭](#)
- [지표 보기 CloudWatch](#)
- [CloudWatch 지표 보고 수준 설정](#)
- [Apache Flink용 Amazon 매니지드 서비스에서 사용자 지정 지표 사용](#)
- [아마존 매니지드 서비스에서 아파치 플링크용 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	<p>*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다.</p> <p>이러한 지표는 애플리케이션의 병목 현상을 식별하는 데 유용할 수 있습니다.</p>

지표	단위	설명	수준	사용 관련 참고 사항
busyTimeMsPerSecond*	밀리초	이 작업 또는 연산자가 초당 사용 중인(유휴 상태도 배압 상태도 아닌) 시간(밀리초)입니다. 값을 계산할 수 없는 경우 NaN이 될 수 있습니다.	작업, 연산자, 병렬	<p>*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다.</p> <p>이러한 지표는 애플리케이션의 병목 현상을 식별하는 데 유용할 수 있습니다.</p>
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	<p>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.</p>

지표	단위	설명	수준	사용 관련 참고 사항
container CPUUtilization	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	<p>컨테이너별로 다음과 같이 계산됩니다.</p> <p>컨테이너가 사용한 총 CPU 시간(초) * 100 / 컨테이너 CPU 제한(CPU/초)</p> <p>CPUUtilization 지표에는 컨테이너 내에서 실행되는 TaskManager JVM 프로세스의 CPU 사용량만 고려됩니다. 동일한 컨테이너 내에서 JVM 외부에서 실행되는 다른 구성 요소도 있습니다. 이 container CPUUtilization 지표는 컨테이너의 CPU 소모 및 이로 인한 장애 측면에서 모든 프로세스를 포함하여 보다 완</p>

지표	단위	설명	수준	사용 관련 참고 사항	
				전한 그림을 제공합니다.	

지표	단위	설명	수준	사용 관련 참고 사항
container MemoryUtilization	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	<p>컨테이너별로 다음과 같이 계산됩니다.</p> <p>컨테이너 메모리 사용량(바이트) * 100/ 포드 배포 사양에 따른 컨테이너 메모리 제한(바이트)</p> <p>HeapMemoryUtilization 및 ManagedMemoryUtilizations 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 container MemoryUtilization 지</p>

지표	단위	설명	수준	사용 관련 참고 사항
				<p>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</p>

지표	단위	설명	수준	사용 관련 참고 사항
container DiskUtilization	백분율	Flink 애플리케이션 클러스터의 작업 관리자 컨테이너 전체 디스크 사용률입니다. 예를 들어 작업 관리자가 5명이고 그에 따라 TaskManager 컨테이너가 다섯 개이고 Apache Flink 용 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	<p>*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다.</p> <p>이러한 지표는 애플리케이션의 병목 현상을 식별하는 데 유용할 수 있습니다.</p>
lastCheckpointSize	바이트	마지막 체크포인트의 총 크기	애플리케이션	<p>이 지표를 사용하여 실행 중인 애플리케이션 스토리지 사용률을 확인할 수 있습니다.</p> <p>이 지표의 값이 증가하면 메모리 누수 또는 병목 현상 등 애플리케이션에 문제가 있음을 의미할 수 있습니다.</p>

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	<p>*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에 서만 사용할 수 있습니다.</p> <p>이는 Java 힙 외부에서 Flink가 관리하는 메모리와 관련이 있습니다. RocksDB 상태 백엔드에 사용되며 애플리케이션에서도 사용할 수 있습니다.</p>

지표	단위	설명	수준	사용 관련 참고 사항
managedMemoryTotal*	바이트	관리형 메모리의 총량입니다.	애플리케이션, 연산자, 작업, 병렬 처리	<p>*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다.</p> <p>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.</p>

지표	단위	설명	수준	사용 관련 참고 사항
managedMemoryUtilization*	백분율	/에서 파생됨 managedMemoryUsed managedMemoryTotal	애플리케이션, 연산자, 작업, 병렬 처리	<p>*Flink 버전 1.13을 구동하는 Managed Service for Apache Flink에서만 사용할 수 있습니다.</p> <p>이는 Java 힙 외부에서 Flink가 관리하는 메모리와 관련이 있습니다. RocksDB 상태 백엔드에 사용되며 애플리케이션에서도 사용할 수 있습니다.</p>
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	<p>*일정 기간 (초/분)에 대한 SUM 통계를 적용하려면:</p> <ul style="list-style-type: none"> 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. <p>지표의 수준은 이 지표가 전체 애플리케이션,</p>

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	<p>*일정 기간 (초/분)에 대한 SUM 통계를 적용하려면:</p> <ul style="list-style-type: none"> 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 $m1$은 기간(초/분)에 대한 SUM 통계입니다. <p>지표의 수준은 이 지표가 전체 애플리케이션,</p>

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	<p>*일정 기간(초/분)에 대한 SUM 통계를 적용하려면:</p> <ul style="list-style-type: none"> 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. <p>지표의 수준은 이 지표가 전체 애플리케이션,</p>

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		<p>*일정 기간 (초/분)에 대한 SUM 통계를 적용하려면:</p> <ul style="list-style-type: none"> 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. <p>이 연산자 또는 작업이 늦게 도착하여 삭제한</p>

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	<p>*일정 기간 (초/분)에 대한 SUM 통계를 적용하려면:</p> <ul style="list-style-type: none"> 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. <p>지표의 수준은 이 지표가 전체 애플리케이션,</p>

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.
KPUs*	개수	애플리케이션에서 사용한 총 KPU 수.	애플리케이션	<p>*이 지표는 청구 기간 (1시간) 당 하나의 샘플을 받습니다. 시간 경과에 따른 KPU 수를 시각화하려면 최소 1시간 동안 MAX 또는 AVG를 사용하십시오.</p> <p>KPU 수에는 KPU가 포함됩니다. orchestration 자세한 내용은 Apache Flink용 관리형 서비스 요금을 참조하십시오.</p>

Kinesis Data Streams 커넥터 메트릭

AWS 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	<ul style="list-style-type: none"> 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 않았음을 나타냅니다.
bytesRequestedPerFetch	바이트	getRecords에 대한 단일 호출에서 요청된 바이트 수입니다.	애플리케이션 (스트림용), 병렬성 (용) ShardId	

아마존 MSK 커넥터 메트릭

AWS 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 영향을 미치지 않습니다.
commitsSuccessful	N/A	오프셋 커밋과 체크포인트가 활성화된 경우, Kafka에 성공적으로 커밋한 총 오프셋 횟수입니다.	애플리케이션, 연산자, 작업, 병렬 처리	
committedoffsets	N/A	각 파티션에 대해 Kafka에 마지막으로 성공적으로 커밋된 오프셋 특정 파티션의 지표는 주제 이름 및	애플리케이션 (주제용), 병렬성 (대상) PartitionId	

지표	단위	설명	수준	사용 관련 참고 사항
		파티션 ID로 지정할 수 있습니다.		
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	

아파치 제플린 메트릭

Studio 노트북의 경우 애플리케이션 수준에서,, AWS KPIs, cpuUtilization 및 등의 지표를 내보냅니다. heapMemoryUtilization oldGenerationGCtime oldGenerationGCCount threadCount 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	jetty_threads_jobs

지표	단위	설명	Prometheus 이름
		Zeppelin 작업 수입니다.	
zeppelinServerUptime	초	서버가 가동되어 실행된 총 시간.	process_uptime_seconds

지표 보기 CloudWatch

Amazon CloudWatch 콘솔 또는 CLI를 사용하여 애플리케이션의 CloudWatch 지표를 볼 수 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.

CLI를 사용하여 지표를 보려면 AWS CLI

- 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 수준](#)을 참조하세요.

Apache Flink용 Amazon 매니지드 서비스에서 사용자 지정 지표 사용

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](#)
- [예](#)
- [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 유형](#)을 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 [범위](#)를 참조하세요.

Apache Flink 지표에 대한 자세한 내용은 [Apache Flink 설명서](#)의 [지표](#)를 참조하세요.

Managed Service for Apache Flink에서 사용자 정의 지표를 생성하려면, RichFunction를 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 [MetricGroup](#) 개체를 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 [CreateApplication](#) 작업의 또는 [MonitoringConfiguration](#) 매개 변수를 사용하여 응용 프로그램의 모니터링 수준을 설정합니다. [MonitoringConfigurationUpdateUpdateApplication](#)

예

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 애플리케이션](#)을 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.

아마존 매니지드 서비스에서 아파치 플링크용 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 사용](#)을 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 비취 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
downtime> 0	평균	0	다운타임이 0보다 크면 애플리케이션이 실패했음을 나타냅니다. 값이 0보다 크면 애플리케이션에서 데이터를 처리하지 않는 것입니다. 모든 애플리케이션에 권장됩니다. 이 Downtime 지표는 정전 기간을 측정합니다. 다운타임이 0보다 크면 애플리케이션이 실패했음을 나타냅니다. 문제 해결에 대한 내용은 참조하십시오 애플리케이션 재시작 중 .

지표 표현식	통계	Threshold	설명
RATE (numberOfFailedCheckpoints) > 0	평균	0	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 기울기에 따라 경보를 울리려면 RATE (numberOfFailed체크포인트) 를 모니터링하는 것이 좋습니다. 모든 애플리케이션에 권장됩니다. 이 지표를 사용하여 애플리케이션 상태 및 체크포인트 진행 상황을 모니터링할 수 있습니다. 애플리케이션은 상태가 정상일 때 상태 데이터를 체크포인트에 저장합니다. 애플리케이션이 입력 데이터 처리를 진행하지 않는 경우 시간 초과로 인해 체크포인트가 실패할 수 있습니다. 문제 해결에 대한 내용은 을

지표 표현식	통계	Threshold	설명
Operator. numRecord sOutPerSecond < 임계값	평균	정상 상태에서 애플리케이션에서 내보내는 최소 레코드 수입니다.	참조하십시오. 체크포인트 시간이 초과되었습니다. 모든 애플리케이션에 권장됩니다. 이 임계값 아래로 떨어지면 애플리케이션이 입력 데이터에 대해 예상한 진행률을 보이지 않고 있음을 의미할 수 있습니다. 문제 해결에 대한 내용은 참조하십시오. 오처리량이 너무 느림.

지표 표현식	통계	Threshold	설명
records_lag_max millisbehindLatest > 임계값	Maximum	정상 상태에서의 최대 예상 지연 시간.	애플리케이션이 Kinesis 또는 Kafka를 사용하는 경우 이러한 지표는 애플리케이션이 지연되고 있으며 현재 로드를 따라잡기 위해 규모를 조정해야 하는지 여부를 나타냅니다. 이는 모든 종류의 애플리케이션에서 쉽게 추적할 수 있는 유용한 일반 지표입니다. 하지만 애플리케이션이 이미 뒤쳐진 경우 등 반응형 스케일링에만 사용할 수 있습니다. 모든 애플리케이션에 권장됩니다. Kafka 소스의 경우 records_lag_max 메트릭을 사용하고 Kinesis 스트림 소스의 millisbehindLatest 경우를 사용하십시오. 이 임계값을 초과하면 애플리케이션이 입력 데이터에 대해 예상한 진행 상황을 보이지 않을 수 있습니다. 문제 해결에 대한 내용은 참조하십시오 처리량이 너무 느림 .

지표 표현식	통계	Threshold	설명
lastCheck pointDuration > 임계값	Maximum	정상 상태에서의 최대 예상 체크포인트 지속 시간	상태에 저장된 데이터 의 양과 체크포인트 를 생성하는 데 걸리 는 시간을 모니터링합 니다. 체크포인트가 늘 어나거나 시간이 오래 걸리면 애플리케이션 은 체크포인트에 지속 적으로 시간을 소비하 게 되고 실제 처리에 소요되는 주기도 줄어 듭니다. 어떤 지점에 서는 체크포인트가 너 무 커지거나 너무 오 래 걸려 실패할 수 있 습니다. 절대값을 모 니터링하는 것 외에도 고객은 RATE(last Checkpoint tSize) 및 RATE(last Checkpoint tDuration) 를 사용하여 변경률을 모니터링하는 것도 고려해야 합니다. 값이 lastCheck pointDuration 계속 증가하면 이 임 계값 이상으로 올라가 면 응용 프로그램이 입 력 데이터에 대해 예상 한 진행률을 보이지 않 거나 배압과 같은 응용

지표 표현식	통계	Threshold	설명
			프로그램 상태에 문제가 있음을 의미할 수 있습니다. 문제 해결에 대한 내용은 을 참조하십시오 .

