KAFKA

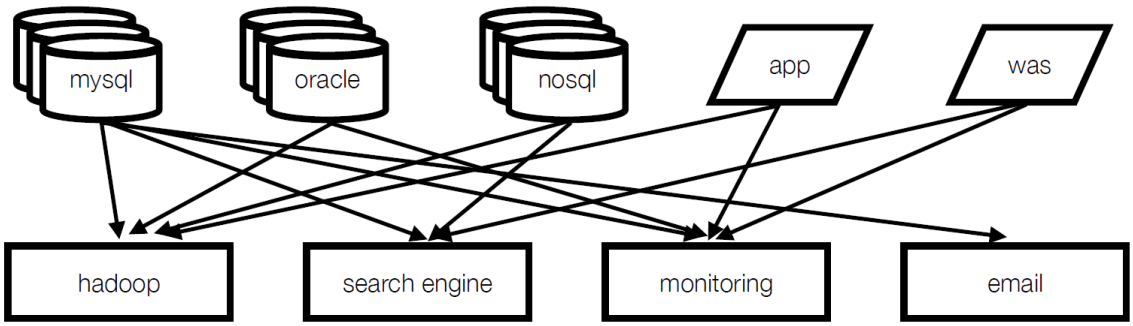
About KAFKA

카프카는 무엇인가

- 대용량, 대규모 메세지 데이터를 빠르게 처리하도록 개발된 분산 메시징 플랫폼
- 링크드 인에서 개발

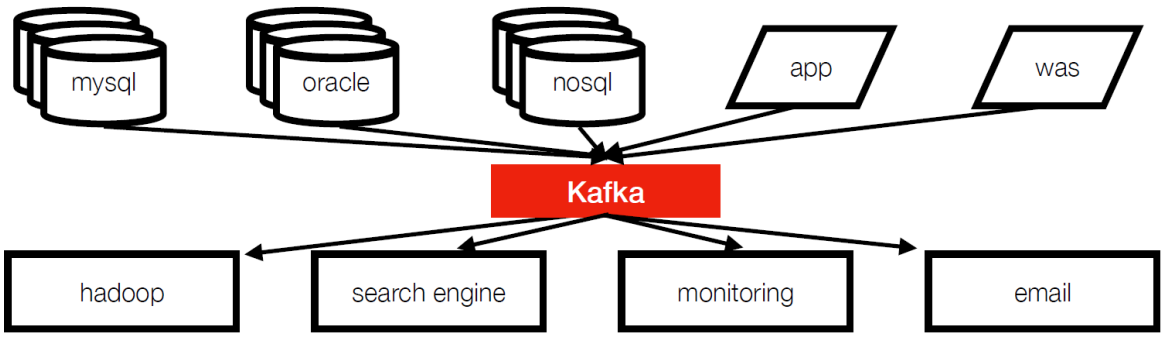
개발 배경

Before Kafka



- Point to Point 연결 방식의 아키텍쳐
- 데이터 연동의 복잡성 증가
- 확장에 어려움이 있음

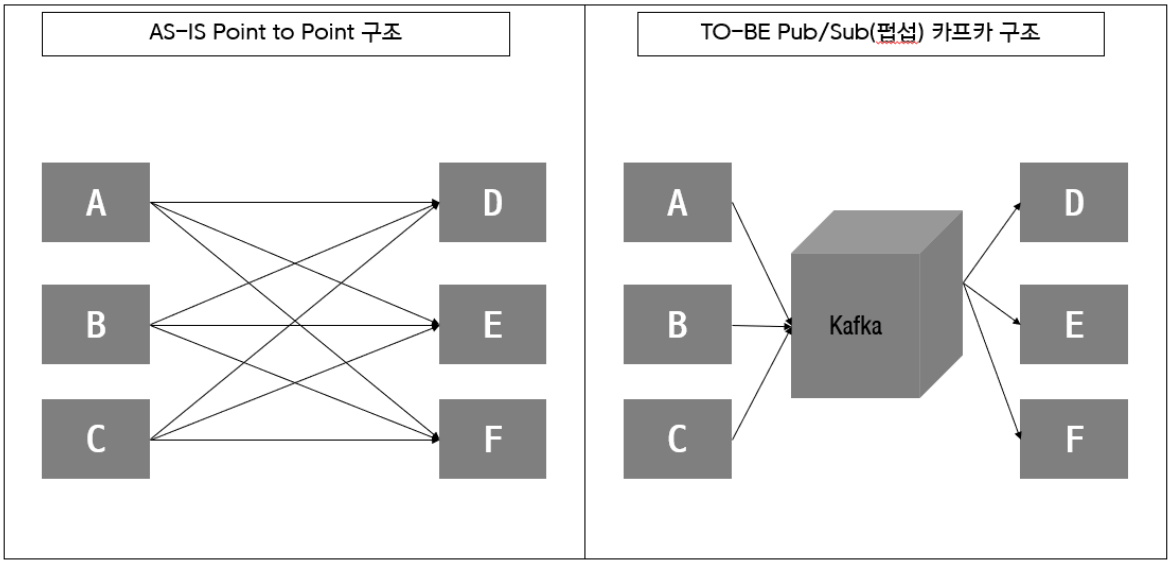
After Kafka



- 생산자(Producer) 와 소비자(Consumer) 로 나눔
- 메세지 데이터를 여러 컨슈머에게 허용
- 확장에 용이
- 노드(Broker) 추가로 처리량을 높일 수 있음

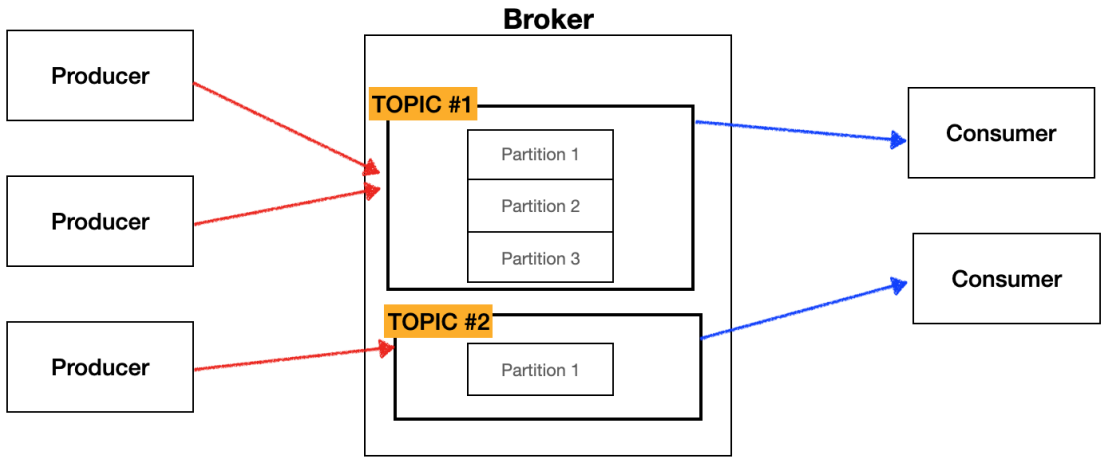
