

SMART HOME

Raspberry Pi

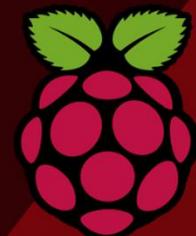
· ESP32

· ESP8266

RUI SANTOS & SARA SANTOS
RANDOM NERD TUTORIALS

SMART HOME

RASPBERRY PI · ESP32 · ESP8266
NODE-RED · INFLUXDB · MQTT



Build a home automation system using Node-RED and InfluxDB on Raspberry Pi to control and monitor ESP32 and ESP8266 boards via MQTT.

Version 1.5

Security Notice

Sorry for writing this notice, but the evidence is clear: piracy for digital products is over the entire internet.

For that reason, we've taken certain steps to protect our intellectual property contained in this eBook.

This eBook contains hidden random strings of text that only apply to your specific eBook version that is unique to your email address. You won't see anything different since those strings are hidden in this PDF. We apologize for having to do that. It means if someone were to share this eBook, we know who did it, and we can take further legal consequences.

You cannot redistribute this eBook. This eBook is for personal use and is only available for purchase at:

- <https://randomnerdtutorials.com/courses>
- <https://rntlab.com/shop>

Please send an email to the author (Rui Santos - hello@ruisantos.me) if you find this eBook anywhere else.

We want to thank you for purchasing this eBook, and we hope you learn a lot and have fun with it!

Disclaimer

This eBook was written for information purposes only. Every effort was made to make this eBook as complete and accurate as possible. The purpose of this eBook is to educate. The authors (Rui Santos and Sara Santos) do not warrant that the information contained in this eBook is fully complete and shall not be responsible for any errors or omissions.

The authors (Rui Santos and Sara Santos) shall have neither liability nor responsibility to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by this eBook.

Throughout this eBook, you will find some links, and some of them are affiliate links. This means the authors (Rui Santos and Sara Santos) earn a small commission from each purchase with that link. Please understand that the authors have experience with all those products and recommend them because they are useful, not because of the small commissions. Please do not spend any money on products unless you feel you need them.

Other Helpful Links:

- [Ask questions in our Forum](#)
- [Join Private Facebook Group](#)
- [Terms and Conditions](#)

Join the Private Facebook Group

This eBook comes with the opportunity to join a private community of like-minded people. If you purchased this eBook, you can join our private Facebook Group today!

Inside the group, you can ask questions and create discussions about everything related to ESP32, ESP32-CAM, ESP8266, Raspberry Pi, Raspberry Pi Pico, Arduino, and much more...

See it for yourself!

1. Go to -> <https://randomnerdtutorials.com/fb>
2. Click the "Join Group" button
3. We'll approve your request within less than 24 hours

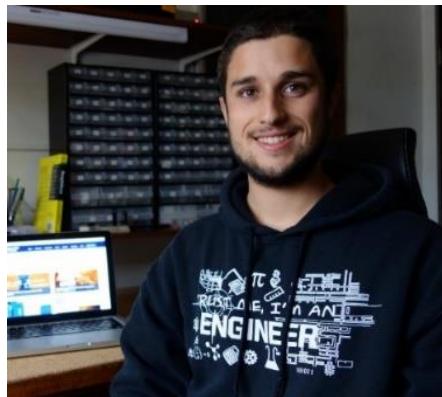


Random Nerd Tutorials Community



About the Authors

This eBook was developed and written by Rui Santos and Sara Santos. We both live in Porto, Portugal, and we have known each other since 2009. If you want to learn more about us, read our [about page](#).



Hi! I'm Rui Santos, the founder of the [Random Nerd Tutorials blog](#). I have a master's degree in Electrical and Computer Engineering from FEUP and I've been running the RNT blog for more than 10 years. I've written hundreds of tutorials covering the usage of different microcontrollers (ESP32, ESP8266, Raspberry Pi, Arduino, and more) on the Internet of Things and Home Automation fields. We also self-published about a [dozen eBooks](#) on these subjects, helping thousands of students, engineers, and hobbyists passionate about electronics all over the world.



Hi! I'm Sara Santos, and I work with Rui at Random Nerd Tutorials since 2015. I have a master's degree in Bioengineering from FEUP. I create, write and edit the tutorials and articles for the [RNT](#) and [Maker Advisor](#) blogs, and I've written several of the eBooks available on the RNT blog. I also help you by answering your questions on our private forum and on our blog's comments section. I love books, writing, cats, and a hot cup of tea. I also love travel and writing about our travel adventures on our [travel blog](#).

Table of Contents

| | |
|--|------------|
| MODULE 0..... | 10 |
| Introduction | 10 |
| Welcome to "SMART HOME with Raspberry Pi, ESP32, and ESP8266"..... | 11 |
| MODULE 1..... | 21 |
| Getting Started with the Raspberry Pi | 21 |
| 1.1 - Introducing the Raspberry Pi | 22 |
| 1.2 - Installing the Operating System..... | 33 |
| 1.3 - Connecting via SSH to the RPi..... | 41 |
| MODULE 2..... | 49 |
| Getting Started with Node-RED..... | 49 |
| 2.1 - Node-RED Introduction..... | 50 |
| 2.2 - Installing Node-RED | 52 |
| 2.3 - Node-RED Overview..... | 59 |
| 2.4 - Node-RED Dashboard..... | 72 |
| 2.5 - Controlling an LED with Node-RED..... | 89 |
| MODULE 3..... | 101 |
| Getting Started with MQTT..... | 101 |
| 3.1 - Introducing MQTT | 102 |
| 3.2 - Installing Mosquitto Broker..... | 109 |
| 3.3 - MQTT with Node-RED | 117 |

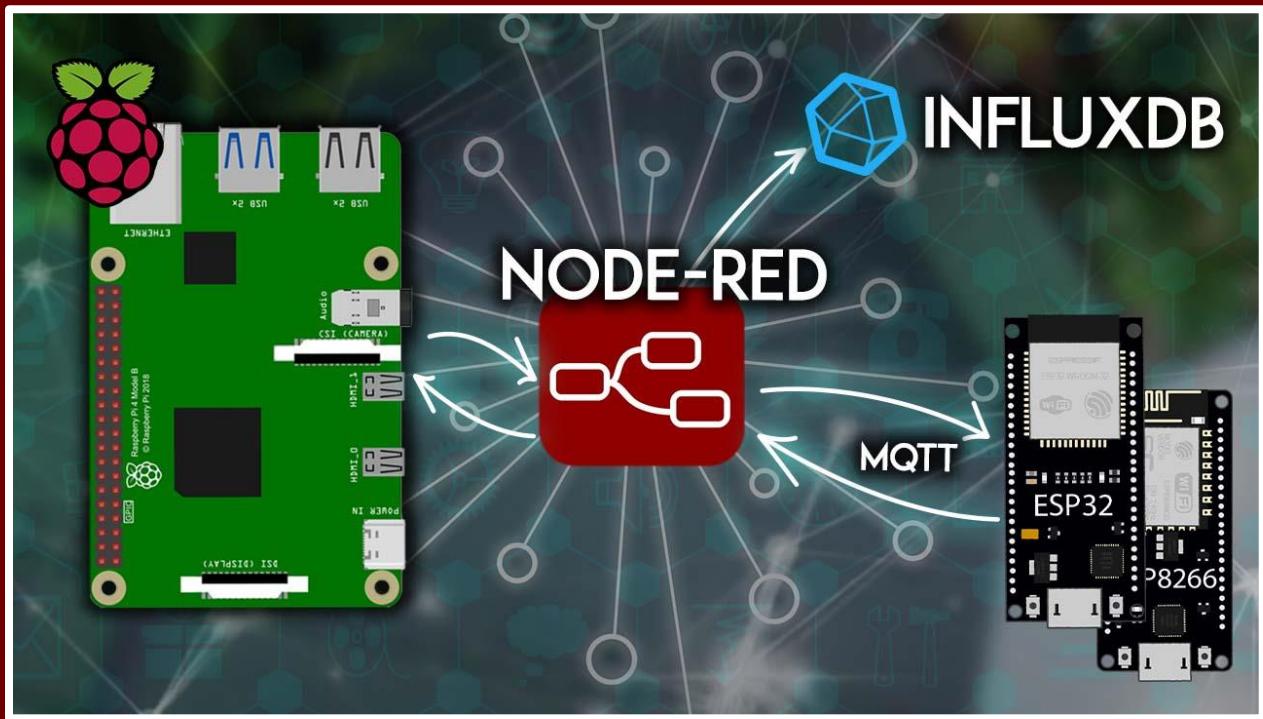
| | |
|---|------------|
| MODULE 4..... | 128 |
| Introducing the ESP32 and ESP8266 Boards | 128 |
| 4.1 - Introducing the ESP8266 | 129 |
| 4.2 - Introducing the ESP32..... | 135 |
| 4.3 - Installing Arduino IDE | 141 |
| 4.4 - Testing the Installation | 146 |
| MODULE 5..... | 150 |
| Connecting the ESP32/ESP826 with Node-RED (MQTT)..... | 150 |
| 5.1 - Connecting the ESP32/ESP8266 with Node-RED (Introduction) | 151 |
| 5.2 - Publish and Subscribe (Basic Example)..... | 153 |
| 5.3 - Publish Sensor Readings | 182 |
| 5.4 - Subscribe to Multiple Topics and Control Multiple Outputs..... | 205 |
| MODULE 6..... | 227 |
| InfluxDB Time-Series Database | 227 |
| 6.1 - Getting Started with InfluxDB | 228 |
| 6.2 - Install InfluxDB (Raspberry Pi)..... | 231 |
| 6.3 - Monitoring your Raspberry Pi using InfluxDB Telegraf | 238 |
| 6.4 - MQTT + Node-RED + InfluxDB..... | 248 |
| 6.5 - Monitoring GPIO States on InfluxDB | 263 |
| 6.6 - Getting Data from InfluxDB..... | 279 |
| 6.7 - Deleting InfluxDB Data | 286 |
| MODULE 7..... | 292 |

| | |
|--|------------|
| Sending Notifications with Node-RED | 292 |
| 7.1 - Email Alerts with Node-RED | 293 |
| 7.2 - Telegram Messages with Node-RED..... | 302 |
| 7.3 - WhatsApp Messages with Node-RED | 311 |
| 7.4 - Motion Detector with Notifications | 317 |
| MODULE 8..... | 348 |
| Adding Rules and Triggering Events..... | 348 |
| 8.1 - Creating Master Switches or Modes..... | 349 |
| 8.2 - Triggering Events Based on Threshold Values | 363 |
| 8.3 - Time-based Events | 373 |
| 8.4 - Time-based Events with Big Timer | 380 |
| MODULE 9..... | 390 |
| Accessing Your Local System from Anywhere | 390 |
| 9.1 - Accessing Your System from Anywhere..... | 391 |
| MODULE 10 | 406 |
| Creating a Cloud Server..... | 406 |
| 10.1 - Introduction (Digital Ocean)..... | 407 |
| 10.2 - Installing Node-RED on Digital Ocean | 413 |
| 10.3 - Mosquitto Cloud MQTT Broker | 420 |
| 10.4 - InfluxDB on Digital Ocean | 425 |
| 10.5 - Setting up a Custom Domain with Cloudflare (HTTPS)..... | 428 |
| MODULE 11 | 441 |

| | |
|--|------------|
| Setting up a Surveillance Camera | 441 |
| 11.1 – Introducing the ESP32-CAM..... | 442 |
| 11.2 – ESP32-CAM Video Streaming..... | 446 |
| 11.3 – Surveillance Camera on Node-RED..... | 453 |
| APPENDIX..... | 461 |
| Linux Commands and More..... | 461 |
| Sending Linux Commands Through the Node-RED UI | 462 |
| Learning Basic Linux Commands..... | 466 |
| WRAPPING UP | 476 |
| Congratulations for completing this course! | 476 |
| Congratulations for completing this course! | 477 |
| Other RNT Courses/eBooks..... | 478 |

MODULE 0

Introduction



This Module introduces the eBook. It covers what you'll learn and build, instructions on how to follow this eBook, and recommended prerequisites. We'll also give you a quick introduction to the main subjects we'll cover throughout the eBook: MQTT, Node-RED software, InfluxDB time-series database, Raspberry Pi, ESP32, and ESP8266 boards.

Welcome to "SMART HOME with Raspberry Pi, ESP32, and ESP8266"

Welcome to the "**SMART HOME with Raspberry Pi, ESP32, and ESP8266**" eBook. This is a hands-on introductory course designed to teach you how to build a home automation system using open-source hardware and software.

This training is addressed to those who find home automation and the internet of things (IoT) subjects interesting.

There's no previous knowledge required to complete the course, but some previous basic knowledge about the ESP32 or ESP8266 boards is beneficial. If there's something extra that you need to learn during the course, we'll point you to the right resource.

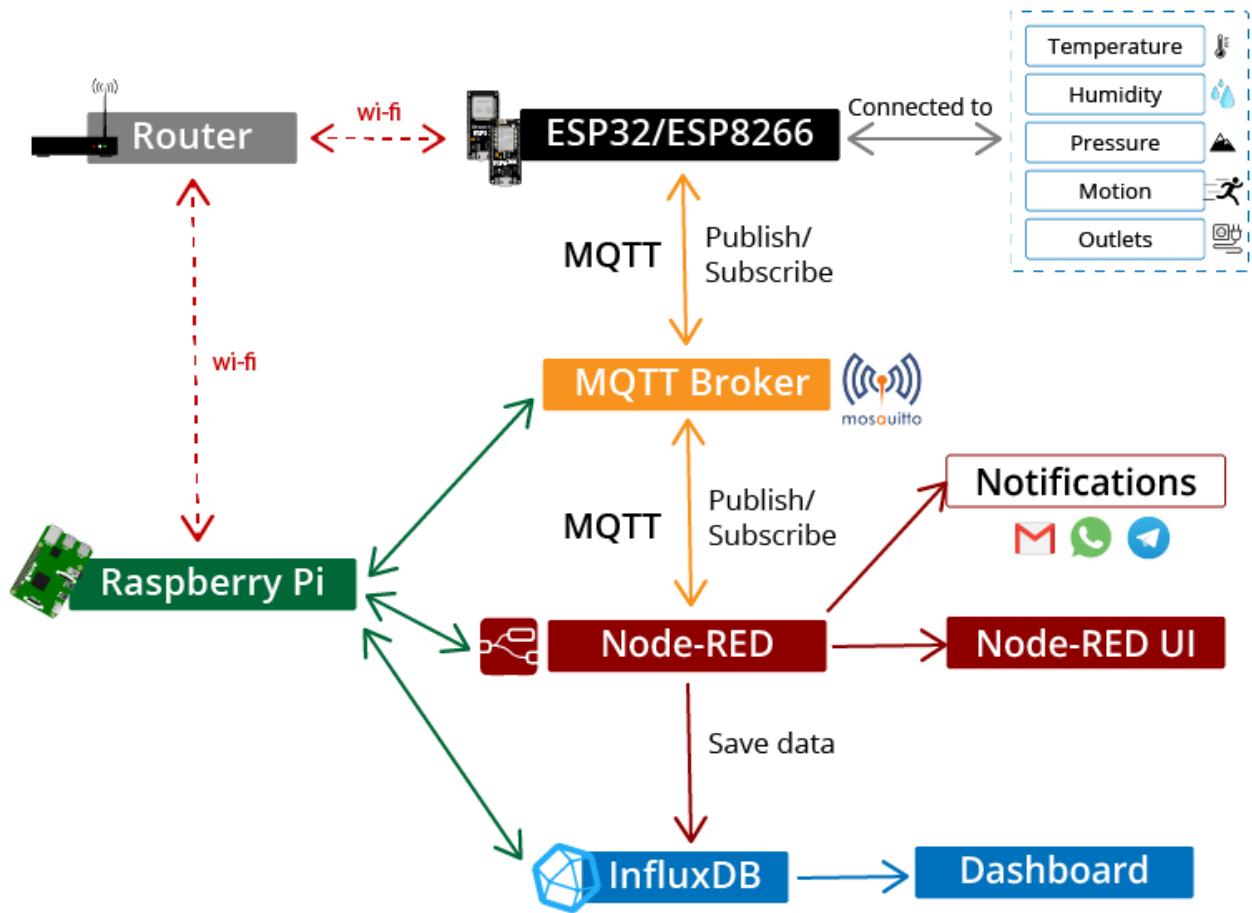
We use a step-by-step teaching approach, so all the modules and corresponding units are straightforward to follow even with no previous experience.

Throughout this eBook, you'll learn how to build a home automation system and we'll cover the following main subjects: Node-RED, Node-RED Dashboard, Raspberry Pi, ESP32, ESP8266, MQTT, and InfluxDB database.

Application Overview

By the end of the course, you'll have a home automation application running on your Raspberry Pi that allows you to monitor and control various ESP32 and/or ESP8266 devices in your house. Then, you can connect the ESP boards to actuators to control something and to sensors to monitor your house.

The following diagram shows a quick overview of the system we're going to build.



- The brain of the operation is the Raspberry Pi. The Raspberry Pi hosts Node-RED software, the MQTT broker, and InfluxDB.
- You'll learn how to connect the ESP32 and ESP8266 boards to your Raspberry Pi. They'll communicate via a protocol called MQTT that uses a publish/subscribe system.
- You can control the ESP32 and ESP8266 boards via MQTT from Node-RED.
- The ESP can send sensor readings to Node-RED via MQTT. The readings can be displayed on gauges and charts on Node-RED UI.
- Node-RED can send the sensor readings and any other data to InfluxDB.
- InfluxDB provides tools to build customizable awesome charts and graphs to display and monitor your data on a dashboard.

- Node-RED can communicate with third-party services to send you notifications. We'll cover sending emails, WhatsApp messages, and Telegram messages.
- You can create rules on Node-RED and schedule events to make things happen automatically. For example, turn something on or off depending on sensor readings, day and time of the week, etc.
- All of this is accessible on your local network.
- At the end, we'll show you how you can create a secure tunnel to access your home automation system from anywhere in the world.
- Finally, you'll also learn how to add a video surveillance camera to your system using an ESP32-CAM.

eBook Overview

This eBook contains 11 Modules and one Appendix:

- **Module 1: Getting Started with the Raspberry Pi**—a quick introduction to the Raspberry Pi board and setting it up to use throughout this course.
- **Module 2: Getting Started with Node-RED**—install Node-RED on the Raspberry Pi, get familiar with it, and start creating simple flows.
- **Module 3: Getting Started with MQTT**—set up everything you need to set the Raspberry Pi as an MQTT server to communicate with other devices.
- **Module 4: Introducing the ESP32 and ESP8266**—a quick introduction to the ESP32 and ESP8266 boards and how to program them using Arduino IDE.
- **Module 5: Connecting the ESP32/ESP8266 with Node-RED (MQTT)**—learn how to establish communication between the ESP boards and Node-RED using MQTT (publish and subscribe) to control outputs and monitor sensors.

- **Module 6: InfluxDB Time-Series Database**—get started with InfluxDB, a time-series database. Learn how to save your data and create dashboards with beautiful and easily-customizable charts.
- **Module 7: Sending Notifications with Node-RED**—learn how to send emails, Telegram messages, and WhatsApp messages via Node-RED to receive a warning, sensor readings, or any other useful information.
- **Module 8: Adding Rules and Triggering Events**—learn how to automate your system by adding rules (like sensor threshold values) and scheduling events based on time and day of the week.
- **Module 9: Access Your Local System from Anywhere**—set up an online cloud server with Node-RED, MQTT broker, and Influx DB so that you can access your home automation system from anywhere in the world.
- **Module 10: Creating a Cloud server**—if you don't have a Raspberry Pi to follow along, you can create your system on the cloud. We provide instructions for installing Node-RED, InfluxDB, and Mosquitto MQTT broker on Digital Ocean (hosting).
- **Module 11: Setting up a Surveillance Camera**—learn how to set up a video streaming web server with the ESP32-CAM and how to add it to your home automation system.
- **Appendix**— quick guide with Linux commands and how to execute Linux commands via Node-RED.

What You'll Learn

By following this eBook, you'll learn about the following subjects:

- **Raspberry Pi:**
 - Get familiar with the Raspberry Pi board;

- Learn how to set up the Raspberry Pi board with the required software and how to establish an SSH connection;
 - Learn basic Linux commands;
 - Control the Raspberry Pi GPIOs using Node-RED.
- **MQTT:**
 - What is MQTT, how does it work, and how to use it in your home automation system;
 - Install an MQTT broker on your Raspberry Pi to handle MQTT messages;
 - Use MQTT to communicate between Node-RED and other devices (ESP32 and ESP8266).
 - **Node-RED:**
 - Learn how to create flows in Node-RED to execute tasks;
 - Build publish and subscribe system flows;
 - Set up a notification system;
 - Schedule events and set up rules;
 - Interface with InfluxDB: send data and make queries.
 - **Node-RED Dashboard:**
 - Create tabs and groups to display different widgets: buttons, text, charts, and gauges;
 - Build flows with switches to control the ESP32/ESP8266 GPIOs;
 - Use charts and gauges to visualize sensor readings.
 - **ESP32 and ESP8266:**
 - Subscribe to MQTT topics (listen to Node-RED messages) and do different tasks depending on the received messages;
 - Publish sensor readings to MQTT topics (send to Node-RED);
 - Subscribe to multiple topics simultaneously;
 - Use interrupts to publish MQTT messages when motion is detected.

- **InfluxDB:**
 - Install InfluxDB on a Raspberry Pi and use Influx Telegraf to monitor the Raspberry Pi system;
 - Create buckets to save data and build dashboards with different visualizations;
 - Send data from Node-RED to InfluxDB;
 - Make queries on Node-RED to get data from InfluxDB.

How to Follow this eBook

We recommend following this eBook linearly, but that's not mandatory. You need to follow Modules 1, 2, 3, and 4 before proceeding to any of the other Modules. You can skip the introductory sections of subjects you're already familiar with.

In some of the Units that you need to create a Node-RED flow, we provide a file with the complete flow that you can upload to Node-RED. However, we always recommend that you follow the step-by-step instructions and build the flow yourself, instead of using the flow from the file.

How to Follow Without a Raspberry Pi

We'll use a Raspberry Pi to install Node-RED, InfluxDB, and Mosquitto MQTT broker. If you don't have a Raspberry Pi, we provide alternative instructions using a virtual machine Linux system on the cloud. We provide instructions using Digital Ocean, which is a hosting service.

Here's what you need to take into account:

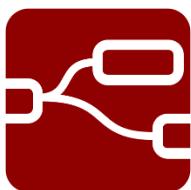
- Instead of following Module 1, follow Unit 10.1 to set up a Linux server on Digital Ocean;
- Instead of Unit 2.2, follow Unit 10.2;

- Don't follow Unit 2.5, which is about controlling the RPi GPIOs;
- Instead of Unit 3.2, follow Unit 10.3;
- Don't follow Unit 3.3, which is about controlling the RPi GPIOs;
- Instead of Unit 6.2, follow Unit 10.4;
- Don't follow Unit 6.3, which is about monitoring the Raspberry Pi;
- Instead of Unit 9.1, follow Unit 10.5.

In summary:

- ~~Module 1~~ → **Unit 10.1**
- ~~Unit 2.2~~ → **Unit 10.2**
- ~~Unit 2.5~~
- ~~Unit 3.2~~ → **Unit 10.3**
- ~~Unit 3.3~~
- ~~Unit 6.2~~ → **Unit 10.4**
- ~~Unit 6.3~~
- ~~Unit 9.1~~ → **Unit 10.5**

Are Node-RED and InfluxDB Free to Use?



Node-RED and InfluxDB are free to use locally on your Raspberry Pi. You won't need to pay anything to use that software. In Module 9, if you want to access your Home Automation System outside your local network, you'll need a domain name. You can use a free domain name or you can get a paid domain name to create a secure tunnel with Cloudflare.

If you don't have a Raspberry Pi and use a Linux server on the cloud (Module 10), you'll need to pay for hosting service.

Prerequisites (recommend but not mandatory)

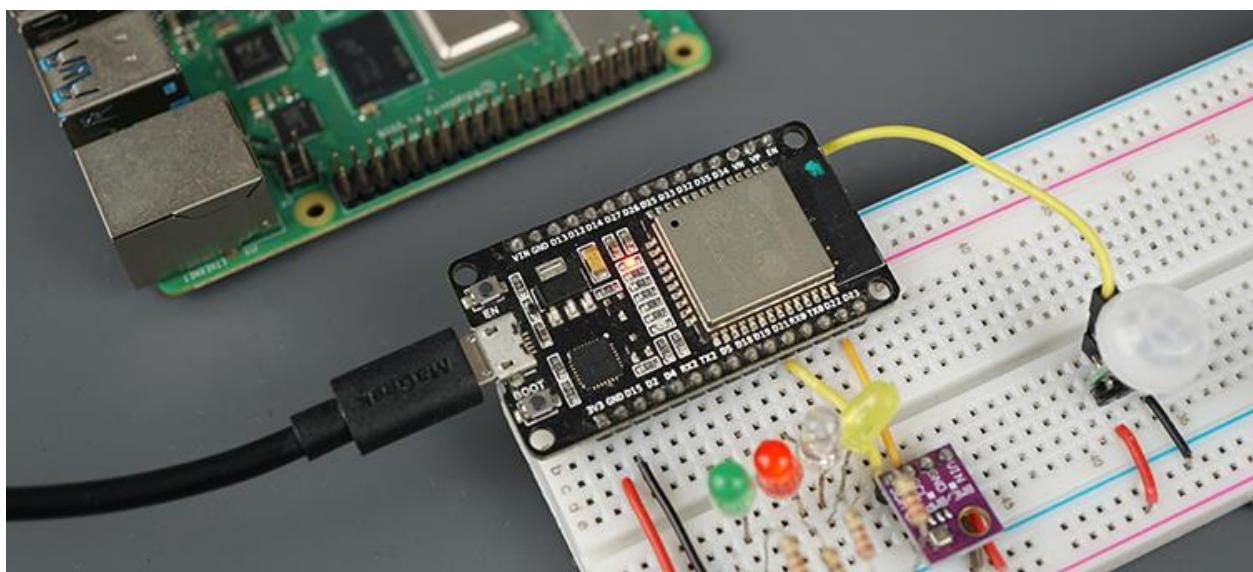
To get a better experience following along with the examples in this eBook, we recommend having some previous knowledge about the following:

Basic programming of the ESP32 and/or ESP8266 with the Arduino Core, like controlling outputs and reading sensors. To get familiar with these boards, you can follow our free tutorials or our premium eBooks:

- [Free ESP32 Tutorials](#)
- [Free ESP8266 Tutorials](#)
- [Learn ESP32 with Arduino IDE \(eBook + Video course\)](#)
- [Home Automation using ESP8266 \(eBook\)](#)

Parts Required

The following list shows all the parts required to complete the projects and examples in this eBook:



- [Raspberry Pi 5](#) or Raspberry Pi 4 (preferable)—also compatible with the Raspberry Pi 3 B+ *
- [MicroSD card](#) for the Raspberry Pi OS (at least 16GB class 10) *
- [Raspberry Pi power supply](#) (5V DC 2A power supply, USB-C) *
- [ESP32 board](#) and/or [ESP8266 board](#)
- [Breadboard](#)
- [Jumper wires](#)
- [LEDs](#) (at least four)
- [220 Ω resistor](#) or similar values (at least four)
- [BME280 sensor temperature, humidity, and pressure sensor](#) (alternatively, you can use any other sensor you're familiar with)
- [Mini PIR motion sensor \(AM312\)](#) or [PIR motion sensor \(HC-SR501\)](#)
- [ESP32-CAM](#)

* Alternatively, you can buy a [Raspberry Pi starter kit](#) that usually comes with all the essential accessories.

Download Source Code and Resources



Each section contains the code and resources you need to complete each step and proceed to the following sections. You can download all the resources for a specific example on the corresponding unit.

Alternatively, you can download the project repository and instantly download all the resources.

- [Download All eBook Resources »](#)

Problems and Difficulties Throughout the Course

As you go through the course, you'll likely encounter some sort of difficulty or technical problem. I highly encourage you to spend a bit of time trying to fix technical problems by yourself. Fixing technical problems yourself is a very good way to learn a new subject.

If you have done your best but you couldn't find the solution for your issue, you can always rely on the community to help you out. You can use the [Private Forum](#) (recommended), [Facebook group](#), or our [Support form](#).

Check the Errata

Before starting any project, please check the errata on the eBook page on the following link to check if there's anything that needs to be fixed.

- <https://rntlab.com/smart-home-raspberry-pi-esp32-esp8266/>

If there isn't an errata on that page, it means there's nothing wrong you need to worry about.

Leave Feedback

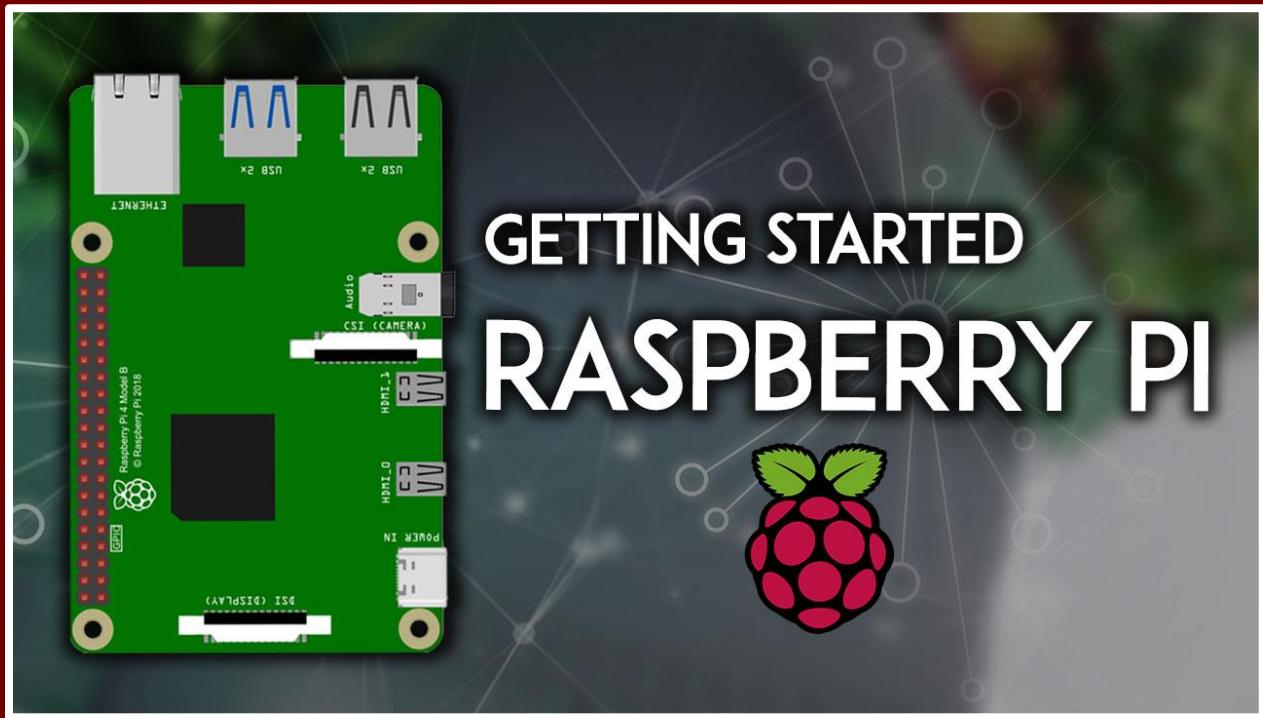
Your feedback is very important so that we can improve the eBook and our learning materials. Suggestions, rectifications, and your opinion are essential.

You can use the following channels to leave feedback:

- [Support form](#)
- [RNT Lab Forum](#)
- [Facebook group](#)

MODULE 1

Getting Started with the Raspberry Pi

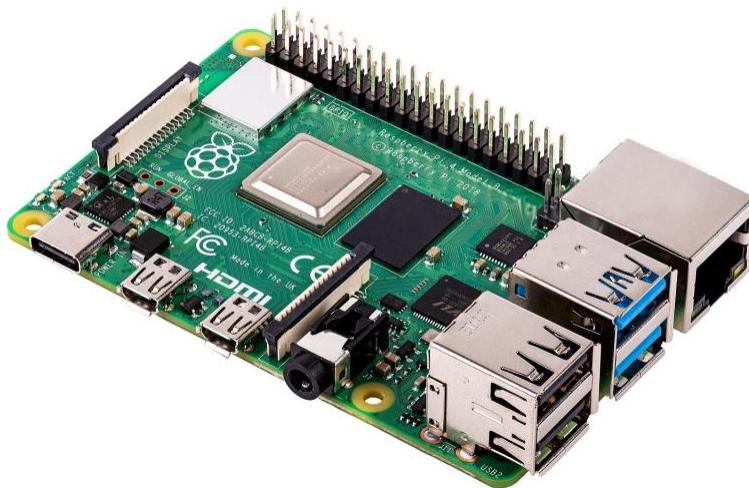


This Module is a quick introduction to the Raspberry Pi. You'll set up your Raspberry Pi for the first time: install the operating system and establish an SSH connection with your Pi using your computer.

1.1 - Introducing the Raspberry Pi

This unit is a quick introduction to the Raspberry Pi. If you're already familiar with the Raspberry Pi, you can skip to unit 1.2. Otherwise, take a quick look at the present section.

The Raspberry Pi is a small computer board about the size of a credit card. It was developed in the United Kingdom by the Raspberry Pi Foundation to promote basic computer science teaching in schools. Since its first general sale in 2012, more than 46 million Raspberry Pi boards have been sold by February 2022. The picture below shows a Raspberry Pi 4 model B+. The most recent is the Raspberry Pi 5.



The Raspberry Pi has become tremendously popular among kids, electronics hobbyists, experienced makers, tinkerers, and even computer scientists. The Raspberry Pi is hackable and small. So, it's the perfect solution for tinkerers!

The Raspberry Pi board used to cost around \$35. However, increasing demand, constraints in the supply chain, and a shortage of chips caused the Raspberry Pi price to skyrocket to more than \$150 (at the time of writing this eBook). Additionally, in many cases, you may need to wait a lot to get one.

Raspberry Pi (Desktop Computer vs Headless)

In some way, you can look at the Raspberry Pi like a normal computer, it has a processor, RAM, USB ports to plug a keyboard and a mouse, an HDMI port to plug a TV or monitor, and you can even connect it to the internet.

You can do most things you do with a regular computer like web browsing, document editing, playing games, coding, and much more. The figure below shows the Raspberry Pi set up as a desktop computer.



Raspberry Pi Headless

However, you don't even need those accessories (mouse, keyboard, and monitor) if you don't want to set it as a Desktop computer. It can run headless and you can control it remotely using Linux commands via a Terminal after establishing an SSH connection—that's what we're going to do throughout this eBook.

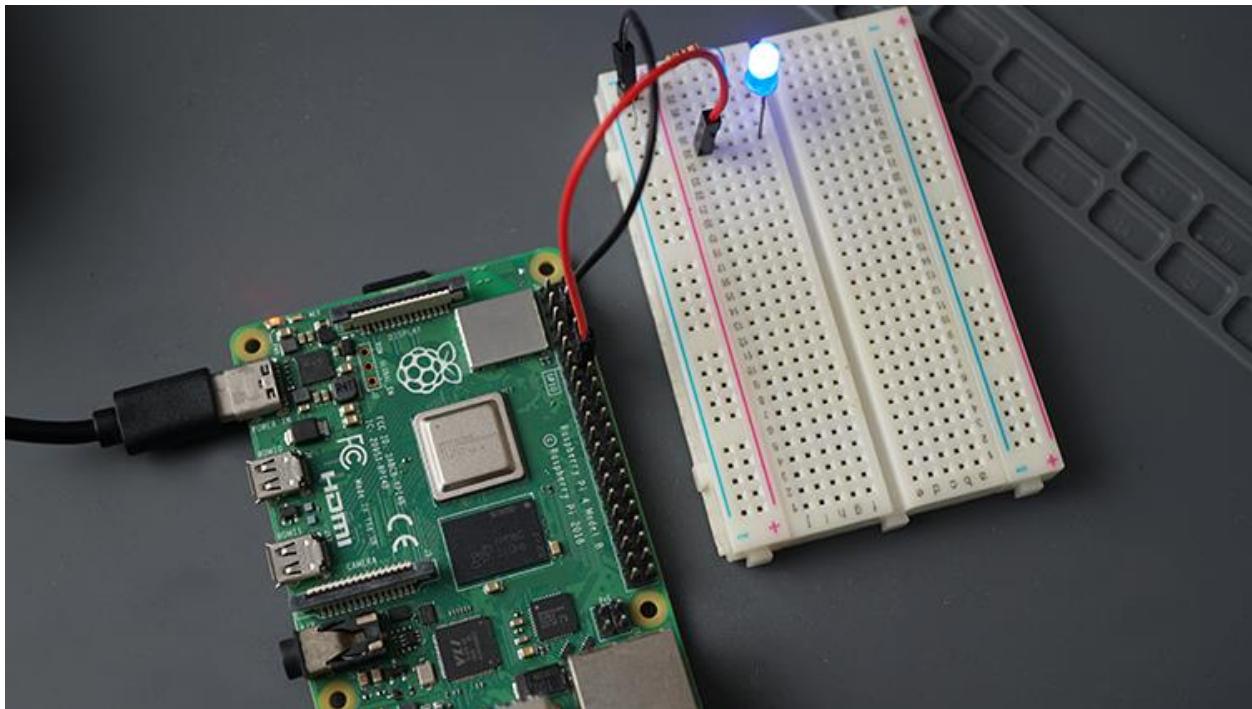
```
pi@raspberrypi: ~
login as: pi
pi@192.168.1.98's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Dec 18 17:35:56 2015 from desktop-c5bt0f.lan
pi@raspberrypi:~ $
```

Raspberry Pi GPIOs

The Raspberry Pi board has one special feature that normal computers don't: General Purpose Input Output (GPIOs) Pins. These GPIOs let you interact with the real world allowing you to build great electronics projects. Inputs can read data from sensors. Output signals can be sent to actuators to turn something on and off.



Raspberry Pi Applications

There are no limits to what you can do with your Raspberry Pi. Here are just some examples:

- Write programs;
- Create [electronics projects](#);
- Build a web server;
- Build a home automation system;
- Use it as a local server for Home Automation applications;
- Set it up as a gateway for your IoT projects;
- Use it as private cloud storage;
- Host an MQTT broker;
- Build a retro gaming console;
- Use your Pi as a desktop computer;
- Make your own CCTV system;
- Robotics;
- And much more.

Different Raspberry Pi Boards

There are different releases of the Raspberry Pi board. Here's a list of the most relevant:

- February 2012: Raspberry Pi 1 Model B (Rev. 1)
- April 2012: Raspberry Pi 1 Model B (Rev. 2)
- February 2013: Raspberry Pi 1 Model A
- July 2014: Raspberry Pi 1 Model B+
- November 2014: Raspberry Pi 1 Model A+
- February 2015: Raspberry Pi 2 Model B

- November 2015: Raspberry Pi Zero
- February 2016: Raspberry Pi 3 Model B
- February 2017: Raspberry Pi Zero W
- 2016: Raspberry Pi 3 Model B
- March 2018: [Raspberry Pi 3 Model B+](#)
- June 2019: [Raspberry Pi 4 Model B](#)
- October 2023: [Raspberry Pi 5](#)

If you don't have a Raspberry Pi board yet, we recommend you get a [Raspberry Pi 5](#), [Raspberry Pi 4](#), or a Raspberry Pi 3 B+.

Raspberry Pi 5 Features

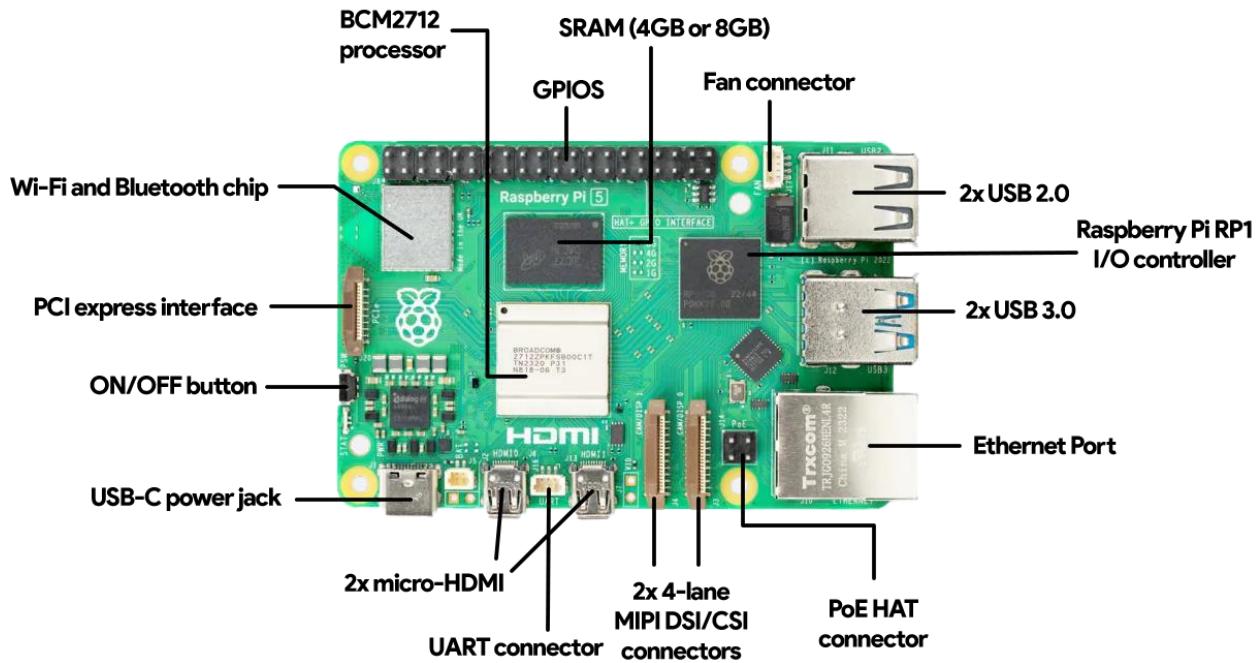


In the following table you can take a look at the Raspberry Pi 5 features:

| | |
|---------------------------|--|
| Architecture | ARM v8-A 64-bit |
| SoC | Broadcom BCM2712 |
| CPU | 2.4 GHz 64-bit quad-core ARM Cortex-A76 |
| RAM | 4GB or 8GB LPDDR4X-4267 SDRAM (depending on the model) |
| Wireless LAN | Dual-band 802.11ac Wi-Fi |
| Bluetooth | Bluetooth 5.0 and BLE |
| Supports PoE | Yes (requires separate PoE HAT) |
| GPIOs | 40 |
| Ports | 2x micro HDMI (dual 4Kp60 HDR) 3.5 mm analog audio-video jack 2x USB 3.0, and 2x USB 2.0 Gigabit Ethernet |
| Camera Interfaces | 2 × 4-lane MIPI camera/display transceivers |
| Display Interfaces | 2 × 4-lane MIPI camera/display transceivers |
| Where to Buy? | Get a Raspberry Pi Official Raspberry Pi Website |

Exploring the Raspberry Pi 5 Model B Board

The figure below shows the Raspberry Pi 5 Model B annotated—most models are similar despite some small differences in the available ports.



Here's a glance at some of the components of the Raspberry Pi:

- **USB ports:** to connect a mouse, a keyboard, or other peripherals. It comes with two USB 3.0 and two USB 2.0 ports;
- **Ethernet port:** to connect to the internet using an Ethernet cable;
- **DSI/CSI connectors:** to connect a display or a [camera with a CSI ribbon](#); you can have one of each, or two displays or two cameras;
- **HDMI connector:** to connect a monitor or TV;
- **Processor:** is the brain of the Raspberry Pi;
- **MicroSD card slot** (under the PCI express interface): to insert a microSD card to store your files and your operating system;
- **USB-C power jack:** to power up your Pi;

- **GPIOs** (general purpose input output pins): connect devices to interact with the outside world like sensors and outputs like LEDs, relays, and motors.

Accessories You Need To Get Started

When you buy a Raspberry Pi board, you only get a bare electronic board that doesn't do much on its own. You need several accessories to get started.

There are a lot of accessories for the Raspberry Pi, but you need at least a microSD card and a power supply.



- **Power supply**: you need a power adapter that provides 2.5A 5V (make sure you check the appropriate power source for the model you're using)
- **MicroSD card**: we recommend getting a microSD card with at least 16GB, class 10 (or 32GB is even better). You need a microSD card to store your files and the Pi's operating system. The Pi doesn't have a hard drive*, so everything you do on your Pi is saved on the microSD card, even the operating system. You can get a microSD card with the operating system preloaded or install the operating system yourself (which we recommend).

* you can get an expansion board that lets you add a SATA hard disk drive (HDD) or solid-state drive (SSD).

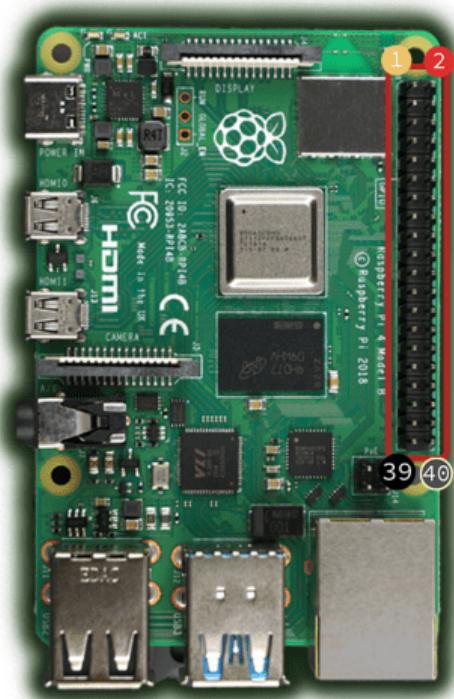
Getting a Raspberry Pi Starter Kit

If this is your first time around the Raspberry Pi, you may consider getting a [Raspberry Pi starter kit](#) that comes with the essential accessories to get started. We have a dedicated article comparing different Raspberry Pi kits: [post about the Best Raspberry Pi Starter Kits.](#)



Raspberry Pi GPIOs Pinout

The following figure shows the Raspberry Pi GPIOs pinout that you can use for future reference.



| | | | |
|---------------|--------|----|----|
| 3V3 | GPIO2 | 1 | 2 |
| I2C SDA | GPIO3 | 3 | 4 |
| I2C SCL | GND | 5 | 6 |
| GPIO4 | | 7 | 8 |
| GND | | 9 | 10 |
| GPIO17 | | 11 | 12 |
| GPIO27 | | 13 | 14 |
| GPIO22 | | 15 | 16 |
| 3V3 | | 17 | 18 |
| SPI MOSI | GPIO10 | 19 | 20 |
| SPI MISO | GPIO9 | 21 | 22 |
| SPI SCLK | GPIO11 | 23 | 24 |
| GND | | 25 | 26 |
| I2C ID EEPROM | GPIO0 | 27 | 28 |
| | GPIO5 | 29 | 30 |
| | GPIO6 | 31 | 32 |
| PWM1 | GPIO13 | 33 | 34 |
| PCM FS | GPIO19 | 35 | 36 |
| | GPIO26 | 37 | 38 |
| GND | | 39 | 40 |

This pinout is the same for Raspberry Pi 5, Raspberry Pi 4, Raspberry Pi 3, Raspberry Pi 2 Model B, Raspberry Pi 1 Model A+, Raspberry Pi Model B+, Raspberry Pi Zero, and Raspberry Pi Zero W. Raspberry Pi 1 Model A and the Raspberry Pi 1 Model B Rev.2 only have the first 26 pins.

For a more detailed description of the Raspberry Pi Pinout, we recommend you reading the following article:

- [Raspberry Pi Pinout Guide: How to use the Raspberry Pi GPIOs?](#)

You can also use the following simpler diagram as a reference for the Raspberry Pi GPIOs.

Raspberry Pi GPIO BCM numbering

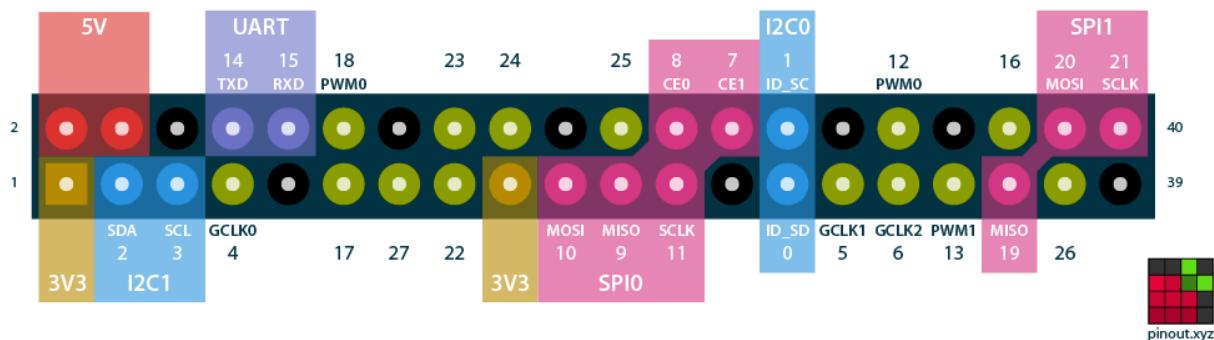


Image source: pinout.xyz

We won't use the Raspberry Pi GPIOs much in this eBook. The Raspberry Pi will act as a gateway that communicates with other devices like the ESP32 and ESP8266 boards, which will then use its GPIOs to read sensors and control outputs. Nevertheless, it's good to be familiar with the Raspberry Pi GPIOs and have a GPIO reference guide.

1.2 - Installing the Operating System

In this section, you'll learn how to install the operating system, set up Wi-Fi, and enable and connect with SSH.

MicroSD Card for Raspberry Pi

The [Raspberry Pi](#) is a computer and like any other computer, it needs an Operating System (OS) installed. The Pi doesn't have built-in memory, so you'll need a [microSD card](#) to install your OS. We'll install the operating system on the microSD card. I recommend using a [microSD card class 10 with at least 16GB of memory](#). Preferably, use a 32GB SD card.



Installing Raspberry Pi OS

We'll install the **Raspberry Pi OS (64-bit)** on the Raspberry Pi.

- 1) Start by connecting the microSD card to your computer.
- 2) Go to the [Raspberry Pi Software page](#).
- 3) Download the **Raspberry Pi Imager** (a tool to flash the OS on the microSD card) for your computer's operating system.

Raspberry Pi Imager is the quick and easy way to install Raspberry Pi OS and other operating systems to a microSD card, ready to use with your Raspberry Pi.

Download and install Raspberry Pi Imager to a computer with an SD card reader. Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.

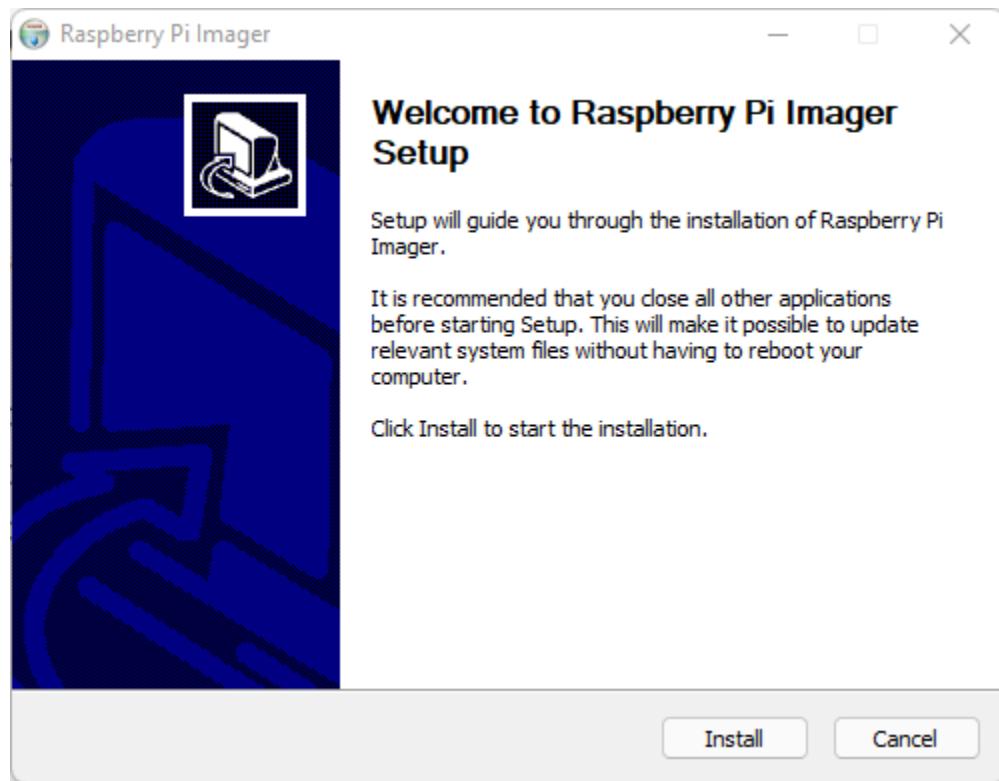
[Download for Windows](#)

[Download for macOS](#)

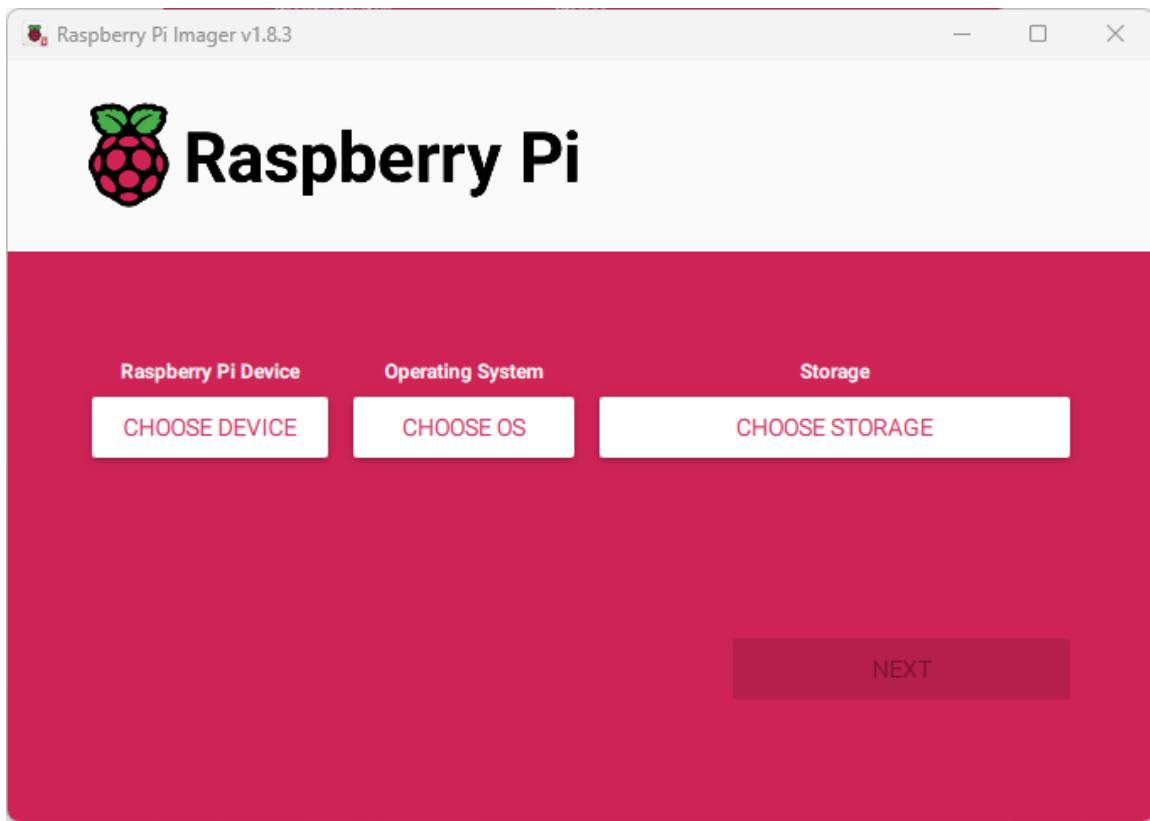
[Download for Ubuntu for x86](#)

The screenshot shows a browser window with the URL raspberrypi.com/software/. The main content is titled "Install Raspberry Pi OS using Raspberry Pi Imager". It includes a brief description of what Raspberry Pi Imager does, instructions on how to download it, and links to download it for Windows, macOS, and Ubuntu for x86. To the right of the text, there is a small screenshot of the "Raspberry Pi Imager v1.8.1" software interface, which has a red header with tabs for "CHOOSE DEVICE", "CHOOSE OS", and "CHOOSE STORAGE", and a "NEXT" button at the bottom.

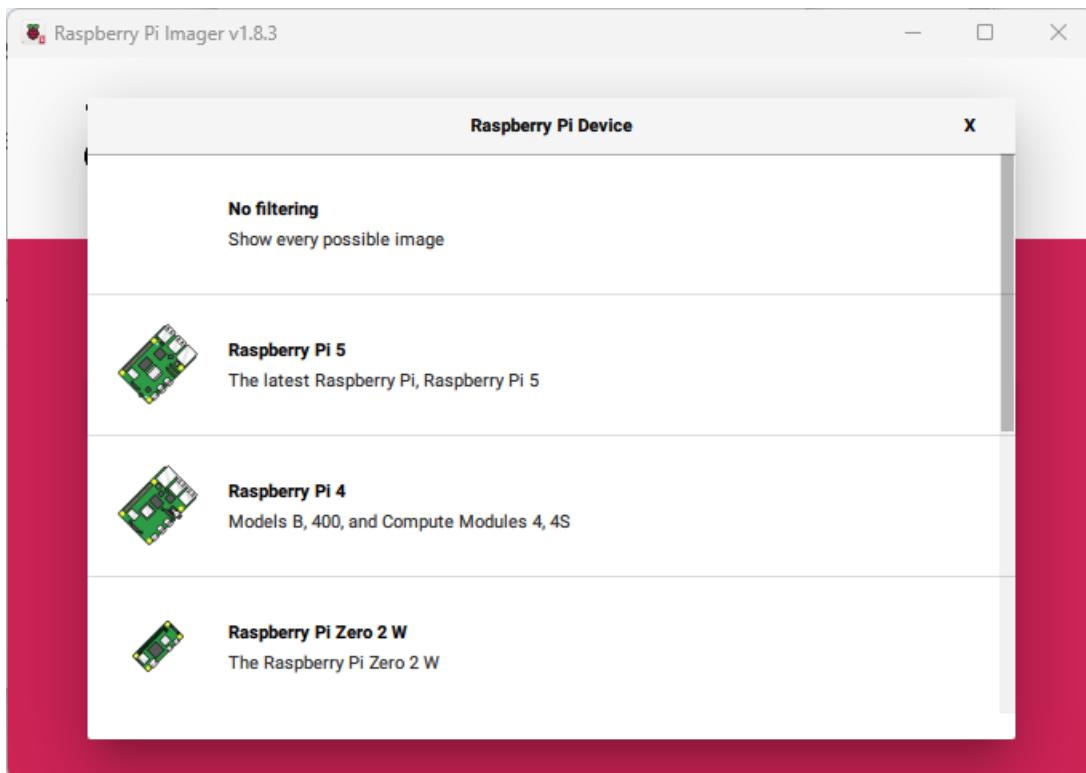
- 4) Click on the downloaded file to install the Raspberry Pi Imager.



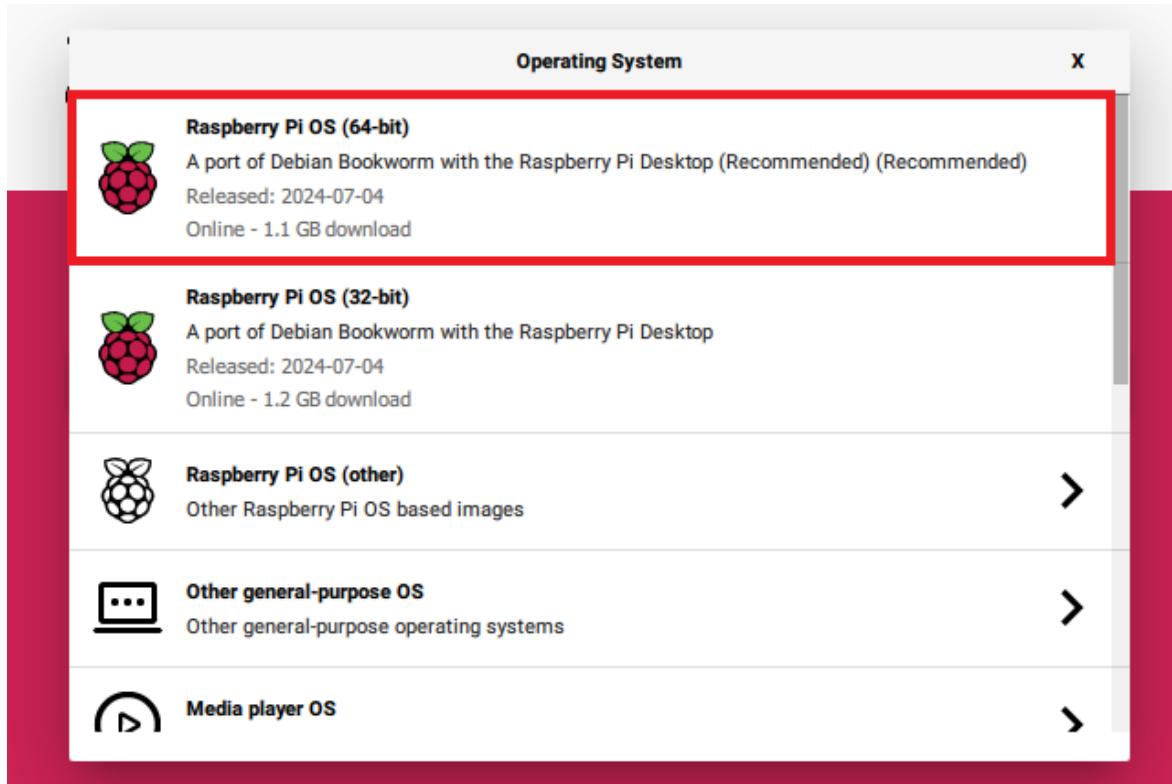
- 5) When the installation is complete, the Raspberry Pi Imager will open.



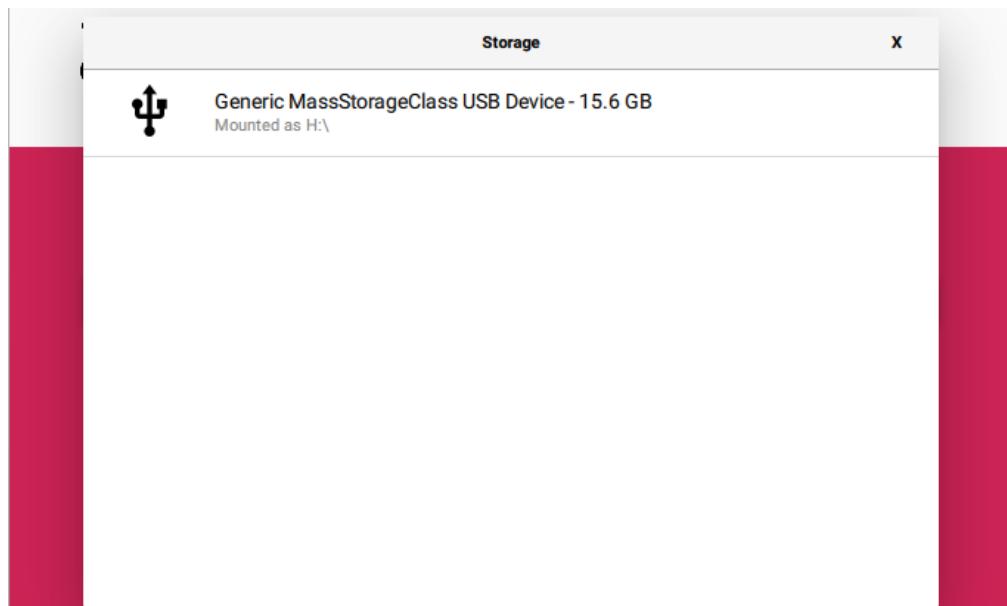
- 6) Click on **Choose Device** to select the Raspberry Pi board you're using.



- 7) Click on **Choose OS** to select the Operating System. Select the **Raspberry Pi OS (64-bit)**.

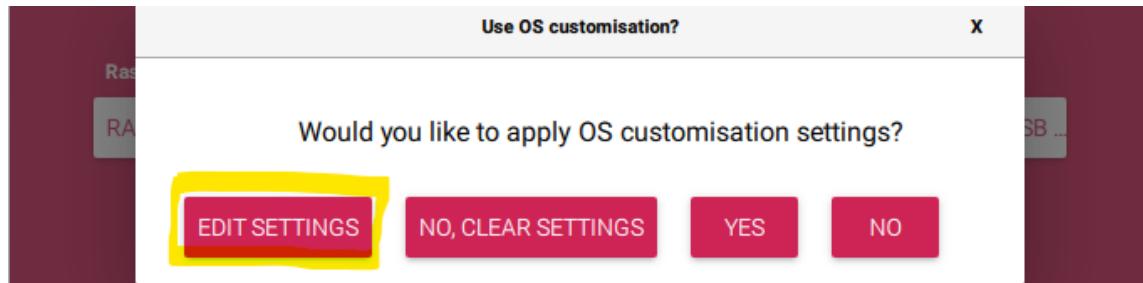


- 8) Choose storage. You must choose the microSD card where you want to install the OS.

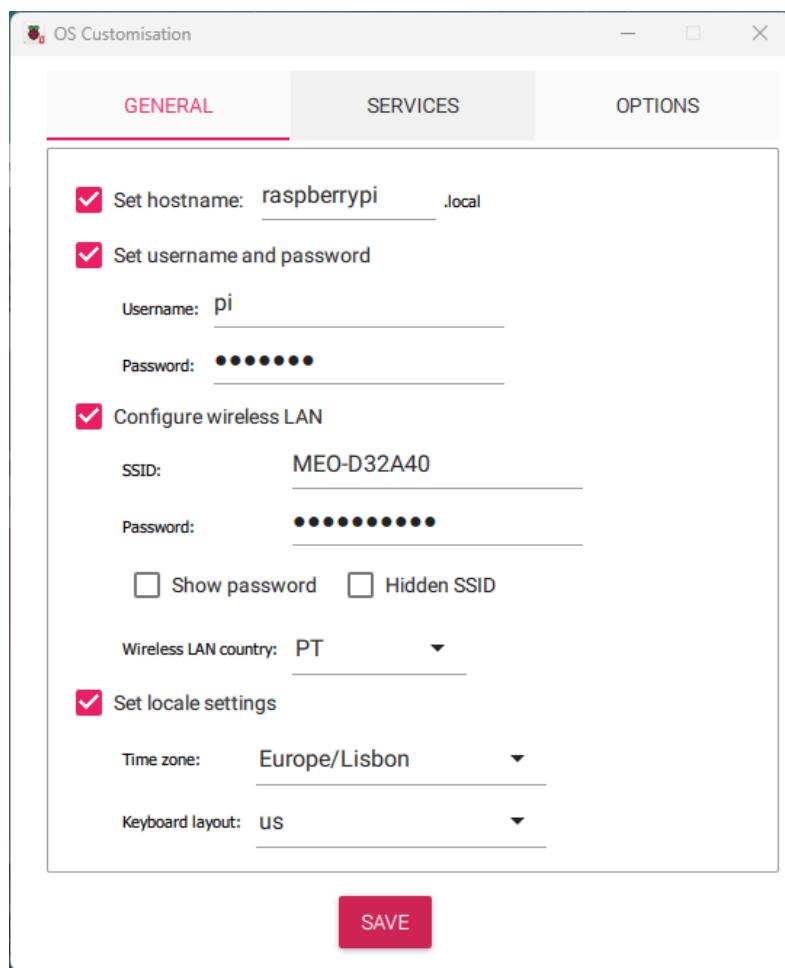


9) Click **Next**. You'll be asked if you would like to apply customisation settings.

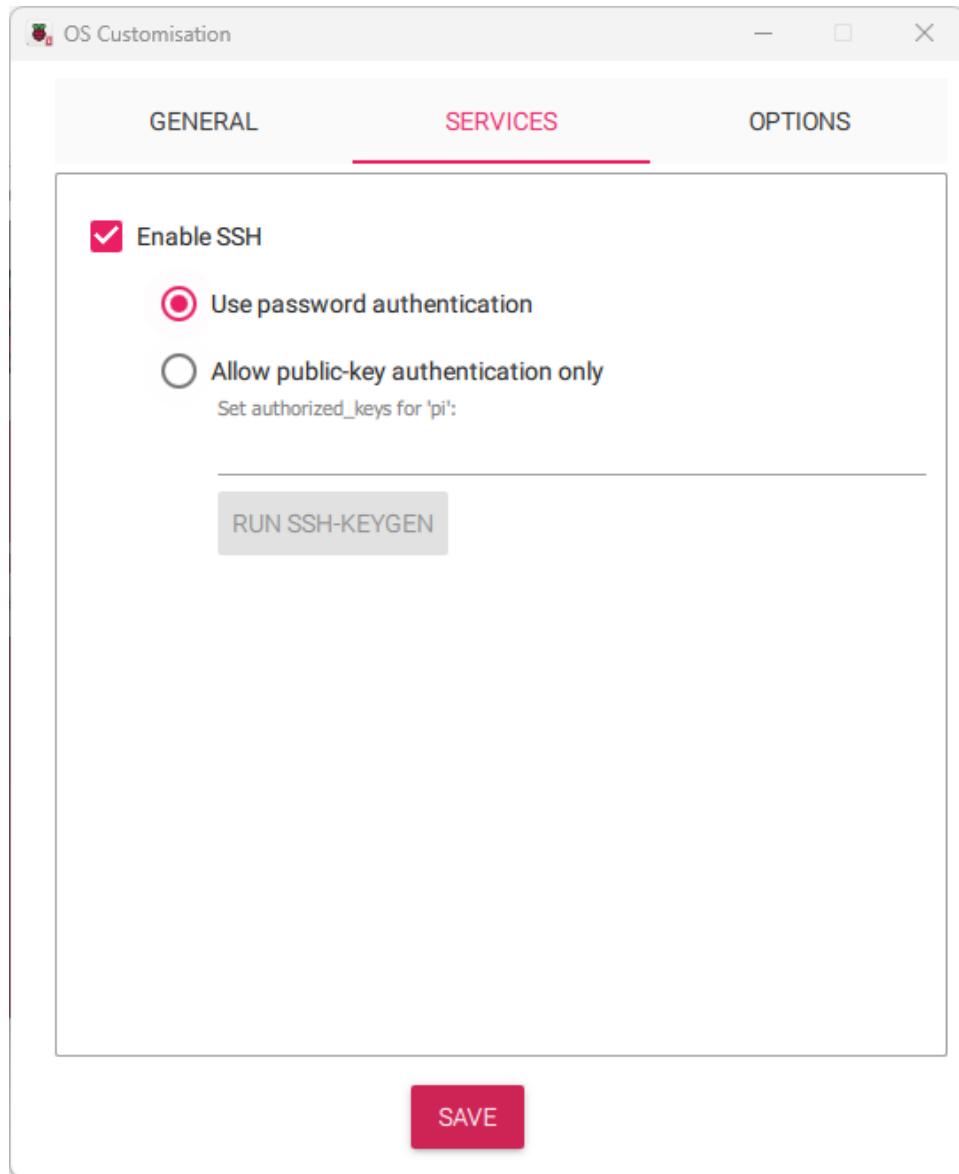
Click on **Edit Settings** to set up the Wi-Fi credentials, and enable SSH.



10) Under the **GENERAL** tab. You can set the hostname (the default will be `raspberrypi`), user and password (memorize the user and password because you'll need it later). Set up Wi-Fi with your local network credentials, and set up your country and time zone. **Don't forget to set your Wireless LAN country!**

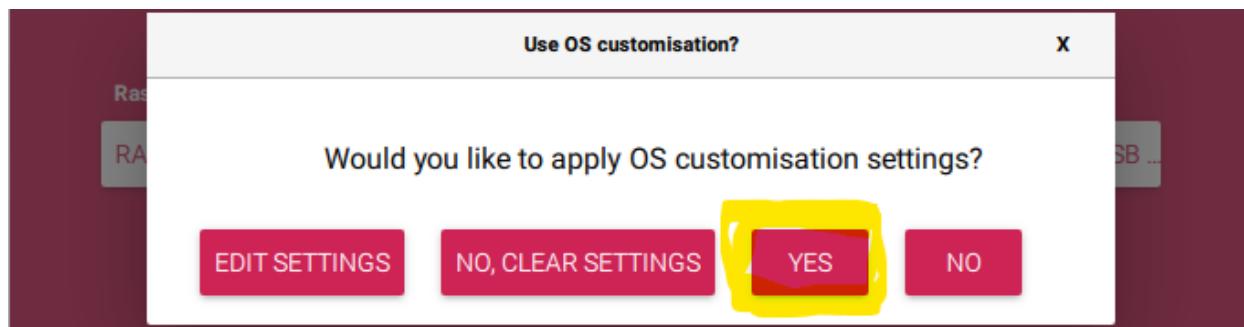


11) Then, click on the **SERVICES** tab and enable SSH with password authentication.

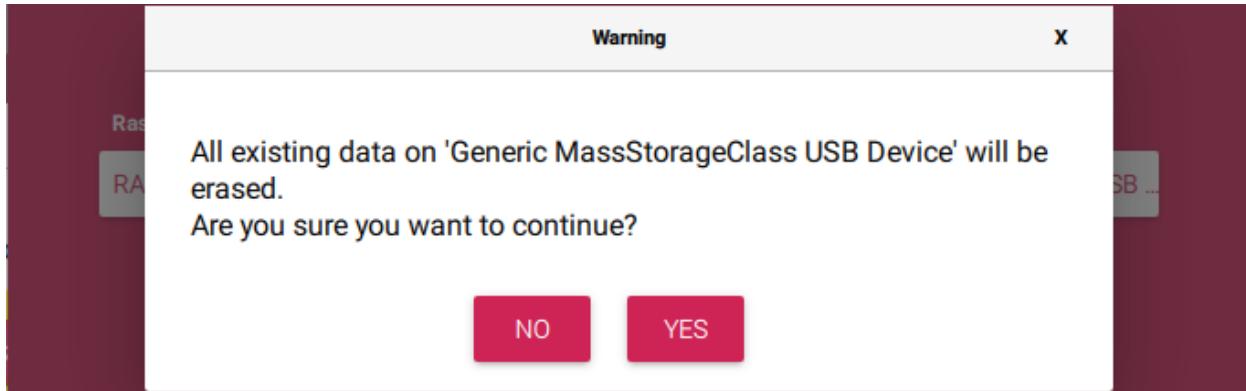


12) Click **Save**. You'll be asked if you want to apply the OS customisation settings.

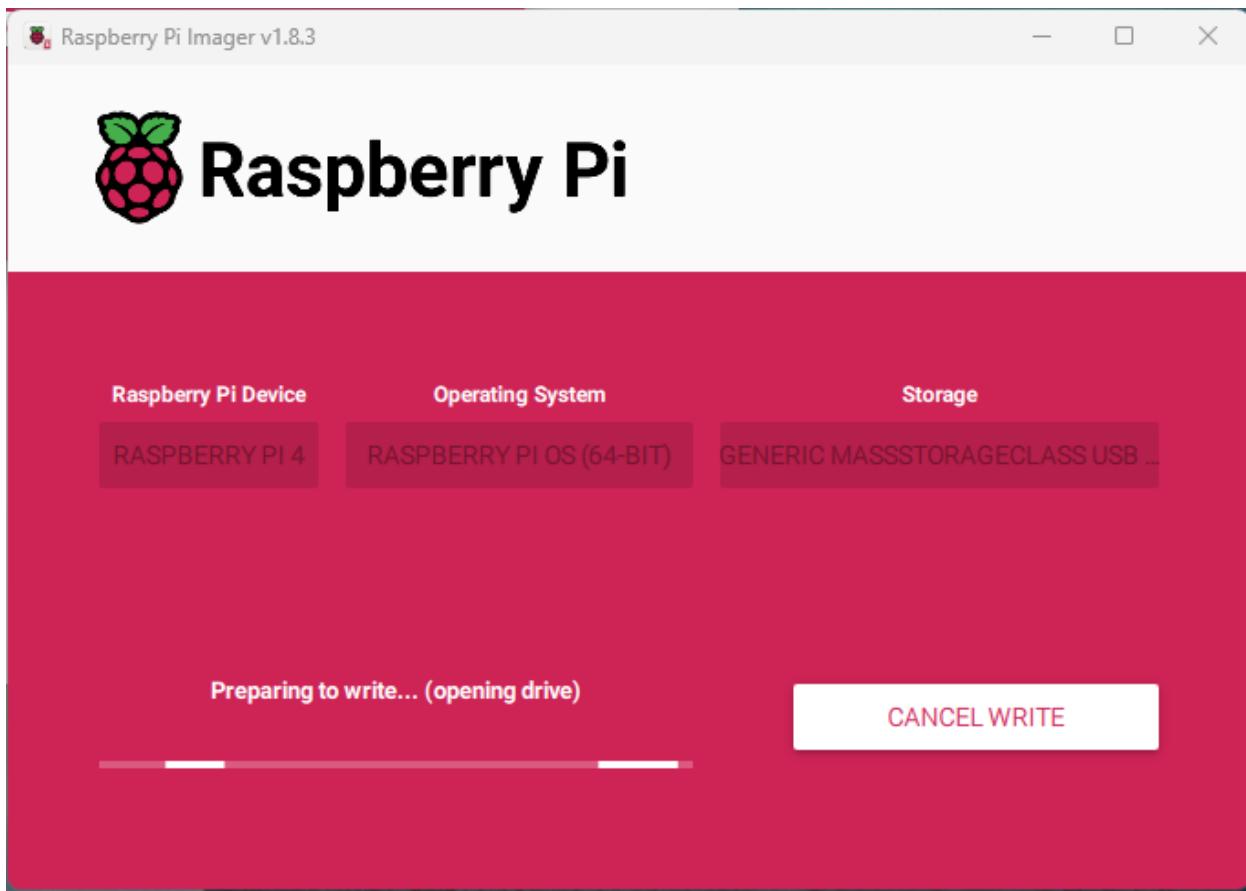
Click **YES**.



