



THE PRACTICAL DEVELOPER

SOFTWARE DEVELOPMENT IS EASY WHEN YOU UNDERSTAND WHAT YOU'RE DOING

[BLOG](#)[TOPICS](#)[MORE](#)[PROJECTS](#)[ABOUT](#)[HOME](#)

Follow @moises_macero

Spring Boot and Kafka – Practical Configuration Examples

On: November 24, 2018 | In: Docker, Kafka, Spring Boot

Spring Boot and Kafka – Practical Configuration Examples

Table of Contents

1. Spring Boot and Kafka – Practical Configuration Examples
2. Multiple consumers in a consumer group
 - 2.1. Logical View
 - 2.2. The Example Use Case
3. Setting up Spring Boot and Kafka
 - 3.1. Starting up Kafka
 - 3.2. Basic Spring Boot and Kafka application
 - 3.3. The Message class
4. Kafka Producer configuration in Spring Boot
 - 4.1. About Kafka Serializers and Deserializers for Java

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

7. Receiving messages with Spring Boot and Kafka in JSON, String and byte[] formats



Quboc

Gamification to improve Code Quality

social



Get [Quboc Open Source](#) for free and improve your code quality



Legacy Code competing between teams

[MORE INFO](#)

GET THE BOOK

A Practical Approach to Microservices.

Build a real project by yourself, and experience pros and cons.

Close and accept

7.0.1. The Typeld Header in Kafka

8. Running the application

8.1. Explanation

9. Try some Kafka Exercises

9.1. Request /hello multiple times

9.2. Reduce the number of partitions

9.3. Change one Consumer's Group Identifier

This blog post shows how to configure Spring Kafka and Spring Boot to send messages using JSON and receive them in multiple formats: JSON, plain Strings or byte arrays. Based on this configuration, you could also switch your Kafka producer from sending JSON to other serialization methods.

This sample application also demonstrates the usage of three Kafka consumers within the same consumer group, so the messages are load-balanced between the three. Each consumer implements a different deserialization approach.

Besides, at the end of this post, you will find some practical exercises in case you want to grasp some Kafka concepts like the Consumer Group and Topic partitions.

