Kafka

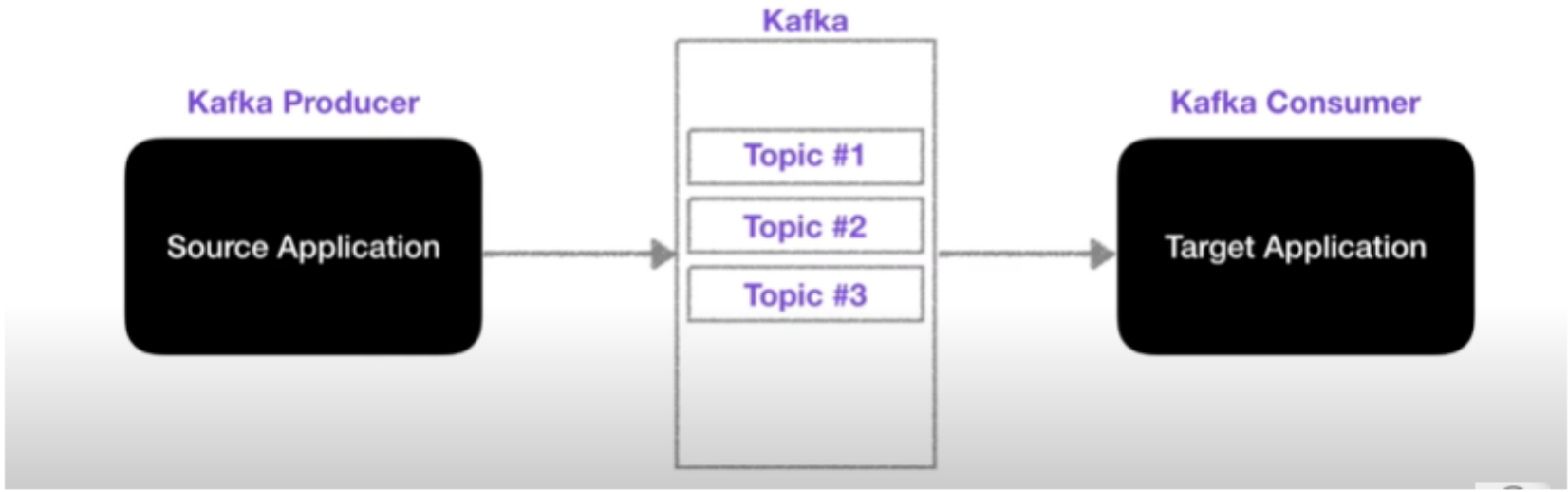
NClab 21st

Network Computing Lab.

Producer, Consumer

Producer와 Consumer

Kafka features



각종 데이터를 담는 topic이라는 개념이 있는데, 쉽게 생각해서 큐(queue) 라고 생각하면 됩니다.
큐에 데이터를 넣는 역할은 Producer가 하고, 큐에서 데이터를 가져오는 역할은 Consumer가 합니다.
Producer와 Consumer은 라이브러리로 돼있어서 어플리케이션에서 구현이 가능합니다.
오가는 데이터 포맷은 거의 제한이 없습니다.(json, tsv, avro etc... 여러 포맷을 지원)

Producer, Consumer

Producer 역할

- Topic에 해당하는 메시지를 생성
- 특정 Topic으로 데이터를 publish
- 처리 실패/재시도

Consumer 역할

- Topic의 partition으로부터 데이터 **polling(가져오기)**
- **Partition offset**(파티션에 있는 데이터의 번호) 위치 기록(commit) -> 어느 지점까지 데이터를 읽었는지 확인하여 **고가용성 보장**
- Consumer group을 통해 **병렬처리**(파티션 개수에 따라 컨슈머를 여러개 배치) -> 주의점: **Consumer 개수** <= **partition개수**

