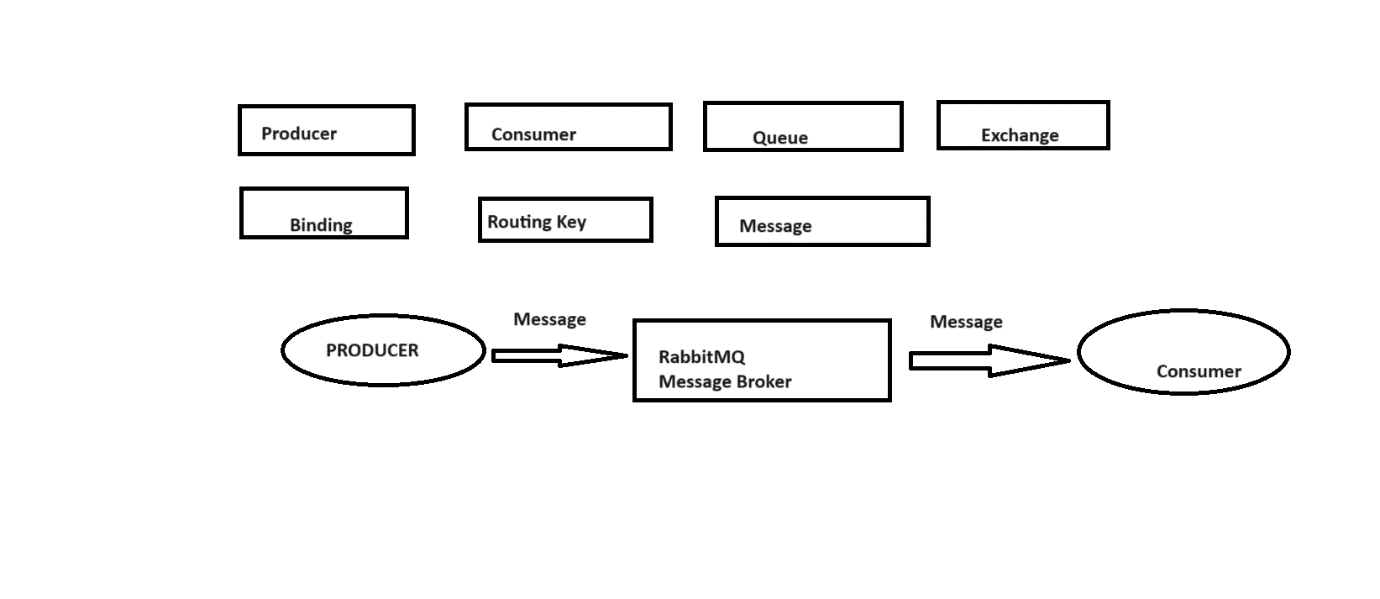
**RabbitMQ Core Concepts and Architecture**



**We will learn about:**

What is Producer?

What is Consumer?

What is Queue?

What is Exchange?

What is Binding?

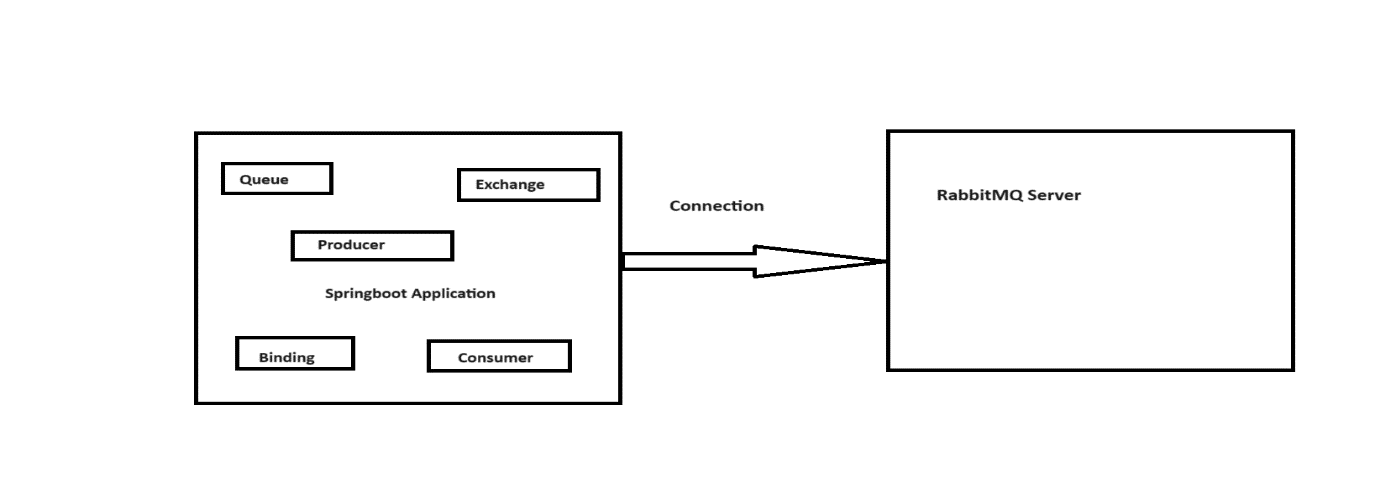
What is Routing key?

What is Message?

**Producer**: It will produce and send the message to the RabbitMQ Broker.

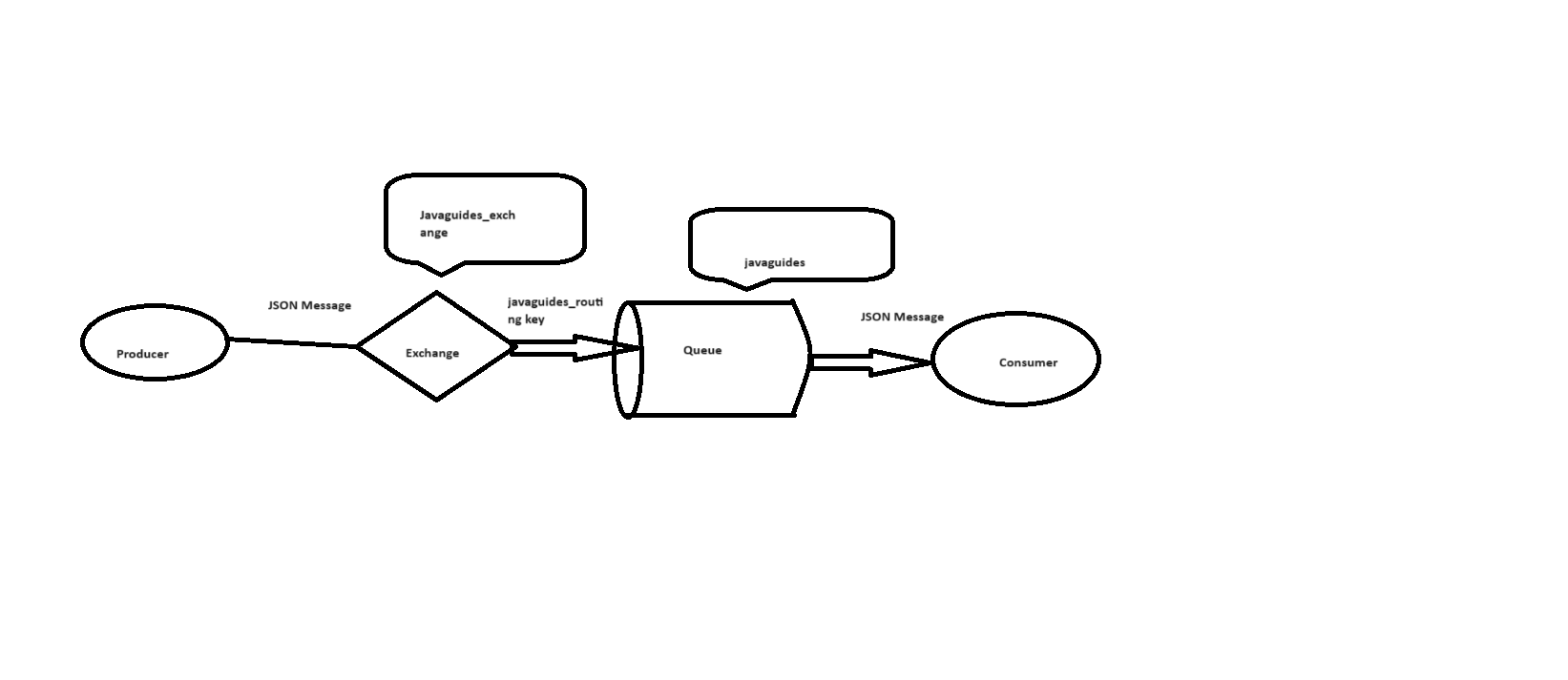
**Consumer**: It will consume the message from the RabbitMQ Broker.

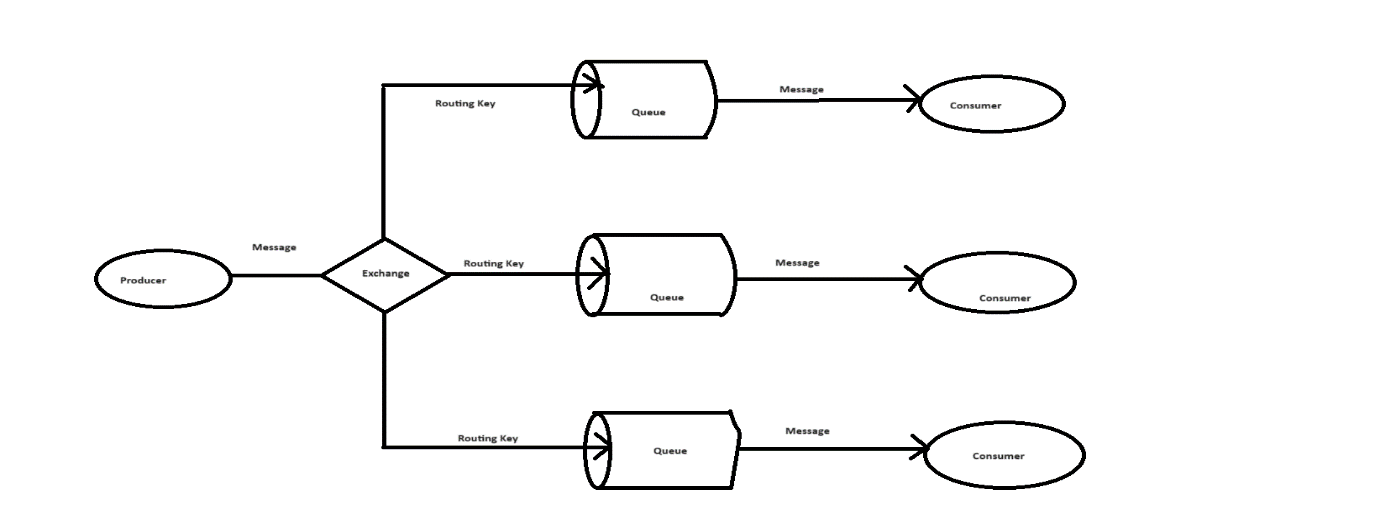
**How to integrate RabbitMQ with Springboot Project using Spring AMQP Library?**

Spring provides Spring AMQP Library which we can use to integrate RabbitMQ into our Springboot project.

**AMQP: Advanced Message Queuing Protocol**

**How to use a message of Type String for the communication between producer and Consumer using a RabbitMQ Broker in our Springboot Application?**

**How to create Multiple Queues and multiple consumers in RabbitMQ Architecture?**

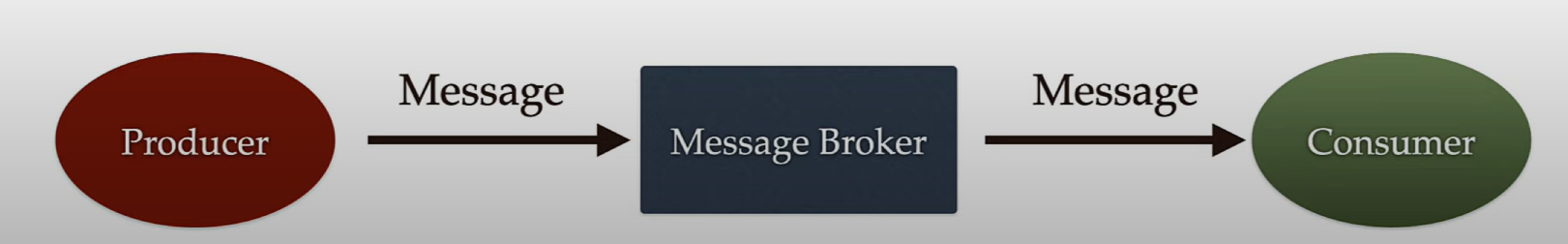


**Ques 1: What is Message Queue?**

**Ans:** Message queuing allows applications to communicate by sending messages to each ither. The message queue provides temporary message storage when the destination program is busy or not connected.

A message queue is made up of a producer, a broker (the message queue software), and a consumer.

A message queue provides an asynchronous communication between applications.



