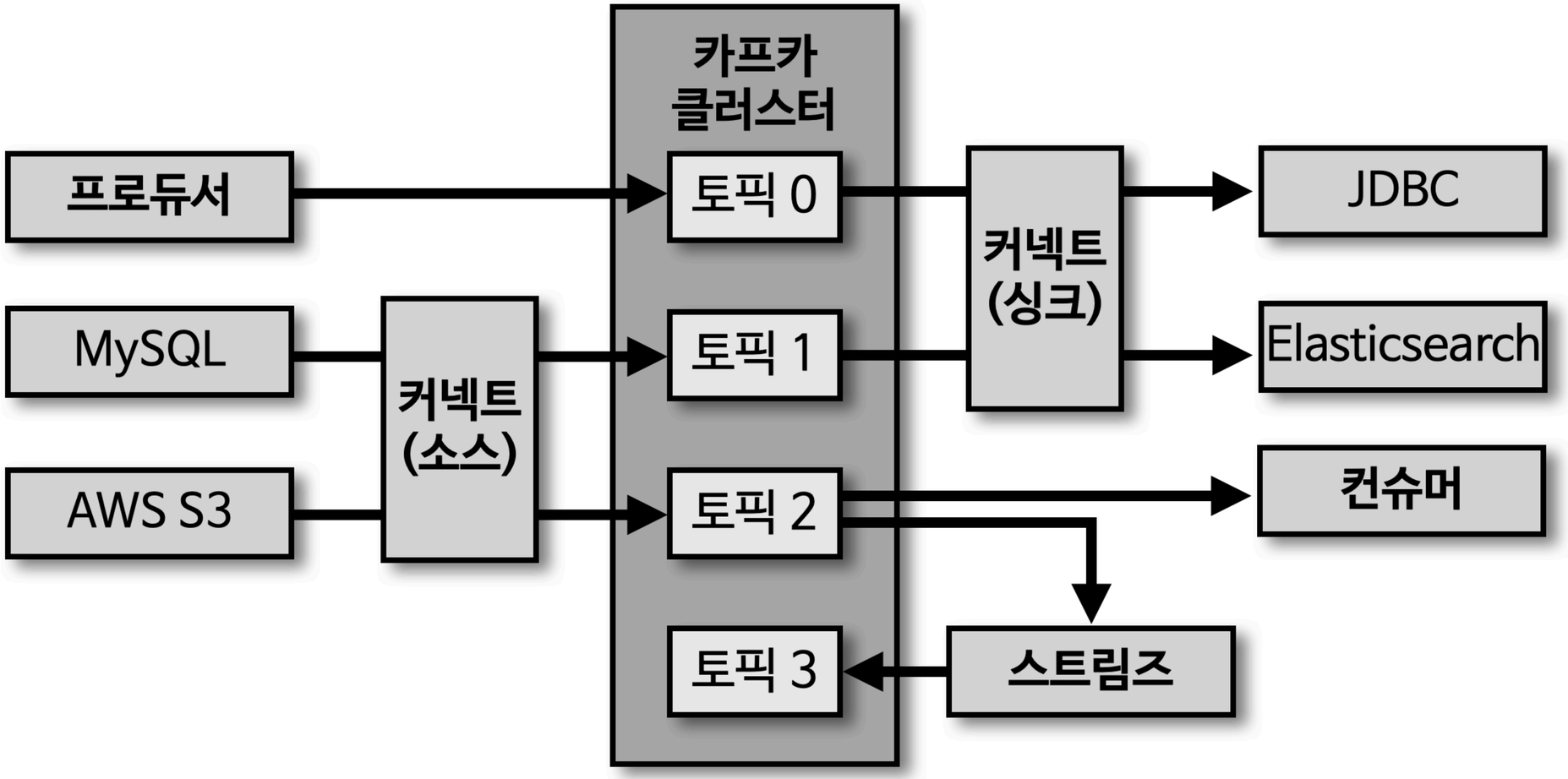
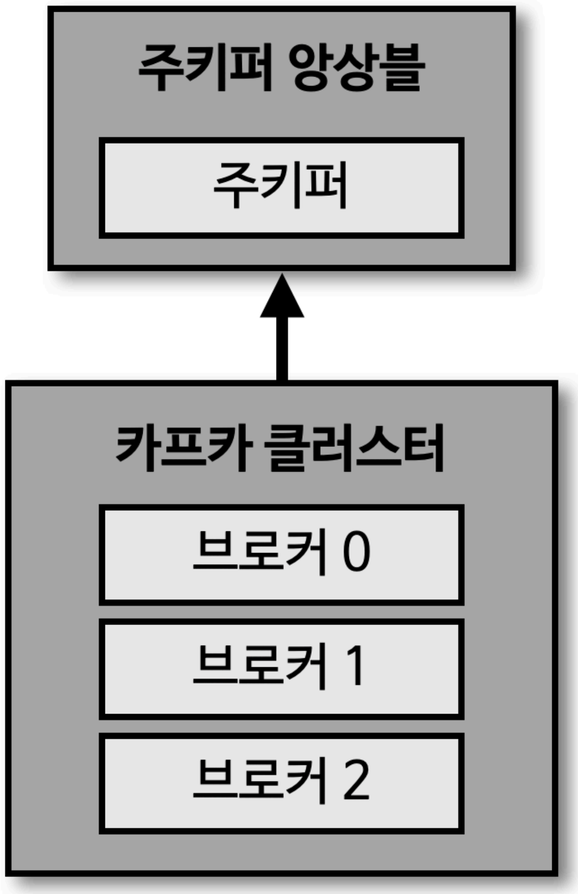
카프카 브로커와 클러스터

카프카 생태계

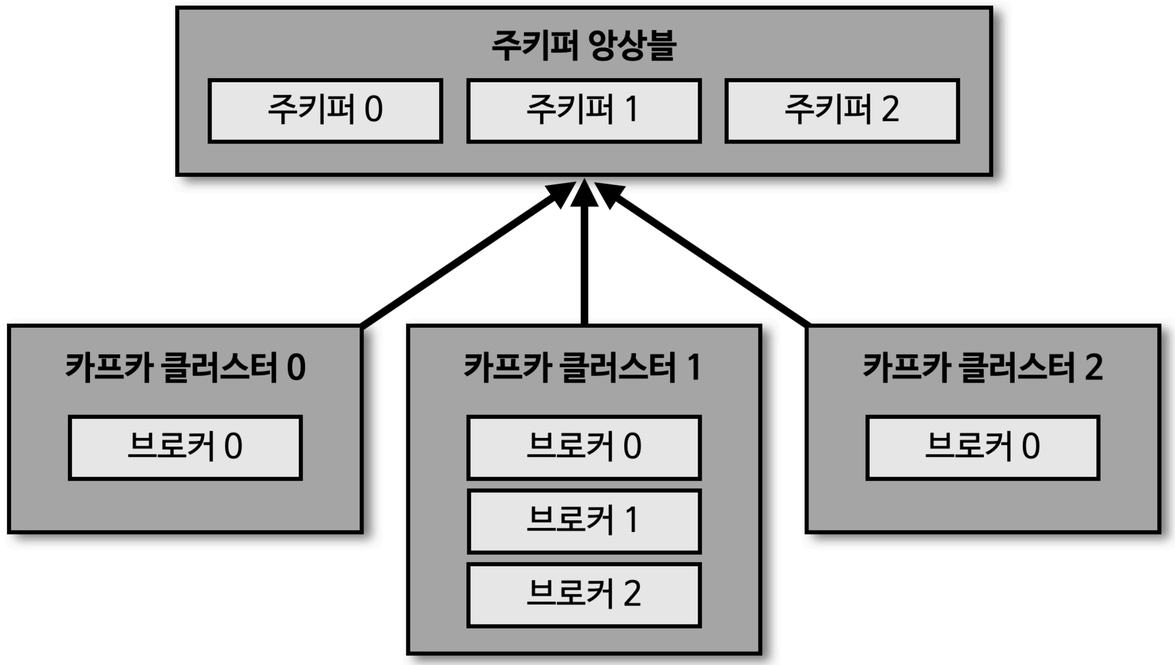

카프카
클러스터
프로듀서
MySQL
AWS S3
커넥트
(소스)
토픽 0
토픽 1
토픽 2
토픽 3
커넥트
(싱크)
JDBC
Elasticsearch
컨슈머
스트림즈

카프카 브로커·클러스터·주키퍼

카프카 브로커는 카프카 클라이언트와 데이터를 주고받기 위 해 사용하는 주체이자, 데이터를 분산 저장하여 장애가 발생하더라도 안전하게 사용할 수 있도록 도와주는 애플리케이션이다. 하나의 서버에는 한 개의 카프카 브로커 프로세스가 실행된다. 카프카 브로커 서버 1대로도 기본 기능이 실행되지만 데이터를 안전하게 보관하고 처리하기 위해 3대 이상의 브로커 서버를 1개의 클러스터로 묶어서 운영한다. 카프카 클러스터로 묶인 브로커들은 프로듀서가 보낸 데이터를 안전하게 분산 저장하고 복제하는 역할을 수행한다.



여러개의 카프카 클러스터가 연결된 주키퍼



- 카프카 클러스터를 실행하기 위해서는 주키퍼가 필요함
- 주키퍼의 서로 다른 znode에 클러스터를 지정하면 됨
- root znode에 각 클러스터별 znode를 생성하고 클러스터 실행시 root가 아닌 하위 znode로 설정
- 카프카 3.0 부터는 주키퍼가 없어도 클러스터 동작 가능

브로커의 역할 - 컨트롤러, 데이터 삭제

컨트롤러
클러스터의 다수 브로커 중 한 대가 컨트롤러의 역할을 한다. 컨트롤러는 다른 브로커들의 상태를 체크하고 브로커가 클러스터에서 빠지는 경우 해당 브로커에 존재하는 리더 파티션을 재분배한다. 카프카는 지속적으로 데이터를 처리해야 하므로 브로커의 상태가 비정상이라면 빠르게 클러스터에서 빼내는 것이 중요하다. 만약 컨트롤러 역할을 하는 브로커에 장애가 생기 면 다른 브로커가 컨트롤러 역할을 한다.

데이터 삭제
카프카는 다른 메시징 플랫폼과 다르게 컨슈머가 데이터를 가져가더라도 토픽의 데이터는 삭제되지 않는다. 또한, 컨슈머나 프로듀서가 데이터 삭제를 요청할 수도 없다. 오직 브로커만이 데이터를 삭제할 수 있다. 데이터 삭제는 파일 단위로 이루어지는데 이 단위를 '로그 세그먼트(log segment)'라고 부른다. 이 세그먼트에는 다수의 데이터가 들어 있기 때문에 일반적인 데이터베이스처럼 특정 데이터를 선별해서 삭제할 수 없다.