지표 표현식	통계	Threshold	설명
lastCheck pointSize > 임계 값	Maximum	정상 조건에서 예상되 는 최대 체크포인트 크 기.	상태에 저장된 데이터 의 양과 체크포인트 를 생성하는 데 걸리 는 시간을 모니터링합 니다. 체크포인트가 늘 어나거나 시간이 오래 걸리면 애플리케이션 은 체크포인트에 지속 적으로 시간을 소비하 게 되고 실제 처리에 소요되는 주기도 줄어 듭니다. 어떤 지점에 서는 체크포인트가 너 무 커지거나 너무 오 래 걸려 실패할 수 있 습니다. 절대값을 모 니터링하는 것 외에도 고객은 RATE(last Checkpoint tSize) 및 RATE(last Checkpoint tDuration) 를 사용하여 변경률을 모니터링하는 것도 고려해야 합니다. 이 값이 lastCheck pointSize 계속 증가하는 경우 이 임계 값을 초과하면 응용 프 로그램이 상태 데이터 를 누적하고 있음을 의 미할 수 있습니다. 상 태 데이터가 너무 커지

지표 표현식	통계	Threshold	설명
heapMemoryUtilization > 임계값	Maximum	이 애플리케이션의 전체 리소스 사용을 잘 나타내며 애플리케이션이 I/O에 바인딩되지 않는 한 사전 예방적 확장에 사용할 수 있습니다. 정상 상태에서의 최대 예상 heapMemoryUtilization 크기, 권장 값은 90%입니다.	면 검사점에서 복구할 때 응용 프로그램의 메모리가 부족해지거나 검사점에서 복구하는데 시간이 너무 오래 걸릴 수 있습니다. 문제 해결에 대한 내용은 참조하십시오. 무한한 국가 성장 이 메트릭을 사용하여 응용 프로그램 전반에서 작업 관리자의 최대 메모리 사용률을 모니터링할 수 있습니다. 응용 프로그램이 이 임계값에 도달하면 더 많은 리소스를 프로비전해야 합니다. 이를 위해서는 자동 크기 조정을 활성화하거나 애플리케이션 병렬성을 높이면 됩니다. 리소스 증가에 대한 자세한 내용은 참조하십시오. 스케일링

지표 표현식	통계	Threshold	설명
cpuUtilization > 임계값	Maximum	이는 애플리케이션의 전체 리소스 사용률을 잘 나타내며 애플리케이션이 I/O에 바인딩되지 않는 한 사전 예방적 확장에 사용할 수 있습니다. 정상 상태에서의 최대 예상 cpuUtilization 크기, 권장 값은 80%입니다.	이 지표를 사용하여 응용 프로그램 전반에서 작업 관리자의 최대 CPU 사용률을 모니터링할 수 있습니다. 응용 프로그램이 이 임계값에 도달하면 자동 크기 조정을 활성화하거나 응용 프로그램 병렬성을 높이면 더 많은 리소스를 프로비전해야 합니다. 리소스 증가에 대한 자세한 내용은 참조하십시오 . 스케일링
threadsCount > 임계값	Maximum	정상 상태에서의 최대 예상 threadsCount 크기.	이 지표를 사용하여 애플리케이션 전반의 작업 관리자에서 스레드 수를 관찰할 수 있습니다. 이 지표가 이 임계값에 도달하면 애플리케이션 코드에서 스레드가 달하지 않고 생성되고 있는지 확인하세요.

지표 표현식	통계	Threshold	설명
<code>(oldGarbageCollectionTime * 100)/60_000 over 1 min period') ></code> 임계값	Maximum	최대 예상 oldGarbageCollectionTime 기간. 일반적인 가비지 수집 시간이 지정된 임계값의 60% 가 되도록 임계값을 설정하는 것이 좋지만 애플리케이션에 맞는 임계값은 달라질 수 있습니다.	이 지표가 계속 증가하면 응용 프로그램 전체의 작업 관리자에 메모리 누수가 발생한 것일 수 있습니다.
<code>RATE(oldGarbageCollectionCount) ></code> 임계값	Maximum	정상 oldGarbageCollectionCount 조건에서 예상되는 최대값입니다. 신청에 맞는 정확한 임계값은 달라질 수 있습니다.	이 지표가 계속 증가하면 응용 프로그램 전체의 작업 관리자에 메모리 누수가 발생한 것일 수 있습니다.
<code>Operator.currentOutputWatermark - Operator.currentInputWatermark ></code> 임계값	최소	정상 조건에서 예상되는 최소 워터마크 증분량 애플리케이션에 맞는 정확한 임계값은 달라질 수 있습니다.	이 지표가 계속 증가한다면 애플리케이션에서 점점 더 오래된 이벤트를 처리하고 있거나 업스트림 하위 작업이 워터마크를 보낸 지 점점 더 오랜 시간이 지났음을 의미할 수 있습니다.

로그에 사용자 지정 메시지 작성 CloudWatch

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 기록합니다.](#)
- [SLF4J 사용하여 CloudWatch 로그에 쓰기](#)

Log4J를 사용하여 CloudWatch 로그에 기록합니다.

1. 애플리케이션의 pom.xml 파일에 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 사용하여 CloudWatch 로그에 쓰기

1. 애플리케이션의 pom.xml 파일에 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"
}

```

다음을 사용하여 Apache Flink API 호출을 위한 관리 서비스를 로깅합니다. AWS CloudTrail

Apache Flink용 관리형 서비스는 Apache Flink용 관리형 서비스에서 사용자, 역할 또는 서비스가 수행한 작업에 대한 기록을 제공하는 AWS 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 관리형 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 CloudTrail 사용 설명서](#)를 참조하십시오.

Apache Flink용 관리형 서비스 정보는 다음을 참조하십시오. 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 최근 이벤트를 보고, 검색하고, 다운로드할 수 있습니다. 자세한 내용은 이벤트 [기록으로 CloudTrail 이벤트 보기를](#) 참조하십시오.

Apache Flink용 Managed Service 이벤트를 포함하여 AWS 계정의 진행 중인 이벤트 기록을 보려면 트레일을 생성하세요. 트레일을 사용하면 CloudTrail Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 기본적으로 콘솔에서 트레일을 생성하면 트레일이 모든 AWS 지역에 적용됩니다. 트레일은 AWS 파티션에 있는 모든 지역의 이벤트를 기록하고 지정한 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가로 분석하고 이에 따라 조치를 취하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음 자료를 참조하십시오.

- [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 사용자에게 대한 정보가 들어 있습니다. 보안 인증 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청이 루트 또는 AWS Identity and Access Management (IAM) 사용자 자격 증명으로 이루어졌는지 여부.
- 역할 또는 페더레이션 사용자에게 대한 임시 보안 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지 여부.

자세한 내용은 [CloudTrail 사용자 ID 요소를 참조하십시오](#).

Apache Flink용 관리형 서비스 로그 파일 항목의 이해

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 [AddApplicationCloudWatchLoggingOption](#) 및 [DescribeApplication](#) 작업을 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": "kinesisanalytics.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": "kinesisanalytics.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"
  }
]
}
```

아파치 플링크용 Amazon 매니지드 서비스의 성능 조정

이 항목에서는 Managed Service for Apache Flink 애플리케이션의 성능을 모니터링하고 개선하는 기술에 대해 설명합니다.

주제

- [성능 문제 해결](#)
- [성능 모범 사례](#)
- [모니터링 성능](#)

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 직렬화](#)를 참조하십시오.

- 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](#)
- [애플리케이션 CPU 지표](#)
- [애플리케이션 병렬성](#)
- [애플리케이션 로깅](#)
- [연산자 병렬성](#)
- [애플리케이션 로직](#)
- [애플리케이션 메모리](#)

CloudWatch 모니터링 수준

CloudWatch 모니터링 수준이 설정을 너무 자세히 설명하도록 설정되어 있지 않은지 확인하십시오.

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 내용을 알아보려면 [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](#) 섹션 및 [스케일링](#) 섹션을 참조하세요.

Note

애플리케이션은 CPU 사용량에 따라서만 자동으로 규모가 조정됩니다. 애플리케이션은 heapMemoryUtilization와(과) 같은 다른 시스템 지표에 따라 Auto Scaling되지 않습니다. 애플리케이션의 다른 지표 사용량이 높은 경우 애플리케이션의 병렬성을 수동으로 높이세요.

애플리케이션 병렬성

애플리케이션의 병렬성을 높이세요. 작업의 ParallelismConfigurationUpdate 매개 변수를 사용하여 응용 프로그램의 병렬 처리를 업데이트합니다. [UpdateApplication](#)

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 병렬성](#) 섹션을 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”](#) 섹션을 참조하세요.

연산자 병렬성

애플리케이션의 워크로드가 작업자 프로세스 간에 균등하게 분산되어 있는지 확인하세요.

애플리케이션 운영자의 워크로드 조정에 대한 자세한 내용은 [연산자 스케일링](#) 섹션을 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 설명서](#)의 [Async I/O](#)를 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 [스케일링](#) 섹션을 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 4:1입니다. 즉, 애플리케이션에는 사용 가능한 연산자 하위 작업 4개당 작업 슬롯 1개가 있습니다. 리소스를 많이 사용하는 연산자가 있는 애플리케이션에는 3:1 또는 2:1의 비율이 필요할 수 있지만, 리소스를 덜 사용하는 애플리케이션에서는 비율이 10:1이면 안정적일 수 있습니다.

[런타임 속성](#)(들)을 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 리소스](#) 섹션을 참조하세요.

애플리케이션의 작업자 프로세스 전반에 걸친 워크로드 분배를 제어하려면 Parallelism 설정과 KeyBy 파티션 방법을 사용하세요. 자세한 내용은 [Apache Flink 설명서](#)에서 다음 주제를 참조하세요.

- [병렬 실행](#)
- [DataStream 변환](#)

외부 종속성 리소스 사용량 모니터링

대상 (예: Kinesis Streams, Firehose, OpenSearch DynamoDB 또는 서비스) 에 성능 병목 현상이 있는 경우 애플리케이션에 역압이 발생합니다. 외부 종속성이 애플리케이션 처리량에 맞게 적절하게 프로 비저닝되었는지 확인하세요.

Note

다른 서비스에 장애가 발생하면 애플리케이션에 장애가 발생할 수 있습니다. 애플리케이션에 서 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 독립형을](#) 참조하십시오.

모니터링 성능

이 섹션에서는 애플리케이션 성능을 모니터링하는 도구에 대해 설명합니다.

메트릭을 사용한 성능 모니터링 CloudWatch

메트릭을 사용하여 애플리케이션의 리소스 사용량, 처리량, 체크포인트, 다운타임을 모니터링합니다. CloudWatch Apache Flink용 관리 서비스 애플리케이션에서 CloudWatch 메트릭을 사용하는 방법에 대한 자세한 내용은 [을 참조하십시오. Apache Flink용 매니지드 서비스의 메트릭과 디멘션](#)

CloudWatch 로그 및 경보를 사용한 성능 모니터링

CloudWatch 로그를 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 리소스 사용 또는 체크포인트 지표가 안전한 임계값을 초과하거나 예상치 못한 애플리케이션 상태 변경과 같은 성능 문제에 대한 알림을 생성할 수 있습니다.

