



- **아파치 카프카 개요**

아파치 카프카로 무엇을 할 수 있는지 알아봅니다.

- **카프카 기초**

카프카의 메시지 송수신 구조와 기본 용어에 대해 설명합니다.

- **카프카 설치**

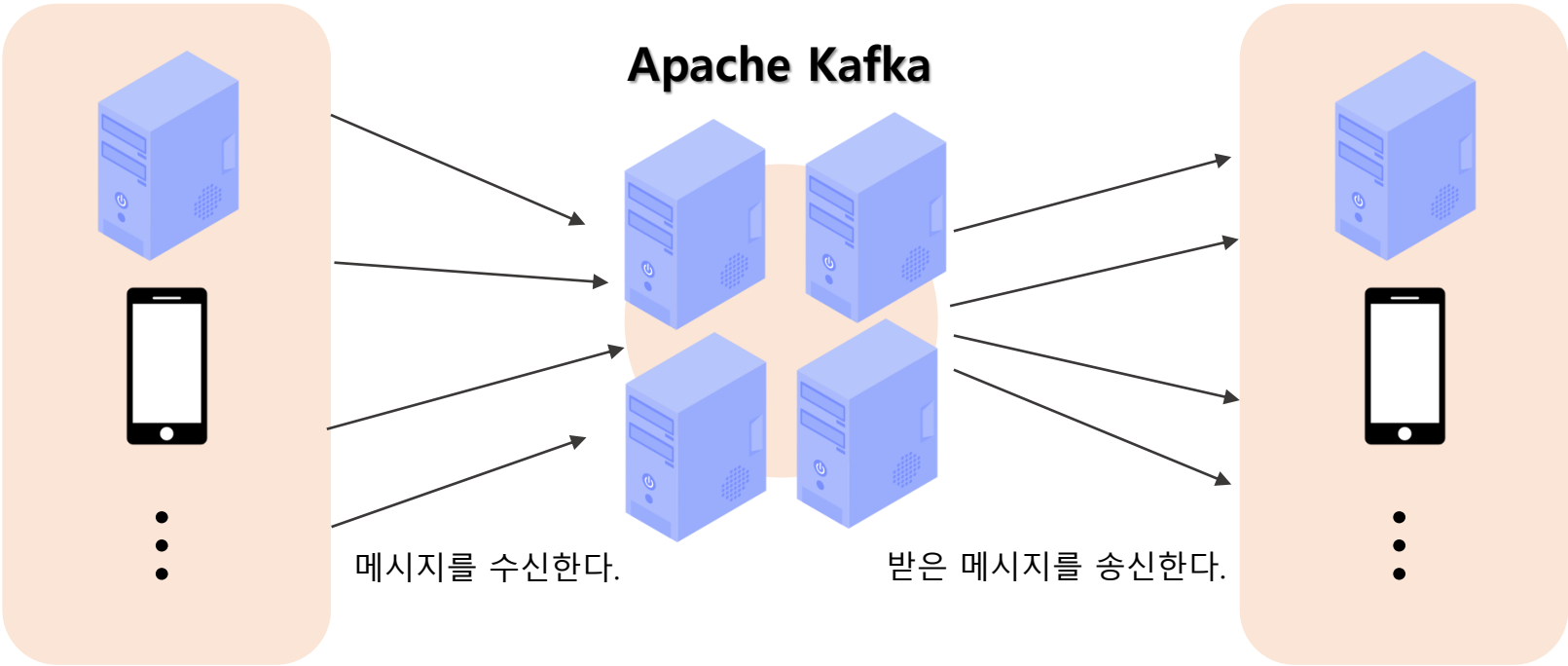
카프카 구축 방법을 설명합니다.

- **카프카 능숙하게 사용하기**

추가적으로 카프카에 대해 알아야할 것을 설명합니다.

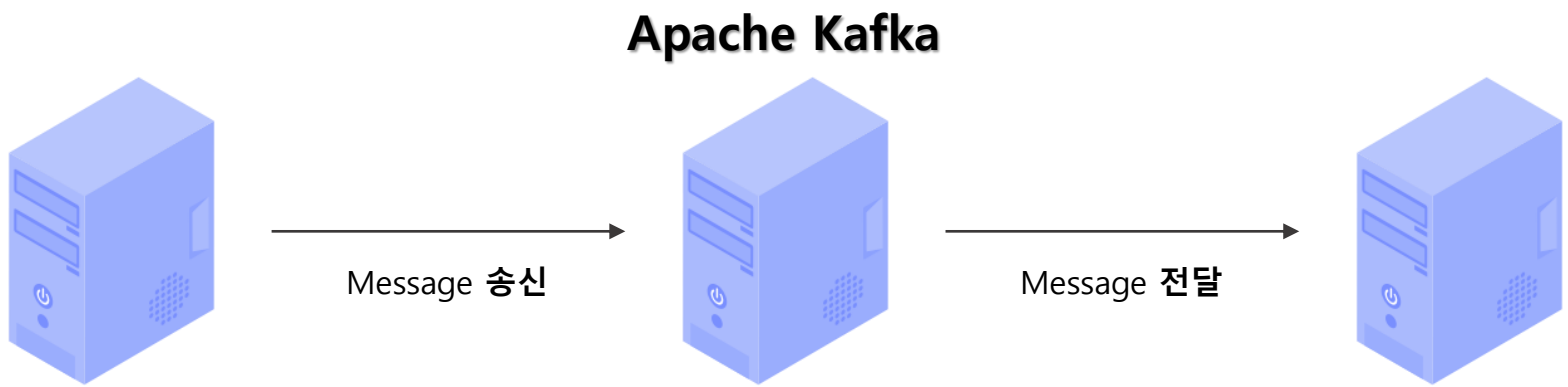
아파치 카프카란?

여러 대의 분산 서버에서 대량의 데이터를 처리하는 분산 메시징 시스템이다.
카프카는 대량의 데이터를 높은 처리량과 실시간으로 취급하기 위한 제품이다.(확장성)



아파치 카프카란?

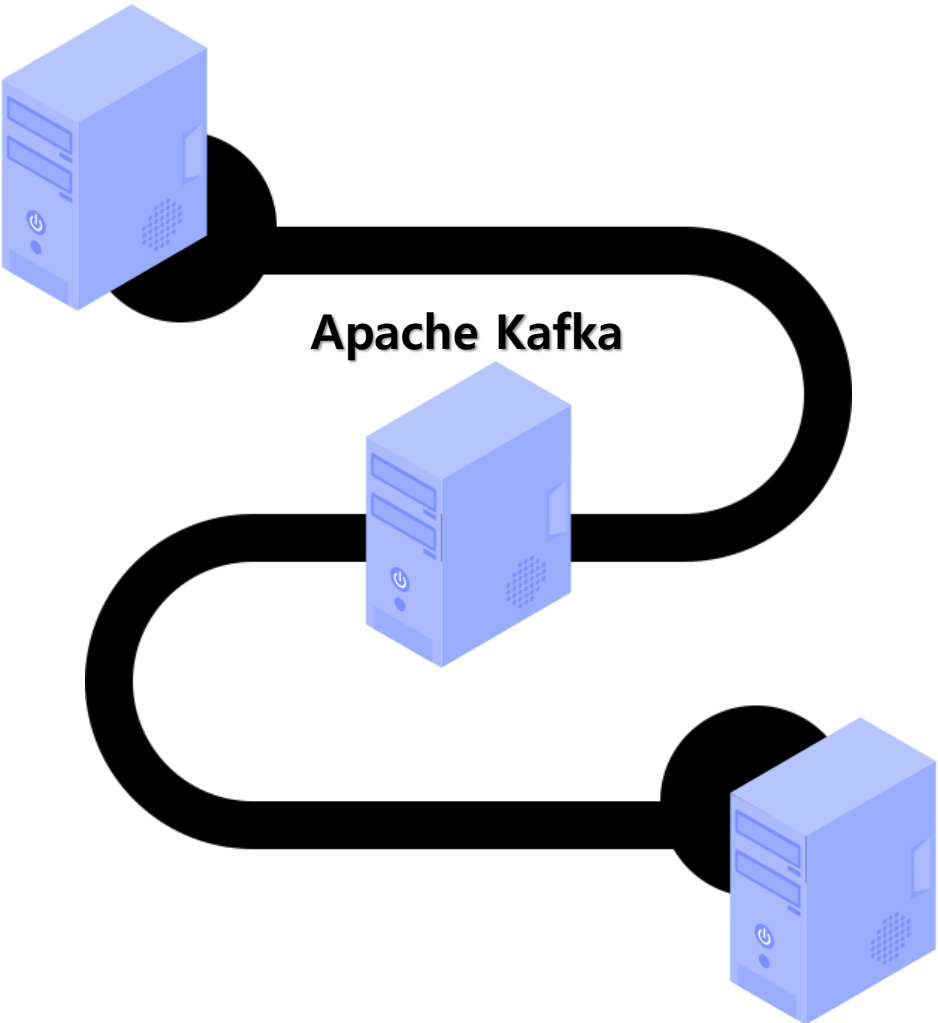
메시지를 받고, 받은 메시지를 다른 시스템이나 장치에 보내기 위해 사용된다.



전달하고 싶은 메시지를 나한테 맡겨!
나한테 보내주면 필요한 시스템이 가져갈거야!

아파치 카프카란?

카프카는 여러 시스템과 장치를 연결하는 중요한 역할을 한다.



카프카의 메시징 모델과 스케일 아웃

요구사항

- ✓ 높은 처리량으로 실시간 처리한다.
- ✓ 임의의 타이밍에 데이터를 읽는다.
- ✓ 다양한 제품과 시스템에 쉽게 연동한다.
- ✓ 메시지를 잃지 않는다.



실현 수단

메시징 모델과 스케일 아웃형 아키텍처

디스크로의 데이터 영속화

이해하기 쉬운 API 제공

전달 보증 (Ack와 오프셋 커밋 이용)

카프카의 메시징 모델과 스케일 아웃

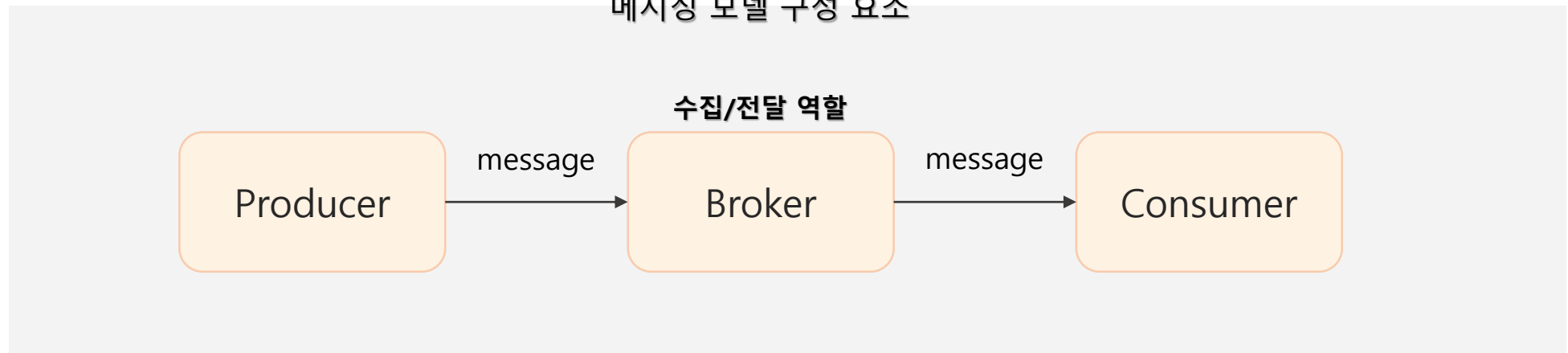
요구사항

- ✓ 높은 처리량으로 실시간 처리한다.
- ✓ 임의의 타이밍에 데이터를 읽는다.
- ✓ 다양한 제품과 시스템에 쉽게 연동한다.



