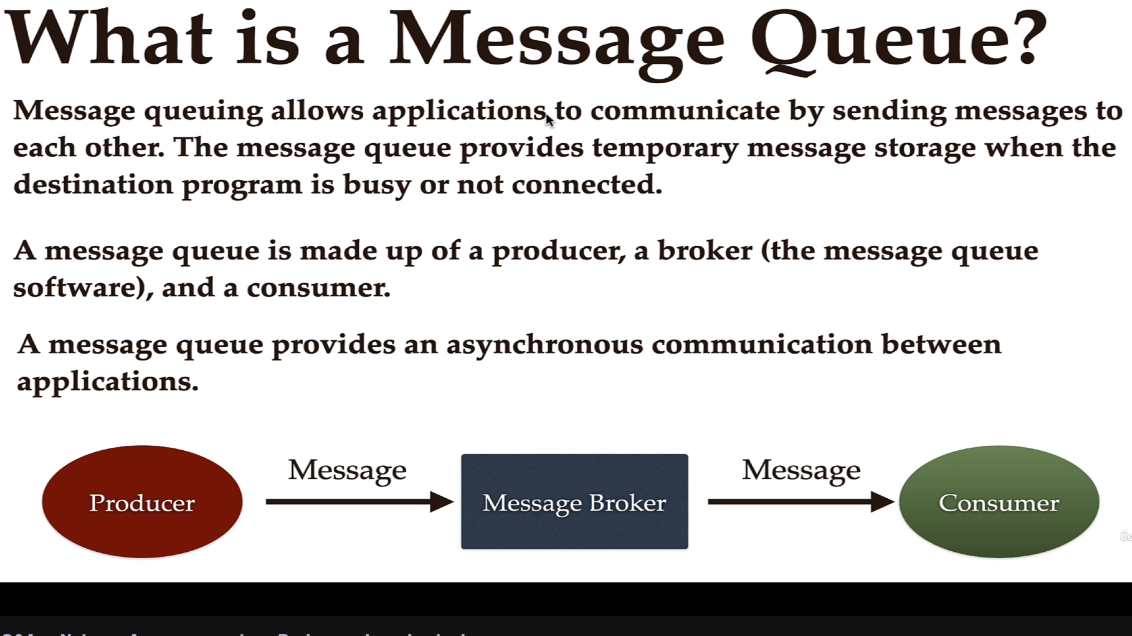
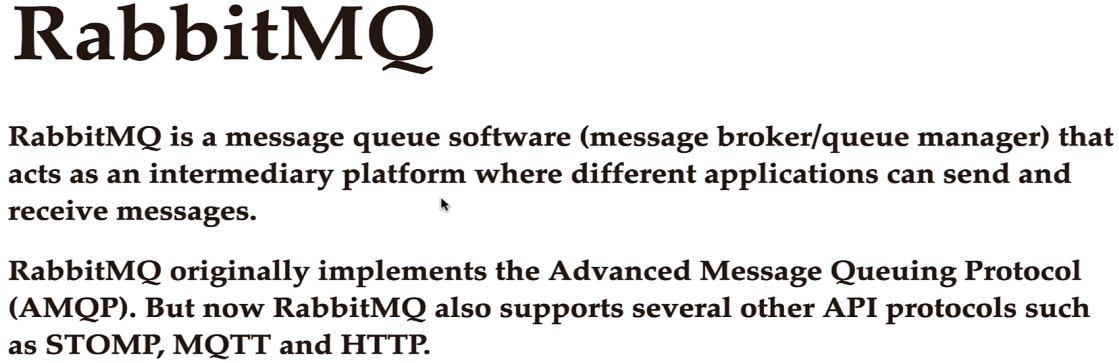
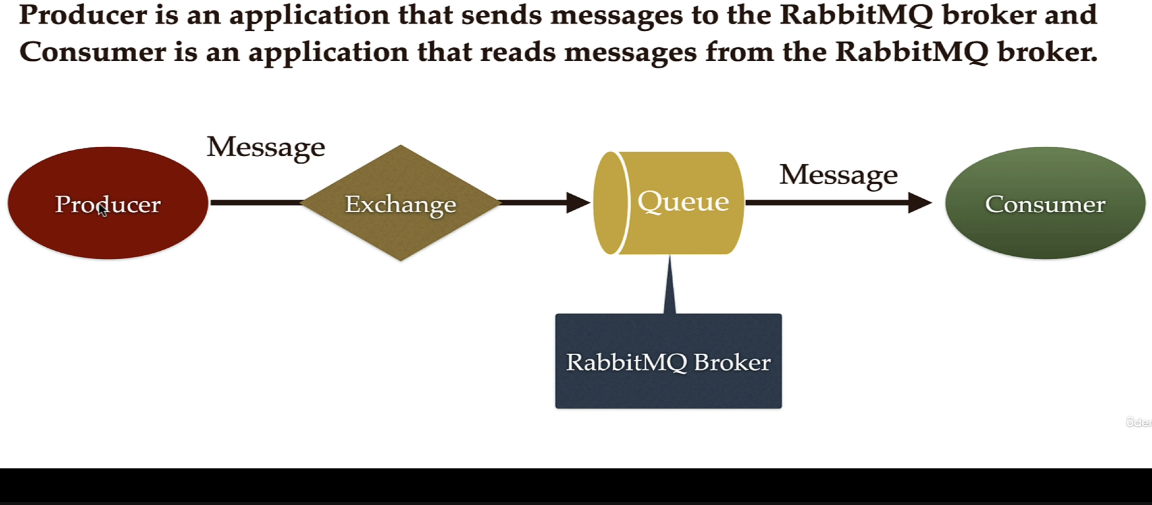
**RabbitMQ**



