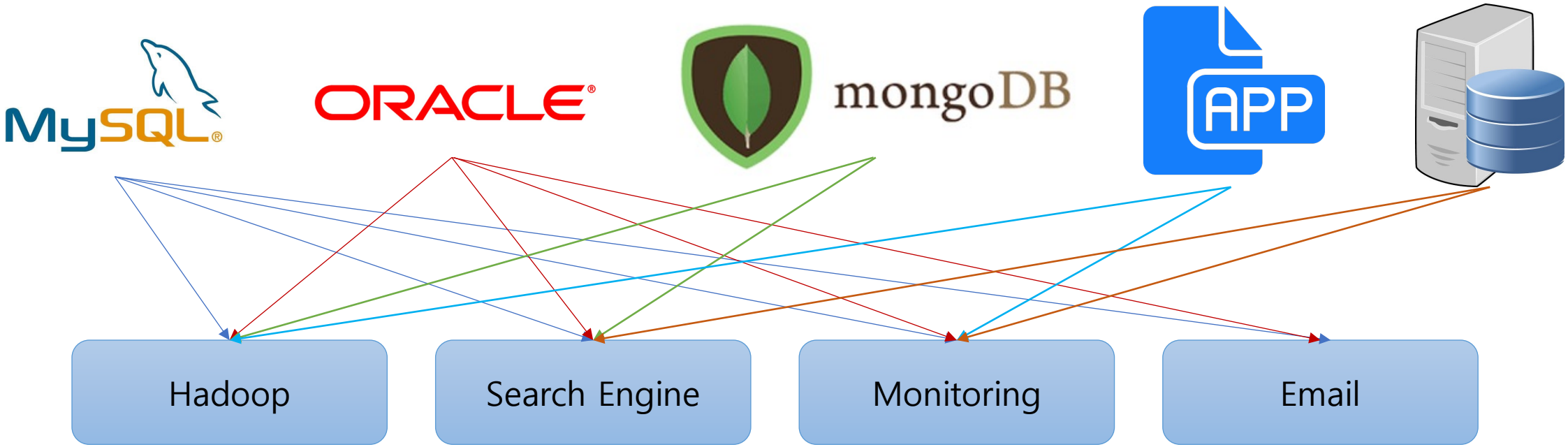

APACHE
kafka®

목차

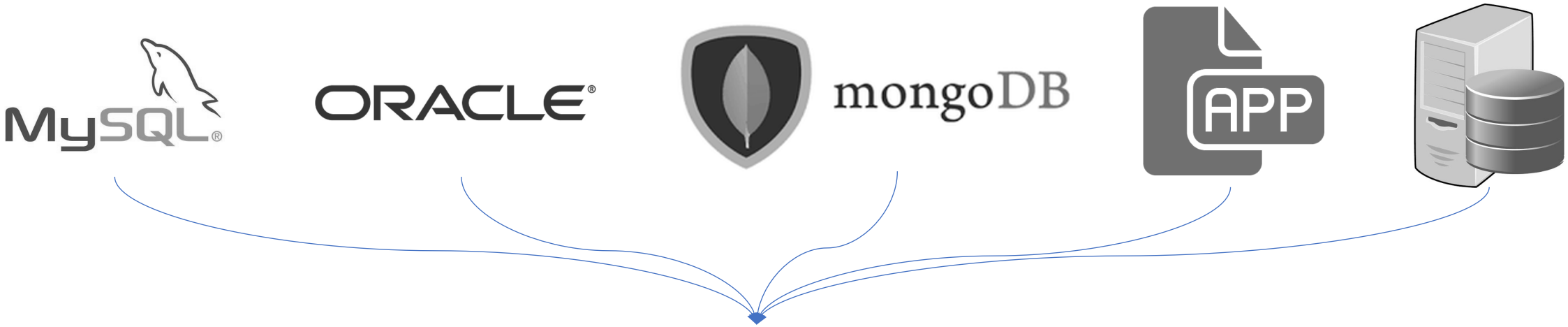
Kafka 탄생 배경



- End-to-End 연결 방식의 아키텍처
- 데이터 연동의 복잡성 증가 (HW, 운영체제, 장애 등)
- 서로 다른 데이터 Pipeline 연결 구조
- 확장이 어려운 구조

- ***모든 시스템으로 데이터를 실시간으로 전송하여 처할 수 있는 시스템***
- ***데이터가 많아지더라도 확장이 용이한 시스템***

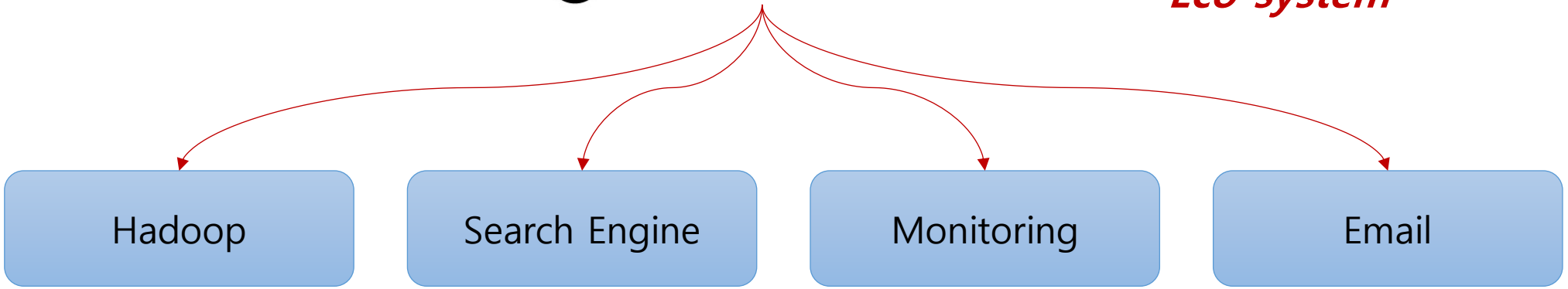
Kafka 데이터 처리 흐름



- *Producer/Consumer 분리*
- *메시지를 여러 Consumer 에게 허용*

APACHE kafka®

- *높은 처리량을 위한 메시지 최적화*
- *Scale-out 가능*
- *Eco-system*



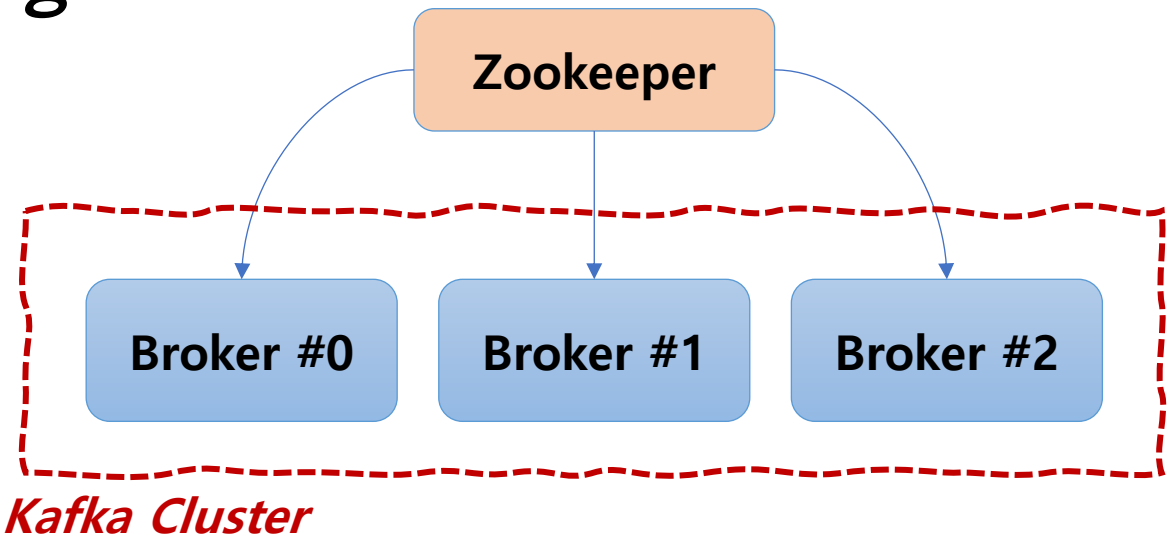
Apache Kafka

- Apache Software Foundation의 Scalar 언어로 된 ***오픈 소스 메시지 브로커 프로젝트(Open Source Message Broker Project)***
- 링크드인(Linked-in)에서 개발, 2011년 오픈 소스화
  - 2014년 11월 링크드인에서 Kafka를 개발하던 엔지니어들이 Kafka개발에 집중하기 위해 Confluent라는 회사 창립
- 실시간 데이터 피드를 관리하기 위해 통일된, 높은 처리량, 낮은 지연 시간을 지닌 플랫폼 제공

- *Apple, Netflix, Shopify, Yelp, Kakao, New York Times, Cisco, Ebay, Paypal, Hyperledger Fabric, Uber, Salesforce.com 등이 사용*

Kafka Broker

- 실행 된 Kafka 애플리케이션 서버 중 1대
- 3대 이상의 Broker Cluster 구성
- Zookeeper 연동
  - Zookeeper 역할
    - 메타데이터 (Broker ID, Controller ID 등) 저장
- n개 Broker 중 1대는 Controller 기능 수행
  - Controller 역할
    - 각 Broker에게 담당 파티션 할당 수행
    - Broker 정상 동작 모니터링 관리
    - Zookeeper에게 저장됨



Zookeeper 사용용도

- **설정 관리(Configuration management)**
  - 클러스터의 설정 정보를 최신으로 유지하기 위한 조율 시스템으로 사용
- **클러스터 관리(Cluster management)**
  - 클러스터의 서버가 추가되거나 제외될 때 그 정보를 클러스터 안 서버들이 공유하는 데 사용
- **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 사용
- **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.

