



Written by **Casper Boutens** on 24 April 2016

How to make an MQTT broker/server with node and connect it to a device (in this case a nodeMCU)

In this guide, I will show you how to make an MQTT broker/server that is able to receive and send data from and to a device.

The guide will be split into two content types. One is the 'how' type. This one will be shown in the table of contents without the '*' behind it and will tell you how to do that step. The other type is the 'why' type. This one will be shown with a '*' behind it and it will tell you why I choose to do something or what something really does. The parts with the '*' behind it won't be essential to make an MQTT broker/server work but I highly recommend reading it. It's important to understand the reason why you need to do something.

Table of content

- General
 - [What do you need](#)
- MQTT broker/server
 - [What is MQTT? *](#)
 - [How does MQTT work? *](#)
 - [Why MQTT over get and post? *](#)
 - [Why am I typing broker/server all the time? *](#)

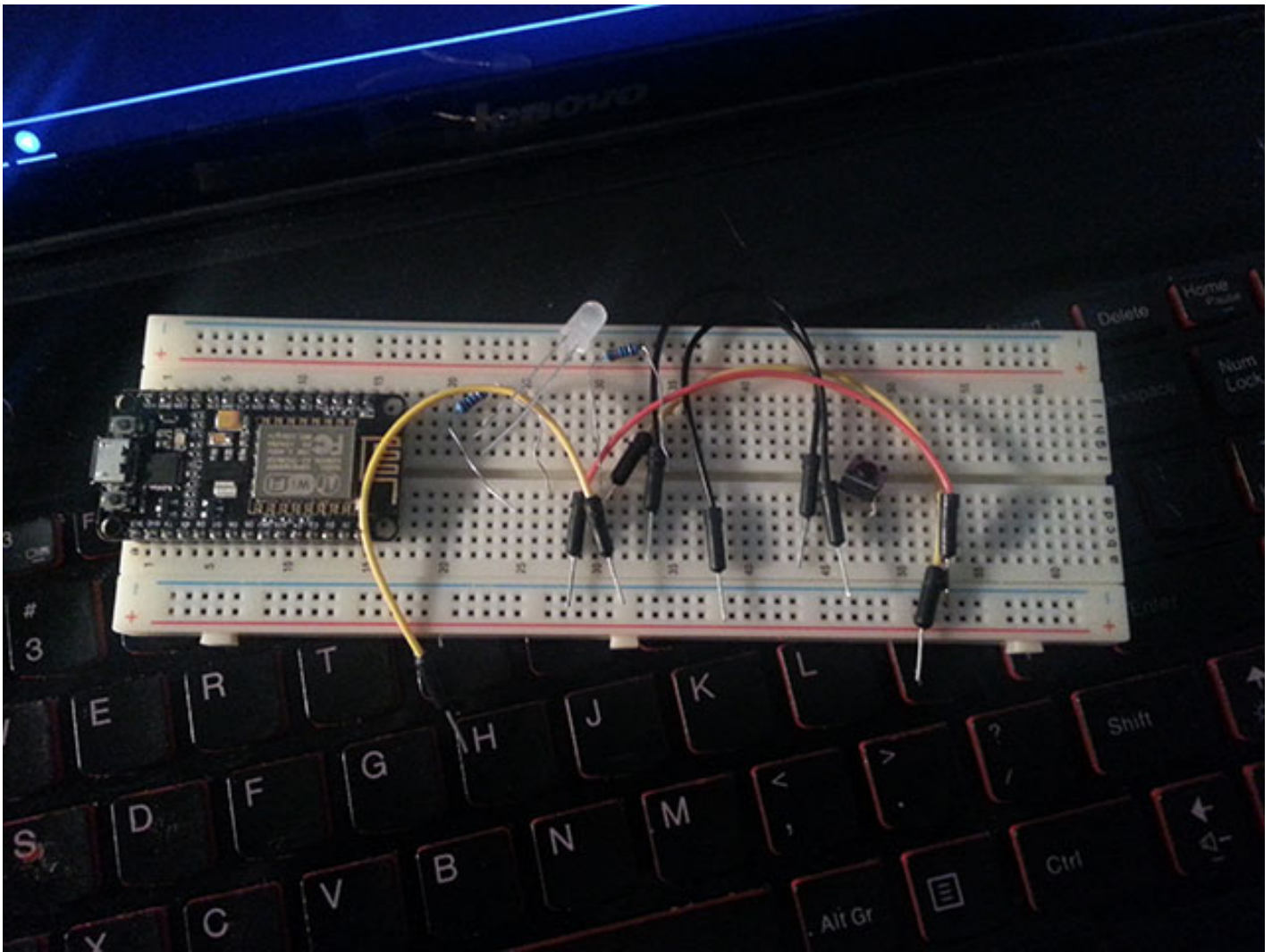
- [Installing nodeJS](#)
- [Why NodeJS ? *](#)
- [Creating the folder to work from](#)
- [What is Mosca ? *](#)
- [Install Mosca](#)
- [Setting up the broker/server](#)
- [Publish a command](#)
- [Start the server](#)
- [nodeMCU](#)
 - [Install the arduino IDE](#)
 - [Setting up the arduino IDE](#)
 - [Connecting the hardware *](#)
 - [Including the libraries that we need](#)
 - [Connect to the WiFi](#)
 - [Connect to the MQTT broker/server](#)
 - [Subscribe to a topic](#)
 - [Activating the light and the button *](#)

What do you need

The thinks that you need to make a barebone system are:

- [A nodeMCU](#) (or an Arduino with a WIFI bundle)
- [A micro to normal USB kable](#)
- [Five Man to man wires](#) (or man to female if you don't use a breadboard)
- [A small resistor](#)
- [A breadboard *](#)
- [An LED *](#)
- [A button *](#)
- [NodeJS](#)
- [Sublime](#) (or any other text editor)
- [The Arduino IDE](#)

The parts with the * are optional, but I use them. More about the products and how to install / use them later.



