Messaging Azure RabbitMQ

Dac Vu Ho

Pace University

Introduction

RabbitMQ is a message-queueing software also known as a message broker or queue manager

A message can include any kind of information. It could, for example, have information about a process or task that should start on another application (which could even be on another server), or it could be just a simple text message. The queue-manager software stores the messages until a receiving application connects and takes a message off the queue. The receiving application then processes the message.

Execution

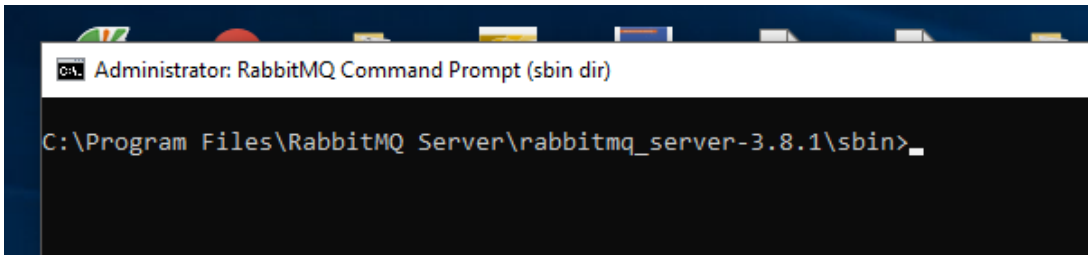
In this project, I will build messaging application with Java and RabbitMQ

The project includes 5 steps**:**

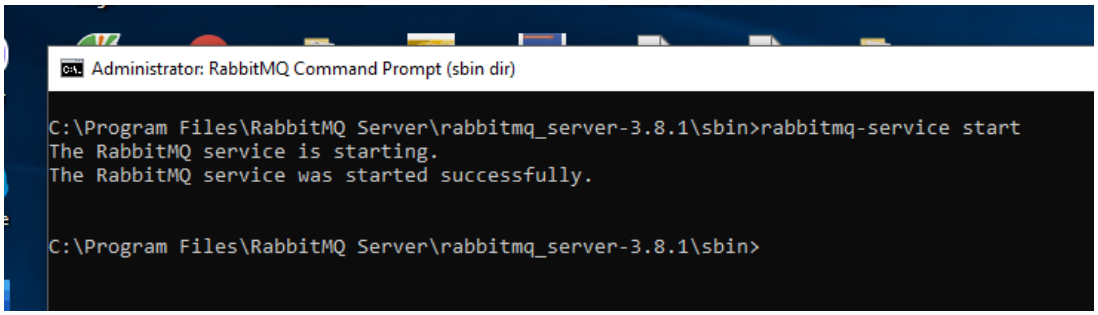
- Install RabitMQ and enable its web interface
- Create a RabbitMQ and Java project
- Send messages to RabbitMQ broker and queue with Java
- Look inside the RabbitMQ queues for messages
- Consume RabbitMQ messages from queue with Java

1. Install RabitMQ and enable its web interface

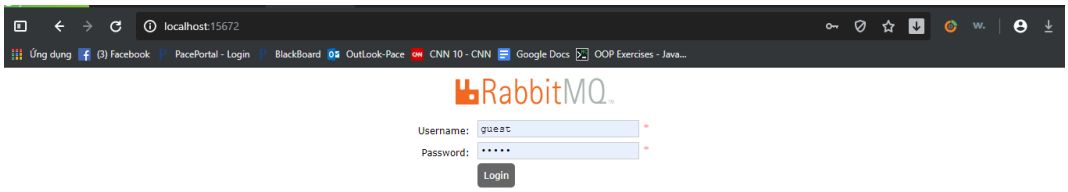
- Download and install RabbitMQ with the latest release of 3.8.1. We also need a line installed on your machine because RabbitMQ is built with the other programing language so you need to get the windows installer for Erlang.
- Download an OTP release, install it and then RabbitMQ
- On the command line, when we have RabbitMQ installed, we will find a new folder called RabbitMQ server on the Program Files, every version RabitMQ we have installed and folder rabbitmq_server-3.8.1



