

# Simplest Spring Kafka Producer and Consumer



Marcos Maia



Jul 13 '19 Updated on Mar 14, 2020 □ 5 min read

#springboot #kafka #java #springkafka

Let's now build and run the simplest example of a Kafka Consumer and then a Kafka Producer using spring-kafka. If you need assistance with Kafka, spring boot or docker which are used in this article, or want to check out the sample application from this post please check the [References](#) section below.

The first step is to create a simple [Spring Boot maven Application](#) and make sure to have spring-kafka dependency to pom.xml

```
<dependency>
  <groupId>org.springframework.kafka</groupId>
  <artifactId>spring-kafka</artifactId>
</dependency>
```

## Create a Spring Kafka Consumer

Let's now write the simplest possible Kafka consumer with spring-kafka using spring-boot default configurations.

Create a class called `SimpleConsumer` and add a method with the `@KafkaListener` annotation.

```
package io.stockgeeks.springkafka.springkafkaapp;

import org.springframework.kafka.annotation.KafkaListener;
import org.springframework.stereotype.Service;

@Service
public class SimpleConsumer {
    @KafkaListener(id = "simple-consumer", topics = "simple-message")
    public void consumeMessage(String message) {
        System.out.println("Got message: " + message);
    }
}
```

This is it, this is all it takes because we are relying on spring-boot default configurations.

## Start Kafka and Zookeeper

As we've seen in details on [this other article](#), we're going to be using docker-compose to run our local Kafka for development, let's start our Kafka and Zookeeper containers:

```
docker-compose up -d
```

Make sure the containers are running:

```
docker ps
```

You should see Kafka and Zookeeper running:

spring-kafka-app gtt:(master) docker ps					
CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
ad7b10f08e3e	wurstmeister/kafka:2.12-2.2.1	"start-kafka.sh"	2 days ago	Up About a minute	0.0.0.0:9092->9092/tcp
45e4274970c2	wurstmeister/zookeeper:latest	"/bin/sh -c '/usr/sb."	2 days ago	Up 2 minutes	22/tcp, 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp
					NAME
					kafka
					zookeeper

## Run the app

Let's now compile and run the application, if you need more detailed instructions please check [this post](#), run the following commands to build and run the application:

```
mvn clean package
```

and let's run it:

```
mvn spring-boot:run
```

The application will start and you will see on the standard output the configurations for the consumer, the Kafka version being used and a message `Started SpringKafkaApplication in x seconds.`

```

2019-07-13 06:57:13.309 INFO 30221 --- main o.a.kafka.common.utils.AppInfoParser : Kafka version : 2.0.1
2019-07-13 06:57:13.346 INFO 30221 --- main o.a.kafka.common.utils.AppInfoParser : Kafka commitId : fa14785e1bd2ce5
2019-07-13 06:57:13.351 INFO 30221 --- main o.s.s.c.ThreadPoolTaskScheduler : Initializing ExecutorService
2019-07-13 06:57:13.363 INFO 30221 --- consumer-o-c-1 org.apache.kafka.clients.Metadata : Cluster ID: Ww7P5ecQj-aw0f2AZ0hhg
2019-07-13 06:57:13.365 INFO 30221 --- consumer-o-c-1 o.a.k.c.c.internals.AbstractCoordinator : [Consumer clientId=consumer-1, groupId=single-consumer] Discovered group coordinator kafka:9092 (id: 2147483646 rack: null)
2019-07-13 06:57:13.369 INFO 30221 --- consumer-o-c-1 o.a.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-2, groupId=single-consumer] Revoking previously assigned partitions []
2019-07-13 06:57:13.369 INFO 30221 --- consumer-o-c-1 o.s.k.l.KafkaMessageListenerContainer : partitions revoked: []
2019-07-13 06:57:13.369 INFO 30221 --- consumer-o-c-1 o.a.k.c.c.internals.AbstractCoordinator : [Consumer clientId=consumer-2, groupId=single-consumer] (Re-)Joining group
2019-07-13 06:57:13.387 INFO 30221 --- consumer-o-c-1 o.a.k.c.c.internals.AbstractCoordinator : [Consumer clientId=consumer-2, groupId=single-consumer] Successfully joined group with generation 13
2019-07-13 06:57:13.388 INFO 30221 --- consumer-o-c-1 o.a.k.c.c.internals.ConsumerCoordinator : [Consumer clientId=consumer-2, groupId=single-consumer] Setting newly assigned partitions [single-message-0]
2019-07-13 06:57:13.402 INFO 30221 --- consumer-o-c-1 o.s.k.l.KafkaMessageListenerContainer : partitions assigned: [single-message-0]
2019-07-13 06:57:13.422 INFO 30221 --- main o.s.s.w.embedded.tomcat.TomcatEmbeddedServer : Tomcat started on port(s): 8080 (http) with context path ''
2019-07-13 06:57:13.425 INFO 30221 --- main l.s.s.s.SpringKafkaApplication : Started SpringKafkaApplication in 2.325 seconds (JVM running for 5.218)

```

Make sure to keep the application running, don't close the terminal window where it's running. Let's now produce a few messages with the Kafka console producer and see our consumer processing the messages and logging them out.

