



Amazon EMR Serverless 사용 설명서

# Amazon EMR



# Amazon EMR: Amazon EMR Serverless 사용 설명서

Copyright © 2025 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon의 상표 및 트레이드 드레스는 Amazon 외 제품 또는 서비스와 함께, Amazon 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

Amazon EMR Serverless란?	1
개념	1
릴리스 버전	1
Application	1
작업 실행	2
작업자	3
사전 초기화된 용량	3
EMR Studio	3
시작하기 위한 사전 조건	4
AWS 계정에 등록	4
관리자 액세스 권한이 있는 사용자 생성	5
권한 부여	6
프로그래밍 방식 액세스 권한 부여	8
AWS CLI 설정	9
콘솔 열기	10
시작하기	11
권한	11
스토리지	11
대화형 워크로드	11
작업 런타임 역할 생성	12
콘솔에서 시작하기	17
1단계: 애플리케이션 생성	18
2단계: 작업 실행 또는 대화형 워크로드 제출	18
3단계: 애플리케이션 UI 및 로그 보기	21
4단계: 정리	22
AWS CLI에서 시작하기	22
1단계: 애플리케이션 생성	22
2단계: 작업 실행 제출	23
3단계: 결과 검토	26
4단계: 정리	27
EMR Serverless 애플리케이션 구성 및 상호 작용	29
애플리케이션 상태	29
EMR Studio 콘솔 사용	30
애플리케이션 만들기	30

EMR Studio 콘솔에서 애플리케이션 나열 .....	31
EMR Studio 콘솔에서 애플리케이션 관리 .....	32
AWS CLI 사용 .....	32
애플리케이션 구성 .....	33
애플리케이션 동작 .....	34
EMR Serverless에서 애플리케이션 작업을 위한 사전 초기화된 용량 .....	35
기본 앱 구성 .....	39
이미지 사용자 지정 .....	44
사전 조건 .....	34
1단계: EMR Serverless 기본 이미지에서 사용자 지정 이미지 생성 .....	46
2단계: 로컬에서 이미지 검증 .....	47
3단계: 이미지를 Amazon ECR 리포지토리에 업로드 .....	48
4단계: 사용자 지정 이미지를 사용하여 애플리케이션 생성 또는 업데이트 .....	48
5단계: EMR Serverless가 사용자 지정 이미지 리포지토리에 액세스하도록 허용 .....	50
고려 사항 및 제한 사항 .....	50
데이터에 연결하도록 EMR Serverless 애플리케이션에 대한 VPC 액세스 구성 .....	51
애플리케이션 생성 .....	51
애플리케이션 구성 .....	54
서브넷 계획 모범 사례 .....	55
아키텍처 옵션 .....	56
x86_64 아키텍처 사용 .....	57
arm64 아키텍처(Graviton) 사용 .....	57
Graviton으로 새 앱 시작 .....	57
기존 앱을 Graviton으로 변환 .....	58
고려 사항 .....	59
작업 동시성 및 대기열 입력 .....	59
동시성 및 대기열 입력의 주요 이점 .....	59
동시성 및 대기열 입력 시작하기 .....	59
동시성 및 대기열 입력에 대한 고려 사항 .....	61
데이터 업로드 .....	62
사전 조건 .....	62
S3 Express One Zone 시작하기 .....	63
작업 실행 .....	65
작업 실행 상태 .....	65
유예 기간이 있는 작업 실행 취소 .....	67
배치 작업의 유예 기간 .....	67

스트리밍 작업의 유예 기간 .....	68
고려 사항 .....	50
EMR Studio 콘솔 사용 .....	71
작업 제출 .....	71
작업 실행 액세스 .....	73
AWS CLI 사용 .....	74
실행 IAM 정책 .....	75
시작하기 .....	75
CLI 명령 예제 .....	76
중요 정보 .....	77
정책 교차 .....	77
서플 최적화 디스크 사용 .....	80
주요 이점 .....	81
시작하기 .....	81
지속적으로 스트리밍되는 데이터를 처리하기 위한 스트리밍 작업 .....	85
고려 사항 및 제한 사항 .....	86
시작하기 .....	87
스트리밍 커넥터 .....	87
로그 관리 .....	90
EMR Serverless 작업을 실행하는 경우 Spark 구성 사용 .....	90
Spark 파라미터 .....	91
Spark 속성 .....	94
리소스 구성 모범 사례 .....	98
Spark 예제 .....	99
EMR Serverless 작업을 실행하는 경우 Hive 구성 사용 .....	100
Hive 파라미터 .....	100
Hive 속성 .....	102
Hive 예제 .....	114
작업 복원력 .....	114
재시도 정책으로 작업 모니터링 .....	117
재시도 정책을 사용한 로깅 .....	118
EMR Serverless에 대한 메타스토어 구성 .....	118
AWS Glue Data Catalog를 메타스토어로 사용 .....	118
외부 Hive 메타스토어 사용 .....	123
EMR Serverless에서 AWS Glue 다중 카탈로그 계층 구조 작업 .....	128
외부 메타스토어 사용 시 고려 사항 .....	129

교차 계정 S3 액세스 .....	130
사전 조건 .....	130
S3 버킷 정책 사용 .....	130
수입된 역할 사용 .....	131
수입된 역할 예제 .....	134
오류 해결 .....	138
오류: 계정이 동시에 사용할 수 있는 최대 vCPU의 서비스 제한에 도달하여 작업이 실패했습니다. ....	139
오류: 애플리케이션이 maximumCapacity 설정을 초과하여 작업이 실패했습니다. ....	139
오류: 애플리케이션이 maximumCapacity를 초과하여 워커를 할당할 수 없어 작업이 실패했습니다. ....	139
오류: S3 액세스가 거부되었습니다. 필요한 S3 리소스에서 작업 런타임 역할의 S3 액세스 권한을 확인하세요. ....	139
오류: ModuleNotFoundError: <module> 모듈이 없습니다. EMR Serverless에서 Python 라이브러리를 사용하는 방법은 사용 설명서를 참조하세요. ....	139
오류: <role name> 역할을 수입할 수 없습니다. 해당 역할이 없거나 필요한 신뢰 관계로 설정되지 않았기 때문입니다. ....	140
대화형 워크로드 실행 .....	141
개요 .....	141
사전 조건 .....	141
권한 .....	142
구성 .....	143
고려 사항 .....	143
Apache Livy 엔드포인트를 통해 대화형 워크로드 실행 .....	145
사전 조건 .....	145
필수 권한 .....	145
시작하기 .....	147
고려 사항 .....	153
로깅 및 모니터링 .....	156
로그 저장 .....	156
관리형 스토리지 .....	157
Amazon S3 .....	158
Amazon CloudWatch .....	160
로그 교체 .....	163
로그 암호화 .....	164
관리형 스토리지 .....	164

Amazon S3 버킷 .....	164
Amazon CloudWatch .....	165
필수 권한 .....	165
Log4j2 구성 .....	169
Log4j2 및 Spark .....	169
모니터링 .....	173
애플리케이션 및 작업 .....	174
Spark 엔진 지표 .....	183
사용량 지표 .....	187
EventBridge를 사용한 자동화 .....	188
샘플 EMR Serverless EventBridge 이벤트 .....	188
리소스에 태그 지정 .....	193
태그란 무엇입니까? .....	193
리소스에 태그 지정 .....	193
태그 지정 제한 사항 .....	194
태그 작업 .....	195
튜토리얼 .....	196
Java 17 사용 .....	196
JAVA_HOME .....	196
spark-defaults .....	197
Hudi 사용 .....	198
Iceberg 사용 .....	199
Python 라이브러리 사용 .....	200
기본 Python 기능 사용 .....	200
Python 가상 환경 빌드 .....	201
Python 라이브러리를 사용하도록 PySpark 작업 구성 .....	202
다양한 Python 버전 사용 .....	203
Delta Lake OSS 사용 .....	204
Amazon EMR 버전 6.9.0 이상 .....	204
Amazon EMR 버전 6.8.0 이하 .....	206
Airflow에서 작업 제출 .....	207
Hive 사용자 정의 함수 사용 .....	209
사용자 지정 이미지 사용 .....	211
사용자 지정 Python 버전 사용 .....	211
사용자 지정 Java 버전 사용 .....	212
데이터 과학 이미지 빌드 .....	212

Apache Sedona를 사용한 지리 공간 데이터 처리 .....	213
사용자 지정 이미지 사용에 대한 라이선스 정보 .....	213
Amazon Redshift에서 Spark 사용 .....	214
Spark 애플리케이션 시작 .....	214
Amazon Redshift에 대한 인증 .....	215
Amazon Redshift에 대한 읽고 쓰기 .....	218
고려 사항 .....	220
DynamoDB에 연결 .....	221
1단계: Amazon S3에 업로드 .....	221
2단계: Hive 테이블 생성 .....	222
3단계: DynamoDB로 복사 .....	223
4단계: DynamoDB에서 쿼리 .....	224
교차 계정 액세스 설정 .....	226
고려 사항 .....	228
보안 .....	230
보안 모범 사례 .....	231
최소 권한의 원칙 적용 .....	231
신뢰할 수 없는 애플리케이션 코드 격리 .....	231
역할 기반 액세스 제어(RBAC) 권한 .....	231
데이터 보호 .....	231
저장 시 암호화 .....	232
전송 중 암호화 .....	234
Identity and Access Management(IAM) .....	235
고객 .....	235
ID를 통한 인증 .....	236
정책을 사용하여 액세스 관리 .....	237
EMR Serverless에서 IAM와의 작동 방식 .....	238
서비스 연결 역할 사용 .....	243
Amazon EMR Serverless에 대한 작업 런타임 역할 .....	248
사용자 액세스 정책 .....	251
태그 기반 액세스 제어를 위한 정책 .....	256
ID 기반 정책 .....	259
정책 업데이트 .....	261
문제 해결 .....	262
Amazon EMR Serverless를 사용한 신뢰할 수 있는 자격 증명 전파 .....	264
개요 .....	265

기능 및 이점 .....	265
작동 방법 .....	266
EMR Serverless의 신뢰된 자격 증명 전파 시작하기 .....	266
EMR Serverless를 사용한 대화형 방식 워크로드의 신뢰할 수 있는 자격 증명 전파 .....	270
EMR Serverless 신뢰 자격 증명 전파 통합 시 고려 사항 및 제한 사항 .....	270
EMR Serverless와 함께 Lake Formation 사용 .....	272
EMR Serverless에 대한 Lake Formation 필터링되지 않은 액세스 .....	272
FGAC를 위한 Lake Formation .....	280
작업자 간 암호화 .....	300
EMR Serverless에서 상호 TLS 암호화 활성화 .....	300
데이터 보호를 위한 Secrets Manager .....	300
보안 암호 작동 방식 .....	301
보안 암호 생성 .....	301
보안 암호 참조 지정 .....	301
보안 암호에 대한 액세스 권한 부여 .....	304
보안 암호 교체 .....	305
데이터 액세스 제어를 위한 S3 Access Grants .....	306
개요 .....	306
애플리케이션 시작 .....	307
고려 사항 .....	308
로깅을 위한 CloudTrail .....	308
CloudTrail의 EMR Serverless 정보 .....	309
EMR Serverless 로그 파일 항목 이해 .....	309
규정 준수 확인 .....	311
복원성 .....	312
인프라 보안 .....	312
구성 및 취약성 분석 .....	313
엔드포인트 및 할당량 .....	314
서비스 엔드포인트 .....	314
Service Quotas .....	319
API 제한 .....	320
기타 고려 사항 .....	50
릴리스 버전 .....	324
EMR Serverless 7.10.0 .....	325
EMR Serverless 7.9.0 .....	325
EMR Serverless 7.8.0 .....	325

EMR Serverless 7.7.0 .....	326
EMR Serverless 7.6.0 .....	326
EMR Serverless 7.5.0 .....	326
EMR Serverless 7.4.0 .....	327
EMR Serverless 7.3.0 .....	327
EMR Serverless 7.2.0 .....	328
EMR Serverless 7.1.0 .....	328
EMR Serverless 7.0.0 .....	329
EMR Serverless 6.15.0 .....	329
EMR Serverless 6.14.0 .....	329
EMR Serverless 6.13.0 .....	330
EMR Serverless 6.12.0 .....	330
EMR Serverless 6.11.0 .....	330
EMR Serverless 6.10.0 .....	331
EMR Serverless 6.9.0 .....	331
EMR Serverless 6.8.0 .....	332
EMR Serverless 6.7.0 .....	333
엔진별 변경 사항 .....	333
EMR Serverless 6.6.0 .....	333
문서 기록 .....	335

# Amazon EMR Serverless란?

Amazon EMR Serverless는 서버리스 런타임 환경을 제공하는 Amazon EMR의 배포 옵션입니다. 그러면 Apache Spark 및 Apache Hive와 같은 최신 오픈 소스 프레임워크를 사용하는 분석 애플리케이션의 작업이 간소화됩니다. EMR Serverless를 사용하면 이러한 프레임워크에서 애플리케이션을 실행하기 위해 클러스터를 구성, 최적화, 보안 또는 운영하지 않아도 됩니다.

EMR Serverless를 사용하면 데이터 처리 작업에 대한 리소스의 과다 프로비저닝 또는 과소 프로비저닝을 방지할 수 있습니다. EMR Serverless는 애플리케이션에 필요한 리소스를 자동으로 결정하고, 이러한 리소스를 가져와 작업을 처리하며, 작업이 완료되면 리소스를 해제합니다. 대화형 데이터 분석과 같이 애플리케이션이 몇 초 내에 응답해야 하는 사용 사례의 경우 애플리케이션을 생성할 때 애플리케이션에 필요한 리소스를 사전 초기화할 수 있습니다.

EMR Serverless를 사용하면 오픈 소스 호환성, 동시성, 널리 사용되는 프레임워크를 위한 최적화된 런타임 성능과 같은 Amazon EMR의 이점을 계속 누릴 수 있습니다.

EMR Serverless는 오픈 소스 프레임워크를 사용하여 애플리케이션을 쉽게 운영하려는 고객에게 적합합니다. 빠른 작업 시작, 자동 용량 관리 및 간단한 비용 제어를 제공합니다.

## 개념

이 섹션에서는 EMR Serverless 사용 설명서 전체에 표시되는 EMR Serverless 용어 및 개념을 다룹니다.

## 릴리스 버전

Amazon EMR 릴리스는 빅 데이터 에코시스템의 오픈 소스 애플리케이션입니다. 각 릴리스에는 애플리케이션을 실행할 수 있도록 EMR Serverless에서 배포 및 구성하기 위해 선택한 다양한 빅 데이터 애플리케이션, 구성 요소 및 기능이 포함되어 있습니다. 애플리케이션 버전을 생성하는 경우 해당 릴리스 버전을 지정합니다. 애플리케이션에서 사용하려는 오픈 소스 프레임워크 버전 및 Amazon EMR 릴리스 버전을 선택합니다. 사전 릴리스 버전에 대해 자세히 알아보려면 [Amazon EMR Serverless 릴리스 버전](#) 섹션을 참조하세요.

## Application

EMR Serverless를 사용하면 오픈 소스 분석 프레임워크를 사용하는 하나 이상의 EMR Serverless 애플리케이션을 생성할 수 있습니다. 애플리케이션을 생성하려면 다음 속성을 지정합니다.

- 사용하려는 오픈 소스 프레임워크 버전에 대한 Amazon EMR 릴리스 버전. 릴리스 버전을 확인하려면 [Amazon EMR Serverless 릴리스 버전](#) 섹션을 참조하세요.
- Apache Spark 또는 Apache Hive와 같이 애플리케이션에서 사용할 특정 런타임.

애플리케이션을 생성한 후 데이터 처리 작업이나 대화형 요청을 애플리케이션에 제출합니다.

각 EMR Serverless 애플리케이션은 다른 애플리케이션과 엄격하게 구분되는 보안 Amazon Virtual Private Cloud(VPC)에서 실행됩니다. 또한 AWS Identity and Access Management(IAM) 정책을 사용하여 애플리케이션에 액세스할 수 있는 사용자 및 역할을 정의합니다. 애플리케이션에서 발생하는 사용 비용을 제어하고 추적하는 제한을 지정할 수도 있습니다.

다음을 수행하는 경우 여러 애플리케이션을 생성하는 방법을 고려합니다.

- 다양한 오픈 소스 프레임워크 사용
- 다양한 사용 사례에 대해 다양한 버전의 오픈 소스 프레임워크 사용
- 한 버전에서 다른 버전으로 업그레이드하는 경우 A/B 테스트 수행
- 테스트 및 프로덕션 시나리오를 위한 별도의 논리적 환경 유지 관리
- 독립적인 비용 제어 및 사용량 추적을 통해 여러 팀에 별도의 논리적 환경 제공
- 다양한 사업부 애플리케이션 분리

EMR Serverless는 한 리전의 여러 가용 영역에서 워크로드가 실행되는 방식을 간소화하는 리전 서비스입니다. EMR Serverless에서 애플리케이션을 사용하는 방법에 대해 자세히 알아보려면 [EMR Serverless 애플리케이션 구성 및 상호 작용](#) 섹션을 참조하세요.

## 작업 실행

작업 실행은 EMR Serverless 애플리케이션에 제출된 요청으로, 애플리케이션이 비동기식으로 실행하고 완료 시점까지 추적합니다. 작업의 예로, Apache Hive 애플리케이션에 제출하는 HiveQL 쿼리 또는 Apache Spark 애플리케이션에 제출하는 PySpark 데이터 처리 스크립트가 있습니다. 작업을 제출하는 경우 작업이 Amazon S3 객체와 같은 AWS 리소스에 액세스하는 데 사용하는 런타임 역할(IAM에서 생성됨)을 지정해야 합니다. 애플리케이션에 여러 작업 실행 요청을 제출할 수 있으며, 각 작업 실행은 다른 런타임 역할을 사용하여 AWS 리소스에 액세스할 수 있습니다. EMR Serverless 애플리케이션은 작업을 수신하는 즉시 실행을 시작하고 여러 작업 요청을 동시에 실행합니다. EMR Serverless가 작업을 실행하는 방법에 대해 자세히 알아보려면 [작업 실행](#) 섹션을 참조하세요.

## 작업자

EMR Serverless 애플리케이션은 내부적으로 작업자를 사용하여 워크로드를 실행합니다. 이러한 작업자의 기본 크기는 애플리케이션 유형 및 Amazon EMR 릴리스 버전을 기반으로 합니다. 작업 실행을 예약하는 경우 이러한 크기를 재정의합니다.

작업을 제출하면 EMR Serverless는 작업에 대해 애플리케이션에 필요한 리소스를 계산하고 작업자를 예약합니다. EMR Serverless는 워크로드를 태스크로 분류하고, 이미지를 다운로드하며, 작업자를 프로비저닝 및 설정하고, 작업이 완료되면 해제합니다. EMR Serverless는 작업의 모든 단계에서 필요한 워크로드 및 병렬 처리를 기반으로 작업자를 자동으로 스케일 업하거나 스케일 다운합니다. 이 자동 조정을 사용하면 애플리케이션이 워크로드를 실행하는 데 필요한 작업자 수를 예측하지 않아도 됩니다.

## 사전 초기화된 용량

EMR Serverless는 작업자를 몇 초 이내에 초기화하고 응답할 수 있도록 사전 초기화된 용량 기능을 제공합니다. 이 용량은 애플리케이션에 대한 작업자의 워밍업을 효과적으로 생성합니다. 각 애플리케이션에 대해 이 기능을 구성하려면 애플리케이션의 `initial-capacity` 파라미터를 설정합니다. 사전 초기화된 용량을 구성하면 반복 애플리케이션 및 시간에 민감한 작업을 구현할 수 있도록 작업이 즉시 시작될 수 있습니다. 사전 초기화된 워커에 대해 자세히 알아보려면 [EMR Serverless 작업 시 애플리케이션 구성](#) 섹션을 참조하세요.

## EMR Studio

EMR Studio는 EMR Serverless 애플리케이션을 관리하는 사용자 콘솔입니다. 첫 번째 EMR Serverless 애플리케이션을 생성할 때 계정에 EMR Studio가 없는 경우 자동으로 생성됩니다. Amazon EMR 콘솔에서 EMR Studio에 액세스하거나, IAM 또는 IAM Identity Center를 통해 ID 공급자(IdP)에서 페더레이션 액세스를 활성화합니다. 이 경우 사용자가 Amazon EMR 콘솔에 직접 액세스하지 않고도 Studio에 액세스하고 EMR Serverless 애플리케이션을 관리할 수 있습니다. EMR Serverless 애플리케이션이 EMR Studio와 작동하는 방식에 대해 자세히 알아보려면 [EMR Studio 콘솔에서 EMR Serverless 애플리케이션 생성](#) 및 [EMR Studio 콘솔에서 작업 실행](#) 섹션을 참조하세요.

# EMR Serverless를 시작하기 위한 사전 조건

이 섹션에서는 EMR Serverless를 실행하기 위한 관리 사전 조건을 설명합니다. 여기에는 계정 구성 및 권한 관리가 포함됩니다.

주제

- [AWS 계정에 등록](#)
- [관리자 액세스 권한이 있는 사용자 생성](#)
- [권한 부여](#)
- [AWS CLI 설치 및 구성](#)
- [콘솔 열기](#)

## AWS 계정에 등록

AWS 계정이 없는 경우 다음 절차에 따라 계정을 생성합니다.

AWS 계정에 가입

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

등록 절차 중 전화 또는 텍스트 메시지를 받고 전화 키패드로 확인 코드를 입력하는 과정이 있습니다.

AWS 계정 루트 사용자에게 가입하면 AWS 계정 루트 사용자가 만들어집니다. 루트 사용자에게는 계정의 모든 AWS 서비스 및 리소스에 액세스할 권한이 있습니다. 보안 모범 사례는 사용자에게 관리 액세스 권한을 할당하고, 루트 사용자만 사용하여 [루트 사용자 액세스 권한이 필요한 작업을](#) 수행하는 것입니다.

AWS는 가입 절차 완료된 후 사용자에게 확인 이메일을 전송합니다. 언제든지 <https://aws.amazon.com/>으로 이동하고 내 계정을 선택하여 현재 계정 활동을 보고 계정을 관리할 수 있습니다.

## 관리자 액세스 권한이 있는 사용자 생성

AWS 계정에 가입하고 AWS 계정 루트 사용자에게 보안 조치를 한 다음, AWS IAM Identity Center을 활성화하고 일상적인 작업에 루트 사용자를 사용하지 않도록 관리 사용자를 생성합니다.

귀하의 AWS 계정 루트 사용자 보호

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

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

2. 루트 사용자의 다중 인증(MFA)을 활성화합니다.

지침은 IAM 사용 설명서의 [AWS 계정 루트 사용자용 가상 MFA 디바이스 활성화\(콘솔\)](#)를 참조하세요.

관리자 액세스 권한이 있는 사용자 생성

1. IAM Identity Center를 활성화합니다.

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

2. IAM Identity Center에서 사용자에게 관리자 액세스 권한을 부여합니다.

IAM Identity Center 디렉터리를 ID 소스로 사용하는 방법에 대한 자습서는 AWS IAM Identity Center 사용 설명서의 [기본 IAM Identity Center 디렉터리로 사용자 액세스 구성](#)을 참조하세요.

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 사용자로 로그인하는 데 도움이 필요한 경우 AWS 로그인 사용 설명서의 [AWS 액세스 포털에 로그인](#)을 참조하세요.

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.

## 권한 부여

프로덕션 환경에서는 더 세밀한 정책을 사용하는 것이 좋습니다. 이러한 정책의 몇 가지 예제는 [EMR Serverless에 대한 사용자 액세스 정책 예제](#) 섹션을 참조하세요. 액세스 관리에 대한 자세한 내용은 IAM 사용 설명서에서 [AWS 리소스에 대한 액세스 관리](#)를 참조하세요.

샌드박스 환경에서 EMR Serverless를 시작해야 하는 사용자의 경우 다음과 유사한 정책을 사용합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRStudioCreate",
      "Effect": "Allow",
      "Action": [
        "elasticmapreduce:CreateStudioPresignedUrl",
        "elasticmapreduce:DescribeStudio",
        "elasticmapreduce:CreateStudio",
        "elasticmapreduce:ListStudios"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "EMRServerlessFullAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:*"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

```

    ]
  },
  {
    "Sid": "AllowEC2ENICreationWithEMRTags",
    "Effect": "Allow",
    "Action": [
      "ec2:CreateNetworkInterface"
    ],
    "Resource": [
      "arn:aws:ec2:*:*:network-interface/*"
    ],
    "Condition": {
      "StringEquals": {
        "aws:CalledViaLast": "ops.emr-serverless.amazonaws.com"
      }
    }
  },
  {
    "Sid": "AllowEMRServerlessServiceLinkedRoleCreation",
    "Effect": "Allow",
    "Action": [
      "iam:CreateServiceLinkedRole"
    ],
    "Resource": [
      "arn:aws:iam:*:*:role/aws-service-role/*"
    ]
  }
]
}

```

액세스 권한을 제공하려면 사용자, 그룹 또는 역할에 권한을 추가하세요:

- AWS IAM Identity Center의 사용자 및 그룹:

권한 세트를 생성합니다. AWS IAM Identity Center 사용 설명서의 [권한 세트 생성](#)의 지침을 따릅니다.

- 보안 인증 공급자를 통해 IAM에서 관리되는 사용자:

ID 페더레이션을 위한 역할을 생성합니다. IAM 사용 설명서의 [Create a role for a third-party identity provider \(federation\)](#)의 지침을 따릅니다.

- IAM 사용자:

- 사용자가 맡을 수 있는 역할을 생성합니다. IAM 사용 설명서에서 [Create a role for an IAM user](#)의 지침을 따릅니다.
- (권장되지 않음) 정책을 사용자에게 직접 연결하거나 사용자를 사용자 그룹에 추가합니다. IAM 사용 설명서에서 [사용자\(콘솔\)에 권한 추가](#)의 지침을 따르세요.

## 프로그래밍 방식 액세스 권한 부여

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

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

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
작업 인력 ID  (IAM Identity Center가 관리하는 사용자)	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	<p>사용하고자 하는 인터페이스에 대한 지침을 따릅니다.</p> <ul style="list-style-type: none"> <li>• AWS CLI에 대해서는 AWS Command Line Interface 사용 설명서에서 <a href="#">AWS IAM Identity Center을 사용하도록 AWS CLI 구성</a>을 참조하세요.</li> <li>• AWS SDK, 도구, AWS API에 대해서는 AWS SDK 및 도구 참조 가이드에서 <a href="#">IAM Identity Center 인증</a>을 참조하세요.</li> </ul>
IAM	임시 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	IAM 사용자 설명서의 <a href="#">AWS 리소스와 함께 임시 자격 증명 사용</a> 에 나와 있는 지침을 따르세요.

프로그래밍 방식 액세스가 필요한 사용자는 누구인가요?	To	액세스 권한을 부여하는 사용자
IAM	(권장되지 않음) 장기 자격 증명을 사용하여 AWS CLI, AWS SDK 또는 AWS API에 대한 프로그래밍 요청에 서명합니다.	<p>사용하고자 하는 인터페이스에 대한 지침을 따릅니다.</p> <ul style="list-style-type: none"> <li>AWS CLI에 대해서는 AWS Command Line Interface 사용 설명서에서 <a href="#">IAM 사용자 자격 증명을 사용한 인증</a>을 참조하세요.</li> <li>AWS SDK와 도구는 AWS SDK 및 도구 참조 가이드에서 <a href="#">장기 자격 증명을 사용한 인증</a>을 참조하세요.</li> <li>AWS API는 IAM 사용자 설명서에서 <a href="#">IAM 사용자의 액세스 키 관리</a>를 참조하세요.</li> </ul>

## AWS CLI 설치 및 구성

EMR Serverless API를 사용하려면 AWS Command Line Interface의 최신 버전(AWS CLI)을 설치해야 합니다. EMR Studio 콘솔에서 EMR Serverless를 사용하는 데 AWS CLI는 필요하지 않으며, [콘솔에서 Amazon Redshift Serverless 시작하기](#)의 단계를 수행하여 CLI 없이 시작할 수 있습니다.

AWS CLI를 설정하려면

1. macOS, Linux 또는 Windows용 AWS CLI의 최신 버전을 설치하려면 [AWS CLI의 최신 버전의 설치 또는 업데이트하기](#)를 참조하세요.
2. AWS CLI를 구성하고 EMR Serverless를 포함하여 AWS 서비스에 대한 액세스 권한 설정을 보안하려면 [aws configure를 사용한 빠른 구성](#)을 참조하세요.
3. 명령 프롬프트에 다음 명령을 입력하여 설정을 확인합니다.

```
aws emr-serverless help
```

파라미터나 프로필로 설정하지 않는 한 AWS CLI 명령은 구성의 기본값 AWS 리전을 사용합니다. 파라미터로 AWS 리전을 설정하려면 각 명령에 `--region` 파라미터를 추가하면 됩니다.

프로필로 AWS 리전(을)를 설정하려면 먼저 `~/.aws/config` 파일 또는 `%UserProfile%/.aws/config` 파일에 이름이 지정된 프로필을 추가합니다(Microsoft Windows의 경우). [AWS CLI에 이름이 지정된 프로필](#)의 단계를 따르세요. 그런 다음, 다음 예제와 유사한 명령을 사용하여 AWS 리전 및 기타 설정을 지정합니다.

```
[profile emr-serverless]
aws_access_key_id = ACCESS-KEY-ID-OF-IAM-USER
aws_secret_access_key = SECRET-ACCESS-KEY-ID-OF-IAM-USER
region = us-east-1
output = text
```

## 콘솔 열기

이 섹션의 콘솔 관련 주제의 대부분은 [Amazon EMR 콘솔](#)에서 시작됩니다. 아직 AWS 계정에 로그인하지 않은 경우 로그인한 후 [Amazon EMR 콘솔](#)을 열고 다음 섹션으로 진행하여 Amazon EMR 시작하기를 계속합니다.

# Amazon EMR Serverless 시작하기

이 자습서는 샘플 Spark 또는 Hive 워크로드를 배포하는 경우 EMR Serverless를 시작하는 데 도움이 됩니다. 자체 애플리케이션을 생성, 실행 및 디버깅합니다. 이 자습서의 대부분에 나오는 기본 옵션을 표시합니다.

EMR Serverless 애플리케이션을 시작하기 전에 먼저 다음 태스크를 완료합니다.

## 주제

- [EMR Serverless를 사용할 수 있는 권한 부여](#)
- [EMR Serverless에 대한 스토리지 준비](#)
- [대화형 워크로드를 실행하도록 EMR Studio 생성](#)
- [작업 런타임 역할 생성](#)
- [콘솔에서 Amazon Redshift Serverless 시작하기](#)
- [AWS CLI에서 시작하기](#)

## EMR Serverless를 사용할 수 있는 권한 부여

EMR Serverless를 사용하려면 EMR Serverless에 대한 권한을 부여하는 정책이 연결된 사용자 또는 IAM 역할이 필요합니다. 사용자를 생성하고 해당 사용자에게 적절한 정책을 연결하려면 [권한 부여](#)의 지침을 수행합니다.

## EMR Serverless에 대한 스토리지 준비

이 자습서에서는 S3 버킷을 사용하여 EMR Serverless 애플리케이션을 사용해 실행할 샘플 Spark 또는 Hive 워크로드의 출력 파일 및 로그를 저장합니다. 버킷을 생성하려면 Amazon Simple Storage Service 콘솔 사용 설명서의 [버킷 생성](#)에 나온 지침을 따르세요. `amzn-s3-demo-bucket`에 대한 추가 참조를 새로 생성된 버킷의 이름으로 바꿉니다.

## 대화형 워크로드를 실행하도록 EMR Studio 생성

EMR Serverless를 사용하여 EMR Studio에서 호스팅되는 노트북을 통해 대화형 쿼리를 실행하려면 S3 버킷 및 [EMR Serverless에 대한 최소 서비스 역할](#) 지정하여 워크스페이스를 생성해야 합니다. 설정 단계는 Amazon EMR 관리 안내서의 [EMR Studio 설정](#)을 참조하세요. 대화형 워크로드에 대한 자세한 내용은 [EMR Studio를 통해 EMR Serverless에서 대화형 워크로드 실행](#) 섹션을 참조하세요.

## 작업 런타임 역할 생성

EMR Serverless에서 작업 실행은 런타임 시 특정 AWS 서비스 및 리소스에 대한 세분화된 권한을 제공하는 런타임 역할을 사용합니다. 이 자습서에서는 퍼블릭 S3 버킷에서 데이터 및 스크립트를 호스팅합니다. *amzn-s3-demo-bucket* 버킷은 출력을 저장합니다.

작업 런타임 역할을 설정하려면 먼저 EMR Serverless에서 새 역할을 사용할 수 있도록 신뢰 정책을 사용하여 런타임 역할을 생성합니다. 그런 다음, 필요한 S3 액세스 정책을 해당 역할에 연결합니다. 다음 단계에서 프로세스를 안내합니다.

### Console

1. <https://console.aws.amazon.com/iam/>에서 IAM 콘솔로 이동합니다.
2. 왼쪽 탐색 창에서 정책을 선택합니다.
3. 정책 생성을 선택하세요.
4. 정책 생성 페이지가 새 탭에서 열립니다. 정책 편집기를 Json으로 선택하고 아래에 정책 JSON을 붙여넣습니다.

#### Important

아래 정책에서 *amzn-s3-demo-bucket*을 [EMR Serverless에 대한 스토리지 준비](#)에서 생성한 실제 버킷 이름으로 바꿉니다. S3 액세스에 대한 기본 정책입니다. 추가 작업 런타임 역할 예제는 [Amazon EMR Serverless에 대한 작업 런타임 역할](#) 섹션을 참조하세요.

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadAccessForEMRSamples",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ]
    }
  ]
}
```

```

    "Resource": [
      "arn:aws:s3::*.elasticmapreduce",
      "arn:aws:s3::*.elasticmapreduce/*"
    ]
  },
  {
    "Sid": "FullAccessToOutputBucket",
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:GetObject",
      "s3:ListBucket",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3::amzn-s3-demo-bucket",
      "arn:aws:s3::amzn-s3-demo-bucket/*"
    ]
  },
  {
    "Sid": "GlueCreateAndReadDataCatalog",
    "Effect": "Allow",
    "Action": [
      "glue:GetDatabase",
      "glue:CreateDatabase",
      "glue:GetDataBases",
      "glue:CreateTable",
      "glue:GetTable",
      "glue:UpdateTable",
      "glue:DeleteTable",
      "glue:GetTables",
      "glue:GetPartition",
      "glue:GetPartitions",
      "glue:CreatePartition",
      "glue:BatchCreatePartition",
      "glue:GetUserDefinedFunctions"
    ],
    "Resource": [
      "*"
    ]
  }
]
}

```

5. 다음을 선택하여 정책의 이름(예: EMRServerlessS3AndGlueAccessPolicy 및 정책 만들기)을 입력합니다.
6. IAM 콘솔의 왼쪽 탐색 창에서 역할을 선택합니다.
7. 역할 생성을 선택합니다.
8. 역할 유형에서 사용자 지정 신뢰 정책을 선택하고 다음 신뢰 정책을 붙여넣습니다. 그러면 Amazon EMR Serverless 애플리케이션에 제출된 작업이 사용자를 대신하여 다른 AWS 서비스에 액세스할 수 있습니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/EMRServerlessExecutionRole",
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

9. 다음을 선택하여 권한 추가 페이지로 이동한 다음, EMRServerlessS3AndGlueAccessPolicy를 선택합니다.
10. 이름, 검토 및 생성 페이지의 역할 이름에서 역할 이름(예: EMRServerlessS3RuntimeRole)을 입력합니다. 새 IAM 역할을 생성하려면 역할 생성을 선택합니다.

## CLI

1. IAM 역할에 사용할 신뢰 정책이 포함된 emr-serverless-trust-policy.json이라는 이름의 파일을 생성합니다. 파일에 다음 정책을 포함해야 합니다.

## JSON

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "EMRServerlessTrustPolicy",
    "Action": [
      "sts:AssumeRole"
    ],
    "Effect": "Allow",
    "Resource": "arn:aws:iam::123456789012:role/
EMRServerlessExecutionRole"
  }
]
}

```

2. EMRServerlessS3RuntimeRole라는 IAM 역할을 생성합니다. 이전 단계에서 생성한 신뢰 정책을 사용합니다.

```

aws iam create-role \
  --role-name EMRServerlessS3RuntimeRole \
  --assume-role-policy-document file://emr-serverless-trust-policy.json

```

출력에서 ARN을 기록합니다. 작업 제출 중에 새 역할의 ARN을 사용하고, 이후 이를 *job-role-arn*으로 참조합니다.

3. 워크로드에 대한 IAM 정책을 정의하는 `emr-sample-access-policy.json` 파일을 생성합니다. 그러면 퍼블릭 S3 버킷에 저장된 스크립트 및 데이터에 대한 읽기 액세스와 *amzn-s3-demo-bucket*에 대한 읽기-쓰기 액세스를 제공합니다.

#### Important

아래 정책에서 *amzn-s3-demo-bucket*을 [EMR Serverless에 대한 스토리지 준비](#)에서 생성한 실제 버킷 이름으로 바꿉니다. AWS Glue 및 S3 액세스에 대한 기본 정책입니다. 추가 작업 런타임 역할 예제는 [Amazon EMR Serverless에 대한 작업 런타임 역할](#) 섹션을 참조하세요.

## JSON

```

{
  "Version": "2012-10-17",

```

```

"Statement": [
  {
    "Sid": "ReadAccessForEMRSamples",
    "Effect": "Allow",
    "Action": [
      "s3:GetObject",
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3::*.elasticmapreduce",
      "arn:aws:s3::*.elasticmapreduce/*"
    ]
  },
  {
    "Sid": "FullAccessToOutputBucket",
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:GetObject",
      "s3:ListBucket",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3::amzn-s3-demo-bucket",
      "arn:aws:s3::amzn-s3-demo-bucket/*"
    ]
  },
  {
    "Sid": "GlueCreateAndReadDataCatalog",
    "Effect": "Allow",
    "Action": [
      "glue:GetDatabase",
      "glue:CreateDatabase",
      "glue:GetDataBases",
      "glue:CreateTable",
      "glue:GetTable",
      "glue:UpdateTable",
      "glue:DeleteTable",
      "glue:GetTables",
      "glue:GetPartition",
      "glue:GetPartitions",
      "glue:CreatePartition",
      "glue:BatchCreatePartition",
      "glue:GetUserDefinedFunctions"
    ]
  }
]

```

```

    ],
    "Resource": [
        "*"
    ]
  }
]
}

```

4. 3단계에서 생성한 정책 파일을 사용하여 IAM 정책 `EMRServerlessS3AndGlueAccessPolicy`를 생성합니다. 다음 단계에서 새 정책의 ARN을 사용하므로 출력에서 ARN을 기록합니다.

```

aws iam create-policy \
  --policy-name EMRServerlessS3AndGlueAccessPolicy \
  --policy-document file://emr-sample-access-policy.json

```

출력에서 새 정책의 ARN을 기록합니다. 다음 단계에서 *policy-arn*을 대체합니다.

5. IAM 정책 `EMRServerlessS3AndGlueAccessPolicy`를 작업 런타임 역할 `EMRServerlessS3RuntimeRole`에 연결합니다.

```

aws iam attach-role-policy \
  --role-name EMRServerlessS3RuntimeRole \
  --policy-arn policy-arn

```

## 콘솔에서 Amazon Redshift Serverless 시작하기

이 섹션에서는 EMR Studio 생성을 포함하여 EMR Serverless 작업을 설명합니다. 또한 작업 실행을 제출하고 로그를 보는 방법도 설명합니다.

완료할 단계

- [1단계: EMR Serverless 애플리케이션 생성](#)
- [2단계: 작업 실행 또는 대화형 워크로드 제출](#)
- [3단계: 애플리케이션 UI 및 로그 보기](#)
- [4단계: 정리](#)

## 1단계: EMR Serverless 애플리케이션 생성

다음과 같이 EMR Serverless를 사용하여 새 애플리케이션을 생성합니다.

1. AWS Management 콘솔에 로그인한 후 <https://console.aws.amazon.com/emr>에서 Amazon EMR 콘솔을 엽니다.
2. 왼쪽 탐색 창에서 EMR Serverless를 선택하여 EMR Serverless 랜딩 페이지로 이동합니다.
3. EMR Serverless 애플리케이션을 생성하거나 관리하려면 EMR Studio UI가 필요합니다.
  - 애플리케이션을 생성하려는 AWS 리전에 이미 EMR Studio가 있는 경우 애플리케이션 관리를 선택하여 EMR Studio로 이동하거나 사용할 Studio를 선택합니다.
  - 애플리케이션을 생성하려는 AWS 리전에 EMR Studio가 없는 경우 시작하기를 선택한 다음, Studio 생성 및 시작dmf 선택합니다. EMR Serverless는 애플리케이션을 생성하고 관리할 수 있도록 EMR Studio를 생성합니다.
4. 새 탭에서 열리는 Studio 생성 UI에 애플리케이션의 이름, 유형 및 릴리스 버전을 입력합니다. 배치 작업만 실행하려면 배치 작업에만 기본 설정 사용을 선택합니다. 대화형 워크로드의 경우 대화형 워크로드에 대한 기본 설정 사용을 선택합니다. 이 옵션을 사용하여 대화형 지원 애플리케이션에서 배치 작업을 실행할 수도 있습니다. 필요한 경우 나중에 이러한 설정을 변경할 수 있습니다.

자세한 내용은 [Studio 생성](#)을 참조하세요.

5. 애플리케이션 생성을 선택하여 첫 번째 애플리케이션을 생성합니다.

작업 실행 또는 대화형 워크로드를 제출하려면 다음 섹션([2단계: 작업 실행 또는 대화형 워크로드 제출](#))으로 계속 진행합니다.

## 2단계: 작업 실행 또는 대화형 워크로드 제출

### Spark job run

이 자습서에서는 PySpark 스크립트를 사용하여 여러 텍스트 파일에서 고유한 단어가 발생하는 횟수를 계산합니다. 퍼블릭 읽기 전용 S3 버킷은 스크립트 및 데이터셋을 모두 저장합니다.

### Spark 작업을 실행하는 방법

1. 다음 명령을 사용하여 샘플 스크립트 wordcount.py를 새 버킷에 업로드합니다.

```
aws s3 cp s3://us-east-1.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py s3://amzn-s3-demo-bucket/scripts/
```

2. [1단계: EMR Serverless 애플리케이션 생성](#)을 완료하면 EMR Studio의 애플리케이션 세부 정보 페이지로 이동합니다. 여기서 작업 제출 옵션을 선택합니다.
3. 작업 제출 페이지에서 다음을 완료합니다.
  - 이름 필드에 작업 실행을 직접 호출하려는 이름을 입력합니다.
  - 런타임 역할 필드에 [작업 런타임 역할 생성](#)에서 생성한 역할의 이름을 입력합니다.
  - 스크립트 위치 필드에 `s3://amzn-s3-demo-bucket/scripts/wordcount.py`를 S3 URI로 입력합니다.
  - 스크립트 인수 필드에 `["s3://amzn-s3-demo-bucket/emr-serverless-spark/output"]`을 입력합니다.
  - Spark 속성 섹션에서 텍스트로 편집을 선택하고 다음 구성을 입력합니다.

```
--conf spark.executor.cores=1 --conf spark.executor.memory=4g --
conf spark.driver.cores=1 --conf spark.driver.memory=4g --conf
spark.executor.instances=1
```

4. 작업 실행을 시작하려면 작업 제출을 선택합니다.
5. 작업 실행 탭에서 실행 중 상태의 새 작업 실행이 표시됩니다.

## Hive job run

자습서의 이 부분에서는 테이블을 생성하고, 몇 개의 레코드를 삽입하며, 개수 집계 쿼리를 실행합니다. Hive 작업을 실행하려면 먼저 단일 작업의 일부로 실행할 모든 Hive 쿼리가 포함된 파일을 생성하고 S3에 파일을 업로드한 다음, Hive 작업을 시작할 때 이 S3 경로를 지정합니다.

### Hive 작업을 실행하는 방법

1. Hive 작업에서 실행하려는 모든 쿼리가 포함된 `hive-query.q1` 파일을 생성합니다.

```
create database if not exists emrserverless;
use emrserverless;
create table if not exists test_table(id int);
drop table if exists Values__Tmp__Table__1;
insert into test_table values (1),(2),(2),(3),(3),(3);
select id, count(id) from test_table group by id order by id desc;
```

2. 다음 명령을 사용하여 S3 버킷에 `hive-query.q1`을 업로드합니다.

```
aws s3 cp hive-query.q1 s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.q1
```

3. [1단계: EMR Serverless 애플리케이션 생성](#)을 완료하면 EMR Studio의 애플리케이션 세부 정보 페이지로 이동합니다. 여기서 작업 제출 옵션을 선택합니다.
4. 작업 제출 페이지에서 다음을 완료합니다.
  - 이름 필드에 작업 실행을 직접 호출하려는 이름을 입력합니다.
  - 런타임 역할 필드에 [작업 런타임 역할 생성](#)에서 생성한 역할의 이름을 입력합니다.
  - 스크립트 위치 필드에 `s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.q1`를 S3 URI로 입력합니다.
  - Hive 속성 섹션에서 텍스트로 편집을 선택하고 다음 구성을 입력합니다.

```
--hiveconf hive.log.explain.output=false
```

- 작업 구성 섹션에서 JSON으로 편집을 선택하고 다음 JSON을 입력합니다.

```
{
  "applicationConfiguration":
  [{
    "classification": "hive-site",
    "properties": {
      "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/scratch",
      "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
      "hive.driver.cores": "2",
      "hive.driver.memory": "4g",
      "hive.tez.container.size": "4096",
      "hive.tez.cpu.vcores": "1"
    }
  ]
}
```

5. 작업 실행을 시작하려면 작업 제출을 선택합니다.
6. 작업 실행 탭에서 실행 중 상태의 새 작업 실행이 표시됩니다.

## Interactive workload

Amazon EMR 6.14.0 이상에서 EMR Studio에서 호스팅되는 노트북을 사용하여 EMR Serverless에서 Spark에 대한 대화형 워크로드를 실행할 수 있습니다. 권한 및 사전 조건을 포함한 자세한 내용은 [EMR Studio를 통해 EMR Serverless에서 대화형 워크로드 실행](#) 섹션을 참조하세요.

애플리케이션을 생성하고 필요한 권한을 설정한 후 다음 단계를 사용하여 EMR Studio에서 대화형 노트북을 실행합니다.

1. EMR Studio의 워크스페이스 탭으로 이동합니다. 그래도 Amazon S3 스토리지 위치 및 [EMR Studio 서비스 역할](#)을 구성해야 하는 경우 화면 상단의 배너에서 Studio 구성 버튼을 선택합니다.
2. 노트북에 액세스하려면 워크스페이스를 선택하거나 새 워크스페이스를 생성합니다. 빠른 시작을 사용하여 새 탭에서 워크스페이스를 엽니다.
3. 새로 열린 탭으로 이동합니다. 왼쪽 탐색 창에서 컴퓨팅 아이콘을 선택합니다. 컴퓨팅 유형으로 EMR Serverless를 선택합니다.
4. 이전 섹션에서 생성한 대화형 지원 애플리케이션을 선택합니다.
5. 런타임 역할 필드에 EMR Serverless 애플리케이션이 작업을 위해 수입할 수 있는 IAM 역할의 이름을 입력합니다. 런타임 역할에 대한 자세한 내용은 Amazon EMR Serverless 사용 설명서에서 [작업 런타임 역할](#)을 참조하세요.
6. 연결을 선택합니다. 최대 1분이 걸릴 수 있습니다. 페이지가 연결되면 새로 고쳐집니다.
7. 커널을 선택하고 노트북을 시작합니다. EMR Serverless에서 예제 노트북을 찾아 워크스페이스에 복사할 수도 있습니다. 예제 노트북에 액세스하려면 왼쪽 탐색의 {...} 메뉴로 이동하여 노트북 파일 이름에 serverless가 있는 노트북을 탐색합니다.
8. 노트북에서 드라이버 로그 링크와 작업을 모니터링하는 지표를 제공하는 실시간 인터페이스인 Apache Spark UI에 대한 링크에 액세스할 수 있습니다. 자세한 내용은 Amazon EMR Serverless 사용 설명서의 [EMR Serverless 애플리케이션 및 작업 모니터링](#)을 참조하세요.

애플리케이션을 Studio Workspace에 연결하면 아직 실행되지 않은 경우 애플리케이션 시작이 자동으로 트리거됩니다. 애플리케이션을 사전에 시작하고 워크스페이스에 연결하기 전에 준비 상태를 유지할 수도 있습니다.

## 3단계: 애플리케이션 UI 및 로그 보기

애플리케이션 UI를 보려면 먼저 작업 실행을 식별합니다. Spark UI 또는 Hive Tez UI에 대한 옵션은 작업 유형에 따라 해당 작업 실행의 첫 번째 옵션 행에서 사용할 수 있습니다. 적절한 옵션을 선택합니다.

Spark UI를 선택한 경우 실행기 탭을 선택하여 드라이버 및 실행기 로그를 봅니다. Hive Tez UI를 선택한 경우 모든 태스크 탭을 선택하여 로그를 확인합니다.

작업 실행 상태가 성공으로 표시되면 S3 버킷에서 작업의 출력을 볼 수 있습니다.

## 4단계: 정리

15분 동안 활동이 없으면 생성한 애플리케이션은 자동으로 중지되지만 그래도 다시 사용하지 않을 리소스는 해제하는 것이 좋습니다.

애플리케이션을 삭제하려면 애플리케이션 나열 페이지로 이동합니다. 생성한 애플리케이션을 선택하고 작업 → 중지를 선택하여 애플리케이션을 중지합니다. 애플리케이션이 STOPPED 상태이면 동일한 애플리케이션을 선택하고 작업 → 삭제를 선택합니다.

Spark 및 Hive 작업 실행에 대한 추가 예제는 [EMR Serverless 작업을 실행하는 경우 Spark 구성 사용](#) 및 [EMR Serverless 작업을 실행하는 경우 Hive 구성 사용](#) 섹션을 참조하세요.

## AWS CLI에서 시작하기

AWS CLI에서 명령을 통해 EMR Serverless를 시작하여 애플리케이션을 생성하고, 작업을 실행하며, 작업 실행 출력을 확인하고, 리소스를 삭제합니다.

### 1단계: EMR Serverless 애플리케이션 생성

[emr-serverless create-application](#) 명령을 사용하여 첫 번째 EMR Serverless 애플리케이션을 생성합니다. 애플리케이션 유형 및 사용하려는 애플리케이션 버전과 연결된 Amazon EMR 릴리스 레이블을 지정해야 합니다. 애플리케이션 이름은 선택 사항입니다.

#### Spark

Spark 애플리케이션을 생성하려면 다음 명령을 실행합니다.

```
aws emr-serverless create-application \  
  --release-label emr-6.6.0 \  
  --type "SPARK" \  
  --name my-application
```

#### Hive

Hive 애플리케이션을 생성하려면 다음 명령을 실행합니다.

```
aws emr-serverless create-application \  
  --release-label emr-6.6.0 \  
  --type "HIVE" \  
  --name my-application
```

```
--release-label emr-6.6.0 \
--type "HIVE" \
--name my-application
```

출력에서 반환된 애플리케이션 ID를 기록합니다. ID를 사용하여 애플리케이션을 시작하고 작업 제출 중에 이후 이를 *application-id*로 참조합니다.

[2단계: EMR Serverless 애플리케이션에 작업 실행 제출](#) 섹션으로 진행하기 전에 애플리케이션이 [get-application](#) API를 사용하여 CREATED 상태에 도달했는지 확인합니다.

```
aws emr-serverless get-application \
--application-id application-id
```

EMR Serverless는 요청된 작업을 수용할 작업자를 생성합니다. 기본적으로 온디맨드로 생성되지만 애플리케이션을 생성할 때 `initialCapacity` 파라미터를 설정하여 사전 초기화된 용량을 지정할 수도 있습니다. 애플리케이션에서 `maximumCapacity` 파라미터와 함께 사용할 수 있는 총 최대 용량을 제한할 수도 있습니다. 이러한 옵션에 대해 자세히 알아보려면 [EMR Serverless 작업 시 애플리케이션 구성](#) 섹션을 참조하세요.

## 2단계: EMR Serverless 애플리케이션에 작업 실행 제출

이제 EMR Serverless 애플리케이션에서 작업을 실행할 준비가 되었습니다.

### Spark

이 단계에서는 PySpark 스크립트를 사용하여 여러 텍스트 파일에서 고유한 단어가 발생하는 횟수를 계산합니다. 퍼블릭 읽기 전용 S3 버킷은 스크립트 및 데이터셋을 모두 저장합니다. 애플리케이션은 Spark 런타임의 출력 파일과 로그 데이터를 사용자가 생성한 S3 버킷의 `/output` 및 `/logs` 디렉터리로 전송합니다.

#### Spark 작업을 실행하는 방법

1. 다음 명령을 사용하여 새 버킷에 실행할 샘플 스크립트를 복사합니다.

```
aws s3 cp s3://us-east-1.elasticmapreduce/emr-containers/samples/wordcount/
scripts/wordcount.py s3://amzn-s3-demo-bucket/scripts/
```

2. 다음 명령에서 *application-id*를 애플리케이션 ID로 대체합니다. *job-role-arn*을 [작업 런타임 역할 생성](#)에서 생성한 런타임 역할 ARN으로 대체합니다. *job-run-name*을 작업 실행

을 직접 호출하려는 이름으로 바꿉니다. 모든 *amzn-s3-demo-bucket* 문자열을 사용자가 생성한 Amazon S3 버킷으로 바꾸고 경로에 */output*을 추가합니다. 그러면 EMR Serverless에서 애플리케이션의 출력 파일을 복사할 수 있는 새 폴더가 버킷에 생성됩니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --name job-run-name \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/scripts/wordcount.py",
      "entryPointArguments": ["s3://amzn-s3-demo-bucket/emr-serverless-
spark/output"],
      "sparkSubmitParameters": "--conf spark.executor.cores=1
--conf spark.executor.memory=4g --conf spark.driver.cores=1 --conf
spark.driver.memory=4g --conf spark.executor.instances=1"
    }
  }'
```

3. 출력에서 반환된 작업 실행 ID를 기록합니다. 다음 단계에서 *job-run-id*를 이 ID로 바꿉니다.

## Hive

이 자습서에서는 테이블을 생성하고, 몇 개의 레코드를 삽입하며, 개수 집계 쿼리를 실행합니다. Hive 작업을 실행하려면 먼저 단일 작업의 일부로 실행할 모든 Hive 쿼리가 포함된 파일을 생성하고 S3에 파일을 업로드한 다음, Hive 작업을 시작할 때 이 S3 경로를 지정합니다.

### Hive 작업을 실행하는 방법

1. Hive 작업에서 실행하려는 모든 쿼리가 포함된 *hive-query.q1* 파일을 생성합니다.

```
create database if not exists emrserverless;
use emrserverless;
create table if not exists test_table(id int);
drop table if exists Values__Tmp__Table__1;
insert into test_table values (1),(2),(2),(3),(3),(3);
select id, count(id) from test_table group by id order by id desc;
```

2. 다음 명령을 사용하여 S3 버킷에 *hive-query.q1*을 업로드합니다.

```
aws s3 cp hive-query.q1 s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.q1
```

- 다음 명령에서 *application-id*를 자체 애플리케이션 ID로 대체합니다. *job-role-arn*을 [작업 런타임 역할 생성](#)에서 생성한 런타임 역할 ARN으로 대체합니다. 모든 *amzn-s3-demo-bucket* 문자열을 사용자가 생성한 Amazon S3 버킷으로 바꾸고 경로에 /output 및 /logs를 추가합니다. 그러면 버킷에 새 폴더가 생성되고, 여기로 EMR Serverless에서 애플리케이션의 출력 및 로그 파일을 복사할 수 있습니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.q1",
      "parameters": "--hiveconf hive.log.explain.output=false"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-serverless-
hive/hive/scratch",
        "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
        "hive.driver.cores": "2",
        "hive.driver.memory": "4g",
        "hive.tez.container.size": "4096",
        "hive.tez.cpu.vcores": "1"
      }
    }],
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://amzn-s3-demo-bucket/emr-serverless-hive/logs"
      }
    }
  }'
```

- 출력에서 반환된 작업 실행 ID를 기록합니다. 다음 단계에서 *job-run-id*를 이 ID로 바꿉니다.

### 3단계: 작업 실행의 출력 검토

작업 실행을 완료하는 데 일반적으로 3~5분이 소요됩니다.

#### Spark

다음 명령을 사용하여 Spark 작업의 상태를 확인할 수 있습니다.

```
aws emr-serverless get-job-run \  
  --application-id application-id \  
  --job-run-id job-run-id
```

로그 대상을 `s3://amzn-s3-demo-bucket/emr-serverless-spark/logs`로 설정하면 `s3://amzn-s3-demo-bucket/emr-serverless-spark/logs/applications/application-id/jobs/job-run-id`에서 이 특정 작업 실행에 대한 로그를 찾을 수 있습니다.

Spark 애플리케이션의 경우 EMR Serverless는 30초마다 이벤트 로그를 S3 로그 대상의 `sparklogs` 폴더로 푸시합니다. 작업이 완료되면 드라이버 및 실행기에 대한 Spark 런타임 로그가 `driver` 또는 `executor`와 같은 작업자 유형에 따라 적절하게 이름이 지정된 폴더에 업로드됩니다. PySpark 작업의 출력은 `s3://amzn-s3-demo-bucket/output/`에 업로드됩니다.

#### Hive

다음 명령을 사용하여 Hive 작업의 상태를 확인할 수 있습니다.

```
aws emr-serverless get-job-run \  
  --application-id application-id \  
  --job-run-id job-run-id
```

로그 대상을 `s3://amzn-s3-demo-bucket/emr-serverless-hive/logs`로 설정하면 `s3://amzn-s3-demo-bucket/emr-serverless-hive/logs/applications/application-id/jobs/job-run-id`에서 이 특정 작업 실행에 대한 로그를 찾을 수 있습니다.

Hive 애플리케이션의 경우 EMR Serverless는 S3 로그 대상의 `HIVE_DRIVER` 폴더에 Hive 드라이버, `TEZ_TASK` 폴더에 Tez 태스크 로그를 지속적으로 업로드합니다. 작업 실행이 `SUCCEEDED`

상태에 도달하면 `configurationOverrides`의 `monitoringConfiguration` 필드에 지정한 Amazon S3 위치에서 Hive 쿼리의 출력을 사용할 수 있습니다.

## 4단계: 정리

이 자습서 작업을 마치면 생성한 리소스를 삭제하는 방법을 고려합니다. 다시 사용할 의도가 없는 리소스는 해제하는 것이 좋습니다.

### 애플리케이션 삭제

애플리케이션을 삭제하려면 다음 명령을 사용합니다.

```
aws emr-serverless delete-application \  
  --application-id application-id
```

### S3 로그 버킷 삭제

S3 로깅 및 출력 버킷을 삭제하려면 다음 명령을 사용합니다. *amzn-s3-demo-bucket*을 [EMR Serverless에 대한 스토리지 준비](#)에서 생성한 S3 버킷 이름으로 바꿉니다.

```
aws s3 rm s3://amzn-s3-demo-bucket --recursive  
aws s3api delete-bucket --bucket amzn-s3-demo-bucket
```

### 작업 런타임 역할 삭제

런타임 역할을 삭제하려면 역할에서 정책을 분리합니다. 그런 다음, 역할과 정책을 모두 삭제할 수 있습니다.

```
aws iam detach-role-policy \  
  --role-name EMRServerlessS3RuntimeRole \  
  --policy-arn policy-arn
```

역할을 삭제하려면 다음 명령을 사용합니다.

```
aws iam delete-role \  
  --role-name EMRServerlessS3RuntimeRole
```

역할에 연결된 정책을 삭제하려면 다음 명령을 사용합니다.

```
aws iam delete-policy \  
  --policy-arn policy-arn
```

Spark 및 Hive 작업 실행에 대한 추가 예제는 [EMR Serverless 작업을 실행하는 경우 Spark 구성 사용](#) 및 [EMR Serverless 작업을 실행하는 경우 Hive 구성 사용](#) 섹션을 참조하세요.

# EMR Serverless 애플리케이션 구성 및 상호 작용

이 섹션에서는 AWS CLI를 사용하여 Amazon EMR Serverless 애플리케이션과 상호 작용하는 방법을 다룹니다. 또한 Spark 및 Hive 엔진에 대한 애플리케이션 구성, 사용자 지정 수행 및 기본값을 설명합니다.

## 주제

- [애플리케이션 상태](#)
- [EMR Studio 콘솔에서 EMR Serverless 애플리케이션 생성](#)
- [AWS CLI에서 EMR Serverless 애플리케이션과 상호 작용](#)
- [EMR Serverless 작업 시 애플리케이션 구성](#)
- [EMR Serverless 이미지 사용자 지정](#)
- [데이터에 연결하도록 EMR Serverless 애플리케이션에 대한 VPC 액세스 구성](#)
- [Amazon EMR Serverless 아키텍처 옵션](#)
- [EMR Serverless 애플리케이션의 작업 동시성 및 대기열 입력](#)

## 애플리케이션 상태

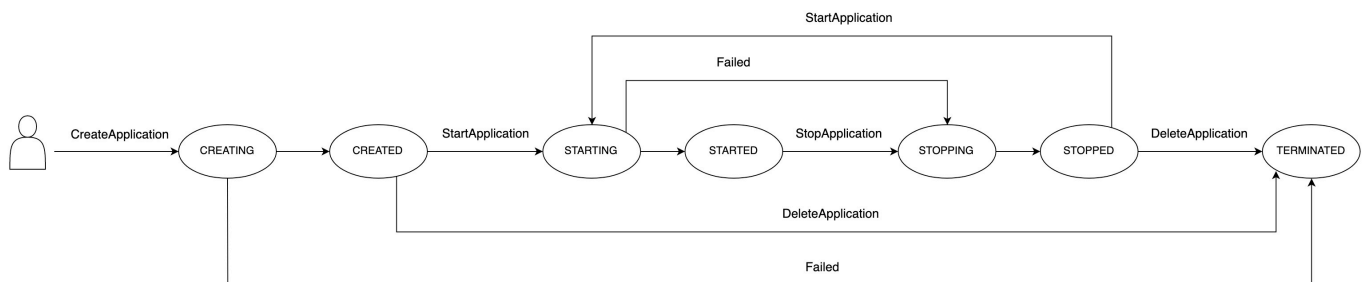
EMR Serverless를 사용하여 애플리케이션을 생성하면 애플리케이션 실행이 CREATING 상태로 전환됩니다. 그런 다음 작업은 성공(0 코드로 종료)하거나 실패(0이 아닌 코드로 종료)할 때까지 다음 상태를 통과합니다.

애플리케이션 상태는 다음과 같을 수 있습니다.

State	설명
생성 중	애플리케이션이 준비 중이며 아직 사용할 준비가 되지 않았습니다.
생성 날짜:	애플리케이션이 생성되었지만 아직 용량을 프로비저닝하지 않았습니다. 초기 용량 구성을 변경하도록 애플리케이션을 수정할 수 있습니다.
Starting(시작 중)	애플리케이션이 시작 중이며 용량을 프로비저닝하고 있습니다.

State	설명
시작됨	애플리케이션이 새 작업을 수락할 준비가 되었습니다. 애플리케이션은 이 상태일 때만 작업을 수락합니다.
Stopping	모든 작업이 완료되었으며 애플리케이션이 용량을 해제하고 있습니다.
Stopped	애플리케이션이 중지되었으며 애플리케이션에서 실행 중인 리소스가 없습니다. 초기 용량 구성을 변경하도록 애플리케이션을 수정할 수 있습니다.
종료됨	애플리케이션이 종료되었으며 애플리케이션 목록에 표시되지 않습니다.

다음 다이어그램은 EMR Serverless 애플리케이션 상태의 궤적을 보여줍니다.



## EMR Studio 콘솔에서 EMR Serverless 애플리케이션 생성

EMR Studio 콘솔에서 EMR Serverless 애플리케이션을 생성하고, 액세스하고, 관리할 수 있습니다. EMR Studio 콘솔로 이동하려면 [콘솔에서 시작하기](#)의 지침을 수행합니다.

### 애플리케이션 만들기

애플리케이션 생성 페이지에서 다음 단계를 수행하여 EMR Serverless 애플리케이션을 생성합니다.

1. 애플리케이션 이름에 애플리케이션을 직접 호출하려는 이름을 입력합니다.

2. 유형 필드에서 애플리케이션 유형으로 Spark 또는 Hive를 선택합니다.
3. 릴리스 버전 필드에서 EMR 릴리스 번호를 선택합니다.
4. 아키텍처 옵션에서 사용할 명령 세트 아키텍처를 선택합니다. 자세한 정보는 [Amazon EMR Serverless 아키텍처 옵션](#) 섹션을 참조하세요.
  - arm64 - Graviton 프로세서를 사용할 64비트 ARM 아키텍처
  - x86\_64 - x86 기반 프로세서를 사용할 64비트 x86 아키텍처
5. 나머지 필드에는 기본 설정과 사용자 지정 설정과 같은 두 가지 애플리케이션 설정 옵션이 있습니다. 이러한 필드는 선택 사항입니다.

기본 설정 - 기본 설정을 사용하면 미리 초기화된 용량으로 애플리케이션을 빠르게 생성할 수 있습니다. 여기에는 Spark용 드라이버 1개와 실행기 1개, Hive용 드라이버 1개와 Tez 태스크 1개가 포함됩니다. 기본 설정에서는 VPC에 대한 네트워크 연결을 활성화하지 않습니다. 애플리케이션은 15분 동안 유휴 상태인 경우 중지되도록 구성되며 작업 제출 시 자동으로 시작됩니다.

사용자 지정 설정 - 사용자 지정 설정을 사용하면 다음 속성을 수정할 수 있습니다.

- 사전 초기화된 용량 - 드라이버 및 실행기 수 또는 Hive Tez 태스크 작업자 수와 각 작업자의 크기.
- 애플리케이션 제한 - 애플리케이션의 최대 용량.
- 애플리케이션 동작 - 애플리케이션의 자동 시작 및 자동 중지 동작.
- 네트워크 연결 - VPC 리소스에 대한 네트워크 연결.
- 태그 - 애플리케이션에 할당하는 사용자 지정 태그.

사전 초기화된 용량, 애플리케이션 제한 및 애플리케이션 동작에 대한 자세한 내용은 [EMR Serverless 작업 시 애플리케이션 구성](#) 섹션을 참조하세요. 네트워크 연결에 대한 자세한 내용은 [데이터에 연결하도록 EMR Serverless 애플리케이션에 대한 VPC 액세스 구성](#) 섹션을 참조하세요.

6. 애플리케이션 페이지에서 애플리케이션 생성을 선택합니다.

## EMR Studio 콘솔에서 애플리케이션 나열

애플리케이션 나열 페이지에서 기존 모든 EMR Serverless 애플리케이션에 액세스할 수 있습니다. 애플리케이션 이름을 선택하여 해당 애플리케이션의 세부 정보 페이지로 이동할 수 있습니다.

## EMR Studio 콘솔에서 애플리케이션 관리

애플리케이션 나열 페이지 또는 특정 애플리케이션의 세부 정보 페이지를 통해 애플리케이션에서 다음 작업을 수행할 수 있습니다.

### 애플리케이션 시작

애플리케이션을 수동으로 시작하려면 이 옵션을 선택합니다.

### 애플리케이션 중지

애플리케이션을 수동으로 중지하려면 이 옵션을 선택합니다. 애플리케이션을 중지하려면 애플리케이션에 실행 중인 작업이 없어야 합니다. 애플리케이션 상태 전환에 대해 자세히 알아보려면 [애플리케이션 상태](#) 섹션을 참조하세요.

### 애플리케이션 구성

애플리케이션 구성 페이지에서 애플리케이션의 선택적 설정을 편집합니다. 대부분의 애플리케이션 설정을 변경할 수 있습니다. 예를 들어, 애플리케이션의 릴리스 레이블을 변경하여 다른 버전의 Amazon EMR로 업그레이드하거나 아키텍처를 x86\_64에서 arm64로 전환할 수 있습니다. 다른 선택적 설정은 애플리케이션 생성 페이지의 사용자 지정 설정 섹션에 있는 설정과 동일합니다. 애플리케이션 설정에 대한 자세한 내용은 [애플리케이션 만들기](#) 섹션을 참조하세요.

### 애플리케이션 삭제

애플리케이션을 수동으로 삭제하려면 이 옵션을 선택합니다. 애플리케이션을 삭제하려면 애플리케이션을 중지해야 합니다. 애플리케이션 상태 전환에 대해 자세히 알아보려면 [애플리케이션 상태](#) 섹션을 참조하세요.

## AWS CLI에서 EMR Serverless 애플리케이션과 상호 작용

AWS CLI에서 개별 애플리케이션을 생성, 설명 및 삭제할 수 있습니다. 한 눈에 확인할 수 있도록 모든 애플리케이션을 나열할 수도 있습니다. 이 섹션에서는 이러한 단계를 수행하는 방법을 설명합니다. 시작, 중지, 애플리케이션 업데이트 등 더 많은 애플리케이션 작업에 대해서는 [EMR Serverless API 참조](#)를 참조하세요. AWS SDK for Java를 사용하여 EMR Serverless API를 사용하는 방법에 대한 예제는 GitHub 리포지토리의 [Java 예제](#)를 참조하세요. AWS SDK for Python (Boto)을 사용하여 EMR Serverless API를 사용하는 방법에 대한 예제는 GitHub 리포지토리의 [Python examples](#)를 참조하세요.

애플리케이션을 생성하려면 create-application을 사용합니다. SPARK 또는 HIVE를 애플리케이션 type으로 지정해야 합니다. 이 명령은 애플리케이션의 ARN, 이름 및 ID를 반환합니다.

```
aws emr-serverless create-application \  
--name my-application-name \  
--type 'application-type' \  
--release-label release-version
```

애플리케이션을 설명하려면 `get-application`을 사용하고 `application-id`를 제공합니다. 이 명령은 애플리케이션의 상태 및 용량 관련 구성을 반환합니다.

```
aws emr-serverless get-application \  
--application-id application-id
```

모든 애플리케이션을 나열하려면 `list-applications`를 직접 호출합니다. 이 명령은 `get-application`과 동일한 속성을 반환하지만 모든 애플리케이션을 포함합니다.

```
aws emr-serverless list-applications
```

애플리케이션을 삭제하려면 `delete-application`을 직접 호출하고 `application-id`를 제공합니다.

```
aws emr-serverless delete-application \  
--application-id application-id
```

## EMR Serverless 작업 시 애플리케이션 구성

EMR Serverless를 사용하면 사용하는 애플리케이션을 구성할 수 있습니다. 예를 들어, 애플리케이션이 스케일 업할 수 있는 최대 용량을 설정하고, 드라이버와 작업자가 응답할 수 있도록 사전 초기화된 용량을 구성하며, 애플리케이션 수준에서 일반적인 런타임 및 모니터링 구성 세트를 지정할 수 있습니다. 다음 페이지에서는 EMR Serverless를 사용하는 경우 애플리케이션을 구성하는 방법을 설명합니다.

### 주제

- [EMR Serverless의 애플리케이션 동작 이해](#)
- [EMR Serverless에서 애플리케이션 작업을 위한 사전 초기화된 용량](#)
- [EMR Serverless에 대한 기본 애플리케이션 구성](#)

## EMR Serverless의 애플리케이션 동작 이해

이 섹션에서는 작업 제출 동작, 조정을 위한 용량 구성 및 EMR Serverless에 대한 작업자 구성 설정을 설명합니다.

### 기본 애플리케이션 동작

**자동 시작** - 기본적으로 애플리케이션은 작업 제출 시 자동 시작하도록 구성됩니다. 이 기능을 끌 수 있습니다.

**자동 중지** - 기본적으로 애플리케이션은 15분의 유휴 상태일 때 자동 중지하도록 구성됩니다. 애플리케이션이 STOPPED 상태로 변경되면 사전 초기화된 용량이 모두 해제됩니다. 애플리케이션이 자동 중지되기 전 유휴 시간을 수정하거나 이 기능을 끌 수 있습니다.

### 최대 용량

애플리케이션이 스케일 업할 수 있는 최대 용량을 구성할 수 있습니다. CPU, 메모리(GB) 및 디스크(GB) 측면에서 최대 용량을 지정할 수 있습니다.

#### Note

모범 사례는 작업자 수에 작업자 크기를 곱하여 지원되는 작업자 크기에 비례하도록 최대 용량을 구성하는 것입니다. 예를 들어, 애플리케이션을 2개의 vCPU, 16GB의 메모리, 20GB의 디스크를 사용하는 50개의 작업자로 제한하려면 최대 용량을 100개의 vCPU, 800GB의 메모리, 1,000GB의 디스크로 설정합니다.

### 지원되는 작업자 구성

다음 표는 EMR Serverless에 대해 지정할 수 있는 지원되는 워커 구성 및 크기를 나열합니다. 워크로드의 필요에 따라 드라이버 및 실행기에 대해 다양한 크기를 구성합니다.

#### 워커 구성 및 크기

CPU	메모리	기본 임시 스토리지
1 vCPU	최소 2GB, 최대 8GB, 1GB 단위로 증분	20GB~200GB
2 vCPU	최소 4GB, 최대 16GB, 1GB 단위로 증분	20GB~200GB

CPU	메모리	기본 임시 스토리지
4 vCPU	최소 8GB, 최대 30GB, 1GB 단위로 증분	20GB~200GB
8 vCPU	최소 16GB, 최대 60GB, 4GB 단위로 증분	20GB~200GB
16 vCPU	최소 32GB, 최대 120GB, 8GB 단위로 증분	20GB~200GB

CPU - 각 작업자는 1개, 2개, 4개, 8개 또는 16개의 vCPU를 보유할 수 있습니다.

메모리 - 각 작업자는 이전 표에 나열된 제한 내에서 메모리(GB로 지정됨)를 보유합니다. Spark 작업에는 메모리 오버헤드가 있습니다. 즉, 사용하는 메모리가 지정된 컨테이너 크기보다 큼니다. 이 오버헤드는 `spark.driver.memoryOverhead` 및 `spark.executor.memoryOverhead` 속성에서 지정됩니다. 오버헤드의 기본값은 컨테이너 메모리의 10%, 최소 384MB입니다. 작업자 크기를 선택하는 경우 이 오버헤드를 고려해야 합니다.

예를 들어, 워커 인스턴스에 대해 4개의 vCPU를 선택하고 사전 초기화된 스토리지 용량이 30GB인 경우 Spark 작업의 실행기 메모리로 약 27GB의 값을 설정해야 합니다. 그러면 사전 초기화된 용량의 사용률이 극대화됩니다. 사용 가능한 메모리는 27GB에 27GB의 10%(2.7GB)를 더하여 총 29.7GB입니다.

