

14장. Service 간의 비동기 통신 (Kafka)



Apache Kafka 개요 및 아키텍처
Apache Kafka 구성요소 및 동작 방식

5. Apache Kafka 개요

1. 개요

Apache 소프트웨어 재단이 스칼라(scala)로 개발한 오픈 소스 메시지 브로커 프로젝트로 **Pub/Sub 모델의 메시지 큐(Queue)를 지원**한다.

마이크로서비스 아키텍처에서 메시지 브로커는 Message Backing Service로써 동작하며, 메시지의 처리를 통해 비동기 어플리케이션, DB동기화,보상 트랜잭션 구현, PUB/SUB 구현 등 다양한 형태의 어플리케이션으로 응용될 수 있다.

Kafka는 대용량 실시간 처리에 특화되어 있으며, 특히 대량의 Batch 작업을 일괄처리 하는데 적합하다.

2. Queue & Pub/Sub

가) Queue

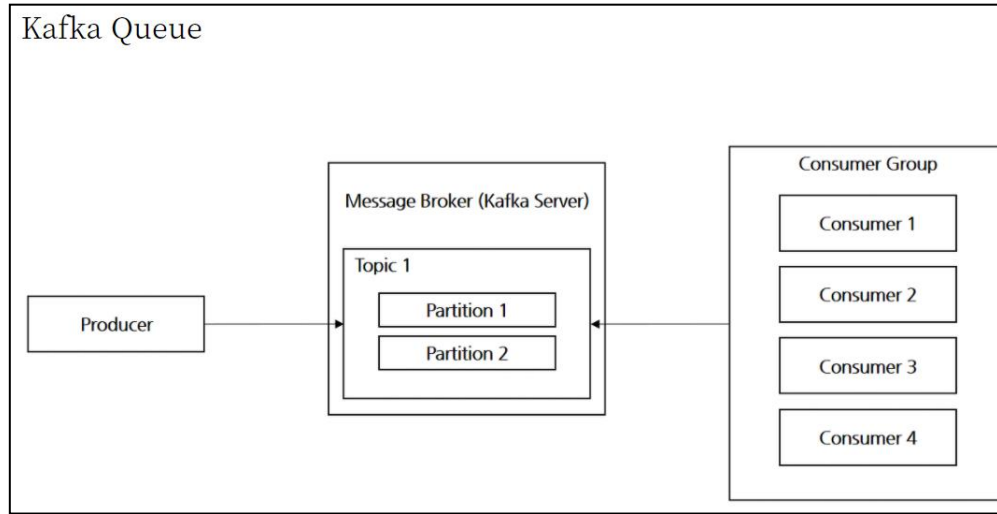
단일 송신자가 단일 수신자에게 데이터를 전송하는 방식으로, 큐에 저장된 메시지는 한 명의 수신자에 의해 한번만 읽을 수 있으며, 읽혀진 메시지는 큐에서 제거한다. Queue방식은 한 번 읽고 지워지는 Event-Driven 방식에 적합하다.

나) Pub/Sub

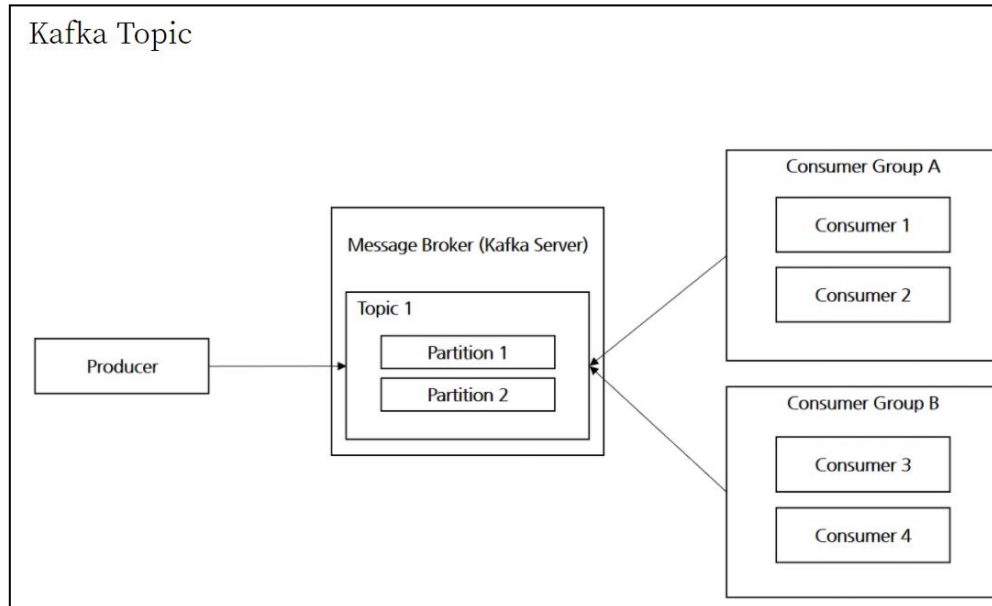
여러 송신자가 메시지를 발행하고 여러 수신자가 구독하는 방식이다. 모든 메시지는 Topic을 구독하는 모든 수신자들에게 전송이 가능하도록 되어있다.

Kafka는 Queue와 Pub/Sub의 장점만을 가지고 만들어졌다.