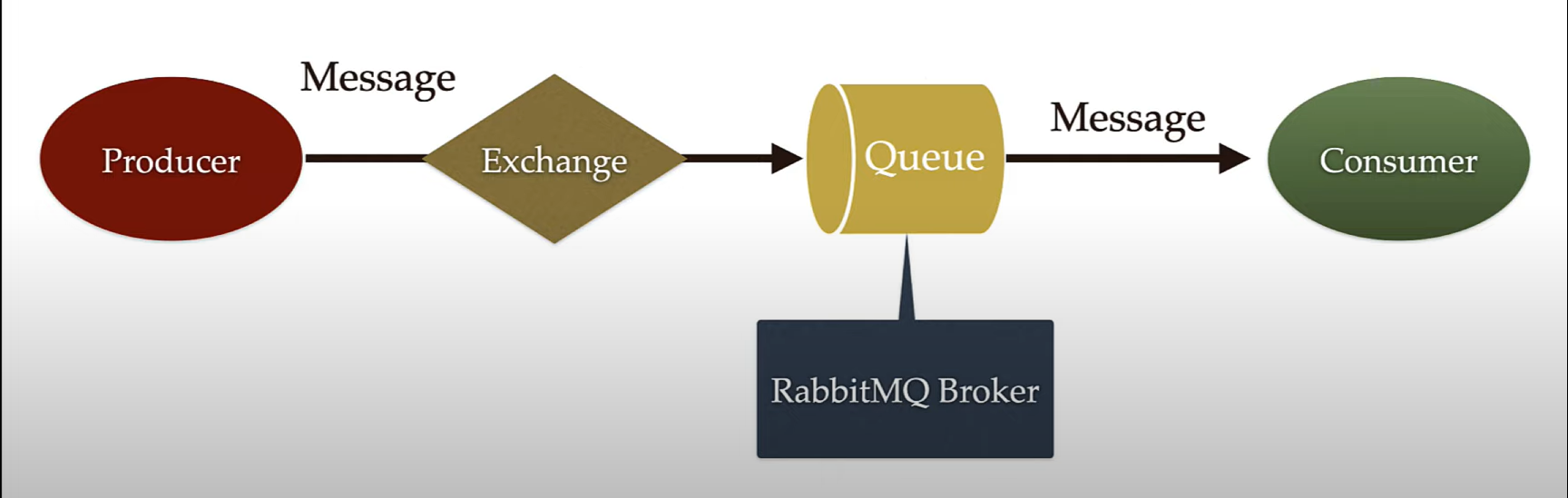
**Ques 2: What is RabbitMQ?**

**Ans:** RabbitMQ is a message queue software (message broker/queue manager) that acts as an intermediary platform where different applications can send and receive messages.

RabbitMQ originally implements the Advanced Message Queuing Protocol (AMQP). But now RabbitMQ also supports several other API protocols such as STOMP, MQTT and HTTP.

**Ques 3: What is Producer and Consumer in RabbitMQ?**

**Ans:** Producer is an application that sends messages to the RabbitMQ broker and Consumer is an application that reads messages from the RabbitMQ broker.



**Producer:** Producer is an application that sends messages. It does not send messages directly to the consumer. It sends messages only to the RabbitMQ broker.

**Consumer:** Consumer is an application that reads messages from the RabbitMQ broker.

**Note: Multiple consumers can subscribe to the RabbitMQ broker.**

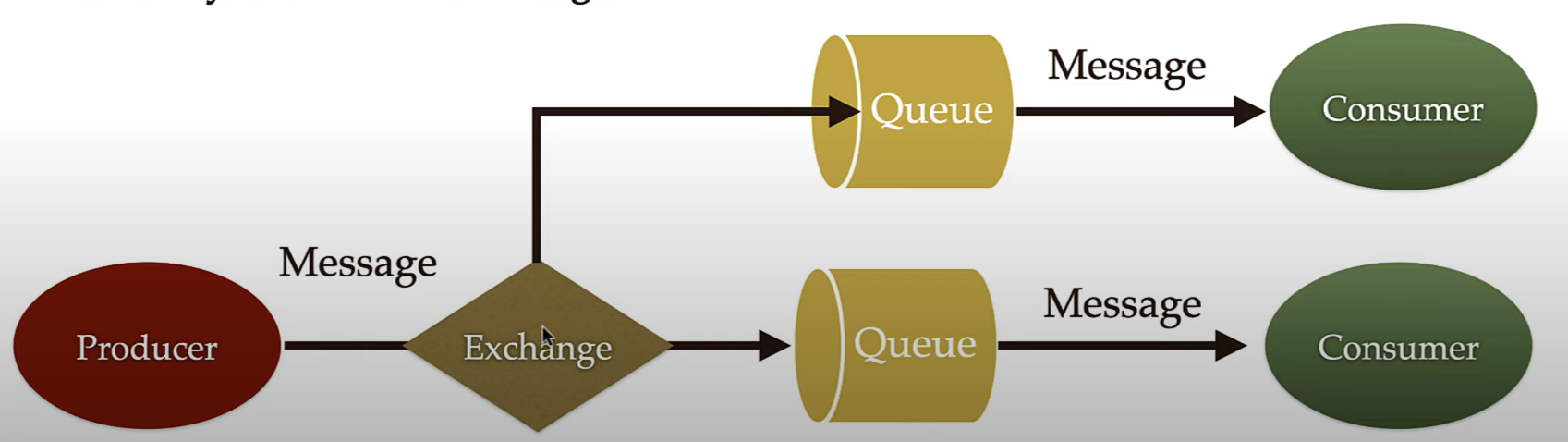
**Queues:** Queue is a buffer or storage in a RabbitMQ broker to store the messages.

The messages are put into a queue by a producer and read from it by a consumer. Once a message is read, it is consumed and removed from the queue. A message can thus only be processed exactly once.

**Message:** Information that is sent from the producer to a consumer through RabbitMQ broker.

It can be of any type like: String format, JSON, Byte Array, Plain text, HTML etc.

**Exchange:** Basically, it acts as an intermediary between the producer and a queue. Instead of sending messages directly to the queue, a producer can send them to an exchange instead. The exchange then sends those messages to one or more queues following a specified set of rules. Thus, the producer does not need to know the queues that eventually receive those messages.

****

**Routing Key:** The routing key is a key that the exchange looks at to decide how to route the message to queues. The routing key is like an address for the message. Using this routing key exchange can recognize the respective message related to the respective queue.

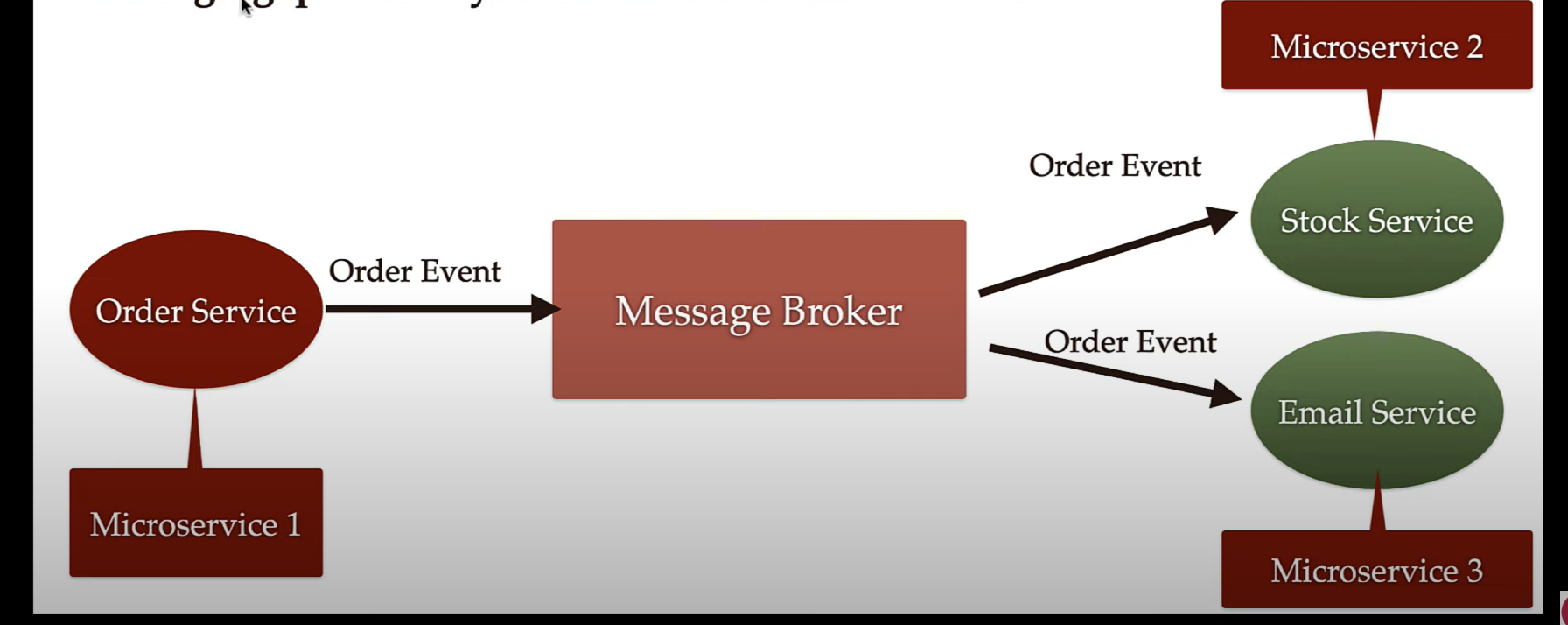
**Binding:** A binding is a link between a queue and an exchange. Binding is done by using routing key.

A diagram of a process

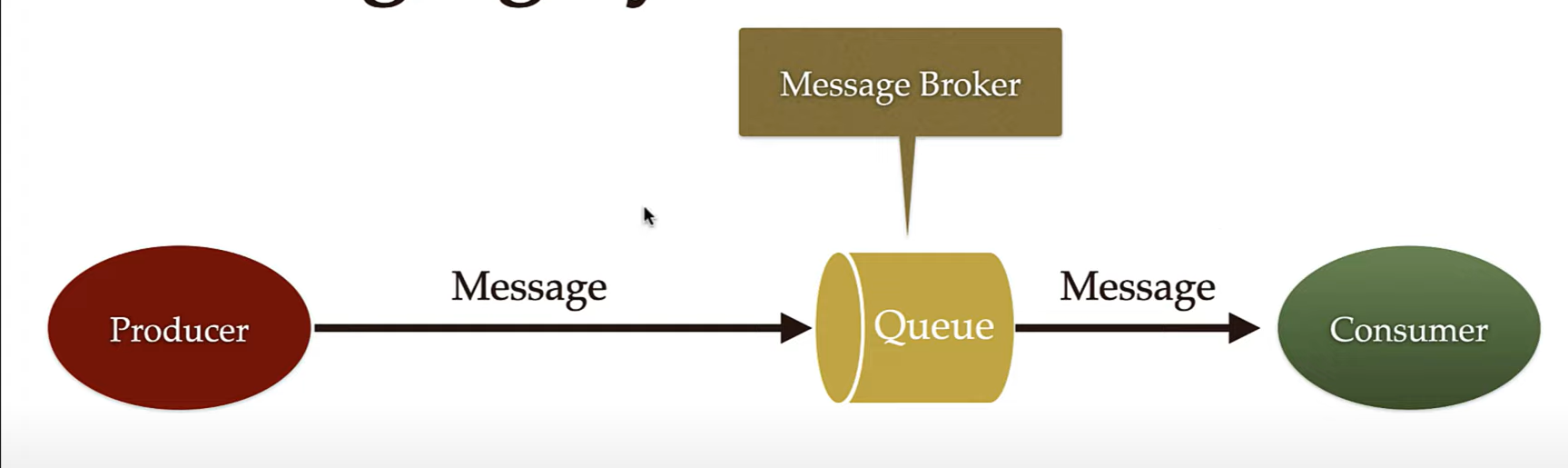
Description automatically generated

**Ques 4: Why we use RabbitMQ in Microservices?**

**Ans:** RabbitMQ is one of the simplest freely available options for implementing messaging queues in your microservices architecture.



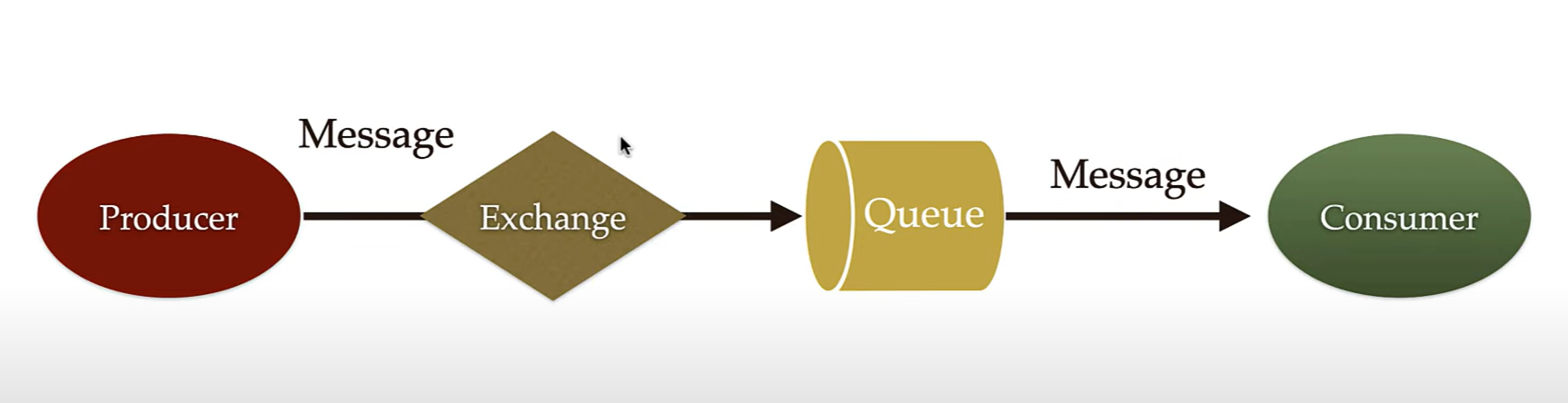
**Typical Messaging System Architecture:** We have Producer and Consumer and between them we have a Message Broker.



