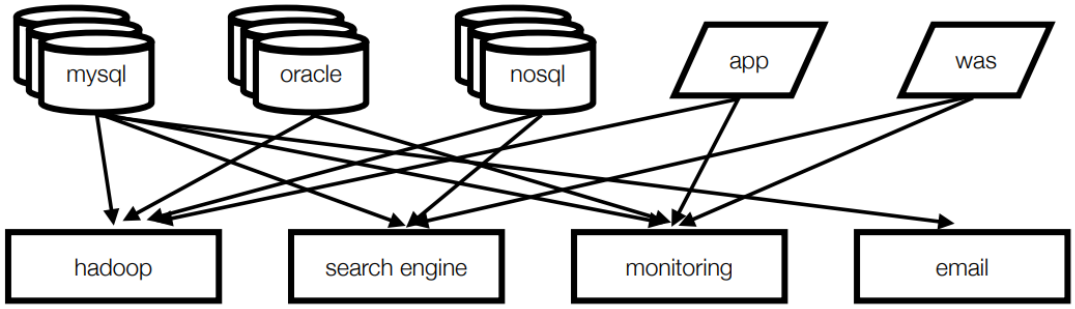
Kafka

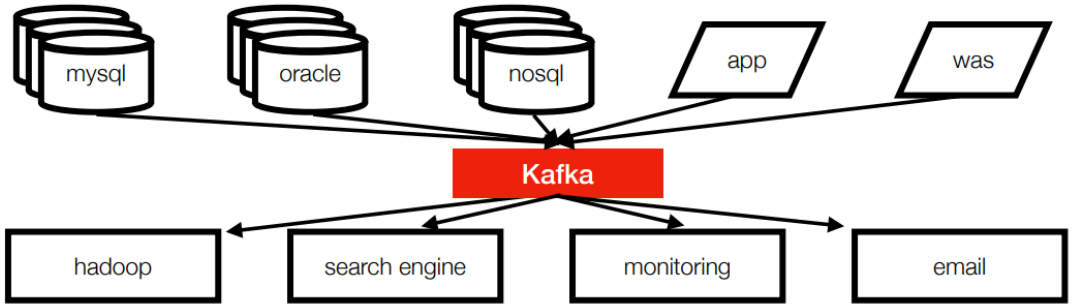
Before Kafka



- 엔드투엔드(end-to-end) 연결 방식의 아키텍쳐
- 데이터 연동의 복잡성 증가(하드웨어, 운영체제, 장애 등)
- 각기 다른 데이터 파이프라인 연결 구조
- 확장에 엄청난 노력 필요

모든 시스템으로 데이터를 전송 실시간 처리도 가능한 것
데이터가 갑자기 많아지더라도 확장이 용이한 시스템이 필요함

After Kafka



- 프로듀서/컨슈머 분리
- 메시지 데이터를 여러 컨슈머에게 허용
- 높은 처리량을 위한 메시지 최적화
- 스케일 아웃 가능
- 관련 생태계 제공

카프카 브로커 & 클러스터

- 카프카 **브로커** : 실행된 카프카 어플리케이션 서버 중 한 대
- **3대 이상의 브로커**로 **클러스터** 구성(홀수 맞출 필요 없이 3대 이상이면 가능)
- **주키퍼**와 연동 (주키퍼는 메타데이터를 저장_브로커id, 컨트롤러id)
- 여러대의 브로커들 중 한대는 **컨트롤러(Controller)**기능을 수행

> 컨트롤러; 각브로커에게 담당 파티션 할당을 수행하거나 브로커가 정상적으로 동작하는지 모니터링한다. 누가 컨트롤러인지는 주키퍼에 저장한다.

