# 빅데이터 마스터과정
# (DAM)

하석재
CEO, 2HCUBE
sjha72@gmail.com

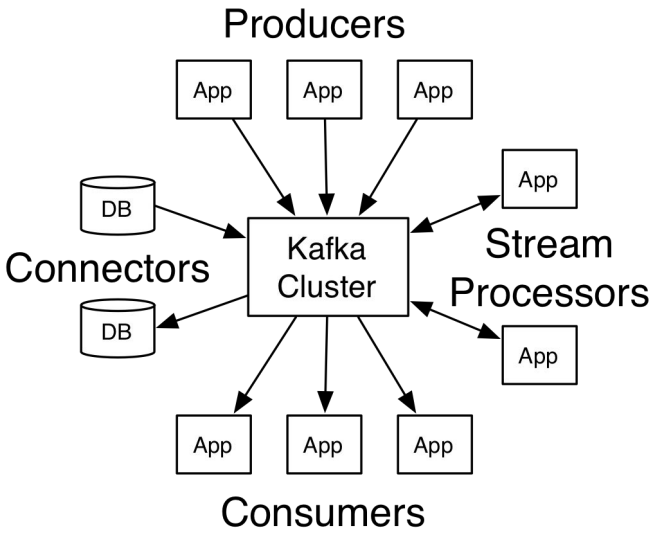
# Apache Kafka

# 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

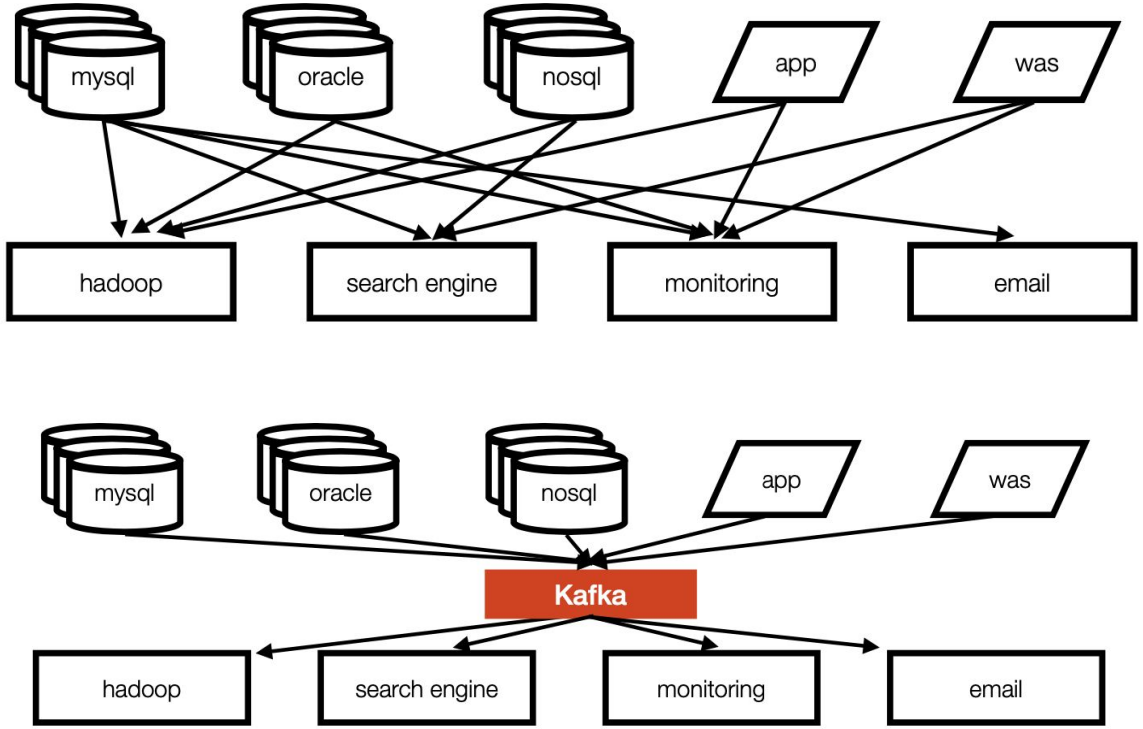
- 대규모로 발생하는 메시지성 데이터 **비동기**방식으로 중계
- 버퍼링하면서 안정적으로 중계
- 데이터를 **topic**에 저장하고 있다가 consumer가 데이터를 전송(소비)



# Apache Kafka

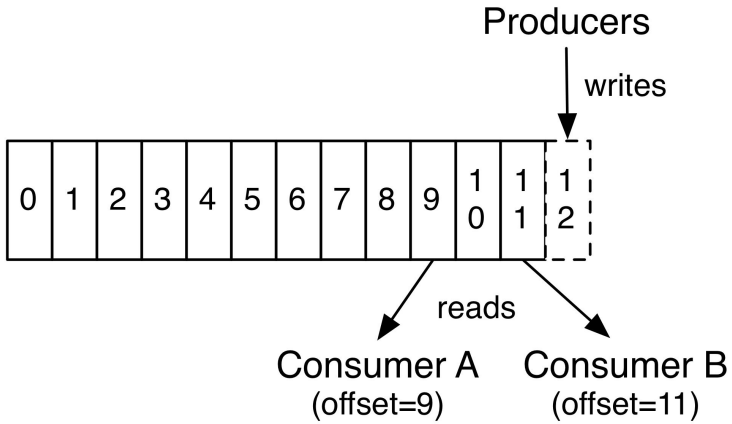
- 용도
    - Messaging System
    - Website Activity Checking/Monitoring:
    - Log Aggregation
    - Stream Processing / Batch Processing:
    - **Buffering**
    - Event sourcing(이벤트를 시간순으로 기록)