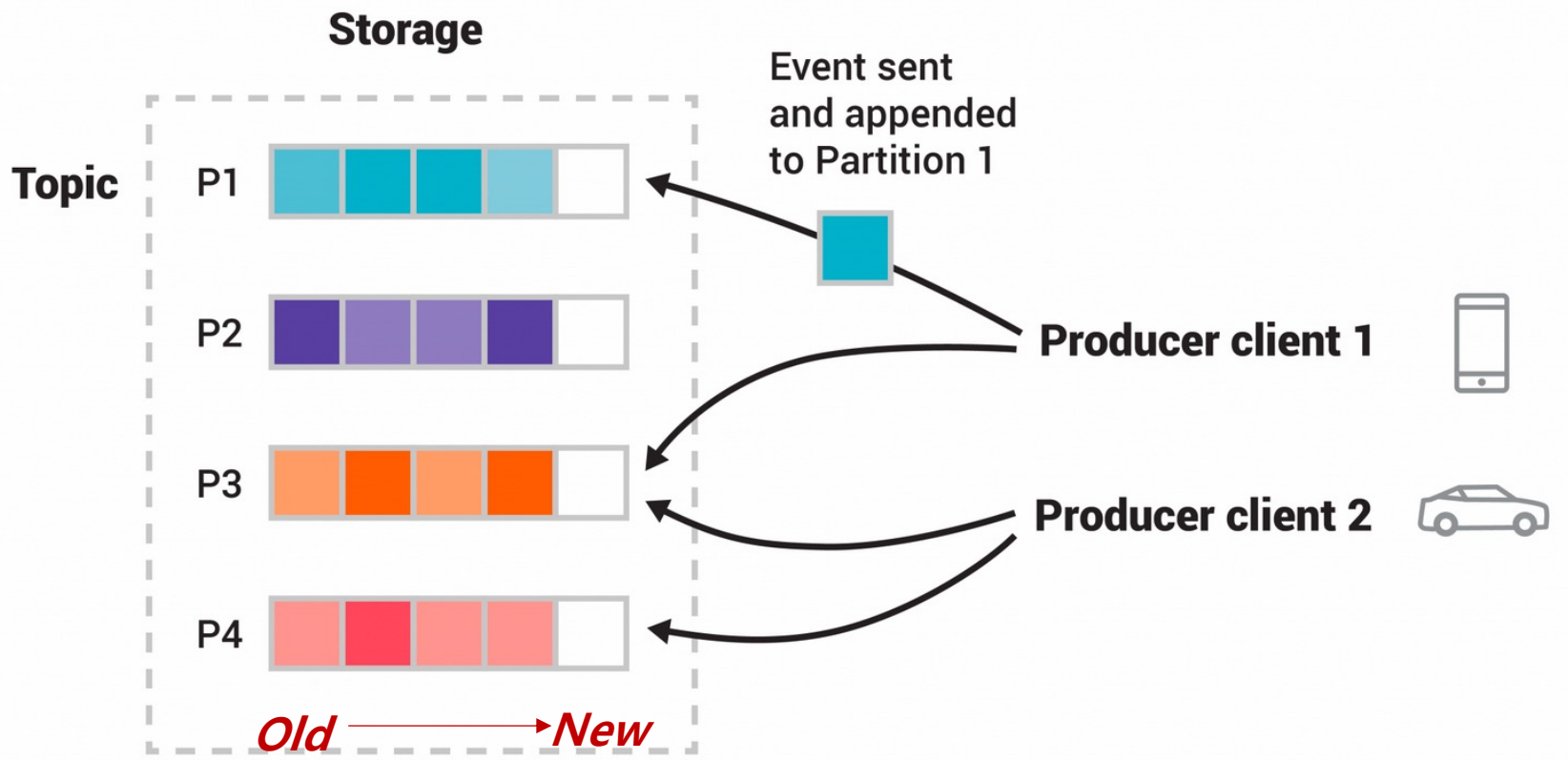
- **Zookeeper 사용처**
  - **Apache Hbase**
    - 클러스터 마스터를 선출하기 위해 사용
    - 현재 이용 가능한 서버가 어떤 것들이 있는지 정보를 저장
    - 클러스터의 메타 데이터를 보관하는데 사용
  - **Apache Kafka**
    - Kafka 서버의 크래시를 감지
    - 새로운 토픽이 생성되었을 때, 토픽의 생성과 소비에 대한 상태를 저장
  - **Facebook Messages**
    - 이메일, SMS, 페이스북 챗 등을 통합한 페이스북 메시지에서 샤딩과 페일오더 컨트롤러의 구현

Record

```
new ProducerRecord<String, String>("topic", "key", "message");


ConsumerRecords<String, String> records = consumer.poll(1000);
for (ConsumerRecord<String, String> record : records) {
        ...
}
```

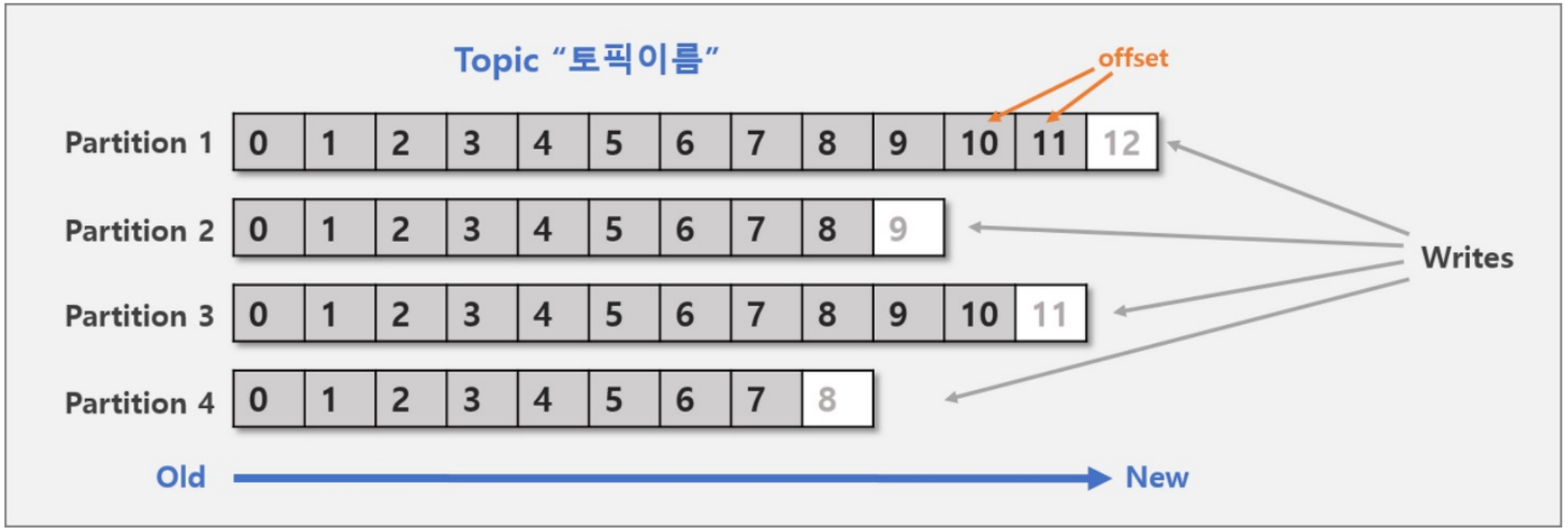
- 객체를 Producer에서 Consumer로 전달하기 위해 Kafka 내부에 byte 형태로 저장할 수 있도록 직렬화/역직렬화하여 사용
- 기본 제공 직렬화 Class: StringSerializer, SHortSerializer 등
- 커스텀 직렬화 Class 사용 가능: Customer Object 직렬화/역직렬화 가능

Topic & Partition



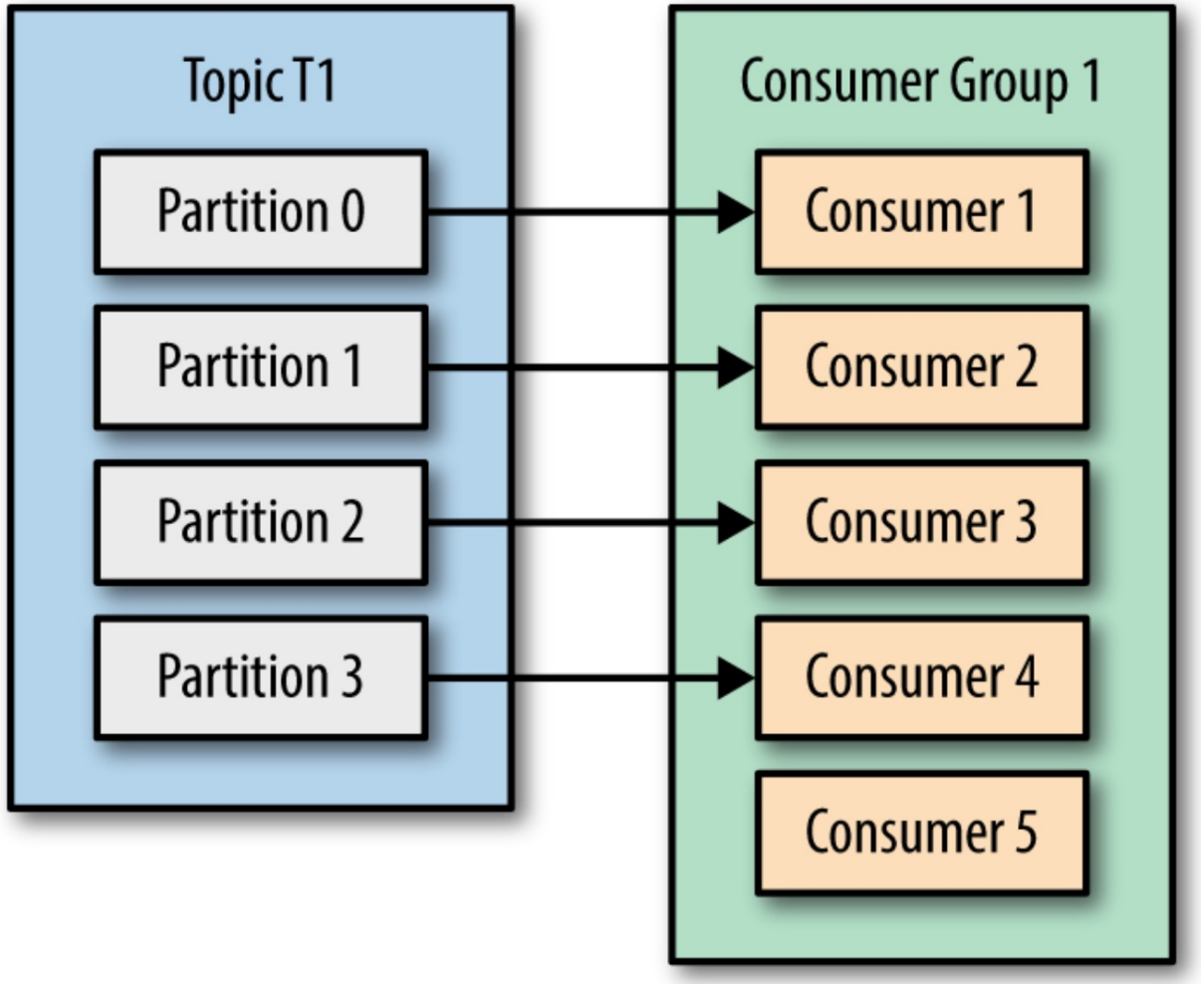
- 메시지 분류 단위
- n개의 파티션 할당 가능
- 각 파티션마다 고유한 Offset을 가짐
- 메시지 처리 순서는 파티션 별로 유지 관리 됨

