Connecting the ESP8266 to a cloud server

In this recipe, we will connect the ESP8266 to the Internet and send data to a cloud server. The cloud server we will be sending data to is `dweet.io`. The data we send to `dweet.io` will be used later in this book, so ensure that you execute this section successfully.

Getting ready

As in the previous recipe, we won't need any extra components here. All we need to do is ensure that the ESP8266 is connected to the computer.

How to do it…

To accomplish this, follow these steps:

1. We will connect the ESP8266 to a local Wi-Fi network that has an active Internet connection.
2. Once the connection is successful, we will send a `GET` request to the cloud server and then display the reply that the server sends back to the ESP8266 board:

```
3.  // Libraries
    #include <ESP8266WiFi.h>
```
Copy

4. Enter the `SSID` and `password`:

```
5.  // SSID
6.  const char* ssid     = "your-ssid";
    const char* password = "your-password";
```
Copy

7. Store the hostname of the cloud server:

```
8.  // Host
    const char* host = "dweet.io";
```
Copy

9. Configure the SSID and password of the Wi-Fi network and connect the ESP8266 to the Wi-Fi network:

```
10. void setup() {
11.
12.   // Serial
13.   Serial.begin(115200);
14.   delay(10);
15.
16.   // We start by connecting to a WiFi network
17.   Serial.println();
18.   Serial.println();
19.   Serial.print("Connecting to ");
20.   Serial.println(ssid);
21.
22.   WiFi.begin(ssid, password);
23.
24.   while (WiFi.status() != WL_CONNECTED) {
25.     delay(500);
26.     Serial.print(".");
27.   }
28.
29.   Serial.println("");
30.   Serial.println("WiFi connected");
31.   Serial.println("IP address: ");
32.   Serial.println(WiFi.localIP());
} Copy
```

33. Delay for five seconds and then print the name of the host we are connecting to on the serial monitor:

```
34. void loop() {
35.
36.   delay(5000);
37.
38.   Serial.print("connecting to ");
      Serial.println(host); Copy
```

39. Connect to the host server:

```
40.   // Use WiFiClient class to create TCP connections
41.   WiFiClient client;
42.   const int httpPort = 80;
43.   if (!client.connect(host, httpPort)) {
44.     Serial.println("connection failed");
45.     return;
    } Copy
```

46. Formulate the URI for the `GET` request we will send to the host server:

```
47.   // We now create a URI for the request
      String url = "/dweet/for/my-thing-name?value=test"; Copy
```

48. Send the `GET` request to the server and check whether the request has been received or if it has timed out:

```
49.   // Send request
50.   Serial.print("Requesting URL: ");
51.   Serial.println(url);
52.
```

```
  client.print(String("GET ") + url + " HTTP/1.1\r\n" +
               "Host: " + host + "\r\n" +
               "Connection: close\r\n\r\n");
  unsigned long timeout = millis();
  while (client.available() == 0) {
    if (millis() - timeout > 5000) {
      Serial.println(">>> Client Timeout !");
      client.stop();
      return;
    }
  }
```
Copy

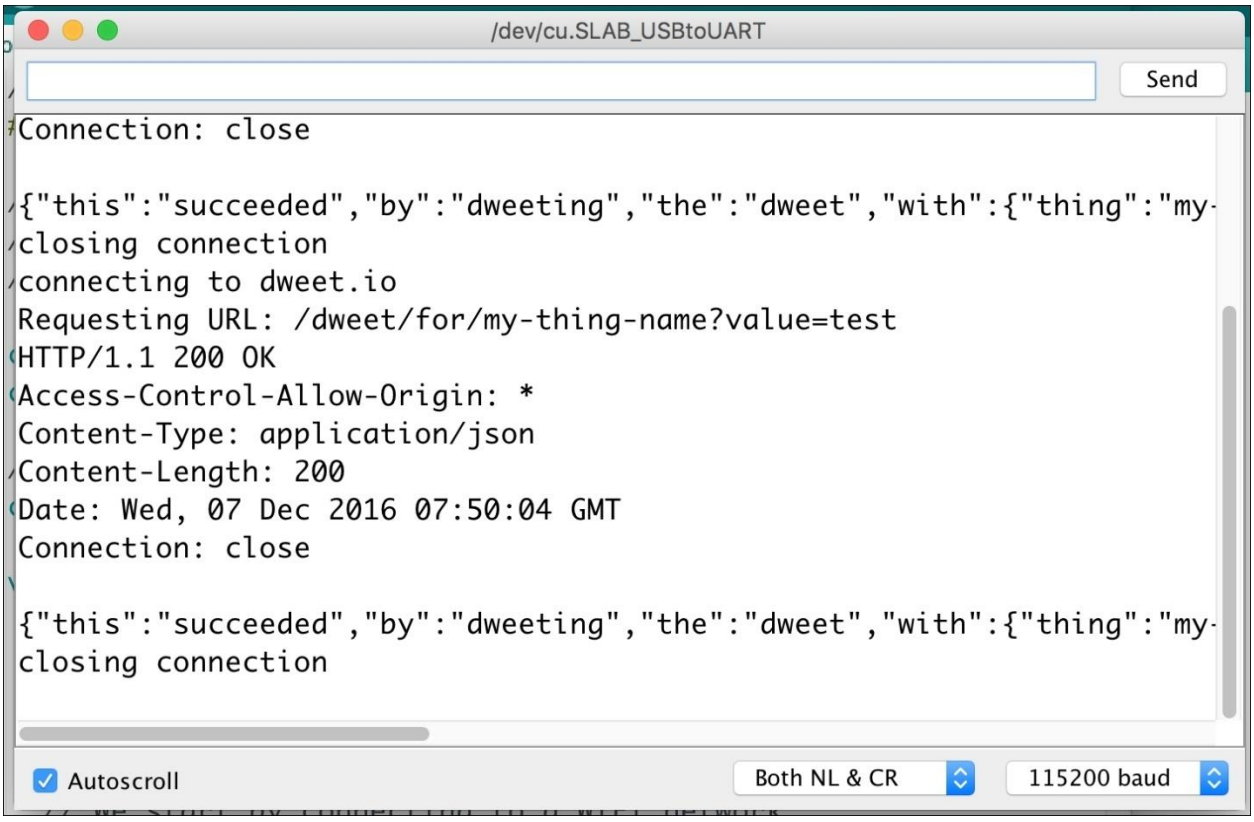
63. Read incoming data from the host server line by line and display the data on the serial monitor.
64. Close the connection after all the data has been received from the server:

```
  // Read all the lines from the answer
  while(client.available()){
    String line = client.readStringUntil('\r');
    Serial.print(line);
  }

  // Close connecting
  Serial.println();
  Serial.println("closing connection");
}
```
Copy

74. Copy the sketch to your Arduino IDE and change the SSID in the code from `your-ssid` to the name of your Wi-Fi network and the password from `your-password` to the password of your Wi-Fi network.
75. Upload the sketch to your ESP8266 board.
76. Open the serial monitor so that you can view the incoming data.

The serial monitor should display data, as shown in the following screenshot:



As you can see from the serial monitor, when you send the `GET` request to `dweet.io` you receive a reply from the server. The reply is enclosed in curly brackets `{}`.

How it works…

The program connects to the Wi-Fi network using the provided password and SSID. It then proceeds to connect to the provided cloud/host server using the `client.connect()` function, and sends the provided URI to the host server using the `client.print()` function.

Once the data has been successfully sent, the sketch waits for a reply from the server. It does this with the `client.available()` function, which checks whether there is incoming data from the server. If there is data available, the sketch reads it and displays it on the serial monitor. The process is repeated until the ESP8266 is turned off.

There's more…

Since you have understood how to connect the ESP8266 to a cloud server, see if you can change the sketch so that it connects to the www.google.com host server and searches for the word Arduino. The results from the server should be displayed on the serial monitor.

HTTP Server on NodeMCU with Arduino IDE

Introduction

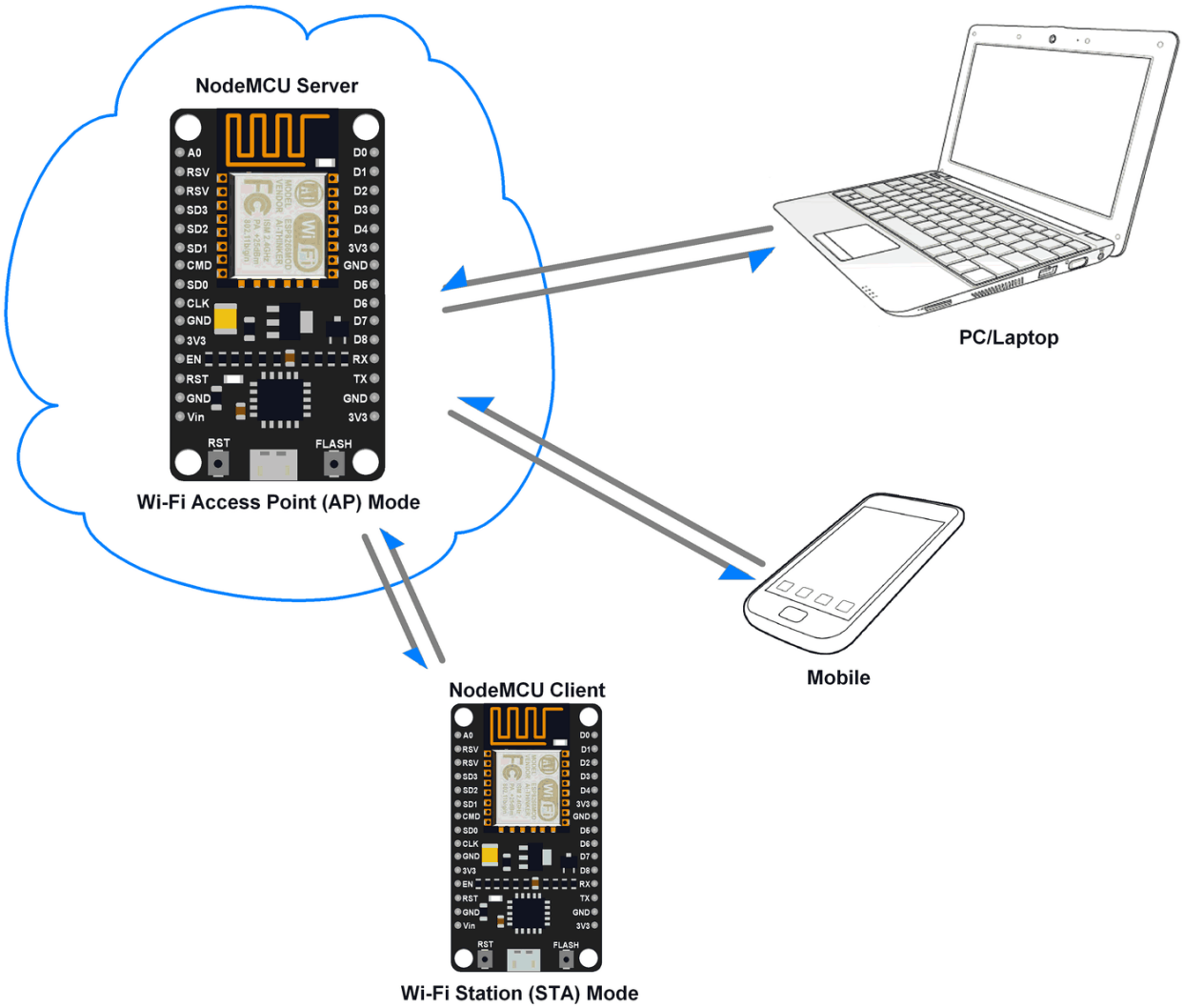
The Hypertext Transfer Protocol (HTTP) is a standard application layer protocol that functions as a request-response protocol between server and client.

It is widely used in IoT (Internet of Things) embedded applications, where every sensor is connected to a server and we have access to control them over the internet.

NodeMCU has Wi-Fi functionality available on board. With this Wi-Fi functionality, NodeMCU can connect to any wi-fi network as a client or it can create a network to which other wi-fi enabled devices can connect.

NodeMCU as HTTP Server using Wi-Fi AP mode

NodeMCU wi-fi has Access Point (AP) mode through which it can create Wireless LAN to which any wi-fi enabled device can connect as shown in the below figure.

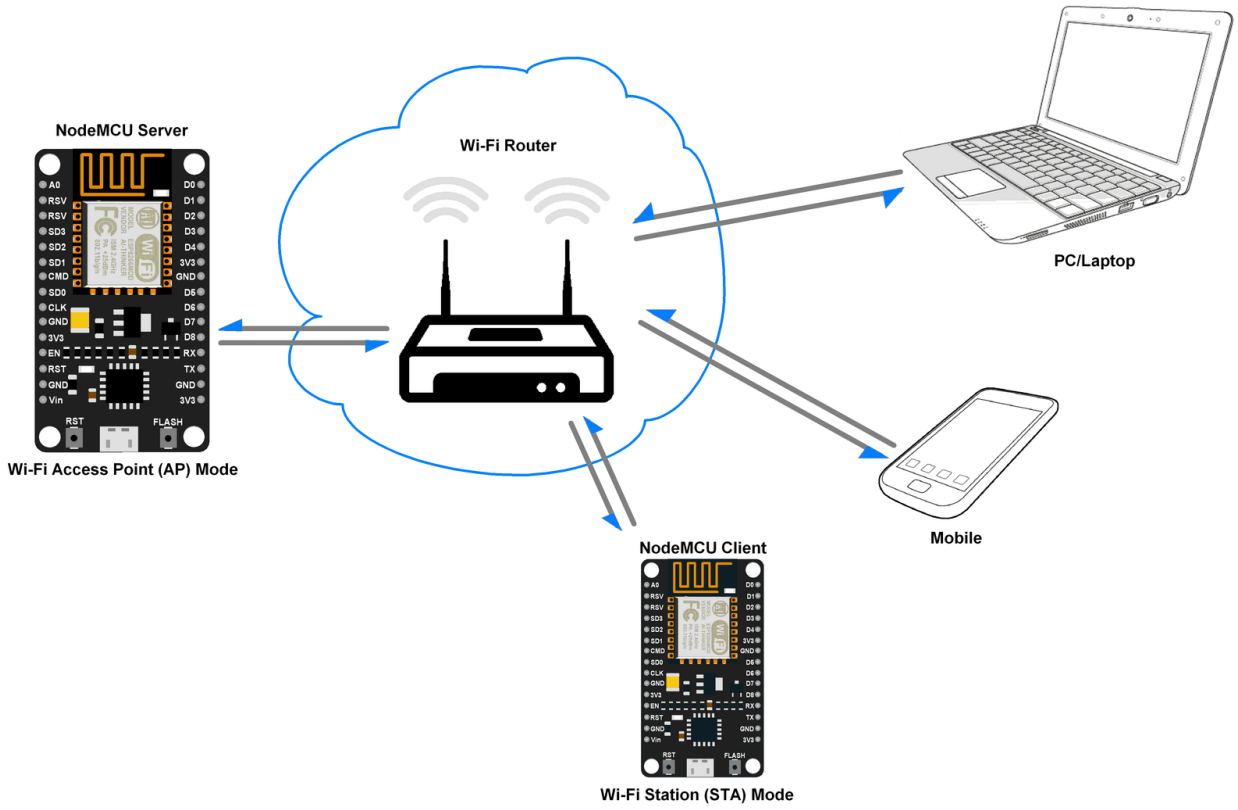


NodeMCU as HTTP Server using Wi-Fi AP mode

We can set SSID and Password for AP mode which will be used to authenticate other devices while connecting to it.

NodeMCU as HTTP Server using Wi-Fi STA mode

NodeMCU has Station (STA) mode using which it can connect to the existing wi-fi network and can act as an HTTP server with an IP address assigned by that network.



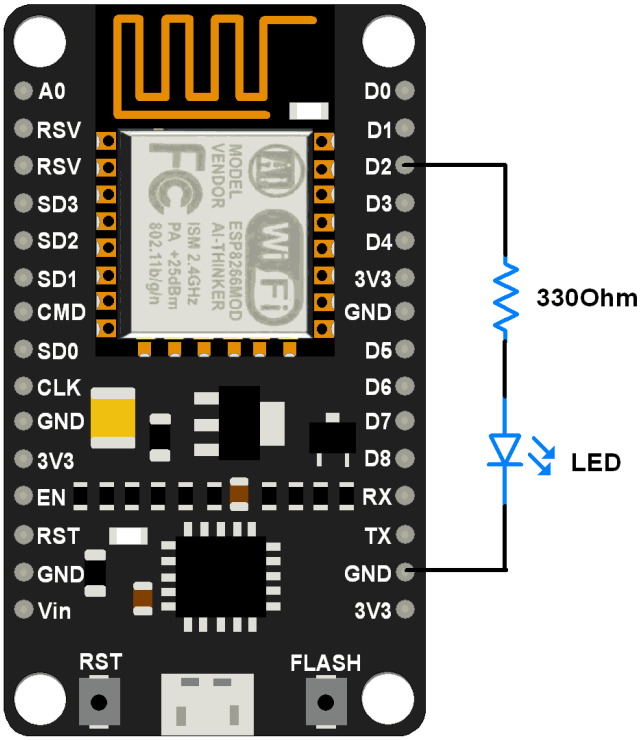
NodeMCU as HTTP Server using Wi-Fi STA mode

NodeMCU gets IP from the Wi-Fi router to which it is connected. With this IP address, it can act as an HTTP server to which any wi-fi device can connect.

Example

Let's write Arduino Sketch to enable NodeMCU as an HTTP server with Wi-Fi STA/AP mode and control an LED connected at the server-side from the client-side.

Here we have connected LED to the pin no. 2 i.e. D2 pin on the NodeMCU board as shown in the below figure.



HTML page for client

As we are making an HTTP server for LED on/off functionality, we are going to make a simple HTML page that will be visible at the client-side and able to take user input for LED on/off. It is a user-friendly representation of button input that takes input from the user on click.

We need to write two HTML pages for LED ON and LED OFF state i.e. when a client clicks the LED ON button, then in the next action, we need to provide options for LED OFF. Below are the two HTML code snippets for LED ON and LED OFF state presentation.

HTML Code Snippet for LED ON

```
<!DOCTYPE html>
<html>
<head><title>LED Control</title></head>
<body>
<h1>LED</h1>
<p>Click to switch LED on and off.</p>
```

```
<form method="get">
<input type="button" value="LED ON" onclick="window.location.href='/ledon'">
</form>
</body>
</html>
```

HTML Code Snippet for LED OFF

```
<!DOCTYPE html>
<html>
<head><title>LED Control</title></head>
<body>
<h1>LED</h1>
<p>Click to switch LED on and off.</p>
<form method="get">
<input type="button" value="LED OFF" onclick="window.location.href='/ledoff'">
</form>
</body>
</html>
```

From the above two HTML snippets, we can see that only forms are different for LED ON and LED OFF state.

Let's have a look at HTML lines

<!DOCTYPE html>: This declaration defines that document as an HTML and helps browsers to display web pages correctly. It must only appear once, at the top of the page.

<html>: This element is the root element of an HTML page

<head>: This element contains meta-information about the document

<title>: This element specifies a title for the document

<body>: This element contains the visible page content i.e. body of document

<h1>: This element defines the largest font size for heading. Similarly, we can use <h2>/<h3>and so on for smaller font sizes of the header.

<p>: This element defines a paragraph.

<form>: This element defines a form that is used to collect user input

window.location.href: This is a property that will tell us the current URL location. Changing the value of the property will redirect the page.

e.g.`window.location.href='/ledon'` will redirect the current page to the URL `current_url/ledon` page. If the current location is `http://192.168.0.1` then it will redirect to `http://192.168.0.1/ledon` page. Page redirect action is taken on click event (e.g. click on the button).

Here we are using the above-mentioned concept (page redirect) to redirect the client from the LED ON page to the LED OFF page and vice versa.

To learn more about HTML refer https://www.w3schools.com/html/default.asp

Now we can send the above HTML snippets when the client connects to the server and also when the client clicks on the button.

Program

In Wi-Fi Access Point (AP) mode, NodeMCU creates a server hence we can set its IP address, IP subnet mask, and IP gateway.

Let's take below SSID, Password to join the network and addresses for AP mode

- SSID = "NodeMCU"
- Password = "12345678"
- IP = "192.168.2.1"
- Sub netmask = "255.255.255.0"
- Gateway = "192.168.2.1"

Arduino Sketch for HTTP server with wi-fi AP mode

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
```

```
/* Put your SSID & Password */
const  char* ssid = "NodeMCU";   // Enter SSID here
const  char* password = "12345678";   //Enter Password here

/* Put IP Address details */
IPAddress local_ip(192,168,2,1);
IPAddress gateway(192,168,2,1);
IPAddress subnet(255,255,255,0);

ESP8266WebServer server(80);

uint8_t LEDpin = D2;
bool LEDstatus = LOW;

void setup() {
  Serial.begin(9600);
  pinMode(LEDpin, OUTPUT);

  WiFi.softAP(ssid, password);
  WiFi.softAPConfig(local_ip, gateway, subnet);
  delay(100);

  server.on("/", handle_OnConnect);
  server.on("/ledon", handle_ledon);
  server.on("/ledoff", handle_ledoff);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");
}
void loop() {
```

```
  server.handleClient();
  if(LEDstatus)
  digitalWrite(LEDpin, HIGH);
  else
  digitalWrite(LEDpin, LOW);
}

void handle_OnConnect() {
  LEDstatus = LOW;
  server.send(200, "text/html", SendHTML(false));
}

void handle_ledon() {
  LEDstatus = HIGH;
  server.send(200, "text/html", SendHTML(true));
}

void handle_ledoff() {
  LEDstatus = LOW;
  server.send(200, "text/html", SendHTML(false));
}

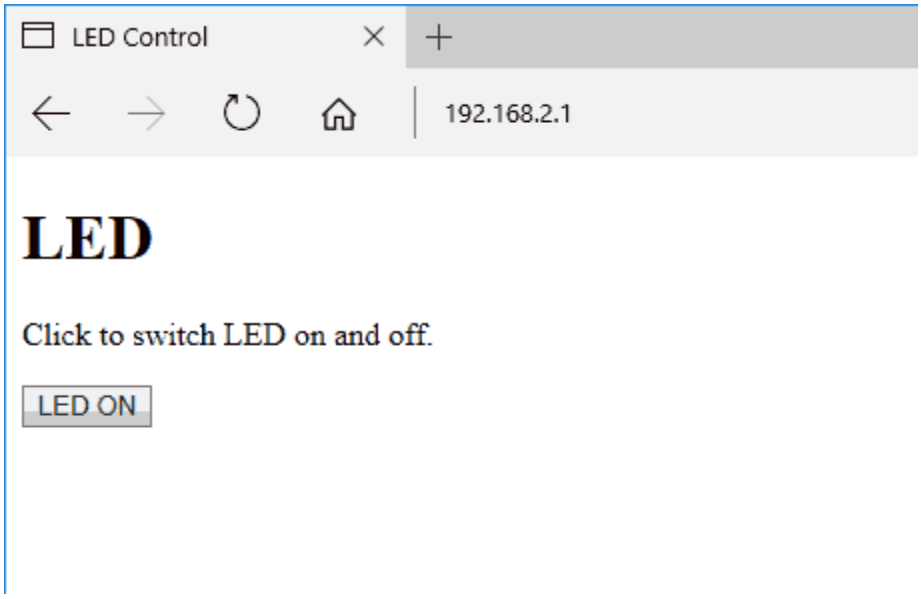
void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}

String SendHTML(uint8_t led){
  String ptr = "<!DOCTYPE html>\n";
  ptr +="<html>\n";
  ptr +="<head>\n";
  ptr +="<title>LED Control</title>\n";
  ptr +="</head>\n";
```

```
  ptr +="<body>\n";
  ptr +="<h1>LED</h1>\n";
  ptr +="<p>Click to switch LED on and off.</p>\n";
  ptr +="<form method=\"get\">\n";
  if(led)
  ptr +="<input type=\"button\" value=\"LED OFF\" onclick=\"window.location.href='/led
off'\">\n";
  else
  ptr +="<input type=\"button\" value=\"LED ON\" onclick=\"window.location.href='/ledo
n'\">\n";
  ptr +="</form>\n";
  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

Note: After successful uploading of the above sketch client needs to connect to the network created by NodeMCU first.

After connecting to NodeMCU network from wifi enter the server address in browser i.e. `http://server_ip_address` e.g. in our case, it is `http://192.168.2.1`. After press the Enter key we can see the HTML page response from the server as shown in the below image. Now just click the button to change the state of the LED.



Now, let's do the HTTP server to NodeMCU using Wi-Fi station mode.

In Wi-Fi Station (STA) mode, NodeMCU gets IP addresses from the Wi-Fi router (access point). If we are also in the same network then we can directly connect to NodeMCU HTTP Server using the IP address only.

Arduino Sketch for HTTP server with wi-fi STA mode

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

/*Put your SSID & Password*/
const char* ssid = "ssid";  // Enter SSID here
const char* password = "password";  //Enter Password here

ESP8266WebServer server(80);

uint8_t LEDpin = D2;
bool LEDstatus = LOW;

void setup() {
  Serial.begin(9600);
```

```
  delay(100);
  pinMode(LEDpin, OUTPUT);

  Serial.println("Connecting to ");
  Serial.println(ssid);

  //connect to your local wi-fi network
  WiFi.begin(ssid, password);

  //check wi-fi is connected to wi-fi network
  while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected..!");
  Serial.print("Got IP: ");  Serial.println(WiFi.localIP());

  server.on("/", handle_OnConnect);
  server.on("/ledon", handle_ledon);
  server.on("/ledoff", handle_ledoff);
  server.onNotFound(handle_NotFound);

  server.begin();
  Serial.println("HTTP server started");
}
void loop() {
  server.handleClient();
  if(LEDstatus)
  digitalWrite(LEDpin, HIGH);
  else
```

```
  digitalWrite(LEDpin, LOW);
}

void handle_OnConnect() {
  LEDstatus = LOW;
  server.send(200, "text/html", SendHTML(false));
}

void handle_ledon() {
  LEDstatus = HIGH;
  server.send(200, "text/html", SendHTML(true));
}

void handle_ledoff() {
  LEDstatus = LOW;
  server.send(200, "text/html", SendHTML(false));
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}

String SendHTML(uint8_t led){
  String ptr = "<!DOCTYPE html>\n";
  ptr +="<html>\n";
  ptr +="<head>\n";
  ptr +="<title>LED Control</title>\n";
  ptr +="</head>\n";
  ptr +="<body>\n";
  ptr +="<h1>LED</h1>\n";
  ptr +="<p>Click to switch LED on and off.</p>\n";
```

```
  ptr +="<form method=\"get\">\n";
  if(led)
  ptr +="<input type=\"button\" value=\"LED OFF\" onclick=\"window.location.href='/ledoff'\">\n";
  else
  ptr +="<input type=\"button\" value=\"LED ON\" onclick=\"window.location.href='/ledon'\">\n";
  ptr +="</form>\n";
  ptr +="</body>\n";
  ptr +="</html>\n";
  return ptr;
}
```

Note: in wi-fi station mode we need to enter the SSID and password of the existing network. After connecting to the WiFi network enter the server address in the browser i.e. `http://assigned_ip_address`. After pressing the Enter key we can see the HTML page response from the server in the browser as shown above for AP mode

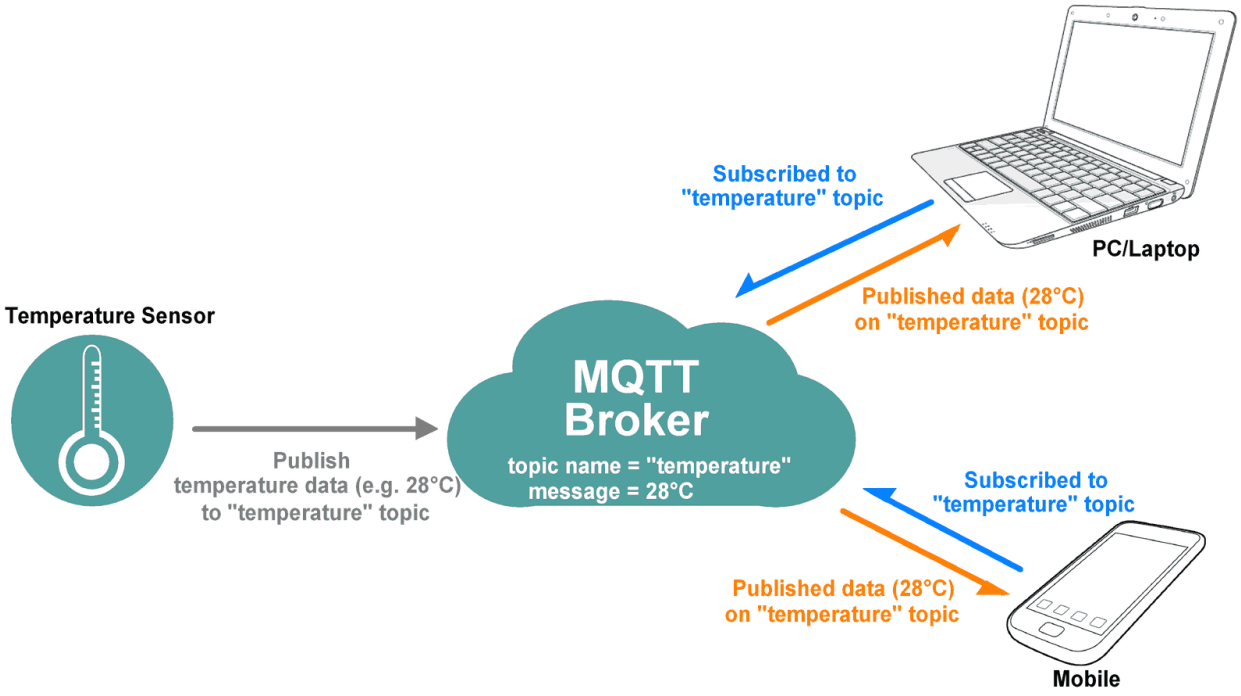
NodeMCU MQTT Client with Arduino IDE

Introduction

MQTT is a lightweight publish-subscribe-based messaging protocol.

- It is quicker (faster) than other request-response based APIs like HTTP.
- It is developed on the base of the TCP/IP protocol.
- It allows remote location devices to connect, subscribe, publish, etc. to a specific topic on the server with the help of a message broker.
- MQTT Broker/Message broker is a module in between the sender and the receiver. It is an element for message validation, transformation, and routing.

- The broker is responsible for distributing messages to the interested clients (subscribed clients) of their interested topic.



For example, if the temperature sensor publishes the temperature data (message) on the topic “temperature” then interested clients who have subscribed to the “temperature” topic get that published temperature data as shown in the above figure.

MQTT is widely used in IoT (Internet of Things) embedded applications, where every sensor is connected to a server and we have access to control them over the internet.

NodeMCU is an open-source IoT platform. It is a firmware which runs on ESP8266 Wi-Fi SoC from Espressif Systems. It has onboard wi-fi available through which IoT applications become easy to build.

The MQTT Client module of NodeMCU is according to version 3.1.1 of the MQTT protocol. Make sure that your broker supports and is correctly configured for version 3.1.1. let’s see the functions used for MQTT on NodeMCU.

MQTT Packet Formation

MQTT uses many packet formats that used to connect to the server and subscribe or publish to the topic on the server.

Refer below link for MQTT OASIS standard. It will help to understand MQTT packet formations.

http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718027

Example

Let's write an Arduino program to configure NodeMCU as MQTT Client to

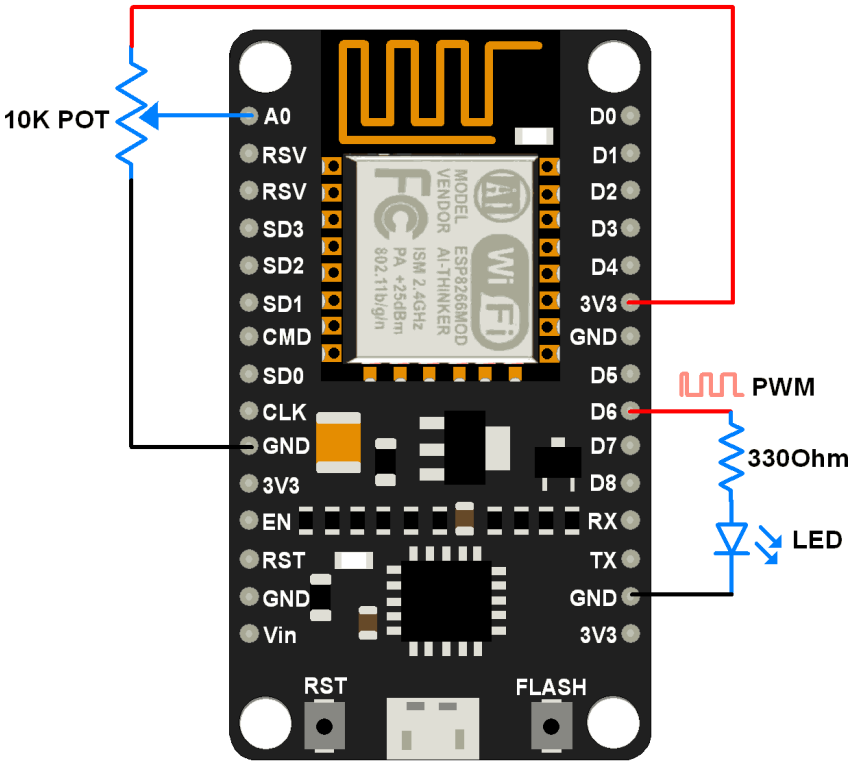
1. Control LED brightness from remote location and
2. Sending voltage across POT(Potentiometer) in digital form to remote location from the Adafruit dashboard.

Here we are using the Adafruit server for MQTT Client demo purpose.

In the IOT platform, Adafruit IO Dashboard allows us to visualize and provides control over the connected devices to the internet. Anyone can visualize and analyze live data from their sensor devices. To learn more and start with Adafruit IO Dashboard refer link https://learn.adafruit.com/adafruit-io-basics-dashboards/creating-a-dashboard

Just sign up and create a dashboard. After the successful creating of the dashboard, we will get the AIO key which is later used to access feed data.

Once we created a dashboard on Adafruit we can add various blocks that can be used to control devices as well as monitor the status of devices. To see more about blocks, refer link https://learn.adafruit.com/adafruit-io-basics-dashboards/adding-blocks



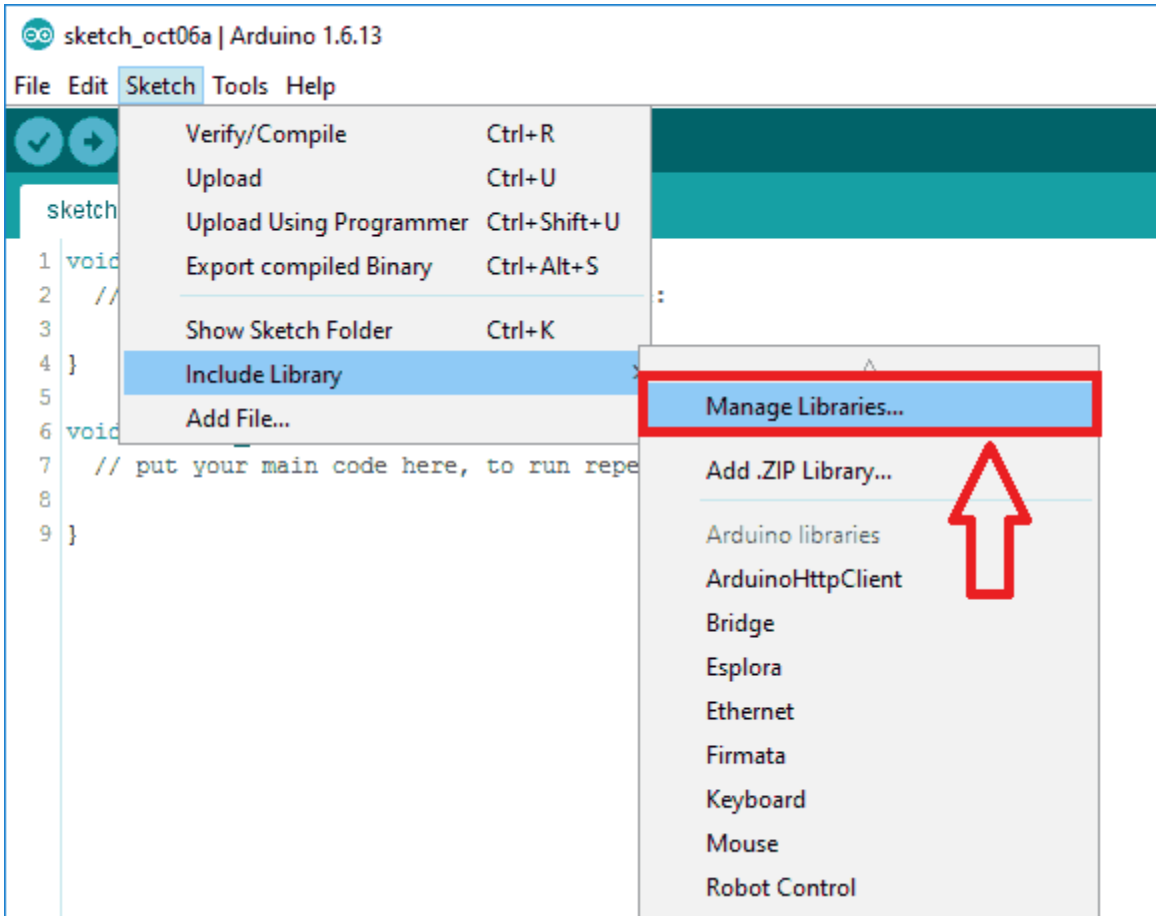
POT and LED connection to NodeMCU

Install required libraries

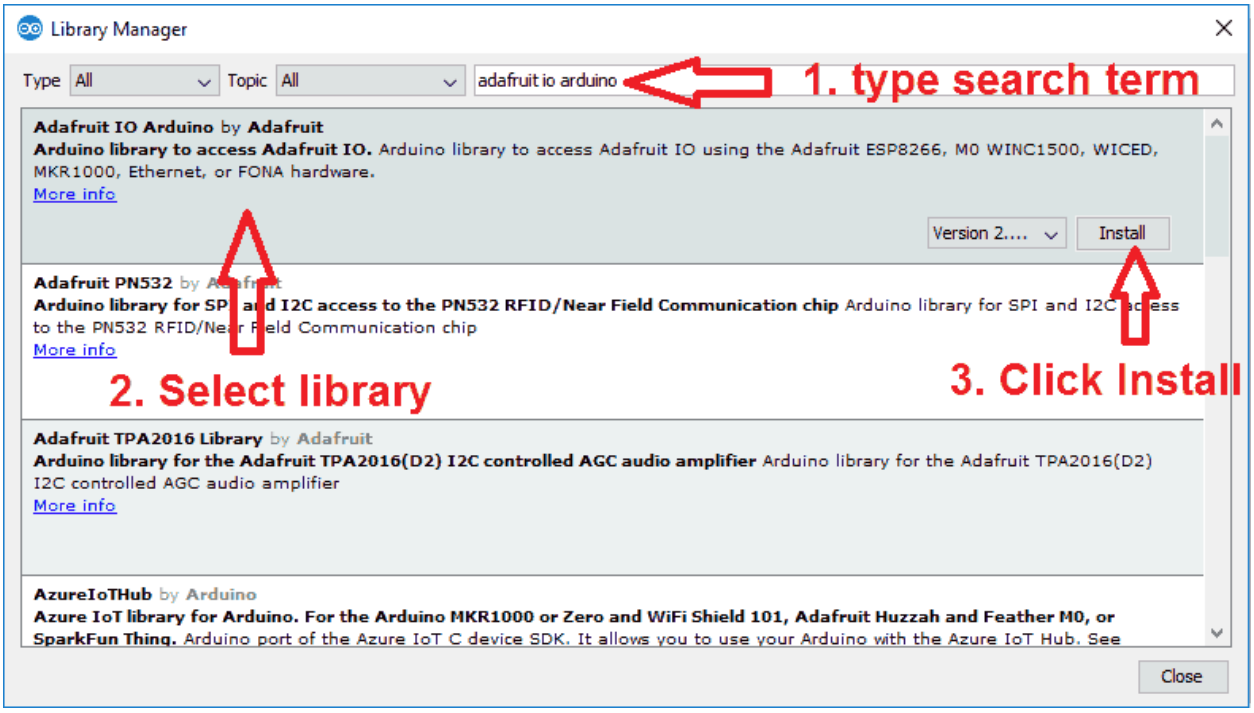
First, refer to Getting Started with NodeMCU using Arduino IDE if you are not installed NodeMCU board packages in Arduino IDE.

Here we are using Adafruit libraries for the above example. We will need to install the **Adafruit IO**, **Adafruit MQTT**, and **ArduinoHttpClient** libraries using the Arduino Library Manager.

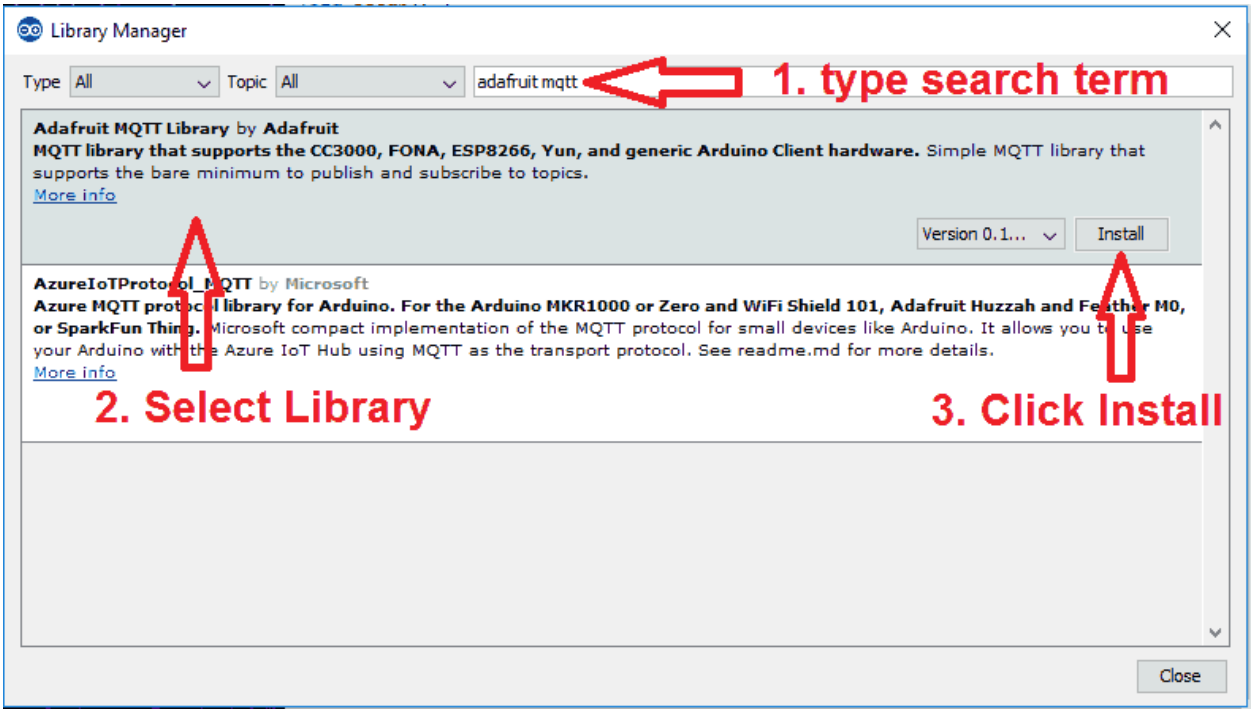
Open the Arduino IDE and navigate to **Sketch -> Include Library -> Manage Libraries…**



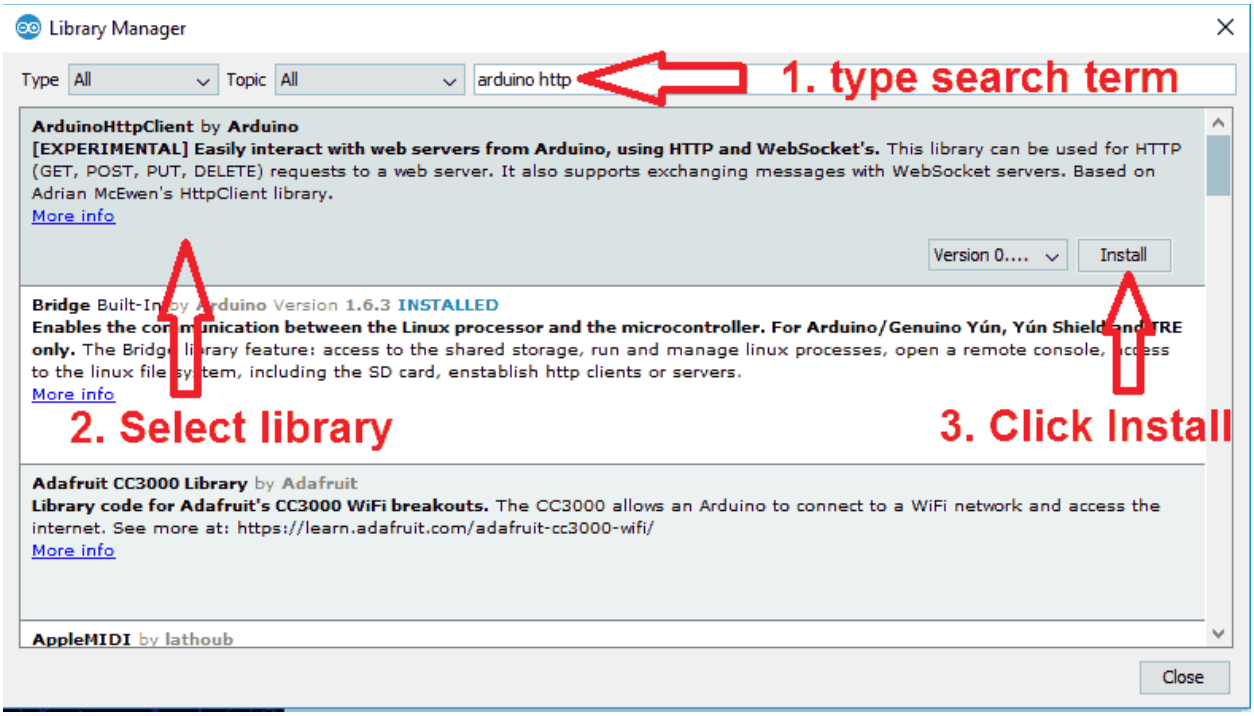
The library Manager window will pop up. Now enter **Adafruit IO Arduino** into the search box, and click Install on the **Adafruit IO Arduino library** option to install version 2.6.0 or higher.



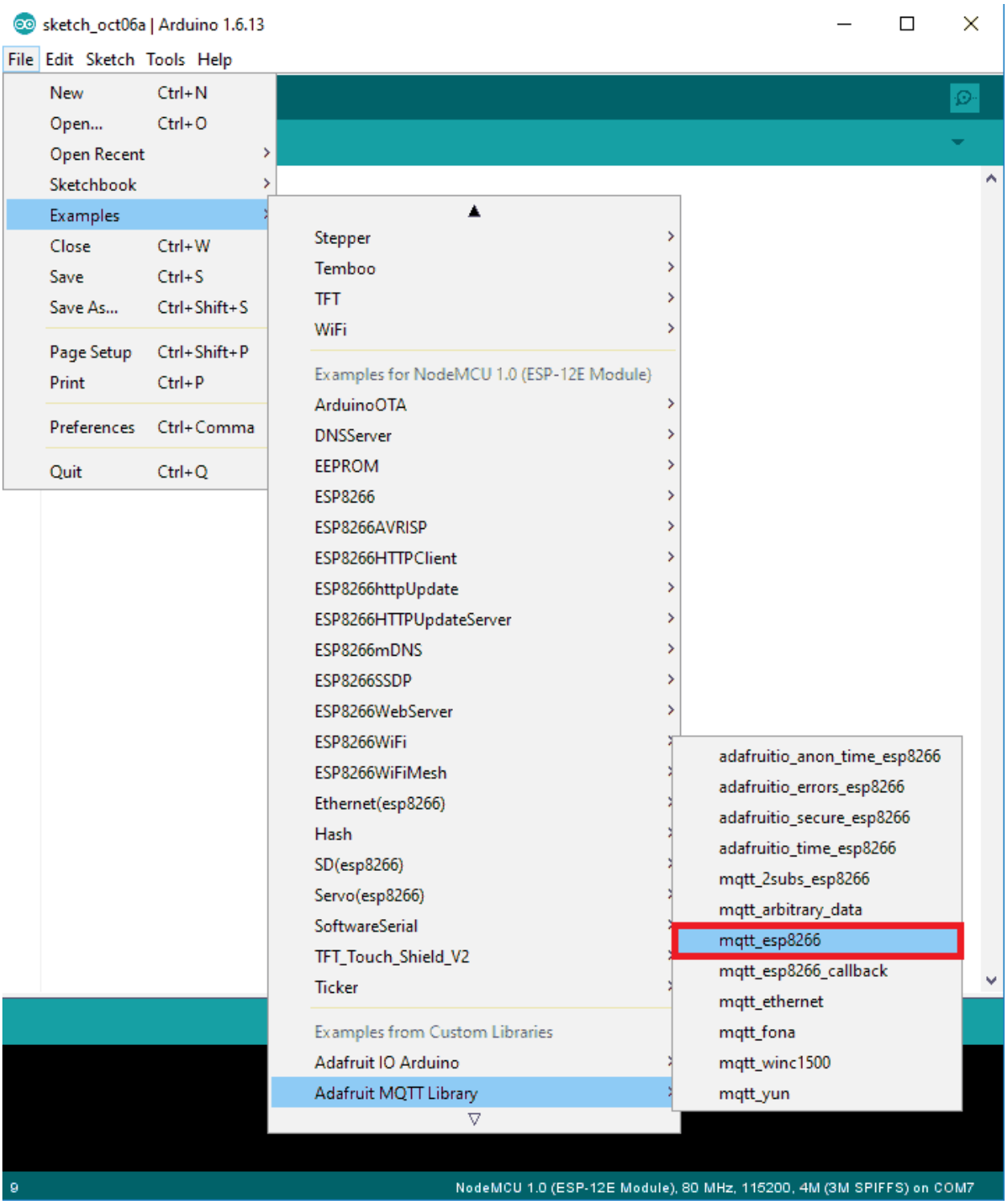
Now enter **Adafruit MQTT** into the search box, and click Install on the **Adafruit MQTT library** option to install version 0.17.0 or higher.



Now enter **Arduino Http Client** into the search box, and click Install on the **ArduinoHttpClient** library option to install version 0.3.0 or higher.



Now open example of Adafruit mqtt io dashboard. To open it navigate to **File -> Examples -> Adafruit MQTT Library -> mqtt_esp8266**



Now edit the wifi and Adafruit io credentials with correct information of example as shown in below image.

We have modified the mqtt_esp8266 example as per our above example as below

Arduino Sketch for MQTT Client

```
/*************************************************

 Adafruit MQTT Library ESP8266 Example


 Must use ESP8266 Arduino from:

  https://github.com/esp8266/Arduino


 Works great with Adafruit's Huzzah ESP board & Feather
```

```
 ----> https://www.adafruit.com/product/2471
 ----> https://www.adafruit.com/products/2821

 Adafruit invests time and resources providing this open source code,
 please support Adafruit and open-source hardware by purchasing
 products from Adafruit!

 Written by Tony DiCola for Adafruit Industries.
 MIT license, all text above must be included in any redistribution
 ****************************************************/
#include <ESP8266WiFi.h>
#include "Adafruit_MQTT.h"
#include "Adafruit_MQTT_Client.h"

/************************* WiFi Access Point *********************************/

#define WLAN_SSID       "...your SSID..."
#define WLAN_PASS       "...your password..."

/************************* Adafruit.io Setup *********************************/

#define AIO_SERVER      "io.adafruit.com"
#define AIO_SERVERPORT  1883                   // use 8883 for SSL
#define AIO_USERNAME    "...your AIO username (see https://accounts.adafruit.com)..."
#define AIO_KEY         "...your AIO key..."