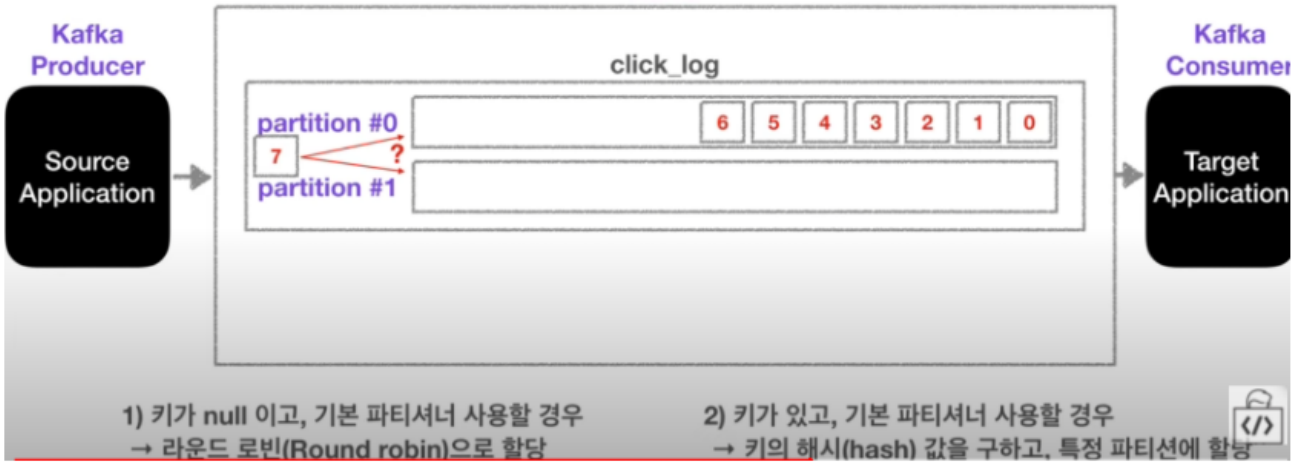
Topic, Partition

Kafka Topic

Topic은 파일시스템 폴더와 유사한 구조를 가집니다.
Topic의 이름은 목적에 따라 어떤 데이터를 담는지 명확하게 명시하여 편리하게 관리할 수 있습니다.

파티션

하나의 토픽은 여러개의 파티션으로 구성될 수 있으며 첫번째 파티션 번호는 0입니다. 하나의 파티션은 큐와 같이 데이터가 끝에서부터 차곡차곡 쌓이게 됩니다.
Kafka Consumer은 끝에서부터(가장오래된 순으로) 데이터를 가져오게 됩니다.
Consumer가 데이터를 가져간다고해서 데이터가 사라지는 것이 아니라 파티션에 그대로 남습니다.(또 다른 Consumer가 연결되어 가져갈 수 있도록 함.)



- 파티션을 늘리는 이유: Consumer 개수를 늘려 데이터 처리를 분산시킬 수 있습니다.
- 파티션의 데이터(record)를 삭제하는 타이밍: 최대 record 보존 시간, 최대 record 보존 크기로 결정
- Kafka는 key를 특정한 해시로 변형시켜 파티션과 1대1 매칭을 시킵니다.
  파티션에는 동일한 key의 value만 쌓이게 됩니다.
  key를 사용할 경우, 키와 파티션 매칭이 깨질 수 있기 때문에 파티션 개수에 유의하여 파티션을 생성합니다.

Broker, Replication, ISR

Broker, Replication, ISR

Broker

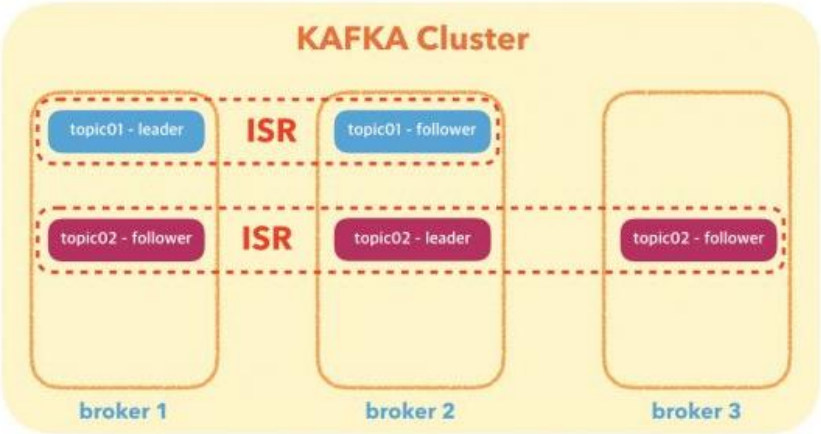
Kafka Brocker은 Kafka가 설치되어 있는 서버 단위를 말합니다.
보통 3개 이상의 broker 이용을 권장합니다.

