Introduction to Kafka with Spring Boot

Table of Contents

1. Kafka Overview

Udemy Course: https://www.udemy.com/course/introduction-to-kafka-with-spring-boot ⌐

1.1 What Is Kafka?

Apache Kafka is a **distributed event streaming platform** used to:

- **Publish and subscribe** to streams of records (events/messages)
- **Store** streams of records in a fault-tolerant way
- **Process** streams of events in real time

It's commonly used for **event-driven systems**, **data pipelines**, **microservices coordination**, and **real-time analytics**.

1.2 Kafka Core Concepts

Concept	Description
Producer	Sends (publishes) messages to Kafka topics
Consumer	Reads messages from Kafka topics
Topic	A logical channel where messages are published (like a queue)
Partition	Each topic is split into partitions (parallelism, scalability)
Broker	A Kafka server that stores and serves messages
Consumer Group	A group of consumers that share the work of consuming messages from a topic
Offset	The position of a message in a partition (Kafka keeps track of what's been read)

1.3 Kafka in Microservices Architecture

| Benefit	Explanation
Loose coupling	Services don't need to call each other directly
Resilience	Consumers can restart and pick up from the last processed message
Scalability	Kafka supports millions of messages/sec with multiple partitions
Asynchronous workflows	Great for Sagas, audit logging, notifications, etc.

1.4 Real-World Use Cases

- **Microservices Communication** (e.g., user signup, order fulfillment)
- **Audit Logs** (e.g., who updated what and when)
- **Data Pipelines** (e.g., ingest data and send to multiple systems)

- **Sagas and Event-Driven Transactions**

2. Message Driven Architectures

2.1 Reactive Manifesto

https://www.reactivemanifesto.org/⧉

2.2 The Reactive Principles

https://www.reactiveprinciples.org/⧉
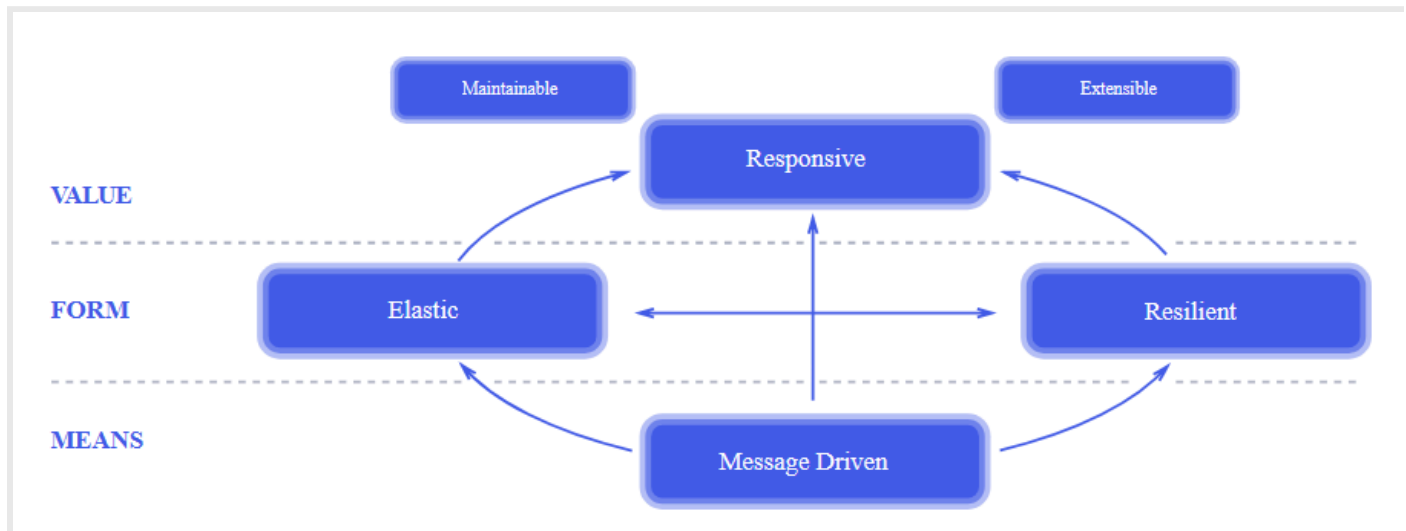
- Stay responsive
- Accept uncertainty
- Embrace failure
- Assert autonomy
- Tailor consistency
- Decouple time
- Decouple space
- Handle dynamics

