**Spring-Kafka to develop Kafka-based messaging solutions**

Apache Kafka is a messaging solution that features:

* Publish and subscribe to streams of records.
* Store streams of records in a fault-tolerant durable way.
* Process streams of records as they occur.

Kafka allows an application to send messages, stores them on a central cluster, and allows these messages to be received by applications that process them. The Kafka cluster stores messages in categories called topics.

Spring-Kafka makes programming easier and quicker. It provides a "template" as a high-level abstraction for sending messages. It also provides support for Message-driven POJOs with @KafkaListener annotations.

This document will provide steps to integrate Spring Boot and Apache Kafka instance.

We assume that Apache Kafka is installed and running.

Here are the steps:

1. Create a Spring Boot Project.
2. Create a Kafka topic to publish messages.
3. Create custom data type to exchange messages.
4. Create a Producer to generate messages.
5. Create a Consumer to consume messages.
6. Create a Rest Controller to send messages.

**Create a Spring Boot Project**

Create a Spring Starter Project with web and kafka 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>

**Create a Kafka topic to publish messages**

Create a topic named ‘test’ in Kafka to publish messages.

**Create custom data type to exchange messages**

**package** com.example.domain;

**import** java.io.Serializable;

**public** **class** EventMessage **implements** Serializable{

/\*\*

\*

\*/

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** Long id;

String description;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getDescription() {

**return** description;

}

**public** **void** setDescription(String description) {

**this**.description = description;

}

**public** EventMessage(Long id, String description) {

**this**.id = id;

**this**.description = description;

}

**public** EventMessage() {

}

}

**Create a Producer to generate messages**

Create a configuration class for your Producer, and implement ProducerFactory. KafkaTemplate uses ProducerFactory to send messages.

**Truststore Client Authentication**: Configure the truststore, and keystore location, type, and password, and security protocol, and ssl enabled protocols, if the client authentication is required by the broker. Restrict access to the broker config file using file system permissions as it stores the password directly.

**ssl.truststore.password** is optional, but recommended. If a password is not set, access to the truststore is still available, but integrity checking is disabled.

Following two are optional settings: -

ssl.enabled.protocols

The list of protocols enabled for SSL connections.

Type: list

Default: TLSv1.2,TLSv1.1,TLSv1

Importance: medium

ssl.truststore.type

The file format of the truststore file.

Type: string

Default: JKS

Importance: medium

**@Configuration**

**@EnableKafka**

**public class ProducerConfiguration** {

@Value("${bootstrap.servers}")

private String bootstrapServers;

@Value("${security.protocol}")

private String securityProtocol;

@Value("${ssl.truststore.location}")

private String sslTruststoreLocation;

@Value("${ssl.truststore.password}")

private String sslTruststorePassword;

@Value("${ssl.enabled.protocols}")

private String sslEnabledProtocols;

@Value("${ssl.truststore.type}")

private String sslTruststoreType;

@Value("${ssl.keystore.type}")

private String sslKeystoreType;

@Value("${ssl.keystore.location}")

private String sslKeystoreLocation;

@Value("${ssl.keystore.password}")

private String sslKeyStorePassword;

@Bean

public Map<String, Object> producerConfigs() {

Map<String, Object> props = new HashMap<>();

props.put(ProducerConfig.BOOTSTRAP\_SERVERS\_CONFIG, bootstrapServers);

props.put(ProducerConfig.CLIENT\_ID\_CONFIG,Constants.CLIENT\_ID);

props.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG, JsonSerializer.class);

if(bootstrapServers.toLowerCase().startsWith(Constants.LOCALHOST))

props.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG, LongSerializer.class);

else{

props.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG,

org.apache.kafka.common.serialization.ByteArraySerializer.class.getName());

props.put(Constants.SECURITY\_PROTOCOL, securityProtocol);

props.put(Constants.SSL\_TRUSTSORE\_LOCATION, sslTruststoreLocation);

props.put(Constants.SSL\_TRUSTSORE\_PASSWORD, sslTruststorePassword);

props.put(Constants.SSL\_ENABLED\_PROTOCOLS, sslEnabledProtocols);

props.put(Constants.SSL\_TRUSTSTORE\_TYPE, sslTruststoreType);

props.put(Constants.SSL\_KEYSTORE\_TYPE, sslKeystoreType);

props.put(Constants.SSL\_KEYSTORE\_LOCATION, sslKeystoreLocation);

props.put(Constants.SSL\_KEYSTORE\_PASSWORD, sslKeyStorePassword);

