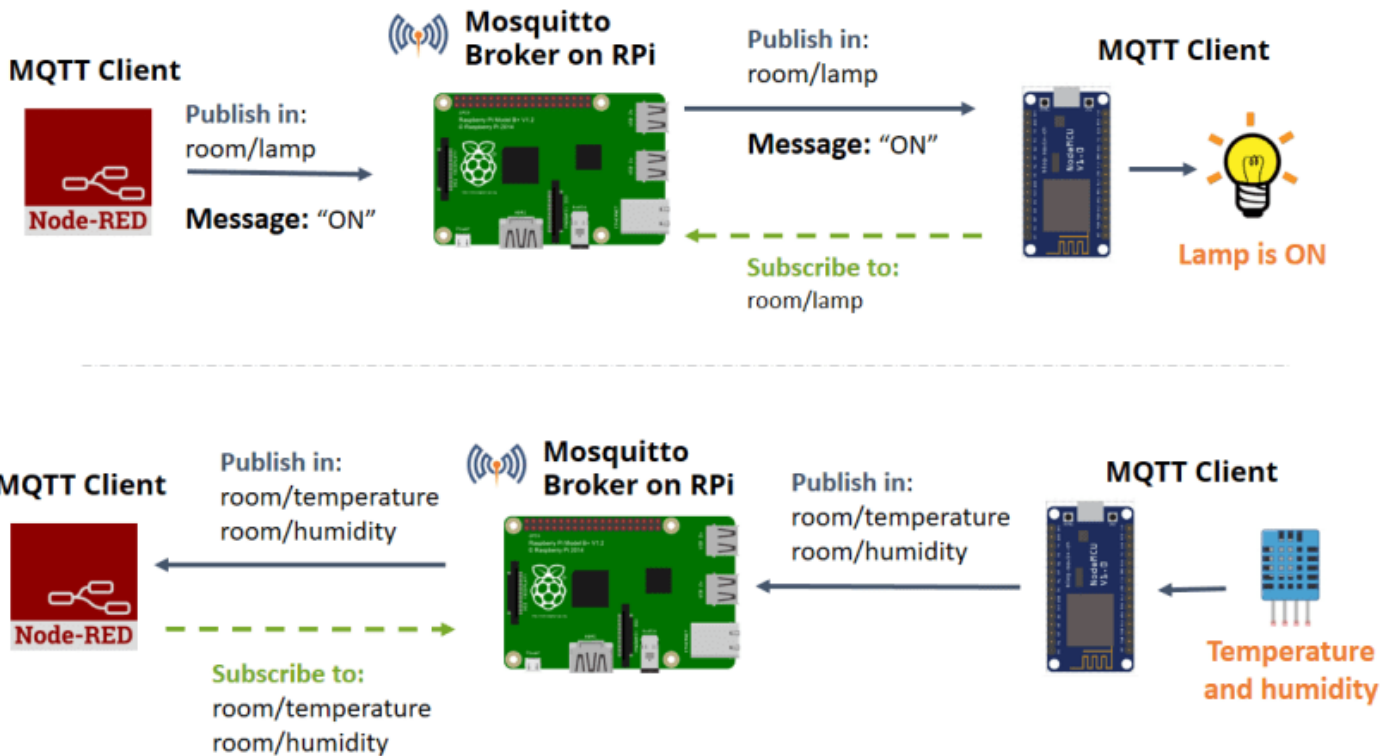


ESP8266 and Node-RED with MQTT (Publish and Subscribe)

In this post we're going to show you how to control ESP8266 outputs and display sensor data from the ESP8266 on Node-RED. The Node-RED software is running on a Raspberry Pi, and the communication between the ESP8266 and the Node-RED software is achieved with the MQTT communication protocol.

The following figure shows an overview of what we're going to do in this tutorial.



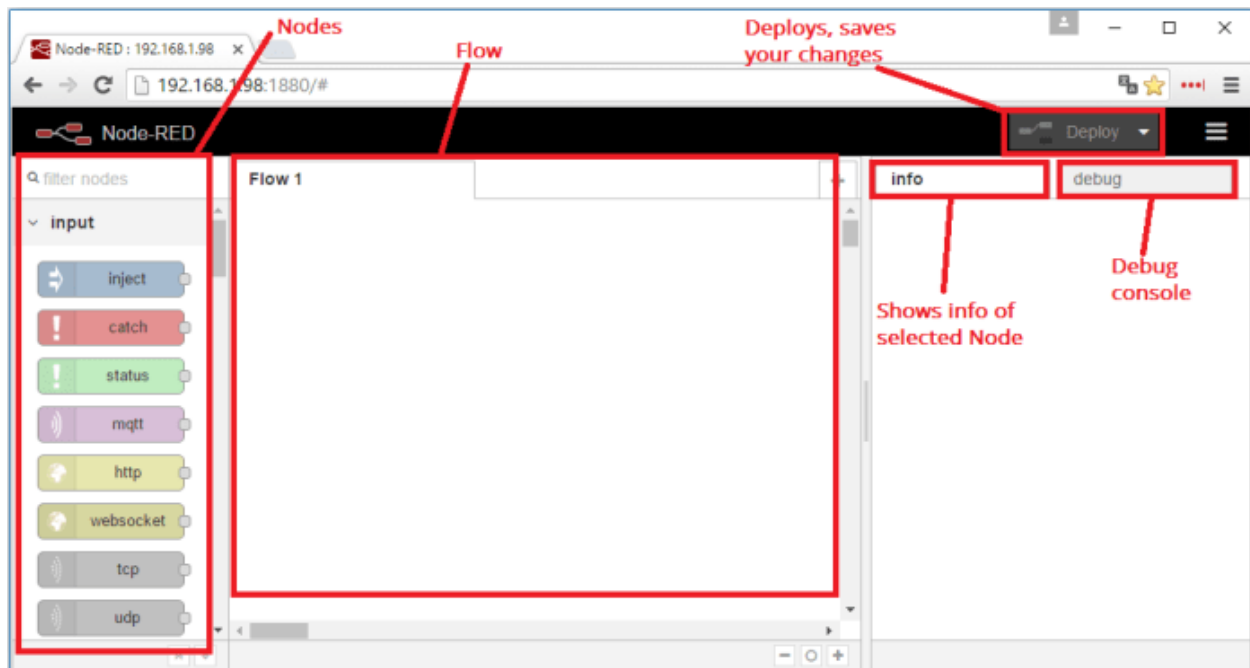
Node-RED and Node-RED Dashboard

You need to have Node-RED and Node-RED Dashboard installed in your [Raspberry Pi](#). The following blog posts are useful for getting started with Node-RED and Node-RED dashboard:

- [Getting started with Node-RED on Raspberry Pi](#)
- [Getting Started with Node-RED Dashboard](#)

Getting Started with Node-RED on Raspberry Pi

This post is an introductory guide to Node-RED. I'll cover what's Node-RED, how to install it, how to use the visual interface to create a simple flow.

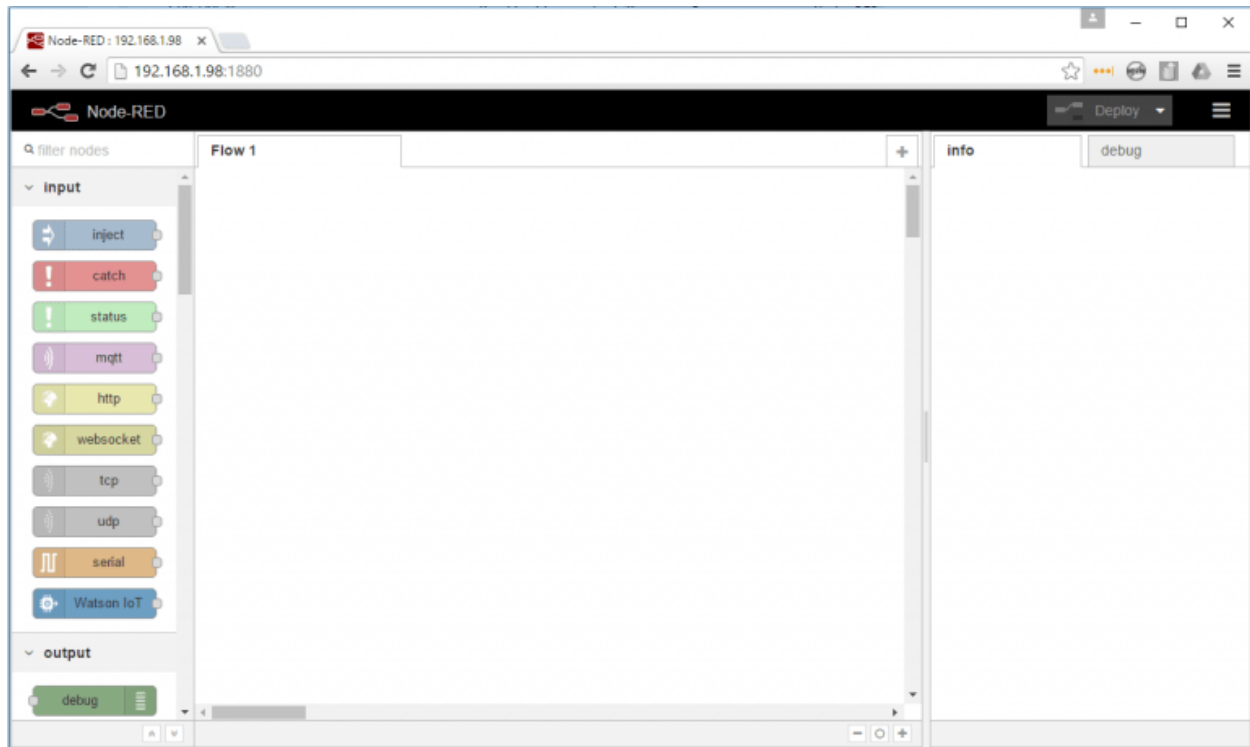


What's Node-RED?