Offset



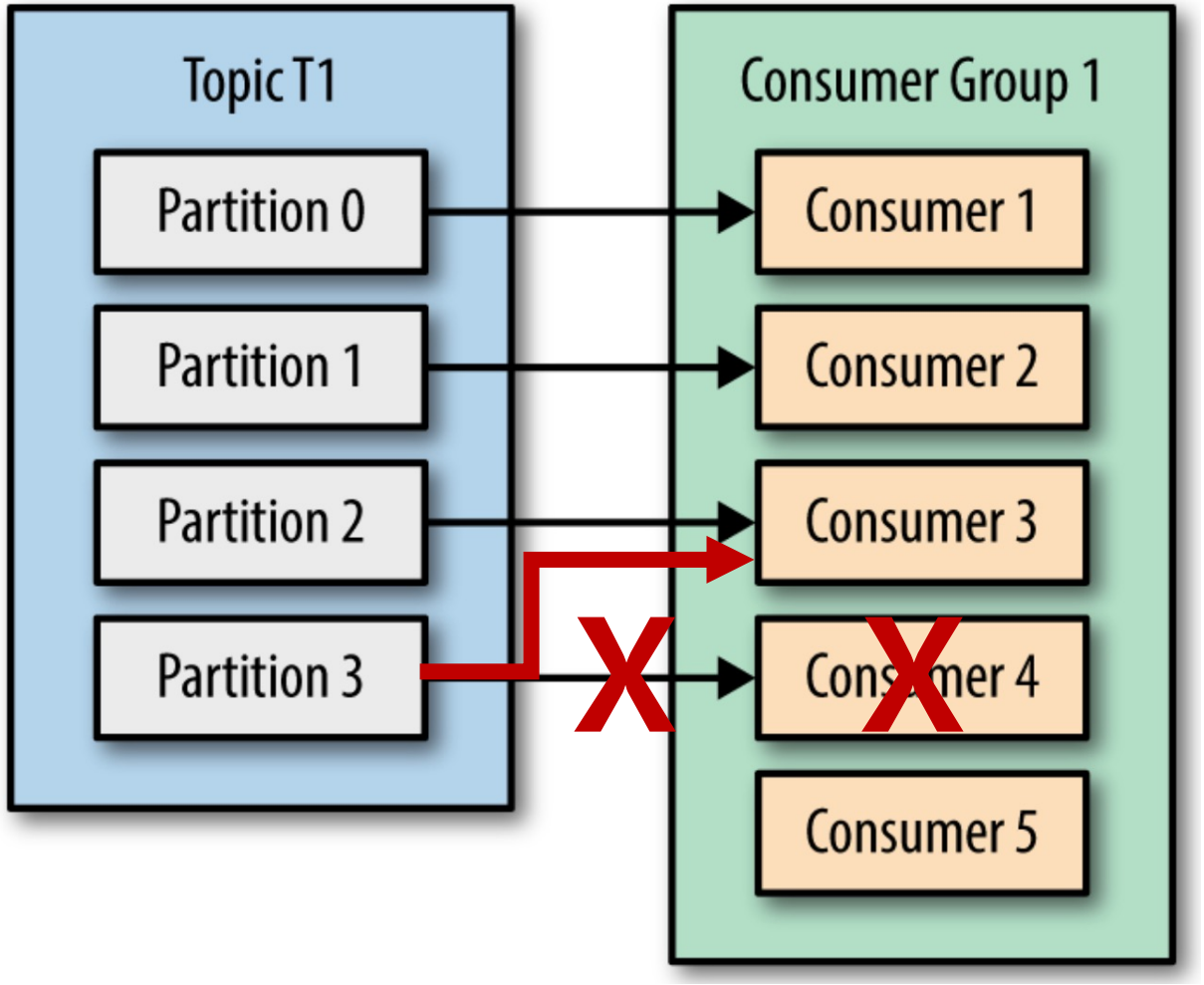
- 파티션 내에서 데이터의 위치를 표시하는 유니크한 숫자
- Consumer는 자신이 어디까지 데이터를 가져갔는지 Offset을 이용하여 관리

파티션 4개인 Topic과 Consumer 5개



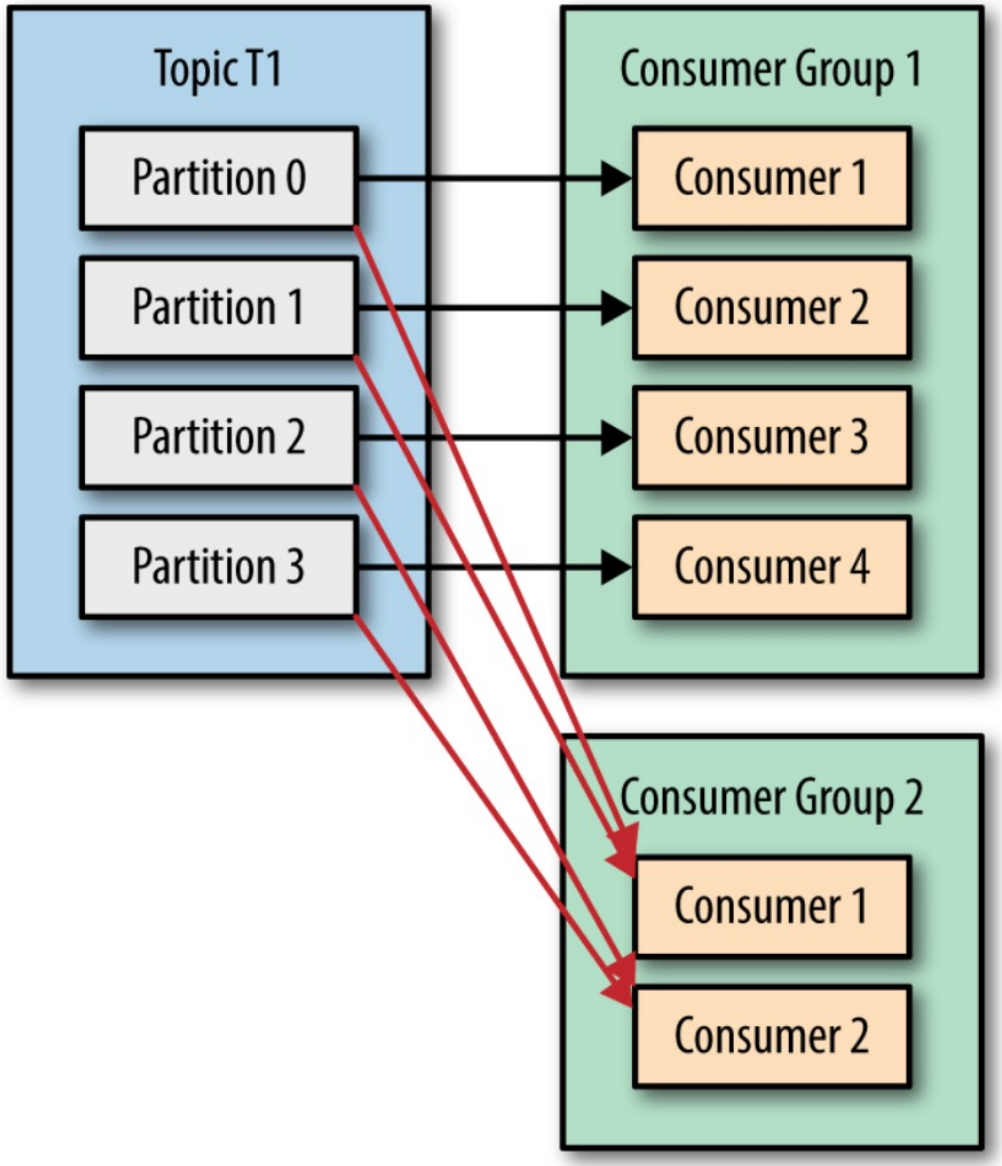
- 가능한 경우: 파티션 개수 >= Consumer 개수
- 불가능한 경우: 파티션 개수 < Consumer 개수
  - 남은 Consumer는 파티션을 할당 받지 못하고 대기 중