디스크 - 최소 크기가 20GB이고 최대 크기가 200GB인 임시 스토리지 디스크로 각 작업자를 구성할 수 있습니다. 작업자당 구성한 20GB를 초과하는 추가 스토리지에 대해서만 비용을 지불합니다.

## EMR Serverless에서 애플리케이션 작업을 위한 사전 초기화된 용량

EMR Serverless는 드라이버와 작업자를 사전 초기화된 상태로 유지하고 몇 초 만에 응답할 수 있도록 지원하는 선택적 기능을 제공합니다. 이 경우 애플리케이션에 대한 작업자의 워밍업을 효과적으로 생성합니다. 이 기능을 사전 초기화된 용량이라고 합니다. 이 기능을 구성하려면 애플리케이션의 `initialCapacity` 파라미터를 사전 초기화할 워커 수로 설정할 수 있습니다. 사전 초기화된 작업자 용량을 사용하여 작업이 즉시 시작됩니다. 반복 애플리케이션 및 시간에 민감한 작업을 구현하려는 경우에 적합합니다.

사전 초기화된 용량을 통해 작업 및 세션을 몇 초 안에 시작할 수 있도록 작업자의 워밍업을 준비합니다. 애플리케이션이 유휴 상태일 때도 프로비저닝된 사전 초기화된 워커에 대한 비용을 지불하게 되므로,

빠른 시작 시간의 이점을 누릴 수 있는 사용 사례에 대해 이 기능을 활성화하고 리소스의 최적 사용을 위해 크기를 조정하는 것이 좋습니다. 유휴 시 EMR Serverless 애플리케이션이 자동 종료됩니다. 예기치 않은 요금을 방지하기 위해 사전 초기화된 워커를 사용하는 경우 이 기능을 켜두는 것이 좋습니다.

작업을 제출하는 경우 `initialCapacity`의 작업자를 사용할 수 있는 경우 작업은 해당 리소스를 사용하여 실행을 시작합니다. 이러한 작업자가 이미 다른 작업에서 사용 중이거나 `initialCapacity`에서 사용할 수 있는 것보다 작업에 더 많은 리소스가 필요한 경우 애플리케이션은 추가 작업자를 요청하고 가져옵니다(해당 애플리케이션에 설정된 리소스의 최대 제한 지원). 작업이 실행을 완료하면 작업이 사용한 작업자를 해제하고 애플리케이션에 사용할 수 있는 리소스 수가 `initialCapacity`로 반환됩니다. 애플리케이션은 작업이 실행을 완료한 후에도 리소스의 `initialCapacity`를 유지 관리합니다. 작업을 더 이상 실행할 필요가 없으면 애플리케이션은 `initialCapacity`를 초과하는 여분의 리소스를 해제합니다.

사전 초기화된 용량을 사용할 수 있으며, 애플리케이션이 시작되면 이 용량을 바로 사용할 수 있습니다. 애플리케이션이 중지되면 사전 초기화된 용량이 비활성화됩니다. 요청된 사전 초기화된 용량이 생성되고 해당 용량을 사용할 준비가 된 경우에만 애플리케이션이 `STARTED` 상태로 전환됩니다. 애플리케이션이 `STARTED` 상태인 전체 시간 동안 EMR Serverless는 사전 초기화된 용량을 작업 또는 대화형 워크로드에서 사용 중이거나 사용 가능하도록 있도록 유지합니다. 이 기능은 해제되거나 실패한 컨테이너의 용량을 복원합니다. 이 경우 `InitialCapacity` 파라미터가 지정하는 작업자 수가 유지 관리됩니다. 사전 초기화된 용량이 없는 애플리케이션의 상태는 `CREATED`에서 `STARTED`로 즉시 변경될 수 있습니다.

특정 기간 사용하지 않는 경우 사전 초기화된 용량을 해제하도록 애플리케이션을 구성할 수 있습니다. 기본값은 15분입니다. 새 작업을 제출하면 중지된 애플리케이션이 자동 시작됩니다. 애플리케이션을 생성하는 경우 이러한 자동 시작 및 중지 구성을 설정하거나 애플리케이션이 `CREATED` 또는 `STOPPED` 상태인 경우에 변경할 수 있습니다.

`InitialCapacity` 수를 변경하고 각 작업자에 대해 CPU, 메모리 및 디스크와 같은 컴퓨팅 구성을 지정할 수 있습니다. 부분 수정은 불가능하므로 값을 변경할 때 모든 컴퓨팅 구성을 지정합니다. 애플리케이션이 `CREATED` 또는 `STOPPED` 상태인 경우에만 구성을 변경할 수 있습니다.

#### Note

애플리케이션의 리소스 사용을 최적화하려면 컨테이너 크기를 사전 초기화된 용량 워커 크기에 맞게 조정하는 것이 좋습니다. 예를 들어, Spark 실행기 크기를 2개 CPU로 구성하고 메모리를 8GB로 구성하지만 사전 초기화된 용량 작업자 크기가 16GB의 메모리를 사용하는 4CPU인 경우 Spark 실행기는 이 작업에 할당될 때 작업자 리소스 중 절반만 사용합니다.

## Spark 및 Hive의 사전 초기화된 용량 사용자 지정

특정 빅 데이터 프레임워크에서 실행되는 워크로드에 대해 사전 초기화된 용량을 추가로 사용자 지정할 수 있습니다. 예를 들어, Apache Spark에서 워크로드가 실행되는 경우 드라이버로 시작하는 워커 수와 실행기로 시작하는 워커 수를 지정할 수 있습니다. 마찬가지로 Apache Hive를 사용하는 경우 Hive 드라이버로 시작하는 워커 수와 Tez 작업을 실행해야 하는 워커 수를 지정할 수 있습니다.

### 사전 초기화된 용량으로 Apache Hive를 실행하는 애플리케이션 구성

다음 API 요청은 Amazon EMR 릴리스 emr-6.6.0을 기반으로 Apache Hive를 실행하는 애플리케이션을 생성합니다. 애플리케이션은 각각 vCPU 2개와 메모리 4GB를 포함하는 사전 초기화된 Hive 드라이버 5개와 각각 vCPU 4개와 메모리 8GB를 포함하는 사전 초기화된 Tez 태스크 작업자 50개로 시작합니다. 이 애플리케이션에서 Hive 쿼리가 실행되면 먼저 사전 초기화된 작업자를 사용하고 즉시 실행을 시작합니다. 사전 초기화된 모든 작업자가 사용 중이고 더 많은 Hive 작업이 제출된 경우 애플리케이션은 총 400개의 vCPU 및 1,024GB 메모리로 확장할 수 있습니다. 선택적으로 DRIVER 또는 TEZ\_TASK 작업자의 용량을 생략할 수 있습니다.

```
aws emr-serverless create-application \
  --type "HIVE" \
  --name my-application-name \
  --release-label emr-6.6.0 \
  --initial-capacity '{
    "DRIVER": {
      "workerCount": 5,
      "workerConfiguration": {
        "cpu": "2vCPU",
        "memory": "4GB"
      }
    },
    "TEZ_TASK": {
      "workerCount": 50,
      "workerConfiguration": {
        "cpu": "4vCPU",
        "memory": "8GB"
      }
    }
  }' \
  --maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
  }'
```

## 사전 초기화된 용량으로 Apache Spark를 실행하는 애플리케이션 구성

다음 API 요청은 Amazon EMR 릴리스 6.6.0을 기반으로 Apache Spark 3.2.0을 실행하는 애플리케이션을 생성합니다. 애플리케이션은 각각 vCPU 2개와 메모리 4GB를 포함하는 사전 초기화된 Spark 드라이버 5개와 각각 vCPU 4개와 메모리 8GB를 포함하는 사전 초기화된 실행기 50개로 시작합니다. 이 애플리케이션에서 Spark 작업이 실행되면 먼저 사전 초기화된 작업자를 사용하고 즉시 실행을 시작합니다. 사전 초기화된 모든 작업자가 사용 중이고 더 많은 Spark 작업이 제출된 경우 애플리케이션은 총 400개의 vCPU 및 1,024GB 메모리로 확장할 수 있습니다. DRIVER 또는 EXECUTOR의 용량을 선택적으로 생략할 수 있습니다.

### Note

Spark는 기본값이 10%인 구성 가능한 메모리 오버헤드를 드라이버 및 실행기에 요청된 메모리에 추가합니다. 작업에서 사전 초기화된 작업자를 사용하려면 초기 용량 메모리 구성이 작업 및 오버헤드에서 요청하는 메모리보다 커야 합니다.

```
aws emr-serverless create-application \
  --type "SPARK" \
  --name my-application-name \
  --release-label emr-6.6.0 \
  --initial-capacity '{
    "DRIVER": {
      "workerCount": 5,
      "workerConfiguration": {
        "cpu": "2vCPU",
        "memory": "4GB"
      }
    },
    "EXECUTOR": {
      "workerCount": 50,
      "workerConfiguration": {
        "cpu": "4vCPU",
        "memory": "8GB"
      }
    }
  }' \
  --maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
  }'
```

## EMR Serverless에 대한 기본 애플리케이션 구성

동일한 애플리케이션에서 제출한 모든 작업에 대해 애플리케이션 수준에서 일반적인 런타임 및 모니터링 구성 세트를 지정할 수 있습니다. 그러면 각 작업에 대해 동일한 구성을 제출해야 하는 경우와 관련된 추가 오버헤드가 줄어듭니다.

다음 시점에 구성을 수정할 수 있습니다.

- [작업 제출 시 애플리케이션 수준 구성을 선언합니다.](#)
- [작업 실행 중에 기본 구성을 재정의합니다.](#)

다음 섹션에서는 추가 컨텍스트에 대한 예제와 세부 정보를 제공합니다.

### 애플리케이션 수준에서 구성 선언

애플리케이션에서 제출하는 작업에 대해 애플리케이션 수준 로깅 및 런타임 구성 속성을 지정할 수 있습니다.

#### **monitoringConfiguration**

애플리케이션으로 제출하는 작업에 대한 로그 구성을 지정하려면 [monitoringConfiguration](#) 필드를 사용합니다. EMR Serverless의 로깅에 대한 자세한 내용은 [로그 저장](#) 섹션을 참조하세요.

#### **runtimeConfiguration**

spark-defaults와 같은 런타임 구성 속성을 지정하려면 runtimeConfiguration 필드에 구성 객체를 제공합니다. 이는 애플리케이션에서 제출하는 모든 작업의 기본 구성에 영향을 미칩니다. 자세한 정보는 [Hive 구성 재정의의 파라미터](#) 및 [Spark 구성 재정의의 파라미터](#) 섹션을 참조하세요.

사용 가능한 구성 분류는 특정 EMR Serverless 릴리스에 따라 다릅니다. 예를 들어, 사용자 지정 Log4j spark-driver-log4j2 및 spark-executor-log4j2에 대한 분류는 릴리스 6.8.0 이상에서만 사용할 수 있습니다. 애플리케이션별 속성 목록은 [Spark 작업 속성](#) 및 [Hive 작업 속성](#) 섹션을 참조하세요.

애플리케이션 수준에서 데이터 보호를 위해 [Apache Log4j2 속성](#), [데이터 보호를 위한 AWS Secrets Manager](#), [Java 17 런타임](#)을 구성할 수도 있습니다.

애플리케이션 수준에서 Secrets Manager 보안 암호를 전달하려면 보안 암호를 사용해 EMR Serverless 애플리케이션을 생성하거나 업데이트해야 하는 사용자 및 역할에 다음 정책을 연결합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SecretsManagerPolicy",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": [
        "arn:aws:secretsmanager:us-east-1:123456789012:secret:my-secret-name-123abc"
      ]
    },
    {
      "Sid": "KMSDecryptPolicy",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:us-east-1:123456789012:key/12345678-1234-1234-1234-123456789012"
      ]
    }
  ]
}
```

보안 암호에 대한 사용자 지정 정책 생성에 대한 자세한 내용은 AWS Secrets Manager 사용 설명서의 [AWS Secrets Manager에 대한 권한 정책 예제](#)를 참조하세요.

 Note

애플리케이션 수준에서 지정하는 runtimeConfiguration은 [StartJobRun](#) API에서 applicationConfiguration에 매핑됩니다.

## 선언 예제

다음 예제에서는 create-application을 사용하여 기본 구성을 선언하는 방법을 보여줍니다.

```
aws emr-serverless create-application \
  --release-label release-version \
  --type SPARK \
  --name my-application-name \
  --runtime-configuration '[
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.driver.cores": "4",
        "spark.executor.cores": "2",
        "spark.driver.memory": "8G",
        "spark.executor.memory": "8G",
        "spark.executor.instances": "2",

        "spark.hadoop.javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",
        "spark.hadoop.javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-  
port/db-name",
        "spark.hadoop.javax.jdo.option.ConnectionUserName": "connection-user-  
name",
        "spark.hadoop.javax.jdo.option.ConnectionPassword":
        "EMR.secret@SecretID"
      }
    },
    {
      "classification": "spark-driver-log4j2",
      "properties": {
        "rootLogger.level": "error",
        "logger.IdentifierForClass.name": "classpathForSettingLogger",
        "logger.IdentifierForClass.level": "info"
      }
    }
  ]' \
  --monitoring-configuration '{
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket/logs/app-level"
    },
    "managedPersistenceMonitoringConfiguration": {
      "enabled": false
    }
  }
```

}'

## 작업 실행 중 구성 재정의

[StartJobRun](#) API를 사용하여 애플리케이션 구성 및 모니터링 구성에 대한 구성 재정의의 지정할 수 있습니다. 그런 다음, EMR Serverless는 애플리케이션 수준과 작업 수준에서 지정한 구성을 병합하여 작업 실행을 위한 구성을 결정합니다.

병합이 수행될 때의 세분화 수준은 다음과 같습니다.

- [ApplicationConfiguration](#) - 분류 유형(예:spark-defaults).
- [MonitoringConfiguration](#) - 구성 유형(예:s3MonitoringConfiguration).

### Note

[StartJobRun](#)에서 제공하는 구성의 우선순위는 애플리케이션 수준에서 제공하는 구성을 대체합니다.

우선순위 지정에 대한 자세한 내용은 [Hive 구성 재정의의 파라미터](#) 및 [Spark 구성 재정의의 파라미터](#) 섹션을 참조하세요.

작업을 시작하는 경우 특정 구성을 지정하지 않으면 애플리케이션에서 상속됩니다. 작업 수준에서 구성을 선언하는 경우 다음 작업을 수행할 수 있습니다.

- 기존 구성 재정의 - 재정의 값으로 StartJobRun 요청에 동일한 구성 파라미터를 제공합니다.
- 추가 구성 추가 - 지정하려는 값으로 StartJobRun 요청에 새 구성 파라미터를 추가합니다.
- 기존 구성 제거 - 애플리케이션 런타임 구성을 제거하려면 제거하려는 구성의 키를 제공하고 구성에 대한 빈 {} 선언을 전달합니다. 작업 실행에 필요한 파라미터가 포함된 분류는 제거하지 않는 것이 좋습니다. 예를 들어, [Hive 작업에 필요한 속성](#)을 제거하려고 하면 작업이 실패합니다.

애플리케이션 모니터링 구성을 제거하려면 관련 구성 유형에 적합한 방법을 사용합니다.

- **cloudWatchLoggingConfiguration** - cloudWatchLogging을 제거하려면 활성화된 플래그를 false로 전달합니다.
- **managedPersistenceMonitoringConfiguration** - 관리형 지속성 설정을 제거하고 기본 활성화된 상태로 되돌리려면 구성에 대한 빈 선언({})을 전달합니다.

- **s3MonitoringConfiguration** - s3MonitoringConfiguration을 제거하려면 구성에 대한 빈 선언({})을 전달합니다.

## 재정의의 예제

다음 예제에서는 start-job-run에서 작업을 제출하는 동안 수행할 수 있는 다양한 작업을 보여줍니다.

```
aws emr-serverless start-job-run \
  --application-id your-application-id \
  --execution-role-arn your-job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
      "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket1/
wordcount_output"]
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [
      {
        // Override existing configuration for spark-defaults in the
application
        "classification": "spark-defaults",
        "properties": {
          "spark.driver.cores": "2",
          "spark.executor.cores": "1",
          "spark.driver.memory": "4G",
          "spark.executor.memory": "4G"
        }
      },
      {
        // Add configuration for spark-executor-log4j2
        "classification": "spark-executor-log4j2",
        "properties": {
          "rootLogger.level": "error",
          "logger.IdentifierForClass.name": "classpathForSettingLogger",
          "logger.IdentifierForClass.level": "info"
        }
      }
    ]
  }
```

```

        // Remove existing configuration for spark-driver-log4j2 from the
application
        "classification": "spark-driver-log4j2",
        "properties": {}
    },
    ],
    "monitoringConfiguration": {
        "managedPersistenceMonitoringConfiguration": {
            // Override existing configuration for managed persistence
            "enabled": true
        },
        "s3MonitoringConfiguration": {
            // Remove configuration of S3 monitoring
        },
        "cloudWatchLoggingConfiguration": {
            // Add configuration for CloudWatch logging
            "enabled": true
        }
    }
}
}'

```

작업 실행 시점에 [Hive 구성 재정의의 파라미터](#) 및 [Spark 구성 재정의의 파라미터](#)에서 설명하는 우선순위 재정의의 순위에 따라 다음 분류 및 구성이 적용됩니다.

- spark-defaults 분류는 작업 수준에서 지정된 속성으로 업데이트됩니다. 이 분류에 대해서는 StartJobRun에 포함된 속성만 고려합니다.
- spark-executor-log4j2 분류는 기존 분류 목록에 추가됩니다.
- spark-driver-log4j2 분류가 제거됩니다.
- managedPersistenceMonitoringConfiguration에 대한 구성은 작업 수준에서의 구성으로 업데이트됩니다.
- s3MonitoringConfiguration에 대한 구성이 제거됩니다.
- cloudWatchLoggingConfiguration에 대한 구성이 기존 모니터링 구성에 추가됩니다.

## EMR Serverless 이미지 사용자 지정

Amazon EMR 6.9.0부터 사용자 지정 이미지를 사용하여 Amazon EMR Serverless를 사용하여 애플리케이션 종속 항목과 런타임 환경을 단일 컨테이너로 패키징할 수 있습니다. 이 경우 워크로드 종속 항목을 관리하는 방법이 간소화되고 패키지 이식성이 개선됩니다. EMR Serverless 이미지를 사용자 지정하면 다음과 같은 이점이 있습니다.

- 워크로드에 최적화된 패키지를 설치하고 구성합니다. 이 패키지는 Amazon EMR 런타임 환경의 퍼블릭 배포에 널리 사용되지 않습니다.
- 로컬 개발 및 테스트를 포함하여 조직 내 현재 설정된 빌드, 테스트 및 배포 프로세스와 EMR Serverless를 통합합니다.
- 조직 내 규정 준수 및 거버넌스 요구 사항을 충족하는 확립된 보안 프로세스(예: 이미지 스캔)를 적용합니다.
- 애플리케이션에 대해 자체 JDK 및 Python 버전을 사용할 수 있습니다.

EMR Serverless는 자체 이미지를 생성하는 경우 기본으로 사용하는 이미지를 제공합니다. 기본 이미지는 이미지가 EMR Serverless와 상호 작용하기 위한 필수 jar, 구성 및 라이브러리를 제공합니다. [Amazon ECR 퍼블릭 갤러리](#)에서 기본 이미지를 찾을 수 있습니다. 애플리케이션 유형(Spark 또는 Hive) 및 릴리스 버전과 일치하는 이미지를 사용합니다. 예를 들어, Amazon EMR 릴리스 6.9.0에서 애플리케이션을 생성하는 경우 다음 이미지를 사용합니다.

유형	이미지
Spark	public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest
Hive	public.ecr.aws/emr-serverless/hive/emr-6.9.0:latest

## 사전 조건

EMR Serverless 사용자 지정 이미지를 생성하기 전에 다음 사전 조건을 완료합니다.

1. EMR Serverless 애플리케이션을 시작하는 데 사용하는 동일한 AWS 리전에서 Amazon ECR 리포지토리를 생성합니다. Amazon ECR 프라이빗 리포지토리를 생성하려면 [프라이빗 리포지토리 생성](#)을 참조하세요.
2. 사용자에게 Amazon ECR 리포지토리에 대한 액세스 권한을 부여하려면 EMR Serverless 애플리케이션을 생성하거나 이 리포지토리의 이미지로 업데이트하는 사용자 및 역할에 다음 정책을 추가합니다.

JSON

```
{
```

```

"Version": "2012-10-17",
"Statement": [
  {
    "Sid": "ECRRepositoryListGetPolicy",
    "Effect": "Allow",
    "Action": [
      "ecr:GetDownloadUrlForLayer",
      "ecr:BatchGetImage",
      "ecr:DescribeImages"
    ],
    "Resource": [
      "arn:aws:ecr:*:123456789012:repository/my-repo"
    ]
  }
]
}

```

Amazon ECR 자격 증명 기반 정책의 추가 예제는 [Amazon Elastic Container Registry 자격 증명 기반 정책 예제](#)를 참조하세요.

## 1단계: EMR Serverless 기본 이미지에서 사용자 지정 이미지 생성

먼저 기본 이미지를 사용하는 FROM 명령으로 시작하는 [Dockerfile](#)을 생성합니다. FROM 명령 이후에 이미지에 적용하려는 수정 사항을 포함합니다. 기본 이미지는 USER를 hadoop으로 자동 설정합니다. 이 설정에는 포함된 모든 수정 사항에 대한 권한이 없습니다. 해결 방법으로 USER를 root로 설정하고 이미지를 수정한 다음, USER를 hadoop:hadoop으로 다시 설정합니다. 일반적인 사용 사례에 대한 샘플을 참조하려면 [EMR Serverless에서 사용자 지정 이미지 사용](#) 섹션을 참조하세요.

```

# Dockerfile
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root
# MODIFICATIONS GO HERE

# EMRS runs the image as hadoop
USER hadoop:hadoop

```

Dockerfile을 보유한 후 다음 명령을 사용하여 이미지를 빌드합니다.

```
# build the docker image
```

```
docker build . -t aws-account-id.dkr.ecr.region.amazonaws.com/my-repository[:tag]or[@digest]
```

## 2단계: 로컬에서 이미지 검증

EMR Serverless는 기본 파일, 환경 변수 및 올바른 이미지 구성을 검증하기 위해 사용자 지정 이미지를 정적으로 확인할 수 있는 오프라인 도구를 제공합니다. 도구를 설치하고 실행하는 방법에 대한 자세한 내용은 [Amazon EMR Serverless Image CLI GitHub](#)를 참조하세요.

도구를 설치한 후 다음 명령을 실행하여 이미지를 검증합니다.

```
amazon-emr-serverless-image \
validate-image -r emr-6.9.0 -t spark \
-i aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/@digest
```

출력은 다음과 유사하게 나타납니다.

```
Amazon EMR Serverless - Image CLI
Version: 0.0.1
... Checking if docker cli is installed
... Checking Image Manifest
[INFO] Image ID: 9e2f4359cf5beb466a8a2ed047ab61c9d37786c555655fc122272758f761b41a
[INFO] Created On: 2022-12-02T07:46:42.586249984Z
[INFO] Default User Set to hadoop:hadoop : PASS
[INFO] Working Directory Set to : PASS
[INFO] Entrypoint Set to /usr/bin/entrypoint.sh : PASS
[INFO] HADOOP_HOME is set with value: /usr/lib/hadoop : PASS
[INFO] HADOOP_LIBEXEC_DIR is set with value: /usr/lib/hadoop/libexec : PASS
[INFO] HADOOP_USER_HOME is set with value: /home/hadoop : PASS
[INFO] HADOOP_YARN_HOME is set with value: /usr/lib/hadoop-yarn : PASS
[INFO] HIVE_HOME is set with value: /usr/lib/hive : PASS
[INFO] JAVA_HOME is set with value: /etc/alternatives/jre : PASS
[INFO] TEZ_HOME is set with value: /usr/lib/tez : PASS
[INFO] YARN_HOME is set with value: /usr/lib/hadoop-yarn : PASS
[INFO] File Structure Test for hadoop-files in /usr/lib/hadoop: PASS
[INFO] File Structure Test for hadoop-jars in /usr/lib/hadoop/lib: PASS
[INFO] File Structure Test for hadoop-yarn-jars in /usr/lib/hadoop-yarn: PASS
[INFO] File Structure Test for hive-bin-files in /usr/bin: PASS
[INFO] File Structure Test for hive-jars in /usr/lib/hive/lib: PASS
[INFO] File Structure Test for java-bin in /etc/alternatives/jre/bin: PASS
[INFO] File Structure Test for tez-jars in /usr/lib/tez: PASS
-----
```

Overall Custom Image Validation Succeeded.

-----

### 3단계: 이미지를 Amazon ECR 리포지토리에 업로드

다음 명령을 사용하여 Amazon ECR 리포지토리에 Amazon ECR 이미지를 푸시합니다. 이미지를 리포지토리로 푸시할 수 있는 올바른 IAM 권한이 있는지 확인합니다. 자세한 내용은 Amazon ECR 사용 설명서의 [이미지 푸시](#)를 참조하세요.

```
# login to ECR repo
aws ecr get-login-password --region region | docker login --username AWS --password-stdin aws-account-id.dkr.ecr.region.amazonaws.com

# push the docker image
docker push aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/@digest
```

### 4단계: 사용자 지정 이미지를 사용하여 애플리케이션 생성 또는 업데이트

애플리케이션을 시작하는 방법에 따라 AWS Management 콘솔 탭 또는 AWS CLI 탭을 선택한 후에 다음 단계를 완료합니다.

#### Console

1. <https://console.aws.amazon.com/lambda>에서 EMR Studio 콘솔에 로그인합니다. 애플리케이션으로 이동하거나 [애플리케이션 생성](#)의 지침을 사용하여 새 애플리케이션을 생성합니다.
2. EMR Serverless 애플리케이션을 생성하거나 업데이트할 때 사용자 지정 이미지를 지정하려면 애플리케이션 설정 옵션에서 사용자 지정 설정을 선택합니다.
3. 사용자 지정 이미지 설정 섹션에서 이 애플리케이션에서 사용자 지정 이미지 사용 확인란을 선택합니다.
4. Amazon ECR 이미지 URI를 이미지 URI 필드에 붙여 넣습니다. EMR Serverless는 애플리케이션의 모든 작업자 유형에 대해 이 이미지를 사용합니다. 또는 다른 사용자 지정 이미지를 선택하고 각 작업자 유형에 대해 다른 Amazon ECR 이미지 URI를 붙여넣을 수 있습니다.

#### CLI

- `image-configuration` 파라미터를 사용하여 애플리케이션을 생성합니다. EMR Serverless는 이 설정을 모든 작업자 유형에 적용합니다.

```
aws emr-serverless create-application \
--release-label emr-6.9.0 \
--type SPARK \
--image-configuration '{
  "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/
@digest"
}'
```

각 작업자 유형에 대해 서로 다른 이미지 설정을 사용하는 애플리케이션을 생성하려면 `worker-type-specifications` 파라미터를 사용합니다.

```
aws emr-serverless create-application \
--release-label emr-6.9.0 \
--type SPARK \
--worker-type-specifications '{
  "Driver": {
    "imageConfiguration": {
      "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-
repository:tag/@digest"
    }
  },
  "Executor" : {
    "imageConfiguration": {
      "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-
repository:tag/@digest"
    }
  }
}'
```

애플리케이션을 업데이트하려면 `image-configuration` 파라미터를 사용합니다. EMR Serverless는 이 설정을 모든 작업자 유형에 적용합니다.

```
aws emr-serverless update-application \
--application-id application-id \
--image-configuration '{
  "imageUri": "aws-account-id.dkr.ecr.region.amazonaws.com/my-repository:tag/
@digest"
}'
```

## 5단계: EMR Serverless가 사용자 지정 이미지 리포지토리에 액세스하도록 허용

Amazon ECR 리포지토리에 다음 리소스 정책을 추가하여 EMR Serverless 서비스 위탁자가 이 리포지토리에서 get, describe 및 download 요청을 사용하도록 허용합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EmrServerlessCustomImageSupport",
      "Effect": "Allow",
      "Action": [
        "ecr:BatchGetImage",
        "ecr:DescribeImages",
        "ecr:GetDownloadUrlForLayer"
      ],
      "Resource": "arn:aws:ecr:*:123456789012:repository/my-repo",
      "Condition": {
        "ArnLike": {
          "aws:SourceArn": "arn:aws:emr-serverless:*:123456789012:/applications/"
        }
      }
    }
  ]
}
```

보안 모범 사례로 aws:SourceArn 조건 키를 리포지토리 정책에 추가합니다. IAM 전역 조건 키 aws:SourceArn을 사용하면 EMR Serverless가 애플리케이션 ARN에만 리포지토리를 사용하도록 보장합니다. Amazon ECR 리포지토리 정책에 대한 자세한 내용은 [프라이빗 리포지토리 생성](#)을 참조하세요.

## 고려 사항 및 제한 사항

사용자 지정 이미지를 작업하는 경우 다음을 고려합니다.

- 애플리케이션에 대한 유형(Spark 또는 Hive) 및 릴리스 레이블(예: `emr-6.9.0`)과 일치하는 올바른 기본 이미지를 사용합니다.
- EMR Serverless는 Docker 파일의 [CMD] 또는 [ENTRYPOINT] 명령을 무시합니다. Docker 파일에서 공통 명령(예: [COPY], [RUN], [WORKDIR])을 사용합니다.
- 사용자 지정 이미지를 생성하는 경우 환경 변수 `JAVA_HOME`, `SPARK_HOME`, `HIVE_HOME`, `TEZ_HOME`을 수정해서는 안 됩니다.
- 사용자 지정 이미지의 크기는 10GB를 초과할 수 없습니다.
- Amazon EMR 기본 이미지에서 바이너리 또는 jar을 수정하는 경우 애플리케이션 또는 작업 시작에 실패할 수 있습니다.
- Amazon ECR 리포지토리는 EMR Serverless 애플리케이션을 시작하는 데 사용하는 동일한 AWS 리전에 있어야 합니다.

## 데이터에 연결하도록 EMR Serverless 애플리케이션에 대한 VPC 액세스 구성

Amazon Redshift 클러스터, Amazon RDS 데이터베이스 또는 VPC 엔드포인트가 있는 Amazon S3 버킷과 같은 VPC 내 데이터 저장소에 연결하도록 EMR Serverless 애플리케이션을 구성할 수 있습니다. EMR Serverless 애플리케이션에는 VPC 내 데이터 저장소에 대한 아웃바운드 연결이 있습니다. 기본적으로 EMR Serverless는 애플리케이션에 대한 인바운드 액세스와 아웃바운드 인터넷 액세스를 모두 차단하여 보안을 강화합니다.

### Note

애플리케이션에 외부 Hive 메타스토어 데이터베이스를 사용하려는 경우 VPC 액세스를 구성해야 합니다. 외부 Hive 메타스토어를 구성하는 방법에 대한 자세한 내용은 [메타스토어 구성](#)을 참조하세요.

## 애플리케이션 생성

애플리케이션 생성 페이지에서 사용자 지정 설정을 선택하고 EMR Serverless 애플리케이션에서 사용할 수 있는 VPC, 서브넷 및 보안 그룹을 지정합니다.

## VPC

데이터 저장소를 포함한 가상 프라이빗 클라우드(VPC) 이름을 선택합니다. 애플리케이션 생성 페이지에는 선택한 AWS 리전에 대한 모든 VPC가 나열됩니다.

## 서브넷

데이터 저장소를 포함하는 VPC 내 서브넷을 선택합니다. 애플리케이션 생성 페이지에는 VPC에 있는 데이터 저장소에 대한 모든 서브넷이 나열됩니다. 퍼블릭 및 프라이빗 서브넷 모두 지원됩니다. 프라이빗 서브넷 또는 퍼블릭 서브넷을 애플리케이션에 전달할 수 있습니다. 퍼블릭 서브넷을 사용할지 또는 프라이빗 서브넷을 사용할지 선택할 때는 몇 가지 관련 고려 사항을 알아야 합니다.

### 프라이빗 서브넷의 경우

- 연결된 라우팅 테이블에는 인터넷 게이트웨이가 없어야 합니다.
- 인터넷에 대한 아웃바운드 연결을 위해 필요한 경우 NAT 게이트웨이를 사용하여 아웃바운드 경로를 구성합니다. NAT 게이트웨이를 구성하려면 [NAT 게이트웨이](#)를 참조하세요.
- Amazon S3 연결의 경우 NAT 게이트웨이 또는 VPC 엔드포인트를 구성합니다. S3 VPC 엔드포인트를 구성하려면 [게이트웨이 엔드포인트 생성](#)을 참조하세요.
- S3 VPC 엔드포인트를 구성하고 액세스를 제어하기 위해 엔드포인트 정책을 연결한 경우, [관리형 스토리지를 사용하는 EMR Serverless의 로깅](#) 지침에 따라 EMR Serverless가 애플리케이션 로그를 저장하고 제공할 수 있는 권한을 부여하세요.
- Amazon DynamoDB와 같은 VPC 외부의 다른 AWS 서비스에 연결하려면 VPC 엔드포인트 또는 NAT 게이트웨이를 구성합니다. AWS 서비스에 대한 VPC 엔드포인트를 구성하려면 [VPC 엔드포인트 작업](#)을 참조하세요.

### Note

프라이빗 서브넷에서 Amazon EMR Serverless를 설정할 때는 Amazon S3에 대한 VPC 엔드포인트도 설정하는 것이 좋습니다. EMR Serverless 애플리케이션이 Amazon S3용 VPC 엔드포인트가 없는 프라이빗 서브넷에 있는 경우 S3 트래픽과 관련된 추가 NAT 게이트웨이 요금이 발생합니다. 이는 VPC 엔드포인트가 구성되지 않은 경우 EMR 애플리케이션과 Amazon S3 간의 트래픽이 VPC 내에 유지되지 않기 때문입니다.

### 퍼블릭 서브넷의 경우

- 이러한 값은 인터넷 게이트웨이에 대한 경로를 갖습니다.
- 아웃바운드 트래픽을 제어하려면 적절한 보안 그룹 구성을 보장합니다.

작업자는 아웃바운드 트래픽을 통해 VPC 내 데이터 저장소에 연결할 수 있습니다. 기본적으로 EMR Serverless는 보안을 개선하기 위해 워커에 대한 인바운드 액세스를 차단합니다. 이 목적은 보안을 개선하는 것입니다.

AWS Config를 사용하는 경우 EMR Serverless는 모든 작업자에 대해 탄력적 네트워크 인터페이스 항목 레코드를 생성합니다. 이 리소스와 관련된 비용을 방지하려면 AWS Config에서 `AWS::EC2::NetworkInterface`를 끄는 방법을 고려합니다.

#### Note

여러 가용 영역에서 여러 서브넷을 선택하는 것이 좋습니다. 선택한 서브넷에 따라 EMR Serverless 애플리케이션에서 시작할 수 있는 가용 영역이 결정되기 때문입니다. 각 워커는 시작되는 서브넷의 IP 주소를 사용합니다. 지정된 서브넷에 시작하려는 작업자 수에 충분한 IP 주소가 있는지 확인합니다. 서브넷 계획에 대한 자세한 내용은 [the section called “서브넷 계획 모범 사례”](#) 섹션을 참조하세요.

#### 서브넷의 고려 사항 및 제한 사항

- 퍼블릭 서브넷이 있는 EMR Serverless는 AWS Lake Formation을 지원하지 않습니다.
- 퍼블릭 서브넷에는 인바운드 트래픽이 지원되지 않습니다.

#### 보안 그룹

데이터 저장소와 통신할 수 있는 하나 이상의 보안 그룹을 선택합니다. 애플리케이션 생성 페이지에는 VPC에 있는 모든 보안 그룹이 나열됩니다. EMR Serverless는 이러한 보안 그룹을 VPC 서브넷에 연결된 탄력적 네트워크 인터페이스와 연결합니다.

#### Note

EMR Serverless 애플리케이션에 대해 별도의 보안 그룹을 생성하는 것이 좋습니다. 보안 그룹에 0.0.0.0/0 또는 ::/0 범위에서 퍼블릭 인터넷에 개방된 포트가 있는 경우 EMR Serverless는 애플리케이션을 생성/업데이트/시작할 수 없습니다. 이를 통해 보안과 격리가 강화되고 네트워크 규칙을 보다 효율적으로 관리할 수 있습니다. 예를 들어, 이는 퍼블릭 IP 주소를 가진 워

커에 대한 예상치 못한 트래픽을 차단합니다. 예를 들어, 다음 섹션의 예제에서 설명한 것처럼 Amazon Redshift 클러스터와 통신하려면 Redshift와 EMR 서버리스 보안 그룹 간의 트래픽 규칙을 정의하세요.

### Example 예제 - Amazon Redshift 클러스터와 통신

1. EMR Serverless 보안 그룹 중 하나에서 Amazon Redshift 보안 그룹에 인바운드 트래픽의 규칙을 추가합니다.

유형	프로토콜	포트 범위	소스
모든 TCP	TCP	5439	emr-serverless-security-group

2. EMR Serverless 보안 그룹 중 하나의 아웃바운드 트래픽에 대한 규칙을 추가합니다. 이 작업은 두 가지 방법으로 수행할 수 있습니다. 먼저 모든 포트에 대한 아웃바운드 트래픽을 엽니다.

유형	프로토콜	포트 범위	대상
모든 트래픽	TCP	ALL	0.0.0.0/0

또는 아웃바운드 트래픽을 Amazon Redshift 클러스터로 제한할 수 있습니다. 이는 애플리케이션이 오직 Amazon Redshift 클러스터와 통신해야 하는 경우에만 유용합니다.

유형	프로토콜	포트 범위	소스
모든 TCP	TCP	5439	redshift-security-group

## 애플리케이션 구성

애플리케이션 구성 페이지에서 기존 EMR Serverless 애플리케이션의 네트워크 구성을 변경할 수 있습니다.

## 작업 실행 세부 정보에 액세스

작업 실행 세부 정보 페이지에서 특정 실행에 대해 작업에서 사용하는 서브넷에 액세스할 수 있습니다. 작업은 지정된 서브넷에서 선택한 하나의 서브넷에서만 실행됩니다.

## 서브넷 계획 모범 사례

AWS 리소스는 Amazon VPC에서 사용 가능한 IP 주소의 하위 세트인 서브넷에서 생성됩니다. 예를 들어, /16 넷마스크가 있는 VPC에는 서브넷 마스크를 사용하여 더 작은 여러 네트워크로 분할할 수 있는 최대 65,536개의 사용 가능한 IP 주소가 있습니다. 예를 들어, /17 마스크와 사용 가능한 IP 주소 32,768개를 사용하여 이 범위를 두 서브넷으로 분할할 수 있습니다. 서브넷은 가용 영역 내에 상주하며 여러 영역에 걸쳐 있을 수 없습니다.

서브넷은 EMR Serverless 애플리케이션 조정 제한을 염두에 두고 설계되어야 합니다. 예를 들어, vCpu 워커 4개를 요청하는 애플리케이션이 있고 최대 4,000 개의 vCpu로 스케일 업할 수 있는 경우 애플리케이션에는 총 1,000개의 네트워크 인터페이스에 대해 최대 1,000개의 워커가 필요합니다. 여러 가용 영역에서 서브넷을 생성하는 것이 좋습니다. 이를 통해 EMR Serverless는 가용 영역에서 장애가 발생한 경우 드물지만 다른 가용 영역에서 작업을 재시도하거나 사전 초기화된 용량을 프로비저닝할 수 있습니다. 따라서 두 개 이상의 가용 영역에 있는 각 서브넷에는 1,000개가 넘는 사용 가능한 IP 주소가 있어야 합니다.

1,000개의 네트워크 인터페이스를 프로비저닝하려면 마스크 크기가 22 이하인 서브넷이 필요합니다. 22보다 큰 마스크는 요구 사항을 충족하지 않습니다. 예를 들어, /23의 서브넷 마스크는 512개의 IP 주소를 제공하는 반면, /22의 마스크는 1,024개를 제공하고 /21의 마스크는 2,048개의 IP 주소를 제공합니다. 다음은 다른 가용 영역에 할당할 수 있는 /16 넷마스크의 VPC에 /22 마스크를 포함하는 4개의 서브넷에 대한 예제입니다. 각 서브넷의 처음 4개 IP 주소와 마지막 IP 주소는 AWS에서 예약하기 때문에 가용 IP 주소와 사용 가능한 IP 주소 간에 5개의 차이가 있습니다.

서브넷 ID	서브넷 주소	서브넷 마스크	IP 주소 범위	사용 가능한 IP 주소	사용 가능한 IP 주소
1	10.0.0.0	255.255.252.0/22	10.0.0.0~ 10.0.3.255	1,024	1,019
2	10.0.4.0	255.255.252.0/22	10.0.4.0~ 10.0.7.255	1,024	1,019
3	10.0.8.0	255.255.252.0/22	10.0.8.0~ 10.0.11.255	1,024	1,019

서브넷 ID	서브넷 주소	서브넷 마스크	IP 주소 범위	사용 가능한 IP 주소	사용 가능한 IP 주소
4	10.0.12.0	255.255.252.0/22	10.0.12.0 ~10.0.15.255	1,024	1,019

더 큰 작업자 크기에 워크로드가 가장 적합한지 평가해야 합니다. 작업자 규모가 크면 필요한 네트워크 인터페이스가 더 적어집니다. 예를 들어, 애플리케이션 조정 제한이 4,000개 vCpu인 16개의 vCpu 워커를 사용하는 경우 네트워크 인터페이스를 프로비저닝하려면 총 250개의 가용 IP 주소에 대해 최대 250개의 워커가 필요합니다. 250개의 네트워크 인터페이스를 프로비저닝하려면 마스크 크기가 24 이하인 여러 가용 영역의 서브넷이 필요합니다. 마스크 크기가 24보다 크면 IP 주소가 250개 미만입니다.

여러 애플리케이션에서 서브넷을 공유하는 경우 각 서브넷은 모든 애플리케이션의 총 조정 제한을 염두에 두고 설계되어야 합니다. 예를 들어, vCpu 워커 4개를 요청하는 애플리케이션이 3개이고 각각 12,000개의 vCpu 계정 수준 서비스 기반 할당량으로 최대 4,000개의 vCpu로 스케일 업할 수 있는 경우 각 서브넷에는 사용 가능한 3,000개의 IP 주소가 필요합니다. 사용하려는 VPC에 IP 주소가 충분하지 않은 경우 사용 가능한 IP 주소 수를 늘립니다. 이 작업은 해당 VPC를 통해 추가 Classless Inter-Domain Routing(CIDR) 블록을 연결하여 수행할 수 있습니다. 자세한 내용을 알아보려면 Amazon VPC 사용 설명서의 [추가 IPv4 CIDR 블록을 VPC와 연결](#)을 참조하세요.

온라인에서 사용할 수 있는 많은 도구 중 하나를 사용하여 서브넷 정의를 빠르게 생성하고 사용 가능한 IP 주소 범위를 검토할 수 있습니다.

## Amazon EMR Serverless 아키텍처 옵션

Amazon EMR Serverless 애플리케이션의 명령 세트 아키텍처에 따라 애플리케이션이 작업을 실행하는 데 사용하는 프로세서 유형이 결정됩니다. Amazon EMR은 애플리케이션에 x86\_64 및 arm64와 같은 두 가지 아키텍처 옵션을 제공합니다. EMR Serverless는 최신 세대의 인스턴스가 사용 가능해지면 해당 인스턴스로 자동 업데이트되므로, 사용자의 추가 노력 없이도 애플리케이션은 최신 인스턴스를 사용할 수 있습니다.

### 주제

- [x86\\_64 아키텍처 사용](#)
- [arm64 아키텍처\(Graviton\) 사용](#)
- [Graviton 지원으로 새 애플리케이션 시작](#)
- [Graviton을 사용하도록 기존 애플리케이션 구성](#)

- [Graviton 사용 시 고려 사항](#)

## x86\_64 아키텍처 사용

x86\_64 아키텍처는 x86 64비트 또는 x64라고도 합니다. x86\_64는 EMR Serverless 애플리케이션의 기본 옵션입니다. 이 아키텍처는 x86 기반 프로세서를 사용하며, 대부분의 서드파티 도구 및 라이브러리와 호환됩니다.

대부분의 애플리케이션은 x86 하드웨어 플랫폼과 호환되며, 기본 x86\_64 아키텍처에서 성공적으로 실행할 수 있습니다. 그러나 애플리케이션이 64비트 ARM과 호환되는 경우 arm64로 전환하여 Graviton 프로세서를 사용해 성능, 컴퓨팅 성능 및 메모리를 개선할 수 있습니다. x86 아키텍처에서 동일한 크기의 인스턴스를 실행할 때보다 arm64 아키텍처에서 인스턴스를 실행하는 경우 드는 비용이 저렴합니다.

## arm64 아키텍처(Graviton) 사용

AWS Graviton 프로세서는 64비트 ARM Neoverse 코어를 사용하여 AWS에서 사용자 지정 설계되었으며, arm64 아키텍처(Arch64 또는 64비트 ARM이라고도 함)를 활용합니다. EMR Serverless에서 사용할 수 있는 AWS Graviton 프로세서 라인에 Graviton3 및 Graviton2 프로세서가 포함됩니다. 이러한 프로세서는 x86\_64 아키텍처에서 실행되는 동급 워크로드에 비해 Spark 및 Hive 워크로드에서 뛰어난 가격 대비 성능을 제공합니다. EMR Serverless는 최신 세대 프로세서로 업그레이드하기 위해 사용자 측의 노력 없이도 사용 가능한 경우 최신 세대 프로세서를 자동으로 사용합니다.

## Graviton 지원으로 새 애플리케이션 시작

다음 방법 중 하나를 사용하여 arm64 아키텍처를 사용하는 애플리케이션을 시작합니다.

### AWS CLI

AWS CLI에서 Graviton 프로세서를 사용하여 애플리케이션을 시작하려면 create-application API에서 architecture 파라미터로 ARM64를 지정합니다. 다른 파라미터에서 애플리케이션에 적합한 값을 제공합니다.

```
aws emr-serverless create-application \  
  --name my-graviton-app \  
  --release-label emr-6.8.0 \  
  --type "SPARK" \  
  --architecture "ARM64" \  
  --region us-west-2
```

## EMR Studio

EMR Studio에서 Graviton 프로세서를 사용하여 애플리케이션을 시작하려면 애플리케이션을 생성하거나 업데이트할 때 아키텍처 옵션으로 arm64를 선택합니다.

## Graviton을 사용하도록 기존 애플리케이션 구성

SDK, AWS CLI 또는 EMR Studio에서 Graviton(arm64) 아키텍처를 사용하도록 기존 Amazon EMR Serverless 애플리케이션을 구성할 수 있습니다.

기존 애플리케이션을 x86에서 arm64로 변환하는 방법

1. `architecture` 파라미터를 지원하는 [AWS CLI/SDK](#)의 최신 메이저 버전을 사용하고 있는지 확인합니다.
2. 실행 중인 작업이 없는지 확인한 다음, 애플리케이션을 중지합니다.

```
aws emr-serverless stop-application \  
  --application-id application-id \  
  --region us-west-2
```

3. Graviton을 사용하도록 애플리케이션을 업데이트하려면 `update-application` API의 `architecture` 파라미터에 대해 ARM64를 지정합니다.

```
aws emr-serverless update-application \  
  --application-id application-id \  
  --architecture 'ARM64' \  
  --region us-west-2
```

4. 이제 애플리케이션의 CPU 아키텍처가 ARM64인지 확인하려면 `get-application` API를 사용합니다.

```
aws emr-serverless get-application \  
  --application-id application-id \  
  --region us-west-2
```

5. 준비가 되면 애플리케이션을 재시작합니다.

```
aws emr-serverless start-application \  
  --application-id application-id \  
  --region us-west-2
```

## Graviton 사용 시 고려 사항

Graviton 지원을 위해 arm64를 사용하여 EMR Serverless 애플리케이션을 시작하기 전에 먼저 다음을 확인합니다.

### 라이브러리 호환성

Graviton(arm64)을 아키텍처 옵션으로 선택하는 경우 서드파티 패키지 및 라이브러리가 64비트 ARM 아키텍처와 호환되는지 확인합니다. 선택한 아키텍처와 호환되는 Python 가상 환경으로 Python 라이브러리를 패키징하는 방법에 대한 자세한 내용은 [EMR Serverless에서 Python 라이브러리 사용](#) 섹션을 참조하세요.

자세한 내용은 GitHub의 [AWS Graviton 시작하기](#) 리포지토리를 참조하세요. 이 리포지토리에는 ARM 기반 Graviton을 시작하는 데 도움이 되는 필수 리소스가 포함되어 있습니다.

## EMR Serverless 애플리케이션의 작업 동시성 및 대기열 입력

Amazon EMR 버전 7.0.0 이상부터 애플리케이션에 대한 작업 실행 대기열 제한 시간 및 동시성 구성을 지정할 수 있습니다. 이 구성을 지정하면 Amazon EMR Serverless는 먼저 작업을 대기열에 입력하고 애플리케이션의 동시성 사용률을 기반으로 실행을 시작합니다. 예를 들어, 작업 실행 동시성이 10인 경우 애플리케이션에서 한 번에 10개의 작업만 실행됩니다. 나머지 작업은 실행 중인 작업 중 하나가 종료될 때까지 대기합니다. 대기열 제한 시간에 일찍 도달하면 작업 제한 시간이 초과됩니다. 자세한 내용은 [작업 실행 상태](#)를 참조하세요.

### 동시성 및 대기열 입력의 주요 이점

작업 동시성 및 대기열 입력은 많은 작업 제출이 필요한 경우 다음과 같은 이점을 제공합니다.

- 애플리케이션 수준 용량 제한을 효율적으로 사용하기 위해 동시 실행 작업을 제어하는 데 도움이 됩니다.
- 대기열에는 구성 가능한 제한 시간 설정과 함께 갑작스러운 작업 제출 버스트가 포함될 수 있습니다.

### 동시성 및 대기열 입력 시작하기

다음 절차에서는 동시성과 대기열 입력을 구현하는 몇 가지 방법을 보여줍니다.

#### 사용AWS CLI

1. 대기열 제한 시간 및 동시 작업 실행을 사용하여 Amazon EMR Serverless 애플리케이션을 생성합니다.

```
aws emr-serverless create-application \
--release-label emr-7.0.0 \
--type SPARK \
--scheduler-configuration '{"maxConcurrentRuns": 1, "queueTimeoutMinutes": 30}'
```

2. 애플리케이션을 업데이트하여 작업 대기열 제한 시간 및 동시성을 변경합니다.

```
aws emr-serverless update-application \
--application-id application-id \
--scheduler-configuration '{"maxConcurrentRuns": 5, "queueTimeoutMinutes": 30}'
```

#### Note

기존 애플리케이션을 업데이트하여 작업 동시성 및 대기열 입력을 활성화할 수 있습니다. 이를 수행하려면 애플리케이션에서 릴리스 레이블이 emr-7.0.0 이상이어야 합니다.

## 사용AWS Management 콘솔

다음 단계에서는 AWS Management 콘솔을 사용하여 작업 동시성과 대기열 입력을 시작하는 방법을 보여줍니다.

1. EMR Studio로 이동하여 릴리스 레이블 EMR-7.0.0 이상의 애플리케이션을 생성하도록 선택합니다.
2. 애플리케이션 설정 옵션에서 사용자 지정 설정 사용 옵션을 선택합니다.
3. 추가 구성 아래에 작업 실행 설정 섹션이 있습니다. 작업 동시성 활성화 옵션을 선택하여 기능을 활성화합니다.
4. 이 옵션을 선택하면 동시 작업 실행 및 대기열 제한 시간을 선택하여 동시 작업 실행 수와 대기열 제한 시간을 각각 구성할 수 있습니다. 이러한 설정에 값을 입력하지 않으면 기본값이 사용됩니다.
5. 애플리케이션 생성을 선택하면 이 기능이 활성화된 상태로 애플리케이션이 생성됩니다. 확인하려면 대시보드로 이동하여 애플리케이션을 선택하고 속성 탭에서 해당 기능이 활성화되었는지 확인합니다.

구성 후 이 기능이 활성화된 작업을 제출합니다.

## 동시성 및 대기열 입력에 대한 고려 사항

동시성 및 대기열 입력을 구현하는 경우 다음을 고려합니다.

- 작업 대기열 및 동시성은 Amazon EMR 릴리스 7.0.0 이상에서 지원됩니다.
- 작업 동시성 및 대기열은 Amazon EMR 릴리스 7.3.0 이상에서 기본적으로 활성화되어 있습니다.
- 시작된 상태에서 애플리케이션의 동시성을 업데이트할 수 없습니다.
- `maxConcurrentRuns`의 유효 범위는 1~1000이고 `queueTimeoutMinutes`의 경우 15~720입니다.
- 계정에 대해 최대 2,000개의 작업이 대기 중 상태에 있을 수 있습니다.
- 동시성과 대기열 입력은 배치 및 스트리밍 작업에 적용됩니다. 대화형 작업에는 사용할 수 없습니다. 자세한 내용은 [EMR Studio를 통해 EMR Serverless에서 대화형 워크로드 실행](#)을 참조하세요.

# EMR Serverless를 사용하여 S3 Express One Zone으로 데이터 가져오기

Amazon EMR 릴리스 7.2.0 이상에서는 작업 및 워크로드를 실행하는 경우 성능 개선을 위해 [Amazon S3 Express One Zone](#) 스토리지 클래스와 함께 EMR Serverless를 사용할 수 있습니다. S3 Express One Zone은 대부분의 지연 시간에 민감한 애플리케이션에서 일관되게 10밀리초 미만의 데이터 액세스를 지원하는 고성능 단일 영역 Amazon S3 스토리지 클래스입니다. 릴리스 시점에 S3 Express One Zone은 Amazon S3에서 지연 시간이 가장 낮고 성능은 가장 뛰어난 클라우드 객체 스토리지를 제공합니다.

## 사전 조건

- S3 Express One Zone 권한 - S3 Express One Zone이 S3 객체에서 GET, PUT 또는 LIST과 같은 작업을 처음 수행하면 스토리지 클래스에서 사용자를 대신하여 CreateSession을 직접 호출합니다. S3A 커넥터가 CreateSession API를 간접적으로 호출할 수 있으려면 IAM 정책에서 s3express:CreateSession 권한을 허용해야 합니다. 이 권한이 있는 정책 예시는 [S3 Express One Zone 시작하기](#) 섹션을 참조하세요.
- S3A 커넥터 - S3 Express One Zone 스토리지 클래스를 사용하는 Amazon S3 버킷에서 데이터에 액세스하도록 Spark를 구성하려면 Apache Hadoop 커넥터 S3A를 사용합니다. 커넥터를 사용하려면 모든 S3 URI가 s3a 체계를 사용하는지 확인해야 합니다. 그렇지 않은 경우 s3 및 s3n 체계에 대해 사용하는 파일 시스템 구현을 변경합니다.

s3 체계를 변경하려면 다음 클러스터 구성을 지정하세요.

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]
```

s3n 체계를 변경하려면 다음 클러스터 구성을 지정하세요.

```
[
  {
    "Classification": "core-site",
    "Properties": {
      "fs.s3n.impl": "org.apache.hadoop.fs.s3a.S3AFileSystem",
      "fs.AbstractFileSystem.s3n.impl": "org.apache.hadoop.fs.s3a.S3A"
    }
  }
]
```

## S3 Express One Zone 시작하기

다음 단계를 수행하여 S3 Express One Zone을 시작합니다.

1. [VPC 엔드포인트를 생성합니다](#). VPC 엔드포인트에 `com.amazonaws.us-west-2.s3express` 엔드포인트를 추가합니다.
2. [Amazon EMR Serverless 시작하기](#)를 따라 Amazon EMR 릴리스 레이블 7.2.0 이상을 사용하여 애플리케이션을 생성합니다.
3. 새로 생성된 VPC 엔드포인트, 프라이빗 서브넷 그룹 및 보안 그룹을 사용하도록 [애플리케이션을 구성합니다](#).
4. 작업 실행 역할에 필요한 `CreateSession` 권한을 추가합니다.

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Resource": [
        "*"
      ],
      "Action": [
        "s3express:CreateSession"
      ],
      "Sid": "AllowS3EXPRESSCreatesession"
    }
  ]
}
```

## 5. 작업을 실행합니다. S3 Express One Zone 버킷에 액세스하려면 S3A 스키마를 사용합니다.

```
aws emr-serverless start-job-run \  
--application-id <application-id> \  
--execution-role-arn <job-role-arn> \  
--name <job-run-name> \  
--job-driver '{  
  "sparkSubmit": {  
  
    "entryPoint": "s3a://<DOC-EXAMPLE-BUCKET>/scripts/wordcount.py",  
    "entryPointArguments":["s3a://<DOC-EXAMPLE-BUCKET>/emr-serverless-spark/output"],  
    "sparkSubmitParameters": "--conf spark.executor.cores=4  
--conf spark.executor.memory=8g --conf spark.driver.cores=4  
--conf spark.driver.memory=8g --conf spark.executor.instances=2  
--conf spark.hadoop.fs.s3a.change.detection.mode=none  
--conf spark.hadoop.fs.s3a.endpoint.region={<AWS_REGION>}  
--conf spark.hadoop.fs.s3a.select.enabled=false  
--conf spark.sql.sources.fastS3PartitionDiscovery.enabled=false  
}'
```

## 작업 실행

애플리케이션을 프로비저닝한 후 애플리케이션에 작업을 제출합니다. 이 섹션에서는 AWS CLI를 사용하여 이러한 작업을 실행하는 방법을 다룹니다. 또한 이 섹션에서는 EMR Serverless에서 사용할 수 있는 각 애플리케이션 유형의 기본값도 식별합니다.

### 주제

- [작업 실행 상태](#)
- [유예 기간이 있는 EMR Serverless 작업 실행 취소](#)
- [EMR Studio 콘솔에서 작업 실행](#)
- [AWS CLI에서 작업 실행](#)
- [실행 IAM 정책](#)
- [서플 최적화 디스크 사용](#)
- [지속적으로 스트리밍되는 데이터를 처리하기 위한 스트리밍 작업](#)
- [EMR Serverless 작업을 실행하는 경우 Spark 구성 사용](#)
- [EMR Serverless 작업을 실행하는 경우 Hive 구성 사용](#)
- [EMR Serverless 작업 복원력](#)
- [EMR Serverless에 대한 메타스토어 구성](#)
- [EMR Serverless에서 다른 AWS 계정의 S3 데이터에 액세스](#)
- [EMR Serverless에서 오류 해결](#)

## 작업 실행 상태

Amazon EMR Serverless 작업 대기열에 작업을 제출하면 작업이 SUBMITTED 상태로 전환됩니다. 작업 상태는 FAILED, SUCCESS 또는 CANCELLING에 도달할 때까지 SUBMITTED에서 RUNNING 상태를 통과합니다.

다음은 가능한 작업 실행 상태입니다.

State	설명
제출됨	EMR Serverless에 작업 실행을 제출할 때 초기 작업 상태. 애플리케이션에 작업이 예약되기를

State	설명
	기다립니다. EMR Serverless는 우선순위를 지정하고 작업 실행을 예약합니다.
대기됨	작업 실행은 애플리케이션 수준 작업 실행 동시성이 완전히 점유되면 이 상태로 대기합니다. 동시성에 대한 자세한 내용은 <a href="#">EMR Serverless 애플리케이션의 작업 동시성 및 대기열 입력</a> 섹션을 참조하세요.
보류중	스케줄러는 애플리케이션 실행의 우선순위를 지정하고 일정을 예약하기 위해 작업 실행을 평가하고 있습니다.
예약됨	EMR Serverless는 애플리케이션에 대한 작업 실행을 예약했으며 작업을 실행할 리소스를 할당하고 있습니다.
실행 중	EMR Serverless는 작업에 처음 필요한 리소스를 할당했으며, 작업이 애플리케이션에서 실행 중입니다. Spark 애플리케이션에서 이는 Spark 드라이버 프로세스가 running 상태임을 의미합니다.
Failed	EMR Serverless가 애플리케이션에 작업 실행을 제출하지 못했거나 실패한 상태로 완료되었습니다. 이 작업 실패에 대한 자세한 내용은 StateDetails 섹션을 참조하세요.
Success	작업 실행이 완료되었습니다.
취소 중	CancelJobRun API에서 작업 실행 취소를 요청했거나 작업 실행 제한 시간이 초과되었습니다. EMR Serverless는 애플리케이션에서 작업을 취소하고 리소스를 해제하려고 합니다.
취소됨	작업 실행이 취소되었으며 사용된 리소스가 해제되었습니다.

## 유예 기간이 있는 EMR Serverless 작업 실행 취소

데이터 처리 시스템에서 갑작스러운 종료 발생하면 자원 낭비, 불완전한 작업 수행 및 잠재적인 데이터 불일치가 발생할 수 있습니다. Amazon EMR Serverless를 사용하면 작업 실행을 취소할 때 유예 기간을 지정할 수 있습니다. 이 기능을 사용하면 작업 종료 전에 진행 중인 작업을 적절하게 정리 및 완료할 수 있습니다.

작업 실행을 취소할 때 최종적으로 종료하기 전에 작업이 정리 작업을 수행할 수 있는 shutdownGracePeriodInSeconds 파라미터를 사용하여 유예 기간(초)을 지정합니다. 동작과 기본 설정은 배치 작업과 스트리밍 작업마다 매우 다릅니다.

### 배치 작업의 유예 기간

배치 작업의 경우 EMR Serverless를 사용하면 유예 기간 동안 실행되는 사용자 지정 정리 작업을 구현할 수 있습니다. 이러한 정리 작업을 애플리케이션 코드의 JVM 종료 후크의 일부로 등록할 수 있습니다.

#### 기본 동작

종료의 기본 동작은 유예 기간이 없는 종료입니다. 이는 다음 두 작업으로 구성됩니다.

- 즉시 종료
- 리소스가 즉시 릴리스됩니다.

#### 구성 옵션

정상 종료되는 설정을 지정할 수 있습니다.

- 종료 유예 기간의 유효한 범위: 15~1,800초(선택 사항)
- 즉시 종료(유예 기간 없음): 0초

### 정상 종료 활성화

배치 작업의 정상 종료를 구현하려면 다음 단계를 따르세요.

1. 사용자 지정 종료 논리가 포함된 애플리케이션 코드에 종료 후크를 추가합니다.

#### Example in Scala

```
import org.apache.hadoop.util.ShutdownHookManager
```

```
// Register shutdown hook with priority (second argument)
// Higher priority hooks run first
ShutdownHookManager.get().addShutdownHook(() => {
    logger.info("Performing cleanup operations...")
}, 100)
```

## ShutdownHookManager 사용

### Example in PySpark

```
import atexit

def cleanup():
    # Your cleanup logic here
    print("Performing cleanup operations...")

# Register the cleanup function
atexit.register(cleanup)
```

2. 작업을 취소할 때 이전에 추가된 후크가 실행될 수 있도록 유예 기간을 지정합니다.

### 예시

```
# Default (immediate termination)
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID

# With 5-minute grace period
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID \
  --shutdown-grace-period-in-seconds 300
```

## 스트리밍 작업의 유예 기간

계산에 외부 데이터 소스에서 읽거나 쓰는 작업이 포함되는 Spark Structured Streaming에서는 갑작스러운 종료로 인해 원치 않는 결과가 발생할 수 있습니다. 스트리밍 작업은 데이터를 마이크로 배치 단위로 처리하며, 이러한 작업을 중간에 중단할 경우 후속 시도에서 중복 처리가 발생할 수 있습니다. 이러한 상황은 이전 마이크로 배치의 최신 체크포인트가 기록되지 않았을 때 발생하며, 이로 인해 스트리

밍 작업이 재시작될 때 동일한 데이터가 다시 처리됩니다. 이러한 중복 처리는 컴퓨팅 리소스를 낭비할 뿐만 아니라 비즈니스 운영에도 영향을 미칠 수 있으므로 갑작스러운 시스템 중단을 방지하는 것이 매우 중요합니다.

EMR Serverless는 스트리밍 쿼리 리스너를 통한 정상 종료를 기본적으로 지원합니다. 이 작업을 수행하면 작업 종료 전에 진행 중인 마이크로 배치를 적절하게 완료할 수 있습니다. 이 서비스는 스트리밍 애플리케이션에 대한 마이크로 배치 간의 정상 종료를 자동으로 관리하여 현재 마이크로 배치가 처리를 완료하고 체크포인트가 올바르게 기록되며 종료 프로세스 중에 새로운 데이터를 수집하지 않고 스트리밍 컨텍스트가 완전히 종료되도록 합니다.

## 기본 동작

- 기본적으로 120초 유예 기간이 활성화되어 있습니다.
- 내장 스트리밍 쿼리 리스너가 정상 종료를 관리합니다.

## 구성 옵션

- 종료 유예 기간의 유효한 범위: 15~1,800초(선택 사항)
- 즉시 종료: 0초

## 정상 종료 활성화

스트리밍 작업에 대한 정상 종료를 구현하려면:

작업을 취소할 때 진행 중인 마이크로 배치가 완료될 때까지 기다리도록 유예 기간을 지정합니다.

## 예시

```
# Default graceful shutdown (120 seconds)
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID

# Custom grace period (e.g. 300 seconds)
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID \
  --shutdown-grace-period-in-seconds 300
```

```
# Immediate Termination
aws emr-serverless cancel-job-run \
  --application-id APPLICATION_ID \
  --job-run-id JOB_RUN_ID \
  --shutdown-grace-period-in-seconds 0
```

## 사용자 지정 종료 후크 추가(선택 사항)

EMR Serverless는 기본적으로 내장된 스트리밍 쿼리 리스너를 통해 정상 종료를 관리하지만, 선택적으로 개별 스트리밍 쿼리에 대해 사용자 지정 종료 논리를 구현할 수 있습니다. EMR Serverless는 정상 종료 리스너를 우선순위 60으로 등록(ShutdownHookManager 사용)합니다. 우선순위가 높은 후크가 먼저 실행되므로 우선순위가 60보다 높은 사용자 지정 정리 작업을 등록하여 EMR Serverless 종료 프로세스가 시작되기 전에 실행되도록 할 수 있습니다.

사용자 지정 후크를 추가하려면 애플리케이션 코드에 종료 후크를 추가하는 방법을 보여주는 이 주제의 첫 번째 예제를 참조하세요. 여기서 우선순위는 100이고 이는 60보다 높은 우선순위입니다. 따라서 이 종료 후크가 먼저 실행됩니다.

### Note

사용자 지정 종료 후크는 선택 사항이며, EMR Serverless에서 자동으로 처리하는 정상 종료 기능에는 필요하지 않습니다.

## 유예 기간 요금 및 배치 지속 시간

유예 기간(120초)의 기본값을 사용하는 경우:

- 배치 지속 시간이 120초 미만인 경우 배치 완료에 필요한 실제 시간에 대해서만 요금이 부과됩니다.
- 배치 지속 시간이 120초를 초과하면 최대 유예 기간(120초)에 대한 요금이 청구되지만, 강제로 종료되므로 쿼리가 정상적으로 종료되지 않을 수 있습니다.

비용을 최적화하고 정상 종료를 보장하려면:

- 배치 지속 시간이 120초를 초과하는 경우: 배치 기간과 일치하도록 유예 기간을 늘리는 것이 좋습니다.
- 배치 지속 시간이 120초 미만인 경우: 실제 처리 시간에 대해서만 요금이 부과되므로 유예 기간을 조정할 필요가 없습니다.

## 고려 사항

### 유예 기간 동작

- 유예 기간은 등록된 종료 후크가 완료될 때까지 시간을 제공합니다.
- 유예 기간보다 훨씬 앞선 경우에도 종료 후크가 완료되는 즉시 작업이 종료됩니다.
- 정리 작업이 유예 기간을 초과하면 작업이 강제로 종료됩니다.

### 서비스 동작

- 유예 기간 종료는 실행 중 상태의 작업에만 사용할 수 있습니다.
- CANCELLING 상태 중의 후속 취소 요청은 무시됩니다.
- 내부 서비스 오류로 인해 EMR Serverless가 유예 기간 종료를 시작하지 못하는 경우:
  - 서비스는 최대 2분 동안 재시도를 수행합니다.
  - 재시도가 실패하면 작업이 강제로 종료됩니다.

### 결제

작업은 유예 기간 동안 소요된 시간을 포함하여 작업이 완전히 종료될 때까지 사용된 컴퓨팅 리소스에 대해 요금이 청구됩니다.

## EMR Studio 콘솔에서 작업 실행

EMR Serverless 애플리케이션에 작업 실행을 제출하고 EMR Studio 콘솔에서 작업을 볼 수 있습니다. EMR Studio 콘솔에서 EMR Serverless 애플리케이션을 생성하거나 탐색하려면 [콘솔에서 시작하기](#)의 지침을 수행합니다.

### 작업 제출

작업 제출 페이지에서 다음과 같이 EMR Serverless 애플리케이션에 작업을 제출할 수 있습니다.

#### Spark

1. 이름 필드에 작업 실행 이름을 입력합니다.
2. 런타임 역할 필드에 EMR Serverless 애플리케이션이 작업 실행을 위해 수임할 수 있는 IAM 역할의 이름을 입력합니다. 런타임 역할에 대한 자세한 내용은 [Amazon EMR Serverless에 대한 작업 런타임 역할](#) 섹션을 참조하세요.

3. 스크립트 위치 필드에 실행하려는 스크립트 또는 JAR에 대한 Amazon S3 위치를 입력합니다. Spark 작업의 경우 스크립트는 Python(.py) 파일 또는 JAR(.jar) 파일일 수 있습니다.
4. 스크립트 위치가 JAR 파일인 경우 기본 클래스 필드에 작업의 진입점인 클래스 이름을 입력합니다.
5. (선택 사항) 나머지 필드의 값을 입력합니다.
  - 스크립트 인수 - 기본 JAR 또는 Python 스크립트에 전달할 인수를 입력합니다. 코드에서 이러한 파라미터를 읽습니다. 배열의 각 인수를 쉼표로 분리합니다.
  - Spark 속성 - Spark 속성 섹션을 확장하고 이 필드에 Spark 구성 파라미터를 입력합니다.

 Note

Spark 드라이버 및 실행기 크기를 지정하는 경우 메모리 오버헤드를 고려합니다. `spark.driver.memoryOverhead` 및 `spark.executor.memoryOverhead` 속성에서 메모리 오버헤드 값을 지정합니다. 메모리 오버헤드의 기본값은 컨테이너 메모리의 10%(최소 384MB)입니다. 실행기 메모리 및 메모리 오버헤드를 합한 값이 작업자 메모리를 초과할 수 없습니다. 예를 들어, 30GB 작업자의 최대 `spark.executor.memory`는 27GB여야 합니다.

- 작업 구성 - 이 필드에서 작업 구성을 지정합니다. 이러한 작업 구성을 사용하여 애플리케이션에 대한 구성 객체를 재정의할 수 있습니다.
  - 추가 설정 - AWS Glue Data Catalog를 메타스토어로 활성화 또는 비활성화하고 애플리케이션 로그 설정을 수정합니다. 메타스토어 구성에 대해 자세히 알아보려면 [EMR Serverless에 대한 메타스토어 구성](#) 섹션을 참조하세요. 애플리케이션 로깅 옵션에 대해 자세히 알아보려면 [로그 저장](#) 섹션을 참조하세요.
  - 태그 - 애플리케이션에 사용자 지정 태그를 할당합니다.
6. 작업 제출을 선택합니다.

## Hive

1. 이름 필드에 작업 실행 이름을 입력합니다.
2. 런타임 역할 필드에 EMR Serverless 애플리케이션이 작업 실행을 위해 수입할 수 있는 IAM 역할의 이름을 입력합니다.
3. 스크립트 위치 필드에 실행하려는 스크립트 또는 JAR에 대한 Amazon S3 위치를 입력합니다. Hive 작업의 경우 스크립트는 Hive(.sql) 파일이어야 합니다.

#### 4. (선택 사항) 나머지 필드의 값을 입력합니다.

- 초기화 스크립트 위치 - Hive 스크립트가 실행되기 전에 테이블을 초기화하는 스크립트의 위치를 입력합니다.
- Hive 속성 - Hive 속성 섹션을 확장하고 이 필드에 Hive 구성 파라미터를 입력합니다.
- 작업 구성 - 작업 구성을 지정합니다. 이러한 작업 구성을 사용하여 애플리케이션에 대한 구성 객체를 재정의할 수 있습니다. Hive 작업의 경우 `hive.exec.scratchdir` 및 `hive.metastore.warehouse.dir`은 hive-site 구성에 필요한 속성입니다.

```
{
  "applicationConfiguration": [
    {
      "classification": "hive-site",
      "configurations": [],
      "properties": {
        "hive.exec.scratchdir": "s3://DOC-EXAMPLE_BUCKET/hive/scratch",
        "hive.metastore.warehouse.dir": "s3://DOC-EXAMPLE_BUCKET/hive/warehouse"
      }
    }
  ],
  "monitoringConfiguration": {}
}
```

- 추가 설정 - AWS Glue Data Catalog를 메타스토어로 활성화 또는 비활성화하고 애플리케이션 로그 설정을 수정합니다. 메타스토어 구성에 대해 자세히 알아보려면 [EMR Serverless에 대한 메타스토어 구성](#) 섹션을 참조하세요. 애플리케이션 로깅 옵션에 대해 자세히 알아보려면 [로그 저장](#) 섹션을 참조하세요.
- 태그 - 애플리케이션에 사용자 지정 태그를 할당합니다.

#### 5. 작업 제출을 선택합니다.

## 작업 실행 액세스

애플리케이션 세부 정보 페이지의 작업 실행 탭에서 작업 실행을 보고 작업 실행에 대해 다음 작업을 수행할 수 있습니다.

**작업 취소** - RUNNING 상태인 작업 실행을 취소하려면 이 옵션을 선택합니다. 작업 실행 전환에 대해 자세히 알아보려면 [작업 실행 상태](#) 섹션을 참조하세요.

작업 복제 - 이전 작업 실행을 복제하고 다시 제출하려면 이 옵션을 선택합니다.

## AWS CLI에서 작업 실행

AWS CLI에서 개별 작업을 생성, 설명 및 삭제할 수 있습니다. 한 눈에 볼 수 있도록 모든 작업을 나열할 수도 있습니다.

새 작업을 제출하려면 `start-job-run`을 사용합니다. 작업별 속성과 함께 실행하려는 애플리케이션의 ID를 제공합니다. Spark 예제는 [EMR Serverless 작업을 실행하는 경우 Spark 구성 사용](#) 섹션을 참조하세요. Hive 예제는 [EMR Serverless 작업을 실행하는 경우 Hive 구성 사용](#) 섹션을 참조하세요. 이 명령은 `application-id`, ARN 및 새 `job-id`를 반환합니다.

