ESCOLA POLITÈCNICA SUPERIOR
UNIVERSITAT DE LLEIDA

DISTRIBUTED COMPUTING

RabbitMQ – components, benefits and application

Marc Cervera Rosell - 47980320C

Academic course 2022 - 2023

Bachelor's degree in computer engineering

Contents

List of Figures

Introduction

The main aim of this report is going to be to discover RabbitMQ software.
As everybody knows, there're three main essential attributes when designing and developing software that helps companies to offer trustworthy digital services and products. These three main attributes are: scalability, resiliency and interoperability.
In the actual market, there is a big amount of technologies that achieve that target. Among these technologies, we can find RabbitMQ.
RabbitMQ is an open-source message broker, sometimes called message-oreinted middleware. This software implements the "Advanced message queuing protocol". The RabbitMQ server is programmed in Erlang and uses the "Open telecom platform" framework to build his distributed execution and error commutation capacities.
RabbitMQ is a cross-platform software and the source code is released under the Mozilla public license.
In a simplified way, RabbitMQ defines queues that will store the messages sent by producers until the consumer apps get the message and process it.
So, It can be said that RabbitMQ, in his broker role, has the work of routing to the correct consumer the messages that the producer sends. To download and starting to use the software, just enter in the RabbitMQ webpage.

Architecture

Originally, RabbitMQ implemented the "Advanced message queuing protocol" (AMQP) but has been extended with a plug-in architecture to support "Streaming text oriented messaging protocol" (STOMP), "MQ telemetry transport" (MQTT), "HTTP and WebSockets" and "RabbitMQ streams".
HTTP is not really a messaging protocol but RabbitMQ can transmit messages over HTTP in three ways. The first one is the web STOMP plugin that supports STOMP messaging to the browser using WebSockets. The second one, is the web MQTT plugin that supports MQTT messaging to the browser using WebSockets. The tird, and the final, way is the management plugin that supports a simple HTTP API to send and receive messages. This is primarily intended for diagnostic purposes but can be used for low volume messaging without reliable delivery.

AMQP

As AMQP is the ”core” protocol supported by the broker, this section will be the most extensive.
RabbitMQ, originally only supported AMQP 0-9-1. All of the variants are fairly similar to each other, with later versions tidying up the unclear or unhelpful parts of earlier versions.
AMQP 0-9-1 is a binary protocol, and defines quite strong messaging semantics. For client it’s a reasonably easy protocol to implement, and as such there are a large number of client libraries available for many different programming languages and environments.
AMQP is an advanced queue messaging protocol that stands out for his fidelity. There’re commercial and open-source servers and interoperable clients for many programming languages, which facilitates their use. It’s used by big corporations that process millions of messages.
The deffinition of AMQP can be simplified in three words: ”message-oriented middleware”.
Behinf this simple definition there’re a lot of features available. Before AMQP there were some message-oriented middlewares such, for example, JMS, but AMQP has become the standard protocol to keep when a queue-based solution is chosen.
AMQP is pretty simple to understand, there’re client applications called producers that create messages and deliver it to an AMQP server, also called, broker. Inside the broker the messages are routed and filtered until arrive to queues where another applications called consumers are connected and get the messages to be processed.