2.3 The Reactive Patterns

- Partition State
- Communicate Facts
- Isolate Mutations
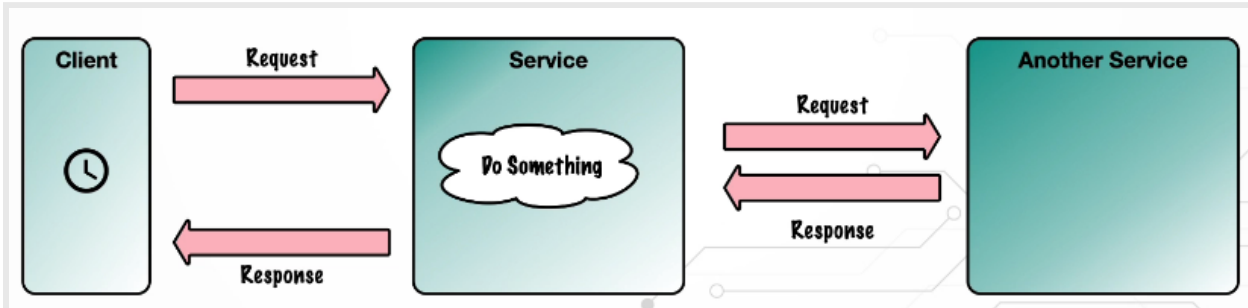- Coordinate Dataflow
- Localize State
- Observe Communications