/************ Global State (you don't need to change this!) ******************/

// Create an ESP8266 WiFiClient class to connect to the MQTT server.
```

```
WiFiClient client;
// or... use WiFiFlientSecure for SSL
//WiFiClientSecure client;

// Setup the MQTT client class by passing in the WiFi client and MQTT server and login details.
Adafruit_MQTT_Client mqtt(&client, AIO_SERVER, AIO_SERVERPORT, AIO_USERNAME, AIO_KEY);

/************************** Feeds ***********************************/

// Setup a feed called 'potValue' for publishing.
// Notice MQTT paths for AIO follow the form: <username>/feeds/<feedname>
Adafruit_MQTT_Publish potValue = Adafruit_MQTT_Publish(&mqtt, AIO_USERNAME "/feeds/potValue");

// Setup a feed called 'ledBrightness' for subscribing to changes.
Adafruit_MQTT_Subscribe ledBrightness = Adafruit_MQTT_Subscribe(&mqtt, AIO_USERNAME "/feeds/ledBrightness");

/************************* Sketch Code ***********************************/

// Bug workaround for Arduino 1.6.6, it seems to need a function declaration
// for some reason (only affects ESP8266, likely an arduino-builder bug).
void MQTT_connect();

uint8_t ledPin = D6;
uint16_t potAdcValue = 0;
uint16_t ledBrightValue = 0;

void setup() {
  Serial.begin(9600);
```

```
  delay(10);

  Serial.println(F("Adafruit MQTT demo"));

  // Connect to WiFi access point.
  Serial.println(); Serial.println();
  Serial.print("Connecting to ");
  Serial.println(WLAN_SSID);

  WiFi.begin(WLAN_SSID, WLAN_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println();

  Serial.println("WiFi connected");
  Serial.println("IP address: "); Serial.println(WiFi.localIP());

  // Setup MQTT subscription for ledBrightness feed.
  mqtt.subscribe(&ledBrightness);
}

void loop() {
  // Ensure the connection to the MQTT server is alive (this will make the first
  // connection and automatically reconnect when disconnected).  See the MQTT_conn
ect
  // function definition further below.
  MQTT_connect();

  // this is our 'wait for incoming subscription packets' busy subloop
  // try to spend your time here
```

```
Adafruit_MQTT_Subscribe *subscription;
while ((subscription = mqtt.readSubscription(200))) {
  if (subscription == &ledBrightness) {
    Serial.print(F("Got LED Brightness : "));
    ledBrightValue = atoi((char *)ledBrightness.lastread);
    Serial.println(ledBrightValue);
    analogWrite(ledPin, ledBrightValue);
  }
}

// Now we can publish stuff!
uint16_t AdcValue = analogRead(A0);
if((AdcValue > (potAdcValue + 7)) || (AdcValue < (potAdcValue - 7))){
  potAdcValue = AdcValue;
  Serial.print(F("Sending pot val "));
  Serial.print(potAdcValue);
  Serial.print("...");
  if (! potValue.publish(potAdcValue)) {
    Serial.println(F("Failed"));
  } else {
    Serial.println(F("OK!"));
  }
}
// ping the server to keep the mqtt connection alive
// NOT required if you are publishing once every KEEPALIVE seconds
/*
if(! mqtt.ping()) {
 mqtt.disconnect();
}
```

```
 */
}

// Function to connect and reconnect as necessary to the MQTT server.
// Should be called in the loop function and it will take care if connecting.
void MQTT_connect() {
  int8_t ret;

  // Stop if already connected.
  if (mqtt.connected()) {
    return;
  }

  Serial.print("Connecting to MQTT... ");

  uint8_t retries = 3;
  while ((ret = mqtt.connect()) != 0) { // connect will return 0 for connected
       Serial.println(mqtt.connectErrorString(ret));
       Serial.println("Retrying MQTT connection in 5 seconds...");
       mqtt.disconnect();
       delay(5000);  // wait 5 seconds
       retries--;
       if (retries == 0) {
         // basically die and wait for WDT to reset me
         while (1);
       }
  }
  Serial.println("MQTT Connected!");
}
```

Getting Date & Time From NTP Server With ESP8266 NodeMCU

Every now and then, you'll come across an idea where keeping time is a top priority. For example, consider a relay that must be activated at a specific time or a data logger that must store values at precise intervals.

The first thing that comes to mind is to use an RTC (Real Time Clock) chip. However, because these chips are not perfectly accurate, you must perform manual adjustments on a regular basis to keep them synchronized.

Instead, it is preferable to employ the Network Time Protocol (NTP). If your ESP8266 project has Internet access, you can obtain date and time (with a precision of a few milliseconds of UTC) for FREE. Also, you don't need any additional hardware.

What is an NTP?

NTP is an abbreviation for Network Time Protocol. It is a standard Internet Protocol (IP) for synchronizing computer clocks over a network.

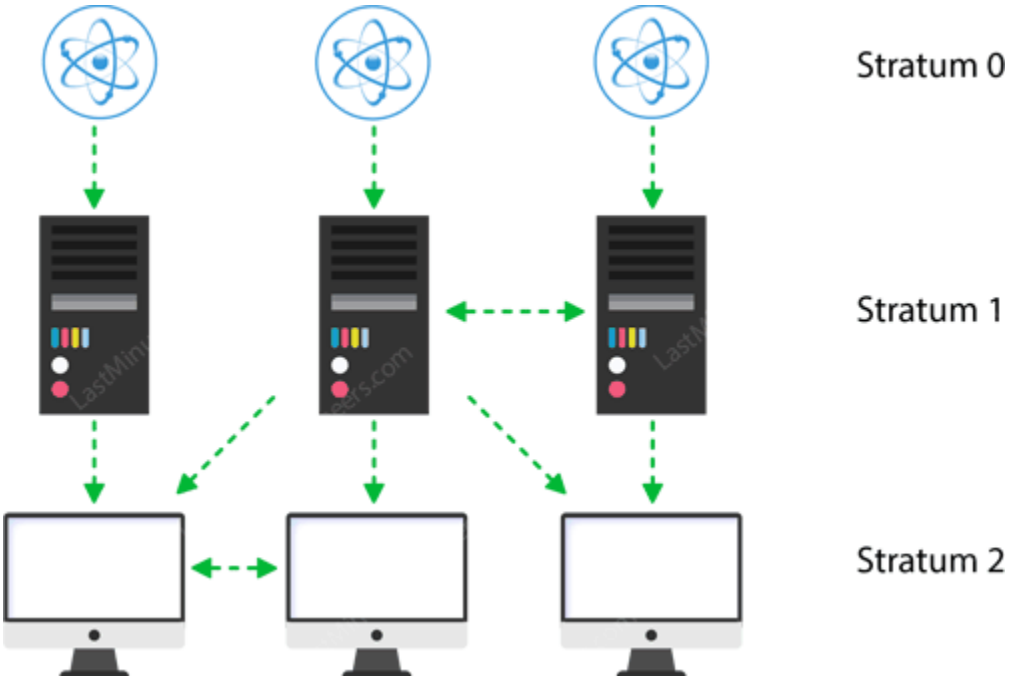
This protocol synchronizes all networked devices to Coordinated Universal Time (UTC) within a few milliseconds (50 milliseconds over the public Internet and under 5 milliseconds in a LAN environment).

Coordinated Universal Time (UTC) is a global time standard that is similar to GMT (Greenwich Mean Time). UTC does not change; it is the same all over the world.

The idea here is to use NTP to set the computer clocks to UTC and then apply any local time zone offset or daylight saving time offset. This allows us to synchronize our computer clocks regardless of location or time zone differences.

NTP Architecture

NTP employs a hierarchical architecture. Each level in the hierarchy is known as a stratum.



At the very top are high-precision timekeeping devices, such as atomic clocks, GPS or radio clocks, known as stratum 0 hardware clocks.

Stratum 1 servers have a direct connection to a stratum 0 hardware clock and therefore provide the most accurate time.

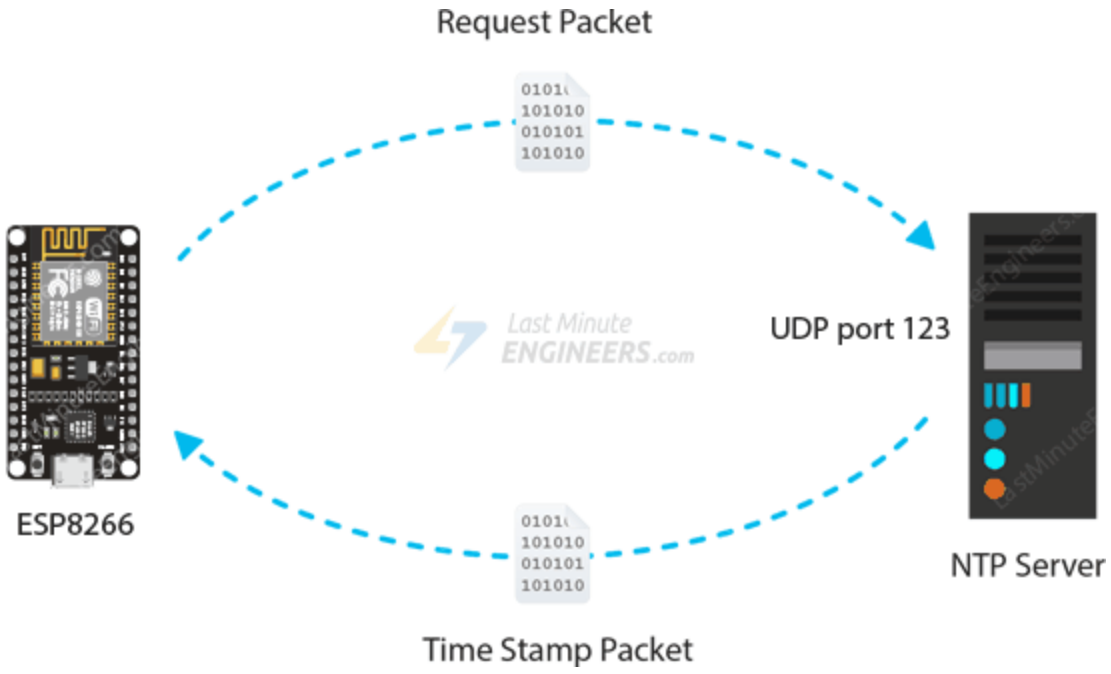
Each stratum in the hierarchy synchronizes with the stratum above and acts as a server for computers in lower stratums.

How NTP Works?

NTP can operate in a number of ways. The most common configuration is to operate in client-server mode.

The fundamental operating principle is as follows:

1. The client device, such as the ESP8266, connects to the NTP server via the User Datagram Protocol (UDP) on port 123.
2. The client then sends a request packet to the NTP server.
3. In response to this request, the NTP server sends a time stamp packet. A time stamp packet contains a variety of data, such as a UNIX timestamp, accuracy, delay, or timezone.
4. A client can then extract the current date and time from it.



Preparing the Arduino IDE

You should have the ESP8266 add-on installed in your Arduino IDE before proceeding with this tutorial. If you haven't installed it yet, follow the tutorial below.

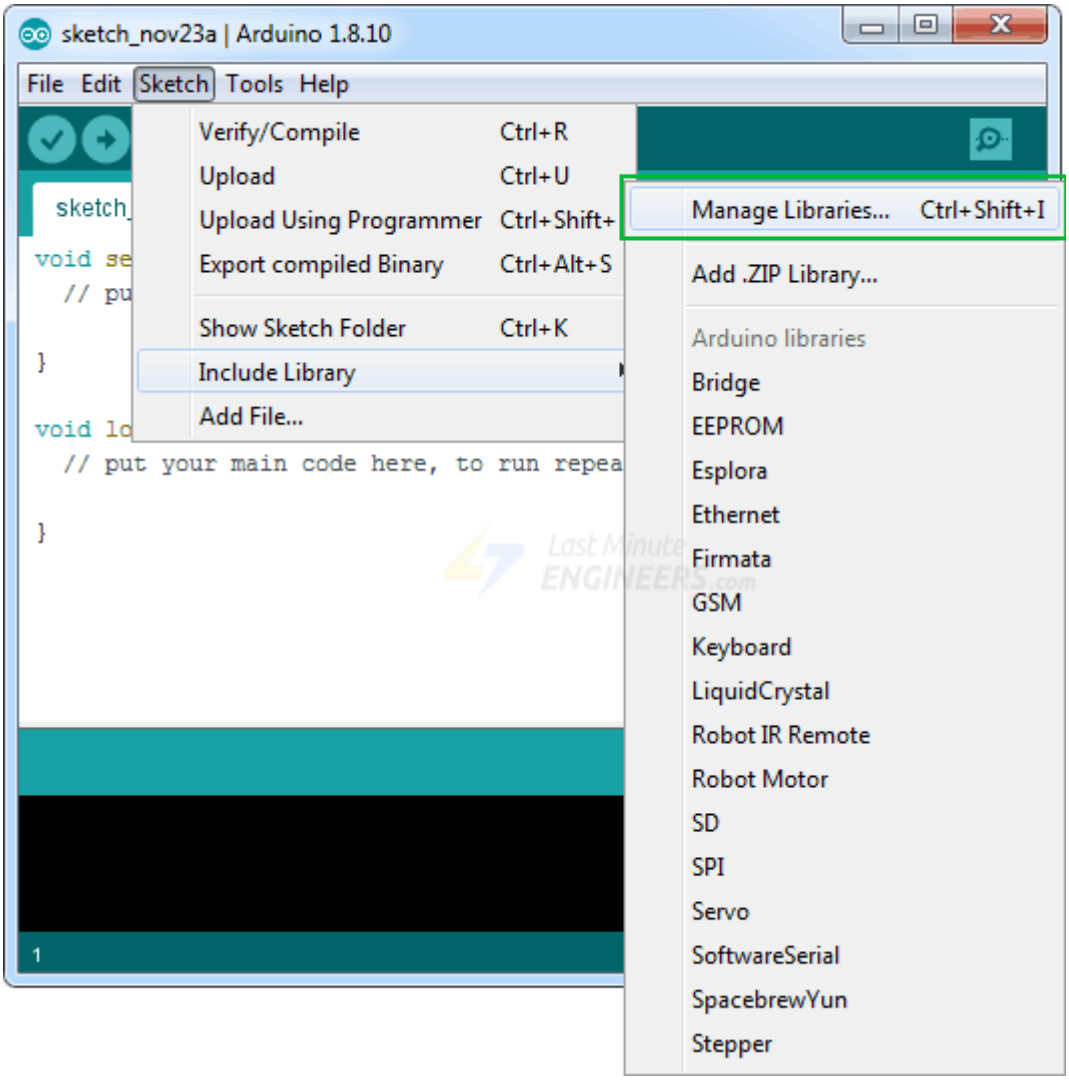
Insight Into ESP8266 NodeMCU Features & Using It With Arduino IDE

The Internet of Things (IoT) has been a trending field in the world of technology. It has changed the way we work. Physical objects and...

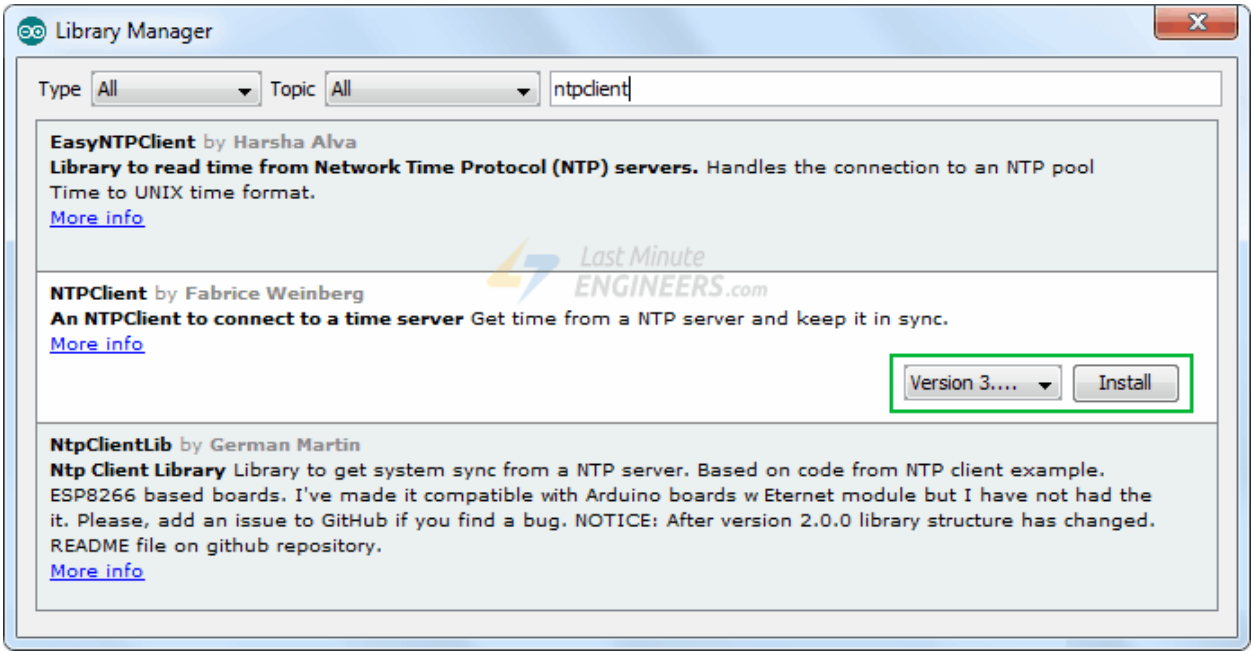
Installing the NTP Client Library

The NTP Client Library simplifies the process of retrieving time and date from an NTP server. Follow the steps below to install this library in your Arduino IDE.

Navigate to Sketch > Include Library > Manage Libraries... Wait for the Library Manager to download the libraries index and update the list of installed libraries.



Filter your search by entering ‘ntpclient’. Look for NTPClient by Fabrice Weinberg. Click on that entry and then choose Install.



Getting Date and Time from NTP Server

The sketch below will show you exactly how to get the date and time from the NTP Server.

```
#include <NTPClient.h>
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

const char *ssid     = "YOUR_SSID";
const char *password = "YOUR_PASS";

const long utcOffsetInSeconds = 3600;

char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday"};

// Define NTP Client to get time
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds);

void setup(){
```

```
  Serial.begin(115200);

  WiFi.begin(ssid, password);

  while ( WiFi.status() != WL_CONNECTED ) {
    delay ( 500 );
    Serial.print ( "." );
  }

  timeClient.begin();
}

void loop() {
  timeClient.update();

  Serial.print(daysOfTheWeek[timeClient.getDay()]);
  Serial.print(", ");
  Serial.print(timeClient.getHours());
  Serial.print(":");
  Serial.print(timeClient.getMinutes());
  Serial.print(":");
  Serial.println(timeClient.getSeconds());
  //Serial.println(timeClient.getFormattedTime());

  delay(1000);
}
```

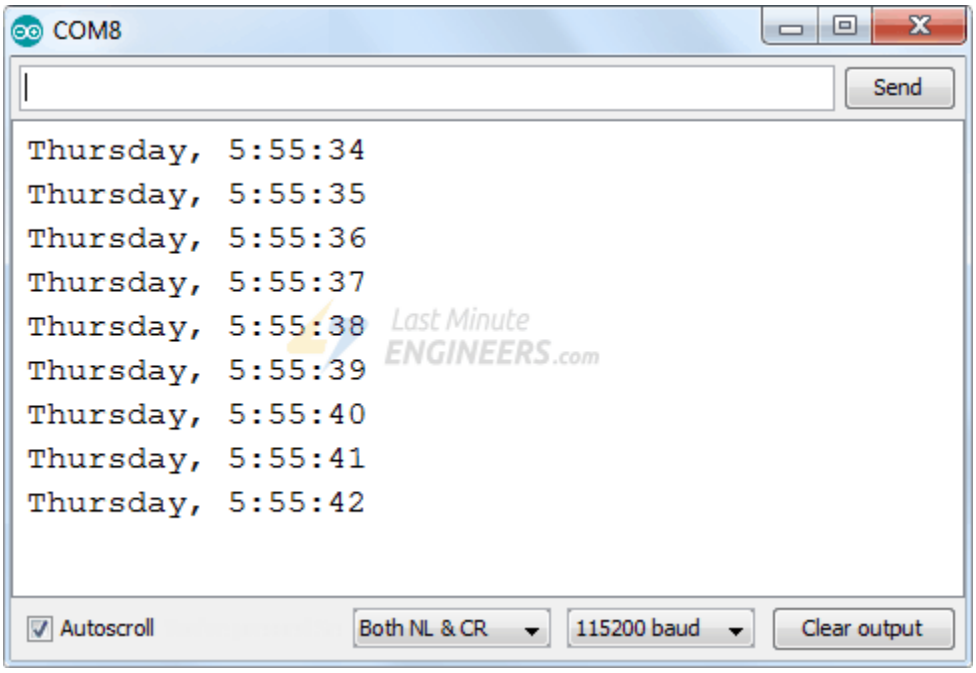
Before you start uploading the sketch, you'll need to make a few changes to make sure it'll work for you.

- Modify the following two variables with your network credentials so that the ESP8266 can connect to an existing network.
- ```
  const char* ssid     = "YOUR_SSID";
  const char* password = "YOUR_PASS";
  ```
- Adjust the UTC offset for your timezone (in seconds). Refer to the list of UTC time offsets. Here are some examples for various time zones:
  ```

- For UTC -5.00 : -5 * 60 * 60 : -18000
- For UTC +1.00 : 1 * 60 * 60 : 3600
- For UTC +0.00 : 0 * 60 * 60 : 0

```
const long utcOffsetInSeconds = 3600;
```

After uploading the sketch, press the RST button on your NodeMCU. The serial monitor should display the date and time every second.



Code Explanation

Let's take a quick look at the code to see how it works. To begin, we include the libraries required for this project.

- NTPClient.h is a time library that handles NTP server synchronization gracefully.
- ESP8266WiFi.h is a library containing the ESP8266-specific WiFi methods we will use to connect to a network.
- WiFiUdp.h library handles UDP protocol tasks such as opening a UDP port, sending and receiving UDP packets, and so on.

```
#include <NTPClient.h>
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
```

A few constants are defined, such as the SSID, WiFi password, and UTC offset. `daysOfTheWeek` 2D array is also defined.

```
const char *ssid     = "YOUR_SSID";
const char *password = "YOUR_PASS";
const long utcOffsetInSeconds = 3600;
char daysOfTheWeek[7][12] = {"Sunday", "Monday", "Tuesday",
"Wednesday", "Thursday", "Friday", "Saturday"};
```

In addition, the address of the NTP Server is specified. pool.ntp.org is a great open NTP project for this kind of thing.

```
WiFiUDP ntpUDP;
NTPClient timeClient(ntpUDP, "pool.ntp.org", utcOffsetInSeconds);
```

pool.ntp.org automatically selects time servers that are physically close to you. However, if you want to select a specific server, use one of the pool.ntp.org sub-zones.

Area	HostName
Worldwide	pool.ntp.org
Asia	asia.pool.ntp.org
Europe	europe.pool.ntp.org
North America	north-america.pool.ntp.org
Oceania	oceania.pool.ntp.org
South America	south-america.pool.ntp.org

In the setup section, we first establish serial communication with the PC and then connect to the WiFi network by calling the `WiFi.begin()` function.

```
Serial.begin(115200);

WiFi.begin(ssid, password);

while ( WiFi.status() != WL_CONNECTED ) {
  delay ( 500 );
  Serial.print ( "." );
}
```

Once the ESP8266 is connected to the network, we use the `begin()` function to initialize the NTP client.

```
timeClient.begin();
```

Now we simply call the `update()` function to obtain the current date and time. This function sends a request packet to an NTP server and parses the received time stamp packet into a readable format.

```
timeClient.update();
```

You can retrieve the current date and time by calling the NTP Client object's methods.

```
Serial.print(daysOfTheWeek[timeClient.getDay()]);
Serial.print(", ");
Serial.print(timeClient.getHours());
Serial.print(":");
Serial.print(timeClient.getMinutes());
Serial.print(":");
Serial.println(timeClient.getSeconds());
```

Performing MQTT Communication with ESP8266/NodeMCU using Arduino IDE

Published December 23, 2020 0

Debasis Parida
Author

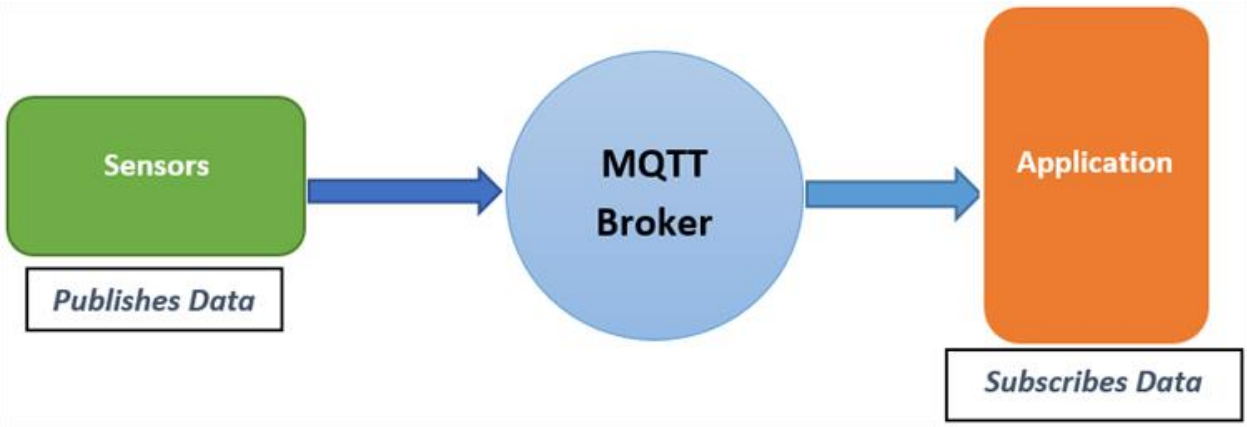
MQTT Communication With Esp8266 Using Arduino IDE

Over the past few years, **IoT (Internet of Things)** devices have become an indistinguishable part of our daily lives. From smart homes, smart bulbs to smart appliances; creators and developers are incorporating this technology to create a network of connected devices that makes our day-to-day life a little more exciting. All this has been made possible because of the ease of communication.

There are many possible ways to communicate among devices, but in commercial and hobby products, a single protocol that is commonly used is **Message Queuing Telemetry Transport (MQTT)**. We previously built a Voice-Controlled FM Radio using Arduino and Google Assistant that utilizes MQTT to communicate with the NodeMCU board. Do check it out if that sounds interesting to you.

In this project, we will be using a free and popular **Eclipse MQTT broker** and learn how to connect an IoT device (in our case, it's a NodeMCU module) to an MQTT broker and transfer data among the **MQTT broker and NodeMCU**.

What is MQTT Protocol?



Before we proceed any further, it's better to have a clear idea about the **MQTT (Message Queuing Telemetry Transport) protocol**. It is a lightweight messaging protocol that uses the publish/subscribe method and translates messages between multiple devices. Using MQTT protocol, we can also send/receive data and control various output devices, like read sensor data, etc. It's developed on top of TCP, which is why it's faster than similar protocols like HTTP. Other than that, it has many other advantages over other protocols like its very lightweight, so it doesn't consume excess memory, it can work with very less network bandwidth, on top of that, it has a robust security protocol inbuilt. These features make it suitable for many applications.

How MQTT Works?

In order to understand the working of the MQTT protocol, we just need to understand three basic things; the above diagram shows that. Also, we have explained it below in the article.

MQTT Client:

An **MQTT client** is any device (it can be a microcontroller or a server) that runs MQTT functions and communicates with a central server, which is known as the "**broker**." The broker handles the data communication between the connected clients.

MQTT Publisher:

When a client wants to send any information, the client is known as a "Publisher". The publisher will publish the information on a particular topic. "**Topic**" is a path where we can publish/subscribe messages. The broker then sends the information published by the user to the clients (also known as **Subscriber**) that have subscribed to that specific topic.

MQTT Subscriber:

The ***MQTT Subscriber*** subscribes to topics on an ***MQTT*** broker to read the messages sent by the broker.

The Eclipse Mosquitto broker

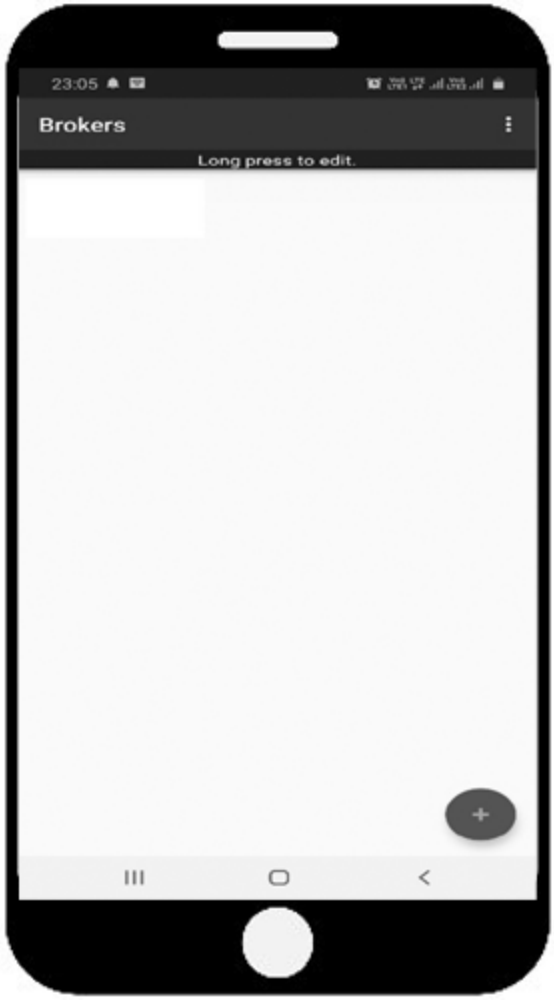
Eclipse Mosquitto is an open-source MQTT broker, which is lightweight and is suitable for use on IoT devices for communication. The **MQTT protocol** provides a lightweight method of transferring information using a publish/subscribe model. If you want to learn more about the topic, you can visit the official mosquito website.

Setting up Eclipse Mosquitto broker:

In order to establish communication with the broker, we need to set it up first. In this project, an Android application is used to **publish** and **subscribe** to the information with the Broker. The following steps will give you a better idea of the setup process.

Step-1:

First, download any “MQTT client” application available in the Google Play Store / App Store and install it. In this project, an application named **“MQTT client”** is used, which looks like the image shown below.



Step-2:

Click on the “+” sign to list the additional options in the application, where we are going to add a new broker. When the button is clicked, a new screen appears as shown below.

Step-3:

Thereafter, the details of the broker need to be filled in the required field. First, click on the option “Enabled” shown in the Application. In this project, the **Eclipse MQTT broker** is used, the details which are to be filled are given below:

Nick Name: Give a Name of Your Preference

Host: mqtt.eclipse.org

Port: 1883

Client ID: Give an ID of Your Preference

The above details need to be filled in their respective fields. All other fields are not necessary and can be left blank. After successful completion, click on the save button to save the Broker details.

Once done, the android application setup process is over and now we can move on to the hardware side of things.

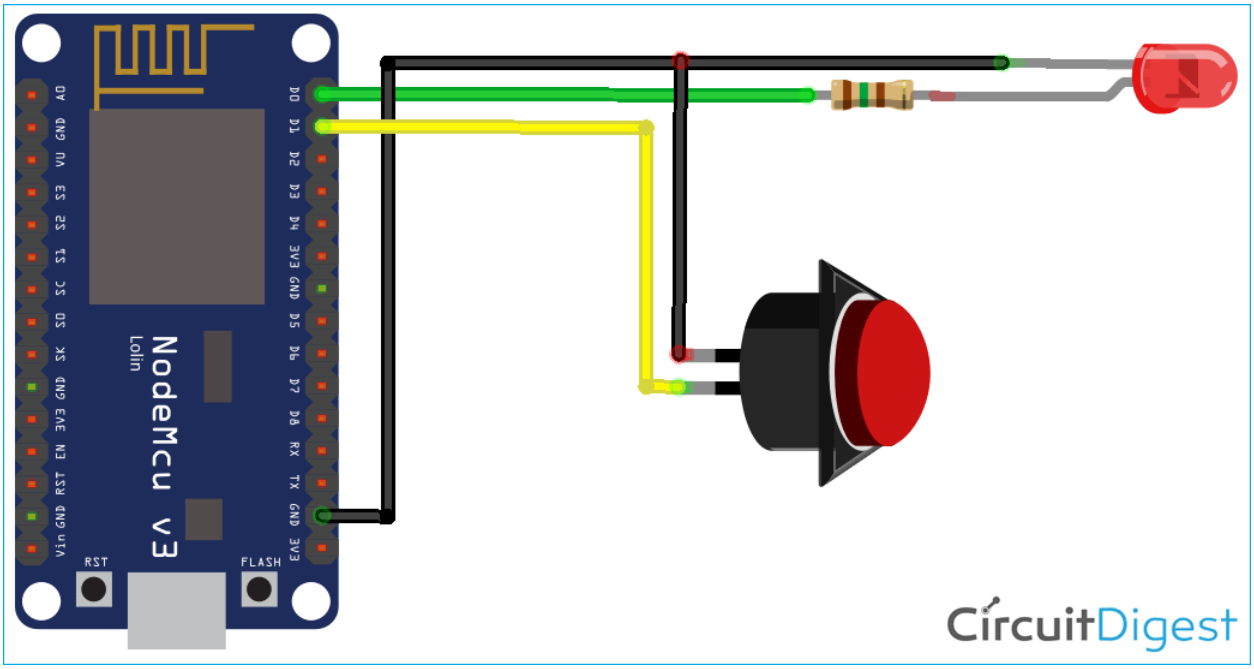
Components Required

A complete list of required parts is described below. As this circuit is simple, you can find all the necessary parts at your local hobby store.

- NodeMCU
- LED
- Breadboard
- Connecting wires
- Programming cable

Eclipse MQTT Test-Circuit - Schematic

The circuit diagram for the Basic MQTT project is given below:



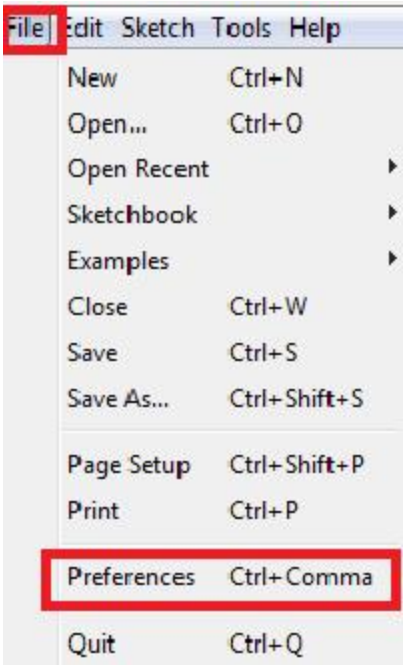
Programming ESP8266 to Establish Communication with Broker

A simple **Arduino code** takes care of all the necessary communications between the MQTT broker and the NodeMCU. In this section, we will learn how this functionality works in detail.

Setup Arduino IDE and Upload the Code:

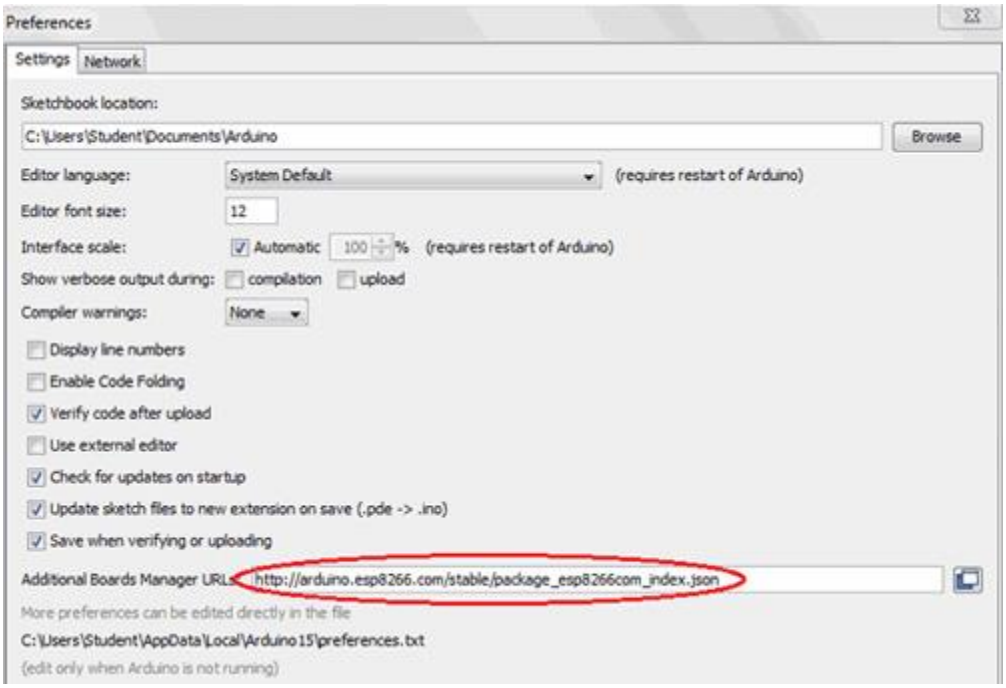
If you are uploading the code to the **NodeMCU** for the first time, you need to set-up the **Arduino IDE** first. To do that, just follow the simple instruction given below.

First, Open Arduino IDE, then go to ***File–>Preferences–>Settings*.**



Next, copy the below URL and paste it in the **'Additional Board Manager URL**' field, and click 'Ok'. You can check the image below to know how we have done that.

Link: ***https://arduino.esp8266.com/stable/package_esp8266com_index.json***



Next, go to ***Tools > Board > Boards Manager***. In the Board's Manager window, Type ***ESP*8266** in the search box and hit enter. Then select the latest version from the dropdown and click on install. The image below will give you a clear idea.

Finally, once the installation is completed, go to ***Tools ->Board -> and select NodeMCU 1.0(ESP-12E Module).*** Now, you can program NodeMCU with Arduino IDE. As we have finished setting

up the Arduino IDE, we can now upload the complete code. But first, read on the quick explanation of the whole code.

Firstly, we have included ***"ESP8266WiFi.h"*** for using ESP8266 and ***"PubSubClient.h"*** for MQTT.

You can find the ESP8266 library prebuilt inside the Arduino library, but you need to download the PubSubClient library from its associated GitHub repository.

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

Then, define the network credentials such as your Wi-Fi username and password. Replace your credentials in place of "admin" & "12345678" respectively.

```
const char* ssid = "admin";
const char* password =  "12345678";
```

Next, we need to configure the **MQTT server**. We have used the Eclipse MQTT server for this project, which is why the server address is given as *"mqtt.eclipse.org".* But if you plan to use any other server like Mosquitto, Adafruit, then you can replace it with your specific server address and port number.

```
const char* mqtt_server = "mqtt.eclipse.org";
const int mqtt_port = 1883;
```

Next, the instances are created for class ***WiFiClient*** and ***PubSubClient***, which will be used throughout the program.

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

In the *setup()* section, we call the ***WiFi.begin()*** first, calling this method will connect the ESP to your preferred HotSpot.

```
WiFi.begin(ssid, password);
```

Next, we check for a successful network connection using the ***WiFi.status()*** *method.* After a successful connection, a message is printed on Serial Monitor with the SSID of the connected network.

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }
  Serial.print("Connected to WiFi :");
  Serial.println(WiFi.SSID());
```

Now, we need to create a broker. For that, we have used the ***setServer*** method; this method takes two arguments which we have predefined earlier. Now, if we want to receive messages from the server, we need to create a **callback function. For that, we are using the *setCallback(callback)*** method.

```
client.setServer(mqtt_server, mqtt_port);
client.setCallback(MQTTcallback);
```

Now, we have used the ***connect (clientID)*** function to connect to the ESP8266 client. Here **clientID** is the name of the client, and it must be unique. If it is connected, then a success message can be shown inside the serial monitor.

```
if (client.connect("ESP8266"))
    {
      Serial.println("connected");
    }
```

```
    else
    {
      Serial.print("failed with state ")
      Serial.println(client.state());
      delay(2000);
    }
```

Next, we call the ***client.subscribe(),*** a built-in MQTT function, which is used to subscribe to a particular topic. For this project, we have used "**esp/test**" as our subscriber name.

```
client.subscribe("esp/test");
```

Now, the ***MQTTcallback*** function is called to check whether any updated information is available or not. If new data is available, this function handles the received data and prints a message in the serial monitor with the original message and topic name where the message is received.

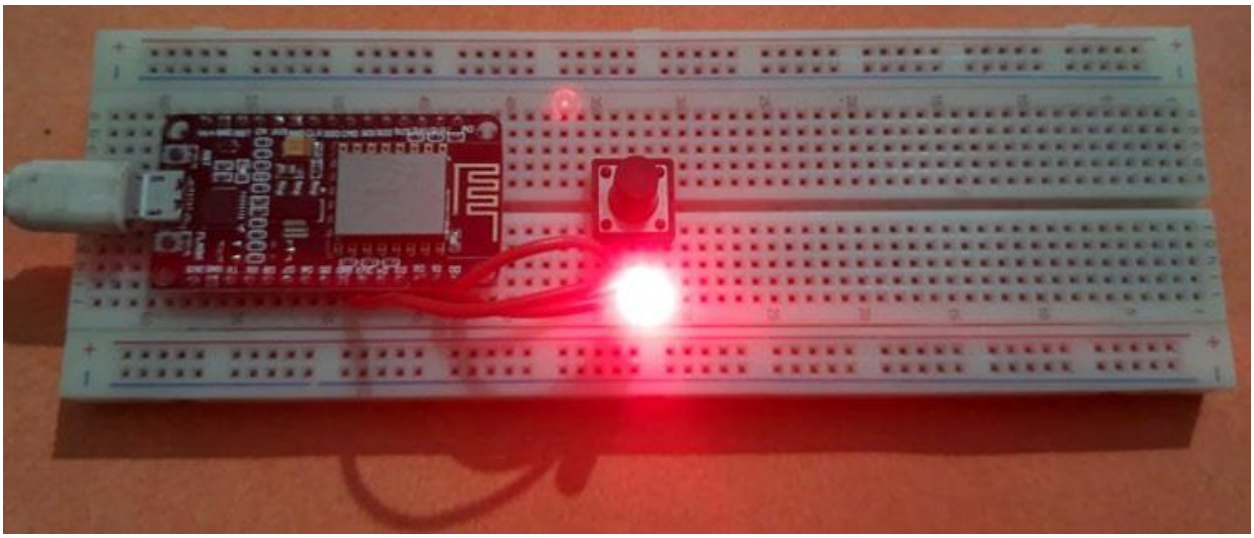
Next, we convert the messages into a string, so that it can be compared and checked for any triggering actions. In this project, an LED is turned ON/OFF using MQTT commands, as shown in the code below.

```
for (int i = 0; i < length; i++)
  {
    message = message + (char)payload[i];
  }
  Serial.print(message);
  if (message == "on")
  {
    digitalWrite(LED, HIGH);
  }
```

Finally, to publish the information on the topic. The ***client.publish()*** function is used. In this project, a push-button status is checked, if the button is pressed, then a message is published to a Topic "**esp/test1**" as shown below.

```
if(digitalRead(D1)==0))
  {
  client.publish("esp/test1", "Hello from ESP8266");
  }
  else;
client.loop();
```

Testing MQTT with ESP8266 using Arduino



For our final testing, we are going to use the Android application, which we have set up earlier.

Open the MQTT client application, and make sure your mobile has an active internet connection. Also, the hotspot to which the NodeMCU is connected should have an active internet connection. Once everything is connected to the internet, we are going to send a **"Hello from ESP8266"** string from the ESP module, which will be reflected inside the Android app, and we will get a

notification. Next, we will send a string from the Android app, which will turn ON an LED that is connected to the ESP8266 Node MCU board.

Step-1: (Subscribe to the Topic):

Click on the Saved MQTT option on App, which we have configured earlier. It will pop up a screen, where it is prompted to **"Subscribe to a Topic".** We have previously configured the topic as "***esp/test1***". So, in the Android app, we will write "***esp/test1***". Click on Subscribe, doing so will present you with a screen like below, where it will be written like "No message received" from the particular Topic.

Now, click the button 'Connected' which is connected to the nodeMCU. Now as per our code, a message "**Hello from ESP8266**" will be published to the Topic and there will be a notification on the screen with the message received as shown below.

Step-2: Publish to the Topic:

Now to publish in the Topic, Click on the UP ARROW button of the Application, and it will open a screen as shown below.

Now, In the Topic field, write "**esp/test**" and in the message field, write "**on**" or "**off**" to turn on and off the LED respectively. For example, if "on" is published to the Topic, then the LED will be turned on and if "off" is published to the Topic, then the LED will be turned off.

I hope you liked the article and learned something new. If you have any questions regarding this article, please feel free to comment below or you can use our forum instead.

Code

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#define LED D0
```

```
const char* ssid = "admin";
const char* password =  "12345678";
const char* mqtt_server = "mqtt.eclipse.org";
const int mqtt_port = 1883;
WiFiClient espClient;
PubSubClient client(espClient);
void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(D1,INPUT_PULLUP);
  Serial.begin(115200);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.println("Connecting to WiFi..");
  }
  Serial.print("Connected to WiFi :");
  Serial.println(WiFi.SSID());
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(MQTTcallback);
  while (!client.connected())
  {
    Serial.println("Connecting to MQTT...");
    if (client.connect("ESP8266"))
    {
      Serial.println("connected");
    }
    else
    {
      Serial.print("failed with state ");
      Serial.println(client.state());
      delay(2000);
    }
```

```
  }
  client.subscribe("esp/test");
}
void MQTTcallback(char* topic, byte* payload, unsigned int length)
{
  Serial.print("Message received in topic: ");
  Serial.println(topic);
  Serial.print("Message:");
  String message;
  for (int i = 0; i < length; i++)
  {
    message = message + (char)payload[i];
  }
  Serial.print(message);
  if (message == "on")
  {
    digitalWrite(LED, HIGH);
  }
  else if (message == "off")
  {
    digitalWrite(LED, LOW);
  }
  Serial.println();
  Serial.println("-----------------------");
}
void loop()
{
  if(digitalRead(D1)==0)
  {
  client.publish("esp/test1", "Hello from ESP8266");
  delay(1000);
  }
  else;
  client.loop();
```

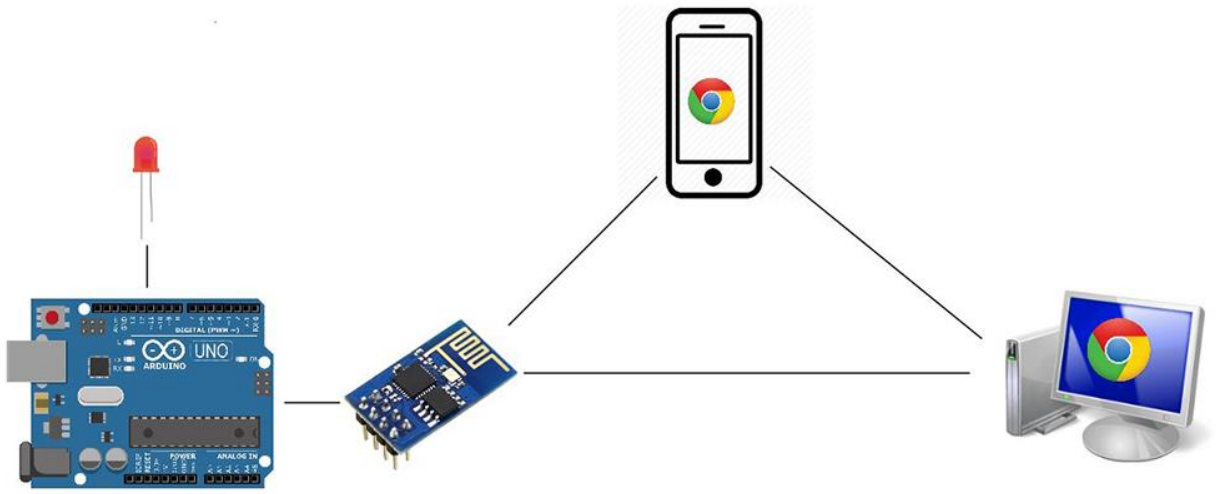
```
}
```

Creating Arduino web server and controlling things via WiFi – Tutorial

Frank Donald December 6, 2017 13 Comments

IOT tutorials

arduino uno, Development boards, diy projects, IOT, Network



IOT has been all around us for quite some time now and it has almost revolutionized the way machines communicate among themselves. As a starting point here is a simple project tutorial on creating Arduino web server that will help you to open the doors to IOT. In this tutorial we are going to host a simple web server as shown in the above image.
Remember you can host a webpage in your ESP8266 but the controlling functionality is pretty less in it. Given that there are only two GPIO pins in it. Also it doesn't have of PWM, timer and other options. ESP866 serves as a communicating medium which enables Arduino to join a LAN. We can use our WiFi enabled mobile phones/ laptops to form a LAN with Arduino. From our devices we will access the Arduino web server, fetch the webpage from it and control the modules or components connected to it. I have made a video explaining this entire tutorial – please do check that out

COMPONENTS REQUIRED:

1. Arduino Uno
2. ESP8266 WiFi module
3. LEDs
4. Connecting wires
5. Birdirectional Logic converter
6. Mobile or Laptop or other WiFi enabled devices

QUICK BRIEF ABOUT ESP8266:

I presume most of us are familiar about ESP8266 chip. For those who are not aware of it, ESP8266 is a versatile low cost WiFi chip that is capable of providing WiFi connectivity to your embedded projects. It can work in both Station and AP mode which makes it a perfect choice for IOT projects. Apart from this it also has two GPIO pins which can do some basic turning ON/OFF.

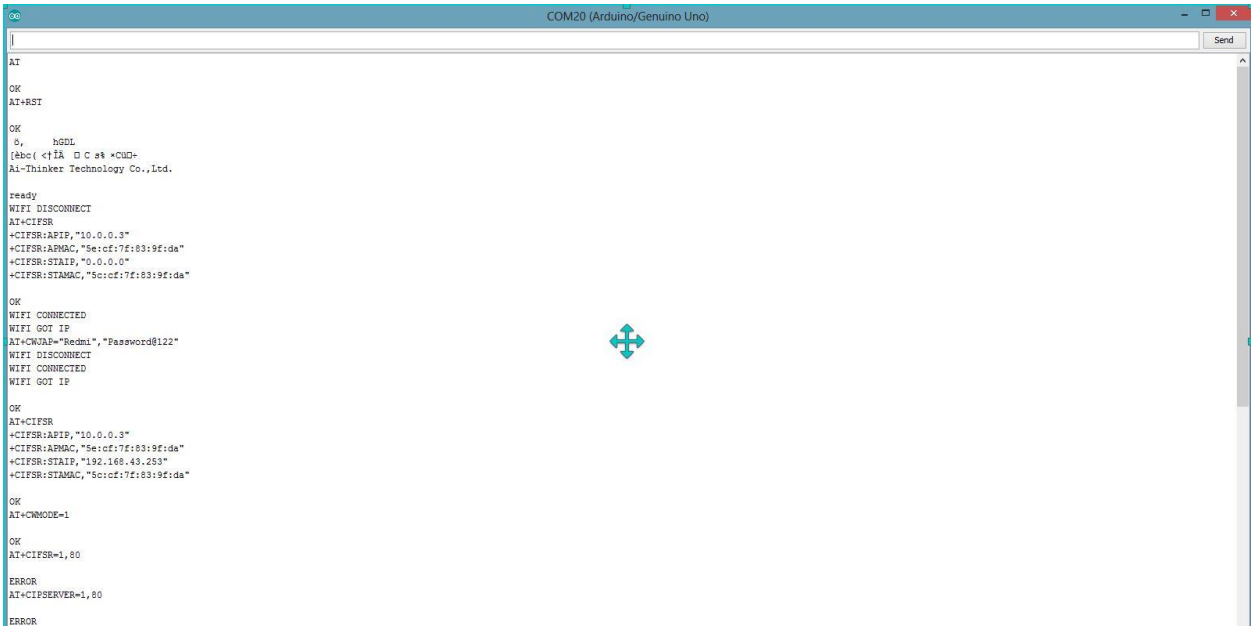
USAGE IN CREATING ARDUINO WEB SERVER:

In this project the whole purpose of ESP8266 is to provide a WiFi link between Arduino and our mobile device. We are creating a Local Area Network (LAN) by using ESP8266 as Station and connect it to Mobile/ Laptop (WiFi enabled devices) which serves as AP/Hotspot. Any devices connected to this AP will be within this LAN (i.e ESP8266 + Arduino and Mobile devices). Therefore they can communicate with each other and access the Arduino web server. In order to do all this we should program Arduino to feed a specific set of AT commands to ESP8266.

AT COMMANDS:

Predefined AT commands will decide all the parameters of WiFi link in ESP module. Using this we can control the way ESP8266 make or release the connection with other WiFi devices. This includes Setting AP password, Joining AP's, Creating server and so on. Refer this link for "Complete AT command list" used in ESP8266

ESP8266 RESPONSE:



In return to the input AT commands, ESP8266 will acknowledge with response codes which indicate whether the input AT command was accepted or declined by the device. This whole command input and response will be transmitted via Serial communication /UART.