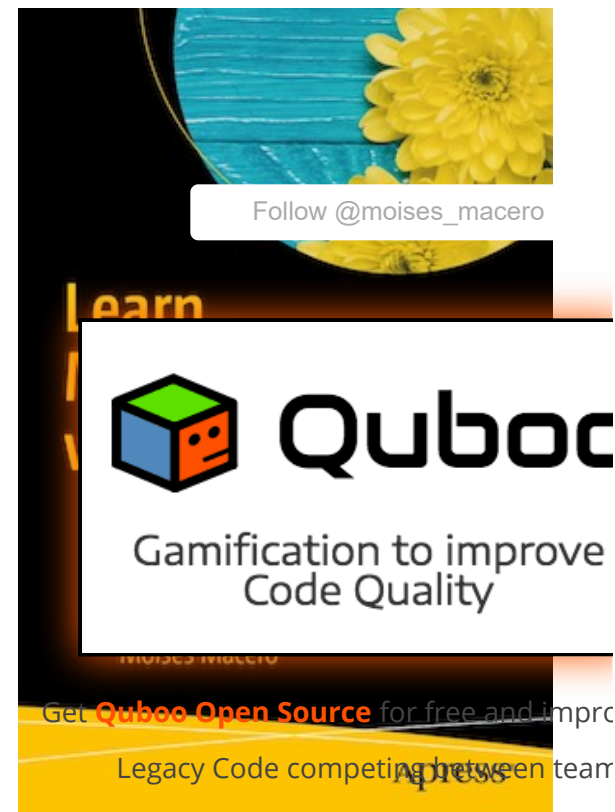
Multiple consumers in a consumer group

Logical View

To better understand the configuration, have a look at the diagram below. As

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

side, there is only one application, but it implements three kafka consumers



Follow @moises_macero

Learn

Quboc

Gamification to improve Code Quality

Get **Quboc Open Source** for free and improve your team's productivity

Legacy Code competing between team

MORE INFO

AMAZON

BUY IT ON
SPRINGER

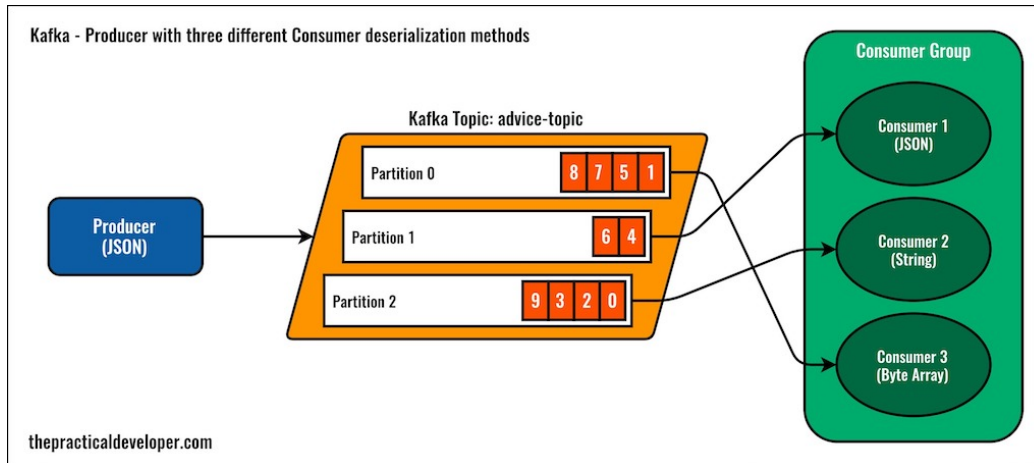
SUBSCRIBE TO TPD

CLICK HERE TO

Close and accept

SEARCH

with the same `group.id` property. This is the configuration needed for having them in the same Kafka Consumer Group.



When we start the application, Kafka assigns each consumer a different partition. This consumer group will receive the messages in a load-balanced manner. Later in this post, you'll see what is the difference if we make them have different group identifiers (you probably know the result if you are familiar with Kafka).

The Example Use Case

The logic we are going to build is simple. Each time we call a given REST endpoint, `hello`, the app will produce a configurable number of messages and send them to the same topic, using a sequence number as the Kafka key. It will wait (using a `CountDownLatch`) for all messages to be consumed before

when they receive a new message.

Q Type Search Term ...

MORE

FOLLOW ME ON TWITTER

Follow @moises_macero

Moisés Macero Retweeted

Mario Fusco



Get **Quboo Open Source** for free and improve your code quality
Apr 17, 2019

Legacy Code competing between teams

Moisés Macero Retweeted

MORE INFO

@iamdeveloper

Embed

View on Twitter

EFFECTIVE MEETINGS

Check out **Agile Meeting Cards**, a new game to improve your meeting quality,

Close and accept

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

Easy, right? Let's see how to build it.

Setting up Spring Boot and Kafka



All the code in this post is available on GitHub: [Kafka and Spring Boot Example](#). If you find it useful, please give it a star!

Starting up Kafka

First, you need to have a running Kafka cluster to connect to. For this application, I will use docker-compose and Kafka running in a single node. This is clearly far from being a production configuration, but it is good enough for the goal of this post.

```

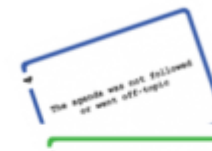
1 version: '2'
2 services:
3   zookeeper:
4     image: wurstmeister/zookeeper
5     ports:
6       - "2181:2181"
7   kafka:
8     image: wurstmeister/kafka
9     ports:
10      - "9092:9092"
11    environment:
12      KAFKA_ADVERTISED_HOST_NAME: 127.0.0.1
13      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181

```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

our topic from the Spring Boot application since we want to pass some custom

MORE



Follow @moises_macero



Quboc

Gamification to improve Code Quality

REC

- Java and JSON – Jackson Serialization with ObjectMapper
- Get [Quboc Open Source](#) for free and improve your code quality
- Spring Boot and Kafka – Practical Configuration Examples
- How to write BDD Unit Tests with BDDMockito and AssertJ

MORE INFO

TAG CLOUD

Close and accept

elasticsearch eureka full-reactive-

configuration anyway. If you want to play around with these Docker images (e.g. to use multiple nodes), have a look at the [wurstmeister/zookeeper](#) image [docs](#).

To start up Kafka and Zookeeper containers, just run `docker-compose up` from the folder where this file lives.

Basic Spring Boot and Kafka application

The easiest way to get a skeleton for our app is to navigate to [start.spring.io](#), fill in the basic details for our project and select Kafka as a dependency. Then, download the zip file and use your favorite IDE to load the sources.

Let's use YAML for our configuration. You may need to rename the `application.properties` file inside `src/main/java/resources` to `application.yml`. These are the configuration values we are going to use for this sample application:

```
1 | spring:
```

```
5 |     auto-offset-reset: earliest
```

```
6 |     # change this property if you are using your own
```

[guide](#) [gamification](#) [hystrix integration](#)

MORE [integration tests](#) [jacoco](#) [java](#) [java9](#)

[jboss](#) [JSON](#) [junit](#) [logstash](#)

[microserv](#) [Follow @moises_macero](#)

[reactive](#) [REST](#) [ribbon](#) [scrum](#) [serialization](#)

servi
sprit
test co
wildt



Quboc

Gamification to improve Code Quality

CATEGORIES

Get **Quboo Open Source** for free and improve
Select Category
Legacy Code competing between team

MORE INFO

Close and accept

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

```

7 |      # Kafka cluster or your Docker IP is different
8 |      bootstrap-servers: localhost:9092
9 |
10 | tpd:
11 |   topic-name: advice-topic
12 |   messages-per-request: 10

```

The first block of properties is Spring Kafka configuration:

- The `group-id` that will be used by default by our consumers.
- The `auto-offset-reset` property is set to `earliest`, which means that the consumers will start reading messages from the earliest one available when there is no existing offset for that consumer.
- The server to use to connect to Kafka, in this case, the only one available if you use the single-node configuration. Note that this property is redundant if you use the default value, `localhost:9092`.

