International Journal of Advance Research and Innovation
Vol. 7(3), Jul-Sept 2019, pp. 68-71
Doi: 10.51976/ijari.731909
www.gla.ac.in/journals/ijari

Article Info
Received: 13 May 2019 | Revised Submission: 20 May 2019 | Accepted: 28 May 2019 | Available Online: 15 Jun 2019

Development of Android Application for Device to Device Communication in IoT Using Rabbit MQ Broker

Deekshitha Arasa and SP Meharunnisa***

ABSTRACT

This paper provides information about using RabbitMQ broker as a real time communication medium for IoT applications. The ability to gather relevant real time information is the main key of the intelligent IoT communication. This can be done by using MQTT protocol which is emerging as an effective "machine to machine" communication protocol for IoT world of connected devices and for mobile applications where bandwidth and battery power are at the premium.

Keywords: *Internet of Things (IoT); Rabbit MQ; MQTT; Intelligent Communicating; Protocol.*

1.0 Introduction

Education IoT applications must be reactive and asynchronous. They should be capable of handling many devices and all messages ingested from them. Asynchronous messaging enables flexibility i.e. an application can send a message out and then it can keep working on the other things where as in synchronized messaging it has to wait for a response in real time. We can write the message to a queue and let the same business logic happen later, no need to wait for the web service to take action.

2.0 Message Queuing for IoT with Rabbit MQ

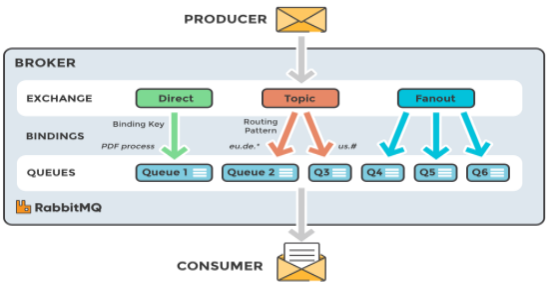
2.1 Introduction to rabbit MQ

RabbitMQ is open source message broker software. It accepts messages from producers (publishers) and provides them to consumers (Subscribers). RabbitMQ acts as a gateway for MQTT, AMQP, STOMP and HTTP protocols.

2.2 Structure of rabbit MQ communication

Fig 1 shows how a message is being passed from producer to consumer through RabbitMQ broker. The client applications called producers create messages and deliver them to the broker (the message queue)[1]. Consumers connect to the queue and subscribe to the messages to be processed. A software in a device can be a producer, or consumer, or both a consumer and a producer of messages. Messages placed onto the queue are stored until the consumer gets them.

Fig 1: Message Flow in Rabbit MQ



Instead of sending messages directly to the queue, the producer sends messages to an exchange. With the help of bindings and routing keys, an exchange accepts messages from the producer application and routes them to message queues. A binding is a link between a queue and an exchange [1].

Steps describe the message flow in RabbitMQ:

1. The producer publishes a message to an exchange. There are different types of exchanges: Direct, Topic and Fanout. Type of exchange should be specified clearly.

**Corresponding Author: Department of Computer Engineering, Dayananda Sagar College of Engineering, Bangalore, Karnataka, India (E-mail: deekshitha.arasa@gmail.com)*
**Department of Computer Engineering, Dayananda Sagar College of Engineering, Bangalore, Karnataka, India (E-mail: meharbhakshi2013@gmail.com)*

2. The exchange receives the message and is now responsible for the routing of the message.
3. Bindings need to be created from the exchange to queues.
4. The messages stay in the queue until they are handled by a consumer.
5. The consumer handles the message.

2.3. Rabbit MQ and server concepts

The following are the main concepts we need to know before we dig deep into RabbitMQ

1. Producer: Application that sends messages.
2. Consumer: Application that receives messages.
3. Queue: Buffer that stores messages.
4. Message: Information that is passed from producer to consumer through RabbitMQ.
5. Connection: A connection is a TCP connection between your application and the RabbitMQ broker.
6. Channel: A channel is a virtual connection inside a connection. Through the channel, messages are published or consumed from the queue.
7. Exchange: Receives messages from producers and pushes them to queues depending on rules defined by the exchange type. A queue needs to be bound to at least one exchange to receive messages.

Following are the types of Exchanges:

a) Direct: It delivers messages to queues based on a message routing key. Here, the binding key of the message should exactly match to the routing key of the message.
b) Fanout: It routes messages to all of the queues that are bound to it.
c) Topic: The topic exchange does a wildcard match between the routing key and the routing pattern specified in the binding.
d) Headers: Headers exchanges use the message header attributes for routing.

8 Binding: A binding is a link between queue and exchange.

9 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 acts like an *address* for the message.