[Node-RED](#) is a powerful open source tool for building Internet of Things (IoT) applications with the goal of simplifying the programming component.

It uses a visual programming that allows you to connect code blocks, known as **nodes**, together to perform a task.

The nodes when wired together are called **flows**.



Why do I think Node-RED is a great solution?

Node-RED is open source and developed by IBM.

The Raspberry Pi runs Node-RED perfectly.

With Node-RED you can spend more time making cool stuff, rather than spending countless hours writing code.

Don't get me wrong. I love programming and there is code that needs to be written throughout this course, but Node-RED allows you to prototype a complex home automation system quickly.

What can you do with Node-RED?

Node-RED makes it easy to:

- Access your RPi GPIOs
- Establish an MQTT connection with other boards (Arduino, ESP8266, etc)
- Create a responsive graphical user interface for your projects

- Communicate with third-party services (IFTTT.com, Adafruit.io, Thing Speak, etc)
- Retrieve data from the web (weather forecast, stock prices, emails. etc)
- Create time triggered events
- Store and retrieve data from a database

Installing Node-RED

Getting Node-RED installed in your Raspberry Pi is quick and easy. It just takes a few commands.

Having an SSH connection established with your Raspberry Pi, enter the following commands to install Node-RED:

Having an SSH connection established with your Raspberry Pi, enter the following commands to install Node-RED:

```
pi@raspberrypi:~ $ bash <(curl -sL https://raw.githubusercontent.com/node-red/raspbian-deb-package/master/resources/update-nodejs-and-nodered)
```

The installation should be completed after a couple of minutes.

Autostart Node-RED on boot

To automatically run Node-RED when the Pi boots up, you need to enter the following command:

```
pi@raspberrypi:~ $ sudo systemctl enable nodered.service
```

Now, restart your Pi so the autostart takes effect:

```
pi@raspberrypi:~ $ sudo reboot
```

Testing the Installation

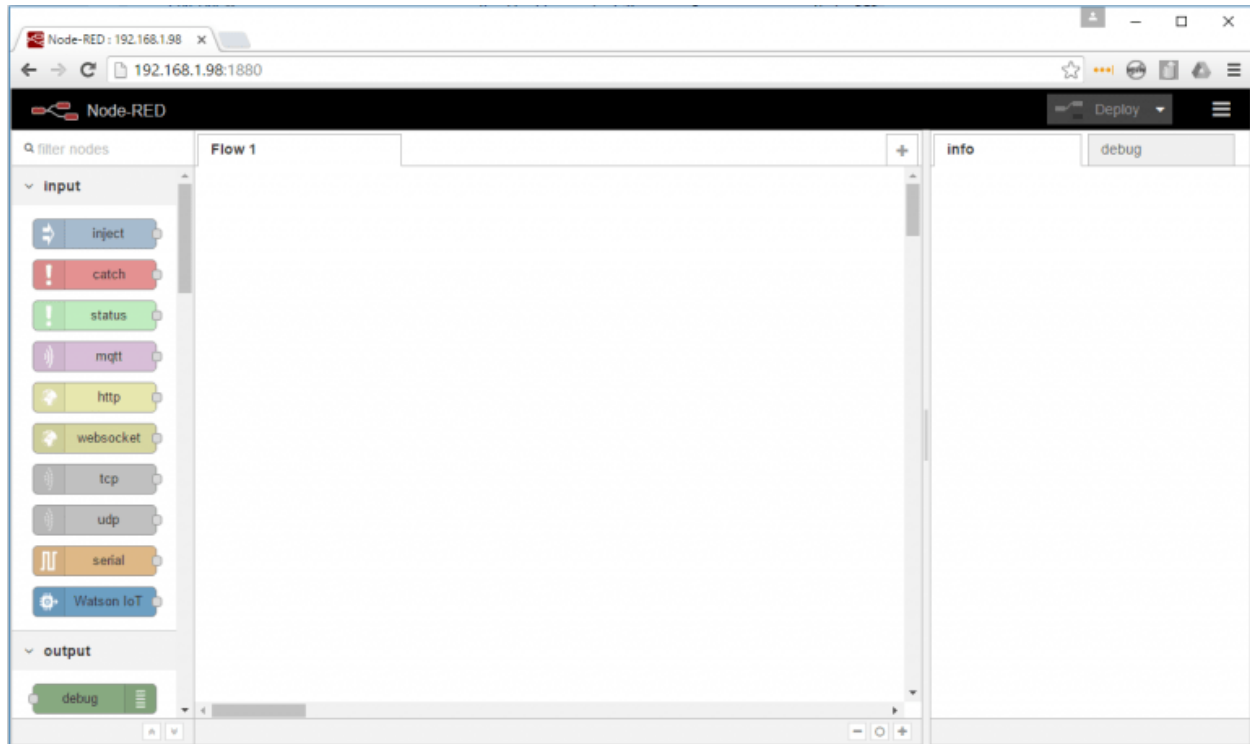
When your Pi is back on, you can test the installation by entering the IP address of your Pi in a web browser followed by the **1880** port number:

http://YOUR_RPi_IP_ADDRESS:1880

In my case is:

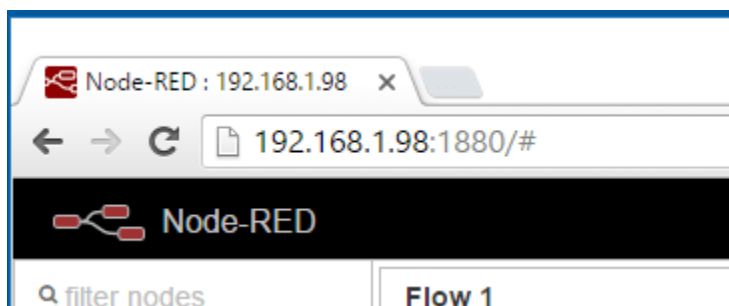
http://192.168.1.98:1880

A page like this loads:



Node-RED overview

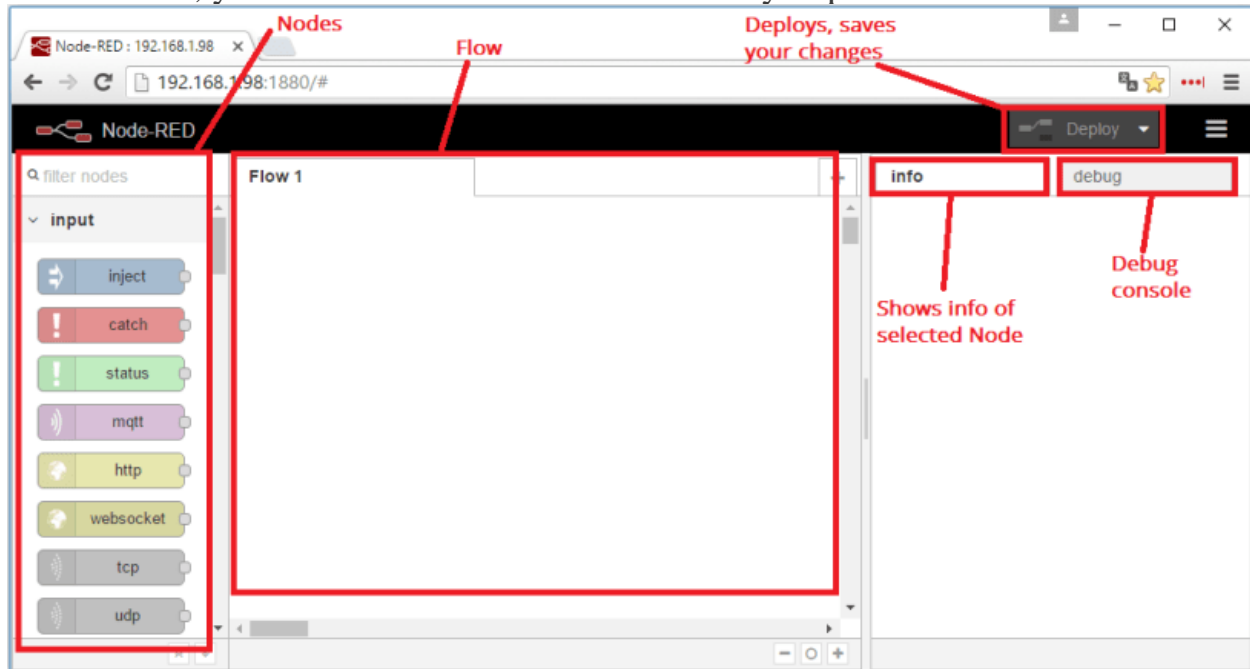
Let's take a look at the Node-RED visual interface.



