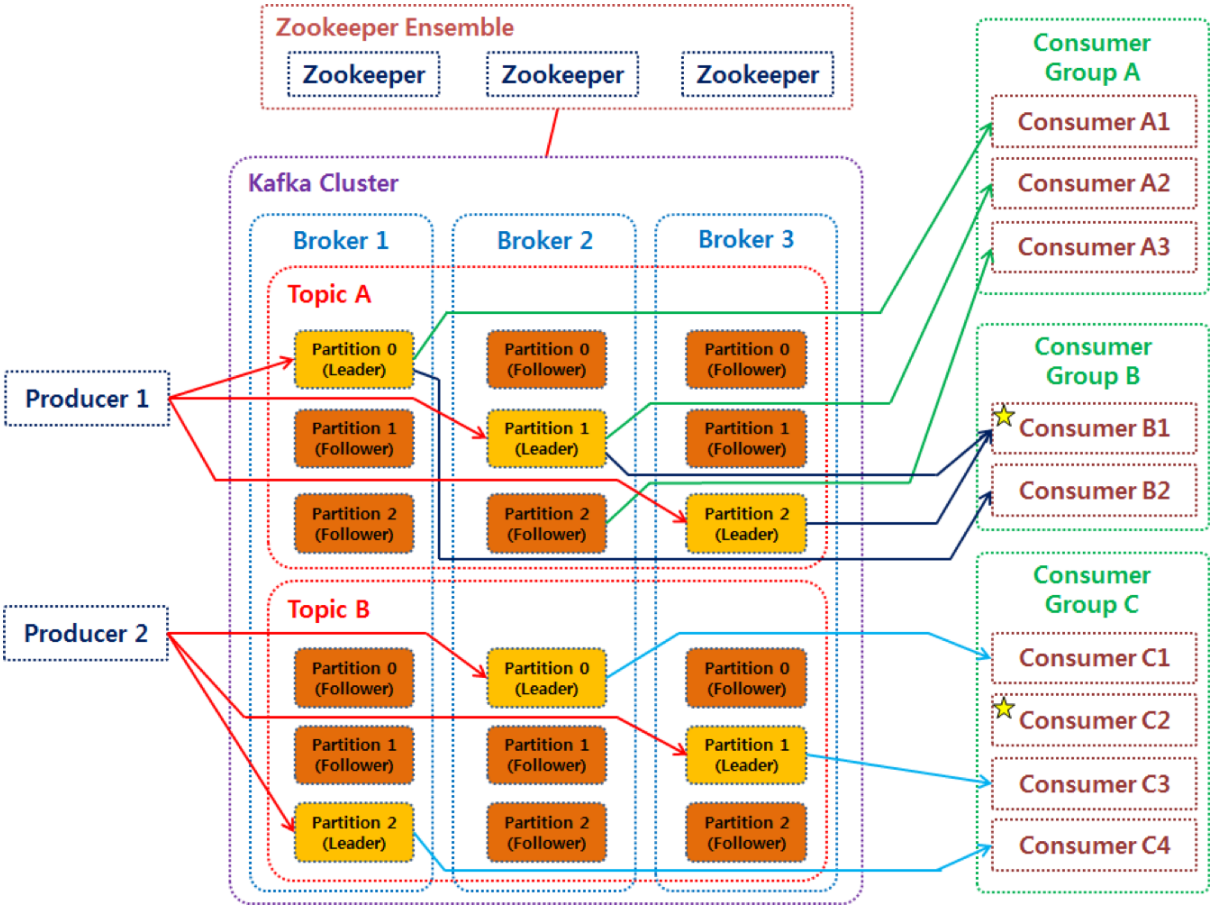
05장 프로듀서의 내부 동작 원리와 구현

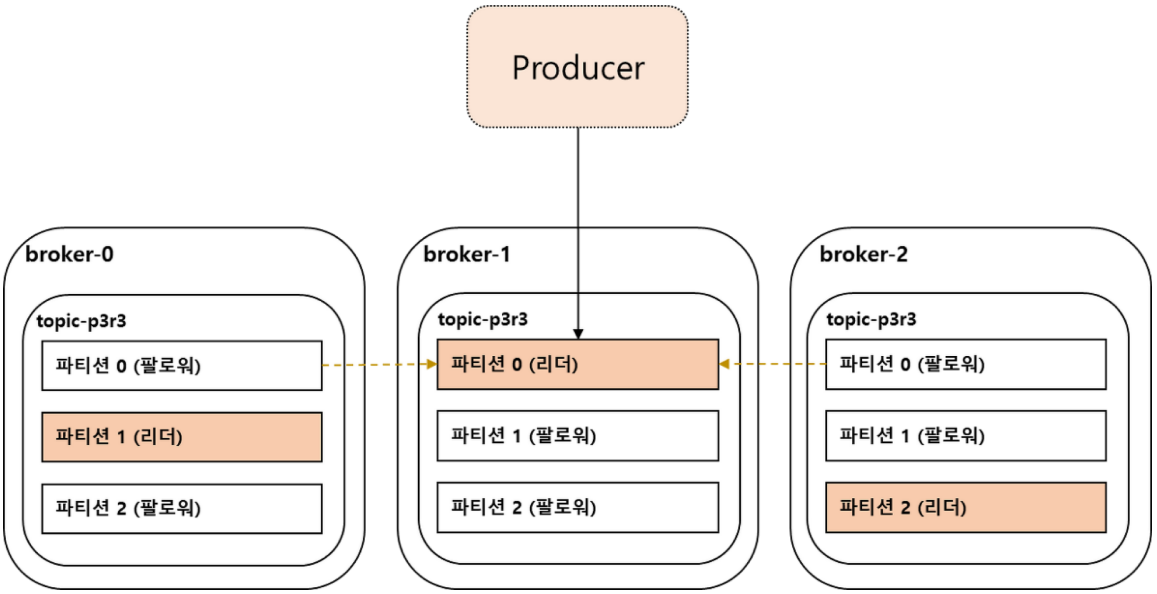
- 카프카 구조



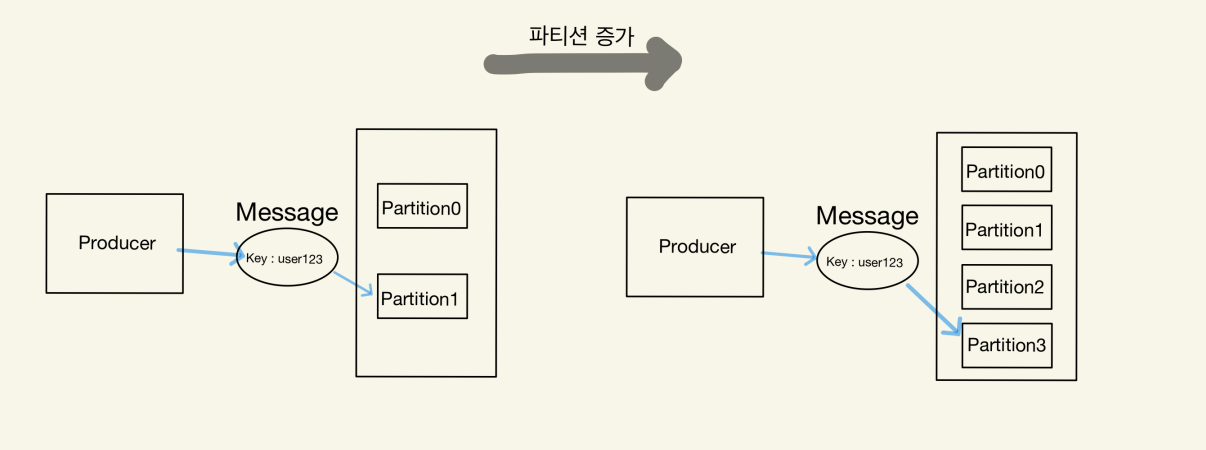
- 프로듀서의 역할은 메시지들을 카프카의 토픽으로 전송하는 것
- 프로듀서가 보내는 메세지들은 파티셔너를 거쳐서 최종적으로 카프카로 전송

5.1 파티셔너

