**Java Message Service (JMS)**

JMS is a standard Java API that allows a Java application to send messages to another application. It is highly scalable and allows us to loosely couple applications using asynchronous messaging. Using JMS we can read, send and read messages. It allows the communication between different components of a distributed application to be loosely coupled, reliable, and asynchronous.

**Elements**

The following are JMS elements:

* **JMS provider**  
  An implementation of the JMS interface for message-oriented middleware (MOM). Providers are implemented as either a Java JMS implementation or an adapter to a non-Java MOM.

**Here are some implementations of JMS is as follows:**

* + Amazon SQS
  + Apache ActiveMQ
  + JBoss Messaging
  + RabbitMQ
* **JMS client**  
  An application or process that produces and/or receives messages.
* **JMS producer/publisher**  
  A JMS client that creates and sends messages.
* **JMS consumer/subscriber**  
  A JMS client that receives messages.
* **JMS message**  
  An object that contains the data being transferred between JMS clients.

A JMS message can be divided into three parts that are as follows:

1. **Header:**It contains the metadata about the message.
2. **Properties:**It can further be subdivided into three sections –
   1. **Application:**The java application sending message.
   2. **Provider:**It is used by the JMS provider and is implementation-specific.
   3. **Standard Properties:**These are defined by the JMS API.
3. **Payload:**This field is the message itself.

* J**MS queue**  
  A staging area that contains messages that have been sent and are waiting to be read (by only one consumer). Contrary to what the name *queue* suggests, messages don’t have to be received in the order in which they were sent. A JMS queue only guarantees that each message is processed only once.
* **JMS topic**  
  A distribution mechanism for publishing messages that are delivered to multiple subscribers.

**Provider implementations**

To use JMS, one must have a JMS provider that can manage the sessions, queues and topics.

The following is a list of common JMS providers:

* Amazon SQS’s Java Messaging Library
* Apache ActiveMQ
* Apache Qpid, using AMQP
* IBM MQ (formerly MQSeries, then WebSphere MQ)
* IBM WebSphere Application Server’s Service Integration Bus (SIBus)
* JBoss Messaging and HornetQ from JBoss
* JORAM from the OW2 Consortium
* Open Message Queue from Oracle
* OpenJMS from the OpenJMS Group
* Oracle WebLogic Server and Oracle AQ
* RabbitMQ from Pivotal Software

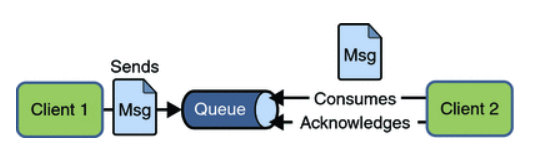
**NOTE:** This is the link for Hello-world example on JMS with Active MQ implementation (<https://examples.javacodegeeks.com/enterprise-java/jms/apache-activemq-hello-world-example/>).

**Models**

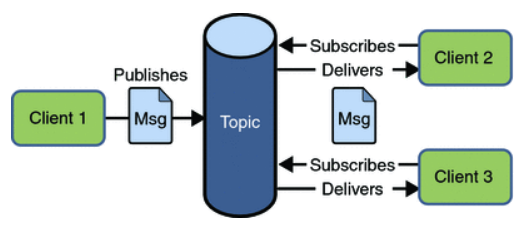
The JMS API supports two distinct models:

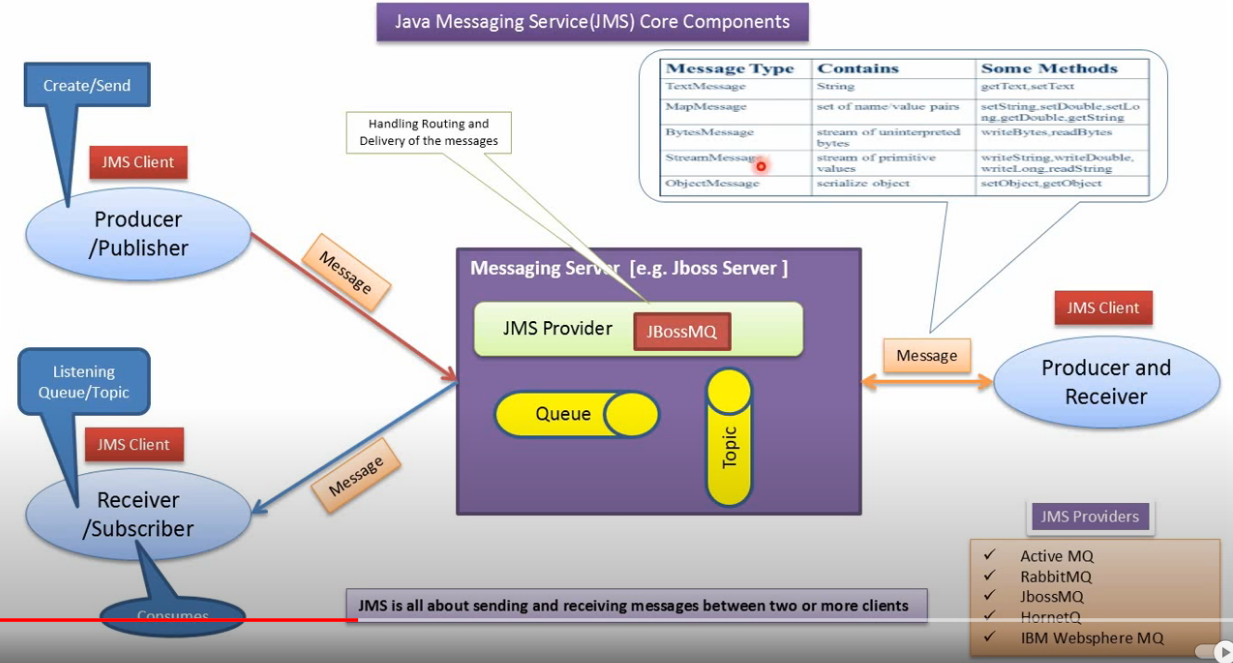
* Point-to-point
* Publish-and-subscribe

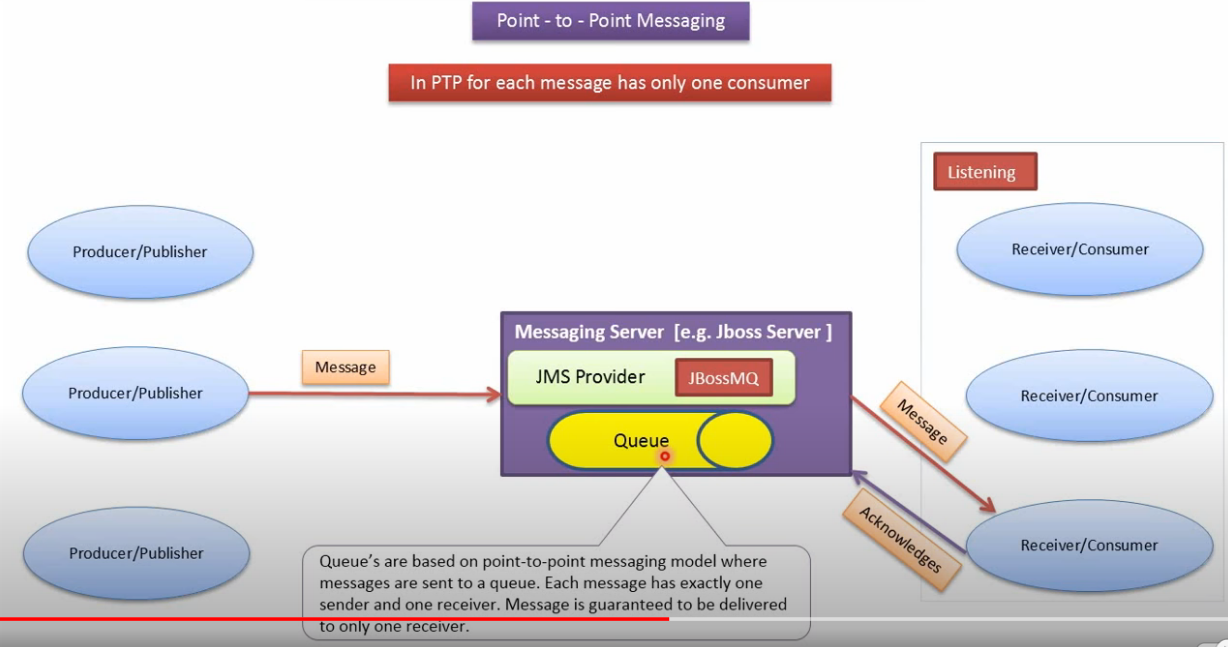
1. **Point-to-point**: In PTP model, one message is **delivered to one receiver** only. Here, **Queue** is used as a message oriented middleware (MOM). The Queue is responsible to hold the message until receiver is ready. In PTP model, there is **no timing dependency** between sender and receiver.

