Spring Kafka Consumer Producer Example

10 minute read

In this post, you're going to learn how to create a Spring Kafka (https://spring.io/projects/spring-kafka) *Hello World* example that uses Spring Boot (https://spring.io/projects/spring-boot) and Maven (https://maven.apache.org/).

(Step-by-step)

So if you're a Spring Kafka beginner, **you'll love this guide**.

Let's get started.

> If you want to learn more about Spring Kafka - head on over to the Spring Kafka tutorials page (https://codenotfound.com/spring-kafka-tutorials).

1. What is Spring Kafka?

The Spring for Apache Kafka (spring-kafka) project **applies core Spring concepts** to the development of Kafka-based messaging solutions.

It provides a *'template'* as a high-level abstraction for sending messages. It also contains support for Message-driven POJOs with `@KafkaListener` annotations and a listener container.

Spring Kafka is a Spring main project (https://spring.io/projects). It is developed and maintained by Pivotal Software (https://pivotal.io/).

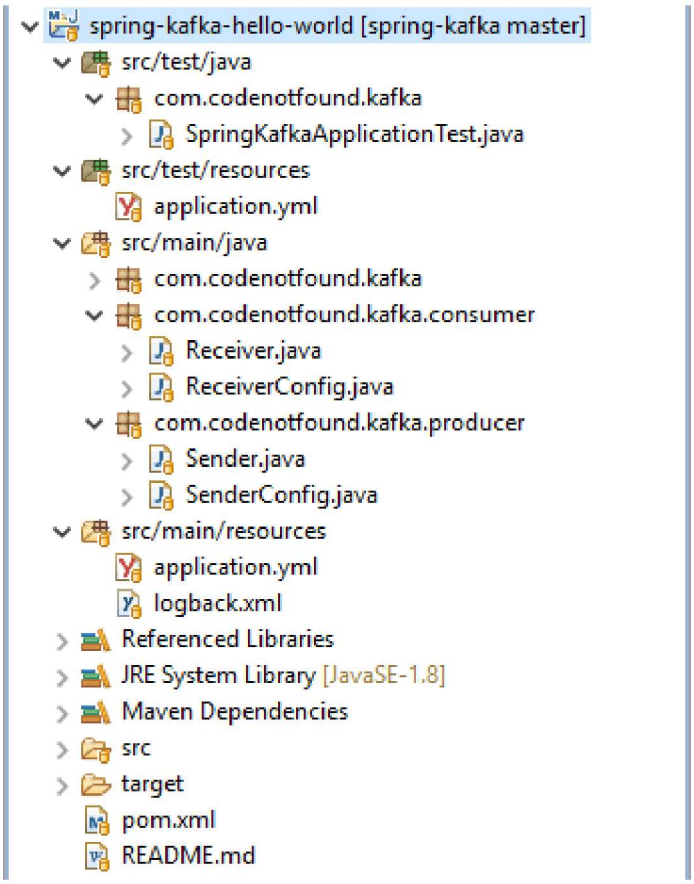
To show how Spring Kafka works let's create a simple Hello World example. We will build a sender to produce the message and a receiver to consume the message.

2. General Project Overview

Tools used:

- Spring Kafka 2.2
- Spring Boot 2.1
- Maven 3.5

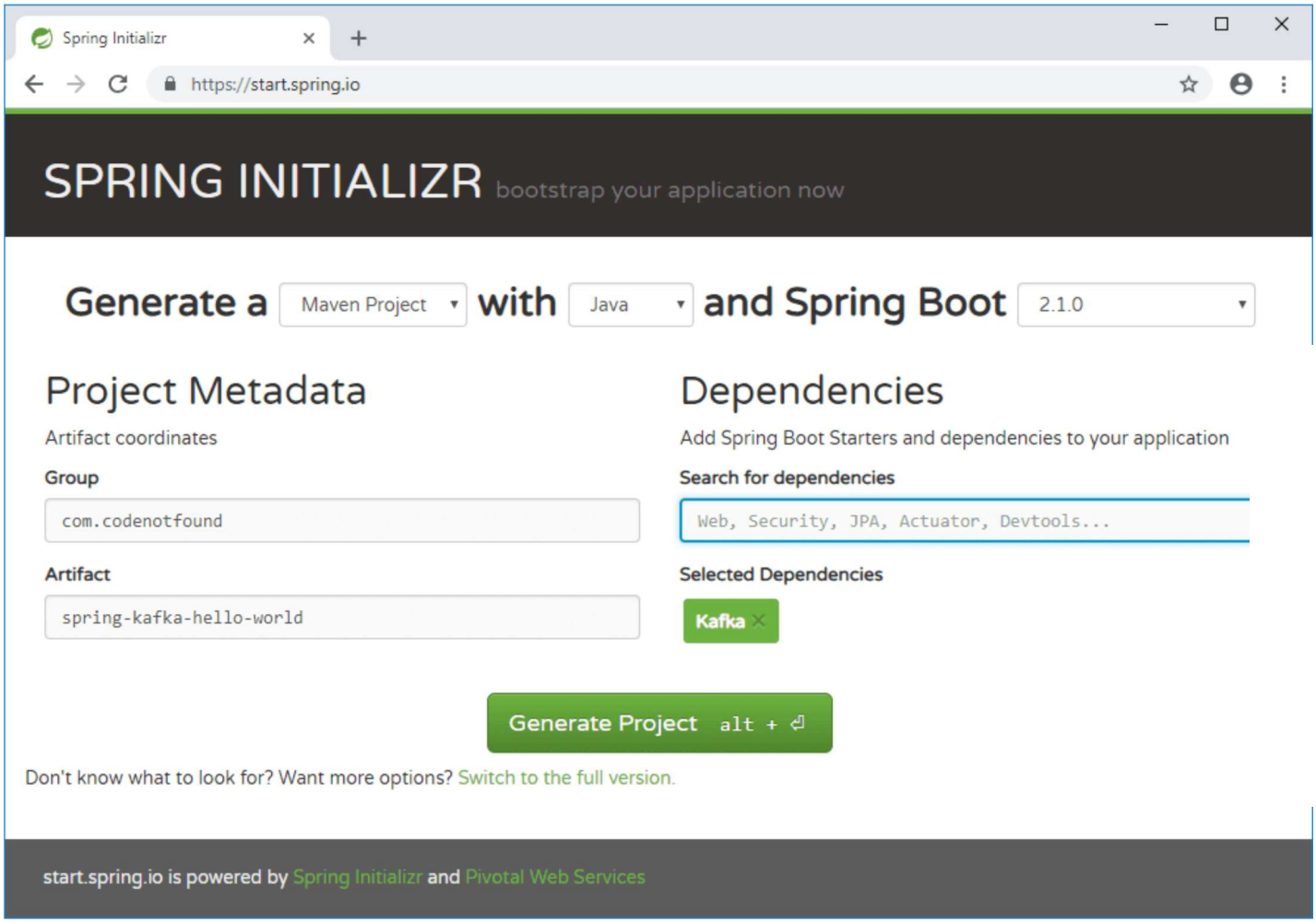
Our project has the following directory structure:



3. Maven Setup

We build and run our example using **Maven**. If not already done, download and install Apache Maven (https://downlinko.com/download-install-apache-maven-windows.html).

Let's use Spring Initializr (https://start.spring.io/) to generate our Maven project. Make sure to select `Kafka` as a dependency.



Click `Generate Project` to generate and download the Spring Boot project template. At the root of the project, you'll find a `pom.xml` file which is the XML representation of the Maven project.

To avoid having to manage the version compatibility of the different Spring dependencies, we will inherit the defaults from the `spring-boot-starter-parent` parent POM.

The generated project contains Spring Boot Starters (https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot-starters) that manage the different Spring dependencies.

The `spring-boot-starter` dependency is the core starter, it includes auto-configuration, logging, and YAML support. The `spring-boot-starter-test` includes the dependencies for testing Spring Boot applications with libraries that include JUnit (http://junit.org/junit4/), Hamcrest (http://hamcrest.org/JavaHamcrest/) and Mockito (http://site.mockito.org/).

A dependency on `spring-kafka` is added. We also include `spring-kafka-test` to have access to an embedded Kafka broker when running our unit test.

> *Note that the **version of Spring Kafka is linked to the version of the Apache Kafka client** that is used. You need to align the version of Spring Kafka to the version of the Kafka broker you connect to. For more information consult the complete Kafka client compatibility list (https://spring.io/projects/spring-kafka#kafka-client-compatibility).*

In the plugins section, you'll find the Spring Boot Maven Plugin (https://docs.spring.io/spring-boot/docs/2.1.0.RELEASE/reference/html/build-tool-plugins-maven-plugin.html): `spring-boot-maven-plugin`. It allows us to build a single, runnable "uber-jar". This is a convenient way to execute and transport code. Also, the plugin allows you to start the example via a Maven command.

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.codenotfound</groupId>
  <artifactId>spring-kafka-hello-world</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <name>spring-kafka-hello-world</name>
  <description>Spring Kafka - Consumer Producer Example</description>
  <url>https://www.codenotfound.com/spring-kafka-consumer-producer-example.html</url>

  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.1.0.RELEASE</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
    <java.version>1.8</java.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter</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>
    <dependency>
      <groupId>org.springframework.kafka</groupId>
      <artifactId>spring-kafka-test</artifactId>
      <scope>test</scope>
```

```
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>

</project>
```

4. Spring Boot Setup

We use Spring Boot so that we have a Spring Kafka application that you can "just run". Start by creating a `SpringKafkaApplication` class. It contains the `main()` method that uses Spring Boot's `SpringApplication.run()` to launch the application.

> *Note that `@SpringBootApplication` is a convenience annotation that adds: `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan`.*

For more information on Spring Boot, check the Spring Boot getting started guide (https://spring.io/guides/gs/spring-boot/).