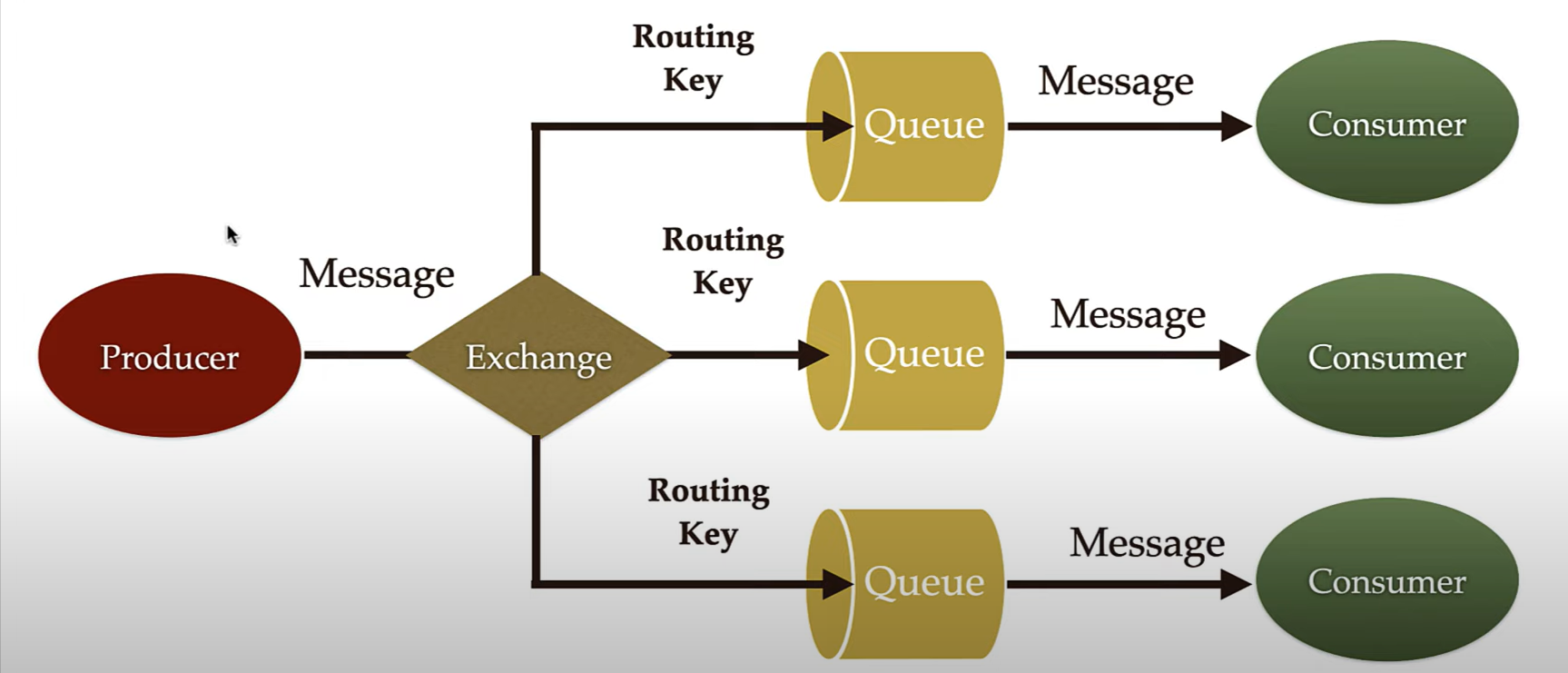
**Simple RabbitMQ Messaging Architecture:** In this RabbitMQ Messaging Architecture there is one more component called as Exchange.

Producer send message to the exchange and Exchange will route the message to the respective Queue with the help of routing key.

**Note:** This is the simple RabbitMQ Architecture but in case of complex application we need to create multiple Queues in RabbitMQ broker.



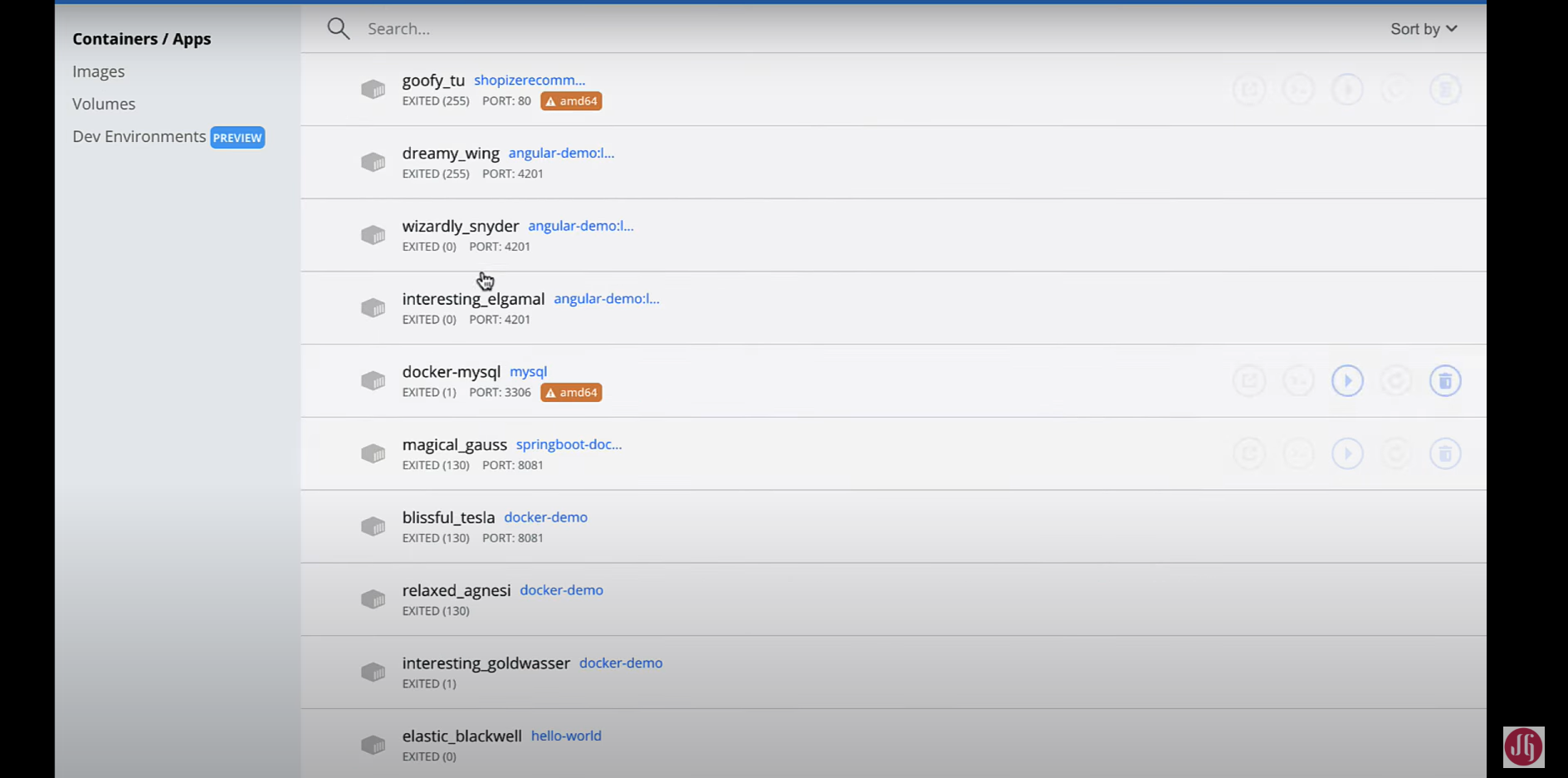
**RabbitMQ Architecture with Multiple Queues:**



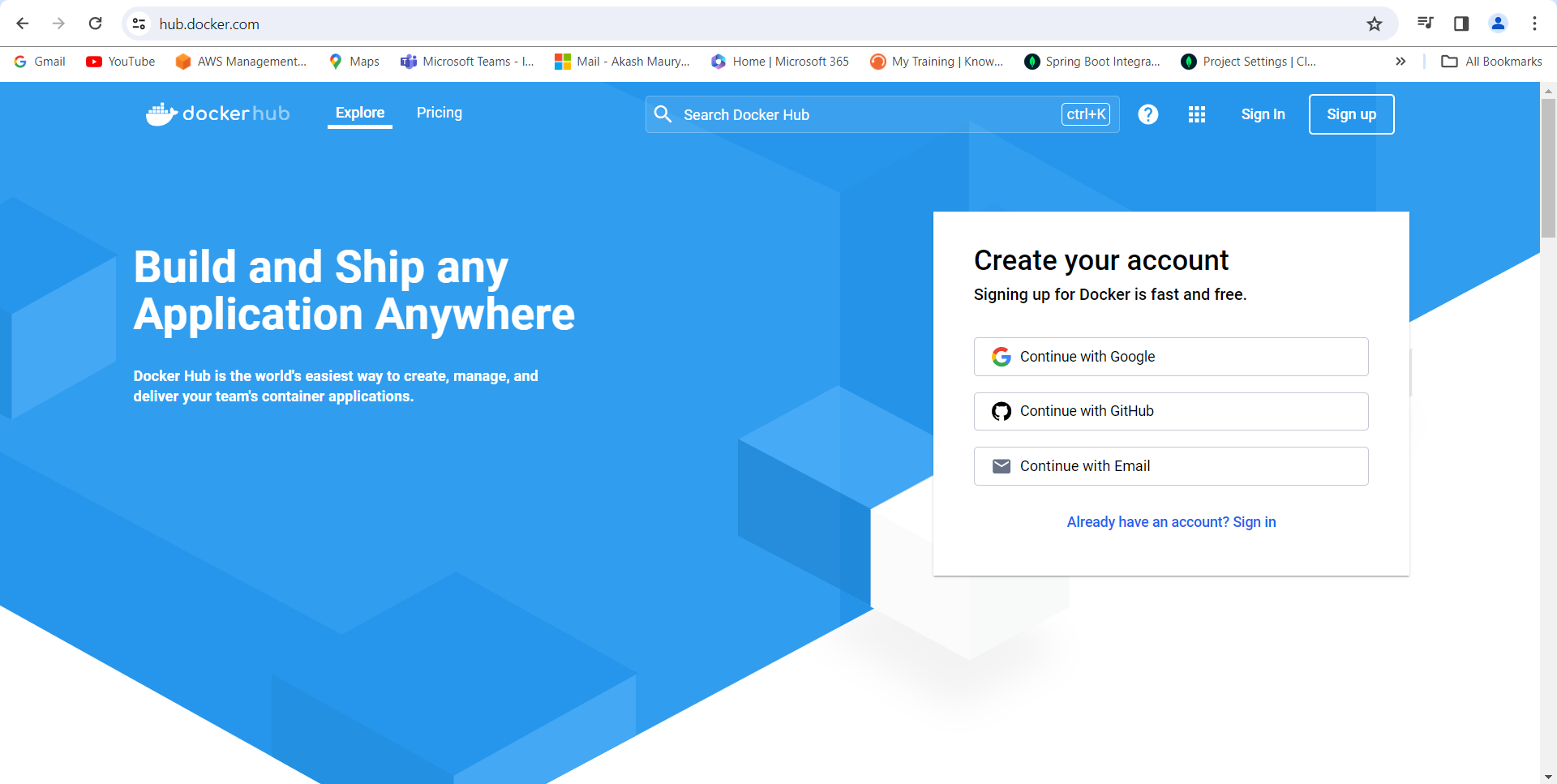
**We can Install and setup our RabbitMQ in windows, Linux and MacOS but Here we are installing RabbitMQ in docker**