각 작업 실행에는 제한 시간이 설정되어 있습니다. 작업 실행이 이 기간을 초과하면 EMR Serverless에서 자동으로 취소합니다. 기본 제한 시간은 12시간입니다. 작업 실행을 시작하는 경우 작업 요구 사항을 충족하는 값으로 이 제한 시간 설정을 구성할 수 있습니다. `executionTimeoutMinutes` 속성을 사용하여 값을 구성합니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --execution-timeout-minutes 15 \
  --job-driver '{
    "hive": {
      "query": "s3://amzn-s3-demo-bucket/scripts/create_table.sql",
      "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/hive/warehouse"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "hive.client.cores": "2",
        "hive.client.memory": "4GIB"
      }
    }]
  }'
```

작업을 설명하려면 `get-job-run`을 사용합니다. 이 명령은 새 작업에 대한 작업별 구성과 설정 용량을 반환합니다.

```
aws emr-serverless get-job-run \
--job-run-id job-id \
--application-id application-id
```

작업을 나열하려면 `list-job-runs`를 사용합니다. 이 명령은 작업 유형, 상태 및 기타 개략적인 수준의 속성을 포함하는 약식 속성 세트를 반환합니다. 전체 작업을 보지 않으려는 경우 보려는 최대 작업 수를 50개까지 지정할 수 있습니다. 다음 예제에서는 두 개의 마지막 작업 실행을 보도록 지정합니다.

```
aws emr-serverless list-job-runs \
--max-results 2 \
--application-id application-id
```

작업을 취소하려면 `cancel-job-run`을 사용합니다. 취소하려는 작업의 `application-id` 및 `job-id`를 제공합니다.

```
aws emr-serverless cancel-job-run \
--job-run-id job-id \
--application-id application-id
```

AWS CLI에서 작업을 실행하는 방법에 대한 자세한 내용은 [EMR Serverless API 참조](#)를 참조하세요.

## 실행 IAM 정책.

EMR Serverless에서 작업 실행을 제출할 때 실행 역할 외에도 실행 IAM 정책을 지정할 수 있습니다. 작업 실행에서 가정하는 결과 권한은 실행 역할과 지정된 실행 IAM 정책의 권한 교차입니다.

## 시작하기

실행 IAM 정책을 사용하는 단계:

`emr-serverless` 애플리케이션을 생성하거나 기존 애플리케이션을 사용한 다음, 다음 `aws cli`를 실행하여 인라인 IAM 정책으로 작업 실행을 시작합니다.

```
aws emr-serverless start-job-run --region us-west-2 \
--application-id application-id \
--execution-role-arn execution-role-arn \
--job-driver job-driver-options \
--execution-iam-policy '{"policy": "inline-policy"}'
```

## CLI 명령 예제

다음 정책이 시스템의 policy.json 파일에 저장되어 있는 경우:

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket",
        "arn:aws:s3:::my-test-bucket/*"
      ],
      "Sid": "AllowS3GetObject"
    }
  ]
}
```

그런 다음, 다음 AWS CLI 명령을 사용하여 이 정책으로 작업을 시작할 수 있습니다.

```
aws emr-serverless start-job-run --region us-west-2 \
  --application-id application-id \
  --execution-role-arn execution-role-arn \
  --job-driver job-driver-options \
  --execution-iam-policy '{
    "policy": "$(jq -c '. | @json' policy.json)'
  }'
```

AWS 및 고객 관리 정책을 모두 사용할 수도 있으며, ARN을 통해 지정할 수도 있습니다.

```
aws emr-serverless start-job-run --region us-west-2 \
  --application-id application-id \
  --execution-role-arn execution-role-arn \
  --job-driver job-driver-options \
  --execution-iam-policy '{
```

```

    "policyArns": [
      "arn:aws:iam::aws:policy/AmazonS3FullAccess",
      "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
    ]
  }'

```

동일한 요청에서 인라인 IAM 정책과 관리형 정책 ARN을 모두 지정할 수 있습니다.

```

aws emr-serverless start-job-run --region us-west-2 \
  --application-id application-id \
  --execution-role-arn execution-role-arn \
  --job-driver job-driver-options \
  --execution-iam-policy '{
    "policy": '$(jq -c '. | @json' policy.json)',
    "policyArns": [
      "arn:aws:iam::aws:policy/AmazonS3FullAccess",
      "arn:aws:iam::aws:policy/CloudWatchLogsFullAccess"
    ]
  }'

```

## 중요 정보

- execution-role-policy에서 policy 필드의 최대 길이는 2048자입니다.
- execution-iam-policy의 policy 필드에 지정된 인라인 IAM 정책 문자열은 json 문자열 표준을 준수해야 하며, 이전 예제와 같이 이스케이프된 줄 바꿈과 따옴표가 없어야 합니다.
- 최대 10개의 관리형 정책 ARNs 목록을 execution-iam-policy의 policyArns 필드에 값으로 지정할 수 있습니다.
- 관리ARNs는 유효한 AWS 또는 고객 관리형 정책 ARN 목록이어야 합니다. 고객 관리형 정책 ARN을 지정하면 정책은 EMR-S JobRun의 동일한 AWS 계정에 속합니다.
- 인라인 IAM 정책과 관리형 정책을 모두 사용하는 경우, 인라인 및 관리형 정책에 사용하는 일반 텍스트는 합쳐서 2,048자를 초과할 수 없습니다.
- JobRun에서 가정한 결과 권한은 실행 역할과 지정된 실행 IAM 정책의 권한 교차입니다.

## 정책 교차

작업 실행에서 가정하는 결과 권한은 실행 역할과 지정된 실행 IAM 정책의 권한 교차입니다. 즉, JobRun이 작동하려면 두 위치 모두에서 필요한 권한을 지정합니다. 그러나 인라인 정책에서 업데이트 하거나 덮어쓰지 않을 권한에 대해 추가적인 포괄 허용 문을 지정할 수 있습니다.

## 예시

다음과 같은 실행 IAM 역할 정책이 제공된 경우:

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowS3"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
      "Resource": [
        "arn:aws:logs:us-west-2:123456789012:log-group::log-stream"
      ],
      "Sid": "AllowLOGSDescribeloggroups"
    },
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:DescribeTable"
      ],
      "Resource": [
        "arn:aws:dynamodb:*:*:table/MyCompany1table"
      ],
      "Sid": "AllowDYNAMODBDescribetable"
    }
  ]
}
```

그리고 다음 인라인 IAM 정책도 있습니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::my-test-bucket/tenant1",
        "arn:aws:s3:::my-test-bucket/tenant1/*"
      ],
      "Sid": "AllowS3GetObject"
    },
    {
      "Effect": "Allow",
      "Action": [
        "logs:*",
        "dynamodb:*"
      ],
      "Resource": [
        "*"
      ],
      "Sid": "AllowLOGS"
    }
  ]
}
```

JobRun에서 가정한 결과 권한은 다음과 같습니다.

JSON

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

```

    "Effect": "Allow",
    "Action": [
        "s3:GetObject",
        "s3:ListBucket"
    ],
    "Resource": [
        "arn:aws:s3:::my-test-bucket/tenant1",
        "arn:aws:s3:::my-test-bucket/tenant1/*"
    ],
    "Sid": "AllowS3GetObject"
},
{
    "Effect": "Allow",
    "Action": [
        "logs:DescribeLogGroups"
    ],
    "Resource": [
        "arn:aws:logs:us-west-2:123456789012:log-group:log-stream"
    ],
    "Sid": "AllowLOGSDescribeLogGroups"
},
{
    "Effect": "Allow",
    "Action": [
        "dynamodb:DescribeTable"
    ],
    "Resource": [
        "arn:aws:dynamodb:*:*:table/MyCompany1table"
    ],
    "Sid": "AllowDYNAMODBDescribeTable"
}
]
}

```

## 서플 최적화 디스크 사용

Amazon EMR 릴리스 7.1.0 이상을 사용하면 Apache Spark 또는 Hive 작업을 실행할 때 서플 최적화 디스크를 사용하여 I/O 집약적 워크로드의 성능을 개선할 수 있습니다. 서플 최적화 디스크는 표준 디스크에 비해 더 높은 IOPS(초당 I/O 작업 수)를 제공하여 서플 작업 중에 데이터 이동을 더 빠르게 하고 지연 시간을 줄일 수 있습니다. 서플 최적화 디스크를 사용하면 워커당 최대 2TB의 디스크 크기를 연결할 수 있으므로 워크로드 요구 사항에 적합한 용량을 구성할 수 있습니다.

## 주요 이점

서플 최적화 디스크는 다음과 같은 이점을 제공합니다.

- 높은 IOPS 성능 - 서플 최적화 디스크는 표준 디스크보다 높은 IOPS를 제공하므로 Spark 및 Hive 작업과 기타 서플 집약적 워크로드 중에 보다 효율적이고 빠른 데이터 서플링이 가능합니다.
- 더 큰 디스크 크기 - 서플 최적화 디스크는 워크당 20GB~2TB의 디스크 크기를 지원하므로 워크로드에 따라 적절한 용량을 선택할 수 있습니다.

## 시작하기

워크플로에서 서플 최적화 디스크를 사용하려면 다음 단계를 참조하세요.

### Spark

1. 다음 명령을 사용하여 EMR Serverless 릴리스 7.1.0 애플리케이션을 생성합니다.

```
aws emr-serverless create-application \
  --type "SPARK" \
  --name my-application-name \
  --release-label emr-7.1.0 \
  --region <AWS_REGION>
```

2. spark.emr-serverless.driver.disk.type 및/또는 spark.emr-serverless.executor.disk.type 파라미터를 포함하거나 서플 최적화 디스크로 실행하도록 Spark 작업을 구성합니다. 사용 사례에 따라 하나 또는 두 파라미터를 모두 사용할 수 있습니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
      "entryPointArguments": ["1"],
      "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi
      --conf spark.executor.cores=4
      --conf spark.executor.memory=20g
      --conf spark.driver.cores=4
      --conf spark.driver.memory=8g
```

```

        --conf spark.executor.instances=1
        --conf spark.emr-serverless.executor.disk.type=shuffle_optimized"
    }
}'

```

자세한 내용은 [Spark 작업 속성](#)을 참조하세요.

## Hive

1. 다음 명령을 사용하여 EMR Serverless 릴리스 7.1.0 애플리케이션을 생성합니다.

```

aws emr-serverless create-application \
  --type "HIVE" \
  --name my-application-name \
  --release-label emr-7.1.0 \
  --region <AWS_REGION>

```

2. hive.driver.disk.type 및/또는 hive.tez.disk.type 파라미터를 포함하거나 셔플 최적화 디스크로 실행하도록 Hive 작업을 구성합니다. 사용 사례에 따라 하나 또는 두 파라미터를 모두 사용할 수 있습니다.

```

aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "s3://<DOC-EXAMPLE-BUCKET>/emr-serverless-hive/query/hive-
query.q1",
      "parameters": "--hiveconf hive.log.explain.output=false"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "hive.exec.scratchdir": "s3://<DOC-EXAMPLE-BUCKET>/emr-
serverless-hive/hive/scratch",
        "hive.metastore.warehouse.dir": "s3://<DOC-EXAMPLE-BUCKET>/emr-
serverless-hive/hive/warehouse",
        "hive.driver.cores": "2",
        "hive.driver.memory": "4g",
        "hive.tez.container.size": "4096",

```

```

        "hive.tez.cpu.vcores": "1",
        "hive.driver.disk.type": "shuffle_optimized",
        "hive.tez.disk.type": "shuffle_optimized"
    }
  ]]
}'

```

자세한 내용은 [Hive 작업 속성](#)을 참조하세요.

## 사전 초기화된 용량으로 애플리케이션 구성

Amazon EMR 릴리스 7.1.0을 기반으로 애플리케이션을 생성하려면 다음 예제를 참조하세요. 이러한 애플리케이션에는 다음과 같은 속성이 있습니다.

- 사전 초기화된 Spark 드라이버 5개(각각 vCPU 2개, 메모리 4GB, 50GB의 셔플 최적화 디스크 포함).
- 사전 초기화된 실행기 50개(각각 vCPU 4개, 메모리 8GB, 500GB의 셔플 최적화 디스크 포함).

이 애플리케이션에서 Spark 작업을 실행하면 먼저 사전 초기화된 작업자를 소비한 다음, 온디맨드 작업자를 최대 용량인 400vCPU 및 1,024GB의 메모리로 확장합니다. 선택적으로 DRIVER 또는 EXECUTOR의 용량을 생략할 수 있습니다.

## Spark

```

aws emr-serverless create-application \
  --type "SPARK" \
  --name <my-application-name> \
  --release-label emr-7.1.0 \
  --initial-capacity '{
    "DRIVER": {
      "workerCount": 5,
      "workerConfiguration": {
        "cpu": "2vCPU",
        "memory": "4GB",
        "disk": "50GB",
        "diskType": "SHUFFLE_OPTIMIZED"
      }
    },
    "EXECUTOR": {
      "workerCount": 50,
      "workerConfiguration": {
        "cpu": "4vCPU",

```

```

        "memory": "8GB",
        "disk": "500GB",
        "diskType": "SHUFFLE_OPTIMIZED"
    }
}
}' \
--maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
}'

```

## Hive

```

aws emr-serverless create-application \
--type "HIVE" \
--name <my-application-name> \
--release-label emr-7.1.0 \
--initial-capacity '{
    "DRIVER": {
        "workerCount": 5,
        "workerConfiguration": {
            "cpu": "2vCPU",
            "memory": "4GB",
            "disk": "50GB",
            "diskType": "SHUFFLE_OPTIMIZED"
        }
    },
    "EXECUTOR": {
        "workerCount": 50,
        "workerConfiguration": {
            "cpu": "4vCPU",
            "memory": "8GB",
            "disk": "500GB",
            "diskType": "SHUFFLE_OPTIMIZED"
        }
    }
}' \
--maximum-capacity '{
    "cpu": "400vCPU",
    "memory": "1024GB"
}'

```

## 지속적으로 스트리밍되는 데이터를 처리하기 위한 스트리밍 작업

EMR Serverless의 스트리밍 작업은 스트리밍 데이터를 거의 실시간으로 분석 및 처리할 수 있는 작업 모드입니다. 이러한 장기 실행 작업은 스트리밍 데이터를 폴링하고 데이터가 도착하면 지속적으로 결과를 처리합니다. 스트리밍 작업은 실시간에 가까운 분석, 사기 탐지 및 추천 엔진과 같이 실시간 데이터 처리가 필요한 태스크에 가장 적합합니다. EMR Serverless 스트리밍 작업은 기본 제공 작업 복원력, 실시간 모니터링, 향상된 로그 관리, 스트리밍 커넥터와의 통합과 같은 최적화를 제공합니다.

다음은 스트리밍 작업의 몇 가지 사용 사례입니다.

- 실시간에 가까운 분석 - Amazon EMR Serverless에서 스트리밍 작업을 사용하면 스트리밍 데이터를 거의 실시간으로 처리할 수 있으므로, 로그 데이터, 센서 데이터 또는 클릭스트림 데이터와 같은 연속 데이터 스트림에 대한 실시간 분석을 수행하여 인사이트를 도출하고 최신 정보를 기반으로 시기 적절한 결정을 내릴 수 있습니다.
- 사기 탐지 - 데이터 스트림을 분석하고 의심스러운 패턴 또는 이상이 발생할 때 이를 식별하는 경우 스트리밍 작업을 사용하여 금융 거래, 신용카드 작업 또는 온라인 활동에서 실시간에 가까운 사기 탐지를 실행할 수 있습니다.
- 추천 엔진 - 스트리밍 작업에서는 사용자 활동 데이터를 처리하고 추천 모델을 업데이트할 수 있습니다. 이를 통해 행동과 선호도에 따라 개인화된 실시간 추천이 제공됩니다.
- 소셜 미디어 분석 - 스트리밍 작업에서 트윗, 게시물, 댓글 등의 소셜 미디어 데이터를 처리할 수 있으므로, 조직은 추세를 모니터링하고, 감정을 분석하며, 브랜드 평판을 실시간에 가깝게 관리할 수 있습니다.
- 사물 인터넷(IoT) 분석 - 스트리밍 작업에서 IoT 디바이스, 센서 및 연결된 기계의 고속 데이터 스트림을 처리하고 분석할 수 있으므로 이상 탐지, 예측 유지 보수 및 기타 IoT 분석 사용 사례를 실행할 수 있습니다.
- 클릭스트림 분석 - 스트리밍 작업은 웹 사이트 또는 모바일 애플리케이션의 클릭스트림 데이터를 처리 및 분석할 수 있습니다. 이러한 데이터를 사용하는 비즈니스는 분석을 실행하여 사용자 행동에 대해 자세히 알아보고, 사용자 경험을 개인화하며, 마케팅 캠페인을 최적화할 수 있습니다.
- 로그 모니터링 및 분석 - 스트리밍 작업은 서버, 애플리케이션 또는 네트워크 디바이스의 로그 데이터를 처리할 수도 있습니다. 이를 통해 이상 탐지, 문제 해결, 시스템 상태 및 성능이 제공됩니다.

### 주요 이점

EMR Serverless에서 스트리밍 작업은 다음 요소를 조합한 작업 복원력을 자동으로 제공합니다.

- 자동 재시도 - EMR Serverless는 사용자의 수동 입력 없이 실패한 모든 작업을 자동으로 재시도합니다.

- 가용 영역(AZ) 복원력 - 원래 AZ에 문제가 발생하는 경우 EMR Serverless는 스트리밍 작업을 정상 AZ로 자동 전환합니다.
- 로그 관리:
  - 로그 교체 - 보다 효율적인 디스크 스토리지 관리를 위해 EMR Serverless는 장기 스트리밍 작업에 대한 로그를 정기적으로 교체합니다. 이렇게 하면 모든 디스크 공간을 소비할 수 있는 로그 누적을 방지합니다.
  - 로그 압축 - 관리형 지속성으로 로그 파일을 효율적으로 관리하고 최적화할 수 있습니다. 또한 압축을 통해 관리형 Spark 기록 서버를 사용하는 경우 디버그 환경을 개선합니다.

## 지원되는 데이터 소스 및 데이터 싱크

EMR Serverless는 다양한 입력 데이터 소스 및 출력 데이터 싱크와 함께 작동합니다.

- 지원되는 입력 데이터 소스 – Amazon Kinesis Data Streams, Amazon Managed Streaming for Apache Kafka, 자체 관리형 Apache Kafka 클러스터. 기본적으로 Amazon EMR 릴리스 7.1.0 이상에는 [Amazon Kinesis Data Streams 커넥터](#)가 포함되어 있으므로 추가 패키지를 빌드하거나 다운로드 하지 않아도 됩니다.
- 지원되는 출력 데이터 싱크 – AWS Glue Data Catalog 테이블, Amazon S3, Amazon Redshift, MySQL, PostgreSQL Oracle, Oracle, Microsoft SQL, Apache Iceberg, Delta Lake, Apache Hudi.

## 고려 사항 및 제한 사항

스트리밍 작업을 사용하는 경우 다음 고려 사항 및 제한 사항에 유의합니다.

- 스트리밍 작업은 [Amazon EMR 릴리스 7.1.0 이상](#)에서 지원됩니다.
- EMR Serverless에서는 스트리밍 작업이 장기간 실행될 것으로 예상하므로, 작업의 런타임을 제한하도록 실행 제한 시간을 설정할 수 없습니다.
- 스트리밍 작업은 [구조화된 스트리밍 프레임워크](#)에 빌드된 Spark 엔진과만 호환됩니다.
- EMR Serverless는 스트리밍 작업을 무기한 재시도하며, 사용자는 최대 시도 횟수를 사용자 지정할 수 없습니다. 실패한 시도 횟수가 시간당 기간에 설정된 임계치를 초과하면 작업 재시도를 중지하기 위해 스래시 방지가 자동으로 포함됩니다. 기본 임계치는 1시간 동안 실패한 시도 5회입니다. 이 임계치를 1~10회 시도로 구성할 수 있습니다. 자세한 내용은 [작업 복원력](#)을 참조하세요.
- 스트리밍 작업에는 런타임 상태 및 진행 상황을 저장하는 체크포인트가 있으므로, EMR Serverless는 최신 체크포인트에서 스트리밍 작업을 재개할 수 있습니다. 자세한 내용은 Apache Spark 설명서의 [Recovering from failures with Checkpointing](#)을 참조하세요.

## 스트리밍 작업 시작하기

스트리밍 작업을 시작하는 방법을 알아보려면 다음 지침을 참조하세요.

1. [Amazon EMR Serverless 시작하기](#)를 수행하여 애플리케이션을 생성합니다. 애플리케이션이 [Amazon EMR 릴리스 7.1.0](#) 이상을 실행해야 합니다.
2. 애플리케이션이 준비되면 다음 AWS CLI 예제와 마찬가지로 mode 파라미터를 STREAMING으로 설정하여 스트리밍 작업을 제출합니다.

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "s3://<streaming script>",
    "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
    "sparkSubmitParameters": "--conf spark.executor.cores=4
      --conf spark.executor.memory=16g
      --conf spark.driver.cores=4
      --conf spark.driver.memory=16g
      --conf spark.executor.instances=3"
  }
}'
```

## 지원되는 스트리밍 커넥터

스트리밍 커넥터는 스트리밍 소스에서 데이터를 쉽게 읽을 수 있고 스트리밍 싱크에 데이터를 쓸 수도 있습니다.

다음은 지원되는 스트리밍 커넥터입니다.

### Amazon Kinesis Data Streams 커넥터

Apache Spark용 [Amazon Kinesis Data Streams 커넥터](#)를 사용하면 Amazon Kinesis Data Streams에서 데이터를 소비하고 쓰는 스트리밍 애플리케이션 및 파이프라인을 빌드할 수 있습니다. 커넥터는 샤드당 최대 2MB/초의 전용 읽기 처리량 속도로 향상된 팬아웃 소비를 지원합니다. 기본적으로 Amazon EMR Serverless 7.1.0 이상에는 커넥터가 포함되어 있으므로, 추가 패키지를 빌드하거나 다운로드하지 않아도 됩니다. 커넥터에 대한 자세한 내용은 GitHub의 [spark-sql-kinesis-connector](#) 페이지를 참조하세요.

다음은 Kinesis Data Streams 커넥터 종속성을 사용하여 작업 실행을 시작하는 방법에 대한 예제입니다.

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "s3://<Kinesis-streaming-script>",
    "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
    "sparkSubmitParameters": "--conf spark.executor.cores=4
      --conf spark.executor.memory=16g
      --conf spark.driver.cores=4
      --conf spark.driver.memory=16g
      --conf spark.executor.instances=3
      --jars /usr/share/aws/kinesis/spark-sql-kinesis/lib/spark-streaming-
sql-kinesis-connector.jar"
  }
}'
```

Kinesis Data Streams에 연결하려면 VPC 액세스를 사용하여 EMR Serverless 애플리케이션을 구성하고 VPC 엔드포인트를 사용하여 프라이빗 액세스를 허용하거나 NAT 게이트웨이를 사용하여 퍼블릭 액세스를 얻어야 합니다. 자세한 내용은 [VPC 액세스 구성](#)을 참조하세요. 또한 필요한 데이터 스트림에 액세스하는 데 필요한 읽기 및 쓰기 권한이 작업 런타임 역할에 있는지 확인해야 합니다. 작업 런타임 역할을 구성하는 방법에 대해 자세히 알아보려면 [Amazon EMR Serverless에 대한 작업 런타임 역할](#)을 참조하세요. 필요한 모든 권한의 전체 목록은 GitHub의 [spark-sql-kinesis-connector](#) 페이지를 참조하세요.

## Apache Kafka 커넥터

Spark의 구조화된 스트리밍에 대한 Apache Kafka 커넥터는 Spark 커뮤니티의 오픈 소스 커넥터이며, Maven 리포지토리에서 사용할 수 있습니다. 이 커넥터를 사용하면 Spark의 구조화된 스트리밍 애플리케이션이 자체 관리형 Apache Kafka 및 Amazon Managed Streaming for Apache Kafka에서 데이터를 읽고 쓸 수 있습니다. 커넥터에 대한 자세한 내용은 Apache Spark 설명서의 [Structured Streaming + Kafka Integration Guide](#)를 참조하세요.

다음 예제에서는 작업 실행 요청에 Kafka 커넥터를 포함하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
```

```
--mode 'STREAMING' \
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "s3://<Kafka-streaming-script>",
    "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
    "sparkSubmitParameters": "--conf spark.executor.cores=4
      --conf spark.executor.memory=16g
      --conf spark.driver.cores=4
      --conf spark.driver.memory=16g
      --conf spark.executor.instances=3
      --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:<KAFKA_CONNECTOR_VERSION>"
  }
}'
```

Apache Kafka 커넥터 버전은 EMR Serverless 릴리스 버전 및 해당 Spark 버전에 따라 달라집니다. 올바른 Kafka 버전을 찾으려면 [Structured Streaming + Kafka Integration Guide](#)를 참조하세요.

Amazon Managed Streaming for Apache Kafka를 IAM 인증과 함께 사용하려면 Kafka 커넥터가 IAM 을 사용하여 Amazon MSK에 연결되도록 다른 종속 항목을 포함합니다. 자세한 내용은 GitHub의 [aws-msk-iam-auth repository](#)를 참조하세요. 또한 작업 런타임 역할에 필요한 IAM 권한이 있는지 확인해야 합니다. 다음 예제에서는 IAM 인증과 함께 커넥터를 사용하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--job-driver '{
  "sparkSubmit": {
    "entryPoint": "s3://<Kafka-streaming-script>",
    "entryPointArguments": ["s3://<DOC-EXAMPLE-BUCKET-OUTPUT>/output"],
    "sparkSubmitParameters": "--conf spark.executor.cores=4
      --conf spark.executor.memory=16g
      --conf spark.driver.cores=4
      --conf spark.driver.memory=16g
      --conf spark.executor.instances=3
      --packages org.apache.spark:spark-sql-
kafka-0-10_2.12:<KAFKA_CONNECTOR_VERSION>,software.amazon.msk:aws-msk-iam-
auth:<MSK_IAM_LIB_VERSION>"
  }
}'
```

Amazon MSK의 Kafka 커넥터 및 IAM 인증 라이브러리를 사용하려면 VPC 액세스를 통해 EMR Serverless 애플리케이션을 구성합니다. 서브넷에는 인터넷 액세스 권한이 있어야 하며, NAT Gateway를 사용하여 Maven 종속 항목에 액세스해야 합니다. 자세한 내용은 [VPC 액세스 구성](#)을 참조하세요. Kafka 클러스터에 액세스하려면 서브넷에 네트워크 연결이 있어야 합니다. 이는 Kafka 클러스터가 자체 관리형인지 여부에 상관없이 Amazon Managed Streaming for Apache Kafka를 사용하는 경우에도 적용됩니다.

## 스트리밍 작업 로그 관리

스트리밍 작업은 Spark 애플리케이션 로그 및 이벤트 로그의 로그 교체와 Spark 이벤트 로그의 로그 압축을 지원합니다. 이 경우 리소스를 효과적으로 관리하는 데 도움이 됩니다.

### 로그 교체

스트리밍 작업은 Spark 애플리케이션 로그 및 이벤트 로그에 대한 로그 교체를 지원합니다. 로그 교체는 장기 스트리밍 작업에서 사용 가능한 모든 디스크 공간을 차지할 수 있는 대형 로그 파일 생성을 방지합니다. 로그 교체를 사용하면 디스크 스토리지를 저장하고 디스크 공간 부족으로 인한 작업 실패를 방지할 수 있습니다. 자세한 정보를 알아보려면 [로그 교체](#)를 참조하세요.

### 로그 압축

스트리밍 작업은 관리형 로깅이 사용 가능한 경우 항상 Spark 이벤트 로그에 대한 로그 압축도 지원합니다. 관리형 로깅에 대한 자세한 내용은 [관리형 스토리지를 사용하는 로깅](#)을 참조하세요. 스트리밍 작업은 장기간 실행될 수 있으며, 이벤트 데이터의 양은 시간이 지남에 따라 누적되어 로그 파일 크기가 크게 증가할 수 있습니다. Spark 기록 서버는 이러한 이벤트를 읽고 Spark 애플리케이션 UI의 메모리에 로드합니다. 이 프로세스는 특히 Amazon S3에 저장된 이벤트 로그가 매우 큰 경우 지연 시간과 비용을 높일 수 있습니다.

로그 압축은 이벤트 로그 크기를 줄이므로, Spark 기록 서버는 언제든지 1GB를 초과하는 이벤트 로그를 로드하지 않아도 됩니다. 자세한 내용은 Apache Spark 설명서의 [Monitoring and Instrumentation](#)을 참조하세요.

## EMR Serverless 작업을 실행하는 경우 Spark 구성 사용

type 파라미터가 SPARK로 설정된 애플리케이션에서 Spark 작업을 실행할 수 있습니다. 작업은 Amazon EMR 릴리스 버전과 호환되는 Spark 버전과 호환되어야 합니다. 예를 들어, Amazon EMR 릴리스 6.6.0을 사용하여 작업을 실행할 때 작업은 Apache Spark 3.2.0과 호환되어야 합니다. 각 릴리스의 애플리케이션 버전에 대한 자세한 내용은 [Amazon EMR Serverless 릴리스 버전](#) 섹션을 참조하세요.

## Spark 작업 파라미터

[StartJobRun API](#)를 사용하여 Spark 작업을 실행하는 경우 다음 파라미터를 지정할 수 있습니다.

### 필수 파라미터

- [Spark 작업 런타임 역할](#)
- [Spark 작업 드라이버 파라미터](#)
- [Spark 구성 재정의 파라미터](#)
- [Spark 동적 리소스 할당 최적화](#)

### Spark 작업 런타임 역할

**executionRoleArn**을 사용하여 애플리케이션이 Spark 작업을 실행하는 데 사용하는 IAM 역할에 대한 ARN을 지정합니다. 이 역할에는 다음 권한이 있어야 합니다.

- S3 버킷 또는 데이터가 상주하는 기타 데이터 소스에서 읽기
- PySpark 스크립트 또는 JAR 파일이 상주하는 S3 버킷 또는 접두사에서 읽기
- 최종 출력을 쓰려는 S3 버킷에 쓰기
- S3MonitoringConfiguration에서 지정하는 S3 버킷 또는 접두사로 로그 쓰기
- S3 버킷의 데이터를 암호화하기 위해 KMS 키를 사용하는 경우 KMS 키에 대한 액세스
- SparkSQL을 사용하는 경우 AWS Glue Data Catalog에 액세스

Spark 작업이 다른 데이터 소스에서 데이터를 읽거나 쓰는 경우 이 IAM 역할에서 적절한 권한을 지정합니다. IAM 역할에 이러한 권한을 제공하지 않으면 작업이 실패할 수 있습니다. 자세한 정보는 [Amazon EMR Serverless에 대한 작업 런타임 역할](#) 및 [로그 저장](#) 섹션을 참조하세요.

### Spark 작업 드라이버 파라미터

**jobDriver**를 사용하여 작업에 대한 입력을 제공합니다. 작업 드라이버 파라미터는 실행하려는 작업 유형에 대해 하나의 값만 허용합니다. Spark 작업의 경우 파라미터 값은 `sparkSubmit`입니다. 이 작업 유형을 사용하여 Spark 제출을 통해 Scala, Java, PySpark 및 기타 지원되는 작업을 실행할 수 있습니다. Spark 작업에는 다음 파라미터가 있습니다.

- **sparkSubmitParameters** - 작업에 전송하려는 추가 Spark 파라미터입니다. 이 파라미터를 사용하여 드라이버 메모리 또는 실행기 수와 같은 기본 Spark 속성(예: `--conf` 또는 `--class`에 정의된 속성)을 재정의합니다.

- **entryPointArguments** - 기본 JAR 또는 Python 파일에 전달하려는 인수의 배열입니다. `entrypoint` 코드를 사용하여 이러한 파라미터를 읽는 작업을 처리해야 합니다. 배열의 각 인수를 쉼표로 분리합니다.
- **entryPoint** - Amazon S3에서 실행하려는 기본 JAR 또는 Python 파일에 대한 참조입니다. Scala 또는 Java JAR을 실행하는 경우 `--class` 인수를 `SparkSubmitParameters`에서 기본 항목 클래스를 지정합니다.

자세한 내용은 [Launching Applications with spark-submit](#)을 참조하세요.

## Spark 구성 재정의의 파라미터

모니터링 수준 및 애플리케이션 수준 구성 속성을 재정의하려면 **configurationOverrides**를 사용합니다. 이 파라미터는 다음 두 필드를 포함하는 JSON 객체를 수락합니다.

- **monitoringConfiguration** - 이 필드를 사용하여 EMR Serverless 작업에서 Spark 작업의 로그를 저장할 Amazon S3 URL(`s3MonitoringConfiguration`)을 지정합니다. 애플리케이션을 호스팅하는 동일한 AWS 계정을 사용하고 작업을 실행하는 동일한 AWS 리전에서 이 버킷을 생성하는지 확인합니다.
- **applicationConfiguration** - 애플리케이션에 대한 기본 구성을 재정의하기 위해 이 필드에서 구성 객체를 제공할 수 있습니다. 간편 구문을 사용하여 구성을 제공하거나 JSON 파일의 구성 객체를 참조할 수 있습니다. 구성 객체는 분류, 속성 및 선택적 중첩 구성으로 이루어져 있습니다. 속성은 해당 파일에서 재정의하려는 설정으로 구성됩니다. 단일 JSON 객체에서 여러 애플리케이션에 대해 다양한 분류를 지정할 수 있습니다.

### Note

사용 가능한 구성 분류는 특정 EMR Serverless 릴리스에 따라 다릅니다. 예를 들어, 사용자 지정 Log4j `spark-driver-log4j2` 및 `spark-executor-log4j2`에 대한 분류는 릴리스 6.8.0 이상에서만 사용할 수 있습니다.

애플리케이션 재정의 및 Spark 제출 파라미터에서 동일한 구성을 사용하는 경우 Spark 제출 파라미터가 우선합니다. 구성의 우선순위는 다음과 같습니다(최상위부터 최하위의 순서).

- `SparkSession`을 생성하는 경우 EMR Serverless에서 제공하는 구성.
- `--conf` 인수와 함께 `sparkSubmitParameters`의 일부로 제공하는 구성.
- 작업을 시작하는 경우 애플리케이션 재정의의 일부로 제공하는 구성.

- 애플리케이션을 생성하는 경우 runtimeConfiguration의 일부로 제공하는 구성.
- 해당 릴리스에 대해 Amazon EMR에서 사용하는 최적화된 구성.
- 애플리케이션의 기본 오픈 소스 구성.

애플리케이션 수준에서 구성을 선언하고 작업 실행 중에 구성을 재정의하는 방법에 대한 자세한 내용은 [EMR Serverless에 대한 기본 애플리케이션 구성](#) 섹션을 참조하세요.

## Spark 동적 리소스 할당 최적화

dynamicAllocationOptimization을 사용하여 EMR Serverless에서 리소스 사용을 최적화합니다. Spark 구성 분류에서 이 속성을 true로 설정하면 EMR Serverless가 실행기 리소스 할당을 최적화하여 EMR Serverless에서 작업자를 생성하고 릴리스하는 속도에 맞게 Spark에서 실행기를 요청하고 취소하는 속도를 조정할 수 있습니다. 이렇게 하면 EMR Serverless는 여러 단계에서 작업자를 더 최적으로 재사용하므로 동일한 성능을 유지 관리하면서 여러 단계에서 작업을 실행할 때 비용이 절감됩니다.

이 속성은 모든 Amazon EMR 릴리스 버전에서 사용할 수 있습니다.

다음은 dynamicAllocationOptimization을 사용하는 샘플 구성 분류입니다.

```
[
  {
    "Classification": "spark",
    "Properties": {
      "dynamicAllocationOptimization": "true"
    }
  }
]
```

동적 할당 최적화를 사용하는 경우 다음을 고려합니다.

- 이 최적화는 동적 리소스 할당을 활성화한 Spark 작업에 대해 사용할 수 있습니다.
- 비용 효율성을 극대화하기 위해 워크로드를 기반으로 작업 수준 설정 spark.dynamicAllocation.maxExecutors 또는 [애플리케이션 수준 최대 용량](#) 설정을 사용해 워커의 조정 상한을 구성하는 것이 좋습니다.
- 단순한 작업에서는 비용 절감 효과를 느끼지 못할 수도 있습니다. 예를 들어, 작업이 작은 데이터세트에서 실행되거나 한 단계에서 실행을 완료하는 경우 Spark에는 더 많은 수의 실행기 또는 다중 조정 이벤트가 필요하지 않을 수 있습니다.

- 큰 단계, 더 작은 단계, 이후 다시 큰 단계로 구성된 시퀀스의 작업은 작업 런타임에서 회귀가 발생할 수 있습니다. EMR Serverless는 리소스를 더 효율적으로 사용하므로, 더 큰 단계에 대해 사용 가능한 작업자 수가 줄어들어 런타임이 길어질 수 있습니다.

## Spark 작업 속성

다음 표에는 선택적 Spark 속성과 Spark 작업을 제출할 때 재정의할 수 있는 기본값이 나와 있습니다.

### 선택적 Spark 속성 및 기본값

키	설명	기본값
<code>spark.archives</code>	Spark에서 각 실행기의 작업 디렉터리로 추출하는 쉼표로 구분된 아카이브 목록. 지원되는 파일 형식은 <code>.jar</code> , <code>.tar.gz</code> , <code>.tgz</code> 및 <code>.zip</code> 입니다. 추출할 디렉터리 이름을 지정하려면 추출하려는 파일 이름 뒤에 #을 추가합니다. 예를 들어, <code>file.zip#directory</code> 입니다.	NULL
<code>spark.authenticate</code>	Spark의 내부 연결 인증을 켜는 옵션.	TRUE
<code>spark.driver.cores</code>	드라이버가 사용하는 코어 수.	4
<code>spark.driver.extraJavaOptions</code>	Spark 드라이버에 대한 추가 Java 옵션.	NULL
<code>spark.driver.memory</code>	드라이버가 사용하는 메모리의 양.	14G
<code>spark.dynamicAllocation.enabled</code>	동적 리소스 할당을 켜는 옵션. 이 옵션은 워크로드에 따라 애플리케이션에 등록된 실행기 수를 스케일 업 또는 스케일 다운합니다.	TRUE

키	설명	기본값
<code>spark.dynamicAllocation.executorIdleTimeout</code>	Spark에서 제거하기 전에 실행기가 유휴 상태를 유지할 수 있는 시간. 이는 동적 할당을 켜는 경우에만 적용됩니다.	60초
<code>spark.dynamicAllocation.initialExecutors</code>	동적 할당을 켜는 경우 실행할 초기 실행기 수.	3
<code>spark.dynamicAllocation.maxExecutors</code>	동적 할당을 켜는 경우 실행기 수의 상한.	6.10.0 이상의 경우 infinity 6.9.0 이하의 경우 100
<code>spark.dynamicAllocation.minExecutors</code>	동적 할당을 켜는 경우 실행기 수의 하한.	0
<code>spark.emr-serverless.allocation.batch.size</code>	실행기 할당의 각 주기에서 요청할 컨테이너 수. 각 할당 주기 사이 간격은 1초입니다.	20
<code>spark.emr-serverless.driver.disk</code>	Spark 드라이버 디스크.	20G
<code>spark.emr-serverless.driverEnv.</code> <b>[KEY]</b>	환경 변수를 Spark 드라이버에 추가하는 옵션.	NULL
<code>spark.emr-serverless.executor.disk</code>	Spark 실행기 디스크.	20G
<code>spark.emr-serverless.memoryOverheadFactor</code>	드라이버 및 실행기 컨테이너 메모리에 추가할 메모리 오버헤드를 설정합니다.	0.1
<code>spark.emr-serverless.driver.disk.type</code>	Spark 드라이버에 연결된 디스크 유형.	표준

키	설명	기본값
<code>spark.emr-serverless.executor.disk.type</code>	Spark 실행기에 연결된 디스크 유형.	표준
<code>spark.executor.cores</code>	각 실행기에서 사용할 코어 수.	4
<code>spark.executor.extraJavaOptions</code>	Spark 실행기에 대한 추가 Java 옵션.	NULL
<code>spark.executor.instances</code>	할당할 Spark 실행기 컨테이너 수.	3
<code>spark.executor.memory</code>	각 실행기가 사용하는 메모리 양.	14G
<code>spark.executorEnv. [KEY]</code>	환경 변수를 Spark 실행기에 추가하는 옵션.	NULL
<code>spark.files</code>	각 실행기의 작업 디렉터리에 배치할 쉼표로 구분된 파일 목록. <code>SparkFiles.get( <i>fileName</i> )</code> 을 사용하여 실행기에서 이러한 파일의 파일 경로에 액세스할 수 있습니다.	NULL
<code>spark.hadoop.hive.metastore.client.factory.class</code>	Hive 메타스토어 구현 클래스.	NULL
<code>spark.jars</code>	드라이버 및 실행기의 런타임 클래스 경로에 추가할 추가 jar.	NULL
<code>spark.network.crypto.enabled</code>	AES 기반 RPC 암호화를 켜는 옵션. 여기에는 Spark 2.2.0에 추가된 인증 프로토콜이 포함됩니다.	FALSE

키	설명	기본값
<code>spark.sql.warehouse.dir</code>	관리형 데이터베이스 및 테이블의 기본 위치.	<code>\$PWD/spark-warehouse</code> 의 값
<code>spark.submit.pyFiles</code>	Python 앱에 대한 PYTHONPATH 에 배치할 .zip, .egg 또는 .py 파일의 쉼표로 구분된 목록.	NULL

다음 표에는 기본 Spark 제출 파라미터가 나와 있습니다.

#### 기본 Spark 제출 파라미터

키	설명	기본값
<code>archives</code>	Spark에서 각 실행기의 작업 디렉터리로 추출하는 쉼표로 구분된 아카이브 목록.	NULL
<code>class</code>	애플리케이션의 기본 클래스 (Java 및 Scala 응용).	NULL
<code>conf</code>	임의의 Spark 구성 속성.	NULL
<code>driver-cores</code>	드라이버가 사용하는 코어 수.	4
<code>driver-memory</code>	드라이버가 사용하는 메모리의 양.	14G
<code>executor-cores</code>	각 실행기에서 사용할 코어 수.	4
<code>executor-memory</code>	실행기가 사용하는 메모리의 양.	14G
<code>files</code>	각 실행기의 작업 디렉터리에 배치할 쉼표로 구분된 파일 목록. <code>SparkFiles.get( <i>fileName</i> )</code> 을 사용	NULL

키	설명	기본값
	하여 실행기에서 이러한 파일의 파일 경로에 액세스할 수 있습니다.	
jars	드라이버 및 실행기 클래스 경로에 포함할 jar의 쉼표로 구분된 목록.	NULL
num-executors	실행할 실행기 수.	3
py-files	Python 앱에 대한 PYTHONPATH 에 배치할 .zip, .egg 또는 .py 파일의 쉼표로 구분된 목록.	NULL
verbose	추가 디버그 출력을 켜는 옵션.	NULL

## 리소스 구성 모범 사례

### StartJobRun API를 통해 드라이버 및 실행기 리소스 구성

#### Note

Spark 드라이버와 실행기 코어 및 메모리 속성은 지정된 경우 StartJobRun API 요청에 직접 지정합니다.

이러한 방식으로 리소스를 구성하면 EMR Serverless가 작업을 실행하기 전에 올바른 리소스를 할당할 수 있습니다. 이는 스크립트 실행이 시작되기 전에 드라이버 및 실행기 워커가 사전 프로비저닝되는 경우가 있기 때문에 너무 늦게 평가되는 .py 또는 .jar 파일과 같은 사용자 스크립트에 제공되는 설정과는 대조적입니다. 작업 제출 중에 이러한 리소스를 구성하는 방법으로는 다음의 두 가지 방법이 지원됩니다.

#### 옵션 1: sparkSubmitParameters 사용

```
"jobDriver": {
```

```
"sparkSubmit": {
  "entryPoint": "s3://your-script-path.py",
  "sparkSubmitParameters": "-conf spark.driver.memory=4g \
-conf spark.driver.cores=2 \
-conf spark.executor.memory=8g \
-conf spark.executor.cores=4"
}
```

## 옵션 2: spark-defaults 분류에 configurationOverrides 사용

```
"configurationOverrides": {
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.driver.memory": "4g",
        "spark.driver.cores": "2",
        "spark.executor.memory": "8g",
        "spark.executor.cores": "4"
      }
    }
  ]
}
```

## Spark 예제

다음 예제에서는 StartJobRun API를 사용하여 Python 스크립트를 실행하는 방법을 보여줍니다. 이 예제를 사용하는 엔드투엔드 자습서는 [Amazon EMR Serverless 시작하기](#) 섹션을 참조하세요. [EMR Serverless Samples](#) GitHub 리포지토리에서 PySpark 작업을 실행하고 Python 종속 항목을 추가하는 방법에 대한 추가 예제를 찾을 수 있습니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/
wordcount/scripts/wordcount.py",
      "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket/
wordcount_output"],
```

```
"sparkSubmitParameters": "--conf spark.executor.cores=1 --conf
spark.executor.memory=4g --conf spark.driver.cores=1 --conf spark.driver.memory=4g --
conf spark.executor.instances=1"
    }
  }'
```

다음 예제에서는 StartJobRun API를 사용하여 Spark JAR을 실행하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
      "entryPointArguments": ["1"],
      "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
spark.executor.cores=4 --conf spark.executor.memory=20g --conf spark.driver.cores=4 --
conf spark.driver.memory=8g --conf spark.executor.instances=1"
    }
  }'
```

## EMR Serverless 작업을 실행하는 경우 Hive 구성 사용

type 파라미터가 HIVE로 설정된 애플리케이션에서 Hive 작업을 실행할 수 있습니다. 작업은 Amazon EMR 릴리스 버전과 호환되는 Hive 버전과 호환되어야 합니다. 예를 들어, Amazon EMR 릴리스 6.6.0을 사용하여 애플리케이션에서 작업을 실행할 때 작업은 Apache Hive 3.1.2와 호환되어야 합니다. 각 릴리스의 애플리케이션 버전에 대한 자세한 내용은 [Amazon EMR Serverless 릴리스 버전](#) 섹션을 참조하세요.

### Hive 작업 파라미터

[StartJobRun API](#)를 사용하여 Hive 작업을 실행하는 경우 다음 파라미터를 지정합니다.

필수 파라미터

- [Hive 작업 런타임 역할](#)
- [Hive 작업 드라이버 파라미터](#)
- [Hive 구성 재정의의 파라미터](#)

## Hive 작업 런타임 역할

**executionRoleArn**을 사용하여 애플리케이션이 Hive 작업을 실행하는 데 사용하는 IAM 역할에 대한 ARN을 지정합니다. 이 역할에는 다음 권한이 있어야 합니다.

- S3 버킷 또는 데이터가 상주하는 기타 데이터 소스에서 읽기
- Hive 쿼리 파일 및 init 쿼리 파일이 상주하는 S3 버킷 또는 접두사에서 읽기
- Hive Scratch 디렉터리와 Hive Metastore 웨어하우스 디렉터리가 상주하는 S3 버킷에 읽기 및 쓰기
- 최종 출력을 쓰려는 S3 버킷에 쓰기
- S3MonitoringConfiguration에서 지정하는 S3 버킷 또는 접두사로 로그 쓰기
- S3 버킷의 데이터를 암호화하기 위해 KMS 키를 사용하는 경우 KMS 키에 대한 액세스
- AWS Glue Data Catalog에 대한 액세스

Hive 작업이 다른 데이터 소스에서 데이터를 읽거나 쓰는 경우 이 IAM 역할에서 적절한 권한을 지정합니다. IAM 역할에 이러한 권한을 제공하지 않으면 작업이 실패할 수 있습니다. 자세한 정보는 [Amazon EMR Serverless에 대한 작업 런타임 역할](#) 섹션을 참조하세요.

## Hive 작업 드라이버 파라미터

**jobDriver**를 사용하여 작업에 대한 입력을 제공합니다. 작업 드라이버 파라미터는 실행하려는 작업 유형에 대해 하나의 값만 허용합니다. 작업 유형으로 hive를 지정하면 EMR Serverless는 Hive 쿼리를 jobDriver 파라미터에 전달합니다. Hive 자산에는 다음 파라미터가 있습니다.

- **query** - Amazon S3에서 실행하려는 Hive 쿼리 파일에 대한 참조입니다.
- **parameters** - 재정의하려는 추가 Hive 구성 속성입니다. 속성을 재정의하려면 이 파라미터에 --hiveconf *property=value*로 전달합니다. 변수를 재정의하려면 이 파라미터에 --hivevar *key=value*로 전달합니다.
- **initQueryFile** - init Hive 쿼리 파일입니다. Hive는 쿼리 전에 이 파일을 실행하고 이를 사용하여 테이블을 초기화할 수 있습니다.

## Hive 구성 재정의의 파라미터

모니터링 수준 및 애플리케이션 수준 구성 속성을 재정의하려면 **configurationOverrides**를 사용합니다. 이 파라미터는 다음 두 필드를 포함하는 JSON 객체를 수락합니다.

- **monitoringConfiguration** - 이 필드를 사용하여 EMR Serverless 작업에서 Hive 작업의 로그를 저장할 Amazon S3 URL(s3MonitoringConfiguration)을 지정합니다. 애플리케이션을 호스팅하는 동일한 AWS 계정을 사용하고 작업을 실행하는 동일한 AWS 리전에서 이 버킷을 생성하는지 확인합니다.
- **applicationConfiguration** - 애플리케이션에 대한 기본 구성을 재정의하도록 이 필드에 구성 객체를 제공할 수 있습니다. 간편 구문을 사용하여 구성을 제공하거나 JSON 파일의 구성 객체를 참조할 수 있습니다. 구성 객체는 분류, 속성 및 선택적 중첩 구성으로 이루어져 있습니다. 속성은 해당 파일에서 재정의하려는 설정으로 구성됩니다. 단일 JSON 객체에서 여러 애플리케이션에 대해 다양한 분류를 지정할 수 있습니다.

#### Note

사용 가능한 구성 분류는 특정 EMR Serverless 릴리스에 따라 다릅니다. 예를 들어, 사용자 지정 Log4j spark-driver-log4j2 및 spark-executor-log4j2에 대한 분류는 릴리스 6.8.0 이상에서만 사용할 수 있습니다.

애플리케이션 재정의 및 Hive 파라미터에서 동일한 구성을 전달하는 경우 Hive 파라미터가 우선합니다. 다음 목록은 가장 높은 우선순위에서 가장 낮은 우선순위로 구성의 우선순위를 나열합니다.

- `--hiveconf property=value`를 사용하여 Hive 파라미터의 일부로 제공하는 구성.
- 작업을 시작하는 경우 애플리케이션 재정의의 일부로 제공하는 구성.
- 애플리케이션을 생성하는 경우 runtimeConfiguration의 일부로 제공하는 구성.
- 해당 릴리스에 대해 Amazon EMR에서 할당하는 최적화된 구성.
- 애플리케이션의 기본 오픈 소스 구성.

애플리케이션 수준에서 구성을 선언하고 작업 실행 중에 구성을 재정의하는 방법에 대한 자세한 내용은 [EMR Serverless에 대한 기본 애플리케이션 구성](#) 섹션을 참조하세요.

## Hive 작업 속성

다음 표에는 Hive 작업 제출 시 구성하는 필수 속성이 나와 있습니다.

## 필수 Hive 작업 속성

설정	설명
<code>hive.exec.scratchdir</code>	EMR Serverless가 Hive 작업 실행 중에 임시 파일을 생성하는 Amazon S3 위치.
<code>hive.metastore.warehouse.dir</code>	Hive의 관리형 테이블에 대한 데이터베이스의 Amazon S3 위치.

다음 표에는 선택적 Hive 속성과 Hive 작업을 제출하는 경우 재정의할 수 있는 기본값이 나와 있습니다.

## 선택적 Hive 속성 및 기본값

설정	설명	기본값
<code>fs.s3.customAWSCredentialsProvider</code>	사용하려는 AWS 자격 증명 제공업체.	<code>com.amazonaws.auth.DefaultAWSCredentialsProviderChain</code>
<code>fs.s3a.aws.credentials.provider</code>	S3A 파일 시스템에서 사용할 AWS 자격 증명 제공업체입니다.	<code>com.amazonaws.auth.DefaultAWSCredentialsProviderChain</code>
<code>hive.auto.convert.join</code>	입력 파일 크기에 따라 공통 조인을 맵 조인으로 자동 변환하는 옵션.	TRUE
<code>hive.auto.convert.join.noconditionaltask</code>	Hive가 입력 파일 크기에 따라 공통 조인을 맵 조인으로 변환하는 경우 최적화를 켜는 옵션.	TRUE
<code>hive.auto.convert.join.noconditionaltask.size</code>	조인은 이 크기 미만의 맵 조인으로 직접 변환됩니다.	최적의 값은 Tez 태스크 메모리를 기반으로 계산됩니다.
<code>hive.cbo.enable</code>	Calcite 프레임워크를 사용하여 비용 기반 최적화를 켜는 옵션.	TRUE

설정	설명	기본값
<code>hive.cli.tez.session.async</code>	Hive 쿼리가 컴파일되는 동안 백그라운드 Tez 세션을 시작하는 옵션. <code>false</code> 로 설정하면 Hive 쿼리가 컴파일된 후 Tez AM이 시작됩니다.	TRUE
<code>hive.compute.query.using.stats</code>	Hive를 활성화하여 메타스토어에 저장된 통계를 사용해 특정 쿼리에 응답하는 옵션. 기본 통계의 경우 <code>hive.stats.autogather</code> 를 TRUE로 설정합니다. 보다 고급 쿼리 컬렉션의 경우 <code>analyze table queries</code> 를 실행합니다.	TRUE
<code>hive.default.fileformat</code>	CREATE TABLE 문의 기본 파일 형식. CREATE TABLE 명령에서 STORED AS [FORMAT]을 지정하는 경우 이 값을 명시적으로 재정의할 수 있습니다.	TEXTFILE
<code>hive.driver.cores</code>	Hive 드라이버 프로세스에 사용할 코어 수.	2
<code>hive.driver.disk</code>	Hive 드라이버의 디스크 크기.	20G
<code>hive.driver.disk.type</code>	Hive 드라이버의 디스크 유형.	표준
<code>hive.tez.disk.type</code>	Tez 작업자의 디스크 크기.	표준

설정	설명	기본값
hive.driver.memory	Hive 드라이버 프로세스당 사용할 메모리 양. Hive CLI 및 Tez Application Master는 이 메모리를 헤드룸의 20%로 균등하게 공유합니다.	6G
hive.emr-serverless.launch.env.[ <b>KEY</b> ]	Hive 드라이버, Tez AM 및 Tez 태스크와 같은 모든 Hive별 프로세스에서 <b>KEY</b> 환경 변수를 설정하는 옵션.	
hive.exec.dynamic.partition	DML/DDI에서 동적 파티션을 켜는 옵션.	TRUE
hive.exec.dynamic.partition.mode	엄격한 모드 또는 비엄격 모드를 사용할지 여부를 지정하는 옵션. 엄격한 모드에서는 실수로 모든 파티션을 덮어쓰는 경우를 대비하여 하나 이상의 정적 파티션을 지정합니다. 비엄격 모드에서는 모든 파티션이 동적으로 허용됩니다.	strict
hive.exec.max.dynamic.partitions	Hive가 총 생성하는 최대 동적 파티션 수.	1000
hive.exec.max.dynamic.partitions.per.node	Hive가 각 매퍼 및 감소기 노드에서 생성하는 최대 동적 파티션 수.	100

설정	설명	기본값
<code>hive.exec.orc.split.strategy</code>	BI, ETL 또는 HYBRID 값 중 하나가 예상됩니다. 사용자 수준 구성이 아닙니다. BI에서는 쿼리 실행과는 반대로, 분할 생성에 더 적은 시간을 소비하도록 지정합니다. ETL에서는 분할 생성에 더 많은 시간을 소비하도록 지정합니다. HYBRID에서는 휴리스틱을 기반으로 앞서 언급된 전략 중 하나를 지정합니다.	HYBRID
<code>hive.exec.reducers.bytes.per.reducer</code>	감소기당 크기. 기본값은 256MB입니다. 입력 크기가 1G인 경우 작업은 4개의 감소기를 사용합니다.	256000000
<code>hive.exec.reducers.max</code>	최대 감소기 수.	256
<code>hive.exec.stagingdir</code>	Hive가 테이블 위치 내부 및 <code>hive.exec.scratchdir</code> 속성에 지정된 스크래치 디렉터리 위치에서 생성하는 임시 파일을 저장하는 디렉터리의 이름.	<code>.hive-staging</code>
<code>hive.fetch.task.conversion</code>	NONE, MINIMAL 또는 MORE 값 중 하나가 예상됩니다. Hive는 선택한 쿼리를 단일 FETCH 태스크로 변환할 수 있습니다. 이 경우 지연 시간이 최소화됩니다.	MORE

설정	설명	기본값
hive.groupby.position.alias	Hive가 GROUP BY 문에서 열 위치 별칭을 사용하도록 하는 옵션.	FALSE
hive.input.format	기본 입력 형식. CombineHiveInputFormat 에 문제가 발생하면 HiveInputFormat 으로 설정합니다.	org.apache.hadoop.hive ql.io.CombineHiveInputFormat
hive.log.explain.output	Hive 로그의 쿼리에 대한 확장 출력에 대한 설명을 켜는 옵션.	FALSE
hive.log.level	Hive 로깅 수준.	INFO
hive.mapred.reduce.tasks.speculative.execution	감소기에 대한 투기적 시작을 켜는 옵션. Amazon EMR 6.10.x 이하에서만 지원됩니다.	TRUE
hive.max-task-containers	최대 동시 컨테이너 수. 구성된 매퍼 메모리에 이 값을 곱하여 계산과 태스크 선점에서 사용하는 사용 가능한 메모리를 결정합니다.	1000
hive.merge.mapfiles	맵 전용 작업이 끝날 때 작은 파일이 병합되는 옵션.	TRUE
hive.merge.size.per.task	작업 종료 시 병합된 파일의 크기.	256000000
hive.merge.tezfiles	Tez DAG 끝에서 작은 파일 병합을 켜는 옵션.	FALSE
hive.metastore.client.factory.class	IMetaStoreClient 인터페이스를 구현하는 객체를 생성하는 팩토리 클래스의 이름.	com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory

설정	설명	기본값
hive.metastore.glue.catalogid	AWS Glue Data Catalog가 메타스토어 역할을 하지만 작업과 다른 AWS 계정에서 실행되는 경우 작업이 실행 중인 AWS 계정의 ID입니다.	NULL
hive.metastore.uris	메타스토어 클라이언트가 원격 메타스토어에 연결하는 데 사용하는 Thrift URI.	NULL
hive.optimize.ppd	조건자 푸시다운을 켜는 옵션.	TRUE
hive.optimize.ppd.storage	스토리지 핸들러에 대한 조건자 푸시다운을 켜는 옵션.	TRUE
hive.orderby.position.alias	Hive가 ORDER BY 문에서 열 위치 별칭을 사용하도록 하는 옵션.	TRUE
hive.prewarm.enabled	Tez에 대한 컨테이너 예열을 켜는 옵션.	FALSE
hive.prewarm.numcontainers	Tez에 대해 예열할 컨테이너 수.	10
hive.stats.autogather	Hive가 INSERT OVERWRITE 명령 중에 기본 통계를 자동으로 수집하도록 하는 옵션.	TRUE
hive.stats.fetch.column.stats	메타스토어에서 열 통계 가져오기를 끄는 옵션. 열 수가 많으면 열 통계 가져오기 비용이 많이 들 수 있습니다.	FALSE

설정	설명	기본값
hive.stats.gather.num.threads	partialscan 및 noscan 분석 명령이 파티션된 테이블에 대해 사용하는 스레드 수. 이는 StatsProvidingRecordReader를 구현하는 파일 형식(예: ORC)에만 적용됩니다.	10
hive.strict.checks.cartesian.product	엄격한 Cartesian 조인 검사를 켜는 옵션. 이러한 검사에서는 카테시안 곱(교차 조인)을 허용하지 않습니다.	FALSE
hive.strict.checks.type.safety	엄격한 유형의 안전 검사를 켜고 string 및 double과 bigint의 비교를 끄는 옵션.	TRUE
hive.support.quoted.identifiers	NONE 또는 COLUMN의 값이 예상됩니다. NONE은 식별자에 영숫자와 밑줄 문자만 유효함을 의미합니다. COLUMN은 열 이름에 모든 문자가 포함될 수 있음을 의미합니다.	COLUMN
hive.tez.auto.reducer.parallelism	Tez 자동 감소기 병렬 처리 기능을 켜는 옵션. Hive는 여전히 데이터 크기를 예측하고 병렬 처리 예측을 설정합니다. Tez는 소스 버텍스의 출력 크기를 샘플링하고 필요에 따라 런타임 시 예측을 조정합니다.	TRUE
hive.tez.container.size	Tez 태스크 프로세스당 사용할 메모리 양.	6144

설정	설명	기본값
<code>hive.tez.cpu.vcores</code>	각 Tez 태스크에 사용할 코어 수.	2
<code>hive.tez.disk.size</code>	각 태스크 컨테이너의 디스크 크기.	20G
<code>hive.tez.input.format</code>	Tez AM에서 분할 생성을 위한 입력 형식.	<code>org.apache.hadoop.hive ql.io.HiveInputFormat</code>
<code>hive.tez.min.partition.factor</code>	자동 감소기 병렬 처리를 켤 때 Tez에서 지정하는 감소기의 하한.	0.25
<code>hive.vectorized.execution.enabled</code>	벡터화된 쿼리 실행 모드를 켜는 옵션.	TRUE
<code>hive.vectorized.execution.reduce.enabled</code>	쿼리 실행의 감소 측 벡터화 모드를 켜는 옵션.	TRUE
<code>javax.jdo.option.ConnectionDriverName</code>	JDBC 메타스토어의 드라이버 클래스 이름.	<code>org.apache.derby.jdbc.EmbeddedDriver</code>
<code>javax.jdo.option.ConnectionPassword</code>	메타스토어 데이터베이스와 연결된 암호.	NULL
<code>javax.jdo.option.ConnectionURL</code>	JDBC 메타스토어의 JDBC 연결 문자열.	<code>jdbc:derby;;databaseName=metastore_db;create=true</code>
<code>javax.jdo.option.ConnectionUserName</code>	메타스토어 데이터베이스와 연결된 사용자 이름.	NULL

설정	설명	기본값
<code>mapreduce.input.fileinputformat.split.maxsize</code>	입력 형식이 <code>org.apache.hadoop.hive ql.io.CombineHiveInputFormat</code> 인 경우 분할 계산 중 분할의 최대 크기. 값이 0이면 제한이 없음을 나타냅니다.	0
<code>tez.am.dag.cleanup.on.completion</code>	DAG가 완료될 때 셔플 데이터 정리를 켜는 옵션.	TRUE
<code>tez.am.emr-serverless.launch.env.[ KEY ]</code>	Tez AM 프로세스에서 <b>KEY</b> 환경 변수를 설정하는 옵션. Tez AM의 경우 이 값은 <code>hive.emr-serverless.launch.env.[ KEY ]</code> 값을 재정의합니다.	
<code>tez.am.log.level</code>	EMR Serverless가 Tez App Primary에 전달하는 루트 로깅 수준.	INFO
<code>tez.am.sleep.time.before.exit.millis</code>	EMR Serverless는 AM 종료 요청 후 이 기간 이후에 ATS 이벤트를 푸시해야 합니다.	0
<code>tez.am.speculation.enabled</code>	느린 태스크의 투기적 시작이 수행되는 옵션. 이 기능은 일부 태스크가 불량 또는 느린 머신으로 인해 느리게 실행될 때 작업 지연 시간을 줄이는 데 도움이 될 수 있습니다. Amazon EMR 6.10.x 이하에서만 지원됩니다.	FALSE

설정	설명	기본값
<code>tez.am.task.max.failed.attempts</code>	태스크에 실패하기 전에 특정 태스크에 대해 실패할 수 있는 최대 시도 횟수. 이 수치는 수동으로 종료된 시도를 계산에 포함하지 않습니다.	3
<code>tez.am.vertex.cleanup.height</code>	모든 종속 버텍스가 완료된 경우 Tez AM에서 버텍스 셔플 데이터를 삭제하는 거리. 값이 0이면 이 기능이 꺼집니다. Amazon EMR 버전 6.8.0 이상에서 이 기능을 지원합니다.	0
<code>tez.client.asynchronous-stop</code>	EMR Serverless가 Hive 드라이버를 종료하기 전에 ATS 이벤트를 푸시하도록 하는 옵션.	FALSE
<code>tez.grouping.max-size</code>	그룹화된 분할의 크기 상한(바이트). 이 제한은 지나치게 큰 분할을 방지합니다.	1073741824
<code>tez.grouping.min-size</code>	그룹화된 분할의 크기 하한(바이트). 이 제한은 너무 많은 작은 분할을 방지합니다.	16777216
<code>tez.runtime.io.sort.mb</code>	Tez가 출력을 정렬하는 경우 소프트웨어 버퍼의 크기가 정렬됩니다.	최적의 값은 Tez 태스크 메모리를 기반으로 계산됩니다.
<code>tez.runtime.unordered.output.buffer.size-mb</code>	Tez에서 디스크에 직접 쓰지 않는 경우 사용할 버퍼의 크기.	최적의 값은 Tez 태스크 메모리를 기반으로 계산됩니다.

설정	설명	기본값
<code>tez.shuffle-vertex-manager.max-src-fraction</code>	EMR Serverless에서 현재 버텍스에 대한 모든 태스크를 예약하기 전에 완료해야 하는 소스 태스크의 비율(ScatterGather 연결의 경우). 현재 버텍스에서 예약할 준비가 된 작업 수는 <code>min-fraction ~ max-fraction</code> 사이에서 선형으로 조정됩니다. 이 값은 기본값 또는 <code>tez.shuffle-vertex-manager.min-src-fraction</code> 중 큰 값으로 설정됩니다.	0.75
<code>tez.shuffle-vertex-manager.min-src-fraction</code>	EMR Serverless에서 현재 버텍스에 대한 태스크를 예약하기 전에 완료해야 하는 소스 태스크의 비율(ScatterGather 연결의 경우).	0.25
<code>tez.task.emr-serverless.launch.env.[KEY]</code>	Tez 태스크 프로세스에서 <b>KEY</b> 환경 변수를 설정하는 옵션. Tez 태스크의 경우 이 값은 <code>hive.emr-serverless.launch.env.[KEY]</code> 값을 재정의합니다.	
<code>tez.task.log.level</code>	EMR Serverless가 Tez 태스크에 전달하는 루트 로깅 수준.	INFO
<code>tez.yarn.ats.event.flush.timeout.millis</code>	종료하기 전에 이벤트가 플러시될 때까지 AM에서 기다려야 하는 최대 시간.	300000

## Hive 작업 예제

다음 코드 예제에서는 StartJobRun API를 사용하여 Hive 쿼리를 실행하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-
query.q1",
      "parameters": "--hiveconf hive.log.explain.output=false"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "hive.exec.scratchdir": "s3://amzn-s3-demo-bucket/emr-serverless-hive/
hive/scratch",
        "hive.metastore.warehouse.dir": "s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/warehouse",
        "hive.driver.cores": "2",
        "hive.driver.memory": "4g",
        "hive.tez.container.size": "4096",
        "hive.tez.cpu.vcores": "1"
      }
    }
  ]
}'
```

[EMR Serverless Samples](#) GitHub 리포지토리에서 Hive 작업을 실행하는 방법에 대한 추가 예제를 찾을 수 있습니다.

## EMR Serverless 작업 복원력

EMR Serverless 릴리스 7.1.0 이상에는 작업 복원력에 대한 지원이 포함되어 있으므로 수동으로 입력하지 않고도 실패한 작업을 자동으로 재시도합니다. 작업 복원력의 또 다른 이점은 AZ에 문제가 발생할 경우 EMR Serverless가 작업 실행을 다른 가용 영역(AZ)으로 이동한다는 점입니다.

작업에 대한 작업 복원력을 활성화하려면 작업에 대한 재시도 정책을 설정합니다. 재시도 정책은 언제라도 실패하면 EMR Serverless에서 작업을 자동으로 재시작하도록 보장합니다. 배치 작업 및 스트리

밍 작업 모두에 대해 재시도 정책이 지원되므로 사용 사례에 따라 작업 복원력을 사용자 지정합니다. 다음 표에서는 배치 및 스트리밍 작업 간 작업 복원력의 동작과 차이를 비교합니다.

	일괄 작업	스트리밍 작업
기본 동작	작업을 다시 실행하지 않습니다.	애플리케이션이 작업을 실행하는 동안 체크포인트를 생성하므로 항상 작업 실행을 재시도합니다.
재시도 지점	배치 작업에는 체크포인트가 없으므로 EMR Serverless는 항상 처음부터 다시 작업을 실행합니다.	스트리밍 작업은 체크포인트를 지원하므로, Amazon S3의 체크포인트 위치에 런타임 상태 및 진행 상황을 저장하도록 스트리밍 쿼리를 구성합니다. EMR Serverless는 체크포인트에서 작업 실행을 재개합니다. 자세한 내용은 <a href="#">Apache Spark 설명서의 Recovering from failures with Checkpointing</a> 을 참조하세요.
최대 재시도 횟수	최대 10회의 재시도를 허용합니다.	스트리밍 작업에는 기본 제공 스래시 방지 제어 기능이 있으므로 1시간 후에도 계속 실패하면 애플리케이션은 작업 재시도를 중지합니다. 1시간 내 기본 재시도 횟수는 5회입니다. 이 재시도 횟수를 1~10회로 구성할 수 있습니다. 최대 시도 횟수는 사용자 지정할 수 없습니다. 값이 1이면 재시도를 수행하지 않음을 나타냅니다.

EMR Serverless에서 작업을 재실행하려고 하면 시도 번호로 작업을 인덱싱하므로 시도 전반에 걸쳐 작업의 수명 주기를 추적합니다.

EMR Serverless API 작업 또는 AWS CLI를 사용하여 작업 복원력을 변경하거나 작업 복원력과 관련된 정보에 액세스합니다. 자세한 내용은 [EMR Serverless API 안내서](#)를 참조하세요.

기본적으로 EMR Serverless는 배치 작업을 재실행하지 않습니다. 배치 작업에 대한 재시도를 활성화하려면 배치 작업 실행을 시작할 때 `maxAttempts` 파라미터를 구성합니다. `maxAttempts` 파라미터는 배치 작업에만 적용됩니다. 기본값은 1입니다. 작업을 다시 실행하지 않음을 의미합니다. 허용되는 값은 1~10(경계 포함)입니다.

다음 예제에서는 작업 실행을 시작할 때 최대 10회의 시도 횟수를 지정하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'BATCH' \
--retry-policy '{
    "maxAttempts": 10
}' \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "/usr/lib/spark/examples/jars/spark-examples-does-not-exist.jar",
        "entryPointArguments": ["1"],
        "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
    }
}'
```

EMR Serverless는 스트리밍 작업이 실패할 경우 스트리밍 작업을 제한 없이 재시도합니다. 복구할 수 없는 반복 장애로 인한 스래싱을 방지하려면 `maxFailedAttemptsPerHour`를 사용하여 스트리밍 작업 재시도에 대한 스래시 방지 제어를 구성합니다. 이 파라미터를 사용하면 EMR Serverless에서 재시도를 중지하기 1시간 전에 허용되는 최대 실패 시도 횟수를 지정할 수 있습니다. 기본값은 5입니다. 허용되는 값은 1~10(경계 포함)입니다.

```
aws emr-serverless start-job-run
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--mode 'STREAMING' \
--retry-policy '{
    "maxFailedAttemptsPerHour": 7
}' \
--job-driver '{
    "sparkSubmit": {
```

```

    "entryPoint": "/usr/lib/spark/examples/jars/spark-examples-does-not-exist.jar",
    "entryPointArguments": ["1"],
    "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi"
  }
}'

```

또한 다른 작업 실행 API 작업을 사용하여 작업에 대한 정보를 얻을 수도 있습니다. 예를 들어, GetJobRun 작업과 함께 attempt 파라미터를 사용하여 특정 작업 시도에 대한 세부 정보를 가져올 수 있습니다. attempt 파라미터를 포함하지 않는 경우 작업은 최신 시도에 대한 정보를 반환합니다.

```

aws emr-serverless get-job-run \
  --job-run-id job-run-id \
  --application-id application-id \
  --attempt 1

```

ListJobRunAttempts 작업은 작업 실행과 관련된 모든 시도에 대한 정보를 반환합니다.

```

aws emr-serverless list-job-run-attempts \
  --application-id application-id \
  --job-run-id job-run-id

```

GetDashboardForJobRun 작업은 작업 실행을 위해 애플리케이션 UI에 액세스하는 데 사용하는 URL을 생성하고 반환합니다. attempt 파라미터를 사용하면 특정 시도에 대한 URL을 가져올 수 있습니다. attempt 파라미터를 포함하지 않는 경우 작업은 최신 시도에 대한 정보를 반환합니다.

```

aws emr-serverless get-dashboard-for-job-run \
  --application-id application-id \
  --job-run-id job-run-id \
  --attempt 1

```

## 재시도 정책으로 작업 모니터링

또한 작업 복원력 지원에는 새 이벤트 EMR Serverless 작업 실행 재시도가 추가되었습니다. EMR Serverless는 작업을 재시도할 때마다 이 이벤트를 게시합니다. 이 알림을 사용하여 작업 재시도를 추적할 수 있습니다. 이벤트에 자세한 내용은 [Amazon EventBridge 이벤트](#)를 참조하세요.

## 재시도 정책을 사용한 로깅

EMR Serverless가 작업을 재시도할 때마다 자체 로그 세트가 생성됩니다. EMR Serverless가 덮어쓰지 않고 이러한 로그를 Amazon S3 및 Amazon CloudWatch에 전달할 수 있도록 하기 위해 EMR Serverless는 S3 로그 경로 및 CloudWatch 로그 스트림 이름의 형식에 작업의 시도 번호를 포함하도록 접두사를 추가합니다.

다음은 해당 형식에 대한 예제입니다.

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'.
```

이 형식을 사용하면 EMR Serverless가 각 작업 시도에 대한 모든 로그를 Amazon S3 및 CloudWatch의 지정된 위치에 게시할 수 있습니다. 자세한 내용을 알아보려면 [로그 저장](#)을 참조하세요.

### Note

EMR Serverless는 재시도가 활성화된 모든 배치 작업 및 모든 스트리밍 작업에서만 이 접두사 형식을 사용합니다.

## EMR Serverless에 대한 메타스토어 구성

Hive 메타스토어는 스키마, 파티션 이름 및 데이터 유형을 포함하여 테이블에 대한 구조 정보를 저장하는 중앙 집중식 위치입니다. EMR Serverless를 사용하면 작업에 액세스할 수 있는 메타스토어에서 이 테이블 메타데이터를 유지할 수 있습니다.

Hive 메타스토어에 대한 두 가지 옵션이 있습니다.

- AWS Glue Data Catalog
- 외부 Apache Hive 메타스토어

## AWS Glue Data Catalog를 메타스토어로 사용

Glue Data Catalog를 AWS 메타스토어로 사용하도록 Spark 및 Hive 작업을 구성할 수 있습니다. 영구 메타스토어가 필요하거나 여러 애플리케이션, 서비스 또는 AWS 계정에서 메타스토어를 공유해야 하는 경우에 이 구성을 사용하는 것이 좋습니다. Data Catalog에 대한 자세한 내용은 [AWS Glue Data Catalog 채우기](#)를 참조하세요. AWS Glue 요금에 대한 자세한 내용은 [AWS Glue 요금](#)을 참조하세요.