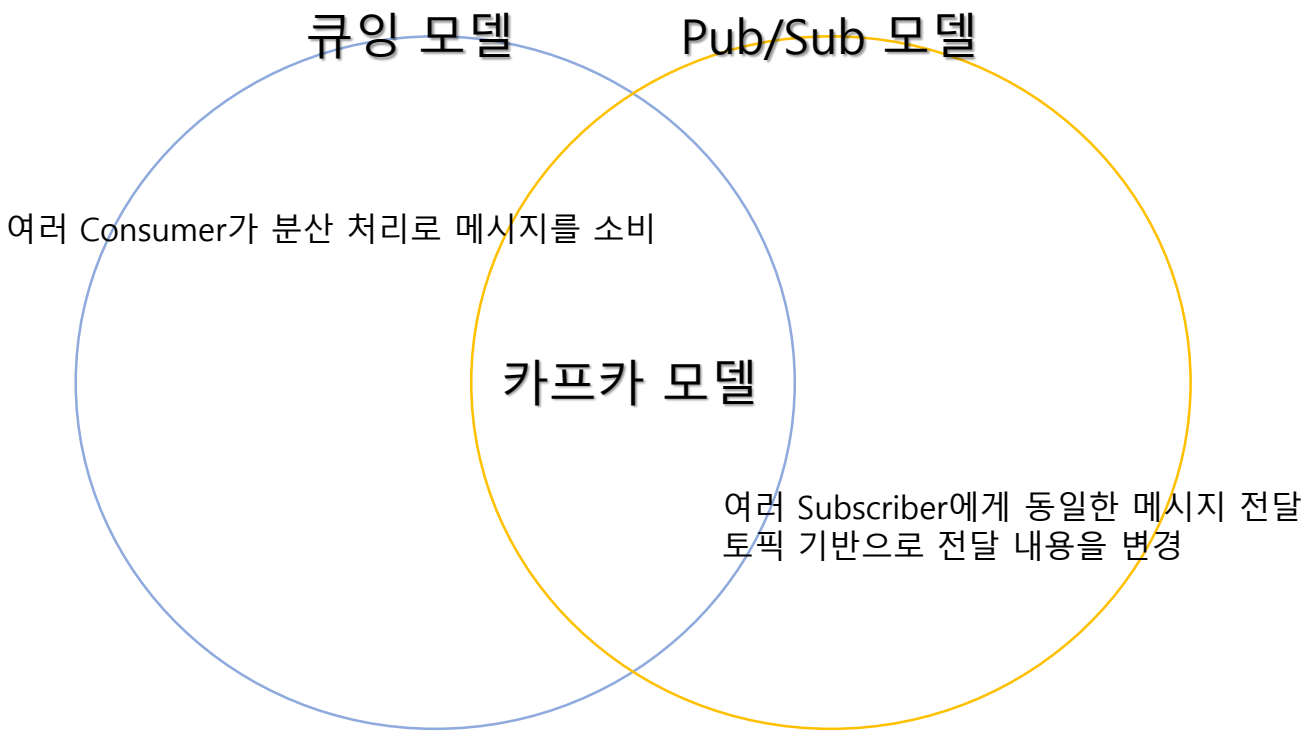
메시징 모델 채택!

메시징 모델 구성 요소



메시징 모델과 스케일 아웃

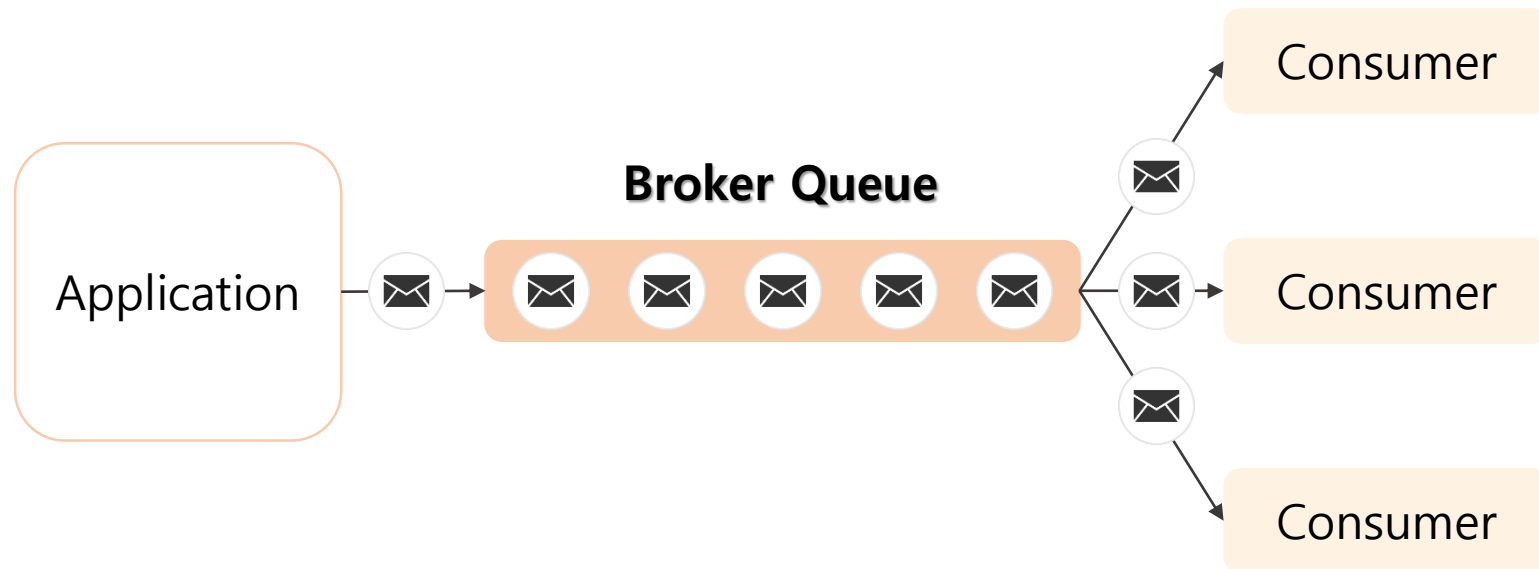
카프카는 기존의 메시징 모델인 '큐잉 모델'과 'Pub/Sub 메시징 모델'의 특징을 **검비**한 형태로 만들어졌다.



메시징 모델과 스케일 아웃

큐잉 모델?

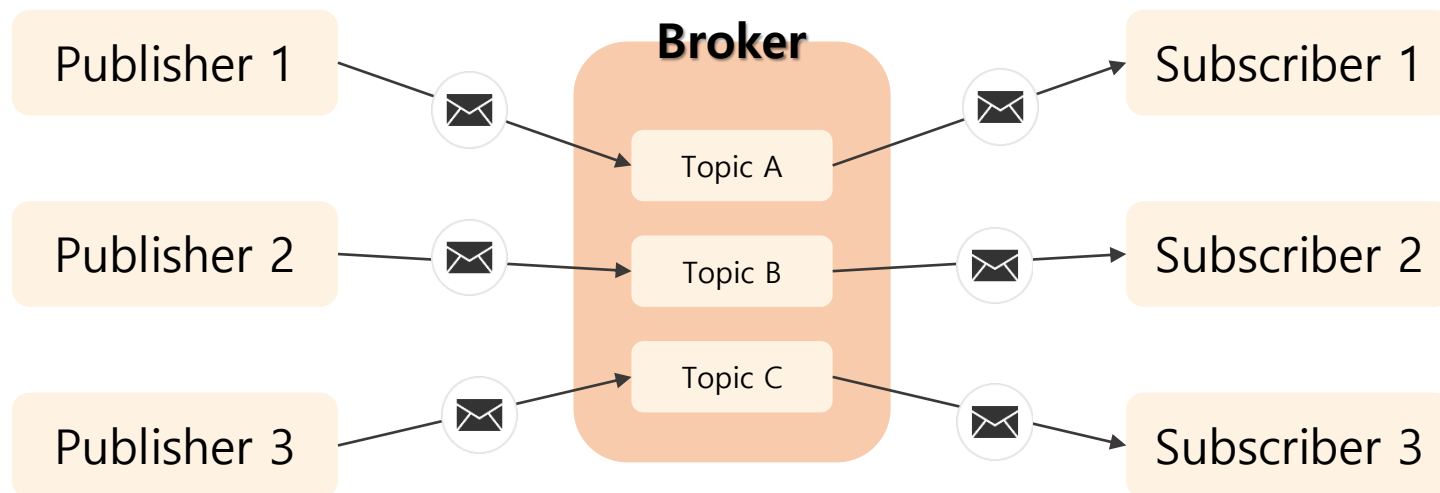
- ✓ Consumer에 의한 메시지 처리가 병렬로 가능하다.
- ✓ 큐에서 추출된 메시지는 Consumer에 도달하면 사라진다.
- ✓ 하나의 메시지는 하나의 Consumer에서 처리된다.



메시징 모델과 스케일 아웃

Pub/Sub 메시징 모델?

- ✓ Publisher는 Broker에 메시지를 보내기만 하고, 더 이상 신경 쓰지 않는다.
- ✓ Publisher 가 보낸 메시지는 Broker 내의 토픽이라 불리는 부분에 등록된다.
- ✓ Subscriber는 브로커 내의 토픽에서 자신이 흥미 있는 것만 선택한다.
- ✓ 큐잉 모델과는 달리 같은 토픽을 구독하는 여러 Subscriber에게는 동일한 메시지가 전달된다.



카프카 사용 기업

yahoo!



