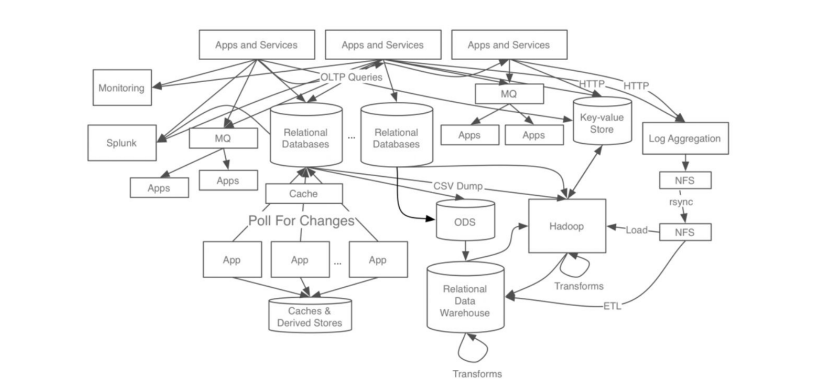
Apache Kafka 이론

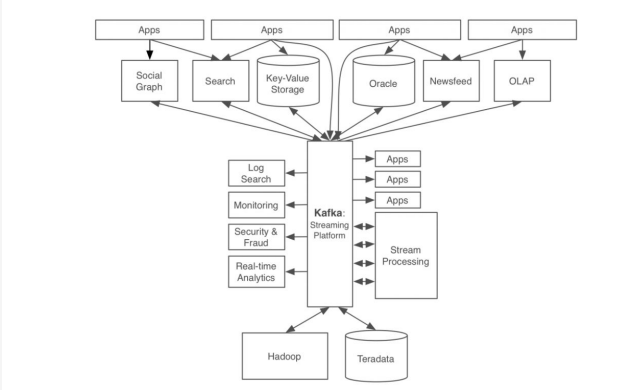
19기 분석 김보겸

Kafka?

- 링크드인(linkedin)에서 개발한 고성능 분산 메시지 처리 시스템



카프카 이전의 링크드인 아키텍쳐



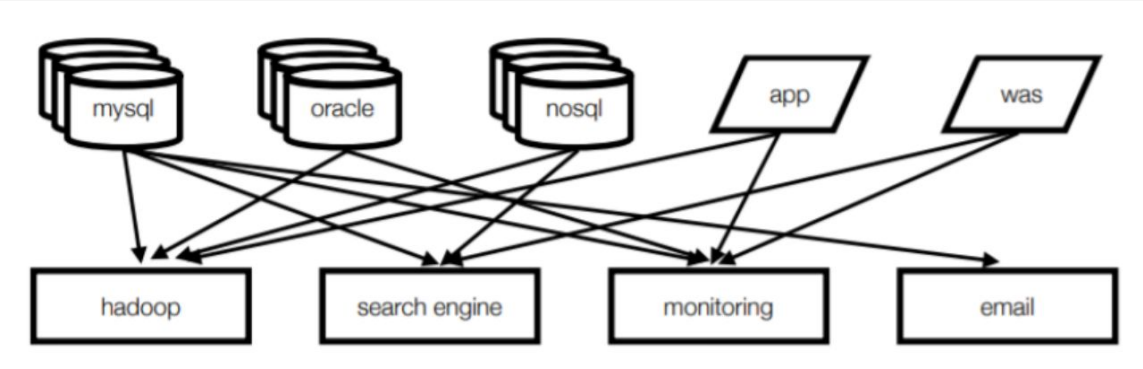
카프카가 개발된 이후의 아키텍쳐

Kafka?

카프카 개발 이전의 아키텍쳐 (Point to Point)

- 데이터를 전송하는 '소스 애플리케이션'과 데이터를 받는 '타겟 애플리케이션'이 각각, 직접적으로 연결

→ 애플리케이션의 개수가 많아질수록 배포와 장애 대응이 어려워짐
→ 데이터 전송의 프로토콜 파편화가 심각해짐

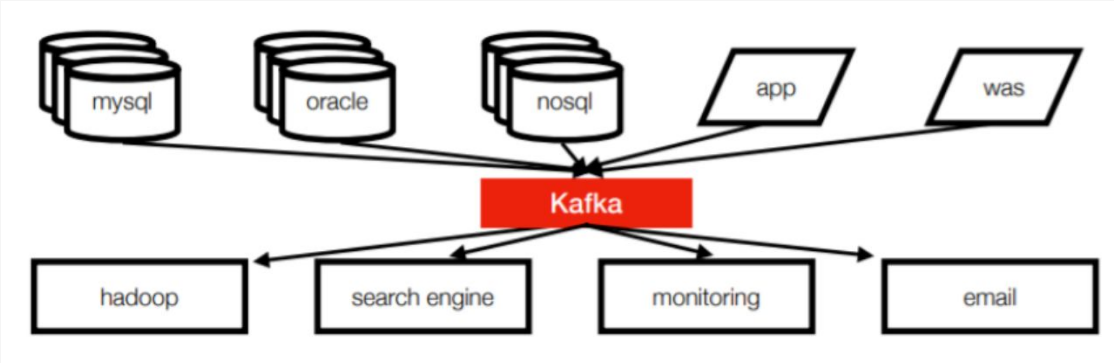


Kafka?