애플리케이션과 동일한 AWS 계정 또는 다른 AWS 계정에서 AWS Glue Data Catalog를 사용하도록 EMR Serverless 작업을 구성할 수 있습니다.

## AWS Glue Data Catalog 구성

Data Catalog를 구성하려면 사용할 EMR Serverless 애플리케이션 유형을 선택합니다.

### Spark

EMR Studio를 사용하여 EMR Serverless Spark 애플리케이션에서 작업을 실행하는 경우 AWS Glue Data Catalog가 기본 메타스토어입니다.

SDK 또는 AWS CLI를 사용하는 경우 작업 실행의 `sparkSubmit` 파라미터에서 `spark.hadoop.hive.metastore.client.factory.class` 구성을 `com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory`로 설정할 수 있습니다. 다음 예제에서는 AWS CLI를 사용하여 Data Catalog를 구성하는 방법을 보여 줍니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/code/pyspark/
extreme_weather.py",
      "sparkSubmitParameters": "--conf
spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory
--conf spark.driver.cores=1 --conf spark.driver.memory=3g --conf
spark.executor.cores=4 --conf spark.executor.memory=3g"
    }
  }'
```

또는 Spark 코드에서 새 `SparkSession`을 생성할 때 이 구성을 설정할 수 있습니다.

```
from pyspark.sql import SparkSession

spark = (
    SparkSession.builder.appName("SparkSQL")
    .config(
        "spark.hadoop.hive.metastore.client.factory.class",
        "com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory",
    )
)
```

```

        .enableHiveSupport()
        .getOrCreate()
    )

    # we can query tables with SparkSQL
    spark.sql("SHOW TABLES").show()

    # we can also them with native Spark
    print(spark.catalog.listTables())

```

## Hive

EMR Serverless Hive 애플리케이션의 경우 Data Catalog가 기본 메타스토어입니다. 즉, EMR Serverless Hive 애플리케이션에서 작업을 실행할 때 Hive는 애플리케이션과 동일한 AWS 계정의 Data Catalog에 메타스토어 정보를 기록합니다. Data Catalog를 메타스토어로 사용하기 위해 가상 프라이빗 클라우드(VPC)가 필요하지 않습니다.

Hive 메타스토어 테이블에 액세스하려면 [AWS Glue에 대한 IAM 권한 설정](#)에서 설명하는 필수 AWS Glue 정책을 추가합니다.

## EMR Serverless 및 AWS Glue Data Catalog에 대한 교차 계정 액세스 구성

EMR Serverless에 대한 교차 계정 액세스를 설정하려면 먼저 다음 AWS 계정에 로그인합니다.

- AccountA - EMR Serverless 애플리케이션을 생성한 AWS 계정.
- AccountB - EMR Serverless 작업 실행에서 액세스하려는 AWS Glue Data Catalog가 포함된 AWS 계정.

1. AccountB의 관리자 또는 기타 승인된 자격 증명이 AccountB의 Data Catalog에 리소스 정책을 연결해야 합니다. 이 정책은 AccountB 카탈로그의 리소스에서 작업을 수행할 수 있는 AccountA별 교차 계정 권한을 부여합니다.

### JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",

```

```

    "Action": [
      "glue:GetDatabase",
      "glue:CreateDatabase",
      "glue:GetDataBases",
      "glue:CreateTable",
      "glue:GetTable",
      "glue:UpdateTable",
      "glue>DeleteTable",
      "glue:GetTables",
      "glue:GetPartition",
      "glue:GetPartitions",
      "glue:CreatePartition",
      "glue:BatchCreatePartition",
      "glue:GetUserDefinedFunctions"
    ],
    "Resource": [
      "arn:aws:glue:*:123456789012:catalog"
    ],
    "Sid": "AllowGLUEGetdatabase"
  }
]
}

```

2. 역할이 AccountB의 Data Catalog 리소스에 액세스할 수 있도록 AccountA에서 EMR Serverless 작업 런타임 역할에 IAM 정책을 추가합니다.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "glue:GetDatabase",
        "glue:CreateDatabase",
        "glue:GetDataBases",
        "glue:CreateTable",
        "glue:GetTable",
        "glue:UpdateTable",
        "glue>DeleteTable",
        "glue:GetTables",
        "glue:GetPartition",

```

```

        "glue:GetPartitions",
        "glue:CreatePartition",
        "glue:BatchCreatePartition",
        "glue:GetUserDefinedFunctions"
    ],
    "Resource": [
        "arn:aws:glue:*:123456789012:catalog"
    ],
    "Sid": "AllowGLUEGetdatabase"
  }
]
}

```

3. 작업 실행을 시작합니다. 이 단계는 AccountA의 EMR Serverless 애플리케이션 유형에 따라 약간 다릅니다.

### Spark

다음 예제와 같이 `spark.hadoop.hive.metastore.glue.catalogid`에서 `sparkSubmitParameters` 속성을 설정합니다. *AccountB-catalog-id*를 AccountB의 Data Catalog ID로 바꿉니다.

```

aws emr-serverless start-job-run \
--application-id "application-id" \
--execution-role-arn "job-role-arn" \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "s3://amzn-s3-demo-bucket/scripts/test.py",
        "sparkSubmitParameters": "--conf
spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.A
--conf spark.hadoop.hive.metastore.glue.catalogid=AccountB-catalog-
id --conf spark.executor.cores=1 --conf spark.executor.memory=1g
--conf spark.driver.cores=1 --conf spark.driver.memory=1g --conf
spark.executor.instances=1"
    }
}' \
--configuration-overrides '{
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/logs/"
        }
    }
}

```

```
}'
```

## Hive

다음 예제와 같이 `hive.metastore.glue.catalogid` 분류에서 `hive-site` 속성을 설정합니다. *AccountB-catalog-id*를 AccountB의 Data Catalog ID로 바꿉니다.

```
aws emr-serverless start-job-run \
--application-id "application-id" \
--execution-role-arn "job-role-arn" \
--job-driver '{
  "hive": {
    "query": "s3://amzn-s3-demo-bucket/hive/scripts/create_table.sql",
    "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/hive/warehouse"
  }
}' \
--configuration-overrides '{
  "applicationConfiguration": [{
    "classification": "hive-site",
    "properties": {
      "hive.metastore.glue.catalogid": "AccountB-catalog-id"
    }
  }]
}'
```

## AWS Glue Data Catalog 사용 시 고려 사항

Hive 스크립트에서 ADD JAR을 사용하여 보조 JAR을 추가할 수 있습니다. 추가 고려 사항은 [AWS Glue Data Catalog 사용 시 고려 사항](#)을 참조하세요.

## 외부 Hive 메타스토어 사용

Amazon Aurora 또는 Amazon RDS for MySQL과 같은 외부 Hive 메타스토어에 연결하도록 EMR Serverless Spark 및 Hive 작업을 구성할 수 있습니다. 이 섹션에서는 Amazon RDS Hive 메타스토어를 설정하고, VPC를 구성하며, 외부 메타스토어를 사용하도록 EMR Serverless 작업을 구성하는 방법을 설명합니다.

## 외부 Hive 메타스토어 생성

1. [VPC 생성](#)의 지침에 따라 프라이빗 서브넷이 있는 Amazon Virtual Private Cloud(Amazon VPC)를 생성합니다.
2. 새 Amazon VPC 및 프라이빗 서브넷을 사용하여 EMR Serverless 애플리케이션을 생성합니다. VPC에서 EMR Serverless 애플리케이션을 구성하는 경우 먼저 지정된 각 서브넷에 대해 탄력적 네트워크 인터페이스를 프로비저닝합니다. 그러면 지정된 보안 그룹이 해당 네트워크 인터페이스에 연결합니다. 이렇게 하면 애플리케이션 액세스 제어 기능이 지원됩니다. VPC를 설정하는 방법에 대한 자세한 내용은 [데이터에 연결하도록 EMR Serverless 애플리케이션에 대한 VPC 액세스 구성](#) 섹션을 참조하세요.
3. Amazon VPC의 프라이빗 서브넷에서 MySQL 또는 Aurora PostgreSQL 데이터베이스를 생성합니다. Amazon RDS 데이터베이스 생성 방법에 대한 자세한 내용은 [Amazon RDS DB 인스턴스 생성](#)을 참조하세요.
4. [Amazon RDS DB 인스턴스 수정](#)의 단계에 따라 EMR Serverless 보안 그룹에서 JDBC 연결을 허용하도록 MySQL 또는 Aurora 데이터베이스의 보안 그룹을 수정합니다. EMR Serverless 보안 그룹 중 하나에서 RDS 보안 그룹에 인바운드 트래픽 규칙을 추가합니다.

유형	프로토콜	포트 범위	소스
모든 TCP	TCP	3306	emr-serverless-security-group

## Spark 옵션 구성

### JDBC 사용

Amazon RDS for MySQL 또는 Amazon Aurora MySQL 인스턴스를 기반으로 Hive 메타스토어에 연결하도록 EMR Serverless Spark 애플리케이션을 구성하려면 JDBC 연결을 사용합니다. 작업 실행의 `spark-submit` 파라미터에서 `--jars`와 함께 `mariadb-connector-java.jar`을 전달합니다.

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/scripts/spark-jdbc.py",
```

```

        "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-
connector-java.jar
        --conf
        spark.hadoop.java.jdbc.option.ConnectionDriverName=org.mariadb.jdbc.Driver
        --conf spark.hadoop.java.jdbc.option.ConnectionUserName=<connection-user-
name>
        --conf spark.hadoop.java.jdbc.option.ConnectionPassword=<connection-
password>
        --conf spark.hadoop.java.jdbc.option.ConnectionURL=<JDBC-Connection-
string>
        --conf spark.driver.cores=2
        --conf spark.executor.memory=10G
        --conf spark.driver.memory=6G
        --conf spark.executor.cores=4"
    }
}' \
--configuration-overrides '{
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {
            "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
        }
    }
}'

```

다음 코드 예제는 Amazon RDS에서 Hive 메타스토어와 상호 작용하는 Spark 진입점 스크립트입니다.

```

from os.path import expanduser, join, abspath
from pyspark.sql import SparkSession
from pyspark.sql import Row
# warehouse_location points to the default location for managed databases and tables
warehouse_location = abspath('spark-warehouse')
spark = SparkSession \
    .builder \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .enableHiveSupport() \
    .getOrCreate()
spark.sql("SHOW DATABASES").show()
spark.sql("CREATE EXTERNAL TABLE `sampledb`.`sparknyctaxi`(`dispatching_base_num`
    string, `pickup_datetime` string, `dropoff_datetime` string, `pulocationid` bigint,
    `dolocationid` bigint, `sr_flag` bigint) STORED AS PARQUET LOCATION 's3://<s3 prefix>/
nyctaxi_parquet/'")
spark.sql("SELECT count(*) FROM sampledb.sparknyctaxi").show()
spark.stop()

```

## Thrift 서비스 사용

Amazon RDS for MySQL 또는 Amazon Aurora MySQL 인스턴스를 기반으로 Hive 메타스토어에 연결하도록 EMR Serverless Hive 애플리케이션을 구성할 수 있습니다. 이를 수행하려면 기존 Amazon EMR 클러스터의 기본 노드에서 Thrift 서버를 실행합니다. 이 옵션은 EMR Serverless 작업 구성을 단순화하는 데 사용할 Thrift 서버가 있는 Amazon EMR 클러스터가 이미 있는 경우에 적합합니다.

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/thriftscript.py",
      "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-
connector-java.jar
      --conf spark.driver.cores=2
      --conf spark.executor.memory=10G
      --conf spark.driver.memory=6G
      --conf spark.executor.cores=4"
    }
  }' \
  --configuration-overrides '{
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
      }
    }
  }'
```

다음 코드 예제는 Thrift 프로토콜을 사용하여 Hive 메타스토어에 연결하는 진입점 스크립트 (thriftscript.py)입니다. hive.metastore.uris 속성은 외부 Hive 메타스토어에서 읽도록 설정해야 합니다.

```
from os.path import expanduser, join, abspath
from pyspark.sql import SparkSession
from pyspark.sql import Row
# warehouse_location points to the default location for managed databases and tables
warehouse_location = abspath('spark-warehouse')
spark = SparkSession \
    .builder \
    .config("spark.sql.warehouse.dir", warehouse_location) \
    .config("hive.metastore.uris", "thrift://thrift-server-host:thrift-server-port") \
```

```

.enableHiveSupport() \
.getOrCreate()
spark.sql("SHOW DATABASES").show()
spark.sql("CREATE EXTERNAL TABLE sampledby.`sparknyctaxi`(`dispatching_base_num`
  string, `pickup_datetime` string, `dropoff_datetime` string, `pulocationid` bigint,
  `dolocationid` bigint, `sr_flag` bigint) STORED AS PARQUET LOCATION 's3://<s3 prefix>/
nyctaxi_parquet/'")
spark.sql("SELECT * FROM sampledby.sparknyctaxi").show()
spark.stop()

```

## Hive 옵션 구성

### JDBC 사용

Amazon RDS MySQL 또는 Amazon Aurora 인스턴스에서 외부 Hive 데이터베이스 위치를 지정하려는 경우 기본 메타스토어 구성을 재정의할 수 있습니다.

#### Note

Hive에서 메타스토어 테이블에 대한 여러 쓰기를 동시에 수행할 수 있습니다. 두 클러스터 사이에서 메타스토어 정보를 공유하는 경우, 동일한 메타스토어 테이블의 다른 파티션에 쓰고 있지 않은 한, 동시에 동일한 메타스토어 테이블에 쓰지 않도록 해야 합니다.

hive-site 분류에서 다음 구성을 설정하여 외부 Hive 메타스토어를 활성화합니다.

```

{
  "classification": "hive-site",
  "properties": {
    "hive.metastore.client.factory.class":
"org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClientFactory",
    "javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",
    "javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-port/db-name",
    "javax.jdo.option.ConnectionUserName": "username",
    "javax.jdo.option.ConnectionPassword": "password"
  }
}

```

### Thrift 서버 사용

Amazon RDS for MySQL 또는 Amazon Aurora MySQL 인스턴스를 기반으로 Hive 메타스토어에 연결하도록 EMR Serverless Hive 애플리케이션을 구성할 수 있습니다. 이를 수행하려면 기존 Amazon

EMR 클러스터의 메인 노드에서 Thrift 서버를 실행합니다. 이 옵션은 Thrift 서버를 실행하고 EMR Serverless 작업 구성을 사용하려는 Amazon EMR 클러스터가 이미 있는 경우에 적합합니다.

EMR Serverless가 원격 Thrift 메타스토어에 액세스할 수 있도록 hive-site 분류에서 다음 구성을 설정합니다. 외부 Hive 메타스토어에서 읽도록 hive.metastore.uris 속성을 설정해야 합니다.

```
{
  "classification": "hive-site",
  "properties": {
    "hive.metastore.client.factory.class":
    "org.apache.hadoop.hive.q1.metadata.SessionHiveMetaStoreClientFactory",
    "hive.metastore.uris": "thrift://thrift-server-host:thrift-server-port"
  }
}
```

## EMR Serverless에서 AWS Glue 다중 카탈로그 계층 구조 작업

AWS Glue 다중 카탈로그 계층 구조와 함께 작동하도록 EMR Serverless 애플리케이션을 구성할 수 있습니다. 다음 예제에서는 AWS Glue 다중 카탈로그 계층 구조에서 EMR-S Spark를 사용하는 방법을 보여줍니다.

다중 카탈로그 계층 구조에 대해 자세히 알아보려면 [Amazon EMR에서 Spark를 사용하여 AWS Glue Data Catalog의 다중 카탈로그 계층 구조 작업](#)을 참조하세요.

## Iceberg 및 AWS Glue Data Catalog와 함께 Redshift Managed Storage(RMS) 사용

다음은 Iceberg와 AWS Glue Data Catalog의 통합을 위해 Spark를 구성하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/myscript.py",
      "sparkSubmitParameters": "--conf spark.sql.catalog.nfgac_rms =
org.apache.iceberg.spark.SparkCatalog
      --conf spark.sql.catalog.rms.type=glue
      --conf spark.sql.catalog.rms.glue.id=Glue RMS catalog ID
      --conf spark.sql.defaultCatalog=rms
      --conf
      spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
```

```
}
}'
```

통합 후 카탈로그에 있는 테이블의 샘플 쿼리:

```
SELECT * FROM my_rms_schema.my_table
```

## Iceberg REST API 및 AWS Glue Data Catalog와 함께 Redshift Managed Storage(RMS) 사용

다음은 Spark를 Iceberg REST 카탈로그와 함께 작동하도록 구성하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
--application-id application-id \
--execution-role-arn job-role-arn \
--job-driver '{
"sparkSubmit": {
"entryPoint": "s3://amzn-s3-demo-bucket/myscript.py",
"sparkSubmitParameters": "
--conf spark.sql.catalog.rms=org.apache.iceberg.spark.SparkCatalog
--conf spark.sql.catalog.rms.type=rest
--conf spark.sql.catalog.rms.warehouse=Glue RMS catalog ID
--conf spark.sql.catalog.rms.uri=Glue endpoint URI/iceberg
--conf spark.sql.catalog.rms.rest.sigv4-enabled=true
--conf spark.sql.catalog.rms.rest.signing-name=glue
--conf
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions"
}'
```

카탈로그에 있는 테이블의 샘플 쿼리:

```
SELECT * FROM my_rms_schema.my_table
```

## 외부 메타스토어 사용 시 고려 사항

- MariaDB JDBC와 호환되는 데이터베이스를 메타스토어로 구성할 수 있습니다. 이러한 데이터베이스의 예로는, RDS for MariaDB, MySQL, Amazon Aurora가 있습니다.
- 메타스토어는 자동으로 초기화되지 않습니다. 메타스토어가 Hive 버전의 스키마로 초기화되지 않은 경우 [Hive 스키마 도구](#)를 사용합니다.

- EMR Serverless는 Kerberos 인증을 지원하지 않습니다. EMR Serverless Spark 또는 Hive 작업에서는 Kerberos 인증과 함께 Thrift 메타스토어 서버를 사용할 수 없습니다.
- 다중 카탈로그 계층 구조를 사용하도록 VPC 액세스를 구성합니다.

## EMR Serverless에서 다른 AWS 계정의 S3 데이터에 액세스

한 AWS 계정에서 Amazon EMR Serverless 작업을 실행하고 다른 AWS 계정에 속한 Amazon S3 버킷의 데이터에 액세스하도록 구성할 수 있습니다. 이 페이지에서는 EMR Serverless에서 S3에 대한 교차 계정 액세스를 구성하는 방법을 설명합니다.

EMR Serverless에서 실행되는 작업은 S3 버킷 정책 또는 수임된 역할을 사용하여 다른 AWS 계정에서 Amazon S3의 데이터에 액세스할 수 있습니다.

### 사전 조건

Amazon EMR Serverless에서 교차 계정 액세스를 설정하려면 다음 두 AWS 계정에 로그인한 상태에서 작업을 완료합니다.

- **AccountA** - Amazon EMR Serverless 애플리케이션을 생성한 AWS 계정입니다. 교차 계정 액세스를 설정하기 전에 이 계정에서 다음과 같은 준비를 완료합니다.
  - 작업을 실행하려는 Amazon EMR Serverless 애플리케이션.
  - 애플리케이션에서 작업을 실행하는 데 필요한 권한이 있는 작업 실행 역할. 자세한 정보는 [Amazon EMR Serverless에 대한 작업 런타임 역할](#) 섹션을 참조하세요.
- **AccountB** - Amazon EMR Serverless 작업에서 액세스하려는 S3 버킷이 포함된 AWS 계정입니다.

### S3 버킷 정책을 사용하여 교차 계정 S3 데이터에 액세스

account A에서 account B의 S3 버킷에 액세스하려면 다음 정책을 account B의 S3 버킷에 연결합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ExamplePermissions1",
      "Effect": "Allow",
```

```

    "Action": [
      "s3:ListBucket"
    ],
    "Resource": [
      "arn:aws:s3:::my-bucket-name"
    ]
  },
  {
    "Sid": "ExamplePermissions2",
    "Effect": "Allow",
    "Action": [
      "s3:PutObject",
      "s3:GetObject",
      "s3:DeleteObject"
    ],
    "Resource": [
      "arn:aws:s3:::my-bucket-name/*"
    ]
  }
]
}

```

S3 버킷 정책을 통한 S3 교차 계정 액세스에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [예제 2: 버킷 소유자가 교차 계정 버킷 권한 부여](#)를 참조하세요.

## 수입된 역할을 사용하여 교차 계정 S3 데이터에 액세스

Amazon EMR Serverless에 대한 교차 계정 액세스를 설정하는 또 다른 방법은 AWS Security Token Service(AWS STS)에서 AssumeRole 작업을 사용하는 것입니다. AWS STS는 사용자가 사용할 권한이 제한된 임시 자격 증명을 요청할 수 있는 글로벌 웹 서비스입니다. AssumeRole로 생성한 임시 보안 자격 증명을 사용하여 EMR Serverless 및 Amazon S3에 대해 API 직접 호출을 수행할 수 있습니다.

다음 단계에서는 수입된 역할을 사용하여 EMR Serverless에서 교차 계정 S3 데이터에 액세스하는 방법을 보여줍니다.

1. AccountB에서 Amazon S3 버킷(*cross-account-bucket*)을 생성합니다. 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [버킷 생성](#)을 참조하세요. DynamoDB에 대한 크로스 계정 액세스를 원하는 경우 AccountB에서 DynamoDB 테이블을 생성합니다. 자세한 내용은 Amazon DynamoDB 개발자 안내서의 [DynamoDB 테이블 생성](#)을 참조하세요.
2. **## ## ##**에 액세스할 수 AccountB에서 Cross-Account-Role-B IAM 역할을 생성합니다.

- a. AWS Management 콘솔에 로그인하여 <https://console.aws.amazon.com/iam/> 에서 IAM 콘솔을 엽니다.
- b. 역할을 선택하고 새 역할(Cross-Account-Role-B)을 생성합니다. IAM 역할 생성에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 역할 생성](#)을 참조하세요.
- c. 다음 정책 명령문에서 볼 수 있듯이 **cross-account-bucket** S3 버킷에 액세스할 수 있는 Cross-Account-Role-B에 대한 권한을 지정하는 IAM 정책을 생성합니다. 그런 다음, IAM 정책을 Cross-Account-Role-B에 연결합니다. 자세한 내용은 IAM 사용자 설명서에서 [IAM 정책 생성](#)을 참조하세요.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:*"
      ],
      "Resource": [
        "arn:aws:s3:::cross-account-bucket",
        "arn:aws:s3:::cross-account-bucket/*"
      ],
      "Sid": "AllowS3"
    }
  ]
}
```

DynamoDB 액세스가 필요한 경우 교차 계정 DynamoDB 테이블에 액세스할 권한을 지정하는 IAM 정책을 생성합니다. 그런 다음, IAM 정책을 Cross-Account-Role-B에 연결합니다. 자세한 내용은 IAM 사용 설명서의 [Amazon DynamoDB: 특정 테이블에 대한 액세스 허용](#)을 참조하세요.

다음은 DynamoDB 테이블(CrossAccountTable)에 대한 액세스를 허용하는 정책입니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```

    "Action": [
      "dynamodb:*"
    ],
    "Resource": [
      "arn:aws:dynamodb:*:123456789012:table/CrossAccountTable"
    ],
    "Sid": "AllowDYNAMODB"
  }
]
}

```

### 3. Cross-Account-Role-B 역할에 대한 신뢰 관계를 편집합니다.

- a. 역할에 대한 신뢰 관계를 구성하려면 2단계에서 생성한 역할(Cross-Account-Role-B)에 대해 IAM 콘솔에서 신뢰 관계 탭을 선택합니다.
- b. 신뢰 관계 편집을 선택합니다.
- c. 다음 정책 문서를 추가합니다. 그러면 AccountA에서 Job-Execution-Role-A가 Cross-Account-Role-B 역할을 수임할 수 있습니다.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/Job-Execution-Role-A",
      "Sid": "AllowSTSAssumerole"
    }
  ]
}

```

4. Cross-Account-Role-B를 수임하기 위해 AWS STS AssumeRole 권한을 AccountA의 Job-Execution-Role-A에 부여합니다.
  - a. AWS 계정 AccountA에 대한 IAM 콘솔에서 Job-Execution-Role-A를 선택합니다.
  - b. 다음 정책 명령을 Job-Execution-Role-A에 추가하여 Cross-Account-Role-B 역할에서 AssumeRole 작업을 허용합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/Cross-Account-Role-B"
      ],
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

## 수입된 역할 예제

계정 내 모든 S3 리소스에 액세스하기 위해 단일 역할을 사용하거나, Amazon EMR 6.11 이상 버전에서는 서로 다른 계정 간 S3 버킷에 액세스할 때 적용할 여러 IAM 역할을 구성할 수 있습니다.

## 주제

- [하나의 수입된 역할로 S3 리소스에 액세스](#)
- [여러 수입된 역할로 S3 리소스에 액세스](#)

## 하나의 수입된 역할로 S3 리소스에 액세스

 Note

단일 수입된 역할을 사용하도록 작업을 구성하면 `entryPoint` 스크립트를 포함하여 작업 전 반의 모든 S3 리소스가 해당 역할을 사용합니다.

단일 수입된 역할을 사용하여 계정 B의 모든 S3 리소스에 액세스하려면 다음 구성을 지정합니다.

1. EMRFS 구성 `fs.s3.customAWSCredentialsProvider`를 `com.amazonaws.emr.AssumeRoleAWSCredentialsProvider`로 지정합니다.
2. Spark의 경우 `spark.emr-serverless.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN` 및 `spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`을 사용하여 드라이버 및 실행기에서 환경 변수를 지정합니다.
3. Hive의 경우 `hive.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`, `tez.am.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`, `tez.task.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN`을 사용하여 Hive 드라이버, Tez 애플리케이션 기본 및 Tez 태스크 컨테이너에서 환경 변수를 지정합니다.

다음 예제에서는 가정된 역할을 사용하여 교차 계정 액세스로 EMR Serverless 작업 실행을 시작하는 방법을 보여줍니다.

## Spark

다음 예제에서는 S3에 대한 교차 계정 액세스로 EMR Serverless Spark 작업 실행을 시작하기 위해 수임된 역할을 사용하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "entrypoint_location",
      "entryPointArguments": [":argument_1:", ":argument_2:"],
      "sparkSubmitParameters": "--conf spark.executor.cores=4 --conf
spark.executor.memory=20g --conf spark.driver.cores=4 --conf spark.driver.memory=8g
--conf spark.executor.instances=1"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "spark-defaults",
      "properties": {
        "spark.hadoop.fs.s3.customAWSCredentialsProvider":
"com.amazonaws.emr.AssumeRoleAWSCredentialsProvider",
        "spark.emr-serverless.driverEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
"arn:aws:iam::AccountB:role/Cross-Account-Role-B",
```

```

        "spark.executorEnv.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
        "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
    }
  ]]
}'

```

## Hive

다음 예제에서는 S3에 대한 교차 계정 액세스를 EMR Serverless Hive 작업 실행을 시작하기 위해 수임된 역할을 사용하는 방법을 보여줍니다.

```

aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "query_location",
      "parameters": "hive_parameters"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "fs.s3.customAWSCredentialsProvider":
        "com.amazonaws.emr.serverless.credentialsprovider.AssumeRoleAWSCredentialsProvider",
        "hive.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
        "arn:aws:iam::AccountB:role/Cross-Account-Role-B",
        "tez.am.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
        "arn:aws:iam::AccountB:role/Cross-Account-Role-B",
        "tez.task.emr-serverless.launch.env.ASSUME_ROLE_CREDENTIALS_ROLE_ARN":
        "arn:aws:iam::AccountB:role/Cross-Account-Role-B"
      }
    }
  ]
}'

```

## 여러 수임된 역할로 S3 리소스에 액세스

EMR Serverless 릴리스 6.11.0 이상을 사용하면 여러 교차 계정 버킷에 액세스할 때 가정할 여러 IAM 역할을 구성할 수 있습니다. 계정 B의 여러 수임된 역할을 사용하여 다른 S3 리소스에 액세스하려면 작업 실행을 시작할 때 다음 구성을 사용합니다.

## 1. EMRFS 구성 fs.s3.customAWSCredentialsProvider를

com.amazonaws.emr.serverless.credentialsprovider.BucketLevelAssumeRoleCredentialsProvider를 지정합니다.

## 2. S3 버킷 이름에서 수입할 계정 B의 IAM 역할로의 매핑을 정의하려면 EMRFS 구성

fs.s3.bucketLevelAssumeRoleMapping을 지정합니다. 값은 bucket1->role1;bucket2->role2 형식이어야 합니다.

예를 들어, arn:aws:iam::*AccountB*:role/Cross-Account-Role-B-1을 사용하여 버킷 bucket1에 액세스하고 arn:aws:iam::*AccountB*:role/Cross-Account-Role-B-2를 사용하여 버킷 bucket2에 액세스할 수 있습니다. 다음 예시는 여러 계정 간 역할을 통해 계정 간 액세스를 EMR Serverless 작업 실행을 시작하는 방법을 보여줍니다.

## Spark

다음 예제에서는 여러 수입된 역할을 사용하여 EMR Serverless Spark 작업 실행을 생성하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "entrypoint_location",
      "entryPointArguments": [":argument_1:", ":argument_2:"],
      "sparkSubmitParameters": "--conf spark.executor.cores=4 --conf
spark.executor.memory=20g --conf spark.driver.cores=4 --conf spark.driver.memory=8g
--conf spark.executor.instances=1"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "spark-defaults",
      "properties": {
        "spark.hadoop.fs.s3.customAWSCredentialsProvider":
"com.amazonaws.emr.serverless.credentialsprovider.BucketLevelAssumeRoleCredentialsProvider"
        "spark.hadoop.fs.s3.bucketLevelAssumeRoleMapping":
"bucket1->arn:aws:iam::AccountB:role/Cross-Account-Role-B-1;bucket2-
>arn:aws:iam::AccountB:role/Cross-Account-Role-B-2"
      }
    }]
  }
```

}'

## Hive

다음 예는 가정된 여러 역할을 사용하여 EMR Serverless Hive 작업 실행을 생성하는 방법을 보여줍니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "query_location",
      "parameters": "hive_parameters"
    }
  }' \
  --configuration-overrides '{
    "applicationConfiguration": [{
      "classification": "hive-site",
      "properties": {
        "fs.s3.customAWSCredentialsProvider":
"com.amazonaws.emr.serverless.credentialsprovider.AssumeRoleAWSCredentialsProvider",
        "fs.s3.bucketLevelAssumeRoleMapping": "bucket1-
>arn:aws:iam::AccountB:role/Cross-Account-Role-B-1;bucket2-
>arn:aws:iam::AccountB:role/Cross-Account-Role-B-2"
      }
    }]
  }'
```

## EMR Serverless에서 오류 해결

다음 정보를 사용하여 Amazon EMR Serverless 작업 시 발생하는 일반적인 문제를 진단하고 수정할 수 있습니다.

### 주제

- [오류: 계정이 동시에 사용할 수 있는 최대 vCPU의 서비스 제한에 도달하여 작업이 실패했습니다.](#)
- [오류: 애플리케이션이 maximumCapacity 설정을 초과하여 작업이 실패했습니다.](#)
- [오류: 애플리케이션이 maximumCapacity를 초과하여 워커를 할당할 수 없어 작업이 실패했습니다.](#)
- [오류: S3 액세스가 거부되었습니다. 필요한 S3 리소스에서 작업 런타임 역할의 S3 액세스 권한을 확인하세요.](#)

- 오류: `ModuleNotFoundError: <module> 모듈이 없습니다.` EMR Serverless에서 Python 라이브러리를 사용하는 방법은 사용 설명서를 참조하세요.
- 오류: `<role name> 역할을 수임할 수 없습니다.` 해당 역할이 없거나 필요한 신뢰 관계로 설정되지 않았기 때문입니다.

오류: 계정이 동시에 사용할 수 있는 최대 vCPU의 서비스 제한에 도달하여 작업이 실패했습니다.

이 오류는 계정이 최대 용량을 초과했기 때문에 EMR Serverless에서 작업을 제출할 수 없음을 나타냅니다. 계정의 최대 용량을 늘립니다. [EMR Serverless Service Quotas](#)에서 서비스 한도를 확인합니다.

오류: 애플리케이션이 `maximumCapacity` 설정을 초과하여 작업이 실패했습니다.

이 오류는 애플리케이션이 구성된 최대 용량을 초과했기 때문에 EMR Serverless에서 작업을 제출할 수 없음을 나타냅니다. 애플리케이션의 최대 용량을 늘립니다.

오류: 애플리케이션이 `maximumCapacity`를 초과하여 워커를 할당할 수 없어 작업이 실패했습니다.

이 오류는 작업을 완료할 수 없음을 나타냅니다. 애플리케이션이 `maximumCapacity` 설정을 초과했기 때문에 워커를 할당할 수 없습니다.

오류: S3 액세스가 거부되었습니다. 필요한 S3 리소스에서 작업 런타임 역할의 S3 액세스 권한을 확인하세요.

이 오류는 S3 리소스에 대한 액세스 권한이 작업에 없음을 나타냅니다. 작업에서 사용해야 하는 S3 리소스에 액세스할 수 있는 권한이 작업 런타임 역할에 있는지 확인합니다. 런타임 역할에 대한 자세한 내용은 [Amazon EMR Serverless에 대한 작업 런타임 역할](#) 섹션을 참조하세요.

오류: `ModuleNotFoundError: <module> 모듈이 없습니다.` EMR Serverless에서 Python 라이브러리를 사용하는 방법은 사용 설명서를 참조하세요.

이 오류는 Spark 작업에 대해 Python 모듈을 사용할 수 없음을 나타냅니다. 종속 Python 라이브러리를 작업에 사용할 수 있는지 확인합니다. Python 라이브러리를 패키징하는 방법에 대한 자세한 내용은 [EMR Serverless에서 Python 라이브러리 사용](#) 섹션을 참조하세요.

오류: <role name> 역할을 수임할 수 없습니다. 해당 역할이 없거나 필요한 신뢰 관계로 설정되지 않았기 때문입니다.

이 오류는 작업에 대해 지정한 작업 런타임 역할이 존재하지 않거나 EMR Serverless 권한에 대한 신뢰 관계가 역할에 없음을 나타냅니다. IAM 역할이 있는지 확인하고 역할의 신뢰 정책을 올바르게 설정했는지 확인하려면 [Amazon EMR Serverless에 대한 작업 런타임 역할](#)의 지침을 참조하세요.

# EMR Studio를 통해 EMR Serverless에서 대화형 워크로드 실행

EMR Serverless 대화형 애플리케이션을 사용하면 EMR Studio에서 호스팅되는 노트북을 사용해 EMR Serverless에서 Spark에 대한 대화형 워크로드를 실행할 수 있습니다.

## 개요

대화형 애플리케이션은 대화형 기능이 활성화된 EMR Serverless 애플리케이션입니다. Amazon EMR Serverless 대화형 애플리케이션을 사용하면 Amazon EMR Studio에서 관리하는 Jupyter Notebook을 사용하여 대화형 워크로드를 실행할 수 있습니다. 이를 통해 데이터 엔지니어, 데이터 과학자 및 데이터 분석가는 EMR Studio를 사용하여 Amazon S3 및 Amazon DynamoDB와 같은 데이터 저장소의 데이터세트로 대화형 분석을 실행할 수 있습니다.

EMR Serverless의 대화형 애플리케이션에 대한 사용 사례로 다음이 포함됩니다.

- 데이터 엔지니어는 EMR Studio에서 IDE 환경을 사용하여 ETL 스크립트를 생성합니다. 스크립트는 온프레미스에서 데이터를 수집하고, 분석을 위해 데이터를 변환하며, Amazon S3에 데이터를 저장합니다.
- 데이터 과학자는 노트북을 사용하여 데이터세트를 탐색하고 데이터세트에서 이상을 감지하도록 기계 학습(ML) 모델을 훈련합니다.
- 데이터 분석가는 데이터세트를 탐색하고 비즈니스 대시보드와 같은 애플리케이션을 업데이트하기 위해 일일 보고서를 생성하는 스크립트를 생성합니다.

## 사전 조건

EMR Serverless에서 대화형 워크로드를 사용하려면 다음 요구 사항을 충족합니다.

- EMR Serverless 대화형 애플리케이션은 Amazon EMR 6.14.0 이상에서 지원됩니다.
- 대화형 애플리케이션에 액세스하고, 제출하는 워크로드를 실행하며, EMR Studio에서 대화형 노트북을 실행하려면 특정 권한과 역할이 필요합니다. 자세한 정보는 [대화형 워크로드에 필요한 권한](#) 섹션을 참조하세요.

## 대화형 워크로드에 필요한 권한

[EMR Serverless에 액세스하는 데 필요한 기본 권한](#) 외에도 IAM 자격 증명 또는 역할에 대한 추가 권한을 구성합니다.

대화형 애플리케이션에 액세스하는 방법

EMR Studio에 대한 사용자 및 워크스페이스 권한을 설정합니다. 자세한 내용은 Amazon EMR 관리 안내서의 [인스턴스 플릿 구성](#)을 참조하세요.

EMR Serverless에서 제출하는 워크로드를 실행하는 방법

작업 런타임 역할 설정. 자세한 정보는 [작업 런타임 역할 생성](#) 섹션을 참조하세요.

EMR Studio에서 대화형 노트북을 실행하는 방법

다음 추가 권한을 EMR Studio 사용자의 IAM 정책에 추가합니다.

- **emr-serverless:AccessInteractiveEndpoints** - 로 지정한 대화형 애플리케이션에 액세스하고 연결할 수 있는 권한을 부여합니다..Resource 이 권한은 EMR Studio Workspace에서 EMR Serverless 애플리케이션에 연결하는 데 필요합니다.
- **iam:PassRole** - 애플리케이션에 연결할 때 사용할 IAM 실행 역할에 액세스할 수 있는 권한을 부여합니다. EMR Studio Workspace에서 EMR Serverless 애플리케이션에 연결하려면 적절한 PassRole 권한이 필요합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessInteractiveAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:AccessInteractiveEndpoints"
      ],
      "Resource": [
        "arn:aws:emr-serverless:*:123456789012:/applications/*"
      ]
    },
    {
      "Sid": "EMRServerlessRuntimeRoleAccess",
      "Effect": "Allow",
      "Action": [
```

```

        "iam:PassRole"
    ],
    "Resource": [
        "arn:aws:iam::123456789012:role/EMRServerlessInteractiveRole"
    ],
    "Condition": {
        "StringLike": {
            "iam:PassedToService": "emr-serverless.amazonaws.com"
        }
    }
}
]
}

```

## 대화형 애플리케이션 구성

다음의 개략적인 단계를 사용하여 AWS Management 콘솔의 Amazon EMR Studio에서 대화형 기능을 통해 EMR Serverless 애플리케이션을 생성합니다.

1. [Amazon EMR Serverless 시작하기](#)의 단계를 수행하여 애플리케이션을 생성합니다.
2. 그런 다음, EMR Studio에서 워크스페이스를 시작하고 컴퓨팅 옵션으로 EMR Serverless 애플리케이션에 연결합니다. 자세한 내용은 [EMR Serverless 시작하기](#) 설명서의 2단계에 있는 대화형 워크로드 탭을 참조하세요.

Studio Workspace에 애플리케이션을 연결하면 아직 실행되지 않은 경우 애플리케이션 시작이 자동으로 트리거됩니다. 애플리케이션을 사전에 시작하고 워크스페이스에 연결하기 전에 준비할 수도 있습니다.

## 대화형 애플리케이션에서의 고려 사항

- EMR Serverless 대화형 애플리케이션은 Amazon EMR 6.14.0 이상에서 지원됩니다.
- EMR Studio는 EMR Serverless 대화형 애플리케이션과 통합된 유일한 클라이언트입니다. 워크스페이스 협업, SQL 탐색기 및 노트북의 프로그래밍 방식 실행과 같은 EMR Studio 기능은 EMR Serverless 대화형 애플리케이션에서 지원되지 않습니다.
- 대화형 애플리케이션은 Spark 엔진에서만 지원됩니다.
- 대화형 애플리케이션은 Python 3, PySpark 및 Spark Scala 커널을 지원합니다.
- 단일 대화형 애플리케이션에서 최대 25개의 동시 노트북을 실행할 수 있습니다.

- 대화형 애플리케이션에서 자체 호스팅 Jupyter Notebook을 지원하는 엔드포인트 또는 API 인터페이스는 없습니다.
- 최적화된 시작 환경을 위해 드라이버 및 실행기에 대해 사전 초기화된 용량을 구성하고 애플리케이션을 사전에 시작하는 것이 좋습니다. 애플리케이션을 사전에 시작하는 경우 워크스페이스에 연결할 준비가 되었는지 확인합니다.

```
aws emr-serverless start-application \
--application-id your-application-id
```

- 기본적으로 autoStopConfig는 애플리케이션에 대해 활성화됩니다. 그러면 30분의 유휴 시간 후에 애플리케이션이 종료됩니다. create-application 또는 update-application 요청의 일부로 이 구성을 변경할 수 있습니다.
- 대화형 애플리케이션을 사용하는 경우 노트북을 실행하도록 커널, 드라이버 및 실행기의 사전 초기화된 용량을 구성하는 것이 좋습니다. 각 Spark 대화형 세션에는 커널 하나와 드라이버 하나가 필요하므로 EMR Serverless는 사전 초기화된 모든 드라이버에 대해 사전 초기화된 커널 작업자를 유지 관리합니다. 기본적으로 EMR Serverless는 드라이버에 대해 사전 초기화된 용량을 지정하지 않더라도 전체 애플리케이션 전체에서 커널 작업자 1개의 사전 초기화된 용량을 유지 관리합니다. 각 커널 작업자는 4개의 vCPU와 16GB의 메모리를 사용합니다. 현재 요금 정보는 [Amazon EMR 요금](#) 페이지를 참조하세요.
- 대화형 워크로드를 실행하려면 AWS 계정에 vCPU 서비스 할당량이 충분해야 합니다. Lake Formation 지원 워크로드를 실행하지 않는 경우 최소 24개의 vCPU를 사용하는 것이 좋습니다. 이 경우 28개 이상의 vCPU를 사용하는 것이 좋습니다.
- 60분 넘게 유휴 상태인 경우 EMR Serverless는 노트북에서 커널을 자동으로 종료합니다. EMR Serverless는 노트북 세션 중에 완료된 마지막 활동의 커널 유휴 시간을 계산합니다. 현재 커널 유휴 제한 시간 설정은 수정할 수 없습니다.
- 대화형 워크로드로 Lake Formation을 활성화하려면 [EMR Serverless 애플리케이션을 생성할 때 runtime-configuration 객체의 spark-defaults 분류에 따라 spark.emr-serverless.lakeformation.enabled 구성을 true로 설정합니다](#). 자세한 내용은 [Amazon EMR에서 Lake Formation 활성화](#)를 참조하세요.

# Apache Livy 엔드포인트를 통해 EMR Serverless에서 대화형 워크로드 실행

Amazon EMR 릴리스 6.14.0 이상을 사용하면 EMR Serverless 애플리케이션을 생성할 때 Apache Livy 엔드포인트를 생성 및 활성화하고 자체 호스팅 노트북 또는 사용자 지정 클라이언트를 통해 대화형 워크로드를 실행할 수 있습니다. Apache Livy 엔드포인트는 다음과 같은 이점을 제공합니다.

- Jupyter Notebook을 통해 Apache Livy 엔드포인트에 안전하게 연결하고 Apache Livy의 REST 인터페이스를 사용하여 Apache Spark 워크로드를 관리할 수 있습니다.
- Apache Spark 워크로드의 데이터를 사용하는 대화형 웹 애플리케이션에 대해 Apache Livy REST API 작업을 사용합니다.

## 사전 조건

EMR Serverless에서 Apache Livy 엔드포인트를 사용하려면 다음 요구 사항을 충족합니다.

- [Amazon EMR Serverless 시작하기](#)의 단계를 완료합니다.
- Apache Livy 엔드포인트를 통해 대화형 워크로드를 실행하려면 특정 권한과 역할이 필요합니다. 자세한 내용은 [대화형 워크로드에 필요한 권한](#)을 참조하세요.

## 필수 권한

EMR Serverless에 액세스하는 데 필요한 권한 외에도 Apache Livy 엔드포인트에 액세스하고 애플리케이션을 실행하려면 IAM 역할에 다음 권한도 추가합니다.

- `emr-serverless:AccessLivyEndpoints - Resource`로 지정한 Livy 지원 애플리케이션에 액세스하고 연결할 수 있는 권한을 부여합니다. Apache Livy 엔드포인트에서 사용할 수 있는 REST API 작업을 실행하려면 이 권한이 필요합니다.
- `iam:PassRole` - Apache Livy 세션을 생성할 때 IAM 실행 역할에 액세스할 수 있는 권한을 부여합니다. EMR Serverless는 이 역할을 사용하여 워크로드를 실행합니다.
- `emr-serverless:GetDashboardForJobRun` - Spark Live UI 및 드라이버 로그 링크를 생성할 수 있는 권한을 부여하고 Apache Livy 세션 결과의 일부로 로그에 대한 액세스를 제공합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessInteractiveAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:AccessLivyEndpoints"
      ],
      "Resource": [
        "arn:aws:emr-serverless*:123456789012:/applications/*"
      ]
    },
    {
      "Sid": "EMRServerlessRuntimeRoleAccess",
      "Effect": "Allow",
      "Action": [
        "iam:PassRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/EMRServerlessExecutionRole"
      ],
      "Condition": {
        "StringLike": {
          "iam:PassedToService": "emr-serverless.amazonaws.com"
        }
      }
    },
    {
      "Sid": "EMRServerlessDashboardAccess",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:GetDashboardForJobRun"
      ],
      "Resource": [
        "arn:aws:emr-serverless*:123456789012:/applications/*"
      ]
    }
  ]
}
```

## 시작하기

Apache Livy 지원 애플리케이션을 생성하고 실행하려면 다음 단계를 수행합니다.

1. Apache Livy 지원 애플리케이션을 생성하려면 다음 명령을 실행합니다.

```
aws emr-serverless create-application \  
--name my-application-name \  
--type 'application-type' \  
--release-label <Amazon EMR-release-version>  
--interactive-configuration '{"livyEndpointEnabled": true}'
```

2. EMR Serverless에서 애플리케이션을 생성한 후 애플리케이션을 시작하여 Apache Livy 엔드포인트를 사용할 수 있도록 합니다.

```
aws emr-serverless start-application \  
--application-id application-id
```

다음 명령을 사용하여 애플리케이션의 상태를 확인합니다. STARTED 상태가 되면 Apache Livy 엔드포인트에 액세스합니다.

```
aws emr-serverless get-application \  
--region <AWS_REGION> --application-id >application_id>
```

3. 다음 URL을 사용하여 엔드포인트에 액세스합니다.

```
https://_<application-id>_.livy.emr-serverless-  
services._<AWS_REGION>_.amazonaws.com
```

엔드포인트가 준비되면 사용 사례에 따라 워크로드를 제출합니다. [SIGv4 프로토콜](#)을 사용하여 엔드포인트에 대한 모든 요청에 서명하고 권한 부여 헤더에서 전달해야 합니다. 다음 방법을 사용하여 워크로드를 실행할 수 있습니다.

- HTTP 클라이언트 - 사용자 지정 HTTP 클라이언트를 사용하여 Apache Livy 엔드포인트 API 작업을 제출합니다.
- Sparkmagic 커널 - sparkmagic 커널을 로컬로 실행하고 Jupyter Notebook에서 대화형 쿼리를 제출합니다.

## HTTP 클라이언트

Apache Livy 세션을 생성하려면 요청 본문의 conf 파라미터에서 `emr-serverless.session.executionRoleArn`을 제출합니다. 다음 예제는 POST `/sessions` 요청 샘플입니다.

```
{
  "kind": "pyspark",
  "heartbeatTimeoutInSeconds": 60,
  "conf": {
    "emr-serverless.session.executionRoleArn": "<executionRoleArn>"
  }
}
```

다음 표에서는 사용 가능한 모든 Apache Livy API 작업을 설명합니다.

API 작업	설명
GET <code>/sessions</code>	모든 활성 대화형 세션의 목록을 반환합니다.
POST <code>/sessions</code>	spark 또는 pyspark를 통해 새 대화형 세션을 생성합니다.
GET <code>/sessions/&lt;sessionId &gt;</code>	세션 정보를 반환합니다.
GET <code>/sessions/&lt;sessionId &gt;/state</code>	세션 상태를 반환합니다.
DELETE <code>/sessions/&lt;sessionId &gt;</code>	세션을 중지하고 삭제합니다.
GET <code>/sessions/&lt;sessionId &gt;/statements</code>	세션의 모든 명령문을 반환합니다.
POST <code>/sessions/&lt;sessionId &gt;/statements</code>	세션에서 명령문을 실행합니다.
GET <code>/sessions/&lt;sessionId &gt;/statements/&lt;statementId &gt;</code>	세션에서 지정된 명령문의 세부 정보를 반환합니다.
POST <code>/sessions/&lt;sessionId &gt;/statements/&lt;statementId &gt;/cancel</code>	이 세션에서 지정된 명령문을 취소합니다.

## Apache Livy 엔드포인트로 요청 전송

HTTP 클라이언트에서 Apache Livy 엔드포인트로 직접 요청을 전송할 수도 있습니다. 그러면 노트북 외부에서 사용 사례에 대한 코드를 원격으로 실행할 수 있습니다.

엔드포인트로 요청 전송을 시작하기 전에 다음 라이브러리를 설치했는지 확인합니다.

```
pip3 install boto3 awscli requests
```

다음은 HTTP 요청을 엔드포인트로 직접 전송하는 Python 스크립트 샘플입니다.

```
from boto3 import client
import requests
from boto3.awsrequest import AWSRequest
from boto3.credentials import Credentials
import boto3.session
import json, pprint, textwrap

endpoint = 'https://<application_id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com'
headers = {'Content-Type': 'application/json'}

session = boto3.session.Session()
signer = client.auth.CrtS3SigV4Auth(session.get_credentials(), 'emr-serverless',
    '<AWS_REGION>')

### Create session request

data = {'kind': 'pyspark', 'heartbeatTimeoutInSecond': 60, 'conf': { 'emr-
serverless.session.executionRoleArn': 'arn:aws:iam::123456789012:role/role1'}}

request = AWSRequest(method='POST', url=endpoint + "/sessions", data=json.dumps(data),
    headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r = requests.post(prepped.url, headers=prepped.headers, data=json.dumps(data))
```

```
pprint.pprint(r.json())

### List Sessions Request

request = AWSRequest(method='GET', url=endpoint + "/sessions", headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r2 = requests.get(prepped.url, headers=prepped.headers)
pprint.pprint(r2.json())

### Get session state

session_url = endpoint + r.headers['location']

request = AWSRequest(method='GET', url=session_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r3 = requests.get(prepped.url, headers=prepped.headers)

pprint.pprint(r3.json())

### Submit Statement

data = {
    'code': "1 + 1"
}

statements_url = endpoint + r.headers['location'] + "/statements"

request = AWSRequest(method='POST', url=statements_url, data=json.dumps(data),
    headers=headers)
```

```
request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r4 = requests.post(prepped.url, headers=prepped.headers, data=json.dumps(data))

pprint.pprint(r4.json())

### Check statements results

specific_statement_url = endpoint + r4.headers['location']

request = AWSRequest(method='GET', url=specific_statement_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r5 = requests.get(prepped.url, headers=prepped.headers)

pprint.pprint(r5.json())

### Delete session

session_url = endpoint + r.headers['location']

request = AWSRequest(method='DELETE', url=session_url, headers=headers)

request.context["payload_signing_enabled"] = False

signer.add_auth(request)

prepped = request.prepare()

r6 = requests.delete(prepped.url, headers=prepped.headers)
```

```
pprint.pprint(r6.json())
```

## Sparkmagic 커널

sparkmagic을 설치하기 전에 먼저 sparkmagic을 설치하려는 인스턴스에 AWS 자격 증명을 구성했는지 확인합니다.

1. [설치 단계](#)에 따라 sparkmagic을 설치합니다. 처음 4단계만 수행하면 됩니다.
2. sparkmagic 커널은 사용자 지정 인증자를 지원하므로, 모든 요청이 SIGv4로 서명되도록 인증자를 sparkmagic 커널과 통합할 수 있습니다.
3. EMR Serverless 사용자 지정 인증자를 설치합니다.

```
pip install emr-serverless-customauth
```

4. 이제 sparkmagic 구성 json 파일에 사용자 지정 인증자 및 Apache Livy 엔드포인트 URL의 경로를 제공합니다. 다음 명령을 사용하여 구성 파일을 엽니다.

```
vim ~/.sparkmagic/config.json
```

다음은 샘플 config.json 파일입니다.

```
{
  "kernel_python_credentials" : {
    "username": "",
    "password": "",
    "url": "https://<application-id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com",
    "auth": "Custom_Auth"
  },

  "kernel_scala_credentials" : {
    "username": "",
    "password": "",
    "url": "https://<application-id>.livy.emr-serverless-
services.<AWS_REGION>.amazonaws.com",
    "auth": "Custom_Auth"
  },

  "authenticators": {
    "None": "sparkmagic.auth.customauth.Authenticator",
    "Basic_Access": "sparkmagic.auth.basic.Basic",
```

```

    "Custom_Auth":
    "emr_serverless_customauth.customauthenticator.EMRServerlessCustomSigV4Signer"
  },
  "livy_session_startup_timeout_seconds": 600,
  "ignore_ssl_errors": false
}

```

5. Jupyter 랩을 시작합니다. 마지막 단계에서 설정한 사용자 지정 인증을 사용해야 합니다.
6. 그리고 다음 노트북 명령과 코드를 실행하여 시작할 수 있습니다.

```
%info //Returns the information about the current sessions.
```

```

%%configure -f //Configure information specific to a session. We supply
executionRoleArn in this example. Change it for your use case.
{
  "driverMemory": "4g",
  "conf": {
    "emr-serverless.session.executionRoleArn":
    "arn:aws:iam::123456789012:role/JobExecutionRole"
  }
}

```

```
<your code> //Run your code to start the session
```

내부적으로 각 명령은 구성된 Apache Livy 엔드포인트 URL을 통해 각 Apache Livy API 작업을 직접 호출합니다. 그런 다음, 사용 사례에 따라 지침을 작성할 수 있습니다.

## 고려 사항

Apache Livy 엔드포인트를 통해 대화형 워크로드를 실행하는 경우 다음 사항을 고려합니다.

- EMR Serverless는 직접 호출자 위탁자를 사용하여 세션 수준 격리를 유지 관리합니다. 세션을 생성하는 직접 호출자 위탁자는 해당 세션에 액세스할 수 있는 유일한 위탁자입니다. 보다 세분화된 격리를 위해 자격 증명을 가정할 때 소스 자격 증명을 구성할 수 있습니다. 이 경우 EMR Serverless는 직접 호출자 위탁자와 소스 자격 증명을 기반으로 세션 수준 격리를 적용합니다. 소스 자격 증명에 대한 자세한 내용은 [위임된 역할로 수행한 작업 모니터링 및 제어](#)를 참고하세요.
- Apache Livy 엔드포인트는 EMR Serverless 릴리스 6.14.0 이상에서 지원됩니다.
- Apache Livy 엔드포인트는 Apache Spark 엔진에서만 지원됩니다.

- Apache Livy 엔드포인트는 Scala Spark 및 PySpark를 지원합니다.
- 기본적으로 autoStopConfig는 애플리케이션에서 활성화됩니다. 즉, 15분의 유휴 상태 이후에 애플리케이션이 종료됩니다. create-application 또는 update-application 요청의 일부로 이 구성을 변경할 수 있습니다.
- 단일 Apache Livy 엔드포인트 지원 애플리케이션에서 최대 25개의 동시 세션을 실행할 수 있습니다.
- 최상의 시작 경험을 위해 드라이버 및 실행기에 대해 사전 초기화된 용량을 구성하는 것이 좋습니다.
- Apache Livy 엔드포인트에 연결하기 전에 애플리케이션을 수동으로 시작해야 합니다.
- Apache Livy 엔드포인트로 대화형 워크로드를 실행하려면 AWS 계정에서 vCPU 서비스 할당량이 충분해야 합니다. 최소 24개의 vCPU를 권장합니다.
- 기본 Apache Livy 세션의 제한 시간은 1시간입니다. 명령문을 1시간 동안 실행하지 않으면 Apache Livy는 세션을 삭제하고 드라이버 및 실행기를 해제합니다. 릴리스 emr-7.8.0부터는 2h(시간), 120m(분), 7200s(초), 7200000ms(밀리초)와 같이 ttl 파라미터를 Livy /sessions POST 요청의 일부로 지정하여 이 값을 설정할 수 있습니다.

#### Note

emr-7.8.0 이전의 경우 이 구성을 변경할 수 없습니다. 다음은 POST /sessions 요청 본문입니다.

```
{
  "kind": "pyspark",
  "heartbeatTimeoutInSeconds": 60,
  "conf": {
    "emr-serverless.session.executionRoleArn": "executionRoleArn"
  },
  "ttl": "2h"
}
```

- Amazon EMR 릴리스 emr-7.8.0부터 Lake Formation을 통한 세분화된 접근 제어 기능을 지원하는 애플리케이션의 경우, 세션별로 해당 설정을 비활성화할 수 있습니다. EMR Serverless 애플리케이션에 대한 세분화된 액세스 제어를 활성화하는 방법에 대한 자세한 내용은 [세분화된 액세스 제어 방법](#)을 참조하세요.

 Note

애플리케이션에 대해 Lake Formation이 활성화되지 않은 경우 세션에 대해 활성화할 수 없습니다. 다음은 POST /sessions 요청 본문입니다.

```
{
  "kind": "pyspark",
  "heartbeatTimeoutInSeconds": 60,
  "conf": {
    "emr-serverless.session.executionRoleArn": "executionRoleArn"
  },
  "spark.emr-serverless.lakeformation.enabled" : "false"
}
```

- 활성 세션만 Apache Livy 엔드포인트와 상호 작용할 수 있습니다. 세션이 완료, 취소 또는 종료되면 Apache Livy 엔드포인트를 통해 액세스할 수 없습니다.

# 로깅 및 모니터링

모니터링은 EMR Serverless 애플리케이션 및 작업의 신뢰성, 가용성 및 성능을 유지 관리하는 중요한 역할을 합니다. EMR Serverless 솔루션의 모든 부분에서 모니터링 데이터를 수집하여 다중 지점 장애가 발생할 경우 더 쉽게 디버깅할 수 있도록 해야 합니다.

주제

- [로그 저장](#)
- [로그 교체](#)
- [로그 암호화](#)
- [Amazon EMR Serverless에 대한 Apache Log4j2 속성 구성](#)
- [EMR Serverless 모니터링](#)
- [Amazon EventBridge를 사용하여 EMR Serverless 자동화](#)

## 로그 저장

EMR Serverless에서 작업 진행 상황을 모니터링하고 작업 실패 문제를 해결하기 위해 EMR Serverless에서 애플리케이션 로그를 저장하고 제공하는 방법을 선택합니다. 작업 실행을 제출하는 경우 관리형 스토리지, Amazon S3 및 Amazon CloudWatch를 로깅 옵션으로 지정합니다.

CloudWatch를 사용하면 사용하려는 로그 유형 및 로그 위치를 지정하거나 기본 유형 및 위치를 수락합니다. Cloudwatch 로그에 대한 자세한 내용은 [the section called “Amazon CloudWatch”](#) 섹션을 참조하세요. 다음 표에서는 관리형 스토리지 및 S3 로깅을 사용하는 경우 [관리형 스토리지](#), [Amazon S3 버킷](#) 또는 둘 다를 선택할 때 예상할 수 있는 로그 위치와 UI 가용성을 나열합니다.

옵션	이벤트 로그	컨테이너 로그	애플리케이션 UI
관리형 스토리지	관리형 스토리지에 저장됨	관리형 스토리지에 저장됨	지원
관리형 스토리지와 S3 버킷 모두	두 장소에 모두 저장됨	S3 버킷에 저장됨	지원
Amazon S3 버킷	S3 버킷에 저장됨	S3 버킷에 저장됨	지원되지 않음 <sup>1</sup>

<sup>1</sup> 관리형 스토리지 옵션을 선택한 상태로 유지하는 것이 좋습니다. 그렇지 않으면 기본 제공 애플리케이션 UI를 사용할 수 없습니다.

## 관리형 스토리지를 사용하는 EMR Serverless에 대한 로깅

기본적으로 EMR Serverless는 애플리케이션 로그를 Amazon EMR 관리형 스토리지에 최대 30일 동안 안전하게 저장합니다.

### Note

기본 옵션을 끄면 Amazon EMR에서 사용자를 대신하여 작업 문제를 해결할 수 없습니다. 예: EMR Serverless Console에서 Spark-UI에 액세스할 수 없습니다.

EMR Studio에서 이 옵션을 끄려면 제출 작업 페이지의 추가 설정 섹션에서 30일 동안 AWS에서 로그 유지 허용 확인란을 선택 취소합니다.

AWS CLI에서 이 옵션을 끄려면 작업 실행을 제출하는 경우 `managedPersistenceMonitoringConfiguration` 구성을 사용합니다.

```
{
  "monitoringConfiguration": {
    "managedPersistenceMonitoringConfiguration": {
      "enabled": false
    }
  }
}
```

EMR Serverless 애플리케이션이 Amazon S3용 VPC 엔드포인트가 있는 프라이빗 서브넷에 있고 엔드포인트 정책을 연결하여 액세스를 제어하는 경우 EMR Serverless가 애플리케이션 로그를 저장하고 제공할 수 있도록 다음 권한을 추가합니다. [Amazon S3에 액세스하는 프라이빗 서브넷에 대한 샘플 정책](#)의 사용 가능한 지역 테이블에서 Resource를 AppInfo 버킷으로 바꿉니다.

JSON

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

```

    "Effect": "Allow",
    "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
    ],
    "Resource": [
        "arn:aws:s3:::prod.us-east-1.appinfo.src",
        "arn:aws:s3:::prod.us-east-1.appinfo.src/*"
    ],
    "Condition": {
        "StringEquals": {
            "aws:PrincipalServiceName": "emr-serverless.amazonaws.com",
            "aws:SourceVpc": "vpc-12345678"
        }
    }
}

```

또한 `aws:SourceVpc` 조건 키를 사용하여 요청이 VPC 엔드포인트가 연결된 VPC를 통해 전달되도록 합니다.

## Amazon S3 버킷을 사용하는 EMR Serverless에 대한 로깅

작업에서 Amazon S3로 로그 데이터를 전송하려면 먼저 작업 실행 역할에 대한 권한 정책에 다음 권한을 포함시킵니다. *amzn-s3-demo-logging-bucket*을 로깅 버킷의 이름으로 바꿉니다.

JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ],
      "Sid": "AllowS3Putobject"
    }
  ]
}

```

```
    }
  ]
}
```

AWS CLI에서 로그를 저장하도록 Amazon S3 버킷을 설정하려면 작업 실행을 시작할 때 `s3MonitoringConfiguration` 구성을 사용합니다. 이를 수행하려면 구성에서 다음 `--configuration-overrides`를 제공합니다.

```
{
  "monitoringConfiguration": {
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket/logs/"
    }
  }
}
```

재시도를 활성화하지 않은 배치 작업의 경우 EMR Serverless는 로그를 다음 경로로 전송합니다.

```
'/applications/<applicationId>/jobs/<jobId>'
```

EMR Serverless는 Spark 드라이버 로그를 다음 경로에 저장합니다.

```
'/applications/<applicationId>/jobs/<jobId>/SPARK_DRIVER/'
```

EMR Serverless는 Spark 실행기 로그를 다음 경로에 저장합니다.

```
'/applications/<applicationId>/jobs/<jobId>/SPARK_EXECUTOR/<EXECUTOR-ID>'
```

<EXECUTOR-ID>는 정수입니다.

EMR Serverless 릴리스 7.1.0 이상에서는 스트리밍 작업 및 배치 작업에 대한 재시도를 지원합니다. 재시도가 활성화된 상태에서 작업을 실행하면 EMR Serverless는 로그 경로 접두사에 시도 번호를 자동으로 추가하므로, 로그를 효과적으로 식별하고 추적할 수 있습니다.

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'
```

## Amazon CloudWatch를 사용하는 EMR Serverless에 대한 로깅

EMR Serverless 애플리케이션에 작업을 제출하는 경우 Amazon CloudWatch를 애플리케이션 로그를 저장하는 옵션으로 선택합니다. 이를 통해 CloudWatch Logs Insights 및 Live Tail과 같은 CloudWatch 로그 분석 기능을 사용할 수 있습니다. 추가 분석을 위해 CloudWatch에서 OpenSearch와 같은 다른 시스템으로 로그를 스트리밍할 수도 있습니다.

EMR Serverless는 드라이버 로그에 대한 실시간 로깅을 제공합니다. CloudWatch Live Tail 기능을 사용하거나 CloudWatch CLI 테일 명령을 통해 로그에 실시간으로 액세스할 수 있습니다.

기본적으로 EMR Serverless에서는 CloudWatch 로깅이 비활성화됩니다. 이를 활성화하려면 [AWS CLI](#)의 구성을 사용합니다.

### Note

Amazon CloudWatch는 로그를 실시간으로 게시하므로 작업자에서 더 많은 리소스가 사용 됩니다. 더 적은 작업자 용량을 선택하면 작업 런타임에 미치는 영향이 커질 수 있습니다. CloudWatch 로깅을 활성화하는 경우 더 큰 워커 용량을 선택하는 것이 좋습니다. 초당 트랜잭션 수(TPS) 비율이 PutLogEvents에서 너무 낮으면 로그 게시가 스로틀링될 수도 있습니다. CloudWatch 스로틀링 구성은 EMR Serverless를 포함한 모든 서비스에 대해 전역으로 적용됩니다. 자세한 내용은 AWS re:post의 [CloudWatch Logs의 스로틀링 오류를 해결하려면 어떻게 해야 합니까?](#)를 참조하세요.

## CloudWatch를 사용하여 로깅하는 데 필요한 권한

작업에서 Amazon CloudWatch로 로그 데이터를 전송하려면 먼저 작업 실행 역할에 대한 권한 정책에 다음 권한을 포함시킵니다.

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:DescribeLogGroups"
      ],
    }
  ],
}
```

```

    "Resource": [
      "arn:aws:logs:*:123456789012:*"
    ],
    "Sid": "AllowLOGSDescribeLoggroups"
  },
  {
    "Effect": "Allow",
    "Action": [
      "logs:PutLogEvents",
      "logs:CreateLogGroup",
      "logs:CreateLogStream",
      "logs:DescribeLogStreams"
    ],
    "Resource": [
      "arn:aws:logs:*:123456789012:log-group:my-log-group-name:*"
    ],
    "Sid": "AllowLOGSPutlogevents"
  }
]
}

```

## AWS CLI

AWS CLI에서 EMR Serverless에 대한 로그를 저장하도록 Amazon CloudWatch를 설정하려면 작업 실행을 시작할 때 `cloudWatchLoggingConfiguration` 구성을 사용합니다. 이를 수행하려면 다음 구성 재정의의 제공합니다. 선택적으로 로그 그룹 이름, 로그 스트림 접두사 이름, 로그 유형 및 암호화 키 ARN도 제공합니다.

선택적 값을 지정하지 않으면 CloudWatch는 기본 로그 스트림 `/applications/applicationId/jobs/jobId/worker-type`을 사용하여 로그를 기본 로그 그룹 `/aws/emr-serverless`에 게시합니다.

EMR Serverless 릴리스 7.1.0 이상에서는 스트리밍 작업 및 배치 작업에 대한 재시도를 지원합니다. 작업에 대한 재시도를 활성화하면 EMR Serverless는 로그 경로 접두사에 시도 번호를 자동으로 추가하므로, 로그를 효과적으로 식별하고 추적할 수 있습니다.

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/worker-type'
```

다음에서는 EMR Serverless의 기본 설정으로 Amazon CloudWatch 로깅을 활성화하는 데 필요한 최소 구성을 보여줍니다.

```
{
  "monitoringConfiguration": {
    "cloudWatchLoggingConfiguration": {
      "enabled": true
    }
  }
}
```

다음 예제에서는 EMR Serverless에 대해 Amazon CloudWatch 로깅을 켤 때 지정하는 모든 필수 및 선택적 구성을 보여줍니다. 지원되는 logTypes 값은 다음 예제에 나열되어 있습니다.

```
{
  "monitoringConfiguration": {
    "cloudWatchLoggingConfiguration": {
      "enabled": true, // Required
      "logGroupName": "Example_logGroup", // Optional
      "logStreamNamePrefix": "Example_logStream", // Optional
      "encryptionKeyArn": "key-arn", // Optional
      "logTypes": {
        "SPARK_DRIVER": ["stdout", "stderr"] //List of values
      }
    }
  }
}
```

기본적으로 EMR Serverless는 드라이버 stdout 및 stderr 로그만 CloudWatch에 게시합니다. 다른 로그를 원하는 경우 logTypes 필드를 사용하여 컨테이너 역할 및 해당 로그 유형을 지정합니다.

다음 목록에서는 logTypes 구성에 지정할 수 있는 지원되는 워커 유형을 보여줍니다.

#### Spark

- SPARK\_DRIVER : ["STDERR", "STDOUT"]
- SPARK\_EXECUTOR : ["STDERR", "STDOUT"]

#### Hive

- HIVE\_DRIVER : ["STDERR", "STDOUT", "HIVE\_LOG", "TEZ\_AM"]
- TEZ\_TASK : ["STDERR", "STDOUT", "SYSTEM\_LOGS"]

## 로그 교체

Amazon EMR Serverless는 Spark 애플리케이션 로그 및 이벤트 로그를 교체할 수 있습니다. 로그 교체는 모든 디스크 공간을 차지할 수 있는 대용량 로그 파일을 생성하는 장기 실행 작업의 문제를 해결하는 데 도움이 됩니다. 로그를 교체하면 디스크 스토리지를 절약하고 디스크에 남은 추가 공간이 없어서 실패하는 작업 수를 줄일 수 있습니다.

로그 교체는 기본적으로 활성화되어 있으며, Spark 작업에만 사용할 수 있습니다.

### Spark 이벤트 로그

#### Note

Spark 이벤트 로그 교체는 모든 Amazon EMR 릴리스 레이블에서 사용할 수 있습니다.

EMR Serverless는 단일 이벤트 로그 파일을 생성하는 대신, 이벤트 로그를 정기적으로 교체하고 이전 이벤트 로그 파일을 제거합니다. 로그 교체는 S3 버킷에 업로드된 로그에 영향을 주지 않습니다.

### Spark 애플리케이션 로그

#### Note

Spark 애플리케이션 로그 교체는 모든 Amazon EMR 릴리스 레이블에서 사용할 수 있습니다.

또한 EMR Serverless는 stdout 및 stderr 파일과 같은 드라이버 및 실행기에 대한 Spark 애플리케이션 로그도 교체합니다. Spark 기록 서버 및 Live UI 링크를 사용하여 Studio에서 로그 링크를 선택해 최신 로그 파일에 액세스할 수 있습니다. 로그 파일은 최신 로그의 잘린 버전입니다. 이전의 교체된 로그를 보려면 로그를 저장할 때 Amazon S3 위치를 지정합니다. 자세한 내용은 [Amazon S3 버킷을 사용하는 EMR Serverless에 대한 로깅](#)을 참조하세요.

다음 위치에서 최신 로그 파일을 찾을 수 있습니다. EMR Serverless는 15초마다 파일을 새로 고칩니다. 이러한 파일의 범위는 0MB~128MB입니다.

```
<example-S3-logUri>/applications/<application-id>/jobs/<job-id>/SPARK_DRIVER/stderr.gz
```

다음 위치에는 이전의 교체된 파일이 포함되어 있습니다. 각 파일은 128MB입니다.

```
<example-S3-logUri>/applications/<application-id>/jobs/<job-id>/SPARK_DRIVER/archived/
stderr_<index>.gz
```

동일한 동작이 Spark 실행기에도 적용됩니다. 이 변경 사항은 S3 로깅에만 적용됩니다. 로그 교체 시 Amazon CloudWatch에 업로드된 로그 스트림에 변경 사항을 도입하지 않습니다.

EMR Serverless 릴리스 7.1.0 이상에서는 스트리밍 및 배치 작업에 대한 재시도를 지원합니다. 작업에서 재시도를 활성화한 경우 EMR Serverless는 해당 작업의 로그 경로에 접두사를 추가하므로 로그를 효과적으로 추적하고 다른 로그와 구분할 수 있습니다. 이 경로에는 교체된 모든 로그가 포함됩니다.

```
'/applications/<applicationId>/jobs/<jobId>/attempts/<attemptNumber>/'.
```

## 로그 암호화

### 관리형 스토리지를 사용하는 EMR Serverless 로그 암호화

자체 KMS 키를 사용하여 관리형 스토리지에서 로그를 암호화하려면 작업 실행을 제출할 때 managedPersistenceMonitoringConfiguration 구성을 사용합니다.

```
{
  "monitoringConfiguration": {
    "managedPersistenceMonitoringConfiguration" : {
      "encryptionKeyArn": "key-arn"
    }
  }
}
```

### Amazon S3 버킷을 사용하는 EMR Serverless 로그 암호화

자체 KMS 키를 사용하여 Amazon S3 버킷에서 로그를 암호화하려면 작업 실행을 제출할 때 s3MonitoringConfiguration 구성을 사용합니다.

```
{
  "monitoringConfiguration": {
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-logging-bucket/logs/",
      "encryptionKeyArn": "key-arn"
    }
  }
}
```

```
}
}
```

## Amazon CloudWatch를 사용하는 EMR Serverless 로그 암호화

자체 KMS 키로 Amazon CloudWatch에서 로그를 암호화하려면 작업 실행을 제출할 때 `cloudWatchLoggingConfiguration` 구성을 사용합니다.

```
{
  "monitoringConfiguration": {
    "cloudWatchLoggingConfiguration": {
      "enabled": true,
      "encryptionKeyArn": "key-arn"
    }
  }
}
```

## 로그 암호화에 필요한 권한

이 섹션의 내용

- [필요한 사용자 권한](#)
- [Amazon S3 및 관리형 스토리지에 대한 암호화 키 권한](#)
- [Amazon CloudWatch에 대한 암호화 키 권한](#)

## 필요한 사용자 권한

작업을 제출하거나 로그 또는 애플리케이션 UI를 보는 사용자는 키를 사용할 권한이 있어야 합니다. 사용자, 그룹 또는 역할에 대한 IAM 정책 또는 KMS 키 정책에서 권한을 지정할 수 있습니다. 작업을 제출하는 사용자에게 KMS 키 권한이 없는 경우 EMR Serverless는 작업 실행 제출을 거부합니다.

