빅데이터 마스터과정
(DAM)

하석재
CEO, 2HCUBE
sjha72@gmail.com

스파크 실습

스파크를 설치하려면

- VirtualBox 설치
- Ubuntu 설치
- 자바 설치
- 하둡 설치

- 파이썬/스칼라 설치
- 스파크 설치

스칼라(Scala) 설치

- 스칼라설치
  - **$ sudo apt install scala**

- 환경변수 설정(~/.bashrc or ~/.profile)
  - **export SCALA_HOME=/usr/bin**

- 스칼라 버전 확인
  - **$ scala -version**

Apache Spark 설치 & 세팅

- 다운로드

  $ **wget** https://dlcdn.apache.org/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz

- 압축해제

  $ tar xvfz spark-3.2.0-bin-hadoop3.2.tgz

Apache Spark 세팅

- 환경설정(~/.bashrc or ~/.profile)
  - **export SPARK_HOME=$HOME/spark-3.2.0-bin-hadoop3.2**
  - **export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin**

  $ cd $SPARK_HOME/conf
  $ cp spark-env.sh.template spark-env.sh

- **$SPARK_HOME/conf/spark-env.sh**
  export SCALA_HOME=/usr/bin
  export SPARK_HOME=/home/ubuntu/spark-3.2.0-bin-hadoop3.2

Apache Spark 세팅

- 하둡 설치(의사분산모드/완전분산모드)
- HDFS 서버 실행 / 종료
  $ start-dfs.sh / $ stop-dfs.sh
  - NameNode / DataNode / Secondary NameNode
- 스파크 서버실행(의사분산모드/완전분산모드)
  - master 실행/종료 **$ start-master.sh / $ stop-master.sh**
  - slave 실행/종료 **$ start-workers.sh / $ stop-workers.sh**
- 실행 확인
  $ jps
  307 Jps
  134 Master
  252 Worker

Apache Spark 세팅

- HDFS 파일복사

  $ hdfs dfs -mkdir /user

  $ hdfs dfs -mkdir /user/ubuntu

  $ hdfs dfs -put $SPARK_HOME/README.md /user/ubuntu/

  $ hdfs dfs -ls /user/ubuntu

- 쉘실행

  $ spark-shell(scala) / $ **pyspark**(python)

스파크 쉘

- 프로토타이핑/디버깅에 유리
- 입력 즉시 결과를 확인
- 스칼라와 파이썬만 지원
  - 자바는 지원하지 않는다
- **spark-shell / pyspark**
  - **:quit / quit()** 으로 탈출
- 스파크 인스턴스를 초기화
  - SparkContext / sqlContext / hiveContext
  - **SparkSession**(2.0 이후부터 통합)
    - sqlContext의 대체
- 4040포트로 모니터링

스파크 스트리밍

배치 / 스트리밍 / 실시간 스트리밍

- 배치(Batch)
  - 전체를 한 번에 실행하는 방식
  - 하둡 맵리듀스 / 스파크
- 스트리밍(Streaming)
  - 일정시간마다 실행되는 방식(mini-batch)
  - 스파크 스트리밍 / 하둡 스트리밍
- 실시간(Real-time)
  - 데이터가 생성되는(도착하는) 즉시 실행하는 방식
  - Kafka with Zookeeper
  - 스파크 스트리밍 + 카프카

스트리밍(비구조화/구조화)

- 스트리밍
  1. 일정시간마다 실행(mini-batch)
  2. 실시간(real-time) - 데이터가 발생하자마자 바로 실행
     - 카프카(Kafka) 결합

- 비구조화 / 구조화
  - Unstructured Streaming / Structured Streaming
  - RDD를 스트리밍 -> 비구조화 스트리밍
  - DF/DT를 스트리밍 -> 구조화 스트리밍(2.2부터)

DStream / Streaming Context

```
import org.apache.spark._
import org.apache.spark.streaming._


val conf = new SparkConf().setMaster("yarn").setAppName("WordCount")
val ssc = new StreamingContext(conf, Seconds(5))
```

Wordcount(Streaming)

```
import org.apache.spark._
import org.apache.spark.streaming._

val ssc = new StreamingContext(sc, Seconds(5))
val lines = ssc.socketTextStream("127.0.0.1", 9999)
val words = lines.flatMap(_.split(" "))    //   a=>a.split()
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)
wordCounts.print()

ssc.start()     // 쓰레드 실행, 버추얼박스 CPU를 2이상 설정
ssc.awaitTermination()
```