## Produce message using the Kafka console producer

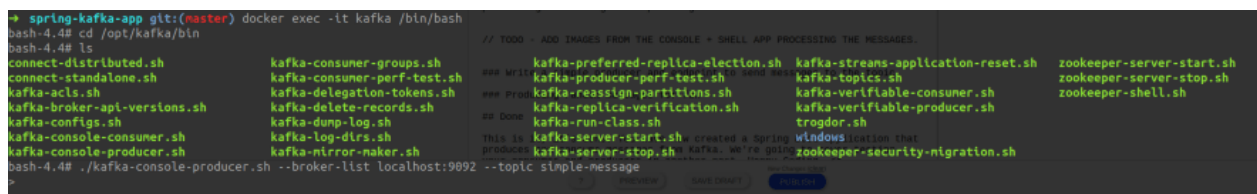
Open a new terminal and enter the Kafka running container so we can use the console producer:

```
docker exec -it kafka /bin/bash
```

Once inside the container `cd /opt/kafka/bin`, the command line scripts for Kafka in this specific image we're using are located in this folder. If you're using different docker images those scripts might be in some other location.

Run the console producer which will enable you to send messages to Kafka:

```
./kafka-console-producer.sh --broker-list localhost:9092 --topic simple-message
```



The screenshot shows a terminal window with the following content:

```
spring-kafka-app git:(master) docker exec -it kafka /bin/bash
bash-4.4# cd /opt/kafka/bin
bash-4.4# ls
connect-distributed.sh  kafka-consumer-groups.sh  kafka-preferred-replica-election.sh  kafka-streams-application-reset.sh  zookeeper-server-start.sh
connect-standalone.sh  kafka-consumer-perf-test.sh  kafka-producer-perf-test.sh  kafka-topics.sh  zookeeper-server-stop.sh
kafka-acls.sh          kafka-delegation-tokens.sh  kafka-reassign-partitions.sh  kafka-verifiable-consumer.sh  zookeeper-shell.sh
kafka-broker-api-versions.sh  kafka-delete-records.sh  kafka-replica-verification.sh  kafka-verifiable-producer.sh
kafka-configs.sh        kafka-dump-log.sh         kafka-run-class.sh             trogdor.sh
kafka-console-consumer.sh  kafka-log-dirs.sh        kafka-server-start.sh          windows
kafka-console-producer.sh  kafka-mirror-maker.sh    kafka-server-stop.sh          zookeeper-security-migration.sh

bash-4.4# ./kafka-console-producer.sh --broker-list localhost:9092 --topic simple-message
```

The console will now block and you can write your message and hit enter, for each time you do this one message will be produced to the `simple-topic`. Try sending a few messages and watch the application standard output in the shell where you are running your Spring Boot application processing the messages and printing them.

```
2019-07-13 00:57:33.402 INFO 30221 --- [-consumer-0-C-1] o.s.k.l.KafkaMessageListenerContainer: you partitions assigned: [single-message-0]
2019-07-13 00:57:33.422 INFO 30221 --- [-consumer-0-C-1] o.s.k.l.KafkaMessageListenerContainer: Tomcat started on port(s): 8080 (http) with context path '/'
2019-07-13 00:57:33.425 INFO 30221 --- [-consumer-0-C-1] main l.s.s.SpringKafkaAppApplication: Started SpringKafkaAppApplication in 2.325 seconds (JVM running for 5.218)
2019-07-13 01:24:28.686 INFO 30221 --- [-consumer-0-C-1] l.s.s.SpringKafkaApp.SimpleConsumer: <D> Consumer got message: Sending my first message
2019-07-13 01:25:43.480 INFO 30221 --- [-consumer-0-C-1] l.s.s.SpringKafkaApp.SimpleConsumer: <D> Consumer got message: Another message
2019-07-13 01:25:49.228 INFO 30221 --- [-consumer-0-C-1] l.s.s.SpringKafkaApp.SimpleConsumer: <D> Consumer got message: Let's try one more time

docker-kafka-console-producer
Topic: single-message
Message: Sending my first message
Message: Another message
Message: Let's try one more time
^C
```

## Write a simple producer

Time to create our spring-kafka producer. Create a class called `SimpleProducer`, we will again use the defaults for the producer as we did for the consumer.

```
package io.stockgeeks.springkafka.springkafkaapp;

import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

@Service
public class SimpleProducer {

    private KafkaTemplate<String, String> simpleProducer;

    public SimpleProducer(KafkaTemplate<String, String> simpleProducer) {
        this.simpleProducer = simpleProducer;
    }

    public void send(String message) {
        simpleProducer.send("simple-message", message);
    }
}
```

## Write an endpoint

Let's now create a simple endpoint which will receive a text message and publish it to Kafka, we're always returning 200 OK for now.

```

package io.stockgeeks.springkafka.springkafkaapp;

import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api")
public class MessageApi {

    private final SimpleProducer simpleProducer;

    public MessageApi(SimpleProducer simpleProducer) {
        this.simpleProducer = simpleProducer;
    }

    @PostMapping("/message")
    public ResponseEntity<String> message(@RequestBody String message) {
        simpleProducer.send(message);
        return ResponseEntity.ok("Message received: " + message);
    }
}

```

Build and run the application.

```
mvn clean package && mvn spring-boot:run
```

There's a good chance you will get an error when running the application on your development machine now, this happens because your application is running inside your normal host network and Kafka and zookeeper are running inside the "docker network".