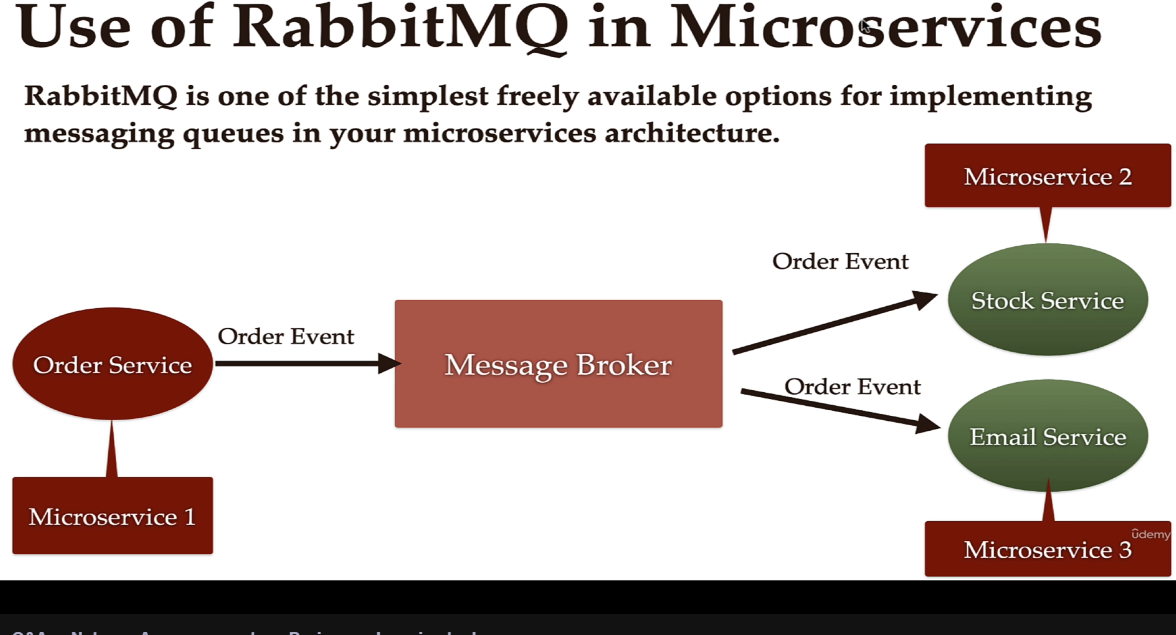




Producer produces the message and it will send that message to the exchange. And then exchange will basically use that message along with some set of rules to route that message to the queue.

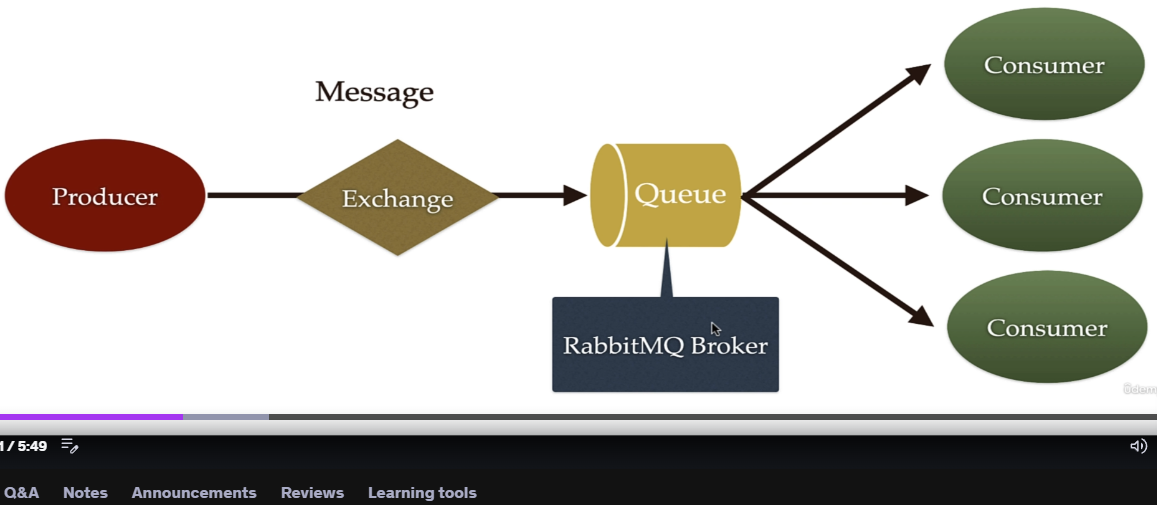
Exchange & Queue 🡪 part of RabbitMQ broker

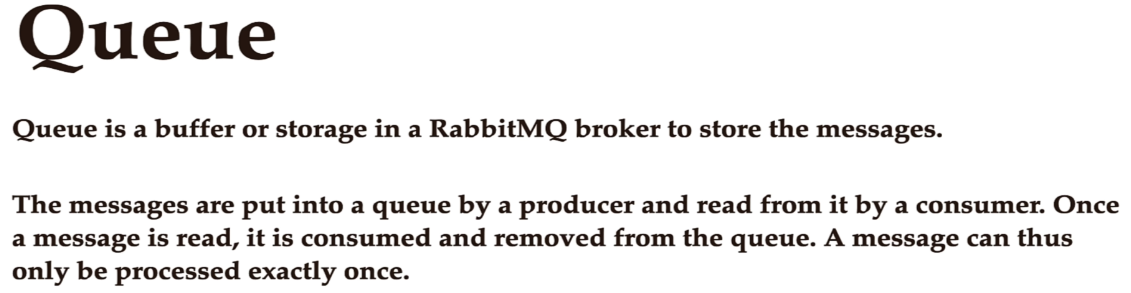
Producer & Consumer can be 2 different applications.



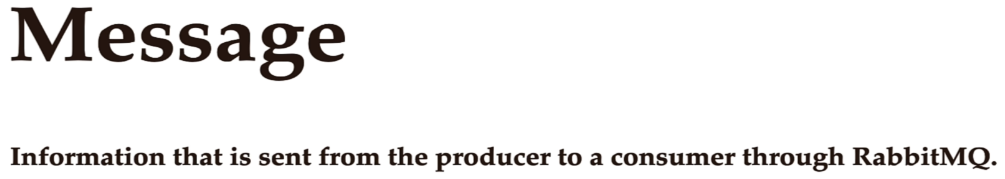
**Core Concepts –**

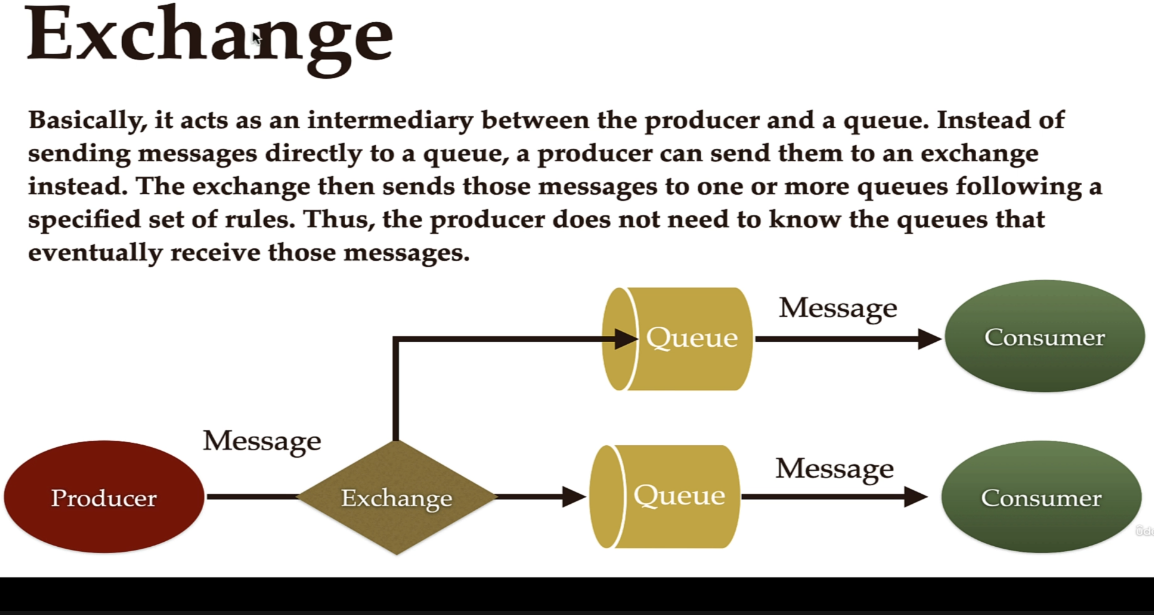
There can be multiple consumers that can subscribe to the RabbitMQ broker –





We can create any number of queue in RabbitMQ broker.