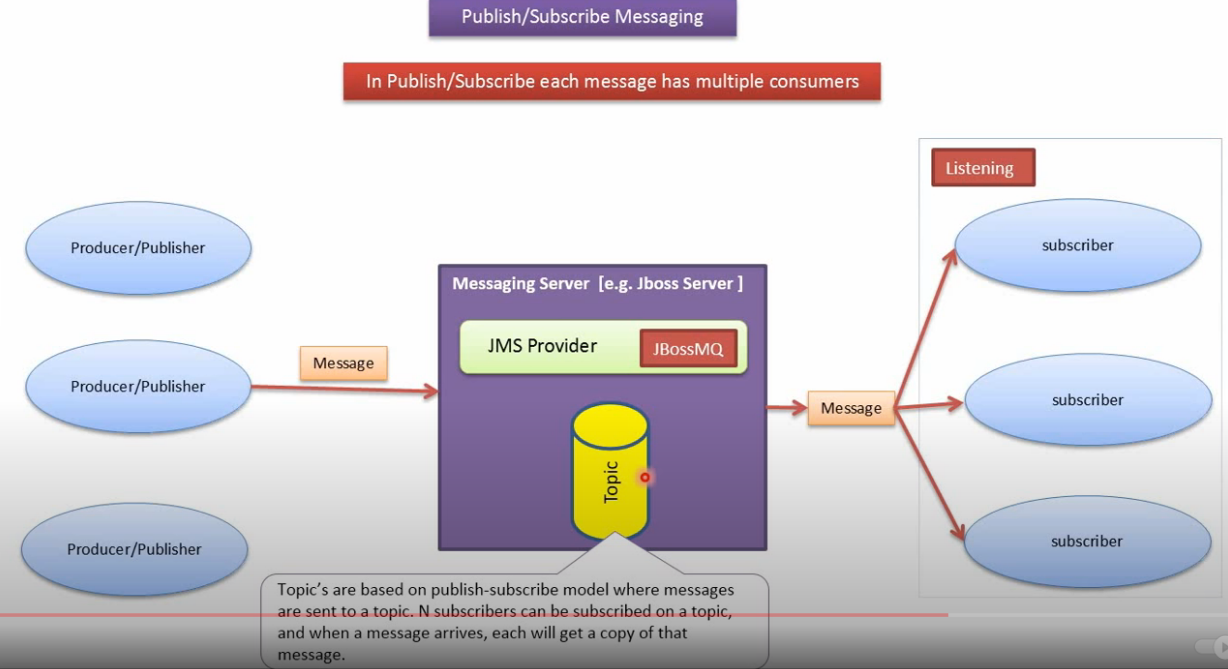


1. **Publish-and-subscribe**: In Pub/Sub model, one message is **delivered to all the subscribers**. It is like broadcasting. Here, **Topic** is used as a message oriented middleware that is responsible to hold and deliver messages. In PTP model, there is **timing dependency** between publisher and subscriber.









Software required:

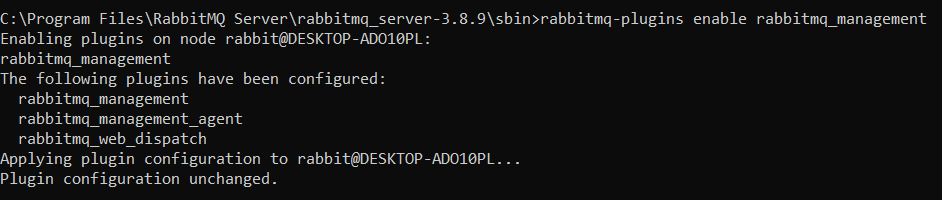
* Erlang: Download the erlang.exe based on the RabitMQ version you are going to use (<https://www.erlang.org/downloads>).
* RabbitMQ

Installation Guide:

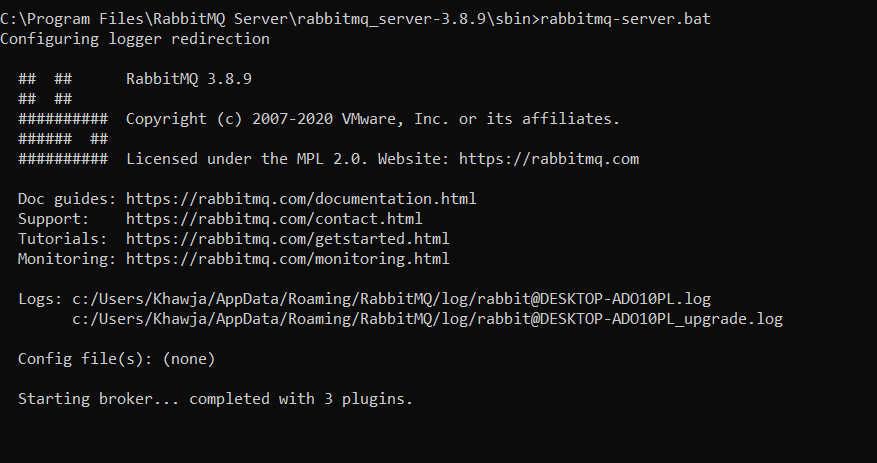
1. First install Erlang.
2. Install RabbitMQ
3. Open cmd prompt from the path where RabbitMq has been installed (C:\Program Files\RabbitMQ Server\rabbitmq\_server-3.8.9\sbin)
4. The management plugin is included in the RabbitMQ distribution. Like any other plugin, it must be enabled before it can be used. That's done using rabbitmq-plugins:



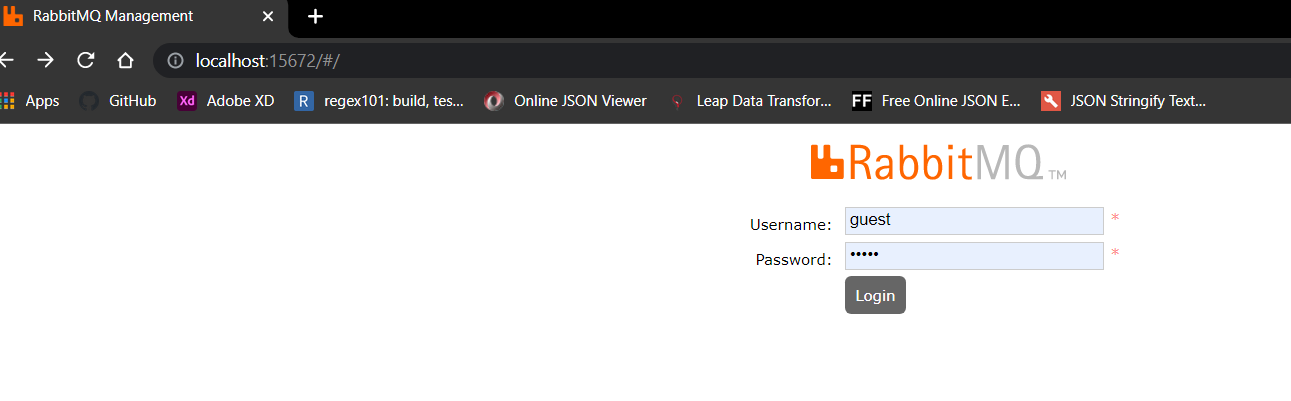
If it’s successfully enabled, we see the output as shown below