5. Apache Kafka 개요



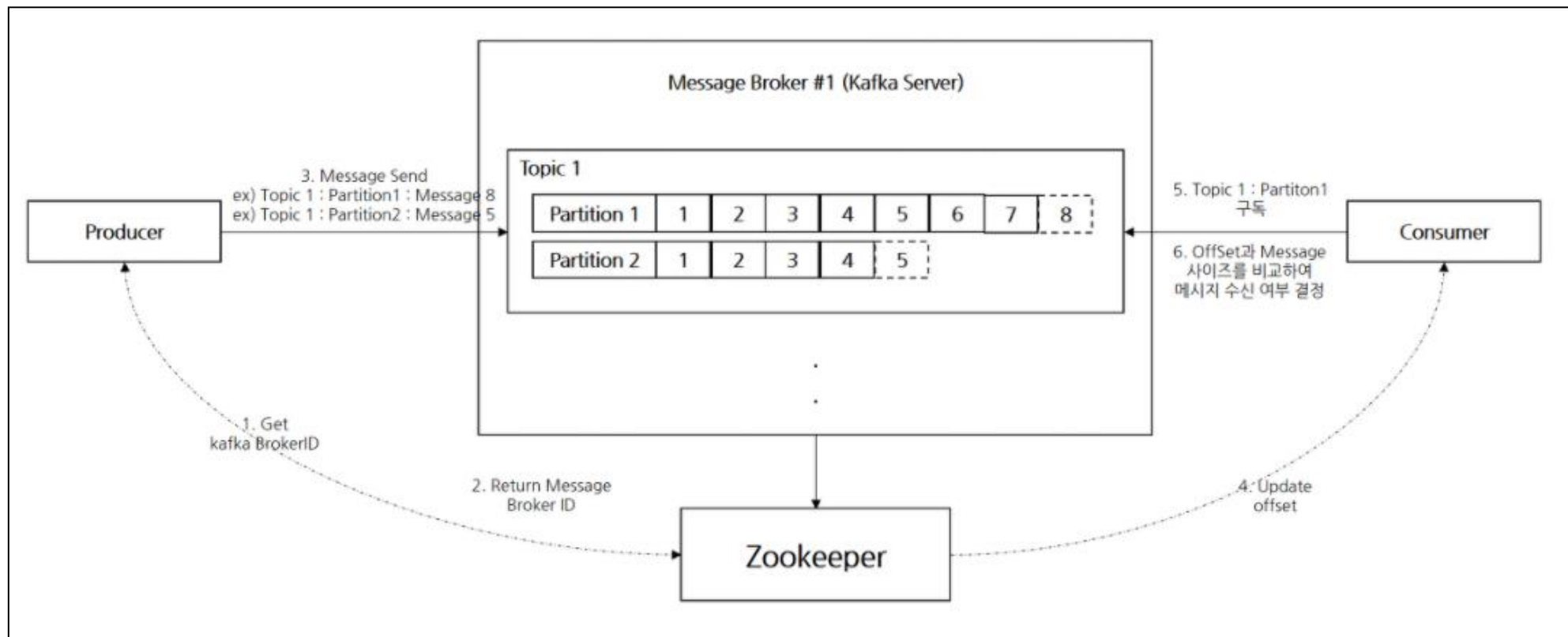
Queue를 설계한 방식으로서, 하나의 Producer와 하나의 Consumer Group로 구성.



Pub/Sub를 설계한 방식으로서, 여러 개의 Consumer Group로 구성.

6. Apache Kafka 아키텍처

3. Kafka 아키텍처



7. Apache Kafka 구성요소

4. 구성요소 개요

가. Zookeeper (Apache Zookeeper)

클러스터 최신 설정정보 관리, 동기화, 리더(leader) 채택 등 클러스터의 서버들이 공유하는 데이터를 관리하기 위한 용도로서 Zookeeper 없이 kafka 구동은 불가능하다.

나. Broker

Kafka Server 를 의미하며, 하나의 클러스터내에서 여러 대를 구축 가능하다. 기본적으로 3대 권장

다. Topic

메시지가 생산되고 소비되는 주제이다.
DB의 테이블 및 파일시스템의 폴더와 매우 유사하다.
유지보수 및 관리목적으로 서로 다른 이름으로 지정한다.

7. Apache Kafka 구성요소

라. Partition

Topic 내에서 메시지가 분산되어 저장되는 Queue 이다. 만약 하나의 Topic에 Partition이 3개 있다면, 3 개의 Partition에 대해서 메시지가 분산되어 저장된다.

Partition내에서는 순서가 보장되지만, Partition 끼리는 메시지 순서를 보장하지 않는다.

마. Log

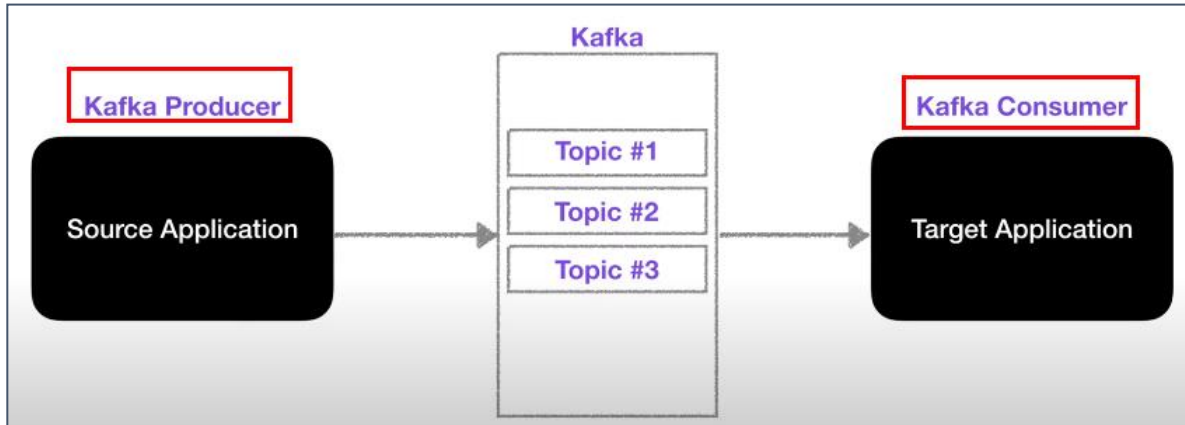
Partition의 하나의 칸을 의미하며 key,value, timestamp로 구성된다.

바. Offset

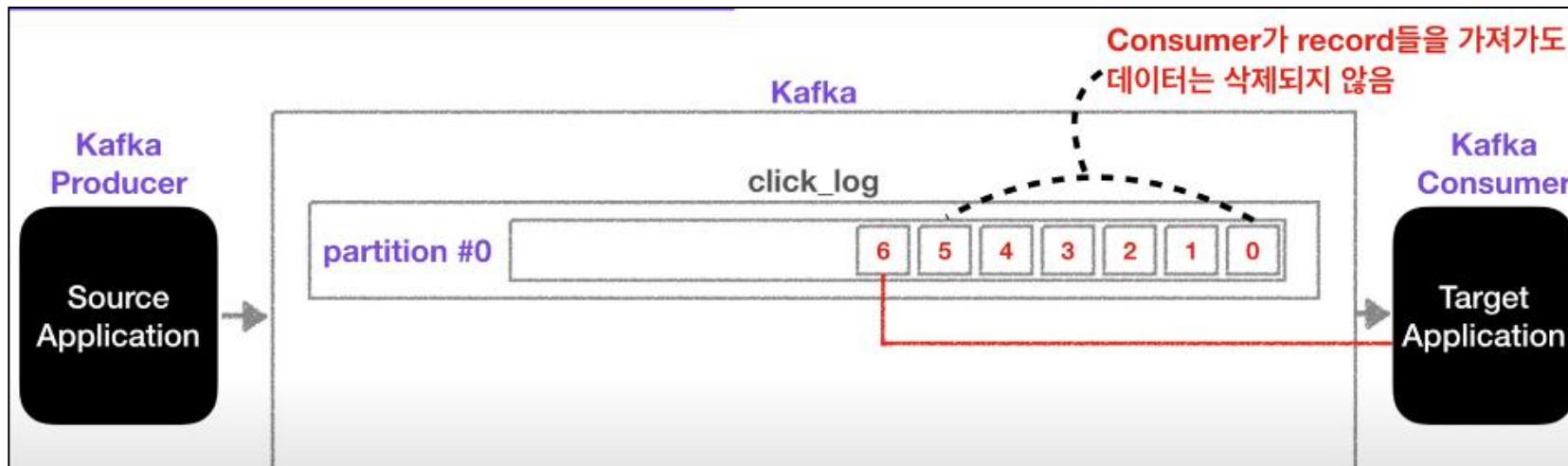
Partition의 각 메시지를 식별할 수 있는 유일한 값으로서 배열의 인덱스 역할이다. 0부터 시작.