# 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

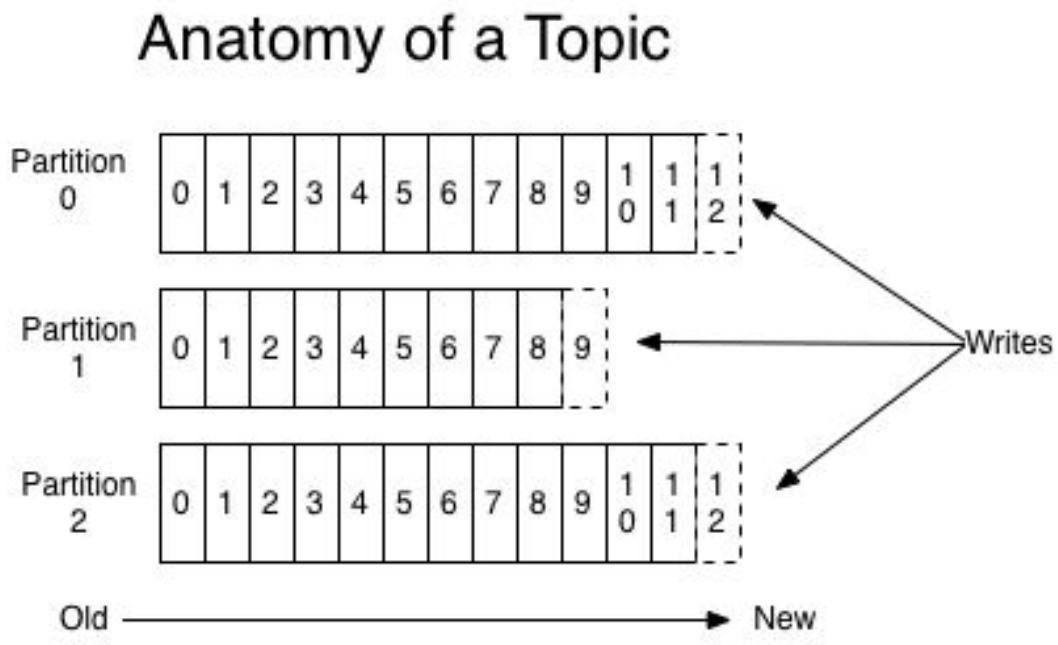
- Publisher/Subscriber 모델
- High Availability / Scalability
- Sequential Store and Process in Disk
  - 장애대응
  - I/O 최적화
- Distributed Processing

# Kafka 아키텍처

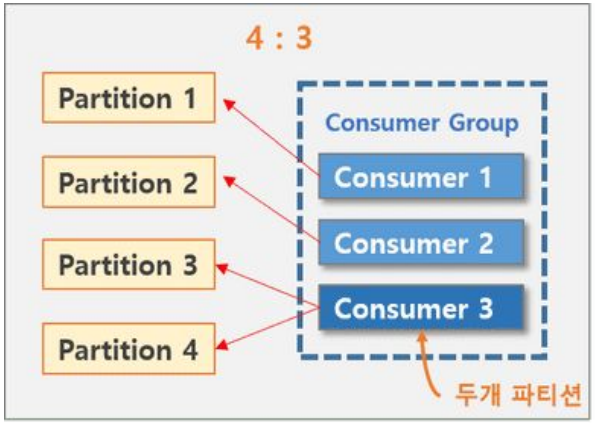
- Pub/Sub 구조
- 브로커(Broker)
- 주키퍼(Zookeeper)
- 토픽(Topic)
- 파티션(Partition)
- 리더(Leader)/팔로워(Follower)
- 컨슈머 그룹(Consumer Group)