**Install and Setup RabbitMQ using Docker**

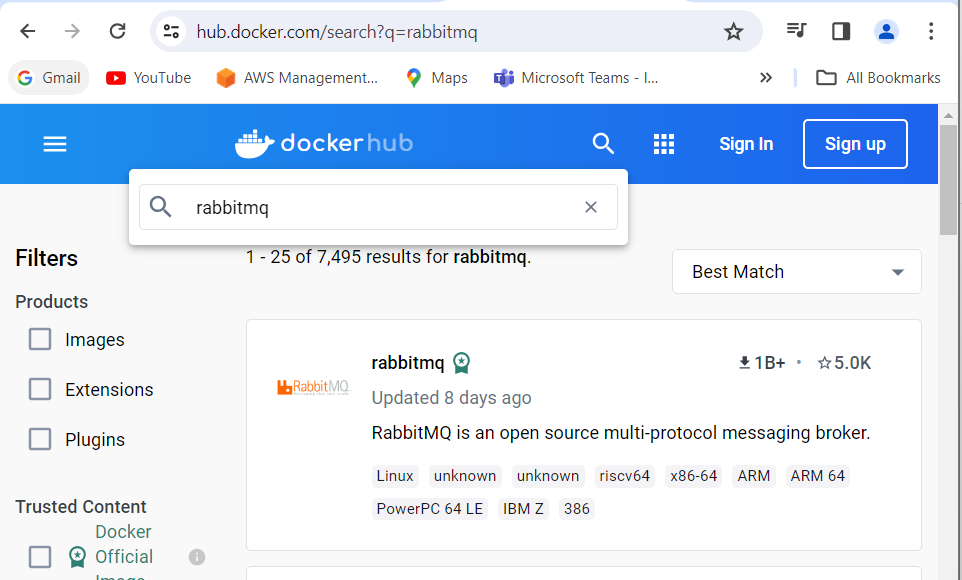
1. **Install and Setup Docker Desktop in your Machine.**
2. **Open the Docker Desktop it will look like this:**



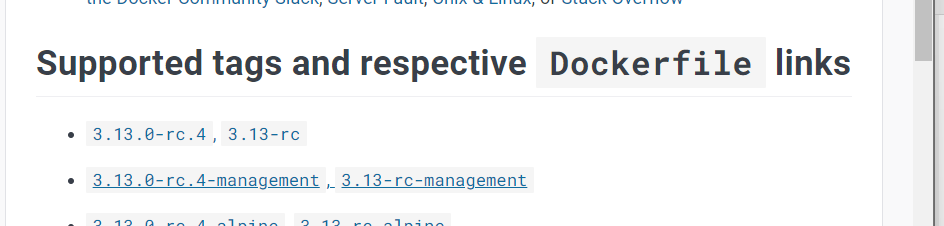
1. **Goto the Images**
2. **Goto the browser and search for docker hub and open the first link it will open the docker official website**



1. **On official website Search the rabbitmq and select the Docker official image**

****

1. **Scroll down and got to the Supported tags and respective Docker file links and select the rabbitmq link which contains management tag**

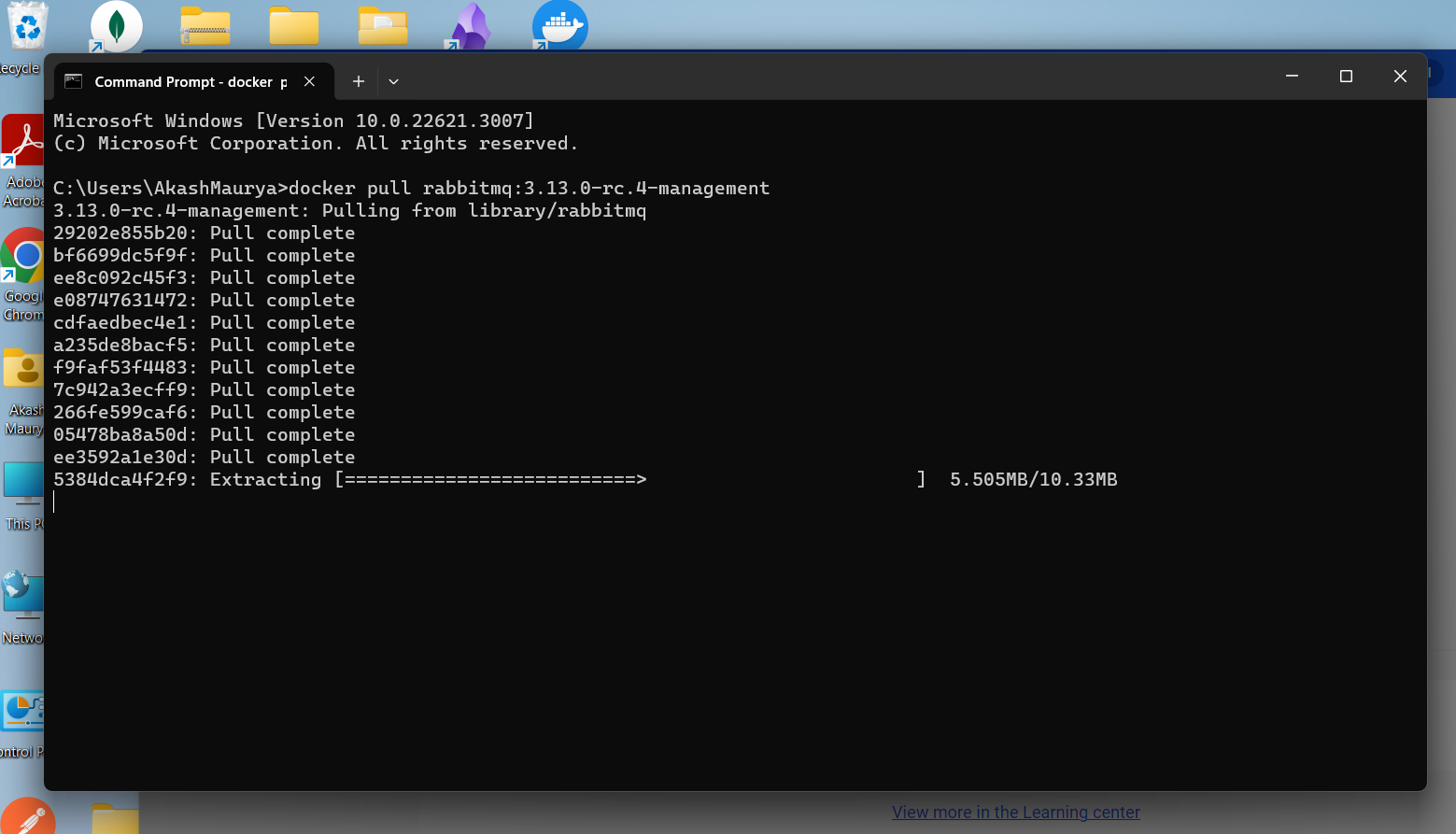




1. **Copy this docker file image name and open the terminal or command prompt and write the command and paste the docker image name**

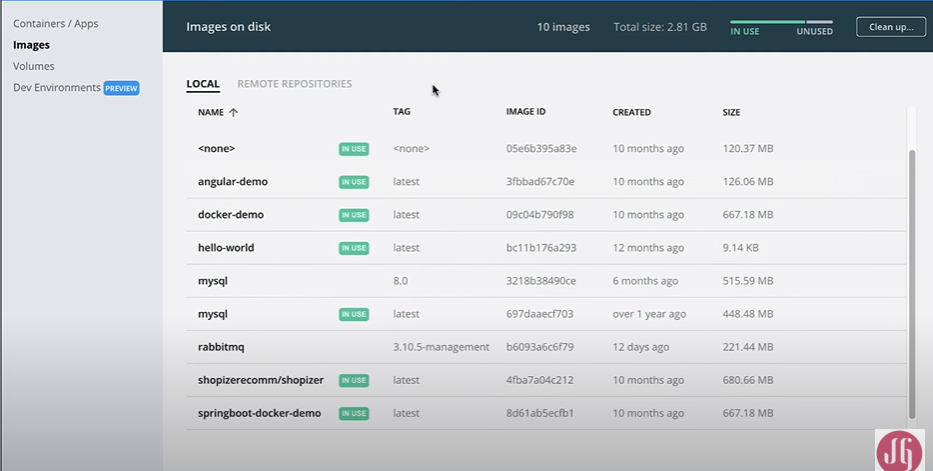
docker pull rabbitmq:[3.13.0-rc.4-management](https://github.com/docker-library/rabbitmq/blob/80011d74327aea3ddd460b189c6533c1f177f48f/3.13-rc/ubuntu/management/Dockerfile)

This command pull the docker image from docker hub and save it into the local docker desktop

 A screenshot of a computer

Description automatically generated

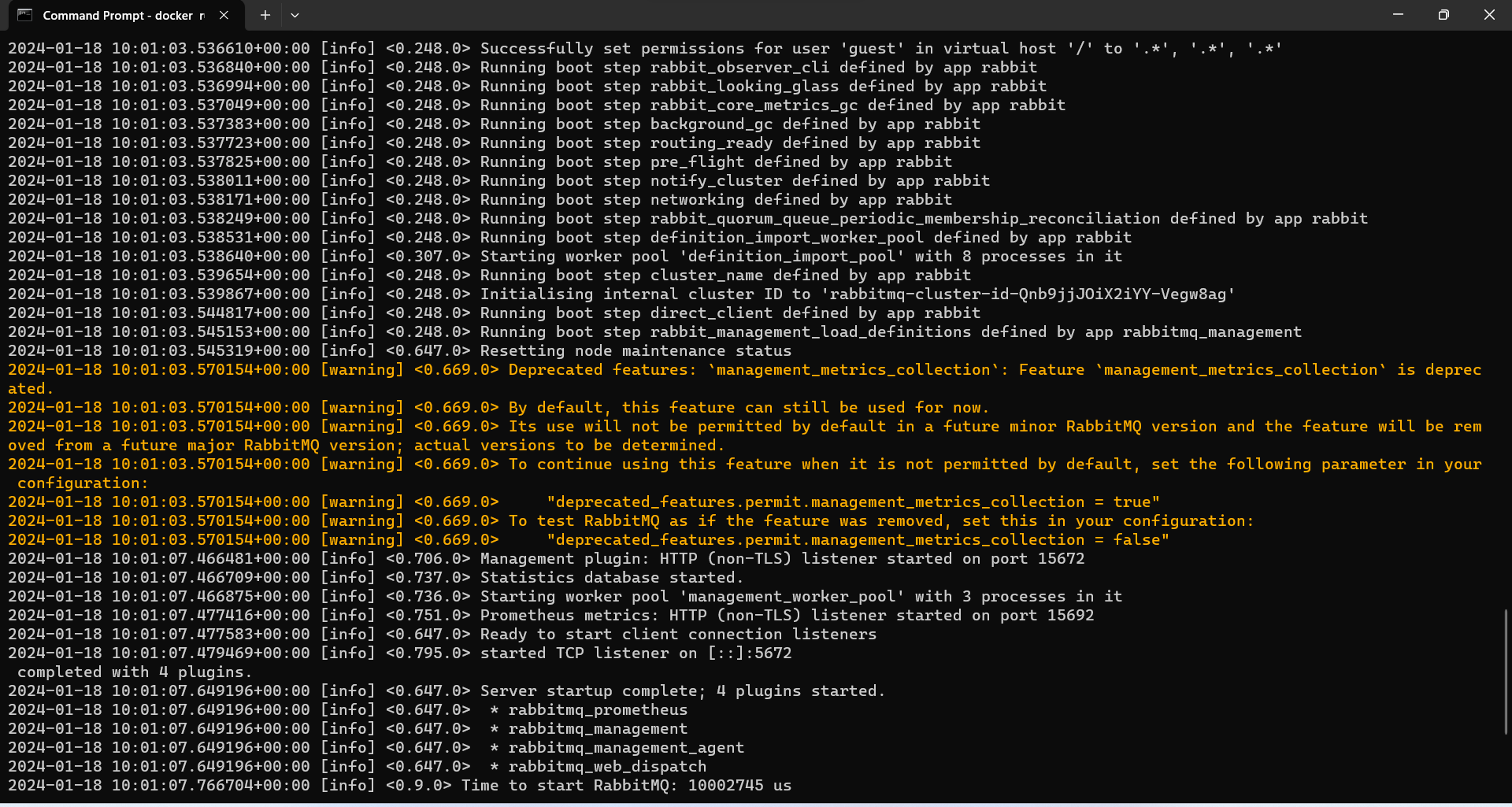
To verify it Goto the Docker desktop-> Images there we can see RabbitMQ



To run the RabbitMQ just type the command: It will start the docker image in docker container

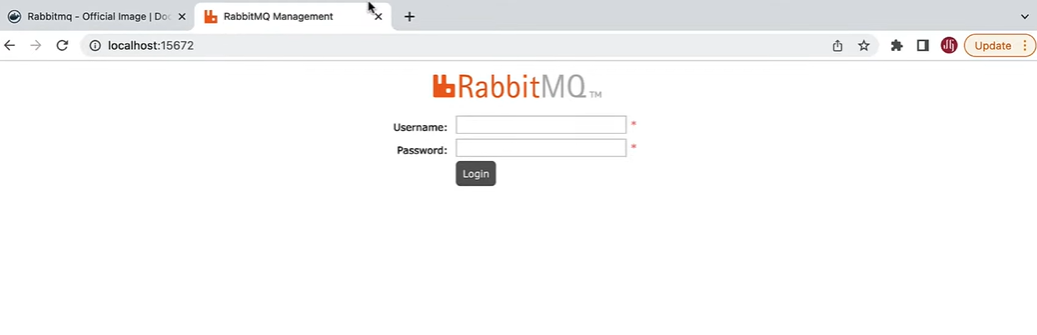
* docker run --rm -it -p 15672:15672 -p 5672:5672 rabbitmq:3-management
* Port 15672 is for RabbitMQ management website or RabbitMQ management plugin
* Port 5672 is used for the RabbitMQ client connections or for connecting our springboot application with the RabbitMQ broker by using RabbitMQ client