When there are multiple queues, exchange don’t know in which queue the message should be routed. In that case, exchange should use a routing key to route that message to the particular queue.

So Producer will basically send the message along with the routing key.



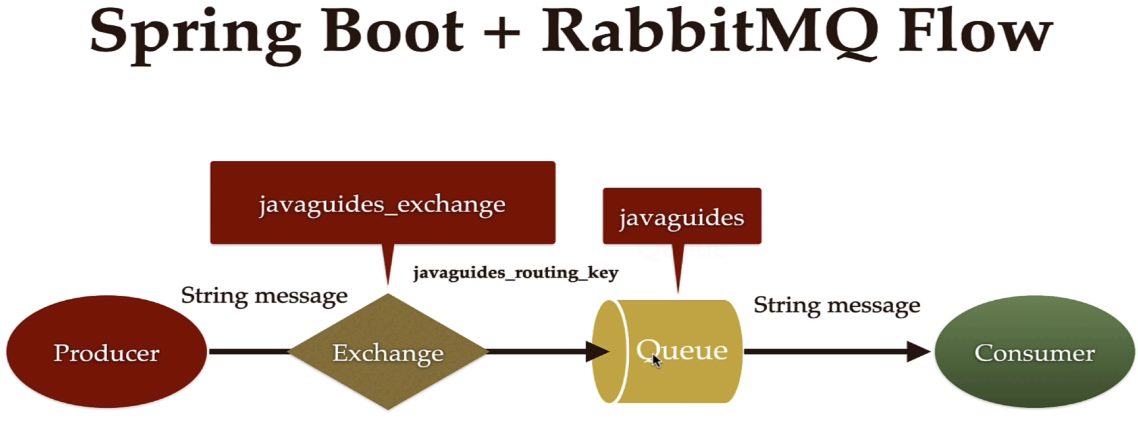
This binding is done by using routing key.

**RabbitMQ Architecture**

****

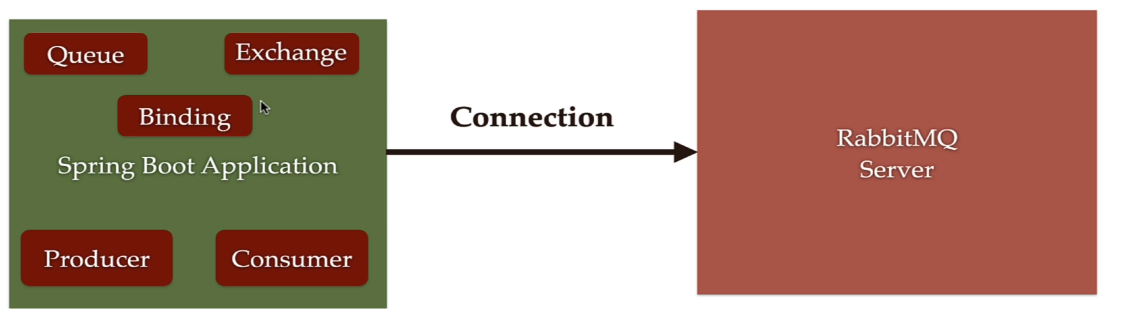
Producer will basically send a message along with the routing key to the exchange, and then exchange will use a routing key to route that message to the respective queues and then consumer will basically consume that message from the respective queues.

Building Springboot + RabbitMQ Producer & Consumer for String message -



Spring team provides Spring AMQP modules to work with different messaging system that uses AMQP (advanced messaging queuing) Protocol like RabbitMQ.

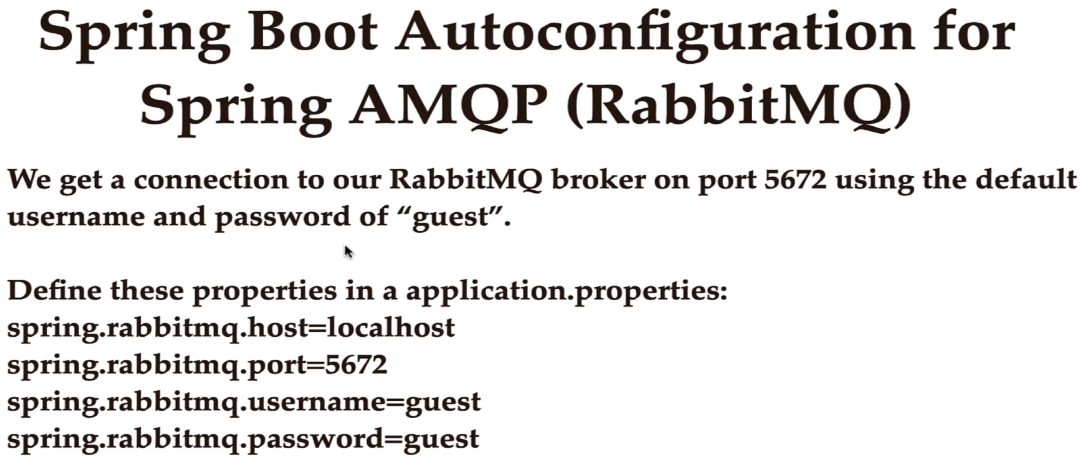
**# Connection between Spring Boot and RabbitMQ**

****

docker pull rabbitmq:3.10.5-management

docker run --rm -it -p 15672:15672 -p 5672:5672 rabbitmq:3.10.5-management

Spring boot autoconfiguration uses these properties behind the scenes to automatically connect our application with the RabbitMQ server.

****These are the default values of properties. If we don’t write these propertes in application.properties file then Springboot will automatically use these default values to connect out app with RabbitMQ broker on port 5672 and uses the username & password as “guest”.