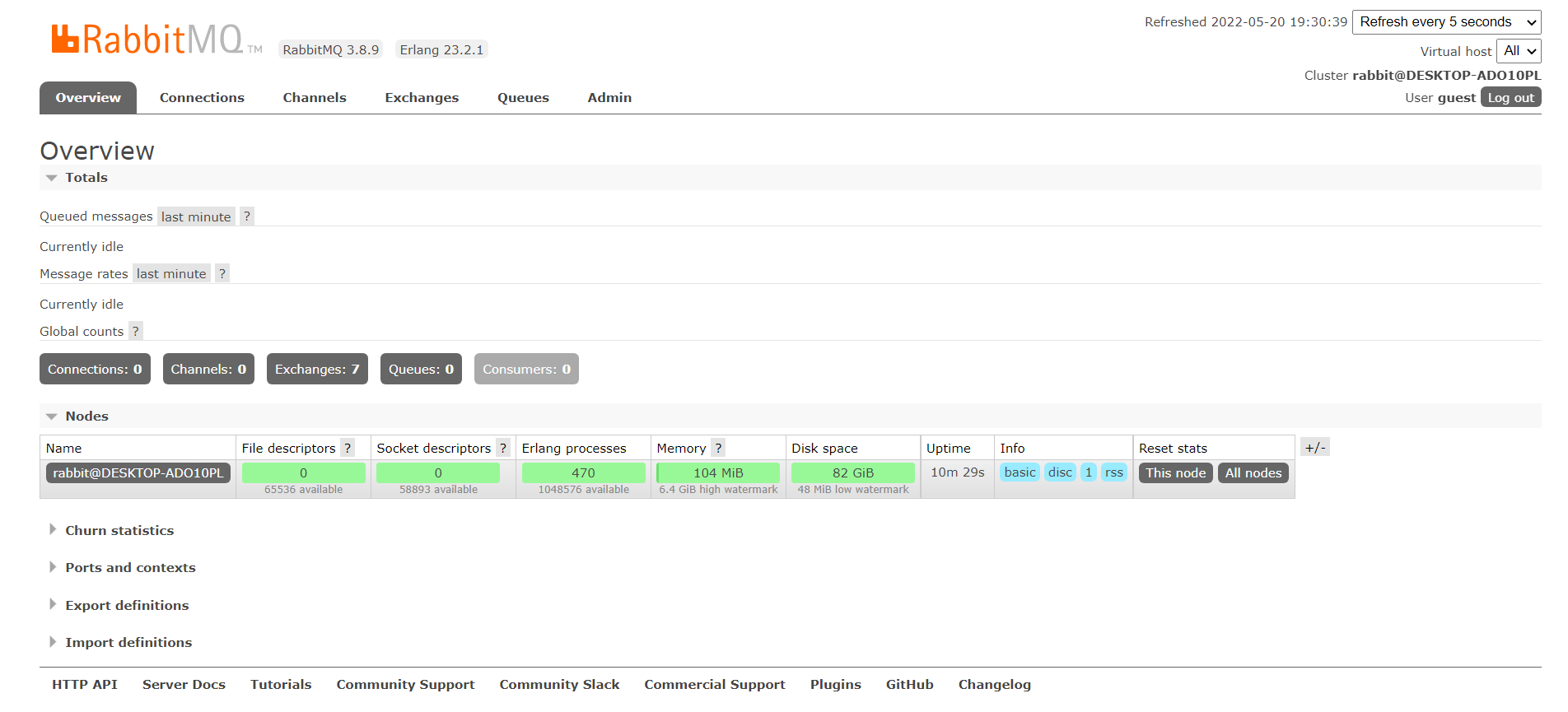
1. Start the server using bat file rabbitmq-server.bat



On successful start hit the URL (<http://localhost:15672/#/>) we should be able to see below login screen.

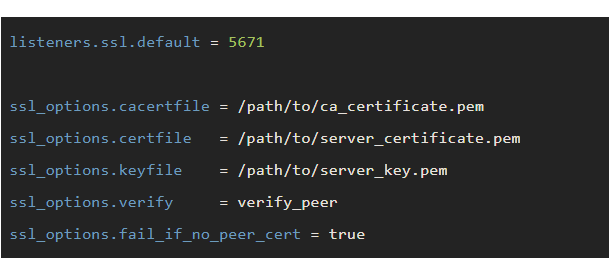


Login with username and password as (guest), we will get the home page of RabbitMQ



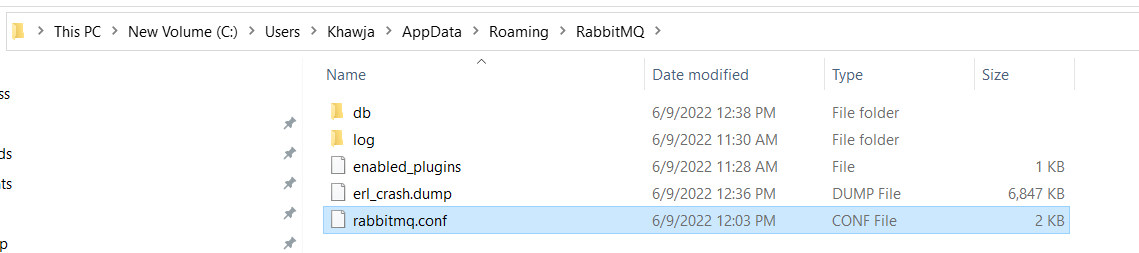
**Providing RabbitMQ TLS support (Transport Layer Security)**