```
package com.codenotfound.kafka;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringKafkaApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringKafkaApplication.class, args);
  }
}
```

> *The below sections will detail how to create a sender and receiver together with their respective configurations. It is also possible to have Spring Boot autoconfigure Spring Kafka (https://codenotfound.com/spring-kafka-boot-example.html) using default values so that actual code that needs to be written is reduced to a bare minimum.*

> *This example will send/receive a simple `String`. If you would like to send more complex objects you could, for example, use an Avro Kafka serializer (https://codenotfound.com/spring-kafka-apache-avro-serializer-deserializer-example.html) or the Kafka Jsonserializer (https://codenotfound.com/spring-kafka-json-serializer-deserializer-example.html) that ships with Spring Kafka.*

We also create an `application.yml` YAML (http://yaml.org/) properties file under `src/main/resources`. Properties from this file will be injected by Spring Boot (https://docs.spring.io/spring-boot/docs/2.1.0.RELEASE/reference/html/boot-features-external-config.html#boot-features-external-config) into our configuration beans using the `@Value` annotation.

```
kafka:
  bootstrap-servers: localhost:9092
```

5. Create a Spring Kafka Message Producer

For sending messages we will be using the `KafkaTemplate` which wraps a `Producer` and provides convenience methods (https://docs.spring.io/spring-kafka/docs/2.2.0.RELEASE/reference/html/_reference.html#kafka-template) to send data to Kafka topics. The template provides asynchronous send methods which return a `ListenableFuture`.

In the `Sender` class, the `KafkaTemplate` is auto-wired as the creation will be done further below in a separate `SenderConfig` class.

For this example, we will use the `send()` method that takes as input a `String` payload that needs to be sent.

> *Note that the Kafka broker default settings cause it to auto-create a topic when a request for an unknown topic is received.*

```
package com.codenotfound.kafka.producer;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.kafka.core.KafkaTemplate;

public class Sender {

  private static final Logger LOGGER =
      LoggerFactory.getLogger(Sender.class);

  @Autowired
  private KafkaTemplate<String, String> kafkaTemplate;

  public void send(String payload) {
    LOGGER.info("sending payload='{}'", payload);
    kafkaTemplate.send("helloworld.t", payload);
  }
}
```

The creation of the `KafkaTemplate` and `Sender` is handled in the `SenderConfig` class. The class is annotated with `@Configuration` which indicates (https://docs.spring.io/spring/docs/5.1.2.RELEASE/spring-framework-reference/core.html#beans-java-basic-concepts) that the class can be used by the Spring IoC container as a source of bean definitions.

In order to be able to use the Spring Kafka template, we need to configure a `ProducerFactory` and provide it in the template's constructor.

The producer factory needs to be set with some mandatory properties amongst which the `'BOOTSTRAP_SERVERS_CONFIG'` property that specifies a list of *host:port* pairs used for establishing the initial connections to the Kafka cluster. Note that this value is configurable as it is fetched from the `application.yml` configuration file.

A message in Kafka is a key-value pair with a small amount of associated metadata. As Kafka stores and transports `Byte` arrays, we need to specify the format from which the key and value will be serialized. In this example we are sending a `String` as payload, as such we specify the `StringSerializer` class which will take care of the needed transformation.

For a complete list of the other configuration parameters, you can consult the Kafka ProducerConfig API (https://kafka.apache.org/20/javadoc/org/apache/kafka/clients/producer/ProducerConfig.html).

```
package com.codenotfound.kafka.producer;

import java.util.HashMap;
import java.util.Map;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.common.serialization.StringSerializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.DefaultKafkaProducerFactory;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;

@Configuration
public class SenderConfig {

  @Value("${kafka.bootstrap-servers}")
  private String bootstrapServers;

  @Bean
  public Map<String, Object> producerConfigs() {
    Map<String, Object> props = new HashMap<>();
    // list of host:port pairs used for establishing the initial connections to the Kakfa cluster
    props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
        bootstrapServers);
    props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
        StringSerializer.class);
    props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
        StringSerializer.class);

    return props;
  }

  @Bean
  public ProducerFactory<String, String> producerFactory() {
    return new DefaultKafkaProducerFactory<>(producerConfigs());
  }