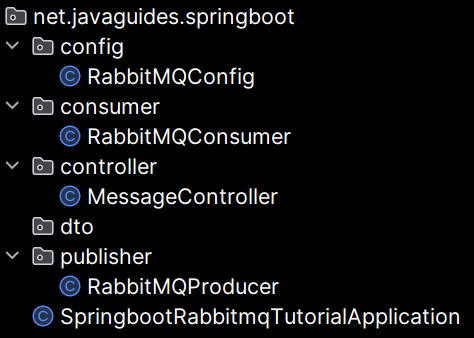
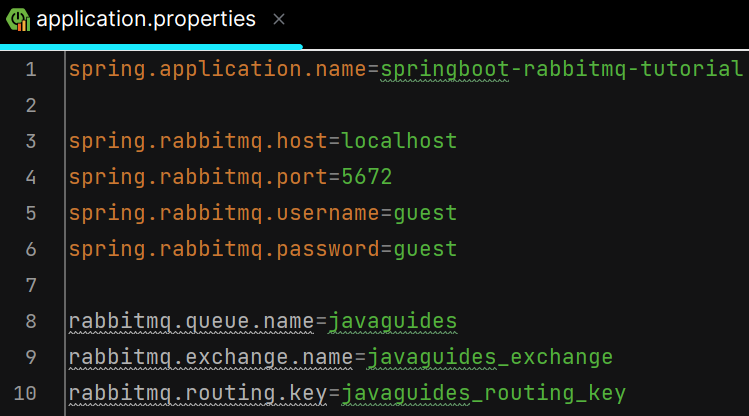
These default values work whenever you install rabbitMQ locally in a development environment. Lets say if your rabbitMQ broker is runing on a different machine, then you have to provide these properties.

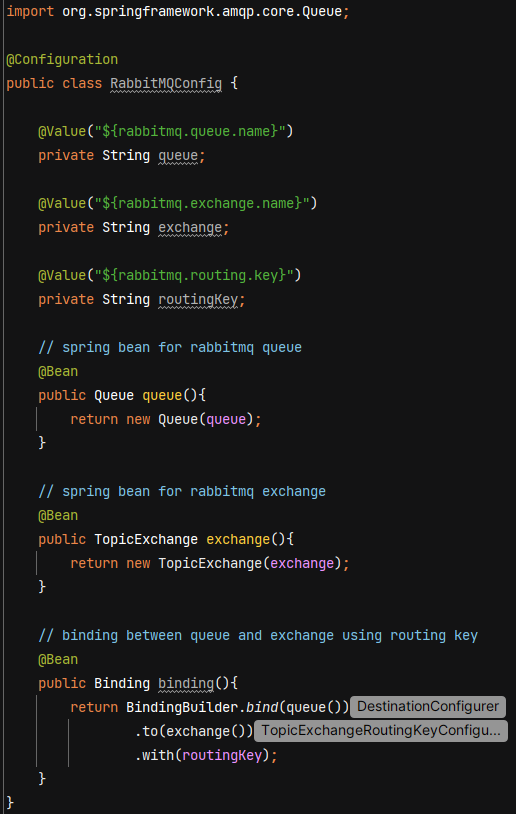
**# Configure RabbitMQ in spring boot app**

We use BindingBuilder class to bind a Queue with Exchange using routing key.

Apart from these beans, we need to also configure a ConnectionFactory, RabbitTemplate and RabbitAdmin. These are infrastructure beans that are required for our app with work with Rabbitmq broker. But spring boot autoconfiguration will automatically configure these 3 beans for us.

We don’t have to explicitly create spring bean or a configuration for these three classes.

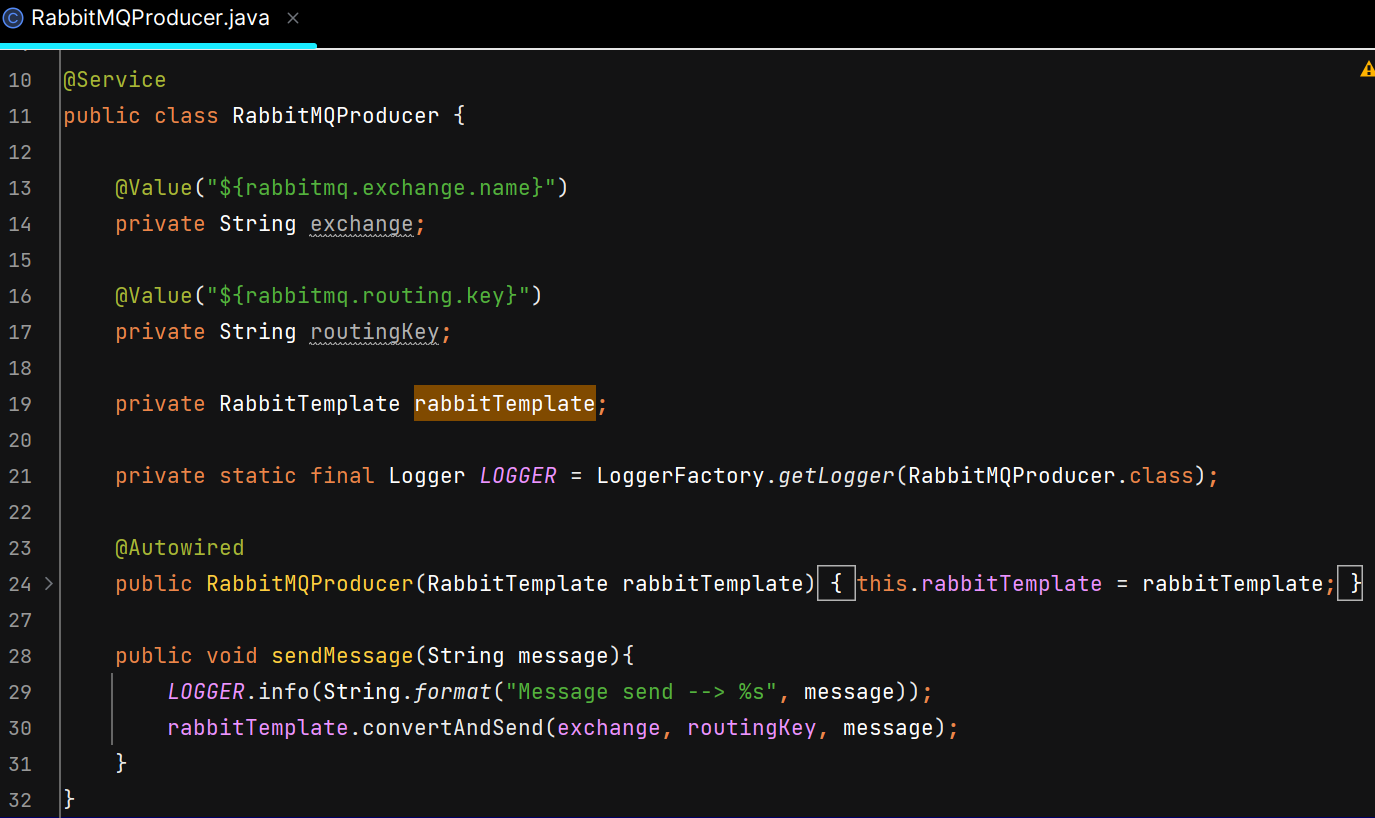


**#Create RabbitMQ Producer**

@Service annotation will create RabbitMQProducer class a spring bean and it will rregister in spring IOC container.

