

## ESP8266: Connecting to MQTT broker

[ESP8266](#) / [117 Comments](#)

The objective of this post is to explain how to connect the ESP8266 to a MQTT broker, hosted on CloudMQTT.

### Introduction

The objective of this post is to explain how to connect the ESP8266 to a MQTT broker. If you are not familiar with the protocol, you can read more about [here](#).

Although this example should work fine with other brokers, we will assume that the broker will be hosted on [CloudMQTT](#).

Since CloudMQTT has a [free](#) plan, we can just create an account and test it. Setting an account is really simple and it's outside the scope of this post. You can check [here](#) how to do it and how to create a broker instance.

After completing the procedure, check the instance information page, which should be similar to the one shown in figure 1. The important credentials that we will be using on the ESP8266 code are the **server**, the **user**, the **password** and the **port**.

## Instance info

|                            |                   |
|----------------------------|-------------------|
| Server                     | m11.cloudmqtt.com |
| User                       |                   |
| Password                   |                   |
| Port                       | 12948             |
| SSL Port                   | 22948             |
| Websockets Port (TLS only) | 32948             |
| Connection limit           | 10                |

Figure 1 – CloudMQTT instance information.

In the ESP8266 side, we will be using an MQTT that supports the ESP8266, called [PubSubClient](#). The library can be installed via Arduino IDE library manager. The coding shown here is based on the examples provided in the library, which I encourage you to try.

Regarding the hardware, the tests shown on this tutorial were performed using a ESP8266 [NodeMCU](#) board.

## The code

First, we start by including the libraries needed for all the functionality. We need the **ESP8266WiFi** library, in order to be able to connect the ESP8266 to a WiFi network, and the **PubSubClient** library, which allows us to connect to a MQTT broker and publish/subscribe messages in topics.

```
1 #include <ESP8266WiFi.h>
2 #include <PubSubClient.h>
```

Then, we will declare some global variables for our connections. Naturally, we need the WiFi credentials, to connect to the WiFi network. You can check [here](#) a previous post explaining in detail how to connect to a WiFi network using the ESP8266.

```
1 const char* ssid = "YourNetworkName";
2 const char* password = "YourNetworkPassword";
```

We will also need the information and credentials of the MQTT server. As explained in the introduction, you will need to know the server address, the port, the username and the password. Use your values in the variables below.

```
1 const char* mqttServer = "m11.cloudmqtt.com";
2 const int mqttPort = 12948;
3 const char* mqttUser = "YourMqttUser";
4 const char* mqttPassword = "YourMqttUserPassword";
```

After that, we will declare an object of class [WiFiClient](#), which allows to establish a connection to a specific IP and port [1]. We will also declare an object of class [PubSubClient](#), which receives as input of the constructor the previously defined WiFiClient.

The constructor of this class can receive other number of arguments, as can be seen in the API [documentation](#).

```
1 | WiFiClient espClient;  
2 | PubSubClient client(espClient);
```

Now, moving for the setup function, we open a serial connection, so we can output the result of our program. We also connect to the WiFi network.

```
1 | Serial.begin(115200);  
2 | WiFi.begin(ssid, password);  
3 |  
4 | while (WiFi.status() != WL_CONNECTED) {  
5 |     delay(500);  
6 |     Serial.print("Connecting to WiFi..");  
7 | }  
8 |  
9 | Serial.println("Connected to the WiFi network");
```

Next, we need to specify the address and the port of the MQTT server. To do so, we call the [setServer](#) method on the PubSubClient object, passing as first argument the address and as second the port. These variables were defined before, in constant strings.

```
1 | client.setServer(mqttServer, mqttPort);
```

Then, we use the [setCallback](#) method on the same object to specify a handling function that is executed when a MQTT message is received. We will analyse the code for this function latter.

```
1 | client.setCallback(callback);
```

Now, we will connect to the MQTT server, still in the setup function. As we did in the connection to the WiFi network, we connect to the server in a loop until we get success.

So, we do a while loop based on the output of the [connected](#) method called on the PubSubClient, which will return true if the connection is established or false otherwise.

To do the actual connection, we call the [connect](#) method, which receives as input the unique identifier of our client, which we will call “ESP8266Client”, and the authentication username and password, which we defined early. This will return true on connection success and false otherwise

In case of failure, we can call the [state](#) method on the the PubSubClient object, which will return a code with information about why the connection failed [2]. Check [here](#) the possible returning values.