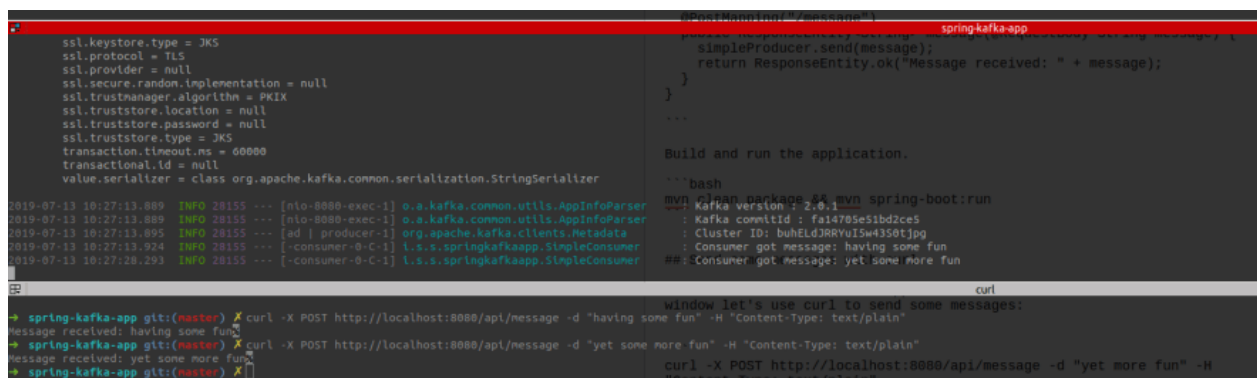
There are some ways to solve this, the best is to pass in your development machine hostname to docker-compose when starting the containers if you open the docker-compose file from this project it has an entry at the `KAFKA_ADVERTISED_LISTENERS`:  
`... LISTENER_DOCKER_EXTERNAL like ${DOCKER_HOST_IP:-kafka}:9092` this tells compose to try to use the passed in hostname or Kafka by default, check the comments in the compose file to find out how to fix it and check the references section below for more details.

## Send some messages with curl

Now make sure to watch the application terminal and in another terminal window let's use curl to send some messages:

```
curl -X POST http://localhost:8080/api/message -d "yet more fun" -H "(
```

You should see the response on the same terminal where curl was executed also check the consumer processing the message and printing it to the terminal where the application is running.



The screenshot shows two terminal windows. The top window displays the logs of a Spring Kafka application. It shows the application starting up, with SSL configuration details and Kafka version 2.0.1. The logs indicate that the application is running successfully and has received messages. The bottom window shows the output of a curl command, which is a POST request to the /api/message endpoint with the message "yet more fun". The curl command is executed in a terminal window, and the output shows the message received.

Done

This is it, it's done. You have now created the simplest possible Spring Boot application that produces and consume messages from Kafka. The reason it looks so simple is that we are relying on Spring Boot and spring-kafka default configurations.

If you want to know more about how Spring Boot or Kafka Works please take a look at the links in the next session where you'll find some references with further details.

We're going to cover testing your consumer and producers in another post. Happy Coding.

## References

[Source code](#) with the application created in this post.

To set up your environment with java, maven, docker and docker-compose, please check how to [set up your environment for the example tutorials](#).

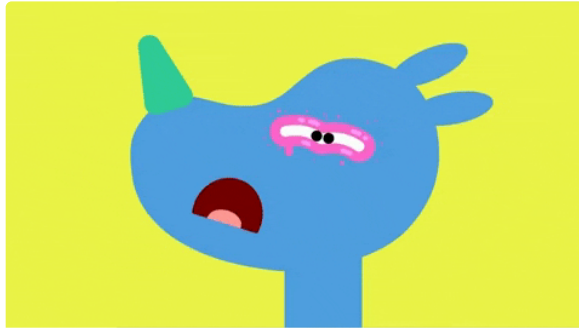
If you need some quick introduction to Kafka: [Kafka - Crash Course](#)

For some insights on how to use docker-compose for local development, please check this post: [One to run them all](#) where you will also learn some useful Kafka commands.

If you're new to Spring Boot, please take a look at [Spring Boot - Crash Course](#)

[Docker compose environment variables](#) to understand the configuration for the Kafka Advertise Listener.





Sore eyes?

[dev.to](#) now has dark mode.

Go to the "misc" section of  
[your settings](#) and select  
night theme ♥



**Marcos Maia**

I am an experienced Developer, Trainer and Speaker. Eager to learn and share knowledge. A bit introspective. Keep coding. Be humble. Help others.

@thegroo  thegroo  mmaia  [gitlab.com/marcosmaia](https://gitlab.com/marcosmaia)

Discussion

---

[code of conduct - report abuse](#)