Replication

서버에 장애가 생겼을 때, 복구 할 수 있는 고가용성을 보장합니다.(Elastic Search에서 공부한 replica 개념과 매우 유사)
replication = ***Leader partition(원본 파티션)*** + ***Follower partition(복제 파티션)***
replication의 개수는 제한됩니다.(replication 개수 <= Kafka broker 개수)

ISR - In Sync Replica

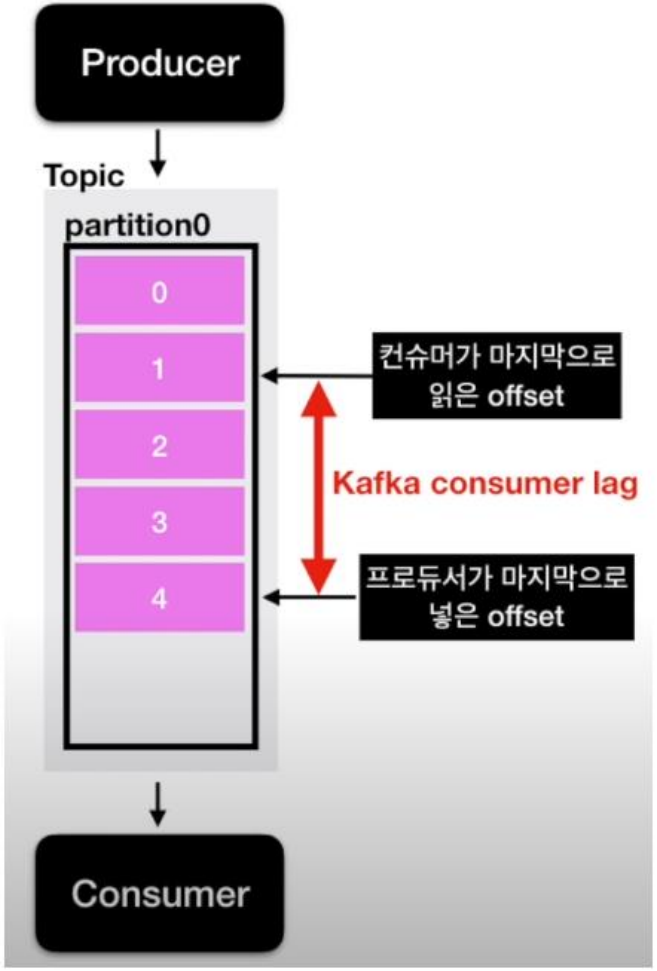
각각의 replication group이 바로 ISR입니다.



Consumer Lag

Lag

Producer가 데이터를 넣어주는 속도가 Consumer가 데이터를 가져가는 속도보다 빠른 경우에
Producer가 데이터를 넣은 오프셋과 Consumer가 데이터를 가져간 오프셋 간에 차이가 발생합니다.
이 **두 오프셋 간의 차이**가 Consumer Lag 입니다.
파티션이 하나가 아니라 여러개라면 Lag도 여러개가 존재할 수 있습니다.
여러개의 Lag중에 **가장 높은 숫자의 Lag**을 records-lag-max라고 합니다.



Burrow – Consumer Lag checking

Burrow

Consumer Lag을 모니터링하기 위해 사용합니다.

Burrow의 특징

- 멀티 카프카 클러스터 지원
- Sliding window를 통한 Consumer의 status(ERROR/WARING/OK) 확인
- HTTP api 제공(다양한 추가 생태계 구축 가능)

Zookeeper?

주키퍼란?