카프카 특징

Pub / Sub 방식



- source Application과 Target Application 갯수가 많아질 수록 data를 전송하는 라인도 복잡해짐
- source Application과 Target Application 갯수가 적으면 data를 전송 하는 라인도 단순
- 발신자(Publish)는 수신자가 누구든 상관없이 메세지를 보내기만 하고 수신자(Subscribe)는 발신자가 누군지 상관없이 필요한 메세지를 수신함.
- 카프카에서는 publish를 프로듀서(producer)라 하고, Subscribe를 컨슈머(Consumer)라 한다.

멀티 프로듀서, 멀티 컨슈머

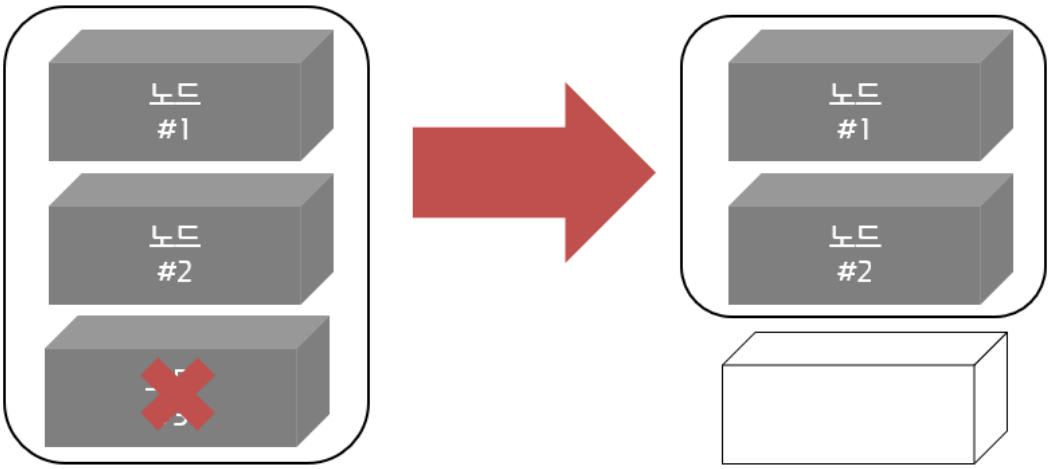


- Topic에 여러 Producer와 Consumer들이 접근 하여 데이터를 주고받을 수 있음.
- 카프카 중앙 집중형 구조로 구성.