8. Apache Kafka 동작 방식

가. 기본적으로 Producer가 메시지를 생산하고 Consumer가 메시지를 소비한다.



나. Topic(click_log명)은 Partition 으로 구성되어 있다.



10. Apache Kafka 간단 실습 (윈도우 환경)

1. Kafka 다운로드

<http://kafka.apache.org/downloads>

DOWNLOAD

2.7.0 is the latest release. The current stable version is 2.7.0.

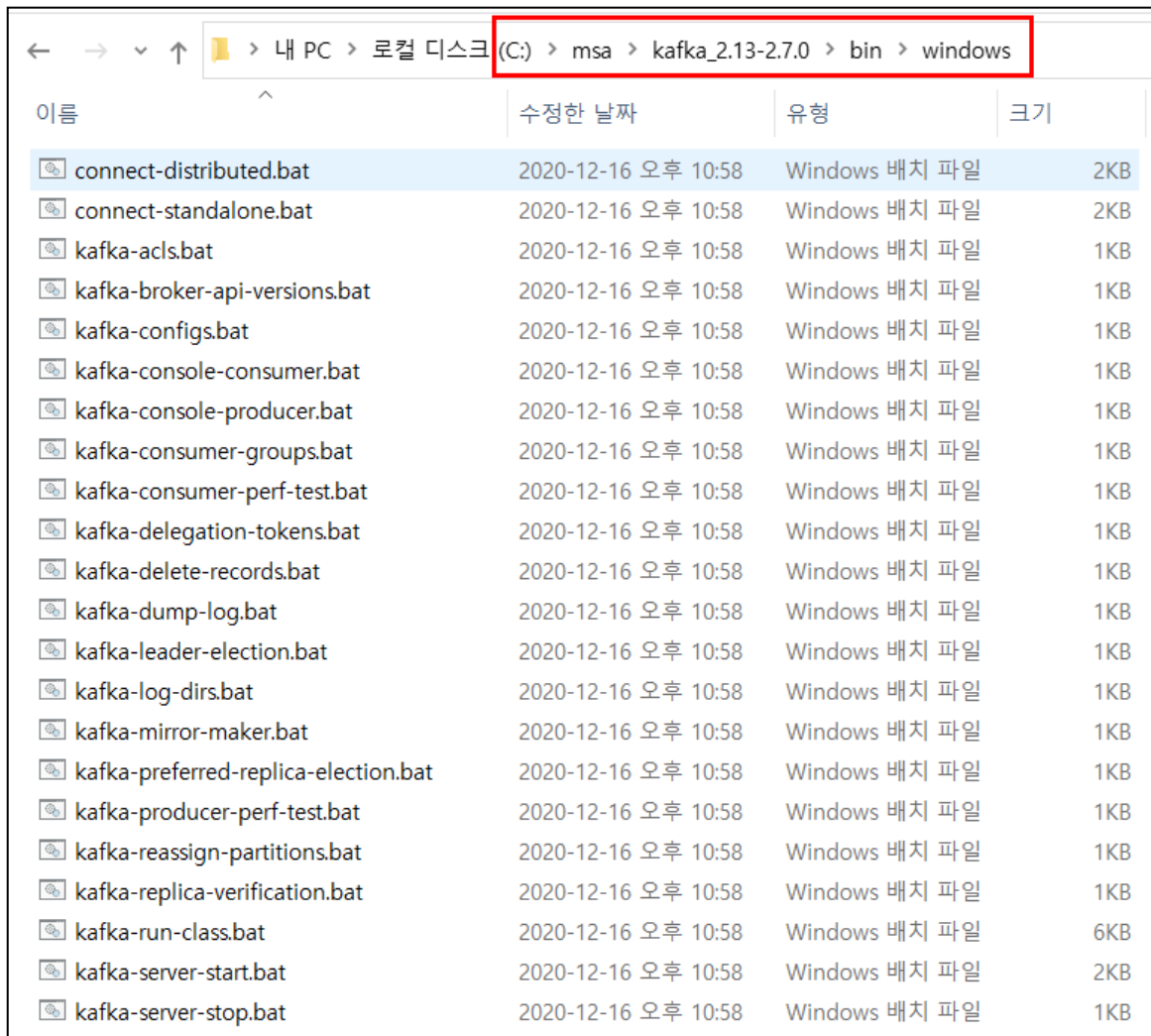
You can verify your download by following these [procedures](#) and using these [KEYS](#).

2.7.0

- Released Dec 21, 2020
- [Release Notes](#)
- Source download: [kafka-2.7.0-src.tgz](#) (asc, sha512)
- Binary downloads:
 - Scala 2.12 - [kafka_2.12-2.7.0.tgz](#) (asc, sha512)
 - Scala 2.13 - [kafka_2.13-2.7.0.tgz](#) (asc, sha512)