Main sections

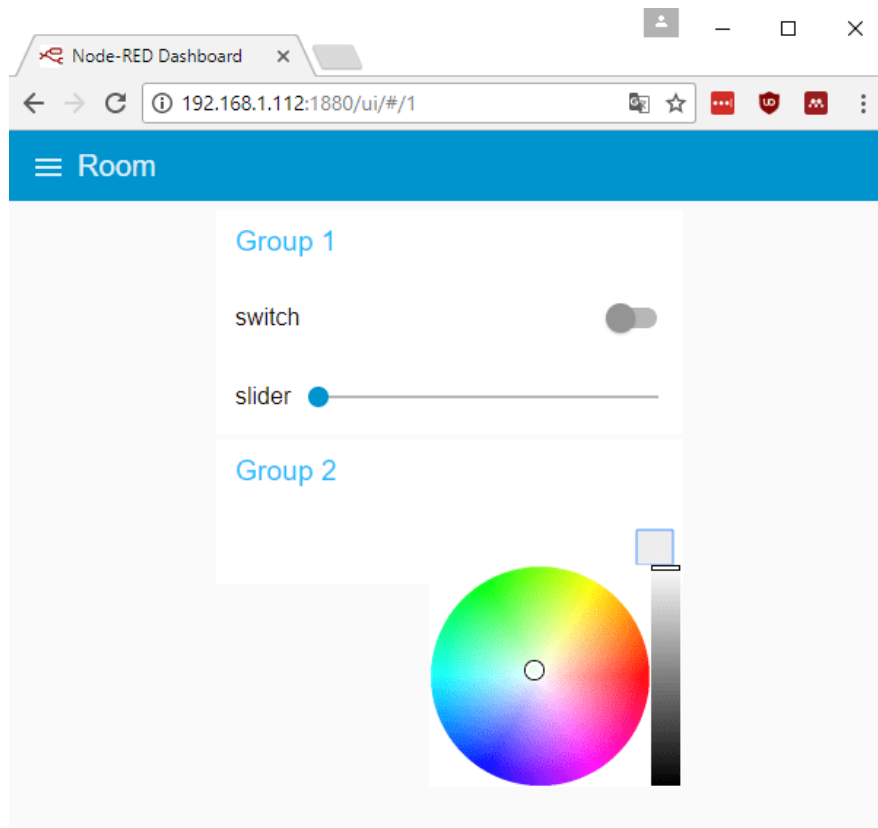
On the left-side, you can see a list with a bunch of blocks. These blocks are called **nodes** and they are separated by their functionality. If you select a node, you can see how it works in the **info** tab.

In the center, you have the **Flow** and this is where you place the nodes.



Getting Started with Node-RED Dashboard

This post is an introduction to Node-RED dashboard with Raspberry Pi. We'll cover how to install Node-RED Dashboard and exemplify how to build a graphical user interface.



What is Node-RED Dashboard?

Node-RED Dashboard is a module that provides a set of nodes in Node-RED to quickly create a live data dashboard.

To learn more about Node-RED Dashboard you can check the following links:

- Node-RED site: <http://flows.nodered.org/node/node-red-dashboard>
- GitHub: <https://github.com/node-red/node-red-dashboard>

Installing Node-RED Dashboard

To install the Node-RED Dashboard run the following commands:

```
pi@raspberrypi:~ $ node-red-stop
pi@raspberrypi:~ $ cd ~/.node-red
pi@raspberrypi:~/.node-red $ npm install node-red-dashboard
```

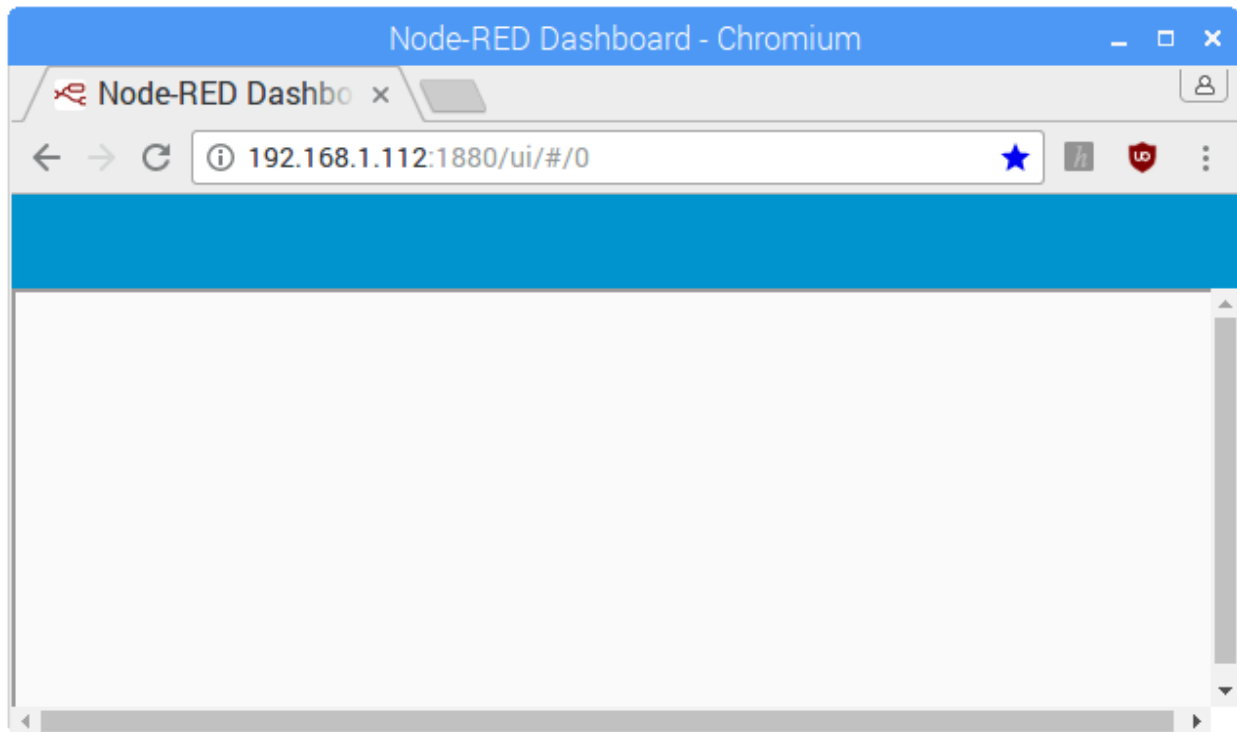
Then, reboot your Pi to ensure that all changes take effect on Node-RED software:

```
pi@raspberrypi:~ $ sudo reboot
```

To open the Node-RED UI, type your [Raspberry Pi IP address](#) in a web browser followed by **:1880/ui** as shown below:

http://Your_RPi_IP_address:1880/ui

At the moment your dashboard is empty – as you can see in the following figure – because we haven't added anything to the dashboard yet:



Creating a UI (User Interface)

In this section we're going to show you how to create your UI (User Interface) in Node-RED.

The Dashboard Layout

Open another tab in your browser to access Node-RED with:

http://Your_RPi_IP_address:1880

Scroll down on the nodes section. You'll see you have a set of nodes called **dashboard** as shown in the following figure:



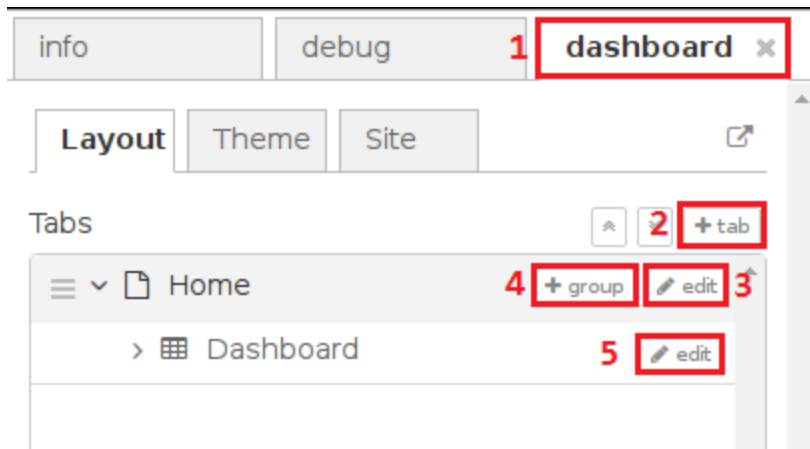
Nodes from the dashboard section provide widgets that show up in your application user interface (UI).

The user interface is organized in **tabs** and **groups**. Tabs are different pages on your user interface, like several tabs in a browser. Inside each tab you have groups that divide the tabs in different sections so that you can organize your widgets.

Every widget should have an associated group that determines where the widget should appear on the user interface.

To create a tab and a group follow the following instructions (see figure below):

- On top right corner of the Node-RED window you have a tab called **dashboard**.
- Select that tab **(1)**. To add a tab to the user interface click on the **+tab** button **(2)**.
- Once created, you can edit the tab by clicking on the **edit** button **(3)**.



