## ****Module 1: Kafka Consumer Introduction****

### ****Step 2: Create a Simple Kafka Producer and Consumer in Java****

#### ****Create a Kafka Topic****

Run the following command:

bin/kafka-topics.sh --create --topic test-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

#### ****Create a Java Maven Project****

mvn archetype:generate -DgroupId=com.kafka -DartifactId=kafka-microservices -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

cd kafka-microservices

#### ****Add Kafka Dependencies (****pom.xml****)****

<dependencies>

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>kafka-clients</artifactId>

<version>3.5.1</version>

</dependency>

</dependencies>

#### ****Implement Kafka Producer****

import org.apache.kafka.clients.producer.\*;

import java.util.Properties;

public class KafkaProducerExample {

public static void main(String[] args) {

String topic = "test-topic";

Properties props = new Properties();

props.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9092");

props.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringSerializer");

props.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringSerializer");

KafkaProducer<String, String> producer = new KafkaProducer<>(props);

for (int i = 0; i < 10; i++) {

ProducerRecord<String, String> record = new ProducerRecord<>(topic, "Message " + i);

producer.send(record);

}

producer.close();

}

}

#### ****Implement Kafka Consumer****

import org.apache.kafka.clients.consumer.\*;

import java.time.Duration;

import java.util.Collections;

import java.util.Properties;

public class KafkaConsumerExample {

public static void main(String[] args) {

String topic = "test-topic";

Properties props = new Properties();

props.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9092");

props.put(ConsumerConfig.GROUP\_ID\_CONFIG, "group1");

props.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

props.put(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, "org.apache.kafka.common.serialization.StringDeserializer");

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

consumer.subscribe(Collections.singletonList(topic));

while (true) {

ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));

for (ConsumerRecord<String, String> record : records) {

System.out.println("Received: " + record.value());

}

}

}

}

#### ****Run the Producer and Consumer****

1. Run the consumer first:

mvn compile exec:java -Dexec.mainClass="KafkaConsumerExample"

1. Run the producer in another terminal:

mvn compile exec:java -Dexec.mainClass="KafkaProducerExample"

## ****Module 2: Kafka Streams****

Kafka Streams is used for real-time data processing.

### ****Step 3: Implement a Kafka Streams Application****

#### ****Add Kafka Streams Dependency****

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>kafka-streams</artifactId>

<version>3.5.1</version>

</dependency>

#### ****Create a Kafka Streams Application****

import org.apache.kafka.common.serialization.Serdes;

import org.apache.kafka.streams.\*;

import org.apache.kafka.streams.kstream.\*;

import java.util.Properties;

public class KafkaStreamsExample {

public static void main(String[] args) {

Properties props = new Properties();

props.put(StreamsConfig.APPLICATION\_ID\_CONFIG, "streams-app");

props.put(StreamsConfig.BOOTSTRAP\_SERVERS\_CONFIG, "localhost:9092");

props.put(StreamsConfig.DEFAULT\_KEY\_SERDE\_CLASS\_CONFIG, Serdes.String().getClass().getName());

props.put(StreamsConfig.DEFAULT\_VALUE\_SERDE\_CLASS\_CONFIG, Serdes.String().getClass().getName());

StreamsBuilder builder = new StreamsBuilder();

KStream<String, String> sourceStream = builder.stream("test-topic");

KStream<String, String> transformedStream = sourceStream.mapValues(value -> value.toUpperCase());

transformedStream.to("output-topic");

KafkaStreams streams = new KafkaStreams(builder.build(), props);

streams.start();

}

}

#### ****Create an Output Topic****

bin/kafka-topics.sh --create --topic output-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1

#### ****Run Kafka Streams****

mvn compile exec:java -Dexec.mainClass="KafkaStreamsExample"

Now, any message sent to test-topic will be converted to uppercase and stored in output-topic.

## ****Module 3: Kafka Streams Microservice****

### ****Step 4: Build a Kafka Streams-Based Microservice****

To integrate with a Spring Boot microservice:

#### ****Add Spring Boot and Kafka Dependencies****

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.kafka</groupId>

<artifactId>spring-kafka</artifactId>

</dependency>

#### ****Spring Boot Kafka Streams Configuration****

import org.apache.kafka.streams.StreamsBuilder;

import org.apache.kafka.streams.kstream.KStream;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

public class KafkaStreamsConfig {

@Bean

public KStream<String, String> processStream(StreamsBuilder builder) {

KStream<String, String> stream = builder.stream("test-topic");

stream.mapValues(value -> value.toUpperCase()).to("output-topic");

return stream;

}

}

#### ****Run the Spring Boot Application****

mvn spring-boot:run

## ****Final Testing****

1. **Start Kafka, Zookeeper, and Topics**
2. **Run Spring Boot Microservice**
3. **Send Messages to Kafka Producer**
4. **Observe Processed Messages in output-topic**