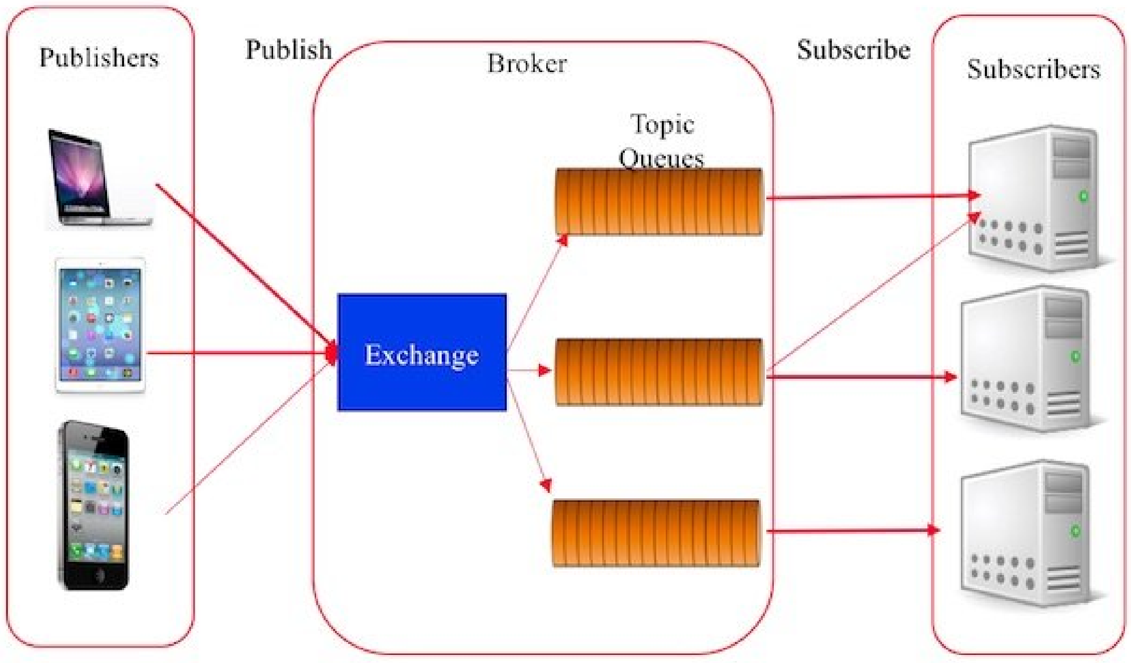

Figure 1: AMQP architecture work

Once seen this, it's time to deep inside the broker, where the magic of AMQP is done.
The broker has three different parts. The exchange, the queues and the bindings.

Exchange

This component is in charge of receiving the messages that have been sent to the broker by a producer and is the responsible of place them in the proper queue according to a routing key. This means that the producer does not send the messages directly to the queue, but send it to an exchange with a routing key.
In this way, if a producer wants to send a message to various queues, does not have to send It to each one of them, but the exchange is responsible for distributing this message to each one of the queues. There're three types of exchange: direct, topic and fanout.

- *Direct exchange*: Takes the routing key that comes inside the message and sends it to the queue that's associated with this exchange and with this routing key.
  Example:

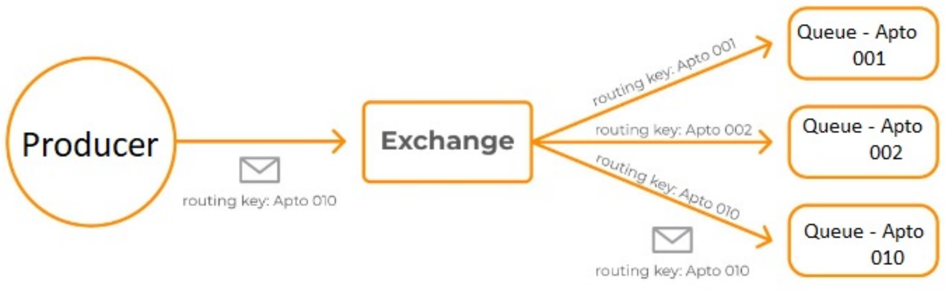

Figure 2: Message sending to a direct exchange

  Let's imagine the situation in what a postman has to deliver the mail to the concierge of a building and this has to put all the letters in the correspondence mailboxes. The process would be: The mail is delivered to the concierge (direct exchange)(the postman acts as the producer) and the concierge looks for the destination apartment and once the destination is found, the concierge puts the mail in the corresponding mailboxes.

- *Topic exchange*: Carries the message to the queues that complain with a pattern in the routing key. For example, if we've the queue "Q1" associated with the exchange "EXCH1" with the routing key "gunLicense.exam.theoretical" and the queue "Q2" associated with the same exchange with the routing key "gunLicense.exam.practical", a

message that's sent to the exchange "EXCH1" with the routing key "gunLicense.exam.*" will be sent to both queues.
Example:

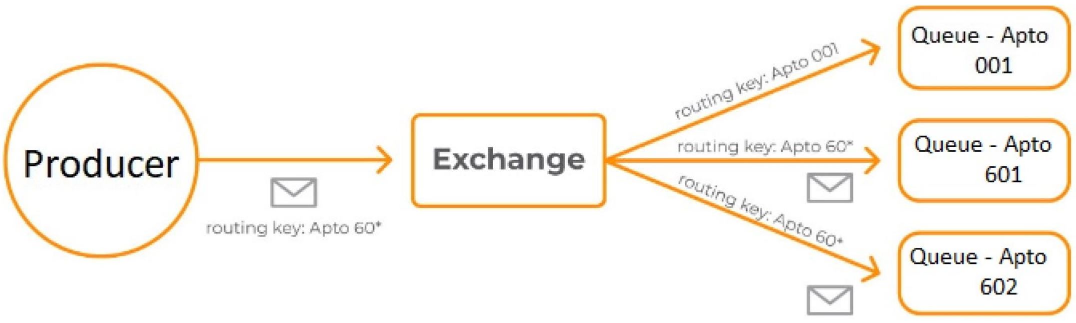

Figure 3: Message sending to a tpoic exchange

In this case, the situation is going to change kind of. Now, the situation is: The postman will deliver to the concierge two letters (both letters are equal) that go, only, to the neighbours of the last floor (let's imagine that in the last floor are the flats 601 and 602). Hence, the postman (producer in this situation) delivers the letters to the concierge and this puts the letters inside the mailboxes, only checking (in this example) the two first numbers 6 and 0 because the digits that identify the last floor are the two first ones.

- *Fanout exchange*: Sends the message to all the queues associated with the exchange, regardless of routing key.

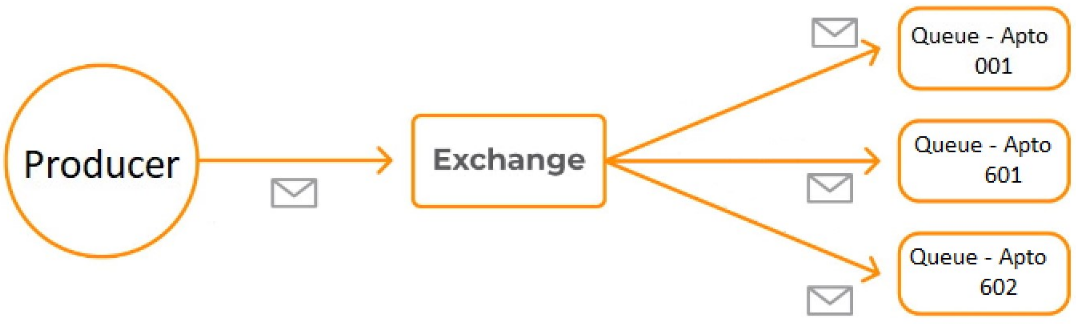

Figure 4: Message sending to a topic exchange

In this final situation, the concierge will receive a letter for the entire neighbourhood. Thus, the concierge will not check neither the floor nor the flat. Instead, will put a copy of the letter in each neighbour's mailbox.

Routing key

Is the identifier that uses the exchange to know the route of a message. At the same time is the identifier that uses a queue to know with which exchange has to associate.

Queue

A queue is a data structure that imitates real queues as the ones we can see in the mailbox or at the neighbourhood supermarket. A message queue is an asynchronous communication way that's used on microservices architectures. The messages are stored in the queue until are received and deleted being every message processed only once.
Hence, the queue is the component that stores messages from the exchange and sends them to the consumers that are listening for these messages.

Binding

A binding is a relationship between an exchange and a queue. This can be simply read as: the queue is interested in messages from this exchange.
Bindings can take an extra routing key parameter.
This is how a binding with a key could be created

```
channel.queue_bind(exchange = excahnge_name,
                   queue = queue_name,
                   routing_key = 'black')
```

The above code, is using the *Pika* Python client.
The meaning of a binding key depends on the exchange type. The *fanout* exchanges, just ignore this value.

