



SUMMIT
ONLINE

EMR 플랫폼 기반의 Spark 워크로드 실행 최적화 방안

정세웅
솔루션즈 아키텍트
AWS Korea

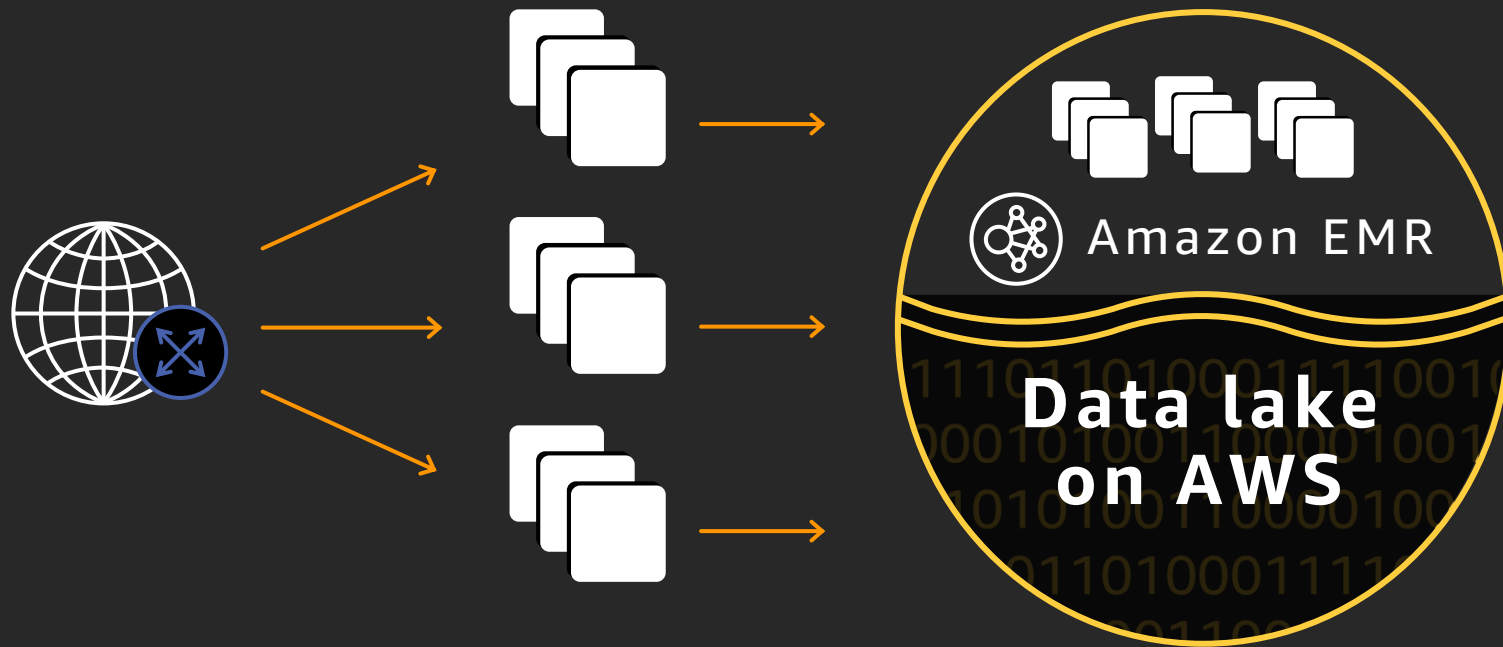
Agenda

- Spark 워크로드 최적화
- Spark on EMR 성능 최적화 방안
- EMR Runtime for Apache Spark
- Apache Hudi - 레코드 레벨의 데이터 Update, delete, insert
- EMR Managed Resize
- Spark Job 디버그 및 모니터링을 위한 Off-cluster Spark Log 관리
- Lake Formation 과 EMR 통합
- Docker 환경에서 Spark 어플리케이션 배포 가능

Spark on EMR

관리형 하둡 플랫폼 - Amazon EMR

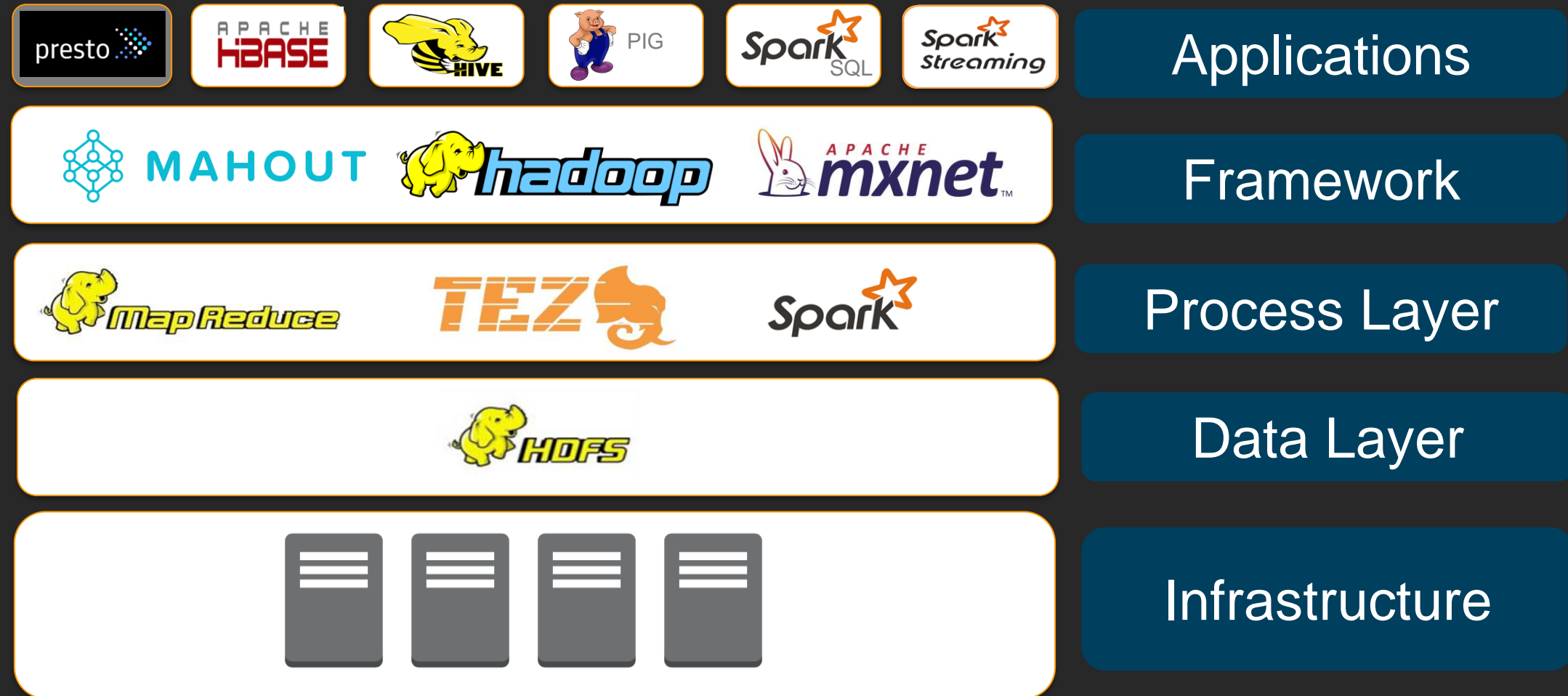
Analytics and ML at scale



- 컴퓨팅 리소스(Amazon EMR)와 저장공간(Amazon S3)의 분리
- PB에서 EB 규모의 데이터까지 저장 / 처리 가능
- 필요한 수 만큼의 노드로 확장 가능
- 최신 Open-source 어플리케이션
- AWS내 보안 기능과 통합
- 자유롭게 커스터마이징, 접근 가능
- 오토 스케일을 통해 Elastic 한 구성
- 초당 과금을 통한 가격 절감