브로커의 역할 - 컨슈머 오프셋 저장, 코디네이터

컨슈머 오프셋 저장
컨슈머 그룹은 토픽이 특정 파티션으로부터 데이터를 가져가서 처리하고 이 파티션의 어느 레코드까지 가져갔는지 확인하기 위해 오프셋을 커밋한다. 커밋한 오프셋은 __consumer_ offsets 토픽에 저장한다. 여기에 저장된 오프셋을 토대로 컨슈머 그룹은 다음 레코드를 가져가서 처리한다.

그룹 코디네이터
코디네이터는 컨슈머 그룹의 상태를 체크하고 파티션을 컨슈머와 매칭되도록 분배하는 역할을 한다. 컨슈머가 컨슈머 그룹에서 빠지면 매칭되지 않은 파티션을 정상 동작하는 컨슈머로 할당하여 끊임없이 데이터가 처리되도록 도와준다. 이렇게 파티션을 컨슈머로 재할당하는 과정을 '리밸런스(rebalance)'라고 부른다.

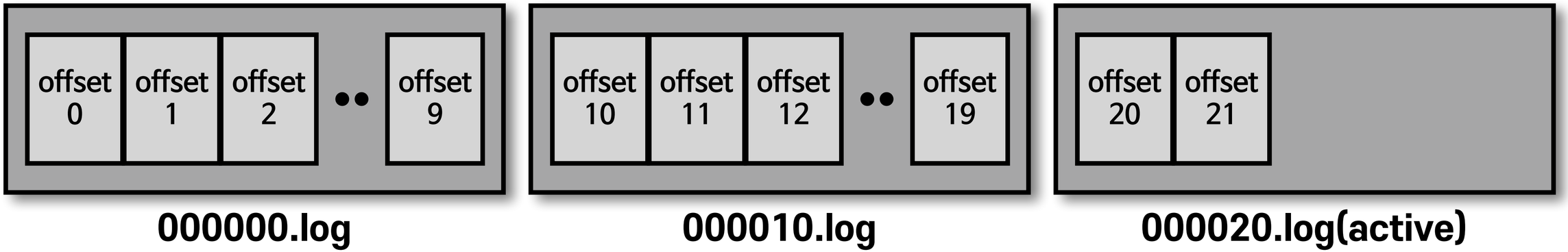
브로커의 역할 - 데이터의 저장

- 카프카를 실행할 때 config/server.properties의 log.dir 옵션에 정의한 디렉토리에 데이터를 저장한다. 토픽 이름과 파티션 번호의 조합으로 하위 디렉토리를 생성하여 데이터를 저장한다.
- hello.kafka 토픽의 0번 파티션에 존재하는 데이터를 확인할 수 있다. log에는 메시지와 메타데이터를 저장한다. index는 메시지의 오프셋을 인덱싱한 정보를 담은 파일이다. timeindex 파일에는 메시지에 포함된 timestamp값을 기준으로 인덱싱한 정보가 담겨 있다.

```
$ ls /tmp/kafka-logs
__consumer_offsets-0   __consumer_offsets-21
...
$ ls /tmp/kafka-logs/hello.kafka-0
00000000000000000000.index  00000000000000000000.log
00000000000000000000.timeindex  leader-epoch-checkpoint
```

로그와 세그먼트

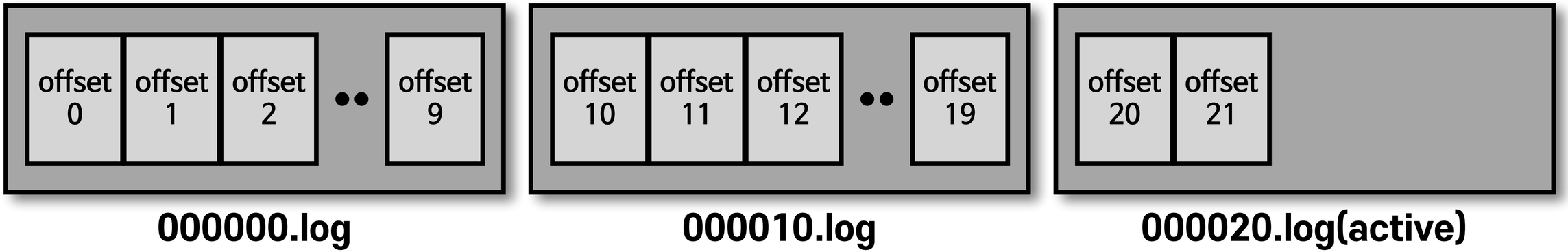
```
$ ls /tmp/kafka-logs/hello.kafka-0
00000000000000000000.log
00000000000000000010.log
00000000000000000020.log
```



- log.segment.bytes : 바이트 단위의 최대 세그먼트 크기 지정. 기본 값은 1GB.
- log.roll.ms(hours) : 세그먼트가 신규 생성된 이후 다음 파일로 넘어가는 시간 주기. 기본 값은 7일.

액티브 세그먼트

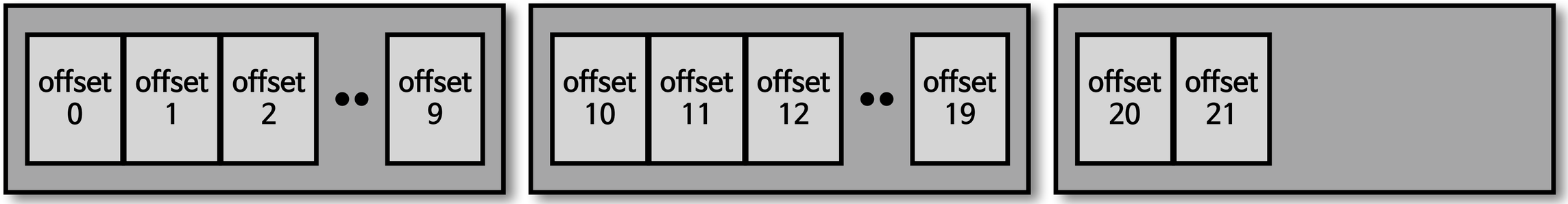
```
$ ls /tmp/kafka-logs/hello.kafka-0
00000000000000000000.log
00000000000000000010.log
00000000000000000020.log
```



가장 마지막 세그먼트 파일(쓰기가 일어나고 있는 파일)을 액티브 세그먼트라고 부른다. 액티브 세그먼트는 브로커의 삭제 대상에서 포함되지 않는다. 액티브 세그먼트가 아닌 세그먼트는 retention 옵션에 따라 삭제 대상으로 지정된다.

세그먼트와 삭제 주기(cleanup.policy=delete)

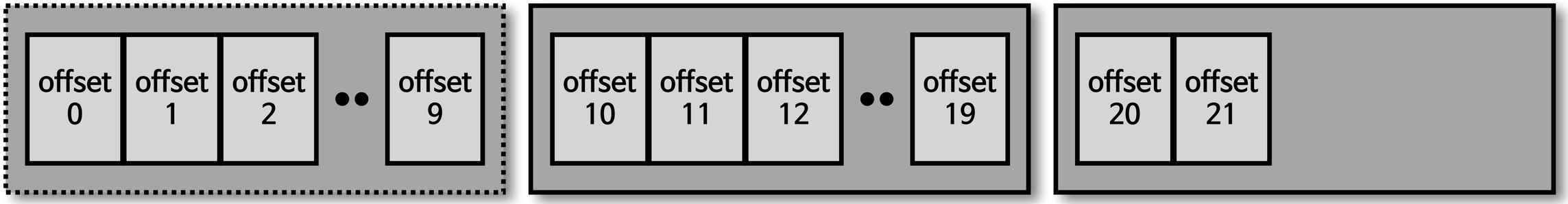
```
$ ls /tmp/kafka-logs/hello.kafka-0
00000000000000000000.log
00000000000000000010.log
00000000000000000020.log
```



- retention.ms(minutes, hours) : 세그먼트를 보유할 최대 기간. 기본 값은 7일.
- retention.bytes : 파티션당 로그 적재 바이트 값. 기본 값은 -1(지정하지 않음)
- log.retention.check.interval.ms : 세그먼트가 삭제 영역에 들어왔는지 확인하는 간격. 기본 값은 5분.

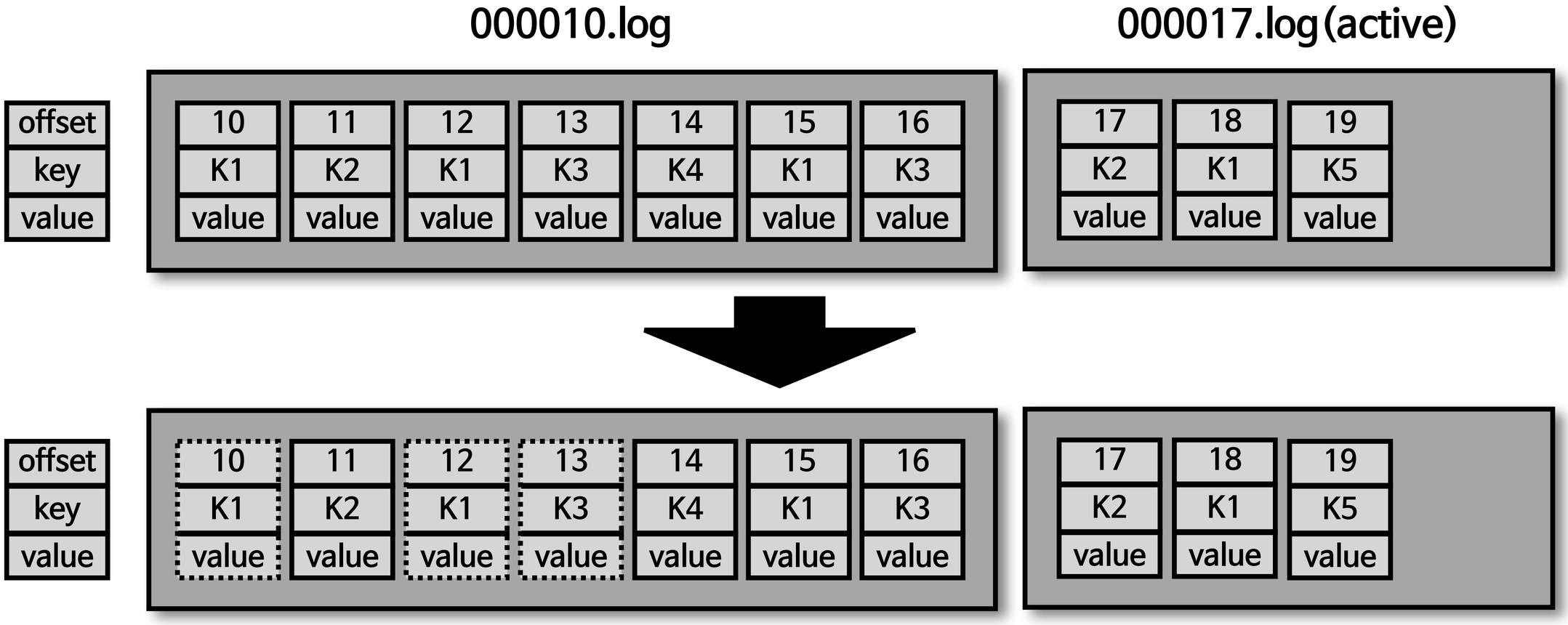
세그먼트의 삭제(cleanup.policy=delete)

```
$ ls /tmp/kafka-logs/hello.kafka-0
00000000000000000000.log
00000000000000000010.log
00000000000000000020.log
```