Disk 저장

- 메세지를 디스크에 저장하고 유지
- 보관 주기동안 메세지가 삭제되지 않기때문에 컨슈머가 메세지 손실없이 가져갈 수 있음

고가용성



- 장애 대응에 강함.
- 고가용성으로 서버에 갑자기 이슈가 생겼을 때, 데이터 손실없이 복구 할 수 있음.

확장성 및 높은 처리량(Throuput)

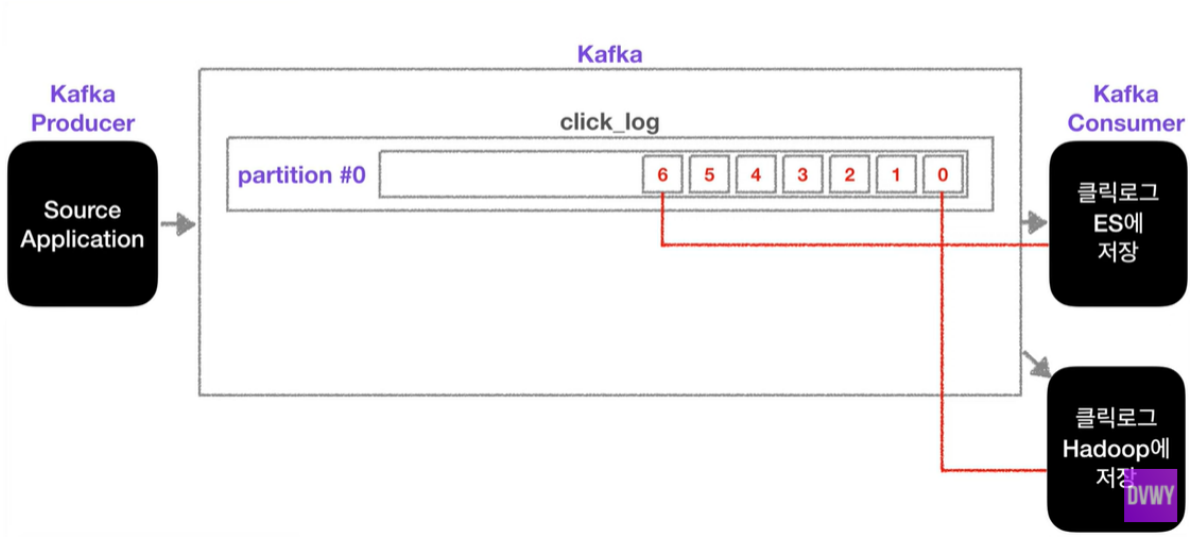
- 필요한 경우 노드(Broker)를 추가하여 처리량을 높일 수 있음.
- 서비스 이용이 증가하여 TPS가 늘어나도 노드를 추가하여 리스크를 감소 할 수 있음
- 서비스 중단없이 온라인 상태에서 작업 가능

