

Kafka 기본

Cloud Platform본부/Modern Tech.팀
박남수

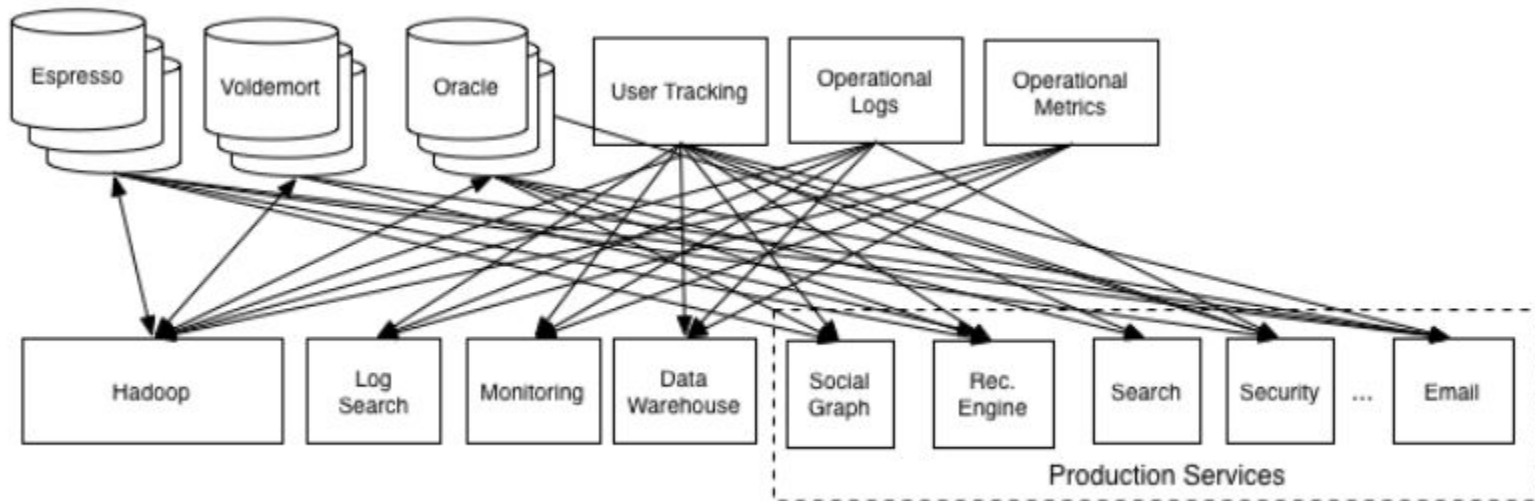
목차

- 아키텍처
- 주요기능
 - 레코드
 - 토픽, 파티션
 - 브로커
 - 주키퍼
 - 클러스터
 - 프로듀서
 - 컨슈머, 컨슈머그룹
- MirrorMaker 2
- 관리 도구
- 실습



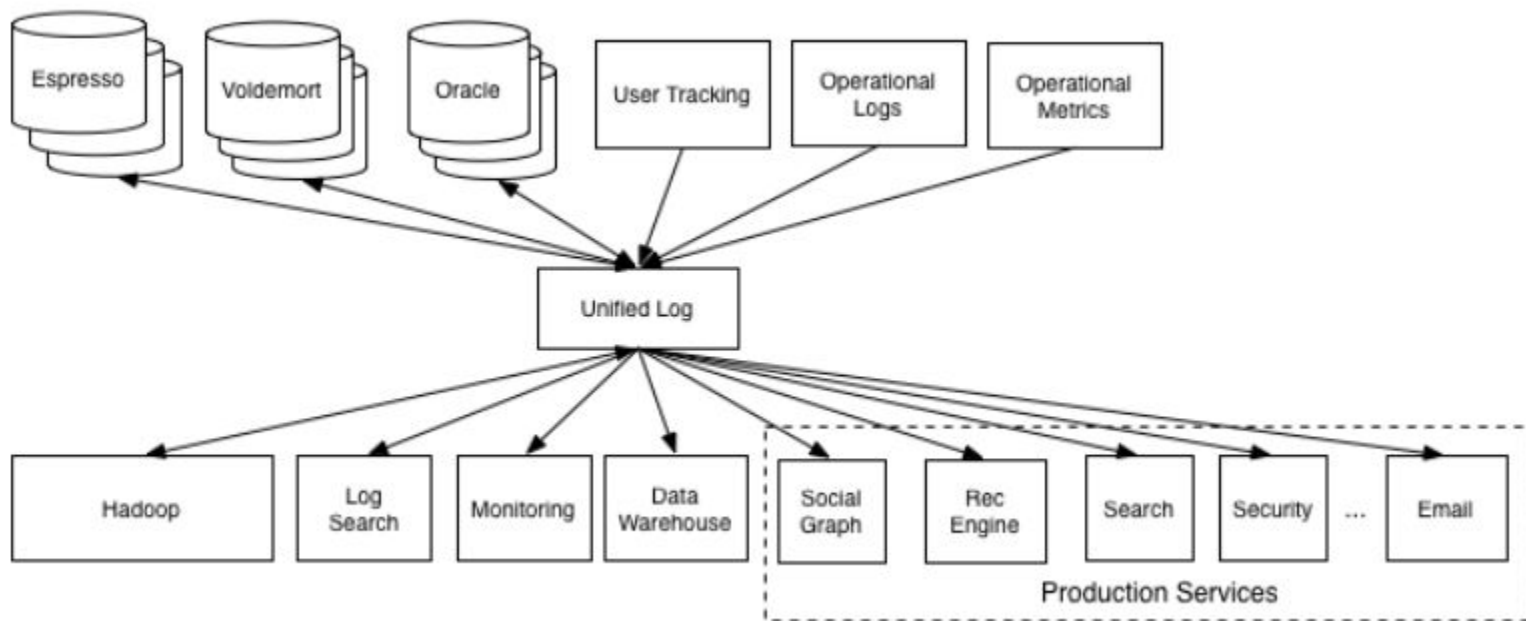
Kafka 소개

Kafka 도입 이전

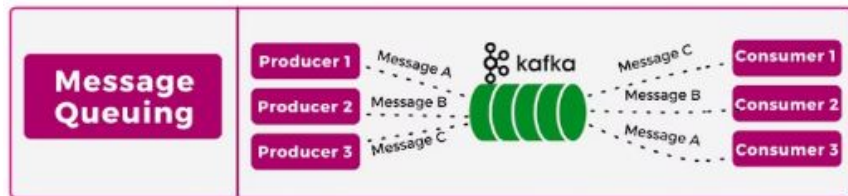
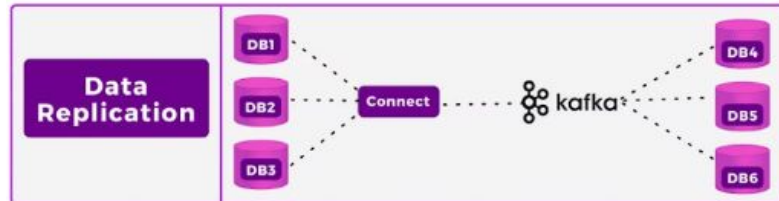
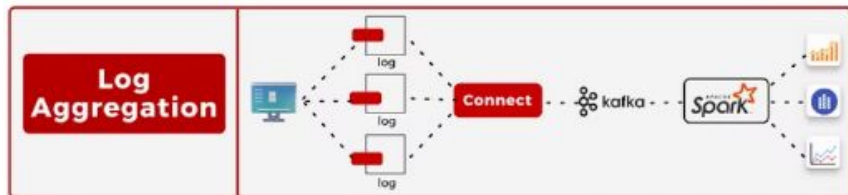
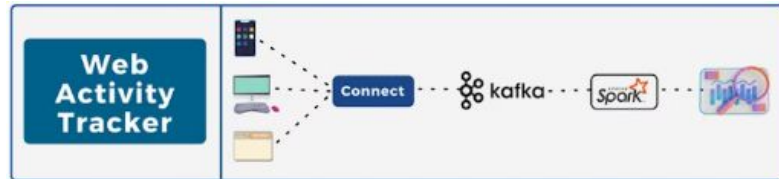
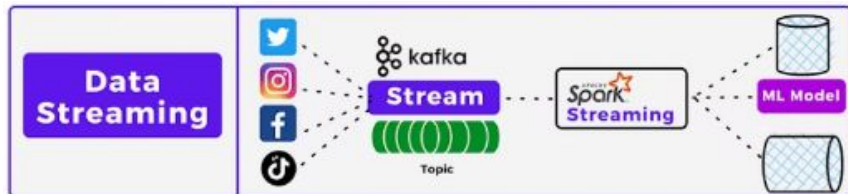