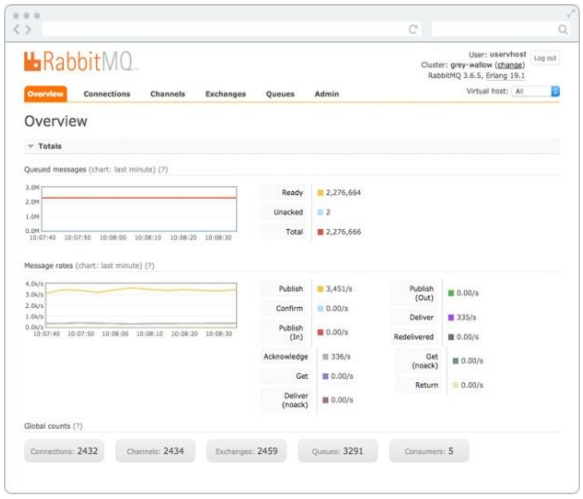
10 Users: It is possible to connect to Rabbit MQ with a given username and password.

11 vhost, Virtual host: A Virtual host provides a way to segregate applications using the same Rabbit MQ instance. Different users can have different access privileges to different vhost and queues and exchanges can be created, so they only exist in one vhost.

3.0 Working with Rabbit MQ Broker

First you need to install Rabbit MQ server. By default it is available in the API local host. In your web browser you need to type http:localhost:15672. You will be redirected to rabbit MQ server login page. By default, the username and password are set to guest. We can change it if we desire to do so.

Fig 2: Home page of RabbitMQ server



RabbitMQ provides a web UI for managing and monitoring your RabbitMQ server. Fig 2 shows a homepage of a RabbitMQ server. Queues, Exchanges, bindings, users can be created and managed using the Web UI.

3.1 Publish and subscribing messages using RabbitMQ

To communicate with RabbitMQ we need a library that understands the same protocol as RabbitMQ. We need to download the client-library for the programming language that you is required for our applications. A client-library is an applications programming interface (API) for writing client applications. A client library has several methods that can be used to communicate with RabbitMQ. The methods should be used when we want to connect to the RabbitMQ broker (using the given parameters, hostname, port number, etc or when we declare a queue or an exchange.

Steps to follow when setting up a connection and publishing a message/consuming a message: [2]

1. First of all, we need to set up/create a connection object. This is the place whare the username, password, connection URL, port etc, will be specified. Between the application and RabbitMQ, a TCP connection will be built when the *start* method is called.
2. A channel needs to be opened and created in the TCP connection. To open a channel, a connection interface can be used and it can be used to send and receive messages.
3. Declare/create a queue. Declaring a queue will create a queue if it does not already exist. All queues need to be declared before using them.
4. In subscriber/consumer: Set up exchanges and bind a queue to an exchange. All exchanges need to be declared before using them. An exchange is responsible for accepting messages from a producer application and routing them to message queues. For messages to be routed to queues, it is necessary for the queues to bind to an exchange.
5. In publisher: Publish a message to an exchange.
6. In subscriber/consumer: Consume a message from a queue.
7. Close the channel and the connection.

3.1 Basic Set up for a java client

3.2.1 For a java client, the Maven dependency would be

```
<dependency>
<groupId>com.rabbitmq</groupId>
<artifactId>amqp-client</artifactId>
<version>4.0.0</version>
</dependency>
```

3.2.2 After running the RabbitMQ broker, we need to establish connection with the java client

```
ConnectionFactory factory = new Connection Factory();
factory.setHost("localhost");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();
factory.setPort(15678);
factory.setUsername("user1");
factory.setPassword("MyPassword");
```

We can use *setPort* to set the port if the default port is not used by the RabbitMQ Server; the default port for RabbitMQ is *15672:*

3.3 To set producer

From the producer side, declare the queue as follows:

```
channel.queueDeclare
("<Queuename>", false, false, false, null);
String message =
"<Queuename>";
channel.basicPublish
("", "<Queuename", null, message.getBytes());
```

Then close the channel and connection.

```
channel.close();
connection.close();
```

3.4 To set up the consumer

Declare the same queue from the consumer side as follows:

```
channel.queueDeclare
("<Queuename", false, false, false, null);
```

Then declare the consumer that will process messages fom the queue asynchronously.

```
Consumer consumer = new DefaultConsumer(channel)
{
@Override
public void handleDelivery(
String consumerTag,
Envelope envelope,
AMQP.BasicProperties properties,
byte[] body) throws IOException {
String message =
new String(body, "UTF-8");
// process the message
}
};
channel.basicConsume("products_queue", true, consumer);
```

4.0 Conclusions

RabbitMQ is a message broker that takes messages and sends them to other places in a pretty smart way.

It is completely language-neutral and while using it you can write and read to them in any language just like you would while using TCP or HTTP.

RabbitMQ runs on all major operating systems and supports a large number of developer platforms such as java, .NET, Python, PHP, Erlang and many more.

References

1. S Shailesh, K Joshi, K Purandare. Performance Analysis of RabbitMQ as a message bus, Department of Computer Engineering, VJTI, Mumbai, Maharashtra, India

2. O Bello, S Zeadally. Intelligent Device-to-Device Communication in the Internet of Things, IEEE, and Sherali Zeadally, Senior, IEEE, 10(3), 2016, 1172-1182