What is MQTT? *

MQTT stands for MQ Telemetry Transport. It is a publish/subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimise network bandwidth and device resource requirements whilst also attempting to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

(source <http://mqtt.org/faq>)

How does MQTT work? *

I'm not going in the deep functionality of MQTT. The MQTT protocol works on a publish and subscribe principle. Devices that want to be informed about certain topics ('topic' is the term for what a device subscribes on) subscribe to that topic on the server. This lets the server know when the device wants data and what date it wants. When the client (the device) or server want to give information they publish it.

For example, the device says the to MQTT broker/server: 'I want to turn my LED on when the user clicks on a button on the website'. The server will put the device in a 'room' where all the devices are that want to know about that topic. When the server receives data about the user that clicks from a website it can publish the data to all the devices that want to know because the server knows which devices wants what.

Why MQTT over get and post? *

I mean the HTTP protocol with the GET and the POST. That method works as follows. When something needs to be sent to the HTTP server, the device will make a POST request with the data. This will be on a route so that the HTTP server knows what to do with it. This is not the best method because HTTP headers (every GET and POST has a header with information about the request) are relatively big. It gets worse. The server can't send data to the device because it has no idea about the device existence. This is way the device has to 'ping' the server once per x time. This 'ping' will ask the server if there is a change or something and the server will respond with a response that holds the data.

Why am I typing broker/server all the time? *

I must be honest. There is no server with the MQTT protocol. It's called a broker. The reason why I'm typing broker/server is because when I started learning

about MQTT I was searching for 'how to build an MQTT server' and when I saw an MQTT broker I was like: 'iewl, scary, I want a server, not a weird broker'.

That is why I'm calling it a broker/server.

Installing nodeJS

If you already have NodeJS on your computer, return; or in other words. You can skip this step.

Installing NodeJS is really easy. Just go to the site <https://nodejs.org/en/> and click on download the stable version. When it's downloaded you open the file and follow the installation process. If you are on a windows PC / laptop, check the box with the text 'add to PATH'. This is needed for NPM and other programs that work with NodeJS. If you don't you will get really annoying bugs later on.

Why NodeJS ? *

There are a lot of programming languages that are able to function as a broker for MQTT. The reason why I choose NodeJS is because it's Javascript and I like Javascript and I'm already familiar with NodeJS. There is not really anything else. This is totally up to you, although, if you are reading this guide you already decided on NodeJS. So let's move on!

Creating the folder to work from

We need to have a place to put the code before we can code. I created a folder named 'MQTT broker' and in that folder I created a file called 'mqttBroker.js'. That is really it for the file setup.

What is Mosca ? *

Mosca is a broker module that we are going to use as the base of our MQTT broker. This way we don't have to reinvent the wheel and we are lucky about that. I wouldn't even know where to start if I had to write a low-level MQTT broker. Mosca made that nice and easy for us.

Install Mosca

Open your terminal and go to the folder of this project. Once there, type this command:

```
npm install  
npm install mosca
```

This will create a folder named `node_modules`. In that folder are all kinds of files that NodeJS needs to run plus Mosca.

Setting up the broker/server

We can finally start coding now that everything is ready. So let's do the broker/server setup. First, I will post a lot of code and then I will explain what it does.

```
var mosca = require('mosca');  
  
var server = new mosca.Server({  
  host: '192.168.0.19',  
  port: 3000  
});  
  
server.on('clientConnected', function(client) {  
  console.log('client connected', client.id);  
});  
  
server.on('clientDisconnected', function(client) {
```

```
    console.log('client disconnected', client.id);
  });

  server.on('published', function(packet, client) {
    console.log(packet);
  });

  server.on('subscribed', function(topic, client) {
    console.log('subscribed: ' + client.id);
  });

  server.on('unsubscribed', function(topic, client) {
    console.log('unsubscribed: ' + client.id);
  });

  server.on('ready', function() {
    console.log('Mosca server is up and running');
  });
```

The first line of code is to import the module that we installed named Mosca. This is so that we can use that module as a variable. I called it mosca.

With the mosca module, I create a new variable that contains a server that listens to that IP and that port. These are important later when we are going to connect the nodeMCU to the server.

The rest is self-explanatory if you know what 'on' does. 'on' is a handler that listens to events. If a client connects to the server it will fire the .on('clientConnected') event.

Publish a command

I can write this part because I know what I want. Normally you would do this after connecting the device to the server and testing if everything works.

I will make a command that tells the device to toggle its LED on and off. This is wrapped in a simple setInterval function so that the server does this over and over again.

Like I typed, I already know what I want to do. I'm going to give the commands a number code. This way it's short, so it doesn't take much bandwidth and it's easy to document.

I want 001 to be LED on and 002 to be LED off.

```
...  
  
var ledCommand = '001';  
  
setInterval(function() {  
  ledCommand = (ledCommand === '001') ? '002' : '001';  
  server.publish({topic: 'LEDToggle', payload: ledCommand});  
}, 1000);
```

On the first line, I create a ledCommand this holds the value that the command should be. Every second I toggle between 001 and 002 and send it to all the devices that are subscribed to the topic 'LEDToggle'.