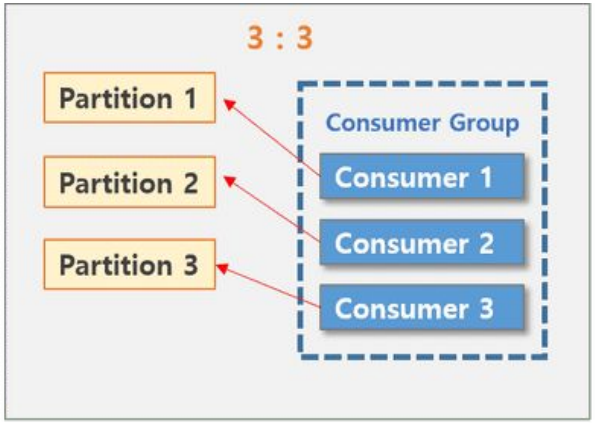
# Apache Kafka

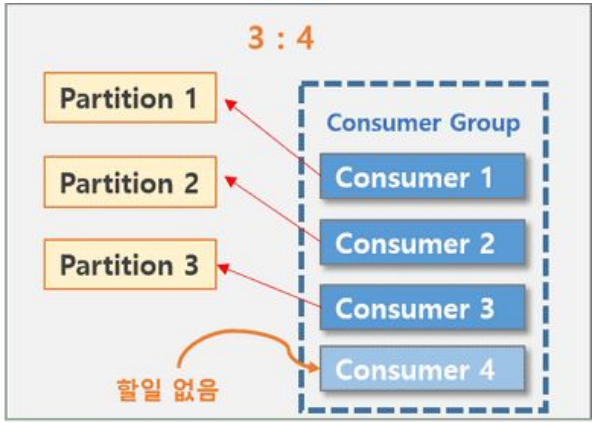
- **토픽(topic)**을 기준으로 메시지관리



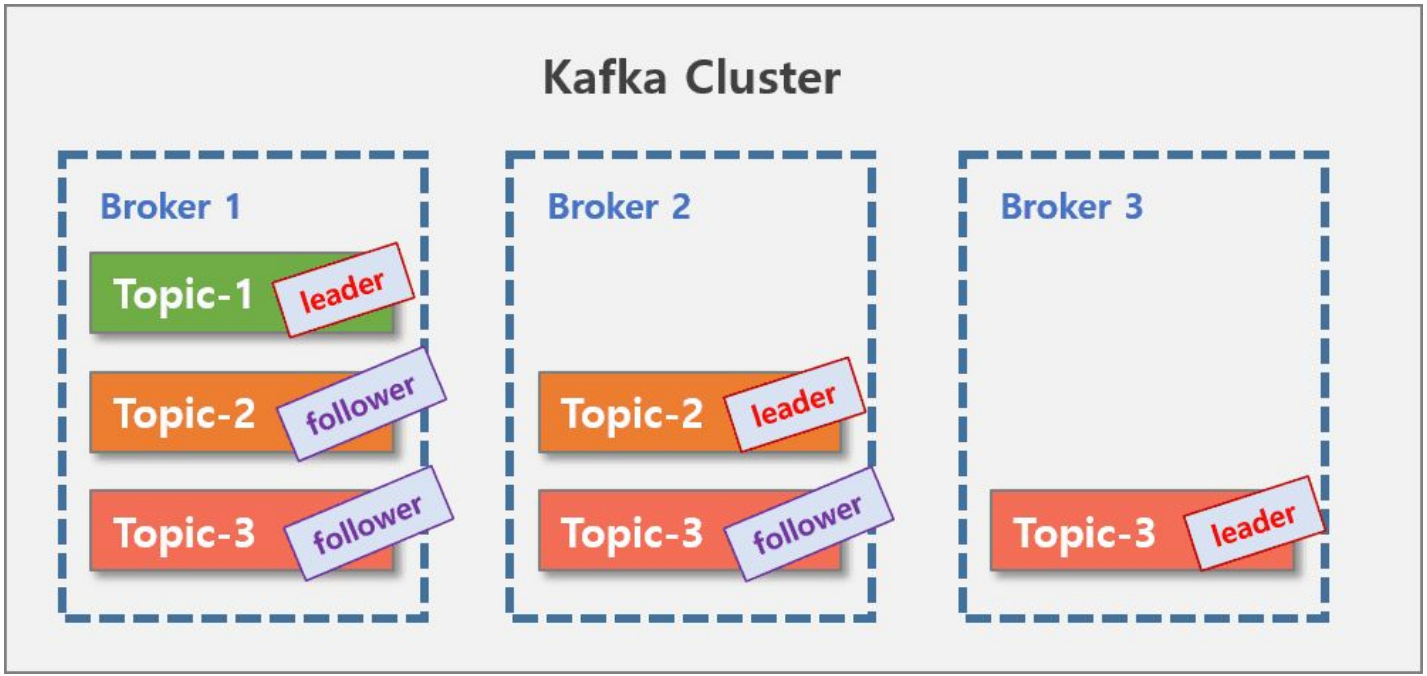
# 파티션과 컨슈머그룹


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

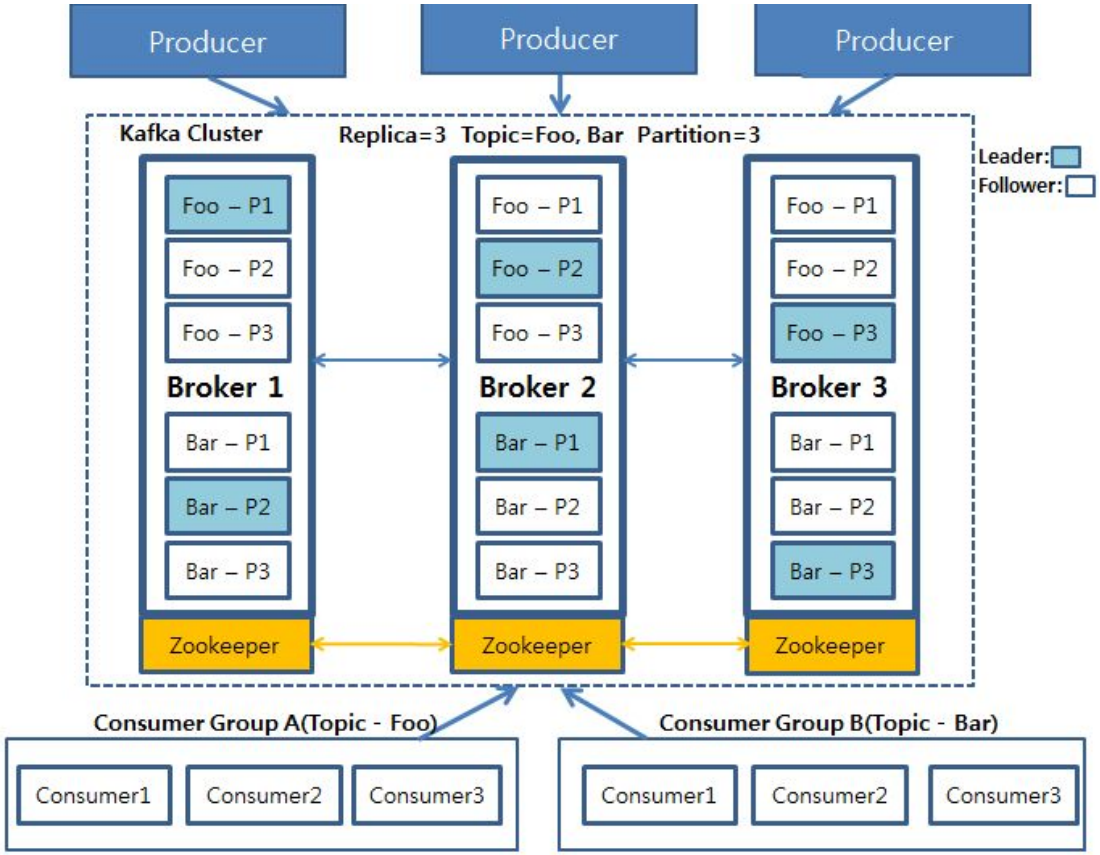
# Kafka leader/follower



# Kafka leader/follower

- ack
  - 0:  ack을 기다리지 않음, 빠름
  - 1:  leader는 데이터를 기록
  - all(-1) : 모든 ISR 확인, 느림/손실가능성 없슴

