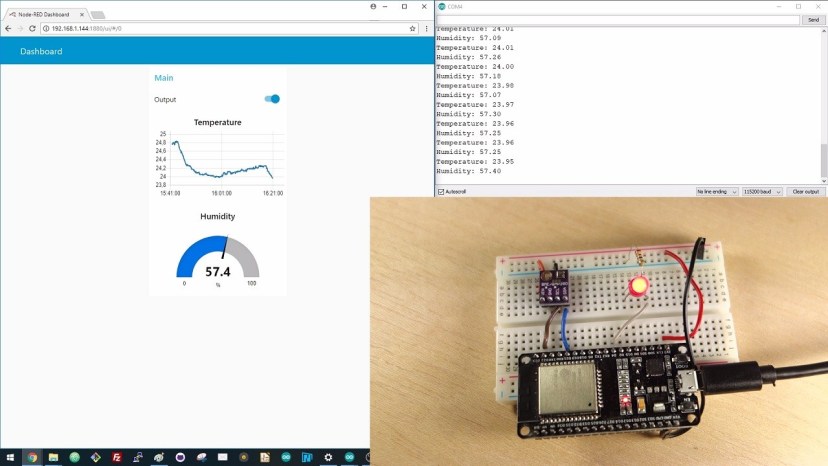
https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/

MQTT communication protocol with the ESP32 to publish messages and subscribe to topics.

As an example, we’ll publish BME280 sensor readings to the Node-RED Dashboard, and control an ESP32 output. The ESP32 we’ll be programmed using Arduino IDE.



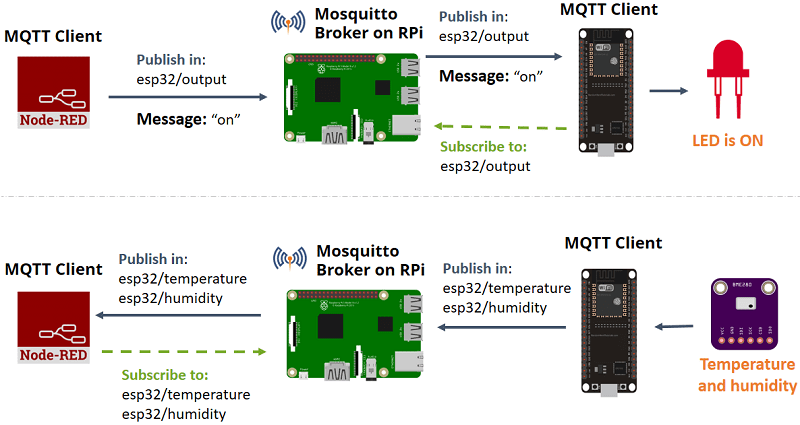
There’s a Node-RED application that controls ESP32 outputs and receives sensor readings from the ESP32 using MQTT communication protocol.

The Node-RED application is running on a Raspberry Pi.

We’ll use the [Mosquitto broker installed](https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/) on the same Raspberry Pi. The broker is responsible for

* receiving all messages,
* filtering the messages,
* decide who is interested in them
* and publishing the messages to all subscribed clients.

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



* The Node-RED application publishes messages (“**on**” or “**off**“) in the topic **esp32/output**. The ESP32 is subscribed to that topic. So, it receives the message with “on” or “off” to turn the LED on or off.
* The ESP32 publishes temperature on the **esp32/temperature** topic and the humidity on the **esp32/humidity** topic. The Node-RED application is subscribed to those topics. So, it receives temperature and humidity readings that can be displayed on a chart or gauge, for example.