Apache Flink용 관리 서비스 애플리케이션을 위한 CloudWatch 경보 생성에 대한 자세한 내용은 [여기](#)를 참조하십시오. [경보](#)

아파치 플링크용 매니지드 서비스 및 스튜디오 노트북 할당량

Note

아파치 플링크 커뮤니티는 아파치 플링크 버전 1.6, 1.8, 1.11을 3년 넘게 지원하지 않고 있습니다. 2024년 11월 5일에 아파치 플링크용 아마존 매니지드 서비스에서 이러한 버전을 더 이상 사용하지 않을 계획입니다. 이 날짜부터 이러한 Flink 버전에 대한 새 애플리케이션을 만들 수 없습니다. 지금은 기존 애플리케이션을 계속 실행할 수 있습니다. Apache Flink용 Amazon Managed Service의 인플레이스 버전 업그레이드 기능을 사용하여 애플리케이션을 안전하게 업그레이드할 수 있습니다. 자세한 내용은 [아파치 플링크의 인플레이스 버전 업그레이드](#)

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 내용을 알아보려면 [AWS Support 센터](#) 섹션을 참조하세요.

Managed Service for Apache Flink를 지원하는 지역 목록은 [Managed Service for Apache Flink 지역 및 엔드포인트](#)를 참조하세요.

- Kinesis 처리 단위(KPU)의 수는 기본적으로 64로 제한됩니다. 이 할당량 증가를 요청하는 방법에 대한 지침은 [Service Quotas](#)의 할당량 증가를 요청하려면을 참조하세요. 새 KPU 한도를 적용해야 하는 애플리케이션 접두사를 지정했는지 확인하세요.

Apache Flink용 관리형 서비스를 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 [스냅샷](#)는 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 사례를 생성](#)합니다.

- `websocketMessageSize` = 5MiB
- `noteSize` = 5MiB
- `noteCount` = 1,000
- Max cumulative UDF size = 100MiB
- Max cumulative dependency jar size = 300MiB

Managed Service for Apache Flink 정비

Apache Flink용 Managed Service for Apache Flink는 운영 체제 및 컨테이너 이미지 보안 업데이트로 애플리케이션을 정기적으로 패치하여 규정 준수를 유지하고 보안 목표를 충족합니다. AWS 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 내용을 알아보려면 [the section called “내결합성: 체크포인트 및 세이브포인트”](#) 섹션을 참조하세요.

Apache Flink용 관리형 서비스가 애플리케이션에 대한 유지 관리를 수행하는 기간을 변경하려면 API 를 사용하십시오. [UpdateApplicationMaintenanceConfiguration](#)

리전	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

리전	정비 창
아시아 태평양(도쿄)	13:00~21:00 UTC
Canada (Central)	03:00~11:00 UTC
China (Beijing)	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
Europe (London)	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
Europe (London)	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 Flink 코드에서 모든 운영자의 UUID를 설정하는 것이 좋습니다. 자세한 정보는 [프로덕션 준비](#) 주제의 모든 연산자의 UUID 설정 내용을 참조하세요.

애플리케이션 유지 관리가 언제 발생했는지 확인하십시오.

API를 사용하여 Apache Flink용 관리형 서비스가 애플리케이션에 대한 유지 관리 작업을 수행했는지 확인할 수 있습니다. `ListApplicationOperations`

다음은 애플리케이션의 유지 관리를 위해 목록을 `ListApplicationOperations` 필터링하는 데 도움이 되는 요청 예제입니다.

```
{
  "ApplicationName": "MyApplication",
  "operation": "ApplicationMaintenance"
}
```

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. 응용 프로그램은 정상인 것처럼 보이지만 운영 이벤트로 인해 응용 프로그램의 부하가 크게 증폭되는 경우가 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 하십시오. 예를 들어 Kinesis 데이터 TRIM_HORIZON 스트림의 경우 시작 위치를 사용하십시오.

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 병렬도는 상태가 처음 생성될 때 설정됩니다.

기본적으로 최대 병렬도는 다음과 같이 설정됩니다.

- 병렬 처리가 128 미만인 경우 128
- $\text{MIN}(\text{nextPowerOfTwo}(\text{parallelism} + (\text{parallelism} / 2)), 2^{15})$: 병렬 처리가 128을 초과하는 경우

애플리케이션을 128 병렬 처리 이상으로 확장하려는 경우 최대 병렬도를 명시적으로 정의해야 합니다.

응용 프로그램 수준에서 최대 병렬성을 정의할 수 있습니다 (연산자 하나 또는 단일 연산자 사용). `env.setMaxParallelism(x)` 다르게 지정하지 않는 한 모든 연산자는 응용 프로그램의 최대 병렬성을 상속합니다.

자세한 내용은 Apache Flink [설명서의 최대 병렬 처리 수 설정을](#) 참조하십시오.

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 내용을 알아보려면 [프로덕션 준비 체크리스트](#)를 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

- [내결합성: 체크포인트 및 세이브포인트](#)
- [지원되지 않는 커넥터 버전](#)
- [성능 및 병렬 처리](#)
- [연산자별 병렬 처리 설정](#)
- [로깅](#)
- [코딩](#)
- [보안 인증 관리](#)
- [샤드/파티션이 거의 없는 소스에서 읽기](#)
- [스튜디오 노트북 새로 고침 간격](#)
- [스튜디오 노트북 최적 성능](#)
- [워터마크 전략과 유휴 샤드가 타임윈도우에 미치는 영향](#)
- [모든 연산자에 대해 UUID 설정](#)
- [메이븐 ServiceResourceTransformer 쉘이드 플러그인에 추가](#)

내결합성: 체크포인트 및 세이브포인트

체크포인트와 세이브포인트를 사용하여 Managed Service for Apache Flink 애플리케이션에서 내결합성을 구현하세요. 애플리케이션을 개발하고 유지 관리할 때는 다음 사항에 유의하세요.

- 애플리케이션에 체크포인트를 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 내결합성을 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 SnapshotsEnabled을 true으로 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 [Apache Flink용 매니지드 서비스의 메트릭과 디멘션](#) 섹션을 참조하세요.

내결합성에 대한 자세한 내용은 [내결합성](#) 섹션을 확인하세요.

지원되지 않는 커넥터 버전

Apache Flink 버전 1.15 이상부터 Apache Flink용 관리형 서비스는 애플리케이션 JAR에 번들로 제공되는 지원되지 않는 Kinesis 커넥터 버전을 사용하는 경우 애플리케이션이 시작되거나 업데이트되지 않도록 자동으로 차단합니다. Apache Flink용 관리 서비스 버전 1.15 이상으로 업그레이드할 때는 최신 Kinesis 커넥터를 사용하고 있는지 확인하십시오. 이 버전은 버전 1.15.2와 같거나 더 최신 버전입니다. Apache Flink용 Managed Service for Apache Flink에서는 다른 모든 버전을 지원하지 않습니다. 이 경우 일관성 문제가 발생하거나 Savepoint로 중지 기능에 장애가 발생하여 정상적인 중지/업데이트 작업이 불가능할 수 있기 때문입니다. [Apache Flink용 Amazon 관리 서비스 버전의 커넥터 호환성에 대한 자세한 내용은 Apache Flink 커넥터를 참조하십시오.](#)

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 해당 서비스를 모니터링하십시오. [CloudWatch](#)
- 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 발

생하거나 연산자 데이터 처리에 병목 현상이 발생할 수 있습니다. [SetParallelism](#)을 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 설정하기](#)를 참조하세요.

조정 구현에 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.setParallelism(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 [경보](#) 섹션을 참조하세요.

로깅 구현에 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 전환 될 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 플러그인](#)을 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 Manager](#)에 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](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 [리밸런싱](#)하여 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-second에 number-of-keys를 곱한 값이 25,000,000 미만일 때 가장 좋은 성능을 보였습니다. 이는 15만 미만 events-per-second에 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 폴링 애플리케이션이 이벤트 시간을 사용하기에 충분하지 않습니다. 또한 소스는 소스에서 다른 모든 연산자에게 이벤트 시간에 대한 정보를 전파하는 워터마크를 생성해야 합니다. [Flink 설명서](#)에는 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 최대 값을 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 설명서](#)에서 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 방지하기 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(1 -> 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("XXXXXXXXXXXXXXXXX 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 kineses put-record --stream-name hp-16 --partition-key 1 --data MQ==
$ aws kineses put-record --stream-name hp-16 --partition-key 2 --data Mg==
$ aws kineses 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 0A==
$ 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:
49627894338503151834508157912666084957565273949901684850,}}}' 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:
49627894338458550344111096666383013521019977226889723986,}}}'}, 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:
49627894338458550344111096666383013521019977226889723986,}}}' 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:
85070591730234615865843651857942052864,EndingHashKey:
106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
49627894338436249598912566043241477802747328865383743554,}}}'}, 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:
63802943797675961899382738893456539648,EndingHashKey:
85070591730234615865843651857942052863},SequenceNumberRange: {StartingSequenceNumber:
49627894338413948853714035420099942084474680503877763122,}}}'}, 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:
49627894338681557796096402897798370703746460841949528306,}}'}, 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:
49627894338659257050897872274656834985473812480443547874,}}'}, 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:
85070591730234615865843651857942052864,EndingHashKey:
106338239662793269832304564822427566079},SequenceNumberRange: {StartingSequenceNumber:
49627894338436249598912566043241477802747328865383743554,}}'} 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:
63802943797675961899382738893456539648,EndingHashKey:
85070591730234615865843651857942052863},SequenceNumberRange: {StartingSequenceNumber:
49627894338413948853714035420099942084474680503877763122,}}'} 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:
21267647932558653966460912964485513216,EndingHashKey:
42535295865117307932921825928971026431},SequenceNumberRange: {StartingSequenceNumber:
49627894338369347363316974173816870647929383780865802258,}}'}, 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:
191408831393027885698148216680369618944,EndingHashKey:
212676479325586539664609129644855132159},SequenceNumberRange: {StartingSequenceNumber:
49627894338547753324905219158949156394110570672913645714,}}'}, 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:
21267647932558653966460912964485513215},SequenceNumberRange: {StartingSequenceNumber:
```

```
49627894338347046618118443550675334929656735419359821826,}}'}, 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:
255211775190703847597530955573826158592,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:
21267647932558653966460912964485513216,EndingHashKey:
42535295865117307932921825928971026431},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:
21267647932558653966460912964485513215},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:
42535295865117307932921825928971026432,EndingHashKey:
63802943797675961899382738893456539647},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:
49627894338636956305699341651515299267201164118937567442,}}'}, 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:
49627894338391648108515504796958406366202032142371782690,}}'} 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:
191408831393027885698148216680369618944,EndingHashKey:
212676479325586539664609129644855132159},SequenceNumberRange: {StartingSequenceNumber:
49627894338547753324905219158949156394110570672913645714,}}'} 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:
212676479325586539664609129644855132160,EndingHashKey:
233944127258145193631070042609340645375},SequenceNumberRange: {StartingSequenceNumber:
49627894338570054070103749782090692112383219034419626146,}}'} 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:
319014718988379809496913694467282698240,EndingHashKey:
340282366920938463463374607431768211455},SequenceNumberRange: {StartingSequenceNumber:
49627894338681557796096402897798370703746460841949528306,}}'} 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:
297747071055821155530452781502797185024,EndingHashKey:
319014718988379809496913694467282698239},SequenceNumberRange: {StartingSequenceNumber:
49627894338659257050897872274656834985473812480443547874,}}}' 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:
49627894338525452579706688535807620675837922311407665282,}}}' 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:
49627894338636956305699341651515299267201164118937567442,}}}' 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:
49627894338614655560500811028373763548928515757431587010,}}}' 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:
49627894338592354815302280405232227830655867395925606578,}}}' 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
XXXXXXXXXXXXXXXXX Window with 3 events; Watermark: 1648030809281, 2022-03-23T10:20:09.281Z
3
1
2
XXXXXXXXXXXXXXXXX Window with 2 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
4
5
XXXXXXXXXXXXXXXXX Window with 1 events; Watermark: 1648030834104, 2022-03-23T10:20:34.104Z
6
XXXXXXXXXXXXXXXXX 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 정의된 Maven 쉐이드 플러그인을 추가하는 것이 좋습니다. [ServiceResourceTransformer](#)

```
<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>
```

아파치 플링크 스테이트풀 함수

[상태 저장 함수](#)는 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 Flink 상태 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 구성](#)에 포함되어 있다는 것입니다. 이 구성은 일반적으로 상태 저장 함수 전용 컨테이너에 패키징되어 Kubernetes에 배포됩니다. 하지만 Managed Service for Apache Flink에서는 불가능합니다.