Once the command runs successfully



After successfully running the command just open the browser and open the rabbitmq management website

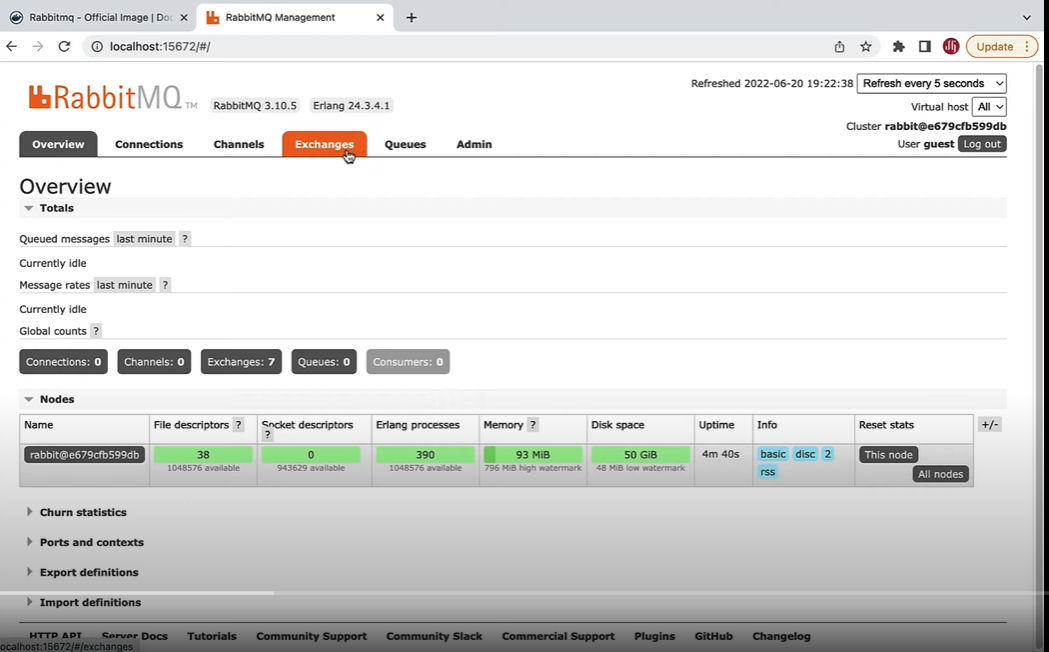
Localhost:15672



Login using default

Username: guest

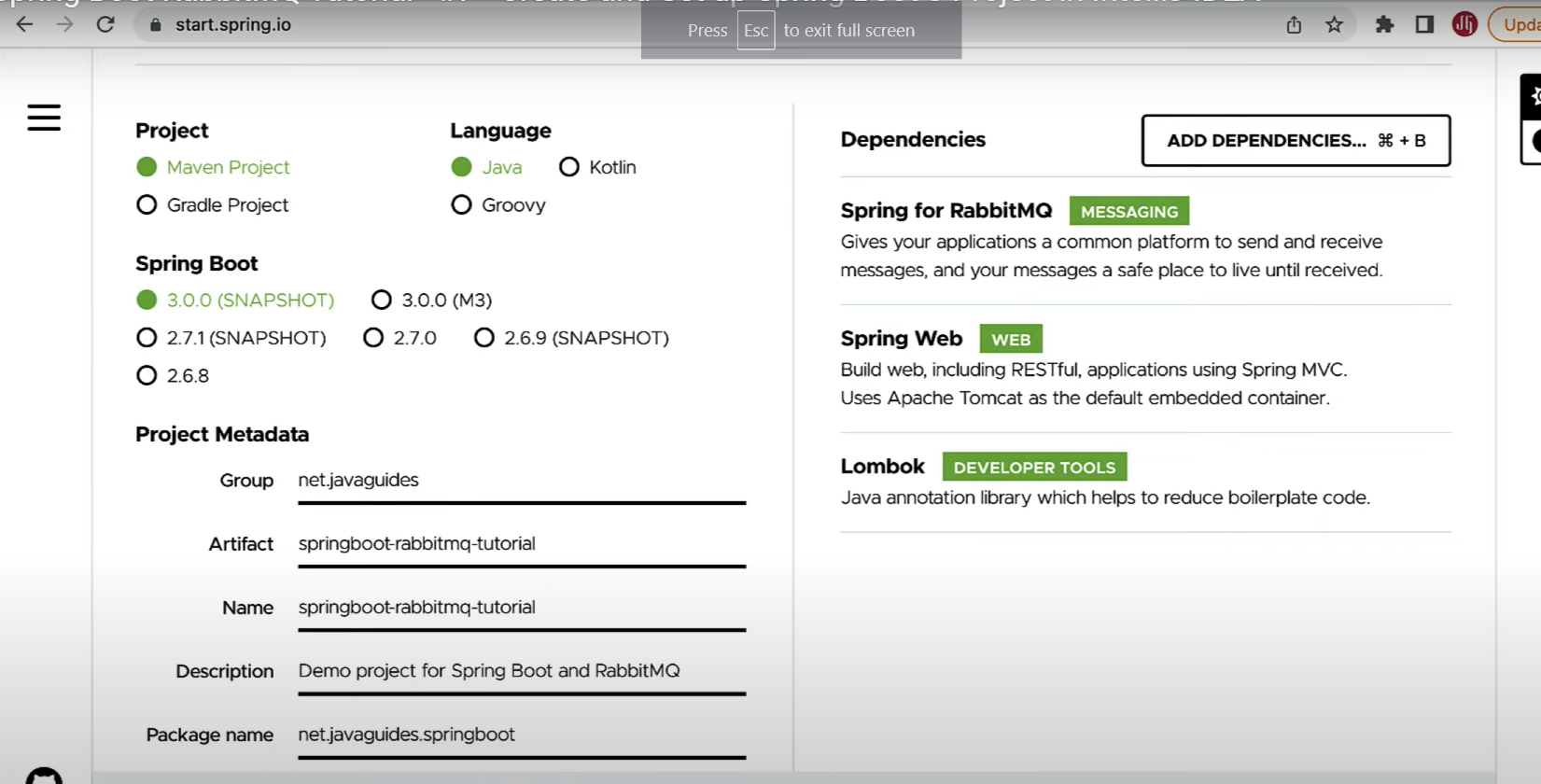
Password: guest



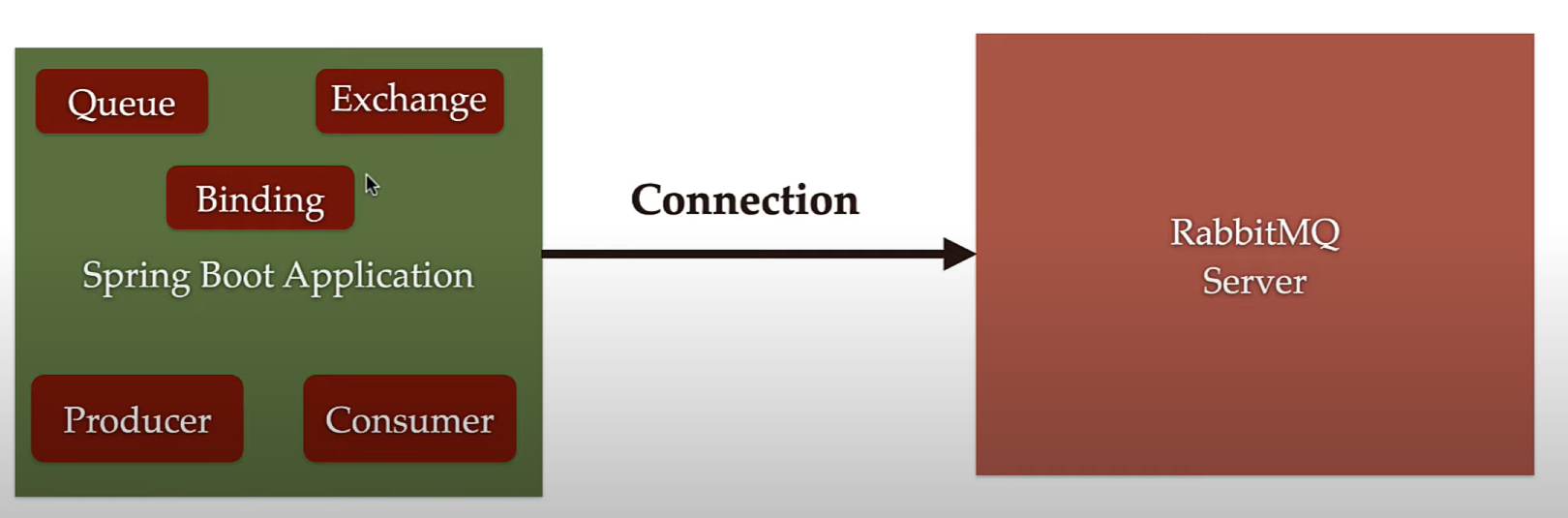
**Create a Spring boot Application.**

* Select a maven project
* Select Java 17
* Add dependencies: S

1. Spring for RabbitMQ
2. Spring web
3. Lombok



How to connect Springboot application with RabbitMQ server which running on a docker container



**Spring boot Autoconfiguration for Spring AMQP(RabbitMQ)**

We get connection to out RabbitMQ broker on port 5672 using the default username and password of ”guest”

Define these properties in a application.properties:

Spring.rabbitmq.host=localhost

Spring.rabbitmq.port=5672

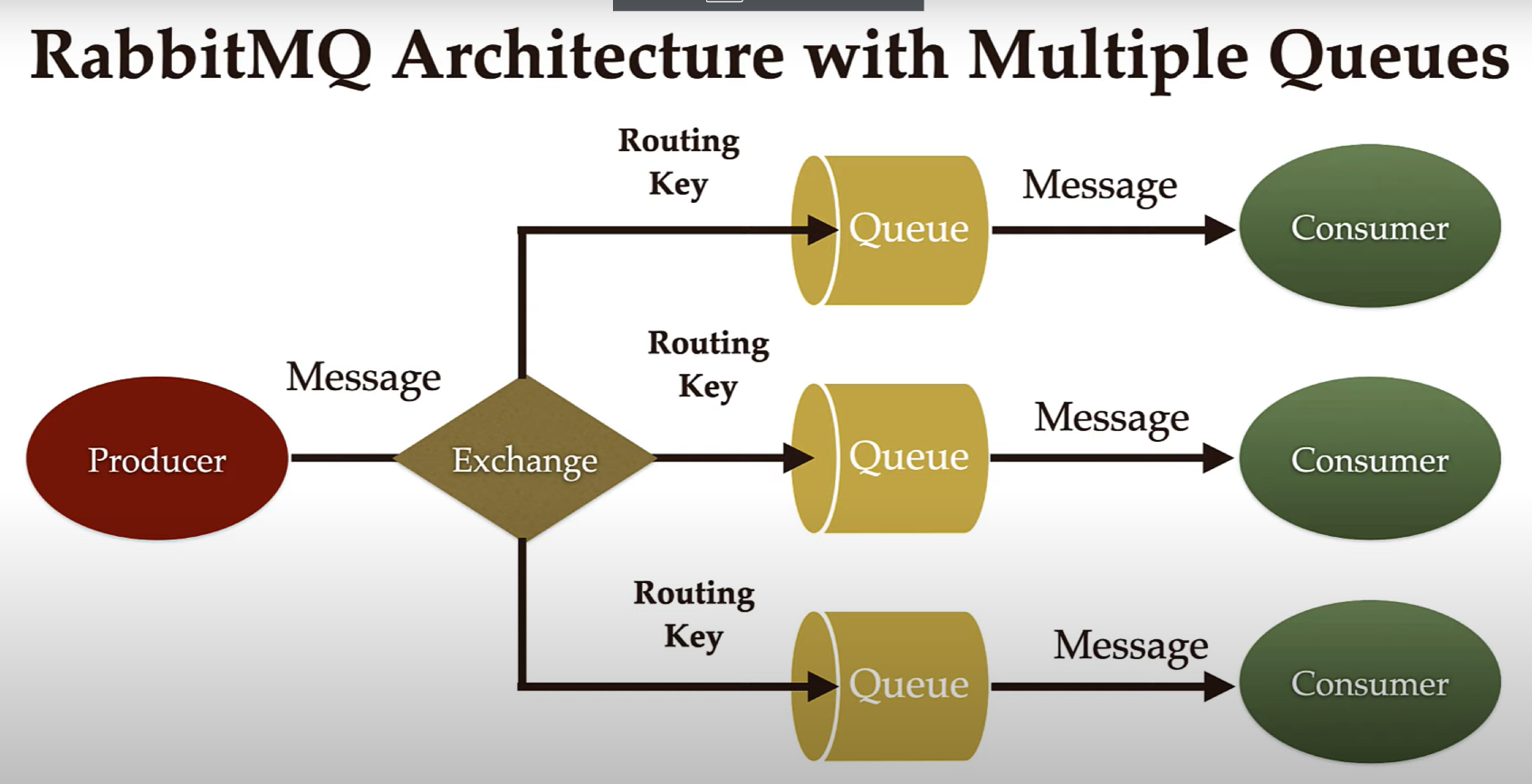
Spring.rabbit.username=guest

Spring.rabbitmq.password=guest

Note**:** In order to connect our Springboot app with RabbitMQ server we use above properties these properties are default values. Springboot autoconfiguration use these properties to automatically connect our application with RabbitMQ server. It means that we don’t need to define these properties in application.properties file because springboot automatically use these default values to connect our springboot application with the broker on port 5672 if our server is running locally in our development environment.