Consumer 5개 중 1대 장애 발생



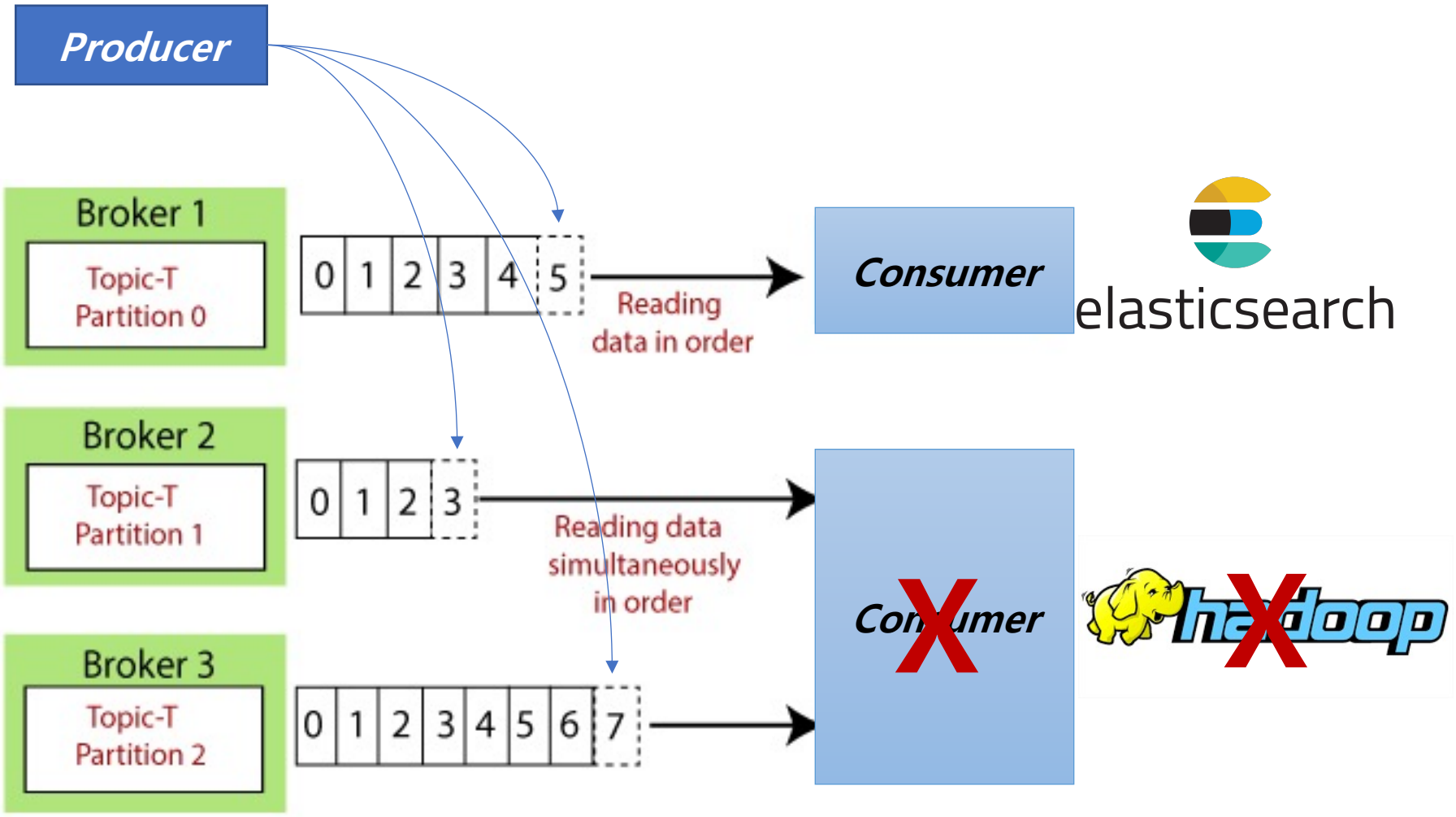
- *Consumer 중 한개가 장애가 발생 한 경우에 대한 대비 가능*
- *Rebalance 발생: 파티션 Consumer 할당 재조정*
- *나머지 Consumer가 파티션으로부터 Polling 수행*

2개 이상의 Consumer Group



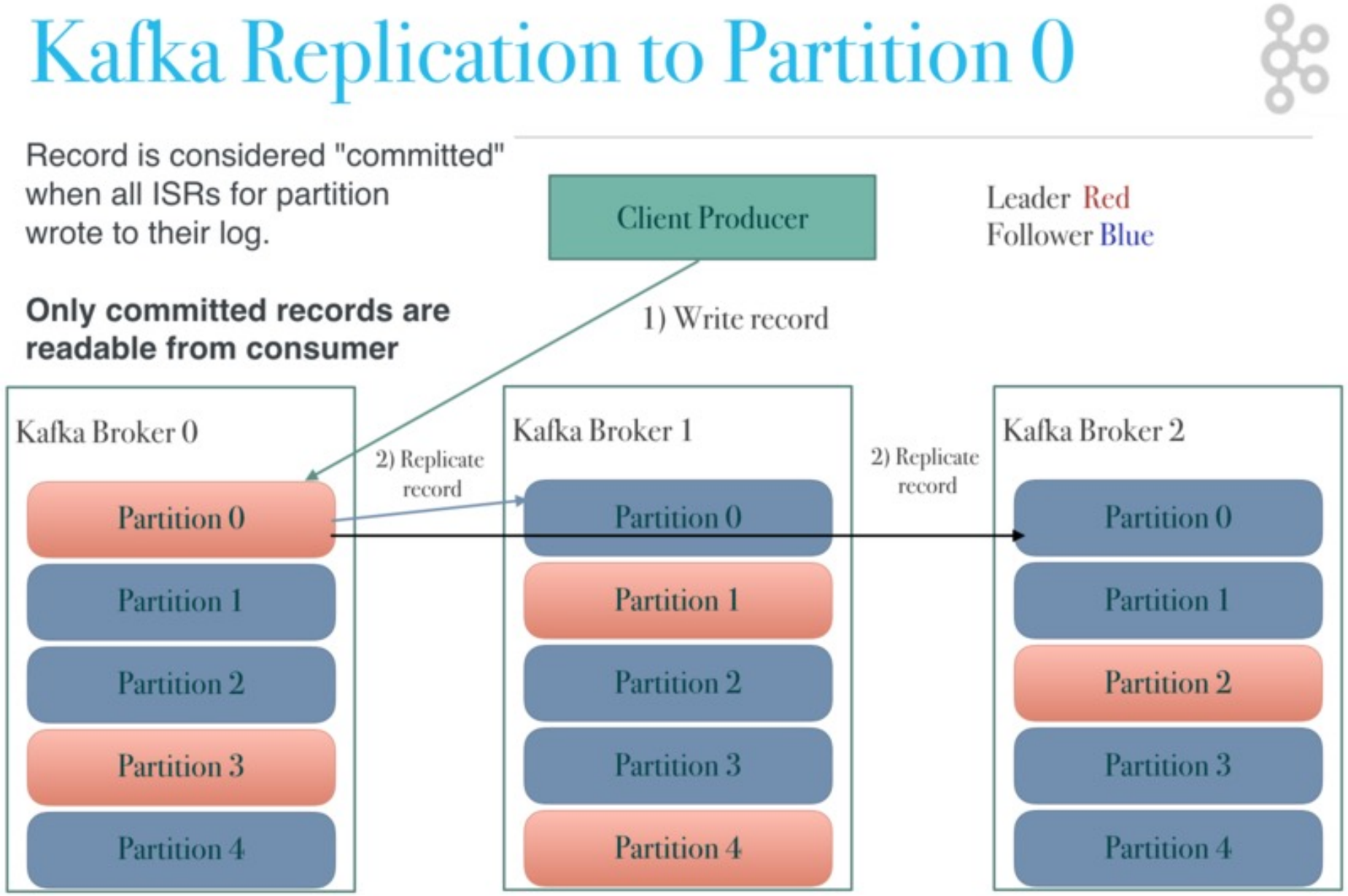
- 목적에 따른 Consumer Group을 분리할 수 있음
- 장애에 대응하기 위해 재입수(또는 재처리) 목적으로 임시 신규 Consumer Group을 생성하여 사용하기도 함

Consumer Group 장애에 격리되는 다른 Consumer Group



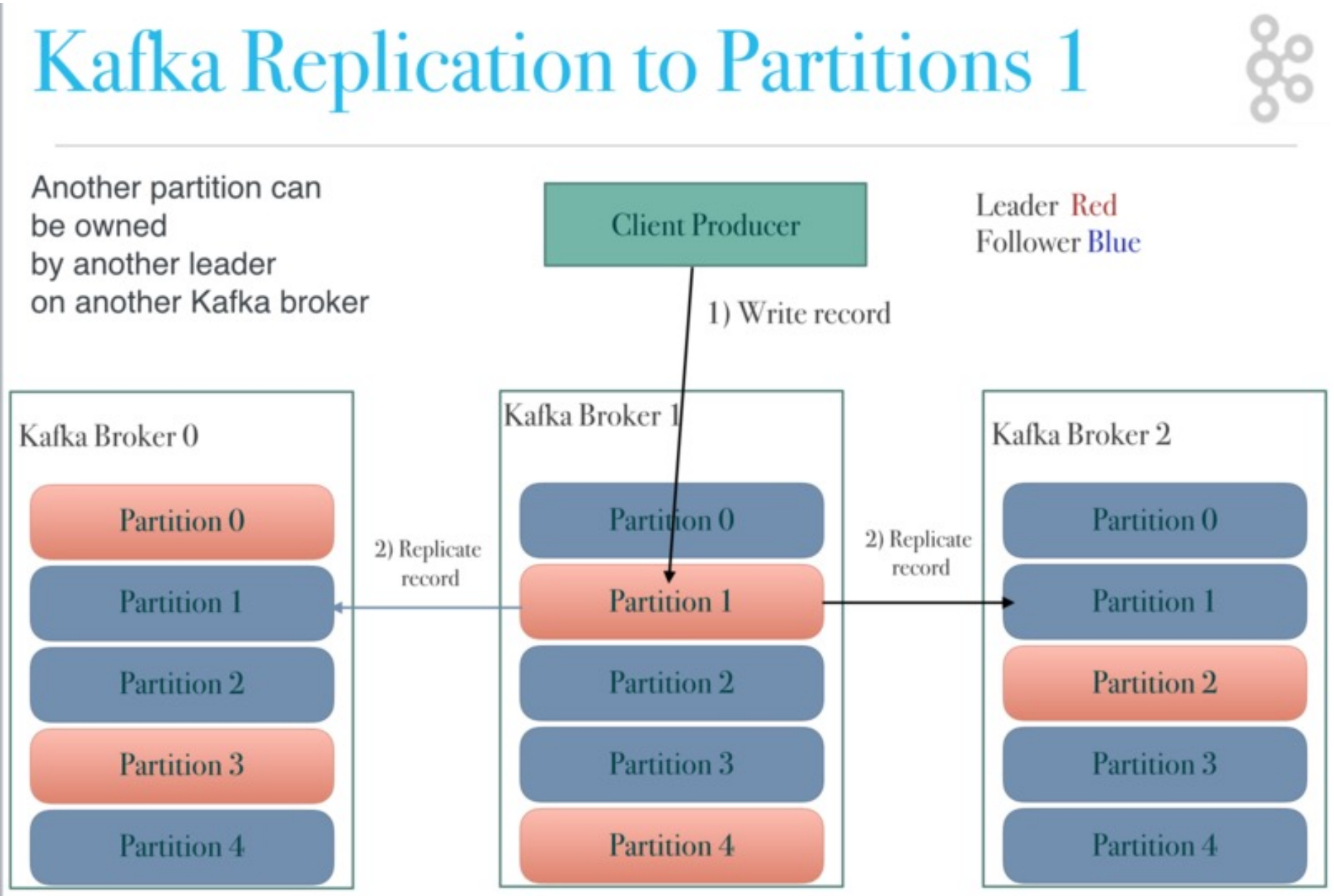
- *Consumer Group 간 간섭 줄임*
- *ex) Hadoop에 이슈가 발생하여 Consumer의 적재 지연이 발생하더라도 Elastic Search에 적재하는 Consumer의 동작에는 이슈가 없음*