  @Bean
  public KafkaTemplate<String, String> kafkaTemplate() {
    return new KafkaTemplate<>(producerFactory());
  }

  @Bean
  public Sender sender() {
```

```
        return new Sender();
    }
}
```

6. Create a Spring Kafka Message Consumer

Like with any messaging-based application, you need to create a receiver that will handle the published messages. The `Receiver` is nothing more than a simple POJO that defines a method for receiving messages. In the below example we named the method `receive()`, but you can name it anything you like.

The `@KafkaListener` annotation creates a `ConcurrentMessageListenerContainer` message listener container behind the scenes for each annotated method. To do so, a factory bean with name `kafkaListenerContainerFactory` is expected that we will configure in the next section.

Using the `topics` element, we specify the topics for this listener.

For more information on the other available elements on the `KafkaListener`, you can consult the documentation (https://docs.spring.io/spring-kafka/api/org/springframework/kafka/annotation/KafkaListener.html).

> *For testing convenience, we added a `CountDownLatch`. This allows the POJO to signal that a message is received. This is something you are not likely to implement in a production application.*

```
package com.codenotfound.kafka.consumer;

import java.util.concurrent.CountDownLatch;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.annotation.KafkaListener;

public class Receiver {

  private static final Logger LOGGER =
      LoggerFactory.getLogger(Receiver.class);

  private CountDownLatch latch = new CountDownLatch(1);

  public CountDownLatch getLatch() {
    return latch;
  }

  @KafkaListener(topics = "helloworld.t")
  public void receive(String payload) {
    LOGGER.info("received payload='{}'", payload);
    latch.countDown();
  }
}
```

The creation and configuration of the different Spring Beans needed for the `Receiver` POJO are grouped in the `ReceiverConfig` class. Similar to the `SenderConfig` it is annotated with `@Configuration`.

> *Note the `@EnableKafka` annotation which enables the detection of the `@KafkaListener` annotation that was used on the previous `Receiver` class.*

The `kafkaListenerContainerFactory()` is used by the `@KafkaListener` annotation from the `Receiver` in order to configure a `MessageListenerContainer`. To create it, a `ConsumerFactory` and accompanying configuration `Map` is needed.

In this example, a number of mandatory properties are set amongst which the initial connection and deserializer parameters.

We also specify a `'GROUP_ID_CONFIG'` which allows to identify the group (https://stackoverflow.com/a/41377616/4201470) this consumer belongs to. Messages will be load balanced over consumer instances that have the same group id.

On top of that, we also set `'AUTO_OFFSET_RESET_CONFIG'` to `"earliest"`. This ensures that our consumer reads from the beginning of the topic even if some messages were already sent before it was able to startup.

For a complete list of the other configuration parameters, you can consult the Kafka ConsumerConfig API (https://kafka.apache.org/20/javadoc/index.html?org/apache/kafka/clients/consumer/ConsumerConfig.html).

```
package com.codenotfound.kafka.consumer;

import java.util.HashMap;
import java.util.Map;
import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.annotation.EnableKafka;
import org.springframework.kafka.config.ConcurrentKafkaListenerContainerFactory;
import org.springframework.kafka.config.KafkaListenerContainerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.DefaultKafkaConsumerFactory;
import org.springframework.kafka.listener.ConcurrentMessageListenerContainer;

@Configuration
@EnableKafka
public class ReceiverConfig {

  @Value("${kafka.bootstrap-servers}")
  private String bootstrapServers;

  @Bean
  public Map<String, Object> consumerConfigs() {
    Map<String, Object> props = new HashMap<>();
    // list of host:port pairs used for establishing the initial connections to the Kafka cluster
    props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
        bootstrapServers);
    props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
        StringDeserializer.class);
    props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
        StringDeserializer.class);
    // allows a pool of processes to divide the work of consuming and processing records
    props.put(ConsumerConfig.GROUP_ID_CONFIG, "helloworld");
    // automatically reset the offset to the earliest offset
    props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

    return props;
  }

  @Bean
  public ConsumerFactory<String, String> consumerFactory() {
    return new DefaultKafkaConsumerFactory<>(consumerConfigs());
  }
```

```
  @Bean
  public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String, String>>
kafkaListenerContainerFactory() {
    ConcurrentKafkaListenerContainerFactory<String, String> factory =
        new ConcurrentKafkaListenerContainerFactory<>();
    factory.setConsumerFactory(consumerFactory());

    return factory;
  }

  @Bean
  public Receiver receiver() {
    return new Receiver();
  }
}
```

7. Testing the Spring Kafka Template & Listener

A basic `SpringKafkaApplicationTest` is provided to verify that we are able to send and receive a message to and from Apache Kafka. It contains a `testReceiver()` unit test case that uses the `Se` bean to send a message to the `'helloworld.t'` topic on the Kafka bus.

We then check if the `CountDownLatch` from the `Receiver` was lowered from 1 to 0 as this indicate[illegible] message was processed by the `receive()` method.

An embedded Kafka broker is started by using the `@EmbeddedKafka` annotation.

As the embedded server is started on a random port, we provide a dedicated `src/test/resources/apppication.yml` properties file for testing which uses the `spring.embedded.kafka.brokers` system property to set the correct address of the broker(s).