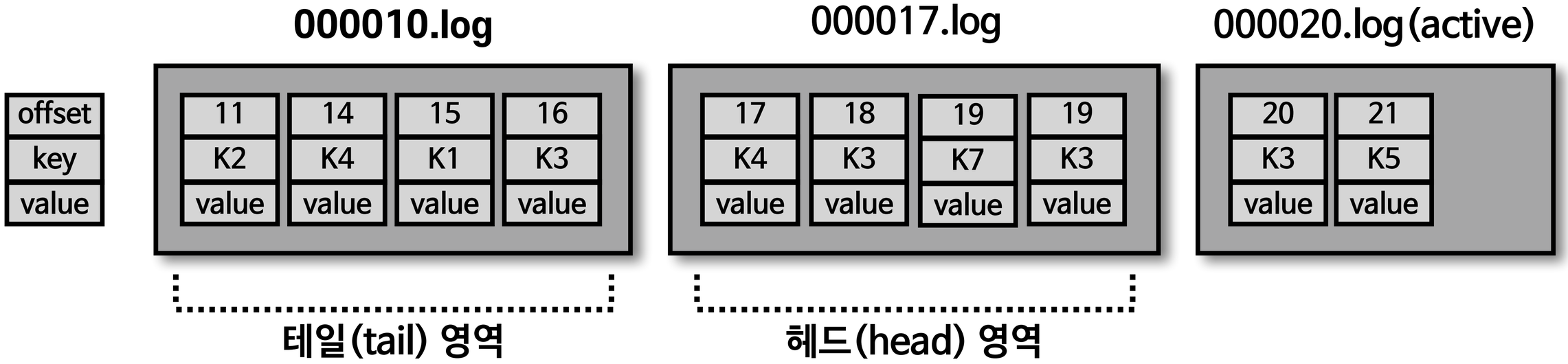
카프카에서 데이터는 세그먼트 단위로 삭제가 발생하기 때문에 로그 단위(레코드 단위)로 개별 삭제는 불가능하다. 또한, 로그(레코드)의 메시지 키, 메시지 값, 오프셋, 헤더 등 이미 적재된 데이터에 대해서 수정 또한 불가능하기 때문에 데이터를 적재할 때(프로듀서) 또는 데이터를 사용할 때(컨슈머) 데이터를 검증하는 것이 좋다.

cleanup.policy=compact



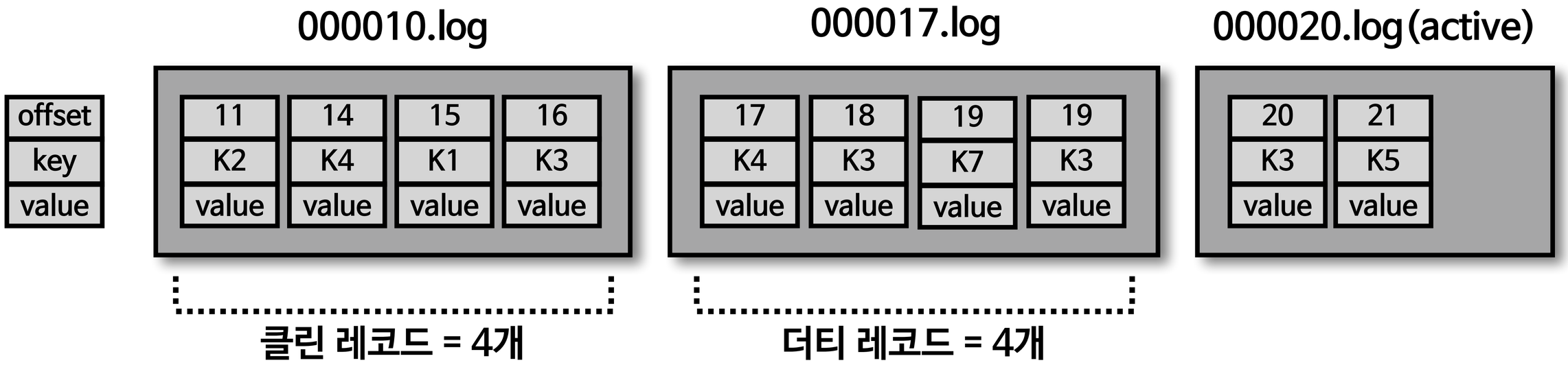
토픽 압축 정책은 일반적으로 생각하는 zip과 같은 압축(compression)과는 다른 개념이다. 여기서 압축이란 메시지 키 별로 해당 메시지 키의 레코드 중 오래된 데이터를 삭제하는 정책을 뜻한다. 그렇기 때문에 삭제(delete) 정책과 다르게 일부 레코드만 삭제가 될 수 있다. 압축은 액티브 세그먼트를 제외한 데이터가 대상이다.

테일/헤드 영역, 클린/더티 로그



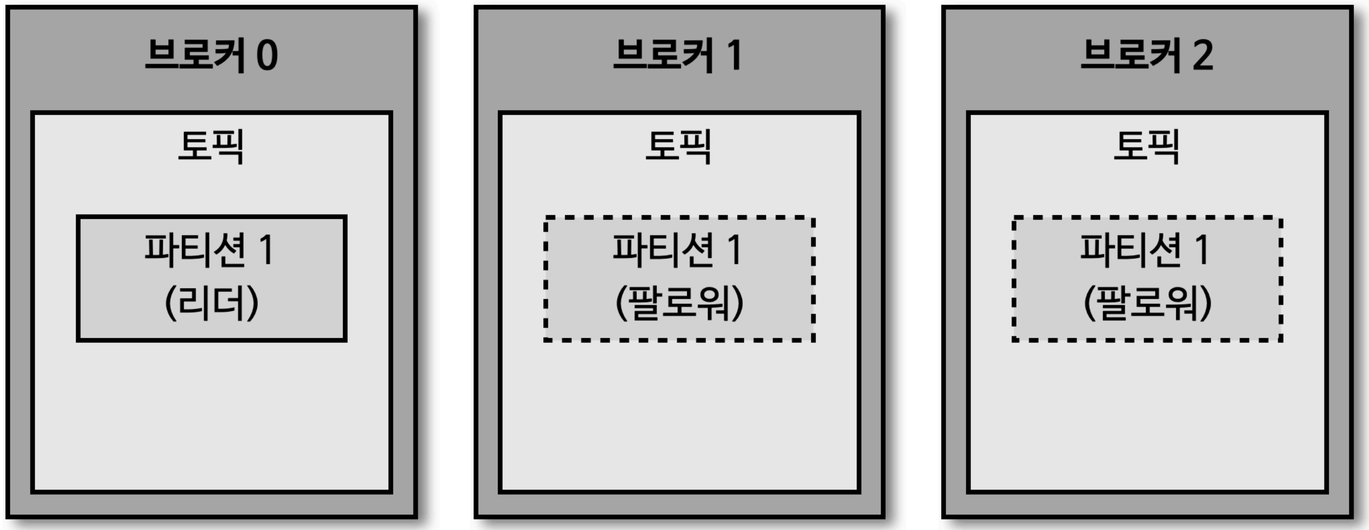
- 테일 영역 : 압축 정책에 의해 압축이 완료된 레코드들. 클린(clean)로그 라고도 부른다. 중복 메시지 키가 없다.
- 헤드 영역 : 압축 정책이 되기 전 레코드들. 더티(dirty)로그 라고도 부른다. 중복된 메시지 키가 있다.

min.cleanable.dirty.ratio



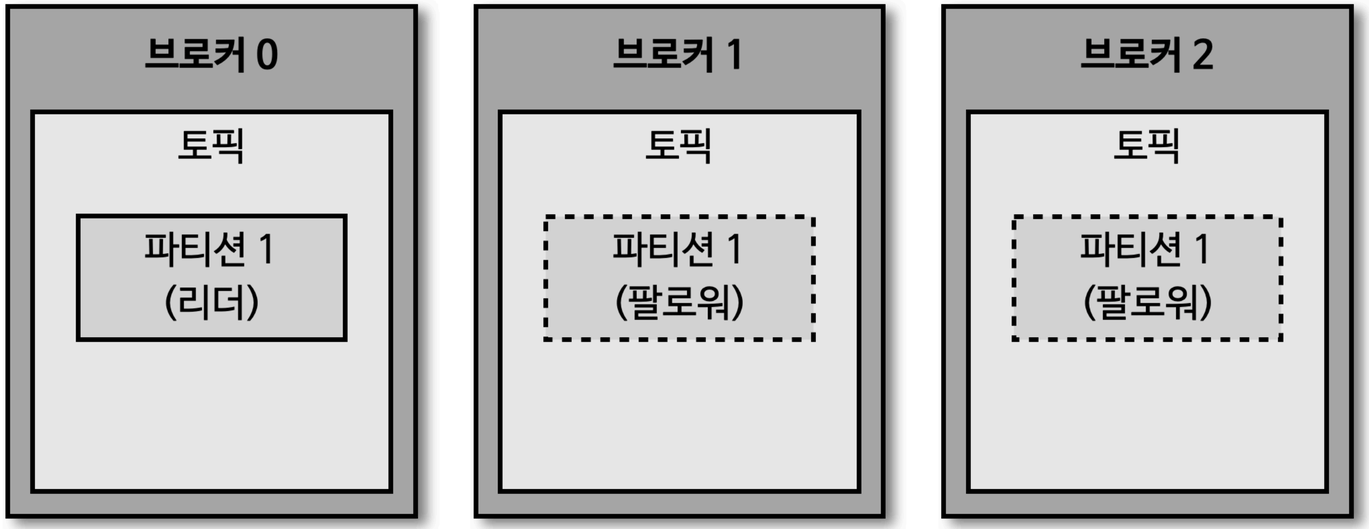
데이터의 압축 시작 시점은 min.cleanable.dirty.ratio 옵션값을 따른다. min.cleanable.dirty.ratio 옵션값은 액티브 세그먼트를 제외한 세그먼트에 남아 있는 테일 영역의 레코드 개수와 헤드 영역의 레코드 개수의 비율을 뜻한다. 예를 들어 0.5로 설정한다면 테일 영역의 레코드 개수가 헤드 영역의 레코드 개수와 동일할 경우 압축이 실행된다. 0.9와 같이 크게 설정하면 한번 압축을 할 때 많은 데이터가 줄어들므로 압축 효과가 좋다. 그러나 0.9 비율이 될 때 까지 용량을 차지하므로 용량 효율이 좋지 않다. 반면 0.1과 같이 작게 설정하면 압축이 자주 일어나서 가장 최신 데이터만 유지할 수 있지만 압축이 자주 발생하기 때문에 브로커에 부담을 줄 수 있다.