Kafka 소개

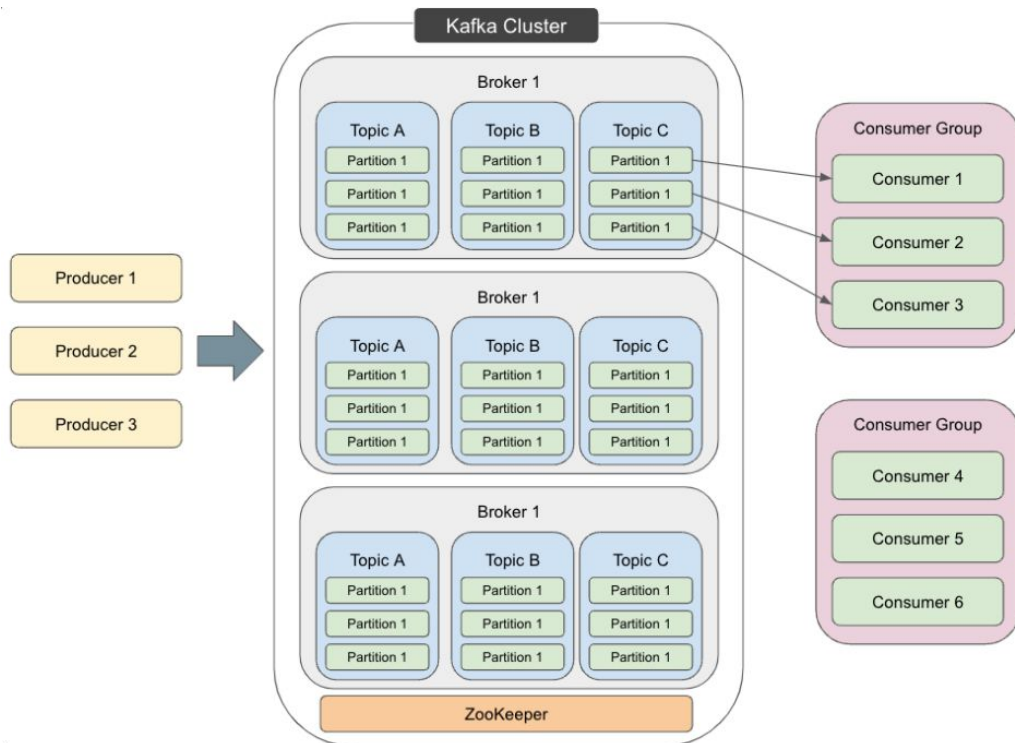
Kafka 도입 이후



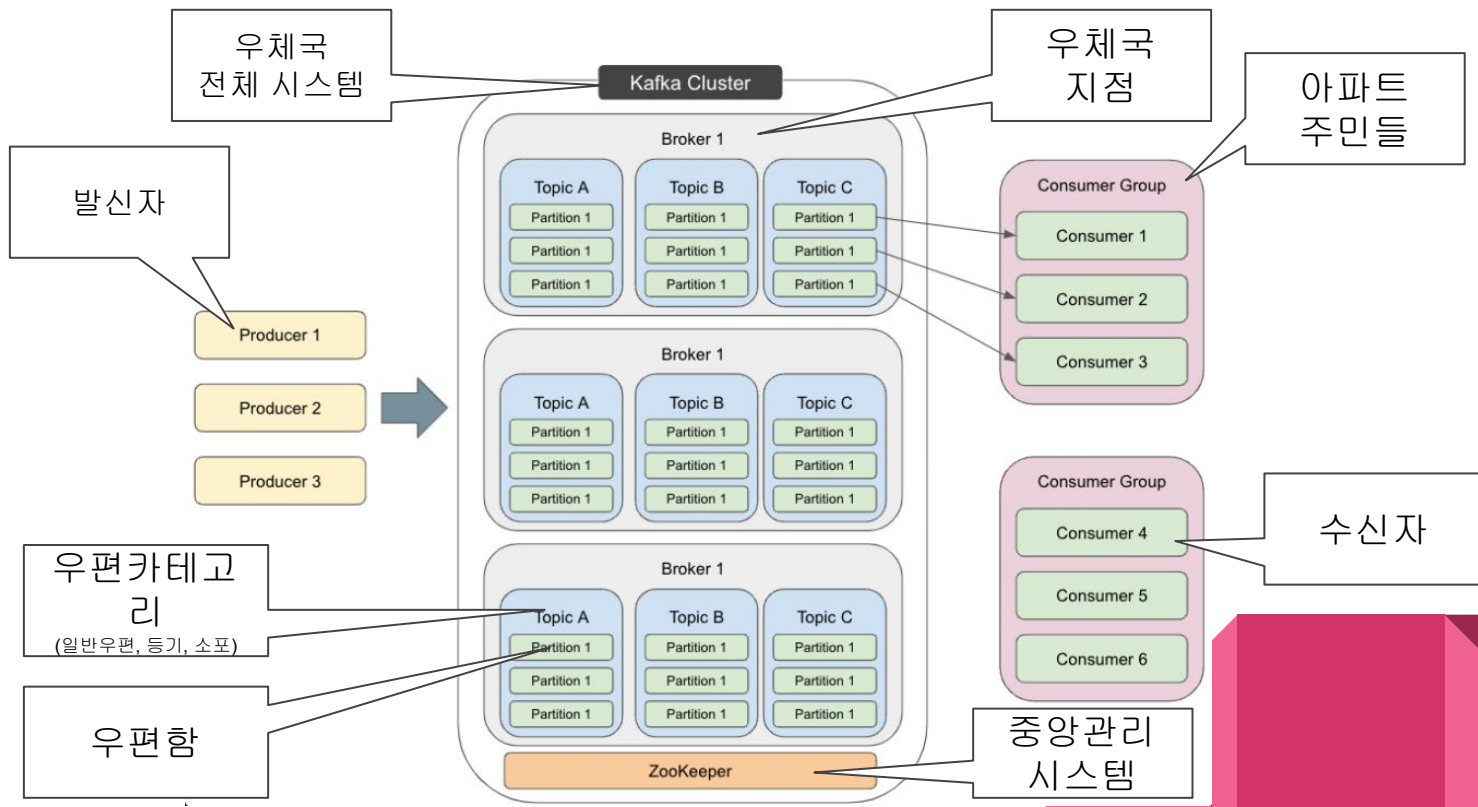
USE CASE Top 5



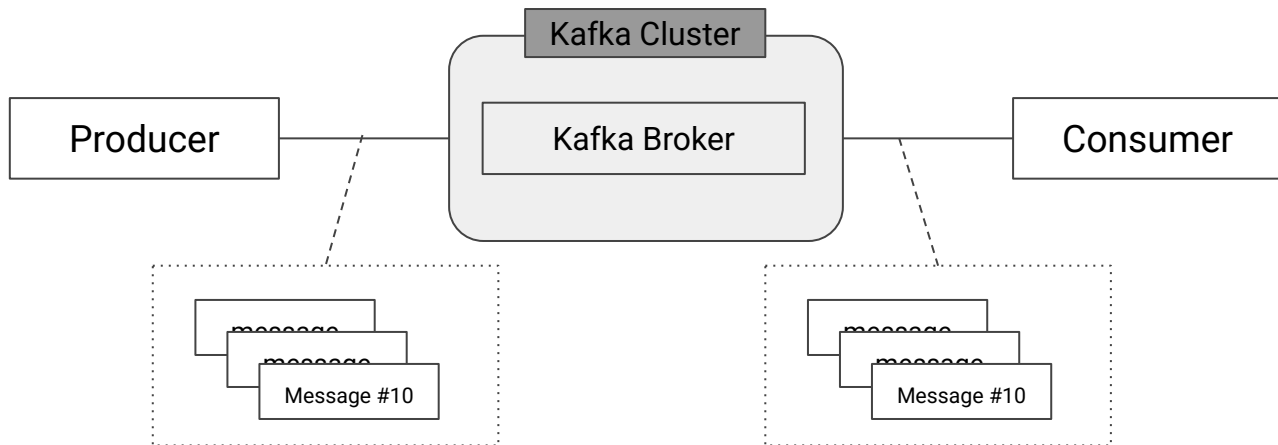
Kafka 아키텍처



Kafka 아키텍처



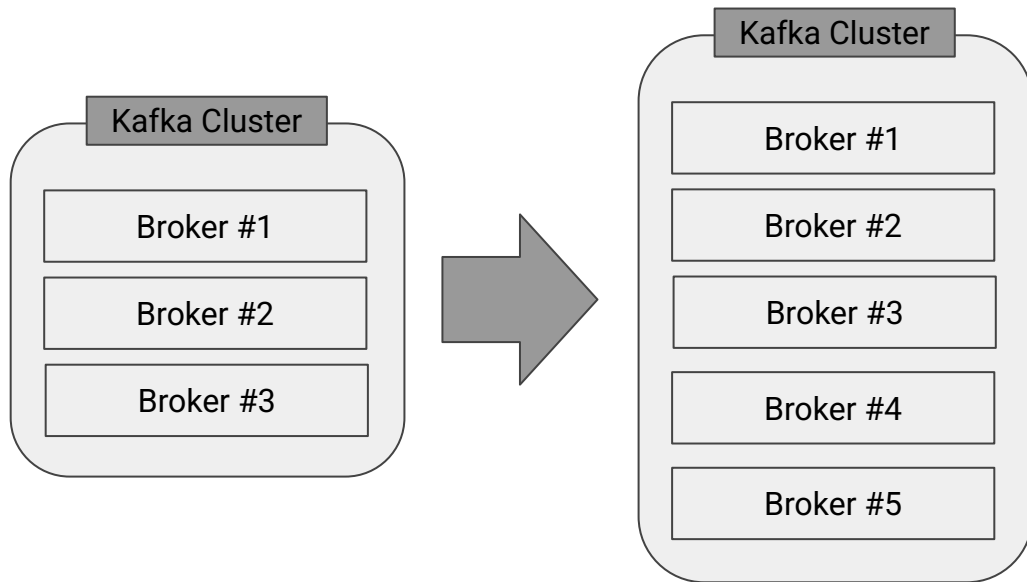
Kafka 특징



❖ 높은 처리량

- 많은 양의 데이터를 묶음 단위로 처리하는 배치로 빠르게 처리
- 동일 목적의 데이터를 여러 파티션에 분배하고 데이터를 병렬처리

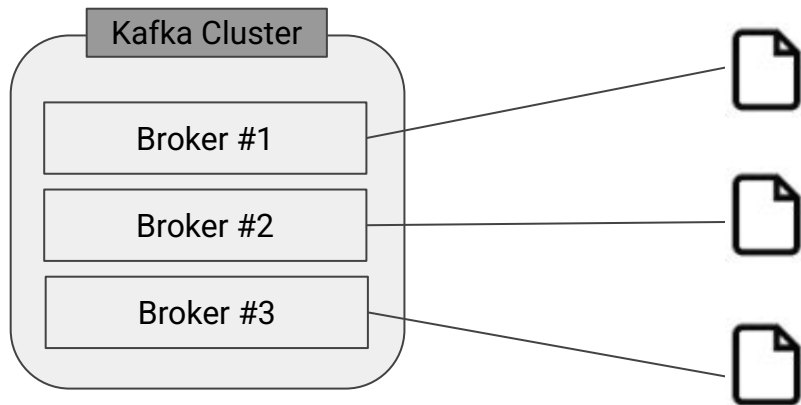
Kafka 특징



❖ 확장성

- 데이터 양에 따라 무중단 스케일인/아웃
- 초기 작은 서버 개수로 시작
- 점진적 서버 증설로 대응

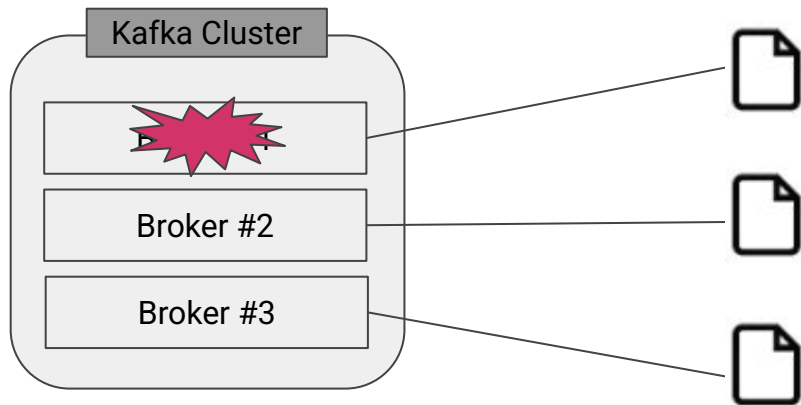
Kafka 특징



❖ 영속성

- 디스크 기반의 파일 시스템에 데이터 저장
- 브로커에 이슈가 발생하여 종료되도 프로세스 재시작으로 안전하게 데이터 처리 가능

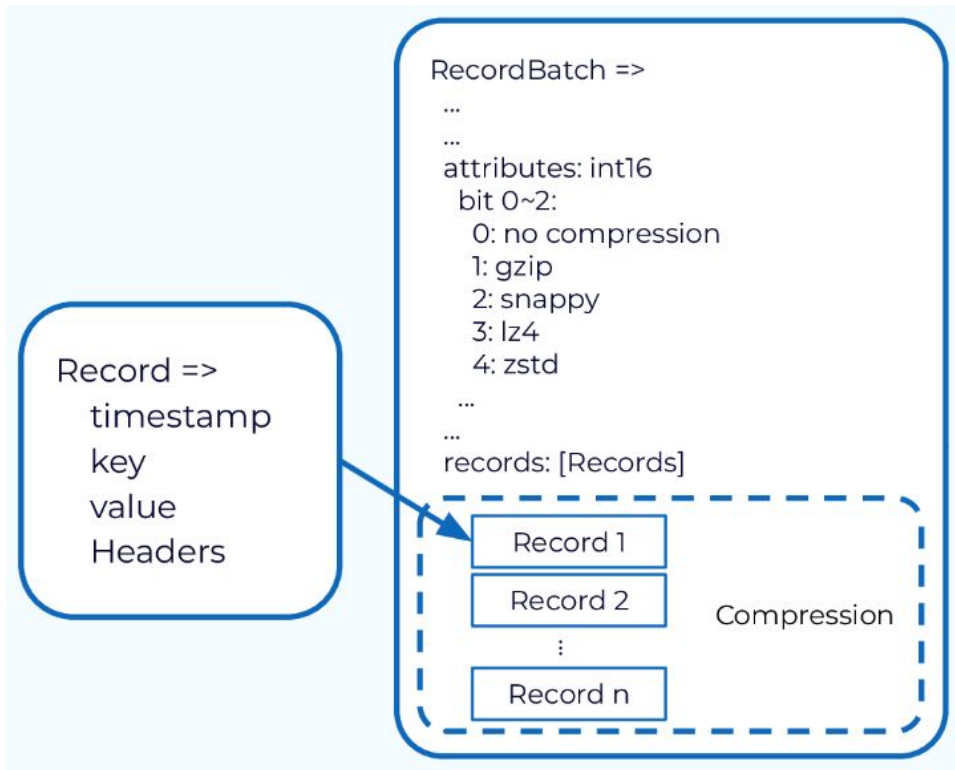
Kafka 특징



❖고가용성

- 데이터 복제를 통해 고가용성의 특징을 가짐
- 일부 브로커 장애 발생시에도 데이터 처리 가능

Kafka 주요 기능 - 레코드