2.4 Enterprise Integration Patterns (EIP)

Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions⧉

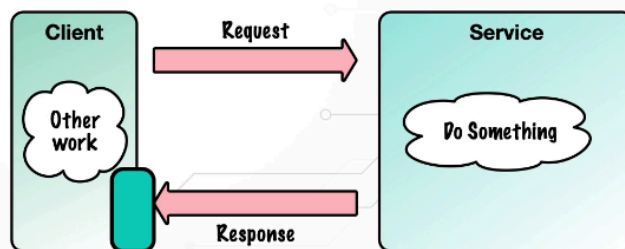# 3. Messaging with Kafka

3.1 Sync and Async Communication

- Sync - Blocking

- Async - Non-Blocking

- Event Driven Communication
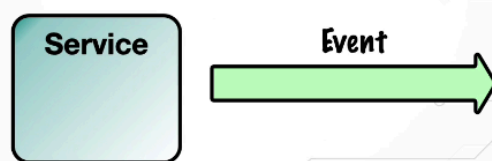

3.2 Kafka & JMS & AMQP

3.2.1 JMS - ActiveMQ
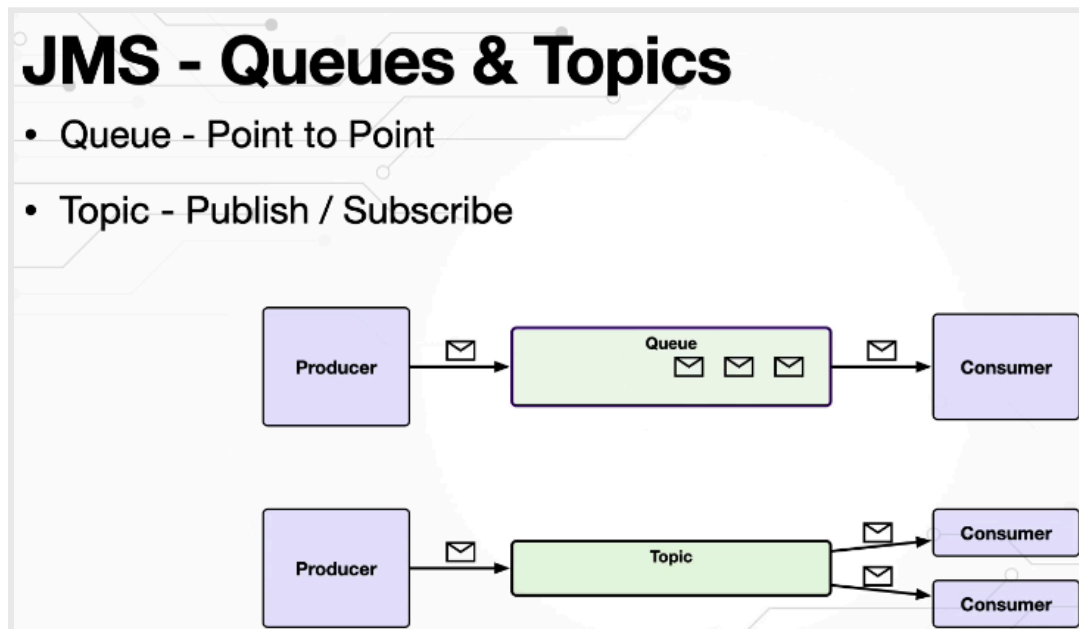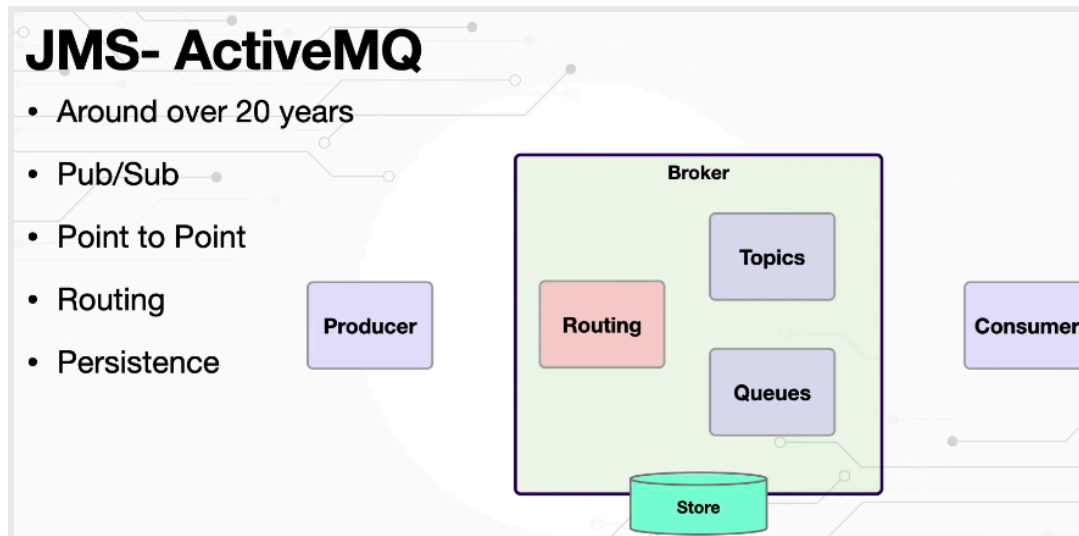
Queue:

- Each message has only one consumer.
- If consumer not available, the message is **retained** until the message is read by the consumer.
- No messages are lost.

Topic (pub/sub):

- Topic can have multiple consumers for each message
- Topic does **not retain** messages. If consumer isn't online when the message is received by the broker, the consumer will never see that message
- Message could be lost

ActiveMQ also support **Virtual Topic** - Hybrid or queues and topics.