스파크 스트리밍

- 윈도우
  - 실행은 5분간격, 계산은 이전 30분 데이터
- 슬라이딩
  - 실행될 때 데이터 이동간격
- 체크포인트
  - 일종의 백업

```
import org.apache.spark._
import org.apache.spark.streaming._

val ssc = new StreamingContext(sc, Seconds(5))
val lines = ssc.socketTextStream("127.0.0.1", 9999)
val words = lines.flatMap(_.split(" "))
val pairs = words.map(word => (word, 1))
val wordCounts = pairs.reduceByKey(_ + _)

ssc.checkpoint("./user/checkpoint")
val wordCountWindow = wordCounts.reduceByKeyAndWindow(
    {(x, y) => x + y}, // 윈도에 들어가는 새로운 배치들의 계산
    {(x, y) => x - y}, // 윈도에서 벗어나는 범위의 값을 계산
    Seconds(30), // 윈도우 시간
    Seconds(10)  // 슬라이딩 시간
)
wordCountWindow.print()
ssc.start()
ssc.awaitTermination()
```

Structured Streaming

- Structured Streaming(2.2부터 정식지원)
- 반정형/정형 스트리밍처리에 사용
    - DStream 으로 처리 내부적으로 RDD변환해서 처리

- **kafka**,flume,kinesis,twitter,mqtt,zeromq과 연동가능

- https://kjhov195.github.io/2019-11-15-structured_streaming_1/

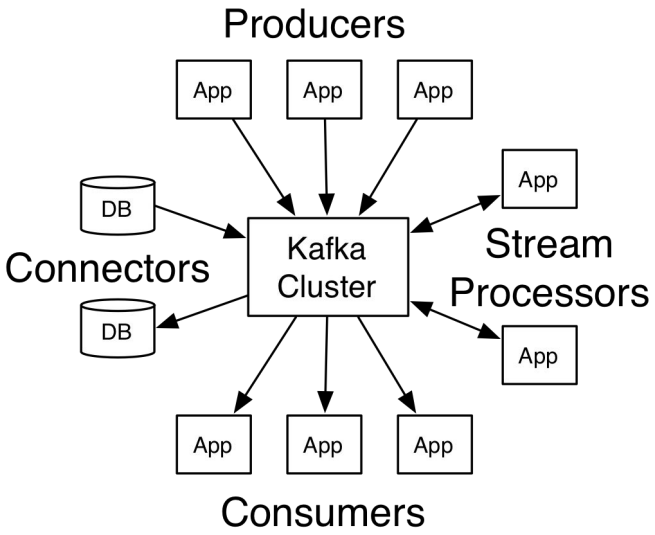
Apache Kafka

Apache Kafka

- Linkedin에서 만든 고성능 분산 메시징용 오픈소스
- MoM(Message-oriented Middleware)용
  - cf. IBM MQ Series / JMS(Java Messaging Service)
- **안정적인 버퍼링(큐잉)**/스트리밍용/Log Aggregation/헬스체크
- 대용량 실시간 로그처리에 특화된 아키텍처
- Producer / Consumer / Broker