We’ll use RabbitTemplate to send the messages. Spring boot automatically configure RabbitTemplate for us. We just need to inject and use it.

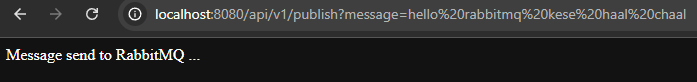
Whenever Spring bean has only 1 parameterized constructor then we can omit @Autowired.



**# Create REST API to Send Message**



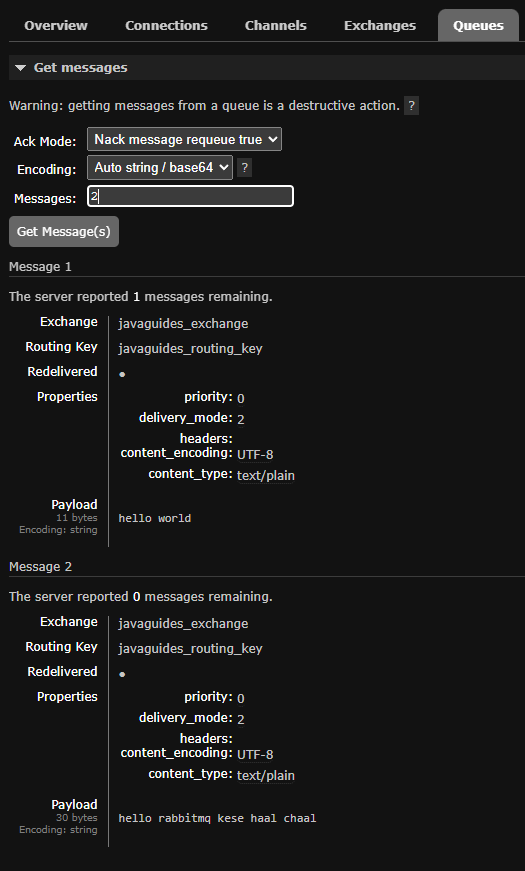
O/P –



In RabbitMQ Management UI –

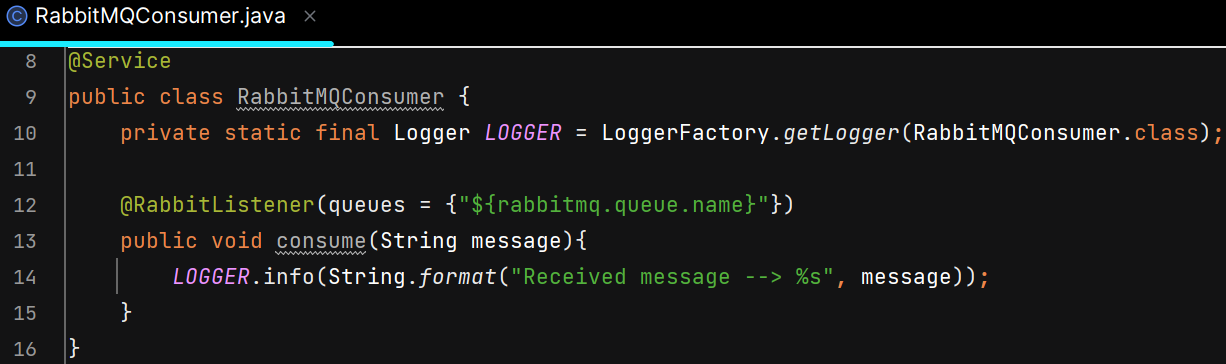


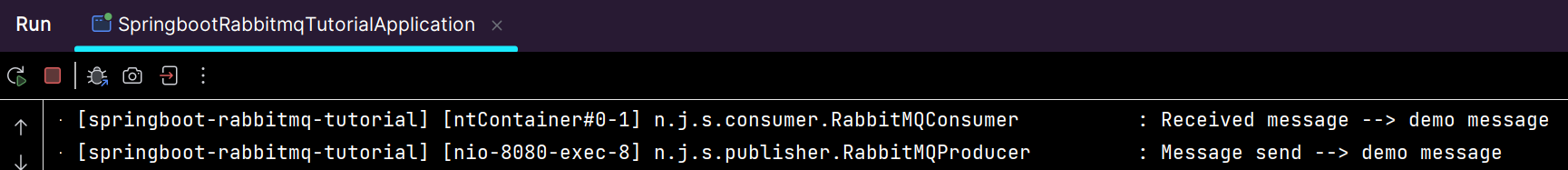


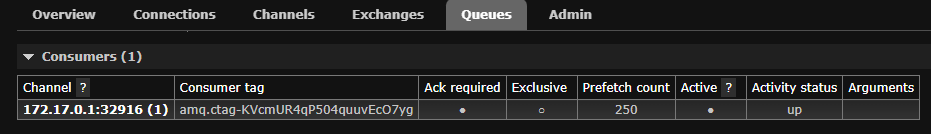
 

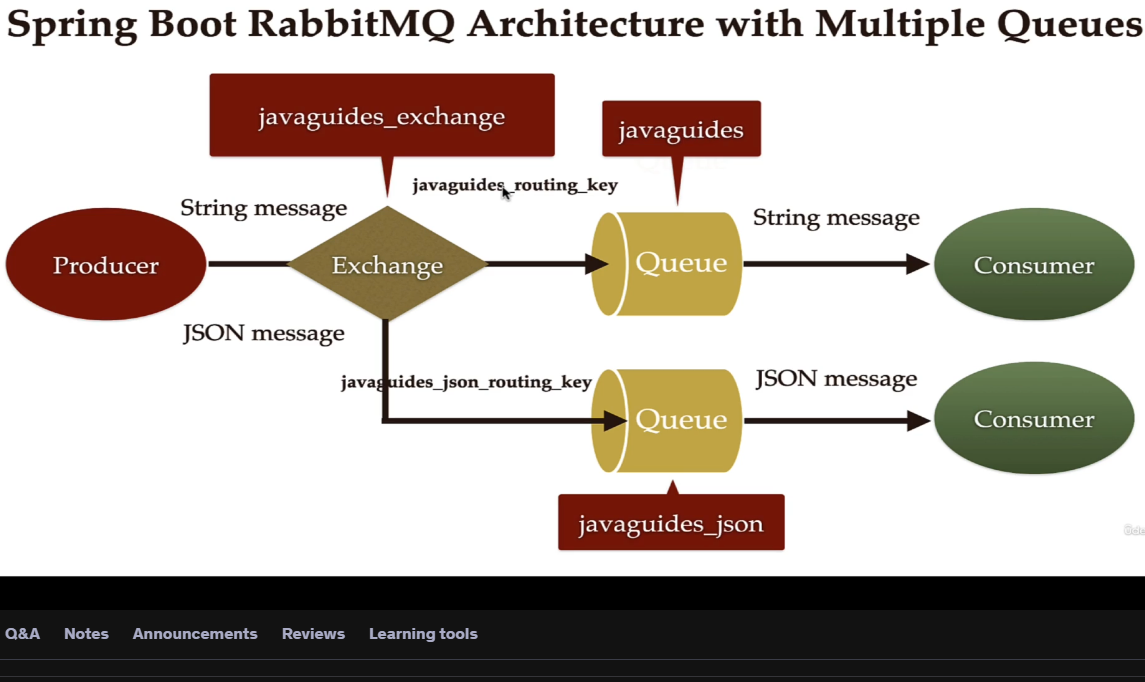