# Kafka Broker 구조

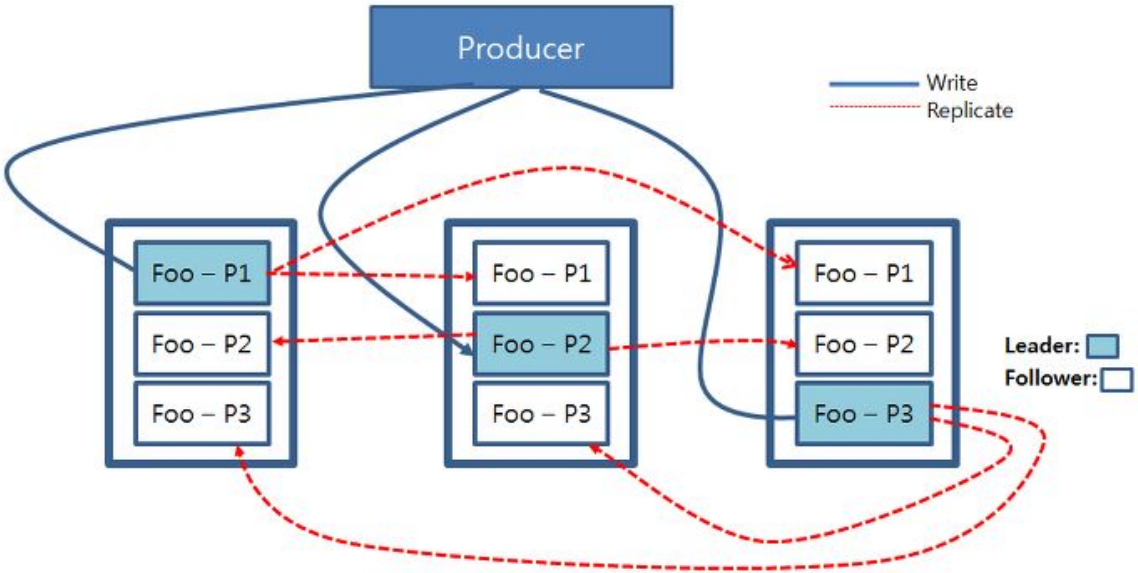


# Consumer Group

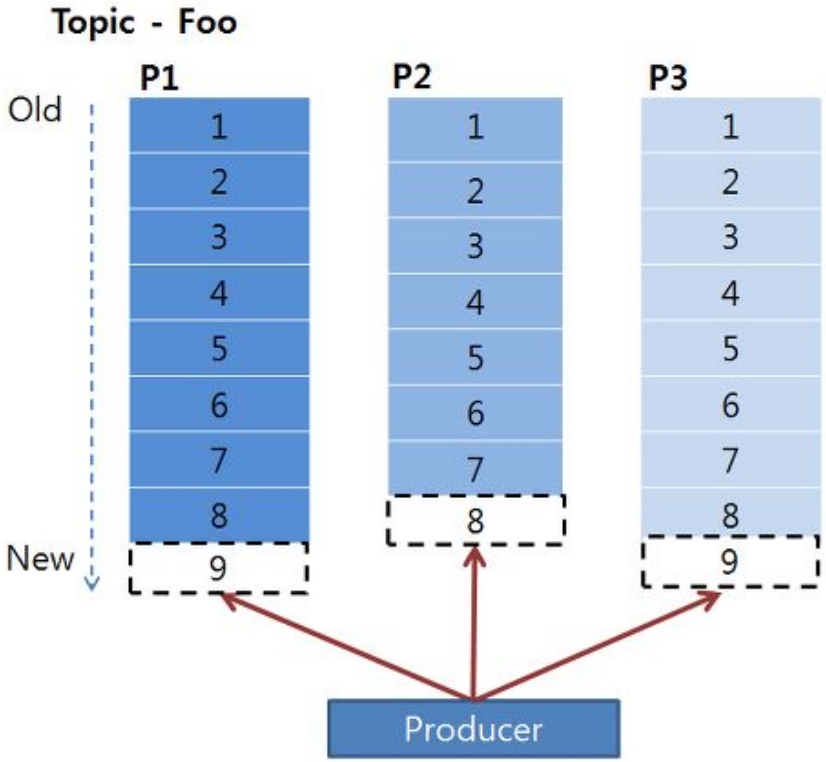
- Consumer들을 묶는 개념
- Consumer 수 만큼 파티션의 데이터 분산처리함(읽을 때의 단위)
  - 파티션이 3개면 3개의 Consumer가 필요함