Record

```
// Producer Record
// topic은 메세지가 보내지는 저장소(e.g. 테이블)과 비슷한 개념
// topic과 함께 key, message를 지정하여 보낸다.
new ProducerRecord<String, String>("topic", "key", "message");

// Consumer Record
// Consumer record를 통해서 topic의 데이터를 다시 record로 받아올 수 있다.
// record도 key(String), value(String) 쌍으로 받아온다.
ConsumerRecord<String, String> records = consumer.poll(1000);
for(ConsumerRecord<String, String> record : records) {
  ...
}
```

- 객체를 프로듀서에서 컨슈머로 전달하기 위해 **Kafka 내부에 byte 형태로 저장**할 수 있도록 **직렬화/역직렬화가 필요**하기 때문에 위와 같은 방식을 사용한다.
- 기본적으로 **StringSerializer, ShortSerializer** 등이 제공된다.

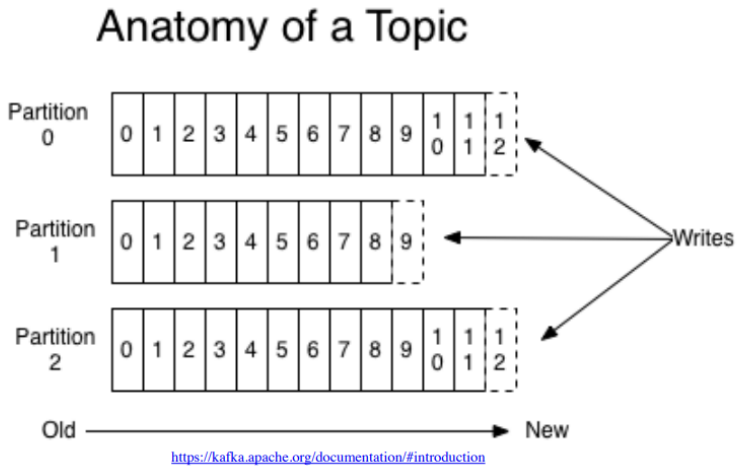
- POJO 사용하기를 원한다면 커스텀 직렬화 class를 통해 Custom Object 직렬화 / 역직렬화 가능

> SK 플래닛에서는 key는 null로 사용하고, value는 Json으로 된 자체 형식을 사용한다고 함
> String으로 되어 있는 Json이지만, byte Array로 Serialize, DeSerialize해서 사용하고 있음.

> Produce, Consume 할때 직렬화와 역직렬화를 동일하게 맞춰주어야 한다.

Topic & Partition

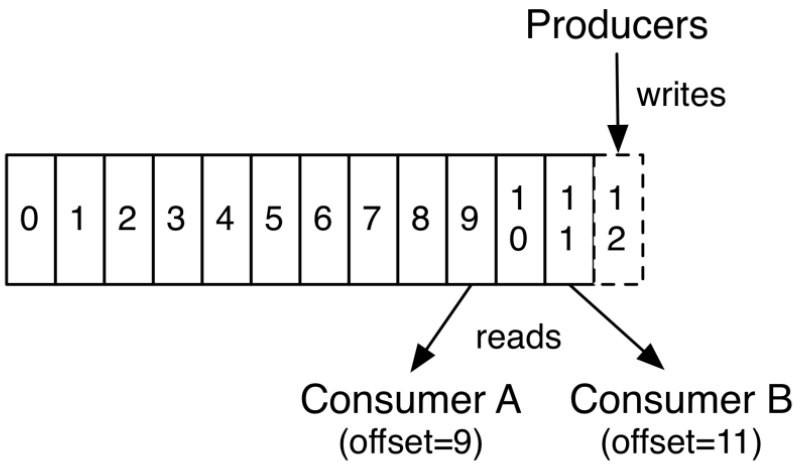
Topic & Partition



https://kafka.apache.org/documentation/#introduction

- Topic에 Partition은 한 개 이상이 반드시 존재해야 한다.
- 각 Partition 내부에서 offset이 존재한다. 0번이 가장 오래된 것, 숫자가 커질수록 최신의 것
- 메세지 처리 순서는 Partition별로 관리된다.
- 파티션이 1개일때는 Queue와 같은 구조가 되어, First In First Out의 방식으로 동작하게 된다.
- 반대로, 파티션이 여러개일 때에는, 메세지 처리 순서가 들어간 순서대로 출력 되는것이 보장되지 않는다.