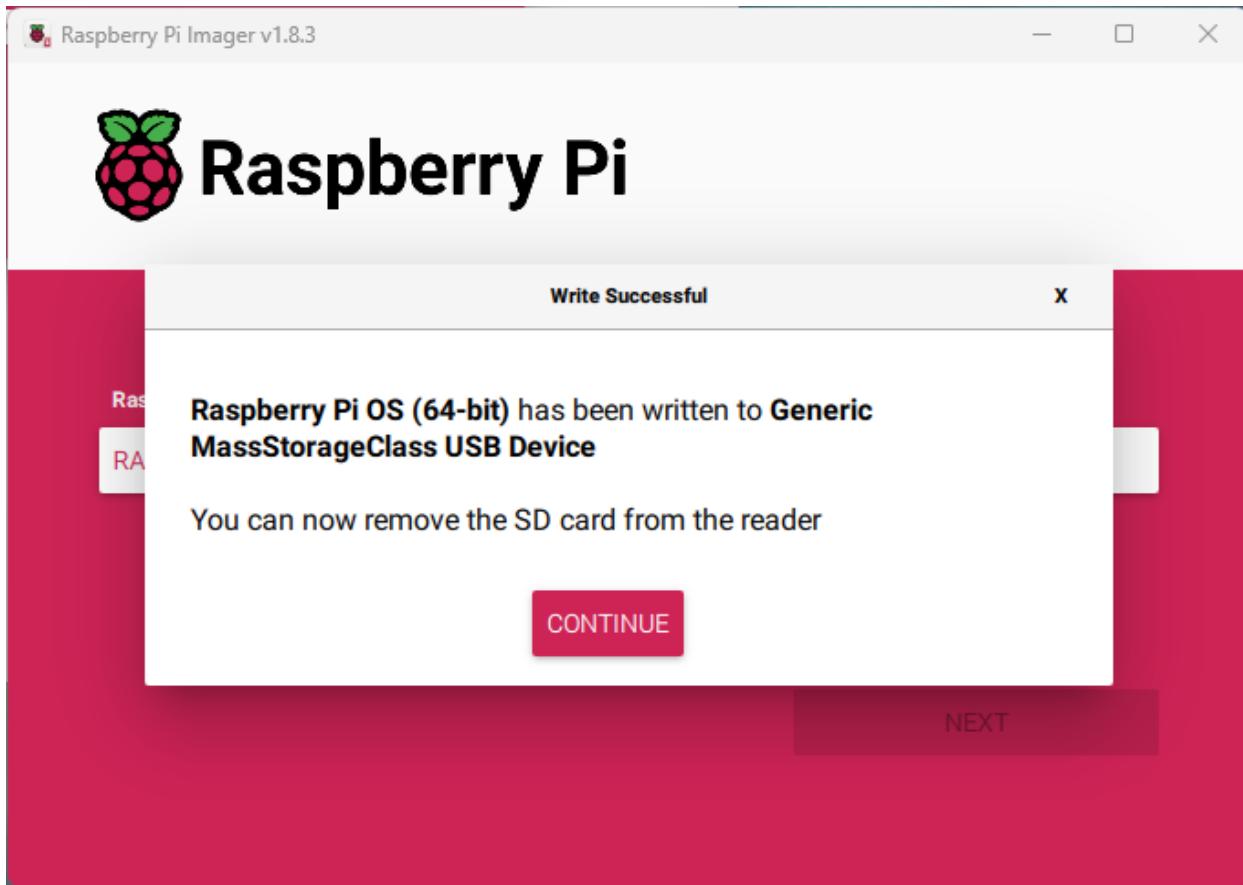
13) Finally, you'll be asked if you want to continue. Click **YES** to start burning the Raspberry Pi OS on the microSD card.



14) Wait a few minutes while it installs the Operating System.



15) When the installation is complete click on **Continue**. It will eject the microSD card safely.



- 16) Now, remove the card from your computer and insert it into your Pi. Then, apply power to the Raspberry Pi to start it and proceed to the next unit.

1.3 - Connecting via SSH to the RPi

For our projects, the Raspberry Pi will run headless (without a monitor, mouse, or keyboard). We'll control the Pi using our computer via SSH.

SSH (which stands for secure shell) is a method of establishing a communication with another computer securely. All data sent via SSH is encrypted. SSH is based on a Unix shell, so it allows you to access your Raspberry Pi files from a remote machine by using terminal commands.

Powering your Raspberry Pi for the first time

With the microSD card inserted, power your Raspberry Pi. If it's your first time powering the Raspberry Pi with a fresh OS installed, it will take some time to setup. After powering your board wait at least 10 minutes before trying to establish an SSH communication.

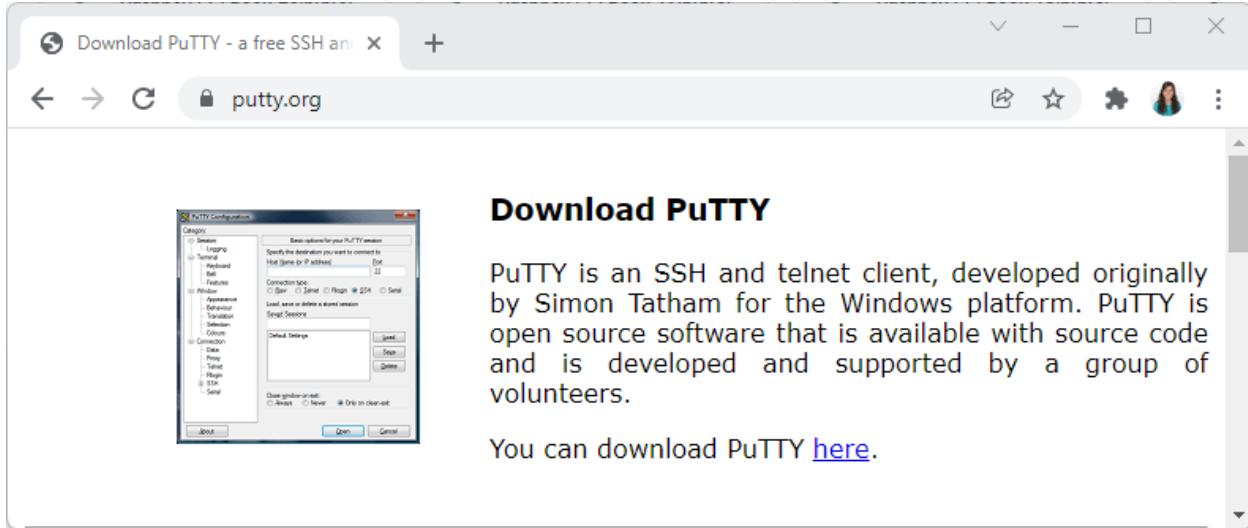
Follow one of the following sections depending on your operating system.

Connecting via SSH - Windows

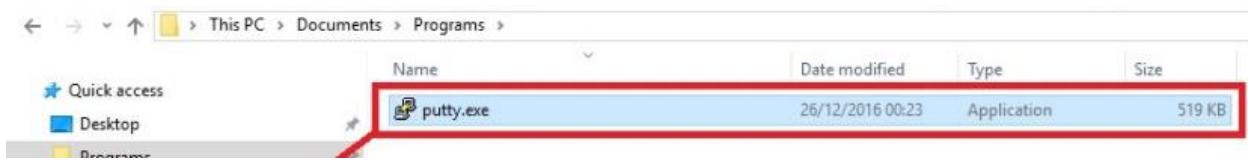
To communicate with the Raspberry Pi via SSH, you need software to handle SSH communication. We'll use **PuTTY**. You need to install it on your computer if you haven't already.

Installing PuTTY

- 1) Open your web browser and go to www.putty.org.
- 2) Download PuTTY. We recommend downloading the *putty.exe* file.



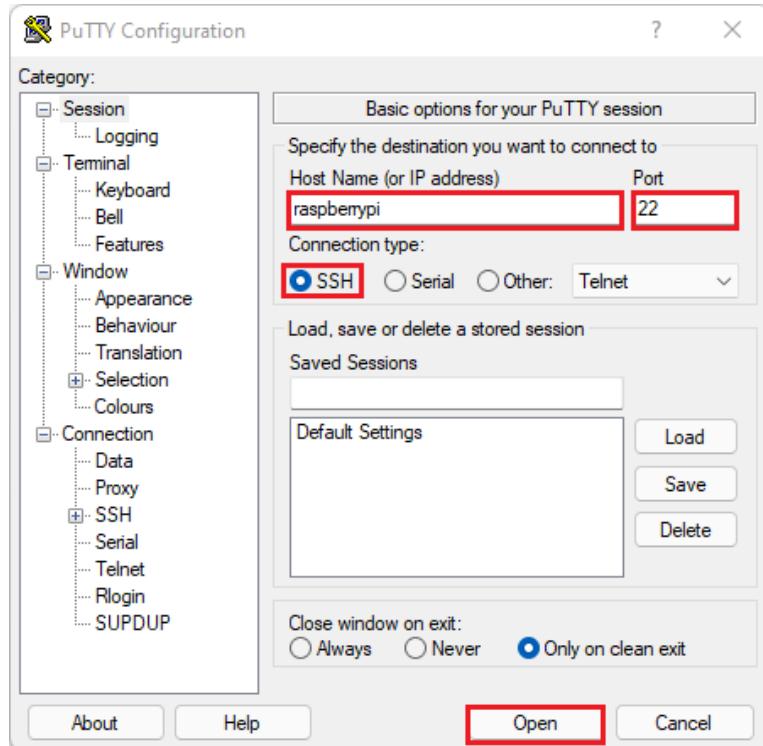
3) Run the *putty.exe* file to execute the software.



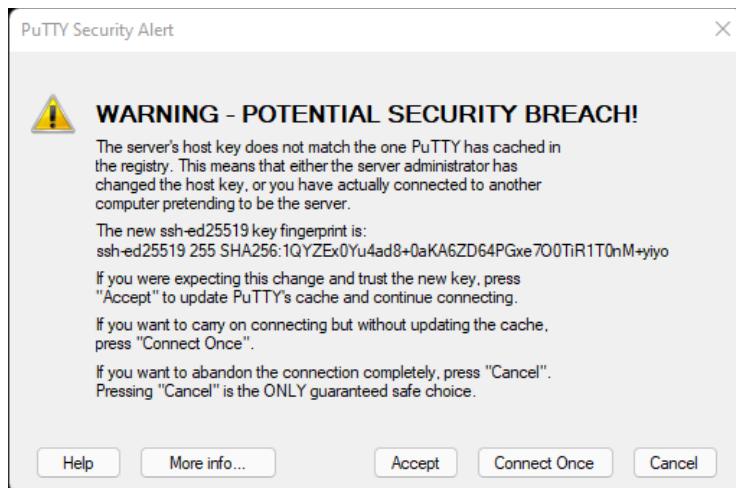
Connecting to the Raspberry Pi via SSH

- 1) With the microSD card inserted and with the Raspberry Pi powered up, open PuTTY on your computer. Select/enter the following options:
 - **Host Name:** raspberrypi¹
 - **Port:** 22
 - **Connection type:** SSH

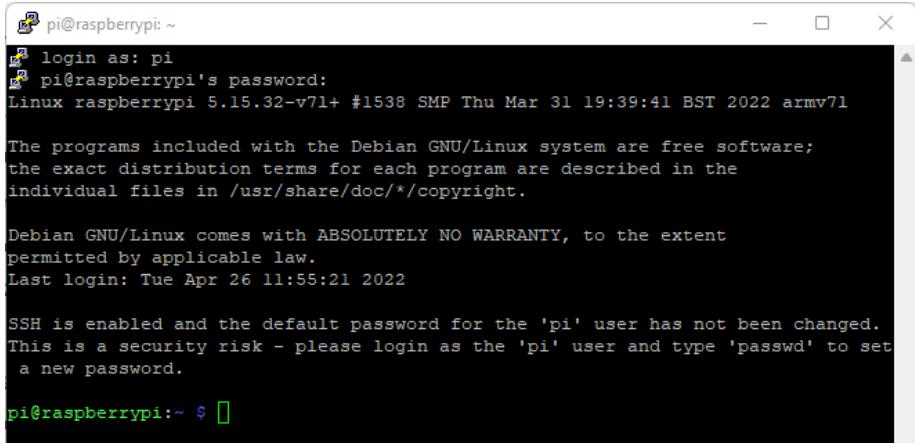
¹ the hostname you set in the Raspberry Pi OS installation process. In our case, we set the hostname to `raspberrypi`



- 2) Click on **Open**. You may get a security warning as follows. If that's the case, click on **Accept**.



- 3) Now, you need to login into your Raspberry Pi using the username and password you set during the installation process.
- 4) On the new window that opens, type your username and hit Enter.
- 5) Then, enter your password and hit Enter. You won't see any characters showing up on the window while you type the password.



A screenshot of a terminal window titled 'pi@raspberrypi: ~'. The window shows a successful SSH login as the 'pi' user. The terminal displays standard Debian system information, including the kernel version (5.15.32-v7l+), the date (Thu Mar 31 19:39:41 BST 2022), and a note about the lack of warranty. It also shows the last login time (Tue Apr 26 11:55:21 2022) and a warning about the default password for the 'pi' user. The prompt 'pi@raspberrypi:~ \$' is visible at the bottom.

Now, you can run Linux commands to interact with your Pi like installing software, running programs, creating folders or files, etc. You can check the Appendix for a list of basic Linux commands.

If PuTTY warns that **the hostname doesn't exist** (and you're sure you've entered the right hostname and that you've set the Wi-Fi credentials correctly), you might need to restart your router so that it assigns an IP address to your Raspberry Pi.

Connecting via SSH - Mac OS X and Linux

In Mac OS X and Linux, you can use the default Terminal window to establish an SSH communication because SSH comes in all Unix-based OSes. Follow these steps:

- 1) Make sure your Raspberry Pi is powered up with the microSD card inserted.
- 2) Open a new Terminal window on your computer.
- 3) Type the following command:

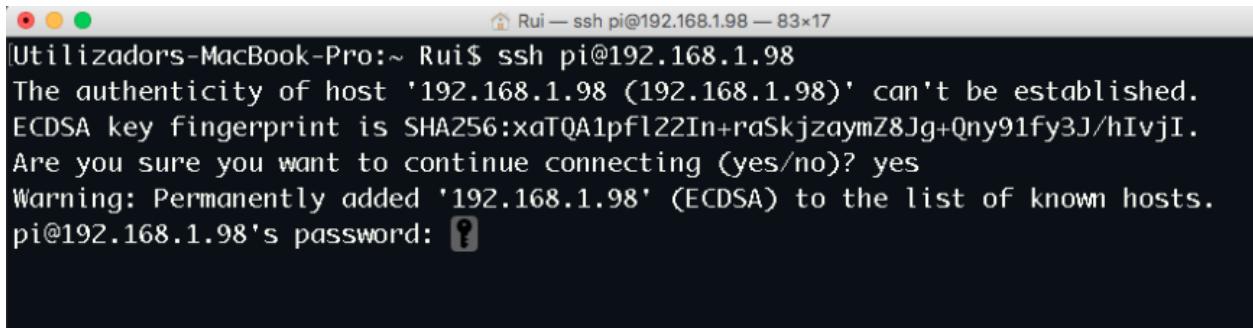
```
sudo ssh pi@raspberrypi
```

or:

```
sudo ssh pi@raspberrypi.local
```

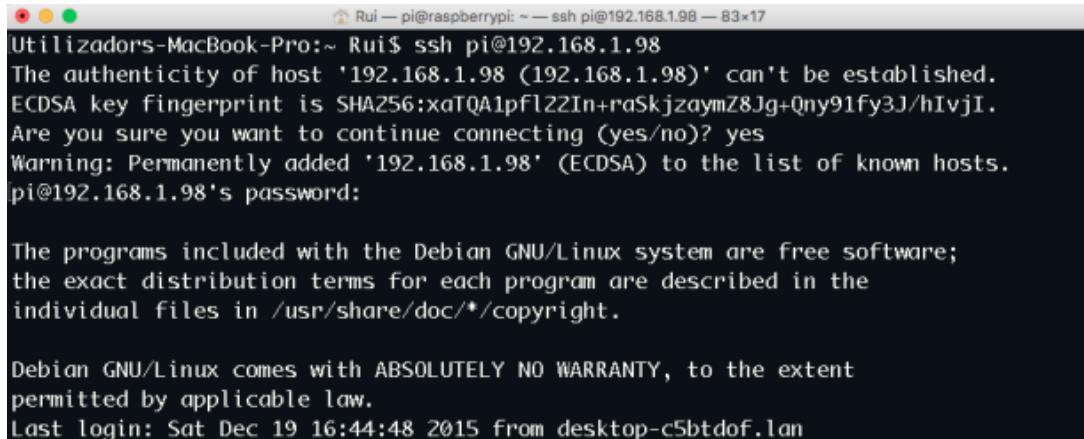
Note: if you defined a different hostname rather than **raspberrypi**, you should use that hostname instead.

- 4) Enter your computer password (so you can run a *sudo* command), and type **yes**.
- 5) When you're asked to type a password for your Raspberry Pi type the password you've set previously for SSH connection, and press **Enter/Return**. Your Terminal window should look like the figure below:



```
Rui — ssh pi@192.168.1.98 — 83x17
Utilizadores-MacBook-Pro:~ Rui$ ssh pi@192.168.1.98
The authenticity of host '192.168.1.98 (192.168.1.98)' can't be established.
ECDSA key fingerprint is SHA256:xaTQA1pf122In+raSkjzaymZ8Jg+Qny91fy3J/hIvjI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.98' (ECDSA) to the list of known hosts.
pi@192.168.1.98's password: 
```

- 6) When you connect your computer to your Raspberry Pi for the first time, you're prompted by a message warning you that you're attempting to establish a connection with an unknown host. Simply click **OK** to proceed.



```
Rui — pi@raspberrypi: ~ — ssh pi@192.168.1.98 — 83x17
Utilizadores-MacBook-Pro:~ Rui$ ssh pi@192.168.1.98
The authenticity of host '192.168.1.98 (192.168.1.98)' can't be established.
ECDSA key fingerprint is SHA256:xaTQA1pf122In+raSkjzaymZ8Jg+Qny91fy3J/hIvjI.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.98' (ECDSA) to the list of known hosts.
pi@192.168.1.98's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Dec 19 16:44:48 2015 from desktop-c5btdof.lan
```

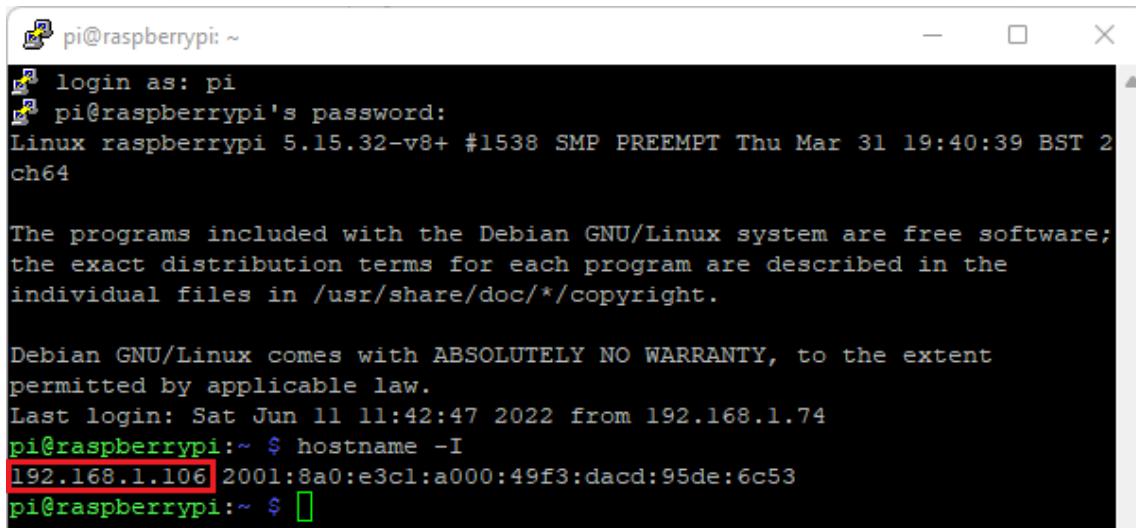
Now, you can run Linux commands to interact with your Pi like installing software, running programs, creating folders or files, etc. You can check the Appendix for a list of basic Linux commands.

Getting the Raspberry Pi IP Address

Knowing your Raspberry Pi IP address will be useful in the future. To get the Pi's IP address, make sure you have an SSH connection established with the Pi. If you've followed the previous section, you should have an SSH connection already established. Run the following command:

```
hostname -I
```

You'll get something as follows with your Raspberry Pi local IP address.



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows the following text:
login as: pi
pi@raspberrypi's password:
Linux raspberrypi 5.15.32-v8+ #1538 SMP PREEMPT Thu Mar 31 19:40:39 BST 2
ch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

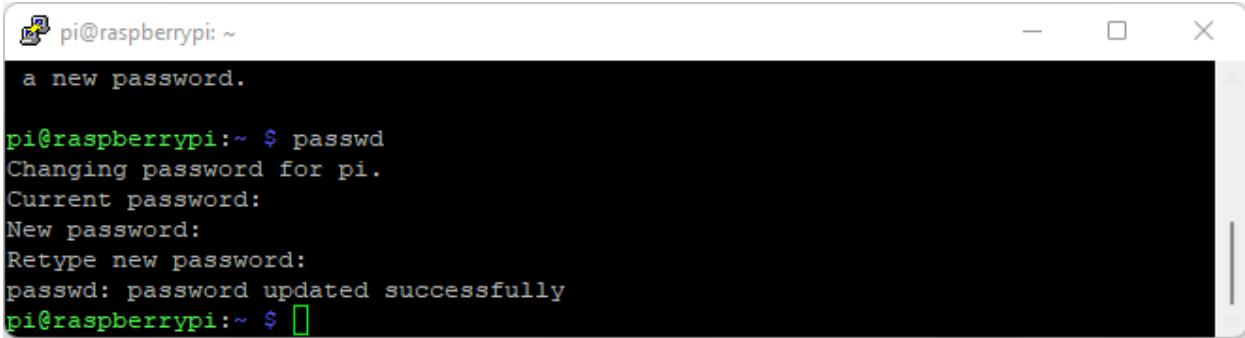
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Jun 11 11:42:47 2022 from 192.168.1.74
pi@raspberrypi:~ \$ hostname -I
192.168.1.106 2001:8a0:e3c1:a000:49f3:daed:95de:6c53
pi@raspberrypi:~ \$

Changing the Password

You can change the password by running the following command:

```
passwd
```

- Insert your current password and hit Enter.
- Type the new password.
- Retype the new password.
- Password was updated successfully.



```
pi@raspberrypi: ~
a new password.

pi@raspberrypi:~ $ passwd
Changing password for pi..
Current password:
New password:
Retype new password:
passwd: password updated successfully
pi@raspberrypi:~ $
```

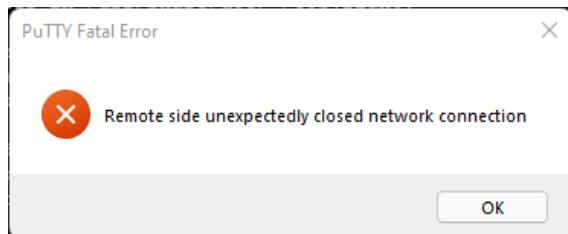
The next time you login into your Pi, you'll use the new password.

Shutting Down

To shut down your Raspberry Pi, simply type this on the command line:

```
sudo poweroff
```

The SSH connection will be shut down right after.



To power up your Raspberry Pi again, you need to remove and apply power again.

Some Useful Information About Using PuTTY

If you've never used PuTTY before, here is some information that might be useful when you get started.

Copy/Paste with PuTTY

To copy and paste things with PuTTY you can't use CTRL+C and CTRL+V.

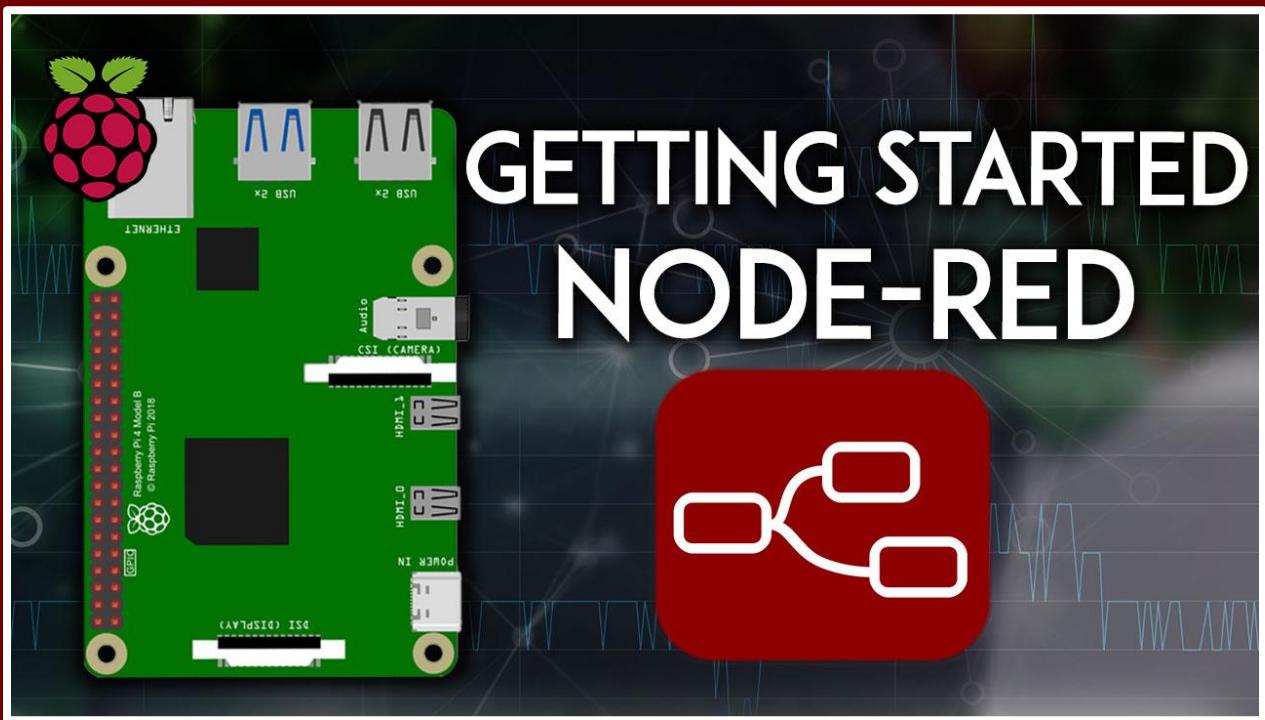
To copy something from PuTTY, simply click and drag over the part of the PuTTY screen you want to copy, then release the mouse button. **The area you highlighted is on the clipboard and you can paste it wherever you want.**

To paste something to PuTTY: on the prompt window, **press the mouse right button**. Whatever you have copied to the clipboard will be pasted into the PuTTY window.

If you want to dive deeper into how PuTTY works, you can check the [PuTTY user manual here](#).

MODULE 2

Getting Started with Node-RED



In this Module, you'll get started with Node-RED software on the Raspberry Pi. You'll install Node-RED, learn some basic concepts, and create simple flows to control the Raspberry Pi GPIOs. You'll also learn about Node-RED dashboard, a set of nodes to easily create a user interface.

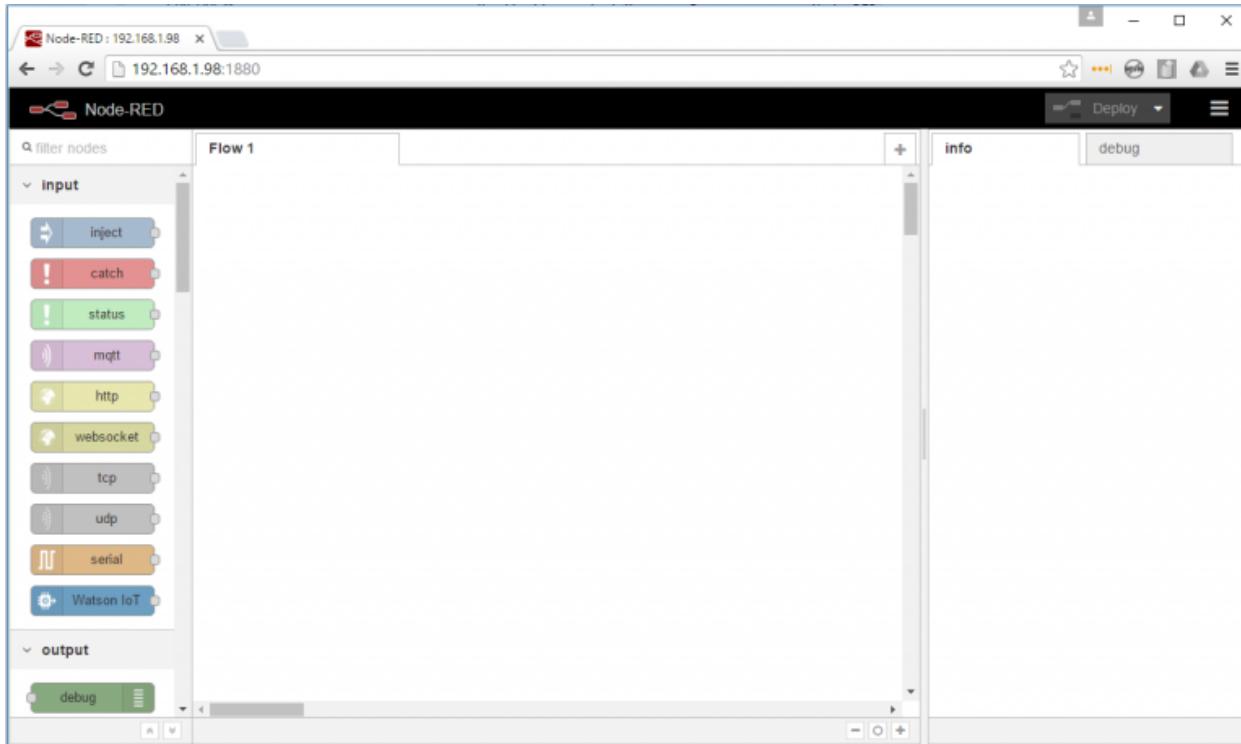
2.1 - Node-RED Introduction

In this unit, you'll get started with Node-RED on the Raspberry Pi. Node-RED is a powerful visual programming open-source tool to build Internet of Things (IoT) applications. In this tutorial, we'll cover what is Node-RED, how to install it, and 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.

Node-RED runs on the web browser and uses 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 is Node-RED a great solution?

- Node-RED is open source and developed by IBM.
- The Raspberry Pi runs Node-RED perfectly.
- It is a visual programming tool, which makes it more accessible to a wider range of users.
- With Node-RED you can spend more time making cool stuff, rather than spending countless hours writing code.
- There are many Node-RED nodes and examples for almost anything you may want to build in your home automation system.
- Node-RED has a big community so you can always ask for help on Node-RED dedicated forums.

What can you do with Node-RED?

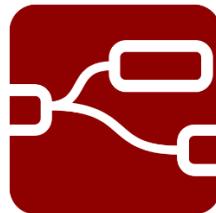
Node-RED makes it easy to:

- Control your RPi GPIOs;
- Establish an MQTT connection with other devices (ESP32, ESP8266, etc.);
- Create a responsive graphical user interface for your projects;
- Communicate with third-party services (IFTTT.com, Adafruit.io, Thing Speak, Home Assistant, InfluxDB, etc.);
- Retrieve data from the web (weather forecast, stock prices, emails, etc.);
- Create time-triggered events;
- Store and retrieve data from a database;
- And much more.

Here's a repository [with some examples of flows and nodes](#) for Node-RED.

2.2 - Installing Node-RED

In this section, you'll install Node-RED on your Raspberry Pi. Getting Node-RED installed on your Raspberry Pi is quick and easy. It just takes a few commands.



Install Node-RED

Having an SSH connection established with your Raspberry Pi, follow the next steps to install Node-RED.

First, update and upgrade your system with the following command:

```
sudo apt update && sudo apt upgrade
```

At some point, you'll be asked if you want to proceed. Click **Y** and then **Enter**. This process may take a few minutes.

When it's completed, enter the following command to install Node-RED:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

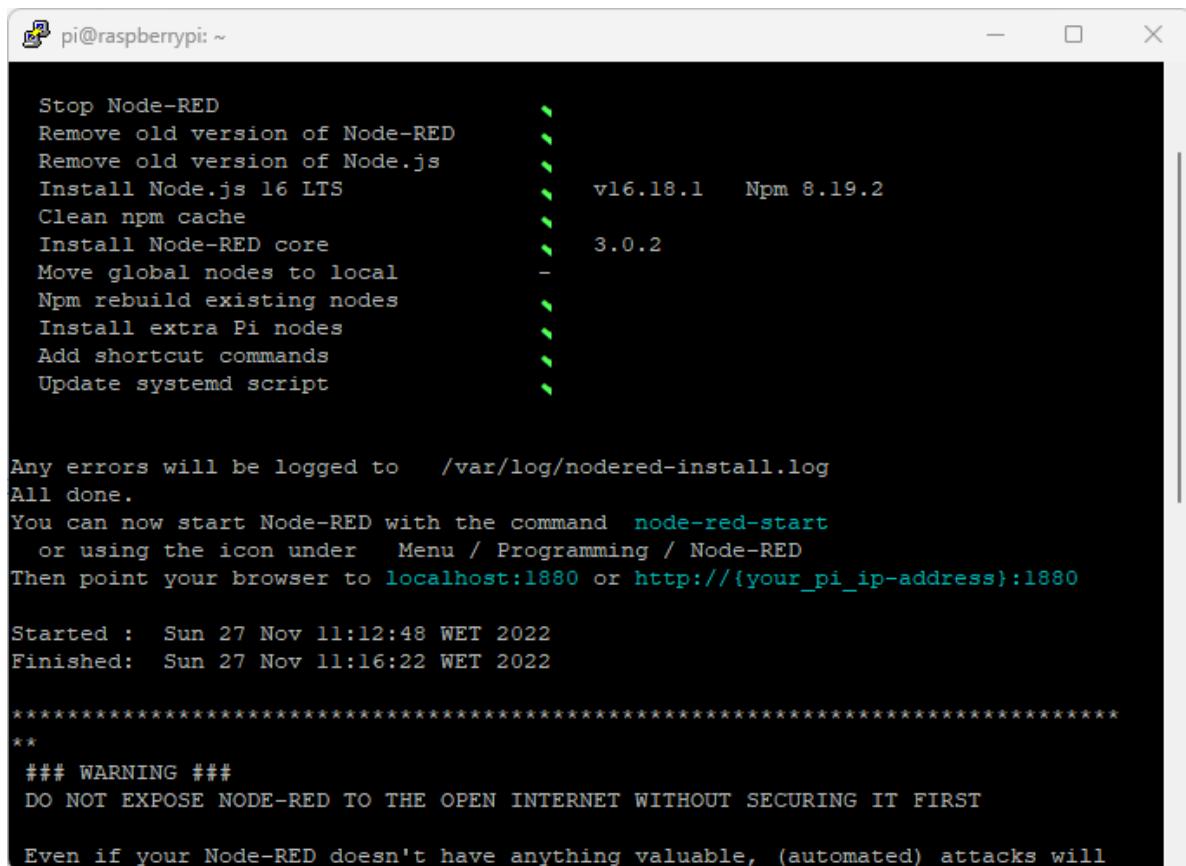
To paste a command on the Terminal window, simply click the mouse right button. **It won't work with CTRL+V**. Then, press Enter to run the command.

If you get an error message like "*Curl: no URL specified*" or a similar error message, it means the command was not copied properly. If that's the case, we recommend going to the Node-RED installation instructions page and copy the command from there. Here's the link for the official installation instructions page:

- <https://nodered.org/docs/getting-started/raspberrypi>

After running the command, you'll be asked: "Would you like to install Pi-specific nodes?" Press **Y** and **Enter**.

It will take a few minutes to install Node-RED. In the end, you should get a similar message on the Terminal window (see next page).



```

pi@raspberrypi: ~
Stop Node-RED
Remove old version of Node-RED
Remove old version of Node.js
Install Node.js 16 LTS
Clean npm cache
Install Node-RED core
Move global nodes to local
Npm rebuild existing nodes
Install extra Pi nodes
Add shortcut commands
Update systemd script

Any errors will be logged to /var/log/nodered-install.log
All done.
You can now start Node-RED with the command node-red-start
or using the icon under Menu / Programming / Node-RED
Then point your browser to localhost:1880 or http://{your_pi_ip-address}:1880

Started : Sun 27 Nov 11:12:48 WET 2022
Finished: Sun 27 Nov 11:16:22 WET 2022

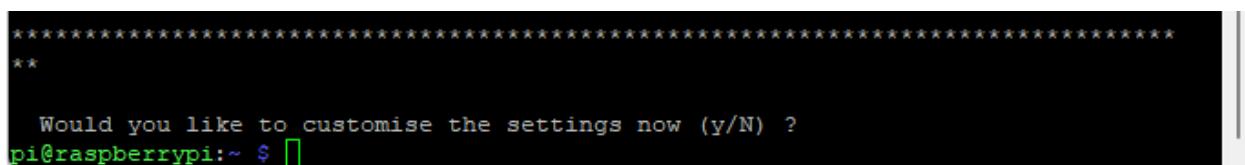
*****
**
### WARNING ***
DO NOT EXPOSE NODE-RED TO THE OPEN INTERNET WITHOUT SECURING IT FIRST

Even if your Node-RED doesn't have anything valuable, (automated) attacks will

```

Configure Node-RED Settings

You'll be asked if you want to customize the settings now. Press **Y** and then press **Enter** (in some versions this step is omitted and it will skip to the screenshot on the next page).



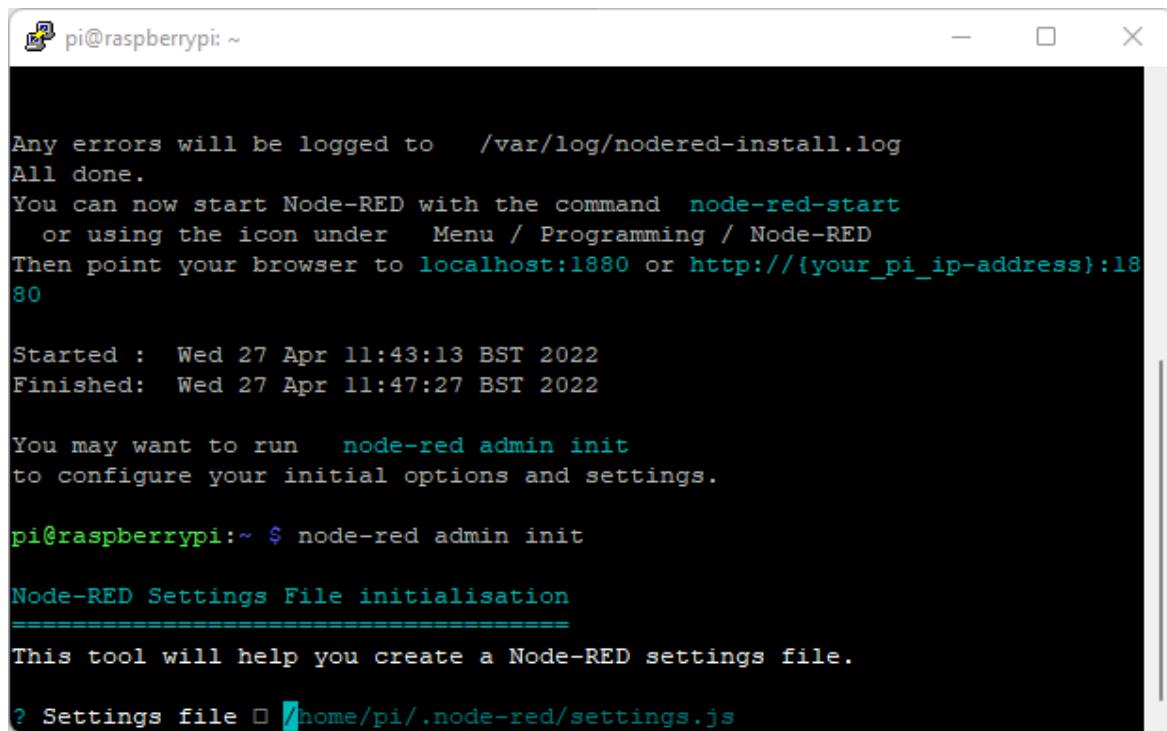
```

*****
**
Would you like to customise the settings now (y/N) ?
pi@raspberrypi:~ $ 

```

If you get an error, run the following command to start customizing the settings:

```
node-red admin init
```



A terminal window titled "pi@raspberrypi: ~" showing the output of the "node-red admin init" command. The output includes instructions for logging errors, starting Node-RED, and pointing a browser to localhost:1880 or a specific IP address. It also shows the start and finish times of the process, and a note about running "node-red admin init" again to configure initial options and settings. The terminal prompt "pi@raspberrypi:~ \$" appears at the bottom, followed by the "Node-RED Settings File initialisation" command.

```
Any errors will be logged to /var/log/nodered-install.log
All done.
You can now start Node-RED with the command node-red-start
or using the icon under Menu / Programming / Node-RED
Then point your browser to localhost:1880 or http://{your_pi_ip-address}:18
80

Started : Wed 27 Apr 11:43:13 BST 2022
Finished: Wed 27 Apr 11:47:27 BST 2022

You may want to run node-red admin init
to configure your initial options and settings.

pi@raspberrypi:~ $ node-red admin init

Node-RED Settings File initialisation
=====
This tool will help you create a Node-RED settings file.

? Settings file □ /home/pi/.node-red/settings.js
```

- Press **Enter** to create a Node-RED Settings file on `/home/pi/.node-red/settings.js` (keep the default `settings.js` file location)
- Do you want to set up user security? **Yes**.
- Enter a username and press **Enter** ([you need to remember it later](#)).
- Enter a password and press **Enter** ([you need to remember it later](#)).
- Then, you need to define user permissions. We'll set full access, make sure the **full access** option is highlighted in blue and press **Enter**.
- You can add other users with different permissions if you want. We'll just create one user for now. You can always add other users later.
- Do you want to enable the Projects feature? **No**.
- Enter a name for your flows file. Press **Enter** to select the default name `flows.json`.

- Provide a passphrase to encrypt your credentials file. Learn more about what is a [passphrase](#).
- Select a theme for the editor. Simply press **Enter** to select default.
- Press **Enter** again to select the default text editor.
- Allow Function nodes to load external modules? **Yes**.

Node-RED configuration was successful. All settings are saved on `settings.js`.

Start Node-RED

Run the following command to start Node-RED:

```
node-red-start
```

You should get a similar message in the Terminal window:

```
pi@raspberrypi: Node-RED console
Settings file written to /home/pi/.node-red/settings.js
pi@raspberrypi:~ $ node-red-start

Start Node-RED

Once Node-RED has started, point a browser at http://192.168.1.106:1880
On Pi Node-RED works better with the Firefox or Chrome browser

Use  node-red-stop                      to stop Node-RED
Use  node-red-start                     to start Node-RED again
Use  node-red-log                       to view the recent log output
Use  sudo systemctl enable nodered.service to autostart Node-RED at every boot
Use  sudo systemctl disable nodered.service to disable autostart on boot

To find more nodes and example flows - go to http://flows.nodered.org

Starting as a systemd service.
27 Nov 11:23:45 - [info]
Welcome to Node-RED
=====
27 Nov 11:23:45 - [info] Node-RED version: v3.0.2
27 Nov 11:23:45 - [info] Node.js version: v16.18.1
27 Nov 11:23:45 - [info] Linux 5.15.61-v8+ arm64 LE
27 Nov 11:23:46 - [info] Loading palette nodes
27 Nov 11:23:49 - [info] Settings file : /home/pi/.node-red/settings.js
27 Nov 11:23:49 - [info] Context store : 'default' [module=memory]
27 Nov 11:23:49 - [info] User directory : /home/pi/.node-red
27 Nov 11:23:49 - [warn] Projects disabled : editorTheme.projects.enabled=false
27 Nov 11:23:49 - [info] Flows file : /home/pi/.node-red/flows.json
27 Nov 11:23:49 - [info] Creating new flow file
27 Nov 11:23:49 - [warn] Encrypted credentials not found
27 Nov 11:23:49 - [info] Server now running at http://127.0.0.1:1880/
27 Nov 11:23:49 - [info] Starting flows
27 Nov 11:23:49 - [info] Started flows
```

Autostart Node-RED on boot

To automatically run Node-RED when the Pi boots up, you need to enter the following command (press **CTRL + C** to stop the previous task before entering the new command).

```
sudo systemctl enable nodered.service
```

This means that as long as your Raspberry Pi is powered up, Node-RED will be running.

Now, restart your Pi so the autostart takes effect. The next time the Raspberry Pi restarts, Node-RED will be already running.

```
sudo reboot
```

Note: If, later on, you want to disable autostart on boot, you can run:
`sudo systemctl disable nodered.service`

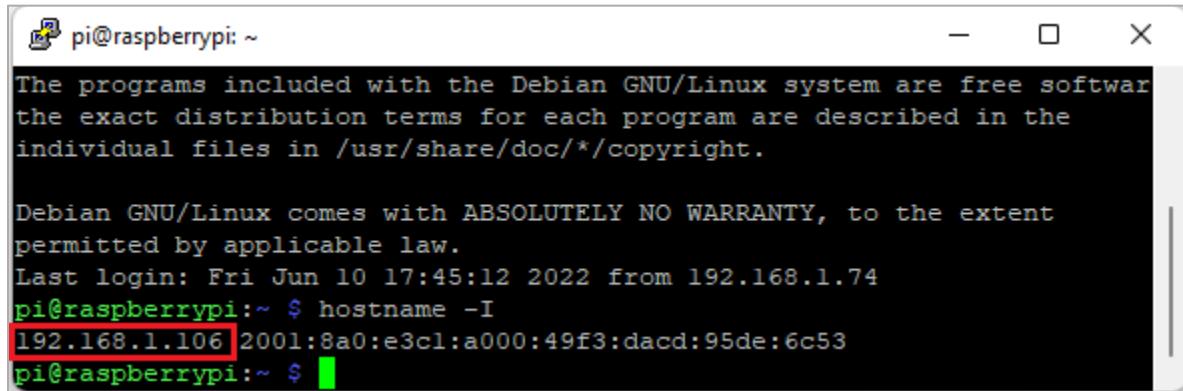
For more information about the installation process, check the [official documentation](#).

Access Node-RED

Node-RED runs on port 1880. To access Node-RED open a browser on your computer and type the Raspberry Pi IP address followed by :1880. For example, in my case:

```
192.168.1.106:1880
```

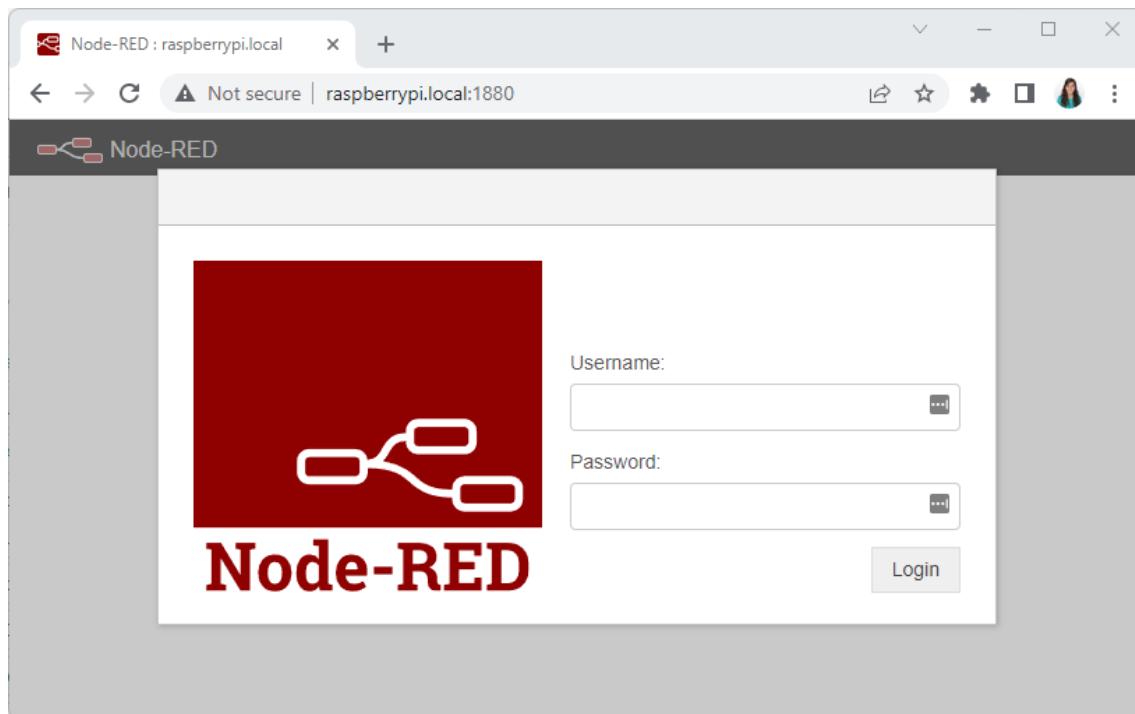
Remember: to get your Raspberry Pi IP address, you can run the following command: `hostname -I`



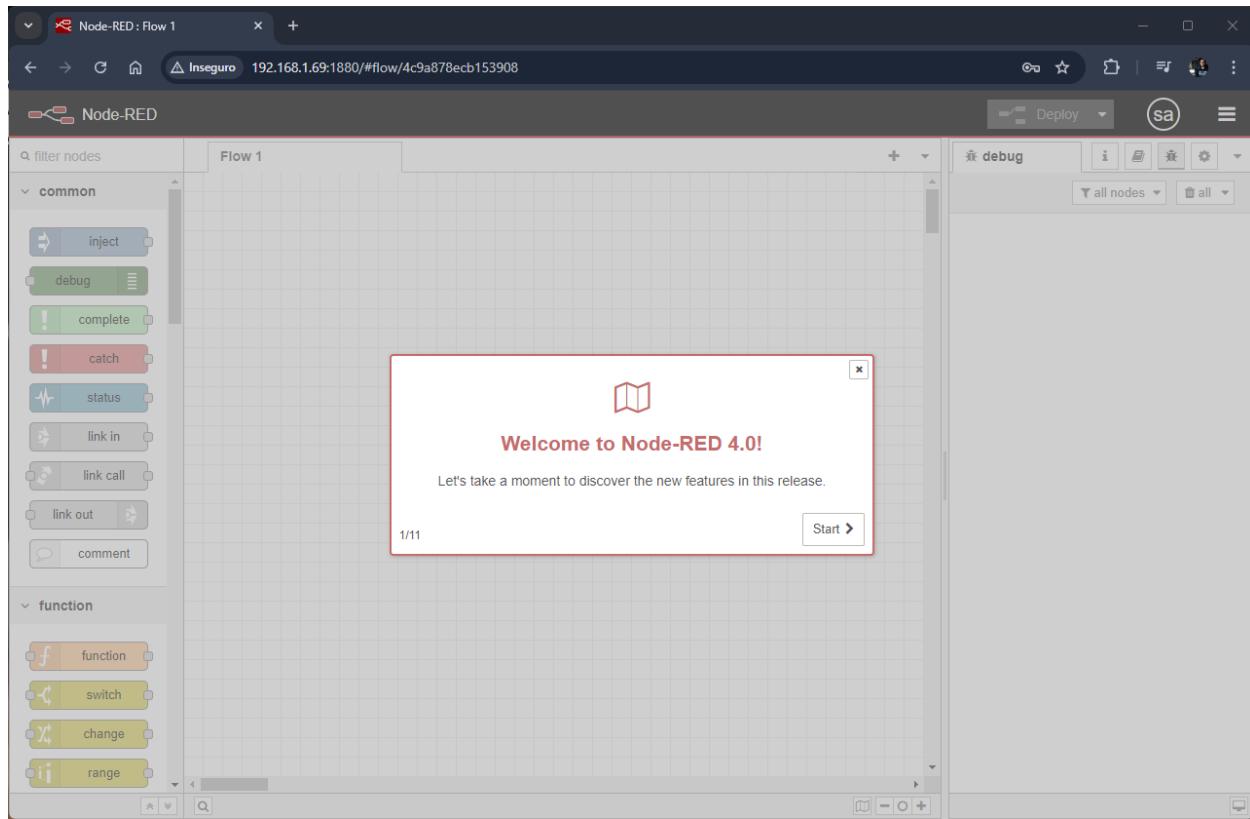
```
pi@raspberrypi: ~
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jun 10 17:45:12 2022 from 192.168.1.74
pi@raspberrypi:~ $ hostname -I
192.168.1.106 2001:8a0:e3c1:a000:49f3:dacd:95de:6c53
pi@raspberrypi:~ $
```

After entering the Raspberry Pi IP address followed by :1880 on the web browser, the Node-RED login page should load. Insert your Node-RED username and password that you've defined when configuring the Node-RED settings.



Now, you have access to Node-RED. You can start building your flows.

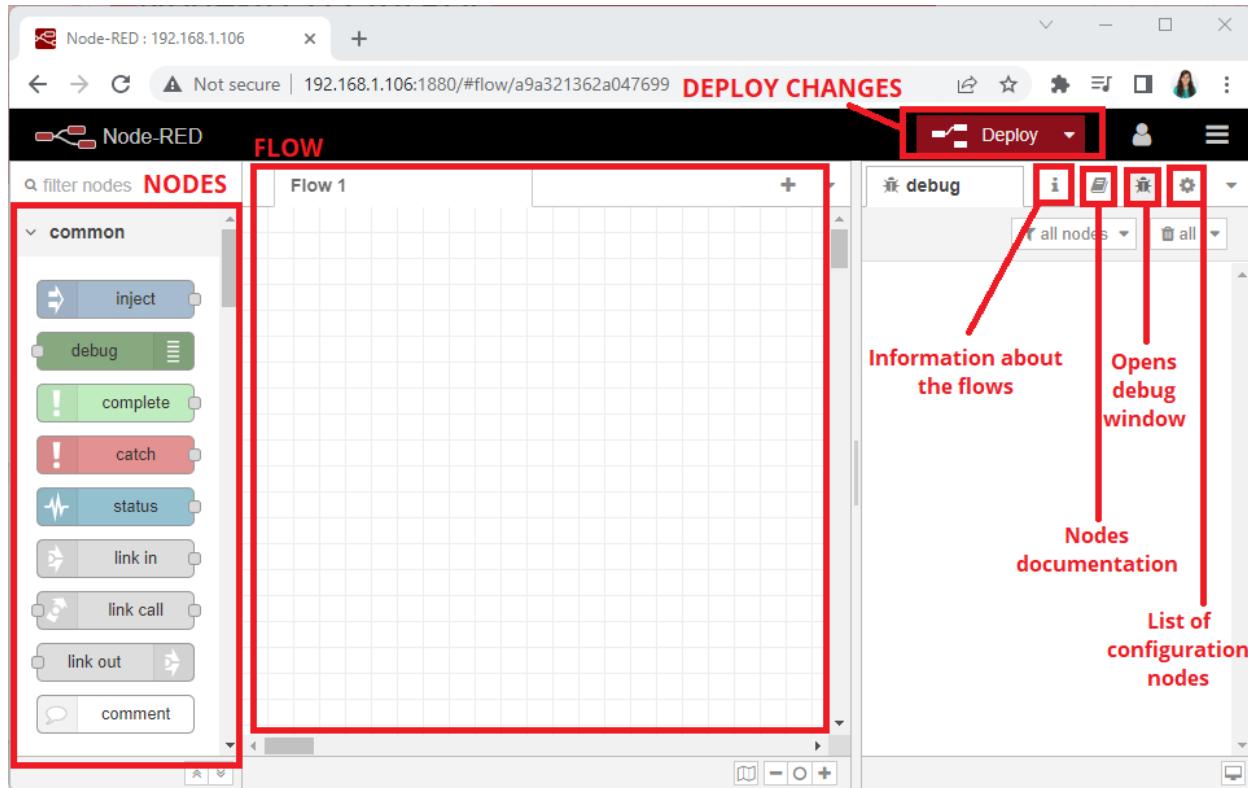


2.3 - Node-RED Overview

In this unit, we'll go through a quick overview of Node-RED. Let's start by taking a look at the Node-RED visual interface.

Node-RED Interface Main Sections

The following picture shows the Node-RED main sections labeled.



Nodes

On the left sidebar, 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 **nodes documentation** tab.

Nodes have input and/or output ports to receive and send information to other nodes. For example, a node receives an input from a previous node, processes that

information, and outputs a different message to another node that will do something with that information. The information passed between nodes is called a **message**.

Flow

The **nodes** are the building blocks of a **flow**. You wire nodes together to create a **flow** that will perform a certain task. A **Flow** is also a tab in the workspace where you place and organize the nodes.

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

Right Sidebar

The right sidebar presents several tools.

Information: shows information about the flows;

Help: shows the nodes' documentation;

Debug: the bug icon opens a debugging window that shows messages passed to debug nodes—it's useful for debugging purposes;

Config nodes: the gear icon shows information about configuration nodes. Configuration nodes do not appear on the main workspace, and they are special nodes that hold reusable configurations that can be shared by several nodes in a flow like MQTT broker settings.

Deploy: the deploy button saves all changes made to the flow and starts running the flow.

Creating a Simple Flow

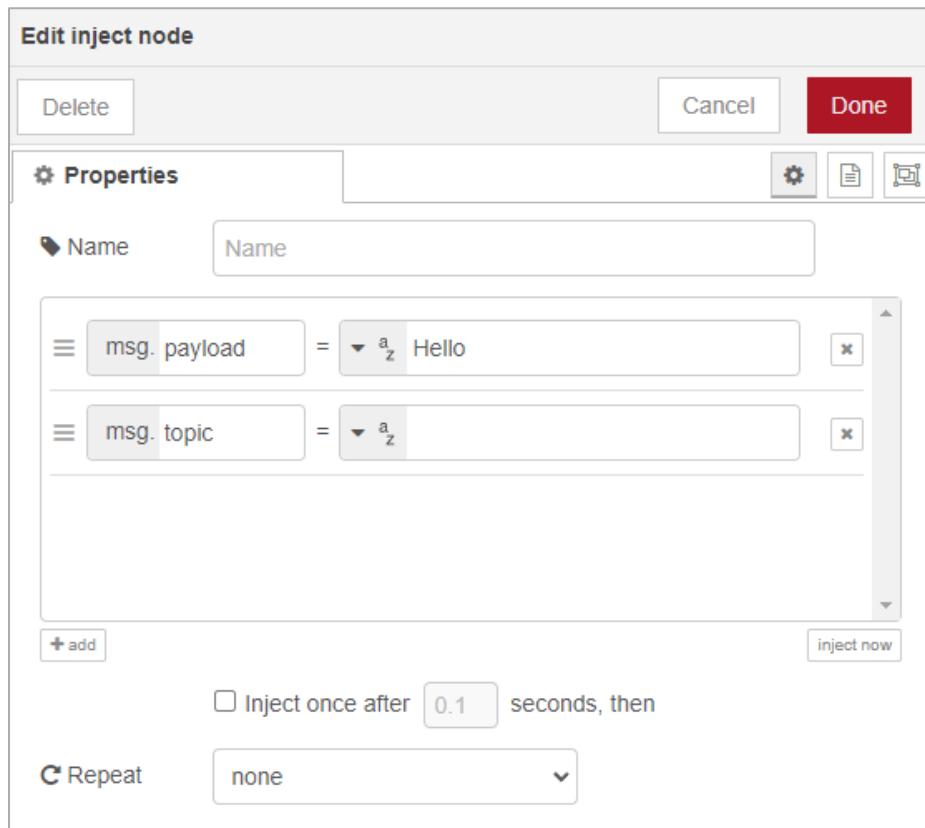
To get you used to the Node-RED interface, let's create a simple flow. The flow we'll create, simply prints a message to the debug console, when triggered.

Drag an **inject** node and a **debug** node to your flow and wire them together. When dragged to the flow, the inject node will change its name to **timestamp** and the debug node to **msg.payload** or **debug 1** (depending on the Node-RED version) by default.



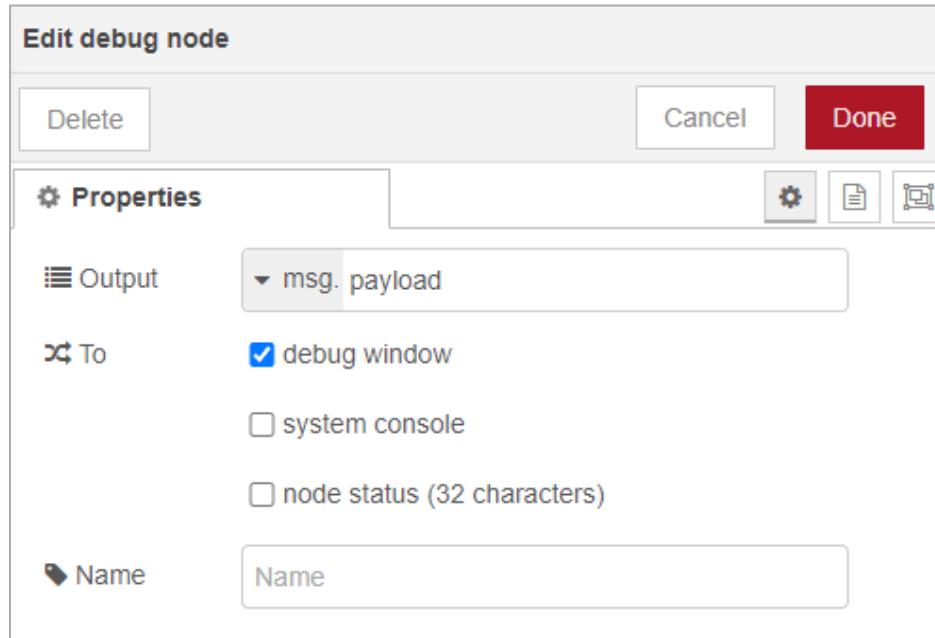
Now, let's edit the **inject** node. Double-click the node. In the figure below, you can see the different settings you can change.

On the **msg.payload** field, select **string** and type **Hello**. Then, click **Done**.



Messages (`msg`) in Node-RED are JavaScript objects that can have multiple properties. The **payload** is the default property most nodes work with. You can think of it as the main content of the message you want to send to the next node. In our case, we're simply sending a text message. We'll take a closer look at the **message** object soon.

We won't edit the **debug** node for now, but you can double-click on it to check its properties.



You can select the output of the debug node, which is `msg.payload`, and where we want to send that output. In our case, we want to send it to the debug window.

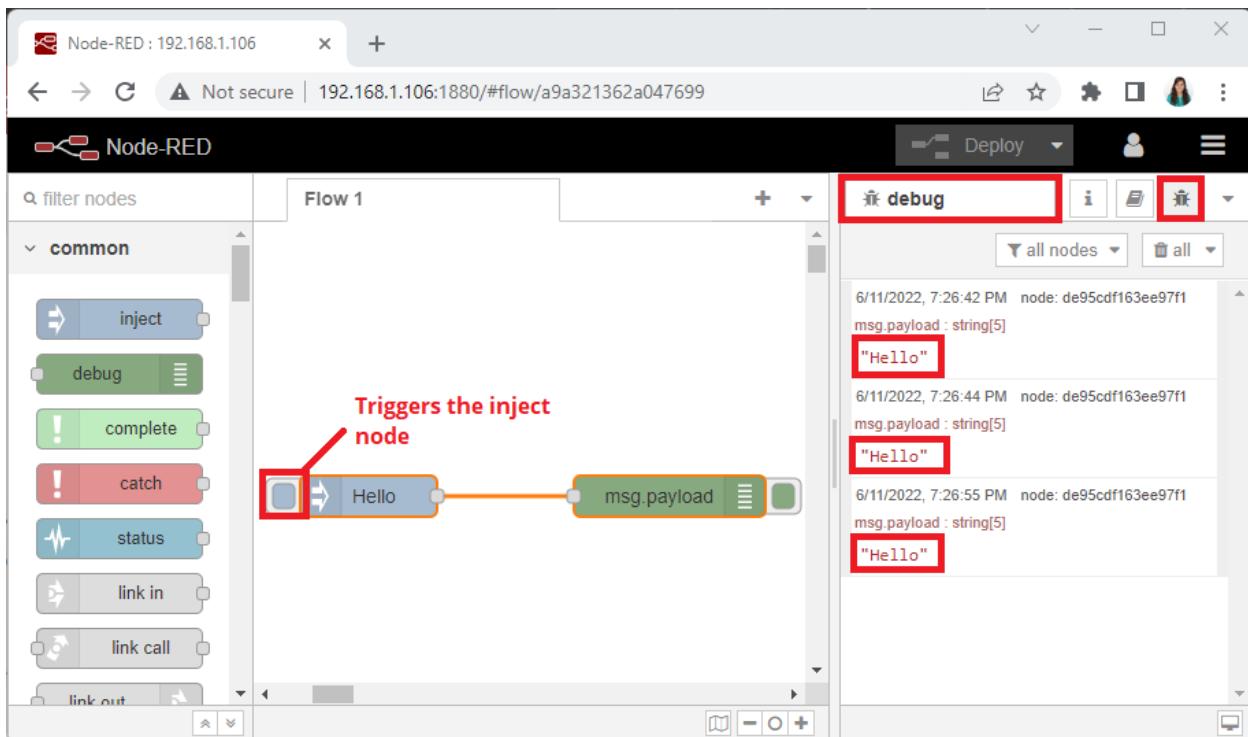
To save your application, you need to click the **Deploy** button in the top right corner.



Your application is saved.

Testing the flow

Let's test our simple flow. Open the debug window. Then, click the inject node to trigger the flow.



As you can see, our message is printed in the debug window when you trigger the inject node. This is a very basic example and it doesn't do anything useful. However, the purpose of this unit is to get you familiar with the Node-RED interface.

Node-RED Message Object

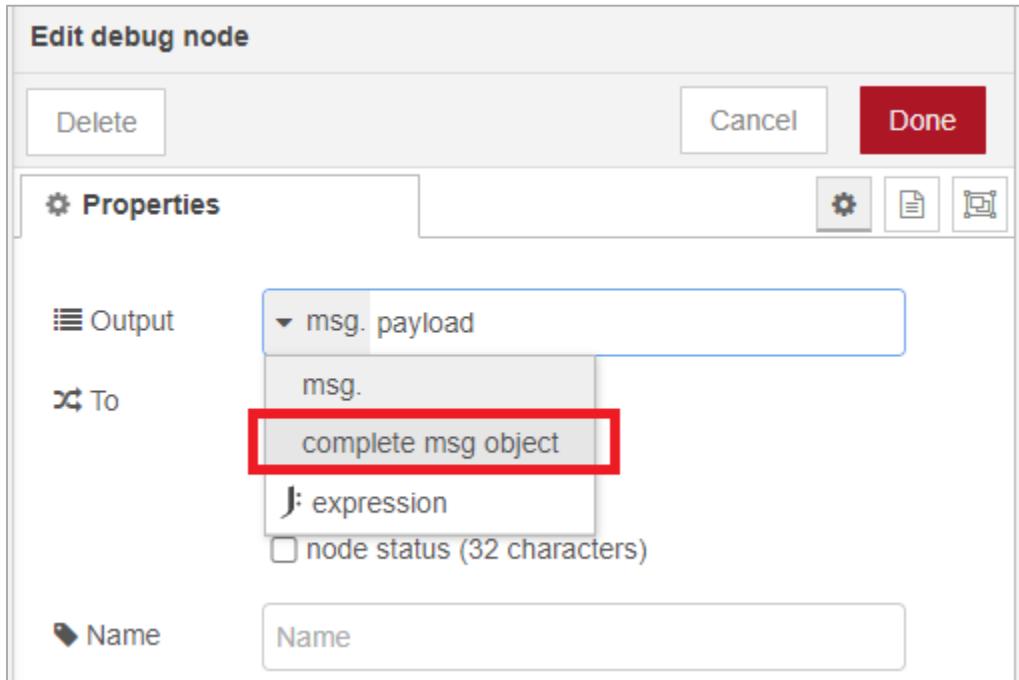
As we've seen previously, nodes exchange data using the `msg` JavaScript object. Objects are like variables, but they can contain multiple values associated with the object properties.

If this is the first time you're reading about JavaScript objects, we recommend [going to this page to learn the basics about this topic](#).

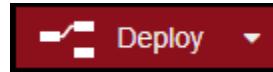
In Node-RED, the `msg` is a standard JavaScript object that can have several properties. Its properties will depend on where the message originated from.

You can see all properties of a determined message in the debugging window. But, for that, you'll need to edit the **debug** node of the flow we created previously.

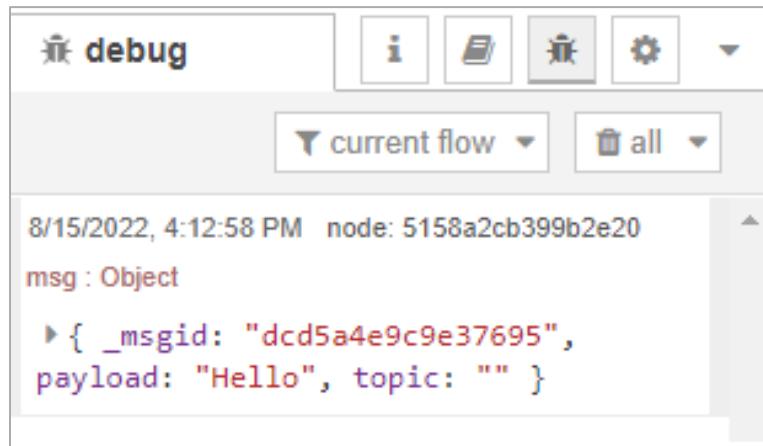
Double-click on the **debug** node and in the **Output** option, select **complete message object**. This will show all message properties on the debugging window.



After making that change, deploy your application.



Open the debugging window again, and then, click on the **Inject** node. You should see all message properties in the debugging window. In this case, the **msg** property coming from an inject node has the following properties:



- **_msgid** is a unique id that Node-RED adds to the messages that allow us to track `msg` objects;
- **payload**: is the default property most nodes work with. In our case, we have set it to `Hello`.
- **topic**: is another message property. In this example, it is empty because we haven't written anything in the **topic** field in the **inject** node. You can add something in that field, and see it being printed on the debugging window.

Modifying the msg Object

There are several nodes and different ways to change the `msg` properties. To change the properties without having to write any JavaScript code you can use a node like the **change** node.

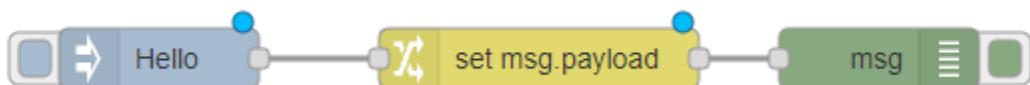


Drag the node to the flow to explore how it works. A quick look at its documentation section shows the following:

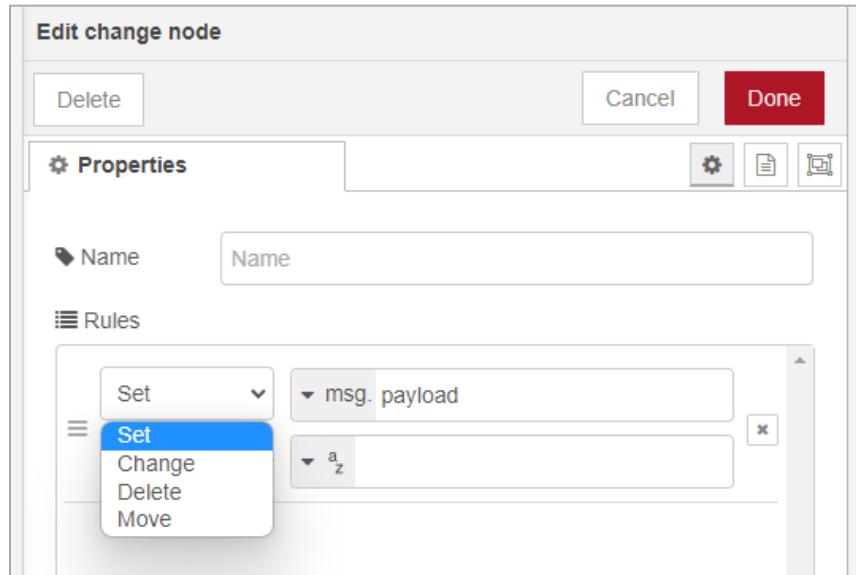
"Set, change, delete or move properties of a message, flow context, or global context."

The node can specify multiple rules that will be applied in the order they are defined."

To better understand how it works, add the **change** node between the **inject** and **debug** nodes as follows.

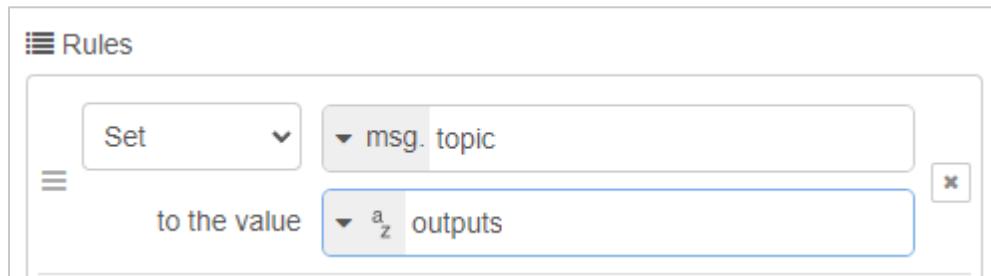


Now, double-click on the **change** node. The following window will show up.



There are several options for rules as mentioned in the documentation: **set**, **change**, **delete** and **move**. You can use those rules on the `payload` property, or other `msg` properties.

For example, at the moment the `topic` property is empty. Let's add something to the `topic` property as follows.

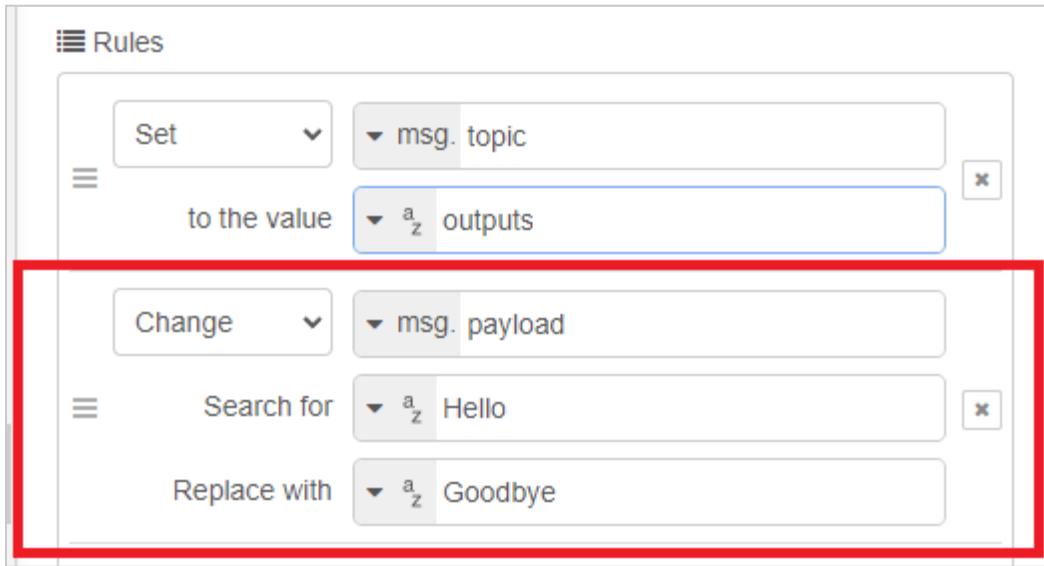


This adds the value `outputs` to the `topic` property.

Now, let's change the content of the `payload`. We set it previously to `Hello`. Let's replace the `Hello` with `Goodbye`. Click on the **+Add** button at the bottom of the window to add another rule.

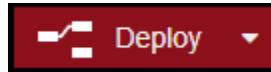


In this case, you can use the **Change** or **Set** options. Let's use the **Change** to see how it works.

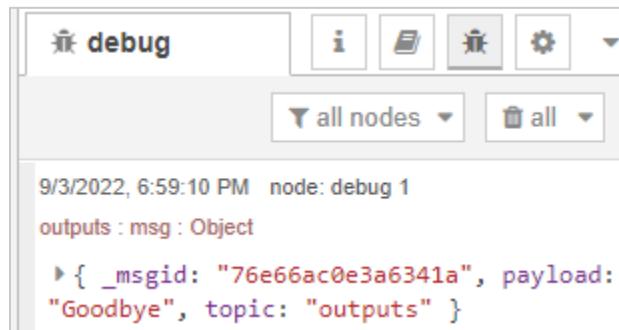


In the **Change** options, it will search for a value and replace it with any other given value. In this case, we want to change the payload and replace Hello with Goodbye.

Finally, click on **Done** and then, **Deploy** your application.



Open the Debugging window and trigger the **Inject** node. You'll get the following on the debugging window.



As you can see, now the **payload** property value is **Goodbye**, and the **topic** property has the **outputs** value as we've set in the **Change** node.

You can also explore the other features of the **Change** node and try the **Delete** and **Move** options—we won't use these much throughout the eBook, but it's useful to know they exist and how they work.

The **Delete** option deletes a property and the **Move** option moves or renames a property.