The second block is application-specific. We define the Kafka topic name and the number of messages to send every time we do an HTTP REST request.

The Message class

This is the Java class that we will use as Kafka message. Nothing complex here, just an immutable class with `@JsonProperty` annotations in the constructor parameters so Jackson can deserialize it properly.

```

1 | package io.tpd.kafkaexample;
2 |

```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

```

6 |     private final String message;
7 |     private final int identifier;

```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between teams

MORE INFO

Close and accept

```

8
9     public PracticalAdvice(@JsonProperty("message") final String message,
10                           @JsonProperty("identifier") final int identifier) {
11         this.message = message;
12         this.identifier = identifier;
13     }
14
15     public String getMessage() {
16         return message;
17     }
18
19     public int getIdentifier() {
20         return identifier;
21     }
22
23     @Override
24     public String toString() {
25         return "PracticalAdvice::toString() {" +
26             "message='" + message + '\'' +
27             ", identifier=" + identifier +
28             "'}";
29     }
30 }

```

Kafka Producer configuration in Spring Boot

To keep the application simple, we will add the configuration in the main Spring Boot class. Eventually, we want to include here both producer and consumer configuration, and use three different variations for deserialization. Remember that you can find the complete source code in the [GitHub repository](#).

First, let's focus on the Producer configuration.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

```

3
4     public static void main(String[] args) {

```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between team

MORE INFO

Close and accept

```
5      SpringApplication.run(KafkaExampleApplication.class, args);
6  }
7
8  @Autowired
9  private KafkaProperties kafkaProperties;
10
11  @Value("${tpd.topic-name}")
12  private String topicName;
13
14  // Producer configuration
15
16  @Bean
17  public Map<String, Object> producerConfigs() {
18      Map<String, Object> props =
19          new HashMap<>(kafkaProperties.buildProducerProperties(
20              props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
21                  StringSerializer.class);
22              props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
23                  JsonSerializer.class);
24              return props;
25  }
26
27  @Bean
28  public ProducerFactory<String, Object> producerFactory() {
29      return new DefaultKafkaProducerFactory<>(producerConfigs());
30  }
31
32  @Bean
33  public KafkaTemplate<String, Object> kafkaTemplate() {
34      return new KafkaTemplate<>(producerFactory());
35  }
36
37  @Bean
38  public NewTopic adviceTopic() {
39      return new NewTopic(topicName, 3, (short) 1);
40  }
```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between team

MORE INFO

Close and accept

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

In this configuration, we are setting up two parts of the application.

- The **KafkaTemplate** instance, which is the object we will use to send messages to Kafka. We don't want to use the default one so we need to inject our custom version in the Spring's application context.
 - We type (with generics) the KafkaTemplate to have a plain String key, and an Object as value. The reason to have Object as a value is that we want to send multiple object types with the same template. The KafkaTemplate accepts as a parameter a ProducerFactory that we also create in our configuration.
 - The ProducerFactory we use is the default one, but we need to explicitly configure here since we want to pass it our custom producer configuration.
 - The Producer Configuration is a simple key-value map. We inject the default properties using `@Autowired` to obtain the **KafkaProperties** bean and then we build our map passing the default values for the producer, and overriding the default Kafka key and value serializers. The producer will serialize keys as Strings using the Kafka library's **StringSerializer** and will do the same for values but this time using JSON, with a **JsonSerializer**, in this case provided by Spring Kafka.
- The **Kafka topic** we're going to use. By injecting a **NewTopic** instance, we're instructing the Kafka's **AdminClient** bean (already in the context) to create a topic with the given configuration. The first parameter is the name (advice-topic, from the app configuration), the second is the number of partitions (3) and the third one is the replication factor (one, since we're using a single node anyway).

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

ABOUT KAFKA SERIALIZERS AND DESERIALIZERS FOR JAVA

MORE

Follow @moises_macero



Get **Quboo Open Source** for free and improve Legacy Code competing between teams

MORE INFO

Close and accept

There are a few basic Serializers available in the core Kafka library ([javadoc](#)) for Strings, all kind of number classes and byte arrays, plus the JSON ones provided by Spring Kafka ([javadoc](#)).