주키퍼는(Zookeeper)는 **분산 코디네이션 서비스를 제공**하는 오픈소스 프로젝트입니다.
주키퍼는 znode로 이루어진 **분산 데이터 모델을 지원하는 시스템**이라고 해도 과언이 아닙니다.
이 데이터 모델은 **리눅스(linux) 파일시스템과 유사한 시스템을 제공**하고 이것이 주키퍼의 핵심입니다.
주키퍼에 채택된 아키텍처와 기법들은 이 데이터 모델을 안정적으로 제공하고자 하기 위함입니다.
이 시스템을 통해 주키퍼는 글로벌락, 클러스터 정보, Leader 선출 등을 구현해야하는 곳에 활용할 수 있습니다.

Zookeeper 사용용도

주키퍼 사용용도

주키퍼는 클러스터에서 구성 서버들끼리 공유되는 데이터를 유지하거나 어떤 연산을 조율하기 위해 주로 사용합니다.
주키퍼는 카프카의 노드 관리를 해주고, 토픽의 offset 정보등을 저장하기 위해 필요합니다.
클러스터 내의 broker에 대한 분산 처리는 Apache ZooKeeper가 담당합니다.

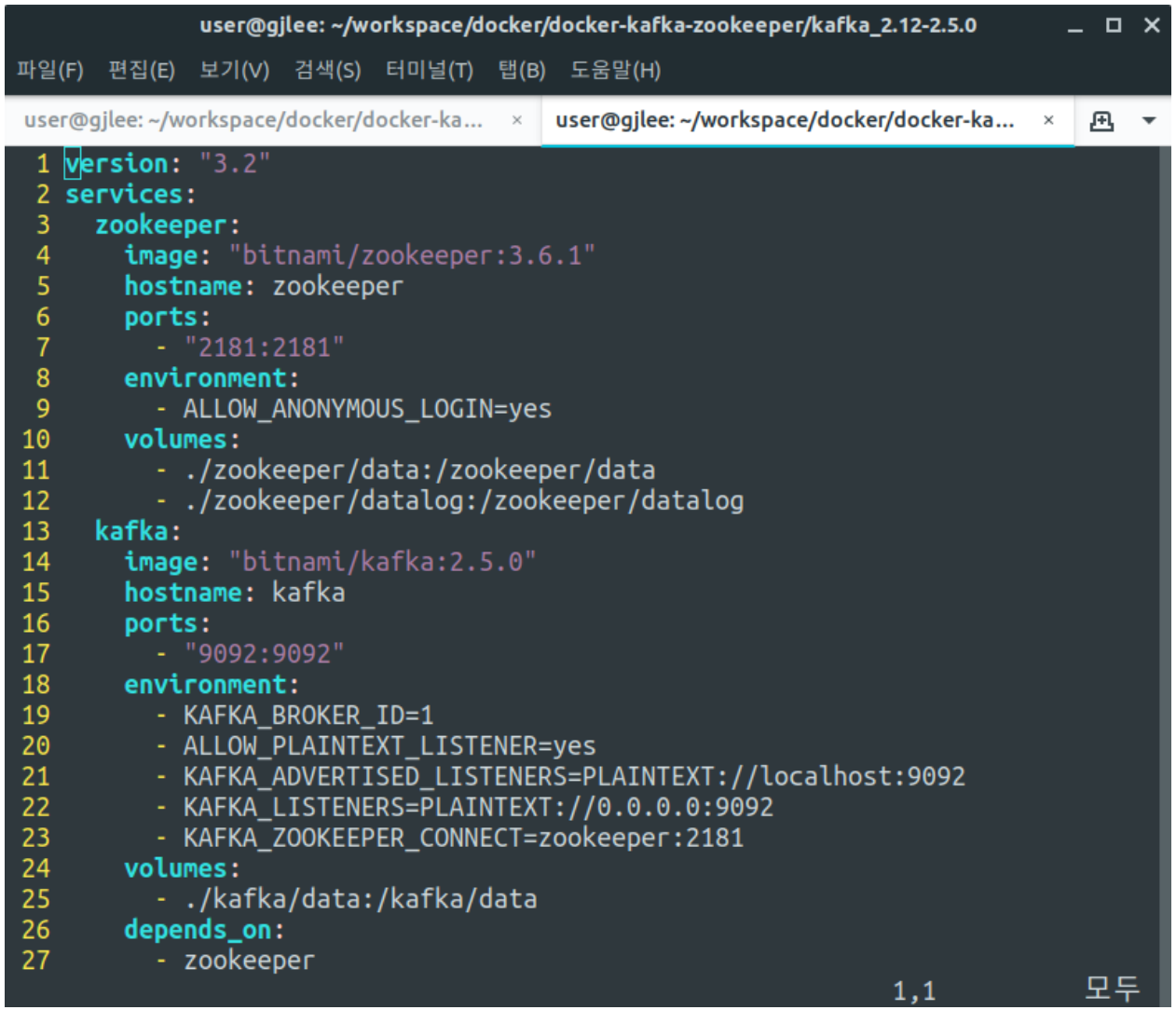
- 설정 관리(Configuration management) : 클러스터의 설정 정보를 최신으로 유지하기 위한 조율 시스템으로 사용됩니다.
- 클러스터 관리(Cluster management) : 클러스터의 서버가 추가되거나 제외될 때 그 정보를 클러스터 안 서버들이 공유하는 데 사용됩니다.
- 리더 채택(Leader selection) : 다중 어플리케이션 중에서 어떤 노드를 리더로 선출할 지를 정하는 로직을 만드는 데 사용됩니다. 주로 복제된 여러 노드 중 연산이 이루어지는 하나의 노드를 택하는 데 사용됩니다.
- 락, 동기화 서비스(Locking and synchronization service) : 클러스터에 쓰기 연산이 빈번할 경우 경쟁상태에 들어갈 가능성이 커집니다. 이는 데이터 불일치를 발생시킵니다. 이 때, 클러스터 전체를 대상을 동기화해(락을 검) 경쟁상태에 들어갈 경우를 사전에 방지합니다.