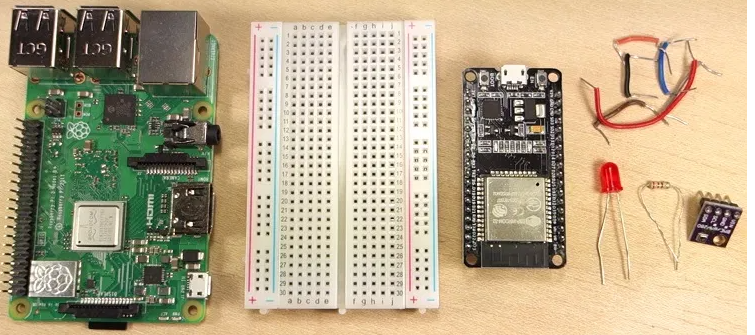
**Note:** there’s also a similar tutorial on how to use the [ESP8266 and Node-RED with MQTT](https://randomnerdtutorials.com/esp8266-and-node-red-with-mqtt/).

**Prerequisites**

* You should be familiar with the Raspberry Pi – [read Getting Started with Raspberry Pi](https://randomnerdtutorials.com/getting-started-with-raspberry-pi/).
* You should have the Raspbian operating system installed in your Raspberry Pi – [read Installing Raspbian Lite, Enabling and Connecting with SSH](https://randomnerdtutorials.com/installing-raspbian-lite-enabling-and-connecting-with-ssh/).
* You need [Node-RED installed on your Pi](https://randomnerdtutorials.com/getting-started-with-node-red-on-raspberry-pi/) and [Node-RED Dashboard](https://randomnerdtutorials.com/getting-started-with-node-red-dashboard/).
* Learn [what’s MQTT and how it works](https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/).

If you like home automation and you want to learn more about Node-RED, Raspberry Pi, ESP8266 and Arduino, we recommend trying our course: [Build a Home Automation System with Node-RED, ESP8266 and Arduino](https://randomnerdtutorials.com/build-a-home-automation-system-for-100/).

We also have a course dedicated to the ESP32: Enroll in [Learn ESP32 with Arduino IDE](https://randomnerdtutorials.com/learn-esp32-with-arduino-ide/) course.



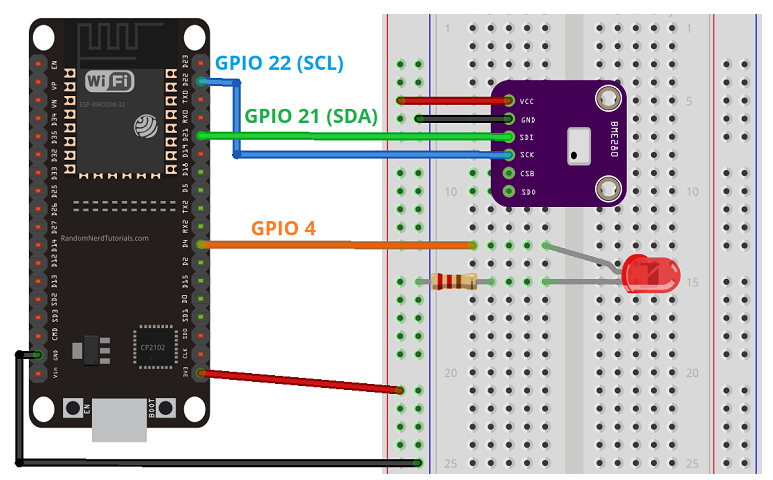
The BME280 sensor (temperature, humidity) can communicate using either SPI or I2C communication protocols (there are modules of this sensor that just communicate with I2C, these just come with four pins).

To use SPI communication protocol, use the following pins:

* SCK – this is the SPI Clock pin
* SDO – MISO
* SDI – MOSI
* CS – Chip Select

To use I2C communication protocol, the sensor uses the following pins:

* SCK – SCL pin
* SDI – SDA pin



LIBRARIES

* PubSubClient allow ESP32 to talk with Node-RED
* BME280, allows sensor to read temperature and humidity
* Adafruit sensor library, for BME280 to work

#### Installing the PubSubClient Library

The [PubSubClient](https://github.com/knolleary/pubsubclient" \t "_blank) library provides a client for doing simple publish/subscribe messaging with a server that supports MQTT (basically allows your ESP32 to talk with Node-RED).

1. [Click here to download the PubSubClient library](https://github.com/knolleary/pubsubclient/archive/master.zip). 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  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.

**Important:**PubSubClient is not fully compatible with the ESP32, but the example provided in this tutorial is working very reliably during our tests.