- 레코드는 **kafka** 의 기본 데이터 단위
- 각 레코드는 **key, value, timestamp** 로 구성
- 레코드는 파티션에 추가되며, 순서대로 기록
- 예시:

`{"user_id": 123, "signup_date": "2024-07-19"}`

Kafka 주요 기능 - 토픽, 파티션



- 토픽은 데이터베이스의 테이블과 유사한 개념
- 토픽은 하나 이상의 파티션으로 구성
- 파티션은 메시지가 저장되는 단위 그룹
- 파티션은 순서가 보장되며, 각 파티션은 고유의 로그를 가짐
- 파티션은 확장은 가능하지만 축소는 불가
- 파티션은 서로 다른 서버에 분산 될 수 있음

Kafka 주요 기능 - 토픽, 파티션

❖ 적정 파티션 수

| | 서버수 | 서버당 메시지 전송 수 | 합계 | 필요 파티션 수 |
|------|-----|--------------|----------|----------|
| 프로듀서 | 4 | 10 메세지/초 | 40 메세지/초 | 4 |
| 컨슈머 | 8 | 5 메세지/초 | 40 메세지/초 | 8 |

❖ 총 데이터 크기 = 파티션 수 × 복제 수 × 데이터 크기



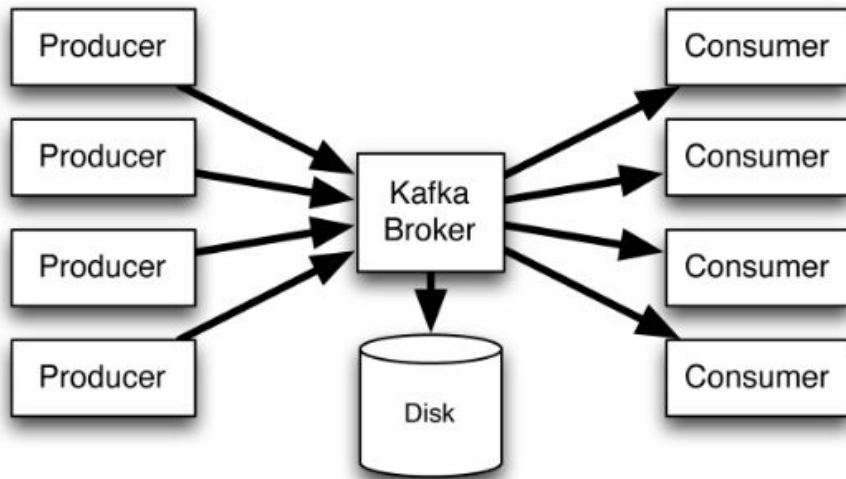
브로커

❖ 브로커

- 일반적으로 '카프카' 라고 불리는 시스템을 말함
- 프로듀서와 컨슈머는 별도의 애플리케이션으로 구성, 브로커는 카프카 자체이기 때문
- '카프카를 통해 메시지를 전달한다' 에서 카프카는 브로커를 의미