3.2.2 AMQP - RabbitMQ
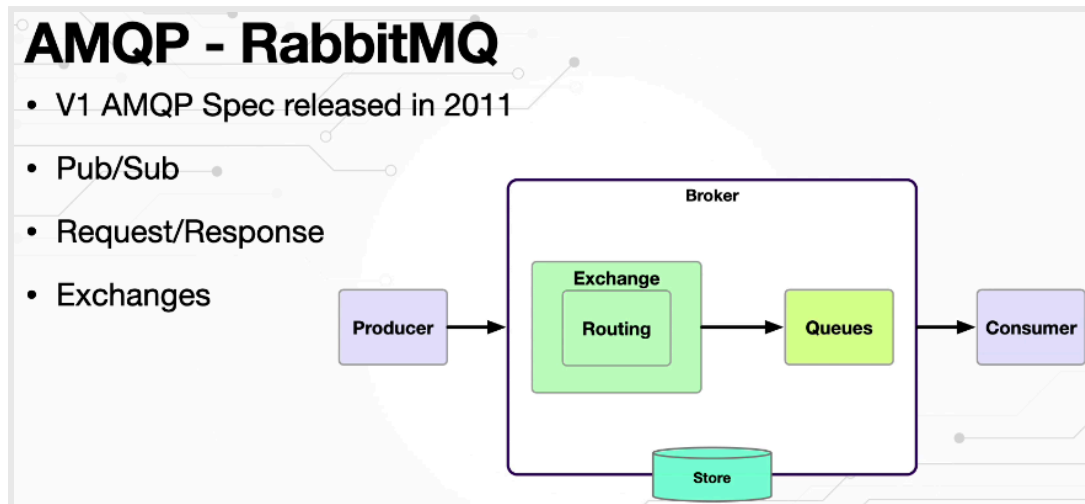
- Message is sent **from producer to the broker**
- An **exchange** is **registered** in the broker
- Combination of **rules and the exchange** determine to which queue or queues

3.2.3 Kafka

- Brokers, Producers, Consumers
  - **Producer** - Sends to Broker
  - **Consumer** - **Pulls** from Broker. Consumer asks for the next message when it has the capacity to process
  - **Broker** - Message Store
- Topics, Partitions, Logs, Offsets
  - **Topics** - Message exchange is done via Topics. A topic is an ordered sequence of events or messages which are stored in a durable way.
  - **Logs**: Is essentially **files** written to disk. **Append only**. **Immutable** which means that once events are added to the topic, the content will not be chagned.
  -
  - **Offsets**: An offset is a **reference** to your **read position**.
  - **Partitions**: High throughput. **Sticky**.
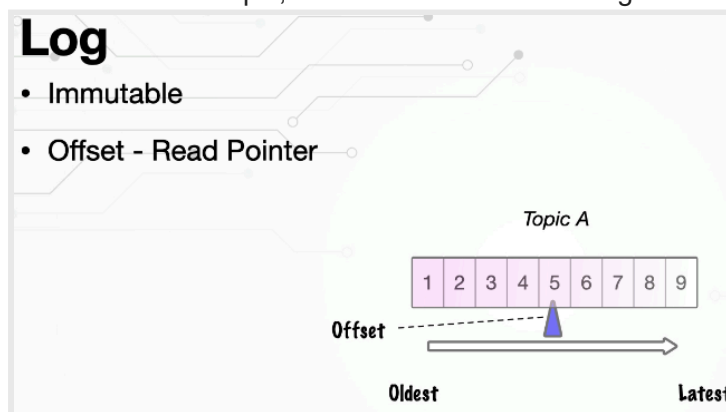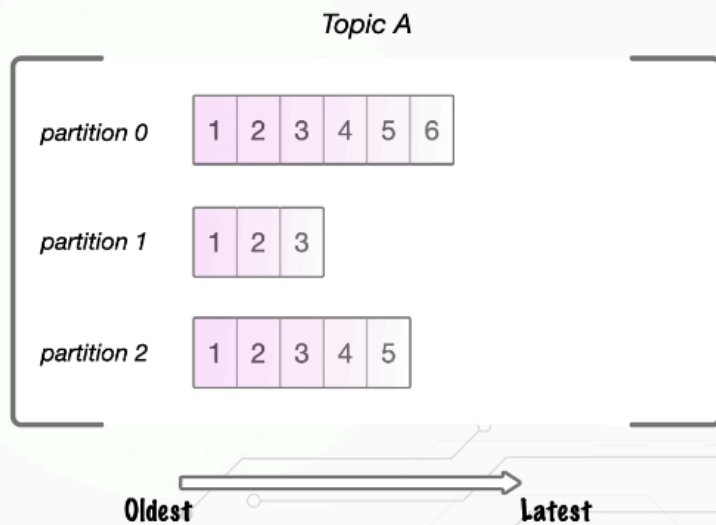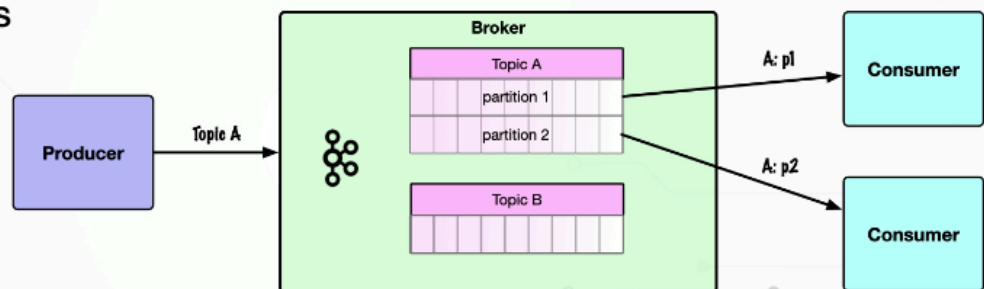
- **Keys** - Determine which **partition** the message is **written to**. Should be **consistent** such as a user ID means that each message with that same user ID would be written to the same partition. This guarantees that events are processed in order as the same consumer will read events from a given partition.
- **Consumer Groups**
- Clusters, Replication
    - **Cluster** - Kafka's **Resiliency**
    - **Replication** -
- Leader Partitions, Follower Partitions
    - **Leader Partitions** - receive messages from the producer. Then these events are replicated to the follower partitions
    - **Follower Partitions** - Never directly receive events from the producer, only via replication