도커를 이용한 Kafka-zookeeper 실습

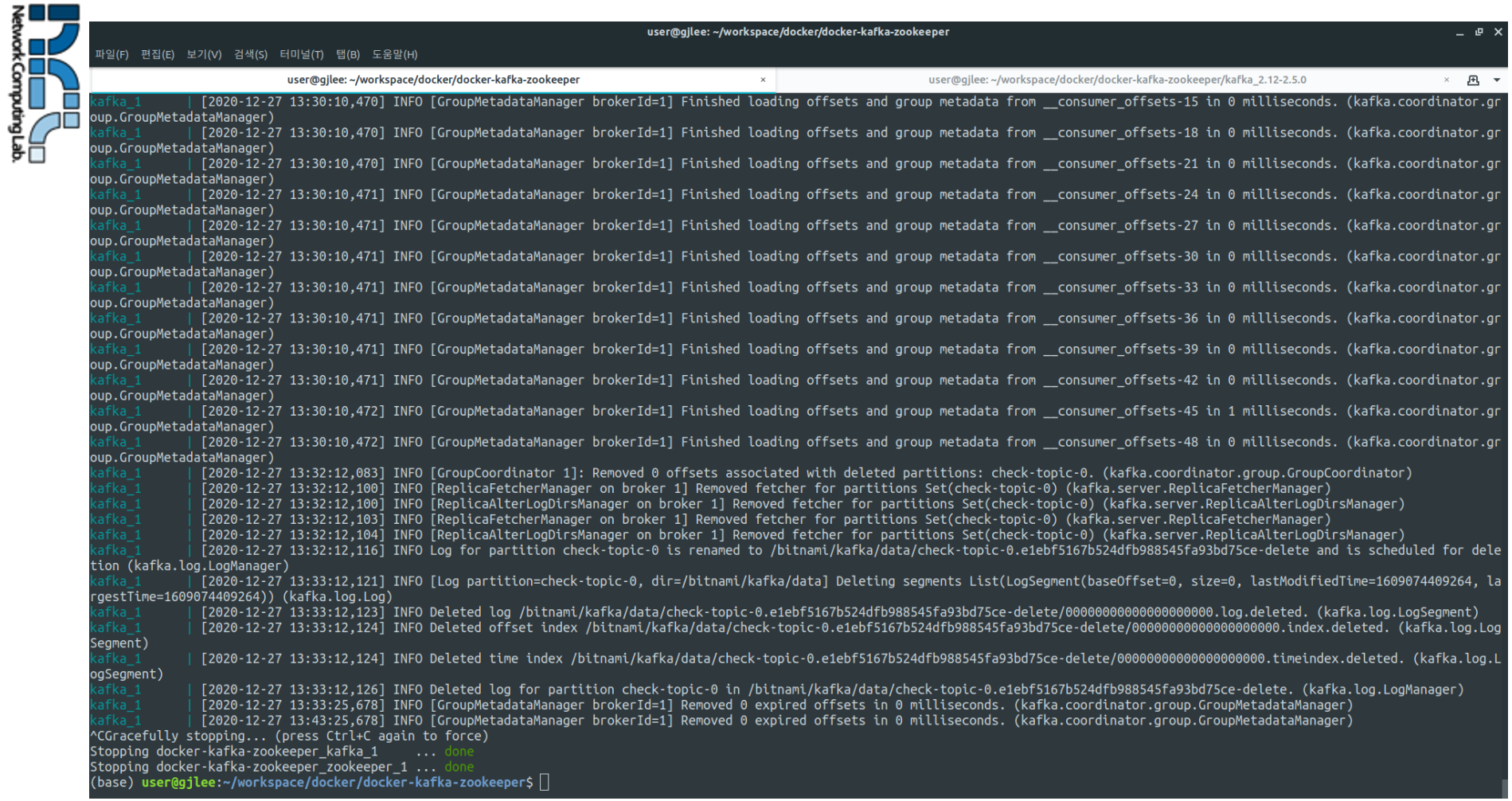
NClab 21st

Network Computing Lab.