- 프로듀서가 토픽으로 메시지를 보낼 때 **해당 토픽의 어느 파티션으로 메시지를 보내야할지 결정**할 때 사용합니다.



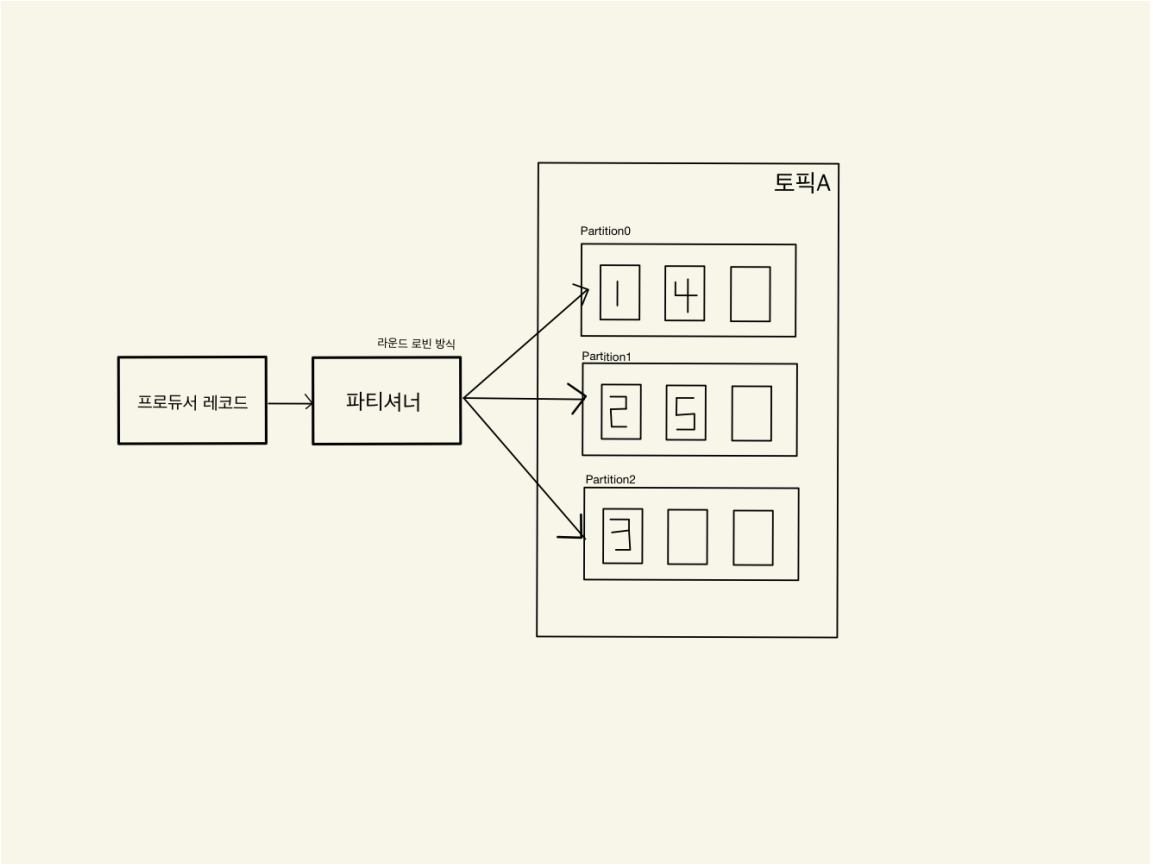
- 프로듀서가 파티션을 결정하는 알고리즘 : 메시지의 키를 해시처리해서 파티션을 구하는 방식을 사용합니다.
- 키값에 따라 파티션이 매핑되고 동일한 키값을 가진 메시지는 동일한 파티션으로 전송합니다.
- 파티션 수가 변경되면 키와 파티션이 매핑된 해시 테이블도 변경됩니다.
- 따라서, 파티션의 수는 변경하지 않는것이 권장됩니다.