**But these default values works whenever we install RabbitMQ locally in a development environment. Let’s say if our RabbitMQ broker is running on a different machine then we have to provide these properties and that machine Ip address or the hostname in application.properties file. So make sure whenever you deploy your rabbitmq broker or a server on production or in any different environment so make sure that as per the environment we have to provide the hostname port, username and password.**

**How to Configure RabbitMQ Queue, Exchange and Binding in our Springboot Application.**

****

In-order to configure RabbitMQ in our application we need to create different component in our springboot application.

For Example:

RabbitMQ Queue

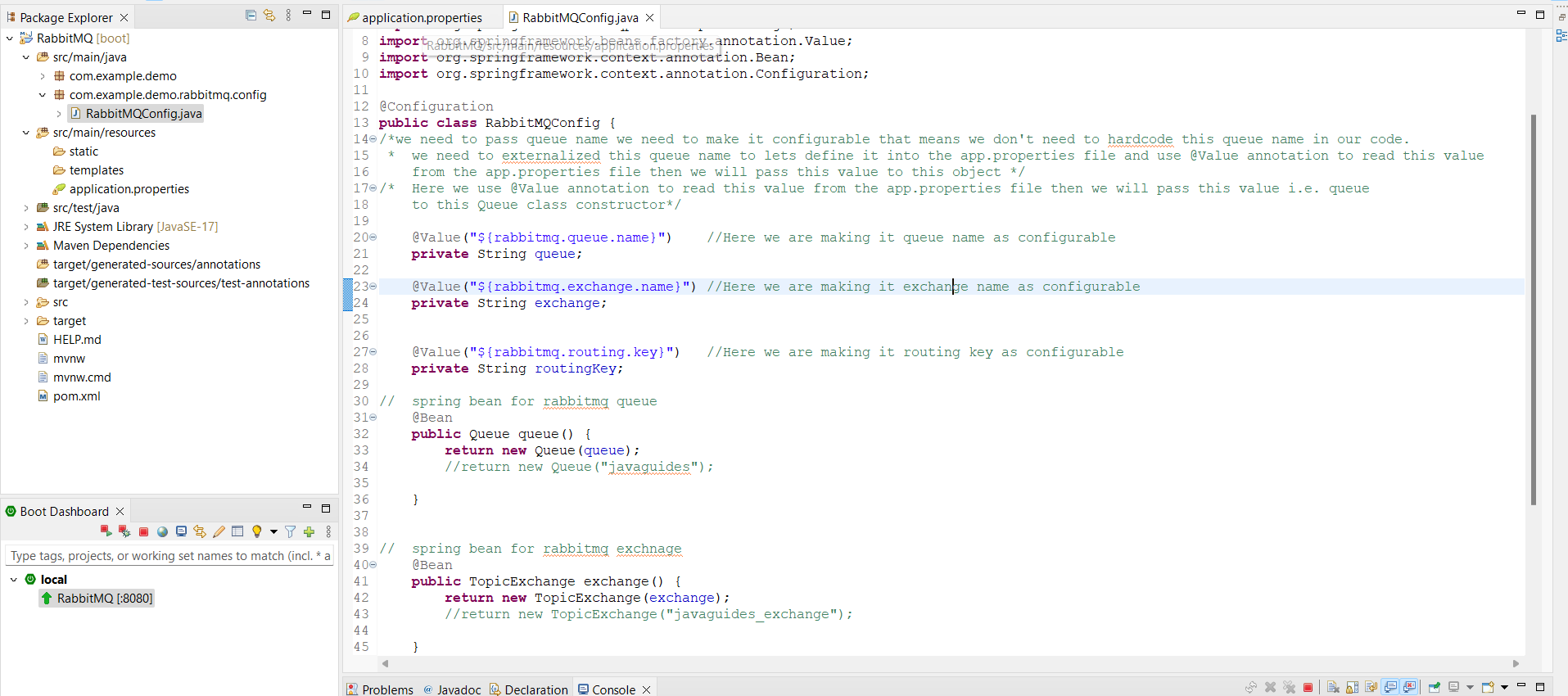
RabbitMQ Exchange

Binding between Queue and Exchange

In Application

1. Create a Configuration Package

* Create a class RabbitMQConfig and make this class as a spring java based configuration by using @Configuration Annotation. Within this class we will define all the spring beans
* Spring Bean to configure for RabbitMQ Queue
* Spring bean to configure for RabbitMQ Exchange
* Spring Bean to configure for RabbitMQ Binding



Apart from above 3 beans we need to configure some more couple of beans which are Infrastructure bean that are required our spring boot application to work with rabbitMQ broker but spring boot autoconfiguration automatically configure those 3 beans for us we don't need to explicitly create these 3 beans for these class.

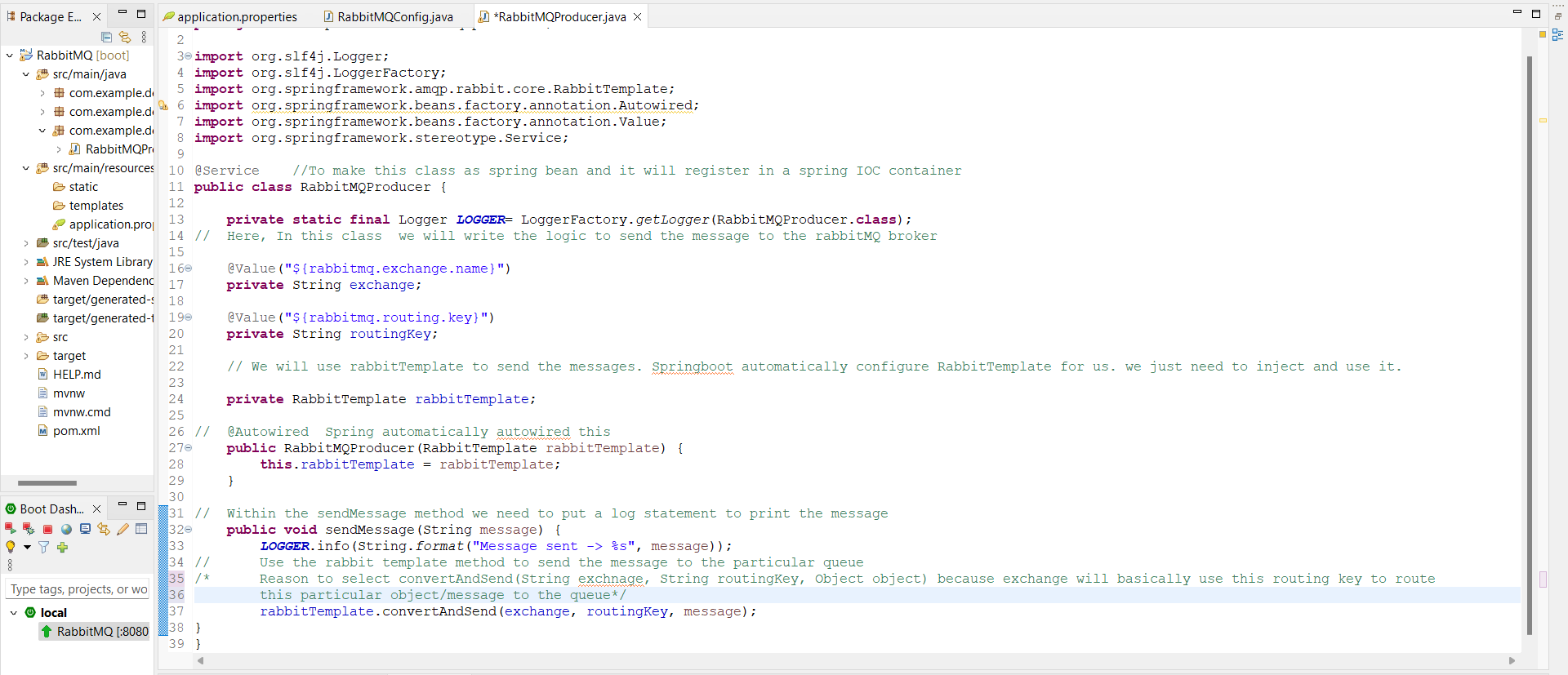
* Bean for ConnectionFactory
* Bean for RabbitTemplate
* Bean for RabbitAdmin

A screenshot of a computer

Description automatically generated

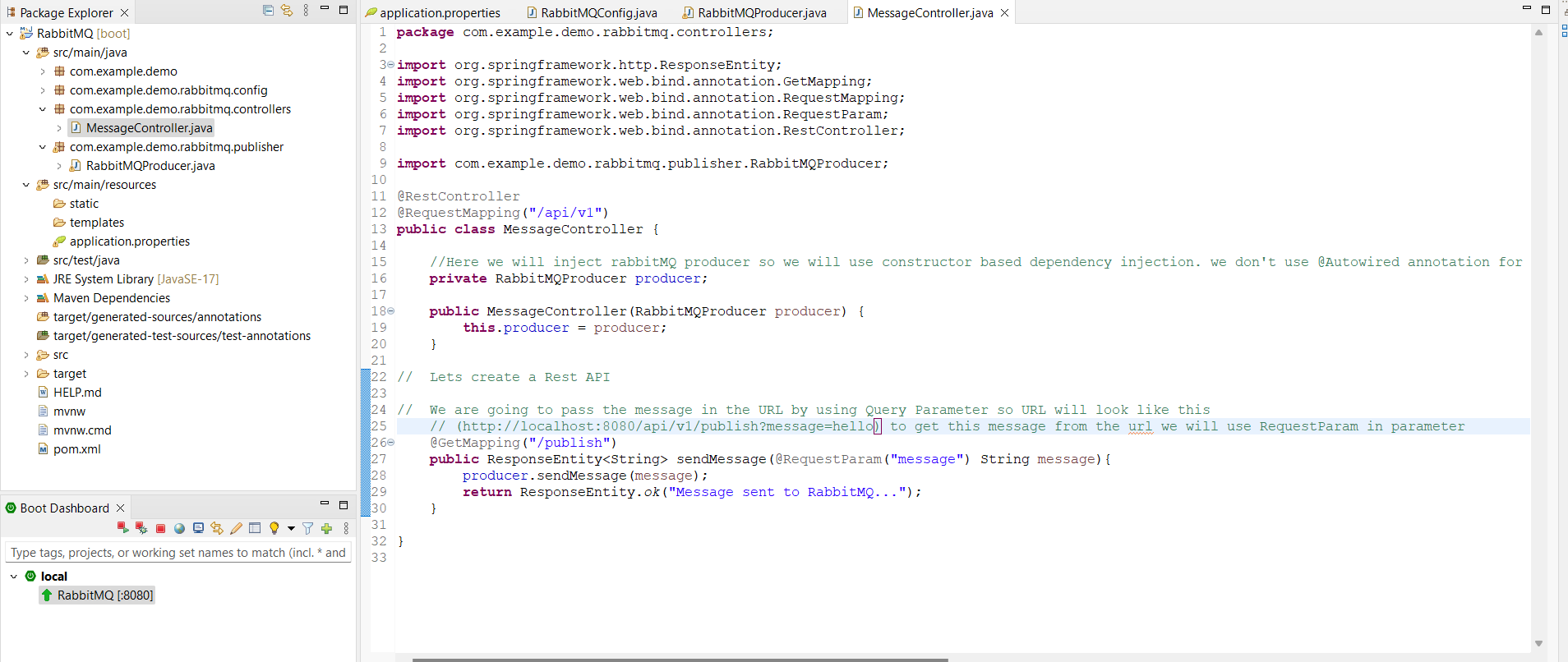
1. Create a Publisher Package to keep all the RabbitMQ Producers

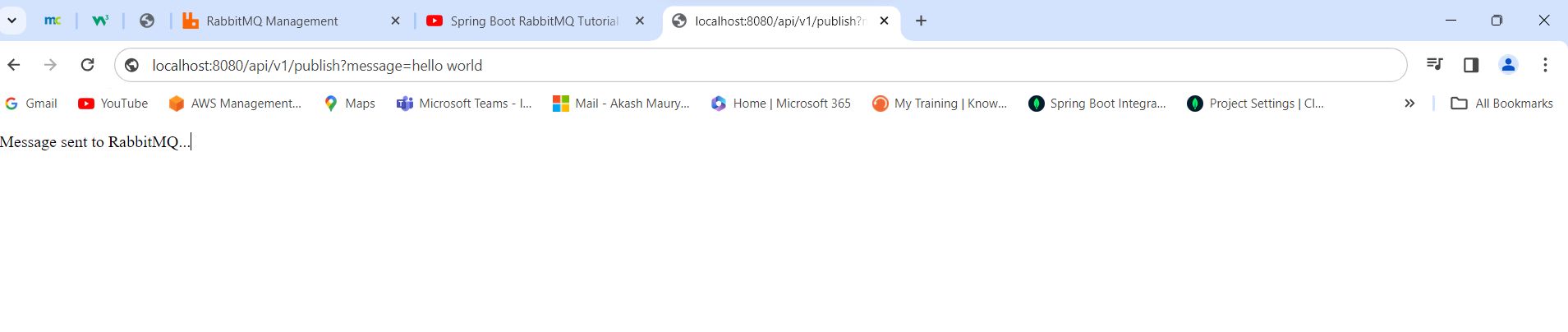
* Create a producer



1. Create a Controller package to keep all the Spring MVC controllers

* Create a Message Controller class to publis the message.