Apache Kafka

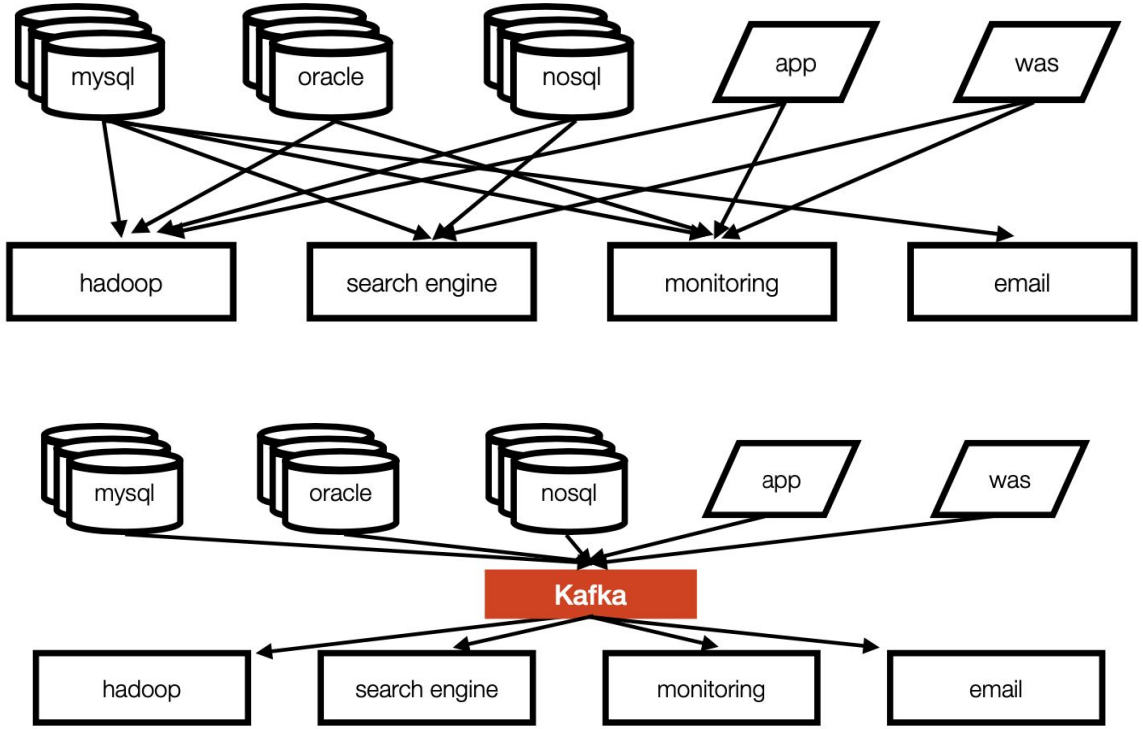
- 대규모로 발생하는 메시지성 데이터 **비동기**방식으로 중계
- 버퍼링하면서 안정적으로 중계
- 데이터를 **topic**에 저장하고 있다가 consumer가 데이터를 전송(소비)



Apache Kafka

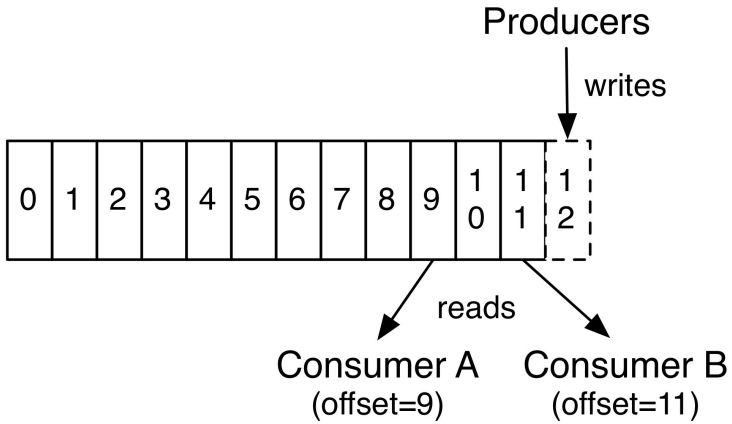
- 용도
    - Messaging System
    - Website Activity Checking/Monitoring:
    - Log Aggregation
    - Stream Processing / Batch Processing:
    - **Buffering**
    - Event sourcing(이벤트를 시간순으로 기록)

카프카 이전-> 이후


mysql
oracle
nosql
app
was
hadoop
search engine
monitoring
email
mysql
oracle
nosql
app
was
Kafka
hadoop
search engine
monitoring
email