Topic 과 Partition

Topic 이란

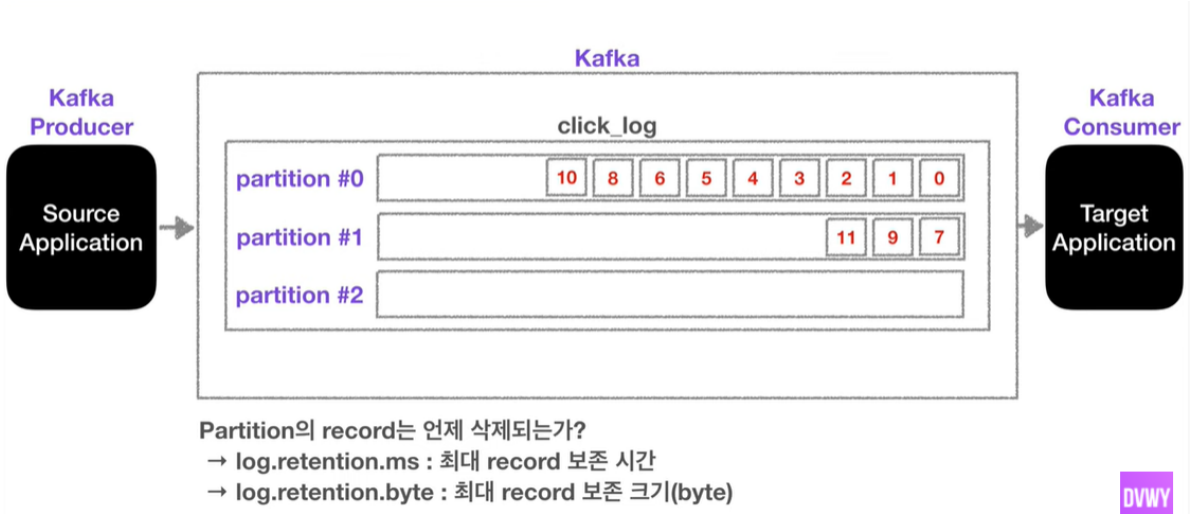
- 카프카에서 데이터를 주고 받는 공간.
- 토픽은 목적에 따라 이름을 갖을 수 있음. 추후 유지보수에 용이. ex) click_log, send_sms, location_log

Partition 이란



- 저장소(Topic) 에서 분리되어진 공간.
- 파티션이 많을 수록 Consumer에게 데이터를 빨리 전달 할 수 있음.

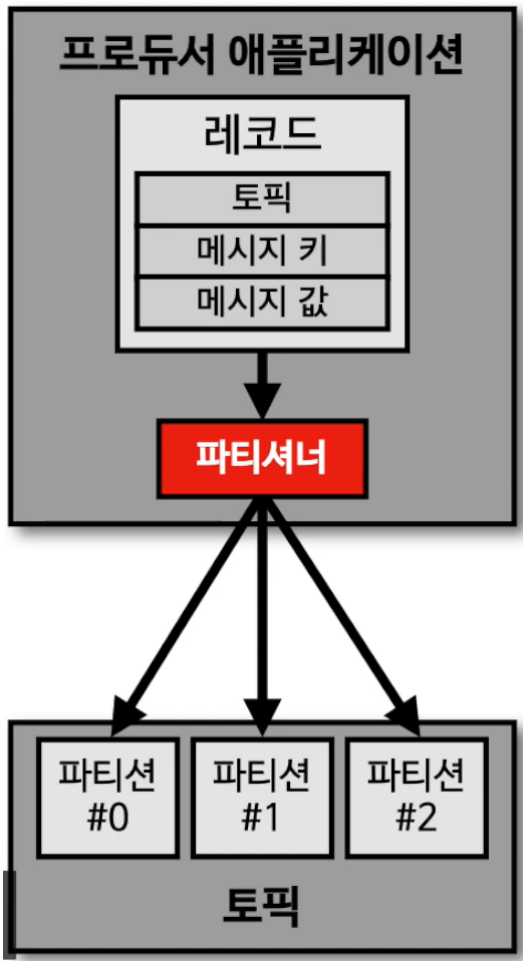
Topic 내부



- 하나의 토픽은 여러개의 파티션으로 구성.
- 카프카 Consumer는 가장 오래된 데이터부터 가져감. 데이터를 가져가도 파티션 내부의 데이터는 삭제되지 않음.
- 새로운 컨슈머가 붙었을 경우 파티션의 0번 데이터 부터 다시 가져감.
    - 컨슈머 그룹이 달라야 함.
- 파티션이 추가된 경우
    - 데이터를 보낼 때 key를 지정하여 어느 파티션에 넣을지 정할 수 있음.
    - key를 지정하지 않는 경우, round-robin으로 파티션이 지정됨.

    - key를 지정하는 경우, key값의 해시값을 구하고 특정 파티션에 할당되게 됨.
- 파티션을 늘리는건 가능하지만 줄일 수 없기때문에 신중해야함.
- 파티션 갯수를 늘리면 컨슈머 갯수를 늘려 데이터 처리를 분산시킬 수 있음.
- 옵션으로 지정한 최대 시간과 용량에 따라 파티션에 적재되어있는 데이터의 삭제 시기를 정할 수 있음.