On top of that, you can create your own Serializers and Deserializers just by implementing `Serializer` or `ExtendedSerializer`, or their corresponding versions for deserialization. That gives you a lot of flexibility to optimize the amount of data traveling through Kafka, in case you need to do so. As you can see in those interfaces, Kafka works with plain byte arrays so, eventually, no matter what complex type you're working with, it needs to be transformed to a `byte[]`.

Knowing that, you may wonder why someone would want to use JSON with Kafka. It's quite inefficient since you're transforming your objects to JSON and then to a byte array. But you have to consider two main advantages of doing this:

- JSON is more readable by a human than an array of bytes. If you want to debug or analyze the contents of your Kafka topics, it's going to be way simpler than looking at bare bytes.
- JSON is a standard, whereas default byte array serializers depend on the programming language implementation. Thus, if you want to consume messages from multiple programming languages, you would need to replicate the (de)serializer logic in all those languages.

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

or speed in your implementation, you may want to create your own serializer and deserializer for your own serializer/deserializer implementation.

MORE

Follow [@moises_macero](#)



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between team

MORE INFO

Close and accept

Sending messages with Spring Boot and Kafka

Following the plan, we create a Rest Controller and use the injected `KafkaTemplate` to produce some JSON messages when the endpoint is requested.

This is the first implementation of the controller, containing only the logic producing the messages.

```

1  @RestController
2  public class HelloKafkaController {
3
4      private static final Logger logger =
5          LoggerFactory.getLogger(HelloKafkaController.class);
6
7      private final KafkaTemplate<String, Object> template;
8      private final String topicName;
9      private final int messagesPerRequest;
10     private CountDownLatch latch;
11
12     public HelloKafkaController(
13         final KafkaTemplate<String, Object> template,
14         @Value("${tpd.topic-name}") final String topicName,
15         @Value("${tpd.messages-per-request}") final int messagesPe
16         this.template = template;
17         this.topicName = topicName;
18         this.messagesPerRequest = messagesPerRequest;
19     }
20
21     @GetMapping("/hello")
22     public String hello() throws Exception {
23         latch = new CountDownLatch(messagesPerRequest);

```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

```

27         );
28         latch.await(60, TimeUnit.SECONDS);

```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between teams

MORE INFO

Close and accept

```

29     logger.info("All messages received");
30     return "Hello Kafka!";
31 }
32 }

```

In the constructor, we pass some configuration parameters and the `KafkaTemplate` that we customized to send String keys and JSON values. Then, when the API client requests the `/hello` endpoint, we send 10 messages (that's the configuration value) and then we block the thread for a maximum of 60 seconds. As you can see, there is no implementation yet for the Kafka consumers to decrease the latch count. After the latch gets unlocked, we return the message `Hello Kafka!` to our client.

This entire lock idea is not a pattern that would see in a real application, but it's good for the sake of this example. That way, you can check the number of messages received. If you prefer, you can remove the latch and return the "Hello Kafka!" message before receiving the messages.

Kafka Consumer configuration

As mentioned previously on this post, we want to demonstrate different ways of deserialization with Spring Boot and Spring Kafka and, at the same time, see how multiple consumers can work in a load-balanced manner when they are part of the same consumer-group.

```

1 @SpringBootApplication

```

```

2 public class KafkaExampleApplication {

```

```

5     SpringApplication.run(KafkaExampleApplication.class, args);
6 }

```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between teams

MORE INFO

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

```

7
8  @Autowired
9  private KafkaProperties kafkaProperties;
10
11  @Value("${tpd.topic-name}")
12  private String topicName;
13
14  // Producer configuration
15  // omitted...
16
17  // Consumer configuration
18
19  // If you only need one kind of deserialization, you only need to
20  // Consumer configuration properties. Uncomment this and remove al
21  // @Bean
22  // public Map<String, Object> consumerConfigs() {
23  //     Map<String, Object> props = new HashMap<>()
24  //         kafkaProperties.buildConsumerProperties()
25  //     };
26  //     props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
27  //         StringDeserializer.class);
28  //     props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
29  //         JsonSerializer.class);
30  //     props.put(ConsumerConfig.GROUP_ID_CONFIG,
31  //         "tpd-loggers");
32  //
33  //     return props;
34  // }
35
36  @Bean
37  public ConsumerFactory<String, Object> consumerFactory() {
38      final JsonSerializer<Object> jsonDeserializer = new JsonDese
39      jsonDeserializer.addTrustedPackages("*");
40      return new DefaultKafkaConsumerFactory<>(
41          kafkaProperties.buildConsumerProperties(), new StringD
42      );

```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and imprc
Legacy Code competing between team

MORE INFO

Close and accept

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

```

46  public ConcurrentKafkaListenerContainerFactory<String, Object> kaf
47  ConcurrentKafkaListenerContainerFactory<String, Object> factor