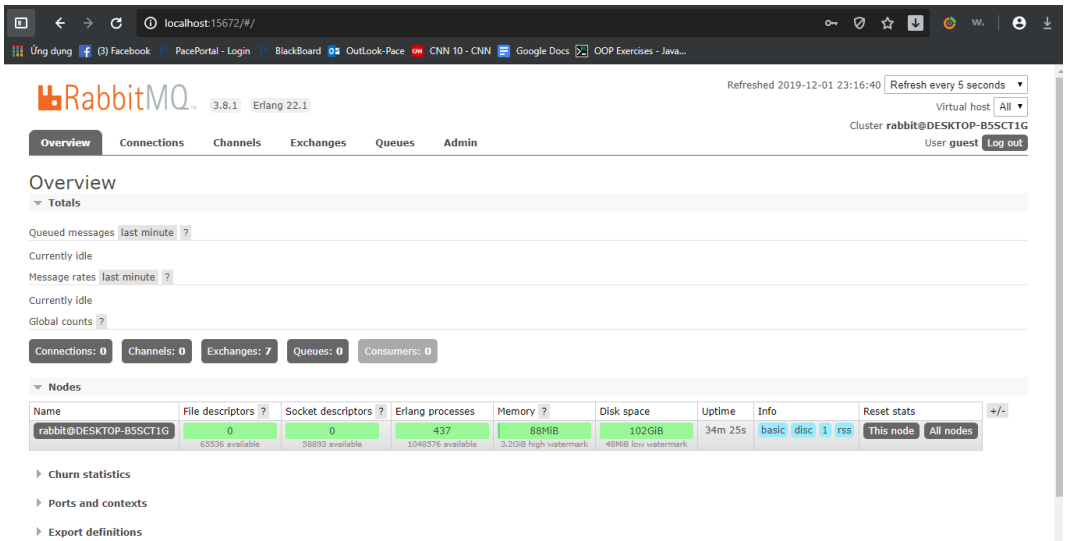
Start RabbitMQ service



- Open the browser window, type localhost:15672 and we will see the RabbitMQ web interface. We log in with “guest” for both user name and password



- And then we are already on the overview page with tabs: Connections, Channels, Exchanges, Queues, Admin

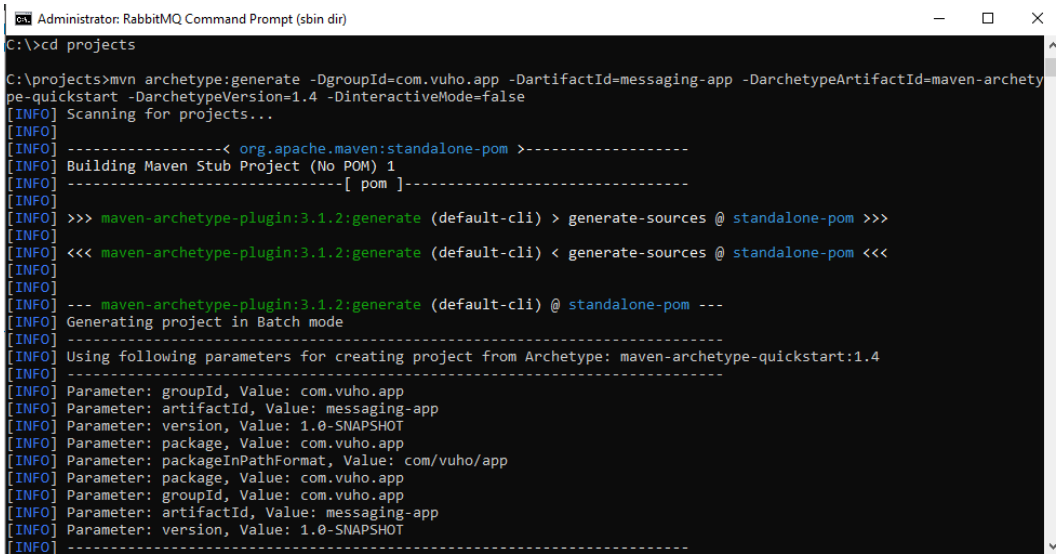


- RabbitMQ is running with Erlang version 22.1

2. Create a RabbitMQ and Java project

- Go to command line, in the “projects” folder, set the group ID and app, execute the following Maven goal:

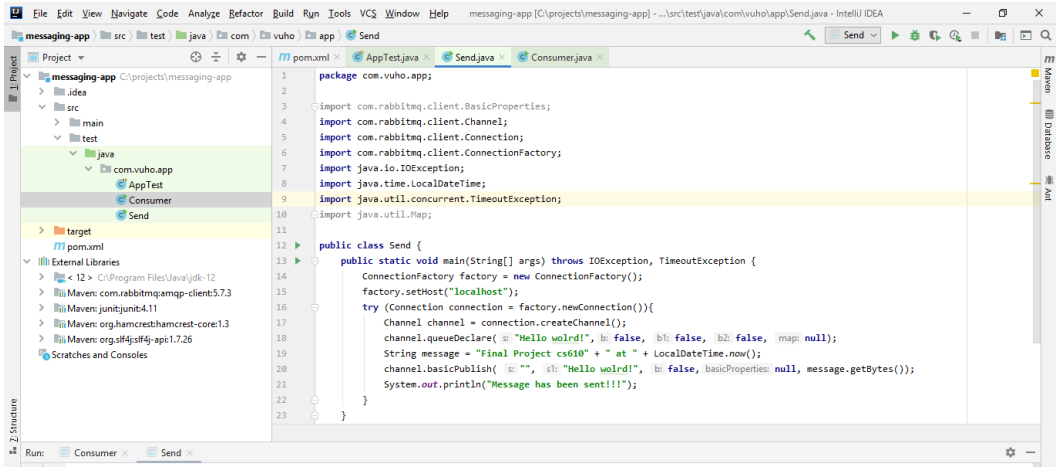
```
mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false
```



- I use IntelliJ to execute the Java code

3. Send messages to RabbitMQ broker and queue with Java

Create Send Class

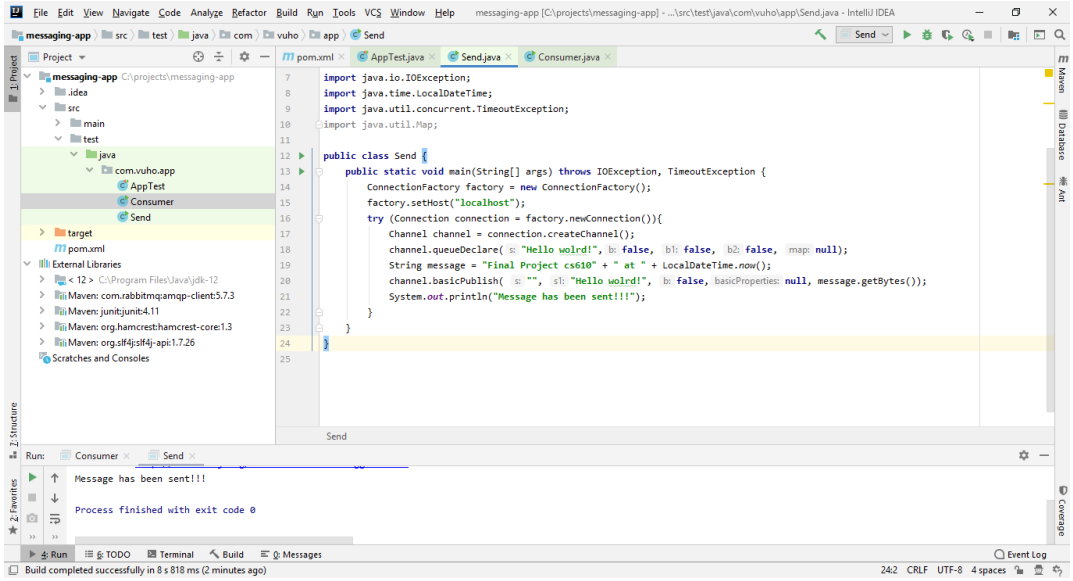



```
ConnectionFactory factory = new ConnectionFactory(); //give the connection to RabbitMQ
```

```
factory.setHost("localhost"); //connection factory with the defauls

Channel channel = connection.createChannel(); //create channel

channel.queueDeclare("Hello wolrd!",false, false, false, null); //
create the queue on the RabbitMQ server with queue name

String message = "Final Project cs610" + " at " + LocalDateTime.now();
//the content of the message that will be sent to the queue

channel.basicPublish( "", "Hello wolrd!", false,null,
message.getBytes()); //send messages to the queue with queue name

System.out.println("Message has been sent!!!"); // print to the console
```