## 예제 키 정책

다음 키 정책에서는 `kms:GenerateDataKey` 및 `kms:Decrypt`에 대한 권한을 제공합니다.

```
{
  "Effect": "Allow",
  "Principal": {
    "AWS": "arn:aws:iam::111122223333:user/user-name"
  },
}
```

```

    "Action": [
      "kms:GenerateDataKey",
      "kms:Decrypt"
    ],
    "Resource": "*"
  }

```

## IAM 정책 예제

다음 IAM 정책은 kms:GenerateDataKey 및 kms:Decrypt에 대한 권한을 제공합니다.

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:*:123456789012:key/12345678-1234-1234-1234-123456789012"
      ],
      "Sid": "AllowKMSGeneratedatakey"
    }
  ]
}

```

Spark 또는 Tez UI를 시작하려면 사용자, 그룹 또는 역할에 다음과 같이 `emr-serverless:GetDashboardForJobRun` API에 액세스할 수 있는 권한을 부여합니다.

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",

```

```

    "Action": [
      "emr-serverless:GetDashboardForJobRun"
    ],
    "Resource": [
      "*"
    ],
    "Sid": "AllowEMRSERVERLESSGetdashboardforjobrun"
  }
]
}

```

## Amazon S3 및 관리형 스토리지에 대한 암호화 키 권한

관리형 스토리지 또는 S3 버킷에서 자체 암호화 키로 로그를 암호화하는 경우 다음과 같이 KMS 키 권한을 구성합니다.

emr-serverless.amazonaws.com 위탁자는 KMS 키에 대한 정책에 다음 권한이 있어야 합니다.

```

{
  "Effect": "Allow",
  "Principal": {
    "Service": "emr-serverless.amazonaws.com"
  },
  "Action": [
    "kms:Decrypt",
    "kms:GenerateDataKey"
  ],
  "Resource": "*"
  "Condition": {
    "StringLike": {
      "aws:SourceArn": "arn:aws:emr-serverless:region:aws-account-id:/
applications/application-id"
    }
  }
}

```

보안 모범 사례로 aws:SourceArn 조건 키를 KMS 키 정책에 추가하는 것이 좋습니다. IAM 전역 조건 키 aws:SourceArn은 EMR Serverless가 애플리케이션 ARN에 대해서만 KMS 키를 사용하도록 하는데 도움이 됩니다.

작업 런타임 역할에는 IAM 정책에 다음 권한이 있어야 합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:GenerateDataKey",
        "kms:Decrypt"
      ],
      "Resource": [
        "arn:aws:kms:*:123456789012:key/12345678-1234-1234-1234-123456789012"
      ],
      "Sid": "AllowKMSGeneratedatakey"
    }
  ]
}
```

## Amazon CloudWatch에 대한 암호화 키 권한

KMS 키 ARN을 로그 그룹에 연결하려면 작업 런타임 역할에 대해 다음 IAM 정책을 사용합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:AssociateKmsKey"
      ],
      "Resource": [
        "arn:aws:logs:*:123456789012:log-group:my-log-group-name:*"
      ],
      "Sid": "AllowLOGSAssociatekmskey"
    }
  ]
}
```

Amazon CloudWatch에 KMS 권한을 부여하도록 KMS 키 정책을 구성합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Id": "key-default-1",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:GenerateDataKey"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "ArnLike": {
          "kms:EncryptionContext:aws:logs:arn": "arn:aws:logs:*:123456789012:*"
        }
      },
      "Sid": "AllowKMSDecrypt"
    }
  ]
}
```

## Amazon EMR Serverless에 대한 Apache Log4j2 속성 구성

이 페이지에서는 StartJobRun에서 EMR Serverless 작업에 대한 사용자 지정 [Apache Log4j 2.x](#) 속성을 구성하는 방법을 설명합니다. 애플리케이션 수준에서 Log4j 분류를 구성하려면 [EMR Serverless에 대한 기본 애플리케이션 구성](#) 섹션을 참조하세요.

## Amazon EMR Serverless에 대한 Spark Log4j2 속성 구성

Amazon EMR 릴리스 6.8.0 이상에서는 [Apache Log4j 2.x](#) 속성을 사용자 지정하여 세분화된 로그 구성을 지정할 수 있습니다. 그러면 EMR Serverless에서 Spark 작업의 문제 해결이 간소화됩니다. 이러한 속성을 구성하려면 spark-driver-log4j2 및 spark-executor-log4j2 분류를 사용합니다.

주제

- [Spark에 대한 Log4j2 분류](#)
- [Spark에 대한 Log4j2 구성 예제](#)
- [샘플 Spark 작업의 Log4j2](#)
- [Spark에 대한 Log4j2 고려 사항](#)

## Spark에 대한 Log4j2 분류

Spark 로그 구성을 사용자 지정하려면 [applicationConfiguration](#)에서 다음 분류를 사용합니다. Log4j 2.x 속성을 구성하려면 다음 [properties](#)를 사용합니다.

### **spark-driver-log4j2**

이 분류는 드라이버에 대한 log4j2.properties 파일의 값을 설정합니다.

### **spark-executor-log4j2**

이 분류는 실행기에 대한 log4j2.properties 파일의 값을 설정합니다.

## Spark에 대한 Log4j2 구성 예제

다음 예제에서는 Spark 드라이버 및 실행기에 대한 Log4j2 구성을 사용자 지정하기 위해 applicationConfiguration을 사용해 Spark 작업을 제출하는 방법을 보여줍니다.

작업을 제출할 때 대신 애플리케이션 수준에서 Log4j 분류를 구성하려면 [EMR Serverless에 대한 기본 애플리케이션 구성](#) 섹션을 참조하세요.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "/usr/lib/spark/examples/jars/spark-examples.jar",
      "entryPointArguments": ["1"],
      "sparkSubmitParameters": "--class org.apache.spark.examples.SparkPi --conf
spark.executor.cores=4 --conf spark.executor.memory=20g --conf spark.driver.cores=4 --
conf spark.driver.memory=8g --conf spark.executor.instances=1"
    }
  }'
  --configuration-overrides '{
```

```

    "applicationConfiguration": [
      {
        "classification": "spark-driver-log4j2",
        "properties": {
          "rootLogger.level": "error", // will only display Spark error logs
          "logger.IdentifierForClass.name": "classpath for setting logger",
          "logger.IdentifierForClass.level": "info"
        }
      },
      {
        "classification": "spark-executor-log4j2",
        "properties": {
          "rootLogger.level": "error", // will only display Spark error logs
          "logger.IdentifierForClass.name": "classpath for setting logger",
          "logger.IdentifierForClass.level": "info"
        }
      }
    ]
  }
}'

```

## 샘플 Spark 작업의 Log4j2

다음 코드 샘플에서는 애플리케이션에 대한 사용자 지정 Log4j2 구성을 초기화하는 동안 Spark 애플리케이션을 생성하는 방법을 보여줍니다.

### Python

#### Example - Python에서 Spark 작업에 Log4j2 사용

```

import os
import sys

from pyspark import SparkConf, SparkContext
from pyspark.sql import SparkSession

app_name = "PySparkApp"
if __name__ == "__main__":
    spark = SparkSession\
        .builder\
        .appName(app_name)\
        .getOrCreate()

```

```

spark.sparkContext._conf.getAll()
sc = spark.sparkContext
log4jLogger = sc._jvm.org.apache.log4j
LOGGER = log4jLogger.LogManager.getLogger(app_name)

LOGGER.info("pyspark script logger info")
LOGGER.warn("pyspark script logger warn")
LOGGER.error("pyspark script logger error")

// your code here

spark.stop()

```

Spark 작업을 실행하는 경우 드라이버에 대해 Log4j2를 사용자 지정하려면 다음 구성을 사용할 수 있습니다.

```

{
  "classification": "spark-driver-log4j2",
  "properties": {
    "rootLogger.level": "error", // only display Spark error logs
    "logger.PySparkApp.level": "info",
    "logger.PySparkApp.name": "PySparkApp"
  }
}

```

## Scala

### Example - Scala에서 Spark 작업에 Log4j2 사용

```

import org.apache.log4j.Logger
import org.apache.spark.sql.SparkSession

object ExampleClass {
  def main(args: Array[String]): Unit = {
    val spark = SparkSession
      .builder
      .appName(this.getClass.getName)
      .getOrCreate()

    val logger = Logger.getLogger(this.getClass);
    logger.info("script logging info logs")
    logger.warn("script logging warn logs")
  }
}

```

```

    logger.error("script logging error logs")

    // your code here
    spark.stop()
  }
}

```

Spark 작업을 실행하는 경우 드라이버에 대해 Log4j2를 사용자 지정하려면 다음 구성을 사용할 수 있습니다.

```

{
  "classification": "spark-driver-log4j2",
  "properties": {
    "rootLogger.level": "error", // only display Spark error logs
    "logger.ExampleClass.level": "info",
    "logger.ExampleClass.name": "ExampleClass"
  }
}

```

## Spark에 대한 Log4j2 고려 사항

다음 Log4j2.x 속성은 Spark 프로세스에 대해 구성할 수 없습니다.

- `rootLogger.appenderRef.stdout.ref`
- `appender.console.type`
- `appender.console.name`
- `appender.console.target`
- `appender.console.layout.type`
- `appender.console.layout.pattern`

구성할 수 있는 Log4j2.x 속성에 대한 자세한 내용은 GitHub의 [log4j2.properties.template 파일](#)을 참조하세요.

## EMR Serverless 모니터링

이 섹션에서는 Amazon EMR Serverless 애플리케이션 및 작업을 모니터링하는 방법을 다룹니다.

## 주제

- [EMR Serverless 애플리케이션 및 작업 모니터링](#)
- [Amazon Managed Service for Prometheus를 사용하여 Spark 지표 모니터링](#)
- [EMR Serverless 사용 지표](#)

## EMR Serverless 애플리케이션 및 작업 모니터링

EMR Serverless에 대한 Amazon CloudWatch 지표를 사용하면 1분 CloudWatch 지표를 수신하고 CloudWatch 대시보드에 액세스하여 거의 실시간에 가까운 EMR Serverless 애플리케이션의 작업 및 성능에 액세스할 수 있습니다.

EMR Serverless는 매분 CloudWatch로 지표를 전송합니다. EMR Serverless는 애플리케이션 수준과 작업, 작업자 유형 및 용량 할당 유형 수준에서 이러한 지표를 내보냅니다.

시작하려면 [EMR Serverless GitHub 리포지토리](#)에서 제공하는 EMR Serverless CloudWatch 대시보드 템플릿을 사용하고 배포합니다.

### Note

[EMR Serverless 대화형 워크로드](#)에서는 애플리케이션 수준 모니터링만 활성화되며, 새로운 작업자 유형 차원(Spark\_Kernel)을 포함합니다. 대화형 워크로드를 모니터링 및 디버깅하기 위해 [EMR Studio Workspace](#) 내에서 로그 및 Apache Spark UI에 액세스할 수 있습니다.

## 지표 모니터링

### Important

지표 표시를 재구성하여 ApplicationName 및 JobName을 차원으로 추가합니다. 릴리스 7.10 이상에서는 이전 지표가 더 이상 업데이트되지 않습니다. EMR 릴리스 7.10 이하의 경우 이전 지표를 계속 사용할 수 있습니다.

## 현재 차원

아래 표에서는 AWS/EMR Serverless 네임스페이스 내에서 사용할 수 있는 EMR Serverless 차원을 설명합니다.

## EMR Serverless 지표의 차원

차원	설명	
ApplicationId	애플리케이션 ID를 사용하여 EMR Serverless 애플리케이션의 모든 지표를 필터링합니다.	
ApplicationName	이름을 사용하여 EMR Serverless 애플리케이션의 모든 지표를 필터링합니다. 이름이 지정되지 않거나 ASCII가 아닌 문자가 포함된 경우 [지정되지 않음]으로 게시됩니다.	
JobId	EMR Serverless의 모든 메트릭을 작업 실행 ID로 필터링합니다.	
JobName	이름을 사용하여 실행되는 EMR Serverless 작업의 모든 메트릭에 대한 필터입니다. 이름이 지정되지 않거나 ASCII가 아닌 문자가 포함된 경우 [지정되지 않음]으로 게시됩니다.	
WorkerType	주어진 작업자 유형의 모든 지표를 필터링합니다. 예를 들어, Spark 작업에 대해 SPARK_DRIVER 및 SPARK_EXECUTORS 를 필터링할 수 있습니다.	
CapacityAllocation Type	주어진 용량 할당 유형의 모든 지표를 필터링합니다. 예를 들어, 사전 초기화된 용량에 대해서는 PreInitCapacity , 이와 모든 항목에 대해서는	

차원	설명	
	OnDemandCapacity 를 필터링할 수 있습니다.	

## 애플리케이션 수준 모니터링

Amazon CloudWatch 지표를 사용하여 EMR Serverless 애플리케이션 수준에서 용량 사용량을 모니터링할 수 있습니다. 또한 CloudWatch 대시보드에서 애플리케이션 용량 사용량을 모니터링하도록 단일 보기를 설정할 수도 있습니다.

### EMR Serverless 애플리케이션 지표

지표	설명	단위	차원
MaxCPUAllowed	애플리케이션에 대해 허용되는 최대 CPU.	vCPU	ApplicationId , ApplicationName
MaxMemoryAllowed	애플리케이션에 대해 허용되는 GB 단위의 최대 메모리.	기가바이트(GB)	ApplicationId , ApplicationName
MaxStorageAllowed	애플리케이션에 대해 허용되는 GB 단위의 최대 스토리지.	기가바이트(GB)	ApplicationId , ApplicationName
CPUAllocated	할당된 총 vCPU 수.	vCPU	ApplicationId , ApplicationName , WorkerType , CapacityAllocationType
IdleWorkerCount	유휴 상태인 총 작업자 수.	개수	ApplicationId , ApplicationName , WorkerType , CapacityAllocationType

지표	설명	단위	차원
MemoryAllocated	할당된 GB 단위의 총 메모리.	기가바이트(GB)	ApplicationId , ApplicationName , WorkerType ,CapacityAllocationType
PendingCreationWorkerCount	생성 보류 중인 총 작업자 수.	개수	ApplicationId , ApplicationName , WorkerType ,CapacityAllocationType
RunningWorkerCount	애플리케이션에서 사용 중인 총 작업자 수.	개수	ApplicationId , ApplicationName , WorkerType ,CapacityAllocationType
StorageAllocated	할당된 GB 단위의 총 디스크 스토리지.	기가바이트(GB)	ApplicationId , ApplicationName , WorkerType ,CapacityAllocationType
TotalWorkerCount	사용 가능한 총 작업자 수.	개수	ApplicationId , ApplicationName , WorkerType ,CapacityAllocationType

## 작업 수준 모니터링

Amazon EMR Serverless는 1분마다 Amazon CloudWatch로 다음과 같은 작업 수준 지표를 전송합니다. 작업 실행 상태별로 집계 작업 실행의 지표 값에 액세스할 수 있습니다. 각 지표의 단위는 개수입니다.

## EMR Serverless 작업 수준 지표

지표	설명	차원
SubmittedJobs	제출됨 상태의 최대 작업 수.	ApplicationId , ApplicationName
PendingJobs	보류 중 상태의 작업 수.	ApplicationId , ApplicationName
ScheduledJobs	예약된 상태의 작업 수.	ApplicationId , ApplicationName
RunningJobs	실행 중 상태의 작업 수.	ApplicationId , ApplicationName
SuccessJobs	성공 상태의 작업 수.	ApplicationId , ApplicationName
FailedJobs	실패 상태의 작업 수.	ApplicationId , ApplicationName
CancellingJobs	취소 중 상태의 작업 수.	ApplicationId , ApplicationName
CancelledJobs	취소됨 상태의 작업 수.	ApplicationId , ApplicationName

엔진별 애플리케이션 UI를 사용하여 EMR Serverless의 실행 중인 작업 및 완료된 작업에 대한 엔진별 지표를 모니터링할 수 있습니다. 실행 중인 작업의 UI에 접근하면 실시간 업데이트가 적용된 라이브 애플리케이션 UI가 표시됩니다. 완료된 작업의 UI에 접근하면 영구 앱 UI가 표시됩니다.

## 작업 실행

EMR Serverless의 실행 중인 작업에 대해서는 엔진별 지표를 제공하는 실시간 인터페이스에 액세스할 수 있습니다. Apache Spark UI 또는 Hive Tez UI를 사용하여 작업을 모니터링하고 디버깅할 수 있습니다. 이러한 UI에 액세스하려면 EMR Studio 콘솔을 사용하거나 AWS Command Line Interface를 사용하여 보안 URL 엔드포인트를 요청합니다.

## 작업 완료됨

완료된 EMR Serverless 작업의 경우 Spark 기록 서버 또는 영구 Hive Tez UI를 사용하여 Spark 또는 Hive 작업 실행에 대한 작업 세부 정보, 단계, 태스크 및 지표에 액세스할 수 있습니다. 이러한 UI에 액세스하려면 EMR Studio 콘솔을 사용하거나 AWS Command Line Interface를 사용하여 보안 URL 엔드포인트를 요청합니다.

## 작업 작업자 수준 모니터링

Amazon EMR Serverless는 AWS/EMRServerless 네임스페이스 및 Job Worker Metrics 지표 그룹에서 사용할 수 있는 다음과 같은 작업 작업자 수준 지표를 Amazon CloudWatch로 전송합니다. EMR Serverless는 작업 수준, 작업자 유형 및 용량 할당 유형 수준에서 작업이 실행되는 동안 개별 작업자로부터 데이터 포인트를 수집합니다. ApplicationId를 차원으로 사용하여 동일한 애플리케이션에 속하는 여러 작업을 모니터링할 수 있습니다.

### EMR Serverless 작업 작업자 수준 지표

지표	설명	단위	차원
WorkerCpuAllocated	작업 실행에서 작업자에 대해 할당된 총 vCPU 코어 수.	vCPU	JobId, JobName, ApplicationId, ApplicationName, WorkerType, 및 CapacityAllocationType
WorkerCpuUsed	작업 실행에서 작업자가 활용하는 총 vCPU 코어 수.	vCPU	JobId, JobName, ApplicationId, ApplicationName, WorkerType, 및 CapacityAllocationType

지표	설명	단위	차원	
WorkerMemoryAllocated	작업 실행에서 작업자에 대해 할당된 GB 단위의 총 메모리.	기가바이트(GB)	JobId, JobName, ApplicationId , ApplicationName , WorkerType , 및 CapacityAllocation Type	
WorkerMemoryUsed	작업 실행에서 작업자가 활용하는 GB 단위의 총 메모리.	기가바이트(GB)	JobId, JobName, ApplicationId , ApplicationName , WorkerType , 및 CapacityAllocation Type	
WorkerEphemeralStorageAllocated	작업 실행에서 작업자에 대해 할당된 임시 스토리지의 바이트 수.	기가바이트(GB)	JobId, JobName, ApplicationId , ApplicationName , WorkerType , 및 CapacityAllocation Type	

지표	설명	단위	차원	
WorkerEphemeralStorageUsed	작업 실행에서 작업자가 활용하는 임시 스토리지의 바이트 수.	기가바이트(GB)	JobId, JobName, ApplicationId, ApplicationName, WorkerType, 및 CapacityAllocationType	
WorkerStorageReadBytes	작업 실행에서 작업자가 스토리지에서 읽은 바이트 수.	바이트	JobId, JobName, ApplicationId, ApplicationName, WorkerType, 및 CapacityAllocationType	
WorkerStorageWriteBytes	작업 실행에서 작업자의 스토리지에 쓴 바이트 수.	바이트	JobId, JobName, ApplicationId, ApplicationName, WorkerType, 및 CapacityAllocationType	

아래 단계에서는 다양한 유형의 지표에 액세스하는 방법을 설명합니다.

## Console

콘솔을 사용하여 애플리케이션 UI에 액세스하는 방법

1. [콘솔에서 시작하기](#)의 지침에 따라 EMR Studio에서 EMR Serverless 애플리케이션으로 이동합니다.
2. 실행 중인 작업에 대한 엔진별 애플리케이션 UI 및 로그에 액세스하려면 다음을 수행합니다.
  - a. RUNNING 상태의 작업을 선택합니다.
  - b. 애플리케이션 세부 정보 페이지에서 작업을 선택하거나 작업에 대한 작업 세부 정보 페이지로 이동합니다.
  - c. UI 표시 드롭다운 메뉴에서 Spark UI 또는 Hive Tez UI를 선택하여 작업 유형에 맞는 애플리케이션 UI로 이동합니다.
  - d. Spark 엔진 로그에 액세스하려면 Spark UI의 실행기 탭으로 이동하여 드라이버의 로그 링크를 선택합니다. Hive 엔진 로그에 액세스하려면 Hive Tez UI에서 적절한 DAG에 대한 로그 링크를 선택합니다.
3. 완료된 작업에 대한 엔진별 애플리케이션 UI 및 로그에 액세스하려면 다음을 수행합니다.
  - a. SUCCESS 상태의 작업을 선택합니다.
  - b. 애플리케이션의 애플리케이션 세부 정보 페이지에서 작업을 선택하거나 작업의 작업 세부 정보 페이지로 이동합니다.
  - c. UI 표시 드롭다운 메뉴에서 Spark 기록 서버 또는 영구 Hive Tez UI를 선택하여 작업 유형에 맞는 애플리케이션 UI로 이동합니다.
  - d. Spark 엔진 로그에 액세스하려면 Spark UI의 실행기 탭으로 이동하여 드라이버의 로그 링크를 선택합니다. Hive 엔진 로그에 액세스하려면 Hive Tez UI에서 적절한 DAG에 대한 로그 링크를 선택합니다.

## AWS CLI

AWS CLI를 사용하여 애플리케이션 UI에 액세스하는 방법

- 실행 중인 작업과 완료된 작업에 대해 애플리케이션 UI에 액세스하는 데 사용할 수 있는 URL을 생성하려면 GetDashboardForJobRun API를 직접 호출합니다.

```
aws emr-serverless get-dashboard-for-job-run /  
--application-id <application-id> /  
--job-run-id <job-id>
```

생성하는 URL은 1시간 동안 유효합니다.

## Amazon Managed Service for Prometheus를 사용하여 Spark 지표 모니터링

Amazon EMR 릴리스 7.1.0 이상에서는 Amazon Managed Service for Prometheus에 EMR Serverless를 통합하여 EMR Serverless 작업 및 애플리케이션에 대한 Apache Spark 지표를 수집할 수 있습니다. 이 통합은 AWS 콘솔, EMR Serverless API 또는 AWS CLI를 사용하여 작업을 제출하거나 애플리케이션을 생성할 때 사용할 수 있습니다.

### 사전 조건

Amazon Managed Service for Prometheus에 Spark 지표를 전달하려면 먼저 다음 사전 조건을 완료합니다.

- [Amazon Managed Service for Prometheus Workspace를 생성합니다.](#) 이 Workspace는 수집 엔드포인트 역할을 합니다. 엔드포인트 - 원격 쓰기 URL에 표시되는 URL을 기록합니다. EMR Serverless 애플리케이션을 생성하는 경우 URL을 지정해야 합니다.
- 모니터링 목적으로 Amazon Managed Service for Prometheus에 작업에 대한 액세스 권한을 부여하려면 작업 실행 역할에 다음 정책을 추가합니다.

```
{
  "Sid": "AccessToPrometheus",
  "Effect": "Allow",
  "Action": ["aps:RemoteWrite"],
  "Resource": "arn:aws:aps:<AWS_REGION>:<AWS_ACCOUNT_ID>:workspace/<WORKSPACE_ID>"
}
```

### 설정

AWS 콘솔을 사용하여 Amazon Managed Service for Prometheus와 통합된 애플리케이션을 생성하는 방법

1. 애플리케이션을 생성하려면 [Amazon EMR Serverless 시작하기](#)를 참조하세요.
2. 애플리케이션을 생성하는 동안 사용자 지정 설정 사용을 선택한 다음, 구성하려는 필드에 정보를 지정하여 애플리케이션을 구성합니다.
3. 애플리케이션 로그 및 지표에서 Amazon Managed Service for Prometheus에 엔진 지표 전달을 선택한 다음, 원격 쓰기 URL을 지정합니다.

4. 원하는 다른 구성 설정을 지정한 다음, 애플리케이션 생성 및 시작을 선택합니다.

## AWS CLI 또는 EMR Serverless API 사용

create-application 또는 start-job-run 명령을 실행하는 경우 Amazon Managed Service for Prometheus와 EMR Serverless 애플리케이션을 통합하도록 AWS CLI 또는 EMR Serverless API를 사용할 수도 있습니다.

### create-application

```
aws emr-serverless create-application \
--release-label emr-7.1.0 \
--type "SPARK" \
--monitoring-configuration '{
    "prometheusMonitoringConfiguration": {
        "remoteWriteUrl": "https://aps-workspaces.<AWS_REGION>.amazonaws.com/
workspaces/<WORKSPACE_ID>/api/v1/remote_write"
    }
}'
```

### start-job-run

```
aws emr-serverless start-job-run \
--application-id <APPLICATION_ID> \
--execution-role-arn <JOB_EXECUTION_ROLE> \
--job-driver '{
    "sparkSubmit": {
        "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
        "entryPointArguments": ["10000"],
        "sparkSubmitParameters": "--conf spark.dynamicAllocation.maxExecutors=10"
    }
}' \
--configuration-overrides '{
    "monitoringConfiguration": {
        "prometheusMonitoringConfiguration": {
            "remoteWriteUrl": "https://aps-workspaces.<AWS_REGION>.amazonaws.com/
workspaces/<WORKSPACE_ID>/api/v1/remote_write"
        }
    }
}'
```

명령에 `prometheusMonitoringConfiguration`을 포함하면 EMR Serverless가 Spark 지표를 수집하고 Amazon Managed Service for Prometheus의 `remoteWriteUrl` 엔드포인트에 쓰는 에이전트와 함께 Spark 작업을 실행해야 함을 나타냅니다. 그런 다음, 시각화, 알림 및 분석을 위해 Amazon Managed Service for Prometheus의 Spark 지표를 사용할 수 있습니다.

## 고급 구성 속성

EMR Serverless는 Spark 내 구성 요소(`PrometheusServlet`)를 사용하여 Spark 지표를 수집하고 성능 데이터를 Amazon Managed Service for Prometheus와 호환되는 데이터로 변환합니다. 기본적으로 EMR Serverless는 Spark에서 기본값을 설정하고 `PrometheusMonitoringConfiguration`을 사용하여 작업을 제출할 때 드라이버 및 실행기 지표를 구문 분석합니다.

다음 표에서는 Amazon Managed Service for Prometheus로 지표를 전송하는 Spark 작업을 제출하는 경우 구성할 수 있는 모든 속성을 설명합니다.

Spark 속성	기본값	설명
<code>spark.metrics.conf</code> <code>.*.sink.prometheusServlet.class</code>	<code>org.apache.spark.metrics.sink.PrometheusServlet</code>	Spark가 Amazon Managed Service for Prometheus로 지표를 전송하는 데 사용하는 클래스. 기본 동작을 재정의하려면 사용자 지정 클래스를 지정합니다.
<code>spark.metrics.conf</code> <code>.*.source.jvm.class</code>	<code>org.apache.spark.metrics.source.JvmSource</code>	Spark가 기본 Java 가상 머신에서 중요한 지표를 수집하고 전송하는 데 사용하는 클래스. JVM 지표 수집을 중지하려면 빈 문자열(예: "")로 설정하여 이 속성을 비활성화합니다. 기본 동작을 재정의하려면 사용자 지정 클래스를 지정합니다.
<code>spark.metrics.conf</code> <code>.driver.sink.prometheusServlet.path</code>	<code>/metrics/prometheus</code>	Amazon Managed Service for Prometheus가 드라이버에서 지표를 수집하는 데 사용하는 고유한 URL. 기본 동작을 재정의하려면 자체 경로를 지정합니다.

Spark 속성	기본값	설명
		니다. 드라이버 지표 수집을 중지하려면 빈 문자열(예: "")로 설정하여 이 속성을 비활성화합니다.
spark.metrics.conf .executor.sink.prometheusServlet.path	/metrics/executor/prometheus	Amazon Managed Service for Prometheus가 실행기에서 지표를 수집하는 데 사용하는 고유한 URL. 기본 동작을 재정의하려면 자체 경로를 지정합니다. 실행기 지표 수집을 중지하려면 빈 문자열(예: "")로 설정하여 이 속성을 비활성화합니다.

Spark 지표에 대한 자세한 내용은 [Apache Spark metrics](#)를 참조하세요.

## 고려 사항 및 제한 사항

Amazon Managed Service for Prometheus를 사용하여 EMR Serverless에서 지표를 수집하는 경우 다음 고려 사항 및 제한 사항을 고려합니다.

- [Amazon Managed Service for Prometheus가 정식 출시된 AWS 리전](#)에서만 EMR Serverless에서 Amazon Managed Service for Prometheus 사용에 대한 지원이 제공됩니다.
- Amazon Managed Service for Prometheus에서 Spark 지표를 수집하기 위해 에이전트를 실행하는 경우 작업자의 리소스가 더 필요합니다. vCPU 작업자 한 명과 같이 더 작은 작업자 크기를 선택하면 작업 실행 시간이 늘어날 수 있습니다.
- EMR Serverless에서 Amazon Managed Service for Prometheus 사용 지원은 Amazon EMR 릴리스 7.1.0 이상에서만 사용 가능합니다.
- 지표를 수집하려면 EMR Serverless를 실행하는 동일한 계정에 Amazon Managed Service for Prometheus를 배포해야 합니다.

## EMR Serverless 사용 지표

Amazon CloudWatch 사용 지표를 사용하여 계정에서 사용하는 리소스에 대한 가시성을 제공할 수 있습니다. 이러한 지표를 사용하여 CloudWatch 그래프 및 대시보드에서 서비스 사용량을 시각화합니다.

EMR Serverless 사용 지표는 서비스 할당량에 해당합니다. 사용량이 서비스 할당량에 가까워지면 경고하는 경보를 구성할 수 있습니다. 자세한 내용은 CloudWatch 사용 설명서의 [Service Quotas 및 Amazon CloudWatch 경보](#)를 참조하세요.

EMR Serverless Service Quotas에 대한 자세한 내용은 [EMR Serverless에 대한 엔드포인트 및 할당량](#) 섹션을 참조하세요.

### EMR Serverless에 대한 서비스 할당량 사용 지표

EMR Serverless는 AWS/Usage 네임스페이스에서 다음과 같은 서비스 할당량 사용 지표를 게시합니다.

지표	설명
ResourceCount	계정에서 실행 중인 지정된 리소스의 총 수. 리소스는 지표와 연결된 <a href="#">차원</a> 으로 정의됩니다.

### EMR Serverless 서비스 할당량 사용 지표에 대한 차원

다음 차원을 사용하여 EMR Serverless에서 게시하는 사용 지표를 세분화할 수 있습니다.

차원	값	설명
Service	EMR Serverless	리소스가 포함된 AWS 서비스의 이름.
Type	리소스	EMR Serverless에서 보고하는 엔터티의 유형.
Resource	vCPU	EMR Serverless에서 추적하는 리소스 유형.
Class	없음	EMR Serverless에서 추적하는 리소스의 클래스.

## Amazon EventBridge를 사용하여 EMR Serverless 자동화

Amazon EventBridge를 사용하여 AWS 서비스를 자동화하고 애플리케이션 가용성 문제나 리소스 변경 같은 시스템 이벤트에 자동으로 응답합니다. EventBridge는 AWS 리소스의 변경 사항을 설명하는 시스템 이벤트의 스트림을 거의 실시간으로 제공합니다. 원하는 이벤트만 표시하도록 간단한 규칙을 작성한 후 규칙과 일치하는 이벤트 발생 시 실행할 자동화 작업을 지정할 수 있습니다. EventBridge를 사용하면 자동으로 다음을 수행할 수 있습니다.

- AWS Lambda 함수 호출
- Amazon Kinesis Data Streams로 이벤트 릴레이
- AWS Step Functions 상태 머신 활성화
- Amazon SNS 주제 또는 Amazon SQS 대기열 알림

예를 들어, EMR Serverless에서 EventBridge를 사용하는 경우 ETL 작업이 성공하면 AWS Lambda 함수를 활성화하거나 ETL 작업이 실패하면 Amazon SNS 주제를 알릴 수 있습니다.

EMR Serverless는 네 가지 종류의 이벤트를 생성합니다.

- 애플리케이션 상태 변경 이벤트 - 애플리케이션의 모든 상태 변경을 내보내는 이벤트. 애플리케이션 상태에 대한 자세한 내용은 [애플리케이션 상태](#) 섹션을 참조하세요.
- 작업 실행 상태 변경 이벤트 - 작업 실행의 모든 상태 변경을 내보내는 이벤트. 이에 대한 자세한 내용은 [작업 실행 상태](#) 섹션을 참조하세요.
- 작업 실행 재시도 이벤트 - Amazon EMR Serverless 릴리스 7.1.0 이상에서 작업 실행의 모든 재시도를 내보내는 이벤트.
- 작업 리소스 사용률 업데이트 이벤트 - 약 30분 간격으로 작업 실행에 대한 리소스 사용률 업데이트를 내보내는 이벤트.

### 샘플 EMR Serverless EventBridge 이벤트

EMR Serverless에서 보고하는 이벤트에는 다음 예제와 같이 source에 `aws.emr-serverless`의 값이 할당됩니다.

애플리케이션 상태 변경 이벤트

다음 예제 이벤트에서는 CREATING 상태의 애플리케이션을 보여줍니다.

```
{
```

```

    "version": "0",
    "id": "9fd3cf79-1ff1-b633-4dd9-34508dc1e660",
    "detail-type": "EMR Serverless Application State Change",
    "source": "aws.emr-serverless",
    "account": "123456789012",
    "time": "2022-05-31T21:16:31Z",
    "region": "us-east-1",
    "resources": [],
    "detail": {
      "applicationId": "00f1cb5c6anuij25",
      "applicationName": "3965ad00-8fba-4932-a6c8-ded32786fd42",
      "arn": "arn:aws:emr-serverless:us-east-1:111122223333:/
applications/00f1cb5c6anuij25",
      "releaseLabel": "emr-6.6.0",
      "state": "CREATING",
      "type": "HIVE",
      "createdAt": "2022-05-31T21:16:31.547953Z",
      "updatedAt": "2022-05-31T21:16:31.547970Z",
      "autoStopConfig": {
        "enabled": true,
        "idleTimeout": 15
      },
      "autoStartConfig": {
        "enabled": true
      }
    }
  }
}

```

## 작업 실행 상태 변경 이벤트

다음 예제 이벤트에서는 SCHEDULED 상태에서 RUNNING 상태로 전환되는 작업 실행을 보여줍니다.

```

{
  "version": "0",
  "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
  "detail-type": "EMR Serverless Job Run State Change",
  "source": "aws.emr-serverless",
  "account": "123456789012",
  "time": "2022-05-31T21:07:42Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "jobRunId": "00f1cbn5g4bb0c01",
    "applicationId": "00f1982r1uukb925",

```

```

    "arn": "arn:aws:emr-serverless:us-east-1:123456789012:/
applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01",
    "releaseLabel": "emr-6.6.0",
    "state": "RUNNING",
    "previousState": "SCHEDULED",
    "createdBy": "arn:aws:sts::123456789012:assumed-role/
TestRole-402dcef3ad14993c15d28263f64381e4cda34775/6622b6233b6d42f59c25dd2637346242",
    "updatedAt": "2022-05-31T21:07:42.299487Z",
    "createdAt": "2022-05-31T21:07:25.325900Z"
  }
}

```

## 작업 실행 재시도 이벤트

다음은 작업 실행 재시도 레코드의 예제입니다.

```

{
  "version": "0",
  "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
  "detail-type": "EMR Serverless Job Run Retry",
  "source": "aws.emr-serverless",
  "account": "123456789012",
  "time": "2022-05-31T21:07:42Z",
  "region": "us-east-1",
  "resources": [],
  "detail": {
    "jobRunId": "00f1cbn5g4bb0c01",
    "applicationId": "00f1982r1uukb925",
    "arn": "arn:aws:emr-serverless:us-east-1:123456789012:/
applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01",
    "releaseLabel": "emr-6.6.0",
    "createdBy": "arn:aws:sts::123456789012:assumed-role/
TestRole-402dcef3ad14993c15d28263f64381e4cda34775/6622b6233b6d42f59c25dd2637346242",
    "updatedAt": "2022-05-31T21:07:42.299487Z",
    "createdAt": "2022-05-31T21:07:25.325900Z",
    //Attempt Details
    "previousAttempt": 1,
    "previousAttemptState": "FAILED",
    "previousAttemptCreatedAt": "2022-05-31T21:07:25.325900Z",
    "previousAttemptEndedAt": "2022-05-31T21:07:30.325900Z",
    "newAttempt": 2,
    "newAttemptCreatedAt": "2022-05-31T21:07:30.325900Z"
  }
}

```

}

## 작업 리소스 사용률 업데이트

다음 예제 이벤트에서는 실행 후 터미널 상태로 전환되는 작업에 대한 최종 리소스 사용률 업데이트를 보여줍니다.

```
{
  "version": "0",
  "id": "00df3ec6-5da1-36e6-ab71-20f0de68f8a0",
  "detail-type": "EMR Serverless Job Resource Utilization Update",
  "source": "aws.emr-serverless",
  "account": "123456789012",
  "time": "2022-05-31T21:07:42Z",
  "region": "us-east-1",
  "resources": [
    "arn:aws:emr-serverless:us-east-1:123456789012:/applications/00f1982r1uukb925/jobruns/00f1cbn5g4bb0c01"
  ],
  "detail": {
    "applicationId": "00f1982r1uukb925",
    "jobRunId": "00f1cbn5g4bb0c01",
    "attempt": 1,
    "mode": "BATCH",
    "createdAt": "2022-05-31T21:07:25.325900Z",
    "startedAt": "2022-05-31T21:07:26.123Z",
    "calculatedFrom": "2022-05-31T21:07:42.299487Z",
    "calculatedTo": "2022-05-31T21:07:30.325900Z",
    "resourceUtilizationFinal": true,
    "resourceUtilizationForInterval": {
      "vCPUHour": 0.023,
      "memoryGBHour": 0.114,
      "storageGBHour": 0.228
    },
    "billedResourceUtilizationForInterval": {
      "vCPUHour": 0.067,
      "memoryGBHour": 0.333,
      "storageGBHour": 0
    },
    "totalResourceUtilization": {
      "vCPUHour": 0.023,
      "memoryGBHour": 0.114,
      "storageGBHour": 0.228
    }
  }
}
```

```
    },  
    "totalBilledResourceUtilization": {  
      "vCPUPercent": 0.067,  
      "memoryGBHour": 0.333,  
      "storageGBHour": 0  
    }  
  }  
}
```

작업이 실행 중 상태로 전환된 경우에만 `startedAt` 필드가 이벤트 내에 표시됩니다.

## 리소스에 태그 지정

EMR Serverless 리소스 관리를 지원하기 위해 태그를 사용하여 각 리소스에 고유한 메타데이터를 할당할 수 있습니다. 이 섹션에서는 태그 함수에 대한 개요를 제공하고 태그를 생성하는 방법을 보여줍니다.

### 주제

- [태그란 무엇입니까?](#)
- [리소스에 태깅](#)
- [태그 지정 제한 사항](#)
- [AWS CLI 및 Amazon EMR Serverless API를 사용하여 태그 작업](#)

## 태그란 무엇입니까?

태그는 AWS 리소스에 할당하는 레이블입니다. 각 태그는 사용자가 정의하는 키와 값으로 구성됩니다. 태그를 사용하면 용도, 소유자 및 환경과 같은 속성으로 AWS 리소스를 분류할 수 있습니다. 동일한 유형의 리소스가 많은 경우 할당한 태그에 따라 특정 리소스를 빠르게 식별합니다. 예를 들어, 각 애플리케이션의 소유자 및 스택 수준을 추적할 수 있도록 Amazon EMR Serverless 애플리케이션에 대한 태그 세트를 정의할 수 있습니다. 각 리소스 유형에 대해 일관된 태그 키 집합을 고안하는 것이 좋습니다.

태그가 리소스에 자동으로 할당되는 것은 아닙니다. 리소스에 태그를 추가한 후에는 언제든지 태그의 값을 수정하거나 리소스에서 태그를 제거할 수 있습니다. 태그는 Amazon EMR Serverless에 대한 시맨틱 의미가 없으며 엄격하게 문자열로 해석됩니다. 해당 리소스에 대해 키가 기존 태그와 동일한 태그를 추가하는 경우 새 값이 이전 값을 덮어씁니다.

IAM을 사용하는 경우 AWS 계정에서 태그를 관리할 수 있는 권한을 가진 사용자를 제어할 수 있습니다. 태그 기반 액세스 제어 정책 예제는 [태그 기반 액세스 제어를 위한 정책](#) 섹션을 참조하세요.

## 리소스에 태깅

신규 또는 기존 애플리케이션 및 작업 실행에 태그를 지정할 수 있습니다. Amazon EMR Serverless API, AWS CLI 또는 AWS SDK를 사용 중인 경우 관련 API 작업의 tags 파라미터를 사용하여 새 리소스에 태그를 적용할 수 있습니다. TagResource API 작업을 사용하여 기존 리소스에 태그를 적용할 수도 있습니다.

일부 리소스 생성 작업을 사용해서 리소스 생성 시 리소스의 태그를 지정할 수 있습니다. 이 경우 리소스를 생성하는 동안 태그를 적용할 수 없는 경우 리소스를 생성하지 못합니다. 이 매커니즘에서는 생성

중에 태그를 지정하려는 리소스가 지정된 태그와 함께 생성되거나 전혀 생성되지 않습니다. 생성 시 리소스에 태그를 지정하면 리소스 생성 후 사용자 지정 태깅 스크립트를 실행하지 않아도 됩니다.

다음 테이블에서는 태그를 지정할 수 있는 Amazon EMR Serverless 리소스를 설명합니다.

### 태그 지정 가능한 리소스

리소스	태그 지원	태그 전달 지원	생성 시 태그 지정 지원(Amazon EMR Serverless API, AWS CLI 및 AWS SDK)	생성을 위한 API(생성 중에 태그를 추가할 수 있음)
Application	예	아니요. 애플리케이션과 연결된 태그는 해당 애플리케이션에 제출된 작업 실행으로 전파되지 않습니다.	예	CreateApplication
작업 실행	예	아니요	예	StartJobRun

## 태그 지정 제한 사항

다음과 같은 기본 제한 사항이 태그에 적용됩니다.

- 각 리소스에는 최대 50개 사용자 생성 태그가 포함될 수 있습니다.
- 각 리소스에 대해 각 태그 키는 고유하며 하나의 값만 가질 수 있습니다.
- 키의 최대 길이는 UTF-8 형식의 유니코드 문자 128자입니다.
- 값의 최대 길이는 UTF-8 형식의 유니코드 문자 256자입니다.
- 허용되는 문자는 UTF-8로 표현할 수 있는 문자, 숫자, 공백 및 `_ . : / = + - @` 문자입니다.
- 태그 키는 빈 문자열일 수 없습니다. 태그 값에 빈 문자열은 지정할 수 있지만, null은 지정할 수 없습니다.
- 태그 키와 값은 대소문자를 구분합니다.
- 키 또는 값의 접두사와 같이 대문자 또는 소문자 조합 또는 `AWS:`를 사용하지 않습니다. 이러한 이름은 AWS 전용으로 예약되어 있습니다.

## AWS CLI 및 Amazon EMR Serverless API를 사용하여 태그 작업

다음 AWS CLI 명령 또는 Amazon EMR Serverless API 작업을 사용하여 리소스에 대한 태그를 추가, 업데이트, 나열 및 삭제합니다.

태그에 대한 CLI 명령 및 API 작업

리소스	태그 지원	태그 전달 지원
하나 이상의 태그를 추가하거나 덮어씁니다.	tag-resource	TagResource
리소스에 대한 태그를 나열합니다.	list-tags-for-resource	ListTagsForResource
하나 이상의 태그를 삭제합니다.	untag-resource	UntagResource

다음 예제는 AWS CLI를 사용하여 리소스에 태그를 지정하거나 태그를 제거하는 방법을 보여줍니다.

기존 애플리케이션 태그 지정

다음 명령은 기존 애플리케이션에 태그를 지정합니다.

```
aws emr-serverless tag-resource --resource-arn resource_ARN --tags team=devs
```

기존 애플리케이션 사용

다음 명령은 기존 애플리케이션에서 태그를 삭제합니다.

```
aws emr-serverless untag-resource --resource-arn resource_ARN --tag-keys tag_key
```

리소스에 대한 태그 나열

다음 명령은 기존 리소스와 연결된 태그를 나열합니다.

```
aws emr-serverless list-tags-for-resource --resource-arn resource_ARN
```

# EMR Serverless용 자습서

이 섹션에서는 EMR Serverless 애플리케이션에서 작업할 때 일반적인 사용 사례를 설명합니다. 여기에는 Spark 작업을 제출하기 위해 Python 및 Python 라이브러리를 사용하고 대규모 데이터세트에 대한 작업을 수행하기 위해 Hudi 및 Iceberg를 비롯한 다양한 도구가 포함되어 있습니다.

## 주제

- [Amazon EMR Serverless에서 Java 17 사용](#)
- [EMR Serverless에서 Apache Hudi 사용](#)
- [EMR Serverless에서 Apache Iceberg 사용](#)
- [EMR Serverless에서 Python 라이브러리 사용](#)
- [EMR Serverless에서 다양한 Python 버전 사용](#)
- [EMR Serverless와 함께 델타 레이크 OSS 사용](#)
- [Airflow에서 EMR Serverless 작업 제출](#)
- [EMR Serverless에서 Hive 사용자 정의 함수 사용](#)
- [EMR Serverless에서 사용자 지정 이미지 사용](#)
- [Amazon EMR Serverless에서 Apache Spark용 Amazon Redshift 통합 사용](#)
- [Amazon EMR Serverless를 사용하여 DynamoDB에 연결](#)

## Amazon EMR Serverless에서 Java 17 사용

Amazon EMR 릴리스 6.11.0 이상을 사용하면 Java Virtual Machine(JVM)에 대해 Java 17 런타임을 사용하도록 EMR Serverless Spark 작업을 구성할 수 있습니다. 다음 방법 중 하나를 사용하여 Java 17로 Spark를 구성합니다.

### JAVA\_HOME

EMR Serverless 6.11.0 이상의 JVM 설정을 재정의하기 위해 `spark.emr-serverless.driverEnv` 및 `spark.executorEnv` 환경 분류에 `JAVA_HOME` 설정을 제공할 수 있습니다.

#### x86\_64

Java 17을 Spark 드라이버 및 실행기의 `JAVA_HOME` 구성으로 지정하기 위해 필요한 속성을 설정합니다.

```
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.x86_64/
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.x86_64/
```

## arm\_64

Java 17을 Spark 드라이버 및 실행기의 JAVA\_HOME 구성으로 지정하기 위해 필요한 속성을 설정합니다.

```
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.aarch64/
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-17-amazon-corretto.aarch64/
```

## spark-defaults

또는 spark-defaults 분류에서 Java 17을 지정하여 EMR Serverless 6.11.0 이상에 대한 JVM 설정을 재정의할 수 있습니다.

## x86\_64

spark-defaults 분류에서 Java 17을 지정합니다.

```
{
  "applicationConfiguration": [
    {
      "classification": "spark-defaults",
      "properties": {
        "spark.emr-serverless.driverEnv.JAVA_HOME" : "/usr/lib/jvm/java-17-amazon-corretto.x86_64/",
        "spark.executorEnv.JAVA_HOME": "/usr/lib/jvm/java-17-amazon-corretto.x86_64/"
      }
    }
  ]
}
```

## arm\_64

spark-defaults 분류에서 Java 17을 지정합니다.

```
{
```

```

"applicationConfiguration": [
  {
    "classification": "spark-defaults",
    "properties": {
      "spark.emr-serverless.driverEnv.JAVA_HOME" : "/usr/lib/jvm/java-17-
amazon-corretto.aarch64/",
      "spark.executorEnv.JAVA_HOME": "/usr/lib/jvm/java-17-amazon-
corretto.aarch64/"
    }
  }
]
}

```

## EMR Serverless에서 Apache Hudi 사용

이 섹션에서는 EMR Serverless 애플리케이션에서 Apache Hudi를 사용하는 방법을 설명합니다. Hudi는 데이터 처리를 더 간단하게 수행하도록 지원하는 데이터 관리 프레임워크입니다.

EMR Serverless 애플리케이션에서 Apache Hudi를 사용하는 방법

1. 해당 Spark 작업 실행에서 필요한 Spark 속성을 설정합니다.

```

spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar,/usr/lib/hudi/hudi-utilities-
bundle.jar,/usr/lib/hudi/hudi-aws-bundle.jar
spark.serializer=org.apache.spark.serializer.KryoSerializer

```

2. Hudi 테이블을 구성된 카탈로그에 동기화하려면 AWS Glue Data Catalog를 메타스토어로 지정하거나 외부 메타스토어를 구성합니다. EMR Serverless는 Hudi 워크로드에 대해 Hive 테이블의 동기화 모드로 hms를 지원합니다. EMR Serverless는 이 속성을 기본적으로 활성화합니다. 메타스토어를 설정하는 방법에 대한 자세한 내용은 [EMR Serverless에 대한 메타스토어 구성](#) 섹션을 참조하세요.

### Important

EMR Serverless는 Hudi 워크로드를 처리하기 위해 Hive 테이블에서 동기화 모드 옵션으로 HIVEQL 또는 JDBC를 지원하지 않습니다. 자세한 내용은 [Sync modes](#)를 참조하세요.

AWS Glue Data Catalog를 메타스토어로 사용하는 경우 Hudi 작업에 대해 다음 구성 속성을 지정할 수 있습니다.

```
--conf spark.jars=/usr/lib/hudi/hudi-spark-bundle.jar,
--conf spark.serializer=org.apache.spark.serializer.KryoSerializer,
--conf
spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveMetastore
```

Amazon EMR의 Apache Hudi 릴리스 버전에 대한 자세한 내용은 [Hudi 릴리스 기록](#)을 참조하세요.

## EMR Serverless에서 Apache Iceberg 사용

이 섹션에서는 EMR Serverless 애플리케이션에서 Apache Iceberg를 사용하는 방법을 설명합니다. Apache Iceberg는 데이터 레이크에서 대규모 데이터세트를 사용하는 데 유용한 테이블 형식입니다.

EMR Serverless 애플리케이션에서 Apache Iceberg를 사용하는 방법

1. 해당 Spark 작업 실행에서 필요한 Spark 속성을 설정합니다.

```
spark.jars=/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar
```

2. Glue Data Catalog를 AWS 메타스토어로 지정하거나 외부 메타스토어를 구성합니다. 메타스토어 설정에 대한 자세한 내용은 [EMR Serverless에 대한 메타스토어 구성](#) 섹션을 참조하세요.

Iceberg에 대해 사용할 메타스토어 속성을 구성합니다. 예를 들어, AWS Glue Data Catalog를 사용하려면 애플리케이션 구성에서 다음 속성을 설정합니다.

```
spark.sql.catalog.dev.warehouse=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions
spark.sql.catalog.dev=org.apache.iceberg.spark.SparkCatalog
spark.sql.catalog.dev.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog
spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveMetastore
```

AWS Glue Data Catalog를 메타스토어로 사용하는 경우 Iceberg 작업에 대해 다음 구성 속성을 지정할 수 있습니다.

```
--conf spark.jars=/usr/share/aws/iceberg/lib/iceberg-spark3-runtime.jar,
```

```
--conf
  spark.sql.extensions=org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions,
--conf spark.sql.catalog.dev=org.apache.iceberg.spark.SparkCatalog,
--conf spark.sql.catalog.dev.catalog-impl=org.apache.iceberg.aws.glue.GlueCatalog,
--conf spark.sql.catalog.dev.warehouse=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
--conf
  spark.hadoop.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSG
```

Amazon EMR의 Apache Iceberg 릴리스 버전에 대한 자세한 내용은 [Iceberg 릴리스 기록](#)을 참조하세요.

## EMR Serverless에서 Python 라이브러리 사용

Amazon EMR Serverless 애플리케이션에서 PySpark 작업을 실행하는 경우 다양한 Python 라이브러리를 종속 항목으로 패키징할 수 있습니다. 이를 수행하려면 기본 Python 기능을 사용하거나, 가상 환경을 빌드하거나, Python 라이브러리를 사용하도록 PySpark 작업을 직접 구성할 수 있습니다. 이 페이지에서는 각 접근 방식을 다룹니다.

### 기본 Python 기능 사용

다음 구성을 설정하는 경우 PySpark를 사용하여 Python 파일(.py), 압축된 Python 패키지(.zip) 및 Egg 파일(.egg)을 Spark 실행기에 업로드할 수 있습니다.

```
--conf spark.submit.pyFiles=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/<.py|.egg|.zip
file>
```

PySpark 작업에 대해 Python 가상 환경을 사용하는 방법에 대한 자세한 내용은 [Using PySpark Native Features](#)를 참조하세요.

EMR Notebook을 사용하는 경우 다음 코드를 실행하여 노트북에서 Python 종속성을 사용할 수 있도록 설정할 수 있습니다.

```
%%configure -f
{
  "conf": {
    "spark.submit.pyFiles":"s3:///amzn-s3-demo-bucket/EXAMPLE-PREFIX/<.py|.egg|.zip
file>
  }
}
```

## Python 가상 환경 빌드

PySpark 커널 내에 여러 Python 라이브러리를 패키징하려는 경우 격리된 Python 가상 환경을 생성할 수도 있습니다.

1. Python 가상 환경을 빌드하려면 다음 명령을 사용합니다. 표시된 예제에서는 `scipy` 및 `matplotlib` 패키지를 가상 환경 패키지에 설치하고 아카이브를 Amazon S3 위치에 복사합니다.

### Important

EMR Serverless에서 사용하는 것과 동일한 버전의 Python(즉, Amazon EMR 릴리스 6.6.0의 경우 Python 3.7.10)을 사용하여 유사한 Amazon Linux 2 환경에서 다음 명령을 실행해야 합니다. [EMR Serverless Samples](#) GitHub 리포지토리에서 예제 Dockerfile을 찾을 수 있습니다.

```
# initialize a python virtual environment
python3 -m venv pyspark_venvsource
source pyspark_venvsource/bin/activate

# optionally, ensure pip is up-to-date
pip3 install --upgrade pip

# install the python packages
pip3 install scipy
pip3 install matplotlib

# package the virtual environment into an archive
pip3 install venv-pack
venv-pack -f -o pyspark_venv.tar.gz

# copy the archive to an S3 location
aws s3 cp pyspark_venv.tar.gz s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/

# optionally, remove the virtual environment directory
rm -fr pyspark_venvsource
```

2. Python 가상 환경을 사용하도록 속성이 설정된 Spark 작업을 제출합니다.

```
--conf spark.archives=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
pyspark_venv.tar.gz#environment
```

```
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=./environment/bin/python
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=./environment/bin/python
--conf spark.executorEnv.PYSPARK_PYTHON=./environment/bin/python
```

원래 Python 바이너리를 재정의하지 않으면 이전 설정 시퀀스의 두 번째 구성은 `--conf spark.executorEnv.PYSPARK_PYTHON=python`입니다.

PySpark 작업에 대해 Python 가상 환경을 사용하는 방법에 대한 자세한 내용은 [Using Virtualenv](#)를 참조하세요. Spark 작업을 제출하는 방법에 대한 자세한 예제는 [EMR Serverless 작업을 실행하는 경우 Spark 구성 사용](#) 섹션을 참조하세요.

## Python 라이브러리를 사용하도록 PySpark 작업 구성

Amazon EMR 릴리스 6.12.0 이상을 사용하면 추가 설정 없이 [pandas](#), [NumPy](#) 및 [PyArrow](#)와 같은 널리 사용되는 데이터 과학 Python 라이브러리를 사용하도록 EMR Serverless PySpark 작업을 직접 구성할 수 있습니다.

다음 예제에서는 PySpark 작업에 대해 각 Python 라이브러리를 패키징하는 방법을 보여줍니다.

### NumPy

NumPy는 수학, 정렬, 무작위 시뮬레이션 및 기본 통계를 위한 다차원 배열 및 작업을 제공하는 과학 컴퓨팅용 Python 라이브러리입니다. NumPy를 사용하려면 다음 명령을 실행합니다.

```
import numpy
```

### pandas

pandas는 NumPy에 빌드된 Python 라이브러리입니다. pandas 라이브러리는 데이터 과학자에게 [DataFrame](#) 데이터 구조 및 데이터 분석 도구를 제공합니다. pandas를 사용하려면 다음 명령을 실행합니다.

```
import pandas
```

### PyArrow

PyArrow는 작업 성능 개선을 위해 메모리 내 열 데이터를 관리하는 Python 라이브러리입니다. PyArrow는 Apache Arrow 다국어 개발 사양을 기반으로 하며, 이는 열 형식으로 데이터를 표현하고 교환하는 표준 방법입니다. PyArrow를 사용하려면 다음 명령을 실행합니다.

```
import pyarrow
```

## EMR Serverless에서 다양한 Python 버전 사용

[EMR Serverless에서 Python 라이브러리 사용](#)의 사용 사례 외에도 Python 가상 환경을 사용하여 Amazon EMR Serverless 애플리케이션에 대한 Amazon EMR 릴리스에서 패키지로 제공되는 버전과 다른 Python 버전으로 작업할 수도 있습니다. 이를 수행하려면 사용하려는 Python 버전으로 Python 가상 환경을 빌드합니다.

Python 가상 환경에서 작업을 제출하는 방법

1. 다음 예제의 명령을 사용하여 가상 환경을 빌드합니다. 이 예제에서는 Python 3.9.9를 가상 환경 패키지에 설치하고 아카이브를 Amazon S3 위치에 복사합니다.

### Important

Amazon EMR 릴리스 7.0.0 이상을 사용하는 경우 EMR Serverless 애플리케이션에 대해 사용하는 것과 유사한 Amazon Linux 2023 환경에서 명령을 실행합니다.  
릴리스 6.15.0 이하를 사용하는 경우 유사한 Amazon Linux 2 환경에서 다음 명령을 실행합니다.

```
# install Python 3.9.9 and activate the venv
yum install -y gcc openssl-devel bzip2-devel libffi-devel tar gzip wget make
wget https://www.python.org/ftp/python/3.9.9/Python-3.9.9.tgz && \
tar xzf Python-3.9.9.tgz && cd Python-3.9.9 && \
./configure --enable-optimizations && \
make altinstall

# create python venv with Python 3.9.9
python3.9 -m venv pyspark_venv_python_3.9.9 --copies
source pyspark_venv_python_3.9.9/bin/activate

# copy system python3 libraries to venv
cp -r /usr/local/lib/python3.9/* ./pyspark_venv_python_3.9.9/lib/python3.9/

# package venv to archive.
# **Note** that you have to supply --python-prefix option
# to make sure python starts with the path where your
```

```
# copied libraries are present.
# Copying the python binary to the "environment" directory.
pip3 install venv-pack
venv-pack -f -o pyspark_venv_python_3.9.9.tar.gz --python-prefix /home/hadoop/
environment

# stage the archive in S3
aws s3 cp pyspark_venv_python_3.9.9.tar.gz s3://<path>

# optionally, remove the virtual environment directory
rm -fr pyspark_venv_python_3.9.9
```

## 2. Python 가상 환경을 사용하도록 속성을 설정하고 Spark 작업을 제출합니다.

```
# note that the archive suffix "environment" is the same as the directory where you
  copied the Python binary.
--conf spark.archives=s3://amzn-s3-demo-bucket/EXAMPLE-PREFIX/
  pyspark_venv_python_3.9.9.tar.gz#environment
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=./environment/bin/
  python
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=./environment/bin/python
--conf spark.executorEnv.PYSPARK_PYTHON=./environment/bin/python
```

PySpark 작업에 대해 Python 가상 환경을 사용하는 방법에 대한 자세한 내용은 [Using Virtualenv](#)를 참조하세요. Spark 작업을 제출하는 방법에 대한 자세한 예제는 [EMR Serverless 작업을 실행하는 경우 Spark 구성 사용](#) 섹션을 참조하세요.

## EMR Serverless와 함께 델타 레이크 OSS 사용

### Amazon EMR 버전 6.9.0 이상

#### Note

Amazon EMR 7.0.0 이상은 Delta Lake 3.0.0(delta-core.jar 이름이 delta-spark.jar로 바뀜)을 사용합니다. Amazon EMR 7.0.0 이상을 사용하는 경우 구성에서 delta-spark.jar를 지정해야 합니다.

Amazon EMR 6.9.0 이상에는 Delta Lake가 포함되어 있으므로 더 이상 Delta Lake를 직접 패키지로 제공하거나 EMR Serverless 작업과 함께 --packages 플래그를 제공하지 않아도 됩니다.

1. EMR Serverless 작업을 제출하는 경우 다음 구성 속성이 있고 `sparkSubmitParameters` 필드에 다음 파라미터를 포함하는지 확인합니다.

```
--conf spark.jars=/usr/share/aws/delta/lib/delta-core.jar,/usr/share/aws/delta/lib/delta-storage.jar
--conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension
--conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog
```

2. 로컬 `delta_sample.py`를 생성하여 델타 테이블 생성 및 읽기를 테스트합니다.

```
# delta_sample.py
from pyspark.sql import SparkSession

import uuid

url = "s3://amzn-s3-demo-bucket/delta-lake/output/%s/" % str(uuid.uuid4())
spark = SparkSession.builder.appName("DeltaSample").getOrCreate()

## creates a Delta table and outputs to target S3 bucket
spark.range(5).write.format("delta").save(url)

## reads a Delta table and outputs to target S3 bucket
spark.read.format("delta").load(url).show
```

3. AWS CLI를 사용하여 `delta_sample.py` 파일을 Amazon S3 버킷에 업로드합니다. 그런 다음, `start-job-run` 명령을 사용하여 기존 EMR Serverless 애플리케이션에 작업을 제출합니다.

```
aws s3 cp delta_sample.py s3://amzn-s3-demo-bucket/code/

aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --name emr-delta \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/code/delta_sample.py",
      "sparkSubmitParameters": "--conf spark.jars=/usr/share/
aws/delta/lib/delta-core.jar,/usr/share/aws/delta/lib/delta-storage.jar --
conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension --conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog"
    }
  }
```

```
}'
```

Delta Lake와 함께 Python 라이브러리를 사용하려면 라이브러리를 [종속 항목으로 패키징](#)하거나 [사용자 지정 이미지로 사용](#)하여 delta-core 라이브러리를 추가할 수 있습니다.

또는 `SparkContext.addPyFile`을 사용하여 delta-core JAR 파일에서 Python 라이브러리를 추가할 수 있습니다.

```
import glob
from pyspark.sql import SparkSession

spark = SparkSession.builder.getOrCreate()
spark.sparkContext.addPyFile(glob.glob("/usr/share/aws/delta/lib/delta-core_*.jar")[0])
```

## Amazon EMR 버전 6.8.0 이하

Amazon EMR 6.8.0 이하를 사용하는 경우 다음 단계를 수행하여 EMR 서버리스 애플리케이션에서 Delta Lake OSS를 사용합니다.

1. Amazon EMR Serverless 애플리케이션의 Spark 버전과 호환되는 오픈 소스 버전의 [Delta Lake](#)를 빌드하려면 [Delta GitHub](#)로 이동한 후 지침을 따릅니다.
2. Delta Lake 라이브러리를 AWS 계정의 Amazon S3 버킷에 업로드합니다.
3. 애플리케이션 구성에서 EMR Serverless 작업을 제출하는 경우 현재 버킷에 있는 Delta Lake JAR 파일을 포함합니다.

```
--conf spark.jars=s3://amzn-s3-demo-bucket/jars/delta-core_2.12-1.1.0.jar
```

4. Delta 테이블에서 읽고 쓸 수 있는지 확인하려면 샘플 PySpark 테스트를 실행합니다.

```
from pyspark import SparkConf, SparkContext
from pyspark.sql import HiveContext, SparkSession

import uuid

conf = SparkConf()
sc = SparkContext(conf=conf)
sqlContext = HiveContext(sc)

url = "s3://amzn-s3-demo-bucket/delta-lake/output/1.0.1/%s/" %
str(uuid.uuid4())
```

```
## creates a Delta table and outputs to target S3 bucket
session.range(5).write.format("delta").save(url)
```

```
## reads a Delta table and outputs to target S3 bucket
session.read.format("delta").load(url).show
```

## Airflow에서 EMR Serverless 작업 제출

Apache Airflow의 Amazon Provider에서는 EMR Serverless 연산자를 제공합니다. 연산자에 대한 자세한 내용은 Apache Airflow 설명서의 [Amazon EMR Serverless Operators](#)를 참조하세요.

`EmrServerlessCreateApplicationOperator`를 사용하여 Spark 또는 Hive 애플리케이션을 생성할 수 있습니다. `EmrServerlessStartJobOperator`를 사용하여 새 애플리케이션으로 하나 이상의 작업을 시작할 수도 있습니다.

Airflow 2.2.2와 함께 Amazon Managed Workflows for Apache Airflow(MWAA)에서 연산자를 사용하려면 `requirements.txt` 파일에 다음 줄을 추가하고 MWAA 환경을 업데이트하여 새 파일을 사용합니다.

```
apache-airflow-providers-amazon==6.0.0
boto3>=1.23.9
```

Amazon 제공업체의 릴리스 5.0.0에는 EMR Serverless 지원이 추가되었습니다. 릴리스 6.0.0은 Airflow 2.2.2와 호환되는 마지막 버전입니다. MWAA의 Airflow 2.4.3에서 이후 버전을 사용할 수 있습니다.

다음 약식 예제에서는 애플리케이션을 생성하고 여러 Spark 작업을 실행한 다음, 애플리케이션을 중지하는 방법을 보여줍니다. 전체 예제는 [EMR Serverless Samples](#) GitHub 리포지토리에서 사용할 수 있습니다. `sparkSubmit` 구성에 대한 자세한 내용은 [EMR Serverless 작업을 실행하는 경우 Spark 구성 사용](#) 섹션을 참조하세요.

```
from datetime import datetime

from airflow import DAG
from airflow.providers.amazon.aws.operators.emr import (
    EmrServerlessCreateApplicationOperator,
    EmrServerlessStartJobOperator,
    EmrServerlessDeleteApplicationOperator,
)
```

```

# Replace these with your correct values
JOB_ROLE_ARN = "arn:aws:iam::account-id:role/emr_serverless_default_role"
S3_LOGS_BUCKET = "amzn-s3-demo-bucket"

DEFAULT_MONITORING_CONFIG = {
    "monitoringConfiguration": {
        "s3MonitoringConfiguration": {"logUri": f"s3://{amzn-s3-demo-bucket}/logs/"}
    },
}

with DAG(
    dag_id="example_endtoend_emr_serverless_job",
    schedule_interval=None,
    start_date=datetime(2021, 1, 1),
    tags=["example"],
    catchup=False,
) as dag:
    create_app = EmrServerlessCreateApplicationOperator(
        task_id="create_spark_app",
        job_type="SPARK",
        release_label="emr-6.7.0",
        config={"name": "airflow-test"},
    )

    application_id = create_app.output

    job1 = EmrServerlessStartJobOperator(
        task_id="start_job_1",
        application_id=application_id,
        execution_role_arn=JOB_ROLE_ARN,
        job_driver={
            "sparkSubmit": {
                "entryPoint": "local:///usr/lib/spark/examples/src/main/python/
pi_fail.py",
            }
        },
        configuration_overrides=DEFAULT_MONITORING_CONFIG,
    )

    job2 = EmrServerlessStartJobOperator(
        task_id="start_job_2",
        application_id=application_id,
        execution_role_arn=JOB_ROLE_ARN,

```

```

    job_driver={
        "sparkSubmit": {
            "entryPoint": "local:///usr/lib/spark/examples/src/main/python/pi.py",
            "entryPointArguments": ["1000"]
        }
    },
    configuration_overrides=DEFAULT_MONITORING_CONFIG,
)

delete_app = EmrServerlessDeleteApplicationOperator(
    task_id="delete_app",
    application_id=application_id,
    trigger_rule="all_done",
)

(create_app >> [job1, job2] >> delete_app)

```

## EMR Serverless에서 Hive 사용자 정의 함수 사용

Hive 사용자 정의 함수(UDF)를 사용하면 레코드 또는 레코드 그룹을 처리하기 위한 사용자 지정 함수를 생성할 수 있습니다. 이 자습서에서는 기존 Amazon EMR Serverless 애플리케이션에서 샘플 UDF를 사용하여 쿼리 결과를 출력하는 작업을 실행합니다. 애플리케이션을 설정하는 방법을 알아보려면 [Amazon EMR Serverless 시작하기](#) 섹션을 참조하세요.

### EMR Serverless에서 UDF를 사용하는 방법

1. 샘플 UDF의 [GitHub](#)로 이동합니다. 리포지토리를 복제하고 사용하려는 git 브랜치로 전환합니다. 리포지토리의 pom.xml 파일에서 소스를 포함하도록 maven-compiler-plugin을 업데이트합니다. 또한 대상 Java 버전 구성을 1.8로 업데이트합니다. `mvn package -DskipTests`를 실행하여 샘플 UDF가 포함된 JAR 파일을 생성합니다.
2. JAR 파일을 생성한 후 다음 명령을 사용하여 S3 버킷에 업로드합니다.

```
aws s3 cp brickhouse-0.8.2-JS.jar s3://amzn-s3-demo-bucket/jars/
```

3. 샘플 UDF 함수 중 하나를 사용하도록 예제 파일을 생성합니다. 이 쿼리를 `udf_example.q`로 저장하고 S3 버킷에 업로드합니다.

```
add jar s3://amzn-s3-demo-bucket/jars/brickhouse-0.8.2-JS.jar;
CREATE TEMPORARY FUNCTION from_json AS 'brickhouse.udf.json.FromJsonUDF';
```

```
select from_json('{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}', map("",
array(cast(0 as int))));
select from_json('{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}', map("",
array(cast(0 as int))))["key1"][2];
```

#### 4. 다음 Hive 작업을 제출합니다.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "hive": {
      "query": "s3://amzn-s3-demo-bucket/queries/udf_example.q",
      "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/
emr-serverless-hive/scratch --hiveconf hive.metastore.warehouse.dir=s3://'$BUCKET'/
emr-serverless-hive/warehouse"
    }
  }' --configuration-overrides '{
  "applicationConfiguration": [{
    "classification": "hive-site",
    "properties": {
      "hive.driver.cores": "2",
      "hive.driver.memory": "6G"
    }
  }],
  "monitoringConfiguration": {
    "s3MonitoringConfiguration": {
      "logUri": "s3://amzn-s3-demo-bucket/logs/"
    }
  }
}
```

#### 5. get-job-run 명령을 사용하여 작업의 상태를 확인합니다. 상태가 SUCCESS로 변경될 때까지 기다립니다.

```
aws emr-serverless get-job-run --application-id application-id --job-run-id job-id
```

#### 6. 다음 명령을 사용하여 출력 파일을 다운로드합니다.

```
aws s3 cp --recursive s3://amzn-s3-demo-bucket/logs/applications/application-id/
jobs/job-id/HIVE_DRIVER/ .
```

stdout.gz 파일은 다음과 비슷합니다.

```
{"key1":[0,1,2], "key2":[3,4,5,6], "key3":[7,8,9]}
2
```

## EMR Serverless에서 사용자 지정 이미지 사용

### 주제

- [사용자 지정 Python 버전 사용](#)
- [사용자 지정 Java 버전 사용](#)
- [데이터 과학 이미지 빌드](#)
- [Apache Sedona를 사용한 지리 공간 데이터 처리](#)
- [사용자 지정 이미지 사용에 대한 라이선스 정보](#)

### 사용자 지정 Python 버전 사용

다른 버전의 Python을 사용하도록 사용자 지정 이미지를 빌드할 수 있습니다. 예를 들어, Spark 작업에 대해 Python 버전 3.10을 사용하려면 다음 명령을 실행합니다.

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

# install python 3
RUN yum install -y gcc openssl-devel bzip2-devel libffi-devel tar gzip wget make
RUN wget https://www.python.org/ftp/python/3.10.0/Python-3.10.0.tgz && \
tar xzf Python-3.10.0.tgz && cd Python-3.10.0 && \
./configure --enable-optimizations && \
make altinstall

# EMRS runs the image as hadoop
USER hadoop:hadoop
```

Spark 작업을 제출하기 전에 다음과 같이 Python 가상 환경을 사용하도록 속성을 설정합니다.

```
--conf spark.emr-serverless.driverEnv.PYSPARK_DRIVER_PYTHON=/usr/local/bin/python3.10
```

```
--conf spark.emr-serverless.driverEnv.PYSPARK_PYTHON=/usr/local/bin/python3.10
--conf spark.executorEnv.PYSPARK_PYTHON=/usr/local/bin/python3.10
```

## 사용자 지정 Java 버전 사용

다음 예제에서는 Spark 작업에 Java 11을 사용하도록 사용자 지정 이미지를 빌드하는 방법을 보여줍니다.

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

# install JDK 11
RUN amazon-linux-extras install java-openjdk11

# EMRS runs the image as hadoop
USER hadoop:hadoop
```

Spark 작업을 제출하기 전에 다음과 같이 Java 11을 사용하도록 Spark 속성을 설정합니다.

```
--conf spark.executorEnv.JAVA_HOME=/usr/lib/jvm/java-11-
openjdk-11.0.16.0.8-1.amzn2.0.1.x86_64
--conf spark.emr-serverless.driverEnv.JAVA_HOME=/usr/lib/jvm/java-11-
openjdk-11.0.16.0.8-
```

## 데이터 과학 이미지 빌드

다음 예제에서는 Pandas 및 NumPy와 같은 일반적인 데이터 과학 Python 패키지를 포함하는 방법을 보여줍니다.

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

# python packages
RUN pip3 install boto3 pandas numpy
RUN pip3 install -U scikit-learn==0.23.2 scipy
RUN pip3 install sk-dist
RUN pip3 install xgboost
```

```
# EMR Serverless runs the image as hadoop
USER hadoop:hadoop
```

## Apache Sedona를 사용한 지리 공간 데이터 처리

다음 예제에서는 지리 공간 처리를 위해 Apache Sedona를 포함하도록 이미지를 빌드하는 방법을 보여줍니다.

```
FROM public.ecr.aws/emr-serverless/spark/emr-6.9.0:latest

USER root

RUN yum install -y wget
RUN wget https://repo1.maven.org/maven2/org/apache/sedona/sedona-core-3.0_2.12/1.3.0-incubating/sedona-core-3.0_2.12-1.3.0-incubating.jar -P /usr/lib/spark/jars/
RUN pip3 install apache-sedona

# EMRS runs the image as hadoop
USER hadoop:hadoop
```

## 사용자 지정 이미지 사용에 대한 라이선스 정보

EMR Serverless를 사용하여 사용자 지정 이미지를 빌드하여 특정 작업을 수행하거나 특정 버전의 소프트웨어 패키지를 사용할 수 있습니다. 사용자 지정 이미지의 수정 및 배포에는 규칙 및 라이선스 약관이 적용될 수 있습니다. 라이선스 텍스트는 이후의 하위 섹션에 표시됩니다.