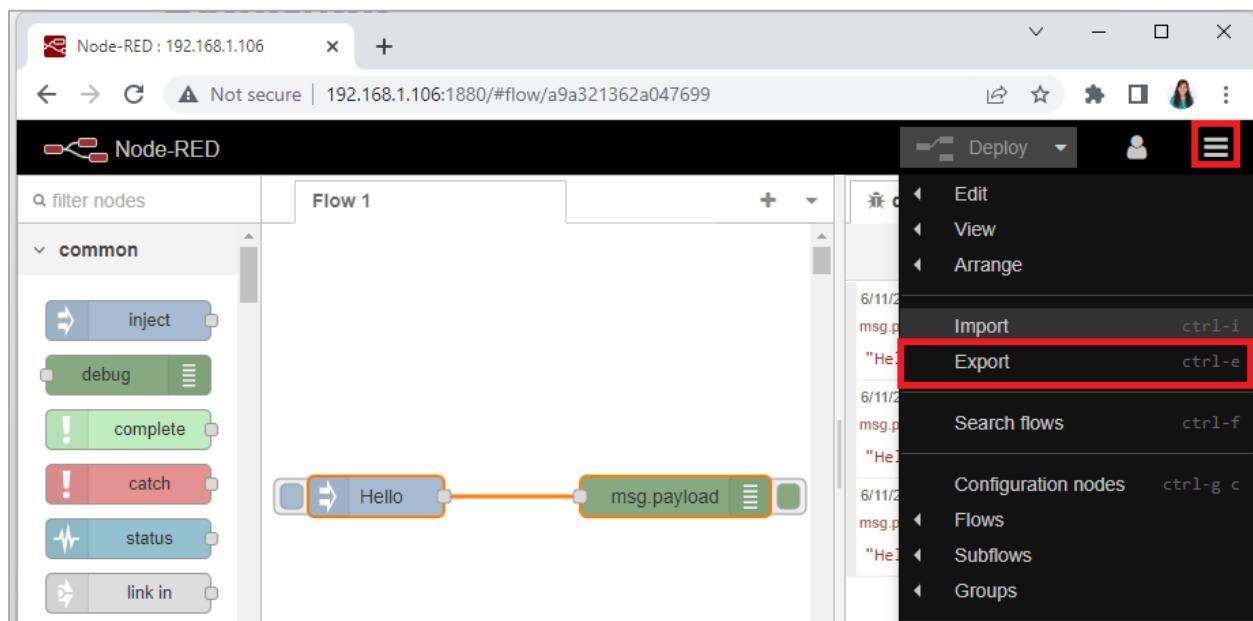
You'll better understand how the **Change** node works throughout the eBook when we start creating some useful flows.

Exporting and Importing Nodes

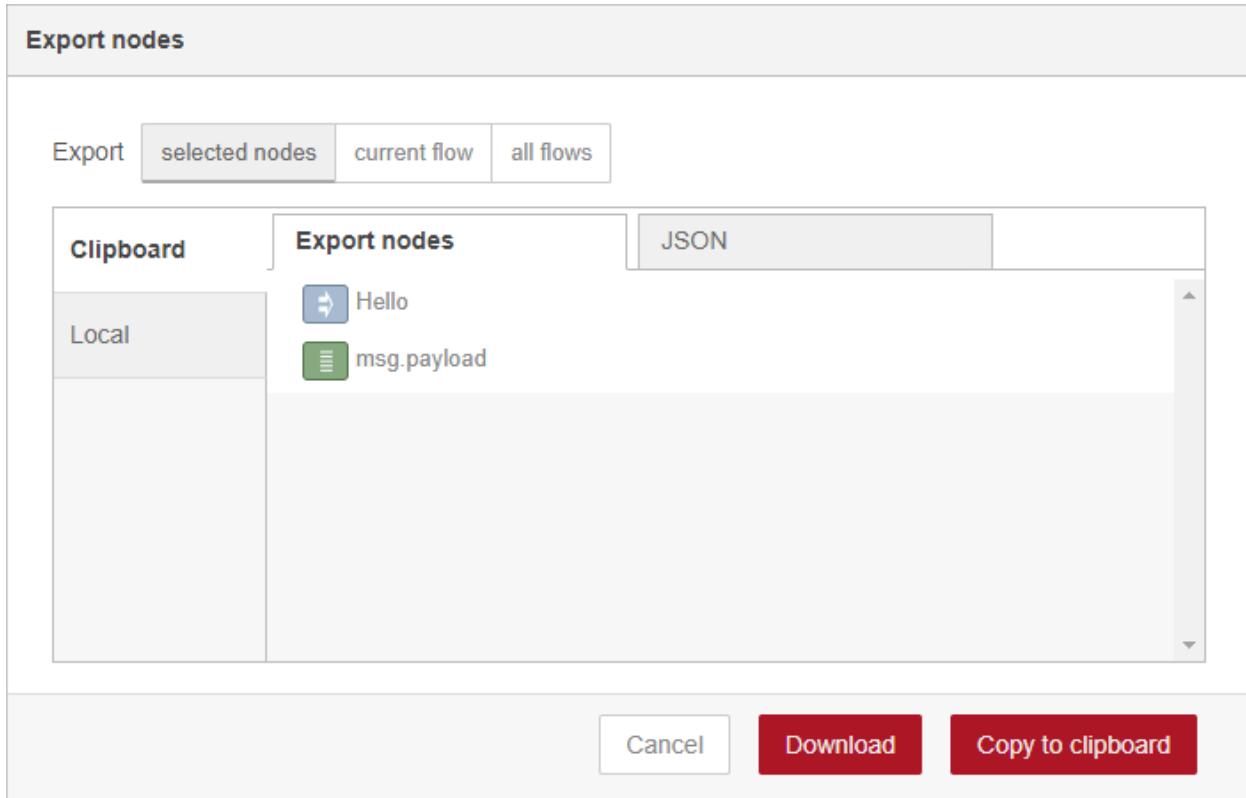
In this section, you'll learn how to save your nodes and flows. This is useful if you need to:

- Backup your Node-RED flow;
- Move your flow to another Raspberry Pi (or machine);
- Share your Node-RED project with others.

Open the main menu, and select the **Export** option.



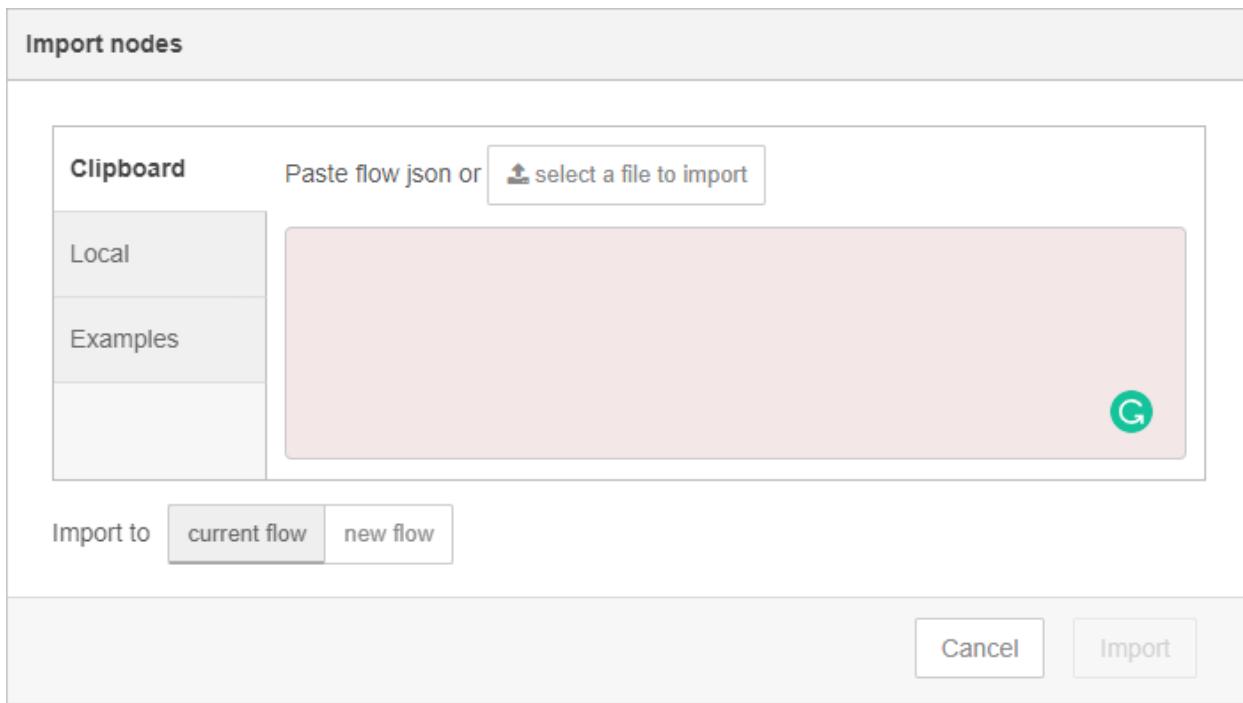
A new window opens. You can select if you want to save the selected nodes, the current flow, or all flows. You can also download the nodes as a JSON file or copy the JSON to the clipboard.



To show you how it works, click on **Download** for the selected nodes. It will download a JSON file called `flows.json`.

You can import those nodes later to another Raspberry Pi or another machine with Node-RED installed. You just need to go to the main menu and select the **Import** option.

On the **Import nodes** window, you can upload a JSON file or paste raw JSON.

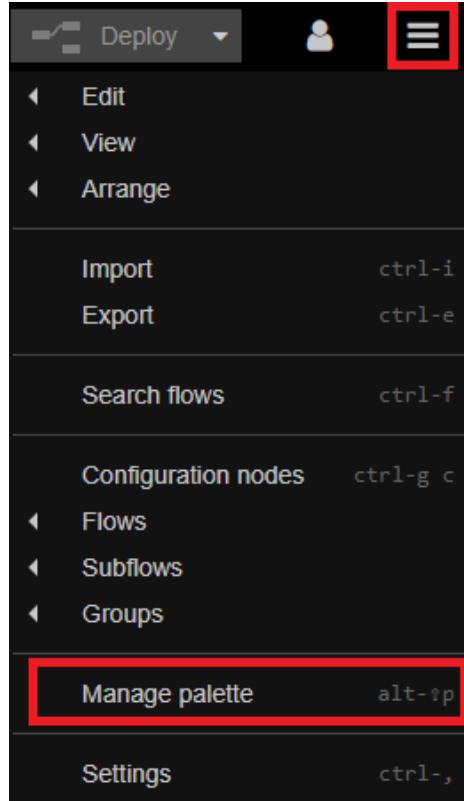


Installing Palette Nodes

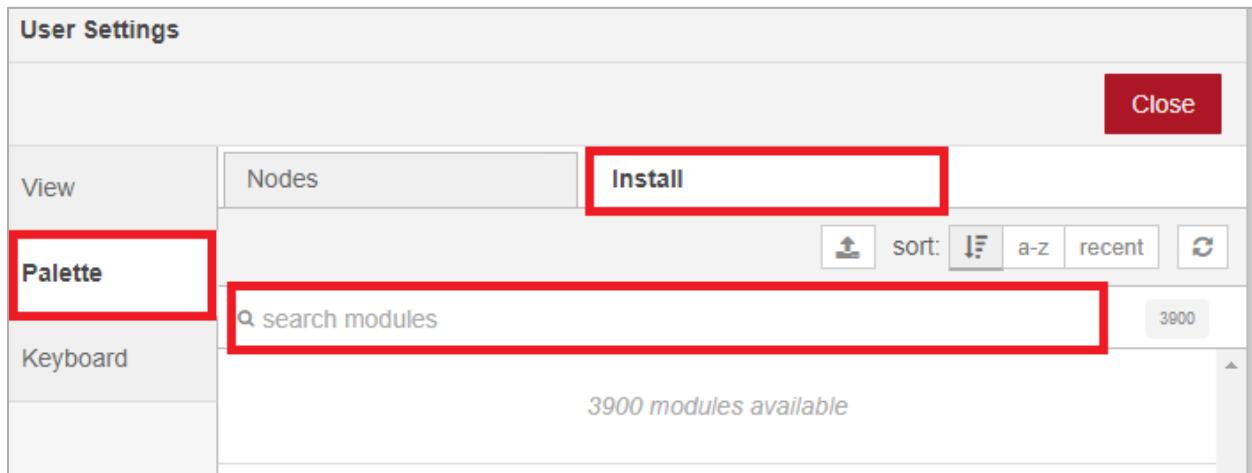
As we've seen previously, Node-RED comes with a bunch of pre-installed nodes on the Palette (left sidebar). There are many more nodes available that you can install and use for your projects. You can find them in the [Node-RED library](#). If you need some specific task for your project, there's probably already a node for that.

For example, if you need to add the feature to send an email to your flow, you can google something like this: "send email node-red node". One of the first search results is this page with the [node-red-node-email](#). It comes with some nodes to send and receive emails.

If you want to install those nodes (or any other nodes) so that you can use them on your flow, go to the main menu and select the option **Manage palette**.



The following window will open. Select the **Install** tab and search for the nodes you want to install, for example, **node-red-node-email**.



And that's how you install nodes on Node-RED.

2.4 - Node-RED Dashboard

This unit is an introduction to the Node-RED dashboard with Raspberry Pi. We'll cover how to install Node-RED Dashboard and exemplify how to build a graphical user interface for your IoT and Home Automation projects.

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. For example, it provides nodes to quickly create a user interface with buttons, sliders, charts, gauges, etc.

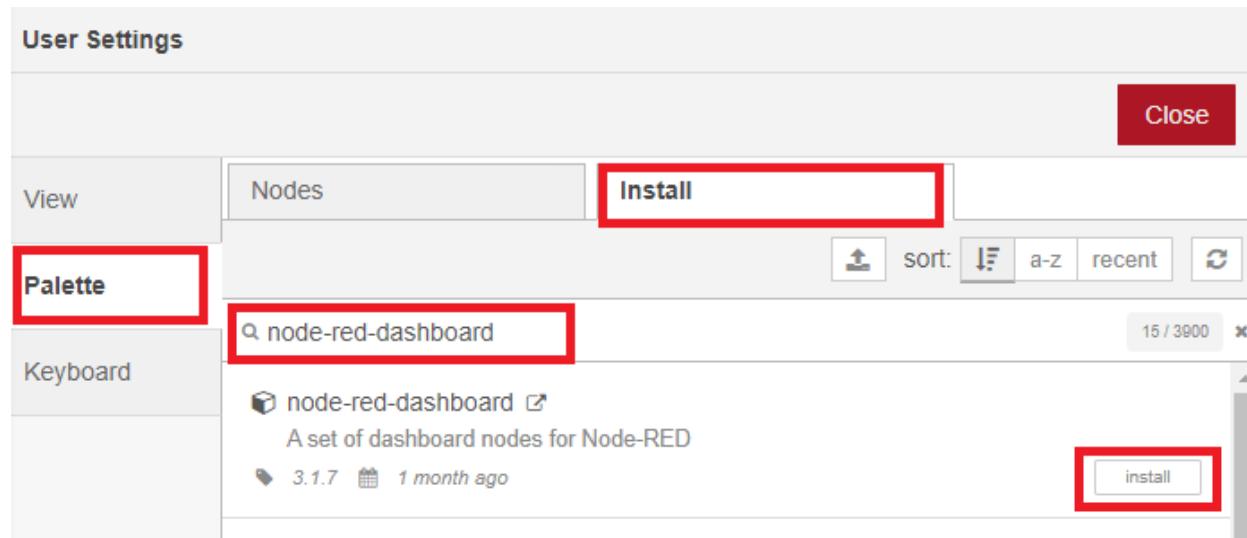
To learn more about Node-RED Dashboard you can check the following links:

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

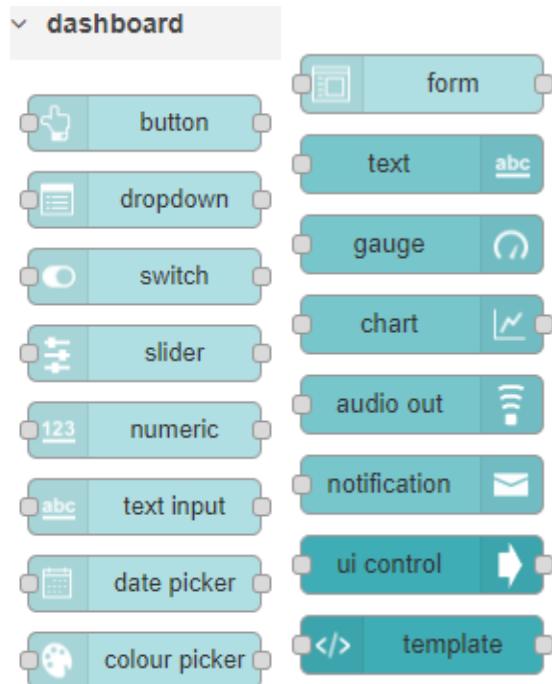
Installing Node-RED Dashboard

You can install Node-RED dashboard nodes using the **Menu > Manage Palette**.

Then, search for `node-red-dashboard` and install it.



After installing, the dashboard nodes will show up on the palette.

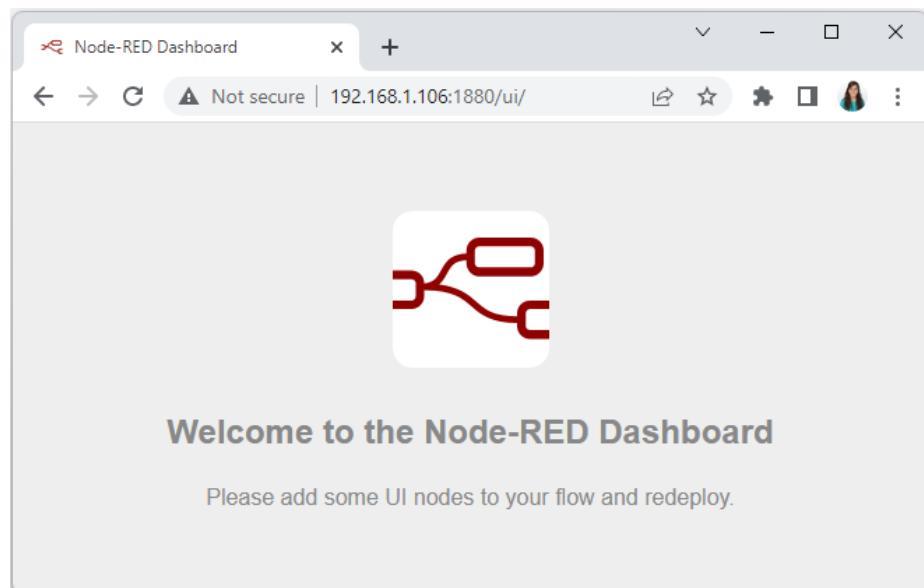


Dashboard nodes provide widgets that show up in your application user interface (UI). The user interface is accessible on the following URL:

```
http://Your_RPi_IP_address:1880/ui
```

For example, in my case:

```
http://192.168.1.106:1880/ui
```



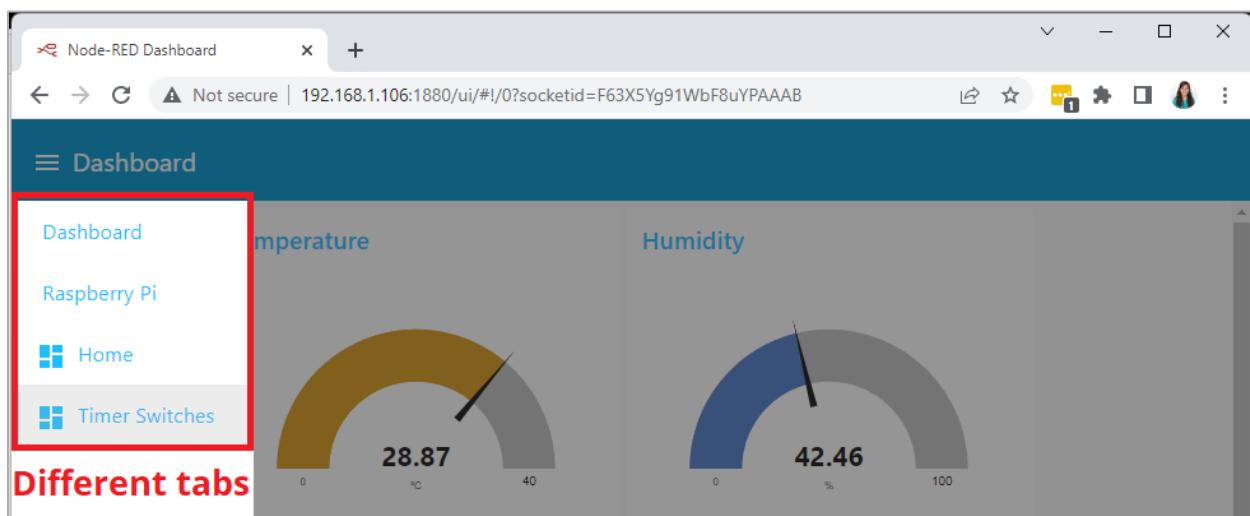
At the moment, you'll see the previous screen when you access the UI. That's because you haven't added any of those dashboard nodes to the flow. We'll do that in the following section.

Creating a UI (User Interface)

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

The Dashboard Layout

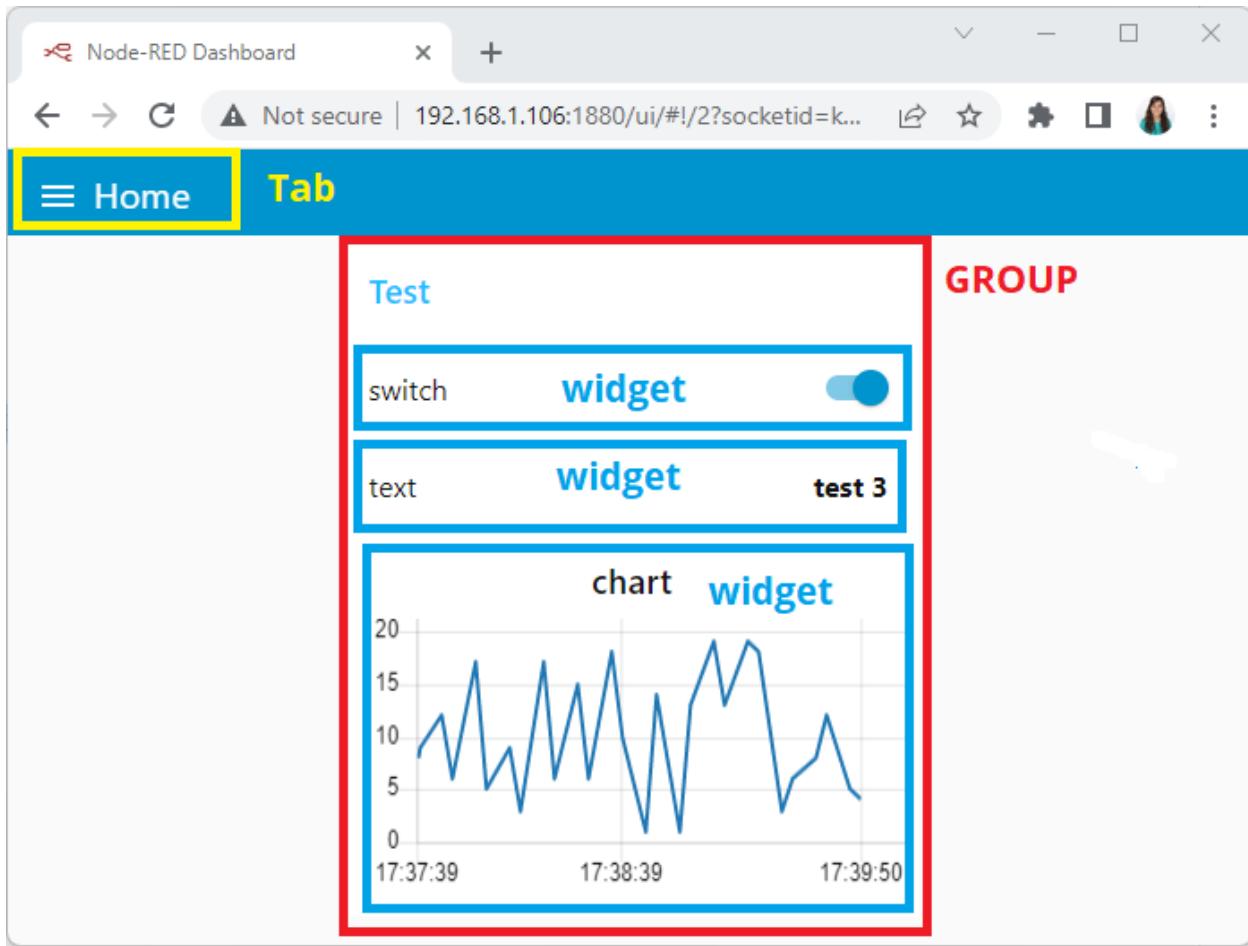
The user interface is organized with tabs and groups. Tabs are different pages on your user interface, like several tabs in a browser (see picture below).



Inside each tab, you have groups that divide the tabs into different sections so that you can organize your widgets (buttons, sliders, charts, gauges, forms, etc.).

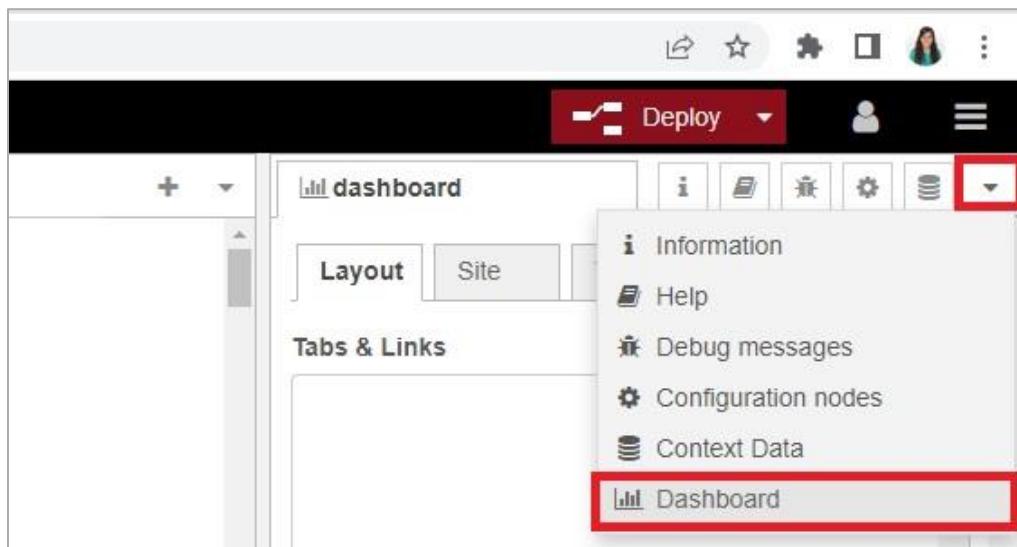
Every widget should have an associated group that determines where the widget should appear on the user interface.

In the picture below, **Home** is the tab, **Test** is the name of the group, and the **switch**, **text**, and **chart** are different widgets.



To create a tab and a group, follow the next instructions.

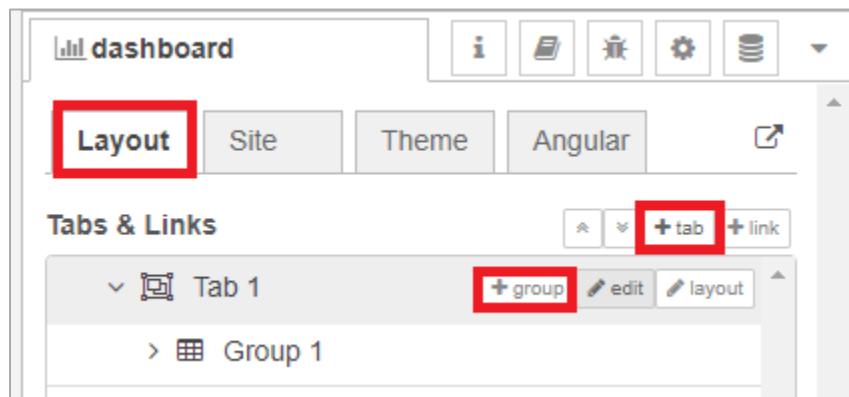
On the top right corner, click on the little arrow icon, and click on **Dashboard**.



Creating Tabs and Groups

Make sure you have the **Layout** tab selected. Then, click on the **+tab** button to create a tab, it will be called **Tab 1** by default.

After creating a tab, you can create several groups under that tab. You need to create at least one group to add your widgets. Click on the **+group** button to create a group inside that tab—it will be called **Group 1** by default.

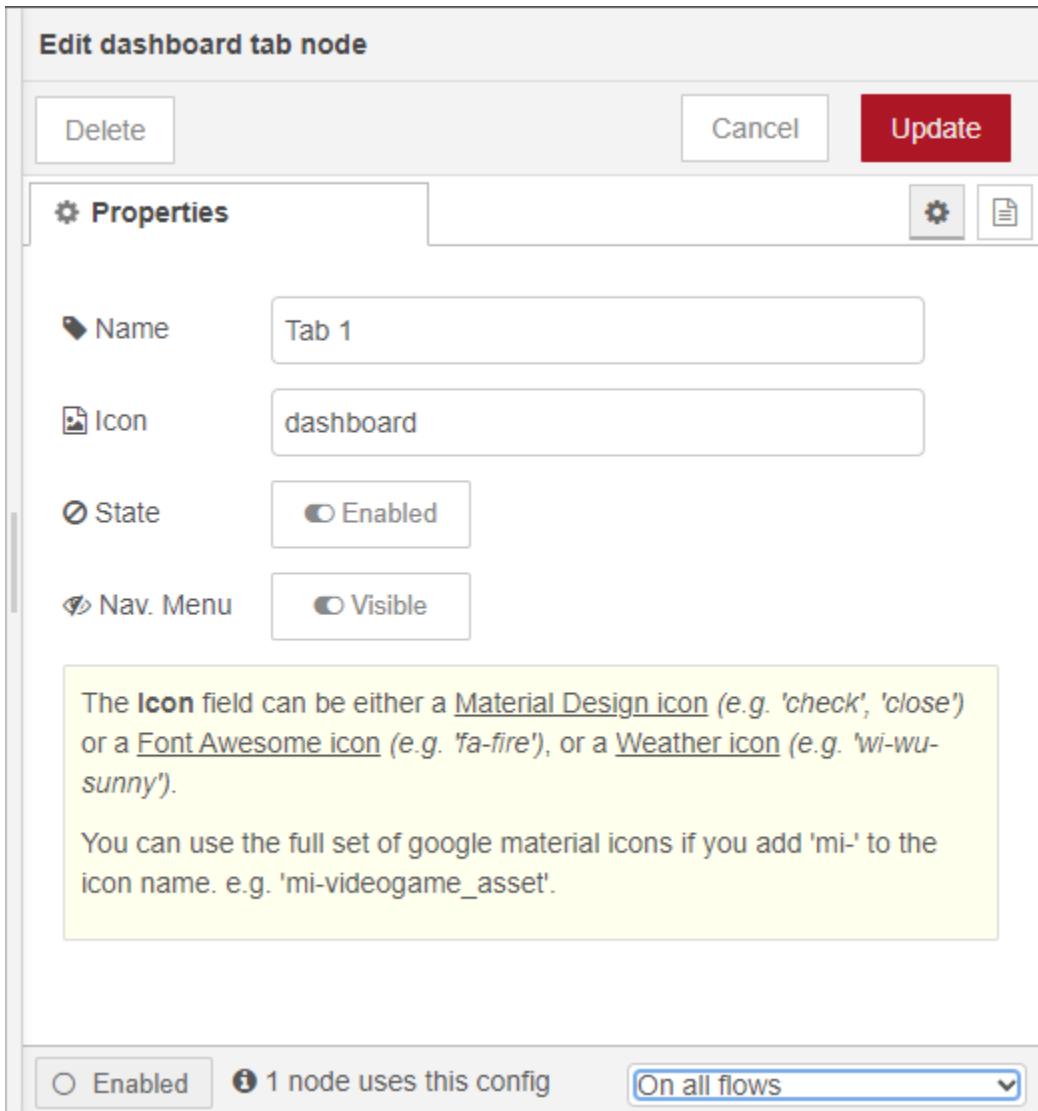


You can click on the Tab and Group **edit** buttons to change their properties. For example, click the **edit** button for **Tab 1**.

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

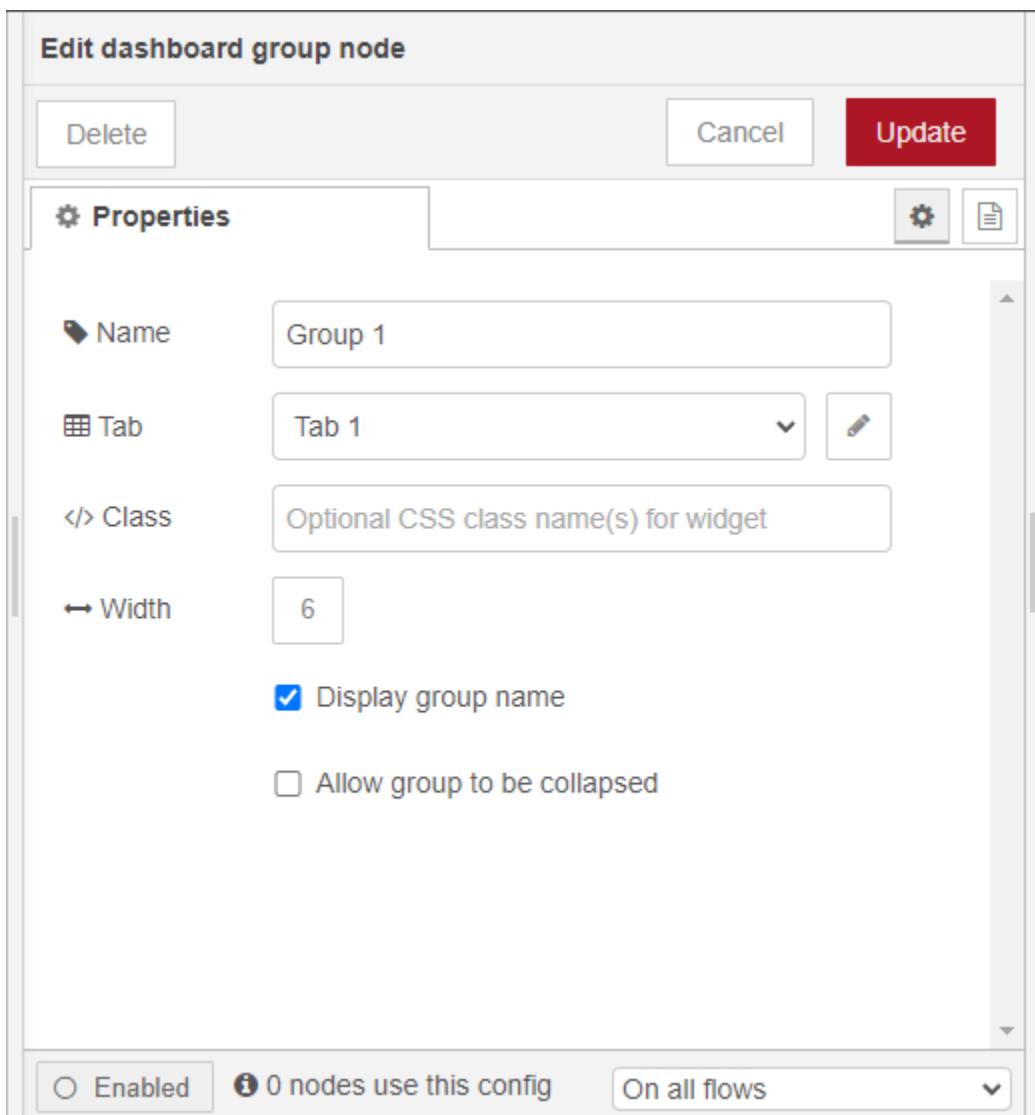
- **Name:** you can call it whatever you want
- **Icon:** you should use a name accordingly to the icon's names in the links provided. By default, it uses the dashboard icon.

After making changes, click the **Update** button (see screenshot below).



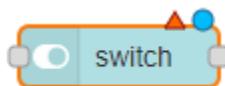
Now, edit the group, by clicking on the group **edit** button.

You can edit its name, select its corresponding tab and change its width. According to the Node-RED dashboard documentation, each group element has a width of 6 'units' (a unit is 48px wide by default with a 6px gap)—see the next screenshot.

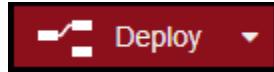


At the moment, you have created a tab and a group using Node-RED dashboard. To see them, you need to add a widget.

For example, add a **switch** (from the *dashboard* section, not from the *function* section—you need to scroll down through the palette to find the dashboard nodes) node to the flow.



Double-click on it to check its properties. By default, it will be added to the group and tab you created previously. Deploy your app.

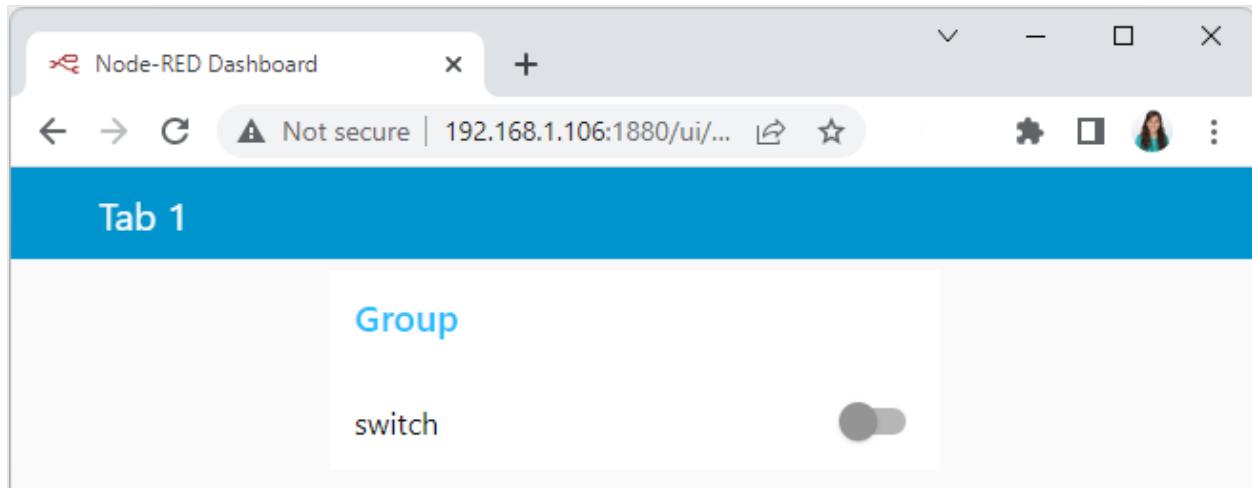


Now, go to the following URL, to access the user interface created.

`http://Your_RPi_IP_address:1880/ui`

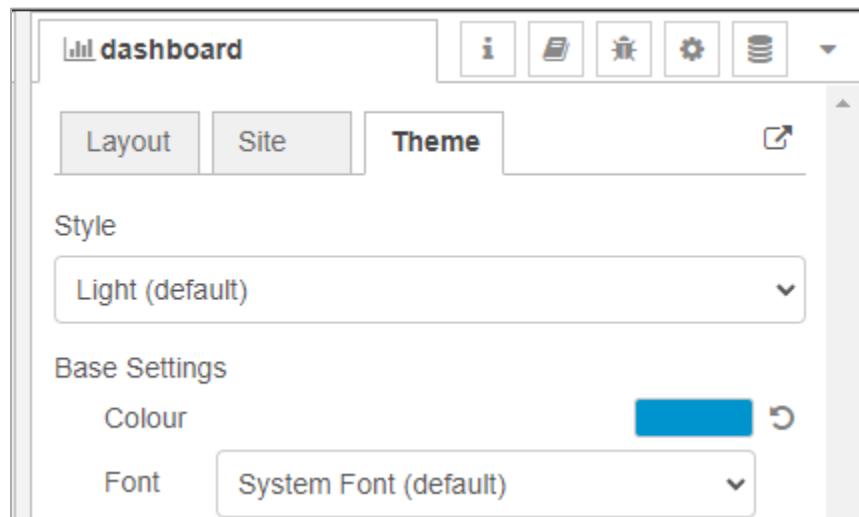
In my case:

`http://192.168.1.106:1880/ui`

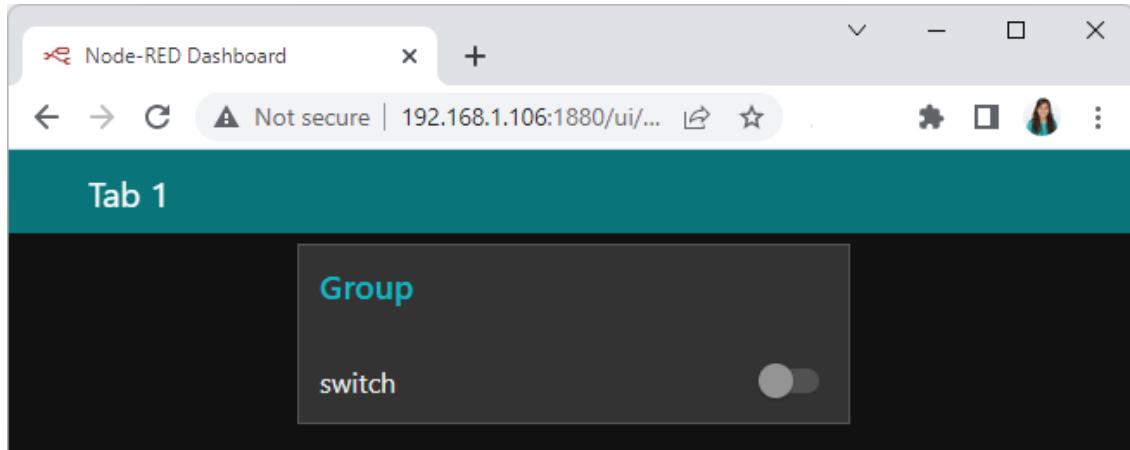


Dashboard Theme

The Node-RED Dashboard has a white background and a light blue bar by default. You can edit its colors and font in the **Theme** tab when you open the **dashboard** option.

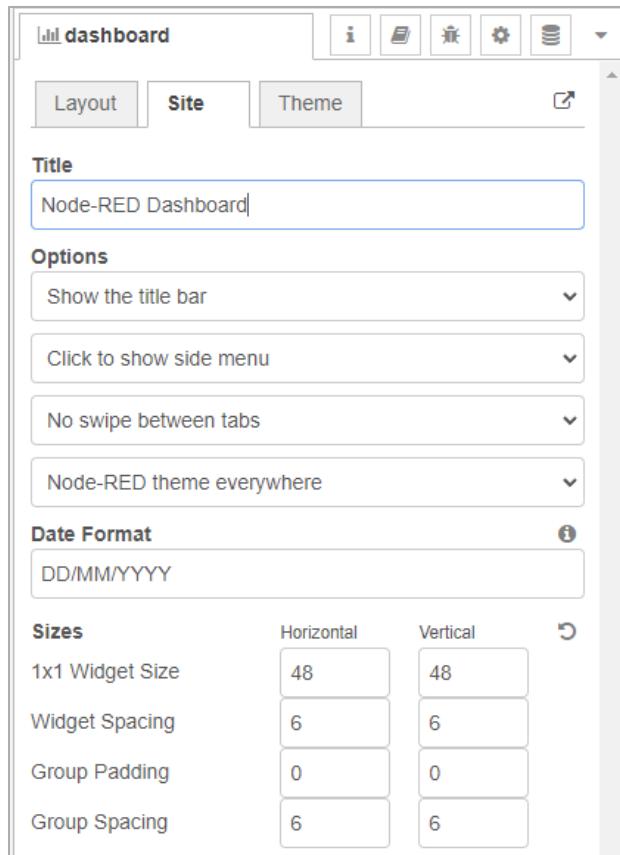


Change the style, deploy the changes and see the Dashboard UI changing its colors.
For example, in the following figure:



Dashboard Site

At the **dashboard** window, you have another tab called **Site** that allows you to do further customization as shown in the figure below.



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 only have one widget. Those changes will be noticeable when you start adding more widgets to the UI.

Creating a User Interface - Example

In this section we're going to make a dashboard example to show you how you can build and edit your dashboard—we won't add functionalities to the widgets, we'll do that in future projects. This dashboard will have the following features:

- Two different tabs: one called Room and another called Garden
- The Room tab will have two groups and the Garden tab will have one group
- We'll add a color picker and a switch to the Room group
- We'll add a chart to the Garden group

Creating the Tabs

On the top right corner of the Node-RED window, select the **dashboard** tab and create two new tabs by clicking on the **+tab** button.

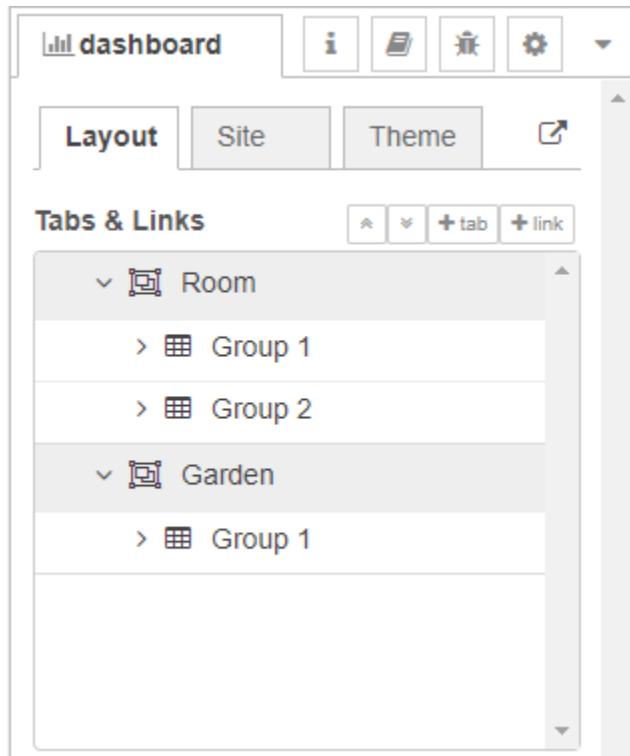
Edit one tab with the following properties:

- Name: **Room**
- Icon: **tv**

And the other one with the following:

- Name: **Garden**
- Icon: **local_florist**

Then, add two groups to the **Room** tab and one group to the **Garden** tab. The following figure shows how your dashboard layout looks.

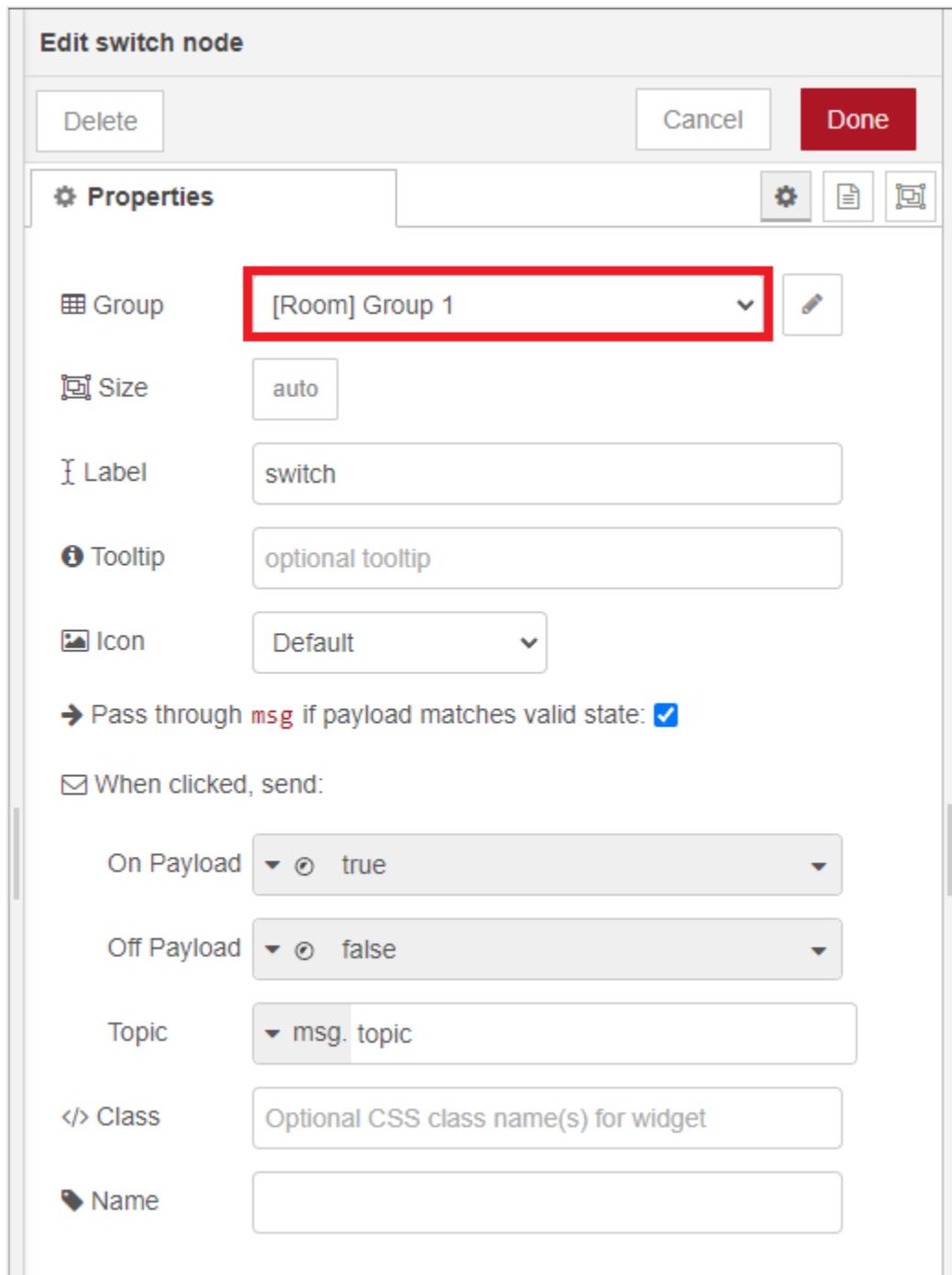


Adding the Widgets

Add a **switch**, a **slider**, a **colour picker** and a **gauge** to the flow as shown in the following figure:



Double click on the **switch**. A new window pops up.



In this new window, you can choose where you want to place your widget on the user interface. In this case, we want it to appear in the **Room tab, Group 1** as highlighted in red in the previous figure.

Then, do the same for the other widgets but add them to the following groups:

- slider: Group 1 [Room]

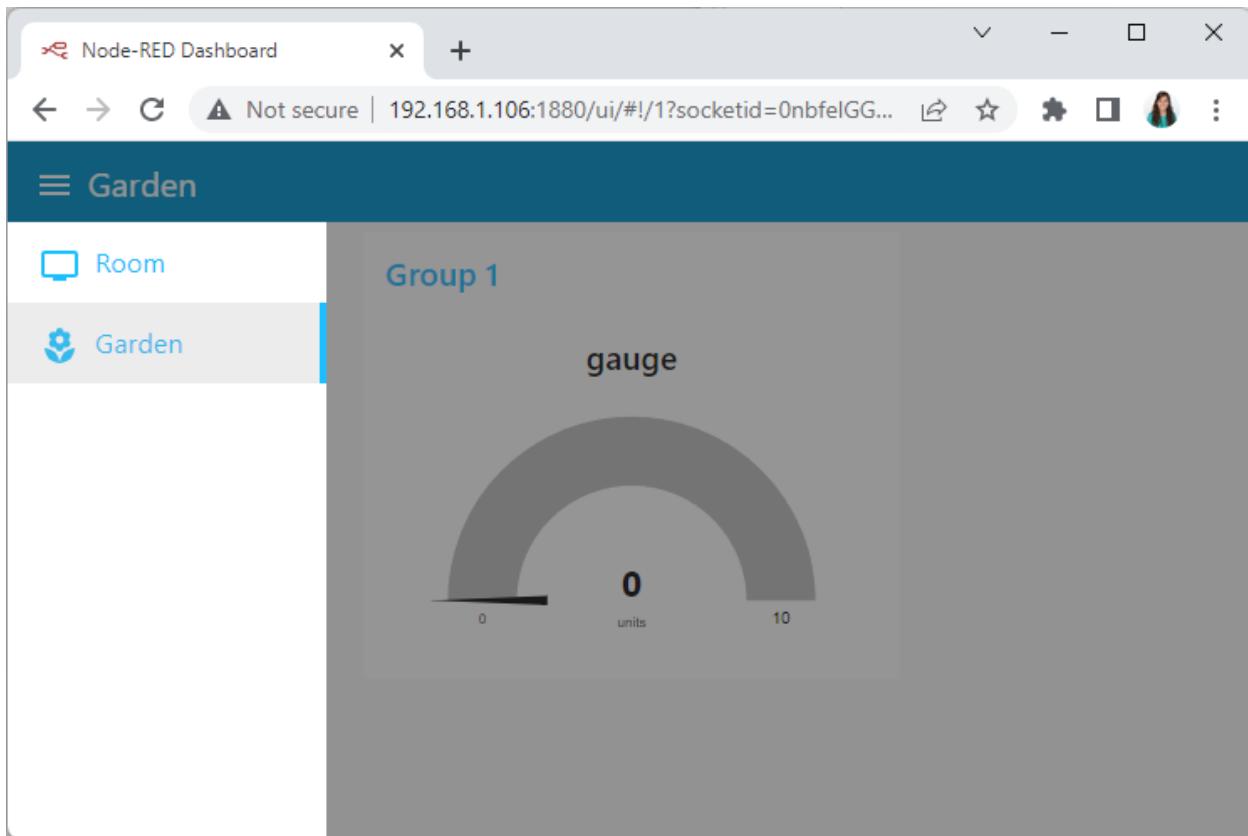
- colour picker: Group 2 [Room]
- gauge: Group 1 [Garden]

Deploy the changes and go to the dashboard UI to see how it looks.

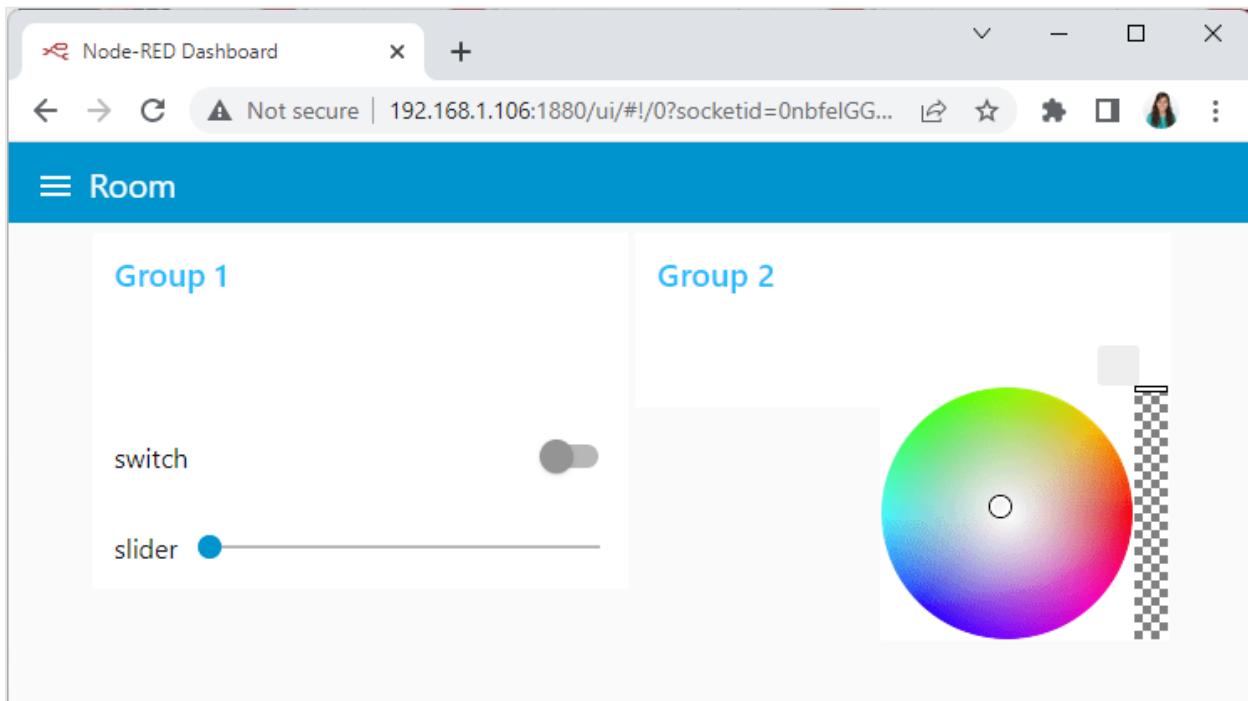


`http://Your_RPi_IP_address:1880/ui`

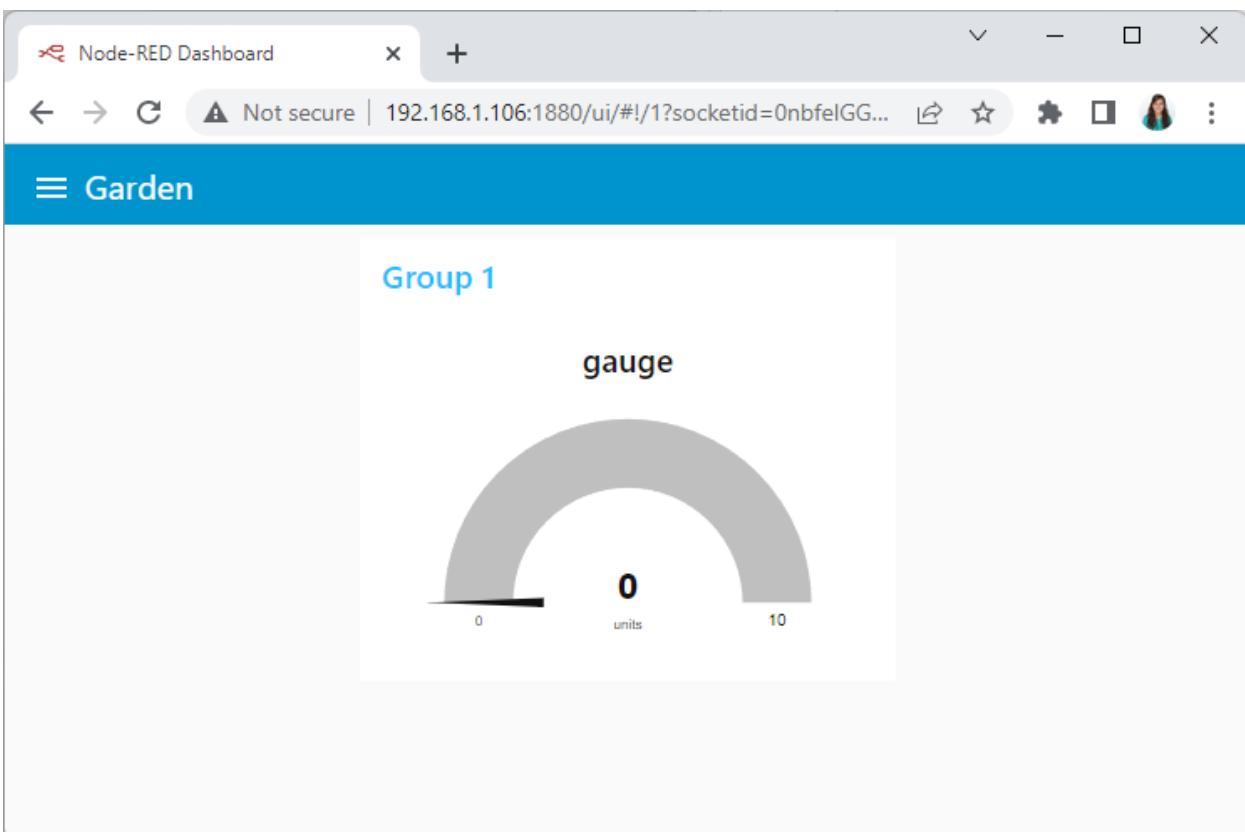
Your dashboard has two tabs: the **Room** and the **Garden**. Click on the menu icon at the top leftside to see the tabs as shown in the following figure:



Here's what the **Room** tab looks like with the two groups (you need to click on the little gray square on Group 2 for the color picker to show up).



And here's how the **Garden** tab looks with one group.



Secure Node-RED Dashboard with Username and Password

As we've seen previously, you can create a dashboard (UI) using Node-RED dashboard nodes. Node-RED editor is protected using username and password, but not the user interface. After creating a user interface, if you want to protect it with username and password, follow the next steps.

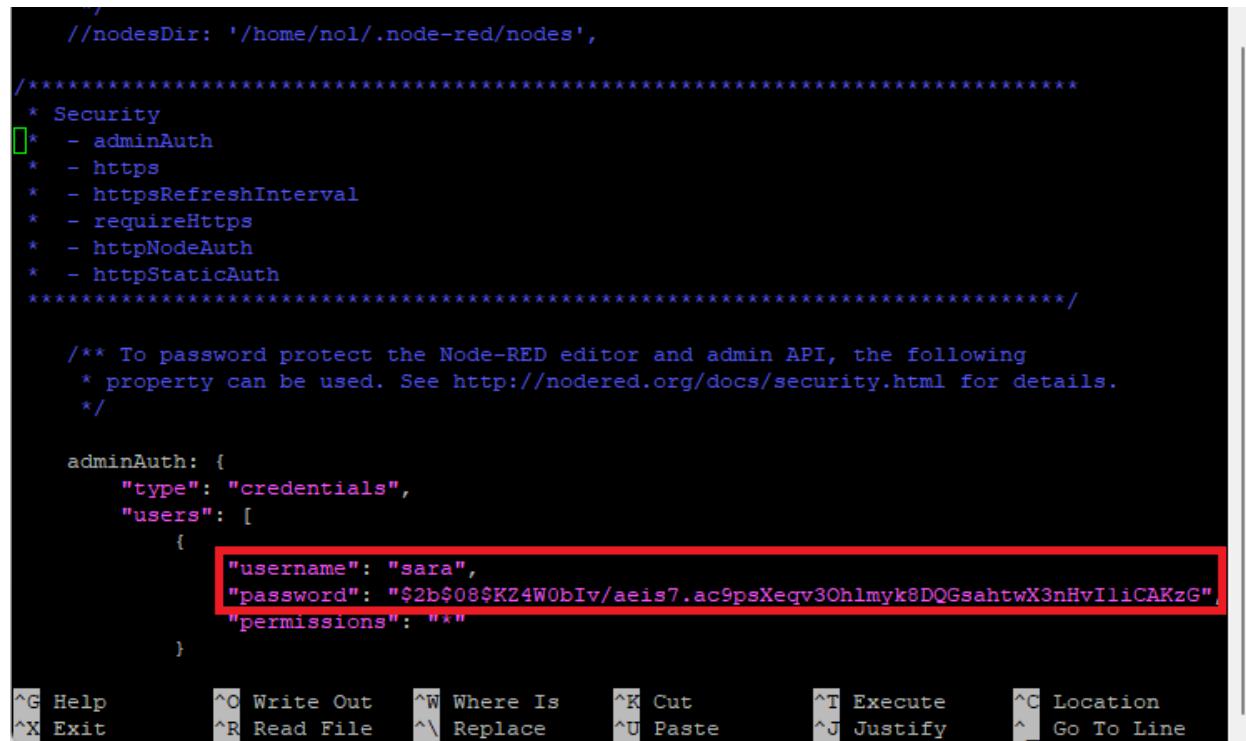
- 1) Establish an SSH connection with your Raspberry Pi and run the following command to edit the `settings.js` file.

```
nano ~/.node-red/settings.js
```

- 2) Scroll down through the file using the arrow keys until you find the section that mentions.

```
/* To password protect the Node-RED editor and admin API, the  
following property can be used...*/
```

- 3) Copy the username and password combination with **CTRL+C**.



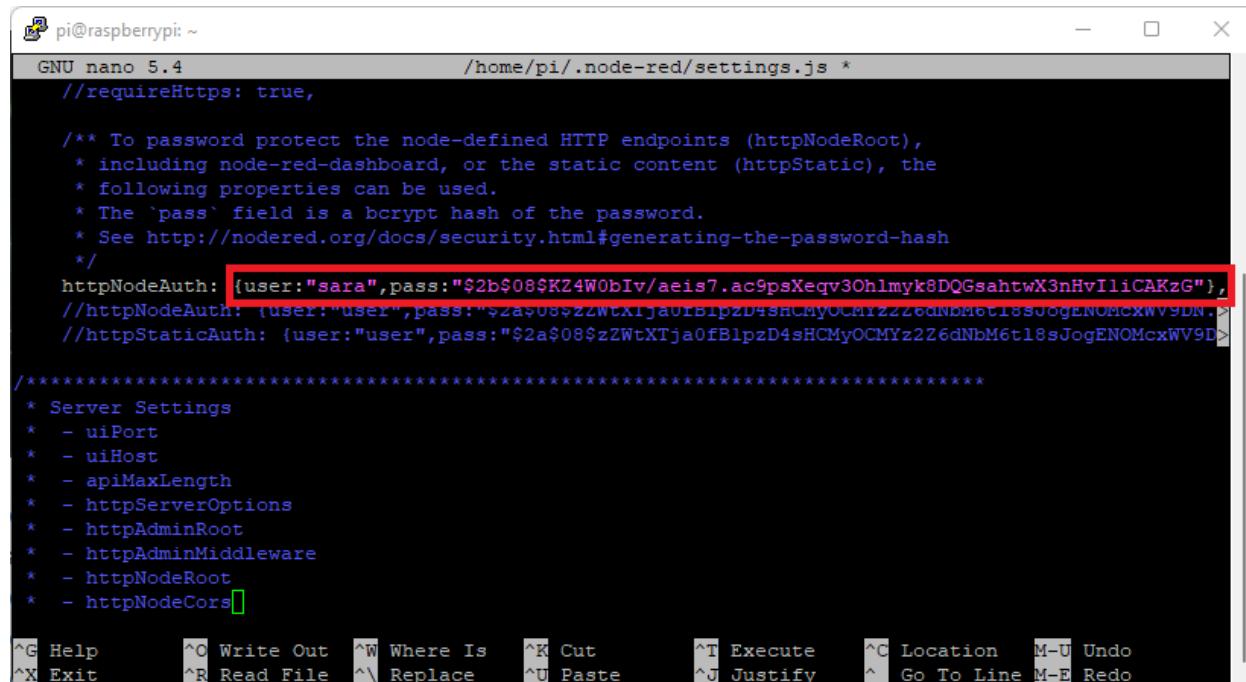
```
//nodesDir: '/home/nol/.node-red/nodes',  
  
/*  
 * Security  
 * - adminAuth  
 * - https  
 * - httpsRefreshInterval  
 * - requireHttps  
 * - httpNodeAuth  
 * - httpStaticAuth  
 */  
  
/** To password protect the Node-RED editor and admin API, the following  
 * property can be used. See http://nodered.org/docs/security.html for details.  
 */  
  
adminAuth: {  
    "type": "credentials",  
    "users": [  
        {  
            "username": "sara",  
            "password": "$2b$08$KZ4W0bIv/aeis7.ac9psXeqv3Oh1myk8DQGsahtwX3nHvIliCAKzG",  
            "permissions": "*"  
        }  
    ]  
}  
  
^G Help      ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location  
^X Exit      ^R Read File    ^\ Replace     ^U Paste         ^J Justify      ^_ Go To Line
```

4) Continue scrolling down the file until you find the following section:

```
/** To password protect the node-defined HTTP endpoints
(httpNodeRoot),
* including node-red-dashboard, or the static content
(httpStatic), the
* following properties can be used.
```

5) Edit the `httpNodeAuth` with the user and password you copied previously. In my case, the file will look as follows:

```
httpNodeAuth: {user:"sara",pass:"$2b$08$KZ4W0bIv/aeis7.ac9psXeqv3
Ohlmyk8DQGsahtwX3nHvI1iCAKzG"},
```



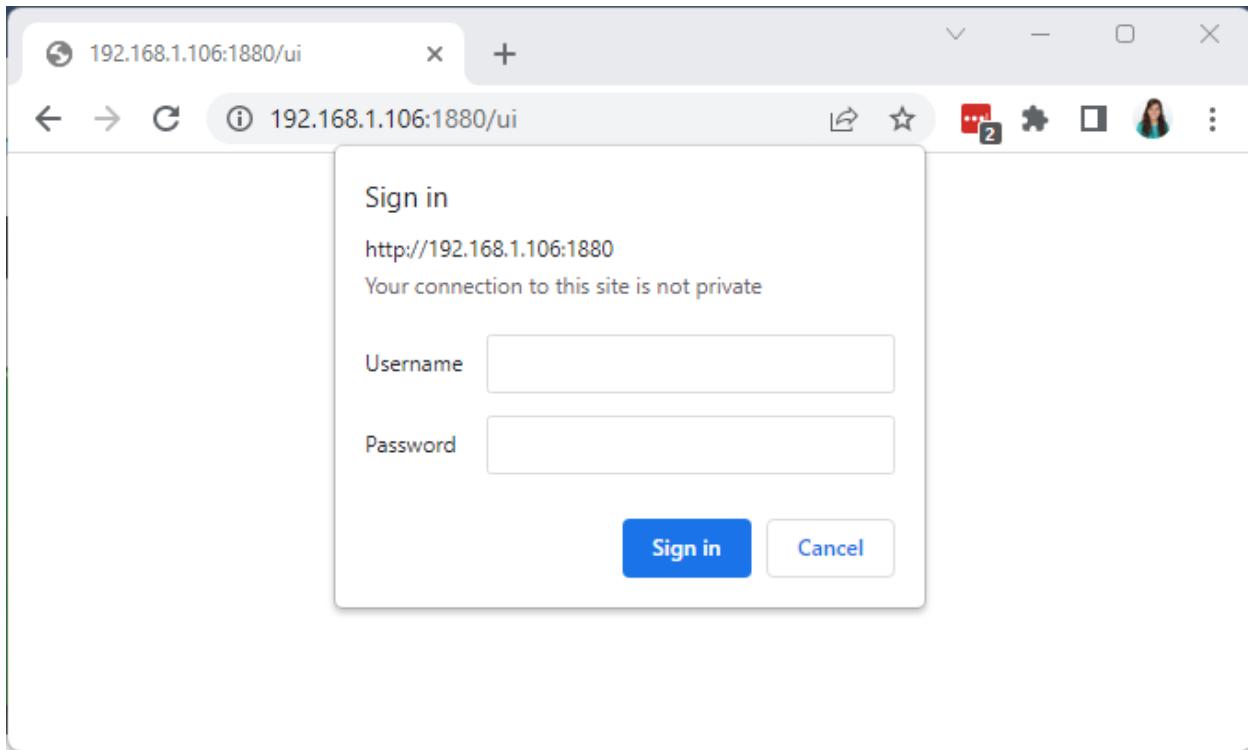
```
pi@raspberrypi: ~
GNU nano 5.4          /home/pi/.node-red/settings.js *
//requireHttps: true,
/** To password protect the node-defined HTTP endpoints (httpNodeRoot),
 * including node-red-dashboard, or the static content (httpStatic), the
 * following properties can be used.
 * The `pass` field is a bcrypt hash of the password.
 * See http://nodered.org/docs/security.html#generating-the-password-hash
 */
httpNodeAuth: {user:"sara",pass:"$2b$08$KZ4W0bIv/aeis7.ac9psXeqv3Ohlmyk8DQGsahtwX3nHvI1iCAKzG"}, //httpNodeAuth: {user:"user",pass:"$2a$08$z2WtXTja0fBlpzD4sHCMyc2Z6dNbM6tl8sJogENOMcxWV9D"} //httpStaticAuth: {user:"user",pass:"$2a$08$z2WtXTja0fBlpzD4sHCMyc2Z6dNbM6tl8sJogENOMcxWV9D"}
*****
* Server Settings
* - uiPort
* - uiHost
* - apiMaxLength
* - httpServerOptions
* - httpAdminRoot
* - httpAdminMiddleware
* - httpNodeRoot
* - httpNodeCors[]

^G Help      ^O Write Out   ^W Where Is    ^K Cut        ^T Execute     ^C Location   M-U Undo
^X Exit      ^R Read File   ^\ Replace     ^U Paste      ^J Justify    ^^ Go To Line M-E Redo
```

- 6) After making the required changes, press **CTRL+X**, then **Y** and **Enter** to save the changes.
- 7) Reboot your Raspberry Pi.

```
sudo reboot
```

Now, the next time you try to access Node-RED dashboard you need to insert a username and password. The user/password combination is the same you use to access Node-RED editor.



Wrapping Up

At the moment, you should be more familiar with Node-RED dashboard nodes and how they work. You can add several widgets like sliders, buttons, forms, charts, and gauges to create a nice application for your IoT or home automation projects.

In the example presented, we only created the user interface, we haven't added any functionalities to the widgets as the aim of this unit was to get you familiar with Node-RED dashboard.

We'll create user interfaces that actually do something from now on. So, continue to the next units.

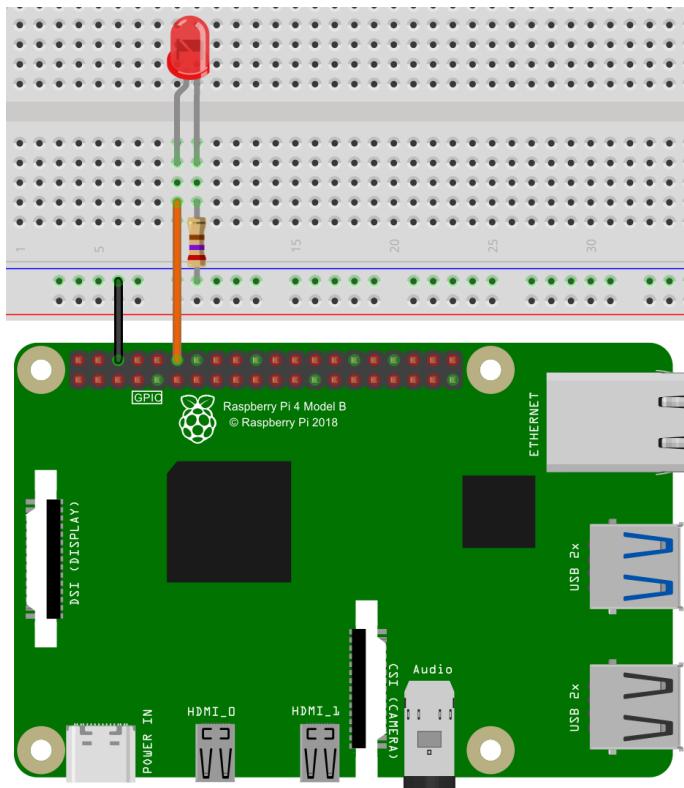
2.5 - Controlling an LED with Node-RED

Controlling an LED it's usually the very first thing you do when you play with a new development board. So, let's control an LED connected to one of the Raspberry Pi GPIOs using Node-RED.

Wiring the Circuit

Here's the hardware required to complete this unit:

- LED (5mm)
- 270Ω or similar resistor
- Breadboard
- Jumper wires

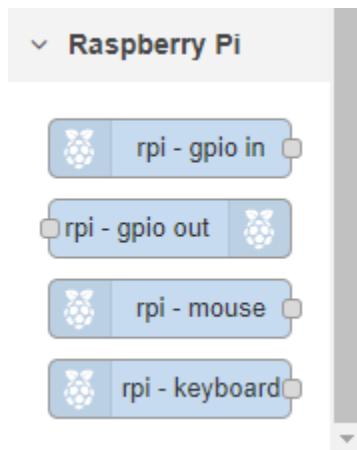


Wire your circuit by following the image above or these instructions:

- Longest lead of the LED → pin 12 (GPIO 18)
- Shortest lead of the LED →resistor
- Resistor connected to GND

Installing Raspberry Pi Specific Nodes

Raspberry Pi Nodes should be installed by default. Check if you have them on the Palette. Raspberry Pi nodes allow you to control the Raspberry Pi hardware (GPIOs, mouse, and keyboard) using Node-RED.



If something went wrong during your installation and those nodes are not installed by default, you can install them now.

Menu > Manage Palette > Install and search for `node-red-node-pi-gpio`. Install that package.

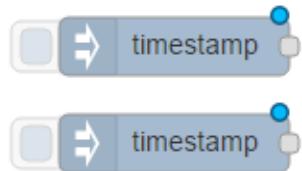
After installing, you should have new nodes under the Raspberry Pi Nodes list.

Creating the Flow

Drag two **inject** nodes into your flow (you can delete the nodes from previous examples).



