



## AWS Glue Apache Spark 작업의 성능 튜닝 모범 사례



## : AWS Glue Apache Spark 작업의 성능 튜닝 모범 사례

# Table of Contents

소개 .....	1
주요 주제 .....	2
아키텍처 .....	2
복원력 있는 분산 데이터세트 .....	3
지연 평가 .....	4
Spark 애플리케이션 용어 .....	5
Parallelism .....	6
Catalyst 옵티마이저 .....	7
성능 문제 조사 .....	10
Spark UI를 사용하여 병목 현상 식별 .....	10
성능 조정을 위한 전략 .....	12
성능 조정을 위한 기준선 전략 .....	12
Spark 작업 성능에 대한 조정 사례 .....	13
클러스터 용량 규모 조정 .....	13
CloudWatch 지표 .....	14
Spark UI .....	14
최신 버전 사용 .....	16
데이터 스캔 양 감소 .....	16
CloudWatch 지표 .....	17
Spark UI .....	17
태스크 병렬화 .....	26
CloudWatch 지표 .....	26
Spark UI .....	27
셔플 최적화 .....	32
CloudWatch 지표 .....	33
Spark UI .....	33
계획 오버헤드 최소화 .....	41
CloudWatch 지표 .....	42
Spark UI .....	42
사용자 정의 함수 최적화 .....	43
표준 Python UDF .....	45
벡터화된 UDF .....	45
Spark SQL .....	46
빅 데이터에 panda 사용 .....	47

---

리소스 .....	48
문서 기록 .....	49
용어집 .....	50
# .....	50
A .....	51
B .....	54
C .....	55
D .....	59
E .....	62
F .....	64
G .....	66
H .....	67
I .....	69
L .....	71
M .....	72
O .....	76
P .....	78
Q .....	81
R .....	81
S .....	84
T .....	88
U .....	89
V .....	90
W .....	90
Z .....	91
.....	xciii

# AWS Glue Apache Spark 작업의 성능 튜닝 모범 사례

Roman Myers, Takashi Onikura, Noritaka Sekiyama, Amazon Web Services(AWS)

2023년 12월([문서 기록](#))

AWS Glue 는 성능 튜닝을 위한 다양한 옵션을 제공합니다. 이 가이드에서는 Apache Spark의 튜닝 AWS Glue 에 대한 주요 주제를 정의합니다. 그런 다음 Apache Spark 작업에 AWS Glue 대해 이를 튜닝할 때 따라야 할 기존 전략을 제공합니다. 이 가이드를 사용하여 AWS Glue에서 사용할 수 있는 지표를 해석함으로써 성능 문제를 식별하는 방법을 알아봅니다. 그런 다음 이러한 문제를 해결하기 위한 전략을 통합하여 성능을 극대화하고 비용을 최소화합니다.

이 가이드에서는 다음과 같은 조정 사례를 다룹니다.

- [클러스터 용량 규모 조정](#)
- [최신 AWS Glue 버전 사용](#)
- [데이터 스캔 양 감소](#)
- [태스크 병렬화](#)
- [계획 오버헤드 최소화](#)
- [셔플 최적화](#)
- [사용자 정의 함수 최적화](#)

## Apache Spark의 주요 주제

이 섹션에서는 Apache Spark 성능을 위해 AWS Glue 조정에 관한 Apache Spark 기본 개념과 주요 주제를 설명합니다. 실제 조정 전략을 논의하기 전에 이러한 개념과 주제를 이해하는 것이 중요합니다.

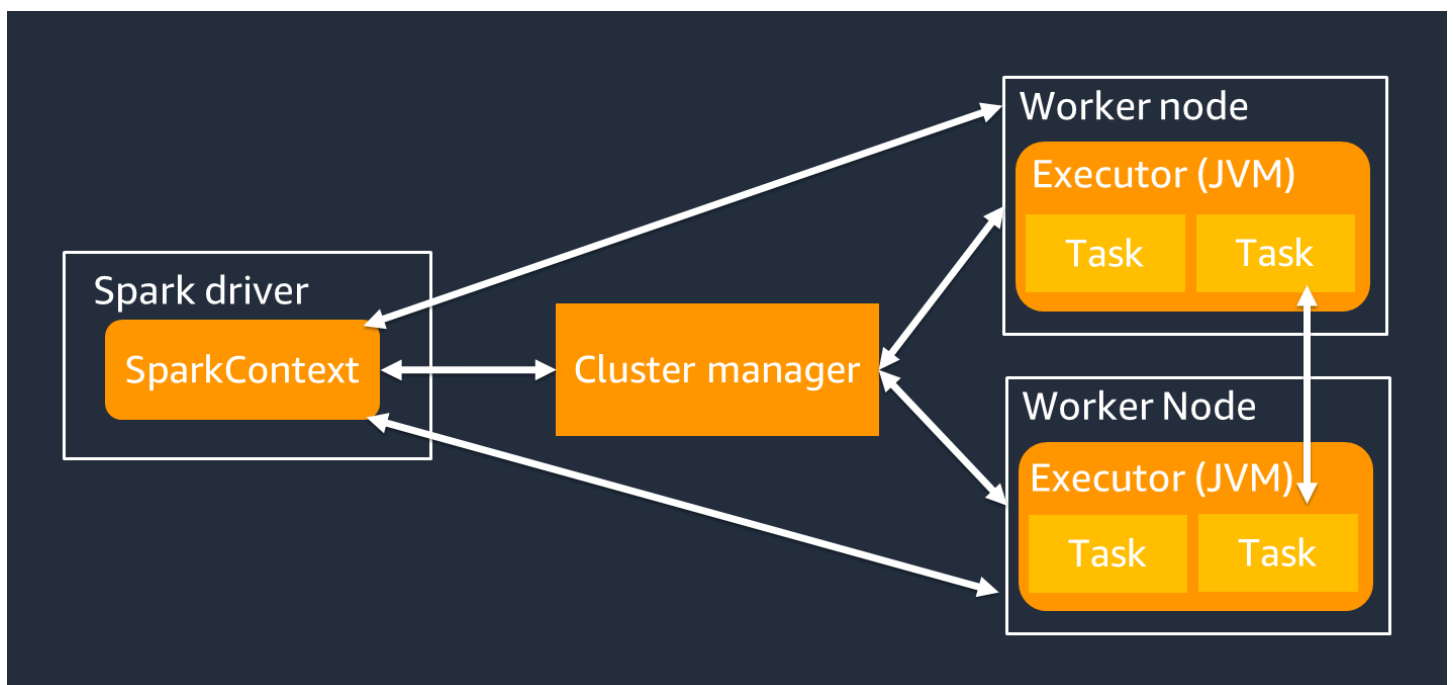
### 아키텍처

Spark 드라이버는 주로 Spark 애플리케이션을 개별 작업자에서 수행할 수 있는 태스크로 분할하는 작업을 담당합니다. Spark 드라이버에는 다음과 같은 책임이 있습니다.

- 코드에서 `main()` 실행
- 실행 계획 생성
- 클러스터의 리소스를 관리하는 클러스터 관리자와 함께 Spark 실행기 프로비저닝
- Spark 실행기에 대한 태스크 예약 및 태스크 요청
- 태스크 진행 상황 및 복구 관리

SparkContext 객체를 사용하여 작업 실행을 위해 Spark 드라이버와 상호 작용합니다.

Spark 실행기는 데이터를 보관하고 Spark 드라이버에서 전달되는 태스크를 실행하는 작업자입니다. Spark 실행기 수는 클러스터 크기에 따라 증가하거나 감소합니다.



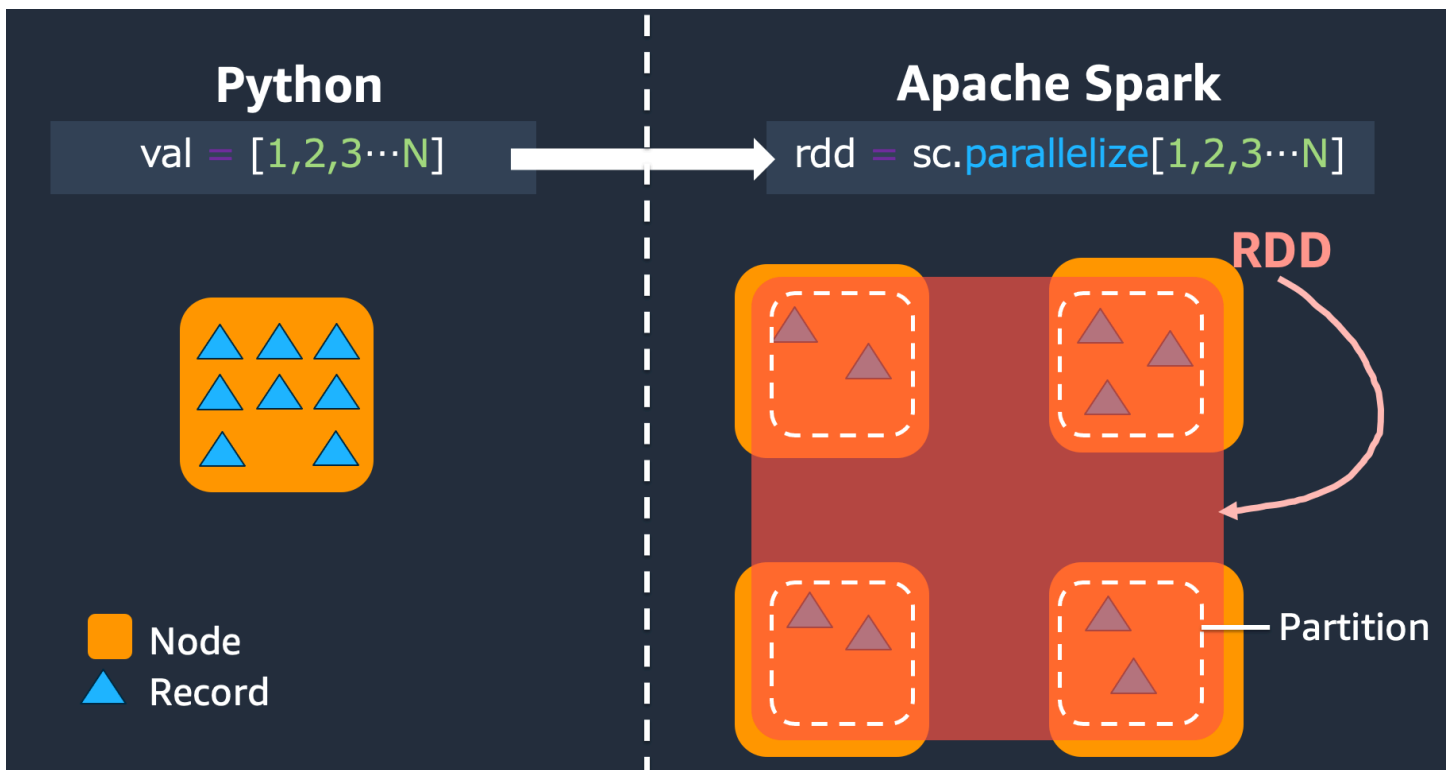
**Note**

Spark 실행기에는 여러 개의 슬롯이 있으므로 여러 태스크를 병렬로 처리할 수 있습니다. Spark는 기본적으로 각 가상 CPU(vCPU) 코어에 대해 하나의 태스크를 지원합니다. 예를 들어 실행기에 4개의 CPU 코어가 있는 경우 4개의 동시 태스크를 실행할 수 있습니다.

## 복원력 있는 분산 데이터세트

Spark는 Spark 실행기 전체에서 대규모 데이터세트를 저장하고 추적하는 복잡한 작업을 수행합니다. Spark 작업에 대한 코드를 작성할 경우 스토리지 세부 정보를 고려할 필요가 없습니다. Spark는 복원력이 뛰어난 분산 데이터세트(RDD) 추상화를 제공합니다. 이 추상화는 병렬로 작동하고 클러스터의 Spark 실행기에서 분할할 수 있는 요소의 모음입니다.

다음 그림에서는 Python 스크립트가 일반적인 환경에서 실행될 때와 Spark 프레임워크(PySpark)에서 실행될 때 메모리에 데이터를 저장하는 방법의 차이를 보여줍니다.



- Python - Python 스크립트에서 `val = [1,2,3...N]`을 작성하면 코드가 실행 중인 단일 시스템의 메모리에 데이터가 유지됩니다.

- PySpark – Spark는 여러 Spark 실행기의 메모리에 분산된 데이터를 로드하고 처리하기 위해 RDD 데이터 구조를 제공합니다. `rdd = sc.parallelize[1,2,3...N]`와 같은 코드를 사용하여 RDD를 생성할 수 있으며, Spark는 여러 Spark 실행기에 데이터를 자동으로 배포하고 메모리에 보관할 수 있습니다.

많은 AWS Glue 작업에서 AWS Glue DynamicFrames 및 Spark DataFrames를 통해 RDD를 사용합니다. RDD에서 데이터의 스키마를 정의하고 해당 추가 정보로 상위 수준 태스크를 수행할 수 있는 추상화입니다. RDD를 내부적으로 사용하기 때문에 데이터는 투명하게 배포되고 다음 코드의 여러 노드에 로드됩니다.

- DynamicFrame

```
dyf= glueContext.create_dynamic_frame.from_options(
    's3', {"paths": [ "s3://<YourBucket>/<Prefix>/" ]},
    format="parquet",
    transformation_ctx="dyf"
)
```

- DataFrame

```
df = spark.read.format("parquet")
    .load("s3://<YourBucket>/<Prefix>")
```

RDD에는 다음과 같은 기능이 있습니다.

- RDD는 파티션이라고 하는 여러 부분으로 분할된 데이터로 구성됩니다. 각 Spark 실행기는 하나 이상의 파티션을 메모리에 저장하고 데이터는 여러 실행기에 분산됩니다.
- RDD는 변경 불가능하므로 생성된 후에는 변경할 수 없습니다. DataFrame을 변경하려면 다음 섹션에 정의된 변환을 사용할 수 있습니다.
- RDD는 사용 가능한 여러 노드에서 데이터를 복제하므로 노드 장애로부터 자동으로 복구할 수 있습니다.

## 지연 평가

RDD는 기존 작업에서 새 데이터셋을 생성하는 변환과 데이터셋에서 계산을 실행한 후 드라이버 프로그램에 값을 반환하는 작업이라는 두 가지 유형의 작업을 지원합니다.

- 변환 - RDD는 변경 불가능하므로 변환을 사용해야만 변경할 수 있습니다.

예를 들어 map은 각 데이터세트 요소를 함수를 통해 전달하고 결과를 나타내는 새 RDD를 반환하는 변환입니다. map 메서드는 출력을 반환하지 않습니다. Spark는 사용자가 결과와 상호 작용할 수 있도록 하는 대신 향후 추상 변환을 저장합니다. Spark는 사용자가 작업을 직접 호출할 때까지 변환에 대해 작업하지 않습니다.

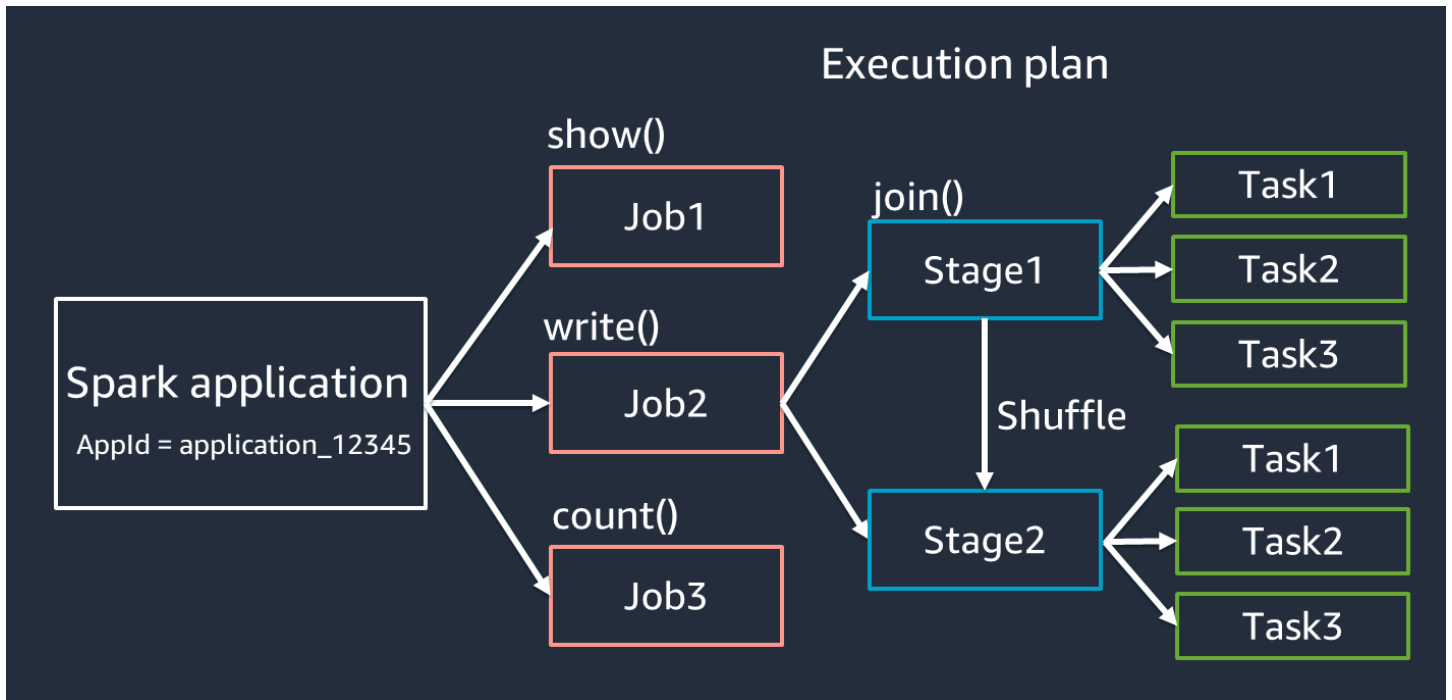
- 작업 - 변환을 사용하여 논리적 변환 계획을 수립합니다. 계산을 시작하려면 write, count show 또는 collect와 같은 작업을 실행합니다.

Spark의 모든 변환은 결과를 즉시 계산하지 않는다는 점에서 지연됩니다. 대신 Spark는 Amazon Simple Storage Service(Amazon S3) 객체와 같은 일부 기본 데이터세트에 적용된 일련의 변환을 기억합니다. 변환은 작업에서 결과를 드라이버에 반환해야 하는 경우에만 계산됩니다. 이 설계를 통해 Spark를 더 효율적으로 실행할 수 있습니다. 예를 들어 map 변환을 통해 생성된 데이터세트가 reduce와 같이 행 수를 크게 줄이는 변환에 의해서만 소비되는 상황을 고려합니다. 그런 다음 매핑된 더 큰 데이터세트를 전달하는 대신 두 변환을 모두 거친 더 작은 데이터세트를 드라이버에 전달할 수 있습니다.

## Spark 애플리케이션 용어

이 섹션에서는 Spark 애플리케이션 용어를 다룹니다. Spark 드라이버는 실행 계획을 생성하고 여러 추상화에서 애플리케이션의 동작을 제어합니다. 다음 용어는 Spark UI를 사용한 개발, 디버깅 및 성능 조정에 중요합니다.

- 애플리케이션 - Spark 세션(Spark 컨텍스트)에 기반합니다. 고유 ID(예: <application\_XXX>)로 식별됩니다.
- 작업 - RDD에 대해 생성된 작업에 기반합니다. 작업은 하나 이상의 단계로 구성됩니다.
- 스테이지 - RDD에 대해 생성된 셔플에 기반합니다. 단계는 하나 이상의 태스크로 구성됩니다. 셔플은 데이터를 재분산하기 위한 Spark의 메커니즘으로, 이를 통해 여러 RDD 파티션에서 다르게 데이터가 그룹화됩니다. join()과 같은 특정 변환에는 셔플이 필요합니다. 셔플은 [셔플 최적화](#) 조정 사례에서 자세히 설명합니다.
- 태스크 - 태스크는 Spark에서 예약한 최소 처리 단위입니다. 각 RDD 파티션에 대해 태스크가 생성되며, 태스크 수는 단계의 최대 동시 실행 수입니다.



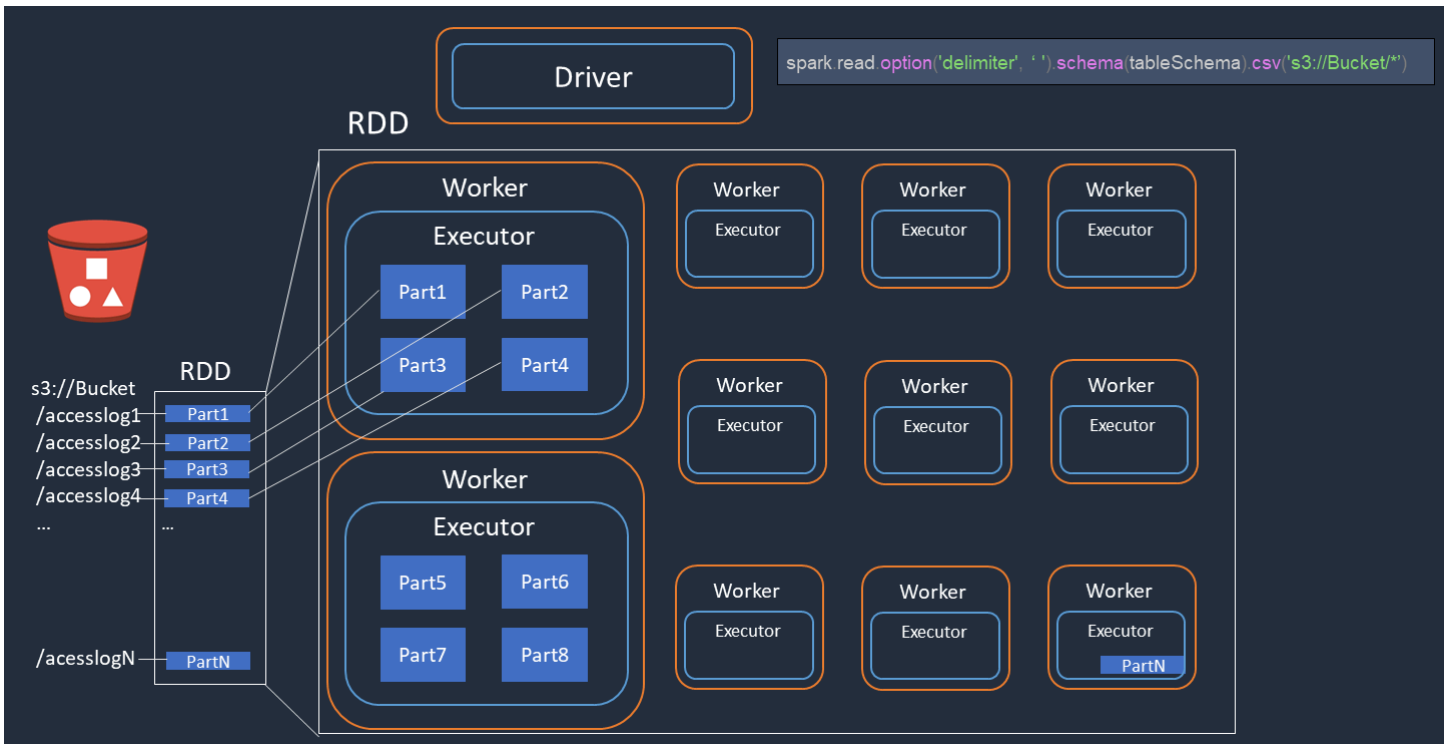
#### Note

병렬화를 최적화할 경우 고려해야 할 가장 중요한 것은 태스크입니다. RDD 수에 따라 조정되는 태스크 수

## Parallelism

Spark는 데이터 로드 및 변환 태스크를 병렬 처리합니다.

Amazon S3에서 액세스 로그 파일(accesslog1 ... accesslogN)의 분산 처리를 수행하는 예제를 고려합니다. 다음 다이어그램에서는 분산 처리 흐름을 보여줍니다.

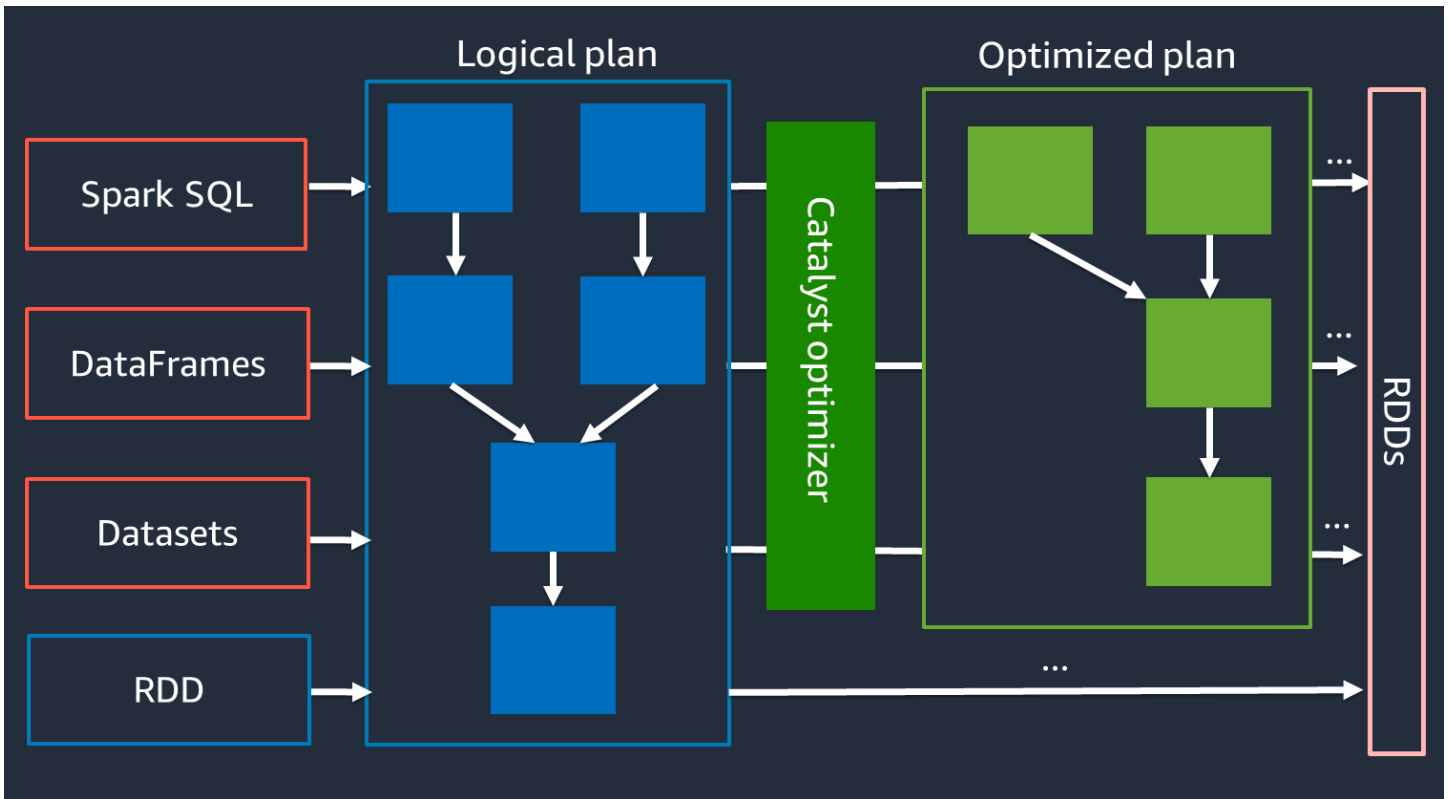


1. Spark 드라이버는 여러 Spark 실행기에서 분산 처리를 위한 실행 계획을 생성합니다.
2. Spark 드라이버는 실행 계획에 따라 각 실행기에 태스크를 할당합니다. 기본적으로 Spark 드라이버는 각 S3 객체(Part1 ... N)에 대해 RDD 파티션(각각 Spark 태스크에 해당)을 생성합니다. 그런 다음 Spark 드라이버는 각 실행기에 태스크를 할당합니다.
3. 각 Spark 태스크는 할당된 S3 객체를 다운로드하여 RDD 파티션의 메모리에 저장합니다. 이 방식으로 여러 Spark 실행기가 할당된 태스크를 병렬로 다운로드하고 처리합니다.

초기 파티션 수 및 최적화에 대한 자세한 내용은 [태스크 병렬화](#) 섹션을 참조하세요.

## Catalyst 옵티마이저

내부적으로 Spark는 [Catalyst 옵티마이저](#)라는 엔진을 사용하여 실행 계획을 최적화합니다. Catalyst에는 다음 다이어그램에 설명된 대로 [Spark SQL](#), [DataFrame](#) 및 [Datasets](#)와 같은 상위 수준 Spark API를 실행할 때 사용할 수 있는 쿼리 옵티마이저가 있습니다.



Catalyst 옵티마이저는 RDD API에서 직접 작동하지 않으므로 상위 수준 API는 일반적으로 하위 수준 RDD API보다 빠릅니다. 복잡한 조인의 경우 Catalyst 옵티마이저는 작업 실행 계획을 최적화하여 성능을 크게 개선할 수 있습니다. Spark UI의 SQL 탭에서 Spark 작업의 최적화된 계획을 볼 수 있습니다.