카프카 개발 이후의 아키텍쳐 (Publish/Subscribe)

- 소스 앱은 카프카에 데이터를 보내기만하면 되고, 타겟 앱은 필요한 데이터를 카프카에서 읽어오면 되는 구조

→ 소스 앱과 타겟 앱 사이의 의존성이 완화
→ 데이터의 양이 증가해도 카프카만 고려하면 됨 (확장 용이)



Kafka?

Pub/Sub 구조란?

- Point to Point 구조가 모든 것들이 하나하나 다 연결되어 있는 구조라고 한다면
- Pub/Sub 구조는 통신이 단방향으로 이루어지는 구조

발신자 (Publish)

- 수신자를 따로 생각하지 않고 카프카에 데이터 전송
- 카프카에서는 토픽(Topic)에 저장

수신자 (Subscribe)

- 원하는 카프카의 특정 토픽을 구독
- 구독하는 발신자가 누구인지는 몰라도 됨

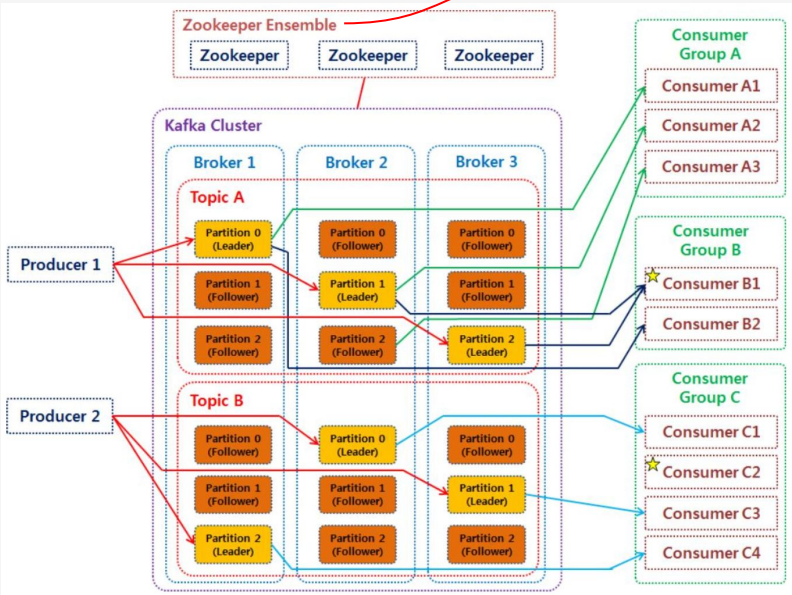
Kafka?

카프카 개발 이후의 아키텍쳐

- 아파트 무인택배보관함 시스템을 생각해봅시다
  - 모든 택배원들이 호수를 방문해서 택배를 직접 놓고 가면 시간과 품이 더 많이 듦
  - 무인택배보관함을 이용할 경우, 기사님들은 호수에 상관없이 한 곳에 넣어두면 되고, 주민들은 가져가기만 하면 됨
    - 택배가 잘못 전송될 확률이 낮아지고, 기사님들은 더 많은 택배 물량을 처리하실 수 있음

Kafka의 구성요소 - Overview

*주키퍼는 카프카에서 점차 빠지는 중 (22.05)