You can edit the tab's name and change its icon:


- **Name:** you can call it whatever you want
- **Icon:** you should use a name accordingly to the icon's names in this link: <https://klarsys.github.io/angular-material-icons>

Edit dashboard tab node


Delete

Cancel

Update

 Name

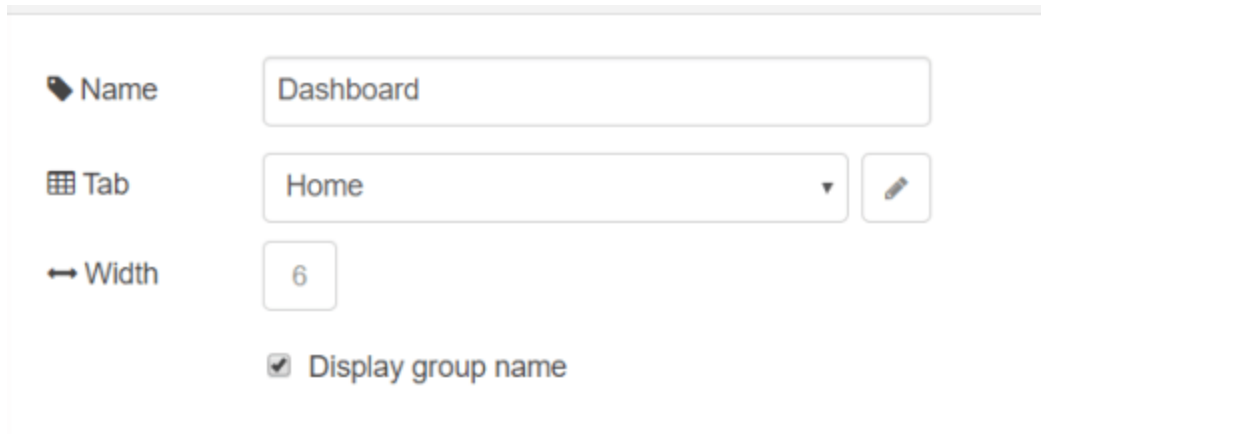
Home

 Icon

dashboard

After creating a tab, you can create several groups under that tab. You need to create at least one group to add your widgets. To add a group to the created tab, you need to click on the **+group** button **(4)**.

Then, you can edit the created group by clicking on the **edit** button **(5)**.

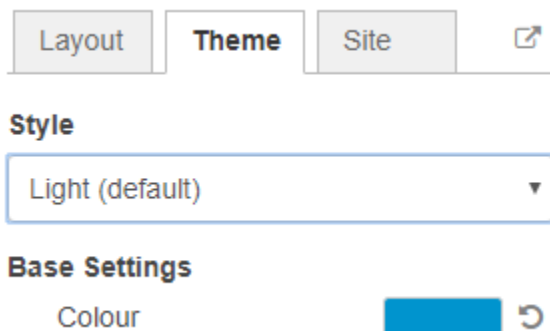


A screenshot of the Node-RED Dashboard configuration interface. It features three main input fields: 'Name' with the value 'Dashboard', 'Tab' with a dropdown menu set to 'Home' and an edit icon, and 'Width' with a numeric input set to '6'. Below these fields is a checkbox labeled 'Display group name' which is checked.

You can edit its name, select its corresponding tab and change its width.

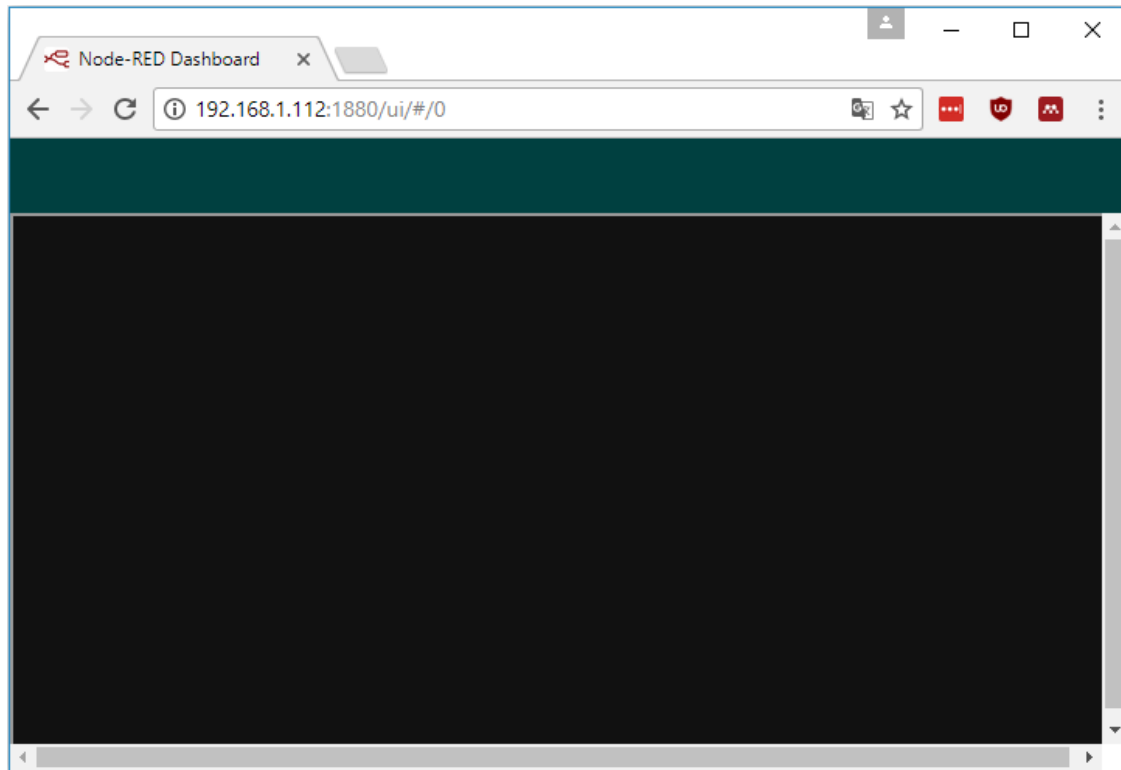
Dashboard Theme

The Node-RED Dashboard has a white background and a light blue bar by default. You can edit its colors in the **Theme** tab on the up right corner as show in the following figure.



A screenshot of the Node-RED Dashboard Theme configuration interface. At the top, there are three tabs: 'Layout', 'Theme' (which is active and highlighted), and 'Site', followed by an external link icon. Below the tabs, the 'Style' section contains a dropdown menu currently set to 'Light (default)'. The 'Base Settings' section includes a 'Colour' label, a blue color swatch, and a refresh icon.

Change the style, deploy the changes and see the Dashboard UI changing its colors. For example, like in the following figure:



Dashboard Site

At the right upper corner of the Node-RED window, you have another tab called **Site** that allows you to do further customization as show in the figure below.

Layout

Theme

Site

Title

Node-RED Dashboard

Options

Show the title bar

No swipe between tabs

Date Format

DD/MM/YYYY

Sizes

Horizontal

Vertical

1x1 Widget Size	48	48
Widget Spacing	6	6
Group Padding	0	0
Group Spacing	6	6

Feel free to change the settings, then deploy the changes and see how the UI looks. At the moment you won't see much difference because you haven't added anything to the dashboard yet. Those changes will be noticeable when you start adding widgets to the UI.

MQTT Protocol

MQTT stands for **MQ Telemetry Transport** and it is a nice lightweight publish and subscribe system where you can publish and receive messages as a client. It is a simple messaging protocol, designed for constrained devices and with low-bandwidth. So, it's the perfect solution for Internet of Things applications.

If you want to learn more about MQTT, watch the video below.

In this article, we're going to introduce you to the MQTT protocol. MQTT stands for **Message Queuing Telemetry Transport**.

Send a command to **control an output**



Read and publish data



It is a lightweight publish and subscribe system where you can publish and receive messages as a client.



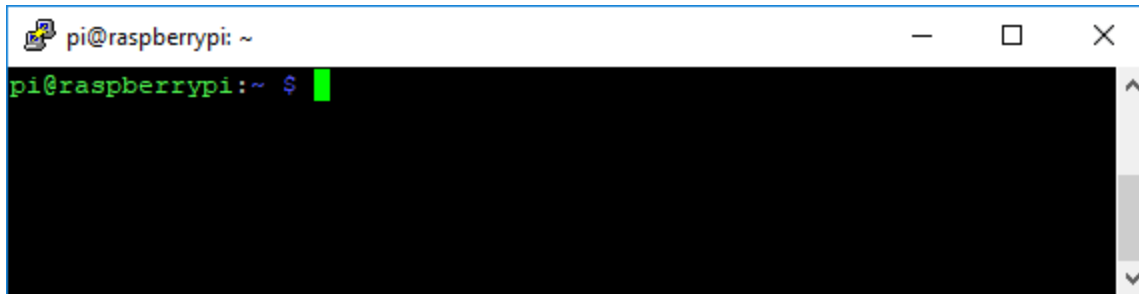
MQTT is a simple messaging protocol, designed for constrained devices with low-bandwidth. So, it's the perfect solution for Internet of Things applications. MQTT allows you to send commands to control outputs, read and publish data from sensor nodes and much more.

Installing Mosquitto Broker

In MQTT, the broker is primarily responsible for **receiving** all messages, **filtering** the messages, **decide** who is interested in it and then **publishing** the message to all subscribed clients.

