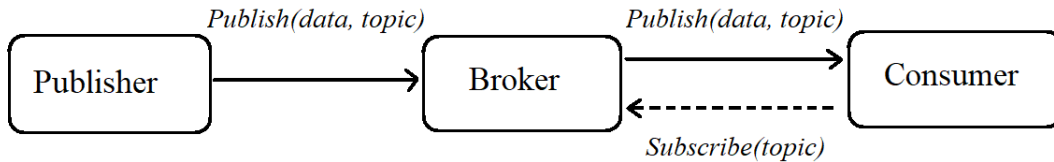


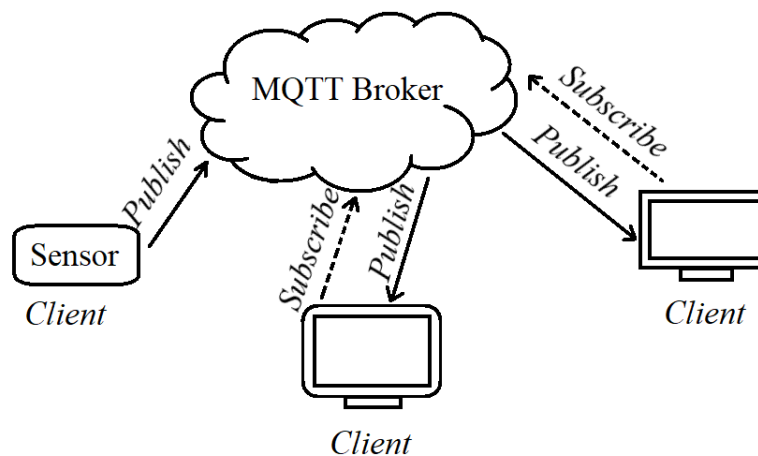
# Getting started with MQTT.

MQTT (Message Queue Telemetry Transport) is a 'machine-to-machine' protocol. It is a publish-subscribe network protocol that enables machines to communicate among themselves. There is a publisher which publishes data to the consumer via a broker.



The MQTT protocol defines a **message broker** and a **number of clients**.

- An **MQTT broker** is a server that receives messages from the clients and then directs the messages to required destination clients.
- An **MQTT client** is any device like a microcontroller or any server that runs an MQTT library. It connects to an MQTT broker over a network.



**CloudMQTT** are managed **Mosquitto servers** in the cloud and it implements MQTT protocol.

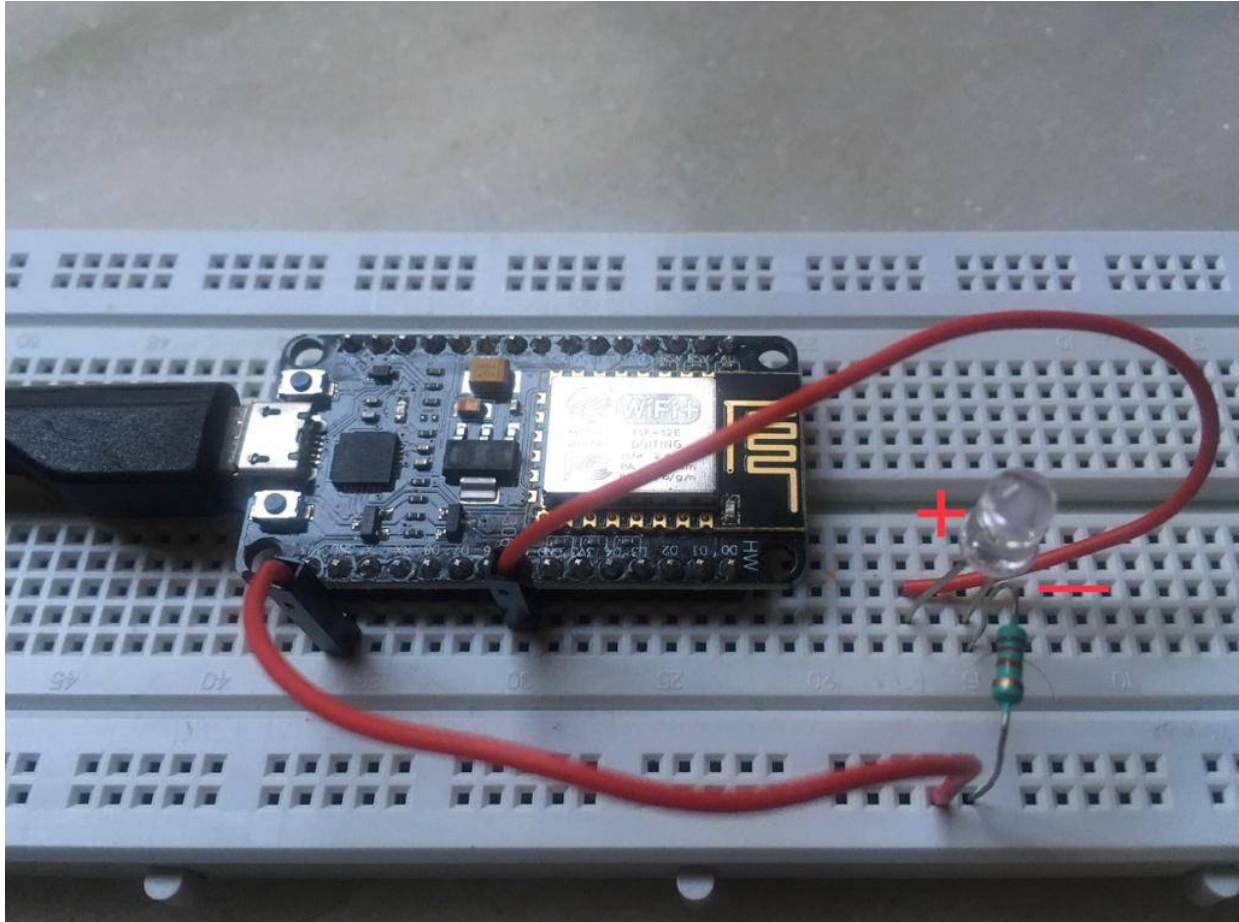
MQTT and Mosquitto are exceptionally useful for bandwidth sensitive applications.