Apache Kafka

- **토픽(topic)**을 기준으로 메시지관리



Kafka의 특징

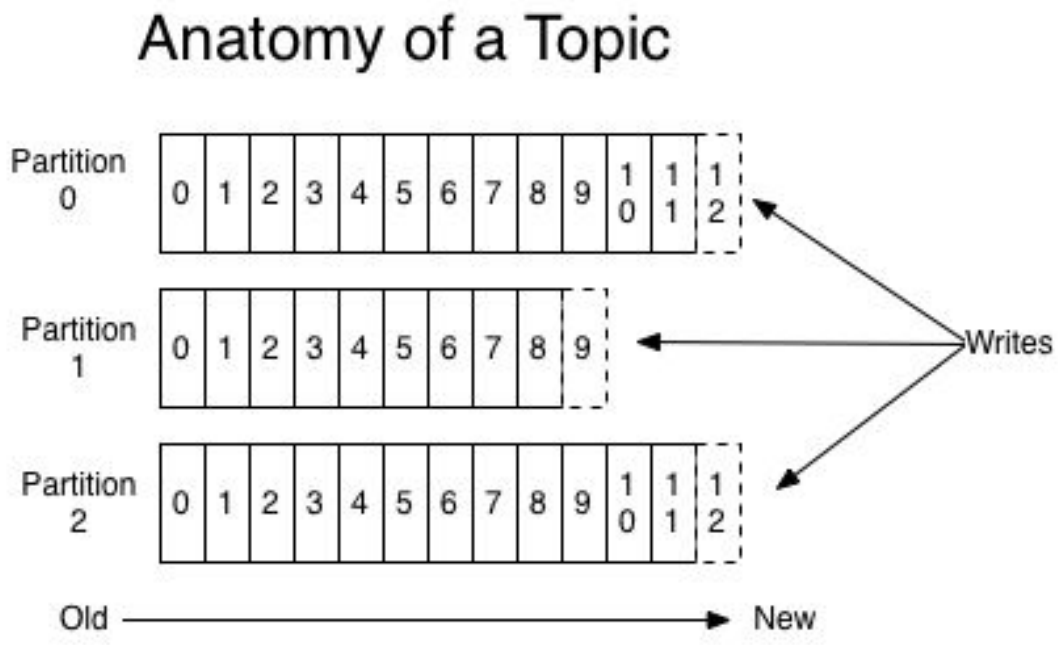
- Publisher/Subscriber 모델
- High Availability / Scalability
- Sequential Store and Process in Disk
  - 장애대응
  - I/O 최적화
- Distributed Processing

Kafka 아키텍처

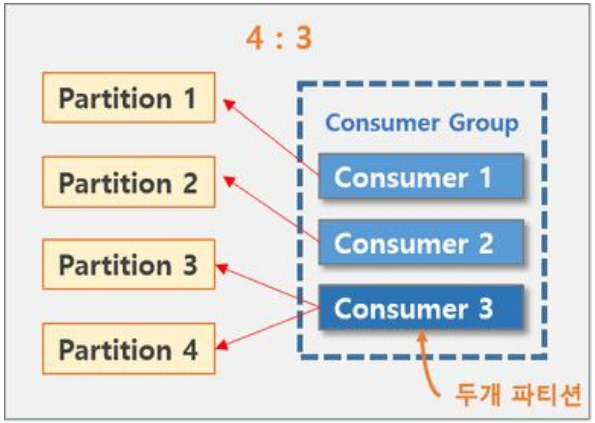
- Pub/Sub 구조
- 브로커(Broker)
- 주키퍼(Zookeeper)
- 토픽(Topic)
- 파티션(Partition)
- 리더(Leader)/팔로워(Follower)
- 컨슈머 그룹(Consumer Group)