Partitional 이란



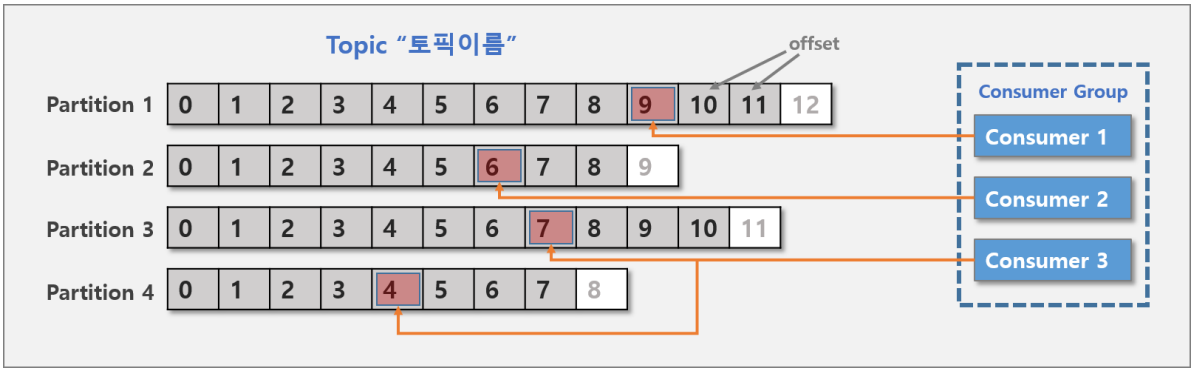
- 프로듀서가 데이터를 보낼 때 무조건 파티셔너를 통해서 전송하게 됨. 파티셔너는 데이터를 토픽의 어떤 파티션에 넣을지 정하는 역할을 함.
- 레코드에 포함된 메시지 키 또는 값에 따라 파티션의 위치가 달라짐.
    - 동일한 메시지 키를 가진 레코드는 동일한 해쉬값을 만들어내기때문에 항상 동일한 파티션으로 들어갈 수 있음.
        - 순서를 지켜서 데이터를 처리하게 할 수 있는 장점이 있음. ex) 서울의 온도를 측정한 데이터
    - 메시지 키가 없는 데이터들은 round-robin 방식으로 적절하게 분배됨.
- Partitional 인터페이스를 사용하여 커스텀 파티셔너를 만들 수 있음.
    - 메세지키, 메세지값, 또는 토픽 이름에 따라 어느 파티션에 데이터를 보낼지 정할 수 있음.
    - 특정 데이터의 처리량을 조절하고 싶은 경우 사용 ex) vip 고객의 데이터

Consumer와 Consumer Group

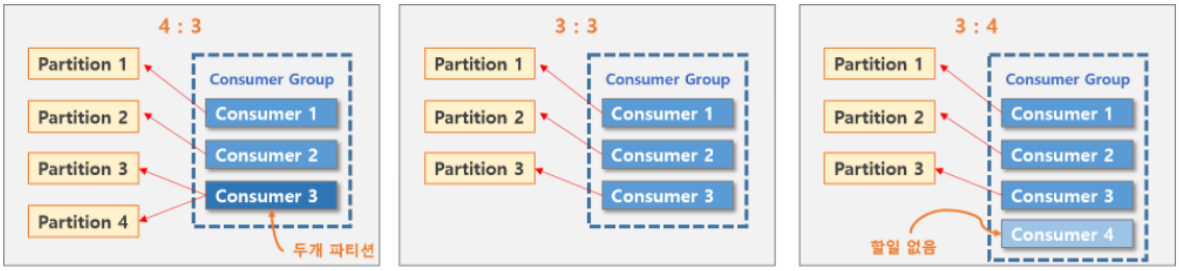
Consumer 란

- 데이터를 소비하는 주체.
- 데이터를 보낸 주체(Producer) 와 상호작용 없이 필요한 데이터를 소비.

Consumer Group 이란



- 하나의 Topic을 수신하기 위한 Consumer들의 그룹.
- Topic 과 Consumer 그룹은 1:N 관계로 매핑.