Virtual host

RabbitMQ is multy-tenant system: connections, exchanges, queues, bindings and some other things belong to virtual hosts. If the user is familiar with virtual hosts in Apache, the idea is similar. However, there's an important difference: in Apache, virtual hosts are defined in

the configuration file; that's no the case of RabbitMQ; virtual hosts are created and deleted using *rebbitmqctl* or HTTP API instead.
A virtual host, provides logical grouping and separation resources. Separation of physical resources is not a goal of virtual hosts and should be considered an implementation detail.
For example, resource permissions in RabbitMQ are scoped per virtual host. A user does not have global permissions, only permissions in one or more virtual hosts. User tags can be considered global permissions but they are an exception to the rule.
Therefore when talking about user permissions it's very important to clarify what virtual host(s) they apply to.
A virtual host has a name. When an AMQP 0-9-1 client connects to RabbitMQ, it specifies a virtual host name to connect to. If the username is not granted permissions, the connection is refused.

Creating and deleting virtual hosts

Using CLI tools:

```
rabbitmqctl add_vhost qua1
```

```
rabbitmqctl delete_vhost qua1
```

Using HTML API

```
curl -u username:pa$sw0rD -X PUT http://rabbitmq.local:15672/api/vhosts/vh1
```

```
curl -u username:pa$sw0rD -X DELETE http://rabbitmq.local:15672/api/vhosts/vh1
```

MQTT

MQTT is M2M communication protocol of type message queue.
It's based on the TCP stack as the communication base. In the MQTT case, every connection keeps opened and is "reused" in every communication. It's a difference, for example, with an HTTP 1.0 petition where every transmission is reused through the connection.
MQTT was created by Dr. Andy Stanford-Clark of IBM and Arlen Nipper de Arcom in 1999 as a mechanism to connect devices used in the petrol industry.
Though initially was a proprietary format, in 2010 was released and in 2014, became a

standard according to OASIS (Organization for the Advancement of Structured Information Standards).
The MQTT work is a push messaging service with the pub-sub pattern. In this type of infrastructures, the clients they connect with a central server named broker.
To filter the messages that are sent to each client, the messages are arranged in topics hierarchically organized. A client can publish a message in a specific topic. Other clients can subscribe to this topic and the broker will send the subscribed messages.

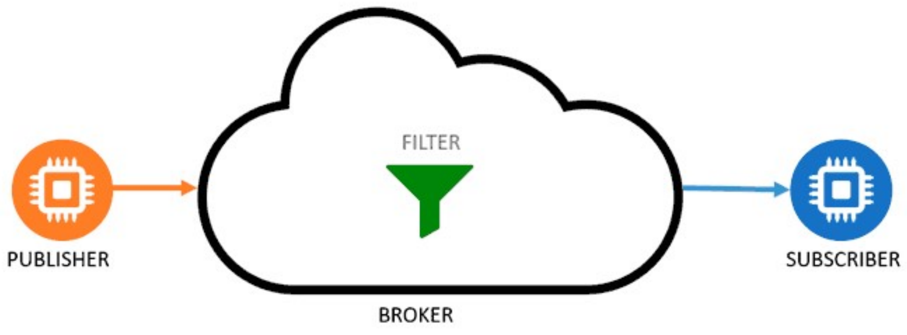

Figure 5: MQTT

The clients start a TCP/IP connection with the broker, which maintain a registration of the connected clients. This connection keeps opened until the client ends it. By default, MQTT uses the ports 1883 and 8883 when works on TLS.
To do that, the client sends a CONNECT message that contains the necessary information. The broker answers with a CONNACK message, that contains the result of the connection. To send the messages, the client uses PUBLISH messages, that contain the topic and the payload.
To subscribe and unsubscribe, the SUBSCRIBE and UNSUBSCRIBE messages are used. The server responds with a SUBACK or with a UNSUBACK.
To make sure that the connection is active, the clients send periodically a PINGREQ message which is answered, by the server, with a PINGRESP, Finally, the client disconnects sending a DISCONNECT message.

RabbitMQ streams

Applications

WebSockets

RabbitMQ vs. market

Advantages

Disadvantages

Conclusions

References

- Wikipedia's RabbitMQ article
- RabbitMQ's webpage
- *Conectar microservicios con colas de mensajes usando Spring y RabbitMQ*'s webpage
- *Conozcamos sobre RabbitMQ, sus componentes y beneficios*'s webpage
- AMQP architecture work picture
- Which protocols does RabbitMQ support?
- Creating and deleting virtual hosts
- MQTT