Check bellow the whole connecting loop.

```
1 | while (!client.connected()) {  
2 |     Serial.println("Connecting to MQTT...");  
3 |  
4 |     if (client.connect("ESP8266Client", mqttUser, mqttPassword)) {
```

```

5
6     Serial.println("connected");
7
8     } else {
9
10    Serial.print("failed with state ");
11    Serial.print(client.state());
12    delay(2000);
13
14    }
15 }

```

To finalize the setup function, we will publish a message on a topic. To do so, we call the `publish` method, which receives as input arguments the topic and the message. In this case, we will publish a “Hello from ESP8266” message on the “esp/test” topic.

```

1 | client.publish("esp/test", "Hello from ESP8266");

```

Then, we will subscribe to that same topic, so we can receive messages from other publishers. To do so, we call the `subscribe` method, passing as input the name of the topic.

```

1 | client.subscribe("esp/test");

```

Check the full setup function bellow.

```

1 | void setup() {
2
3     Serial.begin(115200);
4
5     WiFi.begin(ssid, password);
6
7     while (WiFi.status() != WL_CONNECTED) {
8         delay(500);
9         Serial.println("Connecting to WiFi..");
10    }
11    Serial.println("Connected to the WiFi network");
12
13    client.setServer(mqttServer, mqttPort);
14    client.setCallback(callback);
15
16    while (!client.connected()) {
17        Serial.println("Connecting to MQTT...");
18
19        if (client.connect("ESP8266Client", mqttUser, mqttPassword )) {
20
21            Serial.println("connected");
22
23        } else {
24
25            Serial.print("failed with state ");
26            Serial.print(client.state());
27            delay(2000);
28
29        }
30    }
31
32    client.publish("esp/test", "Hello from ESP8266");
33    client.subscribe("esp/test");
34
35 }

```

## The callback function

After the initialization, we need to specify the callback function, which will handle the incoming messages for the topics subscribed.

The arguments of this function are the name of the topic, the payload (in bytes) and the length of the message. It should return void.

In this function, we will first print the topic name to the serial port, and then print each byte of the message received. Since we also have the length of the message as argument of the function, this can be easily done in a loop.

Check bellow the whole function code.

```
1 void callback(char* topic, byte* payload, unsigned int length) {
2
3     Serial.print("Message arrived in topic: ");
4     Serial.println(topic);
5
6     Serial.print("Message:");
7     for (int i = 0; i < length; i++) {
8         Serial.print((char)payload[i]);
9     }
10
11     Serial.println();
12     Serial.println("-----");
13
14 }
```

## The main loop

In the main loop function, we will just call the `loop` method of the PubSubClient. This function should be called regularly to allow the client to process incoming messages and maintain its connection to the server [2].

```
1 void loop() {
2     client.loop();
3 }
```