Broker Partition Replication



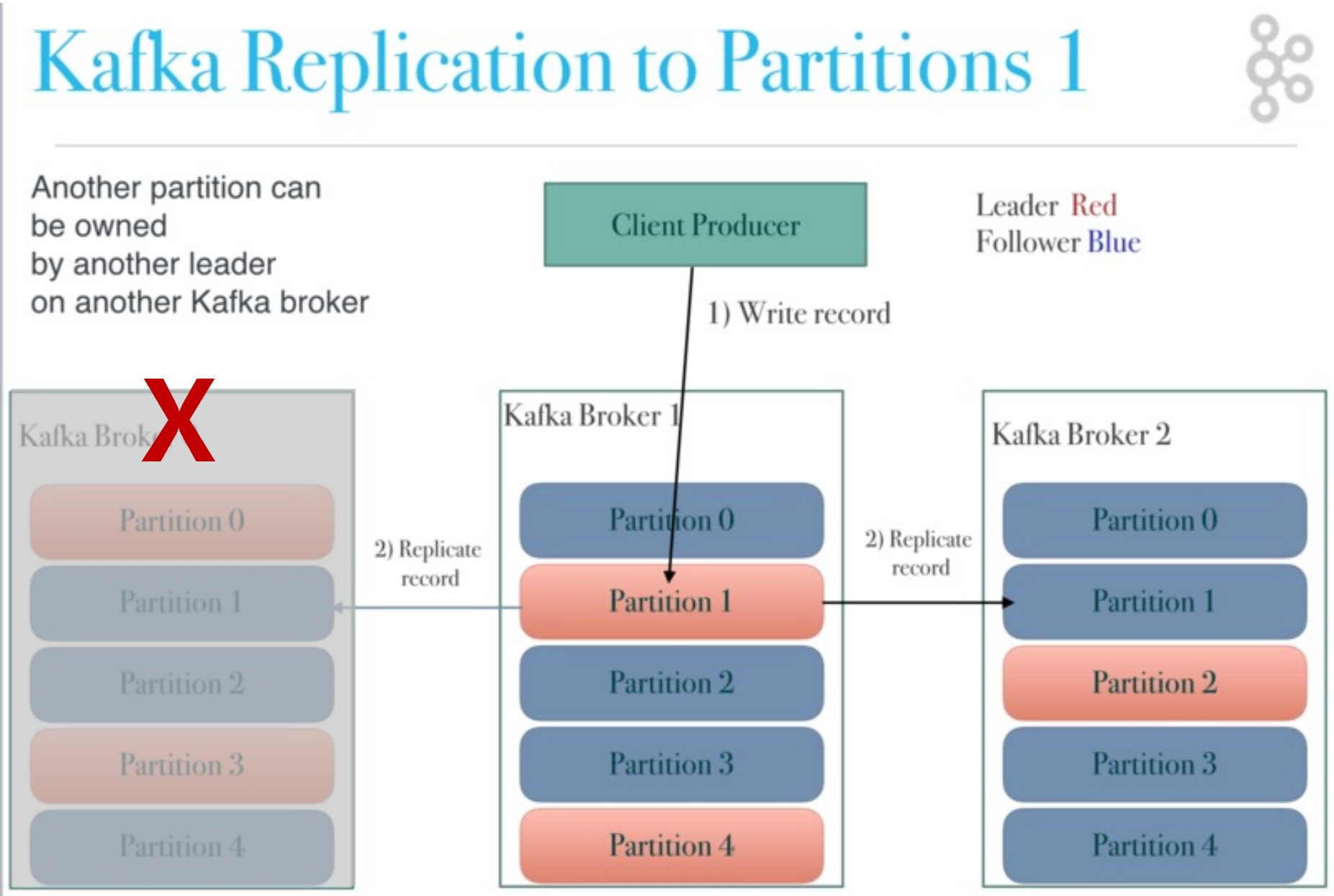
- *Leader: Kafka 클라이언트와 데이터를 주고 받는 역할*
- *Follower: Leader 파티션으로부터 레코드를 지속 복제 (복제하는데 시간 필요). Leader 파티션의 동작이 불가능할 경우 나머지 Follower 중 1개가 Leader로 선출*

Broker Partition Replication



- *ISR(In-Sync Replica): 특정 파티션의 Leader, Follower 가 레코드가 모두 복제되어 Sync가 맞는 상태*

Broker Partition Replication



- *Broker #0 장애 발생 시*
  - *파티션 #0의 Leader가 Broker #1 또는 Broker #2 중에 새로 할당*
  - *Kafka 클라이언트는 새로운 파티션 Leader와 연동*

Ecosystem – Kafka Client

- Kafka와 데이터를 주고받기 위해 사용하는 Java Library
  - https://mvnrepository.com/artifact/org.apache.kafka/kafka-clients
- Producer, Consumer, Admin, Stream 등 Kafka관련 API 제공
- 다양한 3rd party library 존재: C/C++, Node.js, Python, .NET 등
  - https://cwiki.apache.org/confluence/display/KAFKA/Clients
- Kafka Broker 버전과 Client 버전 호환 여부 확인 필요
  - https://blog.voidmainvoid.net/193

Ecosystem – Kafka Streams

- 데이터를 변환(Transformation)하기 위한 목적으로 사용하는 API
- 스트림 프로세싱을 지원하기 위한 다양한 기능을 제공
  - Stateful 또는 Stateless와 같이 상태 기반 스트림 처리 가능
  - Stream api와 DSL(Domain Specific Language)를 동시 지원
  - Exactly-once 처리, 고가용성 특징Kafka security(ACL, SASL 등) 완벽 지원
  - 스트림 처리를 위한 별도 클러스터 (ex, yarn 등) 불필요

Ecosystem – Kafka Connect

- *많은 경우 Kafka Client로 Kafka에 데이터를 저장하지만, Kafka Connect를 통해 Data를 Import/Export 가능*
- *코드 없이 Configuration으로 데이터를 이동시키는 것이 목적*
  - *Standalone mode, Distribution mode 지원*
  - *RESTful API 통해 지원*
  - *Stream 또는 Batch 형태로 데이터 전송 가능*
  - *커스텀 Connector를 통한 다양한 Plugin 제공 (File, S3, Hive, Mysql, etc …)*

Ecosystem – Kafka Mirror maker

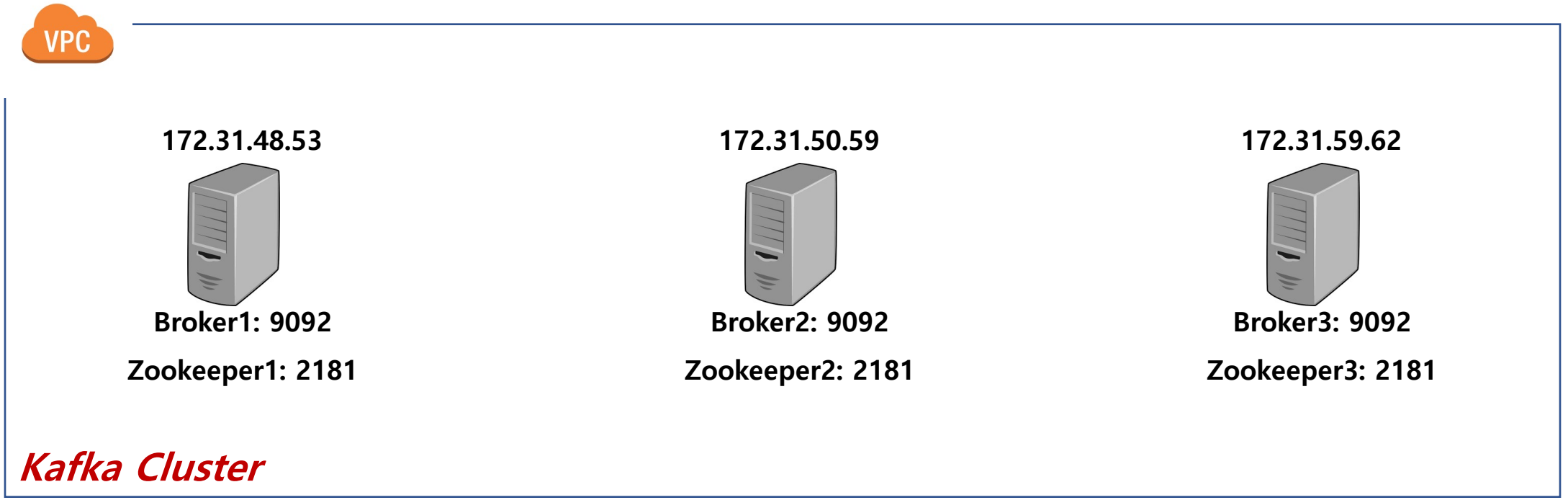
- *특정 Kafka Cluster에서 다른 Kafka Cluster로 Topic 및 Record를 복제하는 Standalone tool*
- *2019년 11월, 기존 MirrorMaker 2.0 release*
- *Cluster간 Topic에 대한 모든 것을 복제하는 것이 목적*
  - *신규 Topic, 파티션 감지 기능 및 Topic 설정 자동 Sync 기능*
  - *양방향 Cluster 토픽 복제*
  - *Mirror Monitoring을 위한 다양한 Metric(Latency, Count 등) 제공*