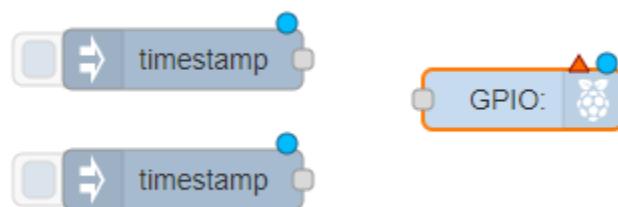
They will look like this in your flow:



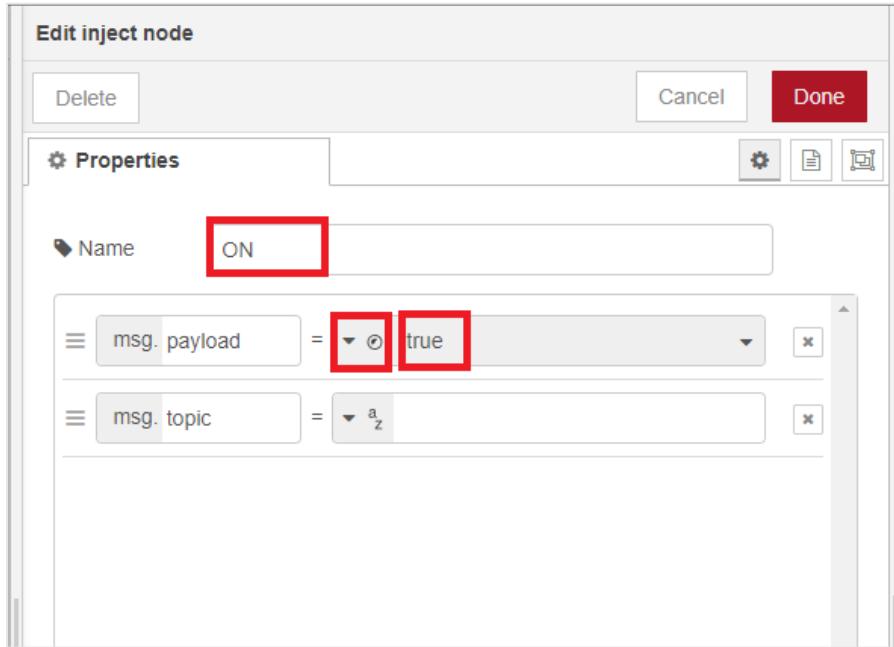
Then, add the **rpi-gpio out** node to your flow:



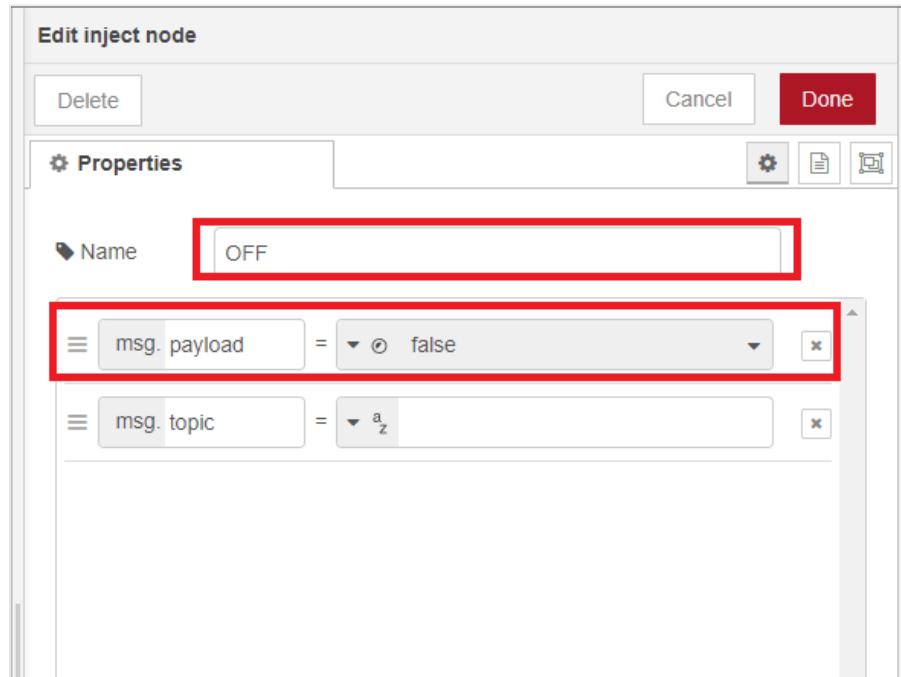
It will look as shown below:



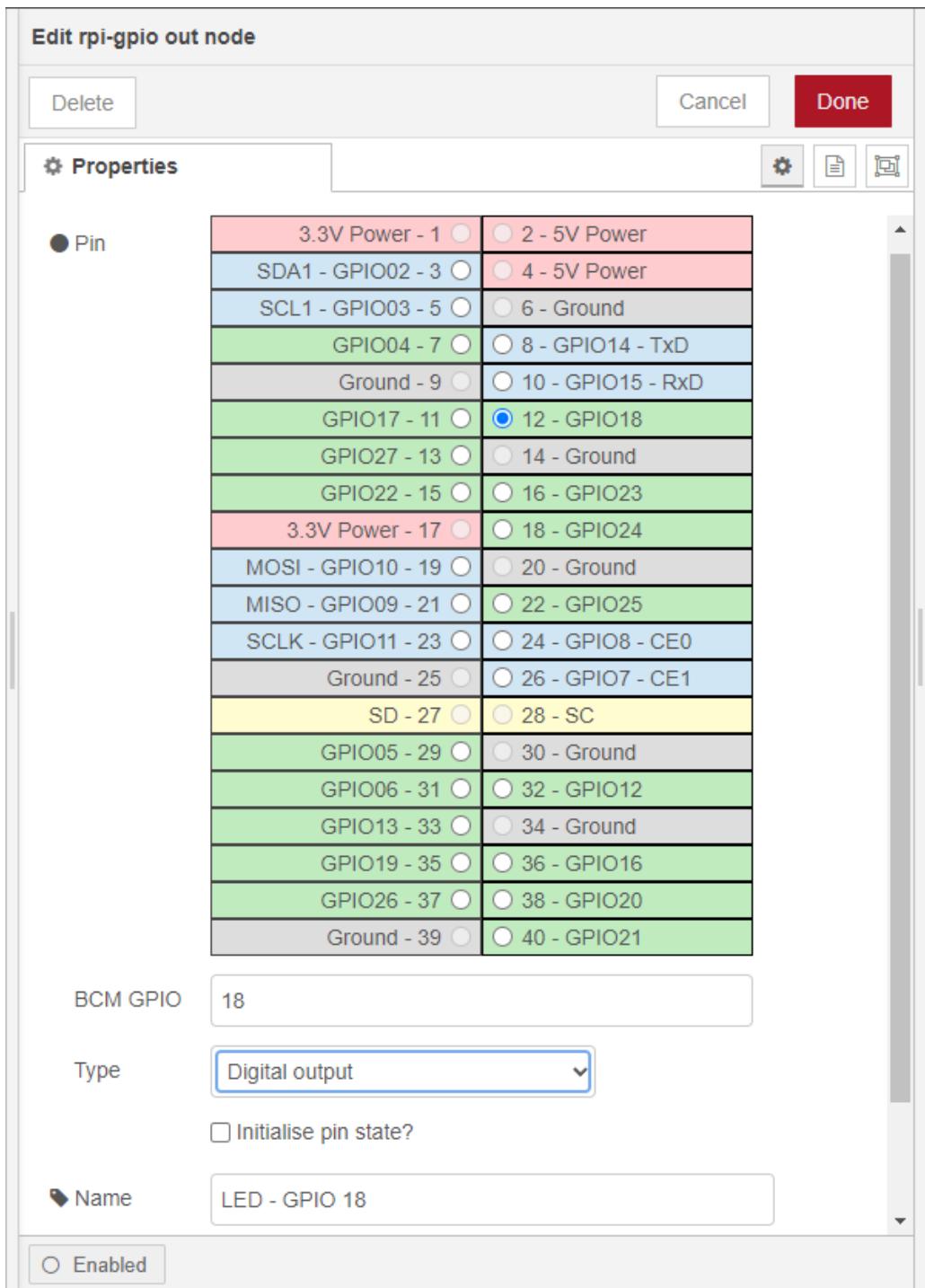
Double-click the first **inject** node to edit its properties. Select **boolean** and **true**. You can also give a name to the node. Because this will turn the LED on, we'll call it **ON**.



Edit the other **inject** node. Select boolean and **false**. Name it **OFF**.

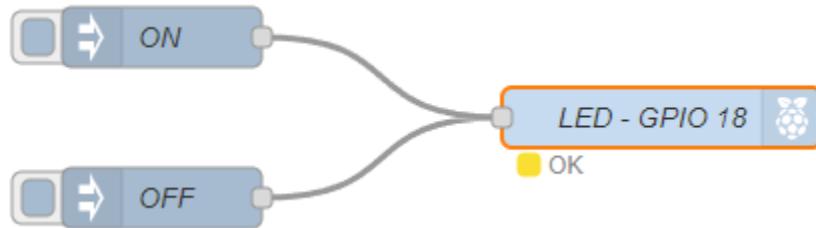


Lastly, edit the **rpi-gpio out** node. Select the GPIO you want to control, in our case it's Pin12 (GPIO 18).

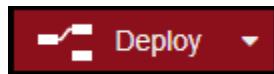


Set the **Type** as a **Digital output** and give a name to the node, for example **LED - GPIO 18**.

Wire the **inject** nodes to the Raspberry Pi node.

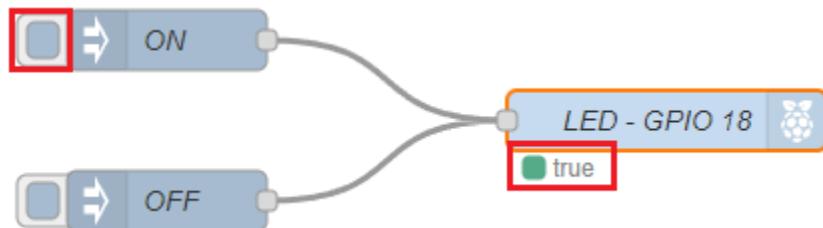


Finally, deploy your application by clicking on the Deploy button.

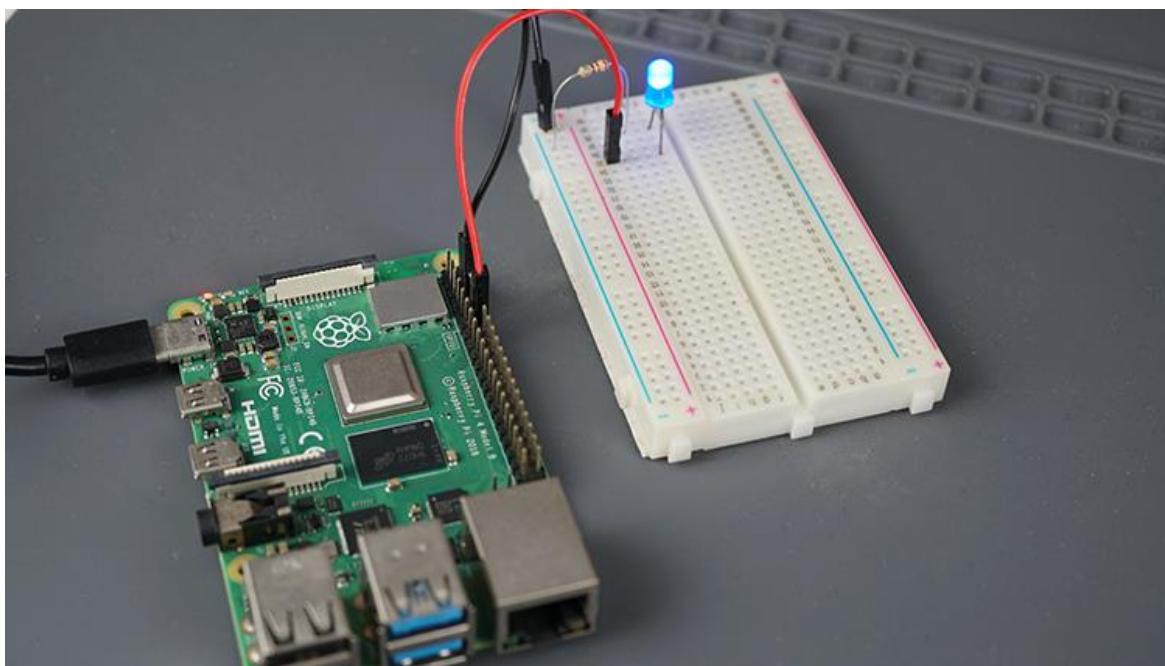


Testing the Flow

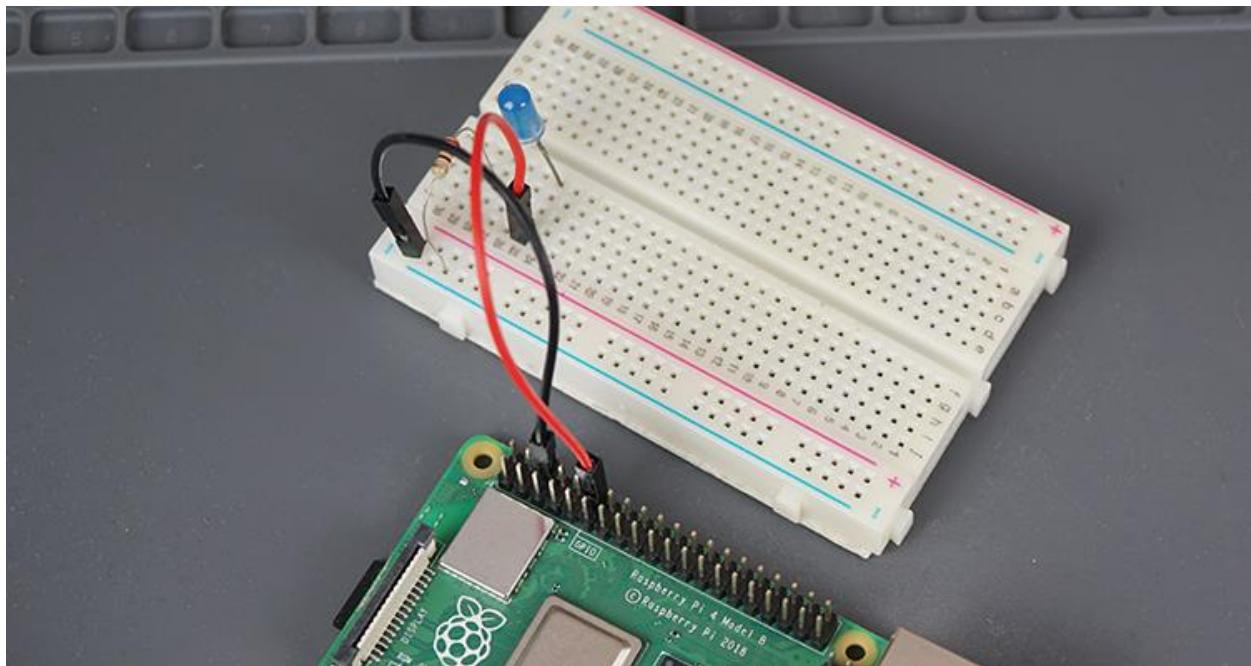
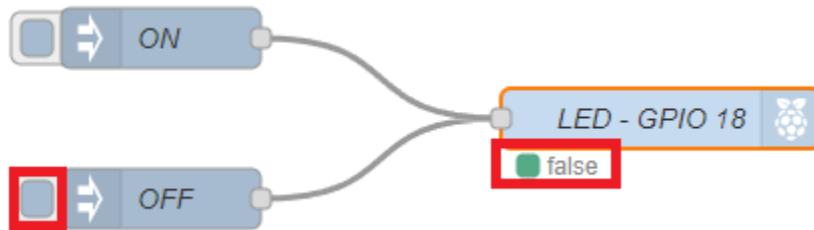
Now, let's test the application. Press the square next to the **ON** node to trigger it. A *true* will show up under the **LED – GPIO 18** node.



The LED attached to the Raspberry Pi should light up.



And when you press the **OFF** node, the LED turns off.



This shows how easy it is to access your Raspberry Pi GPIOs with Node-RED.

Creating a User Interface

Let's create a user interface to control the Raspberry Pi GPIO with a simple **switch** widget to turn the LED on and off.

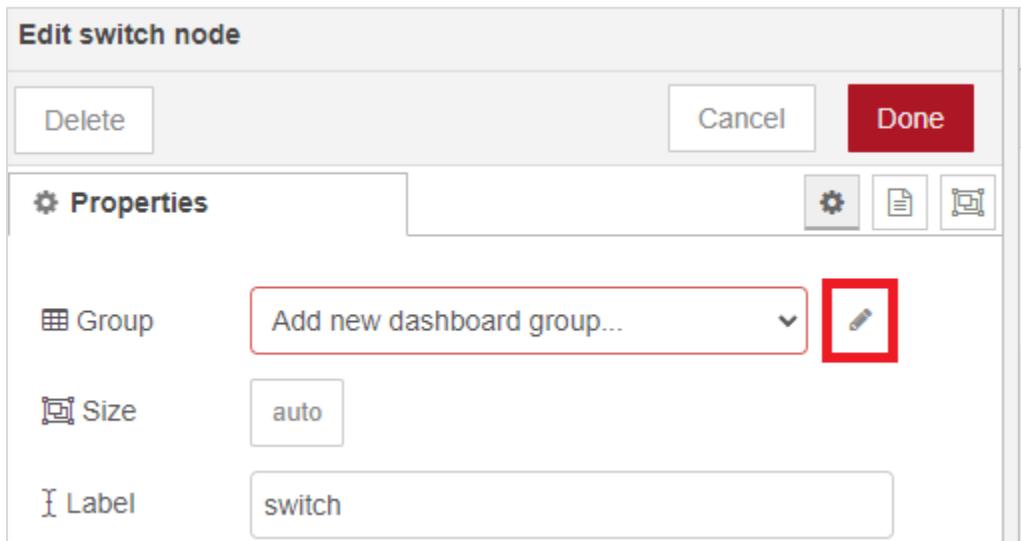
Drag a **switch** node (from the dashboard group) to the flow and wire it to the Raspberry Pi node.



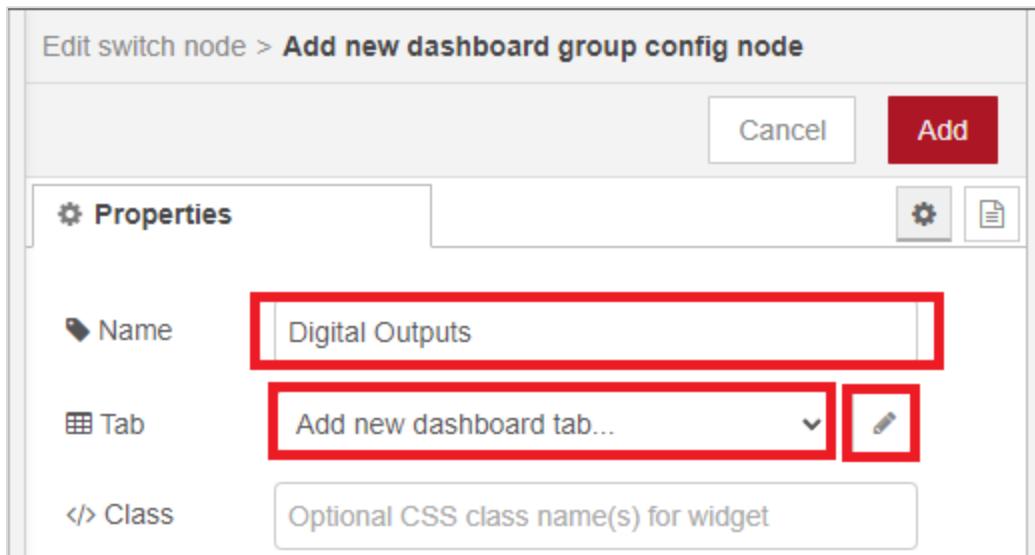
Now, double-click on the **switch** node to edit its properties.

As we've seen previously, each dashboard node must belong to a group within a tab. We'll create a group called **Digital Outputs** inside a tab called **Raspberry Pi**.

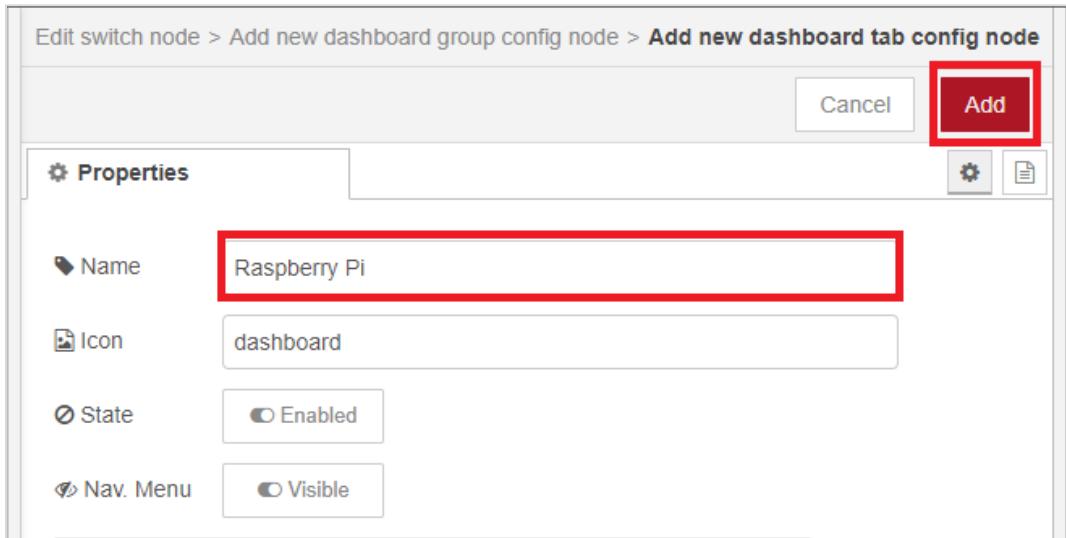
Double-click on the **switch** node. There's a field called **Group** with a little pencil icon. Select **Add new dashboard group...** and click on that pencil.



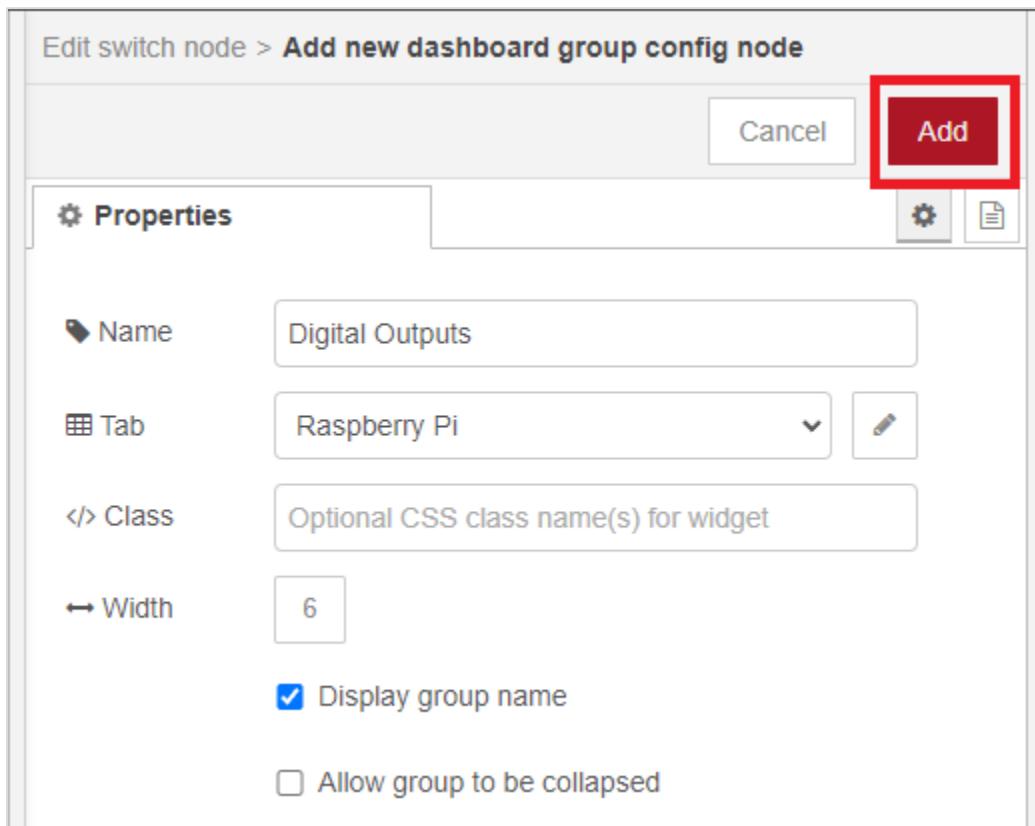
On the new window that opens, you can give a name to your group. We'll call it **Digital Outputs**. Then, on the **Tab** field, select **Add new dashboard tab...** and click on the pencil icon.



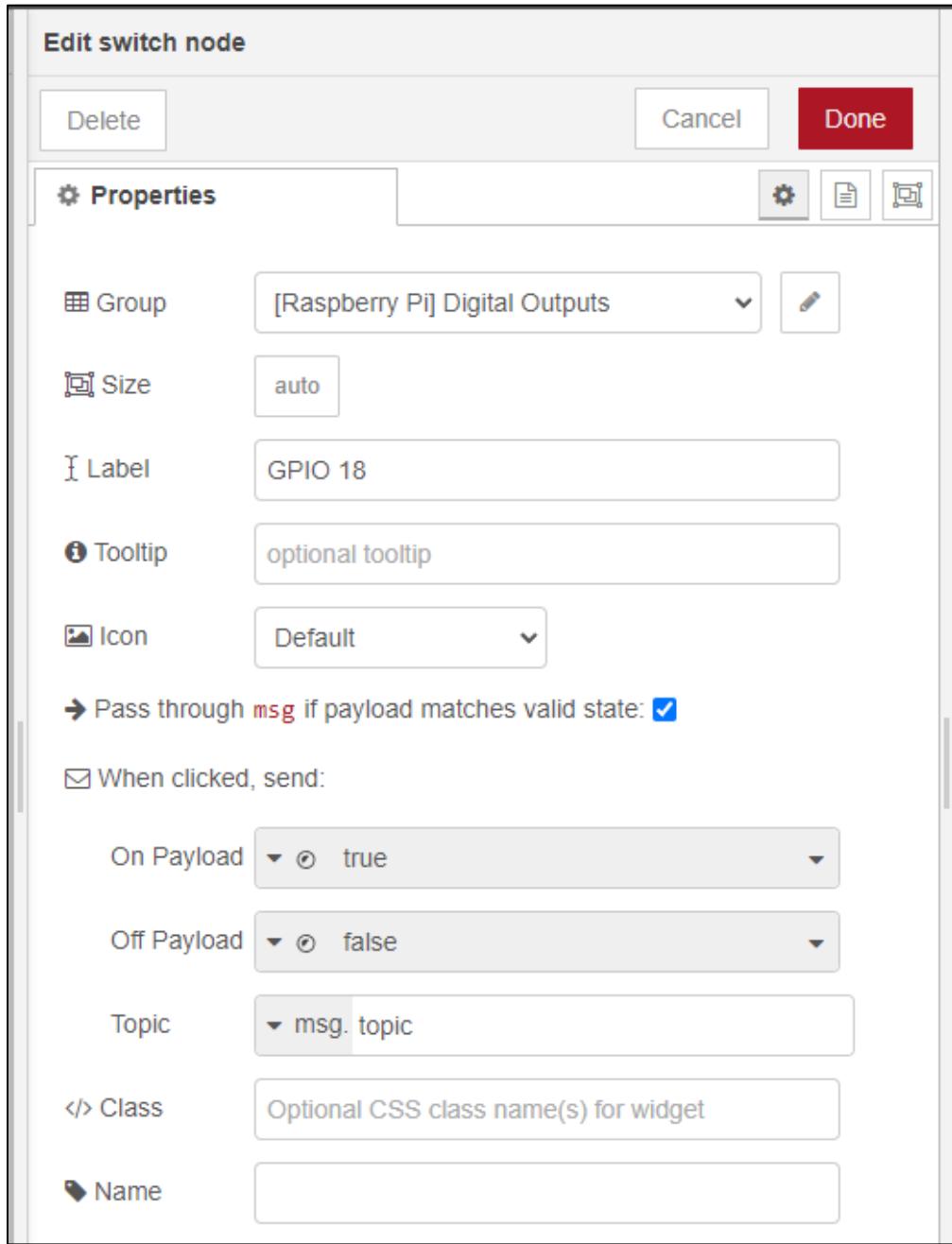
Give a name to your tab. We'll call it **Raspberry Pi**. Then, click **Add**.



Then, click **Add** again.

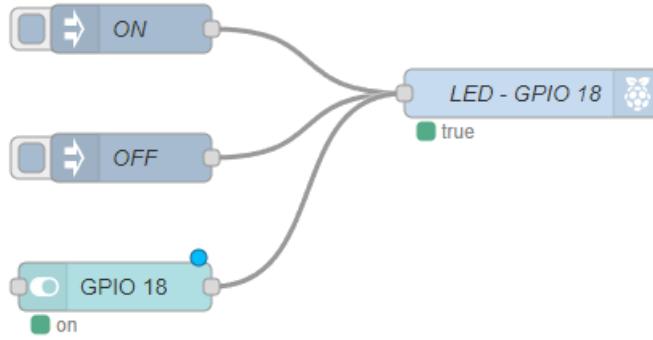


Write a **label** for the switch, we simply wrote **GPIO 18**. Then, set **true** for the **On Payload**, and **false** for the **Off Payload**.

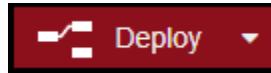


This means that when the switch is turned on, it will send a message with the **true** payload. When the switch is turned off, the message payload will be **false**.

Your flow should look as follows:



Deploy your application.

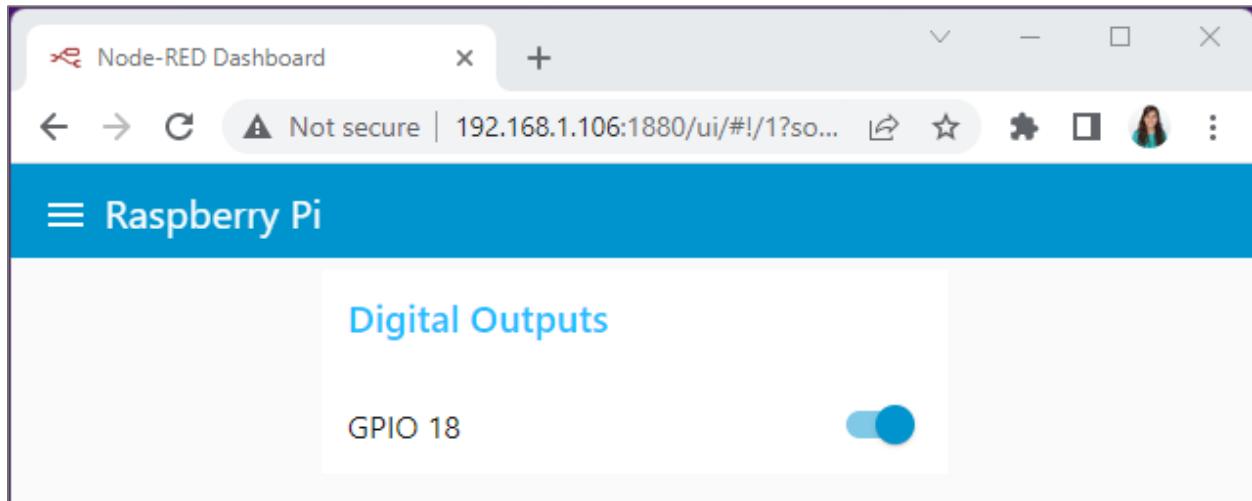


Accessing the User Interface

Now, access the user interface to control the Raspberry Pi GPIO using the switch. Go to the following URL:

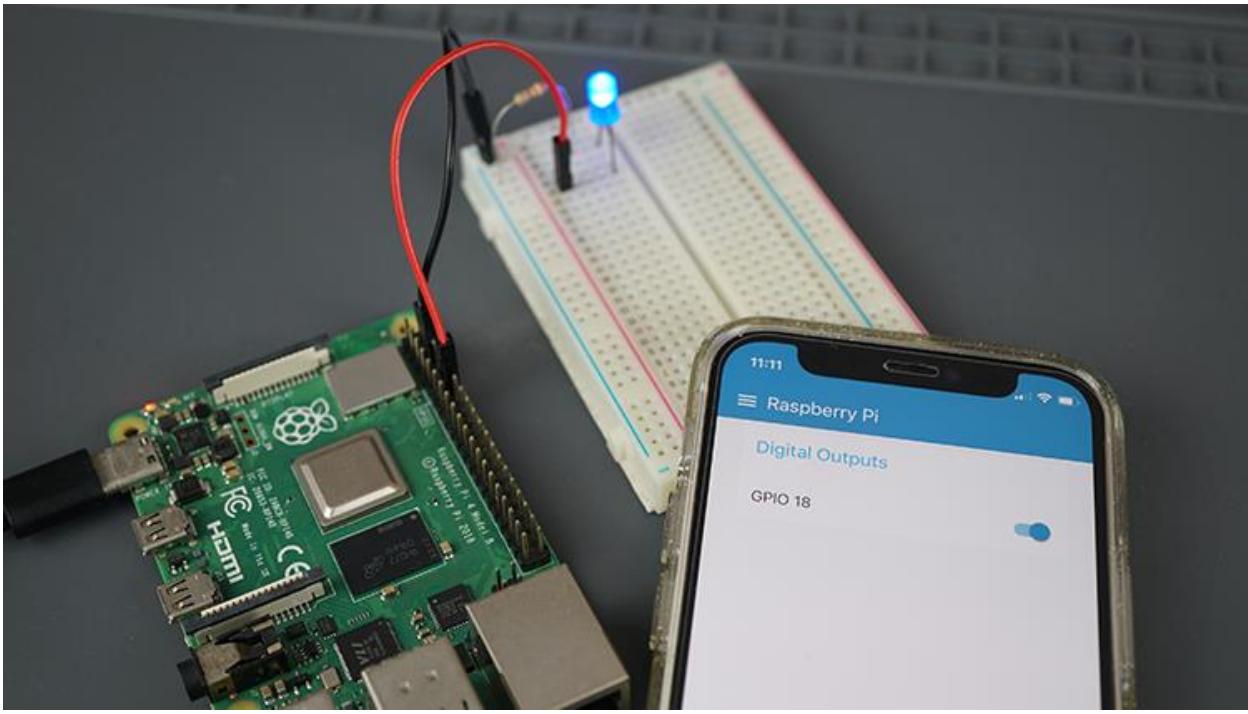
`http://Your_RPi_IP_address:1880/ui`

The user interface will look as follows:



Now, you can use that switch to turn the LED connected to GPIO 18 ON and OFF.

You can also access that Node-RED UI on your smartphone. The web page is mobile responsive.



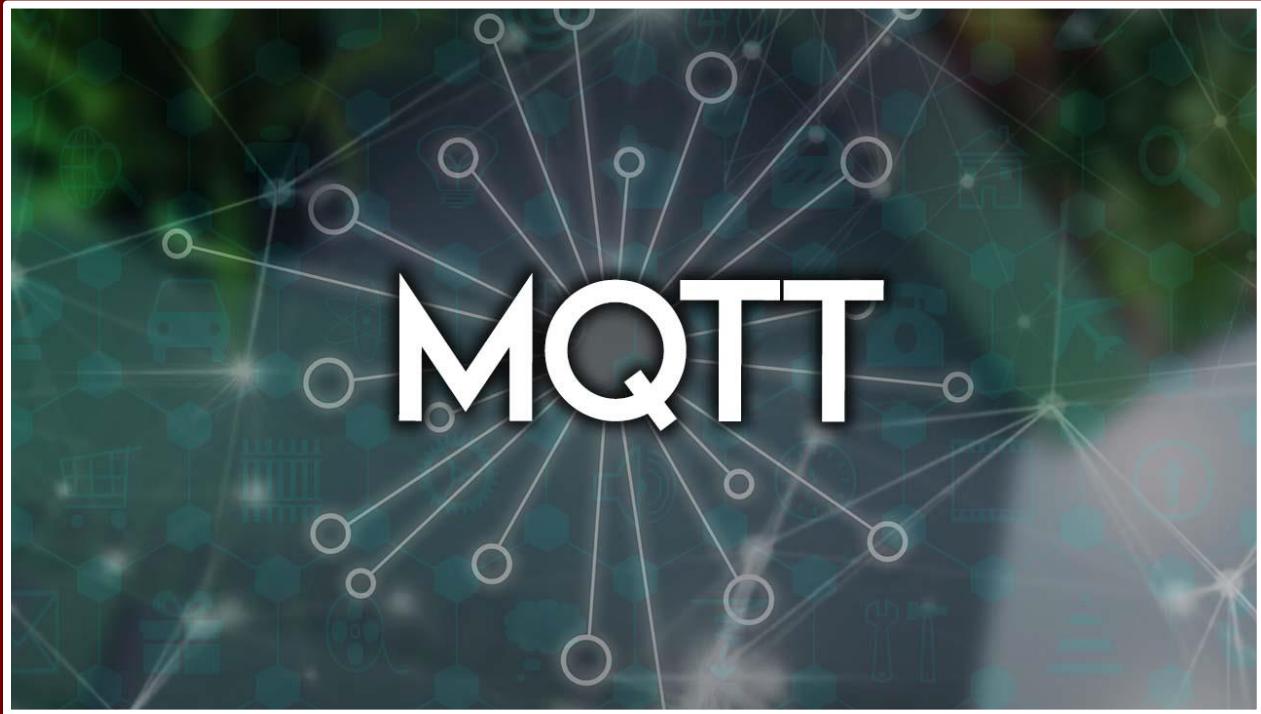
Taking it Further

Connect more LEDs to the Raspberry Pi on different GPIOs and create the corresponding flows to control them. With this simple exercise, you'll be more familiar with Node-RED, Node-RED dashboard, and its menus.

In the next module, we're going to explore the MQTT protocol to establish a communication between the ESP32/ESP8266 boards, the Raspberry Pi, and Node-RED.

MODULE 3

Getting Started with MQTT



This section is an introduction to the MQTT protocol. MQTT stands for Message Queuing Telemetry Transport, a simple messaging protocol suitable for communication between IoT devices. Learn MQTT basic concepts, install an MQTT broker and create a simple publish/subscribe flow.

3.1 - Introducing MQTT

MQTT stands for **M**essage **Q**ueuing **T**elemetry **T**ransport. MQTT is a simple messaging protocol, designed for constrained devices with low bandwidth. So, it's the perfect solution to exchange data between multiple IoT devices.

MQTT communication works as a publish and subscribe system. Devices publish messages on a specific topic. All devices that are subscribed to that topic receive the message.



Its main applications include sending messages to control outputs, reading and publishing data from sensor nodes, and much more.

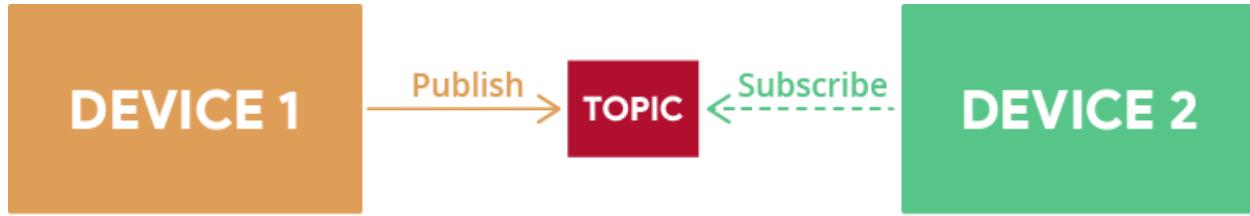
MQTT Basic Concepts

In MQTT there are a few basic concepts that you need to understand:

- Publish/Subscribe
- Messages
- Topics
- Broker

MQTT Publish/Subscribe

The first concept is the *publish/subscribe* system. In a publish and subscribe system, a device can publish a message on a topic, or it can be subscribed to a particular topic to receive messages



- 1) For example, Device 1 publishes on a topic.
- 2) Device 2 is subscribed to the same topic that Device 1 is publishing in.
- 3) So, Device 2 receives the message.

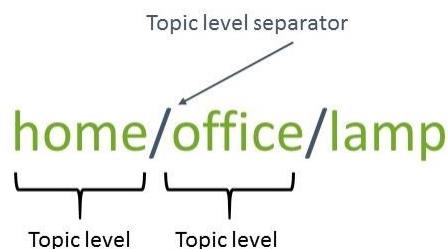
MQTT Messages

Messages are the information that you want to exchange between your devices. It can be a message like a command to control an output or data like sensor readings, for example.

MQTT Topics

Another important concept is the *topics*. Topics are the way you register interest in incoming messages or how you specify where you want to publish the message.

Topics are represented with strings separated by a forward slash. Each forward slash indicates a topic level. Here's an example of how you would create a topic for a lamp in your home office:



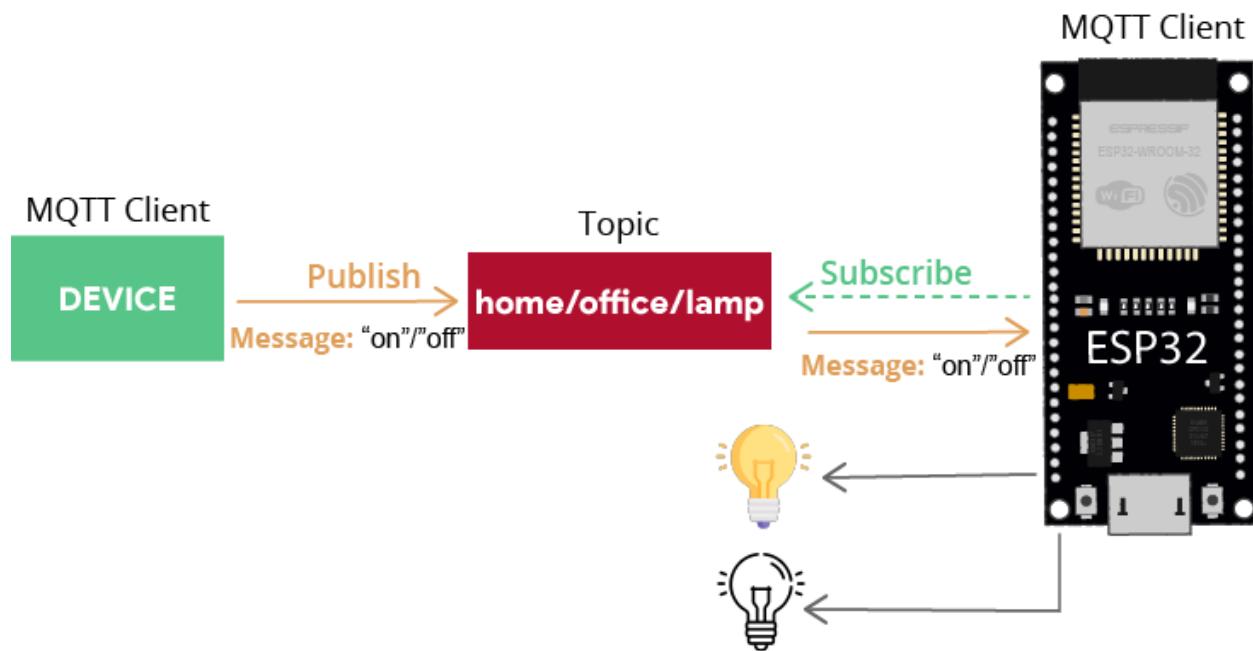
Note: topics are case-sensitive, which makes these two topics different:

home/office/lamp



home/office/LAmp

If you would like to turn on a lamp in your home office using MQTT you can imagine the following scenario:



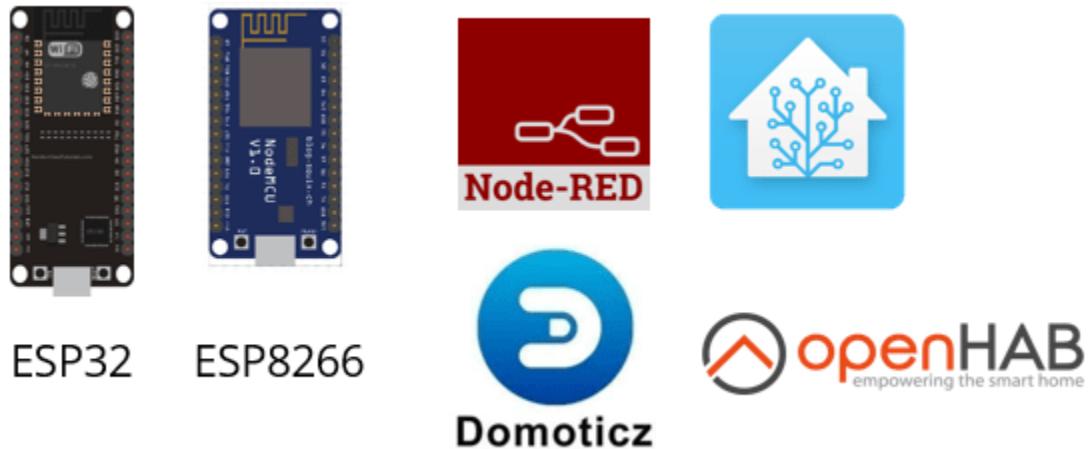
A device publishes “on” and “off” messages on the **home/office/lamp** topic.

You have a device that controls a lamp (it can be an ESP32, ESP8266, or any other board or device). The ESP32 that controls your lamp, is subscribed to that same topic: **home/office/lamp**.

So, when a new message is published on that topic, the ESP32 receives the “on” or “off” messages and turns the lamp on or off.

The device that is publishing the messages is another MQTT client, which can be an ESP32, an ESP8266, or a Home Automation controller platform with MQTT support

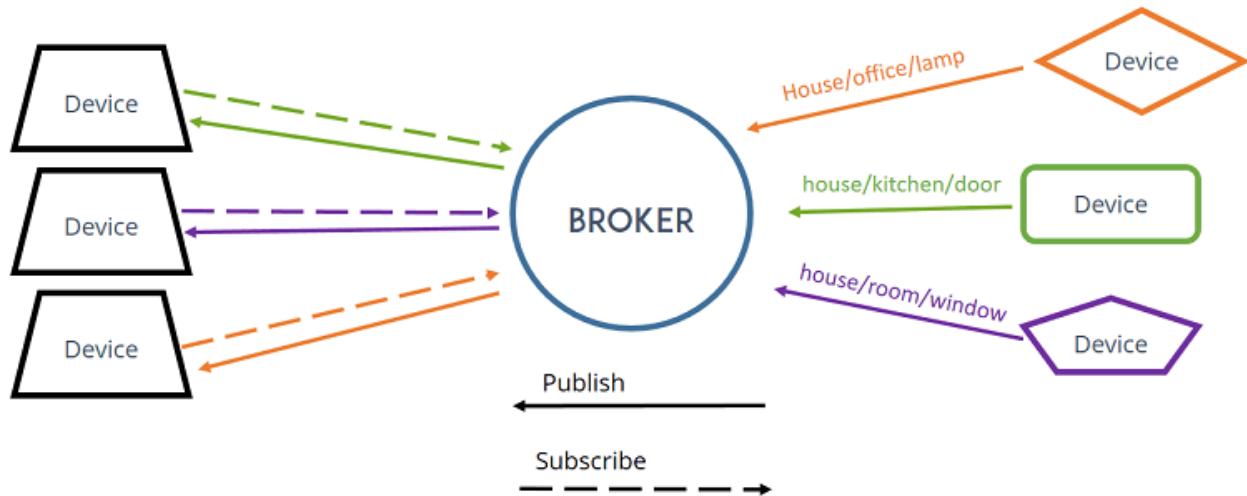
like Node-RED, Home Assistant, Domoticz, or OpenHAB, for example. In our case, we'll be using Node-RED.



MQTT Broker

Finally, another important concept is the *broker*.

The MQTT broker is responsible for receiving all messages, filtering the messages, deciding who is interested in them, and then publishing the message to all subscribed clients.



There are several brokers you can use. In home automation projects, we use the [Mosquitto Broker](#) installed on a Raspberry Pi. You can also install the Mosquitto

broker on your PC (which is not as convenient as using a Raspberry Pi board, because you have to keep your computer running all the time to keep the MQTT connection between your devices).



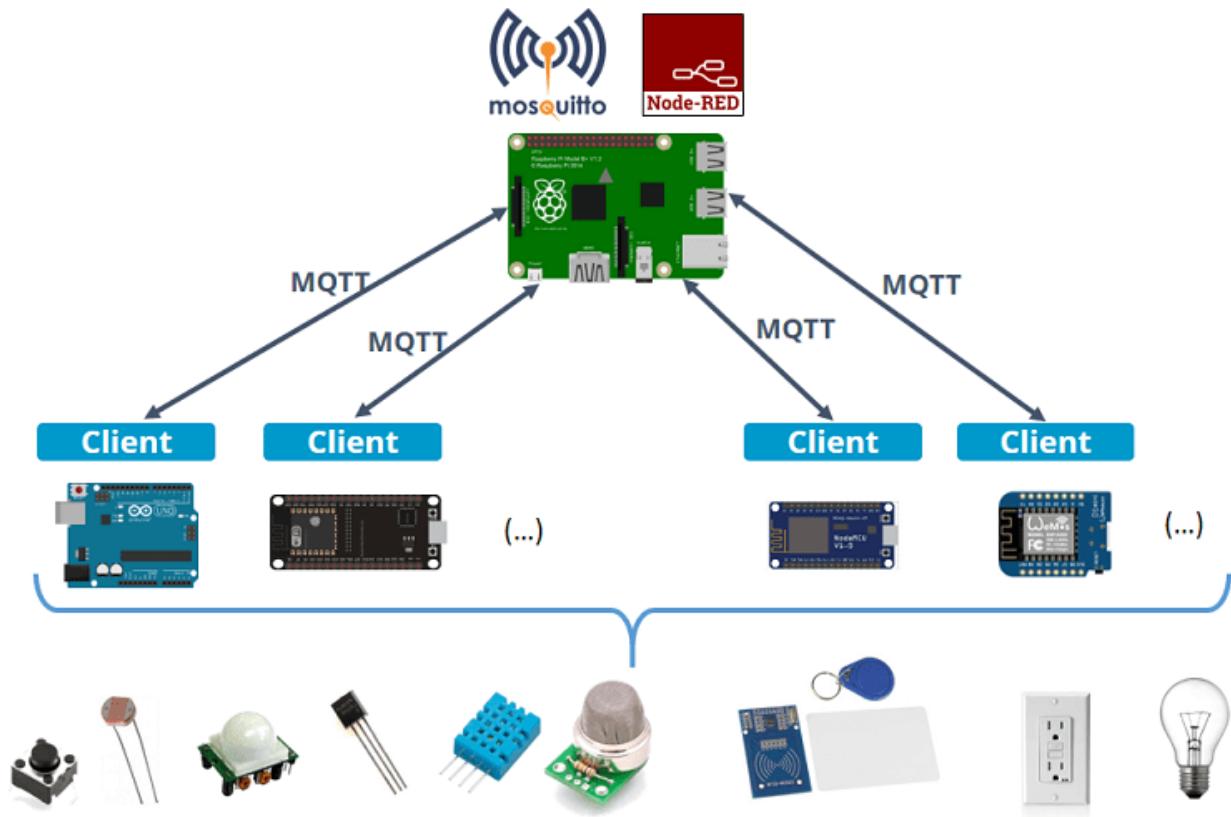
Having the Mosquitto broker installed on a Raspberry Pi on your local network allows you to exchange data between IoT devices that are also connected to that same network.

In the next unit, we'll show you how to install the Mosquitto broker on your Raspberry Pi board.

Alternatively, you can also run Mosquitto MQTT broker in the cloud. Running the MQTT Mosquitto Broker in the cloud allows you to connect several IoT devices from anywhere using different networks as long as they have an Internet connection. [Check unit 10.3.](#)

How to Use MQTT in Home Automation and IoT Projects

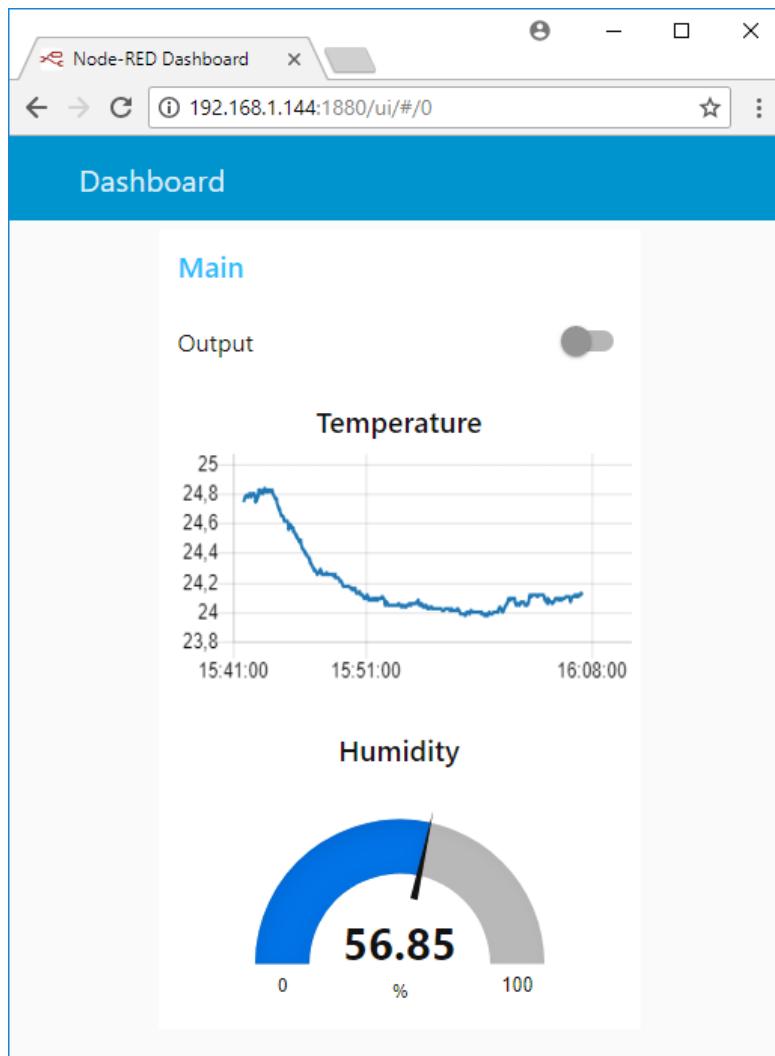
MQTT is great for home automation and internet of things projects. Here's an example of how it can be used in a Home Automation System built with low-cost development boards like a Raspberry Pi, ESP32, ESP8266, and Arduino.



- A Raspberry Pi runs the Mosquitto broker, which is essential for MQTT protocol.
- The same Raspberry Pi runs Node-RED, which is a Home Automation Platform with MQTT support—this means it can subscribe to topics to receive messages from the other IoT devices, and publish messages on specific topics to send messages to other devices.
- Node-RED also allows you to build a User Interface with buttons to control outputs and charts to display sensor readings.
- The Arduino, the ESP32, and ESP8266 can act as MQTT clients that publish and subscribe to topics.
- These boards are connected to actuators like LEDs or relays, and sensors like temperature, humidity, smoke, motion sensors, etc.

- These boards can publish data about the sensor's state on a specific topic, that Node-RED is also subscribed to. This way, Node-RED receives the sensor readings and can display them on the user interface.
- On the other side, Node-RED can publish data on a specific topic to control outputs when you use the buttons on the interface. The other boards are also subscribed to that topic and control the outputs accordingly.

The following image shows an example of a Node-RED UI that allows you to control one output and displays temperature and humidity readings:

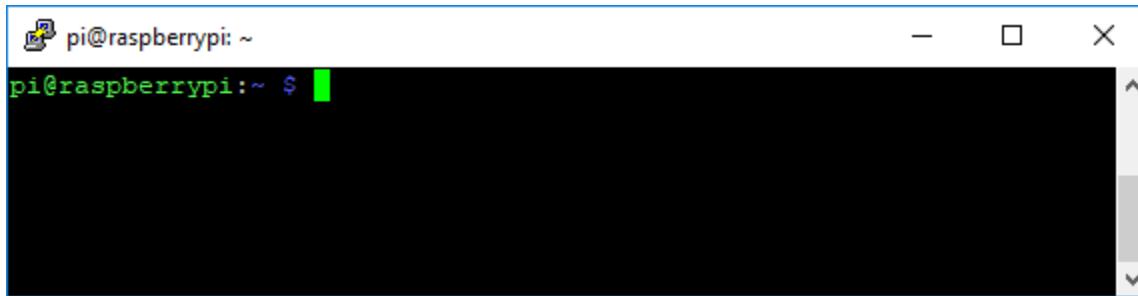


We'll build something similar in the next Modules.

3.2 - Installing Mosquitto Broker

In this section, you'll install the Mosquitto broker on the Raspberry Pi. Later, we'll test the installation to check if everything is working as expected.

- 1) Open a new SSH connection with your Raspberry Pi as you learned earlier.



- 2) Run the following command to upgrade and update your system:

```
sudo apt update && sudo apt upgrade
```

- 3) When asked, press **Y** and **Enter**. It will take some time to update and upgrade (in my case, it took approximately 10 minutes, but it may take longer).
- 4) To install the Mosquitto Broker and Mosquitto Clients, enter the next command:

```
sudo apt install -y mosquitto mosquitto-clients
```

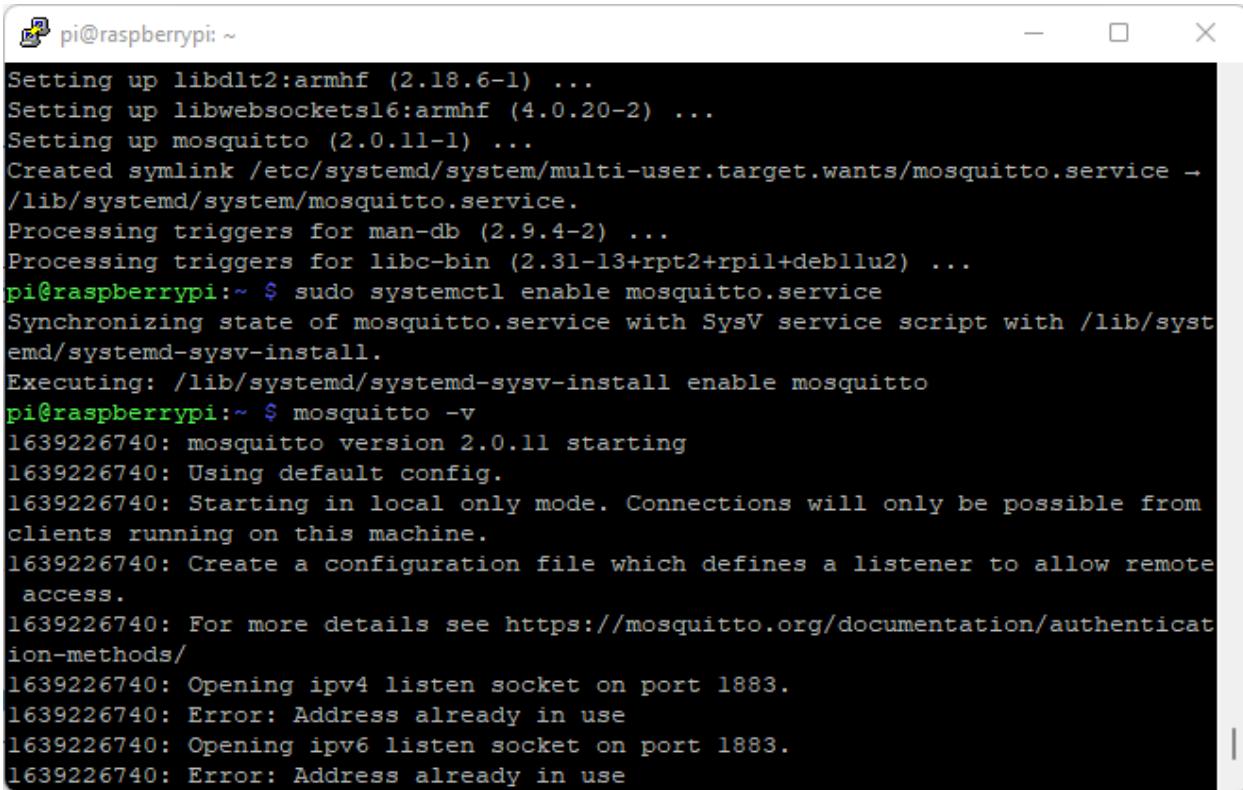
- 5) To make Mosquitto auto start when the Raspberry Pi boots, you need to run the following command (this means that the Mosquitto broker will automatically start when the Raspberry Pi starts):

```
sudo systemctl enable mosquitto.service
```

- 6) Now, test the installation by running the following command:

```
mosquitto -v
```

This returns the Mosquitto version that is currently running on your Raspberry Pi. It will be 2.0.11 or above.



```
pi@raspberrypi: ~
Setting up libdlt2:armhf (2.18.6-1) ...
Setting up libwebsockets16:armhf (4.0.20-2) ...
Setting up mosquitto (2.0.11-1) ...
Created symlink /etc/systemd/system/multi-user.target.wants/mosquitto.service →
/lib/systemd/system/mosquitto.service.
Processing triggers for man-db (2.9.4-2) ...
Processing triggers for libc-bin (2.31-13+rpt2+rpil+deb11u2) ...
pi@raspberrypi:~ $ sudo systemctl enable mosquitto.service
Synchronizing state of mosquitto.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mosquitto
pi@raspberrypi:~ $ mosquitto -v
1639226740: mosquitto version 2.0.11 starting
1639226740: Using default config.
1639226740: Starting in local only mode. Connections will only be possible from
clients running on this machine.
1639226740: Create a configuration file which defines a listener to allow remote
access.
1639226740: For more details see https://mosquitto.org/documentation/authentication-methods/
1639226740: Opening ipv4 listen socket on port 1883.
1639226740: Error: Address already in use
1639226740: Opening ipv6 listen socket on port 1883.
1639226740: Error: Address already in use
```

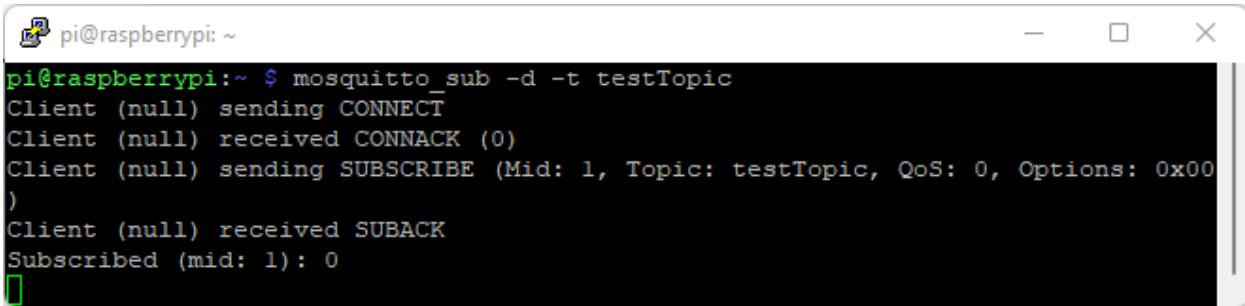
It will prompt the following message: “*Starting in local only mode. Connections will only be possible from clients running on this machine. Create a configuration file which defines a listener to allow remote access.*”

This means that by default, you can’t communicate with the Mosquitto broker from another device. We don’t want this to happen, we want to be able to access using the ESP32 and/or ESP8266 boards. We’ll take a look at this in a few moments.

Testing MQTT Broker Installation

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

```
mosquitto_sub -d -t testTopic
```

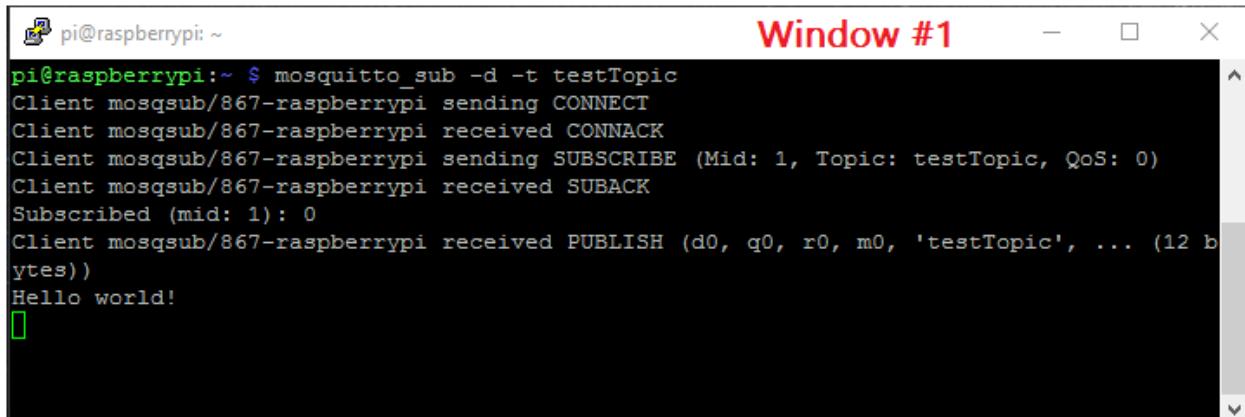


```
pi@raspberrypi:~ $ mosquitto_sub -d -t testTopic
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending SUBSCRIBE (Mid: 1, Topic: testTopic, QoS: 0, Options: 0x00)
Client (null) received SUBACK
Subscribed (mid: 1): 0
```

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

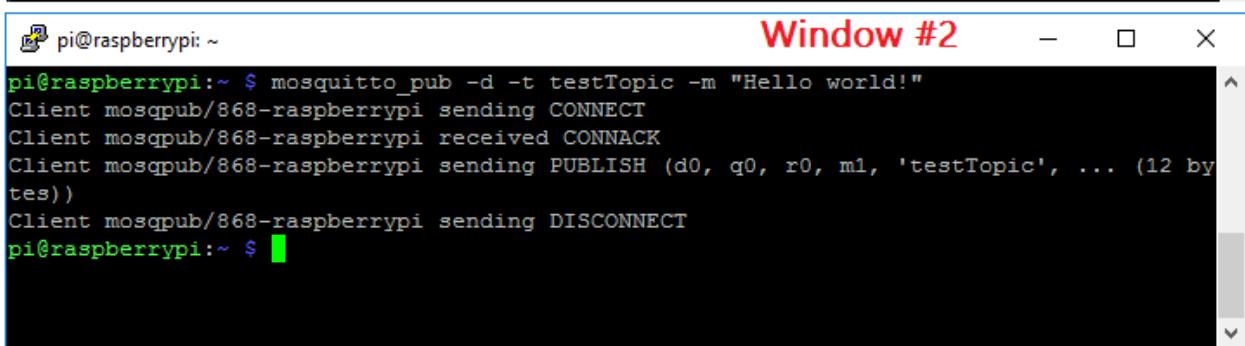
To publish a sample message to **testTopic**, open a terminal Window #2 (a new SSH connection with your Pi) and run the following command:

```
mosquitto_pub -d -t testTopic -m "Hello world!"
```



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!
```



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. This means everything is working as expected.

Enable Remote Access/Authentication

To enable remote access so that we can communicate with other IoT devices, we need to edit/create a configuration file. You can edit the configuration file with one of the following options:

- No authentication;
- Authentication with user and password.

We'll add authentication with user and password.

- 1) Run the following command, but replace `YOUR_USERNAME` with the username you want to use:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd YOUR_USERNAME
```

I'll be using the MQTT user `sara`, so I run the command as follows:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd sara
```

When you run the preceding command with the desired username, you'll be asked to enter a password. No characters will be displayed while you enter the password. Enter the password and memorize the user/pass combination, you'll need it later in your projects to make a connection with the broker.

This previous command creates a password file called `passwd` on the `/etc/mosquitto` directory.

Now, we need to edit the mosquito configuration file so that it only allows authentication with the username and password we've defined.

- 2) Run the following command to edit the configuration file:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

- 3) Add the following line at the top of the file (make sure it is at the top of the file, otherwise it won't work):

```
per_listener_settings true
```

- 4) Add the following three lines to allow connection for authenticated users and tell Mosquitto where the username/password file is located.

```
allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd
```

Your configuration file will look as follows (the new lines are in bold):

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

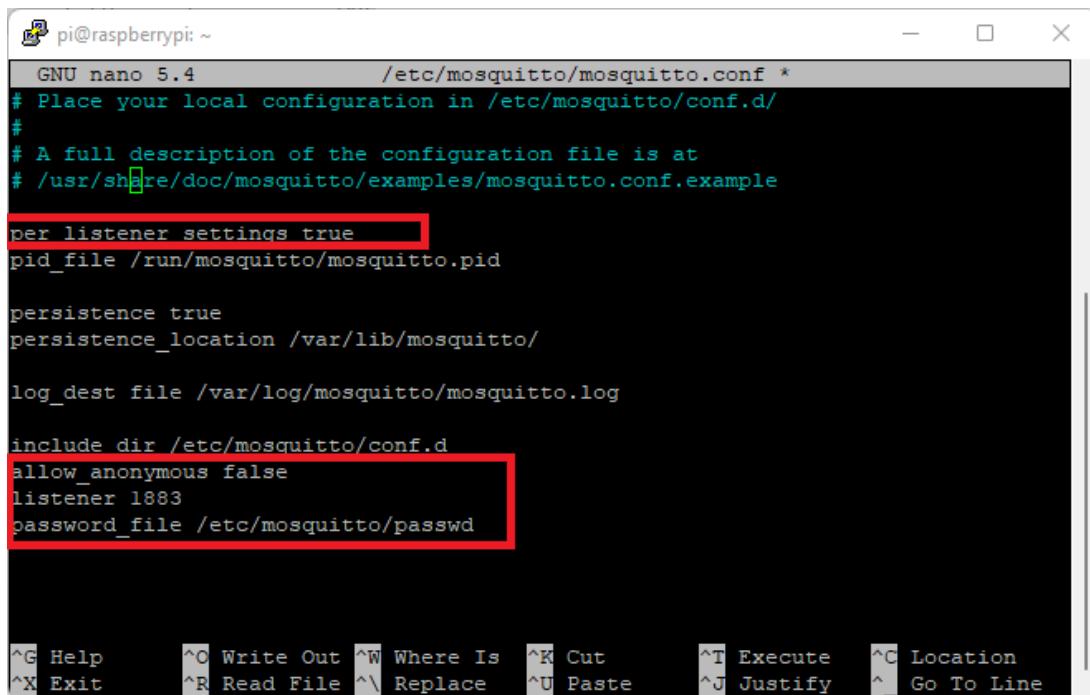
per_listener_settings true

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d
allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd
```



The screenshot shows a terminal window titled "pi@raspberrypi: ~". It displays the contents of the "/etc/mosquitto/mosquitto.conf" file using the GNU nano 5.4 editor. The file includes comments about its location and a full description. The new configuration settings are highlighted with red boxes:

```
GNU nano 5.4          /etc/mosquitto/mosquitto.conf *
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

per_listener_settings true
pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d
allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd
```

The terminal also shows a menu bar with standard nano key bindings at the bottom:

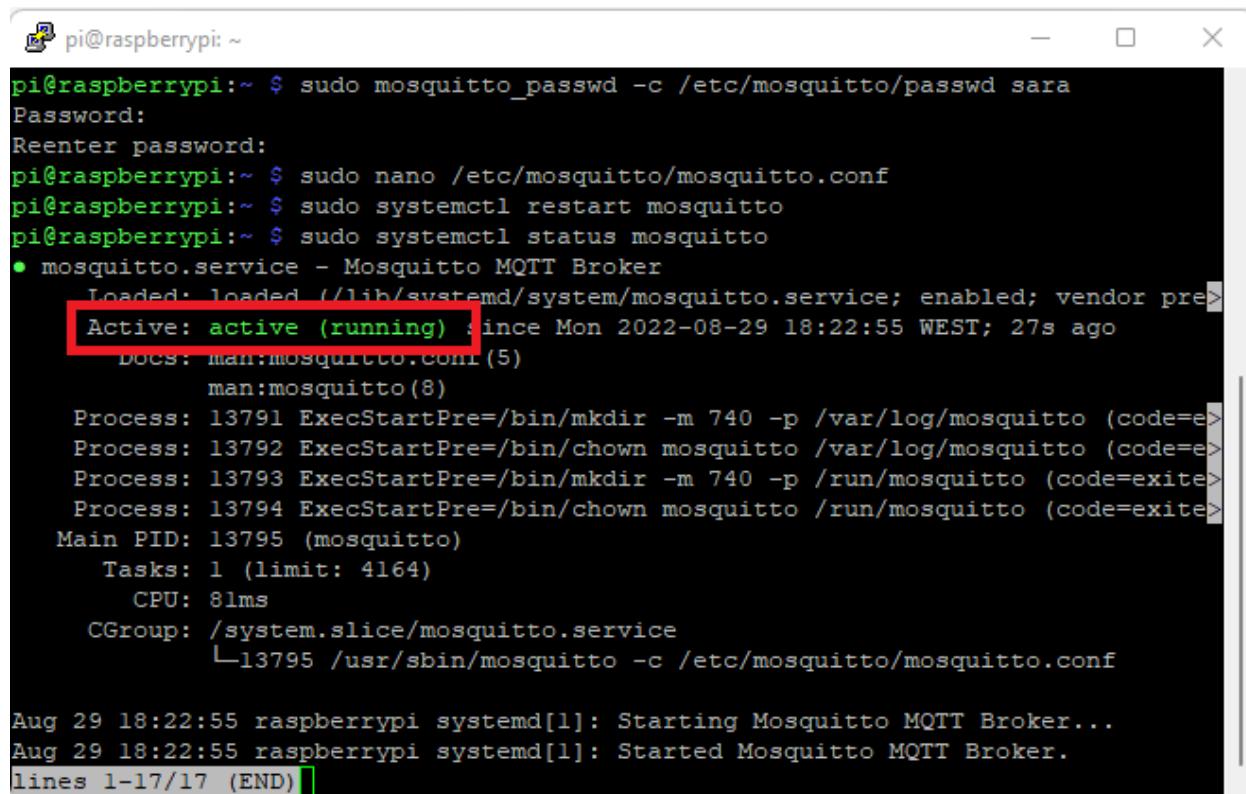
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^ Go To Line

- 7) Press **CTRL-X**, then **Y**, and finally press **Enter** to exit and save the changes.
- 8) Restart Mosquitto for the changes to take effect.

```
sudo systemctl restart mosquitto
```

Wait a few seconds. Then to check if Mosquitto is running, you can type the following command:

```
sudo systemctl status mosquitto
```



The screenshot shows a terminal window titled 'pi@raspberrypi: ~'. The command 'sudo systemctl status mosquitto' is run, displaying the service status. A red box highlights the 'Active: active (running)' line, indicating the service is currently active and running. The output also shows the service was started 27 seconds ago and lists several processes associated with it.

```
pi@raspberrypi:~ $ sudo mosquitto_passwd -c /etc/mosquitto/passwd sara
Password:
Reenter password:
pi@raspberrypi:~ $ sudo nano /etc/mosquitto/mosquitto.conf
pi@raspberrypi:~ $ sudo systemctl restart mosquitto
pi@raspberrypi:~ $ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor pre>
   Active: active (running) since Mon 2022-08-29 18:22:55 WEST; 27s ago
     Docs: man:mosquitto(5)
           man:mosquitto(8)
     Process: 13791 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=e>
     Process: 13792 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=e>
     Process: 13793 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exit>
     Process: 13794 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exit>
   Main PID: 13795 (mosquitto)
      Tasks: 1 (limit: 4164)
        CPU: 81ms
      CGroup: /system.slice/mosquitto.service
              └─13795 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Aug 29 18:22:55 raspberrypi systemd[1]: Starting Mosquitto MQTT Broker...
Aug 29 18:22:55 raspberrypi systemd[1]: Started Mosquitto MQTT Broker.
lines 1-17/17 (END)
```

Now, you have authentication with username and password enabled.

If you try to publish messages using the same command we used before, it won't work because we need to provide a user and password. We'll see how to do that soon.

Remember that every time you want to communicate with the broker, you'll need to provide the username and password.

Add More Users/Change Password

To add more users to an existing password file, or to change the password for an existing user, leave out the `-c` argument as follows:

```
mosquitto_passwd <password file> <username>
```

For example, if I want to change the password for the `sara` user and taking into account that the password file we created was called `passwd`, the command will be as follows:

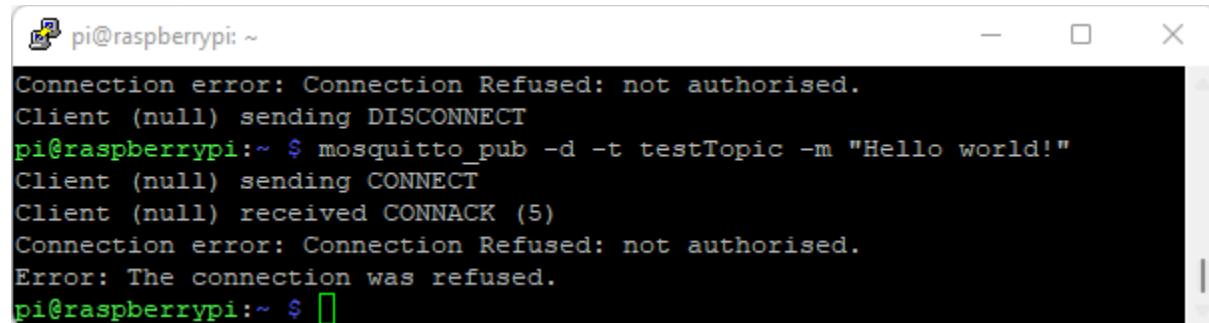
```
sudo mosquitto_passwd /etc/mosquitto/passwd sara
```

Testing MQTT Broker Installation (User and Password)

Now, if you try the commands you've tested previously like the following:

```
mosquitto_pub -d -t testTopic -m "Hello world!"
```

You should get a message as follows "Connection Refused not authorised".



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows the following text output:

```
Connection error: Connection Refused: not authorised.
Client (null) sending DISCONNECT
pi@raspberrypi:~ $ mosquitto_pub -d -t testTopic -m "Hello world!"
Client (null) sending CONNECT
Client (null) received CONNACK (5)
Connection error: Connection Refused: not authorised.
Error: The connection was refused.
pi@raspberrypi:~ $
```

You need to pass the user and password to the command, as follows:

```
mosquitto_pub -d -t testTopic -m "Hello world!" -u user -P pass
```

Replace `user` with your username and `pass` with your password.

This time, you should publish the message successfully.

```
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q0, r0, m1, 'testTopic', ... (12 bytes
))
Client (null) sending DISCONNECT
```

This means that the Mosquitto broker with user/password authentication is set up properly.

To subscribe to an MQTT topic, you also need to pass the username/password. For example, enter the command:

```
mosquitto_sub -d -t testTopic -u user -P pass
```

You can proceed to the next unit.

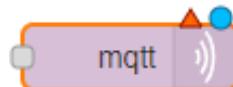
3.3 - MQTT with Node-RED

In this unit, we're going to establish an MQTT communication using Node-RED nodes. You'll learn how to subscribe and publish to topics using Node-RED MQTT nodes.

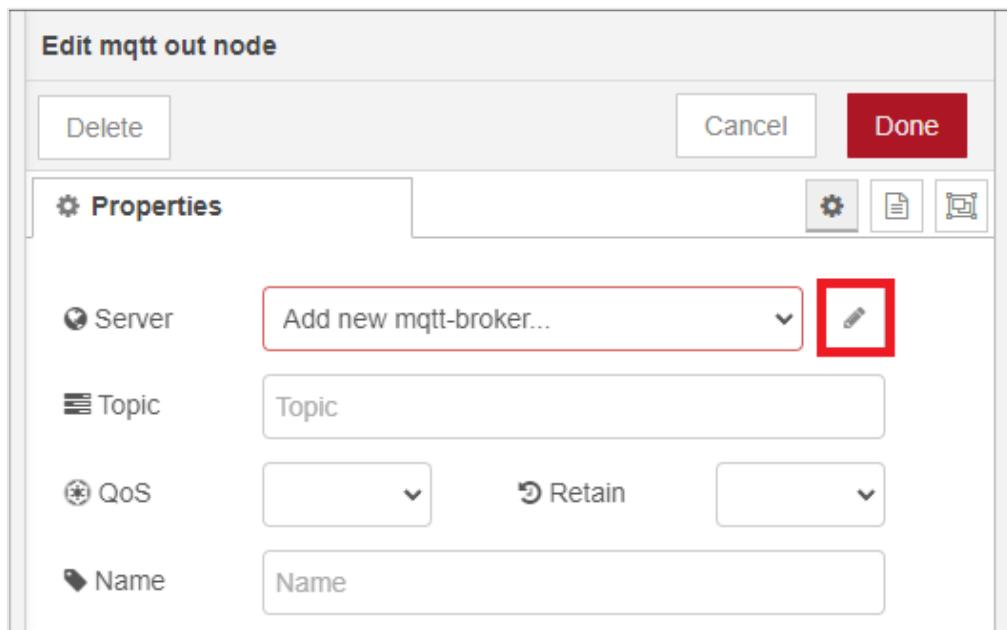
We'll create a publish and a subscribe flow. The publish flow publishes data on a topic. The subscribe flow is subscribed to that same topic to receive the messages.