- 복제본에는 Leader를 선정(하늘색), 읽기/쓰기를 관장하는 구조

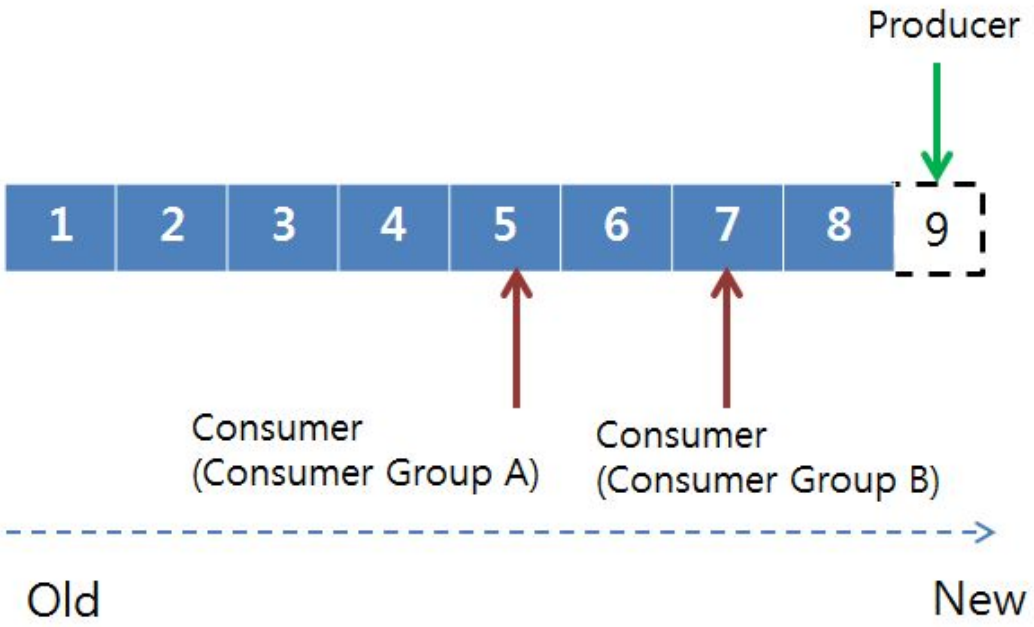
# Consumer Group



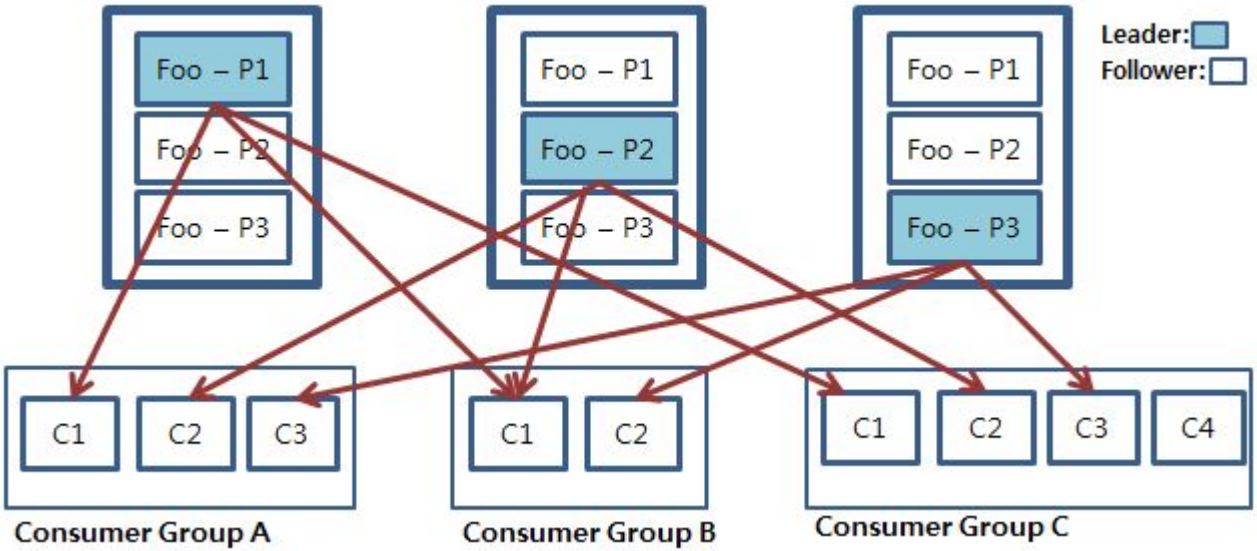
# Producer(파티션)

