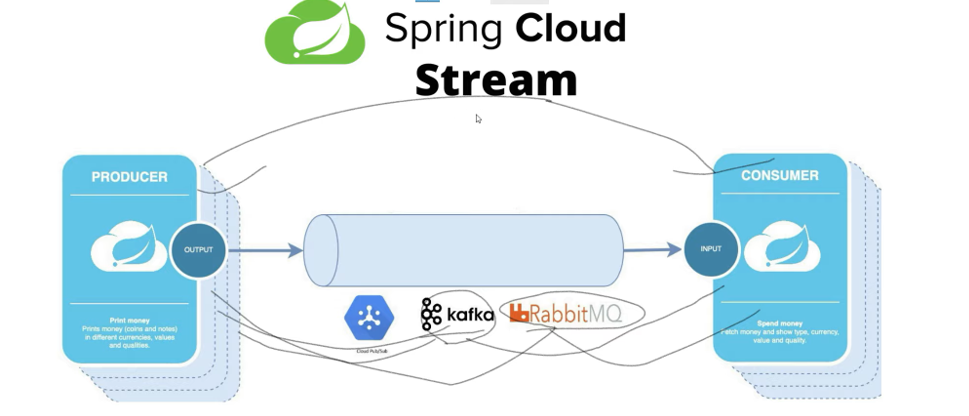
In this tutorial we will understand what is Spring Cloud Stream and how to use it in microservice architecture. Basically, Spring Cloud Stream is a framework for building highly scalable event-driven microservices which connected with shared messaging system. Now let's understand real-time use of Spring Cloud Stream with one architectural diagram. If you can see this diagram, I have two microservices which is up and running. One producer who will publish the event and one consumer who will subscribe the event. So, to perform this event-driven architecture we need one of the messaging system. So, you can see here I have three messaging system. One is RabbitMQ, one is Kafka and another one is Google PubSub. Let's assume as part of my project requirement I need to use Kafka. So, as I am using Kafka, I need to add Kafka related configuration in my producer section. Also, I need to add Kafka related configuration in my consumer section. Let's assume my requirement got changed. I don't want to use Kafka. So, what I need to do I need to again go to the code, and I need to remove Kafka related configuration. Now I want to use RabbitMQ instead of Kafka. So again, go to your code and add RabbitMQ both in producer section also in consumer section. So, if you observe this my application is completely tightly coupled with messaging system. Isn't it? Which does not look good.



So, to solve this problem we can use Spring Cloud Stream where we don't need to specify the implementation details of the messaging system which I am using. Only we need to specify the required binding dependency. Then rest of the things will be taken care by Spring Cloud Stream. So, if I will use Kafka only, I need to add Kafka binder dependency in producer and consumer. Now how my microservice will communicate with each other using Kafka that will be taken care by Spring Cloud Stream. Similarly, if you don't want to use Kafka in future if you want to move to RabbitMQ just remove the Kafka binder dependency in producer and consumer just add RabbitMQ related binder dependency in producer and consumer. Now how they will communicate will be decided and will be handled by Spring Cloud Stream. So in this specific example we are going to use Kafka messaging binder. So, we are going to create two microservices one will publish the event and one will consume the event over the topic and also, we are not going to write any Kafka related configuration in my microservices.

Okay, so we will verify how this Spring Cloud Stream helps us to communicate between microservices using event driven architecture. So, let's quickly create a Spring Boot project to understand this implementation. So as usual let's get started. So, before we jump to the Spring Cloud Stream first you should have basic knowledge about Kafka component. So, if anyone not aware about it you can check out my Apache Kafka playlist and also, I will add this playlist link in video description so that you guys can refer it. Okay, and also, I added all the required command to play with Apache Kafka. So, let's follow this step to up our Kafka server and zookeeper server. So, to up our zookeeper server we need to run this command. So, all the source code you will find in this repository. Okay, so I will mention this repository in video description so that you can find all the command as well. Okay, so first let's up our zookeeper server. Let's go to the directory where you installed your Apache Kafka type here command and we can start our zookeeper server. Yeah, it's up now. So same way Okay. So now let's start the Kafka server. Yeah. So, you can see here our Kafka server is up also our zookeeper server is also up. Okay. Now let's start the development. Let's go to our IntelliJ ID. Here. Let's create two projects.