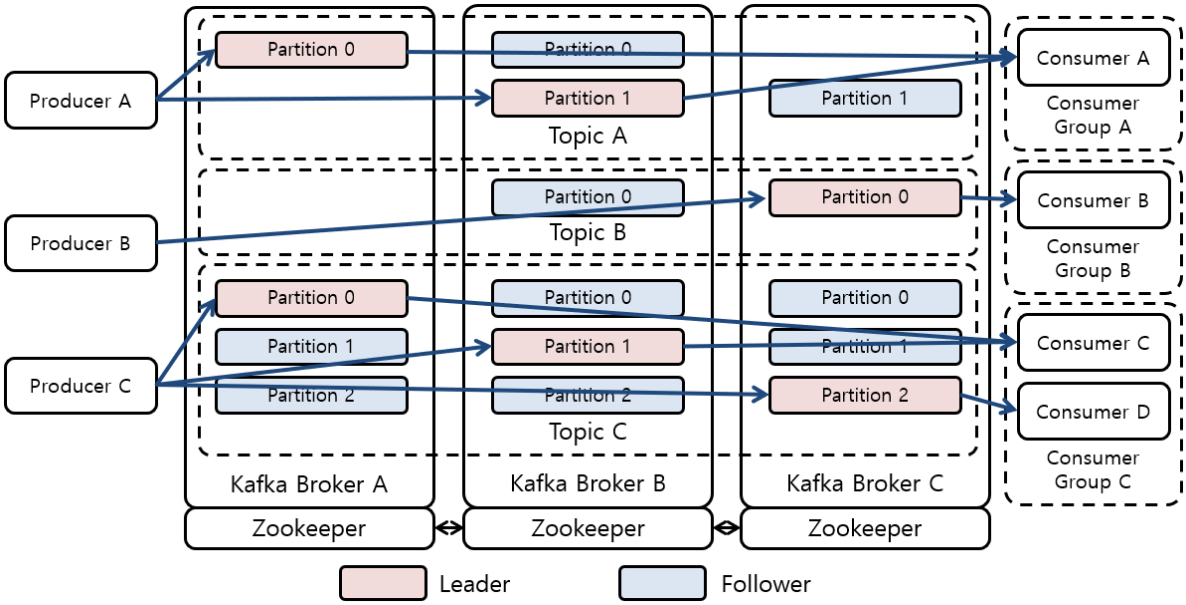
- Partition이 4개이고 Consumer가 3개인 경우
  - Consumer 1,2,3 중 하나가 두개의 Partition에서 데이터를 가져옴
- Partition이 3개이고 Consumer가 3개인 경우
  - 각각 순차적으로 데이터를 읽게됨
- Partition이 3개이고 Consumer가 4개인 경우
  - Consumer4는 할일이 없어서 놀게됨
- 일반적으로 Partition 갯수와 Consumer 갯수를 동일하게 구성

카프카 핵심요소 3가지

Broker

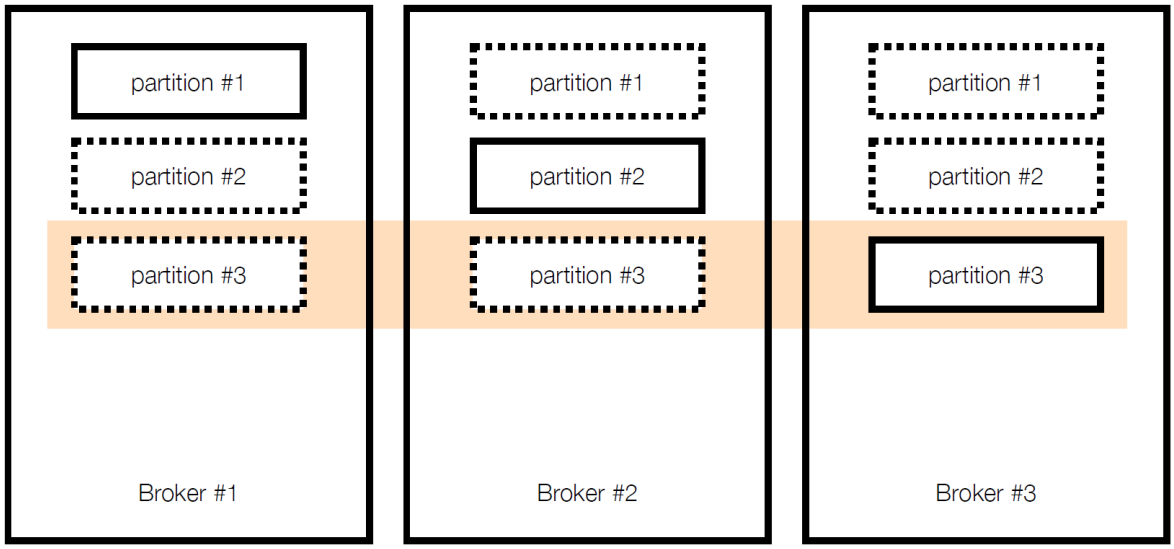
- 카프카가 설치되어있는 서버 단위
- 보통 3개이상 브로커를 설치하여 사용하는 것이 좋음 (장애 허용 관점)