브로커의 역할 - 복제(Replication)



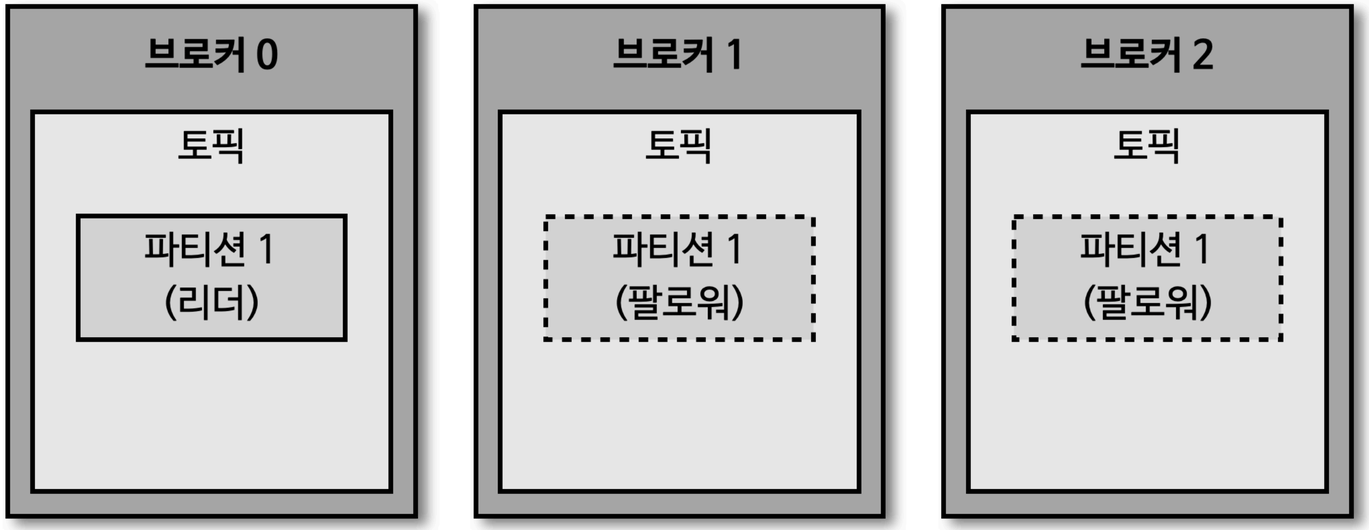
데이터 복제(replication)는 카프카를 장애 허용 시스템(fault tolerant system)으로 동작하도록 하는 원동력이다. 복제의 이유는 클러스터로 묶인 브로커 중 일부에 장애가 발생하더라도 데이터를 유실하지 않고 안전하게 사용하기 위함이다. 카프카의 데이터 복제는 파티션 단위로 이루어진다. 토픽을 생성할 때 파티션의 복제 개수 (replication factor)도 같이 설정되는데 직접 옵션을 선택하지 않으면 브로커에 설정된 옵션 값을 따라간다. 복제 개수의 최솟값은 1(복제 없음)이고 최댓값은 브로커 개수만큼 설정하여 사용할 수 있다.

브로커의 역할 - 복제(Replication)



복제된 파티션은 리더(leader)와 팔로워 (follower)로 구성된다. 프로듀서 또는 컨슈머와 직접 통신하는 파티션을 리더, 나머지 복제 데이터를 가지고 있는 파티션을 팔로워라고 부른다. 통상적으로 리더와 팔로워라고 부르지만 이 책에서는 파티션임을 명확히 하기 위해 파티션의 리더를 '리더 파티션', 파티션의 팔로워를 '팔로워 파티션'이라고 지칭한다. 팔로워 파티션들은 리더 파티션의 오프셋을 확인하여 현재 자신이 가지고 있는 오프셋과 차이가 나는 경우 리더 파티션으로 부터 데이터를 가져와서 자신의 파티션에 저장하는데, 이 과정을 '복제'라고 부른다.

브로커의 역할 - 복제(Replication)

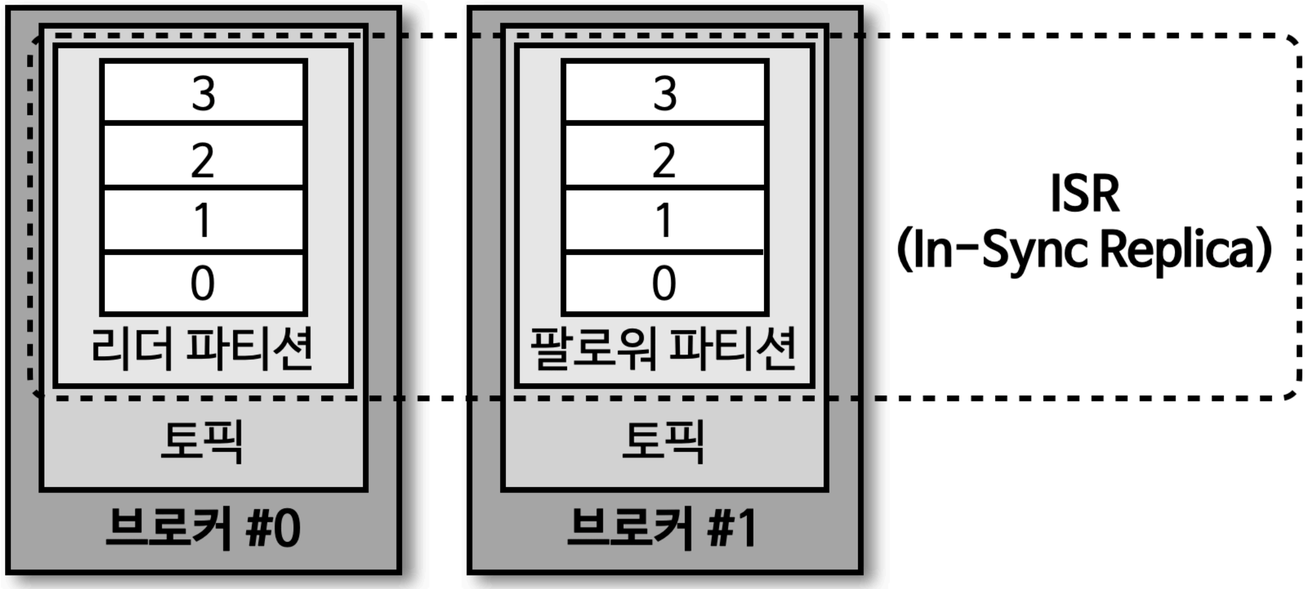


파티션 복제로 인해 나머지 브로커에도 파티션의 데이터가 복제되므로 복제 개수만큼의 저장 용량이 증가한다는 단점이 있다. 그러나 **복제를 통해 데이터를 안전하게 사용할수 있다는 강력한 장점때문에 카프카를 운영할 때 2 이상의 복제 개수를 정하는 것이 중요하다.** 카프카 브로커가 설치된 기업용 서버는 개인용 컴퓨터와 비교가 안될정도로 안정성이 좋지만 서버는 해커로 인한 침입,디스크 오류,네트워크 연결 장애등의 이유로 언제든 장애가 발생할수있다.

브로커에 장애가 발생한 경우

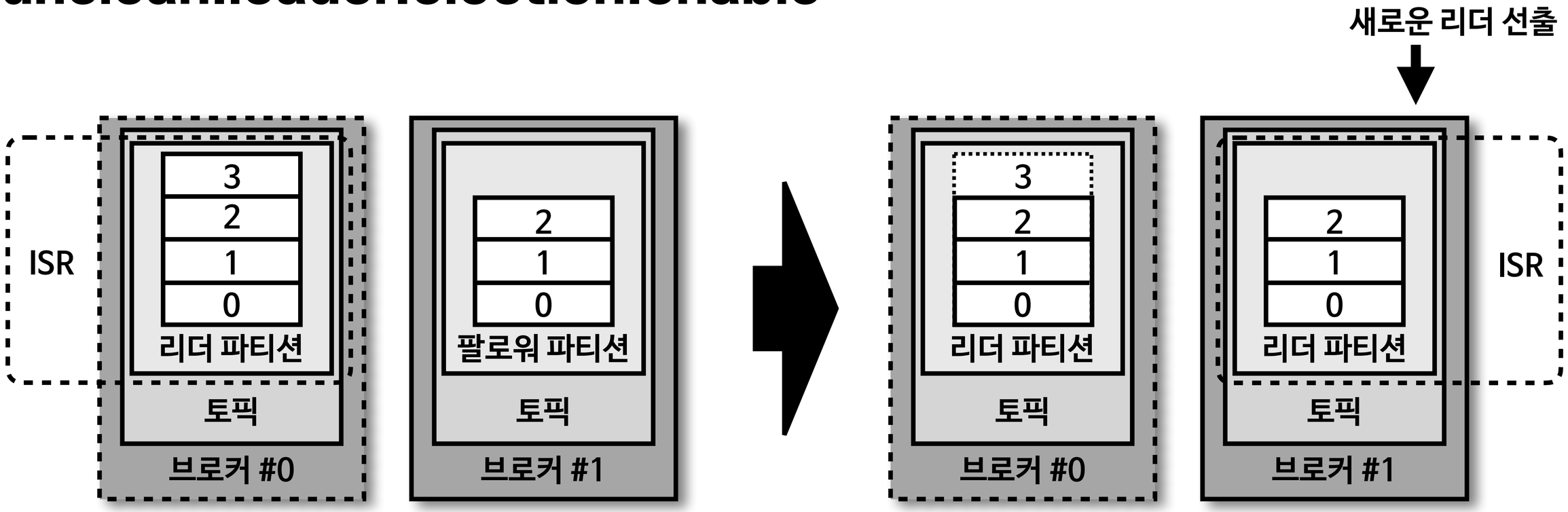
브로커가 다운되면 해당 브로커에 있는 리더파티션은 사용할수 없기 때문에 팔로워파티션 중 하나가 리더 파티션 지위를 넘겨받는다. 이를 통해 데이터가 유실되지 않고 컨슈머나 프로듀서와 데이터를 주고받도록 동작할 수 있다. 운영 시에는 데이터 종류마다 다른 복제 개수를 설정하고 상황에 따라서는 토픽마다 복제 개수를 다르게 설정하여 운영하기도 한다. 데이터가 일부 유실되어도 무관하고 데이터 처리 속도가 중요하다면 1 또는 2로 설정한다. 금융 정보와 같이 유실이 일어나면 안되는 데이터의 경우 복제개수를 3으로 설정하기도 한다.