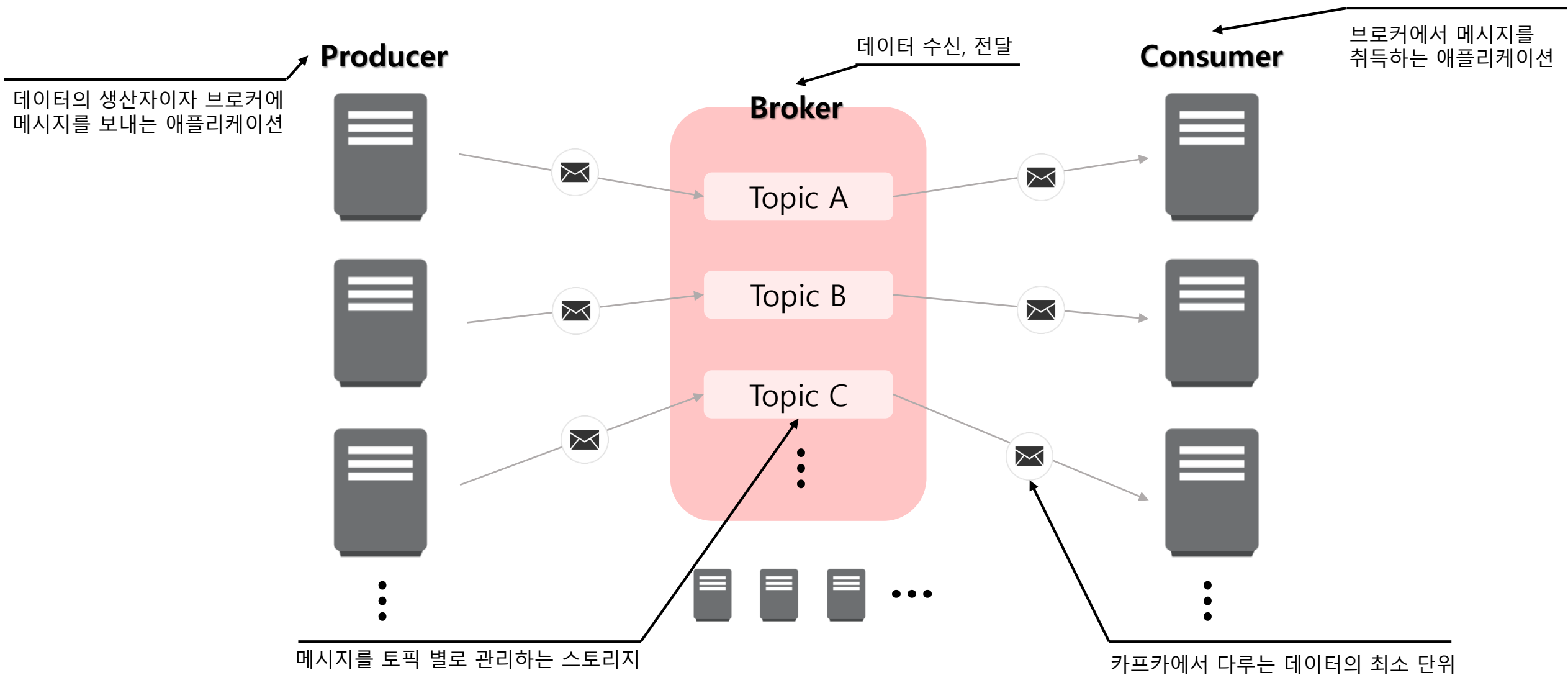
...



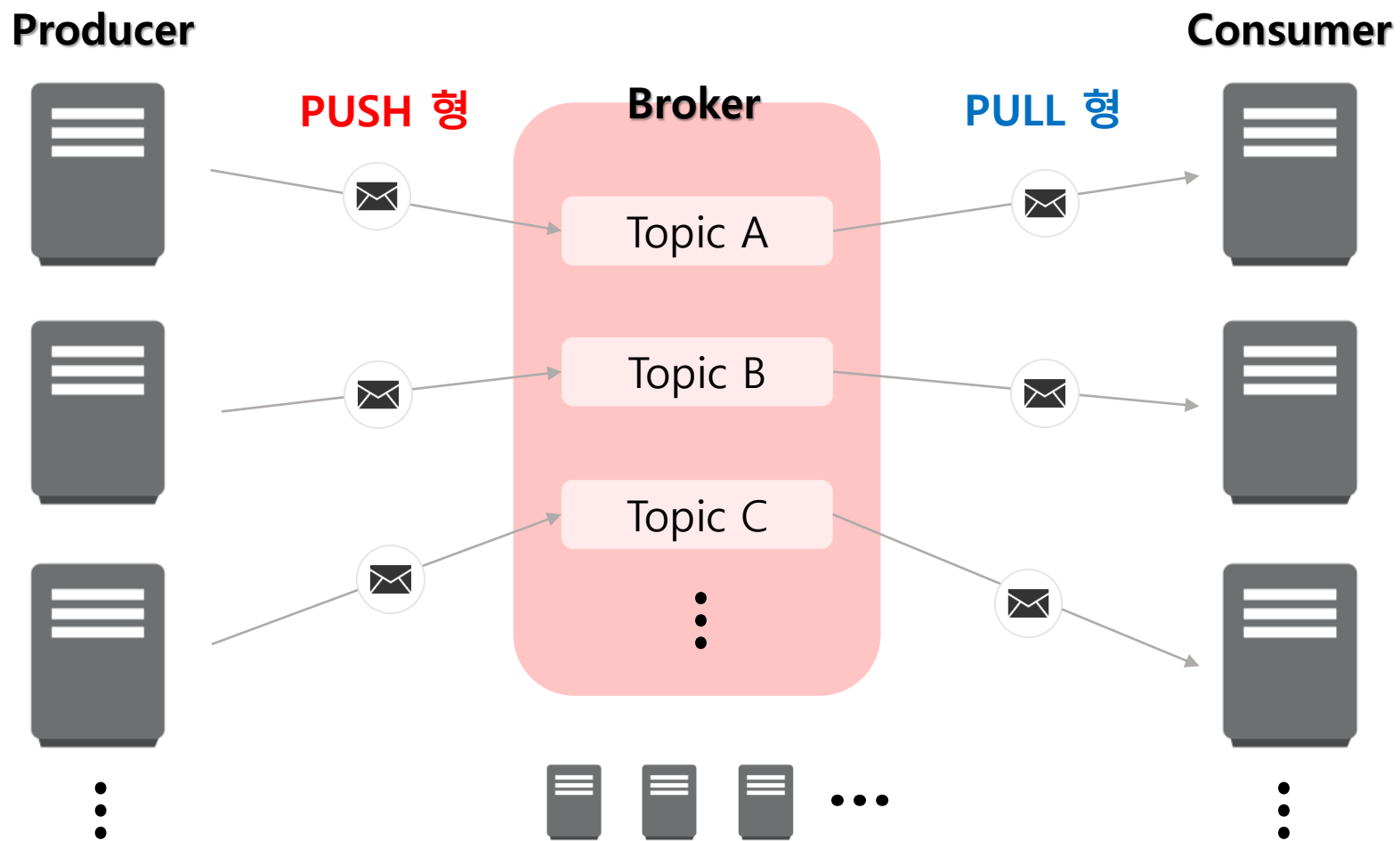
이 장의 내용

- ✓ 메시지 송수신 기본
- ✓ 시스템 구성
- ✓ 분산 메시징을 위한 구조
- ✓ 데이터 견고함을 담보하는 복제 구조

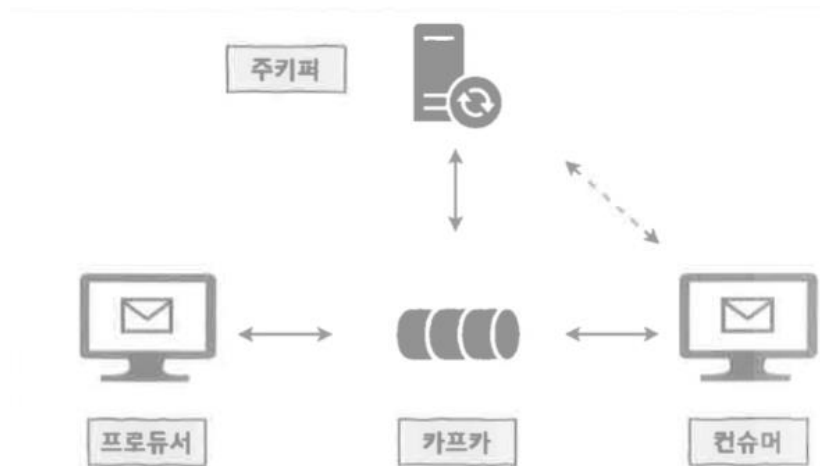
메시지 송수신 기본



메시지 *PUSH/PULL* 형

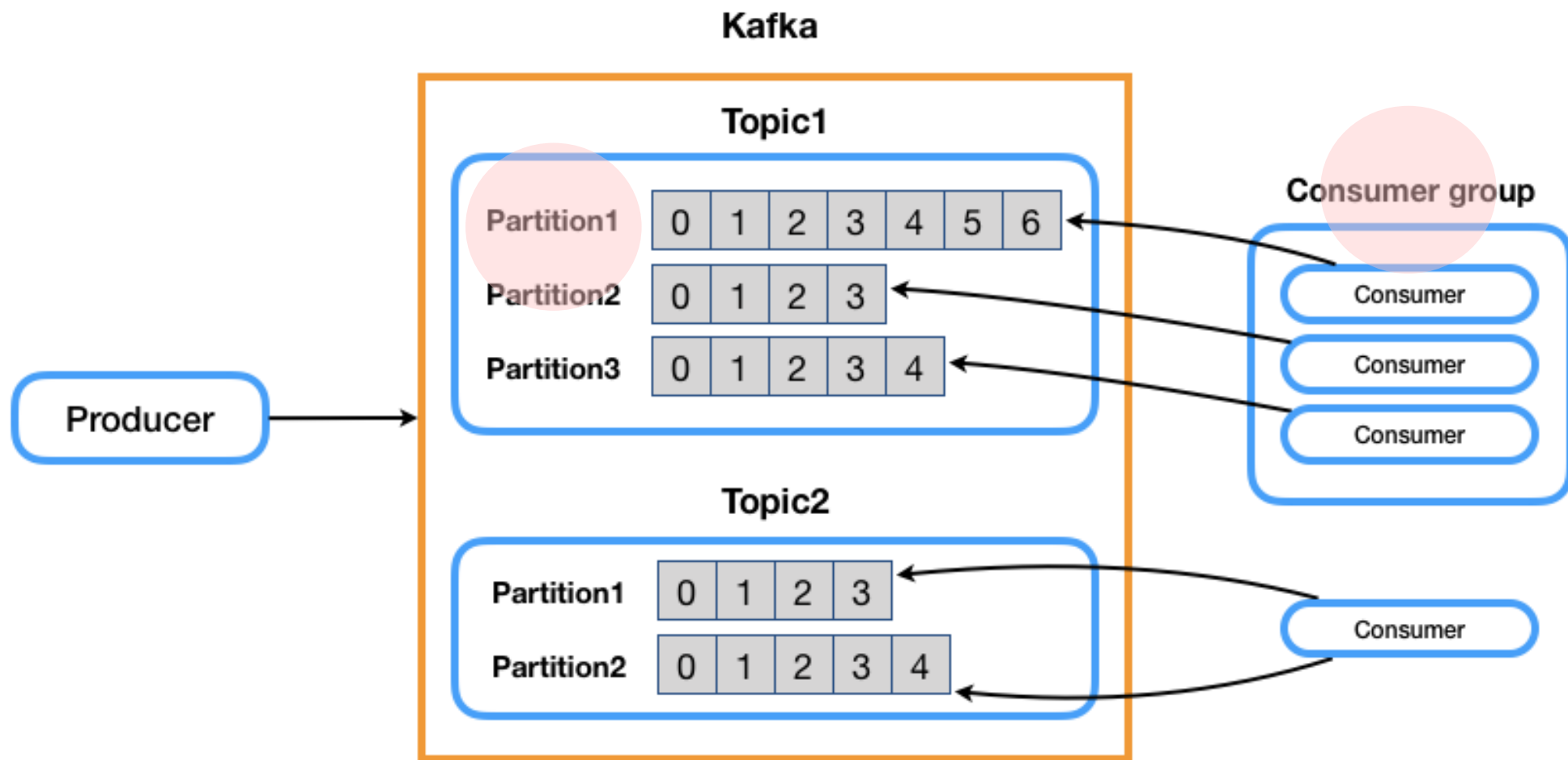


Zookeeper



- ✓ 카프카의 브로커에 있어 분산 처리를 위한 관리도구로 '아파치 주키퍼'가 있다.
- ✓ 컨슈머와 통신하는 부분, 카프카와 직접 통신하기도 한다.
- ✓ 카프카의 메타 데이터(토픽과 파티션 등)를 주키퍼에 저장하고, 카프카의 상태 관리 등을 목적으로 이용한다.
- ✓ 주키퍼 클러스터의 구조상 3,5처럼 홀수로 구성하는 것이 일반적이다.