10. Apache Kafka 간단 실습 (윈도우 환경)

2. 압축 해제 및 bin/windows 폴더

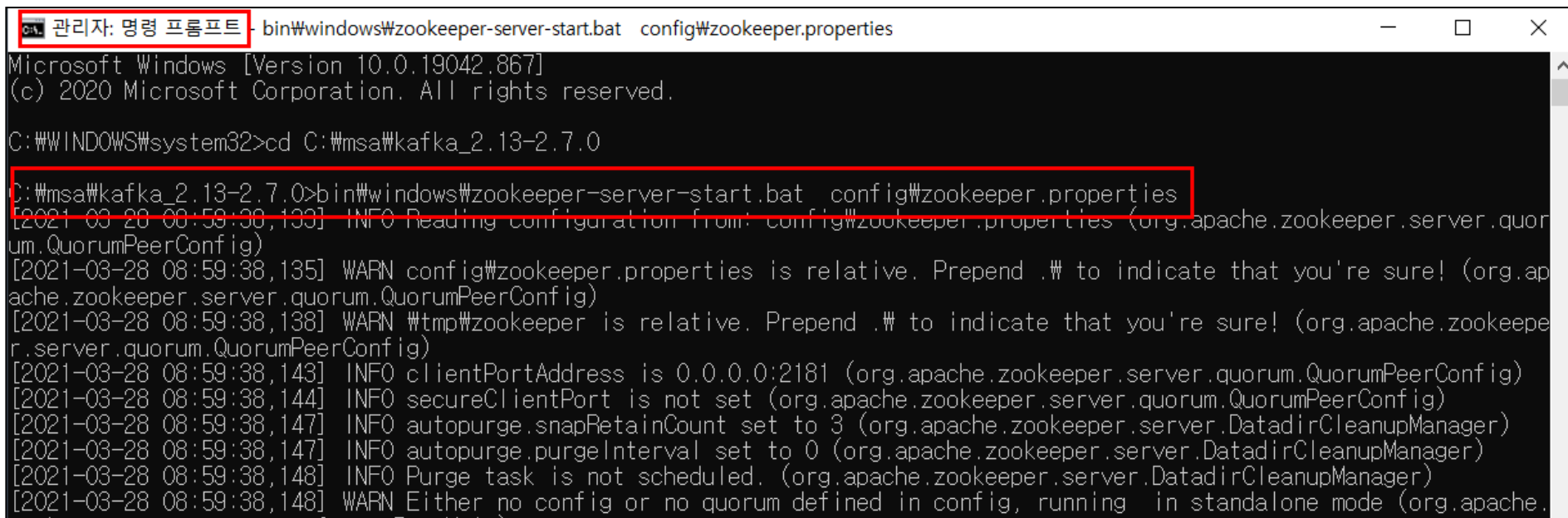


이름	수정된 날짜	유형	크기
connect-distributed.bat	2020-12-16 오후 10:58	Windows 배치 파일	2KB
connect-standalone.bat	2020-12-16 오후 10:58	Windows 배치 파일	2KB
kafka-acls.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-broker-api-versions.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-configs.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-console-consumer.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-console-producer.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-consumer-groups.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-consumer-perf-test.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-delegation-tokens.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-delete-records.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-dump-log.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-leader-election.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-log-dirs.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-mirror-maker.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-preferred-replica-election.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-producer-perf-test.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-reassign-partitions.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-replica-verification.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB
kafka-run-class.bat	2020-12-16 오후 10:58	Windows 배치 파일	6KB
kafka-server-start.bat	2020-12-16 오후 10:58	Windows 배치 파일	2KB
kafka-server-stop.bat	2020-12-16 오후 10:58	Windows 배치 파일	1KB

10. Apache Kafka 간단 실습 (윈도우 환경)

3. Zookeeper 실행 (기본port: 2181)

bin\windows\zookeeper-server-start.bat config\zookeeper.properties



```
관리자: 명령 프롬프트 - bin\windows\zookeeper-server-start.bat config\zookeeper.properties
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\msa\kafka_2.13-2.7.0

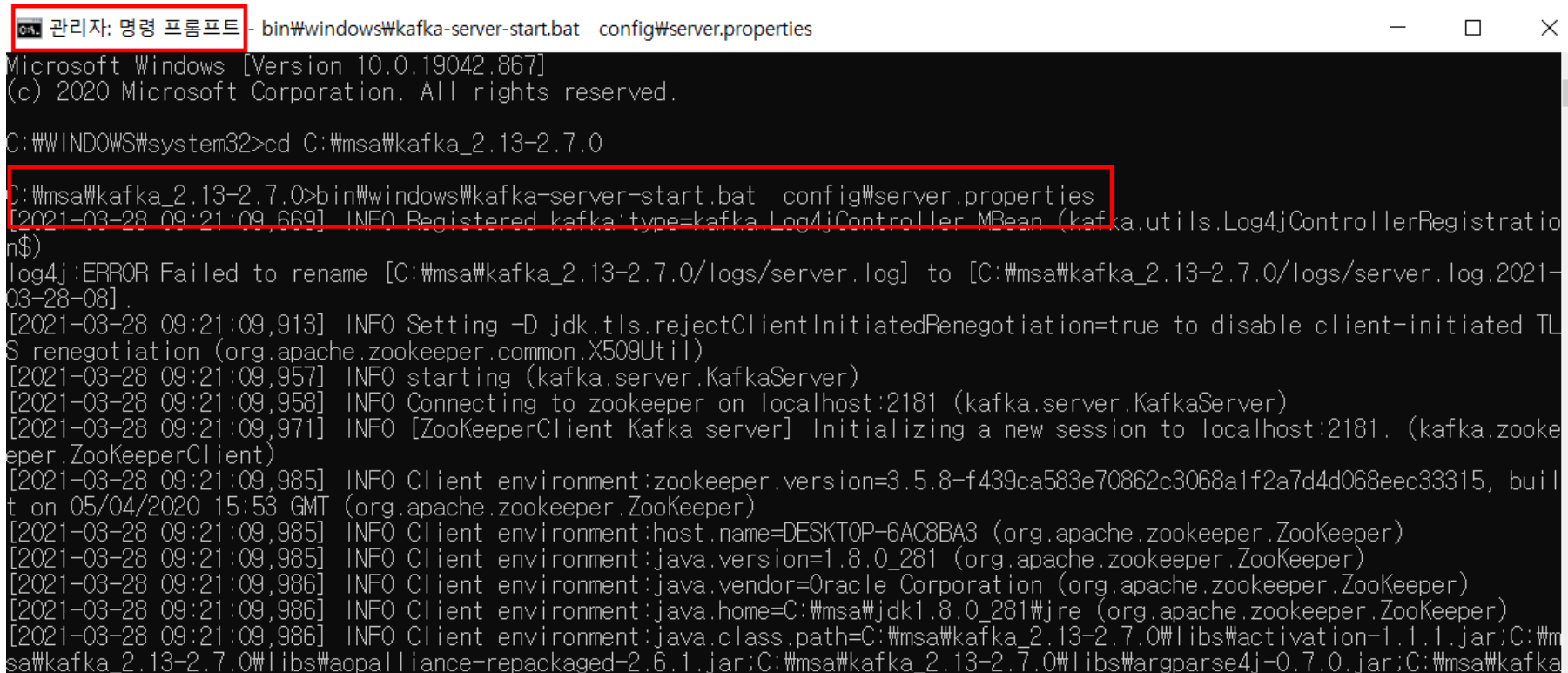
C:\msa\kafka_2.13-2.7.0>bin\windows\zookeeper-server-start.bat config\zookeeper.properties
[2021-03-28 08:59:38,133] INFO Reading configuration from: config\zookeeper.properties (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-03-28 08:59:38,135] WARN config\zookeeper.properties is relative. Prepend .# to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-03-28 08:59:38,138] WARN #tmp#zookeeper is relative. Prepend .# to indicate that you're sure! (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-03-28 08:59:38,143] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-03-28 08:59:38,144] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-03-28 08:59:38,147] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DataDirCleanupManager)
[2021-03-28 08:59:38,147] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanupManager)
[2021-03-28 08:59:38,148] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanupManager)
[2021-03-28 08:59:38,148] WARN Either no config or no quorum defined in config, running in standalone mode (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
```

```
[2021-03-28 08:59:38,143] INFO clientPortAddress is 0.0.0.0:2181 (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-03-28 08:59:38,144] INFO secureClientPort is not set (org.apache.zookeeper.server.quorum.QuorumPeerConfig)
[2021-03-28 08:59:38,147] INFO autopurge.snapRetainCount set to 3 (org.apache.zookeeper.server.DataDirCleanupManager)
[2021-03-28 08:59:38,147] INFO autopurge.purgeInterval set to 0 (org.apache.zookeeper.server.DataDirCleanupManager)
[2021-03-28 08:59:38,148] INFO Purge task is not scheduled. (org.apache.zookeeper.server.DataDirCleanupManager)
```