ISR(In-Sync-Replicas)



ISR은 리더 파티션과 팔로워 파티션이 모두 싱크가 된 상태를 뜻한다. 복제 개수가 2인 토픽을 가정해 보자. 이 토픽에는 리더 파티션 1개와 팔로워 파티션이 1개가 존재할 것이다. 리더 파티션에 0부터 3의 오프셋이 있다고 가정할 때, 팔로워 파티션에 동기화가 완료되려면 0부터 3 까지 오프셋이 존재해야 한다. 동기화가 완료됐다는 의미는 리더 파티션의 모든 데이터가 팔로워 파티션에 복제된 상태를 말하기 때문이다.

unclean.leader.election.enable

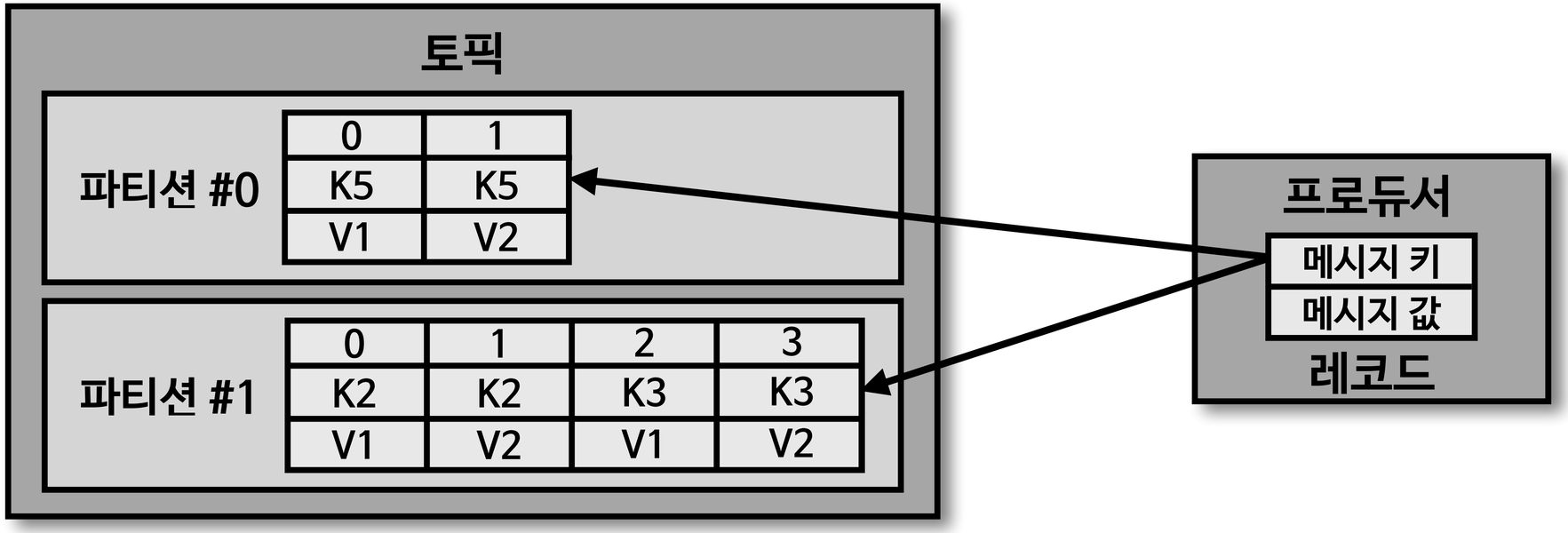


리더 파티션의 데이터를 모두 복제하지 못한 상태이고, 이렇게 싱크가 되지 않은 팔로워 파티션이 리더 파티션으로 선출되면 데이터가 유실될 수 있다. 유실이 발생하더라도 서비스를 중단하지 않고 지속적으로 토픽을 사용하고 싶다면 ISR이 아닌 팔로워 파티션을 리더로 선출하도록 설정할 수 있다.

- unclean.leader.election.enable=true : 유실을 감수함. 복제가 안된 팔로워 파티션을 리더로 승급.
- unclean.leader.election.enable=false : 유실을 감수하지 않음. 해당 브로커가 복구될 때까지 중단.

토픽, 파티션, 레코드

토픽과 파티션

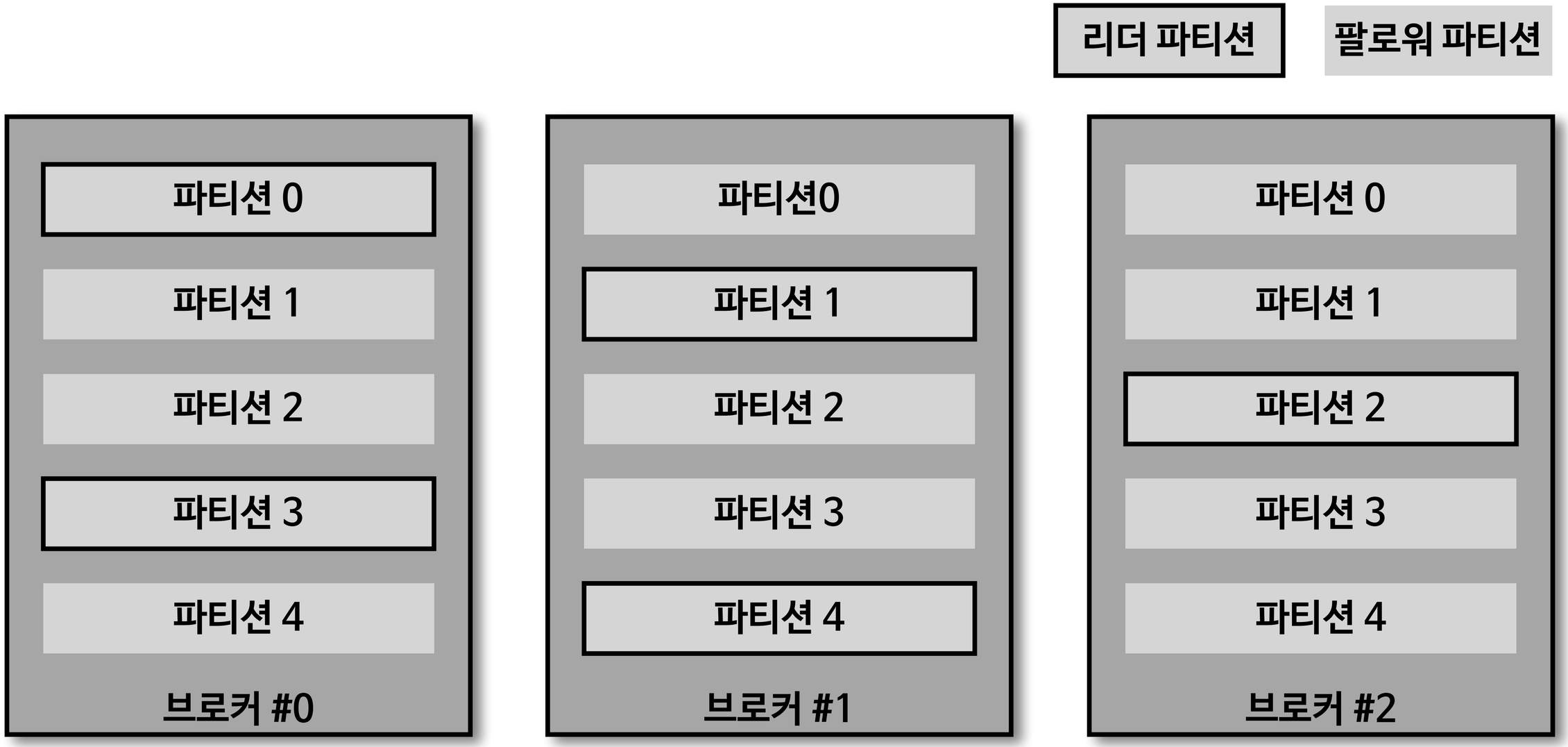


토픽은 카프카에서 데이터를 구분하기 위해 사용하는 단위이다. 토픽은 1개 이상의 파티션을 소유하고 있다. 파티션에는 프로듀서가 보낸 데이터들이 들어가 저장되는데 이 데이터를 '레코드(record)'라고 부른다. 파티션은 자료구조에서 접하는 큐(queue)와 비슷한 구조라고 생각하면 쉽다. First-in-first-out(FIFO) 구조와 같이 먼저 들어간 레코드는 컨슈머가 먼저 가져가게 된다. 다만, 일반적인 자료구조로 사용되는 큐는 데이터를 가져가면(pop) 삭제하지만 카프카에서는 삭제 하지 않는다. 파티션의 레코드는 컨슈머가 가져가는 것과 별개로 관리된다. 이러한 특징 때문에 토픽의 레코드는 다양한 목적을 가진 여러 컨슈머 그룹들이 토픽의 데이터를 여러번 가져갈수있다.