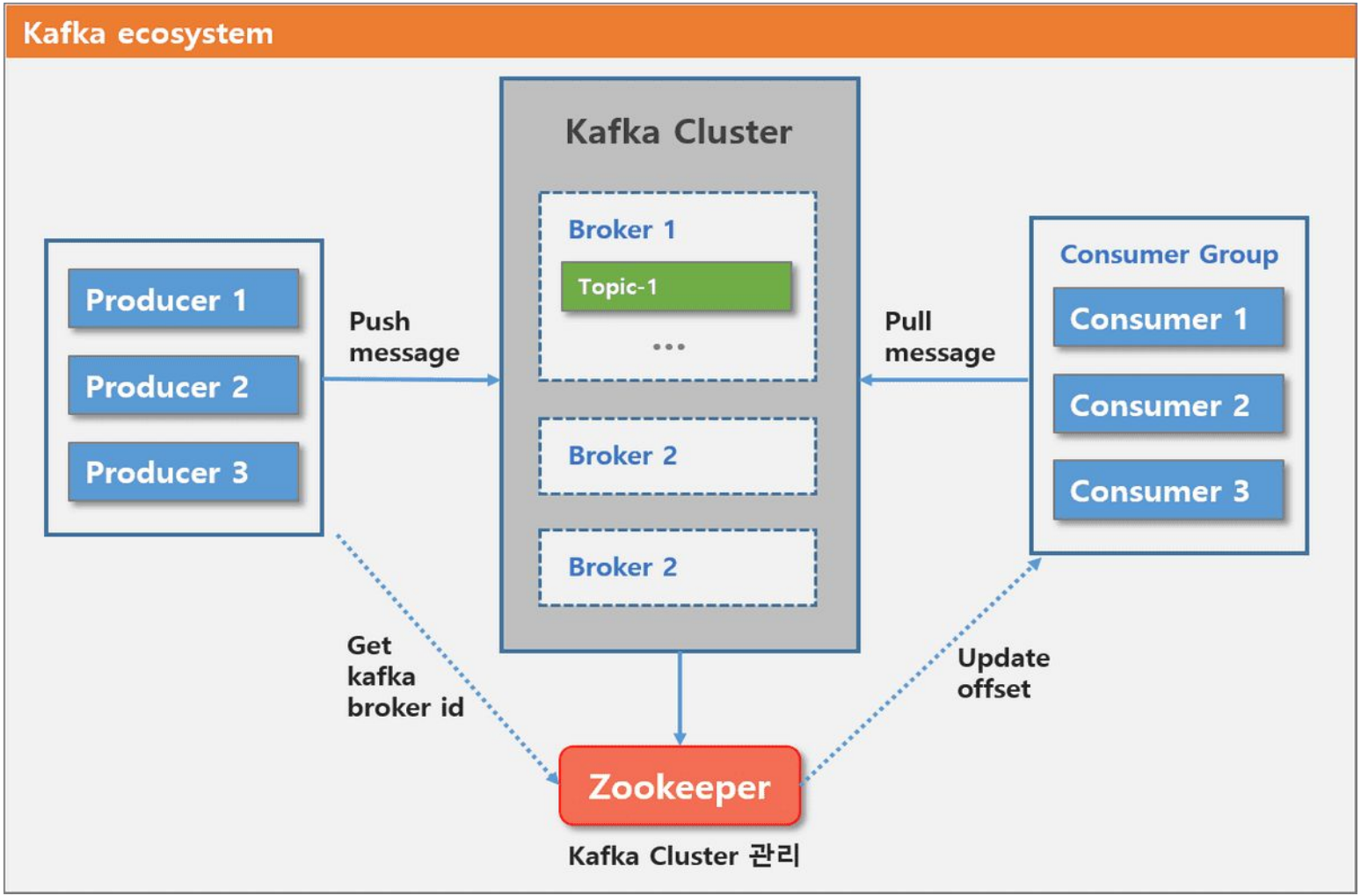


# Consumer


Producer
1
2
3
4
5
6
7
8
9
Consumer
(Consumer Group A)
Consumer
(Consumer Group B)
Old
New

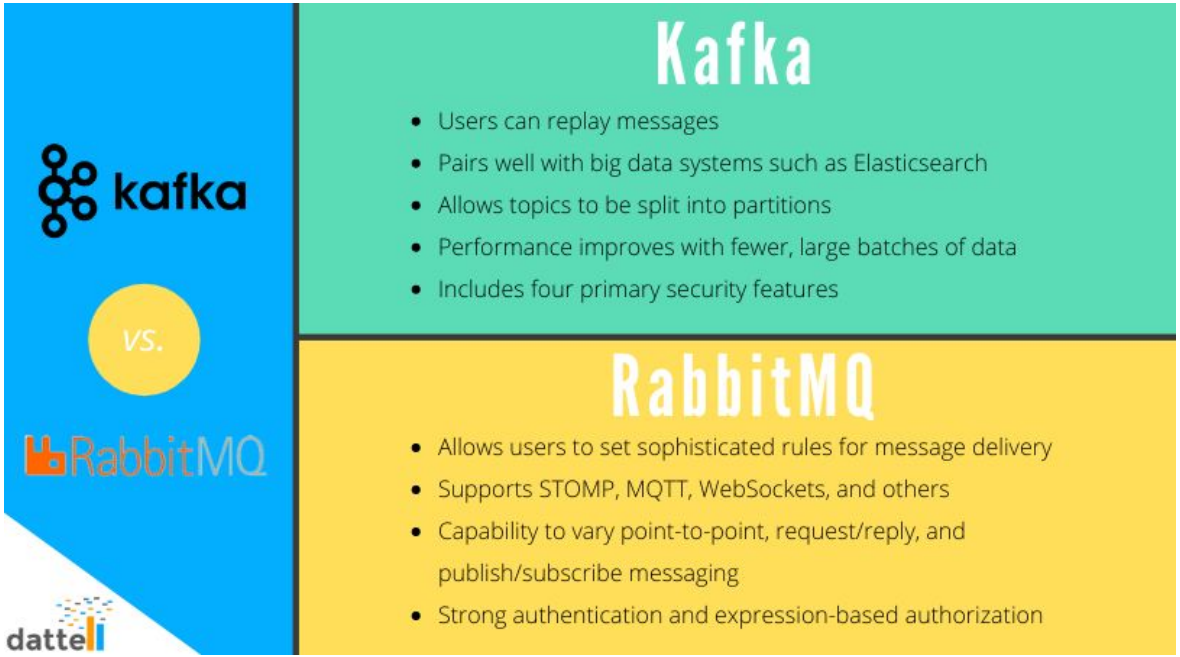
# 카프카 동작


Foo – P1
Foo – P2
Foo – P3
Foo – P1
Foo – P2
Foo – P3
Foo – P1
Foo – P2
Foo – P3
Leader:
Follower:
C1
C2
C3
C1
C2
C1
C2
C3
C4
Consumer Group A
Consumer Group B
Consumer Group C


Kafka ecosystem
Kafka Cluster
Broker 1
Topic-1
...
Broker 2
Broker 2
Producer 1
Producer 2
Producer 3
Push message
Pull message
Consumer Group
Consumer 1
Consumer 2
Consumer 3
Get kafka broker id
Update offset
Zookeeper
Kafka Cluster 관리