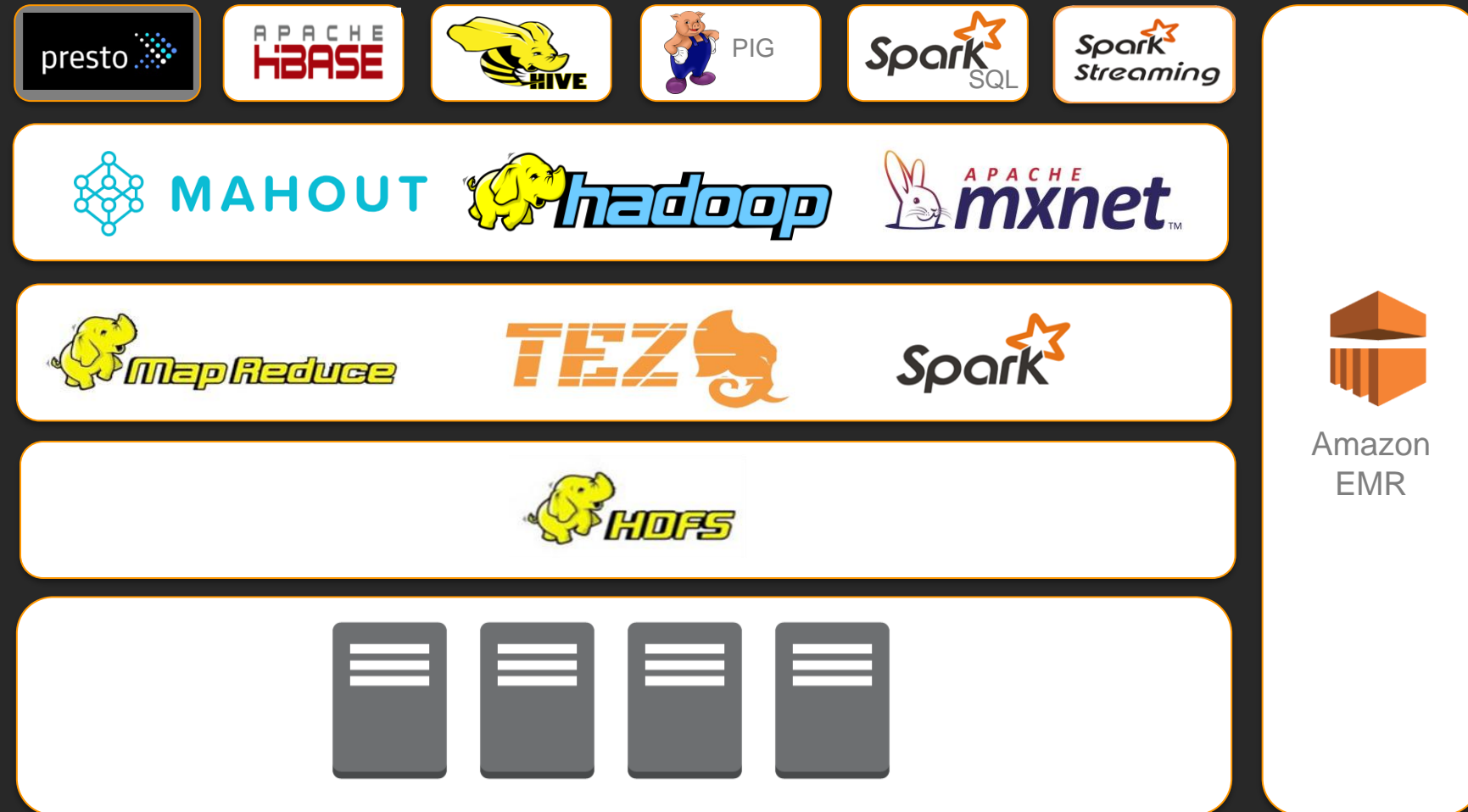
Amazon EMR - Hadoop

엔터프라이즈 레벨의 Hadoop 플랫폼



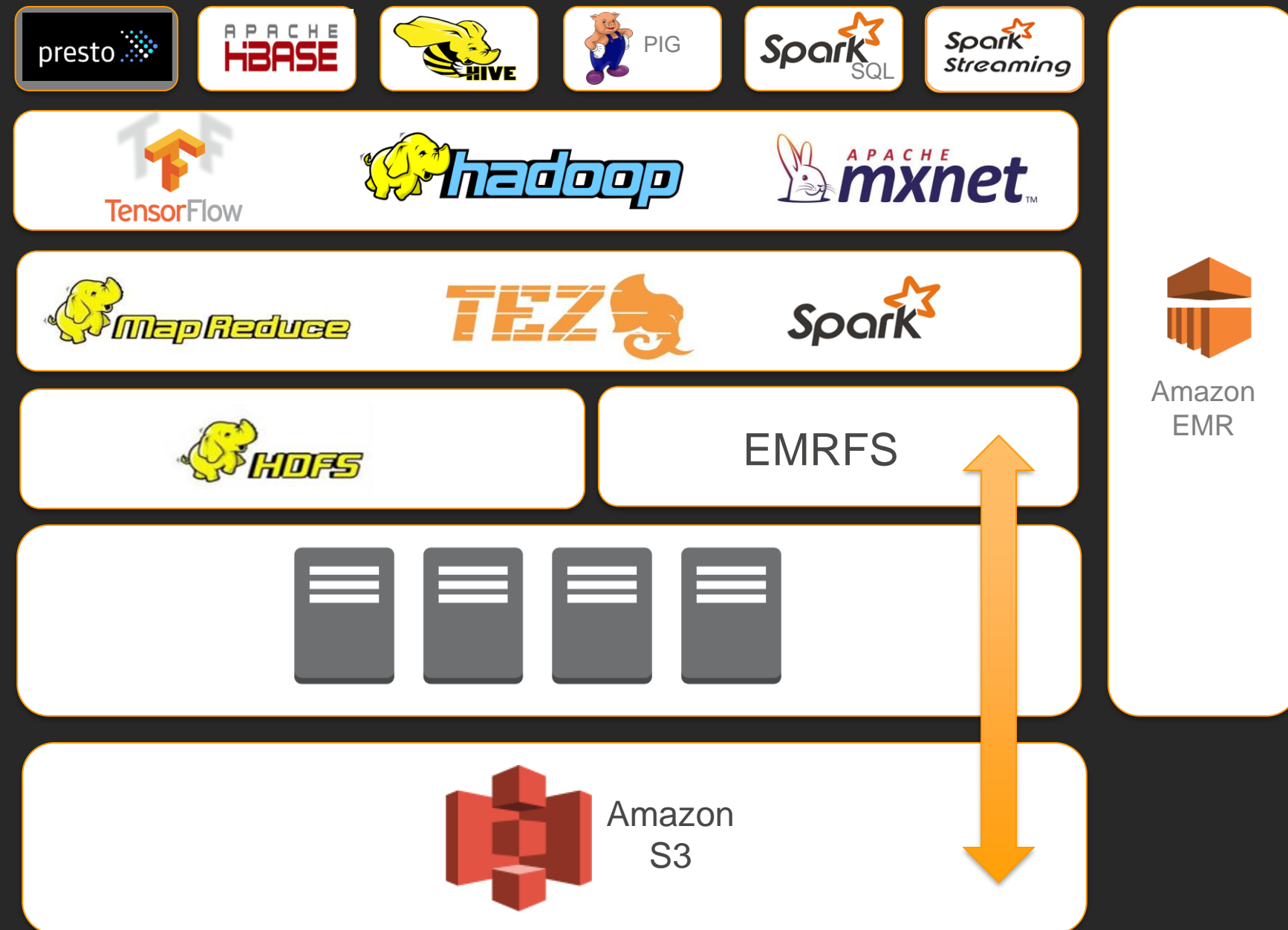
Amazon EMR - Hadoop

엔터프라이즈 레벨의 Hadoop 플랫폼



Amazon EMR - Hadoop

엔터프라이즈 레벨의 Hadoop 플랫폼



EMR 노드 구성

마스터 노드

클러스터 관리

NameNode와 JobTracker 포함

코어 노드

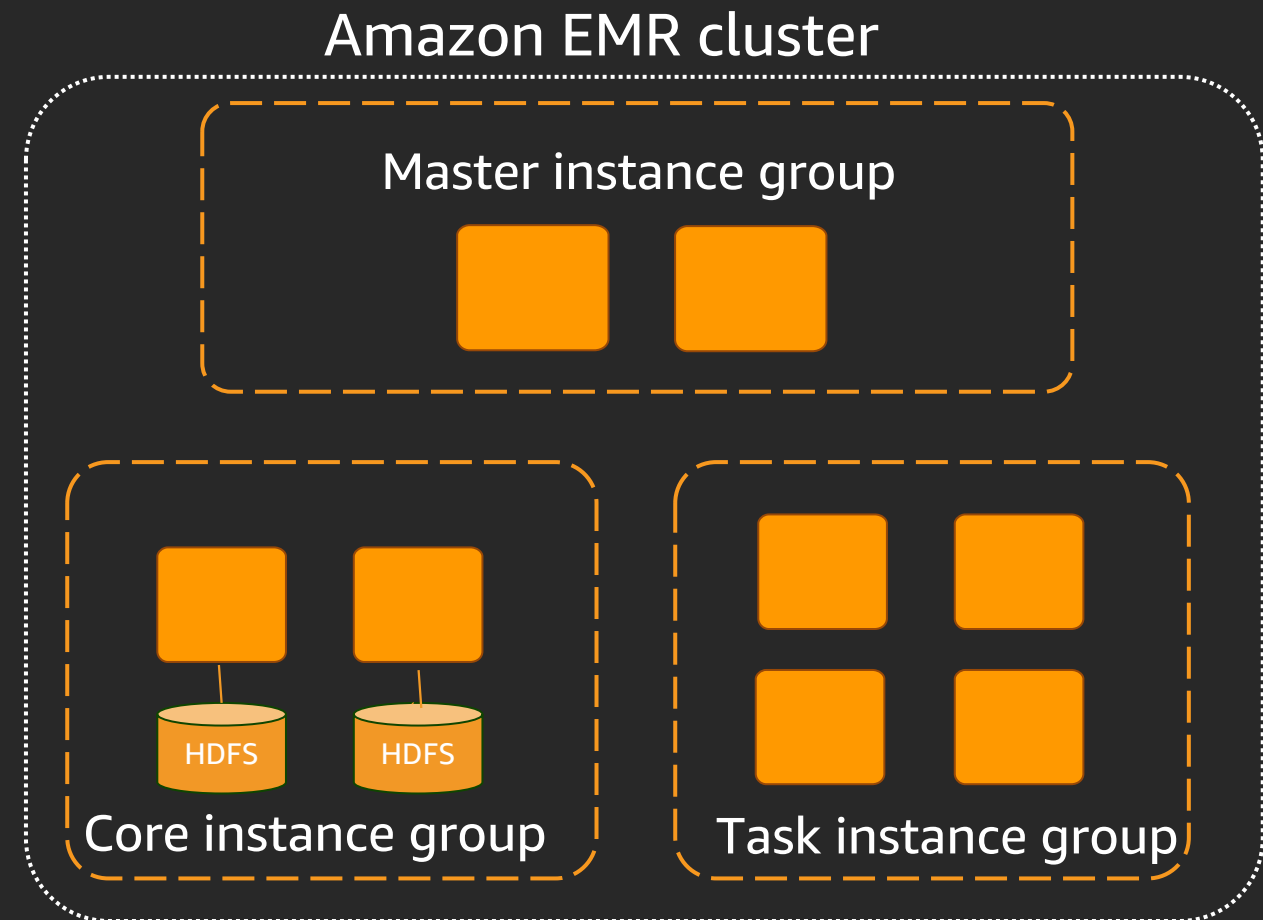
작업 실행을 위한 Task tracker

하둡에서의 DataNode

테스트 노드

Task tracker만 설치

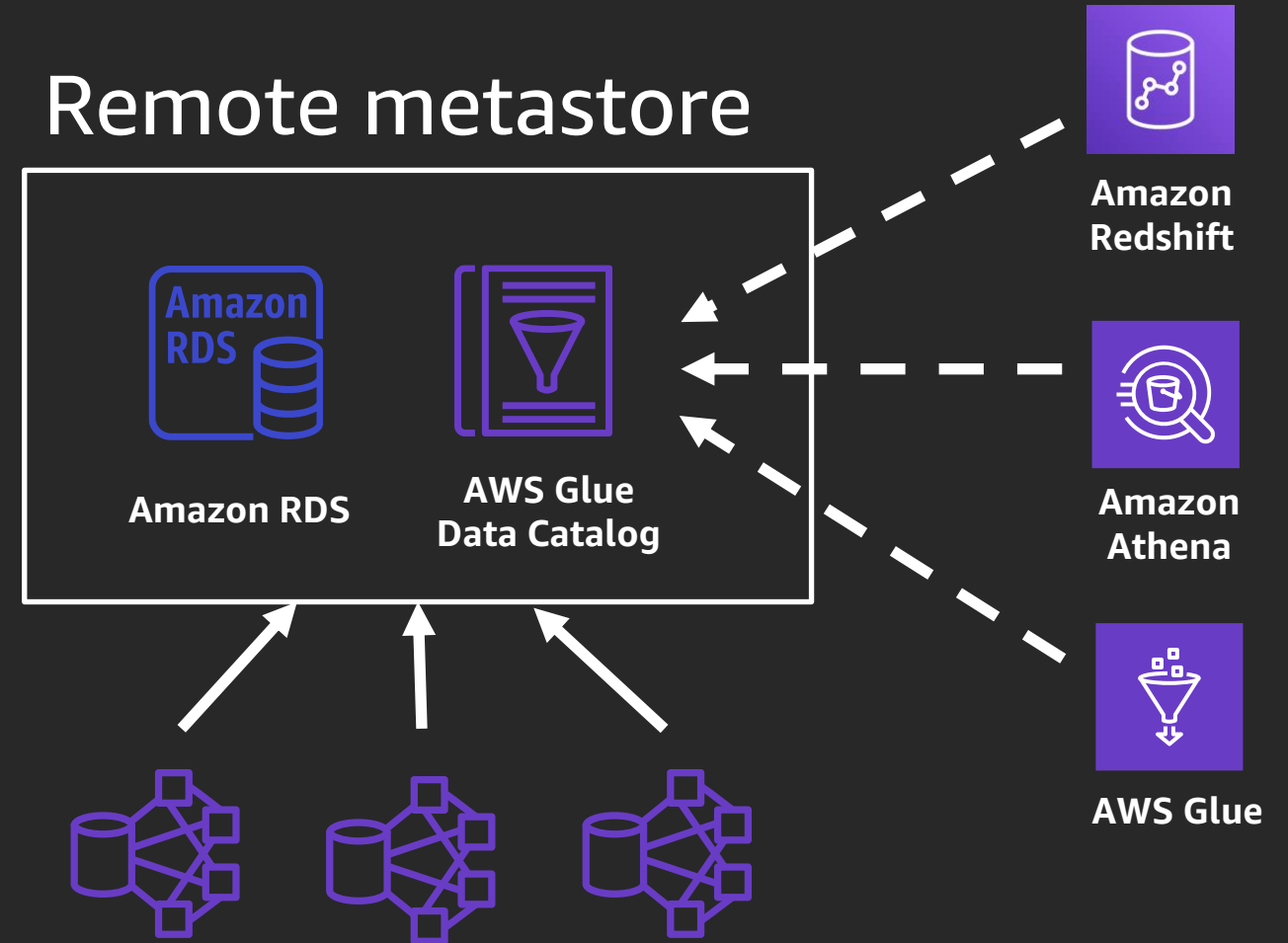
로컬 HDFS 없음



Stateless 클러스터 아키텍처 권장

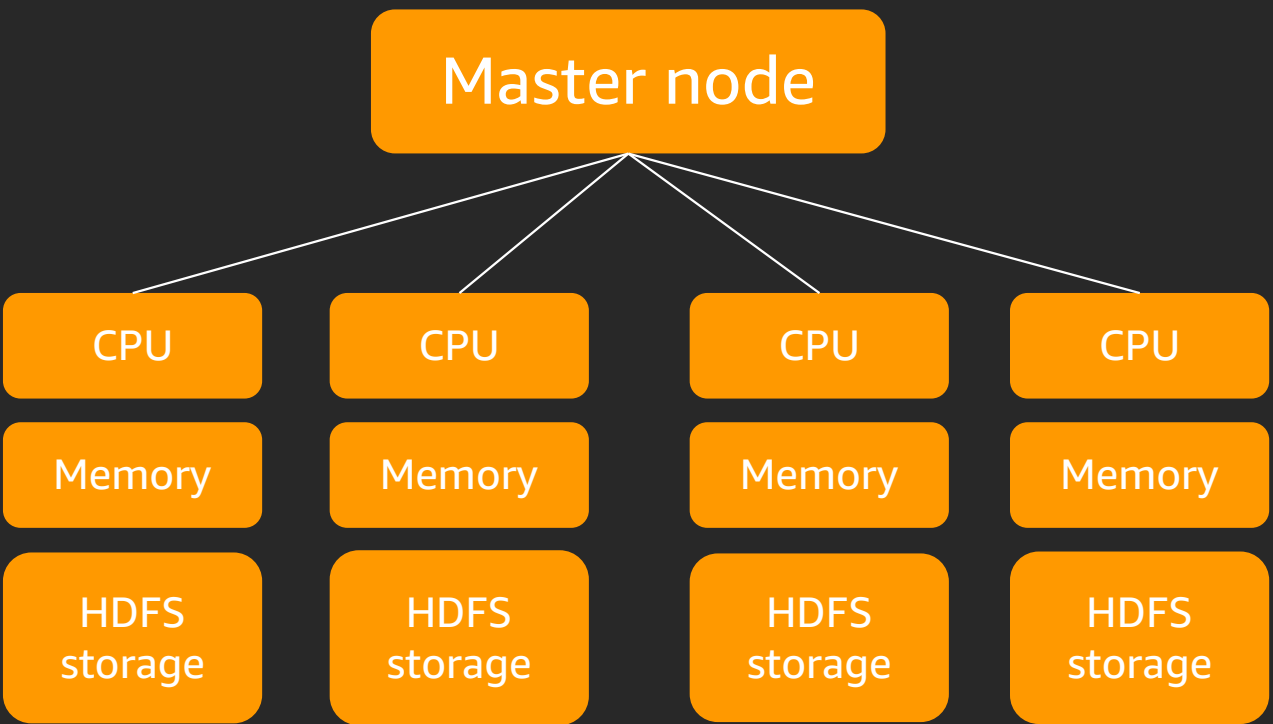
- 클러스터 외부에 메타스토어 유지(Glue Data Catalog 또는 RDS)
- 빠르게 시작하고 목적인 작업을 수행
- 동시에 여러 클러스터가 S3 데이터를 활용 가능

Remote metastore



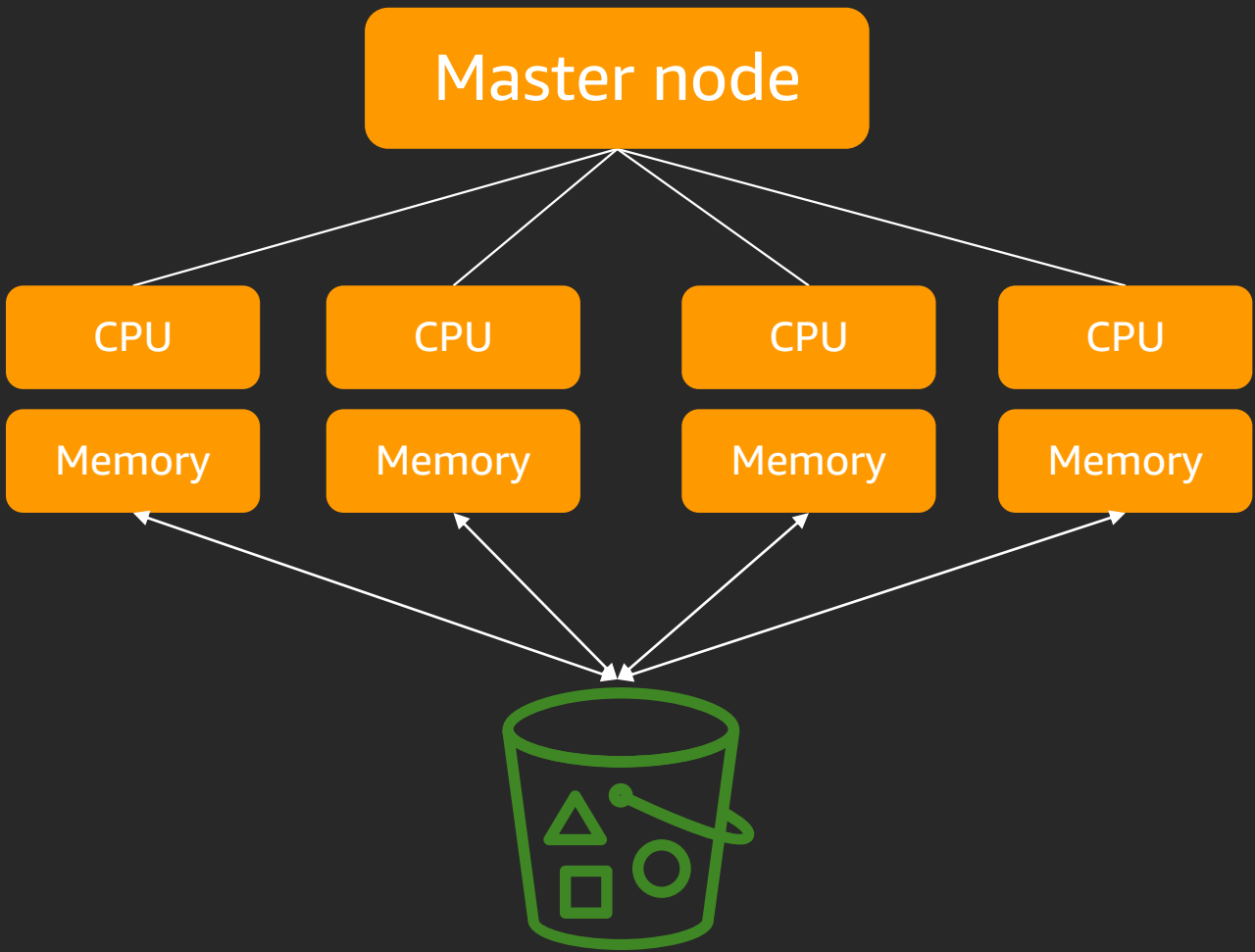
Stateless 클러스터 아키텍처 권장

Old clustering/localized model



HDFS가 3개의 복제본을 가져야 하므로
500-TB 데이터 저장을 위해 1.5-PB 규모의 클러스터 필요

Amazon EMR decoupled model



다수의 EMR 클러스터와 노드가 동시에 EMR file system을 통해 S3 데이터 사용

Apache Spark

Spark는 대량의 데이터 처리를 위한 병렬 처리 플랫폼으로 다양한 유형의 워크로드를 수행할 수 있는 컴포넌트로 구성되어 있음

**Spark
SQL**

**Spark/Struc
tured
Streaming**

**Spark
R**

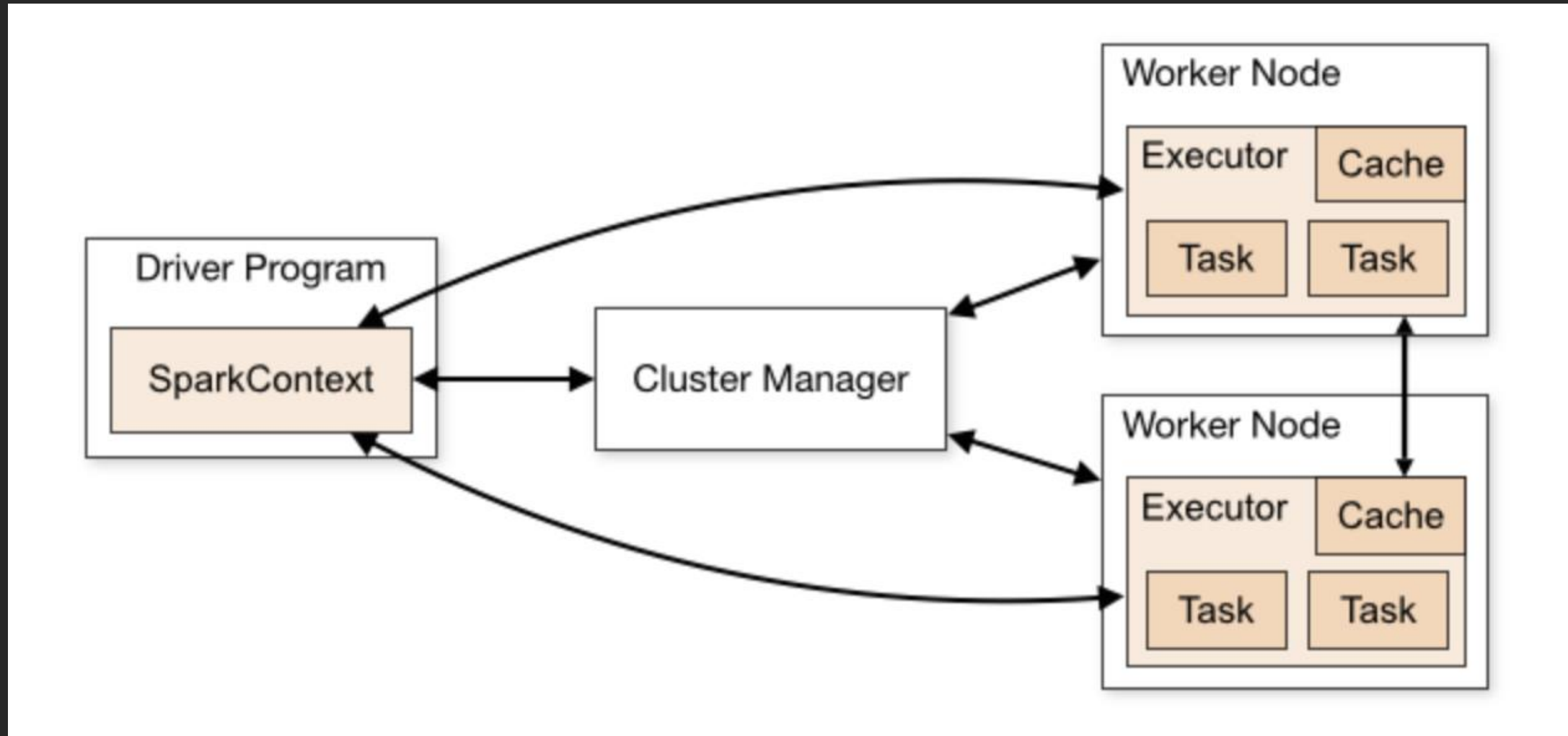
**Spark
ML**

Graph X

Spark Core

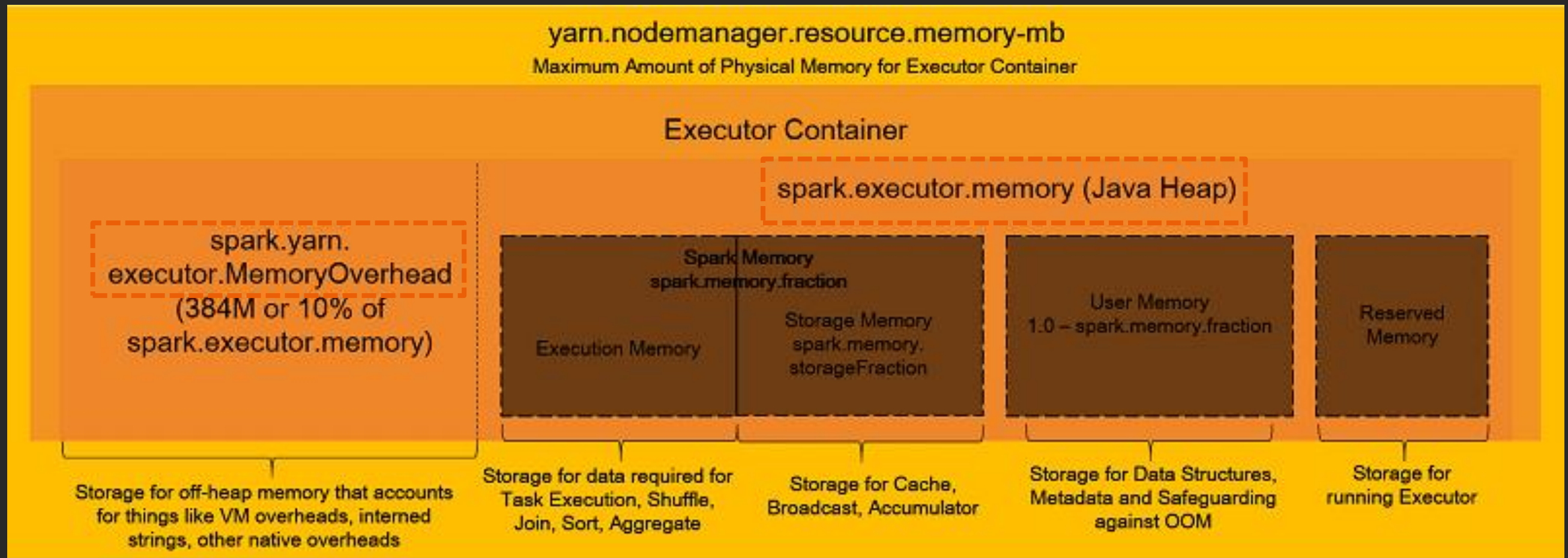
Spark의 실행 모델

Spark는 마스터/워커 아키텍처를 가지고 실질적인 작업을 수행하는 Executor와 해당 Worker를 관리하는 Driver가 있음



Spark 메모리 구성

- ✓ `yarn.nodemanager.resource.memory-mb` : 익스큐터 전체의 물리 메모리 영역
- ✓ `spark.executor.memory` : 익스큐터가 Job 실행에 사용할 수 있는 메모리 영역
- ✓ `spark.shuffle.memoryFraction` : 전체 힙 영역에서 익스큐터와 RDD 데이터 저장에 사용될 비율
- ✓ `spark.storage.memoryFraction` : 할당된 메모리에서 데이터 저장에 사용할 비율
- ✓ `spark.yarn.executor.MemoryOverHead` : VM 관련 오버헤드를 위해 할당, 필요시 조정 가능



Spark 워크로드 최적화 방안

Spark 워크로드 최적화 (1/2)

Spark 작업은 일반적으로 대량의 데이터를 Memory 기반의 리소스에서 빠르게 처리해야 하는 요구사항을 가진다.