Start the server

To start the server that we just created type `node mqttBroker.js` in your terminal.

```
node mqttBroker.js
```

It should say: 'Mosca server is up and running' and after Every second you will get a `console.log` about the publish that you did.


```
C:\Windows\system32\cmd.exe - node mqttBroker.js
payload: 1,
messageId: 'HyRkhISl',
qos: undefined,
retain: undefined }
{ topic: 'LEDToggle',
  payload: 2,
  messageId: '8k1l28ql',
  qos: undefined,
  retain: undefined }
{ topic: 'LEDToggle',
  payload: 1,
  messageId: 'S1llhLcg',
  qos: undefined,
  retain: undefined }
{ topic: 'LEDToggle',
  payload: 2,
  messageId: 'HyZghISe',
  qos: undefined,
  retain: undefined }
{ topic: 'LEDToggle',
  payload: 1,
  messageId: 'SyEeh89e',
  qos: undefined,
  retain: undefined }
{ topic: 'LEDToggle',
  payload: 2,
  messageId: 'HkHxnU9x',
  qos: undefined,
  retain: undefined }
```

Install the Arduino IDE

As easy as the NodeJS install, go to the website of Arduino

<https://www.arduino.cc/en/Main/Software> and download the Arduino Software for your operating system. After it's downloaded, open the file and follow the install wizard.

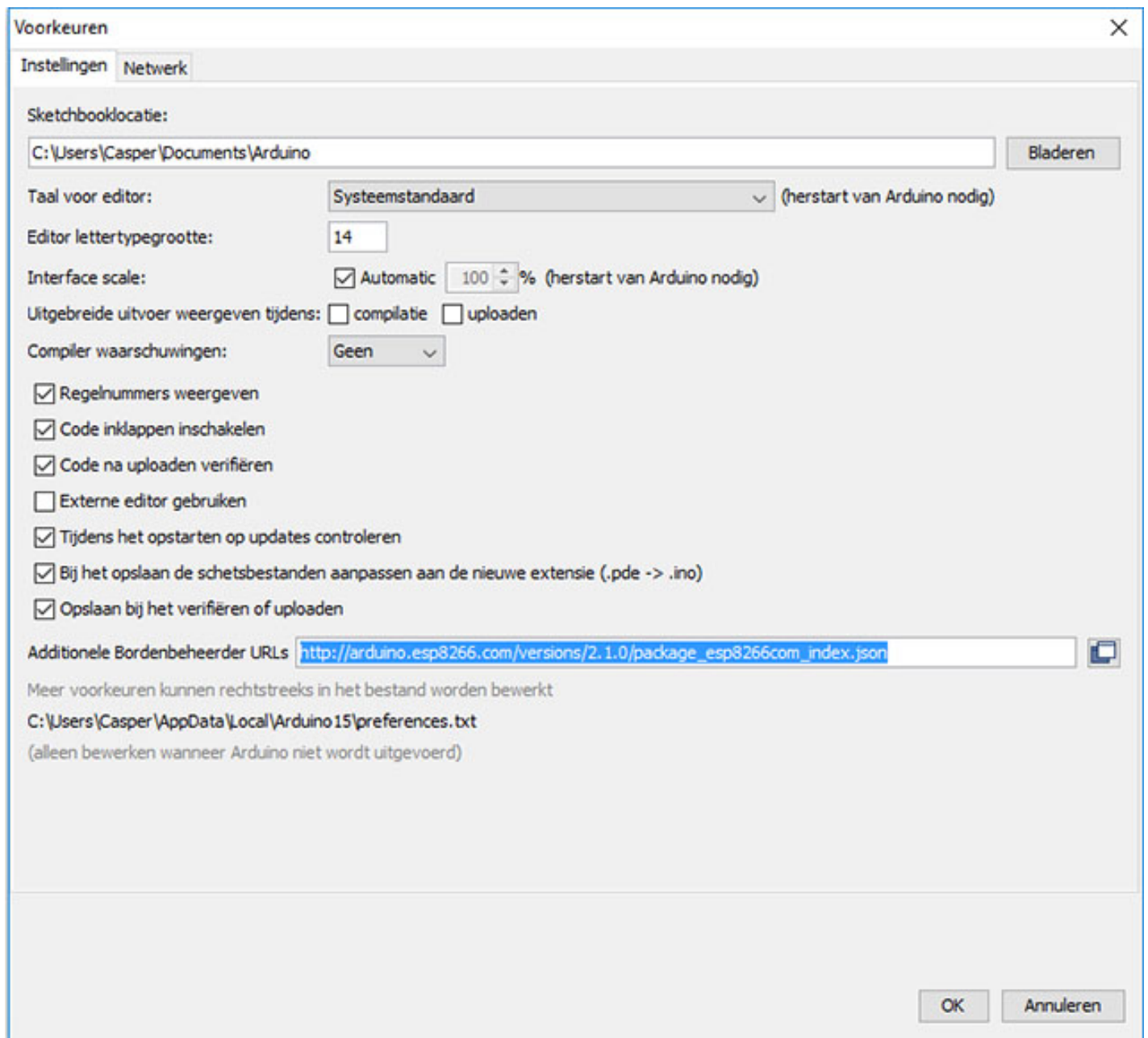
Setting up the Arduino IDE

We do need to do some extra work to get the nodeMCU working.