1. Generate required certs. So, the required certs are listed below …

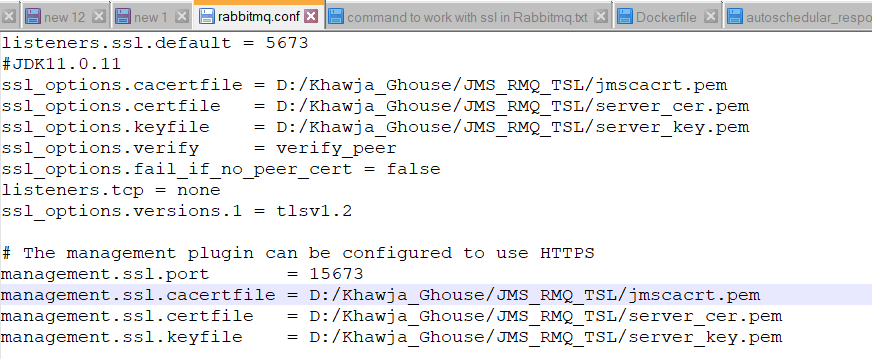


So we have all the command to generate certs in the file **command to work with ssl in Rabbitmq.txt**

1. Create a new rabbitmq.conf file and place inside > echo %appdata% /RabbitMQ/ as shown below

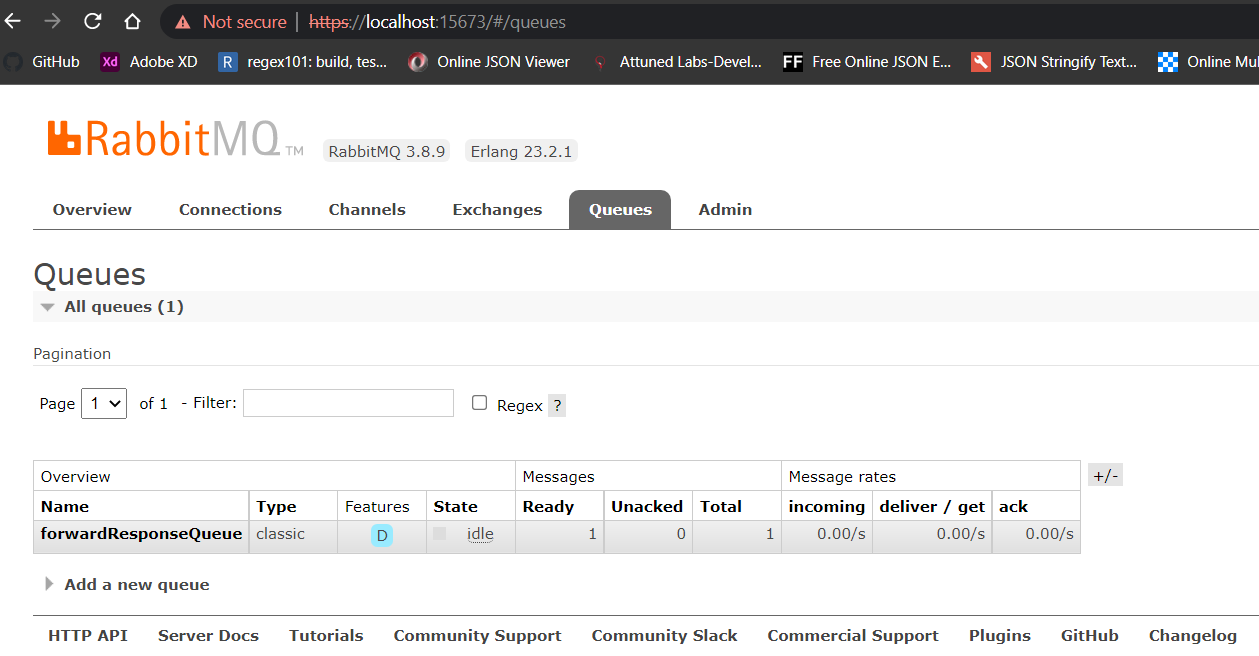
****

1. Map the generated certs in the rabbitmq.conf file as shown below.

****

With Java Version 11.0.3 we are only able to support tlsv1.2.

1. Restart the RabbitMQ Server hit URL (<https://localhost:15673>) we should be able to see the response as shown below



Connecting TLS JMS queue we can’t have normal xml bean, because we can’t set ssl context. We can create bean using JAVA annotation bean configuration.

Added **command to work with SSL in Rabbitmq.txt**  and bean configuration class for JMS TLS.