- 대량의 데이터 작업에서 로딩하는 데이터를 줄여주는 접근이 가장 중요
 - ✓ 최적의 데이터 포맷 활용 - Apache Parquet compressed by Snappy(Spark 2.x의 기본값)
 - ✓ 데이터 스캔 범위를 최소화 한다. - Hive partitions, Bucketing, Push Down, Partitioning Pruning
 - ✓ 특정 파티션에 Data Skew 가 발생하면 파티션 키에 대한 고려가 필요
- 사용 가능한 클러스터의 메모리를 최대한 효율적으로 사용
 - ✓ 작업에서 빈번하게 사용되는 데이터를 메모리에 캐시한다. - `dataFrame.cache()`
 - ✓ 워크로드에 적절한 Spark 설정값을 지정해줘야 함.
 - ✓ 작업 유형에 적합한 파티션 사이즈로 변경한다. - `spark.sql.files.maxPartitionBytes`

Spark 워크로드 최적화 (2/2)

- 리소스를 가장 많이 사용하는 조인 및 셔플링 최적화
 - ✓ 셔플링과 repartitioning 은 가장 비용이 비싼 오퍼레이션이므로 최소화 되도록 Job 모니터링
 - ✓ Spark 2.3 이후 기본 조인은 SortMerge(조인전 각 데이터셋의 정렬이 필요), Broadcast Hash 조인은 대상 테이블의 사이즈가 크게 차이나는 경우 유용, 특히 10Mb 미만 테이블은 자동 Broadcast, 설정을 통해 변경 가능 `spark.sql.autoBroadcastJoinThreshold`
 - ✓ 테이블 사이즈에 따라 조인 순서를 변경하여 대량 데이터 셔플링 방지 - Join Reorder
- 클러스터 규모와 작업 유형에 따라 적절한 환경 변수 설정
 - ✓ 익스큐터 갯수 설정 `--num-executors`
 - ✓ 익스큐터의 코어수 설정 `--executor-cores`
 - ✓ 익스큐터의 메모리 크기를 변경 `--executor-memory`

```
%%configure
```

```
{"executorMemory": "3072M", "executorCores": 4, "numExecutors": 10}
```

- 스토리지 성능 최적화
 - ✓ 데이터 활용 빈도 및 속도 요구치에 맞춰 HDFS, EMRFS(S3) 활용 선택

Spark의 주요 메모리 이슈 유형 (1/2)

- 자바 힙메모리 부족 오류 - Spark instance 수, executor memory, core수 등이 많은 양의 데이터를 처리할 수 있도록 설정되지 않는 경우

```
WARN TaskSetManager: Loss was due to  
java.lang.OutOfMemoryError java.lang.OutOfMemoryError: Java heap space
```

- 물리 메모리 초과 - 가비지 컬렉션과 같은 시스템 작업을 수행하는 데 필요한 메모리를 Spark executor 인스턴스에서 사용할 수 없는 경우

```
Error: ExecutorLostFailure Reason: Container killed by YARN for exceeding limits. 12.4 GB of 12.3 GB  
physical memory used.  
Consider boosting spark.yarn.executor.memoryOverhead.  
Error: ExecutorLostFailure Reason: Container killed by YARN for exceeding limits. 4.5GB of 3GB  
physical memory used limits.  
Consider boosting spark.yarn.executor.memoryOverhead.
```

Spark의 주요 메모리 이슈 유형 (2/1)

- 가상 메모리 초과 - 가비지 컬렉션과 같은 시스템 작업을 수행하는 데 필요한 메모리를 Spark executor 인스턴스에서 사용할 수 없는 경우

Container killed by YARN for exceeding memory limits.
1.1gb of 1.0gb virtual memory used. Killing container.

- 익스큐터 메모리 초과 - Spark executor 물리적 메모리가 YARN에서 할당된 메모리를 초과하는 경우

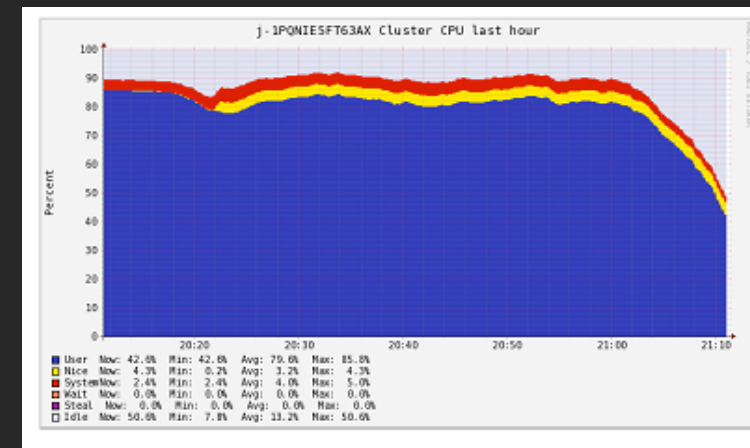
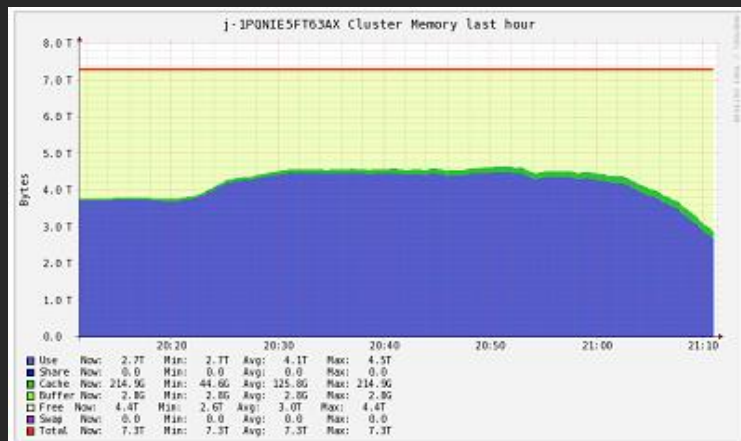
Required executor memory (1024+384 MB) is above
the max threshold (896 MB) of this cluster!
Please check the values of 'yarn.scheduler.maximum-allocation-mb' and/or
'yarn.nodemanager.resource.memory-mb'

Spark 메모리 관련 주요 파라미터 설정 사례

- EMR에서는 spark-defaults 값을 통해 기본값 설정이 되어 있으나 전반적으로 낮게 설정된 값으로 인해 애플리케이션이 클러스터 전체 성능을 사용하지 못하므로 추가 설정이 필요함.
- Spark 구성 파라미터
 - ✓ spark.executor.memory – 작업을 실행하는 각 익스큐터에 사용할 메모리의 크기입니다.
 - ✓ spark.executor.cores – 익스큐터에 할당되는 가상 코어의 수입니다.
 - ✓ spark.driver.memory – 드라이버에 사용할 메모리의 크기입니다.
 - ✓ spark.driver.cores – 드라이버에 사용할 가상 코어의 수입니다.
 - ✓ spark.executor.instances – 익스큐터의 수입니다. spark.dynamicAllocation.enabled가 true로 설정된 경우 외에는 이 파라미터를 설정합니다.
 - ✓ spark.default.parallelism – 사용자가 파티션 수를 설정하지 않았을 때 join, reduceByKey 및 parallelize와 같은 변환에 의해 반환된 RDD의 파티션 수 기본값입니다.

Spark 메모리 관련 주요 파라미터 설정 사례

- r5.12xlarge(48 vCPU, 384 Gb 메모리) 마스터 1대, r5.12xlarge 코어노드 19대의 EMR 클러스터로 S3에 저장된 10TB 데이터 처리 환경
 - ✓ 익스큐터당 5개의 vCPU 할당 `spark.executor.cores = 5` (vCPU)
 - ✓ 인스턴스당 익스큐터수 계산 $(48-1) / 5 = 9$
 - ✓ 인스턴스 메모리 384Gb 중 90%는 각 익스큐터에 할당 `spark.executor.memory = 42 * 0.9 = 37`
 - ✓ 약 10%는 각 익스큐터의 Overhead에 할당 `spark.yarn.executor.memoryOverhead = 42 * 0.1 = 5`
 - ✓ 드라이버 메모리는 익스큐터와 동일하게 `spark.driver.memory = spark.executor.memory`
 - ✓ 전체 익스큐터 수는 `spark.executor.instances = (9 * 19) - 1`(드라이버수) = 170
 - ✓ 병렬처리 값은 `spark.default.parallelism = 170`(익스큐터수) * 5(코어수) * 2 = 1,700



Spark configuration 변경 방법

Spark configuration을 변경하기 위한 방법 여러가지가 있으나 각각은 config 값을 적용하는 시점이 달라 1번부터 우선순위가 가장 높게 반영된다.

1. SparkConf의 Set 함수를 이용하여 Runtime에서 설정 값을 변경한다.

```
conf = spark.sparkContext._conf
      .setAll([('spark.executor.memory', '4g'), ('spark.executor.cores','4')],('spark.driver.memory','4g'))
spark.sparkContext.stop()
spark = SparkSession.builder.config(conf=conf).getOrCreate()
```

2. spark-submit을 통해 설정값을 전달한다.

```
./bin/spark-submit --class org.apache.spark.examples.SparkPi --master yarn --deploy-mode cluster
--executor-memory 20G \ --num-executors 50
```

3. conf/spark-defaults.conf 파일을 직접 수정한다.

```
spark.executor.memory      18971M
spark.executor.cores       4
spark.yarn.executor.memoryOverheadFactor 0.1875
```

EMR 인스턴스 템플릿 및 구성 방법

- Spark 및 Yarn의 구성 파라미터 설정을 EMR 콘솔의 **Edit software settings** 항목을 통해 직접 입력하거나 **Load JSON from S3** 기능을 통해 미리 구성된 json 파일을 로딩할 수 있음

Software Configuration

Release emr-5.29.0

| | | |
|--|---|--|
| <input checked="" type="checkbox"/> Hadoop 2.8.5 | <input type="checkbox"/> Zeppelin 0.8.2 | <input type="checkbox"/> Livy 0.6.0 |
| <input type="checkbox"/> JupyterHub 1.0.0 | <input type="checkbox"/> Tez 0.9.2 | <input type="checkbox"/> Flink 1.9.1 |
| <input type="checkbox"/> Ganglia 3.7.2 | <input type="checkbox"/> HBase 1.4.10 | <input checked="" type="checkbox"/> Pig 0.17.0 |
| <input checked="" type="checkbox"/> Hive 2.3.6 | <input type="checkbox"/> Presto 0.227 | <input type="checkbox"/> ZooKeeper 3.4.14 |
| <input type="checkbox"/> MXNet 1.5.1 | <input type="checkbox"/> Sqoop 1.4.7 | <input type="checkbox"/> Mahout 0.13.0 |
| <input checked="" type="checkbox"/> Hue 4.4.0 | <input type="checkbox"/> Phoenix 4.14.3 | <input type="checkbox"/> Oozie 5.1.0 |
| <input checked="" type="checkbox"/> Spark 2.4.4 | <input type="checkbox"/> HCatalog 2.3.6 | <input type="checkbox"/> TensorFlow 1.14.0 |

Multiple master nodes (optional)
☐ Use multiple master nodes to improve cluster availability. [Learn more](#)

AWS Glue Data Catalog settings (optional)
☐ Use for Hive table metadata
☐ Use for Spark table metadata

Edit software settings
☒ Enter configuration ☐ Load JSON from S3

`classification=config-file-name,properties=[myKey1=myValue1,myKey2=myValue2]`

EMR 클러스터 구성 예시

```
{ "InstanceGroups":[
  { "Name":"AmazonEMRMaster",
    "Market":"ON_DEMAND",
    "InstanceRole":"MASTER",
    "InstanceType":"r5.12xlarge",
    "InstanceCount":1,
    "Configurations":[
      { "Classification": "yarn-site",
        "Properties":
          { "yarn.nodemanager.vmem-check-enabled": "false",
            "yarn.nodemanager.pmem-check-enabled": "false" } },
      { "Classification": "spark",
        "Properties":
          { "maximizeResourceAllocation": "false" } },
      { "Classification": "spark-defaults",
        "Properties":
          { "spark.network.timeout": "800s",
            "spark.executor.heartbeatInterval": "60s",
            "spark.dynamicAllocation.enabled": "false",
            "spark.driver.memory": "21000M",
            "spark.executor.memory": "21000M",
            "spark.executor.cores": "5",
            "spark.executor.instances": "171",
```


EMR 클러스터 구성 예시

```
"spark.memory.fraction": "0.80",  
"spark.memory.storageFraction": "0.30",  
"spark.executor.extraJavaOptions": "-XX:+UseG1GC -XX:+UnlockDiagnosticVMOptions  
-XX:+G1SummarizeConcMark -XX:InitiatingHeapOccupancyPercent=35  
-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps  
-XX:OnOutOfMemoryError='kill -9 %p'",  
"spark.driver.extraJavaOptions": "-XX:+UseG1GC -XX:+UnlockDiagnosticVMOptions  
-XX:+G1SummarizeConcMark -XX:InitiatingHeapOccupancyPercent=35  
-verbose:gc -XX:+PrintGCDetails -XX:+PrintGCDateStamps  
-XX:OnOutOfMemoryError='kill -9 %p'",  
"spark.yarn.scheduler.reporterThread.maxFailures": "5",  
"spark.storage.level": "MEMORY_AND_DISK_SER",  
"spark.rdd.compress": "true",  
"spark.shuffle.compress": "true",  
"spark.shuffle.spill.compress": "true",  
"spark.default.parallelism": "3400" }
```

Spark on EMR 성능 최적화

Spark on EMR 성능 최적화 기본 팁

- 데이터의 크기와 작업의 사이즈에 적절한 메모리 할당이 가능한 instance type을 선택(같은 vCPU를 가진 c 타입 인스턴스 대비 m / r 타입 인스턴스가 두배의 메모리를 가짐)
- 최신 버전의 EMR 버전을 사용한다. (최소 5.24.0 이상) 버전 5.28 이후에는 EMR Runtime for Apache Spark이 포함되어 있어 성능 향상
- 추가적인 성능 최적화를 위해서 워크로드 특성에 맞는 메모리 설정 변경이 필요

Spark on EMR 성능 최적화 (1/5)

- Dynamic Partition Pruning - 쿼리 대상 테이블을 보다 정확하게 선택하여 스토리지에서 읽고 처리하는 데이터량을 줄여 주어 시간과 리소스 절약
 - ✓ Spark Properties : ***spark.sql.dynamicPartitionPruning.enabled*** (EMR 5.24 이후 사용 가능, 5.26 이후 기본적으로 활성화)
 - ✓ 아래의 쿼리에서 동적으로 파티셔닝 된 데이터의 범위를 줄여주어, Where 절의 조건에 맞는 데이터만 필터링 하여 'North America' 영역에 해당하는 파티션 데이터만 읽어서 처리

```
select ss.quarter, ss.region, ss.store, ss.total_sales
from store_sales ss, store_regions sr
where ss.region = sr.region and sr.country = 'North America'
```

Spark on EMR 성능 최적화 (2/5)

- Flattening Scalar Subqueries - 다수개의 서브쿼리를 하나로 통합하여 재작성, 수행 성능을 향상
 - ✓ Spark Properties : *spark.sql.optimizer.flattenScalarSubqueriesWithAggregates.enabled* (EMR 5.24 이후 사용 가능, 5.26 이후 기본적으로 활성화)
 - ✓ 동일한 관계를 사용하는 다수의 스칼라 쿼리를 하나의 쿼리로 통합하여 실행하여 성능을 향상

```
/* 샘플 쿼리 */  
select (select avg(age) from students /* Subquery 1 */  
       where age between 5 and 10) as group1,  
       (select avg(age) from students /* Subquery 2 */  
       where age between 10 and 15) as group2,  
       (select avg(age) from students /* Subquery 3 */  
       where age between 15 and 20) as group3
```

```
/* 최적화 된 쿼리 */  
select c1 as group1, c2 as group2, c3 as group3  
from (select avg (if(age between 5 and 10, age, null)) as c1,  
           avg (if(age between 10 and 15, age, null)) as c2,  
           avg (if(age between 15 and 20, age, null)) as c3 from students);
```

Spark on EMR 성능 최적화 (3/5)

- DISTINCT Before INTERSECT - Intersect 사용시 자동적으로 Left semi join으로 변환, Distinct 연산을 Intersect 하위 항목을 푸시하여 성능 향상
 - ✓ Spark Properties : *spark.sql.optimizer.distinctBeforeIntersect.enabled* (EMR 5.24 이후 사용 가능, 5.26 이후 기본적으로 활성화)

```
/* 샘플 쿼리 */  
(select item.brand brand from store_sales,item  
  where store_sales.item_id = item.item_id)  
intersect  
(select item.brand cs_brand from catalog_sales, item  
  where catalog_sales.item_id = item.item_id)
```

```
/* 최적화 쿼리 */  
select brand from  
  (select distinct item.brand brand from store_sales, item  
    where store_sales.item_id = item.item_id)  
left semi join  
  (select distinct item.brand cs_brand from catalog_sales, item  
    where catalog_sales.item_id = item.item_id) on brand <=> cs_brand
```

Spark on EMR 성능 최적화 (4/5)

- Bloom Filter Join - 사전에 작성된 Bloom Filter를 통해 쿼리 대상 데이터의 범위를 줄여줌으로써 성능을 향상
 - ✓ Spark Properties : ***spark.sql.bloomFilterJoin.enabled*** (EMR 5.24 이후 사용 가능, 5.26 이후 기본적으로 활성화)
 - ✓ 아래의 쿼리에서 조인 전에 sales 테이블에서 item.category가 1, 10, 16에 해당하는 데이터를 먼저 필터링 하므로 조인 성능을 매우 향상시킬 수 있음

```
select count(*) from  
sales, item  
where sales.item_id = item.id and item.category in (1, 10, 16)
```

Spark on EMR 성능 최적화 (5/5)

- Optimized Join Reorder - 쿼리에 적혀있는 테이블의 순서를 필터와 데이터 규모에 따라 재정렬하여 소규모 쿼리를 먼저 수행
 - ✓ Spark Properties : ***spark.sql.optimizer.sizeBasedJoinReorder.enabled*** (EMR 5.24 이후 사용 가능, 5.26 이후 기본적으로 활성화)
 - ✓ Spark 의 기본 동작은 쿼리에 있는 테이블들의 왼쪽에서 오른쪽으로 차례대로 조인하는 것임.
 - ✓ 아래의 쿼리에서 원래 조인 순서는 store_sales, store_returns, store, item 순서이지만