Summary:
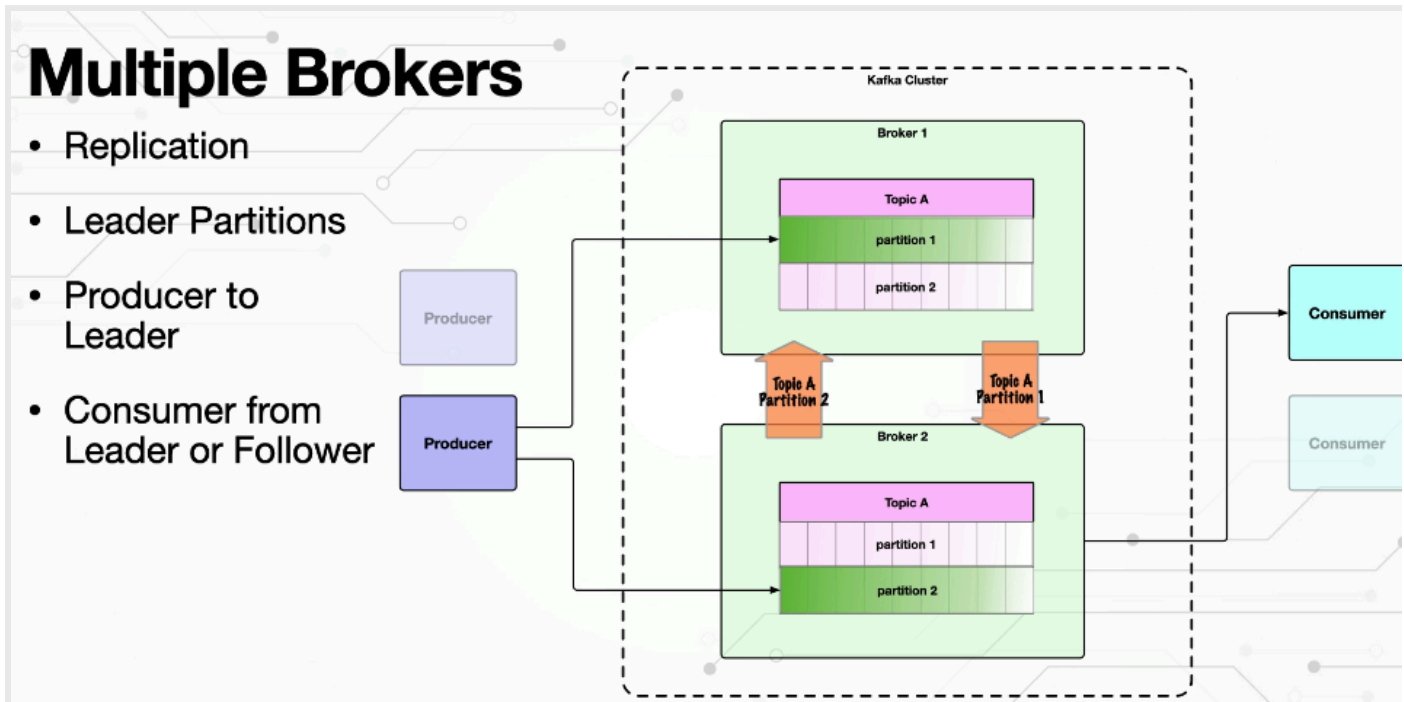
- **Producer** sends keyed messages to the brokers. Based on the **key**, the producer knows which broker to direct the messages to. In the above case,
    - messages on **topic A** with **key 1** are sent to **broker 1 partition 1** (leader partition)
    - messages on **topic A** with **key 2** are sent to **broker 2 partition 2** (leader partition)
- **Consumer** - Our single consumer reads from broker 2