Check the full code bellow

```
1 #include <ESP8266WiFi.h>
2 #include <PubSubClient.h>
3
4 const char* ssid = "YourNetworkName";
5 const char* password = "YourNetworkPassword";
6 const char* mqttServer = "m11.cloudmqtt.com";
7 const int mqttPort = 12948;
8 const char* mqttUser = "YourMqttUser";
9 const char* mqttPassword = "YourMqttUserPassword";
10
11 WiFiClient espClient;
12 PubSubClient client(espClient);
13
14 void setup() {
15
16     Serial.begin(115200);
```

```

17 WiFi.begin(ssid, password);
18
19
20 while (WiFi.status() != WL_CONNECTED) {
21     delay(500);
22     Serial.println("Connecting to WiFi..");
23 }
24 Serial.println("Connected to the WiFi network");
25
26 client.setServer(mqttServer, mqttPort);
27 client.setCallback(callback);
28
29 while (!client.connected()) {
30     Serial.println("Connecting to MQTT...");
31
32     if (client.connect("ESP8266Client", mqttUser, mqttPassword )) {
33
34         Serial.println("connected");
35
36     } else {
37
38         Serial.print("failed with state ");
39         Serial.print(client.state());
40         delay(2000);
41     }
42 }
43
44
45 client.publish("esp/test", "Hello from ESP8266");
46 client.subscribe("esp/test");
47
48 }
49
50 void callback(char* topic, byte* payload, unsigned int length) {
51
52     Serial.print("Message arrived in topic: ");
53     Serial.println(topic);
54
55     Serial.print("Message:");
56     for (int i = 0; i < length; i++) {
57         Serial.print((char)payload[i]);
58     }
59
60     Serial.println();
61     Serial.println("-----");
62
63 }
64
65 void loop() {
66     client.loop();
67 }

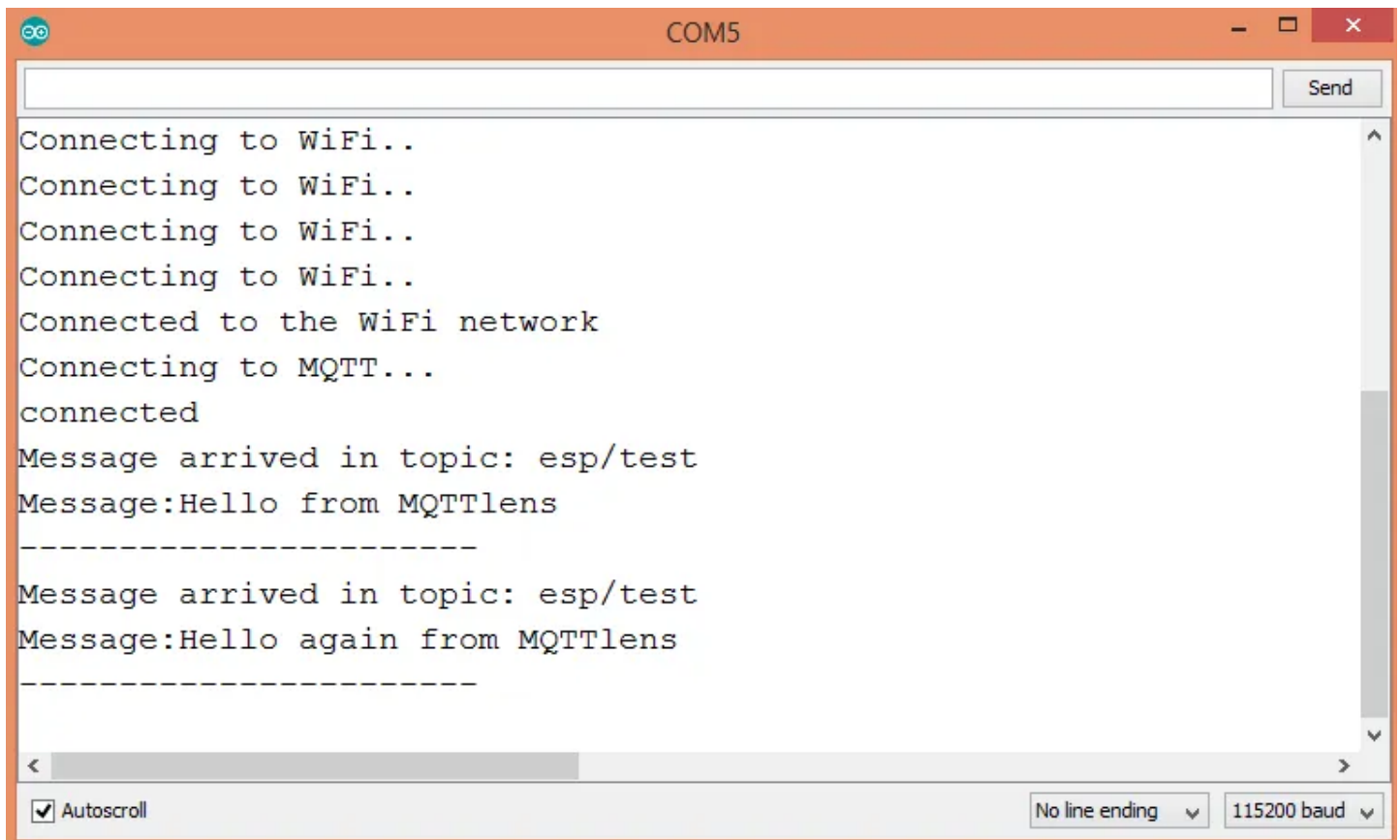
```

## Testing the code

First, make sure the MQTT server is running. Then, to test the code, just upload it and run it on your ESP8266. Open the Arduino IDE serial console, so the output gets printed.

