Apache Kafka in Spring Boot Application

## Introduction

In this tutorial we will guide you through the process of integrating Apache Kafka into a Spring Boot application.

We'll start with the basics of setting up Kafka and Spring Boot, followed by configuring Kafka producers and consumers within your Spring Boot project.

## Download and Setup Apache Kafka (on Windows)

To use Apache Kafka on your local machine, you first need to download and set it up. Go to [kafka.apache.org/quickstart](https://kafka.apache.org/quickstart) and click the “Download” button which will open a page with the suggested Apache Kafka version for your machine. A tar file will be downloaded, unzip it into a known path and rename it as simply “kafka”.

Once everything is set up, we can open two cmds, navigate to the located kafka folder and run the following commands:

1st cmd: .\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

* Zookeeper will run on port 2181.

2nd cmd: .\bin\windows\kafka-server-start.bat .\config\server.properties

* Kafka server will run on port 9092.

Keep both cmds running while using Kafka in your Spring Boot Application.

## Create a Spring Boot Application

Go to [start.spring.io](https://start.spring.io/) and create a new Spring Boot Application with your preferred settings. You have to add two dependencies to your project, namely Spring Web (for Rest APIs) and Spring for Apache Kafka (for Kafka integration). You can also add these two dependencies manually in your pom.xml file.

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.springframework.kafka</groupId>  
 <artifactId>spring-kafka</artifactId>  
</dependency>

## Configure Kafka Producer and Consumer

In the application.propeties file we will write the configuration for our Producer and Consumer. For the Producer we need to specify the Kafka broker server (opened on port 9092) where it needs to connect and the serialization method for both keys and values.

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

\*If you want to send other types of messages (like complex objects) you will need to use other types of serializers provided by spring like:

org.springframework.kafka.support.serializer.JsonSerializer

For the Consumer we will need to specify the kafka broker to connect to, the deserializers for messages, the group id for our Consumer (multiple Consumers can be part of a group), and the auto-offset-reset property which determines what the consumer should do when there is no initial offset in Kafka or if the current offset does not exist anymore on the server.

spring.kafka.consumer.bootstrap-servers=localhost:9092  
spring.kafka.consumer.group-id=myGroup  
spring.kafka.consumer.auto-offset-reset=earliest  
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer  
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer

\*The value “earliest” means that the consumer will start reading from the earliest message available in the topic.

## Create the Kafka Topic

The Kafka Broker we opened locally in our cmd will act as our Kafka Cluster. In the cluster we can create multiple topics which will host our messages. To create a Kafka Topic we are going to create a spring bean with its configuration.

import org.apache.kafka.clients.admin.NewTopic;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.kafka.config.TopicBuilder;  
  
@Configuration  
public class KafkaTopicConfig {  
  
 @Bean  
 public NewTopic tutorialTopic() {  
 return TopicBuilder.*name*("tutorial").build();  
 }  
}

## Create Kafka Producer

Now let’s create the Producer which will send the message to our topic to be consumed by the Consumer. We will create a service class which will use the spring provided Kafka Template to send our message.

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.kafka.core.KafkaTemplate;  
import org.springframework.stereotype.Service;  
  
@Service  
public class KafkaProducer {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(KafkaProducer.class);  
  
 @Autowired  
 private KafkaTemplate<String, String> kafkaTemplate;  
  
 public void sendMessage(String message) {  
 *LOGGER*.info(String.*format*("Message sent %s", message));  
 kafkaTemplate.send("tutorial" /\* our topic name \*/, message);  
 }  
}

\*Note that we are using slf4j logger to whether the message is being send or not

To make use of this service we will create a simple rest API to publish a message.

import com.soatutorial.springbootkafkatutorial.kafka.KafkaProducer;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RequestParam;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("/api/v1/kafka")  
public class MessageController {  
  
 @Autowired  
 private KafkaProducer kafkaProducer;  
  
 // http:localhost:8080/api/v1/kafka/publish?message=hello world  
 @GetMapping("/publish")  
 public ResponseEntity<String> publish(@RequestParam("message") String message) {  
 kafkaProducer.sendMessage(message);  
 return ResponseEntity.*ok*("Message sent to the topic");  
 }  
}

If we hit the URL commented above, we will send a “hello world” message to our Kafka Topic.

## Create Kafka Consumer

For our Consumer we will create another service which will hold a subscriber method to consume the messages from our topic. Spring Kafka library offers the KafkaListener annotation to subscribe a consumer group to a topic.

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.kafka.annotation.KafkaListener;  
import org.springframework.stereotype.Service;  
  
@Service  
public class KafkaConsumer {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(KafkaConsumer.class);  
  
 @KafkaListener(topics = "tutorial", groupId = "myGroup" /\* the group-id defined in application.properties \*/)  
 public void consume(String message) {  
 *LOGGER*.info(String.*format*("Message received -> %s", message));  
 }  
}

## Conclusion

In the end, we can simply run our app with the zookeeper and the kafka server opened, go to a web browser, and access our rest API.

A screen shot of a computer

Description automatically generated

If the message has been successfully published, we can see it logged into the console.



\*Every message found in the topic will be consumed by the Consumer (in the example above, I have sent the message “yes” to the same topic and it was printed once the app started).