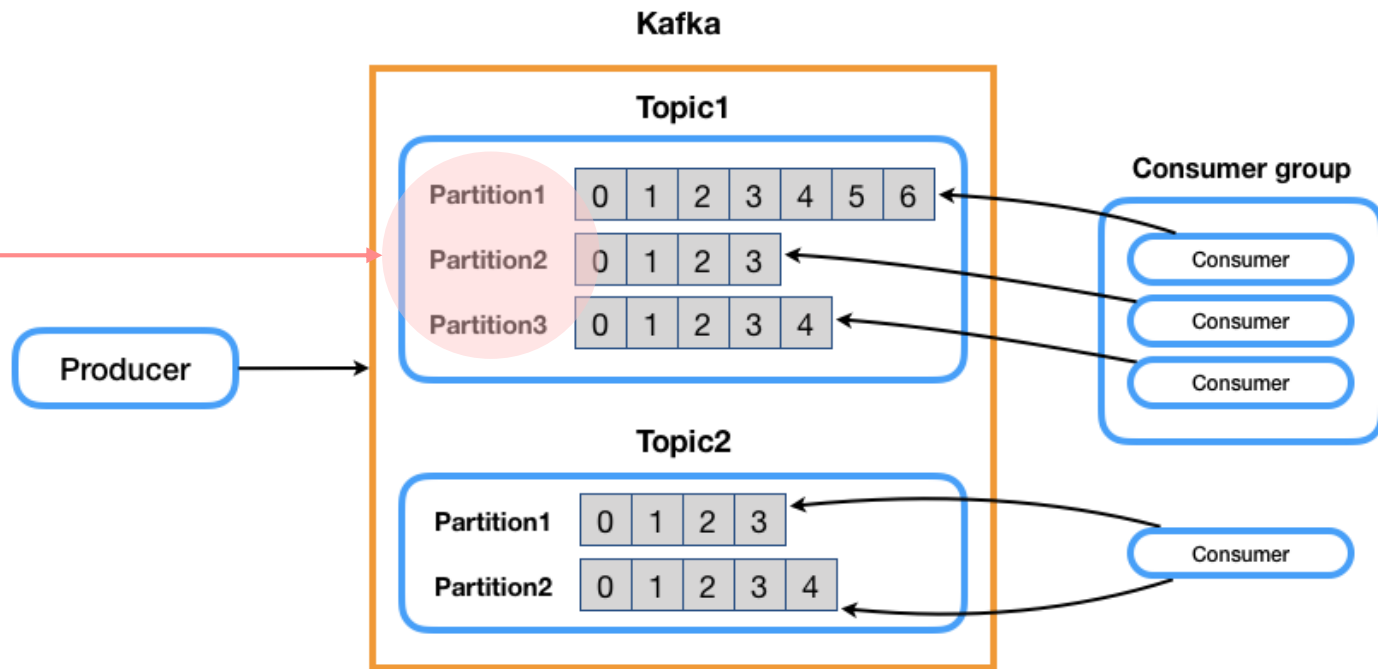
분산 메시지를 위한 구조



분산 메시지를 위한 구조

Partition

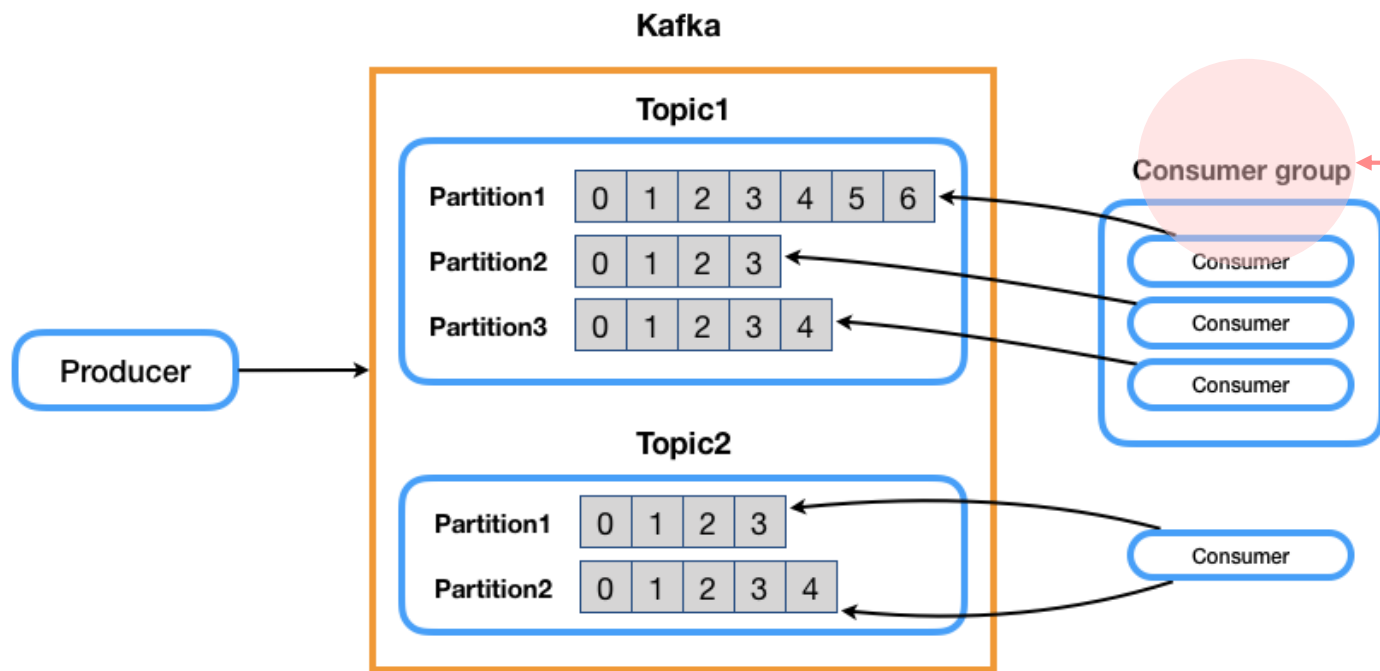
- : 토픽에 대한 대량의 메시지 입출력을 지원하기 위해, 브로커상의 데이터를 읽고 쓰는 것은 파티션이라는 단위로 분할되어 있다.
- : 토픽을 구성하는 파티션은 메시지 수신, 전달을 분산해서 실시함으로써 하나의 토픽에 대한 대규모 데이터 수신과 전달을 지원한다.



분산 메시지를 위한 구조

Consumer Group

- : 단일 애플리케이션 안에서 여러 컨슈머가 단일 토픽이나 여러 파티션에서 메시지를 취득하는 방법으로 컨슈머 그룹이라는 개념이 존재한다.
- : 카프카 클러스터 전체에서 글로벌 ID를 컨슈머 그룹 전체에서 공유하고 여러 컨슈머는 자신이 소속한 컨슈머 그룹을 식별해, 읽어들일 파티션을 분류하고 재시도를 제어한다.

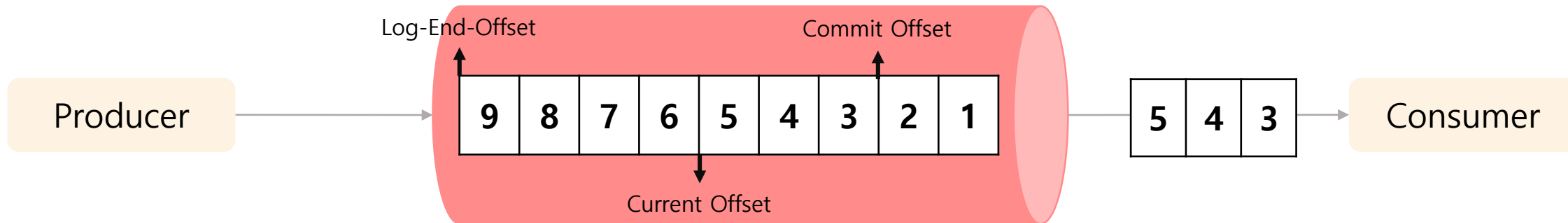


분산 메시지를 위한 구조

오프셋

: 각 파티션에서 수신한 메시지에는 각각 일련번호가 부여되어 있어 파티션 단위로 메시지 위치를 나타내는 오프셋이라는 관리 정보를 이용해 컨슈머가 취득하는 메시지의 범위 및 재시도를 제어한다.

1. Log-End-Offset(LEO): 파티션 데이터의 끝을 나타낸다.
2. Current Offset: 컨슈머가 어디까지 메시지를 읽었는가를 나타낸다.
3. Commit Offset: 컨슈머가 어디까지 커밋했는지를 나타낸다.



분산 메시지를 위한 구조

메시지 송수신

- : 반드시 하나의 메시지 단위로 송수신하지 않는다. (default는 메시지 단위로 송수신)
- : 처리량을 높이기 위해 메시지를 축적하여 배치 처리로 한번에 송수신하는 기능도 제공한다.
- : 배치 처리의 간격에 대해서는 처리량과 대기 시간의 트레이드 오프를 고려한 설계가 필요하다.

컨슈머의 롤백

- : Offset Commit의 구조를 이용해 컨슈머 처리 실패, 고장 시 롤백 메시지 재취득을 실현한다.
- : 메시지를 처리 완료 상태에서 Commit Offset 업데이트 직전의 고장의 경우는 동일한 메시지가 재전송되고, 메시지 중복 처리(또는 중복 허용)에 대한 어플리케이션의 후속 처리가 필요하다.