To keep this simple, for majority of commands ESP8266 will respond with given Input command followed by the characters "OK\r\n". For example if we send the command AT then the ESP response will be

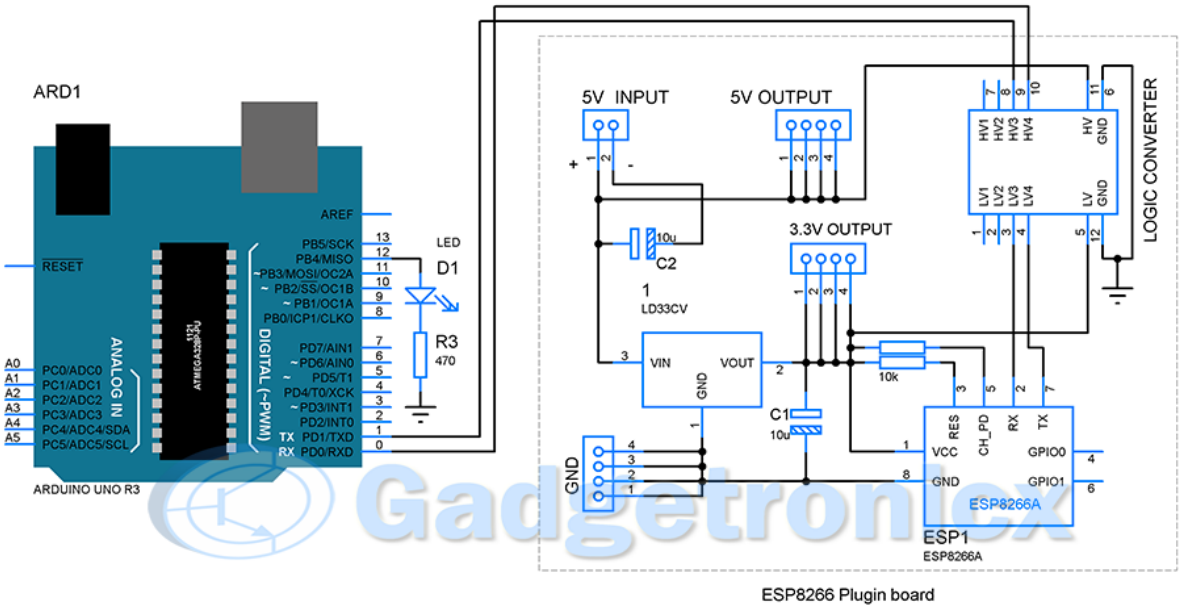
AT\r\n
\r\n
OK\r\n

This is common response for most of the commands. There are also other response codes that we would need to monitor for this project, which will be discussed later in this article.

ERROR DETECTION AND RECOVERY:

ESP8266 modules can be very buggy and it can be very prone to errors while working with Super fast Arduino. Considering this we need to write the code in such a way that Arduino can detect error response from ESP modules. In order to do that we need to check for appropriate response from ESP module for every Input command.

SCHEMATIC DIAGRAM:



From the above schematic diagram, you can see that Arduino is connected to a Bidirectional Logic converter LG1 which basically converts the logic from 5v to 3.3v and vice versa. This is to help both Arduino and ESP module to identify the logic 0 and 1 in order to communicate with each other via UART. Apart from that the schematic diagram is pretty straight forward. The headers in the Plug-in board section is for providing easier connectivity for ESP8266 when using it with Arduino or any other developmental boards.

TESTING YOUR ESP8266:

I strongly recommend this step since ESP8266 can be buggy and it's quite necessary to have this tested. Upload the bare minimum code to the Arduino and connect the ESP8266 as shown in the schematic. Once done use the following commands, but remember to select "Both NL & CR" & Baud rate as 56700 which is default baud rate used by ESP module.

AT – OK
AT+RST
AT+CWMODE=3
AT+CWJAP="SSID","Password"

The above command will make your ESP8266 to join the network you wish and it will remember the AP even after powered down. This is important in terms of this project because the code is developed in an assumption that ESP8266 was already connected to AP or hot spot of mobile device which we will use. This is because connecting to AP through the code seems to make things complex with no considerable benefits to this project.

AT+CIOBAUD=9600

This command is to alter the baud rate at which the ESP module operate. I had to do this since ESP modules worked better with this baud rate than higher speed.

PROJECT WORKING:

Arduino web server can be accessed by the devices within that LAN to control the LED's or other components. ESP8266 will remember the AP's that it has been connected to previously. It will get connected to AP as soon as hotspot in your device is turned ON. Once connected, a LAN will be created. Once Arduino detects a connection via ESP8266, it will start sending commands to ESP module to create a server. I have added a LED to indicate server creation. Mostly it will take about two seconds from the moment ESP is connected to WiFi AP.

Once the Arduino web server is ready, open any browser in the devices connected to the LAN and access the IP address assigned to ESP8266 module. Arduino will send a piece of HTML code to the client via ESP module and a webpage will be displayed on your device browser. HTML codes from the Arduino were built to display button in the webpage. Using these buttons user can feed input back to Arduino. Based on the received inputs, Arduino will activate/deactivate LED's or any other devices that are connected to it.

ESP COMMANDS USED FOR CREATING ARDUINO WEB SERVER:

These are the list of commands that we are going to use to create this Arduino web server.

"AT+RST" – Reset the ESP8266 module
"AT" – Response check
"AT+CIPMODE=3" – Set the module as both Station and AP
"AT+CIPMUX=1" – Setting to allow multiple connections
"AT+CIPSTA=<Local IP address>" – Assign any local IP address but make sure it is compatible with your device.
"AT+CIPSERVER=1,80" – Creating server with port 80
"AT+CIPSEND=<connection ID>,<Byte count>

CHOOSING IP ADDRESS:

I manually set a static IP to my ESP8266 module even though my mobile DHCP will assign IP when connecting with it. I am doing this to make sure I know the IP address of ESP device and I can connect to it without any problem. Without doing this DHCP in my mobile phone will assign IP address which is currently free and you will not know this while using this. But you should take care when choosing your IP address since local IP address differs from device to device. I have read in a documentation that Android phones are programmed to use 192.168.x.x to assign for the devices connecting with it locally. So I have used the IP 192.168.43.253 to assign for ESP module. So search in internet for the IP your devices will use and add it in the code.

HTML WEBPAGE:

So now the server is created in ESP8266. When I am saying server is created, it means that ESP8266 is ready to take client request (that is the browser). Remember ESP module serves only as a communicating medium here, so it's the Arduino here that respond with the HTML webpage when a client request is detected.

```
<!DOCTYPE html>
```

```
<html>

<body>

<h1>Welcome to Arduino LED control Wizard</h1><p>

You can turn ON/OFF LED connected to Arduino using the buttons given
below</p>

<p><b>TURN ON</b>        <a
href="ON"><button>ON</button></a></br></p>

<p><b>TURN OFF</b>       <a
href="OFF"><button>OFF</button></a></br></p>

</body>

</html>
```

ALGORITHM:

1. Input “AT+RST” command and wait until ESP8266 connects with AP.
2. Check for “IP\r\n” in the last four characters of ESP response, this indicate that WiFi got IP.
3. Once connected start feeding the commands to activate the server mode in ESP8266
4. Monitor the response for each input commands.
5. Proceed with the next command once the last four characters of response is “OK\r\n” throughout the Server sequence commands.
6. Force Arduino to wait for the client request from any browser connected within the LAN.
7. Once request is received check for connection ID provided by ESP8266 and add it to the command “AT+CIPSEND”
8. Check for the character ‘>’ in the response string from EP module.
9. In case of incorrect response repeat step 7.
10. Once correct response is verified send the webpage HTML code from Arduino through the ESP module to client browser.
11. Wait until you receive response from ESP8266 module and check for the last four characters “OK\r\n” in the response string. Webpage should have been successfully displayed in the browser now.
12. Force Arduino to monitor the incoming GET request from Client (button press)
13. Once client GET request is received check for URL parameters in the request, for example GET: /ON , in which /ON is the URL parameter.
14. Activate/ Deactivate LED/ other devices or do something based on the URL parameter in the response.

WORKING VIDEO:

CODE:

```
/*Simple code to create webserver in Arduino and control things within the
network

 Created by Frank Donald */
```

```
char Input_buffer[550];

boolean command_flag=false;

int i,j;

boolean stringComplete=false;

boolean request_processed=false;

boolean WiFiConnect=false;

short int command_count=0;

char send_bytes[]="AT+CIPSEND= ,379\r\n";

boolean client_request=false;

boolean flag=true;

char connection_id;

short int page_input_pos;



void setup() {

  pinMode(13, OUTPUT);

  pinMode(12, OUTPUT);

  digitalWrite(13,LOW);

  digitalWrite(12,LOW);

  Serial.begin(9600);

  delay(5000);

  Serial.print("AT+RST\r\n");

  delay(200);

  while(WiFiConnect==false)                                  //Wait till
Connecting to WiFi hotspot

  {
```

```
    serial_check();

    WiFiCheck();                                          //WiFi check
subroutine

  }



  while(command_flag==false)                             //Looping for commands

  {

    command_input();

  }

  clear_buffer();                                         //Clearing the array

  while(client_request==false) //Waitingfor anyclient/browser to initiate the
request

  {

    serial_check();

    brow_req();                                           //sub routine to
check browser request

    brow_resp();                                          //sub routine for
response from Arduino

  }

  flag=false;

}



void loop() {

  serial_check();

  page_input_pos=Search_webrequest();                     //Monitor webpage
indefinitely

  activate();

}
```

```
void activate()

{

  if(flag==true)

  {


if(Input_buffer[page_input_pos]=='O'&&Input_buffer[page_input_pos+1]=='N')

    digitalWrite(12,HIGH);

    else
if(Input_buffer[page_input_pos]=='O'&&Input_buffer[page_input_pos+1]=='F')

    digitalWrite(12,LOW);

  }

}



void WiFiCheck()                                          //Check whether WiFi
is connected

{

  if(stringComplete==true&&Input_buffer[j-4]=='I')   //Check for the status
WIFI GOT IP

  {

   clear_buffer();

   WiFiConnect=true;

   delay(500);

  }

  else

  {
```

```
    clear_buffer();
    stringComplete=false;
  }
}

int Search_webrequest()                                    //repeated loop to
check button inputs from Webpage
{
  if(stringComplete==true)
  {
    for(i=0;i<j-2;i++)
    {

if(Input_buffer[i]=='G'&&Input_buffer[i+1]=='E'&&Input_buffer[i+2]=='T')
      {
        flag=true;
        return i+5;
      }
    }
  }
}

void brow_req()                                            //sub routine
to monitor the browser request
{
  if(stringComplete==true&&request_processed==false)          //Checking for
presence of char in input buffer and check whether the client request has
already processed
```

```
   {
      if(flag==true)
      connection_id=(char)Input_buffer[0];
      send_bytes[11]=connection_id;                              //Adding
connection ID to the CIPSEND command
      Serial.print(send_bytes);
      clear_buffer();
      delay(1000);
      request_processed=true;                                    //Changing the
flag to true indicating that Client request is processed
   }
}


void brow_resp()                                                 //Sub routine to
respond to client request
{
  if(request_processed==true&&stringComplete==true)         //Checking the
flag on client request process
  {
    serial_check();
    if(Input_buffer[j-2]=='>')                                   //Checking for
the Send data signal from ESP  module
    {
      memset(Input_buffer,'\0', sizeof(Input_buffer));
      Serial.print("<!DOCTYPE html><html><body><h1>Welcome to Arduino LED
control Wizard</h1><p>You can turn ON/OFF LED connected to Arduino using the
buttons given below</p><p><b>TURN
ON</b>        <a
href=\"ON\"><button>ON</button></a></br></p><p><b>TURN
OFF</b>      <a
href=\"OFF\"><button>OFF</button></a></br></p></body></html>");
```

```
      delay(2000);

      serial_check();

      //while(command_response_check==false);

      clear_buffer();

      request_processed=false;

      client_request=true;

    }

    else

    {

      clear_buffer();

      request_processed=false; //If request response didnt turn out
successful mark the flags so that another CIPSEND command can be passed

      flag=false;

    }

  }

}



void command_input()                                          //Sub routine for
commands to create server using ESP module

{

  serial_check();

  if(command_response_check(Input_buffer)==true) //Checking the response and
Tracking the counts of command to handle in case of error response from ESP

  {command_count=command_count+1;}

  else

  delay(1000);
```

```
  clear_buffer();

  switch(command_count)                                    //Enter commands
sequentially in creating server

  {

    case 0: { Serial.print("AT\r\n");

              delay(500);

              break;

              }

    case 1: {

              Serial.print("AT+CWMODE=3\r\n");

              delay(1000);

              break;}

    case 2: {

              Serial.print("AT+CIPSTA=\"192.168.43.253\"\r\n");

              delay(1000);

              break;

              }

    case 3: {

              Serial.print("AT+CIPMUX=1\r\n");

              delay(1000);

              break;

            }

    case 4: {

              Serial.print("AT+CIPSERVER=1,80\r\n");

              delay(1000);
```

```
              break;
             }
    case 5: {
              command_flag=true;
              digitalWrite(13,HIGH);
              break;
             }
  }
}


void clear_buffer()                                                    //Clearing
buffer
{
  memset(Input_buffer, '\0', sizeof(Input_buffer));
  stringComplete=false;
}


boolean command_response_check(char *a)                                //Checking
for OK Response from ESP
{
  if(a[j-4]=='O'&&a[j-3]=='K'&&a[j-2]=='\r'&&a[j-1]=='\n')
  return true;
  else
  { delay(1000);
    return false; }
}
```

```
void serial_check() {                                          //Serial char
available check

  i=0;

  while (Serial.available()) {

    delay(2);

    char inChar = Serial.read();

    Input_buffer[i]=(char)inChar;

    i=i+1;

    stringComplete = true;

    j=i;

   }

}
```

NOTE:

1. The above code can only handle request from one client only, so you cannot use more than one client at a time , also ESP module is capable of handling 4 connections at time which leaves us with 2 client limitations all the time.
2. LED is here for just an indication you can do some cool stuffs like PWM, Timer, Data receiving and so but of course with tweaks in the code but concept remains the same.

LED Control via Thingspeak server using NodeMCU (ESP8266)

Published May 29, 2020

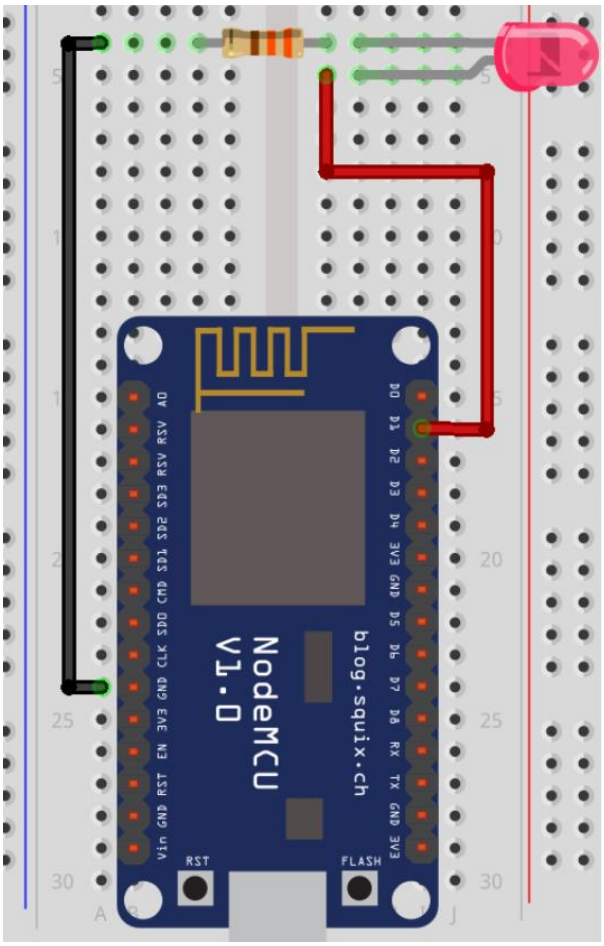
Apache-2.0

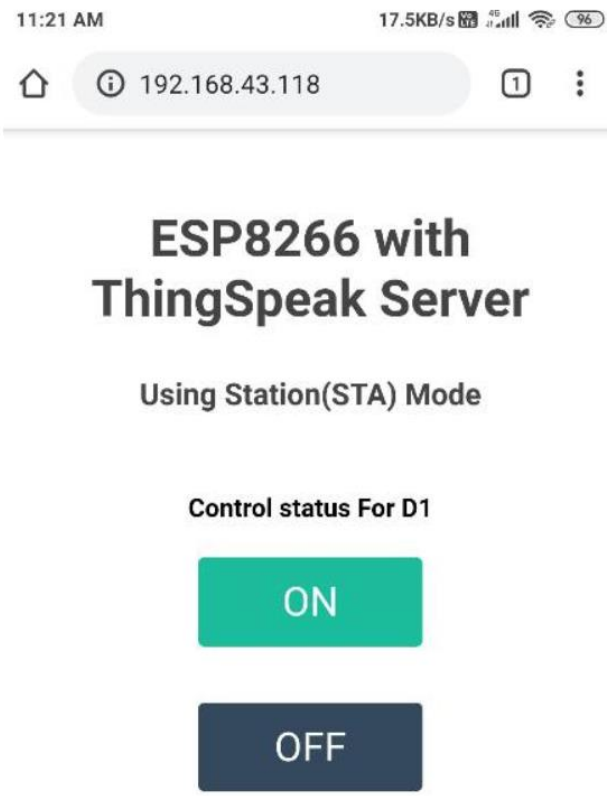
3 hours to build

Intermediate

nodemcu

In this project we can control the LED from anywhere in the world using Thingspeak server.





Powered By

MOUSER ELECTRONICS

Components Used

NodeMCU x 1
NodeMCUNodeMCU

Datasheet

Connecting Wire Jumper Wires x 1

Powered By

MOUSER ELECTRONICS

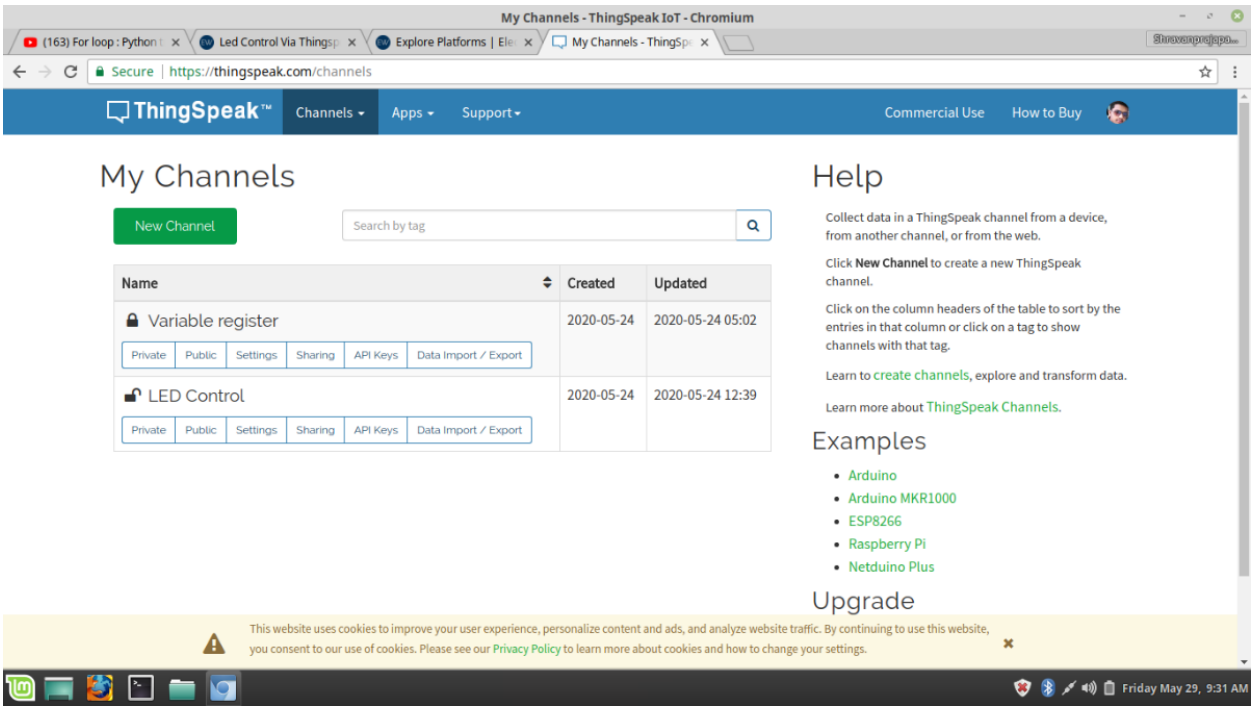
Components Used

Connecting Wire Breadboard wires		
LED 5mm LED 5mm	x 1	Datasheet

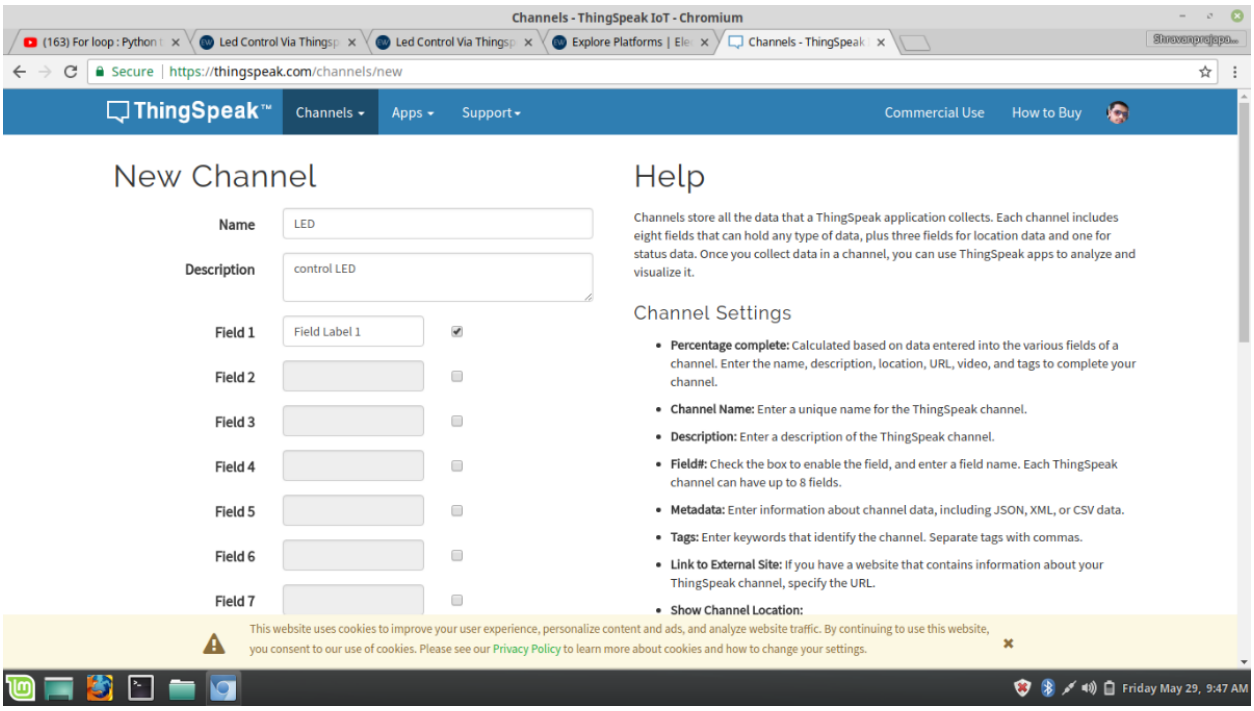
Description

Step1: Setting ThingsSpeak

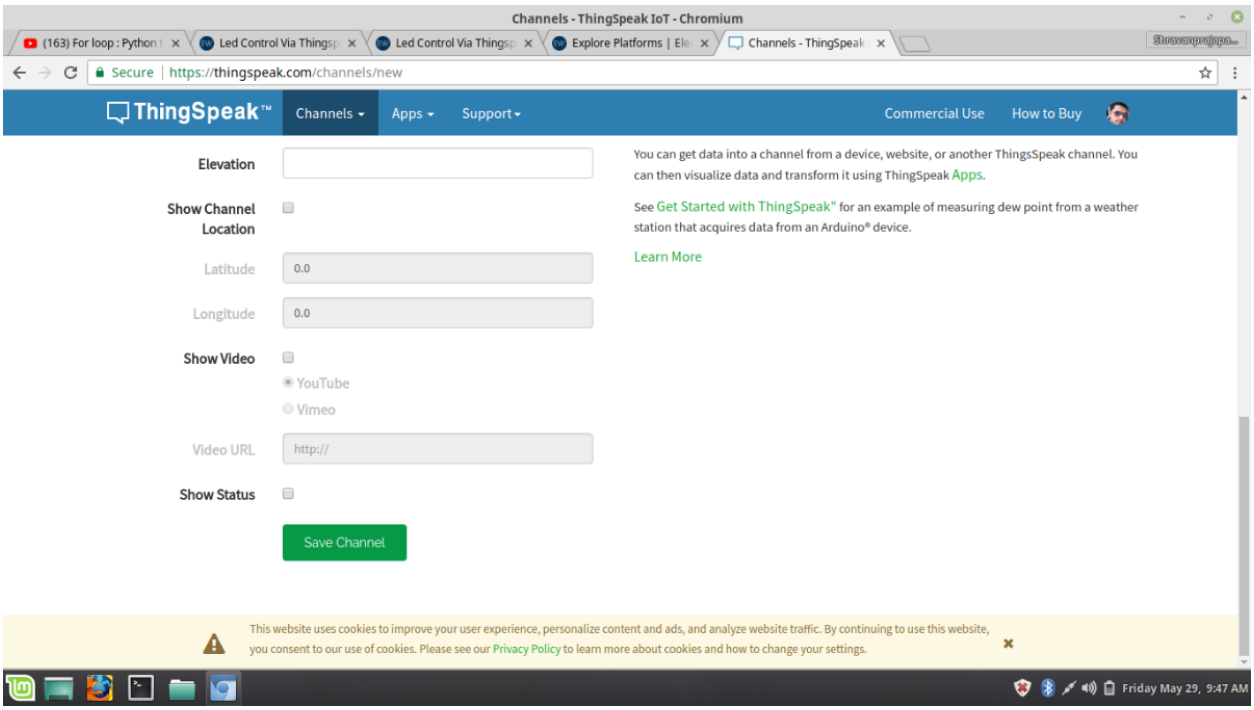
- First login to the Thingspeak server. https://thingspeak.com/login
- If you are new user then create the new account.
- after login you show this type of webpage, i have already created two project . but if you are new user then click on the **New Channel.**



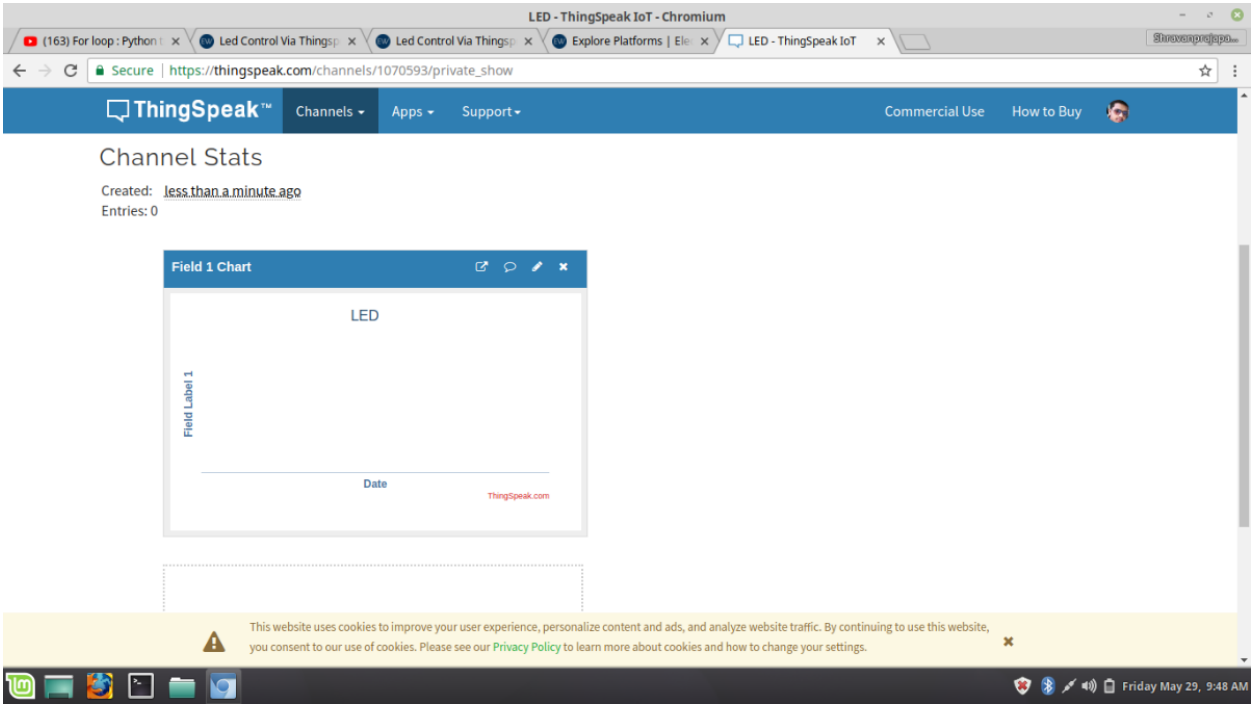
- After click the new channel and first write the name of the new channel .
- if you want give the description then write the description. and select the Field 1 and don't forget the click on checkbox, after click on the check box scroll the page and click on the save button.



- I give the new channel name is LED then select the Field 1 and scroll the page and click on the save channel button.



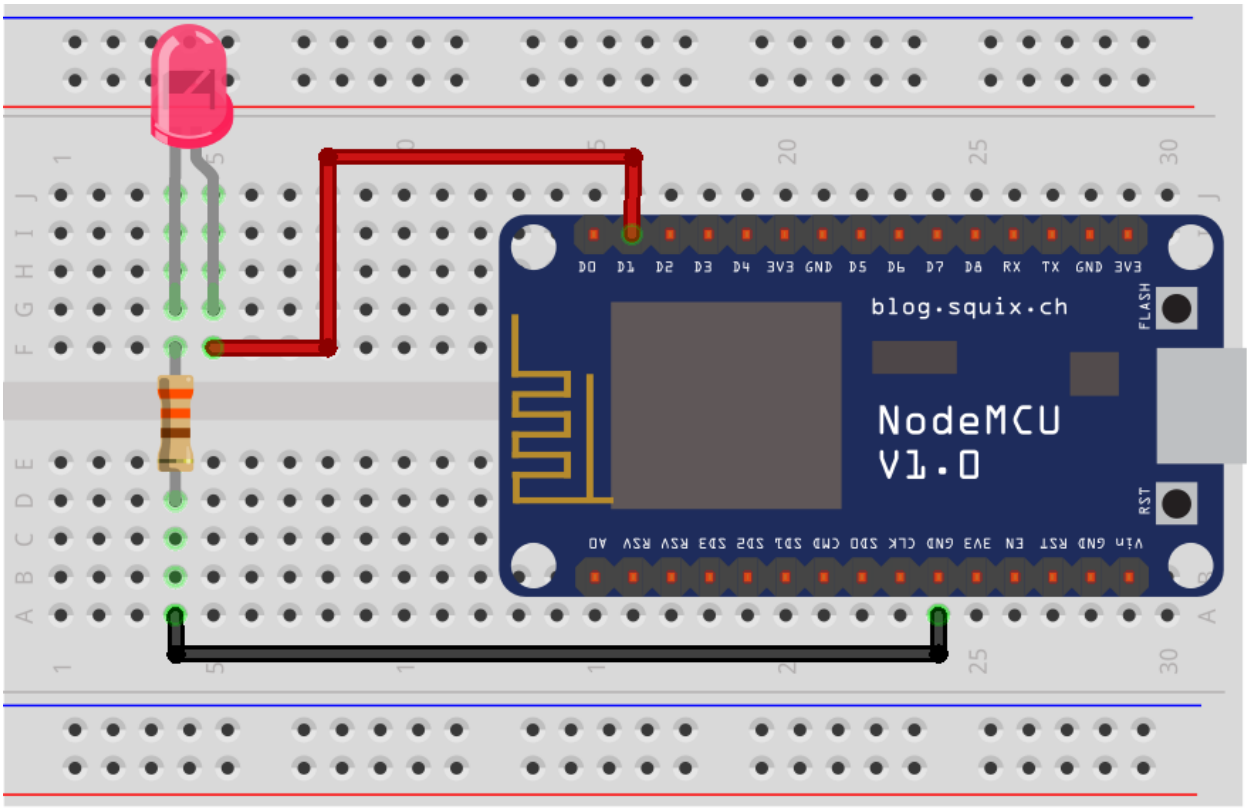
- Now show your field 1 as a charts.



- Here your filed is empty because you do not upload any data on the channel. Now our goal to blink the led via webserver.
- Before controlling the led via webpage you should some knowledge about the HTTP protocol if you don't have knowledge then don't worry read the block from here https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/
- This link give the most of the knowledge about the how to connect the Wi-Fi with esp8266 and also basic knowledge about the STA mode and Access point.

Hardware

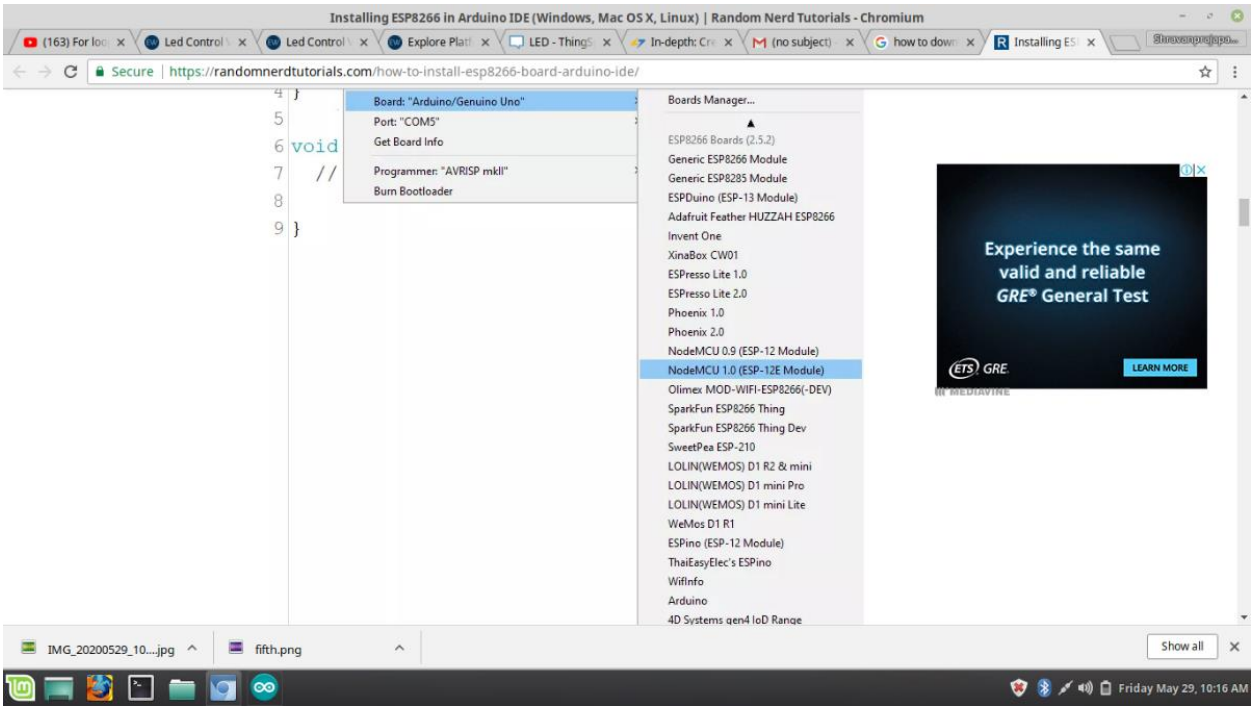
- Now we have to go to the hardware side , connect the LED with D1 pin with NodeMCU.



Firmware

- Now write the code in **Arduino IDE.** open the Arduino IDE and select the NodeMCU 12E board if you do not show this borad then refer to the link how to download this board.https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/

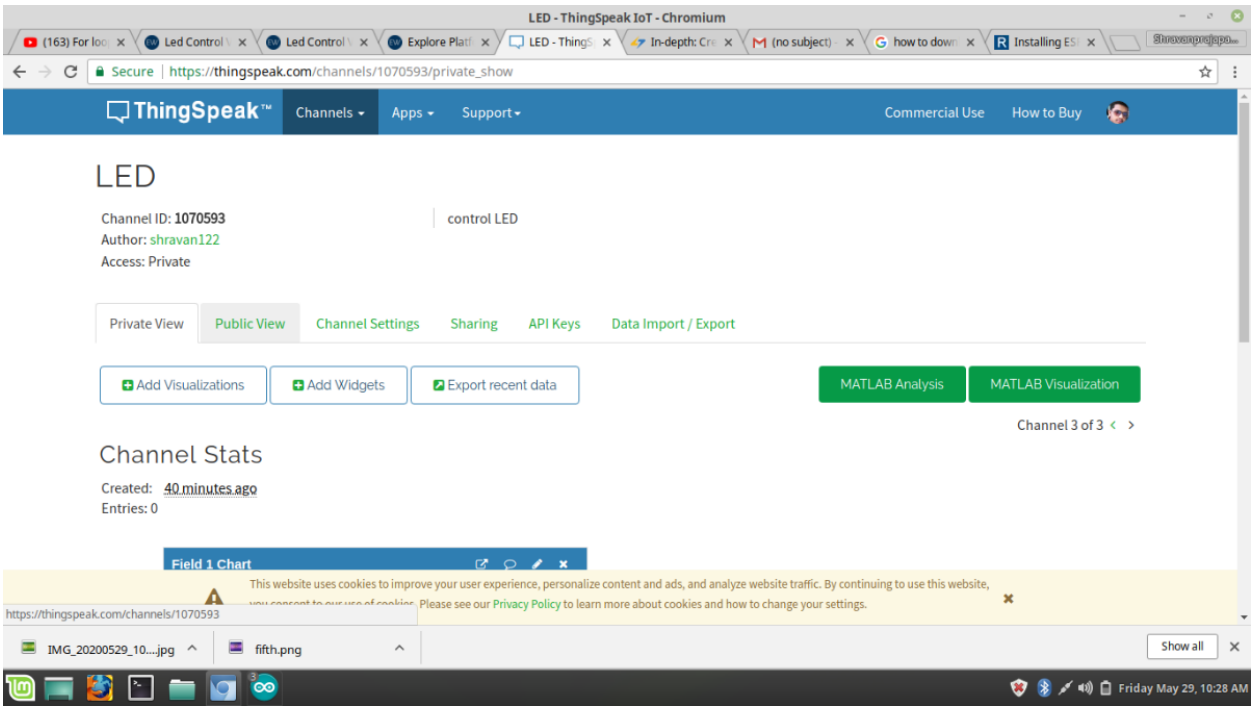
After installing click on the NodeMCU 1.0 borad.



- Now start the coding but before writing code we have to install one more library. go to the **sketch -> Include library** and click on **manage library.**
- Include the thingspeak library.
- Now start to writing the code.

```
#include<ThingSpeak.h>
#include<ESP8266WiFi.h>
#include<ESP8266WebServer.h>
```

- Show the Thingspeak server where we create the our new channel. and copy the channel id and pate to the our Arduino IDE.



- in this page **Channel ID: 1070593**

```
unsigned long channel_num=1070593;

const char* ssid="shravan";

const char* password="8905552134";

int led;

unsigned int value;

WiFiClient client;   // make the client of the WiFi which connect to the ThingSpeak webServer

ESP8266WebServer server(80);    // make the naother server
```

- To create the WiFi connection make the ssid, password.
- In your side ssid and password are different. Then create the object of the **WiFiClient** class is client. this object is used to connect the device to the Thingspeak sever.
- Then make the another server where we create out HTML page, using this HTML page we can send the data to Thingspeak server.

```
void setup()
```

```
{
 pinMode(D1,OUTPUT);
 digitalWrite(D1,0);
 Serial.begin(115200);
 WiFi.begin(ssid,password);    // connect to the wifi STA connection
 while(WiFi.status()!=WL_CONNECTED)
 {
   delay(500);
   Serial.print(".");
 }
 Serial.println(WiFi.localIP());    // print the wifi local ip
 server.on("/",handleonconnect);    // in urt type the "/" then call the handle on connect function
 ThingSpeak.begin(client);      // connect the client to the thingSpeak server
 server.begin();   // start the server
}
```

- Make the D1 as an OUTPUT pin and initially LOW the LED. then select the baudrate 115200 and and connect the ESP8266 with wifi to the station mode. and print the local ip on the serial port. using **WiFi.localIP() .**
- There are many method to connect the esp8266 with server here we use the **server.on(url,function)** method. here '/ ' means root path when we hit on the web browser. then client send the request. on server.
- Now connect the client to the thingSpeak server using **ThingSpeak.begin(client).**
- start the server where we show the HTML page.

void loop()

```
{
 server.handleClient();    // it realy handle the Client
 led=ThingSpeak.readFloatField(channel_num,1);    // rad the last data of the field 1
```

```
 if(led==1)
 {
   digitalWrite(D1,1);
 }
 else if(led==0)
 {
   digitalWrite(D1,0);
 }
}
```

- In void loop() function, server.handleclient(); it is actually handle the client.
- ThingSpeak.readFloatField (channel_id , 1); this function return the int value where we pass on the HTML page link. here 1 mean Field number. upper we create the only field1.
- if this function return the 1 then we on the led other wise low.

```
void handleonconnect()
{
 server.send(200,"text/html",SendHTML());
}
```

- Here handleonconnect function where we pass the server.on("/",handleonconnect).
- Here we use the **server.send(200,"text/html",SendHTML())** , 200 means ok response, content type is html and SendHTML() is function but we put the argument string so this function return the string this show you next quotes.

HTML

```
String SendHTML(void){
 String ptr = "<!DOCTYPE html> <html>\n";
```

```
 ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";

 ptr +="<title>LED Control</title>\n";

 ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";

 ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";

 ptr +=".button {display: block;width: 80px;background-color: #1abc9c;border: none;color: white;padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor: pointer;border-radius: 4px;}\n";

 ptr +=".button-on {background-color: #1abc9c;}\n";

 ptr +=".button-on:active {background-color: #16a085;}\n";

 ptr +=".button-off {background-color: #34495e;}\n";

 ptr +=".button-off:active {background-color: #2c3e50;}\n";

 ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";

 ptr +="</style>\n";

 ptr +="</head>\n";

 ptr +="<body>\n";

 ptr +="<h1>ESP8266 with ThingSpeak Server</h1>\n";

 ptr +="<h3>Using Station(STA) Mode</h3>\n";

 ptr +="<h4>Control status For D1</h4>\n";

 ptr +="<a class=\"button button-on\" href=\"https://api.thingspeak.com/update?api_key=MOHD33LYGVXTG5UF&field1=1\">ON</a>\n";

 ptr +="<a class=\"button button-off\" href=\"https://api.thingspeak.com/update?api_key=MOHD33LYGVXTG5UF&field1=0\">OFF</a>\n";

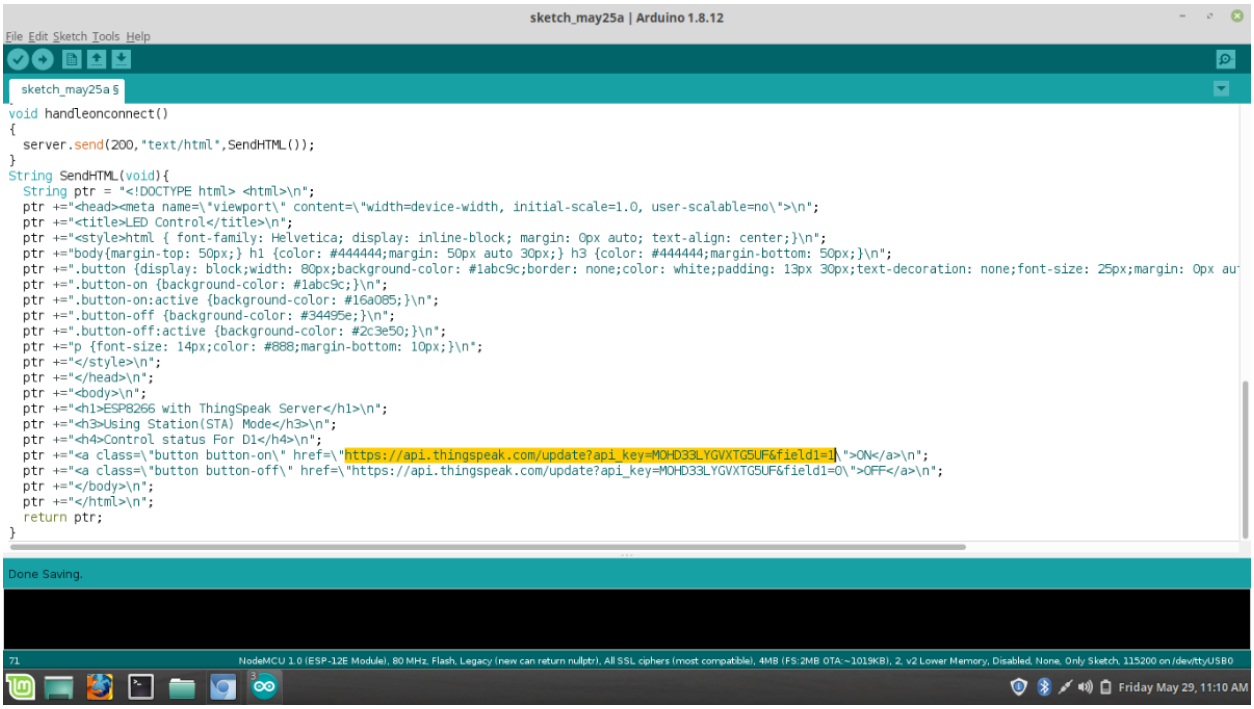
 ptr +="</body>\n";

 ptr +="</html>\n";

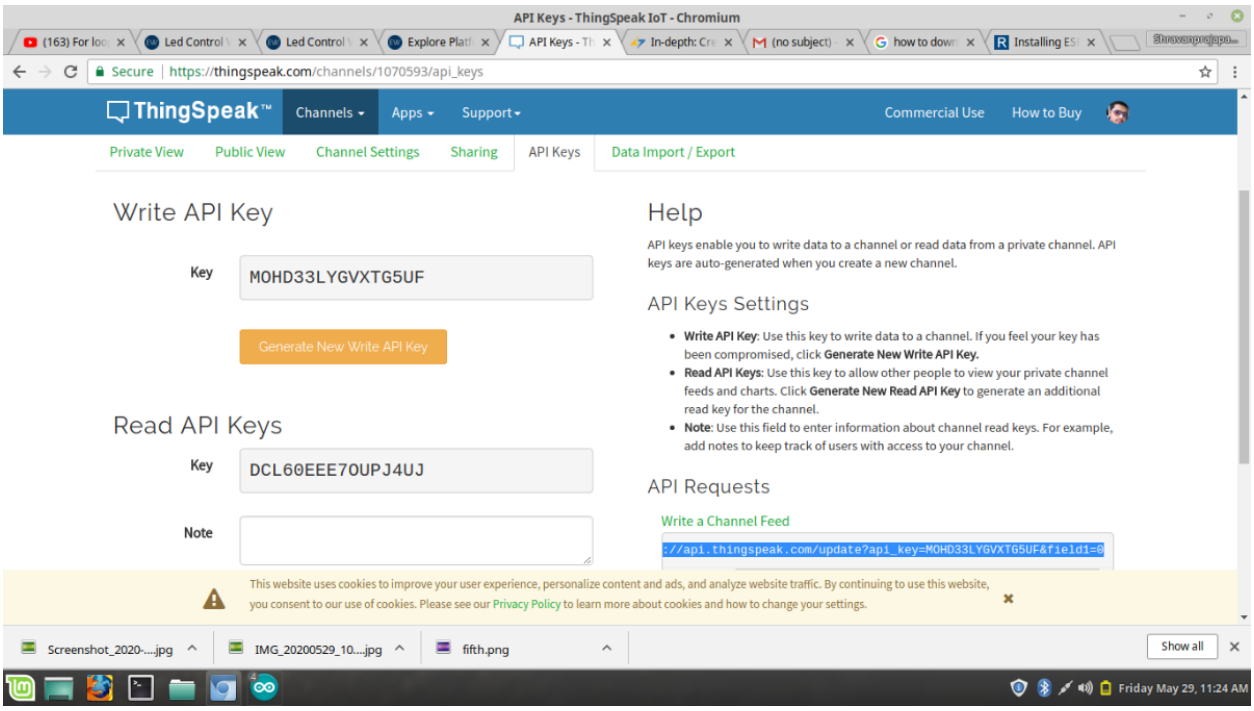
 return ptr;

}
```

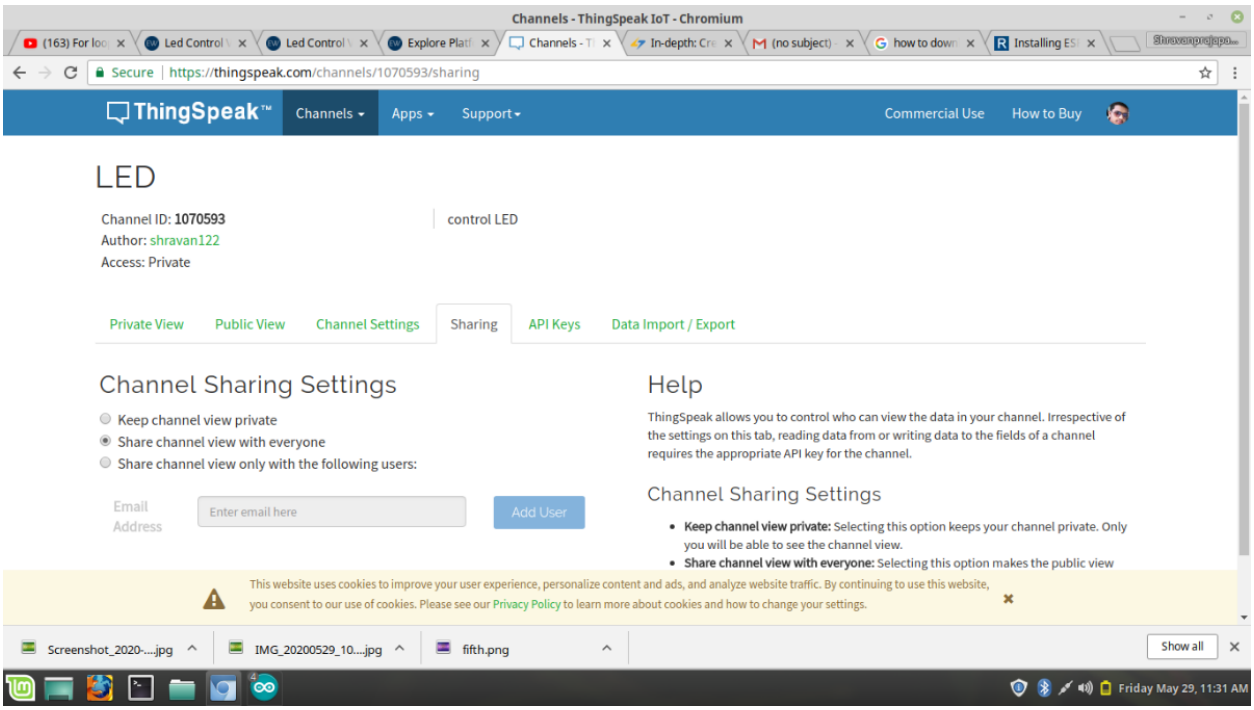
- That is my HTML code if you are an embedded engineer then do not learn the HTML and CSS, but knowledge abut the how it work we press the any url.
- Here we highlighted on the HTML string so this is the **main thing**.