❖ 역할

- 메시지 저장 및 전달
- 토픽과 파티션 관리
- 클라이언트(프로듀서/컨슈머) 요청 처리
- 데이터 복제 및 장애 복구 관리

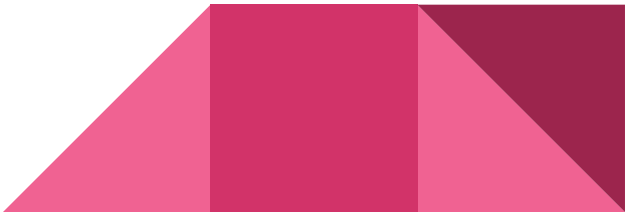


출처 : <https://mapr.com/ebooks/streaming-architecture/chapter-04-apache-kafka-overview.html>

브로커

❖ 브로커 주요 특징

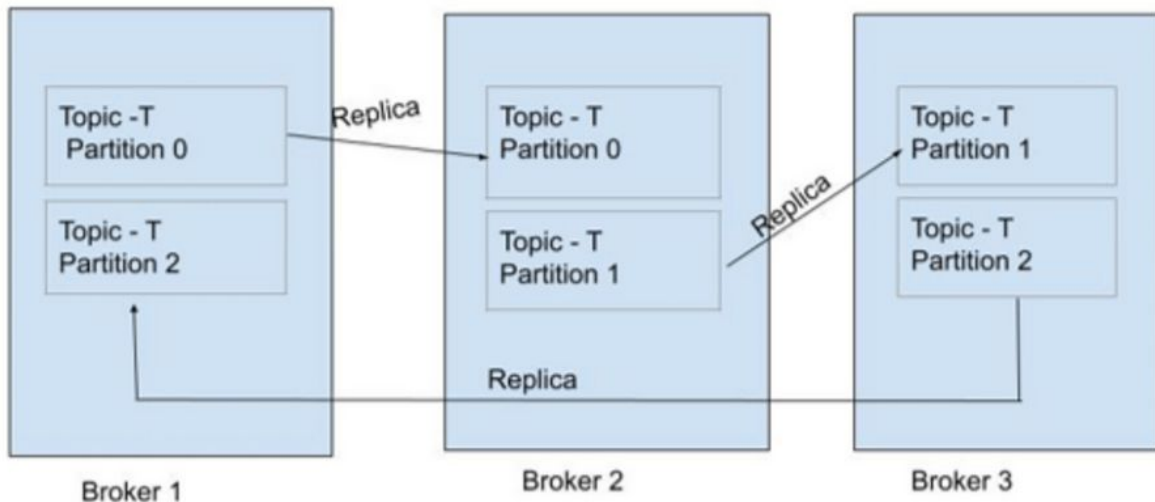
- **Peer-to-Peer** 구조: 카프카 클러스터 내의 모든 브로커가 클라이언트 요청을 처리할 수 있는 대등한 구조
- 확장성: 브로커를 추가하여 클러스터를 쉽게 확장 가능
- 고가용성: 여러 브로커가 데이터를 복제하여 저장함으로써 장애 발생 시에도 서비스 연속성 보장
- 성능: 최적화된 네트워크 프로토콜을 사용하여 고성능 메시지 처리 지원



브로커 - 데이터 복제

❖ 복제

- 파티션은 리더 파티션과 팔로워 파티션으로 구성
 - 리더파티션: 프로듀서, 컨슈머와 직접 통신하는 파티션
 - 팔로우 파티션: 리더 파티션을 계속 복제하여 데이터를 분산 저장하는 용도, 리더 파티션이 존재하는 브로커와 다른 브로커에 존재



주키퍼

분산 시스템에서 중요한 역할을 하는 코디네이션 서비스

❖ 주요 역할

- 분산 코디네이션 서비스
 - 분산 시스템의 노드 간 조율을 쉽게 할 수 있도록 돕는 오픈소스 프로젝트
- 구성 관리
 - 분산 시스템의 설정 정보를 중앙에서 관리하고, 변경 사항을 모든 노드에 즉시 반영
- 네이밍 서비스
 - 분산 시스템의 각 노드를 식별할 수 있도록 네이밍 서비스 제공
- 동기화
 - 여러 노드 간의 동기화를 통해 데이터 일관성 유지
- 그룹 서비스
 - 클러스터 내의 노드 그룹을 관리하고, 노드의 상태를 모니터링

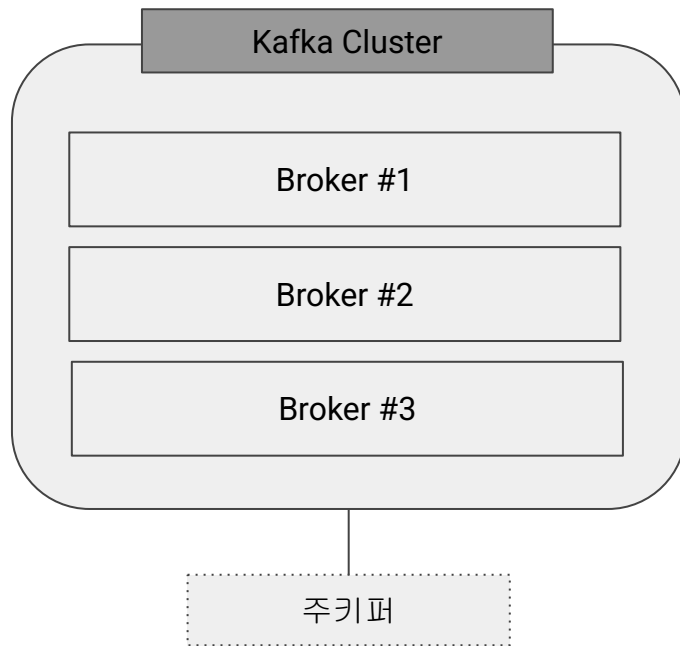
❖ Kafka v4.0 제거될 예정 (KRaft 모드로 대체)

카프카 KRaft 모드

KRaft 모드는 Kafka가 단순화 되어 확장성, 안정성, 일관성 보장

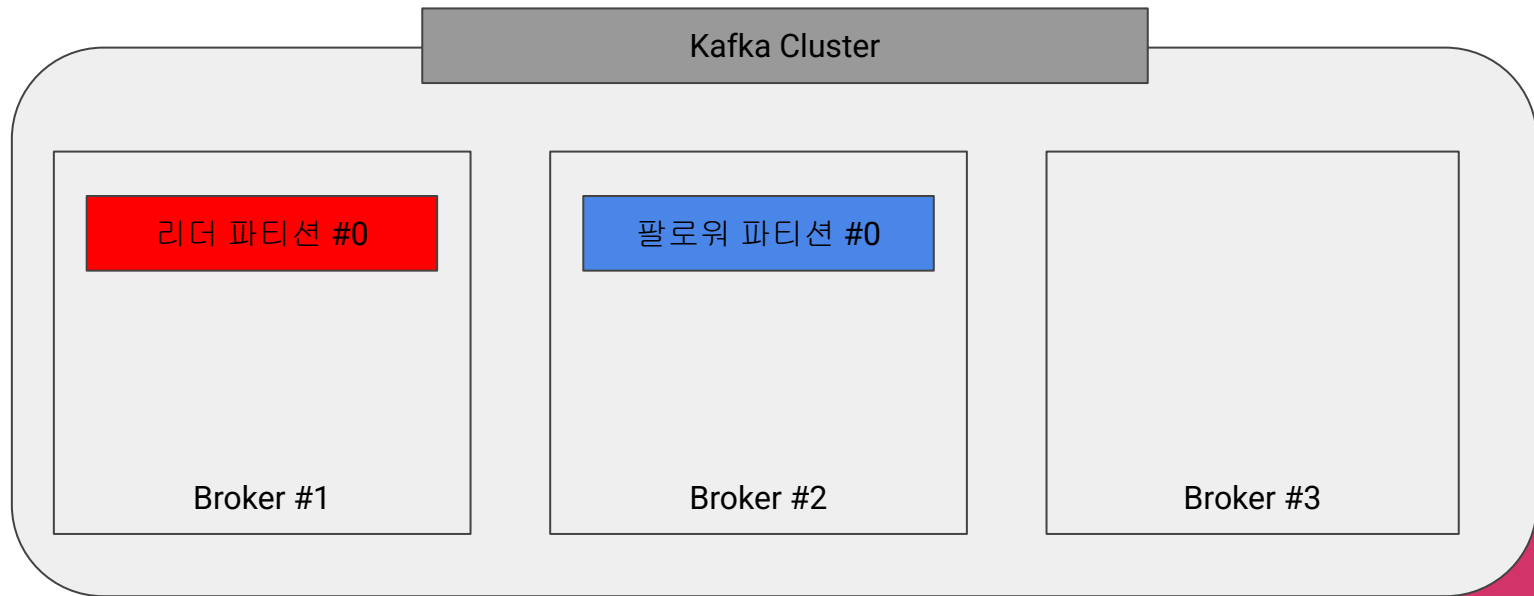
- ❖ 확장성
 - 적합한 수의 브로커로 크기가 조절이 되어 처리량, 대기 시간, 요구사하응^ㄹ 충족하는 컴퓨팅을 통해 파티션 확장 가능
- ❖ 즉각적 장애 조치
 - Kafka 내부에서 동작하는 시스템으로 메타데이터 장애조치가 거의 즉각적 처리
- ❖ 관리 용이성
 - Kafka 자체에 포함되어 zookeeper 와 다르게 별도 관리가 필요하지 않음
- ❖ Production 레벨 Kafka 3.3.0 버전에서 정식으로 릴리즈