### 적응형 쿼리 실행

Catalyst 최적화 프로그램은 적응형 쿼리 실행이라는 프로세스를 통해 런타임 최적화를 수행합니다. 적응형 쿼리 실행은 런타임 통계를 사용하여 작업이 실행되는 동안 쿼리의 실행 계획을 다시 최적화합니다. 적응형 쿼리 실행은 다음 섹션에 설명된 대로 셔플 후 파티션 병합, 정렬 병합 조인을 브로드캐스트 조인으로 변환, 스큐 조인 최적화 등 성능 문제에 대한 여러 솔루션을 제공합니다.

적응형 쿼리 실행은 AWS Glue 3.0 이상에서 사용할 수 있으며, 4.0(Spark AWS Glue 3.3.0) 이상에서 기본적으로 활성화됩니다. 코드에서 `spark.conf.set("spark.sql.adaptive.enabled", "true")`를 사용하여 적응형 쿼리 실행을 켜거나 끌 수 있습니다.

### 사후 셔플 파티션 병합

이 기능은 map 출력 통계를 기반으로 각 셔플 후 RDD 파티션(병합)을 줄입니다. 쿼리를 실행할 때 셔플 파티션 번호의 조정을 단순화합니다. 데이터세트에 맞게 셔플 파티션 번호를 설정할 필요가 없습니다. Spark는 초기 셔플 파티션 수가 충분히 많으면 런타임에 적절한 셔플 파티션 번호를 선택할 수 있습니다.

`spark.sql.adaptive.enabled` 및

`spark.sql.adaptive.coalescePartitions.enabled`가 모두 `true`로 설정된 경우 셔플 후 파티션 병합이 활성화됩니다. 자세한 내용은 [Apache Spark 설명서](#)를 참조하세요.

정렬 병합 조인을 브로드캐스트 조인으로 변환

이 기능은 크기가 상당히 다른 두 데이터셋을 조인하는 시기를 인식하고 해당 정보를 기반으로 보다 효율적인 조인 알고리즘을 채택합니다. 자세한 내용은 [Apache Spark 설명서](#)를 참조하세요. 조인 전략은 [셔플 최적화](#) 섹션에서 설명합니다.

스큐 조인 최적화

데이터 스큐는 Spark 작업에서 가장 일반적인 병목 현상 중 하나입니다. 데이터가 특정 RDD 파티션(따라서 특정 태스크)으로 스큐되어 애플리케이션의 전체 처리 시간이 지연되는 상황을 설명합니다. 이 경우 종종 조인 작업의 성능이 저하될 수 있습니다. 스큐 조인 최적화 기능은 스큐된 태스크를 대략적으로 균등한 크기의 태스크로 분할하고 필요한 경우 복제하여 정렬 병합 조인의 스큐를 동적으로 처리합니다.

이 기능은 `spark.sql.adaptive.skewJoin.enabled`가 `true`로 설정된 경우 활성화됩니다. 자세한 내용은 [Apache Spark 설명서](#)를 참조하세요. 데이터 스큐는 [셔플 최적화](#) 섹션에서 자세히 설명합니다.

## Spark UI를 사용하여 성능 문제 조사

모범 사례를 적용하여 AWS Glue 작업의 성능을 조정하기 전에 성능을 프로파일링하고 병목 현상을 식별하는 것이 좋습니다. 이렇게 하면 올바른 업무에 집중할 수 있습니다.

빠른 분석을 위해 [Amazon CloudWatch 지표](#)는 작업 지표의 기본 보기를 제공합니다. [Spark UI](#)는 성능 조정을 위한 심층 보기를 제공합니다. AWS Glue에서 Spark UI를 사용하려면 [AWS Glue 작업에 대해 Spark UI를 활성화](#)해야 합니다. Spark UI에 익숙해지면 [Spark 작업 성능을 조정하는 전략](#)에 따라 조사 결과를 기반으로 병목 현상의 영향을 식별하고 줄입니다.

## Spark UI를 사용하여 병목 현상 식별

Spark UI를 열면 Spark 애플리케이션이 테이블에 나열됩니다. 기본적으로 AWS Glue 작업의 앱 이름은 nativespark-<Job Name>-<Job Run ID>입니다. 작업 실행 ID를 기반으로 대상 Spark 앱을 선택하여 작업 탭을 엽니다. 스트리밍 작업 실행과 같은 불완전한 작업 실행은 불완전한 애플리케이션 표시에 나열됩니다.

작업 탭에는 Spark 애플리케이션의 모든 작업에 대한 요약이 표시됩니다. 단계 또는 태스크 실패를 확인하려면 총 작업 수를 확인합니다. 병목 현상을 찾으려면 기간을 선택하여 정렬합니다. 설명 열에 표시된 링크를 선택하여 장기 실행 작업의 세부 정보를 자세히 살펴봅니다.

**Spark Jobs** (?)

User: spark  
 Total Uptime: 7.7 min  
 Scheduling Mode: FIFO  
 Completed Jobs: 7

▶ Event Timeline

◄ Completed Jobs (7)

Page: 1 1 Pages. Jump to 1. Show 100 items in a page. Go

Job id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
3	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2023/03/30 06:49:02	6.5 min	1/1 (1 skipped)	5/5 (799 skipped)
0	showString at NativeMethodAccessorImpl.java:0 showString at NativeMethodAccessorImpl.java:0	2023/03/30 06:48:15	29 s	1/1	799/799
2	parquet at NativeMethodAccessorImpl.java:0 parquet at NativeMethodAccessorImpl.java:0	2023/03/30 06:48:48	14 s	1/1	799/799

작업 세부 정보 페이지에는 단계가 나열됩니다. 이 페이지에서는 지속 시간, 성공한 태스크 수 및 총 태스크 수, 입력 및 출력 수, 셔플 읽기 및 셔플 쓰기 양과 같은 전체 인사이트를 볼 수 있습니다.

**Details for Job 3**

Status: SUCCEEDED  
 Submitted: 2023/03/30 06:49:02  
 Duration: 6.5 min  
 Associated SQL Query: 2  
 Completed Stages: 1  
 Skipped Stages: 1

▶ Event Timeline  
 ▶ DAG Visualization

▶ **Completed Stages (1)**

Page: 1 1 Pages. Jump to 1 . Show 100 items in a page. Go

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output	Shuffle Read	Shuffle Write
5	parquet at NativeMethodAccessorImpl.java:0	+details 2023/03/30 06:49:02	6.5 min	5/5		10.2 GiB	11.9 GiB	

실행기 탭에는 Spark 클러스터 용량이 자세히 표시됩니다. 총 코어 수를 확인할 수 있습니다. 다음 스크린샷에 표시된 클러스터에는 총 316개의 활성 코어와 512개의 코어가 포함되어 있습니다. 기본적으로 각 코어는 한 번에 하나의 Spark 태스크를 처리할 수 있습니다.

**Executors**

▶ Show Additional Metrics

**Summary**

	RDD Blocks	Storage Memory	Disk Used	Cores	Active Tasks	Failed Tasks	Complete Tasks	Total Tasks
<b>Active(80)</b>	0	0.0 B / 465.9 GiB	0.0 B	316	10	0	2399	2399
<b>Dead(49)</b>	0	0.0 B / 285.4 GiB	0.0 B	196	10	0	3	3
<b>Total(129)</b>	0	0.0 B / 751.3 GiB	0.0 B	512	10	0	2402	2402

작업 세부 정보 페이지에 표시된 5/5 값에 따라 5단계가 가장 긴 단계이지만 512개 중 5개의 코어만 사용합니다. 이 단계의 병렬화는 너무 낮지만 시간이 많이 걸리기 때문에 병목 현상으로 식별할 수 있습니다. 성능을 개선하기 위해 이유를 파악하길 원합니다. 일반적인 성능 병목 현상의 영향을 인식하고 줄이는 방법에 대해 자세히 알아보려면 [Spark 작업 성능 조정을 위한 전략](#)을 참조하세요.

# Spark 작업 성능 조정을 위한 전략

파라미터를 조정할 준비를 할 때는 다음과 같은 모범 사례를 고려하세요.

- 문제를 식별하기 전에 성능 목표를 결정합니다.
- 조정 파라미터를 변경하기 전에 지표를 사용하여 문제를 식별합니다.

작업을 조정할 때 가장 일관된 결과를 얻으려면 조정 작업에 대한 기준선 전략을 수립하세요.

## 성능 조정을 위한 기준선 전략

일반적으로 성능 조정은 다음 워크플로우에서 수행됩니다.

1. 성능 목표를 결정합니다.
2. 지표를 측정합니다.
3. 병목 현상을 식별합니다.
4. 병목 현상이 미치는 영향을 줄입니다.
5. 의도한 목표를 달성할 때까지 2~4단계를 반복합니다.

먼저 성능 목표를 결정합니다. 예를 들어 목표 중 하나는 3시간 이내에 AWS Glue 작업 실행을 완료하는 것일 수 있습니다. 목표를 정의한 후 작업 성능 지표를 측정합니다. 목표를 달성하기 위해 지표 및 병목 현상의 추세를 식별합니다. 특히 병목 현상 식별은 문제 해결, 디버깅 및 성능 조정에서 가장 중요합니다. Spark 애플리케이션을 실행하는 동안 Spark는 Spark 이벤트 로그에 각 태스크의 상태 및 통계를 기록합니다.

에서는 Spark 기록 서버에서 제공하는 [Spark 웹 UI](#)를 통해 Spark 지표를 볼 수 있습니다. AWS Glue for Spark 작업은 [Spark 이벤트 로그](#)를 Amazon S3에서 지정한 위치로 보낼 수 있습니다. AWS Glue 또한 예제 [AWS CloudFormation 템플릿](#)과 [Dockerfile](#)을 제공하여 Amazon EC2 인스턴스 또는 로컬 컴퓨터에서 Spark 기록 서버를 시작하므로 이벤트 로그와 함께 Spark UI를 사용할 수 있습니다.

성능 목표를 결정하고 이러한 목표를 평가하기 위한 지표를 식별한 후에 다음 섹션의 전략을 사용하여 병목 현상을 식별하고 해결할 수 있습니다.

## Spark 작업 성능에 대한 조정 사례

Spark 작업에 대한 성능 튜닝에는 다음 전략을 사용할 수 있는 AWS Glue 있습니다.

- AWS Glue 리소스:
  - [클러스터 용량 규모 조정](#)
  - [최신 AWS Glue 버전 사용](#)
- Spark 애플리케이션:
  - [데이터 스캔 양 감소](#)
  - [태스크 병렬화](#)
  - [셔플 최적화](#)
  - [계획 오버헤드 최소화](#)
  - [사용자 정의 함수 최적화](#)

이러한 전략을 사용하기 전에 Spark 작업에 대한 지표 및 구성에 액세스할 수 있어야 합니다. [AWS Glue 설명서](#)에서 이 정보를 확인할 수 있습니다.

AWS Glue 리소스 관점에서 AWS Glue 작업자를 추가하고 최신 AWS Glue 버전을 사용하여 성능을 개선할 수 있습니다.

Apache Spark 애플리케이션 관점에서 성능을 개선할 수 있는 여러 전략에 액세스할 수 있습니다. 불필요한 데이터가 Spark 클러스터에 로드되는 경우 이를 제거하여 로드된 데이터의 양을 줄일 수 있습니다. Spark 클러스터 리소스를 적게 사용하고 데이터 I/O가 적은 경우 병렬화할 태스크를 식별할 수 있습니다. 또한 상당한 시간이 걸리는 경우 조인과 같은 과중한 데이터 전송 작업을 최적화할 수 있습니다. 작업 쿼리 계획을 최적화하거나 개별 Spark 태스크의 계산 복잡성을 줄일 수도 있습니다.

이러한 전략을 효율적으로 적용하려면 지표를 참조하여 적용 가능한 시기를 식별해야 합니다. 자세한 내용은 다음의 각 섹션을 참조하세요. 이러한 기법은 성능 조정뿐만 아니라 메모리 부족(OOM) 오류와 같은 일반적인 문제를 해결하는 데도 효과적입니다.

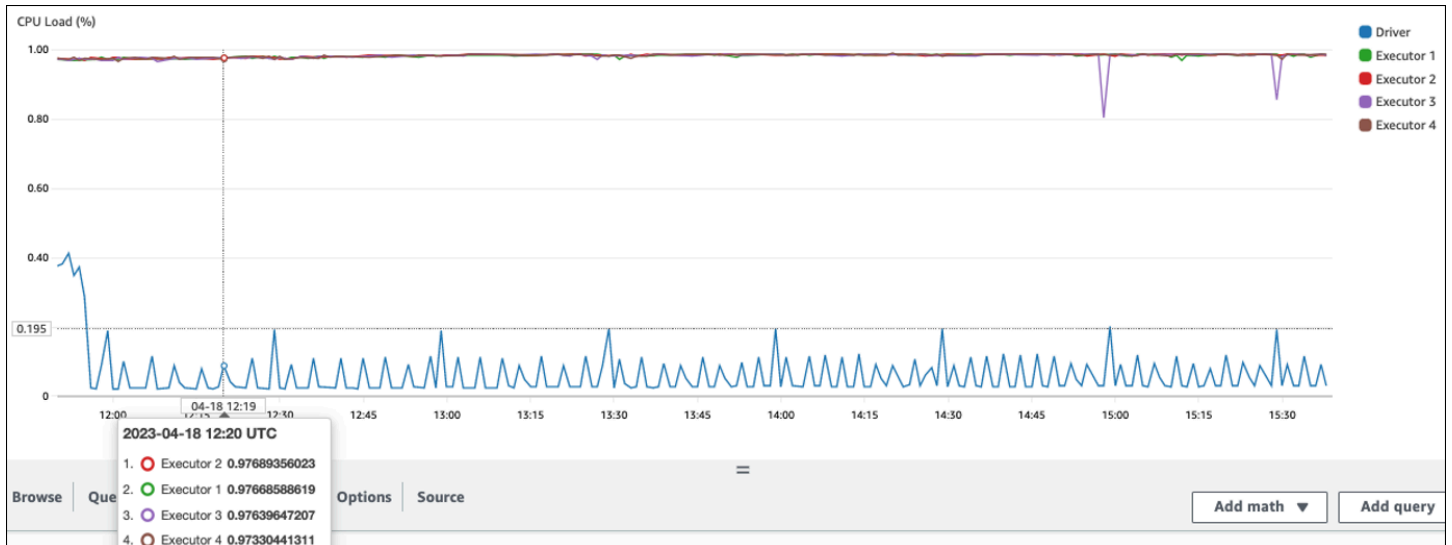
## 클러스터 용량 규모 조정

작업이 너무 오래 걸리지만 실행기가 충분한 리소스를 소비하고 Spark가 사용 가능한 코어에 비해 대량의 태스크를 생성하는 경우 클러스터 용량을 조정하는 방법을 고려합니다. 적절한지 평가하려면 다음 지표를 사용합니다.

## CloudWatch 지표

- CPU 로드 및 메모리 사용률을 확인하여 실행기가 충분한 리소스를 사용하고 있는지 확인합니다.
- 작업 실행 시간을 확인하여 처리 시간이 너무 길어 성능 목표를 달성할 수 없는지 평가합니다.

다음 예제에서는 4개의 실행기가 97%를 초과하는 CPU 로드에서 실행되고 있지만 약 3시간 후에도 처리가 완료되지 않습니다.



### Note

CPU 부하가 낮으면 클러스터 용량을 조정해도 도움이 되지 않을 수 있습니다.

## Spark UI

작업 탭 또는 단계 탭에서 각 작업 또는 단계의 태스크 수를 볼 수 있습니다. 다음 예제에서 Spark는 58100 태스크를 생성했습니다.

Stages for All Jobs					
Completed Stages: 1					
- Completed Stages (1)					
Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input
0	count at DynamicFrame.scala:1414	2023/04/18 10:59:10	4.8 h	58100/58100	28.4 GB

실행기 탭에서 총 실행기 및 태스크 수를 볼 수 있습니다. 다음 스크린샷에서 각 Spark 실행기에는 4개의 코어가 있으며 4개의 태스크를 동시에 수행할 수 있습니다.

Executor ID	Address	Status	RDD Blocks	Storage Memory	Disk Used	Cores
driver	172.35.229.149:37603	Active	0	0.0 B / 6.3 GB	0.0 B	0
1	172.34.249.100:34733	Active	0	0.0 B / 6.3 GB	0.0 B	4
2	172.35.72.25:38929	Active	0	0.0 B / 6.3 GB	0.0 B	4
3	172.34.49.138:39961	Active	0	0.0 B / 6.3 GB	0.0 B	4
4	172.36.70.76:39323	Active	0	0.0 B / 6.3 GB	0.0 B	4

이 예제에서 Spark 태스크 수(58100)는 실행기가 동시에 처리할 수 있는 16개의 태스크(4개의 실행기 × 4개의 코어)보다 훨씬 큼니다.

이러한 증상이 관찰되면 클러스터 규모 조정을 고려합니다. 다음 옵션을 사용하여 클러스터 용량을 조정할 수 있습니다.

- Enable AWS Glue Auto Scaling - [Auto Scaling](#)은 AWS Glue 버전 3.0 이상의 AWS Glue 추출, 변환 및 로드(ETL) 및 스트리밍 작업에 사용할 수 있습니다.는 각 단계의 파티션 수 또는 작업 실행 시 마이크로배치가 생성되는 속도에 따라 클러스터에서 작업자를 AWS Glue 자동으로 추가하고 제거합니다.

오토 스케일링이 활성화되어 있더라도 작업자 수가 증가하지 않는 상황이 관찰되면 작업자를 수동으로 추가하는 방법을 고려합니다. 그러나 한 단계에 대해 수동으로 규모를 조정하면 이후 단계에서 많은 작업자가 유휴 상태가 되어 성능이 개선되지 않으면서 비용이 더 많이 들 수 있습니다.

오토 스케일링을 활성화하면 CloudWatch 실행기 지표에서 실행기 수를 볼 수 있습니다. 다음 지표를 사용하여 Spark 애플리케이션의 실행기 수요를 모니터링합니다.

- `glue.driver.ExecutorAllocationManager.executors.numberAllExecutors`
- `glue.driver.ExecutorAllocationManager.executors.numberMaxNeededExecutors`

지표에 대한 자세한 내용은 [Amazon CloudWatch 지표 AWS Glue 를 사용한 모니터링](#)을 참조하세요.

- 스케일 아웃: AWS Glue 작업자 수 증가 - AWS Glue 작업자 수를 수동으로 늘릴 수 있습니다. 유휴 작업자를 관찰할 때까지만 작업자를 추가합니다. 이때 작업자를 더 추가하면 결과가 개선되지 않고 비용이 증가합니다. 자세한 내용은 [태스크 병렬화](#)를 참조하세요.
- 스케일 업: 더 큰 작업자 유형 사용 - 코어, 메모리 및 스토리지가 더 많은 작업자를 사용하도록 AWS Glue 작업자의 인스턴스 유형을 수동으로 변경할 수 있습니다. 작업자 유형이 클수록 메모리 집약적인 데이터 변환, 왜곡된 집계, 페타바이트 규모의 데이터와 관련된 엔터티 감지 검사 등 집약적인 데이터 통합 작업을 수직적으로 스케일링하고 실행할 수 있습니다.

작업 쿼리 계획이 상당히 크기 때문에 Spark 드라이버에 더 큰 용량이 필요한 경우에도 스케일 업이 도움이 됩니다. 작업자 유형 및 성능에 대한 자세한 내용은 [AWS 빅 데이터 블로그 게시물 더 큰 새 작업자 유형 G.4X 및 G.8X로 Apache Spark 작업에 AWS Glue 맞게 조정을 참조하세요.](#)

또한 더 큰 작업자를 사용하면 필요한 총 작업자 수를 줄일 수 있으므로 조인과 같은 집약적인 작업에서 셔플을 줄여 성능이 개선됩니다.

## 최신 AWS Glue 버전 사용

최신 AWS Glue 버전을 사용하는 것이 좋습니다. 작업 성능을 자동으로 개선할 수 있는 여러 최적화 및 업그레이드가 각 버전에 빌드되어 있습니다. 예를 들어 AWS Glue 4.0은 다음과 같은 새로운 기능을 제공합니다.

- 새로 최적화된 Apache Spark 3.3.0 런타임 - AWS Glue 4.0은 Apache Spark 3.3.0 런타임을 기반으로 구축되어 오픈 소스 Spark에 비교 가능한 성능 개선을 제공합니다. Spark 3.3.0 런타임은 Spark 2.x의 많은 혁신을 기반으로 빌드되었습니다.
- 향상된 Amazon Redshift 커넥터 - AWS Glue 4.0 이상 버전은 Apache Spark에 Amazon Redshift 통합을 제공합니다. 통합은 기존 오픈 소스 커넥터를 기반으로 빌드되며 성능과 보안을 위해 기능을 개선합니다. 통합을 통해 애플리케이션 성능이 최대 10배 빨라집니다. 자세한 내용은 [Amazon Redshift integration with Apache Spark](#)에 대한 블로그 게시물을 참조하세요.
- CSV 및 JSON 데이터를 사용한 벡터화된 읽기에 대한 SIMD 기반 실행 - AWS Glue 버전 3.0 이상 버전은 행 기반 리더에 비해 전체 작업 성능을 크게 높일 수 있는 최적화된 리더를 추가합니다. CSV 데이터에 대한 자세한 내용은 [Optimize read performance with vectorized SIMD CSV reader](#)를 참조하세요. JSON 데이터에 대한 자세한 내용은 [Using vectorized SIMD JSON reader with Apache Arrow columnar format](#)을 참조하세요.

각 AWS Glue 버전에는 커넥터, 드라이버 및 라이브러리 업데이트를 포함하여 정렬의 업그레이드가 포함됩니다. 자세한 내용은 [AWS Glue 버전](#) 및 [버전 4.0으로 AWS Glue AWS Glue 작업 마이그레이션](#)을 참조하세요.

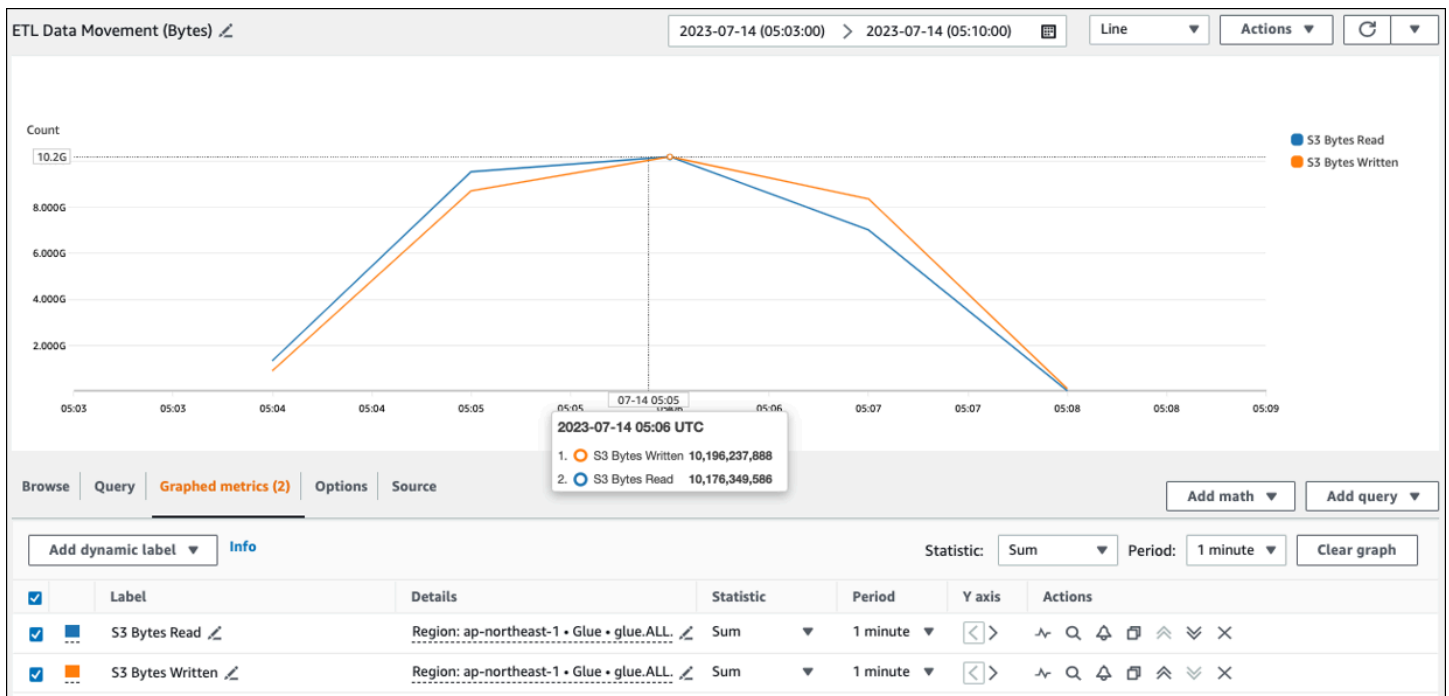
## 데이터 스캔 양 감소

시작할 때 필요한 데이터만 로드하는 방법을 고려합니다. 각 데이터 소스에 대해 Spark 클러스터에 로드되는 데이터의 양을 줄임으로써 성능을 개선할 수 있습니다. 이 접근 방식이 적절한지 평가하려면 다음 지표를 사용합니다.

[Spark UI](#) 섹션에 설명된 대로 [CloudWatch 지표](#)에서 Amazon S3의 읽기 바이트 및 Spark UI에서 추가 세부 정보를 확인할 수 있습니다.

## CloudWatch 지표

[ETL 데이터 이동\(바이트\)](#)에서 Amazon S3의 대략적인 읽기 크기를 확인할 수 있습니다. 이 지표는 이전 보고서 이후 모든 실행기가 Amazon S3에서 읽은 바이트 수를 보여줍니다. 이를 사용하여 Amazon S3의 ETL 데이터 이동을 모니터링할 수 있으며, 읽기와 외부 데이터 소스의 수집 속도를 비교할 수 있습니다.



예상보다 큰 S3 바이트 읽기 데이터 포인트가 관찰되는 경우 다음 솔루션을 고려합니다.

## Spark UI

Spark UI AWS Glue 용의 스테이지 탭에서 입력 및 출력 크기를 볼 수 있습니다. 다음 예제의 경우 2단계에서는 47.4GiB 입력 및 47.7GiB 출력을 읽는 반면, 5단계에서는 61.2MiB 입력 및 56.6MiB 출력을 읽습니다.

### Stages for All Jobs

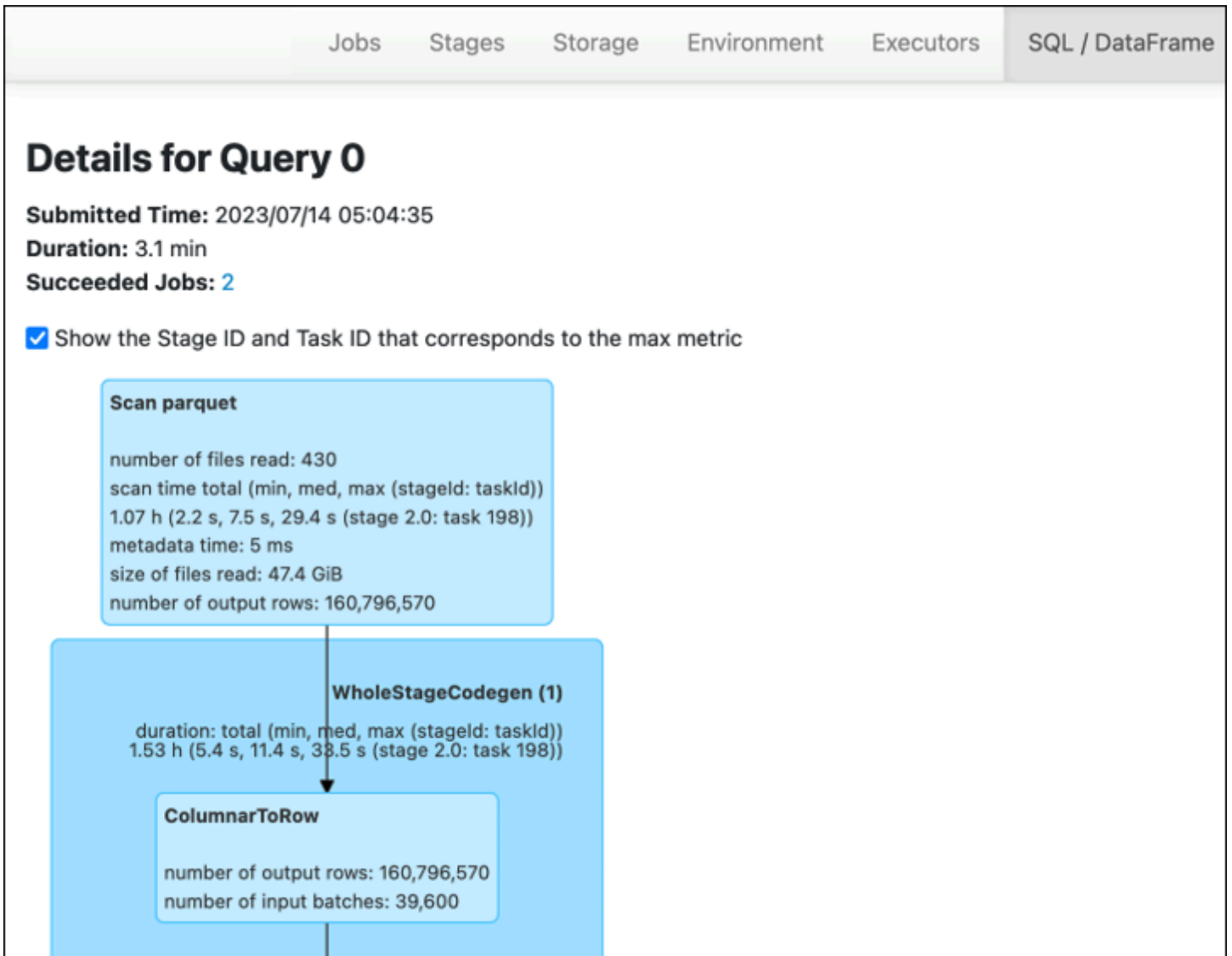
Completed Stages: 6

▼ Completed Stages (6)

Page: 1 1 Pages. Jump to 1 . Show

Stage Id ▾	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output
5	<a href="#">parquet at NativeMethodAccessorImpl.java:0</a> <a href="#">+details</a>	2023/07/14 05:09:49	15 s	414/414	61.2 MiB	56.6 MiB
4	<a href="#">load at NativeMethodAccessorImpl.java:0</a> <a href="#">+details</a>	2023/07/14 05:09:47	0.6 s	1/1		
3	Listing leaf files and directories for 43 paths: s3://amazon-reviews-pds/parquet/product_category=Apparel, ... <a href="#">load at NativeMethodAccessorImpl.java:0</a> <a href="#">+details</a>	2023/07/14 05:09:46	1 s	43/43		
2	<a href="#">parquet at NativeMethodAccessorImpl.java:0</a> <a href="#">+details</a>	2023/07/14 05:04:36	3.1 min	414/414	47.4 GiB	47.7 GiB
1	<a href="#">load at NativeMethodAccessorImpl.java:0</a> <a href="#">+details</a>	2023/07/14 05:04:31	2 s	1/1		
0	Listing leaf files and directories for 43 paths: s3://amazon-reviews-pds/parquet/product_category=Apparel, ... <a href="#">load at NativeMethodAccessorImpl.java:0</a> <a href="#">+details</a>	2023/07/14 05:04:13	6 s	43/43		

AWS Glue 작업에서 Spark SQL 또는 DataFrame 접근 방식을 사용하면 SQL / DataFrame 탭에 이러한 단계에 대한 더 많은 통계가 표시됩니다. 이 경우 2단계에서 읽은 파일 수: 430, 읽은 파일 크기: 47.4GiB, 출력 행 수: 160,796,570이 표시됩니다.