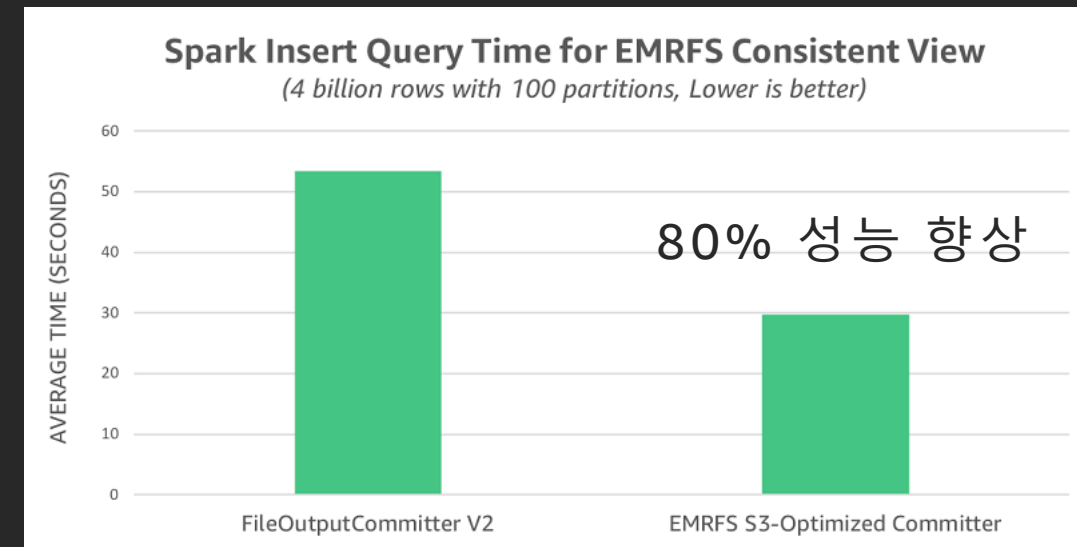
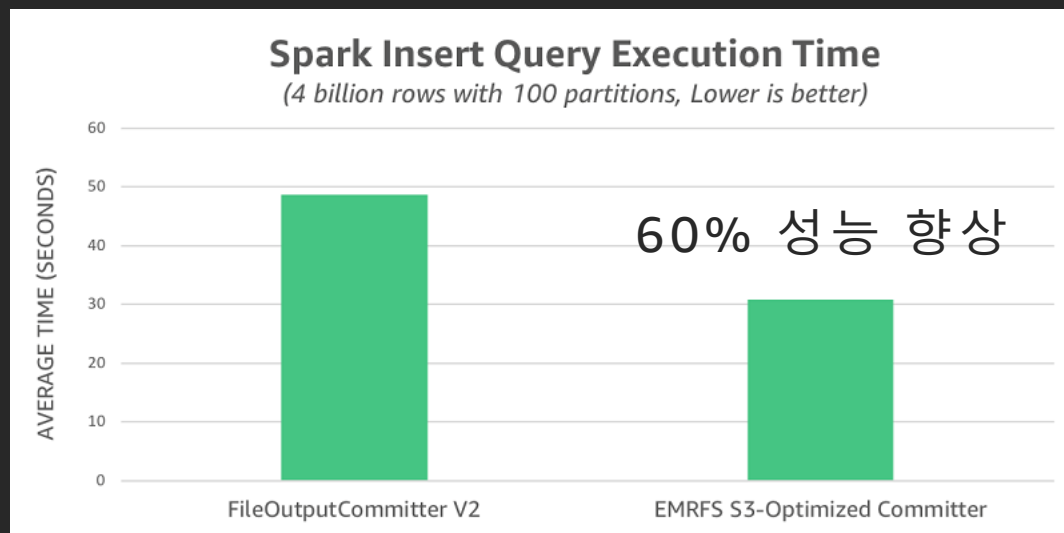
```
select ss.item_value, sr.return_date, s.name, i.desc,  
from store_sales ss, store_returns sr, store s, item i  
where ss.id = sr.id and ss.store_id = s.id and ss.item_id = i.id and s.country = 'USA'
```

- ✓ 실제 조인 실행 순서는 1. **store_sales** 와 **store** (store에 country 필터가 있으므로) 2. **store_returns** 3. **item** 순서이며, **Item**에 필터가 추가되면 **item**이 **store_returns** 보다 먼저 조인 되도록 재정렬 될 수 있음

S3를 통한 Spark 성능 향상 (1/2)

- EMRFS S3 최적화된 커미터 사용
 - ✓ Spark Properties : ***spark.sql.parquet.fs.optimized.committer.optimization-enabled*** (EMR 5.19 이후 사용 가능, 5.20 이후 기본적으로 활성화)
 - ✓ 기본적으로 S3 multipart upload 옵션이 활성화 된 상태에서 Spark SQL / DataFrames / Datasets 에서 Parquet 형식으로 저장할 때 사용 가능
 - ✓ 테스트 환경 - EMR 5.19 (Master m5d.2xlarge / Core Node m5d.2xlarge * 8)
 - ✓ Input Data : 15Gb (100개의 parquet 파일)

```
INSERT OVERWRITE DIRECTORY 's3://${bucket}/perf-test/${trial_id}'  
USING PARQUET SELECT * FROM range(0, ${rows}, 1, ${partitions});
```



S3를 통한 Spark 성능 향상 (2/2)

- S3 Select를 통해 데이터 필터링을 S3로 푸시다운
 - ✓ Spark, Presto, Hive에서 S3 Select 를 통해 대용량 데이터 필터링을 S3 레벨에서 사전 처리 가능(EMR 5.17 이후 사용 가능)
 - ✓ CSV, JSON, Parquet 파일 형식 지원 / Bzip2, Gzip, Snappy 압축 파일 지원
 - ✓ 기본적인 Where 조건에서의 특정 컬럼 기반의 필터링에 유용, 단 쿼리에 명시되어야 함
 - ✓ 집계함수, 형변환이 포함된 필터링 등은 S3로 푸시다운 되지 않음
 - ✓ 테이블 및 파일 형식은 다음과 같이 선언하며, 쿼리는 일반적인 Where 조건과 동일하게 사용

```
CREATE TEMPORARY VIEW MyView
(number INT, name STRING)
USING s3selectCSV
OPTIONS (path "s3://path/to/my/datafiles", header "true", delimiter "\t")
```

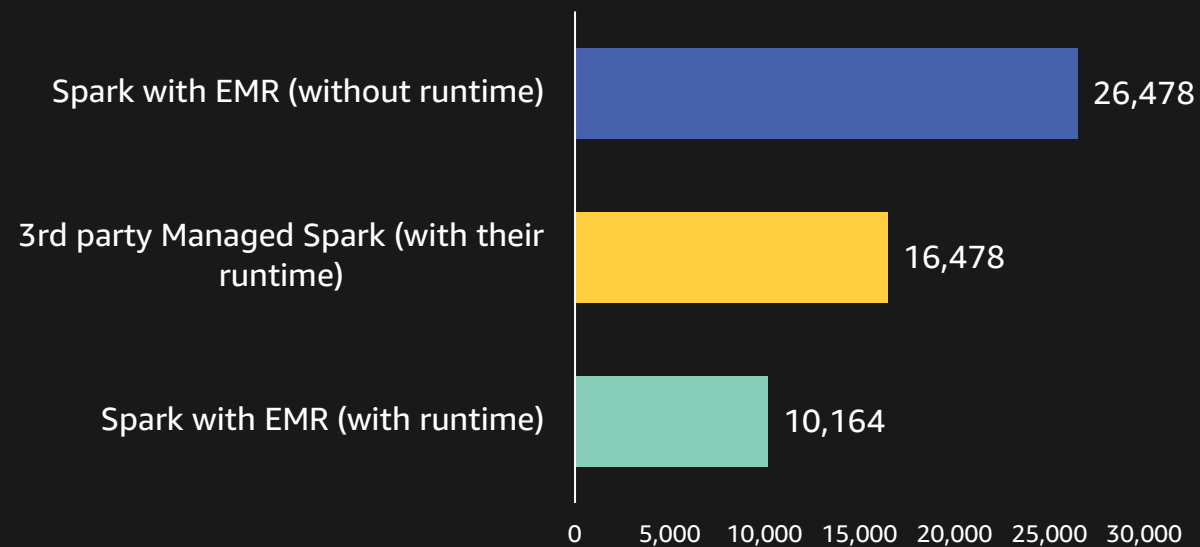
```
SELECT *
FROM MyView
WHERE number > 10;
```

EMR Runtime for Apache Spark

Spark 성능 향상을 위한 성능 최적화 EMR 제공

2.6배 성능과 1/10 가격으로 Spark 성능 최적화 Runtime 을 EMR에 포함

Runtime total on 104 queries (seconds - lower is better)



**Based on TPC-DS 3TB Benchmarking running 6 node C4x8 extra large clusters and EMR 5.28, Spark 2.4*

Spark 워크로드 실행 성능 향상을 위한 Runtime 내장

최상의 성능 제공

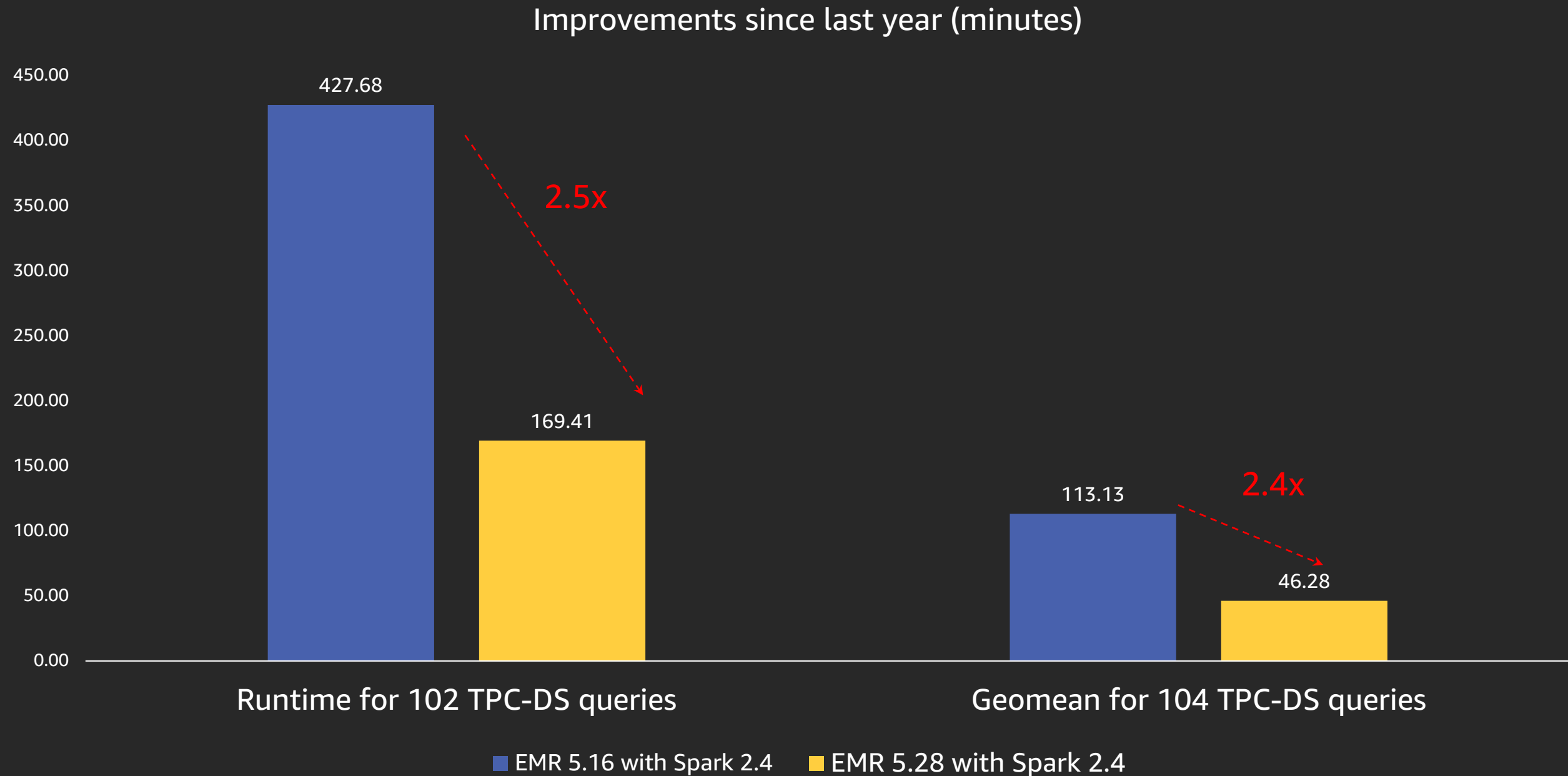
- 기존 Runtime 미포함 버전과 비교하여 2.6배 성능 향상
- 3rd party 에서 제공하는 Spark 패키지 대비 1.6배 성능 우위

비용 효율성

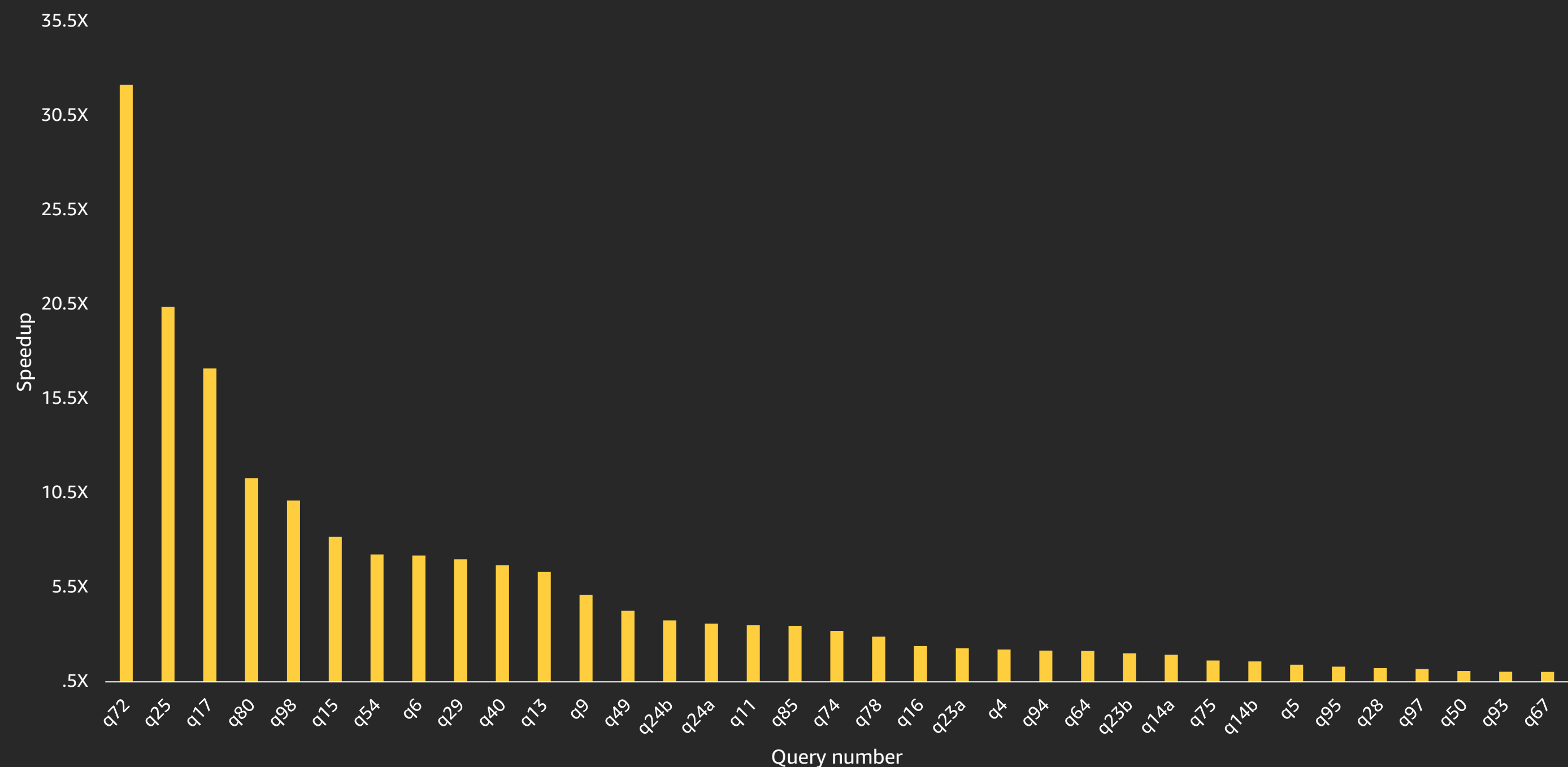
- 3rd party 에서 제공하는 Spark 패키지 대비 1/10 가격

오픈소스 Apache Spark API와 100% 호환

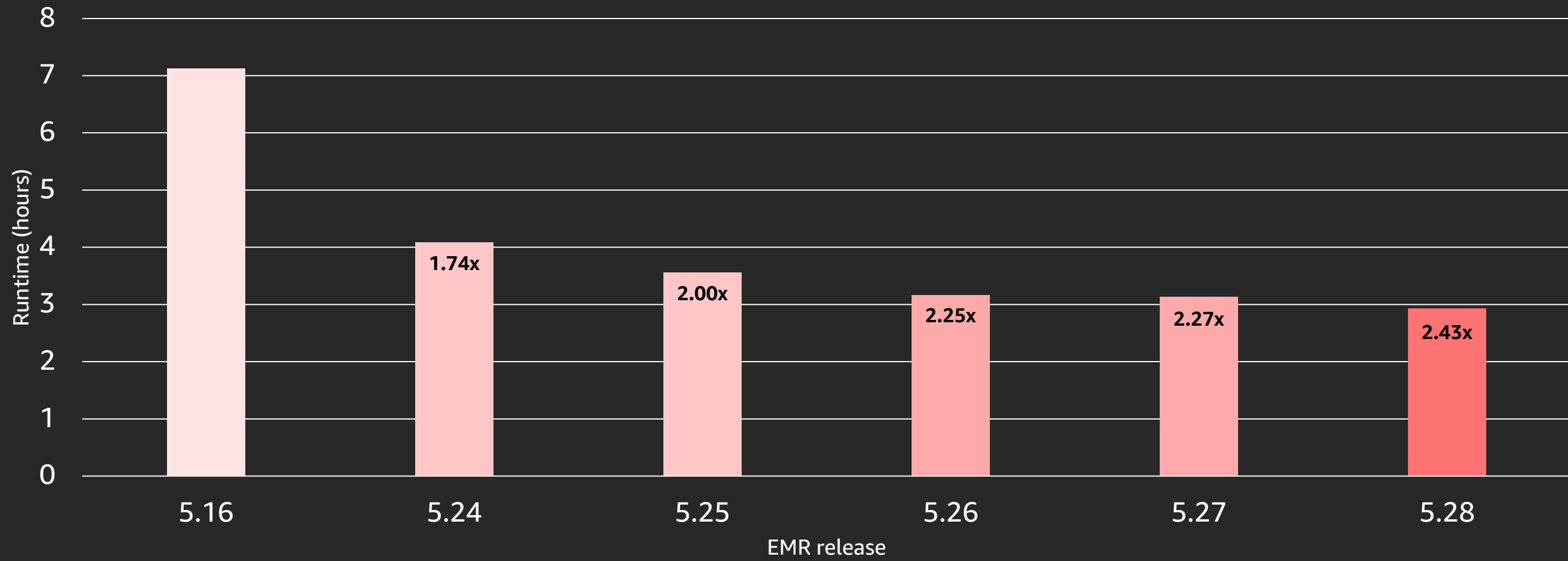
최신 버전의 EMR을 통한 성능 향상과 비용 절감



Long-running 쿼리에 대해서 평균 5배의 성능 향상



EMR 버전에 따른 비용 절감 / 성능 향상



Spark를 위한 Runtime의 최적화 방법

- Spark 작업 실행을 위한 최적의 configuration 값 셋팅
 - CPU/disk ratios, driver/executor conf, heap/GC, native overheads, instance defaults
- 데이터 작업 플랜 생성의 최적화
 - [Dynamic partition pruning](#), join reordering
- Query execution 최적화
 - [Data pre-fetch](#) and more
- Job startup 설정 셋팅
 - [Eager executor allocation](#), and more