10. Apache Kafka 간단 실습 (윈도우 환경)

4. Kafka 서버 실행 (기본port: 9092)

```
bin\windows\kafka-server-start.bat config\server.properties
```



```
관리자: 명령 프롬프트 - bin\windows\kafka-server-start.bat config\server.properties
Microsoft Windows [Version 10.0.19042.867]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\msa\kafka_2.13-2.7.0

C:\msa\kafka_2.13-2.7.0>bin\windows\kafka-server-start.bat config\server.properties
[2021-03-28 09:21:09,669] INFO Registered kafka-type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
log4j:ERROR Failed to rename [C:\msa\kafka_2.13-2.7.0\logs\server.log] to [C:\msa\kafka_2.13-2.7.0\logs\server.log.2021-03-28-08]
[2021-03-28 09:21:09,913] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2021-03-28 09:21:09,957] INFO starting (kafka.server.KafkaServer)
[2021-03-28 09:21:09,958] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2021-03-28 09:21:09,971] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2021-03-28 09:21:09,985] INFO Client environment:zookeeper.version=3.5.8-f439ca583e70862c3068a1f2a7d4d068eec33315, built on 05/04/2020 15:53 GMT (org.apache.zookeeper.ZooKeeper)
[2021-03-28 09:21:09,985] INFO Client environment:host.name=DESKTOP-6AC8BA3 (org.apache.zookeeper.ZooKeeper)
[2021-03-28 09:21:09,985] INFO Client environment:java.version=1.8.0_281 (org.apache.zookeeper.ZooKeeper)
[2021-03-28 09:21:09,986] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2021-03-28 09:21:09,986] INFO Client environment:java.home=C:\msa\jdk1.8.0_281\jre (org.apache.zookeeper.ZooKeeper)
[2021-03-28 09:21:09,986] INFO Client environment:java.class.path=C:\msa\kafka_2.13-2.7.0\libs\activation-1.1.1.jar;C:\msa\kafka_2.13-2.7.0\libs\apollon-repackaged-2.6.1.jar;C:\msa\kafka_2.13-2.7.0\libs\argparse4j-0.7.0.jar;C:\msa\kafka_2.13-2.7.0\libs\...
```

10. Apache Kafka 간단 실습 (윈도우 환경)

5. Topic 생성

bin\windows\kafka-topics.bat 로 명령어 확인

```
C:\WINDOWS\system32>cd C:\msa\kafka_2.13-2.7.0
```

```
C:\msa\kafka_2.13-2.7.0>bin\windows\kafka-topics.bat
```

Create, delete, describe, or change a topic.

Option

Description

```
--alter
```

Alter the number of partitions, replica assignment, and/or configuration for the topic.

```
--at-min-isr-partitions
```

if set when describing topics, only show partitions whose isr count is equal to the configured minimum. Not supported with the `--zookeeper` option.

```
--bootstrap-server <String: server to connect to>
```

REQUIRED: The Kafka server to connect to. In case of providing this, a direct Zookeeper connection won't be required.

```
--command-config <String: command
config property file>
```

Property file containing configs to be passed to Admin Client. This is used only with `--bootstrap-server` option for describing and altering broker

10. Apache Kafka 간단 실습 (윈도우 환경)

```
bin\windows\kafka-topics.bat --create --bootstrap-server  
localhost:9092 --topic msa
```

```
C:\msa\kafka_2.13-2.7.0>bin\windows\kafka-topics.bat --create --bootstrap-server localhost:9092 --topic msa  
Created topic msa.  
C:\msa\kafka_2.13-2.7.0>
```

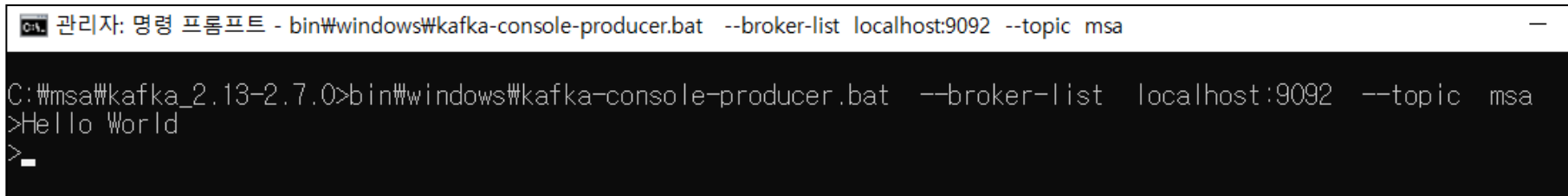
```
bin\windows\kafka-topics.bat --list --bootstrap-server localhost:9092
```

```
C:\msa\kafka_2.13-2.7.0>bin\windows\kafka-topics.bat --list --bootstrap-server localhost:9092  
msa
```

10. Apache Kafka 간단 실습 (윈도우 환경)

6. Producer 이용한 메시지 송신

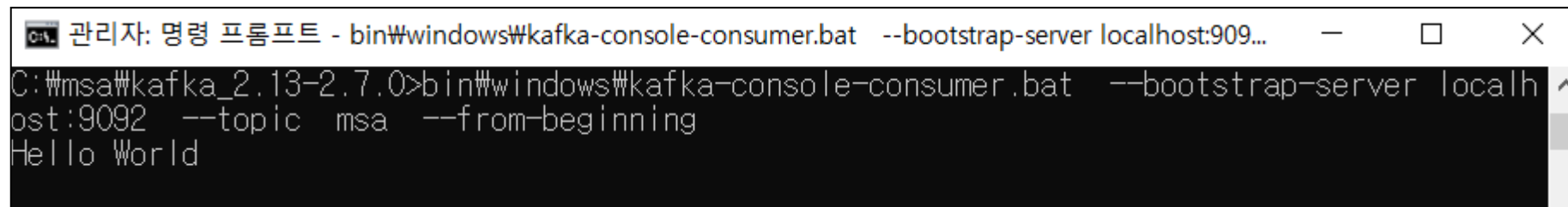
```
bin\windows\kafka-console-producer.bat --broker-list localhost:9092  
--topic msa
```



```
C:\msa\kafka_2.13-2.7.0>bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic msa  
>Hello World  
>
```