읽고 있는 데이터와 사용 중인 데이터 사이에서 크기에 상당한 차이가 있는 경우 다음 솔루션을 시도해 보세요.

### Amazon S3

Amazon S3에서 읽을 때 작업에 로드되는 데이터의 양을 줄이려면 데이터 세트의 파일 크기, 압축, 파일 형식 및 파일 레이아웃(파티션)을 고려하세요. AWS Glue Spark 작업은 원시 데이터의 ETL에 자주 사용되지만 효율적인 분산 처리를 위해서는 데이터 소스 형식의 기능을 검사해야 합니다.

- 파일 크기 - 입력 및 출력의 파일 크기를 중간 범위(예: 128MB) 이내로 유지하는 것이 좋습니다. 너무 작은 파일과 너무 큰 파일은 문제가 될 수 있습니다.

작은 파일이 많으면 다음과 같은 문제가 발생합니다.

- 많은 객체에 대한 요청(예: List, Get 또는 Head)을 수행하는 데 필요한 오버헤드로 인한 Amazon S3에서 과도한 네트워크 I/O 로드(동일한 양의 데이터를 저장하는 몇 개의 객체와 비교한 경우).
- Spark 드라이버에서 과도한 I/O 및 처리 로드가 발생하여 많은 파티션과 태스크가 생성되고 과도한 병렬 처리가 발생합니다.

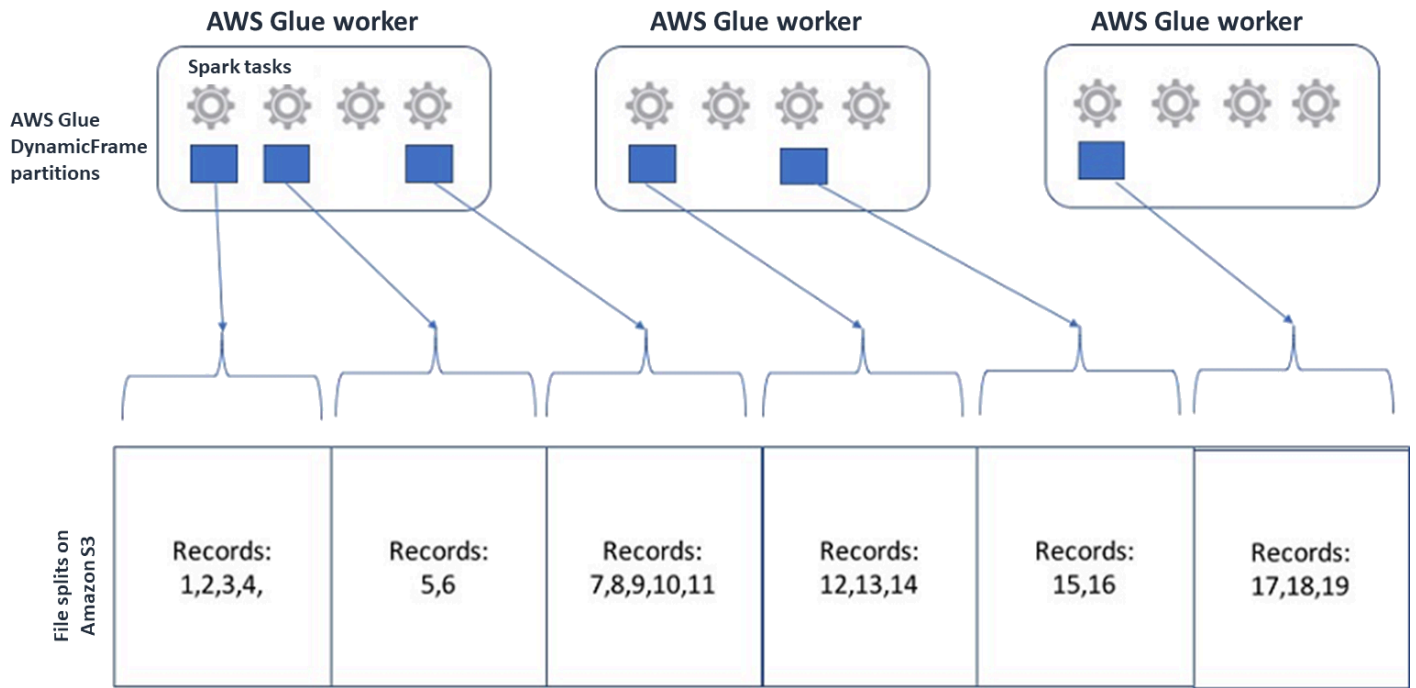
반면 파일 유형이 분할 가능하지 않고(예: gzip) 파일이 너무 큰 경우 단일 태스크가 전체 파일 읽기를 완료할 때까지 Spark 애플리케이션이 기다려야 합니다.

각 작은 파일에 대해 Apache Spark 태스크가 생성될 때 발생하는 과도한 병렬 처리를 줄이려면 [DynamicFrames에 파일 그룹화](#)를 사용합니다. 이 접근 방식은 Spark 드라이버에서 OOM 예외가 발생할 가능성을 줄입니다. 파일 그룹화를 구성하려면 `groupFiles` 및 `groupSize` 파라미터를 설정합니다. 다음 코드 예제에서는 이러한 파라미터와 함께 ETL 스크립트에서 AWS Glue `DynamicFrame` API를 사용합니다.

```
dyf = glueContext.create_dynamic_frame_from_options("s3",
    {'paths': ["s3://input-s3-path/"],
    'recurse': True,
    'groupFiles': 'inPartition',
    'groupSize': '1048576'},
    format="json")
```

- 압축 - S3 객체가 수백 메가바이트인 경우 압축하는 것이 좋습니다. 다양한 압축 형식이 있으며, 크게 두 가지 유형으로 분류할 수 있습니다.
- gzip과 같은 분할할 수 없는 압축 형식을 사용하려면 한 작업자가 전체 파일의 압축을 풀어야 합니다.
- bzip2 또는 LZO(인덱스)와 같은 분할 가능한 압축 형식은 병렬화할 수 있는 파일의 부분 압축 해제를 허용합니다.

Spark 및 기타 일반적인 분산 처리 엔진의 경우 소스 데이터 파일을 엔진이 병렬로 처리할 수 있는 청크로 분할합니다. 이러한 단위를 종종 분할이라고 합니다. 데이터가 분할 가능한 형식이면 최적화된 AWS Glue 리더는 `GetObject` API에 특정 블록만 검색하는 `Range` 옵션을 제공하여 S3 객체에서 분할을 검색할 수 있습니다. 다음 다이어그램을 고려하여 실제로 어떻게 작동하는지 확인합니다.



압축된 데이터는 파일이 최적의 크기이거나 파일이 분할 가능한 한 애플리케이션의 속도를 크게 높일 수 있습니다. 데이터 크기가 작을수록 Amazon S3에서 스캔한 데이터와 Amazon S3에서 Spark 클러스터로의 네트워크 트래픽이 줄어듭니다. 반면 데이터를 압축 및 압축 해제하려면 더 많은 CPU가 필요합니다. 필요한 컴퓨팅 양은 압축 알고리즘의 압축 비율에 따라 조정됩니다. 분할 가능한 압축 형식을 선택할 경우 이 장단점을 고려합니다.

### Note

gzip 파일은 일반적으로 분할할 수 없지만 개별 Parquet 블록을 gzip으로 압축할 수 있으며 이러한 블록은 병렬화할 수 있습니다.

- 파일 형식 - 열 형식을 사용합니다. [Apache Parquet](#) 및 [Apache ORC](#)는 널리 사용되는 열 기반 데이터 형식입니다. Parquet 및 ORC는 열 기반 압축을 사용하고 데이터 유형에 따라 각 열을 인코딩 및 압축하여 데이터를 효율적으로 저장합니다. Parquet 인코딩에 대한 자세한 내용은 [Parquet encoding definitions](#)를 참조하세요. Parquet 파일도 분할할 수 있습니다.

열 형식은 열별로 값을 그룹화하고 블록에 함께 저장합니다. 열 기반 형식을 사용하는 경우 사용할 계획이 없는 열에 해당하는 데이터 블록을 건너뛸 수 있습니다. Spark 애플리케이션에서는 필요한 열만 검색할 수 있습니다. 일반적으로 압축률이 높거나 데이터 블록을 건너뛰면 Amazon S3에서 바이트를 더 적게 읽어 성능이 향상됩니다. 두 형식 모두 I/O를 줄이기 위해 다음과 같은 푸시다운 접근 방식도 지원합니다.

- 프로젝션 푸시다운 - 프로젝션 푸시다운은 애플리케이션에 지정된 열만 검색하는 기법입니다. 다음 예제와 같이 Spark 애플리케이션에서 열을 지정합니다.
  - DataFrame 예제: `df.select("star_rating")`
  - Spark SQL 예제: `spark.sql("select star_rating from <table>")`
- 조건자 푸시다운 - 조건자 푸시다운은 WHERE 및 GROUP BY 절을 효율적으로 처리하는 기법입니다. 두 형식 모두에는 열 값을 나타내는 데이터 블록이 있습니다. 각 블록에는 최댓값 및 최솟값과 같은 블록에 대한 통계가 들어 있습니다. Spark는 이러한 통계를 사용하여 애플리케이션에 사용되는 필터 값에 따라 블록을 읽거나 건너뛴지를 결정할 수 있습니다. 이 기능을 사용하려면 다음과 같이 다음 예제에 나온 대로 조건에 필터를 더 추가합니다.
  - DataFrame 예제: `df.select("star_rating").filter("star_rating < 2")`
  - Spark SQL 예제: `spark.sql("select * from <table> where star_rating < 2")`
- 파일 레이아웃 - 데이터 사용 방식에 따라 S3 데이터를 다른 경로의 객체에 저장하면 관련 데이터를 효율적으로 검색할 수 있습니다. 자세한 내용은 Amazon S3 설명서의 [접두사를 사용한 객체 구성](#)을 참조하세요. AWS Glue 는 key=value 형식의 Amazon S3 접두사에 키와 값을 저장하고 Amazon S3 경로를 통해 데이터를 분할할 수 있도록 지원합니다. 데이터를 분할하면 각 다운스트림 분석 애플리케이션이 스캔하는 데이터의 양을 제한하여 성능을 향상시키고 비용을 절감할 수 있습니다. 자세한 내용은 [에서 ETL 출력을 위한 파티션 관리를 참조하세요 AWS Glue](#).

분할은 테이블을 여러 부분으로 나누고 다음 예제와 같이 year, month, day 등의 열 값을 기반으로 관련 데이터를 그룹화된 파일에 보관합니다.

```
# Partitioning by /YYYY/MM/DD
s3://<YourBucket>/year=2023/month=03/day=31/0000.gz
s3://<YourBucket>/year=2023/month=03/day=01/0000.gz
s3://<YourBucket>/year=2023/month=03/day=02/0000.gz
s3://<YourBucket>/year=2023/month=03/day=03/0000.gz
...
```

AWS Glue Data Catalog의 테이블로 데이터세트를 모델링하여 데이터세트의 파티션을 정의할 수 있습니다. 그런 다음 다음과 같이 파티션 정리를 사용하여 데이터 스캔 양을 제한할 수 있습니다.

- For AWS Glue DynamicFrame, 설정 `push_down_predicate`(또는 `catalogPartitionPredicate`).

```
dyf = Glue_context.create_dynamic_frame.from_catalog(
    database=src_database_name,
    table_name=src_table_name,
    push_down_predicate = "year='2023' and month = '03'",
```

```
)
```

- Spark DataFrame의 경우 파티션을 정리할 고정 경로를 설정합니다.

```
df = spark.read.format("json").load("s3://<YourBucket>/year=2023/month=03/*/*.gz")
```

- Spark SQL의 경우 데이터 카탈로그에서 파티션을 정리하도록 where 절을 설정할 수 있습니다.

```
df = spark.sql("SELECT * FROM <Table> WHERE year= '2023' and month = '03'")
```

- 데이터를 쓸 때 날짜별로 분할하려면 다음과 같이 열의 날짜 정보로 DataFrame의 DynamicFrame 또는 [partitionBy\(\)에서 partitionKeys](#)를 AWS Glue 설정합니다.

- DynamicFrame

```
glue_context.write_dynamic_frame_from_options(
    frame= dyf, connection_type='s3',format='parquet'
    connection_options= {
        'partitionKeys': ["year", "month", "day"],
        'path': 's3://<YourBucket>/<Prefix>/'
    }
)
```

- DataFrame

```
df.write.mode('append')\
    .partitionBy('year', 'month', 'day')\
    .parquet('s3://<YourBucket>/<Prefix>/')
```

이렇게 하면 출력 데이터의 소비자 성능이 향상될 수 있습니다.

입력 데이터셋을 생성하는 파이프라인을 변경할 수 있는 액세스 권한이 없는 경우 분할은 옵션이 아닙니다. 대신 glob 패턴을 사용하여 불필요한 S3 경로를 제외할 수 있습니다. DynamicFrame에서 읽을 때 [제외](#)를 설정합니다. 예를 들어 다음 코드는 2023년 1월~9월의 일수를 제외합니다.

```
dyf = glueContext.create_dynamic_frame.from_catalog(
    database=db,
    table_name=table,
    additional_options = { "exclusions": "[\\""**year=2023/month=0[1-9]**\\"" ] },
    transformation_ctx='dyf'
)
```

Data Catalog의 테이블 속성에서 제외를 설정할 수도 있습니다.

- 키: exclusions
- 값: ["\*\*year=2023/month=0[1-9]\*\*"]
- Amazon S3 파티션이 너무 많음 - 값이 수천 개인 ID 열과 같이 광범위한 값을 포함하는 열에서 Amazon S3 데이터를 분할하지 마세요. 그러면 가능한 파티션 수는 파티션을 분할한 모든 필드의 곱이므로 버킷의 파티션 수가 크게 증가할 수 있습니다. 파티션이 너무 많으면 다음과 같은 상황이 나타날 수 있습니다.
  - Data Catalog에서 파티션 메타데이터를 검색할 경우 지연 시간 증가
  - 작은 파일 수 증가(더 많은 Amazon S3 API 요청(List, Get, Head)이 필요함)

예를 들어 partitionBy 또는 partitionKeys에서 날짜 유형을 설정하면 yyyy/mm/dd와 같은 날짜 수준 분할이 많은 사용 사례에 적합합니다. 그러나 yyyy/mm/dd/<ID>는 너무 많은 파티션을 생성하여 전반적으로 성능에 부정적인 영향을 미칠 수 있습니다.

반면 실시간 처리 애플리케이션과 같은 일부 사용 사례에는 yyyy/mm/dd/hh와 같은 많은 파티션이 필요합니다. 사용 사례에 상당한 파티션이 필요한 경우 [AWS Glue 파티션 인덱스](#)를 사용하여 Data Catalog에서 파티션 메타데이터를 검색하는 데 걸리는 지연 시간을 줄이는 방법을 고려합니다.

## 데이터베이스 및 JDBC

데이터베이스에서 정보를 검색할 때 데이터 스캔을 줄이려면 SQL 쿼리에서 where 조건자(또는 절)를 지정할 수 있습니다. SQL 인터페이스를 제공하지 않는 데이터베이스는 쿼리 또는 필터링을 위한 자체 메커니즘을 제공합니다.

Java Database Connectivity(JDBC) 연결을 사용하는 경우 다음 파라미터에 대한 where 절과 함께 선택 쿼리를 제공합니다.

- DynamicFrame의 경우 [sampleQuery](#) 옵션을 사용합니다. create\_dynamic\_frame.from\_catalog를 사용할 때 다음과 같이 additional\_options 인수를 구성합니다.

```
query = "SELECT * FROM <TableName> where id = 'XX' AND"
datasource0 = glueContext.create_dynamic_frame.from_catalog(
    database = db,
    table_name = table,
    additional_options={
        "sampleQuery": query,
```

```

    "hashexpression": key,
    "hashpartitions": 10,
    "enablePartitioningForSampleQuery": True
  },
  transformation_ctx = "datasource0"
)

```

using `create_dynamic_frame.from_options`에서 다음과 같이 `connection_options` 인수를 구성합니다.

```

query = "SELECT * FROM <TableName> where id = 'XX' AND"
datasource0 = glueContext.create_dynamic_frame.from_options(
  connection_type = connection,
  connection_options={
    "url": url,
    "user": user,
    "password": password,
    "dbtable": table,
    "sampleQuery": query,
    "hashexpression": key,
    "hashpartitions": 10,
    "enablePartitioningForSampleQuery": True
  }
)

```

- DataFrame의 경우 [쿼리](#) 옵션을 사용합니다.

```

query = "SELECT * FROM <TableName> where id = 'XX'"
jdbcDF = spark.read \
  .format('jdbc') \
  .option('url', url) \
  .option('user', user) \
  .option('password', pwd) \
  .option('query', query) \
  .load()

```

- Amazon Redshift의 경우 Amazon [Redshift Spark 커넥터](#)에서 푸시다운 지원을 활용하려면 AWS Glue 4.0 이상을 사용합니다.

```

dyf = glueContext.create_dynamic_frame.from_catalog(
  database = "redshift-dc-database-name",
  table_name = "redshift-table-name",

```

```

redshift_tmp_dir = args["temp-s3-dir"],
additional_options = {"aws_iam_role": "arn:aws:iam::role-account-id:role/rs-role-
name"}
)

```

- 다른 데이터베이스의 경우 해당 데이터베이스에 대한 설명서를 참조하세요.

## AWS Glue 옵션

- 모든 연속 작업 실행에 대한 전체 스캔을 방지하고 마지막 작업 실행 중에 존재하지 않은 데이터만 처리하려면 [작업 북마크](#)를 활성화합니다.
- 처리할 입력 데이터의 양을 제한하려면 작업 북마크를 사용하여 [제한된 실행](#)을 활성화합니다. 이렇게 하면 각 작업 실행에 대해 스캔한 데이터의 양을 줄일 수 있습니다.

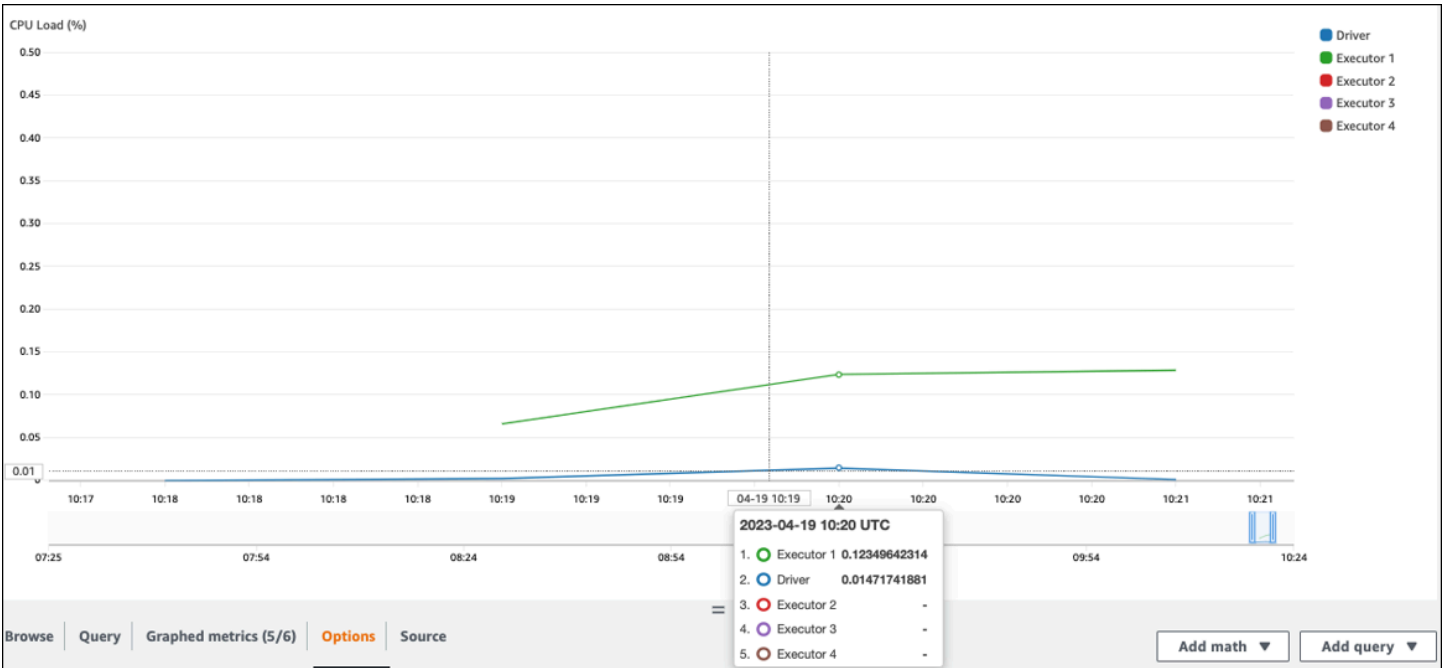
## 태스크 병렬화

성능을 최적화하려면 데이터 로드 및 변환에 대한 태스크를 병렬화하는 것이 중요합니다. [Apache Spark의 주요 주제](#)에서 설명한 것처럼 복원력이 뛰어난 분산 데이터세트(RDD) 파티션의 수는 병렬화 수준을 결정하기 때문에 중요합니다. Spark가 생성하는 각 태스크는 1:1 기준으로 RDD 파티션에 해당합니다. 최상의 성능을 얻으려면 RDD 파티션 수를 결정하는 방법과 해당 수를 최적화하는 방법을 이해해야 합니다.

병렬 처리가 충분하지 않은 경우 [CloudWatch 지표](#) 및 Spark UI에 다음 증상이 기록됩니다.

## CloudWatch 지표

CPU 로드 및 메모리 사용률을 확인합니다. 작업 단계에서 일부 실행기가 처리 중이 아닌 경우 병렬 처리를 개선하는 것이 좋습니다. 이 경우 시각화된 기간에 실행기 1이 태스크를 수행했지만 나머지 실행기(2, 3, 4)는 수행하지 않았습니다. Spark 드라이버에서 해당 실행기에 태스크가 할당되지 않았다고 추론할 수 있습니다.

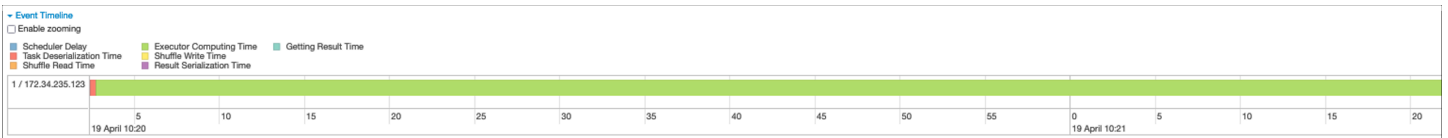


## Spark UI

Spark UI의 Stage 탭에서 단계의 태스크 수를 볼 수 있습니다. 이 경우 Spark는 하나의 태스크만 수행했습니다.

Index	ID	Attempt	Status	Locality Level	Executor ID	Host	Launch Time	Duration	Task Deserialization Time	GC Time	Result Serialization Time	Input Size / Records	Write Time	Shuffle Write Size / Records
0	1	0	SUCCESS	ANY	1	172.34.235.123	2023/04/19 10:20:02	1.3 min	0.3 s	0.4 s	1 ms	2.0 GB / 7135819	12 ms	59.0 B / 1

또한 이벤트 타임라인에는 실행기 1이 하나의 태스크를 처리하는 것으로 표시됩니다. 즉, 이 단계의 작업은 하나의 실행기에서 완전히 수행되었고 다른 실행기는 유휴 상태였습니다.



이러한 증상이 관찰되면 각 데이터 소스에 대해 다음 솔루션을 시도해 보세요.

## Amazon S3에서 데이터 로드 병렬화

Amazon S3에서 데이터 로드를 병렬화하려면 먼저 기본 파티션 수를 확인합니다. 그런 다음 대상 파티션 수를 수동으로 결정할 수 있지만 파티션이 너무 많지 않도록 해야 합니다.

### 기본 파티션 수 결정

Amazon S3의 경우 초기 Spark RDD 파티션 수(각각 Spark 태스크에 해당)는 Amazon S3 데이터세트의 기능(예: 형식, 압축 및 크기)에 따라 결정됩니다. Amazon S3에 저장된 CSV 객체에서 AWS Glue DynamicFrame 또는 Spark DataFrame을 생성하면 다음과 같이 초기 RDD 파티션(NumPartitions) 수를 대략적으로 계산할 수 있습니다.

- 객체 크기가 64MB 이하인 경우:  $\text{NumPartitions} = \text{Number of Objects}$
- 객체 크기가 64MB를 초과하는 경우:  $\text{NumPartitions} = \text{Total Object Size} / 64 \text{ MB}$
- 분할 불가(gzip):  $\text{NumPartitions} = \text{Number of Objects}$

[데이터 스캔 양 감소](#) 섹션에서 설명한 대로 Spark는 큰 S3 객체를 병렬로 처리할 수 있는 분할로 나눕니다. 객체가 분할 크기보다 크면 Spark는 객체를 분할하고 각 분할에 대해 RDD 파티션(및 태스크)을 생성합니다. Spark의 분할 크기는 데이터 형식과 런타임 환경에 기반하지만 이는 합리적인 시작 근사치입니다. 일부 객체는 gzip과 같은 분할할 수 없는 압축 형식을 사용하여 압축되므로 Spark는 이를 분할할 수 없습니다.

NumPartitions 값은 데이터 형식, 압축, AWS Glue 버전, AWS Glue 작업자 수 및 Spark 구성에 따라 다를 수 있습니다.

예를 들어 Spark DataFrame을 사용하여 단일 10GB csv.gz 객체를 로드하면 gzip은 분할할 수 없으므로 Spark 드라이버는 하나의 RDD 파티션(NumPartitions=1)만 생성합니다. 따라서 다음 그림에 설명된 대로 하나의 특정 Spark 실행기에 과도한 로드가 생기고 나머지 실행기에 태스크가 할당되지 않습니다.

[Spark Web UI](#) 단계 탭에서 단계의 실제 태스크 수(NumPartitions)를 확인하거나 코드에서 `df.rdd.getNumPartitions()`를 실행하여 병렬화를 확인합니다.

10GB gzip 파일이 생기면 해당 파일을 생성하는 시스템이 분할 가능한 형식으로 파일을 생성할 수 있는지 검사합니다. 옵션이 아닌 경우 [클러스터 용량을 조정](#)하여 파일을 처리해야 할 수 있습니다. 로드한 데이터에 대해 변환을 효율적으로 실행하려면 다시 분할을 사용하여 클러스터의 작업자 간에 RDD를 재조정해야 합니다.

수동으로 대상 파티션 수 확인

데이터의 속성과 Spark의 특정 기능 구현에 따라 기본 작업을 계속 병렬화할 수 있더라도 NumPartitions 값이 낮아질 수 있습니다. NumPartitions가 너무 작은 경우 `df.repartition(N)`을 실행하여 파티션 수를 늘리면 처리가 여러 Spark 실행기에 분산될 수 있습니다.

이 경우 `df.repartition(100)`을 실행하면 `NumPartitions`가 1에서 100으로 증가하고 각각 다른 실행기에 할당할 수 있는 태스크를 포함하는 100개의 데이터 파티션이 생성됩니다.