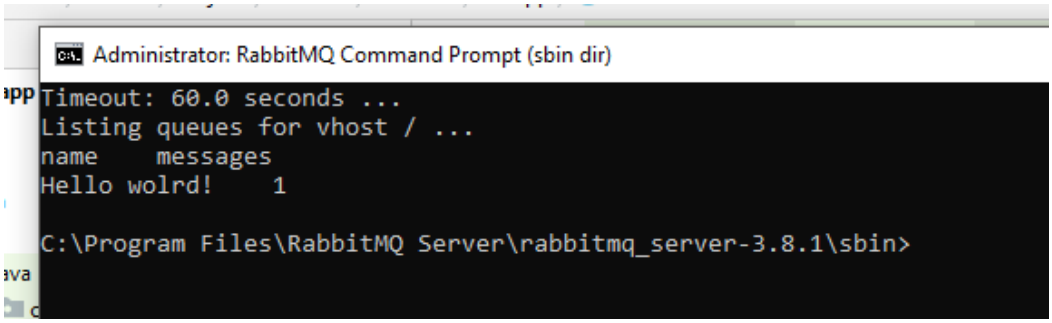
- Let's run the Send class… Successfully!!!

Message has been sent

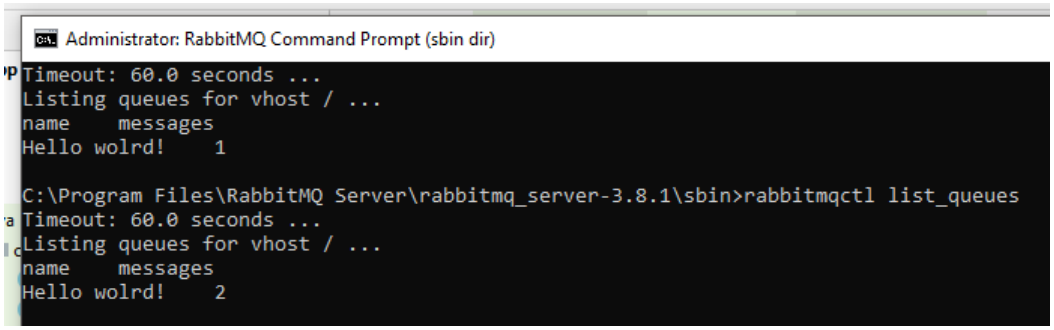


4. Look inside RabbitMQ queues for messages

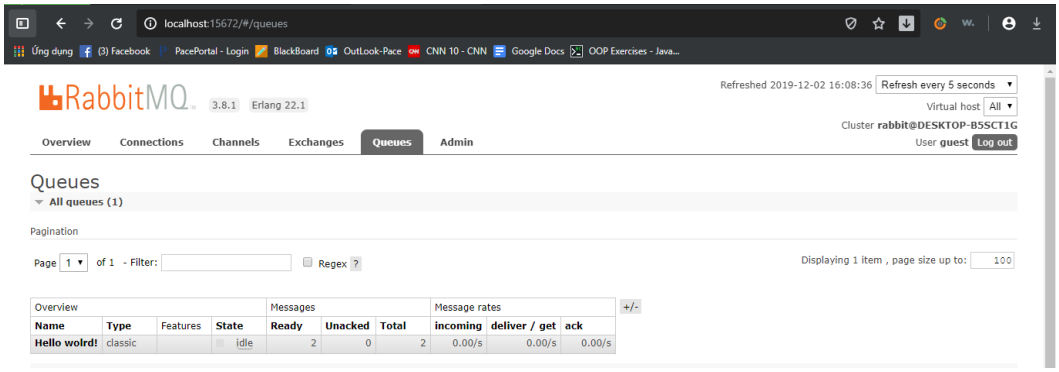
- Open up command line again, go to RabbitMQ server directory sbin and use RabbitMQ command line tool a parameter called list_queues



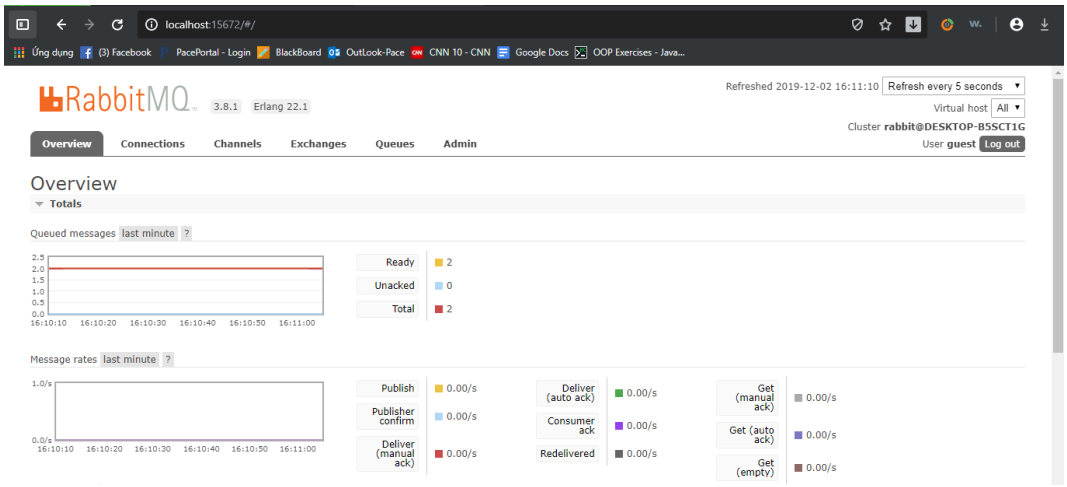
- We can see all names Hello world! (1 queue) and has one message inside. We continue go back Send class and send another message, and then open up the command line again to see the difference
