Mosquitto MQTT - IoT Platform Series

Know and Understand MQTT protocol. Then implement and build a project to subscribe and publish on the public server of test.mosquitto.org.

Difficulté **Moyen** | Durée **1 heure(s)** | Catégories **Électronique, Robotique**

Coût **5 USD ($)**

Sommaire

Introduction

MQTT stands for Message-Queue-Telemetry-Transport, is a **publish/subscribe** protocol for **machine-to-machine** communication. This simple protocol, is easy to implement for any client. Termed as the **Pub** and **Sub,** both are used for same purpose but with different methods.

Matériaux

- Mosquitto_MQTT_-_IoT_Platform_Series_mosquitto_mqtt_using_pubsub_client_library.ino

Outils

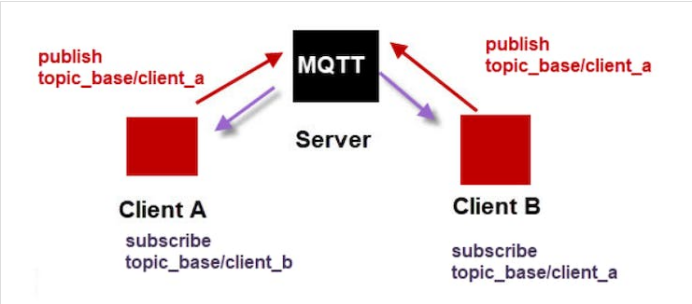
Étape 1 - Getting Started

Here, there are 2 sections - **Publish and Subscribe**. And then there is a middleman - **Broker**. Let us see in depth

- IoT Devices play the role to collect sensor data and send to the cloud (broker). While **PC / Server / Mobile devices** play the role to monitor and receive the sensor data to be viewed - Here, **IoT Device** is a **Publisher**, and **PC Devices** are **Subscriber**.

> [EXAMPLE] When a **user1** publishes an image on social media, then only the **user2** subscribed to **user1** can view/receive the image. Here, the **user1** is the **PUBLISHER**, **user2** is the **SUBSCRIBER**, and the **user1's account** is the **BROKER**.

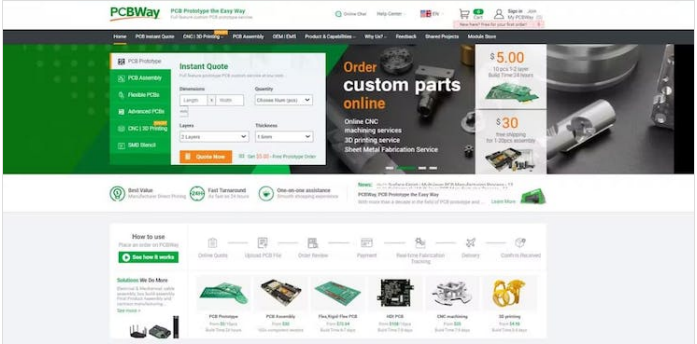
- According to the above analogy, the image that is published is the data, that was **transferred from user1 to user2** . And that is the exact scenario in an MQTT Pub/Sub model.
- We have a more secure layer to make sure the data is shared t **hrough a specific path, we call th**at 'topic', When user**1** publishes data o topic, the subscriber automatically receives if already connected to the broker. Hence, **the** LOW latency.



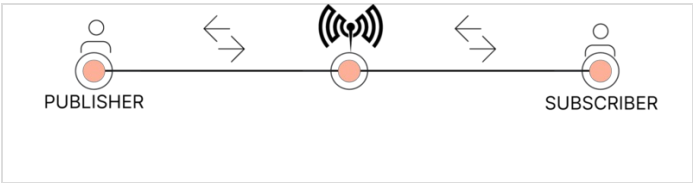
Étape 2 - Get PCBs for Your Projects Manufactured

You must check out PCBWAY for ordering PCBs online for cheap!

You get 10 good-quality PCBs manufactured and shipped to your doorstep for cheap. You will also get a discount on shipping on your first order. Upload your Gerber files onto PCBWAY to get them manufactured with good quality and quick turnaround time. PCBWay now could provide a complete product solution, from design to enclosure production. Check out their online Gerber viewer function. With reward points, you can get free stuff from their gift shop.