`repartition(N)` 작업은 전체 데이터를 균등하게 나누어(10GB/100개 파티션 = 100MB/파티션) 특정 파티션에 대한 데이터 스큐를 방지합니다.

### Note

`join`과 같은 셔플 작업이 실행되면 `spark.sql.shuffle.partitions` 또는 `spark.default.parallelism`의 값에 따라 파티션 수가 동적으로 증가하거나 감소합니다. 이를 통해 Spark 실행기 사이에서 데이터를 보다 효율적으로 교환할 수 있습니다. 자세한 내용은 [Spark 설명서](#)를 참조하세요.

파티션의 목표 수를 결정할 때 목표는 프로비저닝된 AWS Glue 작업자의 사용을 극대화하는 것입니다. AWS Glue 작업자 수와 Spark 작업 수는 vCPUs. Spark는 각 vCPU 코어에 대해 하나의 태스크를 지원합니다. AWS Glue 버전 3.0 이상에서는 다음 공식을 사용하여 대상 파티션 수를 계산할 수 있습니다.

```
# Calculate NumPartitions by WorkerType
numExecutors = (NumberOfWorkers - 1)
numSlotsPerExecutor =
  4 if WorkerType is G.1X
  8 if WorkerType is G.2X
  16 if WorkerType is G.4X
  32 if WorkerType is G.8X
NumPartitions = numSlotsPerExecutor * numExecutors

# Example: Glue 4.0 / G.1X / 10 Workers
numExecutors = ( 10 - 1 ) = 9 # 1 Worker reserved on Spark Driver
numSlotsPerExecutor = 4 # G.1X has 4 vCpu core ( Glue 3.0 or later )
NumPartitions = 9 * 4 = 36
```

이 예제에서 각 G.1X 작업자는 Spark 실행기(`spark.executor.cores = 4`)에 4개의 vCPU 코어를 제공합니다. Spark는 각 vCPU 코어에 대해 하나의 태스크를 지원하므로 G.1X Spark 실행기는 4개의 태스크(`numSlotPerExecutor`)를 동시에 실행할 수 있습니다. 태스크에 동일한 시간이 걸리는 경우 이 파티션 수는 클러스터를 완전히 사용합니다. 그러나 일부 태스크가 다른 태스크보다 오래 걸리므로 유휴 코어가 생성됩니다. 이 경우 병목 현상 태스크를 나누고 효율적으로 예약하려면 `numPartitions`에 2 또는 3을 곱하는 방법을 고려합니다.

파티션이 너무 많음

파티션 수가 너무 많으면 태스크 수가 너무 많습니다. 이로 인해 Spark 실행기간의 관리 태스크 및 데이터 교환과 같은 분산 처리와 관련된 오버헤드 때문에 Spark 드라이버에 과부하가 발생합니다.

작업의 파티션 수가 목표 파티션 수보다 훨씬 큰 경우 파티션 수를 줄이는 방법을 고려합니다. 다음 옵션을 사용하여 파티션을 줄일 수 있습니다.

- 파일 크기가 매우 작은 경우 AWS Glue [groupFiles](#)를 사용합니다. Apache Spark 태스크 시작으로 인한 과도한 병렬화를 줄여 각 파일을 처리할 수 있습니다.
- `coalesce(N)`를 사용하여 파티션을 병합합니다. 이는 비용이 저렴한 프로세스입니다. 파티션 수를 줄이면 `repartition(N)`에서 셔플을 수행하여 각 파티션에 균등하게 레코드의 양을 분산하므로 `coalesce(N)`가 `repartition(N)`보다 선호됩니다. 이 경우 비용과 관리 오버헤드가 증가합니다.
- Spark 3.x 적응형 쿼리 실행을 사용합니다. [Apache Spark 섹션의 주요 주제](#)에서 설명한 대로 적응형 쿼리 실행은 파티션 수를 자동으로 병합하는 함수를 제공합니다. 실행을 수행할 때까지 파티션 수를 알 수 없는 경우 이 접근 방식을 사용할 수 있습니다.

## JDBC에서 데이터 로드 병렬화

Spark RDD 파티션 수는 구성에 의해 결정됩니다. 기본적으로 SELECT 쿼리를 통해 전체 소스 데이터 세트를 스캔하기 위해 단일 태스크만 실행됩니다.

AWS Glue DynamicFrames과 Spark DataFrames 여러 작업에서 병렬화된 JDBC 데이터 로드를 지원합니다. 이는 `where` 조건자를 사용하여 하나의 SELECT 쿼리를 여러 쿼리로 분할하여 수행됩니다. JDBC에서 읽기를 병렬화하려면 다음 옵션을 구성합니다.

- For AWS Glue DynamicFrame, 설정 `hashfield`(또는 `hashexpression`) 및 `hashpartition`. 자세한 내용은 [JDBC 테이블을 병렬로 읽기](#)를 참조하세요.

```
connection_mysql8_options = {
  "url": "jdbc:mysql://XXXXXXXXXX.XXXXXXX.us-east-1.rds.amazonaws.com:3306/test",
  "dbtable": "medicare_tb",
  "user": "test",
  "password": "XXXXXXXXXX",
  "hashexpression": "id",
  "hashpartitions": "10"
}
datasource0 = glueContext.create_dynamic_frame.from_options(
  'mysql',
  connection_options=connection_mysql8_options,
  transformation_ctx= "datasource0"
```

)

- Spark DataFrame의 경우 numPartitions, partitionColumn, lowerBound, upperBound를 설정합니다. 자세한 내용은 [JDBC To Other Databases](#)를 참조하세요.

```
df = spark.read \
    .format("jdbc") \
    .option("url", "jdbc:mysql://XXXXXXXXXX.XXXXXXX.us-east-1.rds.amazonaws.com:3306/test") \
    .option("dbtable", "medicare_tb") \
    .option("user", "test") \
    .option("password", "XXXXXXXXXX") \
    .option("partitionColumn", "id") \
    .option("numPartitions", "10") \
    .option("lowerBound", "0") \
    .option("upperBound", "1141455") \
    .load()

df.write.format("json").save("s3://bucket_name/Tests/sparkjdbc/with_parallel/")
```

## ETL 커넥터를 사용할 경우 DynamoDB에서 데이터 로드 병렬화

Spark RDD 파티션 수는 dynamodb.splits 파라미터에 의해 결정됩니다. Amazon DynamoDB에서 읽기를 병렬화하려면 다음 옵션을 구성합니다.

- dynamodb.splits의 값을 늘립니다.
- for [Spark의 ETL에 대한 연결 유형 및 옵션에 설명된 공식에 따라 파라미터를 최적화 AWS Glue 합니다.](#)

## Kinesis Data Streams에서 데이터 로드 병렬화

Spark RDD 파티션 수는 소스 Amazon Kinesis Data Streams 데이터 스트림의 샤드 수에 의해 결정됩니다. 데이터 스트림에 샤드가 몇 개뿐인 경우 Spark 태스크는 몇 개뿐입니다. 이로 인해 다운스트림 프로세스에서 병렬화가 낮아질 수 있습니다. Kinesis Data Streams에서 읽기를 병렬화하려면 다음 옵션을 구성합니다.

- Kinesis Data Streams에서 데이터를 로드할 경우 더 많은 병렬화를 얻으려면 샤드 수를 늘립니다.
- 마이크로 배치의 로직이 충분히 복잡한 경우 불필요한 열을 삭제한 후 배치 시작 시 데이터를 재분할하는 방법을 고려합니다.

자세한 내용은 [AWS Glue 스트리밍 ETL 작업의 비용 및 성능을 최적화하는 모범 사례를 참조하세요.](#)

## 데이터 로드 후 태스크 병렬화

데이터 로드 후 태스크를 병렬화하려면 다음 옵션을 사용하여 RDD 파티션 수를 늘립니다.

- 특히 로드 자체를 병렬화할 수 없는 경우 초기 로드 직후 더 많은 수의 파티션을 생성하기 위해 데이터를 다시 분할합니다.

파티션 수를 지정하여 `DynamicFrame` 또는 `DataFrame`에서 `repartition()`을 직접 호출합니다. 일반적으로 사용 가능한 코어 수의 2~3배가 적합합니다.

그러나 분할된 테이블을 작성할 경우 파일이 크게 증가할 수 있습니다(각 파티션이 잠재적으로 각 테이블 파티션에 파일을 생성할 수 있음). 이를 방지하기 위해 열을 기준으로 `DataFrame`을 다시 분할할 수 있습니다. 이렇게 하면 쓰기 전에 데이터가 구성되도록 테이블 파티션 열이 사용됩니다. 테이블 파티션에서 작은 파일을 가져오지 않고도 더 많은 수의 파티션을 지정할 수 있습니다. 그러나 일부 파티션 값이 대부분의 데이터로 끝나고 작업 완료가 지연되는 데이터 스큐를 피하도록 주의하세요.

- 셔플이 있는 경우 `spark.sql.shuffle.partitions` 값을 늘립니다. 또한 셔플 작업 시 메모리 문제를 해결하는 데 도움이 될 수 있습니다.

2,001개가 넘는 셔플 파티션이 있는 경우 Spark는 압축된 메모리 형식을 사용합니다. 숫자가 이에 근사한 경우 보다 효율적인 표현을 얻기 위해 해당 한도를 초과하는 `spark.sql.shuffle.partitions` 값을 설정하려고 할 수 있습니다.

## 셔플 최적화

`join()` 및 `groupByKey()`와 같은 특정 작업에서는 Spark가 셔플을 수행해야 합니다. 셔플은 데이터를 재분산하기 위한 Spark의 메커니즘으로, 이를 통해 여러 RDD 파티션에서 다르게 데이터가 그룹화됩니다. 셔플 작업은 성능 병목 현상을 해결하는 데 도움이 될 수 있습니다. 그러나 셔플 작업에는 일반적으로 Spark 실행기 사이에서 데이터를 복사하는 작업이 포함되므로 셔플은 복잡하고 비용이 많이 드는 작업입니다. 예를 들어 셔플에서는 다음과 같은 비용이 발생합니다.

- 디스크 I/O:
  - 디스크에서 많은 수의 중간 파일을 생성합니다.
- 네트워크 I/O:
  - 많은 네트워크 연결이 필요합니다(연결 수 = Mapper × Reducer).

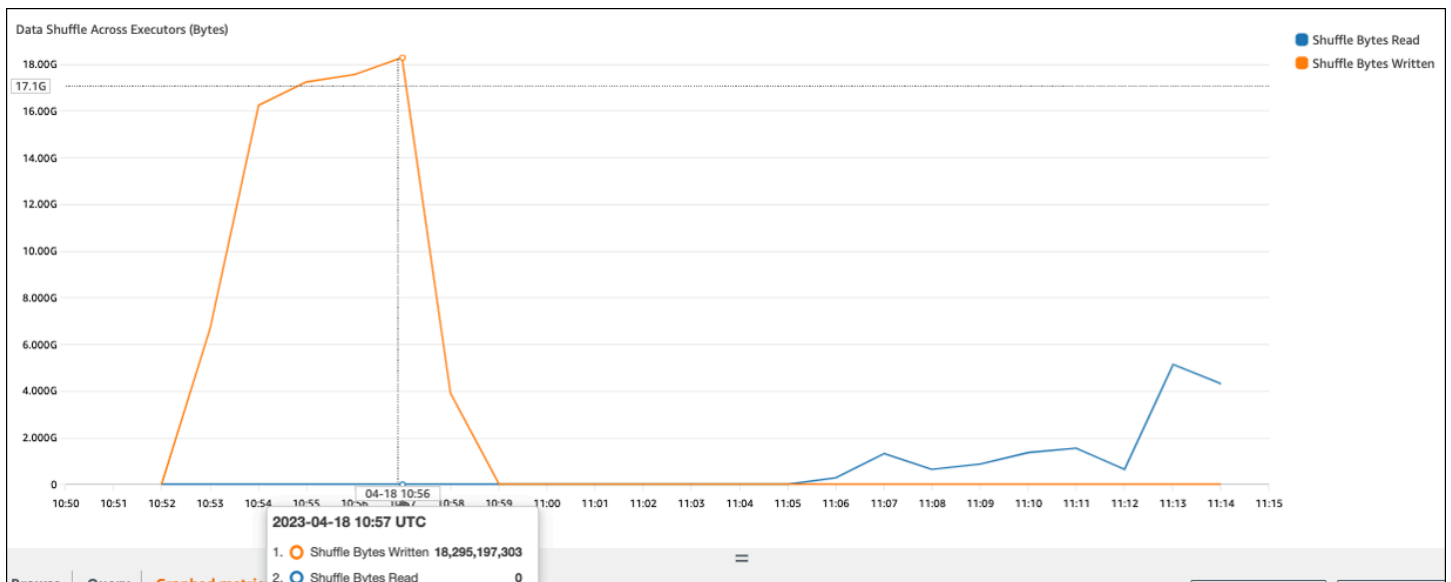
- 레코드는 다른 Spark 실행기에서 호스팅될 수 있는 새 RDD 파티션으로 집계되므로 데이터세트의 상당 부분이 네트워크를 통해 Spark 실행기 사이에서 이동할 수 있습니다.
- CPU 및 메모리 로드:
  - 값을 정렬하고 데이터세트를 병합합니다. 이러한 작업은 실행기에서 계획되어 실행기에 과도한 로드를 부과합니다.

셔플은 Spark 애플리케이션의 성능 저하에서 가장 중요한 요인 중 하나입니다. 중간 데이터를 저장하는 동안 실행기의 로컬 디스크에서 공간을 소진하여 Spark 작업이 실패할 수 있습니다.

CloudWatch 지표와 Spark UI에서 셔플 성능을 평가할 수 있습니다.

## CloudWatch 지표

셔플 바이트 쓰기 값이 셔플 바이트 읽기와 비교하여 높은 경우 Spark 작업은 `join()` 또는 `groupByKey()`와 같은 [셔플 작업](#)을 사용할 수 있습니다.



## Spark UI

Spark UI의 스테이지 탭에서 셔플 읽기 크기/레코드 값을 확인할 수 있습니다. 실행기 탭에서도 볼 수 있습니다.

다음 스크린샷에서 각 실행기는 약 18.6GB/4,020,000개 레코드를 셔플 프로세스와 교환합니다(총 셔플 읽기 크기는 약 75GB).

셔플 유출(디스크) 옆에는 디스크로의 많은 데이터 유출 메모리가 표시되며, 이로 인해 디스크가 가득 차거나 성능 문제가 발생할 수 있습니다.

- Aggregated Metrics by Executor				
Executor ID ▲	Address	Shuffle Read Size / Records	Shuffle Spill (Memory)	Shuffle Spill (Disk)
1	172.35.205.23:46731	18.6 GB / 40210300	98.1 GB	16.8 GB
2	172.35.195.173:46185	18.7 GB / 40246767	117.2 GB	17.3 GB
3	172.36.135.106:35913	18.6 GB / 40253921	101.6 GB	16.6 GB
4	172.34.131.223:46879	18.6 GB / 40190741	99.5 GB	16.4 GB

이러한 증상이 관찰되고 성능 목표에 비해 단계가 너무 오래 걸리거나 Out Of Memory 또는 No space left on device 오류로 인해 실패하는 경우 다음 솔루션을 고려합니다.

## 조인 최적화

테이블을 조인하는 `join()` 작업은 가장 일반적으로 사용되는 셔플 작업이지만 종종 성능 병목 현상이 되곤 합니다. 조인은 비용이 많이 드는 작업이므로 비즈니스 요구 사항에 필수적인 경우가 아니면 사용하지 않는 것이 좋습니다. 다음 질문을 통해 데이터 파이프라인을 효율적으로 사용하고 있는지 다시 확인합니다.

- 재사용할 수 있는 다른 작업에서도 수행되는 조인을 다시 계산하고 있나요?
- 출력의 소비자가 사용하지 않는 값으로 외래 키를 확인하기 위해 조인하고 있나요?

조인 작업이 비즈니스 요구 사항에 필수적임을 확인한 후라면 요구 사항을 충족하는 방식으로 조인을 최적화하기 위해 다음 옵션을 참조하세요.

### 조인 전에 푸시다운 사용

조인을 수행하기 전에 DataFrame에서 불필요한 행과 열을 필터링합니다. 이 경우 다음과 같은 이점이 있습니다.

- 셔플 중에 데이터 전송량 감소
- Spark 실행기에서 처리량 감소
- 데이터 스캔량 감소

```
# Default
df_joined = df1.join(df2, ["product_id"])
```

```
# Use Pushdown
df1_select =
  df1.select("product_id", "product_title", "star_rating").filter(col("star_rating")>=4.0)
df2_select = df2.select("product_id", "category_id")
df_joined = df1_select.join(df2_select, ["product_id"])
```

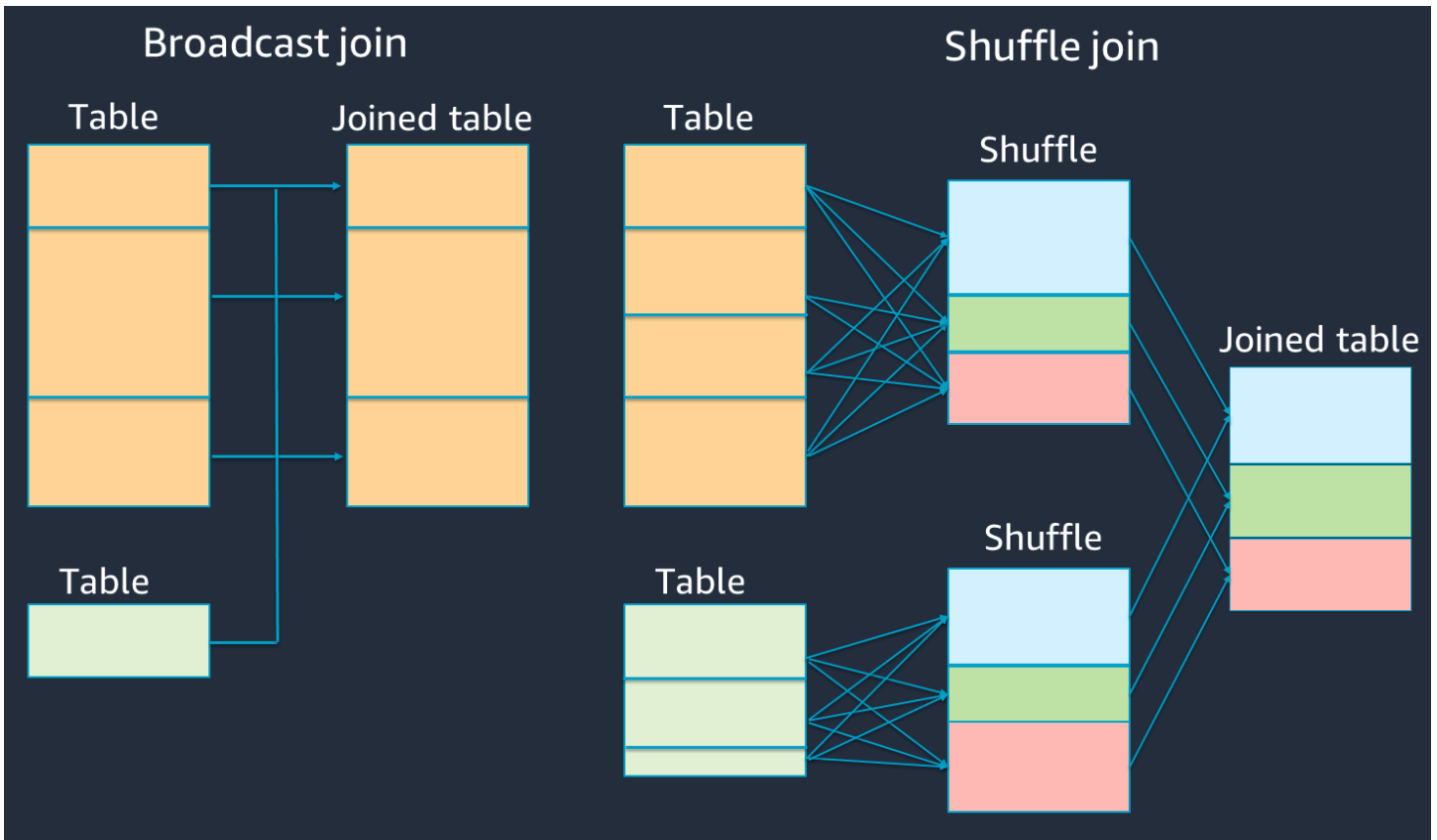
## DataFrame 조인 사용

RDD API 또는 DynamicFrame 조인 대신 SparkSQL, DataFrame 및 Datasets와 같은 [Spark 상위 수준 API](#)를 사용해 보세요. `dyf.toDF()`와 같은 메서드 직접 호출을 사용하여 DynamicFrame을 DataFrame으로 변환할 수 있습니다. [Apache Spark의 주요 주제](#) 섹션에서 설명한 대로 이러한 조인 작업은 내부적으로 Catalyst 옵티마이저의 쿼리 최적화를 활용합니다.

## 셔플 및 브로드캐스트 해시 조인 및 힌트

Spark는 셔플 조인과 브로드캐스트 해시 조인이라는 두 가지 유형의 조인을 지원합니다. 브로드캐스트 해시 조인은 셔플 작업이 필요하지 않으며 셔플 조인보다 필요한 처리가 적을 수 있습니다. 그러나 작은 테이블을 큰 테이블에 조인할 때만 적용됩니다. 단일 Spark 실행기의 메모리에 적합할 수 있는 테이블을 조인할 경우 브로드캐스트 해시 조인을 사용하는 방법을 고려합니다.

다음 다이어그램에서는 브로드캐스트 해시 조인 및 셔플 조인에 대한 상위 수준 구조와 단계를 보여줍니다.



각 조인의 세부 정보는 다음과 같습니다.

- 셔플 조인:
  - 셔플 해시 조인은 정렬 없이 두 테이블을 조인하고 두 테이블 간에 조인을 분산합니다. Spark 실행기의 메모리에 저장할 수 있는 작은 테이블의 조인에 적합합니다.
  - 정렬 병합 조인은 조인할 두 테이블을 키별로 분산하고 조인하기 전에 정렬합니다. 큰 테이블의 조인에 적합합니다.
- 브로드캐스트 해시 조인:
  - 브로드캐스트 해시 조인은 더 작은 RDD 또는 테이블을 각 워커 노드로 푸시합니다. 그런 다음 더 큰 RDD 또는 테이블의 각 파티션과 맵 측 결합을 수행합니다.

이 방법은 RDD 또는 테이블 중 하나가 메모리에 적합하거나 메모리에 적합하도록 만들 수 있는 조인에 적절합니다. 셔플이 필요하지 않으므로 가능하면 브로드캐스트 해시 조인을 수행하는 것이 좋습니다. 다음과 같이 조인 힌트를 사용하여 Spark에 브로드캐스트 조인을 요청할 수 있습니다.

```
# DataFrame
from pyspark.sql.functions import broadcast
```

```
df_joined= df_big.join(broadcast(df_small), right_df[key] == left_df[key],
    how='inner')

-- SparkSQL
SELECT /*+ BROADCAST(t1) / FROM t1 INNER JOIN t2 ON t1.key = t2.key;
```

조인 힌트에 대한 자세한 내용은 [Join hints](#)를 참조하세요.

In AWS Glue 3.0 이상에서는 [적응형 쿼리 실행](#) 및 추가 파라미터를 활성화하여 브로드캐스트 해시 조인을 자동으로 활용할 수 있습니다. 적응형 쿼리 실행은 조인 축의 런타임 통계가 적응형 브로드캐스트 해시 조인 임계치보다 작을 경우 정렬-병합 조인을 브로드캐스트 해시 조인으로 변환합니다.

In AWS Glue 3.0에서는 이를 설정하여 적응형 쿼리 실행을 활성화할 수 있습니다. `spark.sql.adaptive.enabled=true`. 적응형 쿼리 실행은 AWS Glue 4.0에서 기본적으로 활성화됩니다.

셔플 및 브로드캐스트 해시 조인과 관련된 추가 파라미터를 설정할 수 있습니다.

- `spark.sql.adaptive.localShuffleReader.enabled`
- `spark.sql.adaptive.autoBroadcastJoinThreshold`

관련 파라미터에 대한 자세한 내용은 [Converting sort-merge join to broadcast join](#)을 참조하세요.

In AWS Glue 3.0 이상에서는 셔플에 다른 조인 힌트를 사용하여 동작을 조정할 수 있습니다.

```
-- Join Hints for shuffle sort merge join
SELECT /*+ SHUFFLE_MERGE(t1) / FROM t1 INNER JOIN t2 ON t1.key = t2.key;
SELECT /*+ MERGEJOIN(t2) / FROM t1 INNER JOIN t2 ON t1.key = t2.key;
SELECT /*+ MERGE(t1) / FROM t1 INNER JOIN t2 ON t1.key = t2.key;

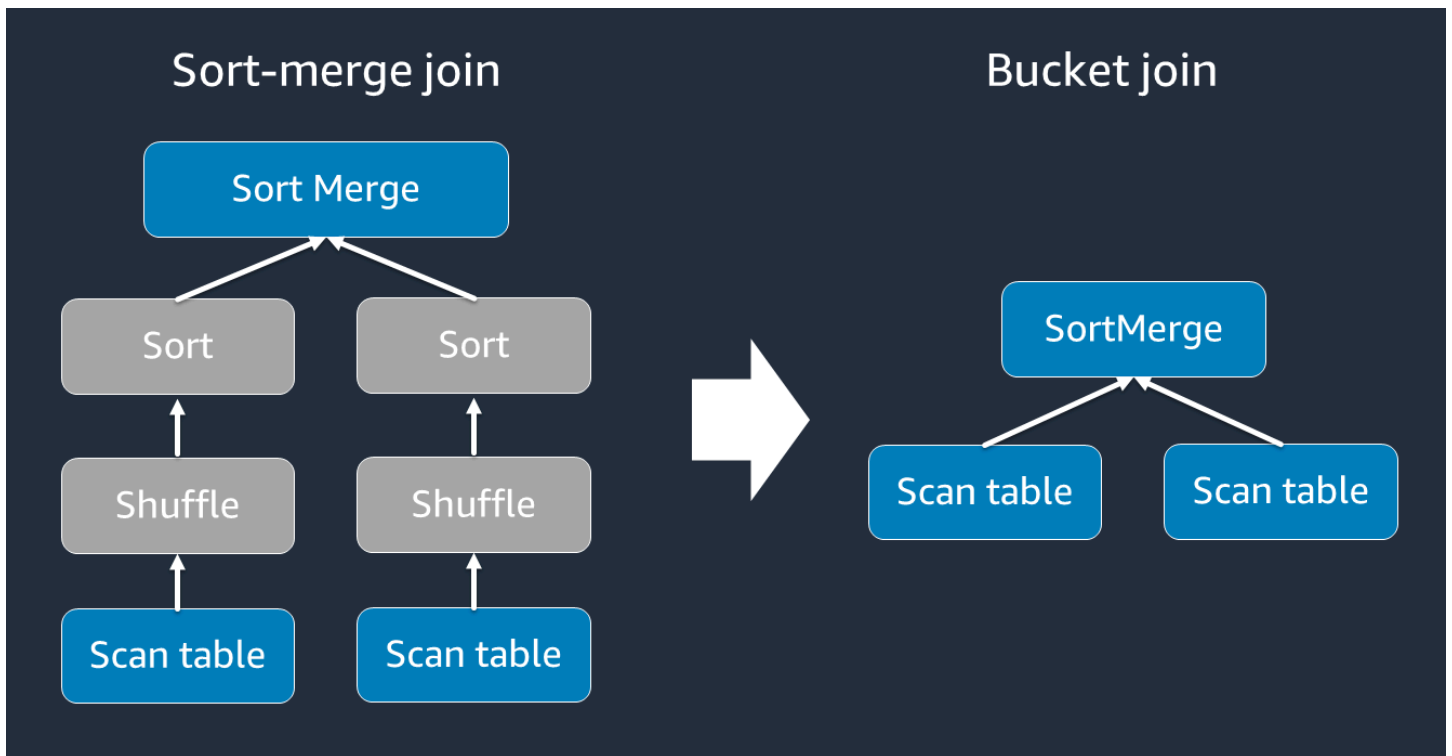
-- Join Hints for shuffle hash join
SELECT /*+ SHUFFLE_HASH(t1) / FROM t1 INNER JOIN t2 ON t1.key = t2.key;

-- Join Hints for shuffle-and-replicate nested loop join
SELECT /*+ SHUFFLE_REPLICATE_NL(t1) / FROM t1 INNER JOIN t2 ON t1.key = t2.key;
```

## 버킷팅 사용

정렬 병합 조인에는 셔플과 정렬 그리고 병합이라는 두 단계가 필요합니다. 이 두 단계로 인해 Spark 실행기에 과부하가 걸리고, 일부 실행기가 병합되고 다른 실행기가 동시에 정렬될 때 OOM 및 성능 문

제가 발생할 수 있습니다. 이러한 경우 **버킷팅**을 사용하여 효율적으로 조인할 수 있습니다. 버킷팅은 조인 키에 대한 입력을 사전 셔플 및 사전 정렬한 다음 정렬된 데이터를 중간 테이블에 씁니다. 정렬된 중간 테이블을 미리 정의하여 큰 테이블을 조인할 때 셔플 및 정렬 단계의 비용을 줄일 수 있습니다.



버킷 테이블은 다음과 같은 경우에 유용합니다.