다음은 아파치 플링크용 매니지드 서비스에 대한 StateFun Python 예제를 수정한 것입니다.

아파치 플링크 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.

아파치 플링크 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:

- [아파치 플링크 구성](#)
- [스테이트 백엔드](#)
- [체크포인트](#)
- [세이브포인팅](#)
- [힙 크기](#)
- [버퍼 디블로팅](#)
- [수정 가능한 Flink 구성 속성](#)
- [구성된 Flink 속성 보기](#)

아파치 플링크 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 [구성](#)을 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 센터](#)를 참조하세요. [Apache Flink 대시보드](#)를 사용하여 애플리케이션의 현재 구성을 확인할 수 있습니다.

스테이트 백엔드

Managed Service for Apache Flink는 임시 데이터를 상태 백엔드에 저장합니다. 아파치 플링크용 매니지드 서비스는 RockSDB를 사용합니다. StateBackend 다른 백엔드를 설정하기 위해 `setStateBackend`(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 설명서](#)에서 [Flink 1.8.0의 State TTL](#)을 참조하세요.

상태 백엔드에 대한 자세한 내용은 [Apache Flink 설명서의 상태 백엔드](#)를 참조하세요.

체크포인트

Managed Service for Apache Flink는 다음 값이 포함된 기본 체크포인트 구성을 사용합니다. 이 값 중 일부는 변경할 수 있습니다. [설정해야 합니다. CheckpointConfiguration ConfigurationType](#) Apache CUSTOM Flink용 관리 서비스가 수정된 체크포인트 값을 사용하도록 하려면 로 설정하십시오.

설정	수정할 수 있나요?	수정할 수 있다면 방법이 무엇입니까?	기본 값
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?	기본 값
		AWS CloudFormation	
정렬되지 않은 체크포인트	수정 가능	지원 사례	False
동시 체크포인트 수	수정 불가능	N/A	1
체크포인트 모드	수정 불가능	N/A	딱 한 번
체크포인트 보존 정책	수정 불가능	N/A	실패 시
체크포인트 타임아웃	수정 불가능	N/A	60분
유지된 최대 체크포인트	수정 불가능	N/A	1
재시작 전략	수정 불가능	N/A	10초마다 무한 재시도가 가능한 딜레이를 수정했습니다.
체크포인트 및 세이브포인트 위치	수정 불가능	N/A	내구성이 뛰어난 체크포인트와 세이브포인트 데이터를 서비스 소유의 S3 버킷에 저장합니다.
상태 백엔드 메모리 임계값	수정 불가능	N/A	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. 애플리케이션의 `AllowNonRestoredState` 매개변수를 로 설정하여 작업이 성공하도록 허용할 수 있습니다. [FlinkRunConfiguration](#) `true` 이렇게 하면 재개 작업이 새 프로그램에 매핑할 수 없는 상태를 건너뛰도록 할 수 있습니다.

자세한 내용을 알아보려면 [Apache Flink의 설명서](#)에서 [복원되지 않은 상태 허용](#)을 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”](#) 섹션을 참조하세요.

JVM 힙 크기에 대한 자세한 내용은 [Apache Flink 설명서](#)의 [구성](#)을 참조하세요.

버퍼 디블로팅

버퍼 디블로팅은 배압이 높은 애플리케이션에 도움이 될 수 있습니다. 애플리케이션에 체크포인트/세이브포인트에 장애가 발생하는 경우 이 기능을 활성화하면 유용할 수 있습니다. 이를 위해서는 [지원 사례](#)를 요청하세요.

자세한 내용은 [Apache Flink 설명서](#)의 [버퍼 디블로팅 메커니즘](#)을 참조하세요.

수정 가능한 Flink 구성 속성

다음은 [지원 사례](#)를 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 메트릭

RocksDB 네이티브 메트릭은 배송되지 않습니다. CloudWatch 활성화되면 Flink 대시보드 또는 사용자 지정 도구를 사용하여 Flink REST API에서 이러한 지표에 액세스할 수 있습니다.

Apache Flink용 관리형 서비스를 통해 고객은 API를 사용하여 최신 Flink [REST API](#) (또는 사용 중인 지원되는 버전) 에 읽기 전용 모드로 액세스할 수 있습니다. [CreateApplicationPresignedUrl](#) 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 옵션 TaskManager

`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 예약 옵션

`slot.request.timeout:`

플링크 웹 UI의 고급 옵션

`web.timeout:`

구성된 Flink 속성 보기

Apache Flink Dashboard를 통해 다음 단계에 따라 [지원 사례](#)를 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.

Apache Flink가 Amazon VPC의 리소스에 액세스할 수 있도록 관리형 서비스 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](#)
- [Apache Flink용 VPC 연결 관리 서비스 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:

- 가상 사설 클라우드 (VPC) 는 사용자 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 사용하면 인터넷 게이트웨이, NAT 디바이스, VPN 연결 또는 연결 PrivateLink 없이 VPC를 지원하는 서비스 AWS 및 VPC 엔드포인트 서비스에 비공개로 연결할 수 있습니다. AWS Direct Connect 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 내용을 알아보려면 [Amazon Virtual Private Cloud 사용자 가이드](#)를 참조하세요.

Managed Service for Apache Flink는 애플리케이션의 VPC 구성에 제공된 서브넷 중 하나에서 [탄력적 네트워크 인터페이스](#)를 생성합니다. VPC 서브넷에서 생성되는 탄력적 네트워크 인터페이스의 수는 애플리케이션의 KPU당 병렬 처리 및 병렬성에 따라 달라질 수 있습니다. 애플리케이션 조정에 대한 자세한 내용을 알아보려면 [스케일링](#) 섹션을 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 내용을 알아보려면 [Amazon Managed Service for Apache Flink의 자격 증명 및 액세스 관리](#) 섹션을 참조하세요.

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

콘솔 (예: CloudWatch Logs 또는 Amazon VPC) 을 사용하여 애플리케이션 리소스를 지정하면 콘솔은 애플리케이션 실행 역할을 수정하여 해당 리소스에 액세스할 권한을 부여합니다. 콘솔을 사용하지 않고 애플리케이션을 생성하는 경우에는 애플리케이션의 실행 역할을 수동으로 수정만 하면 됩니다.

Apache Flink용 VPC 연결 관리 서비스 애플리케이션을 위한 인터넷 및 서비스 액세스

기본적으로 계정에서 VPC에 Managed Service for Apache Flink 애플리케이션을 연결할 때 VPC가 액세스 권한을 제공하지 않는 한 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 엔드포인트](#)를 제공합니다. VPC 엔드포인트를 사용하여 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/flink>에서 Managed Service for Apache Flink 콘솔을 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 내용을 알아보려면 [API 예제 코드](#) 섹션을 참조하십시오.

애플리케이션 생성

[CreateApplication](#) 작업을 사용하면 생성 중에 애플리케이션에 VPC 구성을 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

VPC 구성이 생성된 후 이 [AddApplicationVpcConfiguration](#) 작업을 사용하여 애플리케이션에 VPC 구성을 추가할 수 있습니다.

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"  
}
```

애플리케이션 업데이트

[UpdateApplication](#) 작업을 사용하면 애플리케이션의 모든 VPC 구성을 한 번에 업데이트할 수 있습니다.

UpdateApplication 작업을 위한 다음 예 요청 코드는 애플리케이션의 모든 VPC 구성을 업데이트합니다.

```
{
  "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 복제](#) 섹션을 참조하십시오.

Managed Service for Apache Flink 문제 해결

다음은 Amazon Managed Service for Apache Flink와 관련하여 발생할 수 있는 문제를 해결하는 데 도움이 되는 내용입니다.

주제

- [개발 문제 해결](#)
- [런타임 문제 해결](#)

개발 문제 해결

주제

- [FlinkRuntimeException: “허용되지 않은 구성 변경이 감지되었습니다.”](#)
- [시스템 롤백 모범 사례](#)
- [Hudi 구성 모범 사례](#)
- [Apache Flink 플레임 그래프](#)
- [EFO 커넥터 1.15.2의 자격 증명 공급자 문제](#)
- [지원되지 않는 Kinesis 커넥터가 있는 애플리케이션](#)
- [컴파일 오류: “프로젝트에 대한 종속성을 해결할 수 없습니다.”](#)
- [잘못된 선택: “kinesisanalyticsv2”](#)
- [UpdateApplication 액션은 애플리케이션 코드를 다시 로드하는 것이 아닙니다.](#)
- [S3 StreamingFileSink FileNotFoundExceptions](#)
- [FlinkKafkaConsumer 세이브 포인트 사용 중지 관련 문제](#)
- [Flink 1.15 Async Sink 교착 상태](#)
- [리샤딩 중에 Amazon Kinesis 데이터 스트림의 소스 처리 순서가 잘못되었습니다.](#)

FlinkRuntimeException: “허용되지 않은 구성 변경이 감지되었습니다.”

Flink 버전 1.19 이상부터 Flink 작업 코드를 사용하여 Flink 작업 구성 변경을 비활성화했습니다.

이러한 지원되지 않는 구성 변경을 수행하면 다음과 같은 예외가 발생합니다.

```
Caused by: org.apache.flink.client.program.ProgramInvocationException: The main method
  caused an error: Not allowed configuration change(s) were detected:
  - Configuration execution.checkpointing.unaligned.enabled:true not allowed.
  - Setter CheckpointConfig#setCheckpointStorage has been used
    at
  org.apache.flink.client.program.PackagedProgram.callMainMethod(PackagedProgram.java:372)
    at
  org.apache.flink.client.program.PackagedProgram.invokeInteractiveModeForExecution(PackagedProgram.java:108)
    at org.apache.flink.client.ClientUtils.executeProgram(ClientUtils.java:108)
    at
  org.apache.flink.client.deployment.application.DetachedApplicationRunner.tryExecuteJobs(DetachedApplicationRunner.java:108)
  ... 9 more
```

예외 메시지는 지금까지 사용한 허용되지 않은 구성 또는 세터를 나타냅니다. 이전 예제에서는 응용 프로그램이 구성 `execution.checkpointing.unaligned.enabled` 및 사용을 `setCheckpointStorage` 시도합니다.

코드에서 계속 수정할 수 있는 구성은 다음과 같습니다.

1. 파이프라인. `auto-watermark-interval`
2. 파이프라인. `closure-cleaner-level`
3. 파이프라인. `max-병렬 처리`
4. 파이프라인 이름
5. 파이프라인. 오퍼레이터 체인. `chain-operators-with-different-최대 병렬성`
6. 파이프라인. 오퍼레이터 체이닝 활성화
7. 파이프라인. `vertex-description-mode`
8. 파이프라인. `vertex-name-include-index-접두사`
9. 파이썬 실행 모드
10. 파이썬 오퍼레이터 체이닝 활성화

여전히 지원 사례를 사용하여 일부 작업 또는 클러스터 구성에 대한 수정을 요청할 수 있습니다. 자세한 내용은 [수정 가능한 Flink 구성 속성](#)을 참조하십시오.

시스템 롤백 모범 사례

Apache Flink용 Amazon Managed Service의 자동 시스템 롤백 및 운영 가시성 기능을 사용하면 애플리케이션 관련 문제를 식별하고 해결할 수 있습니다.

시스템 롤백

코드 버그 또는 권한 문제와 같은 고객 오류로 인해 애플리케이션 업데이트 또는 조정 작업이 실패할 경우, Amazon Managed Service for Apache Flink는 사용자가 이 기능을 선택한 경우 이전 실행 버전으로 자동으로 롤백을 시도합니다. 자세한 정보는 [Apache Flink용 관리형 서비스 애플리케이션의 시스템 롤백 활성화](#)를 참조하세요. 자동 롤백이 실패하거나 사용자가 옵트인하거나 옵트아웃하지 않은 경우 애플리케이션이 해당 상태로 전환됩니다. READY 신청서를 업데이트하려면 다음 단계를 완료하세요.