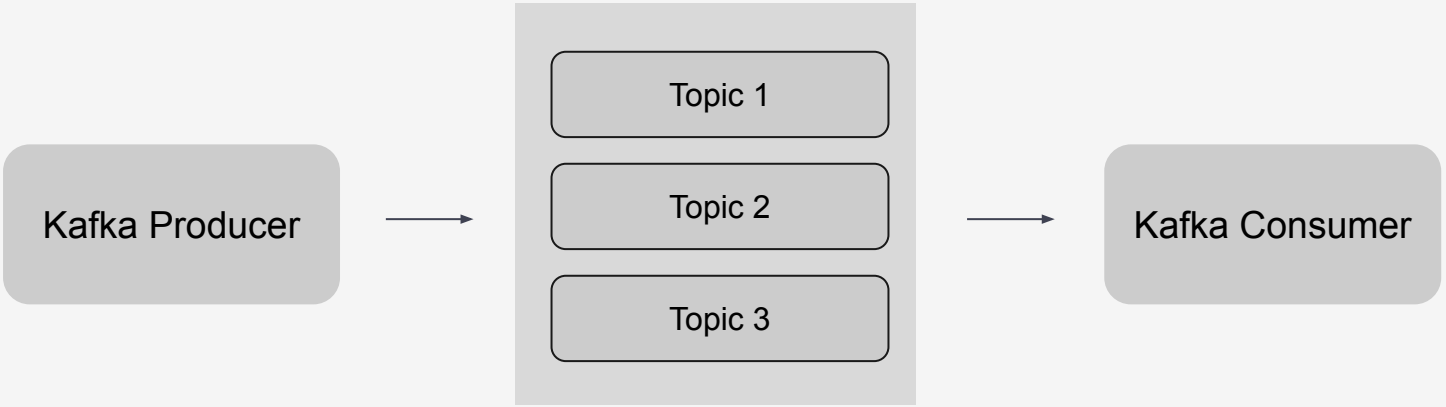


복잡하니 하나하나 자세히 알아봅시다 😉

Kafka의 구성요소 - Overview

가장 기본적인 아이디어

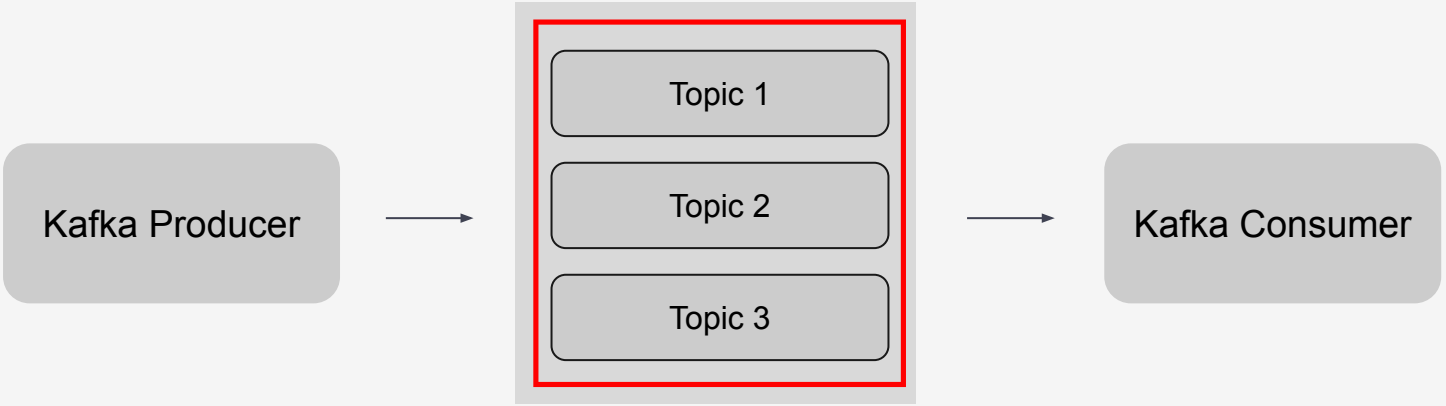
- 카프카에 데이터를 넣는 '프로듀서(Producer)'와 그 데이터를 꺼내가는 '컨슈머(Consumer)'
- 카프카에서 데이터는 '토픽'이라는 공간 안에 들어가게 됨



Kafka의 구성요소 - Topic

토픽 (Topic)

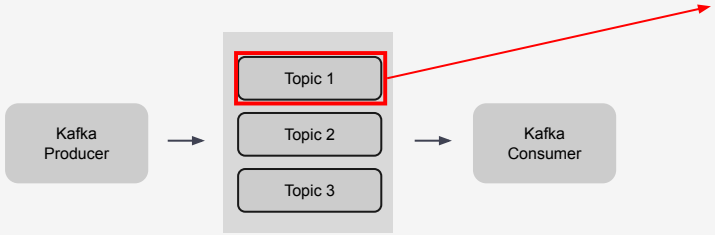
- 카프카에서 메시지를 구분하는 단위 (DB 테이블 혹은 파일 시스템의 폴더와 유사)
  - 프로듀서가 토픽에 데이터를 넣고, 컨슈머는 토픽에서 데이터를 가져감
- 여러 개의 토픽을 생성 가능 (토픽마다 이름을 다르게 설정 가능)
- 하나의 토픽은 (이후에 설명할) 여러개의 파티션(Partition)으로 구성되어 있음

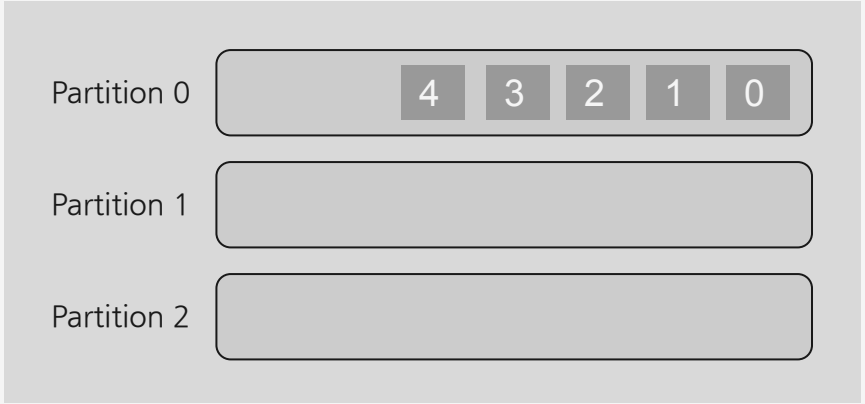


Kafka의 구성요소 - Topic

토픽 (Topic) - 파티션 (Partition)