}

return props;

}

@Bean

public **ProducerFactory<Long, EventMessage>** producerFactory() {

return new **DefaultKafkaProducerFactory<>(producerConfigs())**;

}

@Bean

public **KafkaTemplate**<Long, EventMessage> kafkaTemplate() {

return new **KafkaTemplate<>(producerFactory());**

}

}

**@Service**

**public class EventProducer** {

@Autowired

**private** KafkaTemplate<Long, EventMessage> kafkaTemplate;

**private** **final** Logger logger = LoggerFactory.*getLogger*(EventProducer.**class**);

**public** **void** sendEventMessage(String topic, String evtMsg){

EventMessage msg = **new** EventMessage(**new** Long((**new** Random()).nextInt(9)),evtMsg);

Message<EventMessage> message = MessageBuilder

.*withPayload*(msg)

.setHeader(KafkaHeaders.***TOPIC***, topic)

.build();

logger.info("Producing message: " + msg.getDescription());

kafkaTemplate.send(message);

}

}

Configure application with **application.properties**: -

logging.level.root=ERROR

logging.level.org.springframework.web=ERROR

logging.level.com.example=INFO

spring.main.banner-mode=off

server.port=9000

app.consumer.topic.subscribed-to=test

spring.kafka.consumer.group-id=group\_id

# bootstrap.server

bootstrap.servers=localhost:9092

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

security.protocol=SSL

ssl.truststore.location=/opt/kafkaprivate/security/pki/kafkaca-certs.jks

ssl.truststore.password=xxxxxxx

ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1

ssl.truststore.type=JKS

ssl.keystore.type=JKS

ssl.keystore.location=/opt/kafkaprivate/security/pki/kafkaserver.jks

ssl.keystore.password=xxxxxxx

ssl.key.password=xxxxxxx

**Create a Consumer to consume messages**

To enable the kafka consumer, consumerFactory() and kafkaListenerContainerFactory() methods should be implemented in the ConsumerConfiguration class.

**@Configuration**

**@EnableKafka**

**public class ConsumerConfiguration** {

@Value("${bootstrap.servers}")

private String bootstrapServers;

@Value("${spring.kafka.consumer.auto-offset-reset}")

private String autoOffsetReset;

@Value("${security.protocol}")

private String securityProtocol;

@Value("${ssl.truststore.location}")

private String sslTruststoreLocation;

@Value("${ssl.truststore.password}")

private String sslTruststorePassword;

@Value("${ssl.enabled.protocols}")

private String sslEnabledProtocols;

@Value("${ssl.truststore.type}")

private String sslTruststoreType;

@Value("${ssl.keystore.type}")

private String sslKeystoreType;

@Value("${ssl.keystore.location}")

private String sslKeystoreLocation;

@Value("${ssl.keystore.password}")

private String sslKeyStorePassword;

public Map<String, Object> consumerConfigs() {

Map<String, Object> props = new HashMap<>();

props.put(ConsumerConfig.BOOTSTRAP\_SERVERS\_CONFIG, bootstrapServers);

props.put(ConsumerConfig.AUTO\_OFFSET\_RESET\_CONFIG, autoOffsetReset);

props.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, LongDeserializer.class);

props.put(ConsumerConfig.VALUE\_DESERIALIZER\_CLASS\_CONFIG, JsonDeserializer.class);

if(bootstrapServers.toLowerCase().startsWith(Constants.LOCALHOST))

props.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG, LongDeserializer.class);

else{

props.put(ConsumerConfig.KEY\_DESERIALIZER\_CLASS\_CONFIG,

org.apache.kafka.common.serialization.ByteArrayDeserializer.class.getName());