수동 롤백

애플리케이션이 계속 진행되지 않고 오랫동안 일시적인 상태에 있거나 애플리케이션이 성공적으로 전환되었지만 성공적으로 업데이트된 Flink 애플리케이션에서 처리 오류와 같은 다운스트림 문제가 발생하는 경우 API를 사용하여 수동으로 롤백할 수 있습니다. `Running RollbackApplication`

1. 호출 `RollbackApplication` - 이렇게 하면 이전 실행 버전으로 되돌아가고 이전 상태로 복원됩니다.
2. API를 사용하여 롤백 작업을 모니터링합니다. `DescribeApplicationOperation`
3. 롤백이 실패할 경우 이전 시스템 롤백 단계를 사용하십시오.

운영 가시성

`ListApplicationOperationsAPI`는 애플리케이션의 모든 고객 및 시스템 운영 기록을 보여줍니다.

1. 목록에서 실패한 작업의 `OperationID`를 가져옵니다.
2. 전화를 걸어 상태 `DescribeApplicationOperation` 및 상태설명을 확인하십시오.
3. 작업이 실패한 경우 설명에는 조사해야 할 잠재적 오류가 나와 있습니다.

일반적인 오류 코드 버그: 롤백 기능을 사용하여 마지막 작업 버전으로 되돌릴 수 있습니다. 버그를 해결하고 업데이트를 다시 시도하세요.

권한 문제: `DescribeApplicationOperation` 를 사용하여 필요한 권한을 확인하십시오. 애플리케이션 권한을 업데이트하고 다시 시도하세요.

Apache Flink용 Amazon 관리형 서비스 문제: 지원 사례를 AWS Health Dashboard 확인하거나 여십시오.

Hudi 구성 모범 사례

Apache Flink용 관리 서비스에서 Hudi 커넥터를 실행하려면 다음과 같이 구성을 변경하는 것이 좋습니다.

hoodie.embed.timeline.server 비활성화

Flink의 Hudi 커넥터는 작업 병렬성이 높을 때 성능을 향상시키기 위해 Flink jobmanager (JM) 에 내장된 타임라인 (TM) 서버를 설정하여 메타데이터를 캐시합니다. Apache Flink용 Managed Service for Apache Flink에서는 이 내장 서버를 사용하지 않도록 설정하는 것이 좋습니다. JM과 TM 간의 비 Flink 통신은 비활성화되기 때문입니다.

이 서버가 활성화된 경우 Hudi writes는 먼저 JM의 임베디드 서버에 연결을 시도한 다음 Amazon S3에서 메타데이터를 읽습니다. 즉, Hudi로 인해 연결 시간 초과가 발생하여 Hudi 쓰기가 지연되고 Apache Flink용 관리형 서비스의 성능이 저하됩니다.

Apache Flink 플레임 그래프

플레임 그래프는 플레임 그래프를 지원하는 Managed Service for Apache Flink버전의 애플리케이션에서 기본적으로 활성화됩니다. [Flink 설명서](#)에 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 내용을 알아보려면 [AWS 지원 센터](#)를 참조하세요.

EFO 커넥터 1.15.2의 자격 증명 공급자 문제

Kinesis 데이터 스트림 EFO 커넥터 버전 최대 1.15.2에는 FlinkKinesisConsumer가 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

Apache Flink용 관리형 서비스 Flink용 Apache Flink 버전 1.15 이상에서는 [애플리케이션 JAR 또는 아카이브 \(ZIP\) 에 번들로 제공되는 지원되지 않는 Kinesis Connector 버전 \(1.15.2 이전 버전\) 을 사용하는 경우 애플리케이션의 시작 또는 업데이트가 자동으로 거부됩니다.](#)

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-kinesis에 대한 애플리케이션 종속성을 업데이트하십시오. Maven을 프로젝트의 빌드 도구로 사용하고 있다면 [Maven 종속성 업데이트](#) 을 따르세요. Gradle을 사용하고 있다면 [Gradle 종속성 업데이트](#) 을 따르세요.
- 애플리케이션을 재패키징합니다.
- Amazon S3 버킷에 업로드합니다.
- 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-kinesis`을 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: “프로젝트에 대한 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.html>)의 버전 1.8.2)를 다운로드해야 합니다. Apache Flink 소스 코드를 다운로드, 컴파일 및 설치하는 방법에 대한 지침은 [the section called “아파치 플링크 Kinesis Streams 커넥터를 이전 아파치 플링크 버전과 함께 사용하기”](#) 섹션을 참조하십시오.

잘못된 선택: “kinesisanalyticsv2”

Managed Service for Apache Flink API의 v2를 사용하려면 최신 버전 AWS Command Line Interface (AWS CLI)이 필요합니다.

[업그레이드에 대한 자세한 내용은 사용 설명서의 AWS CLI 설치를 참조하십시오. AWS Command Line Interface](#)

UpdateApplication 액션은 애플리케이션 코드를 다시 로드하는 것이 아닙니다.

지정된 S3 객체 버전이 없는 경우 [UpdateApplication](#) 작업은 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 내용을 알아보려면 [버전 관리 활성화 또는 비활성화](#)를 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 FileNotFoundException으로 실행될 수 있습니다. 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 [파일 시스템에서 지원하는 파일 시스템에 레코드를 기록합니다](#). 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.

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 서비스 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 CreateSnapshot을 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_SNAPSHOT을 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 Apache Flink용 관리형 서비스 버전 1.15에서 수정되었습니다. 애플리케이션을 Apache Flink용 관리형 서비스의 최신 버전으로 마이그레이션하여 재발을 방지하세요. 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 스케일링 중에 중단될 수 있습니다. 이 문제에 대해 게시된 수정 사항이 없으므로 이 [문제를 KafkaSource](#) 완화하려면 새 버전으로 업그레이드하는 것이 좋습니다.

스냅샷이 활성화된 상태로 FlinkKafkaConsumer를 사용하는 경우 Flink 작업에서 저장점 API로 중지 요청을 처리할 때 `ClosedException`를 보고하는 런타임 오류가 발생하여 FlinkKafkaConsumer이 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 [버그](#)로 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](#)
- [Python 애플리케이션 업데이트](#)

Java 애플리케이션 업데이트

아래 절차에 따라 Java 애플리케이션을 업데이트하십시오.

`flink-connector-kinesis`

애플리케이션은 `flink-connector-kinesis`을 사용하는 경우:

Kinesis 커넥터는 셰이딩을 사용하여 AWS SDK를 비롯한 일부 종속성을 커넥터 jar에 패키징합니다. AWS SDK 버전을 업데이트하려면 다음 절차를 사용하여 이러한 셰이딩된 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 재배포 규칙을 추가하여 업데이트된 AWS SDK 클래스를 패키지 키지로 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 음영처리된 AWS SDK 클래스를 제외하십시오.
 - b. 업데이트된 AWS SDK 클래스를 Kinesis 커넥터가 예상하는 패키지로 재배포합니다.

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("software/amazon/kinesis/shaded/software/amazon/awssdk/**/*")
    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

SDK BOM (재료 명세서) 에 플랫폼 종속성을 추가하여 프로젝트에 AWS 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 요건](#)과 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 SDK 버전을 지정하여 커넥터 jar를 컴파일하고 설치합니다. AWS 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 SDK 버전을 지정하여 커넥터 jar를 컴파일하고 설치합니다. AWS 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-키네시스-파이어호스

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 디렉터리로 이동합니다.

```
cd flink-1.15.4/flink-connectors/flink-connector-aws-kinesis-firehose/
```


4. 필요한 SDK 버전을 지정하여 커넥터 jar를 컴파일하고 설치합니다. AWS 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 디렉터리로 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.

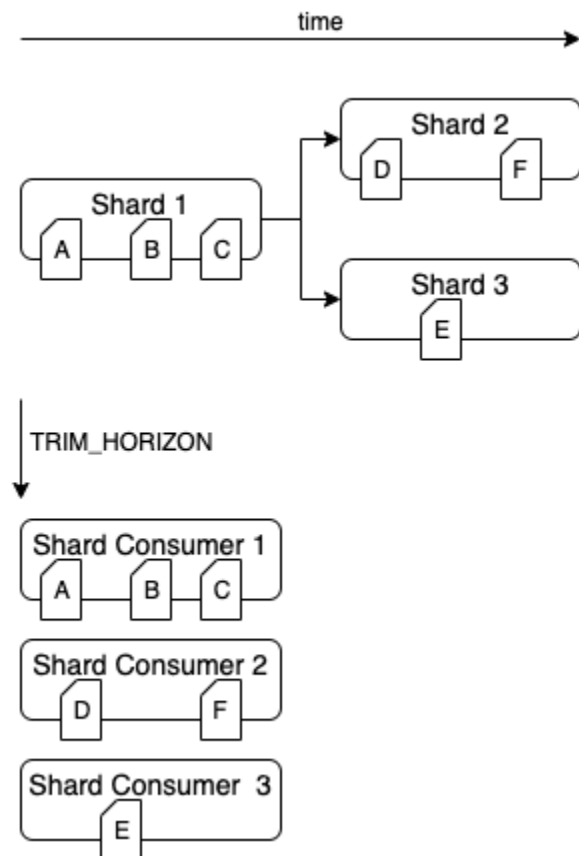
```
mvn clean install -DskipTests -Dfast -Dflink.version=1.15.4 -  
Daws.sdk.version=2.20.144
```

5. 결과 jar는 다음에서 확인할 수 있습니다.

```
flink-sql-connector-dynamodb/target/flink-sql-connector-dynamodb-3.0.0.jar
```

리샤딩 중에 Amazon Kinesis 데이터 스트림의 소스 처리 순서가 잘못되었습니다.

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 문제](#)입니다. 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 재시작 중](#)
- [처리량이 너무 느림](#)
- [무한한 국가 성장](#)
- [I/O 바운드 연산자](#)
- [Kinesis 데이터 스트림의 업스트림 또는 소스 조절](#)
- [체크포인트](#)
- [체크포인트 시간이 초과되었습니다.](#)
- [Apache Beam 애플리케이션의 체크포인트 실패](#)
- [역압](#)
- [데이터 편중](#)
- [상태 편중](#)
- [여러 지역의 리소스와 통합](#)

문제 해결 도구

애플리케이션 문제를 탐지하는 기본 도구는 CloudWatch 알람입니다. CloudWatch 경보를 사용하면 애플리케이션의 오류 또는 병목 상태를 나타내는 CloudWatch 지표에 대한 임계값을 설정할 수 있습니다. 권장 CloudWatch 경보에 대한 자세한 내용은 [아마존 매니지드 서비스에서 아파치 플링크용 CloudWatch 알람 사용하기](#)

애플리케이션 문제

이 섹션에는 Managed Service for Apache Flink 애플리케이션에서 발생할 수 있는 오류 조건에 대한 솔루션이 포함되어 있습니다.

주제

- [애플리케이션이 일시적 상태에서 멈췄습니다.](#)
- [스냅샷 생성 실패](#)
- [VPC의 리소스에 액세스할 수 없음](#)
- [Amazon S3 버킷에 쓸 때 데이터가 손실됩니다.](#)
- [애플리케이션이 RUNNING 상태이지만 데이터를 처리하지 않고 있습니다.](#)
- [스냅샷, 애플리케이션 업데이트 또는 애플리케이션 중지 오류: `InvalidApplicationConfigurationException`](#)
- [`java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/보안/cacerts`](#)

애플리케이션이 일시적 상태에서 멈췄습니다.

