aws

Apache Spark AWS Glue 작업의 성능 조정 모범 사례

: Apache Spark AWS Glue 작업의 성능 조정 모범 사례

Table of Contents

Apache Spark AWS Glue 작업의 성능 조정 모범 사례

로만 마이어스, 타카시 오니쿠라, 노리타카 세키야마, Amazon Web Services ()AWS

2023년 12월(문서 기록)

AWS Glue 성능 튜닝을 위한 다양한 옵션을 제공합니다. 이 가이드에서는 Apache Spark AWS Glue 튜닝에 대한 주요 주제를 정의합니다. 그런 다음 Apache Spark AWS Glue 작업에 맞게 이를 조정할 때 따라야 할 기본 전략을 제공합니다. 이 가이드를 통해 에서 사용할 수 있는 메트릭을 해석하여 성능 문제를 식별하는 방법을 알아보십시오. AWS Glue그런 다음 이러한 문제를 해결하기 위한 전략을 통합하여 성능을 극대화하고 비용을 최소화하십시오.

이 가이드에서는 다음과 같은 조정 방법을 다룹니다.

- 클러스터 용량 확장
- 최신 AWS Glue 버전 사용
- 데이터 스캔량을 줄이세요.
- 작업 병렬화
- 계획 오버헤드를 최소화합니다.
- 셔플 최적화
- 사용자 정의 함수 최적화

아파치 스파크의 주요 주제

이 섹션에서는 Apache Spark의 기본 개념과 Apache Spark 성능 AWS Glue 튜닝에 대한 주요 주제를 설명합니다. 실제 튜닝 전략을 논의하기 전에 이러한 개념과 주제를 이해하는 것이 중요합니다.

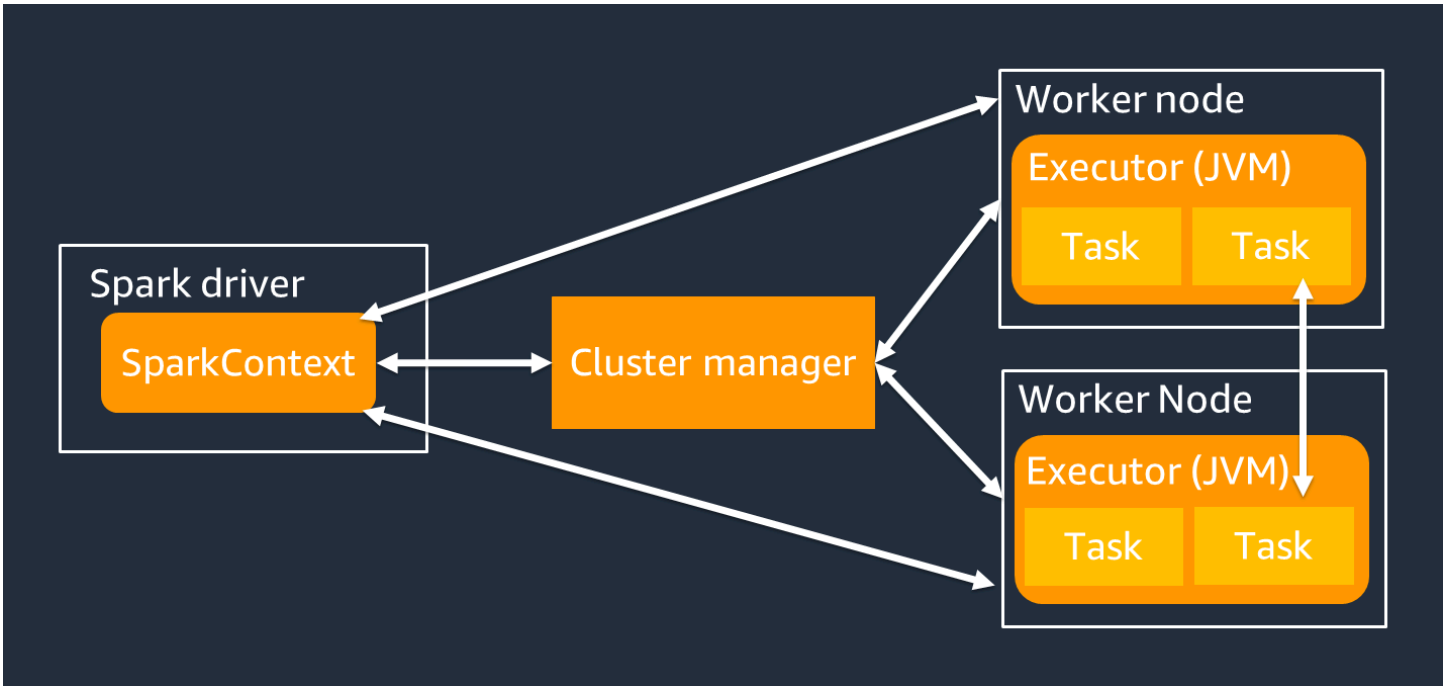
아키텍처

Spark 드라이버는 주로 Spark 애플리케이션을 개별 작업자가 수행할 수 있는 작업으로 분할하는 역할을 합니다. Spark 드라이버의 책임은 다음과 같습니다.

- `main()`코드에서 실행
- 실행 계획 생성
- 클러스터의 리소스를 관리하는 클러스터 관리자와 함께 Spark 실행자를 프로비저닝합니다.
- Spark 실행자를 위한 작업 예약 및 작업 요청
- 작업 진행 및 복구 관리

`SparkContext`객체를 사용하여 작업 실행을 위해 Spark 드라이버와 상호 작용합니다.

Spark 실행기는 Spark 드라이버에서 전달된 데이터를 보관하고 작업을 실행하는 작업자입니다. Spark 실행기의 수는 클러스터 크기에 따라 증가하거나 감소합니다.

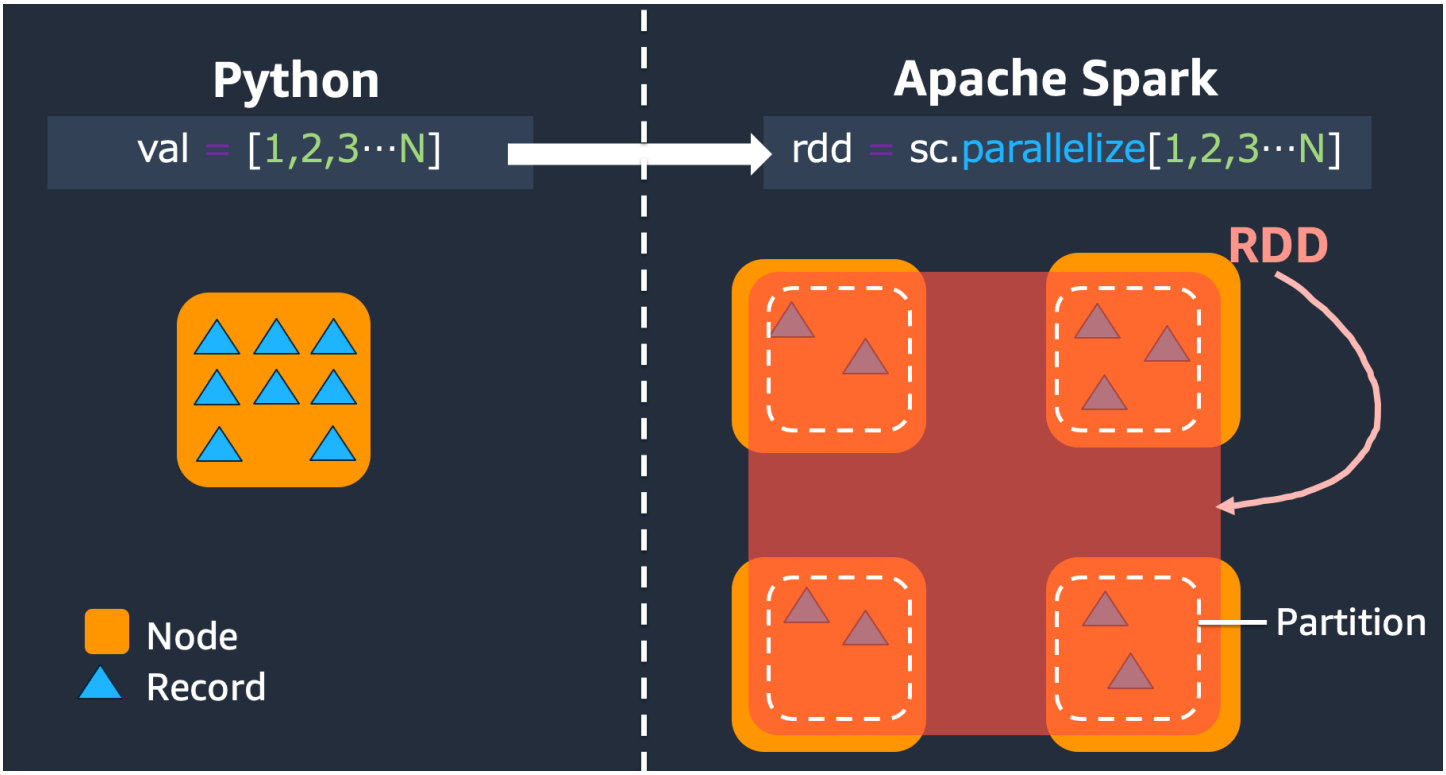


> **Note**
>
> Spark 실행기에는 여러 개의 슬롯이 있으므로 여러 작업을 병렬로 처리할 수 있습니다. Spark는 기본적으로 각 가상 CPU (vCPU) 코어에 대해 하나의 작업을 지원합니다. 예를 들어 실행기에 CPU 코어가 4개인 경우 4개의 작업을 동시에 실행할 수 있습니다.

복원력이 뛰어난 분산 데이터세트

Spark는 Spark 실행기 전반에서 대규모 데이터 세트를 저장하고 추적하는 복잡한 작업을 수행합니다. Spark 작업용 코드를 작성할 때는 스토리지의 세부 사항을 고려할 필요가 없습니다. Spark는 병렬로 작동할 수 있고 클러스터의 Spark 실행기 간에 분할될 수 있는 요소 모음인 복원력 있는 분산 데이터세트 (RDD) 추상화를 제공합니다.

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



- Python — Python 스크립트를 `val = [1,2,3...N]` 작성하면 코드가 실행되는 단일 시스템의 메모리에 데이터가 보관됩니다.

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

  대부분의 AWS Glue 작업에서는 RDD를 통해 Spark를 사용합니다. AWS Glue DynamicFramesDataFrames 이는 RDD의 데이터 스키마를 정의하고 해당 추가 정보를 사용하여 상위 수준의 작업을 수행할 수 있는 추상화입니다. 내부적으로 RDD를 사용하기 때문에 다음 코드에서는 데이터가 투명하게 분산되고 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 파티션이라는 여러 부분으로 나누어진 데이터로 구성됩니다. 각 Spark 실행기는 하나 이상의 파티션을 메모리에 저장하고 데이터는 여러 실행기에 분산됩니다.
- RDD는 변경할 수 없습니다. 즉, 생성된 후에는 변경할 수 없습니다. DataFramea를 변경하려면 다음 섹션에 정의된 변형을 사용할 수 있습니다.
- RDD는 사용 가능한 노드 전체에 데이터를 복제하므로 노드 장애 발생 시 자동으로 복구할 수 있습니다.

지연 평가

RDD는 두 가지 유형의 연산을 지원합니다. 하나는 기존 데이터셋에서 새 데이터세트를 생성하는 변환이고, 다른 하나는 데이터셋에서 계산을 실행한 후 드라이버 프로그램에 값을 반환하는 액션입니다.

- 변환 — RDD는 변경할 수 없으므로 변환을 통해서만 변경할 수 있습니다.

예를 들어, map 는 각 데이터셋 요소를 함수를 통해 전달하고 결과를 나타내는 새 RDD를 반환하는 변환입니다. 참고로 이 map 메서드는 출력값을 반환하지 않습니다. Spark는 사용자가 결과와 상호 작용할 수 있도록 하는 대신 미래를 위한 추상 변환을 저장합니다. Spark는 사용자가 작업을 호출할 때까지 변환에 영향을 주지 않습니다.

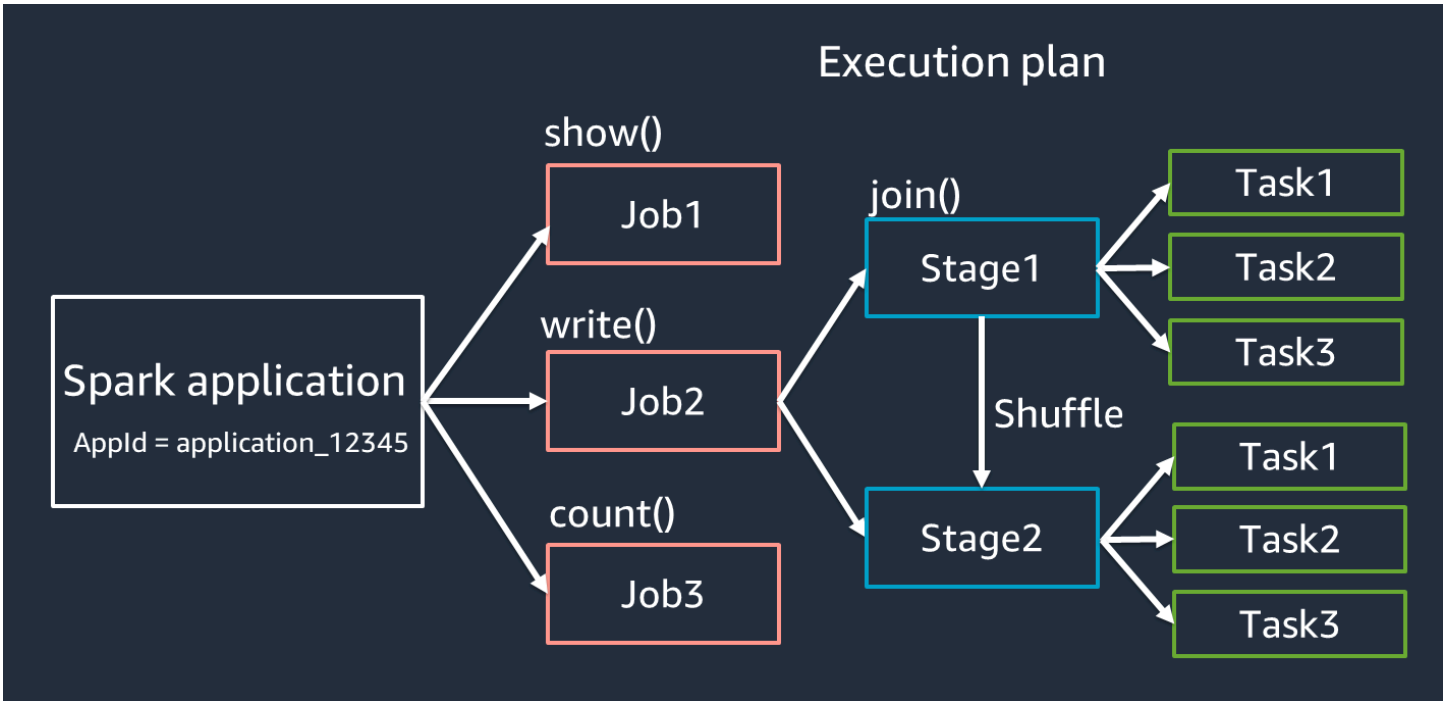
- 조치 — 변환을 사용하여 논리적 변환 계획을 수립할 수 있습니다. 계산을 시작하려면,, 또는 같은 write 작업을 실행합니다. count show collect

  Spark의 모든 변환은 결과를 즉시 계산하지 않는다는 점에서 느립니다. 대신 Spark는 Amazon Simple Storage Service (Amazon S3) 객체와 같은 일부 기본 데이터세트에 적용된 일련의 변환을 기억합니다. 변환은 작업 시 결과를 드라이버에 반환해야 하는 경우에만 계산됩니다. 이러한 설계 덕분에 Spark를 더 효율적으로 실행할 수 있습니다. 예를 들어, map 변환을 통해 생성된 데이터셋이 행 수를 크게 줄이는 변환에 의해서만 소비되는 상황을 생각해 보십시오. reduce 그러면 매핑된 더 큰 데이터셋을 전달하는 대신 두 번의 변환을 모두 거친 작은 데이터셋을 드라이버에 전달할 수 있습니다.

Spark 애플리케이션 용어

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

- 애플리케이션 — Spark 세션을 기반으로 합니다 (Spark 컨텍스트). 다음과 같은 고유한 ID로 식별됩니다. <application_XXX>
- 작업 — RDD에 대해 생성된 작업을 기반으로 합니다. 작업은 하나 이상의 단계로 구성됩니다.
- 스테이지 — RDD용으로 만든 셔플을 기반으로 합니다. 스테이지는 하나 이상의 작업으로 구성됩니다. 셔플은 데이터를 재분배하여 RDD 파티션 간에 서로 다르게 그룹화하는 Spark의 메커니즘입니다. 예를 들어 특정 변환에는 셔플이 필요합니다. join() 셔플은 셔플 최적화 조정 실습에서 더 자세히 설명합니다.
- 태스크 — 태스크는 Spark에서 스케줄링하는 최소 처리 단위입니다. 작업은 각 RDD 파티션에 대해 생성되며 작업 수는 스테이지의 최대 동시 실행 수입니다.

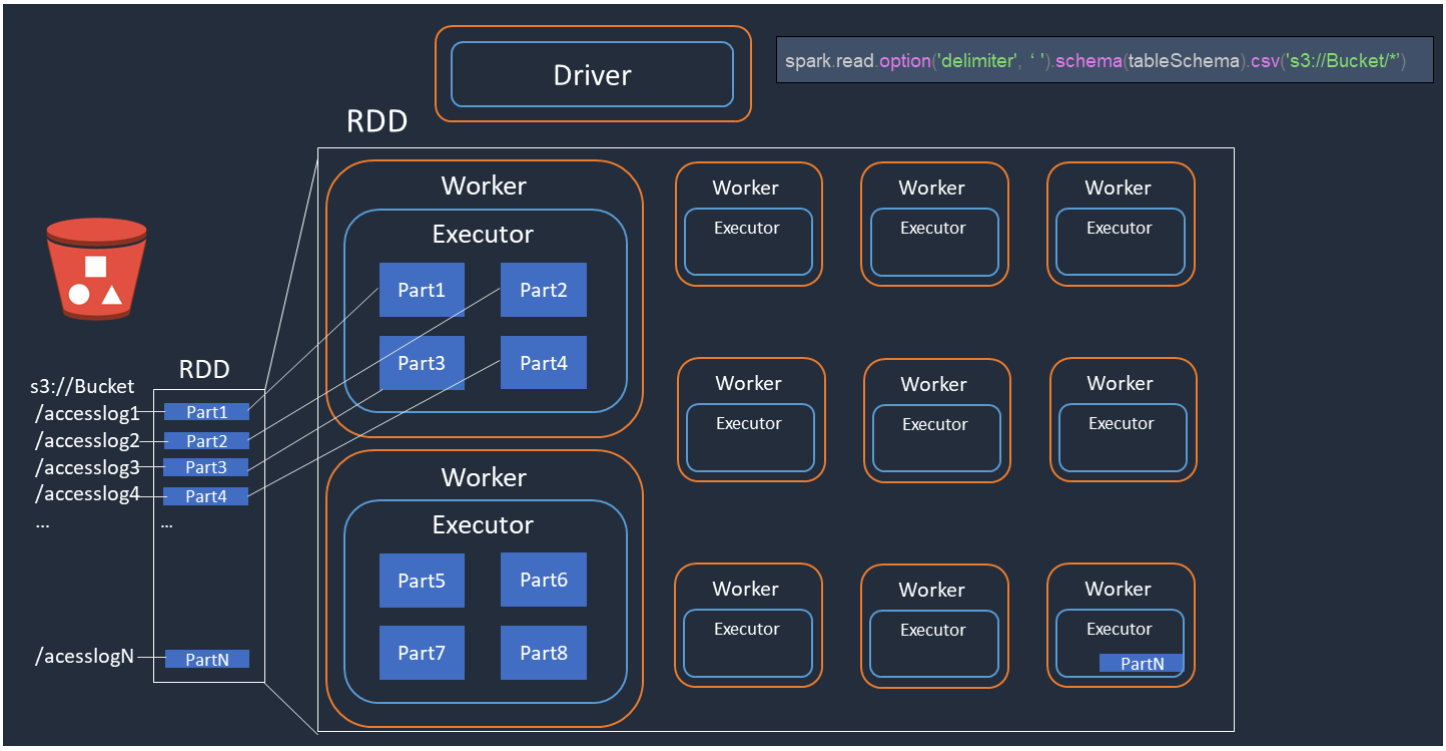


> **Note**
>
> 작업은 병렬성을 최적화할 때 고려해야 할 가장 중요한 사항입니다. 작업 수는 RDD 수에 따라 조정됩니다.

Parallelism

Spark는 데이터 로드 및 변환 작업을 병렬화합니다.

Amazon S3에서 액세스 로그 파일 (이름이 `accesslog1 ... accesslogN` 지정됨) 의 분산 처리를 수행하는 예를 생각해 보십시오. 다음 다이어그램은 분산 처리 흐름을 보여줍니다.