Replication



- 다른 Broker로 Partition을 복제
- 원본 partiton을 Leader partition, 복제본 partition을 Follower partition이라고 함.
- 모든 Read/Write는 Leader를 통해서만 일어남!!
- partition의 고가용성을 위해 사용됨.
    - 원본 partition(Leader partition)이 훼손되더라도 Follower partition이 존재하므로 복제본으로 복구가 가능함. 그리고 Leader partition의 역할을 승계함
    - producer가 topic의 partition에 데이터를 전달 할 때, 데이터를 전달받는 주체가 Leader partition임
- producer에는 ack라는 상세 옵션이 있음.
    - ack = 0 일경우 : Leader partition에 데이터를 전달하고 응답값은 받지않음
        - 데이터가 잘 전송되었는지, 나머지 Follower partition에도 잘 전송되었는지 보장할 수 없음.
        - 속도는 빠르지만 데이터 유실 가능성이 있음!
    - ack = 1 일 경우 : Leader partition에 데이터를 전달하고 응답값을 받음
        - Leader partition에 데이터 전달 유무는 알 수 있지만, 나머지 Follower partition에 잘 전송되었는지 보장이 안됨.
        - 데이터 유실 가능성이 있음
    - ack = all : Leader partition에 데이터를 전달하고, 나머지 Follower partition에도 데이터가 잘 전송되었는지 확인 절차를 거침
        - 데이터 유실의 가능성은 없음.
        - 확인절차가 많기때문에 속도가 현저히 느림.
- replication을 무턱대고 만들 수 없는 이유
    - replication이 많아질수록 브로커의 리소스 사용량이 많아짐.

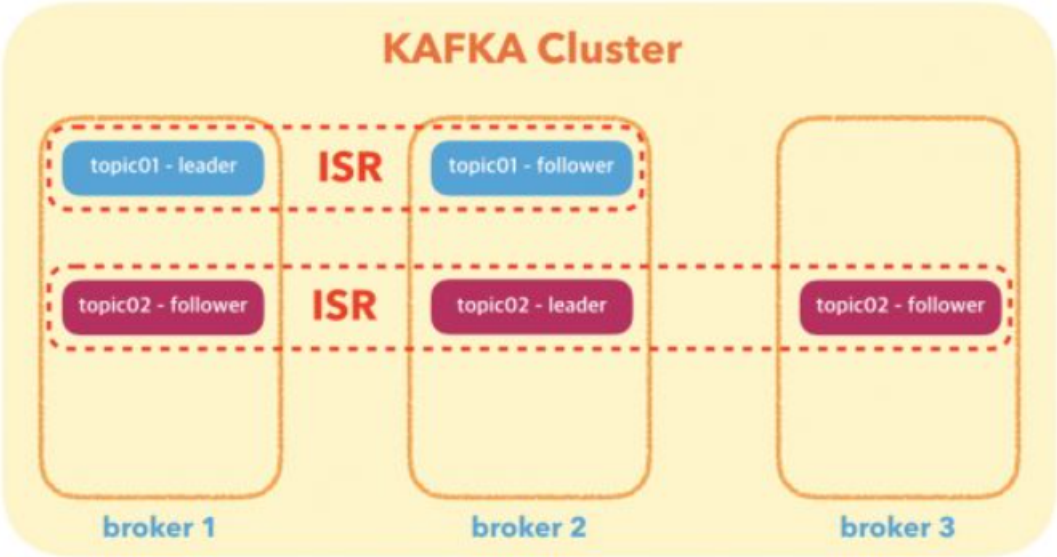
ISR(In Sync Replica)



- 다른 브로커와 공통된 Topic 안에 Partition과 Replication이 할당 된 상태.
- 특정 파티션의 리더, 팔로워가 각 브로커에 모두 복제되어 싱크가 맞는 상태를 의미
- ISR인 상태에서는 브로커 한개가 죽더라도 복제본이 존재하므로 복구가 가능.
- Leader Partition 의 브로커가 장애상태일 경우 Follower Partition이 Leader의 역할을 함

