TUTORIAL 2: Getting started with MQTT

Last revision: January 26th, 2022

1 INTRODUCTION & MOTIVATION

A pervasive system must often report events and actuate upon their receival. Some examples: when a user leaves his/her bed, the guiding lights turn on; or when a user leaves the stove unattended an intelligent power plug turns it off. The way an event communication can be achieved in several ways, and one good choice is to use message queue technology and in this course we will present the MQTT[1] technology.

In this tutorial you will learn how to set up a MQTT broker, specifically how to install, configure and test the Mosquito MQTT broker. You'll also develop a simple publisher and subscriber in Python, using the Paho MQTT library. There are API's for most type of programming languages, incl. Java, C#, C, C++, Perl.

By the end of this tutorial, you should have a broker installed in your computer and/or on the Raspberry Pi that you configured in the previous tutorial. Also, you should have completed a simple publisher and subscriber Python program that will let you get started writing your project applications.

2 WHAT TO READ BEFORE THE TUTORIAL?

It is expected that you have a basic knowledge of the following topics:

- The publish/subscribe model, although a brief introduction is presented in section 3, and how it compares with other communication models such as HTTP;
- What the Internet of Things (IoT) is and how does the publish/subscribe model fit.

3 MQTT: A BRIEF INTRODUCTION

In this section a brief introduction to MQTT will be presented. By no means it contains everything you need to know about MQTT, but it is enough to get you started.

The MQTT protocol is a good fit for the IoT for two reasons: 1) the client implementation has a small footprint, which enables its implementation in devices with little resources (such as microprocessor-based sensors); and 2) the bandwidth consumption is lower in

[1] MQTT used to stand for Message Queuing Telemetry Transport, but it is not considered an acronym anymore.

comparison with other protocols such as HTTP. We recommend reading [1-3] for further information.

MQTT is a protocol based on the publish/subscribe model, where clients[2], "**the publishers**", send messages (can also be called events) with categorized data that is received and processed by other clients that subscribe to it: "**the subscribers**". The communication between publishers and subscribers is managed by a server also called "**broker**", whose responsibility is to distribute the published messages to the subscribers. Note that the clients never communicate directly between themselves; they always need a broker as intermediary.

The categorization of messages is made using "**topics**". Topics are an essential part of MQTT, as they allow the broker to filter messages based on its topic, that then are sent to the subscribers who requested them. Follows some examples of topics:

```
powerplug/state
powerplug/power
powerplug/state/set
```

In the above examples, you can see that topics have different levels: `powerplug/state` and `powerplug/power` have two levels, while `powerplug/state/set` has three levels – the levels are always separated by a "/" (slash). This allows creating hierarchies within the topics and allows the subscribers to only receive messages they are interested in. For example, a subscriber might only be interested in receiving the events that report the power of the plug, which is published within the topic `powerplug/power`. Other subscribers might want to receive all topics of the power plug; in this case they should subscribe to the topic `powerplug/#`, where hash ('`#`') is a wild card that indicates to subscribe to all topics whose top level is `powerplug`. The way topics are structured depends on the application being built. More information about the topics subject can be found in [4].

One last aspect of MQTT is the payload of the messages. These can have any format: JSON, XML, non-structured text, binary, etc. This is highly application dependent, so it won't be addressed here.

In Error: Reference source not found it is depicted a simplified MQTT architecture with four sensors publishing events (in the left) and five subscribers: two sensors (a light and a power plug, in the left) and three generic clients, in the right. As you can see, all events the sensors publish and subscribe to have a topic associated. The published events are depicted as "`<topic> <payload>`". For example, the events published by the PIR sensor have the

[2] According with the MQTT standard [1], the terminology client and server is also used to refer, respectively, to the publishers/subscribers and brokers.

topic `pir/presence` and the payload `{"presence": true}`. In this example, all messages are in JSON format and is up to the subscribers to parse them.