Why Kafka?

- Pub sub messaging
- Realtime stream processing
- Fault tolerant
- High availability
- High scalability
- High throughput

3.3 Message vs Event

- **Message** - is something sent over Asynchronous communications. It's the **Envelope**.
- **Event** - is something that happened. It's the **letter** within the envelope.

3.3.1 Kafka Message

- Key - important for message process ordering
- Headers - meta information
- Payload - Holds the event
    - Max default size 1Mb
    - Consider sensitive data
    - Is data sufficient for the consumers

4 Installing and Running Kafka

4.1 Install Kafka Locally on Windows

4.1.1 Installation

- **wsl - Need specifical handling when the Spring Boot application on Windows wants to connect to it. See the Spring Boot application section**
- JDK

```
> wsl --status
> launch ubuntu
> sudo apt update
> cd ~
> mkdir tools
```

- download Kafka binary (4.0) to /home/dxu/tools
- extract

```
> mkdir kafka
> cd kafka
> tar -xvf ../kafka_2.13-4.0.0.tgz
> cd  kafka_2.13-4.0.0
```

4.1.2 Initialize and Start up the Kafka Broker

Kafka 4.0.0 only supports **KRaft mode** (Zookeeper has been fully removed)

4.1.2.1 Setup Kafka 4.0.0 (Single-Node KRaft Mode)

- Edit config/**server.properties**

```
# Unique ID for this broker
node.id=1

# Cluster controller quorum
controller.quorum.voters=localhost:9093

# List of addresses this broker will listen on
listeners=PLAINTEXT://localhost:9092,CONTROLLER://localhost:9093

# Bind listener names to ports
listener.security.protocol.map=CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT

# Define controller listener name
controller.listener.names=CONTROLLER

# Storage directory
log.dirs=/tmp/kraft-logs
```

- **Set up the KAFKA_CLUSTER_ID** env variable which is a unique identifier

```
> export KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"
> echo $KAFKA_CLUSTER_ID
```

- **Format the log directory** location which is a directory on disk that will be used for the persistent store of the message topics and all the meta data for Kafka

```
> bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c config/server.properties --standalone
Result:
Formatting dynamic metadata voter directory /tmp/kraft-logs with metadata.version 4.0-IV3.
```

4.1.2.2 Start Kafka

```
bin/kafka-server-start.sh config/server.properties
```

Create startup script **start-kafka.sh** for convenience (**must be in unix format**):

```
#!/bin/bash

KAFKA_DIR="$HOME/tools/kafka/kafka_2.13-4.0.0"
LOG_DIR="/tmp/kraft-logs"
SERVER_CONFIG="$KAFKA_DIR/config/server.properties"

echo "📦  Kafka directory: $KAFKA_DIR"
echo "📁  Log directory: $LOG_DIR"

# Configure server.properties if needed (as before)...

# ✅ Only format if log directory does not exist
if [ ! -f "$LOG_DIR/meta.properties" ]; then
  echo "🆕 First time setup: formatting storage"
  export KAFKA_CLUSTER_ID="$($KAFKA_DIR/bin/kafka-storage.sh random-uuid)"
  $KAFKA_DIR/bin/kafka-storage.sh format -t "$KAFKA_CLUSTER_ID" -c "$SERVER_CONFIG" --standalone
else
  echo "ℹ️   Kafka storage already initialized. Skipping format."
fi

echo "🚀 Starting Kafka..."
$KAFKA_DIR/bin/kafka-server-start.sh "$SERVER_CONFIG"
```