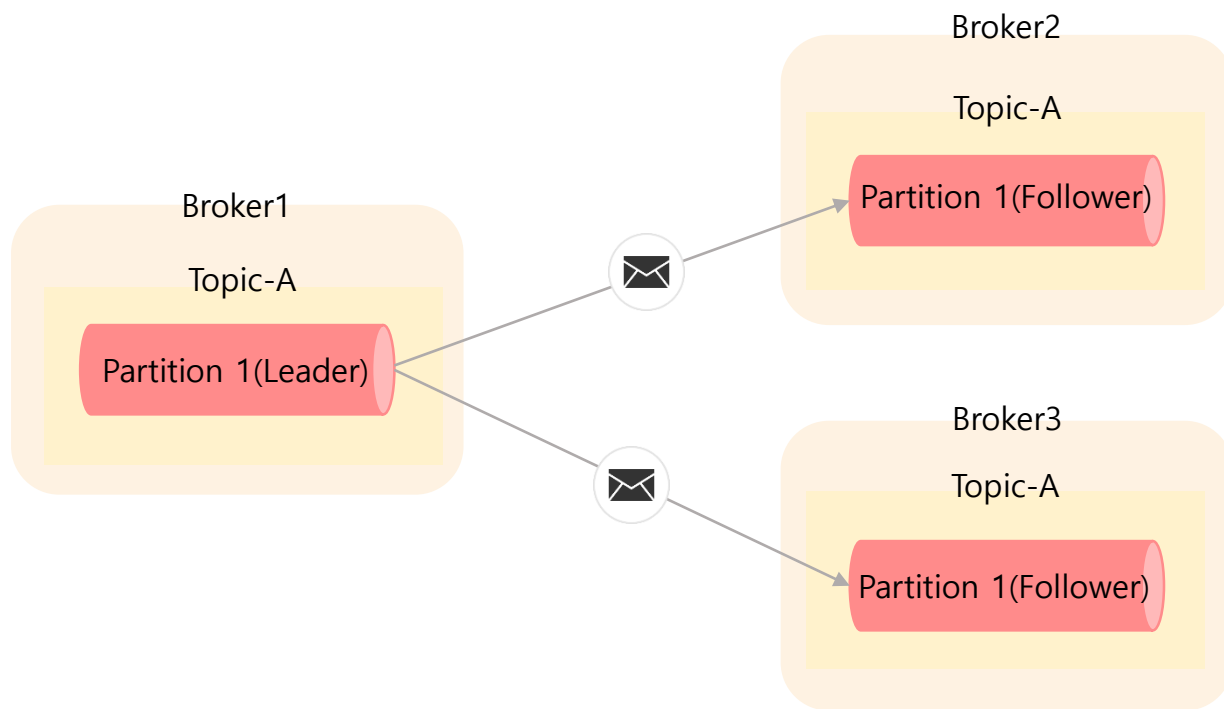
메시지 전송 시 파티셔닝

- : 프로듀서에서 송신하는 메시지를 어떻게 파티션으로 보낼지 결정하는 파티셔닝 기능을 제공한다.
- : 두 가지 방법

1. Key의 해시 값을 사용한 송신(지정된 파티션으로 송신)
2. 라운드로빈에 의한 송신

데이터의 견고성을 높이는 복제 구조

- 카프카는 메시지를 중계함과 동시에 서버가 고장 났을 때에 수신한 메시지를 잃지 않기 위해 복제 구조를 갖추고 있다.
- 파티션은 단일 또는 여러 개의 레플리카로 구성되어 토픽 단위로 레플리카 수를 지정할 수 있다.
- 레플리카 중 하나는 Leader이며, 나머지는 Follower라고 불린다.
- Follower는 Leader로부터 메시지를 계속적으로 취득하여 복제를 유지하도록 동작한다.
- 프로듀서/컨슈머와의 데이터 교환은 Leader가 맡고 있다.



토픽 설정

- 파티션 수 = 1
- 레플리카 수 = 3

레플리카의 동기 상태

복제 완료 최신 오프셋 (High Watermark)

- High Watermark는 복제가 완료된 오프셋이다.
- 컨슈머는 High Watermark까지 기록된 메시지를 취득할 수 있다.

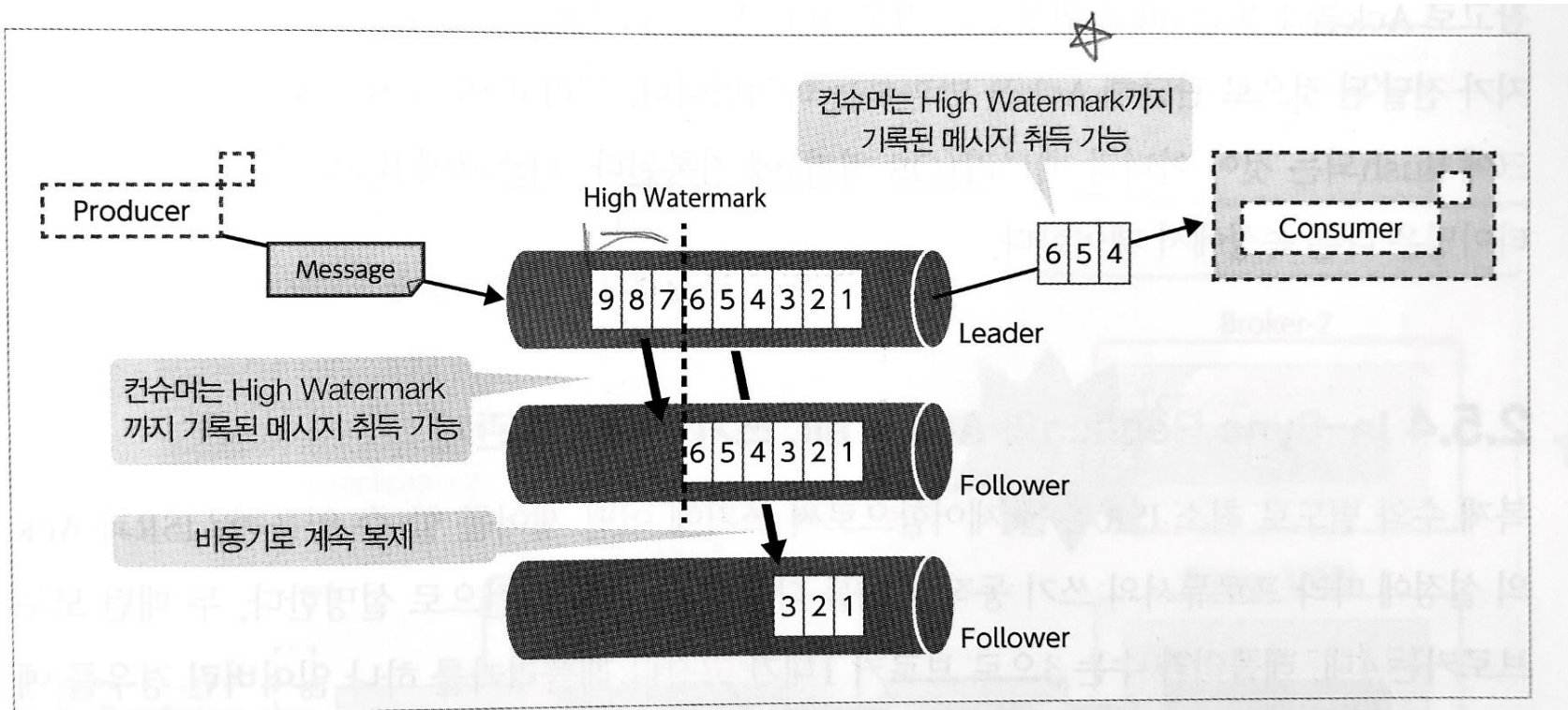


그림 2-12 복제의 상세 구조

레플리카의 동기 상태

프로듀서의 메시지 도달 보증 수준

- 브로커에서 프로듀서로 메시지가 송신된 것을 나타내는 Ack 를 어느 타이밍에 송신할 것인지를 제어한다.

Ack 설정	설명
0	프로듀서는 메시지 송신 시 ack를 기다리지 않고 다음 메시지를 송신한다.
1	Leader Replica에 메시지가 전달되면 ack를 반환한다.
all	모든 ISR의 수만큼 복제되면 ACK를 반환한다.

Windows 환경에서 binary 설치

1. 주키퍼 다운로드 (<http://www.apache.org/dyn/closer.cgi/zookeeper/>)



COMMUNITY-LED DEVELOPMENT "THE APACHE WAY"

Projects ▾

People ▾

Community ▾

License ▾

Sponsors ▾



We suggest the following mirror site for your download:

<http://apache.mirror.cdnetworks.com/zookeeper/>

Other mirror sites are suggested below.

It is essential that you verify the integrity of the downloaded file using the PGP signature (`.asc` file) or a hash (`.md5` or `.sha*` file).

Please only use the backup mirrors to download KEYS, PGP signatures and hashes (SHA* etc) -- or if no other mirrors are working.

HTTP

<http://apache.mirror.cdnetworks.com/zookeeper/>

<http://apache.tt.co.kr/zookeeper/>

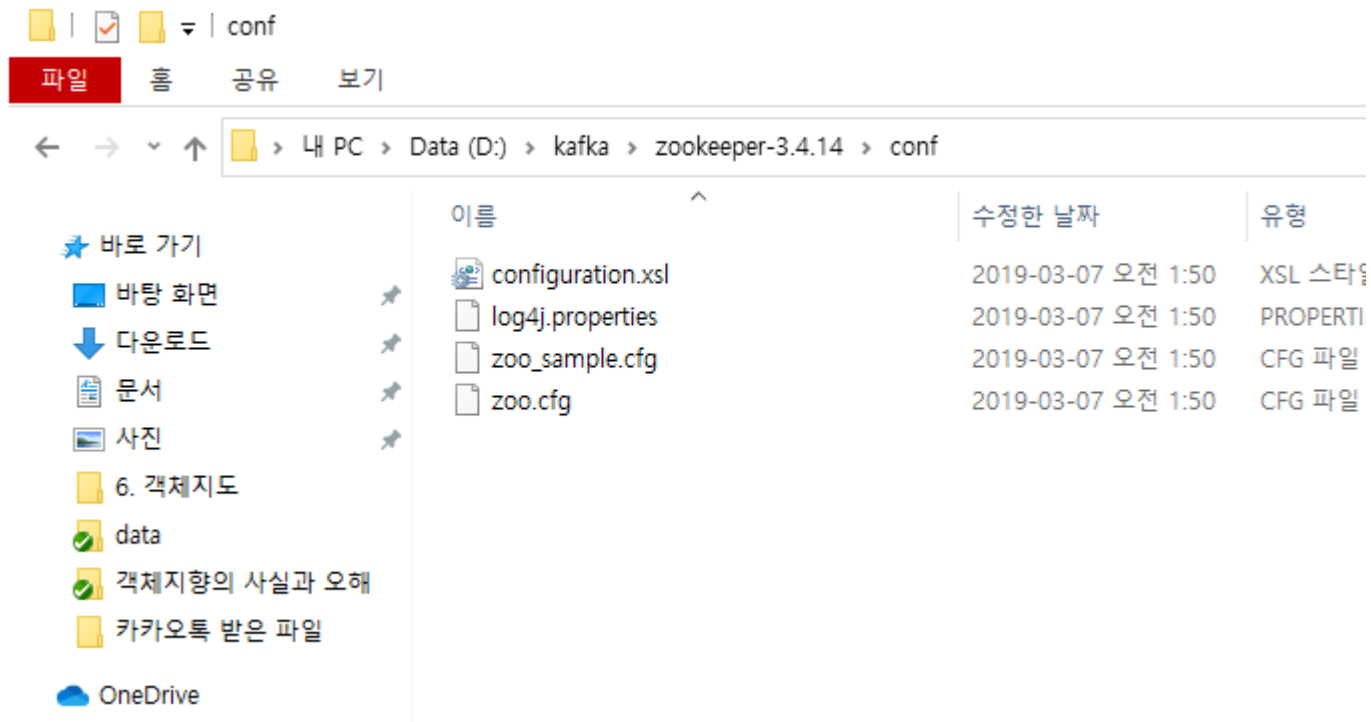
<http://mirror.apache-kr.org/zookeeper/>

<http://mirror.navercorp.com/apache/zookeeper/>

Windows 환경에서 binary 설치

2. 주키퍼 설정

주키퍼 홈 경로에 data 폴더 생성 후
conf 폴더 하위 zoo_sample.cfg 파일 zoo.cfg 이름으로 복사



Windows 환경에서 binary 설치

3. 주키퍼 설정

zoo.cfg 파일 내 data 경로 추가

```
# The number of milliseconds of each tick
tickTime=2000
# The number of ticks that the initial
# synchronization phase can take
initLimit=10
# The number of ticks that can pass between
# sending a request and getting an acknowledgement
syncLimit=5
# the directory where the snapshot is stored.
# do not use /tmp for storage, /tmp here is just
# example sakes.
dataDir=data
# the port at which the clients will connect
clientPort=2181
# the maximum number of client connections.
```

Windows 환경에서 binary 설치

4. 아파치 카프카 다운로드 (<https://kafka.apache.org/downloads>)



GET STARTED

DOCS

POWERED BY

COMMUNITY

DOWNLOAD KAFKA

DOWNLOAD

2.6.0 is the latest release. The current stable version is 2.6.0.