**Few kafka start-up commands->**

**Apache-Kafka Binary Distribution**[**Download**](http://apachemirror.wuchna.com/kafka/2.3.1/kafka_2.11-2.3.1.tgz)**.**

**Strat Zookeeper server**

**.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties**

**Strat Kafka server**

**.\bin\windows\kafka-server-start.bat .\config\server.properties**

**Create Topic**

**.\bin\windows\kafka-topics.bat --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic NewTopic**

**List down all available topics**

**.\bin\windows\kafka-topics.bat --list --zookeeper localhost:2181**

**Produce a message**

**.\bin\windows\kafka-console-producer.bat --broker-list localhost:9092 --topic TestTopic**

**Consume a message**

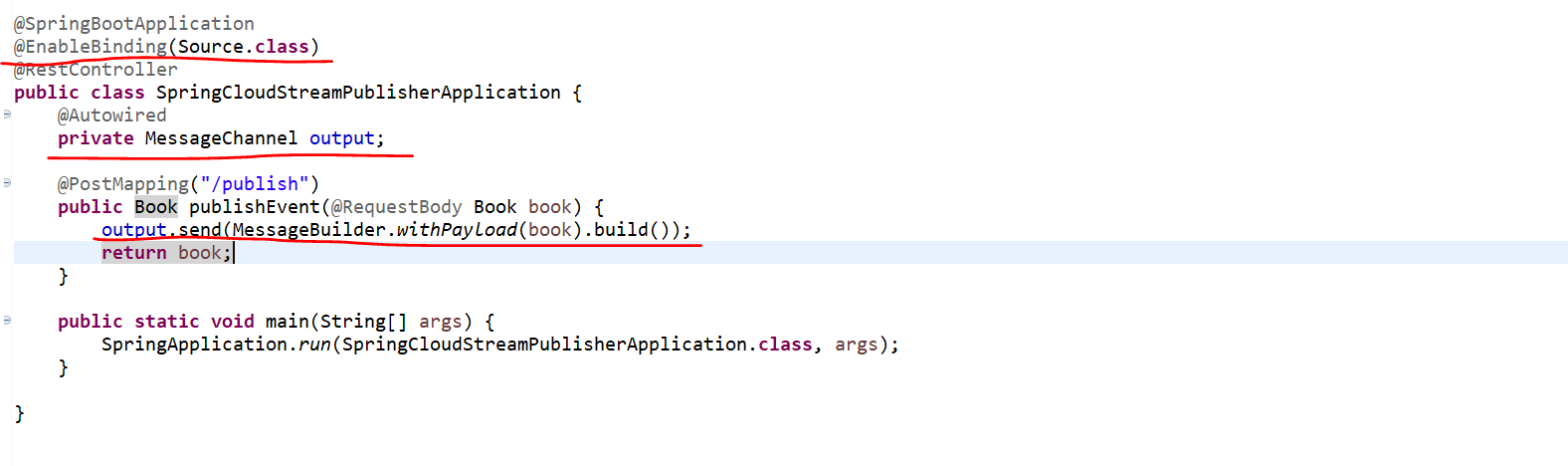
**.\bin\windows\kafka-console-consumer.bat --bootstrap-server localhost:9092 --topic TestTopic**