4.2 Sending and Receiving

- **Create script consumer.sh to start the consumer**

> **./consumer my.first.topic**

```
#!/bin/bash
echo $1

cd $HOME/tools/kafka/kafka_2.13-4.0.0

bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic $1
```

- **Create script producer.sh to start the producer**

```
#!/bin/bash
echo $1

cd $HOME/tools/kafka/kafka_2.13-4.0.0

bin/kafka-console-producer.sh --bootstrap-server localhost:9092 --topic $1
```

> ./producer.sh my.first.topic

it will show "**>**" which means it's ready to send message

Type some message to test:

In the consumer window, it will show the message consumed:

4.3 CLI Tools - Topic Tool

1. List topics

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

2. Create a new topic

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

3. Modify an existing topic

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

4. Delete a topic

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

5. View details of a topic

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

```
(base) dxu@dxpsdesktop:~/tools/kafka/kafka_2.13-4.0.0$ bin/kafka-topics.sh --bootstrap-server localhost:9092 --describe --topic my.first.t
opic
Topic: my.first.topic    TopicId: Yo9p0CUHQneL5lzw5GLylA PartitionCount: 3        ReplicationFactor: 1      Configs: segment.bytes=1073741824
        Topic: my.first.topic    Partition: 0      Leader: 1      Replicas: 1      Isr: 1  Elr:        LastKnownElr:
        Topic: my.first.topic    Partition: 1      Leader: 1      Replicas: 1      Isr: 1  Elr:        LastKnownElr:
        Topic: my.first.topic    Partition: 2      Leader: 1      Replicas: 1      Isr: 1  Elr:        LastKnownElr:
```

4.4 CLI Tools - Consumer Group Tool

Consumer Group - A collection of consumers working together

When we start consumer, we can specify the consumer group. If not specify, the system generates the consumer group

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

1. List consumer groups

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

```
(base) dxu@dxpsdesktop:~/tools/kafka/kafka_2.13-4.0.0$ bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --list
console-consumer-72586
my.new.group
```

2. Getting details about a consumer group

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

```
(base) dxu@dxpsdesktop:~/tools/kafka/kafka_2.13-4.0.0$ bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group my.new.group

GROUP           TOPIC          PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG        CONSUMER-ID                                              HOST          CLIENT-ID
my.new.group    cg.demo.topic  0          0               0               0          console-consumer-f723dfa3-46fc-4652-97bd-6d124138f32b  /127.0.0.1    console-consumer
```

5. Implement Spring Boot Application with Kafka

> ⚠️ For the code and configuration details, see [Kafka Integration](#)

5.1 Troubleshooting

> ❗ Type your error message here.

✅ **Kafka in WSL2: How to Connect from Windows Apps**

When Kafka runs under **WSL2**, you **cannot** use localhost:9092 from a Windows-based Spring Boot app. Instead:

🔧 **Step 1: Update Kafka server.properties**

Replace localhost with your **WSL2 instance's IP address**:

server.properties

```
listeners=PLAINTEXT://<WSL2-IP>:9092,CONTROLLER://<WSL2-IP>:9093
advertised.listeners=PLAINTEXT://<WSL2-IP>:9092,CONTROLLER://<WSL2-IP>:9093
```

🔧 **Step 2: Update Spring Boot config (application.yml or .properties)**

```yaml
spring:
  kafka:
    bootstrap-servers: 172.27.223.47:9092
```

> ⚠️ 🔄 Important Notes:
>
> - If WSL2 restarts, the IP may change. You can write a script to auto-update it or assign a static IP using advanced WSL network bridging.
> - Don't forget to open port 9092 on your WSL2 firewall if needed.