# 카프카 성능(vs RabbitMQ)

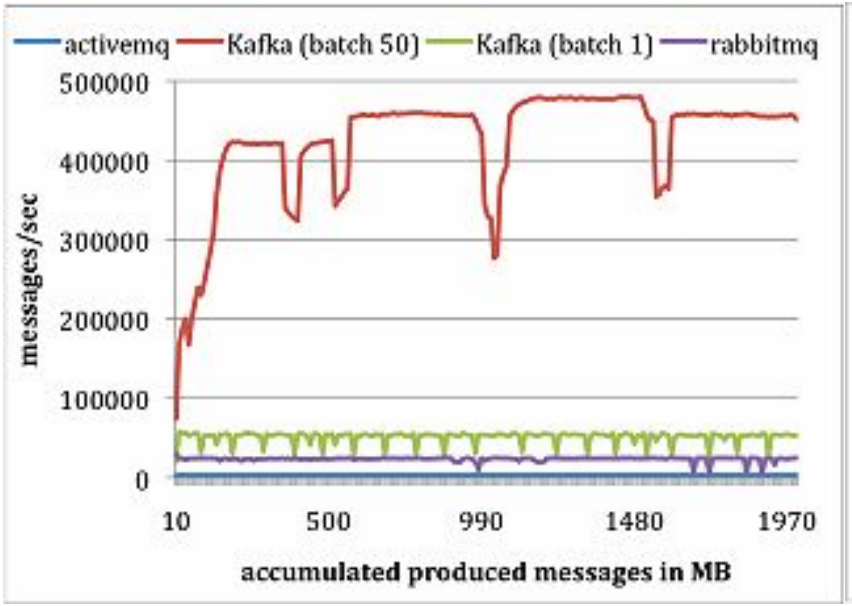
- https://dattell.com/data-architecture-blog/kafka-vs-rabbitmq-how-to-choose-an-open-source-message-broker/

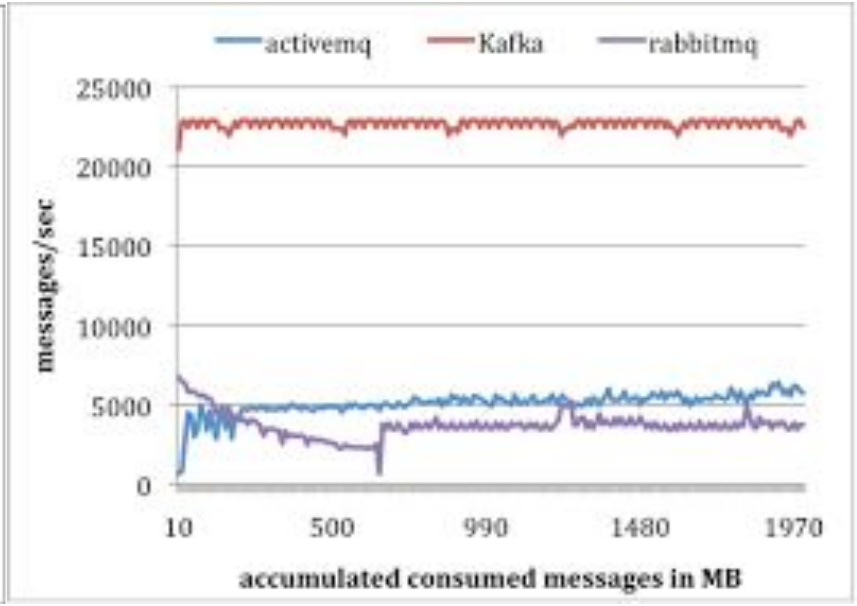


# 카프카가 빠른 이유

- 제로카피(Zero Copy)
  - 큐의 복사가 실제 일어나지 않고 주소의 복사만 일어나게 됨
  - 많은 수의 큐/버퍼를 관리해야 하는 상황에서 큰 성능차 발생

# 카프카 성능(vs RabbitMQ/ActiveMQ)



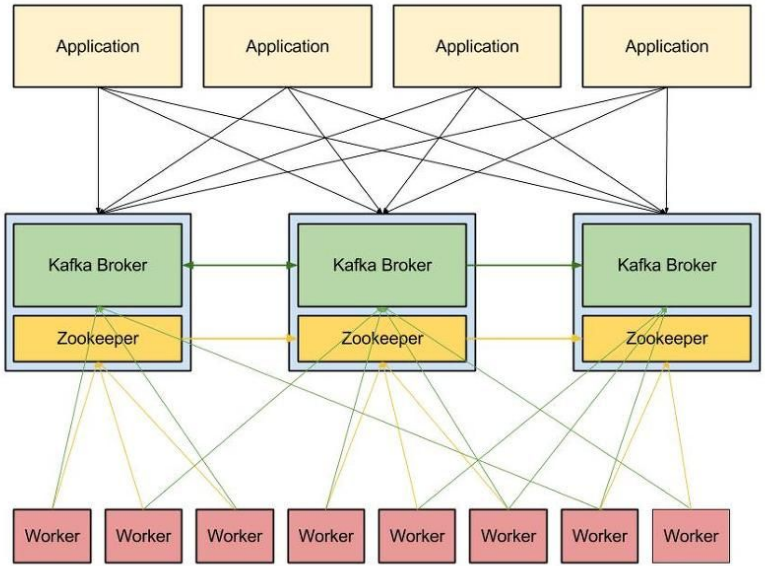
# 주의사항

- **파티션의 개수는 추가만 가능함**
  - 줄이는 것 불가능
  - 늘리는 것을 신중하게 결정할 필요
  - 파티션 추가하면 Consumer 추가해야 함