```

```

48         new ConcurrentKafkaListenerContainerFactory<>();
49         factory.setConsumerFactory(consumerFactory());
50
51         return factory;
52     }
53
54     // String Consumer Configuration
55
56     @Bean
57     public ConsumerFactory<String, String> stringConsumerFactory() {
58         return new DefaultKafkaConsumerFactory<>(
59             kafkaProperties.buildConsumerProperties(), new StringDe
60         );
61     }
62
63     @Bean
64     public ConcurrentKafkaListenerContainerFactory<String, String> kaf
65         ConcurrentKafkaListenerContainerFactory<String, String> factor
66         new ConcurrentKafkaListenerContainerFactory<>();
67         factory.setConsumerFactory(stringConsumerFactory());
68
69         return factory;
70     }
71
72     // Byte Array Consumer Configuration
73
74     @Bean
75     public ConsumerFactory<String, byte[]> byteArrayConsumerFactory()
76         return new DefaultKafkaConsumerFactory<>(
77             kafkaProperties.buildConsumerProperties(), new StringDe
78         );
79     }
80
81     @Bean
82     public ConcurrentKafkaListenerContainerFactory<String, byte[]> kaf
83         ConcurrentKafkaListenerContainerFactory<String, byte[]> factor

```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and imprc
Legacy Code competing between team

MORE INFO

Close and accept

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

This configuration may look extense but take into account that, to demonstrate these three types of deserialization, we have repeated three times the creation of the `ConsumerFactory` and the `KafkaListenerContainerFactory` instances so we can switch between them in our consumers.

The basic steps to configure a consumer are:

1. [Omitted] Set up the Consumer properties in a similar way as we did for the Producer. We can skip this step since the only configuration we need is the Group ID, specified in the Spring Boot properties file, and the key and value deserializers, which we will override while creating the customized consumer and `KafkaListener` factories. **If you only need one configuration**, meaning always the same type of Key and Value deserializers, this commented code block **is the only thing you need**. Adjust the deserializer types to the ones you want to use.
2. Create the `ConsumerFactory` to be used by the `KafkaListenerContainerFactory`. We create three, switching the value deserializer in each case to 1) a JSON deserializer, 2) a String deserializer and 3) a Byte Array deserializer.
 - a. Note that, after creating the JSON Deserializer, we're including an extra step to specify that we trust all packages. You can fine-tune this in your application if you want. If we don't do this, we will get an error message saying something like: `java.lang.IllegalArgumentException: The`

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

3. Construct the `KafkaListenerContainerFactory` (a concurrent one) using the previously configured Consumer Factory. Again, we do this three times to use

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and imprc
Legacy Code competing between team

MORE INFO

Close and accept

a different one per instance.

Receiving messages with Spring Boot and Kafka in JSON, String and byte[] formats

It's time to show how the Kafka consumers look like. We will use the `@KafkaListener` annotation since it simplifies the process and takes care of the deserialization to the passed Java type.

```

1  @RestController
2  public class HelloKafkaController {
3
4      private static final Logger logger =
5          LoggerFactory.getLogger(HelloKafkaController.class);
6
7      private final KafkaTemplate<String, Object> template;
8      private final String topicName;
9      private final int messagesPerRequest;
10     private CountDownLatch latch;
11
12     public HelloKafkaController(
13         final KafkaTemplate<String, Object> template,
14         @Value("${tpd.topic-name}") final String topicName,
15         @Value("${tpd.messages-per-request}") final int messagesPe
16     {
17         this.template = template;
18         this.topicName = topicName;
19         this.messagesPerRequest = messagesPerRequest;
20     }
21
22     @GetMapping("/hello")
23     public String hello() throws Exception {

```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

```

26         new PracticalAdvice("A Practical Advice", i))
27     };

```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between team

MORE INFO

Close and accept


```

28     latch.await(60, TimeUnit.SECONDS);
29     logger.info("All messages received");
30     return "Hello Kafka!";
31 }
32
33 @KafkaListener(topics = "advice-topic", clientIdPrefix = "json",
34               containerFactory = "kafkaListenerContainerFactory")
35 public void listenAsObject(ConsumerRecord<String, PracticalAdvice>
36                           @Payload PracticalAdvice payload) {
37     logger.info("Logger 1 [JSON] received key {}: Type [{}] | Payload: {} | Payload Type: {} | Payload Value: {}",
38               typeIdHeader(cr.headers()), payload, cr.toString());
39     latch.countDown();
40 }
41
42 @KafkaListener(topics = "advice-topic", clientIdPrefix = "string",
43               containerFactory = "kafkaListenerStringContainerFactory")
44 public void listenAsString(ConsumerRecord<String, String> cr,
45                           @Payload String payload) {
46     logger.info("Logger 2 [String] received key {}: Type [{}] | Payload: {} | Payload Type: {} | Payload Value: {}",
47               typeIdHeader(cr.headers()), payload, cr.toString());
48     latch.countDown();
49 }
50
51 @KafkaListener(topics = "advice-topic", clientIdPrefix = "bytearray",
52               containerFactory = "kafkaListenerByteArrayContainerFactory")
53 public void listenAsByteArray(ConsumerRecord<String, byte[]> cr,
54                              @Payload byte[] payload) {
55     logger.info("Logger 3 [ByteArray] received key {}: Type [{}] | Payload: {} | Payload Type: {} | Payload Value: {}",
56               typeIdHeader(cr.headers()), payload, cr.toString());
57     latch.countDown();
58 }
59
60 private static String typeIdHeader(Headers headers) {
61     return StreamSupport.stream(headers.spliterator(), false)
62         .filter(header -> header.key().equals("__TypeId__"))
63         .findFirst().map(header -> new String(header.value()));

