<https://www.udemy.com/course/introduction-to-kafka-with-spring-boot>

## 1. Install and Launch Kafka Server

### 1.1 Install Kafka in WSL

* Download package to /home/dxu/apps
* mkdir kafka and extract kafka\_2.13-3.7.0.tgz to this folder

> tar -xvf ../kafka\_2.13-3.7.0.tgz

* Set cluster id

> cd kafka\_2.13-3.7.0

> KAFKA\_CLUSTER\_ID="$(bin/kafka-storage.sh random-uuid)"

> echo $KAFKA\_CLUSTER\_ID

* Format logs

> bin/kafka-storage.sh format -t $KAFKA\_CLUSTER\_ID -c config/kraft/server.properties



* Start Kafka Server

> bin/kafka-server-start.sh config/kraft/server.properties

* Shut down Kafka Server

> bin/kafka-server-stop.sh

### 1.2 Docker

Create docker-compose.yml:

version: '2'  
services:  
 zookeeper:  
 image: confluentinc/cp-zookeeper:7.4.4  
 environment:  
 ZOOKEEPER\_CLIENT\_PORT: 2181  
 ZOOKEEPER\_TICK\_TIME: 2000  
 ports:  
 - 22181:2181  
  
 kafka:  
 image: confluentinc/cp-kafka:7.4.4  
 depends\_on:  
 - zookeeper  
 ports:  
 - 29092:29092  
 environment:  
 KAFKA\_BROKER\_ID: 1  
 KAFKA\_ZOOKEEPER\_CONNECT: zookeeper:2181  
 KAFKA\_ADVERTISED\_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT\_HOST://localhost:29092  
 KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT\_HOST:PLAINTEXT  
 KAFKA\_INTER\_BROKER\_LISTENER\_NAME: PLAINTEXT  
 KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR: 1

> docker compose up -d

## 2. Send and Receive Message

### 2.1 Receive

> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic my.first.topic

### 2.2 Send

> bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic my.first.topic

> >my first message

## 3. Topic CLI Tools

### 3.1 List Topics

> bin/kafka-topics.sh --bootstrap-server localhost:9092 --list

### 3.2 Create Topic

> bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic my.new.topic

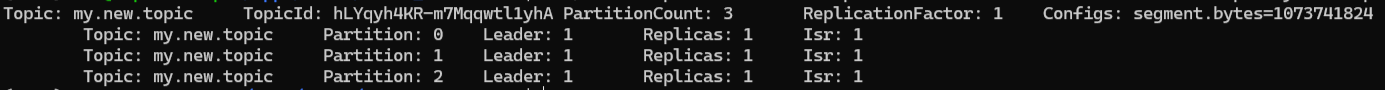
### 3.3 Show Information of a Topic

> bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic my.new.topic



### 3.4 Update a Topic

> bin/kafka-topics.sh --bootstrap-server localhost:9092 --alter --topic my.new.topic --partitions 3

Now if show the topic information again, it shows:  


### 3.5 Delete a Topic

> bin/kafka-topics.sh --bootstrap-server localhost:9092 --delete --topic my.new.topic

## 3. Consumer Group CLI Tools

> bin/kafka-topics.sh --bootstrap-server localhost:9092 --create --topic cg.demo.topic --partitions 5

### 3.1 List Consumer Groups

> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 –list

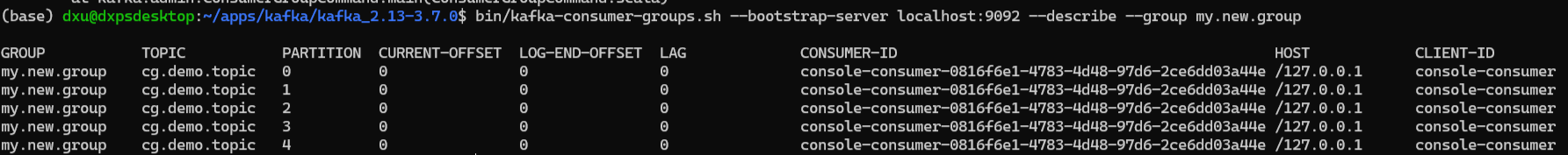
### 3.2 Set Consumer Group for a Topic

> bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic cg.demo.topic --group my.new.group

List consumer groups again to check the new consumer group “my.new.group”

### 3.3 Show Consumer Group Info

> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my.new.group



### 3.4 Check Consumer Group State

> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my.new.group --state



3.5 Show Members of a Consumer Group

> bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my.new.group –members



## 4. Coding Kafka with Spring Boot

### 4.1 Consumer

Create microservice “dispatch”. Commit #2

https://github.com/freeever/spring-kafka-intro

Start DispatchApplication

Start Kafka Server at 9092

Send message(s) to Kafka topic “order.created”.

**Result:**

The KafkaListener in the Spring Boot application consumes the message, and print in the console:  


### 4.2 JSON Deserializer and Error Handling

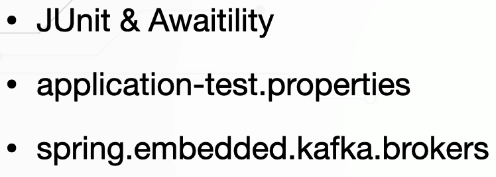
### 4.3 Produce

## 5. Spring Boot Integration Test



@DirtiesContext: This annotation is particularly useful when a test has changed the state of the application context in such a way that it cannot be easily reset and might affect subsequent tests. Using @DirtiesContext ensures that subsequent tests will operate in a clean environment, avoiding interference from changes made by the dirtied context.

@EmbeddedKafka: Provided by Spring Kafka Test library, which results in an embedded Kafka broker being started that the test and the application will connect to.



Awaitility: Wait until a condition to become true, which is necessary with asynchronous messaging. E.g. await for the outbound events to be consumed by the test consumer.