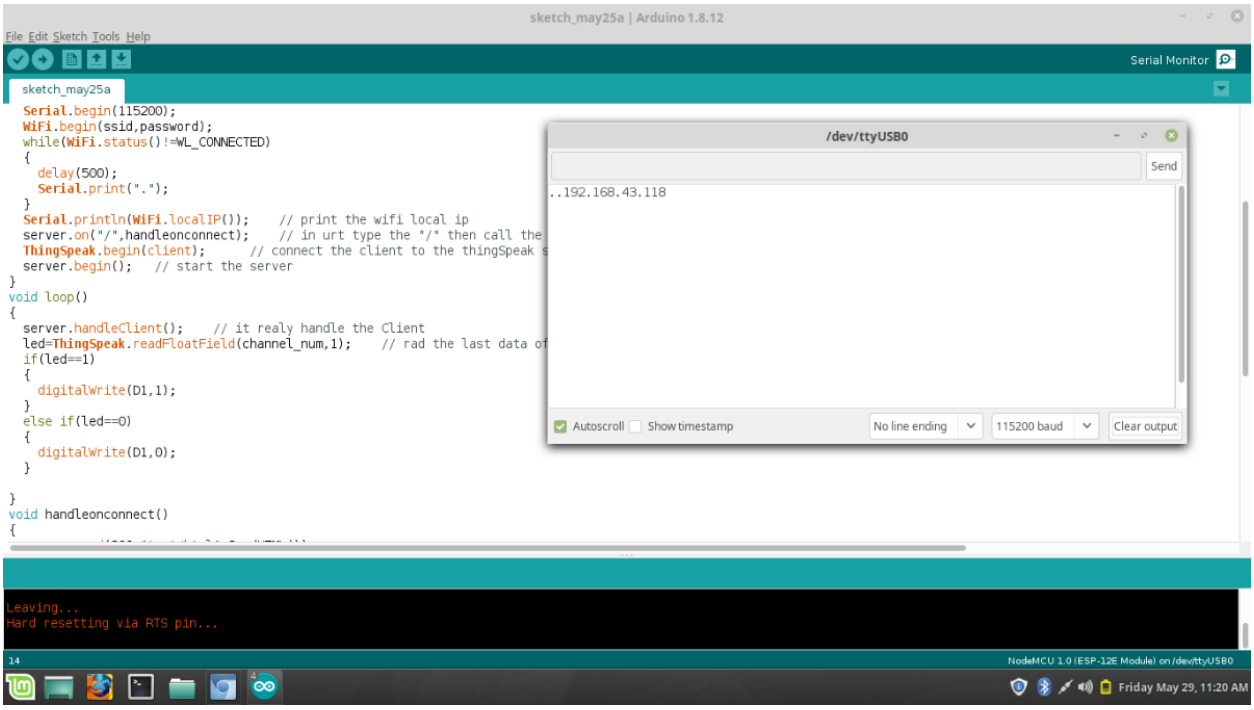
- **https://api.thingspeak.com/update?api_key=MOHD33LYGVXTG5UF&field1=0** this url is copy from the thingserver here api_key is different in your case last one &field1=0 mean we we press this url then send the 0 on your thingspeak server to the field1 and &field1=1 mean when we press this url then send the 1 on your thingspeak server to the field1. but we don't press the url using HTML code we only clink on the ON button to turn on the led and OFF button for turn off the led.
- Now go to the thingspeak server click on the **API KEYS** and copy the **Write API KEY** and paste the HTML code code which highlighted on the upper HTML code pic.
- **I again says that api write key is the different in your case.**



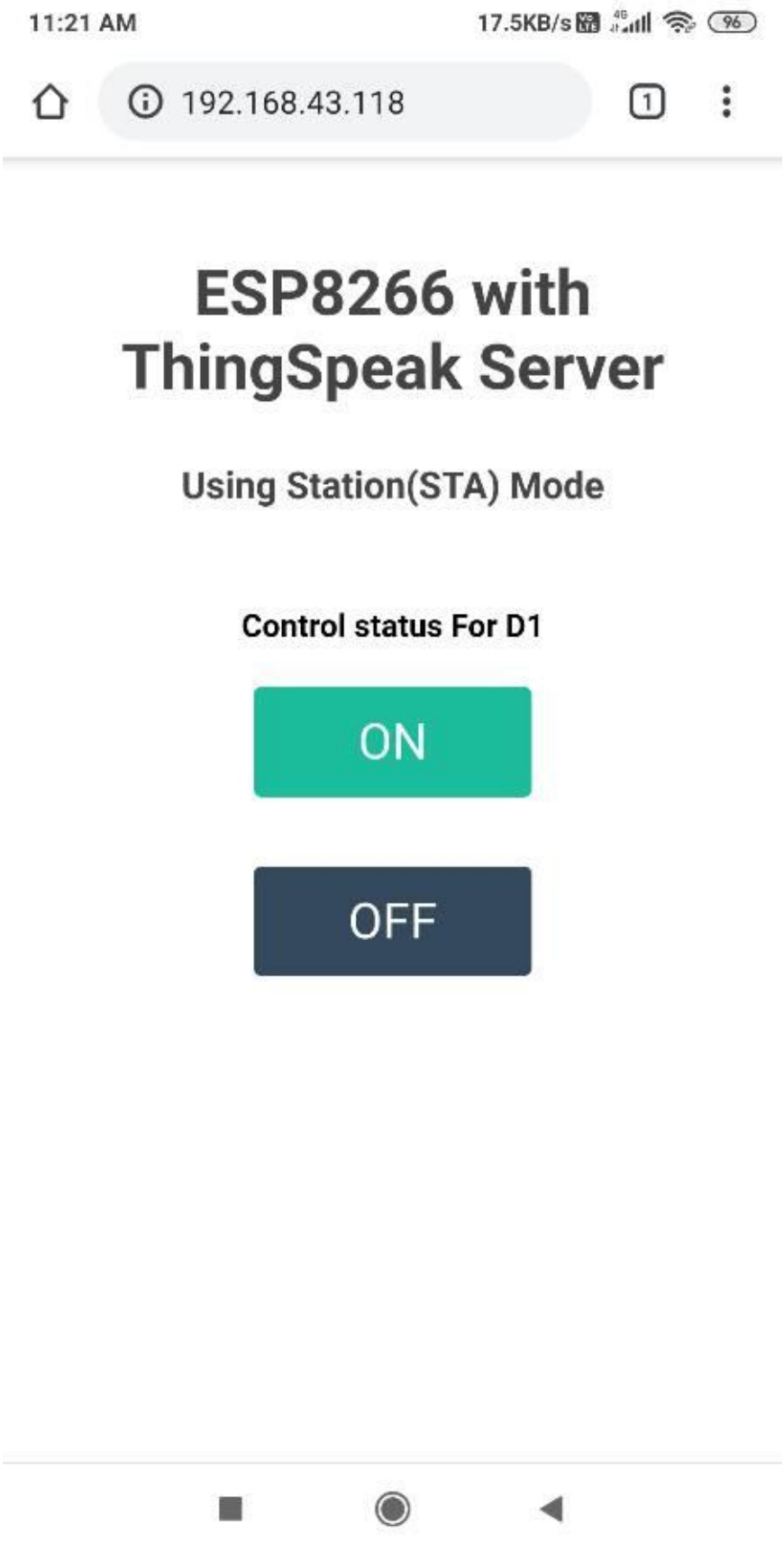
- one more thing we have to sharing in public otherwise we can not send the data from the server.
- so for that click on **sharing button** and click the **Share channel view with everyone.**



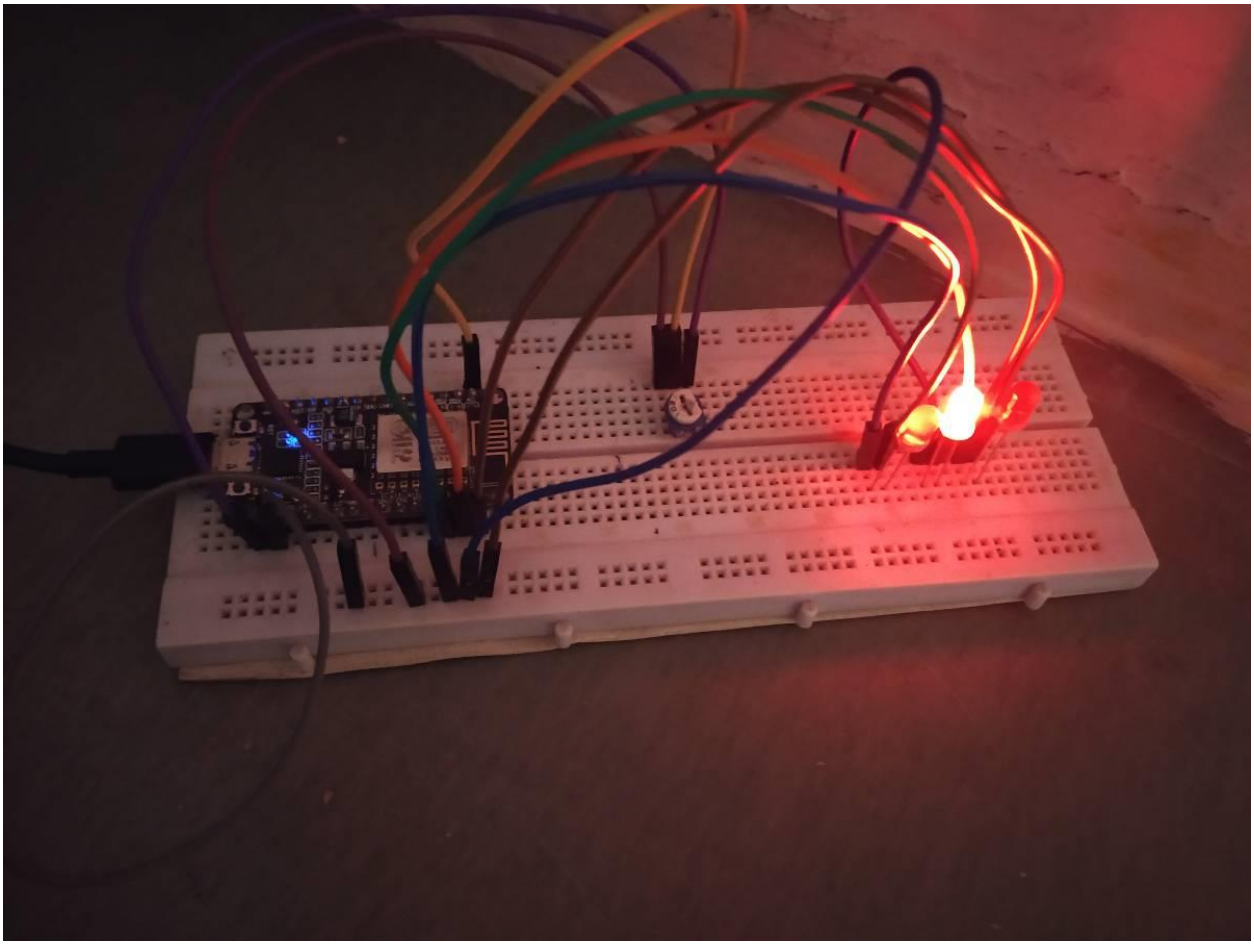
- Finally upload the code on the NodeMCU. show the serial port wifi is connect or not if connect then give the local ip where we print on serial.



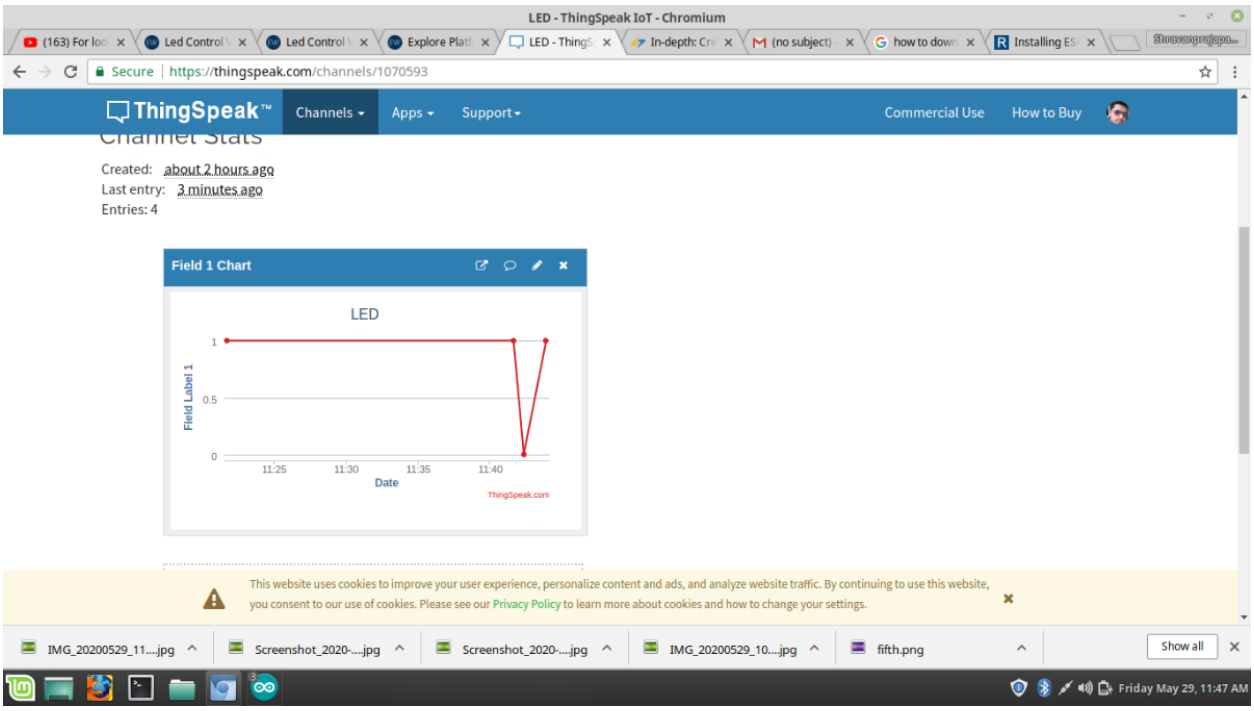
- Finally this IP copy on the Chrome browser. then show the this type.


11:21 AM
17.5KB/s
192.168.43.118
ESP8266 with
ThingSpeak Server
Using Station(STA) Mode
Control status For D1
ON
OFF

- Now click on the **ON button** LED is ON .
- Now click on the **OFF button** LED is OFF.



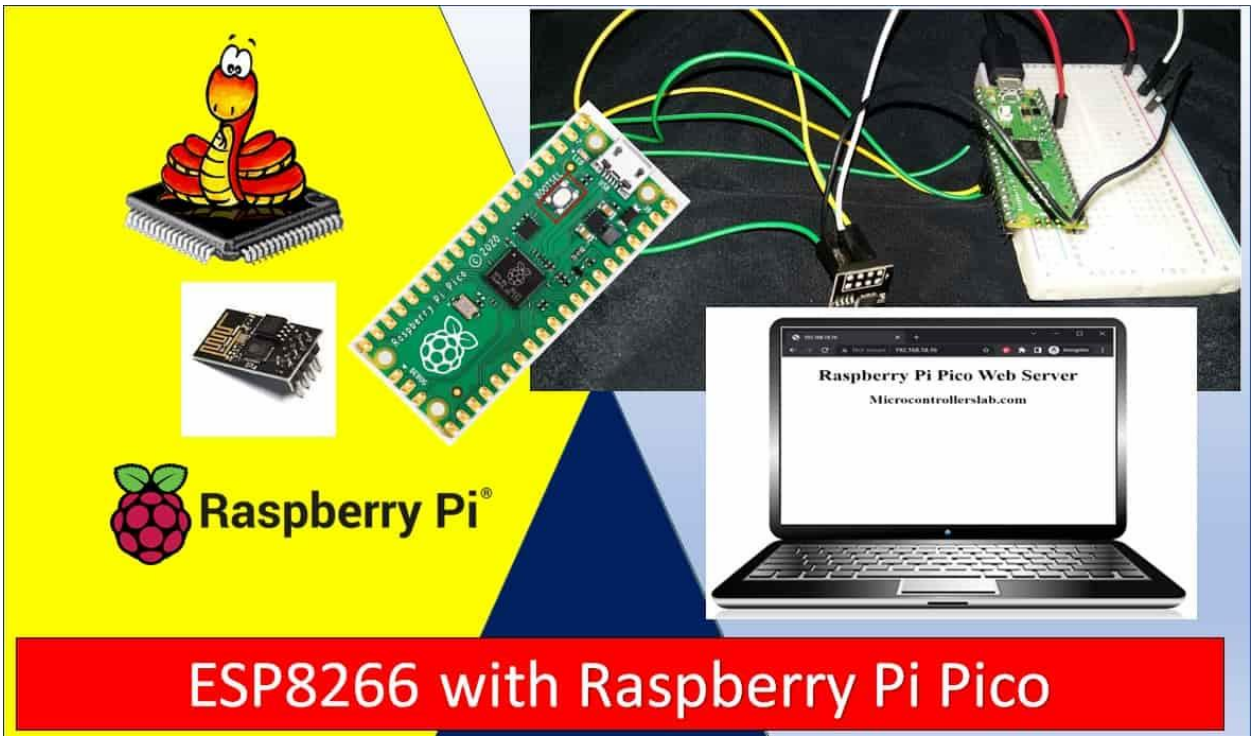
- So finally LED is ON.
- Now show the status of the thingSpeak server.



- using this you can control anything from anywhere in the world using this WebServer.
- If you query about this project then contact me using this mail- **shravanprajapati122@gmail.com**

Interface ESP8266 WiFi Module with Raspberry Pi Pico

In this tutorial, you will learn how to interface ESP8266-01 WiFi module with Rasperry Pi Pico to get internet connectivity. We will create a TCP Web Server with Raspberry Pi Pico by using AT commands of the ESP-01 module and use the serial port of the Pico board to configure the ESP-01 WiFi module to connect with WiFi and to create a web server. We will use Thonny IDE to program Raspberry Pi Pico with ESP-01 in MircoPython.



Raspberry Pi Pico is a low-cost, high-performance board based on the Raspberry Pi RP2040 microcontroller chip. But, it does not support WiFi capabilities hence we have to use an external WiFi module to enable WiFi connectivity for Raspberry Pi Pico.

ESP8266 WiFi Module

In August 2014 Espressif Systems launched their first raw module which is manufactured by a third part AI-Thinker and module referred to as ESP-01 module.

ESP8266 delivers a highly integrated WiFi solution that meets the needs of the Internet of Things industries such as low cost, efficient power usage, trustworthy performance, and compact design.

The ESP8266 WIFI module is basically a complete WiFi solution, which has self-contained integrated TCP/IP protocol stack that can be easily connected to the microcontroller for gaining access to any WiFi network.

We will use send AT commands to ESP8266 module over UART from Raspberry Pi Pico to configure the TCP Web server.

There are many ESP8266 WiFi modules available in the market ranging from ESP-01 to ESP-12. But in this tutorial, we are using ESP-01. AT commands are the same for all these ESP modules.

ESP-01 Module pinout

The ESP8266-01 WiFi module consists of two rows of eight pins. The pin configuration diagram is shown in the figure below:

It has a total of 8 pins of which 6 pins are active.

Label	Description
3.3V	Supply 3.3 volts pin
GND	Ground pin
RST	Reset Pin
CH_PD/EN	Chip Power and Enable pin
GPIO 0 to 3	UART interface and input/output pins

- Pin_1, which is a GND pin, is directly connected to the ground for power on this module.
- Pins_2 and 3, which are the GPIO 2 and GPIO 0, these pins decide in which mode the module would be a start-up, in other words, these are mode selected pins.
- Pins_4 and 5, which are RX and TX, these pins are used for communication purposes and program the module.
- Pin_6 which is CH_PD, it is called chip power-down pin.
- Pin_7 which is the RST pin and this pin is used to reset module.
- Pin_8 is a VCC pin that is used to power the module. The operating voltage of ESP-01 is 3.3 volts.

Interfacing ESP-01 Wi-Fi Module with Raspberry Pi Pico

We will require the following components:

- Raspberry Pi Pico
- ESP8266 ESP-01 Module
- Connecting Wires
- Breadboard (not necessary)

As we have seen above the ESP-01 module consists of 8 pins. However, we will use 5 pins to connect it with the Raspberry Pi Pico board. These include the VCC, EN, GND, RX and TX pins. RX and TX pins of the module will be connected with the UART pins of the Pi Pico board. Let us first have a look at the Raspberry Pi Pi UART Pins.

Raspberry Pi Pico UART Pins

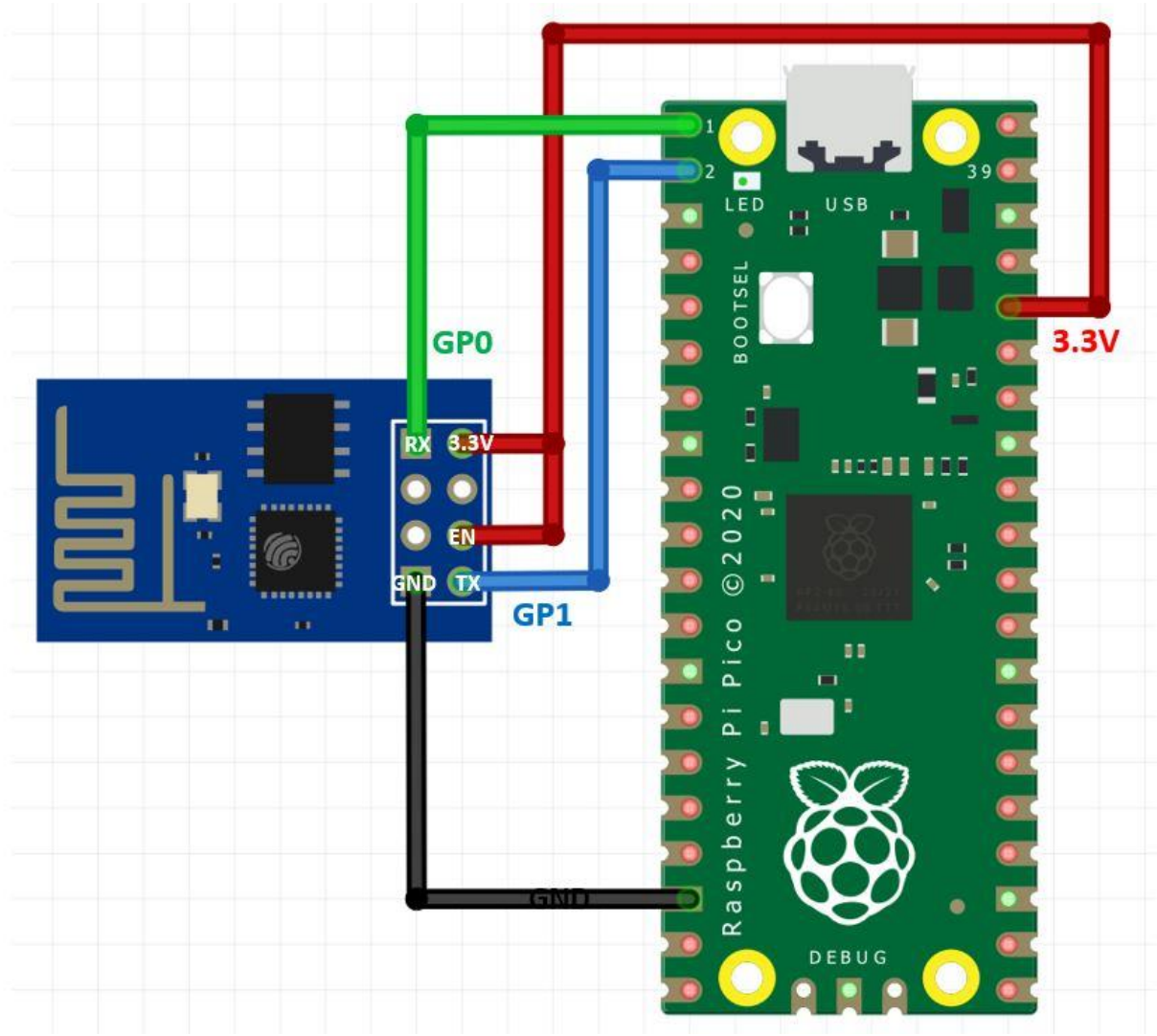
Raspberry Pi Pico contains two identical UART peripherals with separate 32×8 Tx and 32×12 Rx FIFOs.

The following table lists the GPIO pins for both UART peripherals which are exposed on Raspberry Pi Pico development board pinouts.

UART Pins	GPIO Pins
UART0-TX	GP0/GP12/GP16
UART0-RX	GP1/GP13/GP17
UART1-TX	GP4/GP8
UART1-RX	GP5/GP9

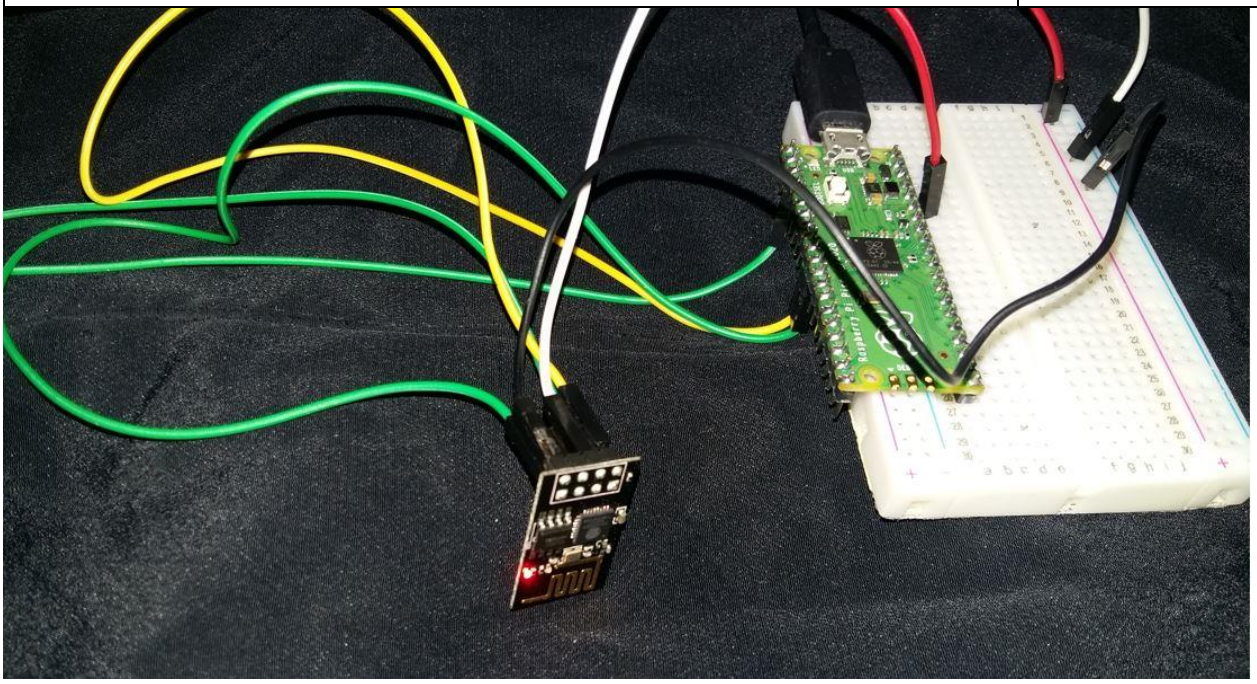
Connection Diagram

For this tutorial, we will use UART0 TX and RX pins of Raspberry Pi Pico. Follow the connection diagram below to connect the two devices.



Raspberry Pi Pico	ESP-01
3.3V	VCC
3.3V	EN
GND	GND
GP1 (UART0 RX)	TX

Raspberry Pi Pico	ESP-01
GP0 (UART0 TX)	RX



Raspberry Pi Pico Web Server MicroPython Script

The following MicriPython script of Raspberry Pi Pico sends AT commands to the ESP8266 module to configure ESP8266 as a TCP web server. We will see details of all AT commands in the next sections.

```
import uos
import machine
import utime

recv_buf="" # receive buffer global variable

print()
print("Machine: \t" + uos.uname()[4])
print("MicroPython: \t" + uos.uname()[3])

uart0 = machine.UART(0, baudrate=115200)
print(uart0)
```

```
def Rx_ESP_Data():
    recv=bytes()
    while uart0.any()>0:
        recv+=uart0.read(1)
    res=recv.decode('utf-8')
    return res

def Connect_WiFi(cmd, uart=uart0, timeout=3000):
    print("CMD: " + cmd)
    uart.write(cmd)
    utime.sleep(7.0)
    Wait_ESP_Rsp(uart, timeout)
    print()

def Send_AT_Cmd(cmd, uart=uart0, timeout=3000):
    print("CMD: " + cmd)
    uart.write(cmd)
    Wait_ESP_Rsp(uart, timeout)
    print()

def Wait_ESP_Rsp(uart=uart0, timeout=3000):
    prvMills = utime.ticks_ms()
    resp = b""
    while (utime.ticks_ms()-prvMills)<timeout:
        if uart.any():
            resp = b"".join([resp, uart.read(1)])
    print("resp:")
    try:
        print(resp.decode())
    except UnicodeError:
        print(resp)

Send_AT_Cmd('AT\r\n')          #Test AT startup
Send_AT_Cmd('AT+GMR\r\n')      #Check version information
Send_AT_Cmd('AT+CIPSERVER=0\r\n')      #Check version information
Send_AT_Cmd('AT+RST\r\n')      #Check version information
Send_AT_Cmd('AT+RESTORE\r\n')  #Restore Factory Default Settings
Send_AT_Cmd('AT+CWMODE?\r\n')  #Query the WiFi mode
Send_AT_Cmd('AT+CWMODE=1\r\n') #Set the WiFi mode = Station mode
Send_AT_Cmd('AT+CWMODE?\r\n')  #Query the WiFi mode again
#Send_AT_Cmd('AT+CWLAP\r\n', timeout=10000) #List available APs
Connect_WiFi('AT+CWJAP="HUAWEI-u67E","4uF77R2n"\r\n', timeout=5000)
#Connect to AP
Send_AT_Cmd('AT+CIFSR\r\n')    #Obtain the Local IP Address
utime.sleep(3.0)
Send_AT_Cmd('AT+CIPMUX=1\r\n')    #Obtain the Local IP Address
```

```
utime.sleep(1.0)
Send_AT_Cmd('AT+CIPSERVER=1,80\r\n')    #Obtain the Local IP Address
utime.sleep(1.0)
print ('Starting connection to ESP8266...')
while True:
    res =""
    res=Rx_ESP_Data()
    utime.sleep(2.0)
    if '+IPD' in res: # if the buffer contains IPD(a connection), then
respond with HTML handshake
        id_index = res.find('+IPD')
        print("resp:")
        print(res)
        connection_id =  res[id_index+5]
        print("connectionId:" + connection_id)
        print ('! Incoming connection - sending webpage')
        uart0.write('AT+CIPSEND='+connection_id+',200'+'\r\n')  #Send
a HTTP response then a webpage as bytes the 108 is the amount of bytes
you are sending, change this if you change the data sent below
        utime.sleep(1.0)
        uart0.write('HTTP/1.1 200 OK'+'\r\n')
        uart0.write('Content-Type: text/html'+'\r\n')
        uart0.write('Connection: close'+'\r\n')
        uart0.write(''+'\r\n')
        uart0.write('<!DOCTYPE HTML>'+'\r\n')
        uart0.write('<html>'+'\r\n')
        uart0.write('<body><center><h1>Raspberry Pi Pico Web
Server</h1></center>'+'\r\n')

uart0.write('<center><h2>Microcontrollerslab.com</h2></center>'+'\r\n'
)
        uart0.write('</body></html>'+'\r\n')
        utime.sleep(4.0)
        Send_AT_Cmd('AT+CIPCLOSE='+ connection_id+'\r\n') # once file
sent, close connection
        utime.sleep(2.0)
        recv_buf="" #reset buffer
        print ('Waiting For connection...')
```

How does the Code Works?

We will start by importing the machine module and the uos module. Next, we will also import utime module to incorporate delays.

```
import uos
```

```
import machine
import utime
```
Then, we will print the information about our current operating system in the Thonny shell terminal. We will uos.uname() and print the operating system version and release.

```
print()
print("Machine: \t" + uos.uname()[4])
print("MicroPython: \t" + uos.uname()[3])
```
Initialize UART Communication

Then we will create an uart object by using UART() and specify the UART channel as the first parameter and the baud rate as the second parameter. We are using UART0 in this case with baud rate 115200 for the uart communication. ESP8266 has a default baud rate of 115200 hence we will use the same baud rate here for Raspberry Pi Pico UART communication in order to create synchronization. Moreover we will also print the UART details in the shell terminal.

```
uart0 = machine.UART(0, baudrate=115200)
print(uart0)
```
This Connect_WiFi() function is used to connect ESP8266 with WiFi.

```
def Connect_WiFi(cmd, uart=uart0, timeout=3000):
  print("CMD: " + cmd)
  uart.write(cmd)
  utime.sleep(7.0)
  Wait_ESP_Rsp(uart, timeout)
  print()
```
Next, we will define three functions. The first one is Rx_ESP_Data(). This reads the serial data being received. This data is decoded from UTF-8 format and returned.

```
def Rx_ESP_Data():
    recv=bytes()
    while uart0.any()>0:
        recv+=uart0.read(1)
    res=recv.decode('utf-8')
    return res
```

The second function is Send_AT_Cmd(cmd, uart=uart0, timeout=3000). It takes in three parameters, the AT command, the UART channel and the response time. This function will be used to it send an AT command to ESP8266 via uart0. The response time is set to 3 seconds.

```
def Send_AT_Cmd(cmd, uart=uart0, timeout=3000):
    print("CMD: " + cmd)
    uart.write(cmd)
    Wait_ESP_Rsp(uart, timeout)
    print()
```

The Wait_ESP_Rsp(uart=uart0, timeout=3000) function waits for 3 seconds to get the response from ESP8266. After receiving the data from ESP8266 it concatenates the received bytes and prints them on the shell terminal.

```
def Wait_ESP_Rsp(uart=uart0, timeout=3000):
    prvMills = utime.ticks_ms()
    resp = b""
    while (utime.ticks_ms()-prvMills)<timeout:
        if uart.any():
            resp = b"".join([resp, uart.read(1)])
    print("resp:")
    try:
        print(resp.decode())
    except UnicodeError:
        print(resp)
```

AT Commands

Now let us look at the series of AT commands that we will send through UART0 to ESP8266.

```
Send_AT_Cmd('AT\r\n')           #Test AT startup
Send_AT_Cmd('AT+GMR\r\n')       #Check version information
Send_AT_Cmd('AT+CIPSERVER=0\r\n')
Send_AT_Cmd('AT+RST\r\n')       #Check version information
Send_AT_Cmd('AT+RESTORE\r\n')   #Restore Factory Default Settings
Send_AT_Cmd('AT+CWMODE?\r\n')   #Query the WiFi mode
Send_AT_Cmd('AT+CWMODE=1\r\n') #Set the WiFi mode = Station mode
Send_AT_Cmd('AT+CWMODE?\r\n')   #Query the WiFi mode again
Send_AT_Cmd('AT+CWJAP="HUAWEI-u67E","4uF77R2n"\r\n', timeout=5000)
#Connect to AP
```

```
utime.sleep(3.0)
Send_AT_Cmd('AT+CIFSR\r\n')    #Obtain the Local IP Address
utime.sleep(3.0)
Send_AT_Cmd('AT+CIPMUX=1\r\n')
utime.sleep(1.0)
Send_AT_Cmd('AT+CIPSERVER=1,80\r\n')    #Obtain the Local IP Address
utime.sleep(1.0)
```

AT: This type of command is used to test the startup function of WiFi module. The response would be ok, against this command if everything is ok.

```
Send_AT_Cmd('AT\r\n')          #Test AT startup
```

AT+GMR : This type of AT command is used to check the version of AT command and we used SDK version of AT command in this type of WIFI module.

```
Send_AT_Cmd('AT+GMR\r\n')      #Check version information
```

AT+CIPSERVER=0: This configures the ESP8266 as server and sets the mode as 0 which means delete server (need to follow by restart)

```
Send_AT_Cmd('AT+CIPSERVER=0\r\n')
```

AT+RST: This type of command is used for reset the WiFi module when it is in working condition. The response would be ok, when reset the module.

```
Send_AT_Cmd('AT+RST\r\n')
```

AT+RESTORE: This type of command is used to restore factory settings means, when this command is entered then all the parameters are reset automatically to default one's.

```
Send_AT_Cmd('AT+RESTORE\r\n')  #Restore Factory Default Settings
```

AT+CWMODE? : This type of command is used to query the WiFi mode of ESP8266.

```
Send_AT_Cmd('AT+CWMODE?\r\n')  #Query the WiFi mode
```

AT+CWMODE=1 : This sets the WiFi mode of ESP8266 in this case in station mode.

```
Send_AT_Cmd('AT+CWMODE=1\r\n') #Set the WiFi mode = Station mode
```

AT+CWJAP="SSID","PASSWORD"\r\n', timeout=TIME_ms : This connects the ESP8266 with an AP whose SSID and password are given, The timeout here is the reconnection time.

```
Connect_WiFi('AT+CWJAP="HUAWEI-u67E","4uF77R2n"\r\n', timeout=5000)
#Connect to AP
```

AT+CIFSR: This command obtains the local IP address.

```
Send_AT_Cmd('AT+CIFSR\r\n')
```

AT+CIPMUX=1:This command is used to enable multiple connections (maximum 4)

```
Send_AT_Cmd('AT+CIPMUX=1\r\n')
```

AT+CIPSERVER=1: This command configures ESP8266 as server.

```
Send_AT_Cmd('AT+CIPSERVER=1,80\r\n')
```

while loop

Inside the while loop, we will first call Rx_ESP_Data() which returns the data that the ESP8266 receives. This is saved in the variable 'res.' Now after a delay of 2 seconds, we will check if the buffer contains an IPD connection or not. If it does then, respond with an HTML handshake.

Print the response in the shell terminal.

```
res =""
res=Rx_ESP_Data()
print("resp:")
print(res)
```

Obtain the connection ID and print it.

```
id_index = res.find('+IPD')
connection_id =  res[id_index+5]
print("connectionId:" + connection_id)
```

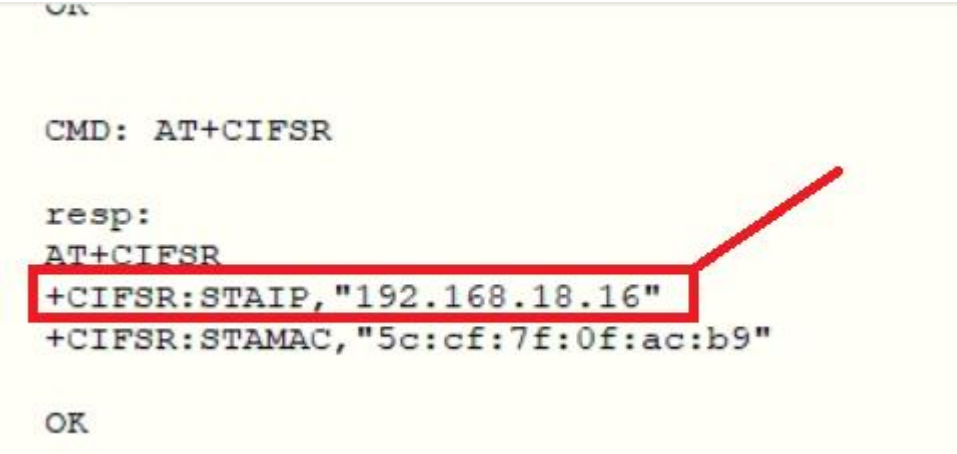
Then by using the uart object on the write() method, we will send the bytes to the UART. First, we are writing the AT command: AT+CIPSEND='ID', 'LENGTH' This will set the length of the data that will be sent. Next after a delay of 1 second, we will write the HTML body that will build the web page to the serial port. After that, we will close the multiple connections as we are sending the AT command: **AT+CIPCLOSE='ID'**. Then we will reset the buffer and wait for the connection.

```
print ('! Incoming connection - sending webpage')
        uart0.write('AT+CIPSEND='+connection_id+',200'+'\r\n')  #Send
a HTTP response then a webpage as bytes the 108 is the amount of bytes
you are sending, change this if you change the data sent below
        utime.sleep(1.0)
        uart0.write('HTTP/1.1 200 OK'+'\r\n')
        uart0.write('Content-Type: text/html'+'\r\n')
        uart0.write('Connection: close'+'\r\n')
        uart0.write(''+'\r\n')
        uart0.write('<!DOCTYPE HTML>'+'\r\n')
        uart0.write('<html>'+'\r\n')
        uart0.write('<body><center><h1>Raspberry Pi Pico Web
Server</h1></center>'+'\r\n')
```

```
uart0.write('<center><h2>Microcontrollerslab.com</h2></center>'+'\r\n'
)
        uart0.write('</body></html>'+'\r\n')
        utime.sleep(4.0)
        Send_AT_Cmd('AT+CIPCLOSE='+ connection_id+'\r\n') # once file
sent, close connection
        utime.sleep(2.0)
        recv_buf="" #reset buffer
        print ('Waiting For connection...')
```

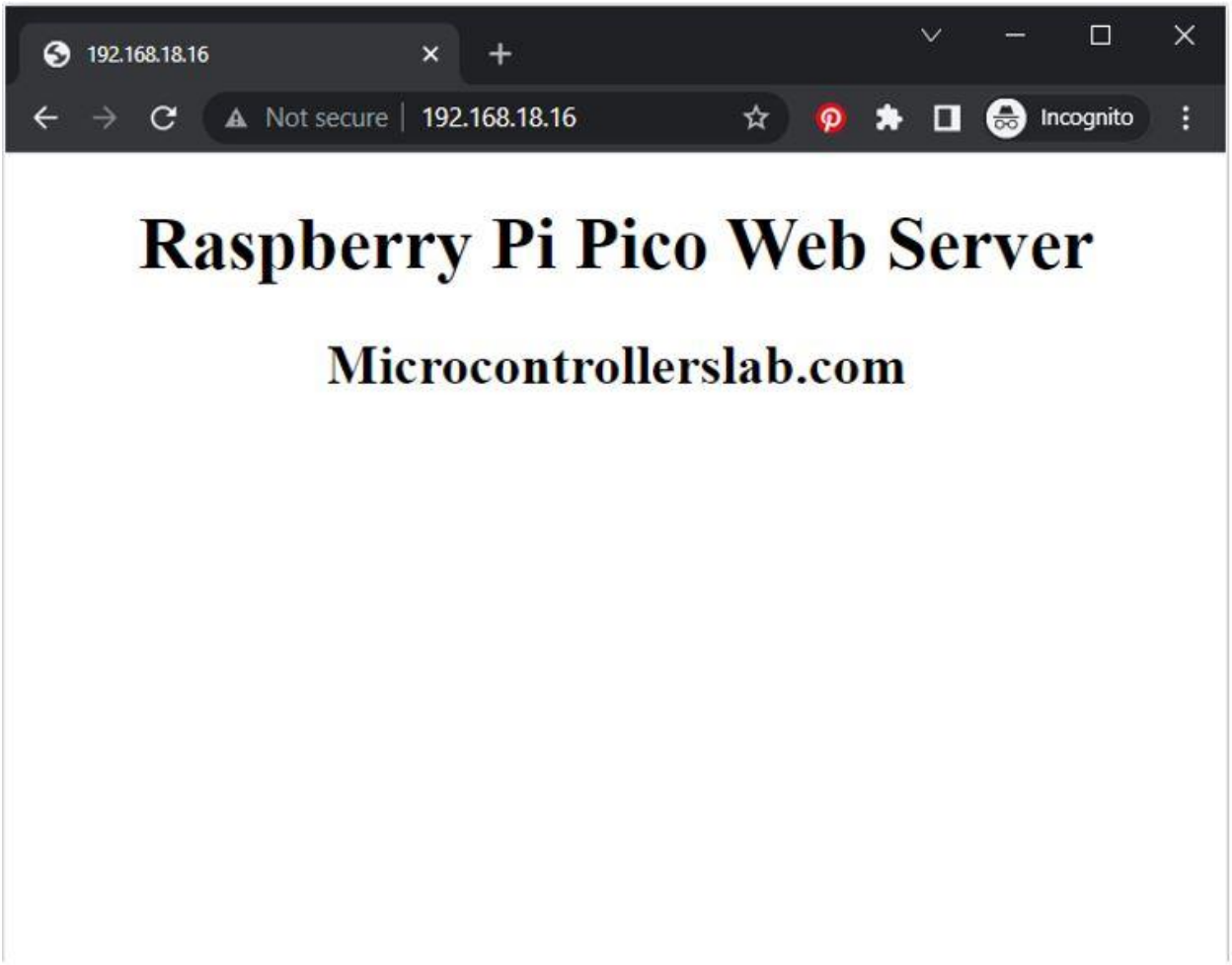
Demo

To test the MicroPython script, upload the main.py file to your board and get an IP address to access the web server:



In the shell terminal of your IDE, you will be able to view the following messages:

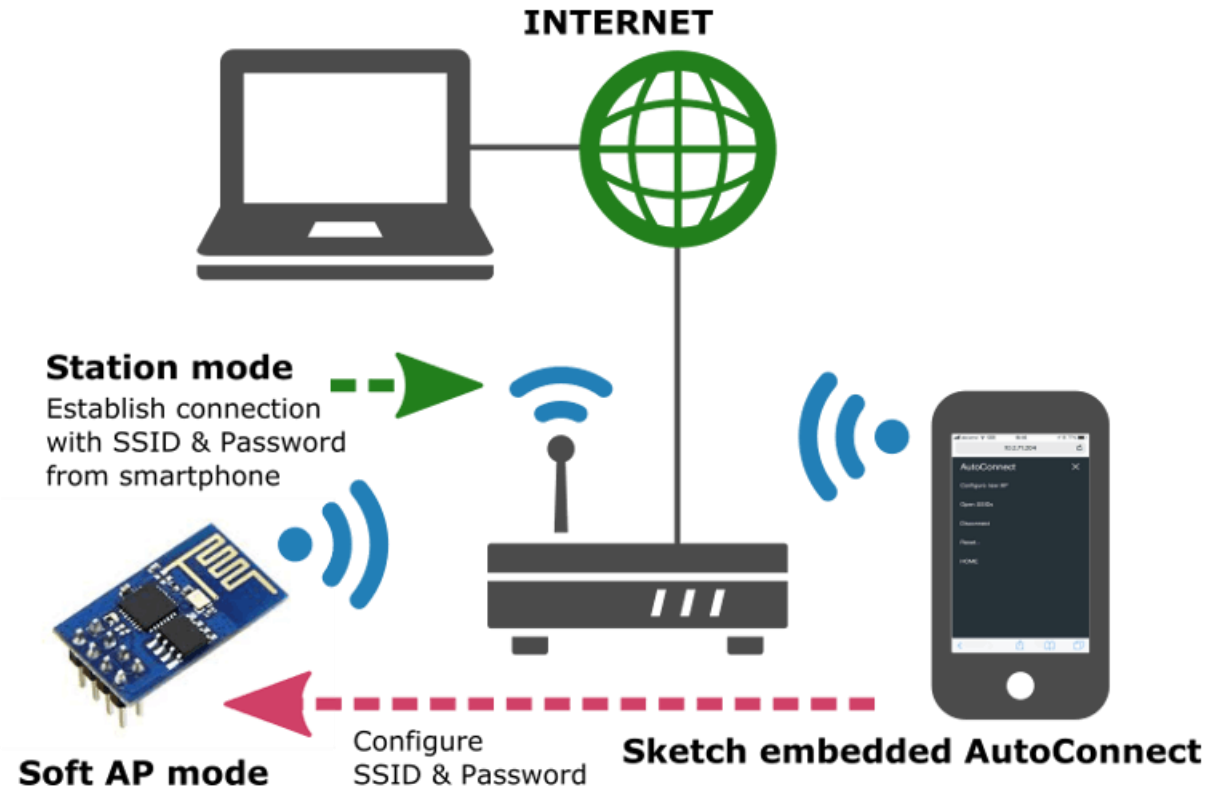
Now go to the web browser of your system and search with the IP address that was displayed in the shell terminal. The web page will open up.


192.168.18.16
Not secure | 192.168.18.16
Incognito
Raspberry Pi Pico Web Server
Microcontrollerslab.com

ESP8266/ESP32 Connect WiFi Made Easy

WLAN configuration at runtime on the ESP8266/ESP32 web interface for Arduino.

Beginner Full instructions provided 1 hour 68,320



Things used in this project

Hardware components

NodeMCU ESP8266 Breakout Board

AutoConnect works with either ESP8266 or ESP32. Which choose the options that you want.

× 1

Espressif ESP32S

AutoConnect works with either ESP8266 or ESP32. Which choose the options that you want.

× 1

Software apps and online services

Arduino IDE

Arduino library the AutoConnect and PageBuilder required.
https://hieromon.github.io/AutoConnect
https://github.com/Hieromon/PageBuilder

Story

If you use ESP8266 or ESP32 with Arduino Core you will certainly be issuing instructions: **WiFi.begin(SSID, Password)**. It can connect WLAN easily but the access point to be connected is fixed and inflexible.

Whenever the access point to be connected changes, do you rewrite the sketch and upload it?

You use AutoConnect as an Arduino library, you forget time and effort for the revising the sketch.

It an Arduino library united with ESP8266WebServer class for ESP8266 and WebServer class for ESP32. Easily implementing the Web interface constituting the WLAN for ESP8266/ESP32 WiFi connection. With this library to make a sketch easily which connects from ESP8266/ESP32 to the access point at runtime by the web interface without hard-coded SSID and password.

Overview

ESP8266 join to WLAN at runtime

No need pre-coded SSID & password

It is no needed hard-coding in advance the SSID and Password into the sketch to connect between ESP8266/ESP32 and WLAN. You can input SSID & Password from a smartphone via the web interface at runtime.

Simple usage

AutoConnect control screen will be displayed automatically for establishing new connections. It aids by the captive portal when vested the connection cannot be detected.By using the AutoConnect menu, to manage the connections convenient.

Store the established connection

The connection authentication data as credentials are saved automatically in EEPROM of ESP8266/ESP32 and You can select the past SSID from the AutoConnect menu.

Easy to embed in

AutoConnect can be embedded easily into your sketch, just "**begin**" and "**handleClient**". The AutoConnect API is the same for between ESP8266 and ESP32.

Lives with the your sketches

The sketches which provide the web page using ESP8266WebServer for ESP8266 or WebServer for ESP32 there is, AutoConnect will not disturb it. AutoConnect can use an already instantiated ESP8266WebServer or WebServer object, or itself can assign it.

Step #1 - Installation

Arduino IDE

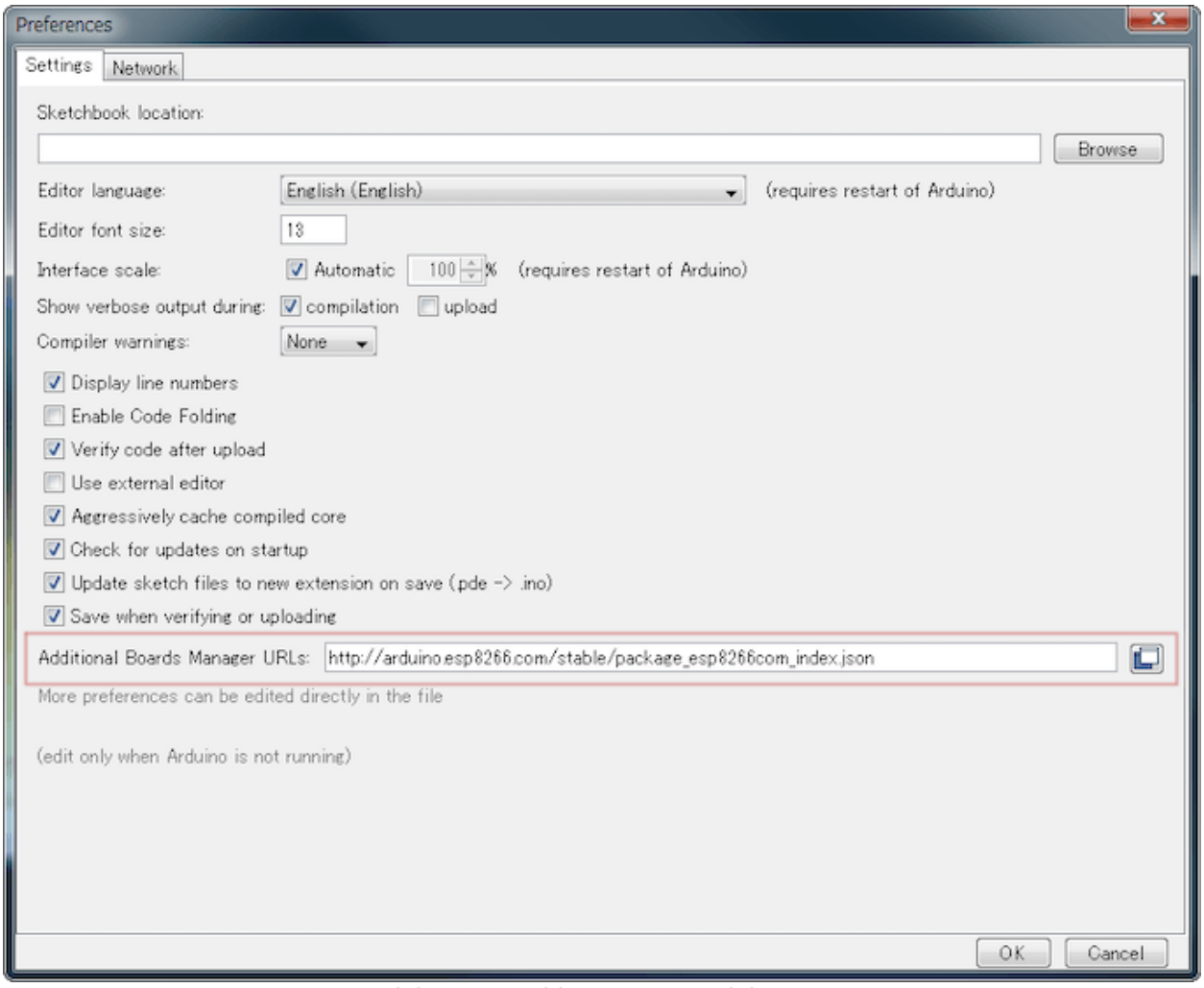
The current upstream at the 1.8 level or later is needed. Please install from the official Arduino IDE download page.

ESP8266 Arduino core

This library as the AutoConnect targets sketches made on the assumption of ESP8266 Community's Arduino core. Install third-party platform using the Boards Manager of Arduino IDE. The package URL is http://arduino.esp8266.com/stable/package_esp8266com_index.json

It can be introduced according to the following procedure.

- Open Arduino IDE, Go menu [File] > [Preference]
- Enter the [Additional Boards Manager URLs:] as http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Click OK.