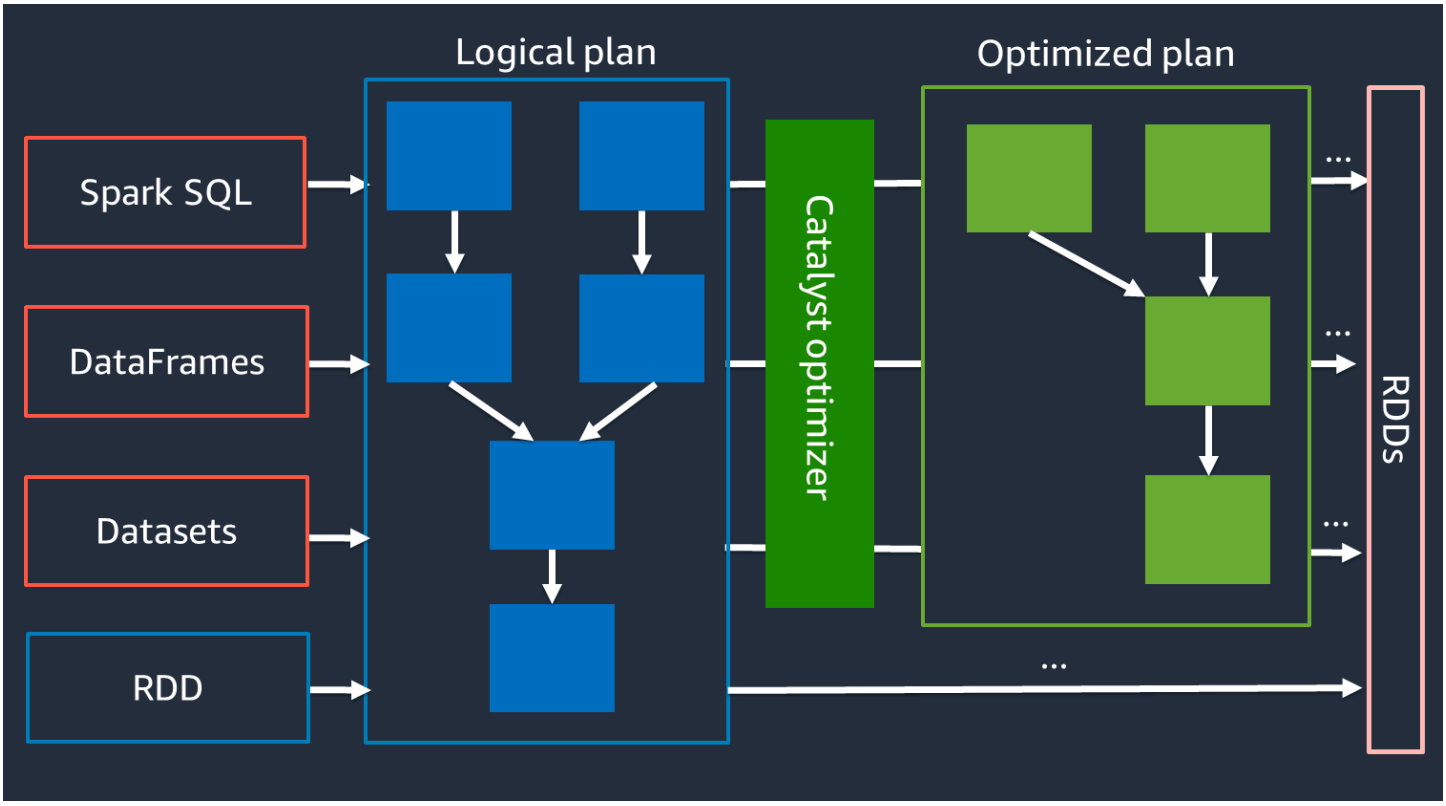


1. Spark 드라이버는 여러 Spark 실행기에 걸친 분산 처리를 위한 실행 계획을 생성합니다.
2. Spark 드라이버는 실행 계획을 기반으로 각 실행기에 작업을 할당합니다. 기본적으로 Spark 드라이버는 각 S3 객체 () 에 대해 RDD 파티션 (각각 Spark 작업에 해당) 을 생성합니다. `Part1 ... N` 그러면 Spark 드라이버가 각 실행기에 작업을 할당합니다.
3. 각 Spark 작업은 할당된 S3 객체를 다운로드하여 RDD 파티션의 메모리에 저장합니다. 이러한 방식으로 여러 Spark 실행기가 할당된 작업을 병렬로 다운로드하고 처리합니다.

초기 파티션 수 및 최적화에 대한 자세한 내용은 병렬화 작업 섹션을 참조하십시오.

카탈리스트 옵티마이저

Spark는 내부적으로 Catalyst 옵티마이저라는 엔진을 사용하여 실행 계획을 최적화합니다. Catalyst에는 다음 다이어그램에 설명된 대로 Spark SQL DataFrame, 데이터 세트와 같은 높은 수준의 Spark API를 실행할 때 사용할 수 있는 쿼리 최적화 프로그램이 있습니다.



Catalyst 최적화 프로그램은 RDD API와 직접 작동하지 않으므로 일반적으로 상위 수준 API가 하위 수준 RDD API보다 빠릅니다. 복잡한 조인의 경우 Catalyst 최적화 프로그램은 작업 실행 계획을 최적화하여 성능을 크게 향상시킬 수 있습니다. Spark UI의 SQL 탭에서 Spark 작업의 최적화된 계획을 확인할 수 있습니다.

적응형 쿼리 실행

Catalyst 최적화 프로그램은 적응형 쿼리 실행이라는 프로세스를 통해 런타임 최적화를 수행합니다. Adaptive Query Execution은 런타임 통계를 사용하여 작업이 실행되는 동안 쿼리의 실행 계획을 다시 최적화합니다. Adaptive Query Execution은 다음 섹션에 설명된 대로 셔플 이후 파티션 병합, 정렬-병합 조인을 브로드캐스트 조인으로 변환, 스큐 조인 최적화 등 성능 문제를 해결하는 여러 솔루션을 제공합니다.

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

셔플 이후 파티션 통합

이 기능은 출력 통계를 기반으로 각 셔플 이후 RDD 파티션 (병합) 을 줄입니다. `map` 쿼리를 실행할 때 셔플 파티션 번호를 간단하게 조정할 수 있습니다. 데이터세트에 맞게 셔플 파티션 번호를 설정할 필요

가 없습니다. 초기 셔플 파티션 수가 충분히 많으면 Spark는 런타임 시 적절한 셔플 파티션 번호를 선택할 수 있습니다.

둘 다 true로 설정된 경우 셔플 이후 파티션 병합이 활성화됩니다. `spark.sql.adaptive.enabled` `spark.sql.adaptive.coalescePartitions.enabled` 자세한 내용은 Apache Spark 설명서를 참조하십시오.

정렬-병합 조인을 브로드캐스트 조인으로 변환

이 기능은 크기가 크게 다른 두 데이터셋을 조인하는 경우를 인식하여 해당 정보를 기반으로 더 효율적인 조인 알고리즘을 채택합니다. 자세한 내용은 Apache Spark 설명서를 참조하십시오. 조인 전략은 셔플 최적화 섹션에서 설명합니다.

스큐 조인 최적화

데이터 스큐는 Spark 작업의 가장 일반적인 병목 현상 중 하나입니다. 데이터가 특정 RDD 파티션 (즉, 특정 작업) 에 치우쳐 애플리케이션의 전체 처리 시간이 지연되는 상황을 설명합니다. 이로 인해 종종 조인 작업의 성능이 저하될 수 있습니다. 스큐 조인 최적화 기능은 치우친 작업을 대략 같은 크기의 작업으로 분할 (필요한 경우 복제) 하여 정렬-병합 조인의 편차를 동적으로 처리합니다.

이 `spark.sql.adaptive.skewJoin.enabled` 기능은 true로 설정된 경우 활성화됩니다. 자세한 내용은 Apache Spark 설명서를 참조하십시오. 데이터 스큐는 셔플 최적화 섹션에서 자세히 설명합니다.

Spark UI를 사용하여 성능 문제를 조사하세요

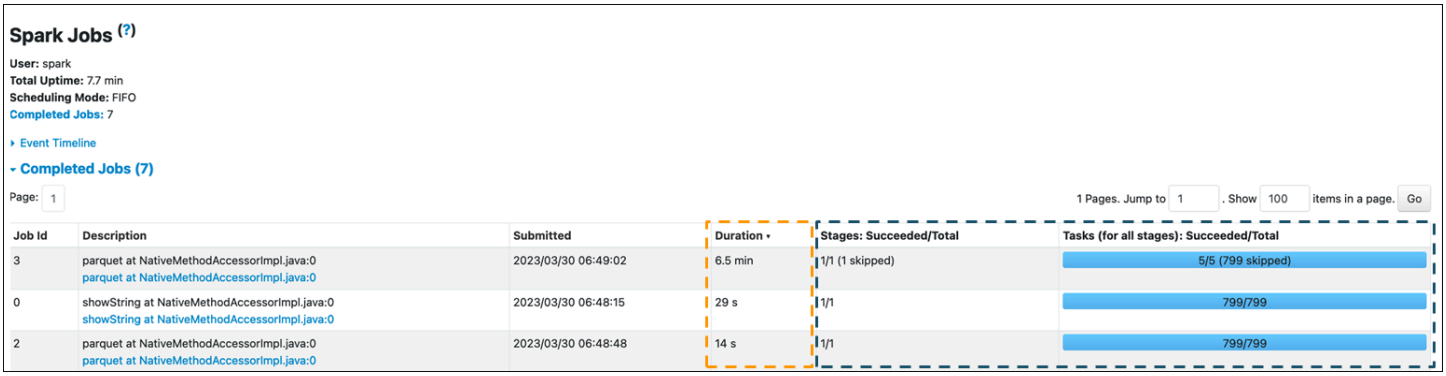
모범 사례를 적용하여 AWS Glue 작업 성능을 조정하기 전에 성능을 프로파일링하고 병목 현상을 식별하는 것이 좋습니다. 이렇게 하면 올바른 일에 집중하는 데 도움이 됩니다.

빠른 분석을 위해 Amazon CloudWatch 지표는 작업 지표의 기본 보기를 제공합니다. Spark UI는 성능 튜닝을 위한 심층적인 보기를 제공합니다. 에서 Spark UI를 사용하려면 작업에 AWS GlueSpark UI를 활성화해야 합니다. AWS Glue Spark UI에 익숙해지면 Spark 작업 성능 조정 전략을 따라 결과를 기반으로 병목 현상의 영향을 식별하고 줄이십시오.

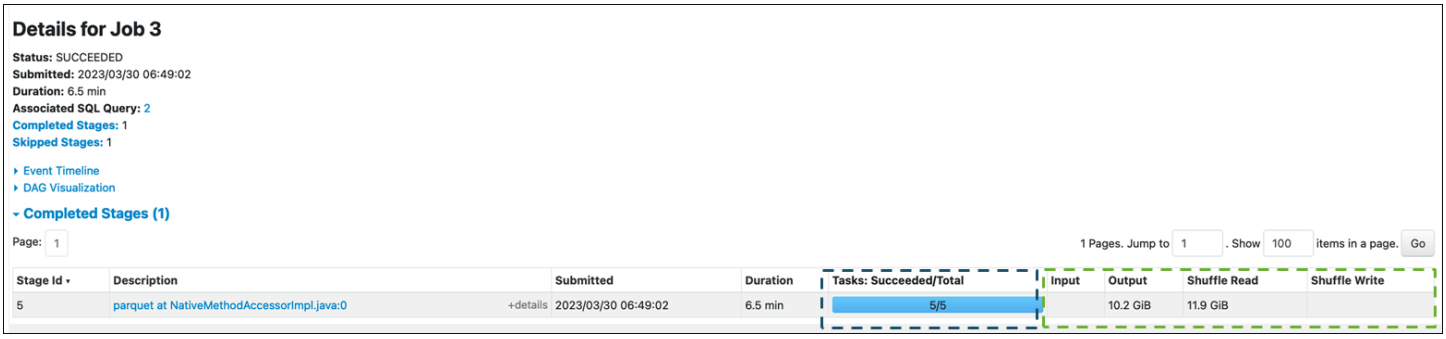
Spark UI를 사용하여 병목 현상을 식별하십시오.

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

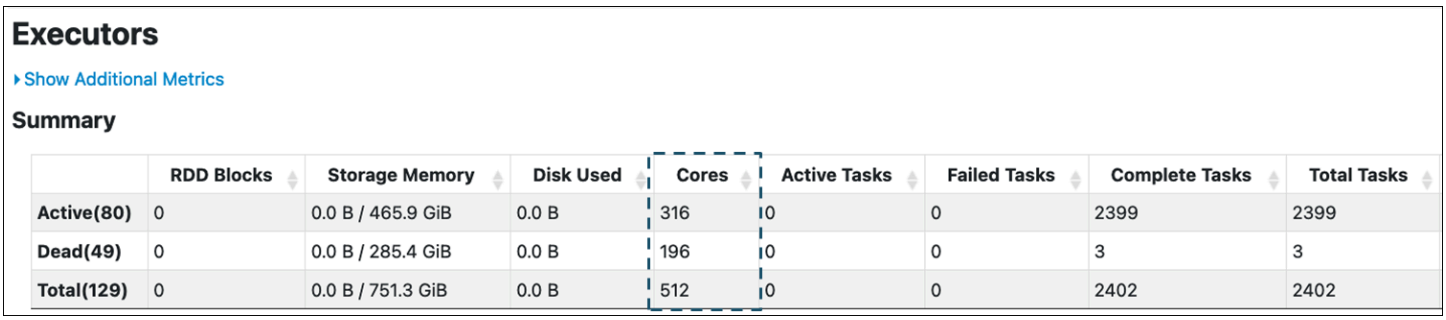
작업 탭에는 Spark 애플리케이션의 모든 작업에 대한 요약이 표시됩니다. 단계 또는 작업 실패를 확인하려면 총 작업 수를 확인하세요. 병목 현상을 찾으려면 기간을 선택하여 정렬하십시오. 설명 열에 표시된 링크를 선택하여 장기 실행 작업의 세부 정보를 자세히 살펴보세요.



Job 세부 정보 페이지에는 단계가 나열됩니다. 이 페이지에서는 기간, 성공한 작업 수와 총 작업 수, 입력 및 출력 수, 셔플 읽기 및 쓰기 셔플 양과 같은 전반적인 정보를 볼 수 있습니다.



Executor 탭에는 Spark 클러스터 용량이 자세히 표시됩니다. 총 코어 수를 확인할 수 있습니다. 다음 스크린샷에 표시된 클러스터에는 316개의 활성 코어와 총 512개의 코어가 포함되어 있습니다. 기본적으로 각 코어는 Spark 작업 하나를 동시에 처리할 수 있습니다.



[Job Details for Job] 페이지에 5/5 표시된 값을 기준으로 볼 때, 5단계는 가장 긴 단계이지만 512개 중 5개의 코어만 사용합니다. 이 단계의 병렬도는 매우 낮지만 시간이 많이 걸리기 때문에 병목 현상으로 식별할 수 있습니다. 성능을 개선하려면 그 이유를 이해해야 합니다. 일반적인 성능 병목 현상을 인식하고 그 영향을 줄이는 방법에 대해 자세히 알아보려면 Spark 작업 성능 조정 전략을 참조하십시오.

Spark 작업 성능을 조정하기 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 성과 목표를 결정하세요. 예를 들어, 목표 중 하나는 3시간 이내에 AWS Glue 작업 실행을 완료하는 것일 수 있습니다. 목표를 정의한 후에는 작업 성과 지표를 측정하세요. 지표의 추세와 병목 현상을 파악하여 목표를 달성하세요. 특히 병목 현상을 식별하는 것은 문제 해결, 디버깅 및 성능 튜닝에 가장 중요합니다. Spark 애플리케이션을 실행하는 동안 Spark는 Spark 이벤트 로그에 각 작업의 상태와 통계를 기록합니다.

AWS Glue에서는 Spark 기록 서버에서 제공하는 Spark 웹 UI를 통해 Spark 지표를 볼 수 있습니다. AWS Glue Spark의 경우 작업은 Amazon S3에서 지정한 위치로 Spark 이벤트 로그를 전송할 수 있습니다. AWS Glue 또한 Amazon EC2 인스턴스 또는 로컬 컴퓨터에서 Spark 기록 서버를 시작하기 위한 예제 AWS CloudFormation 템플릿과 Dockerfile을 제공하므로 이벤트 로그와 함께 Spark UI를 사용할 수 있습니다.

성능 목표를 결정하고 이러한 목표를 평가하는 지표를 식별한 후에는 다음 섹션의 전략을 사용하여 병목 현상을 식별하고 해결할 수 있습니다.

Spark 작업 성능을 위한 조정 사례

다음 전략을 사용하여 Spark 작업의 성능을 조정할 AWS Glue 수 있습니다.

- AWS Glue 리소스:
  - 클러스터 용량을 확장하세요.
  - 최신 버전 AWS Glue 사용
- 스파크 애플리케이션:
  - 데이터 스캔량을 줄이십시오.
  - 작업을 병렬화합니다.
  - 셔플 최적화
  - 계획 오버헤드를 최소화하세요.
  - 사용자 정의 함수 최적화

이러한 전략을 사용하기 전에 Spark 작업에 대한 지표 및 구성에 액세스할 수 있어야 합니다. AWS Glue 설명서에서 이 정보를 찾을 수 있습니다.

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

Apache Spark 애플리케이션 관점에서 보면 성능을 개선할 수 있는 몇 가지 전략에 액세스할 수 있습니다. 불필요한 데이터가 Spark 클러스터에 로드되는 경우 이를 제거하여 로드되는 데이터의 양을 줄일 수 있습니다. Spark 클러스터 리소스가 충분히 사용되지 않고 데이터 I/O가 낮은 경우 병렬화할 작업을 식별할 수 있습니다. 시간이 많이 걸리는 경우 조인과 같은 대용량 데이터 전송 작업을 최적화하는 것도 좋습니다. 또한 작업 쿼리 계획을 최적화하거나 개별 Spark 작업의 계산 복잡성을 줄일 수 있습니다.

이러한 전략을 효율적으로 적용하려면 메트릭을 참조하여 적용 가능한 시기를 식별해야 합니다. 자세한 내용은 다음 각 섹션을 참조하십시오. 이러한 기법은 성능 조정뿐 아니라 out-of-memory (OOM) 오류와 같은 일반적인 문제를 해결하는 데에도 효과적입니다.

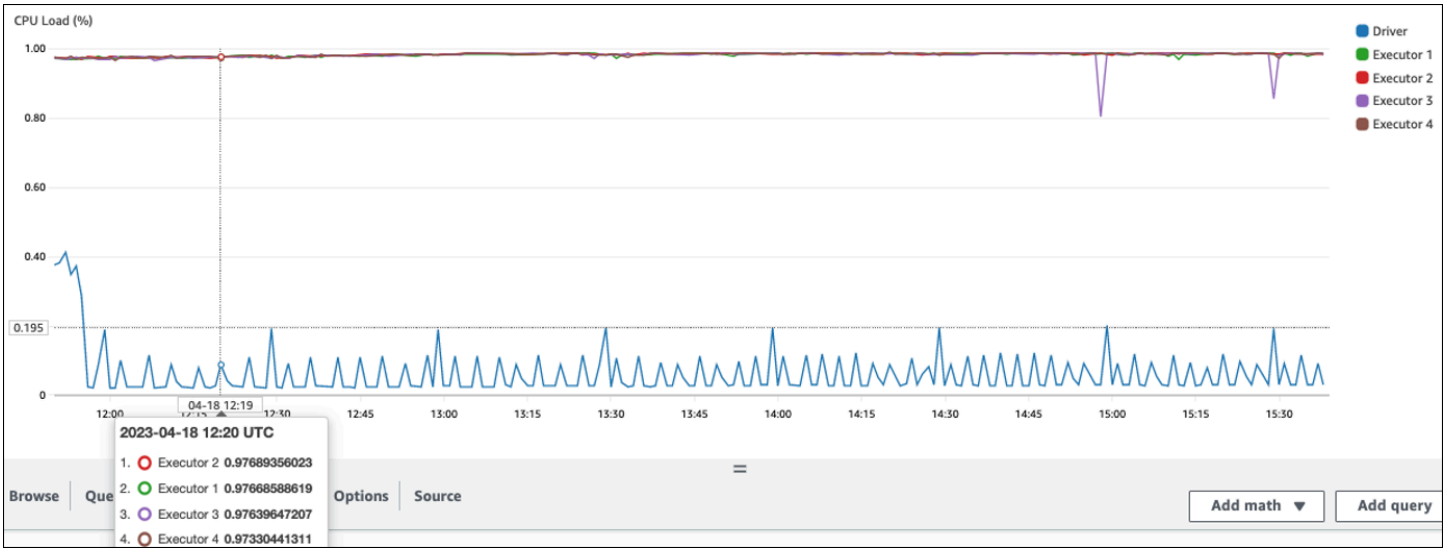
클러스터 용량을 확장하세요.

작업에 너무 많은 시간이 걸리지만 실행기가 충분한 리소스를 소비하고 있고 Spark가 사용 가능한 코어에 비해 대량의 작업을 생성하고 있다면 클러스터 용량을 확장하는 것을 고려해 보세요. 이것이 적절한지 평가하려면 다음 측정치를 사용하세요.

CloudWatch 지표

- CPU로드 및 메모리 사용률을 확인하여 실행기가 충분한 리소스를 소비하고 있는지 확인하세요.
- 처리 시간이 성능 목표를 달성하기에 너무 길지 않은지 확인하려면 작업이 얼마나 오래 실행되었는지 확인하세요.

다음 예제에서는 실행기 4개가 97% 이상의 CPU 부하로 실행되고 있지만 약 3시간이 지나도 처리가 완료되지 않았습니다.