- We will see Hello world! queue with two messages



- The problem is the RabbitMQ command line tool does not give user the opportunity to have look inside the queue and print out for example the message bodies. Therefore, we are going to use the web interface to do that. By login localhost:15672 then click Queue, we can see an overview of all the queues. In Hello world! queue, it has 2 messages inside, no message rates that means no incoming messages, no delivered messages so that is all being set to zero messages



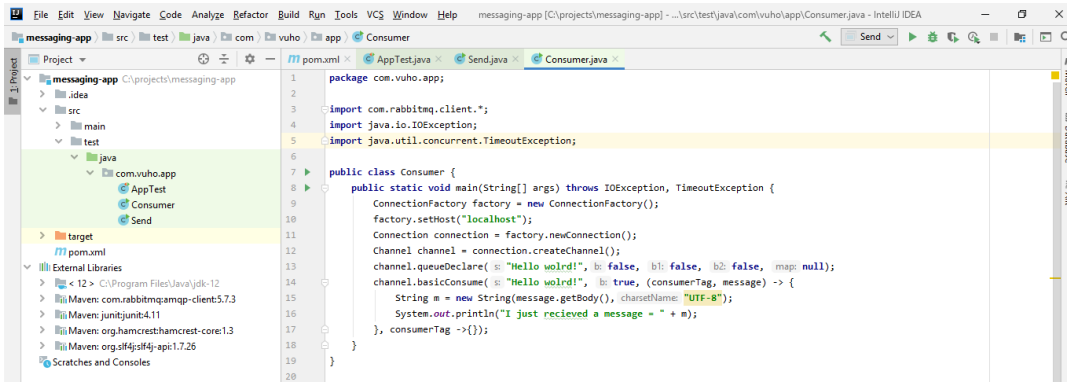
- Click on Overview, we will see a couple of statistics