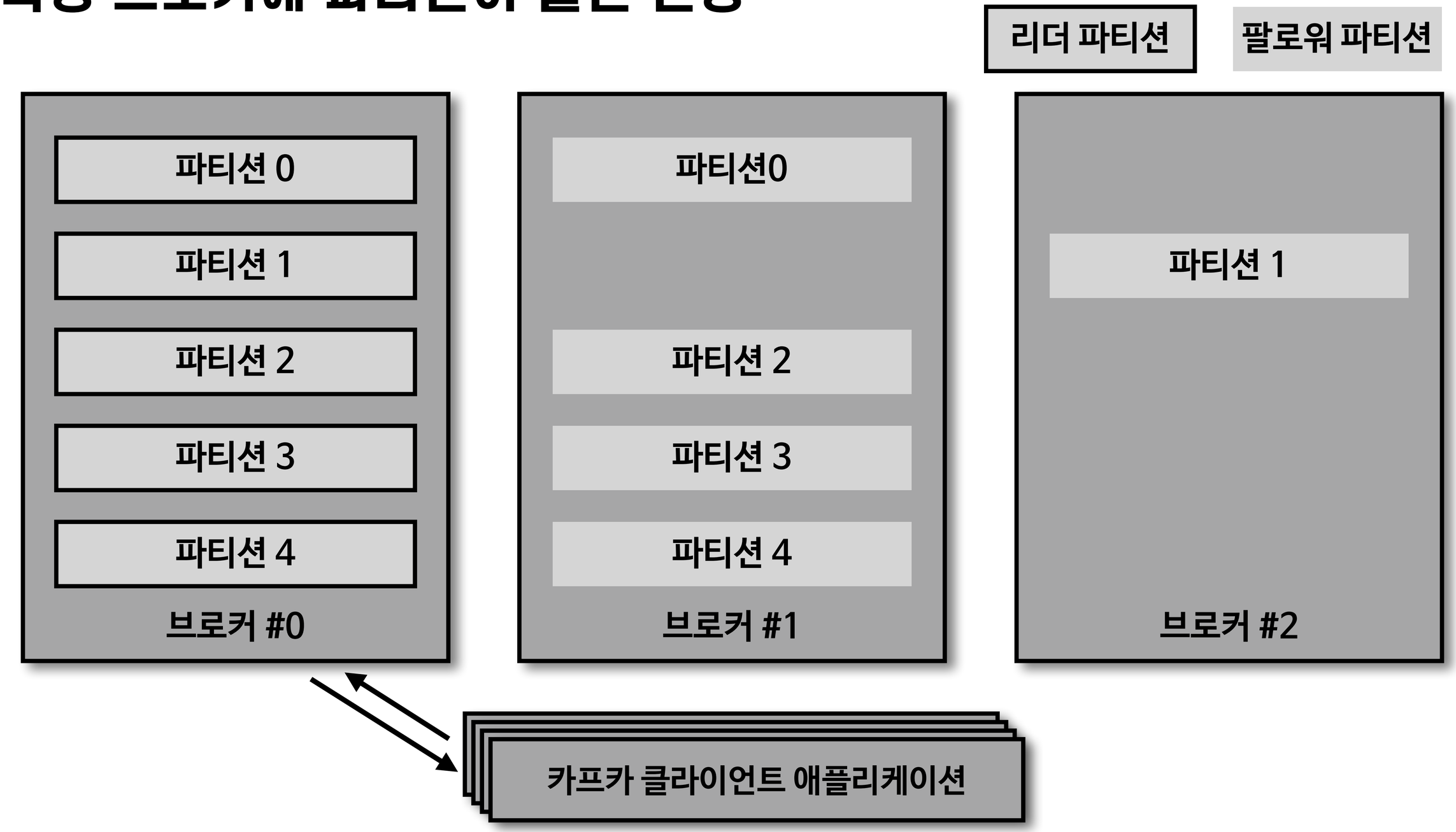
토픽 생성시 파티션이 배치되는 방법

파티션이 5개인 토픽을 생성했을 경우 그림과 같이 0번 브로커부터 시작하여 round-robin 방식으로 리더 파티션들이 생성된다. 카프카 클라이언트는 리더 파티션이 있는 브로커와 통신하여 데이터를 주고 받으므로 여러 브로커에 골고루 네트워크 통신을 하게 된다. 이를 통해, 데이터가 특정 서버(여기서는 브로커)와 통신이 집중되는(hot spot) 현상을 막고 선형 확장(linear scale out)을 하여 데이터가 많아지더라도 자연스럽게 대응할 수 있게 된다.

토픽 생성시 파티션이 배치되는 방법



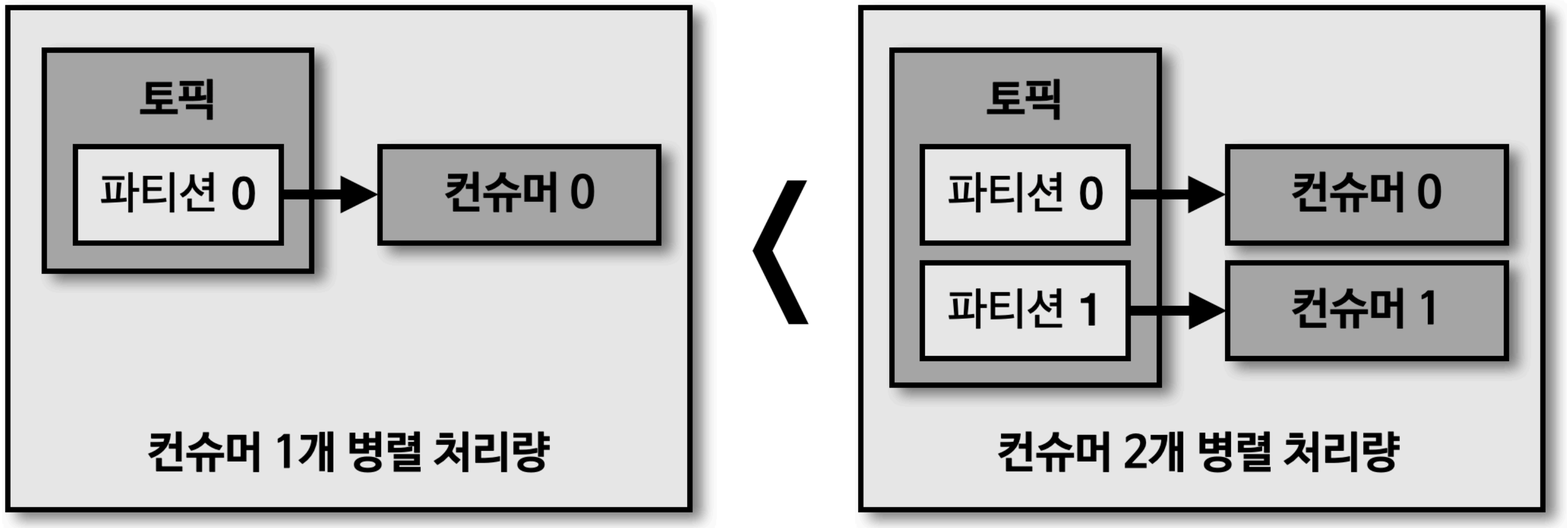
특정 브로커에 파티션이 쏠린 현상



특정 브로커에 파티션이 쏠린 현상

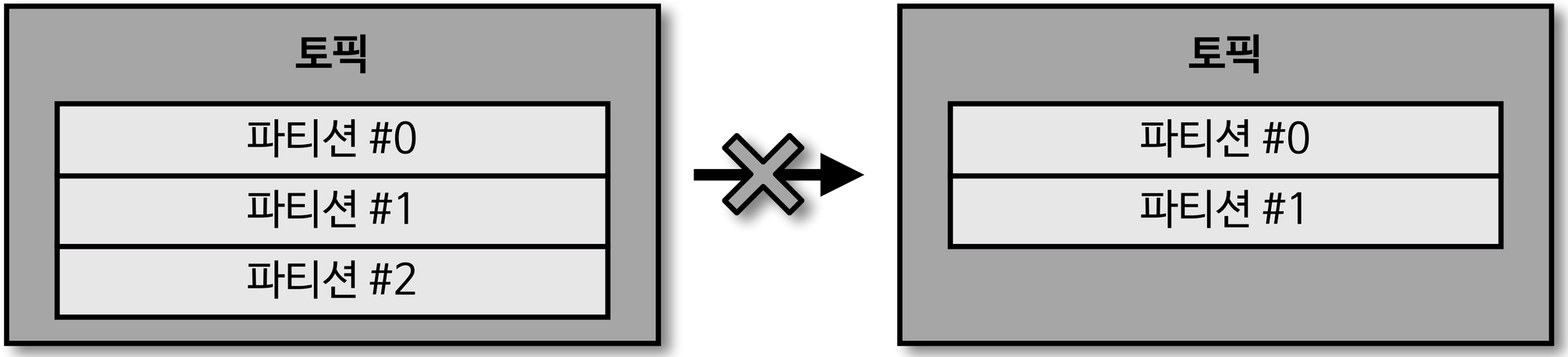
특정 브로커에 파티션이 몰리는 경우에는 kafka-reassign-partitions.sh 명령으로 파티션을 재분배할 수 있다.

파티션 개수와 컨슈머 개수의 처리량



파티션은 카프카의 병렬처리의 핵심으로써 그룹으로 묶인 컨슈머들이 레코드를 병렬로 처리 할 수 있도록 매칭된다. 컨슈머의 처리량이 한정된 상황에서 많은 레코드를 병렬로 처리하는 가장 좋은 방법은 컨슈머의 개수를 늘려 스케일 아웃하는 것이다. 컨슈머 개수를 늘림과 동시에 파티션 개수도 늘리면 처리량이 증가하는 효과를 볼 수있다.

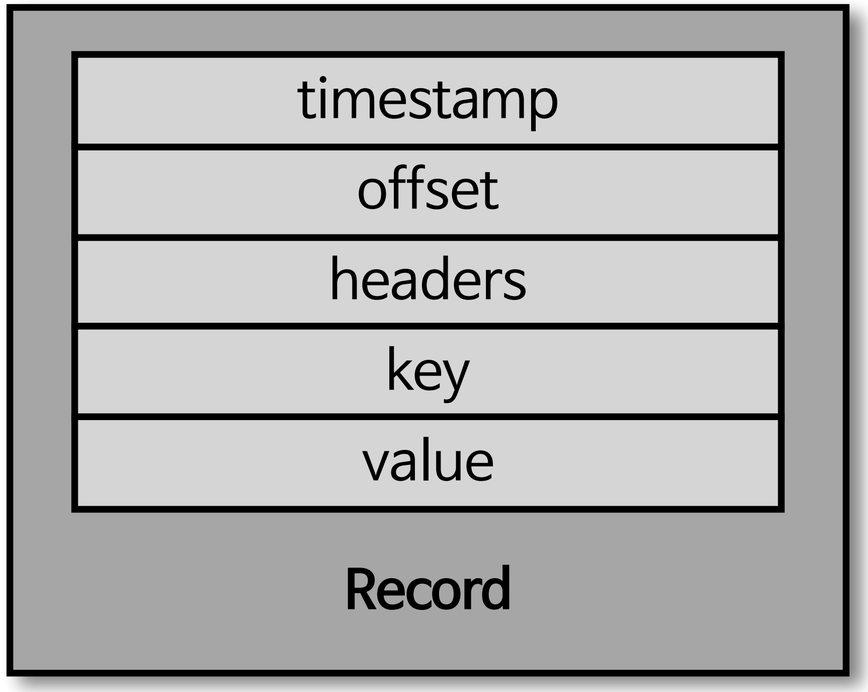
파티션 개수를 줄이는 것은 불가능



카프카에서 파티션 개수를 줄이는 것은 지원하지 않는다. 그러므로 파티션을 늘리는 작업을 할 때는 신중히 파티션 개수를 정해야 한다. 한번 늘리면 줄이는 것은 불가능하기 때문에 토픽을 삭제하고 재생성하는 방법 외에는 없기 때문이다. 카프카에서는 파티션의 데이터를 세그먼트로 저장하고 있으며 만에 하나 지원을 한다고 하더라도 여러 브로커에 저장된 데이터를 취합하고 정렬해야하는 복잡한 과정을 거쳐야 하기 때문에 클러스터에 큰 영향이 가게 된다. KIP-694에서 파티션 개수를 줄이는 것을 논의했지만 더 이상 진행되지 않고 있다.