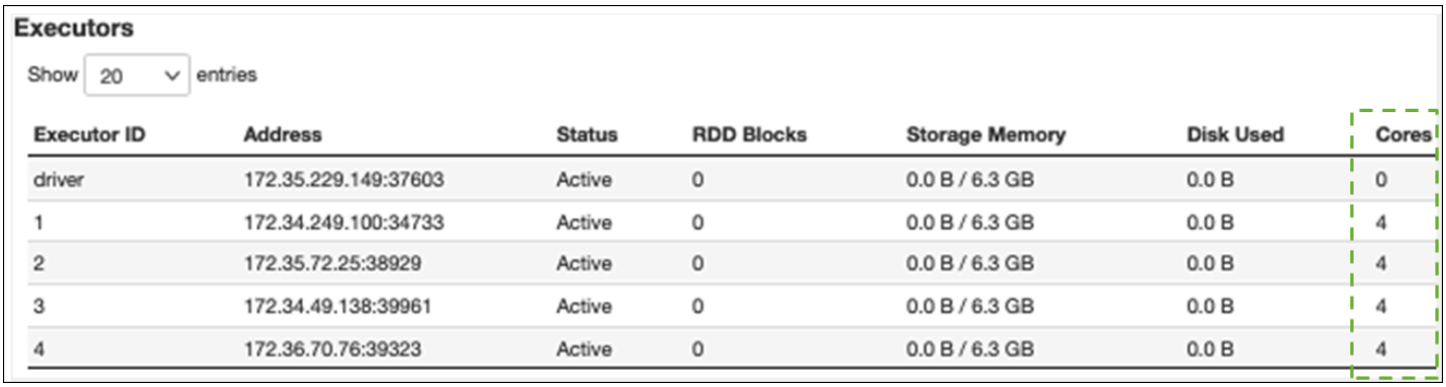


> **Note**
>
> CPU부하가 낮으면 클러스터 용량을 확장해도 효과가 없을 수 있습니다.

Spark UI

Job 탭 또는 Stage 탭에서 각 작업 또는 단계의 작업 수를 확인할 수 있습니다. 다음 예제에서 Spark는 `58100` 작업을 생성했습니다.

실행자 탭에서는 총 실행자 및 작업 수를 확인할 수 있습니다. 다음 스크린샷에서 각 Spark 실행기는 4개의 코어를 가지고 있으며 4개의 작업을 동시에 수행할 수 있습니다.



Executors

Show 20 entries

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 작업 () 의 수는 `58100`) 실행기가 동시에 처리할 수 있는 16개 작업 (실행기 4개 × 코어 4개) 보다 훨씬 많습니다.

이러한 증상이 나타나면 클러스터를 확장해 보세요. 다음 옵션을 사용하여 클러스터 용량을 확장할 수 있습니다.

- AWS Glue Auto Scaling 활성화 - AWS Glue 버전 3.0 이상에서 AWS Glue 추출, 변환, 로드 (ETL) 및 스트리밍 작업에 Auto Scaling을 사용할 수 있습니다. AWS Glue 각 단계의 파티션 수 또는 작업 실행 시 마이크로배치가 생성되는 속도에 따라 클러스터에서 작업자를 자동으로 추가하고 제거합니다.

  Auto Scaling을 활성화했는데도 작업자 수가 증가하지 않는 상황이 발생하면 작업자를 수동으로 추가하는 것을 고려해 보십시오. 하지만 한 스테이지를 수동으로 스케일링하면 이후 단계에서 많은 작업자가 유휴 상태가 되어 성능은 전혀 향상되지 않으면 비용이 더 많이 들 수 있다는 점에 유의하십시오.

  Auto Scaling을 활성화하면 실행자 지표에서 실행자 수를 확인할 수 있습니다. CloudWatch 다음 지표를 사용하여 Spark 애플리케이션의 실행기에 대한 수요를 모니터링하십시오.

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

  지표에 대한 자세한 내용은 Amazon CloudWatch 지표를 AWS Glue 사용한 모니터링을 참조하십시오.
- 스케일 아웃: 작업자 수 늘리기 — AWS Glue 작업자 수를 AWS Glue 수동으로 늘릴 수 있습니다. 유휴 근로자가 관찰될 때까지만 작업자를 추가하십시오. 이때 근로자를 더 추가하면 결과가 개선되지 않고 비용이 증가하게 됩니다. 자세한 내용은 작업 병렬화를 참조하십시오.

- 확장: 더 큰 작업자 유형 사용 — 더 많은 코어, 메모리 및 스토리지를 갖춘 AWS Glue 작업자를 사용하도록 작업자의 인스턴스 유형을 수동으로 변경할 수 있습니다. 작업자 유형이 크면 메모리를 많이 사용하는 데이터 변환, 치우친 집계, 페타바이트 규모의 데이터를 포함하는 개체 감지 검사 등 집약적인 데이터 통합 작업을 수직으로 확장하고 실행할 수 있습니다.

  확장은 작업 쿼리 계획이 상당히 크기 때문에 Spark 드라이버에 더 큰 용량이 필요한 경우에도 도움이 됩니다. 작업자 유형 및 성능에 대한 자세한 내용은 AWS 빅 데이터 블로그 게시물 더 큰 새 작업자 유형 G.4X 및 AWS Glue G.8X를 사용하여 Apache Spark 작업을 확장하기를 참조하십시오.

  또한 대규모 작업자를 사용하면 필요한 총 작업자 수를 줄일 수 있으며, 조인과 같이 집중적인 작업에서 셔플을 줄여 성능을 높일 수 있습니다.

최신 버전 AWS Glue 사용

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

- 새롭게 최적화된 Apache Spark 3.3.0 런타임 — AWS Glue 4.0은 Apache Spark 3.3.0 런타임을 기반으로 빌드되어 오픈 소스 Spark와 비슷한 수준의 성능 향상을 제공합니다. Spark 3.3.0 런타임은 Spark 2.x의 다양한 혁신 기능을 기반으로 합니다.
- 향상된 Amazon Redshift 커넥터 — AWS Glue 4.0 이상 버전은 아파치 스파크를 위한 Amazon Redshift 통합을 제공합니다. 통합은 기존 오픈 소스 커넥터를 기반으로 하며 성능 및 보안을 위해 커넥터를 강화합니다. 통합을 통해 애플리케이션이 최대 10배 더 빠르게 작동할 수 있습니다. 자세한 내용은 Amazon Redshift와 Apache Spark의 통합에 대한 블로그 게시물을 참조하십시오.
- SIMDCSV및 JSON 데이터를 포함한 벡터화된 읽기에 대한 기반 실행 — AWS Glue 버전 3.0 이상 버전에는 행 기반 판독기에 비해 전반적인 작업 성능을 크게 향상시킬 수 있는 최적화된 판독기가 추가되었습니다. CSV데이터에 대한 자세한 내용은 벡터화된 판독기를 사용한 읽기 성능 최적화를 참조하십시오. SIMD CSV JSON데이터에 대한 자세한 내용은 Apache Arrow 열 형식 형식의 벡터화된 SIMD JSON 판독기 사용을 참조하십시오.

각 AWS Glue 버전에는 커넥터, 드라이버 및 라이브러리 업데이트를 비롯한 많은 업그레이드 중에서 이러한 종류의 업그레이드가 포함됩니다. 자세한 내용은 AWS Glue 버전 및 AWS Glue 버전 4.0으로의 AWS Glue 작업 마이그레이션을 참조하십시오.

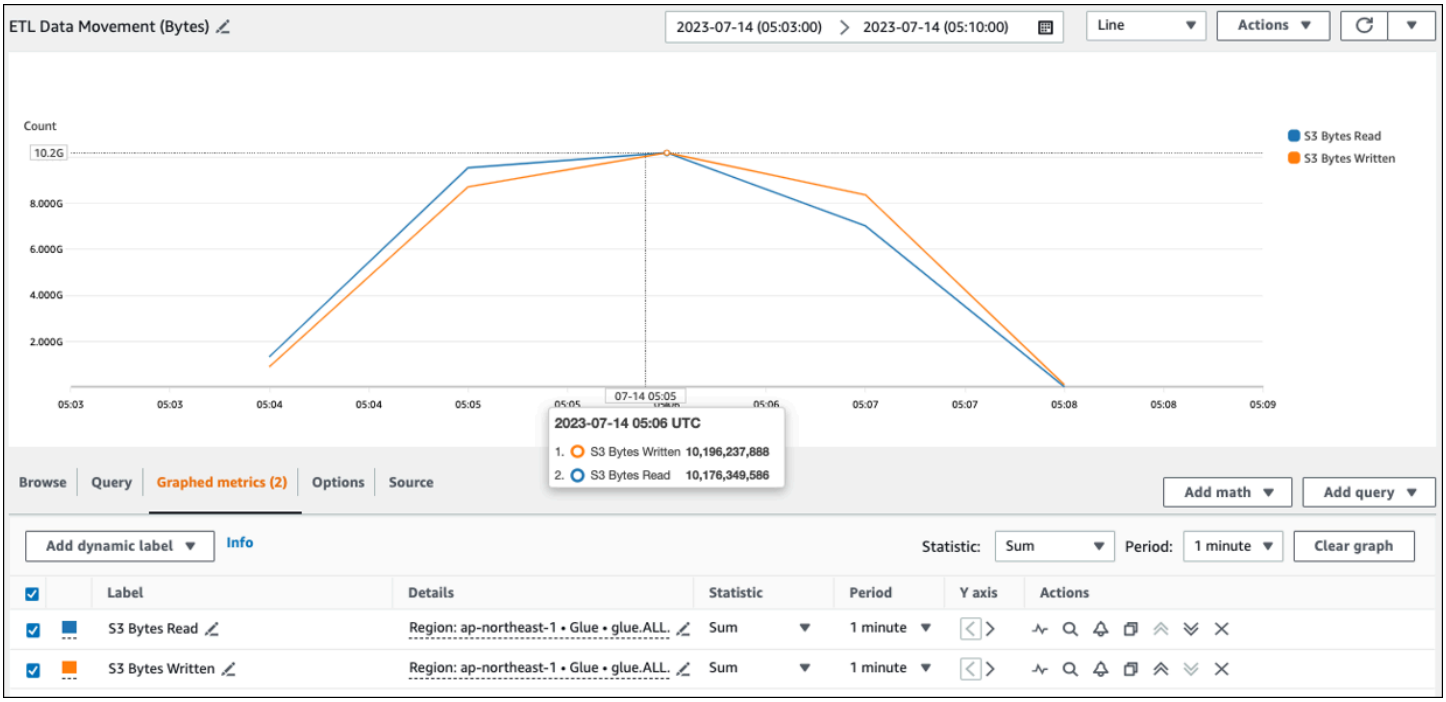
데이터 스캔량을 줄이십시오.

시작하려면 필요한 데이터만 로드하는 것이 좋습니다. 각 데이터 소스에 대해 Spark 클러스터에 로드되는 데이터의 양을 줄이는 것만으로도 성능을 개선할 수 있습니다. 이 접근 방식이 적절한지 평가하려면 다음 측정치를 사용하세요.

Spark UI 섹션에 설명된 대로 CloudWatch지표에서 Amazon S3의 읽은 바이트를 확인하고 Spark UI에서 자세한 내용을 확인할 수 있습니다.

CloudWatch 메트릭스

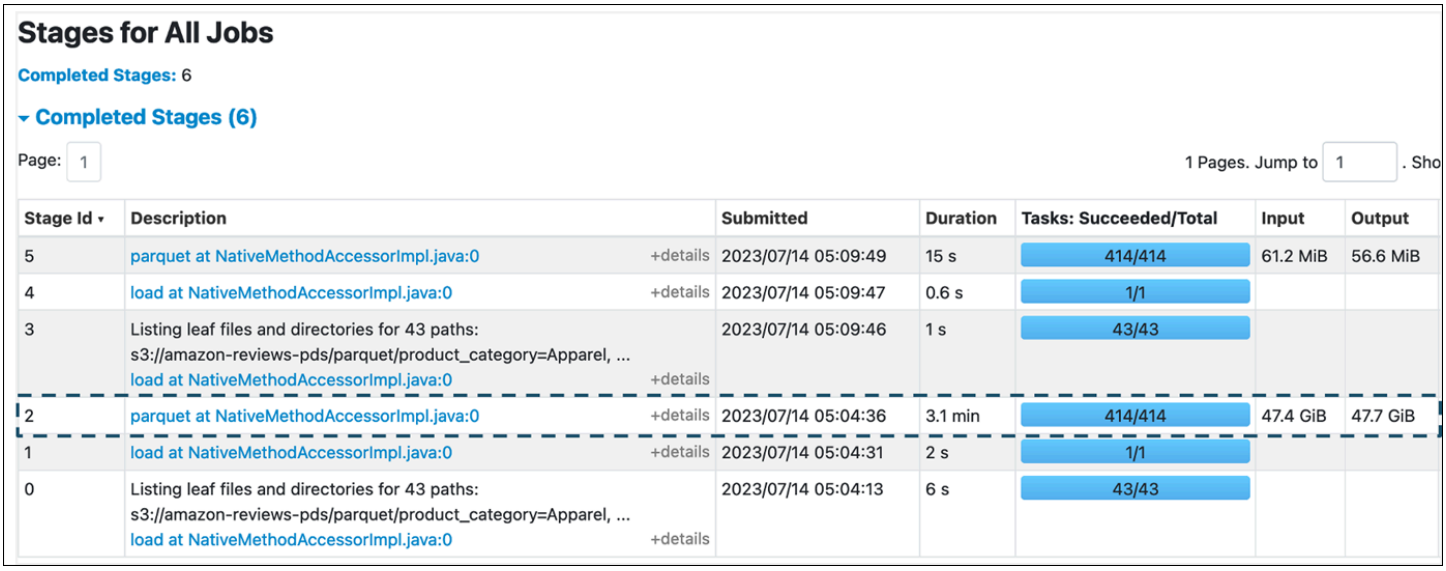
Amazon S3의 대략적인 읽기 크기는 ETL데이터 이동 (바이트) 에서 확인할 수 있습니다. 이 지표는 이전 보고서 이후 모든 실행자가 Amazon S3에서 읽은 바이트 수를 보여줍니다. 이를 사용하여 Amazon S3의 ETL 데이터 이동을 모니터링하고 읽기와 외부 데이터 소스의 수집 속도를 비교할 수 있습니다.



S3바이트 읽기 데이터 포인트가 예상보다 큰 경우 다음 해결 방법을 고려해 보십시오.

Spark UI

Spark UI의 AWS Glue 스테이지 탭에서 입력 및 출력 크기를 확인할 수 있습니다. 다음 예제에서 스테이지 2는 47.4GiB 입력과 47.7GiB 출력을 읽고, 스테이지 5에서는 61.2MiB 입력과 56.6MiB 출력을 읽습니다.



Stages for All Jobs

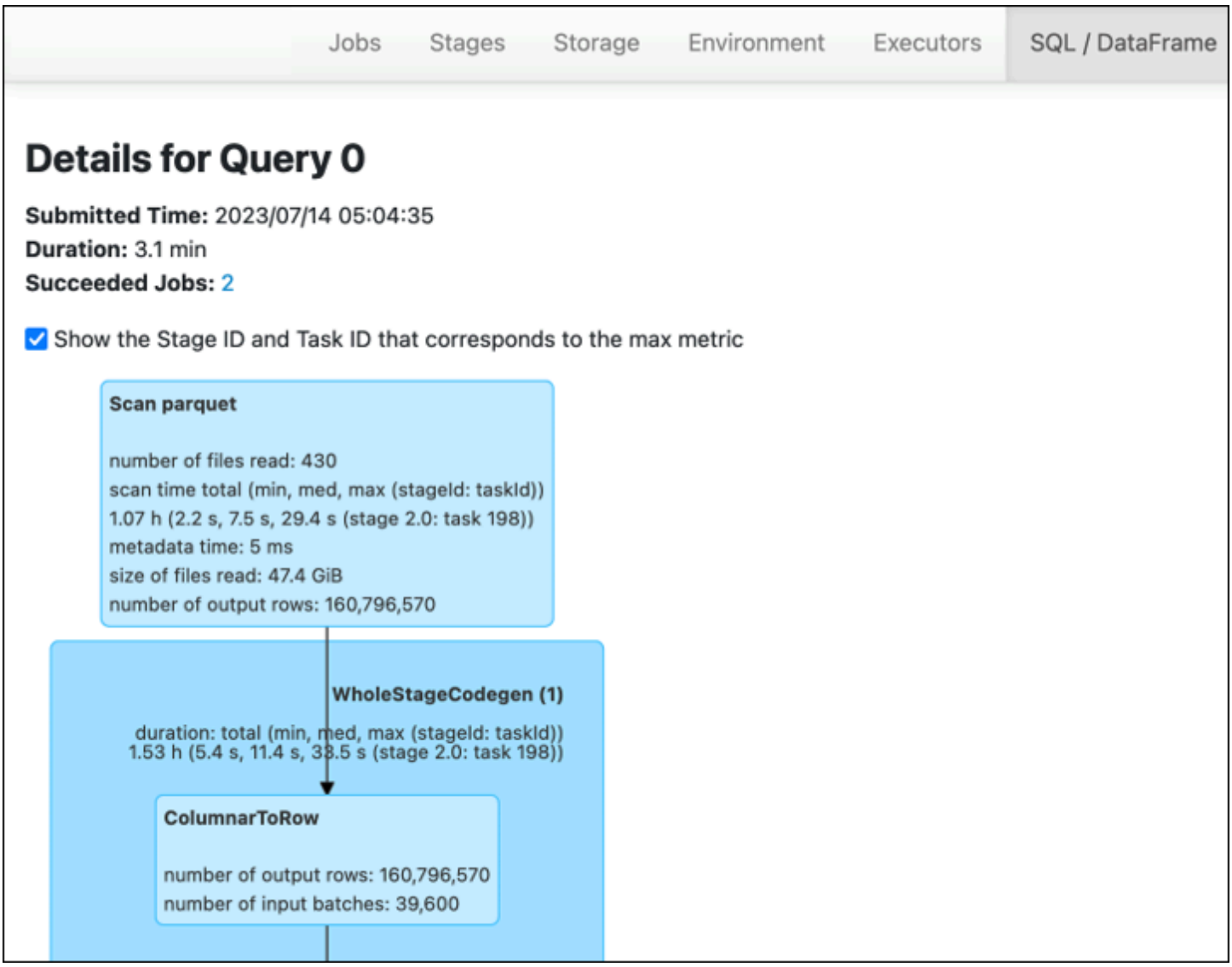
Completed Stages: 6

Completed Stages (6)

Page: 1 1 Pages. Jump to 1 . Sho

Stage Id	Description	Submitted	Duration	Tasks: Succeeded/Total	Input	Output
5	parquet at NativeMethodAccessorImpl.java:0 +details	2023/07/14 05:09:49	15 s	414/414	61.2 MiB	56.6 MiB
4	load at NativeMethodAccessorImpl.java:0 +details	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, ... load at NativeMethodAccessorImpl.java:0 +details	2023/07/14 05:09:46	1 s	43/43		
2	parquet at NativeMethodAccessorImpl.java:0 +details	2023/07/14 05:04:36	3.1 min	414/414	47.4 GiB	47.7 GiB
1	load at NativeMethodAccessorImpl.java:0 +details	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, ... load at NativeMethodAccessorImpl.java:0 +details	2023/07/14 05:04:13	6 s	43/43		

작업에서 Spark SQL 또는 DataFrame 접근 방식을 사용하는 경우 /D 탭에 이러한 단계에 대한 추가 통계가 표시됩니다. AWS Glue SQL ataFrame 이 경우 2단계에서는 읽은 파일 수: 430, 읽은 파일 크기: 47.4 GiB, 출력 행 수: 160,796,570을 보여줍니다.



읽고 있는 데이터와 사용 중인 데이터 간에 크기가 크게 차이가 나는 경우 다음 해결 방법을 시도해 보십시오.

Amazon S3

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

- 파일 크기 — 입력과 출력의 파일 크기를 적당한 범위 (예: 128MB) 이내로 유지하는 것이 좋습니다. 너무 작은 파일과 너무 큰 파일은 문제를 일으킬 수 있습니다.

  작은 파일이 많으면 다음과 같은 문제가 발생합니다.

- 동일한 양의 데이터를 저장하는 소수의 객체에 비해 많은 객체를 요청 (예: `ListGet`, 또는`Head`) 하려면 오버헤드가 필요하기 때문에 Amazon S3에서 네트워크 I/O 부하가 심합니다.
- Spark 드라이버에 I/O 및 처리 부하가 심하면 파티션과 작업이 많이 생성되고 병렬 처리가 과도하게 발생합니다.