- 하나의 토픽은 여러 개의 파티션으로 구성되어 있음
- 파티션 내부에서 데이터는 큐와 같이 끝에서부터 차곡차곡 들어감
- 파티션이 여러 개인 경우 데이터 전송
  - Key 값이 존재하는 경우: Key 값의 hash 값을 이용해 할당
  - Key 값이 존재하지 않는 경우: Round-Robin 방식으로 할당

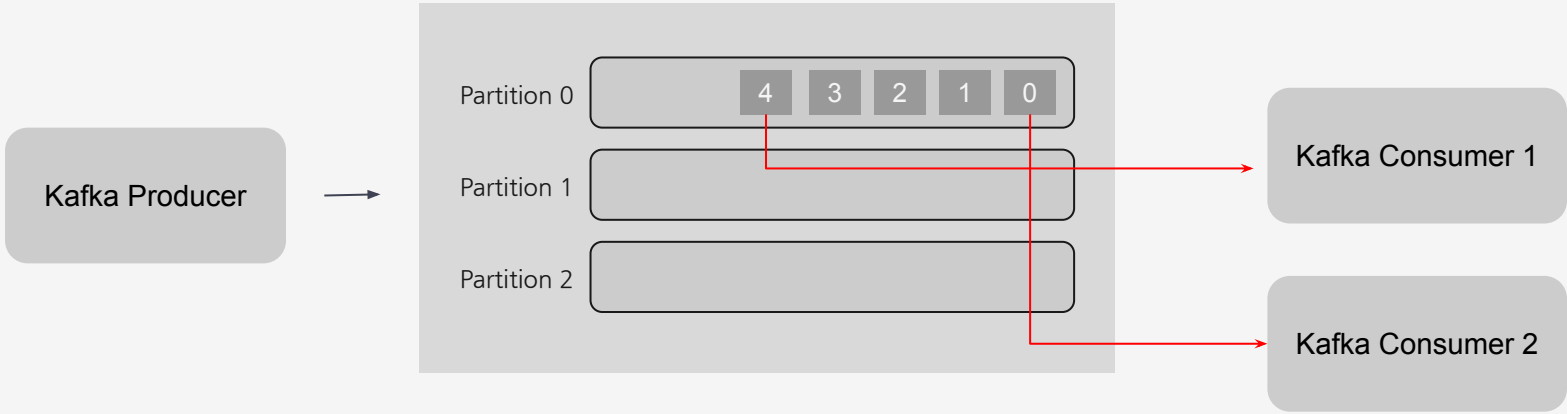




Kafka의 구성요소 - Topic

토픽 (Topic) - 파티션 (Partition)

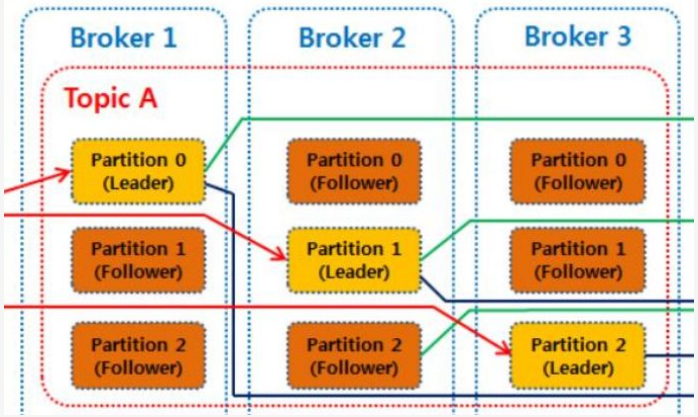
- 카프카 컨슈머(Consumer)는 지정된 파티션의 데이터 중 가장 오래된 데이터부터 순서대로 가져감
- 컨슈머가 데이터를 가져가더라도 파티션 내부에서 데이터는 삭제되지 않음
  - 다른 컨슈머가 동일한 데이터를 가져갈 수 있음 (동일한 데이터에 대해 다른 용도로 처리가능)



Kafka의 구성요소 - Broker

브로커 (Broker)