### 사용자 지정 이미지에 적용되는 라이선스

Copyright Amazon.com 및 그 계열사, 모든 권리 보유. 이 소프트웨어는 [AWS 고객 계약에](#) 따른 AWS 콘텐츠이며 권한 없이 배포할 수 없습니다. AWS 라이선스 허여자는 [AWS 지적 재산 라이선스](#)의 권한 외에도 다음 추가 권한을 부여합니다.

다음 조건이 충족되는 경우 AWS 콘텐츠의 파생물을 생성, 복사 및 사용할 수 있습니다.

- AWS 사용자는 콘텐츠 자체를 수정하지 않으며, 파생물은 전적으로 사용자가 새 콘텐츠를 추가한 결과입니다.
- 내부 복제본은 위의 저작권 고지를 유지합니다.
- 수정 여부에 관계없이 소스 또는 바이너리 형식의 외부 배포는 본 라이선스의 약관에 따라 허용되지 않습니다.

사용자 지정 이미지 사용에 대한 자세한 내용은 [EMR Serverless에서 사용자 지정 이미지 사용](#)을 참조하세요.

## Amazon EMR Serverless에서 Apache Spark용 Amazon Redshift 통합 사용

Amazon EMR 릴리스 6.9.0 이상에서 모든 릴리스 이미지에 [Apache Spark](#)와 Amazon Redshift 간 커넥터가 포함됩니다. 이 커넥터를 사용하면 Amazon EMR Serverless에서 Spark를 사용하여 Amazon Redshift에 저장된 데이터를 처리할 수 있습니다. 통합은 [spark-redshift 오픈 소스 커넥터](#)를 기반으로 합니다. Amazon EMR Serverless의 경우 [Apache Spark용 Amazon Redshift 통합](#)이 기본 통합으로 포함됩니다.

### 주제

- [Apache Spark용 Amazon Redshift 통합을 사용하여 Spark 애플리케이션 시작](#)
- [Apache Spark용 Amazon Redshift 통합으로 인증](#)
- [Amazon Redshift에서 읽고 쓰기](#)
- [Spark 커넥터 사용 시 고려 사항 및 제한 사항](#)

## Apache Spark용 Amazon Redshift 통합을 사용하여 Spark 애플리케이션 시작

EMR Serverless 6.9.0과의 통합을 사용하려면 필수 Spark-Redshift 종속 항목을 Spark 작업과 함께 전달합니다. Redshift 커넥터 관련 라이브러리를 포함하려면 `--jars`를 사용합니다. `--jars` 옵션에서 지원하는 다른 파일 위치에 액세스하려면 Apache Spark 설명서에서 [Advanced Dependency Management](#) 섹션을 참조하세요.

- `spark-redshift.jar`
- `spark-avro.jar`
- `RedshiftJDBC.jar`
- `minimal-json.jar`

Amazon EMR 릴리스 6.10.0 이상에서는 `minimal-json.jar` 종속성이 필요하지 않으며 기본적으로 다른 종속성을 각 클러스터에 자동으로 설치합니다. 다음 예제에서는 Apache Spark용 Amazon Redshift 통합을 사용하여 Spark 애플리케이션을 시작하는 방법을 보여줍니다.

## Amazon EMR 6.10.0 +

EMR Serverless 릴리스 6.10.0 이상에서 Apache Spark용 Amazon Redshift 통합을 사용해 Amazon EMR Serverless에서 Spark 작업을 시작합니다.

```
spark-submit my_script.py
```

## Amazon EMR 6.9.0

EMR Serverless 릴리스 6.9.0에서 Apache Spark에 대한 Amazon Redshift 통합을 통해 Amazon EMR Serverless에서 Spark 작업을 시작하려면 다음 예제와 같이 `--jars` 옵션을 사용합니다. `--jars` 옵션과 함께 나열된 경로는 JAR 파일의 기본 경로입니다.

```
--jars
  /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,
  /usr/share/aws/redshift/spark-redshift/lib/spark-redshift.jar,
  /usr/share/aws/redshift/spark-redshift/lib/spark-avro.jar,
  /usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar
```

```
spark-submit \
  --jars /usr/share/aws/redshift/jdbc/RedshiftJDBC.jar,/usr/share/aws/redshift/
spark-redshift/lib/spark-redshift.jar,/usr/share/aws/redshift/spark-redshift/lib/
spark-avro.jar,/usr/share/aws/redshift/spark-redshift/lib/minimal-json.jar \
  my_script.py
```

## Apache Spark용 Amazon Redshift 통합으로 인증

보안 인증을 검색하고 Amazon Redshift에 연결하는 데 AWS Secrets Manager를 사용합니다.

자격 증명을 Secrets Manager에 저장하여 Amazon Redshift에 대해 안전하게 인증한 후 Spark 작업에서 `GetSecretValue` API를 직접 호출하여 가져올 수 있습니다.

```
from pyspark.sql import SQLContextimport boto3

sc = # existing SparkContext
sql_context = SQLContext(sc)
```

```
secretsmanager_client = boto3.client('secretsmanager',
    region_name=os.getenv('AWS_REGION'))
secret_manager_response = secretsmanager_client.get_secret_value(
    SecretId='string',
    VersionId='string',
    VersionStage='string'
)
username = # get username from secret_manager_response
password = # get password from secret_manager_response
url = "jdbc:redshift://redshifthost:5439/database?user=" + username + "&password="
    + password

# Access to Redshift cluster using Spark
```

## JDBC 드라이버를 사용하여 Amazon Redshift에 인증

### JDBC URL에서 사용자 이름 및 암호 설정

JDBC URL에서 Amazon Redshift 데이터베이스 이름과 암호를 지정하여 Amazon Redshift 클러스터에 대해 Spark 작업을 인증할 수 있습니다.

#### Note

URL에 데이터베이스 보안 인증을 전달하면 URL에 액세스할 수 있는 모든 사용자도 보안 인증에 액세스할 수 있습니다. 이 방법은 안전한 옵션이 아니므로 일반적으로 권장되지 않습니다.

애플리케이션의 보안이 문제가 아닌 경우 다음 형식을 사용하여 JDBC URL에서 사용자 이름과 암호를 설정할 수 있습니다.

```
jdbc:redshift://redshifthost:5439/database?user=username&password=password
```

## Amazon EMR Serverless 작업 실행 역할로 IAM 기반 인증 사용

Amazon EMR Serverless 릴리스 6.9.0부터 Amazon Redshift JDBC 드라이버 버전 2.1 이상이 환경에 함께 패키지로 제공됩니다. JDBC 드라이버 2.1 이상을 사용하면 JDBC URL을 지정하고 원시 사용자 이름과 암호는 포함하지 않을 수 있습니다.

대신, `jdbc:redshift:iam://` 스키마를 지정합니다. 이를 통해 EMR Serverless 작업 실행 역할을 사용하여 자격 증명을 자동으로 가져오도록 JDBC 드라이버에 지시합니다. 자세한 내용은 Amazon

Redshift 관리 안내서에서 [IAM 보안 인증을 사용하도록 JDBC 또는 ODBC 연결 구성](#)을 참조하세요. 이 URL의 예제는 다음과 같습니다.

```
jdbc:redshift:iam://examplecluster.abc123xyz789.us-west-2.redshift.amazonaws.com:5439/  
dev
```

제공된 조건을 충족하는 경우 작업 실행 역할에 다음 권한이 필요합니다.

권한	작업 실행 역할에 필요한 경우 조건
redshift:GetClusterCredentials	JDBC 드라이버가 Amazon Redshift에서 보안 인증을 가져오는 데 필요함
redshift:DescribeCluster	Amazon Redshift 클러스터를 지정하고 엔드포인트 대신 JDBC URL에 AWS 리전을 지정하는 경우 필요함
redshift-serverless:GetCredentials	JDBC 드라이버가 Amazon Redshift Serverless에서 보안 인증을 가져오는 데 필요함
redshift-serverless:GetWorkgroup	Amazon Redshift Serverless를 사용하고 있고 작업 그룹 이름 및 리전 측면에서 URL을 지정하는 경우 필요함

## 다른 VPC 내에서 Amazon Redshift에 연결

VPC에서 프로비저닝된 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹을 설정하는 경우 Amazon EMR Serverless 애플리케이션이 리소스에 액세스하도록 VPC 연결을 구성합니다. EMR Serverless 애플리케이션에서 VPC 연결을 구성하는 방법에 대한 자세한 내용은 [데이터에 연결하도록 EMR Serverless 애플리케이션에 대한 VPC 액세스 구성](#) 섹션을 참조하세요.

- 프로비저닝된 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹에 대한 퍼블릭 액세스가 가능한 경우 EMR Serverless 애플리케이션을 생성할 때 NAT 게이트웨이가 연결된 하나 이상의 프라이빗 서브넷을 지정할 수 있습니다.
- 프로비저닝된 Amazon Redshift 클러스터 또는 Amazon Redshift Serverless 작업 그룹에 대한 퍼블릭 액세스가 불가능한 경우 [데이터에 연결하도록 EMR Serverless 애플리케이션에 대한 VPC 액세스 구성](#)에 설명된 대로 Amazon Redshift 클러스터에 대한 Amazon Redshift 관리형 VPC 엔드포인트를 생성해야 합니다. 또는 Amazon Redshift 관리 안내서의 [Amazon Redshift Serverless에 연결](#)에 설명된 대로 Amazon Redshift Serverless 작업 그룹을 생성할 수 있습니다. 클러스터 또는 하위 그룹

을 EMR Serverless 애플리케이션을 생성하는 경우 사용자가 지정한 프라이빗 서브넷에 연결해야 합니다.

#### Note

IAM 기반 인증을 사용하고 EMR Serverless 애플리케이션의 프라이빗 서브넷에 연결된 NAT 게이트웨이가 없는 경우 Amazon Redshift 또는 Amazon Redshift Serverless의 해당 서브넷에서도 VPC 엔드포인트를 생성해야 합니다. 이렇게 하면 JDBC 드라이버가 자격 증명을 가져올 수 있습니다.

## Amazon Redshift에서 읽고 쓰기

다음 코드 예제는 데이터 소스 API와 SparkSQL을 통해 Amazon Redshift 데이터베이스에서 샘플 데이터를 읽고 쓰는 데 PySpark를 사용합니다.

### Data source API

PySpark를 사용하여 데이터 소스 API를 통해 Amazon Redshift 데이터베이스에서 샘플 데이터를 읽고 씁니다.

```
import boto3
from pyspark.sql import SQLContext

sc = # existing SparkContext
sql_context = SQLContext(sc)

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::account-id:role/role-name"

df = sql_context.read \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
    .option("dbtable", "table-name") \
    .option("tempdir", "s3://path/for/temp/data") \
    .option("aws_iam_role", "aws-iam-role-arn") \
    .load()

df.write \
    .format("io.github.spark_redshift_community.spark.redshift") \
    .option("url", url) \
```

```
.option("dbtable", "table-name-copy") \
.option("tempdir", "s3://path/for/temp/data") \
.option("aws_iam_role", "aws-iam-role-arn") \
.mode("error") \
.save()
```

## SparkSQL

PySpark를 사용하여 SparkSQL을 통해 Amazon Redshift 데이터베이스에서 샘플 데이터를 읽고 씁니다.

```
import boto3
import json
import sys
import os
from pyspark.sql import SparkSession

spark = SparkSession \
    .builder \
    .enableHiveSupport() \
    .getOrCreate()

url = "jdbc:redshift:iam://redshifthost:5439/database"
aws_iam_role_arn = "arn:aws:iam::account-id:role/role-name"

bucket = "s3://path/for/temp/data"
tableName = "table-name" # Redshift table name

s = f"""CREATE TABLE IF NOT EXISTS {table-name} (country string, data string)
    USING io.github.spark_redshift_community.spark.redshift
    OPTIONS (dbtable '{table-name}', tempdir '{bucket}', url '{url}', aws_iam_role
    '{aws-iam-role-arn}' ); """

spark.sql(s)

columns = ["country" ,"data"]
data = [("test-country", "test-data") ]
df = spark.sparkContext.parallelize(data).toDF(columns)

# Insert data into table
df.write.insertInto(table-name, overwrite=False)
df = spark.sql(f"SELECT * FROM {table-name}")
df.show()
```

## Spark 커넥터 사용 시 고려 사항 및 제한 사항

- Amazon EMR의 Spark에서 Amazon Redshift로의 JDBC 연결을 위해 SSL을 켜는 것이 좋습니다.
- 모범 사례로 AWS Secrets Manager에서 Amazon Redshift 클러스터의 보안 인증을 관리하는 것이 좋습니다. 예제로 [Using AWS Secrets Manager to retrieve credentials for connecting to Amazon Redshift](#)를 참조하세요.
- Amazon Redshift 인증 파라미터에 대해 `aws_iam_role` 파라미터를 사용하여 IAM 역할을 전달하는 것이 좋습니다.
- 현재 `tempformat` 파라미터는 Parquet 형식을 지원하지 않습니다.
- `tempdir` URI는 Amazon S3 위치를 가리킵니다. 이 임시 디렉터리는 자동으로 정리되지 않으므로, 추가 비용이 발생할 수 있습니다.
- Amazon Redshift에 대한 다음 권장 사항을 고려합니다.
  - Amazon Redshift 클러스터에 대한 퍼블릭 액세스를 차단하는 것이 좋습니다.
  - [Amazon Redshift 감사 로깅](#)을 켜는 것이 좋습니다.
  - [Amazon Redshift 저장 데이터 암호화](#)를 켜는 것이 좋습니다.
- Amazon S3에 대한 다음 권장 사항을 고려합니다.
  - [Amazon S3 버킷에 대한 퍼블릭 액세스를 차단](#)하는 것이 좋습니다.
  - [Amazon S3 서버 측 암호화](#)를 사용하여 사용된 Amazon S3 버킷을 암호화하는 것이 좋습니다.
  - [Amazon S3 수명 주기 정책](#)을 사용하여 Amazon S3 버킷에 대한 보존 규칙을 정의하는 것이 좋습니다.
- Amazon EMR은 오픈 소스에서 이미지로 가져온 코드를 항상 확인합니다. 보안을 위해 Spark에서 Amazon S3로의 다음 인증 방법은 지원되지 않습니다.
  - `hadoop-env` 구성 분류에서 AWS 액세스 키 설정
  - `tempdir` URI에서 AWS 액세스 키 인코딩

커넥터 사용 및 지원되는 파라미터에 대한 자세한 내용은 다음 리소스를 참조하세요.

- Amazon Redshift 관리 안내서의 [Apache Spark용 Amazon Redshift 통합](#)
- Github의 [spark-redshift community repository](#)

# Amazon EMR Serverless를 사용하여 DynamoDB에 연결

이 자습서에서는 [미국 지명위원회](#)에서 Amazon S3 버킷에 데이터 하위 세트를 업로드한 다음, Amazon EMR Serverless에서 Hive 또는 Spark를 사용하여 쿼리할 수 있는 Amazon DynamoDB 테이블로 데이터를 복사합니다.

## 1단계: Amazon S3 버킷에 데이터 업로드

Amazon S3 버킷을 생성하려면 Amazon Simple Storage Service 콘솔 사용 설명서의 [버킷 생성](#)에 나온 지침을 따르세요. `amzn-s3-demo-bucket`에 대한 참조를 새로 생성된 버킷의 이름으로 바꿉니다. 이제 EMR Serverless 애플리케이션에서 작업을 실행할 준비가 되었습니다.

1. 다음 명령을 사용하여 샘플 데이터 아카이브(features.zip)를 다운로드합니다.

```
wget https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/samples/features.zip
```

2. 아카이브에서 features.txt 파일을 추출하고 파일의 처음 몇 줄을 확인합니다.

```
unzip features.zip
head features.txt
```

결과는 다음과 비슷합니다.

```
1535908|Big Run|Stream|WV|38.6370428|-80.8595469|794
875609|Constable Hook|Cape|NJ|40.657881|-74.0990309|7
1217998|Gooseberry Island|Island|RI|41.4534361|-71.3253284|10
26603|Boone Moore Spring|Spring|AZ|34.0895692|-111.410065|3681
1506738|Missouri Flat|Flat|WA|46.7634987|-117.0346113|2605
1181348|Minnow Run|Stream|PA|40.0820178|-79.3800349|1558
1288759|Hunting Creek|Stream|TN|36.343969|-83.8029682|1024
533060|Big Charles Bayou|Bay|LA|29.6046517|-91.9828654|0
829689|Greenwood Creek|Stream|NE|41.596086|-103.0499296|3671
541692|Button Willow Island|Island|LA|31.9579389|-93.0648847|98
```

여기에 나온 각 줄의 필드는 고유 식별자, 이름, 자연 특성 유형, 주, 위도, 경도, 고도(피트)를 나타냅니다.

3. Amazon S3에 데이터 업로드

```
aws s3 cp features.txt s3://amzn-s3-demo-bucket/features/
```

## 2단계: Hive 테이블 생성

Apache Spark 또는 Hive를 사용하여 Amazon S3에 업로드된 데이터가 포함된 새 Hive 테이블을 생성합니다.

### Spark

Spark에서 Hive 테이블을 생성하려면 다음 명령을 실행합니다.

```
import org.apache.spark.sql.SparkSession

val sparkSession = SparkSession.builder().enableHiveSupport().getOrCreate()

sparkSession.sql("CREATE TABLE hive_features \
  (feature_id BIGINT, \
  feature_name STRING, \
  feature_class STRING, \
  state_alpha STRING, \
  prim_lat_dec DOUBLE, \
  prim_long_dec DOUBLE, \
  elev_in_ft BIGINT) \
  ROW FORMAT DELIMITED \
  FIELDS TERMINATED BY '|' \
  LINES TERMINATED BY '\n' \
  LOCATION 's3://amzn-s3-demo-bucket/features';")
```

이제 features.txt 파일의 데이터로 고유 Hive 테이블을 채워졌습니다. 데이터가 테이블에 있는지 확인하려면 다음 예제와 같이 Spark SQL 쿼리를 실행합니다.

```
sparkSession.sql(
  "SELECT state_alpha, COUNT(*) FROM hive_features GROUP BY state_alpha;")
```

### Hive

Hive에서 Hive 테이블을 생성하려면 다음 명령을 실행합니다.

```
CREATE TABLE hive_features
  (feature_id          BIGINT,
```

```
feature_name      STRING ,
feature_class     STRING ,
state_alpha       STRING,
prim_lat_dec      DOUBLE ,
prim_long_dec     DOUBLE ,
elev_in_ft        BIGINT)
ROW FORMAT DELIMITED
FIELDS TERMINATED BY '|'
LINES TERMINATED BY '\n'
LOCATION 's3://amzn-s3-demo-bucket/features';
```

이제 `features.txt` 파일의 데이터를 포함하는 Hive 테이블이 있습니다. 데이터가 테이블에 있는지 확인하려면 다음 예제와 같이 HiveQL 쿼리를 실행합니다.

```
SELECT state_alpha, COUNT(*) FROM hive_features GROUP BY state_alpha;
```

### 3단계: DynamoDB로 데이터 복사

Spark 또는 Hive를 사용하여 데이터를 새 DynamoDB 테이블에 복사합니다.

#### Spark

이전 단계에서 생성한 Hive 테이블의 데이터를 DynamoDB로 복사하려면 [DynamoDB로 데이터 복사](#)의 1~3단계를 수행합니다. 그러면 새 DynamoDB 테이블(Features)이 생성됩니다. 그리고 다음 예제와 같이 텍스트 파일에서 직접 데이터를 읽고 DynamoDB 테이블에 복사할 수 있습니다.

```
import com.amazonaws.services.dynamodbv2.model.AttributeValue
import org.apache.hadoop.dynamodb.DynamoDBItemWritable
import org.apache.hadoop.dynamodb.read.DynamoDBInputFormat
import org.apache.hadoop.io.Text
import org.apache.hadoop.mapred.JobConf
import org.apache.spark.SparkContext

import scala.collection.JavaConverters._

object EmrServerlessDynamoDbTest {

  def main(args: Array[String]): Unit = {

    jobConf.set("dynamodb.input.tableName", "Features")
    jobConf.set("dynamodb.output.tableName", "Features")
```

```

    jobConf.set("dynamodb.region", "region")

    jobConf.set("mapred.output.format.class",
"org.apache.hadoop.dynamodb.write.DynamoDBOutputFormat")
    jobConf.set("mapred.input.format.class",
"org.apache.hadoop.dynamodb.read.DynamoDBInputFormat")

    val rdd = sc.textFile("s3://amzn-s3-demo-bucket/ddb-connector/")
    .map(row => {
        val line = row.split("\\|")
        val item = new DynamoDBItemWritable()

        val elevInFt = if (line.length > 6) {
            new AttributeValue().withN(line(6))
        } else {
            new AttributeValue().withNULL(true)
        }

        item.setItem(Map(
            "feature_id" -> new AttributeValue().withN(line(0)),
            "feature_name" -> new AttributeValue(line(1)),
            "feature_class" -> new AttributeValue(line(2)),
            "state_alpha" -> new AttributeValue(line(3)),
            "prim_lat_dec" -> new AttributeValue().withN(line(4)),
            "prim_long_dec" -> new AttributeValue().withN(line(5)),
            "elev_in_ft" -> elevInFt)
            .asJava)
            (new Text(""), item)
        })
    rdd.saveAsHadoopDataset(jobConf)
}

```

## Hive

이전 단계에서 생성한 Hive 테이블의 데이터를 DynamoDB로 복사하려면 [DynamoDB로 데이터 복사](#)의 지침을 따릅니다.

## 4단계: DynamoDB에서 데이터 쿼리

Spark 또는 Hive를 사용하여 DynamoDB 테이블을 쿼리합니다.

## Spark

이전 단계에서 생성한 DynamoDB 테이블에서 데이터를 쿼리하기 위해 Spark SQL 또는 Spark MapReduce API를 사용할 수 있습니다.

Example - Spark SQL을 사용하여 DynamoDB 테이블 쿼리

다음 Spark SQL 쿼리는 모든 기능 유형의 목록을 사전순으로 반환합니다.

```
val dataframe = sparkSession.sql("SELECT DISTINCT feature_class \
FROM ddb_features \
ORDER BY feature_class;")
```

다음 Spark SQL 쿼리는 문자 M으로 시작하는 모든 레이크의 목록을 반환합니다.

```
val dataframe = sparkSession.sql("SELECT feature_name, state_alpha \
FROM ddb_features \
WHERE feature_class = 'Lake' \
AND feature_name LIKE 'M%' \
ORDER BY feature_name;")
```

다음 Spark SQL 쿼리는 1마일보다 큰 세 개 이상의 기능이 있는 모든 상태 목록을 반환합니다.

```
val dataframe = sparkSession.dql("SELECT state_alpha, feature_class, COUNT(*) \
FROM ddb_features \
WHERE elev_in_ft > 5280 \
GROUP by state_alpha, feature_class \
HAVING COUNT(*) >= 3 \
ORDER BY state_alpha, feature_class;")
```

Example - Spark MapReduce API를 사용하여 DynamoDB 테이블 쿼리

다음 MapReduce 쿼리는 모든 기능 유형의 목록을 사전순으로 반환합니다.

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
  classOf[DynamoDBItemWritable])
  .map(pair => (pair._1, pair._2.getItem))
  .map(pair => pair._2.get("feature_class").getS)
  .distinct()
  .sortBy(value => value)
  .toDF("feature_class")
```

다음 MapReduce 쿼리는 문자 M으로 시작하는 모든 레이크의 목록을 반환합니다.

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
  classOf[DynamoDBItemWritable])
  .map(pair => (pair._1, pair._2.getItem))
  .filter(pair => "Lake".equals(pair._2.get("feature_class").getS))
  .filter(pair => pair._2.get("feature_name").getS.startsWith("M"))
  .map(pair => (pair._2.get("feature_name").getS,
    pair._2.get("state_alpha").getS))
  .sortBy(_._1)
  .toDF("feature_name", "state_alpha")
```

다음 MapReduce 쿼리는 1마일보다 큰 세 개 이상의 기능이 있는 모든 상태 목록을 반환합니다.

```
val df = sc.hadoopRDD(jobConf, classOf[DynamoDBInputFormat], classOf[Text],
  classOf[DynamoDBItemWritable])
  .map(pair => pair._2.getItem)
  .filter(pair => pair.get("elev_in_ft").getN != null)
  .filter(pair => Integer.parseInt(pair.get("elev_in_ft").getN) > 5280)
  .groupBy(pair => (pair.get("state_alpha").getS, pair.get("feature_class").getS))
  .filter(pair => pair._2.size >= 3)
  .map(pair => (pair._1._1, pair._1._2, pair._2.size))
  .sortBy(pair => (pair._1, pair._2))
  .toDF("state_alpha", "feature_class", "count")
```

## Hive

이전 단계에서 생성한 DynamoDB 테이블에서 데이터를 쿼리하려면 [DynamoDB 테이블에서 데이터 쿼리](#)의 지침을 따릅니다.

## 교차 계정 액세스 설정

에서 교차 계정 액세스를 설정하려면 다음 단계를 수행합니다. 예제에서 AccountA는 Amazon EMR Serverless 애플리케이션을 생성한 계정이고 AccountB는 Amazon DynamoDB가 위치한 계정입니다.

1. AccountB에서 DynamoDB 테이블을 생성합니다. 자세한 내용은 [1단계: 테이블 생성](#)을 참조하세요.
2. AccountB에서 DynamoDB 테이블에 액세스할 수 있는 Cross-Account-Role-B IAM 역할을 생성합니다.

- a. AWS Management 콘솔에 로그인하여 <https://console.aws.amazon.com/iam/> 에서 IAM 콘솔을 엽니다.
- b. 역할을 선택하고 새 역할(Cross-Account-Role-B)을 생성합니다. IAM 역할 생성에 대한 자세한 내용은 사용 설명서에서 [IAM 역할 생성](#)을 참조하세요.
- c. 교차 계정 DynamoDB 테이블에 액세스할 수 있는 권한을 부여하는 IAM 정책을 생성합니다. 그런 다음, IAM 정책을 Cross-Account-Role-B에 연결합니다.

다음은 DynamoDB 테이블 CrossAccountTable에 대한 액세스 권한을 부여하는 정책입니다.

- d. Cross-Account-Role-B 역할에 대한 신뢰 관계를 편집합니다.

역할에 대한 신뢰 관계를 구성하려면 2단계: Cross-Account-Role-B에서 생성한 역할에 대해 IAM 콘솔에서 신뢰 관계 탭을 선택합니다.

신뢰 관계 편집을 선택하고 다음 정책 문서를 추가합니다. 이 문서에서는 AccountA의 Job-Execution-Role-A에서 이 Cross-Account-Role-B 역할을 수임하도록 허용합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/Job-Execution-Role-A",
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

- e. Cross-Account-Role-B를 수임할 수 있는 - STS Assume role 권한을 AccountA의 Job-Execution-Role-A에 부여합니다.

AWS 계정 AccountA에 대한 IAM 콘솔에서 Job-Execution-Role-A를 선택합니다. 다음 정책 명령을 Job-Execution-Role-A에 추가하여 Cross-Account-Role-B 역할에서 AssumeRole 작업을 허용합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": [
        "arn:aws:iam::123456789012:role/Cross-Account-Role-B"
      ],
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

- f. 코어 사이트 분류에서 값이 `com.amazonaws.emr.AssumeRoleAWSCredentialsProvider`인 `dynamodb.customAWSCredentialsProvider` 속성을 설정합니다. `ASSUME_ROLE_CREDENTIALS_ROLE_ARN` 환경 변수를 `Cross-Account-Role-B`의 ARN 값으로 설정합니다.
3. `Job-Execution-Role-A`를 사용하여 Spark 또는 Hive 작업을 실행합니다.

## 고려 사항

DynamoDB 커넥터를 Apache Spark 또는 Apache Hive와 함께 사용하는 경우 이러한 동작 및 제한 사항에 유의합니다.

## Apache Spark와 함께 DynamoDB 커넥터 사용 시 고려 사항

- Spark SQL은 스토리지 핸들러 옵션을 사용하는 Hive 테이블 생성을 지원하지 않습니다. 자세한 내용은 Apache Spark 설명서의 [Specifying storage format for Hive tables](#)를 참조하세요.
- Spark SQL은 스토리지 핸들러를 사용하는 `STORED BY` 작업을 지원하지 않습니다. 외부 Hive 테이블을 통해 DynamoDB 테이블과 상호 작용하려면 먼저 Hive를 사용하여 테이블을 생성합니다.
- 쿼리를 DynamoDB 쿼리로 변환하기 위해 DynamoDB 커넥터는 조건자 푸시다운을 사용합니다. 조건자 푸시다운은 DynamoDB 테이블의 파티션 키에 매핑된 열을 기준으로 데이터를 필터링합니다.

조건자 푸시다운은 MapReduce API에서가 아니라 Spark SQL에서 커넥터를 사용하는 경우에만 작동합니다.

## Apache Hive와 함께 DynamoDB 커넥터 사용 시 고려 사항

### 최대 매퍼 수 조정

- SELECT 쿼리를 사용하여 DynamoDB에 매핑되는 외부 Hive 테이블에서 데이터를 읽는 경우 EMR Serverless의 맵 태스크 수는 DynamoDB 테이블에 대해 구성된 총 읽기 처리량을 맵 태스크당 처리량으로 나눈 값으로 계산됩니다. 맵 작업당 기본 처리량은 100입니다.
- Hive 작업은 DynamoDB에 대해 구성된 읽기 처리량에 따라 EMR Serverless 애플리케이션당 구성된 최대 컨테이너 수를 초과하는 맵 태스크 수를 사용할 수 있습니다. 또한 장기 실행 Hive 쿼리는 DynamoDB 테이블의 프로비저닝된 모든 읽기 용량을 소비할 수 있습니다. 이는 다른 사용자에게 부정적인 영향을 미칩니다.
- `dynamodb.max.map.tasks` 속성을 사용하여 맵 태스크의 상한을 설정할 수 있습니다. 또한 이 속성을 사용하여 태스크 컨테이너 크기에 따라 각 맵 태스크에서 읽는 데이터의 양을 조정할 수 있습니다.
- `dynamodb.max.map.tasks` 속성을 Hive 쿼리 수준 또는 `start-job-run` 명령의 `hive-site` 분류에서 설정할 수 있습니다. 이 값은 1보다 크거나 같아야 합니다. Hive에서 쿼리를 처리하는 경우 DynamoDB 테이블에서 데이터를 읽으면 결과 Hive 작업은 `dynamodb.max.map.tasks`의 값보다 더 많이 사용하지 않습니다.

### 태스크당 쓰기 처리량 조정

- EMR Serverless의 태스크당 쓰기 처리량은 DynamoDB 테이블에 대해 구성된 총 쓰기 처리량을 `mapreduce.job.maps` 속성 값으로 나눈 값으로 계산됩니다. Hive의 경우 이 속성의 기본값은 2입니다. 따라서 Hive 작업의 마지막 단계에서 처음 두 태스크는 모든 쓰기 처리량을 소비할 수 있습니다. 이로 인해 동일한 작업 또는 다른 작업에서 다른 태스크의 쓰기가 스로틀링됩니다.
- 쓰기 스로틀링을 방지하기 위해 최종 단계의 태스크 수 또는 태스크당 할당하려는 쓰기 처리량을 기반으로 `mapreduce.job.maps` 속성 값을 설정할 수 있습니다. EMR Serverless에서 `start-job-run` 명령의 `mapred-site` 분류에서 이 속성을 설정합니다.

# 보안

AWS는 클라우드 보안을 가장 중요하게 생각합니다. AWS 고객으로서 여러분은 가장 높은 보안 요구 사항을 충족하기 위해 설계된 데이터 센터 및 네트워크 아키텍처의 혜택을 받게 됩니다.

보안은 AWS와 사용자의 공동 책임입니다. [공동 책임 모델](#)은 이 사항을 클라우드 내 보안 및 클라우드의 보안으로 설명합니다.

- 클라우드의 보안: AWS는 AWS Cloud에서 AWS 서비스를 실행하는 인프라를 보호할 책임이 있습니다. AWS는 안전하게 사용할 수 있는 서비스도 제공합니다. 서드 파티 감사원은 정기적으로 [AWS 규정 준수 프로그램](#)의 일환으로 보안 효과를 테스트하고 검증합니다. Amazon EMR Serverless에 적용되는 규정 준수 프로그램에 대한 자세한 내용은 [규정 준수 프로그램 제공 범위 내 AWS 서비스](#)를 참조하세요.
- 클라우드 내 보안 – 귀하의 책임은 귀하가 사용하는 AWS 서비스에 의해 결정됩니다. 또한 귀하는 데이터의 민감도, 회사 요구 사항, 관련 법률 및 규정을 비롯한 기타 요소에 대해서도 책임이 있습니다.

이 설명서는 Amazon EMR Serverless를 사용할 때 공동 책임 모델을 적용하는 방법을 이해하는 데 도움이 됩니다. 각 항목에서는 Amazon EMR Serverless 구성 방법과 다른 AWS 서비스를 사용해 보안 및 규정 준수 목표를 충족하는 방법을 안내합니다.

## 주제

- [Amazon EMR Serverless의 보안 모범 사례](#)
- [데이터 보호](#)
- [Amazon EMR Serverless의 Identity and Access Management\(IAM\)](#)
- [Amazon EMR Serverless를 사용한 신뢰할 수 있는 자격 증명 전파](#)
- [EMR Serverless와 함께 Lake Formation 사용](#)
- [작업자 간 암호화](#)
- [EMR Serverless를 사용하는 데이터 보호를 위한 Secrets Manager](#)
- [EMR Serverless에서 Amazon S3 Access Grants 사용](#)
- [AWS CloudTrail을 사용하여 Amazon EMR Serverless API 직접 호출 로깅](#)
- [Amazon EMR Serverless에 대한 규정 준수 확인](#)
- [Amazon EMR Serverless의 복원력](#)
- [Amazon EMR Serverless의 인프라 보안](#)
- [Amazon EMR Serverless에서 구성 및 취약성 분석](#)

## Amazon EMR Serverless의 보안 모범 사례

Amazon EMR Serverless는 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 권한의 원칙 적용

Amazon EMR Serverless는 실행 역할과 같은 IAM 역할을 사용하는 애플리케이션에 대한 세분화된 액세스 정책을 제공합니다. 애플리케이션 조치와 로그 대상에 대한 액세스 등 작업에 필요한 최소한의 권한만 실행 역할에 부여하는 것이 좋습니다. 또한 애플리케이션 코드가 변경될 때마다 정기적으로 권한을 확인하기 위해 작업을 감사하는 것이 좋습니다.

### 신뢰할 수 없는 애플리케이션 코드 격리

EMR Serverless는 서로 다른 EMR Serverless 애플리케이션에 속한 작업 사이에서 완전한 네트워크 격리를 생성합니다. 작업 수준 격리를 원하는 경우 작업을 서로 다른 EMR Serverless 애플리케이션으로 격리하는 방법을 고려합니다.

### 역할 기반 액세스 제어(RBAC) 권한

관리자는 EMR Serverless 애플리케이션에 대한 역할 기반 액세스 제어(RBAC) 권한을 엄격하게 제어해야 합니다.

## 데이터 보호

Amazon EMR Serverless에서 AWS [공동 책임 모델](#)을 데이터 보호에 적용합니다. 이 모델에서 설명하는 것처럼 AWS는(는) 모든 AWS 클라우드를 실행하는 글로벌 인프라를 보호할 책임이 있습니다. 이 인프라에서 호스팅되는 콘텐츠에 대한 통제를 유지하는 것은 사용자의 책임입니다. 이 콘텐츠에는 사용하는 AWS 서비스에 대한 보안 구성 및 관리 태스크가 포함됩니다. 데이터 프라이버시에 대한 자세한 내용은 [데이터 프라이버시 FAQ](#)를 참조하세요. 유럽의 데이터 보호에 대한 자세한 내용은 AWS 보안 블로그에서 [AWS Shared Responsibility Model and GDPR](#) 블로그 게시물을 참조하세요.

데이터를 보호하려면 AWS 계정 보안 인증을 보호하고 AWS Identity and Access Management(IAM)를 사용해 개별 사용자 계정을 설정하는 것이 좋습니다. 이러한 방식에서는 각 사용자에게 자신의 직무를 충실히 이행하는 데 필요한 권한만 부여됩니다. 또한 다음과 같은 방법으로 데이터를 보호하는 것이 좋습니다.

- 각 계정에 다중 인증(MFA)을 사용하세요.
- SSL/TLS를 사용하여 AWS 리소스와 통신하세요. TLS 1.2 이상을 사용하는 것이 좋습니다.
- AWS CloudTrail로 API 및 사용자 활동 로깅을 설정합니다.
- AWS 암호화 솔루션을 AWS 서비스 내의 모든 기본 보안 컨트롤과 함께 사용합니다.
- Amazon S3에 저장된 개인 데이터를 검색하고 보호하는 데 도움이 되는 Amazon Macie와 같은 고급 관리형 보안 서비스를 사용합니다.
- Amazon EMR Serverless 암호화 옵션을 사용해 유틸리티 및 전송 중 데이터를 암호화합니다.
- 명령행 인터페이스 또는 API를 통해 AWS에 액세스할 때 FIPS 140-2 검증된 암호화 모듈이 필요한 경우, FIPS 엔드포인트를 사용합니다. 사용 가능한 FIPS 엔드포인트에 대한 자세한 내용은 [Federal Information Processing Standard\(FIPS\) 140-2](#)를 참조하세요.

이름 필드와 같은 자유 형식 필드에 고객 계정 번호와 같은 중요 식별 정보를 절대 입력하지 마세요. 여기에는 Amazon EMR Serverless 또는 기타 AWS 서비스에서 콘솔, API, AWS CLI 또는 AWS SDK를 사용하여 작업하는 경우가 포함됩니다. Amazon EMR Serverless 또는 기타 서비스에 입력하는 모든 데이터를 진단 로그에 포함할 수 있습니다. 외부 서버에 URL을 제공할 때 해당 서버에 대한 요청을 검증하기 위해 자격 증명 정보를 URL에 포함시키지 마세요.

## 저장 시 암호화

데이터 암호화는 권한 없는 사용자가 클러스터 및 관련 데이터 스토리지 시스템에서 데이터를 읽는 것을 방지하는 데 도움이 됩니다. 여기에는 영구 미디어에 저장된 데이터인 유틸리티 데이터와 네트워크를 이동하는 동안 가로챌 수 있는 데이터인 전송 중 데이터가 포함됩니다.

데이터 암호화에는 키와 인증서가 필요합니다. AWS Key Management Service에서 관리하는 키, Amazon S3에서 관리하는 키, 사용자가 제공하는 사용자 지정 제공업체의 키 및 인증서 등 여러 가지 옵션을 선택할 수 있습니다. AWS KMS를 키 공급자로 사용할 경우 스토리지 및 암호화 키 사용에 요금이 적용됩니다. 자세한 내용은 [AWS KMS 요금](#)을 참조하세요.

암호화 옵션을 지정하기 전에 사용할 키 및 인증서 관리 시스템을 결정합니다. 그런 다음 암호화 설정의 일부로 지정하는 사용자 지정 제공업체에 대한 키와 인증서를 생성합니다.

## Amazon S3에서 EMRFS 데이터에 대한 유틸리티 데이터 암호화

각 EMR Serverless 애플리케이션은 EMR 파일 시스템(EMRFS)을 포함하는 특정 릴리스 버전을 사용합니다. Amazon S3 암호화는 Amazon S3에서 읽고 쓰는 EMR 파일 시스템(EMRFS) 객체와 함께 작동합니다. 유틸리티 데이터 암호화를 활성화하는 경우 Amazon S3 서버 측 암호화(SSE) 또는 클라이언트

트 측 암호화(CSE)를 기본 암호화 모드로 지정할 수 있습니다. 선택적으로 버킷별 암호화 재정의의를 사용하여 개별 버킷에 서로 다른 암호화 방법을 지정할 수 있습니다. Amazon S3 암호화가 활성화되어 있는지 여부와 상관없이 전송 계층 보안(TLS)은 EMR 클러스터 노드 및 Amazon S3 간에 전송 중인 EMRFS 객체를 암호화합니다. Amazon S3 CSE를 고객 관리형 키와 함께 사용하는 경우 EMR Serverless 애플리케이션에서 작업을 실행하는 데 사용되는 실행 역할은 키에 대한 액세스를 보유해야 합니다. Amazon S3 암호화에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [암호화를 사용하여 데이터 보호](#)를 참조하세요.

#### Note

AWS KMS를 사용할 때는 스토리지와 암호화 키 사용에 요금이 적용됩니다. 자세한 내용은 [AWS KMS 요금](#)을 참조하세요.

## Amazon S3 서버 측 암호화

모든 Amazon S3 버킷에는 기본적으로 암호화가 구성되어 있으며, Amazon S3 버킷에 업로드되는 모든 새 객체는 저장 시 자동으로 암호화됩니다. Amazon S3는 데이터를 디스크에 쓸 때 객체 수준에서 데이터를 암호화하고 액세스 시 데이터를 해독합니다. 자세한 내용은 Amazon Simple Storage Service 개발자 안내서의 [서버 측 암호화를 사용하여 데이터 보호](#)를 참조하세요.

Amazon EMR Serverless에서 SSE를 지정하는 경우 다음 두 가지 키 관리 시스템 중에서 선택할 수 있습니다.

- SSE-S3 - Amazon S3에서는 자동으로 키를 관리합니다. EMR Serverless에서는 추가 설정이 필요하지 않습니다.
- SSE-KMS - AWS KMS key를 사용하여 Amazon EMR에 적합한 정책을 설정합니다. EMR Serverless에서는 추가 설정이 필요하지 않습니다.

Amazon S3에 쓰는 데이터에 대해 AWS KMS 암호화를 사용하려면 StartJobRun API를 사용할 때 두 가지 옵션이 있습니다. Amazon S3에 쓰는 모든 항목에 대해 암호화를 활성화하거나 특정 버킷에 쓰는 데이터에 대해 암호화를 활성화할 수 있습니다. StartJobRun API에 대한 자세한 내용은 [EMR Serverless API 참조](#)를 참조하세요.

Amazon S3에 쓰는 모든 데이터에 대해 AWS KMS 암호화를 설정하려면 StartJobRun API를 직접 호출할 때 다음 명령을 사용합니다.

```
--conf spark.hadoop.fs.s3.enableServerSideEncryption=true
```

```
--conf spark.hadoop.fs.s3.serverSideEncryption.kms.keyId=<kms_id>
```

특정 버킷에 쓰는 데이터에 대한 AWS KMS 암호화를 설정하려면 StartJobRun API를 직접 호출할 때 다음 명령을 사용합니다.

```
--conf spark.hadoop.fs.s3.bucket.<amzn-s3-demo-bucket1>.enableServerSideEncryption=true
--conf spark.hadoop.fs.s3.bucket.<amzn-s3-demo-bucket1>.serverSideEncryption.kms.keyId=<kms-id>
```

고객 제공 키(SSE-C)를 사용하는 SSE는 EMR Serverless에서 사용할 수 없습니다.

## Amazon S3 클라이언트 측 암호화

Amazon S3 클라이언트 측 암호화를 사용하면 모든 Amazon EMR 릴리스에서 사용 가능한 EMRFS 클라이언트에서 Amazon S3 암호화 및 암호 해독이 수행됩니다. 객체는 Amazon S3에 업로드되기 전에 암호화되고 다운로드된 후 암호 해독됩니다. 지정하는 공급자는 클라이언트가 사용하는 암호화 키를 제공합니다. 클라이언트는 AWS KMS에서 제공하는 키(CSE-KMS) 또는 클라이언트 측 루트 키(CSE-C)를 제공하는 사용자 지정 Java 클래스를 사용할 수 있습니다. 암호화 세부 사항은 지정된 공급자 및 암호 해독되거나 암호화되는 객체의 메타데이터에 따라 CSE-KMS 및 CSE-C 간에 약간 다릅니다. Amazon S3 CSE를 고객 관리형 키와 함께 사용하는 경우 EMR Serverless 애플리케이션에서 작업을 실행하는 데 사용되는 실행 역할은 키에 대한 액세스를 보유해야 합니다. 추가 KMS 요금이 적용될 수 있습니다. 이러한 차이에 대한 자세한 내용은 Amazon Simple Storage Service 사용 설명서의 [클라이언트 측 암호화를 사용하여 데이터 보호](#)를 참조하세요.

## 로컬 디스크 암호화

임시 스토리지에 저장된 데이터는 업계 표준 AES-256 암호화 알고리즘을 사용하여 서비스 소유 키로 암호화됩니다.

## 키 관리

KMS 키를 자동으로 회전하도록 KMS를 구성할 수 있습니다. 이렇게 하면 1년에 한 번 키가 회전되고 이전 키는 무기한 저장되므로 데이터를 계속 해독할 수 있습니다. 자세한 내용은 [고객 관리형 키 교체](#)를 참조하세요.

## 전송 중 암호화

Amazon EMR Serverless에서 다음과 같은 애플리케이션별 암호화 기능을 사용할 수 있습니다.

- Spark
  - 기본적으로 Spark 드라이버와 실행기 간 통신은 인증되고 내부적으로 사용됩니다. 드라이버와 실행기 간 RPC 통신은 암호화됩니다.
- Hive
  - AWS Glue 메타스토어와 EMR Serverless 애플리케이션 간 통신은 TLS를 통해 수행됩니다.

Amazon S3 버킷 IAM 정책에 [aws:SecureTransport](#) 조건을 사용하여 HTTPS(TLS)를 통해 암호화된 연결만 허용해야 합니다.

## Amazon EMR Serverless의 Identity and Access Management(IAM)

AWS Identity and Access Management(IAM)는 관리자가 AWS 리소스에 대한 액세스를 안전하게 제어할 수 있도록 지원하는 AWS 서비스입니다. IAM 관리자는 어떤 사용자가 Amazon EMR Serverless 리소스를 사용할 수 있도록 인증(로그인됨)되고 권한이 부여(권한 있음)될 수 있는지 제어합니다. IAM은 추가 비용 없이 사용할 수 있는 AWS 서비스입니다.

### 주제

- [고객](#)
- [ID를 통한 인증](#)
- [정책을 사용하여 액세스 관리](#)
- [EMR Serverless에서 IAM와의 작동 방식](#)
- [EMR Serverless에 대한 서비스 연결 역할 사용](#)
- [Amazon EMR Serverless에 대한 작업 런타임 역할](#)
- [EMR Serverless에 대한 사용자 액세스 정책 예제](#)
- [태그 기반 액세스 제어를 위한 정책](#)
- [EMR Serverless에 대한 자격 증명 기반 정책 예제](#)
- [AWS 관리형 정책에 대한 Amazon EMR Serverless 업데이트](#)
- [Amazon EMR Serverless 자격 증명 및 액세스 문제 해결](#)

### 고객

AWS Identity and Access Management(IAM) 사용 방법은 역할에 따라 다릅니다.

- 서비스 사용자 - 기능에 액세스할 수 없는 경우 관리자에게 권한을 요청합니다([Amazon EMR Serverless 자격 증명 및 액세스 문제 해결](#) 참조).
- 서비스 관리자 - 사용자 액세스를 결정하고 권한 요청을 제출합니다([Amazon EMR Serverless의 Identity and Access Management\(IAM\)](#) 참조).
- IAM 관리자 - 액세스를 관리하기 위한 정책을 작성합니다([EMR Serverless에 대한 자격 증명 기반 정책 샘플](#) 참조).

## ID를 통한 인증

인증은 ID 자격 증명을 사용하여 AWS에 로그인하는 방식입니다. AWS 계정 루트 사용자나 IAM 사용자 또는 IAM 역할을 수임하여 인증받아야 합니다.

AWS IAM Identity Center(IAM Identity Center), Single Sign-On 인증 또는 Google/Facebook 자격 증명과 같은 자격 증명 소스의 자격 증명을 사용하여 페더레이션 자격 증명으로 로그인할 수 있습니다. 로그인하는 방법에 대한 자세한 내용은 AWS 로그인 사용 설명서의 [AWS 계정에 로그인하는 방법](#) 섹션을 참조하세요.

프로그래밍 방식 액세스를 위해 AWS는 요청에 암호화 방식으로 서명할 수 있는 SDK 및 CLI를 제공합니다. 자세한 내용은 IAM 사용 설명서의 [API 요청용 AWS Signature Version 4](#) 섹션을 참조하세요.

## AWS 계정 루트 사용자

AWS 계정을 생성하는 경우에는 모든 AWS 서비스 서비스와 리소스에 대한 완전한 액세스 권한이 있는 AWS 계정 루트 사용자라는 단일 로그인 자격 증명으로 시작합니다. 일상적인 작업에 루트 사용자를 사용하지 않을 것을 강력히 권장합니다. 루트 사용자 자격 증명이 필요한 작업은 IAM 사용 설명서의 [루트 사용자 자격 증명이 필요한 작업](#) 섹션을 참조하세요.

## 페더레이션 자격 증명

가장 좋은 방법은 인간 사용자가 ID 제공업체와의 페더레이션을 사용하여 임시 보안 인증으로 AWS 서비스에 액세스하도록 하는 것입니다.

페더레이션 자격 증명은 엔터프라이즈 사용자 디렉터리, 웹 ID 제공업체 또는 Directory Service의 사용자, 자격 증명 소스의 자격 증명을 사용하여 AWS 서비스에 액세스합니다. 페더레이션 자격 증명은 임시 자격 증명을 제공하는 역할을 수임합니다.

중앙 집중식 액세스 관리를 위해 AWS IAM Identity Center를 추천합니다. 자세한 정보는 AWS IAM Identity Center 사용 설명서의 [IAM Identity Center란 무엇인가요?](#) 섹션을 참조하세요.

## IAM 사용자 및 그룹

[IAM 사용자](#)는 단일 개인 또는 애플리케이션에 대한 특정 권한을 가진 자격 증명입니다. 장기 자격 증명에 있는 IAM 사용자 대신 임시 자격 증명을 사용하는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서에서 [임시 자격 증명을 사용하여 AWS에 액세스하려면 인간 사용자가 ID 공급자와의 페더레이션을 사용하도록 요구](#)를 참조하세요.

[IAM 그룹](#)은 IAM 사용자 모음을 지정하고 대규모 사용자 집합에 대한 권한 관리를 더 쉽게 만듭니다. 자세한 내용은 IAM 사용 설명서의 [IAM 사용자 사용 사례](#) 섹션을 참조하세요.

## IAM 역할

[IAM 역할](#)은 임시 자격 증명을 제공하는 특정 권한이 있는 자격 증명입니다. [사용자에서 IAM 역할\(콘솔\)로 전환](#)하거나 AWS CLI 또는 AWS API 작업을 직접 호출하여 역할을 수입할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [역할 수입 방법](#)을 참조하세요.

IAM 역할은 페더레이션 사용자 액세스, 임시 IAM 사용자 권한, 교차 계정 액세스, 교차 서비스 액세스 및 Amazon EC2에서 실행되는 애플리케이션에 유용합니다. 자세한 내용은 IAM 사용 설명서의 [교차 계정 리소스 액세스](#)를 참조하세요.

## 정책을 사용하여 액세스 관리

정책을 생성하고 AWS ID 또는 리소스에 연결하여 AWS에서 내 액세스를 제어합니다. 정책은 자격 증명이나 리소스와 연결될 때 해당 권한을 정의합니다. AWS는 위탁자가 요청을 보낼 때 이러한 정책을 평가합니다. 대부분의 정책은 AWS에 JSON 문서로 저장됩니다. JSON 정책 문서에 대한 자세한 내용은 IAM 사용 설명서의 [JSON 정책 개요](#) 섹션을 참조하세요.

정책을 사용하여 관리자는 어떤 위탁자가 어떤 리소스에 대해 어떤 조건에서 작업을 수행할 수 있는지 정의하여 누가 무엇을 액세스할 수 있는지 지정합니다.

기본적으로, 사용자 및 역할에는 어떠한 권한도 없습니다. IAM 관리자는 IAM 정책을 생성하고 사용자가 수입할 수 있는 역할에 이를 추가합니다. IAM 정책은 작업을 수행하기 위해 사용하는 방법과 관계없이 해당 권한을 정의합니다.

## ID 기반 정책

자격 증명 기반 정책은 자격 증명(사용자, 사용자 그룹 또는 역할)에 연결하는 JSON 권한 정책 문서입니다. 이러한 정책은 자격 증명이 수행할 수 있는 작업, 대상 리소스 및 이에 관한 조건을 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 [고객 관리형 정책으로 사용자 지정 IAM 권한 정의](#)를 참조하세요.

자격 증명 기반 정책은 인라인 정책(단일 자격 증명에 직접 포함) 또는 관리형 정책(여러 자격 증명에 연결된 독립 실행형 정책)일 수 있습니다. 관리형 정책 또는 인라인 정책을 선택하는 방법을 알아보려면 IAM 사용 설명서의 [관리형 정책 및 인라인 정책 중에서 선택](#) 섹션을 참조하세요.

## 리소스 기반 정책

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 예를 들어, IAM 역할 신뢰 정책 및 Amazon S3 버킷 정책이 있습니다. 리소스 기반 정책을 지원하는 서비스에서 서비스 관리자는 이러한 정책을 사용하여 특정 리소스에 대한 액세스를 통제할 수 있습니다. 리소스 기반 정책에서 [위탁자를 지정](#)해야 합니다.

리소스 기반 정책은 해당 서비스에 있는 인라인 정책입니다. 리소스 기반 정책에서는 IAM의 AWS 관리형 정책을 사용할 수 없습니다.

## 기타 정책 타입

AWS는 더 일반적인 정책 유형에 따라 부여되는 최대 권한을 설정할 수 있는 추가 정책 유형을 지원합니다.

- 권한 경계 - 자격 증명 기반 정책에서 IAM 엔터티에 부여할 수 있는 최대 권한을 설정합니다. 자세한 정보는 IAM 사용 설명서의 [IAM 엔터티의 권한 범위](#)를 참조하세요.
- 서비스 제어 정책(SCP) - AWS Organizations 내 조직 또는 조직 단위에 대한 최대 권한을 지정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 [서비스 제어 정책](#)을 참조하세요.
- 리소스 제어 정책(RCP) - 계정 내 리소스에 사용할 수 있는 최대 권한을 설정합니다. 자세한 내용은 AWS Organizations 사용 설명서의 [리소스 제어 정책\(RCP\)](#)을 참조하세요.
- 세션 정책 - 역할 또는 페더레이션 사용자에게 대해 임시 세션을 생성할 때 파라미터 자격으로 전달되는 고급 정책입니다. 자세한 내용은 IAM 사용 설명서의 [세션 정책](#)을 참조하세요.

## 여러 정책 유형

여러 정책 유형이 요청에 적용되는 경우, 결과 권한은 이해하기가 더 복잡합니다. 여러 정책 유형이 관련될 때 AWS가 요청을 허용할지를 결정하는 방법을 알아보려면 IAM 사용 설명서의 [정책 평가 로직](#)을 참조하세요.

## EMR Serverless에서 IAM와의 작동 방식

IAM을 사용하여 Amazon EMR Serverless에 대한 액세스를 관리하기 전에 Amazon EMR Serverless에서 사용할 수 있는 IAM 기능을 알아봅니다.

## EMR Serverless에서 사용할 수 있는 IAM 기능

IAM 기능	Amazon EMR Serverless 지원
<a href="#">ID 기반 정책</a>	예
<a href="#">리소스 기반 정책</a>	아니요
<a href="#">정책 작업</a>	예
<a href="#">정책 리소스</a>	예
<a href="#">정책 조건 키</a>	아니요
<a href="#">ACLs</a>	아니요
<a href="#">ABAC(정책의 태그)</a>	예
<a href="#">임시 보안 인증</a>	예
<a href="#">보안 주체 권한</a>	예
<a href="#">서비스 역할</a>	아니요
<a href="#">서비스 링크 역할</a>	예

EMR Serverless 및 기타 AWS 서비스가 대부분의 IAM 기능과 작동하는 방법을 개괄적으로 알아보려면 IAM 사용 설명서에서 [IAM으로 작업하는 AWS 서비스](#)를 참조하세요.

## EMR Serverless에 대한 자격 증명 기반 정책

ID 기반 정책 지원: 예

ID 기반 정책은 IAM 사용자, 사용자 그룹 또는 역할과 같은 ID에 연결할 수 있는 JSON 권한 정책 문서입니다. 이러한 정책은 사용자 및 역할이 어떤 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 제어합니다. 자격 증명 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서에서 [고객 관리형 정책으로 사용자 지정 IAM 권한 정의](#)를 참조하세요.

IAM ID 기반 정책을 사용하면 허용되거나 거부되는 작업과 리소스뿐 아니라 작업이 허용되거나 거부되는 조건을 지정할 수 있습니다. JSON 정책에서 사용하는 모든 요소에 대해 알아보려면 IAM 사용 설명서의 [IAM JSON 정책 요소 참조](#)를 참조하세요.

## EMR Serverless에 대한 자격 증명 기반 정책 샘플

Amazon EMR Serverless 자격 증명 기반 정책 예제를 보려면 [EMR Serverless에 대한 자격 증명 기반 정책 예제](#) 섹션을 참조하세요.

## Amazon EMR 내 리소스 기반 정책

리소스 기반 정책 지원: 아니요

리소스 기반 정책은 리소스에 연결하는 JSON 정책 설명서입니다. 리소스 기반 정책의 예제는 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 위탁자가 해당 리소스와 어떤 조건에서 어떤 작업을 수행할 수 있는지를 정의합니다. 리소스 기반 정책에서 [위탁자를 지정](#)해야 합니다. 보안 주체에는 계정, 사용자, 역할, 페더레이션 사용자 또는 AWS 서비스가 포함될 수 있습니다.

교차 계정 액세스를 활성화하려는 경우, 전체 계정이나 다른 계정의 IAM 개체를 리소스 기반 정책의 위탁자로 지정할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [교차 계정 리소스 액세스](#)를 참조하세요.

## EMR Serverless에 대한 정책 작업

정책 작업 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

JSON 정책의 Action 요소는 정책에서 액세스를 허용하거나 거부하는 데 사용할 수 있는 작업을 설명합니다. 연결된 작업을 수행할 수 있는 권한을 부여하기 위한 정책에 작업을 포함하세요.

EMR Serverless 작업의 목록을 보려면 서비스 승인 참조의 [Amazon EMR Serverless의 작업, 리소스 및 조건 키](#)를 참조하세요.

EMR Serverless의 정책 작업은 작업 앞에 다음 접두사를 사용합니다.

```
emr-serverless
```

단일 문에서 여러 작업을 지정하려면 다음과 같이 쉼표로 구분합니다.

```
"Action": [
    "emr-serverless:action1",
    "emr-serverless:action2"
```

]

Amazon EMR Serverless 자격 증명 기반 정책 예제를 보려면 [EMR Serverless에 대한 자격 증명 기반 정책 예제](#) 섹션을 참조하세요.

## EMR Serverless에 대한 정책 리소스

정책 리소스 지원: 예

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Resource JSON 정책 요소는 작업이 적용되는 하나 이상의 객체를 지정합니다. 모범 사례에 따라 [Amazon 리소스 이름\(ARN\)](#)을 사용하여 리소스를 지정합니다. 리소스 수준 권한을 지원하지 않는 작업의 경우, 와일드카드(\*)를 사용하여 해당 문이 모든 리소스에 적용됨을 나타냅니다.

```
"Resource": "*"

```

Amazon EMR Serverless 리소스 유형 및 해당 ARN의 목록을 보려면 서비스 승인 참조의 [Amazon EMR에서 정의한 리소스](#)를 참조하세요. 각 리소스의 ARN을 지정하는 작업에 대해 알아보려면 [Actions, resources, and condition keys for Amazon EMR Serverless](#)를 참조하세요.

Amazon EMR Serverless 자격 증명 기반 정책 예제를 보려면 [EMR Serverless에 대한 자격 증명 기반 정책 예제](#) 섹션을 참조하세요.

## EMR Serverless에 대한 정책 조건 키

정책 조건 키 지원

서비스별 정책 조건 키 지원

아니요

관리자는 AWS JSON 정책을 사용하여 누가 무엇에 액세스할 수 있는지를 지정할 수 있습니다. 즉, 어떤 위탁자가 어떤 리소스와 어떤 조건에서 작업을 수행할 수 있는지를 지정할 수 있습니다.

Condition 요소는 정의된 기준에 따라 문이 실행되는 시기를 지정합니다. 같거나 작음과 같은 [조건 연산자](#)를 사용하여 정책의 조건을 요청의 값과 일치시키는 조건식을 생성할 수 있습니다. 모든 AWS 전역 조건 키를 보려면 IAM 사용 설명서의 [AWS 전역 조건 컨텍스트 키](#)를 참조하세요.

Amazon EMR Serverless 조건 키 목록을 보고 조건 키를 사용할 수 있는 작업 및 리소스에 대해 알아보려면 서비스 승인 참조에서 [Actions, resources, and condition keys for Amazon EMR Serverless](#)를 참조하세요.

모든 Amazon EC2 작업은 `aws:RequestedRegion` 및 `ec2:Region` 조건 키를 지원합니다. 자세한 내용은 [예제: 특정 리전으로 액세스 제한](#)을 참조하세요.

## EMR Serverless의 액세스 제어 목록(ACL)

ACL 지원: 아니요

액세스 제어 목록(ACL)은 어떤 위탁자(계정 멤버, 사용자 또는 역할)가 리소스에 액세스할 수 있는 권한을 가지고 있는지를 제어합니다. ACL은 JSON 정책 문서 형식을 사용하지 않지만 리소스 기반 정책과 유사합니다.

## EMR Serverless를 사용한 속성 기반 액세스 제어(ABAC)

ABAC(속성 기반 액세스 제어) 지원

ABAC 지원(정책의 태그)	예
-----------------	---

속성 기반 액세스 제어(ABAC)는 태그라고 불리는 속성을 기반으로 권한을 정의하는 권한 부여 전략입니다. IAM 엔터티 및 AWS 리소스에 태그를 연결하면 위탁자의 태그가 리소스 태그와 일치할 때 작업을 허용하도록 ABAC 정책을 설계할 수 있습니다.

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 내용은 IAM 사용 설명서의 [ABAC 권한 부여를 통한 권한 정의](#)를 참조하세요. ABAC 설정 단계가 포함된 자습서를 보려면 IAM 사용 설명서의 [속성 기반 액세스 제어\(ABAC\) 사용](#)을 참조하세요.

## EMR Serverless에서 임시 자격 증명 사용

임시 자격 증명 지원: 예

임시 자격 증명은 AWS 리소스에 대한 단기 액세스를 제공하며 페더레이션 또는 전환 역할을 사용할 때 자동으로 생성됩니다. AWS에서는 장기 액세스 키를 사용하는 대신 임시 자격 증명을 동적으로 생

성할 것을 권장합니다. 자세한 내용은 IAM 사용 설명서의 [IAM 임시 보안 자격 증명](#) 및 [IAM으로 작업하는 AWS 서비스](#) 섹션을 참조하세요.

## EMR Serverless에 대한 교차 서비스 위탁자 권한

전달 액세스 세션(FAS) 지원: 예

전달 액세스 세션(FAS)은 AWS 서비스를 직접 호출하는 위탁자의 권한과 요청하는 AWS 서비스를 함께 사용하여 다운스트림 서비스에 대한 요청을 수행합니다. FAS 요청 시 정책 세부 정보는 [전달 액세스 세션](#)을 참조하세요.

## EMR Serverless의 서비스 역할

서비스 역할 지원	아니요
-----------	-----

## EMR Serverless에 대한 서비스 연결 역할

서비스 링크 역할 지원	예
--------------	---

서비스 연결 역할 생성 또는 관리에 대한 자세한 내용은 [IAM으로 작업하는 AWS 서비스](#)를 참조하세요. 서비스 연결 역할 열에서 Yes이(가) 포함된 서비스를 테이블에서 찾습니다. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

## EMR Serverless에 대한 서비스 연결 역할 사용

Amazon EMR Serverless는 AWS Identity and Access Management(IAM) [서비스 연결 역할](#)을 사용합니다. 서비스 연결 역할은 EMR Serverless 서비스에 직접 연결된 고유한 유형의 IAM 역할입니다. 서비스 연결 역할은 EMR Serverless에서 사전 정의하며 서비스에서 다른 AWS 서비스를 자동으로 호출하기 위해 필요한 모든 권한을 포함합니다.

필요한 권한을 수동으로 추가할 필요가 없으므로 서비스 연결 역할은 EMR Serverless를 더 쉽게 설정할 수 있습니다. EMR Serverless에서 서비스 연결 역할의 권한을 정의하므로 다르게 정의하지 않은 한, EMR Serverless에서만 해당 역할을 수입할 수 있습니다. 정의된 권한에는 신뢰 정책과 권한 정책이 포함되며 이 권한 정책은 다른 IAM 엔티티에 연결할 수 없습니다.

먼저 관련 리소스를 삭제한 후에만 서비스 연결 역할을 삭제할 수 있습니다. 이렇게 하면 리소스에 대한 액세스 권한을 부주의로 삭제할 수 없기 때문에 EMR Serverless 리소스가 보호됩니다.

서비스 연결 역할을 지원하는 기타 서비스에 대한 자세한 내용을 알아보려면 [AWS IAM으로 작업하는 서비스](#)를 참조하고 서비스 연결 역할 열에 예가 표시된 서비스를 찾으세요. 해당 서비스에 대한 서비스 연결 역할 설명서를 보려면 예(Yes) 링크를 선택합니다.

## EMR Serverless에 대한 서비스 연결 역할 권한

EMR Serverless는 AWSServiceRoleForAmazonEMRServerless라는 서비스 연결 역할을 사용하여 사용자 대신, AWS API를 직접 호출할 수 있습니다.

AWSServiceRoleForAmazonEMRServerless 서비스 연결 역할은 역할을 수임하기 위해 다음 서비스를 신뢰합니다.

- ops.emr-serverless.amazonaws.com

AmazonEMRServerlessServiceRolePolicy라는 이름의 역할 권한 정책은 EMR Serverless가 지정된 리소스에서 다음 작업을 완료하도록 허용합니다.

### Note

관리형 정책 콘텐츠는 수시로 변경되므로 여기에 수록된 정책은 이전 버전일 수 있습니다. AWS Management 콘솔에서 최신 [AmazonEMRServerlessServiceRolePolicy](#) 정책을 확인합니다.

- 작업: ec2:CreateNetworkInterface
- 작업: ec2:DeleteNetworkInterface
- 작업: ec2:DescribeNetworkInterfaces
- 작업: ec2:DescribeSecurityGroups
- 작업: ec2:DescribeSubnets
- 작업: ec2:DescribeVpcs
- 작업: ec2:DescribeDhcpOptions
- 작업: ec2:DescribeRouteTables
- 작업: cloudwatch:PutMetricData

다음은 전체 AmazonEMRServerlessServiceRolePolicy 정책입니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EC2PolicyStatement",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface",
        "ec2:DeleteNetworkInterface",
        "ec2:DescribeNetworkInterfaces",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "ec2:DescribeDhcpOptions",
        "ec2:DescribeRouteTables"
      ],
      "Resource": [
        "*"
      ]
    },
    {
      "Sid": "CloudWatchPolicyStatement",
      "Effect": "Allow",
      "Action": [
        "cloudwatch:PutMetricData"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "cloudwatch:namespace": [
            "AWS/EMRServerless",
            "AWS/Usage"
          ]
        }
      }
    }
  ]
}
```

EMR Serverless 위탁자가 이 역할을 수임하도록 허용하려면 다음 신뢰 정책을 역할에 연결합니다.

JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "sts:AssumeRole"
      ],
      "Resource": "arn:aws:iam::123456789012:role/aws-service-role/emr-serverless.amazonaws.com/AWSServiceRoleForEMRServerless",
      "Sid": "AllowSTSAssumerole"
    }
  ]
}
```

IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 권한](#)을 참조하세요.

## EMR Serverless에 대한 서비스 연결 역할 생성

서비스 링크 역할은 수동으로 생성할 필요가 없습니다. AWS Management 콘솔(EMR Studio 사용), AWS CLI 또는 AWS API에서 새 EMR Serverless 애플리케이션을 생성하면 EMR Serverless가 서비스 연결 역할을 자동으로 생성합니다. IAM 엔터티(사용자, 그룹, 역할 등)가 서비스 링크 역할을 생성하고 편집하거나 삭제할 수 있도록 권한을 구성할 수 있습니다.

IAM을 사용하여 AWSServiceRoleForEMRCleanup 서비스 연결 역할을 생성하는 방법

서비스 연결 역할을 생성해야 하는 IAM 개체의 권한 정책에 다음 명령문을 추가합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "iam:CreateServiceLinkedRole"
  ],
  "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless*",
}
```

```
"Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"}}
}
```

이 서비스 연결 역할을 삭제했다가 다시 생성해야 하는 경우 동일한 프로세스를 사용하여 계정에서 역할을 다시 생성할 수 있습니다. 새 EMR Serverless 애플리케이션을 생성하면 EMR Serverless가 서비스 연결 역할을 자동으로 다시 생성합니다.

IAM 콘솔을 사용하여 EMR Serverless 사용 사례로 서비스 연결 역할을 생성할 수도 있습니다. AWS CLI 또는 AWS API에서 ops.emr-serverless.amazonaws.com 서비스 이름의 서비스 연결 역할을 생성합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 생성](#)을 참조하세요. 이 서비스 연결 역할을 삭제하면 동일한 프로세스를 사용하여 역할을 다시 생성할 수 있습니다.

## EMR Serverless에 대한 서비스 연결 역할 편집

다양한 엔터티가 역할을 참조할 수 있으므로 EMR Serverless는 AWSServiceRoleForAmazonEMRServerless 서비스 연결 역할을 편집할 수 없습니다. EMR Serverless 서비스 연결 역할이 EMR Serverless에 필요한 모든 권한을 포함하므로 EMR Serverless 서비스 연결 역할에서 사용하는 AWS 소유의 IAM 정책을 편집할 수 없습니다. 하지만 IAM을 사용하여 역할의 설명을 편집할 수 있습니다.

IAM을 사용하여 AWSServiceRoleForEMRCleanup 서비스 연결 역할의 설명을 편집하는 방법

서비스 연결 역할의 설명을 편집해야 하는 IAM 개체의 권한 정책에 다음 명령문을 추가합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "iam: UpdateRoleDescription"
  ],
  "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless*",
  "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"}}
}
```

자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 편집](#)을 참조하세요.

## EMR Serverless에 대한 서비스 연결 역할 삭제

서비스 연결 역할이 필요한 기능 또는 서비스가 더 이상 필요 없는 경우에는 해당 역할을 삭제하는 것이 좋습니다. 이 경우 적극적으로 모니터링하거나 유지하지 않는 미사용 엔터티가 없도록 합니다. 그러

나 서비스 연결 역할을 삭제하려면 먼저 모든 리전에서 모든 EMR Serverless 애플리케이션을 삭제합니다.

#### Note

역할에 연결된 리소스를 삭제하려 할 때 EMR Serverless 서비스가 역할을 사용 중이면 삭제에 실패할 수 있습니다. 이 문제가 발생하면 몇 분 기다렸다가 작업을 다시 시도하세요.

IAM을 사용하여 AWSServiceRoleForEMRCleanup 서비스 연결 역할을 삭제하는 방법

서비스 연결 역할을 삭제해야 하는 IAM 개체의 권한 정책에 다음 명령문을 추가합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "iam:DeleteServiceLinkedRole",
    "iam:GetServiceLinkedRoleDeletionStatus"
  ],
  "Resource": "arn:aws:iam::*:role/aws-service-role/ops.emr-serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless*",
  "Condition": {"StringLike": {"iam:AWSServiceName": "ops.emr-serverless.amazonaws.com"}}
}
```

IAM을 사용하여 수동으로 서비스 연결 역할을 삭제하는 방법

IAM 콘솔, AWS CLI 또는 AWS API를 사용하여 AWSServiceRoleForAmazonEMRServerless 서비스 연결 역할을 삭제합니다. 자세한 내용은 IAM 사용 설명서의 [서비스 연결 역할 삭제](#)를 참조하세요.

## EMR Serverless 서비스 연결 역할에 대해 지원되는 리전

EMR Serverless는 서비스가 제공되는 모든 리전에서 서비스 연결 역할을 사용하도록 지원합니다. 자세한 내용은 [AWS 리전 및 엔드포인트](#)를 참조하세요.

## Amazon EMR Serverless에 대한 작업 런타임 역할

사용자를 대신하여 다른 서비스를 직접 호출하는 경우 EMR Serverless 작업 실행에서 수입할 수 있는 IAM 역할 권한을 지정할 수 있습니다. 여기에는 모든 데이터 소스, 대상 및 Amazon Redshift 클러스터 및 DynamoDB 테이블과 같은 기타 AWS 리소스에서의 Amazon S3에 대한 액세스가 포함됩니다. 역할을 생성하는 방법에 대해 자세히 알아보려면 [작업 런타임 역할 생성](#) 섹션을 참조하세요.

## 샘플 런타임 정책

다음과 같은 런타임 정책을 작업 런타임 역할에 연결할 수 있습니다. 다음 작업 런타임 정책은 다음을 허용합니다.

- EMR 샘플이 포함된 Amazon S3 버킷에 대한 읽기 액세스.
- S3 버킷에 대한 전체 액세스.
- AWS Glue Data Catalog에 대한 생성 및 읽기 액세스.

DynamoDB와 같은 다른 AWS 리소스에 대한 액세스를 추가하려면 런타임 역할을 생성할 때 정책에 해당 리소스에 대한 권한을 포함해야 합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ReadAccessForEMRSamples",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*elasticmapreduce",
        "arn:aws:s3:::*elasticmapreduce/*"
      ]
    },
    {
      "Sid": "FullAccessToS3Bucket",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:GetObject",
        "s3:ListBucket",
        "s3:DeleteObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket",
        "arn:aws:s3:::amzn-s3-demo-bucket/*"
      ]
    }
  ]
}
```

```

    ]
  },
  {
    "Sid": "GlueCreateAndReadDataCatalog",
    "Effect": "Allow",
    "Action": [
      "glue:GetDatabase",
      "glue:CreateDatabase",
      "glue:GetDataBases",
      "glue:CreateTable",
      "glue:GetTable",
      "glue:UpdateTable",
      "glue>DeleteTable",
      "glue:GetTables",
      "glue:GetPartition",
      "glue:GetPartitions",
      "glue:CreatePartition",
      "glue:BatchCreatePartition",
      "glue:GetUserDefinedFunctions"
    ],
    "Resource": [
      "*"
    ]
  }
]
}

```

## 역할 권한 전달

사용자가 승인된 역할만 전달할 수 있도록 사용자 역할에 IAM 권한 정책을 연결할 수 있습니다. 이를 통해 관리자는 특정 작업 런타임 역할을 EMR Serverless 작업에 전달할 수 있는 사용자를 제어할 수 있습니다. 권한 설정에 대해 자세히 알아보려면 [사용자에게 AWS 서비스에 역할을 전달할 수 있는 권한 부여](#)를 참조하세요.