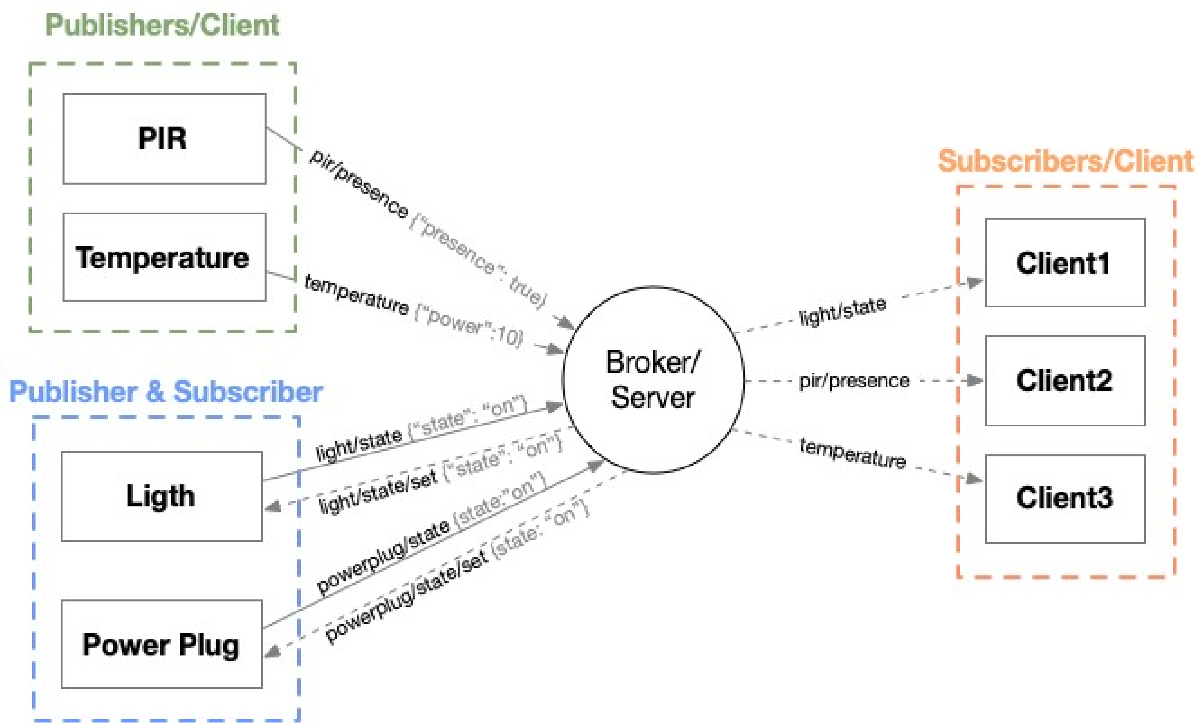


Figure . Architecture of a MQTT system.

The two clients in the blue box, specifically the light and the power plug, are both publishers and subscribers. The power plug publishes messages to the topic `powerplug/state` and it subscribed to messages with the topic `powerplug/state/set` – here we can assume that this topic can be used to turn the plug on/off. The power plug publishes events to the topic `powerplug/state`, but since this topic is not subscribed by any client, the broker will never send them, until a client subscribe to the topic.

Finally, three generic subscribers are presented in the orange box in the right. As you can see, Client1 subscribed to messages that are published to the topic `light/state`, Client2 subscribed to messages published to `pir/presence`, and Client3 subscribed to messages published to the topic `temperature`. Whenever the sensors publish their events, the subscribers will receive them and can perform the correspondent actions.

4 MATERIALS

You will need the following materials:

- Eclipse Mosquitto (you can download it in https://mosquitto.org/download/)

- MQTT Explorer (you can download it in http://mqtt-explorer.com)
- Paho MQTT library (installed via PIP)
- CEP2 MQTT Examples: https://github.com/jmiranda-au/cep2mqttexamples

5 MOSQUITTO BROKER

In this section you will install, configure and test a MQTT broker: Eclipse Mosquitto. It can be natively installed in Windows, macOS and Linux. Here only the installations for Windows and Linux are addressed, since these are the two platforms you will most likely use. Besides installing it natively, you can run Mosquitto in a Docker container [5]. We will not cover this option here, but it can be useful (and simpler) in some systems, such as macOS, although there are Docker related details that you have to learn first in order to interface with Mosquitto running in a container.