Ecosystem – Applications

- *confluent/ksqlDB*
  - *SQL 구문을 통한 Stream data processing 지원*
- *confluent/Schema Registry*
  - *AVRO 기반의 스키마 저장소*
- *confluent/REST Proxy*
  - *REST API를 통한 Consumer/Producer*
- *linkedin/Kafka burrow*
  - *Consumer Lag 수집 및 분석*
- *yahoo/CMAK*
  - *Kafka Cluster manager*
- *uber/uReplicator*
  - *Kafka Cluster 간 Topic 복제(전달)*
- *Spark stream*
  - *다양한 소스(Kafka)로부터 실시간 데이터 처리*

Kafka 설치

- AWS EC2에 설치



Kafka 설치

- *AWS EC2에 설치*

Name	인스턴스 ID	인스턴스 상태	인스턴스 ...	상...	경보 상태	가용 영역	퍼블릭 IPv4 DNS	퍼블릭 IPv4 ...
ec2-kafka-1	i-00ab7074...	실행 중	t2.small	–	경보 없음	us-east-1e	ec2-54-210-52-15...	54.210.52.156
ec2-kafka-2	i-05871498...	실행 중	t2.small	–	경보 없음	us-east-1e	ec2-100-25-196-2...	100.25.196.246
ec2-kafka-3	i-010369a0...	실행 중	t2.small	–	경보 없음	us-east-1e	ec2-52-3-245-165...	52.3.245.165

인바운드 규칙

유형	프로토콜	포트 범위	소스
모든 TCP	TCP	0 - 65535	sg-04d89403c737cdd9b (ec2-kafka-sg)
SSH	TCP	22	175.192.16.174/32
모든 ICMP - IPv4	ICMP	전체	sg-04d89403c737cdd9b (ec2-kafka-sg)

```
$ sudo yum install java-1.8.0-openjdk
$ wget http://archive.apache.org/dist/kafka/2.1.1/kafka_2.11-2.1.1.tgz
```

Kafka 설치

- **AWS EC2에 설치**
  - **Kafka 실행 최소 Heap size 설정 제거**
    - *$ export KAFKA_HEAP_OPTS="-Xmx400m -Xms400m"*
  - **Kafka 2.5.0은 1G의 Heap memory가 default**
  - **테스트용 EC2인 t2.micro에 실행하기 위해 Heap size 환경 변수 선언**

```
- Xmx6g –Xms 6g –XX:MetaspaceSize=96m –XX:+UseG2GC
- XX:MaxGCPauseMillis=20 –XX:InitiatingHeapOccupancyPercent=35 –XX:G1HeapRegionSie=16M
- XX:MinMetaspaceFreeRatio=50 –XX:MaxMetaspaceFreeRatio=80
```

  - **Linked-in에서 테스트한 최적의 Java Option 추천값**
  - **60대 브로커, 5만개 파티션, Replication-factor 2로 구성시**
    - *300MB/sec inbound, 1GB/sec outbound 보장*
    - *환산: 1TB/hour inboud, 3TB/hour outbound*

Kafka 설치

- **Zookeeper 설정 (ec2-kafka-1, ec2-kafka-2, ec2-kafka-3)**

$ vi ***[kafka_2.11-2.1.1]/config/zookeeper.properties***

```
# the directory where the snapshot is stored.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# disable the per-ip limit on the number of connection
s since this is a non-production config
maxClientCnxns=0

initLimit=5
syncLimit=2

server.1=172.31.48.53:2888:3888
server.2=172.31.50.59:2888:3888
server.3=172.31.59.62:2888:3888
```

← *팔로워가 리더와 초기에 연결하는 시간에 대한 타임아웃*

← *팔로워가 리더와 동기화 하는 데에 대한 타임아웃*

```
$ mkdir /tmp/zookeeper
$ echo 1 > /tmp/zookeeper/myid
```

- ***(ec2-kafka-1, ec2-kafka-2, ec2-kafka-3)***

Kafka 설치

- ***Kafka 설정 (ec2-kafka-1, ec2-kafka-2, ec2-kafka-3)***

  $ vi ***[kafka_2.11-2.1.1]/config/server.properties***

```
broker.id=1
listeners=PLAINTEXT://:9092
advertised.listeners=PLAINTEXT://172.31.48.53:9092
zookeeper.connect=172.31.48.53:2181,172.31.50.59:2181,172.31.59.62:2181

broker.id=2
listeners=PLAINTEXT://:9092
advertised.listeners=PLAINTEXT://172.31.50.59:9092
zookeeper.connect=172.31.48.53:2181,172.31.50.59:2181,172.31.59.62:2181

broker.id=3
listeners=PLAINTEXT://:9092
advertised.listeners=PLAINTEXT://172.31.59.62:9092
zookeeper.connect=172.31.48.53:2181,172.31.50.59:2181,172.31.59.62:2181
```

Kafka 서버 구동

- Zookeeper 및 Kafka 서버 구동 (ec2-kafka-1, ec2-kafka-2, ec2-kafka-3)

  $ [kafka_2.11-2.1.1]/bin/zookeeper-server-start.sh [kafka_2.11-2.1.1]/bin/zookeeper.properties

  $ [kafka_2.11-2.1.1]/bin/kafka-server-start.sh [kafka_2.11-2.1.1]/bin/server.properties

- Topic 생성

```
$ cd [kafka_2.11-2.1.1]
$ bin/kafka-topics.sh --create --zookeeper 172.31.48.53:2181,172.31.50.59:2181,172.31.59.62:2181 ₩
        --replication-factor 3 --partitions 1 --topic GameLog
```

- Topic 목록 확인

```
$ bin/kafka-topics.sh --list --zookeeper 172.31.48.53:2181,172.31.50.59:2181,172.31.59.62:218
```

Kafka 테스트 1)

- 메시지 생산

```
$ bin/kafka-console-producer.sh --broker-list 172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092 ₩
        --topic GameLog
```

- 메시지 소비

```
(cmd 17) $ bin/kafka-console-consumer.sh --bootstrap-server ₩
        172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092 --topic GameLog --from-beginning
```

```
(cmd 18) $ bin/kafka-console-consumer.sh --bootstrap-server ₩
        172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092 --topic GameLog –group testgroup ₩
        --from-beginning
```

Kafka 테스트 1)

- **Consumer Group 테스트**

(cmd 19) $ bin/kafka-consumer-groups.sh --bootstrap-server ₩

172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092 --list

(cmd 20) $ bin/kafka-consumer-groups.sh --bootstrap-server ₩

172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092 --group testgroup --describe

```
Consumer group 'testgroup' has no active members.

TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID     HOST            CLIENT-ID
GameLog         0          18              18              0               -               -               -
```

파티션 — Consumer Offset — 현재 토픽의 마지막 Offset — Consumer Lag

Kafka 테스트 1)

- **Consumer Group 테스트**

  → Producer 추가 작업 실행

(cmd 20) $ bin/kafka-consumer-groups.sh --bootstrap-server ₩
172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092 --group testgroup --describe

```
Consumer group 'testgroup' has no active members.

TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG
GameLog         0          18              21              3
```

마지막 Offset과 지연 발생
차이가 있음

Kafka 테스트 1)

- **Consumer Group 테스트**

→ Offset 초기화

(cmd 21) $ bin/kafka-consumer-groups.sh --bootstrap-server ₩

172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092 --group testgroup ₩

--topic GameLog --reset-offsets --to-earliest --execute

```
TOPIC                          PARTITION  NEW-OFFSET
GameLog                        0          0
```

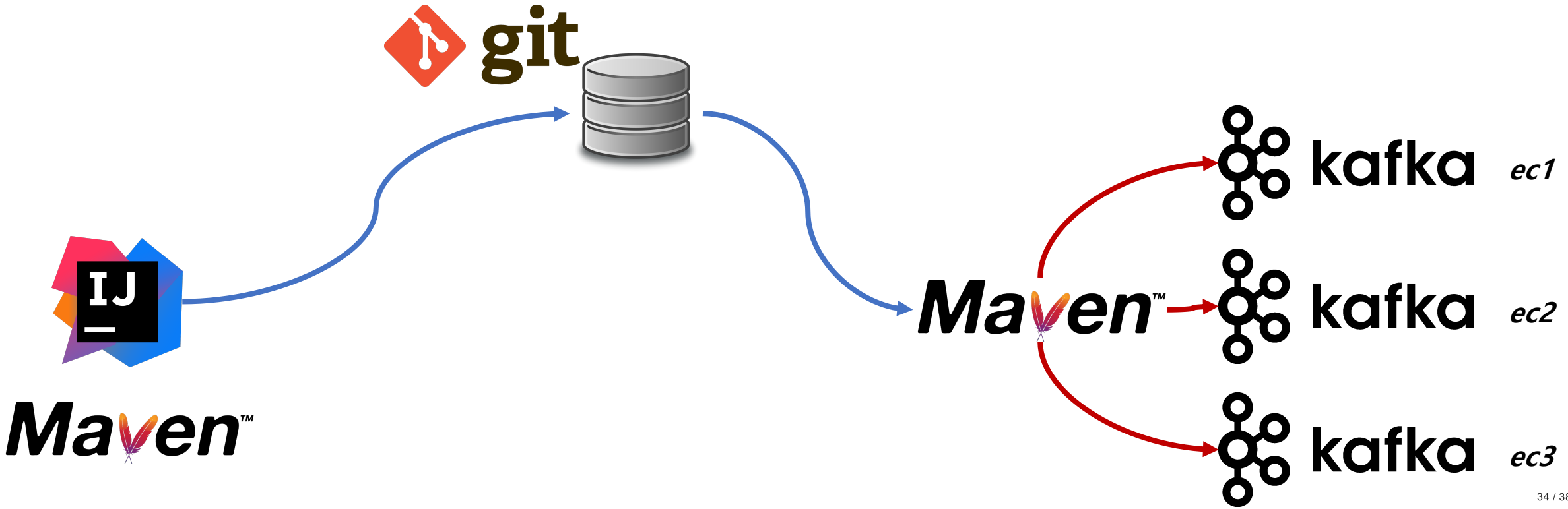
(cmd 20) $ bin/kafka-consumer-groups.sh --bootstrap-server ₩