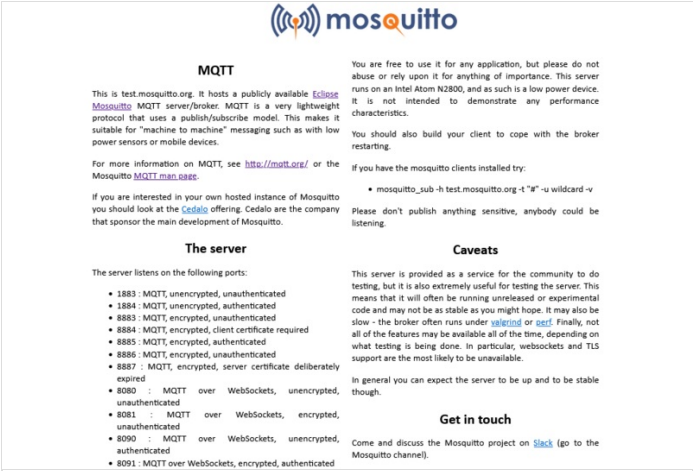
Étape 3 - MQTT Broker



Whenever there is a pub-sub model used as a message communication protocol, we require a broker that can transfer the information in the required device. This can be done by sending the message under correct topic.
Let us understand this -

- A Broker is a runtime server (continuously running service), which can have 2 types of clients - **Publisher (seller)** & **Subscriber (buyer)**
- For instance, when a seller **sells a product** through a broker to a buyer, then it is using the Broker's service to reach & find a secured buyer.
- Similarly, when publisher publishes a piece of information, the data reaches to the subscriber through the Broker.
- The broker is responsible for having specific storage space where it can expect data from the publisher to store temporarily and then send to the subscriber.
- In the pub-sub MQTT, clients talk to each other through an MQTT broker.
- There are many MQTT Brokers in the market. It is even possible to create our own broker, or use an open-source broker '**paho**'.
- For the current project, we shall first understand the mechanism and then watch a trial movement of data on **Mosquitto MQTT Broker.**

Étape 4 - Mosquitto Platform



mosquitto

MQTT

This is test.mosquitto.org. It hosts a publicly available Eclipse Mosquitto MQTT server/broker. MQTT is a very lightweight protocol that uses a publish/subscribe model. This makes it suitable for "machine to machine" messaging such as with low power sensors or mobile devices.

For more information on MQTT, see http://mqtt.org/ or the Mosquitto MQTT man page.

If you are interested in your own hosted instance of Mosquitto you should look at the Cedalo offering. Cedalo are the company that sponsor the main development of Mosquitto.

The server

The server listens on the following ports:

- 1883 : MQTT, unencrypted, unauthenticated
- 1884 : MQTT, unencrypted, authenticated
- 8883 : MQTT, encrypted, unauthenticated
- 8884 : MQTT, encrypted, client certificate required
- 8885 : MQTT, encrypted, authenticated
- 8886 : MQTT, encrypted, unauthenticated
- 8887 : MQTT, encrypted, server certificate deliberately expired
- 8080 : MQTT over WebSockets, unencrypted, unauthenticated
- 8081 : MQTT over WebSockets, encrypted, unauthenticated
- 8090 : MQTT over WebSockets, unencrypted, authenticated
- 8091 : MQTT over WebSockets, encrypted, authenticated

You are free to use it for any application, but please do not abuse or rely upon it for anything of importance. This server runs on an Intel Atom N2800, and as such is a low power device. It is not intended to demonstrate any performance characteristics.

You should also build your client to cope with the broker restarting.

If you have the mosquitto clients installed try:

- mosquitto_sub -h test.mosquitto.org -t "#" -u wildcard -v

Please don't publish anything sensitive, anybody could be listening.

Caveats

This server is provided as a service for the community to do testing, but it is also extremely useful for testing the server. This means that it will often be running unreleased or experimental code and may not be as stable as you might hope. It may also be slow - the broker often runs under valgrind or perf. Finally, not all of the features may be available all of the time, depending on what testing is being done. In particular, websockets and TLS support are the most likely to be unavailable.

In general you can expect the server to be up and to be stable though.

Get in touch

Come and discuss the Mosquitto project on Slack (go to the Mosquitto channel).

Now that we understand how MQTT works, let us use a cloud MQTT service and send data across the internet. In this article, we'll be using Mosquitto MQTT - **test.mosquitto.org**
Under the Server section, we can see different ports provide feature-separated servers. These servers act like channels for sharing data over the cloud. Let us understand it first -

- **MQTT Broker Port (default: 1883):** This is the standard port used for MQTT communication. MQTT clients use i to connect to the Mosquitto broker and publish/subscribe to topics. It operates over TCP.
- **MQTT Broker SSL/TLS Port (default: 8883):** This is the secure version of the MQTT broker port. It uses SSL/TLS encryption to provide secure communication between MQTT clients and the Mosquitto broker. Clients connect to this port to establish a secure connection.
- **WebSocket Port (default: 9001):** Mosquitto also supports MQTT over WebSockets, allowing MQTT clients to connect to the broker using the WebSocket protocol. The WebSocket port is used for WebSocket-based MQTT communication.
- **WebSocket SSL/TLS Port (default: 9443):** This is the secure WebSocket port used for encrypted WebSocket-based MQTT communication. It provides a secure connection using SSL/TLS encryption.