5.1.1 라운드 로빈 전략 (아파치 카프카 2.4~)

- 키값을 지정하지 않으면 라운드 로빈 알고리즘을 사용해 프로듀서는 레코드를 파티션으로 랜덤 전송합니다.
- 파티셔너를 거친 메시지들은 바로 전송되지 않고 프로듀서의 버퍼메모리 영역에서 대기합니다. 이 과정에서
  비효율적인 모습이 발생할 수 있습니다.

> 시나리오
>
> - 파티션 수 : 3
> - 배치전송을 위한 최소 레코드 수 : 3
> - 메시지 전송시 키값을 사용하지 않음 : 라운드로빈 방식으로 파티션에 배정

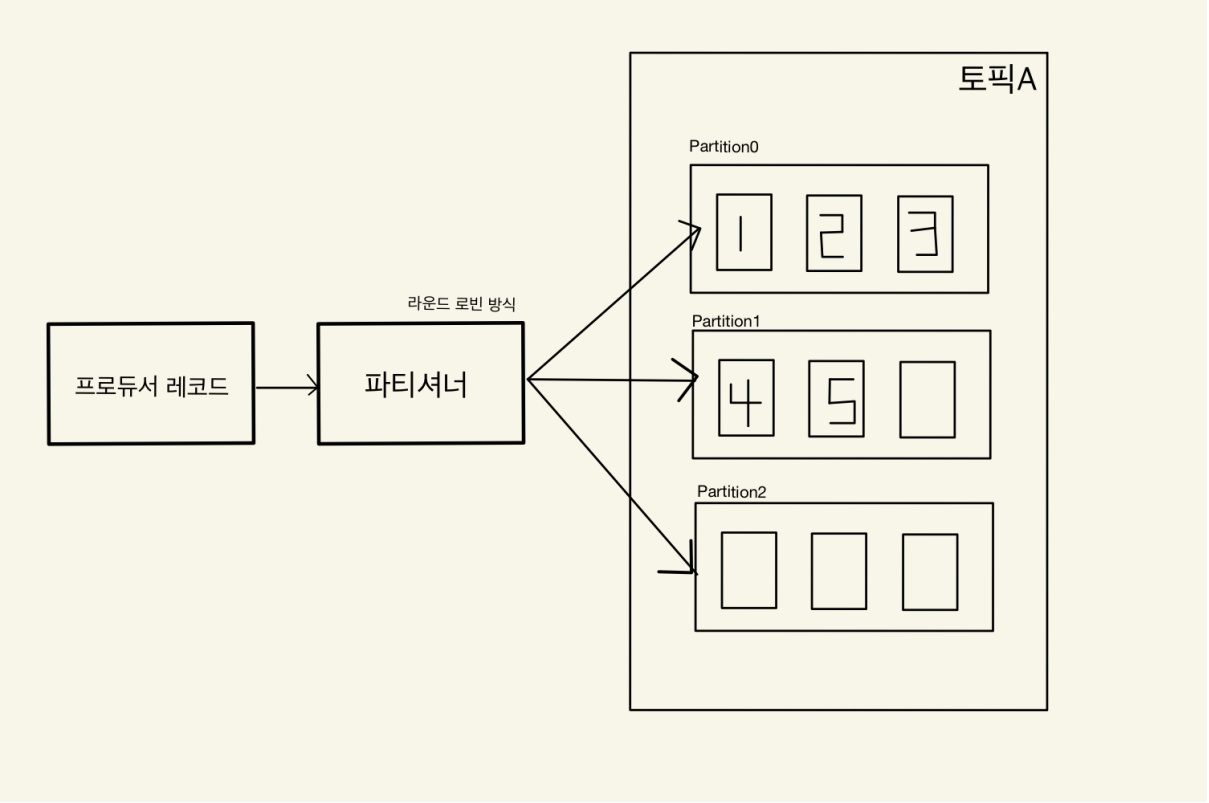


> 동작 순서
> 1. 첫번째 메시지가 파티셔너를 통해 파티션 0에 할당받습니다.
> 2. 이 후 4개의 메시지들은 라운드 로빈 전략에 의해 파티션에 하나씩 순차적으로 할당 됩니다.
> 3. 하지만 각 파티션들은 배치전송을 위한 최소 메시지 수가 충족되지 않아서 전송되지 않습니다.