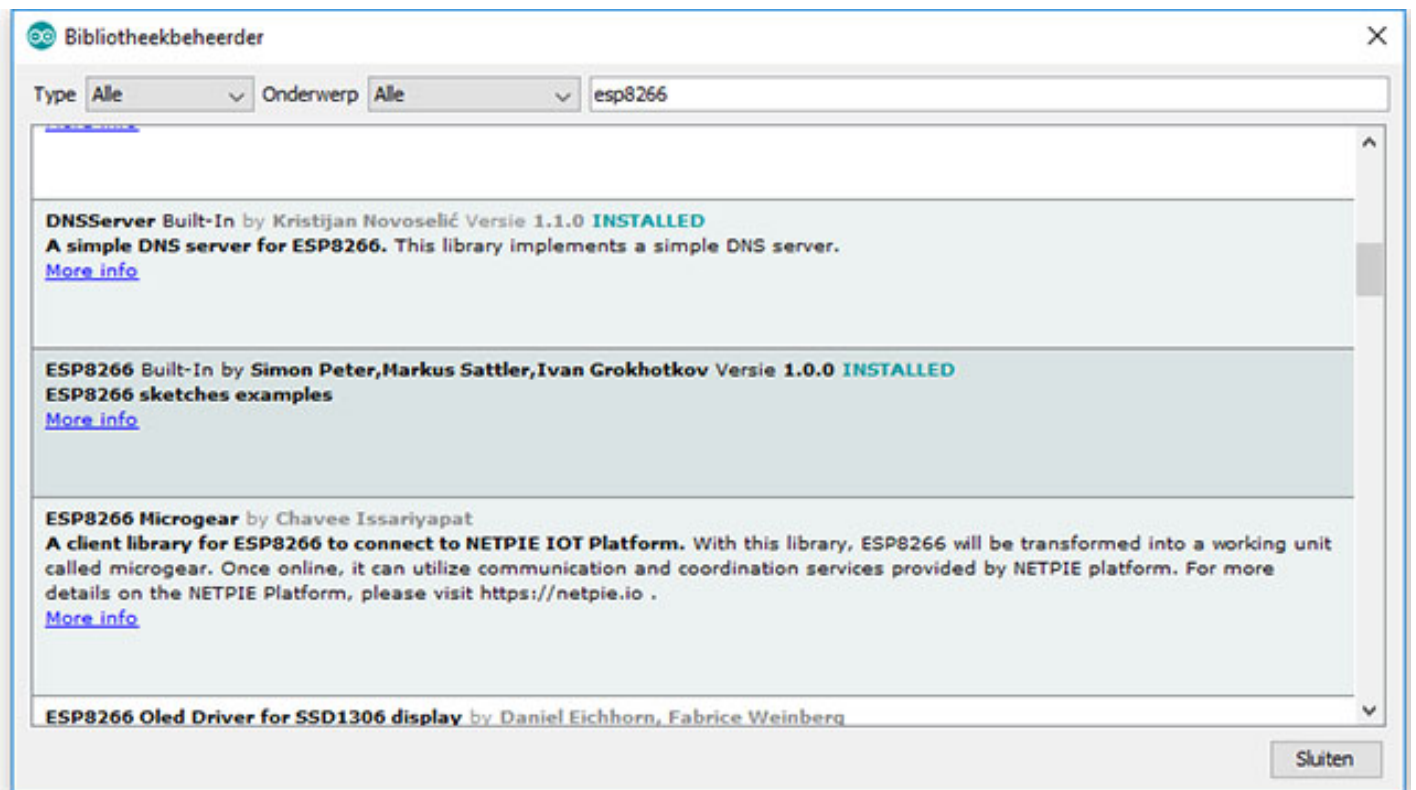
Start the Arduino IDE and go to 'Arduino -> preferences'. In the input field of 'Additional Boards Manager URLs add this link:

http://arduino.esp8266.com/versions/2.2.0/package_esp8266com_index.json.

This will give the Arduino IDE the information needed to work with the nodeMCU.



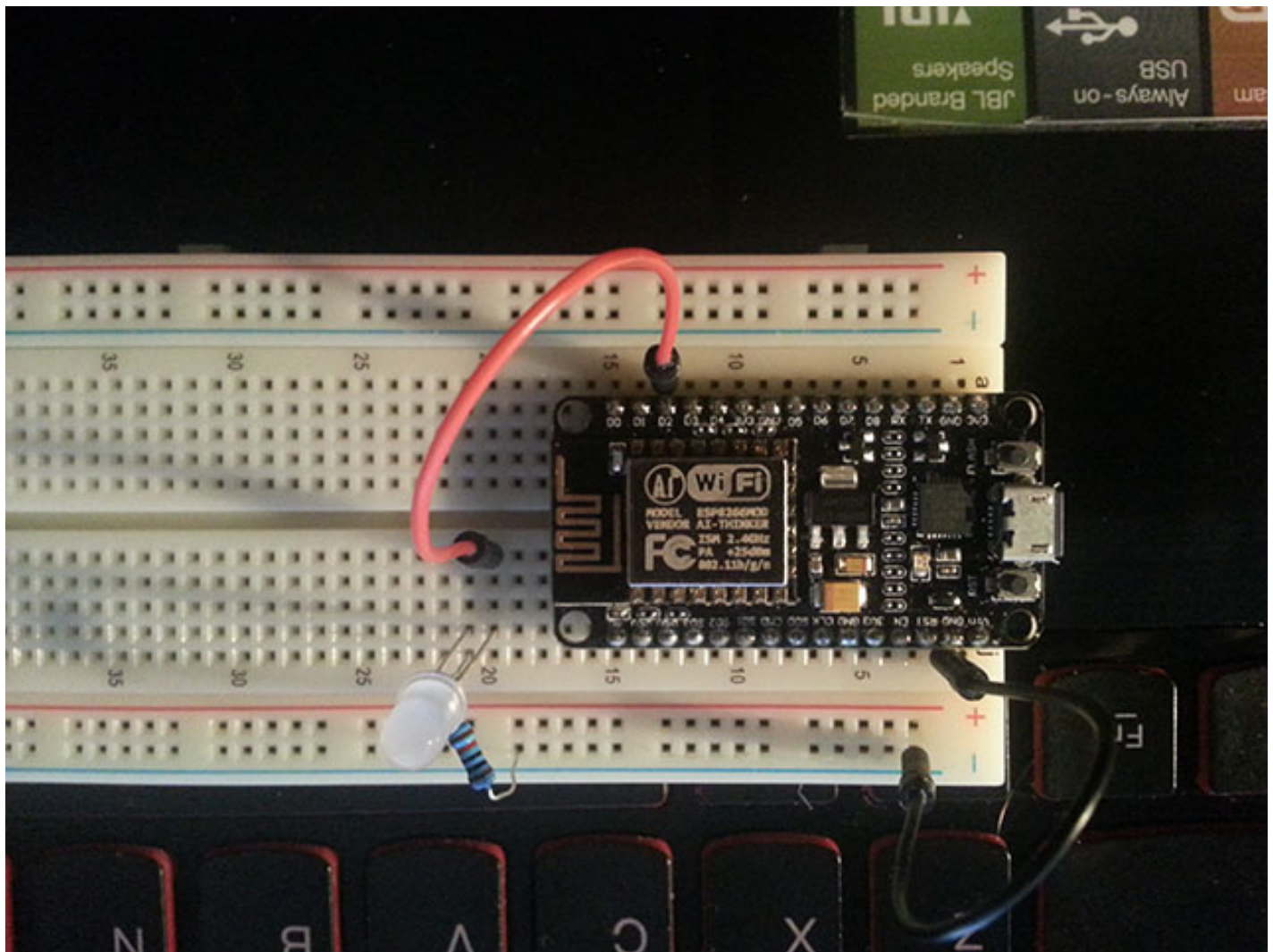
Next we have to install the Chipset. Go to 'tools -> Board -> Boardmanager' and type in the search bar esp8266 and press install. Restart the IDE and everything is ready!