There are several brokers you can use. In this tutorial we're going to use the **Mosquitto Broker** which needs to be installed on Raspberry Pi.

Open a new Raspberry Pi terminal window:

A terminal window titled 'pi@raspberrypi: ~' with standard window controls. The prompt 'pi@raspberrypi:~ \$' is shown with a green cursor.

To install the Mosquitto Broker enter these next commands:

```
pi@raspberrypi:~ $ sudo apt update  
pi@raspberrypi:~ $ sudo apt install -y mosquitto mosquitto-clients
```

You'll have to type **Y** and press **Enter** to confirm the installation. To make Mosquitto auto start on boot up enter:

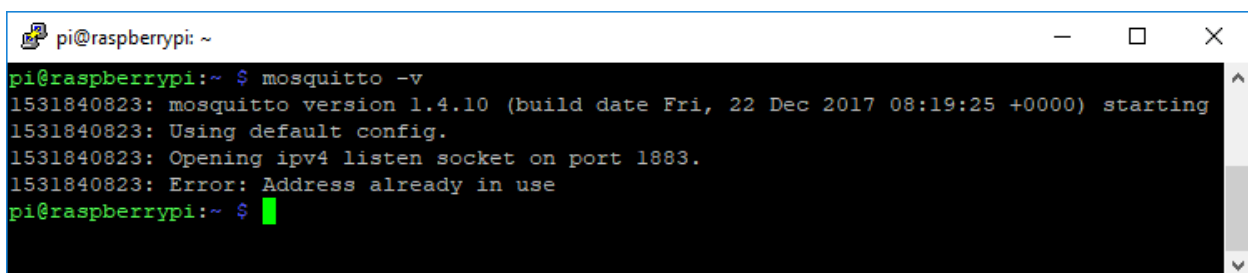
```
pi@raspberrypi:~ $ sudo systemctl enable mosquitto.service
```

Testing Installation

Send the command:

```
pi@raspberrypi:~ $ mosquitto -v
```

This returns the Mosquitto version that is currently running in your Raspberry Pi. It should be 1.4.X or above.

A terminal window titled 'pi@raspberrypi: ~' showing the output of the 'mosquitto -v' command. The output includes the version (1.4.10), build date, and a warning about the address already being in use.

```
pi@raspberrypi:~ $ mosquitto -v  
1531840823: mosquitto version 1.4.10 (build date Fri, 22 Dec 2017 08:19:25 +0000) starting  
1531840823: Using default config.  
1531840823: Opening ipv4 listen socket on port 1883.  
1531840823: Error: Address already in use  
pi@raspberrypi:~ $
```

Note: sometimes the command *mosquitto -v* prompts a warning message saying “*Error: Address already in use*“. That warning message means that your Mosquitto Broker is already running, so don't worry about that.

After [installing MQTT Broker](#), I recommend installing an MQTT Client to test the Broker installation and publish sample messages.

The next command shows how to install MQTT Mosquitto Client:

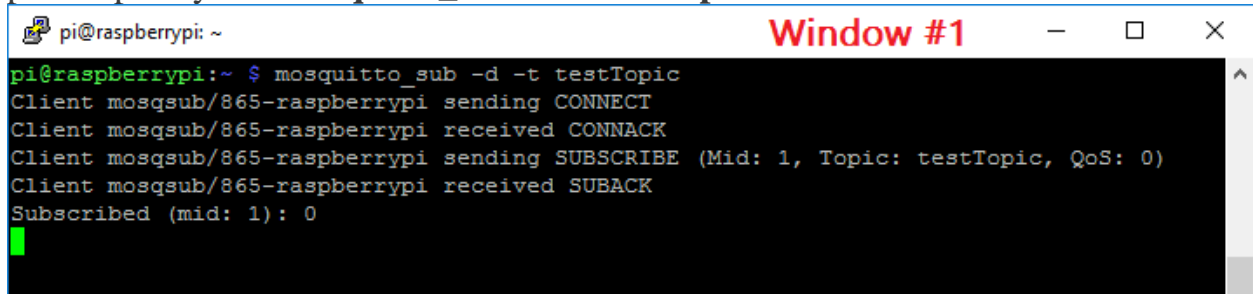
pi@raspberrypi:~ \$ **sudo apt-get install mosquitto-clients**
You'll have to type **Y** and press **Enter** to confirm the installation.
Run Mosquitto on background as a daemon:

pi@raspberrypi:~ \$ **mosquitto -d**

Subscribing to testTopic Topic

To subscribe to an MQTT topic with Mosquitto Client open a terminal Window #1 and enter the command:

pi@raspberrypi:~ \$ **mosquitto_sub -d -t testTopic**



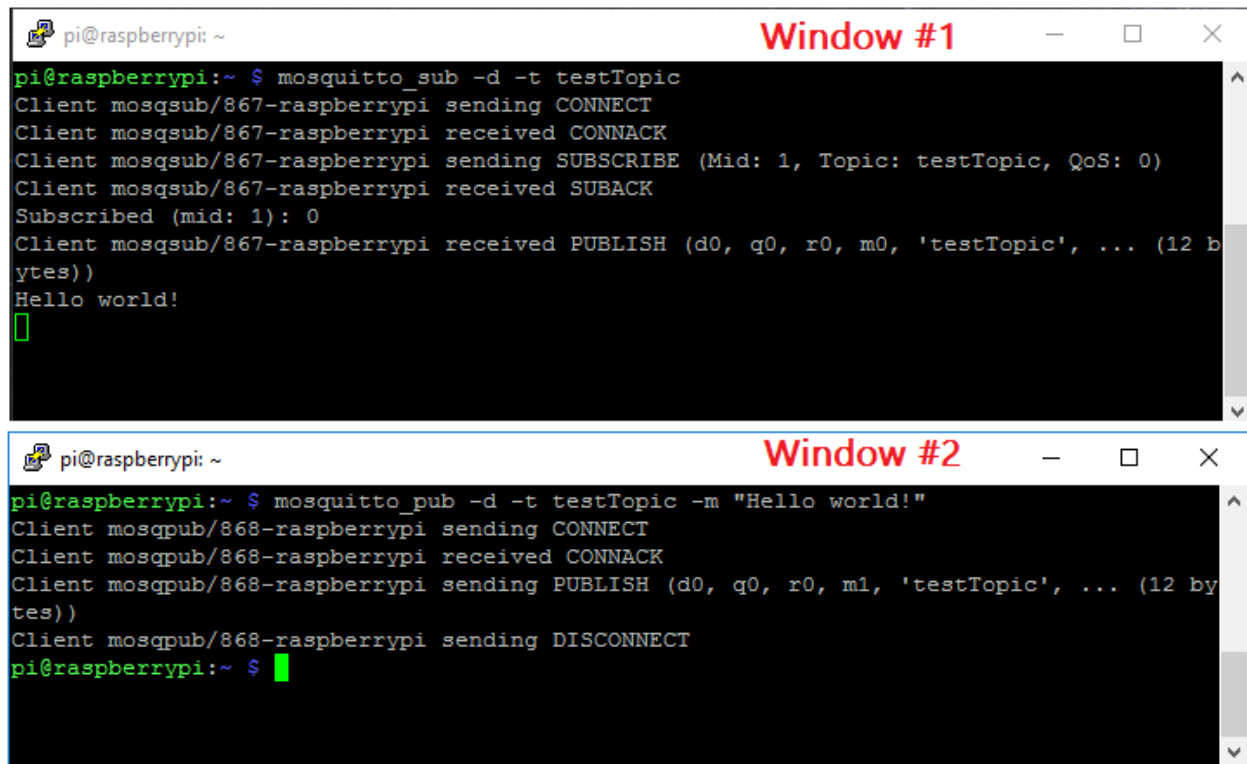
```
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic
Client mosqsub/865-raspberrypi sending CONNECT
Client mosqsub/865-raspberrypi received CONNACK
Client mosqsub/865-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)
Client mosqsub/865-raspberrypi received SUBACK
Subscribed (mid: 1): 0
```

You're now subscribed to a topic called **testTopic**.

Publishing "Hello World!" Message to testTopic Topic

To publish a sample message to **testTopic**, open a terminal Window #2 and run this command:

pi@raspberrypi:~ \$ **mosquitto_pub -d -t testTopic -m "Hello world!"**



```
pi@raspberrypi: ~  
Window #1  
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic  
Client mosqsub/867-raspberrypi sending CONNECT  
Client mosqsub/867-raspberrypi received CONNACK  
Client mosqsub/867-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)  
Client mosqsub/867-raspberrypi received SUBACK  
Subscribed (mid: 1): 0  
Client mosqsub/867-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))  
Hello world!  
█  
  
pi@raspberrypi: ~  
Window #2  
pi@raspberrypi:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"  
Client mosqpub/868-raspberrypi sending CONNECT  
Client mosqpub/868-raspberrypi received CONNACK  
Client mosqpub/868-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'testTopic', ... (12 bytes))  
Client mosqpub/868-raspberrypi sending DISCONNECT  
pi@raspberrypi:~ $ █
```

The message “**Hello World!**” is received in Window #1 as illustrated in the figure above.