- 지연시간 증가 문제를 해결하기 위해, 관리자는 프로듀서 옵션을 조정해서 특정 시간 초과시 카프카로 전송하도록 설정 할 수 있습니다. → 하지만, 배치와 압축의 효과를 얻지 못합니다.

5.1.2 스티키 파티셔닝 전략

- 하나의 파티션에 레코드 수를 먼저 채워서 카프카로 배치 전송하는 전략
- 각 파티션은 배치 전송을 위한 최소메시지를 채우는 순간 즉시 전송



5.2 프로듀서의 배치

- 프로듀서는 처리량을 높이기 위해서 배치 전송을 권장합니다.
- 따라서, 메시지 전송전 프로듀서 버퍼 메모리에 토픽의 파티션 별로 메시지를 잠시 보관합니다.
- 배치 전송을 위한 옵션

> - buffer.memory(기본값 32KB) : 메시지를 담아두는 프로듀서의 버퍼 메모리 옵션
> - bath.size(기본값 16KB) : 배치전송을 위해 메시지를 묶는 단위
> - linger.ms(기본값0) : 배치 전송을 위해 버퍼 메모리에서 대기하는 최대 시간, 0으로 설정하면 배치 전송을 기다리지 않고 바로 전송

- 배치 전송은 불필요한 I/O를 줄일 수 있어서 효율적입니다.

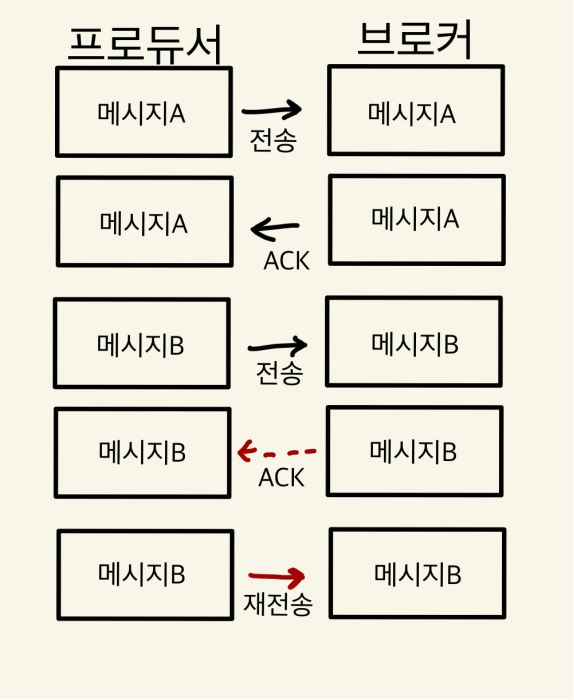
    - 1000개의 메시지 전송
        - 배치 전송 X : → 1000번의 요청과 응답
        - 배치 전송(100개) : 10번의 요청과 응답
- 목표에 따라서 배치 전송 옵션 설정이 필요합니다.
    - 지연없는 메시지 전송 : linger.ms을 낮게 설정합니다.
    - 처리량 높이기 : batch.size를 높게 설정 합니다.
- 주의할 점
    - buffer.memory 는 batch.size보다 충분히 커야합니다.
    - batch.size = 16KB, partition = 3 이라면, buffer.memory의 크기는 최소 48KB + a (전송실패시 재전송)이어야 합니다.
    - 압축기능을 같이 사용하면 더욱 효율적으로 사용 가능합니다.

5.3 중복 없는 전송(멱등성 전송)

- 메시지 시스템들의 메시지 전송 방식
    1. 적어도 한 번 전송(at-least-once)
    2. 최대 한 번 전송(most-once)
    3. 정확히 한 번 전송(exactly-once)

5.3.1 적어도 한번 전송

- 카프카의 기본 전송 방식



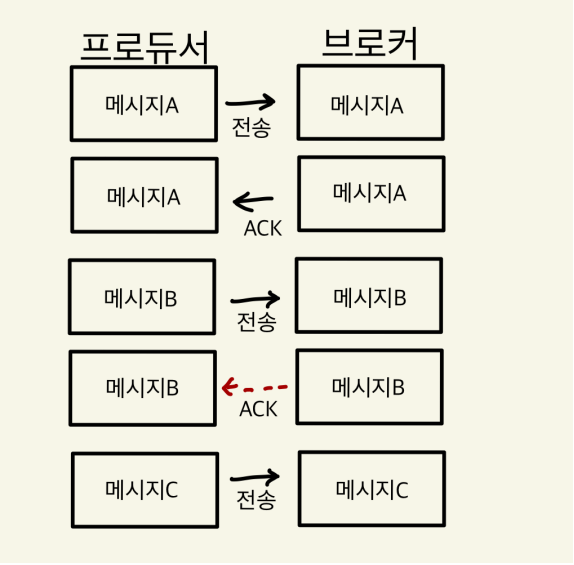
- 시나리오

1. 프로듀서가 브로커의 특정 토픽으로 메시지A 전송
2. 브로커는 메시지A를 기록하고, 잘받았다는 ACK 응답
3. 프로듀서가 브로커의 특정 토픽으로 메시지B 전송요청
4. 네트워크 오류 또는 브로커 장애로 인해 프로듀서는 브로커로 부터 ACK 응답을 받지 못함
5. 메시지B 재전송 (해당 ACK를 받을 때 까지)

- 프로듀서는 브로커가 메시지B 저장까지 성공하고 ACK를 보내지 못한 경우인지, 브로커가 메시지B를 받지 못한 경우인지 알 수 없습니다.
- 만약, 브로커가 메시지B를 받아서 기록하고 ACK만 프로듀서로 못보냈을 경우, 브로커는 메시지B가 재전송될때 메시지를 중복 저장하게 되는 문제가 발생합니다.