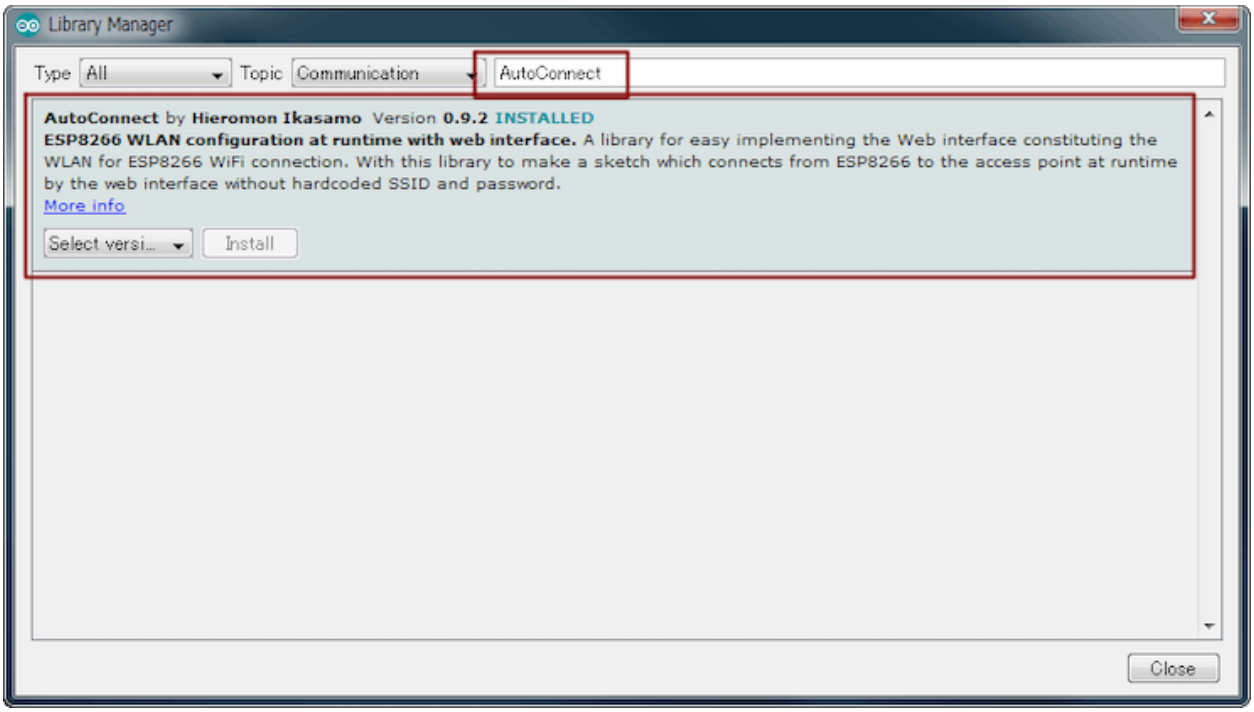
Arduino IDE, add ESP8266 Arduino core

If you want to use both ESP8266 and ESP32, add multiple URLs into Additional Board Manager URLs field, separating them with commas. The arduino-esp32 core package URL is https://dl.espressif.com/dl/package_esp32_index.json

Install the AutoConnect library

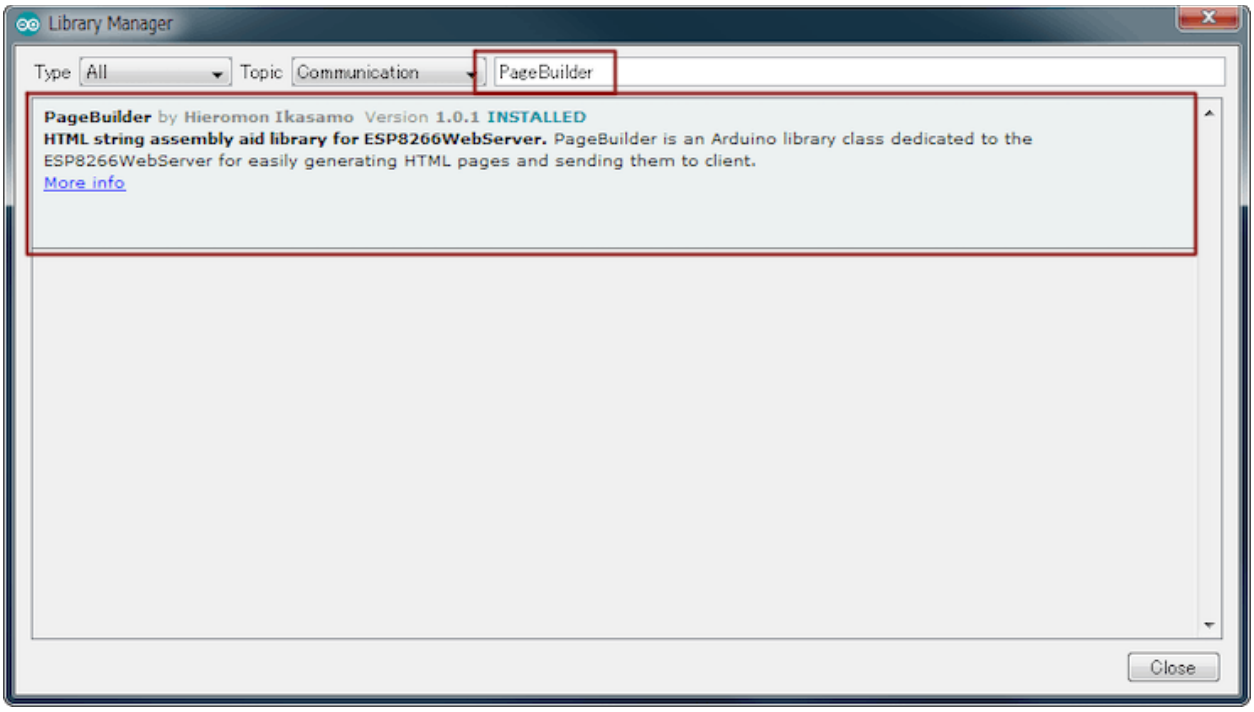
AutoConnect is available on the Arduino IDE Library manager. You can install the AutoConnect library from it easily.

- On Arduino IDE, Go menu [Sketch] -> [Include Library] -> [Manage Libraries...]
- Then pop up the Library manager, choose [Topic] and [Communication] from drop down list.
- Enter the Filter: **AutoConnect**.
- Library manager will show the like as display below.
- Select a latest version, click [Install].



Install the AutoConnect from Library manager

You also need an additional library called PageBuilder. It can be installed from the library manager too, the procedure is same as the above. It's [Communication] for [Topic], Filter is [PageBuilder].



Install the additional necessary library, the PageBuilder

Step #2 - Let's taste the AutoConnect ability.

Upload the sketch

Open the Arduino IDE, wrtie the "simple.ino" sketch and upload it. The feature of this sketch is that the SSID and Password are not coded.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <AutoConnect.h>
ESP8266WebServer Server;
AutoConnect      Portal(Server);
void rootPage() {
  char content[] = "Hello, world";
  Server.send(200, "text/plain", content);
}
void setup() {
  delay(1000);
  Serial.begin(115200);
  Serial.println();
  Server.on("/", rootPage);
  if (Portal.begin()) {
```

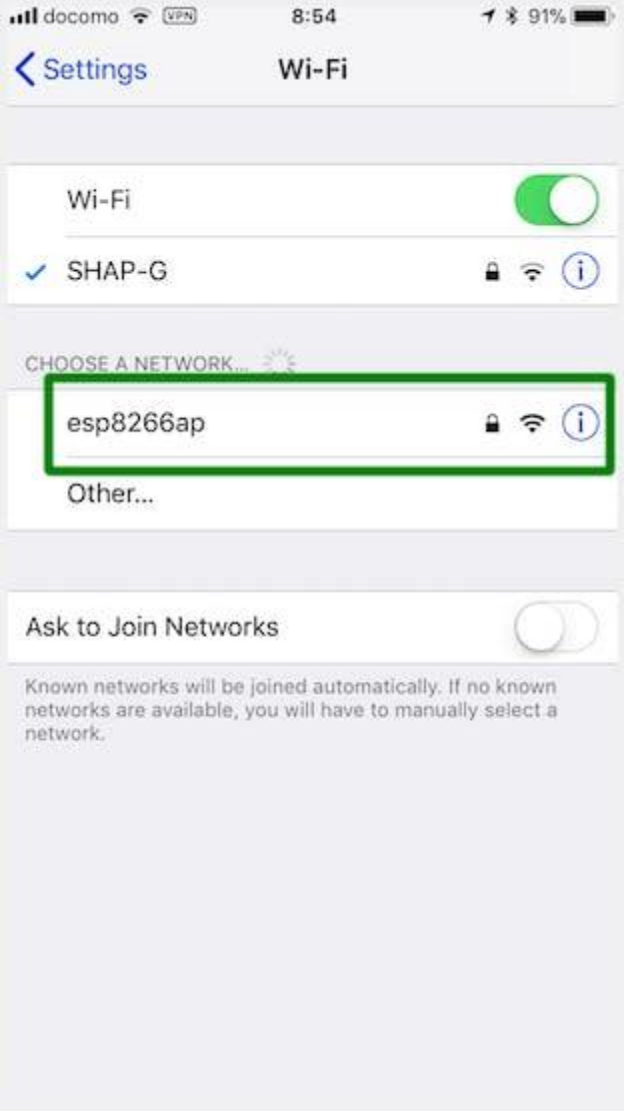
```
    Serial.println("HTTP server:" + WiFi.localIP().toString());
  }
}
void loop() {
  Portal.handleClient();
}
```

The above code assumes ESP8266. The include directives and ESP8266WebServer class can be modified and applied to ESP32 as follows.

```
#include <WiFi.h>
#include <WebServer.h>
WerServer Server;
```

Run at first

Once reset the ESP8266, starts it. After about 30 seconds, please take the smartphone and open Wi-Fi setting Apps. You can see the **esp8266ap** in the list of "*CHOOSE A NETWORK...*". Then tap the esp8266ap and enter password **12345678**, a something screen pops up automatically as shown below.

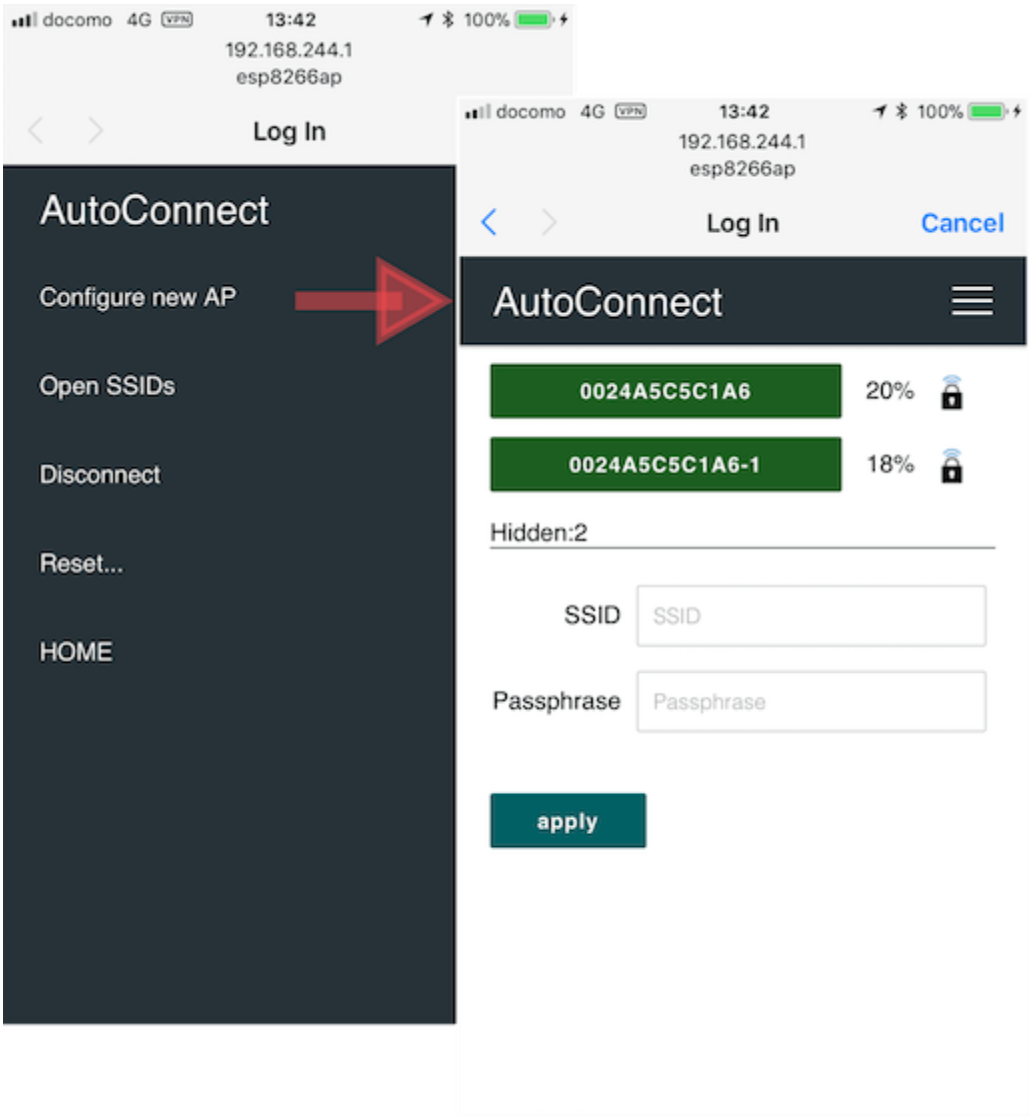


1 / 2

The hamburger icon is the control menu of AutoConnect seems at the upper right. You can connect to the access point any placed by using this menu.

Join to the access point

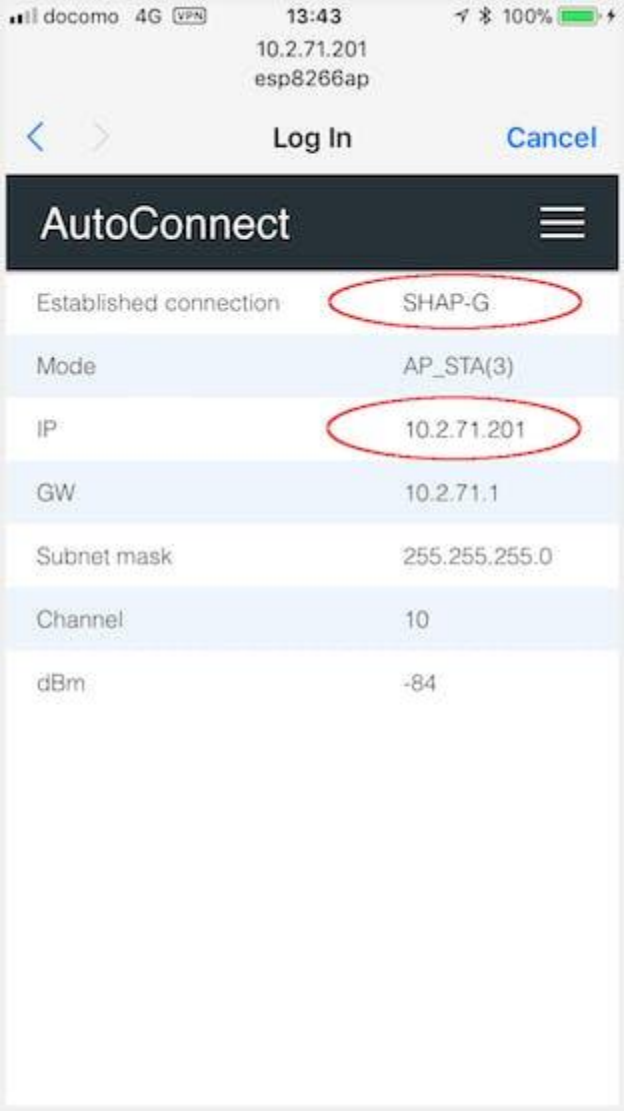
Here, tap "*Configure new AP*" then SSID configuration screen would be shown. Enter the **SSID** and **Passphrase** and tap "*apply"* to start connecting.



Configuration the SSID for the access point

Connection establishment

After connection established, the current status screen will appear. It is already connected to WLAN.

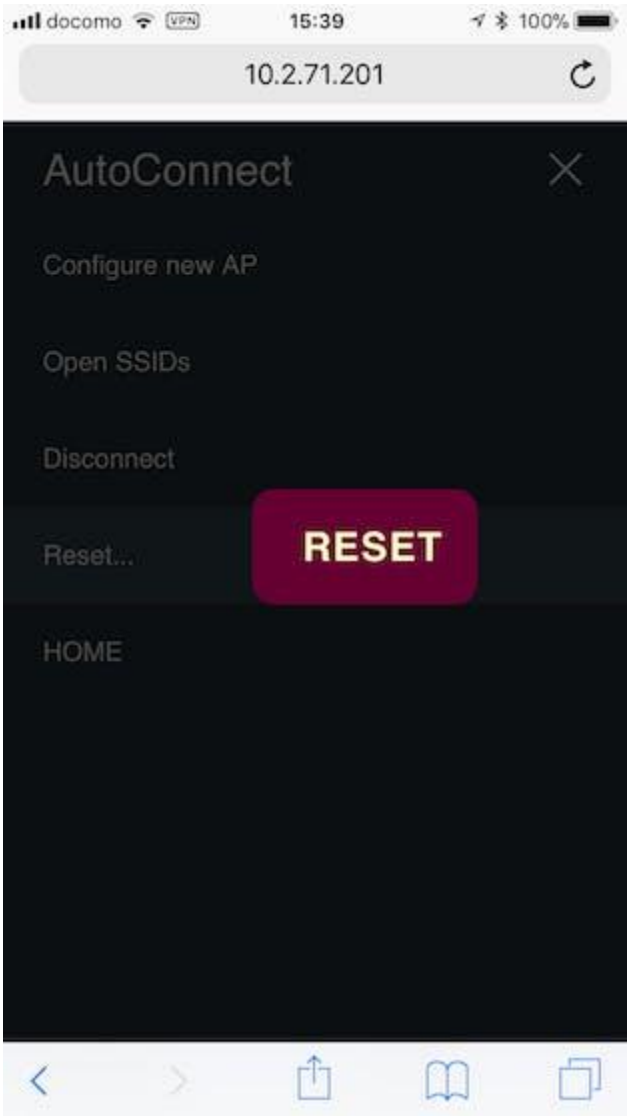


A connection established

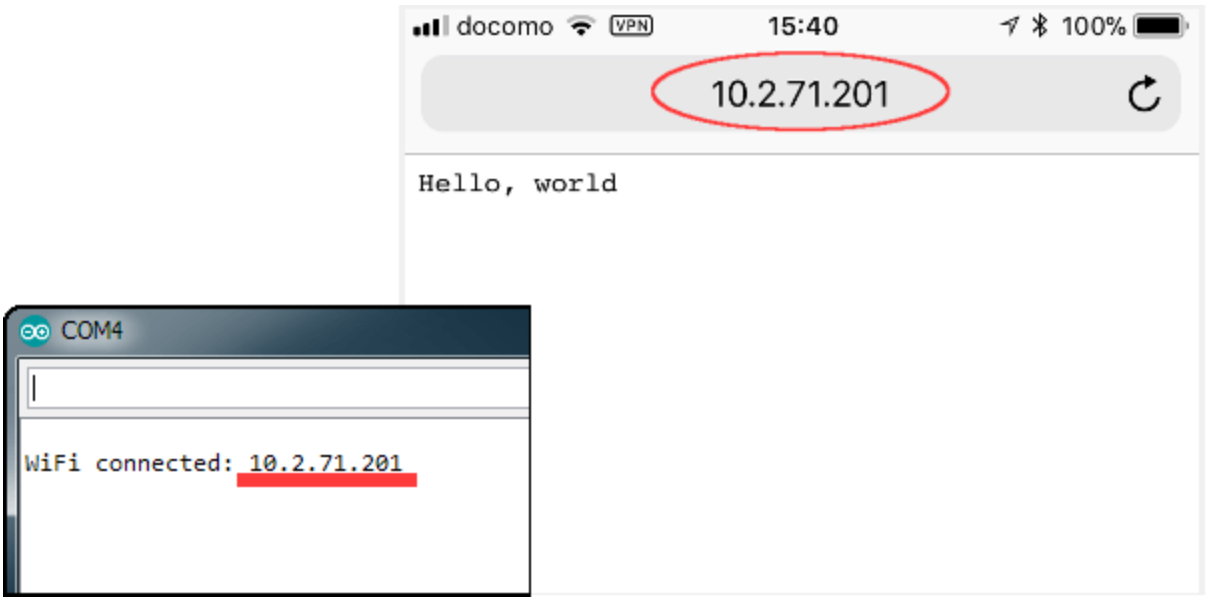
After connection established, you have two choices for the next step.

For one, continues execution the sketch action while keeping this connection. You can access the ESP8266 with connected IP address from the browser on PC which is participating in the same WLAN. You will see the web page as "Hello, world".

Or, "RESET..." can be selected from the menu. The ESP8266 resets and starts rebooting. After the rebooting complete, ESP8266 will restore the connection which was established previously.



Reset the ESP8266



Run for usually

Once established connection can be used continuously. You can operate from the AutoConnect menu if you want to connect to another access point.

AutoConnect also remembers past access points. It can be selected from the AutoConnect menu. You do not need to re-enter the password.

Select from the past SSID

Step #3 - Use with MQTT client

The effect of AutoConnect is not only for ESP8266 as the Web server. It has a benefit for something WiFi client too.

AutoConnect is effective too when publishing with MQTT from various measurement points. Even if the SSID is different for each measurement point, it is no need to modify the sketch.

This step tries to publish the WiFi signal strength of ESP8266 with MQTT. It uses the ThingSpeak for MQTT broker. ESP8266 publishes the RSSI value to

the channel created on ThingSpeak as MQTT client. This example is well suited to demonstrate the usefulness of AutoConnect, as RSSI values are measured at each access point usually. Just adding a few lines of code makes it unnecessary to upload sketches with the different SSIDs rewrite for each access point.

Advance procedure

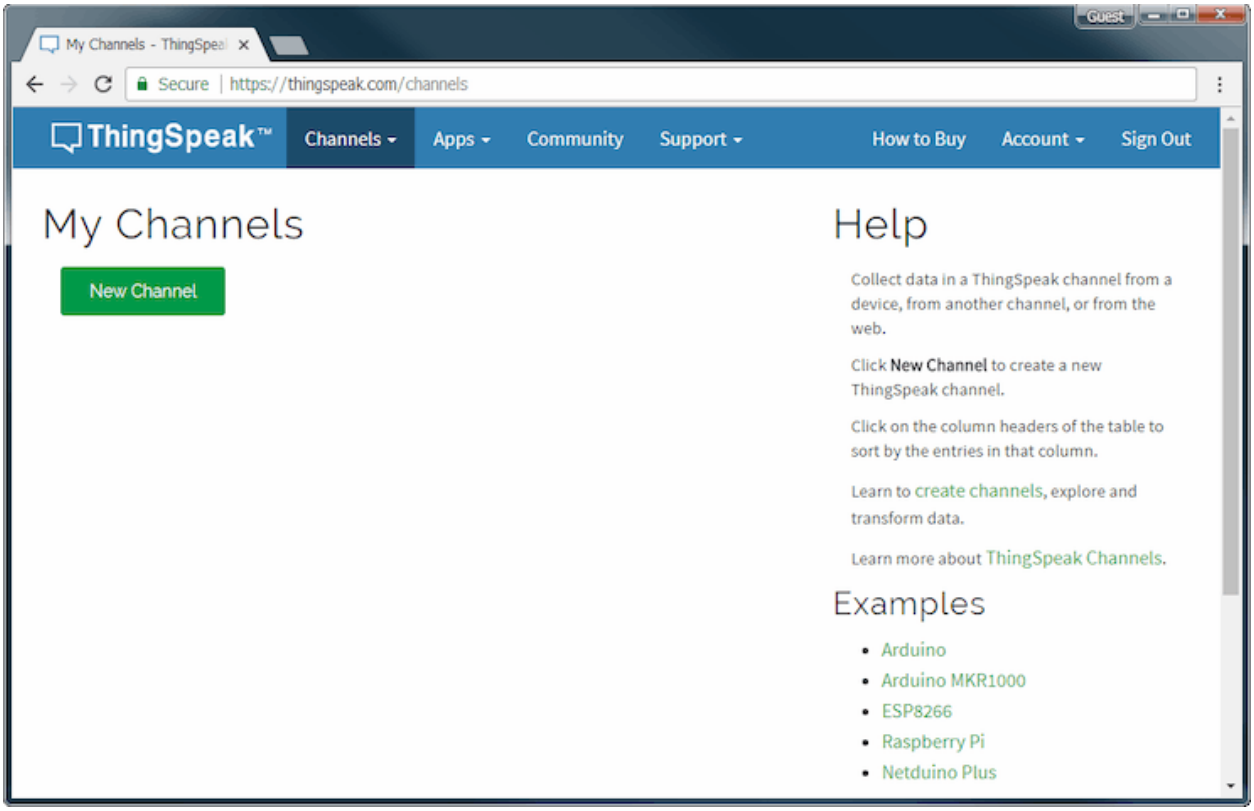
- Arduino Client for MQTT - It's the PubSubClient, install it to Arduino IDE from here. If you have the latest version already, this step does not need.
- Create a channel on ThingSpeak.
- Get the Channel API Keys from ThingSpeak, put its keys to the sketch.

The ThingSpeak is the open IoT platform. It is capable of sending data privately to the cloud and analyzing, visualizing its data. If you do not have an account of ThingSpeak, you need that account to proceed further. ThingSpeak is free for use within the scope of this example. (As of March 21, 2018). You can sign up with the ThingSpeak sign-up page. (But you are entrusted with the final judgment of account creation. Create an account at your own risk.)

From here onward, the steps will proceed on the premise that ThingSpeak account is available.

Create a channel on ThingSpeak

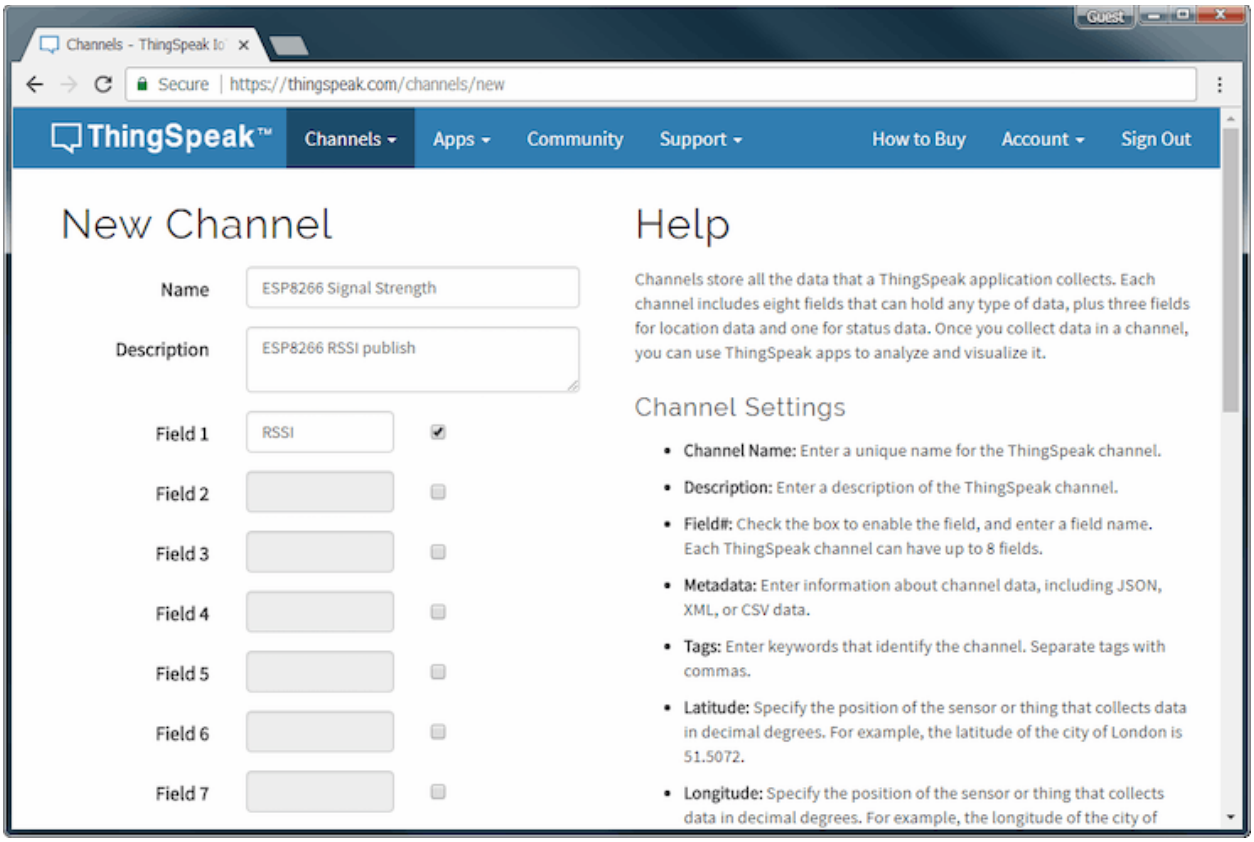
Sign in ThingSpeak. Select [Chanels] to show the My Channels, then click [New Channel].



Open My Channels

At the New Channel screen, enter each field as a below. And click [Save Channel] at the bottom of the screen to save.

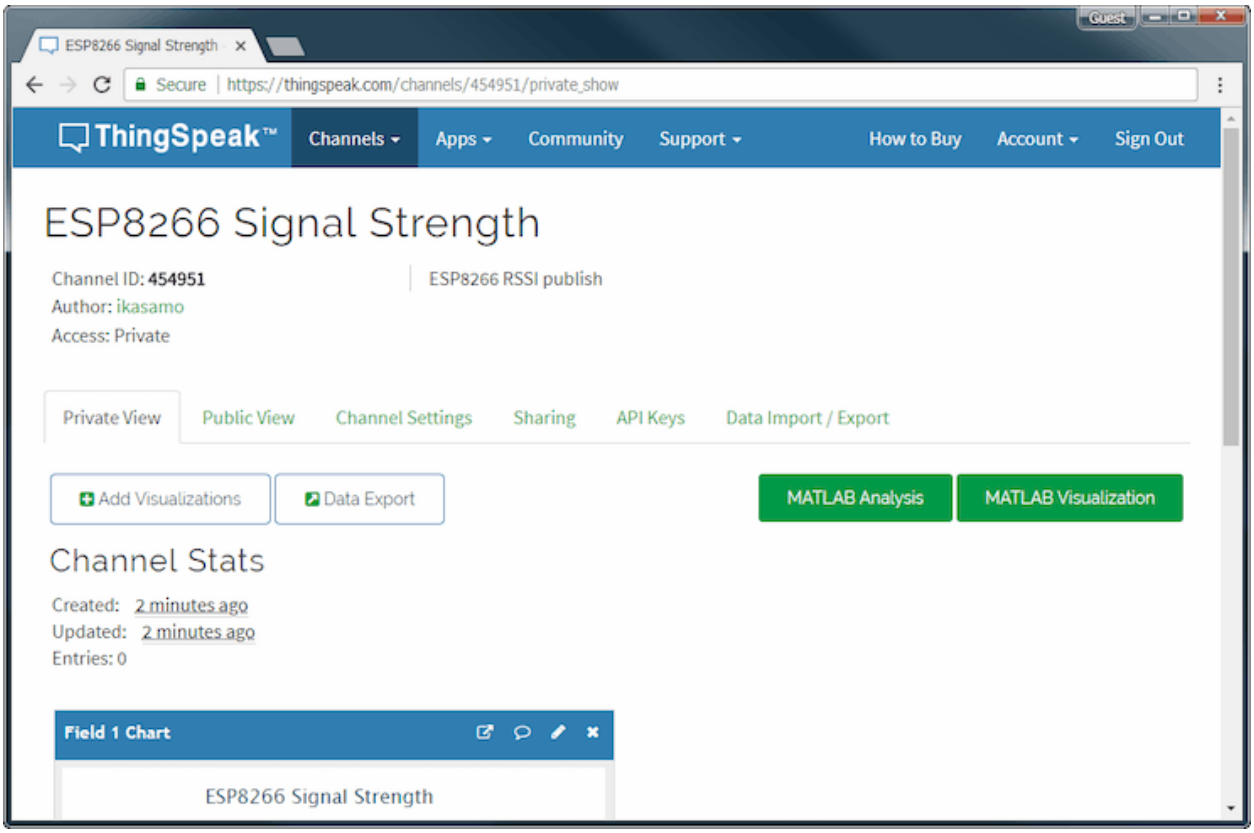
- Name: **ESP8266 Signal Strength**
- Description: **ESP8266 RSSI publish**
- Field1: **RSSI**



Create a channel, save it

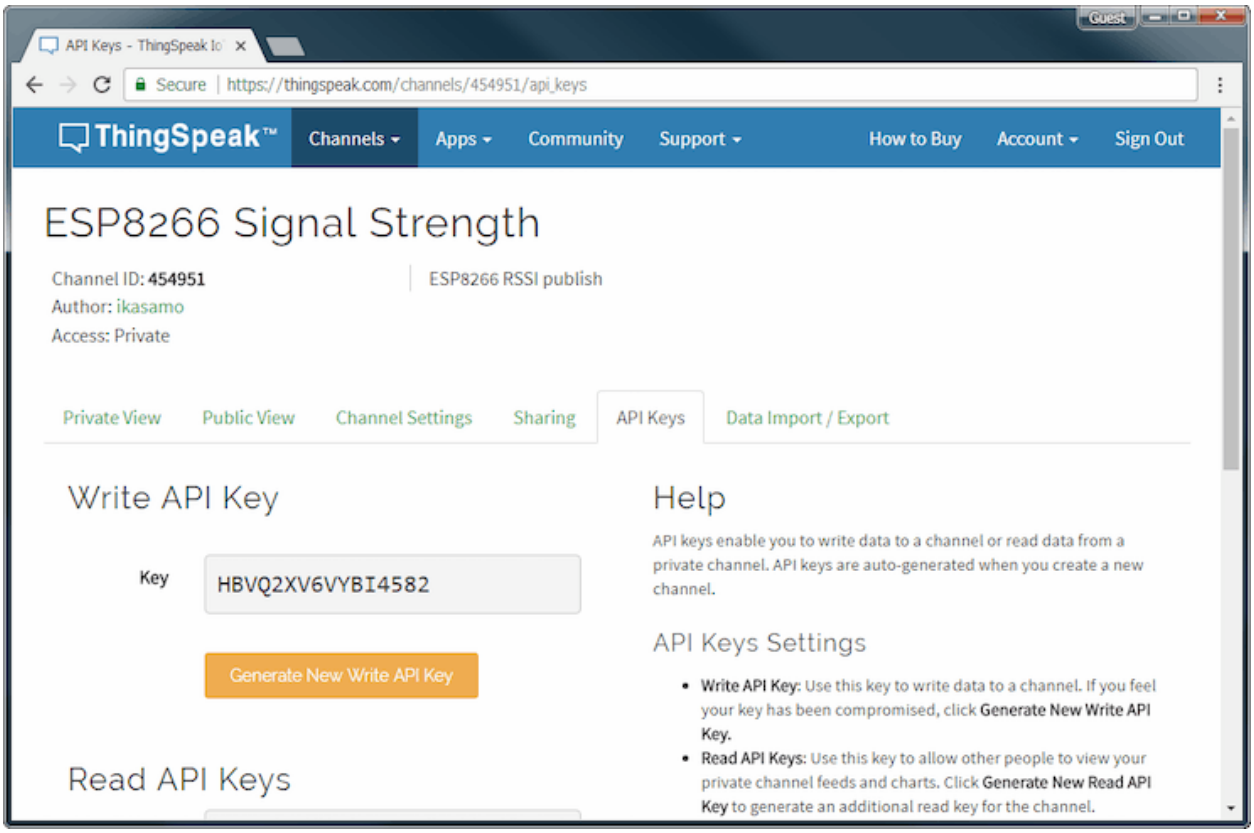
Get Channel ID and API Keys

The channel successfully created, you can see the channel status screen as a below. **Channel ID** is displayed there. (454951 in the example below, but your channel ID should be different.)



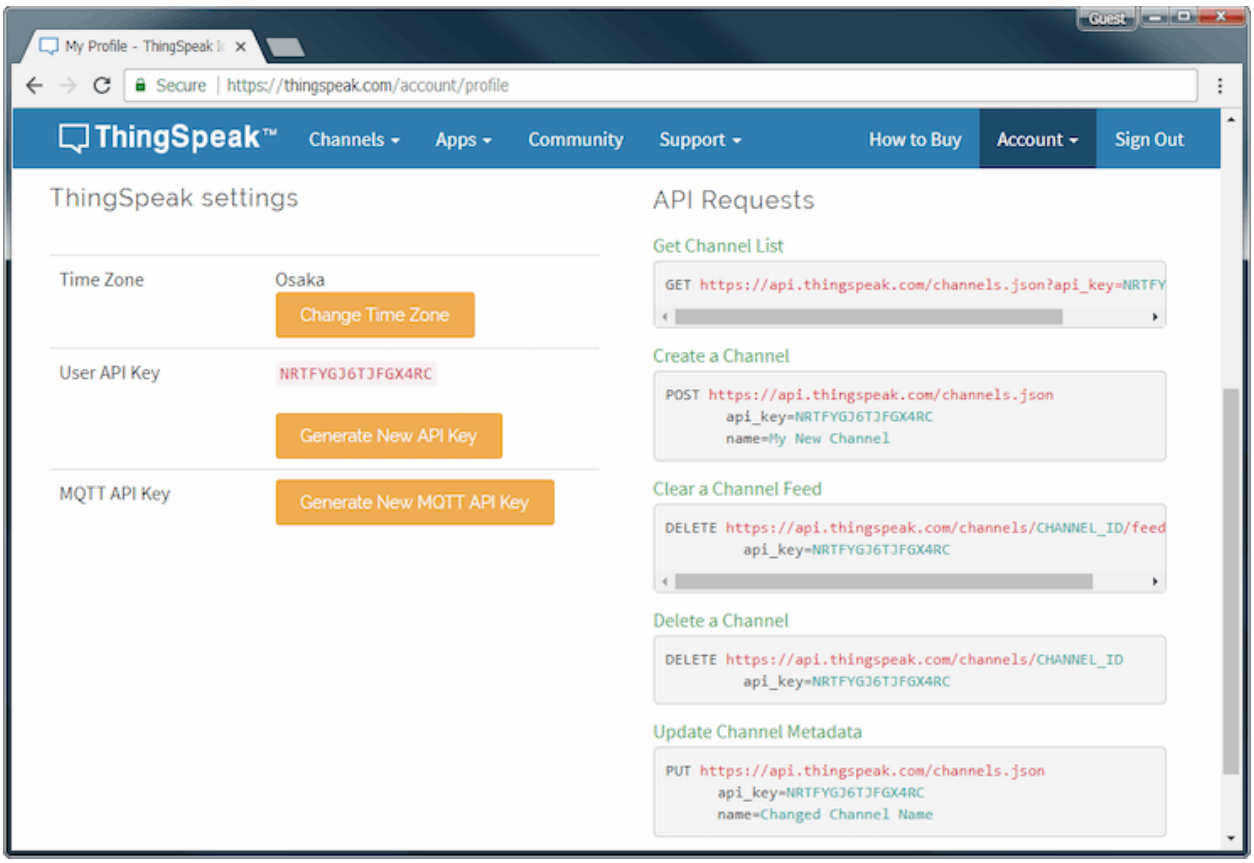
Channel ID

Here, switch the channel status tab to [API Keys]. The API key required to publish the message is the **Write API Key**.



API Key

The last key you need is the **User API Key** and can be confirmed it in the user profile. Pull down [Account] from the top menu, select [My profile]. Then you can see the ThingSpeak settings and the User API Key is displayed middle of this screen.



User API Key

Now, publishing the ESP8266 signal strength to MQTT broker is ready. Next will upload the sketch to publish the message to the channel you just created. Of course, it includes connection assistance by AutoConnect.

The sketch, Publishes messages

The complete code of the sketch is mqttRSSI.ino. Replace the following #define in a sketch with **User API Key**, **Write API Key** and **Channel ID**.

```
#define MQTT_USER_KEY          "****************"  // Replace to User API Key.
#define CHANNEL_ID             "******"            // Replace to Channel ID.
#define CHANNEL_API_KEY_WR     "****************"  // Replace to the write API Key.
```

After Keys updated, compile the sketch and upload it.

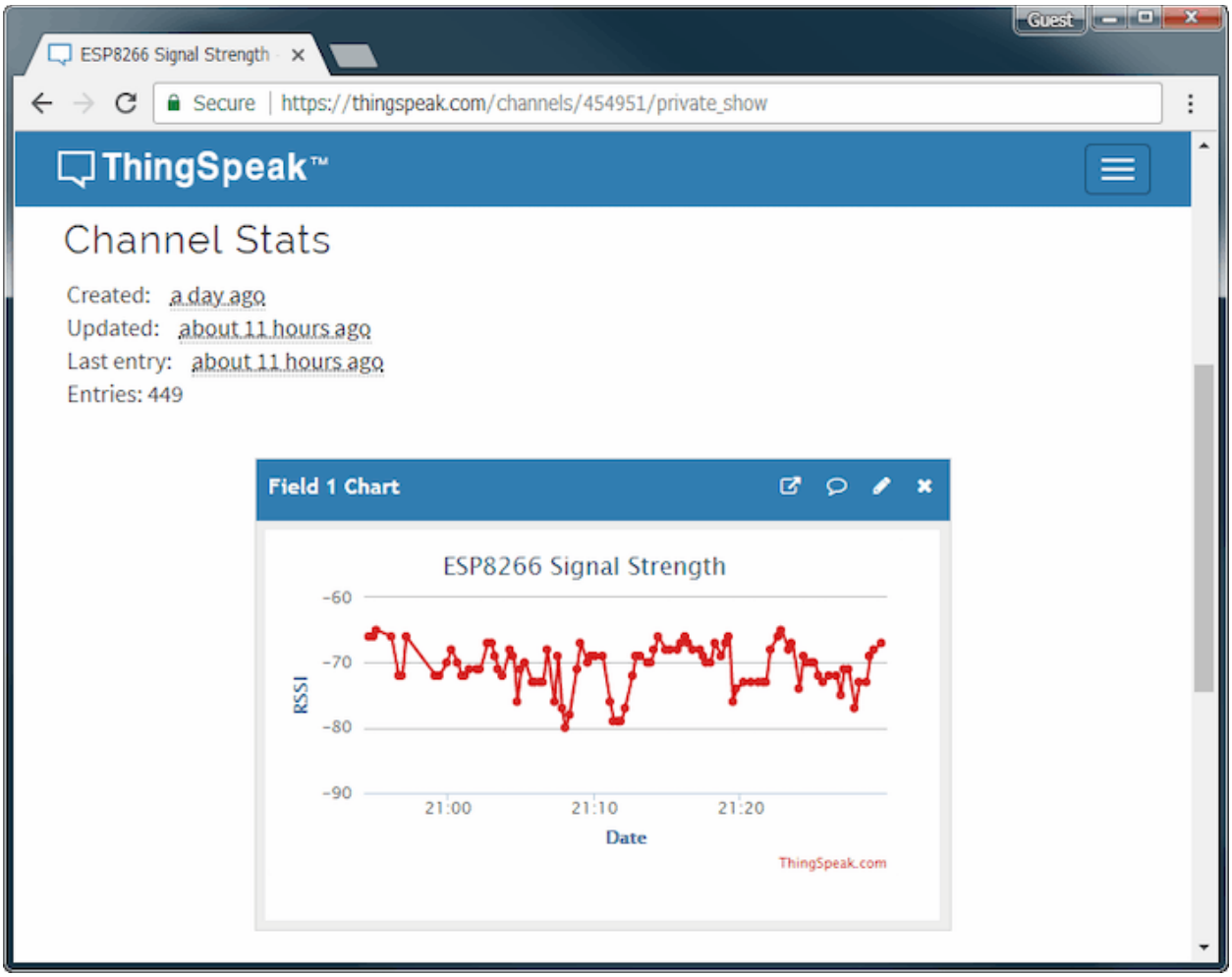
Connect WiFi

Run the sketch after resets once. If it fails the first WiFi.begin, you can see the **esp8266ap** as an access point of the WiFi settings with your smartphone.

Let connects the esp8266ap and invokes AutoConnect menu. Configure the SSID you want to connect and resets by [RESET...] menu.

Publish messages

After reboot complete, the message publishing will start via the access point now set. The message carries RSSI as the current WiFi signal strength. The signal strength variations in RSSI are displayed on ThingSpeak's Channel status screen.



RSSI signal strength variations

Step #4 - How embed to your sketches

I think you are already aware that SSID & Password is missing and WiFi.begin() is not coded. The code required to connect to WiFi is the following four parts only.

1. #include directive

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <PubSubClient.h>
#include <AutoConnect.h>
```

#include directive

2. Declare AutoConnect

```
AutoConnect  portal;
WiFiClient   wifiClient;
PubSubClient mqttClient(wifiClient);
```

Declare AutoConnect

3. Invokes "begin()"

```
  Serial.print("WiFi ");
  if (portal.begin()) {
    Serial.println("connected:" + WiFi.SSID());
    Serial.println("IP:" + WiFi.localIP().toString());
  } else {
    Serial.println("connection failed:" + String(WiFi.s
    while (1) {
```

Invokes AutoConnect::begin()

4. Performs "handleClient()" in "loop()"

```
void loop() {
  if (millis() - lastPub > MQTT_UPDATE_INTERVAL) {
    if (!mqttClient.connected()) {
      mqttConnect();
    }
    String item = String("field1=") + String(WiFi.RSSI(
    mqttPublish(item);
    mqttClient.loop();
    lastPub = millis();
  }
  portal.handleClient();
}
```

AutoConnect::handleClient()

Step #5 - Simply get it done quickly

You can easily add your own web screen that can consist of representative HTML elements as the styled TEXT, INPUT, BUTTON, CHECKBOX, RADIO, SELECT, SUBMIT into the menu. It can be invoked from the AutoConnect menu and parameters can be passed. These HTML elements that make up the user-owned screen can be easily loaded from the JSON description stored in PROGMEM, SPIFFS or SD.

```
{
  "title": "MQTT Setting",
  "uri": "/mqtt_setting",
  "menu": true,
  "element": [
    {
      "name": "header",
      "type": "ACText",
      "value": "<h2>>MQTT broker settings</h2>",
      "style": "text-align:center;color:#2f4f4f;"
    },
    {
      "name": "caption",
      "type": "ACText",
      "value": "Publishing the WiFi signal strength to MQTT
      "style": "font-family:serif;color:#4682b4;"
    },
    {
      "name": "mqttserver",
      "type": "ACInput",
      "value": "",
      "placeholder": "MQTT broker server",
      "label": "Server"
    },
    {
      "name": "channelid",
      "type": "ACInput",
      "value": "",
      "label": "Channel ID"
```

Json document for the custom Web page

You can easily incorporate the web interface as like this into your sketch.

The above sketch is available in GitHub AutoConnect repository.

Appendix - More usage and FAQs

Full documentation is available on https://hieromon.github.io/AutoConnect/, some quick links at the list:

- The Installation is the installation procedure and requirements for AutoConnect library.
- Getting started with the most simple sketch for using AutoConnect.

- The Basic usage guides to using the library correctly.
- Details are explained in the Advanced usage.
- Details and usage of custom Web pages are explained in the Custom Web pages.
- The API reference describes the AutoConnect functions specification.
- There are hints in Examples for making sketches with AutoConnect.
- FAQ.