**THINGS REQUIRED FOR LED CONTROL:**

1. CloudMQTT account.
2. ESP8266.
3. LED.
4. Resistor(220 ohm).
5. Breadboard and jumper wires.

#### **CIRCUIT CONNECTION:**

Connect pin D5 of ESP8266 to LED positive. Negative lead of LED goes to GND pin of ESP8266 via resistor as shown below.



#### **CREATE CLOUDMQTT INSTANCE:**

1. Head to <https://www.cloudmqtt.com/> and create an account and click on +Create New Instance.
2. Choose 'cute cat' plan and give it a 'name' as following.

CloudMQTT List all instances ▾ @gmail.com ▾

## Create new instance

No credit card Please [add a credit card](#) if you want to subscribe to a paid plan

Missing billing information Please [fill in all required information](#) if you want to subscribe to a paid plan

Plan Region Configure (Dedicated plans only) Confirm

### Select a plan and name - Step 1 of 4

Name


Plan Cute Cat (Free) ▾

Tags

Tags are used to separate your instances between projects. This is primarily used in the project listing view for easier navigation and access control.

Tags allow admins to manage team members access to different groups of instances.

Plan



Cute Cat

See the plan page to learn about the different

3. Then choose a data center.

CloudMQTT List all instances ▾ @gmail.com ▾

## Create new instance


No credit card Please [add a credit card](#) if you want to subscribe to a paid plan

Missing billing information Please [fill in all required information](#) if you want to subscribe to a paid plan