7. Consumer 이용한 메시지 수신

```
bin\windows\kafka-console-consumer.bat --bootstrap-server  
localhost:9092 --topic msa --from-beginning
```



```
C:\msa\kafka_2.13-2.7.0>bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic msa --from-beginning  
Hello World
```

15장. SpringBoot + Kafka 실습



Publisher 프로젝트 구축

Subscriber 프로젝트 구축

1. Publisher 프로젝트

1. kafka_msa_publisher 프로젝트 생성

2. dependency 설정

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```


1. Publisher 프로젝트

3. application.properties 속성 설정

```
application.properties
1
2 server.port=8001
3 kafka.bootstrap=localhost:9092
4 ping.topic.name=msa
```

4. Ping/Pong 엔티티 작성

```
Ping.java
1 package com.kafka.domain.entity;
2
3 import java.io.Serializable;
4
5 public class Ping implements Serializable {
6
7     private String msg;
8     private String name;
9
10    public Ping() {
11    }
12
13    public Ping(String msg, String name) {
14        this.msg = msg;
15        this.name = name;
16    }
17 }
```

```
Pong.java
1 package com.kafka.domain.entity;
2
3 public class Pong {
4
5     private String name;
6     private String message;
7
8     public Pong(String name, String message) {
9         this.name = name;
10        this.message = message;
11    }
12 }
```

1. Publisher 프로젝트

5. KafkaPublisherConfig 정의

Producer가 Message를 Publishing하기 위한 ProducerConfig를 구성한다.

```
KafkaPublisherConfig.java
1 package com.kafka.publisher.config;
2
3 import java.util.HashMap;
4
5
6
7
8 @Configuration
9 public class KafkaPublisherConfig {
10
11     @Value(value = "${kafka.bootstrap}")
12     private String bootstrap;
13
14
15     @Bean
16     public ProducerFactory<String, Ping> pingProducerFactory() {
17         Map<String, Object> configProps = new HashMap<>();
18         configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrap);
19         configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
20         configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
21         return new DefaultKafkaProducerFactory<>(configProps);
22     }
23
24
25     @Bean
26     public KafkaTemplate<String, Ping> pingKafkaTemplate() {
27         return new KafkaTemplate<>(pingProducerFactory());
28     }
29 }
```

1. Publisher 프로젝트

6. Publisher 구성

```
@Component
public class PingPublisher {

    @Autowired
    private KafkaTemplate<String, Ping> pingKafkaTemplate;

    @Value(value = "${ping.topic.name}")
    private String pingTopicName;

    public Pong pingAndPong(Ping ping) throws Exception {
        ListenableFuture<SendResult<String, Ping>> future = pingKafkaTemplate.send(pingTopicName, ping);

        future.addCallback(new ListenableFutureCallback<SendResult<String, Ping>>() {
            @Override
            public void onSuccess(SendResult<String, Ping> result) {
                Ping g = result.getProducerRecord().value();
                System.out.println("Sent message=[" + g.toString() + "] with offset=[" + result.getRecordMetadata().offset() + "]);
            }

            @Override
            public void onFailure(Throwable ex) {
                System.out.println("Unable to send message=[" + ping.toString() + "] due to : " + ex.getMessage());
            }
        });
        return new Pong("LG-CNS", "Hello~!");
    }
}
```

1. Publisher 프로젝트

6. Service 구성

```
@Service
public class PingService{

    @Autowired
    PingPublisher pingPublisher;

    public Pong pingAndPong(Ping ping) throws Exception {
        return pingPublisher.pingAndPong(ping);
    }
}
```

7. Controller 구성

```
@RestController
@RequestMapping(value = "/kafka")
public class KafkaPublisherController {

    @Autowired
    PingService pingService;

    @RequestMapping(value = "/publish", method=RequestMethod.POST)
    public Pong pingAndPong(@RequestBody final Ping ping) throws Exception {
        return pingService.pingAndPong(ping);
    }
}
```

2. Subscriber 프로젝트

1. kafka_msa_subscriber 프로젝트 생성

2. dependency 설정

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.9.2</version>
</dependency>
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger-ui</artifactId>
  <version>2.9.2</version>
</dependency>
```

2. Subscriber 프로젝트

3. application.properties 속성 설정

```
application.properties
1
2 server.port=8002
3 kafka.bootstrap=localhost:9092
4 ping.topic.name=msa
```

4. Ping/Pong 엔티티 작성

```
Ping.java
1 package com.kafka.domain.entity;
2
3 import java.io.Serializable;
4
5 public class Ping implements Serializable {
6
7     private String msg;
8     private String name;
9
10    public Ping() {
11    }
12
13    public Ping(String msg, String name) {
14        this.msg = msg;
15        this.name = name;
16    }
17 }
```

```
Pong.java
1 package com.kafka.domain.entity;
2
3 public class Pong {
4
5     private String name;
6     private String message;
7
8     public Pong(String name, String message) {
9         this.name = name;
10        this.message = message;
11    }
12 }
```

2. Subscriber 프로젝트

5. KafkaSubscriberConfig 정의

Consumer가 Message를 Receive하기 위한 ConsumerConfig를 구성한다.

```
@Configuration
public class KafkaSubscriberConfig {

    @Value(value = "${kafka.bootstrap}")
    private String bootstrap;

    public ConsumerFactory<String, Ping> pingConsumerFactory() {
        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrap);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "pong"); // Consumer를 식별하는 고유 아이디.
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false"); // offset을 주기적으로 commit 할지 여부
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest"); // earliest: 맨 처음부터 다시, latest: 이전꺼 무시 새로입력 데이터부터 읽기
        return new DefaultKafkaConsumerFactory<>(props, new StringDeserializer(),
            new JsonDeserializer<>(Ping.class, false));
    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory<String, Ping> pingKafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, Ping> factory = new ConcurrentKafkaListenerContainerFactory<>();
        //ENABLE_AUTO_COMMIT_CONFIG= false로 지정했을때, 어떻게 commit할지를 지정한다.
        factory.getContainerProperties().setAckMode(ContainerProperties.AckMode.MANUAL_IMMEDIATE); // 즉각적으로 ack 요청한다.
        factory.setConsumerFactory(pingConsumerFactory());
        return factory;
    }
}
```

2. Subscriber 프로젝트

6. Subscriber 구성

```
@Component
public class PingSubscriber {

    @Autowired
    KafkaSubscriberService service;