Code

- **mqttRSSI.ino**
- simple.ino

mqttRSSI.ino

Arduino

ESP8266 publish the RSSI as the WiFi signal strength to ThingSpeak channel.

```
/*
  ESP8266 publish the RSSI as the WiFi signal strength to ThingSpeak channel.
  This example is for explaining how to use the AutoConnect library.

  In order to execute this example, the ThingSpeak account is needed. Sing up
  for New User Account and create a New Channel via My Channels.
  For details, please refer to the project page.
  https://www.hackster.io/hieromon-ikasamo/esp8266-connect-wifi-make-easily-d75f45

  This example is based on the environment as of March 20, 2018.
  Copyright (c) 2018 Hieromon Ikasamo.
  This software is released under the MIT License.
  https://opensource.org/licenses/MIT
*/

#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <PubSubClient.h>
#include <AutoConnect.h>

#define MQTT_USER_KEY        "****************"  // Replace to User API Key.
#define CHANNEL_ID           "******"            // Replace to Channel ID.
#define CHANNEL_API_KEY_WR   "****************"  // Replace to the write API Key.
```

```
#define MQTT_UPDATE_INTERVAL 15000
#define MQTT_TOPIC           "channels/" CHANNEL_ID "/publish/" CHANNEL_API_KEY_WR
#define MQTT_USER_ID         "anyone"
#define MQTT_SERVER          "mqtt3.thingspeak.com"

AutoConnect  portal;
WiFiClient   wifiClient;
PubSubClient mqttClient(wifiClient);

bool mqttConnect() {
  static const char alphanum[] = "0123456789"
                                 "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
                                 "abcdefghijklmnopqrstuvwxyz";  // For random
generation of client ID.
  char    clientId[9];

  uint8_t retry = 10;
  while (!mqttClient.connected()) {
    Serial.println("Attempting MQTT broker:" MQTT_SERVER);

    for (uint8_t i = 0; i < 8; i++) {
      clientId[i] = alphanum[random(62)];
    }
    clientId[8] = '\0';

    if (mqttClient.connect(clientId, MQTT_USER_ID, MQTT_USER_KEY)) {
      Serial.println("Established:" + String(clientId));
      return true;
    } else {
      Serial.print("Connection failed:" + String(mqttClient.state()));
      if (!--retry)
        return false;
    }
    delay(3000);
  }
}

void mqttPublish(String msg) {
  String path = String(MQTT_TOPIC);
  int    tLen = path.length();
  char   topic[tLen];
  path.toCharArray(topic, tLen + 1);

  int    mLen = msg.length();
  char   payload[mLen];
  msg.toCharArray(payload, mLen + 1);

  mqttClient.publish(topic, payload);
}

unsigned long   lastPub = 0;

void setup() {
  delay(1000);
  Serial.begin(115200);
```

```
  Serial.println();

  Serial.print("WiFi ");
  if (portal.begin()) {
    Serial.println("connected:" + WiFi.SSID());
    Serial.println("IP:" + WiFi.localIP().toString());
  } else {
    Serial.println("connection failed:" + String(WiFi.status()));
    while (1) {
      delay(100);
      yield();
    }
  }
  mqttClient.setServer(MQTT_SERVER, 1883);
}

void loop() {
  if (millis() - lastPub > MQTT_UPDATE_INTERVAL) {
    if (!mqttClient.connected()) {
      mqttConnect();
    }
    String item = String("field1=") + String(WiFi.RSSI());
    mqttPublish(item);
    mqttClient.loop();
    lastPub = millis();
  }
  portal.handleClient();
}
```

simple.ino

Arduino

"Hello, world" web server on ESP8266. It's for the AutoConenct experience.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <AutoConnect.h>

ESP8266WebServer Server;
AutoConnect      Portal(Server);

void rootPage() {
  char content[] = "Hello, world";
  Server.send(200, "text/plain", content);
}

void setup() {
```

```
  delay(1000);
  Serial.begin(115200);
  Serial.println();

  Server.on("/", rootPage);
  if (Portal.begin()) {
    Serial.println("HTTP server:" + WiFi.localIP().toString());
  }
}

void loop() {
    Portal.handleClient();
}
```

How to Configure an ESP Mesh Network using Arduino IDE – Communicate among and between ESP32, ESP8266, and NodeMCU

Published January 18, 2021 8


Debashis Das
Author

ESP Wi-Fi MESH Network

Internet of Things (IoT) has seen exponential growth over the past couple of years. A new study from International Data Corporation (IDC) estimates that there will be almost 42 billion connected devices within the year 2025, generating over 80 zettabytes (ZB) of data. As the number of IoT

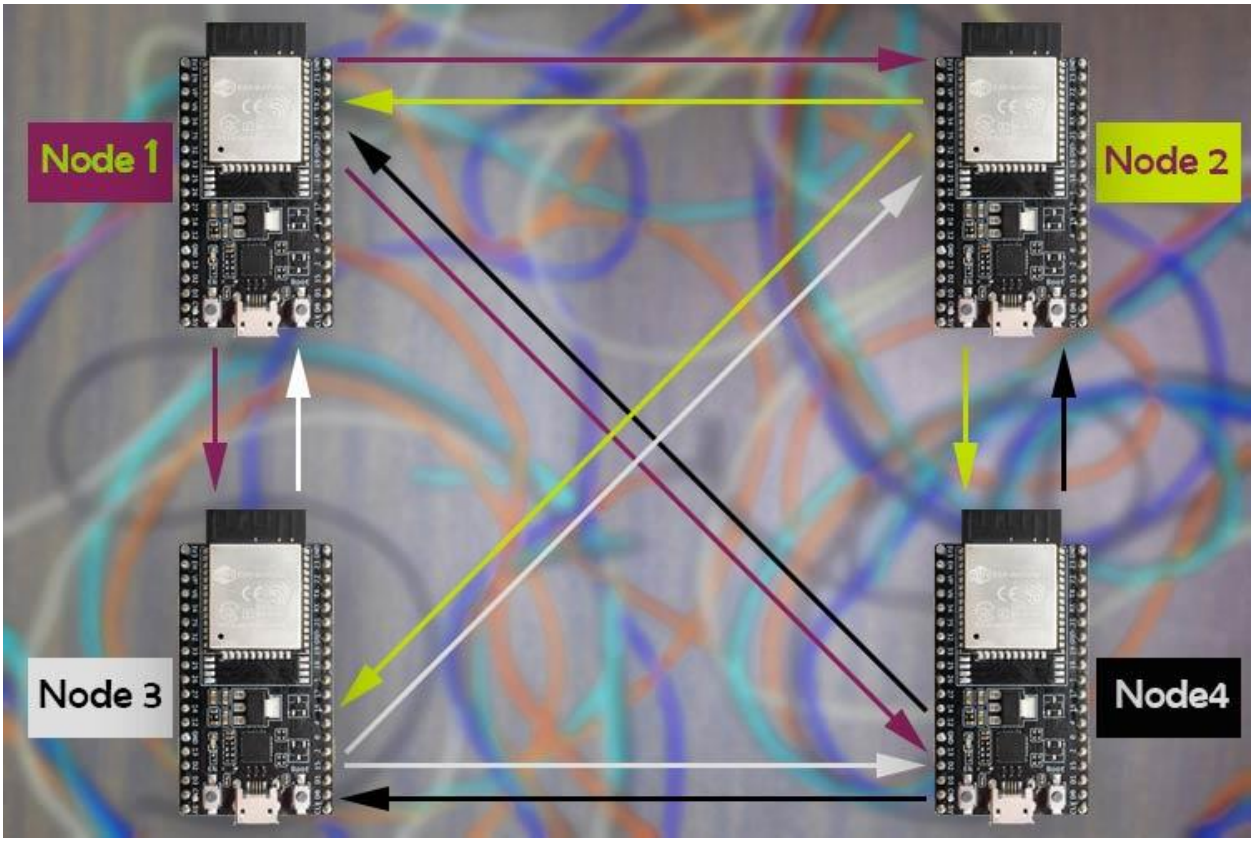
devices grows; the amount of data grows, along with that, grows the need for superior network instruments; which can support this load.

However, if we consider a common host (like a generic router), it can connect to a limited number of nodes, less than 32 to be exact. And with an increasing number of IoT devices that could be in our home or industry, this is not sufficient. Currently, there are two solutions to this problem: The first one is to use a **Mesh Router** that can handle a lot more connections compared to a generic one, or we can use a network protocol known as **Mesh Network**.

So in this article, we are going to make a simple **ESP Mesh network** setup that consists of four ESP devices that will communicate with each other with the help of a **Wi-Fi Mesh Network**. And finally, we are going to connect a single ESP to our laptop in order to get data from all the four sensors on the network. Note that we will be using both the ESP32 and ESP8266 boards in this tutorial so that you can create an **ESP8266 Mesh network** or **ESP32 Mesh network** using the same method.

What is ESP-MESH and How it Work?

According to the official documentation of ESP-MESH, it is a self-organizing and self-healing network meaning the network can be built and maintained autonomously. For more information, visit the ESP-MESH official documentation.



A mesh network is a group of connected devices on a network that is acting as a single network. **ESP-Mesh** is quite different from a traditional mesh setup. In an ESP-Mesh, the node or a single device can connect to its neighbor simultaneously. One single node can connect to multiple nodes and they can relay data from one node to another. This process is not only efficient but it's also redundant. If one of any nodes fails; the data from other nodes can reach its destination without a problem. This also opens possibilities of achieving interconnection without needing a central node, which significantly extends the coverage area of the mesh-network. With these features, this network is less prone to conjunction because the total number of nodes in the network is not limited by a single central node.

For the sake of simplicity, we have decided to use four ESP modules; but if you are building this network, you can use as many ESPdevices as you can. To build the mesh network, we are going

to use the **painlessMesh library** for Arduino which has support for both the ESP8266 and the ESP32 modules.

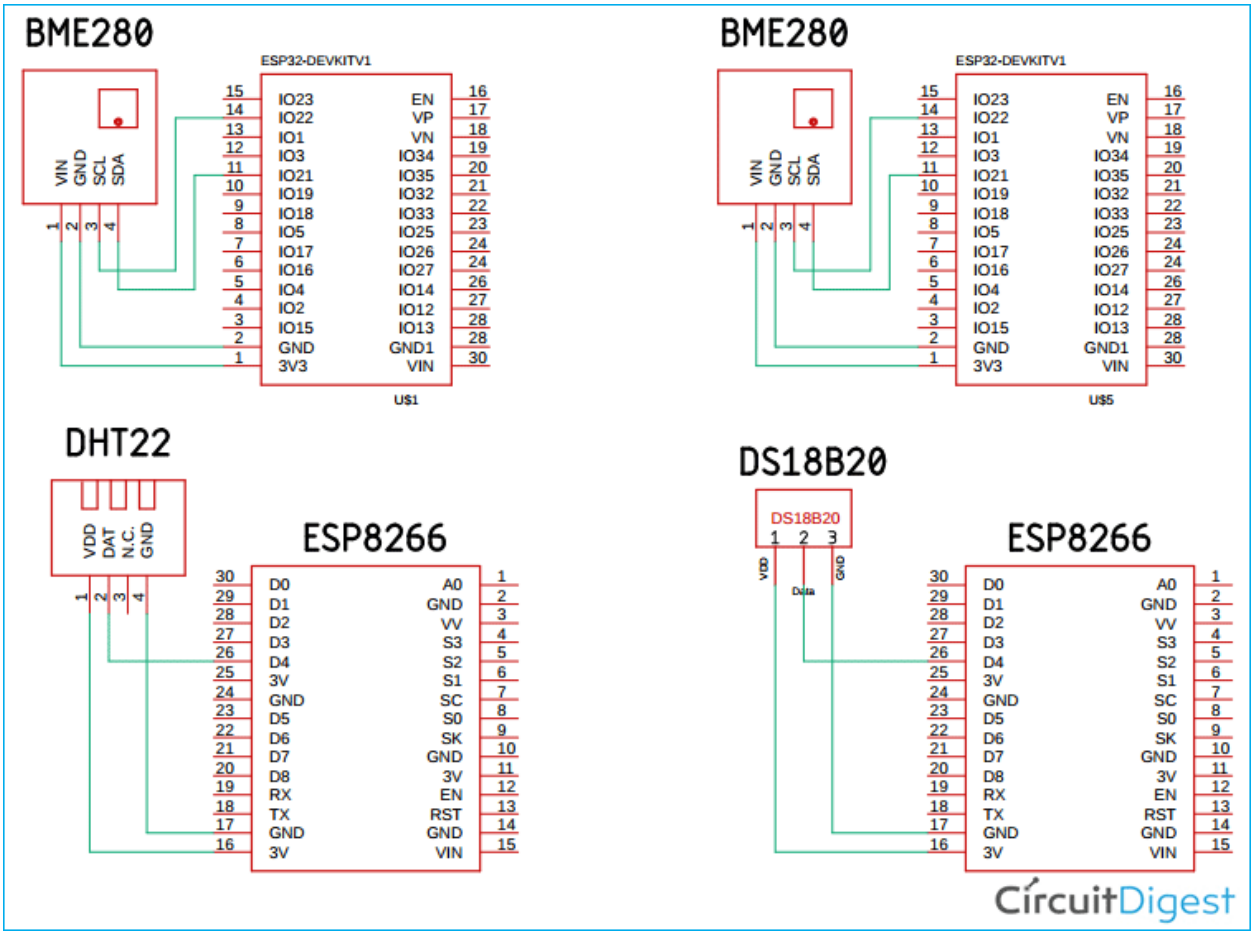
Components Required to build the Mesh Network with ESP

The list of components required for this project is given below. To build this project, I have used components that are pretty generic and you can find them in your local hobby store.

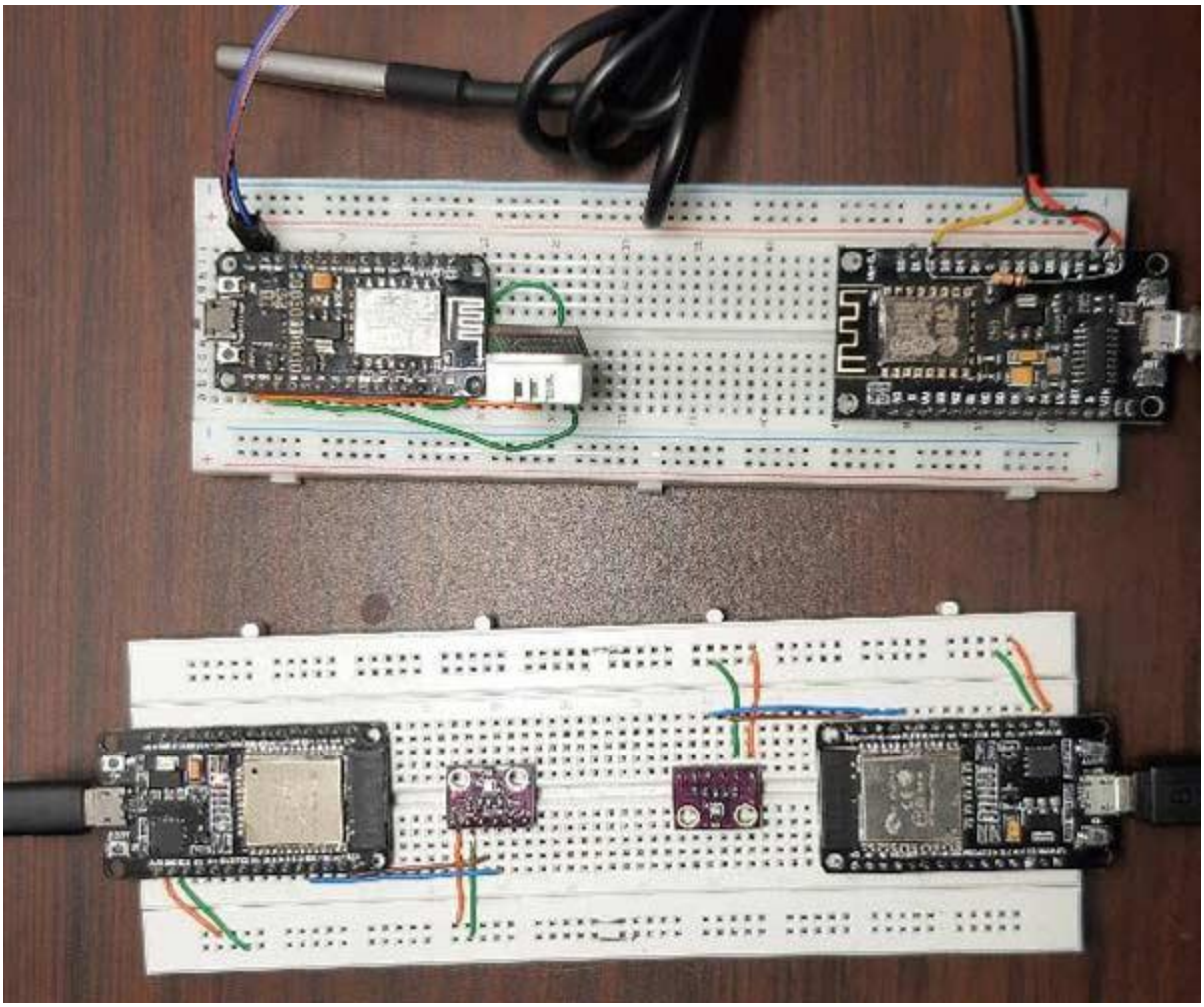
- NodeMCU(ESP8266) - 2
- ESP32 Dev Board - 2
- BMP280 Sensor - 2
- DHT22 Sensor - 1
- DS18B20 Sensor - 1
- Breadboard
- USB Cable (for power and data)

ESP Wi-Fi Mesh - Circuit Diagram

The schematic shown below is used to construct the hardware section for **ESP8266 and ESP32 based Wi-Fi Mesh Network**.

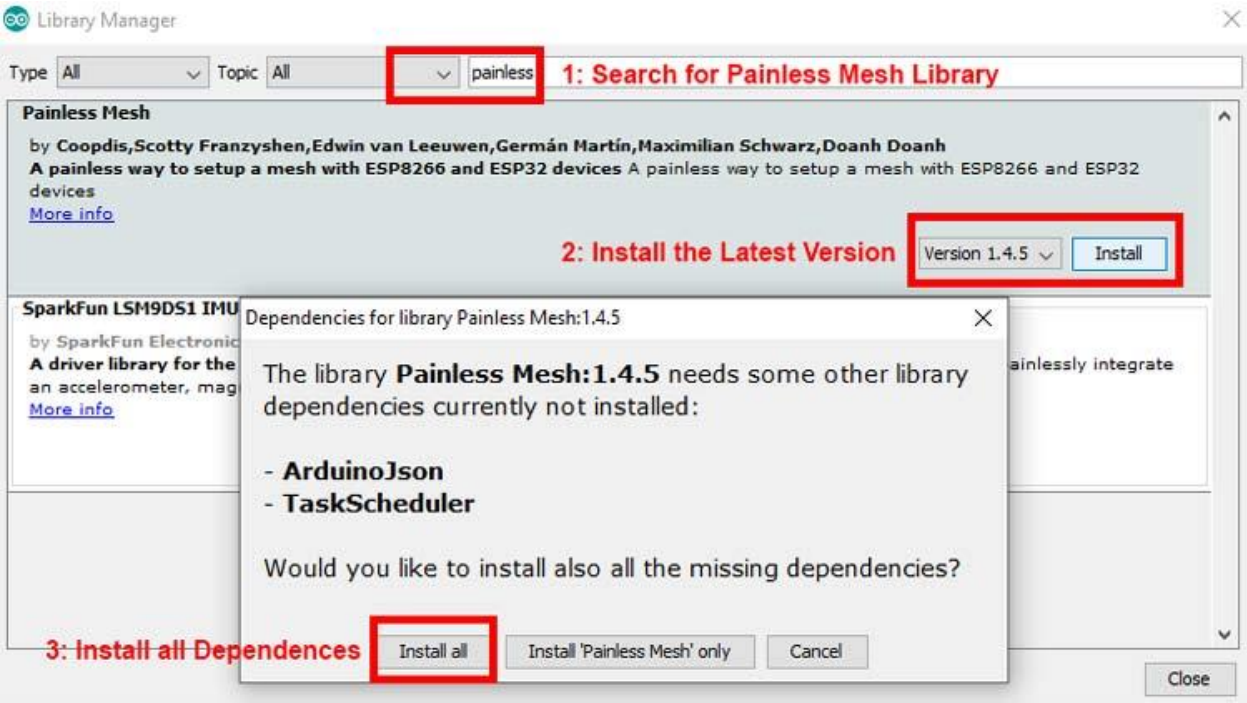


For this circuit, we have connected two BME280 sensors with the ESP32 board, we have connected a DHT22 sensor to one of the ESP8266 board, and we have connected the DS18B20 sensor with another ESP8266 board. We have previously used all these sensors in different projects individually. Here, we will be using them on different NodeMCU/ESP32 boards and then connect them all through an **ESP Mesh network.** An image for the hardware setup is shown below.

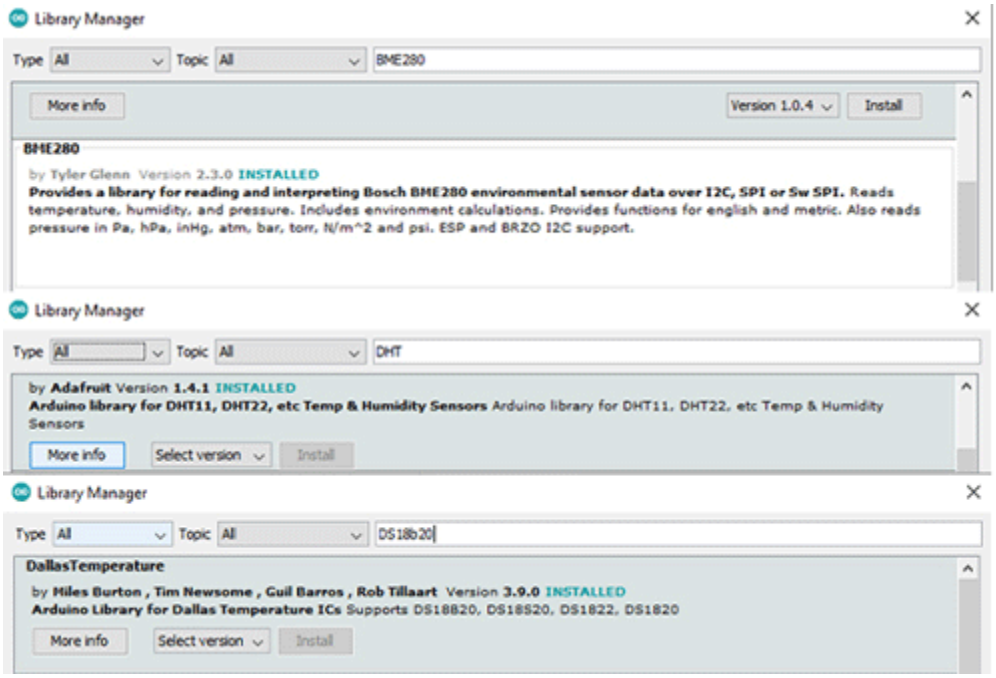


Programming the ESP8266 and ESP32 for Mesh Networking

For this article, we are going to use Arduino IDE to program the ESP32 and the ESP8266 boards. Here, we will use the **Painless Mesh library** to construct our mesh network. To install the library, go to **Sketch->Include Library->Manage Libraries** and search for the **painlessMesh. O**nce done, just click install and the library will be installed in the Arduino IDE. As you can see in the image below, once you click Install, this library asks you to install additional dependencies. You need to install those for the library to work.



As you already know from the hardware section, we are going to use a DS18B20 Sensor, A DHT22 sensor, and two BME 280 sensors. We need to install all of those, and it can be simply done with the board manager method.



You can also download these libraries from the link given below. Once we have downloaded and installed all the required libraries, we can move on to creating our code.

- Download Painless Mesh Library
- Download DHT Sensor Library
- Download BME280 Library
- Download DallasTemperature Library

Note: The code explanation you see next is the code used in all four boards. We designed the code so that we can tweak it a little and we can upload it to any of our ESP boards despite the fact that it's an ESP32 or an ESP8266 board.

Upload the Code to ESP32 Board in which a BME280 Sensor is Connected:

As you can see in the hardware schematic, we have connected a BME280 sensor. For that, you need to uncomment the **macro for the BME_280** sensor and give it a unique Node Name. In our case, we have used **Node_1** and **Node_2** for our two ESP32 boards to which we have attached the BME280 sensor

```
#define BME_280
//#define DHT22
//#define DS18B20
//#define ENABLE_LOG
String nodeName = "NODE_1";
```

Upload the Code to ESP8266 Board in Which A DHT Sensor is Connected:

In one of our ESP8266 boards, we have a DHT22 sensor and in another one, we have a DS18B20 sensor. To upload the code to the DHT22 board, we have to follow the same process. First, we

uncomment the macro for DHT22 and then comment out the macro for BME280 and upload the code to the board to which we have connected the DHT 22 sensor.

```
//#define BME_280
#define DHT22
//#define DS18B20
//#define ENABLE_LOG
String nodeName = "NODE_3";
```

Upload the Code to ESP8266 Board in Which A DS18B20 Sensor is Connected:

The process stays exactly the same for this board also. We uncomment the macro for the DS18B20 sensor and we comment on other macros.

```
//#define BME_280
//#define DHT22
#define DS18B20
//#define ENABLE_LOG
String nodeName = "NODE_3";
```

Finally, to enable or disable other log statements, you can uncomment the ENABLE_LOG macro. Now that we have an understanding of how the code works, we can move further and explain the code.

We will start our code by including the **painlessMesh** library and the **Arduino_JSON** library.

```
#include <painlessMesh.h>
#include <Arduino_JSON.h>
```

Next, we define some macros which we will use to enable or disable the parts of our code. This is needed because not all the nodes use the same sensor type and. So for including or excluding parts of our code, we can fit four different codes into a single file.

```
//#define BME_280
#define DHT22
//#define DS18B20
//#define ENABLE_LOG
```

Next, we define a String type variable nodeName. This will be used to uniquely identify the nodes in the network. Along with that we also define the float type variable to store the temperature, humidity, and barometric pressure data.

```
String nodeName = "NODE_4"; // Name needs to be unique
float temp(NAN), hum(NAN), pres(NAN);
```

From this step onwards, we will be using **#ifdef** and **#endif** macros to include or exclude parts of our code. The first section of the code is for the BME280 sensor. As said earlier, we will start by **#ifdef BME_280** statement. Next, we define all the required libraries for the BME280 sensor. The BME sensor also uses the wire library, so we define that as well. Next, we make a **BME280I2C** object **bme.** Next, using the **scope resolution operator,** we access the variable of the class and we finish that off with the endif statement.

```
#ifdef BME_280
#include <BME280I2C.h>
#include <Wire.h>
BME280I2C bme;
BME280::TempUnit tempUnit(BME280::TempUnit_Celsius);
BME280::PresUnit presUnit(BME280::PresUnit_Pa);
#endif
```

We do the same for the DHT library too. We start with the **#ifdef DHT22** statement followed by including the **DHT.h** library. Next, we define the PIN for DHT, and add a prototype for DHT22. Thereafter, we create an object by passing the above-defined statements. And we finish off with the **#endif** statement.

```
#ifdef DHT22
#include "DHT.h"
#define DHTPIN 4
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
#endif
```

We also do the same for the DS18B20 sensor. We start with the **#ifdef DS18B20** statement.

As the DS18B20 sensor requires **OneWire** library, we include that along with the **DallasTemperature** library. Next, we define the pin for the sensor and create an OneWire object by passing the pin variable. Next, we pass the address of the OneWire object to the DallasTemperature object by making a new DallasTemperature object.

```
#ifdef DS18B20
#include <OneWire.h>
#include <DallasTemperature.h>
const int oneWireBus = 4;
OneWire oneWire(oneWireBus);
DallasTemperature ds18b20(&oneWire);
#endif
```

Next, we define the Wi-Fi credentials along with the port number. This credential and port number should stay the same for all nodes within the network.

```
#define   MESH_PREFIX "whateverYouLike"
#define   MESH_PASSWORD "somethingSneaky"
#define   MESH_PORT 5555
```

Next, we make three instances. One is for the **Scheduler,** another is for the **painlessMesh** and the final one is for the JSON library **JSONVar**

```
Scheduler userScheduler; // to control your task
painlessMesh  mesh;
JSONVar myVar;
```

Next, we have created a task that is like a thread that always runs and calls a function after some time. The defined task below will be used to send a broadcast message to all the nodes. The task instantly takes three parameters. First one defines how often the task will call the function, Next, it asks for the life of the task, and finally, it takes a pointer to the calling function.

```
Task taskSendMessage( TASK_SECOND * 1 , TASK_FOREVER, &sendMessage );
```

Next, we have our calling function **sendMessage().** This function calls another function that returns a string of JSON. As the name implies, the sendMessage task is used to send the message to all the nodes.

```
void sendMessage() {
  String msg = construnct_json();
  mesh.sendBroadcast( msg );
  taskSendMessage.setInterval( random( TASK_SECOND * 1, TASK_SECOND * 5 ));
}
```

Next, we have our **receivedCallback()** function. Whenever a new message arrives, this function gets a call. If you want to do something with the received message, you need to tweak this function to get your job done. This function takes two arguments - the node id and message as a pointer.

```
void receivedCallback( uint32_t from, String &msg ) {
  Serial.printf("startHere: Received from %u msg=%s\n", from, msg.c_str());
}
```

Next, we have our **newConnectionCallback()** function. This function gets a call whenever there is a new device is added to the network, and it sends a print statement to the serial monitor.

```
void newConnectionCallback(uint32_t nodeId) {
  Serial.printf("--> startHere: New Connection, nodeId = %u\n", nodeId);
}
```

Next, we have our **nodeTimeAdjustedCallback()**. This callback function takes care of all the timing necessities required by the painless mesh.

```
void nodeTimeAdjustedCallback(int32_t offset) {
  Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(), offset);
}
```

Next, we have our *setup()* function. In the setup, we initialize the serial and print the node name. This is helpful because once all the nodes are programed, we can identify the nodes easily with the serial monitor. We also have the **setDebugMsgTypes( )** class of the mesh object that logs any ERROR | STARTUP messages. Next, we initialize the mesh by-passing the SSID, Password, and Port Number to the *init()* function. Please keep in mind that these functions also need the pointer to the Scheduler in order to work properly.

```
Serial.begin(115200);
Serial.println(nodeName);
mesh.setDebugMsgTypes( ERROR | STARTUP );  // set before init() so that you can see s
tartup messages
mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT );
```

Now, we will initialize all the callback functions which we have discussed above. These callback functions will be called whenever a certain task is needed to be performed.

```
 mesh.onReceive(&receivedCallback);
 mesh.onNewConnection(&newConnectionCallback);
```

```
 mesh.onChangedConnections(&changedConnectionCallback);
 mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
```

Now we add the task to the task scheduler, and enabled it with the help of the taskSendMessage.enable() method. when it's executed, different tasks start to run simultaneously in the background.

```
userScheduler.addTask( taskSendMessage );
taskSendMessage.enable();
```

Next, in the setup section, we have all the necessary **ifdef** and **endif** macros which are used to initialize different sensors depending upon the requirement. First, we will config the BME280 sensor. The code below initializes the BME280 sensor and checks for the version of the sensor as the BME280 sensor comes with many different versions.

```
#ifdef BME_280
  Wire.begin();
  while (!bme.begin())
  {
    Serial.println("Could not find BME280 sensor!");
    delay(1000);
  }
  // bme.chipID(); // Deprecated. See chipModel().
  switch (bme.chipModel())
  {
    case BME280::ChipModel_BME280:
      Serial.println("Found BME280 sensor! Success.");
      break;
    case BME280::ChipModel_BMP280:
      Serial.println("Found BMP280 sensor! No Humidity available.");
      break;
    default:
```

```
      Serial.println("Found UNKNOWN sensor! Error!");
  }
#endif
```

Finally, we have configured the DHT22 and the DS18B20 sensor with the help of the **ifdef** and **#endif** method. And this marks the end of the *setup()* function.

```
#ifdef DHT22
  Serial.println(F("DHTxx Begin!"));
  dht.begin();
#endif
#ifdef DS18B20
  ds18b20.begin();
  Serial.println(F("DS18B20 Begin!"));
#endif
```

Next, we have our loop. In this code, the loop dose does not do much, it just updates the mesh with the help of **mesh.update()** method. It takes care of all the tasks. These tasks will not work if this update method is not present.

```
void loop()
{
  mesh.update();
  // construnct_json();
}
```

Next, we have our final function. This function constructs the JSON string and returns it to the calling function. In this function, we start by calling the **bme.read()** method and we pass all the predefined variables that get updated with new values. Next, we divide the pressure value by 100 because we want to convert it into millibar. Next, we define our JSON array and put the Sensor name, the node name, the temperature, pressure value, and return the values using

the **JSON.stringify()** function. Finally, we define another ifdef and endif macros to enable or disable log parameters.

```
String construnct_json()
{
#ifdef BME_280
  bme.read(pres, temp, hum, tempUnit, presUnit);
  pres = pres / 100;
  myVar["Sensor Type"] = "BME280";
  myVar["Node Name"] = nodeName;
  myVar["Temperature"] = serialized(String(temp, 2));
  myVar["pres"] = serialized(String(pres, 2));
#ifdef ENABLE_LOG
  Serial.println(JSON.stringify(myVar));
#endif
  return JSON.stringify(myVar);
#endif
```

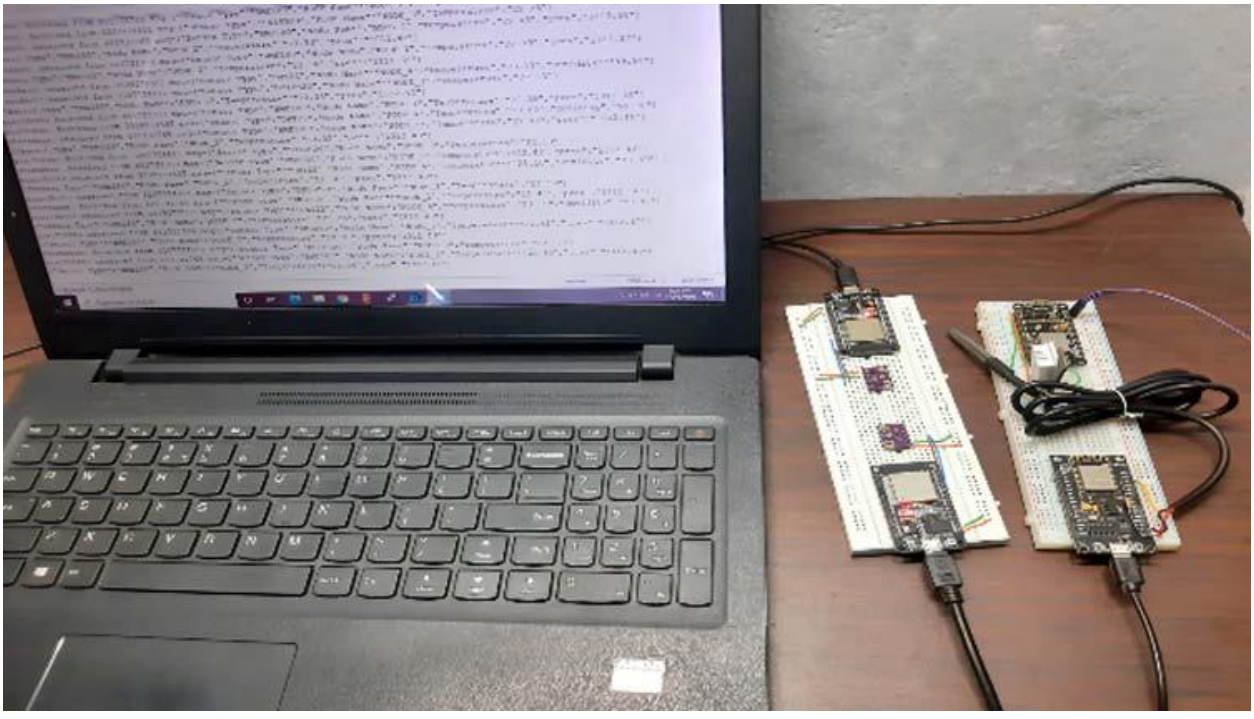
Finally, we do the same for the DHT22 and the DS18B20 sensor code.

```
#ifdef DHT22
  temp = dht.readTemperature();
  hum = dht.readHumidity();
  myVar["Sensor Type"] = "DHT22";
  myVar["Node Name"] = nodeName;
  myVar["Temperature"] = serialized(String(temp));
  myVar["Humidity"] = serialized(String(hum));
#ifdef ENABLE_LOG
  Serial.println(JSON.stringify(myVar));
#endif
  return JSON.stringify(myVar);
#endif
#ifdef DS18B20
```

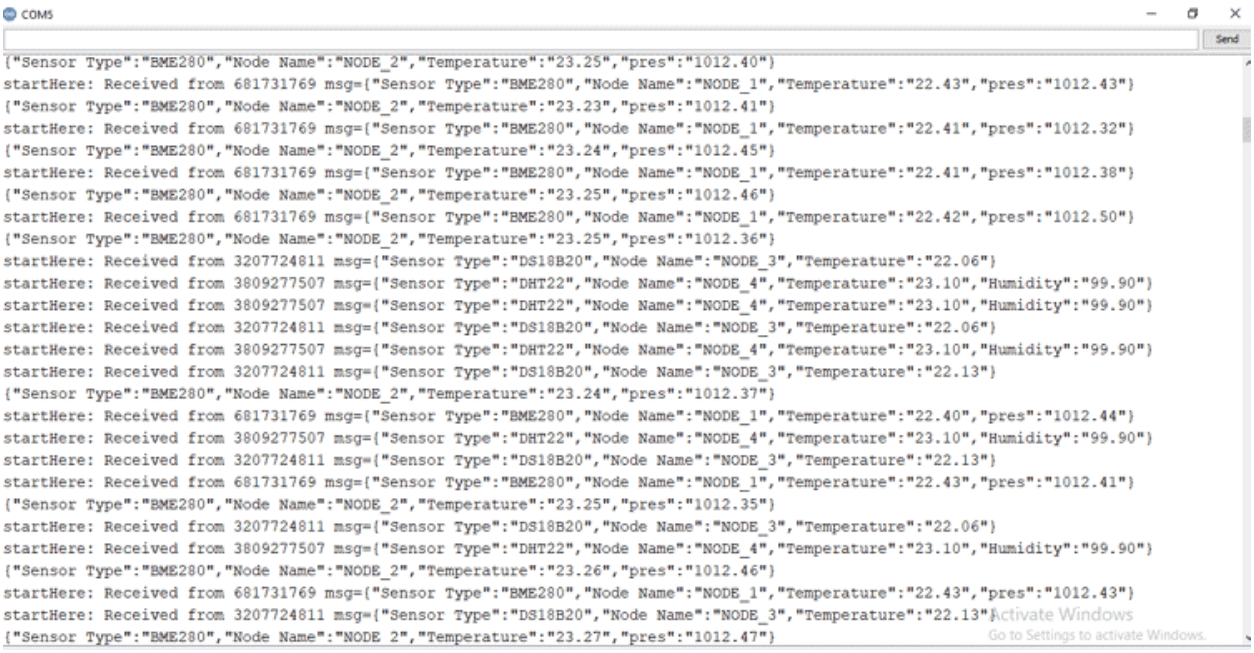
```
  ds18b20.requestTemperatures();
  temp = ds18b20.getTempCByIndex(0);
  myVar["Sensor Type"] = "DS18B20";
  myVar["Node Name"] = nodeName;
  myVar["Temperature"] = serialized(String(temp));
#ifdef ENABLE_LOG
  Serial.println(JSON.stringify(myVar));
#endif
  return JSON.stringify(myVar);
#endif
}
```

Now, as the coding process is finished; we comment or uncomment the defined macros on top; according to our board which we are going to upload. Next, we give each node a unique name and we simply upload the code. If everything is all right, the code will compile and upload properly without any error.

ESP8266 and ESP32 based Mesh Network - Testing



The test setup for the ESp8266 and esp32 based mesh network is shown below. As you can see in the above image, I have connected power to all the esp modules and you can also see the output data on the screen of the laptop. A screenshot of the serial monitor windows is shown below.



In the above window, you can see that we are easily receiving the data from all four sensors.

The complete working of the project can also be found in the video linked below. Hope you enjoyed the project and found it interesting to build your own. If you have any questions, please leave them in the comment section below. You can also write all your technical questions on forums to get them answered or to start a discussion.

Code

```
#include <painlessMesh.h>
#include <Arduino_JSON.h>
//#define BME_280
#define DHT22
//#define DS18B20
//#define ENABLE_LOG
```

```
String nodeName = "NODE_4"; // Name needs to be unique
float temp(NAN), hum(NAN), pres(NAN);
//########################## Init_BME280 ##########################
#ifdef BME_280
#include <BME280I2C.h>
#include <Wire.h>
BME280I2C bme;
BME280::TempUnit tempUnit(BME280::TempUnit_Celsius);
BME280::PresUnit presUnit(BME280::PresUnit_Pa);
#endif
//_____________________________ End of _BME280 _____________________________
//########################## Init_DHT22 ##########################
#ifdef DHT22
#include "DHT.h"
#define DHTPIN 4
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
#endif
//_____________________________ End of DHT22 _____________________________
//########################## Init_Ds18B20 ##########################
#ifdef DS18B20
#include <OneWire.h>
#include <DallasTemperature.h>
const int oneWireBus = 4;
OneWire oneWire(oneWireBus);
DallasTemperature ds18b20(&oneWire);
#endif
//_____________________________ End of DHT22 _____________________________
#define   MESH_PREFIX     "whateverYouLike"
#define   MESH_PASSWORD   "somethingSneaky"
#define   MESH_PORT       5555
Scheduler userScheduler; // to control your personal task
painlessMesh  mesh;
JSONVar myVar;
```

```
void sendMessage() {
  String msg = construnct_json();
  mesh.sendBroadcast( msg );
  taskSendMessage.setInterval( random( TASK_SECOND * 1, TASK_SECOND * 5 ));
}
Task taskSendMessage( TASK_SECOND * 1 , TASK_FOREVER, &sendMessage );
// Needed for painless library
void receivedCallback( uint32_t from, String &msg ) {
  Serial.printf("startHere: Received from %u msg=%s\n", from, msg.c_str());
}
void newConnectionCallback(uint32_t nodeId) {
  Serial.printf("--> startHere: New Connection, nodeId = %u\n", nodeId);
}
void changedConnectionCallback() {
  Serial.printf("Changed connections\n");
}
void nodeTimeAdjustedCallback(int32_t offset) {
  Serial.printf("Adjusted time %u. Offset = %d\n", mesh.getNodeTime(), offset);
}
void setup()
{
  Serial.begin(115200);
  Serial.println(nodeName);
//  mesh.setDebugMsgTypes( ERROR | STARTUP | MESH_STATUS | CONNECTION | SYNC | COMMUN
ICATION | GENERAL | MSG_TYPES | REMOTE ); // all types on
  mesh.setDebugMsgTypes( ERROR | STARTUP );  // set before init() so that you can see
startup messages
  mesh.init( MESH_PREFIX, MESH_PASSWORD, &userScheduler, MESH_PORT );
  mesh.onReceive(&receivedCallback);
  mesh.onNewConnection(&newConnectionCallback);
  mesh.onChangedConnections(&changedConnectionCallback);
  mesh.onNodeTimeAdjusted(&nodeTimeAdjustedCallback);
  userScheduler.addTask( taskSendMessage );
  taskSendMessage.enable();
```

```
#ifdef BME_280
  Wire.begin();
  while (!bme.begin())
  {
    Serial.println("Could not find BME280 sensor!");
    delay(1000);
  }
  // bme.chipID(); // Deprecated. See chipModel().
  switch (bme.chipModel())
  {
    case BME280::ChipModel_BME280:
      Serial.println("Found BME280 sensor! Success.");
      break;
    case BME280::ChipModel_BMP280:
      Serial.println("Found BMP280 sensor! No Humidity available.");
      break;
    default:
      Serial.println("Found UNKNOWN sensor! Error!");
  }
#endif
#ifdef DHT22
  Serial.println(F("DHTxx test!"));
  dht.begin();
#endif
#ifdef DS18B20
  ds18b20.begin();
#endif
}
void loop()
{
  mesh.update();
  // construnct_json();
}
String construnct_json()
```

```
{
#ifdef BME_280
  bme.read(pres, temp, hum, tempUnit, presUnit); // update with new values
  pres = pres / 100;
  myVar["Sensor Type"] = "BME280";
  myVar["Node Name"] = nodeName;
  myVar["Temperature"] = serialized(String(temp, 2)); // serialized need to conver fl
ot values
  myVar["pres"] = serialized(String(pres, 2));// serialized need to conver flot value
s
#ifdef ENABLE_LOG
  Serial.println(JSON.stringify(myVar)); //stringify converts the arry to a string
#endif
  return JSON.stringify(myVar);
#endif
#ifdef DHT22
  temp = dht.readTemperature();
  hum = dht.readHumidity();
  myVar["Sensor Type"] = "DHT22";
  myVar["Node Name"] = nodeName;
  myVar["Temperature"] = serialized(String(temp));
  myVar["Humidity"] = serialized(String(hum));
#ifdef ENABLE_LOG
  Serial.println(JSON.stringify(myVar));
#endif
  return JSON.stringify(myVar);
#endif
#ifdef DS18B20
  ds18b20.requestTemperatures();
  temp = ds18b20.getTempCByIndex(0);
  myVar["Sensor Type"] = "DS18B20";
  myVar["Node Name"] = nodeName;
  myVar["Temperature"] = serialized(String(temp));
#ifdef ENABLE_LOG
```

```
  Serial.println(JSON.stringify(myVar));
#endif
  return JSON.stringify(myVar);
#endif
}
```

```
\#include
String apiKey = "H38TEGNC0XKW43BB";
//Enter your Write API key  from ThingSpeak const char *ssid =  "how2electronics";
// replace with your wifi  ssid and wpa2 key const char *pass =  "alhabibi";

const char* server = "api.thingspeak.com";
#define DHTPIN 0 //pin where the dht11 is connected

DHT dht(DHTPIN, DHT11);
WiFiClient client;

void setup()
{
Serial.begin(115200);
delay(10);
dht.begin();

Serial.println("Connecting to ");
Serial.println(ssid);

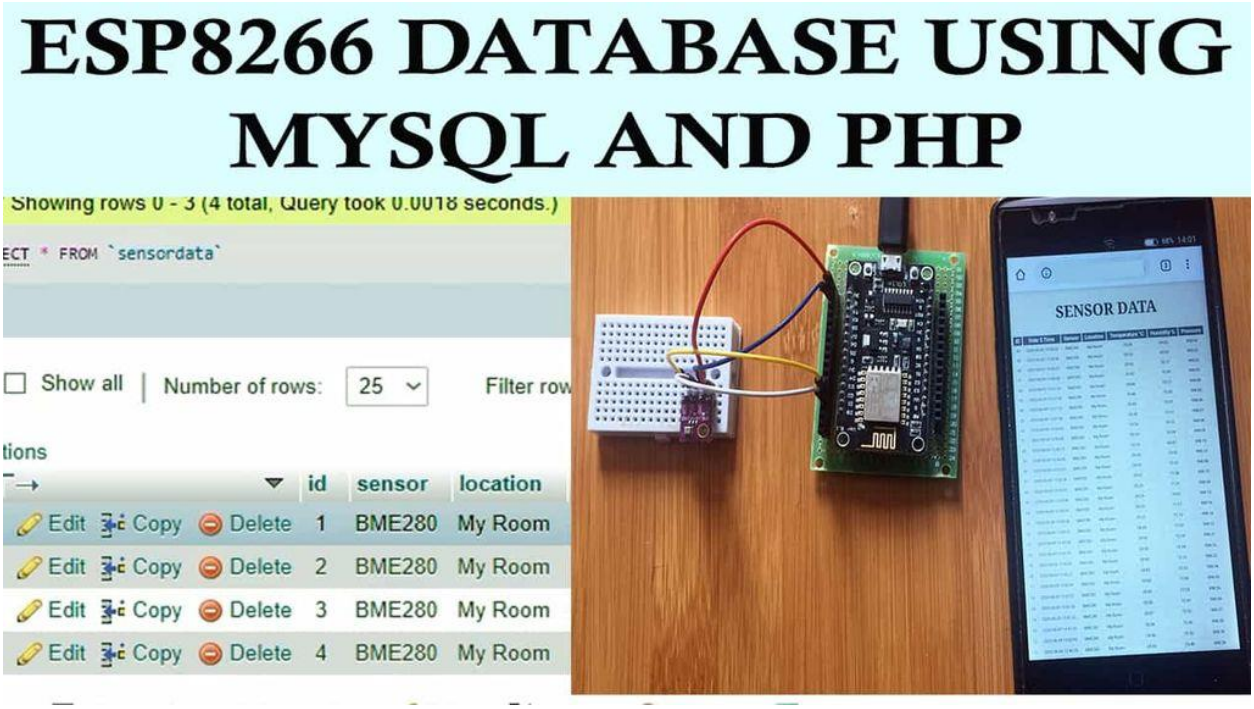
WiFi.begin(ssid, pass);

while (WiFi.status() != WL_CONNECTED)
{
delay(500);
Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

}
void loop()
{
float h = dht.readHumidity();
float t = dht.readTemperature();
if (isnan(h) || isnan(t))
{
Serial.println("Failed to read from DHT sensor!");
return;
}
if (client.connect(server,80))  //"184.106.153.149" or api.thingspeak.com
{
String postStr = apiKey;
postStr +="&field1=";
postStr += String(t);
postStr +="&field2=";
postStr += String(h);
postStr += "\r\n\r\n";
client.print("POST /update HTTP/1.1\n");
client.print("Host: api.thingspeak.com\n");
client.print("Connection: close\n");
client.print("X-THINGSPEAKAPIKEY:  "+apiKey+"\n");
client.print("Content-Type:  application/x-www-form-urlencoded\n");
client.print("Content-Length: ");
client.print(postStr.length());
client.print("\n\n");
client.print(postStr);
Serial.print("Temperature: ");
Serial.print(t);
Serial.print(" degrees Celcius, Humidity:  ");
Serial.print(h);
```

```
Serial.println("%. Send to Thingspeak.");
}
client.stop();
Serial.println("Waiting...");
// thingspeak needs minimum 15 sec delay between updates
delay(1000);
}
```

ow to Connect ESP8266 TO MYSQL Database Using PHP and Arduino IDE.



Often times we need to keep track of data from sensors installed in a given place even if we are far away from that place. This can be achieved over the internet where the sensor readings can be observed over a webpage.

In this tutorial I am going to show you how the ESP3266 Nodemcu can be used as a client to send data from sensors to a MYSQL database using PHP script. This data can then be accessed and displayed on a webpage anywhere around the world as long as there is internet connection.

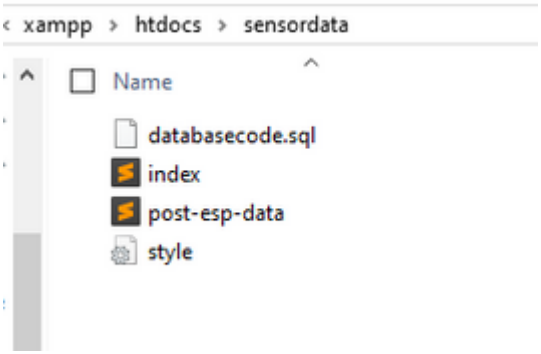
To be able to access the information in the database on a website you need to have a domain name and a hosting account. However, since I am giving a demonstration of how this works, I am going to be using a local host with the help of XAMPP web server to host MySQL database locally on your Windows PC.

In case you need more guidance on how to install XAMPP local server please visit the link below for a step-by-step guide;

- How to Install XAMPP on your Local Computer.

Arranging project files in the XAMPP web server

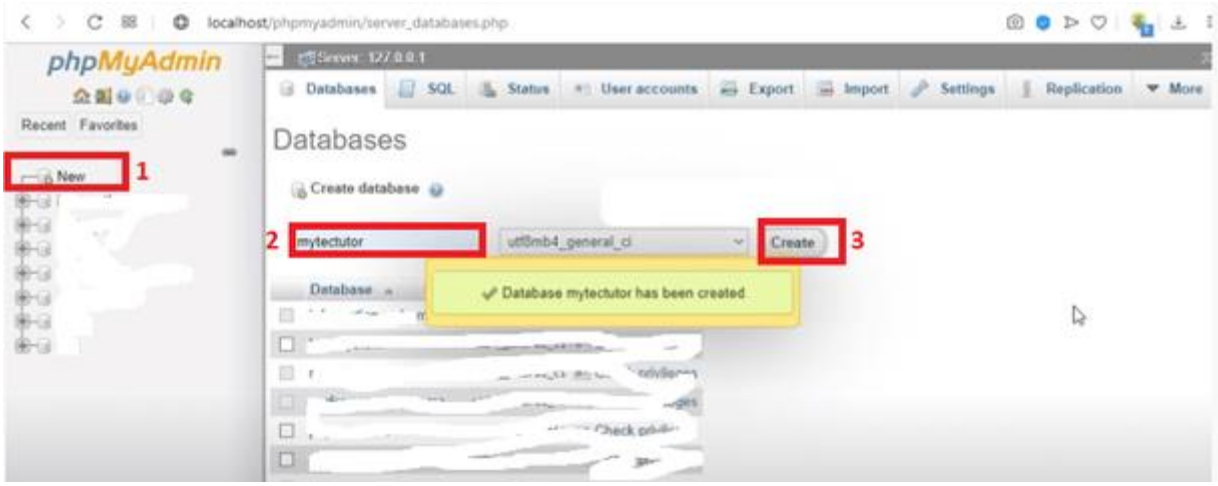
Since we are going to be using XAMPP, you need to have a folder to hold all project files for example in this case I'll place all the project files inside sensordata folder in the path **c: xampp \ htdocs \ sensordata** as shown below.



The files include;
databasecode.sql: for creating a table in the MSQL database.
index.php: PHP script that runs to display the database content onto a web page.
post-esp-data.php : contains the PHP script that is responsible for receiving incoming requests from the ESP8266 and inserting the data into a MySQL database.
style.css: for modifying the appearance of our web page.

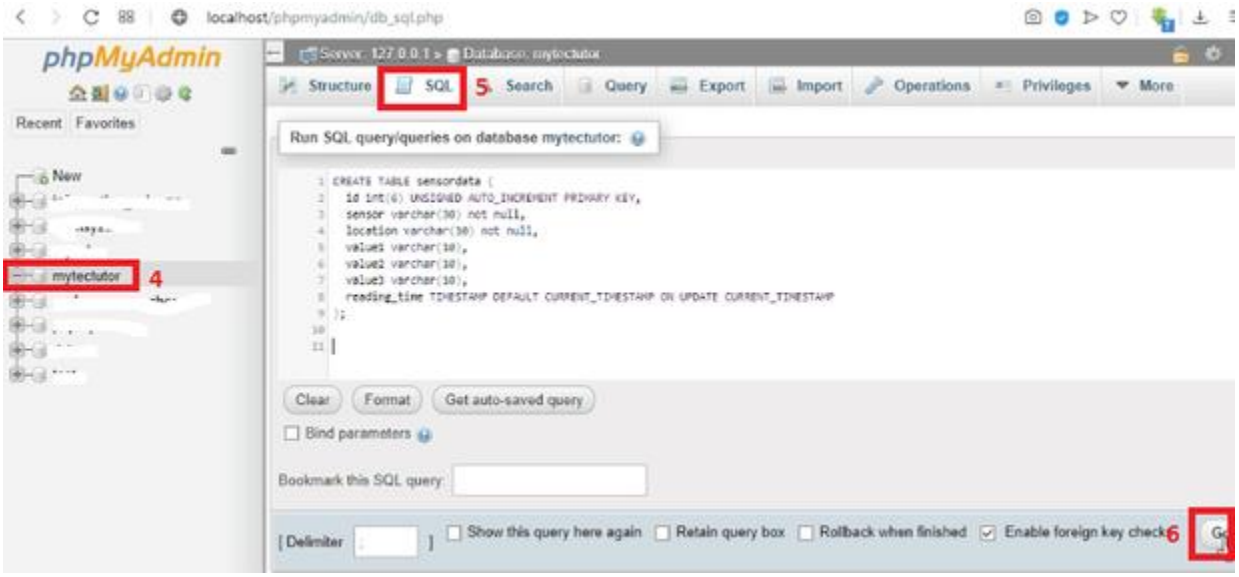
Preparing the MySQL Database and adding SQL table in the Database.

To access the MYSQL database use the URL: ***localhost/phpmyadmin/*** Then to create a new database you click **New > Enter database name> Create** as illustrated below where I created a database named mytectutor;



To create a new SQL table in the database, select the new database created (mytectutor) and click SQL tab and insert the SQL query below.

```
CREATE TABLE sensorData (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
sensor VARCHAR(30) NOT NULL,
location VARCHAR(30) NOT NULL,
value1 VARCHAR(10),
value2 VARCHAR(10),
value3 VARCHAR(10),
reading_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```



Play

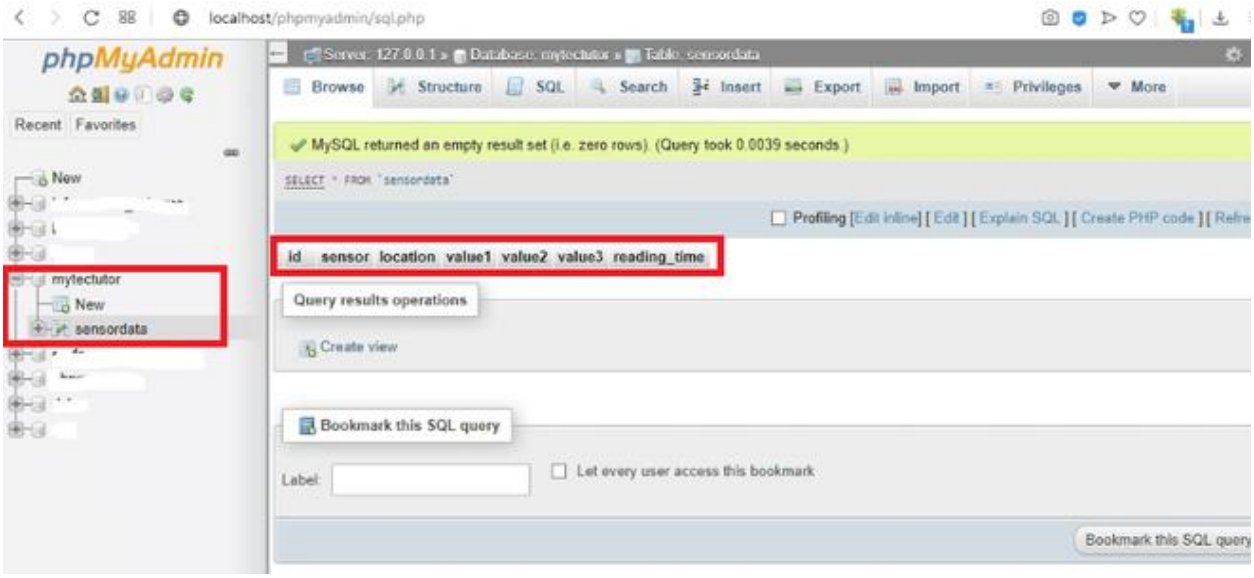
Unmute

Loaded: 98.82%

Remaining Time -0:34

ESP8266 and MYSQL Database

Now you can see the new table called **sensordata** in the **mytectutor** database as shown in the figure below;



Inserting data in the MySQL Database using PHP script.