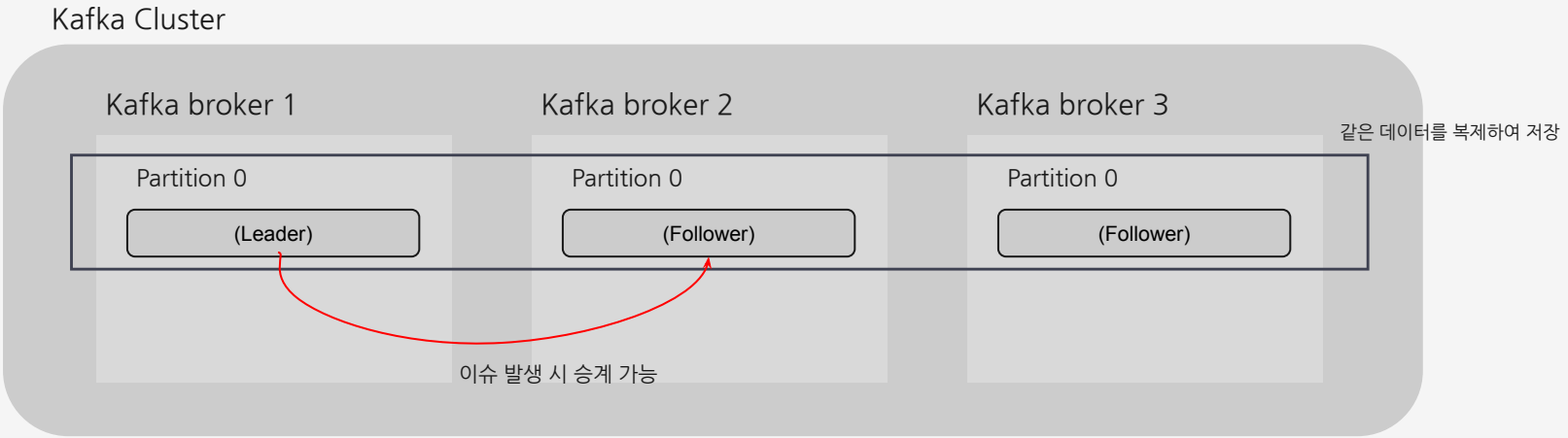
- 카프카의 서버: 브로커 안에 데이터를 담는 '토픽'이 들어간다고 생각해주세요
- 3대 이상의 브로커로 클러스터를 구성하는 것을 권장
  - N개의 브로커 중 1대는 컨트롤러 (Controller) 기능을 수행함
    - 각 브로커에게 담당 파티션 할당 / 브로커 정상 동작 모니터링



Kafka의 구성요소 - Broker

브로커 (Broker) - 토픽 레플리케이션 (Topic Replication)

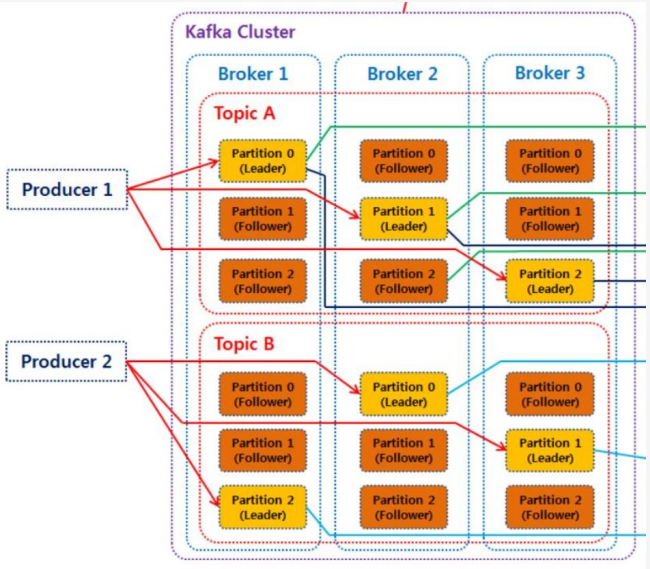
- 여러 대의 브로커를 설정하여 동일한 데이터를 복제해서 각각 저장하는 레플리케이션(replication)을 사용할 수 있음
  - 원본 파티션을 리더 파티션 (Leader Partition), 복제본을 팔로워 파티션 (Follower Partition)이라 함
  - 리더 파티션, 팔로우 파티션을 합쳐 ISR (In Sync Replica)라고 부름
- 어떤 브로커가 모종의 이유로 사용불가하게 되더라도 복제된 다른 팔로워 파티션을 통해 기능 정상 수행 가능



Kafka의 구성요소 - Producer

프로듀서 (Producer)

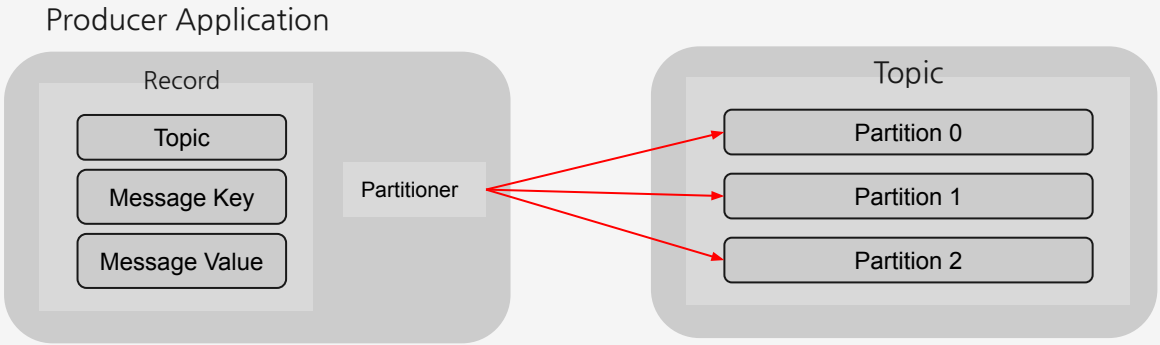
- 데이터를 토픽에 생성하는 역할을 수행
- 메시지의 키 값을 설정하여 특정 토픽에 보낼 수 있음