Apache Hudi - 레코드 레벨의 데이터 Update, delete, insert

Data Lake 운영 환경에서의 Data Update 이슈

- 1. Latest snapshot of data
- 2. Historical view

MySQL database



Data lake

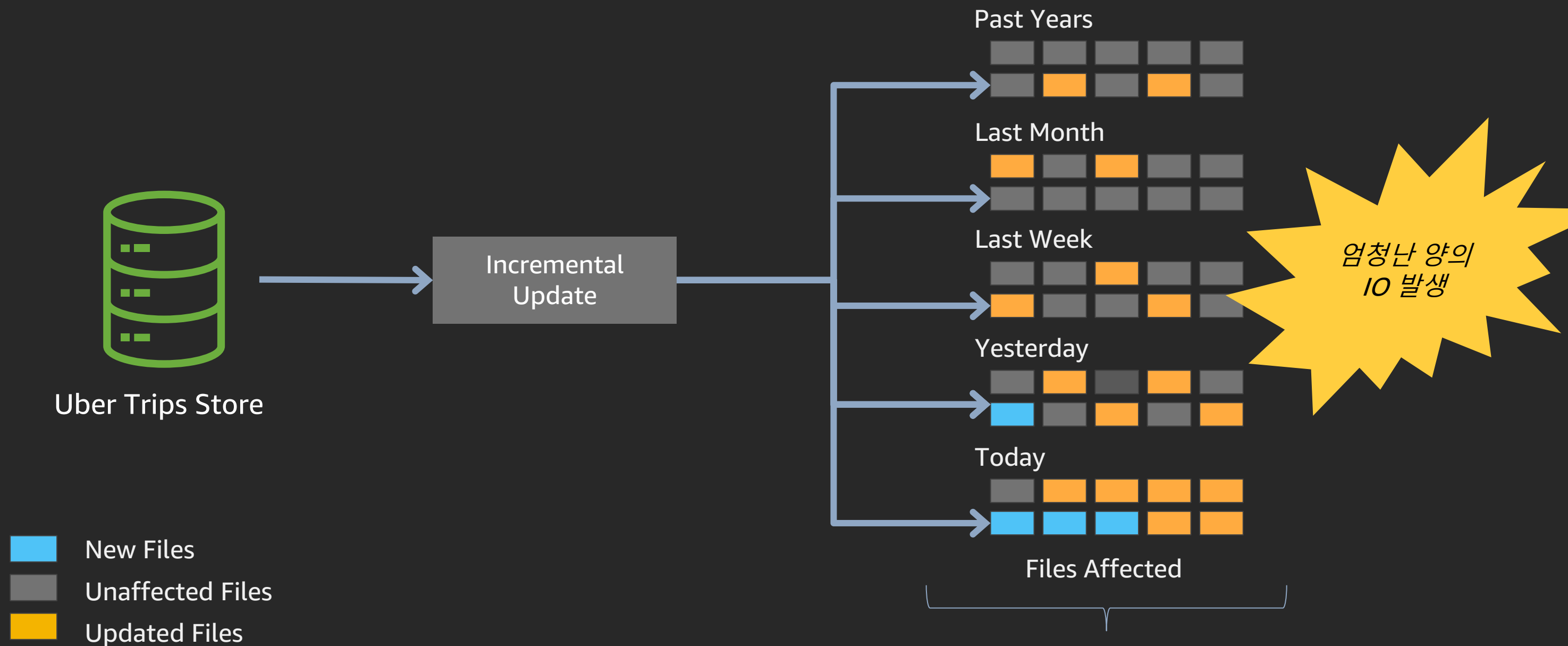


| Order ID | Quantity | Date |
|----------|----------|------------|
| 001 | 10 | 01/01/2019 |
| 001 | 15 | 01/02/2019 |
| 002 | 20 | 01/01/2019 |
| 002 | 20 | 01/02/2019 |

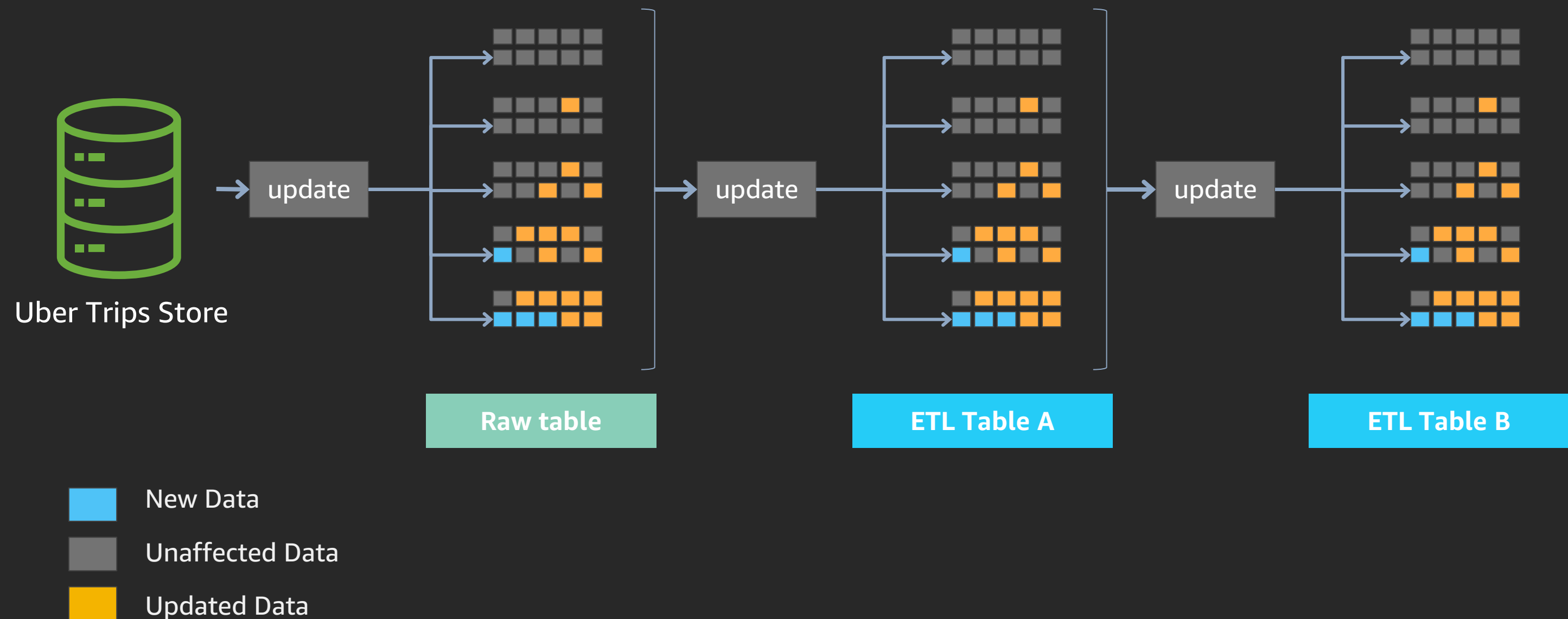
| Action | Order ID | Quantity | Date |
|--------|----------|----------|------------|
| I | 001 | 10 | 01/01/2019 |
| U | 001 | 15 | 01/02/2019 |
| I | 002 | 20 | 01/01/2019 |
| D | 002 | 20 | 01/02/2019 |

- 3. Incremental processing
- 4. Low end-to-end SLA

Uber의 실제 유스케이스 - Incremental Update

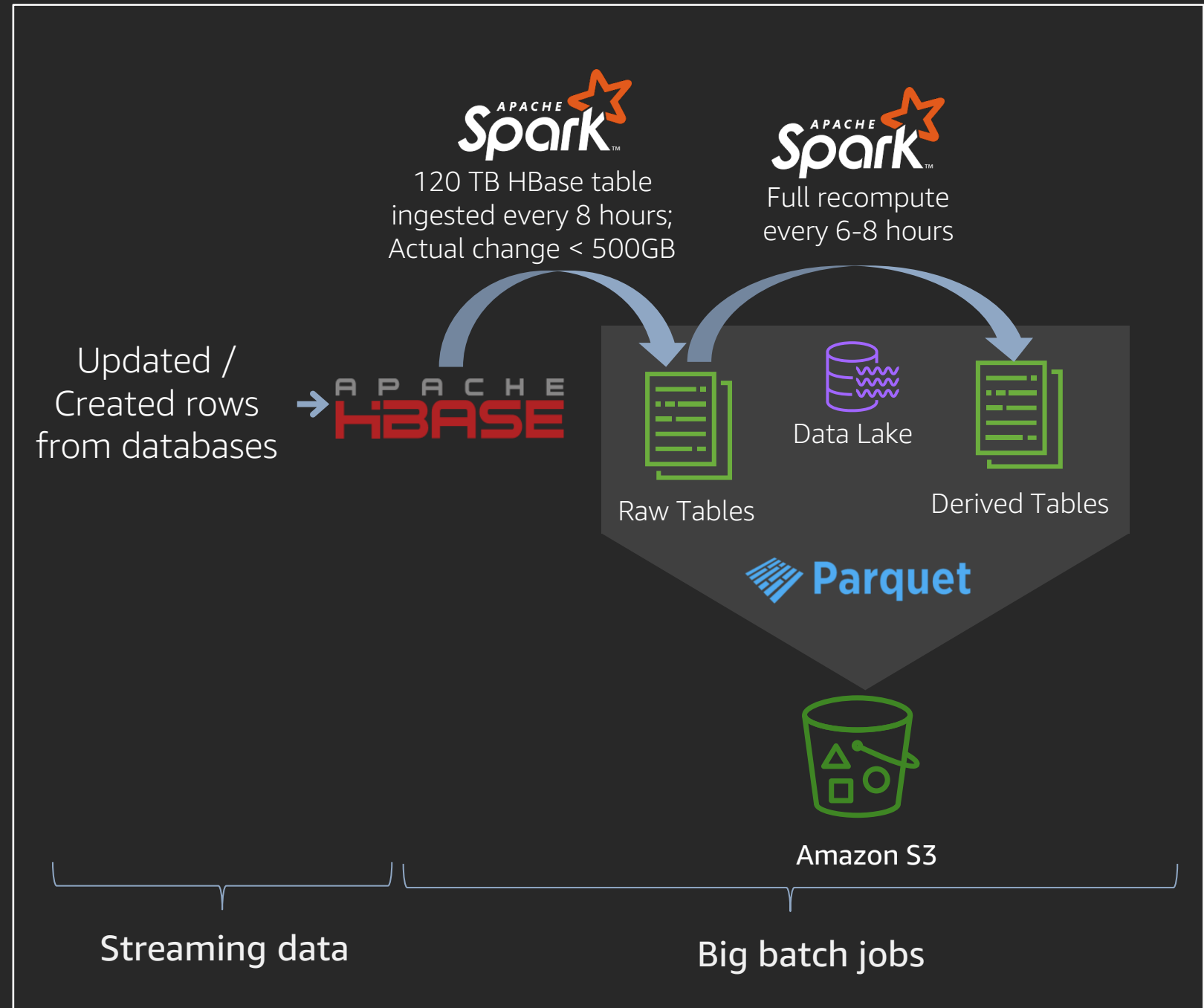


Uber의 실제 유스케이스 - Cascading Effects



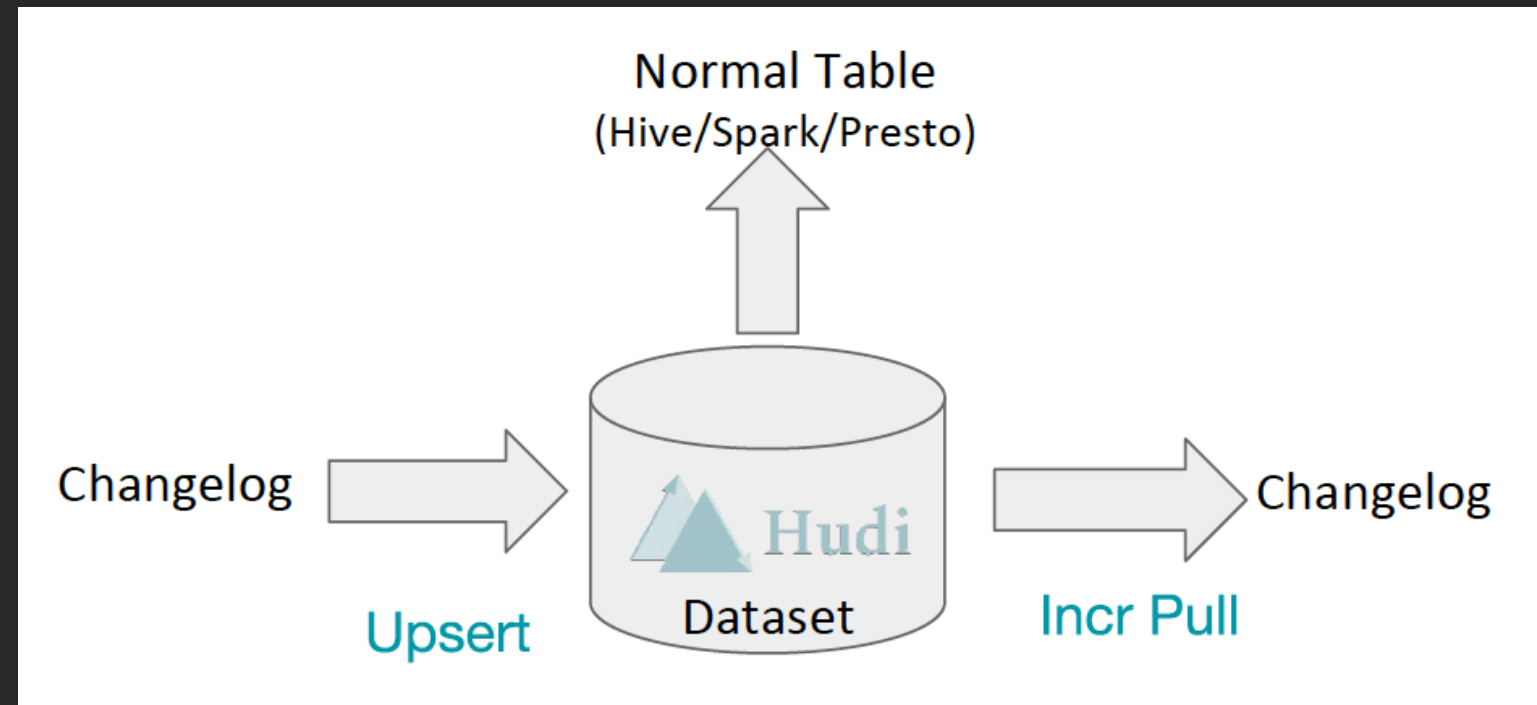
Slow Data Lake 이슈

매일 Hbase에 업데이트된 500GB 데이터를
이후 데이터 레이크에 반영하기 위해 실제
120TB 데이터를 처리, 8시간이 소요

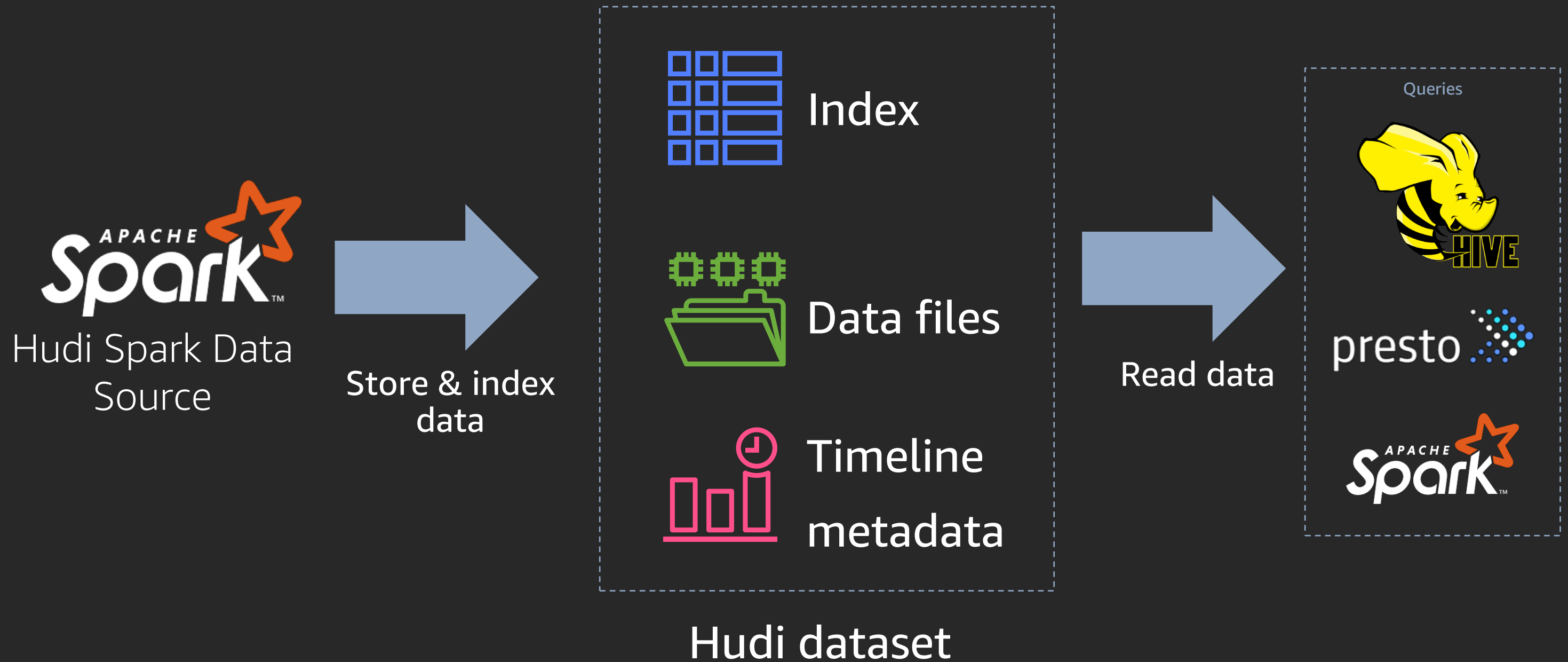


Apache Hudi (Hadoop upserts and incrementals)

- 스파크 기반의 데이터 관리 레이어
 - S3, Hive metastore와 호환 가능
 - Spark-SQL, Hive, Presto를 통해 데이터 쿼리 가능
 - Hudi CLI, DeltaStreamer, Hive Catalog Sync와 같은 다양한 인터페이스 제공