172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092 --group testgroup --describe

```
Consumer group 'testgroup' has no active members.

TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG
GameLog         0          0               21              21
```

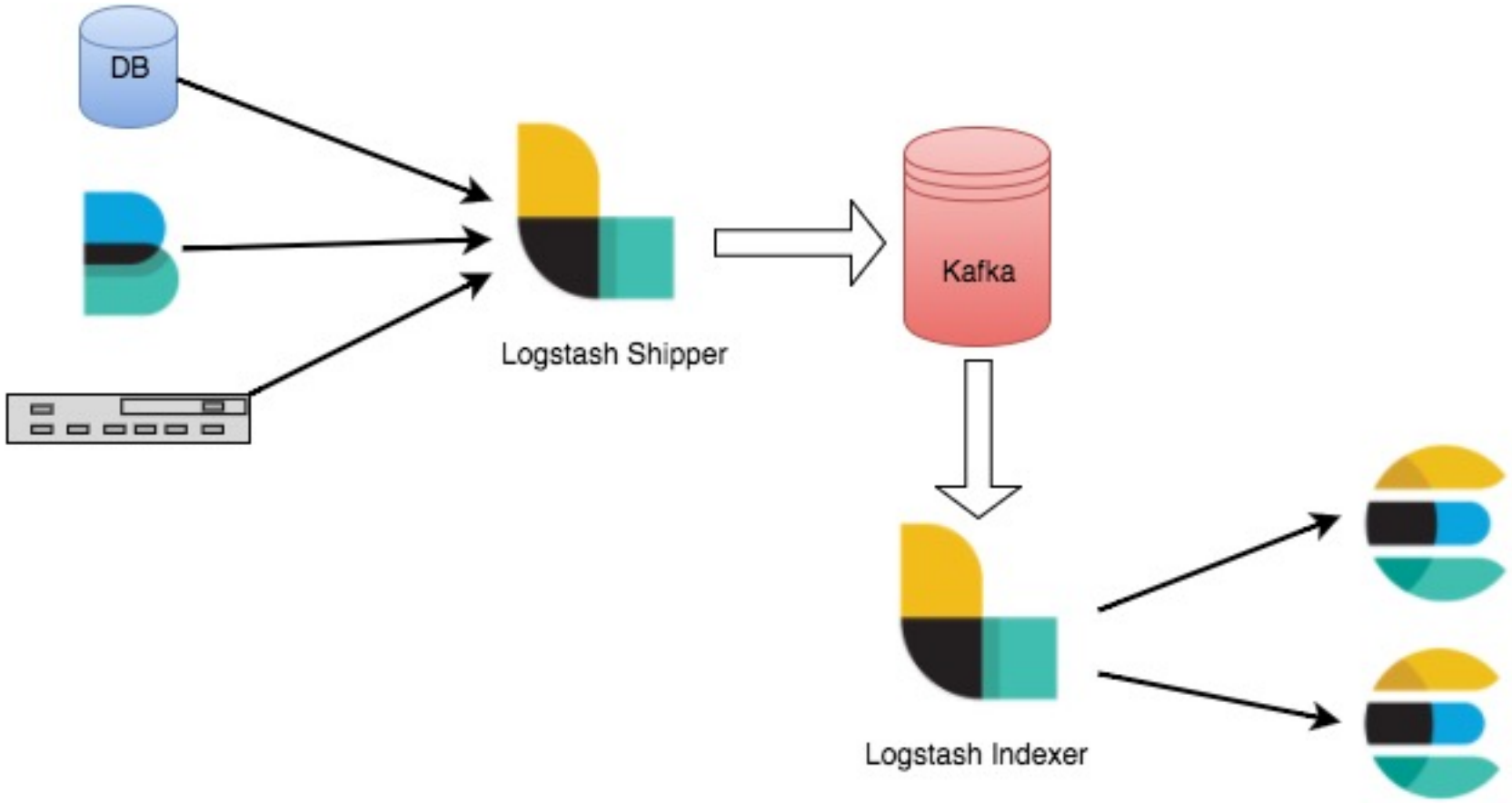
Kafka 테스트 2)

- *Spring Boot 예제*
  - *simple-kafka-producer*
  - *simple-kafka-consumer*



Kafka 테스트 3)

- Kafka로 Log를 보내는 방법
  - logstash (filebeat)
  - logback
  - log4j



Kafka 테스트 3)

- Kafka로 Log를 보내는 방법 - logback
  - my-restful-service

```
<!-- kafkaAppender -->
<appender name="kafkaAppender" class="com.github.danielwegener.logback.kafka.KafkaAp
    <encoder class="com.github.danielwegener.logback.kafka.encoding.LayoutKafkaMessa
        <layout class="ch.qos.logback.classic.PatternLayout">
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</patte
        </layout>
    </encoder>
    <topic>test-logback</topic>
    <producerConfig>bootstrap.servers=172.31.48.53:9092,172.31.50.59:9092,172.31.59.62:9092</producerConfig>
</appender>

<!-- kafkaAppender with Logstash -->
<appender name="logstashKafkaAppender" class="com.github.danielwegener.logback.kafka.KafkaAppender"...>

<!-- logger -->
<logger name="org.apache.kafka" level="ERROR"/>
<logger name="com.example.myrestfulservices" level="DEBUG">
    <appender-ref ref="kafkaAppender" />
    <!--
    <appender-ref ref="logstashKafkaAppender" />
    -->
</logger>
```

logback-spring.xml

```
<dependency>
    <groupId>com.github.danielwegener</groupId>
    <artifactId>logback-kafka-appender</artifactId>
    <version>0.1.0</version>
</dependency>

<dependency>
    <groupId>net.logstash.logback</groupId>
    <artifactId>logstash-logback-encoder</artifactId>
    <version>4.8</version>
</dependency>
```

pom.xml

Kafka 테스트 3)

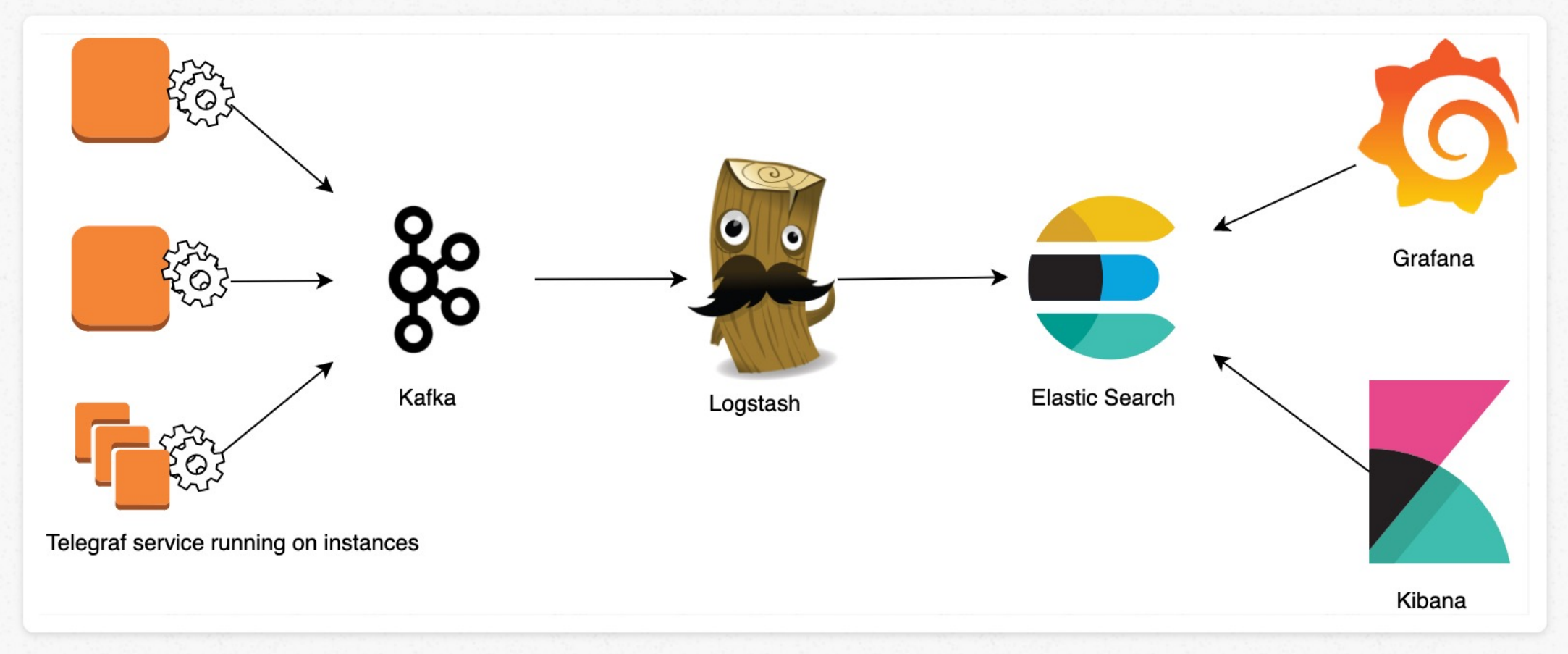
- Kafka로 Log를 보내는 방법 - logback

```java
@RestController
public class UserController {
    private static final Logger log = LoggerFactory.getLogger(UserController.class);

    @Autowired
    private UserDaoService service;

    @GetMapping("/users")
    public List<User> retrieveAllUsers() {
        List<User> users = service.findAll();
        for (User user : users) {
            log.debug(user.toString());
        }
        return users;
    }
```