    @KafkaListener(topics = "${ping.topic.name}", containerFactory = "pingKafkaListenerContainerFactory")
    public void pingListener(Ping ping, Acknowledgment ack) {
        try {
            System.out.println("Received ping message: " + ping);
            System.out.println("필요시 서비스 호출: " + service.working());

            ack.acknowledge();
        } catch (Exception e) {
            String msg = "시스템에 예상치 못한 문제가 발생했습니다";
            System.out.println("Recieved ping message: " + msg + e);
        }
    }
}
```

7. Service 구성

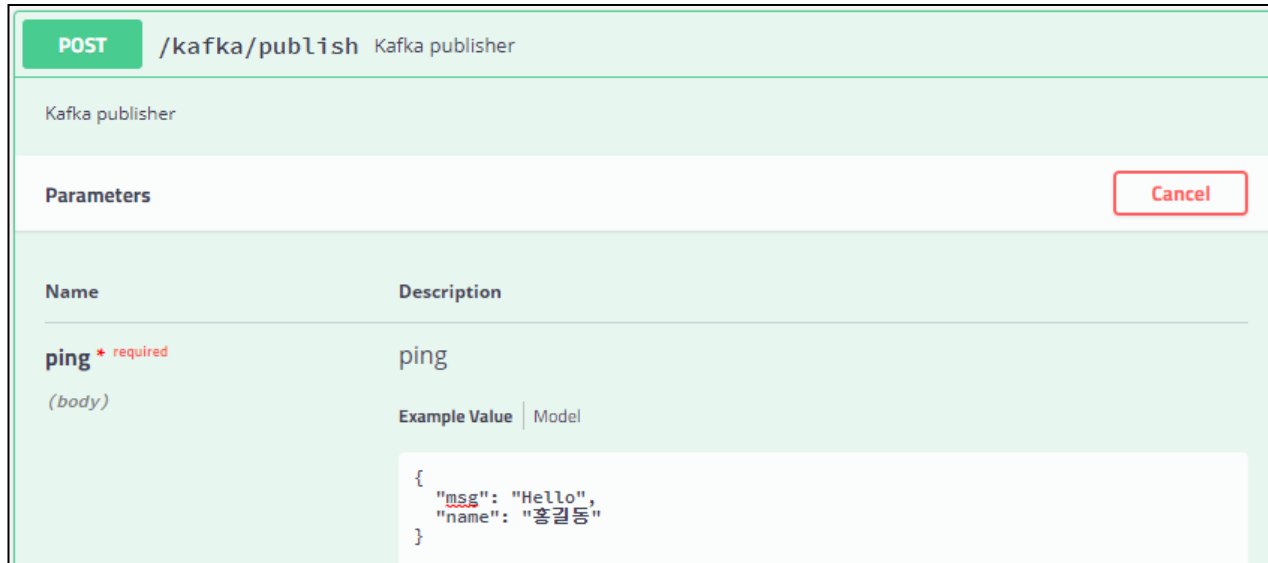
```
@Component
public class KafkaSubscriberService {

    public String working() {
        return "KafkaSubscriberService.working";
    }
}
```


3. 실행

1) Zookeeper/Kafka 서버 실행하고 publisher 실행

2) Swagger 에서 메시지 요청



The image shows a Swagger UI interface for a REST API. The top bar indicates a **POST** request to the endpoint `/kafka/publish`, with the description "Kafka publisher". Below this, the text "Kafka publisher" is repeated. A "Parameters" section is visible, with a "Cancel" button on the right. The parameters table has two columns: "Name" and "Description". A single parameter is listed: "ping" with a red asterisk and the text "required" next to it. Below the parameter name, it says "(body)". To the right of the parameter name, the description "ping" is shown. Below the description, there are tabs for "Example Value" and "Model". The "Example Value" tab is selected, showing a JSON object:

```
{  "msg": "Hello",  "name": "홍길동"}
```

Name	Description
ping * required (body)	ping

Example Value | Model

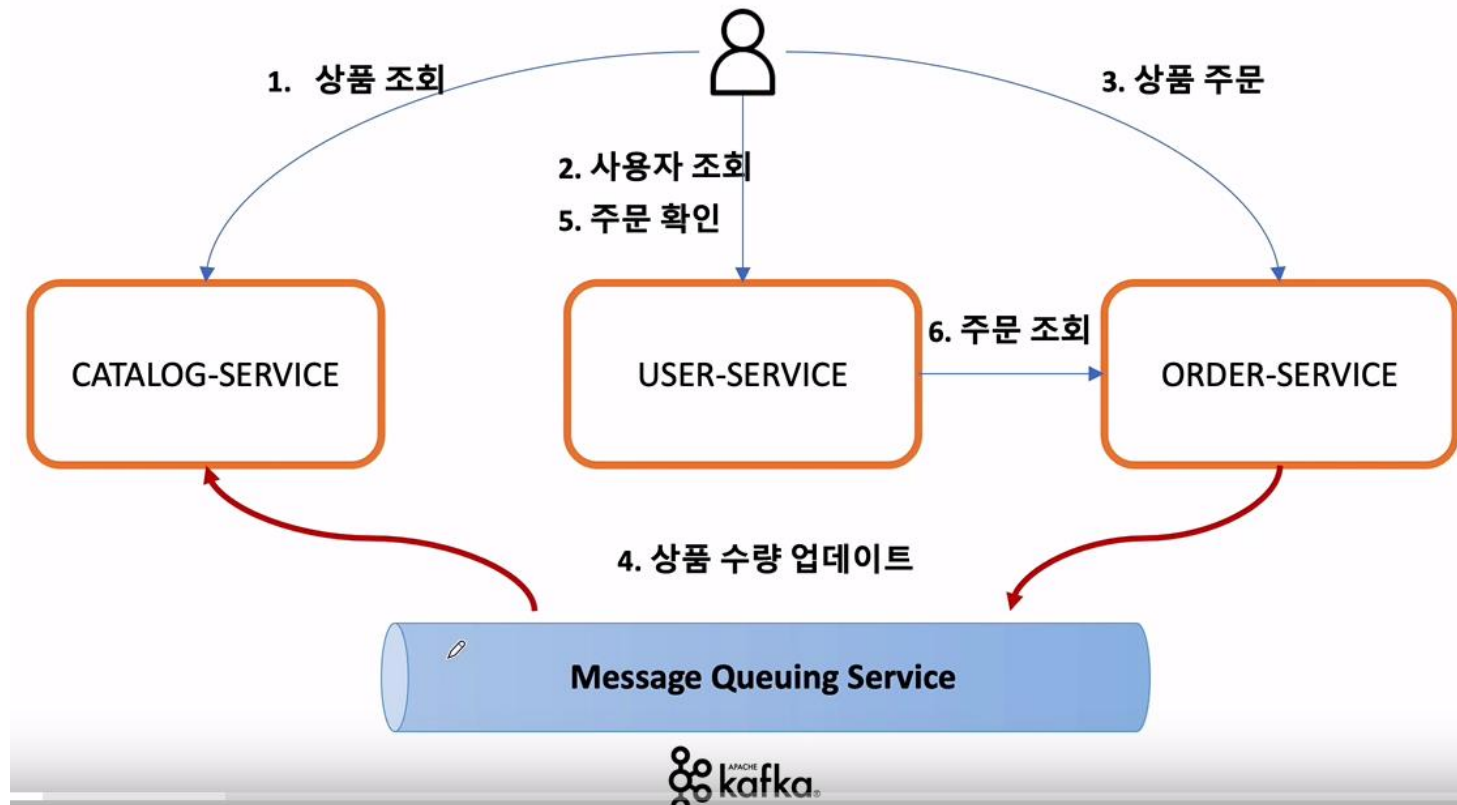
```
{  "msg": "Hello",  "name": "홍길동"}
```

3) subscriber 실행

Recieved ping message: Hello, 홍길동!
필요시 서비스 호출: KafkaSubscriberService.working

4. 데이터 동기화를 위한 Kafka 활용

- Order 서비스에 요청된 주문의 수량 정보를 Catalogs 서비스에 반영
- Order 서비스에서 Kafka Topic으로 메시지 전송 (producer)
- Catalogs 서비스에서 Kafka Topic에 전송된 메시지 취득 (consumer)



4. 데이터 동기화를 위한 Kafka 활용

Order 서비스에서 Publish 작업 처리

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
```

```
@Configuration
public class KafkaPublisherConfig {

    @Bean
    public ProducerFactory<String, Order> pingProducerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        configProps.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class);
        configProps.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, JsonSerializer.class);
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public KafkaTemplate<String, Order> pingKafkaTemplate() {
        return new KafkaTemplate<>(pingProducerFactory());
    }
}
```

```
@Component
public class OrderPublisher {

    @Autowired
    private KafkaTemplate<String, Order> orderKafkaTemplate;

    public void insertOrder(Order order) {
        ListenableFuture<SendResult<String, Order>> future = orderKafkaTemplate.send("msa", order);
    }
}
```