다음은 EMR Serverless 서비스 위탁자에 작업 런타임 역할을 전달할 수 있도록 허용하는 정책 예제입니다.

```

{
  "Effect": "Allow",
  "Action": "iam:PassRole",
  "Resource": "arn:aws:iam::1234567890:role/JobRuntimeRoleForEMRServerless",
  "Condition": {

```

```

        "StringLike": {
            "iam:PassedToService": "emr-serverless.amazonaws.com"
        }
    }
}

```

## 런타임 역할과 연결된 관리형 권한 정책

EMR Studio 콘솔을 통해 EMR Serverless에 작업 실행을 제출하면 애플리케이션과 연결할 런타임 역할을 선택하는 단계가 있습니다. 콘솔의 각 선택 항목과 관련된 기본 관리 정책이 있으며, 이를 숙지하는 것이 중요합니다. 이러한 세 가지 선택 항목은 다음과 같습니다.

1. 모든 버킷 - 이 옵션을 선택하면 모든 버킷에 대한 전체 액세스를 제공하는 [AmazonS3FullAccess](#) AWS 관리형 정책을 지정합니다.
2. 특정 버킷 - 선택한 각 버킷의 Amazon 리소스 이름(ARN) 식별자를 지정합니다. 기본 관리형 정책은 포함되어 있지 않습니다.
3. 없음 - 관리형 정책 권한이 포함되지 않습니다.

특정 버킷을 추가하는 것이 좋습니다. 모든 버킷을 선택하는 경우 모든 버킷에 대한 전체 액세스 권한이 설정된다는 점에 유의하세요.

## EMR Serverless에 대한 사용자 액세스 정책 예제

EMR Serverless 애플리케이션과 상호 작용하는 경우 각 사용자가 수행할 작업에 따라 사용자를 위한 세분화된 정책을 설정할 수 있습니다. 다음 정책은 사용자에게 적합한 권한을 설정하는 데 도움이 될 수 있는 예제입니다. 이 섹션에서는 EMR Serverless 정책에만 중점을 둡니다. EMR Studio 사용자 정책 샘플은 [EMR Studio 사용자 권한 구성](#)을 참조하세요. IAM 사용자(위탁자)에 정책을 연결하는 방법에 대한 자세한 내용은 IAM 사용 설명서의 [IAM 정책 관리](#)를 참조하세요.

### 고급 사용자 정책

EMR Serverless에 필요한 모든 작업을 부여하려면 AmazonEMRServerlessFullAccess 정책을 생성하고 필요한 IAM 사용자, 역할 또는 그룹에 연결합니다.

다음은 고급 사용자가 EMR Serverless 애플리케이션을 생성 및 수정하고 작업 제출 및 디버깅과 같은 기타 작업을 수행할 수 있도록 허용하는 샘플 정책입니다. 여기에는 EMR Serverless가 다른 서비스에 요구하는 모든 작업이 표시됩니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication",
        "emr-serverless:UpdateApplication",
        "emr-serverless>DeleteApplication",
        "emr-serverless:ListApplications",
        "emr-serverless:GetApplication",
        "emr-serverless:StartApplication",
        "emr-serverless:StopApplication",
        "emr-serverless:StartJobRun",
        "emr-serverless:CancelJobRun",
        "emr-serverless:ListJobRuns",
        "emr-serverless:GetJobRun"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}
```

VPC에 대한 네트워크 연결을 활성화하면 EMR Serverless 애플리케이션에서 VPC 리소스와 통신할 Amazon EC2 탄력적 네트워크 인터페이스(ENI)를 생성합니다. 다음 정책은 새로운 EC2 ENI가 EMR Serverless 애플리케이션의 컨텍스트에서만 생성되도록 보장합니다.

 Note

EMR Serverless 애플리케이션을 시작하는 컨텍스트를 제외하고 사용자가 EC2 ENI를 생성할 수 없도록 이 정책을 설정하는 것이 좋습니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowEC2ENICreationWithEMRTags",
      "Effect": "Allow",
      "Action": [
        "ec2:CreateNetworkInterface"
      ],
      "Resource": [
        "arn:aws:ec2:*:*:network-interface/*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:CalledViaLast": "ops.emr-serverless.amazonaws.com"
        }
      }
    }
  ]
}
```

특정 서브넷에 대한 EMR Serverless 액세스를 제한하려면 각 서브넷에 태그 조건으로 태그를 지정할 수 있습니다. 이 IAM 정책은 EMR Serverless 애플리케이션이 허용된 서브넷 내에서만 EC2 ENI를 생성할 수 있도록 보장합니다.

```
{
  "Sid": "AllowEC2ENICreationInSubnetAndSecurityGroupWithEMRTags",
  "Effect": "Allow",
  "Action": [
    "ec2:CreateNetworkInterface"
  ],
  "Resource": [
    "arn:aws:ec2:*:*:subnet/*",
    "arn:aws:ec2:*:*:security-group/*"
  ],
  "Condition": {
    "StringEquals": {
      "aws:ResourceTag/KEY": "VALUE"
    }
  }
}
```

```
}
}
```

### ⚠ Important

첫 번째 애플리케이션을 생성하는 관리자 또는 고급 사용자인 경우 EMR Serverless 서비스 연결 역할을 생성할 수 있도록 권한 정책을 구성해야 합니다. 자세한 내용은 [EMR Serverless에 대한 서비스 연결 역할 사용](#) 섹션을 참조하세요.

다음 IAM 정책은 계정에 대한 EMR Serverless 서비스 연결 역할을 생성할 수 있도록 허용합니다.

```
{
  "Sid": "AllowEMRServerlessServiceLinkedRoleCreation",
  "Effect": "Allow",
  "Action": "iam:CreateServiceLinkedRole",
  "Resource": "arn:aws:iam::account-id:role/aws-service-role/ops.emr-serverless.amazonaws.com/AWSServiceRoleForAmazonEMRServerless"
}
```

## 데이터 엔지니어 정책

다음은 사용자가 EMR Serverless 애플리케이션에 대한 읽기 전용 권한과 작업을 제출하고 디버깅할 수 있는 기능을 허용하는 샘플 정책입니다. 이 정책은 작업을 명시적으로 거부하지 않기 때문에 다른 정책 설명을 사용하더라도 지정된 작업에 대한 액세스를 허용할 수 있습니다.

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:ListApplications",
        "emr-serverless:GetApplication",
        "emr-serverless:StartApplication",
        "emr-serverless:StartJobRun",
        "emr-serverless:CancelJobRun",

```

```

        "emr-serverless:ListJobRuns",
        "emr-serverless:GetJobRun"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

## 액세스 제어에 태그 사용

세분화된 액세스 제어를 위해 태그 조건을 사용할 수 있습니다. 예를 들어, 팀 이름으로 태그가 지정된 EMR Serverless 애플리케이션에만 작업을 제출할 수 있도록 한 팀에서 사용자를 제한할 수 있습니다.

### JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "EMRServerlessActions",
      "Effect": "Allow",
      "Action": [
        "emr-serverless:ListApplications",
        "emr-serverless:GetApplication",
        "emr-serverless:StartApplication",
        "emr-serverless:StartJobRun",
        "emr-serverless:CancelJobRun",
        "emr-serverless:ListJobRuns",
        "emr-serverless:GetJobRun"
      ],
      "Resource": [
        "*"
      ]
    }
  ]
}

```

## 태그 기반 액세스 제어를 위한 정책

자격 증명 기반 정책의 조건을 사용하여 태그를 기반으로 애플리케이션 및 작업 실행에 대한 액세스를 제어할 수 있습니다.

다음 예제에서는 EMR Serverless 조건 키와 함께 조건 연산자를 사용할 수 있는 다양한 시나리오와 방법을 보여줍니다. 이러한 IAM 정책 명령문은 데모용일 뿐이며 프로덕션 환경에서 사용해서는 안 됩니다. 다양한 방법으로 정책 명령문을 결합하여 요구 사항에 따라 권한을 부여하거나 거부할 수 있습니다. IAM 정책 계획 및 테스트에 대한 자세한 내용은 [IAM 사용 설명서](#)를 참조하세요.

### Important

태깅 작업에 대한 권한을 명시적으로 거부하는 것은 중요한 고려 사항입니다. 이렇게 하면 사용자가 리소스에 태그를 지정하지 못하므로, 부여할 의도가 없는 권한이 부여되지 않도록 방지할 수 있습니다. 리소스에 대한 태그 지정 작업이 거부되지 않는 경우 사용자는 태그를 수정하여 태그 기반 정책의 의도를 우회할 수 있습니다. 태그 지정 작업을 거부하는 정책의 예제는 [태그를 추가 또는 제거하는 액세스 거부](#) 섹션을 참조하세요.

아래 예제에서는 EMR Serverless 애플리케이션에서 허용되는 작업을 제어하는 데 사용되는 자격 증명 기반 권한 정책을 보여줍니다.

### 특정 태그 값이 있는 리소스에서만 작업 허용

다음 정책 예제에서 StringEquals 조건 연산자는 dev를 department 태그의 값과 일치시키려고 시도합니다. department 태그가 애플리케이션에 추가되지 않았거나 dev 값이 포함되지 않은 경우 정책이 적용되지 않으며 이 정책에 따라 작업이 허용되지 않습니다. 작업을 허용하는 다른 정책 명령문이 없는 경우 사용자는 이 값과 함께 이 태그가 있는 애플리케이션에서만 작업할 수 있습니다.

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:GetApplication"
      ],

```

```

    "Resource": [
      "*"
    ],
    "Condition": {
      "StringEquals": {
        "aws:ResourceTag/department": "dev"
      }
    },
    "Sid": "AllowEMRSERVERLESSGetapplication"
  }
]
}

```

또한 조건 연산자를 사용하여 여러 태그 값을 지정할 수 있습니다. 예를 들어, department 태그에 dev 또는 test 값이 포함된 애플리케이션에서 작업을 허용하려면 이전 예제의 조건 블록을 다음으로 대체할 수 있습니다.

```

"Condition": {
  "StringEquals": {
    "emr-serverless:ResourceTag/department": ["dev", "test"]
  }
}

```

## 리소스 생성 시 태그 지정이 필요함

아래 예제에서 가상 클러스터를 생성하는 경우 태그를 적용해야 합니다.

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {

```

```

        "StringEquals": {
            "aws:RequestedRegion": "us-east-1"
        },
        "Sid": "AllowEMRSERVERLESSCreateapplication"
    }
}

```

다음 정책 명령문에서는 애플리케이션에 department 태그가 있는 경우에만 사용자가 애플리케이션을 생성할 수 있도록 허용합니다. 이 태그에는 어떤 값이든 포함될 수 있습니다.

## JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "emr-serverless:CreateApplication"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
          "aws:RequestedRegion": ["us-east-1", "us-west-2"]
        }
      },
      "Sid": "AllowEMRSERVERLESSCreateapplication"
    }
  ]
}

```

## 태그를 추가 또는 제거하는 액세스 거부

이 정책은 값이 dev가 아닌 department 태그가 있는 EMR Serverless 애플리케이션에서 사용자가 태그를 추가하거나 제거하는 것을 방지합니다.

## JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
        "emr-serverless:TagResource",
        "emr-serverless:UntagResource"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalTag/department": "dev"
        }
      },
      "Sid": "AllowEMRSERVERLESSTagresource"
    }
  ]
}
```

## EMR Serverless에 대한 자격 증명 기반 정책 예제

기본적으로 사용자 및 역할은 Amazon EMR Serverless 리소스를 생성하거나 수정할 수 있는 권한이 없습니다. 사용자에게 사용자가 필요한 리소스에서 작업을 수행할 권한을 부여하려면 IAM 관리자가 IAM 정책을 생성하면 됩니다.

이러한 예제 JSON 정책 문서를 사용하여 IAM ID 기반 정책을 생성하는 방법을 알아보려면 IAM 사용 설명서의 [IAM 정책 생성\(콘솔\)](#)을 참조하세요.

각 리소스 유형에 대한 ARN의 형식을 포함하여 Amazon OpenSearch EMR Serverless에서 정의한 작업 및 리소스 유형에 대한 자세한 내용은 서비스 승인 참조의 [Actions, resources, and condition keys for Amazon EMR Serverless](#)를 참조하세요.

### 주제

- [정책 모범 사례](#)
- [사용자가 자신의 고유한 권한을 볼 수 있도록 허용](#)

## 정책 모범 사례

### Note

EMR Serverless는 관리형 정책을 지원하지 않으므로 아래 나열된 첫 번째 사례는 적용되지 않습니다.

자격 증명 기반 정책에 따라 계정에서 사용자가 Amazon EMR Serverless 리소스를 생성, 액세스 또는 삭제할 수 있는지 여부가 결정됩니다. 이 작업으로 인해 AWS 계정에 비용이 발생할 수 있습니다. ID 기반 정책을 생성하거나 편집할 때는 다음 지침과 권장 사항을 따릅니다.

- AWS 관리형 정책으로 시작하고 최소 권한을 향해 나아가기 - 사용자 및 워크로드에 권한 부여를 시작하려면 많은 일반 사용 사례에 대한 권한을 부여하는 AWS 관리형 정책을 사용합니다. AWS 계정에서 사용할 수 있습니다. 사용 사례에 고유한 AWS 고객 관리형 정책을 정의하여 권한을 줄이는 것이 좋습니다. 자세한 내용은 IAM 사용 설명서의 [AWS 관리형 정책](#) 또는 [AWS 직무에 대한 관리형 정책](#)을 참조하세요.
- 최소 권한 적용 - IAM 정책을 사용하여 권한을 설정하는 경우, 작업을 수행하는 데 필요한 권한만 부여합니다. 이렇게 하려면 최소 권한으로 알려진 특정 조건에서 특정 리소스에 대해 수행할 수 있는 작업을 정의합니다. IAM을 사용하여 권한을 적용하는 방법에 대한 자세한 정보는 IAM 사용 설명서에 있는 [IAM의 정책 및 권한](#)을 참조하세요.
- IAM 정책의 조건을 사용하여 액세스 추가 제한 - 정책에 조건을 추가하여 작업 및 리소스에 대한 액세스를 제한할 수 있습니다. 예를 들어, SSL을 사용하여 모든 요청을 전송해야 한다고 지정하는 정책 조건을 작성할 수 있습니다. CloudFormation과 같이, 특정 AWS 서비스를 통해 사용되는 경우에만 서비스 작업에 대한 액세스 권한을 부여할 수도 있습니다. 자세한 내용은 IAM 사용 설명서의 [IAM JSON 정책 요소: 조건](#)을 참조하세요.
- IAM Access Analyzer를 통해 IAM 정책을 확인하여 안전하고 기능적인 권한 보장 - IAM Access Analyzer에서는 IAM 정책 언어(JSON)와 모범 사례가 정책에서 준수되도록 새로운 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 Analyzer에서 정책 검증](#)을 참조하세요.
- 다중 인증(MFA) 필요 - AWS 계정에 IAM 사용자 또는 루트 사용자가 필요한 시나리오가 있는 경우, 추가 보안을 위해 MFA를 설정합니다. API 작업을 직접 호출할 때 MFA가 필요하다면 정책에 MFA 조건을 추가합니다. 자세한 내용은 IAM 사용 설명서의 [MFA를 통한 보안 API 액세스](#)를 참조하세요.

IAM의 모범 사례에 대한 자세한 내용은 IAM 사용 설명서의 [IAM의 보안 모범 사례](#)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 AWS CLI 또는 AWS API를 사용하여 프로그래밍 방식으로 이 태스크를 완료할 수 있는 권한이 포함됩니다.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ViewOwnUserInfo",
      "Effect": "Allow",
      "Action": [
        "iam:GetUserPolicy",
        "iam:ListGroupsForUser",
        "iam:ListAttachedUserPolicies",
        "iam:ListUserPolicies",
        "iam:GetUser"
      ],
      "Resource": ["arn:aws:iam::*:user/${aws:username}"]
    },
    {
      "Sid": "NavigateInConsole",
      "Effect": "Allow",
      "Action": [
        "iam:GetGroupPolicy",
        "iam:GetPolicyVersion",
        "iam:GetPolicy",
        "iam:ListAttachedGroupPolicies",
        "iam:ListGroupPolicies",
        "iam:ListPolicyVersions",
        "iam:ListPolicies",
        "iam:ListUsers"
      ],
      "Resource": "*"
    }
  ]
}
```

## AWS 관리형 정책에 대한 Amazon EMR Serverless 업데이트

이 서비스가 이러한 변경 사항을 추적하기 시작한 이후부터 Amazon EMR Serverless의 AWS 관리형 정책 업데이트에 대한 세부 정보를 확인합니다. 이 페이지의 변경 사항에 대한 자동 알림을 받아보려면 Amazon EMR Serverless [문서 기록](#) 페이지에서 RSS 피드를 구독합니다.

변경 사항	설명	날짜
AmazonEMRServerlessServiceRolePolicy – 기존 정책에 대한 업데이트	Amazon EMR Serverless는 <a href="#">AmazonEMRServerlessServiceRolePolicy</a> 정책에 Sid CloudWatchPolicyStatement 및 EC2PolicyStatement 를 추가했습니다.	2024년 1월 25일
AmazonEMRServerlessServiceRolePolicy – 기존 정책에 대한 업데이트	Amazon EMR Serverless는 Amazon EMR Serverless가 "AWS/Usage" 네임스페이스에서 vCPU 사용량에 대한 집계된 계정 지표를 게시할 수 있는 새로운 권한을 추가했습니다.	2023년 4월 20일
Amazon EMR Serverless에서 변경 사항 추적 시작	Amazon EMR Serverless가 AWS 관리형 정책에 대한 변경 사항 추적을 시작했습니다.	2023년 4월 20일

## Amazon EMR Serverless 자격 증명 및 액세스 문제 해결

다음 정보를 사용하여 Amazon EMR Serverless 및 IAM으로 작업하는 경우 발생할 수 있는 일반적인 문제를 진단하고 수정할 수 있습니다.

### 주제

- [Amazon EMR Serverless에서 작업을 수행할 권한이 없음](#)
- [iam:PassRole을 수행하도록 인증되지 않음](#)
- [내 AWS 계정 외부의 사용자가 내 Amazon EMR Serverless 리소스에 액세스할 수 있도록 허용하려고 하는 경우](#)

- [EMR Studio에서 라이브 UI/Spark 기록 서버를 열어 작업을 디버깅할 수 없거나 `get-dashboard-for-job-run`을 사용하여 로그를 가져오려고 할 때 API 오류가 발생합니다.](#)

## Amazon EMR Serverless에서 작업을 수행할 권한이 없음

AWS Management 콘솔에서 작업을 수행할 권한이 없다는 메시지가 나타나는 경우 관리자에게 지원을 요청하세요. 관리자는 사용자 이름과 비밀번호를 제공한 사람입니다.

다음 예제 오류는 mateojackson 사용자가 콘솔을 사용하여 가상 *my-example-widget* 리소스에 대한 세부 정보를 확인하려고 하지만 가상 `emr-serverless:RedWidget` 권한이 없을 때 발생합니다.

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform: emr-serverless:RedWidget on resource: my-example-widget
```

이 경우 Mateo는 *my-example-widget* 작업을 사용하여 `emr-serverless:RedWidget` 리소스에 액세스하도록 허용하는 정책을 업데이트하라고 관리자에게 요청합니다.

## iam:PassRole을 수행하도록 인증되지 않음

`iam:PassRole` 작업을 수행할 수 있는 권한이 없다는 오류가 수신되면 Amazon EMR Serverless에 역할을 전달할 수 있도록 정책을 업데이트해야 합니다.

일부 AWS 서비스에서는 새로운 서비스 역할 또는 서비스 연결 역할을 생성하는 대신 해당 서비스에 기존 역할을 전달할 수 있습니다. 이렇게 하려면 사용자가 서비스에 역할을 전달할 수 있는 권한을 가지고 있어야 합니다.

다음 예제 오류는 marymajor라는 IAM 사용자가 콘솔을 사용하여 Amazon EMR Serverless에서 작업을 수행하려고 하는 경우에 발생합니다. 하지만 작업을 수행하려면 서비스 역할이 부여한 권한이 서비스에 있어야 합니다. Mary는 서비스에 역할을 전달할 수 있는 권한을 가지고 있지 않습니다.

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform: iam:PassRole
```

이 경우, Mary가 `iam:PassRole` 작업을 수행할 수 있도록 Mary의 정책을 업데이트해야 합니다.

도움이 필요한 경우 AWS 관리자에게 문의하세요. 관리자는 로그인 자격 증명을 제공한 사람입니다.

## 내 AWS 계정 외부의 사용자가 내 Amazon EMR Serverless 리소스에 액세스할 수 있도록 허용하려고 하는 경우

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 참조하세요.

- Amazon EMR Serverless에서 이러한 기능을 지원하는지 여부를 알아보려면 [Amazon EMR Serverless의 Identity and Access Management\(IAM\)](#) 섹션을 참조하세요.
- 소유하고 있는 AWS 계정의 리소스에 대한 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [자신이 소유한 다른 AWS 계정의 IAM 사용자에게 대한 액세스 권한 제공](#)을 참조하세요.
- 리소스에 대한 액세스 권한을 서드 파티 AWS 계정에게 제공하는 방법을 알아보려면 IAM 사용 설명서의 [서드 파티가 소유한 AWS 계정에 대한 액세스 제공](#)을 참조하세요.
- ID 페더레이션을 통해 액세스 권한을 제공하는 방법을 알아보려면 IAM 사용 설명서의 [외부에서 인증된 사용자에게 액세스 권한 제공\(ID 페더레이션\)](#)을 참조하세요.
- 교차 계정 액세스에 대한 역할과 리소스 기반 정책 사용의 차이점을 알아보려면 IAM 사용 설명서의 [IAM의 크로스 계정 리소스 액세스](#)를 참조하세요.

EMR Studio에서 라이브 UI/Spark 기록 서버를 열어 작업을 디버깅할 수 없거나 **get-dashboard-for-job-run**을 사용하여 로그를 가져오려고 할 때 API 오류가 발생합니다.

로깅에 EMR Serverless 관리형 스토리지를 사용하고 EMR Serverless 애플리케이션이 Amazon S3 용 VPC 엔드포인트가 있는 프라이빗 서브넷에 있고 엔드포인트 정책을 연결하여 액세스를 제어하는 경우 VPC 정책에서 [관리형 스토리지를 사용하여 EMR Serverless 로깅](#)에 언급된 권한을 EMR Serverless가 애플리케이션 로그를 저장하고 제공할 수 있도록 Amazon S3 게이트웨이 엔드포인트에 추가합니다.

## Amazon EMR Serverless를 사용한 신뢰할 수 있는 자격 증명 전파

Amazon EMR 릴리스 7.8.0 이상을 사용하면 Apache Livy 엔드포인트를 통해 EMR Serverless를 사용하여 AWS IAM Identity Center에서 대화형 방식 워크로드로 사용자 ID를 전파할 수 있습니다. Apache Livy 대화형 워크로드는 제공된 ID를 Amazon S3, Lake Formation 및 Amazon Redshift와 같은 다운스

트림 서비스로 추가로 전파하여, 이러한 다운스트림 환경에서 사용자 ID를 통한 안전한 데이터 접근을 가능하게 합니다. 다음 섹션에서는 EMR Serverless를 사용하여 Apache Livy 엔드포인트를 통해 대화형 방식 워크로드를 시작하고 전파하는 데 필요한 개념 개요, 사전 조건 및 단계를 제공합니다.

## 개요

[IAM Identity Center](#)는 규모와 유형에 관계없이 AWS에서 모든 조직의 인력 인증 및 권한 부여에 권장되는 접근 방식입니다. Identity Center를 사용하면 AWS에서 사용자 자격 증명을 생성 및 관리하거나 Microsoft Active Directory, Okta, Ping Identity, JumpCloud, Google Workspace 및 Microsoft Entra ID(이전의 Azure AD)를 포함한 기존 자격 증명 소스를 연결할 수 있습니다.

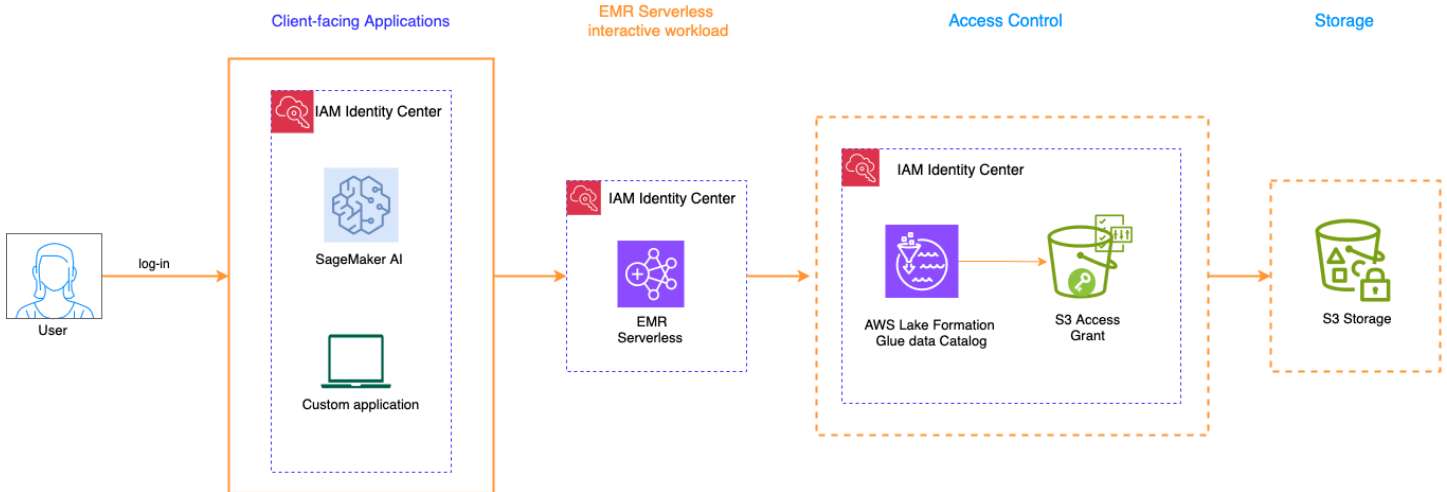
[신뢰할 수 있는 자격 증명 전파](#)는 연결된 AWS 서비스의 관리자가 서비스 데이터에 대한 액세스 권한을 부여하고 감사하는 데 사용할 수 있는 AWS IAM Identity Center 기능입니다. 이 데이터에 대한 액세스는 그룹 연결과 같은 사용자 속성을 기반으로 합니다. 신뢰할 수 있는 ID 전파를 설정하려면 연결된 AWS 서비스의 관리자와 IAM Identity Center 관리자 간의 협업이 필요합니다. 자세한 내용은 IAM Identity Center 사용 설명서의 [사전 조건 및 고려 사항](#)을 참조하세요.

## 기능 및 이점

EMR 서버리스 Apache Livy 엔드포인트와 IAM Identity Center [신뢰할 수 있는 자격 증명 전파의 통합](#)은 다음과 같은 이점을 제공합니다.

- AWS Lake Formation 관리형 AWS Glue 카탈로그 테이블에서 Identity Center 자격 증명을 사용하여 테이블 수준 인증을 적용할 수 있습니다.
- Amazon Redshift 클러스터에서 Identity Center ID로 권한 부여를 적용할 수 있습니다.
- 감사를 위해 사용자 작업의 엔드 투 엔드 추적이 가능합니다.
- S3 Access Grants 관리 S3 접두사에 대해 Identity Center 자격 증명을 사용하여 Amazon S3 접두사 수준 인증을 적용하는 기능입니다.

## 작동 방법



## 사용 사례 예제

### 데이터 준비 및 특성 엔지니어링

여러 연구 팀의 데이터 과학자는 통합 데이터 플랫폼을 사용하여 복잡한 프로젝트에서 협업합니다. 기업 인증 정보를 사용하여 SageMaker AI에 로그인하면, 즉시 여러 AWS 계정에 걸쳐 있는 방대한 공유 데이터 레이크에 접근할 수 있습니다. 새로운 기계 학습 모델에 대한 특성 엔지니어링을 시작하면 EMR Serverless를 통해 시작된 Spark 세션은 전파된 ID를 기반으로 Lake Formation의 열 및 행 수준 보안 정책을 적용합니다. 과학자는 익숙한 도구를 사용하여 데이터를 효율적으로 준비하고 특성을 엔지니어링할 수 있으며, 규정 준수 팀은 모든 데이터 상호 작용이 자동으로 추적 및 감사되도록 보장합니다. 이 안전한 협업 환경은 규제가 적용되는 산업에 필요한 엄격한 데이터 보호 표준을 유지하면서 연구 파이프라인을 가속화합니다.

## EMR Serverless의 신뢰된 자격 증명 전파 시작하기

이 섹션에서는 AWS IAM Identity Center와 통합하고 [신뢰할 수 있는 자격 증명 전파](#)를 활성화하도록 Apache Livy Endpoint로 EMR-Serverless 애플리케이션을 구성하는 데 도움이 됩니다.

### 사전 조건

- 신뢰할 수 있는 자격 증명 전파를 사용하도록 설정할 AWS 리전 내 Identity Center 인스턴스(EMR Serverless Apache Livy 엔드포인트)를 만들려고 합니다. Identity Center 인스턴스는 AWS 계정의 단일 리전에만 존재할 수 있습니다. [IAM Identity Center 활성화](#) 및 [ID 소스에서 IAM Identity Center로 사용자 및 그룹 프로비저닝](#)을 참조하세요.

- 대화형 워크로드가 상호 작용하여 데이터에 액세스하는 Lake Formation, S3 Access Grants 또는 Amazon Redshift 클러스터 등의 다운스트림 서비스에 대해 신뢰할 수 있는 자격 증명 전파를 활성화합니다.

신뢰할 수 있는 자격 증명 전파가 활성화된 EMR Serverless 애플리케이션 생성 시 필요한 권한

[EMR Serverless에 액세스하는 데 필요한 기본 권한](#) 외에도 신뢰할 수 있는 자격 증명 전파가 활성화된 EMR Serverless 애플리케이션을 만드는 데 사용되는 IAM ID 또는 역할에 대한 추가 권한을 구성해야 합니다. 신뢰할 수 있는 자격 증명 전파를 위해 EMR Serverless는 계정에서 단일 서비스 관리형 Identity Center 애플리케이션을 생성/부트스트랩하며, 이 서비스는 자격 증명 유효성 검사 및 다운스트림으로의 자격 증명 전파를 위해 활용합니다.

```
"sso:DescribeInstance",
"sso:CreateApplication",
"sso:DeleteApplication",
"sso:PutApplicationAuthenticationMethod",
"sso:PutApplicationAssignmentConfiguration",
"sso:PutApplicationGrant",
"sso:PutApplicationAccessScope"
```

- `sso:DescribeInstance` - `identity-center-configuration` 파라미터에 지정한 IAM Identity Center `instanceArn`을 설명하고 유효성을 검사할 수 있는 권한을 부여합니다.
- `sso:CreateApplication` - `trusted-identity-propatgion` 작업에 사용되는 EMR Serverless 관리형 IAM Identity Center 애플리케이션을 생성할 수 있는 권한을 부여합니다.
- `sso:DeleteApplication` - EMR Serverless 관리형 IAM Identity Center 애플리케이션을 정리할 수 있는 권한을 부여합니다.
- `sso:PutApplicationAuthenticationMethod` - `emr-serverless` 서비스 보안 주체가 IAM Identity Center 애플리케이션과 상호 작용할 수 있도록 EMR Serverless 관리형 IAM Identity Center 애플리케이션에 `authenticationMethod`를 배치할 수 있는 권한을 부여합니다.
- `sso:PutApplicationAssignmentConfiguration` - IAM Identity Center 애플리케이션에서 `"User-assignment-not-required"`를 설정할 수 있는 권한을 부여합니다.
- `sso:PutApplicationGrant` - IAM Identity Center 애플리케이션에 `token-exchange`, `introspectToken`, `refreshToken` 및 `revokeToken` 권한을 적용할 수 있는 권한을 부여합니다.
- `sso:PutApplicationAccessScope` - 신뢰할 수 있는 자격 증명 전파가 활성화된 다운스트림 범위를 IAM Identity Center 애플리케이션에 적용할 수 있는 권한을 부여합니다. 당사는

"redshift:connect", "lakeformation:query" 및 "s3:read\_write" 범위를 적용하여 이러한 서비스에 대한 trusted-identity-propagation을 활성화합니다.

## 신뢰할 수 있는 자격 증명 전파가 활성화된 EMR Serverless 애플리케이션 생성

애플리케이션에서 신뢰할 수 있는 자격 증명 전파를 사용하려면 `--identity-center-configuration` 필드에 `identityCenterInstanceArn`을 지정해야 합니다. 다음 예시 명령을 사용하여 신뢰할 수 있는 자격 증명 전파가 활성화된 EMR Serverless 애플리케이션을 생성합니다.

### Note

또한 Apache Livy 엔드포인트에 대해서만 신뢰할 수 있는 자격 증명 전파가 활성화되도록 `--interactive-configuration '{"livyEndpointEnabled":true}'`를 지정합니다.

```
aws emr-serverless create-application \
  --release-label emr-7.8.0 \
  --type "SPARK" \
  --identity-center-configuration '{"identityCenterInstanceArn" :
  "arn:aws:sso:::instance/ssoins-123456789"}' \
  --interactive-configuration '{"livyEndpointEnabled":true}'
```

- `identity-center-configuration` – (선택 사항) 지정된 경우 Identity Center가 신뢰할 수 있는 자격 증명 전파를 활성화합니다.
- `identityCenterInstanceArn` – (필수) Identity Center 인스턴스 ARN입니다.

필요한 Identity Center 권한(이전에 언급됨)이 없는 경우 먼저 신뢰할 수 있는 자격 증명 전파 없이 EMR Serverless 애플리케이션을 생성하고(예: `--identity-center-configuration` 파라미터를 지정하지 않음) 나중에 Identity Center 관리자에게 `update-application` API를 호출하여 신뢰할 수 있는 자격 증명 전파를 활성화하도록 요청합니다. 아래 예제를 참조하세요.

```
aws emr-serverless update-application \
  --application-id applicationId \
  --identity-center-configuration '{"identityCenterInstanceArn" :
  "arn:aws:sso:::instance/ssoins-123456789"}'
```

EMR Serverless는 사용자 계정에 서비스 관리형 Identity Center 애플리케이션을 생성하여 자격 증명 및 다운스트림 서비스로의 자격 증명 전파에 활용합니다. EMR Serverless에서 생성한 관리형 Identity

Center 애플리케이션은 사용자 계정의 신뢰할 수 trusted-identity-propagation가 활성화된 모든 EMR Serverless 애플리케이션에서 공유됩니다.

#### Note

관리형 Identity Center 애플리케이션의 설정을 수동으로 수정하지 마세요. 변경 사항은 계정의 모든 신뢰할 수 있는 자격 증명 전파가 활성화된 EMR Serverless 애플리케이션에 영향을 미칠 수 있습니다.

## 자격 증명 전파를 위한 작업 실행 역할 권한

EMR-Serverless는 자격 증명 강화 job-execution-role 자격 증명을 활용하여 자격 증명을 다운스트림 AWS 서비스에 전파하므로 작업 실행 역할의 신뢰 정책은 S3 액세스 권한 부여, Lake Formation 또는 Amazon Redshift와 같은 다운스트림 서비스에 trusted-identity-propagation를 허용하도록 자격 증명으로 작업 실행 역할 자격 증명을 향상시킬 수 있는 추가 권한 sts:SetContext가 있어야 합니다. 역할을 생성하는 방법에 대한 자세한 내용은 [작업 런타임 역할 생성](#)을 참조하세요.

### JSON

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

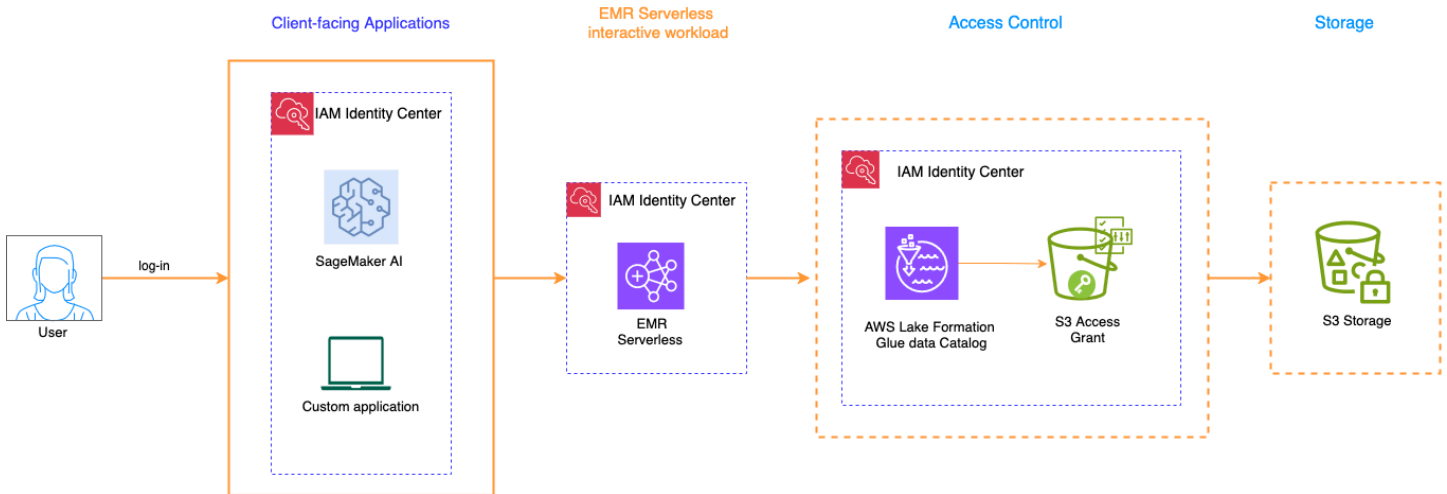
추가로 JobExecutionRole에는 사용자 ID를 사용하여 데이터를 가져오기 위해 작업 실행에서 호출하는 다운스트림 AWS 서비스에 대한 권한이 필요합니다. S3 Access Grant, Lake Formation을 구성하려면 아래 링크를 참조하세요.

- [EMR Serverless와 함께 Lake Formation 사용](#)

- [EMR Serverless에서 Amazon S3 Access Grants 사용](#)

## EMR Serverless를 사용한 대화형 방식 워크로드의 신뢰할 수 있는 자격 증명 전파

Apache Livy 엔드포인트를 통해 대화형 워크로드에 ID를 전파하는 단계는 사용자가 Amazon SageMaker AI와 같은 AWS 관리형 개발 환경과 상호작용하는지 아니면 클라이언트 대면 애플리케이션으로서 자체 호스팅된 노트북 환경과 상호작용하는지에 따라 달라집니다.



## AWS 관리형 개발 환경

다음 AWS 관리형 클라이언트 연결 애플리케이션은 EMR-Serverless Apache Livy 엔드포인트를 통한 신뢰할 수 있는 자격 증명 전파를 지원합니다.

- [Amazon SageMaker AI](#)

## 고객 관리형 자체 호스팅 노트북 환경

사용자 지정 개발 애플리케이션의 사용자에게 신뢰할 수 있는 자격 증명 전파를 활성화하려면 AWS 보안 블로그에서 [신뢰할 수 있는 자격 증명 전파를 사용하여 프로그래밍 방식으로 AWS 서비스 액세스](#)를 참조하세요.

## EMR Serverless 신뢰 자격 증명 전파 통합 시 고려 사항 및 제한 사항

EMR Serverless 애플리케이션과 함께 IAM Identity Center 신뢰할 수 있는 자격 증명 전파를 사용할 때 다음 사항을 고려하세요.

- Identity Center를 통한 신뢰할 수 있는 자격 증명 전파는 Amazon EMR 7.8.0 이상에서 지원되며 Apache Spark에서만 지원됩니다.
- 신뢰할 수 있는 자격 증명 전파는 [Apache Livy 엔드포인트를 통해 EMR Serverless를 사용하는 대화형 방식 워크로드](#)에만 사용할 수 있습니다. EMR Studio를 통한 대화형 방식 워크로드는 신뢰할 수 있는 자격 증명 전파를 지원하지 않습니다.
- 배치 작업 및 스트리밍 작업은 trusted-identity-propagation을 지원하지 않습니다.
- 신뢰할 수 있는 자격 증명 전파를 사용하는 AWS Lake Formation을 사용하는 세분화된 액세스 제어는 [Apache Livy 엔드포인트를 통해 EMR Serverless를 사용하는 대화형 워크로드](#)에 사용할 수 있습니다.
- Amazon EMR을 통한 신뢰할 수 있는 자격 증명 전파는 다음 AWS 리전에서 지원됩니다.
  - af-south-1 – 아프리카(케이프타운)
  - ap-east-1 – 아시아 태평양(홍콩)
  - ap-northeast-1 – 아시아 태평양 (도쿄)
  - ap-northeast-2 - 아시아 태평양(서울)
  - ap-northeast-3 – 아시아 태평양(오사카)
  - ap-south-1 - 미국 서부(캘리포니아 북부)
  - ap-southeast-1 – 아시아 태평양(싱가포르)
  - ap-southeast-2 – 아시아 태평양(시드니)
  - ap-southeast-3 – 아시아 태평양(자카르타)
  - ca-central-1 - 캐나다(중부)
  - ca-west-1 – 캐나다(캘거리)
  - eu-central-1 – EU(프랑크푸르트)
  - eu-north-1 - 유럽(스톡홀름)
  - eu-south-1 – 유럽(밀라노)
  - eu-south-2 – 유럽(스페인)
  - eu-west-1 – EU(아일랜드)
  - eu-west-2 - 유럽(런던)
  - eu-west-3 - 유럽(파리)
  - me-central-1 – 중동(UAE)
  - me-south-1 – 중동(바레인)

- us-east-1 – 미국 동부(버지니아 북부)
- us-east-2 – 미국 동부(오하이오)
- us-west-1 – 미국 서부(캘리포니아 북부)
- us-west-2 – 미국 서부(오리건)

## EMR Serverless와 함께 Lake Formation 사용

Lake Formation을 사용하여 세분화된 액세스 제어를 제공하는 방법을 알아봅니다.

### EMR Serverless에 대한 Lake Formation 필터링되지 않은 액세스

Amazon EMR 릴리스 7.8.0 이상에서는 작업 런타임 역할에 전체 테이블 권한이 있는 Glue Data Catalog와 함께 AWS Lake Formation을 세분화된 액세스 제어 제한 없이 활용할 수 있습니다. 이 기능을 사용하면 EMR Serverless Spark 배치 및 대화형 방식 작업에서 Lake Formation으로 보호되는 테이블을 읽고 쓸 수 있습니다. Lake Formation 및 EMR Serverless와 함께 사용하는 방법에 대해 자세히 알아보려면 다음 섹션을 참조하세요.

#### 전체 테이블 액세스 권한으로 Lake Formation 사용

EMR Serverless Spark 작업 또는 작업의 런타임 역할에 전체 테이블 액세스 권한이 있는 대화형 세션에서 AWS Lake Formation 보호 Glue Data Catalog 테이블에 액세스할 수 있습니다. EMR Serverless 애플리케이션에서 AWS Lake Formation을 활성화할 필요가 없습니다. Spark 작업이 전체 테이블 액세스(FTA)로 구성된 경우 AWS Lake Formation 자격 증명은 AWS Lake Formation 등록 테이블에 대한 Amazon S3 데이터를 읽거나 쓰는 데 사용되는 반면, 작업의 런타임 역할 자격 증명은 AWS Lake Formation에 등록되지 않은 테이블을 읽거나 쓰는 데 사용됩니다.

#### Important

세분화된 액세스 제어를 위해 AWS Lake Formation을 활성화하지 마세요. 작업은 전체 테이블 액세스(FTA)와 세분화된 액세스 제어(FGAC)를 동일한 EMR 클러스터 또는 애플리케이션에서 실행할 수 없습니다.

#### 1단계: Lake Formation에서 전체 테이블 액세스 활성화

전체 테이블 액세스(FTA) 모드를 사용하려면 AWS Lake Formation에서 IAM 세션 태그 검증 없이 타사 쿼리 엔진이 데이터에 액세스하도록 허용해야 합니다. 활성화하려면 [Application integration for full table access](#)의 단계를 따릅니다.

**Note**

교차 계정 테이블에 액세스할 때는 생산자와 소비자 계정 모두에서 전체 테이블 액세스를 활성화합니다. 동일한 방식으로 교차 리전 테이블에 액세스할 때 생산자와 소비자 리전 모두에서 이 설정을 활성화합니다.

**2단계: 작업 런타임 역할에 대한 IAM 권한 설정**

기본 데이터에 대한 읽기 또는 쓰기 액세스의 경우 작업 런타임 역할에는 Lake Formation 권한 외에도 `lakeformation:GetDataAccess` IAM 권한이 필요합니다. 이 권한을 통해 Lake Formation은 데이터에 액세스하기 위한 임시 보안 인증 요청을 승인합니다.

다음은 Amazon S3의 스크립트에 액세스할 수 있는 IAM 권한을 제공하는 방법, S3에 로그 업로드, AWS Glue API 권한 및 Lake Formation에 액세스할 수 있는 권한에 대한 정책 예제입니다.

**JSON**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ScriptAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3::*.amzn-s3-demo-bucket/scripts"
      ]
    },
    {
      "Sid": "LoggingAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/logs/*"
      ]
    }
  ],
}
```

```

{
  "Sid": "GlueCatalogAccess",
  "Effect": "Allow",
  "Action": [
    "glue:Get*",
    "glue:Create*",
    "glue:Update*"
  ],
  "Resource": [
    "*"
  ]
},
{
  "Sid": "LakeFormationAccess",
  "Effect": "Allow",
  "Action": [
    "lakeformation:GetDataAccess"
  ],
  "Resource": [
    "*"
  ]
}
]
}

```

## 2.1단계 Lake Formation 권한 구성

- S3에서 데이터를 읽는 Spark 작업에는 Lake Formation SELECT 권한이 필요합니다.
- S3에서 데이터를 쓰거나 삭제하는 Spark 작업에는 Lake Formation ALL(SUPER) 권한이 필요합니다.
- Glue Data Catalog와 상호 작용하는 Spark 작업에는 DESCRIBE, ALTER, DROP 권한이 필요합니다.

자세한 내용은 [데이터 카탈로그 리소스에 권한 부여](#)를 참조하세요.

3단계: Lake Formation을 사용하여 전체 테이블 액세스를 위한 Spark 세션 초기화

## 사전 조건

AWS Lake Formation 테이블에 액세스하려면 Glue Data Catalog를 메타스토어로 구성합니다.

설정을 다음과 같이 설정하여 Glue 카탈로그를 메타스토어로 구성합니다.

```
--conf spark.sql.catalogImplementation=hive
--conf
  spark.hive.metastore.client.factory.class=com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalog
```

EMR Serverless용 데이터 카탈로그 활성화에 대한 자세한 내용은 [EMR Serverless용 메타스토어 구성](#)을 참조하세요.

AWS Lake Formation에 등록된 테이블에 액세스하려면 Spark 초기화 중에 다음 구성을 설정하여 AWS Lake Formation 자격 증명을 사용하도록 Spark를 구성해야 합니다.

## Hive

```
--conf
  spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormation
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

## Iceberg

```
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.warehouse=S3_DATA_LOCATION
--conf spark.sql.catalog.spark_catalog.client.region=REGION
--conf spark.sql.catalog.spark_catalog.type=glue
--conf spark.sql.catalog.spark_catalog.glue.account-id=ACCOUNT_ID
--conf spark.sql.catalog.spark_catalog.glue.lakeformation-enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

## Delta Lake

```
--conf
  spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.AWSLakeFormation
--conf spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true
--conf spark.hadoop.fs.s3.folderObject.autoAction.disabled=true
--conf spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true
--conf spark.sql.catalog.createDirectoryAfterTable.enabled=true
--conf spark.sql.catalog.dropDirectoryBeforeTable.enabled=true
```

- `spark.hadoop.fs.s3.credentialsResolverClass=com.amazonaws.glue.accesscontrol.A`  
Lake Formation에 등록된 테이블에 대해 AWS Lake Formation S3 자격 증명을 사용하도록 EMR 파  
일 시스템(EMRFS) 또는 EMR S3A를 구성합니다. 테이블이 등록되지 않은 경우 작업의 런타임 역할  
자격 증명을 사용합니다.
- `spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true` 및  
`spark.hadoop.fs.s3.folderObject.autoAction.disabled=true`: S3 폴더를 생성할 때  
\$folder\$ 접미사 대신 콘텐츠 유형 헤더 `application/x-directory`를 사용하도록 EMRFS를 구성합니다.  
Lake Formation 자격 증명은 \$folder\$ 접미사가 있는 테이블 폴더 읽기를 허용하지 않으므로 Lake  
Formation 테이블을 읽을 때 필요합니다.
- `spark.sql.catalog.skipLocationValidationOnCreateTable.enabled=true`: 생성 전  
에 테이블 위치가 비어 있는지 확인하는 작업을 건너뛰도록 Spark를 구성합니다. Lake Formation 등  
록 테이블에서는 필수입니다. 위치가 비어 있는지 확인하기 위한 Lake Formation 자격 증명은 Glue  
Data Catalog 테이블을 생성한 후에만 사용할 수 있기 때문입니다. 이 구성이 없으면 작업의 런타임  
역할 자격 증명에 빈 테이블 위치를 검증합니다.
- `spark.sql.catalog.createDirectoryAfterTable.enabled=true`: Hive 메타스토어에서  
테이블 생성 후 Amazon S3 폴더를 생성하도록 Spark를 구성합니다. Lake Formation 등록 테이블에  
서는 필수입니다. S3 폴더를 생성하기 위한 Lake Formation 자격 증명은 Glue Data Catalog 테이블  
을 생성한 후에만 사용할 수 있기 때문입니다.
- `spark.sql.catalog.dropDirectoryBeforeTable.enabled=true`: Hive 메타스토어에  
서 테이블 삭제 전 S3 폴더를 제거하도록 Spark를 구성합니다. Lake Formation 등록 테이블에서  
는 필수입니다. Glue Data Catalog에서 테이블을 삭제한 후에는 S3 폴더를 삭제하기 위한 Lake  
Formation 자격 증명을 사용할 수 없기 때문입니다.
- `spark.sql.catalog.<catalog>.glue.lakeformation-enabled=true`: Lake Formation  
등록 테이블에 AWS Lake Formation S3 자격 증명을 사용하도록 Iceberg 카탈로그를 구성합니다.  
테이블이 등록되지 않은 경우 기본 환경 자격 증명을 사용합니다.

## SageMaker Unified Studio에서 전체 테이블 액세스 모드 구성

JupyterLab 노트북의 대화형 Spark 세션에서 Lake Formation 등록 테이블에 액세스하려면 호환성 권  
한 모드를 사용합니다. `%%configure` 매직 명령을 사용하여 Spark 구성을 설정합니다. 테이블 유형을  
기반으로 구성 선택:

### For Hive tables

```
%%configure -f
{
```

```

    "conf": {
      "spark.hadoop.fs.s3.credentialsResolverClass":
"com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialResolver",
      "spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject": true,
      "spark.hadoop.fs.s3.folderObject.autoAction.disabled": true,
      "spark.sql.catalog.skipLocationValidationOnCreateTable.enabled": true,
      "spark.sql.catalog.createDirectoryAfterTable.enabled": true,
      "spark.sql.catalog.dropDirectoryBeforeTable.enabled": true
    }
  }
}

```

## For Iceberg tables

```

%%configure -f
{
  "conf": {
    "spark.sql.catalog.spark_catalog":
"org.apache.iceberg.spark.SparkSessionCatalog",
    "spark.sql.catalog.spark_catalog.warehouse": "S3_DATA_LOCATION",
    "spark.sql.catalog.spark_catalog.client.region": "REGION",
    "spark.sql.catalog.spark_catalog.type": "glue",
    "spark.sql.catalog.spark_catalog.glue.account-id": "ACCOUNT_ID",
    "spark.sql.catalog.spark_catalog.glue.lakeformation-enabled": "true",
    "spark.sql.catalog.dropDirectoryBeforeTable.enabled": "true",
  }
}

```

## For Delta Lake tables

```

%%configure -f
{
  "conf": {
    "spark.hadoop.fs.s3.credentialsResolverClass":
"com.amazonaws.glue.accesscontrol.AWSLakeFormationCredentialResolver",
    "spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject": true,
    "spark.hadoop.fs.s3.folderObject.autoAction.disabled": true,
    "spark.sql.catalog.skipLocationValidationOnCreateTable.enabled": true,
    "spark.sql.catalog.createDirectoryAfterTable.enabled": true,
    "spark.sql.catalog.dropDirectoryBeforeTable.enabled": true
  }
}

```

### 자리 표시자 바꾸기:

- S3\_DATA\_LOCATION: S3 버킷 경로
- REGION: AWS 리전(예: us-east-1)
- ACCOUNT\_ID: AWS계정 ID

#### Note

노트북에서 Spark 작업을 실행하기 전에 이러한 구성을 설정해야 합니다.

### 지원 작업

이러한 작업은 AWS Lake Formation 자격 증명을 사용하여 테이블 데이터에 액세스합니다.

- CREATE TABLE
- ALTER TABLE
- INSERT INTO
- INSERT OVERWRITE
- UPDATE
- MERGE INTO
- DELETE FROM
- ANALYZE TABLE
- REPAIR TABLE
- DROP TABLE
- Spark 데이터 소스 쿼리
- Spark 데이터 소스 쓰기

#### Note

위에 나열되지 않은 작업은 계속해서 IAM 권한을 사용하여 테이블 데이터에 액세스합니다.

## 고려 사항

- 전체 테이블 액세스가 활성화되지 않은 작업을 사용하여 Hive 테이블이 생성되고, 삽입된 레코드가 없는 경우 전체 테이블 액세스를 사용하는 작업의 후속 읽기 또는 쓰기는 실패합니다. 전체 테이블 액세스 권한이 없는 EMR Spark가 \$folder\$ 접미사를 테이블 폴더 이름에 추가하기 때문입니다. 이 문제는 다음과 같은 방법으로 해결할 수 있습니다.
- FTA가 활성화되지 않은 작업의 테이블에 1개 이상의 행을 삽입합니다.
- S3의 폴더 이름에 \$folder\$ 접미사를 사용하지 않도록 FTA가 활성화되지 않은 작업을 구성합니다. Spark 구성 `spark.hadoop.fs.s3.useDirectoryHeaderAsFolderObject=true`를 설정하면 가능합니다.
- AWS S3 콘솔 또는 AWS S3 CLI를 사용하여 테이블 위치 `s3://path/to/table/table_name`에 S3 폴더를 생성합니다.
- 전체 테이블 액세스는 Amazon EMR 릴리스 7.8.0부터 EMR 파일 시스템(EMRFS)에서 지원되며, Amazon EMR 릴리스 7.10.0부터 S3A 파일 시스템에서 지원됩니다.
- Hive, Iceberg 및 Delta 테이블의 경우 전체 테이블 액세스가 지원됩니다. Hudi 테이블은 전체 테이블 액세스를 지원하지 않습니다.
- Formation Fine 세분 영역 액세스 제어(FGAC) 규칙 또는 Glue Data Catalog 뷰가 있는 테이블을 참조하는 작업은 실패합니다. FGAC 규칙 또는 Glue Data Catalog 뷰로 테이블을 쿼리하려면 FGAC 모드를 사용해야 합니다. AWS 설명서: [세분화된 액세스 제어를 위해 EMR Serverless와 AWS Lake Formation 사용](#)에서 설명하는 단계에 따라 FGAC 모드를 활성화할 수 있습니다.
- 전체 테이블 액세스는 Spark 스트리밍을 지원하지 않습니다.
- Lake Formation 테이블에 Spark DataFrame을 쓰는 경우 Hive 및 Iceberg 테이블 `df.write.mode("append").saveAsTable(table_name)`에는 APPEND 모드만 지원됩니다.
- 외부 테이블을 생성하려면 IAM 권한이 필요합니다.
- Lake Formation은 Spark 작업 내에서 자격 증명을 임시로 캐시하므로 현재 실행 중인 Spark 배치 작업 또는 대화형 방식 세션에는 권한 변경 내용이 반영되지 않을 수 있습니다.
- 역할에 대한 서비스 연결 역할: [Lake Formation 요구 사항](#)이 아닌 사용자 정의 역할을 사용합니다.

# 세분화된 액세스 제어를 위해 AWS Lake Formation와 함께 EMR Serverless 사용

## 개요

Amazon EMR 릴리스 7.2.0 이상을 사용하면 AWS Lake Formation을 사용하여 S3에서 지원하는 Data Catalog 테이블에 세분화된 액세스 제어를 적용할 수 있습니다. 이 기능을 사용하면 Amazon EMR Serverless Spark 작업 내에서 read 쿼리에 대한 테이블, 행, 열 및 셀 수준 액세스 제어를 구성할 수 있습니다. Apache Spark 배치 작업 및 대화형 세션에 대한 세분화된 액세스 제어를 구성하려면 EMR Studio를 사용합니다. Lake Formation 및 EMR Serverless와 함께 사용하는 방법에 대해 자세히 알아보려면 다음 섹션을 참조하세요.

Amazon EMR Serverless를 AWS Lake Formation과 함께 사용하면 추가 요금이 발생합니다. 자세한 내용은 [Amazon EMR 요금](#)을 참조하세요.

## EMR Serverless에서 AWS Lake Formation과의 작동 방식

EMR Serverless를 Lake Formation과 함께 사용하면 EMR Serverless에서 작업을 실행하는 경우 Lake Formation 권한 제어를 적용하기 위해 각 Spark 작업에 권한 계층을 적용할 수 있습니다. EMR Serverless는 [Spark 리소스 프로파일](#)을 사용하여 작업을 효과적으로 실행하도록 두 개의 프로파일을 생성합니다. 사용자 프로파일은 사용자 제공 코드를 실행하는 반면, 시스템 프로파일은 Lake Formation 정책을 적용합니다. 자세한 내용은 [AWS Lake Formation](#) 및 [고려 사항 및 제한 사항](#)을 참조하세요.

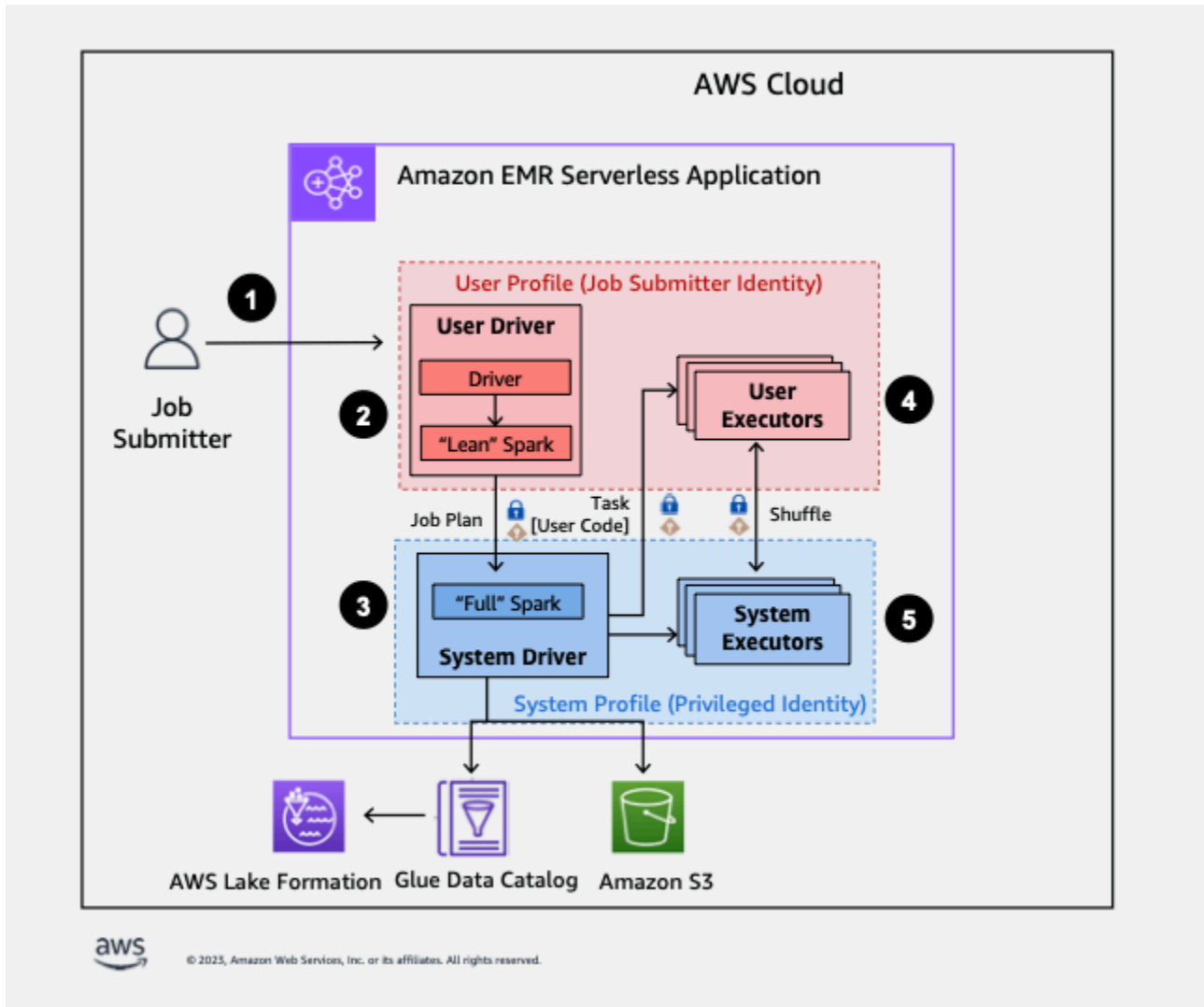
Lake Formation에서 사전 초기화된 용량을 사용하는 경우 최소 2개의 Spark 드라이버를 사용하는 것이 좋습니다. Lake Formation이 활성화된 각 작업에는 사용자 프로파일과 시스템 프로파일에 대해 하나씩, 두 개의 Spark 드라이버가 사용됩니다. 최상의 성능을 위해 Lake Formation을 사용하지 않는 경우와 비교하여 Lake Formation 지원 작업에 대해 2배의 드라이버 수를 사용합니다.

EMR Serverless에서 Spark 작업을 실행하는 경우 동적 할당이 리소스 관리 및 클러스터 성능에 미치는 영향도 고려합니다. 리소스 프로파일당 최대 실행기 수의 `spark.dynamicAllocation.maxExecutors` 구성은 사용자 및 시스템 실행기 모두에 적용됩니다. 이 숫자를 허용되는 최대 실행기 수와 같도록 구성하면 사용 가능한 모든 리소스를 사용하는 한 가지 유형의 실행기로 인해 작업 실행이 중단될 수 있으며, 이로 인해 작업 실행 시 다른 실행기가 작동하지 않습니다.

따라서 리소스가 부족하지 않도록 EMR Serverless는 리소스 프로파일당 기본 최대 실행기 수를 `spark.dynamicAllocation.maxExecutors` 값의 90%로 설정합니다. 0에서 1 사이의 값으로 `spark.dynamicAllocation.maxExecutorsRatio`를 지정하는 경우 이 구성을 재정의할 수 있습니다. 또한 리소스 할당 및 전체 성능을 최적화하도록 다음 속성을 구성할 수도 있습니다.

- `spark.dynamicAllocation.cachedExecutorIdleTimeout`
- `spark.dynamicAllocation.shuffleTracking.timeout`
- `spark.cleaner.periodicGC.interval`

다음은 Amazon EMR Serverless가 Lake Formation 보안 정책에 따라 보호되는 데이터에 액세스하는 방법에 대한 개략적인 개요입니다.



1. 사용자가 Spark 작업을 AWS Lake Formation 지원 EMR Serverless 애플리케이션에 제출합니다.
2. EMR Serverless는 사용자 드라이버로 작업을 전송하고 사용자 프로파일에서 작업을 실행합니다. 사용자 드라이버는 태스크를 시작하고, 실행기를 요청하며, S3 또는 Glue Catalog에 액세스할 수 없는 린 버전의 Spark를 실행합니다. 작업 계획을 빌드합니다.
3. EMR Serverless는 시스템 드라이버라는 두 번째 드라이버를 설정하고 시스템 프로파일(권한 있는 자격 증명 포함)에서 실행합니다. EMR Serverless는 통신을 위해 두 드라이버 사이에서 암호화

된 TLS 채널을 설정합니다. 사용자 드라이버는 채널을 사용하여 작업 계획을 시스템 드라이버로 전송합니다. 시스템 드라이버는 사용자가 제출한 코드를 실행하지 않습니다. 전체 Spark를 실행하고 S3 및 데이터 액세스를 위해 Data Catalog와 통신합니다. 실행기를 요청하고 작업 계획을 일련의 실행 단계로 컴파일합니다.

4. 그런 다음, EMR Serverless는 사용자 드라이버 또는 시스템 드라이버를 사용하여 실행기에서 단계를 실행합니다. 모든 단계의 사용자 코드는 사용자 프로파일 실행기에서만 실행됩니다.
5. AWS Lake Formation에서 보호하는 Data Catalog 테이블에서 데이터를 읽는 단계 또는 보안 필터를 적용하는 단계는 시스템 실행기로 위임됩니다.

## Amazon EMR에서 Lake Formation 활성화

Lake Formation을 활성화하려면 [EMR Serverless 애플리케이션을 생성](#)할 때 런타임 구성 파라미터의 `spark-defaults` 분류 아래에서 `spark.emr-serverless.lakeformation.enabled`를 `true`로 설정합니다.

```
aws emr-serverless create-application \
  --release-label emr-7.10.0 \
  --runtime-configuration '{
    "classification": "spark-defaults",
    "properties": {
      "spark.emr-serverless.lakeformation.enabled": "true"
    }
  }' \
  --type "SPARK"
```

EMR Studio에서 새 애플리케이션을 생성하는 경우 Lake Formation을 활성화할 수도 있습니다. 추가 구성 아래에서 사용할 수 있는 세분화된 액세스 제어를 위해 Lake Formation 사용을 선택합니다.

Lake Formation을 EMR Serverless와 함께 사용하면 [위커 간 암호화](#)가 기본적으로 활성화되므로 위커 간 암호화를 다시 명시적으로 활성화하지 않아도 됩니다.

### Spark 작업에 대해 Lake Formation 활성화

개별 Spark 작업에 대해 Lake Formation을 활성화하려면 `spark-submit`을 사용할 때 `spark.emr-serverless.lakeformation.enabled`를 `true`로 설정합니다.

```
--conf spark.emr-serverless.lakeformation.enabled=true
```

## 작업 런타임 역할 IAM 권한

Lake Formation 권한은 AWS Glue Data Catalog 리소스, Amazon S3 위치 및 해당 위치의 기본 데이터에 대한 액세스를 제어합니다. IAM 권한은 Lake Formation 및 AWS Glue API와 리소스에 대한 액세스를 제어합니다. Data Catalog의 테이블에 액세스할 수 있는 Lake Formation 권한이 있더라도 (SELECT) glue:Get\* API 작업에 IAM 권한이 없으면 작업이 실패합니다.

다음은 S3의 스크립트에 액세스할 수 있는 IAM 권한을 제공하는 방법, S3에 로그 업로드, AWS Glue API 권한 및 Lake Formation에 액세스할 수 있는 권한에 대한 정책 예제입니다.

### JSON

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "ScriptAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::*.amzn-s3-demo-bucket/scripts",
        "arn:aws:s3:::*.amzn-s3-demo-bucket/*"
      ]
    },
    {
      "Sid": "LoggingAccess",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::amzn-s3-demo-bucket/logs/*"
      ]
    },
    {
      "Sid": "GlueCatalogAccess",
      "Effect": "Allow",
      "Action": [
        "glue:Get*",

```

```

        "glue:Create*",
        "glue:Update*"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Sid": "LakeFormationAccess",
    "Effect": "Allow",
    "Action": [
        "lakeformation:GetDataAccess"
    ],
    "Resource": [
        "*"
    ]
}
]
}

```

## 작업 런타임 역할에 대한 Lake Formation 권한 설정

먼저 Lake Formation에 Hive 테이블의 위치를 등록합니다. 그런 다음, 원하는 테이블에서 작업 런타임 역할에 대한 권한을 생성합니다. Lake Formation에 대한 자세한 내용은 [AWS Lake Formation 개발자 안내서의 AWS Lake Formation이란?](#)을 참조하세요.

Lake Formation 권한을 설정한 후 Amazon EMR Serverless에서 Spark 작업을 제출할 수 있습니다. Spark 작업에 대한 자세한 내용은 [Spark 예제](#)를 참조하세요.

## 작업 실행 제출

Lake Formation 권한 부여 설정을 완료한 후 [EMR Serverless에서 Spark 작업을 제출](#)할 수 있습니다. 다음 섹션에서는 작업 실행 속성을 구성 및 제출하는 방법의 예를 보여줍니다.

## 오픈 테이블 형식 지원

EMR Serverless는 Apache Hive, Apache Iceberg를 지원하며 릴리스 7.6.0부터는 Delta Lake 및 Apache Hudi를 지원합니다. 작업 지원에 대한 내용은 다음 탭을 참조하세요.

## Hive

작업	참고
읽기 작업	완전 지원
중분 쿼리	해당 사항 없음
시간 이동 쿼리	이 테이블 형식에 해당되지 않음
DML INSERT	IAM 권한만 있음
DML UPDATE	이 테이블 형식에 해당되지 않음
DML DELETE	이 테이블 형식에 해당되지 않음
DDL 명령	IAM 권한만 있음
메타데이터 테이블	이 테이블 형식에 해당되지 않음
저장 프로시저	해당 사항 없음
테이블 유지 관리 및 유틸리티 기능	해당 사항 없음

## Iceberg

작업	참고
읽기 작업	완전 지원
중분 쿼리	완전 지원
시간 이동 쿼리	완전 지원
DML INSERT	IAM 권한만 있음
DML UPDATE	IAM 권한만 있음
DML DELETE	IAM 권한만 있음

작업	참고
DDL 명령	IAM 권한만 있음
메타데이터 테이블	지원되지만 특정 테이블은 숨겨집니다. 자세한 내용은 <a href="#">고려 사항 및 제한 사항</a> 을 참조하세요.
저장 프로시저	register_table 및 migrate를 제외하고 지원됩니다. 자세한 내용은 <a href="#">고려 사항 및 제한 사항</a> 을 참조하세요.
테이블 유지 관리 및 유틸리티 기능	해당 사항 없음

Iceberg의 Spark 구성: 다음 샘플은 Iceberg를 사용하여 Spark를 구성하는 방법을 보여줍니다. Iceberg 작업을 실행하려면 다음 spark-submit 속성을 제공합니다.

```
--conf spark.sql.catalog.spark_catalog=org.apache.iceberg.spark.SparkSessionCatalog
--conf spark.sql.catalog.spark_catalog.warehouse=<S3_DATA_LOCATION>
--conf spark.sql.catalog.spark_catalog.glue.account-id=<ACCOUNT_ID>
--conf spark.sql.catalog.spark_catalog.client.region=<REGION>
--conf spark.sql.catalog.spark_catalog.glue.endpoint=https://
glue.<REGION>.amazonaws.com
```

## Hudi

작업	참고
읽기 작업	완전 지원
중분 쿼리	완전 지원
시간 이동 쿼리	완전 지원
DML INSERT	IAM 권한만 있음
DML UPDATE	IAM 권한만 있음
DML DELETE	IAM 권한만 있음

작업	참고
DDL 명령	IAM 권한만 있음
메타데이터 테이블	지원되지 않음
저장 프로시저	해당 사항 없음
테이블 유지 관리 및 유틸리티 기능	지원되지 않음

다음 샘플은 파일 위치 및 사용에 필요한 기타 속성을 지정하여 Hudi로 Spark를 구성하는 예제입니다.

Hudi를 위한 Spark 구성: 노트북에서 사용할 때 이 코드 조각은 Spark에서 Hudi 기능을 활성화하는 Hudi Spark 번들 JAR 파일의 경로를 지정합니다. 또한 Glue Data Catalog를 AWS 메타스토어로 사용하도록 Spark를 구성합니다.

```
%%configure -f
{
  "conf": {
    "spark.jars": "/usr/lib/hudi/hudi-spark-bundle.jar",
    "spark.hadoop.hive.metastore.client.factory.class":
    "com.amazonaws.glue.catalog.metastore.AWSGlueDataCatalogHiveClientFactory",
    "spark.serializer": "org.apache.spark.serializer.JavaSerializer",
    "spark.sql.catalog.spark_catalog":
    "org.apache.spark.sql.hudi.catalog.HoodieCatalog",
    "spark.sql.extensions":
    "org.apache.spark.sql.hudi.HoodieSparkSessionExtension"
  }
}
```

AWS Glue를 사용하는 Hudi를 위한 Spark 구성: 노트북에서 이 코드 조각을 사용하면 Hudi가 지원되는 데이터 레이크 형식으로 활성화되고 Hudi 라이브러리 및 종속성을 사용할 수 있습니다.

```
%%configure
{
  "--conf": "spark.serializer=org.apache.spark.serializer.JavaSerializer --conf
spark.sql.catalog.spark_catalog=org.apache.spark.sql.hudi.catalog.HoodieCatalog --
conf
spark.sql.extensions=org.apache.spark.sql.hudi.HoodieSparkSessionExtension",
  "--datalake-formats": "hudi",
}
```

```
--enable-glue-datacatalog": True,
"--enable-lakeformation-fine-grained-access": "true"
}
```

## Delta Lake

작업	참고
읽기 작업	완전 지원
중분 쿼리	완전 지원
시간 이동 쿼리	완전 지원
DML INSERT	IAM 권한만 있음
DML UPDATE	IAM 권한만 있음
DML DELETE	IAM 권한만 있음
DDL 명령	IAM 권한만 있음
메타데이터 테이블	지원되지 않음
저장 프로시저	해당 사항 없음
테이블 유지 관리 및 유틸리티 기능	지원되지 않음

Delta Lake를 사용하는 EMR Serverless: EMR Serverless에서 Delta Lake를 Lake Formation과 함께 사용하려면 다음 명령을 실행합니다.

```
spark-sql \
  --conf
  spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension,com.amazonaws.emr.recordserver
  \
  --conf
  spark.sql.catalog.spark_catalog=org.apache.spark.sql.delta.catalog.DeltaCatalog \
```

## 디버깅 작업

### Note

이 기능을 사용하면 필터링되지 않은 민감한 정보가 포함될 수 있는 시스템 프로파일 워커의 stdout 및 stderr 로그에 액세스할 수 있습니다. 다음 권한은 비프로덕션 데이터에 액세스하는 데만 사용합니다. 프로덕션 작업에 사용하기 위해 생성된 애플리케이션의 경우 데이터 액세스 권한이 높은 관리자 또는 사용자에게만 이러한 권한을 추가하는 것이 좋습니다.