Creating the *Publish* Flow

First, drag an **MQTT output** node to the flow.



You need to connect Node-RED to your MQTT broker. Double-click the MQTT output node. On the **Server** field, select **Add new mqtt-broker** and click on the pencil icon to add your MQTT broker configurations.



The following window opens.

Properties

Connection **Security** **Messages**

Name: Name

Server: localhost **Port**: 1883

Connect automatically

Use TLS

Protocol: MQTT V3.1.1

Client ID: Leave blank for auto generated

Keep Alive: 60

Session: Use clean session

Type **localhost** in the **Server** field and all the other settings are configured properly by default. We set **localhost** because the Mosquitto broker is running on the same machine that Node-RED is hosted.

Then, click on the **Security** tab to add the user and password combination you set when installing the MQTT broker.

Edit mqtt out node > **Edit mqtt-broker node**

Delete **Cancel** **Update**

Properties

Name: Name

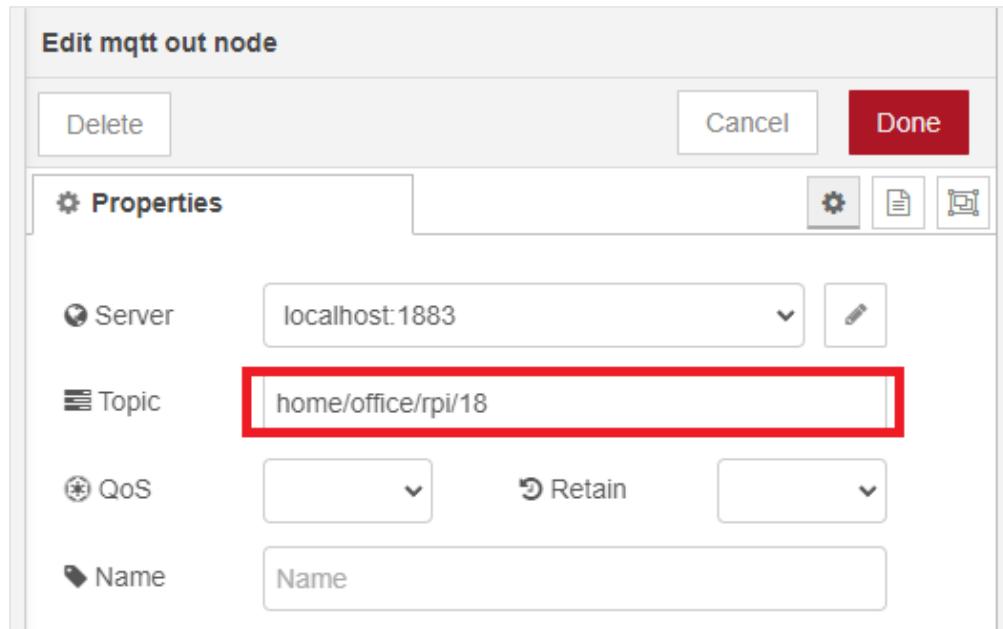
Connection **Security** **Messages**

Username: sara

Password:

When it's done, click **Update/Add**. The node will automatically connect to the Mosquitto broker.

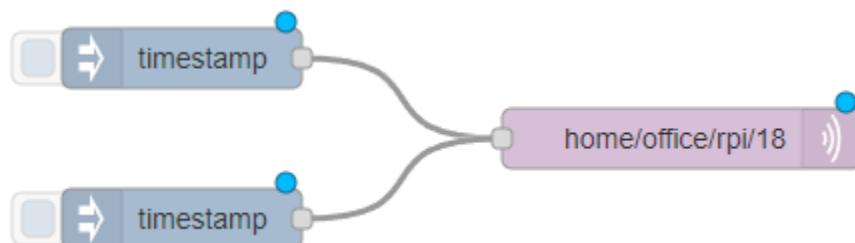
Now, let's imagine you would like to control the Raspberry Pi GPIO 18, and your RPi is located in your home office. A good choice for a topic that controls GPIO 18 would be **home/office/rpi/18**.



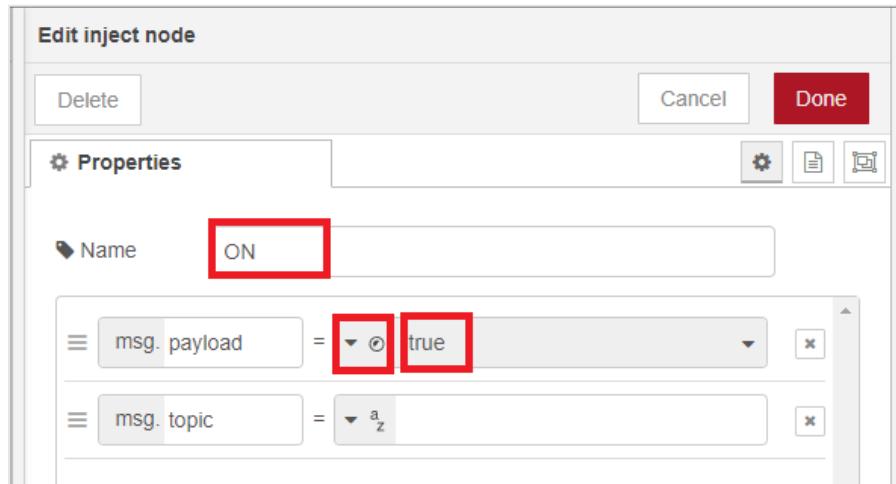
Press **Done** to save your node.

There are only two messages that make sense to publish to any device that is subscribed to this topic. Those messages can be **true (1)** or **false (0)**. This turns the Raspberry Pi pin either on or off.

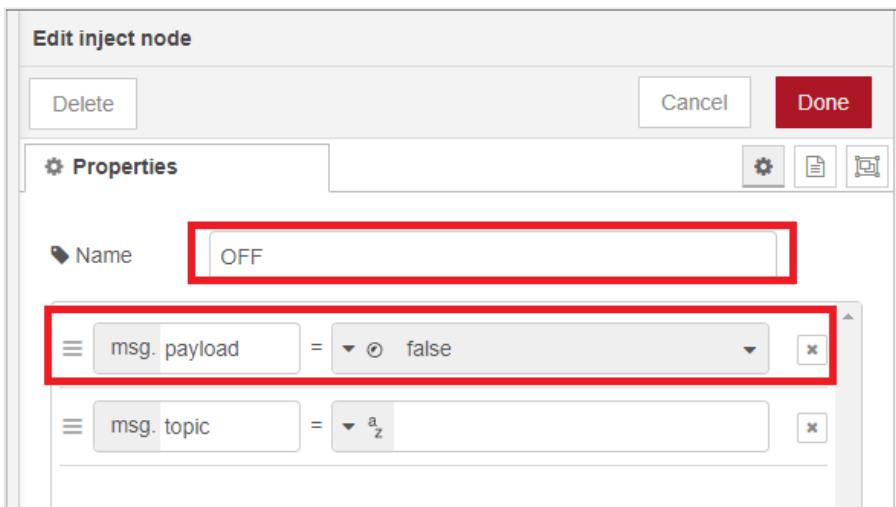
Drag two **Inject** nodes to your flow and connect them to the MQTT node.



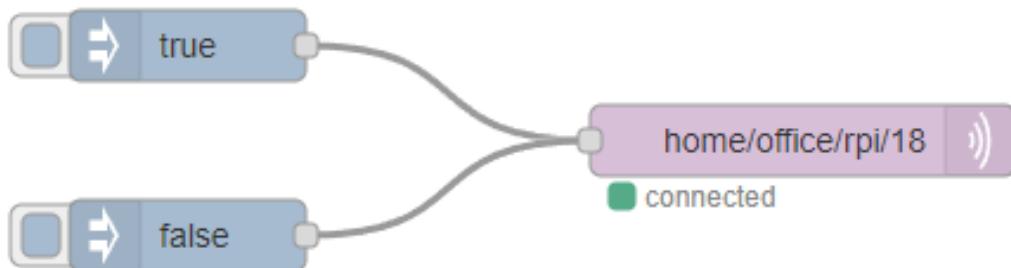
Double-click the first **Inject** node to edit its properties. Select **boolean** and type **true**. Press **Done**.



Edit the other **Inject** node. Select **boolean** and type **false**. Press **Done**.



Deploy your flow. You should see the **connected** message below the MQTT node.



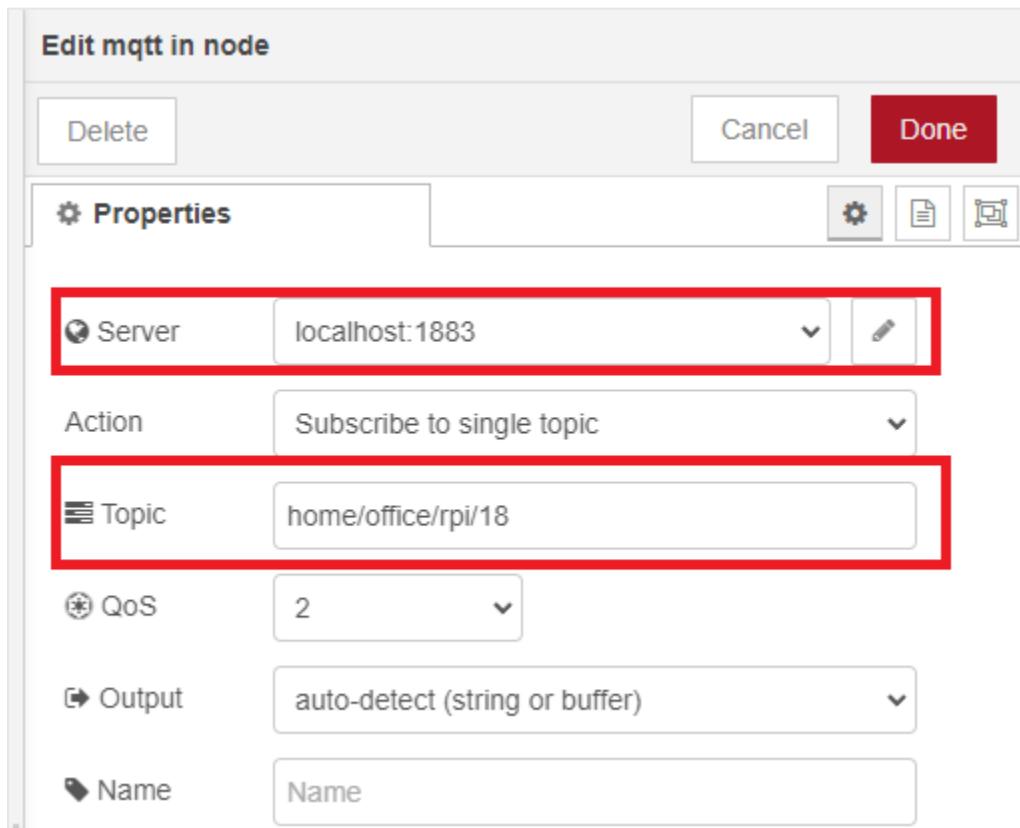
Creating the *Subscribe* Flow

We need a node that is subscribed to this exact topic, so it receives the message and ultimately does something.

Drag an **MQTT input** node to the flow.



Select the MQTT broker you created previously. Type the same topic we added previously **home/office/rpi/18**. Make sure the **Action** is set to **Subscribe to single topic**.

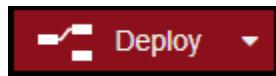


Drag a **Debug** node, so we can print the received message. Connect both nodes.

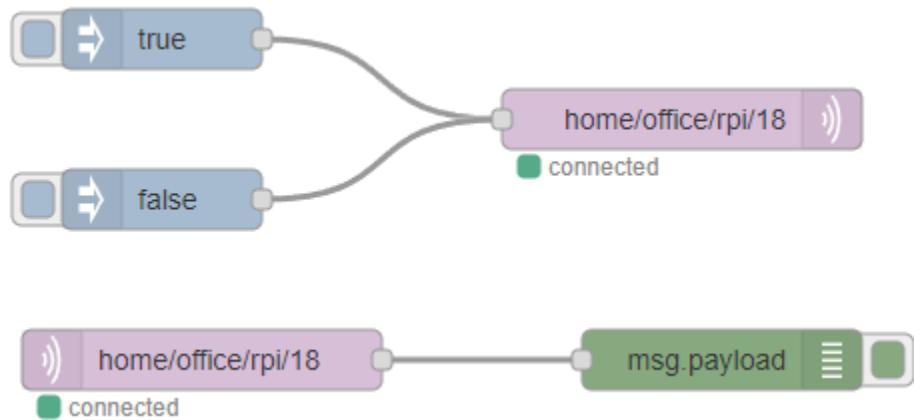


Testing the MQTT Connection

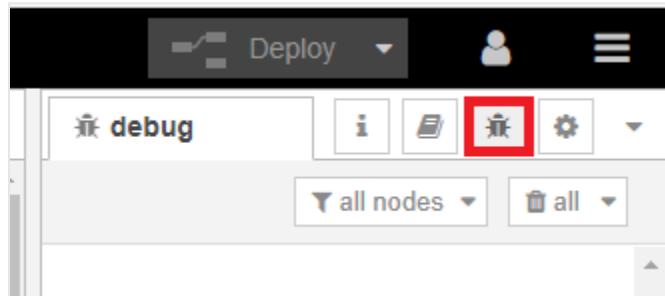
Press the **Deploy** button on the top-right corner to save your application.



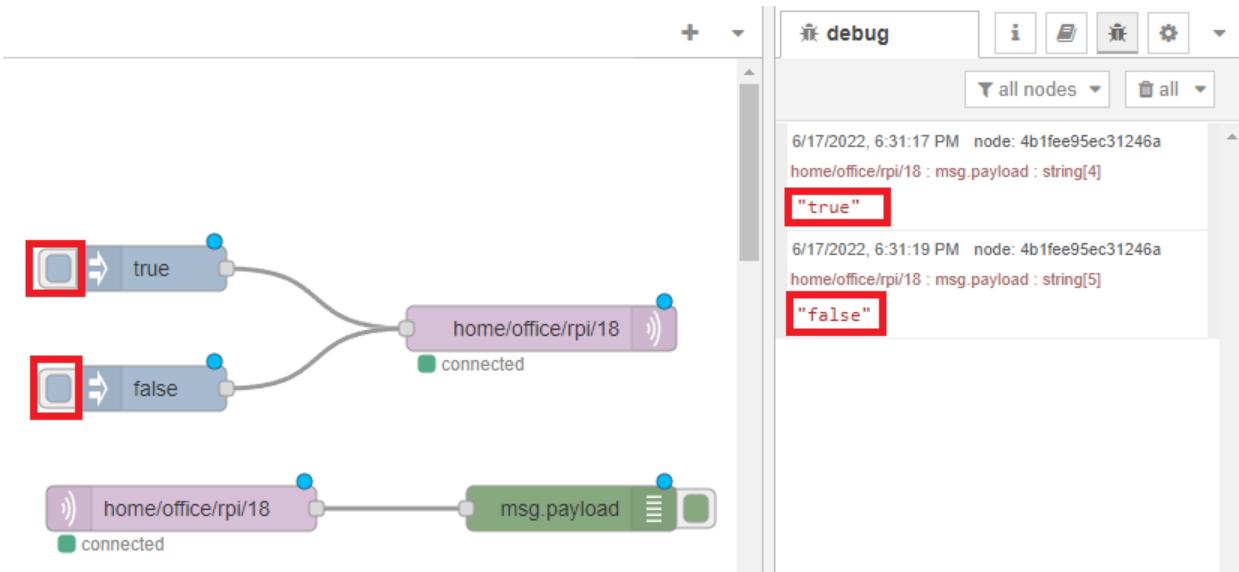
Your flow should look as follows with the **connected** message below the MQTT nodes.



Open the **debug** window—click on the bug icon on the right sidebar.



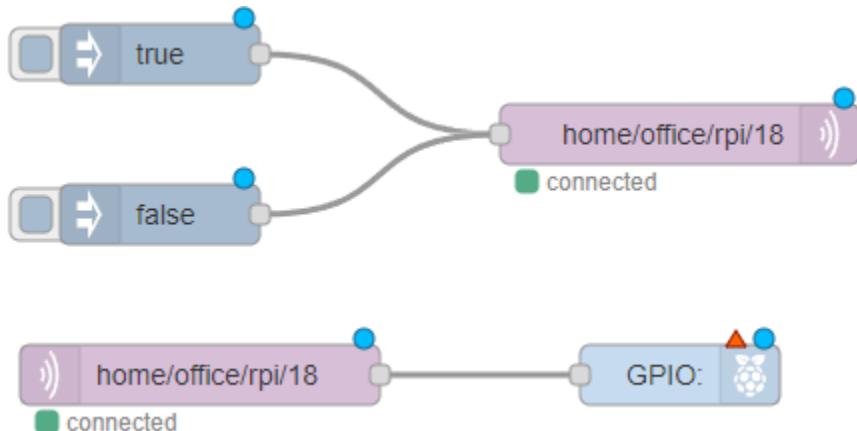
Trigger the **Inject** nodes. As you can see the corresponding message is being received and printed in the **debug** window.



This is how you make a publish and subscribe system.

Connecting the *rpi gpio out* node

Now, delete the **debug** node. Drag a **rpi-gpio out** node and wire it to the MQTT input node as shown below.



Lastly, edit the **rpi-gpio out** node. Set the type as a **Digital output** and **GPIO 18** (that's where your LED will be connected to). Click **Done**.

Edit rpi-gpio out node

[Delete](#) [Cancel](#) [Done](#)

Properties

Pin

| | | |
|--------------------|----------------------------------|-------------------|
| 3.3V Power - 1 | <input type="radio"/> | 2 - 5V Power |
| SDA1 - GPIO02 - 3 | <input type="radio"/> | 4 - 5V Power |
| SCL1 - GPIO03 - 5 | <input type="radio"/> | 6 - Ground |
| GPIO04 - 7 | <input type="radio"/> | 8 - GPIO14 - TxD |
| Ground - 9 | <input type="radio"/> | 10 - GPIO15 - RxD |
| GPIO17 - 11 | <input checked="" type="radio"/> | 12 - GPIO18 |
| GPIO27 - 13 | <input type="radio"/> | 14 - Ground |
| GPIO22 - 15 | <input type="radio"/> | 16 - GPIO23 |
| 3.3V Power - 17 | <input type="radio"/> | 18 - GPIO24 |
| MOSI - GPIO10 - 19 | <input type="radio"/> | 20 - Ground |
| MISO - GPIO09 - 21 | <input type="radio"/> | 22 - GPIO25 |
| SCLK - GPIO11 - 23 | <input type="radio"/> | 24 - GPIO8 - CE0 |
| Ground - 25 | <input type="radio"/> | 26 - GPIO7 - CE1 |
| SD - 27 | <input type="radio"/> | 28 - SC |
| GPIO05 - 29 | <input type="radio"/> | 30 - Ground |
| GPIO06 - 31 | <input type="radio"/> | 32 - GPIO12 |
| GPIO13 - 33 | <input type="radio"/> | 34 - Ground |
| GPIO19 - 35 | <input type="radio"/> | 36 - GPIO16 |
| GPIO26 - 37 | <input type="radio"/> | 38 - GPIO20 |
| Ground - 39 | <input type="radio"/> | 40 - GPIO21 |

BCM GPIO: 18

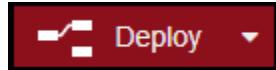
Type: Digital output

Initialise pin state?

Name: LED - GPIO 18

Enabled

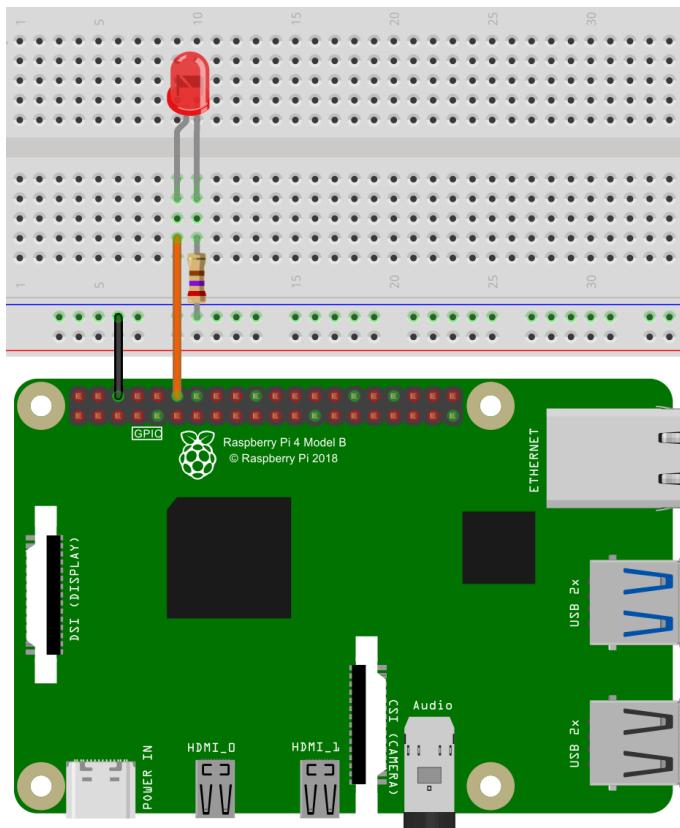
Deploy your application.



Wiring the Circuit

Here's the hardware required to complete this example (same as in the previous module):

- LED (5mm)
- 270Ω resistor or similar
- Breadboard
- Jumper wires



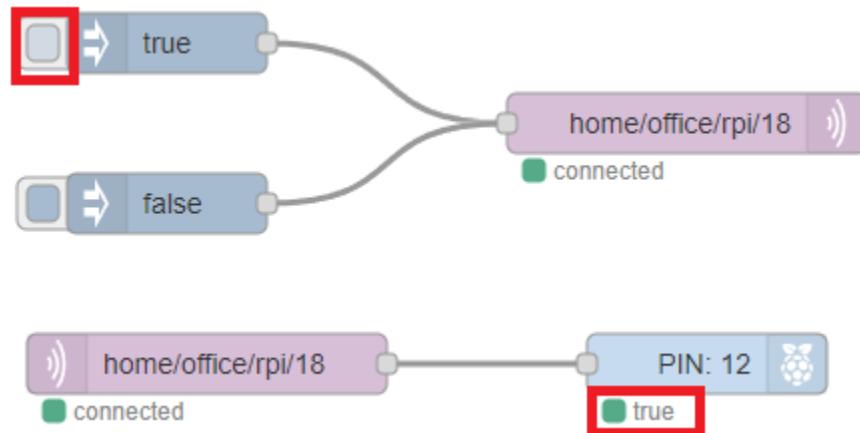
Wire your circuit by following the image above or these instructions:

- Longest lead of the LED → pin 12 (GPIO 18)
- Shortest lead of the LED → 270Ω resistor
- Resistor connected to GND

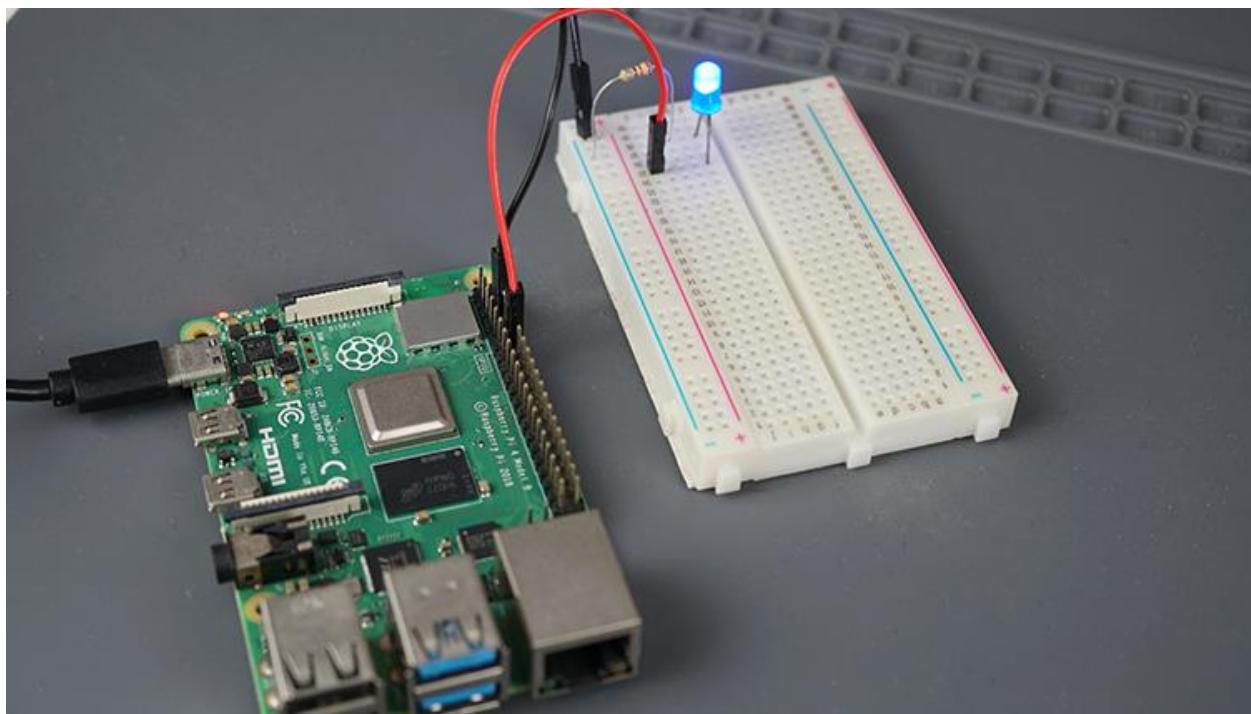
Demonstration

Now you can control the LED on and off using the MQTT publish and subscribe flow you created.

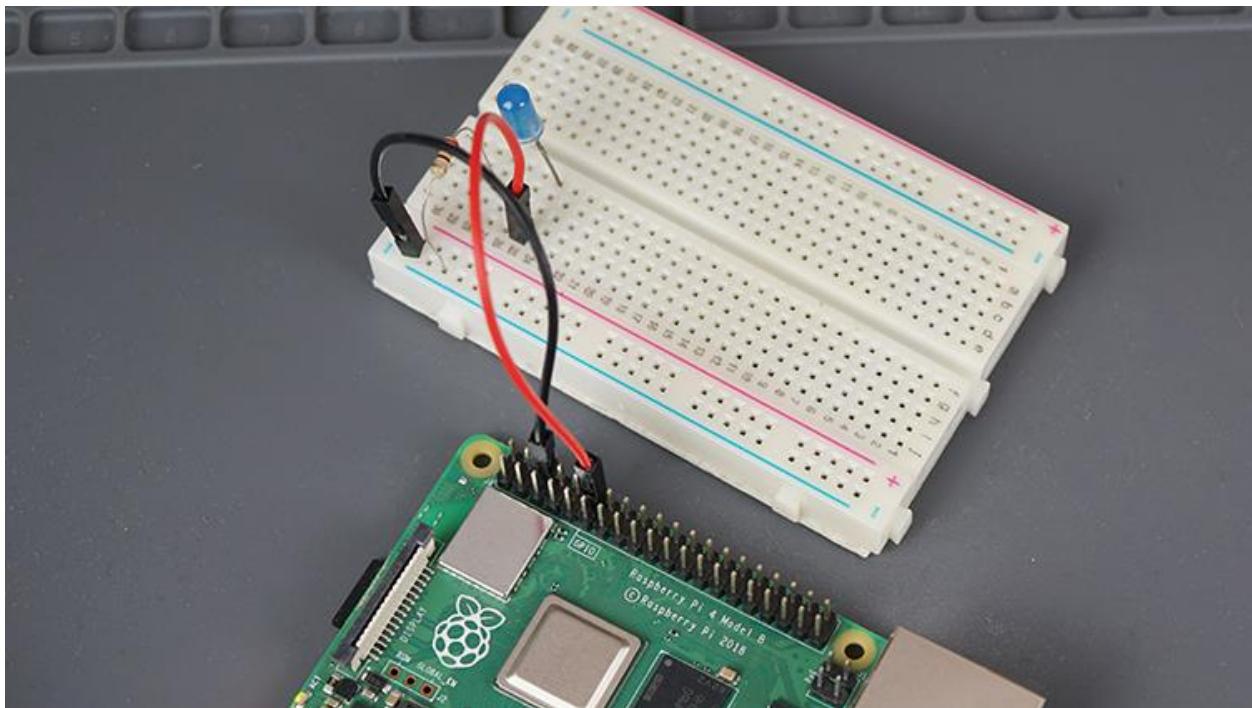
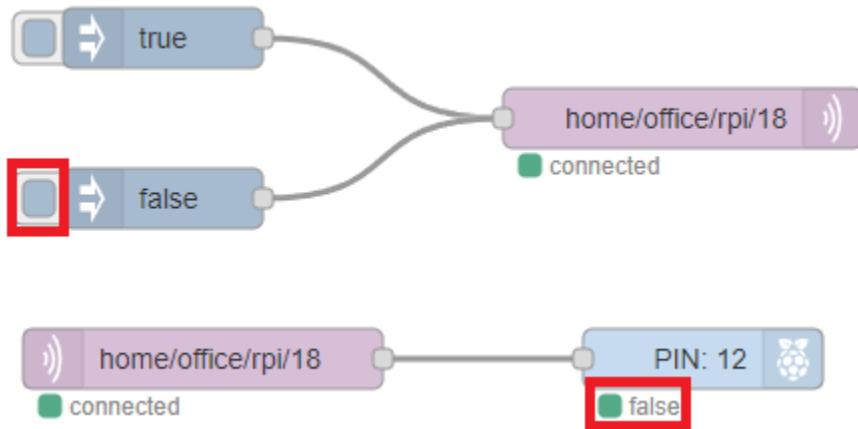
Now, press the square of the **true** node.



Your LED turns on:



And when you press the **false** node, the LED will turn off.



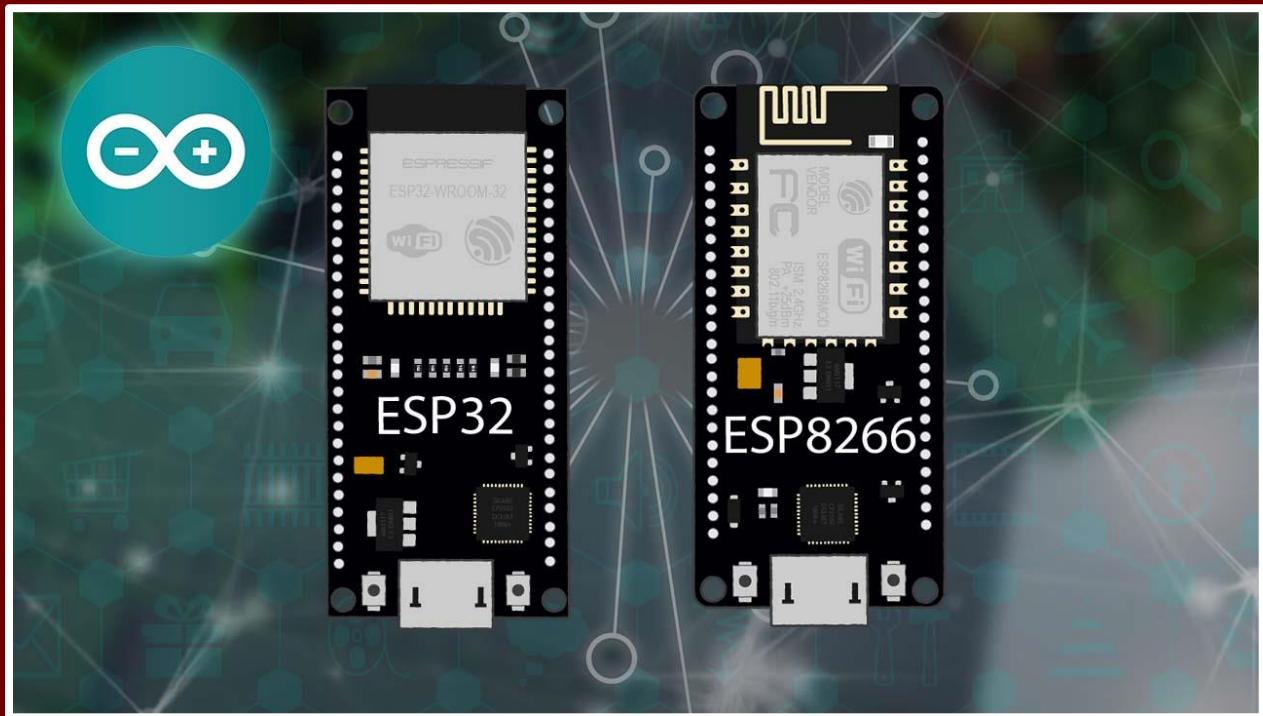
Wrapping Up

Congratulations! You've built an MQTT publish/subscribe flow. This example may not seem like much, but it's the starting point for more complex publish/subscribe flows and more advanced projects.

In the following modules, you'll learn how to connect the ESP32 and ESP8266 boards to your main server (Raspberry Pi) via MQTT to control outputs and read sensor data.

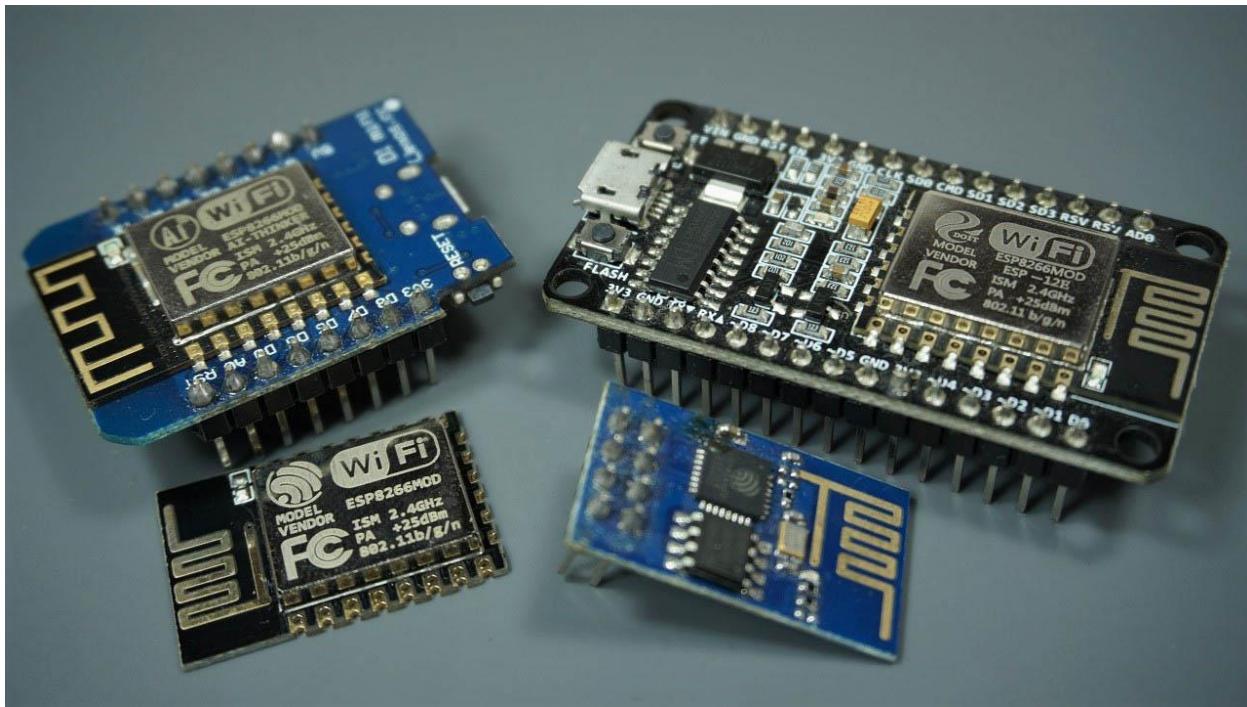
MODULE 4

Introducing the ESP32 and ESP8266 Boards



This Module is a quick introduction to the ESP32 and ESP8266 boards. We'll take a quick look at its features, specifications, and pinout. We'll also show you how to program your boards using Arduino IDE.

4.1 - Introducing the ESP8266



The ESP8266 is a Wi-Fi module great for IoT and Home Automation projects. This section is a quick introduction to the ESP8266 development board. You can skip this section if you're already familiar with the ESP8266.

Introducing the ESP8266

The ESP8266 is a \$4 (up to \$10) Wi-Fi module. With the ESP8266, you can control inputs and outputs as you would do with an Arduino, but with Wi-Fi capabilities. This means you can bring your projects online, which is great for home automation and internet of things applications.

ESP8266 Models

At the moment, there is a wide variety of development boards with the ESP8266 chip that differ in the number of GPIOs, antenna styles, size, breadboard compatibility, etc. The best ESP8266 development board for your project will depend on what you

intend to do. The most widely used ESP8266 development boards are the [ESP-01](#), the [ESP8266-12E NodeMCU Kit](#), and the [WeMos D1 Mini](#).

For a quick comparison of these boards, you can take a look at the following table:

| | ESP-01 | ESP-12E NODEMCU | WeMos D1 MINI |
|--------------------------------|---|--|---|
| |  |  |  |
| Where to Buy? | ESP-01 | ESP-12E NodeMCU | WeMos D1 Mini |
| GPIOs | 4 (including TX and RX) | 11 | 11 |
| ADC Pins | 1 | 1 | 1 |
| Flash Memory | 1MB (upgraded version) | 4MB | 4MB |
| Breadboard friendly | x | ✓ | ✓ |
| USB to Serial Converter | x | ✓ | ✓ |
| Size | 24.75mm x 14.5mm (0.97" x 0.57") | 48.55mm x 25.6mm (1.91" x 1") | 34.2mm x 25.6mm (1.35" x 1") |
| Price | \$ | \$\$ | \$\$ |

For a more in-depth comparison between the different ESP8266 boards, we recommend reading the following article:

- [Best ESP8266 Wi-Fi Development Boards](#)

If you have to pick only one board to follow the projects in this eBook, we highly recommend using the [ESP8266-12E NodeMCU Kit](#). However, any ESP8266 board should work.

ESP8266 Pinout

The ESP8266 boards come with several GPIOs (general purpose input output) pins that you can use to connect peripherals like sensors and actuators. Most pins have specific functions, so it's good to know how to identify the pins and their features.

In the next paragraphs, we'll show you the pinout diagrams for the most popular ESP8266 boards. If you've started playing with the ESP8266 boards recently, we recommend reading the following article, that explains in a great detail almost everything you need to know about the ESP8266 GPIOs:

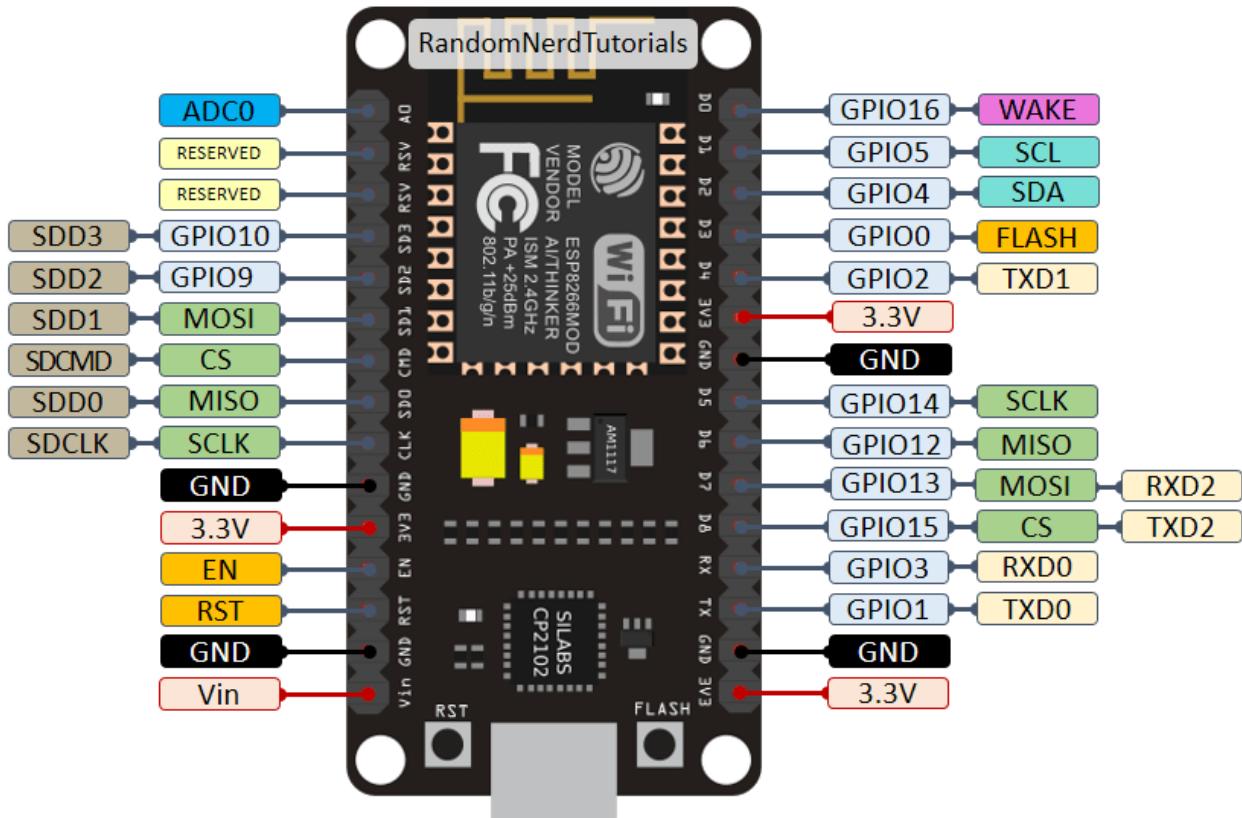
- [ESP8266 Pinout Reference: Which GPIO pins should you use?](#)

ESP8266-12E NodeMCU Kit Pinout

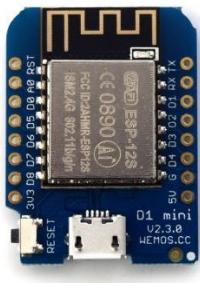


The ESP-12E NodeMCU Kit is one of the most used ESP8266 development boards. It features 4MB of flash memory, access to 11 GPIO pins, and one analog-to-digital converter (ADC) pin with 10-bit resolution. The board has a built-in voltage regulator, and you can power up the module using the mini USB socket or the *Vin* pin. Uploading code to the board is as easy as uploading code to an Arduino board. There's no need for an FTDI programmer, as it comes with a built-in USB-to-serial converter.

The following figure describes the ESP8266-12E GPIOs and their functionalities:

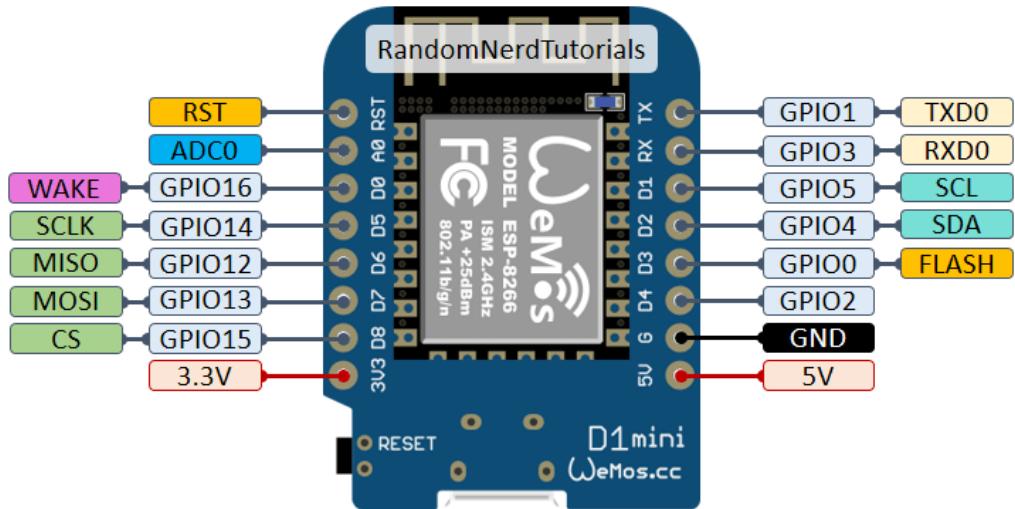


WeMos D1 Mini Pinout



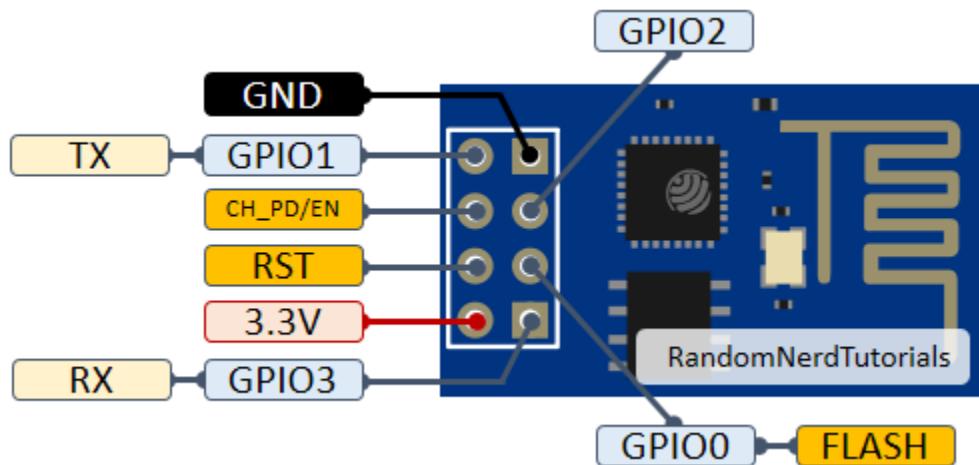
The WeMos D1 Mini offers 4MB flash memory, 11GPIOs, and 1 ADC pin in a minimal setup. So, if you need something that consumes less power for a more economical project, this is a good option. Additionally, the community has developed a wide variety of shields for the D1 mini board, which allows you to build small and simple setups with almost no wiring required. The WeMos D1 mini board is also an excellent option for beginners.

If this is your board, you can use the following pinout as a reference.



ESP8266-01 Pinout

We don't recommend using an ESP-01 board to follow this eBook if this is your first time using an ESP8266 board or if you're a beginner. If you're already familiar with ESP8266 boards, you can use this board. However, we didn't test the examples with this board. Additionally, the ESP-01 doesn't have enough GPIO pins for most of the examples presented. Nevertheless, here's the pinout for that board.



Remember that you'll need an [FTDI programmer](#) to connect this board to your computer or an [ESP-01 adapter/programmer](#).

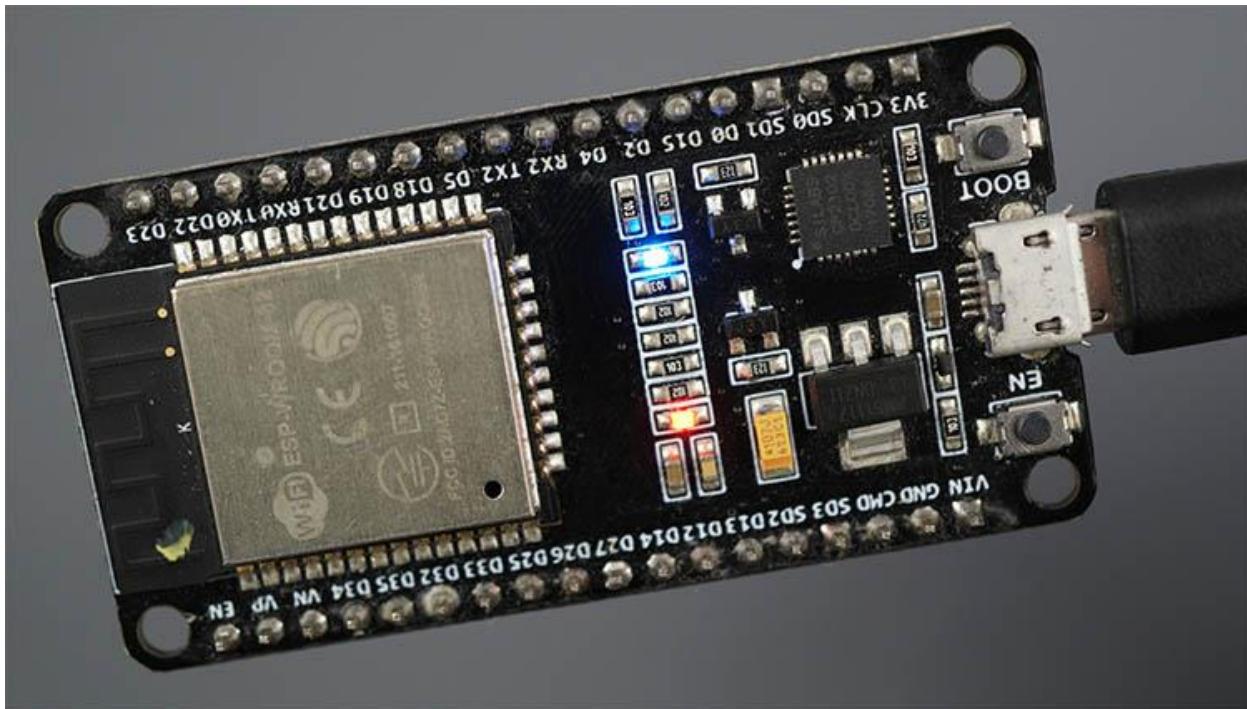
Programming Environments

The ESP8266 can be programmed in different ways. You can use the C/C++ Arduino core using Arduino IDE, LUA (NodeMCU firmware), and MicroPython.

We'll cover programming the **ESP8266 using Arduino IDE**, exclusively.

The following section is an introduction to the ESP32 board. If you won't use it to follow this eBook, you can skip that section.

4.2 - Introducing the ESP32



This unit introduces the ESP32 board, which is the ESP8266 successor. The ESP32 is loaded with lots of new features. It combines Wi-Fi and Bluetooth wireless capabilities.

For a more detailed explanation of the differences between the ESP32 and ESP8266, you can read the following article:

- [ESP32 vs ESP8266 – Pros and Cons](#)

You can find the ESP32 as a standalone chip or as a full-featured development board. There are many different models of ESP32 boards. I encourage you to visit the [ESP32.net website](#), where each ESP32 chip and development board is listed. You can compare their differences and features.

We have an article on the [Maker Advisor website](#) comparing some of the most popular ESP32 development boards. You can read the article on the following link:

- [ESP32 Development Boards Review and Comparison](#)

Introducing the ESP32 DOIT DEVKIT V1 Board

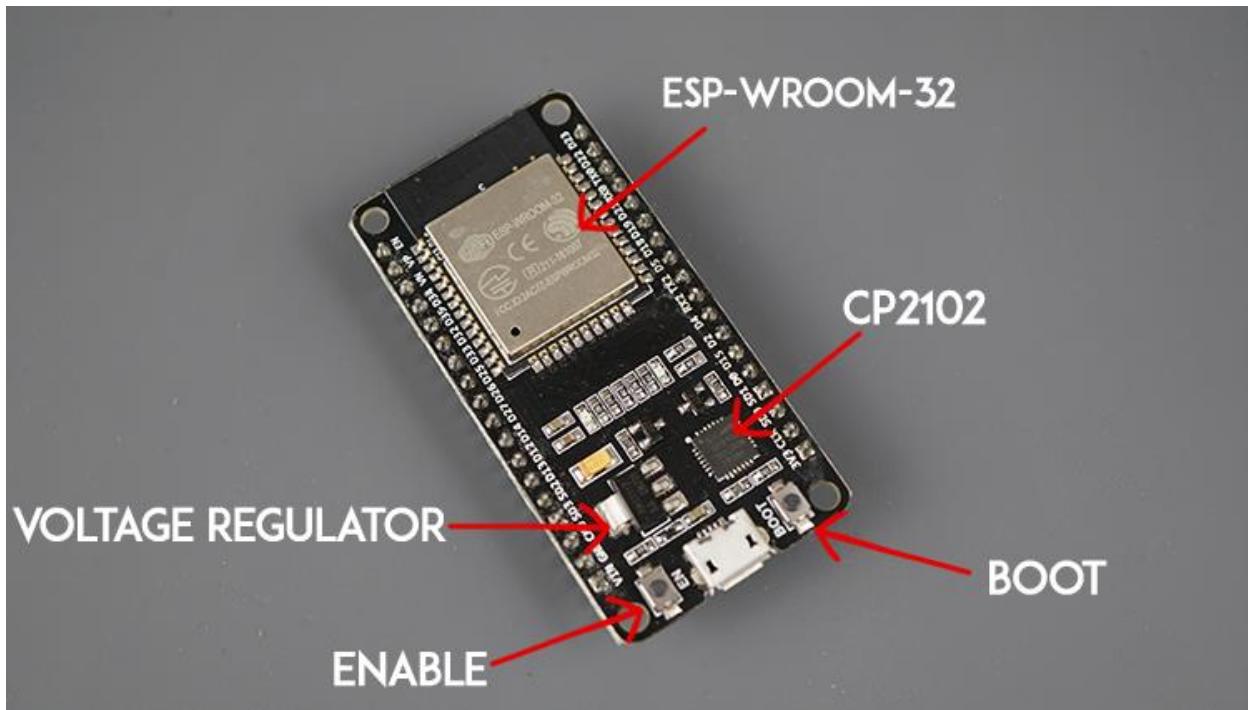
Throughout this eBook, we'll be using the [ESP32 DEVKIT V1 DOIT board](#) for the ESP32 examples, but any other ESP32 board should work just fine.



ESP32 Features

The ESP32 V1 DOIT board comes with the ESP-WROOM-32 chip. It has a 3.3V voltage regulator that drops the input voltage to power the ESP32 chip. It comes with a CP2102 chip (or a CH340 depending on the model) that allows you to plug the ESP32 into your computer to program it.

The board has two on-board buttons: the ENABLE and the BOOT button. The ENABLE button (or RST button on other models) reboots/restarts your ESP32. The BOOT button reboots the ESP32 in programming mode.



ESP32 Specifications

When it comes to the ESP32 chip specifications:

- The ESP32 is dual-core—this means it has two processors;
- It has Wi-Fi and Bluetooth built-in;
- It runs 32-bit programs;
- The clock frequency can go up to 240MHz, and it has 512 kB RAM;
- This particular board has 36 pins, 18 in each row. There are versions with 30, 38, and other numbers of pins;
- It also has various peripherals available, like capacitive touch, ADCs, DACs, UART, SPI, and I2C.

| Specifications – ESP32 DEVKIT V1 DOIT | |
|---------------------------------------|--|
| Number of cores | 2 (dual-core) |
| Wi-Fi | 2.4 GHz up to 150 Mbit/s |
| Bluetooth | BLE (Bluetooth Low Energy) and legacy Bluetooth |
| Architecture | 32 bit |
| Clock frequency | Up to 240 MHz |
| RAM | 512 KB |
| Pins | 30, 36, or 38 depending on the board model |
| Peripherals | Capacitive touch, ADC (analog-to-digital converter), DAC (digital-to-analog converter), I ² C (inter-integrated circuit), UART (universal asynchronous receiver/transmitter), CAN 2.0 (controller area network), SPI (serial peripheral interface), I ² S (integrated inter-IC sound), RMII (reduced media-independent interface), PWM (pulse width modulation). |

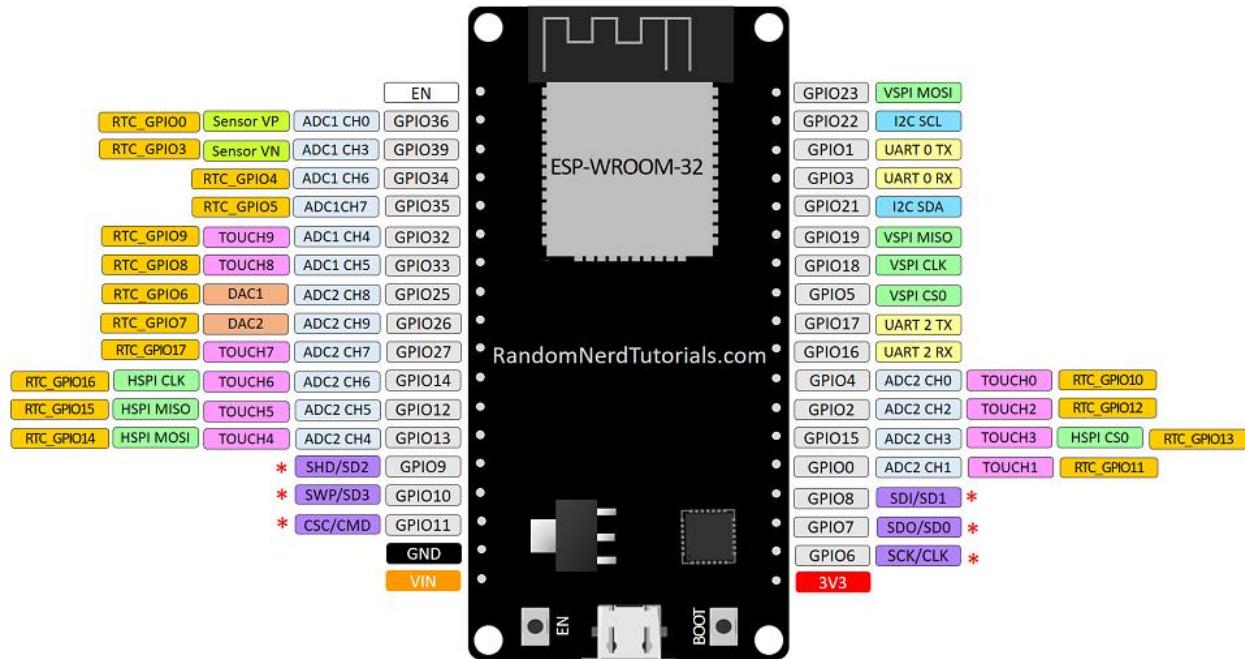
ESP32 Pinout

The following figure describes the board GPIOs and their functionalities. We recommend printing this pinout for future reference. You can download the pinout in *.pdf* or *.png* files:

- [Printable version](#)
- [Image version 30 pins](#)
- [Image version 36 pins](#)

ESP32 DEVKIT V1 – DOIT

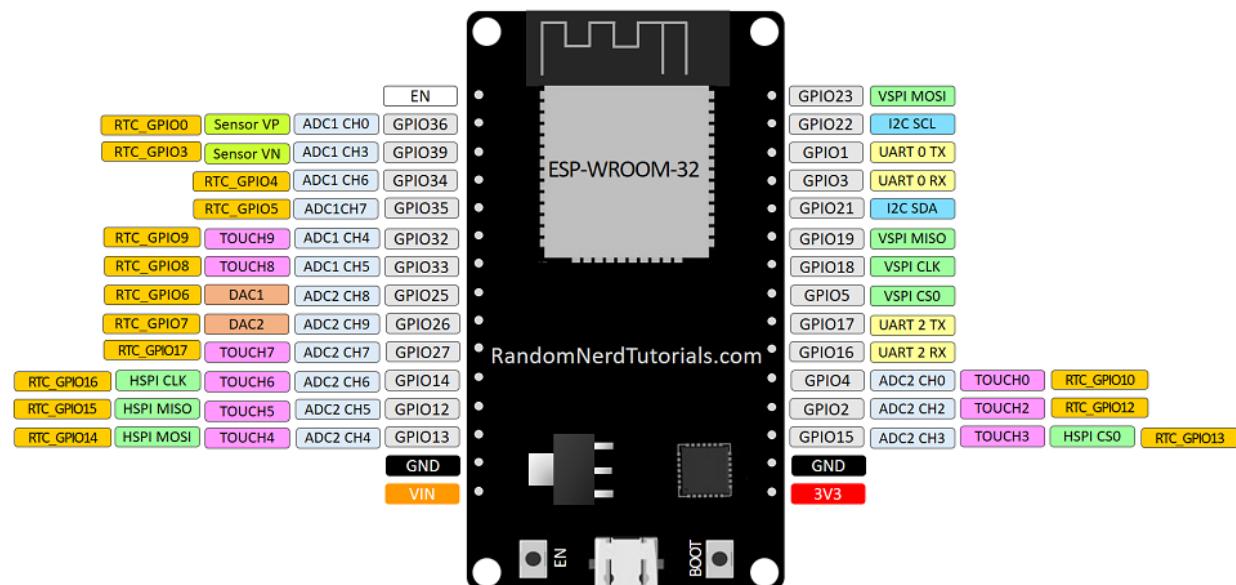
version with 36 GPIOs



* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

ESP32 DEVKIT V1 – DOIT

version with 30 GPIOs



For more information about the ESP32 GPIOs, we recommend reading the following article:

- [ESP32 Pinout Reference: Which GPIO pins should you use?](#)

Programming Environments

You can program the ESP32 in different programming environments. You can use the Arduino IDE, Espressif IDF (IoT Development Framework), MicroPython, JavaScript, LUA, etc. Throughout this eBook, we'll focus on programming the **ESP32 with Arduino IDE**.

Proceed to the next section to install Arduino IDE and start programming your boards.

4.3 - Installing Arduino IDE

We'll program the ESP32 and ESP8266 boards using Arduino IDE. You can also use VS Code with the PlatformIO extension. We provide instructions on how to install and program the boards using Arduino IDE. If you prefer to use VS Code instead, we recommend, following the tutorial on the next URL:

- [Getting Started with VS Code and PlatformIO IDE for ESP32 and ESP8266](#)

Requirements

You need JAVA installed on your computer. If you don't, go to the following website to download and install the latest version:

- <https://www.java.com/en/download/>

Downloading Arduino IDE

To download the Arduino IDE, visit the following URL:

- <https://www.arduino.cc/en/software>

The screenshot shows the Arduino IDE 2.3.2 download page. On the left, there's a teal button with two white circles and a plus sign, followed by the text "Arduino IDE 2.3.2". Below this, a paragraph describes the new features of the release. A link to the documentation is provided. At the bottom, there's a "SOURCE CODE" section linking to GitHub. On the right, a teal sidebar titled "DOWNLOAD OPTIONS" lists download links for Windows, Linux, and macOS, along with their respective file types and system requirements. A "Release Notes" link is also present.

Arduino IDE 2.3.2

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the [Arduino IDE 2.0 documentation](#).

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on [GitHub](#).

DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file

Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits
macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

[Release Notes](#)

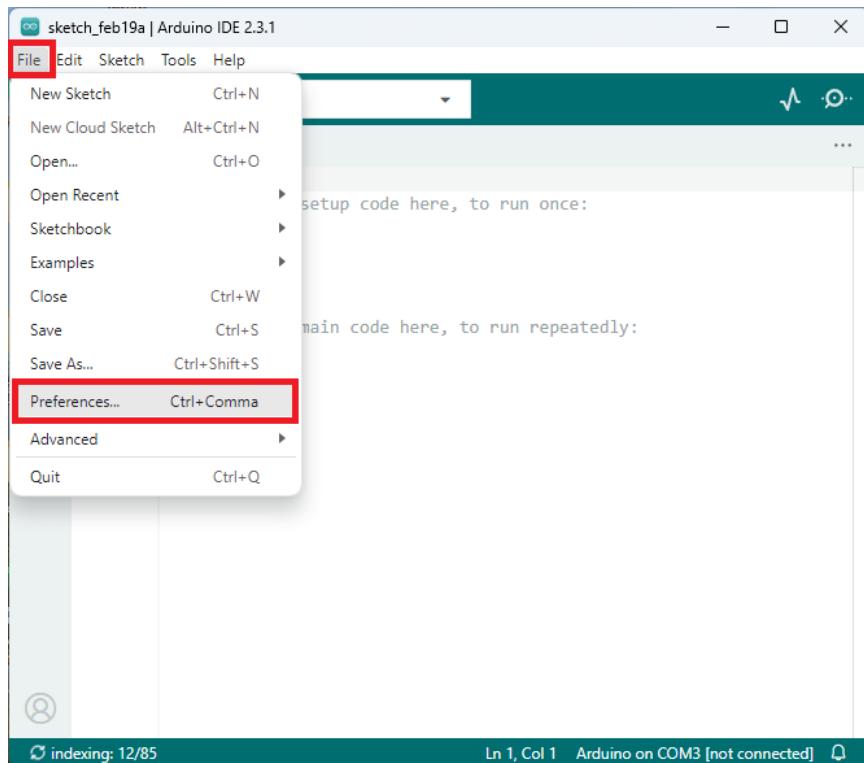
- **Windows:** extract the downloaded folder, run the executable file and follow the instructions in the installation guide.
- **Mac OS X:** copy the downloaded file into your *application* folder.
- **Linux:** extract the downloaded file, and open the *arduino-ide* file that will launch the IDE.

If you have any doubts, you can go to the [Arduino Installation Guide](#).

Installing the ESP32/ESP8266 add-ons for Arduino IDE

To program the ESP32 and ESP8266 boards using Arduino IDE, you need to install the ESP32 and ESP8266 add-on. Follow the next steps to install them.

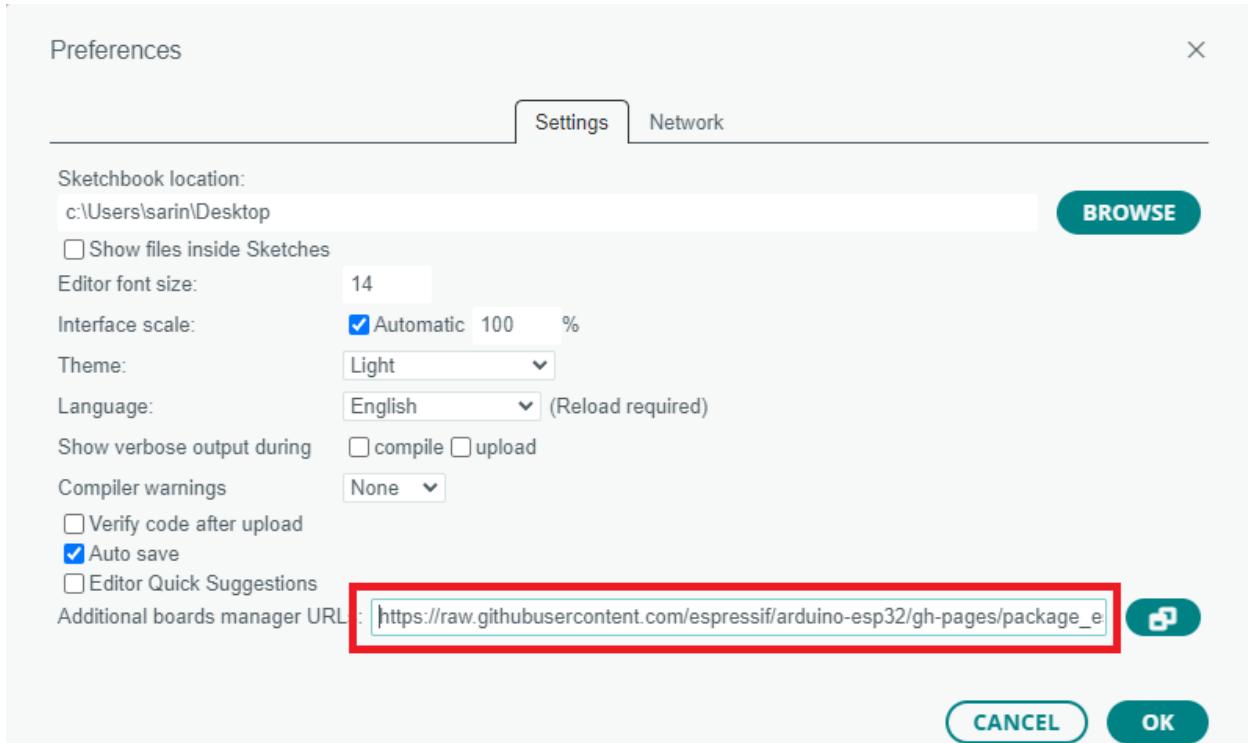
Open the **Preferences** window in the Arduino IDE. Go to **File > Preferences**:



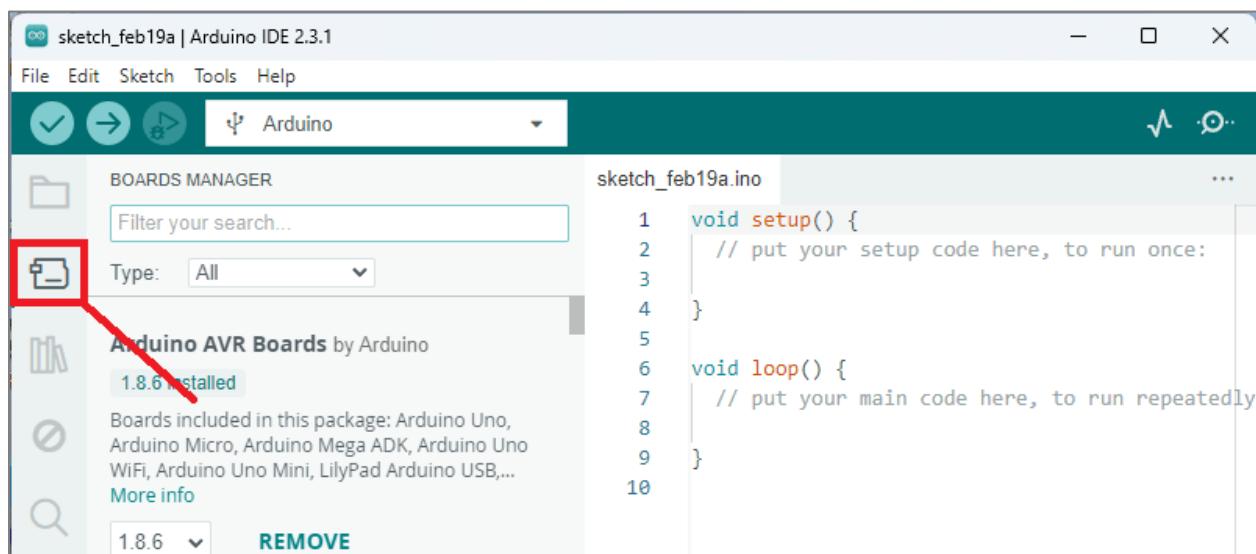
Copy and paste the following into the "Additional Board Manager URLs" field.

https://raw.githubusercontent.com/espressif/arduino-esp32/master/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

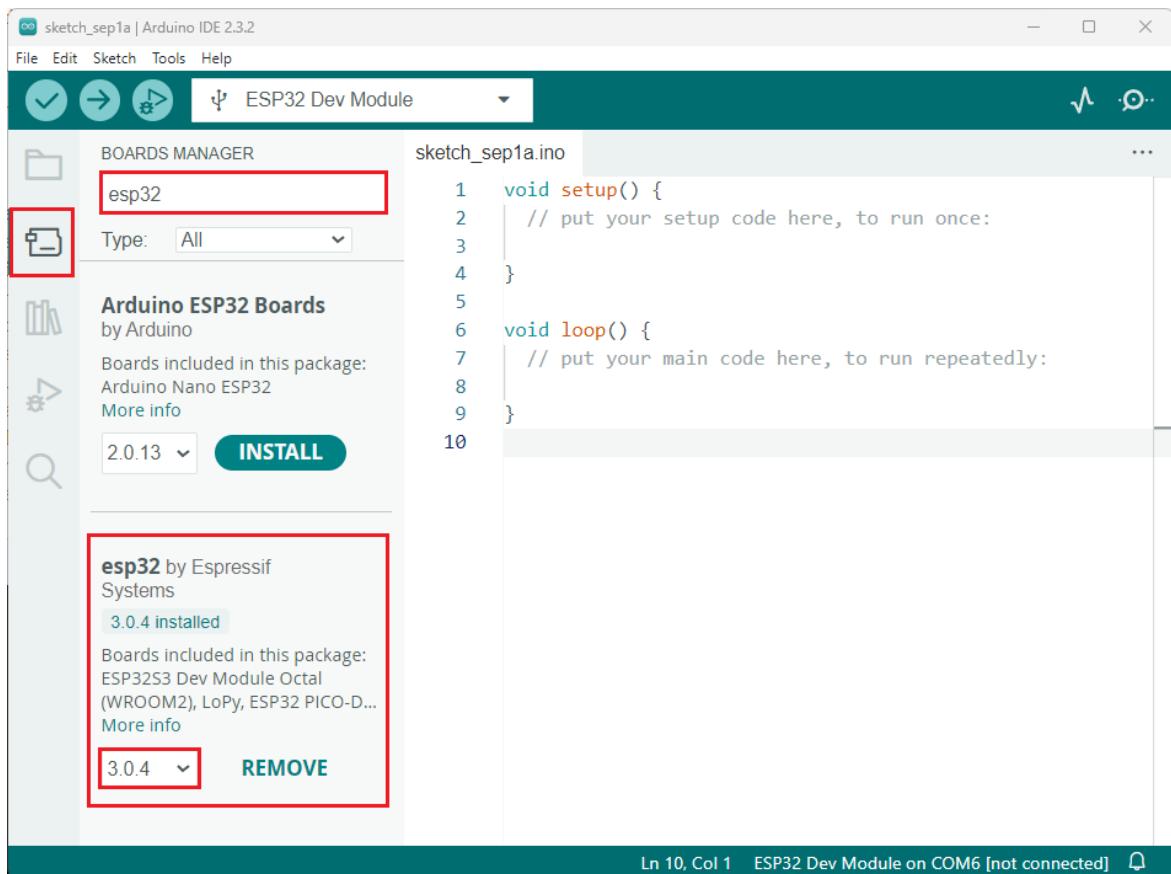
See the figure below. Then, click the "OK" button.



Open the **Boards Manager**. Go to **Tools > Board > Boards Manager...** or you can simply click the Boards Manager icon in the left-side corner.

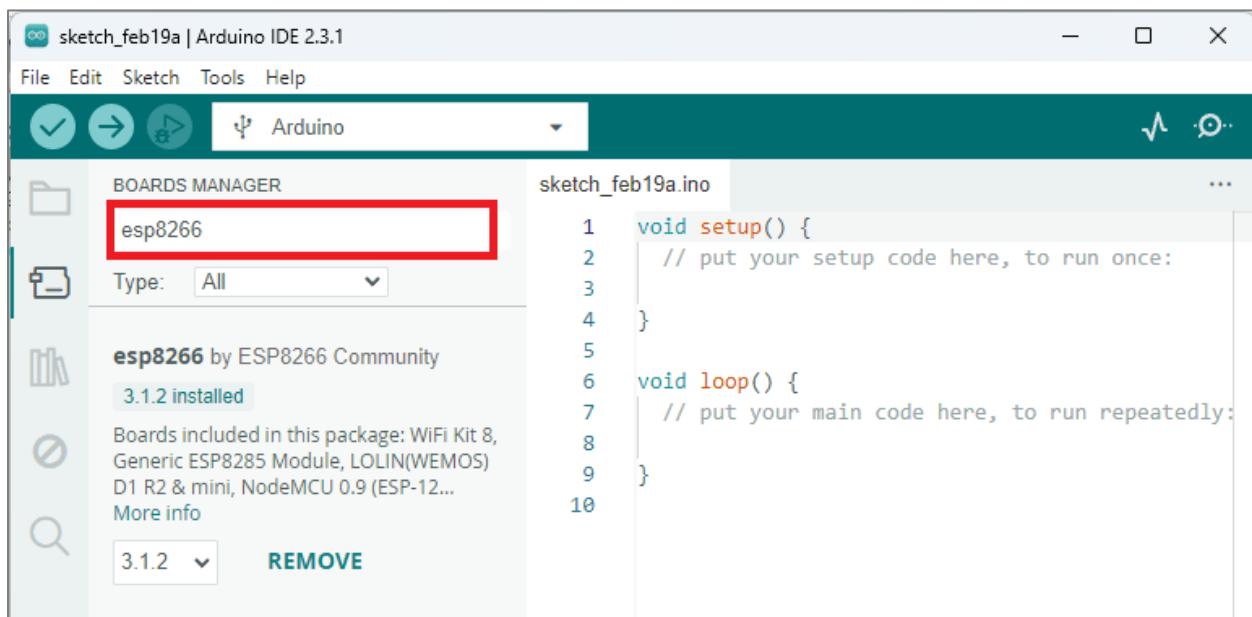


Search for ESP32 and install the "**ESP32 by Espressif Systems**":



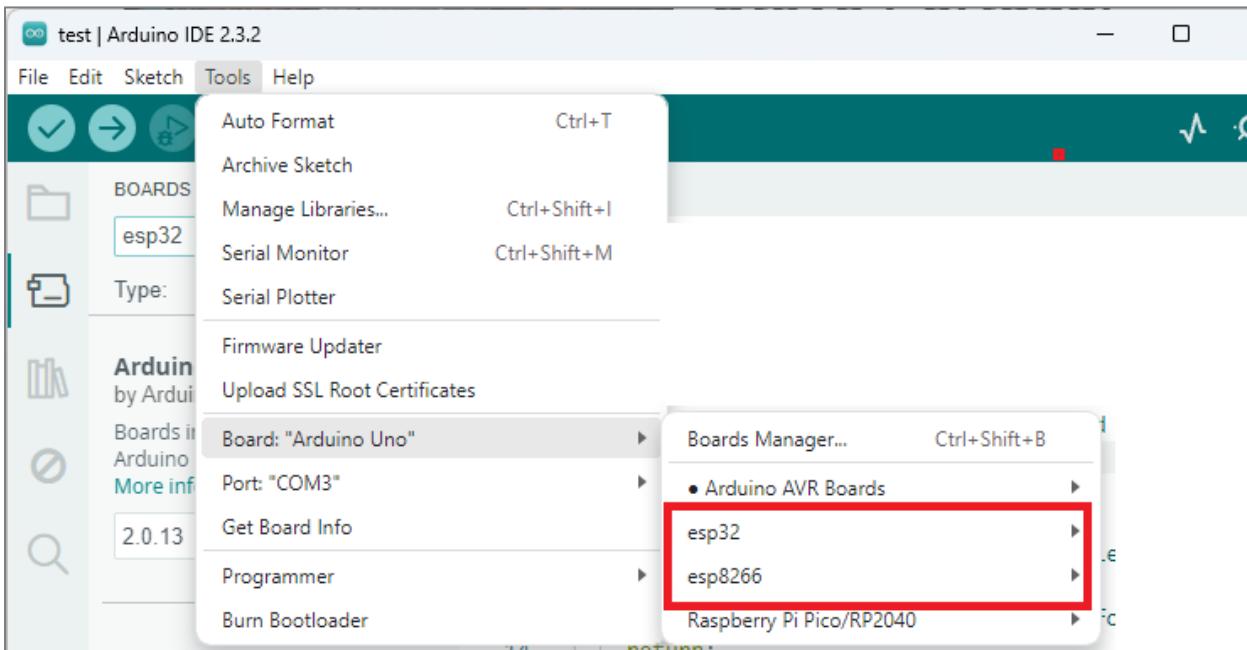
You should install version 3. That's it. It will be installed after a few seconds.