5.1 Windows

In Windows download the executable and install Mosquitto. It is recommended that you do the standard installation and try to set Mosquitto to run as a service. If you don't do this, then you'll have to start Mosquitto manually every time you start your computer.

IMPORTANT: version 2.x of Mosquitto requires some modifications in the configuration file. Specifically, you will have to add the configuration `allow_anonymous true`, so that clients can access the broker without authentication. More information can be found in [6].

5.2 Linux

The installation process depends on the distribution you are using. For Debian based distributions, such as Ubuntu or Raspberry Pi OS, you can use the command `sudo apt-get install mosquitto mosquitto-clients` (note the `sudo` for administrator permissions). The second package, `mosquitto-clients`, is not mandatory, but its installation is advisable, since it installs MQTT clients that will later allow you to test the broker.

After you install the broker, it should automatically be running as a service (or daemon). To check if it is running, you can execute the command `sudo systemctl status mosquitto`. If it isn't, you can start Mosquitto using the command `sudo systemctl start mosquitto` – if you replace `start` with `stop`, the service will stop running.

5.3 Testing your broker

To test that your broker is working properly, you can use generic MQTT clients for publishing and subscribing. Three clients will be presented: the Mosquitto clients (`mosquitto_pub` and `mosquitto_sub`) and MQTT Explorer. Both clients are available for Windows and Linux.

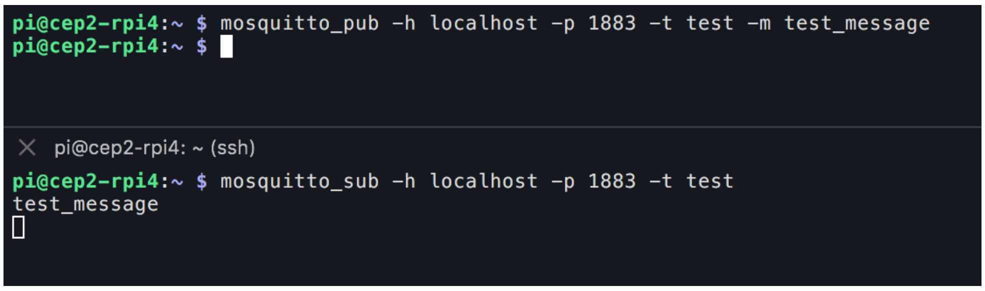
The Mosquitto broker used for demonstration is running on a Raspberry Pi, as well as the Mosquitto clients, while MQTT Explorer is running on a remote computer that connects to the broker running in the Raspberry Pi.

5.3.1 mosquitto_pub and mosquitto_sub

The Mosquitto clients are two simple command line utilities that can be used for publishing and subscribing to/from a MQTT broker. An example usage is depicted in Figure 1, with the publisher on top (`mosquitto_pub`) and the subscriber on the bottom (`mosquitto_sub`).

The commands' parameters are similar, with the exception of the message for the publisher:

- **-h:** the host where the client must connect. In the example the clients are connected to the localhost interface (127.0.0.1), which is where the broker is running. You use either a host name or an IP address;
- **-p:** the port where the client will connect. The default MQTT port is 1883. Unless the broker listens on another port, this parameter can be omitted;
- **-t:** the topic where the client will publish or subscribe. In the example it is `test`;
- **-m:** the message to be published. This option only applies to `mosquitto_pub`. In the example above the message is `test_message`. If you want to send a message with special characters, you should put them inside quotation marks (" or '). For example, for sending a message with a JSON format, the parameter would have to be `-m '{"message":"test_message"}'` – note the single and double quotation marks!



```
pi@cep2-rpi4:~ $ mosquitto_pub -h localhost -p 1883 -t test -m test_message
pi@cep2-rpi4:~ $
```

```
pi@cep2-rpi4: ~ (ssh)
pi@cep2-rpi4:~ $ mosquitto_sub -h localhost -p 1883 -t test
test_message
```