카프카 클러스터



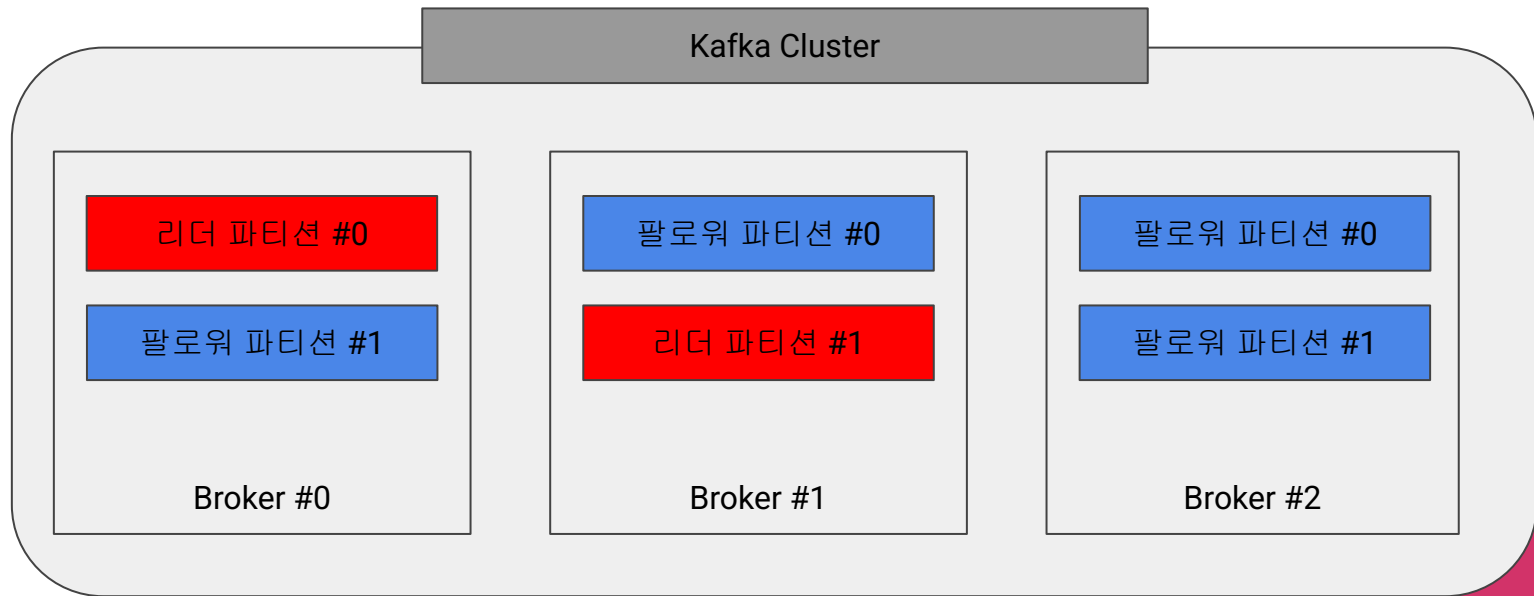
카프카 클러스터

파티션=1, 복제=2 인 토픽



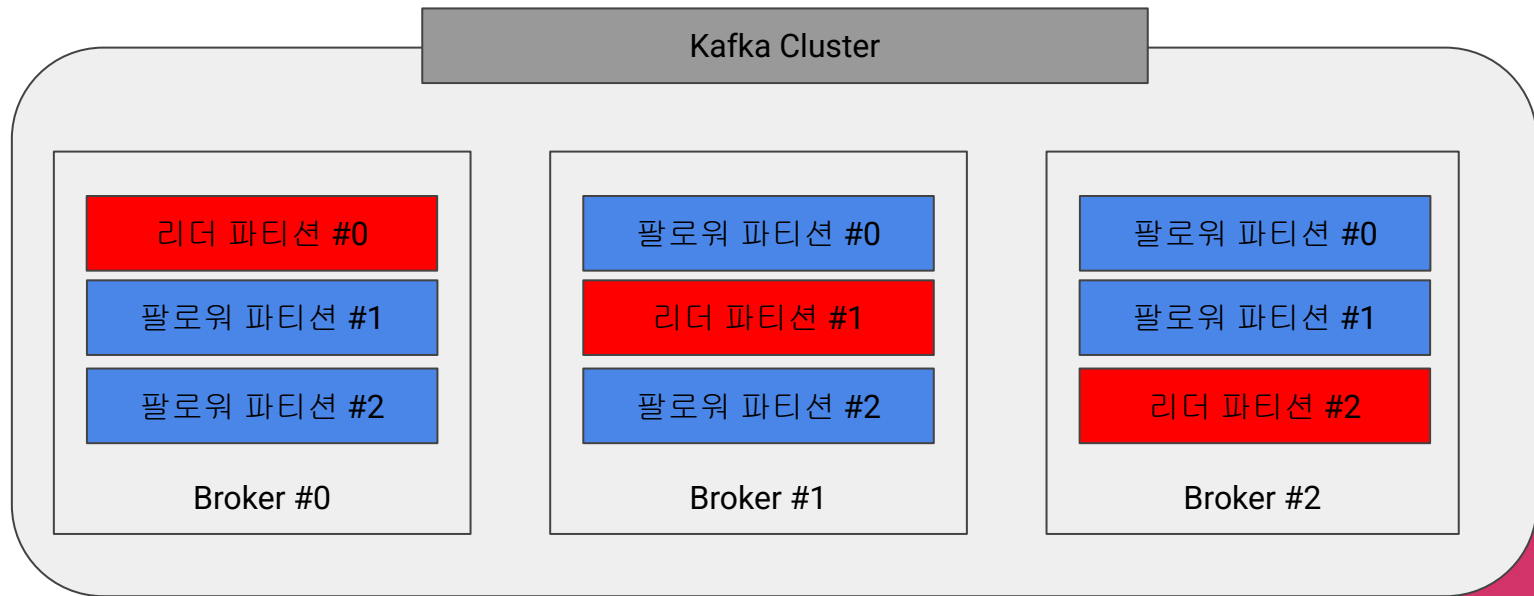
카프카 클러스터

파티션=2, 복제=3 인 토픽



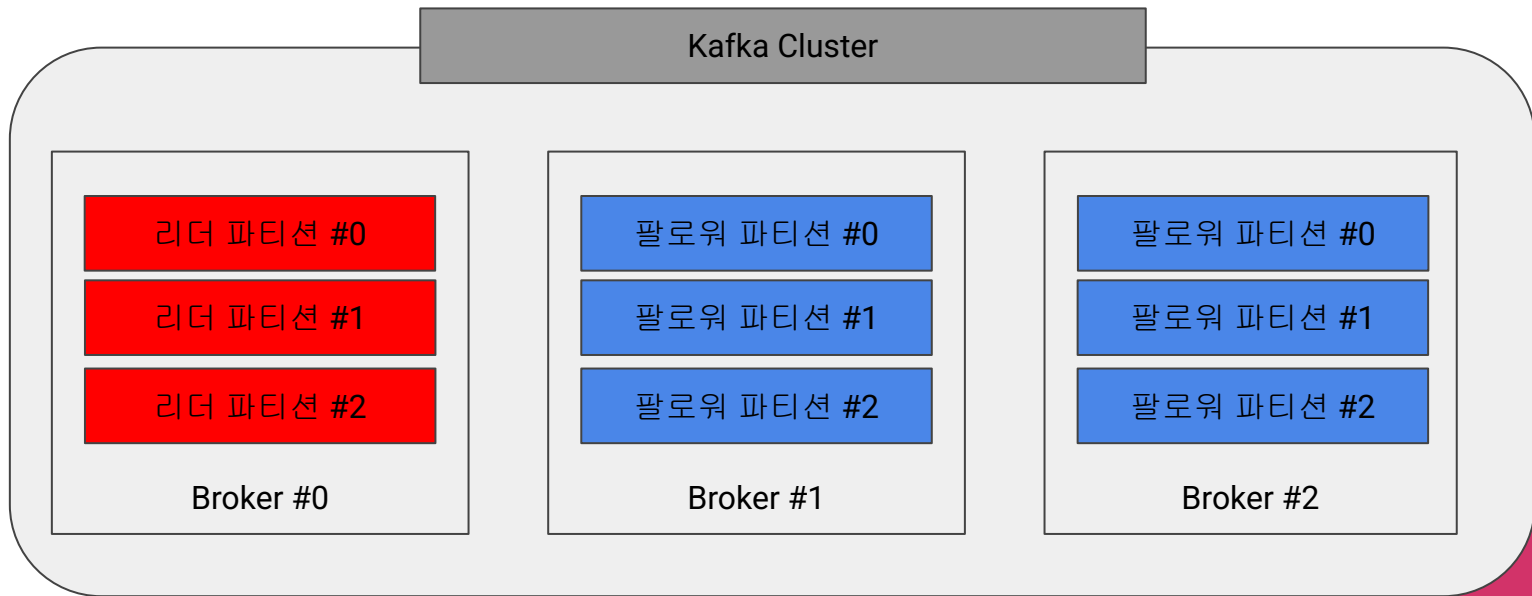
카프카 클러스터

파티션=3, 복제=3 인 토픽



카프카 클러스터

파티션=3, 복제=3 인 토픽 - 리더 파티션이 몰려있는 경우 (reassign 필요) # 실습



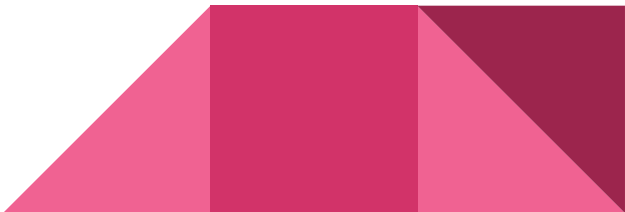
프로듀서

카프카 클러스터에 메시지를 발행(**publish**)하는 클라이언트 애플리케이션

주요 특징

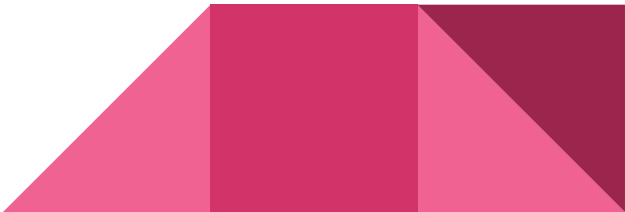
- 메시지를 특정 토픽으로 전송
- 키-값 쌍 형태로 메시지를 전송
- 메시지 전송 시 파티셔닝을 통해 메시지 분배를 제어
- 비동기 또는 동기 방식으로 메시지를 전송
- 배치 처리를 통해 성능을 최적화

프로듀서 동작 과정

1. 메시지 생성: 키-값 쌍으로 **Producer Record** 객체 생성
 2. 직렬화: 메시지를 바이트 배열로 변환
 3. 파티셔닝: 메시지를 보낼 파티션 결정
 4. 배치 처리: 메시지를 버퍼에 모아서 배치로 전송
 5. 전송: 배치를 브로커로 전송
- 

프로듀서

- `bootstrap.servers`: 브로커 목록
- `key.serializer, value.serializer`: 메시지 직렬화 방식