Kafka의 구성요소 - Producer

프로듀서 (Producer) - 파티셔너 (Partitioner)

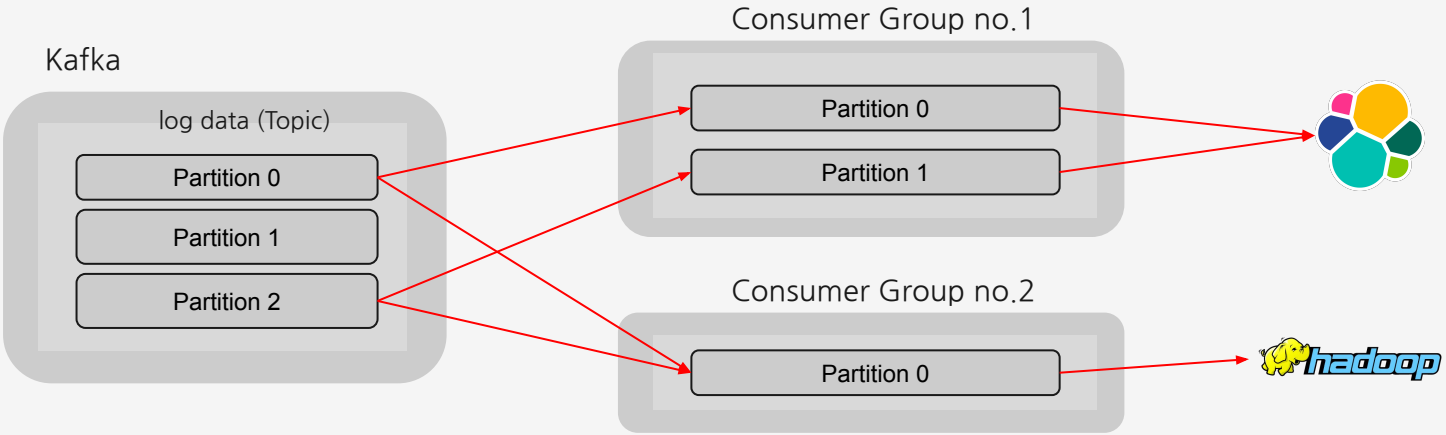
- 프로듀서는 반드시 파티셔너를 통해 브로커로 데이터를 전송
- 파티셔너는 데이터를 토픽의 어떤 파티션 내에 넣을지 결정하는 역할 수행
  - Key 값을 가진 경우: 레코드는 파티셔너에 의해 특정한 해쉬값으로 생성되어 그 값을 기준으로 할당
  - Key 값이 없는 경우: Round-Robin 방식으로 파티션에 할당됨
- 커스텀 파티셔너를 만들어 데이터 전송 방식(어떤 내용의 데이터를 얼만큼 보낼 것인가)을 커스터마이징 할 수 있음



Kafka의 구성요소 - Consumer

컨슈머 (Consumer)

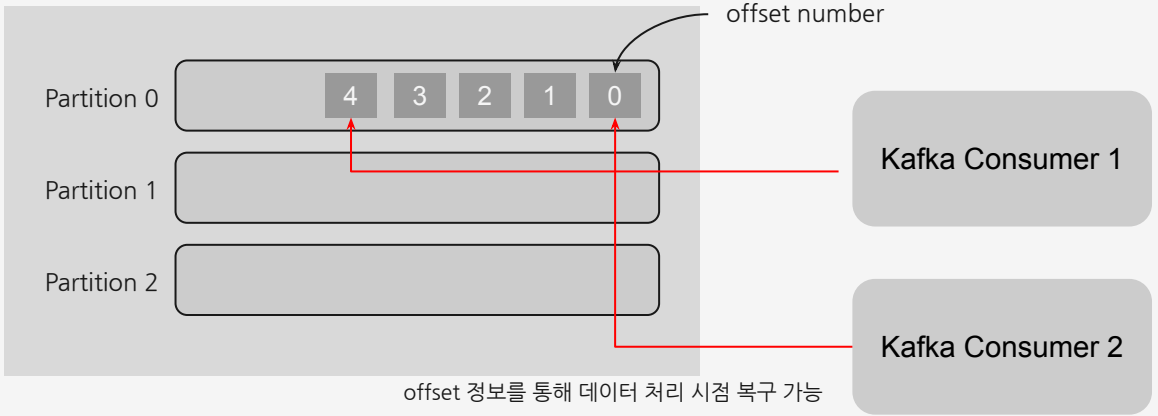
- 컨슈머가 토픽에서 데이터를 가져오는 것을 '폴링 (Polling)'이라고 함
  - 컨슈머가 토픽에서 데이터를 가져가더라도 데이터가 사라지지 않음
- 컨슈머를 여러 개 만들 수 있으며, 컨슈머 그룹 (Consumer Group)'으로 묶어 관리할 수 있음
  - 그룹에 속하는 컨슈머가 여러 개이면 로드밸런싱을 통해 메시지를 분배 가능
  - 컨슈머 그룹들끼리는 서로 영향을 미치지 않음 (토픽 안에 있는 데이터를 변경하거나 하지 않음)