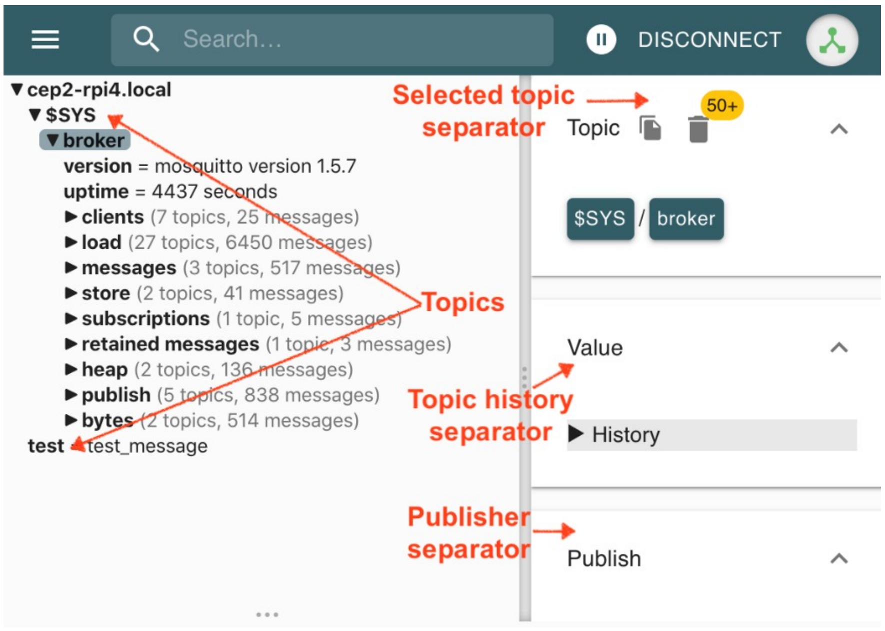
Figure 1. Example usage of Mosquitto clients. On top the publisher (mosquitto_pub) and the subscriber (mosquitto_sub) on the bottom.

5.3.2 MQTT Explorer

While the Mosquitto clients are useful for simple tests or sensors simulation, MQTT Explorer offers other features that are useful during the development of applications, such as subscribing to multiple topics and tracking topics messages' history.

The interface of MQTT Explorer is depicted in Figure 2. The left separator displays the topics MQTT Explorer is subscribing (by default it subscribes to all topics that are received in the broker), as well as the last message published to that topic. For example, for the topic test, the last published message was `test_message` (this message is the message that was published by `mosquitto_pub` in the example in section 5.3.1).

On the right, three separators are depicted: selected topic, topic history and publisher. The selected topic separator displays the levels of the topic currently selected in the topics window. The topic history separator displays the history of the messages received in that topic. Finally, in the publisher separator it is possible to publish messages to any topic defined by the user.



```
▼cep2-rpi4.local
  ▼$SYS
    ▼broker
      version = mosquitto version 1.5.7
      uptime = 4437 seconds
      ▶clients (7 topics, 25 messages)
      ▶load (27 topics, 6450 messages)
      ▶messages (3 topics, 517 messages)
      ▶store (2 topics, 41 messages)
      ▶subscriptions (1 topic, 5 messages)
      ▶retained messages (1 topic, 3 messages)
      ▶heap (2 topics, 136 messages)
      ▶publish (5 topics, 838 messages)
      ▶bytes (2 topics, 514 messages)
  test = test_message
```

Figure 2. MQTT Explorer client.

5.4 A simple publisher/subscriber in Python

In this section two simple Python clients, a publisher and a subscriber, will be presented. These can be cloned from the Git repository indicated in section 4 (Materials). All examples are in Python3. The clients use the Paho MQTT library which is the most widely used MQTT library in Python. It can be installed via PIP with the following command: `sudo pip3 install paho-mqtt`. The documentation of the library can be found in [7].