#### Installing the BME280 library

To take readings from the BME280 sensor module we’ll use the [Adafruit\_BME280 library](https://github.com/adafruit/Adafruit_BME280_Library). Follow the next steps to install the library in your Arduino IDE:

1. [Click here to download](https://github.com/adafruit/Adafruit_BME280_Library/archive/master.zip) the Adafruit-BME280 library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get **Adafruit-BME280-Library-master** folder
3. Rename your folder from  to **Adafruit\_BME280\_Library**
4. Move the **Adafruit\_BMPE280\_Library** folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Alternatively, you can go to **Sketch** > **Include Library** > **Manage Libraries** and type “**adafruit bme280**” to search for the library. Then, click install.

#### Installing the Adafruit\_Sensor library

To use the BME280 library, you also need to install the [Adafruit\_Sensor library](https://github.com/adafruit/Adafruit_Sensor" \t "_blank). Follow the next steps to install the library:

1. [Click here to download](https://github.com/adafruit/Adafruit_Sensor/archive/master.zip) the Adafruit\_Sensor library. You should have a .zip folder in your Downloads folder
2. Unzip the .zip folder and you should get **Adafruit\_Sensor-master** folder
3. Rename your folder from  to **Adafruit\_Sensor**
4. Move the **Adafruit\_Sensor** folder to your Arduino IDE installation libraries folder
5. Finally, re-open your Arduino IDE

Sketch in ESP32

#include <WiFi.h>

#include <PubSubClient.h>

#include <Wire.h>

#include <Adafruit\_BME280.h>

#include <Adafruit\_Sensor.h>

const char\* ssid = "REPLACE\_WITH\_YOUR\_SSID";

const char\* password = "REPLACE\_WITH\_YOUR\_PASSWORD";

const char\* mqtt\_server = "192.168.1.144 MQTT\_BROKER\_IP";

WiFiClient espClient;

PubSubClient client(espClient);

long lastMsg = 0;

char msg[50];

int value = 0;

//uncomment the following lines if you're using SPI

/\*#include <SPI.h>

#define BME\_SCK 18

#define BME\_MISO 19

#define BME\_MOSI 23

#define BME\_CS 5\*/

Adafruit\_BME280 bme; // I2C

//Adafruit\_BME280 bme(BME\_CS); // hardware SPI

//Adafruit\_BME280 bme(BME\_CS, BME\_MOSI, BME\_MISO, BME\_SCK); // software SPI

float temperature = 0;

float humidity = 0;

const int ledPin = 4;

void setup() {

Serial.begin(115200);

// (you can also pass in a Wire library object like &Wire2)

//status = bme.begin();

if (!bme.begin(0x76)) {

Serial.println("Could not find BME280 sensor, check wiring!");

while (1);

}

setup\_wifi();

* TCP Port: **1883**
* Websocket Port: **8083**

client.setServer(mqtt\_server, 1883);

client.setCallback(callback);

pinMode(ledPin, OUTPUT);

}

void setup\_wifi() {

delay(10);

Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL\_CONNECTED) {

delay(500);

Serial.print(".");

}

Serial.println("WiFi connected");

Serial.println("IP address: ");

Serial.println(WiFi.localIP());

}

void callback(char\* 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();

// If a message is received on the topic esp32/output, you check if the message is either "on" or "off".

// Changes the output state according to the message

if (String(topic) == "esp32/output") {

Serial.print("Changing output to ");

if(messageTemp == "on"){

Serial.println("on");

digitalWrite(ledPin, HIGH);

}

else if(messageTemp == "off"){

Serial.println("off");

digitalWrite(ledPin, LOW);

}

}

}

void reconnect() {

// Loop until we're reconnected

while (!client.connected()) {

Serial.print("Attempting MQTT connection...");

// Attempt to connect

if (client.connect("ESP8266Client")) {

Serial.println("connected");

// Subscribe

client.subscribe("esp32/output");

} else {

Serial.print("failed, rc=");

Serial.print(client.state());

Serial.println(" try again in 5 seconds");