애플리케이션이 일시적 상태 (STARTING, UPDATINGSTOPPING, 또는AUTOSCALING) 에 있는 경우 Force 매개변수가 로 설정된 [StopApplication](#) 액션을 사용하여 애플리케이션을 중지할 수 있습니다. true DELETING 상태에서는 애플리케이션을 강제로 중지할 수 없습니다. 또는 애플리케이션이 UPDATING 또는 AUTOSCALING 상태인 경우 이전 실행 버전으로 회귀할 수 있습니다. 애플리케이션을 회귀시키면 마지막으로 성공한 스냅샷의 상태 데이터가 로드됩니다. 애플리케이션에 스냅샷이 없는 경우 Managed Service for Apache Flink는 회귀 요청을 거부합니다. 애플리케이션 롤백에 대한 자세한 내용은 [RollbackApplication](#) 액션을 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 실패](#)를 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 복원에 성공하려면 응용 프로그램의 AllowNonRestoredState 매개 변수를 [FlinkRunConfiguration](#)로 true 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 정보는 [스냅샷](#)을 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 false로 설정하고 요청을 재시도하십시오.

애플리케이션 연산자가 제대로 프로비저닝되지 않은 경우 스냅샷이 실패할 수 있습니다. 연산자 성능 조정에 대한 자세한 설명은 [연산자 스케일링](#)을 참조하십시오.

애플리케이션이 정상 상태로 돌아오면 애플리케이션 SnapshotsEnabled 속성을 true로 설정하는 것이 좋습니다.

VPC의 리소스에 액세스할 수 없음

애플리케이션이 Amazon VPC에서 실행되는 VPC를 사용하는 경우, 다음을 수행하여 애플리케이션이 해당 리소스에 액세스할 수 있는지 확인하십시오:

- CloudWatch 로그에서 다음 오류를 확인하세요. 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 내용은 [로그 및 모니터링](#)을 참조하십시오.

Amazon S3 버킷에 쓸 때 데이터가 손실됩니다.

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. 아파치 플링크 1.6.2를 사용하여 Amazon S3 버킷에 쓰려면 Firehose를 사용하는 것이 좋습니다. Apache Flink용 관리형 서비스와 함께 Firehose를 사용하는 방법에 대한 자세한 내용은 [Firehose 싱크](#)를 참조하십시오.

애플리케이션이 RUNNING 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 정보는 [애플](#)

[리케이션 CloudWatch 로깅 옵션 사용](#)을 참조하세요. 로그 스트림에는 애플리케이션 문제를 해결하는데 사용할 수 있는 메시지가 포함됩니다.

스냅샷, 애플리케이션 업데이트 또는 애플리케이션 중지 오류:
InvalidApplicationConfigurationException

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. Apache Flink용 관리 서비스 콘솔에서 또는 작업의 SnapshotsEnabledUpdate 매개 변수를 사용하여 이 작업을 수행할 수 있습니다.

[UpdateApplication](#)

- 스냅샷을 생성할 수 없는 이유를 조사하세요. 자세한 설명은 [애플리케이션이 일시적 상태에서 멈췄습니다](#). 섹션을 참조하세요.
- 애플리케이션이 정상 상태로 돌아오면 스냅샷을 다시 활성화하십시오.

java.nio.file.NoSuchFileException: /usr/local/openjdk-8/lib/보안/cacerts

SSL 신뢰 저장소의 위치는 이전 배포에서 업데이트되었습니다. `ssl.truststore.location` 파라미터에 대해 다음 값을 대신 사용하세요.

```
/usr/lib/jvm/java-11-amazon-corretto/lib/security/cacerts
```

애플리케이션 재시작 중

사용하시는 애플리케이션이 정상이지 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 FAILED로의 상태 변경을 포함합니다. 다음 CloudWatch Logs Insights 쿼리를 사용하여 애플리케이션 로그에서 이러한 상태 변경을 쿼리할 수 있습니다. [오류 분석: 애플리케이션 작업 관련 장애](#)

원인 및 해결 방법

다음과 같은 상태 하에서 애플리케이션은 불안정해지고 반복적으로 다시 시작될 수 있습니다.

- 운영자가 예외를 발생시키는 경우: 응용 프로그램에서 연산자의 예외가 처리되지 않으면 운영자가 오류를 처리할 수 없다고 해석하여 응용 프로그램이 페일오버됩니다. “정확히 한 번” 처리하라는 의미를 유지하기 위해 애플리케이션이 최근의 체크포인트에서 다시 시작됩니다. 따라서 이러한 재시작 기간 동안에는 Downtime의 값이 0이 아닙니다. 이러한 일이 발생하지 않도록 하려면 애플리케이션 코드에서 재시도 가능한 예외를 모두 처리하는 것이 좋습니다.

애플리케이션 로그를 검색하여 애플리케이션 상태를 RUNNING에서 FAILED로 변경할 것을 요청함으로써 이 상태의 원인을 조사할 수 있습니다. 자세한 내용을 알아보려면 [the section called “오류 분석: 애플리케이션 작업 관련 장애”](#) 섹션을 참조하세요.

- Kinesis 데이터 스트림이 제대로 프로비저닝되지 않음: 애플리케이션의 소스 또는 싱크가 Kinesis 데이터 스트림인 경우 스트림의 [메트릭에](#) 오류가 있는지 확인하십시오. ReadProvisionedThroughputExceeded WriteProvisionedThroughputExceeded

이러한 오류가 표시되면 스트림의 샤드 수를 늘려 Kinesis 스트림의 가용 처리량을 늘릴 수 있습니다. 자세한 내용을 알아보려면 [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 대상) 가 제대로 프로비저닝되었는지, 읽기 또는 쓰기 제한이 발생하지 않거나 주기적으로 사용할 수 없는지 확인하십시오.

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 내용을 알아보려면 [연산자 스케일링을 적절하게 관리](#) 섹션을 참조하십시오.
- 응용 프로그램 실패 `DaemonException`: 이 오류는 Apache Flink 1.11 이전 버전을 사용하는 경우 응용 프로그램 로그에 나타납니다. KPL 0.14 또는 그 이후 버전을 사용하려면 Apache Flink를 이후 버전으로 업그레이드해야 할 수 있습니다.
- 또는 로 `TimeoutException` 응용 프로그램이 실패합니다 `RemoteTransportException`. 작업 관리자가 충돌하는 경우 응용 프로그램 로그에 이러한 오류가 나타날 수 있습니다. `FlinkException` 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 확인해 CPU 또는 메모리 사용량이 비정상적으로 급증했는지 확인하세요.
- 애플리케이션에서 처리량 문제를 확인하십시오. 자세한 내용을 알아보려면 [성능 문제 해결](#) 섹션을 참조하세요.
- 애플리케이션 로그에서 애플리케이션 코드에서 발생하는 처리되지 않은 예외가 있는지 확인하십시오.
- Not Found 오류로 인해 애플리케이션이 실패함: 애플리케이션에서 Apache Beam을 사용하지만 종속성 또는 종속성 버전이 올바르게 JaxbAnnotationModule 않은 경우 이 오류가 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 내용을 알아보려면 [아파치 플링크용 매니지드 서비스와 CloudFormation 함께 사용](#) 섹션을 참조하세요.

- `Java.io.IOException`에서 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.

- 귀하의 하위 작업 및 네트워크 버퍼별로 더 많은 메모리가 할당되도록 더 낮은 값의 `parallelismPerKpu`를 구성할 수 있습니다. `parallelismPerKpu`를 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 모니터링](#)을 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 재시작 중](#) 섹션을 참조하세요.
- 처리량 지연이 일정하다면 애플리케이션이 성능을 위해 최적화되었는지 확인하세요. 애플리케이션 성능 최적화에 대한 자세한 설명은 [성능 문제 해결](#)을 참조하십시오.
- 처리량 지연이 급증하지 않고 계속 증가하고 있으며 애플리케이션이 성능에 맞게 최적화되어 있다면 애플리케이션 리소스를 늘려야 합니다. 애플리케이션 리소스 증가에 대한 자세한 설명은 [스케일링](#)을 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 구성](#)에서 높은 잠복 환경 섹션을 참조하십시오.

애플리케이션 소스의 느린 처리량 또는 소비자 지연 증가에 대한 문제 해결 단계는 [성능 문제 해결](#) 섹션을 참조하십시오.

무한한 국가 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 해결책

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 내용은 Apache [Flink 설명서의 상태 유지 시간 \(TTL\)](#) 을 참조하십시오.
- 귀하는 지금 Apache Beam 버전 2.25.0 이상을 사용하는 애플리케이션을 실행하고 있습니다. 주요 실험과 가치를 [확장하여 새 버전의 읽기 변환을 사용하지 않도록 선택할 수 있습니다](#) [BeamApplicationProperties](#). use_deprecated_read 자세한 내용을 알아보려면 [Apache Beam 설명서](#)를 참조하세요.

간혹 애플리케이션은 상태 크기가 계속 커지는 상황에 직면하고 있는데, 이는 장기적으로 지속 가능하지 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. 중분 집계가 가능한 를 사용하지 않는 한 [AggregateFunctions](#)Flink는 전체 창 의 이벤트를 상태로 유지해야 합니다.

또한 프로세스 함수를 사용하여 맞춤 연산자를 구현하는 경우 애플리케이션은 더 이상 비즈니스 로직에 필요하지 않은 데이터를 상태로부터 제거해야 합니다. 이 경우 [상태를](#) 사용하여 처리 시간에 따라 데이터를 자동으로 에이징 time-to-live 아웃할 수 있습니다. Managed Service for Apache Flink는 중분 체크포인트를 사용하므로 상태 존속 시간은 [RocksDB 압축](#)에 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 경우 처리량의 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 [AsyncIO](#)를 사용하여 기능을 구현하는 것이 좋습니다.

또한 I/O 바인딩 연산자가 있는 애플리케이션의 경우 Apache Flink용 관리 서비스 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 늘리면 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 수는 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.

`CloudWatch ReadProvisionedThroughputExceeded` 지표를 확인하고 이 지표가 0보다 큰 장기간 또는 지속 기간을 확인하면 이를 확인할 수 있습니다.

계속되는 오류를 관찰하여 Apache Flink용 Amazon 관리형 서비스 애플리케이션의 CloudWatch 로그에서도 이를 확인할 수 있습니다. `LimitExceededException`

해결 방법: 다음 두 가지 중 하나를 수행하여 이 시나리오를 해결할 수 있습니다.

- 통화당 불러오는 레코드 수의 기본 한도를 낮추십시오. `GetRecords`
- Apache Flink용 Amazon 매니지드 서비스 애플리케이션에서 적응형 읽기를 활성화하십시오. 적응형 읽기 기능에 대한 자세한 내용을 알아보려면 [SHARD_USE_ADAPTIVE_READS](#)를 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 이 단계는 짧으며 단순히 하위 작업으로 승인을 보내고 커밋 메시지를 수행하는 것입니다 (예: Kafka 싱크 사용). `JobManager`

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 탭](#)으로 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 모니터링](#)을 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 체크포인트하는 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.