5.3.2 최대 한 번 전송

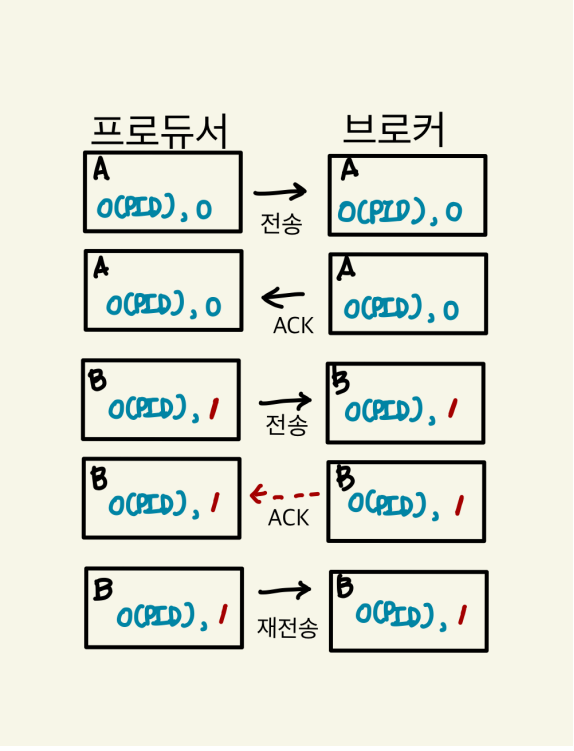
- 메시지 손실을 감안 하더라도 중복전송을 하지 않기 위한 방법
- 대량 로그 수집, IoT같은 환경에서 사용



- 시나리오

1. 프로듀서가 브로커의 특정 토픽으로 메시지A 전송
2. 브로커는 메시지A를 기록하고, 잘받았다는 ACK 프로듀서로 응답
3. 다음 메시지B 전송
4. 프로듀서는 브로커로부터 ACK를 응답받지 못함
5. 프로듀서는 브로커가 메시지B를 잘 받았다고 가정하고 메시지C 전송

5.3.3 중복 없이 전송



- 시나리오

1. 프로듀서가 브로커의 특정 토픽으로 메시지A를 전송. 이 때 PID(producer Id)와 메시지 번호 0을 헤더에 포함하여 전송
2. 브로커는 메시지A를 저장하고, PID와 메시지 번호를 메모리에 기록. 그리고 프로듀서로 ACK 응답
3. 프로듀서는 메시지 B 전송(PID=0, 메시지번호 =1)
4. 브로커는 메시지B를 저장하고(PID와 메시지 번호 기록). 그리고 프로듀서로 ACK를 응답하지 못함
5. 프로듀서는 메시지B를 재전송

- 시나리오만 보면, 적어도 한번 전송과 동일해 보이지만 프로듀서가 메시지B를 전송한 후 브로커의 동작에서 차이를 보입니다.
- 브로커는 PID와 메시지 번호를 저장하면서, 중복으로 메시지를 저장하지 않을 수 있습니다.
- 만약 프로듀서가 동일한 메시지를 여러번 보내도, 브로커는 메시지를 중복 저장하지 않습니다.
- 프로듀서가 보낸 메시지의 시퀀스번호(메시지 번호)가 브로커가 가지고 있는 메시지의 시퀀스번호보다 정확히 하나 큰 경우에만 브로커는 메시지를 저장합니다.
- PID와 메시지 번호는 브로커의 메모리에 저장되면서, 파티션의 리더가 변경되더라도 중복없는 메시지 전송을 보장할 수 있습니다.
- 중복없는 전송을 위한 프로듀서 설정

> - enable.idempotence(기본값 false) : 프로듀서가 중복없는 전송을 허용할지 결정 (true로 설정시 중복없는 전송)
> - max.in.flight.requests.per.connection(기본값 5) : ACK를 받지 않은 상태에서 하나의 커넥션에서 보낼 수 있는 최대 요청수 (1~5로 설정)
> - acks : all로 설정
> - retries(기본값 5) : ACK를 받지 못한 경우 재시도 수, 0보다 커야함

- 실습
  - producer.config

```
enable.idempotence=true
acks=all
max.in.flight.requests.per.connection=5
retries=5
```

```
kafka-console-producer --bootstrap-server kafka1:9091 --to
```

```
kafka-dump-log --deep-iteration --files 00000000000000000(
```

- PID, 시퀀스 번호(first, last), 마지막 오프셋 정보 확인 가능

5.4 정확히 한 번 전송

- 중복없는 전송은 정확히 한번 전송의 일부 기능
- 카프카에서 정확히 한 번 전송은 트랜잭션과 같은 전체적인 프로세스 처리를 의미합니다.
- 전체적인 프로세스를 관리하기 위해 카프카에는 정확히 한 번 처리를 담당하는 별도의 프로세스인 트랜잭션API가 존재합니다.