**Application - spring-cloud-stream-publisher**

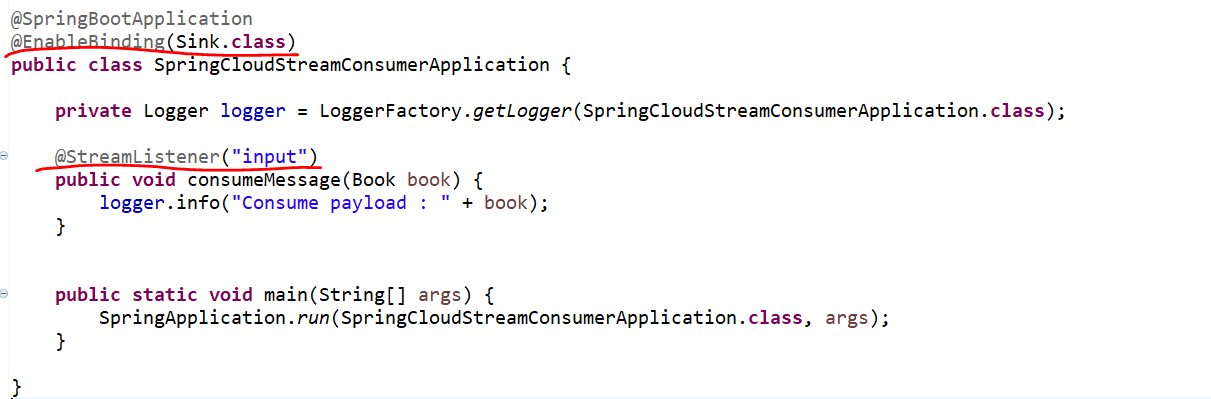
**Dependencies- Cloud Stream, Spring for Apache Kafka, Lombok, Spring Web**

**Application - spring-cloud-stream-consumer**

**Dependencies- Cloud Stream, Spring for Apache Kafka, Lombok, Spring Web**

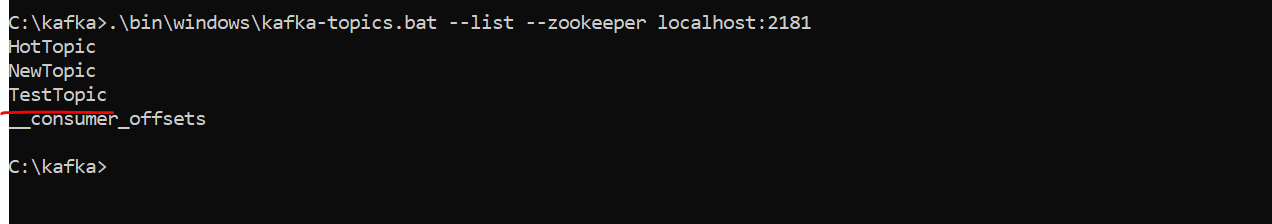


So go to our producer. Go to our main class src main Java. So, open the main class here. We need to annotate @**EnableBinding**. Okay, and inside that we need to specify source. So that source should be imported from org spring framework.cloud.stream messaging at this class. Now adding this annotation, we inform to spring cloud stream. This is what my publisher which will be source. So now what we can do we can create one domain class which will be something like book or employee. Now go to your main class instead of writing one more controller class. I will write it here we can write one method. So, to send the message over the network, we need one Message Channel. So, let's inject the message channel. This is my rest controller. I need to add the URL mapping here. So, this will be post mapping will give some URL called publish. Yeah, so this rest endpoint will publish the message over the message channel.as we are using Apache Kafka here. We need to create one more topic because Kafka consumer and publisher can talk over the topic, isn't it? So, we'll create it later. So, let's build our consumer.



Now go to your consumer. So, go to your main class. Let it open. So, in consumer, we need to add this annotation @**EnableBindings** and here we need to provide the **Sink.class**. Okay, so in publisher section, we added source and in consumer or who is going to subscribe we added sink. Okay,

Now let's write a method. so here we need to add annotation @**StreamListener** and we need to give the channel which will be input. Okay, as well publishing the event we are sending book object. So that's the reason we need to add this book domain in consumer section as well. Okay. So, if you observe here, we created our publisher and subscriber and we are using Kafka. So already we know to work with Kafka. We need to create one topic because on the topic only consumer and publisher both can talk to that topic. Okay, publisher will push the message to topic and consumer will consume that message from topic.



Okay, so we'll use this **TestTopic** topic stream. So, we created publisher, and we created subscriber. Now we need to give the name of our topic so that my publisher can publish the event over this topic and my consumer can consume the message from this topic. Okay. So, what we can do first go to our publisher.