// Wait 5 seconds before retrying

delay(5000);

}

}

}

void loop() {

if (!client.connected()) {

reconnect();

}

client.loop();

long now = millis();

if (now - lastMsg > 5000) {

lastMsg = now;

// Temperature in Celsius

temperature = bme.readTemperature();

// Uncomment the next line to set temperature in Fahrenheit

// (and comment the previous temperature line)

//temperature = 1.8 \* bme.readTemperature() + 32; // Temperature in Fahrenheit

// Convert the value to a char array

char tempString[8];

dtostrf(temperature, 1, 2, tempString);

Serial.print("Temperature: ");

Serial.println(tempString);

client.publish("esp32/temperature", tempString);

humidity = bme.readHumidity();

// Convert the value to a char array

char humString[8];

dtostrf(humidity, 1, 2, humString);

Serial.print("Humidity: ");

Serial.println(humString);

client.publish("esp32/humidity", humString);

}

}

In the reconnect() function, you can subscribe to MQTT topics. In this case, the ESP32 is only subscribed to the**esp32/output**:

In the callback() function, the ESP32 receives the MQTT messages of the subscribed topics. According to the MQTT topic and message, it turns the LED on or off:

In the *loop(),* new readings are being published every 5 seconds:

if (now - lastMsg > 5000) { ... }

By default the ESP32 is sending the temperature in Celsius, but you can uncomment the last line to send the temperature in Fahrenheit

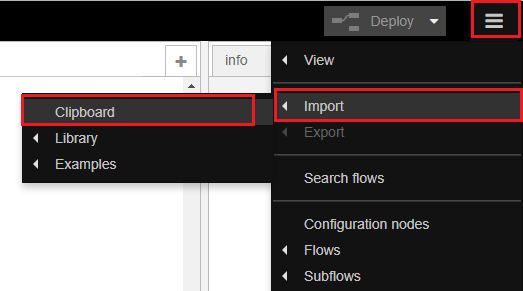
**Creating the Node-RED flow**

Before creating the flow, you need to have installed in your Raspberry Pi:

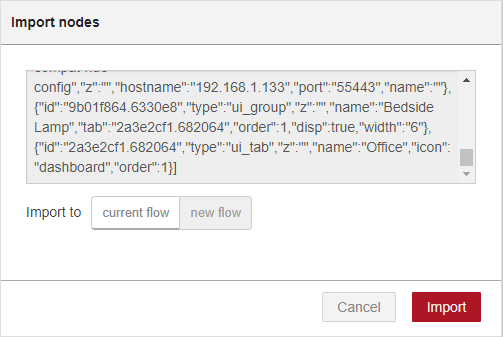
* [Node-RED](https://randomnerdtutorials.com/getting-started-with-node-red-on-raspberry-pi/)
* [Node-RED Dashboard](https://randomnerdtutorials.com/getting-started-with-node-red-dashboard/)
* [Mosquitto Broker](https://randomnerdtutorials.com/how-to-install-mosquitto-broker-on-raspberry-pi/)

After that, import the Node-RED flow provided. Go to the [GitHub repository](https://raw.githubusercontent.com/RuiSantosdotme/Random-Nerd-Tutorials/master/Projects/ESP32-MQTT/Node_RED_Flow_ESP32_MQTT_Publish_Subscribe.txt) or click the figure below to see the raw file, and copy the code provided.

Next, in the Node-RED window, at the top right corner, select the menu, and go to **Import** > **Clipboard**.



Then, paste the code provided and click **Import**.