 - `StringSerializer, IntegerSerializer, JsonSerializer, ByteArraySerializer ...`
- `acks`: 메시지 전송 신뢰성 수준
 - `ack=0` : 서버의 응답을 기다리지 않음. 성공 보장안됨
 - `ack=1` : 서버가 수신 후 응답. 복제 기다리지 않음
 - `ack=all or -1` : 최소 한개 복제 후 응답. 유실방지
- `batch.size`: 배치 크기
- `linger.ms`: 배치 전송 대기 시간



컨슈머와 컨슈머 그룹

- 컨슈머

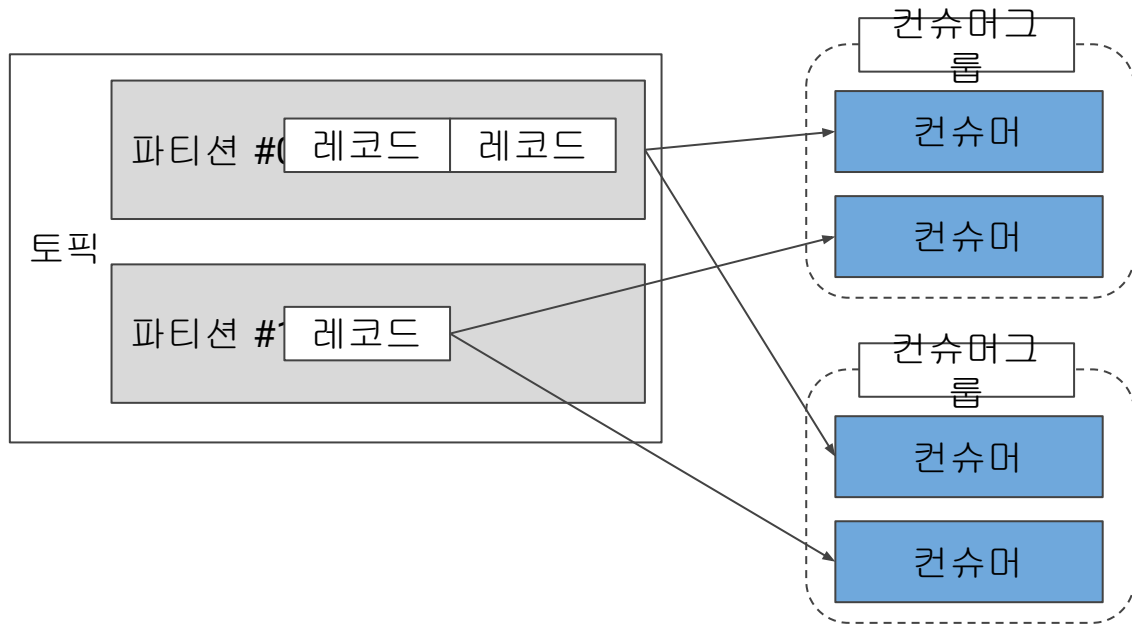
- 토픽의 메시지를 읽어서 처리
- 메시지가 생성된 순서대로 읽음
- 메시지의 오프셋을 기록하여 읽는 메시지 위치를 관리
- 중단 되었다 다시 시작하더라도 다음 메시지부터 처리
- 컨슈머는 컨슈머 그룹의 멤버로 동작

- 컨슈머 그룹

- 컨슈머들은 컨슈머 그룹에 속하며 컨슈머 그룹은 하나 이상의 컨슈머로 구성
- 한 토픽을 소비하는 여러 컨슈머 그룹 동작 가능
- 컨슈머 그룹 별로 각 파티션의 메시지 오프셋 관리
- 토픽의 각 파티션은 컨슈머 그룹 내에서 하나의 컨슈머만 소비할 수 있음



컨슈머와 컨슈머 그룹



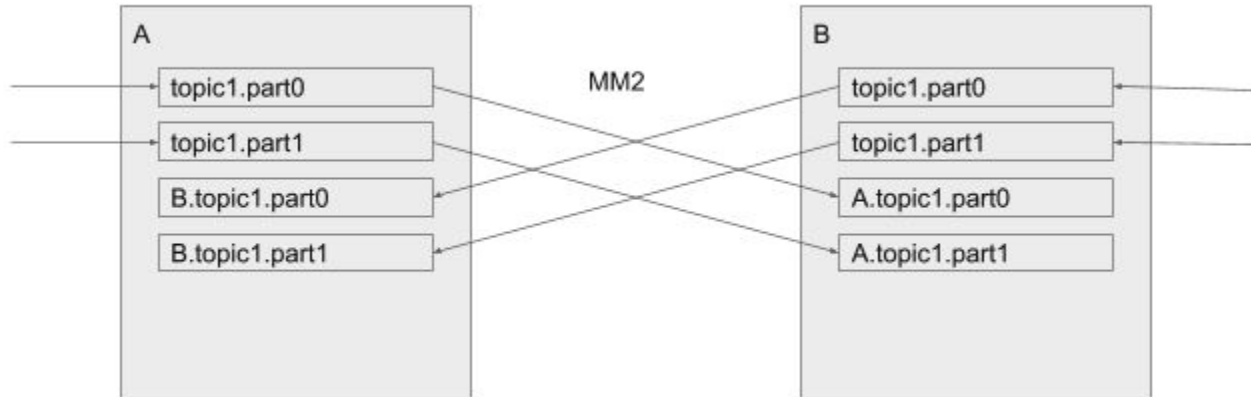
Kafka MirrorMaker 2

또 다른 **Kafka** 클러스터를 대상으로 데이터를 미러링하는 기능

- 각 클러스터 간 토픽 설정에 대한 동기화 기능
- 모니터링에 필요한 클러스터 간 **metric** 제공
- 클러스터간 데이터 미러링을 단일 **properties** 로 정의
- 새로운 **Topic, Partition** 감지 기능



