Very Simple Spring Boot with Kafka – 2022

**How to setup kafka in windows 10**

Source Link: https://www.goavega.com/install-apache-kafka-on-windows/

After extracting folder, create “**data**” folder and inside data, create **kafka** and **zookeeper** folder.

Update zookeeper data directory path in “**config/zookeeper.properties**” configuration file.

**dataDir=C:/devsoftware/kafka\_2.12-3.2.0/data/zookeeper**

Update Apache Kafka log file path in “**config/server.properties**” configuration file.

**log.dirs=C:/devsoftware/kafka\_2.12-3.2.0/data/kafka**

**Go to directory in command prompt**

**E:\devsoftwares\kafka\_2.13-3.2.0\bin\windows**

**Start Zookeeper with command**: **zookeeper-server-start.bat ../../config/zookeeper.properties**

**Start Kafka Server with command**: **kafka-server-start.bat ../../config/server.properties**

Create a topic using the following command

**kafka-topics.bat --create --topic testTopic1 --bootstrap-server localhost:9092**

See the list of topics: **kafka-topics.bat --bootstrap-server=localhost:9092 --list**

**Maven (pom.xml)**

<parent>

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

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

<version>2.0.5.RELEASE</version>

<relativePath />

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>11</java.version>

</properties>

<dependencies>

<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>org.springframework.boot</groupId>

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

<scope>test</scope>

</dependency>

</dependencies>

**SpringBoot Main Application**

@SpringBootApplication

**public** **class** KafkaSpringBootMain {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(KafkaSpringBootMain.**class**, args);

}

}

**Sample Controller**

@RestController

@RequestMapping(value = "/kafka")

**public** **class** SampleController {

@Autowired

**private** ProducerService producer;

@PostMapping(value = "/publish")

**public** ResponseEntity<String> sendInfo(@RequestParam("msg") String msg) {

producer.sendMessage(msg);

**return** **new** ResponseEntity<String>("Success", HttpStatus.***OK***);

}

}

**public** **interface** ProducerService {

**void** sendMessage(String msg);

}

**Service Layer**

**Producer**

**@Service**

**public** **class** ProducerServiceImpl **implements** ProducerService {

@Value(value = "${kafka.topic.name}") **🡸 Topic name from application.yml**

**private** String topicName;

@Autowired

**private** **KafkaTemplate<String, String> kafkaTemplate**;

@Override

**public** **void** sendMessage(String msg) {

**kafkaTemplate.send(topicName, msg);**

System.***out***.println("Msg sent successfully ...");

}

}

**Consumer**

**@Service**

**public** **class** ConsumerService {

**@KafkaListener(topics = "${kafka.topic.name}") 🡸 Topic name from application.yml**

**public** **void** consume(String msg) {

System.***out***.println("Message received by consumer as: "+msg);

}

}

**Spring Application Configuration (application.yml)**

spring:

kafka:

consumer:

bootstrap-servers: localhost:9092

group-id: group\_id

auto-offset-reset: earliest

**key-deserializer**: org.apache.kafka.common.serialization.StringDeserializer

**value-deserializer**: org.apache.kafka.common.serialization.StringDeserializer

producer:

bootstrap-servers: localhost:9092

**key-serializer**: org.apache.kafka.common.serialization.StringSerializer

**value-serializer**: org.apache.kafka.common.serialization.StringSerializer

kafka:

topic:

name: testTopic1

**It can also written as**

server:

port: 9000

spring.kafka:

consumer:

bootstrap-servers: localhost:9092

group-id: group\_id

auto-offset-reset: earliest

key-deserializer: org.apache.kafka.common.serialization.StringDeserializer

value-deserializer: org.apache.kafka.common.serialization.StringDeserializer

producer:

bootstrap-servers: localhost:9092

key-serializer: org.apache.kafka.common.serialization.StringSerializer

value-serializer: org.apache.kafka.common.serialization.StringSerializer

kafka.topic.name: testTopic1

**application.properties**

server.port=9000

#Kafka Producer

spring.kafka.producer.bootstrap-servers=localhost:9092

**spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer**

**spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer**

# Kafka Consumer

spring.kafka.consumer.bootstrap-servers=localhost:9092

**spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer**

**spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer**

spring.kafka.consumer.group-id=saga-order-grp-id

spring.kafka.consumer.auto-offset-reset=latest

**What is groupId in Kafka?**

A **consumer group** is a set of consumers which cooperate to consume data from some topics. The partitions of all the topics are divided among the consumers in the group. As new group members arrive and old members leave, the partitions are re-assigned so that each member receives a proportional share of the partitions. This is known as rebalancing the group. The group ID is very important to how different consumers "**load balance**" partitions. For example, if you have a topic with 10 partitions then two consumers with the same groupId will read from 5 partitions each. If you have two consumers with different group ids, both consumers will read from 10 partitions. In this sense, the groupId is how you define a "consumer group" or group of consumers reading from a given topic/partitions.