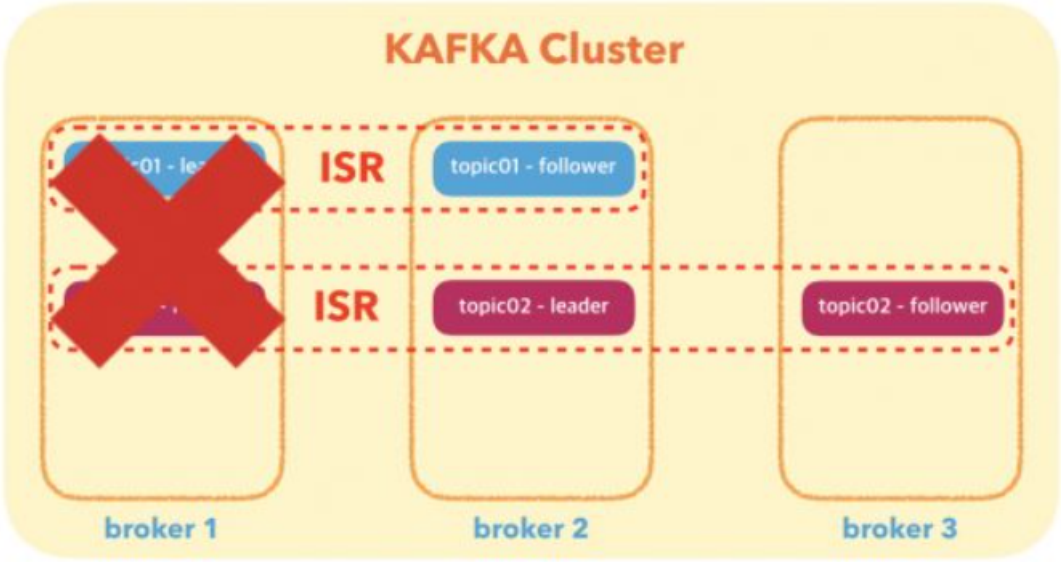
ISR상태에서의 장애 상황

모든 브로커가 정상



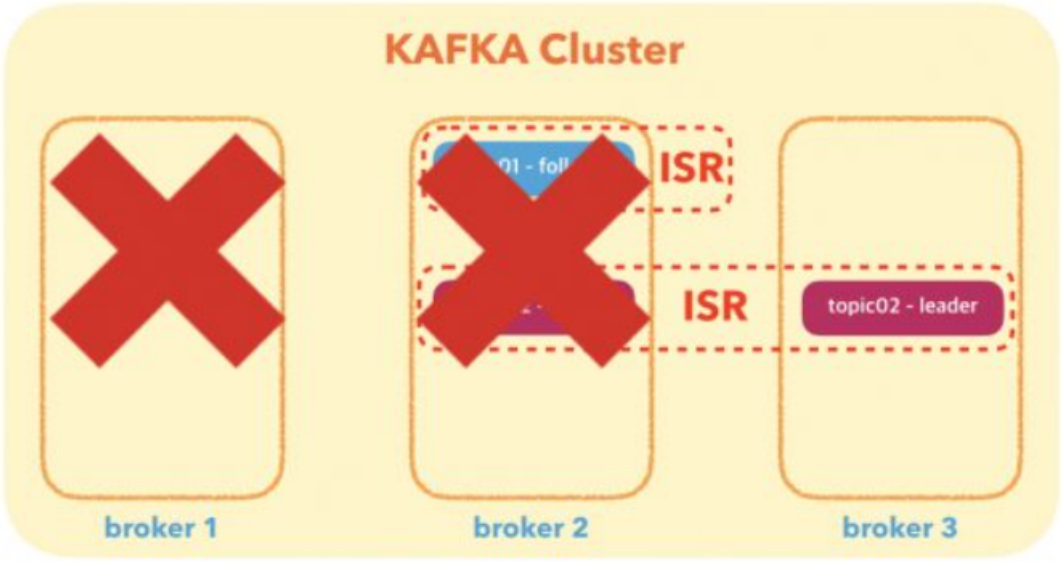
- Topic01은 브로커 1,2 에서 ISR 상태.
- Topic02는 브로커 1,2,3 에서 ISR 상태

브로커 1 장애 발생



- Topic01의 Follower partiton이 브로커 2에 있기 때문에 Leader partition으로 전환되어 정상작동
- Topic02의 Leader partiton이 브로커 2에 있었기 때문에 정상작동

브로커 1,2 장애 발생



- Topic01의 ISR내에 더이상 follower가 없기때문에 더이상 데이터를 받을 수 없음
- Topic02는 브로커3에 follower가 있었기 때문에 Leader로 전환되어 정상작동

장애가 난 브로커들이 정상적으로 작동하면 브로커들 안에 포함된 partition들이 follower가 되어 leader 파티션의 데이터를 가지고와 Sync를 맞춘다.

추후 다룰 내용

- Zookeeper
- 카프카 튜닝 방안
    - Throughtpu 최대화 방안
    - Latency 최소화 방안
- RabbitMq와의 차이점
- Data 저장 방식
- 실습