- `account_id`와 같은 동일한 키를 통해 자주 조인되는 데이터
- 공통 열에서 버킷팅할 수 있는 기본 및 델타 테이블과 같은 일일 누적 테이블 로드

다음 코드를 사용하여 버킷팅된 테이블을 생성할 수 있습니다.

```
df.write.bucketBy(50, "account_id").sortBy("age").saveAsTable("bucketed_table")
```

조인 전 조인 키의 DataFrames 다시 분할

조인 전에 조인 키에서 두 DataFrames를 다시 분할하려면 다음 명령문을 사용합니다.

```
df1_repartitioned = df1.repartition(N,"join_key")
df2_repartitioned = df2.repartition(N,"join_key")
df_joined = df1_repartitioned.join(df2_repartitioned,"product_id")
```

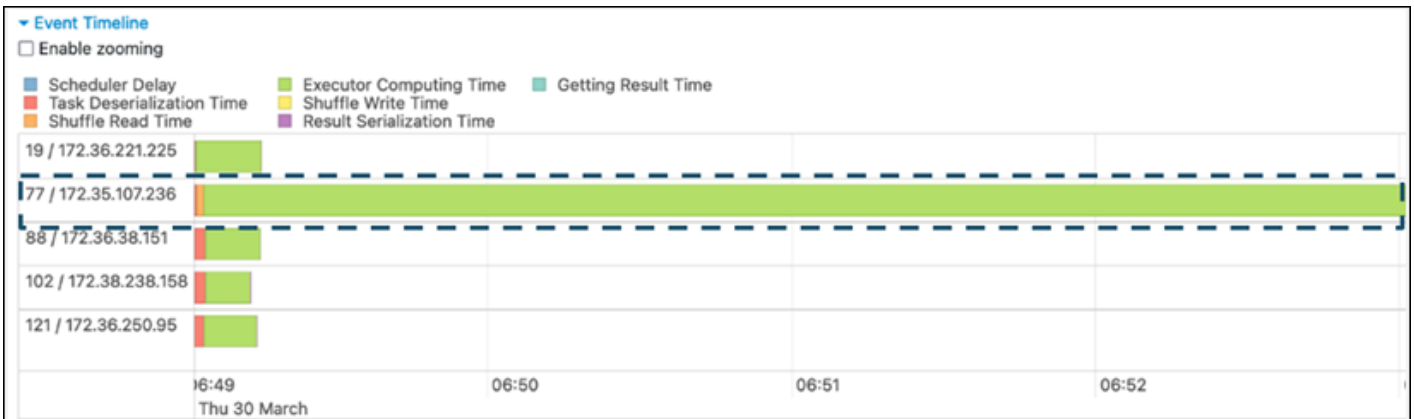
이렇게 하면 조인을 시작하기 전에 조인 키에서 두 개의 개별 RDD가 분할됩니다. 두 RDD가 동일한 분할 코드로 동일한 키에 분할되는 경우 조인에 대해 셔플을 수행하기 전에 함께 조인하려는 RDD 레코드가 동일한 작업자에 함께 존재할 가능성이 큼니다. 이렇게 하면 조인 중에 네트워크 활동 및 데이터 스큐를 줄여 성능이 향상될 수 있습니다.

## 데이터 스큐 재정의

데이터 스큐는 Spark 작업에서 가장 일반적인 병목 현상 원인 중 하나입니다. 데이터가 RDD 파티션에 균일하게 분산되지 않을 경우 나타납니다. 이 경우 해당 파티션에 대한 태스크가 다른 태스크보다 훨씬 오래 걸리므로 애플리케이션의 전체 처리 시간이 지연됩니다.

데이터 스큐를 식별하려면 Spark UI에서 다음 지표를 평가합니다.

- Spark UI의 단계 탭에서 이벤트 타임라인 페이지를 검토합니다. 다음 스크린샷에서는 고르지 않은 작업 분산을 볼 수 있습니다. 균등하지 않게 분산되거나 실행하는 데 너무 오래 걸리는 태스크는 데이터 스큐를 나타낼 수 있습니다.



- 또 다른 중요한 페이지는 Spark 태스크에 대한 통계를 보여주는 요약 지표입니다. 다음 스크린샷은 지속 시간, GC 시간, 유출(메모리), 유출(디스크) 등에 대한 백분위수를 포함한 지표를 보여줍니다.

Metric	Min	25th percentile	Median	75th percentile	Max
Duration	9 s	10 s	11 s	13 s	6.4 min
GC Time	0.0 ms	0.2 s	0.3 s	0.4 s	1 s
Spill (memory)	0.0 B	0.0 B	0.0 B	0.0 B	16.7 GiB
Spill (disk)	0.0 B	0.0 B	0.0 B	0.0 B	10.2 GiB
Output Size / Records	8.3 MiB / 12651	9.4 MiB / 21462	36.1 MiB / 63860	92.9 MiB / 258057	10.1 GiB / 20370130
Shuffle Read Size / Records	9.8 MiB / 12651	11.7 MiB / 21462	43.4 MiB / 63860	122.6 MiB / 258057	11.8 GiB / 20370130

태스크가 균등하게 분산되면 모든 백분위수에서 비슷한 숫자가 표시됩니다. 데이터 스큐가 있는 경우 각 백분위수에 편향된 값이 표시됩니다. 이 예제에서 태스크 지속 시간은 최소, 25번째 백분위수, 중앙값, 75번째 백분위수에서 13초 미만입니다. 최대 태스크는 75번째 백분위수보다 100배 더 많은

데이터를 처리했지만 해당 지속 시간은 6.4분으로 약 30배 더 길습니다. 즉, 하나 이상의 태스크(또는 태스크의 최대 25%)이 나머지 태스크보다 훨씬 오래 걸렸습니다.

데이터 스큐가 표시되면 다음을 시도하세요.

- AWS Glue 3.0을 사용하는 경우를 설정하여 적응형 쿼리 실행을 활성화합니다. `spark.sql.adaptive.enabled=true`. 적응형 쿼리 실행은 기본적으로 AWS Glue 4.0에서 활성화됩니다.

다음 관련 파라미터를 설정하여 조인이 도입한 데이터 스큐에 적응형 쿼리 실행을 사용할 수도 있습니다.

- `spark.sql.adaptive.skewJoin.skewedPartitionFactor`
- `spark.sql.adaptive.skewJoin.skewedPartitionThresholdInBytes`
- `spark.sql.adaptive.advisoryPartitionSizeInBytes=128m` (128 mebibytes or larger should be good)
- `spark.sql.adaptive.coalescePartitions.enabled=true` (when you want to coalesce partitions)

자세한 내용은 [Apache Spark 설명서](#)를 참조하세요.

- 조인 키에 대해 큰 범위의 값이 있는 키를 사용합니다. 셔플 조인에서 키의 각 해시 값에 대해 파티션이 결정됩니다. 조인 키의 카디널리티가 너무 낮으면 해시 함수가 여러 파티션에서 데이터를 분산하는 잘못된 작업을 수행할 가능성이 더 큽니다. 따라서 애플리케이션 및 비즈니스 로직이 이를 지원하는 경우 더 높은 카디널리티 키 또는 복합 키를 사용하는 방법을 고려합니다.

```
# Use Single Primary Key
df_joined = df1_select.join(df2_select, ["primary_key"])

# Use Composite Key
df_joined = df1_select.join(df2_select, ["primary_key", "secondary_key"])
```

## 캐시 사용

반복 DataFrames 사용하는 경우 `df.cache()` 또는 `df.persist()`를 사용하여 각 Spark 실행기의 메모리 및 디스크에 계산 결과를 캐싱함으로써 추가 셔플 또는 계산을 피합니다. Spark는 디스크에 RDD를 유지하거나 여러 노드([스토리지 수준](#))에서 복제하는 작업도 지원합니다.

예를 들어 `df.persist()`를 추가하여 DataFrames를 유지할 수 있습니다. 캐시가 더 이상 필요하지 않은 경우 `unpersist`를 사용하여 캐싱된 데이터를 삭제할 수 있습니다.

```
df = spark.read.parquet("s3://<Bucket>/parquet/product_category=Books/")
df_high_rate = df.filter(col("star_rating")>=4.0)
df_high_rate.persist()

df_joined1 = df_high_rate.join(<Table1>, ["key"])
df_joined2 = df_high_rate.join(<Table2>, ["key"])
df_joined3 = df_high_rate.join(<Table3>, ["key"])
...
df_high_rate.unpersist()
```

## 불필요한 Spark 작업 제거

`count`, `show` 또는 `collect`와 같은 불필요한 작업을 실행하지 마세요. [Apache Spark 섹션의 주요 주제](#)에서 설명한 대로 Spark는 지연됩니다. 변환된 각 RDD는 작업을 실행할 때마다 다시 계산될 수 있습니다. Spark 작업을 많이 사용하면 각 작업에 대해 여러 소스 액세스, 태스크 계산 및 셔플 실행이 직접 호출됩니다.

상용 환경에서 `collect()` 또는 기타 작업이 필요하지 않은 경우 제거하는 것을 고려합니다.

### Note

상용 환경에서는 Spark `collect()`를 최대한 사용하지 마세요. `collect()` 작업은 Spark 실행기의 모든 계산 결과를 Spark 드라이버에 반환하므로 이로 인해 Spark 드라이버가 OOM 오류를 반환할 수 있습니다. OOM 오류를 방지하기 위해 Spark는 기본적으로 `spark.driver.maxResultSize = 1GB`를 설정합니다. 그러면 Spark 드라이버에 반환되는 최대 데이터 크기를 1GB로 제한합니다.

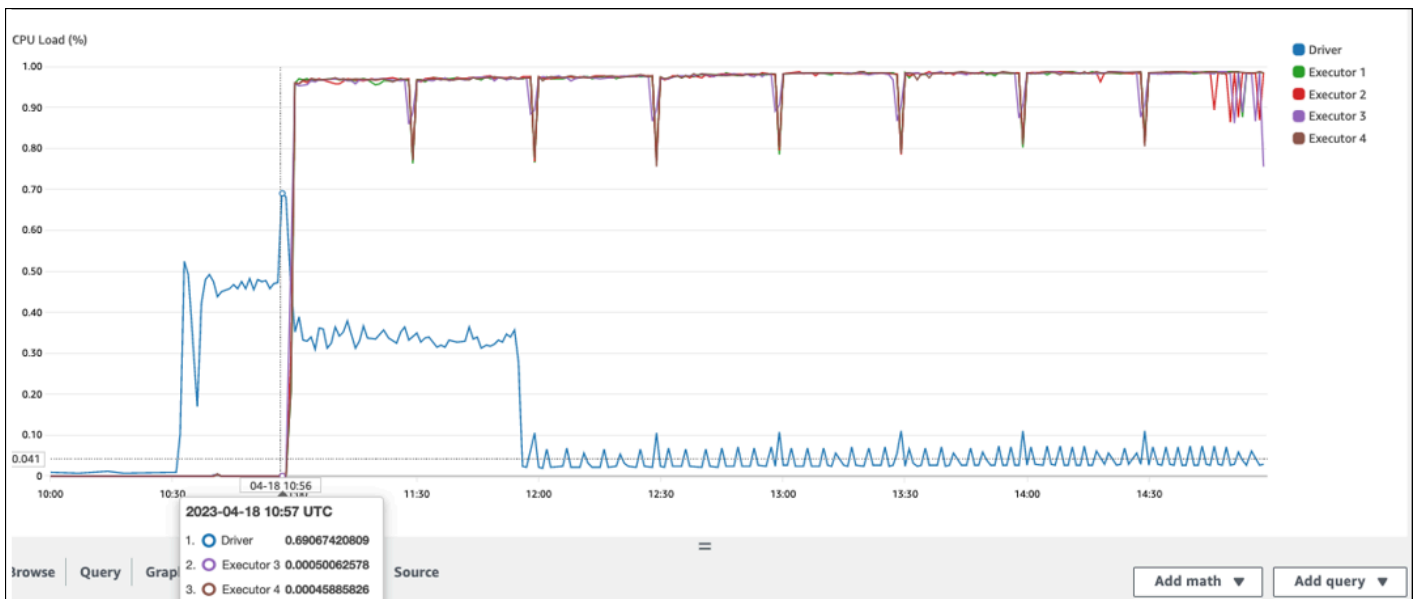
## 계획 오버헤드 최소화

[Apache Spark의 주요 주제](#)에서 설명한 대로 Spark 드라이버는 실행 계획을 생성합니다. 이 계획에 따라 분산 처리를 위해 Spark 실행기에 태스크가 할당됩니다. 그러나 작은 파일이 많거나 AWS Glue Data Catalog에 파티션이 많은 경우 Spark 드라이버가 병목 현상의 원인이 될 수 있습니다. 높은 계획 오버헤드를 식별하려면 다음 지표를 평가합니다.

## CloudWatch 지표

CPU 로드 및 메모리 사용률에서 다음 상황을 확인합니다.

- Spark 드라이버 CPU 로드 및 메모리 사용률은 높음으로 기록됩니다. 일반적으로 Spark 드라이버는 데이터를 처리하지 않으므로 CPU 로드 및 메모리 사용률이 급증하지 않습니다. 그러나 Amazon S3 데이터 소스에 작은 파일이 너무 많은 경우 모든 S3 객체를 나열하고 많은 수의 작업을 관리하면 리소스 사용률이 높아질 수 있습니다.
- Spark 실행기에서 처리가 시작되기 전에 긴 간격이 나타납니다. 다음 예제 스크린샷에서는 AWS Glue 작업이 10:00에 시작되더라도 Spark 실행기의 CPU 로드가 10:57까지 너무 낮습니다. 이는 Spark 드라이버가 실행 계획을 생성하는 데 시간이 오래 걸릴 수 있음을 나타냅니다. 이 예제에서는 Data Catalog에서 많은 파티션을 검색하고 Spark 드라이버에서 많은 작은 파일을 나열하는 데 시간이 오래 걸립니다.



## Spark UI

Spark UI의 작업 탭에서 제출 시간을 볼 수 있습니다. 다음 예제에서 Spark 드라이버는 작업이 10:00:00에 시작되었더라도 10:56:46에 AWS Glue job0을 시작했습니다.

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	count at DynamicFrame.scala:1414 count at DynamicFrame.scala:1414	2023/04/18 10:56:46	4.9 h	1/1	58100/58100

작업 탭에서 태스크(모든 단계): 성공/총계 시간을 볼 수도 있습니다. 이 경우 작업 수는 58100으로 기록됩니다. [태스크 병렬화](#) 페이지의 Amazon S3 섹션에서 설명한 대로 태스크 수는 대략적으로 S3 객체 수에 해당합니다. 즉, Amazon S3에는 약 58,100개의 객체가 있습니다.

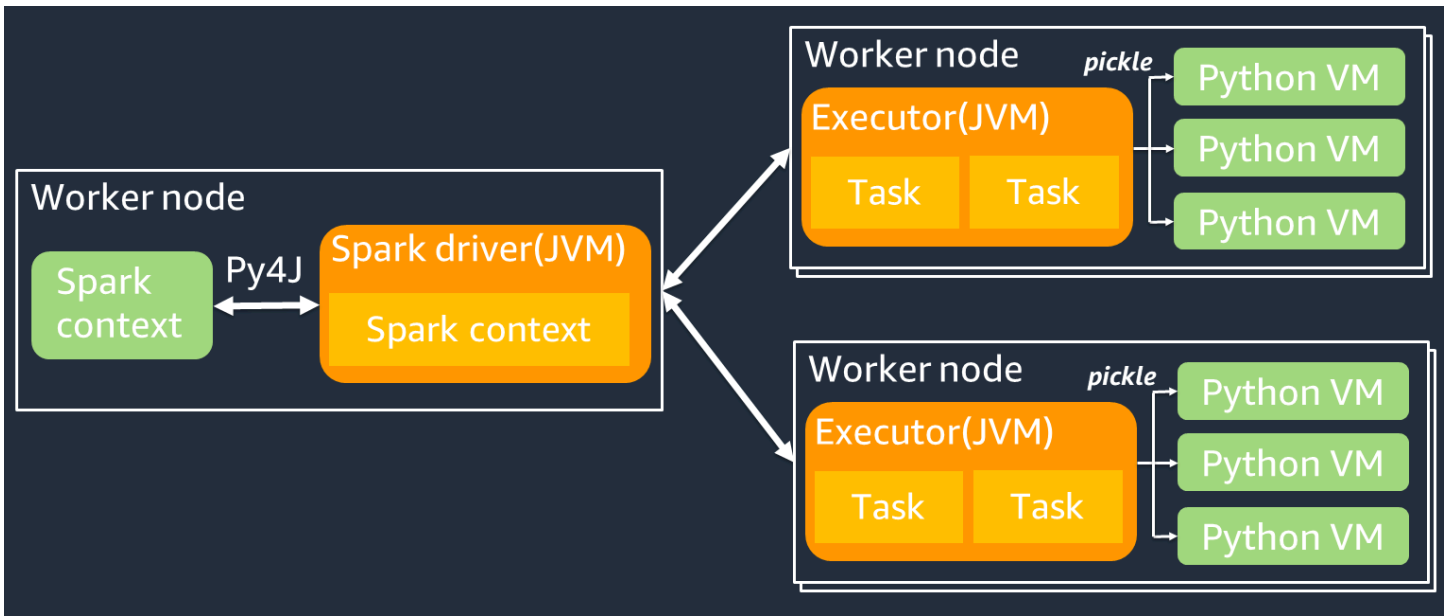
이 작업 및 타임라인에 대한 자세한 내용은 스테이지 탭을 검토하세요. Spark 드라이버에서 병목 현상이 관찰되면 다음 솔루션을 고려합니다.

- Amazon S3에 파일이 너무 많으면 [작업 병렬화](#) 페이지의 파티션이 너무 많음 섹션에 있는 과도한 병렬화에 대한 지침을 고려합니다.
- Amazon S3에 파티션이 너무 많으면 [데이터 스캔 양 감소](#) 페이지의 Amazon S3 파티션이 너무 많음 섹션에 있는 과도한 분할에 대한 지침을 고려합니다. Data Catalog에서 파티션 메타데이터를 검색하는 데 걸리는 지연 시간을 줄이기 위해 파티션이 많은 경우 [AWS Glue 파티션 인덱스](#)를 활성화합니다. 자세한 내용은 [AWS Glue 파티션 인덱스를 사용하여 쿼리 성능 개선을 참조하세요](#).
- JDBC에 파티션이 너무 많으면 hashpartition 값을 낮춥니다.
- DynamoDB에 파티션이 너무 많으면 dynamodb.splits 값을 낮춥니다.
- 스트리밍 작업에 파티션이 너무 많으면 샤드 수를 줄입니다.

## 사용자 정의 함수 최적화

PySpark의 사용자 정의 함수(UDF) 및 RDD.map은 종종 성능을 크게 저하시킬 수 있습니다. Spark의 기본 Scala 구현에서 Python 코드를 정확하게 나타내는 데 필요한 오버헤드 때문입니다.

다음 다이어그램은 의 아키텍처입니다. PySpark를 사용하는 경우 Spark 드라이버는 Py4j 라이브러리를 사용하여 Python에서 Java 메서드를 직접 호출합니다. Spark SQL 또는 DataFrame 기본 제공 함수를 직접 호출할 때 함수는 최적화된 실행 계획을 사용하여 각 실행기의 JVM에서 실행되므로 Python과 Scala 간에 성능 차이가 거의 없습니다.



map/ mapPartitions/ udf 사용 등 자체 Python 로직을 사용하는 경우 태스크는 Python 런타임 환경에서 실행됩니다. 두 환경을 관리하면 오버헤드 비용이 발생합니다. 또한 JVM 런타임 환경의 기본 제공 함수에서 사용할 수 있도록 메모리의 데이터를 변환해야 합니다. Pickle은 JVM과 Python 런타임 간 교환에 기본적으로 사용되는 직렬화 형식입니다. 그러나 이 직렬화 및 역직렬화 비용은 매우 높으므로 Java 또는 Scala로 작성된 UDF는 Python UDF보다 더 빠릅니다.

PySpark에서 직렬화 및 역직렬화 오버헤드를 방지하려면 다음을 고려합니다.

- 기본 제공 Spark SQL 함수 사용 - 자체 UDF 또는 맵 함수를 Spark SQL 또는 DataFrame 기본 제공 함수로 대체하는 방법을 고려합니다. Spark SQL 또는 DataFrame 기본 제공 함수를 실행할 때 태스크는 각 실행기의 JVM에서 처리되므로 Python과 Scala 간에 성능 차이가 거의 없습니다.
- Scala 또는 Java에서 UDF 구현 - JVM에서 실행되므로 Java 또는 Scala로 작성된 UDF를 사용하는 방법을 고려합니다.
- 벡터화된 워크로드에 Apache Arrow 기반 UDF 사용 - Arrow 기반 UDF를 사용하는 방법을 고려합니다. 이 기능을 벡터화된 UDF(Pandas UDF)라고도 합니다. [Apache Arrow](#)는 JVM과 Python 프로세스 간에 데이터를 효율적으로 전송하는 데 사용할 수 있는 언어에 구애받지 않는 인 메모리 데이터 형식입니다. 이는 현재 Pandas 또는 NumPy 데이터로 작업하는 Python 사용자에게 가장 유용합니다.

Arrow는 열 형식(벡터화된 형식)입니다. 이 기능의 사용은 자동이 아니며 최대한 활용하고 호환성을 보장하기 위해 구성 또는 코드를 약간 변경해야 할 수 있습니다. 자세한 내용과 제한 사항은 [Apache Arrow in PySpark](#)를 참조하세요.

다음 예제에서는 표준 Python, 벡터화된 UDF 및 Spark SQL에서 기본 증분 UDF를 비교합니다.

## 표준 Python UDF

예제 시간은 3.20(초)입니다.

예제 코드

```
# DataSet
df = spark.range(10000000).selectExpr("id AS a","id AS b")

# UDF Example
def plus(a,b):
    return a+b
spark.udf.register("plus",plus)

df.selectExpr("count(plus(a,b))").collect()
```

실행 계획

```
== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[], functions=[count/pythonUDF0#124])
+- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#580]
+- HashAggregate(keys=[], functions=[partial_count/pythonUDF0#124])
+- Project [pythonUDF0#124]
+- BatchEvalPython [plus(a#116L, b#117L)], [pythonUDF0#124]
+- Project [id#114L AS a#116L, id#114L AS b#117L]
+- Range (0, 10000000, step=1, splits=16)
```

## 벡터화된 UDF

예제 시간은 0.59(초)입니다.

벡터화된 UDF는 이전 UDF 예제보다 5배 빠릅니다. Physical Plan을 확인하면 ArrowEvalPython를 볼 수 있으며, 이는 이 애플리케이션이 Apache Arrow에 의해 벡터화되었음을 의미합니다. 벡터화된 UDF를 활성화하려면 코드에서 `spark.sql.execution.arrow.pyspark.enabled = true`를 지정해야 합니다.

예제 코드

```
# Vectorized UDF
from pyspark.sql.types import LongType
```

```

from pyspark.sql.functions import count, pandas_udf

# Enable Apache Arrow Support
spark.conf.set("spark.sql.execution.arrow.pyspark.enabled", "true")

# DataSet
df = spark.range(10000000).selectExpr("id AS a", "id AS b")

# Annotate pandas_udf to use Vectorized UDF
@pandas_udf(LongType())
def pandas_plus(a,b):
    return a+b
spark.udf.register("pandas_plus", pandas_plus)

df.selectExpr("count(pandas_plus(a,b))").collect()

```

## 실행 계획

```

== Physical Plan ==
AdaptiveSparkPlan isFinalPlan=false
+- HashAggregate(keys=[], functions=[count(pythonUDF0#1082L)],
  output=[count(pandas_plus(a, b))#1080L])
+- Exchange SinglePartition, ENSURE_REQUIREMENTS, [id=#5985]
+- HashAggregate(keys=[], functions=[partial_count(pythonUDF0#1082L)],
  output=[count#1084L])
+- Project [pythonUDF0#1082L]
+- ArrowEvalPython [pandas_plus(a#1074L, b#1075L)], [pythonUDF0#1082L], 200
+- Project [id#1072L AS a#1074L, id#1072L AS b#1075L]
+- Range (0, 10000000, step=1, splits=16)

```

## Spark SQL

예제 시간은 0.087(초)입니다.

Python 런타임 없이 각 실행기의 JVM에서 태스크가 실행되므로 Spark SQL은 벡터화된 UDF보다 훨씬 빠릅니다. UDF를 기본 제공 함수로 교체할 수 있는 경우 교체하는 것이 좋습니다.

### 예제 코드

```

df.createOrReplaceTempView("test")
spark.sql("select count(a+b) from test").collect()

```

## 빅 데이터에 panda 사용

이미 [pandas](#)에 익숙하고 빅 데이터에 Spark를 사용하려는 경우 Spark. AWS Glue 4.0 이상에서 pandas API를 사용할 수 있습니다. 시작하기 위해 공식 노트북 [Quickstart: Pandas API on Spark](#)를 사용할 수 있습니다. 자세한 내용은 [PySpark 설명서](#)를 참조하세요.

## 리소스

- [AWS Glue](#)
- [Performance Tuning\(Spark SQL 가이드\)](#)
- [AWS Glue Optimization Workshop](#)

## 문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 [RSS 피드](#)를 구독하십시오.

변경 사항	설명	날짜
<a href="#">최초 게시</a>	—	2024년 1월 2일

# AWS 권장 가이드 용어집

다음은 AWS 권장 가이드에서 제공하는 전략, 가이드 및 패턴에서 일반적으로 사용되는 용어입니다. 용어집 항목을 제안하려면 용어집 끝에 있는 피드백 제공 링크를 사용하십시오.

## 숫자

### 7가지 전략

애플리케이션을 클라우드로 이전하기 위한 7가지 일반적인 마이그레이션 전략 이러한 전략은 Gartner가 2011년에 파악한 5가지 전략을 기반으로 하며 다음으로 구성됩니다.

- 리팩터링/리아키텍트 - 클라우드 네이티브 기능을 최대한 활용하여 애플리케이션을 이동하고 해당 아키텍처를 수정함으로써 민첩성, 성능 및 확장성을 개선합니다. 여기에는 일반적으로 운영 체제와 데이터베이스 이식이 포함됩니다. 예: 온프레미스 Oracle 데이터베이스를 Amazon Aurora PostgreSQL 호환 에디션으로 마이그레이션합니다.
- 리플랫폼(리프트 앤드 리세이프) - 애플리케이션을 클라우드로 이동하고 일정 수준의 최적화를 도입하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 AWS 클라우드의 Amazon Relational Database Service(Amazon RDS) for Oracle로 마이그레이션합니다.
- 재구매(드롭 앤드 슝) - 일반적으로 기존 라이선스에서 SaaS 모델로 전환하여 다른 제품으로 전환합니다. 예: 고객 관계 관리(CRM) 시스템을 Salesforce.com으로 마이그레이션합니다.
- 리호스팅(리프트 앤드 시프트) - 애플리케이션을 변경하지 않고 클라우드로 이동하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 AWS 클라우드클라우드의 EC2 인스턴스에 있는 Oracle로 마이그레이션합니다.
- 재배포(하이퍼바이저 수준의 리프트 앤 시프트) - 새 하드웨어를 구매하거나, 애플리케이션을 다시 작성하거나, 기존 운영을 수정하지 않고도 인프라를 클라우드로 이동합니다. 온프레미스 플랫폼에서 동일한 플랫폼의 클라우드 서비스로 서버를 마이그레이션합니다. 예: Microsoft Hyper-V 애플리케이션을 로 마이그레이션합니다 AWS.
- 유지(보관) - 소스 환경에 애플리케이션을 유지합니다. 대규모 리팩터링이 필요하고 해당 작업을 나중에 연기하려는 애플리케이션과 비즈니스 차원에서 마이그레이션할 이유가 없어 유지하려는 레거시 애플리케이션이 여기에 포함될 수 있습니다.
- 사용 중지 - 소스 환경에서 더 이상 필요하지 않은 애플리케이션을 폐기하거나 제거합니다.

## A

### A2A(Agent-to-Agent)

작업 위임 및 상태 전송 agent-to-agent 공동 작업을 위한 상태 저장 프로토콜입니다.

### ABAC

[속성 기반 액세스 제어](#)를 참조하세요.

### 추상화된 서비스

[관리형 서비스](#)를 참조하세요.

### ACID

[원자성, 일관성, 격리성, 내구성](#)을 참조하세요.

### 능동-능동 마이그레이션

양방향 복제 도구 또는 이중 쓰기 작업을 사용하여 소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되고, 두 데이터베이스 모두 마이그레이션 중 연결 애플리케이션의 트랜잭션을 처리하는 데이터베이스 마이그레이션 방법입니다. 이 방법은 일회성 전환이 필요한 대신 소규모의 제어된 배치로 마이그레이션을 지원합니다. 더 유연하지만 [액티브 패시브 마이그레이션](#)보다 더 많은 작업이 필요합니다.

### 능동-수동 마이그레이션

소스 데이터베이스와 대상 데이터베이스가 동기화된 상태로 유지되지만 소스 데이터베이스만 연결 애플리케이션의 트랜잭션을 처리하고 데이터는 대상 데이터베이스로 복제되는 데이터베이스 마이그레이션 방법입니다. 대상 데이터베이스는 마이그레이션 중 어떤 트랜잭션도 허용하지 않습니다.

### 에이전트

목표를 달성하기 위한 도구를 사용하여 자율적으로 추론, 계획 및 조치를 취할 수 있는 AI 시스템입니다.

### 에이전트 운영

대규모 프로덕션 환경에서 AI 에이전트를 구축, 테스트, 배포 및 실행하기 위한 운영 사례입니다.