Kafka의 구성요소 - Consumer

컨슈머 (Consumer) - 오프셋 커밋 (Offset Commit)

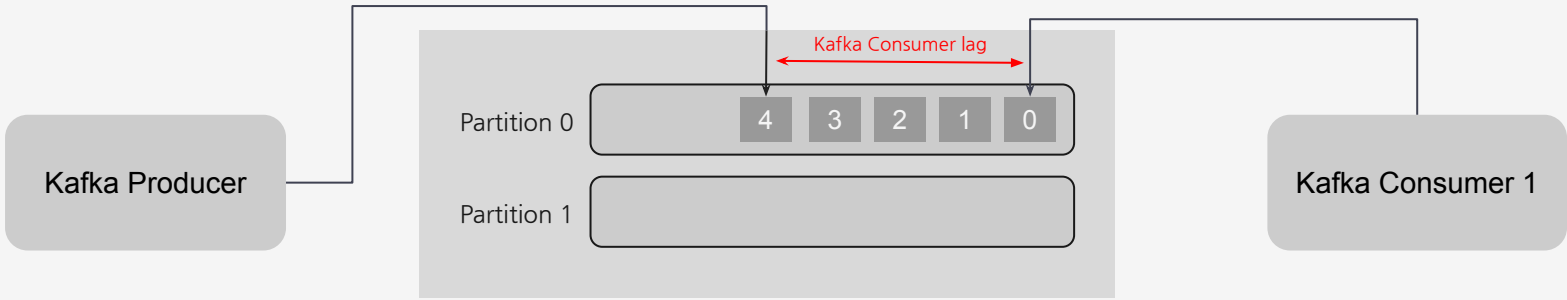
- 파티션 안의 데이터는 고유한 번호를 가지게 되는데, 이 번호를 오프셋(Offset)이라 부름
  - 오프셋은 토픽별로 그리고 파티션 별로 별개로 지정
- 컨슈머들은 파티션에 자신이 가져간 데이터의 오프셋을 기록하고 있음
- 각 파티션 내 데이터들에 대해 어디까지 읽었는지를 업데이트하는 동작을 커밋(Commit)한다고 함
- 컨슈머가 갑자기 동작이 중지되어도 이 오프셋 정보를 통해 데이터 처리 시점을 복구할 수 있음



Kafka의 구성요소 - Consumer

컨슈머 (Consumer) - 컨슈머 랙 (Consumer Lag)

- 복습: 프로듀서는 파티션 내에 데이터를 넣고, 컨슈머는 데이터를 가져감 / 파티션 내 데이터는 고유의 오프셋을 가짐
- 프로듀서가 넣은 데이터의 오프셋과 컨슈머가 마지막으로 읽은 오프셋 간의 차이가 발생 가능
- 이 차이를 컨슈머 랙(Consumer Lag)이라 부름
  - 컨슈머의 기능이 저하되거나 비정상동작을 하는 경우 주로 발생
- 한 개의 토픽과 컨슈머 그룹에 대한 lag이 여러개 존재할 수 있고, 그 중 높은 숫자의 lag을 records-lag-max라고 부름



Kafka Streams

- 토픽에 있는 데이터를 낮은 지연과 함께 빠른 속도로 처리 가능 (자바 라이브리 / JVM 기반 언어로 사용 가능)

- 카프카와 완벽히 호환가능
  - 카프카의 최신 버전과 완벽히 호환 / 유실이나 중복 처리되지 않고 한 번만 처리되는 기능을 갖추고 있음

- 스케쥴링 도구가 따로 필요 없음
  - 스파크 스트리밍을 운영하기 위해서는 클러스터 관리자, 리소스 매니저가 필요한 반면 스트림즈에서는 필요 없음
  - 컨슈머 애플리케이션 배포하는 것처럼 원하는 만큼 애플리케이션을 띄워서 배포하면 됨

- 스트림즈DSL과 프로세서API를 제공
  - 스트림즈DSL을 사용해서 대부분 구현 가능 (이벤트 기반 데이터 처리를 할 때 필요한 map, join, window등을 제공)
  - KStream, Ktable, GlobalKTable을 통해 카프카를 대규모 key-value 저장소로도 활용할 수 있도록 함
  - 스트림즈DSL에 없는 기능은 프로세서API를 통해 구현