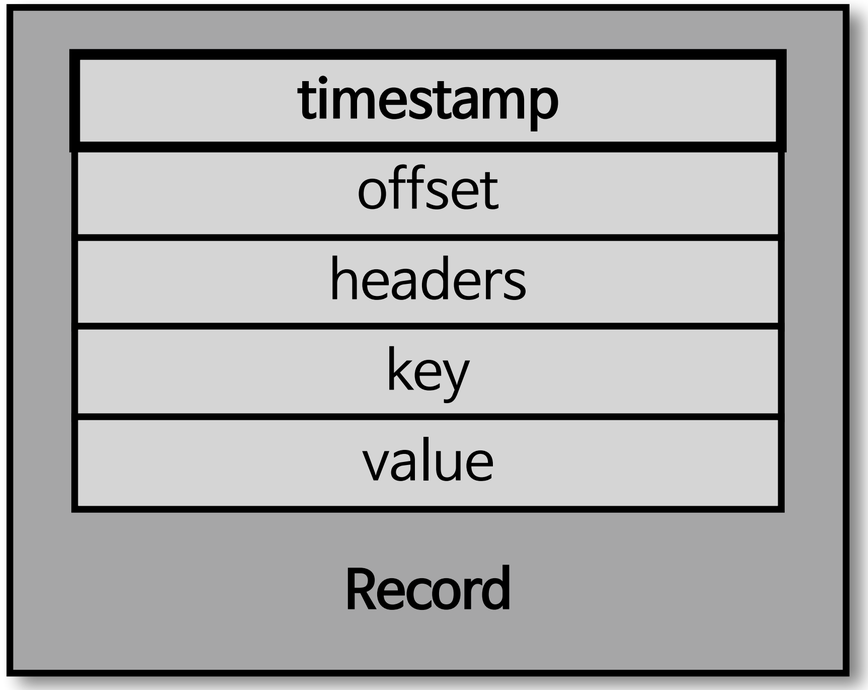
레코드 상세히 살펴보기

레코드



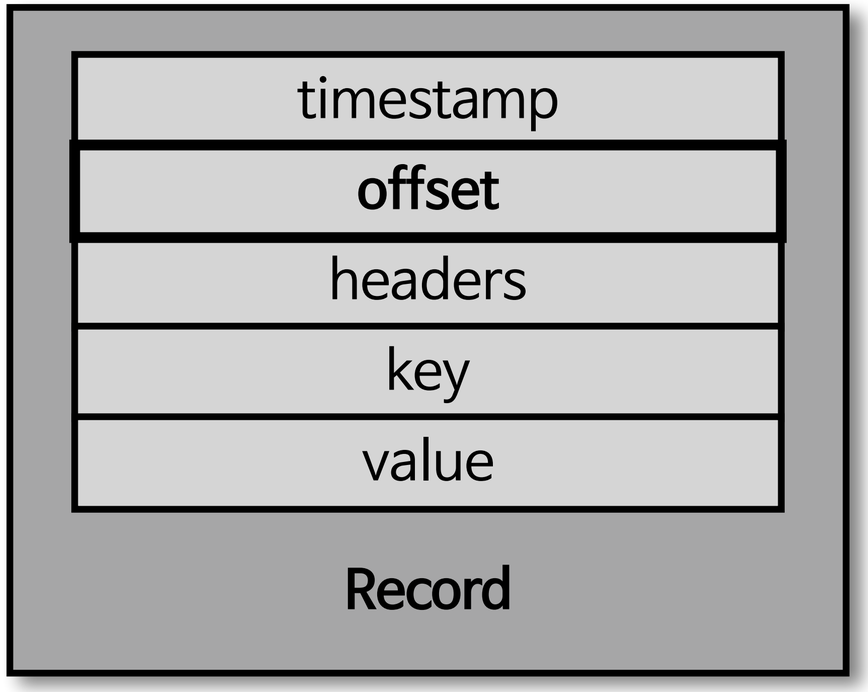
레코드는 타임스탬프, 헤더, 메시지 키, 메시지 값, 오프셋으로 구성되어 있다. 프로듀서가 생성한 레코드가 브로커로 전송되면 오프셋과 타임스탬프가 지정되어 저장된다. 브로커에 한번 적재된 레코드는 수정할 수 없고 로그 리텐션 기간 또는 용량에 따라서만 삭제된다.

레코드-타임스탬프



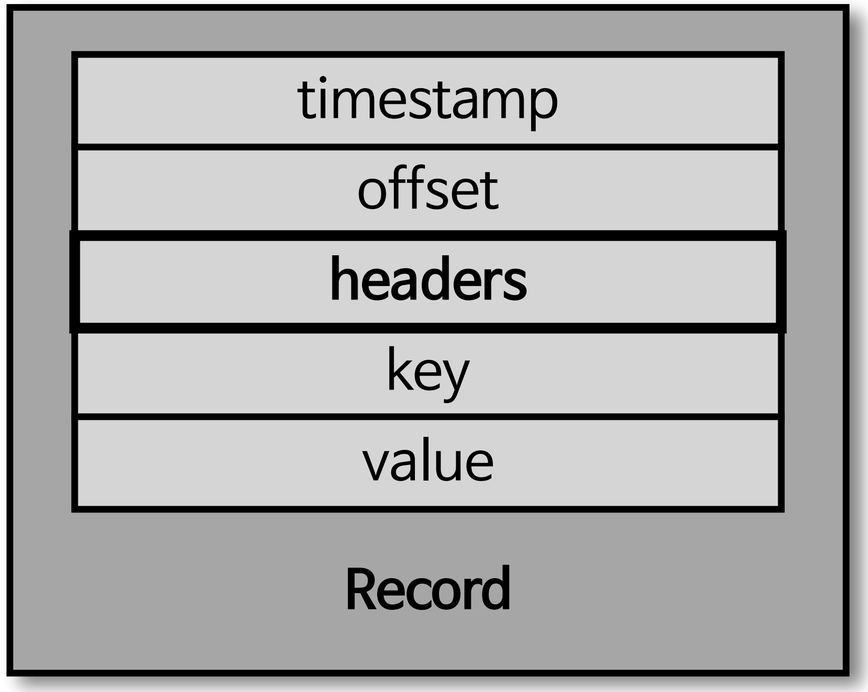
레코드의 타임스탬프는 스트림 프로세싱에서 활용하기 위한 시간을 저장하는 용도로 사용된다. 카프카 0.10.0.0 이후 버전부터 추가된 타임스탬프는 Unix timestamp가 포함되며 프로듀서에서 따로 설정하지 않으면 기본값으로 ProducerRecord 생성 시간(CreateTime)이 들어간다. 또는 브로커 적재 시간(LogAppendTime)으로 설정할 수도 있다. 해당 옵션은 토픽 단위로 설정 가능하며 message.timestamp.type 을 사용한다.

레코드-오프셋



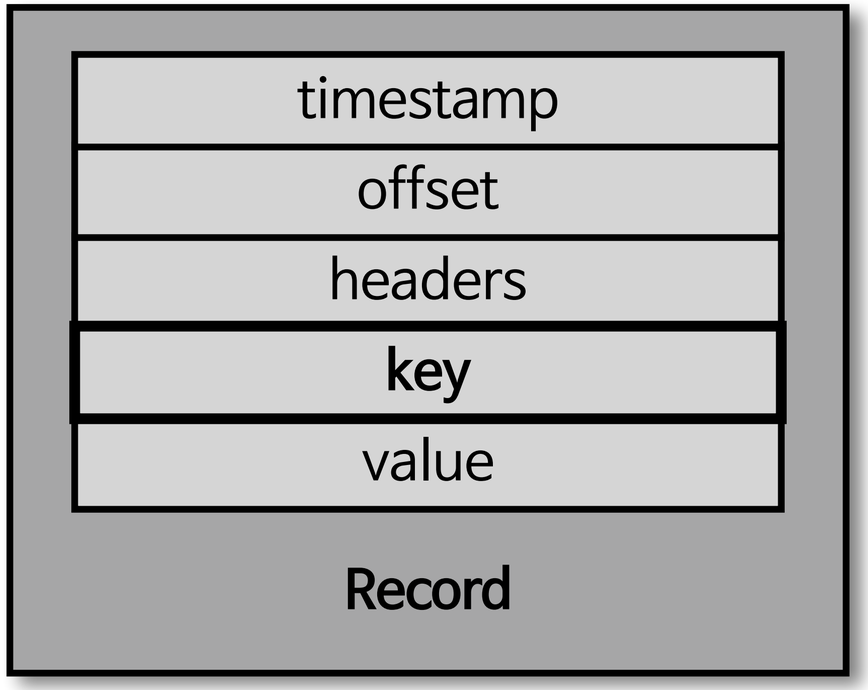
레코드의 오프셋은 프로듀서가 생성한 레코드에는 존재하지 않는다. 프로듀서가 전송한 레코드가 브로커에 적재될 때 오프셋이 지정된다. 오프셋은 0부터 시작되고 1씩 증가한다. 컨슈머는 오프셋을 기반으로 처리가 완료된 데이터와 앞으로 처리해야할 데이터를 구분한다. 각 메시지는 파티션별로 고유한 오프셋을 가지므로 컨슈머에서 중복 처리를 방지하기 위한 목적으로도 사용한다.

레코드-헤더



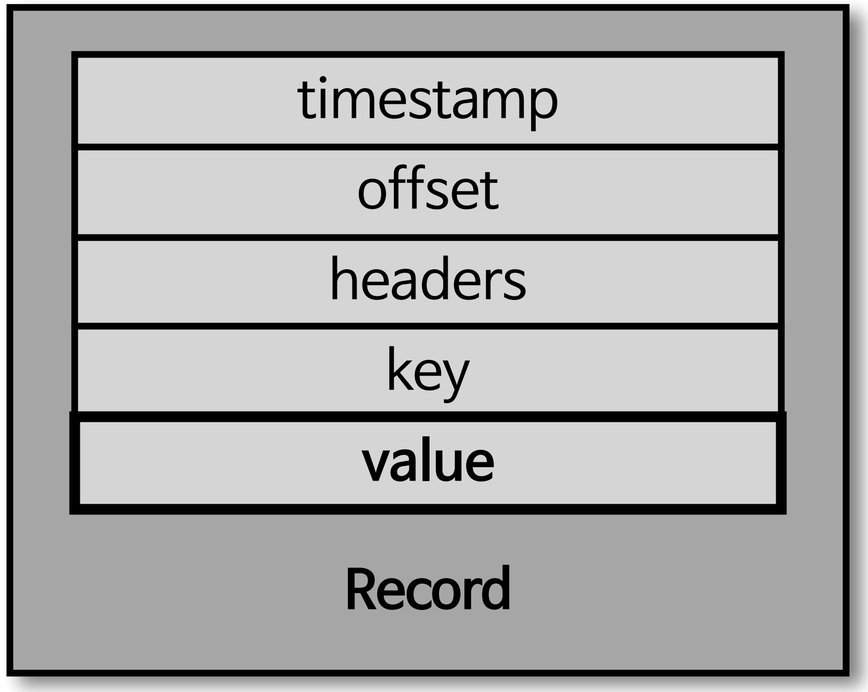
레코드의 헤더는 0.11부터 제공된 기능이다. key/value 데이터를 추가할 수 있으며 레코드의 스키마 버전이나 포맷과 같이 데이터 프로세싱에 참고할만한 정보를 담아서 사용할 수 있다.