```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between team

MORE INFO

Close and accept

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

There are three listeners in this class. First, let's describe the

`@KafkaListener` annotation's parameters:

- All listeners are consuming from the same topic, `advice-topic`. This parameter is mandatory.
- The parameter `clientIdPrefix` is optional. I'm using it here so the logs are more human-friendly. You will know which consumer does what by its name prefix. Kafka will append a number this prefix.
- The `containerFactory` parameter is optional, you can also rely on naming convention. If you don't specify it, it will look for a bean with the name `kafkaListenerContainerFactory`, which is also the default name used by Spring Boot when autoconfiguring Kafka. You can also override it by using the same name (although it looks like magic for someone who doesn't know about the convention). We need to set it explicitly because we want to use a different one for each listener, to be able to use different deserializers.

Note that the first argument passed to all listeners is the same, a

`ConsumerRecord`. The second one, annotated with `@Payload` is redundant if we use the first. We can access the payload using the method `value()` in

`ConsumerRecord`, but I included it so you see how simple it's to get directly the message payload by inferred deserialization.

The Typed Header in Kafka

representation since you will only see a byte array in the output

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between teams

MORE INFO

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept

of `ConsumerRecord`'s `toString()` method. This `TypeId` header can be useful for deserialization, so you can find the type to map the data to. It's not needed for JSON deserialization because that specific deserializer is made by the Spring team and they infer the type from the method's argument.

Running the application



All the code in this post is available on GitHub: [Kafka and Spring Boot Example](#). If you find it useful, please give it a star!

Now that we finished the Kafka producer and consumers, we can run Kafka and the Spring Boot app:

```
1 $ docker-compose up -d
2 Starting kafka-example_zookeeper_1 ... done
3 Starting kafka-example_kafka_1 ... done
4 $ mvn spring-boot:run
5 ...
```

The Spring Boot app starts and the consumers are registered in Kafka, which assigns a partition to them. We configured the topic with three partitions, so each consumer gets one of them assigned.

```
1 [Consumer clientId=string-0, groupId=tpd-loggers] Successfully joined g
2 [Consumer clientId=string-0, groupId=tpd-loggers] Setting newly assigne
3 [Consumer clientId=bytearray-0, groupId=tpd-loggers] Successfully joine
4 [Consumer clientId=bytearray-0, groupId=tpd-loggers] Setting newly assi
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

```
7 partitions assigned: [advice-topic-1]
8 partitions assigned: [advice-topic-2]
```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and imprc
Legacy Code competing between team

MORE INFO

Close and accept

```
9 | partitions assigned: [advice-topic-0]
```

We can try now an HTTP call to the service. You can use your browser or `curl`, for example:

```
1 | $ curl localhost:8080/hello
```

The output in the logs should look like this:

```
1 | INFO 15292 --- [ntainer#1-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
2 | INFO 15292 --- [ntainer#2-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
3 | INFO 15292 --- [ntainer#1-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
4 | INFO 15292 --- [ntainer#2-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
5 | INFO 15292 --- [ntainer#1-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
6 | INFO 15292 --- [ntainer#2-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
7 | INFO 15292 --- [ntainer#1-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
8 | INFO 15292 --- [ntainer#2-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
9 | INFO 15292 --- [ntainer#0-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
10 | INFO 15292 --- [ntainer#0-0-C-1] i.tpd.kafkaexample.HelloKafkaControllo
11 | INFO 15292 --- [nio-8080-exec-1] i.tpd.kafkaexample.HelloKafkaControllo
```

Explanation

Kafka is hashing the message key (a simple string identifier) and, based on that, placing messages into different partitions. Each consumer gets the messages in its assigned partition and uses its deserializer to convert it to a Java object.

Remember, our producer always sends JSON values.

As you can see in the logs, each deserializer manages to do its task so the String consumer prints the raw JSON message, the Byte Array shows the byte

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between team

MORE INFO

Close and accept

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

look at the logged ConsumerRecord and you'll see the headers, the assigned partition, the offset, etc.

And that's how you can Send and Receive JSON messages with Spring Boot and Kafka. I hope that you found this guide useful, below you have some code variations so you can explore a bit more how Kafka works.

Should you have any feedback, let me know via [Twitter](#) or comments.

Try some Kafka Exercises

If you are new to Kafka, you may want to try some code changes to better understand how Kafka works.

Request /hello multiple times

Make a few requests and then look at how the messages are distributed across partitions. **Kafka messages with the same key are always placed in the same partitions.** This feature is very useful when you want to make sure that all messages for a given user, or process, or whatever logic you're working on, are received by the same consumer in the same order as they were produced, no matter how much load balancing you're doing.

Reduce the number of partitions

MORE

Follow [@moises_macero](#)



Quboc

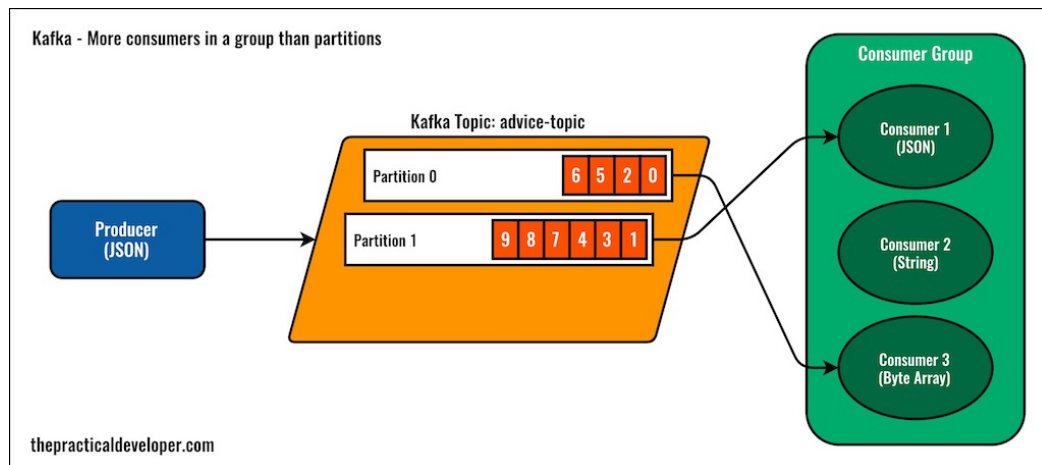
Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between teams

MORE INFO

Close and accept

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)



First, make sure to restart Kafka so you just discard the previous configuration.

Then, redefine the topic in the application to have only 2 partitions:

```
1 @Bean
2 public NewTopic adviceTopic() {
3     return new NewTopic(topicName, 2, (short) 1);
4 }
```

Now, run the app again and do a request to the `/hello` endpoint.

```
1 Logger 3 [ByteArray] received key 0: Type [io.tpd.kafkaexample.Practic
2 Logger 3 [ByteArray] received key 2: Type [io.tpd.kafkaexample.Practic
3 Logger 3 [ByteArray] received key 5: Type [io.tpd.kafkaexample.Practic
4 Logger 3 [ByteArray] received key 6: Type [io.tpd.kafkaexample.Practic
5 Logger 1 [JSON] received key 1: Type [N/A] | Payload: PracticalAdvice:
6 Logger 1 [JSON] received key 3: Type [N/A] | Payload: PracticalAdvice:
7 Logger 1 [JSON] received key 4: Type [N/A] | Payload: PracticalAdvice:
8 Logger 1 [JSON] received key 7: Type [N/A] | Payload: PracticalAdvice:
9 Logger 1 [JSON] received key 8: Type [N/A] | Payload: PracticalAdvice:
10 Logger 1 [JSON] received key 9: Type [N/A] | Payload: PracticalAdvice:
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

behavior since there are no more partitions available for it within the same

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and imprc

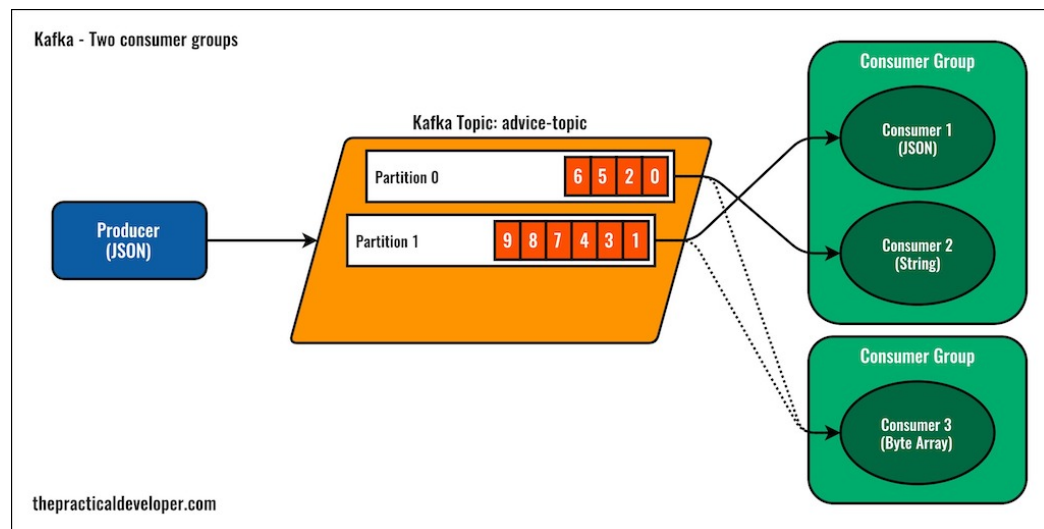
Legacy Code competing between team

MORE INFO

Close and accept

consumer group.