A basic publisher could be written as follows:

```
import paho.mqtt.client as mqtt

client = mqtt.Client()
client.connect("localhost", 1883)
client.publish("testtopic", "test message")
client.disconnect()
```

The client must first connect to the broker (in this case it is running in the same computer) and then, when successfully connected, it can publish messages. In this example it publishes “`test message`” to the topic “`testtopic`”.

A basic subscriber could be written as follows:

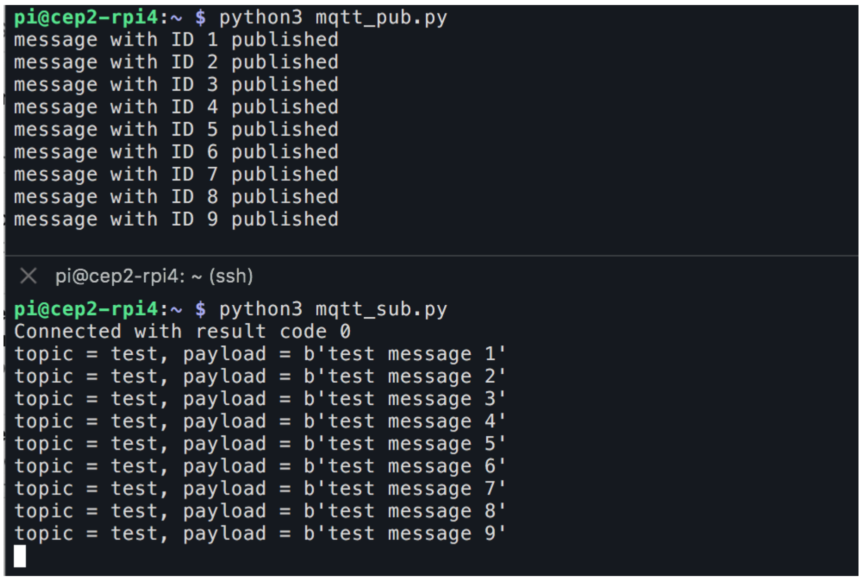
```
import paho.mqtt.client as mqtt

def on_message(client, userdata, msg):
    print(f"topic = {msg.topic}, payload = {msg.payload}")

client = mqtt.Client()
client.on_message = on_message
client.connect("localhost", 1883)
client.subscribe("testtopic")
client.loop_forever()
```

Like the publisher, the client must first connect, then subscribe to the topics it wants to receive (in this example “`testtopic`”) and then wait for receiving messages. When a message is received, the callback defined with the property `client.on_message` will be invoked. In this case it only prints the topic and the message. The `loop_forever()` function blocks the application to keep listening for incoming messages.