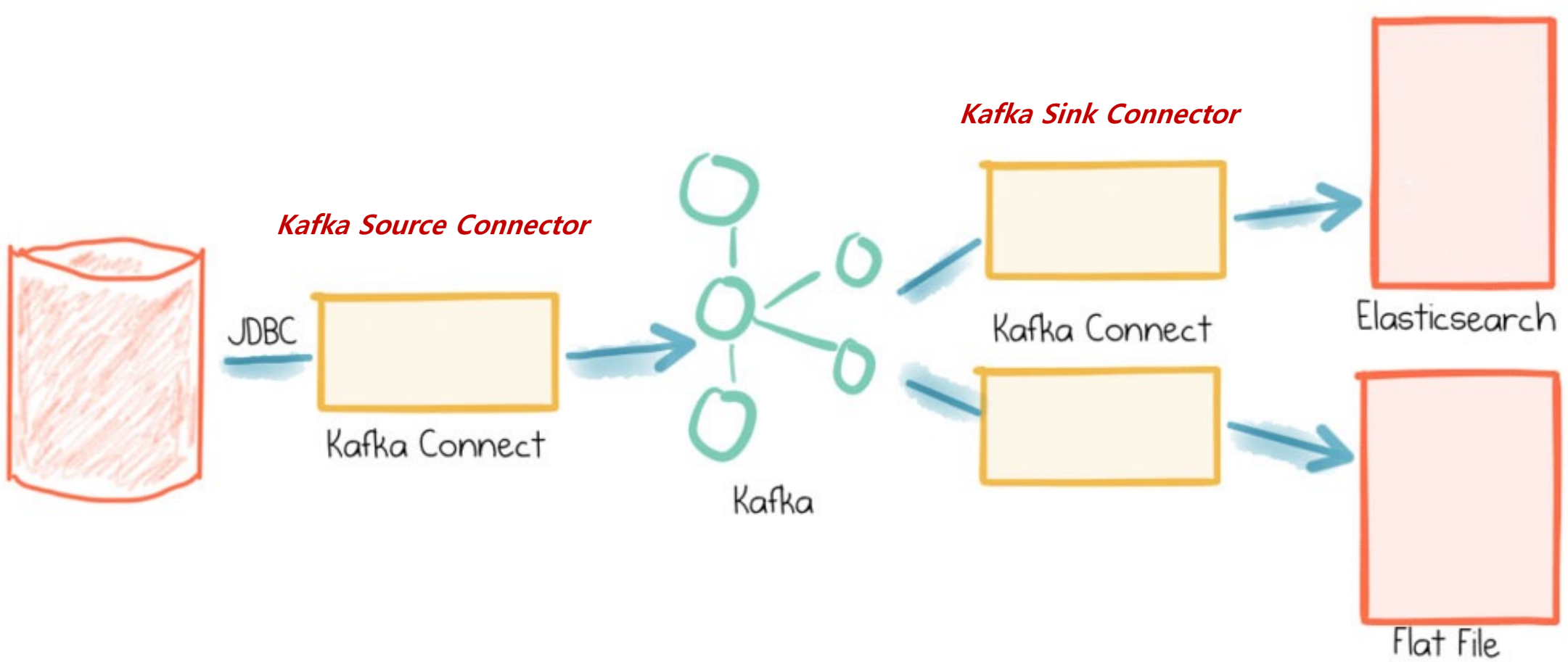
UserController.java

Kafka 테스트 4)

- *telegraf*
  - *kafka-consumer-save-metric*



Kafka Connect



Kafka Connect

- ***Kafka Connect 설치***
  - curl -O http://packages.confluent.io/archive/5.5/confluent-community-5.5.2-2.12.tar.gz
- ***Kafka Connect 설정***
  - $KAFKA_HOME/config/connectd-distributed.properties 설정 (기본으로 사용)
- ***Kafka Connect 실행***
  - $KAFKA_HOME/bin/connect-distributed $KAFKA_HOME/etc/kafka/connect-distributed.properties
- ***Topic 목록 확인***
  - ./kafka_2.12-2.6.0/bin/kafka-topics.sh --bootstrap-server localhost:9092 --list

```
__consumer_offsets
connect-configs
connect-offsets
connect-status
```

- *Kafka Source Connect 추가 (Mysql)*

```
echo '
{
    "name" : "my-source-connect",
    "config" : {
        "connector.class" : "io.confluent.connect.jdbc.JdbcSourceConnector",
        "connection.url":"jdbc:mysql://localhost:3306/my_db",
        "connection.user":"root",
        "connection.password":"test1357",
        "mode": "incrementing",
        "incrementing.column.name" : "id",
        "table.whitelist":"users",
        "topic.prefix" : "my_topic_",
        "tasks.max" : "1"
    }
}
' | curl -X POST -d @- http://localhost:8083/connectors --header "content-Type:application/json"
```

```
./kafka_2.12-2.6.0/bin/kafka-topics.sh  --bootstrap-server localhost:9092 --list
__consumer_offsets
connect-configs
connect-offsets
connect-status
my_topic_users
```

Kafka Connect

- ***Kafka Connect 목록 확인***
  - curl http://localhost:8083/connectors | jq
- ***Kafka Connect 확인***
  - curl http://localhost:8083/connectors/my-source-connect/status | jq


```
{
  "name": "my-source-connect",
  "connector": {
    "state": "RUNNING",
    "worker_id": "218.38.137.27:8083"
  },
  "tasks": [
    {
      "id": 0,
      "state": "RUNNING",
      "worker_id": "218.38.137.27:8083"
    }
  ],
  "type": "source"
}
```

- *Mysql에서 데이터 추가 1)*

```
1 • insert into my_db.users(user_id, name) values('test2', 'TEST ADMIN');
2
3 • SELECT * FROM my_db.users;
```

100% 27:3

Result Grid | Filter Rows: Search | Edit: | Export/Import:

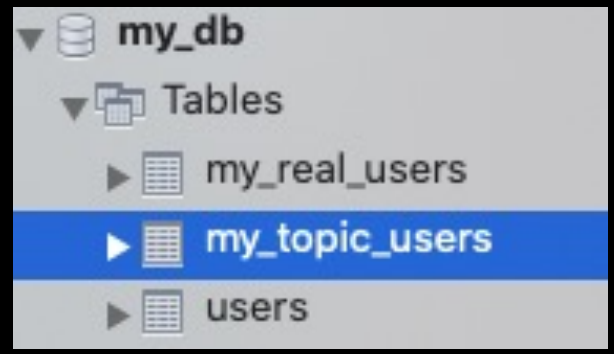
id	user_id	name
1	test1	TEST USER
2	test2	TEST ADMIN
NULL	NULL	NULL

```
bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic my_topic_users --from-beginning
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"id"},{"type":"string","optional":true,"fie
ld":"user_id"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"users"},"payload":{"id":1,"user
_id":"test1","name":"TEST USER"}}
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"id"},{"type":"string","optional":true,"fie
ld":"user_id"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"users"},"payload":{"id":2,"user
_id":"test2","name":"TEST ADMIN"}}
```

- *Kafka Sink Connect 추가 (Mysql)*

```
echo '
{
    "name":"my-sink-connect",
    "config":{
        "connector.class":"io.confluent.connect.jdbc.JdbcSinkConnector",
        "connection.url":"jdbc:mysql://localhost:3306/my_db",
        "connection.user":"root",
        "connection.password":"test1357",
        "auto.create":"true",
        "auto.evolve":"true",
        "delete.enabled":"false",
        "tasks.max":"1",
        "table.whitelist":"my_real_users",
        "topics":"my_topic_users"
    }
}
'| curl -X POST -d @- http://localhost:8083/connectors --header "content-Type:application/json"
```

Topic의 이름과 같은 테이블 생성



- *Mysql에서 데이터 추가 2)*

```
1 • insert into my_db.users(user_id, name) values('test3', 'TEST MANAGER');
2 • SELECT * FROM my_db.users;
```

100% 27:2

Result Grid | Filter Rows: Search | Edit: | Export/Impor

id	user_id	name
1	test1	TEST USER
2	test2	TEST ADMIN
3	test3	TEST MANAGER
NULL	NULL	NULL

```
1 • SELECT * FROM my_db.my_topic_users;
```

100% 36:1

Result Grid | Filter Rows: Search | Exp

id	user_id	name
1	test1	TEST USER
2	test2	TEST ADMIN
3	test3	TEST MANAGER

```
bin/kafka-console-consumer --bootstrap-server localhost:9092 --topic my_
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"fie
e,"field":"user_id"},{"type":"string","optional":true,"field":"name"}],"op
"id":1,"user_id":"test1","name":"TEST USER"}}
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"id"},{"type":"string","optional":tru
e,"field":"user_id"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"users"},"payload":{
"id":2,"user_id":"test2","name":"TEST ADMIN"}}
{"schema":{"type":"struct","fields":[{"type":"int32","optional":false,"field":"id"},{"type":"string","optional":tru
e,"field":"user_id"},{"type":"string","optional":true,"field":"name"}],"optional":false,"name":"users"},"payload":{
"id":3,"user_id":"test3","name":"TEST MANAGER"}}
```

Kafka Connect

- *Resource 준비*
  - *mkdir ~/connect-quickstart*
  - *seq 1000 > ~/connect-quickstart/input.txt*
  - *touch ~/connect-quickstart/output.txt*

- *Kafka Source Connect 추가 (File)*

```
echo '
{
    "name": "file-source",
    "config": {
        "connector.class":"org.apache.kafka.connect.file.FileStreamSourceConnector",
        "tasks.max":"1",
        "topic":"connect-quickstart",
        "file": "/home/vagrant/connect-quickstart/input.txt"
    }
}
' | curl -s -X POST -d @- http://localhost:8083/connectors -H "Content-Type: application/json"
```

- *Kafka Sink Connect 추가 (File)*

```
echo '
{
    "name": "file-sink",
    "config": {
        "connector.class":"org.apache.kafka.connect.file.FileStreamSinkConnector",
        "tasks.max":"1",
        "topics":"connect-quickstart",
        "file": "/home/vagrant/connect-quickstart/output.txt"
    }
}
' | curl -X POST -d @- http://localhost:8083/connectors -H "Content-Type: application/json"
```