Upon running, the ESP8266 will send the “Hello from ESP8266” message, which will not be printed on the serial. After that, the ESP8266 subscribes the same topic to which the hello message was sent.

If any other producer sends a message to the “esp/test” topic, then it will be printed in the serial console, as shown in figure 2.



```
COM5
Connecting to WiFi..
Connecting to WiFi..
Connecting to WiFi..
Connecting to WiFi..
Connected to the WiFi network
Connecting to MQTT...
connected
Message arrived in topic: esp/test
Message:Hello from MQTtlens
-----
Message arrived in topic: esp/test
Message:Hello again from MQTtlens
-----
```

The screenshot shows a serial console window with a title bar that includes a refresh icon, the text "COM5", and standard window controls. A "Send" button is located at the top right. The main area displays a series of log messages. The first four lines are "Connecting to WiFi..", followed by "Connected to the WiFi network". Then, "Connecting to MQTT..." is followed by "connected" on the next line. Two messages are then received on the "esp/test" topic: "Message arrived in topic: esp/test" followed by "Message:Hello from MQTtlens", and then "Message arrived in topic: esp/test" followed by "Message:Hello again from MQTtlens". Each message pair is separated by a line of dashes. At the bottom, there is a scrollbar, an "Autoscroll" checkbox which is checked, and two dropdown menus set to "No line ending" and "115200 baud".

**Figure 2** – Messages sent to the “esp/test” topic.

For this tutorial, I used [MQTtlens](#), a Google Chrome application, which connects to a MQTT broker and is able to subscribe and publish to MQTT topics [3]. This is a very useful application that I really recommend for this type of tests.

For the test, MQTtlens was subscribing the “esp/test” topic before connecting the ESP8266. As seen in figure 3, the “Hello from ESP8266” message was printed. After that, two hello messages were sent by MQTtlens, which can be seen in the same figure. The messages sent are the ones shown in figure 2, which were received by the ESP8266, as expected.