5.4.1 디자인

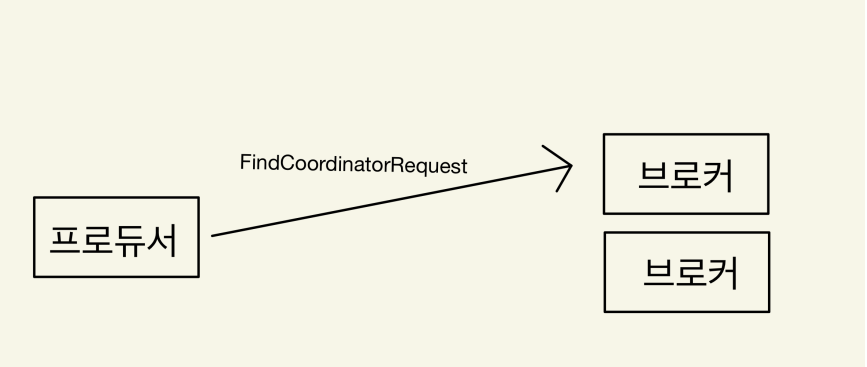
- 프로듀서가 보내는 메시지들은 원자적으로 실행되어 최종적으로 실패 또는 성공하게 됩니다.
- 구성
  - Transaction Coordinator : 프로듀서에 의해 전송된 메시지를 관리하고 현재 트랜잭션의 상태를 관리합니다. 주 역할은 Producer ID와 transactional.id를 매핑하고 해당 트랜잭션 전체를 관리하는 것입니다.
  - __transaction_state : transaction 관리를 위한 카프카 내부 토픽입니다. 관련된 로그를 기록하기 위한 토픽이며 프로듀서가 직접 기록하지는 않고, 프로듀서가 Transaction Coordinator에 알리고, Transaction Coordinator가 직접 로그를 기록하는 방식으로 관리됩니다.
  - Control message : 트랜잭션을 사용하면 메시지가 정상적으로 커밋된 것인지, 실패된 것인지 식별할 수 있어야 합니다. 카프카에서는 이를 식별하기 위한 정보로서 컨트롤 메시지 라는 것을 사용합니다. payload에는 message의 value는 포함되지 않고 애플리케이션에 노출되지 않습니다. 컨트롤 메시지는 오직 브로커와 클라이언트 통신에서만 사용됩니다.

5.4.2 프로듀서 설정 추가

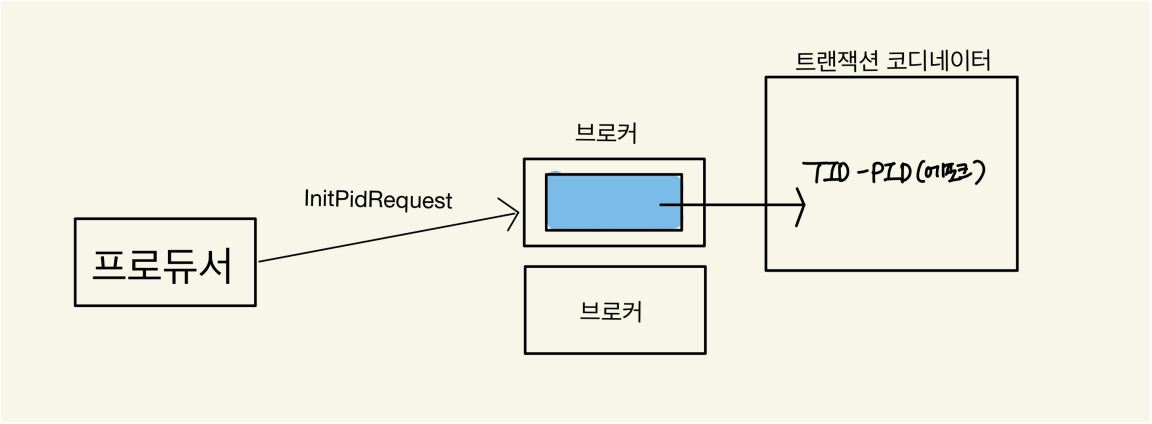
- TRANSACTION_ID_CONFIG 옵션 필요 , 프로듀서마다 고유한 ID 필요
  - 예시) 2개의 프로듀서가 있다면 2개의 다른 프로듀서 ID필요

5.4.3 단계별 동작

1. 트랜잭션 코디네이터 찾기



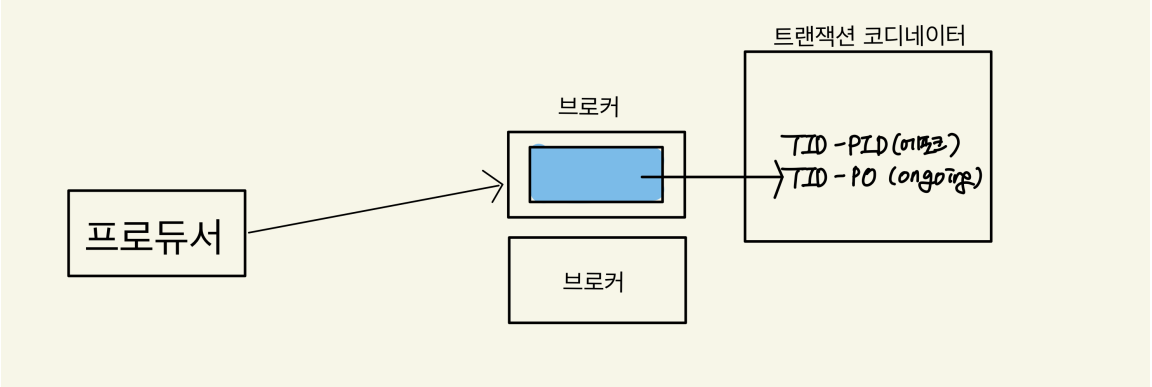
   - 트랜잭션 코디네이터는 브로커에 존재합니다. (존재하지 않을때는 새로 생성)
   - 프로듀서는 브로커에게 FindCoordinatorRequest를 보내서 트랜잭션 코디네이터의 위치를 찾습니다.
   - __transaction_state 토픽(내부토픽의)의 transactional.id를 기반으로 해시하여 파티션이 결정되는데, 이 파티션의 리더가 있는 브로커가 트랜잭션 코디네이터의 브로커로 최종 선정됩니다
   - transactional.id는 정확히 하나의 코디네이터만 갖고 있습니다.
2. 프로듀서 초기화