We shall be using **1883** port to send data and monitor. As we know, MQTT has 3 services - Publisher, Broker, and Subscriber. In this case, mosquito MQTT Cloud is already playing the role of a broker.
Now, we'd be using **ESP32 Dev Board**, which has a wifi chip and is able to connect to the Internet, playing the **role of a Publisher** for sharing its temperature and humidity data from the sensor.
On the other hand, we shall use the PC to view this data as a **Subscriber.** This will enable us to fully understand the working principle of the MQTT protocol used in IoT Communication between devices.

Étape 5 - Publisher (ESP32)

PubSubClient To set up the ESP32 for MQTT, we need to install a library - . This library has functions that use variables as mentioned below to send data to the broker.

mqtt_server : This variable represents the address or IP of the MQTT broker. We shall be using "test.mosquitto.org"

mqtt_port : This variable represents the port number of the MQTT broker. In our case 1883.

mqtt_topic : This variable represents the topic to which the publisher will send messages. For Example "schoolofiot/device1". Where '**schoolofiot**' is the general-most topic level. And '**device1'** is a sub-level.

The provided code is an Arduino sketch that uses the ESP32 WiFi module and the PubSubClient library to connect to an MQTT broker and publish temperature and humidity data. Let's break down the code step by step:

1. Include necessary libraries:

```
#include <WiFi.h>
#include <PubSubClient.h>
```

This code includes the required libraries for the ESP32 WiFi module and the MQTT client functionality.

2. Define WiFi & MQTT Server variables:

```
const char* ssid = "XXXXXXXXXX";
const char* password = "XXXXXXXXXX";
const char* mqtt_server = "test.mosquitto.org";
```

mqtt_server These variables store the SSID (network name) and password for the WiFi network you want to connect to. The variable holds the IP address or hostname of the MQTT broker.

3. Declare global variables and objects:

```
WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
float temperature = 0;
float humidity = 0;
```

Here, a WiFi client object (`espClient`) and an MQTT client object (`client`) are declared. The variable `lastMsg` stores the timestamp of the last message, and the `msg` is a character array for message storage. The `value`, `temperature`, and `humidity` variables are used to hold the respective sensor values.

4. Setup function:

```
void setup()
Serial.begin(115200);
setup_wifi();
client.setServer(mqtt_server, 1883);
client.setCallback(callback);
}
```

`setup()` The function is called once at the start of the program. It initializes the serial communication, sets up the WiFi connection, configures the MQTT server and port, and sets the callback function to handle incoming messages.

5. WiFi setup function:

```
void setup_wifi() {
// ...
}
```

`setup_wifi()` The function handles the connection to the WiFi network using the provided SSID and password. It waits until the connection is established and prints the local IP address to the serial monitor.

6. MQTT callback function:

```
void callback(char* topic, byte* message, unsigned int length) {
```

```
// ...
```

```
}
```

This function is called when a message is received from the MQTT broker. It prints the received message along with the corresponding topic.

7. MQTT reconnection function:

```
void reconnect() {
// ...
}
```

`reconnect()` The function is responsible for reconnecting to the MQTT broker if the connection is lost. It attempts to connect to the broker using a randomly generated client ID. If the connection is successful, it prints a success message. Otherwise, it waits for 5 seconds before retrying.

8. Main loop:

```
void loop() {
if (!client.connected()) {
reconnect();
}
client.loop();
long now = millis();
if (now - lastMsg > 2000) {
lastMsg = now;
sendData();
}
}
```

`loop()` `setup()` The function is the main program loop that runs continuously after the function. It checks if the MQTT client is connected and, if not, attempts to reconnect. It also calls the function to maintain the MQTT client's internal state.

`client.loop()` `sendData()` Every 2 seconds, it calls the function to publish temperature and humidity data.

9. Publish sensor data function:

```
void sendData() {
// ...
}
```

`sendData()` The function is responsible for publishing temperature and humidity data to specific MQTT topics. It generates random values for temperature and humidity, converts them to strings, and publishes them along with the corresponding topic.