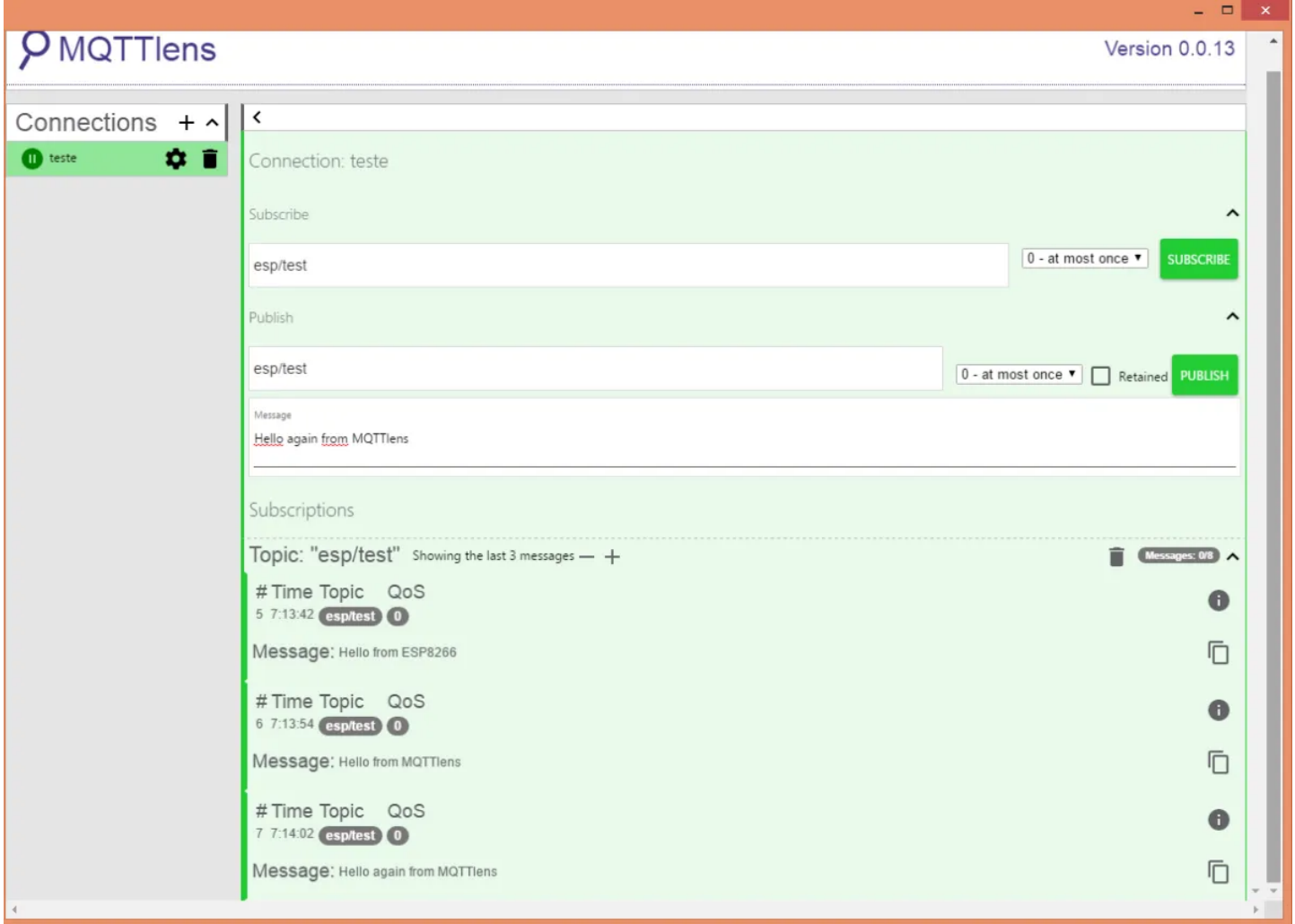


Figure 3 – Testing the program with MQTTlens.

## Related content

- [MQTTlens application](#)
- [Cloud MQTT](#)
- [MQTT.org](#)
- [List of MQTT brokers](#)
- [ESP8266 MQTT library](#)
- [Original MQTT library example](#)

## References

[1] <https://www.arduino.cc/en/Reference/WiFiClient>

[2] <http://pubsubclient.knolleary.net/api.html>

[3] <https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm>



# Technical details

- [ESP8266 libraries](#): v2.3.0
- [PubSubClient library](#): v2.6.0

Share this:

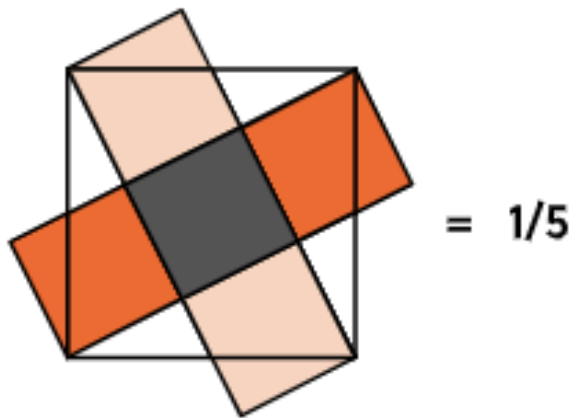


Like this:



7 bloggers like this.

What fraction of the area is grey?



[← Previous Post](#)

[Next Post →](#)

117 thoughts on “ESP8266: Connecting to MQTT broker”

[Older Comments](#)



DOUG

AUGUST 5, 2020 AT 5:49 PM

Does this sketch get uploaded to the Arduino (Uno in my case) or onto the 8266 itself? I'm just starting out with Arduino and programming with other boards like ESP8266

Loading...

[Reply](#)

## New TicWatch Pro 3 Smartwatch

Mobvoi

Start your smart life with TicWatch Pro 3.The latest platform upgarde your experience.

[Older Comments](#)

### Leave a Reply

Enter your comment here...

What fraction



Search ...



Sort by

Relevance



## Categories

[C#](#) (9)

[Electronics](#) (14)

[ESP32](#) (326)

[ESP8266](#) (99)

[IoT](#) (3)

[Javascript](#) (18)

[LinkIt Smart](#) (14)

[Micro:bit](#) (30)

[Microcontrollers](#) (10)

[Misc](#) (16)

[OBLOQ](#) (15)

[Python](#) (57)

[Raspberry Pi](#) (15)

[Sipeed M1](#) (4)

[SQL](#) (5)

Follow Blog via Email

Enter your email address to follow this blog and receive notifications of new posts by email.

Email Address

Follow

SAMSUNG

Tap in to  
tune out

with Active  
Noise Cancellation

BUY NOW

Galaxy Bu

Features including ANC  
connection to a compa

-24%

-53%

-28%

-33%