PHP script for receiving incoming requests from the ESP8266 and inserting the data into a MySQL database is in the **post-esp-data.php** file. You can download this script from the link provided below;

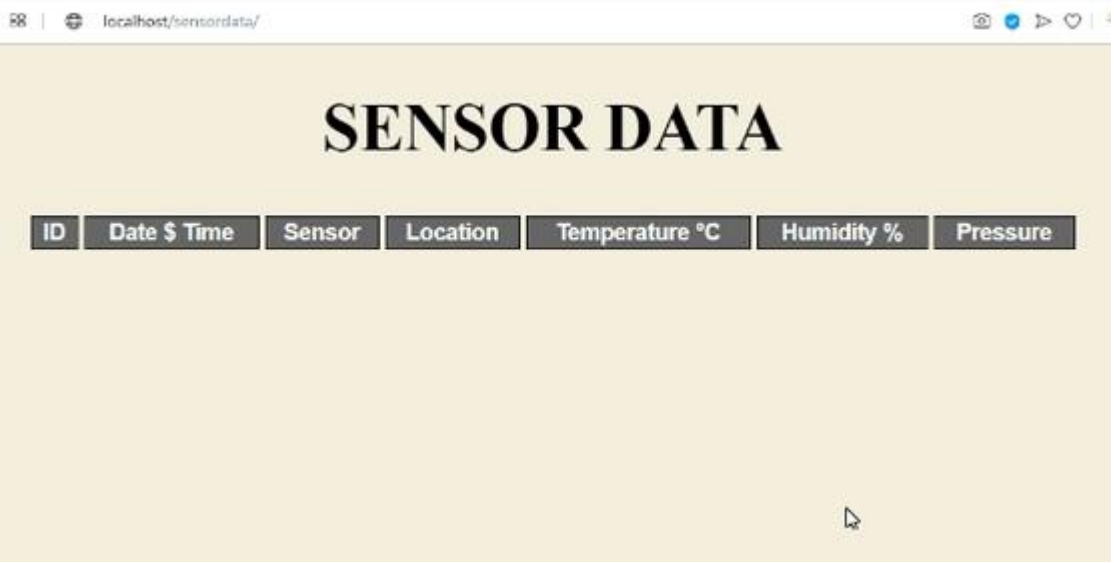
You need to take note of the values below:

```
$servername = "localhost";
$dbname = "Replace with your database name";
$username = "Replace with your database username";
$password = "Replace with your database password";
$api_key_value = "tPmAT5Ab3j7F9";
```

Displaying the content in the Database on the Web page.

The script for displaying the database content on a web page is in the **index.php** file. Don't forget to replace the values of $dbname,

$username and $password with those corresponding to your database. Now the web page can be accessed from the URL: *http://localhost/sensordata* and will appear as shown below.

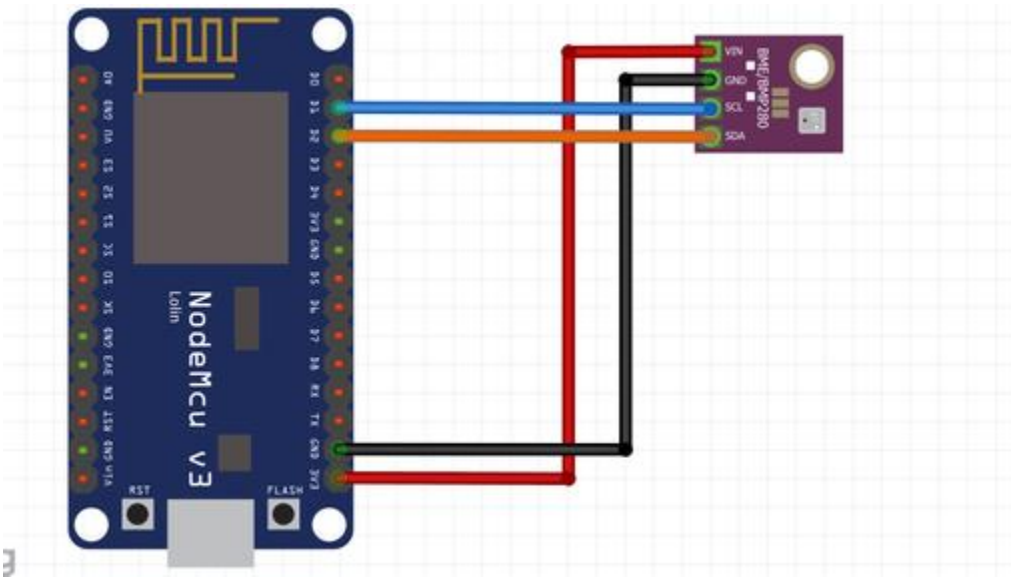


The web page displays the table but is currently empty because there is no data in the database. The appearance of the webpage is determined by the script in the style.css file. Next we shall use the ESP8266 Nodemcu to insert sensor readings into the database.

Download project files: ESP8266 MYSQL Database

Connecting the ESP8266 Nodemcu with BME 280 sensor.

To insert data into the database, we are going to use the ESP8266 Nodemcu and BME280 sensor so that we can insert temperature, humidity, pressure readings into our database every 30 seconds. The setup is as shown in the schematic below.



Code for ESP8266 Nodemcu with BME 280 sensor.

```
#ifdef ESP32
  #include <WiFi.h>
  #include <HTTPClient.h>
#else
  #include <ESP8266WiFi.h>
  #include <ESP8266HTTPClient.h>
  #include <WiFiClient.h>
#endif

#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

// Replace with your network credentials
const char* ssid     = "replace with your WiFi ssid";
const char* password = "replace with your WiFi password";

// REPLACE with your Domain name and URL path or IP address
with path
const char* serverName = "http://ip_address/sensordata/post-
esp-data.php";
```

```
// Keep this API Key value to be compatible with the PHP code
provided in the project page.
// If you change the apiKeyValue value, the PHP file /post-esp-
data.php also needs to have the same key
String apiKeyValue = "tPmAT5Ab3j7F9";

String sensorName = "BME280";
String sensorLocation = "My Room";

#define SEALEVELPRESSURE_HPA (1013.25)
Adafruit_BME280 bme;  // I2C
void setup() {
  Serial.begin(115200);

  WiFi.begin(ssid, password);
  Serial.println("Connecting");
  while(WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.print("Connected to WiFi network with IP Address: ");
  Serial.println(WiFi.localIP());

  // (you can also pass in a Wire library object like &Wire2)
  bool status = bme.begin(0x76);
  if (!status) {
    Serial.println("Could not find a valid BME280 sensor, check
wiring or change I2C address!");
    while (1);
  }
}
void loop() {
  //Check WiFi connection status
  if(WiFi.status()== WL_CONNECTED){
    HTTPClient http;

    // Your Domain name with URL path or IP address with path
    http.begin(serverName);
```

```
    // Specify content-type header
    http.addHeader("Content-Type", "application/x-www-form-
urlencoded");

    // Prepare your HTTP POST request data
    String httpRequestData = "api_key=" + apiKeyValue +
"&sensor=" + sensorName
                           + "&location=" + sensorLocation +
"&value1=" + String(bme.readTemperature())
+ "&value2=" + String(bme.readHumidity()) + "&value3=" +
String(bme.readPressure()/100.0F) + "";
    Serial.print("httpRequestData: ");
    Serial.println(httpRequestData);

    // You can comment the httpRequestData variable above
    // then, use the httpRequestData variable below (for
testing purposes without the BME280 sensor)
    //String httpRequestData =
"api_key=tPmAT5Ab3j7F9&sensor=BME280&location=Office&value1=24.
75&value2=49.54&value3=1005.14";
 // Send HTTP POST request
    int httpResponseCode = http.POST(httpRequestData);

    // If you need an HTTP request with a content type:
text/plain
    //http.addHeader("Content-Type", "text/plain");
    //int httpResponseCode = http.POST("Hello, World!");

    // If you need an HTTP request with a content type:
application/json, use the following:
    //http.addHeader("Content-Type", "application/json");
    //int httpResponseCode =
http.POST("{\"value1\":\"19\",\"value2\":\"67\",\"value3\":\"78
\"}");
  if (httpResponseCode>0) {
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
    }
    else {
      Serial.print("Error code: ");
```

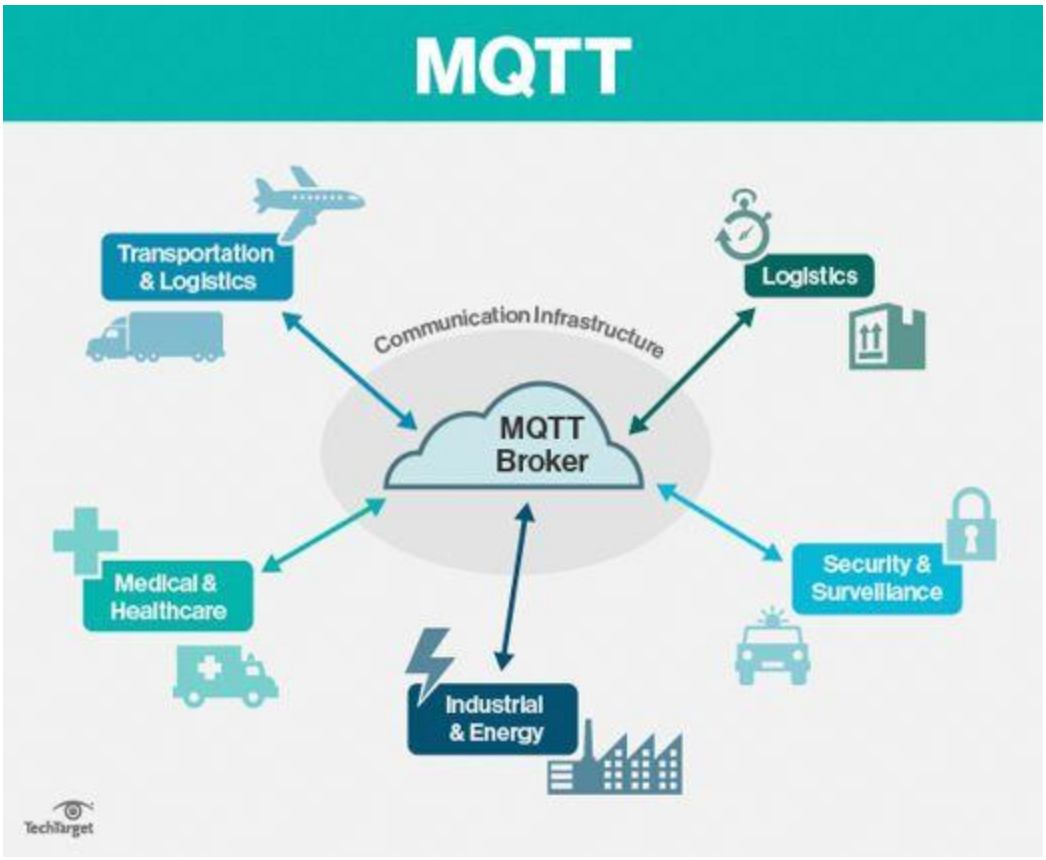
```
      Serial.println(httpResponseCode);
    }
    // Free resources
    http.end();
  }
  else {
    Serial.println("WiFi Disconnected");
  }
  //Send an HTTP POST request every 15 seconds
  delay(15000);
}
```

MQTT và ESP8266

MQTTESP8266

MQTT là gì ? Cách sử dụng MQTT như thế nào ? Các ví dụ về MQTT với ESP8266 được thực hiện ra sao ? Trong bài viết này mình sẽ trình bày một số khái niệm và các bước thực hiện MQTT với ví dụ minh họa cơ bản trên ESP8266 cho mọi người cùng tham khảo.

Trước khi bắt đầu mình cùng xem qua một hình ảnh cơ bản về MQTT



Với hình ảnh trên thì ta có thể thấy MQTT Broker là một trạm trung chuyển, nói đơn giản dễ hiểu thì coi như nó là một trung tâm thông tin, có nhiệm vụ nhận thông tin từ rất nhiều các thiết bị khác nhau và chuyển các thông tin này tới từng thiết bị riêng biệt.

Sẽ có một câu hỏi được đặt ra là Broker MQTT này sẽ lấy ở đâu ra ? Chúng ta sẽ có 2 cách

- Tự tạo Broker MQTT trên máy tính, raspberry, server,.. tùy vào điều kiện hiện có: cách này tốn kém thêm chi phí, phức tạp, không dành cho người mới bắt đầu.
- Sử dụng các dịch vụ MQTT broker có sẵn: cách này dễ thở hơn chút, sử dụng cái sẵn có, thao tác cấu hình lấy thông tin để sử dụng, phù hợp với những người muốn thử nghiệm, tham khảo, tận dụng cái có sẵn. Trong bài viết này mình sẽ sử dụng cách này.

Để hiểu hơn một chút về cách thức hoạt động của MQTT Broker và ESP8266 ta sẽ đi vào ví dụ cụ thể

Chuẩn bị

Phần cứng

- ESP8266 (NodeMCU)

Phần mềm

- Arduino IDE
- Thư viện Pubsubclient
- MQTTlens cho Chrome

Cách kết nối nối cũng như dùng Arduino cho ESP8266 và các vấn đề liên quan các bạn tham khảo ở đây giúp mình nếu các bạn mới bắt đầu tìm hiểu.

Thực hành 1

Sau khi chuẩn bị đầy đủ đồ nghề thì mình bắt tay vào thử chút thôi. Về cách làm mình sẽ dùng ESP8266 (NodeMCU) làm Client để kết nối lên một dịch vụ MQTT Broker là CloudMQTT với 2 nhiệm vụ chính, nhiệm vụ 1 là tham khảo từ Phan Tiến Sang, nhiệm vụ 2 là thay đổi 1 chút để điều khiển LED

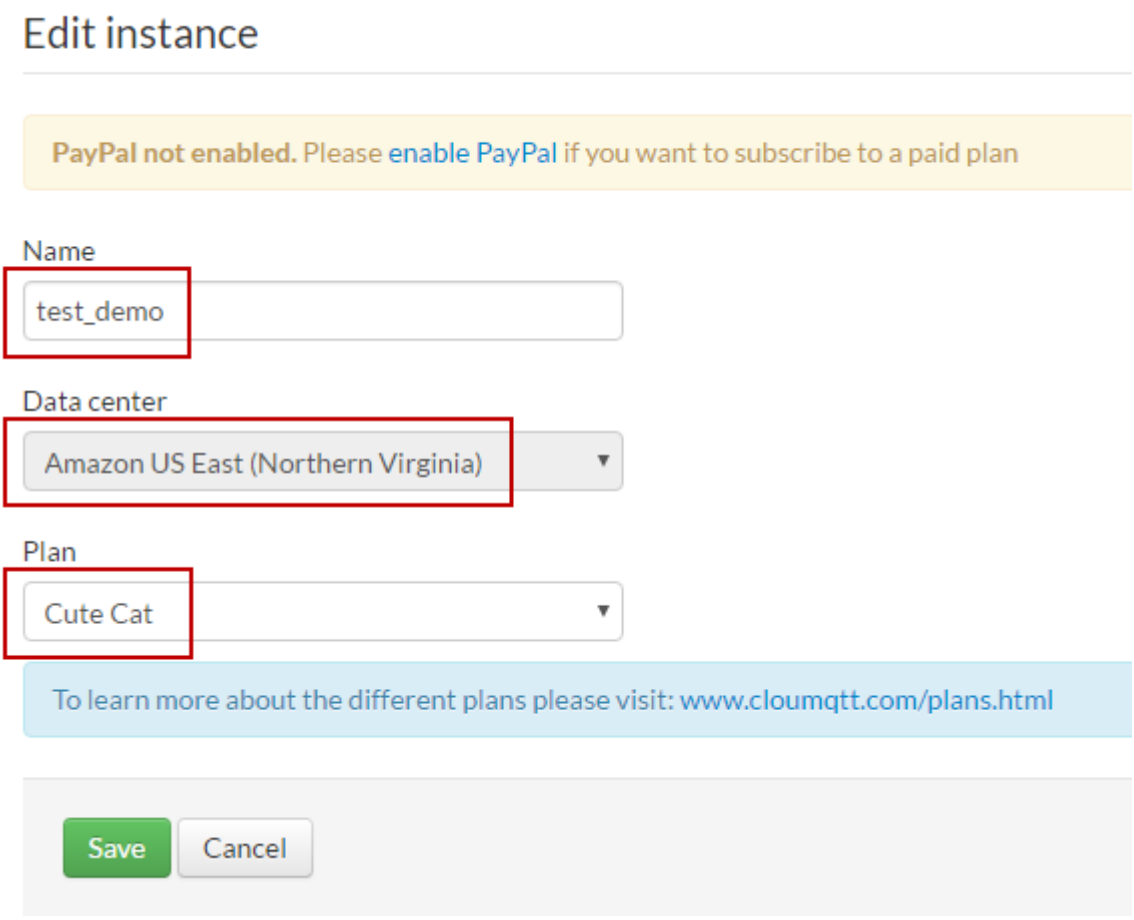
- Nhiệm vụ 1: Gửi (publish) dữ liệu lên broker, và nhận thông tin (subcribe) từ broker, mục đích chính là kiểm tra publish và subcribe giữa ESP8266 và MQTT Broker.

- Nhiệm vụ 2: Nhận(subcribe) thông tin từ broker, kiểm tra dữ liệu và thực hiện bật tắt LED

Tạo tài khoản và cấu hình CloudMQTT

Bước 1: Bạn vào trực tiếp trang CloudMQTT để tạo một tài khoản cho mình, sẽ có một đường dẫn để kích hoạt thông tin và cấu hình cho tài khoản của bạn, trừ email và mật khẩu ra thì các ô còn lại các bạn nhập gì tùy thích.

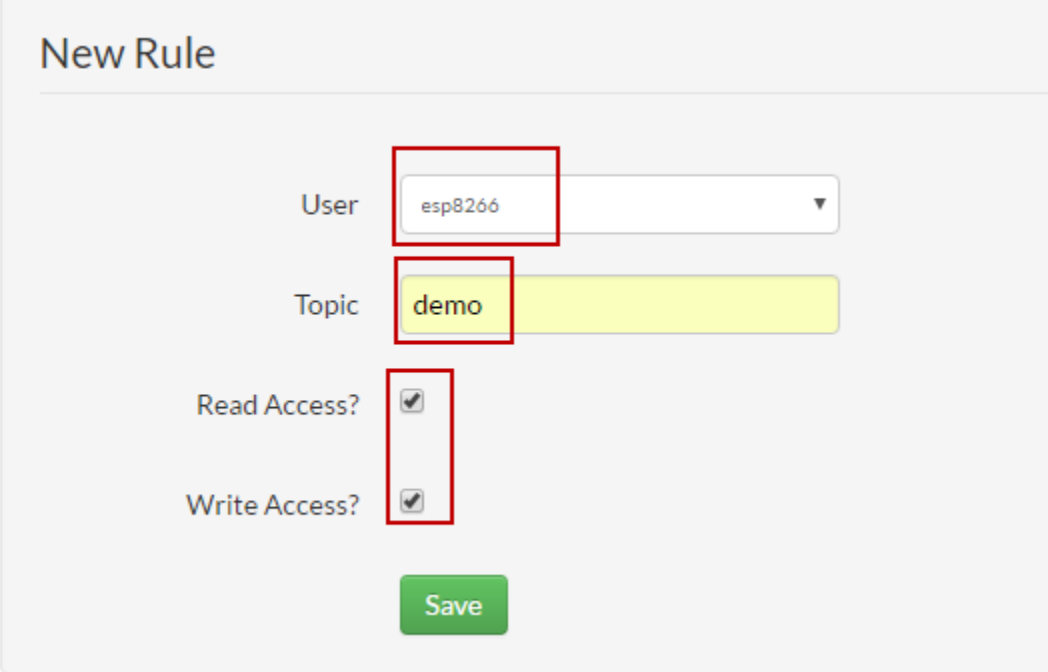
Bước 2: Tạo MQTT Broker bằng cách ấn vào nút Create màu xanh, nhập các thông tin vào như hình.



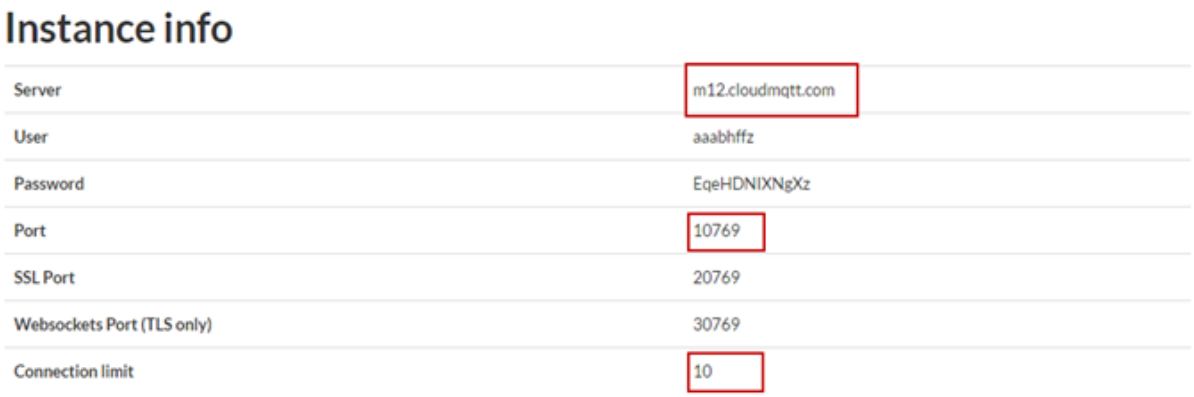
Bước 3: Chọn Details để cấu hình thông tin

Trong manage User điền tên user là esp8266, password là 123456, sau đó Save

Trong New Rule chọn user như hình, tên topic đặt demo, tick chọn Read,Write



Vậy là xong bước cấu hình broker, kết quả, ta sẽ quan tâm tới 3 ô khoanh đỏ



Lập trình ESP8266

Publish thông tin lên topic

Để ESP8266 có thể publish và subcribe dữ liệu lên MQTT broker thì cần phải có thư viện MQTT, ở đây mình dùng thư viện sẵn có là PubSubClient để thử nghiệm cho nhanh

```
#include <ESP8266WiFi.h>

#include <PubSubClient.h>

// Cập nhật thông tin

// Thông tin về wifi

#define ssid "ten_wifi"

#define password "password"

// Thông tin về MQTT Broker

#define mqtt_server "m12.cloudmqtt.com" // Thay bằng thông tin của bạn

#define mqtt_topic_pub "demo"   //Giữ nguyên nếu bạn tạo topic tên là demo

#define mqtt_topic_sub "demo"

#define mqtt_user "esp8266"    //Giữ nguyên nếu bạn tạo user là esp8266 và pass là 123456

#define mqtt_pwd "123456"



const uint16_t mqtt_port = 10769; //Port của CloudMQTT



WiFiClient espClient;

PubSubClient client(espClient);



long lastMsg = 0;
```

```
char msg[50];

int value = 0;



void setup() {

  Serial.begin(115200);

  setup_wifi();

  client.setServer(mqtt_server, mqtt_port);

  client.setCallback(callback);

}

// Hàm kết nối wifi

void setup_wifi() {

  delay(10);

  Serial.println();

  Serial.print("Connecting to ");

  Serial.println(ssid);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {

    delay(500);

    Serial.print(".");

  }

  Serial.println("");

  Serial.println("WiFi connected");
```

```
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
// Hàm call back để nhận dữ liệu
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}
// Hàm reconnect thực hiện kết nối lại khi mất kết nối với MQTT Broker
void reconnect() {
  // Chờ tới khi kết nối
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Thực hiện kết nối với mqtt user và pass
    if (client.connect("ESP8266Client",mqtt_user, mqtt_pwd)) {
      Serial.println("connected");
      // Khi kết nối sẽ publish thông báo
```

```
      client.publish(mqtt_topic_pub, "ESP_reconnected");
      // ... và nhận lại thông tin này
      client.subscribe(mqtt_topic_sub);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Đợi 5s
      delay(5000);
    }
  }
}
void loop() {
  // Kiểm tra kết nối
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  // Sau mỗi 2s sẽ thực hiện publish dòng hello world lên MQTT broker
  long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;
```

```
    ++value;

    snprintf (msg, 75, "hello world #%ld", value);

    Serial.print("Publish message: ");

    Serial.println(msg);

    client.publish(mqtt_topic_pub, msg);

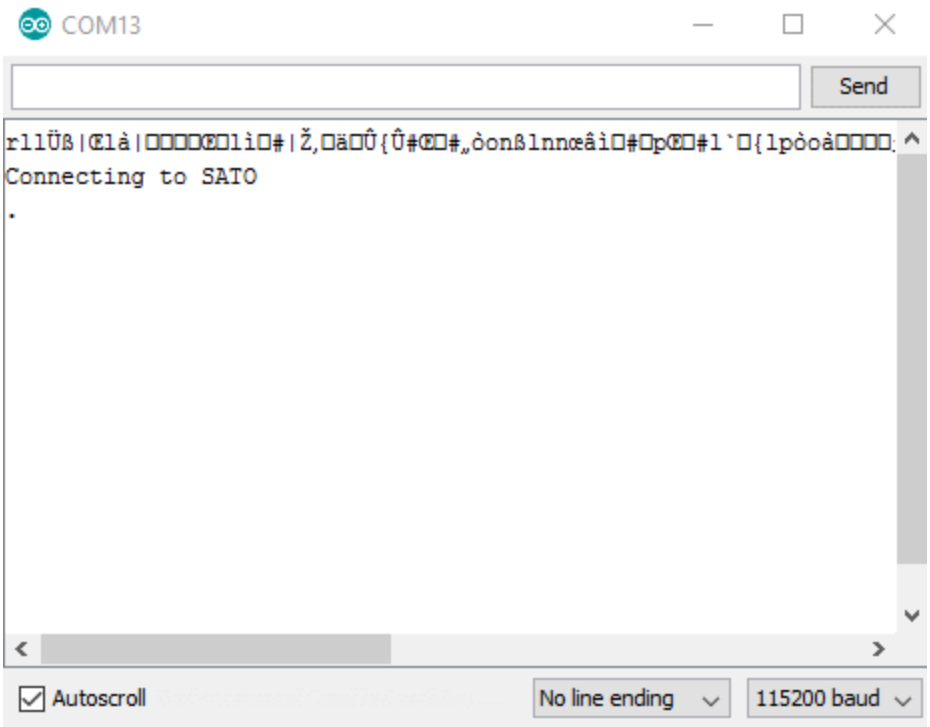
  }
}
```

Arduino

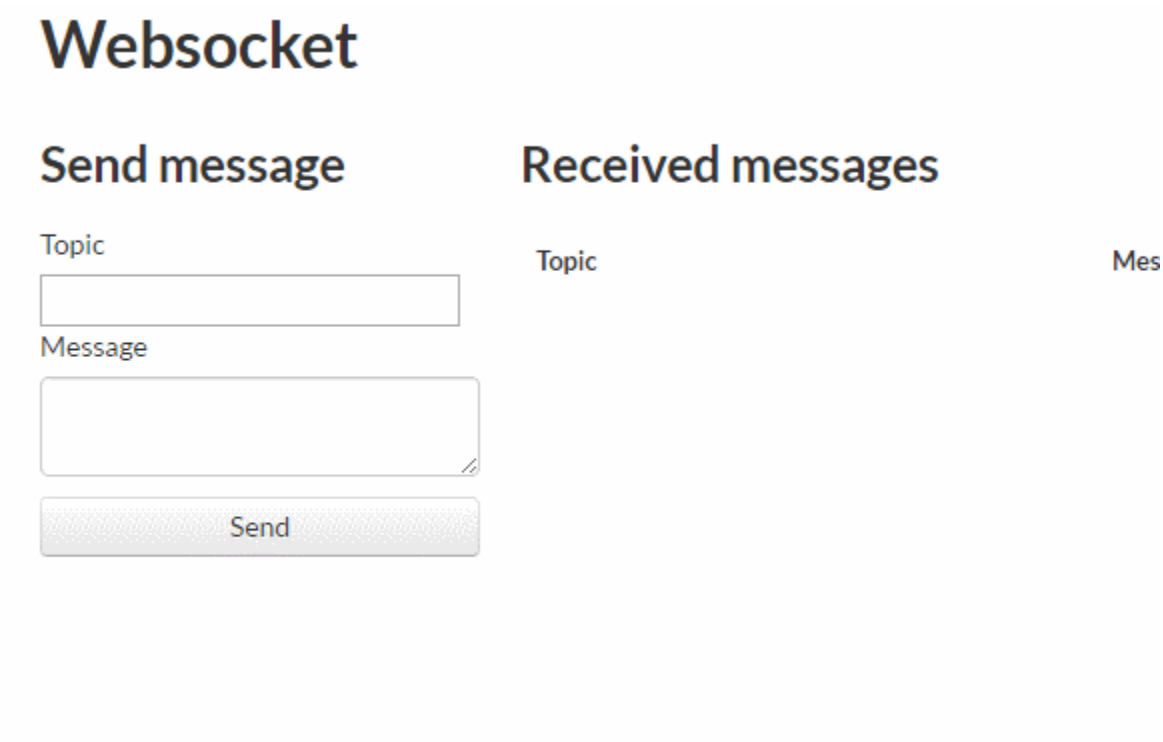
Mình giải thích qua một chút về chương trình, chương trình này sẽ thực hiện kết nối với mạng wifi trước để có kết nối tới internet, sau đó thực hiện gửi thông tin lên broker bằng lệnh `publish`, sau đó lắng nghe thông tin có trên Broker(ở đây là thông tin do chính nó gửi lên) và in thông tin này ra terminal của Arduino thông qua hàm `callback`.

Kết quả

Log Arduino



Log Websocket UI



Bật tắt LED

Coi như đã giao tiếp thực hiện publish và subcribe trên MQTT Broker. Giờ thực hiện cải tiến thêm một chút, là từ giao diện Websocket UI Send message xuống, ở dưới khi nhận được message thì ESP8266 thực hiện bật/tắt LED có sẵn trên kit.

```
#include <ESP8266WiFi.h>

#include <PubSubClient.h>

// Cập nhật thông tin

// Thông tin về wifi

#define ssid "ten_wifi"

#define password "password"

// Thông tin về MQTT Broker

#define mqtt_server "m12.cloudmqtt.com" // Thay bằng thông tin của bạn

#define mqtt_topic_pub "demo"   //Giữ nguyên nếu bạn tạo topic tên là demo

#define mqtt_topic_sub "demo"

#define mqtt_user "esp8266"    //Giữ nguyên nếu bạn tạo user là esp8266 và pass là 123456

#define mqtt_pwd "123456"



const uint16_t mqtt_port = 10769; //Port của CloudMQTT

const byte ledPin = D0;



WiFiClient espClient;

PubSubClient client(espClient);
```

```
long lastMsg = 0;
char msg[50];
int value = 0;


void setup() {
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);
  pinMode(ledPin, OUTPUT);
}
// Hàm kết nối wifi
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
```

```
  }

  Serial.println("");

  Serial.println("WiFi connected");

  Serial.println("IP address: ");

  Serial.println(WiFi.localIP());

}

// Hàm call back để nhận dữ liệu

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

  Serial.print("Message arrived [");

  Serial.print(topic);

  Serial.print("] ");

  for (int i = 0; i < length; i++) {

    char receivedChar = (char)payload[i];

    Serial.print(receivedChar);

    if (receivedChar == '1')

      // Kiểm tra nếu tin nhận được là 1 thì bật LED và ngược lại

      digitalWrite(ledPin, HIGH);

    if (receivedChar == '0')

      digitalWrite(ledPin, LOW);

  }

  Serial.println();

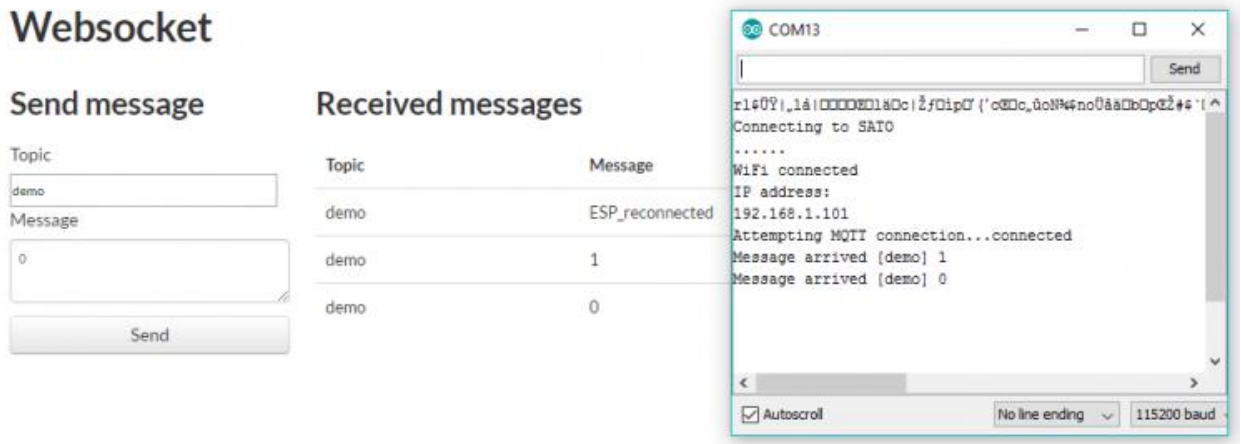
}
```

```
// Hàm reconnect thực hiện kết nối lại khi mất kết nối với MQTT Broker
void reconnect() {
  // Chờ tới khi kết nối
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Thực hiện kết nối với mqtt user và pass
    if (client.connect("ESP8266Client",mqtt_user, mqtt_pwd)) {
      Serial.println("connected");
      // Khi kết nối sẽ publish thông báo
      client.publish(mqtt_topic_pub, "ESP_reconnected");
      // ... và nhận lại thông tin này
      client.subscribe(mqtt_topic_sub);
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Đợi 5s
      delay(5000);
    }
  }
}
void loop() {
```

```arduino
  // Kiểm tra kết nối

  if (!client.connected()) {

    reconnect();

  }

  client.loop();
}
```

Arduino

Với chương trình này mình sẽ thay đổi và thực hiện kiểm tra gói tin từ Broker gửi về để điều khiển LED trong hàm `callback`
Kết quả



Các khái niệm trong MQTT ?

Một số ví dụ trên có thể cho chúng ta phần nào hình dung được về MQTT, tất nhiên nó vẫn dừng lại ở mức độ cơ bản. Nói lại cách giao tiếp thông thường khi một device kết nối với server, sẽ gửi POST request với giá trị của sensor(ở đây ví dụ ta sẽ gửi dữ liệu từ sensor lên server), sau đó sẽ là ngắt kết nối với server, đợi khoảng 1 thời gian nào đó, sau đó lặp lại các bước trên. Vậy mỗi lần cần gửi giá trị của sensor thì device lại phải thực hiện kết nối với server lại, việc kết nối nhiều lần ở đây ta thấy là không cần thiết.

Ngược lại, nếu chúng ta muốn gửi data từ server về device, thì chỉ có cách duy nhất là device phải kết nối với server và phải thực hiện kiểm tra nếu có update dữ liệu mới. Đây rõ ràng không phải là cách hiệu quả nhất để gửi dữ liệu. Giao thức HTTP sẽ sử dụng phương thức giao tiếp là request-response, phương thức này thì tốt cho việc transfer các dữ liệu lớn nhưng nó không phải là protocol hữu hiệu nhất khi chúng ta chỉ cần gửi vài byte dữ liệu(giá trị sensor), và đó là lý do cần phải dùng MQTT

MQTT (Message Queuing Telemetry Transport) là một giao thức gởi dạng publish/subscribe sử dụng cho các thiết bị Internet of Things với băng thông thấp, độ tin cậy cao và khả năng được sử dụng trong mạng lưới không ổn định. Bởi vì giao thức này sử dụng băng thông thấp trong môi trường có độ trễ cao nên nó là một giao thức lý tưởng cho các ứng dụng M2M. MQTT cũng là giao thức sử dụng trong Facebook Messager.

Các định nghĩa **“subscribe”, “publish”, “qos”, “retain”, “last will and testament (lwt)”** trong giao thức MQTT là gì ?

Publish, subscribe

Trong một hệ thống sử dụng giao thức MQTT, nhiều node trạm (gọi là mqtt client – gọi tắt là client) kết nối tới một MQTT server (gọi là broker). Mỗi client sẽ đăng ký một vài kênh (topic), ví dụ như “/client1/channel1”, “/client1/channel2”. Quá trình đăng ký này gọi là **“subscribe”**, giống như chúng ta đăng ký nhận tin trên một kênh Youtube vậy. Mỗi client sẽ nhận được dữ liệu khi bất kỳ trạm nào khác gởi dữ liệu và kênh đã đăng ký. Khi một client gởi dữ liệu tới kênh đó, gọi là **“publish”**.

QoS

Ở đây có 3 tuỳ chọn *_QoS (Qualities of service)_* * khi “publish” và “subscribe”:

- **QoS0** Broker/client sẽ gởi dữ liệu đúng 1 lần, quá trình gởi được xác nhận bởi chỉ giao thức TCP/IP, giống kiểu đem con bỏ chợ.

- **QoS1** Broker/client sẽ gởi dữ liệu với ít nhất 1 lần xác nhận từ đầu kia, nghĩa là có thể có nhiều hơn 1 lần xác nhận đã nhận được dữ liệu.
- **QoS2** Broker/client đảm bảm khi gởi dữ liệu thì phía nhận chỉ nhận được đúng 1 lần, quá trình này phải trải qua 4 bước bắt tay.

Bạn có thể tham khảo thêm về **QoS**

Một gói tin có thể được gởi ở bất kỳ QoS nào, và các client cũng có thể subscribe với bất kỳ yêu cầu QoS nào. Có nghĩa là client sẽ lựa chọn QoS tối đa mà nó có để nhận tin. Ví dụ, nếu 1 gói dữ liệu được publish với QoS2, và client subscribe với QoS0, thì gói dữ liệu được nhận về client này sẽ được broker gởi với QoS0, và 1 client khác đăng ký cùng kênh này với QoS 2, thì nó sẽ được Broker gởi dữ liệu với QoS2.

Một ví dụ khác, nếu 1 client subscribe với QoS2 và gói dữ liệu gởi vào kênh đó publish với QoS0 thì client đó sẽ được Broker gởi dữ liệu với QoS0. QoS càng cao thì càng đáng tin cậy, đồng thời độ trễ và băng thông đòi hỏi cũng cao hơn.

Retain

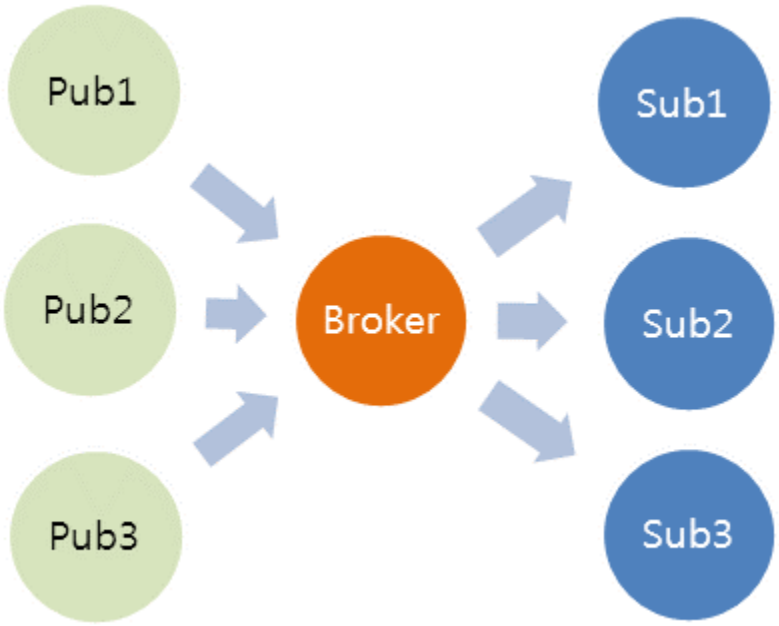
Nếu RETAIN được set bằng 1, khi gói tin được publish từ Client,
Broker **PHẢI** lưu trữ lại gói tin với QoS, và nó sẽ được gởi đến bất kỳ Client nào subscribe cùng kênh trong tương lai. Khi một Client kết nối tới Broker và subscribe, nó sẽ nhận được gói tin cuối cùng có RETAIN = 1 với bất kỳ topic nào mà nó đăng ký trùng. Tuy nhiên, nếu Broker nhận được gói tin mà có QoS = 0 và RETAIN = 1, nó sẽ huỷ tất cả các gói tin có RETAIN = 1 trước đó. Và phải lưu gói tin này lại, nhưng hoàn toàn có thể huỷ bất kỳ lúc nào.

Khi publish một gói dữ liệu đến Client, Broker phải se RETAIN = 1 nếu gói được gởi như là kết quả của việc subscribe mới của Client (giống như tin nhắn ACK báo subscribe thành công). RETAIN phải bằng 0 nếu không quan tâm tới kết quả của việc subscribe.

LWT

Gói tin LWT (last will and testament) không thực sự biết được Client có trực tuyến hay không, cái này do gói tin KeepAlive đảm nhận. Tuy nhiên gói tin LWT như là thông tin điều gì sẽ xảy đến sau khi thiết bị ngoại tuyến.

MQTT hoạt động như thế nào ?



MQTT rất nhẹ và nhanh, chỉ cần vài byte để kết nối tới server và kết nối này được giữ mọi lúc. Kết nối của nó sẽ dùng ít dữ liệu và thời gian hơn giao thức HTTP. Vậy MQTT hoạt động ra sao ?

Toàn bộ hệ thống của chúng ta sẽ bao gồm rất nhiều clients và 1 broker duy nhất, tất cả các device của chúng ta được gọi chung là clients, client cũng có thể là điện thoại, là laptop, là pc hoặc là esp8266. Mỗi device này chỉ kết nối với một broker duy nhất và chúng không kết nối với nhau.

Toàn bộ hệ thống sẽ dựa trên phương thức kết nối là publish – subcribe, mỗi client có thể là một publisher – gửi messages, hoặc là subcriber – lắng nghe message tới, hoặc đồng thời là cả 2 trong cùng 1 thời điểm.

Broker ở đây là loại server với nhiệm vụ là cho phép publish mesage từ publisher và chuyển tiếp nó tới subcriber.

Vậy làm thế nào để broker hiểu được để chuyển đúng message tới subscriber ? Chúng ta thử nghĩ nếu mỗi subscriber đều nhận được mọi message, trong khi đó có rất nhiều publisher gửi message trong hệ thống, nghĩa là subscriber sẽ nhận được 1 đống tin nhắn rác, để giải quyết vấn đề này thì cần có các topic.

Mỗi publisher sẽ gửi tới một hoặc nhiều topic và mỗi subscriber sẽ subscribe từ một hoặc nhiều topic có liên quan. Các topic này sẽ có dạng cấu trúc cây, và được phân chia bởi dấu /

Ví dụ 1: Chúng ta có nhiệt độ môi trường ở một số nơi khác nhau và tôi thì thích nhiệt độ ở trong vườn nhà mình. Tôi sẽ dùng smartphone để subscribe topic là `Sensors/Garden/Temperature`. Còn device dùng để đo nhiệt độ môi trường là ESP8266 sẽ được đặt ở trong vườn, kết nối với internet và publish giá trị nhiệt độ lên topic `Sensors/Garden/Temperature`, và lúc này thì smartphone của tôi sẽ nhận được thông tin giá trị nhiệt độ hiện tại.
Ví dụ 2: Một ví dụ khác khi bạn dùng với LWT và keep alive là ta có 1 cảm biến, nó gửi những dữ liệu quan trọng và rất không thường xuyên. Nó có publish trước với Broker một tin nhắn lwt ở topic `/node/gone-offline` với tin nhắn `id` của nó. Và tôi cũng subscribe topic `/node/gone-offline`, sẽ gởi SMS tới điện thoại thôi mỗi khi nhận được tin nhắn nào ở kênh mà tôi subscribe.
Trong quá trình hoạt động, cảm biến luôn giữ kết nối với Broker bởi việc luôn gởi gói tin keepAlive. Nhưng nếu vì lý do gì đó, cảm biến này chuyển sang ngoại tuyến, kết nối tới Broker timeout do Broker không còn nhận được gói keepAlive.
Lúc này, do cảm biến của tôi đã đăng ký LWT, do vậy broker sẽ đóng kết nối của Cảm biến, đồng thời sẽ publish một gói tin là Id của cảm biến vào kênh `/node/gone-offline`, dĩ nhiên là tôi cũng sẽ nhận được tin nhắn báo rằng cảm biến yêu quý của mình đã ngoại tuyến.
Tóm gọn: Ngoài việc đóng kết nối của Client đã ngoại tuyến, gói tin LWT có thể được định nghĩa trước và được gởi bởi Broker tới kênh nào đó khi thiết bị đăng ký LWT ngoại tuyến.

Một ví dụ minh họa sống động về MQTT pub sub lên các topic bạn có thể xem tại đây

Thực hành 2

Tiếp tục từ bài thực hành 1 thì chúng ta sẽ phát triển thêm một chút nữa, xây dựng hệ thống với 2 client là eps8266 và laptop(pub sub lên broker dùng mqttlen) và broker là cloudMQTT

ESP8266 sẽ thực hiện publish data lên topic esp/test sau mỗi 10s và subcribe data từ broker

```
#include <ESP8266WiFi.h>

#include <PubSubClient.h>



const char* ssid = "wifi_name";

const char* password =  "pass";

const char* mqttServer = "m12.cloudmqtt.com";

const int mqttPort = 10769;

const char* mqttUser = "your_username";

const char* mqttPassword = "your_password";



long lastMsg = 0;

char msg[50];

int value = 0;



WiFiClient espClient;
```

```
PubSubClient client(espClient);



void setup() {



  Serial.begin(115200);



  WiFi.begin(ssid, password);



  while (WiFi.status() != WL_CONNECTED) {

    delay(500);

    Serial.println("Connecting to WiFi..");

  }

  Serial.println("Connected to the WiFi network");



  client.setServer(mqttServer, mqttPort);

  client.setCallback(callback);



  while (!client.connected()) {

    Serial.println("Connecting to MQTT...");



    if (client.connect("ESP8266Client", mqttUser, mqttPassword )) {
```

```
      Serial.println("connected");


    } else {


      Serial.print("failed with state ");
      Serial.print(client.state());
      delay(2000);


    }
  }
  //Pub Hello to esp/test topic and sub this topic
  client.publish("esp/test", "Hello from ESP8266");
  client.subscribe("esp/test");


}


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


  Serial.print("Message arrived in topic: ");
  Serial.println(topic);


  Serial.print("Message:");
```

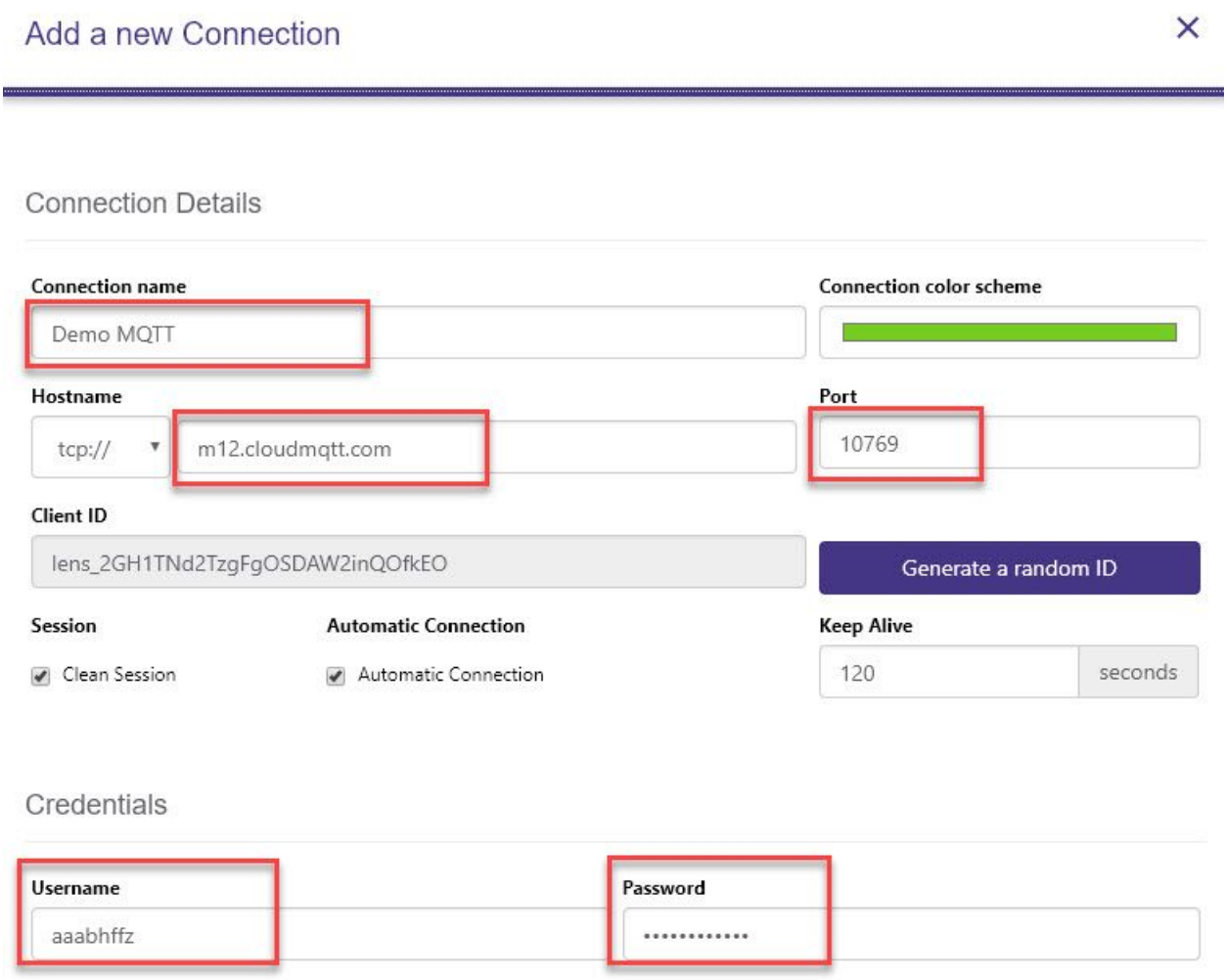
```
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }

  Serial.println();
  Serial.println("-----------------------");

}

void loop() {
  client.loop();
  long now = millis();
  if (now - lastMsg > 8000) {
    lastMsg = now;
    ++value;
    snprintf (msg, 75, "hello world #%ld", value);

    //    client.publish(mqtt_topic_pub, msg);
    if (value < 10) {
      Serial.print("Publish message: ");
      Serial.println(msg);
      client.publish("esp/test", msg);
```

```
    }

  }
}
```

Arduino

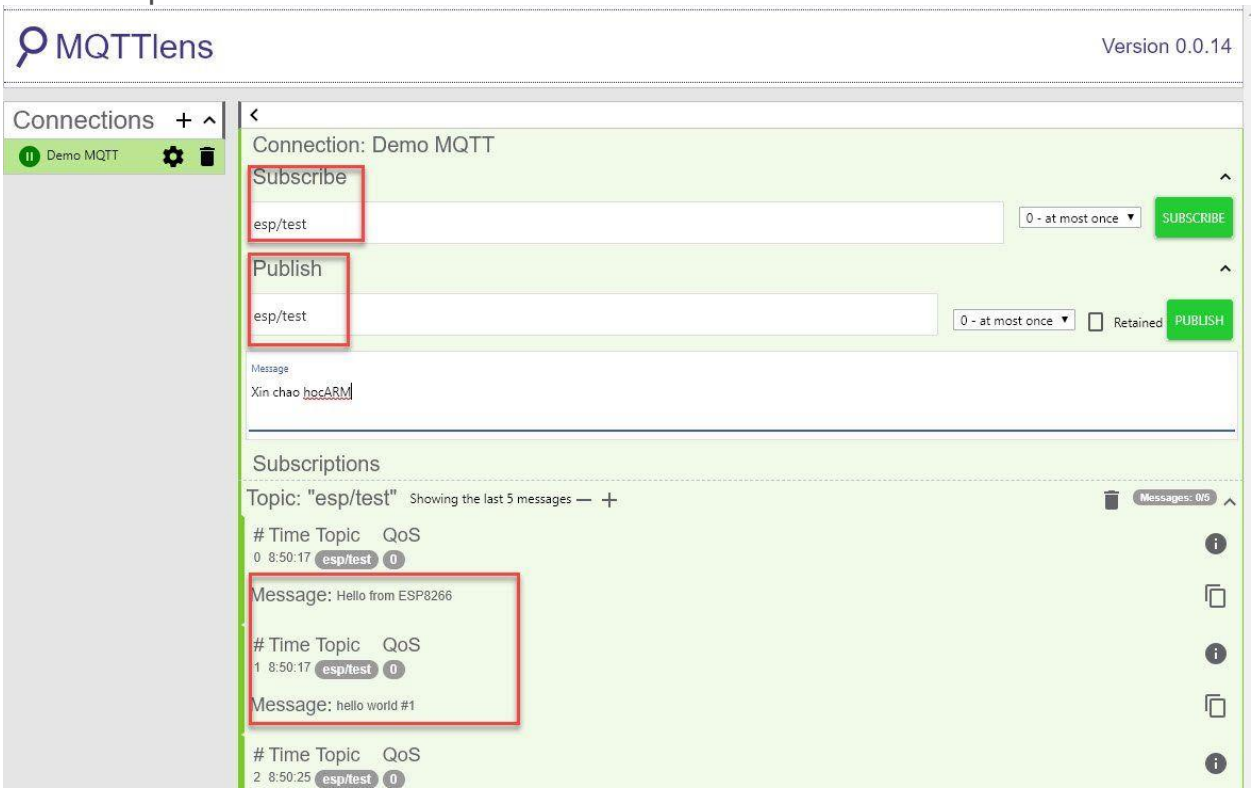
PC thực hiện publish/subscribe data lên topic esp/test bằng mqttlen

- Cấu hình MQTTlens, bạn tạo mới một connection với tên bất kỳ, hostname là tên server cloudmqtt với port là 10769, lưu ý phần credentials nhập username và password trên cloudmqtt của bạn vào

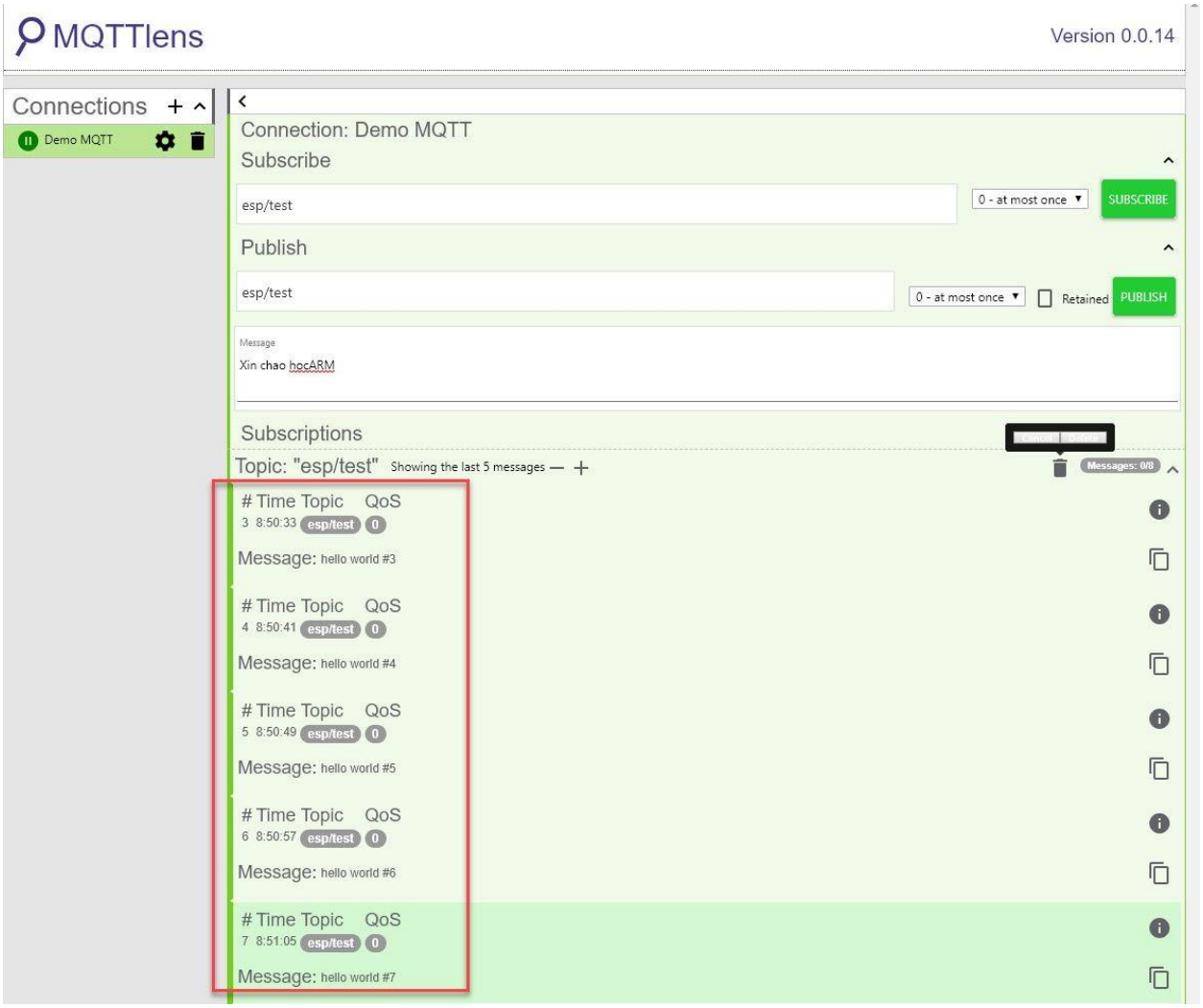


- Publish/Subcribe với MQTTlen: để nhận được message đưa lên broker tại topic esp/test thì ta nhập topic này tại phần subscribe,

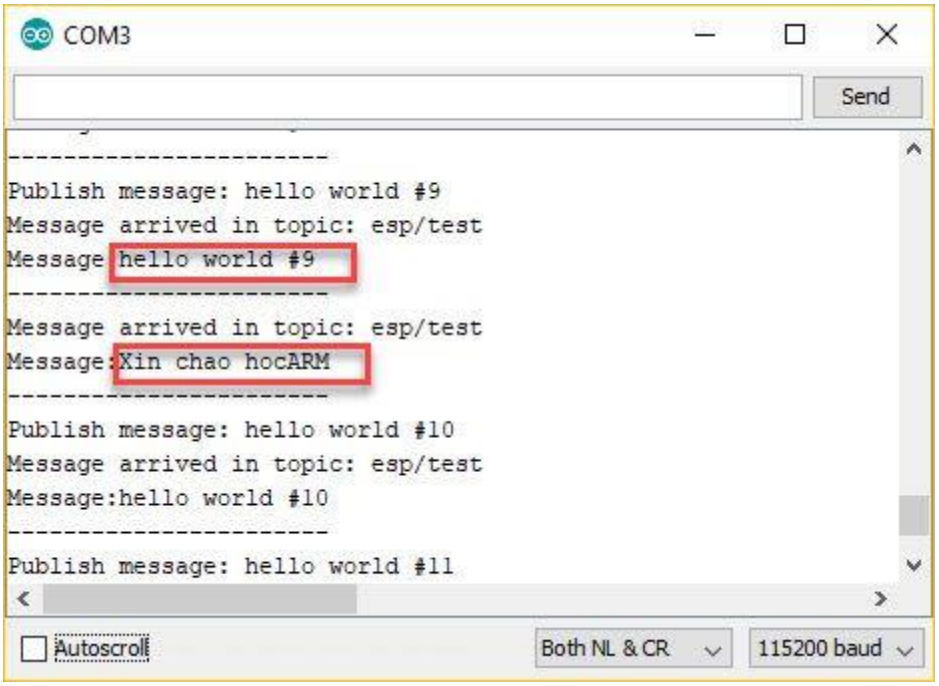
sau đó ấn nút subscribe, và để publish message lên topic esp/test trên broker thì ta nhập topic và nội dung message, sau đó ấn publish.