```
kafka:
  bootstrap-servers: ${spring.embedded.kafka.brokers}
```

> *Below test case can also be executed after you install Kafka and Zookeeper (https://codenotfound.com/apache-kafka-download-installation.html) on your local system. Just comment out `@EmbeddedKafka` and change the `'bootstrap-servers'` property of the application properties file located in `src/test/resources` to the address of the local broker.*

```
package com.codenotfound.kafka;

import static org.assertj.core.api.Assertions.assertThat;
import java.util.concurrent.TimeUnit;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.kafka.test.context.EmbeddedKafka;
import org.springframework.test.annotation.DirtiesContext;
import org.springframework.test.context.junit4.SpringRunner;
import com.codenotfound.kafka.consumer.Receiver;
import com.codenotfound.kafka.producer.Sender;

@RunWith(SpringRunner.class)
@SpringBootTest
@DirtiesContext
@EmbeddedKafka(partitions = 1,
    topics = {SpringKafkaApplicationTest.HELLOWORLD_TOPIC})
public class SpringKafkaApplicationTest {

  static final String HELLOWORLD_TOPIC = "helloworld.t";

  @Autowired
  private Receiver receiver;

  @Autowired
  private Sender sender;

  @Test
  public void testReceive() throws Exception {
    sender.send("Hello Spring Kafka!");

    receiver.getLatch().await(10000, TimeUnit.MILLISECONDS);
    assertThat(receiver.getLatch().getCount()).isEqualTo(0);
  }
}
```

In order to run above test case, open a command prompt in the project root folder and execute following Maven command:

```
mvn test
```

The result is a successful build during which a Hello World message is sent and received using Kafka.

```
.   ____          _            __ _ _
/\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
\\/  ___)| |_)| | | | | || (_| |  ) ) ) )
'  |____| .__|_| |_|_| |_\__, | / / / /
=========|_|==============|___/=/_/_/_/
:: Spring Boot ::        (v2.1.0.RELEASE)