- 토픽의 내용은 지정된 기간만큼 보관
  - 데이터저장용량이 막대하게 필요
  - **기본 7일** -> 2~3일로 수정필요

# Apache Kafka

- Zookeeper와 같이 구성(**필수**)




http://blog.mmlac.com/log-transport-with-apache-kafka

# Real-time 이란

# 실시간(Real-time)...

- “찐”
- 바로 그 때(in-time)  cf. Just In-Time(JIT) / JIT 생산방식
- 스케줄러/큐잉시스템 변경
  - Round Robin -> Weighted RR(WRR) -> Dynamic WRR(DWRR)
- 큐잉시스템 변경
  - 우선순위 큐(Priority Queue)
    - 우선순위별로 별도의 큐 유지
  - 윈도우 0-31
  - 자바 0-9
  - 리눅스 100 이상

# 실시간(Real-time)...

- 우선순위 역전 현상
  - 낮은 순위에서 높은 우선순위보다 좋은 서비스를 받는 상황
- 방지방안
  - 보장 가능?
    - Soft Real-time / Hard Real-time
      - 스케줄러 변경 -> rate monotonic(RM:단조감소) 스케줄러
      - 리눅스 -> 실시간 리눅스
      - **Near real-time cf. Pseudo-Random Number**

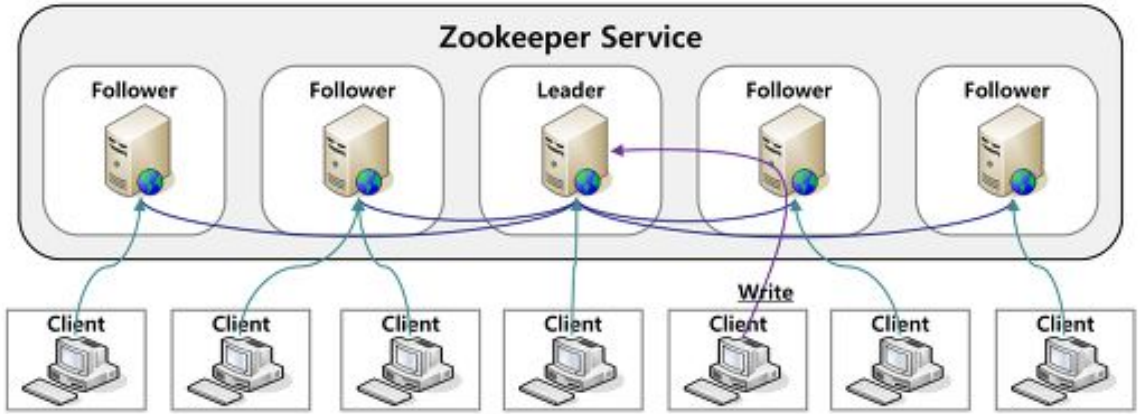
  - 고가용성(HA: 99.99999%)

# Apache Zookeeper

# Apache Zookeeper

- 고가용성(HA)을 제공하기 위한 분산 코디네이션용 오픈 소스 프로젝트
  - cf. DBMS복제(Replication), 네트워크 이중화/다중화
- SPOF(Single Point of Failure)가 있는 시스템의 단점을 보완하기 위해 제안(reliable distributed coordination)
- N개의 서버로 단일 주키퍼 클러스터를 제공
- 분산 시스템에서 리더를 선출(Election)

# Apache Zookeeper



# Apache Zookeeper

- Master / Slaves
  - 하나의 마스터와 여러 개의 슬레이브 구성
- **Active / Stand-by**
  - 장애가 발생하면 Fail-over
  - 전환에 시간이 걸리면 문제가 생길 가능성 있슴
- **Active / Active**
  - 동시에 여러 개의 Active를 구성
  - 어떤 상황에도 Active가 하나 이상 존재

# 카프카(도커) 설정

- 도커 설치

**$ sudo apt install docker.io**

- 도커컴포즈 설치

**$ sudo apt install docker-compose**

# 카프카설정(도커컴포즈)

**$ nano $HOME/docker-compose.yaml**

```
version: '2'
services:
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
  kafka:
    image: wurstmeister/kafka:2.12-2.5.0
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_ADVERTISED_HOST_NAME: 127.0.0.1
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
```

# 카프카설정(도커컴포즈)

- Kafka+Zookeeper 실행

**$ docker-compose -f $HOME/docker-compose.yaml up -d**

- 실행확인(kafka, zookeeper이 모두 up인지 확인)

**$ docker ps -a**

- 주키퍼(컨테이너) IP주소 확인

**$ docker inspect zookeeper | grep IPAddress**

- 카프카(컨테이너) 접속

**$ docker exec -it kafka bash**

**#**

# Kafka/Zookeeper 실습

- Topic 생성

**# kafka-topics.sh --create --zookeeper 172.20.0.2:2181 --replication-factor 1 --partitions 1 --topic test-Topic**

- Topic 리스트 확인

**# kafka-topics.sh --list --bootstrap-server localhost:9092**

- Topic 삭제

**# kafka-topics.sh --delete --zookeeper 172.20.0.2:2181 --topic test-Topic**

# Kafka/Zookeeper 실습

- Producer 사용(테스트)

```
# kafka-console-producer.sh --broker-list localhost:9092 -topic test-Topic
> Hello
> World
```

# Kafka/Zookeeper 실습

- Consumer 1,2,3는 개별 putty 생성 후 접속

**$ sudo docker exec -it kafka bash**

- Consumer 1

**# kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-Topic --from-beginning**

Hello
World

- Consumer 2

**# kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-Topic --from-beginning**

Hello
World

# Kafka/Zookeeper 실습

- Consumer3 (실시간모드)
  **# kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic test-Topic**

# Kafka/Zookeeper 실습

- Producer 사용(테스트)

```
# kafka-console-producer.sh --broker-list localhost:9092 -topic test-Topic
> Hello
> World

> a
> b
> c
```

# 감사합니다