4. 데이터 동기화를 위한 Kafka 활용

```
@Service("orderService")
public class OrderServiceImpl implements OrderService {

    @Autowired
    OrderRepository orderRepository;
    @Autowired
    CustomerComposite customerComposite;
    @Autowired
    CustomerFeignClient customerFeignClient;

    @Autowired
    OrderPublisher orderPublisher;

    //RestTemplate
    @Override
    public int insertOrder(Order order) throws Exception {
        Customer customer = customerComposite.retrieveCustomer(order.getUserId());
        order.setName(customer.getName());

        ///// Kafka 연동/////
        orderPublisher.insertOrder(order);
        /////

        return orderRepository.insertOrder(order);
    }
}
```

4. 데이터 동기화를 위한 Kafka 활용

Catalogs 서비스에서 Subscribe 작업 처리

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
```

```
@Configuration
public class KafkaSubscriberConfig {

    public ConsumerFactory<String, Order> orderConsumerFactory() {
        Map<String, Object> props = new HashMap<>();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(ConsumerConfig.GROUP_ID_CONFIG, "pong"); // Consumer를 식별하는 고유 아이디.
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class);
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "false"); // offset을 주기적으로 commit 할지 여부
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest"); // earliest: 맨 처음부터 다시, latest: 이전꺼 무시 새로입력
        return new DefaultKafkaConsumerFactory<>(props, new StringDeserializer(),
            new JsonSerializer<>(Order.class, false));
    }

    @Bean
    public ConcurrentKafkaListenerContainerFactory<String, Order> orderKafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, Order> factory = new ConcurrentKafkaListenerContainerFactory<>();
        //ENABLE_AUTO_COMMIT_CONFIG= false로 지정했을때, 어떻게 commit할지를 지정한다.
        factory.getContainerProperties().setAckMode(ContainerProperties.AckMode.MANUAL_IMMEDIATE); // 즉각적으로 ack 요청한다.
        factory.setConsumerFactory(orderConsumerFactory());
        return factory;
    }
}
```

4. 데이터 동기화를 위한 Kafka 활용

```
@Component
public class CatalogsSubscriber {

    @Autowired
    CatalogsService catalogsService;

    @KafkaListener(topics = "msa", containerFactory = "orderKafkaListenerContainerFactory")
    public void pingListener(Order order, Acknowledgment ack) {
        try {
            System.out.println("Received Order message: " + order);

            int cnt = catalogsService.updateQuantity(order);
            System.out.println(order.getProductId()+" 에 해당하는 상품 갯수가 " + order.getQuantity() + " 만큼 감소되었습니다.");
            ack.acknowledge();
        } catch (Exception e) {
            String msg = "주문 정보에 따른 quantity 수정에 문제가 발생했습니다";
            System.out.println("Recieved ping message: " + msg + e);
        }
    }
}
```

```
@Service("catalogsService")
public class CatalogsServiceImpl implements CatalogsService {

    @Autowired
    CatalogsRepository catalogsRepository;

    @Override
    public int updateQuantity(Order order) throws Exception {
        return catalogsRepository.updateQuantity(order);
    }
}
```

```
@Mapper
public interface CatalogsRepository {

    public int updateQuantity(Order order) throws Exception;
}
```

```
<update id="updateQuantity"
parameterType="com.ecommerce.catalogs.subscriber.Order">
    update MSA_CATALOGS
    set stock = stock - #{quantity}
    where product_id = #{productId}
</update>
```

4. 데이터 동기화를 위한 Kafka 활용

실행전 데이터

Order 서비스의 msa_order 테이블 데이터

	orderid [PK] character varying (50)	userid character varying (50)	name character varying (20)	productid character varying (50)	quantity integer	unitprice integer
1	A1111	1111	홍길동	CATALOG-0001	10	500
2	B1111	1111	홍길동	CATALOG-0002	5	200

Catalogs 서비스의 msa_catalogs 테이블 데이터

	product_id [PK] character varying (50)	product_name character varying (50)	stock integer	price integer	create_date date
1	CATALOG-0001	Java	100	1500	2021-05-15
2	CATALOG-0002	SQL	100	900	2021-05-15
3	CATALOG-0003	JSP	100	1200	2021-05-15

4. 데이터 동기화를 위한 Kafka 활용

실행후 데이터

Order 서비스의 msa_order 테이블 데이터

	orderid [PK] character varying (50)	userid character varying (50)	name character varying (20)	productid character varying (50)	quantity integer	unitprice integer	totalprice integer	create_date date
1	A1111	1111	홍길동	CATALOG-0001	10	500	5000	2021-05-15
2	B1111	1111	홍길동	CATALOG-0002	5	200	1000	2021-05-15
3	66ecda1d-9847-4e99-a12e-edede...	2222	박문각	CATALOG-0001	10	500	5000	2021-05-15

publish



subscribe

Catalogs 서비스의 msa_catalogs 테이블 데이터

	product_id [PK] character varying (50)	product_name character varying (50)	stock integer	price integer	create_date date
1	CATALOG-0002	SQL	100	900	2021-05-15
2	CATALOG-0003	JSP	100	1200	2021-05-15
3	CATALOG-0001	Java	90	1500	2021-05-15