Apache Hudi (incubating)는 데이터 추상화 레이어

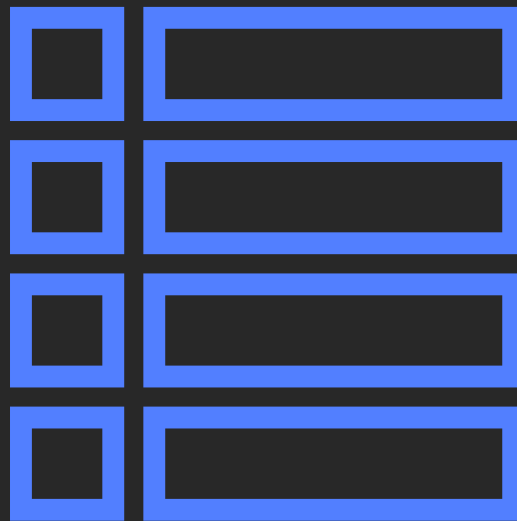


워크로드 유형에 따라 두가지 Storage type 지원

Copy On Write

Read heavy

읽기 성능 최적화
비교적 예측 가능한
작은 규모의 워크로드



Hudi
Dataset

Merge On Read

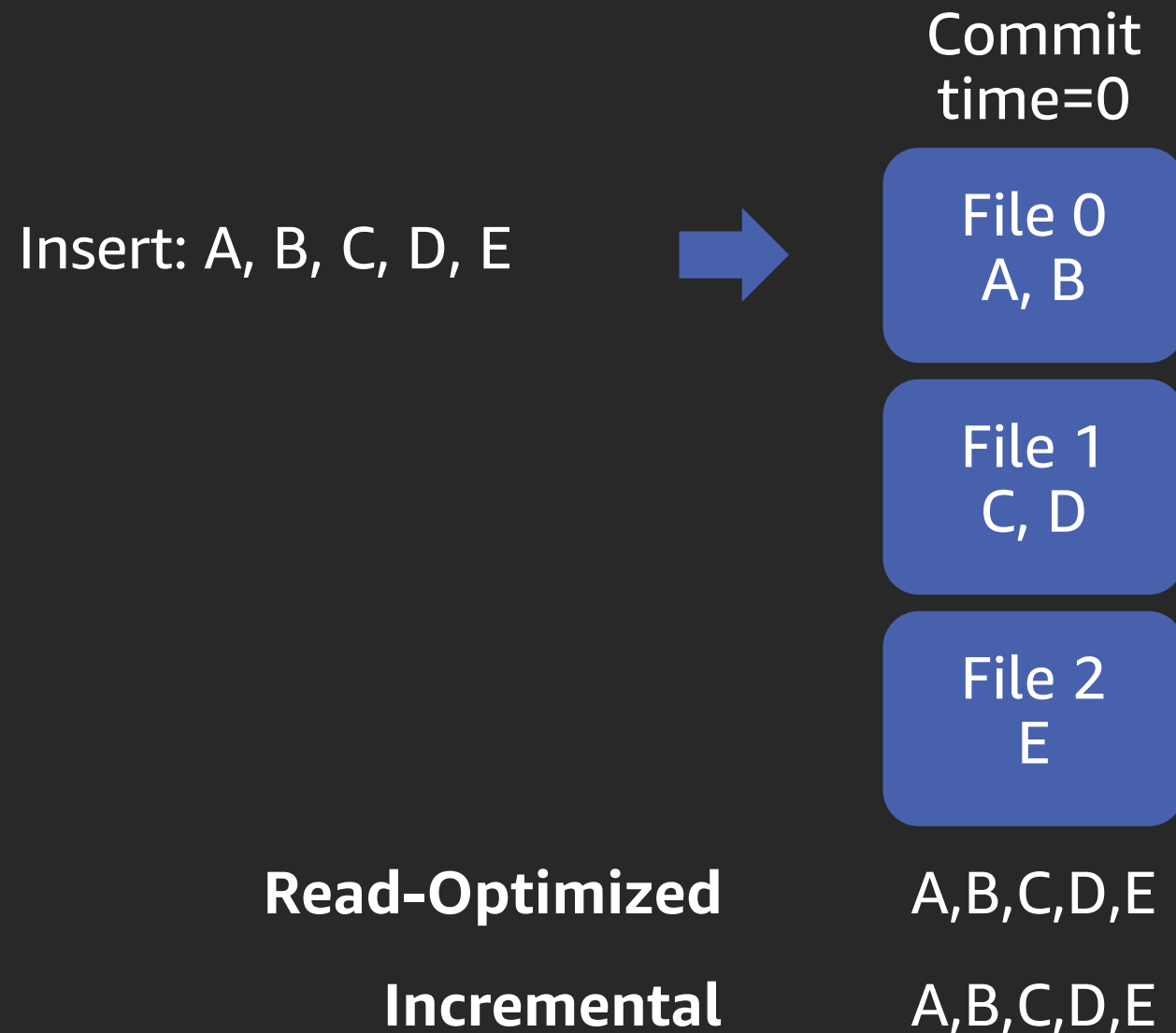
Write heavy

데이터 변경을 즉시 활용
어드밴스드 모드
워크로드 변화 대응 가능

Hudi Storage types & Views - Copy on Write

Storage Type: Copy On Write

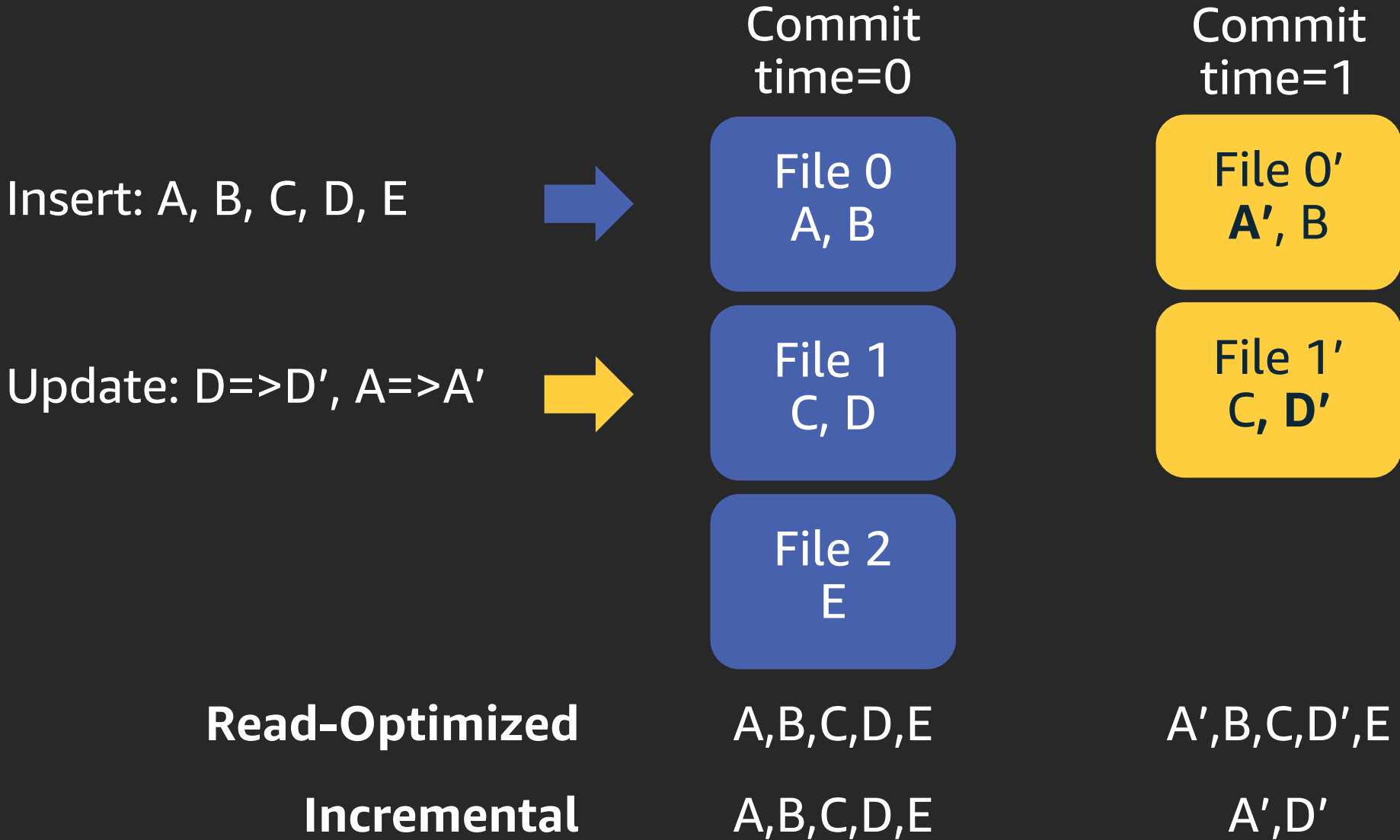
Views/Queries: Read-Optimized, Incremental



Hudi Storage types & Views - Copy on Write

Storage Type: Copy On Write

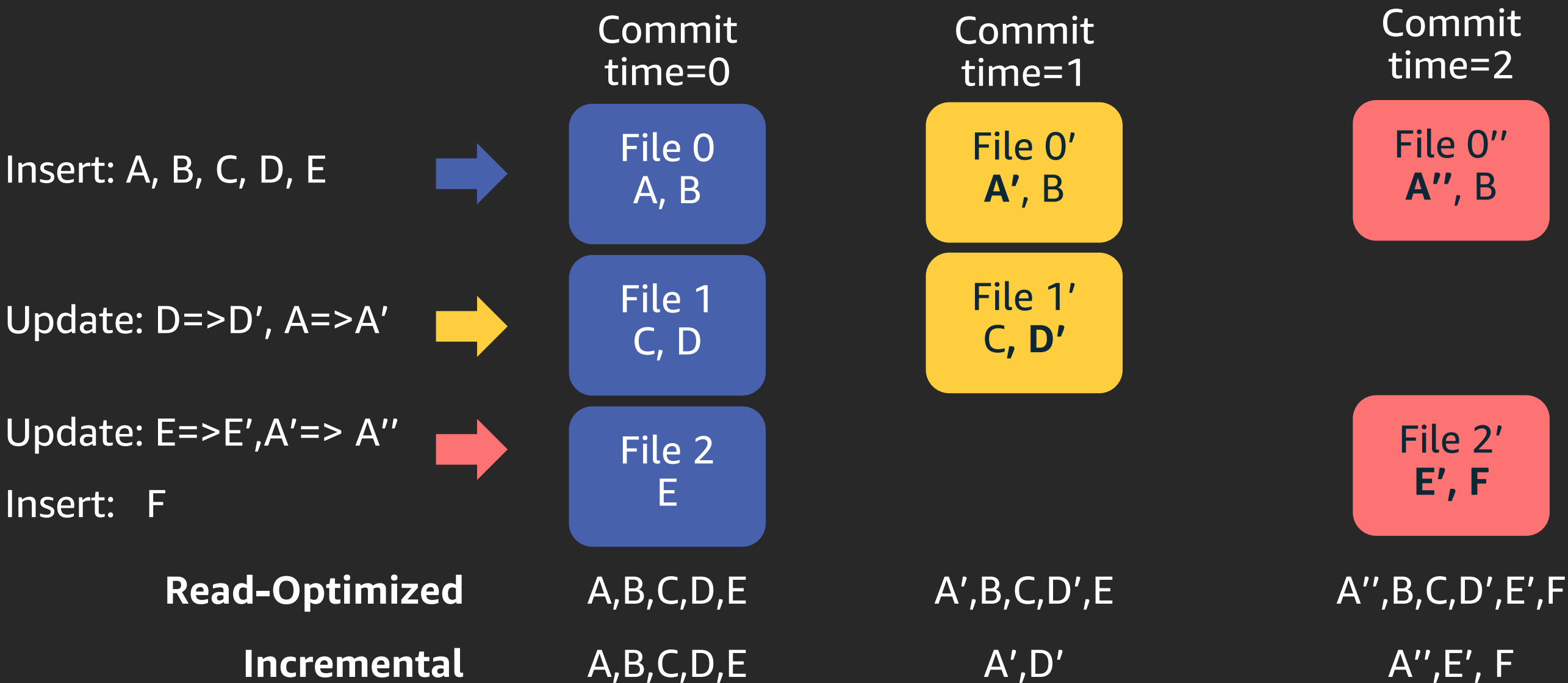
Views/Queries: Read-Optimized, Incremental



Hudi Storage types & Views - Copy on Write

Storage Type: Copy On Write

Views/Queries: Read-Optimized, Incremental



Hudi Storage types & Views - Copy on Write

Storage Type: Copy On Write

Views/Queries: Read-Optimized, Incremental

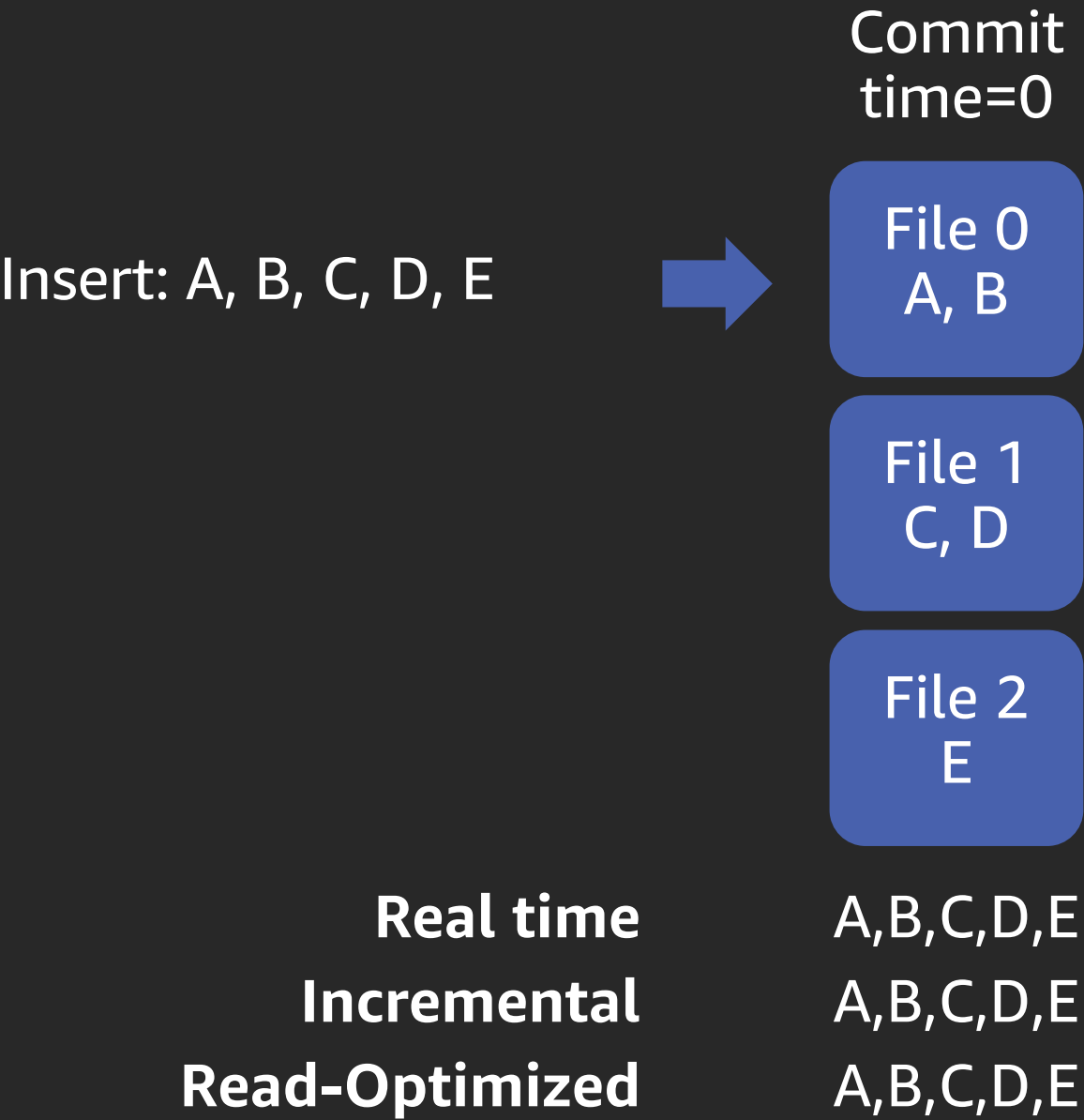
언제 사용할 것인가?