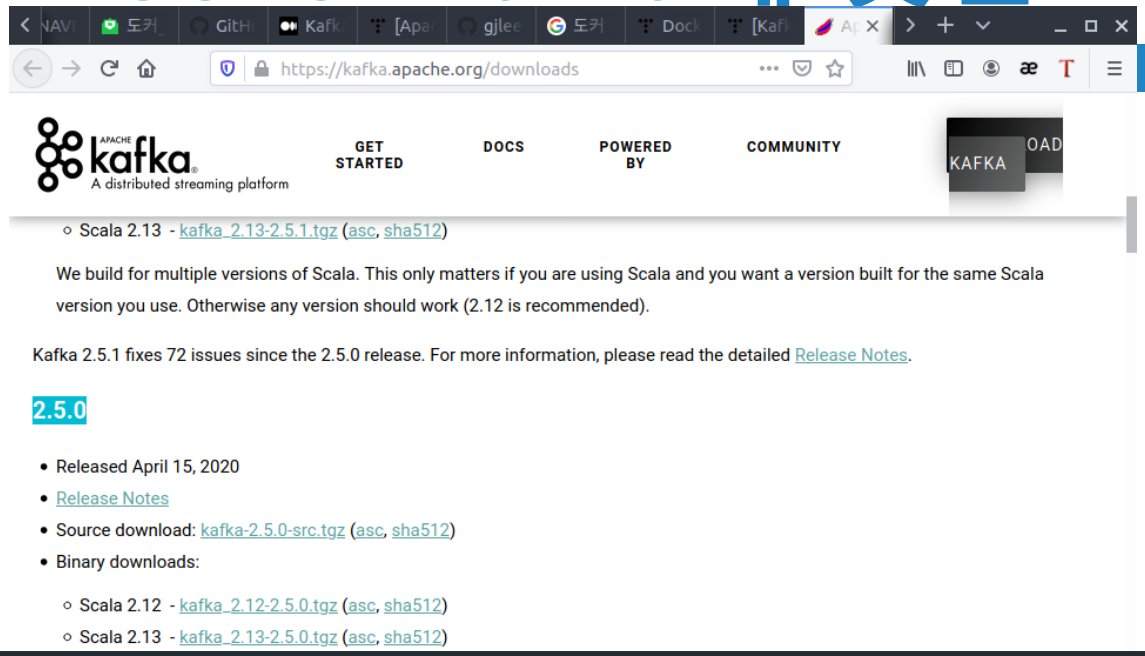
Docker-compose.yml

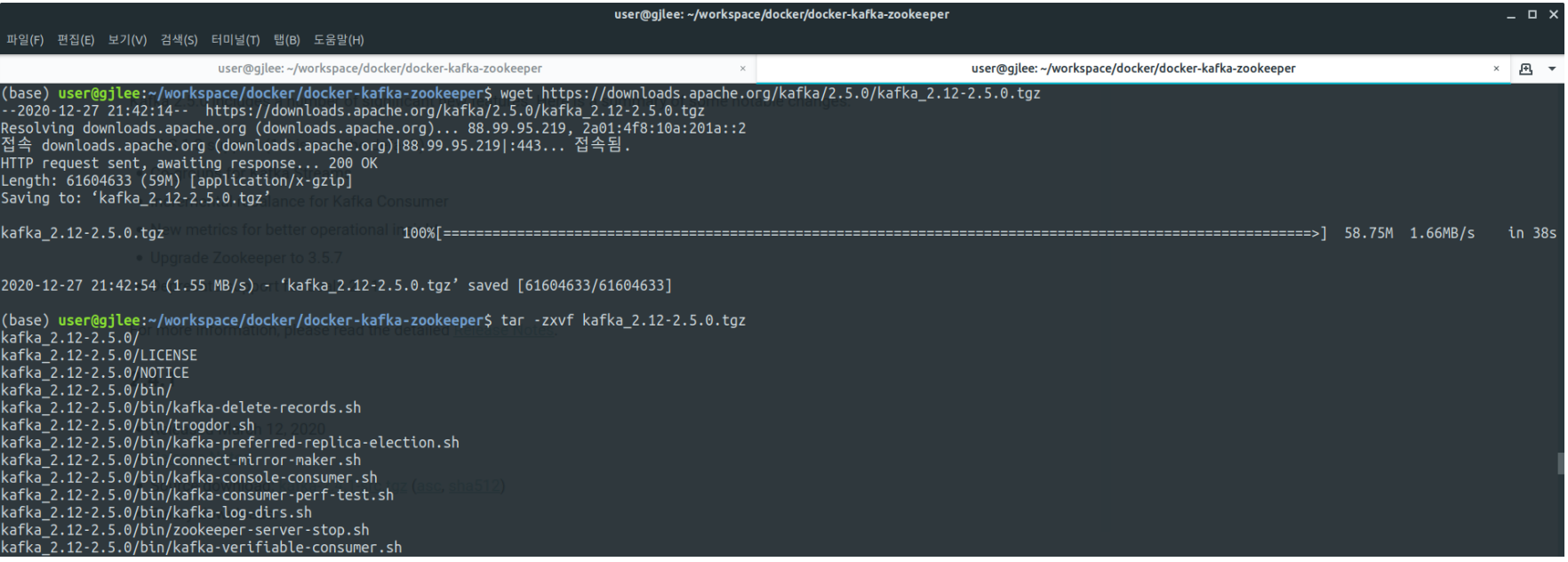


Docker-compose up



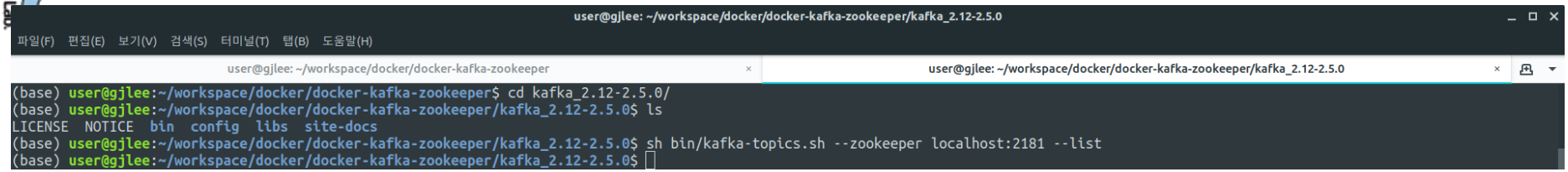