Plan Region Configure (Dedicated plans only) Confirm

### Select a region and data center - Step 2 of 4

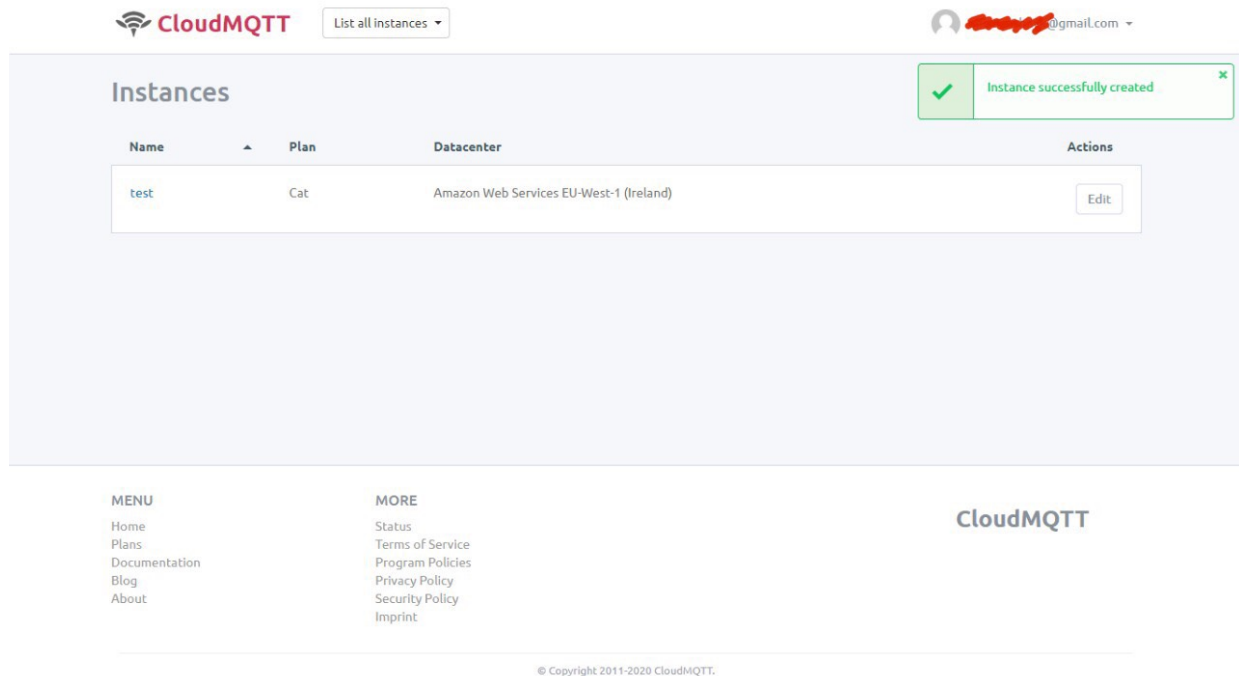
Data center US-East-1 (Northern Virginia) ▾



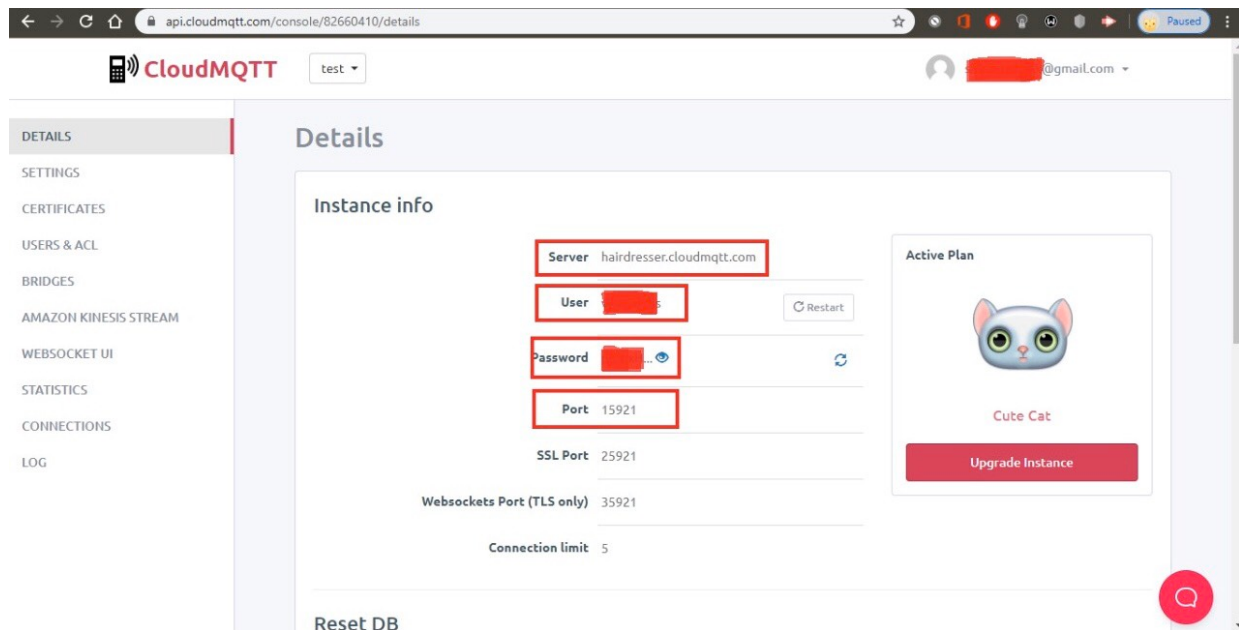
« Back

Cancel Review

4. Complete the next two steps and you are done.



5. Finally note down the following credentials(**Server, User, Password, Port**).



**Note:** WebSocket UI .

The WebSocket UI is where one can view data transfers in action.