- 현재 작업이 데이터 업데이트를 위해서 전체 테이블/파티션을 다시 쓰기 할때
- 현재 워크로드가 비교적 증감이 일정하고, 갑작스럽게 피크를 치지 않을 때
- 데이터가 이미 Parquet 파일 형태로 저장되어 있을 때
- 오퍼레이션 관련 가장 간단한 요구사항을 가지고 있을 때

Hudi Storage types & Views - Merge On Read

Storage type: Merge On Read

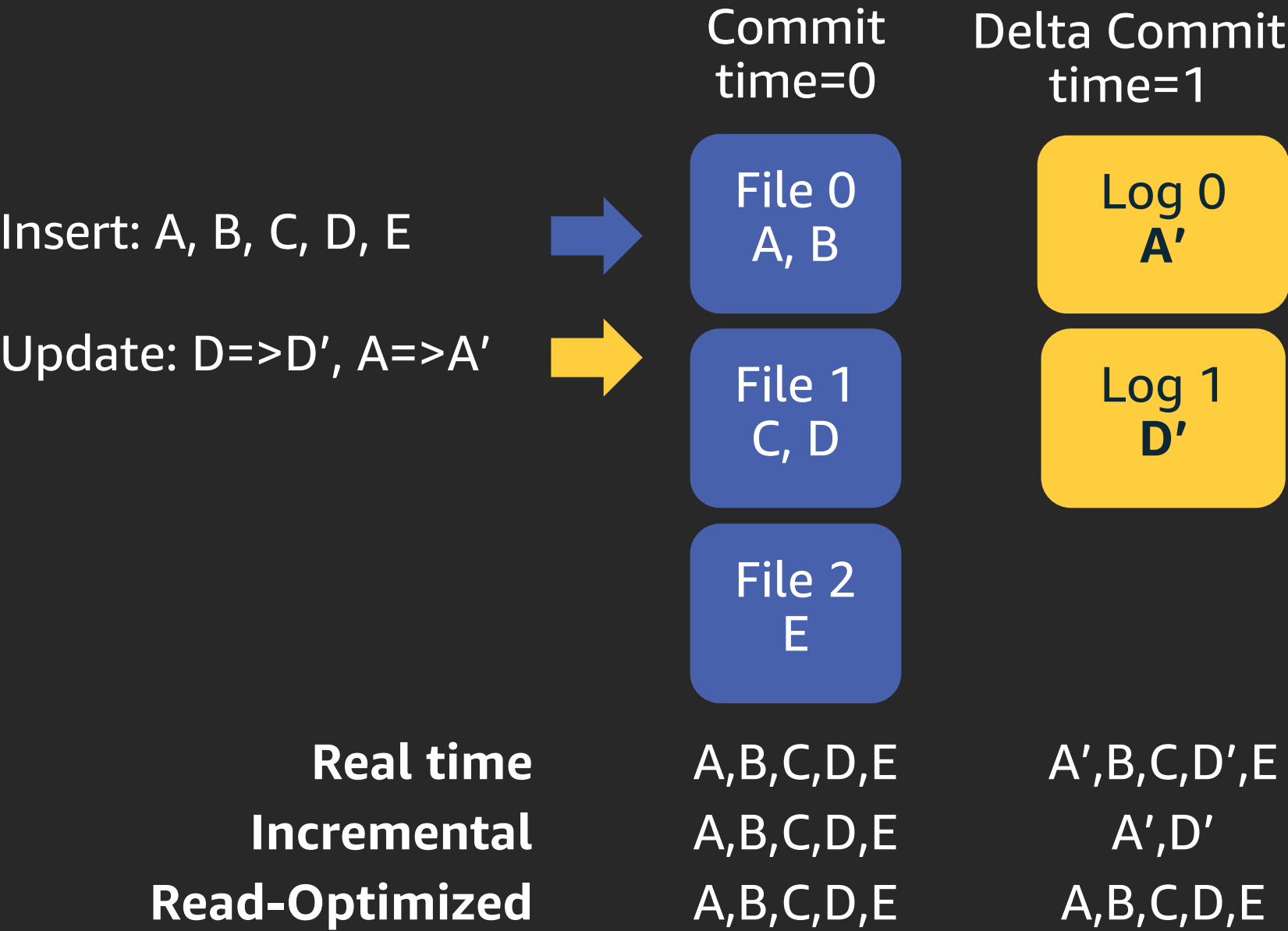
Views/Queries: Read Optimized, Incremental, Real Time



Hudi Storage types & Views - Merge On Read

Storage type: Merge On Read

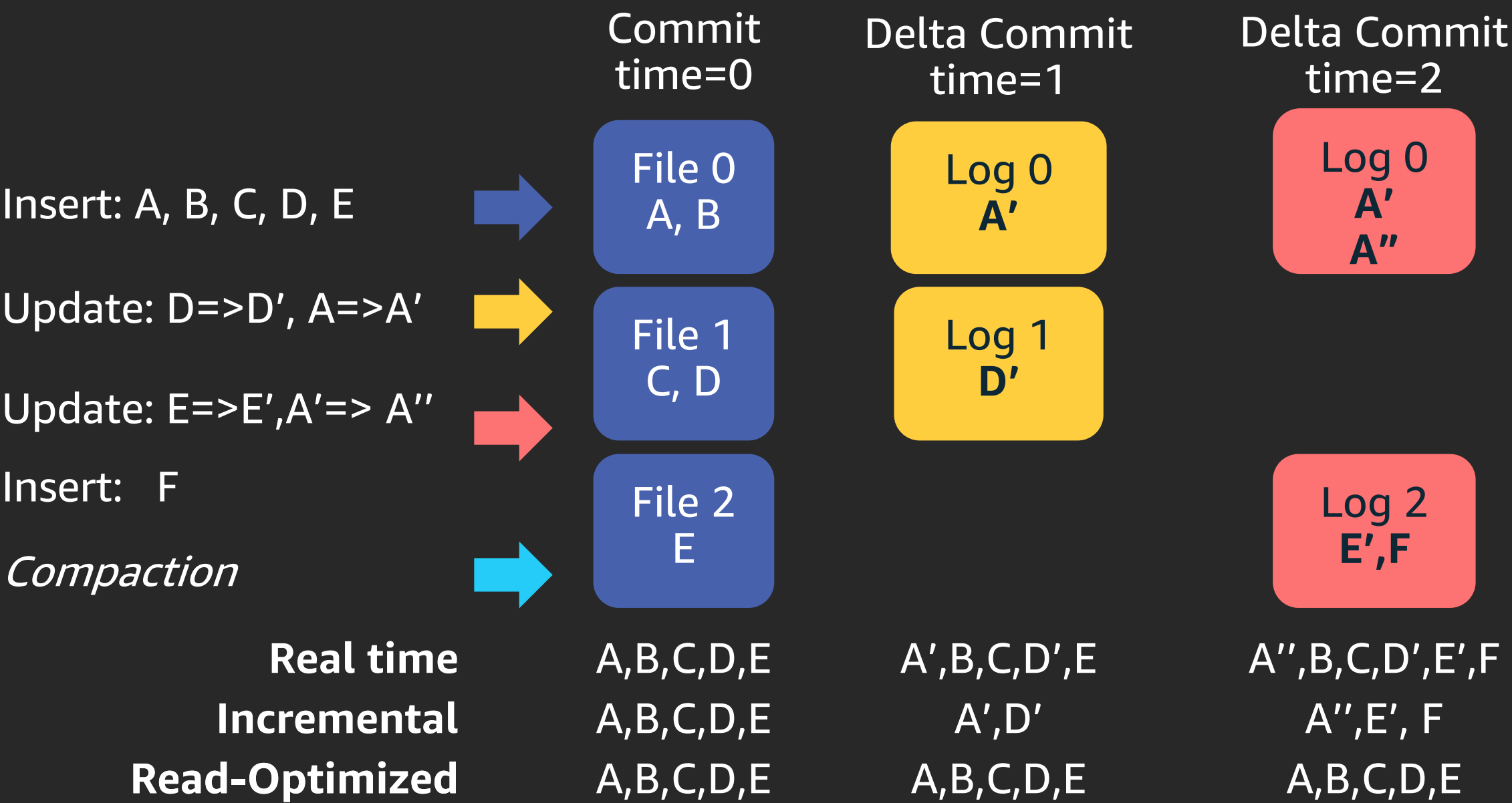
Views/Queries: Read Optimized, Incremental, Real Time



Hudi Storage types & Views - Merge On Read

Storage type: Merge On Read

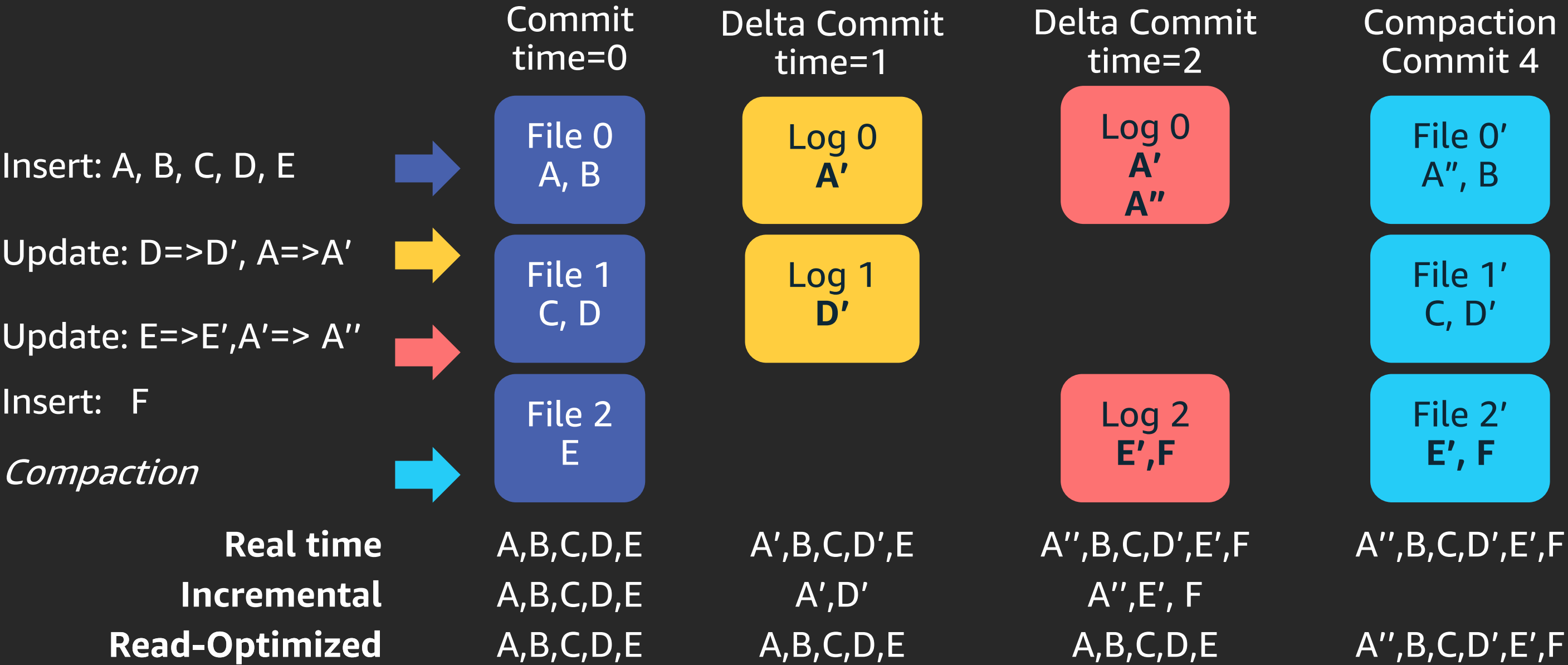
Views/Queries: Read Optimized, Incremental, Real Time



Hudi Storage types & Views - Merge On Read

Storage type: Merge On Read

Views/Queries: Read Optimized, Incremental, Real Time



Hudi Storage types & Views - Merge On Read

Storage type: Merge On Read

Views/Queries: Read Optimized, Incremental, Real Time

언제 사용할 것인가?

- 데이터 수집 시점에 최대한 빠르게 쿼리가 필요할 때
- 워크로드에 갑작스런 변화나 쿼리 패턴의 변화에 대응해야 할 때
 - 사례 : 데이터베이스 변경분의 벌크 업데이트로 인해 대량의 기존 S3 파티션 데이터의 변경 필요

Read-Optimized

Hudi DataSet Sample Code

- Hudi Data Set 저장을 위한 설정 - Storage Type, Record Key, Partition Key

```
/******  
Our Hudi Options for our Product Reviews Dataset.  
*****/  
val hudiOptions = Map[String,String](  
  HoodieWriteConfig.TABLE_NAME -> hudiTableName,  
  
  //For this data set, we will configure it to use the COPY_ON_WRITE storage strategy.  
  //You can also choose MERGE_ON_READ  
  DataSourceWriteOptions.STORAGE_TYPE_OPT_KEY -> "COPY_ON_WRITE",  
  //These three options configure what Hudi should use as its record key,  
  //partition column, and precombine key.  
  DataSourceWriteOptions.RECORDKEY_FIELD_OPT_KEY -> "review_id",  
  DataSourceWriteOptions.PRECOMBINE_FIELD_OPT_KEY -> "timestamp",  
  DataSourceWriteOptions.PARTITIONPATH_FIELD_OPT_KEY -> "review_date",
```

Hudi DataSet Sample Code

- Hudi Data Set 형식으로 S3에 저장 - Bulk Insert

```
/** *****  
Lets write our input dataset to Hudi.  
*****/  
(sourceData2015.write  
  .format("org.apache.hudi")  
  .options(hudiOptions)  
  
  //Operation Key tells Hudi whether this is an Insert, Upsert, or Bulk Insert operation  
  .option(DataSourceWriteOptions.OPERATION_OPT_KEY,  
          DataSourceWriteOptions.BULK_INSERT_OPERATION_OPT_VAL)  
  
  .mode(SaveMode.Overwrite)  
  .save(hudiTablePath))
```

Hudi DataSet Sample Code

- Hudi Data Set 형식으로 저장된 데이터를 로딩하여 SparkSQL로 쿼리

```
/** *****  
Querying Hudi data is easy. We set the format to "org.apache.hudi"  
*****  
val readOptimizedHudiViewDF = (spark.read  
    .format("org.apache.hudi")  
    .load(hudiTablePath + "/*/*/*/*")  
    .cache())
```

```
/** *****  
Lets take a look at our data. Lets say someone says there is something odd going on with star ratings.  
*****  
readOptimizedHudiViewDF.registerTempTable("amazon_product_reviews_raw_ro_table");  
spark.sql("""select star_rating, count(*) from amazon_product_reviews_raw_ro_table  
            group by star_rating order by star_rating ASC""").show()
```

Hudi DataSet Sample Code

- 변경사항을 업데이트/삭제 하기 위해서 대상 데이터를 생성하여 Append

```
/** *****  
Before, if you wanted to update data in S3, you had to read the old data, merge with the new data,  
and then overwrite the old data. Now, with Hudi, you can directly update the data in-place.  
*****/  
(upsertdf.write  
  .format("org.apache.hudi")  
  .options(hudiOptions)  
  .option(DataSourceWriteOptions.OPERATION_OPT_KEY,  
          DataSourceWriteOptions.UPSERT_OPERATION_OPT_VAL)  
  .mode(SaveMode.Append)  
  .save(hudiTablePath))  
  
//Deletion  
(deleteRowsDf.write  
  .format("org.apache.hudi")  
  .options(hudiOptions)  
  //We set the operation to UPSERT  
  .option(DataSourceWriteOptions.OPERATION_OPT_KEY,  
          DataSourceWriteOptions.UPSERT_OPERATION_OPT_VAL)  
  //We set the Payload Class to be empty record  
  .option(DataSourceWriteOptions.PAYLOAD_CLASS_OPT_KEY,  
          "org.apache.hudi.EmptyHoodieRecordPayload")  
  .mode(SaveMode.Append)  
  .save(hudiTablePath))
```

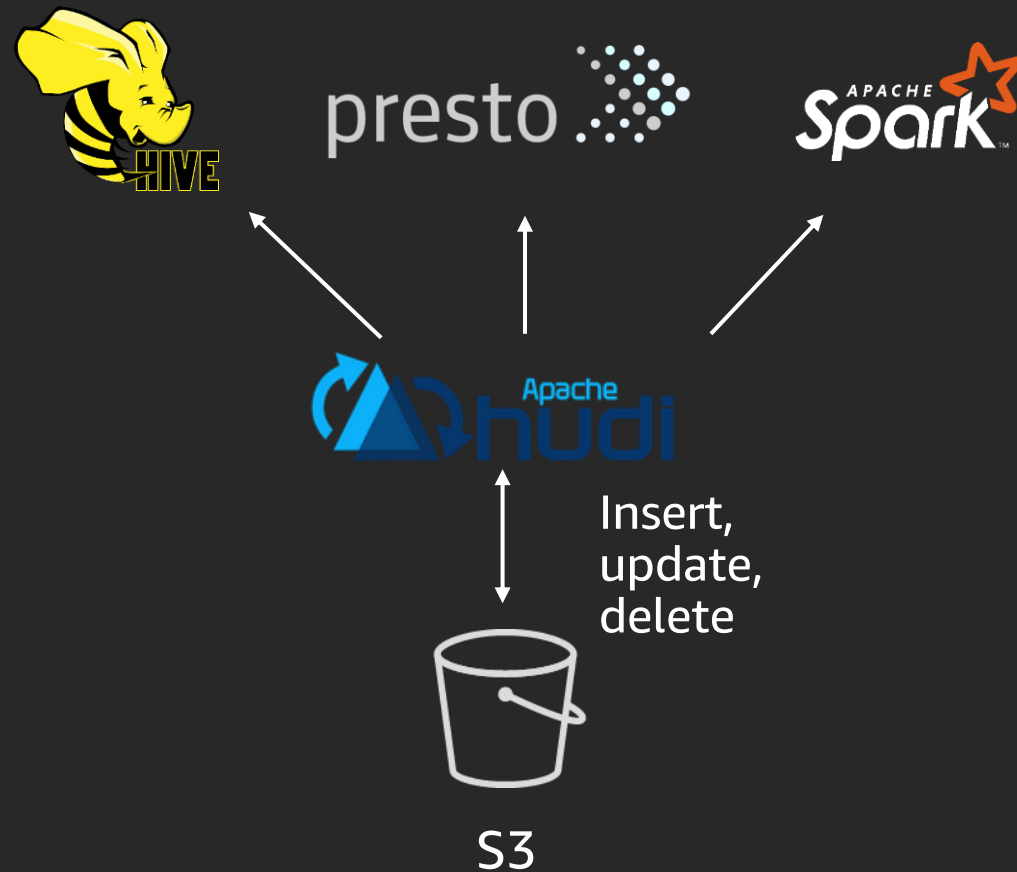
Hudi Dataset 사용을 통한 얻는 장점

Apache HUDI 오픈소스 커뮤니티 기술 지원

Spark, Hive, Presto 지원

Data Lake에서 다음을 가능하게 한다.

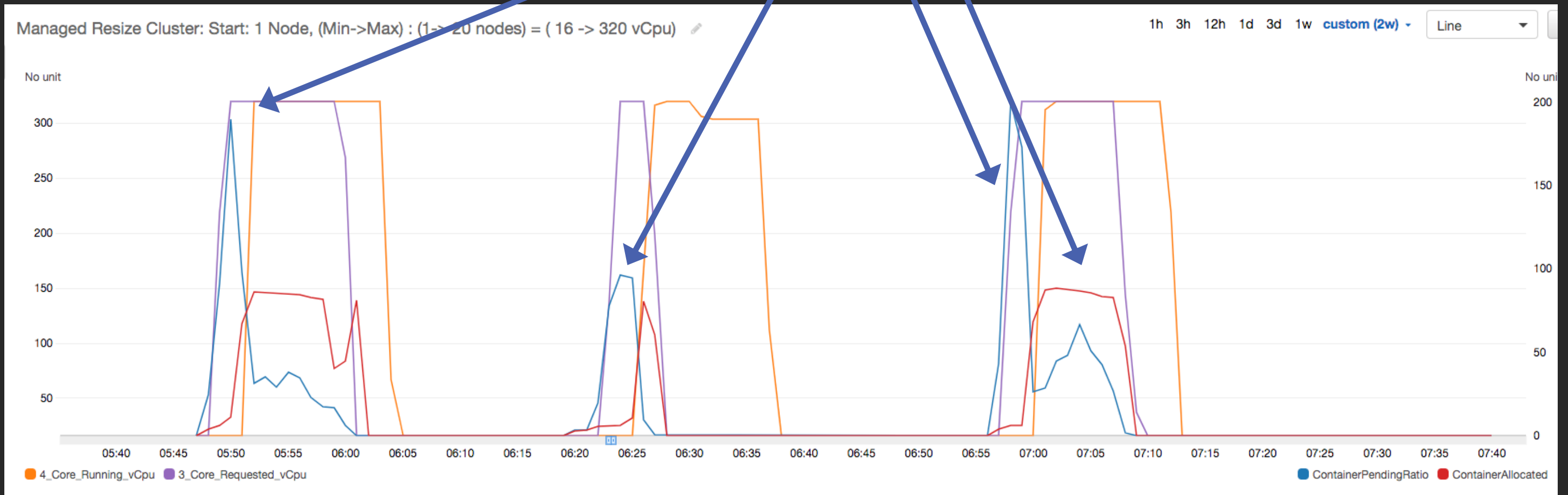
- a) 개인정보 관련법 준수
- b) 실시간 스트림 데이터와 변경분 데이터(CDC) 활용을 효율적으로
- c) 빈번하게 변경되는 데이터 관리
- d) 변경 히스토리의 관리 및 롤백 가능