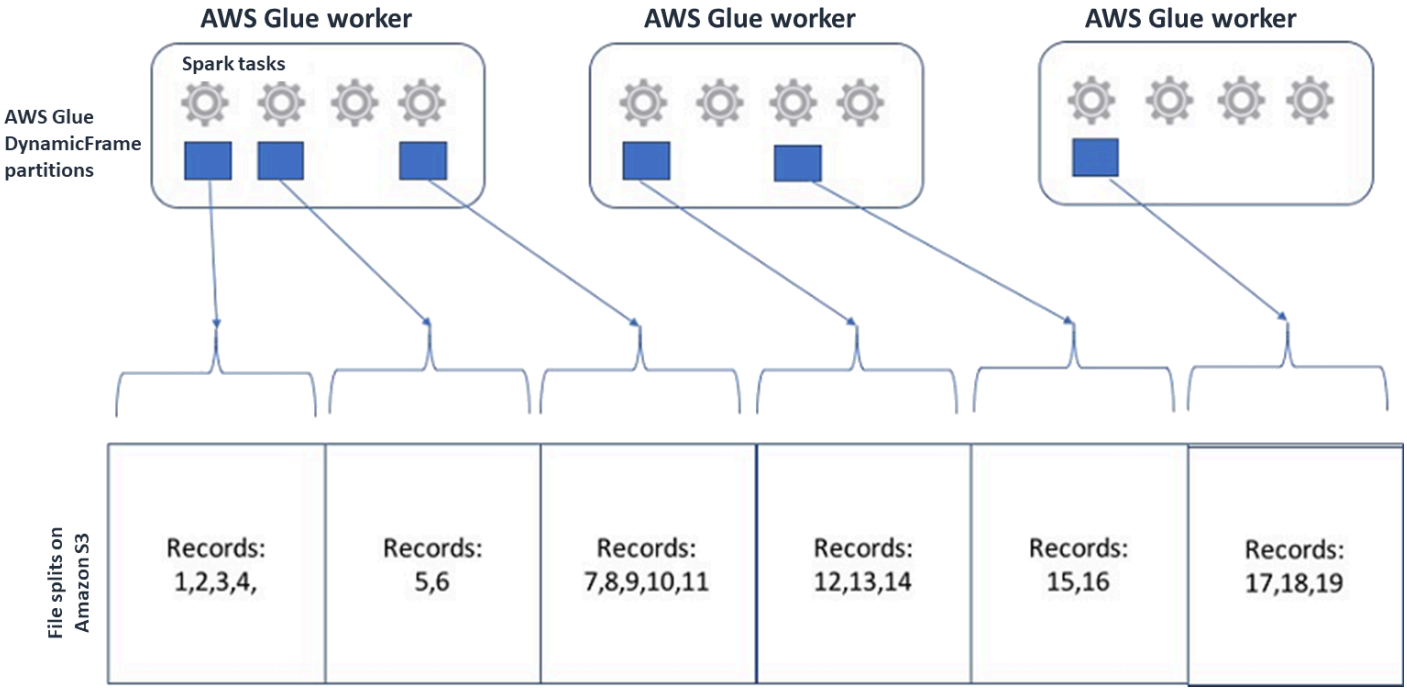
반면, 파일 유형이 분할할 수 없고 (예: gzip) 파일이 너무 크면 Spark 응용 프로그램은 단일 작업으로 전체 파일 읽기가 완료될 때까지 기다려야 합니다.

작은 파일마다 Apache Spark 태스크를 생성할 때 발생하는 과도한 병렬 처리를 줄이려면 파일 그룹화를 사용하십시오. DynamicFrames 이 방법을 사용하면 Spark 드라이버에서 예외가 발생할 가능성이 줄어듭니다. OOM 파일 그룹화를 구성하려면 `groupFiles` 및 `groupSize` 매개변수를 설정합니다. 다음 코드 예제에서는 ETL 스크립트에서 이러한 매개 변수와 함께 를 사용합니다. AWS Glue DynamicFrame API

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

- 압축 - S3 객체의 크기가 수백 메가바이트인 경우 압축을 고려해 보십시오. 압축 형식은 다양하며 크게 두 가지 유형으로 분류할 수 있습니다.
  - gzip과 같이 분할할 수 없는 압축 형식의 경우 작업자 한 명이 전체 파일의 압축을 풀어야 합니다.
  - bzip2 또는 LZO (인덱싱됨) 과 같은 분할 가능한 압축 형식을 사용하면 파일을 부분적으로 압축 해제하여 병렬화할 수 있습니다.

  Spark (및 기타 일반적인 분산 처리 엔진) 의 경우 소스 데이터 파일을 엔진이 병렬로 처리할 수 있는 청크로 분할합니다. 이러한 유닛을 흔히 스플릿이라고 합니다. 데이터가 분할 가능한 형식이 되면 최적화된 AWS Glue 리더는 에 특정 블록만 검색하는 `Range` 옵션을 제공하여 S3 객체에서 `GetObject` API 분할을 검색할 수 있습니다. 이 방법이 실제로 어떻게 작동하는지 보려면 다음 다이어그램을 살펴보십시오.



파일 크기가 최적이거나 파일을 분할할 수 있는 한 압축된 데이터를 사용하면 애플리케이션 속도를 크게 높일 수 있습니다. 데이터 크기가 작을수록 Amazon S3에서 스캔한 데이터와 Amazon S3에서 Spark 클러스터로 전달되는 네트워크 트래픽이 줄어듭니다. 반면, 데이터를 압축하고 압축 해제하려면 더 많은 CPU 작업이 필요합니다. 필요한 컴퓨팅 양은 압축 알고리즘의 압축률에 따라 조정됩니다. 분할 가능한 압축 형식을 선택할 때는 이 절충점을 고려하십시오.

> **Note**
>
> gzip 파일은 일반적으로 분할할 수 없지만 gzip으로 개별 파켓 블록을 압축하여 해당 블록을 병렬화할 수 있습니다.

- 파일 형식 — 열 형식을 사용합니다. 아파치 파켓과 아파치는 널리 사용되는 컬럼형 데이터 ORC 형식입니다. 열 기반 압축을 사용하고 데이터 유형에 따라 각 열을 인코딩 및 압축하여 데이터를 효율적으로 파켓하고 ORC 저장합니다. Parquet 인코딩에 대한 자세한 내용은 Parquet 인코딩 정의를 참조하십시오. 파켓 파일도 분할할 수 있습니다.

  컬럼 형식은 값을 열별로 그룹화하고 블록으로 함께 저장합니다. 열 형식 사용 시 사용하지 않을 열에 해당하는 데이터 블록을 건너뛸 수 있습니다. Spark 애플리케이션은 필요한 열만 검색할 수 있습니다. 일반적으로 압축률이 높거나 데이터 블록을 건너뛰면 Amazon S3에서 읽는 바이트 수가 줄어들어 성능이 향상됩니다. 또한 두 형식 모두 I/O를 줄이기 위한 다음과 같은 푸시다운 접근 방식을 지원합니다.

- 프로젝션 푸시다운 — 프로젝션 푸시다운은 애플리케이션에 지정된 열만 검색하는 기법입니다. 다음 예와 같이 Spark 애플리케이션에서 열을 지정합니다.
  - DataFrame 예: `df.select("star_rating")`
  - 스파크 SQL 예시: `spark.sql("select start_rating from <table>")`
- 프리디케이트 푸시다운 — 프리디케이트 푸시다운은 AND 조항을 효율적으로 처리하기 위한 기법입니다. `WHERE GROUP BY` 두 형식 모두 열 값을 나타내는 데이터 블록을 포함합니다. 각 블록에는 최대값 및 최소값과 같은 블록에 대한 통계가 들어 있습니다. Spark는 이러한 통계를 사용하여 애플리케이션에서 사용되는 필터 값에 따라 블록을 읽을지 아니면 건너뛰어야 하는지를 결정할 수 있습니다. 이 기능을 사용하려면 다음 예와 같이 조건에 필터를 더 추가하세요.
  - DataFrame 예: `df.select("star_rating").filter("star_rating < 2")`
  - 스파크 SQL 예시: `spark.sql("select * from <table> where star_rating < 2")`
- 파일 레이아웃 — 데이터 사용 방식에 따라 서로 다른 경로의 객체에 S3 데이터를 저장하면 관련 데이터를 효율적으로 검색할 수 있습니다. 자세한 내용은 Amazon S3 설명서의 접두사를 사용한 객체 구성을 참조하십시오. AWS Glue Amazon S3 경로를 기준으로 데이터를 분할하여 키와 값을 Amazon S3 접두사에 `key=value` 같은 형식으로 저장할 수 있습니다. 데이터를 분할하면 각 다운스트림 분석 애플리케이션에서 스캔하는 데이터의 양을 제한하여 성능을 개선하고 비용을 절감할 수 있습니다. 자세한 내용은 출력을 위한 ETL 파티션 관리를 참조하십시오. AWS Glue

  파티셔닝은 테이블을 여러 부분으로 나누고 다음 예와 같이 연도, 월, 일과 같은 열 값을 기준으로 그룹화된 파일에 관련 데이터를 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
...
```

  의 테이블을 사용하여 데이터세트를 모델링하여 데이터셋의 파티션을 정의할 수 있습니다. AWS Glue Data Catalog그런 다음 다음과 같이 파티션 프루닝을 사용하여 데이터 스캔량을 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'",
)
```

- DataFrameSpark의 경우 파티션을 프루닝할 고정 경로를 설정합니다.

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

- Spark의 SQL 경우 where 절을 설정하여 데이터 카탈로그에서 파티션을 정리할 수 있습니다.

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

- 를 사용하여 데이터를 쓸 때 날짜별로 파티션을 AWS Glue나누려면 다음과 DataFrame 같이 열에 날짜 정보를 포함하여 DynamicFrame or partitionBy() 를 설정합니다 partitionKeys.
  - 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 데이터 소비자의 성능이 향상될 수 있습니다.

입력 데이터세트를 생성하는 파이프라인을 변경할 수 있는 권한이 없는 경우 파티셔닝은 옵션이 아닙니다. 대신 글로브 패턴을 사용하여 불필요한 S3 경로를 제외할 수 있습니다. 가져올 때 제외를 설정하십시오. DynamicFrame 예를 들어, 다음 코드는 2023년에 01~09개월의 날짜를 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'
)
```

데이터 카탈로그의 테이블 속성에서 제외를 설정할 수도 있습니다.

- 키: `exclusions`
- 값: `["**year=2023/month=0[1-9]/**"]`

- 너무 많은 Amazon S3 파티션 — 수천 개의 값이 있는 ID 열과 같이 광범위한 값을 포함하는 열로 Amazon S3 데이터를 분할하지 마십시오. 가능한 파티션 수는 파티션을 나눈 모든 필드를 곱하기 때문에 이렇게 하면 버킷의 파티션 수가 크게 늘어날 수 있습니다. 파티션이 너무 많으면 다음과 같은 문제가 발생할 수 있습니다.
  - 데이터 카탈로그에서 파티션 메타데이터를 검색하는 데 걸리는 지연 시간 증가
  - 더 많은 Amazon S3 API 요청 (`ListGet`, 및`Head`) 이 필요한 소형 파일 수 증가

  예를 들어`partitionKeys`, `partitionBy` 또는 에서 날짜 유형을 설정하는 경우 날짜 수준 `yyyy/mm/dd` 파티셔닝은 많은 사용 사례에 적합합니다. 하지만 너무 많은 파티션이 생성되어 전체적으로 성능에 부정적인 영향을 미칠 `yyyy/mm/dd/<ID>` 수 있습니다.

  반면 실시간 처리 애플리케이션과 같은 일부 사용 사례에는 다음과 같은 파티션이 많이 필요합니다`yyyy/mm/dd/hh`. 사용 사례에 대규모 파티션이 필요한 경우 AWS Glue 파티션 인덱스를 사용하여 데이터 카탈로그에서 파티션 메타데이터를 검색하는 데 걸리는 지연 시간을 줄이는 것이 좋습니다.

데이터베이스 및 JDBC

데이터베이스에서 정보를 검색할 때 데이터 스캔을 줄이려면 쿼리에 `where` 술어 (또는 절) 를 지정할 수 있습니다. SQL SQL인터페이스를 제공하지 않는 데이터베이스는 쿼리 또는 필터링을 위한 자체 메커니즘을 제공합니다.

Java 데이터베이스 연결 (JDBC) 연결을 사용하는 경우 다음 매개 변수에 대한 `where` 절과 함께 선택 쿼리를 제공하십시오.

- 의 DynamicFrame 경우 sampleQuery옵션을 사용하십시오. 를 사용하는 `create_dynamic_frame.from_catalog` 경우 다음과 같이 `additional_options` 인수를 구성하십시오.

```
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_options경우 다음과 같이 connection_options 인수를 구성하십시오.

```
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
    }
)
```

- 의 DataFrame 경우 쿼리 옵션을 사용하십시오.

```
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 AWS Glue 4.0 이상을 사용하여 Amazon Redshift Spark 커넥터의 푸시다운 지원을 활용하십시오.

```
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 데이터베이스의 설명서를 참조하십시오.

AWS Glue 옵션

- 모든 연속 작업 실행을 전체 스캔하지 않고 마지막 작업 실행 중에 없었던 데이터만 처리하려면 작업 북마크를 활성화하십시오.
- 처리할 입력 데이터의 양을 제한하려면 작업 북마크를 사용하여 제한된 실행을 활성화하십시오. 이렇게 하면 각 작업 실행에서 스캔한 데이터의 양을 줄이는 데 도움이 됩니다.

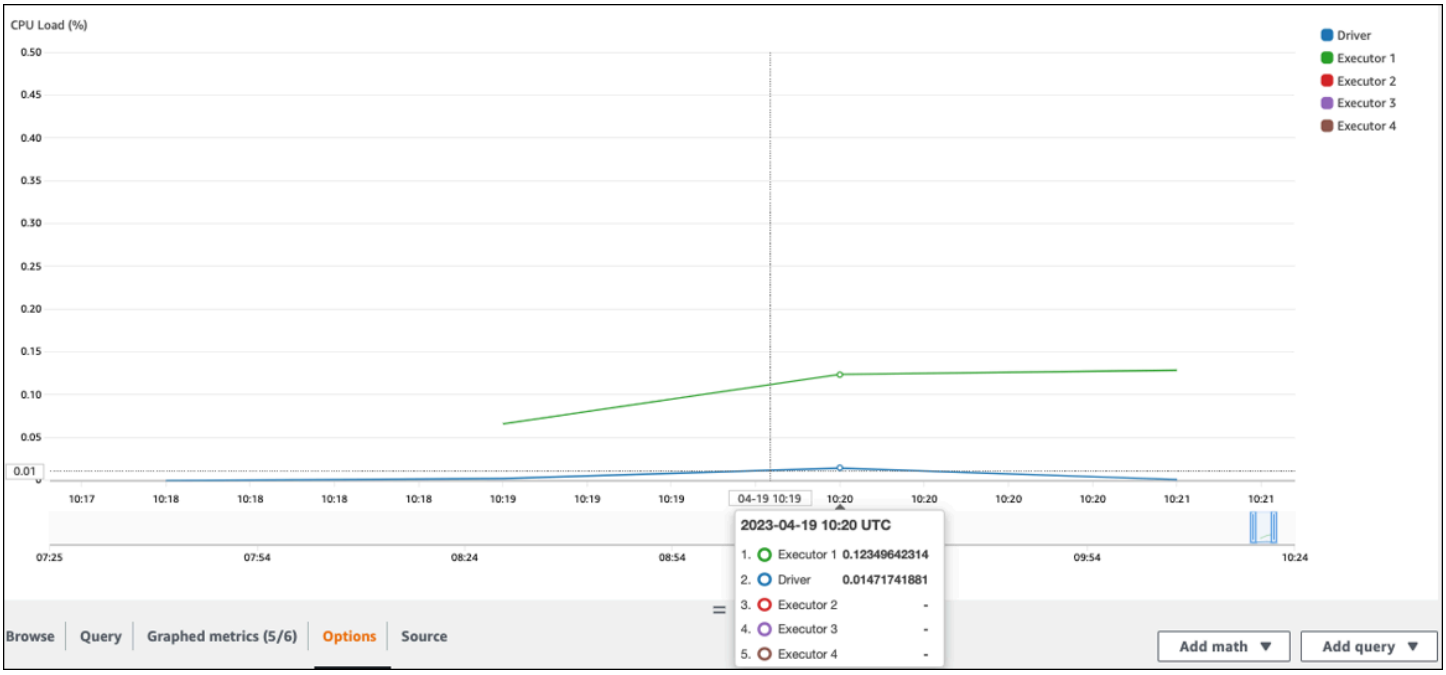
작업을 병렬화합니다.

성능을 최적화하려면 데이터 로드 및 변환 작업을 병렬화하는 것이 중요합니다. Apache Spark의 주요 항목에서 설명했듯이 복원력이 있는 분산 데이터 세트 (RDD) 파티션의 수는 병렬 처리 정도를 결정하기 때문에 중요합니다. Spark에서 생성하는 각 작업은 1:1 기준으로 파티션에 해당합니다. RDD 최상의 성능을 얻으려면 파티션 수를 결정하는 방법과 RDD 파티션 수를 최적화하는 방법을 이해해야 합니다.

병렬 처리가 충분하지 않은 경우 다음과 같은 현상이 CloudWatch메트릭과 Spark UI에 기록됩니다.

CloudWatch 메트릭스

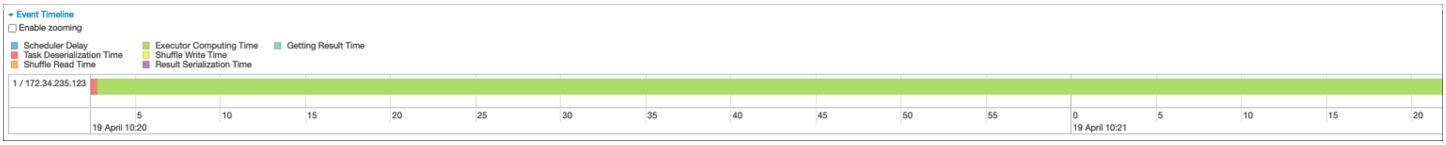
CPU로드 및 메모리 사용률을 확인하세요. 일부 실행기가 작업 단계에서 처리를 하지 않는 경우 병렬 처리를 개선하는 것이 좋습니다. 이 경우 시각화된 기간 동안 실행자 1은 작업을 수행했지만 나머지 실행자 (2, 3, 4) 는 수행하지 않았습니다. Spark 드라이버가 해당 실행기에 작업을 할당하지 않았음을 유추할 수 있습니다.



Spark UI

Spark UI의 스테이지 탭에서 스테이지의 작업 수를 확인할 수 있습니다. 이 경우 Spark는 단 하나의 작업만 수행했습니다.

또한 이벤트 타임라인에는 Executor 1이 하나의 작업을 처리하는 모습이 표시됩니다. 즉, 이 단계의 작업은 한 실행자에서만 수행되고 다른 실행기는 유휴 상태였습니다.



이러한 증상이 나타나는 경우 각 데이터 원본에 대해 다음 해결 방법을 시도해 보십시오.

Amazon S3에서 데이터 로드를 병렬화합니다.

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

기본 파티션 수를 결정하십시오.

Amazon S3의 경우 초기 Spark RDD 파티션 수 (각 파티션은 Spark 작업에 해당) 는 Amazon S3 데이터 세트의 기능 (예: 형식, 압축, 크기) 에 따라 결정됩니다. Amazon S3에 저장된 DataFrame CSV 객체에서 AWS Glue DynamicFrame 또는 Spark를 생성할 때 초기 RDD 파티션 수 (`NumPartitions`) 는 다음과 같이 대략적으로 계산할 수 있습니다.

- 객체 크기 <= 64MB: `NumPartitions = Number of Objects`
- 64메가바이트 이상의 오브젝트 크기: `NumPartitions = Total Object Size / 64 MB`
- 분할 불가 (gzip): `NumPartitions = Number of Objects`

데이터 스캔량 줄이기 섹션에서 설명한 것처럼 Spark는 대용량 S3 객체를 병렬로 처리할 수 있는 분할로 나눕니다. 객체가 분할 크기보다 큰 경우 Spark는 객체를 분할하고 분할할 때마다 RDD 파티션 (및 작업) 을 생성합니다. Spark의 분할 크기는 데이터 형식과 런타임 환경에 따라 달라지지만, 시작 시 이 정도면 적당합니다. 일부 객체는 gzip과 같은 분할할 수 없는 압축 형식을 사용하여 압축되므로 Spark는 객체를 분할할 수 없습니다.

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

예를 들어 DataFrame Spark를 사용하여 10GB `csv.gz` 객체 하나를 로드하는 경우 gzip은 분할할 수 없으므로 Spark 드라이버는 RDD 파티션 (`NumPartitions=1`) 을 하나만 생성합니다. 이로 인해 특정 Spark 실행기 하나에 과부하가 발생하고 다음 그림에 설명된 것처럼 나머지 실행기에는 작업이 할당되지 않습니다.

Spark Web UI Stage 탭에서 스테이지의 실제 작업 수 (`NumPartitions`) 를 확인하거나 코드를 실행하여 `df.rdd.getNumPartitions()` 병렬성을 확인하십시오.

10GB의 gzip 파일이 있는 경우 해당 파일을 생성하는 시스템에서 파일을 분할 가능한 형식으로 생성할 수 있는지 확인하십시오. 이 옵션을 사용할 수 없는 경우 파일을 처리하기 위해 클러스터 용량을 확장해야 할 수 있습니다. 로드한 데이터에서 변환을 효율적으로 실행하려면 repartition을 사용하여 클러스터의 작업자 RDD 간에 균형을 재조정해야 합니다.

목표 파티션 수를 수동으로 결정하십시오.

데이터의 속성과 Spark의 특정 기능 구현에 따라 기본 작업을 병렬화할 수 있더라도 결국 낮은 `NumPartitions` 값이 나올 수 있습니다. `NumPartitions`크기가 너무 작으면 처리를 여러 Spark `df.repartition(N)` 실행기에 분산할 수 있도록 실행하여 파티션 수를 늘리십시오.

이 경우 실행 횟수가 `NumPartitions` 1개에서 100개로 `df.repartition(100)` 늘어나 데이터 파티션 100개가 생성되고 각 파티션에는 다른 실행기에 할당할 수 있는 작업이 포함됩니다.

이 작업은 전체 데이터를 균등하게 `repartition(N)` 분할하므로 (10GB/100 파티션 = 100MB/파티션당) 데이터가 특정 파티션으로 치우치지 않습니다.

> **Note**
>
> 등의 `join` 셔플 작업을 실행하면 또는 값에 따라 파티션 수가 동적으로 늘거나 줄어듭니다. `spark.sql.shuffle.partitions` `spark.default.parallelism` 이렇게 하면 Spark 실행기 간에 데이터를 더 효율적으로 교환할 수 있습니다. 자세한 내용은 Spark 설명서를 참조하십시오.