You can verify your download by following these [procedures](#) and using these [KEYS](#).

2.6.0

- Released Aug 3, 2020
- [Release Notes](#)
- Source download: [kafka-2.6.0-src.tgz](#) ([asc](#), [sha512](#))
- Binary downloads:
 - Scala 2.12 - [kafka_2.12-2.6.0.tgz](#) ([asc](#), [sha512](#))
 - Scala 2.13 - [kafka_2.13-2.6.0.tgz](#) ([asc](#), [sha512](#))

Windows 환경에서 binary 설치

5. 카프카 설정 - log 경로 수정 (\${kafkaHome}\config\server.properties)

```
# The receive buffer (SO_RCVBUF) used by the socket server
socket.receive.buffer.bytes=102400

# The maximum size of a request that the socket server will accept (pr
socket.request.max.bytes=104857600

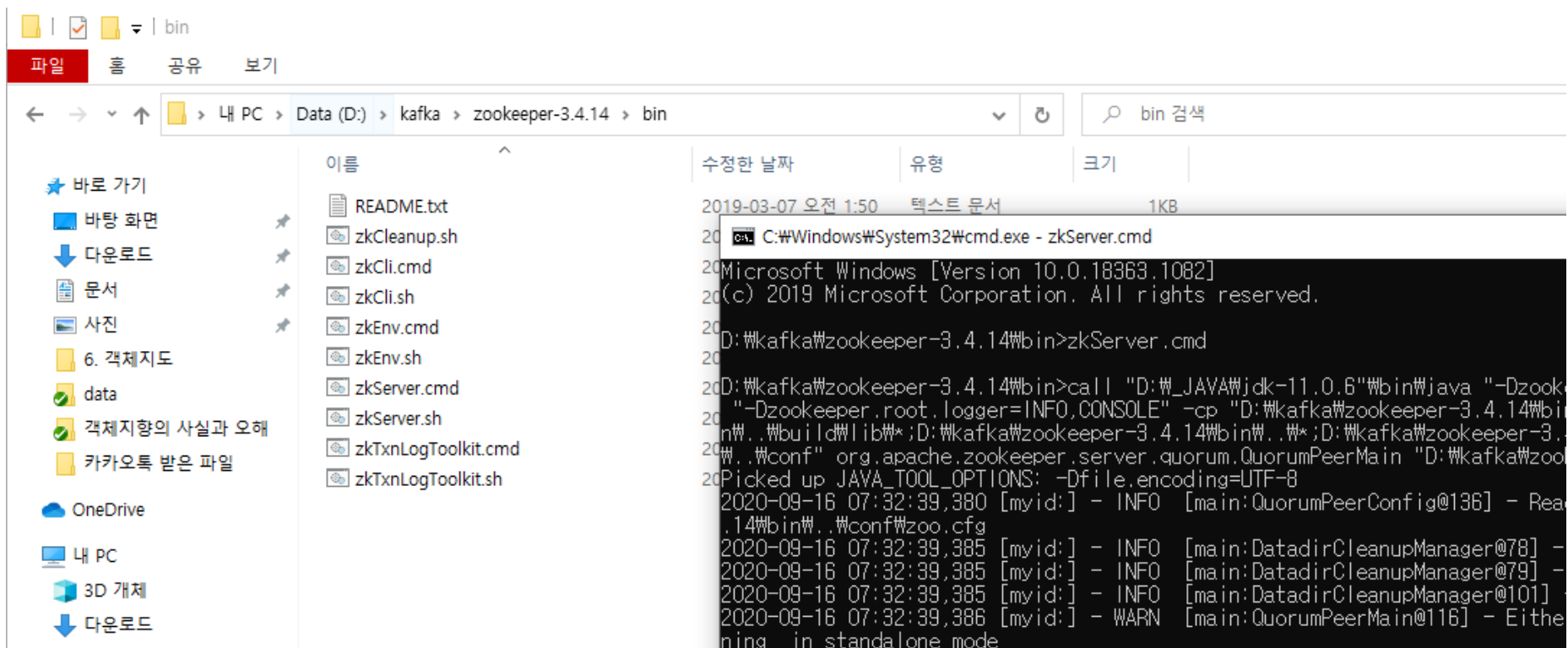
##### Log Basics #####

# A comma separated list of directories under which to store log files
log.dirs=D:\kafka\logs

# The default number of log partitions per topic. More partitions allow
# parallelism for consumption, but this will also result in more files
# the brokers.
num.partitions=1
```

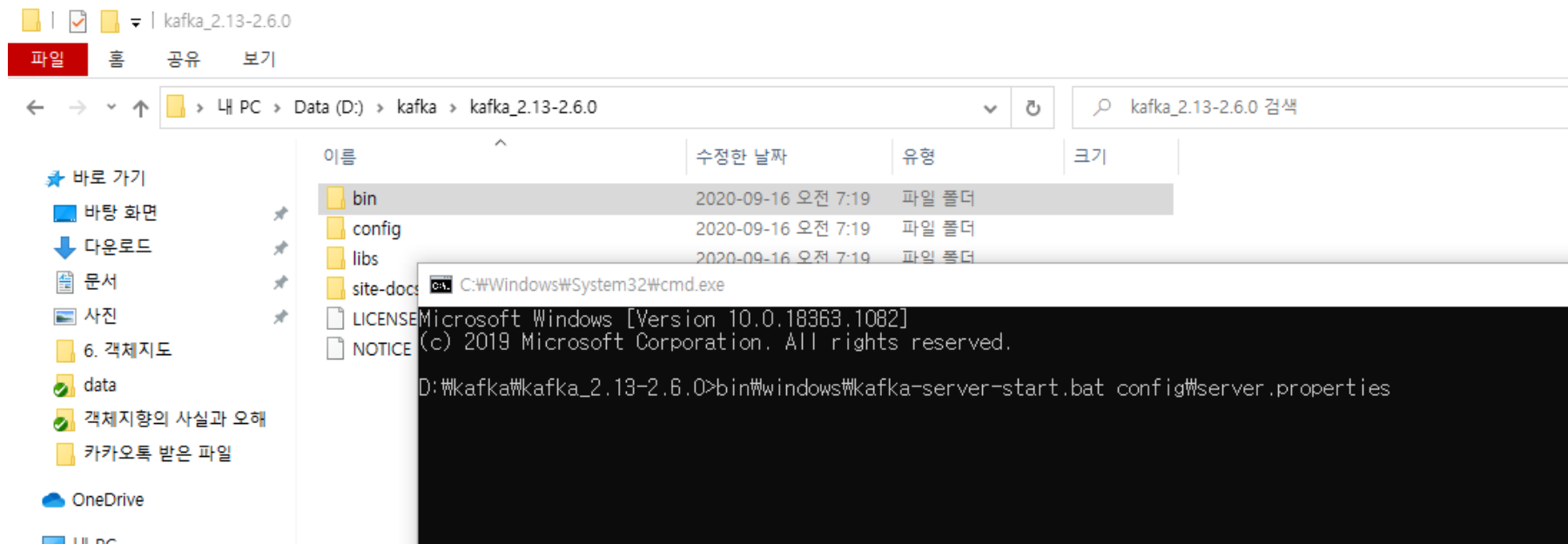
Windows 환경에서 binary 설치

6. 주키퍼 실행



Windows 환경에서 binary 설치

7. 카프카 실행



Windows 환경에서 binary 설치

8. 카프카 예제

8-1. 카프카 토픽 생성

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\kafka\kafka_2.13-2.6.0>bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic test
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Created topic test.

D:\kafka\kafka_2.13-2.6.0>
```

8-2. 카프카 토픽 리스트 조회

```
C:\Windows\System32\cmd.exe
D:\kafka\kafka_2.13-2.6.0>bin\windows\kafka-topics.bat --list --zookeeper localhost:2181
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
test

D:\kafka\kafka_2.13-2.6.0>
```

Windows 환경에서 binary 설치

8-3. 카프카 컨슈머 시작

```
C:\Windows\System32\cmd.exe - bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test

D:\kafka\kafka_2.13-2.6.0>bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
```

8-4. 카프카 프로듀서 시작 및 메시지 전송

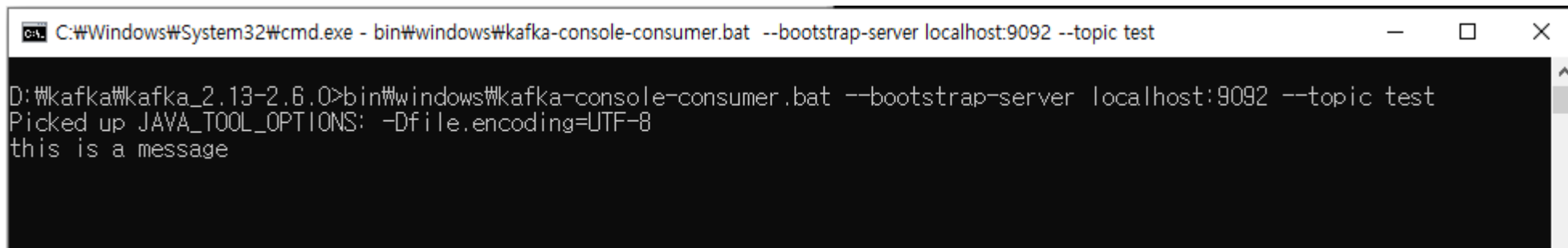
```
C:\Windows\System32\cmd.exe - bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic test

Microsoft Windows [Version 10.0.18363.1082]
(c) 2019 Microsoft Corporation. All rights reserved.

D:\kafka\kafka_2.13-2.6.0>bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic test
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
>this is a message
>
```


Windows 환경에서 binary 설치

8-5. 컨슈머에서 브로커에 쌓인 메시지 PULL



```
C:\Windows\System32\cmd.exe - bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test

D:\kafka\kafka_2.13-2.6.0>bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
this is a message
```

Docker로 설치 – 단일 서버

docker-compose.yml 에 설치 설정 정보를
입력하여 기동하는 방법도 있습니다.

Linux와 windows에 docker가 설치된 환경에서
실행 가능합니다.