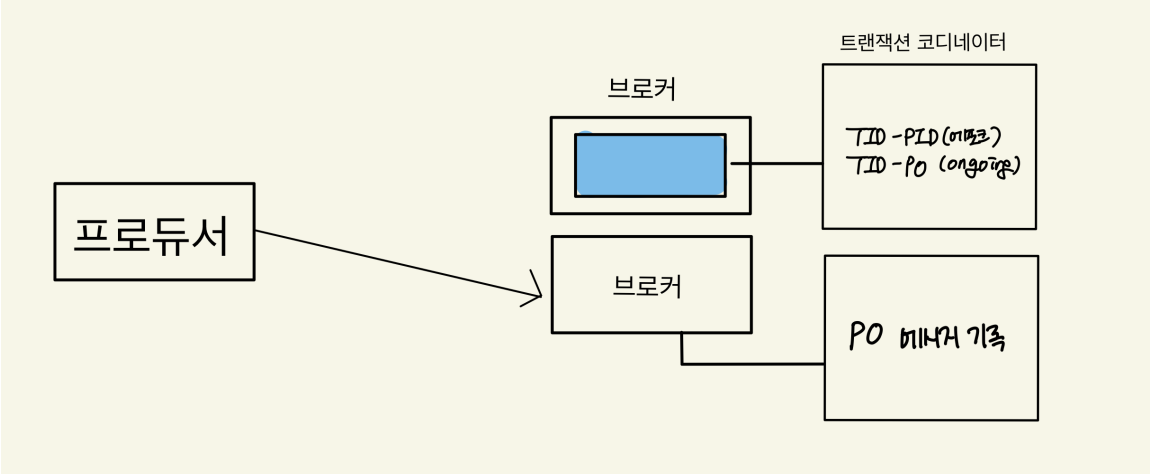
- `InitTransactions()` 메소드를 이용해 트랜잭션 전송을 위한 InitPidRequest를 트랜잭션 코디네이터로 보냅니다.

- 트랜잭션 코디네이터는 TID, PID를 매핑하고 해당 정보를 트랜잭션 로그에 기록합니다.
- 그런 다음 PID 에포크(epoch)를 한 단계 올리는 동작을 하게되고, PID 에포크가 올라감에 따라 이전의 동일한 PID와 이전 에포크에 대한 쓰기 요청은 무시하게 됩니다.

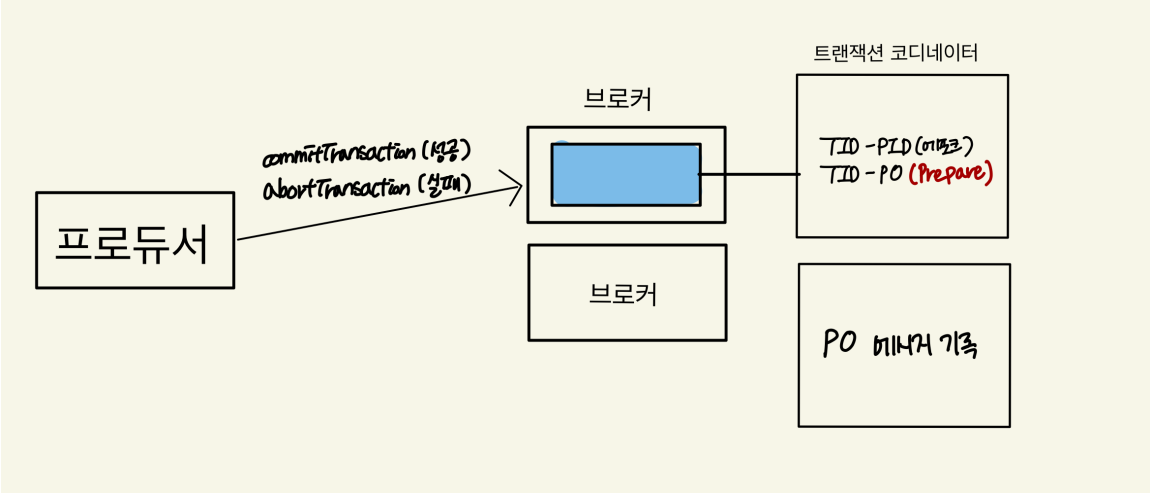
3. 트랜잭션 시작
    - `beginTransaction()` 메소드를 이용해 새로운 트랜잭션의 시작을 알림니다.
    - 트랜잭션 코디네이터 관점에서는 첫 번째 레코드가 전송될때까지 트랜잭션이 시작된 것은 아닙니다.
4. 트랜잭션 상태 추가



    - 프로듀서는 토픽 파티션 정보를 트랜잭션 코디네이터에게 전달하고, 트랜잭션 코디네이터는 기록합니다.
    - 트랜잭션 코디네이터에는 PID-Partition Index로 키를 만들어 status log를 기록합니다.
    - 최초 상태는 Ongoing 상태로 입력됩니다.
5. 메시지 전송
    - 다른 브로커에 위치한 message 전용 Leader Partition으로 메시지 전송합니다.
    - 해당 메시지에는 PID, 에포크, 시퀀스번호가 함께 전송됩니다.