대상 파티션 수를 결정할 때 목표는 AWS Glue 프로비전된 작업자의 사용을 극대화하는 것입니다. AWS Glue 작업자 수와 Spark 작업 수는 작업자 수에 따라 달라집니다. vCPUs Spark는 각 v CPU 코어에 대해 하나의 작업을 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 워커는 Spark 실행기에 v CPU 코어 4개를 제공합니다 (). `spark.executor.cores = 4` Spark는 v CPU Core당 하나의 작업을 지원하므로 G.1X Spark 실행기는 4개의 작업을 동시에 실행할 수 있습니다 (). `numSlotPerExecutor` 작업에 동일한 시간이 소요되는 경우 이 수의 파티션을 사용하면 클러스터를 최대한 활용할 수 있습니다. 하지만 일부 작업은 다른 작업보다 시간이 오래 걸려 코어가 유휴 상태가 됩니다. 이런 경우에는 2 또는 3을 `numPartitions` 곱하여 병목 현상이 발생하는 작업을 분류하고 효율적으로 일정을 잡는 것이 좋습니다.

파티션이 너무 많습니다.

파티션 수가 너무 많으면 작업 수가 너무 많아집니다. 이로 인해 관리 작업 및 Spark 실행자 간 데이터 교환과 같은 분산 처리와 관련된 오버헤드로 인해 Spark 드라이버에 과부하가 발생합니다.

작업의 파티션 수가 목표 파티션 수보다 훨씬 많으면 파티션 수를 줄이는 것이 좋습니다. 다음 옵션을 사용하여 파티션을 줄일 수 있습니다.

- 파일 크기가 매우 작은 경우 사용하십시오 AWS Glue groupFiles. Apache Spark 작업을 실행하여 각 파일을 처리할 때 발생하는 과도한 병렬 처리를 줄일 수 있습니다.
- 파티션을 `coalesce(N)` 병합하는 데 사용합니다. 이는 비용이 적게 드는 프로세스입니다. 셔플을 `repartition(N)` 수행하여 각 파티션의 레코드 양을 `repartition(N)` 균등하게 분배하므로 파티션 수를 줄일 때는 이 보다 `coalesce(N)` 선호됩니다. 이로 인해 비용과 관리 오버헤드가 증가합니다.
- Spark 3.x 적응형 쿼리 실행을 사용하세요. Apache Spark의 주요 항목에서 설명한 것처럼 적응형 쿼리 실행은 파티션 수를 자동으로 통합하는 기능을 제공합니다. 실행을 수행하기 전까지 파티션 수를 알 수 없는 경우 이 방법을 사용할 수 있습니다.

에서 데이터 로드를 병렬화합니다. JDBC

Spark RDD 파티션 수는 구성에 따라 결정됩니다. 기본적으로 쿼리를 통해 전체 소스 데이터세트를 스캔하는 작업은 한 번만 실행됩니다. `SELECT`

둘 다 AWS Glue DynamicFrames Spark와 Spark는 DataFrames 여러 작업에 걸쳐 병렬화된 JDBC 데이터 로드를 지원합니다. 이는 `where` 조건자를 사용하여 하나의 `SELECT` 쿼리를 여러 쿼리로 분할하는 방식으로 이루어집니다. 에서 읽기를 JDBC 병렬화하려면 다음 옵션을 구성하십시오.

- For AWS Glue DynamicFrame, 설정 `hashfield` (또는`hashexpression`)). `hashpartition` 자세히 알아보려면 JDBC테이블에서 병렬로 읽기를 참조하십시오.

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

```
    transformation_ctx= "datasource0"
)
```

- Spark의 경우 DataFrame, numPartitions partitionColumnlowerBound, 및 upperBound 를 설정합니다. 자세히 JDBC알아보려면 다른 데이터베이스로을 참조하십시오.

```
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/")
```

커넥터를 사용할 때 DynamoDB에서 데이터 로드를 병렬화합니다. ETL

Spark RDD 파티션의 수는 파라미터에 의해 결정됩니다. dynamodb.splits Amazon DynamoDB에서 읽기를 병렬화하려면 다음 옵션을 구성하십시오.

- 의 가치를 높이십시오. dynamodb.splits
- AWS Glue Spark용 ETL in의 연결 유형 및 옵션에 설명된 공식에 따라 매개변수를 최적화하십시오.

Kinesis Data Streams에서 데이터 로드를 병렬화합니다.

Spark RDD 파티션의 수는 원본 Amazon Kinesis Data Streams 데이터 스트림의 샤드 수에 따라 결정됩니다. 데이터 스트림에 샤드가 몇 개만 있는 경우 Spark 작업은 몇 개만 남게 됩니다. 이로 인해 다운스트림 프로세스의 병렬성이 낮아질 수 있습니다. Kinesis Data Streams에서 읽기를 병렬화하려면 다음 옵션을 구성하십시오.

- Kinesis Data Streams에서 데이터를 로드할 때 병렬성을 높이려면 샤드 수를 늘리십시오.
- 마이크로 배치의 로직이 충분히 복잡한 경우 일괄 처리 시작 시 불필요한 열을 삭제한 후 데이터를 다시 파티셔닝하는 것을 고려해 보십시오.

자세한 내용은 스트리밍 작업의 비용 및 성능 최적화를 위한 모범 사례를 참조하십시오. AWS Glue ETL

데이터 로드 후 작업을 병렬화합니다.

데이터 로드 후 작업을 병렬화하려면 다음 옵션을 사용하여 RDD 파티션 수를 늘리십시오.

- 데이터를 다시 파티셔닝하여 더 많은 파티션을 생성합니다. 특히 로드 자체를 병렬화할 수 없는 경우 초기 로드 직후에 더 많이 생성합니다.

  파티션 수를 지정하여 `repartition()` DynamicFrame on이나 DataFrame 를 호출하십시오. 일반적으로 사용 가능한 코어 수의 2~3배입니다.

  하지만 파티션을 나눈 테이블을 작성할 때 이로 인해 파일이 폭증할 수 있습니다. 각 파티션은 잠재적으로 각 테이블 파티션에 파일을 생성할 수 있습니다. 이를 방지하기 위해 DataFrame 기준 열을 다시 분할할 수 있습니다. 여기에는 테이블 파티션 열이 사용되므로 쓰기 전에 데이터가 정리됩니다. 테이블 파티션에 작은 파일을 넣지 않고도 더 많은 수의 파티션을 지정할 수 있습니다. 그러나 일부 파티션 값에 대부분의 데이터가 포함되고 작업 완료가 지연되는 데이터 왜곡을 피해야 합니다.
- 셔플이 있는 경우에는 값을 늘리십시오. `spark.sql.shuffle.partitions` 셔플링할 때 발생하는 메모리 문제에도 도움이 될 수 있습니다.

  셔플 파티션이 2,001개 이상인 경우 Spark는 압축 메모리 형식을 사용합니다. 숫자가 이에 가까우면 `spark.sql.shuffle.paritions` 값을 이 한도 이상으로 설정하여 더 효율적으로 표현할 수 있습니다.

셔플 최적화

`join()`및 와 같은 특정 작업의 경우 Spark에서 `groupByKey()` 셔플을 수행해야 합니다. 셔플은 데이터를 재분배하여 파티션 간에 다르게 그룹화되도록 하는 Spark의 메커니즘입니다. RDD 셔플링은 성능 병목 현상을 해결하는 데 도움이 될 수 있습니다. 그러나 셔플은 일반적으로 Spark 실행기 간에 데이터를 복사하는 것이므로 셔플은 복잡하고 비용이 많이 드는 작업입니다. 예를 들어 셔플을 사용하면 다음과 같은 비용이 발생합니다.

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

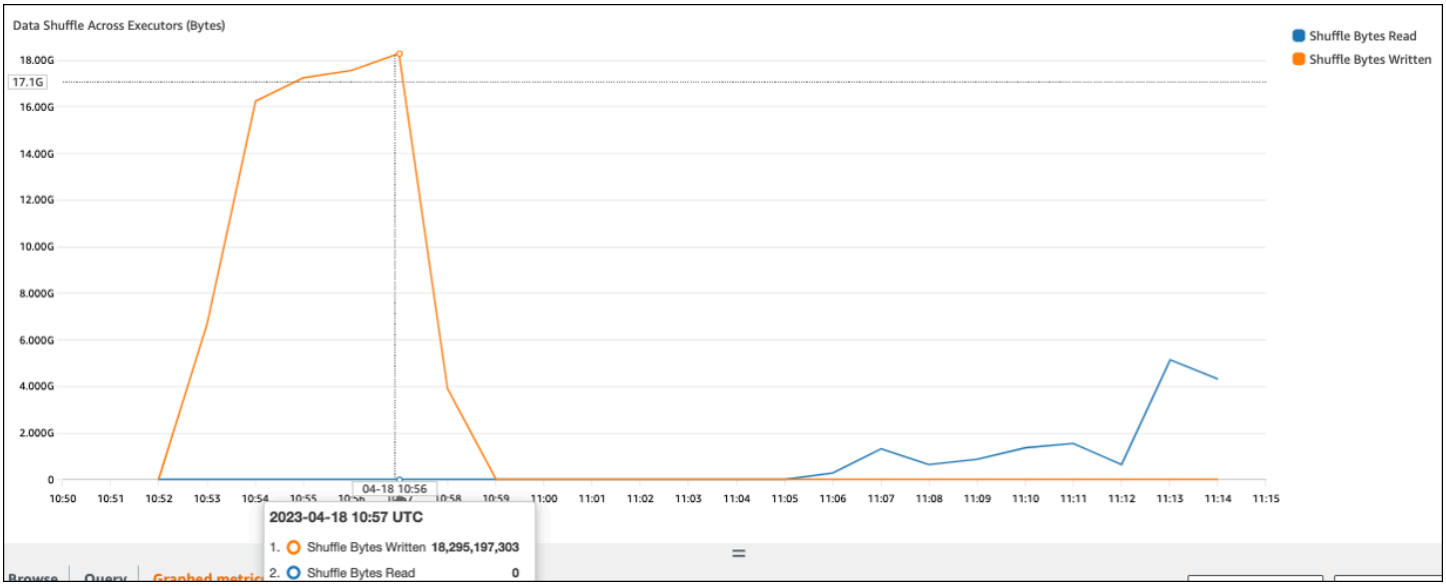
  - 레코드는 다른 Spark 실행기에서 호스팅될 수 있는 새 RDD 파티션에 집계되므로 데이터세트의 상당 부분이 네트워크를 통해 Spark 실행기 간에 이동할 수 있습니다.
- CPU및 메모리 로드:
  - 값을 정렬하고 데이터 세트를 병합합니다. 이러한 작업은 실행자에서 계획되므로 실행자에 많은 부하가 가해집니다.

셔플은 Spark 애플리케이션 성능 저하의 가장 큰 요인 중 하나입니다. 중간 데이터를 저장하는 동안 실행기의 로컬 디스크 공간이 고갈되어 Spark 작업이 실패할 수 있습니다.

셔플 성능은 CloudWatch 메트릭과 Spark UI에서 평가할 수 있습니다.

CloudWatch 메트릭스

셔플 바이트 쓰기 값이 읽은 바이트와 비교하여 높으면 Spark 작업에서 또는 같은 셔플 연산을 사용할 수 있습니다. `join()` `groupByKey()`

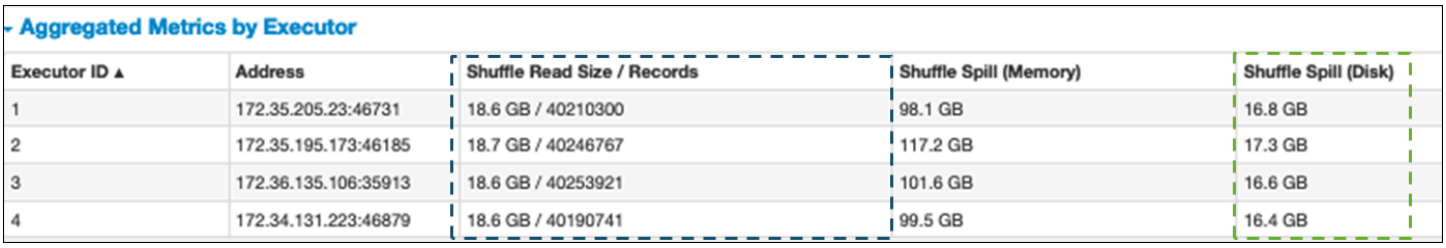


Spark UI

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

다음 스크린샷에서 각 실행기는 셔플 프로세스를 통해 약 18.6GB/4020000개의 레코드를 교환하며 총 셔플 읽기 크기는 약 75GB입니다.

Shuffle Spill (Disk) 열에는 디스크로 유출된 많은 양의 데이터 메모리가 표시되어 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 나타나고 스테이지가 성능 목표에 비해 너무 오래 `Out Of Memory` 걸리거나 `No space left on device` 오류와 함께 실패하는 경우 다음 해결 방법을 고려해 보십시오.

조인을 최적화하세요.

테이블을 조인하는 `join()` 작업은 가장 일반적으로 사용되는 셔플 작업이지만 성능 병목 현상이 발생하는 경우가 많습니다. 조인은 비용이 많이 드는 작업이므로 비즈니스 요구 사항에 꼭 필요한 경우가 아니면 사용하지 않는 것이 좋습니다. 다음 질문을 통해 데이터 파이프라인을 효율적으로 사용하고 있는지 다시 한 번 확인하세요.

- 재사용할 수 있는 다른 작업에서도 수행되는 조인을 다시 계산하고 있습니까?
- 출력 시 소비자가 사용하지 않는 값으로 외래 키를 해석하는 데 참여하고 계신가요?

조인 작업이 비즈니스 요구 사항에 필수적인지 확인한 후 요구 사항에 맞는 방식으로 조인을 최적화하기 위한 다음 옵션을 참조하십시오.

가입하기 전에 푸시다운을 사용하세요.

조인을 수행하기 DataFrame 전에 에서 불필요한 행과 열을 걸러냅니다. 여기에는 다음과 같은 이점이 있습니다.

- 셔플 중에 전송되는 데이터 양을 줄입니다.
- Spark 실행기의 처리량을 줄입니다.
- 데이터 스캔량을 줄입니다.

```
# 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"])
```

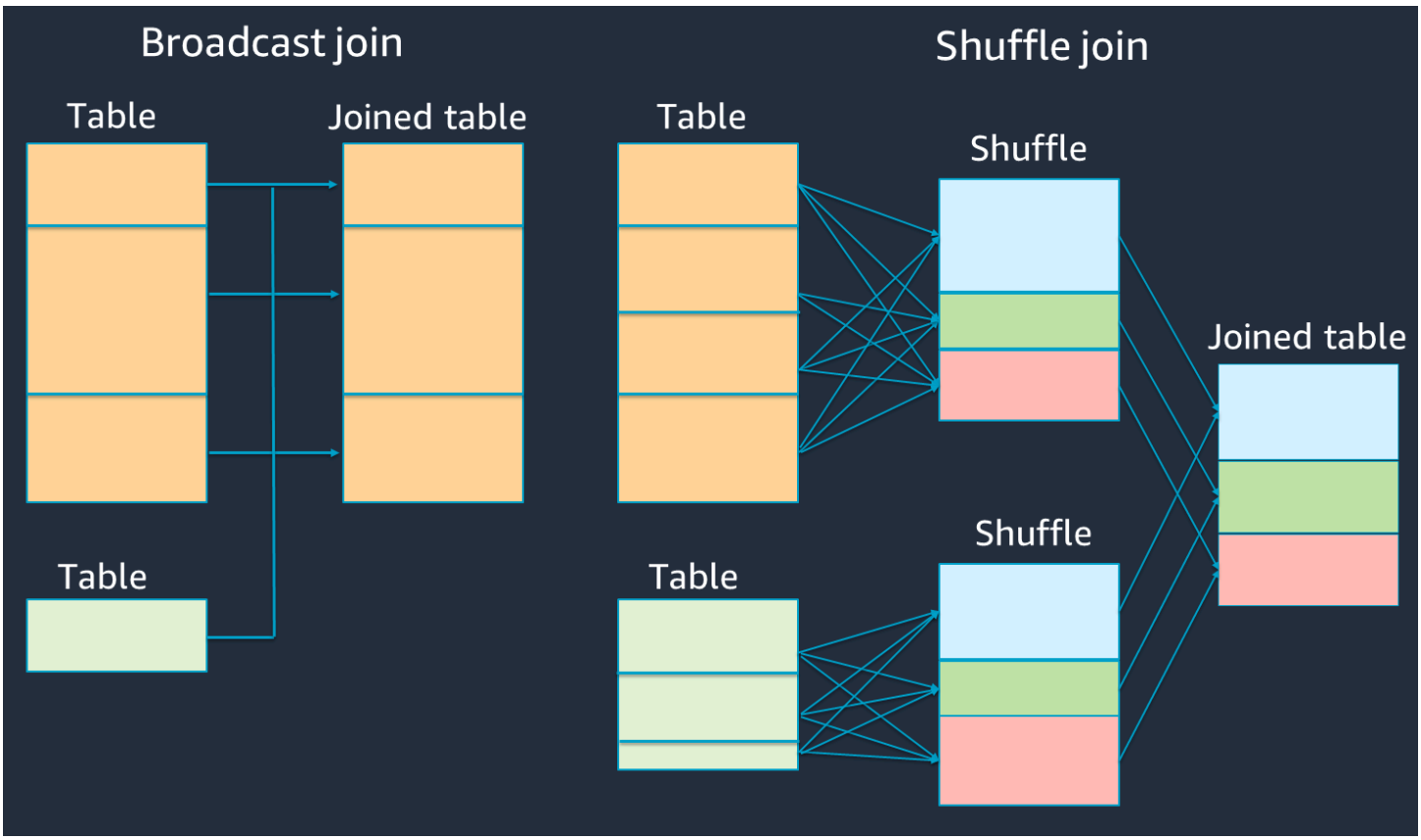
DataFrame Join 사용

또는 조인 대신 SparkSQL, DataFrame, Datasets와 API 같은 Spark 상위 레벨을 사용해 보세요. RDD API DynamicFrame DataFrame 와 같은 메서드 DynamicFrame 호출을 사용하여 변환할 수 있습니다. `dyf.toDF()` Apache Spark의 주요 항목에서 설명한 것처럼 이러한 조인 작업은 내부적으로 Catalyst 최적화 프로그램의 쿼리 최적화를 활용합니다.

셔플 및 브로드캐스트 해시 조인 및 힌트

Spark는 셔플 조인과 브로드캐스트 해시 조인이라는 두 가지 유형의 조인을 지원합니다. 브로드캐스트 해시 조인은 셔플이 필요하지 않으며 셔플 조인보다 처리가 덜 필요할 수 있습니다. 하지만 작은 테이블을 큰 테이블에 조인하는 경우에만 적용됩니다. 단일 Spark 실행자의 메모리에 들어갈 수 있는 테이블을 조인할 때는 브로드캐스트 해시 조인을 사용하는 것이 좋습니다.

다음 다이어그램은 브로드캐스트 해시 조인과 셔플 조인의 상위 구조 및 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 테이블을 조인하는 데 적합합니다.
  - sort-merge 조인은 조인할 두 테이블을 키로 분산하고 조인 전에 정렬합니다. 대형 테이블을 조인하는 데 적합합니다.
- 브로드캐스트 해시 조인:
  - 브로드캐스트 해시 조인은 더 작은 RDD OR 테이블을 각 워커 노드로 푸시합니다. 그런 다음 더 큰 OR 테이블의 각 파티션과 맵 측 결합을 수행합니다. RDD

    테이블 RDDs 또는 테이블 중 하나를 메모리에 담을 수 있거나 메모리에 맞게 만들 수 있는 경우 조인에 적합합니다. 브로드캐스트 해시 조인은 셔플이 필요하지 않으므로 가능하면 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 조인 힌트를 참조하십시오.

AWS Glue 3.0 이상에서는 적응형 쿼리 실행 및 추가 매개변수를 활성화하여 브로드캐스트 해시 조인을 자동으로 활용할 수 있습니다. 적응형 쿼리 실행은 어느 쪽 조인의 런타임 통계가 적응형 브로드캐스트 해시 조인 임계값보다 작을 때 정렬-병합 조인을 브로드캐스트 해시 조인으로 변환합니다.

AWS Glue 3.0에서는 설정을 통해 적응형 쿼리 실행을 활성화할 수 있습니다. `spark.sql.adaptive.enabled=true` 적응형 쿼리 실행은 AWS Glue 4.0에서 기본적으로 활성화되어 있습니다.

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 sort-merge 조인을 브로드캐스트 조인으로 변환을 참조하십시오.