### 집계 함수

행 그룹에서 작동하고 그룹에 대한 단일 반환 값을 계산하는 SQL 함수입니다. 집계 함수의 예로 SUM 및 MAX가 있습니다.

## AI

[인공 지능](#)을 참조하세요.

## AIOps

[인공 지능 운영](#)을 참조하세요.

## 익명화

데이터세트에서 개인 정보를 영구적으로 삭제하는 프로세스입니다. 익명화는 개인 정보 보호에 도움이 될 수 있습니다. 익명화된 데이터는 더 이상 개인 데이터로 간주되지 않습니다.

## 안티 패턴

솔루션이 다른 솔루션보다 비생산적이거나 비효율적이거나 덜 효과적이어서 반복되는 문제에 자주 사용되는 솔루션입니다.

## 애플리케이션 제어

맬웨어로부터 시스템을 보호하기 위해 승인된 애플리케이션만 사용하도록 허용하는 보안 접근 방식입니다.

## 애플리케이션 포트폴리오

애플리케이션 구축 및 유지 관리 비용과 애플리케이션의 비즈니스 가치를 비롯하여 조직에서 사용하는 각 애플리케이션에 대한 세부 정보 모음입니다. 이 정보는 [포트폴리오 탐색 및 분석 프로세스](#)의 핵심이며 마이그레이션, 현대화 및 최적화할 애플리케이션을 식별하고 우선순위를 정하는 데 도움이 됩니다.

## 인공 지능

컴퓨터 기술을 사용하여 학습, 문제 해결, 패턴 인식 등 일반적으로 인간과 관련된 인지 기능을 수행하는 것을 전문으로 하는 컴퓨터 과학 분야입니다. 자세한 내용은 [What is Artificial Intelligence?](#)를 참조하십시오.

## 인공 지능 운영(AIOps)

기계 학습 기법을 사용하여 운영 문제를 해결하고, 운영 인시던트 및 사용자 개입을 줄이고, 서비스 품질을 높이는 프로세스입니다. AWS 마이그레이션 전략에서 AIOps가 사용되는 방법에 대한 자세한 내용은 [운영 통합 가이드](#)를 참조하십시오.

## 비대칭 암호화

한 쌍의 키, 즉 암호화를 위한 퍼블릭 키와 복호화를 위한 프라이빗 키를 사용하는 암호화 알고리즘입니다. 퍼블릭 키는 복호화에 사용되지 않으므로 공유할 수 있지만 프라이빗 키에 대한 액세스는 엄격히 제한되어야 합니다.

## 원자성, 일관성, 격리성, 내구성(ACID)

오류, 정전 또는 기타 문제가 발생한 경우에도 데이터베이스의 데이터 유효성과 운영 신뢰성을 보장하는 소프트웨어 속성 세트입니다.

## ABAC(속성 기반 액세스 제어)

부서, 직무, 팀 이름 등의 사용자 속성을 기반으로 세분화된 권한을 생성하는 방식입니다. 자세한 내용은 AWS Identity and Access Management (IAM) 설명서의 [용 ABAC AWS](#)를 참조하세요.

## 신뢰할 수 있는 데이터 소스

가장 신뢰할 수 있는 정보 소스로 간주되는 기본 버전의 데이터를 저장하는 위치입니다. 익명화, 편집 또는 가명화와 같은 데이터 처리 또는 수정의 목적으로 신뢰할 수 있는 데이터 소스의 데이터를 다른 위치로 복사할 수 있습니다.

## 가용 영역

다른 가용 영역의 장애로부터 격리 AWS 리전 되고 동일한 리전의 다른 가용 영역에 저렴하고 지연 시간이 짧은 네트워크 연결을 제공하는 내의 고유한 위치입니다.

## AWS 클라우드 채택 프레임워크(AWS CAF)

조직이 클라우드로 성공적으로 전환 AWS 하기 위한 효율적이고 효과적인 계획을 개발하는 데 도움이 되는 지침 및 모범 사례 프레임워크입니다. AWS CAF는 지침을 비즈니스, 사람, 거버넌스, 플랫폼, 보안 및 운영이라는 6가지 중점 영역으로 구성합니다. 비즈니스, 사람 및 거버넌스 관점은 비즈니스 기술과 프로세스에 초점을 맞추고, 플랫폼, 보안 및 운영 관점은 전문 기술과 프로세스에 중점을 둡니다. 예를 들어, 사람 관점은 인사(HR), 직원 배치 기능 및 인력 관리를 담당하는 이해관계자를 대상으로 합니다. 이러한 관점에서 AWS CAF는 성공적인 클라우드 채택을 위해 조직을 준비하는 데 도움이 되는 인력 개발, 교육 및 커뮤니케이션에 대한 지침을 제공합니다. 자세한 내용은 [AWS CAF 웹사이트](#)와 [AWS CAF 백서](#)를 참조하세요.

## AWS 워크로드 검증 프레임워크(AWS WQF)

데이터베이스 마이그레이션 워크로드를 평가하고, 마이그레이션 전략을 권장하고, 작업 견적을 제공하는 도구입니다. AWS WQF는 AWS Schema Conversion Tool (AWS SCT)에 포함되어 있습니다. 데이터베이스 스키마 및 코드 객체, 애플리케이션 코드, 종속성 및 성능 특성을 분석하고 평가 보고서를 제공합니다.

## B

### 악성 봇

개인 또는 조직을 방해하거나 해를 입히기 위한 [봇](#)입니다.

### BCP

[비즈니스 연속성 계획](#)을 참조하세요.

### 동작 그래프

리소스 동작과 시간 경과에 따른 상호 작용에 대한 통합된 대화형 뷰입니다. Amazon Detective에서 동작 그래프를 사용하여 실패한 로그인 시도, 의심스러운 API 직접 호출 및 유사한 작업을 검사할 수 있습니다. 자세한 내용은 Detective 설명서의 [Data in a behavior graph](#)를 참조하십시오.

### 빅 엔디안 시스템

가장 중요한 바이트를 먼저 저장하는 시스템입니다. [엔디안](#)도 참조하세요.

### 바이너리 분류

바이너리 결과(가능한 두 클래스 중 하나)를 예측하는 프로세스입니다. 예를 들어, ML 모델이 “이 이메일이 스팸인가요, 스팸이 아닌가요?”, ‘이 제품은 책임가요, 자동차인가요?’ 등의 문제를 예측해야 할 수 있습니다.

### 블룸 필터

요소가 세트의 멤버인지 여부를 테스트하는 데 사용되는 메모리 효율성이 높은 확률론적 데이터 구조입니다.

### 블루/그린(Blue/Green) 배포

동일하지만 별개의 두 환경을 생성하는 배포 전략입니다. 하나의 환경(파란색)에서 현재 애플리케이션 버전을 실행하고 새 애플리케이션 버전은 다른 환경(녹색)에서 실행합니다. 이 전략을 사용하면 영향을 최소화하면서 신속하게 롤백할 수 있습니다.

### bot

인터넷을 통해 자동화된 태스크를 실행하고 인적 활동이나 상호 작용을 시뮬레이션하는 소프트웨어 애플리케이션입니다. 인터넷에서 정보를 인덱싱하는 웹 크롤러와 같이 유용하거나 이로운 봇도 있습니다. 악성 봇이라고 하는 다른 일부 봇은 개인 또는 조직을 방해하거나 해를 입히기 위한 봇입니다.

## 봇넷

[맬웨어](#)에 감염되고 봇 허더 또는 봇 운영자와 같은 단일 당사자가 제어하는 [봇](#) 네트워크입니다. 봇넷은 봇의 규모와 봇의 영향 범위를 확대하는 가장 잘 알려진 메커니즘입니다.

## 브랜치

코드 리포지토리의 포함된 영역입니다. 리포지토리에 생성되는 첫 번째 브랜치가 기본 브랜치입니다. 기존 브랜치에서 새 브랜치를 생성한 다음 새 브랜치에서 기능을 개발하거나 버그를 수정할 수 있습니다. 기능을 구축하기 위해 생성하는 브랜치를 일반적으로 기능 브랜치라고 합니다. 기능을 출시할 준비가 되면 기능 브랜치를 기본 브랜치에 다시 병합합니다. 자세한 내용은 [About branches](#)(GitHub 설명서)를 참조하십시오.

## 긴급 액세스 권한

예외적인 상황에서 승인된 프로세스를 통해 사용자가 일반적으로 액세스할 권한이 없는데 액세스할 수 있는 빠른 방법입니다. 자세한 내용은 AWS Well-Architected 지침의 [Implement break-glass procedures](#) 지표를 참조하세요.

## 브라운필드 전략

사용자 환경의 기존 인프라 시스템 아키텍처에 브라운필드 전략을 채택할 때는 현재 시스템 및 인프라의 제약 조건을 중심으로 아키텍처를 설계합니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 [그린필드](#) 전략을 혼합할 수 있습니다.

## 버퍼 캐시

가장 자주 액세스하는 데이터가 저장되는 메모리 영역입니다.

## 사업 역량

기업이 가치를 창출하기 위해 하는 일(예: 영업, 고객 서비스 또는 마케팅)입니다. 마이크로서비스 아키텍처 및 개발 결정은 비즈니스 역량에 따라 이루어질 수 있습니다. 자세한 내용은 백서의 [AWS에서 컨테이너화된 마이크로서비스 실행의 비즈니스 역량 중심의 구성화](#) 섹션을 참조하십시오.

## 비즈니스 연속성 계획(BCP)

대규모 마이그레이션과 같은 중단 이벤트가 운영에 미치는 잠재적 영향을 해결하고 비즈니스가 신속하게 운영을 재개할 수 있도록 지원하는 계획입니다.

# C

## CAF

[AWS Cloud Adoption Framework](#)를 참조하세요.

## 카나리 배포

최종 사용자에게 제공하는 느린 증분 릴리스 버전입니다. 확신이 들면 새 버전을 배포하고 현재 버전을 완전히 교체합니다.

## CCoE

[클라우드 혁신 센터](#)를 참조하세요.

## CDC

[데이터 캡처 변경](#)을 참조하세요.

### 변경 데이터 캡처(CDC)

데이터베이스 테이블과 같은 데이터 소스의 변경 내용을 추적하고 변경 사항에 대한 메타데이터를 기록하는 프로세스입니다. 대상 시스템의 변경 내용을 감사하거나 복제하여 동기화를 유지하는 등의 다양한 용도로 CDC를 사용할 수 있습니다.

## 카오스 엔지니어링

시스템의 복원력을 테스트하기 위해 의도적으로 장애나 중단 이벤트를 도입합니다. [AWS Fault Injection Service \(AWS FIS\)](#)를 사용하여 AWS 워크로드에 스트레스를 주고 응답을 평가하는 실험을 수행할 수 있습니다.

## CI/CD

[지속적 통합 및 지속적 전송](#)을 참조하세요.

## 분류

예측을 생성하는 데 도움이 되는 분류 프로세스입니다. 분류 문제에 대한 ML 모델은 이산 값을 예측합니다. 이산 값은 항상 서로 다릅니다. 예를 들어, 모델이 이미지에 자동차가 있는지 여부를 평가해야 할 수 있습니다.

## 시민 개발자

전문 기술 없이 노코드/로우코드 플랫폼을 사용하여 AI 애플리케이션을 생성하는 비즈니스 사용자입니다.

## 클라이언트측 암호화

대상이 데이터를 AWS 서비스 수신하기 전에 로컬에서 데이터를 암호화합니다.

## 클라우드 혁신 센터(CCoE)

클라우드 모범 사례 개발, 리소스 동원, 마이그레이션 타임라인 설정, 대규모 혁신을 통한 조직 선도 등 조직 전체에서 클라우드 채택 노력을 추진하는 다분야 팀입니다. 자세한 내용은 AWS 클라우드 엔터프라이즈 전략 블로그의 [CCoE 게시물](#)을 참조하세요.

## 클라우드 컴퓨팅

원격 데이터 스토리지와 IoT 디바이스 관리에 일반적으로 사용되는 클라우드 기술 클라우드 컴퓨팅은 일반적으로 [엣지 컴퓨팅](#) 기술에 연결되어 있습니다.

## 클라우드 운영 모델

IT 조직에서 하나 이상의 클라우드 환경을 구축, 성숙화 및 최적화하는 데 사용되는 운영 모델입니다. 자세한 내용은 [클라우드 운영 모델 구축](#)을 참조하십시오.

## 클라우드 채택 단계

조직이 AWS 클라우드로 마이그레이션할 때 일반적으로 거치는 4단계는 다음과 같습니다.

- 프로젝트 - 개념 증명 및 학습 목적으로 몇 가지 클라우드 관련 프로젝트 실행
- 기반 - 클라우드 채택 확장을 위한 기초 투자(예: 랜딩 존 생성, CCoE 정의, 운영 모델 구축)
- 마이그레이션 - 개별 애플리케이션 마이그레이션
- Re-invention - 제품 및 서비스 최적화와 클라우드 혁신

이러한 단계는 Stephen Orban이 블로그 게시물 [The Journey Toward Cloud-First and the Stages of Adoption](#) on the AWS 클라우드 Enterprise Strategy 블로그에서 정의했습니다. AWS 마이그레이션 전략과 어떤 관련이 있는지에 대한 자세한 내용은 [마이그레이션 준비 가이드](#)를 참조하세요.

## CMDB

[구성 관리 데이터베이스](#)를 참조하세요.

## 코드 리포지토리

소스 코드와 설명서, 샘플, 스크립트 등의 기타 자산이 버전 관리 프로세스를 통해 저장되고 업데이트되는 위치입니다. 일반적인 클라우드 리포지토리로 GitHub 또는 Bitbucket Cloud가 포함됩니다. 코드의 각 버전을 브랜치라고 합니다. 마이크로서비스 구조에서 각 리포지토리는 단일 기능 전용입니다. 단일 CI/CD 파이프라인은 여러 리포지토리를 사용할 수 있습니다.

## 콜드 캐시

비어 있거나, 제대로 채워지지 않았거나, 오래되었거나 관련 없는 데이터를 포함하는 버퍼 캐시입니다. 주 메모리나 디스크에서 데이터베이스 인스턴스를 읽어야 하기 때문에 성능에 영향을 미치며, 이는 버퍼 캐시에서 읽는 것보다 느립니다.

## 콜드 데이터

거의 액세스되지 않고 일반적으로 과거 데이터인 데이터. 이런 종류의 데이터를 쿼리할 때는 일반적으로 느린 쿼리가 허용됩니다. 이 데이터를 성능이 낮고 비용이 저렴한 스토리지 계층 또는 클래스로 옮기면 비용을 절감할 수 있습니다.

## 컴퓨터 비전(CV)

기계 학습을 사용하여 디지털 이미지 및 비디오와 같은 시각적 형식에서 정보를 분석하고 추출하는 [AI](#) 필드입니다. 예를 들어 Amazon SageMaker AI는 CV에 대한 이미지 처리 알고리즘을 제공합니다.

## 구성 드리프트

워크로드의 경우 구성이 예상되는 상태에서 변경됩니다. 이로 인해 워크로드가 규정을 준수하지 않을 수 있으며, 이는 일반적으로 점진적이고 의도되지 않은 작업입니다.

## 구성 관리 데이터베이스(CMDB)

하드웨어 및 소프트웨어 구성 요소와 해당 구성을 포함하여 데이터베이스와 해당 IT 환경에 대한 정보를 저장하고 관리하는 리포지토리입니다. 일반적으로 마이그레이션의 포트폴리오 탐색 및 분석 단계에서 CMDB의 데이터를 사용합니다.

## 규정 준수 팩

규정 준수 및 보안 검사를 사용자 지정하기 위해 조합할 수 있는 AWS Config 규칙 및 수정 작업의 모음입니다. YAML 템플릿을 사용하여 적합성 팩을 AWS 계정 및 리전 또는 조직 전체에 단일 엔티티로 배포할 수 있습니다. 자세한 내용은 AWS Config 설명서의 [적합성 팩](#)을 참조하세요.

## 지속적 통합 및 지속적 전달(CI/CD)

소프트웨어 릴리스 프로세스의 소스, 빌드, 테스트, 스테이징 및 프로덕션 단계를 자동화하는 프로세스입니다. CI/CD는 일반적으로 파이프라인으로 설명됩니다. CI/CD를 통해 프로세스를 자동화하고, 생산성을 높이고, 코드 품질을 개선하고, 더 빠르게 제공할 수 있습니다. 자세한 내용은 [지속적 전달의 이점](#)을 참조하십시오. CD는 지속적 배포를 의미하기도 합니다. 자세한 내용은 [지속적 전달\(Continuous Delivery\)과 지속적인 개발](#)을 참조하십시오.

## CV

[컴퓨터 비전](#)을 참조하세요.

## D

### 저장 데이터

스토리지에 있는 데이터와 같이 네트워크에 고정되어 있는 데이터입니다.

### 데이터 분류

중요도와 민감도를 기준으로 네트워크의 데이터를 식별하고 분류하는 프로세스입니다. 이 프로세스는 데이터에 대한 적절한 보호 및 보존 제어를 결정하는 데 도움이 되므로 사이버 보안 위험 관리 전략의 중요한 구성 요소입니다. 데이터 분류는 AWS Well-Architected Framework에서 보안 원칙의 구성 요소입니다. 자세한 내용은 [데이터 분류](#)를 참조하십시오.

### 데이터 드리프트

프로덕션 데이터와 ML 모델 학습에 사용된 데이터 간의 상당한 차이 또는 시간 경과에 따른 입력 데이터의 의미 있는 변화. 데이터 드리프트는 ML 모델 예측의 전반적인 품질, 정확성 및 공정성을 저하시킬 수 있습니다.

### 전송 중 데이터

네트워크를 통과하고 있는 데이터입니다. 네트워크 리소스 사이를 이동 중인 데이터를 예로 들 수 있습니다.

### 데이터 메시

중앙 집중식 관리 및 거버넌스를 통해 분산되고 탈중앙화된 데이터 소유권을 제공하는 아키텍처 프레임워크입니다.

### 데이터 최소화

꼭 필요한 데이터만 수집하고 처리하는 원칙입니다. 에서 데이터를 최소화하면 개인 정보 보호 위험, 비용 및 분석 탄소 발자국을 줄일 수 있습니다.

### 데이터 경계

신뢰할 수 있는 자격 증명만 예상 네트워크에서 신뢰할 수 있는 리소스에 액세스하도록 하는 데 도움이 되는 AWS 환경의 예방 가드레일 세트입니다. 자세한 내용은 [데이터 경계 구축을 참조하세요 AWS](#).

### 데이터 사전 처리

원시 데이터를 ML 모델이 쉽게 구문 분석할 수 있는 형식으로 변환하는 것입니다. 데이터를 사전 처리한다는 것은 특정 열이나 행을 제거하고 누락된 값, 일관성이 없는 값 또는 중복 값을 처리함을 의미할 수 있습니다.

## 데이터 출처

라이프사이클 전반에 걸쳐 데이터의 출처와 기록을 추적하는 프로세스(예: 데이터 생성, 전송, 저장 방법).

## 데이터 주체

데이터를 수집 및 처리하는 개인입니다.

## 데이터 웨어하우스

분석과 같은 비즈니스 인텔리전스를 지원하는 데이터 관리 시스템입니다. 데이터 웨어하우스에는 보통 많은 양의 기록 데이터가 포함되며 일반적으로 쿼리 및 분석에 사용됩니다.

## 데이터 정의 언어(DDL)

데이터베이스에서 테이블 및 객체의 구조를 만들거나 수정하기 위한 명령문 또는 명령입니다.

## 데이터베이스 조작 언어(DML)

데이터베이스에서 정보를 수정(삽입, 업데이트 및 삭제)하기 위한 명령문 또는 명령입니다.

## DDL

[데이터 정의 언어](#)를 참조하세요.

## 딥 앙상블

예측을 위해 여러 딥 러닝 모델을 결합하는 것입니다. 딥 앙상블을 사용하여 더 정확한 예측을 얻거나 예측의 불확실성을 추정할 수 있습니다.

## 딥 러닝

여러 계층의 인공 신경망을 사용하여 입력 데이터와 관심 대상 변수 간의 매핑을 식별하는 ML 하위 분야입니다.

## 심층 방어

네트워크와 그 안의 데이터 기밀성, 무결성 및 가용성을 보호하기 위해 컴퓨터 네트워크 전체에 일련의 보안 메커니즘과 제어를 신중하게 계층화하는 정보 보안 접근 방식입니다. 이 전략을 채택하면 AWS Organizations 구조의 여러 계층에 여러 컨트롤을 AWS 추가하여 리소스를 보호할 수 있습니다. 예를 들어, 심층 방어 접근 방식은 다단계 인증, 네트워크 세분화 및 암호화를 결합할 수 있습니다.

## 위임된 관리자

에서 AWS Organizations 호환되는 서비스는 AWS 멤버 계정을 등록하여 조직의 계정을 관리하고 해당 서비스에 대한 권한을 관리할 수 있습니다. 이러한 계정을 해당 서비스의 위임된 관리자라고

합니다. 자세한 내용과 호환되는 서비스 목록은 AWS Organizations 설명서의 [AWS Organizations](#)와 함께 사용할 수 있는 AWS 서비스를 참조하십시오.

## 배포

대상 환경에서 애플리케이션, 새 기능 또는 코드 수정 사항을 사용할 수 있도록 하는 프로세스입니다. 배포에는 코드 베이스의 변경 사항을 구현한 다음 애플리케이션 환경에서 해당 코드베이스를 구축하고 실행하는 작업이 포함됩니다.

## 개발 환경

[환경](#)을 참조하세요.

## 탐지 제어

이벤트 발생 후 탐지, 기록 및 알림을 수행하도록 설계된 보안 제어입니다. 이러한 제어는 기존의 예방적 제어를 우회한 보안 이벤트를 알리는 2차 방어선입니다. 자세한 내용은 AWS에서 보안 제어 구현의 [탐지 제어](#)를 참조하세요.

## 개발 가치 흐름 매핑 (DVSM)

소프트웨어 개발 라이프사이클에서 속도와 품질에 부정적인 영향을 미치는 제약 조건을 식별하고 우선 순위를 지정하는 데 사용되는 프로세스입니다. DVSM은 원래 린 제조 방식을 위해 설계된 가치 흐름 매핑 프로세스를 확장합니다. 소프트웨어 개발 프로세스를 통해 가치를 창출하고 이동하는 데 필요한 단계와 팀에 중점을 둡니다.

## 디지털 트윈

건물, 공장, 산업 장비 또는 생산 라인과 같은 실제 시스템을 가상으로 표현한 것입니다. 디지털 트윈은 예측 유지 보수, 원격 모니터링, 생산 최적화를 지원합니다.

## 차원 테이블

[스타 스키마](#)에서 팩트 테이블의 정량적 데이터에 대한 데이터 속성을 포함하는 더 작은 테이블을 말합니다. 차원 테이블 속성은 일반적으로 텍스트 필드나 텍스트처럼 동작하는 개별 숫자입니다. 이러한 속성은 보통 쿼리 제약, 필터링 및 결과 세트 레이블 지정에 사용됩니다.

## 재해

워크로드 또는 시스템이 기본 배포 위치에서 비즈니스 목표를 달성하지 못하게 방해하는 이벤트입니다. 이러한 이벤트는 자연재해, 기술적 오류, 의도하지 않은 구성 오류 또는 멀웨어 공격과 같은 사람의 행동으로 인한 결과일 수 있습니다.

## 재해 복구(DR)

[재해](#)로 인한 가동 중지 시간 및 데이터 손실을 최소화하기 위해 사용하는 전략 및 프로세스입니다. 자세한 내용은 AWS Well-Architected Framework의 [Disaster Recovery of Workloads on AWS: Recovery in the Cloud](#)를 참조하세요.

## DML

[데이터베이스 조작 언어](#)를 참조하세요.

## 도메인 기반 설계

구성 요소를 각 구성 요소가 제공하는 진화하는 도메인 또는 핵심 비즈니스 목표에 연결하여 복잡한 소프트웨어 시스템을 개발하는 접근 방식입니다. 이 개념은 에릭 에반스에 의해 그의 저서인 도메인 기반 디자인: 소프트웨어 중심의 복잡성 해결(Boston: Addison-Wesley Professional, 2003)에서 소개되었습니다. Strangler Fig 패턴과 함께 도메인 기반 설계를 사용하는 방법에 대한 자세한 내용은 [컨테이너 및 Amazon API Gateway를 사용하여 기존의 Microsoft ASP.NET\(ASMX\) 웹 서비스를 점진적으로 현대화하는 방법](#)을 참조하십시오.

## DR

[재해 복구](#)를 참조하세요.

## 드리프트 감지

기준이 되는 구성과의 편차 추적을 말합니다. 예를 들어 AWS CloudFormation 를 사용하여 [시스템 리소스의 드리프트를 감지](#)하거나 사용하여 AWS Control Tower 거버넌스 요구 사항 준수에 영향을 미칠 수 있는 [랜딩 존의 변경 사항을 감지](#)할 수 있습니다.

## DVSM

[개발 가치 흐름 매핑](#)을 참조하세요.

## E

### EDA

[탐색 데이터 분석](#)을 참조하세요.

### EDI

[전자 데이터 교환](#)을 참조하세요.

## 엣지 컴퓨팅

IoT 네트워크의 엣지에서 스마트 디바이스의 컴퓨팅 성능을 개선하는 기술 엣지 컴퓨팅은 [클라우드 컴퓨팅](#)에 비해 보다 통신 지연 시간을 줄이고 응답 시간을 개선할 수 있습니다.

## 전자 데이터 교환(EDI)

조직 간 비즈니스 문서의 자동화된 교환을 나타냅니다. 자세한 내용은 [전자 데이터 교환\(EDI\)이란 무엇인가요?](#)를 참조하세요.

## 암호화

사람이 읽을 수 있는 일반 텍스트 데이터를 사이퍼텍스트로 변환하는 컴퓨팅 프로세스입니다.

## 암호화 키

암호화 알고리즘에 의해 생성되는 무작위 비트의 암호화 문자열입니다. 키의 길이는 다양할 수 있으며 각 키는 예측할 수 없고 고유하게 설계되었습니다.

## 엔디안

컴퓨터 메모리에 바이트가 저장되는 순서입니다. 빅 엔디안 시스템은 가장 중요한 바이트를 먼저 저장합니다. 리틀 엔디안 시스템은 가장 덜 중요한 바이트를 먼저 저장합니다.

## 엔드포인트

[서비스 엔드포인트](#)를 참조하세요.

## 엔드포인트 서비스

Virtual Private Cloud(VPC)에서 호스팅하여 다른 사용자와 공유할 수 있는 서비스입니다. 를 사용하여 엔드포인트 서비스를 생성하고 다른 AWS 계정 또는 AWS Identity and Access Management (IAM) 보안 주체에 권한을 AWS PrivateLink 부여할 수 있습니다. 이러한 계정 또는 보안 주체는 인터페이스 VPC 엔드포인트를 생성하여 엔드포인트 서비스에 비공개로 연결할 수 있습니다. 자세한 내용은 Amazon Virtual Private Cloud(VPC) 설명서의 [엔드포인트 서비스 생성](#)을 참조하십시오.

## 엔터프라이즈 리소스 계획(ERP)

엔터프라이즈의 주요 비즈니스 프로세스(예: 회계, [MES](#), 프로젝트 관리)를 자동화하고 관리하는 시스템입니다.

## 봉투 암호화

암호화 키를 다른 암호화 키로 암호화하는 프로세스입니다. 자세한 내용은 AWS Key Management Service (AWS KMS) 설명서의 [봉투 암호화](#)를 참조하세요.

## 환경

실행 중인 애플리케이션의 인스턴스입니다. 다음은 클라우드 컴퓨팅의 일반적인 환경 유형입니다.

- 개발 환경 - 애플리케이션 유지 관리를 담당하는 핵심 팀만 사용할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. 개발 환경은 변경 사항을 상위 환경으로 승격하기 전에 테스트하는 데 사용됩니다. 이러한 유형의 환경을 테스트 환경이라고도 합니다.
- 하위 환경 - 초기 빌드 및 테스트에 사용되는 환경을 비롯한 애플리케이션의 모든 개발 환경입니다.
- 프로덕션 환경 - 최종 사용자가 액세스할 수 있는 실행 중인 애플리케이션의 인스턴스입니다. CI/CD 파이프라인에서 프로덕션 환경이 마지막 배포 환경입니다.
- 상위 환경 - 핵심 개발 팀 이외의 사용자가 액세스할 수 있는 모든 환경입니다. 프로덕션 환경, 프로덕션 이전 환경 및 사용자 수용 테스트를 위한 환경이 여기에 포함될 수 있습니다.

## 에픽

애자일 방법론에서 작업을 구성하고 우선순위를 정하는 데 도움이 되는 기능적 범주입니다. 에픽은 요구 사항 및 구현 작업에 대한 개괄적인 설명을 제공합니다. 예를 들어, AWS CAF 보안 에픽에는 ID 및 액세스 관리, 탐지 제어, 인프라 보안, 데이터 보호 및 인시던트 대응이 포함됩니다. AWS 마 이그레이션 전략의 에픽에 대한 자세한 내용은 [프로그램 구현 가이드](#)를 참조하십시오.

## ERP

[엔터프라이즈 리소스 계획](#)을 참조하세요.

### 탐색 데이터 분석(EDA)

데이터 세트를 분석하여 주요 특성을 파악하는 프로세스입니다. 데이터를 수집 또는 집계한 다음 초기 조사를 수행하여 패턴을 찾고, 이상을 탐지하고, 가정을 확인합니다. EDA는 요약 통계를 계산하고 데이터 시각화를 생성하여 수행됩니다.