props.put(Constants.SECURITY\_PROTOCOL, securityProtocol);

props.put(Constants.SSL\_TRUSTSORE\_LOCATION, sslTruststoreLocation);

props.put(Constants.SSL\_TRUSTSORE\_PASSWORD, sslTruststorePassword);

props.put(Constants.SSL\_ENABLED\_PROTOCOLS, sslEnabledProtocols);

props.put(Constants.SSL\_TRUSTSTORE\_TYPE, sslTruststoreType);

props.put(Constants.SSL\_KEYSTORE\_TYPE, sslKeystoreType);

props.put(Constants.SSL\_KEYSTORE\_LOCATION, sslKeystoreLocation);

props.put(Constants.SSL\_KEYSTORE\_PASSWORD, sslKeyStorePassword);

}

return props;

}

@Bean

public **ConsumerFactory<Long, EventMessage>** consumerFactory() {

return new DefaultKafkaConsumerFactory<>(

consumerConfigs(),

new LongDeserializer(),

new JsonDeserializer<>(EventMessage.class));

}

@Bean

public **ConcurrentKafkaListenerContainerFactory<Long, EventMessage>** kafkaListenerContainerFactory() {

ConcurrentKafkaListenerContainerFactory<Long, EventMessage> factory =

new ConcurrentKafkaListenerContainerFactory<>();

factory.setConsumerFactory(consumerFactory());

return factory;

}

}

**@Service**

**public class Consumer**{

private static final Logger logger = LoggerFactory.getLogger(Consumer.class);

**@KafkaListener(topics = "${app.consumer.topic.subscribed-to}", groupId = "${spring.kafka.consumer.group-id}")**

public void consume(EventMessage message,

@Headers MessageHeaders headers) {

logger.info("Consumed message: " + message.getDescription());

}

}

**Create a Rest Controller to send messages**

Create a rest resource that enables sending messages to the Kafka topic.

package com.example.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.example.event.producer.EventProducer;

@RestController

@RequestMapping(value = "/kafka")

public class EventMessageController {

@Value("${app.consumer.topic.subscribed-to}")

private String topicToPublish;

private final EventProducer eventProducer;

@Autowired

EventMessageController(EventProducer eventProducer) {

this.eventProducer = eventProducer;

}

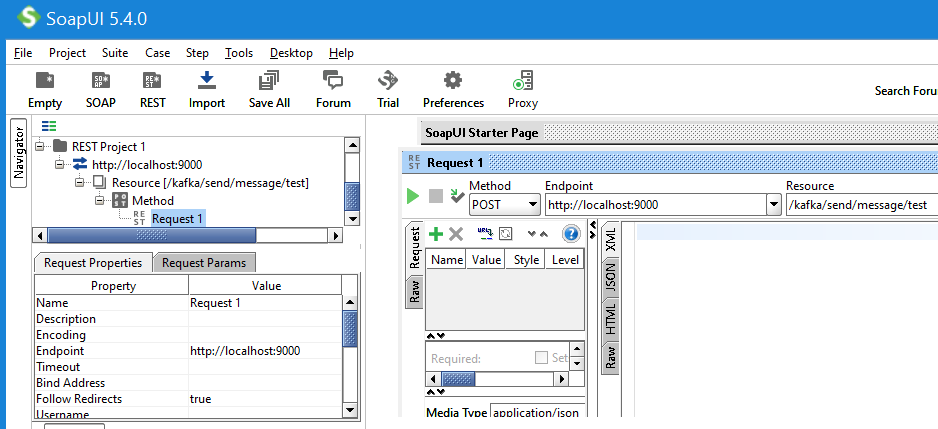
@PostMapping(path = "/send/message/{event}")

public void sendEventMessage(@PathVariable String event) {

this.eventProducer.sendEventMessage(topicToPublish, event);

}

}

Now, if we run our Spring Boot application, we can invoke rest resource http://localhost:9000/ kafka/send/message/{event} using the Post method. 

This will trigger our Producer to publish the message to the Kafka topic.



Our Consumer will receive the message and it will be logged in the application console.