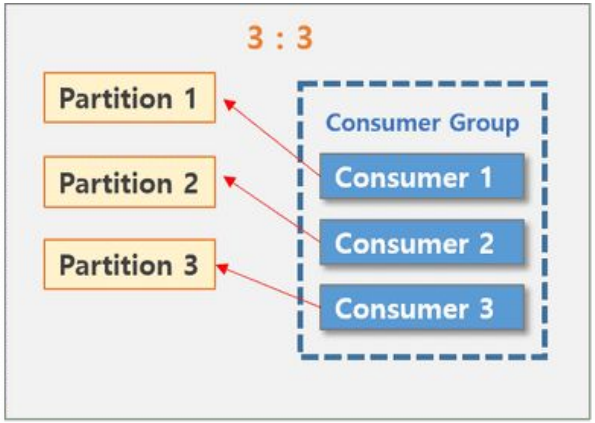
Apache Kafka

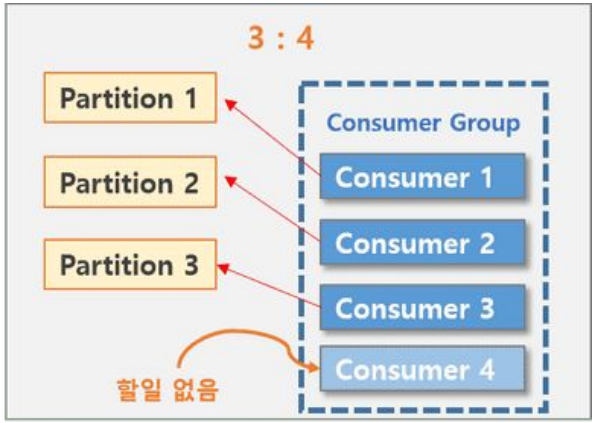
- **토픽(topic)**을 기준으로 메시지관리



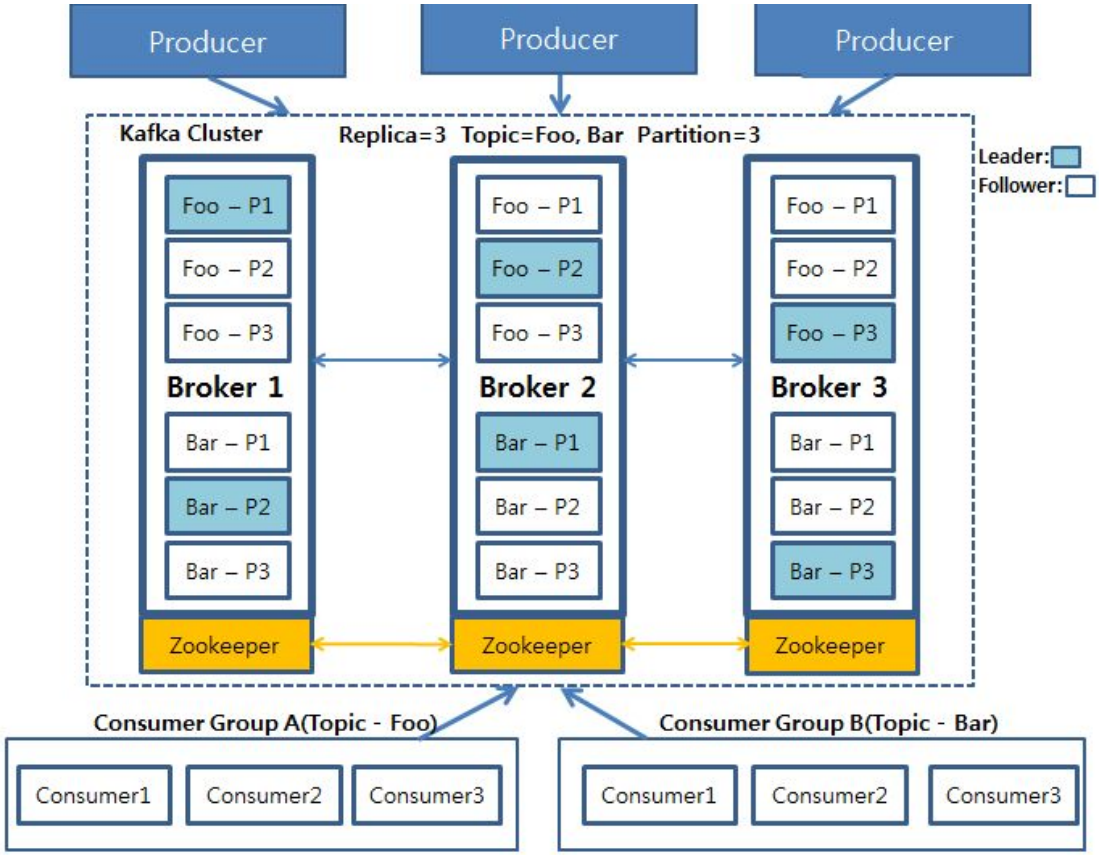
파티션과 컨슈머그룹


4 : 3
Partition 1
Partition 2
Partition 3
Partition 4
Consumer Group
Consumer 1
Consumer 2
Consumer 3
두개 파티션


3 : 3
Partition 1
Partition 2
Partition 3
Consumer Group
Consumer 1
Consumer 2
Consumer 3


3 : 4
Partition 1
Partition 2
Partition 3
Consumer Group
Consumer 1
Consumer 2
Consumer 3
Consumer 4
할일 없음

Kafka Broker 구조



감사합니다