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

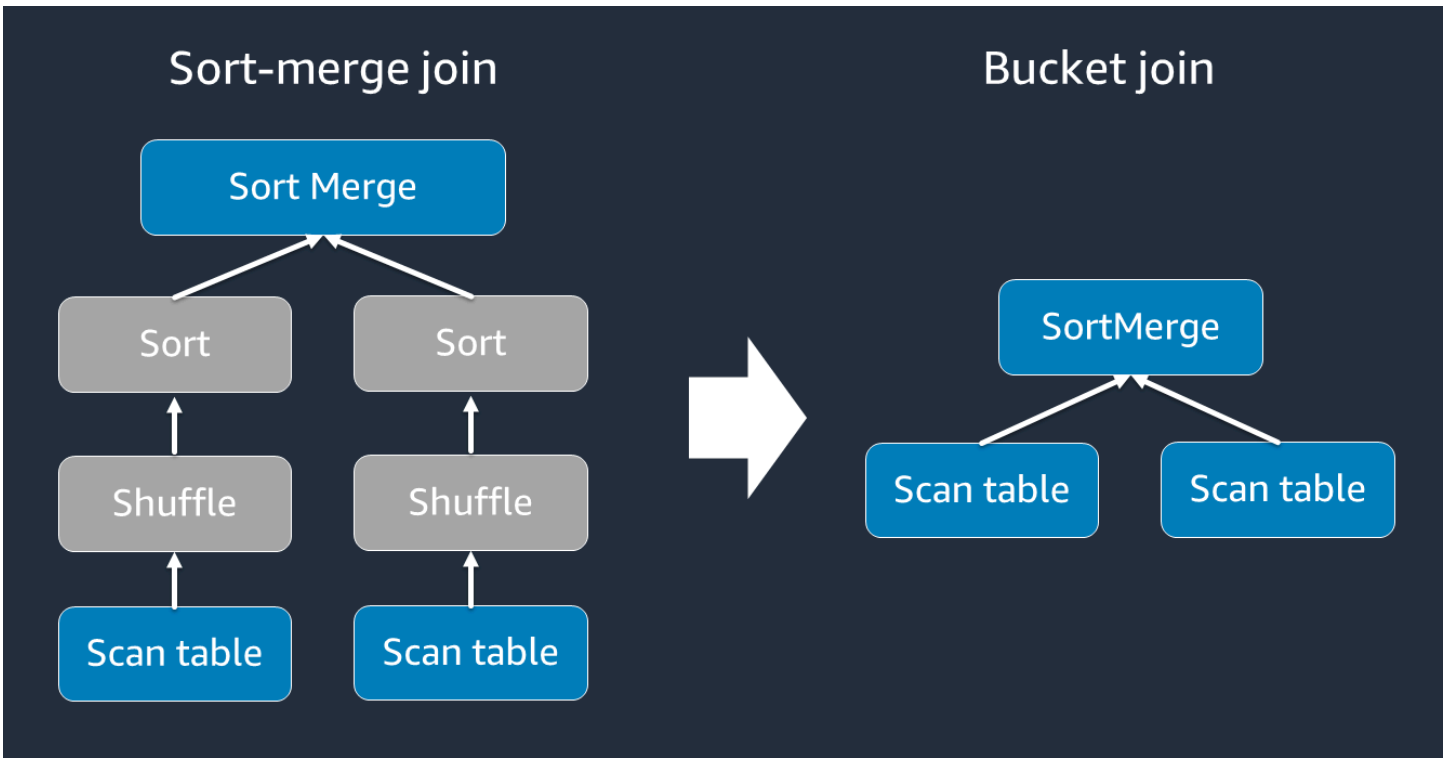
```
-- Join Hints for shuffle sort merge join
SELECT /*+ SHUFFLE_MERGE(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
SELECT /*+ MERGEJOIN(t2) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;
SELECT /*+ MERGE(t1) /  FROM t1 INNER JOIN t2 ON t1.key = t2.key;

-- 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;
```

버켓팅 사용하기

정렬-병합 조인에는 셔플과 정렬, 병합이라는 두 단계가 필요합니다. 이 두 단계는 Spark 실행기에 과부하가 걸리고 일부 실행기가 병합되고 다른 실행기가 OOM 동시에 정렬될 때 성능 문제가 발생할 수 있습니다. 이런 경우에는 버켓팅을 사용하여 효율적으로 조인할 수 있습니다. 버킷팅은 조인 키의 입력을 미리 섞고 미리 정렬한 다음 정렬된 데이터를 중간 테이블에 기록합니다. 정렬된 중간 테이블을 미리 정의하여 대형 테이블을 조인할 때 셔플 및 정렬 단계의 비용을 줄일 수 있습니다.



버킷이 있는 테이블은 다음과 같은 경우에 유용합니다.

- 데이터는 다음과 같이 동일한 키를 통해 자주 조인됩니다. `account_id`
- 일별 누적 테이블 (예: 공통 열에 버킷으로 지정할 수 있는 기본 및 델타 테이블) 로드

다음 코드를 사용하여 버킷이 있는 테이블을 만들 수 있습니다.

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

조인 전에 조인 키를 다시 DataFrames 파티셔닝합니다.

조인 전에 조인 DataFrames 키에서 두 키를 다시 분할하려면 다음 명령문을 사용하십시오.

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

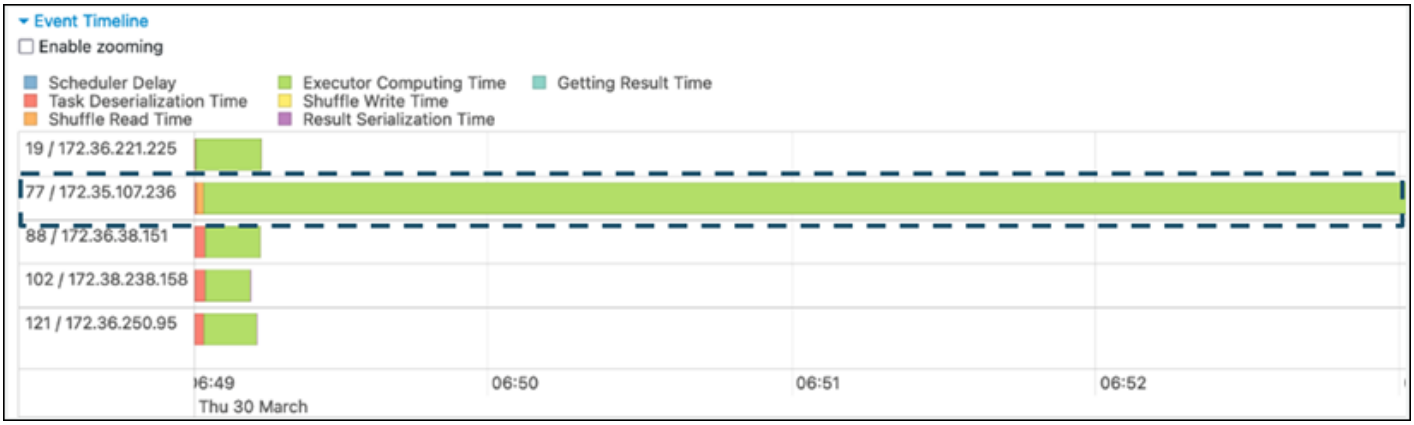
이렇게 하면 조인을 시작하기 전에 조인 RDDs 키에서 두 개의 파티션 (여전히 분리되어 있음) 이 분할됩니다. 두 RDDs 레코드가 동일한 키로 동일한 파티셔닝 코드를 사용하여 파티셔닝된 경우, 함께 조인하려는 RDD 레코드가 조인을 위해 섞이기 전에 동일한 워커에 같은 위치에 배치될 가능성이 높습니다. 이렇게 하면 조인 중에 네트워크 활동 및 데이터 왜곡이 줄어들어 성능이 향상될 수 있습니다.

데이터 편차를 극복하세요.

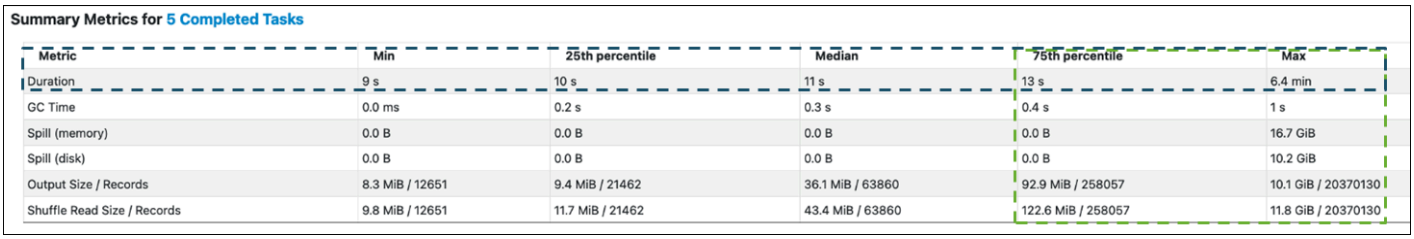
데이터 왜곡은 Spark 작업 병목 현상의 가장 일반적인 원인 중 하나입니다. 데이터가 파티션 전체에 균일하게 분산되지 않을 때 발생합니다. RDD 이로 인해 해당 파티션의 작업이 다른 파티션보다 훨씬 오래 걸리고 응용 프로그램의 전체 처리 시간이 지연됩니다.

데이터 편향을 식별하려면 Spark UI에서 다음 지표를 평가하세요.

- Spark UI의 스테이지 탭에서 이벤트 타임라인 페이지를 살펴보세요. 다음 스크린샷에서 작업이 고르지 않게 분산되어 있는 것을 확인할 수 있습니다. 작업이 고르지 않게 분산되거나 실행 시간이 너무 오래 걸리는 것은 데이터 왜곡을 의미할 수 있습니다.



- 또 다른 중요한 페이지는 Spark 작업에 대한 통계를 보여주는 요약 지표입니다. 다음 스크린샷은 지속 시간, GC 시간, 유출 (메모리), 유출 (디스크) 등에 대한 백분위수가 포함된 메트릭을 보여줍니다.



Summary Metrics for 5 Completed Tasks

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

  작업이 균등하게 분배되면 모든 백분위수에서 비슷한 수치를 볼 수 있습니다. 데이터 편차가 있는 경우 각 백분위수에서 매우 편향된 값을 볼 수 있습니다. 이 예제에서 작업 지속 시간은 최소, 25번째 백분위수, 중앙값, 75번째 백분위수 단위로 13초 미만입니다. Max 작업은 75번째 백분위수보다 100배 많은 데이터를 처리했지만, 6.4분이라는 시간은 약 30배 더 깁니다. 즉, 하나 이상의 작업 (또는 작업의 최대 25%) 이 나머지 작업보다 훨씬 오래 걸렸습니다.

데이터 왜곡이 보이면 다음을 시도해 보세요.

- AWS Glue 3.0을 사용하는 경우 설정을 `spark.sql.adaptive.enabled=true` 통해 적응형 쿼리 실행을 활성화하십시오. 적응형 쿼리 실행은 AWS Glue 4.0에서 기본적으로 활성화됩니다.

  다음과 같은 관련 매개변수를 설정하여 조인으로 인한 데이터 왜곡에 대해 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 설명서를 참조하십시오.

- 조인 키에는 값 범위가 넓은 키를 사용하십시오. 셔플 조인에서는 키의 각 해시 값에 대해 파티션이 결정됩니다. 조인 키의 카디널리티가 너무 낮으면 해시 함수가 파티션 간에 데이터를 분산시키는 잘못된 역할을 할 가능성이 높습니다. 따라서 애플리케이션과 비즈니스 로직에서 지원하는 경우 더 높은 카디널리티 키나 복합 키를 사용하는 것이 좋습니다.

```
# 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

반복적으로 DataFrames 사용할 때는 계산 결과를 각 Spark 실행기의 메모리와 디스크에 `df.cache()` 캐시하거나 `df.persist()` 사용하여 추가 셔플 또는 계산을 수행하지 마십시오. 또한 Spark는 RDDs 디스크에 보관하거나 여러 노드에 걸쳐 복제하는 기능 (스토리지 수준) 을 지원합니다.

예를 들어, 추가하여 지속할 수 있습니다. DataFrames `df.persist()` 캐시가 더 이상 필요하지 않은 경우 를 `unpersist` 사용하여 캐시된 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 액션을 제거합니다.

`count`, `show`, 또는 같은 불필요한 작업은 실행하지 마세요. `collect` Apache Spark의 주요 항목 섹션에서 설명한 것처럼 Spark는 게으르다. 변환된 각 변환은 작업을 실행할 때마다 다시 RDD 계산될 수

있습니다. 많은 Spark 액션을 사용하는 경우 각 액션에 대한 다중 소스 액세스, 작업 계산 및 셔플 실행이 호출됩니다.

상용 환경에서 필요하지 `collect()` 않거나 다른 작업이 없는 경우 해당 작업을 제거하는 것이 좋습니다.

> **Note**
>
> 상업용 `collect()` 환경에서는 Spark를 최대한 사용하지 마세요. 이 `collect()` 액션은 Spark 실행기의 모든 계산 결과를 Spark 드라이버로 반환하며, 이로 인해 Spark 드라이버에서 오류가 반환될 수 있습니다. OOM OOM오류를 방지하기 위해 Spark는 `spark.driver.maxResultSize = 1GB` 기본적으로 Spark 드라이버에 반환되는 최대 데이터 크기를 1GB로 제한하도록 설정합니다.

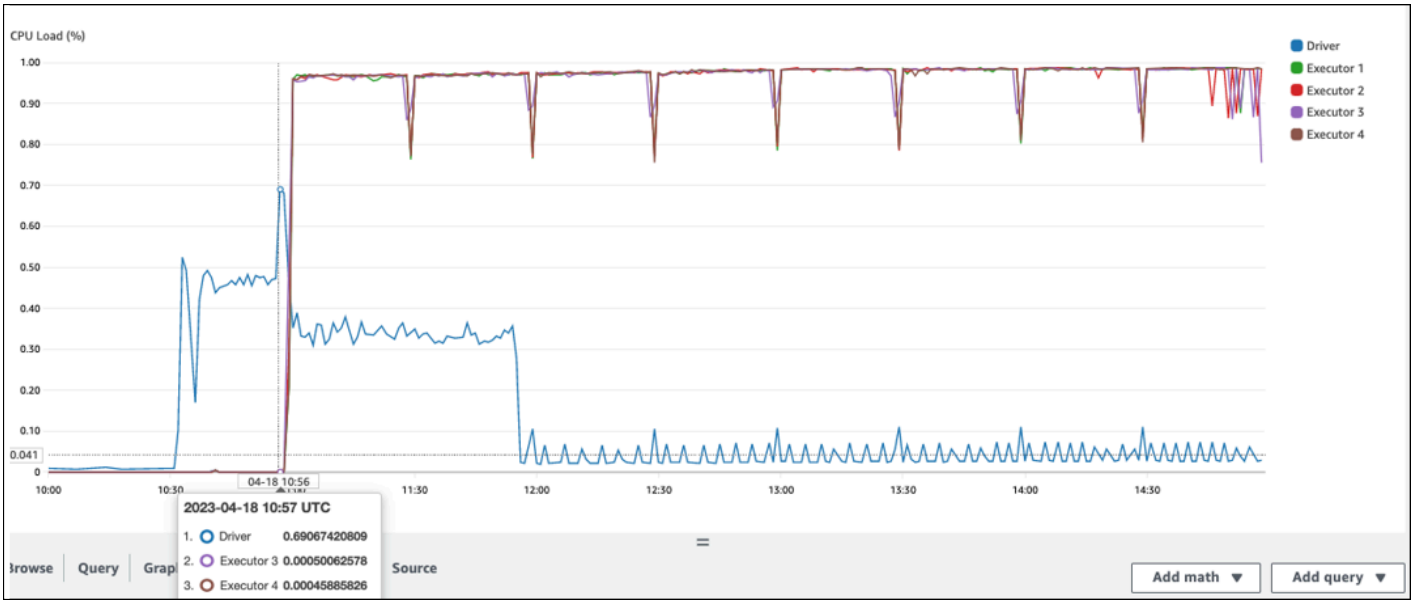
계획 오버헤드를 최소화하세요.

Apache Spark의 주요 항목에서 설명한 것처럼 Spark 드라이버는 실행 계획을 생성합니다. 이 계획에 따라 작업은 분산 처리를 위해 Spark 실행기에 할당됩니다. 그러나 작은 파일이 많거나 파티션 수가 많은 경우 Spark 드라이버에 병목 현상이 발생할 수 있습니다. AWS Glue Data Catalog 높은 계획 오버헤드를 식별하려면 다음 지표를 평가하세요.

CloudWatch 지표

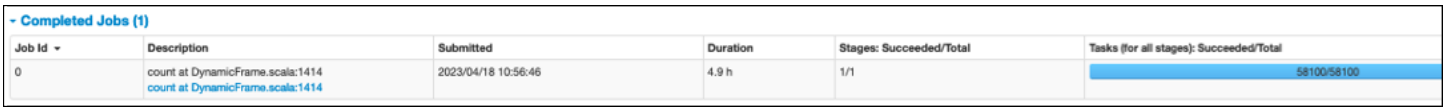
다음과 같은 상황에 대한 CPU로드 및 메모리 사용률을 확인하십시오.

- Spark 드라이버 CPU부하 및 메모리 사용률은 높은 것으로 기록됩니다. 일반적으로 Spark 드라이버는 데이터를 처리하지 않으므로 CPU 로드 및 메모리 사용률이 급증하지 않습니다. 하지만 Amazon S3 데이터 원본에 작은 파일이 너무 많으면 모든 S3 객체를 나열하고 많은 작업을 관리하면 리소스 사용률이 높아질 수 있습니다.
- Spark 실행기에서 처리가 시작되기까지 긴 공백이 있습니다. 다음 예제 스크린샷에서는 작업이 10:00에 시작되었음에도 불구하고 Spark 실행기의 CPU 부하가 10:57 까지 너무 낮습니다. AWS Glue 이는 Spark 드라이버가 실행 계획을 생성하는 데 시간이 오래 걸릴 수 있음을 나타냅니다. 이 예제에서는 데이터 카탈로그에서 많은 파티션을 검색하고 Spark 드라이버에 있는 많은 수의 작은 파일을 나열하는 데 시간이 오래 걸립니다.



Spark UI

Spark UI의 Job 탭에서 제출 시간을 확인할 수 있습니다. 다음 예제에서 Spark 드라이버는 작업이 10:00:00 에 시작되었음에도 불구하고 10:56:46 에 job0을 시작했습니다. AWS Glue



또한 Job 탭에서 태스크 (모든 단계): 성공/총 시간을 확인할 수 있습니다. 이 경우 작업 수는 다음과 같이 기록됩니다. `58100` 작업 병렬화 페이지의 Amazon S3 섹션에 설명된 대로 작업 수는 대략 S3 객체 수와 일치합니다. 즉, Amazon S3에는 약 58,100개의 객체가 있습니다.

이 작업과 일정에 대한 자세한 내용은 스테이지 탭을 참조하십시오. Spark 드라이버에서 병목 현상이 발생하는 경우 다음 해결 방법을 고려해 보십시오.

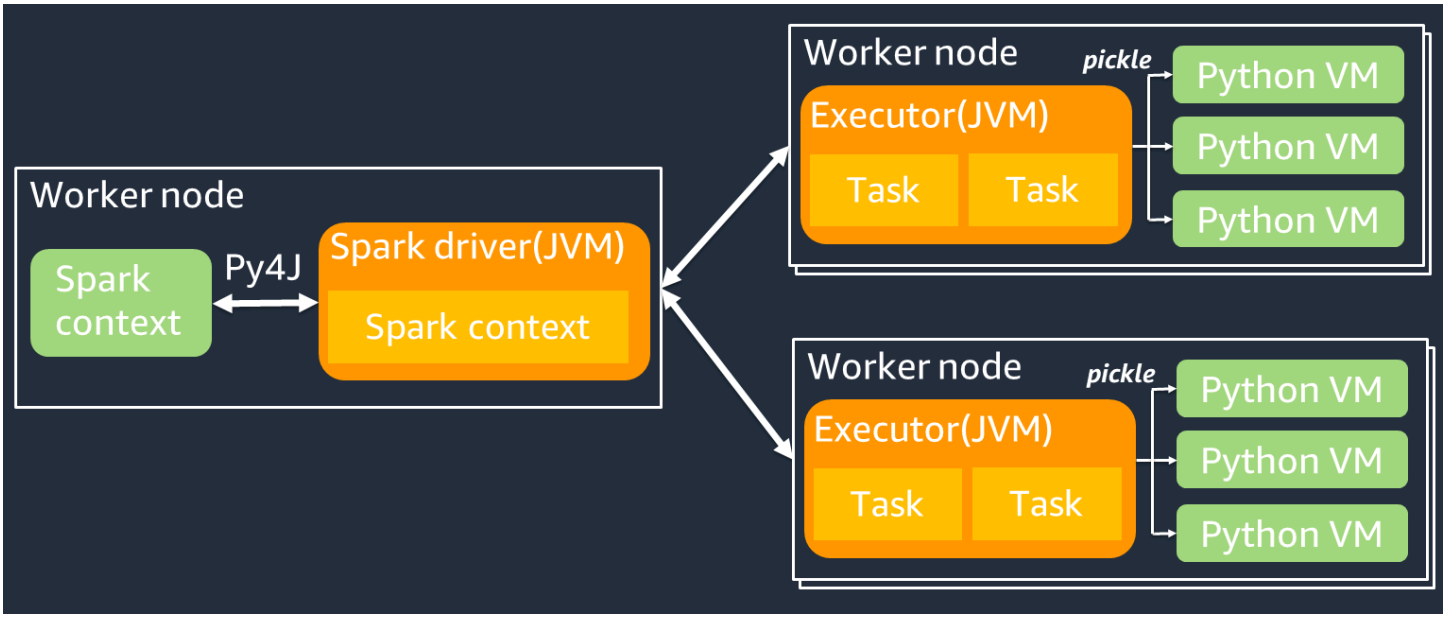
- Amazon S3에 파일이 너무 많으면 병렬화 작업 페이지의 너무 많은 파티션 섹션에서 과도한 병렬화에 대한 지침을 고려하십시오.
- Amazon S3에 파티션이 너무 많으면 데이터 스캔량 줄이기 페이지의 너무 많은 Amazon S3 파티션 섹션에 있는 과도한 파티셔닝에 대한 지침을 고려하십시오. 파티션이 많은 경우 AWS Glue 파티션 인덱스를 활성화하여 데이터 카탈로그에서 파티션 메타데이터를 검색하는 데 걸리는 지연 시간을 줄이십시오. 자세한 내용은 AWS Glue 파티션 인덱스를 사용한 쿼리 성능 개선을 참조하십시오.
- JDBC파티션이 너무 많으면 `hashpartition` 값을 낮추세요.
- DynamoDB에 파티션이 너무 많으면 값을 낮추십시오. `dynamodb.splits`

- 스트리밍 작업에 파티션이 너무 많으면 샤드 수를 줄이십시오.

사용자 정의 함수 최적화

사용자 정의 함수 (UDFs) 및 `RDD.map` 에서는 PySpark 종종 성능이 크게 저하됩니다. 이는 Spark의 기본 Scala 구현에서 Python 코드를 정확하게 표현하는 데 필요한 오버헤드 때문입니다.

다음 다이어그램은 작업 아키텍처를 보여줍니다. PySpark 를 사용하면 PySpark Spark 드라이버가 Py4J 라이브러리를 사용하여 Python에서 Java 메서드를 호출합니다. Spark SQL 또는 DataFrame 내장 함수를 호출할 때 함수는 최적화된 실행 계획을 JVM 사용하여 각 실행기에서 실행되므로 Python과 Scala 간에 성능 차이가 거의 없습니다.



를 사용하는 것과 같이 자체 Python 로직을 사용하는 `map/ mapPartitions/ udf` 경우 작업은 Python 런타임 환경에서 실행됩니다. 두 환경을 관리하면 오버헤드 비용이 발생합니다. 또한 JVM 런타임 환경의 내장 함수에서 사용하려면 메모리의 데이터를 변환해야 합니다. Pickle은 Python 런타임과 JVM Python 런타임 간의 교환에 기본적으로 사용되는 직렬화 형식입니다. 하지만 이 직렬화 및 역직렬화 비용은 매우 높기 때문에 Java나 Scala로 UDFs 작성하는 것이 Python보다 빠릅니다. UDFs

직렬화 및 역직렬화 오버헤드를 피하려면 다음 사항을 고려하세요. PySpark

- 내장 Spark SQL 함수 사용 — 자체 함수 UDF 또는 맵 함수를 Spark 또는 내장 함수로 대체해 보세요. SQL DataFrame Spark SQL 또는 DataFrame 내장 함수를 실행할 때는 각 실행기에서 작업을 처리하기 때문에 Python과 Scala의 성능 차이는 거의 없습니다. JVM

- Scala 또는 Java로 구현 UDFs — Java 또는 UDF Scala로 작성된 함수는 에서 실행되므로 사용을 고려해 보십시오. JVM
- 벡터화된 워크로드에는 Apache Arrow 기반 사용 — 화살표 기반 사용을 UDFs 고려해 보세요. UDFs 이 기능을 벡터라이즈드 (Pandas) 라고도 합니다. UDF UDF Apache Arrow는 Python 프로세스 간에 데이터를 효율적으로 전송하는 데 사용할 AWS Glue 수 있는 언어에 구애받지 않는 인메모리 데이터 형식입니다. JVM 이것은 현재 Pandas 또는 NumPy 데이터를 사용하는 Python 사용자에게 가장 유용합니다.

  화살표는 열 형식 (벡터화) 형식입니다. 사용법은 자동으로 이루어지지 않으므로 최대한 활용하고 호환성을 보장하기 위해 구성이나 코드를 약간 변경해야 할 수 있습니다. 자세한 내용 및 제한 사항은 의 Apache Arrow를 참조하십시오. PySpark

  다음 예제는 표준 UDF Python의 기본 증분을 Vectorized와 Spark의 기본 UDF 증분을 비교합니다. SQL

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)
```