- Publish a gap message:

```
client.publish("schoolofiot/gap", "----------");
```

`----------` This line publishes a message consisting of a series of dashes () to the MQTT topic "**schoolofiot/gap**". It is used to indicate a separation or gap between different sets of data.

- Read and publish temperature data:

```
temperature = random(30, 40);
char tempString[8];
dtostrf(temperature, 1, 2, tempString);
Serial.print("Temperature: ");
Serial.println(tempString);
String tempdata = "Temperature: " + String(tempString);
client.publish("schoolofiot/temperature", tempdata.c_str());
```

`temperature` `dtostrf()` These lines generate a random temperature value between 30 and 40 degrees, store it in the variable, and use function to convert decimal point data to String.

```
dtostrf(floatValue, minStringWidth, numAfterDecimal,
charBuf_to_store_string);
```

This function takes four parameters to convert double into an ASCII value stored inside string:

1. floatValue: The first parameter that takes the float value we want to convert into a string.

2. minStringWidth: This is the second parameter that defines the minimum field width of the output string.

3. numAfterDecimal: The third parameter is precision which describes the number of digits after the decimal point.

4. charBuffer: Final argument is where the string will be stored. This is a kind of char array having a defined size.

The temperature value is then printed to the serial monitor and concatenated with the string "**Temperature:** ". The resulting string is stored in the variable. Finally, the string is published to the MQTT topic **schoolofiot/temperature** using the function.

```
tempdata
tempdata
client.publish()
```

- Read and publish humidity data:

```
humidity = random(60, 70);
char humString[8];
dtostrf(humidity, 1, 2, humString);
Serial.print("Humidity: ");
Serial.println(humString);
"mString humdata = "Humidity: " + String(humString);
client.publish("schoolofiot/humidity", humdata.c_str());
```

```
humidity
sendData()
```

These lines generate a random humidity value between 60 and 70 percent, store it in the variable.

Overall, the function generates random temperature and humidity values, converts them to strings, and publishes them to specific MQTT topics for further processing or monitoring.

Final Code can be found in the Code section

But to confirm this, we also need to read the data from other the side - **Subscriber**.

Étape 6 - Subscriber (Windows PC)

To set up the Subscriber on PC, we need to install Mosquitto MQTT Applcation. This application can create a broker, publisher & subscriber - all sections

To install Mosquitto MQTT on your PC from the official website and make changes to the configuration file for listener 1883 and allow anonymous connections, you can follow these steps:

1. Download and Install Mosquitto:

- Go to the official Mosquitto website (https://mosquitto.org/).
- Navigate to the "*Downloads*" section.
- Choose the appropriate installer for your operating system (Windows x64 in this case) and download it
- Install the application in desired location.

2. Edit Configuration File:

- Open the installation directory where Mosquitto is installed.
- Locate the file (usually found in the main directory).

  mosquitto.conf

- Open in a text editor of your choice. Add the below 2 lines -

  mosquitto.conf

```
listener 1883
allow_anonymous true
```

- It should look somewhat like this -
- We can uncomment ad make changes in the file as well, but adding only 2 lines on the top is more simple and noticeable.

3. Run Mosquitto Subscriber

- **We** can run the Mosquitto broker and then subscribe to the topic we desire. But running directly the subscriber is best in our case.

  mosquitto_sub.exe

- Open the folder/directory where the mosquitto.exe along with is present.
- Run the ***PowerShell/CMD*** terminal from within the directory. For windows, *open the directory > Press shift + right-mouse-button(right-click)*, and we'd see options for running a terminal like powershell.
- On the terminal, enter below command -

```
> .\mosquitto_sub -h test.mosquitto.org -t "schoolofiot/#"
```

In the above command, if you noticed, I did not subscribe to a specific topic. As per the topics we published (from ESP32), like "**schoolofiot/gap**", "**schoolofiot/temperature**" or "**schoolofiot/humidity**".

- The reason is, gap, temperature & humidity comes under the ***general topic*** of **schoolofiot** level. So, to access/view any data published as a sub-level of **schoolofiot**, we can use '**#**'.
- Apart from this, in case we need to subscribe to a specific topic (like temperature), we can use command like this -

```
> .\mosquitto_sub -h test.mosquitto.org -t "schoolofiot/temperature"
```

Therefore, no matter what name is put under the ***general topic,*** we can subscribe to it and view all of them together.

Hurray! ⍰

We have learned another IoT Platform - Mosquitto MQTT (By Eclipse)