* Run the application and open brower and type this URL : [http://localhost:8080/api/v1/publish?message=hello%20world](http://localhost:8080/api/v1/publish?message=hello%20world)
* Check the RabbitMQ Browser Queues and Exchanges message is stored in the Queues or not?

It is stored in the Queue

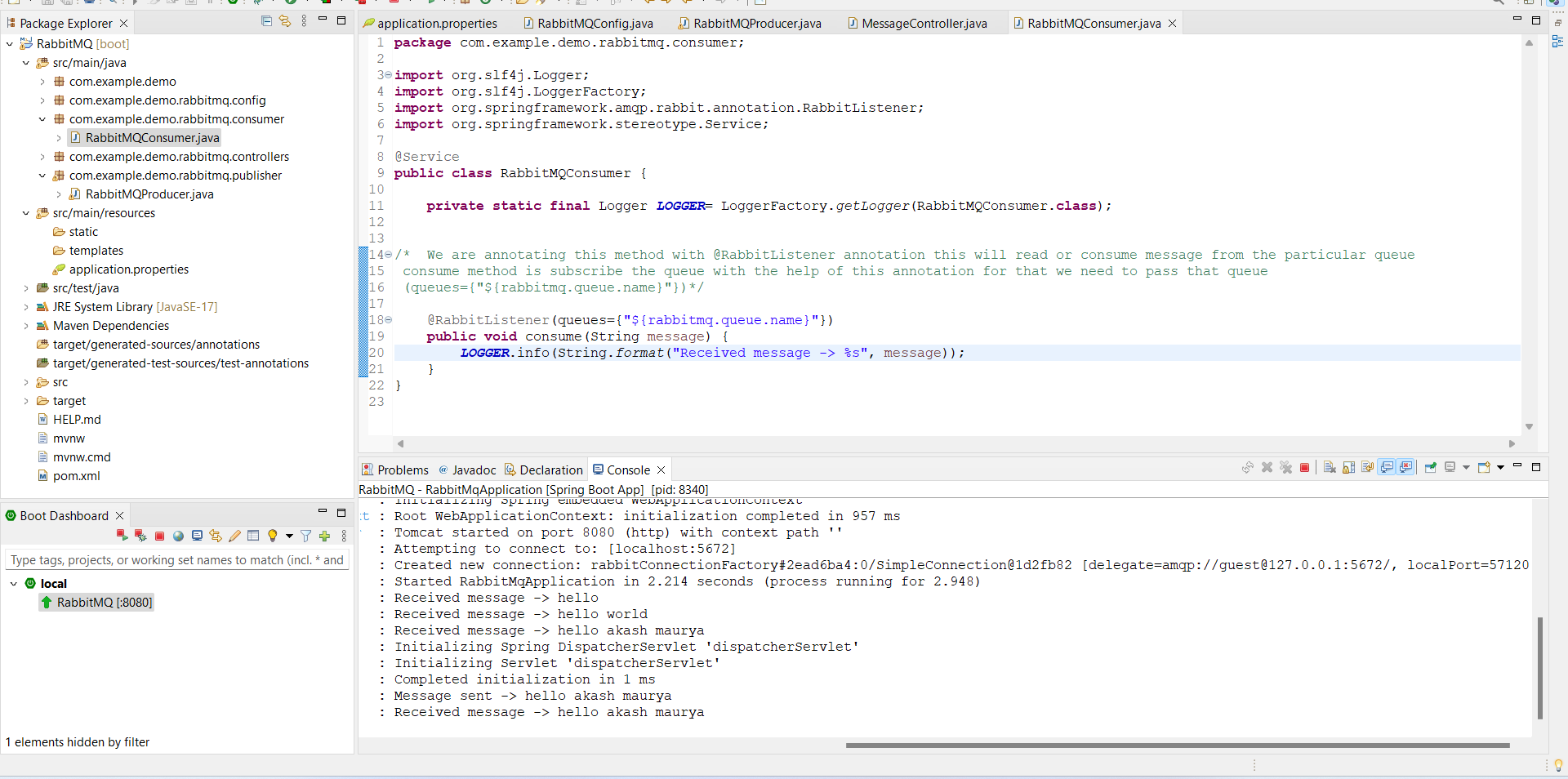
A screenshot of a computer

Description automatically generated

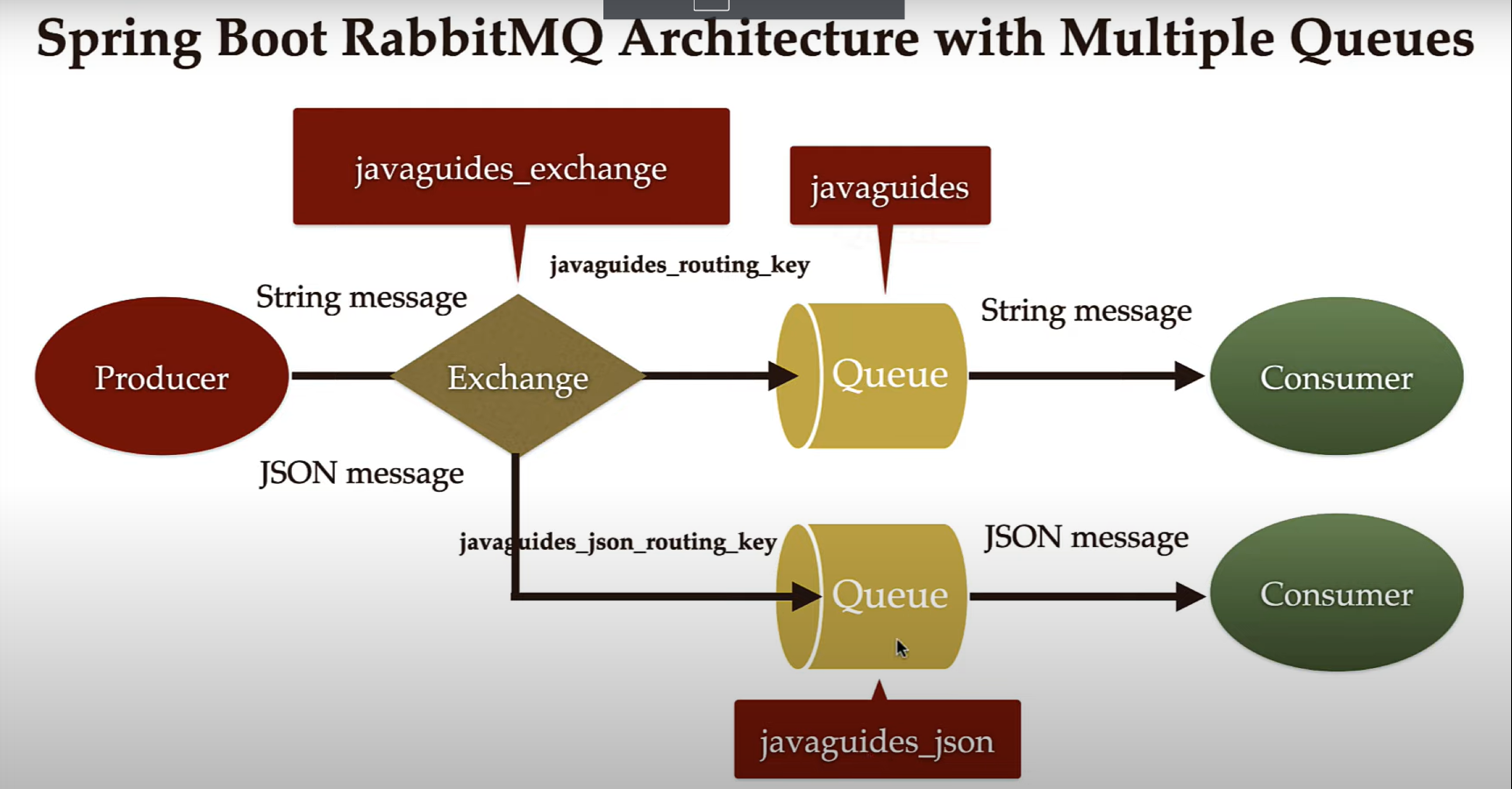
A screenshot of a computer

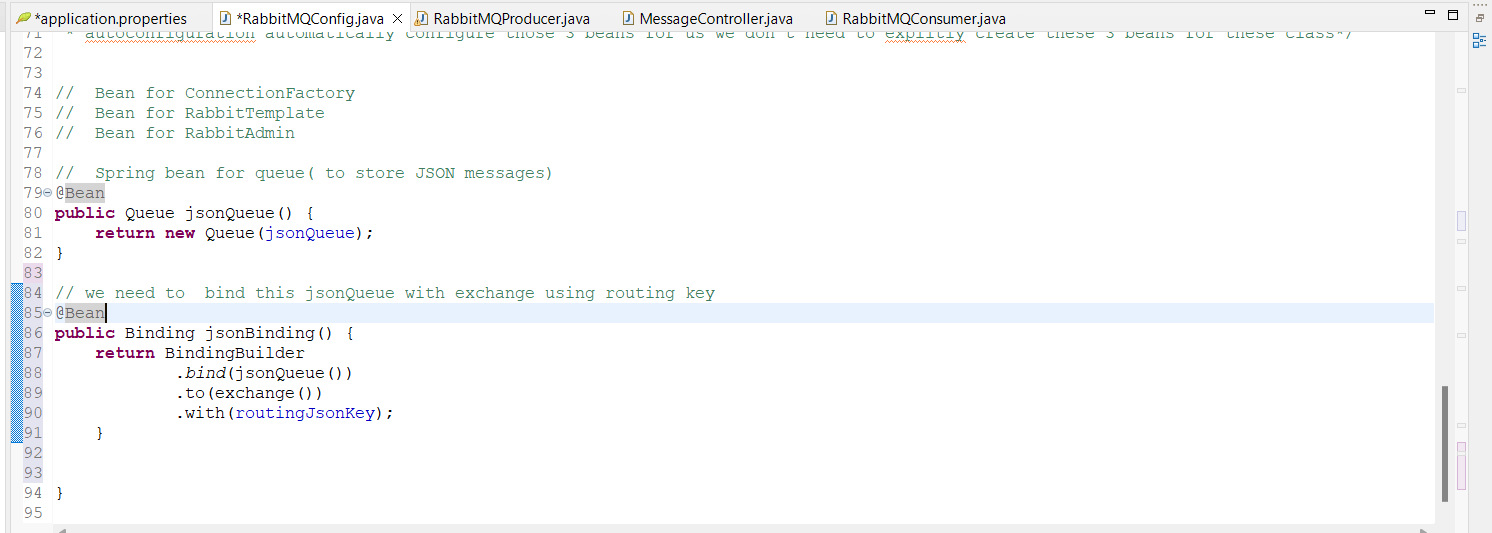
Description automatically generated

1. Create a consumer Package to keep all the RabbitMQ Consumers

* Create a Consumer class 

How to Configure RabbitMQ to use JSON Message for the communication between Producer and Consumer or Create POJO class to serialize/Deserialize

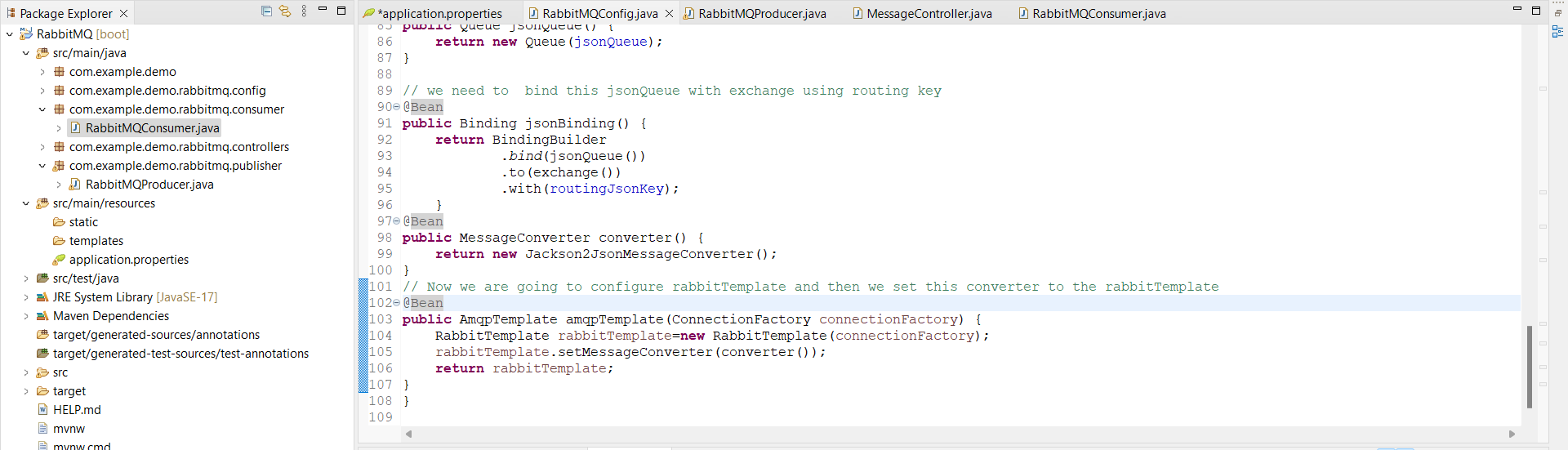


In above architecture, we are going to create one more Queue (javaguides\_json) which stores the JSON messages and we are going to bind this Queue with Exchange using the routing key (javaguides\_json\_routing\_key) 

After this we need to create a spring RabbitTemplate to send Json messages earlier we are using the default rabbitTemplate which is provided by Spring autoconfiguration but we have to create a rabbitTemplate which should support sending JSON messages.