## SETTING UP CLIENT-1 USING PYTHON:

1. First , install Paho Python Library as follows:

```
pip3 install paho-mqtt
```

It provides a client class which allows applications to connect to an MQTT broker.

2. You can write this code in any python editor of your choice. Then import the MQTT client class as follows:

```
import paho.mqtt.client as mqtt
```

If running this command alone gives you an error, it means that your paho-MQTT library is not installed correctly.

3. To connect with the server `connect()` method is used but as this MQTT server is password protected the username and password are also required.

4. 

```
client.connect("type_your_server_address", type_your_port_number, 60)
client.username_pw_set("type_your_username", "type_your_password")
```

4. The callback functions `on_connect()` and `on_message()` will be called when the connection with the server is established. These will be called automatically by the client upon establishing a connection with the broker and on receiving a message respectively. The `subscribe()` method accepts the topic name as its argument.

5. 

```
def on_connect( client, userdata, flags, rc):
    print ("Connected with Code :" + str(rc))
    client.subscribe("esp/led_control/#")
def on_message( client, userdata, msg):
    print ( str(msg.payload) )
```

6. The `publish()` method publishes a message from our client to the MQTT server.

This sends messages to the broker and also from the broker to any clients subscribing to matching topics. It takes the following arguments:

- **topic:** the topic on which the message should be published.
- **payload:** the actual message to be sent. If no message is specified or set to 'None' then a zero-length message will be used.
- **QoS:** the quality of service level to be used.
- **retain:** Setting this to True, the last known or retained messages will be set for the topic.

This 'while loop' turns ON LED and displays it, waits for few seconds as it takes time for the message to reach the client. After this, the LED is turned OFF.

- ```
while True:
    client.publish("esp/led_control", "1")
    print ("LED ON")
    time.sleep(6)
    client.publish("esp/led_control", "0")
    print ("LED OFF")
    time.sleep(4)
```

6. To publish and subscribe in the same script, `loop_start()` and `loop_stop()` methods are used. Calling **loop\_start()** runs a thread in the background to call **loop()** automatically. This frees up the main thread and also handles reconnecting to the broker. Calling **loop\_stop()** stops the background thread.

**CODE TO PUBLISH AND SUBSCRIBE TOPICS:**

```

import time
import paho.mqtt.client as mqtt

# Callback Function on Connection with MQTT Server
def on_connect( client, userdata, flags, rc):
    print ("Connected with Code :" +str(rc))
    client.subscribe("esp/led_control/#") # Subscribe Topic from here

# Callback Function on Receiving the Subscribed Topic/Message
def on_message( client, userdata, msg):
    # print the message received from the subscribed topic
    print ( str(msg.payload) )

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("type_your_server_address", type_your_port_number, 60)
client.username_pw_set("type_your_username", "type_your_password")

client.loop_start()
time.sleep(1)
while True:
    client.publish("esp/control_led", "1")
    print ("LED ON")
    time.sleep(15)
    client.publish("esp?control_led", "0")
    print ("LED OFF")
    time.sleep(4)
client.loop_stop()
client.disconnect()

```

## Setting up client-2(ESP8266):

1. First, include ESP8266WiFi library(to connect to a WiFi network) and PubSubClient library(to connect to an MQTT broker and to publish or subscribe messages of a particular topic.)

```

#include<ESP8266WiFi.h>
#include<PubSubClient.h>

```

2. Next include few global variables. SSID, password of your WiFi; MQTT server address, port number, user and password from your CloudMQTT console. Also type a random 'device\_id'. Simply replace your values below:

```

//to connect to WiFi
const char *ssid = "type_your_ssid";
const char *password = "type_your_password";
//to connect to CloudMQTT
const char *mqtt_server = "type_your_mqtt_server_address";
const int mqttPort = type_your_port_number;
const char *mqtt_user = "type_your_user";
const char *mqtt_pass = "type_your_password";
const char *device_id = "esp8266";

```

3. Now declare two objects. One, of class `WiFiClient` to connect to a specific IP and port, and the other of class `PubSubClient` which receives as input, the constructor of previously defined `WiFiClient`.

```
WiFiClient espClient;  
PubSubClient client(espClient);
```

4. In the `setup()` function, open the serial monitor. The method `setServer()` accepts MQTT server address and port number as its arguments. The method `setCallback()` accepts as argument a function which is defined later in this segment. Finally set D5 as output pin to control LED.

```
void setup(){  
  Serial.begin(115200);  
  client.setServer(mqtt_server, mqttPort);  
  client.setCallback(callback);  
  pinMode(D5, output);  
}
```

5. The `reconnect()` function is called repeatedly till the client establishes a successful connection. The while loop below runs until the client(esp8266) connects to the MQTT broker. `connect()` method accepts as arguments: **clientId**(chosen randomly), **MQTT user and password**. It returns a boolean value i.e. true if a connection is established and false otherwise.