## F

### 팩트 테이블

[스타 스키마](#)의 중앙 테이블입니다. 비즈니스 운영에 대한 정량적 데이터를 저장합니다. 일반적으로 팩트 테이블은 측정값이 있는 열 및 차원 테이블에 대한 외래 키가 있는 열과 같이 두 가지 열 유형을 포함합니다.

## 빠른 실패

개발 수명 주기를 줄이기 위해 빈번한 증분 테스트를 사용하는 철학입니다. 애자일 접근 방식의 핵심입니다.

## 장애 격리 경계

에서 장애의 영향을 제한하고 워크로드의 복원력을 개선하는 데 도움이 되는 가용 영역, AWS 리전 컨트롤 플레인 또는 데이터 플레인과 같은 AWS 클라우드경계입니다. 자세한 내용은 [AWS 장애 격리 경계](#)를 참조하세요.

## 기능 브랜치

[브랜치](#)를 참조하세요.

## 기능

예측에 사용하는 입력 데이터입니다. 예를 들어, 제조 환경에서 기능은 제조 라인에서 주기적으로 캡처되는 이미지일 수 있습니다.

## 기능 중요도

모델의 예측에 특성이 얼마나 중요한지를 나타냅니다. 이는 일반적으로 SHAP(Shapley Additive Descriptions) 및 통합 그래디언트와 같은 다양한 기법을 통해 계산할 수 있는 수치 점수로 표현됩니다. 자세한 내용은 [기계 학습 모델 해석 가능성을 참조하세요 AWS](#).

## 기능 변환

추가 소스로 데이터를 보강하거나, 값을 조정하거나, 단일 데이터 필드에서 여러 정보 세트를 추출하는 등 ML 프로세스를 위해 데이터를 최적화하는 것입니다. 이를 통해 ML 모델이 데이터를 활용할 수 있습니다. 예를 들어, 날짜 '2021-05-27 00:15:37'을 '2021년', '5월', '목', '15일'로 분류하면 학습 알고리즘이 다양한 데이터 구성 요소와 관련된 미묘한 패턴을 학습하는 데 도움이 됩니다.

## 퓨샷 프롬프팅

유사한 태스크를 수행하도록 요청하기 전에 [LLM](#)에 태스크와 원하는 출력을 보여주는 몇 가지 예제를 제공합니다. 이 기법은 모델이 프롬프트에 포함된 예제(샷)에서 학습하는 컨텍스트 내 학습을 적용합니다. 퓨샷 프롬프팅은 특정 형식 지정, 추론 또는 분야별 지식이 필요한 태스크에 효과적일 수 있습니다. [제로샷 프롬프팅](#)도 참조하세요.

## FGAC

[세분화된 액세스 제어](#)를 참조하세요.

## 세분화된 액세스 제어(FGAC)

여러 조건을 사용하여 액세스 요청을 허용하거나 거부합니다.

## 플래시컷 마이그레이션

단계적 접근 방식을 사용하는 대신 [변경 데이터 캡처](#)를 통해 지속적 데이터 복제를 사용하여 최단 시간에 데이터를 마이그레이션하는 데이터베이스 마이그레이션 방법입니다. 목표는 가동 중지 시간을 최소화하는 것입니다.

### FM

[파운데이션 모델](#)을 참조하세요.

### 파운데이션 모델(FM)

일반화되고 레이블이 지정되지 않은 데이터의 대규모 데이터세트에서 훈련된 대규모 딥 러닝 신경망입니다. FM은 언어 이해, 텍스트 및 이미지 생성, 자연어 대화와 같은 다양한 일반 태스크를 수행할 수 있습니다. 자세한 내용은 [파운데이션 모델이란?](#)을 참조하세요.

### FM 게이트웨이

[파운데이션 모델에](#) 대한 액세스를 제어하고 정규화하는 중앙 집중식 중개자입니다. LLM 게이트웨이이라고도 합니다.

## G

### 생성형 AI

대량의 데이터에서 훈련되었으며 간단한 텍스트 프롬프트를 사용하여 이미지, 비디오, 텍스트, 오디오와 같은 새 콘텐츠와 아티팩트를 생성할 수 있는 [AI](#) 모델의 하위 세트입니다. 자세한 내용은 [생성형 AI란 무엇인가요?](#)를 참조하세요.

### 지리적 차단

[지리적 제한](#)을 참조하세요.

### 지리적 제한(지리적 차단)

Amazon CloudFront에서 특정 국가의 사용자가 콘텐츠 배포에 액세스하지 못하도록 하는 옵션입니다. 허용 목록 또는 차단 목록을 사용하여 승인된 국가와 차단된 국가를 지정할 수 있습니다. 자세한 내용은 CloudFront 설명서의 [콘텐츠의 지리적 배포 제한](#)을 참조하십시오.

### Gitflow 워크플로

하위 환경과 상위 환경이 소스 코드 리포지토리의 서로 다른 브랜치를 사용하는 방식입니다. Gitflow 워크플로는 레거시로 간주되며 [트렁크 기반 워크플로](#)는 선호되는 현대적 접근 방식입니다.

## 골든 이미지

시스템 또는 소프트웨어의 새 인스턴스를 배포하기 위한 템플릿으로 사용되는 해당 시스템 또는 소프트웨어의 스냅샷입니다. 예를 들어 제조 분야에서는 골든 이미지를 사용하여 여러 디바이스에서 소프트웨어를 프로비저닝할 수 있으며 이를 통해 딥이스 제조 작업의 속도, 확장성 및 생산성을 개선할 수 있습니다.

## 브라운필드 전략

새로운 환경에서 기존 인프라의 부재 시스템 아키텍처에 대한 그린필드 전략을 채택할 때 [브라운필드](#)라고도 하는 기존 인프라와의 호환성 제한 없이 모든 새로운 기술을 선택할 수 있습니다. 기존 인프라를 확장하는 경우 브라운필드 전략과 그린필드 전략을 혼합할 수 있습니다.

## 가드레일

조직 단위(OU) 전체에서 리소스, 정책 및 규정 준수를 관리하는 데 도움이 되는 중요 규칙입니다. 예방 가드레일은 규정 준수 표준에 부합하도록 정책을 시행하며, 서비스 제어 정책과 IAM 권한 경계를 사용하여 구현됩니다. 탐지 가드레일은 정책 위반 및 규정 준수 문제를 감지하고 해결을 위한 알림을 생성하며, 이는 AWS Config Amazon GuardDuty AWS Security Hub CSPM, , AWS Trusted Advisor Amazon Inspector 및 사용자 지정 AWS Lambda 검사를 사용하여 구현됩니다.

## 가드레일(AI)

책임감 있고 안전한 AI 동작을 보장하기 위해 [에이전트](#) 입력 및 출력을 필터링, 검증 및 제약하는 안전 메커니즘입니다.

## H

### HA

[고가용성](#)을 참조하세요.

### 이기종 데이터베이스 마이그레이션

다른 데이터베이스 엔진을 사용하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Oracle에서 Amazon Aurora로) 이기종 마이그레이션은 일반적으로 리아키텍트 작업의 일부이며 스키마를 변환하는 것은 복잡한 작업일 수 있습니다. AWS 는 스키마 변환에 도움이 되는 [AWS SCT](#)를 [제공](#)합니다.

## 높은 가용성(HA)

문제나 재해 발생 시 개입 없이 지속적으로 운영할 수 있는 워크로드의 능력. HA 시스템은 자동으로 장애 조치되고, 지속적으로 고품질 성능을 제공하고, 성능에 미치는 영향을 최소화하면서 다양한 부하와 장애를 처리하도록 설계되었습니다.

## 히스토리언 현대화

제조 산업의 요구 사항을 더 잘 충족하도록 운영 기술(OT) 시스템을 현대화하고 업그레이드하는 데 사용되는 접근 방식입니다. 히스토리언은 공장의 다양한 출처에서 데이터를 수집하고 저장하는 데 사용되는 일종의 데이터베이스입니다.

## 홀드아웃 데이터

[기계 학습](#) 모델을 훈련하는 데 사용되는 데이터세트에서 보류되는 레이블이 지정된 기록 데이터의 일부입니다. 홀드아웃 데이터를 사용하여 모델 예측을 홀드아웃 데이터와 비교해 모델 성능을 평가할 수 있습니다.

## human-in-the-loop(HitL)

중요한 결정 시점에서 인적 검토 및 승인을 위해 [에이전트](#) 실행이 일시 중지되는 워크플로 패턴입니다.

## 동종 데이터베이스 마이그레이션

동일한 데이터베이스 엔진을 공유하는 대상 데이터베이스로 소스 데이터베이스 마이그레이션(예: Microsoft SQL Server에서 Amazon RDS for SQL Server로) 동종 마이그레이션은 일반적으로 리호스팅 또는 리플랫폼 작업의 일부입니다. 네이티브 데이터베이스 유틸리티를 사용하여 스키마를 마이그레이션할 수 있습니다.

## 핫 데이터

자주 액세스하는 데이터(예: 실시간 데이터 또는 최근 번역 데이터). 일반적으로 이 데이터에는 빠른 쿼리 응답을 제공하기 위한 고성능 스토리지 계층 또는 클래스가 필요합니다.

## 핫픽스

프로덕션 환경의 중요한 문제를 해결하기 위한 긴급 수정입니다. 핫픽스는 긴급하기 때문에 일반적인 DevOps 릴리스 워크플로 외부에서 실행됩니다.

## 하이퍼케어 기간

전환 직후 마이그레이션 팀이 문제를 해결하기 위해 클라우드에서 마이그레이션된 애플리케이션을 관리하고 모니터링하는 기간입니다. 일반적으로 이 기간은 1~4일입니다. 하이퍼케어 기간이 끝나면 마이그레이션 팀은 일반적으로 애플리케이션에 대한 책임을 클라우드 운영 팀에 넘깁니다.

## I

## IaC

[코드형 인프라](#)를 참조하세요.

## 자격 증명 기반 정책

AWS 클라우드 환경 내에서 권한을 정의하는 하나 이상의 IAM 보안 주체에 연결된 정책입니다.

## 유휴 애플리케이션

90일 동안 평균 CPU 및 메모리 사용량이 5~20%인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하거나 온프레미스에 유지하는 것이 일반적입니다.

## IIoT

[산업용 사물 인터넷](#)을 참조하세요.

## 변경 불가능한 인프라

기존 인프라를 업데이트, 패치 또는 수정하는 대신 프로덕션 워크로드에 대한 새 인프라를 배포하는 모델입니다. 변경 불가능한 인프라는 [변경 가능한 인프라](#)보다 본질적으로 더 일관되고 안정적이며 예측 가능합니다. 자세한 내용은 AWS Well-Architected Framework의 [변경 불가능한 인프라를 사용하여 배포](#) 모범 사례를 참조하세요.

## 인바운드(수신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 외부에서 네트워크 연결을 수락, 검사 및 라우팅하는 VPC입니다. [AWS Security Reference Architecture](#)에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

## 증분 마이그레이션

한 번에 전체 전환을 수행하는 대신 애플리케이션을 조금씩 마이그레이션하는 전환 전략입니다. 예를 들어, 처음에는 소수의 마이크로서비스나 사용자만 새 시스템으로 이동할 수 있습니다. 모든 것이 제대로 작동하는지 확인한 후에는 레거시 시스템을 폐기할 수 있을 때까지 추가 마이크로서비스 또는 사용자를 점진적으로 이동할 수 있습니다. 이 전략을 사용하면 대규모 마이그레이션과 관련된 위험을 줄일 수 있습니다.

## Industry 4.0

연결성, 실시간 데이터, 자동화, 분석 및 AI/ML의 발전을 통해 제조 프로세스의 현대화를 나타내기 위해 2016년에 [Klaus Schwab](#)에서 도입한 용어입니다.

## 인프라

애플리케이션의 환경 내에 포함된 모든 리소스와 자산입니다.

### 코드형 인프라(IaC)

구성 파일 세트를 통해 애플리케이션의 인프라를 프로비저닝하고 관리하는 프로세스입니다. IaC는 새로운 환경의 반복 가능성, 신뢰성 및 일관성을 위해 인프라 관리를 중앙 집중화하고, 리소스를 표준화하고, 빠르게 확장할 수 있도록 설계되었습니다.

### 산업용 사물 인터넷(IIoT)

제조, 에너지, 자동차, 의료, 생명과학, 농업 등의 산업 부문에서 인터넷에 연결된 센서 및 디바이스의 사용 자세한 내용은 [산업용 사물 인터넷\(IoT\) 디지털 트랜스포메이션 전략 구축](#)을 참조하십시오.

### 검사 VPC

AWS 다중 계정 아키텍처에서는 VPC(동일하거나 다른 AWS 리전), 인터넷 및 온프레미스 네트워크 간의 네트워크 트래픽 검사를 관리하는 중앙 집중식 VPCs. [AWS Security Reference Architecture](#)에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

### 사물 인터넷(IoT)

인터넷이나 로컬 통신 네트워크를 통해 다른 디바이스 및 시스템과 통신하는 센서 또는 프로세서가 내장된 연결된 물리적 객체의 네트워크 자세한 내용은 [IoT란?](#)을 참조하십시오.

### 해석력

모델의 예측이 입력에 따라 어떻게 달라지는지를 사람이 이해할 수 있는 정도를 설명하는 기계 학습 모델의 특성입니다. 자세한 내용은 [기계 학습 모델 해석 가능성을 참조하세요 AWS](#).

### IoT

[사물 인터넷](#)을 참조하세요.

### IT 정보 라이브러리(ITIL)

IT 서비스를 제공하고 이러한 서비스를 비즈니스 요구 사항에 맞게 조정하기 위한 일련의 모범 사례 ITIL은 ITSM의 기반을 제공합니다.

### IT 서비스 관리(ITSM)

조직의 IT 서비스 설계, 구현, 관리 및 지원과 관련된 활동 클라우드 운영을 ITSM 도구와 통합하는 방법에 대한 자세한 내용은 [운영 통합 가이드](#)를 참조하십시오.

## ITIL

[IT 정보 라이브러리](#)를 참조하세요.

## ITSM

[IT 서비스 관리](#)를 참조하세요.

## L

## 레이블 기반 액세스 제어(LBAC)

사용자 및 데이터 자체에 각각 보안 레이블 값을 명시적으로 할당하는 필수 액세스 제어(MAC)를 구현한 것입니다. 사용자 보안 레이블과 데이터 보안 레이블 간의 교차 부분에 따라 사용자가 볼 수 있는 행과 열이 결정됩니다.

## 랜딩 존

랜딩 존은 확장 가능하고 안전한 잘 설계된 다중 계정 AWS 환경입니다. 조직은 여기에서부터 보안 및 인프라 환경에 대한 확신을 가지고 워크로드와 애플리케이션을 신속하게 시작하고 배포할 수 있습니다. 랜딩 존에 대한 자세한 내용은 [안전하고 확장 가능한 다중 계정 AWS 환경 설정](#)을 참조하십시오.

## 대규모 언어 모델(LLM)

방대한 양의 데이터에서 사전 훈련된 딥 러닝 [AI](#) 모델입니다. LLM은 질문에 대한 답변, 문서 요약, 텍스트를 다른 언어로 번역, 문장 완성과 같은 여러 태스크를 수행할 수 있습니다. 자세한 내용은 [대규모 언어 모델\(LLM\)이란 무엇인가요?](#)를 참조하세요.

## 대규모 마이그레이션

300대 이상의 서버 마이그레이션입니다.

## LBAC

[레이블 기반 액세스 제어](#)를 참조하세요.

## 최소 권한

작업을 수행하는 데 필요한 최소 권한을 부여하는 보안 모범 사례입니다. 자세한 내용은 IAM 설명서의 [최소 권한 적용](#)을 참조하십시오.

## 리프트 앤드 시프트

[7R](#)을 참조하세요.

## 리틀 엔디안 시스템

가장 덜 중요한 바이트를 먼저 저장하는 시스템입니다. [엔디안](#)도 참조하세요.

## LLM

[대규모 언어 모델](#)을 참조하세요.

## 하위 환경

[환경](#)을 참조하세요.

## M

### 기계 학습(ML)

패턴 인식 및 학습에 알고리즘과 기법을 사용하는 인공지능의 한 유형입니다. ML은 사물 인터넷 (IoT) 데이터와 같은 기록된 데이터를 분석하고 학습하여 패턴을 기반으로 통계 모델을 생성합니다. 자세한 내용은 [기계 학습](#)을 참조하십시오.

### 기본 브랜치

[브랜치](#)를 참조하세요.

### 맬웨어

컴퓨터 보안 또는 프라이버시를 위협하도록 설계된 소프트웨어입니다. 맬웨어는 컴퓨터 시스템을 방해하거나 민감한 정보를 유출하거나 무단 액세스 권한을 확보할 수 있습니다. 맬웨어의 예로 바이러스, 웜, 랜섬웨어, 트로이 목마, 스파이웨어, 키로거 등이 있습니다.

### 관리형 서비스

AWS 서비스는 인프라 계층, 운영 체제 및 플랫폼을 AWS 운영하며 사용자는 엔드포인트에 액세스하여 데이터를 저장하고 검색합니다. 관리형 서비스의 예로 Amazon Simple Storage Service(Amazon S3) 및 Amazon DynamoDB가 있습니다. 이를 추상화된 서비스라고도 합니다.

### 제조 실행 시스템(MES)

원자재를 생산 현장에서 완제품으로 변환하는 생산 프로세스를 추적, 모니터링, 문서화 및 제어하기 위한 소프트웨어 시스템입니다.

## MAP

[Migration Acceleration Program](#)을 참조하세요.

## MCP

[모델 컨텍스트 프로토콜](#)을 참조하세요.

### Model Context Protocol(MCP)

[에이전트 간??? 통신](#)을 위한 상태 비저장 프로토콜입니다.

### MCP 서버

[모델 컨텍스트 프로토콜](#)을 통해 하나 이상의 [도구를](#) 노출하는 서비스입니다.

### 메커니즘

도구를 생성하고 도구 채택을 유도한 다음 조정을 위해 결과를 검사하는 전체 프로세스입니다. 메커니즘은 작동 시 자체적으로 강화하고 개선하는 주기입니다. 자세한 내용은 AWS Well-Architected Framework의 [메커니즘 구축](#)을 참조하세요.

### 멤버 계정

조직의 일부인 관리 계정을 AWS 계정 제외한 모든 계정. AWS Organizations 하나의 계정은 한 번에 하나의 조직 멤버만 될 수 있습니다.

## MES

[제조 실행 시스템](#)을 참조하세요.

### 메시지 큐 원격 분석 전송(MQTT)

리소스 제약이 있는 [IoT](#) 디바이스에 대한 [게시 및 구독](#) 패턴을 기반으로 하는 경량 Machine-to-Machine(M2M) 통신 프로토콜입니다.

### 마이크로서비스

잘 정의된 API를 통해 통신하고 일반적으로 소규모 자체 팀이 소유하는 소규모 독립 서비스입니다. 예를 들어, 보험 시스템에는 영업, 마케팅 등의 비즈니스 역량이나 구매, 청구, 분석 등의 하위 영역에 매핑되는 마이크로 서비스가 포함될 수 있습니다. 마이크로서비스의 이점으로 민첩성, 유연한 확장, 손쉬운 배포, 재사용 가능한 코드, 복원력 등이 있습니다. 자세한 내용은 [AWS 서버리스 서비스를 사용하여 마이크로서비스 통합](#)을 참조하세요.

### 마이크로서비스 아키텍처

각 애플리케이션 프로세스를 마이크로서비스로 실행하는 독립 구성 요소를 사용하여 애플리케이션을 구축하는 접근 방식입니다. 이러한 마이크로서비스는 경량 API를 사용하여 잘 정의된 인터페이스를 통해 통신합니다. 애플리케이션의 특정 기능에 대한 수요에 맞게 이 아키텍처의 각 마이크로서비스를 업데이트, 배포 및 조정할 수 있습니다. 자세한 내용은 [에서 마이크로서비스 구현을 참조하세요 AWS](#).

## Migration Acceleration Program(MAP)

조직이 클라우드로 전환하기 위한 강력한 운영 기반을 구축하고 초기 마이그레이션 비용을 상쇄하는 데 도움이 되는 컨설팅 지원, 교육 및 서비스를 제공하는 AWS 프로그램입니다. MAP에는 레거시 마이그레이션을 체계적인 방식으로 실행하기 위한 마이그레이션 방법론과 일반적인 마이그레이션 시나리오를 자동화하고 가속화하는 도구 세트가 포함되어 있습니다.

### 대규모 마이그레이션

애플리케이션 포트폴리오의 대다수를 웨이브를 통해 클라우드로 이동하는 프로세스로, 각 웨이브에서 더 많은 애플리케이션이 더 빠른 속도로 이동합니다. 이 단계에서는 이전 단계에서 배운 모범 사례와 교훈을 사용하여 팀, 도구 및 프로세스의 마이그레이션 팩토리를 구현하여 자동화 및 민첩한 제공을 통해 워크로드 마이그레이션을 간소화합니다. 이것은 [AWS 마이그레이션 전략](#)의 세 번째 단계입니다.

### 마이그레이션 팩토리

자동화되고 민첩한 접근 방식을 통해 워크로드 마이그레이션을 간소화하는 다기능 팀입니다. 마이그레이션 팩토리 팀에는 일반적으로 스프린트에서 일하는 운영, 비즈니스 분석가 및 소유자, 마이그레이션 엔지니어, 개발자, DevOps 전문가가 포함됩니다. 엔터프라이즈 애플리케이션 포트폴리오의 20~50%는 공장 접근 방식으로 최적화할 수 있는 반복되는 패턴으로 구성되어 있습니다. 자세한 내용은 이 콘텐츠 세트의 [클라우드 마이그레이션 팩토리 가이드](#)와 [마이그레이션 팩토리에 대한 설명](#)을 참조하십시오.

### 마이그레이션 메타데이터

마이그레이션을 완료하는 데 필요한 애플리케이션 및 서버에 대한 정보 각 마이그레이션 패턴에는 서로 다른 마이그레이션 메타데이터 세트가 필요합니다. 마이그레이션 메타데이터의 예로는 대상 서브넷, 보안 그룹 및 AWS 계정이 있습니다.

### 마이그레이션 패턴

사용되는 마이그레이션 전략, 마이그레이션 대상, 마이그레이션 애플리케이션 또는 서비스를 자세히 설명하는 반복 가능한 마이그레이션 작업입니다. 예: AWS Application Migration Service를 사용하여 Amazon EC2로 마이그레이션을 리호스팅합니다.

## Migration Portfolio Assessment(MPA)

AWS 클라우드로 마이그레이션하는 비즈니스 사례를 검증하기 위한 정보를 제공하는 온라인 도구입니다. MPA는 상세한 포트폴리오 평가(서버 적정 규모 조정, 가격 책정, TCO 비교, 마이그레이션 비용 분석)와 마이그레이션 계획(애플리케이션 데이터 분석 및 데이터 수집, 애플리케이션 그룹화, 마이그레이션 우선순위 지정, 웨이브 계획)을 제공합니다. [MPA 도구](#)(로그인 필요)는 모든 AWS 컨설턴트와 APN 파트너 컨설턴트가 무료로 사용할 수 있습니다.

## 마이그레이션 준비 상태 평가(MRA)

AWS CAF를 사용하여 조직의 클라우드 준비 상태에 대한 인사이트를 얻고, 강점과 약점을 식별하고, 식별된 격차를 해소하기 위한 행동 계획을 수립하는 프로세스입니다. 자세한 내용은 [마이그레이션 준비 가이드](#)를 참조하십시오. MRA는 [AWS 마이그레이션 전략](#)의 첫 번째 단계입니다.

### 마이그레이션 전략

워크로드를 AWS 클라우드로 마이그레이션하는 데 사용되는 접근 방식입니다. 자세한 내용은 이 용어집의 [7R 항목](#)과 [조직을 동원하여 대규모 마이그레이션 가속화](#)를 참조하세요.

## ML

[기계 학습](#)을 참조하세요.

### 현대화

비용을 절감하고 효율성을 높이고 혁신을 활용하기 위해 구식(레거시 또는 모놀리식) 애플리케이션과 해당 인프라를 클라우드의 민첩하고 탄력적이고 가용성이 높은 시스템으로 전환하는 것입니다. 자세한 내용은 [AWS 클라우드에서 애플리케이션을 현대화하기 위한 전략](#)을 참조하세요.

### 현대화 준비 상태 평가

조직 애플리케이션의 현대화 준비 상태를 파악하고, 이점, 위험 및 종속성을 식별하고, 조직이 해당 애플리케이션의 향후 상태를 얼마나 잘 지원할 수 있는지를 확인하는 데 도움이 되는 평가입니다. 평가 결과는 대상 아키텍처의 청사진, 현대화 프로세스의 개발 단계와 마일스톤을 자세히 설명하는 로드맵 및 파악된 격차를 해소하기 위한 실행 계획입니다. 자세한 내용은 [AWS 클라우드에서 애플리케이션의 현대화 준비 상태 평가](#)를 참조하세요.

### 모놀리식 애플리케이션(모놀리식 유형)

긴밀하게 연결된 프로세스를 사용하여 단일 서비스로 실행되는 애플리케이션입니다. 모놀리식 애플리케이션에는 몇 가지 단점이 있습니다. 한 애플리케이션 기능에 대한 수요가 급증하면 전체 아키텍처 규모를 조정해야 합니다. 코드 베이스가 커지면 모놀리식 애플리케이션의 기능을 추가하거나 개선하는 것도 더 복잡해집니다. 이러한 문제를 해결하기 위해 마이크로서비스 아키텍처를 사용할 수 있습니다. 자세한 내용은 [마이크로서비스로 모놀리식 유형 분해](#)를 참조하십시오.

## MPA

[Migration Portfolio Assessment](#)를 참조하세요.

## MQTT

[메시지 큐 원격 분석 전송](#)을 참조하세요.

## 멀티클래스 분류

여러 클래스에 대한 예측(2개 이상의 결과 중 하나 예측)을 생성하는 데 도움이 되는 프로세스입니다. 예를 들어, ML 모델이 '이 제품은 책인가요, 자동차인가요, 휴대폰인가요?' 또는 '이 고객이 가장 관심을 갖는 제품 범주는 무엇인가요?'라고 물을 수 있습니다.

## 변경 가능한 인프라

프로덕션 워크로드에 대한 기존 인프라를 업데이트하고 수정하는 모델입니다. 일관성, 신뢰성 및 예측 가능성을 높이기 위해 AWS Well-Architected Framework에서는 [변경 불가능한 인프라](#)를 모범 사례로 사용할 것을 권장합니다.

## O

### OAC

[오리진 액세스 제어](#)를 참조하세요.

### OAI

[오리진 액세스 ID](#)를 참조하세요.

### OCM

[조직 변경 관리](#)를 참조하세요.

## 오프라인 마이그레이션

마이그레이션 프로세스 중 소스 워크로드가 중단되는 마이그레이션 방법입니다. 이 방법은 가동 중지 증가를 수반하며 일반적으로 작고 중요하지 않은 워크로드에 사용됩니다.

## OI

[운영 통합](#)을 참조하세요.

### OLA

[운영 수준 계약](#)을 참조하세요.

## 온라인 마이그레이션

소스 워크로드를 오프라인 상태로 전환하지 않고 대상 시스템에 복사하는 마이그레이션 방법입니다. 워크로드에 연결된 애플리케이션은 마이그레이션 중에도 계속 작동할 수 있습니다. 이 방법은 가동 중지 차단 또는 최소화를 수반하며 일반적으로 중요한 프로덕션 워크로드에 사용됩니다.

## OPC-UA

[Open Process Communications - Unified Architecture\(OPC-UA\)](#)를 참조하세요.

### Open Process Communications - Unified Architecture(OPC-UA)

산업 자동화를 위한 Machine-to-Machine(M2M) 통신 프로토콜입니다. OPC-UA는 데이터 암호화, 인증 및 권한 부여 체계에 관한 상호 운용성 표준을 제공합니다.

### 운영 수준 협약(OLA)

서비스 수준에 관한 계약(SLA)을 지원하기 위해 직무 IT 그룹이 서로에게 제공하기로 약속한 내용을 명확히 하는 계약입니다.

### 운영 준비 상태 검토(ORR)

인시던트 및 잠재적 장애의 범위를 이해, 평가 또는 예방하거나 줄이는 데 도움이 되는 질문 체크리스트 및 관련 모범 사례입니다. 자세한 내용은 AWS Well-Architected Framework의 [운영 준비 상태 검토\(ORR\)](#)를 참조하세요.

### 운영 기술(OT)

물리적 환경에서 작동하여 산업 운영, 장비 및 인프라를 제어하는 하드웨어 및 소프트웨어 시스템입니다. 제조 분야에서 OT 및 정보 기술(IT) 시스템의 통합은 [Industry 4.0](#) 트랜스포메이션의 주요 중점 사항입니다.

### 운영 통합(OI)

클라우드에서 운영을 현대화하는 프로세스로 준비 계획, 자동화 및 통합을 수반합니다. 자세한 내용은 [운영 통합 가이드](#)를 참조하십시오.

### 조직 트레일

조직 내 모든에 대한 모든 이벤트를 로깅 AWS CloudTrail 하는 AWS 계정에서 생성된 추적입니다 AWS Organizations. 이 트레일은 조직에 속한 각 AWS 계정에 생성되고 각 계정의 활동을 추적합니다. 자세한 내용은 CloudTrail 설명서의 [Creating a trail for an organization](#)을 참조하십시오.