Understanding what a consumer group is in Kafka requires the understanding of what a partition is first.

**Topics store data in partitions**. **These partitions are key to Kafka's scalability and resiliency** with respect to data loss. **Partitions work to store and replicate a topic's data across all available nodes in the cluster**.

**Consumers read from topics. Since topics are collections of partitions,** consumers exist in collections to distribute the work load of reading data evenly among all available worker "threads" or consumers in the "consumer group". Put another way, consumers exist as groups to read from the groups of partitions. This is why it's so important to give a consumer a group id.

**Offset Management**

After the consumer receives its assignment from the coordinator, it must determine the initial position for each assigned partition. When the group is first created, before any messages have been consumed, the position is set according to a configurable offset reset policy (auto.offset.reset). Typically, consumption starts either at the **earliest** offset or the **latest** offset.

Earliest — when the consumer application is initialized the first time or binds to a topic and wants to consume the historical messages present in a topic, the consumer should configure **auto. offset. reset** to earliest. Latest — This is the default offset reset value if you have not configured any.

**Spring Cloud Kafka Stream – 2025**

Kafka Streams simplifies application development by building on the Kafka producer and consumer libraries and leveraging the native capabilities of Kafka to offer data parallelism, distributed coordination, fault tolerance, and operational simplicity.

**Where to use Consumer - Producer:**

1. If there are single consumers, consume the message process but not spill to other topics.
2. Batch processing - if there is a requirement to collect a message or kind of batch processing it's good to use a normal traditional way.

**Where to use Kafka Stream:**

1. If you consume messages from one topic, transform and publish to other topics Kafka Stream is best suited.
2. Real-time processing, real-time analytics, and Machine learning.

Complete Example on Kafka Stream Producer and Consumer. Let us create two microservices, one for **producer** and another for **consumer**.

**Relevant Pom.xml**

<parent>

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

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

<version>3.4.2</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

**<java.version>17</java.version>**

**<spring-cloud.version>2024.0.0</spring-cloud.version>**

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>kafka-streams</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-stream</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-stream-binder-kafka</artifactId>

</dependency>

</dependencies>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-dependencies</artifactId>

<version>${spring-cloud.version}</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

**Relevant Code for Producer**

**application.properties for Producer**

server.port=8082

**spring.cloud.stream.kafka.binder.brokers=localhost:9092**

**spring.cloud.stream.bindings.producer-out-0.destination**=testTopic1

**spring.cloud.stream.bindings.producer-out-0.producer.useNativeEncoding**=true

**spring.cloud.stream.kafka.bindings.producer-out-0.producer.configuration.value.serializer**= **com.ddlab.rnd.serializer.MessageSerializer**

**MessageSerializer for Producer**

It is required to convert Employee object type to Byte array.

**public** **class** MessageSerializer **implements** Serializer<Employee> {

**private** **final** ObjectMapper objectMapper = **new** ObjectMapper();

@Override

**public** **byte**[] serialize(String topic, Employee data) {

**try** {

**return** objectMapper.writeValueAsBytes(data);

} **catch** (JsonProcessingException e) {

**throw** **new** SerializationException(e);

}

}

}

**Producer Code**

@Component

**public** **class** KafkaProducer {

@Autowired

**private** StreamBridge streamBridge;

**public** **void** sendMessage(Employee emp) {

streamBridge.send("producer-out-0", emp);

}

}

**MessageDeSerializer for Consumer**

It is required to convert Byte array to Employee object type.

**public** **class** MessageDeSerializer **implements** Deserializer<Employee> {

**private** **final** ObjectMapper objectMapper = **new** ObjectMapper();

@Override

**public** Employee deserialize(String topic, **byte**[] data) {

**try** {

**return objectMapper.readValue(new String(data), Employee.class);**

} **catch** (IOException e) {

**throw** **new** SerializationException(e);

}

}

}

**AutoRun class for Producer**

@Component

**public** **class** KafkaConsumer {

@Bean

**public** Consumer<Employee> consumer() {

**return** message -> System.***out***.println("Consumer received message as: " + message);

}

}

**Relevant Code for Consumer**

**application.properties for Consumer**

server.port=8081

**spring.cloud.stream.kafka.binder.brokers**=localhost:9092

**spring.cloud.stream.bindings.consumer-in-0.destination**=testTopic1

**spring.cloud.stream.bindings.consumer-in-0.consumer.use-native-decoding**=true

# Deserialize the message to Employee Object

**spring.cloud.stream.kafka.bindings.consumer-in-0.consumer.configuration.value.deserializer=**

com.ddlab.rnd.deserializer.MessageDeSerializer

**Consumer Code**

@Component

**public** **class** KafkaConsumer {

@Bean

**public** Consumer<Employee> consumer() {

**return** message -> System.***out***.println("Consumer received message as: " + message);

}

}