```
version: "2.4"

services:
  zookeeper:
    image: wurstmeister/zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    volumes:
      - "zookeeper_data:/wurstmeister"
  kafka:
    image: wurstmeister/kafka
    container_name: kafka
    ports:
      - "9092:9092"
    environment:
      KAFKA_LISTENERS: "INTERNAL://:29092,EXTERNAL://:9092"
      KAFKA_ADVERTISED_LISTENERS: "INTERNAL://kafka:29092,EXTERNAL://172.16.100.168:9092"
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT"
      KAFKA_INTER_BROKER_LISTENER_NAME: "INTERNAL"
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
    volumes:
      - "/var/run/docker.sock:/var/run/docker.sock"
      - "kafka_data:/wurstmeister"
    depends_on:
      - "zookeeper"

volumes:
  zookeeper_data:
    driver: local
  kafka_data:
    driver: local
```

Docker로 설치 - 클러스터링 3대 구성

docker-compose.yml 에 zookeeper 3대 설정 정보 추가

```
services:
  zookeeper1:
    image: wurstmeister/zookeeper
    container_name: zookeeper1
    ports:
      - "2181:2181"
    volumes:
      - "zookeeper_data:/wurstmeister"
  zookeeper2:
    image: wurstmeister/zookeeper
    container_name: zookeeper2
    ports:
      - "2182:2182"
    volumes:
      - "zookeeper_data:/wurstmeister"
  zookeeper3:
    image: wurstmeister/zookeeper
    container_name: zookeeper3
    ports:
      - "2183:2183"
    volumes:
      - "zookeeper_data:/wurstmeister"
```

Docker로 설치 - 클러스터링 3대 구성

docker-compose.yml 에 kafka 3대 설정 정보 추가

```
kafka2:
  image: wurstmeister/kafka
  container_name: kafka2
  ports:
    - "9092:9092"
  environment:
    KAFKA_LISTENERS: "INTERNAL://:29092,EXTERNAL://:9092"
    KAFKA_ADVERTISED_LISTENERS: "INTERNAL://kafka2:29092,EXTERNAL://localhost:9092"
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT"
    KAFKA_INTER_BROKER_LISTENER_NAME: "INTERNAL"
    KAFKA_ZOOKEEPER_CONNECT: "zookeeper1:2181,zookeeper2:2182,zookeeper3:2183"
    KAFKA_DEFAULT_REPLICATION_FACTOR: "3"
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: "3"
    KAFKA_NUM_PARTITIONS: "3"
    KAFKA_BROKER_ID: "2"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
    - "kafka_data:/wurstmeister"
  depends_on:
    - "zookeeper1"
    - "zookeeper2"
    - "zookeeper3"

kafka3:
  image: wurstmeister/kafka
  container_name: kafka3
  ports:
    - "9093:9093"
  environment:
    KAFKA_LISTENERS: "INTERNAL://:29092,EXTERNAL://:9093"
    KAFKA_ADVERTISED_LISTENERS: "INTERNAL://kafka3:29092,EXTERNAL://localhost:9093"
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT"
    KAFKA_INTER_BROKER_LISTENER_NAME: "INTERNAL"
    KAFKA_ZOOKEEPER_CONNECT: "zookeeper1:2181,zookeeper2:2182,zookeeper3:2183"
    KAFKA_DEFAULT_REPLICATION_FACTOR: "3"
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: "3"
    KAFKA_NUM_PARTITIONS: "3"
    KAFKA_BROKER_ID: "3"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
    - "kafka_data:/wurstmeister"
  depends_on:
    - "zookeeper1"
    - "zookeeper2"
    - "zookeeper3"
```

```
kafka4:
  image: wurstmeister/kafka
  container_name: kafka4
  ports:
    - "9094:9094"
  environment:
    KAFKA_LISTENERS: "INTERNAL://:29092,EXTERNAL://:9094"
    KAFKA_ADVERTISED_LISTENERS: "INTERNAL://kafka4:29092,EXTERNAL://localhost:9094"
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT"
    KAFKA_INTER_BROKER_LISTENER_NAME: "INTERNAL"
    KAFKA_ZOOKEEPER_CONNECT: "zookeeper1:2181,zookeeper2:2182,zookeeper3:2183"
    KAFKA_DEFAULT_REPLICATION_FACTOR: "3"
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: "3"
    KAFKA_NUM_PARTITIONS: "3"
    KAFKA_BROKER_ID: "4"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
    - "kafka_data:/wurstmeister"
  depends_on:
    - "zookeeper1"
    - "zookeeper2"
    - "zookeeper3"
```

이 장의 내용

- ✓ 컨슈머 그룹
- ✓ 오프셋 커밋
- ✓ 파티션 재배포

컨슈머 그룹이란?

- ✓ 컨슈머는 컨슈머 그룹이라 불리는 하나 이상의 컨슈머들로 이루어진 그룹을 형성하여 메시지를 얻는다.
- ✓ 컨슈머 그룹은 Group ID라는 ID로 구분된다.
- ✓ 이 Group ID는 KafkaConsumer를 생성할 때 지정하는 옵션으로 group.id라는 파라미터로 지정한다.
- ✓ 컨슈머 그룹은 Group ID가 동일한 컨슈머끼리 형성된다.
- ✓ 카프카 클러스터에서 수신할 메시지는 컨슈머 그룹 안에서 하나의 컨슈머가 수신한다. (분산 처리)
- ✓ 이러한 특성으로 컨슈머 그룹은 하나의 데이터 처리를 여러 컨슈머에서 분산 처리하기 위해 사용된다.

오프셋 커밋이란?

- 카프카를 사용하는 시스템에서는 컨슈머가 카프카 클러스터에서 메시지를 얻어 처리한다.
- 이 때 컨슈머는 어느 메시지까지 처리를 완료했는지 카프카 클러스터에 기록을 남길 수 있다.
- 이 기록을 남기는 처리를 오프셋 커밋이라고 한다.
- 오프셋 커밋의 기록은 컨슈머 그룹 단위로 이루어진다.
- 컨슈머 그룹마다 각 토픽의 파티션에서 어느 오프셋까지 처리 완료했는지 정보를 기록한다.
- 오프셋 커밋 정보에 의해 컨슈머는 카프카에서 메시지 수신 처리를 재개할 때 어떤 메시지부터 재개해야 하는지 알 수 있어 장애에 대한 대처가 가능하다.
- 커밋된 오프셋 정보는 `_consumer_offsets` 라는 전용 토픽에 기록된다.
- 오프셋 커밋 방법에는 두 가지가 있다. Auto Offset Commit과 Manual Offset Commit

자동 오프셋 커밋

- 일정 간격마다 자동으로 오프셋 커밋을 하는 방식이다.
- 컨슈머의 옵션 `enable.auto.commit`을 `true`로 설정하여 사용할 수 있다.
- 오프셋 커밋의 간격은 `KafkaConsumer`의 옵션 `auto.commit.interval.ms`로 지정할 수 있고, 기본은 5초다
- 컨슈머에 장애가 발생했을 때 메시지가 손실되거나 여러 메시지의 재처리(중복)가 발생할 수 있는 것이 단점이다.

수동 오프셋 커밋

- 수동 오프셋 커밋은 컨슈머 애플리케이션 안에서 `KafkaConsumer`의 `commitSync` 또는 `commitAsync`라는 메서드를 이용하여 오프셋 커밋을 실행한다.
- 메시지 손실을 발생하지 않도록 하는 것이 장점이다.
- 컨슈머 장애 시 메시지 중복 최소화

자동 오프셋 리셋

- 컨슈머는 앞서 언급한 오프셋 커밋 정보를 참조하여 메시지 처리를 시작할 오프셋을 결정한다.
- 그러나 시작할 때 오프셋 커밋 기록이 존재하지 않는 경우나 기록되어 있는 오프셋이 유효하지 않은 경우는 지정된 정책에 따라 초기화를 실시하여 메시지 처리를 시작할 오프셋을 결정한다.
- 이것이 자동 오프셋 리셋이다.
- `auto.offset.reset` 이라는 옵션으로 지정한다.

정책	자동 오프셋 리셋의 동작
latest	해당 파티션의 가장 새로운 오프셋으로 초기화된다. 따라서 카프카 클러스터에 이미 존재하는 메시지는 처리되지 않는다.
earliest	해당 파티션에 존재하는 가장 오래된 오프셋으로 초기화된다. 카프카 클러스터에 이미 존재하는 메시지 모두에 대해 처리를 실시한다.
none	유효한 오프셋 커밋 정보가 없는 경우에 예외를 반환한다.

파티션 재배치

- 카프카에서 파티션은 하나 이상의 복제본(replica)를 가지며, 카프카 클러스터 중 어느 하나의 브로커에 보관되어 있다.
- 복제본의 배치를 변경하는 주된 이유는 카프카 클러스터의 브로커를 증감시키는 경우다.
- 카프카에서 계획적인 정지나 장애로 인한 정지를 불문하고 브로커가 보유하고 있던 복제본이 다른 브로커로 자동으로 이동하지 않는다.
- 따라서 브로커 수를 항시적으로 줄이는 경우에는 감소시킬 브로커가 보유하고 있는 복제본을 미리 다른 브로커로 이동시켜 필요한 복제본의 수를 확보해야 한다.
- 또한 브로커를 추가하여 클러스터를 확장시킬 경우에도 새로 추가한 브로커에 복제본을 배치하여 메시지의 송수신 부하를 균등하게 배분해야 한다.
- 파티션 재배치로 복제본을 다른 브로커에 배치시키는 경우에 먼저 새로운 복제본을 배치할 브로커에 복제본을 생성해 동기화하고, 동기화를 완료한 후 불필요해진 복제본을 제거한다.
- 따라서 재배치 중에는 일반적인 메시지 송수신에 필요한 처리와 더불어 재배치로 새로 생성되는 복제본의 동기화가 필요하다.

Q&A

Kafka 3대 설치 구성

docker-compose.yml 에 kafka 3대 설정 정보 추가하여 기동

```
kafka2:
  image: wurstmeister/kafka
  container_name: kafka2
  ports:
    - "9092:9092"
  environment:
    KAFKA_LISTENERS: "INTERNAL://:29092,EXTERNAL://:9092"
    KAFKA_ADVERTISED_LISTENERS: "INTERNAL://kafka2:29092,EXTERNAL://localhost:9092"
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT"
    KAFKA_INTER_BROKER_LISTENER_NAME: "INTERNAL"
    KAFKA_ZOOKEEPER_CONNECT: "zookeeper1:2181,zookeeper2:2182,zookeeper3:2183"
    KAFKA_DEFAULT_REPLICATION_FACTOR: "3"
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: "3"
    KAFKA_NUM_PARTITIONS: "3"
    KAFKA_BROKER_ID: "2"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
    - "kafka_data:/wurstmeister"
  depends_on:
    - "zookeeper1"
    - "zookeeper2"
    - "zookeeper3"

kafka3:
  image: wurstmeister/kafka
  container_name: kafka3
  ports:
    - "9093:9093"
  environment:
    KAFKA_LISTENERS: "INTERNAL://:29092,EXTERNAL://:9093"
    KAFKA_ADVERTISED_LISTENERS: "INTERNAL://kafka3:29092,EXTERNAL://localhost:9093"
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT"
    KAFKA_INTER_BROKER_LISTENER_NAME: "INTERNAL"
    KAFKA_ZOOKEEPER_CONNECT: "zookeeper1:2181,zookeeper2:2182,zookeeper3:2183"
    KAFKA_DEFAULT_REPLICATION_FACTOR: "3"
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: "3"
    KAFKA_NUM_PARTITIONS: "3"
    KAFKA_BROKER_ID: "3"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
    - "kafka_data:/wurstmeister"
  depends_on:
    - "zookeeper1"
    - "zookeeper2"
    - "zookeeper3"