This is consumer go to the publisher and create one Yml file. So, create a file application dot Yml. Here we can write spring cloud. Then stream then you can specify bindings. Okay, and here we need to give the output. And here in output section, we need to give the topic name. Okay, so just give the destination and give the topic name. So, the topic name, let me copy it. **TestTopic** topic. Okay, so now my publisher will publish the book object over this **TestTopic** topic. Now, let's configure same in our consumer section. So here he will output it. He will publish it and in consumer we need to input it. Okay, so let me copy the same. Also here let me change the server port server dot port. I'll give something like 9192 save it now go to your consumer. Here also, let me create one Yml file. Yeah, let me add it and here we need to give the input. Okay, because this is what my consumer now saves it. Now we are good to run our application. What we can do. Let's run our publisher first. It will be open port 9192 then from postman will publish the event.

This is consumer go to the publisher and create one Yml file. So, create a file application dot Yml. Here we can write spring cloud. Then stream then you can specify bindings. Okay, and here we need to give the output and here in output section, we need to give the topic name. Okay, so just give the destination and give the topic name.

**Producer application.yml**



**Consumer application.yml**



**Now what we can do, let's go to our postman and let's build a book object and we'll send it.**

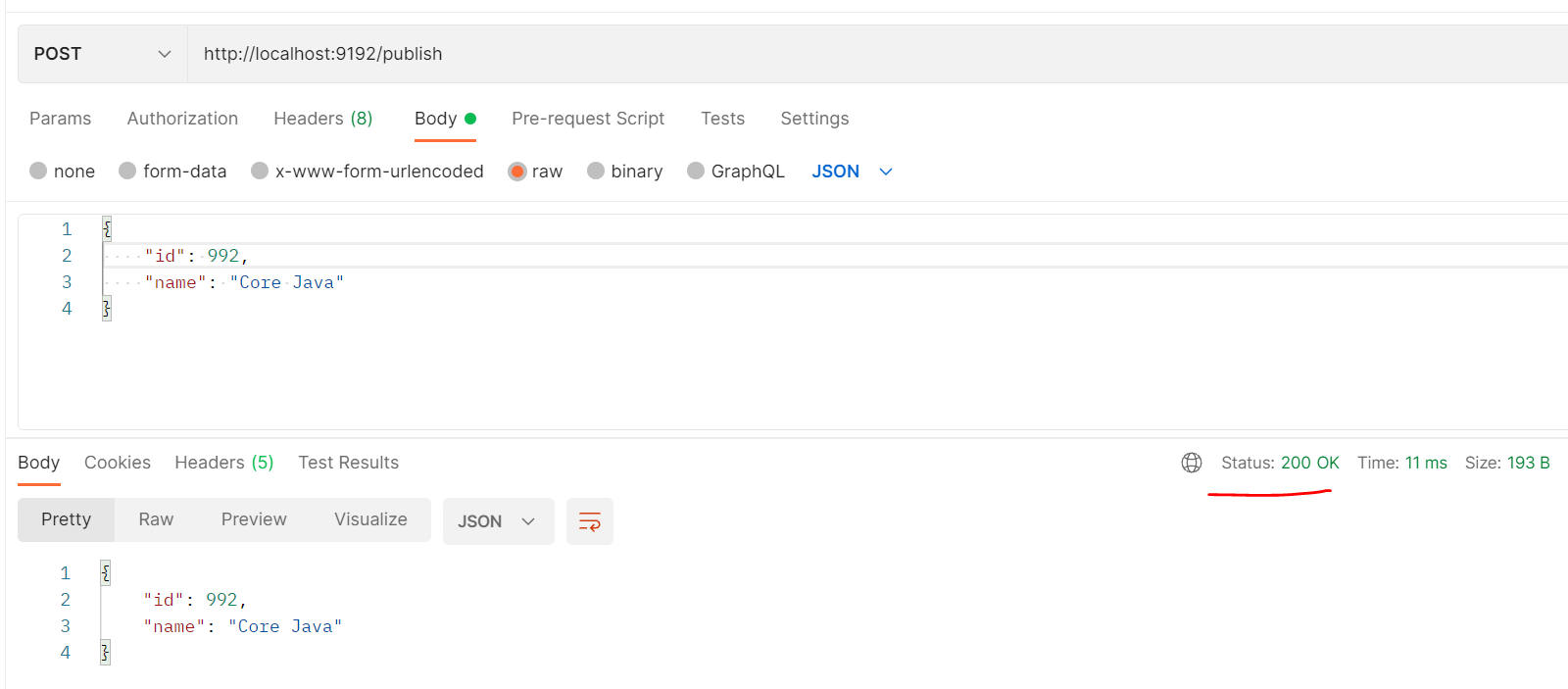
**POST** - <http://localhost:9192/publish>