레코드-메시지 키



메시지 키는 처리하고자 하는 메시지 값의 분류하기 위한 용도로 사용되며, 이를 파티셔닝이라고 부른다. 파티셔닝에 사용하는 메시지 키는 파티셔너(Partitioner)에 따라 토픽의 파티션 번호가 정해진다. 메시지 키는 필수 값이 아니며, 지정하지 않으면 null로 설정된다. 메시지 키가 null인 레코드는 특정 토픽의 파티션에 라운드 로빈으로 전달된다. null이 아닌 메시지 키는 해쉬값에 의해서 특정 파티션에 매핑되어 전달된다.(기본 파티셔너의 경우)

레코드-메시지 값



레코드의 메시지 값은 실질적으로 처리할 데이터가 담기는 공간이다. 메시지 값의 포맷은 제네릭으로 사용자에 의해 지정된다. Float, Byte[], String 등 다양한 형태로 지정 가능하며 필요에 따라 사용자 지정 포맷으로 직렬화/역직렬화 클래스를 만들어 사용할 수도 있다. 브로커에 저장된 레코드의 메시지 값은 어떤 포맷으로 직렬화되어 저장되었는지 알 수 없기 때문에 컨슈머는 미리 역직렬화 포맷을 알고 있어야 한다.

유지보수하기 좋은 토픽 이름 정하기

토픽 이름 제약 조건

- 빈 문자열 토픽 이름은 지원하지 않는다.
- 토픽 이름은 마침표 하나(.) 또는 마침표 둘(..)로 생성될 수 없다.
- 토픽 이름의 길이는 249자 미만으로 생성되어야한다.
- 토픽 이름은 영어 대, 소문자와 숫자 0부터 9 그리고 마침표(.), 언더바(__), 하이픈(-) 조합으로 생성할 수 있다. 이외의 문자열이 포함된 토픽 이름은 생성 불가하다.
- 카프카 내부 로직 관리 목적으로 사용되는 2개 토픽(__consumer_offsets, __transaction_state)과 동일한 이름으로 생성 불가능하다.
- 카프카 내부적으로 사용하는 로직 때문에 토픽 이름에 마침표(.)와 언더바(_)가 동시에 들어가면 안된다. 생성은 할 수 있지만 사용시 이슈가 발생할 수 있기 때문에 마침표(.)와 언더바(_)가 들어간 토픽 이름을 사용하면 WARNING 메시지가 발생한다.

의미 있는 토픽 이름 작명 방법

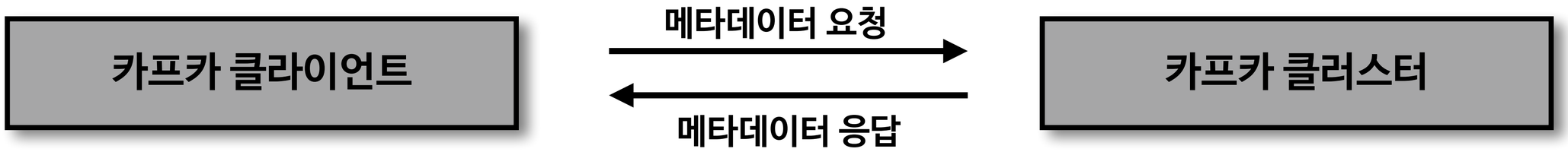
토픽 이름은 데이터의 얼굴이다. 토픽 이름을 모호하게 작성하면 유지보수 시 큰 어려움을 겪을 수 있다. 예를 들어, test-20210204, abcd, bigdata, test와 같은 토픽 이름은 어떤 용도로 누가 사용하고 있는지, 어떻게 만들어졌는지 알 수 없으므로 지양해야 한다. 중요한 것은 토픽 이름에 대한 규칙을 사전에 정의하고 구성원들이 그 규칙을 잘 따르는 것이다. 아무리 규칙을 정해도 따르지 않으면 예측하지 못한 방향으로 토픽 이름이 생성될 것이 고 이것들은 기술 부채(technical debt)로 남아 카프카 운영자의 머리를 아프게 할 것이다. 게다가 카프카는 토픽 이름 변경을 지원하지 않으므로 이름을 변경하기 위해서는 삭제 후 다시 생성하는 것 외에는 방법이 없다. 따라서 다소 까다롭더라도 카프카 클러스터 사용자에게 토픽 생성에 대한 규칙을 인지시키고 규칙에 따르도록 하는 것이 중요하다. 규칙을 따르지 않 은 토픽 이름의 경우 주기적으로 확인하고 검토하여 사용 여부를 판단해야 한다. 만일 단발성 으로 생성된 토픽이라면 삭제 처리하고, 실제로 사용을 위한 토픽이라면 삭제 후 신규로 토픽 을 만드는 것을 권장한다.

토픽 작명의 템플릿과 예시

<환경>.<팀-명>.<애플리케이션-명>.<메시지-타입>
  예시) prd.marketing-team.sms-platform.json
<프로젝트-명>.<서비스-명>.<환경>.<이벤트-명>
  예시) commerce.payment.prd.notification
<환경>.<서비스-명>.<JIRA-번호>.<메시지-타입>
  예시) dev.email-sender.jira-1234.email-vo-custom
<카프카-클러스터-명>.<환경>.<서비스-명>.<메시지-타입>
  예시) aws-kafka.live.marketing-platform.json

카프카 브로커와 클라이언트가 통신하는 방법

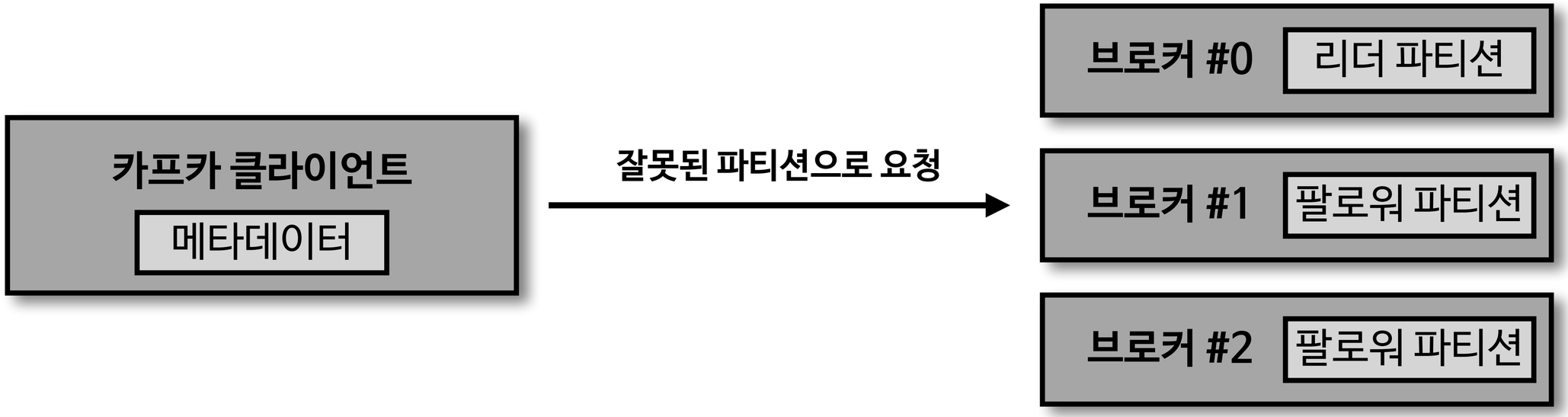
클라이언트 메타데이터



카프카 클라이언트는 통신하고자 하는 리더 파티션의 위치를 알기 위해 데이터를 주고(프로듀서) 받기(컨슈머) 전에 메타데이터를 브로커로부터 전달받는다. 메타데이터는 다음과 같은 옵션을 통해 리프래쉬된다.

카프카 프로듀서 메타데이터 옵션
- metadata.max.age.ms : 메타데이터를 강제로 리프래시하는 간격. 기본값은 5분.
- metadata.max.idle.ms : 프로듀서가 유휴상태일 경우 메타데이터를 캐시에 유지하는 기간. 예를 들어 프로듀서가 특정 토픽으로 데이터를 보낸 이후 지정한 시간이 지나고 나면 강제로 메타데이터를 리프래시. 기본값은 5분.

클라이언트 메타데이터가 이슈가 발생한 경우



카프카 클라이언트는 반드시 리더 파티션과 통신해야 한다. 만약 메타데이터가 현재의 파티션 상태에 맞게 리프래시되지 않은 상태에서 잘못된 브로커로 데이터를 요청하면 LEADER_NOT_AVAILABLE 익셉션이 발생한다. 이 에러는 클라이언트(프로듀서 또는 컨슈머)가 데이터를 요청한 브로커에 리더 파티션이 없는 경우 나타나며 대부분의 경우 메타데이터 리프래시 이슈로 발생한다. 이 에러가 자주 발생한다면 메타데이터 리프래시 간격을 확인하고 클라이언트가 정상적인 메타데이터를 가지고 있는지 확인해야 한다.

퀴즈

정리

- 카프카 클러스터는 브로커들의 묶음. 클러스터는 1대 이상의 브로커로 이루어져 있다.
- 카프카 브로커를 실행하기 위해서는 주키퍼가 필수(3.0.0 부터는 제외 가능)
- 프로듀서에서 전송된 데이터는 디렉토리에 저장
- 브로커는 복제, 컨트롤러, 데이터 삭제, 오프셋 저장, 코디네이터 역할을 수행
- 1개의 주키퍼에는 여러 카프카 클러스터를 운영할 수 있음
- 토픽은 1개 이상의 파티션으로 구성되어 있음
- 파티션은 FIFO 구조이지만 컨슈머가 데이터를 가져가더라도 바로 삭제되지 않음
- 컨슈머의 처리량을 늘리기 위해 파티션 개수를 늘릴 수 있음

퀴즈

1) 컨슈머가 가져간 데이터는 파티션에서 삭제된다 (O/X)
2) 카프카에서 데이터 복제의 단위는 토픽이다 (O/X)
3) 파티션은 1개의 리더 파티션과 1개 이상의 팔로워 파티션으로 이루어져 있다 (O/X)
4) 프로듀서와 컨슈머는 팔로워 파티션과 통신하여 데이터 처리량을 늘릴 수 있다 (O/X)
5) 레코드는 타임스탬프, 메시지 키, 메시지 값, 오프셋으로 이루어져 있다 (O/X)
6) 다음 파티션에서 압축(compact)이 실행된다면, 남은 모든 오프셋의 배열은? (주관식, 숫자 나열)