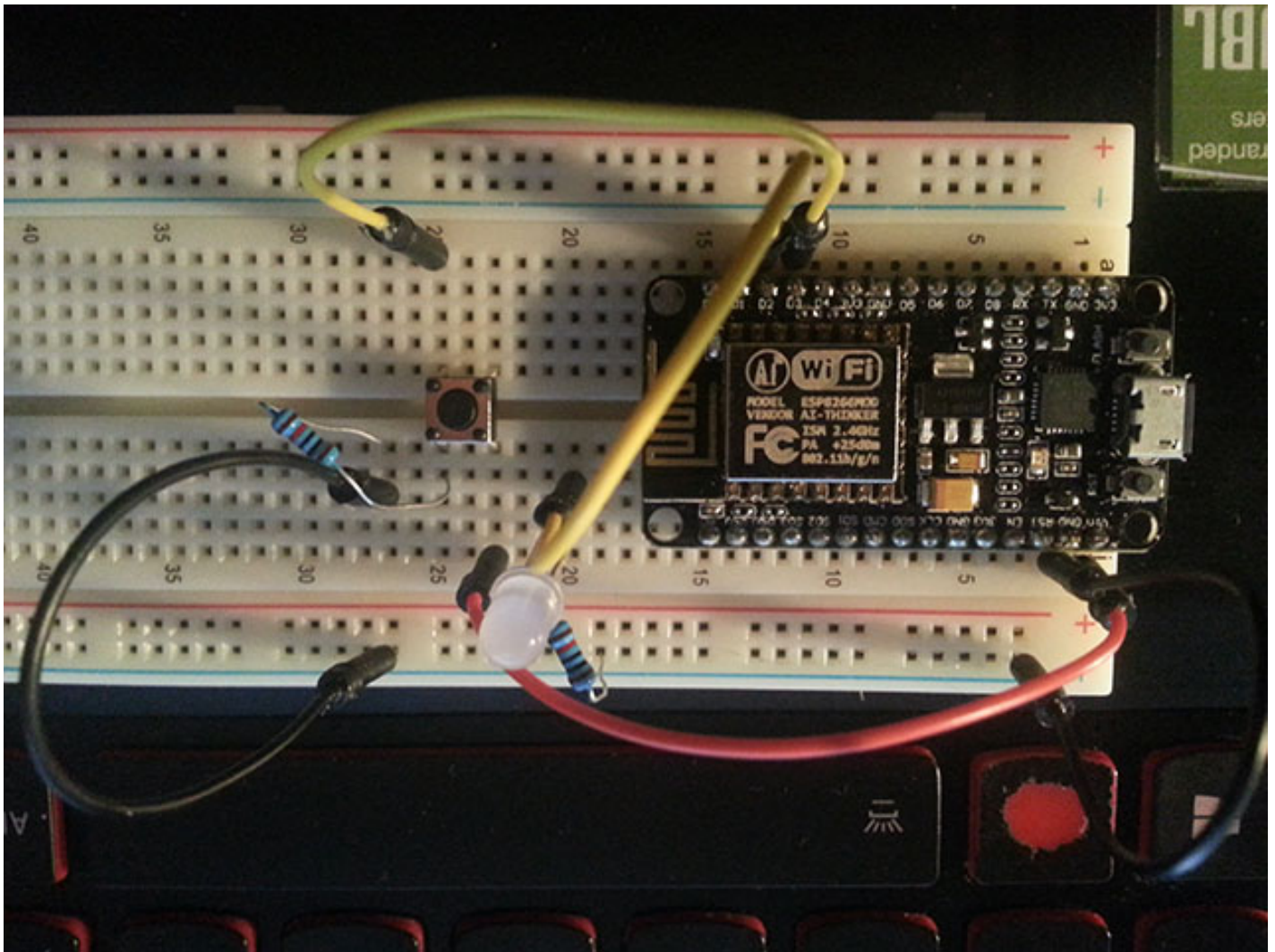
Connecting the hardware *

This is not a required step to follow because I will only connect an LED and a button to show that the system is working. You will probably need some other stuff like a sensor or something.

Both are easy to connect. Let's start with the LED. Connect the long leg of the LED to the D2 (or a digital pin if you are not using the nodeMCU). Then, connect the short leg to the resistor and the resistor to the ground.