Publishing a Message to Multiple Clients

Having Window #1 still subscribed to topic testTopic, open a new terminal Window #3 and run this command to subscribe to **testTopic** topic:

```
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic
```

On Window #2 publish the “**Hello World!**” message:

```
pi@raspberrypi:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"
```

```
pi@raspberrypi: ~  
Window #1  
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic  
Client mosqsub/919-raspberrypi sending CONNECT  
Client mosqsub/919-raspberrypi received CONNACK  
Client mosqsub/919-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)  
Client mosqsub/919-raspberrypi received SUBACK  
Subscribed (mid: 1): 0  
Client mosqsub/919-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))  
Hello world!  
Client mosqsub/919-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))  
Hello world!  
█  
pi@raspberrypi: ~  
Window #2  
pi@raspberrypi:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"  
Client mosqpub/920-raspberrypi sending CONNECT  
Client mosqpub/920-raspberrypi received CONNACK  
Client mosqpub/920-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'testTopic', ... (12 bytes))  
Client mosqpub/920-raspberrypi sending DISCONNECT  
pi@raspberrypi:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"  
Client mosqpub/922-raspberrypi sending CONNECT  
Client mosqpub/922-raspberrypi received CONNACK  
Client mosqpub/922-raspberrypi sending PUBLISH (d0, q0, r0, m1, 'testTopic', ... (12 bytes))  
Client mosqpub/922-raspberrypi sending DISCONNECT  
pi@raspberrypi:~ $ █  
pi@raspberrypi: ~  
Window #3  
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic  
Client mosqsub/921-raspberrypi sending CONNECT  
Client mosqsub/921-raspberrypi received CONNACK  
Client mosqsub/921-raspberrypi sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0)  
Client mosqsub/921-raspberrypi received SUBACK  
Subscribed (mid: 1): 0  
Client mosqsub/921-raspberrypi received PUBLISH (d0, q0, r0, m0, 'testTopic', ... (12 bytes))  
Hello world!  
█
```

Since two clients are subscribed to **testTopic** topic, they will both receive “**Hello world!**” message.

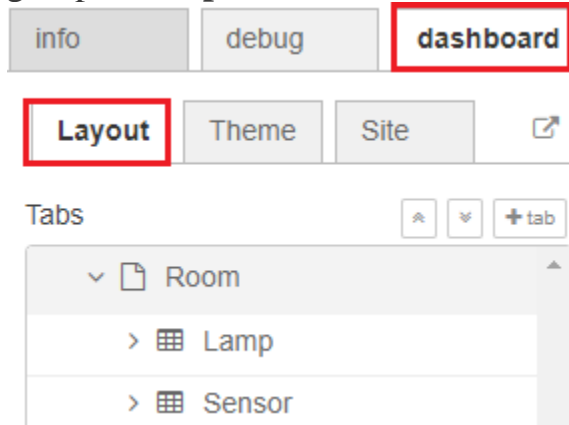
Establishing an MQTT communication with Node-RED

In this section we’re going to establish an MQTT communication using the Node-RED nodes.

Dashboard Layout

The first step is to create the dashboard layout. In this example, we'll have a button to control an ESP8266 output; a chart and a gauge to display temperature and humidity readings from the DHT11 sensor.

On the top right corner of the Node-RED window, select the **Layout** tab under the **dashboard** tab. Create a tab called **Room** and inside the Room tab, create two groups: **Lamp** and **Sensor** as shown in figure below.



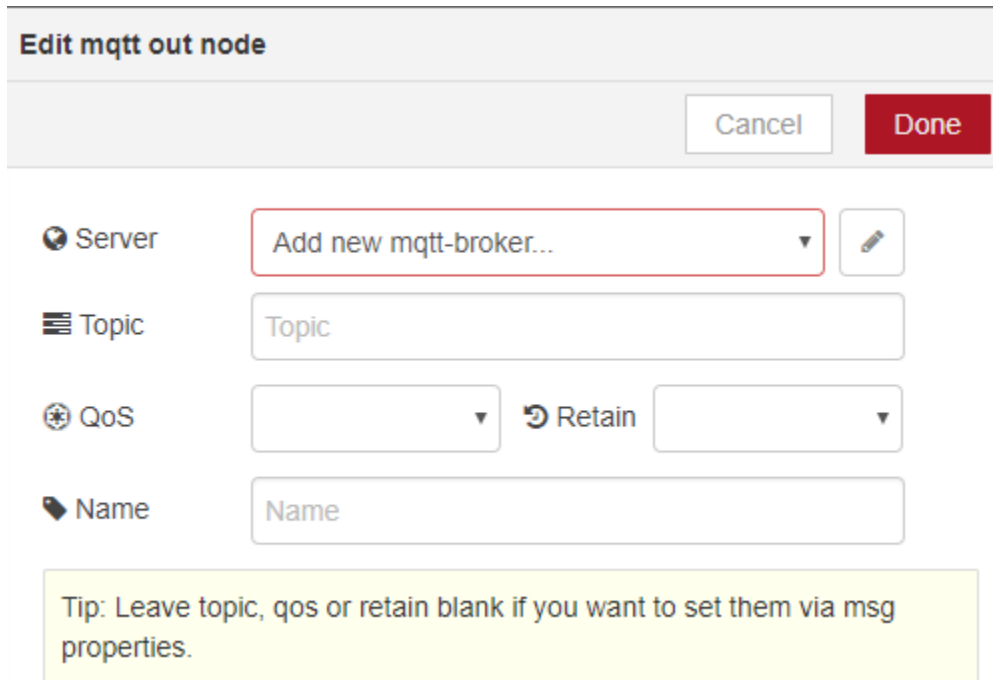
Creating the Flow

Drag the following nodes to the flow – see figure below:



- **switch** – this will control the ESP8266 output
- **mqtt output node** – this will publish a message to the ESP8266 accordingly to the switch state
- **2x mqtt input nodes** – these nodes will be subscribed to the temperature and humidity topics to receive sensor data from the ESP
- **chart** – will display the temperature sensor readings
- **gauge** – will display the humidity sensor readings

Node-RED and the MQTT broker need to be connected. To connect the MQTT broker to Node-RED, double-click the MQTT output node. A new window pops up – as shown in figure below.



Edit mqtt out node

Cancel Done

Server Add new mqtt-broker...

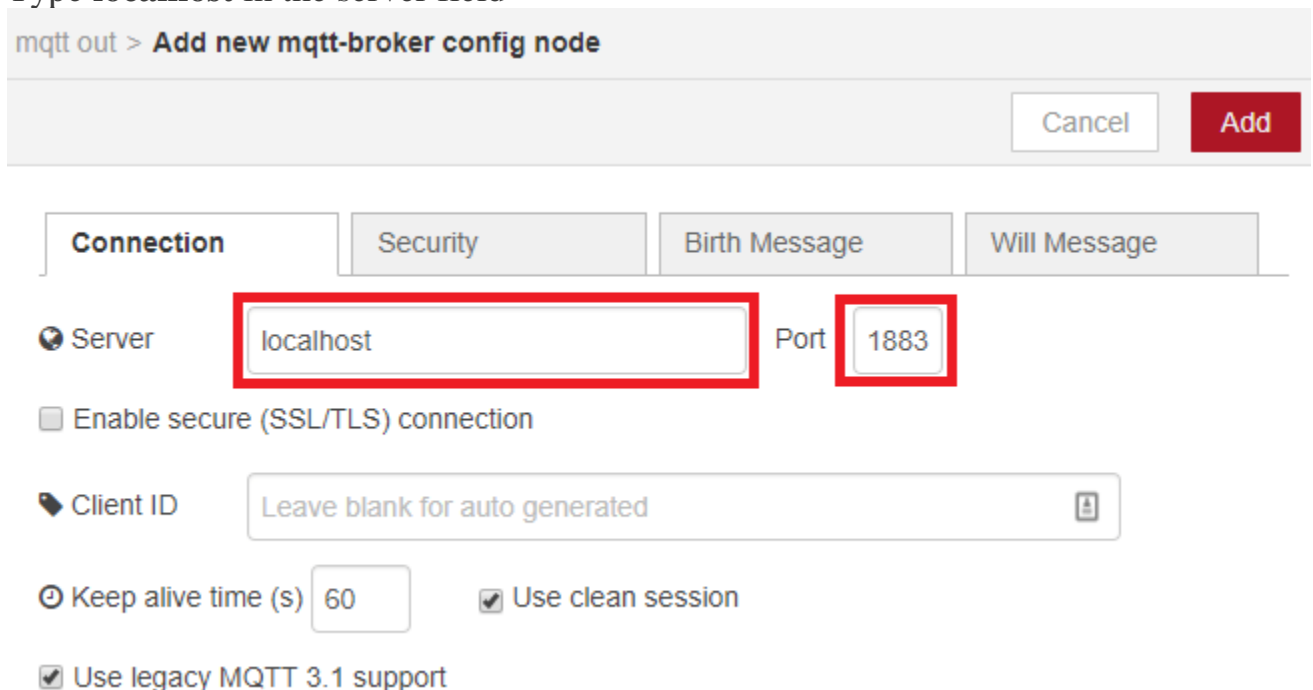
Topic Topic

QoS Retain

Name Name

Tip: Leave topic, qos or retain blank if you want to set them via msg properties.