### 조직 변경 관리(OCM)

사람, 문화 및 리더십 관점에서 중대하고 파괴적인 비즈니스 혁신을 관리하기 위한 프레임워크입니다. OCM은 변화 채택을 가속화하고, 과도기적 문제를 해결하고, 문화 및 조직적 변화를 주도함으로써 조직이 새로운 시스템 및 전략을 준비하고 전환할 수 있도록 지원합니다. AWS 마이그레이션 전략에서는 클라우드 채택 프로젝트에 필요한 변경 속도 때문에이 프레임워크를 인력 가속화라고 합니다. 자세한 내용은 [사용 가이드](#)를 참조하십시오.

## 오리진 액세스 제어(OAC)

CloudFront에서 Amazon Simple Storage Service(S3) 콘텐츠를 보호하기 위해 액세스를 제한하는 고급 옵션입니다. OAC는 AWS KMS (SSE-KMS)를 사용한 모든 서버 측 암호화 AWS 리전와 S3 버킷에 대한 동적 PUT 및 DELETE 요청에서 모든 S3 버킷을 지원합니다.

## 오리진 액세스 ID(OAI)

CloudFront에서 Amazon S3 콘텐츠를 보호하기 위해 액세스를 제한하는 옵션입니다. OAI를 사용하면 CloudFront는 Amazon S3가 인증할 수 있는 보안 주체를 생성합니다. 인증된 보안 주체는 특정 CloudFront 배포를 통해서만 S3 버킷의 콘텐츠에 액세스할 수 있습니다. 더 세분화되고 향상된 액세스 제어를 제공하는 [OAC](#)도 참조하십시오.

## ORR

[운영 준비 상태 검토](#)를 참조하세요.

## OT

[운영 기술](#)을 참조하세요.

## 아웃바운드(송신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 내에서 시작된 네트워크 연결을 처리하는 VPC입니다. [AWS Security Reference Architecture](#)에서는 애플리케이션과 더 넓은 인터넷 간의 양방향 인터페이스를 보호하기 위해 인바운드, 아웃바운드 및 검사 VPC로 네트워크 계정을 설정할 것을 권장합니다.

## P

### 권한 경계

사용자나 역할이 가질 수 있는 최대 권한을 설정하기 위해 IAM 보안 주체에 연결되는 IAM 관리 정책입니다. 자세한 내용은 IAM 설명서의 [권한 경계](#)를 참조하십시오.

### 개인 식별 정보(PII)

직접 보거나 다른 관련 데이터와 함께 짝을 지을 때 개인의 신원을 합리적으로 추론하는 데 사용할 수 있는 정보입니다. PII의 예로는 이름, 주소, 연락처 정보 등이 있습니다.

## PII

[개인 식별 정보](#)를 참조하세요.

## 플레이북

클라우드에서 핵심 운영 기능을 제공하는 등 마이그레이션과 관련된 작업을 캡처하는 일련의 사전 정의된 단계입니다. 플레이북은 스크립트, 자동화된 런북 또는 현대화된 환경을 운영하는 데 필요한 프로세스나 단계 요약의 형태를 취할 수 있습니다.

## PLC

[프로그래밍 가능 로직 컨트롤러](#)를 참조하세요.

## PLM

[제품 수명 주기 관리](#)를 참조하세요.

## 정책

권한 정의([ID 기반 정책](#) 참조), 액세스 조건 지정([리소스 기반 정책](#) 참조), AWS Organizations 내 조직의 모든 계정에 대한 최대 권한 정의([서비스 제어 정책](#) 참조)와 같은 작업을 수행할 수 있는 객체입니다.

## 다국어 지속성

데이터 액세스 패턴 및 기타 요구 사항을 기반으로 독립적으로 마이크로서비스의 데이터 스토리지 기술 선택. 마이크로서비스가 동일한 데이터 스토리지 기술을 사용하는 경우 구현 문제가 발생하거나 성능이 저하될 수 있습니다. 요구 사항에 가장 적합한 데이터 저장소를 사용하면 마이크로서비스를 더 쉽게 구현하고 성능과 확장성을 높일 수 있습니다.

## 포트폴리오 평가

마이그레이션을 계획하기 위해 애플리케이션 포트폴리오를 검색 및 분석하고 우선순위를 정하는 프로세스입니다. 자세한 내용은 [마이그레이션 준비 상태 평가](#)를 참조하십시오.

## 조건자

보통 WHERE 절에 있는 true 또는 false를 반환하는 쿼리 조건입니다.

## 푸시다운 조건자

전송 전에 쿼리의 데이터를 필터링하는 데이터베이스 쿼리 최적화 기법입니다. 이렇게 하면 관계형 데이터베이스에서 검색하고 처리해야 하는 데이터의 양이 줄고 쿼리 성능이 향상됩니다.

## 예방적 제어

이벤트 발생을 방지하도록 설계된 보안 제어입니다. 이 제어는 네트워크에 대한 무단 액세스나 원치 않는 변경을 방지하는 데 도움이 되는 1차 방어선입니다. 자세한 내용은 Implementing security controls on AWS의 [Preventative controls](#)를 참조하십시오.

## 보안 주체

작업을 수행하고 리소스에 액세스할 수 있는 AWS IAM 엔터티입니다. 이 엔터티는 일반적으로 , AWS 계정 IAM 역할 또는 사용자의 루트 사용자입니다. 자세한 내용은 IAM 설명서의 [역할 용어 및 개념](#)의 보안 주체를 참조하십시오.

## 개인 정보 보호 중심 설계

전체 개발 프로세스에서 개인 정보를 고려하는 시스템 엔지니어링에서의 접근 방식입니다.

## 프라이빗 호스팅 영역

Amazon Route 53에서 하나 이상의 VPC 내 도메인과 하위 도메인에 대한 DNS 쿼리에 응답하는 방법에 대한 정보가 담긴 컨테이너입니다. 자세한 내용은 Route 53 설명서의 [프라이빗 호스팅 영역 작업](#)을 참조하십시오.

## 선제적 제어

규정 미준수 리소스의 배포를 방지하도록 설계된 [보안 제어](#)입니다. 이러한 제어는 리소스를 프로비저닝하기 전에 리소스를 스캔합니다. 리소스가 제어를 준수하지 않으면 프로비저닝되지 않습니다. 자세한 내용은 AWS Control Tower 설명서의 [제어 참조 가이드](#)를 참조하고 보안 [제어 구현의 사전 예방적 제어](#)를 참조하세요. AWS

## 제품 수명 주기 관리(PLM)

설계, 개발 및 출시부터 성장 및 성숙도를 거쳐 거부 및 제거에 이르기까지 전체 수명 주기 동안 제품의 데이터 및 프로세스 관리를 나타냅니다.

## 프로덕션 환경

[환경](#)을 참조하세요.

## 프로그래밍 가능 로직 컨트롤러(PLC)

제조 분야에서 기계를 모니터링하고 제조 프로세스를 자동화하는 매우 안정적이고 적응력이 뛰어난 컴퓨터입니다.

## 프롬프트 체이닝

한 [LLM](#) 프롬프트의 출력을 다음 프롬프트의 입력으로 사용하여 더 나은 응답을 생성합니다. 이 기법은 복잡한 작업을 하위 작업으로 나누거나 예비 응답을 반복적으로 세부 조정하거나 확장하는데 사용됩니다. 이를 통해 모델 응답의 정확성과 관련성을 개선하고 보다 세분화되고 개인화된 결과를 얻을 수 있습니다.

## 가명화

데이터세트의 개인 식별자를 자리 표시자 값으로 바꾸는 프로세스입니다. 가명화는 개인 정보를 보호하는 데 도움이 될 수 있습니다. 가명화된 데이터는 여전히 개인 데이터로 간주됩니다.

## 게시/구독(pub/sub)

여러 마이크로서비스에서 비동기 통신을 지원하여 확장성과 응답성을 개선하는 패턴입니다. 예를 들어 마이크로서비스 기반 [MES](#)에서 마이크로서비스는 다른 마이크로서비스가 구독할 수 있는 채널에 이벤트 메시지를 게시할 수 있습니다. 시스템은 게시 서비스를 변경하지 않고도 새 마이크로 서비스를 추가할 수 있습니다.

## Q

### 쿼리 계획

SQL 관계형 데이터베이스 시스템의 데이터에 액세스하는 데 사용되는 명령어와 같은 일련의 단계입니다.

### 쿼리 계획 회귀

데이터베이스 서비스 최적화 프로그램이 데이터베이스 환경을 변경하기 전보다 덜 최적의 계획을 선택하는 경우입니다. 통계, 제한 사항, 환경 설정, 쿼리 파라미터 바인딩 및 데이터베이스 엔진 업데이트의 변경으로 인해 발생할 수 있습니다.

## R

### RACI 매트릭스

[Responsible, Accountable, Consulted, Informed\(RACI\)](#)를 참조하세요.

### RAG

[검색 증강 생성](#)을 참조하세요.

### 랜섬웨어

결제 완료될 때까지 컴퓨터 시스템이나 데이터에 대한 액세스를 차단하도록 설계된 악성 소프트웨어입니다.

### RASCI 매트릭스

[Responsible, Accountable, Consulted, Informed\(RACI\)](#)를 참조하세요.

## RCAC

[행 및 열 액세스 제어](#)를 참조하세요.

### 읽기 전용 복제본

읽기 전용 용도로 사용되는 데이터베이스의 사본입니다. 쿼리를 읽기 전용 복제본으로 라우팅하여 기본 데이터베이스의 로드를 줄일 수 있습니다.

### 리아키텍팅

[7R](#)을 참조하세요.

### Recovery Point Objective(RPO)

마지막 데이터 복구 시점 이후 허용되는 최대 시간입니다. 이에 따라 마지막 복구 시점과 서비스 중단 사이에 허용되는 데이터 손실로 간주되는 범위가 결정됩니다.

### Recovery Time Objective(RTO)

서비스 중단과 서비스 복원 사이의 허용 가능한 지연 시간입니다.

### 리팩터링

[7R](#)을 참조하세요.

### 리전

지리적 영역의 AWS 리소스 모음입니다. 각 AWS 리전은 내결함성, 안정성 및 복원력을 제공하기 위해 서로 격리되고 독립적입니다. 자세한 내용은 [계정에서 사용할 수 있는 AWS 리전 지정](#)을 참조하세요.

### 회귀

숫자 값을 예측하는 ML 기법입니다. 예를 들어, '이 집은 얼마에 팔릴까?'라는 문제를 풀기 위해 ML 모델은 선형 회귀 모델을 사용하여 주택에 대해 알려진 사실(예: 면적)을 기반으로 주택의 매매 가격을 예측할 수 있습니다.

### 리호스팅

[7R](#)을 참조하세요.

### 릴리스

배포 프로세스에서 변경 사항을 프로덕션 환경으로 승격시키는 행위입니다.

## 재배치

[7R](#)을 참조하세요.

## 리플랫폼

[7R](#)을 참조하세요.

## 재구매

[7R](#)을 참조하세요.

## 복원력

중단에 저항하거나 중단을 복구할 수 있는 애플리케이션의 기능입니다. [고가용성](#) 및 [재해 복구](#)는 AWS 클라우드에서 복원력을 계획할 때 일반적인 고려 사항입니다. 자세한 내용은 [AWS 클라우드 복원력](#)을 참조하세요.

## 리소스 기반 정책

Amazon S3 버킷, 엔드포인트, 암호화 키 등의 리소스에 연결된 정책입니다. 이 유형의 정책은 액세스가 허용된 보안 주체, 지원되는 작업 및 충족해야 하는 기타 조건을 지정합니다.

## RACI(Responsible, Accountable, Consulted, Informed) 매트릭스

마이그레이션 활동 및 클라우드 운영에 참여하는 모든 당사자의 역할과 책임을 정의하는 매트릭스입니다. 매트릭스 이름은 매트릭스에 정의된 책임 유형에서 파생됩니다. 실무 담당자 (R), 의사 결정권자 (A), 업무 수행 조언자 (C), 결과 통보 대상자 (I). 지원자는 (S) 선택사항입니다. 지원자를 포함하면 매트릭스를 RASCI 매트릭스라고 하고, 지원자를 제외하면 RACI 매트릭스라고 합니다.

## 대응 제어

보안 기준에서 벗어나거나 부정적인 이벤트를 해결하도록 설계된 보안 제어입니다. 자세한 내용은 AWS에서 보안 제어 구현의 [대응 제어](#)를 참조하세요.

## retain

[7R](#)을 참조하세요.

## 사용 중지

[7R](#)을 참조하세요.

## 검색 증강 세대(RAG)

응답을 생성하기 전에 [LLM](#)이 훈련 데이터 소스 외부에 있는 신뢰할 수 있는 데이터 소스를 참조하는 [생성형 AI](#) 기술입니다. 예를 들어 RAG 모델은 조직의 지식 기반 또는 사용자 지정 데이터에 대

한 시맨틱 검색을 수행할 수 있습니다. 자세한 내용은 [검색 증강 생성\(RAG\)이란 무엇인가요?](#)를 참조하세요.

## 교체

공격자가 자격 증명에 액세스하는 것을 더욱 어렵게 만들기 위해 [보안 암호](#)를 주기적으로 업데이트하는 프로세스입니다.

## 행 및 열 액세스 제어(RCAC)

액세스 규칙이 정의된 기본적이고 유연한 SQL 표현식을 사용합니다. RCAC는 행 권한과 열 마스크로 구성됩니다.

## RPO

[목표 복구 시점\(RPO\)](#)을 참조하세요.

## RTO

[목표 복구 시간\(RTO\)](#)을 참조하세요.

## 런북

특정 작업을 수행하는 데 필요한 일련의 수동 또는 자동 절차입니다. 일반적으로 오류율이 높은 반복 작업이나 절차를 간소화하기 위해 런북을 만듭니다.

## S

### SAML 2.0

많은 ID 제공업체(idP)에서 사용하는 개방형 표준입니다. 이 기능을 사용하면 연동 SSO(Single Sign-On)를 AWS Management Console 사용할 수 있으므로 사용자는 조직의 모든 사용자에게 대해 IAM에서 사용자를 생성하지 않고도 로그인하거나 AWS API 작업을 호출할 수 있습니다. SAML 2.0 기반 페더레이션에 대한 자세한 내용은 IAM 설명서의 [SAML 2.0 기반 페더레이션 정보](#)를 참조하십시오.

### SCADA

[감독 제어 및 데이터 획득](#)을 참조하세요.

### SCP

[서비스 제어 정책](#)을 참조하세요.

## 보안 암호

에는 암호 또는 사용자 자격 증명과 같이 암호화된 형식으로 저장하는 AWS Secrets Manager 기밀 또는 제한된 정보가 있습니다. 보안 암호 값과 메타데이터로 구성됩니다. 보안 암호 값은 바이너리, 단일 문자열 또는 여러 문자열일 수 있습니다. 자세한 내용은 AWS Secrets Manager 설명서의 [Secrets Manager 보안 암호란 무엇인가요?](#)를 참조하세요.

## 보안 중심 설계

전체 개발 프로세스에서 보안을 고려하는 시스템 엔지니어링에서의 접근 방식입니다.

## 보안 제어

위협 행위자가 보안 취약성을 악용하는 능력을 방지, 탐지 또는 감소시키는 기술적 또는 관리적 가드레일입니다. 보안 제어는 [예방](#), [감지](#), [대응](#), [선제적](#)과 같은 기본적인 네 가지 보안 제어 유형으로 구분됩니다.

## 보안 강화

공격 표면을 줄여 공격에 대한 저항력을 높이는 프로세스입니다. 더 이상 필요하지 않은 리소스 제거, 최소 권한 부여의 보안 모범 사례 구현, 구성 파일의 불필요한 기능 비활성화 등의 작업이 여기에 포함될 수 있습니다.

## 보안 정보 및 이벤트 관리(SIEM) 시스템

보안 정보 관리(SIM)와 보안 이벤트 관리(SEM) 시스템을 결합하는 도구 및 서비스입니다. SIEM 시스템은 서버, 네트워크, 디바이스 및 기타 소스에서 데이터를 수집, 모니터링 및 분석하여 위협과 보안 침해를 탐지하고 알림을 생성합니다.

## 보안 응답 자동화

보안 이벤트에 자동으로 응답하거나 이를 해결하도록 설계된 사전 정의되고 프로그래밍된 작업입니다. 이러한 자동화는 보안 모범 사례를 구현하는 데 도움이 되는 [탐지](#) 또는 [대응](#) AWS 보안 제어 역할을 합니다. 자동화된 응답 작업의 예로 VPC 보안 그룹 수정, Amazon EC2 인스턴스 패치 적용 또는 자격 증명 교체 등이 있습니다.

## 서버 측 암호화

대상에서 데이터를 수신하는 AWS 서비스에 의한 데이터 암호화.

## 서비스 제어 정책(SCP)

AWS Organizations에 속한 조직의 모든 계정에 대한 권한을 중앙 집중식으로 제어하는 정책입니다. SCP는 관리자가 사용자 또는 역할에 위임할 수 있는 작업에 대해 제한을 설정하거나 가드레일을 정의합니다. SCP를 허용 목록 또는 거부 목록으로 사용하여 허용하거나 금지할 서비스 또는 작

업을 지정할 수 있습니다. 자세한 내용은 AWS Organizations 설명서의 [서비스 제어 정책을](#) 참조하세요.

## 서비스 엔드포인트

에 대한 진입점의 URL입니다 AWS 서비스. 엔드포인트를 사용하여 대상 서비스에 프로그래밍 방식으로 연결할 수 있습니다. 자세한 내용은 AWS 일반 참조의 [AWS 서비스 엔드포인트](#)를 참조하십시오.

## 서비스 수준에 관한 계약(SLA)

IT 팀이 고객에게 제공하기로 약속한 내용(예: 서비스 가동 시간 및 성능)을 명시한 계약입니다.

## 서비스 수준 지표(SLI)

오류 발생률, 가용성 또는 처리량과 같은 서비스의 성능 측면에 대한 측정값입니다.

## 서비스 수준 목표(SLO)

[서비스 수준 지표](#)로 측정되는 서비스의 상태를 나타내는 목표 지표입니다.

## 공동 책임 모델

클라우드 보안 및 규정 준수를 AWS 위해와 공유하는 책임을 설명하는 모델입니다. AWS 는 클라우드의 보안을 담당하는 반면, 사용자는 클라우드의 보안을 담당합니다. 자세한 내용은 [공동 책임 모델](#)을 참조하십시오.

## 새도우 AI

조직 내 관리형 채널 외부에서 구축되거나 사용되는 승인되지 않은 [AI](#) 애플리케이션입니다.

## SIEM

[보안 정보 및 이벤트 관리 시스템](#)을 참조하세요.

## 단일 장애점(SPOF)

애플리케이션을 중단시킬 수 있는 애플리케이션의 중요한 단일 구성 요소에서 발생하는 장애입니다.

## SLA

[서비스 수준 계약](#)을 참조하세요.

## SLI

[서비스 수준 지표](#)를 참조하세요.

## SLO

[서비스 수준 목표](#)를 참조하세요.

### 분할 앤 시드 모델

현대화 프로젝트를 확장하고 가속화하기 위한 패턴입니다. 새로운 기능과 제품 릴리스가 정의되면 핵심 팀이 분할되어 새로운 제품 팀이 만들어집니다. 이를 통해 조직의 역량과 서비스 규모를 조정하고, 개발자 생산성을 개선하고, 신속한 혁신을 지원할 수 있습니다. 자세한 내용은 [AWS 클라우드에서 애플리케이션을 현대화하기 위한 단계별 접근 방식](#)을 참조하세요.

## SPOF

[단일 장애점](#)을 참조하세요.

### 스타 스키마

하나의 큰 팩트 테이블을 사용하여 트랜잭션 또는 측정된 데이터를 저장하고 하나 이상의 더 작은 차원 테이블을 사용하여 데이터 속성을 저장하는 데이터베이스 조직 구조입니다. 이 구조는 [데이터 웨어하우스](#)에서 또는 비즈니스 인텔리전스 목적으로 사용하도록 설계되었습니다.

### Strangler Fig 패턴

레거시 시스템을 폐기할 수 있을 때까지 시스템 기능을 점진적으로 다시 작성하고 교체하여 모놀리식 시스템을 현대화하기 위한 접근 방식. 이 패턴은 무화과 덩굴이 나무로 자라 결국 숙주를 압도하고 대체하는 것과 비슷합니다. [Martin Fowler](#)가 모놀리식 시스템을 다시 작성할 때 위험을 관리하는 방법으로 이 패턴을 도입했습니다. 이 패턴을 적용하는 방법의 예는 [컨테이너 및 Amazon API Gateway를 사용하여 기존의 Microsoft ASP.NET\(ASMX\) 웹 서비스를 점진적으로 현대화하는 방법](#)을 참조하십시오.

### 서브넷

VPC의 IP 주소 범위입니다. 서브넷은 단일 가용 영역에 상주해야 합니다.

### 감독 제어 및 데이터 획득(SCADA)

제조 분야에서 하드웨어와 소프트웨어를 사용하여 물리적 자산과 프로덕션 작업을 모니터링하는 시스템입니다.

### 대칭 암호화

동일한 키를 사용하여 데이터를 암호화하고 복호화하는 암호화 알고리즘입니다.

## 합성 테스트

사용자 상호 작용을 시뮬레이션하여 잠재적 문제를 감지하거나 성능을 모니터링하는 방식으로 진행되는 시스템 테스트입니다. [Amazon CloudWatch Synthetics](#)를 사용하여 이러한 테스트를 생성할 수 있습니다.

## 시스템 프롬프트

[LLM](#)에 컨텍스트, 명령 또는 지침을 제공하여 동작을 지시하는 기법입니다. 시스템 프롬프트는 컨텍스트를 설정하고 사용자와의 상호 작용을 위한 규칙을 설정하는 데 도움이 됩니다.

## T

### tags

AWS 리소스를 구성하기 위한 메타데이터 역할을 하는 키-값 페어입니다. 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색, 필터링할 수 있습니다. 자세한 내용은 [AWS 리소스에 태그 지정](#)을 참조하십시오.

### 대상 변수

지도 ML에서 예측하려는 값으로, 결과 변수라고도 합니다. 예를 들어, 제조 설정에서 대상 변수는 제품 결함일 수 있습니다.

### 작업 목록

런북을 통해 진행 상황을 추적하는 데 사용되는 도구입니다. 작업 목록에는 런북의 개요와 완료해야 할 일반 작업 목록이 포함되어 있습니다. 각 일반 작업에 대한 예상 소요 시간, 소유자 및 진행 상황이 작업 목록에 포함됩니다.

### 테스트 환경

[환경](#)을 참조하세요.

### 훈련

ML 모델이 학습할 수 있는 데이터를 제공하는 것입니다. 훈련 데이터에는 정답이 포함되어야 합니다. 학습 알고리즘은 훈련 데이터에서 대상(예측하려는 답)에 입력 데이터 속성을 매핑하는 패턴을 찾고, 이러한 패턴을 캡처하는 ML 모델을 출력합니다. 그런 다음 ML 모델을 사용하여 대상을 모르는 새 데이터에 대한 예측을 할 수 있습니다.

### tool

[에이전트](#)가 외부 시스템에서 작업을 수행하기 위해 호출할 수 있는 함수 또는 API입니다.

## Transit Gateway

VPC와 온프레미스 네트워크를 상호 연결하는 데 사용할 수 있는 네트워크 전송 허브입니다. 자세한 내용은 AWS Transit Gateway 설명서의 [전송 게이트웨이란 무엇입니까?](#)를 참조하세요.

### 트렁크 기반 워크플로

개발자가 기능 브랜치에서 로컬로 기능을 구축하고 테스트한 다음 해당 변경 사항을 기본 브랜치에 병합하는 접근 방식입니다. 이후 기본 브랜치는 개발, 프로덕션 이전 및 프로덕션 환경에 순차적으로 구축됩니다.

### 신뢰할 수 있는 액세스

사용자를 대신하여 AWS Organizations 및 해당 계정에서 조직에서 작업을 수행할 수 있도록 지정한 서비스에 권한을 부여합니다. 신뢰할 수 있는 서비스는 필요할 때 각 계정에 서비스 연결 역할을 생성하여 관리 작업을 수행합니다. 자세한 내용은 설명서의 [다른 AWS 서비스와 AWS Organizations 함께 사용](#)을 참조하세요 AWS Organizations .

### 튜닝

ML 모델의 정확도를 높이기 위해 훈련 프로세스의 측면을 여러 변경하는 것입니다. 예를 들어, 레이블링 세트를 생성하고 레이블을 추가한 다음 다양한 설정에서 이러한 단계를 여러 번 반복하여 모델을 최적화하는 방식으로 ML 모델을 훈련할 수 있습니다.

### 피자 두 판 팀

피자 두 판이면 충분한 소규모 DevOps 팀. 피자 두 판 팀 규모는 소프트웨어 개발에 있어 가능한 최상의 공동 작업 기회를 보장합니다.

## U

### 불확실성

예측 ML 모델의 신뢰성을 저해할 수 있는 부정확하거나 불완전하거나 알려지지 않은 정보를 나타내는 개념입니다. 불확실성에는 두 가지 유형이 있습니다. 인식론적 불확실성은 제한적이고 불완전한 데이터에 의해 발생하는 반면, 우연한 불확실성은 데이터에 내재된 노이즈와 무작위성에 의해 발생합니다.

### 차별화되지 않은 작업

애플리케이션을 만들고 운영하는 데 필요하지만 최종 사용자에게 직접적인 가치를 제공하거나 경쟁 우위를 제공하지 못하는 작업을 헤비 리프팅이라고도 합니다. 차별화되지 않은 작업의 예로는 조달, 유지보수, 용량 계획 등이 있습니다.

## 상위 환경

[환경](#)을 참조하세요.

## V

### 정리

스토리지를 회수하고 성능을 향상시키기 위해 증분 업데이트 후 정리 작업을 수반하는 데이터베이스 유지 관리 작업입니다.

### 버전 제어

리포지토리의 소스 코드 변경과 같은 변경 사항을 추적하는 프로세스 및 도구입니다.

### VPC 피어링

프라이빗 IP 주소를 사용하여 트래픽을 라우팅할 수 있게 하는 두 VPC 간의 연결입니다. 자세한 내용은 Amazon VPC 설명서의 [VPC 피어링이란?](#)을 참조하십시오.

### 취약성

시스템 보안을 손상시키는 소프트웨어 또는 하드웨어 결함입니다.

## W

### 웹 캐시

자주 액세스하는 최신 관련 데이터를 포함하는 버퍼 캐시입니다. 버퍼 캐시에서 데이터베이스 인스턴스를 읽을 수 있기 때문에 주 메모리나 디스크에서 읽는 것보다 빠릅니다.

### 웜 데이터

자주 액세스하지 않는 데이터입니다. 이런 종류의 데이터를 쿼리할 때는 일반적으로 적절히 느린 쿼리가 허용됩니다.

### 창 함수

현재 레코드와 어떤 식으로든 관련된 행 그룹에서 계산을 수행하는 SQL 함수입니다. 창 함수는 이동 평균을 계산하거나 현재 행의 상대적 위치를 기반으로 행 값에 액세스하는 등의 태스크를 처리하는 데 유용합니다.

## 워크로드

고객 대면 애플리케이션이나 백엔드 프로세스 같이 비즈니스 가치를 창출하는 리소스 및 코드 모음입니다.

## 워크스트림

마이그레이션 프로젝트에서 특정 작업 세트를 담당하는 직무 그룹입니다. 각 워크스트림은 독립적이지만 프로젝트의 다른 워크스트림을 지원합니다. 예를 들어, 포트폴리오 워크스트림은 애플리케이션 우선순위 지정, 웨이브 계획, 마이그레이션 메타데이터 수집을 담당합니다. 포트폴리오 워크스트림은 이러한 자산을 마이그레이션 워크스트림에 전달하고, 마이그레이션 워크스트림은 서버와 애플리케이션을 마이그레이션합니다.

## WORM

[Write Once, Read Many\(WORM\)](#)를 참조하세요.

## WQF

[AWS Workload Qualification Framework](#)를 참조하세요.

## Write Once Read Many(WORM)

데이터를 한 번 쓰고 데이터가 삭제되거나 수정되지 않도록 하는 스토리지 모델입니다. 권한 있는 사용자는 필요한 만큼 여러 번 데이터를 읽을 수 있지만 데이터를 변경할 수는 없습니다. 이 데이터 스토리지 인프라는 [변경 불가능](#)한 항목으로 간주됩니다.

## Z

### 제로데이 익스플로잇

[제로데이 취약성](#)을 악용하는 공격(일반적으로 맬웨어)입니다.

### 제로데이 취약성

프로덕션 시스템의 명백한 결함 또는 취약성입니다. 위협 행위자는 이러한 유형의 취약성을 사용하여 시스템을 공격할 수 있습니다. 개발자는 공격의 결과로 취약성을 인지하는 경우가 많습니다.

### 제로샷 프롬프팅

태스크를 수행하기 위해 [LLM](#)에 명령을 제공하지만 안내에 도움이 되는 예제(샷)는 제공하지 않습니다. LLM은 사전 훈련된 지식을 사용하여 태스크를 처리해야 합니다. 제로샷 프롬프팅의 효과는 태스크의 복잡성과 프롬프트의 품질에 따라 달라집니다. [퓨샷 프롬프팅](#)도 참조하세요.

## 좀비 애플리케이션

평균 CPU 및 메모리 사용량이 5% 미만인 애플리케이션입니다. 마이그레이션 프로젝트에서는 이러한 애플리케이션을 사용 중지하는 것이 일반적입니다.

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