Upon a successful connection, we can publish and subscribe topics using `publish()` and `subscribe()` methods respectively. If the connection fails, then the `state()` method helps to identify the error. For eg. if it returns -2, it means a network error occurred. You can [click here](#) to infer other error messages and know more about `PubSubClient`.

- 1.
- ```
while (!client.connected()){  
  Serial.print("Attempting MQTT connection...");  
  if (client.connect(device_id, mqtt_user, mqtt_pass )){  
    Serial.println("connected");  
    client.subscribe("esp/led_control"); // topic-name  
  }  
  else{  
    Serial.print("failed, rc=");  
    Serial.print(client.state());  
    Serial.println(" try again in 5 seconds");  
    delay(5000);  
  }  
}
```

6. Callback function handles the incoming messages from the topic subscribed. Topic name is stored in a character array and then converted to a string data type.

- 7.
- ```
void callback(char *topic, byte *payload, unsigned int length){  
  Serial.print("Message arrived [");  
  Serial.print(topic);  
  Serial.println("] ");  
  int i;  
  for (i = 0; i < length; i++){
```

```

        message_buff[i] = payload[i];
    }
    message_buff[i] = '\0';
    String msgString = String(message_buff);
    Serial.println(msgString);

```

7. We now compare the received topic name(sent by the broker) to “esp/led\_control”. If there’s a match, we enter the loop and turn the LED ON or OFF accordingly.

```

if (strcmp(topic, "esp/led_control") == 0){
    if (msgString == "1"){
        digitalWrite(D5, HIGH ); // turns LED ON
    }
    if (msgString == "0"){
        digitalWrite(D5, LOW); // turns LED OFF
    }
}
}

```

1. To make this happen repeatedly, the `loop()` function is used.

2.

```

void loop(){
    if (!client.connected()){
        reconnect();
    }
    client.loop();
}

```

## COMPLETE ARDUINO CODE:

```

#include <ESP8266WiFi.h>
#include <PubSubClient.h>

//to connect to WiFi
const char *ssid = "type_your_ssid";
const char *password = "type_your_password";
//to connect to CloudMQTT
const char *mqtt_server = "type_your_mqtt_server_address";
const int mqttPort = type_your_port_number;
const char *mqtt_user = "type_your_user";
const char *mqtt_pass = "type_your_password";
const char *device_id = "esp8266";

WiFiClient espClient;
PubSubClient client(espClient);

char message_buff[100];
void callback(char *topic, byte *payload, unsigned int length){
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.println("] ");
    int i;
    for (i = 0; i < length; i++){