1. Click the **Add new mqtt-broker** option.
2. Type **localhost** in the server field



mqtt out > Add new mqtt-broker config node

Cancel Add

Connection Security Birth Message Will Message

Server localhost Port 1883

☐ Enable secure (SSL/TLS) connection

Client ID Leave blank for auto generated

Keep alive time (s) 60 ☒ Use clean session

☒ Use legacy MQTT 3.1 support

3. All the other settings are configured properly by default.
4. Press **Add** and the MQTT output node automatically connects to your broker.
Edit all the other nodes properties as shown in the following figures:

- **switch** – the switch sends an **on** string message when it's on; and sends an **off** string message when it's off. This node will publish on the **room/lamptopic**. Your ESP will then be subscribed to this topic, to receive its messages.

Edit switch node

Cancel

Done

Group

Lamp [Room]

Size

auto

Label

Lamp

Icon

Default

→ If **msg** arrives on input, pass through to output: ☒

☒ When clicked, send:

On Payload

on

Off Payload

off

Topic

room/lamp

Name


Lamp

- **mqtt output node.** This node is connected to the mosquitto broker and it will publish in the **room/lamp** topic.


Edit mqtt out node


Cancel

Done


 Server

localhost:1883





 Topic

room/lamp

 QoS

2

 Retain

 Name


Lamp

- **mqtt input node.** This node is subscribed to the **room/temperature** topic to receive temperature sensor data from the ESP8266. The ESP8266 will be publishing the temperature readings on this topic.


Edit mqtt in node


Cancel

Done


 Server

localhost:1883




 Topic

room/temperature

 QoS

2

 Name

Temperature

- **chart.** The chart will display the readings received on the **room/temperature** topic.

Edit chart node

Cancel
Done

Group
Sensor [Room]

Size
auto

Label
Temperature

Type
Line chart

X-axis
last 1 hours OR 1000 points

X-axis Label
HH:mm:ss

Y-axis
min 0 max 40

Legend
None Interpolate step

Series Colours

Blank label
display this text before valid data arrives

Name
Temperature

- **mqtt input node.** This node is subscribed to the **room/humidity** topic to receive humidity sensor data from the ESP8266. The ESP8266 will be

publishing the humidity readings on this same topic.

Edit mqtt in node

Cancel

Done

Server

localhost:1883

Topic

room/humidity

QoS

2

Name

Humidity

- **gauge.** The gauge will display the readings received on the **room/humidity** topic.

Report this ad

 **MEDIAVINE**

Edit gauge node

Cancel

Done

Group

Sensor [Room]

Size

auto

Type

Gauge

Title

Humidity

Value format

{{value}}

Label

units

Range

min 0

max 100

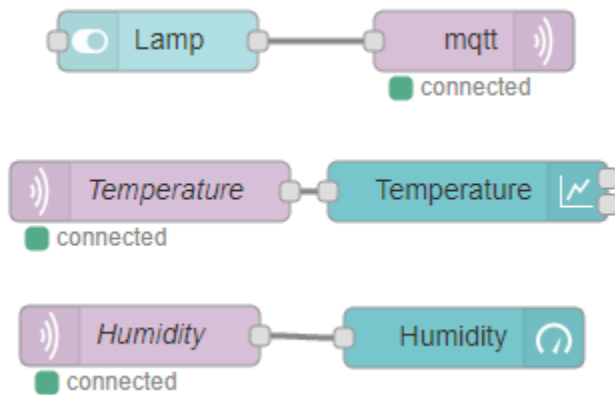
Name

Humidity

Wire your nodes as shown in the figure below.

Report this ad

MEDIAVINE



Your Node-RED application is ready. Click the **Deploy** button on the top right corner.



The Node-RED application is ready. To see how your dashboard looks go to <http://your-pi-ip-address/ui>.

Now, follow the next sections to prepare your ESP8266.

Preparing your Arduino IDE

We'll program the ESP8266 using the Arduino IDE. In order to upload code to your ESP8266 using the Arduino IDE, you need to install the ESP8266 add-on ([How to Install the ESP8266 Board in Arduino IDE](#)). You'll also need to install two additional libraries to have everything ready for your ESP8266.

Installing the PubSubClient Library

The [PubSubClient](#) library provides a client for doing simple publish/subscribe messaging with a server that supports MQTT (basically allows your ESP8266 to talk with Node-RED).

- 1) [Click here to download the PubSubClient library](#). You should have a `.zip` folder in your Downloads folder
- 2) Unzip the `.zip` folder and you should get **pubsubclient-master** folder
- 3) Rename your folder from **pubsubclient-master** to **pubsubclient**
- 4) Move the **pubsubclient** folder to your Arduino IDE installation **libraries** folder

5) Then, re-open your Arduino IDE

The library comes with a number of example sketches. See File >Examples > PubSubClient within the Arduino IDE software.

Installing the DHT Sensor Library

The [DHT sensor library](#) provides an easy way of using any DHT sensor to read temperature and humidity with your ESP8266 or Arduino boards.

1) [Click here to download the DHT sensor library](#). You should have a .zip folder in your Downloads

2) Unzip the .zip folder and you should get **DHT-sensor-library-master** folder

3) Rename your folder from **DHT-sensor-library-master** to **DHT**

4) Move the **DHT** folder to your Arduino IDE installation **libraries** folder

5) Then re-open your Arduino IDE

For more information about the DHT11 sensor and the ESP8266, read [ESP8266 DHT11/DHT22 Temperature and Humidity Web Server with Arduino IDE](#).

Selecting the right board on Arduino IDE

You also need to select the right board on Arduino IDE:

1) Go to Tools and select “NodeMCU 1.0 (ESP-12E Module)”.

2) Select your ESP port number under the Tools > Port > COM4 (in my case)

Uploading code

Now, you can upload the following code to your ESP8266. This code publishes messages of the temperature and humidity from the DHT11 sensor on the **room/temperature** and **room/humidity** topics through MQTT protocol.

The ESP is subscribed to the **room/lamp** topic to receive the messages published on that topic by the Node-RED application, to turn the lamp on or off.

The code is well commented on where you need to make changes. **You need to edit the code with your own SSID, password and RPi IP address.**

This code is also compatible with other DHT sensors – you just need to uncomment and comment the right lines of code to choose your sensor.

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

All the resources for this project:
<https://randomnerdtutorials.com/>

*****/

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

// Uncomment one of the lines below for whatever DHT sensor type you're using!

```
#define DHTTYPE DHT11 // DHT 11
// #define DHTTYPE DHT21 // DHT 21 (AM2301)
// #define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321
```

// Change the credentials below, so your ESP8266 connects to your router

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

// Change the variable to your Raspberry Pi IP address, so it connects to your MQTT broker

```
const char* mqtt_server = "REPLACE_WITH_YOUR_RPI_IP_ADDRESS";
```

// Initializes the espClient. You should change the espClient name if you have multiple ESPs running in your home automation system

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

// DHT Sensor - GPIO 5 = D1 on ESP-12E NodeMCU board

```
const int DHTPin = 5;
```

// Lamp - LED - GPIO 4 = D2 on ESP-12E NodeMCU board

```
const int lamp = 4;
```

// Initialize DHT sensor.

```
DHT dht(DHTPin, DHTTYPE);
```

// Timers auxiliar variables

```
long now = millis();
long lastMeasure = 0;
```

// Don't change the function below. This functions connects your ESP8266 to your router

```
void setup_wifi() {  
    delay(10);  
    // We start by connecting to a WiFi network  
    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.print("WiFi connected - ESP IP address: ");  
    Serial.println(WiFi.localIP());  
}
```

// This functions is executed when some device publishes a message to a topic that your ESP8266 is subscribed to

// Change the function below to add logic to your program, so when a device publishes a message to a topic that

// your ESP8266 is subscribed you can actually do something

```
void callback(String topic, byte* message, unsigned int length) {  
    Serial.print("Message arrived on topic: ");  
    Serial.print(topic);  
    Serial.print(". Message: ");  
    String messageTemp;  
  
    for (int i = 0; i < length; i++) {  
        Serial.print((char)message[i]);  
        messageTemp += (char)message[i];  
    }  
    Serial.println();  
}
```

// Feel free to add more if statements to control more GPIOs with MQTT

// If a message is received on the topic room/lamp, you check if the message is either on or off. Turns the lamp GPIO according to the message

```
if(topic=="room/lamp"){
```

```

Serial.print("Changing Room lamp to ");
if(messageTemp == "on"){
    digitalWrite(lamp, HIGH);
    Serial.print("On");
}
else if(messageTemp == "off"){
    digitalWrite(lamp, LOW);
    Serial.print("Off");
}
}
Serial.println();
}

// This functions reconnects your ESP8266 to your MQTT broker
// Change the function below if you want to subscribe to more topics with your
ESP8266
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        /*
        YOU MIGHT NEED TO CHANGE THIS LINE, IF YOU'RE HAVING
        PROBLEMS WITH MQTT MULTIPLE CONNECTIONS
        To change the ESP device ID, you will have to give a new name to the ESP8266.
        Here's how it looks:
        if (client.connect("ESP8266Client")) {
        You can do it like this:
        if (client.connect("ESP1_Office")) {
        Then, for the other ESP:
        if (client.connect("ESP2_Garage")) {
        That should solve your MQTT multiple connections problem
        */
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("room/lamp");
        } else {
            Serial.print("failed, rc=");