For the button, you need to have one leg go to the 5V (vim). Another leg to the ground with a resistor between it, and one leg to the D3.



Including the libraries that we need

I'm going to presume that you have the basic understanding of Arduino IDE because MQTT is not the first project that you pickup. If it is, I highly recommend that you do some tutorials on the [Arduino website](#).

We have to include (it's the same thing as require in Javascript) for the WiFi and MQTT to work. That looks something like this:

```
sketch_apr24a | Arduino 1.6.8
Bestand  Bgwerken  Schets  Hulpmiddelen  Help

sketch_apr24a $
1 //include Wifi library
2 #include <ESP8266WiFi.h>
```

```
3 //include MQTT library
4 #include <PubSubClient.h>
5
6 void setup() {
7     // put your setup code here, to run once
8
9 }
10
11 void loop() {
12     // put your main code here, to run repeatedly
13
14 }
```

Opslaan voltooid.

14

NodeMCU 1.0 (ESP-12E Module), 80 MHz, Serial, 115200, 4M (3M SPIFFS) on COM8

Sorry but I can't use my code syntax library that I use for the Javascript because C code is so different.

Connect to the WiFi

The device must be connected to the internet to actually get to the server. That is why I wanted to use the nodeMCU. That device has a WiFi module in it.

```
//initialize the mqttClient library
WiFiClient wifiConnection;

...
//wifi settings
const char* wifiSsid = "Osso";
const char* wifiPassword = "twijfelachtig123";
...
void setup() {
  // put your setup code here, to run once
  Serial.begin(115200);

  //print what the programme is connecting to
  Serial.print("Connecting to ");
  Serial.println(wifiSsid);

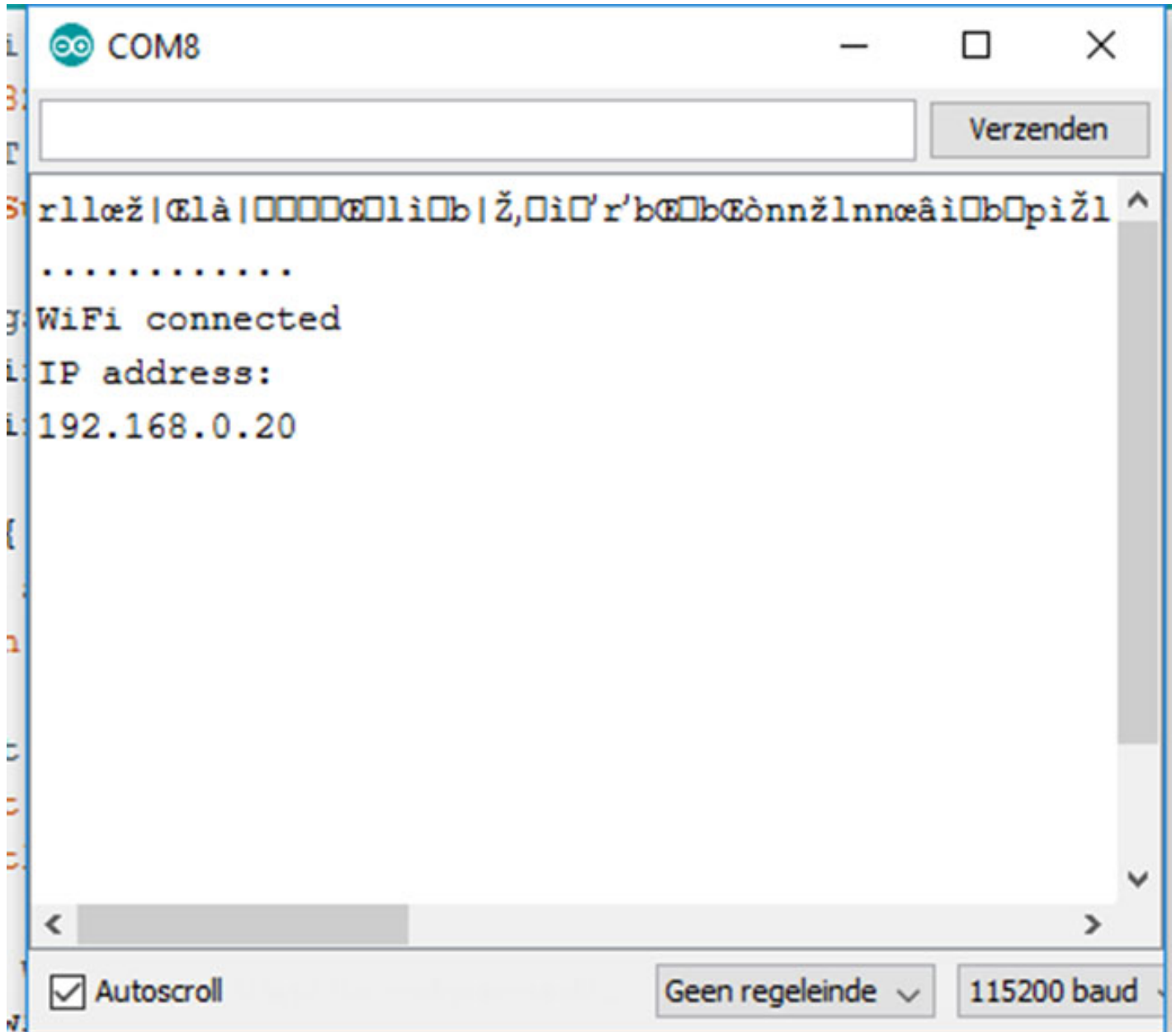
  //start the wifi library by connection to the wifi
  WiFi.begin(wifiSsid, wifiPassword);

  //while the wifi library is not fully connected
  while (WiFi.status() != WL_CONNECTED) {
    //print '.' every 0.5s
    delay(500);
    Serial.print(".");
  }

  //print that the wifi library is connected and the IP
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  ...
}
```

The code is surprisingly simple. I create two const chars for the wifi SSID (name) and password so that I don't have to hardcode stuff in. And with WiFi.begin I

signal that I want to connect to the WiFi. The rest is all Serial.print data to show that it works.