Phần subscriptions sẽ là thông tin của các message gửi lên topic, ở đây ta có thể thấy thông tin từ esp8266 gửi lên broker thông qua topic là esp/test



Kết quả sau khi publish và subcribe với ESP8266: ta thấy các message publish lên sẽ có dạng hello world #x, khi có một message được publish từ mqttlens thì ESP sẽ nhận được tin với nội dung Xin chao hocARM



Tạm kết

Vậy là mình đã đi từ thực hành trước để các bạn có thể hình dung được sơ bộ MQTT, sau đó là một ít khái niệm cơ bản, và cuối cùng lại quay về với thực hành trên 2 device pub/sub lên 1 topic qua broker. Ta có thể nhận thấy với thư viện PubsubClient thì chỉ thấy được pub, sub, mình cũng có kiểm tra xem có Qos, LWT không nhưng chưa thấy trong thư viện. Vẫn còn rất nhiều điều phải làm với MQTT, còn một cách hay ho hơn là kết hợp MQTT và tạo giao diện điều khiển với nodered thông qua bài viết sau.

Còn nếu bạn muốn chi tiết hơn về cách tự tạo MQTT Broker và Client thì sao ? Mời các bạn theo dõi bài viết sau

MQTT Client và MQTT Broker

MQTTESP8266New Update

Như chúng ta đã tìm hiểu ở phần trước về MQTT và thực hành cơ bản, ta đã biết được MQTT là gì, cách để sử dụng MQTT và dùng ESP8266 trong MQTT như thế nào, giờ thì mình sẽ đi cụ thể hơn về 2 thành phần của MQTT là MQTT Client và MQTT Broker với một số loại chính cũng như là cách để chúng ta có thể setup, lập trình và test nó.

MQTT và ESP8266 | Học ARM

MQTT là gì ? Cách sử dụng MQTT như thế nào ? Các ví dụ về MQTT với ESP8266 được thực hiện ra sao ? Trong bài viết này mình sẽ trình bày một số khái niệm và các bước thực hiện MQTT với ví dụ minh họa cơ bản trên ESP8266 cho mọi người cùng tham khảo. Trước khi bắt đầu mình cùng xem qua một hình ảnh c...

SCHọc ARM

MQTT Client

2 thành phần publisher và subscriber là đặc trưng tạo nên giao thức MQTT. Các MQTT Client không kết nối trực tiếp với nhau, mọi gói dữ liệu được gửi đi đều thông qua MQTT Broker. Để có thể triển khai các ứng dụng của MQTT Client, chúng ta cần MQTT Broker (sẽ được trình bày trong phần sau). Ở phần này chúng ta sẽ làm quen với giao thức MQTT bằng các ví dụ sử dụng MQTT Client thông dụng và các dịch vụ MQTT Broker miễn phí và phổ biến, 2 trong số chúng là test.mostquitto.org(đã ra đi) và cloudmqtt.com(hết băng thông).

Lưu ý là 2 server trên 1 loại mostquitto hiện nay đã hết sử dụng được và cloudmqtt thì gói cat free của họ cũng đã hết băng thông. Cách khắc phục đơn giản là tự setup server mqtt local để test thôi, mình có hướng dẫn cách setup broker ở phía dưới của bài viết nên ae khỏi lo nhé
Dưới đây là một số loại MQTT client cơ bản để mình có thể test cơ bản, lập trình với js và lập trình với ESP8266.

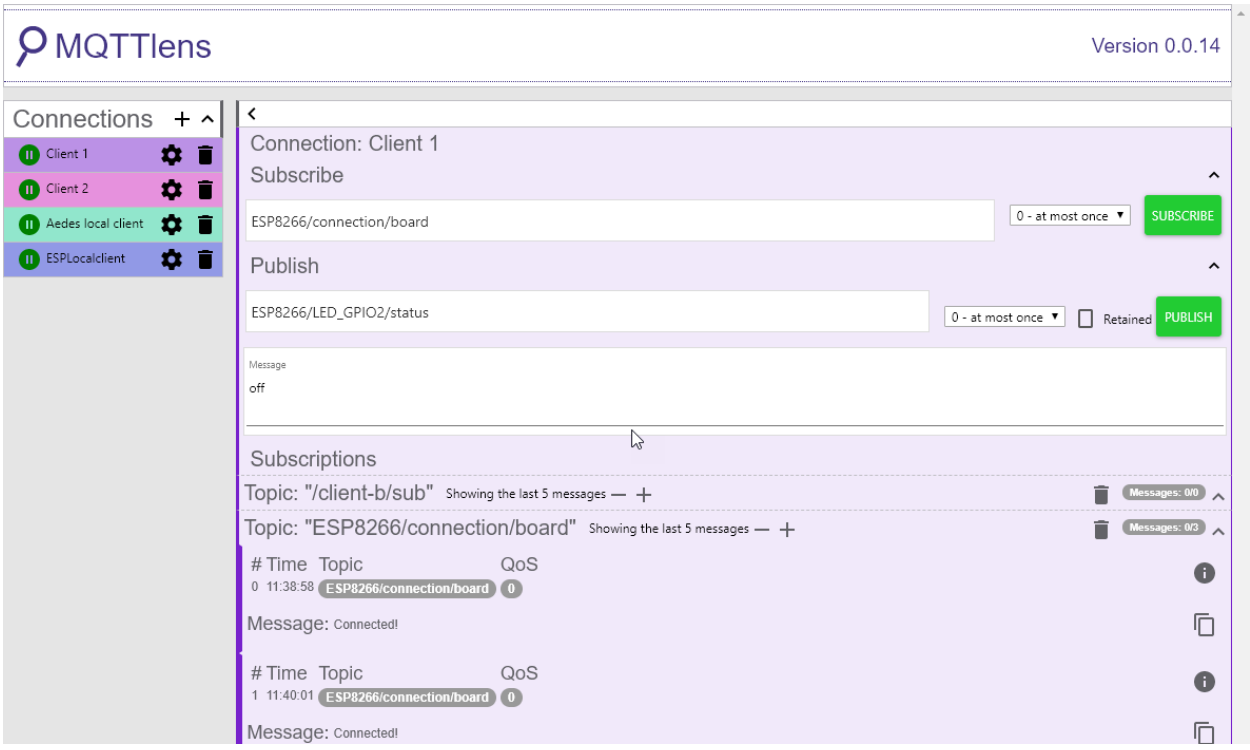
MQTT Lens

MQTT Lens là một tiện ích mở rộng của Chrome (Chrome Extension), nó sử dụng trình duyệt Chrome để kết nối đến MQTT Broker cũng như test các tính năng publish/subcribe của giao thức MQTT. Đây là một công cụ rất hữu ích để kiểm tra kết nối đến MQTT Broker và kiểm tra việc gửi và nhận gói tin.

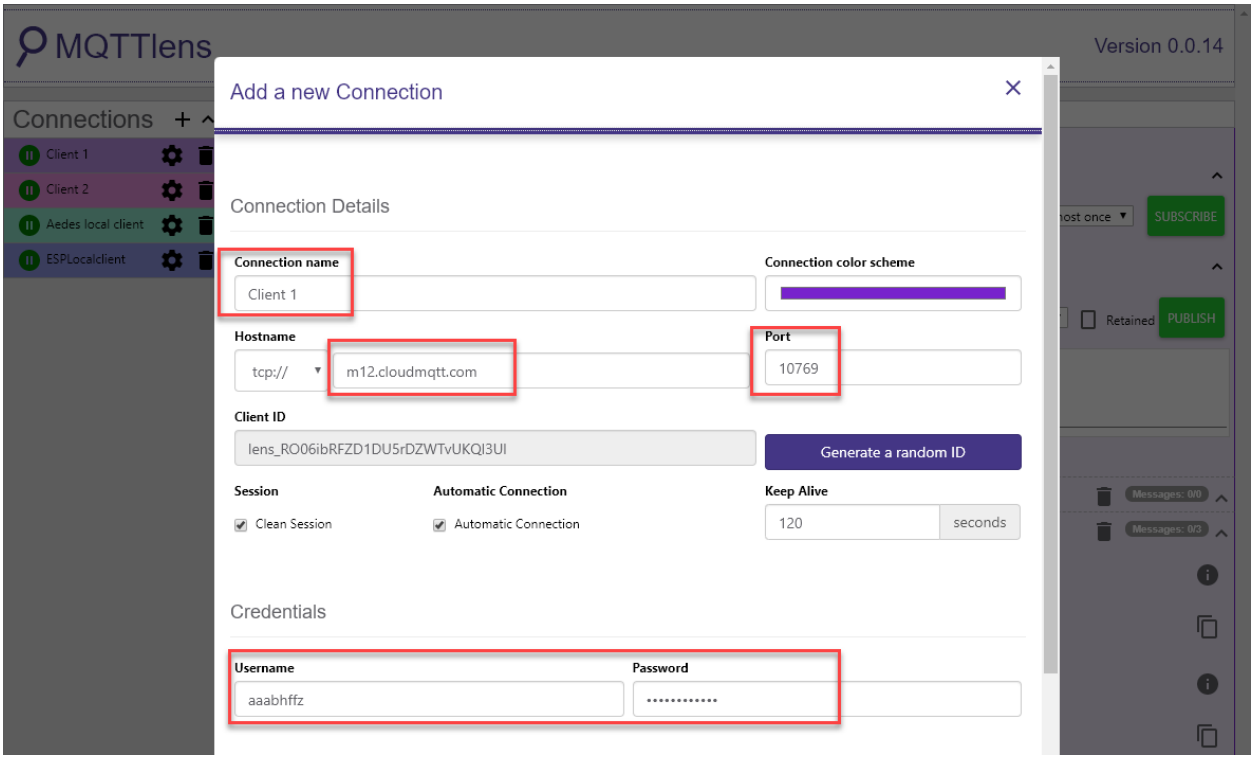
Chúng ta sẽ sử dụng công cụ này với dịch vụ MQTT Broker của cloudmqtt được trình bày như các bước bên dưới:

Bước 1: Cài đặt trình duyệt Chrome, và tìm kiếm extension MQTTlen với google,
link https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm?hl=vi

Bước 2: Thêm và khởi chạy MQTT lens



Bước 3 : Tạo 1 MQTT Client kết nối đến MQTT Broker cloudmqtt tự publish và subscribe message của chính mình như các bước bên dưới

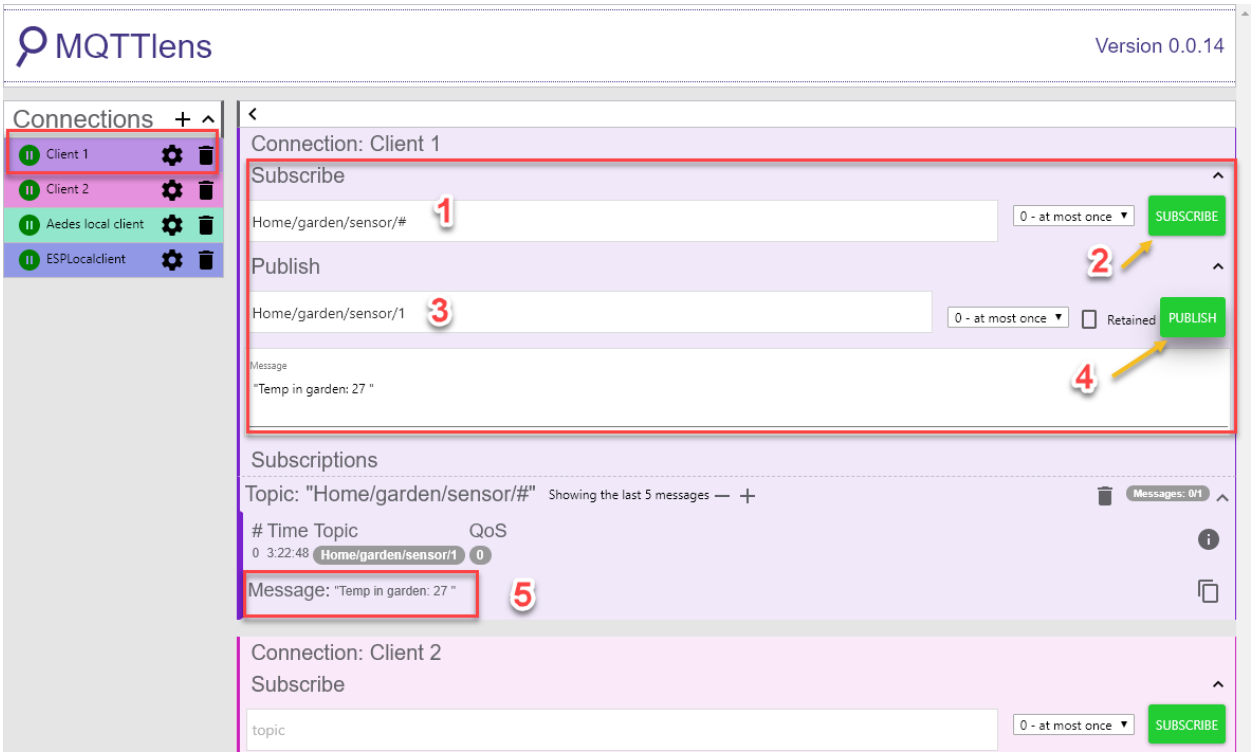


Chúng ta sẽ tạo 1 connection có tên Client 1 với host name của MQTT Broker là m12.cloudmqtt.com(thông tin theo tài khoản của bạn), Broker này sẽ giúp trao đổi dữ liệu của các Client với nhau và lắng nghe các Client ở port 10769(port được cung cấp bởi cloudmqtt)

Ở connection này sẽ đăng kí nhận gói tin tại topic `Home/garden/sensor/#` (kí tự # cho phép subcribe các topic `Home/garden/sensor/1`, `Home/garden/sensor/2`, vv...). Tiếp theo chúng ta sẽ pulish 1 gói dữ liệu với nội dung "Temp in garden: 27" tại topic `Home/garden/sensor/1`.

Kết quả

Tại mục subrcriptions, chúng ta sẽ nhận được gói dữ liệu đã publish do đã subcribe topic Home/garden/sensor/# như hình bên dưới.



MQTT.js

MQTT.js là một thư viện MQTT client, được viết bằng ngôn ngữ JavaScript trên nền tảng Node.js và hỗ trợ MQTT Over Websocket (MOW).Đây là dự án mã nguồn mở và ta có thể tải MQTT.js mới nhất tại MQTT.js

Cài đặt

Trước tiên ta cần kiểm tra hệ điều hành đã hỗ trợ Node.js trước khi cài đặt MQTT.js. Nếu chưa thì cài NodeJS tại nodejs.org/en/
Khởi tạo một dự án Node.js. Cần tạo một thư mục project, sau đó mở terminal (linux OS) hoặc Command Prompt (trên Windowns OS) và dùng lệnh:

```
npm init
```

Bash

Khi chạy lệnh này, ta phải khai báo thêm một số thông tin cho dự án như tên, phiên bản, keywords, tác giả,... Sau khi tạo xong, trong thư mục vừa tạo sẽ xuất hiện một file là `package.json` với nội dung là các phần đã khai báo.

File này cũng chứa thuộc tính dùng để lưu trữ các package chúng ta đã cài đặt.
Tiếp theo chúng ta sẽ cài MQTT.js, sử dụng lệnh:

```
npm install mqtt --save
```

Bash

Sau khi cài đặt xong, bạn có thể sử dụng module mqtt để thực hiện việc kết nối MQTT Client với Broker, publish message hay subscribe topic. Tất nhiên, toàn bộ các file liên quan đến thư viện sẽ nằm trong thư mục node_modules, trong thư mục dự án.
Để hiểu rõ hơn cách hoạt động của MQTT.js, chúng ta sẽ tạo ra thêm 1 số file code Javascript (file .js) là client-a.js và client-b.js thực hiện subcribe và puslish các gói tin.

Lập trình

2 Client này sẽ kết nối vào cùng 1 MQTT Broker.

Client A sẽ subscribe topic `/client-a/sub`, nếu nhận bất kỳ dữ liệu nào được publish vào kênh này, client A sẽ publish message `Hello from client A` vào topic `/client-b/sub` và đóng kết nối, kết thúc.
Client B sẽ subscribe topic `/client-b/sub`, nếu nhận bất kỳ dữ liệu nào được publish vào kênh này, client B sẽ đóng kết nối và kết thúc. Ngay khi kết nối vào Broker, client B sẽ public 1 gói tin `Hello from client B` vào topic`/client-a/sub`
Chương trình client A

```
// tạo biến mqtt sử dụng các chức năng của module mqtt

var mqtt = require('mqtt');

// tạo option sử dụng thuộc tính connect để kết nối đến broket MQTT

var options = {

    port: 10769,

    host: 'mqtt://m12.cloudmqtt.com',
```

```
    clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8),
    username: 'aaabhffz',
    password: 'EqeHDNIXN',
    keepalive: 60,
    reconnectPeriod: 1000,
    protocolId: 'MQIsdp',
    protocolVersion: 3,
    clean: true,
    encoding: 'utf8'
};
var client = mqtt.connect('mqtt://m12.cloudmqtt.com', options);


// function có chức năng subscribe 1 topic nếu đã kết nối thành công đến broker
client.on('connect', function() {
    console.log('Client A connected')
    // client subcribe topic /client-a/sub
    client.subscribe('/client-a/sub')
})
// function có chức năng gửi 1 gói tin đễn đến topic đã đăng kí
client.on('message', function(topic, message) {
    // in ra màn hình console 1 message ở định dạng string
    console.log(message.toString())
```

```
    // publish gói tin 'Hello from client A' đến topic /client-b/sub

    client.publish('/client-b/sub', 'Hello from client A')

    // đóng kết nối của client

    client.end()

})

console.log('Client A started')
```

JavaScript

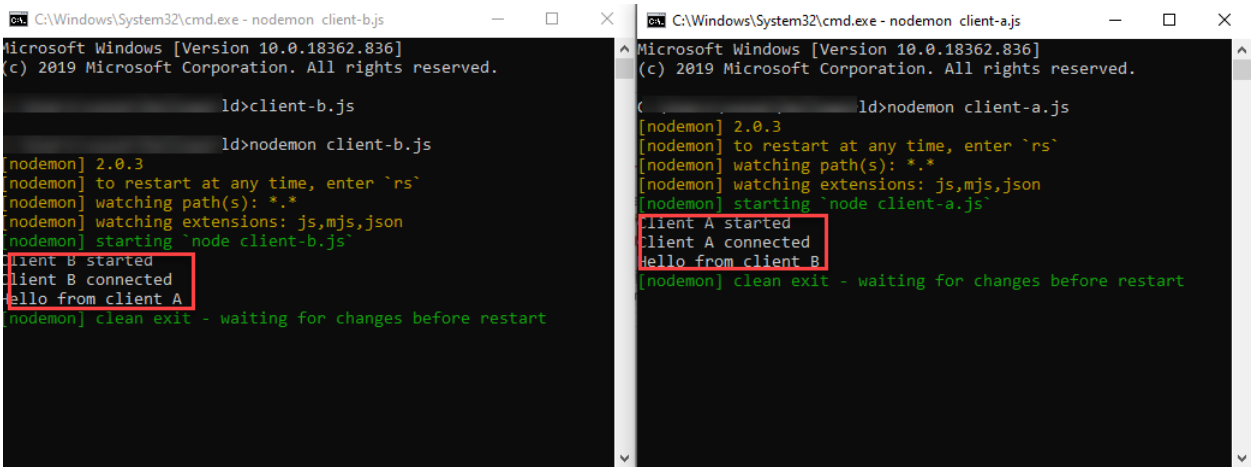
Chương trình client B

```
var mqtt = require('mqtt');

var options = {

    port: 10769,

    host: 'mqtt://m12.cloudmqtt.com',

    clientId: 'mqttjs_' + Math.random().toString(16).substr(2, 8),

    username: 'aaabhffz',

    password: 'EqeHDNIXN',

    keepalive: 60,

    reconnectPeriod: 1000,

    protocolId: 'MQIsdp',

    protocolVersion: 3,

    clean: true,

    encoding: 'utf8'

};
```

```
var client = mqtt.connect('mqtt://m12.cloudmqtt.com', options);



client.on('connect', function() {

    console.log('Client B connected')

    // client subcribe topic /client-b/sub

    client.subscribe('/client-b/sub')

    // publish gói tin 'Hello from client B' đến topic /client-a/sub

    client.publish('/client-a/sub', 'Hello from client B')

})



client.on('message', function(topic, message) {

    // in ra màn hình console 1 message ở định dạng string

    console.log(message.toString())

    // đóng kết nối của client

    client.end()

})

console.log('Client B started')
```

JavaScript

Kết quả



Ngoài ra, MQTT.js còn cung cấp thêm các lệnh để có thể tương tác với Broker thông qua terminal. Để làm được điều này, chúng ta cài đặt MQTT.js như một module toàn cục bằng cách sử dụng lệnh:

```
npm install mqtt -g
```

Bash

Bạn có thể kiểm tra bằng cách mở 2 màn hình terminal, ở màn hình 1 (tạm gọi là subscriber) sẽ subscribe vào topic tên là "topicA" bằng lệnh:

```
mqtt sub -t 'topicA' -h 'test.mosquitto.org' -v
```

Bash

Ở terminal thứ 2 (tạm gọi là publisher) thực hiện publish một tin nhắn với nội dung "hello subscriber" tới "topicA":

```
mqtt pub -t 'topicA' -h 'test.mosquitto.org' -m 'hello subscriber'
```

Bash

Các options:

- -t : MQTT topic, nơi sẽ thực hiện pushlish 1 message.
- -h : Xác định máy chủ sẽ kết nối đến.
- -m : Gửi 1 message dùng command line.-v : verbose, option cho phép ghi lại nhật kí hoạt động của các tập tin trong file cấu hình.

Để xem thêm các API khác trong MQTT.js, bạn có thể sử dụng lệnh: mqtt help [command].
Phần này chỉ là tham khảo cho anh em thôi nhé

ESP8266 MQTT Client

Có khá nhiều thư viện MQTT cho ESP8266 trên Arudino, ở đây mình dùng thư viện phổ biến là PubSubClient, bạn có thể tải thư viện này và bỏ vào trong thư mục library của Arduino

Chương trình

Chúng ta sẽ tạo một biến espClient thuộc lớp WiFiClient, biến này được khai báo là MQTT Client và sử dụng các thuộc tính của thư viện PubSubClient. Tại hàm `setup()` sẽ thiết lập ESP8266 ở chế độ station, kết nối đến mạng wifi. Bên cạnh đó hàm `setup()` cũng sẽ thực hiện chức năng tự động kết nối lại với MQTT Broker khi xảy ra mất kết nối đồng thời thực hiện các chức năng publish, subscribe của 1 MQTT Client thông qua hàm `reconnect()`. Hàm `callback()` có nhiệm vụ lấy dữ liệu của các puslisher trên topic đã subscribe và kiểm tra nội dung của message để điều khiển led ở GPIO2. Hàm `loop()` có chức năng kết nối Client là ESP8266 với Broker, thực hiện chức năng publish 1 message và subscribe topic. client.loop() sẽ kiểm tra thời gian kết nối của Client với gói KEEP_ALIVE để đưa ra các thông tin về trạng thái kết nối của ESP8266 đồng thời lấy dữ liệu của message từ buffer để gửi đến các Client đã subcribe topic.

```
#include <ESP8266WiFi.h>

#include <PubSubClient.h>

#define ssid "SCKT"      //Thay bằng wifi nhà bạn

#define password "huhuhu"

// Thông tin về MQTT Broker

#define mqtt_server "m12.cloudmqtt.com" // Thay bằng thông tin của bạn
```

```
#define mqtt_user "aaabhffz"    //Thay bằng user va pass của bạn
#define mqtt_pwd "EqeHDNIXN"
const uint16_t mqtt_port = 10769; //Port của CloudMQTT


WiFiClient espClient;
PubSubClient client(espClient);


void setup() {
  pinMode(2, OUTPUT);
  Serial.begin(115200);
  // hàm thực hiện chức năng kết nối Wifi và in ra địa chỉ IP của ESP8266
  setup_wifi();
  // cài đặt server là broker.mqtt-dashboard.com và lắng nghe client ở port 1883
  client.setServer(mqtt_server, mqtt_port);
  // gọi hàm callback để thực hiện các chức năng publish/subcribe
  client.setCallback(callback);
  // gọi hàm reconnect() để thực hiện kết nối lại với server khi bị mất kết nối
  reconnect();
}


void setup_wifi() {
  delay(10);
```

```
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);
  // kết nối đến mạng Wifi
  WiFi.begin(ssid, password);
  // in ra dấu . nếu chưa kết nối được đến mạng Wifi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // in ra thông báo đã kết nối và địa chỉ IP của ESP8266
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
void callback(char* topic, byte* payload, unsigned int length) {
  //in ra tên của topic và nội dung nhận được từ kênh MQTT lens đã publish
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
```

```
    Serial.print((char)payload[i]);
  }
  // kiểm tra nếu dữ liệu nhận được từ topic ESP8266/LED_GPIO2/status là chuỗi "on"
  // sẽ bậtled GPIO2, nếu là chuỗi "off" sẽ tắt led GPIO2
  if ((char)payload[0] == 'o' && (char)payload[1] == 'n') //on
    digitalWrite(2, LOW);
  else if ((char)payload[0] == 'o' && (char)payload[1] == 'f' && (char)payload[2] == 'f') //off
    digitalWrite(2, HIGH);
  Serial.println();
}
void reconnect() {
  // lặp cho đến khi được kết nối trở lại
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP8266", mqtt_user, mqtt_pwd)) {
      Serial.println("connected");
      // publish gói tin "Connected!" đến topic ESP8266/connection/board
      client.publish("ESP8266/connection/board", "Connected!");
      // đăng kí nhận gói tin tại topic ESP8266/LED_GPIO2/status
      client.subscribe("ESP8266/LED_GPIO2/status");
    } else {
```

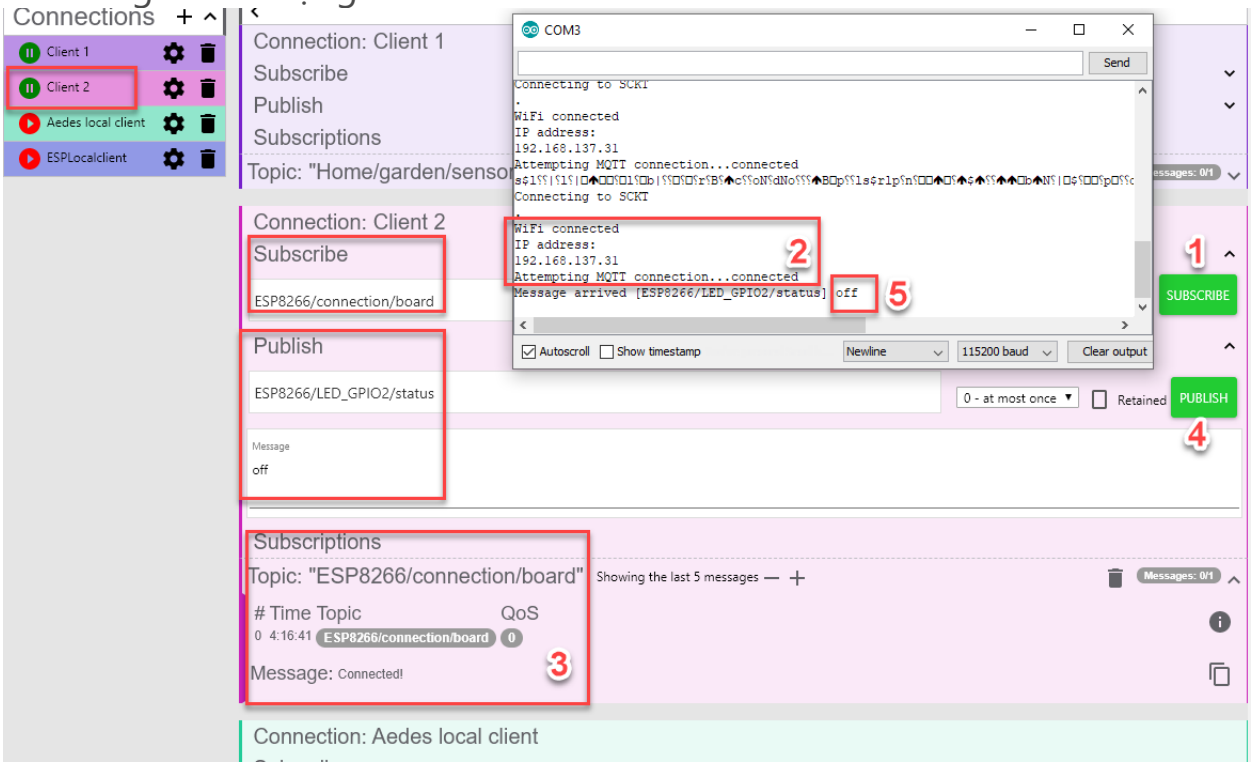
```
      // in ra màn hình trạng thái của client khi không kết nối được với MQTT broker
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // delay 5s trước khi thử lại
      delay(5000);
    }
  }
}
void loop() {
  // kiểm tra nếu ESP8266 chưa kết nối được thì sẽ thực hiện kết nối lại
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

Arduino

Kết quả

Mở MQTT lens trên trình duyệt Chrome, tạo 1 connection với host name `m12.cloudmqtt.com`, sử dụng port 10769. Thực hiện subscribe topic `ESP8266/connection/board`. Sau khi nhấn nút subscribe trên MQTT lens sẽ xuất hiện 1 message gửi từ esp8288 với nội dung connnected. Thực hiện pushlish các message vào topic `ESP8266/LED_GPIO2/status`. Nếu pushlish message với nội dung `on`, led GPIO2 trên board sẽ sáng, pushlish

message `off` led GPIO2 trên board sẽ tắt. Các message với nội dung khác thì vẫn sẽ hiển thị dữ liệu nhận được trên serial terminal của Arduino nhưng sẽ không có tác dụng điều khiển led GPIO2.



MQTT Broker

Các dịch vụ MQTT Broker hiện có miễn phí đều dùng để thử nghiệm, ta có thể sẽ phải trả phí dịch vụ với những ứng dụng lớn cần băng thông rộng và tốc độ đáp ứng nhanh, cộng với việc dữ liệu có thể bị tấn công do độ bảo mật thông tin chưa cao, chưa kể là một số thứ miễn phí sau thời gian sẽ bị quá tải và không còn đủ để ta dùng nữa. Do đó cần thiết tự xây dựng 1 MQTT Broker. Việc tự thiết lập 1 MQTT broker giúp chúng ta có thể sử dụng giao thức MQTT trên máy local mà không cần kết nối đến các dịch vụ MQTT Broker ở mạng internet. Quá trình truyền, nhận và xử lí dữ liệu diễn ra 1 cách nhanh chóng cũng như bảo mật thông tin của người dùng. Tuy nhiên, để tạo được 1 MQTT Broker với đầy đủ tính năng của giao thức MQTT đòi hỏi chúng ta phải có kiến thức tốt về giao thức MQTT cũng như các ngôn ngữ lập trình bổ trợ cho việc xây dựng nó. Để bắt đầu, ta sẽ tạo ra

1 MQTT Broker đơn giản bằng cách dùng 1 module hỗ trợ sẵn có đó là Aedes(Mosca bản cũ).

Aedes

Aedes(Mosca) là 1 trong số rất nhiều server MQTT Broker của giao thức MQTT. Tương tự như CloudMqtt, nó có 1 số đặc điểm như sau:

- Nó là 1 Node.js Broker, được viết bằng ngôn ngữ JavaScript vì vậy để có thể xây dựng MQTT Broker, chúng ta cần Node.js để chạy. Aedes(Mosca) có thể nhúng vào ứng dụng của bạn nếu ứng dụng này được viết bằng Node.js
- Aedes(Mosca) là 1 multi-transport MQTT Broker, có nghĩa là nó hỗ trợ tất cả các chức năng publish, subscribe của các broker khác. Danh sách các publish/subscribe broker được hỗ trợ bao gồm RabbitMQ, Redis, Mosquitto, ZeroMQ. Ở phần này chúng ta sẽ tạo ta 1 MQTT Broker đơn giản dùng Aedes(Mosca) với sự hỗ trợ của cơ sở dữ liệu Mongodb

Mục tiêu

- Chúng ta sẽ tạo 3 MQTT Client là:
    - ESP8266 subscribe trên topic `ESP8266/LED_GPIO2/status` và publish lên topic `ESP8266/connection/board`
    - MQTT Client Sub trên máy tính subscribe topic `ESP8266/LED_GPIO2/status` +`test` + `ESP8266/connection/board`,
    - MQTT Client Pub trên máy tính để publish message trên topic `test` và `ESP8266/LED_GPIO2/status`.
- Dùng Aedes tạo 1 MQTT Broker trên máy tính cá nhân nhằm broadcast messages đến các MQTT Client.

Code server Aedes rất đơn giản, chỉ có vài dòng như sau, sau khi server start lên ta sẽ có địa chỉ để truy cập vào là `localhost:1883`, hoặc `dia_chi_ip:1883` với `dia_chi_ip` ở đây là ip của máy mình, bạn có thể dùng lệnh ipconfig(win) và ifconfig(linux) để check. Của mình là 192.168.1.150 và mình dùng nó làm địa chỉ trong code của client

```
var aedes = require('aedes')()

var server = require('net').createServer(aedes.handle)

var port = 1883



server.listen(port, function () {

        console.log('server listening on port', port)

})
```

JavaScript

Code client subscribe Aedes, nhớ thay địa chỉ IP nha anh em

```
const mqtt = require("mqtt");



const client = mqtt.connect("mqtt://192.168.1.150:1883");



client.on("connect", ack => {

  console.log("connected!");

  // console.log(ack);

  client.subscribe("ESP8266/LED_GPIO2/status", err => {

    console.log(err);

  });

  client.subscribe("test", err => {

    console.log(err);

  });
```

```
  client.subscribe("ESP8266/connection/board", err => {
    console.log(err);
  });


  client.on("message", (topic, message) => {
    console.log(topic);
    // message is Buffer
    console.log(message.toString());
  });
});


client.on("error", err => {
  console.log(err);
});
```

JavaScript

Code client publish

```
const mqtt = require("mqtt");


const client = mqtt.connect("mqtt://192.168.1.150:1883");


client.on("connect", ack => {
        console.log("connected!");
```

```
        setInterval(_ => {
                client.publish("test", "Hello mqtt " + new Date().getTime());
        }, 3000);

        setInterval(_ => {
                client.publish("ESP8266/LED_GPIO2/status", "on");
        }, 8000);
});

client.on("error", err => {
        console.log(err);
});
```

JavaScript

Code client Aedes với ESP8266

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#define ssid "SCKT"
#define password "XXXX"
// Thông tin về MQTT Broker
#define mqtt_server "192.168.1.150" // Thay bằng thông tin IP của bạn
//#define mqtt_user "aaabhffz"    //Nếu có user và pass thì uncomment ra dùng
```

```
//#define mqtt_pwd "EqeHDNIXN"

const uint16_t mqtt_port = 1883; //Port của MQTT broker



WiFiClient espClient;

PubSubClient client(espClient);



void setup() {

  pinMode(2, OUTPUT);

  Serial.begin(115200);

  // hàm thực hiện chức năng kết nối Wifi và in ra địa chỉ IP của ESP8266

  setup_wifi();

  // cài đặt server là broker.mqtt-dashboard.com và lắng nghe client ở port 1883

  client.setServer(mqtt_server, mqtt_port);

  // gọi hàm callback để thực hiện các chức năng publish/subcribe

  client.setCallback(callback);

  // gọi hàm reconnect() để thực hiện kết nối lại với server khi bị mất kết nối

  reconnect();

}



void setup_wifi() {

  delay(10);

  Serial.println();
```

```
  Serial.print("Connecting to ");
  Serial.println(ssid);
  // kết nối đến mạng Wifi
  WiFi.begin(ssid, password);
  // in ra dấu . nếu chưa kết nối được đến mạng Wifi
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // in ra thông báo đã kết nối và địa chỉ IP của ESP8266
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}
void callback(char* topic, byte* payload, unsigned int length) {
  //in ra tên của topic và nội dung nhận được từ kênh MQTT lens đã publish
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
```

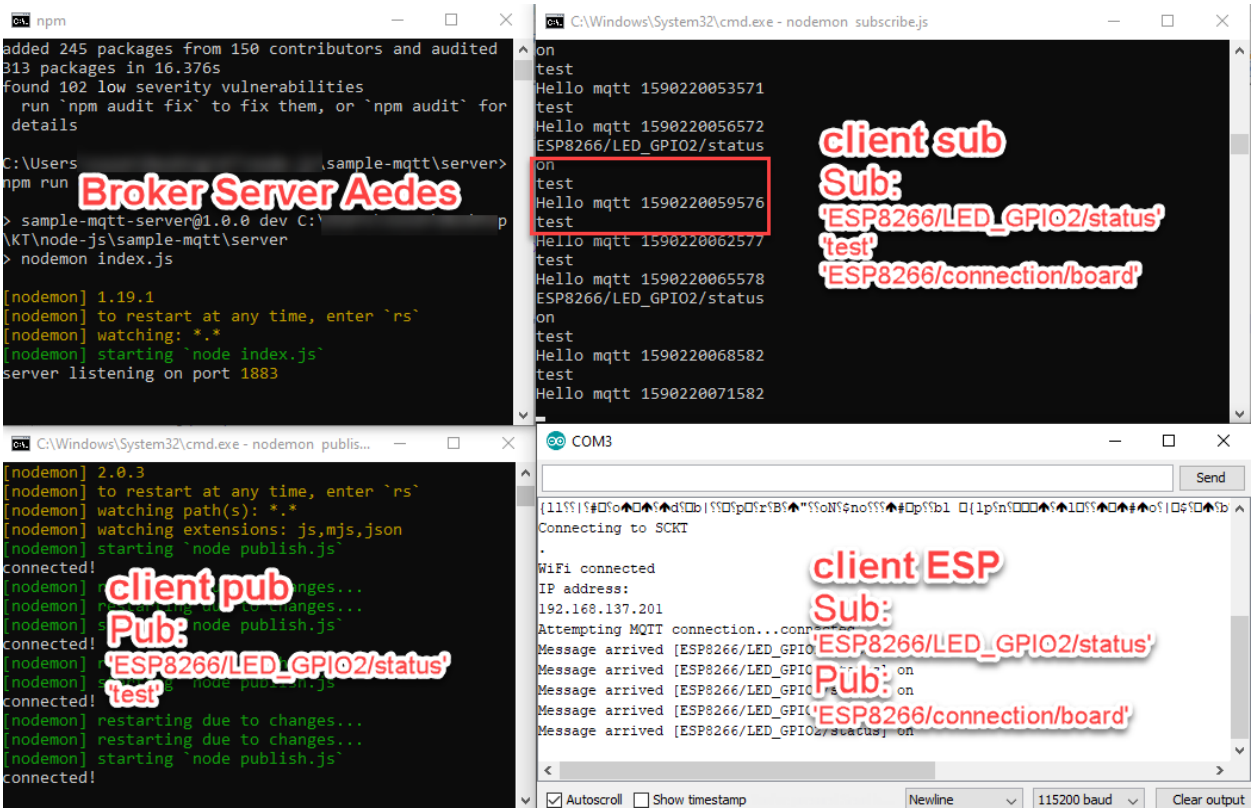
```
  }

  // kiểm tra nếu dữ liệu nhận được từ topic ESP8266/LED_GPIO2/status là chuỗi "on"

  // sẽ bậtled GPIO2, nếu là chuỗi "off" sẽ tắt led GPIO2

  if ((char)payload[0] == 'o' && (char)payload[1] == 'n') //on

    digitalWrite(2, LOW);

  else if ((char)payload[0] == 'o' && (char)payload[1] == 'f' && (char)payload[2] ==
'f') //off

    digitalWrite(2, HIGH);

  Serial.println();

}

void reconnect() {

  // lặp cho đến khi được kết nối trở lại

  while (!client.connected()) {

    Serial.print("Attempting MQTT connection...");

    // Nếu có user pass thì dùng code đã comment

    //if (client.connect("ESP8266", mqtt_user, mqtt_pwd)) {

    if (client.connect("ESP8266")) {

      Serial.println("connected");

      // publish gói tin "Connected!" đến topic ESP8266/connection/board

      client.publish("ESP8266/connection/board", "Connected!");

      // đăng kí nhận gói tin tại topic ESP8266/LED_GPIO2/status

      client.subscribe("ESP8266/LED_GPIO2/status");
```

```
    } else {
      // in ra màn hình trạng thái của client khi không kết nối được với MQTT broker
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // delay 5s trước khi thử lại
      delay(5000);
    }
  }
}
void loop() {
  // kiểm tra nếu ESP8266 chưa kết nối được thì sẽ thực hiện kết nối lại
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

Arduino

Kết quả

4 khung cửa sổ server client và các mô tả publish/subscribe cho anh em quan sát



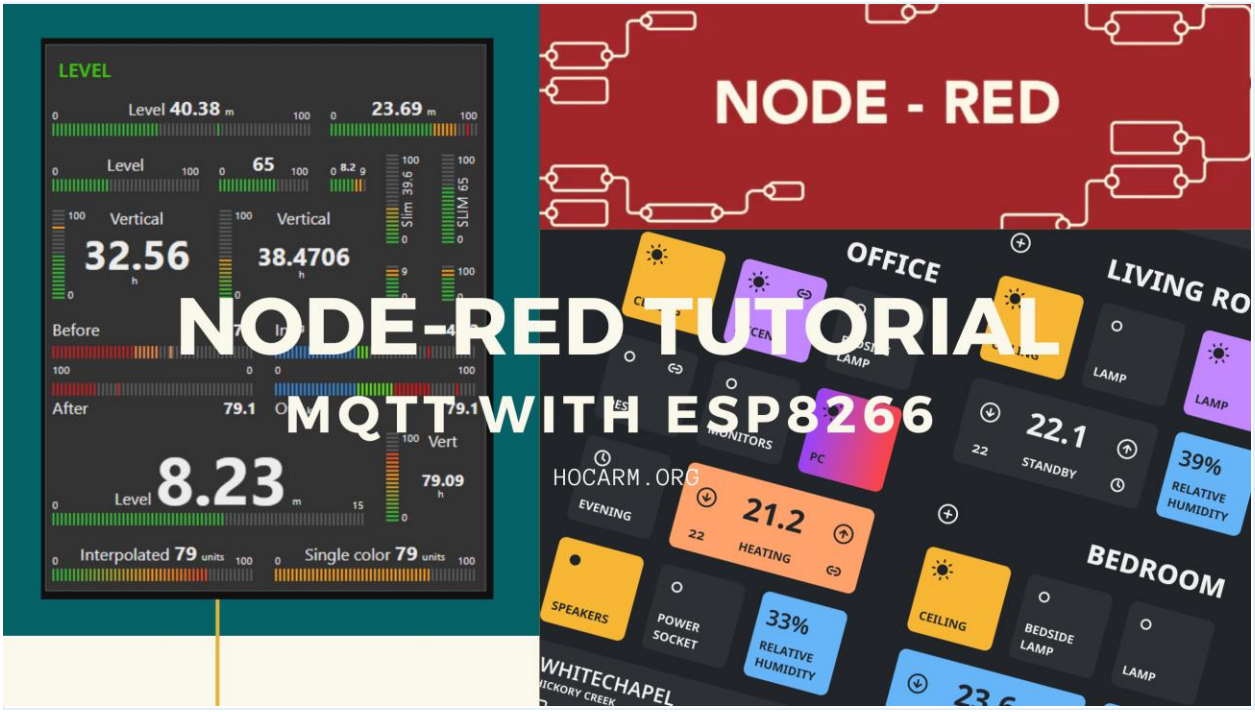
Mosquitto

Mình có bài viết cài đặt mosquitto với ubuntu ở đây các bạn có thể tham khảo

Node-red MQTT và ESP8266 | Học ARM

Node-red MQTT và ESP8266phối hợp với nhau thì sẽ như thế nào ? Trong bài viết này mình sẽ hướng dẫn về cách để tạo broker MQTT để sử dụng ở local, dùng Node-red tạo giao diện điều khiển ON/OFF cho LED trên NodeMCU(ESP8266) qua MQTT Bắt đầu Để bắt đầu thì mình nghĩ chúng ta nên xem lại bài viết về M...

SCHọc ARM



Mosquitto là 1 MQTT Broker viết bằng ngôn ngữ lập trình C. Một số đặc điểm nổi bật của mosquitto là tốc độ truyền nhận và xử lí dữ liệu nhanh, độ ổn định cao, được sử dụng rộng rãi và phù hợp với những ứng dụng embedded. Thích hợp cho các hệ thống nhỏ chạy trên máy local như Raspberry Pi, bên cạnh đó Mosquitto cũng được hỗ trợ các giao thức TLS/SSL (các giao thức nhằm xác thực server và client, mã hóa các message để bảo mật dữ liệu).
Một số nhược điểm của mosquitto là khó thiết kế khi làm những ứng dụng lớn và ít phương thức xác thực thiết bị nên khả năng bảo mật vẫn chưa tối ưu.

EMQ

EMQ (Erlang MQTT Broker) là một MQTT Broker được viết bằng ngôn ngữ lập trình Erlang. Ưu điểm của EMQ là tính ổn định cao, thích hợp để thiết kế các hệ thống lớn do khả năng mở rộng ứng dụng dễ dàng cũng như khá dễ để cài đặt. Ngoài ra EMQ còn hỗ trợ nhiều phương thức xác thực người

dùng, phát triển và cập nhật tính năng liên tục bởi cộng đồng developer. Tuy nhiên điểm yếu của MQTT broker này là khó đối với những người mới bắt đầu. Thông tin về EMQ có thể xem tại trang https://www.emqx.io/

Code tham khảo của bài viết các bạn có thể xem tại

hocarm/nodejs-socketio-tutorial

Contribute to hocarm/nodejs-socketio-tutorial development by creating an account on GitHub.

hocarmGitHub

Tạm kết

Từ những nội dung đã trình bày ở trên, chúng ta phần nào hiểu rõ về cách thức hoạt động của giao thức MQTT cũng như vai trò của nó trong các ứng dụng IoT. Ngoài ra ta có thể hiểu được cách giao tiếp của MQTT Client và MQTT Server, cách dùng các công cụ để kiểm tra kết nối MQTT, tạo ra server cũng như client để bắn MQTT cho nhau, cuối cùng là dùng ESP8266 như một client để pub/sub data lên server. Thật tuyệt vời phải không các bạn.