EMR Managed Resize

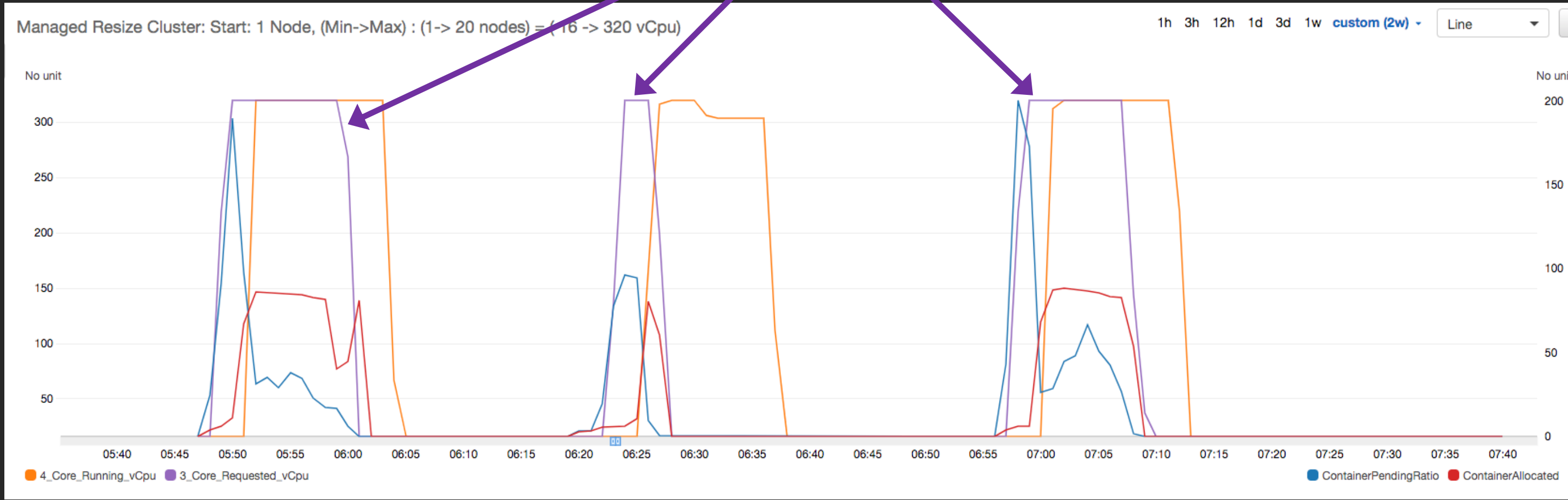
EMR Managed resize

Load patterns



EMR Managed resize

Requested scale



EMR Managed resize

Scale down



EMR Managed resize (베타)

EMR 클러스터 리사이즈를 자동적으로 관리

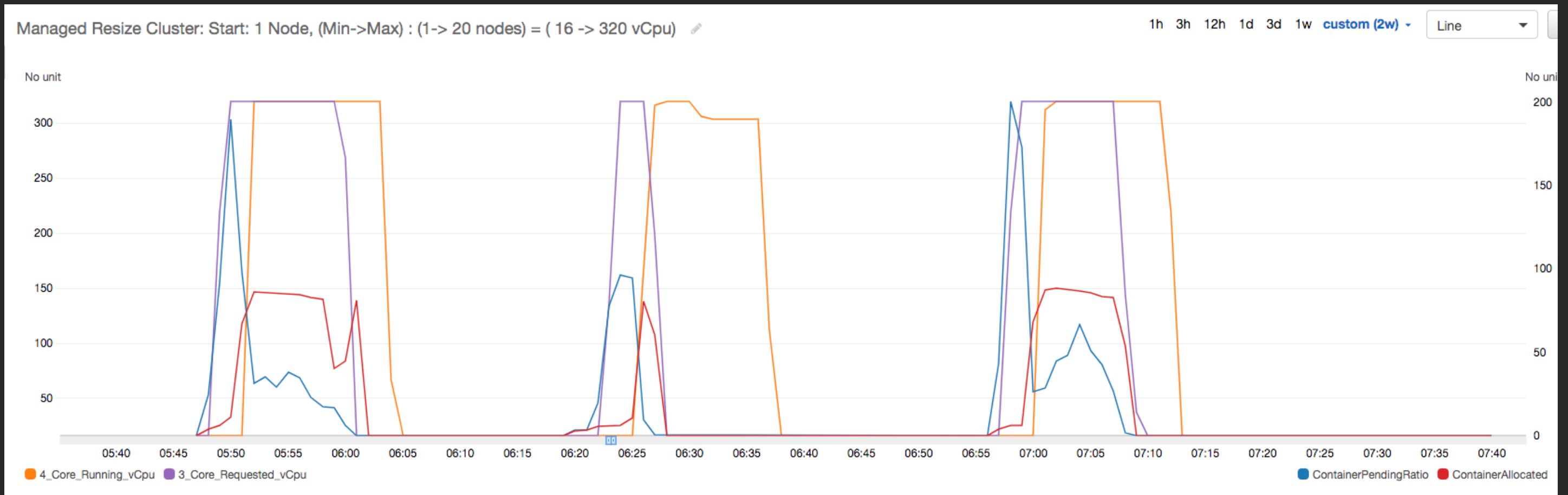
- 최대 / 최소 node 수만 지정하면 다른 설정은 필요 없음
- 모니터링 데이터 기반으로 1분 내외의 빠른 스케일 아웃 가능
- 워크로드에 따라 전체 사용 비용이 20~60% 까지 절약

기존 방식의 오토스케일링 클러스터도 운영 가능

- 사용자의 선택에 따라 직접 커스텀 메트릭을 사용하여 설정하거나
- Managed resize 옵션을 통해 자동화 가능

EMR Managed resize (베타)

63% savings compared to fixed 20-node cluster



Spark Job 디버그를 간편하게 하는 Off-cluster Spark History Service

Off-cluster persistent Spark History Service

Amazon EMR

Clusters

Security configurations

Block public access

VPC subnets

Events

Notebooks

Git repositories

Help

What's new

Clone

Terminate

AWS CLI export

Cluster: DO NOT TERMINATE Parag Test

Terminated

Terminated by user request

Summary

Application history

Monitoring

Hardware

Configurations

Events

Steps

Bootstrap actions

Connections:

--

Master public DNS:

ec2-3-90-84-213.compute-1.amazonaws.com

SSH

History service:

Spark history server UI

(SSH tunneling not required)

Tags:

--

Summary

ID: j-3ODPTHGROJPUA

Creation date: 2019-10-25 13:30 (UTC-8)

End date: 2019-11-24 18:17 (UTC-8)

Elapsed time: 4 weeks

After last step completes: Cluster waits

Termination protection: Off

Configuration details

Release label: emr-5.27.0

Hadoop distribution: Amazon 2.8.5

Applications: Spark 2.4.4, Livy 0.6.0, Hive 2.3.5, Zeppelin 0.8.1, JupyterHub 1.0.0

Log URI: s3://aws-logs-418620260174-us-east-1/elasticmapreduce/

EMRFS consistent view: Disabled

Custom AMI ID: --

Network and hardware

Availability zone: us-east-1e

Security and access

Key name: paragnc_emr_dev_new_us-east-1



Off-cluster persistent Spark History Service

Amazon EMR

Clusters

Security configurations

Block public access

VPC subnets

Events

Notebooks

Git repositories

Help

What's new

Clone

Terminate

AWS CLI export

Cluster: DO NOT TERMINATE Parag Test

Terminated

Terminated by user request

Summary

Application history

Monitoring

Hardware

Configurations

Events

Steps

Bootstrap actions

Connections:

Master public DNS:

History service:

Tags:

Summary

Network and hardware

ID: j-3ODPT

Creation date: 2019-10-

End date: 2019-11-

Elapsed time: 4 weeks

After last step Cluster w

completes:

Termination Off

protection:

Availability zone: us-east-1

APACHE Spark 2.4.4

Jobs

Stages

Storage

Environment

Executors

livy-session-18 application UI

Spark Jobs (?)

User: livy

Total Uptime: 1.0 h

Scheduling Mode: FIFO

Event Timeline

Enable zooming

Executors

Jobs

0

5

10

15

20

25

30

35

40

45

50

55

0

5

10

15

20

23 November 02:11

23 November 02:12

Executor driver added

Executor 1 added

Executor 2 added

Executor 3 added

Executor 4 added

Executor 5 added

Executor 6 added

Executor 1 removed

Executor 2 removed

Executor 4 removed

Executor 6 removed

Executor 3


Executor 5

EMR 어플리케이션 로그 off-cluster 설정

Amazon EMR, Hadoop에서는 클러스터의 상태와 어플리케이션 로그파일을 생성하여 기본적으로 마스터 노드에 기록되며 다음과 같이 확인 가능

- SSH를 통해 마스터 노드 저장 경로(/mnt/var/log)에 연결하여 작업 유형별 로그 파일 확인
- EMR Console에서 Spark history server UI를 통해 확인
- 지정된 S3에 로그를 자동 저장

Spark history server UI


 Use the Spark history server UI to view scheduler stages and tasks, RDD sizes and memory usage, environmental information running executors. Available on running clusters or for up to 30 days for terminated clusters. [Learn more](#)

[Spark history server UI](#) (SSH tunneling not required)

High-level application history

Information about completed Spark applications is shown up to seven days. [Learn more](#)

YARN applications (9)

Filter: All applications 9 applications (all loaded) 

| Application ID | Type | Action | Status | Start time (UTC+9) | Duration |
|--|-------|----------------|-----------|--------------------------|----------|
| application_1581394535946_0009 | Spark | livy-session-8 | Succeeded | 2020-02-12 17:12 (UTC+9) | 2.6 h |
| application_1581394535946_0008 | Spark | livy-session-7 | Succeeded | 2020-02-12 15:56 (UTC+9) | 2.2 h |

Create Cluster - Advanced Options [Go to quick options](#)


Step 1: Software and Steps


Step 2: Hardware

Step 3: General Cluster Settings

General Options

Cluster name

☒ Logging 

S3 folder 

EMR 어플리케이션 로그 보기

EMR 클러스터 생성시 마스터 노드의 로그를 S3에 저장 설정 가능, 저장된 로그를 Athena를 통해 탐색

```
CREATE EXTERNAL TABLE `myemrlogs` ( `data` string COMMENT 'from deserializer')  
ROW FORMAT DELIMITED  
FIELDS TERMINATED BY '|'  
LINES TERMINATED BY '\n'  
STORED AS INPUTFORMAT 'org.apache.hadoop.mapred.TextInputFormat'  
OUTPUTFORMAT 'org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat'  
LOCATION 's3://aws-logs-123456789012-us-west-2/elasticmapreduce/j-2ABCDE34F5GH6'
```

예 – ERROR, WARN, INFO, EXCEPTION, FATAL 또는 DEBUG에 대한 Namenode 애플리케이션 로그 쿼리

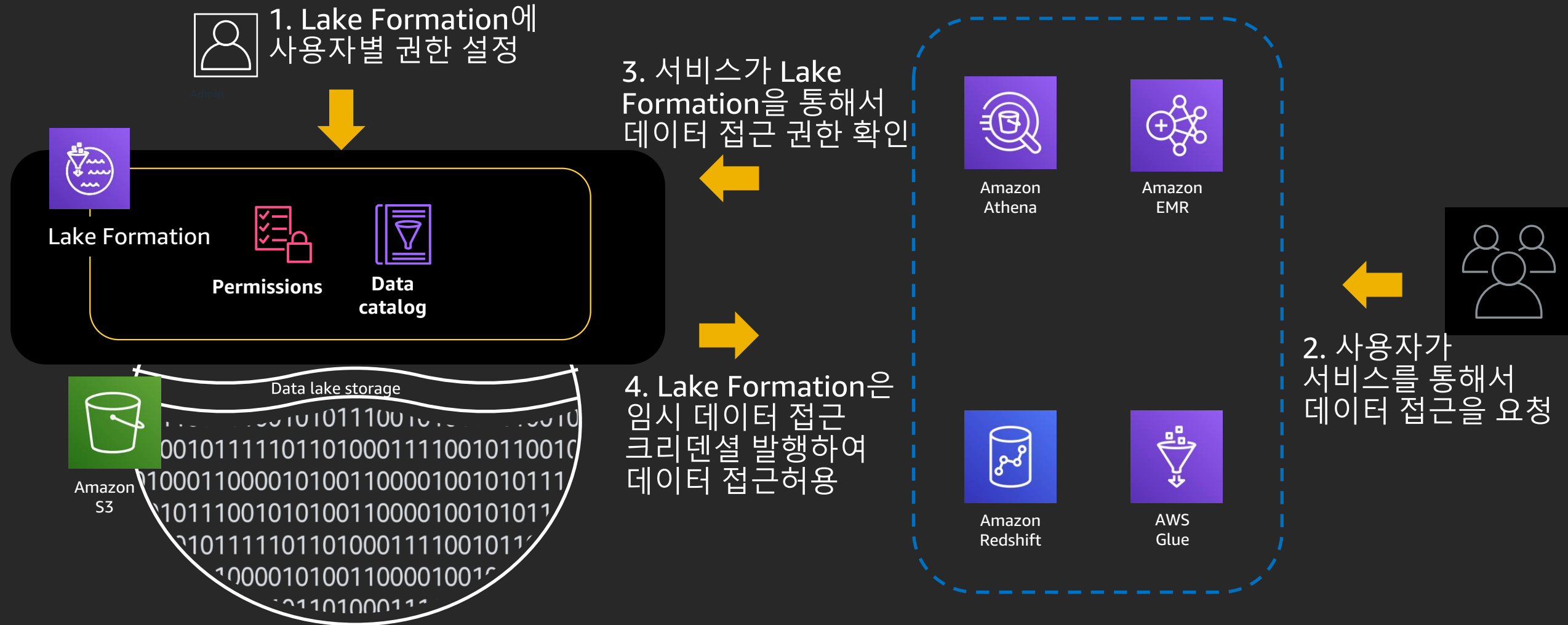
```
SELECT "data", "$PATH" AS filepath FROM "default"."myemrlogs" WHERE regexp_like("$PATH",'namenode') AND  
regexp_like(data, 'ERROR|WARN|INFO|EXCEPTION|FATAL|DEBUG') limit 100;
```

예 – 작업 job_1561661818238_0004 및 Failed Reduces에 대한 Hadoop-Mapreduce 파티션 쿼리

```
SELECT data, "$PATH" FROM "default"."mypartitionedemrlogs" WHERE logtype='hadoop-mapreduce' AND  
regexp_like(data,'job_1561661818238_0004|Failed Reduces') limit 100;
```

Lake Formation과 EMR의 통합

한번 설정으로 모든 데이터를 안전하게 Lake Formation



Lake Formation을 통한 데이터 관리

- 테이블의 컬럼레벨까지의 세부 레벨의 권한 관리가 가능
- AWS Glue Data Catalog와 통합된 메타스토어 제공
- 내부 ID 관리 시스템(AD, Auth0, Okta)와 통합 인증 시스템 연동
- SAML 2.0 파일 지원을 통한 관리 지원
- 다양한 어플리케이션에서 지원
 - Spark SQL
 - EMR Notebooks과 Zeppelin with Livy

Docker 환경에서 Spark application 실행 - EMR 6.0.0(Beta)

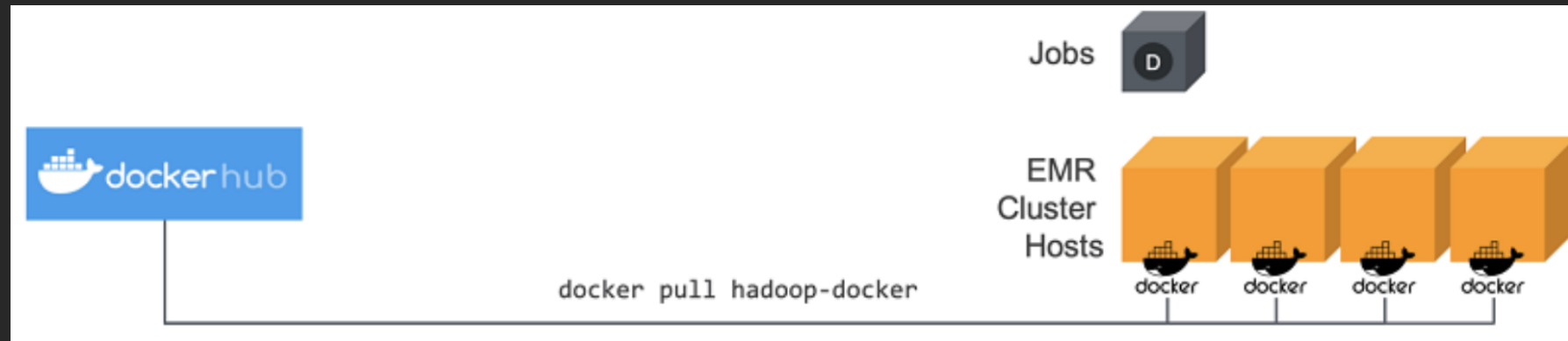
Hadoop 3.0에서 Docker 지원

- Hadoop 3.1.0, Spark 2.4.3 부터 지원, EMR 6.0.0에서 지원 시작
- EMR에서 Docker를 활용함으로써 다음과 같은 장점을 가질 수 있다.
 - 복잡성 감소 - 번들 라이브러리와 어플리케이션의 종속성을 관리해준다.
 - 사용률 향상 - 동일 클러스터에서 다수 버전의 EMR이나 어플리케이션 실행 가능
 - 민첩성 향상 - 새로운 버전의 소프트웨어 신속하게 테스트 하고 생산
 - 응용 프로그램 이식성 - 사용자 운영 환경을 변경하지 않고 여러 OS에서 실행

Docker Registry 선택 옵션 제공

Public subnet

- 인터넷을 통해 YARN에서 Docker Hub와 같은 공개 리파지토리를 선택한 디플로이 지원



Private subnet

- AWS PrivateLink를 통해 Amazon ECR 리파지토리 정보를 통한 디플로이 지원



Docker를 이용한 EMR 클러스터 구성

다음과 같이 container-executor.json 파일을 생성하고 CLI를 통해 EMR 6.0.0 (베타) 클러스터를 시작 가능

```
[ { "Classification": "container-executor",  
    "Configurations": [ {  
      "Classification": "docker",  
      "Properties": { "docker.trusted.registries": "local,centos, your-public-repo,123456789123.dkr.ecr.us-east-  
1.amazonaws.com",  
                    "docker.privileged-containers.registries": "local,centos, your-public-repo,123456789123.dkr.ecr.us-east-  
1.amazonaws.com" } } ] } ]
```

```
$ aws emr create-cluster \  
  --name "EMR-6-Beta Cluster" \  
  --region $REGION \  
  --release-label emr-6.0.0-beta \  
  --applications Name=Hadoop Name=Spark \  
  --service-role EMR_DefaultRole \  
  --ec2-attributes KeyName=$KEYPAIR,InstanceProfile=EMR_EC2_DefaultRole,SubnetId=$SUBNET_ID \  
  --instance-groups InstanceGroupType=MASTER,InstanceCount=1,InstanceType=$INSTANCE_TYPE  
InstanceGroupType=CORE,InstanceCount=2,InstanceType=$INSTANCE_TYPE \  
  --configuration file:///container-executor.json
```


여러분의 소중한 피드백을 기다립니다!
강연 평가 및 설문 조사에 참여해 주세요.

감사합니다