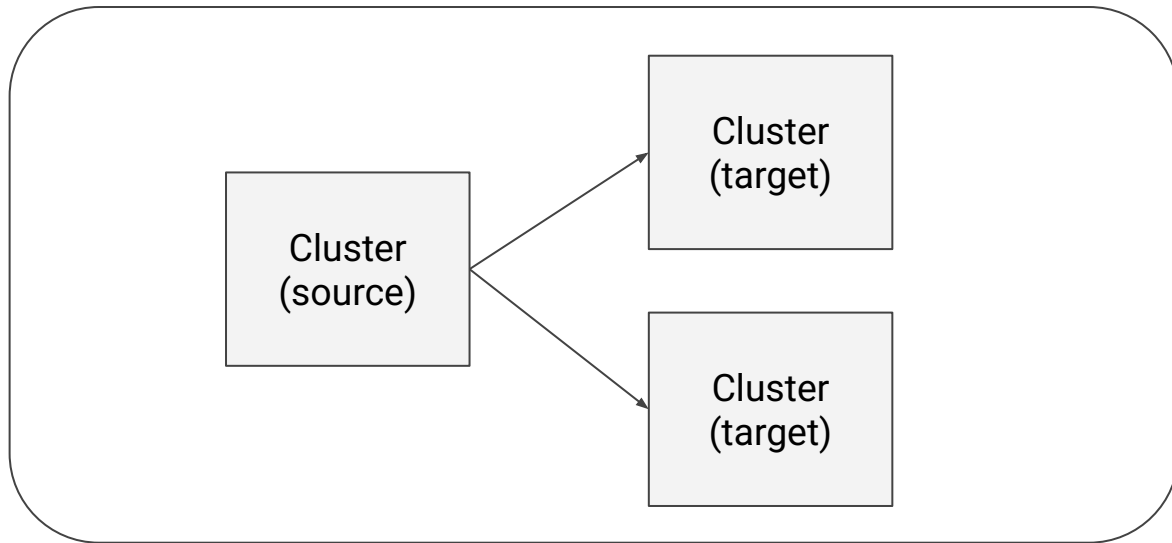
Kafka MirrorMaker 2



출처 : <https://cwiki.apache.org/confluence/display/KAFKA/KIP-382%3A+MirrorMaker+2.0>

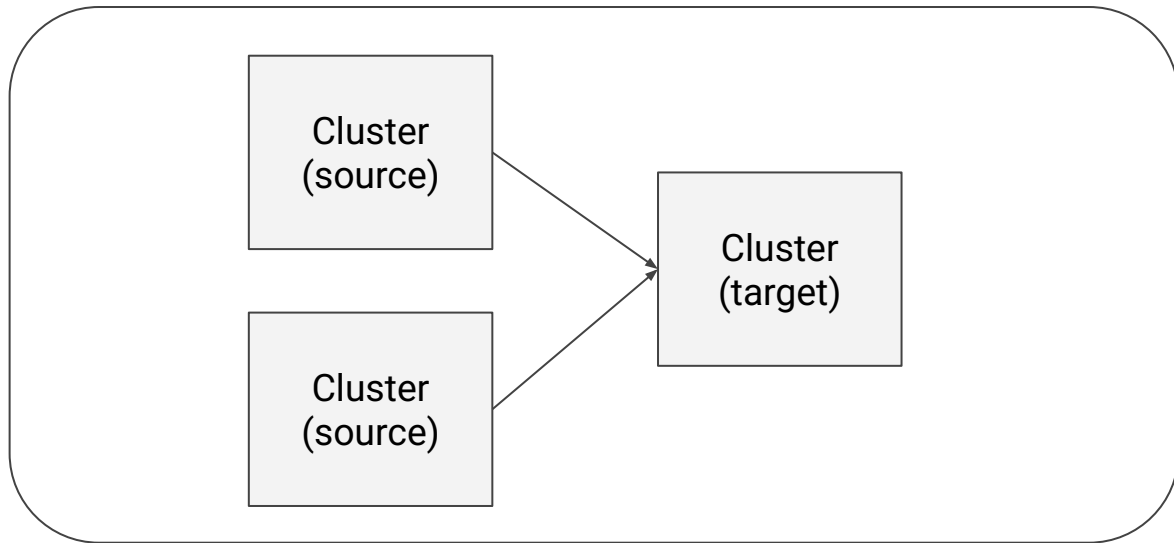
Kafka MirrorMaker 2 (Fan-out)

1개의 클러스터로부터 2개 이상의 클러스터로 데이터를 분산하는 방식



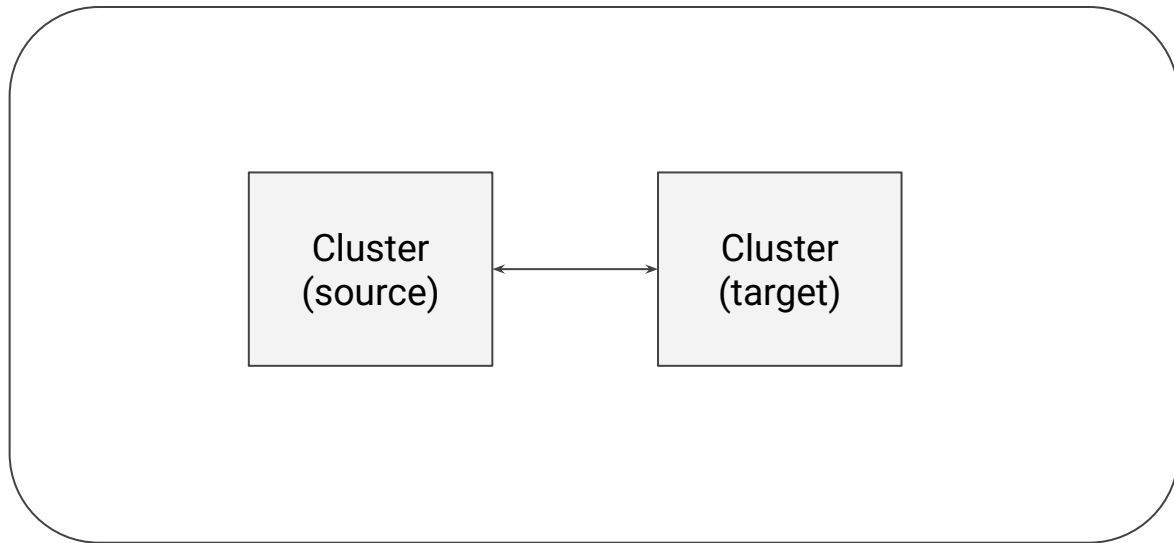
Kafka MirrorMaker 2 (Aggregation)

2개 이상의 클러스터로부터 1개의 클러스터로 데이터를 미러링하여 활용하는 방식



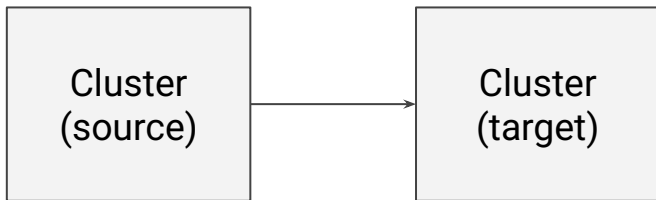
Kafka MirrorMaker 2 (Active-Active)

양쪽 클러스터 간에 서로의 데이터를 미러링 하는 방식

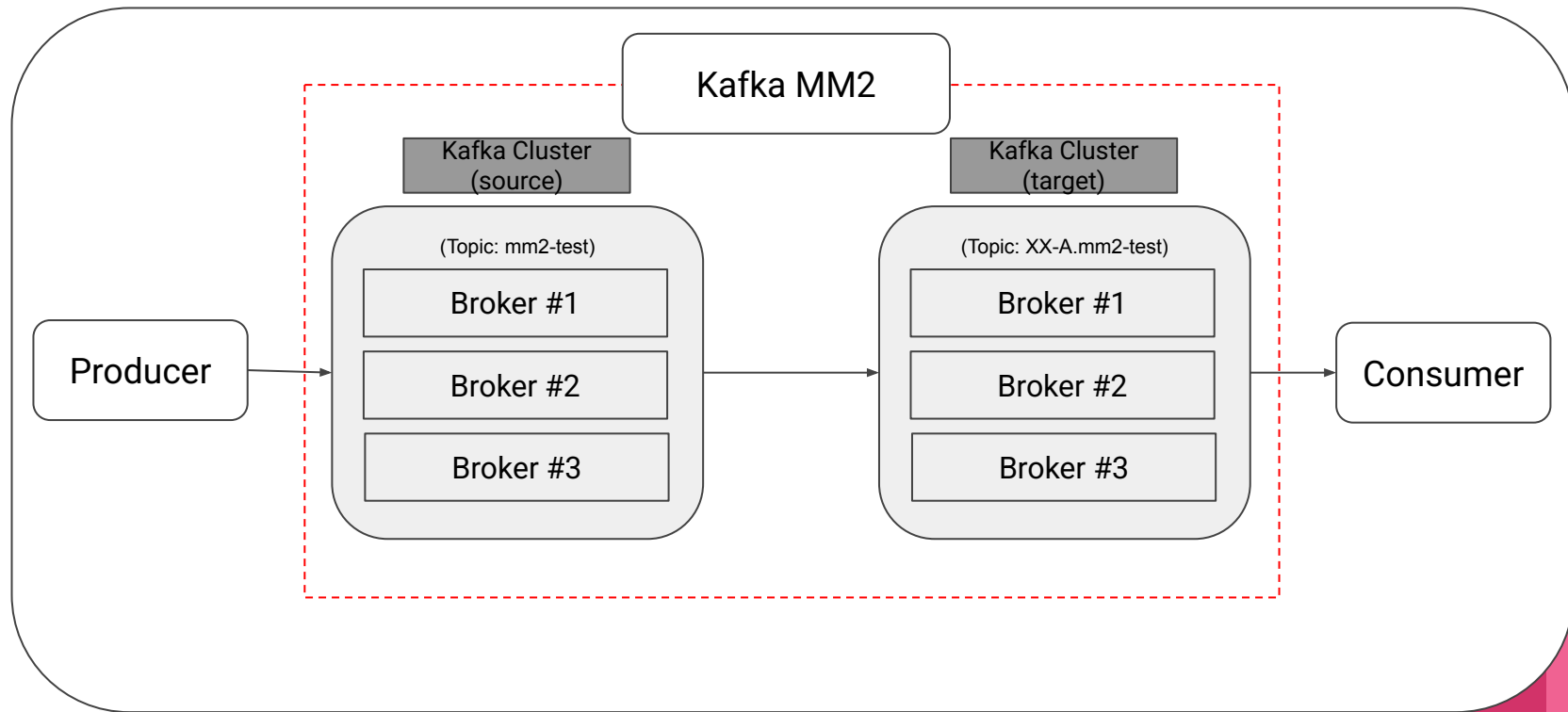


Kafka MirrorMaker 2 (Active-Passive)