Connect to the MQTT broker/server

After the device is connected to the WiFi, it can connect to the MQTT broker/server. This is not a lot harder. There is, however, one thing that is a bit different and that is the callback. The callback is a function that is called when the device gets a publish. but again, it is setting the const chars and connect, then wait until the connection is completed. The IP must be the IP of your host computer.


```

...
//initialize the mqtt library
PubSubClient mqttClient(wifiConnection);
...
//mqtt settings
const char* mqttHost = "192.168.0.19";
const int mqttPort = 3000;
...
void setup() {
    ...
    //connect to the wifiConnection with mqtt
    mqttClient.setServer(mqttHost, mqttPort);

    //set the callback when the programme recieves a callback
    mqttClient.setCallback(callback);

    while (!mqttClient.connected()) {
        Serial.print("Attempting MQTT connection...");
        if (mqttClient.connect("device1")) {
            Serial.println("connected");
        } else {
            Serial.print("failed, rc=");
            Serial.print(mqttClient.state());
            Serial.println(" try again in one second");
            // Wait a second before retrying
            delay(1000);
        }
    }
}
...
void callback(char *topic, byte *payload, unsigned int length) {

    char message_buff[3];
    int i = 0;

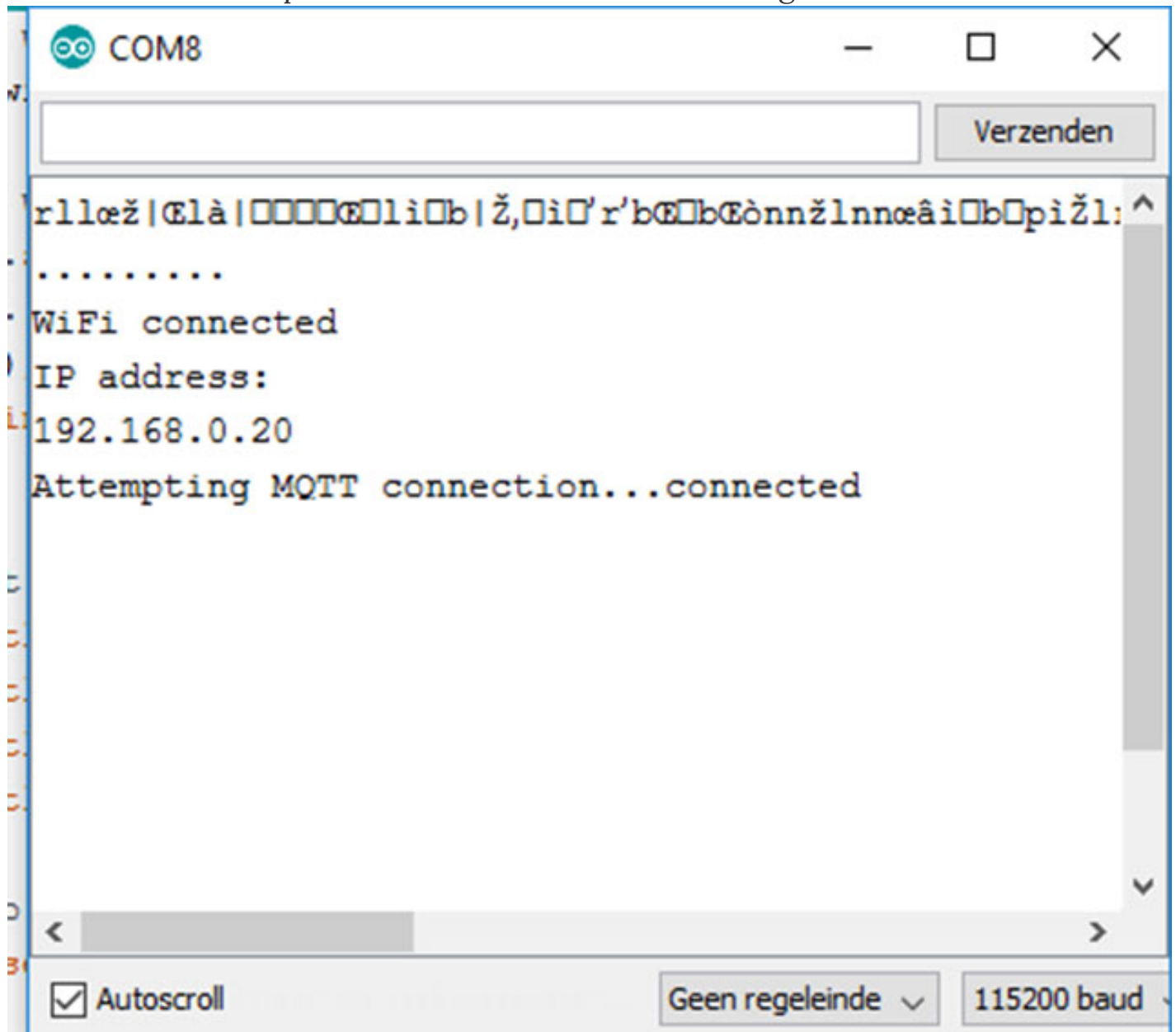
    //for loop that loops through the payload array
    for(i = 0; i < length; i++) {
        message_buff[i] = payload[i];
    }

    //\0 is null, this makes it that the String typecast understands where to stop
    message_buff[i] = *"\0";

    Serial.println(String(message_buff));
}

```

Like I typed. The code is really similar to the WiFi connect. The callback function is a bit complicated for this guide. If you want to know more about that stuff you should read this article. Let's look what the outcome is. If everything went well, the console should print out that the connections are good.