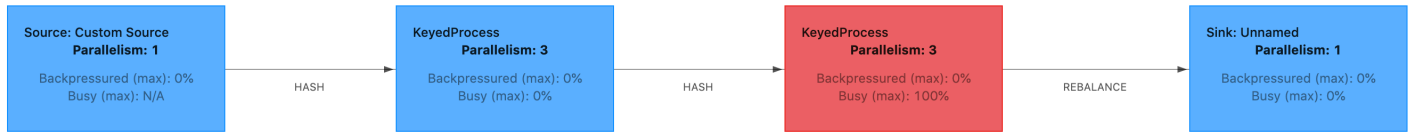
비동기 지속 시간이 길면 상태를 Amazon S3에 업로드하는 데 많은 시간이 소요되고 있음을 알 수 있습니다. 이는 상태가 크거나 업로드되는 상태 파일이 많은 경우 발생할 수 있습니다. 이런 경우에는 애플리케이션이 상태를 어떻게 사용하는지 조사하고 가능한 경우 Flink 네이티브 데이터 구조가 사용되고 있는지 확인하는 것이 좋습니다 ([관련 상태 사용](#)). Managed Service for Apache Flink는 Amazon S3 호출 횟수를 최소화하는 방식으로 Flink를 구성하여 호출이 너무 오래 걸리지 않도록 합니다. 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 운영자가 바쁜 상황의 예입니다. JobGraph KeyedProcess



Flink Flame Graphs나 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 상

태인 JVM 스레드 덤프를 사용합니다. 스레드 상태는 스레드의 스택 트레이스와 몇 가지 추가 정보로 표시됩니다. 실제로 프레임 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(KeyedProcessOperator
  at app//org.apache.flink.streaming.runtime.tasks.OneInputStreamTask
$StreamTaskNetworkOutput.emitRecord(OneInputStreamTask.java:205)
  at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(AbstractStreamTask
  at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(AbstractStreamTask
  at app//
org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOneInputProcessor
  ...
```

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 이름 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. 오퍼레이터 스레드는 오퍼레이터와 이름이 같기 때문에 오퍼레이터 스레드는 이름으로 식별할 수 있습니다. 예를 들어, 오퍼레이터 스레드는 KeyedProcess (1/3) #0 스레드가 KeyedProcess 오퍼레이터의 스레드와 첫 번째 (3개 중) 하위 태스크의 스레드와 관련이 있는 하위 태스크를 나타냅니다.

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).

어떤 Task Manager를 사용할지 결정하기 위해 특정 연산자를 선택할 때 TaskManagers 탭의 덤프를 사용할 수 있습니다. 이는 연산자가 연산자의 여러 하위 작업에서 실행되고 있으며 다른 작업 관리자에 서 실행될 수 있음을 나타냅니다.

Host	LOG	Bytes received	Records received	Bytes sent	Records sent	Status
ip-142-151-131-22:61 21	LOG	936 B	0	0 B	0	RUNNING
ip-142-151-146-195:6 121	LOG	103 KB	1,423	71.1 KB	1,422	RUNNING

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 추적은 KeyedProcess 운영자가 제공한 것이며 첫 번째 하위 작업입니다.

```
"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(KeyedProcessOperator
  at app//org.apache.flink.streaming.runtime.tasks.OneInputStreamTask
$StreamTaskNetworkOutput.emitRecord(OneInputStreamTask.java:205)
  at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.processElement(AbstractStreamTask
  at app//
org.apache.flink.streaming.runtime.io.AbstractStreamTaskNetworkInput.emitNext(AbstractStreamTask
  at app//
org.apache.flink.streaming.runtime.io.StreamOneInputProcessor.processInput(StreamOneInputProcessor
  ...
```

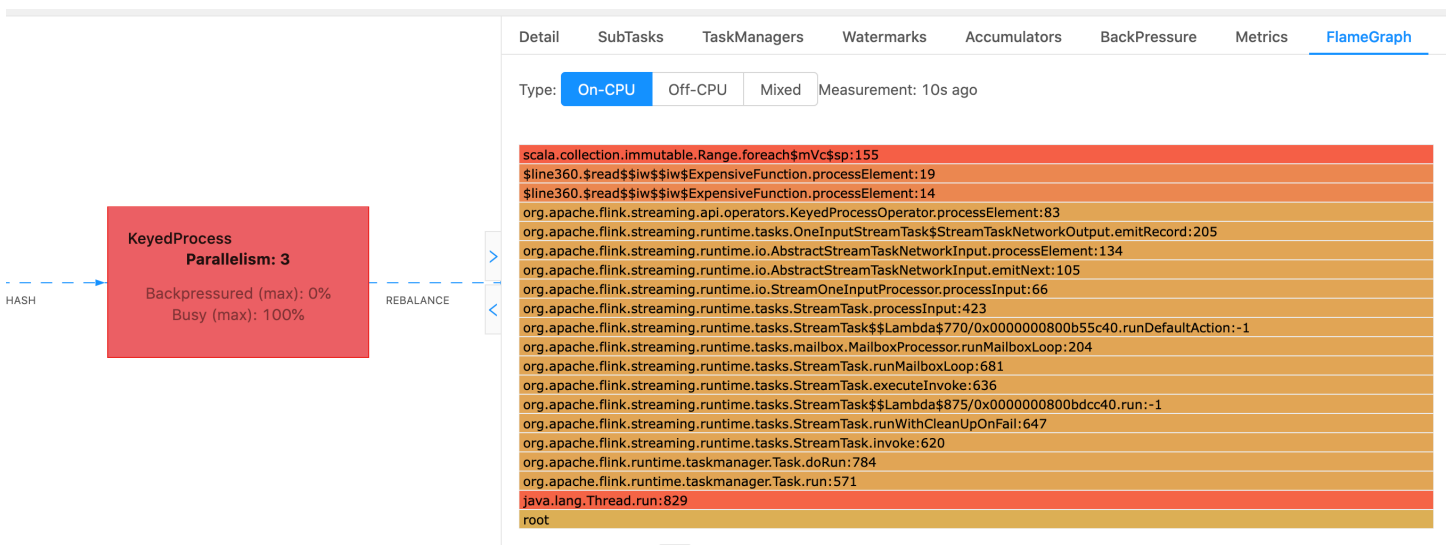
이름이 같은 연산자가 여러 개 있는 경우 혼란스러울 수 있지만 연산자의 이름을 지정하여 이러한 문제를 피할 수 있습니다. 예:

```
....
.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 그래프](#)를 참조하십시오.

Flink에서는 연산자를 선택한 다음 탭을 선택하여 웹 UI를 통해 운영자의 불꽃 그래프에 액세스할 수 있습니다. FlameGraph 샘플이 충분히 수집되면 플레임 그래프가 표시됩니다. 다음은 체크포인트에 많은 시간이 ProcessFunction 걸렸던 것에 FlameGraph 대한 내용입니다.



이것은 매우 간단한 플레임 그래프로, 모든 CPU 시간이 오퍼레이터의 Foreach 록에 processElement 소비되고 있음을 보여줍니다. ExpensiveFunction 또한 라인 번호도 확인할 수 있어 코드 실행이 이루어지는 위치를 파악하는 데 도움이 됩니다.

체크포인트 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 사용자 지정 직렬 변환기를 사용하는 방법에 대한 자세한 내용은 Apache Flink [설명서의 데이터 유형 및 직렬화를](#) 참조하십시오.
- 메모리 부족 오류
- 다음 지표의 스큐 또는 꾸준한 증가
 - heapMemoryUtilization
 - oldGenerationGCtime
 - oldGenerationGCCount
 - lastCheckpointSize
 - lastCheckpointDuration

체크포인트 모니터링에 대한 자세한 내용은 Apache Flink 설명서의 체크포인트 [모니터링](#)을 참조하십시오.

원인 및 해결 방법

애플리케이션 로그 오류 메시지는 직접 실패의 원인이 표시됩니다. 일시적 오류에는 다음과 같은 원인이 있을 수 있습니다:

- 애플리케이션에 KPU 프로비저닝이 충분하지 않음. 애플리케이션 프로비저닝 향상에 대한 자세한 설명은 [스케일링](#)을 참조하십시오.
- 애플리케이션 상태 크기가 너무 큼. lastCheckpointSize지표를 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 oldGenerationGCCount을 모니터링하세요.

Apache Beam 애플리케이션의 체크포인트 실패

Beam 애플리케이션이 0ms로 [shutdownSourcesAfterIdleMs](#) 설정되도록 구성된 경우 작업이 “FINISHED” 상태이기 때문에 체크포인트가 트리거되지 않을 수 있습니다. 이 섹션에서는 그러한 상황의 증상과 해결 방법을 설명합니다.

증상

Apache Flink용 관리 서비스 애플리케이션 CloudWatch 로그로 이동하여 다음 로그 메시지가 기록되었는지 확인하십시오. 다음 로그 메시지는 일부 작업이 완료되어 체크포인트가 트리거되지 않았음을 나타냅니다.

```
{
  "locationInformation":
    "org.apache.flink.runtime.checkpoint.CheckpointCoordinator.onTriggerFailure(CheckpointCoordinator)",
  "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.

Detail	SubTasks	TaskManagers	Watermarks	Accumulators	BackPressure	Metrics	FlameGraph						
ID	Bytes Received	Records Received	Bytes Sent	Records Sent	Attempt	Host	Start Time	Duration	Status	More			
0	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	13m 57s	RUNNING	...			
1	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	3s	FINISHED	...			
2	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	3s	FINISHED	...			
3	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	3s	FINISHED	...			
4	0 B	0	0 B	0	1	sea3-ws-aggr-r3-pc-1	2022-06-06 11:16:03	3s	FINISHED	...			

원인

shutdownSourcesAfterIdleMs 구성된 시간 (밀리초) 동안 유휴 상태였던 소스를 종료하는 Beam 구성 변수입니다. 일단 소스가 종료되면 체크포인트를 더 이상 사용할 수 없습니다. 이로 인해 [체크포인트 오류](#)가 발생할 수 있습니다.

작업이 “FINISHED” 상태로 전환되는 원인 중 하나는 0ms로 설정된 shutdownSourcesAfter IdleMs 경우입니다. 즉, 유휴 상태인 작업은 즉시 종료됩니다.

Solution

작업이 즉시 “완료” 상태로 전환되지 않도록 하려면 long.max_value로 설정합니다 shutdownSourcesAfterIdleMs . 이것은 두 가지 방법으로 수행될 수 있습니다.

- 옵션 1: Apache Flink용 관리 서비스 애플리케이션 구성 페이지에서 빔 구성을 설정한 경우 새 키 값 쌍을 추가하여 다음과 같이 Ms를 설정할 수 있습니다. shutdpwnSourcesAfteridle

Group	Key	Value
BeamApplicationProperties	ShutdownSourcesAfterIdleMs	9223372036854775807

- 옵션 2: 빔 구성이 JAR 파일에 설정된 경우 다음과 같이 설정할 수 있습니다. shutdownSourcesAfter IdleMs

```

        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 방법](#)에서 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 프로파일링이 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 ParallelismPerKPU를 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 싱크](#)는 단순히 애플리케이션을 실행하는 동안 수신되는 모든 이벤트를 무시합니다. 즉, 싱크가 없는 애플리케이션은 실행되지 않습니다. 이는 처리량 테스트, 프로파일링, 애플리케이션이 적절하게 확장되고 있는지 확인하는 데 매우 유용합니다. 또한 싱크로 인해 역압이나 애플리케이션이 발생하는지 확

인하는 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 IO](#) 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/OutPerSecond의 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 완료하는데 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 리소스와 통합

Flink 구성에서 지역 간 복제에 필요한 설정을 통해 Managed Service for Apache Flink 애플리케이션과 다른 지역에 있는 Amazon S3 버킷에 쓸 수 있도록 StreamingFileSink의 사용을 활성화할 수 있습니다. 이를 위해서는 [AWS Support 센터](#)에 지원 티켓을 제출하십시오.

아파치 플링크용 아마존 매니지드 서비스의 문서 기록

다음 표에서는 Managed Service for Apache Flink의 최신 릴리스 이후 이 설명서에서 변경된 중요 사항에 대해 설명합니다.