**#Create RabbitMQ Consumer**

@RabbitListener annotation is used to read or consume the message from the particular queue. It tells Spring when a message comes into this queue, call this method and pass the message in. Basically that method will consume the message from that particular queue.



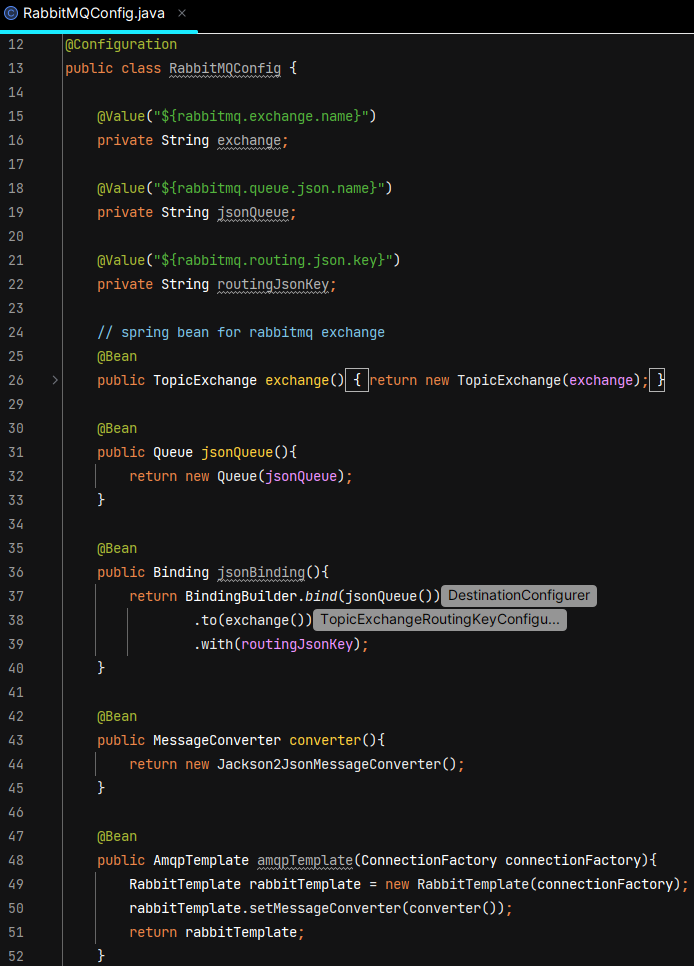
O/P – 

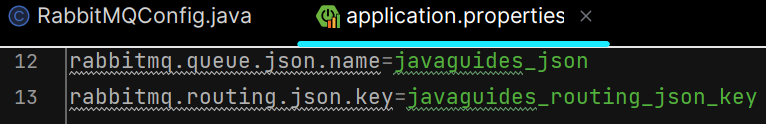


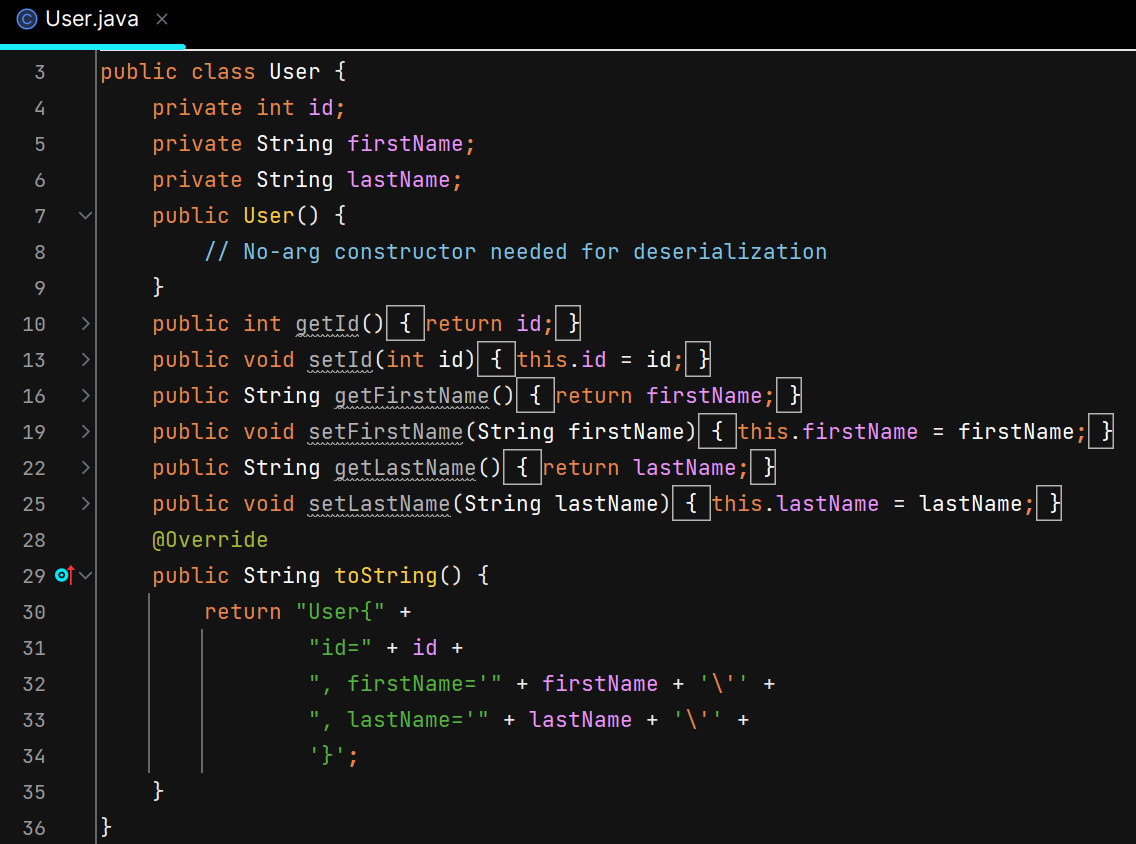


**# Configure RabbitMQ use JSON and Create POJO Class to Serialize/Deserialize**

Earlier we were using the default RabbitTemplate provided by spring autoconfiguration but now we have to create our own RabbitTemplate and set a JSON message converter to that RabbitTemplate so that it should support to send a JSON message.