```

```

    Serial.print(client.state());
    Serial.println(" try again in 5 seconds");
    // Wait 5 seconds before retrying
    delay(5000);
  }
}

// The setup function sets your ESP GPIOs to Outputs, starts the serial
communication at a baud rate of 115200
// Sets your mqtt broker and sets the callback function
// The callback function is what receives messages and actually controls the LEDs
void setup() {
  pinMode(lamp, OUTPUT);

  dht.begin();

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

// For this project, you don't need to change anything in the loop function. Basically
it ensures that you ESP is connected to your broker
void loop() {

  if (!client.connected()) {
    reconnect();
  }
  if (!client.loop())
    client.connect("ESP8266Client");

  now = millis();
  // Publishes new temperature and humidity every 30 seconds
  if (now - lastMeasure > 30000) {
    lastMeasure = now;
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    float h = dht.readHumidity();

```

```

// Read temperature as Celsius (the default)
float t = dht.readTemperature();
// Read temperature as Fahrenheit (isFahrenheit = true)
float f = dht.readTemperature(true);

// Check if any reads failed and exit early (to try again).
if (isnan(h) || isnan(t) || isnan(f)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
}

// Computes temperature values in Celsius
float hic = dht.computeHeatIndex(t, h, false);
static char temperatureTemp[7];
dtostrf(hic, 6, 2, temperatureTemp);

// Uncomment to compute temperature values in Fahrenheit
// float hif = dht.computeHeatIndex(f, h);
// static char temperatureTemp[7];
// dtostrf(hic, 6, 2, temperatureTemp);

static char humidityTemp[7];
dtostrf(h, 6, 2, humidityTemp);

// Publishes Temperature and Humidity values
client.publish("room/temperature", temperatureTemp);
client.publish("room/humidity", humidityTemp);

Serial.print("Humidity: ");
Serial.print(h);
Serial.print(" %\t Temperature: ");
Serial.print(t);
Serial.print(" *C ");
Serial.print(f);
Serial.print(" *F\t Heat index: ");
Serial.print(hic);
Serial.println(" *C ");
// Serial.print(hif);
// Serial.println(" *F");
}

```

[View raw code](#)

This is helpful to check if the ESP has established a successful connection to your router and to the Mosquitto broker. You can also see the messages the ESP is receiving and publishing.



Parts required

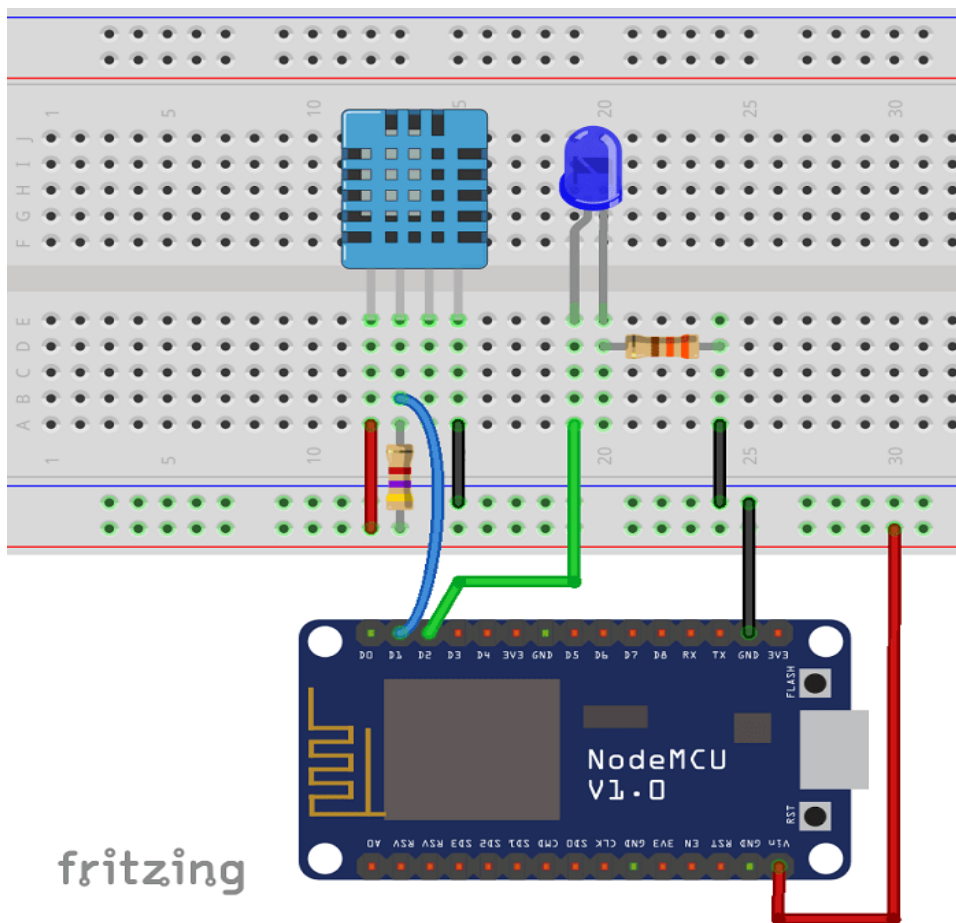
These are the parts required to build the circuit (click the links below to find the best price at [Maker Advisor](#)):

- [Raspberry Pi](#) – [read Best Raspberry Pi 3 Starter Kits](#)
- [ESP8266 \(ESP-12E nodemcu\)](#) – [read Best ESP8266 Wi-Fi Development Boards](#)
- [DHT11 temperature and humidity sensor](#)
- [Breadboard](#)
- [330 \$\Omega\$ resistor](#)
- [LED](#)
- [4700 \$\Omega\$ resistor](#)

You can use the preceding links or go directly to [MakerAdvisor.com/tools](#) to find all the parts for your projects at the best price!

Schematics

Here are the schematics for this project's circuit.

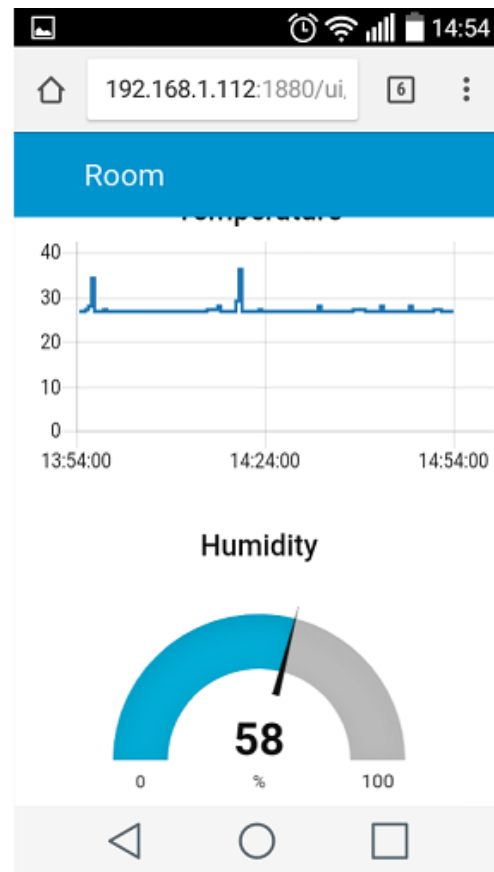
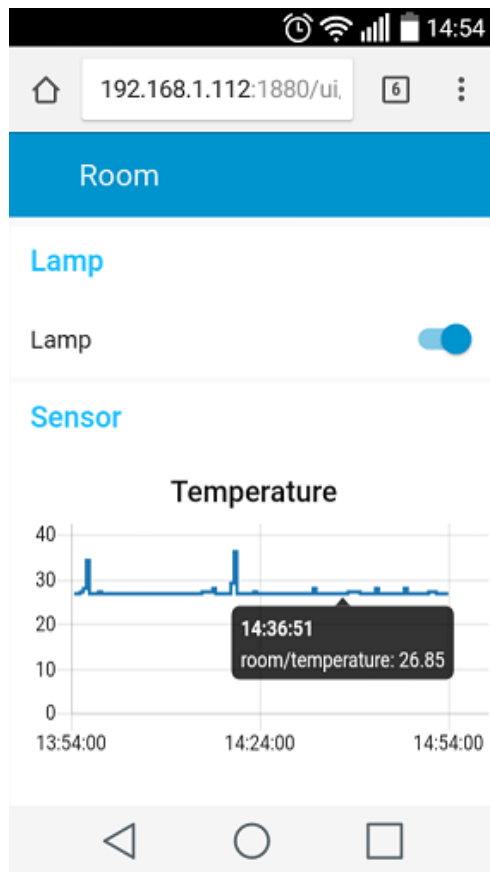


Demonstration

Congratulations! Your project is now completed.

Go to <http://your-pi-ip-address/ui> to control the ESP with the Node-RED application. You can access your application in any browser on the same network that your Pi (watch the video demonstration below).

The application should look something like the figure below.



Wrapping up

In this tutorial we've shown you the basic concepts that will allow you to turn on lights and monitor sensors on your ESP using Node-RED and the MQTT communication protocol. You can follow these basic steps to build more advanced projects.

We hope you've found this tutorial useful.