벡터화 UDF

예제 시간은 0.59 (초) 입니다.

벡터화된 UDF 결과는 이전 예제보다 5배 더 빠릅니다. UDF Physical Plan확인해보면 이 애플리케이션이 Apache ArrowEvalPython Arrow로 벡터화되었음을 알 수 있습니다. Vectorized를 활성화하려면 코드에 지정해야 UDF 합니다. `spark.sql.execution.arrow.pyspark.enabled = true`

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)
```

스파크 SQL

예제 시간은 0.087 (초) 입니다.

SQLSpark는 Vectorized보다 훨씬 빠릅니다. UDF Python 런타임 없이 JVM 각 실행기에서 작업이 실행되기 때문입니다. 를 내장 함수로 대체할 수 UDF 있다면 그렇게 하는 것이 좋습니다.

예제 코드

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

판다를 빅데이터에 활용하기

이미 판다에 익숙하고 Spark를 빅 데이터에 사용하고 싶다면 Spark에서 판다를 사용할 수 있습니다. API AWS Glue 4.0 이상에서는 이를 지원합니다. 시작하려면 공식 노트북 퀵스타트: Pandas API on Spark를 사용할 수 있습니다. 자세한 내용은 설명서를 참조하십시오. PySpark

리소스

- AWS Glue
- 성능 튜닝 (Spark SQL 가이드)
- AWS Glue 최적화 워크숍

문서 기록

아래 표에 이 가이드의 주요 변경 사항이 설명되어 있습니다. 향후 업데이트에 대한 알림을 받으려면 RSS 피드를 구독하십시오.

변경 사항	설명	날짜
최초 게시	—	2024년 1월 2일

AWS 규범적 지침 용어집

다음은 규범적 지침에서 제공하는 AWS 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 마이그레이션하십시오.
- 리플랫포밍(리프트 앤드 리셰이프) - 애플리케이션을 클라우드로 이동하고 일정 수준의 최적화를 도입하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 오라클용 Amazon RDS (Amazon RDS) 로 마이그레이션합니다. AWS 클라우드
- 재구매(드롭 앤드 숍) - 일반적으로 기존 라이선스에서 SaaS 모델로 전환하여 다른 제품으로 전환합니다. 예: 고객 관계 관리 (CRM) 시스템을 Salesforce.com으로 마이그레이션하십시오.
- 리호스팅(리프트 앤드 시프트) - 애플리케이션을 변경하지 않고 클라우드로 이동하여 클라우드 기능을 활용합니다. 예: 온프레미스 Oracle 데이터베이스를 의 EC2 인스턴스에서 Oracle로 마이그레이션합니다. AWS 클라우드
- 재배치(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 다음으로 마이그레이션하십시오. AWS
- 유지(보관) - 소스 환경에 애플리케이션을 유지합니다. 대규모 리팩터링이 필요하고 해당 작업을 나중으로 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

ABAC

속성 기반 액세스 제어를 참조하십시오.

추상화된 서비스

관리형 서비스를 참조하십시오.

산

원자성, 일관성, 격리성, 내구성을 참조하십시오.

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 마이그레이션보다 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.

집계 함수

행 그룹에서 연산을 수행하고 그룹에 대한 단일 반환값을 계산하는 SQL 함수입니다. 집계 함수의 예로는 및 등이 `SUM` 있습니다. `MAX`

AI

인공 지능을 참조하십시오.

AIOps

인공 지능 운영을 참조하십시오.

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

시스템을 멀웨어로부터 보호하기 위해 승인된 애플리케이션만 사용할 수 있는 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 검색 및 분석 프로세스의 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 가이드를 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 AWS ABAC for를 참조하십시오.

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 격리되고 동일한 지역 내 다른 가용 영역에 저렴하고 지연 시간이 짧은 네트워크 연결을 제공하는 별도의 위치. AWS 리전

AWS 클라우드 채택 프레임워크 (AWS CAF)

조직이 클라우드로 성공적으로 AWS 전환하기 위한 효율적이고 효과적인 계획을 개발하는 데 도움이 되는 지침 및 모범 사례 프레임워크입니다. AWS CAF는 지침을 관점이라고 하는 6가지 중점 영역, 즉 비즈니스, 사람, 거버넌스, 플랫폼, 보안, 운영으로 분류합니다. 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 조직이 성공적인 클라우드 채택을 준비할 수 있도록 인력 개발, 교육 및 커뮤니케이션에 대한 지침을 제공합니다. 자세한 내용은 AWS CAF 웹 사이트와 AWS CAF 백서를 참조하십시오.

AWS 워크로드 검증 프레임워크 (AWS WQF)

데이터베이스 마이그레이션 워크로드를 평가하고 마이그레이션 전략을 권장하며 작업 예상치를 제공하는 도구입니다. AWS WQF는 () 에 포함됩니다. AWS Schema Conversion Tool AWS SCT데이터베이스 스키마 및 코드 객체, 애플리케이션 코드, 종속성 및 성능 특성을 분석하고 평가 보고서를 제공합니다.

B

배드 봇

개인이나 조직을 방해하거나 피해를 입히려는 의도를 가진 봇입니다.

BCP

비즈니스 연속성 계획을 참조하십시오.

동작 그래프

리소스 동작과 시간 경과에 따른 상호 작용에 대한 통합된 대화형 뷰입니다. Amazon Detective에서 동작 그래프를 사용하여 실패한 로그온 시도, 의심스러운 API 호출 및 유사한 작업을 검사할 수 있습니다. 자세한 내용은 Detective 설명서의 Data in a behavior graph를 참조하십시오.

빅 엔디안 시스템

가장 중요한 바이트를 먼저 저장하는 시스템입니다. 엔디안도 참조하십시오.

바이너리 분류

바이너리 결과(가능한 두 클래스 중 하나)를 예측하는 프로세스입니다. 예를 들어, ML 모델이 “이 이메일이 스팸인가요, 스팸이 아닌가요?”, '이 제품은 책인가요, 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

서로 다르지만 동일한 환경을 두 개 만드는 배포 전략입니다. 현재 애플리케이션 버전을 한 환경 (파란색) 에서 실행하고 다른 환경 (녹색) 에서 새 애플리케이션 버전을 실행합니다. 이 전략을 사용하면 영향을 최소화하면서 신속하게 롤백할 수 있습니다.

bot

인터넷을 통해 자동화된 작업을 실행하고 사람의 활동이나 상호 작용을 시뮬레이션하는 소프트웨어 애플리케이션입니다. 인터넷에서 정보를 인덱싱하는 웹 크롤러와 같은 일부 봇은 유용하거나 유용합니다. 배드 봇으로 알려진 일부 다른 봇은 개인이나 조직을 방해하거나 피해를 입히기 위한 것입니다.

봇넷

멀웨어에감염되어 봇 허더 또는 봇 운영자로 알려진 단일 당사자의 통제 하에 있는 봇 네트워크. 봇넷은 봇과 그 영향을 확장하는 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 브랜치 정보 (문서) 를 참조하십시오. GitHub

브레이크 글래스 액세스

예외적인 상황에서 승인된 프로세스를 통해 사용자가 일반적으로 액세스 권한이 없는 데이터에 빠르게 액세스할 수 AWS 계정 있는 수단입니다. 자세한 내용은 Well-Architected AWS 지침의 브레이크 글래스 절차 구현 표시기를 참조하십시오.

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

클라우드 채택 프레임워크를 참조하십시오AWS .

카나리아 배포

최종 사용자에게 버전을 느리고 점진적으로 릴리스하는 것입니다. 확신이 들면 새 버전을 배포하고 현재 버전을 완전히 교체합니다.

CCoE

클라우드 센터 오브 엑설런스를 참조하십시오.

CDC

변경 데이터 캡처를 참조하십시오.

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 장애를 일으키는 이벤트를 발생시키는 행위 AWS Fault Injection Service (AWS FIS) 를 사용하여 AWS 워크로드에 스트레스를 주는 실험을 수행하고 응답을 평가할 수 있습니다.

CI/CD

지속적 통합 및 지속적 전달을 참조하십시오.

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.

클라이언트측 암호화

대상이 데이터를 AWS 서비스 수신하기 전에 데이터를 로컬로 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 기업 전략 블로그의 CCoE 게시물을 참조하십시오.

클라우드 컴퓨팅

원격 데이터 스토리지와 IoT 디바이스 관리에 일반적으로 사용되는 클라우드 기술 클라우드 컴퓨팅은 일반적으로 엣지 컴퓨팅 기술과 연결됩니다.

클라우드 운영 모델

IT 조직에서 하나 이상의 클라우드 환경을 구축, 성숙화 및 최적화하는 데 사용되는 운영 모델입니다. 자세한 내용은 클라우드 운영 모델 구축을 참조하십시오.

클라우드 채택 단계

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

- 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

Stephen Orban은 기업 전략 블로그의 클라우드 우선주의를 향한 여정 및 채택 단계에 대한 블로그 게시물에서 이러한 단계를 정의했습니다. AWS 클라우드 이들이 AWS 마이그레이션 전략과 어떤 관련이 있는지에 대한 자세한 내용은 마이그레이션 준비 가이드를 참조하십시오.

CMDB

구성 관리 데이터베이스를 참조하십시오.

코드 리포지토리

소스 코드와 설명서, 샘플, 스크립트 등의 기타 자산이 버전 관리 프로세스를 통해 저장되고 업데이트되는 위치입니다. 일반 클라우드 리포지토리에는 또는 이 포함됩니다 GitHub . AWS CodeCommit코드의 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 분야. 예를 들어 AWS Panorama 는 온프레미스 카메라 네트워크에 CV를 추가하는 디바이스를 제공하고, SageMaker Amazon은 CV용 이미지 처리 알고리즘을 제공합니다.

구성 드리프트

워크로드의 경우 구성이 예상 상태에서 변경됩니다. 이로 인해 워크로드가 규정을 준수하지 않게 될 수 있으며, 일반적으로 점진적이고 의도하지 않은 방식으로 진행됩니다.

구성 관리 데이터베이스(CMDB)

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

규정 준수 팩

AWS Config 규정 준수 및 보안 검사를 사용자 지정하기 위해 조합할 수 있는 규칙 및 수정 조치 모음입니다. YAML 템플릿을 사용하여 한 AWS 계정 및 지역 또는 조직 전체에 단일 엔티티로 적합성 팩을 배포할 수 있습니다. 자세한 내용은 설명서의 적합성 팩을 참조하십시오. AWS Config

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 이점을 참조하십시오. CD는 지속적 배포를 의미하기도 합니다. 자세한 내용은 지속적 전달(Continuous Delivery)과 지속적인 개발을 참조하십시오.

CV

컴퓨터 비전을 참조하십시오.

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 프레임워크의 보안 핵심 요소입니다. 자세한 내용은 데이터 분류를 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 거버넌스와 함께 분산되고 분산된 데이터 소유권을 제공하는 아키텍처 프레임워크입니다.

데이터 최소화

꼭 필요한 데이터만 수집하고 처리하는 원칙입니다. 에서 데이터 최소화를 실천하면 개인 정보 보호 위험, 비용 및 분석에 따른 탄소 발자국을 줄일 AWS 클라우드 수 있습니다.

데이터 경계

신뢰할 수 있는 ID만 예상 네트워크에서 신뢰할 수 있는 리소스에 액세스하도록 하는 데 도움이 되는 AWS 환경 내 일련의 예방 가드레일입니다. 자세한 내용은 데이터 경계 구축을 참조하십시오. 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 시스템. 데이터 웨어하우스에는 일반적으로 대량의 과거 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

데이터베이스 정의 언어를 참조하십시오.

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.

defense-in-depth

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 AWS여러 컨트롤을 추가하여 리소스를 보호하는 데 도움이 됩니다. 예를 들어 다단계 인증, 네트워크 세분화, 암호화를 결합한 defense-in-depth 접근 방식을 사용할 수 있습니다.

위임된 관리자

에서 AWS Organizations호환 가능한 서비스는 AWS 구성원 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 Organizations와 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

환경을 참조하십시오.

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 Implementing security controls on AWS의 Detective controls를 참조하십시오.

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.

치수 표

스타 스키마에서 팩트 테이블의 양적 데이터에 대한 데이터 속성을 포함하는 작은 테이블입니다. 차원 테이블 속성은 일반적으로 텍스트처럼 동작하는 텍스트 필드 또는 불연속형 숫자입니다. 이러한 속성은 일반적으로 쿼리 제한, 필터링 및 결과 집합 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)

재해로 인한 다운타임과 데이터 손실을 최소화하기 위해 사용하는 전략과 프로세스입니다. 자세한 내용은 워크로드의 재해 복구 AWS: AWS Well-Architected 프레임워크에서의 클라우드 복구를 참조하십시오.

DML

데이터베이스 조작 언어를 참조하십시오.

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 방법을 참조하십시오.

DR

재해 복구를 참조하십시오.

드리프트 감지

기준 구성으로부터의 편차 추적. 예를 들어 시스템 리소스의 편차를 감지하는 AWS CloudFormation 데 사용하거나 거버넌스 요구 사항 준수에 영향을 미칠 수 있는 착륙 지대의 변경 사항을 탐지하는 AWS Control Tower 데 사용할 수 있습니다.

DVSM

개발 가치 흐름 매핑을 참조하십시오.

E

EDA

탐색적 데이터 분석을 참조하십시오.

엣지 컴퓨팅

IoT 네트워크의 엣지에서 스마트 디바이스의 컴퓨팅 성능을 개선하는 기술 클라우드 컴퓨팅과 비교할 때 엣지 컴퓨팅은 통신 대기 시간을 줄이고 응답 시간을 개선할 수 있습니다.

암호화

사람이 읽을 수 있는 일반 텍스트 데이터를 암호문으로 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 엔드포인트를 참조하십시오.

엔드포인트 서비스

Virtual Private Cloud(VPC)에서 호스팅하여 다른 사용자와 공유할 수 있는 서비스입니다. 다른 주체 AWS 계정 또는 AWS Identity and Access Management (IAM) 보안 주체에 권한을 부여하여 엔드포인트 서비스를 생성하고 권한을 부여할 수 있습니다. AWS PrivateLink 이러한 계정 또는 보안 주체는 인터페이스 VPC 엔드포인트를 생성하여 엔드포인트 서비스에 비공개로 연결할 수 있습니

다. 자세한 내용은 Amazon Virtual Private Cloud(VPC) 설명서의 엔드포인트 서비스 생성을 참조하십시오.

ERP (전사적 자원 관리)

기업의 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 암호화를 참조하십시오.

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, 사고 대응 등이 포함됩니다. AWS 마이그레이션 전략의 에픽에 대한 자세한 내용은 프로그램 구현 가이드를 참조하십시오.

ERP

엔터프라이즈 리소스 계획을 참조하십시오.

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 스키마의 중앙 테이블. 비즈니스 운영에 대한 정량적 데이터를 저장합니다. 일반적으로 팩트 테이블에는 측정값이 포함된 열과 차원 테이블의 외부 키가 포함된 열 등 두 가지 유형의 열이 포함됩니다.

빨리 실패하세요

빈번하고 점진적인 테스트를 통해 개발 라이프사이클을 단축하는 철학. 이는 애자일 접근 방식의 중요한 부분입니다.

장애 격리 경계

장애 영향을 제한하고 워크로드의 복원력을 개선하는 데 도움이 되는 가용 영역 AWS 리전, 컨트롤 플레인 또는 데이터 플레인과 같은 경계 AWS 클라우드자세한 내용은 AWS 장애 격리 경계를 참조하십시오.

기능 브랜치

브랜치를 참조하십시오.

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 그레디언트와 같은 다양한 기법을 통해 계산할 수 있는 수치 점수로 표현됩니다. 자세한 내용은 다음을AWS사용한 기계 학습 모델 해석 가능성을 참조하십시오.

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.

FGAC

세분화된 액세스 제어를 참조하십시오.

세분화된 액세스 제어(FGAC)

여러 조건을 사용하여 액세스 요청을 허용하거나 거부합니다.

플래시컷 마이그레이션

단계별 접근 방식 대신 변경 데이터 캡처를 통한 지속적인 데이터 복제를 통해 최단 시간에 데이터를 마이그레이션하는 데이터베이스 마이그레이션 방법입니다. 목표는 가동 중지 시간을 최소화하는 것입니다.

G

지리적 차단

지리적 제한을 참조하십시오.

지리적 제한(지리적 차단)

CloudFrontAmazon에서는 특정 국가의 사용자가 콘텐츠 배포에 액세스하지 못하도록 하는 옵션을 제공합니다. 허용 목록 또는 차단 목록을 사용하여 승인된 국가와 차단된 국가를 지정할 수 있습니다. 자세한 내용은 설명서의 콘텐츠의 지리적 배포 제한을 참조하십시오. CloudFront

Gitflow 워크플로

하위 환경과 상위 환경이 소스 코드 리포지토리의 서로 다른 브랜치를 사용하는 방식입니다. Gitflow 워크플로는 레거시로 간주되며 트렁크 기반 워크플로는 현대적이고 선호되는 접근 방식입니다.

브라운필드 전략

새로운 환경에서 기존 인프라의 부재 시스템 아키텍처에 대한 그린필드 전략을 채택할 때 브라운필드라고도 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, 이들은, Amazon AWS Config AWS Security Hub GuardDuty AWS Trusted Advisor, Amazon Inspector 및 사용자 지정 AWS Lambda 검사를 사용하여 구현됩니다.

H

하

고가용성을 확인하세요.

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 SCT를 제공합니다.

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.

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 리플랫포밍 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. 긴급성 때문에 핫픽스는 일반적으로 일반적인 DevOps 릴리스 워크플로 외부에서 만들어집니다.

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

인프라를 코드로 보세요.

자격 증명 기반 정책

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

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 인터넷을 참조하십시오.

불변의 인프라

기존 인프라를 업데이트, 패치 또는 수정하는 대신 프로덕션 워크로드용 새 인프라를 배포하는 모델입니다. 변경 불가능한 인프라는 기본적으로 변경 가능한 인프라보다 더 일관되고 안정적이며 예측 가능합니다. 자세한 내용은 Well-Architected AWS 프레임워크의 변경 불가능한 인프라를 사용한 배포 모범 사례를 참조하십시오.

인바운드(수신) VPC

AWS 다중 계정 아키텍처에서 VPC는 애플리케이션 외부에서 네트워크 연결을 허용, 검사 및 라우팅합니다. AWS Security Reference Architecture에서는 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 것

이 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

Klaus Schwab이 연결성, 실시간 데이터, 자동화, 분석 및 AI/ML의 발전을 통한 제조 프로세스의 현대화를 지칭하기 위해 2016년 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 구축을 참조하십시오.

검사 VPC

AWS 다중 계정 아키텍처에서 VPC (동일하거나 AWS 리전다른), 인터넷 및 온프레미스 네트워크 간의 네트워크 트래픽 검사를 관리하는 중앙 집중식 VPC입니다. AWS Security Reference Architecture에서는 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 Machine learning model interpretability with AWS를 참조하십시오.

IoT

사물 인터넷을 참조하십시오.

IT 정보 라이브러리(TIL)

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

IT 서비스 관리(TSM)

조직의 IT 서비스 설계, 구현, 관리 및 지원과 관련된 활동 클라우드 운영을 ITSM 도구와 통합하는 방법에 대한 자세한 내용은 운영 통합 가이드를 참조하십시오.

ITIL

IT 정보 라이브러리를 참조하십시오.

ITSM

IT 서비스 관리를 참조하십시오.

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