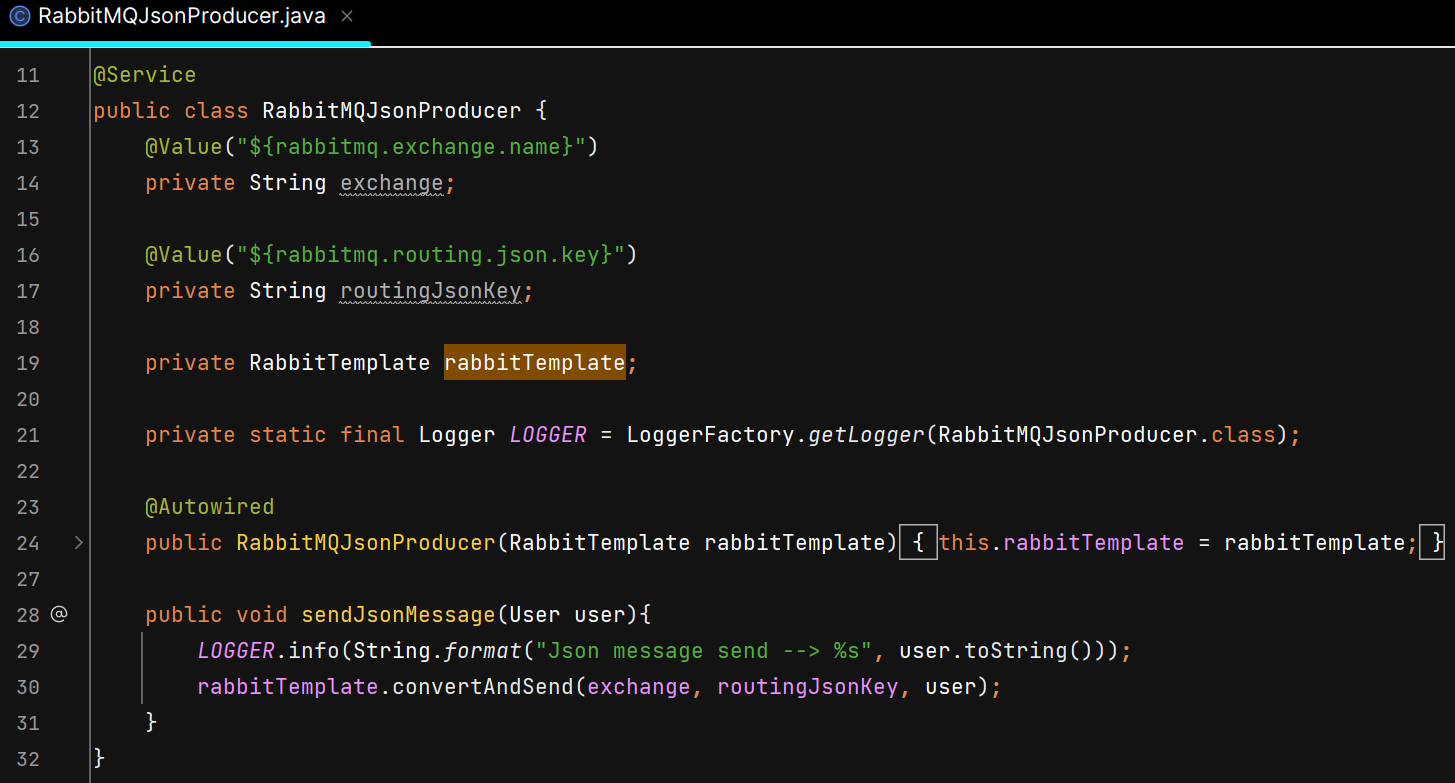




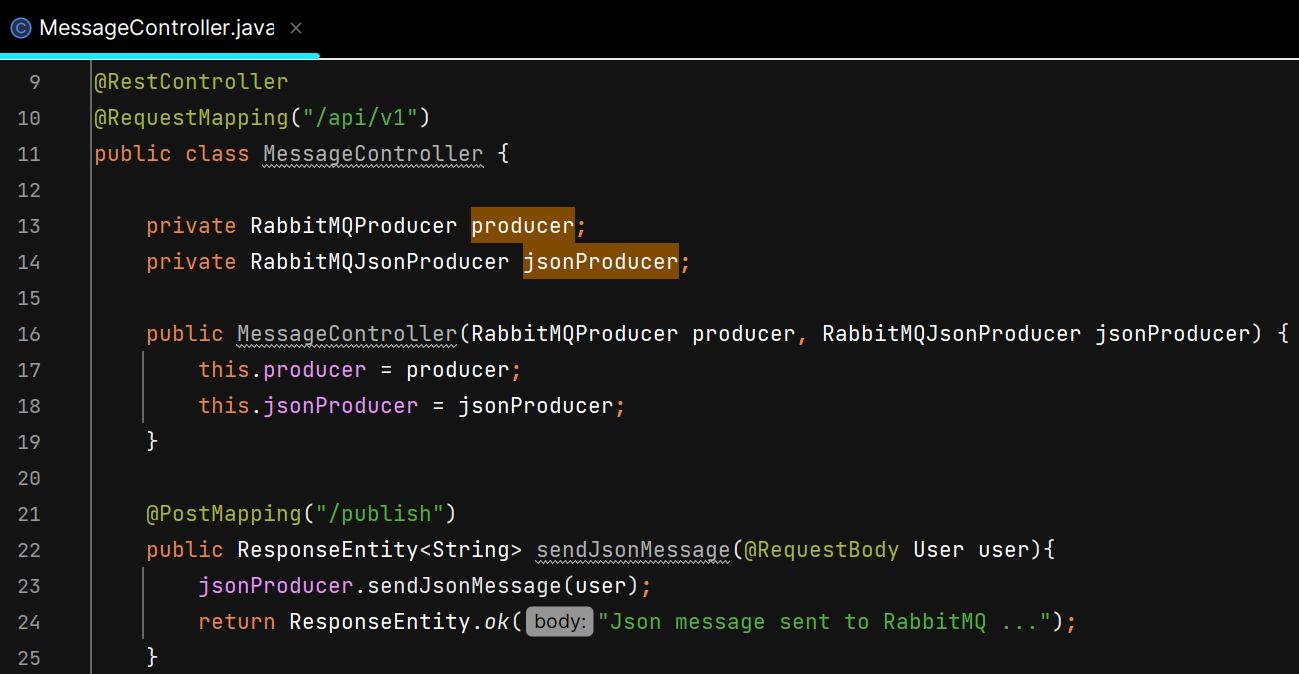


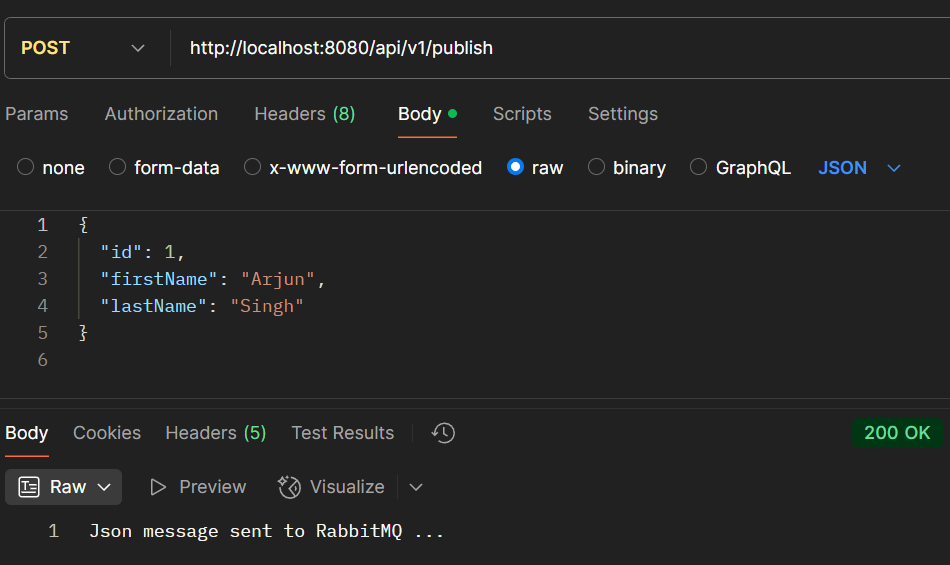
**# Create RabbitMQ Producer to produce JSON Message**

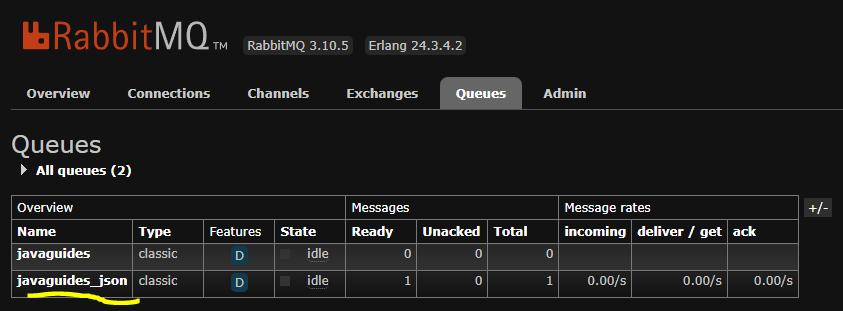
RabbitTemplate internally uses MessageConverter to convert the Java object into JSON and vice versa.

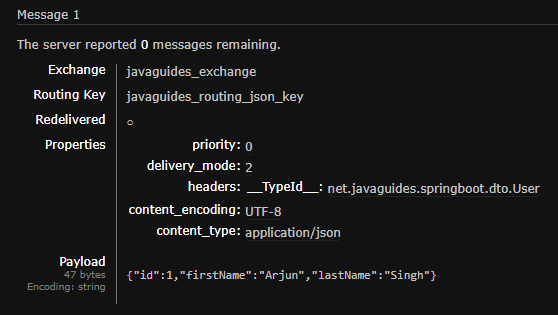


**# Create REST API to Send JSON Object**



O/P – 



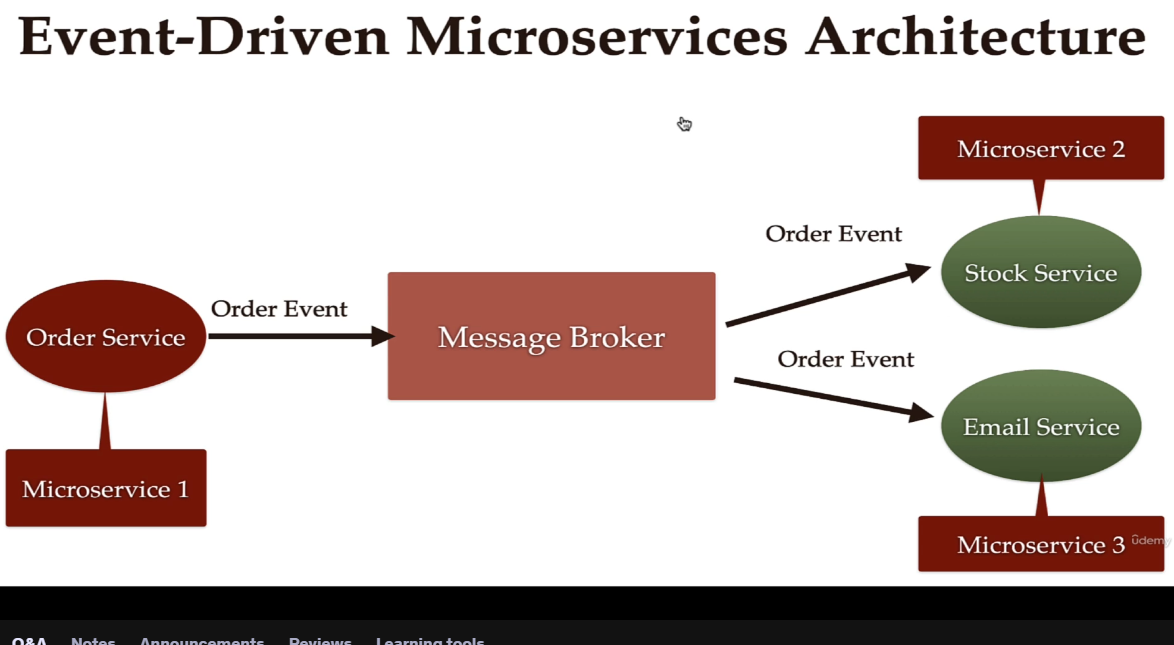


**# Create RabbitMQ Consumer to Consume Message**



O/P –





Whenever the customer places an order, at that time Order Event is created. That event is published to Message broker and then Stock Service & Email Service will consume that message.

In Event driven microservices architecture, we create multiple microservices and we use a Message Broker as a messaging system for asynchronous communication between multiple microservices.

Advantages –

1. Improves flexibility and maintainability
2. High scalability – u can create as many mircoservices you want
3. Improved availability – if any one microservice goes down it won’t impact other microservices