Producer & Consumer



https://kafka.apache.org/documentation/#introduction

- 프로듀서는 접근한 파티션의 각각의 오프셋이 가지고 있는 레코드를 가져간다.
- 다른 기능을 가진 컨슈머들이 동일한 데이터를 여러 번 가져갈 수 있다.
  (Consumer B가 11번을 가져갔다는 것은 이미 0~10번을 가져갔다는 뜻임)
- 프로듀서는 레코드를 생성하여 브로커로 전송한다.
- 전송된 레코드는 파티션에 신규 오프셋과 함께 기록됨
- **컨슈머는 브로커로부터 레코드를 요청하여 가져감(polling)**
  _ 브로커가 Consumer로 보내는 개념이 아님.

Kafka log & Segment

- 보낸 레코드는 어디에 저장되는가? 실제로는 **파일시스템 단위로 저장**된다.
  DB에 저장되는 것이 아닌 File에 저장된다.
  메세지가 저장될 때는 Segment 파일이라고 하는 ###.index, ###.log, ###.timeindex 와 같은 식으로 저장이 된다.
- 파일은 브로커 또는 토픽에 설정된 **일정 시간 또는 용량 기준에 따라 삭제 또는 압축**이 되게 된다. 삭제된 레코드는 더이상 사용할 수 없다.
- 카프카에 들어간 데이터는 영원하게 사용할 수도 있지만, 보통은 일정 기간, 용량으로 옵션을 주기 때문에 언젠가는 사라진다.

주요 용어 정리

Kafka Client

- Kafka와 데이터를 주고받기 위해 사용하는 Java Library
  - https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients
- Producer, Consumer, Admin, Stream 등 Kafka 관련 api 제공
- Java를 사용하지 않는 사용자들을 위한 3rd Party library를 제공
- Kafka broker 버전과 client 버전의 하위호환 확인이 필요하다.

Kafka Streams

- 데이터를 변환하기 위한 목적으로 사용하는 API
- 스트림 프로세싱을 지원하기 위한 다양한 기능을 제공
  - Stateful 또는 Stateless와 같이 상태기반 스트림 처리가 가능하다
  - Stream API와 DSL(Domain Specific Language)를 동시에 지원한다.
  - Exactly-once처리, 고가용성 처리
    (장애가 나더라도 각각의 offset record를 한번씩만 처리한다)
  - 스트림 처리를 위한 별도 클러스터 불필요
  - Kafka Security 지원

Kafka Connect

- Producer나 Consumer를 Java로 직접 구현해도 된다. 하지만 Kafka connect를 사용하면 데이터를 import또는 export 하는 코드가 제공된다.
- 코드 없이 Configuration으로 이동시키는 것이 목적이다.
- REST API 인터페이스를 통해 제어한다.
- Stream 또는 Batch 형태로 데이터를 전송할 수 있다.
- Standalone mode, distribution mode를 지원한다.
- 커스텀 connector를 통한 다양한 plugin을 제공한다.
  (조회된 데이터를 File, S3, Hive, MySql 등에 데이터를 저장하는 것을 코드작성 없이 가능하도록 한다.)

Kafka Mirror Maker

- 특정 카프카 클러스터에서 다른 카프카 클러스터로 Topic 및 Record를 복제하는 Standalone tool이다.
- 클러스터 간 토픽의 모든것을 복제하는 것을 목적으로 한다.
- 신규 토픽, 파티션 감지기능 및 토픽 설정 자동 Sync 기능을 가지고 있다.
- 양방향 클러스터 토픽 복제 역시 지원하고 있다.
- Mirror Maker가 잘 동작하는지 확인하기 위한 monitoring metric도 제공한다.

그 외 Kafka 생태계를 지탱하는 application들

- confluent/ksqlDB : sql 구문을 통한 stream data processing 지원
  (sql구문을 통해서 data streaming, stream data processing을 지원한다.)
- Schema Registry
- REST proxy : Consumer, Producer를 REST api를 통해서 데이터를 넣고 빼고 하는 기능을 제공하며, open source로 올라와있다.