Still on the **Boards Manager** menu, search for "**esp8266**" and press the **Install** button for the "**ESP8266 by ESP8266 Community**".



After this, restart your Arduino IDE.

Then, go to **Tools > Board** and check that you have ESP32 and ESP8266 boards available.



And that's it. You have the ESP32 and ESP8266 add-ons installed in your Arduino IDE.

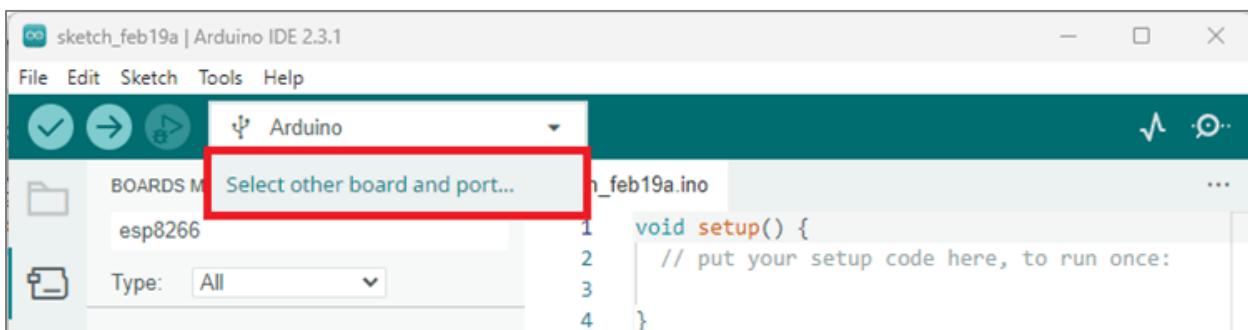
4.4 - Testing the Installation

In this unit, we'll quickly check if the installation was successful and if we can upload new code to the ESP32 or ESP8266.

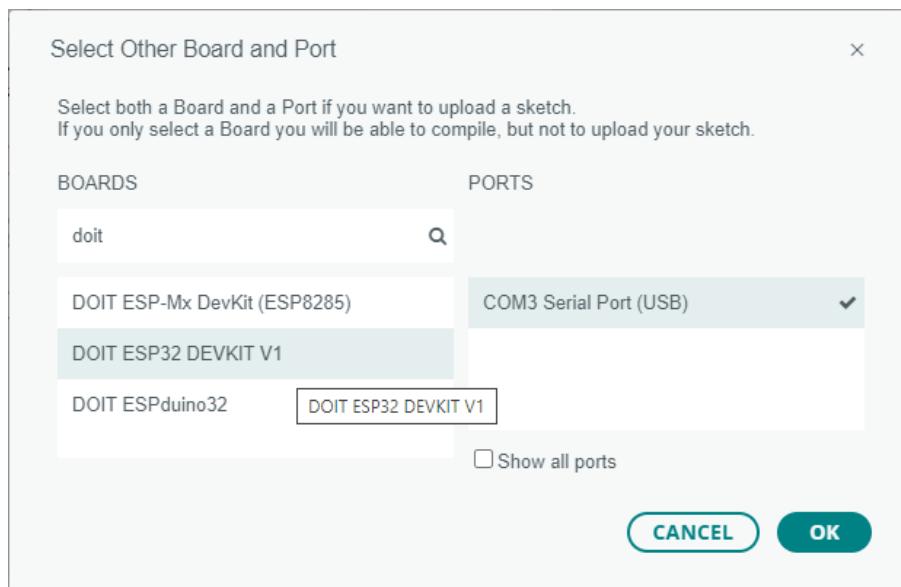
We'll simply upload an example sketch from the library of available examples.

Plug the ESP32 or ESP8266 board into your computer. With your Arduino IDE open, follow these steps:

- 1) Select your Board in **Tools > Board** menu or on the top drop-down menu, click on "Select other board and port..."



A new window, as shown below, will open. Search for your ESP32 or ESP8266 board model.

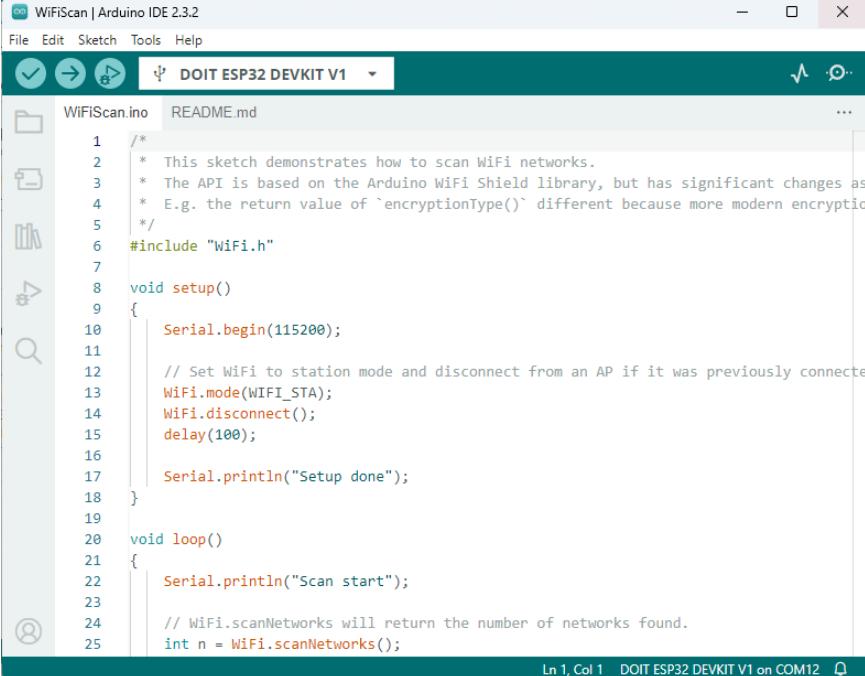


Select the board model you're using, and the COM port. In our example, we're using the DOIT ESP32 DEVKIT V1. Click **OK** when you're done.

If you don't see the COM Port in your Arduino IDE, you need to install the [CP210x USB to UART Bridge VCP Drivers](#) or other drivers depending on the board model you're using).

We recommend taking a look at this tutorial: [Install ESP32/ESP8266 USB Drivers – CP210x USB to UART Bridge](#)

- 2) Open one of the following examples depending on the board you're using. The examples are similar—search for wi-fi networks within the range of your board.
 - ESP32: **File > Examples > WiFi (ESP32) > WiFiScan**
 - ESP8266: **File > Examples > ESP8266WiFi > WiFiScan**
- 3) A new sketch opens in your Arduino IDE:



The screenshot shows the Arduino IDE 2.3.2 interface with the title bar "WiFiScan | Arduino IDE 2.3.2". The toolbar includes standard icons for file operations. The board selection dropdown shows "DOIT ESP32 DEVKIT V1". The main window displays the "WiFiScan.ino" sketch. The code is as follows:

```
/* WiFiScan.ino
 * This sketch demonstrates how to scan WiFi networks.
 * The API is based on the Arduino WiFi Shield library, but has significant changes as
 * E.g. the return value of `encryptionType()` different because more modern encryption
 */
#include "WiFi.h"

void setup()
{
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was previously connected
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    Serial.println("Setup done");
}

void loop()
{
    Serial.println("Scan start");

    // WiFi.scanNetworks will return the number of networks found.
    int n = WiFi.scanNetworks();
```

The status bar at the bottom indicates "Ln 1, Col 1 DOIT ESP32 DEVKIT V1 on COM12".

- 4) Press the **Upload** button in the Arduino IDE. Wait a few seconds while the code compiles and uploads to your board.



Note: if you see a lot of dots on the debugging window, followed by an upload error, that means your board doesn't go into flashing mode automatically. Click the Upload button again, and when you start seeing the dots on the debugging window, press the onboard BOOT button for a couple of seconds.

- 5) If everything went as expected, it will upload successfully after a few seconds.

You'll get a similar message:

```
Output
Writing at 0x000b7163... (96 %)
Writing at 0x000bc7e1... (100 %)
Wrote 721168 bytes (468882 compressed) at 0x00010000 in 7.7 seconds (effective 747.8 kbi)
Hash of data verified.

Leaving...
Hard resetting via RTS pin...
```

- 6) Open the Arduino IDE Serial Monitor at a baud rate of 115200:



- 7) Press the ESP32 on-board Enable button or the ESP8266 RST button and you should see the networks available near your board.

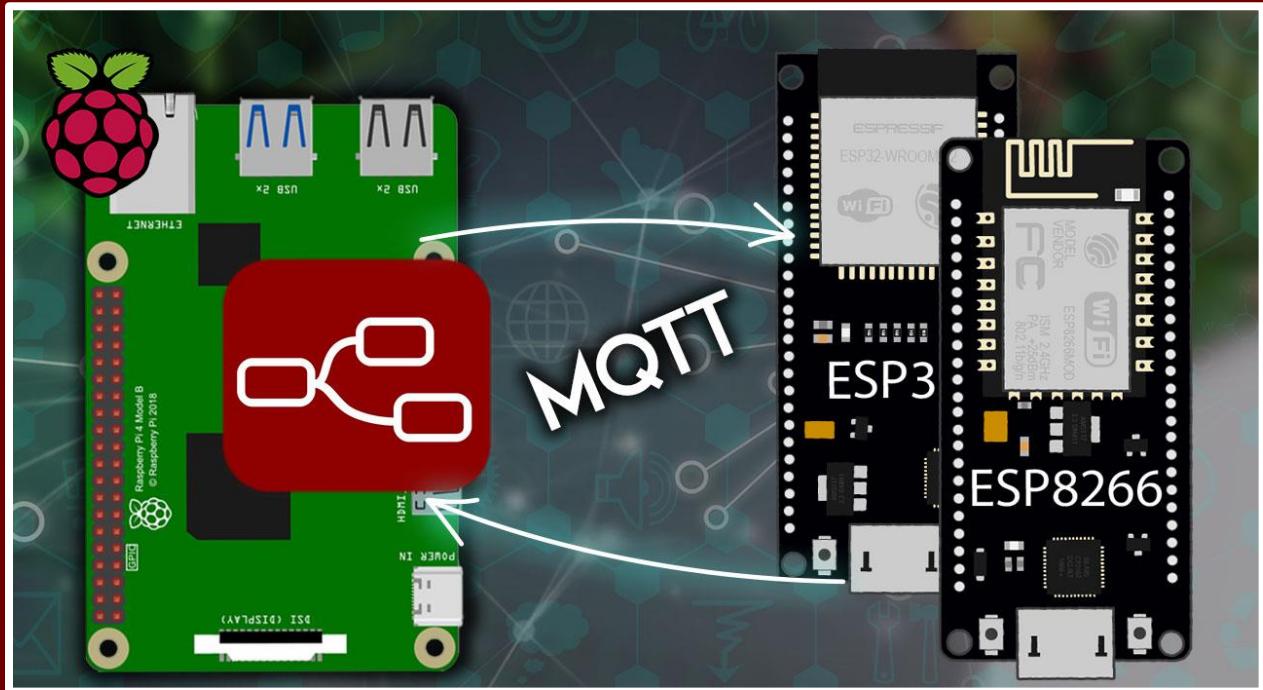
| Nr | SSID | RSSI | CH | Encryption |
|----|-------------|------|----|------------|
| 1 | MEO-D3 A :0 | -54 | 1 | WPA2 |
| 2 | MEO-Wi. | -55 | 1 | open |
| 3 | MEO-029: i^ | -74 | 11 | WPA+WPA2 |
| 4 | MEO-029: | -81 | 11 | WPA2 |
| 5 | MEO-Wi' . | -82 | 11 | open |
| 6 | MEO-Wi i | -86 | 6 | open |
| 7 | MEO-f 087 | -87 | 6 | WPA+WPA2 |
| 8 | MEO- Jt B0 | -92 | 1 | WPA+WPA2 |
| 9 | MEO- iF | -92 | 1 | open |
| 10 | MEO- 2495 | -92 | 5 | WPA+WPA2 |

Everything seems to be working as expected, you can proceed to the next module.

If you're having other issues uploading code to your ESP32 board, we recommend taking a quick look at the following troubleshooting guide: [ESP32 Troubleshooting Guide](#).

MODULE 5

Connecting the ESP32/ESP826 with Node-RED (MQTT)

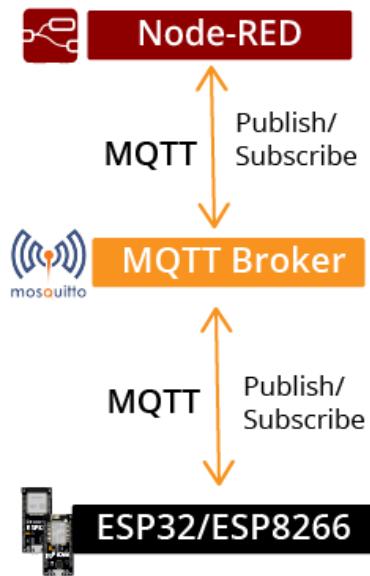


In this Module, you'll learn how to program your ESP32 and ESP8266 boards to connect to your broker to publish messages and subscribe to topics. This way, we'll be able to establish a communication between the ESP boards and Node-RED using MQTT. You'll learn how to control the ESP outputs using Node-RED dashboard and to display sensor readings published by your boards.

5.1 - Connecting the ESP32/ESP8266 with Node-RED (Introduction)

The ESP32, ESP8266, and Node-RED can communicate with each other using MQTT communication. They act as MQTT clients that subscribe and publish messages on certain topics. The MQTT broker receives all messages and sends them to the corresponding subscribers.

In previous units, you learned how to create a publish and subscribe flow using Node-RED. Now, we'll see how to set the ESP32 or ESP8266 boards as MQTT clients that can publish and subscribe to topics and communicate with the Node-RED MQTT client.



Our aim is the following:

- The ESP32/ESP8266 boards can send messages to Node-RED;
- Node-RED can receive messages from the ESP boards;
- The ESP boards can receive messages from Node-RED;
- Node-RED can send messages to the ESP boards.

Installing the AsyncMQTT Client Library

There are several MQTT libraries that you can use to set the ESP32 or ESP8266 as an MQTT client. We'll use the *AsyncMQTT Client* library.

You can check the library GitHub page on the following link:

- <https://randomnerdtutorials.com/async-mqtt-client>

Here's the library documentation for future reference:

- <https://randomnerdtutorials.com/async-mqtt-client-docs>

1) Click on the following link to download the library ZIP file:

- AsyncMQTT Client: <https://randomnerdtutorials.com/async-mqtt-client-download>

Alternatively, you can go to the library GitHub page, click on the **Code** button, and then, **Download ZIP**.

The library needs the following dependencies depending on the board your using:

- **ESP32:** [AsyncTCP library](#)
- **ESP8266:** [ESPAsyncTCP library](#)

2) Click on the following links to download the libraries:

- AsyncTCP: <https://github.com/me-no-dev/AsyncTCP/archive/refs/heads/master.zip>
- ESPAsyncTCP: <https://github.com/me-no-dev/ESPAsyncTCP/archive/refs/heads/master.zip>

Now that you've downloaded all the required library files, let's install them.

Open your Arduino IDE, go to **Sketch > Include Library > Add .ZIP library** and select the ZIP file for the *AsyncMQTT Client* library.

Repeat the process to install the dependencies (the AsyncTCP or ESPAsyncTCP).

After installing the libraries, restart your Arduino IDE and proceed to the next unit.

5.2 - Publish and Subscribe (Basic Example)

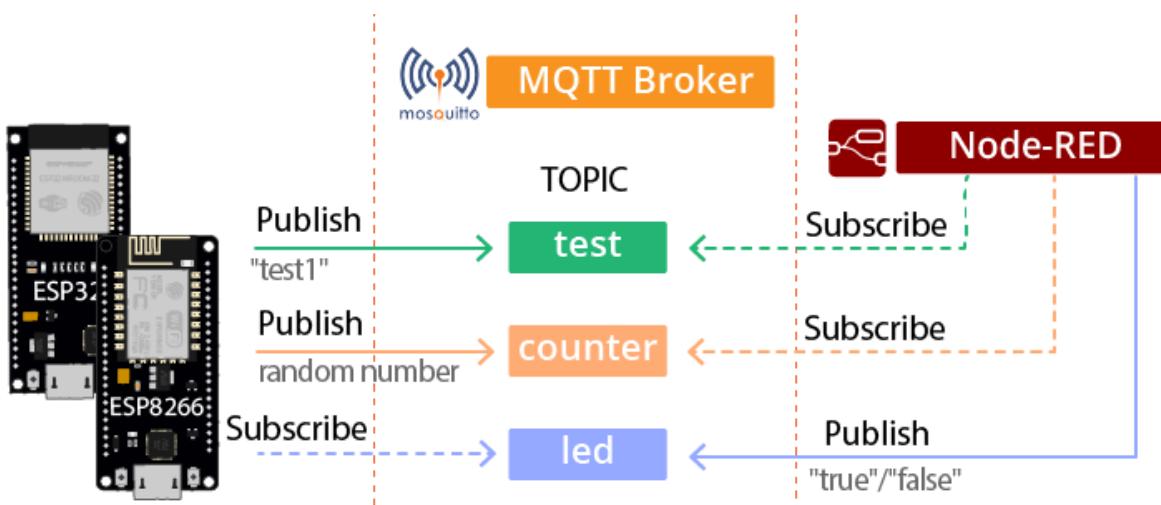
In this unit, we'll program the ESP32/ESP8266 to subscribe and publish to topics.

With this simple example, you'll learn:

- How to publish to topics with the ESP boards;
- How to subscribe to topics;
- Read a message that is published on a topic the ESP is subscribed to;
- Perform a task depending on the received message.

Project Overview

We'll create a simple example that does the following:



- When the ESP connects to the MQTT broker it publishes several messages on the `test` topic.
- The ESP publishes a message containing a random number on a topic (`counter`) periodically.
- The ESP is subscribed to a topic called `led`.

We'll create a flow in Node-RED to communicate with the ESP:

- It is subscribed to the `test` topic, so that we know the ESP initialized properly.
- It is subscribed to the topic `counter` to receive the numbers sent by the ESP. We'll display those values on a chart.
- It publishes `true` and `false` messages on the `led` topic. The ESP is subscribed to that topic, so it receives the messages. When it receives the `true` message it turns an LED on; when it receives the `false` message, it turns the LED off.

Publish/Subscribe Code

The code is slightly different for each board. The differences are highlighted in light yellow. Upload the correct code for the board you're using.

- [Download Sketch folder ESP32](#)
- [Download Sketch folder ESP8266](#)

This example is based on the examples available in the library *examples* folder. You can check all the examples on the following link:

- <https://github.com/marvinroger/async-mqtt-client/tree/develop/examples>

ESP32 – Code

Here's the code for the ESP32. You need to insert your network credentials (SSID and password), the broker IP address, and the broker username and password. The lines where you need to insert your details are highlighted in light green.

```
#include <WiFi.h>
extern "C" {
    #include "freertos/FreeRTOS.h"
    #include "freertos/timers.h"
}
#include <AsyncMqttClient.h>

#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"
```

```

#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106)*/
#define MQTT_PORT 1883
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"

AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;

unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

void WiFiEvent(WiFiEvent_t event) {
    Serial.printf("[WiFi-event] event: %d\n", event);
    switch(event) {
        case ARDUINO_EVENT_WIFI_STA_GOT_IP:
            Serial.println("WiFi connected");
            Serial.println("IP address: ");
            Serial.println(WiFi.localIP());
            connectToMqtt();
            break;
        case ARDUINO_EVENT_WIFI_STA_DISCONNECTED:
            Serial.println("WiFi lost connection");
            xTimerStop(mqttReconnectTimer, 0); // ensure we don't reconnect to MQTT while
reconnecting to Wi-Fi
            xTimerStart(wifiReconnectTimer, 0);
            break;
    }
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);

    // Subscribe to topic "led" when it connects to the broker
    uint16_t packetIdSub = mqttClient.subscribe("led", 2);
    Serial.print("Subscribing at QoS 2, packetId: ");
    Serial.println(packetIdSub);

    // Publish on the "test" topic with qos 0
}

```

```

    mqttClient.publish("test", 0, true, "test 1");
    Serial.println("Publishing at QoS 0");
    // Publish on the "test" topic with qos 1
    uint16_t packetIdPub1 = mqttClient.publish("test", 1, true, "test 2");
    Serial.print("Publishing at QoS 1, packetId: ");
    Serial.println(packetIdPub1);
    // Publish on the "test" topic with qos 2
    uint16_t packetIdPub2 = mqttClient.publish("test", 2, true, "test 3");
    Serial.print("Publishing at QoS 2, packetId: ");
    Serial.println(packetIdPub2);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        xTimerStart(mqttReconnectTimer, 0);
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message
    // Save the message in a variable
    String receivedMessage;
    for (int i = 0; i < len; i++) {
        Serial.println((char)payload[i]);
        receivedMessage += (char)payload[i];
    }
    // Turn the LED on or off accordingly to the message content
    if (strcmp(topic, "led") == 0) {
        if (receivedMessage == "true"){
            digitalWrite(LED_BUILTIN, HIGH);
        }
        if (receivedMessage == "false"){
            digitalWrite(LED_BUILTIN, LOW);
        }
    }
    Serial.println("Publish received.");
    Serial.print(" topic: ");
}

```

```

Serial.println(topic);
Serial.print(" qos: ");
Serial.println(properties.qos);
Serial.print(" dup: ");
Serial.println(properties.dup);
Serial.print(" retain: ");
Serial.println(properties.retain);
Serial.print(" len: ");
Serial.println(len);
Serial.print(" index: ");
Serial.println(index);
Serial.print(" total: ");
Serial.println(total);
}

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();
    pinMode (LED_BUILTIN, OUTPUT);

    mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
    wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));

    WiFi.onEvent(WiFiEvent);

    mqttClient.onConnect(onMqttConnect);
    mqttClient.onDisconnect(onMqttDisconnect);
    mqttClient.onSubscribe(onMqttSubscribe);
    mqttClient.onUnsubscribe(onMqttUnsubscribe);
    mqttClient.onMessage(onMqttMessage);
    mqttClient.onPublish(onMqttPublish);
    mqttClient.setServer(MQTT_HOST, MQTT_PORT);
    mqttClient.setCredentials(BROKER_USER, BROKER_PASS);

    connectToWifi();
}

void loop() {
    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 10 seconds)
    // it publishes a new MQTT message
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;
}

```

```

    // Publish an MQTT message on topic counter
    uint16_t packetIdPub1 = mqttClient.publish("counter", 1, true,
String(random(0,20)).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", "counter",
packetIdPub1);
}

```

ESP8266 – Code

Here's the code for the ESP8266. You need to insert your network credentials (SSID and password), the broker IP address, and the broker username and password. The lines where you need to insert your details are highlighted in light green.

```

#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>

#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106)*/
#define MQTT_PORT 1883
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"

AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;

WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
Ticker wifiReconnectTimer;

unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values

void connectToWifi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
  Serial.println("Connecting to MQTT...");
  mqttClient.connect();
}

```

```

void onWifiConnect(const WiFiEventStationModeGotIP& event) {
    Serial.println("Connected to Wi-Fi.");
    connectToMqtt();
}

void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
    Serial.println("Disconnected from Wi-Fi.");
    mqttReconnectTimer.detach(); // ensure we don't reconnect to MQTT while
reconnecting to Wi-Fi
    wifiReconnectTimer.once(2, connectToWifi);
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);

    // Subscribe to topic "led" when it connects to the broker
    uint16_t packetIdSub = mqttClient.subscribe("led", 2);
    Serial.print("Subscribing at QoS 2, packetId: ");
    Serial.println(packetIdSub);

    // Publish on the "test" topic with qos 0
    mqttClient.publish("test", 0, true, "test 1");
    Serial.println("Publishing at QoS 0");
    // Publish on the "test" topic with qos 1
    uint16_t packetIdPub1 = mqttClient.publish("test", 1, true, "test 2");
    Serial.print("Publishing at QoS 1, packetId: ");
    Serial.println(packetIdPub1);
    // Publish on the "test" topic with qos 2
    uint16_t packetIdPub2 = mqttClient.publish("test", 2, true, "test 3");
    Serial.print("Publishing at QoS 2, packetId: ");
    Serial.println(packetIdPub2);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        mqttReconnectTimer.once(2, connectToMqtt);
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
}

```

```

    Serial.println(packetId);
}

void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message

    // Save the message in a variable
    String receivedMessage;
    for (int i = 0; i < len; i++) {
        Serial.println((char)payload[i]);
        receivedMessage += (char)payload[i];
    }
    // Turn the LED on or off accordingly to the message content
    if (strcmp(topic, "led") == 0) {
        if (receivedMessage == "true"){
            digitalWrite(LED_BUILTIN, HIGH);
        }
        if (receivedMessage == "false"){
            digitalWrite(LED_BUILTIN, LOW);
        }
    }
    Serial.println("Publish received.");
    Serial.print(" topic: ");
    Serial.println(topic);
    Serial.print(" qos: ");
    Serial.println(properties.qos);
    Serial.print(" dup: ");
    Serial.println(properties.dup);
    Serial.print(" retain: ");
    Serial.println(properties.retain);
    Serial.print(" len: ");
    Serial.println(len);
    Serial.print(" index: ");
    Serial.println(index);
    Serial.print(" total: ");
    Serial.println(total);
}
void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();
    pinMode(LED_BUILTIN, OUTPUT);

    wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
    wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);

    mqttClient.onConnect(onMqttConnect);
}

```

```

mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onSubscribe(onMqttSubscribe);
mqttClient.onUnsubscribe(onMqttUnsubscribe);
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT);
mqttClient.setCredentials(BROKER_USER, BROKER_PASS);

connectToWifi();
}

void loop() {
    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 10 seconds)
    // it publishes a new MQTT message
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;

        // Publish an MQTT message on topic counter
        uint16_t packetIdPub1 = mqttClient.publish("counter", 1, true,
String(random(0,20)).c_str());
        Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", "counter",
packetIdPub1);
    }
}

```

How the Code Works

Apart from small differences when it comes to the Wi-Fi events, the codes for the ESP32 and ESP8266 boards are similar.

Include Libraries

First, we need to include the required libraries. To set the board as an MQTT client, we're using the `AsyncMQTTClient` library.

The ESP32 uses the `WiFi` library to use WiFi, and the ESP8266 uses the `ESP8266WiFi` library. Those libraries are slightly different. To use timer events, the ESP8266 imports the `ticker` library, while the ESP32 uses `freertos` timers.

So, for the ESP32, we include the following:

```
#include <WiFi.h>
```

```
extern "C" {
    #include "freertos/FreeRTOS.h"
    #include "freertos/timers.h"
}
#include <AsyncMqttClient.h>
```

And for the ESP8266:

```
#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>
```

Wi-Fi Credentials and Broker Details

You need to include your network credentials on the following lines (SSID and password) so that your board can establish a Wi-Fi connection with your router.

```
#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"
```

You need to insert your MQTT broker IP address. It corresponds to the Raspberry Pi IP address:

```
#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106)*/
```

The MQTT broker port is 1883.

```
#define MQTT_PORT 1883
```

Insert your MQTT broker username and password on the following lines.

```
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"
```

AsyncMQTTClient

The following line creates an `AsyncMQTTClient` object called `mqttClient`.

```
AsyncMqttClient mqttClient;
```

Timers for Wi-Fi Events

Then, we create timers for the Wi-Fi events. This section is slightly different for each board, but they'll do the same task. This section is needed so that we can detect Wi-Fi and MQTT events like connecting and disconnecting and automatically call functions (asynchronously) when those events happen. The following lines are just creating the timers and handlers. This is how it's done with the ESP32:

```
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;
```

And this is how it's done with the ESP8266:

```
Ticker mqttReconnectTimer;

WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
Ticker wifiReconnectTimer;
```

Publish Interval

The following line creates auxiliary variables that will be used later to publish MQTT messages periodically. We'll send messages every 10 seconds (10000 milliseconds—**interval** variable).

```
unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values
```

Connect to Wi-Fi

The `connectToWiFi()` function will be called later to connect your ESP board to Wi-Fi.

```
void connectToWiFi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}
```

Connect to MQTT

As the name suggests, the `connectToMqtt()` function connects the MQTT client (your board) to the broker.

```
void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}
```

Wi-Fi Events

The ESP32 and ESP8266 are programmed differently to detect Wi-Fi events (connect and disconnect).

In the case of the ESP32, when there is a Wi-Fi event of type `ARDUINO_EVENT_WIFI_STA_GOT_IP` (which means that the ESP connected successfully to the router and was assigned an IP address on the network), we connect to the MQTT broker by calling the `connectToMQTT()` function.

When it detects Wi-Fi is disconnected, we stop reconnecting to the MQTT broker and start the timer to try to reconnect to Wi-Fi.

```
void WiFiEvent(WiFiEvent_t event) {
    Serial.printf("[WiFi-event] event: %d\n", event);
    switch(event) {
        case ARDUINO_EVENT_WIFI_STA_GOT_IP:
            Serial.println("WiFi connected");
            Serial.println("IP address: ");
            Serial.println(WiFi.localIP());
            connectToMqtt();
            break;
        case ARDUINO_EVENT_WIFI_STA_DISCONNECTED:
            Serial.println("WiFi lost connection");
            xTimerStop(mqttReconnectTimer, 0); // ensure we don't reconnect to MQTT while
            // reconnecting to Wi-Fi
            xTimerStart(wifiReconnectTimer, 0);
            break;
    }
}
```

We have an article about the ESP32 Wi-Fi functions and events that you can check on the following link:

- [ESP32 Useful Wi-Fi Library Functions](#)

In the case of the ESP8266, it works similarly, but the commands are different.

The `onWiFiConnect()` function will be called when the board connects successfully to Wi-Fi.

```
void onWifiConnect(const WiFiEventStationModeGotIP& event) {
    Serial.println("Connected to Wi-Fi.");
    connectToMqtt();
}
```

The `onWiFiDisconnect()` function will be called when the Wi-Fi connection is lost. In this case, it will disconnect from the broker, and start the timer to try to connect to Wi-Fi again.

```
void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
    Serial.println("Disconnected from Wi-Fi.");
    mqttReconnectTimer.detach(); // ensure we don't reconnect to MQTT while
reconnecting to Wi-Fi
    wifiReconnectTimer.once(2, connectToWifi);
}
```

onMQTTConnect()

The `onMQTTConnect()` function will be called when the board is connected successfully to the MQTT broker. In this example, we print a debugging message about the current MQTT session.

```
void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
```

Subscribe to Topics

If you want to subscribe to any topics, it's a good idea to do that right after the ESP connects to the broker, so put it inside the `onMQTTConnect()` function.

To subscribe to topics, simply call the `subscribe()` method on the `mqttClient` object. It accepts as arguments the topic you want to subscribe to and the QoS. This function outputs the *packet Id* (which is simply a number to identify the packet).

The following lines show how to subscribe to the `led` topic with QoS 2 and print the packet id.

```
// Subscribe to topic "led" when it connects to the broker
uint16_t packetIdSub = mqttClient.subscribe("led", 2);
Serial.print("Subscribing at QoS 2, packetId: ");
Serial.println(packetIdSub);
```

QoS

You may be wondering what is QoS and what exactly it means. QoS means quality of service and it is a way to guarantee that the MQTT message is delivered. It can be one of the following levels:

QoS 0 — at most once: There is no guarantee of delivery. The recipient does not acknowledge receipt of the message and the message is not stored or re-transmitted by the sender.

QoS 1— at least once: guarantees that each message is received only once by the intended recipients.

QoS 2—exactly once: the message is always delivered exactly once; QoS 2 is the safest and slowest quality of service level.

If you want to learn more about the details of QoS, we recommend reading the following article:

[Quality of Service \(QoS\) 0,1, & 2 MQTT Essentials](#)

We also publish three test messages with different levels of QoS to the `test` topic. To publish a message, simply call the `publish()` method on the `mqttClient` object. It accepts the following arguments in this order: **topic, QoS, retain** (true or false) and the message you want to send. The function returns the packet id.

```

// Publish on the "test" topic with qos 0
mqttClient.publish("test", 0, true, "test 1");
Serial.println("Publishing at QoS 0");
// Publish on the "test" topic with qos 1
uint16_t packetIdPub1 = mqttClient.publish("test", 1, true, "test 2");
Serial.print("Publishing at QoS 1, packetId: ");
Serial.println(packetIdPub1);
// Publish on the "test" topic with qos 2
uint16_t packetIdPub2 = mqttClient.publish("test", 2, true, "test 3");
Serial.print("Publishing at QoS 2, packetId: ");
Serial.println(packetIdPub2);

```

retain (true or false) what does it mean?

A retained message is a normal MQTT message with the retained flag set to true. The broker stores the last retained message and the corresponding QoS for that topic. Each client that subscribes to a topic pattern that matches the topic of the retained message receives the retained message immediately after they subscribe. The broker stores only one retained message per topic.

Why is it useful? Retained messages help newly-subscribed clients get a status update immediately after they subscribe to a topic. This is particularly useful to get the status of a device. For example, GPIO 2 is currently HIGH, and the ESP32 suddenly resets. It doesn't know what the last state of GPIO 2 was. However, if the state is a retained message, it will receive it right after subscribing to the topic and can update the state of GPIO 2 immediately.

Learn more: [Retained Messages](#)

onMQTTDisconnect()

The `onMQTTDisconnect()` function will be called when the broker suddenly disconnects. When that happens, we try to reconnect. These functions are slightly different depending on the board.

This is for the ESP32

```

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
  Serial.println("Disconnected from MQTT.");
  if (WiFi.isConnected()) {
    xTimerStart(mqttReconnectTimer, 0);

```

```
}
```

And this is for the ESP8266

```
void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");
    if (WiFi.isConnected()) {
        mqttReconnectTimer.once(2, connectToMqtt);
    }
}
```

onMqttSubscribe()

The `onMQTTSubscribe()` function will run when you subscribe to a topic. In this example, it just prints messages about the packet ID and QoS, but you can add other tasks.

```
void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}
```

onMqttUnsubscribe()

The `onMqttUnsubscribe()` message will run when you unsubscribe from a topic. In this case, it simply prints information about the packet id.

```
void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}
```

onMqttMessage()

The `onMqttMessage()` runs when you receive a message on a topic that the ESP is subscribed to.

```
void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties
properties, size_t len, size_t index, size_t total) {
```

Inside this function, you can decide what to do with the message (save it in a global variable, perform a certain task depending on the message content, etc.).

In our case, we save the message in the `receivedMessage` variable.

```
// Do whatever you want when you receive a message
// Save the message in a variable
String receivedMessage;
for (int i = 0; i < len; i++) {
    Serial.println((char)payload[i]);
    receivedMessage += (char)payload[i];
}
```

Then, we check in which topic the message was published. The topic is automatically passed to the function in the `topic` variable.

We use the `strcmp()` function to check if the message we received is on the `led` topic or not.

```
if (strcmp(topic, "led") == 0) {
```

The `strcmp()` compares two strings character by character. If the strings are equal, the function returns 0.

If the received message is `true`, we turn the ESP built-in LED on.

```
if (receivedMessage == "true"){
    digitalWrite(LED_BUILTIN, HIGH);
}
```

If the message is `false`, we turn the built-in LED off.

```
if (receivedMessage == "false"){
    digitalWrite(LED_BUILTIN, LOW);
}
```

We also print other information about the message: topic, QoS, retain, index and size.

```
Serial.println("Publish received.");
Serial.print(" topic: ");
Serial.println(topic);
Serial.print(" qos: ");
Serial.println(properties.qos);
```

```
Serial.print(" dup: ");
Serial.println(properties.dup);
Serial.print(" retain: ");
Serial.println(properties.retain);
Serial.print(" len: ");
Serial.println(len);
Serial.print(" index: ");
Serial.println(index);
Serial.print(" total: ");
Serial.println(total);
```

onMqttPublish()

The `onMqttPublish()` message runs when we publish a message on a topic. In this case, it simply prints the packet id in the Serial Monitor. You can add other tasks that may make sense for your project.

```
void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}
```

setup()

In the `setup()`, we initialize the Serial Monitor at a baud rate of 115200.

```
void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();
```

Set the built-in LED as an output.

```
pinMode(LED_BUILTIN, OUTPUT);
```

ESP32

The following lines create the `mqttReconnectTimer` and the `wifiReconnectTimer` tasks that will run when the MQTT or Wi-Fi disconnects, respectively.

```
mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));
```

Finally, add the WiFiEvent function to the `onEvent` Wi-Fi handler. This means that whenever a Wi-Fi event happens, the `WiFiEvent()` function will run.

```
WiFi.onEvent(WiFiEvent);
```

ESP8266

In the case of the ESP8266, that's how we create the handlers for when the Wi-Fi is connected or disconnected.

```
wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);
```

Assigning callback functions

The AsyncMQTT is an asynchronous library. This means that it works with events and callback functions. You create the callback functions and then, assign them to a certain event. When that event happens, the board will automatically run the corresponding callback function. This means that you can be running other tasks on the `loop()` and whenever one of the events happens, the corresponding callback functions will run.

Important: one thing you need to keep in mind when creating the callback functions (all those functions we have taken a look at previously) is to never use blocking functions inside them like `delay()` or `yield()` or methods that use `delay()` or `yield()`.

On the following lines, we assign the callback functions to their corresponding events.

```
mqttClient.onConnect(onMqttConnect);
mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onSubscribe(onMqttSubscribe);
mqttClient.onUnsubscribe(onMqttUnsubscribe);
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
```

Set the MQTT Server and Credentials

The next two lines set the MQTT host and port, and set the credentials.

```
mqttClient.setServer(MQTT_HOST, MQTT_PORT);
mqttClient.setCredentials(BROKER_USER, BROKER_PASS);
```

Connect to Wi-Fi

Finally, we connect to Wi-Fi by calling the `connectToWiFi()` function.

```
connectToWiFi();
```

loop()

In the `loop()`, we send a random number to the `counter` topic every 10000 milliseconds (`interval` variable).

```
void loop() {
    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 10 seconds)
    // it publishes a new MQTT message
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;

        // Publish an MQTT message on topic esp/bme680/temperature
        uint16_t packetIdPub1 = mqttClient.publish("counter", 1, true,
String(random(0,20)).c_str());
        Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", "counter",
packetIdPub1);
    }
}
```

That's pretty much how the code works.

Uploading the Code

After inserting your network credentials and MQTT broker details, you can upload the code to your board.

First, go to **Tools > Board** and select the board you're using. Then, select the COM port in **Tools > Port**. Finally, press the **Upload** button.



After successfully uploading the code to the board, open the Serial Monitor at a baud rate of 115200.



Press the board RST button to restart it and start running the code.

You should see the ESP board successfully subscribing to the `led` topic and publishing on the `test` and `counter` topics.

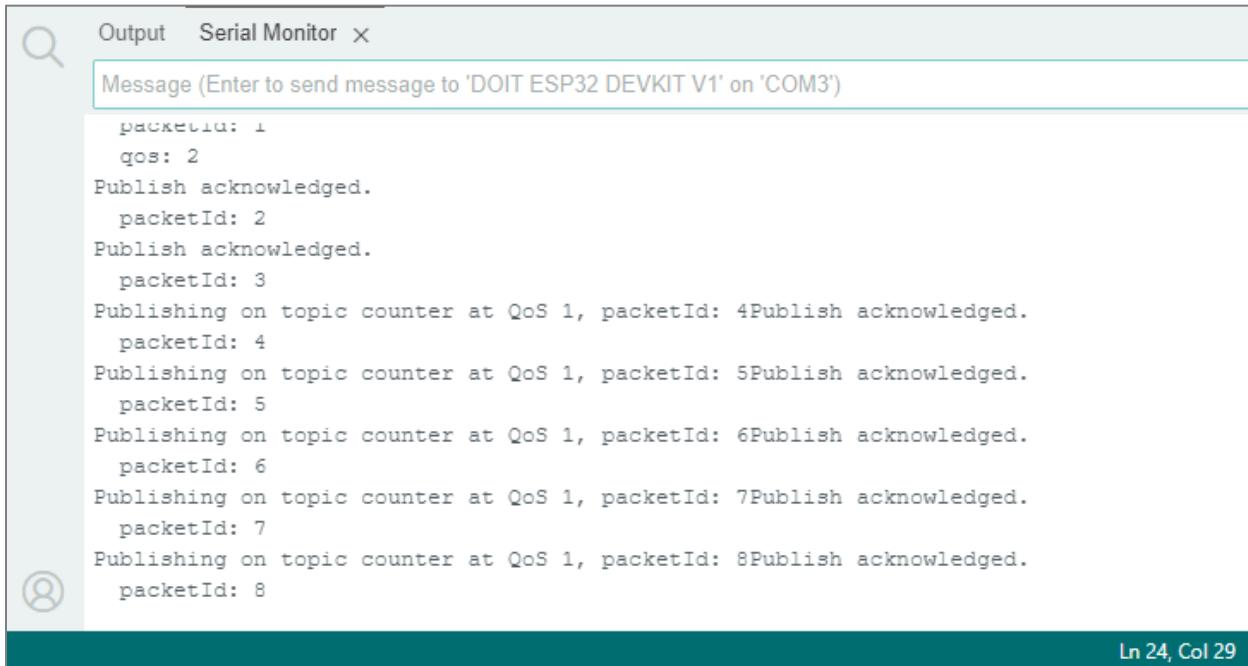


The screenshot shows the Arduino Serial Monitor window. The title bar says "Output Serial Monitor". The main area displays the following log output:

```
[WiFi-event] event: 0
[WiFi-event] event: 2
[WiFi-event] event: 4
[WiFi-event] event: 7
WiFi connected
IP address:
192.168.1.90
Connecting to MQTT...
Connected to MQTT.
Session present: 0
Subscribing at QoS 2, packetId: 1
Publishing at QoS 0
Publishing at QoS 1, packetId: 2
Publishing at QoS 2, packetId: 3
Subscribe acknowledged.
  nectack. 1
```

At the bottom right of the monitor window, there is a status bar with "Ln 24, Col 29".

Let it run for a while to check if everything is working as expected.



The screenshot shows a Serial Monitor window with the following text output:

```
Output Serial Monitor x
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM3')
packetId: 1
qos: 2
Publish acknowledged.
packetId: 2
Publish acknowledged.
packetId: 3
Publishing on topic counter at QoS 1, packetId: 4Publish acknowledged.
packetId: 4
Publishing on topic counter at QoS 1, packetId: 5Publish acknowledged.
packetId: 5
Publishing on topic counter at QoS 1, packetId: 6Publish acknowledged.
packetId: 6
Publishing on topic counter at QoS 1, packetId: 7Publish acknowledged.
packetId: 7
Publishing on topic counter at QoS 1, packetId: 8Publish acknowledged.
packetId: 8
```

Ln 24, Col 29

Then, we'll create a Node-RED flow that will read the messages from the ESP (test and counter topics) and publish to the topic (led) that the ESP is subscribed to.

Creating the Node-RED Flow

We'll create a simple flow that receives the ESP messages (test and counter topics) and sends messages to the ESP (led topic).

You can download the flow and import it to Node-RED. Don't forget you'll need to modify the broker node to insert your details.

- [Download Node-RED flow](#)

However, we recommend following the step-by-step instructions instead to get you more familiar with creating Node-RED flows.

Open your browser and go to your Node-RED page.

`http://Your_RPi_IP_address:1880`

For example, in my case:

<http://192.168.1.106:1880>

Drag **two MQTT in** nodes and one **MQTT out** node.

Edit one **MQTT in** node to subscribe to the **test** topic. See the picture below. You should have already set up the MQTT broker in previous units.

Edit mqtt in node

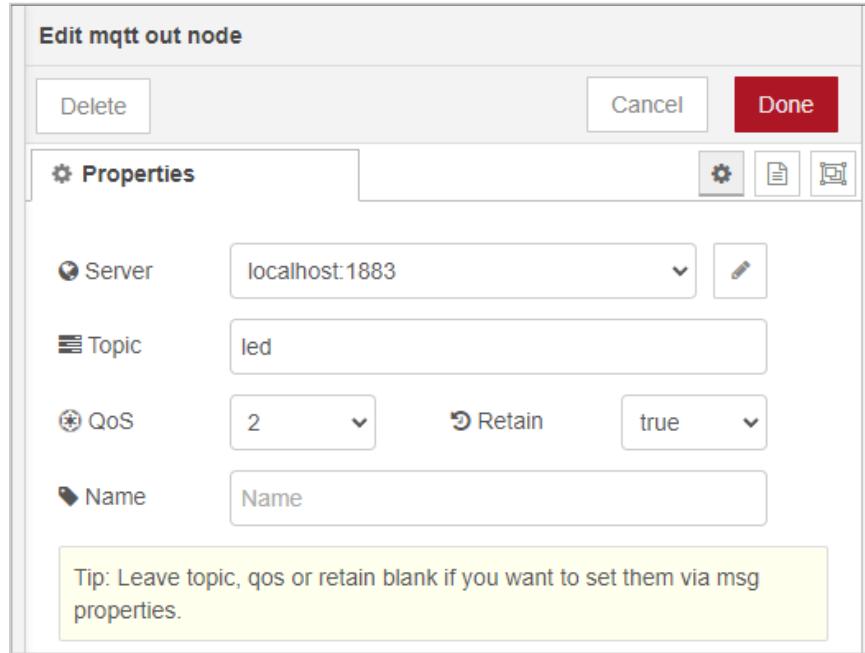
| | | |
|--|---------------------------------------|---|
| <input type="button" value="Delete"/> | <input type="button" value="Cancel"/> | <input style="background-color: red; color: white; font-weight: bold;" type="button" value="Done"/> |
| Properties | | |
| <input checked="" type="radio"/> Server | localhost:1883 | <input type="button" value=""/> |
| Action | Subscribe to single topic | <input type="button" value=""/> |
| <input checked="" type="radio"/> Topic | test | |
| <input checked="" type="radio"/> QoS | 2 | <input type="button" value=""/> |
| Output | auto-detect (string or buffer) | <input type="button" value=""/> |
| <input checked="" type="radio"/> Name | Name | |

Edit the other **MQTT in** node to subscribe to the **counter** topic. See the picture below.

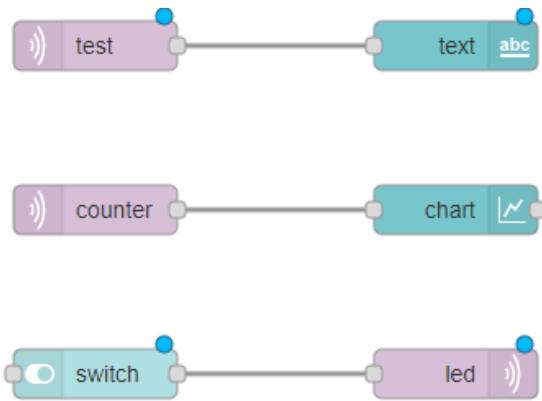
Edit mqtt in node

| | | |
|--|---------------------------------------|---|
| <input type="button" value="Delete"/> | <input type="button" value="Cancel"/> | <input style="background-color: red; color: white; font-weight: bold;" type="button" value="Done"/> |
| Properties | | |
| <input checked="" type="radio"/> Server | localhost:1883 | <input type="button" value=""/> |
| Action | Subscribe to single topic | <input type="button" value=""/> |
| <input checked="" type="radio"/> Topic | counter | |
| <input checked="" type="radio"/> QoS | 2 | <input type="button" value=""/> |
| Output | auto-detect (string or buffer) | <input type="button" value=""/> |
| <input checked="" type="radio"/> Name | Name | |

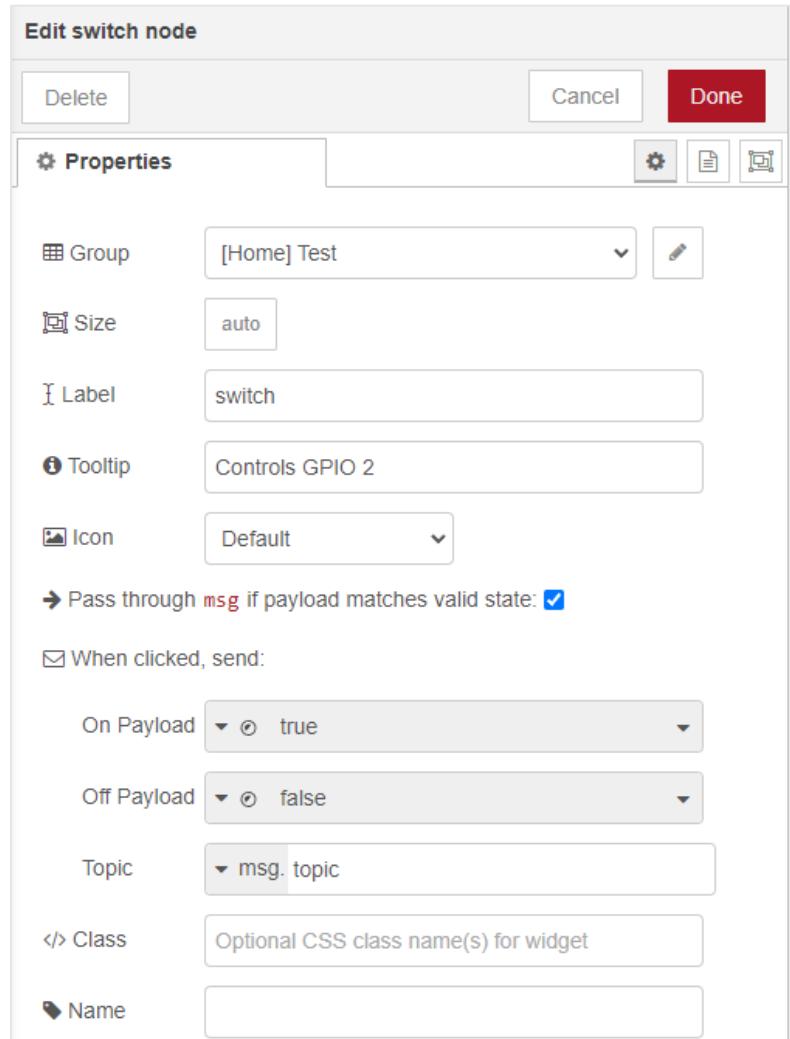
Edit the **MQTT out** node to publish to the `led` topic, set retained to `true`, and select QoS 2.



Then, drag a **switch** node, a **text** node, and a **chart** node. Wire the nodes as follows.



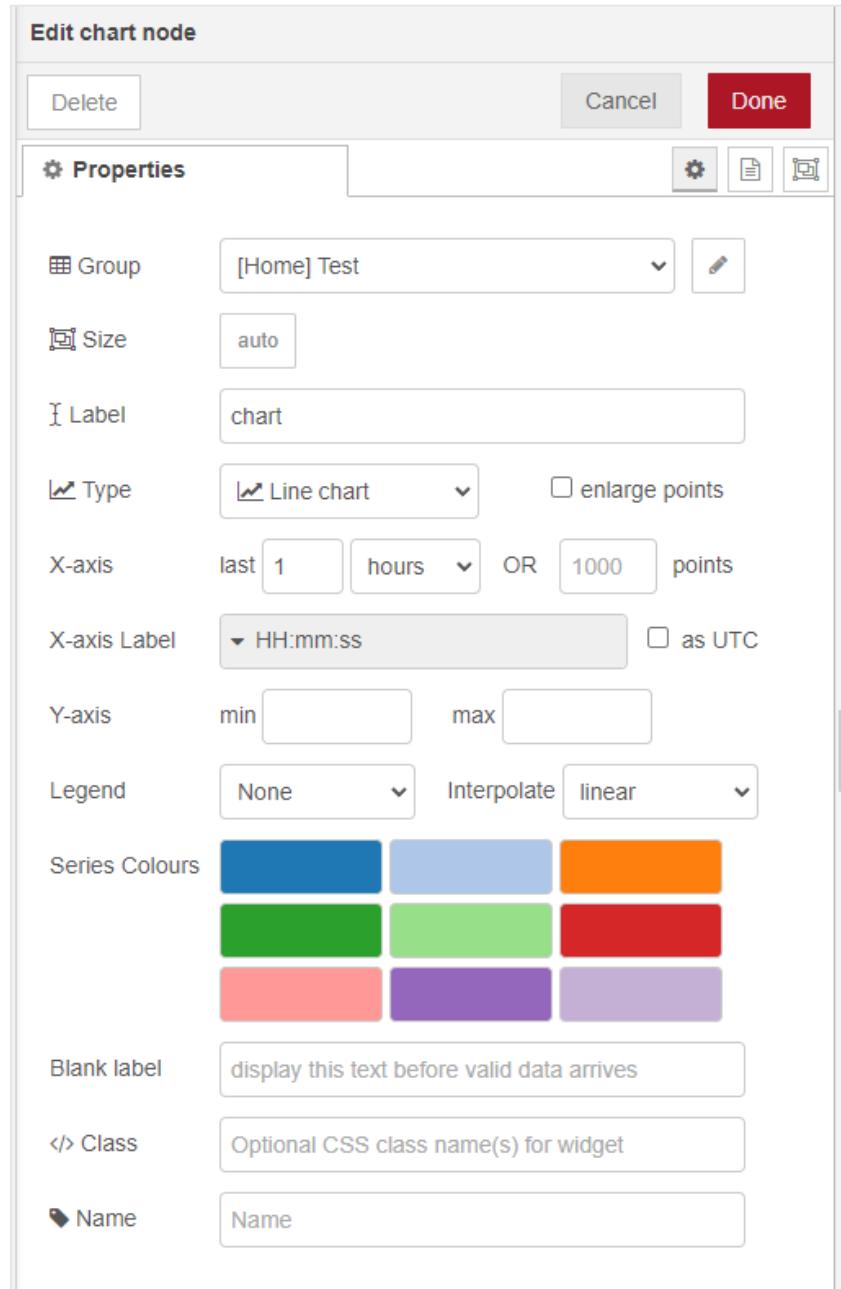
Edit the dashboard nodes to assign them to a specific dashboard and group (see unit 2.4 if you don't remember how to do that). For example, here are the properties of my **switch** node.



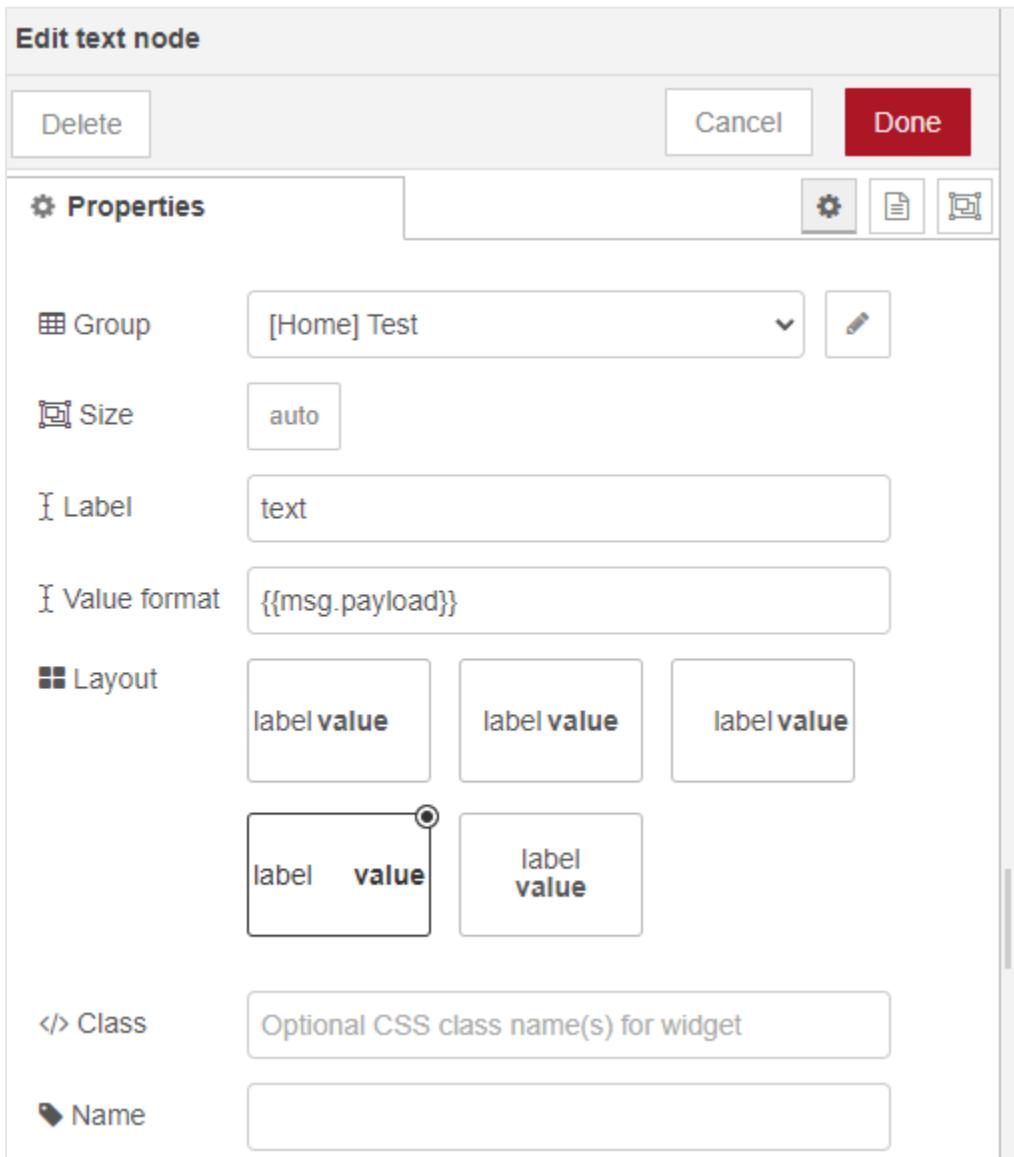
Note: the ESP8266 built-in LED works with inverted logic. It turns on with a LOW signal and turns OFF with a HIGH signal. So, if you're using an ESP8266, you should change the On Payload message to **false**, and the Off Payload message to **true**.

You can change the other settings like the label, tooltip, topic, etc.

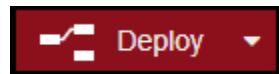
Here are the settings for the **chart** node. Again you can change its properties as you wish.



And finally, the **text** node. It simply prints text with a label that you define on the dashboard.



When you're done, **Deploy** your application.



Testing the Example

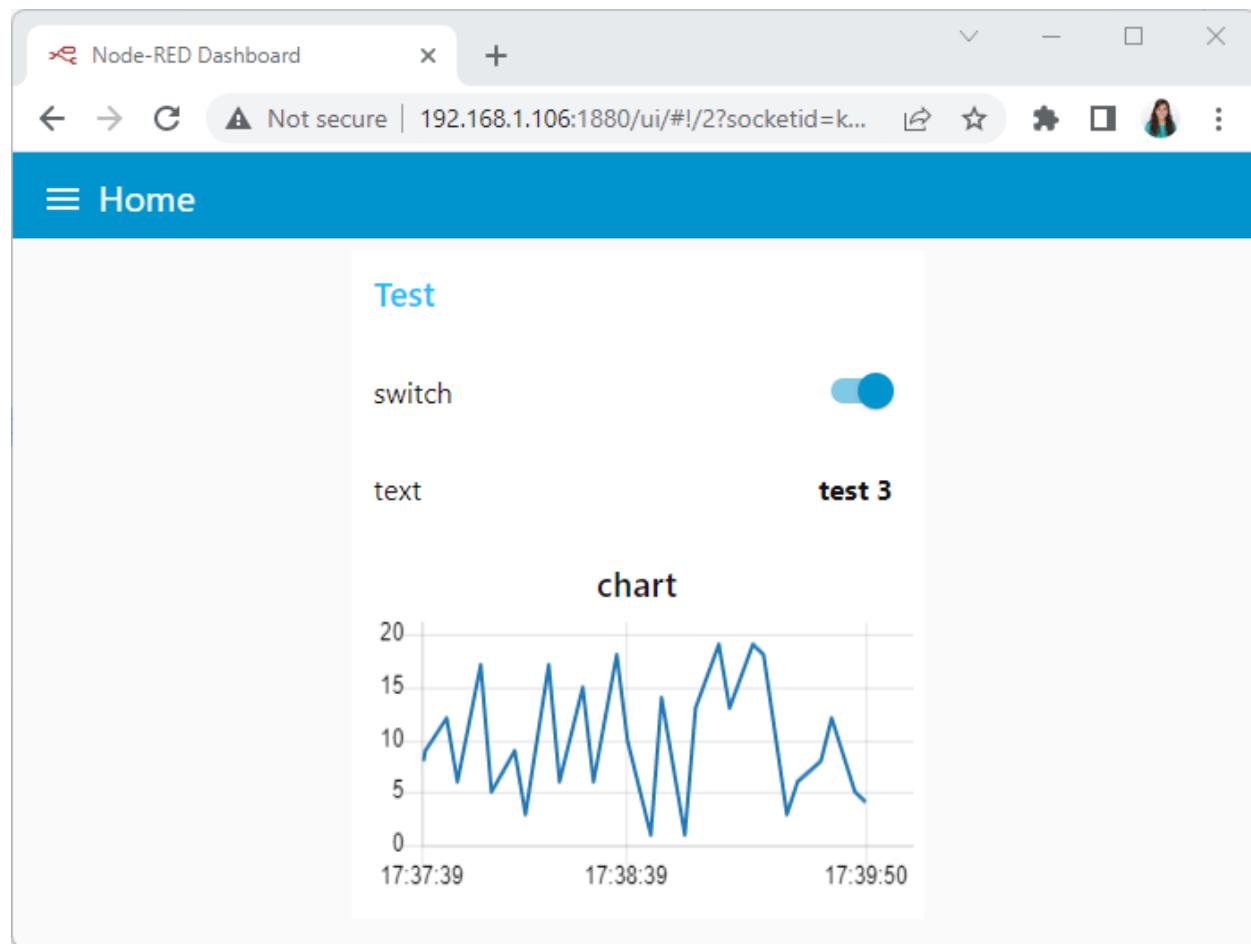
Go to the Node-RED UI. The user interface is accessible on the following URL:

`http://Your_RPi_IP_address:1880/ui`

For example, in my case:

`http://192.168.1.106:1880/ui`

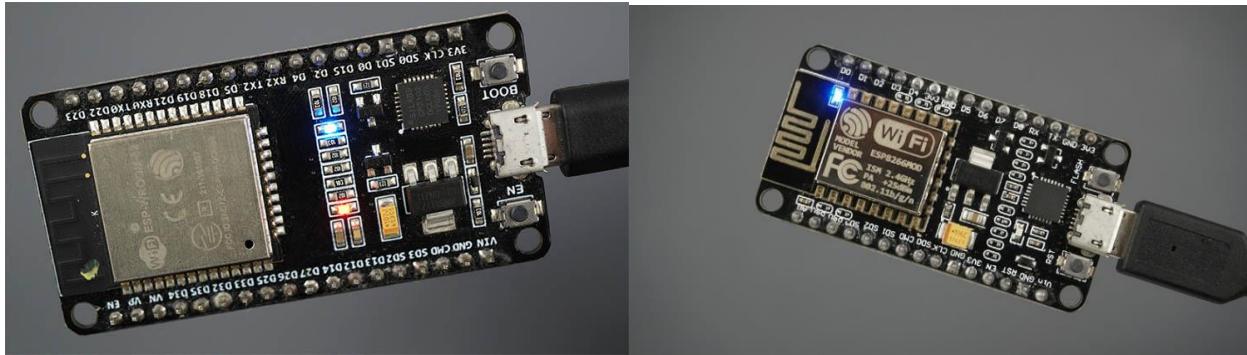
Your user interface should look as follows:



The chart should be receiving the random values published by the ESP on the `counter` topic. On the `text` label, you should have the message published on the `test` topic.

Click on the switch. It should turn the ESP built-in LED on and off.

The following pictures show the ESP32 and ESP8266 with the built-in LED on.



Testing Retained Messages

To test if the retained messages are working as expected. Turn the LED on using the switch widget. Reset the ESP by clicking on the RST button or remove and apply power. A few seconds after restarting, the LED should turn on automatically. This happens because the message published on the `led` topic has the retain flag set to true. So, all devices that subscribe to that topic will automatically receive the last message published on that topic.

Wrapping Up

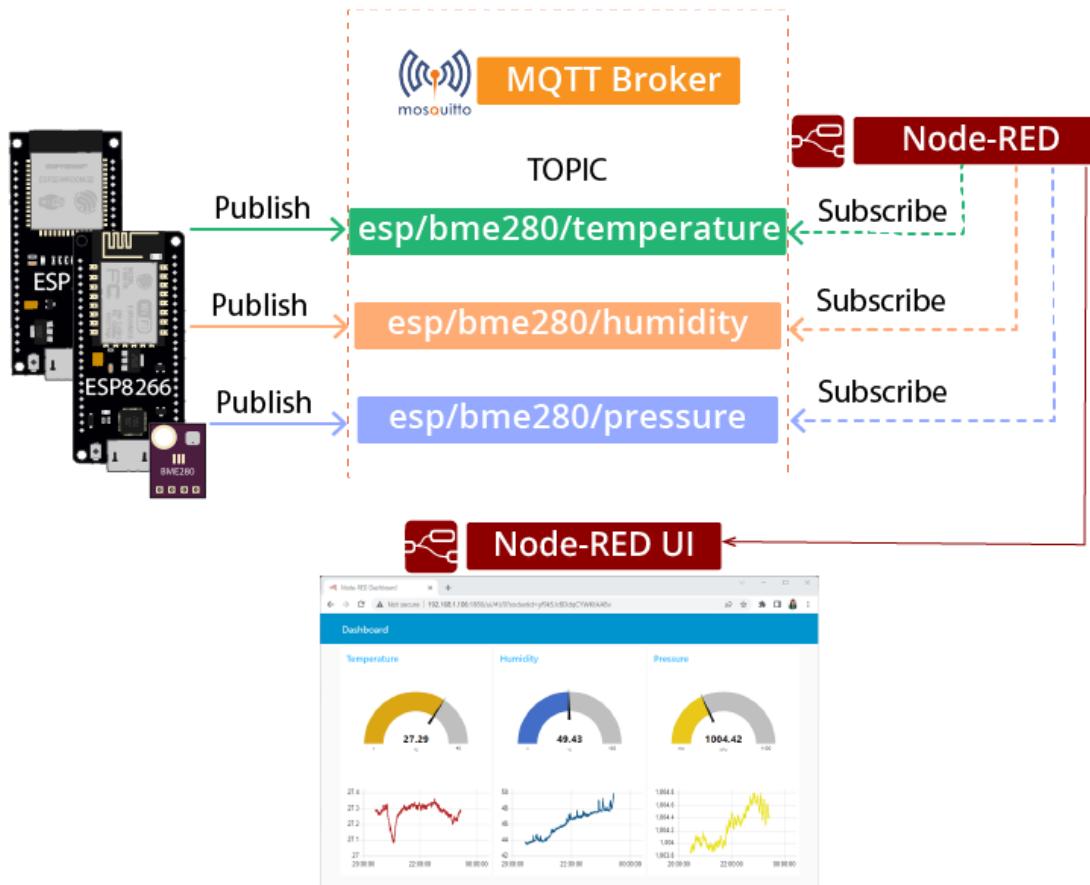
Congratulations! In this unit, you learned how to establish a communication with Node-RED and the ESP32/ESP8266 using MQTT. By now, you have acquired all the basic concepts. In this example, we used sample messages. The idea is to replace those messages with sensor readings or other useful data—that's what we're going to do in the next unit.

5.3 - Publish Sensor Readings

In this section, you'll send sensor readings periodically from the ESP32 or ESP8266 to Node-RED via MQTT communication protocol. As an example, we'll use a BME280 sensor, and we'll send temperature, humidity, and pressure readings. Finally, we'll create an interface using Node-RED dashboard to display the readings on gauges and charts.

Project Overview

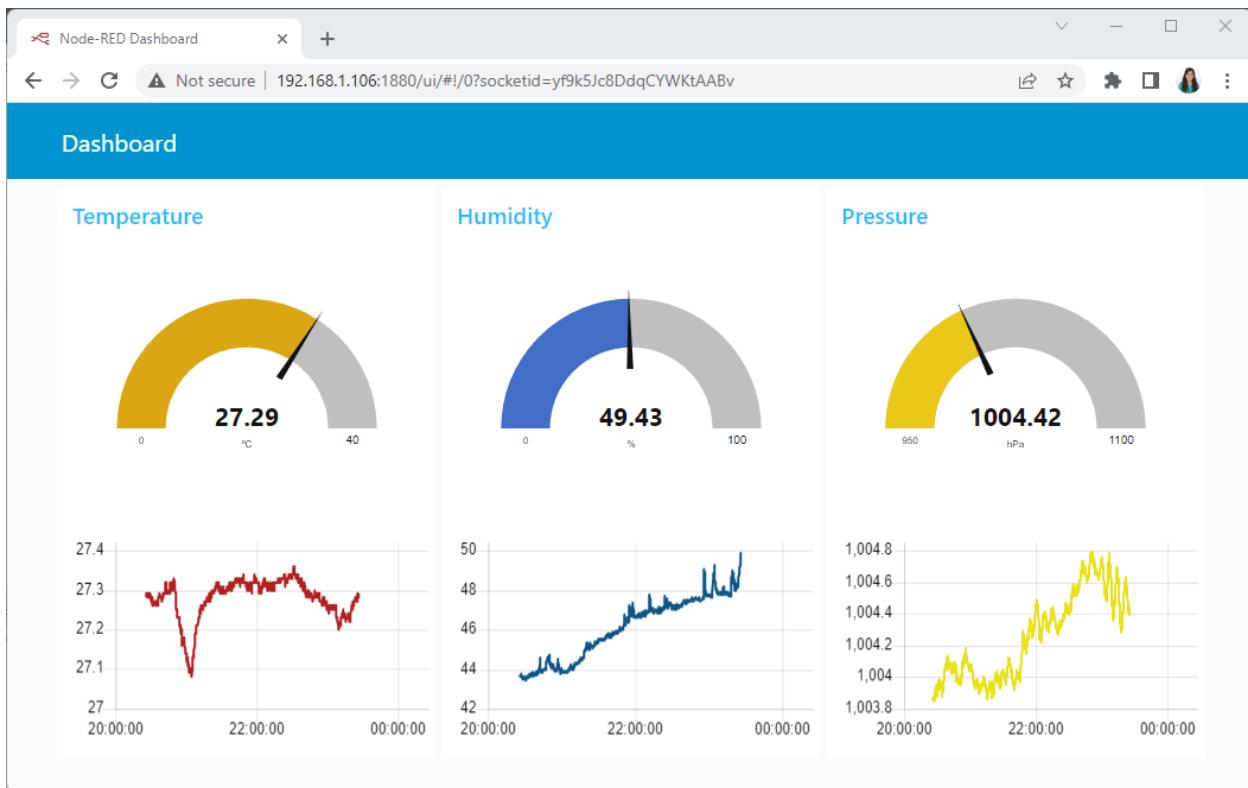
Here's an overview of the project we'll build:



- The ESP32/ESP8266 requests temperature, humidity, and pressure readings from the BME280 sensor;

- The temperature readings are published in the following topic: `esp/bme280/temperature`;
- Humidity readings are published in the `esp/bme280/humidity` topic;
- Pressure readings are published in the `esp/bme280/pressure` topic;
- Node-RED is subscribed to those topics;
- Node-RED receives the sensor readings and displays them on gauges and charts (Node-RED UI).

The following figure shows the dashboard you'll build:



Parts Required

To complete this project, you need the following parts:

- [ESP32](#) or [ESP8266](#)
- [BME280](#) (you can use any other sensor you're familiar with as long as you make the needed modifications in the code.)

- [Jumper wires](#)
- [Breadboard](#)

Wiring the Circuit

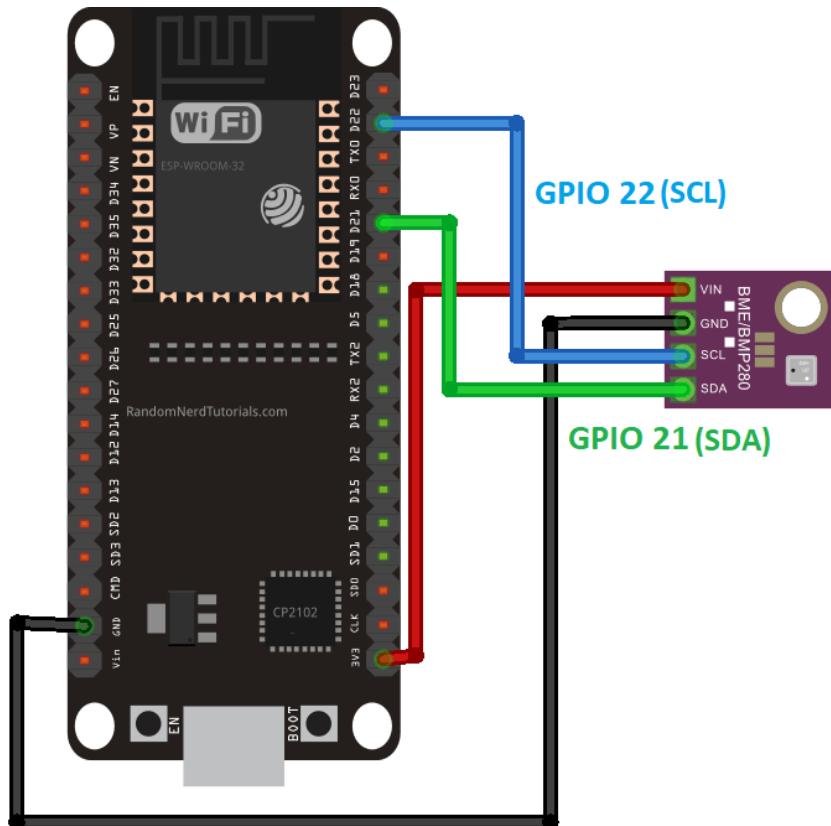
The BME280 sensor uses I2C communication protocol. Wire the BME280 sensor to your ESP32 or ESP8266 on the I2C pins:

| | | |
|----------------|-------------------------|-------------------------|
| ESP32 | SDA: GPIO 21 | SCL: GPIO22 |
| ESP8266 | SDA: GPIO 4 (D2) | SCL: GPIO 5 (D1) |

You can also use one of the following diagrams as a reference.

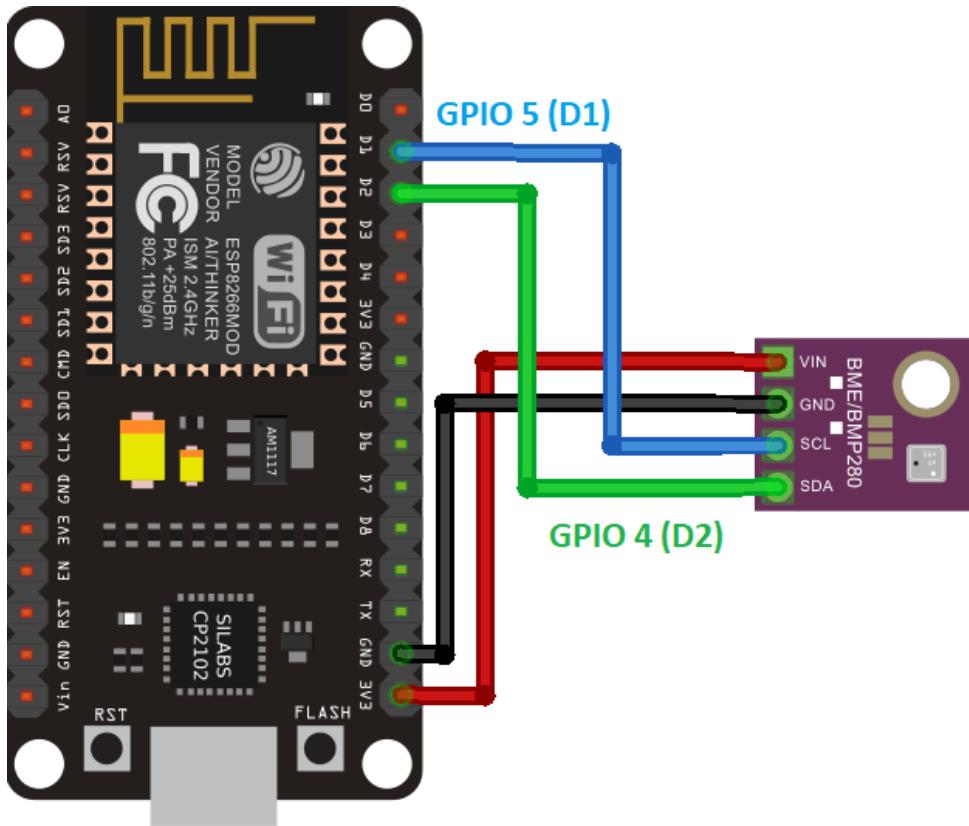
ESP32

Wire the BME280 sensor to your ESP32 board as shown in the following schematic diagram.