```



```

        message_buff[i] = payload[i];
    }
    message_buff[i] = '\0';
    String msgString = String(message_buff);
    Serial.println(msgString);
    if (strcmp(topic, "esp/led_control") == 0){
        if (msgString == "1"){
            digitalWrite(D5, HIGH ); // turns LED ON
        }
        if (msgString == "0"){
            digitalWrite(D5, LOW); // turns LED OFF
        }
    }
}

void reconnect(){
    while (!client.connected()){
        Serial.print("Attempting MQTT connection...");
        if (client.connect(device_id, mqtt_user, mqtt_pass )){
            Serial.println("connected");
            client.subscribe("esp/led_control"); // topic-name
        }
        else{
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            delay(5000);
        }
    }
}

void setup(){
    Serial.begin(115200);
    client.setServer(mqtt_server, mqttPort);
    client.setCallback(callback);
    pinMode(D5, output); // setting D5 as output to control LED
}

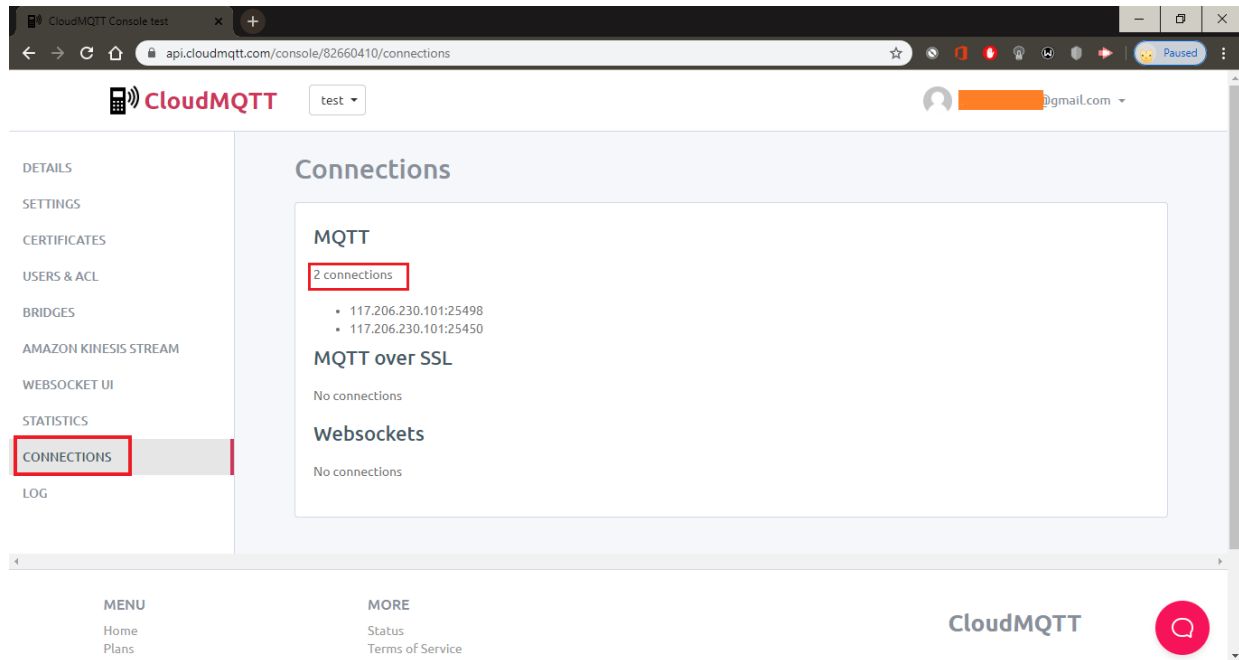
void loop()
{
    if (!client.connected()){
        reconnect();
    }
    client.loop();
}

```

#### EXECUTION STEPS:

- Upload the above code on ESP8266.
- Log onto cloudMQTT platform and open console.
- Open 'serial monitor' in Arduino IDE and press 'Reset' button on ESP8266.
- Run python code.

You can check if both the devices have connected to broker or not by clicking on 'connections' option. There are two connections visible: One is Paho client and the other is ESP8266 client.



#### WORKING:

We have two clients connected to our broker(CloudMQTT) in this example:

1. Paho Python client which can both publish and subscribe the topics and
2. ESP8266 PubSubClient(only subscribes topics)

Paho client publishes the LED states to the broker which are then subscribed by both the clients. ESP8266 client controls LED and Paho client simply displays the message.

The received message i.e. '1' or '0' can be seen on both the serial monitor and command prompt as both the clients subscribe to the topic.

**Finally here is LED control in live action:**

COM5

Send

Message arrived [esp/led\_control]1  
1 — received message  
Message arrived [esp/led\_control]0  
0  
Message arrived [esp/led\_control]1  
1  
Message arrived [esp/led\_control]0  
0  
Message arrived [esp/led\_control]1  
1  
Message arrived [esp/led\_control]0  
0

Autoscroll

Show timestamp

Both NL & CR

115200 baud

Clear output

Command Prompt - python esp\_cldmqtttest.py

KeyboardInterrupt  
C:\Users\SHEIKH\python>python esp\_cldmqtttest.py  
Connected with Code :5  
ON  
Connected with Code :0  
OFF  
b'0' — received message  
ON  
b'1'  
OFF  
b'0'  
ON  
b'1' — printed after publishing  
OFF  
b'0'  
ON  
b'1'  
OFF  
b'0'

CloudMQTT Console test

api.cloudmqtt.com/console/8...

Paused

Client ID

Clear

Received messages

| Topic           | Message |
|-----------------|---------|
| esp/led_control | 0       |
| esp/led_control | 1       |
| esp/led_control | 0       |
| esp/led_control | 1       |
| esp/led_control | 0       |
| esp/led_control | 1       |
| esp/led_control | 0       |