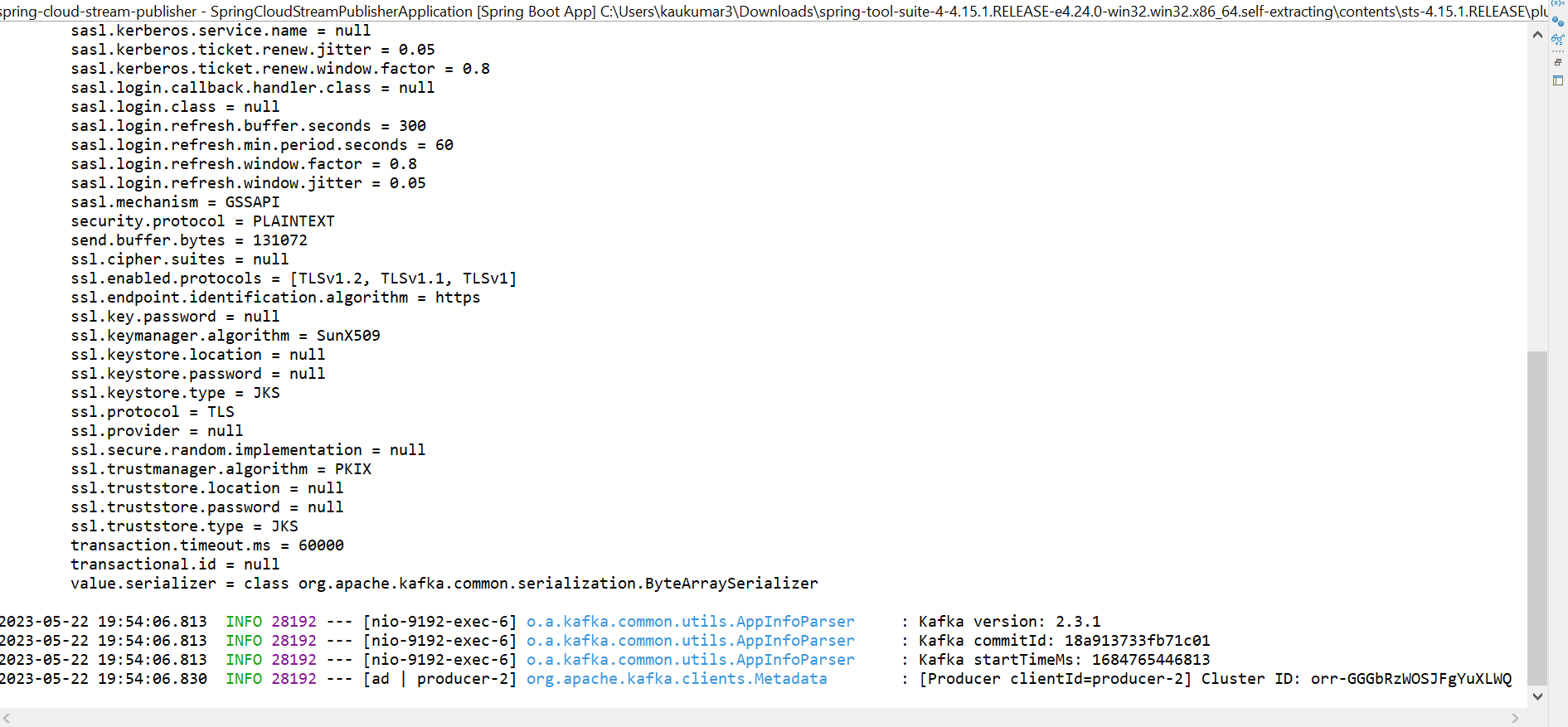
**{**

**"id": 992,**

**"name": "Core Java"**

**}**





**----------------------------------------------------------------------------------------------------------------------------------------**