- 자체적인 로컬 상태 저장소를 사용
  - 로컬의 rocksdb를 통해 상태를 저장하고, 이 상태에 대한 변환 정보를 카프카의 변경로그 토픽에 저장

Kafka Connect

데이터 파이프라인을 반복적으로 개발, 운영에 사용하기 위한 도구

커넥트 (Connect)

커넥터(Connector)를 동작하도록 실행해주는 프로세스

싱크 커넥터 (Sink Connector)
- 특정 토픽에 있는 데이터를 특정 저장소에 저장을 하는 역할 (컨슈머 비슷)

소스 커넥터 (Source Connector)
- 데이터베이스로부터 데이터를 가져와서 토픽에 넣는 역할 (프로듀서와 비슷)

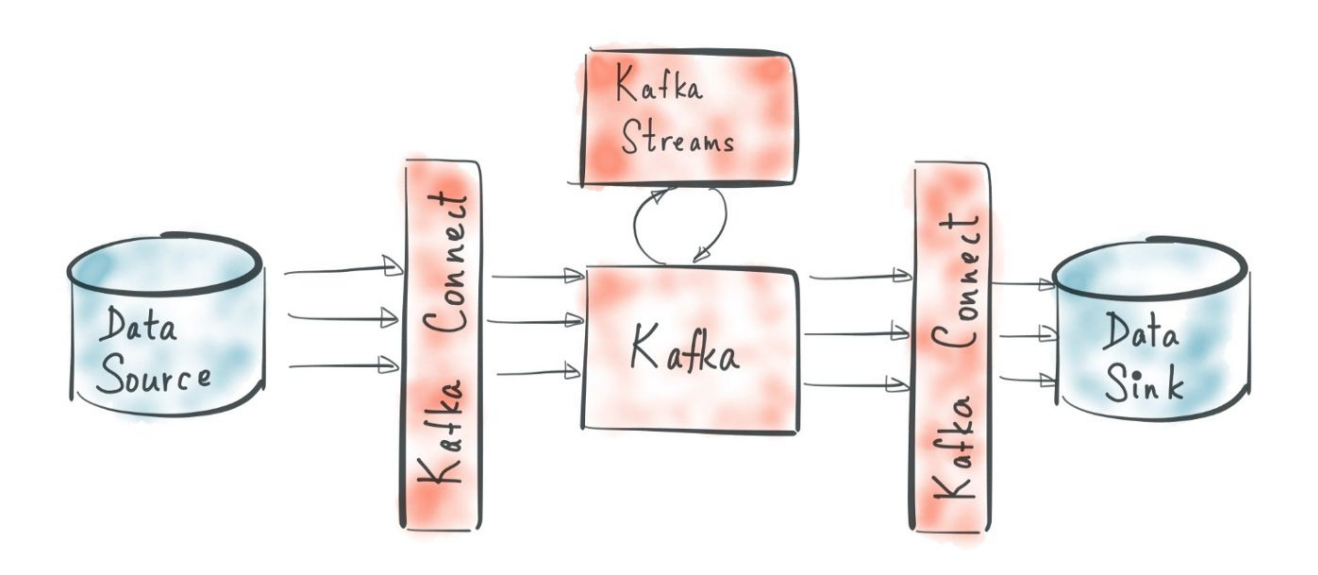
커넥터 (Connector)

데이터를 처리하는 코드가 담긴 jar 패키지,
일련의 템플릿과 같은 특정 동작을 위한 코드 뭉치

단일 실행모드 커넥트
- 간단한 데이터 파이프라인 구성 혹은 개발 용도

분산 모드 커넥트
- 여러 개의 프로세스를 하나의 클러스터로 묶어서 운영하는 방식,
  즉 2개 이상의 커넥트가 하나의 클러스터로 묶임

Kafka Connect + Streams



참고자료

- Steve Jang, [카프카] Pub/Sub 구조와 프로듀서, 컨슈머
- 데브원영, 빅데이터의 기본 아파치 카프카! 개요 및 설명 | What is apache kafka?
- 데브원영, 아파치 카프카 | 데이터가 저장되는 토픽에 대해서 알아봅시다
- 데브원영, 아파치 카프카 | 데이터를 카프카로 전송하는 프로듀서
- 데브원영, 아파치 카프카 | Broker, Replication, ISR 핵심요소 3가지!
- 데브원영, 아파치 카프카 | 카프카 컨슈머 역할 및 코드예제
- 데브원영, 카프카 컨슈머 Lag이란? Lag에 대해서 알아봅시다
- 데브원영, 카프카 스트림즈! 대용량, 폭발적인 성능의 실시간 데이터 처리!
- 데브원영, 데이터 파이프라인을 가장 효율적으로 개발, 배포, 운영하는 방법! | 카프카 커넥트 Kafka Connect
- Rohan Mudaliar, How to Build your First Real-Time Streaming(CDC) system(Kafka Streams and Aggregation-Part3)

경청해주셔서 감사합니다