ESP8266

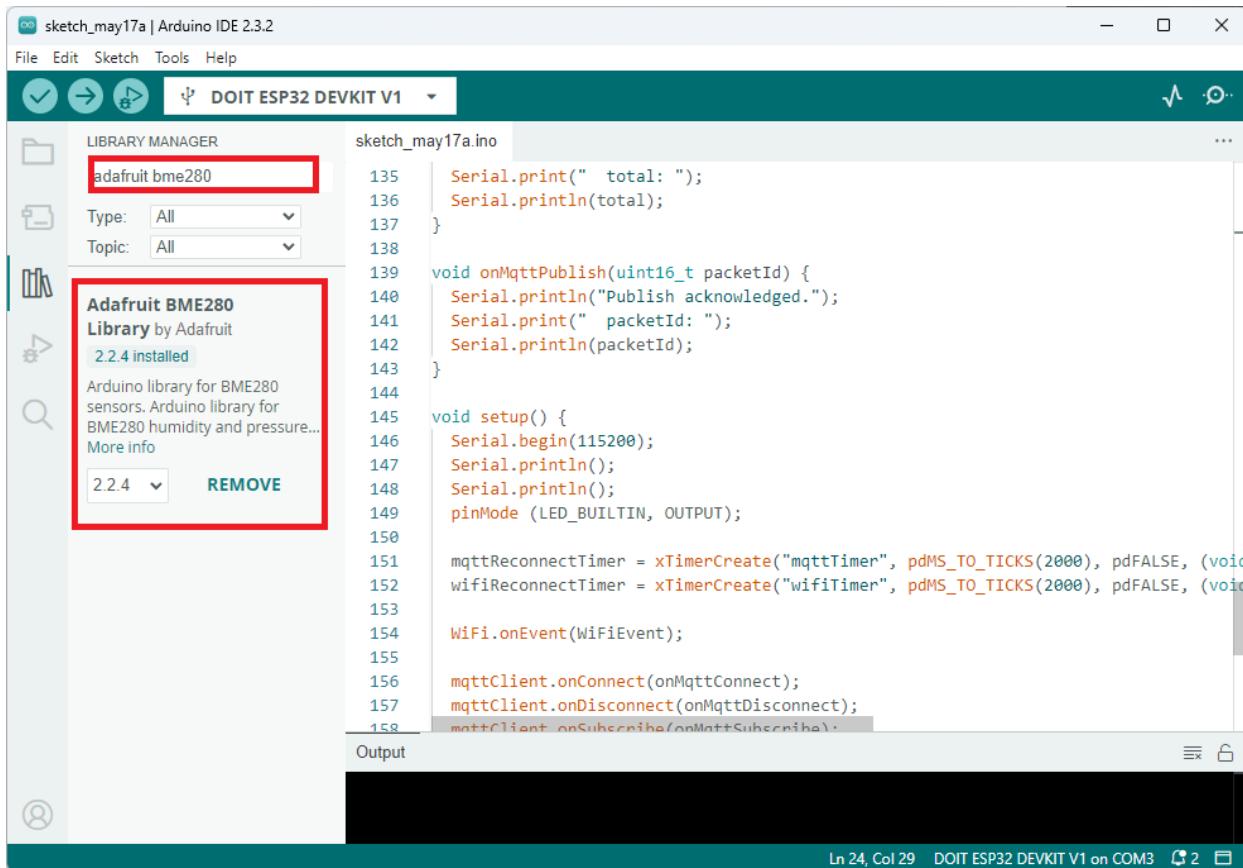
If you're using an ESP8266, follow the next diagram instead.



Installing Required Libraries

There are different ways to get data from the BME280 sensor. We'll use the [Adafruit BME280 library](#). Follow the next steps to install the library.

- 1) In the Arduino IDE, go to **Sketch > Include Library > Manage Libraries**. The Library Manager should open.
- 2) Search for **adafruit bme280** on the search box and install the library. Also install any dependencies currently not installed (usually the Adafruit Bus IO and the Adafruit Unified Sensor libraries).



3) After installing the libraries, restart your Arduino IDE.

ESP32/ESP8266 Publish Sensor Readings – Code

The code is slightly different for each board. Upload the correct code for the board you're using.

- [Download Sketch folder ESP32](#)
- [Download Sketch folder ESP8266](#)

ESP32 – Code

Here's the code for the ESP32. You need to insert your network credentials (SSID and password), the broker IP address, and the broker username and password. The lines where you need to insert your details are highlighted in light green.

```

#include <Arduino.h>

#include <WiFi.h>
extern "C" {
    #include "freertos/FreeRTOS.h"
    #include "freertos/timers.h"
}
#include <AsyncMqttClient.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// BROKER
#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106) */
#define MQTT_PORT 1883
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"

//MQTT Topics
#define MQTT_PUB_TEMP "esp/bme280/temperature"
#define MQTT_PUB_HUM "esp/bme280/humidity"
#define MQTT_PUB_PRES "esp/bme280/pressure"

// BME280 I2C
Adafruit_BME280 bme;

// Variables to hold sensor readings
float temp;
float hum;
float pres;

AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;

unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

void WiFiEvent(WiFiEvent_t event) {
    Serial.printf("[WiFi-event] event: %d\n", event);
    switch(event) {
        case ARDUINO_EVENT_WIFI_STA_GOT_IP:
            Serial.println("WiFi connected");
            Serial.println("IP address: ");

```

```

        Serial.println(WiFi.localIP());
        connectToMqtt();
        break;
    case ARDUINO_EVENT_WIFI_STA_DISCONNECTED:
        Serial.println("WiFi lost connection");
        xTimerStop(mqttReconnectTimer, 0); // ensure we don't reconnect to MQTT
while reconnecting to Wi-Fi
        xTimerStart(wifiReconnectTimer, 0);
        break;
    }
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        xTimerStart(mqttReconnectTimer, 0);
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message
    Serial.println("Publish received.");
    Serial.print(" topic: ");
    Serial.println(topic);
    Serial.print(" qos: ");
    Serial.println(properties.qos);
    Serial.print(" dup: ");
    Serial.println(properties.dup);
    Serial.print(" retain: ");
    Serial.println(properties.retain);
    Serial.print(" len: ");
    Serial.println(len);
    Serial.print(" index: ");
    Serial.println(index);
    Serial.print(" total: ");
    Serial.println(total);
}

```

```

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();

    // Initialize BME280 sensor
    if (!bme.begin(0x76)) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
        while (1);
    }

    mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
    wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));

    WiFi.onEvent(WiFiEvent);

    mqttClient.onConnect(onMqttConnect);
    mqttClient.onDisconnect(onMqttDisconnect);
    mqttClient.onSubscribe(onMqttSubscribe);
    mqttClient.onUnsubscribe(onMqttUnsubscribe);
    mqttClient.onMessage(onMqttMessage);
    mqttClient.onPublish(onMqttPublish);
    mqttClient.setServer(MQTT_HOST, MQTT_PORT);
    mqttClient.setCredentials(BROKER_USER, BROKER_PASS);

    connectToWifi();
}

void loop() {

    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 10 seconds)
    // it publishes a new MQTT message
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;

        // New BME280 sensor readings
        temp = bme.readTemperature();
        //temp = 1.8*bme.readTemperature() + 32;
        hum = bme.readHumidity();
        pres = bme.readPressure()/100.0F;

        // Publish an MQTT message on topic esp/BME2800/temperature
        uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
        Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_TEMP, packetIdPub1);
        Serial.printf("Message: %.2f \n", temp);
    }
}

```

```

    // Publish an MQTT message on topic esp/BME2800/humidity
    uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(hum).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", MQTT_PUB_HUM, packetIdPub2);
    Serial.printf("Message: %.2f \n", hum);

    // Publish an MQTT message on topic esp/BME2800/pressure
    uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_PRES, 1, true,
String(pres).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", MQTT_PUB_PRES, packetIdPub3);
    Serial.printf("Message: %.3f \n", pres);
}
}

```

ESP8266 - Code

Here's the code for the ESP8266. You need to insert your network credentials (SSID and password), the broker IP address, and the broker username and password. The lines where you need to insert your details are highlighted in light green.

```

#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// BROKER
#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106) */
#define MQTT_PORT 1883
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"

//MQTT Topics
#define MQTT_PUB_TEMP "esp/bme280/temperature"
#define MQTT_PUB_HUM "esp/bme280/humidity"
#define MQTT_PUB_PRES "esp/bme280/pressure"

// BME280 I2C
Adafruit_BME280 bme;

// Variables to hold sensor readings
float temp;
float hum;

```

```

float pres;

AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;

WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
Ticker wifiReconnectTimer;

unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

void onWifiConnect(const WiFiEventStationModeGotIP& event) {
    Serial.println("Connected to Wi-Fi.");
    connectToMqtt();
}

void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
    Serial.println("Disconnected from Wi-Fi.");
    mqttReconnectTimer.detach(); // ensure we don't reconnect to MQTT while
reconnecting to Wi-Fi
    wifiReconnectTimer.once(2, connectToWifi);
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        mqttReconnectTimer.once(2, connectToMqtt);
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
}

```

```

    Serial.println(packetId);
}

void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message
    Serial.println("Publish received.");
    Serial.print(" topic: ");
    Serial.println(topic);
    Serial.print(" qos: ");
    Serial.println(properties.qos);
    Serial.print(" dup: ");
    Serial.println(properties.dup);
    Serial.print(" retain: ");
    Serial.println(properties.retain);
    Serial.print(" len: ");
    Serial.println(len);
    Serial.print(" index: ");
    Serial.println(index);
    Serial.print(" total: ");
    Serial.println(total);
}

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();

    // Initialize BME280 sensor
    if (!bme.begin(0x76)) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
        while (1);
    }
}

wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);

mqttClient.onConnect(onMqttConnect);
mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onSubscribe(onMqttSubscribe);
mqttClient.onUnsubscribe(onMqttUnsubscribe);
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT);
mqttClient.setCredentials(BROKER_USER, BROKER_PASS);

connectToWifi();
}

void loop() {
    unsigned long currentMillis = millis();
}

```

```

// Every X number of seconds (interval = 10 seconds)
// it publishes a new MQTT message
if (currentMillis - previousMillis >= interval) {
    // Save the last time a new reading was published
    previousMillis = currentMillis;

    // New BME280 sensor readings
    temp = bme.readTemperature();
    //temp = 1.8*bme.readTemperature() + 32;
    hum = bme.readHumidity();
    pres = bme.readPressure()/100.0F;

    // Publish an MQTT message on topic esp/BME2800/temperature
    uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_TEMP, packetIdPub1);
    Serial.printf("Message: %.2f \n", temp);

    // Publish an MQTT message on topic esp/BME2800/humidity
    uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(hum).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ",
MQTT_PUB_HUM, packetIdPub2);
    Serial.printf("Message: %.2f \n", hum);

    // Publish an MQTT message on topic esp/BME2800/pressure
    uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_PRES, 1, true,
String(pres).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_PRES, packetIdPub3);
    Serial.printf("Message: %.3f \n", pres);
}
}

```

How the Code Works

The code is very similar to the previous example. So, we'll just take a look at the new parts. The new parts are highlighted in a light orange color.

Include Libraries

First, we need to include the libraries to read from the BME280.

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

MQTT Topics

Define the topics that you want your board to publish. In this case, we'll publish on the following topics to send temperature, humidity, and pressure.

```
//MQTT Topics
#define MQTT_PUB_TEMP "esp/bme280/temperature"
#define MQTT_PUB_HUM "esp/bme280/humidity"
#define MQTT_PUB_PRES "esp/bme280/pressure"
```

BME280 Sensor

Then, initialize an `Adafruit_BME280` object called `bme`. This creates an instance that we'll use to refer to the sensor. It automatically assigns the right I2C pins depending on the board you're using.

```
Adafruit_BME280 bme;
```

The `temp`, `hum`, and `pres` variables will hold the temperature, humidity, and pressure values from the BME280 sensor.

```
float temp;
float hum;
float pres;
```

Initialize the Sensor

In the `setup()`, you need to initialize your sensor. The following lines initialize the sensor and print an error message in case the board can't establish a connection with the sensor.

```
// Initialize BME280 sensor
if (!bme.begin(0x76)) {
    Serial.println("Could not find a valid BME280 sensor, check wiring!");
    while (1);
}
```

Get Sensor Readings

In the `loop()`, we get sensor readings every 10 seconds (`interval` variable). You can increase that value after checking that everything is working as expected. The readings are saved on the corresponding variables `temp`, `hum`, and `pres`.

```
unsigned long currentMillis = millis();
// Every X number of seconds (interval = 10 seconds)
// it publishes a new MQTT message
if (currentMillis - previousMillis >= interval) {
    // Save the last time a new reading was published
    previousMillis = currentMillis;

    // New BME280 sensor readings
    temp = bme.readTemperature();
    //temp = 1.8*bme.readTemperature() + 32;
    hum = bme.readHumidity();
    pres = bme.readPressure()/100.0F;
```

Publish to Topics

After getting the readings, we publish them in the corresponding topics. As we've seen previously, to publish a message, you just need to use the `publish()` method on the `mqttClient` object. It accepts as arguments: the topic, QoS, retain (`true` or `false`), and the message.

```
// Publish an MQTT message on topic esp32/BME2800/temperature
uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", MQTT_PUB_TEMP,
packetIdPub1);
Serial.printf("Message: %.2f \n", temp);

// Publish an MQTT message on topic esp32/BME2800/humidity
uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(hum).c_str());
Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", MQTT_PUB_HUM,
packetIdPub2);
Serial.printf("Message: %.2f \n", hum);

// Publish an MQTT message on topic esp32/BME2800/pressure
uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_PRES, 1, true,
String(pres).c_str());
Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", MQTT_PUB_PRES,
packetIdPub3);
Serial.printf("Message: %.3f \n", pres);
```

Uploading the Code

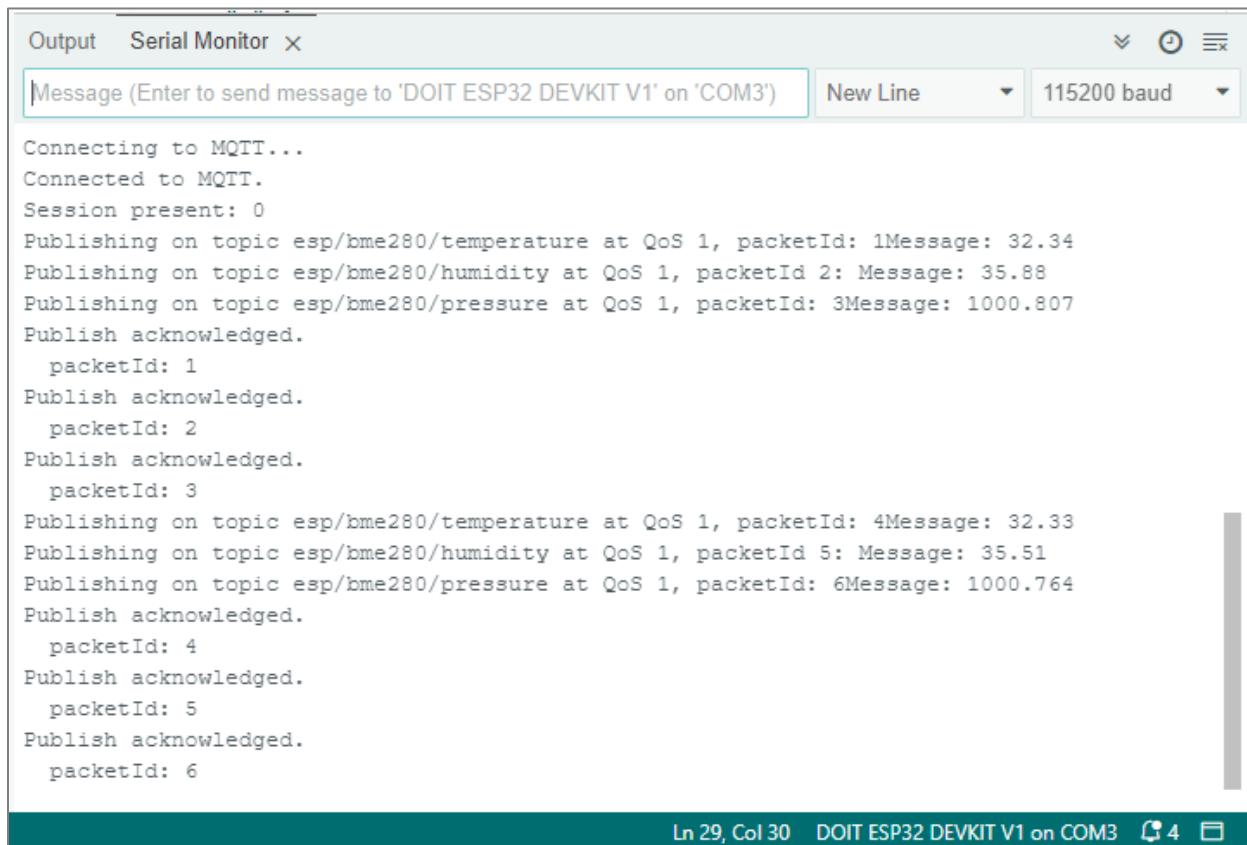
After inserting your network credentials and MQTT broker details, you can upload the code to your board. First, go to **Tools > Board** and select the board you're using. Then, select the COM port in **Tools > Port**. Finally, press the **Upload** button.



After successfully uploading the code to the board, open the Serial Monitor at a baud rate of 115200.



Press the board RST button to restart it and start running the code. The ESP should connect to Wi-Fi and to the MQTT broker and then, it should start publishing the messages periodically.



The screenshot shows the Arduino Serial Monitor window. The title bar says "Output" and "Serial Monitor". The main area displays the following text:

```
Message (Enter to send message to 'DOIT ESP32 DEVKIT V1' on 'COM3')
New Line 115200 baud

Connecting to MQTT...
Connected to MQTT.
Session present: 0
Publishing on topic esp/bme280/temperature at QoS 1, packetId: 1Message: 32.34
Publishing on topic esp/bme280/humidity at QoS 1, packetId 2: Message: 35.88
Publishing on topic esp/bme280/pressure at QoS 1, packetId: 3Message: 1000.807
Publish acknowledged.
  packetId: 1
Publish acknowledged.
  packetId: 2
Publish acknowledged.
  packetId: 3
Publishing on topic esp/bme280/temperature at QoS 1, packetId: 4Message: 32.33
Publishing on topic esp/bme280/humidity at QoS 1, packetId 5: Message: 35.51
Publishing on topic esp/bme280/pressure at QoS 1, packetId: 6Message: 1000.764
Publish acknowledged.
  packetId: 4
Publish acknowledged.
  packetId: 5
Publish acknowledged.
  packetId: 6
```

At the bottom, it says "Ln 29, Col 30 DOIT ESP32 DEVKIT V1 on COM3" and has icons for copy, cut, and paste.

The ESP32 or ESP8266 is now publishing on the `esp/bme280/temp`, `esp/bme280/hum`, and `esp/bme280/pres` topics.

Now, let's create a Node-RED flow that subscribes to those topics and displays the readings on the user interface.

Troubleshooting: if you get the message "Could not find a valid BME280 sensor, check wiring!" [check this troubleshooting guide](#).

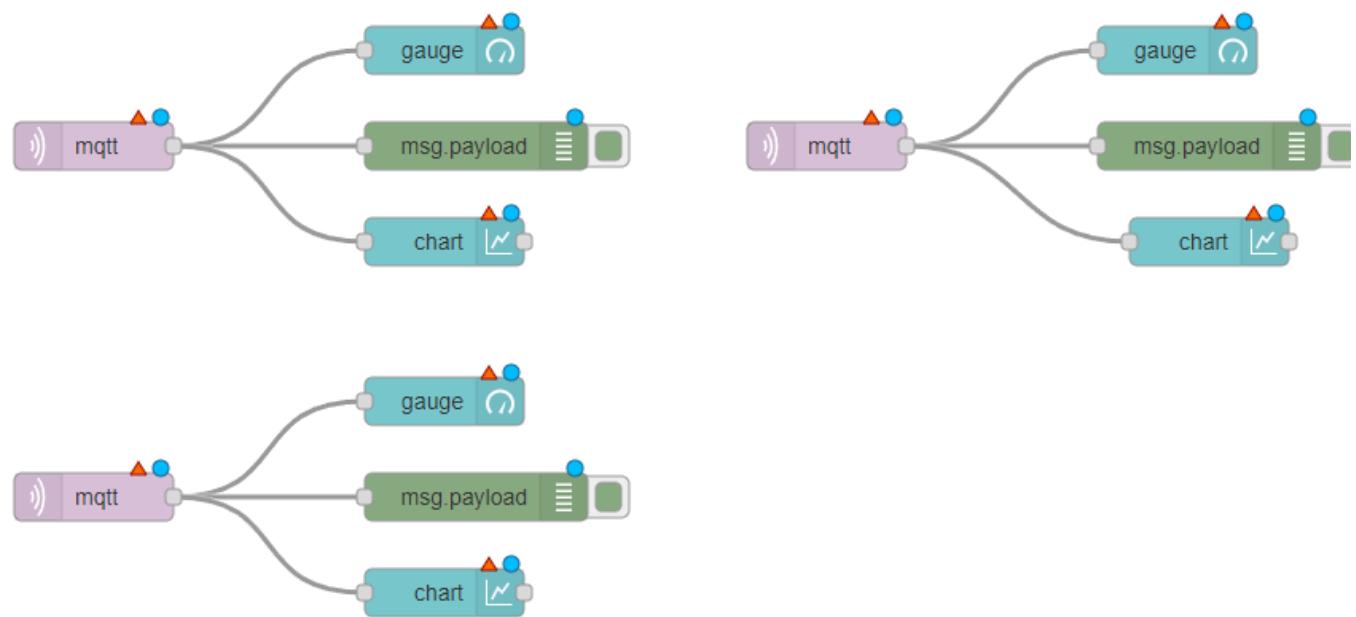
Creating the Node-RED Flow

Follow the next instructions to build the Node-RED flow. You can also download the flow on the following link. After importing the flow, don't forget to edit the MQTT broker node to insert your details (IP address and username, and password).

- [Download Node-RED flow](#)

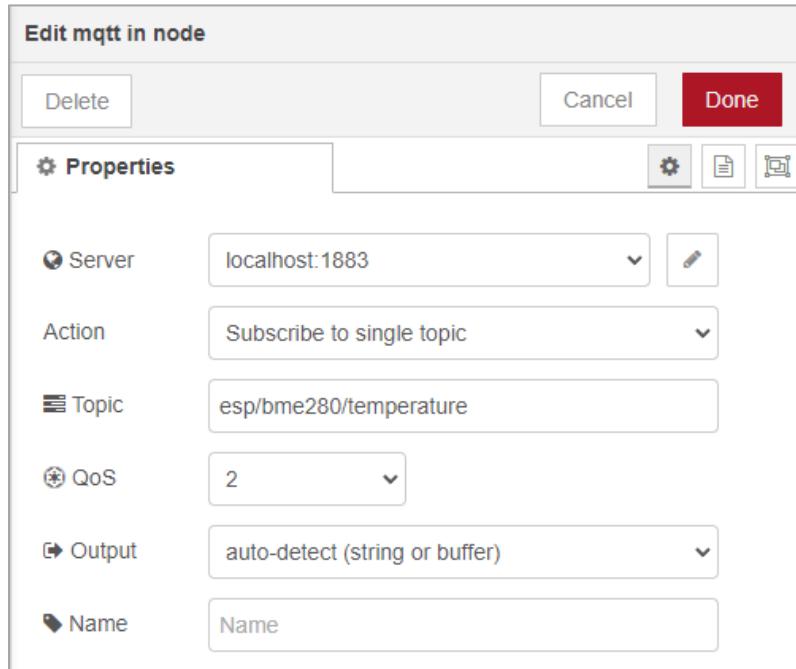
Drag the following nodes to the flow: **3x MQTT In** nodes, **3x chart** nodes, **3x gauge** nodes, and **3x debug** nodes.

Wire each MQTT node to one debug, one chart, and one gauge node.

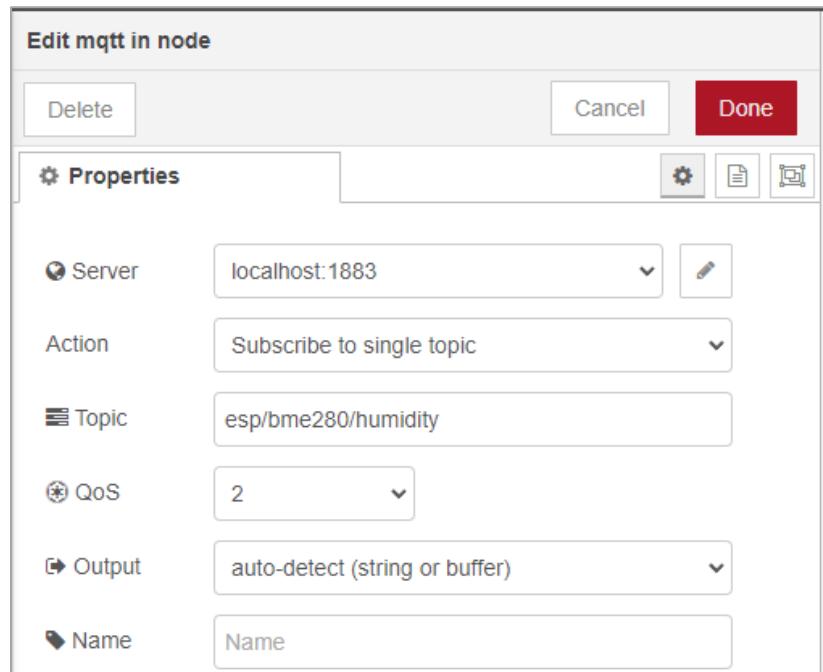


Now, you need to edit each MQTT node to subscribe to one of the topics that the ESP is publishing.

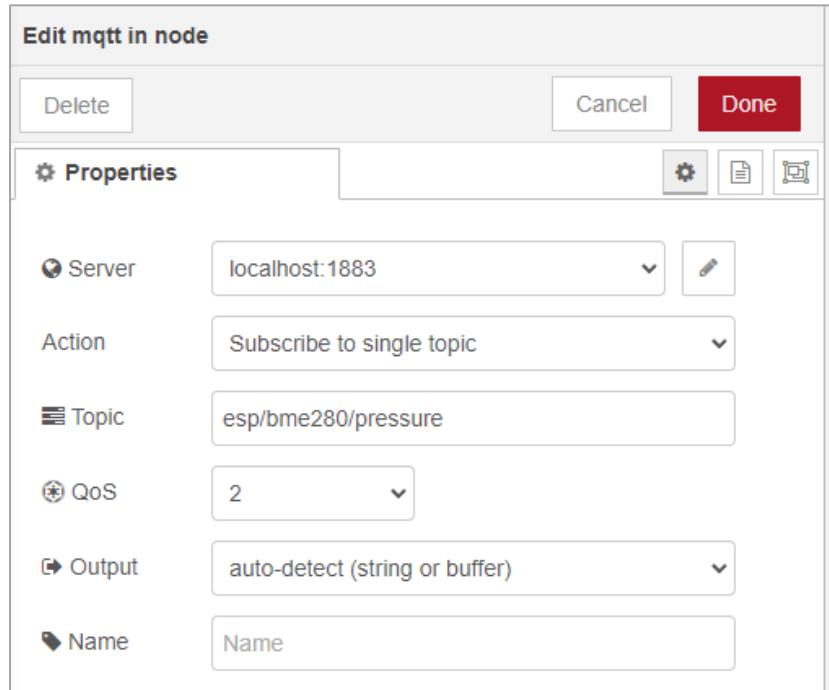
Here's the node that subscribes to the `esp/bme280/temperature` topic.



This is the node that subscribes to the `esp/bme280/humidity` topic.

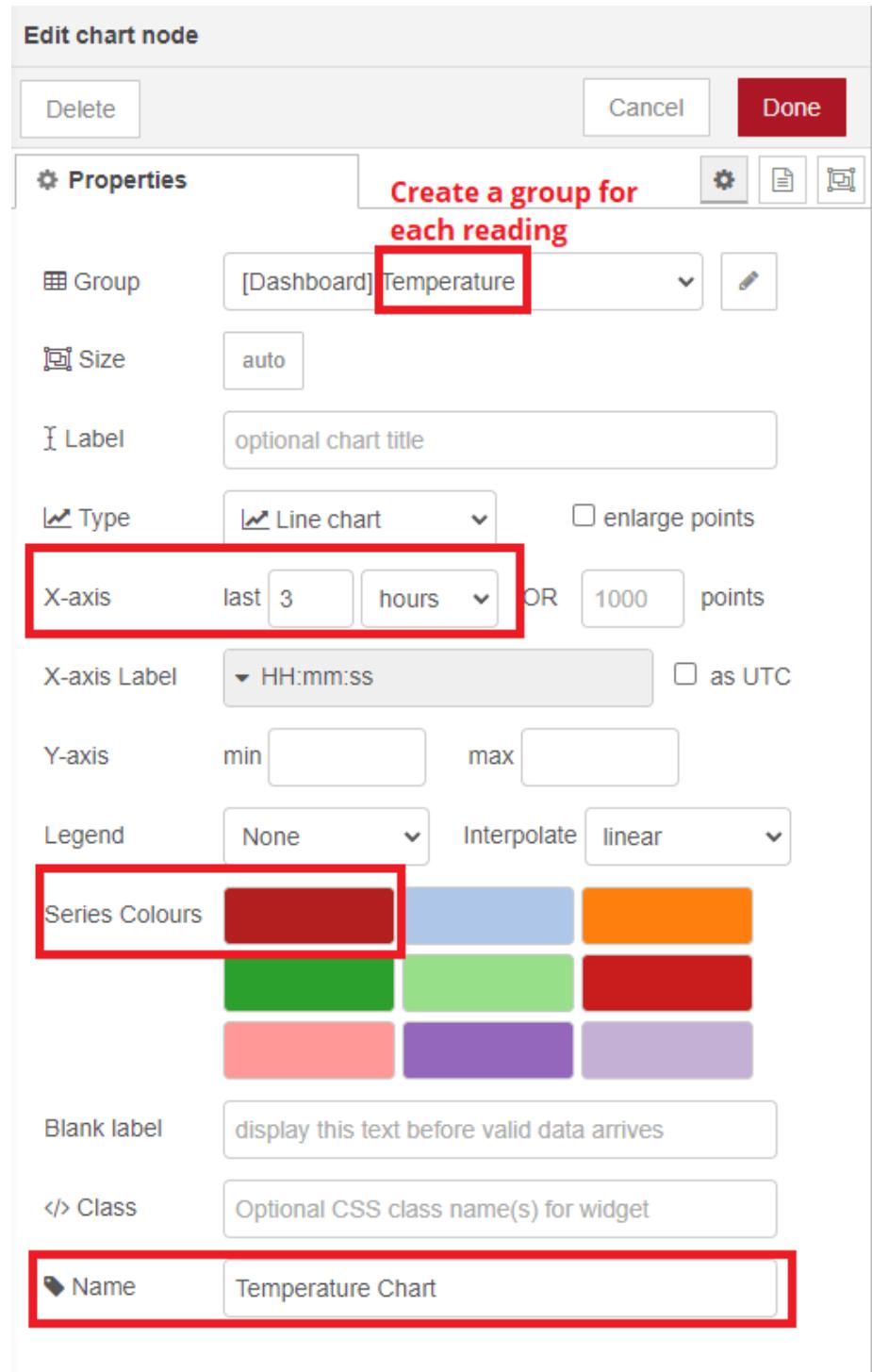


And finally, here's the node that subscribes to the `esp/bme280/pressure` topic.



Then, you can edit each chart and gauge node with the corresponding titles and different colors, for example. You also need to choose a place where the widgets will show up on the dashboard. We created a tab called `dashboard`, and three groups, one for the temperature widgets, other for the humidity widgets, and finally one for the pressure widgets. You can choose to display the widgets differently.

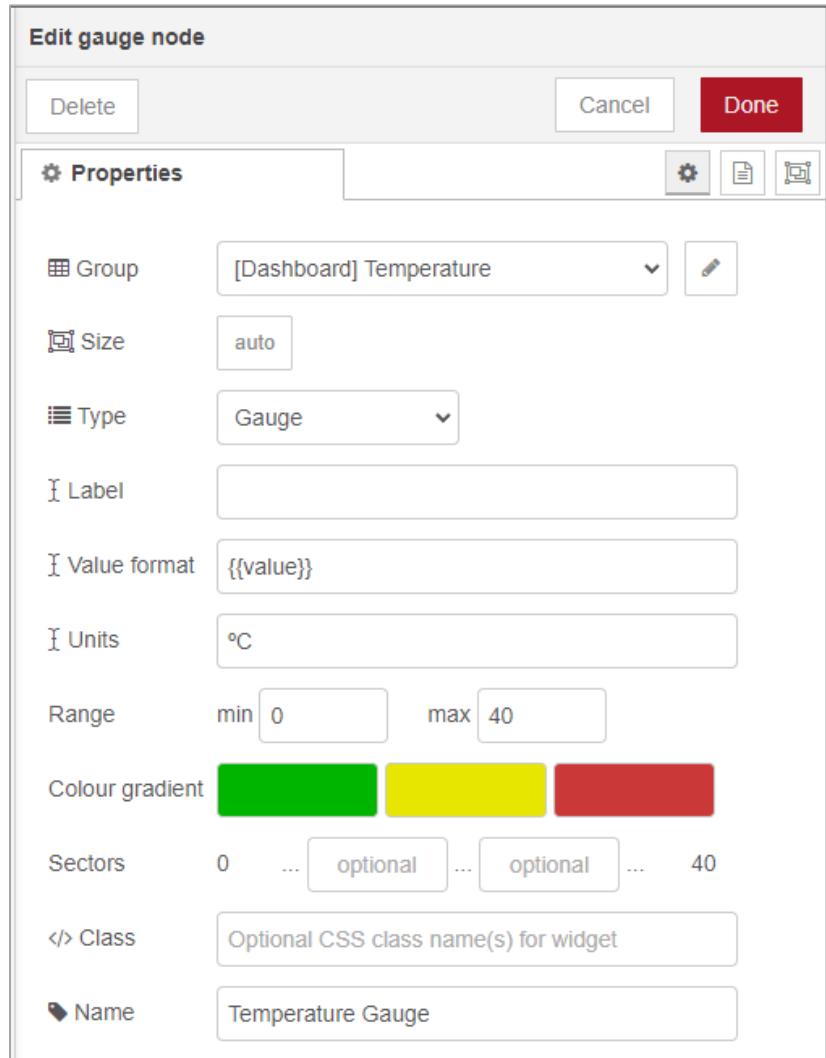
Here's the chart node for the temperature:



We set the **X-axis** to display the last three hours, but you can change that to any other value. We also change the **Series Colours** so that each reading has a different color. Finally, we labeled the node with the **Temperature Chart** name. You can set a

range for the chart, or if you leave it blank, the y-axis range will adjust to the values being displayed.

Now, you also need to edit the gauge node.

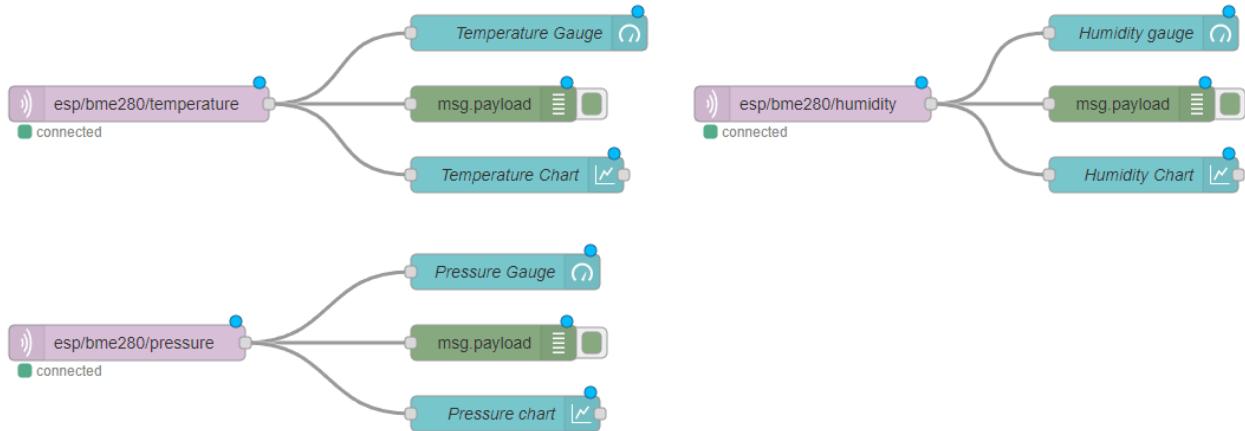


We added the temperature gauge to the `Temperature` group, added the units for the temperature, defined a minimum and maximum for the gauge range, and gave a name to the node.

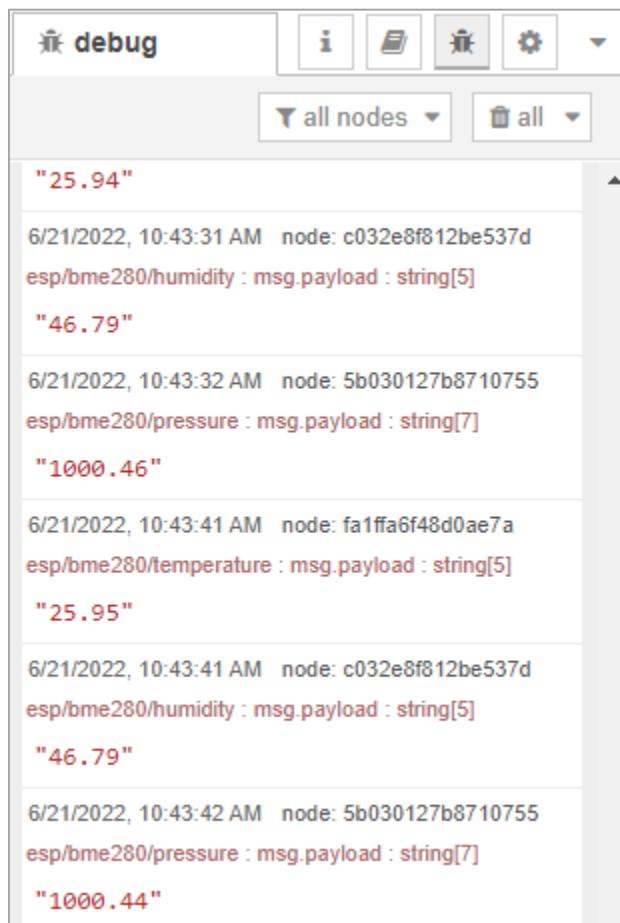
Edit the humidity and pressure nodes the same way. Choose different colors for the charts and gauges, and place them on the corresponding group. Deploy your application.



If everything went as expected, the flow will look as follows—with the **connected** message under the MQTT nodes.

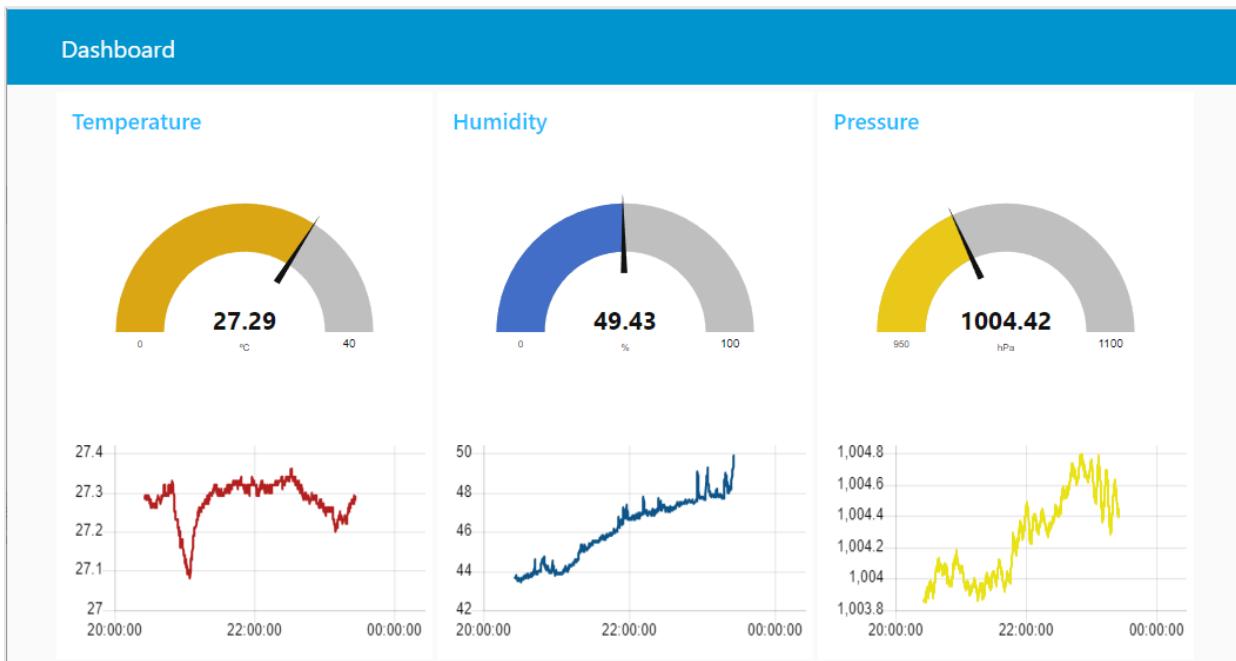


Open the debug window, you should start receiving the readings.



Testing the Flow

Go to the Node-RED UI to check out the user interface.



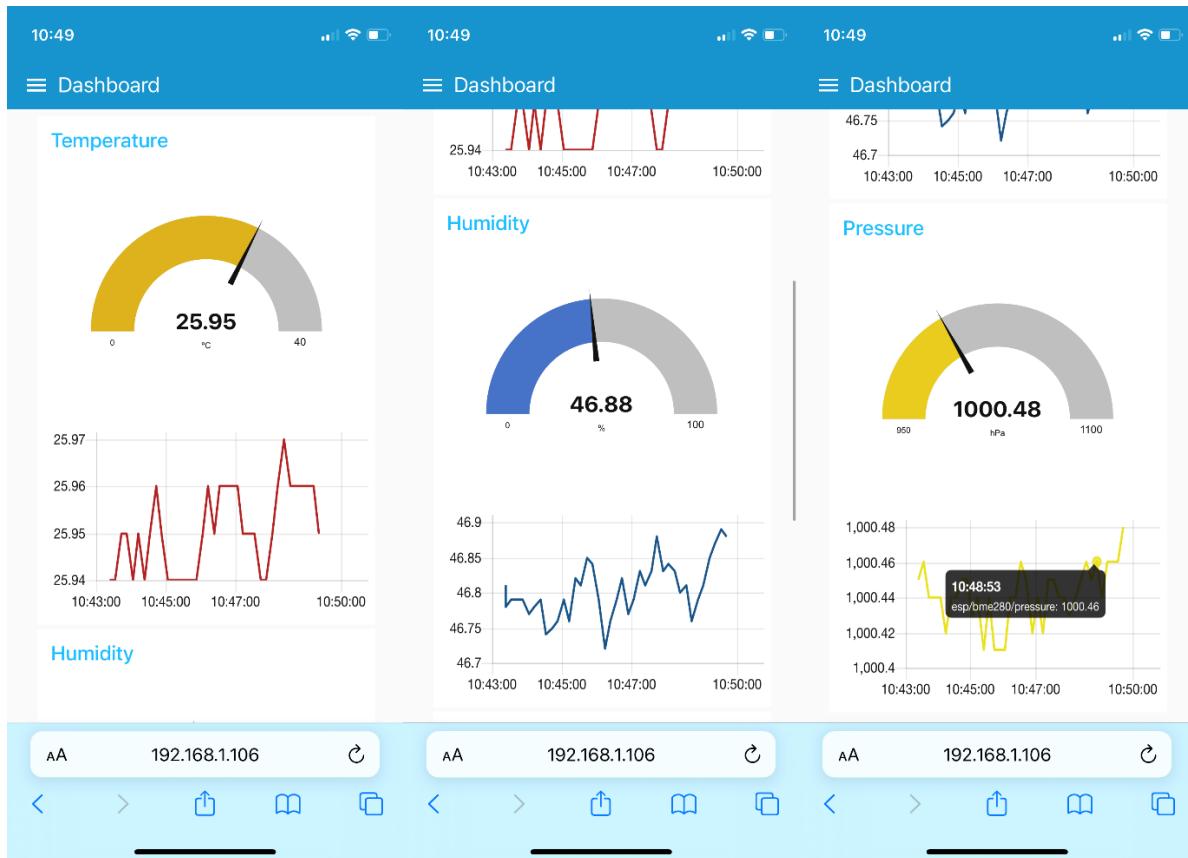
You should get something similar to the previous picture. You can also access Node-RED UI on your smartphone. The interface is mobile responsive—see pictures on the next page.

Wrapping Up

Congratulations! You just created an interface to display your sensor readings. As an example, we sent BME280 sensor readings. You can modify the project to send readings from any other sensor. We have several tutorials on our blog that cover how to publish sensor readings from several sensors:

- [ESP8266 with BME680](#)
- [ESP32 with BME680](#)
- [ESP8266 with DHT22/DHT11](#)
- [ESP32 with DHT22/DHT11](#)

- [ESP8266 DS18B20](#)
- [ESP32 DS18B20](#)



Every time you make any changes and deploy your flow, the sensor readings that were displayed on the charts are gone. Additionally, any readings that are older than 3 hours (or any other period you've defined on the chart settings) don't show up on the chart anymore.

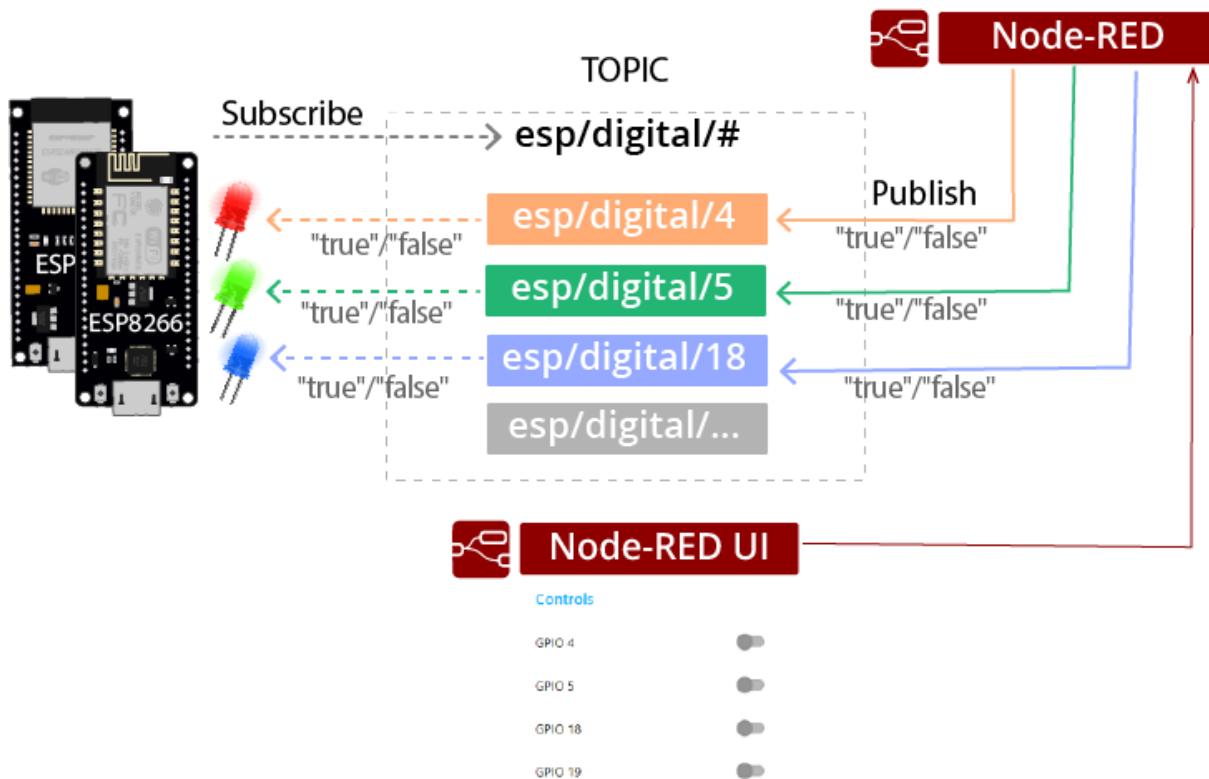
It would be useful to save the readings on a database and have a long record of the data history. That's what you're going to do in the next Module. You'll learn how to save your reading in InfluxDB, a real-time database.

5.4 - Subscribe to Multiple Topics and Control Multiple Outputs

In unit 5.2 you learned how to subscribe to a single topic and control the state of an ESP output depending on the message content. In that example, you specified in the ESP code which GPIO you wanted to control. In this unit, you'll learn how to subscribe to multiple topics in one go and control any output ON and OFF without having to previously declare it on the ESP32 or ESP8266 code.

Project Overview

Here's a summary of the project we'll build:



- We'll create a Node-RED flow with switches to control the ESP GPIOs;
- Node-RED publishes on the following topics `esp/digital/<gpio>`, in which `<gpio>` represents the GPIO number you want to control. For example, if you want to send a message to control GPIO 2, you would publish `true` or `false` on the following topic `esp/digital/2`.
- The ESP32/ESP8266 is subscribed to the multi-level topic `esp/digital/#`, so it receives all messages of any topic that begins with `esp/digital`. This means it receives all messages, regardless of the GPIO you want to control. Read the following section to learn more about MQTT topics.

More About MQTT Topics

Topics are represented with strings separated by a forward slash. Each forward slash indicates a topic level. Here's an example of how you would create a topic for a digital output (GPIO 2) for the ESP:

`esp/digital/2`

Note: topics are case-sensitive, which makes these two topics different:

`ESP/Digital/2` ≠ `esp/digital/2`

If you wanted to control GPIO 4 instead, you could use the following topic:

`esp/digital/4`

Multi-level Wildcard

MQTT Clients can subscribe to a topic, or can use wildcards to subscribe to multiple topics at once.

The multi-level wildcard is represented by the `#` symbol. The multi-level wildcard must be placed as the last character in the topic and preceded by a forward slash.

For example:

```
esp/digital/#
```

When a client subscribes to a topic with a multi-level wildcard, it receives all messages of a topic that begins with the pattern before the wildcard character. For example, if your ESP32/ESP8266 board is subscribed to the `esp/digital/#` topic, it will receive all messages on topics like: `esp/digital/2`, `esp/digital/4`, `esp/digital/board2/4`, for example.

So, now it's easy to understand that if the ESP subscribes to the `esp/digital/#` topic, and Node-RED publishes on the `esp/digital/<gpio>`, in which `<gpio>` represents the GPIO number you want to control, the ESP will be able to receive all messages regardless of the GPIO number. So, you don't need to know beforehand on the ESP which GPIO we want to control.

Note: if you subscribe to the wildcard topic `#`, you'll receive all messages sent to the MQTT broker. You should only do this for debugging purposes.

ESP32/ESP8266 Subscribe to Multi-level Topic – Code

The code is slightly different for each board. Upload the correct code for the board you're using.

- [Download Sketch folder ESP32](#)
- [Download Sketch folder ESP8266](#)

We added the lines of code to subscribe to the multi-level topic and control outputs to the previous code (publish sensor readings)—you can remove the sensor readings part if you only want to subscribe to topics and control outputs.

ESP32 – Code

Here's the code for the ESP32. You need to insert your network credentials (SSID and password), the broker IP address, and the broker username and password. The lines where you need to insert your details are highlighted in light green. The new sections of code are highlighted in a light orange color.

```
#include <Arduino.h>

#include <WiFi.h>
extern "C" {
    #include "freertos/FreeRTOS.h"
    #include "freertos/timers.h"
}
#include <AsyncMqttClient.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// BROKER
#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106)*/
#define MQTT_PORT 1883
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"

//MQTT Publish Topics
#define MQTT_PUB_TEMP "esp/bme280/temperature"
#define MQTT_PUB_HUM "esp/bme280/humidity"
#define MQTT_PUB_PRES "esp/bme280/pressure"

//MQTT Subscribe Topics
#define MQTT_SUB_DIGITAL "esp/digital/#"

// BME280 I2C
Adafruit_BME280 bme;

// Variables to hold sensor readings
float temp;
float hum;
float pres;

AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;

unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values
```

```

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

void WiFiEvent(WiFiEvent_t event) {
    Serial.printf("[WiFi-event] event: %d\n", event);
    switch(event) {
        case ARDUINO_EVENT_WIFI_STA_GOT_IP:
            Serial.println("WiFi connected");
            Serial.println("IP address: ");
            Serial.println(WiFi.localIP());
            connectToMqtt();
            break;
        case ARDUINO_EVENT_WIFI_STA_DISCONNECTED:
            Serial.println("WiFi lost connection");
            xTimerStop(mqttReconnectTimer, 0); // ensure we don't reconnect to MQTT
while reconnecting to Wi-Fi
            xTimerStart(wifiReconnectTimer, 0);
            break;
    }
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
    //Subscribe to topics
    // Subscribe to topic MQTT_SUB_DIGITAL when it connects to the broker
    uint16_t packetIdSub1 = mqttClient.subscribe(MQTT_SUB_DIGITAL, 2);
    Serial.print("Subscribing at QoS 2, packetId: ");
    Serial.println(packetIdSub1);

}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        xTimerStart(mqttReconnectTimer, 0);
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
}

```

```

    Serial.println(packetId);
}

void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties
properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message

    // Save the message in a variable
    String receivedMessage;
    for (int i = 0; i < len; i++) {
        Serial.println((char)payload[i]);
        receivedMessage += (char)payload[i];
    }
    // Save topic in a String variable
    String receivedTopic = String(topic);
    Serial.println("Received Topic: ");
    Serial.print(receivedTopic);

    // Check which GPIO we want to control
    int stringLen = receivedTopic.length();
    // Get the index of the last slash
    int lastSlash = receivedTopic.lastIndexOf("/");
    // Get the GPIO number (it's after the last slash "/")
    // esp/digital/GPIO
    String gpio = receivedTopic.substring(lastSlash+1, stringLen);
    Serial.print("DIGITAL GPIO: ");
    Serial.println(gpio);
    Serial.println("STATE: ");
    Serial.println(receivedMessage);

    // Check if it is DIGITAL
    if (receivedTopic.indexOf("digital") > 0) {
        //Set the specified GPIO as output
        pinMode(gpio.toInt(), OUTPUT);
        //Control the GPIO
        if (receivedMessage == "true"){
            digitalWrite(gpio.toInt(), HIGH);
        }
        else{
            digitalWrite(gpio.toInt(), LOW);
        }
    }
    Serial.println("Publish received.");
    Serial.print(" topic: ");
    Serial.println(topic);
    Serial.print(" qos: ");
    Serial.println(properties.qos);
    Serial.print(" dup: ");
    Serial.println(properties.dup);
    Serial.print(" retain: ");
    Serial.println(properties.retain);
    Serial.print(" len: ");
    Serial.println(len);
    Serial.print(" index: ");
    Serial.println(index);
    Serial.print(" total: ");
    Serial.println(total);
}

```

```

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();

    // Initialize BME280 sensor
    if (!bme.begin(0x76)) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
        while (1);
    }

    mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
    wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));

    WiFi.onEvent(WiFiEvent);

    mqttClient.onConnect(onMqttConnect);
    mqttClient.onDisconnect(onMqttDisconnect);
    mqttClient.onSubscribe(onMqttSubscribe);
    mqttClient.onUnsubscribe(onMqttUnsubscribe);
    mqttClient.onMessage(onMqttMessage);
    mqttClient.onPublish(onMqttPublish);
    mqttClient.setServer(MQTT_HOST, MQTT_PORT);
    mqttClient.setCredentials(BROKER_USER, BROKER_PASS);

    connectToWifi();
}

void loop() {

    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 10 seconds)
    // it publishes a new MQTT message
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;

        // New BME280 sensor readings
        temp = bme.readTemperature();
        //temp = 1.8*bme.readTemperature() + 32;
        hum = bme.readHumidity();
        pres = bme.readPressure()/100.0F;

        // Publish an MQTT message on topic esp32/BME2800/temperature
        uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
        Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_TEMP, packetIdPub1);
        Serial.printf("Message: %.2f \n", temp);
}

```

```

    // Publish an MQTT message on topic esp32/BME2800/humidity
    uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(hum).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", MQTT_PUB_HUM, packetIdPub2);
    Serial.printf("Message: %.2f \n", hum);

    // Publish an MQTT message on topic esp32/BME2800/pressure
    uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_PRES, 1, true,
String(pres).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", MQTT_PUB_PRES, packetIdPub3);
    Serial.printf("Message: %.3f \n", pres);
}
}

```

ESP8266 – Code

Here's the code for the ESP8266. You need to insert your network credentials (SSID and password), the broker IP address, and the broker username and password. The lines where you need to insert your details are highlighted in light green. The new lines of code are highlighted in light orange.

```

#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// BROKER
#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106)*/
#define MQTT_PORT 1883
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"

//MQTT Topics
#define MQTT_PUB_TEMP "esp/bme280/temperature"
#define MQTT_PUB_HUM "esp/bme280/humidity"
#define MQTT_PUB_PRES "esp/bme280/pressure"

//MQTT Subscribe Topics
#define MQTT_SUB_DIGITAL "esp/digital/#"

// BME280 I2C

```

```

Adafruit_BME280 bme;

// Variables to hold sensor readings
float temp;
float hum;
float pres;

AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;

WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
Ticker wifiReconnectTimer;

unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

void onWifiConnect(const WiFiEventStationModeGotIP& event) {
    Serial.println("Connected to Wi-Fi.");
    connectToMqtt();
}

void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
    Serial.println("Disconnected from Wi-Fi.");
    mqttReconnectTimer.detach(); // ensure we don't reconnect to MQTT while
reconnecting to Wi-Fi
    wifiReconnectTimer.once(2, connectToWifi);
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
    //Subscribe to topics
    // Subscribe to topic MQTT_SUB_DIGITAL when it connects to the broker
    uint16_t packetIdSub1 = mqttClient.subscribe(MQTT_SUB_DIGITAL, 2);
    Serial.print("Subscribing at QoS 2, packetId: ");
    Serial.println(packetIdSub1);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        mqttReconnectTimer.once(2, connectToMqtt());
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
}

```

```

    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message

    // Save the message in a variable
    String receivedMessage;
    for (int i = 0; i < len; i++) {
        Serial.println((char)payload[i]);
        receivedMessage += (char)payload[i];
    }
    // Save topic in a String variable
    String receivedTopic = String(topic);
    Serial.println("Received Topic: ");
    Serial.print(receivedTopic);

    // Check which GPIO we want to control
    int stringLen = receivedTopic.length();
    // Get the index of the last slash
    int lastSlash = receivedTopic.lastIndexOf("/");
    // Get the GPIO number (it's after the last slash "/")
    // esp/digital/GPIO
    String gpio = receivedTopic.substring(lastSlash+1, stringLen);
    Serial.print("DIGITAL GPIO: ");
    Serial.println(gpio);
    Serial.println("STATE: ");
    Serial.println(receivedMessage);

    // Check if it is DIGITAL
    if (receivedTopic.indexOf("digital") > 0) {
        //Set the specified GPIO as output
        pinMode(gpio.toInt(), OUTPUT);
        //Control the GPIO
        if (receivedMessage == "true") {
            digitalWrite(gpio.toInt(), HIGH);
        }
        else{
            digitalWrite(gpio.toInt(), LOW);
        }
    }
    Serial.println("Publish received.");
    Serial.print(" topic: ");
    Serial.println(topic);
    Serial.print(" qos: ");
    Serial.println(properties.qos);
    Serial.print(" dup: ");
    Serial.println(properties.dup);
}

```

```

    Serial.print("  retain: ");
    Serial.println(properties.retain);
    Serial.print("  len: ");
    Serial.println(len);
    Serial.print("  index: ");
    Serial.println(index);
    Serial.print("  total: ");
    Serial.println(total);
}

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print("  packetId: ");
    Serial.println(packetId);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();

    // Initialize BME280 sensor
    if (!bme.begin(0x76)) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
        while (1);
    }

    wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
    wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);

    mqttClient.onConnect(onMqttConnect);
    mqttClient.onDisconnect(onMqttDisconnect);
    mqttClient.onSubscribe(onMqttSubscribe);
    mqttClient.onUnsubscribe(onMqttUnsubscribe);
    mqttClient.onMessage(onMqttMessage);
    mqttClient.onPublish(onMqttPublish);
    mqttClient.setServer(MQTT_HOST, MQTT_PORT);
    mqttClient.setCredentials(BROKER_USER, BROKER_PASS);

    connectToWifi();
}

void loop() {

    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 10 seconds)
    // it publishes a new MQTT message
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;

        // New BME280 sensor readings
        temp = bme.readTemperature();
        //temp = 1.8*bme.readTemperature() + 32;
        hum = bme.readHumidity();
        pres = bme.readPressure()/100.0F;

        // Publish an MQTT message on topic esp32/BME2800/temperature
    }
}

```

```

    uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_TEMP, packetIdPub1);
    Serial.printf("Message: %.2f \n", temp);

    // Publish an MQTT message on topic esp32/BME2800/humidity
    uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(hum).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ", 
MQTT_PUB_HUM, packetIdPub2);
    Serial.printf("Message: %.2f \n", hum);
    // Publish an MQTT message on topic esp32/BME2800/pressure
    uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_PRES, 1, true,
String(pres).c_str());
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_PRES, packetIdPub3);
    Serial.printf("Message: %.3f \n", pres);
}
}

```

How the Code Works

The code is similar to the ones we've seen in previous units. So, we'll just take a look at the relevant parts for this example (highlighted in light orange color).

Define the Subscribe Topic

Define the MQTT topic you want your board to subscribe to. We'll subscribe to multiple topics at once by using the multi-level wildcard (#).

```
//MQTT Subscribe Topics
#define MQTT_SUB_DIGITAL "esp/digital/#"
```

Subscribe to Topics

Inside the `onMqttConnect()` function, subscribe to the topic.

```
void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
    //Subscribe to topics
    // Subscribe to topic MQTT_SUB_DIGITAL when it connects to the broker
    uint16_t packetIdSub1 = mqttClient.subscribe(MQTT_SUB_DIGITAL, 2);
    Serial.print("Subscribing at QoS 2, packetId: ");
    Serial.println(packetIdSub1);
}
```

Receive Messages and Control Outputs

Whenever a message is received on a subscribed topic, the `onMqttMessage()` runs.

```
void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message
```

So, we must add what we want the ESP to do when it receives a message. We need to get the GPIO we want to control (it's the last topic level), and the state (it's the payload message).

First, we save the message on the `receivedMessage` variable.

```
// Save the message in a variable
String receivedMessage;
for (int i = 0; i < len; i++) {
    Serial.println((char)payload[i]);
    receivedMessage += (char)payload[i];
}
```

Then, we convert the `topic` to a String variable (`receivedTopic`) so it's easier to manipulate later.

```
// Save topic in a String variable
String receivedTopic = String(topic);
Serial.print("Received Topic: ");
Serial.println(receivedTopic);
```

The GPIO we want to control comes in the topic, and it is the last level of the topic. For example: `esp/digital/12`. We can split the string in the last slash, and we'll have the GPIO number.

First, we get the length of the topic.

```
int stringLen = receivedTopic.length();
```

Then, we can use the `lastIndexOf()` to discover the position of the last slash. The `lastIndexOf()` method locates a string within another String. It starts searching from the end of the String.

```
// Get the index of the last slash
int lastSlash = receivedTopic.lastIndexOf("/");
```

Once we know the position of the last slash, we can split the string using the `substring()` method. The `substring()` method gets a substring of a string starting and ending at determined positions. Learn more about the [substring\(\) method](#).

```
// Get the GPIO number (it's after the last slash "/")
// esp/digital/GPIO
String gpio = receivedTopic.substring(lastSlash+1, stringLen);
```

Then, we control the LED on or off accordingly to the message content using the `digitalWrite()` function:

```
if (receivedTopic.indexOf("digital") > 0) {
    //Set the specified GPIO as output
    pinMode(gpio.toInt(), OUTPUT);
    //Control the GPIO
    if (receivedMessage == "true"){
        digitalWrite(gpio.toInt(), HIGH);
    }
    else{
        digitalWrite(gpio.toInt(), LOW);
    }
}
```

As you can see, we don't need to hardcode any GPIO numbers, because we declare them and control them accordingly to the topics received.

The rest of the code is the same as the previous unit.

Creating the Node-RED Flow

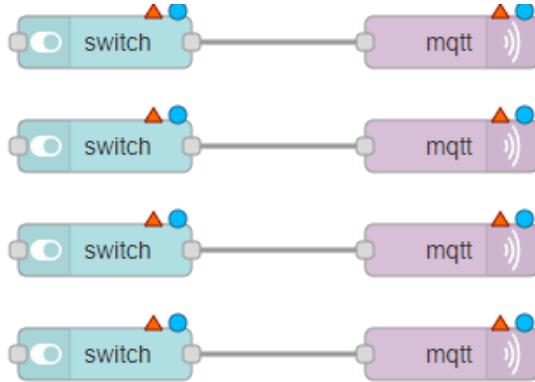
Now, let's create the flow with switches to control the ESP32 or ESP8266 GPIOs. You can add this flow to your previous flow (sensor readings) or create a new flow only with the switches.

You can also download the flow at the following link. After importing the flow, don't forget to edit the MQTT broker node to insert your details (IP address, username, and password).

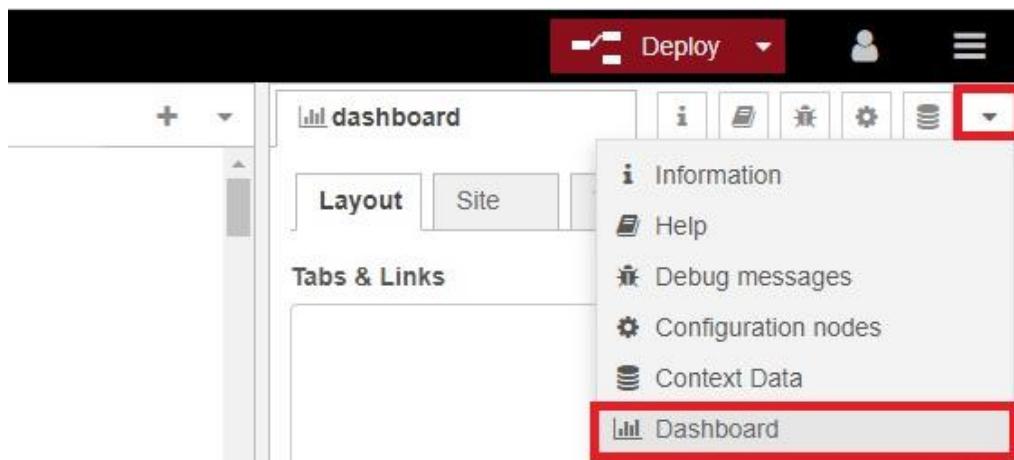
- [Download Node-RED flow](#)

As an example, we'll create four switches to control four GPIOs of the ESP32. You can control more or less GPIOs. The flow will work for any GPIOs that can work properly as outputs. The code that is running on the ESP will be able to interpret the topic and control any GPIO we want.

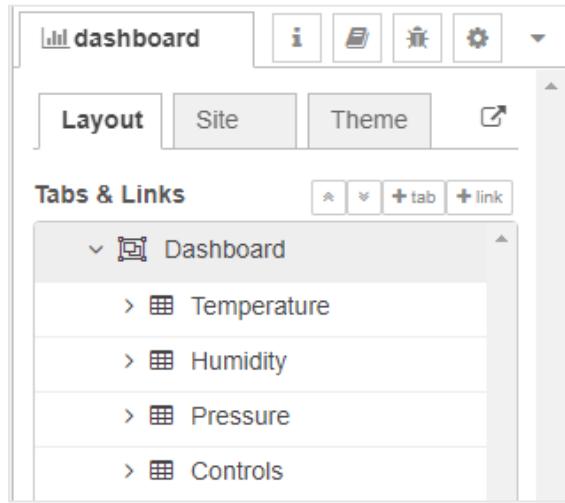
Drag four **switches** and four **MQTT out** nodes to the flow. Wire each switch to an MQTT node.



Now, you need to edit each switch with a different label, so that you know which GPIO you're controlling. You also need to choose a place for your dashboard nodes. You can add them to an existing group, or create a new group just for the switches.



We created a new group on our **Dashboard** tab called **Controls** where we'll be placing the switches.



Double-click on a switch node to edit its properties. The following switch node will control GPIO 4. You can control any other GPIOs.

Edit switch node

Delete Cancel Done

Properties

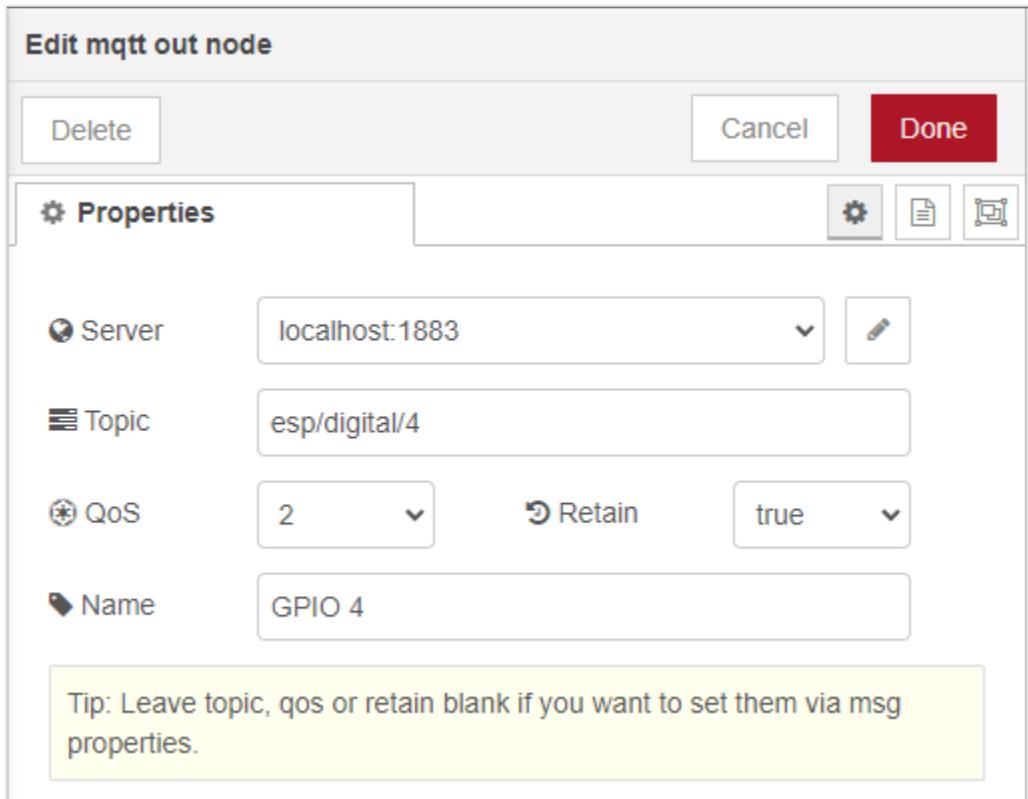
| | |
|---------|----------------------|
| Group | [Dashboard] Controls |
| Size | auto |
| Label | GPIO 4 |
| Tooltip | Controls GPIO 4 |
| Icon | Default |

→ Pass through `msg` if payload matches valid state:

✉ When clicked, send:

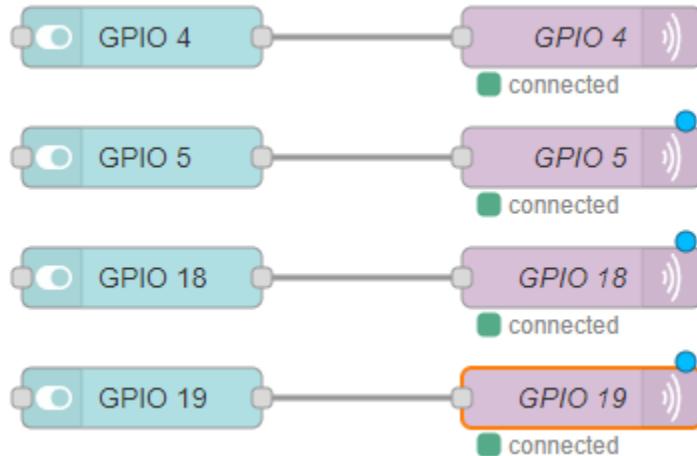
| | |
|-------------|---------------------------------------|
| On Payload | true |
| Off Payload | false |
| Topic | msg.topic |
| </> Class | Optional CSS class name(s) for widget |
| Name | |

Then, edit the corresponding MQTT node to publish on the topic `esp/digital/4`. Set retain to `true` and QoS to 2.



Proceed the same way for the other nodes, but using different GPIO numbers. We've chosen GPIOs 5, 18, and 19 for the ESP32. For the ESP8266, the best output pins are GPIOs 12, 13, 14, and 15.

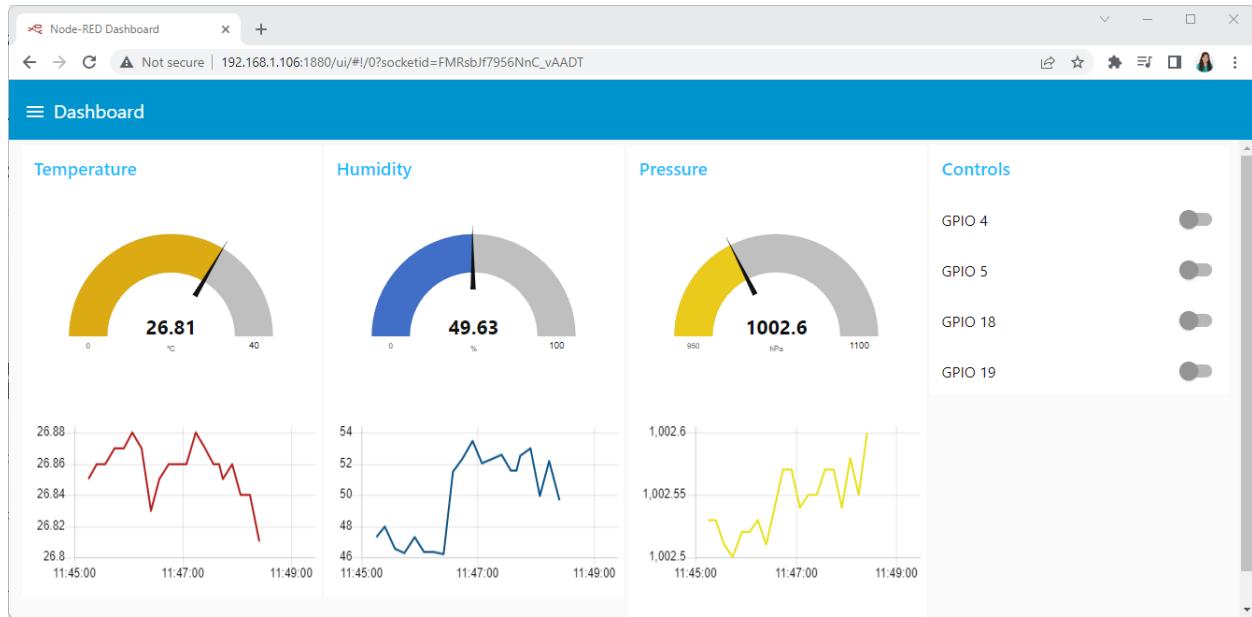
Here's what the flow looks like:



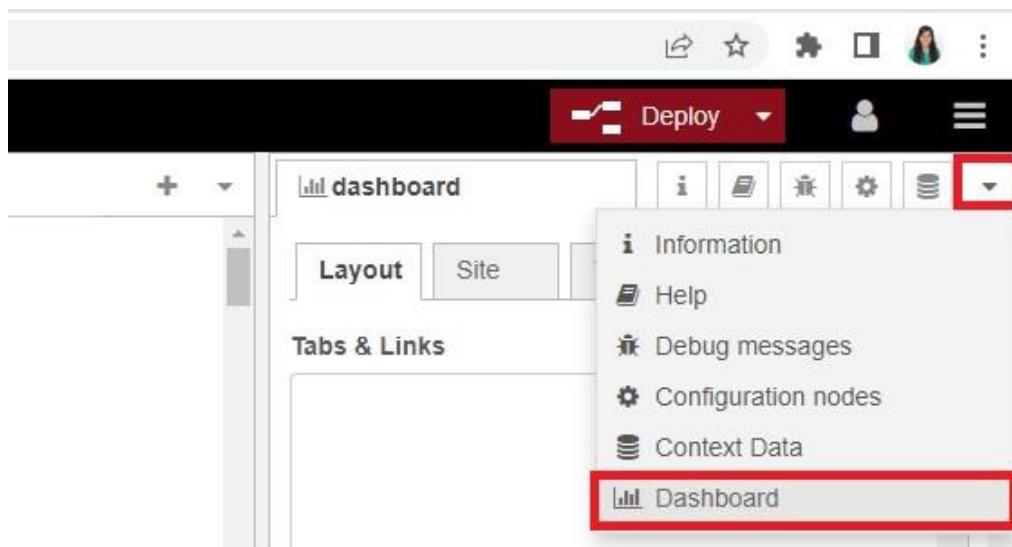
Deploy your application.



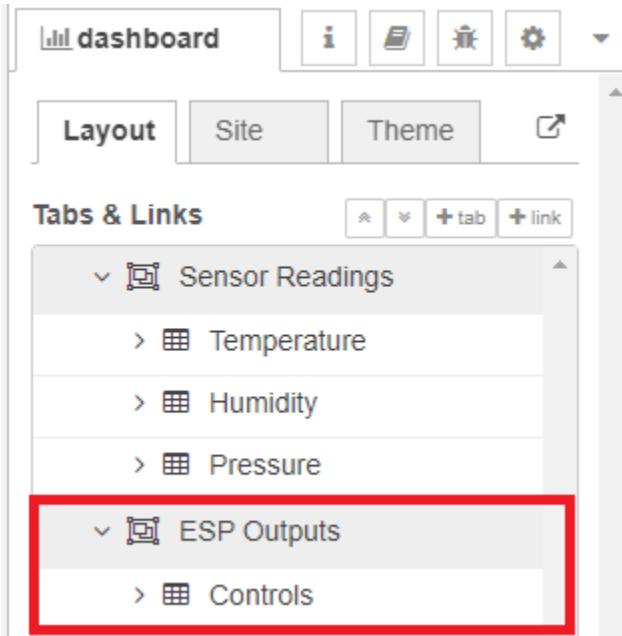
If you added your controls to the same tab of the sensor readings, your dashboard will look like this.