kafka4:
  image: wurstmeister/kafka
  container_name: kafka4
  ports:
    - "9094:9094"
  environment:
    KAFKA_LISTENERS: "INTERNAL://:29092,EXTERNAL://:9094"
    KAFKA_ADVERTISED_LISTENERS: "INTERNAL://kafka4:29092,EXTERNAL://localhost:9094"
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: "INTERNAL:PLAINTEXT,EXTERNAL:PLAINTEXT"
    KAFKA_INTER_BROKER_LISTENER_NAME: "INTERNAL"
    KAFKA_ZOOKEEPER_CONNECT: "zookeeper1:2181,zookeeper2:2182,zookeeper3:2183"
    KAFKA_DEFAULT_REPLICATION_FACTOR: "3"
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: "3"
    KAFKA_NUM_PARTITIONS: "3"
    KAFKA_BROKER_ID: "4"
  volumes:
    - "/var/run/docker.sock:/var/run/docker.sock"
    - "kafka_data:/wurstmeister"
  depends_on:
    - "zookeeper1"
    - "zookeeper2"
    - "zookeeper3"
```

Zookeeper 서버 홀수 구성 이유 설명

홀수 구성 이유 설명 URL

설정에 의해 replica 복제되는 것을 로그상에서 확인

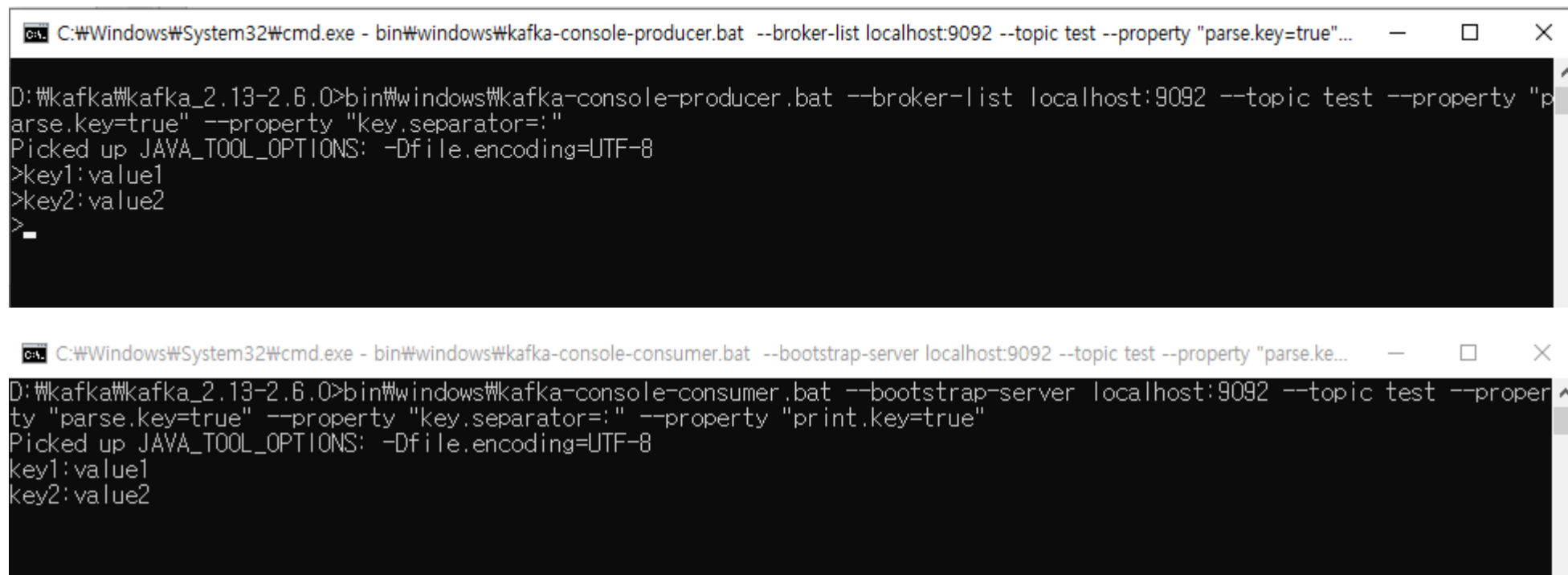
```
D:\kafka\kafka_2.13-2.6.0\bin>windows\kafka-topics.bat --zookeeper localhost:2181, localhost:2182, localhost:2183 --topic kafkaTest --partitions 1 --replication-factor 3 --create
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Created topic kafkaTest.

D:\kafka\kafka_2.13-2.6.0\bin>windows\kafka-topics.bat --zookeeper localhost:2181, localhost:2182, localhost:2183 --topic kafkaTest --describe
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Topic: kafkaTest      PartitionCount: 1      ReplicationFactor: 3      Configs:
  Topic: kafkaTest      Partition: 0      Leader: 4      Replicas: 4,2,3 Isr: 4,2,3
```

Kafka-console-producer.sh를 실행 후 메시지를 보내면 value값으로 전송되는데 key와 value를 동시에 보내려면 어떻게 해야하나요?

Producer, Consumer 실행 옵션에 추가

- parsing.key : key와 value 파싱 활성화 여부
- key.separator : key와 value 파싱 구분자
- print.key : console 창 출력 여부



```
C:\Windows\System32\cmd.exe - bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic test --property "parse.key=true"...
```

```
D:\kafka\kafka_2.13-2.6.0>bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic test --property "p
arse.key=true" --property "key.separator=:"
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
>key1:value1
>key2:value2
>
```

```
C:\Windows\System32\cmd.exe - bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --property "parse.ke...
```

```
D:\kafka\kafka_2.13-2.6.0>bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic test --proper
ty "parse.key=true" --property "key.separator=:" --property "print.key=true"
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
key1:value1
key2:value2
```

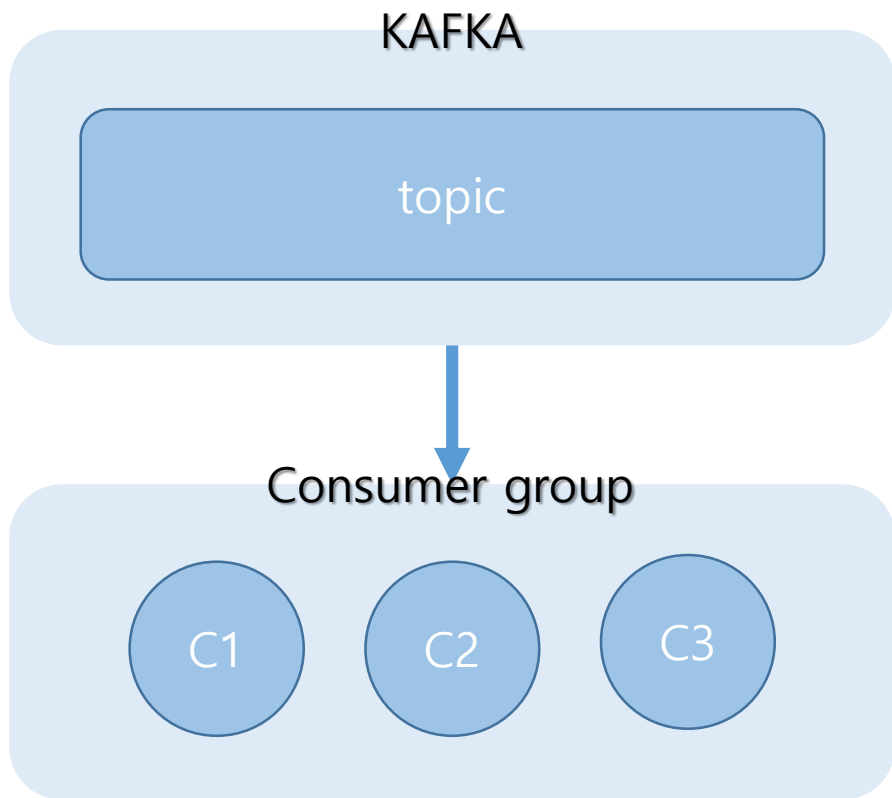

Kafka에서 zookeeper라는게 있는데 어떤 역할을 하는지 쉽게 설명해줄 수 있나요?

분산 시스템을 설계하다보면, 가장 문제점 중의 하나가 분산된 시스템간의 정보를 어떻게 공유할 것이고, 클러스터에 있는 서버들의 상태를 체크할 필요가 있다.

Zookeeper

- 분산 메시지 큐의 정보를 관리해 주는 역할을 합니다.
- Kafka는 주로 다중 브로커로 구성되기 때문에 kafka를 띄우기 위해서는 zookeeper가 반드시 실행되어야 합니다.
- Cluster 상태에 대해 카프카 큐의 producer나 consumer에게 통보합니다.
- 어떤 브로커가 어떤 topic partition에 대해서 leader인지 기록하고, 이러한 정보를 producer나 consumer가 메시지를 읽고 쓸 수 있게 제공합니다.

책 325p에 *consumer group* 설명이 나오는데 *consumer group*을 사용하지 않으면 무슨 문제가 있을까요? 예를 들면 *ConsumerA*와 *ConsumerB*가 *consumer group*이 없는 경우와 있는 경우에 다른점은 무엇인가요?



1. Consumer 1에서 장애가 발생하더라도, 동일 Consumer group 내의 다른 Consumer가 계속해서 읽을 수 있도록 리밸런싱 합니다. Partition 하나 당 customer는 단 한 개만 할당될 수 있습니다.
2. 오프셋 기능 사용 가능

partitions 2개, **userMessage** Topic 생성

```
D:\kafka\kafka_2.13-2.6.0>bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 2 --topic userMessage
Picked up JAVA_TOOL_OPTIONS: -Dfile.encoding=UTF-8
Created topic userMessage.
```

Producer message 전송 전 partitioner.class config 설정 추가

```
public class TestProducer {  
    public static void main(String[] args) {  
  
        String brokers = "localhost:9092";  
        String topic = "userMessage";  
  
        Properties configs = new Properties();  
        configs.put("bootstrap.servers", brokers);  
        configs.put("acks", "all");  
        configs.put("retries", 0);  
        configs.put("batch.size", 16384);  
        configs.put("linger.ms", 1);  
        configs.put("buffer.memory", 33554432);  
        configs.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");  
        configs.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");  
        configs.put("partitioner.class", "spectra.kafka.test.CustomPatitioner");  
  
        KafkaProducer<String, String> producer = new KafkaProducer<String, String>(configs);  
  
        System.out.println("Produces messages");  
  
        String userName = "shjeon-1";  
        producer.send(new ProducerRecord<String, String>(topic, userName), new Callback() {  
            public void onCompletion(RecordMetadata metadata, Exception e) {  
                if (e != null) {  
                    // ...  
                }  
            }  
        });  
    }  
}
```

CustomPartitioner 구현

```
public class CustomPartitioner implements Partitioner {  
    @Override  
    public int partition(String topic, Object o, byte[] bytes, Object o1, byte[] bytes1, Cluster cluster) {  
        List<PartitionInfo> partitions = cluster.partitionsForTopic(topic);  
  
        int numPartitions = partitions.size();  
  
        System.out.println("partition size : " + numPartitions);  
  
        // TODO : 파티션 선택 로직  
        return 0;  
    }  
}
```

userMessage partition 2개로 설정하였음 => 0 또는 1로 셋팅

결과1) partition size, producer partition 출력 (partition 1로 전송)

```
Run: TestProducer x
D:\_JAVA\jdk1.8.0_65\bin\java.exe ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Produces messages
partition size : 2
User: shjeon-2, Partition: 1
```

```
TestConsumer x
D:\_JAVA\jdk1.8.0_65\bin\java.exe ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Receive message: shjeon-2, Partition: 1, Offset: 0
```

결과1) partition size, producer partition 출력 (partition 0으로 전송)

```
Run: TestConsumer x TestProducer x
D:\_JAVA\jdk1.8.0_65\bin\java.exe ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Produces messages
partition size : 2
User: shjeon-1, Partition: 0
```

```
Run: TestConsumer x TestProducer x
D:\_JAVA\jdk1.8.0_65\bin\java.exe ...
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Receive message: shjeon-2, Partition: 1, Offset: 0
Receive message: shjeon-1, Partition: 0, Offset: 1
|
```

감사합니다.