Basically, we are going to set a JSON message converter to the rabbitTemplate so that rabbitTemplate can support sending JSON messages

Now, lets create a spring bean for the message converter



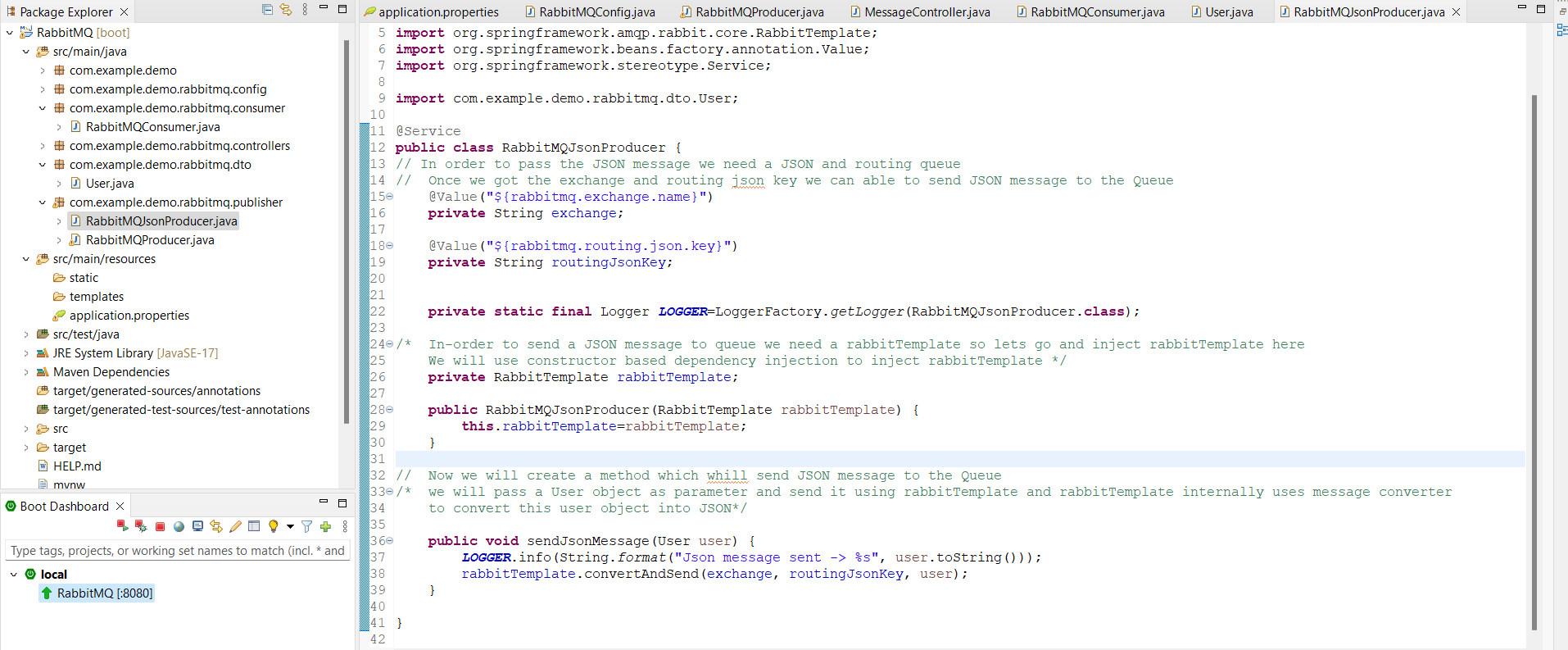
Now RabbitTemplate will support the JSON message for the communication.

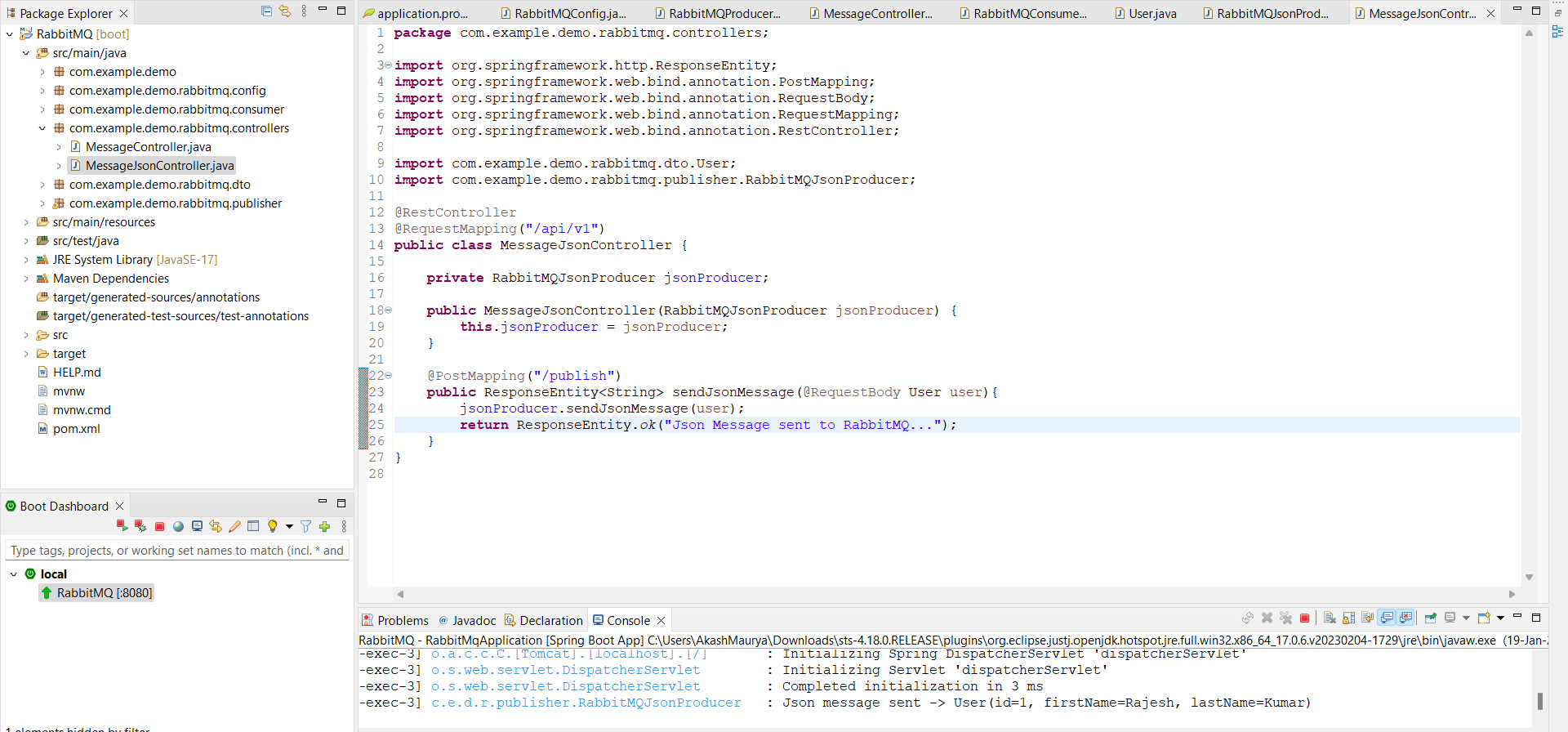
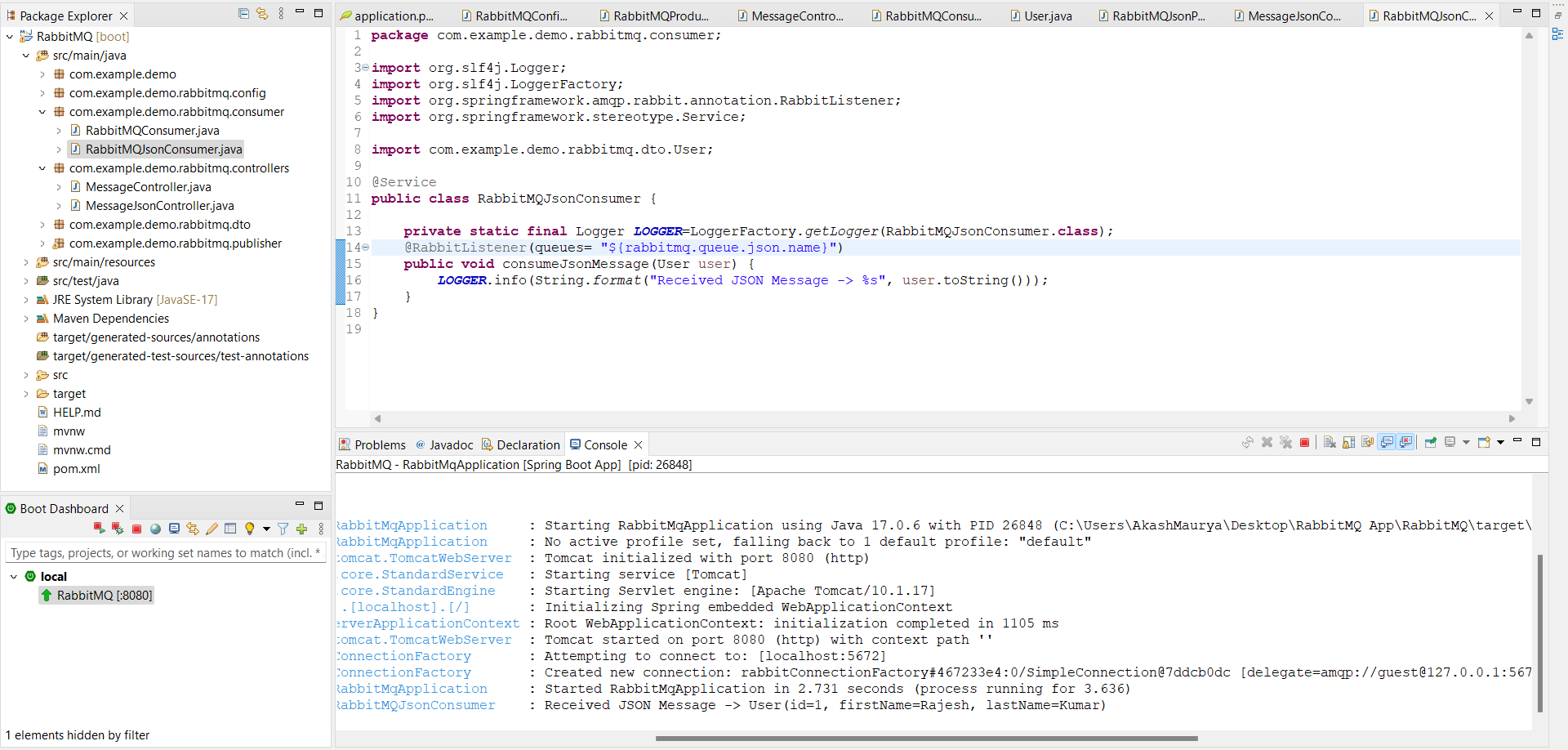
1. Now we will create DTO class for the serialization and Deserialization

Goto DTO package and create a class User and define few instance variables in that class. A screenshot of a computer

Description automatically generated

1. Now we will create a RabbitMQ producer to send a JSON Message to the exchange and exchange will use the routing key to route that message to the rabbitmq queue.



1. Now, we will create a RestAPI that will send a JSON message to this producer and this producer will send this message to exchange and the exchange will send this message to the respective Queue using the routing key. 
2. Now we will write the RabbitMQConsumer to receive or consume this JSON Message 

END…..