06:34:04.959 [main] WARN  k.server.BrokerMetadataCheckpoint - No meta.properties file under dir
C:\Users\CODENO~1\AppData\Local\Temp\kafka-7816218183283567156\meta.properties
06:34:05.521 [main] WARN  k.server.BrokerMetadataCheckpoint - No meta.properties file under dir
C:\Users\CODENO~1\AppData\Local\Temp\kafka-7816218183283567156\meta.properties
06:34:05.959 [main] INFO  c.c.kafka.SpringKafkaApplicationTest - Starting SpringKafkaApplicationTest
DESKTOP-2RB3C1U with PID 14968 (started by Codenotfound in C:\Users\Codenotfound\repos\spring-
kafka\spring-kafka-hello-world)
06:34:05.959 [main] INFO  c.c.kafka.SpringKafkaApplicationTest - No active profile set, falling back
default profiles: default
06:34:06.943 [main] INFO  c.c.kafka.SpringKafkaApplicationTest - Started SpringKafkaApplicationTest i
3.375 seconds (JVM running for 4.482)
06:34:07.193 [main] INFO  c.codenotfound.kafka.producer.Sender - sending payload='Hello Spring Kafka!
06:34:07.349 [org.springframework.kafka.KafkaListenerEndpointContainer#0-0-C-1] INFO
c.c.kafka.consumer.Receiver - received payload='Hello Spring Kafka!'
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 9.406 s - in
com.codenotfound.kafka.SpringKafkaApplicationTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.659 s
[INFO] Finished at: 2018-11-24T06:34:12+01:00
[INFO] ------------------------------------------------------------------------
```

If you would like to run the above code sample you can get the full source code here (https://github.com/code-not-found/spring-kafka/tree/master/spring-kafka-hello-world).

In this getting started tutorial you learned how to create a Spring Kafka template and Spring Kafka listener to send/receive messages.

If you found this sample useful or have a question you would like to ask, drop a line below!

Tags: Apache Kafka | Consumer | Example | Hello World | Maven | Producer | Spring Boot | Spring Kafka | Spring | Tutorial

Categories: Spring Kafka

Updated: November 24, 2018

LEAVE A COMMENT

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

sarath • 7 months ago

Getting error while running above application

org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'org.springframework.cloud.netflix.eureka.EurekaClientAutoConfiguration': Unsatisfied dependency expressed through constructor parameter 0; nested exception is org.springframework.beans.factory.NoSuchBeanDefinitionException: No qualifying bean of ty 'org.springframework.core.env.ConfigurableEnvironment' available: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {}

Reply • Share ›

Sreelakshmi • 8 months ago

Hi,
Is there a way to receive acknowledgement from Kafka after producing the message, like call using kafka teamplate

Reply • Share ›

codenotfound Mod ➚ Sreelakshmi • 8 months ago • edited

Hi,

You can configure the *KafkaTemplate* with a *ProducerListener* to get an async callback with the results of the send (success or failure).

If you wish to block the sending thread, to await the result, you can invoke the future's *get()* method.

Some examples: https://docs.spring.io/spri...

Reply • Share ›

anis hakim • a year ago

hi everyone can you guys tell me how can i receive message when i connect late (it means the consumer connect after the message was send from the producer)

Reply • Share ›

codenotfound Mod ➚ anis hakim • 10 months ago

Hi,

You can use *AUTO_OFFSET_RESET_CONFIG* and set it to *earliest*.

This flag tells Kafka where to start reading offsets in case you do not have any 'commit' yet. In others words, it will start from the beginning even if you connect late.

For more info on offsets check the Kafka documentation: https://kafka.apache.org/do...

Reply • Share ›

tuan ➚ anis hakim • a year ago

Same confusing LOL.

Reply • Share ›

Ramesh Jangala • a year ago

I am new to Kafka and tried with provided receiver code snippets and mentioned all the prope as you mentioned. When running unable to read/consume messages from kafkalistner annot method. seeing the warning of "Connection to node -1 could not be established. Broker may be available."

Reply • Share ›

codenotfound Mod ➚ Ramesh Jangala • 10 months ago

Hi,

There can be a number of reasons why this occurs. First thing to look at would be if th version of Spring Kafka matches the version of the Kafka broker you are connection t https://spring.io/projects/...

Reply • Share ›

Marina N • a year ago

Hi, great tutorial.
I wonder If there is a way to implement prioritised consuming of let's say, three different topics. I can not see how it can be done with @KafkaListener annotated method, perhaps there is way with KafkaConsumer's 'poll' method override or some other idea ?
Tnx in advance,
Marina

Reply • Share ›

codenotfound Mod ➚ Marina N • a year ago

Hi Marina,

Could you give a bit more context on the specific problem you are trying to solve? Thanks.

Reply • Share ›

Marina N ➚ codenotfound • a year ago

Hi, thanks for the quick response.

I have three topics, HighPriority, MedPriority, LowPriority. I need to configure the consumer to always consume first from HighPriority messages, and only if there are none such, to proceed with medium and low priority

are none such, to proceed with medium and low priority.

Right now, I have @KafkaListener method on the three topics, and it works fine consuming records from all three non prioritized.

I have a vague idea of implementing this (priority setting) with pausing the consumer on lower priority topics if there are new messages in HighPriority topic (if (HighPriority.endOffsets > HighPriority.CommitedOffsets) and resuming it later but:

1. I am not sure if this is the right approach for the problem;

2. Right now can't configure @KafkaListener with KafkaConsumer as instructed here:
https://github.com/spring-p...
(I would need this to access the offsets of the topics/paritions)

This is possibly vague question, but I would any advice for how to proceed or s reference to code example solving similar issue.

Reply • Share ›

codenotfound Mod → Marina N • a year ago

I think you are on the right track by pausing the consumer on the lower priority topics.

What version of Apache Kafka are you connecting to?

Reply • Share ›

Marina N → codenotfound • a year ago

1.1.0

Reply • Share ›

Ravi Kumar Reddy Lekkala • a year ago • edited

Hi, I got below exception while running this.
Appreciate your help?

Error creating bean with name 'kafkaSender': Unsatisfied dependency expressed through field 'kafkaTemplate'
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name 'kafkaSender': Unsatisfied dependency expressed through field 'kafkaTemplate';
nested exception is org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'kafkaTemplate' defined in class path
resource [org/springframework/boot/autoconfigure/kafka/KafkaAutoConfiguration.class]: Bean instantiation via factory method failed;
nested exception is org.springframework.beans.BeanInstantiationException: Failed to instantiate [org.springframework.kafka.core.KafkaTemplate]:
Factory method 'kafkaTemplate' threw exception; nested exception is
java.lang.NoClassDefFoundError: org/apache/kafka/common/header/Headers

Caused by: org.springframework.beans.factory.BeanCreationException: Error creating bean with

name 'kafkaTemplate' defined in class path resource

see more

• Reply • Share ›

codenotfound Mod ➚ Ravi Kumar Reddy Lekkala • a year ago

That's strange what version of Kafka are you trying to connect to?

• Reply • Share ›

Saurav Sharma • 2 years ago

Hi, Thanks for great tutorial. Could you please share how to achieve Exactly once Kafka Consumer in here?

• Reply • Share ›

codenotfound Mod ➚ Saurav Sharma • a year ago

Hi,

You would need to use the Kafka Streams API that is available since version 0.11 and "processing.guarantee=exactly_once".

If I find some time I might add an example.

• Reply • Share ›

Djeison Selzlein • 2 years ago

Hi there, thank you for the great tutorial!

I just have a question, what if I need an @Autowired service in my Receiver class, would you recommend to annotate this Receiver with @Component and remove the @Bean declaration from the configuration class?
Thank you once again!

• Reply • Share ›

codenotfound Mod ➚ Djeison Selzlein • 2 years ago • edited

Hi,

In the end both approaches would work. The Sender class is defined using *@Bean* and also has an *@Autowired* towards the *KafkaTemplate*.

If you want a bit more detail on the difference between the two approaches I suggest you checkout following thread: https://stackoverflow.com/q...

1 • Reply • Share ›

Avatar This comment was deleted.

codenotfound Mod ➚ Guest • 2 years ago

Hi,

Thanks for your comment. There are two *application.yml* files in the project. One under *src/main/resources* and one under *src/test/resources*. The latter does contain *${spring.embedded.kafka.brokers}*.

Is this what you mean?

Reply • Share ›

ganesh kumar • 2 years ago

Thanks for this post!

This code runs good in my local workspace but the issue is shutting down time. Zookeeper or Kafka server is getting shut down by 1 minute. Could you please explain why?

2017-12-20 21:06:17.183 WARN 5848 --- [ry:/127.0.0.1:0] o.apache.zookeeper.server.NIOServerCnxn : caught end of stream exception

org.apache.zookeeper.server.ServerCnxn$EndOfStreamException: Unable to read additiona data from client sessionid 0x160749068230000, likely client has closed socket
at org.apache.zookeeper.server.NIOServerCnxn.doIO(NIOServerCnxn.java:228) ~[zookeepe 3.4.9.jar:3.4.9-1757313]
at org.apache.zookeeper.server...(NIOServerCnxnFactory.java:203) [zookeeper-3.4.9.jar:3.4. 1757313]
at java.lang.Thread.run(Thread.java:744) [na:1.7.0_45]

Reply • Share ›

codenotfound Mod ganesh kumar • 2 years ago

Hi,

How are you starting ZooKeeper and Kafka?

Reply • Share ›

ganesh kumar codenotfound • 2 years ago

Zookeeper in local runs normally but kafka embedded is stopping the server.

Reply • Share ›

Will Bowditch • 2 years ago

Hi,
when I set the embeddedKafka ClassRule, I get a NoClassDefFoundError: org/apache/kafka/common/security/auth/SecurityProtocol

There appears to be different paths for SecurityProtocol,
org.apache.kafka.common.protocol.SecurityProtocol
and
org.apache.common.security.auth.SecurityProtocol.

Any idea for a solution to this?

Reply • Share ›

codenotfound Mod ➚ Will Bowditch • 2 years ago

Hi,

What version of Spring Kafka are you using?

^ | v • Reply • Share ›

rafeeq • 2 years ago

It will be useful if you can add some integration test

^ | v • Reply • Share ›

codenotfound Mod ➚ rafeeq • 2 years ago

Hi,

Could you specify what you mean by this? If you comment out the *ClassRule* then you have a test case that integrates with a standalone Kafka installation.

Or did you have something else in mind?

^ | v • Reply • Share ›

rafeeq ➚ codenotfound • 2 years ago

What i wanted is in the producer test, you are putting a message and consumi immediately to check the message, in consumer, you are sending a message a checking for the Latch count to be 0 , in a integration test, if you send message the producer, it needs to verify that the message was sent, and in the consumer we need to confirm that the same message was received. It has to be in a integrated manner, not like a unit test