Change one Consumer's Group Identifier



Keep the changes from the previous case, the topic has now only 2 partitions. We are now **changing the group id** of one of our consumers, so it's working independently.

```

1 @KafkaListener(topics = "advice-topic", clientIdPrefix = "bytearray",
2     containerFactory = "kafkaListenerByteArrayContainerFactory",
3     groupId = "tpd-loggers-2")
4 public void listenAsByteArray(ConsumerRecord<String, byte[]> cr,
5     @Payload byte[] payload) {
6     logger.info("Logger 3 [ByteArray] received a payload with size {}",
7         latch.countDown());
8 }

```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use. To find out more, including how to control cookies, see here: [Cookie Policy](#)

to change the `CountDownLatch` so it expects twice the number of messages.

MORE

Follow @moises_macero



Get **Quboo Open Source** for free and improve Legacy Code competing between team

MORE INFO

Close and accept

```
1 latch = new CountDownLatch(messagesPerRequest * 2);
```

Why? This time, let's explain what is going to happen before running the app. As I described at the beginning of this post, **when consumers belong to the same Consumer Group they're (conceptually) working on the same task**. We're implementing a load-balanced mechanism in which concurrent workers get messages from different partitions without needing to process each other's messages.

In this example, I also changed the "task" of the last consumer to better understand this: it's printing something different. Since we changed the group id, this consumer will work independently and Kafka will assign both partitions to it. The Byte Array consumer will receive all messages, working separately from the other two.

```
1 Logger 3 [ByteArray] received a payload with size 47
2 Logger 3 [ByteArray] received a payload with size 47
3 Logger 3 [ByteArray] received a payload with size 47
4 Logger 3 [ByteArray] received a payload with size 47
5 Logger 2 [String] received key 1: Type [io.tpd.kafkaexample.PracticalA
6 Logger 2 [String] received key 3: Type [io.tpd.kafkaexample.PracticalA
7 Logger 2 [String] received key 4: Type [io.tpd.kafkaexample.PracticalA
8 Logger 2 [String] received key 7: Type [io.tpd.kafkaexample.PracticalA
9 Logger 2 [String] received key 8: Type [io.tpd.kafkaexample.PracticalA
10 Logger 3 [ByteArray] received a payload with size 47
11 Logger 2 [String] received key 9: Type [io.tpd.kafkaexample.PracticalA
12 Logger 3 [ByteArray] received a payload with size 47
13 Logger 3 [ByteArray] received a payload with size 47
14 Logger 3 [ByteArray] received a payload with size 47
15 Logger 3 [ByteArray] received a payload with size 47
```

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

```
19 Logger 1 [JSON] received key 5: Type [N/A] | Payload: PracticalAdvice:
20 Logger 1 [JSON] received key 6: Type [N/A] | Payload: PracticalAdvice:
```

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between team

MORE INFO

Close and accept

21 | All messages received

With these exercises, and changing parameters here and there, I think you can grasp better the concepts. Remember: if you liked this post please share it or comment on [Twitter](#).

Related Posts

Java and JSON – Jackson Serialization with O...

Serialization and Deserialization with ObjectMapper This guide contains examples that show you how to serialize and deserialize from Java to JSON. It ...

Sending and receiving JSON messages with Spring Bo...

Exchange JSON messages with Spring Boot AMQP and RabbitMQ Setting up a Spring Boot application using AMQP with RabbitMQ is not really difficult, and ...

MORE

Follow @moises_macero



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between teams

MORE INFO

3 COMMENTS



Very useful. Thanks

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Good article.... Thanks

Close and accept



ANONYMOUS | 4 WEEKS AGO | [PERMALINK](#) | [REPLY](#)



thanks!!

ANONYMOUS | 2 MONTHS AGO | [PERMALINK](#) | [REPLY](#)

MORE

Follow @moises_macero

COMMENTS

Enter your comment here...

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)



My name is Moisés Macero, Software Developer, Architect, and Writer. I like sharing knowledge and coaching. How can I help you?



Quboc

Gamification to improve
Code Quality

Get **Quboo Open Source** for free and improve
Legacy Code competing between teams



©2019 thepracticaldeveloper.com | [Privacy Policy](#)

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: [Cookie Policy](#)

Close and accept