소스 클러스터의 데이터를 목적지인 클러스터로 데이터를 미러링하는 방식



Kafka MirrorMaker 2 (실습)



Kafdrop

Apache Kafka를 위한 웹 기반 UI 도구

- 브로커 정보
- 토픽 목록 및 상세 정보
- 파티션 정보
- 컨슈머 그룹 정보
- 메시지 내용 조회 기능
- 토픽생성, 삭제

Kafka Cluster Overview

| | |
|-----------------------------------|----------------|
| Bootstrap servers | localhost:9092 |
| Total topics | 8 |
| Total partitions | 87 |
| Total preferred partition leader | 100% |
| Total under-replicated partitions | 0 |

Brokers

| ID | Host | Port | Rack | Controller | Number of partitions (% of total) |
|----|-----------|------|------|------------|-----------------------------------|
| 0 | localhost | 9092 | - | Yes | 87 (100%) |

Topics

| Name <input type="text" value="sync"/> (4) | Partitions | % Preferred | # Under-replicated | Custom Config |
|--|------------|-------------|--------------------|---------------|
| StrikeSettleST.bet-sync.2019-09-24-191801 | 16 | 100% | 0 | No |
| StrikeSettleST.market-sync.2019-09-24-191801 | 1 | 100% | 0 | No |
| WinST.bet-sync.2019-09-24-191742 | 16 | 100% | 0 | No |
| WinST.market-sync.2019-09-24-191742 | 1 | 100% | 0 | No |

CMAK (Cluster Manager for Apache Kafka)

대규모 데이터 스트리밍 환경에서 **Kafka** 클러스터를 효율적으로 관리하기 위해 사용

- 클러스터 관리
- 토픽관리
- 파티션 정보
- 컨슈머 그룹 관리
- 파티션 관리

The screenshot displays the CMAK web interface for managing a Kafka cluster. The top navigation bar includes the 'Kafka Manager' logo, a dropdown for 'eds_test_cluster', and links for 'Cluster', 'Brokers', 'Topic', 'Preferred Replica Election', and 'Reassign Partitions'. The breadcrumb trail shows 'Clusters / eds_test_cluster / Topics / test3'.

The main section is titled 'test3' and contains two primary panels:

- Topic Summary:** A table providing key metrics for the 'test3' topic.

| Metric | Value |
|-----------------------------|----------|
| Replication | 2 |
| Number of Partitions | 2 |
| Total number of Brokers | 2 |
| Number of Brokers for Topic | 2 |
| Preferred Replicas % | 100 |
| Brokers Skewed % | 0 |
| Brokers Spread % | 100 |
| Under-replicated % | 0 |
| Config | Value |
| retention.ms | 86400000 |
| max.message.bytes | 400 |
- Operations:** A section with three buttons: 'Generate Partition Assignments', 'Reassign Partitions', and 'Delete Topic'.

Below the operations panel is the **Partitions by Broker** table:

| Broker | # of Partitions | Partitions | Skewed? |
|--------|-----------------|------------|---------|
| 0 | 2 | (1,0) | false |
| 1 | 2 | (1,0) | false |

At the bottom, the **Partition Information** table provides detailed data for each partition:

| Partition | Leader | Replicas | In Sync Replicas | Preferred Leader? | Under Replicated? |
|-----------|--------|----------|------------------|-------------------|-------------------|
| 0 | 1 | (1,0) | (1,0) | true | false |
| 1 | 0 | (0,1) | (0,1) | true | false |

[실습] Topic 생성하기


```
$ ./kafka-topics.sh --bootstrap-server localhost:9092 \  
--topic test --create
```

파티션 지정 생성하기

```
$ ./kafka-topics.sh --bootstrap-server localhost:9092 \  
--partition 2 \ # 브로커의 개수와 상관없이 지정할 수 있음  
--topic test --create
```

파티션 복제 개수 지정 생성하기

```
$ ./kafka-topics.sh --bootstrap-server localhost:9092 \  
--replication-factor 2 \ # 브로커 개수 보다 작거나 같아야 함  
--topic test --create
```



[실습] Topic 생성하기

retention.ms 지정하여 생성

```
$ ./kafka-topics.sh --bootstrap-server localhost:9092 \
```

```
--partition 2 \
```

```
--config retention.ms=60000 \ # 60초
```

```
--topic test --create
```

--config retention.ms 옵션은 토픽의 메시지가 보존되는 기간을 밀리초 단위로 설정

토픽 삭제 정책은 **delete**(삭제), **compact**(압축) 가 있음

기본 정책은 **delete**



[실습] Topic 상세 조회

```
$ ./kafka-topics.sh --bootstrap-server localhost:9092 \  
--topic test \  
--describe
```

```
Topic: test      TopicId: cuJWwERBR8yiGn4xPxNhTQ PartitionCount: 1      ReplicationFactor: 1      Configs: segment.bytes=1073741824  
Topic: test      Partition: 0      Leader: 1      Replicas: 1      Isr: 1
```



[실습] Topic 목록 조회

```
./kafka-topics.sh --bootstrap-server localhost:9092 \  
--list
```



[실습] 테스트 데이터 수신(consumer)

```
./kafka-console-consumer.sh --bootstrap-server localhost:9092 \  
--topic test  
>college
```




[실습] 테스트 데이터 보내기(producer)

```
./kafka-console-producer.sh --bootstrap-server localhost:9092 \  
--topic test
```

- 메시지 값만 보내기

```
./kafka-console-producer.sh --bootstrap-server localhost:9092 \  
--topic test \  
--property "parse.key=true" \  
--property "key.separator=:"  
>skcc:college
```



[실습] 테스트 데이터 수신(consumer)

- 저장된 데이터 확인하기

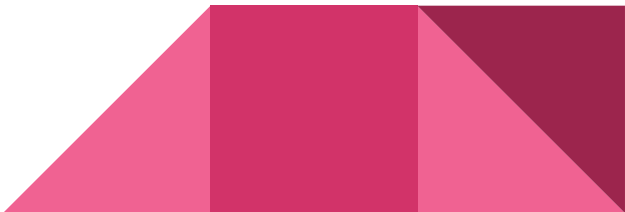
```
./kafka-console-consumer.sh --bootstrap-server localhost:9092 \  
--topic test \  
--property "print.key=true" \  
--property "key.separator=:" \  
--from-beginning  
:college  
abc:def  
123:456  
1234:0987:567  
234:456:789
```

--from-beginning : 토픽 가장 처음 넣은 레코드부터 출력

--max-messages : 가져올 메시지의 개수를 지정할 수 있음

--partition : 토픽의 특정 파티션만 지정

--whitelist : 여러개 토픽으로 부터 데이터를 가져올 수 있음



[실습] 테스트 데이터 수신(consumer)

❖ 컨슈머 그룹

```
./kafka-console-consumer.sh --bootstrap-server localhost:9092 \  
--topic test \  
--group test-group
```

--group 옵션 : 컨슈머 그룹을 이용하여 토픽 데이터를 수신

- 컨슈머 그룹을 사용하면 데이터를 병렬로 처리하여 효율성을 높일 수 있다.
- Offset 관리를 통해 메시지의 중복 처리를 방지하고 데이터의 일관성을 유지한다.
- 컨슈머 그룹의 Offset 정보는 __consumer_offsets 토픽에 저장되어 관리된다.

[실습] 컨슈머 그룹 조회

```
./kafka-consumer-group.sh --bootstrap-server localhost:9092 --list  
test-group
```

- 컨슈머 그룹 정보 조회

```
./kafka-consumer-group.sh --bootstrap-server localhost:9092 \  
--group test-group \  
--describe
```

| GROUP | TOPIC | PARTITION | CURRENT-OFFSET | LOG-END-OFFSET | LAG | CONSUMER-ID | HOST | CLIENT-ID |
|------------|-------|-----------|----------------|----------------|-----|---|------------|------------------|
| test-group | test | 0 | 10 | 10 | 0 | console-consumer-4f77f849-9768-4baa-8d8f-9685b3199d89 | /127.0.0.1 | console-consumer |