^ | v • Reply • Share ›

Probal Sikder • 2 years ago

In my case, test fails saying :

org.junit.ComparisonFailure: expected:<[2]> but was:<[0]>
....
.runBeforeTestMethod:46 expected:<[2]> but was:<[0]>

Any idea what am I missing here?

^ | v • Reply • Share ›

Elvinas Ancuta ➚ Probal Sikder • 2 years ago

in your application.yml file inside the test folder, change the bootstrap-servers vale from localhost:blah to ${spring.embedded.kafka.brokers}, then the error will disappear

^ | v • Reply • Share ›

codenotfound Mod ➚ Probal Sikder • 2 years ago

Hi,

That's strange, what is your assert in your test method?

Reply • Share ›

Probal Sikder ➚ codenotfound • 2 years ago

Thanks for your quick response.
assert is exactly same as provided in this example.
assertThat(receiver.getLatch().getCount()).isEqualTo(0);

Btw, I presumed this example will run a embedded kafka with spring boot. Is this what I presumed wrong?

Reply • Share ›

rafeeq ➚ Probal Sikder • 2 years ago

I know what the issue is, Unless the message is sent and consumed by Receive method, the latch count will not be 0 it has higher value. I got t same error in a integration test where i do not send and consume immediately in the same test, i send the message and use a Simple consumer to verify that the message was received in this case the Latc count isnt 0, it has a higher value .

Reply • Share ›

codenotfound Mod ➚ Probal Sikder • 2 years ago

The embedded Kafka is started as part of the Unit test case using a *ClassRule*. It is not part of Spring Boot. How are you running your test case?

Reply • Share ›

Probal Sikder ➚ codenotfound • 2 years ago

Just running the mvn test

Reply • Share ›

codenotfound Mod ➚ Probal Sikder • 2 years ago

And what version of Maven are you using?

Reply • Share ›

Probal Sikder ➚ codenotfound • 2 years ago

maven 3.5.0

Reply • Share ›

codenotfound Mod ➚ Probal Sikder • 2 years ago

I think this error comes from the *waitForAssignment()* method. Are you using the *@ClassRule* to start an embedded broker?

Reply • Share ›

Vinicius Correia • 2 years ago

Hello there

congrats for the awesome work. It is soooo useful for many of us. Great job indeed!

I would like to produce and consume messages without using the SpringKafkaApplicationTest class. I want to do it into my main class, how may I achieve that?

Thanks in advance!

▲ | ▼ • Reply • Share ›

codenotfound Mod → Vinicius Correia • 2 years ago

Hi,

Thanks for your kind words!

When you start the above example as a standalone JAR (you can do this using Mave *mvn spring-boot:run*) the listener is created at startup and starts listening to the *hellow* topic. So nothing to do here.

As for sending you would just need to autowire the *Sender* bean (for example in the m method) and use it based on some input, for example you could expose a *RestContro* to trigger it.

Hope this makes sense.

▲ | ▼ • Reply • Share ›

Vishal Aggarwal → codenotfound • 2 years ago

Thanks for providing such a nice code.

I am new to spring.

Can you share a sample code to execute the above code from main instead of running it as test.

▲ | ▼ • Reply • Share ›

codenotfound Mod → Vishal Aggarwal • 2 years ago

Hi,

Thanks for your comment.

What scenario do you have in mind? Are you listening for messages? Or do you want to send messages (what would be the trigger)?

▲ | ▼ • Reply • Share ›

Vishal Aggarwal → codenotfound • 2 years ago

I want to listen messages, I am not producing anything.

▲ | ▼ • Reply • Share ›

codenotfound Mod → Vishal Aggarwal • 2 years ago

You can run by executing following Maven command: *mvn spring-boot:run*

• Reply • Share ›

Vishal Aggarwal ➔ codenotfound • 2 years ago

Thanks.

• Reply • Share ›

Mhavir Teraiya • 2 years ago

Hi Team,

Greetings...!!!

I must say great work and very nicely documented.

I have one small query , how can i receive & print offset no. for particular record i received.
How can we implement this solution in the same example of yours.

Hoping for your reply.

Thank you...!!

Regards,
Mahavir Teraiya

• Reply • Share ›

codenotfound Mod ➔ Mhavir Teraiya • 2 years ago • edited

Hi,

Thanks for the nice feedback.

In order to get the offset for a particular record you could add this metadata by specifying additional parameters using the @Header annotation.

```
@KafkaListener(topics = "${kafka.topic.helloworld}") public void receive(@Payload String payload, @Header(KafkaHeaders.OFFSET) String offset)
{
LOGGER.info("received payload='{}' with offset='{}'", payload, offset);
latch.countDown();
}
```

For more information you can consult: https://docs.spring.io/spri...

• Reply • Share ›

Mhavir Teraiya ➔ codenotfound • 2 years ago

Hi Thank you so much for sharing this,
It almost worked for me , I just stuck at one place. Can you please show how can i implement
KafkaHeaders in this solution.

Because when i am doing this it asking me to create method for KafkaHeaders

which i dont know.

I am using same your code for receiver configuration and receiver , can you please take reference of your code and reply me with the modified code .

Counting on you.

Thank you...!!!

Regards,
Mahavir Teraiya

Reply • Share ›

Shubham Jain • 2 years ago

Hi

I wanted to know how to consume messages as a key value pair. I have tried

```
@KafkaListener(topic = "topic_name")
public void doSomething(String key, String val) {
print(key + " " val);
}
```

But it just prints the val two times

Reply • Share ›

Load more comments