Docker kafka에 맞는 버전 다운로드



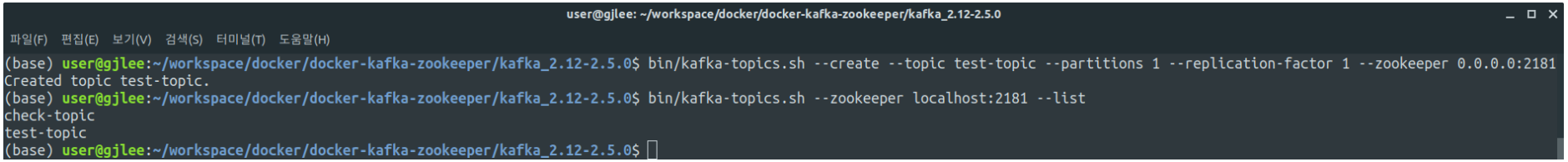


Topic 생성

토픽 리스트 확인

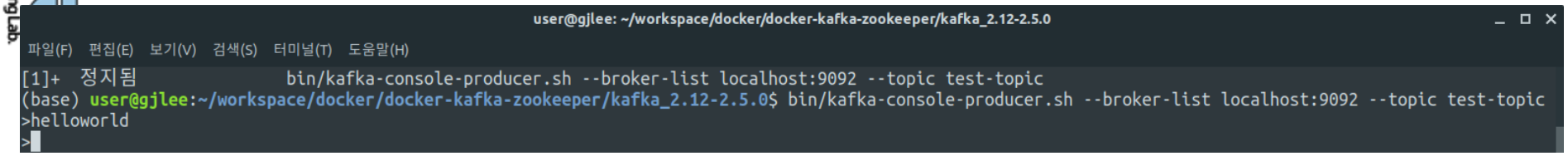