<https://raw.githubusercontent.com/RuiSantosdotme/Random-Nerd-Tutorials/master/Projects/ESP32-MQTT/Node_RED_Flow_ESP32_MQTT_Publish_Subscribe.txt>

[

{"id":"9e58624.7faaba","type":"mqtt out","z":"c02b79b2.501998","name":"","topic":"esp32/output","qos":"","retain":"","broker":"10e78a89.5b4fd5","x":610,"y":342,"wires":[]},

{"id":"abf7079a.653be8","type":"mqtt in","z":"c02b79b2.501998","name":"","topic":"esp32/temperature","qos":"2","broker":"10e78a89.5b4fd5","x":484,"y":249,"wires":[["cc79021b.9a751","21eae8f8.2971b8"]]},

{"id":"83cf37cf.c76988","type":"ui\_switch","z":"c02b79b2.501998","name":"","label":"Output","group":"61285987.c20328","order":0,"width":0,"height":0,"passthru":true,"decouple":"false","topic":"","style":"","onvalue":"on","onvalueType":"str","onicon":"","oncolor":"","offvalue":"off","offvalueType":"str","officon":"","offcolor":"","x":469,"y":342,"wires":[["9e58624.7faaba"]]},

{"id":"cc79021b.9a751","type":"debug","z":"c02b79b2.501998","name":"","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"false","x":681,"y":216,"wires":[]},

{"id":"4aecba01.78ce64","type":"mqtt in","z":"c02b79b2.501998","name":"","topic":"esp32/humidity","qos":"2","broker":"10e78a89.5b4fd5","x":473,"y":133,"wires":[["22efa7b7.544a28","df37e6b7.64c1c8"]]},

{"id":"22efa7b7.544a28","type":"debug","z":"c02b79b2.501998","name":"","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"false","x":670,"y":100,"wires":[]},

{"id":"21eae8f8.2971b8","type":"ui\_chart","z":"c02b79b2.501998","name":"","group":"61285987.c20328","order":0,"width":0,"height":0,"label":"Temperature","chartType":"line","legend":"false","xformat":"HH:mm:ss","interpolate":"linear","nodata":"","dot":false,"ymin":"","ymax":"","removeOlder":1,"removeOlderPoints":"","removeOlderUnit":"3600","cutout":0,"useOneColor":false,"colors":["#1f77b4","#aec7e8","#ff7f0e","#2ca02c","#98df8a","#d62728","#ff9896","#9467bd","#c5b0d5"],"useOldStyle":false,"x":681,"y":276,"wires":[[],[]]},

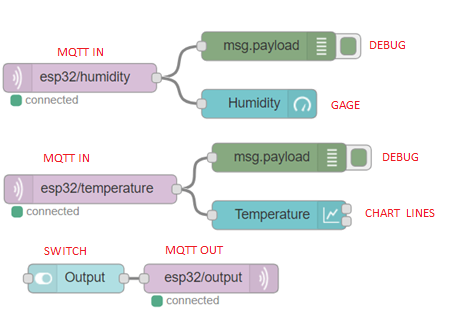
{"id":"df37e6b7.64c1c8","type":"ui\_gauge","z":"c02b79b2.501998","name":"","group":"61285987.c20328","order":0,"width":0,"height":0,"gtype":"gage","title":"Humidity","label":"%","format":"{{value}}","min":0,"max":"100","colors":["#00b3d9","#0073e6","#001bd7"],"seg1":"33","seg2":"66","x":660,"y":160,"wires":[]},

{"id":"10e78a89.5b4fd5","type":"mqtt-broker","z":"","name":"","broker":"localhost","port":"1883","clientid":"","usetls":false,"compatmode":true,"keepalive":"60","cleansession":true,"birthTopic":"","birthQos":"0","birthPayload":"","closeTopic":"","closeQos":"0","closePayload":"","willTopic":"","willQos":"0","willPayload":""},

{"id":"61285987.c20328","type":"ui\_group","z":"","name":"Main","tab":"e7c46d5e.a1283","disp":true,"width":"6","collapse":false},

{"id":"e7c46d5e.a1283","type":"ui\_tab","z":"","name":"Dashboard","icon":"dashboard"}]

The following nodes should load:



After making any changes, click the **Deploy** button to save all the changes.



## Node-RED UI

Now, your Node-RED application is ready. To access Node-RED UI and see how your application looks, access any browser in your local network and type:

http://**Your\_RPi\_IP\_address:1880**/ui

Your application should look as shown in the following figure. You can control the LED on and off with the switch or you can view temperature readings in a chart and the humidity values in a gauge.