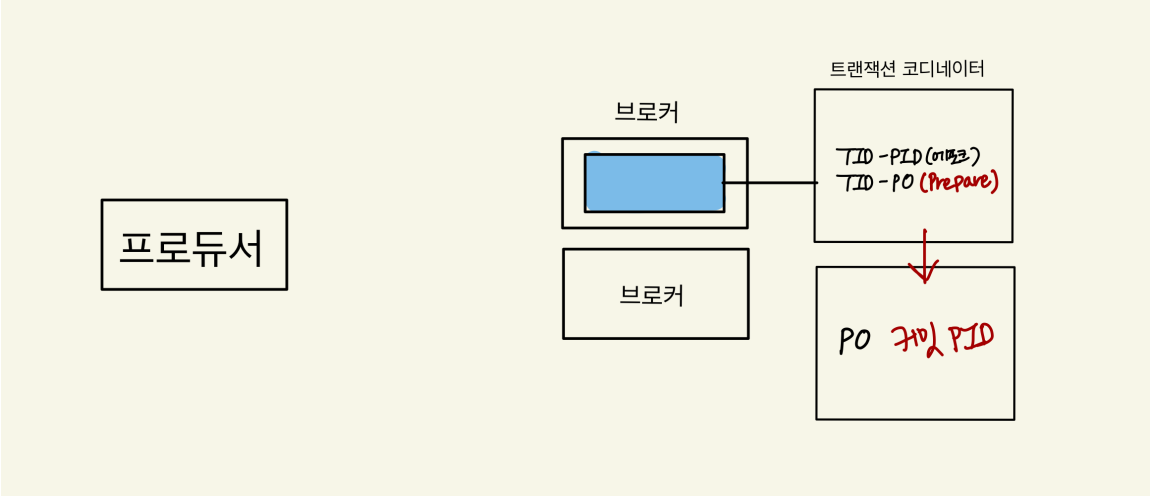


6. 트랜잭션 종료 요청



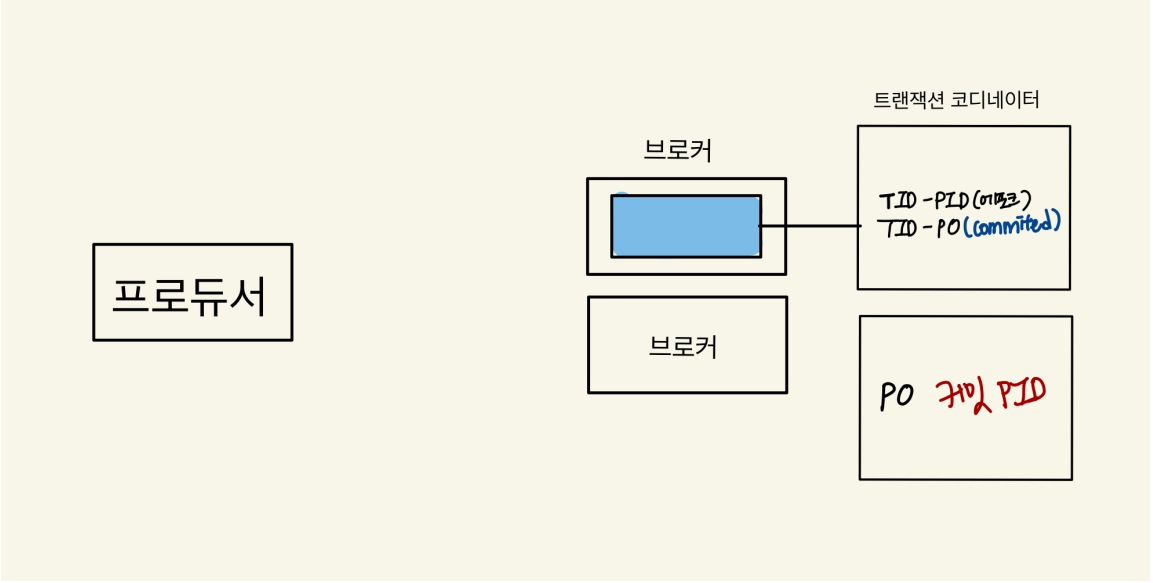
- 메시지 전송을 완료한 프로듀서는 `commitTransaction()` 혹은 `abortTransaction()` 메소드 중 하나를 반드시 호출해야합니다.
- 트랜잭션 코디네이터는 PrepareCommit 혹은 PrepareAbort를 트랜잭션 로그에 기록합니다.

7. 사용자 토픽에 표시 요청



- 트랜잭션 코디네이터는 메시지가 전송된 파티션에 컨트롤 메시지를 기록 합니다.
- 컨트롤 메시지도 메시지므로 오프셋을 증가 시킵니다. (오프셋: 0 → 2로 증가)

8. 트랜잭션 완료



- 트랜잭션 코디네이터는 완료됨(Committed)이라고 트랜잭션 로그에 기록 후, 프로듀서에 알립니다.