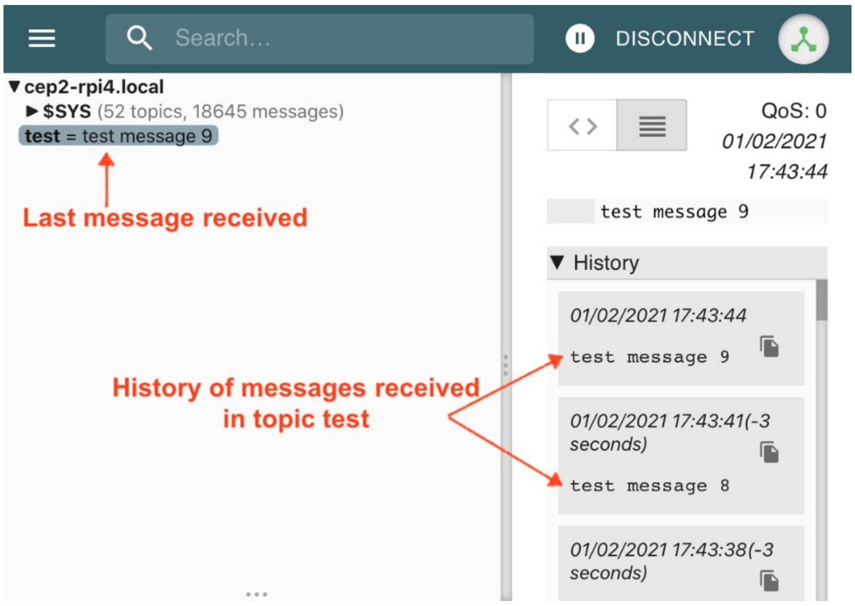
Attached to this tutorial follow two more extensive examples, `mqtt_pub.py` and `mqtt_sub.py`, that are respectively a publisher and a subscriber. More details about Paho MQTT are addressed in the comments of the code. In Figure 3 and Figure 4 it is depicted the output of the `mqtt_pub.py` and `mqtt_sub.py` clients and MQTT Explorer.



```
pi@cep2-rpi4:~ $ python3 mqtt_pub.py
message with ID 1 published
message with ID 2 published
message with ID 3 published
message with ID 4 published
message with ID 5 published
message with ID 6 published
message with ID 7 published
message with ID 8 published
message with ID 9 published
```

```
pi@cep2-rpi4: ~ (ssh)
pi@cep2-rpi4:~ $ python3 mqtt_sub.py
Connected with result code 0
topic = test, payload = b'test message 1'
topic = test, payload = b'test message 2'
topic = test, payload = b'test message 3'
topic = test, payload = b'test message 4'
topic = test, payload = b'test message 5'
topic = test, payload = b'test message 6'
topic = test, payload = b'test message 7'
topic = test, payload = b'test message 8'
topic = test, payload = b'test message 9'
```

Figure 3. Output of mqtt_pub (top) and mqtt_sub (bottom).



▼cep2-rpi4.local
▶ $SYS (52 topics, 18645 messages)
test = test message 9

QoS: 0
01/02/2021
17:43:44

test message 9

▼ History

01/02/2021 17:43:44
test message 9

01/02/2021 17:43:41(-3 seconds)
test message 8

01/02/2021 17:43:38(-3 seconds)

Figure 4. MQTT Explorer with the messages published by mqtt_pub.

6 NEXT STEPS

6.1 MQTT features

Several aspects of MQTT were not addressed in this tutorial, specifically: QoS (Quality of Service), retained messages, authentication and security, etc. From these three, QoS has an important role in message delivery to the subscribers. For example: the Python publishers use a QoS of 0 (Once - not guaranteed) to deliver the messages. What is the guarantee that the messages will be delivered? If your application is critical, is this level acceptable?

Other features of the protocol should be considered based on the requirements of your project. More information about these MQTT features can be found in [2-3].

6.2 Paho MQTT

The companion Python clients are very simple and are by no means enough for an application that is required to be always running. One example: how can a disconnection be handled using the library (tip: the callbacks play an essential role in this)?

The MQTT clients provided to you are either publisher or subscriber. But, what if your application needs to be a publisher and a subscriber? How can you merge them in the same application? What other client functions of the library will you have to use?

7 FURTHER READING

It is recommended that you read more about MQTT and its architectural components in [8], where you can also find more information and examples on how to use Paho MQTT.

8 REFERENCES

[1] MQTT Version 3.1.1 Plus Errata 01: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.html - _Toc442180821

[2] MQTT for Beginners Tutorials and Course: http://www.steves-internet-guide.com/mqtt-basics-course/

[3] HiveMQ's MQTT Essentials: https://www.hivemq.com/tags/mqtt-essentials/

[4] Designing MQTT Topics for AWS IoT Core: https://docs.aws.amazon.com/whitepapers/latest/designing-mqtt-topics-aws-iot-core/designing-mqtt-topics-aws-iot-core.html

[5] Running the eclipse-mosquitto MQTT Broker in a docker container: https://blog.feabhas.com/2020/02/running-the-eclipse-mosquitto-mqtt-broker-in-a-docker-container/

[6] Mosquitto MQTT Broker: http://www.steves-internet-guide.com/mosquitto-broker/

[7] Paho MQTT Python Client's documentation: https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php

[8] Beginners Guide To The MQTT Protocol: http://www.steves-internet-guide.com/mqtt/