The screenshot shows a serial monitor window titled "COM8". At the top, there is a text input field and a button labeled "Verzenden". The main area displays the following text in a monospaced font:

```
.....  
WiFi connected  
IP address:  
192.168.0.20  
Attempting MQTT connection...connected
```

At the bottom of the window, there is a horizontal scrollbar, a checkbox labeled "Autoscroll" which is checked, a dropdown menu showing "Geen regeleinde", and a label "115200 baud".

At this point you should see a different message in your terminal because a client has connected!

```
Selecteren C:\Windows\system32\cmd.exe
qos: undefined,
retain: undefined }
{ topic: 'LEDToggle',
  payload: 2,
  messageId: 'H1CNkv5x',
  qos: undefined,
  retain: undefined }
{ topic: 'LEDToggle',
  payload: 1,
  messageId: 'HyySFDqe',
  qos: undefined,
  retain: undefined }
{ topic: 'LEDToggle',
  payload: 2,
  messageId: 'B1lBKDqg',
  qos: undefined,
  retain: undefined }
client connected device1
{ topic: '$SYS/rkchaH5x/new/clients',
  payload: 'device1',
  messageId: 'HJWBFD9l',
  qos: undefined,
  retain: undefined }
{ topic: 'LEDToggle',
  payload: 1,
  messageId: 'rJxbSKPql',
  qos: undefined,
  retain: undefined }
{ topic: 'LEDToggle',
  payload: 2,
```

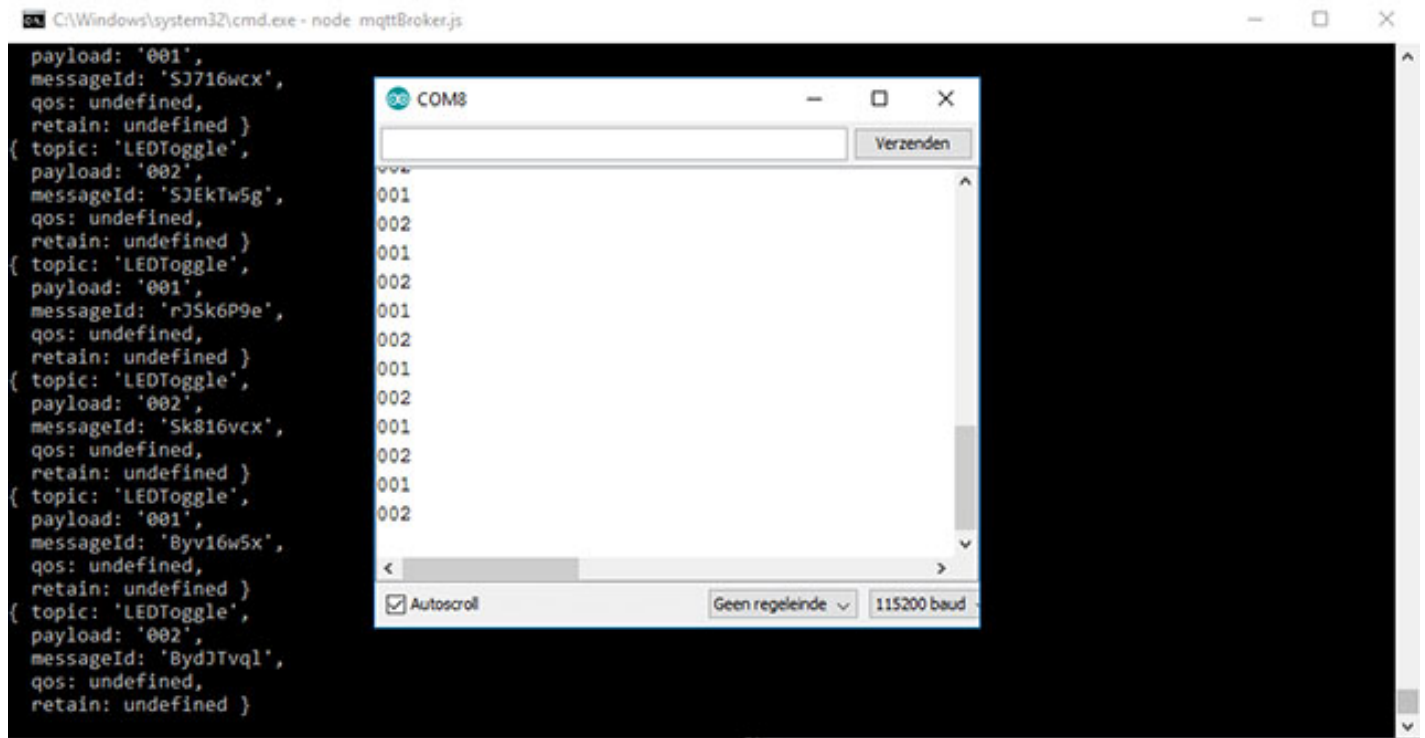
Subscribe to a topic

This step is so easy it's almost scary!

```
void setup() {
  ...
  mqttClient.subscribe("LEDToggle");
}
...
void loop() {
  mqttClient.loop();
}
```

`mqttClient.loop();` is their own loop function that keeps everything running smoothly.

So now the server knows that the client wants to have that information that the server is constantly publishing. If everything went like it should then we should have the connection!



Now we are basically done. I do want to connect the light and the button for you so that you can see it working.

Activating the light and the button

I won't be explaining this step anymore because this is really basic arduino code. Here is a video of the working end product.