Landing Zone은 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 설정을 참조하십시오.

대규모 마이그레이션

300대 이상의 서버 마이그레이션입니다.

LBAC

레이블 기반 액세스 제어를 참조하십시오.

최소 권한

작업을 수행하는 데 필요한 최소 권한을 부여하는 보안 모범 사례입니다. 자세한 내용은 IAM 설명서의 최소 권한 적용을 참조하십시오.

리프트 앤드 시프트

7 R을 참조하십시오.

리틀 엔디안 시스템

가장 덜 중요한 바이트를 먼저 저장하는 시스템입니다. 엔디안 참조.

하위 환경

환경 참조.

M

기계 학습(ML)

패턴 인식 및 학습에 알고리즘과 기법을 사용하는 인공 지능의 한 유형입니다. ML은 사물 인터넷(IoT) 데이터와 같은 기록된 데이터를 분석하고 학습하여 패턴을 기반으로 통계 모델을 생성합니다. 자세한 내용은 기계 학습을 참조하십시오.

기본 브랜치

브랜치 참조.

악성 코드

컴퓨터 보안 또는 개인 정보를 침해하도록 설계된 소프트웨어 멀웨어는 컴퓨터 시스템을 방해하거나, 민감한 정보를 유출하거나, 무단 액세스를 얻을 수 있습니다. 멀웨어의 예로는 바이러스, 웜, 랜섬웨어, 트로이 목마, 스파이웨어, 키로거 등이 있습니다.

매니지드 서비스

AWS 서비스 인프라 계층, 운영 체제 및 플랫폼을 AWS 운영하며 사용자는 엔드포인트에 액세스하여 데이터를 저장하고 검색합니다. 관리형 서비스의 예로는 아마존 심플 스토리지 서비스 (Amazon S3) 와 아마존 DynamoDB가 있습니다. 이러한 서비스를 추상화된 서비스라고도 합니다.

제조 실행 시스템 (MES)

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

MAP

Migration Acceleration 프로그램을 참조하십시오.

기구

도구를 만들고 도구 채택을 유도한 다음 결과를 검토하여 조정하는 전체 프로세스입니다. 메커니즘은 작동하면서 자체적으로 강화되고 개선되는 사이클입니다. 자세한 내용은 AWS Well-Architected 프레임워크에서의 메커니즘 구축을 참조하십시오.

멤버 계정

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

MES

제조 실행 시스템을 참조하십시오.

메시지 큐 텔레메트리 전송 (MQTT)

퍼블리시/구독 패턴을 기반으로 하는 리소스가 제한된 IoT 디바이스를 위한 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 서버리스 서비스를 사용하여 마이크로서비스 통합을 참조하십시오. AWS

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 마이크로서비스 구현을 참조하십시오. AWS

Migration Acceleration Program(MAP)

조직이 클라우드로 전환하기 위한 강력한 운영 기반을 구축하고 초기 마이그레이션 비용을 상쇄할 수 있도록 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 전략의 세 번째 단계입니다.

마이그레이션 팩토리

자동화되고 민첩한 접근 방식을 통해 워크로드 마이그레이션을 간소화하는 다기능 팀입니다. 마이그레이션 팩토리 팀에는 일반적으로 운영, 비즈니스 분석가 및 소유자, 마이그레이션 엔지니어, 개발자 및 스프린트에서 일하는 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 가이드와 마이그레이션 팩토리에 대한 설명을 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, 계정 등이 있습니다. AWS

마이그레이션 패턴

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

Migration Portfolio Assessment(MPA)

로 마이그레이션하기 위한 비즈니스 사례를 검증하기 위한 정보를 제공하는 온라인 도구입니다. AWS 클라우드 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 컨설턴트와 APN 파트너 AWS 컨설턴트에게 무료로 제공됩니다.

마이그레이션 준비 상태 평가(MRA)

CAF를 사용하여 조직의 클라우드 준비 상태에 대한 통찰력을 얻고, 강점과 약점을 파악하고, 식별된 격차를 해소하기 위한 실행 계획을 수립하는 프로세스입니다. AWS 자세한 내용은 마이그레이션 준비 가이드를 참조하십시오. MRA는 AWS 마이그레이션 전략의 첫 번째 단계입니다.

마이그레이션 전략

워크로드를 로 마이그레이션하는 데 사용된 접근 방식. AWS 클라우드자세한 내용은 이 용어집의 7R 항목 및 대규모 마이그레이션 가속화를 위한 조직 동원을 참조하십시오.

ML

기계 학습을 참조하십시오.

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 의 AWS 클라우드애플리케이션 현대화 전략을 참조하십시오.

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 에서 애플리케이션의 현대화 준비 상태 평가를 참조하십시오. AWS 클라우드

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 분해를 참조하십시오.

MPA

마이그레이션 포트폴리오 평가를 참조하십시오.

MQTT

메시지 큐 원격 분석 전송을 참조하십시오.

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 워크로드를 위해 기존 인프라를 업데이트하고 수정하는 모델입니다. 일관성, 안정성 및 예측 가능성을 개선하기 위해 AWS Well-Architected Framework는 변경 불가능한 인프라를 모범 사례로 사용할 것을 권장합니다.

O

OAC

원본 액세스 제어를 참조하십시오.

좋아요

원본 액세스 ID를 참조하십시오.

OCM

조직 변경 관리를 참조하십시오.

오프라인 마이그레이션

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

OI

운영 통합을 참조하십시오.

안녕하세요.

운영 수준 계약을 참조하십시오.

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

오픈 프로세스 커뮤니케이션 - 통합 아키텍처를 참조하십시오.

오픈 프로세스 커뮤니케이션 - 통합 아키텍처 (OPC-UA)

산업 machine-to-machine 자동화를 위한 (M2M) 통신 프로토콜. OPC-UA는 데이터 암호화, 인증 및 권한 부여 체계와 함께 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 검토 (ORR)

인시던트 및 발생 가능한 실패의 범위를 이해, 평가, 예방 또는 줄이는 데 도움이 되는 질문 및 관련 모범 사례로 구성된 체크리스트입니다. 자세한 내용은 Well-Architected AWS 프레임워크의 운영 준비 상태 검토 (ORR) 를 참조하십시오.

운영 기술 (OT)

물리적 환경과 함께 작동하여 산업 운영, 장비 및 인프라를 제어하는 하드웨어 및 소프트웨어 시스템. 제조 분야에서는 OT와 정보 기술 (IT) 시스템의 통합이 인더스트리 4.0 혁신의 핵심 초점입니다.

운영 통합(OI)

클라우드에서 운영을 현대화하는 프로세스로 준비 계획, 자동화 및 통합을 수반합니다. 자세한 내용은 운영 통합 가이드를 참조하십시오.

조직 트레일

이를 통해 AWS CloudTrail 생성되는 트레일은 조직 AWS 계정 내 모든 사용자의 모든 이벤트를 기록합니다. AWS Organizations이 트레일은 조직에 속한 각 AWS 계정 에 생성되고 각 계정의 활동을 추적합니다. 자세한 내용은 CloudTrail 설명서에서 조직을 위한 트레일 만들기를 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. 클라우드 채택 프로젝트에 필요한 변화 속도 때문에 AWS 마이그레이션 전략에서는 이 프레임워크를 사용자 가속화라고 합니다. 자세한 내용은 사용 가이드를 참조하십시오.

오리진 액세스 제어(OAC)

CloudFront에서는 Amazon Simple Storage Service (Amazon S3) 콘텐츠의 보안을 위해 액세스를 제한하는 향상된 옵션을 제공합니다. OAC는 모든 S3 버킷 AWS 리전, AWS KMS (SSE-KMS) 를 사용한 서버 측 암호화, S3 버킷에 대한 동적 및 요청을 모두 지원합니다. PUT DELETE

오리진 액세스 ID(OAI)

CloudFront에서는 Amazon S3 콘텐츠 보안을 위해 액세스를 제한하는 옵션입니다. OAI를 사용하면 Amazon S3가 인증할 수 있는 보안 주체를 CloudFront 생성합니다. 인증된 보안 주체는 특정 배

포를 통해서만 S3 버킷의 콘텐츠에 액세스할 수 있습니다. CloudFront 더 세분화되고 향상된 액세스 제어를 제공하는 OAC도 참조하십시오.

또는

운영 준비 상태 검토를 참조하십시오.

아니요

운영 기술을 참조하십시오.

아웃바운드(송신) VPC

AWS 다중 계정 아키텍처에서 애플리케이션 내에서 시작되는 네트워크 연결을 처리하는 VPC입니다. AWS Security Reference Architecture에서는 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 경계를 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 정보를 참조하십시오.

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

프로그래머블 로직 컨트롤러를 참조하십시오.

PLM

제품 라이프사이클 관리를 참조하십시오.

정책

권한을 정의 (ID 기반 정책 참조) 하거나, 액세스 조건을 지정 (리소스 기반 정책 참조) 하거나, 조직 내 모든 계정에 대한 최대 권한을 정의 AWS Organizations (서비스 제어 정책 참조) 할 수 있는 개체입니다.

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 스토어를 사용하면 마이크로서비스를 더 쉽게 구현하고 성능과 확장성을 높일 수 있습니다. 자세한 내용은 마이크로서비스에서 데이터 지속성 활성화를 참조하십시오.

포트폴리오 평가

마이그레이션을 계획하기 위해 애플리케이션 포트폴리오를 검색 및 분석하고 우선순위를 정하는 프로세스입니다. 자세한 내용은 마이그레이션 준비 상태 평가를 참조하십시오.

조건자

일반적으로 조항에 있는 `true` `false` OR를 반환하는 쿼리 조건입니다. `WHERE`

조건부 푸시다운

전송하기 전에 쿼리의 데이터를 필터링하는 데이터베이스 쿼리 최적화 기법입니다. 이렇게 하면 관계형 데이터베이스에서 검색하고 처리해야 하는 데이터의 양이 줄어들고 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 controls를 참조하십시오.

보안 주체

작업을 수행하고 리소스에 액세스할 수 AWS 있는 엔티티 이 엔티티는 일반적으로 IAM 역할의 루트 사용자 또는 사용자입니다. AWS 계정자세한 내용은 IAM 설명서의 역할 용어 및 개념의 보안 주체를 참조하십시오.

개인 정보 보호 중심 설계

전체 엔지니어링 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 작업을 참조하십시오.

사전 예방 제어

규정을 준수하지 않는 리소스의 배포를 방지하도록 설계된 보안 제어입니다. 이러한 컨트롤은 리소스를 프로비저닝하기 전에 리소스를 스캔합니다. 리소스가 컨트롤과 호환되지 않으면 프로비저닝되지 않습니다. 자세한 내용은 AWS Control Tower 설명서의 컨트롤 참조 안내서를 참조하고 보안 제어 구현의 사전 제어를 참조하십시오. AWS

제품 라이프사이클 관리 (PLM)

설계, 개발, 출시부터 성장 및 성숙도, 폐기 및 제거에 이르는 전체 라이프사이클에 걸쳐 제품에 대한 데이터 및 프로세스를 관리하는 것입니다.

프로덕션 환경

환경을 참조하십시오.

프로그래머블 로직 컨트롤러 (PLC)

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

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 (게시/구독)

마이크로서비스 간의 비동기 통신을 통해 확장성과 응답성을 개선할 수 있는 패턴입니다. 예를 들어 마이크로서비스 기반 MES에서 마이크로서비스는 다른 마이크로서비스가 구독할 수 있는 채널에 이벤트 메시지를 게시할 수 있습니다. 시스템은 게시 서비스를 변경하지 않고도 새 마이크로서비스를 추가할 수 있습니다.

Q

쿼리 계획

SQL 관계형 데이터베이스 시스템의 데이터에 액세스하는 데 사용되는 일련의 단계 (예: 지침).

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

RACI (책임, 책임, 상담, 정보 제공) 를 참조하십시오.

랜섬웨어

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

RASCI 매트릭스

책임, 책임, 상담, 정보 제공 (RACI) 을 참조하십시오.

RCAC

행 및 열 액세스 제어를 참조하십시오.

읽기 전용 복제본

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

재설계

7 R을 참조하십시오.

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

7 R을 참조하십시오.

리전

지리적 AWS 영역별 리소스 모음. AWS 리전 각각은 격리되어 있고 서로 독립적이므로 내결함성, 안정성 및 복원력을 제공합니다. 자세한 내용은 사용할 수 있는 AWS 리전 계정 지정을 참조하십시오.

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

7 R을 참조하십시오.

release

배포 프로세스에서 변경 사항을 프로덕션 환경으로 승격시키는 행위입니다.

고쳐 놓다

7 R을 참조하십시오.

리플랫포밍

7 R을 참조하십시오.

환매

7 R을 참조하십시오.

복원력

장애를 견디거나 장애를 복구할 수 있는 애플리케이션의 능력 고가용성 및 재해 복구는 복원력을 계획할 때 일반적으로 고려해야 할 사항입니다. AWS 클라우드자세한 내용은 AWS 클라우드 복원력을 참조하십시오.

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 결

정권자 (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 Implementing security controls on AWS의 Responsive controls를 참조하십시오.

retain

7 R을 참조하십시오.

은퇴

7 R을 참조하십시오.

회전

공격자가 자격 증명에 액세스하는 것을 더 어렵게 만들기 위해 암호를 주기적으로 업데이트하는 프로세스입니다.

행 및 열 액세스 제어(RCAC)

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

RPO

복구 지점 목표를 참조하십시오.

RTO

복구 시간 목표를 참조하십시오.

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 (IdPs) 가 사용하는 개방형 표준입니다. 이 기능을 사용하면 페더레이션 싱글 사인온 (SSO) 이 가능하므로 조직의 모든 사용자를 위해 IAM에서 사용자를 생성하지 않고도 사용자가 AWS API 작업에 AWS Management Console 로그인하거나 API 작업을 호출할 수 있습니다.

SAML 2.0 기반 페더레이션에 대한 자세한 내용은 IAM 설명서의 SAML 2.0 기반 페더레이션 정보를 참조하십시오.

SCADA

감독 제어 및 데이터 수집을 참조하십시오.

SCP

서비스 제어 정책을 참조하십시오.

secret

에는 AWS Secrets Manager암호화된 형태로 저장하는 비밀번호나 사용자 자격 증명과 같은 기밀 또는 제한된 정보. 비밀 값과 해당 메타데이터로 구성됩니다. 비밀 값은 바이너리, 단일 문자열 또는 여러 문자열일 수 있습니다. 자세한 내용은 Secrets Manager 시크릿에는 무엇이 들어 있나요? 를 참조하십시오. Secrets Manager 설명서에서 확인할 수 있습니다.

보안 제어

위협 행위자가 보안 취약성을 악용하는 능력을 방지, 탐지 또는 감소시키는 기술적 또는 관리적 가드레일입니다. 보안 제어에는 예방적, 탐정적, 대응적, 사전 예방적 제어의 네 가지 기본 유형이 있습니다.

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 대응 자동화

보안 이벤트에 자동으로 대응하거나 보안 이벤트를 해결하도록 설계된 사전 정의되고 프로그래밍된 조치입니다. 이러한 자동화는 보안 모범 사례를 구현하는 데 도움이 되는 탐지 또는 대응형 보안 제어 역할을 합니다. AWS 자동 응답 조치의 예로는 VPC 보안 그룹 수정, Amazon EC2 인스턴스 패치, 자격 증명 교체 등이 있습니다.

서버 측 암호화

수신자에 의한 목적지의 데이터 암호화 AWS 서비스

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 작업을 지정할 수 있습니다. 자세한 내용은 AWS Organizations 설명서의 서비스 제어 정책을 참조하십시오.

서비스 엔드포인트

의 진입점 URL입니다 AWS 서비스. 엔드포인트를 사용하여 대상 서비스에 프로그래밍 방식으로 연결할 수 있습니다. 자세한 내용은 AWS 일반 참조의 AWS 서비스 엔드포인트를 참조하십시오.

서비스 수준에 관한 계약(SLA)

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

서비스 수준 표시기 (SLI)

오류율, 가용성 또는 처리량과 같은 서비스의 성능 측면을 측정하는 것입니다.

서비스 수준 목표 (SLO)

서비스 수준 지표로 측정되는 서비스 상태를 나타내는 대상 지표입니다.

공동 책임 모델

클라우드 보안 및 규정 준수에 AWS 대한 책임을 공유하는 것을 설명하는 모델입니다. AWS 클라우드의 보안을 책임지는 반면, 사용자는 클라우드에서의 보안을 담당합니다. 자세한 내용은 공동 책임 모델을 참조하십시오.

시엠

보안 정보 및 이벤트 관리 시스템을 참조하십시오.

단일 장애 지점 (SPOF)

응용 프로그램의 중요한 단일 구성 요소에서 발생한 오류로 인해 시스템이 중단될 수 있습니다.

SLA

SLA (서비스 수준 계약) 를 참조하십시오.

SLI

서비스 수준 표시기 참조.

SLO

서비스 수준 목표를 참조하십시오.

split-and-seed 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 의 애플리케이션 현대화를 위한 단계별 접근 방식을 참조하십시오. AWS 클라우드

SPOF

단일 장애 지점 보기.

스타 스키마

하나의 큰 팩트 테이블을 사용하여 트랜잭션 또는 측정 데이터를 저장하고 하나 이상의 작은 차원 테이블을 사용하여 데이터 속성을 저장하는 데이터베이스 구성 구조입니다. 이 구조는 데이터 웨어하우스에서 사용하거나 비즈니스 인텔리전스 용도로 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 Fowler가 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 방법을 참조하십시오.

서브넷

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

감독 통제 및 데이터 수집 (SCADA)

제조 시 하드웨어와 소프트웨어를 사용하여 물리적 자산과 생산 작업을 모니터링하는 시스템입니다.

대칭 암호화

동일한 키를 사용하여 데이터를 암호화하고 복호화하는 암호화 알고리즘입니다.

합성 테스트

잠재적 문제를 감지하거나 성능을 모니터링하기 위해 사용자 상호 작용을 시뮬레이션하는 방식으로 시스템을 테스트합니다. Amazon CloudWatch Synthetics를 사용하여 이러한 테스트를 생성할 수 있습니다.

T

tags

리소스 구성을 위한 메타데이터 역할을 하는 키-값 쌍. AWS 태그를 사용하면 리소스를 손쉽게 관리, 식별, 정리, 검색 및 필터링할 수 있습니다. 자세한 내용은 AWS 리소스에 태그 지정을 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

환경을 참조하십시오.

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.

전송 게이트웨이

VPC와 온프레미스 네트워크를 상호 연결하는 데 사용할 수 있는 네트워크 전송 허브입니다. 자세한 내용은 AWS Transit Gateway 설명서의 트랜짓 게이트웨이란 무엇입니까? 를 참조하십시오.

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

조직 내 AWS Organizations 및 해당 계정에서 사용자를 대신하여 작업을 수행하도록 지정한 서비스에 권한 부여 신뢰할 수 있는 서비스는 필요할 때 각 계정에 서비스 연결 역할을 생성하여 관

리 작업을 수행합니다. 자세한 내용은 AWS Organizations 설명서의 다른 AWS 서비스와 AWS Organizations 함께 사용을 참조하십시오.

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 판만 들고 배불리 먹을 수 있는 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. 자세한 내용은 Quantifying uncertainty in deep learning systems 가이드를 참조하십시오.

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

환경을 보세요.

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

웜 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.

윈도우 함수

현재 레코드와 어떤 식으로든 관련된 행 그룹에 대해 계산을 수행하는 SQL 함수입니다. 윈도우 함수는 이동 평균을 계산하거나 현재 행의 상대적 위치를 기반으로 행 값에 액세스하는 등의 작업을 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.

웜

한 번 쓰고, 많이 읽으세요.

WQF

AWS 워크로드 검증 프레임워크를 참조하십시오.

한 번 작성하고 여러 번 읽기 (WORM)

데이터를 한 번 쓰고 데이터가 삭제되거나 수정되지 않도록 하는 스토리지 모델입니다. 인증된 사용자는 필요한 만큼 데이터를 여러 번 읽을 수 있지만 변경할 수는 없습니다. 이 데이터 스토리지 인프라는 변경할 수 없는 것으로 간주됩니다.

Z

제로데이 익스플로잇

제로데이 취약점을 악용하는 공격 (일반적으로 멀웨어) 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.

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.