- API 버전: 2018-05-23
- 최신 설명서 업데이트: 2023년 8월 30일

변경 사항	설명	날짜
Kinesis Data Analytics에서 Managed Service for Apache Flink로 변경	서비스 엔드포인트, API, 명령 줄 인터페이스, IAM 액세스 정책, CloudWatch 지표 또는 결제 대시보드는 변경되지 않습니다. AWS 기존 애플리케이션은 이전과 같이 계속 작동합니다. 자세한 내용을 알아보려면 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 포 API를 사용하여 Kinesis Data Analytics 애플리케이션을 생성합니다. 자세한 내용을 알아보려면 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 애플리케이션을 지원합니다.	2021년 10월 13일

변경 사항	설명	날짜
	<p>Apache Flink 표 API를 사용하여 Kinesis Data Analytics 애플리케이션을 생성합니다. 자세한 내용을 알아보려면 시작하기: Flink 1.13.2 섹션을 참조하세요.</p>	
Python에 대한 지원	<p>Managed Service for Apache Flink는 이제 Apache Flink 표 API 및 SQL과 Python을 사용하는 애플리케이션을 지원합니다. 자세한 내용을 알아보려면 Python 사용 섹션을 참조하세요.</p>	2021년 3월 25일
Apache Flink 1.11.1에 대한 지원	<p>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 내용을 알아보려면 애플리케이션 생성 섹션을 참조하세요.</p>	2020년 11월 19일
Apache Flink 대시보드	<p>Apache Flink 대시보드를 사용하여 애플리케이션 상태 및 성능을 모니터링할 수 있습니다. 자세한 내용을 알아보려면 Apache Flink 대시보드 섹션을 참조하세요.</p>	2020년 11월 19일

변경 사항	설명	날짜
EFO 컨슈머	고급 팬아웃(EFO) 컨슈머를 사용하여 Kinesis Data Stream에서 읽는 애플리케이션을 생성할 수 있습니다. 자세한 내용을 알아보려면 EFO 컨슈머 섹션을 참조하세요.	2020년 10월 6일
Apache Beam	Apache Beam을 사용하여 스트리밍 데이터를 처리하는 애플리케이션을 생성합니다. 자세한 내용을 알아보려면 아파치 플링크용 매니지드 서비스와 CloudFormation 함께 사용 섹션을 참조하세요.	2020년 9월 15일
성능	애플리케이션 성능 문제를 해결하는 방법 및 성능이 우수한 애플리케이션을 생성하는 방법. 자세한 내용을 알아보려면 성능 섹션을 참조하세요.	2020년 7월 21일
맞춤 키스토어	전송 중 암호화를 위해 맞춤 키스토어를 사용하는 Amazon MSK 클러스터에 액세스하는 방법. 자세한 내용을 알아보려면 사용자 지정 트러스트 스토어 섹션을 참조하세요.	2020년 6월 10일
CloudWatch 알람	Apache Flink용 관리형 서비스를 사용하여 CloudWatch 경보를 생성하기 위한 권장 사항. 자세한 정보는 경보 을 참조하세요.	2020년 6월 5일

변경 사항	설명	날짜
CloudWatch 새 지표	아파치 플링크용 관리형 서비스는 이제 Amazon Metrics에 22개의 지표를 내보냅니다. CloudWatch 자세한 정보는 Apache Flink용 매니지드 서비스의 메트릭과 디멘션 을 참조하세요.	2020년 5월 12일
CloudWatch 사용자 지정 지표	애플리케이션별 지표를 정의하고 이를 Amazon Metrics로 내보냅니다. CloudWatch 자세한 정보는 사용자 지정 지표 을 참조하세요.	2020년 5월 12일
예: 다른 계정의 Kinesis 스트림에서 읽기	Apache Flink용 관리형 서비스 애플리케이션에서 다른 AWS 계정으로 Kinesis 스트림에 액세스하는 방법을 알아봅니다. 자세한 정보는 크로스 계정 을 참조하세요.	2020년 3월 30일
Apache Flink 1.8.2에 대한 지원	Managed Service for Apache Flink는 이제 Apache Flink 1.8.2 를 사용하는 애플리케이션을 지원합니다. Flink StreamingFileSink 커넥터를 사용하여 출력을 S3에 직접 기록할 수 있습니다. 자세한 정보는 애플리케이션 생성 을 참조하세요.	2019년 12월 17일

변경 사항	설명	날짜
Managed Service for Apache Flink VPC	가상 프라이빗 클라우드에 연결되도록 Managed Service for Apache Flink 애플리케이션을 구성합니다. 자세한 내용을 알아보려면 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 내용을 알아보려면 모범 사례 섹션을 참조하세요.	2019년 10월 14일
Managed Service for Apache Flink 애플리케이션 로그 분석	CloudWatch 로그 인사이트를 사용하여 Apache Flink용 관리형 서비스 애플리케이션을 모니터링할 수 있습니다. 자세한 정보는 로그 분석 을 참조하세요.	2019년 6월 26일
Managed Service for Apache Flink 애플리케이션 런타임 속성	Managed Service for Apache Flink에서 런타임 속성으로 작업합니다. 자세한 내용을 알아보려면 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 태깅 사용 섹션을 참조하십시오.	2019년 5월 8일

변경 사항	설명	날짜
<p>를 사용하여 Apache Flink API 호출에 대한 관리 서비스를 로깅합니다. AWS CloudTrail</p>	<p>Apache Flink용 관리형 서비스는 Apache Flink용 관리형 서비스에서 사용자, 역할 또는 서비스가 수행한 작업에 대한 기록을 제공하는 AWS 서비스와 통합되어 있습니다. AWS CloudTrail 자세한 정보는 사용 AWS CloudTrail을 참조하세요.</p>	<p>2019년 3월 22일</p>
<p>애플리케이션 만들기 (Firehose 싱크)</p>	<p>Amazon Kinesis 데이터 스트림을 소스로 사용하고 Amazon Data Firehose 스트림을 싱크로 사용하여 Apache Flink용 관리형 서비스를 생성하는 연습을 해 보십시오. 자세한 정보는 Firehose 싱크을 참조하세요.</p>	<p>2018년 12월 13일</p>
<p>공개 릴리스</p>	<p>Java 애플리케이션용 Managed Service for Apache Flink 개발자 가이드가 처음으로 릴리스되었습니다.</p>	<p>2018년 11월 27일</p>

아파치 플링크용 매니지드 서비스 API 예제 코드

이 항목에는 Managed Service for Apache Flink 작업을 위한 예 요청 블록이 포함되어 있습니다.

AWS Command Line Interface (AWS CLI) 와 함께 JSON을 작업의 입력으로 사용하려면 요청을 JSON 파일에 저장합니다. 그런 다음 `--cli-input-json` 파라미터를 사용하여 파일 명칭을 작업에 전달합니다.

다음 예에서는 작업과 함께 JSON 파일을 사용하는 방법을 보여줍니다.

```
$ aws kinesisanalyticstv2 start-application --cli-input-json file://start.json
```

에서 JSON을 사용하는 방법에 대한 자세한 내용은 사용 AWS CLI설명서의 [CLI 스켈레톤 및 CLI 입력 JSON 파라미터 생성](#)을 참조하십시오. [AWS Command Line Interface](#)

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

다음 예제 [AddApplicationCloudWatchLoggingOption](#)작업 요청 코드는 Apache Flink용 관리형 서비스 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

다음 예제 [AddApplicationInput](#)작업 요청 코드는 Apache Flink용 관리형 서비스 애플리케이션에 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

다음 예제 [AddApplicationInputProcessingConfiguration](#) 작업 요청 코드는 Apache Flink용 관리 서비스 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](#)작업에 대한 다음 예제 요청 코드는 Kinesis 데이터 스트림을 Apache Flink용 관리형 서비스 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

다음 예제 [AddApplicationReferenceDataSource](#)작업 요청 코드는 CSV 애플리케이션 참조 데이터 소스를 Apache Flink용 관리형 서비스 애플리케이션에 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](#) 작업에 대한 다음 예제 요청 코드는 기존 애플리케이션에 VPC 구성을 추가합니다.

```

{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 9,
  "VpcConfiguration": {
    "SecurityGroupIds": [ "sg-0123456789abcdef0" ],
    "SubnetIds": [ "subnet-0123456789abcdef0" ]
  }
}

```

CreateApplication

[CreateApplication](#) 작업에 대한 다음 예제 요청 코드는 Apache Flink용 관리형 서비스 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](#)작업에 대한 다음 예제 요청 코드는 애플리케이션 상태의 스냅샷을 생성합니다.

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

DeleteApplication

[DeleteApplication](#)작업에 대한 다음 예제 요청 코드는 Apache Flink용 관리형 서비스 애플리케이션을 삭제합니다.

```
{"ApplicationName": "MyApplication",
 "CreateTimestamp": 12345678912}
```

DeleteApplicationCloudWatchLoggingOption

다음 예제 [DeleteApplicationCloudWatchLoggingOption](#)작업 요청 코드는 Apache Flink용 관리형 서비스 애플리케이션에서 Amazon CloudWatch 로깅 옵션을 삭제합니다.

```
{
  "ApplicationName": "MyApplication",
  "CloudWatchLoggingOptionId": "3.1"
  "CurrentApplicationVersionId": 3
}
```

DeleteApplicationInputProcessingConfiguration

[DeleteApplicationInputProcessingConfiguration](#)작업에 대한 다음 예제 요청 코드는 Apache Flink용 관리형 서비스 애플리케이션에서 입력 처리 구성을 제거합니다.

```
{
```

```
"ApplicationName": "MyApplication",  
"CurrentApplicationVersionId": 4,  
"InputId": "2.1"  
}
```

DeleteApplicationOutput

[DeleteApplicationOutput](#)작업에 대한 다음 예제 요청 코드는 Apache Flink용 관리 서비스 애플리케이션에서 애플리케이션 출력을 제거합니다.

```
{  
  "ApplicationName": "MyApplication",  
  "CurrentApplicationVersionId": 4,  
  "OutputId": "4.1"  
}
```

DeleteApplicationReferenceDataSource

[DeleteApplicationReferenceDataSource](#)작업에 대한 다음 예제 요청 코드는 Apache Flink용 관리 서비스 애플리케이션에서 애플리케이션 참조 데이터 소스를 제거합니다.

```
{  
  "ApplicationName": "MyApplication",  
  "CurrentApplicationVersionId": 5,  
  "ReferenceId": "5.1"  
}
```

DeleteApplicationSnapshot

[DeleteApplicationSnapshot](#)작업에 대한 다음 예제 요청 코드는 애플리케이션 상태의 스냅샷을 삭제합니다.

```
{  
  "ApplicationName": "MyApplication",  
  "SnapshotCreationTimestamp": 12345678912,  
  "SnapshotName": "MySnapshot"  
}
```


DeleteApplicationVpcConfiguration

[DeleteApplicationVpcConfiguration](#)작업에 대한 다음 예제 요청 코드는 애플리케이션에서 기존 VPC 구성을 제거합니다.

```
{
  "ApplicationName": "MyApplication",
  "CurrentApplicationVersionId": 9,
  "VpcConfigurationId": "1.1"
}
```

DescribeApplication

[DescribeApplication](#)작업에 대한 다음 예제 요청 코드는 Apache Flink용 관리형 서비스 애플리케이션에 대한 세부 정보를 반환합니다.

```
{"ApplicationName": "MyApplication"}
```

DescribeApplicationSnapshot

[DescribeApplicationSnapshot](#)작업에 대한 다음 예제 요청 코드는 애플리케이션 상태 스냅샷에 대한 세부 정보를 반환합니다.

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

DiscoverInputSchema

다음 예제 [DiscoverInputSchema](#)작업 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"
}

```

다음 예제 [DiscoverInputSchema](#) 작업 요청 코드는 참조 소스에서 스키마를 생성합니다.

```

{
  "S3Configuration": {
    "BucketARN": "arn:aws:s3:::mybucket",
    "FileKey": "TickerReference.csv"
  },
  "ServiceExecutionRole": "arn:aws:iam::123456789123:role/myrole"
}

```

ListApplications

다음 예제 [ListApplications](#) 작업 요청 코드는 계정의 Apache Flink 관리 서비스 애플리케이션 목록을 반환합니다.

```

{
  "ExclusiveStartApplicationName": "MyApplication",
  "Limit": 50
}

```

ListApplicationSnapshots

[ListApplicationSnapshots](#) 작업에 대한 다음 예제 요청 코드는 애플리케이션 상태의 스냅샷 목록을 반환합니다.

```

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

```

```
"NextToken": "aBcDeFgHiJkLmNoPqRsTuVwXyZ0123"
}
```

StartApplication

다음 예제 [StartApplication](#) 작업 요청 코드는 Apache Flink용 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](#) 작업에 대한 다음 예제 요청 코드는 Apache Flink용 관리 서비스 애플리케이션을 중지합니다.

```
{"ApplicationName": "MyApplication"}
```

UpdateApplication

[UpdateApplication](#) 작업에 대한 다음 예제 요청 코드는 Apache Flink용 관리형 서비스 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 내용을 알아보려면 [Managed Service for Apache Flink API 참조](#)를 참조하십시오.

이 콘텐츠는 릴리스 버전으로 이동되었습니다. [릴리스 버전](#)을 참조하세요.