EMR-7.3.0 이상에서는 EMR Serverless를 통해 Lake Formation 지원 배치 작업에 대한 자체 디버깅 기능을 사용할 수 있습니다. 이를 위해서는 [GetDashboardForJobRun](#) API에서 새 파라미터 `accessSystemProfileLogs`를 사용합니다. `accessSystemProfileLogs`가 true로 설정된 경우 Lake Formation 지원 EMR Serverless 배치 작업을 디버깅하는 데 사용할 수 있는 시스템 프로파일 워커의 stdout 및 stderr 로그에 액세스할 수 있습니다.

```
aws emr-serverless get-dashboard-for-job-run \
  --application-id application-id
  --job-run-id job-run-id
  --access-system-profile-logs
```

### 필수 권한

`GetDashboardForJobRun`을 사용하여 Lake Formation 지원 배치 작업을 디버깅하려는 보안 주체에는 반드시 다음 추가 권한이 있어야 합니다.

```
{
  "Sid": "AccessSystemProfileLogs",
  "Effect": "Allow",
  "Action": [
    "emr-serverless:GetDashboardForJobRun",
    "emr-serverless:AccessSystemProfileLogs",
    "glue:GetDatabases",
    "glue:SearchTables"
  ],
  "Resource": [
    "arn:aws:emr-serverless:region:account-id:/applications/applicationId/jobruns/jobId",
    "arn:aws:glue:region:account-id:catalog",
    "arn:aws:glue:region:account-id:database/*",
  ]
}
```

```

    "arn:aws:glue:region:account-id:table/*/*"
  ]
}
```

## 고려 사항

디버깅용 시스템 프로파일 로그는 작업과 동일한 계정 내 Lake Formation의 데이터베이스 또는 테이블에 액세스하는 작업에 대해 확인할 수 있습니다. 다음 시나리오의 경우 표시되지 않습니다.

- Lake Formation 권한을 사용하여 관리되는 Data Catalog에 교차 계정 데이터베이스 및 테이블이 있는 경우
- Lake Formation 권한을 사용하여 관리되는 Data Catalog에 리소스 링크가 있는 경우

## Glue Data Catalog 뷰 작업

EMR Serverless와 함께 사용할 수 있도록 AWS Glue Data Catalog에서 뷰를 생성하고 관리할 수 있습니다. 일반적으로 AWS Glue Data Catalog 뷰라고 합니다. 이러한 보기는 여러 SQL 쿼리 엔진을 지원하므로 EMR Serverless, Amazon Athena, Amazon Redshift 등 여러 AWS 서비스에서 동일한 뷰에 액세스할 수 있다는 점에서 유용합니다.

Data Catalog에서 뷰를 생성하면 AWS Lake Formation에서 리소스 권한 부여 및 태그 기반 액세스 제어를 사용하여 이에 대한 액세스 권한을 부여할 수 있습니다. 이 액세스 제어 방법을 사용하면 보기를 생성할 때 참조한 테이블에 대한 추가 액세스를 구성하지 않아도 됩니다. 이러한 권한 부여 방법을 정의자 시맨틱이라고 하며 이러한 보기를 정의자 보기라고 합니다. Lake Formation에서 액세스 제어에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 [Granting and revoking permissions on Data Catalog resources](#)를 참조하세요.

Data Catalog 보기는 다음 사용 사례에 유용합니다.

- 세분화된 액세스 제어 - 사용자에게 필요한 권한에 따라 데이터 액세스를 제한하는 뷰를 생성할 수 있습니다. 예를 들어, 데이터 카탈로그의 뷰를 사용하여 인사 관리(HR) 부서 소속이 아닌 직원은 개인 식별 정보(PII)를 보지 못하게 할 수 있습니다.
- 완전한 정의 완료 - Data Catalog의 뷰에 필터를 적용하면 데이터 카탈로그의 뷰에서 사용할 수 있는 데이터 레코드가 항상 완전한지 확인할 수 있습니다.
- 향상된 보안 - 보기를 생성하는 데 사용되는 쿼리 정의를 완료합니다. 이러한 이점은 데이터 카탈로그의 보기가 악의적 액터의 SQL 명령에 영향을 받지 않는다는 것을 의미합니다.
- 간단한 데이터 공유 - 데이터를 이동하지 않고도 다른 AWS 계정과 데이터를 공유합니다. 자세한 내용은 [Cross-account data sharing in Lake Formation](#)을 참조하세요.

## Data Catalog 뷰 생성

Data Catalog 뷰는 다양한 방법으로 생성할 수 있습니다. 여기에는 AWS CLI 또는 Spark SQL을 사용하는 방법이 포함됩니다. 몇 가지 예는 다음과 같습니다.

### Using SQL

다음은 Data Catalog 뷰를 생성하기 위한 구문을 보여줍니다. MULTI DIALECT 뷰 유형을 기록합니다. 이 작업을 수행하면 데이터 카탈로그 보기가 다른 보기와 구별됩니다. SECURITY 조건자는 DEFINER로 지정됩니다. 이는 DEFINER 의미 체계가 있는 Data Catalog 뷰를 나타냅니다.

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW [IF NOT EXISTS] view_name
[(column_name [COMMENT column_comment], ...) ]
[ COMMENT view_comment ]
[TBLPROPERTIES (property_name = property_value, ... )]
SECURITY DEFINER
AS query;
```

다음은 구문을 따르는 샘플 CREATE 문입니다.

```
CREATE PROTECTED MULTI DIALECT VIEW catalog_view
SECURITY DEFINER
AS
SELECT order_date, sum(totalprice) AS price
FROM source_table
GROUP BY order_date
```

또한 SQL을 사용하여 테스트 실행 모드에서 뷰를 만들어 실제로 리소스를 생성하지 않고도 뷰 생성을 테스트할 수 있습니다. 이 옵션을 사용하면 입력의 유효성을 검사하는 '테스트 실행'이 수행되고, 검사가 성공하면 뷰를 나타내는 AWS Glue 테이블 객체의 JSON이 반환됩니다. 이 경우 실제 뷰는 생성되지 않습니다.

```
CREATE [ OR REPLACE ] PROTECTED MULTI DIALECT VIEW view_name
SECURITY DEFINER
[ SHOW VIEW JSON ]
AS view-sql
```

## Using the AWS CLI

 Note

CLI 명령을 사용하는 경우 뷰 생성에 사용된 SQL은 구문 분석되지 않습니다. 이로 인해 뷰는 생성되었지만, 쿼리가 성공하지 못하는 경우가 발생할 수 있습니다. 뷰를 생성하기 전에 SQL 구문을 테스트합니다.

다음 CLI 명령을 사용하여 뷰를 만듭니다.

```
aws glue create-table --cli-input-json '{
  "DatabaseName": "database",
  "TableInput": {
    "Name": "view",
    "StorageDescriptor": {
      "Columns": [
        {
          "Name": "col1",
          "Type": "data-type"
        },
        ...
        {
          "Name": "col_n",
          "Type": "data-type"
        }
      ],
      "SerdeInfo": {}
    },
    "ViewDefinition": {
      "SubObjects": [
        "arn:aws:glue:aws-region:aws-account-id:table/database/referenced-table1",
        ...
        "arn:aws:glue:aws-region:aws-account-id:table/database/referenced-tableN",
      ],
      "IsProtected": true,
      "Representations": [
        {
          "Dialect": "SPARK",
          "DialectVersion": "1.0",
          "ViewOriginalText": "Spark-SQL",
          "ViewExpandedText": "Spark-SQL"
        }
      ]
    }
  }
}
```

```

    }
  ]
}
}'

```

## 지원되는 뷰 작업

다음 명령 조각은 Data Catalog 뷰를 사용하는 다양한 방법을 보여줍니다.

### • CREATE VIEW

Data Catalog 뷰를 생성합니다. 다음은 기존 테이블에서 뷰를 생성하는 방법을 보여주는 샘플입니다.

```

CREATE PROTECTED MULTI DIALECT VIEW catalog_view
SECURITY DEFINER AS SELECT * FROM my_catalog.my_database.source_table

```

### • ALTER VIEW

사용 가능한 구문:

- ALTER VIEW view\_name [FORCE] ADD DIALECT AS query
- ALTER VIEW view\_name [FORCE] UPDATE DIALECT AS query
- ALTER VIEW view\_name DROP DIALECT

FORCE ADD DIALECT 옵션을 사용하여 새 엔진 언어에 따라 스키마 및 하위 객체를 강제 업데이트할 수 있습니다. FORCE를 사용하여 다른 엔진 언어를 업데이트하지 않아도 이로 인해 쿼리 오류가 발생할 수 있습니다. 이와 관련한 샘플은 다음과 같습니다.

```

ALTER VIEW catalog_view FORCE ADD DIALECT
AS
SELECT order_date, sum(totalprice) AS price
FROM source_table
GROUP BY orderdate;

```

다음은 언어를 업데이트하기 위해 뷰를 변경하는 방법을 보여줍니다.

```

ALTER VIEW catalog_view UPDATE DIALECT AS
SELECT count(*) FROM my_catalog.my_database.source_table;

```

## • DESCRIBE VIEW

뷰를 설명하는 데 사용할 수 있는 구문:

- `SHOW COLUMNS {FROM|IN} view_name [{FROM|IN} database_name]` - 사용자에게 뷰를 설명하는 데 필요한 AWS Glue 및 Lake Formation 권한이 있는 경우 권한으로 열이 나열될 수 있습니다. 다음은 열을 표시하기 위한 몇 가지 샘플 명령입니다.

```
SHOW COLUMNS FROM my_database.source_table;
SHOW COLUMNS IN my_database.source_table;
```

- `DESCRIBE view_name` - 사용자에게 뷰를 설명하는 데 필요한 AWS Glue 및 Lake Formation 권한이 있는 경우 권한으로 메타데이터와 함께 뷰의 열이 나열될 수 있습니다.

## • DROP VIEW

사용 가능한 구문:

- `DROP VIEW [ IF EXISTS ] view_name`

다음 샘플은 삭제 전에 뷰가 존재하는지 테스트하는 DROP 문을 보여줍니다.

```
DROP VIEW IF EXISTS catalog_view;
```

## • SHOW CREATE VIEW

- `SHOW CREATE VIEW view_name` - 지정된 뷰를 생성하는 SQL 문을 보여줍니다. 다음은 Data Catalog 뷰 생성을 보여주는 샘플입니다.

```
SHOW CREATE TABLE my_database.catalog_view;
CREATE PROTECTED MULTI DIALECT VIEW my_catalog.my_database.catalog_view (
  net_profit,
  customer_id,
  item_id,
  sold_date)
TBLPROPERTIES (
  'transient_lastDdlTime' = '1736267222')
SECURITY DEFINER AS SELECT * FROM
my_database.store_sales_partitioned_1f WHERE customer_id IN (SELECT customer_id
from source_table limit 10)
```

## • SHOW VIEWS

일반 뷰, 다중 언어 뷰(MDV) 및 Spark 언어가 없는 MDV와 같은 카탈로그의 모든 뷰를 나열합니다. 사용 가능한 구문은 다음과 같습니다.

- SHOW VIEWS [{ FROM | IN } database\_name] [LIKE regex\_pattern]:

다음은 뷰를 표시하는 샘플 명령입니다.

```
SHOW VIEWS IN marketing_analytics LIKE 'catalog_view*';
```

Data Catalog 뷰 생성 및 구성에 대한 자세한 내용은 AWS Lake Formation 개발자 안내서의 [AWS Glue Data Catalog 뷰 빌드](#)를 참조하세요.

## Data Catalog 뷰 쿼리

Data Catalog 뷰를 생성한 후 AWS Lake Formation 세분화된 액세스 제어가 활성화된 Amazon EMR Serverless Spark 작업을 사용하여 쿼리할 수 있습니다. 작업 런타임 역할에는 Data Catalog 뷰에 대한 Lake Formation SELECT 권한이 있어야 합니다. 뷰에서 참조된 기본 테이블에 대한 액세스 권한을 부여하지 않아도 됩니다.

모든 설정을 마치면 보기를 쿼리할 수 있습니다. 예를 들어, EMR Studio에서 EMR Serverless 애플리케이션을 생성한 후 다음 쿼리를 실행하여 뷰에 액세스합니다.

```
SELECT * from my_database.catalog_view LIMIT 10;
```

유용한 함수는 `invoker_principal`입니다. 이 함수는 EMRS 작업 런타임 역할의 고유 식별자를 반환합니다. 이 함수는 호출 보안 주체에 따라 뷰 출력을 제어하는 용도로 사용할 수 있습니다. 이 함수를 사용하여 호출 역할에 따라 쿼리 결과를 구체화하는 조건을 뷰에 추가할 수 있습니다. 이 함수를 사용하려면 작업 런타임 역할에 `LakeFormation:GetDataLakePrincipal` IAM 작업 권한이 있어야 합니다.

```
select invoker_principal();
```

예를 들어, 이 함수를 WHERE 절에 추가하여 쿼리 결과를 구체화할 수 있습니다.

## 고려 사항 및 제한 사항

Data Catalog 뷰를 생성할 때 적용되는 사항은 다음과 같습니다.

- Amazon EMR 7.6 이상을 사용하는 경우 Data Catalog 뷰만 생성할 수 있습니다.

- Data Catalog 뷰를 정의하는 사용자에는 뷰에서 액세스하는 기본 테이블에 대해 SELECT 액세스 권한이 있어야 합니다. 특정 기본 테이블의 정의자 역할에 Lake Formation 필터가 적용된 경우 Data Catalog 뷰 생성에 실패합니다.
- 기본 테이블에는 Lake Formation의 IAMAllowedPrincipals 데이터 레이크 권한이 없어야 합니다. 이 권한이 있는 경우 다중 언어 뷰는 IAMAllowedPrincipals 권한이 없는 테이블만 참조라는 오류가 발생할 수 있습니다.
- 테이블의 Amazon S3 위치를 Lake Formation 데이터 레이크 위치로 등록해야 합니다. 테이블이 등록되지 않은 경우 다중 언어 뷰는 Lake Formation 관리형 테이블만 참조 오류가 발생할 수 있습니다. Lake Formation에 Amazon S3 위치를 등록하는 자세한 방법은 AWS Lake Formation 개발자 가이드에서 [Amazon S3 위치 등록](#)을 참조하세요.
- PROTECTED Data Catalog 보기만 생성할 수 있습니다. UNPROTECTED 보기는 지원되지 않습니다.
- Data Catalog 뷰 정의의 다른 AWS 계정에 있는 테이블을 참조할 수 없습니다. 또한 별도의 리전에 있는 동일한 계정의 테이블을 참조할 수 없습니다.
- 계정 또는 리전 간에 데이터를 공유하려면 리소스 링크를 사용하여 전체 뷰를 계정 및 리전 간에 공유합니다.
- 사용자 정의 함수(UDF)는 지원되지 않습니다.
- Iceberg 테이블 기반 뷰를 사용할 수 있습니다. 오픈 테이블 형식의 Apache Hudi 및 Delta Lake도 지원됩니다.
- Data Catalog 보기는 다른 보기를 참조할 수 없습니다.
- AWS Glue Data Catalog 뷰 스키마는 항상 소문자로 저장됩니다. 예를 들어, DDL 문을 사용하여 Castle이라는 열을 만드는 경우 생성된 열은 castle과 같이 소문자로 표시됩니다. 그런 다음 DML 쿼리의 열 이름을 Castle 또는 CASTLE로 지정하면 EMR Spark는 쿼리를 실행할 이름을 소문자로 만듭니다. 하지만 열 머리글은 쿼리에서 지정한 대소문자를 사용하여 표시됩니다.

DML 쿼리에 지정된 열 이름이 Glue Data Catalog의 열 이름과 일치하지 않는 경우 쿼리가 실패하도록 하려면 `spark.sql.caseSensitive=true`를 설정합니다.

## 고려 사항 및 제한 사항

EMR Serverless와 함께 Lake Formation을 사용하는 경우 다음 고려 사항 및 제한 사항을 고려합니다.

### Note

EMR Serverless에서 Spark 작업에 대해 Lake Formation을 활성화하면 작업이 시스템 드라이버 및 사용자 드라이버를 시작합니다. 시작 시 사전 초기화된 용량을 지정한 경우 드라이버는

사전 초기화된 용량부터 프로비저닝하고 시스템 드라이버 수는 사용자가 지정한 사용자 드라이버 수와 같습니다. 온디맨드 용량을 선택하면 EMR Serverless는 사용자 드라이버 외에도 시스템 드라이버를 시작합니다. Lake Formation 작업을 사용하는 EMR Serverless와 관련된 비용을 예측하려면 [AWS Pricing Calculator](#)를 사용합니다.

Lake Formation을 사용하는 Amazon EMR Serverless는 지원되는 모든 [EMR Serverless 리전](#)에서 사용할 수 있습니다.

- Amazon EMR Serverless는 Apache Hive, Apache Iceberg, Delta Lake 및 Hudi 테이블에 대한 읽기 작업에 대해 Lake Formation을 통한 세분화된 액세스 제어를 지원합니다. Apache Hive 형식으로는, Parquet, ORC 및 xSV가 포함됩니다.
- Lake Formation 지원 애플리케이션은 [사용자 지정 EMR Serverless 이미지](#) 사용을 지원하지 않습니다.
- Lake Formation 작업의 경우 DynamicResourceAllocation을 끌 수 없습니다.
- Spark 작업에서 Lake Formation만 사용할 수 있습니다.
- Lake Formation을 사용하는 EMR Serverless는 작업 전체에서 단일 Spark 세션만 지원합니다.
- Lake Formation을 사용하는 EMR Serverless는 리소스 링크를 통해 공유되는 교차 계정 테이블 쿼리만 지원합니다.
- 다음은 지원되지 않습니다.
  - 복원력 있는 분산 데이터세트(RDD)
  - Spark 스트리밍
  - Lake Formation에 부여된 권한으로 쓰기
  - 중첩된 열에 대한 액세스 제어
- EMR Serverless는 다음을 포함하여 시스템 드라이버의 완전한 격리를 저해할 수 있는 기능을 차단합니다.
  - UDT, HiveUDF 및 사용자 지정 클래스가 포함된 사용자 정의 함수
  - 사용자 지정 데이터 소스
  - Spark 확장, 커넥터 또는 메타스토어에 대한 추가 jar 제공
  - ANALYZE TABLE 명령
- 액세스 제어, EXPLAIN PLAN 및 DDL 작업(예: DESCRIBE TABLE)을 적용하려면 제한된 정보를 노출하지 않습니다.

- EMR Serverless는 Lake Formation 지원 애플리케이션의 시스템 드라이버 Spark 로그에 대한 액세스를 제한합니다. 시스템 드라이버는 관리자 권한으로 실행되므로 시스템 드라이버가 생성하는 이벤트 및 로그에는 민감한 정보가 포함될 수 있습니다. 권한이 없는 사용자 또는 코드가 이 민감한 데이터에 액세스하지 못하도록 EMR Serverless는 시스템 드라이버 로그에 대한 액세스를 비활성화했습니다.

시스템 프로파일 로그는 항상 관리형 스토리지에 유지되며, 이는 비활성화할 수 없는 필수 설정입니다. 이러한 로그는 고객 관리형 KMS 키 또는 AWS 관리형 KMS 키를 사용하여 안전하게 저장되고 암호화됩니다.

EMR Serverless 애플리케이션이 Amazon S3용 VPC 엔드포인트가 있는 프라이빗 서브넷에 있고 엔드포인트 정책을 연결하여 액세스를 제어하는 경우, 작업이 AWS 관리형 Amazon S3로 로그 데이터를 전송하기 전에 [logging.html#jobs-log-storage-managed-storage](https://docs.aws.amazon.com/emr/latest/ServerlessGuide/logging.html#jobs-log-storage-managed-storage) VPC 정책에 자세히 설명된 권한을 Amazon S3 게이트웨이 엔드포인트에 포함합니다. 문제 해결 요청은 AWS Support에 문의하세요.

- Lake Formation에 테이블 위치를 등록한 경우 EMR Serverless 작업 런타임 역할에 대한 IAM 권한과 관계없이 데이터 액세스 경로는 Lake Formation에 저장된 자격 증명을 통과합니다. 테이블 위치에 등록된 역할을 잘못 구성하면 테이블 위치에 대한 S3 IAM 권한이 있는 역할을 사용하는 제출된 작업이 실패합니다.
- Lake Formation 테이블에 쓰는 경우 Lake Formation에 부여된 권한이 아닌 IAM 권한을 사용합니다. 작업 런타임 역할에 필요한 S3 권한이 있는 경우 이를 사용하여 쓰기 작업을 실행할 수 있습니다.
- Amazon EMR 7.9.0부터 Spark FGAC는 s3a:// 스키마와 함께 사용할 때 S3AFileSystem를 지원합니다.

다음은 Apache Iceberg를 사용하는 경우 고려 사항 및 제한 사항입니다.

- Apache Iceberg는 세션 카탈로그에서만 사용할 수 있으며, 임의로 이름이 지정된 카탈로그에서는 사용할 수 없습니다.
- Lake Formation에 등록된 Iceberg 테이블은 메타데이터 테이블 history, metadata\_log\_entries, snapshots, files, manifests, refs만 지원합니다. Amazon EMR은 partitions, path, summaries와 같이 민감한 데이터를 포함할 수 있는 열을 숨깁니다. 이 제한 사항은 Lake Formation에 등록되지 않은 Iceberg 테이블에 적용되지 않습니다.
- Lake Formation에 등록하지 않은 테이블은 모든 Iceberg 저장 프로시저를 지원합니다. register\_table 및 migrate 절차는 어떤 테이블에서도 지원되지 않습니다.
- V1 대신 Iceberg DataFrameWriterV2를 사용하는 것이 좋습니다.

## 문제 해결

문제 해결 방법은 다음 섹션을 참조하세요.

### 로깅

EMR Serverless는 Spark 리소스 프로파일을 사용하여 작업 실행을 분할합니다. EMR Serverless는 사용자 프로파일을 사용하여 사용자가 제공한 코드를 실행하는 반면, 시스템 프로파일은 Lake Formation 정책을 적용합니다. 사용자 프로파일로 실행된 태스크의 로그에 액세스할 수 있습니다.

Lake Formation 지원 작업 디버깅에 대한 자세한 내용은 [디버깅 작업](#)을 참조하세요.

### Live UI 및 Spark 기록 서버

Live UI 및 Spark 기록 서버에는 사용자 프로파일에서 생성된 모든 Spark 이벤트 및 시스템 드라이버에서 생성된 수정된 이벤트가 있습니다.

실행기 탭에서 사용자 및 시스템 드라이버의 모든 태스크를 볼 수 있습니다. 그러나 로그 링크는 사용자 프로파일에서만 사용할 수 있습니다. 또한 출력 레코드 수와 같은 일부 정보는 Live UI에서 수정됩니다.

### Lake Formation 권한이 부족하여 작업 실패

작업 런타임 역할에 액세스 중인 테이블에서 SELECT 및 DESCRIBE를 실행할 권한이 있는지 확인합니다.

### RDD 실행이 실패한 작업

EMR Serverless는 현재 Lake Formation 지원 작업에서 탄력적 분산 데이터세트(RDD) 작업을 지원하지 않습니다.

### Amazon S3의 데이터 파일에 액세스할 수 없음

Lake Formation에서 데이터 레이크의 위치를 등록했는지 확인합니다.

### 보안 검증 예외

EMR Serverless에서 보안 검증 오류를 탐지했습니다. AWS Support에 문의하여 지원을 받으세요.

### 여러 계정에서 AWS Glue Data Catalog 및 테이블 공유

여러 계정에서 데이터베이스와 테이블을 공유하고 Lake Formation을 계속 사용할 수 있습니다. 자세한 내용은 [Cross-account data sharing in Lake Formation](#) 및 [AWS Lake Formation을 사용하여 AWS](#)

[Glue Data Catalog 데이터베이스 및 테이블 교차 계정을 공유하려면 어떻게 해야 하나요?](#)를 참조하세요.

## 작업자 간 암호화

Amazon EMR 버전 6.15.0 이상을 사용하면 Spark 작업 실행의 워커 간 상호 TLS 암호화 통신을 활성화할 수 있습니다. 활성화되면 EMR Serverless는 작업 실행에서 프로비저닝된 각 작업자에 대해 고유한 인증서를 자동으로 생성하고 배포합니다. 이러한 작업자가 제어 메시지를 교환하거나 셔플 데이터를 전송하기 위해 통신하는 경우 상호 TLS 연결을 설정하고 구성된 인증서를 사용하여 서로의 자격 증명을 확인합니다. 작업자가 다른 인증서를 확인할 수 없는 경우 TLS 핸드셰이크에 실패하고 EMR Serverless가 해당 인증서 간 연결을 중단합니다.

EMR Serverless에서 Lake Formation을 사용하는 경우 상호 TLS 암호화가 기본적으로 활성화됩니다.

## EMR Serverless에서 상호 TLS 암호화 활성화

Spark 애플리케이션에서 상호 TLS 암호화를 활성화하려면 [EMR Serverless 애플리케이션 생성](#) 시 `spark.ssl.internode.enabled`를 `true`로 설정합니다. AWS 콘솔을 사용하여 EMR Serverless 애플리케이션을 생성하는 경우 사용자 지정 설정 사용을 선택한 다음, 애플리케이션 구성을 확장하고 `runtimeConfiguration`를 입력합니다.

```
aws emr-serverless create-application \
--release-label emr-6.15.0 \
--runtime-configuration '{
  "classification": "spark-defaults",
  "properties": {"spark.ssl.internode.enabled": "true"}
}' \
--type "SPARK"
```

개별 Spark 작업 실행에 대해 상호 TLS 암호화를 활성화하려면 `spark-submit` 사용 시 `spark.ssl.internode.enabled`를 `true`로 설정합니다.

```
--conf spark.ssl.internode.enabled=true
```

## EMR Serverless를 사용하는 데이터 보호를 위한 Secrets Manager

AWS Secrets Manager는 데이터베이스 보안 인증, API 키 및 기타 보안 암호 정보를 보호하는 보안 암호 저장소 서비스입니다. 그런 다음, 코드에서 하드코딩된 자격 증명을 Secrets Manager에 대한 API 직접 호출로 교체할 수 있습니다. 그러면 보안 암호가 해당 위치에 있지 않기 때문에 여러분의 코드

를 검사하는 누군가에 의해 보안 암호가 손상되지 않도록 방지할 수 있습니다. 개요는 [AWS Secrets Manager 사용 설명서](#)를 참조하세요.

Secrets Manager는 AWS Key Management Service키를 사용하여 보안 암호를 암호화합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 [보안 암호 암호화 및 복호화](#)를 참조하세요.

사용자가 지정한 일정에 따라 Secrets Manager가 자동으로 보안 암호를 교체하도록 구성할 수 있습니다. 따라서 단기 보안 암호로 장기 보안 암호를 교체할 수 있어 손상 위험이 크게 줄어듭니다. 자세한 내용은 AWS Secrets Manager 사용 설명서에서 [AWS Secrets Manager 보안 암호 교체](#)를 참조하세요.

Amazon EMR Serverless는 AWS Secrets Manager와 통합되므로 Secrets Manager에 데이터를 저장하고 구성에서 보안 암호 ID를 사용할 수 있습니다.

## EMR Serverless에서 보안 암호를 사용하는 방법

Secrets Manager에 데이터를 저장하고 EMR Serverless에 대한 구성에서 보안 암호 ID를 사용하는 경우 민감한 구성 데이터를 일반 텍스트로 EMR Serverless에 전달하지 않고 외부 API에 노출하지 않습니다. 키값 페어가 Secrets Manager에 저장한 보안 암호의 보안 암호 ID를 포함함을 나타내는 경우 EMR Serverless는 작업을 실행하기 위해 작업자로 구성 데이터를 전송할 때 보안 암호를 검색합니다.

구성의 키값 페어가 Secrets Manager에 저장된 보안 암호에 대한 참조를 포함함을 나타내려면 구성 값에 `EMR.secret@` 주석을 추가합니다. 보안 암호 ID 주석이 있는 구성 속성의 경우 EMR Serverless는 Secrets Manager를 직접 호출하고 작업 실행 시 보안 암호를 확인합니다.

## 보안 암호 생성 방법

보안 암호를 생성하려면 AWS Secrets Manager 사용 설명서에서 [AWS Secrets Manager 보안 암호 생성](#)의 단계를 따릅니다. 3단계에서 일반 텍스트 필드를 선택하여 민감한 값을 입력합니다.

## 구성 분류에서 보안 암호 제공

다음 예제에서는 StartJobRun의 구성 분류에서 보안 암호를 제공하는 방법을 보여줍니다. 애플리케이션 수준에서 Secrets Manager에 대한 분류를 구성하려면 [EMR Serverless에 대한 기본 애플리케이션 구성](#) 섹션을 참조하세요.

예제에서 `SecretName`을 검색할 보안 암호의 이름으로 바꿉니다. 자세한 정보는 [보안 암호 생성 방법](#) 섹션을 참조하세요.

이 섹션의 내용

- [보안 암호 참조 지정 - Spark](#)
- [보안 암호 참조 지정 - Hive](#)

## 보안 암호 참조 지정 - Spark

Example - Spark에 대한 외부 Hive 메타스토어 구성에서 보안 암호 참조 지정

```
aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://amzn-s3-demo-bucket/scripts/spark-jdbc.py",
      "sparkSubmitParameters": "--jars s3://amzn-s3-demo-bucket/mariadb-
connector-java.jar
      --conf
spark.hadoop.javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver
      --conf spark.hadoop.javax.jdo.option.ConnectionUserName=connection-user-
name
      --conf
spark.hadoop.javax.jdo.option.ConnectionPassword=EMR.secret@SecretName
      --conf spark.hadoop.javax.jdo.option.ConnectionURL=jdbc:mysql://db-host:db-
port/db-name
      --conf spark.driver.cores=2
      --conf spark.executor.memory=10G
      --conf spark.driver.memory=6G
      --conf spark.executor.cores=4"
    }
  }' \
  --configuration-overrides '{
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://amzn-s3-demo-bucket/spark/logs/"
      }
    }
  }'
```

Example - **spark-defaults** 분류에서 외부 Hive 메타스토어 구성에 대한 보안 암호 참조 지정

```
{
  "classification": "spark-defaults",
  "properties": {

    "spark.hadoop.javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver"
    "spark.hadoop.javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-
port/db-name"
```

```

        "spark.hadoop.javax.jdo.option.ConnectionUserName": "connection-user-name"
        "spark.hadoop.javax.jdo.option.ConnectionPassword":
"EMR.secret@SecretName",
    }
}

```

## 보안 암호 참조 지정 - Hive

Example - Hive에 대한 외부 Hive 메타스토어 구성에서 보안 암호 참조 지정

```

aws emr-serverless start-job-run \
  --application-id "application-id" \
  --execution-role-arn "job-role-arn" \
  --job-driver '{
    "hive": {
      "query": "s3://amzn-s3-demo-bucket/emr-serverless-hive/query/hive-query.q1",
      "parameters": "--hiveconf hive.exec.scratchdir=s3://amzn-s3-demo-bucket/emr-
serverless-hive/hive/scratch
                    --hiveconf hive.metastore.warehouse.dir=s3://amzn-s3-demo-bucket/
emr-serverless-hive/hive/warehouse
                    --hiveconf javax.jdo.option.ConnectionUserName=username
                    --hiveconf
javax.jdo.option.ConnectionPassword=EMR.secret@SecretName
                    --hiveconf
hive.metastore.client.factory.class=org.apache.hadoop.hive.q1.metadata.SessionHiveMetaStoreCli
                    --hiveconf
javax.jdo.option.ConnectionDriverName=org.mariadb.jdbc.Driver
                    --hiveconf javax.jdo.option.ConnectionURL=jdbc:mysql://db-host:db-
port/db-name"
    }
  }' \
  --configuration-overrides '{
    "monitoringConfiguration": {
      "s3MonitoringConfiguration": {
        "logUri": "s3://amzn-s3-demo-bucket"
      }
    }
  }'

```

Example - **hive-site** 분류에서 외부 Hive 메타스토어 구성에 대한 보안 암호 참조 지정

```

{
  "classification": "hive-site",

```

```

    "properties": {
      "hive.metastore.client.factory.class":
"org.apache.hadoop.hive.ql.metadata.SessionHiveMetaStoreClientFactory",
      "javax.jdo.option.ConnectionDriverName": "org.mariadb.jdbc.Driver",
      "javax.jdo.option.ConnectionURL": "jdbc:mysql://db-host:db-port/db-name",
      "javax.jdo.option.ConnectionUserName": "username",
      "javax.jdo.option.ConnectionPassword": "EMR.secret@SecretName"
    }
  }
}

```

## EMR Serverless에서 보안 암호를 검색할 액세스 권한 부여

EMR Serverless가 Secrets Manager로부터 보안 암호 값을 검색할 수 있도록 하려면 생성 시 보안 암호에 다음 정책 명령문을 추가합니다. EMR Serverless가 보안 암호 값을 읽으려면 고객 관리형 KMS 키를 사용하여 보안 암호를 생성해야 합니다. 자세한 내용은 AWS Secrets Manager 사용 설명서의 [KMS 키에 대한 권한](#)을 참조하세요.

다음 정책에서 *applicationId*를 애플리케이션의 ID로 바꿉니다.

### 보안 암호에 대한 리소스 정책

EMR Serverless에서 보안 암호 값을 검색할 수 있도록 AWS Secrets Manager에서 보안 암호에 대한 리소스 정책에 다음 권한을 포함해야 합니다. 특정 애플리케이션만 이 보안 암호를 검색할 수 있도록 하려면 선택적으로 정책의 조건으로 EMR Serverless 애플리케이션 ID를 지정할 수 있습니다.

### JSON

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {

```

```

        "aws:SourceArn": "arn:aws:emr-serverless:*:123456789012:/applications/
    "*"
    }
    },
    "Sid": "AllowSECRETSMANAGERGetsecretvalue"
}
]
}

```

고객 관리형 AWS Key Management Service(AWS KMS) 키에 대해 다음 정책을 사용하여 보안 암호를 생성합니다.

고객 관리형 AWS KMS 키에 대한 정책

```

{
  "Sid": "Allow EMR Serverless to use the key for decrypting secrets",
  "Effect": "Allow",
  "Principal": {
    "Service": [
      "emr-serverless.amazonaws.com"
    ]
  },
  "Action": [
    "kms:Decrypt",
    "kms:DescribeKey"
  ],
  "Resource": "*",
  "Condition": {
    "StringEquals": {
      "kms:ViaService": "secretsmanager.AWS ##.amazonaws.com"
    }
  }
}

```

## 보안 암호 교체

교체는 보안 암호를 주기적으로 업데이트하는 것입니다. 사용자가 지정한 일정에 따라 자동으로 보안 암호를 교체하도록 AWS Secrets Manager를 구성할 수 있습니다. 그러면 장기 보안 암호를 단기 보안 암호로 교체할 수 있습니다. 이를 통해 침해 위험을 줄일 수 있습니다. EMR Serverless는 작업이 실행 중 상태로 전환될 때 주석이 달린 구성에서 보안 암호 값을 검색합니다. 사용자 또는 프로세스가

Secrets Manager에서 보안 암호 값을 업데이트하는 경우 작업이 업데이트된 값을 가져올 수 있도록 새 작업을 제출해야 합니다.

 Note

이미 실행 중 상태인 작업은 업데이트된 보안 암호 값을 가져올 수 없습니다. 이로 인해 작업이 실패할 수 있습니다.

## EMR Serverless에서 Amazon S3 Access Grants 사용

### EMR Serverless에 대한 S3 Access Grants 개요

Amazon EMR 릴리스 6.15.0 이상을 사용하면 Amazon S3 Access Grants에서 EMR Serverless의 Amazon S3 데이터에 대한 액세스를 강화하기 위해 사용 가능한 확장 가능한 액세스 제어 솔루션이 제공됩니다. S3 데이터에 대한 권한 구성이 복잡하거나 대규모인 경우 Access Grants를 사용하여 사용자, 역할 및 애플리케이션을 위한 S3 데이터 권한을 확장할 수 있습니다.

S3 Access Grants를 사용하여 Amazon S3 데이터에 대한 액세스를 EMR Serverless 애플리케이션에 액세스할 수 있는 자격 증명에 연결된 런타임 역할 또는 IAM 역할에 의해 부여된 권한 이상으로 강화합니다.

자세한 내용은 Amazon EMR 관리 가이드의 [Amazon EMR을 위한 S3 Access Grants를 사용한 액세스 관리](#) 및 Amazon Simple Storage Service 사용 설명서의 [S3 Access Grants를 사용한 액세스 관리](#)를 참조하세요.

이 섹션에서는 Amazon S3의 데이터에 대한 액세스를 제공하기 위해 S3 Access Grants를 사용하는 EMR Serverless 애플리케이션을 시작하는 방법을 설명합니다. 기타 Amazon EMR 배포를 이용하여 S3 Access Grants를 사용하는 단계를 확인하려면 다음 설명서를 참조하세요.

- [Amazon EMR을 통해 S3 Access Grants 사용](#)
- [Amazon EMR on EKS에서 S3 Access Grants 사용](#)

## 데이터 관리를 위해 S3 Access Grants를 사용하여 Amazon EMR Serverless 애플리케이션 시작

EMR Serverless에서 S3 Access Grants를 활성화하고 Spark 애플리케이션을 시작할 수 있습니다. 애플리케이션에서 S3 데이터에 대한 요청이 발생할 경우 Amazon S3에서는 특정 버킷, 접두사 또는 객체로 범위가 지정된 임시 보안 인증을 제공합니다.

1. EMR Serverless 애플리케이션에 대한 작업 실행 역할을 설정합니다. Spark 작업을 실행하고 S3 Access Grants(s3:GetDataAccess 및 s3:GetAccessGrantsInstanceForPrefix)를 실행하는 데 필요한 필수 IAM 권한을 포함합니다.

```
{
  "Effect": "Allow",
  "Action": [
    "s3:GetDataAccess",
    "s3:GetAccessGrantsInstanceForPrefix"
  ],
  "Resource": [
    //LIST ALL INSTANCE ARNS THAT THE ROLE IS ALLOWED TO QUERY
    "arn:aws_partition:s3:Region:account-id1:access-grants/default",
    "arn:aws_partition:s3:Region:account-id2:access-grants/default"
  ]
}
```

### Note

S3에 직접 액세스할 수 있는 추가 권한이 있는 작업 실행에 대한 IAM 역할을 지정하는 경우 사용자는 S3 Access Grants의 권한이 없더라도 역할에서 허용하는 데이터에 액세스할 수 있습니다.

2. 다음 예제와 같이 Amazon EMR 릴리스 레이블이 6.15.0 이상이고 spark-defaults 분류를 포함하는 EMR Serverless 애플리케이션을 시작합니다. *red text*의 값을 사용 시나리오에 적합한 값으로 바꾸세요.

```
aws emr-serverless start-job-run \
  --application-id application-id \
  --execution-role-arn job-role-arn \
  --job-driver '{
    "sparkSubmit": {
      "entryPoint": "s3://us-east-1.elasticmapreduce/emr-containers/samples/wordcount/scripts/wordcount.py",
```

```

    "entryPointArguments": ["s3://amzn-s3-demo-destination-bucket1/
wordcount_output"],
    "sparkSubmitParameters": "--conf spark.executor.cores=1 --conf
spark.executor.memory=4g --conf spark.driver.cores=1 --conf spark.driver.memory=4g
--conf spark.executor.instances=1"
  }
}' \
--configuration-overrides '{
  "applicationConfiguration": [{
    "classification": "spark-defaults",
    "properties": {
      "spark.hadoop.fs.s3.s3AccessGrants.enabled": "true",
      "spark.hadoop.fs.s3.s3AccessGrants.fallbackToIAM": "false"
    }
  }]
}'

```

## EMR Serverless에서의 S3 Access Grants 고려 사항

Amazon EMR Serverless에서 Amazon S3 Access Grants를 사용하는 경우의 중요 지원, 호환성 및 동작 정보를 확인하려면 Amazon EMR 관리 안내서의 [Amazon EMR에서의 Amazon S3 Access Grants 고려 사항](#)을 참조하세요.

## AWS CloudTrail을 사용하여 Amazon EMR Serverless API 직접 호출 로깅

Amazon EMR Serverless는 EMR Serverless에서 사용자, 역할 또는 AWS 서비스가 수행한 작업 레코드를 제공하는 서비스인 AWS CloudTrail과 통합됩니다. CloudTrail은 EMR Serverless에 대한 모든 API 직접 호출을 이벤트로 캡처합니다. 캡처되는 호출에는 EMR Serverless 콘솔로부터의 직접 호출과 EMR Serverless API 작업에 대한 코드 직접 호출이 포함됩니다. 추적을 생성하면 EMR Serverless에 대한 이벤트를 포함한 CloudTrail 이벤트를 지속해서 Amazon S3 버킷에 배포할 수 있습니다. 트레일을 구성하지 않은 경우에도 CloudTrail 콘솔의 이벤트 기록에서 최신 이벤트를 볼 수 있습니다. CloudTrail에서 수집한 정보를 사용하여 EMR Serverless에 수행된 요청, 요청이 수행된 IP 주소, 요청을 수행한 사람, 요청이 수행된 시간 및 추가 세부 정보를 확인할 수 있습니다.

에 대한 자세한 내용은 [AWS CloudTrail 사용 설명서](#)를 참조하세요.

## CloudTrail의 EMR Serverless 정보

CloudTrail은 계정 생성 시 AWS 계정에서 사용되도록 설정됩니다. EMR Serverless에서 활동이 발생하면 해당 활동은 이벤트 기록의 다른 AWS 서비스 이벤트와 함께 CloudTrail 이벤트에 기록됩니다. AWS 계정에서 최신 이벤트를 확인, 검색 및 다운로드할 수 있습니다. 자세한 내용은 [CloudTrail 이벤트 기록을 사용하여 이벤트 보기](#)를 참조하세요.

EMR Serverless에 대한 이벤트를 포함하여 AWS 계정에서 진행 중인 이벤트 기록을 위해 추적을 생성합니다. CloudTrail은 추적을 사용하여 Amazon S3 버킷으로 로그 파일을 전송할 수 있습니다. 콘솔에서 추적을 생성하면 기본적으로 모든 AWS 리전에 추적이 적용됩니다. 추적은 AWS 파티션에 있는 모든 리전의 이벤트를 로깅하고 지정된 Amazon S3 버킷으로 로그 파일을 전송합니다. 또한 CloudTrail 로그에서 수집된 이벤트 데이터를 추가 분석 및 작업하도록 다른 AWS 서비스를 구성할 수 있습니다. 자세한 내용은 다음을 참조하세요.

- [추적 생성 개요](#)
- [CloudTrail 지원 서비스 및 통합](#)
- [CloudTrail에 대한 Amazon SNS 알림 구성](#)
- [여러 리전에서 CloudTrail 로그 파일 받기](#) 및 [여러 계정에서 CloudTrail 로그 파일 받기](#)

모든 EMR Serverless 작업은 CloudTrail에 의해 로깅되며 [EMR Serverless API 참조](#)에 문서화됩니다. 예를 들어, CreateApplication, StartJobRun, CancelJobRun 작업을 직접 호출하면 CloudTrail 로그 파일에 항목이 생성됩니다.

모든 이벤트 또는 로그 항목에는 요청을 생성했던 사용자에 대한 정보가 포함됩니다. ID 정보를 이용하면 다음을 쉽게 판단할 수 있습니다.

- 요청을 루트로 했는지 아니면 AWS Identity and Access Management(IAM) 사용자 보안 인증으로 했는지 여부입니다.
- 역할 또는 페더레이션 사용자에게 대한 임시 보안 인증을 사용하여 요청이 생성되었는지 여부.
- 다른 AWS 서비스에서 요청했는지.

자세한 내용은 [CloudTrail userIdentity 요소](#)를 참조하세요.

## EMR Serverless 로그 파일 항목 이해

추적이란 지정한 S3 버킷에 이벤트를 로그 파일로 입력할 수 있게 하는 구성입니다. CloudTrail 로그 파일에는 하나 이상의 로그 항목이 포함될 수 있습니다. 이벤트는 모든 소스의 단일 요청을 나타내며 요

청된 작업, 작업 날짜와 시간, 요청 파라미터 등에 대한 정보를 포함합니다. CloudTrail 로그 파일은 퍼블릭 API 직접 호출의 주문 스택 트레이스가 아니므로 특정 순서로 표시되지 않습니다.

다음은 CreateApplication 작업을 보여주는 CloudTrail 로그 항목이 나타낸 예시입니다.

```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AIDACKCEVSQ6C2EXAMPLE:admin",
    "arn": "arn:aws:sts::012345678910:assumed-role/Admin/admin",
    "accountId": "012345678910",
    "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AIDACKCEVSQ6C2EXAMPLE",
        "arn": "arn:aws:iam::012345678910:role/Admin",
        "accountId": "012345678910",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "creationDate": "2022-06-01T23:46:52Z",
        "mfaAuthenticated": "false"
      }
    }
  },
  "eventTime": "2022-06-01T23:49:28Z",
  "eventSource": "emr-serverless.amazonaws.com",
  "eventName": "CreateApplication",
  "awsRegion": "us-west-2",
  "sourceIPAddress": "203.0.113.0",
  "userAgent": "PostmanRuntime/7.26.10",
  "requestParameters": {
    "name": "my-serverless-application",
    "releaseLabel": "emr-6.6",
    "type": "SPARK",
    "clientToken": "0a1b234c-de56-7890-1234-567890123456"
  },
  "responseElements": {
    "name": "my-serverless-application",
    "applicationId": "1234567890abcdef0",
  },
}
```

```

    "arn": "arn:aws:emr-serverless:us-west-2:555555555555:/
applications/1234567890abcdef0"
  },
  "requestID": "890b8639-e51f-11e7-b038-EXAMPLE",
  "eventID": "874f89fa-70fc-4798-bc00-EXAMPLE",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "012345678910",
  "eventCategory": "Management"
}

```

## Amazon EMR Serverless에 대한 규정 준수 확인

서드파티 감사자는 다음을 포함하는 여러 AWS 규정 준수 프로그램의 일환으로 EMR Serverless의 보안 및 규정 준수를 평가합니다.

- SOC(시스템 및 조직 제어)
- PCI DSS(지불 카드 산업 데이터 보안 표준)
- 연방정부의 위험 및 인증 관리 프로그램(FedRAMP)(Moderate)
- HIPAA(미국 건강 보험 양도 및 책임에 관한 법)

AWS는 [규정 준수 프로그램 제공 범위 내 AWS 서비스](#) 페이지에서 특정 규정 준수 프로그램 범위 내 자주 업데이트되는 AWS 서비스의 목록을 제공합니다.

AWS Artifact를 사용하여 다운로드할 수 있는 타사 감사 보고서가 있습니다. 자세한 내용은 [AWS Artifact의 보고서 다운로드](#)를 참조하세요.

AWS 규정 준수 프로그램에 대한 자세한 내용은 [AWS 규정 준수 프로그램](#)을 참조하세요.

EMR Serverless 사용 시 귀하의 규정 준수 책임은 데이터의 민감도, 조직의 규정 준수 목표, 관련 법률과 규정에 따라 결정됩니다. EMR Serverless 사용 시 HIPAA, PCI 또는 FedRAMP(Moderate)와 같은 표준을 준수해야 하는 경우 AWS에서는 도움이 되는 다음 리소스를 제공합니다.

- [보안 및 규정 준수 빠른 시작 안내서](#)에서는 아키텍처 고려 사항에 대해 설명하고 보안 및 규정 준수에 중점을 둔 기본 환경을 AWS에 배포하기 위한 단계를 제공합니다.
- [AWS 고객 규정 준수 안내서](#)를 통해 규정 준수의 관점에서 공동 책임 모델을 이해할 수 있습니다. 이 가이드에서는 AWS 서비스를 보호하기 위한 모범 사례를 요약하고 여러 프레임워크(미국 표준 기술

연구소(NIST), 결제 카드 산업 보안 표준 위원회(PCI), 국제 표준화기구(ISO) 등에서 보안 컨트롤에 대한 지침을 매핑합니다.

- [AWS Config](#)를 사용하여 리소스 구성이 내부 관행, 업계 지침 및 규정을 준수하는 정도를 평가할 수 있습니다.
- [AWS 규정 준수 리소스](#)는 사용자의 업계와 위치에 적용할 수 있는 워크북 및 안내서 모음입니다.
- [AWS Security Hub](#)는 보안 업계 표준 및 모범 사례 준수 여부를 확인하는 데 도움이 되는 AWS 내 보안 상태에 대한 포괄적인 관점을 제공합니다.
- [AWS Audit Manager](#) - 이 AWS 서비스는 AWS 사용을 지속해서 감사하여 위험을 관리하고 규정 및 업계 표준을 준수하는 방법을 간소화할 수 있도록 지원합니다.

## Amazon EMR Serverless의 복원력

AWS 글로벌 인프라는 AWS 리전 및 가용 영역을 중심으로 구축됩니다. AWS 리전은 물리적으로 분리되고 격리된 다수의 가용 리전을 제공하며 이러한 가용 리전은 짧은 지연 시간, 높은 처리량 및 높은 중복성을 갖춘 네트워크에 연결되어 있습니다. 가용 영역을 사용하면 중단 없이 영역 간에 자동으로 장애 극복 조치가 이루어지는 애플리케이션 및 데이터베이스를 설계하고 운영할 수 있습니다. 가용 영역은 기존의 단일 또는 복수 데이터 센터 인프라보다 가용성, 내결함성, 확장성이 뛰어납니다.

AWS 리전 및 가용 영역에 대한 자세한 내용은 [AWS 글로벌 인프라](#)를 참조하세요.

AWS 글로벌 인프라 외에도 Amazon EMR Serverless는 데이터 복원성과 백업 요구 사항을 지원하고 자 EMRFS를 통한 Amazon S3와의 통합을 제공합니다.

## Amazon EMR Serverless의 인프라 보안

관리형 서비스인 Amazon EMR은 AWS 글로벌 네트워크 보안으로 보호됩니다. AWS 보안 서비스와 AWS의 인프라 보호 방법에 대한 자세한 내용은 [AWS Cloud Security](#)을 참조하세요. 인프라 보안에 대한 모범 사례를 사용하여 AWS 환경을 설계하려면 보안 원칙 AWS Well-Architected Framework의 [인프라 보호](#)를 참조하세요.

AWS에서 게시한 API 직접 호출을 사용하여 네트워크를 통해 Amazon EMR에 액세스합니다. 고객은 다음을 지원해야 합니다.

- Transport Layer Security(TLS) TLS 1.2는 필수이며 TLS 1.3을 권장합니다.
- DHE(Ephemeral Diffie-Hellman) 또는 ECDHE(Elliptic Curve Ephemeral Diffie-Hellman)와 같은 완전 전송 보안(PFS)이 포함된 암호 제품군 Java 7 이상의 최신 시스템은 대부분 이러한 모드를 지원합니다.

# Amazon EMR Serverless에서 구성 및 취약성 분석

AWS가 게스트 운영 체제(OS), 데이터베이스 패치, 방화벽 구성, 재해 복구 등의 기본 보안 작업을 처리합니다. 적합한 제3자가 이 절차를 검토하고 인증하였습니다. 자세한 정보는 다음 리소스를 참조하세요.

- [Amazon EMR Serverless에 대한 규정 준수 확인](#)
- [공동 책임 모델](#)
- [Web Services: 보안 프로세스의 개요](#)

# EMR Serverless에 대한 엔드포인트 및 할당량

## 서비스 엔드포인트

AWS 서비스에 프로그래밍 방식으로 연결하려면 엔드포인트를 사용합니다. 엔드포인트는 AWS 웹 서비스를 위한 진입점의 URL입니다. 표준 AWS 엔드포인트 외에도 일부 AWS 서비스 서비스는 일부 리전에서 FIPS 엔드포인트를 제공합니다. 다음 표에서는 EMR Serverless에 대한 서비스 엔드포인트를 나열합니다. 자세한 내용은 [AWS 서비스 엔드포인트](#)를 참조하세요.

### EMR Serverless 서비스 엔드포인트

지역명	지역	엔드포인트	프로토콜
미국 동부(오하이오)	us-east-2 (use2-az1, use2-az2, use2-az3 가용 영역으로 제한됨)	emr-serverless.us-east-2.amazonaws.com	HTTPS
미국 동부(버지니아 북부)	us-east-1 (use1-az1, use1-az2, use1-az4, use1-az5, use1-az6 가용 영역으로 제한됨)	emr-serverless.us-east-1.amazonaws.com  emr-serverless-fips.us-east-1.amazonaws.com	HTTPS
미국 서부(캘리포니아 북부)	us-west-1	emr-serverless.us-west-1.amazonaws.com	HTTPS
미국 서부(오리건)	us-west-2	emr-serverless.us-west-2.amazonaws.com	HTTPS

지역명	지역	엔드포인트	프로토콜
		emr-serverless-fips.us-west-2.amazonaws.com	
아프리카(케이프타운)	af-south-1	emr-serverless.af-south-1.amazonaws.com	HTTPS
아시아 태평양(홍콩)	ap-east-1	emr-serverless.ap-east-1.amazonaws.com	HTTPS
아시아 태평양(자카르타)	ap-southeast-3	emr-serverless.ap-southeast-3.amazonaws.com	HTTPS
아시아 태평양(멜버른)	ap-southeast-4	emr-serverless.ap-southeast-4.amazonaws.com	HTTPS
아시아 태평양(뭄바이)	ap-south-1	emr-serverless.ap-south-1.amazonaws.com	HTTPS

지역명	지역	엔드포인트	프로토콜
아시아 태평양(오사카)	ap-northeast-3	emr-serverless.ap-northeast-3.amazonaws.com	HTTPS
아시아 태평양(서울)	ap-northeast-2	emr-serverless.ap-northeast-2.amazonaws.com	HTTPS
아시아 태평양(싱가포르)	ap-southeast-1	emr-serverless.ap-southeast-1.amazonaws.com	HTTPS
아시아 태평양(시드니)	ap-southeast-2	emr-serverless.ap-southeast-2.amazonaws.com	HTTPS
아시아 태평양(도쿄)	ap-northeast-1	emr-serverless.ap-northeast-1.amazonaws.com	HTTPS
캐나다(중부)	ca-central-1 (cac1-az1 및 cac1-az2 가용 영역으로 제한됨)	emr-serverless.ca-central-1.amazonaws.com	HTTPS

지역명	지역	엔드포인트	프로토콜
캐나다 서부(캘거리)	ca-west-1	emr-serverless.ca-west-1.amazonaws.com	HTTPS
유럽(프랑크푸르트)	eu-central-1	emr-serverless.eu-central-1.amazonaws.com	HTTPS
유럽(취리히)	eu-central-2	emr-serverless.eu-central-2.amazonaws.com	HTTPS
유럽(아일랜드)	eu-west-1	emr-serverless.eu-west-1.amazonaws.com	HTTPS
유럽(런던)	eu-west-2	emr-serverless.eu-west-2.amazonaws.com	HTTPS
유럽(밀라노)	eu-south-1	emr-serverless.eu-south-1.amazonaws.com	HTTPS
유럽(파리)	eu-west-3	emr-serverless.eu-west-3.amazonaws.com	HTTPS

지역명	지역	엔드포인트	프로토콜
유럽(스페인)	eu-south-2	emr-serverless.eu-south-2.amazonaws.com	HTTPS
유럽(스톡홀름)	eu-north-1	emr-serverless.eu-north-1.amazonaws.com	HTTPS
이스라엘(텔아비브)	il-central-1	emr-serverless.il-central-1.amazonaws.com	HTTPS
중동(바레인)	me-south-1	emr-serverless.me-south-1.amazonaws.com	HTTPS
중동(UAE)	me-central-1	emr-serverless.me-central-1.amazonaws.com	HTTPS
남아메리카(상파울루)	sa-east-1	emr-serverless.sa-east-1.amazonaws.com	HTTPS
중국(베이징)	cn-north-1 (cnn1-az1 및 cnn1-az2 가용 영역으로 제한됨)	emr-serverless.cn-north-1.amazonaws.com.cn	HTTPS

지역명	지역	엔드포인트	프로토콜
AWS GovCloud(미국 동부)	us-gov-east-1	emr-serverless.us-gov-east-1.amazonaws.com	HTTPS
AWS GovCloud(미국 서부)	us-gov-west-1	emr-serverless.us-gov-west-1.amazonaws.com	HTTPS

## Service Quotas

Service Quotas(한도라고도 함)는 AWS 계정에서 사용할 수 있는 서비스 리소스 또는 작업의 최대 수를 말합니다. EMR Serverless는 1분마다 서비스 할당량 사용 지표를 수집하여 AWS/Usage 네임스페이스에 게시합니다.

### Note

새 AWS 계정에서는 초기 할당량이 낮지만 시간이 지남에 따라 증가할 수 있습니다. Amazon EMR Serverless는 각 AWS 리전 내에서 계정 사용량을 모니터링하고 사용량에 따라 할당량을 자동으로 늘립니다.

다음 표에는 EMR Serverless의 서비스 할당량이 나와 있습니다. 자세한 내용은 [AWS 서비스 할당량](#)을 참조하세요.

명칭	기본 한도	조정 가능?	설명
계정당 최대 동시 vCPU 수	16	예	현재 AWS 리전의 계정에 대해 동시에 실행할 수 있는 최대 vCPU 수.

명칭	기본 한도	조정 가능?	설명
계정당 최대 대기 중인 작업 수	2000	예	현재 AWS 리전의 계정에 대해 최대 대기 중인 작업 수.

## API 제한

다음은 AWS 계정에서 리전당 API 제한을 설명합니다.

리소스	기본 할당량
<a href="#">ListApplications</a>	초당 트랜잭션 10개. 초당 50개의 트랜잭션 버스트.
<a href="#">CreateApplication</a>	초당 트랜잭션 1개. 초당 25개의 트랜잭션 버스트.
<a href="#">DeleteApplication</a>	초당 트랜잭션 1개. 초당 25개의 트랜잭션 버스트.
<a href="#">GetApplication</a>	초당 트랜잭션 10개. 초당 50개의 트랜잭션 버스트.
<a href="#">UpdateApplication</a>	초당 트랜잭션 1개. 초당 25개의 트랜잭션 버스트.
<a href="#">ListJobRuns</a>	초당 트랜잭션 1개. 초당 25개의 트랜잭션 버스트.
<a href="#">StartJobRun</a>	초당 트랜잭션 1개. 초당 25개의 트랜잭션 버스트.
<a href="#">GetDashboardForJobRun</a>	초당 트랜잭션 1개. 초당 2개의 트랜잭션 버스트.
<a href="#">CancelJobRun</a>	초당 트랜잭션 1개. 초당 25개의 트랜잭션 버스트.

리소스	기본 할당량
<a href="#">GetJobRun</a>	초당 트랜잭션 10개. 초당 50개의 트랜잭션 버스트.
<a href="#">StartApplication</a>	초당 트랜잭션 1개. 초당 25개의 트랜잭션 버스트.
<a href="#">StopApplication</a>	초당 트랜잭션 1개. 초당 25개의 트랜잭션 버스트.

## 기타 고려 사항

다음 목록에는 EMR Serverless와 관련된 다른 고려 사항이 포함되어 있습니다.

- EMR Serverless는 다음 AWS 리전에서 사용 가능합니다.

- 미국 동부(오하이오)
- 미국 동부(버지니아 북부)
- 미국 서부(캘리포니아 북부)
- 미국 서부(오리건)
- 아프리카(케이프타운)
- 아시아 태평양(홍콩)
- 아시아 태평양(자카르타)
- 아시아 태평양(뭄바이)
- 아시아 태평양(오사카)
- 아시아 태평양(서울)
- 아시아 태평양(싱가포르)
- 아시아 태평양(시드니)
- 아시아 태평양(도쿄)
- 캐나다(중부)
- 유럽(프랑크푸르트)
- 유럽(아일랜드)
- 유럽(런던)
- 유럽(밀라노)
- 유럽(파리)
- 유럽(스페인)
- 유럽(스톡홀름)
- 중동(바레인)
- 중동(UAE)
- 남아메리카(상파울루)

---

- AWS GovCloud(미국 동부)

- AWS GovCloud(미국 서부)

이러한 리전과 연결된 엔드포인트 목록은 [서비스 엔드포인트](#) 섹션을 참조하세요.

- 작업 실행의 기본 제한 시간은 12시간입니다. `startJobRun` API 또는 AWS SDK의 `executionTimeoutMinutes` 속성을 사용하여 이 설정을 변경할 수 있습니다. 작업 실행 제한 시간을 초과하지 않으려면 `executionTimeoutMinutes`를 0으로 설정할 수 있습니다. 예를 들어, 스트리밍 애플리케이션이 있는 경우 스트리밍 작업을 계속 실행할 수 있도록 하려면 `executionTimeoutMinutes`를 0으로 설정합니다.
- `getJobRun` API의 `billedResourceUtilization` 속성은 작업 실행에 대해 AWS에서 청구하는 집계 vCPU, 메모리 및 스토리지를 보여줍니다. 비용이 청구되는 리소스로는, 작업자의 최소 사용 시간 1분과 작업자당 20GB를 초과하는 추가 스토리지가 포함됩니다. 이러한 리소스에는 유휴 상태의 사전 초기화된 작업자에 대한 사용량이 포함되지 않습니다.
- VPC 연결이 없으면 작업은 동일한 AWS 리전에 있는 일부 AWS 서비스 엔드포인트에 액세스할 수 있습니다. 이러한 리소스로는 S3, AWS Glue, AWS Lake Formation, Amazon CloudWatch Logs, AWS KMS, AWS Security Token Service, Amazon DynamoDB, AWS Secrets Manager이 포함됩니다. VPC 연결을 활성화하여 [AWS PrivateLink](#)를 통해 다른 AWS 서비스에 액세스할 수 있지만, 사용자가 이를 수행하지 않아도 됩니다. 외부 서비스에 액세스하려면 VPC를 사용하여 애플리케이션을 생성합니다.
- EMR Serverless는 HDFS를 지원하지 않습니다. 워커의 로컬 디스크는 EMR Serverless가 작업 실행 중에 데이터를 셔플링하고 처리하는 데 사용하는 임시 스토리지입니다.

# Amazon EMR Serverless 릴리스 버전

Amazon EMR 릴리스는 빅 데이터 에코시스템의 오픈 소스 애플리케이션입니다. 각 릴리스에는 작업을 실행할 때 Amazon EMR Serverless를 배포 및 구성하기 위해 선택한 여러 빅 데이터 애플리케이션, 구성 요소 및 기능이 포함되어 있습니다.

Amazon EMR 6.6.0 이상을 사용하여 EMR Serverless를 배포합니다. 이 배포 옵션은 이전 Amazon EMR 릴리스 버전에서 사용할 수 없습니다. 작업을 제출하는 경우 지원되는 다음 릴리스 중 하나를 지정합니다.

## 주제

- [EMR Serverless 7.10.0](#)
- [EMR Serverless 7.9.0](#)
- [EMR Serverless 7.8.0](#)
- [EMR Serverless 7.7.0](#)
- [EMR Serverless 7.6.0](#)
- [EMR Serverless 7.5.0](#)
- [EMR Serverless 7.4.0](#)
- [EMR Serverless 7.3.0](#)
- [EMR Serverless 7.2.0](#)
- [EMR Serverless 7.1.0](#)
- [EMR Serverless 7.0.0](#)
- [EMR Serverless 6.15.0](#)
- [EMR Serverless 6.14.0](#)
- [EMR Serverless 6.13.0](#)
- [EMR Serverless 6.12.0](#)
- [EMR Serverless 6.11.0](#)
- [EMR Serverless 6.10.0](#)
- [EMR Serverless 6.9.0](#)
- [EMR Serverless 6.8.0](#)
- [EMR Serverless 6.7.0](#)
- [EMR Serverless 6.6.0](#)

## EMR Serverless 7.10.0

다음 표에는 EMR Serverless 7.10.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.5
Apache Hive	3.1.3
Apache Tez	0.10.2

### EMR Serverless 7.10.0 릴리스 정보

- EMR Serverless에 대한 지표 - ApplicationName 및 JobName 차원에 초점을 맞추기 위해 모니터링 지표가 재구성됩니다. 이전 지표는 향후 더 이상 업데이트되지 않습니다. 자세한 내용은 [EMR Serverless 애플리케이션 및 작업 모니터링](#)을 참조하세요.

## EMR Serverless 7.9.0

다음 표에는 EMR Serverless 7.9.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.5
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 7.8.0

다음 표에는 EMR Serverless 7.8.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.4
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 7.7.0

다음 표에는 EMR Serverless 7.7.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.3
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 7.6.0

다음 표에는 EMR Serverless 7.6.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.3
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 7.5.0

다음 표에는 EMR Serverless 7.5.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.2
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 7.4.0

다음 표에는 EMR Serverless 7.4.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.2
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 7.3.0

다음 표에는 EMR Serverless 7.3.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.1
Apache Hive	3.1.3
Apache Tez	0.10.2

### EMR Serverless 7.3.0 릴리스 정보

- EMR Serverless를 사용한 작업 동시성 및 대기열 - Amazon EMR 릴리스 7.3.0 이상에서 새 EMR Serverless 애플리케이션을 생성할 때 작업 동시성 및 대기열은 기본적으로 활성화됩니다. 자세한

내용은 [the section called “작업 동시성 및 대기열 입력”](#)을 참조하세요. 여기에는 동시성과 대기열을 시작하는 방법이 자세히 설명되어 있으며, 기능 고려 사항 목록도 포함되어 있습니다.

## EMR Serverless 7.2.0

다음 표에는 EMR Serverless 7.2.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.1
Apache Hive	3.1.3
Apache Tez	0.10.2

### EMR Serverless 7.2.0 릴리스 정보

- EMR Serverless와 함께 Lake Formation - 이제 AWS Lake Formation을 사용하여 S3에서 지원하는 Data Catalog 테이블에 세분화된 액세스 제어를 적용할 수 있습니다. 이 기능을 사용하면 EMR Serverless Spark 작업 내에서 읽기 쿼리에 대한 테이블, 행, 열 및 셀 수준 액세스 제어를 구성할 수 있습니다. 자세한 정보는 [the section called “FGAC를 위한 Lake Formation”](#) 및 [the section called “고려 사항”](#) 섹션을 참조하세요.

## EMR Serverless 7.1.0

다음 표에는 EMR Serverless 7.1.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.0
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 7.0.0

다음 표에는 EMR Serverless 7.0.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.5.0
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 6.15.0

다음 표에는 EMR Serverless 6.15.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

### EMR Serverless 6.15.0 릴리스 정보

- TLS 지원 - Amazon EMR Serverless 릴리스 6.15.0 이상을 사용하면 Spark 작업 실행에서 워커 간 상호 TLS 암호화 통신을 활성화할 수 있습니다. 이 기능이 활성화되면 EMR Serverless는 TLS 핸드셰이크 중에 작업자가 서로를 인증하는 데 사용하고, 데이터를 안전하게 처리하기 위해 암호화된 채널을 설정하는 작업 실행에 따라 프로비저닝하는 각 작업자에 대해 고유한 인증서를 자동으로 생성합니다. 상호 TLS 암호화에 대한 자세한 내용은 [워커 간 암호화](#)를 참조하세요.

## EMR Serverless 6.14.0

다음 표에는 EMR Serverless 6.14.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 6.13.0

다음 표에는 EMR Serverless 6.13.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.4.1
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 6.12.0

다음 표에는 EMR Serverless 6.12.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.4.0
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 6.11.0

다음 표에는 EMR Serverless 6.11.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.3.2
Apache Hive	3.1.3
Apache Tez	0.10.2

### EMR Serverless 6.11.0 릴리스 정보

- [다른 계정의 S3 리소스 액세스](#) - 릴리스 6.11.0 이상을 사용하면 EMR Serverless의 다른 AWS 계정에서 Amazon S3 버킷에 액세스할 때 수입할 여러 IAM 역할을 구성할 수 있습니다.

## EMR Serverless 6.10.0

다음 표에는 EMR Serverless 6.10.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.3.1
Apache Hive	3.1.3
Apache Tez	0.10.2

### EMR Serverless 6.10.0 릴리스 정보

- 릴리스가 6.10.0 이상인 EMR Serverless 애플리케이션에서 `spark.dynamicAllocation.maxExecutors` 속성의 기본값은 `infinity`입니다. 이전 릴리스는 기본적으로 100입니다. 자세한 정보는 [Spark 작업 속성](#) 섹션을 참조하세요.

## EMR Serverless 6.9.0

다음 표에는 EMR Serverless 6.9.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.3.0
Apache Hive	3.1.3
Apache Tez	0.10.2

## EMR Serverless 6.9.0 릴리스 정보

- Apache Spark용 Amazon Redshift 통합은 Amazon EMR 릴리스 6.9.0 이상에 포함되어 있습니다. 이전의 오픈 소스 도구였던, 이 기본 통합은 Spark 커넥터로, Amazon Redshift와 Amazon Redshift Serverless에서 데이터를 읽고 쓰는 Apache Spark 애플리케이션을 빌드할 수 있습니다. 자세한 내용은 [Amazon EMR Serverless에서 Apache Spark용 Amazon Redshift 통합 사용](#) 섹션을 참조하세요.
- EMR Serverless 릴리스 6.9.0에서 AWS Graviton2(arm64) 아키텍처에 대한 지원을 추가합니다. create-application 및 update-application API의 architecture 파라미터를 사용하여 arm64 아키텍처를 선택할 수 있습니다. 자세한 정보는 [Amazon EMR Serverless 아키텍처 옵션](#) 섹션을 참조하세요.
- 이제 EMR Serverless Spark 및 Hive 애플리케이션에서 직접 Amazon DynamoDB 테이블을 내보내고, 가져오며, 쿼리하고, 조인할 수 있습니다. 자세한 정보는 [Amazon EMR Serverless를 사용하여 DynamoDB에 연결](#) 섹션을 참조하세요.

## 알려진 문제

- Apache Spark용 Amazon Redshift 통합을 사용하고 Parquet 형식의 time, timetz, timestamp 또는 timestamptz(마이크로초 정밀도)를 사용하는 경우 커넥터는 시간 값을 가장 가까운 밀리초 값으로 반올림합니다. 해결 방법으로 텍스트 언로드 형식 unload\_s3\_format 파라미터를 사용합니다.

## EMR Serverless 6.8.0

다음 표에는 EMR Serverless 6.8.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.3.0
Apache Hive	3.1.3
Apache Tez	0.9.2

## EMR Serverless 6.7.0

다음 표에는 EMR Serverless 6.7.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.2.1
Apache Hive	3.1.3
Apache Tez	0.9.2

## 엔진별 변경 사항, 개선 사항 및 해결된 문제

다음 테이블에는 새 엔진별 기능이 나와 있습니다.

변경 사항	설명
Feature	이제 Tez 스케줄러에서 컨테이너 선점 대신, Tez 태스크 선점 지원

## EMR Serverless 6.6.0

다음 표에는 EMR Serverless 6.6.0에서 사용할 수 있는 애플리케이션 버전이 나와 있습니다.

Application	버전
Apache Spark	3.2.0

Application	버전
Apache Hive	3.1.2
Apache Tez	0.9.2

## EMR Serverless 초기 릴리스 정보

- EMR Serverless는 Spark 구성 분류 spark-defaults를 지원합니다. 이 분류는 Spark의 spark-defaults.conf XML 파일에서 값을 변경합니다. 구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.
- EMR Serverless는 Hive 구성 분류 hive-site, emrfs-site, tez-site 및 core-site를 지원합니다. 이 분류는 Hive의 hive-site.xml 파일, Tez의 tez-site.xml 파일, Amazon EMR의 EMRFS 설정 또는 Hadoop의 core-site.xml 파일에서 각각 값을 변경할 수 있습니다. 구성 분류를 사용하면 애플리케이션을 사용자 지정할 수 있습니다. 자세한 내용은 [애플리케이션 구성](#)을 참조하세요.

## 엔진별 변경 사항, 개선 사항 및 해결된 문제

- 다음 표에는 Hive 및 Tez 백포트가 나와 있습니다.

### Hive 및 Tez 변경 사항

변경 사항	설명
백포트	<a href="#">TEZ-4430</a> : tez.task.launch.cmd-opts 속성 관련 문제 수정됨
백포트	<a href="#">HIVE-25971</a> : 열린 캐시 스레드 풀로 인한 Tez 태스크 종료 지연 수정됨

## 문서 기록

다음 표에서는 EMR Serverless의 최신 릴리스가 발표된 이후 이 설명서에서 변경된 중요 사항에 대해 설명합니다. 이 설명서의 업데이트에 대한 자세한 내용을 보려면 RSS 피드를 구독하면 됩니다.

변경 사항	설명	날짜
<a href="#">새로운 릴리스</a>	<a href="#">EMR Serverless 7.2.0</a>	2024년 7월 25일
<a href="#">새로운 릴리스</a>	<a href="#">EMR Serverless 7.1.0</a>	2024년 4월 17일
<a href="#">기존 정책에 대한 업데이트.</a>	새 Sid CloudWatchPolicyStatement , EC2PolicyStatement 를 <a href="#">AmazonEMRServerlessServiceRolePolicy 정책</a> 에 추가했습니다.	2024년 1월 25일
<a href="#">새로운 릴리스</a>	<a href="#">EMR Serverless 7.0.0</a>	2023년 12월 29일
<a href="#">새로운 릴리스</a>	<a href="#">EMR Serverless 6.15.0</a>	2023년 11월 17일
<a href="#">새로운 특성</a>	<a href="#">EMR Serverless(6.11 이상)와 다른 계정의 Amazon S3 버킷에 액세스할 때 수입할 여러 IAM 역할 구성</a>	2023년 10월 18일
<a href="#">새로운 릴리스</a>	<a href="#">EMR Serverless 6.14.0</a>	2023년 10월 17일
<a href="#">새로운 특성</a>	<a href="#">EMR Serverless에 대한 기본 애플리케이션 구성</a>	2023년 9월 25일
<a href="#">기본 Hive 속성에 대한 업데이트</a>	hive.driver.disk , hive.tez.disk.size , hive.tez.auto.reducer.parallelism , tez.grouping.min-s	2023년 9월 12일

ize [Hive 작업 속성](#)의 기본값을 업데이트했습니다.

<a href="#">새로운 릴리스</a>	<a href="#">EMR Serverless 6.13.0</a>	2023년 9월 11일
<a href="#">새로운 릴리스</a>	<a href="#">EMR Serverless 6.12.0</a>	2023년 7월 21일
<a href="#">새로운 릴리스</a>	<a href="#">EMR Serverless 6.11.0</a>	2023년 6월 8일
<a href="#">서비스 연결 역할 정책 업데이트</a>	"AWS/Usage" 네임스페이스에서 계정 수준 사용량을 게시하도록 <a href="#">AmazonEMR ServerlessServiceRolePolicy</a> SLR 역할을 업데이트했습니다.	2023년 4월 20일
<a href="#">EMR Serverless 일반 가용성 (GA)</a>	EMR Serverless의 첫 번째 퍼블릭 릴리스입니다.	2022년 6월 1일