토픽 생성 및 확인

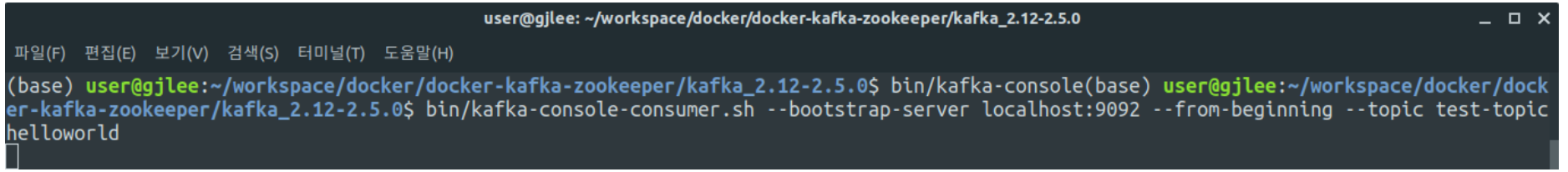


메시지 전송 테스트

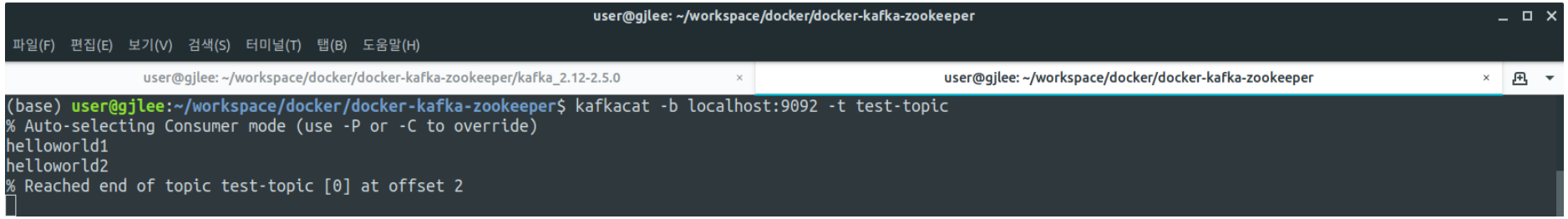
producer



consumer



Kafkacat을 통한 확인



감사합니다.