- Scroll down, we will see the message body by click Get Messages(s) item

5. Consume RabbitMQ messages from queues with Java

- Create Consumer class



`ConnectionFactory factory = new ConnectionFactory();` //open up connection to the RabbitMQ server

`Channel channel = connection.createChannel();` // create channel

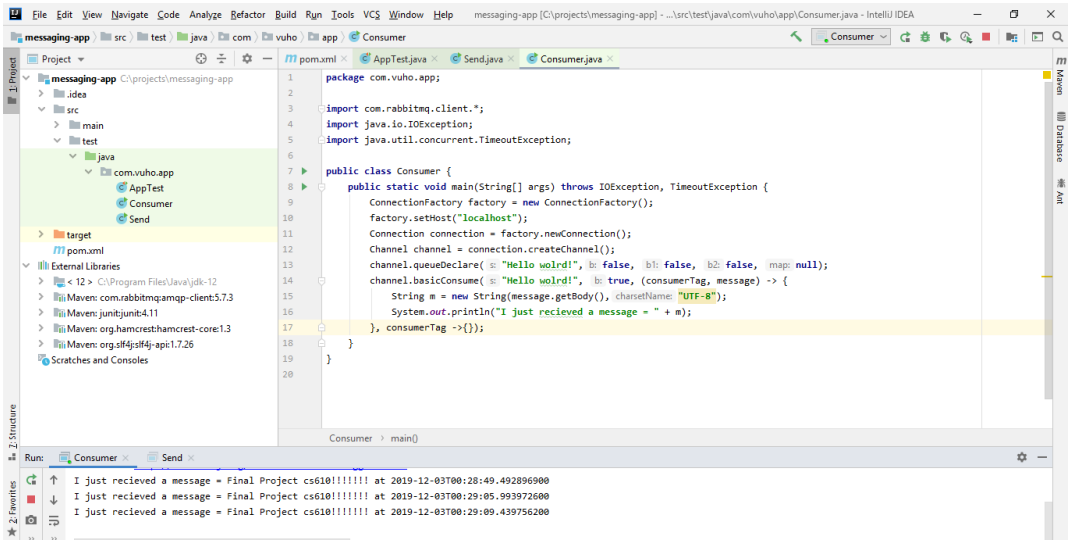
`channel.queueDeclare("Hello wolrd!",false, false, false, null);` //queue decleration. On the blank RabbitMQ instance if we start our Consumer before Send, we will not have any queues and that why we can just make sure to declare the queue in the Consumer

`channel.basicConsume("Hello wolrd!", true, (consumerTag, message) ->` {//the parameter list in basicConsume method include String queue (the queue name to consume message); Boolean (the client acknowledge the message or not, the message will be requeue again if we do not acknowledge it. Set 'true' because we want to acknowledge the message

`String m = new String(message.getBody(),"UTF-8");` //convert the bytes to a string

- We try to run Send class three times so we expect three messages. We go back to the consumer tab and as we can see, we just received the message three times

printed out the console with three different time stamps because we appended local date time



Conclusion

The project enables me to understand the message broker which is an architectural pattern for message validation, transformation and routing. RabbitMQ is a message broker software acts as an intermediate platform when it comes to processing communication between two applications. RabbitMQ is one such open-source enterprise messaging system modeled on the Advanced Message Queuing Protocol (AMQP) standard. RabbitMQ is fast, reliable, and a flexible messaging solution and therefore my preferred choice for enterprise-level message broker service.