You may want to create different tabs: one for the sensor readings, and another for the controls. In that case, you need to go to your dashboard

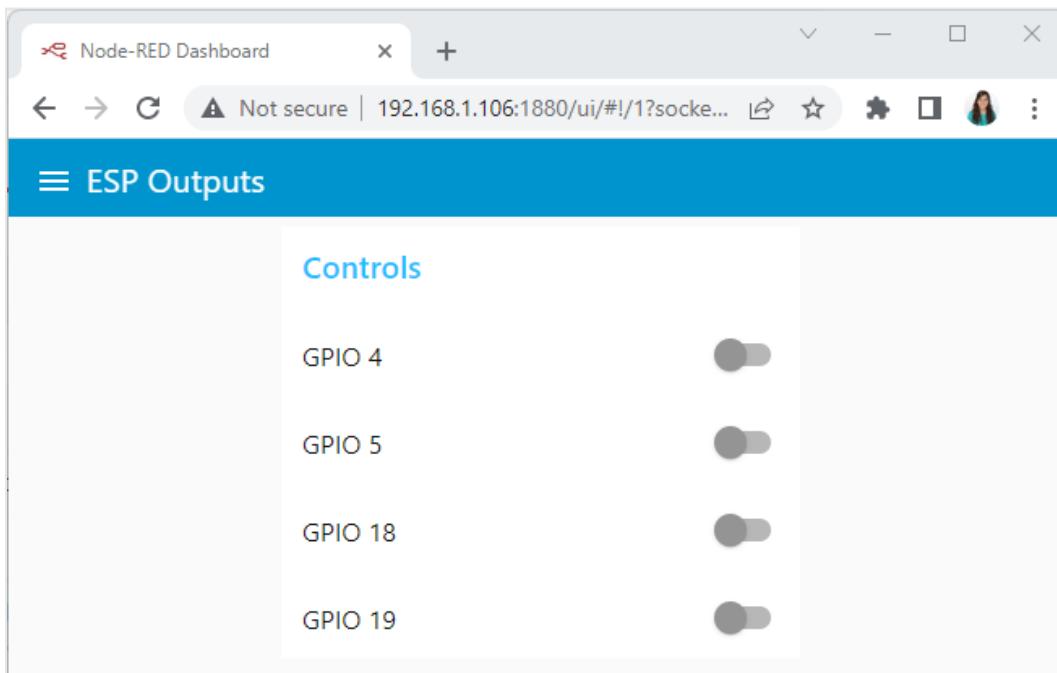


Create a new tab and group. For example:



Then, you need to edit the **switch** nodes so that they are placed on that tab and group.

This is what the dashboard looks like if you place the controls on a different tab.



Choose the arrangement that you like most.

Wiring the Circuit

Now, to see the project in action, wire 4 LEDs to the chosen GPIOs. You can also add any other components that work with ON and OFF commands.

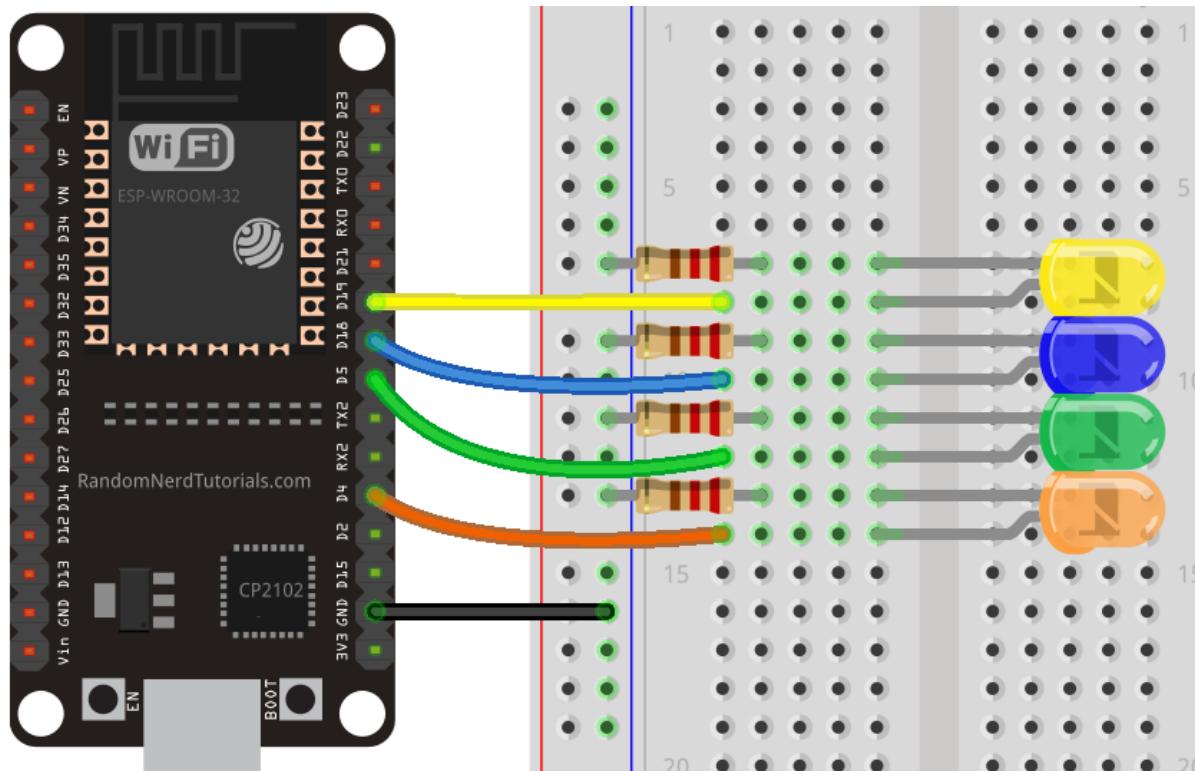
Parts Required

- [4x LEDs](#)
- [4x 220 Ω resistor](#)
- [Breadboard](#)
- [Jumper wires](#)

For example, for the ESP32, we've chosen GPIOs 4, 5, 18, and 19. And for the ESP8266, we've chosen GPIOs 12, 13, 14, and 15.

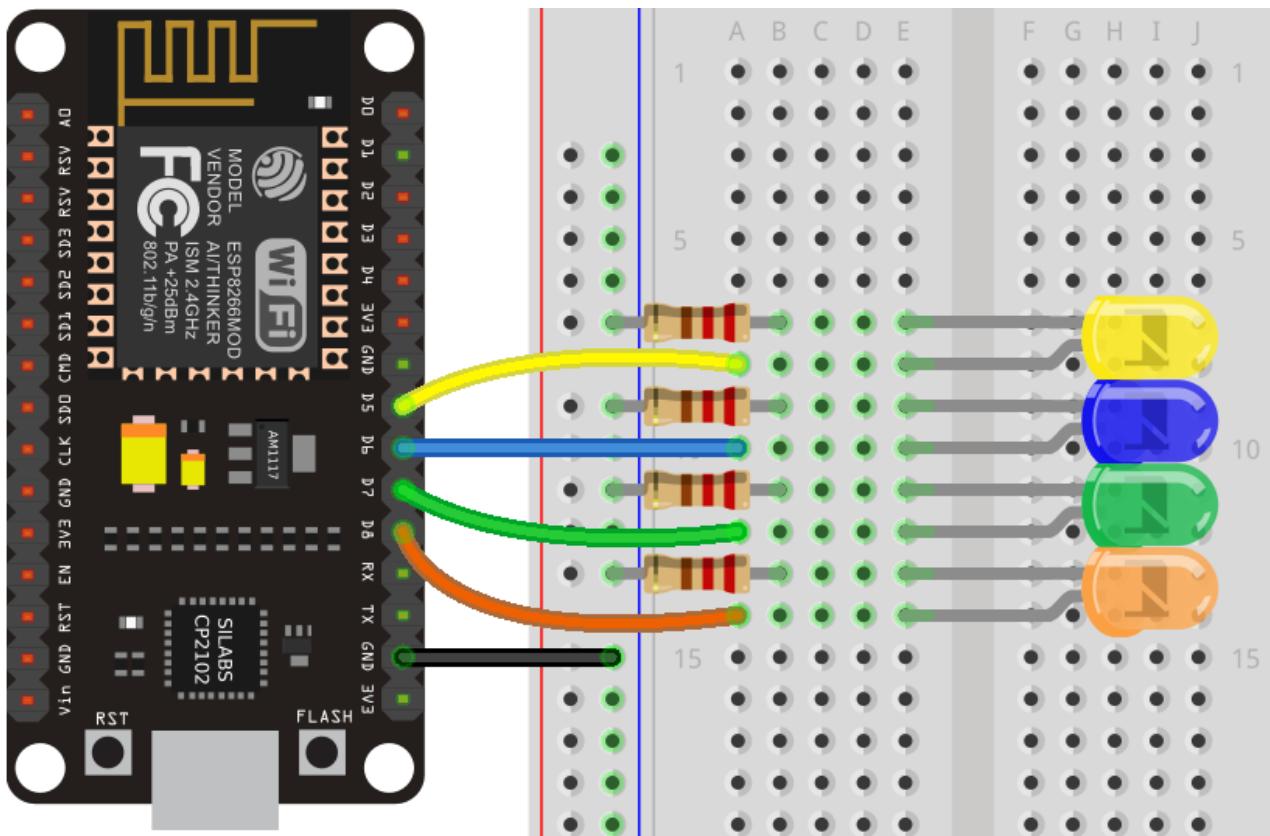
ESP32

Wire four LEDs to the ESP32.



ESP8266

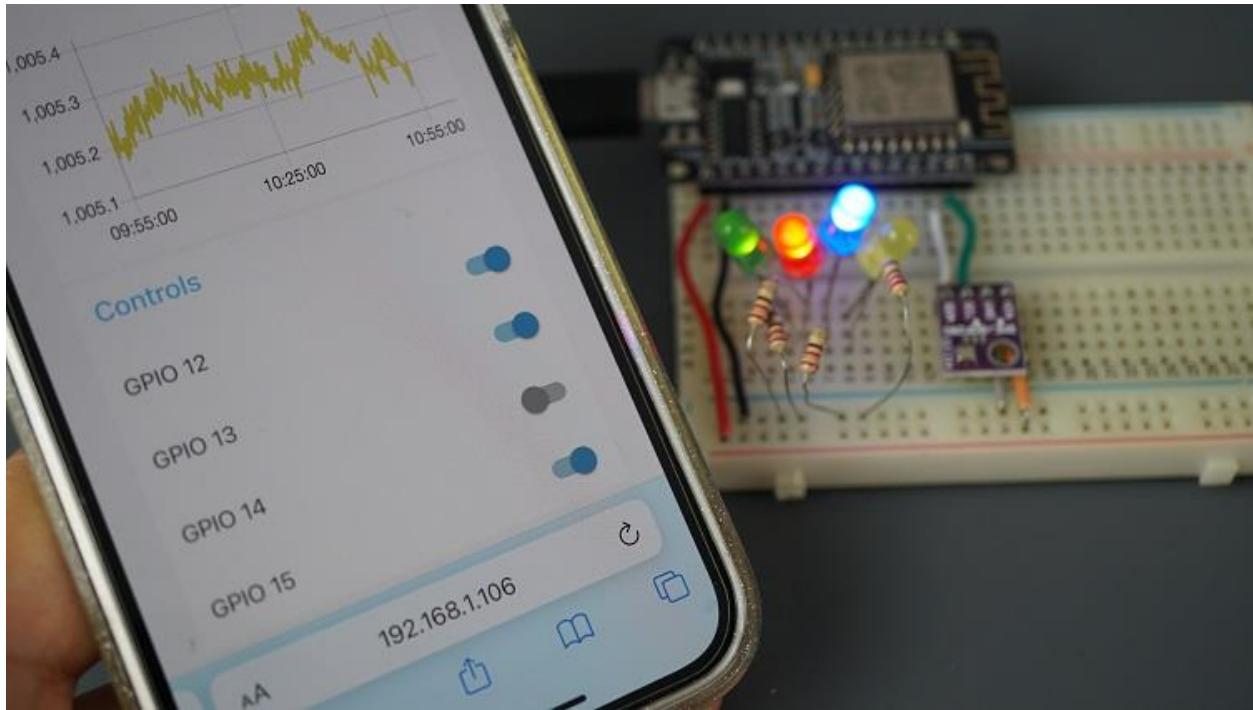
Wire four LEDs to the ESP8266.



Testing the Project

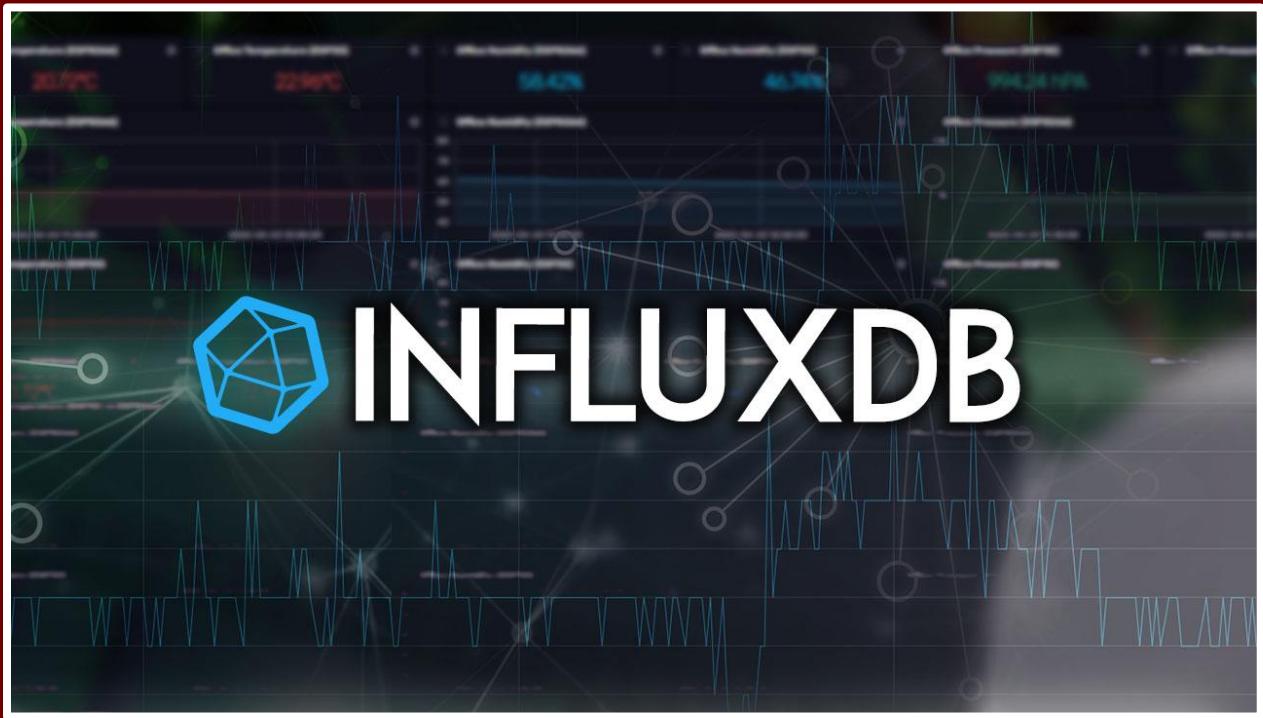
After wiring the circuit, you can test the project. Click on the dashboard switches and see the LEDs changing their state accordingly.

Don't forget that you can also access the dashboard from your smartphone.



MODULE 6

InfluxDB Time-Series Database



InfluxDB is a time-series database. Each record on the database is associated with a timestamp, which makes it ideal for data logging, IoT, and home automation projects. InfluxDB also provides dashboard tools to visualize data in different formats like charts, gauges, histograms, etc. You can easily connect Node-RED to InfluxDB and save your readings on the database and use their dashboard to analyze the data in many different formats.

6.1 - Getting Started with InfluxDB

InfluxDB is an open-source high-performance time-series database (TSDB) that can store large amounts of data per second. Each data point you submit to the database is associated with a particular timestamp. So, it is ideal for IoT data logging projects like storing data from your weather station sensors.



You can run [InfluxDB in InfluxDB Cloud](#), or locally on your laptop or [Raspberry Pi](#).

InfluxDB cloud is a great alternative because it hosts your data that can be accessed from anywhere. However, the free cloud plan, only lets you save data until the last 30 days. Depending on your project application, that might be a good alternative. Or if you're willing to pay for a premium plan, InfluxDB cloud might be the best solution.

Throughout the eBook, we'll be using InfluxDB installed on the Raspberry Pi. However, if you prefer to use the cloud-based InfluxDB instead, it works similarly and you can use it throughout this course as an alternative to the local server.

InfluxDB Key Terms

Before getting started, there are some important terms you need to understand. We'll just take a look at the most relevant terms, you can read the complete [glossary](#). Don't worry if some of the terms are confusing. You'll better understand those terms when you start writing to the database.

In InfluxDB, a **bucket** is named location where the time series data is stored. All buckets have a **retention period**—it defines the duration of time that a bucket retains data. Points with timestamps older than the retention period are dropped.

Data in InfluxDB is stored in **tables** within rows and columns. A set of data in a row is known as **point** (similar to a row in a SQL database table). Each point has a **measurement**, a **tag** set, a field **key**, a field **value**, and a **timestamp**;

Columns store tag sets (indexed) and field sets. The only required column is **time**, which stores timestamps and is included in all InfluxDB tables.

For example, take a look at the following example of data stored on InfluxDB.

bucket: my_bucket

| _time | _measurement | location | scientist | _field | _value |
|----------------------|--------------|----------|-----------|--------|--------|
| 2019-08-18T00:00:00Z | census | klamath | anderson | bees | 23 |
| 2019-08-18T00:00:00Z | census | portland | mullen | ants | 30 |
| 2019-08-18T00:06:00Z | census | klamath | anderson | bees | 28 |
| 2019-08-18T00:06:00Z | census | portland | mullen | ants | 32 |

Image source: <https://docs.influxdata.com/influxdb/v2.2/reference/key-concepts/data-elements/>

- **tag**: the key-value pair in InfluxDB's data structure that records metadata. Tags are an optional part of InfluxDB's data structure but they are useful for storing commonly-queried metadata; tags are indexed so queries on tags are performant.

- **tag key:** tag keys are strings and store metadata. Tag keys are indexed so queries on tag keys are processed quickly. Tag keys in the previous table are `location` and `scientist`.
- **tag value:** tag values are strings and they store metadata. Tag values are indexed so queries on tag values are processed quickly. Tag values include the name of the scientists (`anderson` and `mullen`) and the location name (`klamath` and `portland`).
- **field:** the key-value pair in InfluxDB's data structure that records metadata and the actual data value. Fields are required in InfluxDB's data structure and they are not indexed – queries on field values scan all points that match the specified time range and, as a result, are not performant relative to tags.
- **field key:** the key of the key-value pair. Field keys are strings and they store metadata—`bees` and `ants` are keys.
- **field value:** the value of a key-value pair. Field values are the actual data; they can be strings, floats, integers, or booleans. A field value is always associated with a timestamp. Field values are not indexed – queries on field values scan all points that match the specified time range and, as a result, are not performant. Field values are `23`, `30`, `28`, and `32`.
- **measurement:** the part of InfluxDB's structure that describes the data stored in the associated fields. A measurement acts as a container for tags, fields, and timestamps. In this case, the measurement is `census`.

We also recommend taking a quick look at the [InfluxDB key concepts](#).

6.2 - Install InfluxDB (Raspberry Pi)

In this Unit, you'll learn how to install InfluxDB on the Raspberry Pi.



As mentioned previously, we'll use InfluxDB locally hosted on the Pi board. However, you can use the cloud plan if you prefer. You just need to sign up for a free account and you are ready to go: <https://cloud2.influxdata.com/signup>

Follow the next instructions to install InfluxDB on the Raspberry Pi. You can also check the following link to check the official installation instructions:

- <https://www.influxdata.com/downloads/> (follow the Ubuntu & Debian ARM 64-bit instructions)

Run the following commands sequentially to install InfluxDB on your server.

```
wget -q https://repos.influxdata.com/influxdata-archive_compatible.key
```

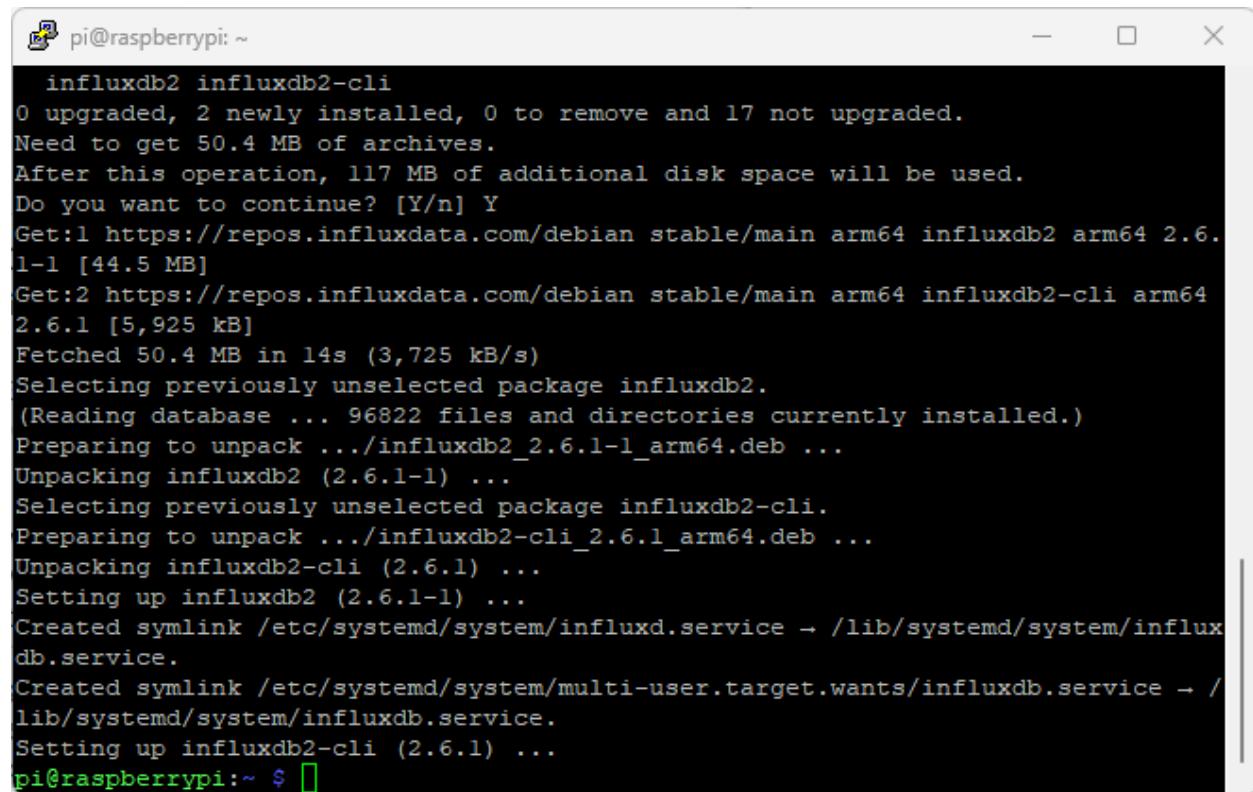
```
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c' | sha256sum -c && cat influxdata-archive_compatible.key | gpg --dearmor | sudo tee /etc/apt/trusted.gpg.d/influxdata-archive_compatible.gpg > /dev/null
```

```
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compatible.gpg] https://repos.influxdata.com/debian stable main' | sudo tee /etc/apt/sources.list.d/influxdata.list
```

```
sudo apt-get update && sudo apt-get install influxdb2
```

When prompted, press **Y** and hit **Enter**.

After installing, the Terminal window should look as follows:



```
pi@raspberrypi: ~
influxdb2 influxdb2-cli
0 upgraded, 2 newly installed, 0 to remove and 17 not upgraded.
Need to get 50.4 MB of archives.
After this operation, 117 MB of additional disk space will be used.
Do you want to continue? [Y/n] Y
Get:1 https://repos.influxdata.com/debian stable/main arm64 influxdb2 arm64 2.6.1-1 [44.5 MB]
Get:2 https://repos.influxdata.com/debian stable/main arm64 influxdb2-cli arm64 2.6.1 [5,925 kB]
Fetched 50.4 MB in 14s (3,725 kB/s)
Selecting previously unselected package influxdb2.
(Reading database ... 96822 files and directories currently installed.)
Preparing to unpack .../influxdb2_2.6.1-1_arm64.deb ...
Unpacking influxdb2 (2.6.1-1) ...
Selecting previously unselected package influxdb2-cli.
Preparing to unpack .../influxdb2-cli_2.6.1_arm64.deb ...
Unpacking influxdb2-cli (2.6.1) ...
Setting up influxdb2 (2.6.1-1) ...
Created symlink /etc/systemd/system/influxd.service → /lib/systemd/system/influxdb.service.
Created symlink /etc/systemd/system/multi-user.target.wants/influxdb.service → /lib/systemd/system/influxdb.service.
Setting up influxdb2-cli (2.6.1) ...
pi@raspberrypi:~ $
```

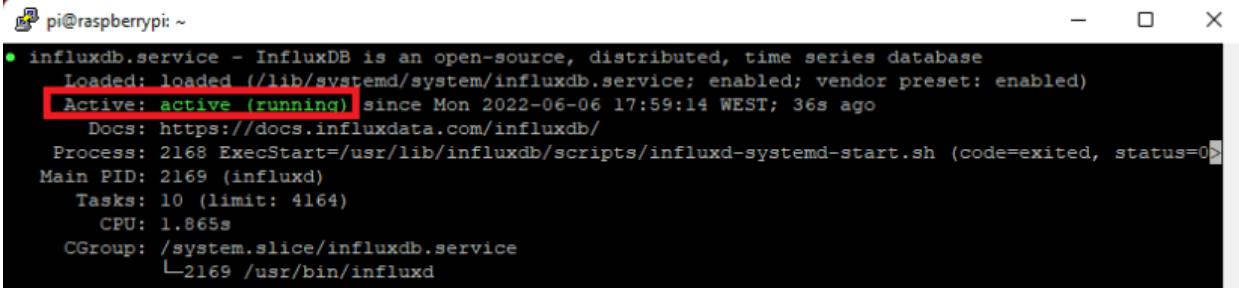
Start InfluxDB by running the following command:

```
sudo service influxdb start
```

Check if InfluxDB is running:

```
sudo service influxdb status
```

You should get something as follows. That means that InfluxDB is successfully running as a service.

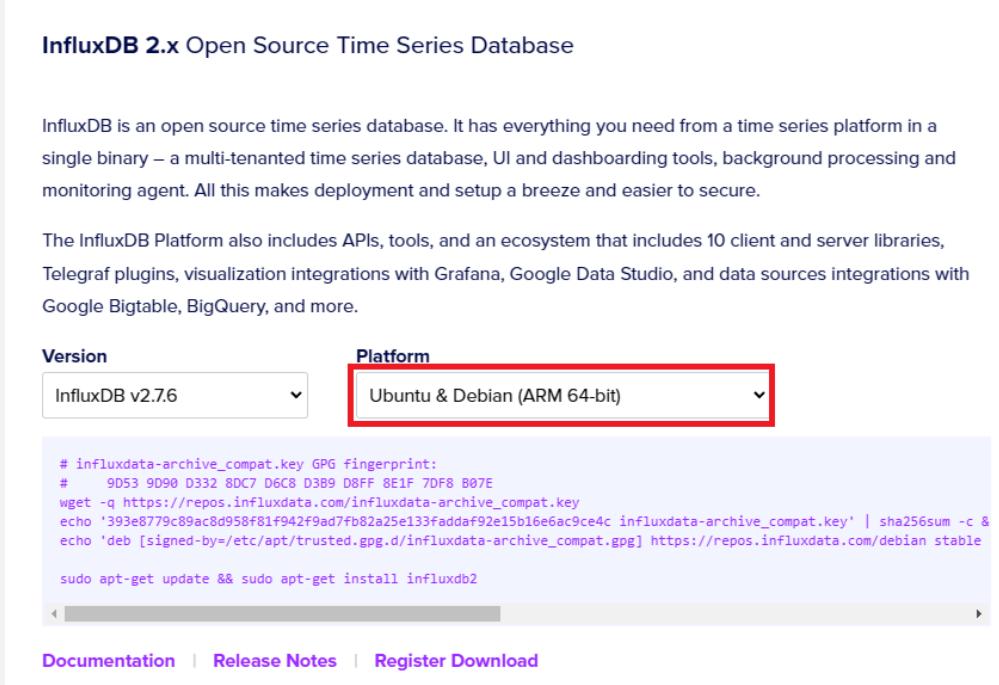


```
pi@raspberrypi: ~
● influxdb.service - InfluxDB is an open-source, distributed, time series database
  Loaded: loaded (/lib/systemd/system/influxdb.service; enabled; vendor preset: enabled)
  Active: active (running) since Mon 2022-06-06 17:59:14 WEST; 36s ago
    Docs: https://docs.influxdata.com/influxdb/
 Process: 2168 ExecStart=/usr/lib/influxdb/scripts/influxd-systemd-start.sh (code=exited, status=0)
Main PID: 2169 (influxd)
  Tasks: 10 (limit: 4164)
    CPU: 1.865s
   CGroup: /system.slice/influxdb.service
           └─2169 /usr/bin/influxd
```

The instructions are not working?

If the instructions are not working it might be due to an update on the installation procedure. If that's the case, we recommend that you go to the next link and follow the official instructions:

<https://www.influxdata.com/downloads/>



InfluxDB 2.x Open Source Time Series Database

InfluxDB is an open source time series database. It has everything you need from a time series platform in a single binary – a multi-tenanted time series database, UI and dashboarding tools, background processing and monitoring agent. All this makes deployment and setup a breeze and easier to secure.

The InfluxDB Platform also includes APIs, tools, and an ecosystem that includes 10 client and server libraries, Telegraf plugins, visualization integrations with Grafana, Google Data Studio, and data sources integrations with Google Bigtable, BigQuery, and more.

Version **Platform**

```
# influxdata-archive_compat.key GPG fingerprint:
#      9D53 9D90 D332 8DC7 D6C8 D3B9 D8FF 8E1F 7DF8 B07E
wget -q https://repos.influxdata.com/influxdata-archive_compat.key
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c influxdata-archive_compat.key' | sha256sum -c &
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg] https://repos.influxdata.com/debian stable'
```

[Documentation](#) | [Release Notes](#) | [Register Download](#)

Make sure you select **Ubuntu & Debian (ARM 64-bit)**. Then, simply copy the commands provided.

Accessing InfluxDB on Raspberry Pi

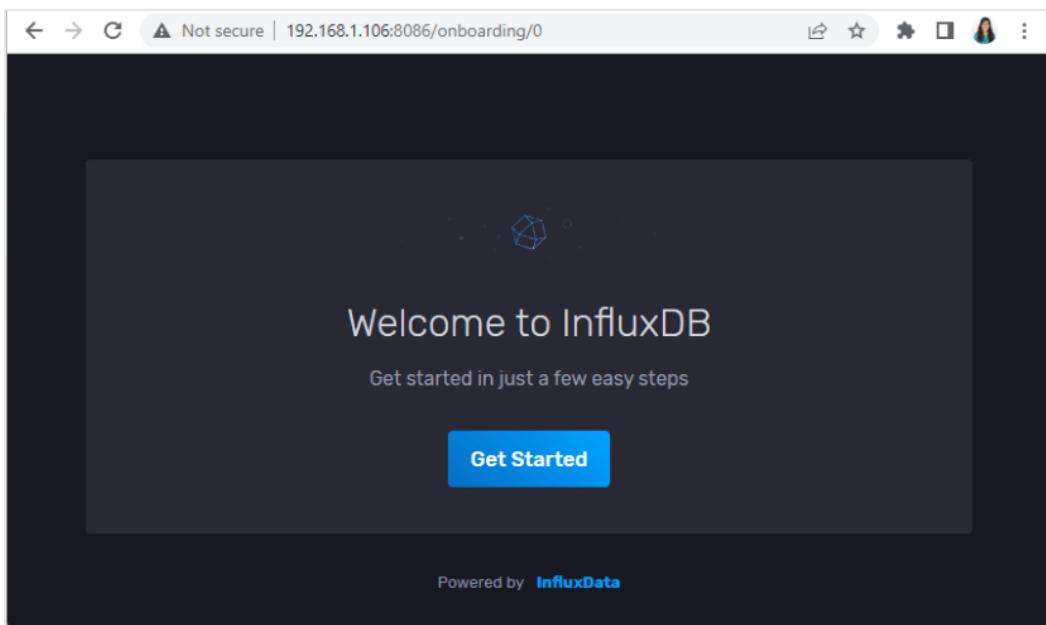
Now, you can access InfluxDB on your Raspberry Pi IP address port 8086. My Raspberry Pi IP address is 192.168.1.106, so to access InfluxDB, I just need to type the following in my web browser:

```
192.168.1.106:8086
```

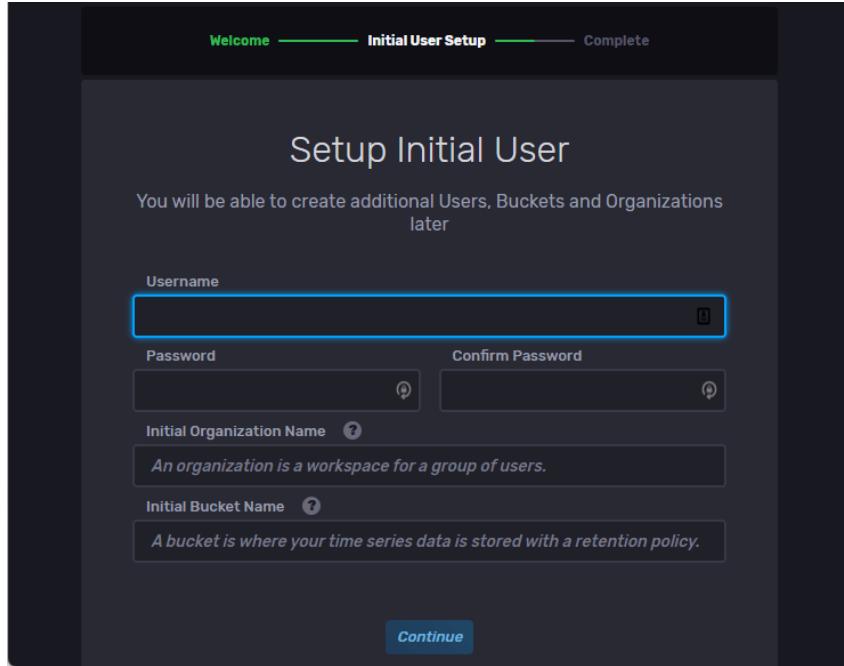
To get your Raspberry Pi IP address, you can run the following command:

```
hostname -I
```

When you first access InfluxDB, you'll see the following screen. Click on **Get Started**.



You'll need to set up an initial user. Fill in the form details, you can use whatever names you want. **You need to remember the username, password, and organization name** so that you can access InfluxDB data later on. Then, click on **Continue** to proceed.



Make sure you copy your operator API token before proceeding. You won't be able to see it again.

The screenshot shows the 'Initial User Setup' step with the message 'You are ready to go!'. A warning box says 'Make sure to copy your operator API token now.' and explains that it enables superuser privileges. Below is the copied token:
AI8GZGE4jcWu4C2qZclsDpy61e7Na7vE2vftmnI_5sRUjC9d6_nJ8EMGmRaHNBN2aRAJe4N7zzvcatgPGa
JdPw==
A 'COPY TO CLIPBOARD' button is shown below the token. Below the token, a message says 'Your InfluxDB has 1 organization, 1 user, and 1 bucket.' At the bottom, there are three buttons: 'QUICK START', 'ADVANCED', and 'CONFIGURE LATER'.

Then, you can click on **Quick Start**.

Let's start collecting data!

QUICK START

ADVANCED

CONFIGURE LATER

Timing is everything!

This will set up local metric collection and allow you to explore the features of InfluxDB quickly.

Whoa looks like you're an expert!

This allows you to set up Telegraf, scrapers, and much more.

I've got this...

Jump into InfluxDB and set up data collection when you're ready.

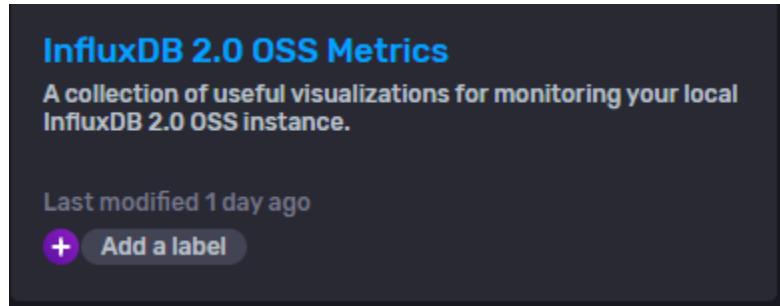
You'll be redirected to the **Getting Started** screen.

The screenshot shows a web browser window titled "Get Started | RNT | InfluxDB". The URL is "192.168.1.79:8086/orgs/62274aaf1a85196e". The main content is titled "Get Started" and includes two large cards: "Python" with its logo and "Node.js" with its logo. To the left is a sidebar with icons for Home, Recent, Dashboard, Metrics, Tasks, and Help. On the right is a sidebar with a search bar, a note about keyboard shortcuts, and a "USEFUL LINKS" section containing links to InfluxDB University, Get Started with Flux, Explore Metrics, Build a Dashboard, Write a Task, Report a bug, and Community Forum.

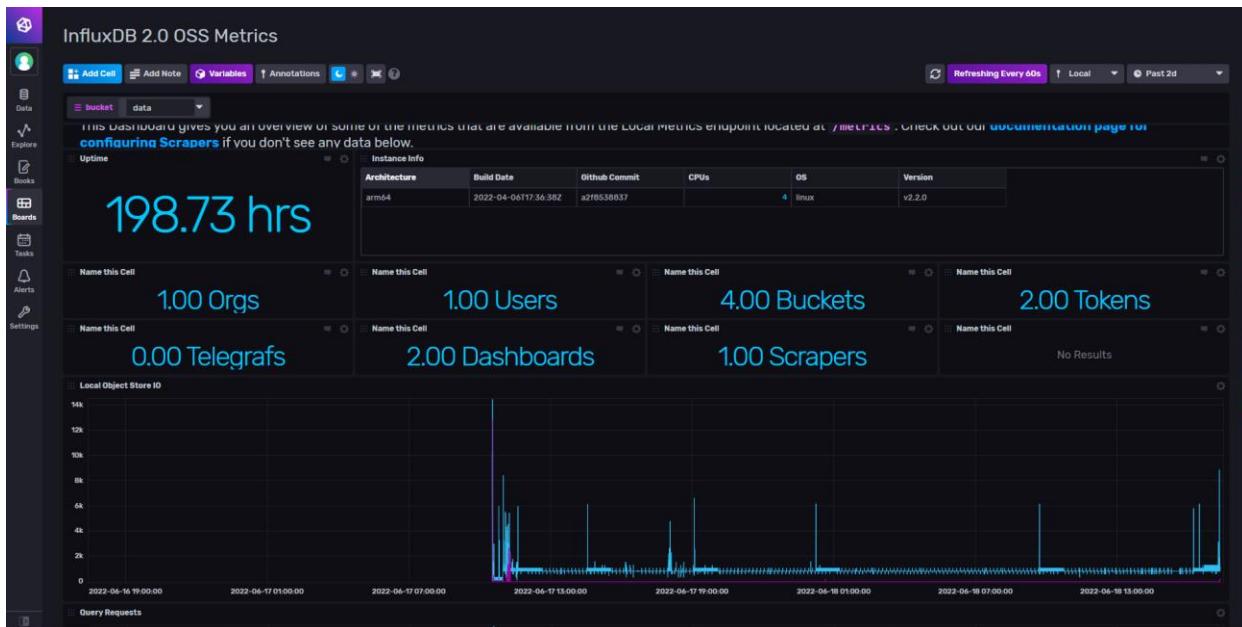
InfluxDB should already have set up a dashboard with some system info for you to explore. On the left sidebar click on the **Dashboard** icon.



Then click **on InfluxDB 2.0 OSS Metrics** "A collection of useful visualizations for monitoring your local InfluxDB 2.0 OSS instance".



Click on that to check some useful information about the local InfluxDB.



Wrapping Up

You've successfully installed InfluxDB on your Raspberry Pi. You can also collect metrics from the Raspberry Pi board (CPU usage, memory usage, RAM usage, CPU usage, etc.) to monitor the system using InfluxDB Telegraf. Telegraf is InfluxData's data collection agent for collecting and reporting metrics. We think this is a useful feature to monitor your Raspberry Pi. We'll cover this subject in the next unit. Following the next unit is optional, and you can skip it if you only want to save the data received on Node-RED (via MQTT) in InfluxDB.

6.3 - Monitoring your Raspberry Pi using InfluxDB Telegraf

You can collect metrics from the Raspberry Pi board (CPU usage, memory usage, disk usage, system load, CPU and GPU temperatures, and other useful data) to monitor the system using InfluxDB Telegraf. Telegraf is InfluxData's data collection agent for collecting and reporting metrics.



Raspberry Pi Monitoring Template

There is an InfluxDB template that can be used to monitor your Raspberry Pi Linux system. An InfluxDB template is a prepackaged set of InfluxDB configurations that contain everything from dashboards and Telegraf configurations to notifications and alerts.

Templates reduce the setup time by giving you resources that are already configured for your use case. In this case, you'll get a set of configurations that are ready to use to monitor your Raspberry Pi. To learn more about InfluxDB templates, you can check the following link:

- [InfluxDB Templates](#)

We'll use the [Raspberry Pi Monitoring template](#) that includes the following:

- one bucket: `rasp-pi` (7d retention)
- labels: `raspberry-pi` + Telegraf plugin labels
 - Diskio input plugin

- Mem input plugin
- Net input plugin
- Processes input plugin
- Swap input plugin
- System input plugin
- one Telegraf configuration
- one dashboard: Raspberry Pi System
- two variables: `bucket` and `linux_host`

Follow the next instructions to install the Raspberry Pi Monitoring template.

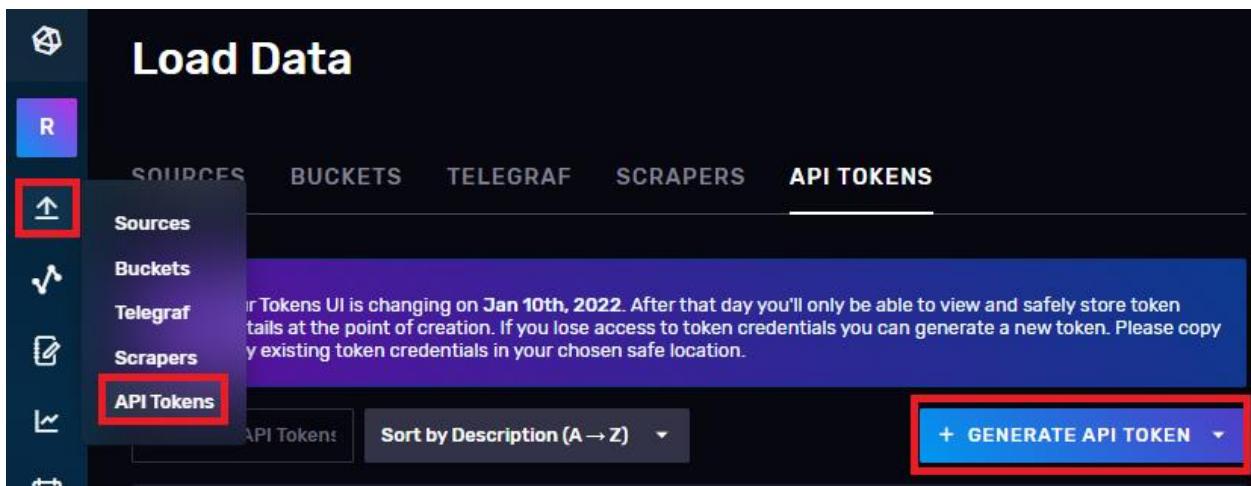
Getting an InfluxDB Token

You need to get an InfluxDB token to be able to install the Raspberry Pi Monitoring template using Influx CLI.

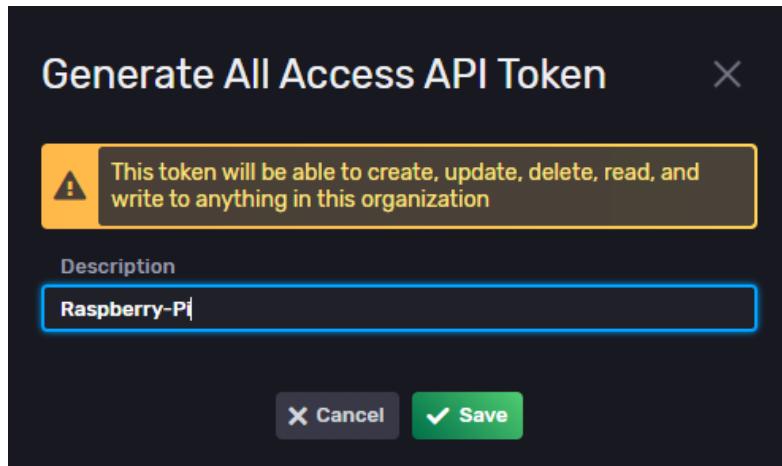
Go to your InfluxDB Interface:

```
http://Your_RPi_IP_address:8086
```

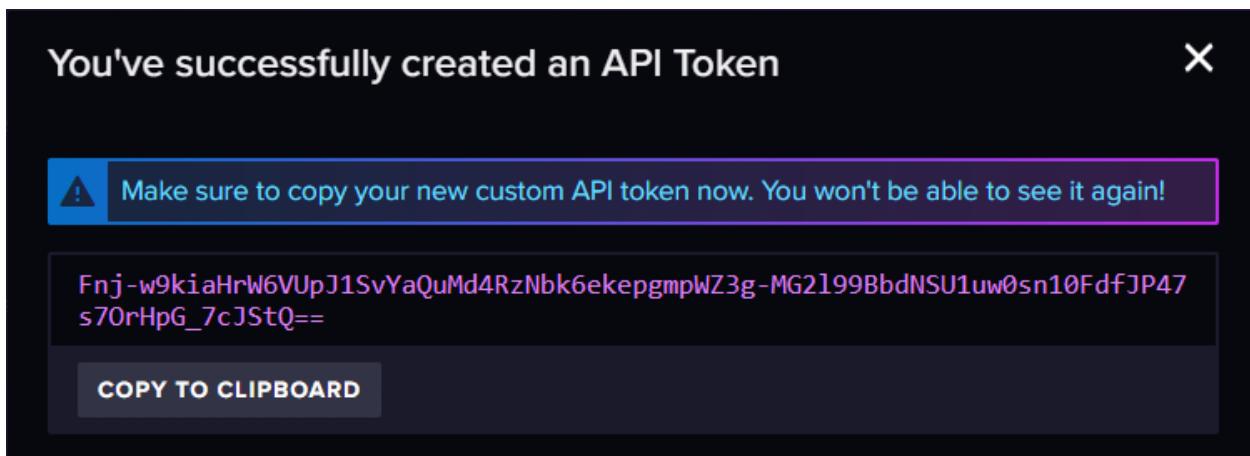
Once you're on InfluxDB, on the left sidebar click on **Data/Load Data** and then, select the **API Tokens** tab. Click on the **+ Generate API Token** to generate a new API Token. We'll generate an **All Access API Token**.



Give a description to the API Token (for example `Raspberry-Pi`).



After that, copy the API token to a safe place. You won't be able to see it again.



The new API Token should be on the list:

The screenshot shows a dark-themed interface for managing API tokens. The top navigation bar includes tabs for SOURCES, BUCKETS, TELEGRAF, SCRAPERS, and API TOKENS, with the latter being the active tab. A purple info box states: "Our Tokens UI has changed. You are only able to view and safely store token details at the point of creation. If you lose access to token credentials, you can generate a new token." Below this are search and filter controls: "Filter Tokens..." and "Sort by Description (A → Z)". A large red box highlights a specific token entry for "Raspberry-Pi". At the bottom of this entry are details: "Created at: 2023-03-08 11:53:57", "Owner: sara", and "Last Modified: 1 second ago". To the right of the token list are "GENERATE API TOKEN" and "DELETE" buttons.

Influx CLI

Influx CLI is InfluxDB command line interface that contains commands to manage many aspects of InfluxDB, including buckets, organizations, users, tasks, etc. It's automatically installed when you install InfluxDB.

Provide required authentication credentials

To avoid having to pass your InfluxDB host, API token, and organization with each command, you can store them in an influx CLI configuration (config). For that, you need to use the `influx config create` command and pass your information.

First, edit the following command with your information and then run it in your Raspberry Pi terminal window:

```
influx config create --config-name influx-config --host-url  
http://YOUR_RASPBERRY_PI_IP_ADDRESS:8086 --org <your-org> --  
token <your-auth-token> --active
```

Replace `YOUR_RASPBERRY_PI_IP_ADDRESS` with your Pi's IP address.

Replace `<your-org>` with your InfluxDB organization name

Replace the `<your-auth-token>` with the API token you got in the previous step.

For example, in my case, the command looks as follows:

```
influx config create --config-name influx-config --host-url  
http://192.168.1.106:8086 --org RNT --token v_od_mG--  
9_srf_OnaaODihPDX34suToP7XEH47v6x77CMxakZaoYHzF7Ec9mLT-  
CuXXXXXXXXXXXXXXXXXXXXvQCSSw== --active
```

Applying the Raspberry Pi Template

Now, run the following command to apply the Raspberry Pi template.

```
influx apply -f  
https://raw.githubusercontent.com/influxdata/community-  
templates/master/raspberry-pi/raspberry-pi-system.yml
```

Installing Telegraf

Run the following commands sequentially to install Telegraf

```
wget -q https://repos.influxdata.com/influxdata-archive_compatible.key
```

```
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c
influxdata-archive_compatible.key' | sha256sum -c && cat influxdata-
archive_compatible.key | gpg --dearmor | sudo tee
/etc/apt/trusted.gpg.d/influxdata-archive_compatible.gpg > /dev/null
```

```
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-
archive_compatible.gpg] https://repos.influxdata.com/debian stable
main' | sudo tee /etc/apt/sources.list.d/influxdata.list
```

```
sudo apt-get update && sudo apt-get install telegraf
```

The instructions are not working?

If the instructions are not working it might be due to an update on the installation procedure. If that's the case, we recommend that you go to the next link and follow the official instructions: <https://www.influxdata.com/downloads/>

Scroll down to the Telegraf section. Select the platform **Ubuntu & Debian**. Then, copy the commands provided.

Telegraf open source data collector

Telegraf is a plugin-driven server agent for collecting and sending metrics and events from databases, systems, and IoT sensors. Telegraf is written in Go and compiles into a single binary with no external dependencies, and requires a very minimal memory footprint.

With 200+ plugins already written by subject matter experts on the data in the community, it is easy to start collecting metrics from your endpoints.

For additional architecture (e.g. i386, riscv64, etc.) and operating system (e.g BSD, etc.) downloads please see the [Telegraf GitHub Releases](#) page.



Now, you need to add the following environment variables to your Telegraf environment:

- `INFLUX_HOST`: InfluxDB URL `http://YOUR_RASPBERRY_PI_IP_ADDRESS:8086`
- `INFLUX_TOKEN`: Your InfluxDB Cloud API token
- `INFLUX_ORG`: Your InfluxDB Cloud organization name.

For that, edit the following commands with your own details:

```
export INFLUX_HOST=http://YOUR_RASPBERRY_PI_IP_ADDRESS:8086
```

```
export INFLUX_TOKEN=YOUR_API_TOKEN
```

```
export INFLUX_ORG=YOUR_ORG_NAME
```

For example, in my case, the commands look as follows:

```
export INFLUX_HOST=http://192.168.1.106:8086
```

```
export INFLUX_TOKEN=v_od_mG--9_srf_OnaaODihPDX34suToP7XEH47v6x
```

```
export INFLUX_ORG=RNT
```

Then, run it on your Raspberry Pi one command at a time.

 pi@raspberrypi: ~
pi@raspberrypi:~ \$ export INFLUX_HOST=
pi@raspberrypi:~ \$ export INFLUX_TOKEN=
pi@raspberrypi:~ \$ export INFLUX_ORG=

Starting Telegraf

Start Telegraf as a service. For the exact command, you need to go to your InfluxDB interface, and then **Data/Load Data > Telegraf**.



Click on the rpi **Setup Instructions** link.

A screenshot of the InfluxDB 'Load Data' screen. The 'TELEGRAF' tab is selected. On the left is a sidebar with various icons. The main area shows a table with one row. The row has a 'rpi' label, 'rpi monitoring' description, 'Bucket: rasp-pi' label, and a 'Setup Instructions' button. The 'Setup Instructions' button is highlighted with a red box. There are other buttons for 'Edit' and 'Delete' in the same row. At the bottom of the table, there is a '+' button followed by the text 'raspberry-pl' and a 'X' button.

Copy the command to start Telegraf.

Telegraf Setup Instructions

1. Install the Latest Telegraf

You can install the latest Telegraf by visiting the [InfluxData Downloads](#) page. If you already have Telegraf installed on your system, make sure it's up to date. You will need version 1.9.2 or higher.

2. Configure your API Token

Your API token is required for pushing data into InfluxDB. You can copy the following command to your terminal window to set an environment variable with your API token.

```
export INFLUX_TOKEN=<INFLUX_TOKEN>
```

[Copy to Clipboard](#) [Generate New API Token](#) CLI

3. Start Telegraf

Finally, you can run the following command to start the Telegraf agent running on your machine.

```
telegraf --config http://192.168.1.106:8086/api/v2/telegrafs/09d5e49737bd5000
```

[Copy to Clipboard](#) CLI

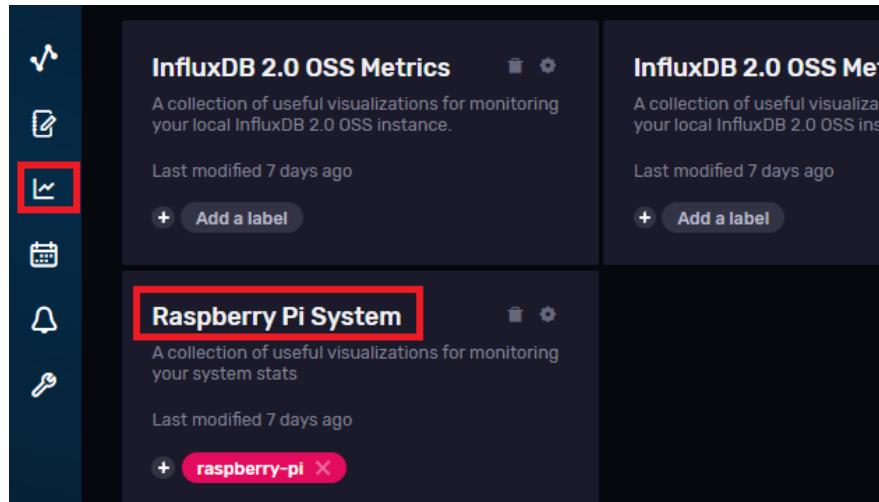
Establish an SSH connection and run that command to start Telegraf. You should get something similar to the next image in your Terminal window.

```
pi@raspberrypi:~ $ telegraf --config http://192.168.1.106:8086/api/v2/telegrafs/09d5e49737bd5000
2022-08-16T17:08:09Z I! Starting Telegraf 1.23.3
2022-08-16T17:08:09Z I! Loaded inputs: cpu disk diskio exec mem net processes swap system temp
2022-08-16T17:08:09Z I! Loaded aggregators:
2022-08-16T17:08:09Z I! Loaded processors:
2022-08-16T17:08:09Z I! Loaded outputs: influxdb_v2
2022-08-16T17:08:09Z I! Tags enabled: host=raspberrypi
2022-08-16T17:08:09Z I! [agent] Config: Interval:10s, Quiet:false, Hostname:"raspberrypi", Flush I
```

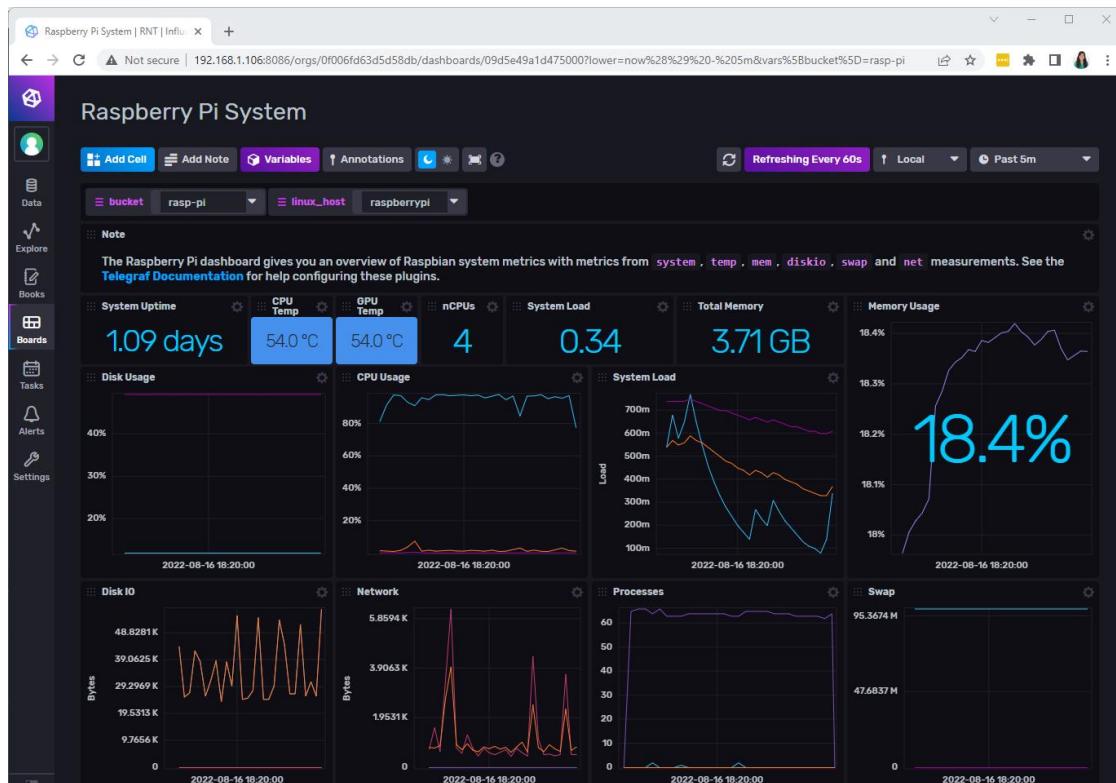
At the moment, Telegraf should be collecting data from the Raspberry Pi and sending it to the corresponding bucket on InfluxDB.

Raspberry Pi System Dashboard

Now you can monitor your Raspberry Pi system on a Dashboard in InfluxDB. In your InfluxDB user interface, go to **Boards/Dashboards** (left sidebar) and click on the **Raspberry Pi System** "A collection of useful visualizations for monitoring your system stats".

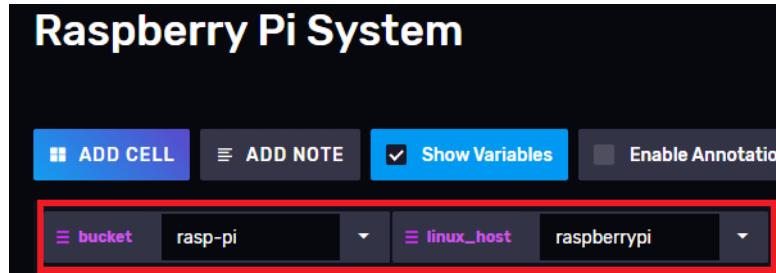


The following Raspberry Pi System Dashboard loads.



You'll get access to the Raspberry Pi System Dashboard. You can check useful information like the CPU and GPU temperatures, total memory, memory usage, and much more.

If you don't see any data on the dashboard, make sure you select the right bucket `rasp-pi` and the `linux_host` is `raspberrypi` (or whatever name you've given to your Raspberry Pi).



Congratulations! You set up a system on InfluxDB to monitor your Raspberry Pi.

In the next unit, you'll learn how to send your sensors' data (received on Node-RED) to InfluxDB.

6.4 - MQTT + Node-RED + InfluxDB

At this point, we assume you have an ESP32 or ESP8266 sending sensor readings to Node-RED via MQTT (see projects 5.3 and 5.4). We'll save the readings that Node-RED receives via MQTT in the InfluxDB database.

Creating a Bucket

We'll start by creating a new bucket in InfluxDB to save your data. As we've seen previously a bucket is a named location where your data is stored.

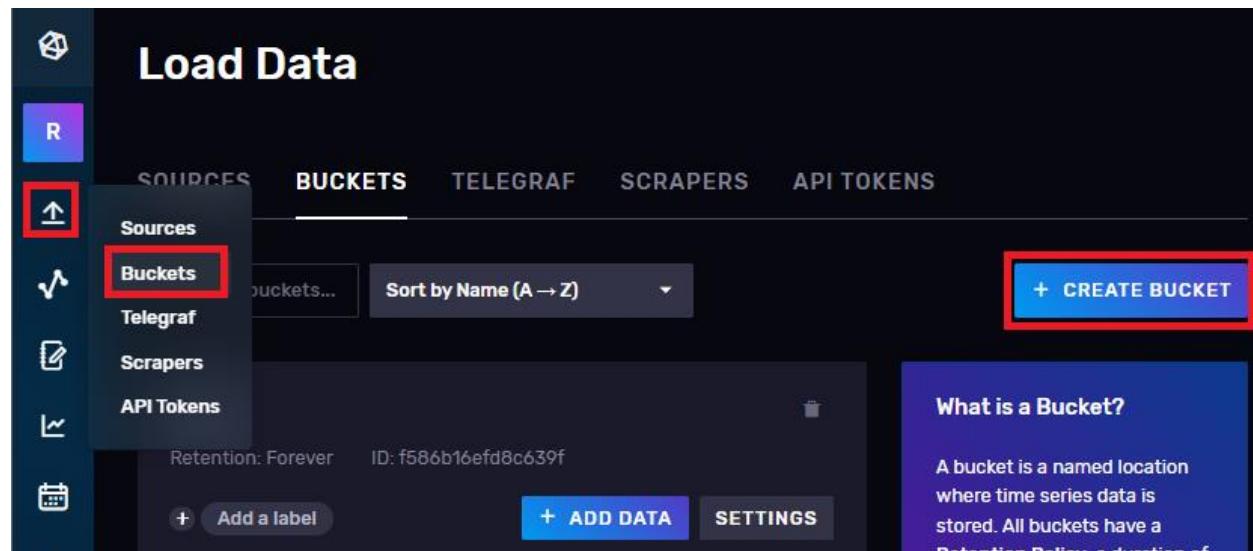
Go to your InfluxDB dashboard on the following link:

```
http://Your_RPi_IP_address:8086
```

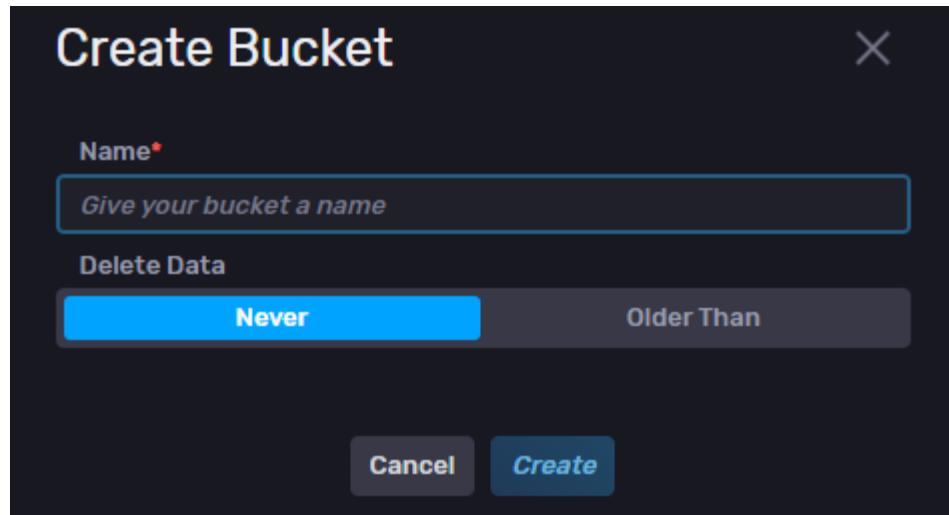
For example, in my case:

```
http://192.168.1.106:8086
```

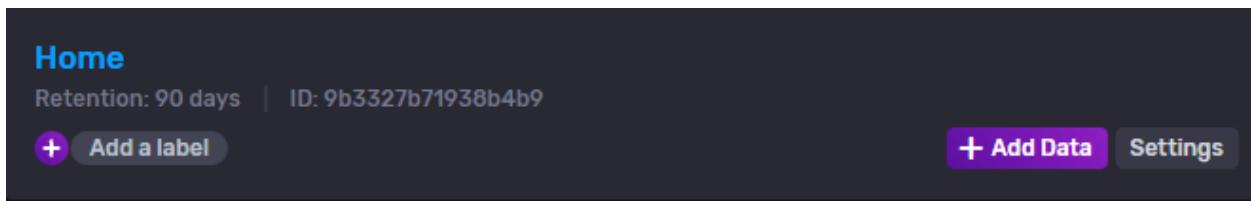
On the left sidebar click on **Data/Load Data** and then, select the **Buckets** tab. Click on the **+ Create Bucket** to create a new bucket.



Give a name to your bucket, for example, Home, and define a retention policy (**Older Than**) for the bucket.



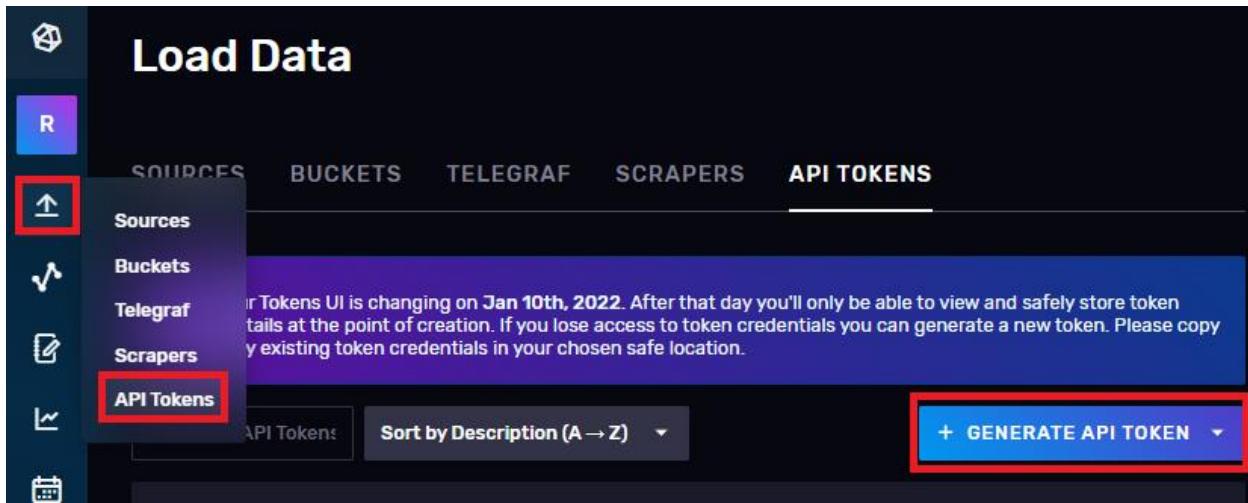
Your new bucket should show up on the list of buckets.



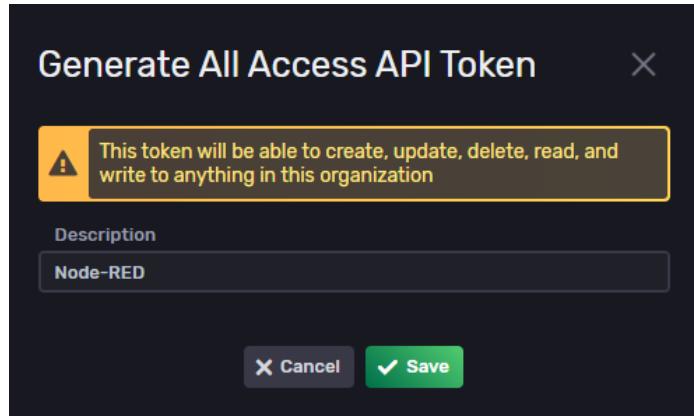
Getting an InfluxDB Token

You need to get an InfluxDB token to be able to interact with the database from Node-RED.

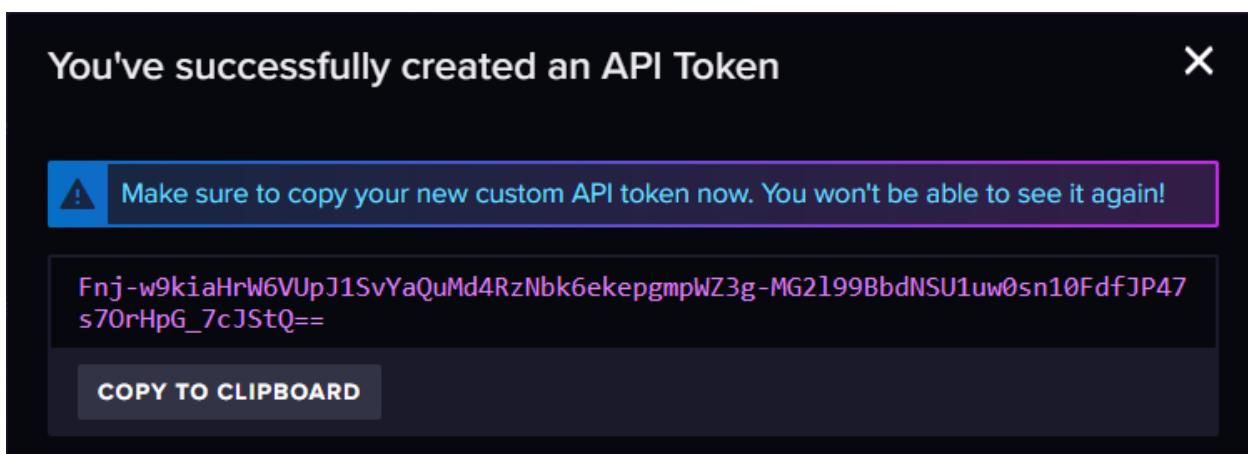
On the left sidebar click on **Data** and then, select the **API Tokens** tab. Click on the **+ Generate API Token** to generate a new API Token. We'll generate an **All Access API Token**.



Give a description to the API Token (for example Node-RED because that's where we'll be using the token).



Copy the API token to a safe place. You won't be able to see it again.

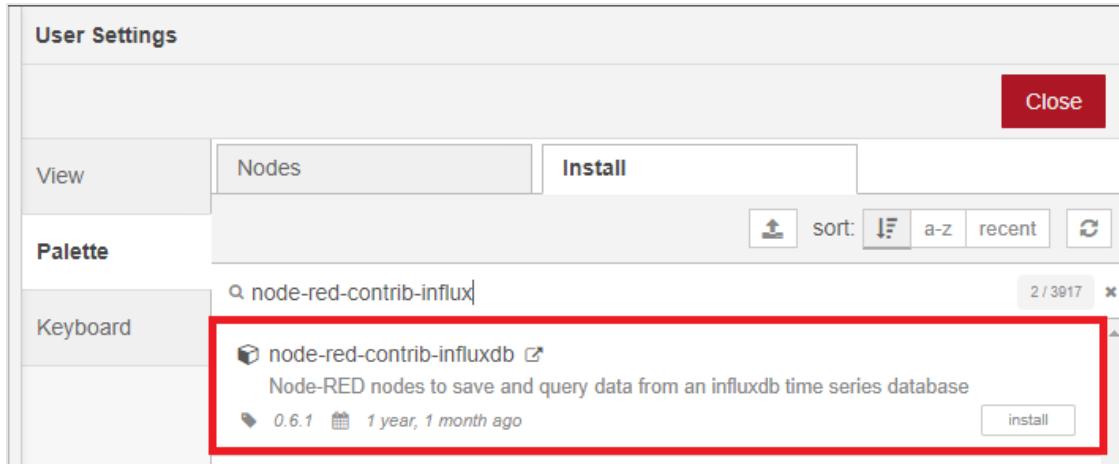


Install InfluxDB Nodes

There are some nodes that make it easy to establish a communication between Node-RED and InfluxDB.

Go to Node-RED. It's on the Raspberry Pi IP address, port 1880.

Go to **Menu** (top right corner) > **Manage Palette** > **Install** and search for `node-red-contrib-influxdb`. Install those nodes.



After installing you should have the InfluxDB nodes on the palette under the **storage** section.



Organizing your Data

To understand how you must save your data in InfluxDB, we recommend taking a quick look at the following InfluxDB tutorial:

<https://docs.influxdata.com/influxdb/v2.2/reference/key-concepts/data-elements/>

The actual data is stored in the *field* value. The field key describes the field value. For example, the field key can be `temperature`, and the field value `25` (the actual temperature value). Your fields need to be associated with a *measurement*. It describes the data stored and the associated fields. So, we can have different fields in the same measurement. For example, besides the `temperature`, you can also have `humidity`.

Additionally, you can also add *tags* to your measurements to better organize the data. For example, imagine you have multiple devices publishing temperature, and humidity. To distinguish between the readings, you can add a tag name `device` that describes which device published the reading. You can add multiple tags to your readings. For example, you could have another tag called `location` that describes where the device is taking the readings.

For our example, we'll organize the data as follows:

- **measurement:** `readings`
- **tags:** `device`, `sensor`, and `location`
- **fields:** `temperature`, `humidity`, and `pressure`

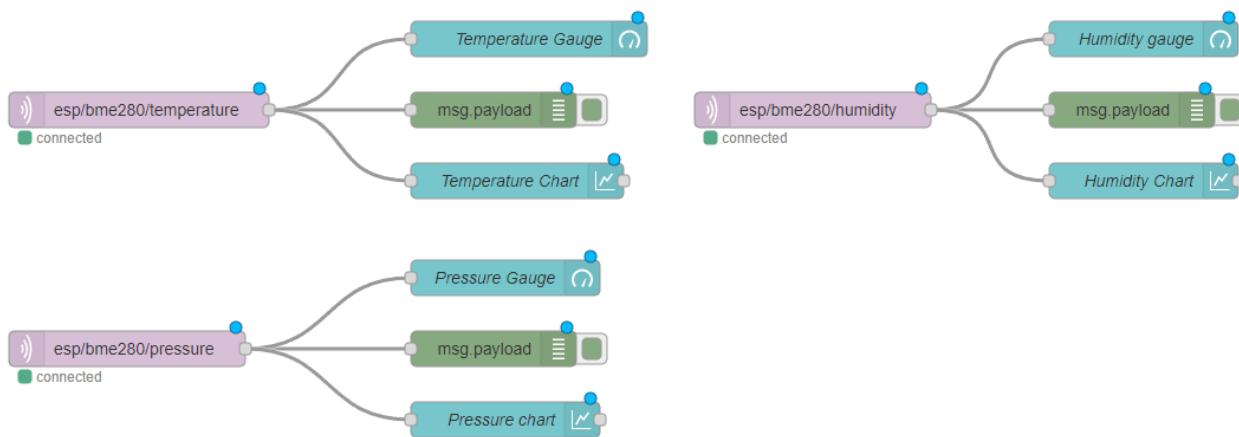
Here's an example of what your bucket data would look like:

| time | measurement | device | sensor | location | field | value |
|-------------|-------------|--------|--------|----------|-------------|-------|
| <timestamp> | readings | ESP32 | BME280 | office | temperature | 25.02 |
| <timestamp> | readings | ESP32 | BME280 | office | humidity | 52 |

| | | | | | | |
|-------------|----------|---------|--------|--------|-------------|---------|
| <timestamp> | readings | ESP8266 | BME280 | room | pressure | 1003.05 |
| <timestamp> | readings | ESP32 | BME280 | office | temperature | 27.05 |

Creating the Flow

At the moment, you should have the following nodes on the flow that receive the sensor readings from the ESP32/ESP8266 via MQTT.



If you don't, you can download the flow on the following link (don't forget you need to edit the MQTT broker details):

- [Download Node-RED Flow](#)

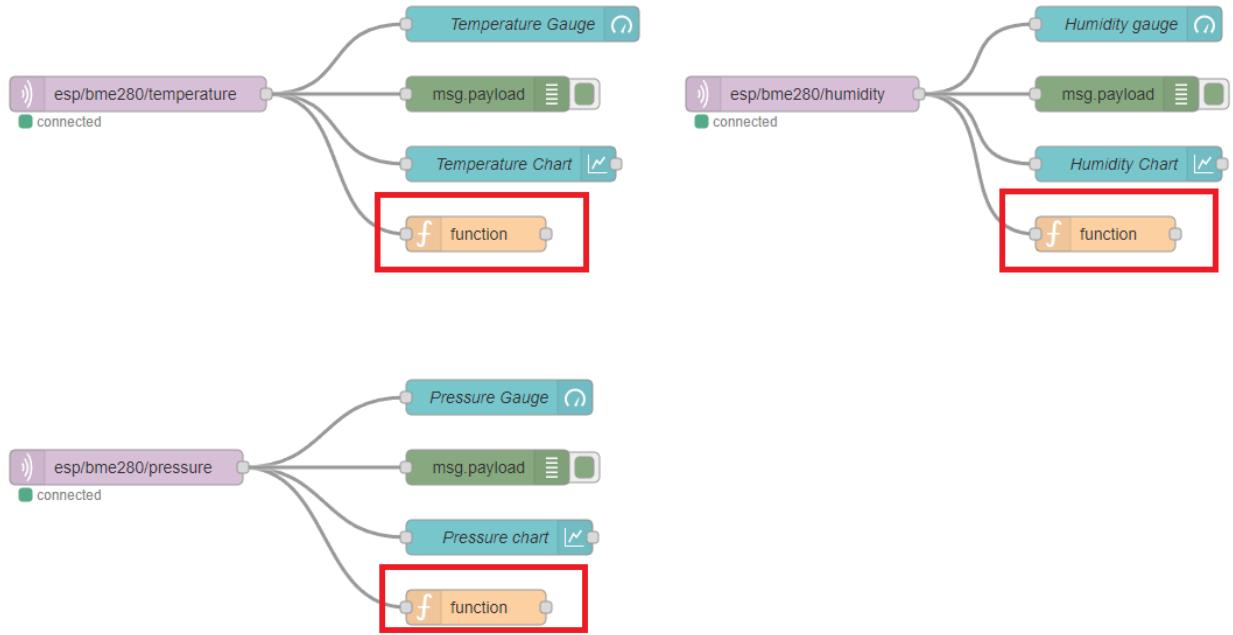
The readings received from the ESP32/ESP8266 are String variables. We need to convert the String to a number before sending it to InfluxDB. That value is the *field value* on our bucket. To do that, we can use a **function** node, and then, write the appropriate code to output the message

Additionally, we also need to modify the payload to include the *field key*, and *tags* if needed.

Drag **three function nodes** to the flow and wire each of them to one of the **MQTT in** nodes.



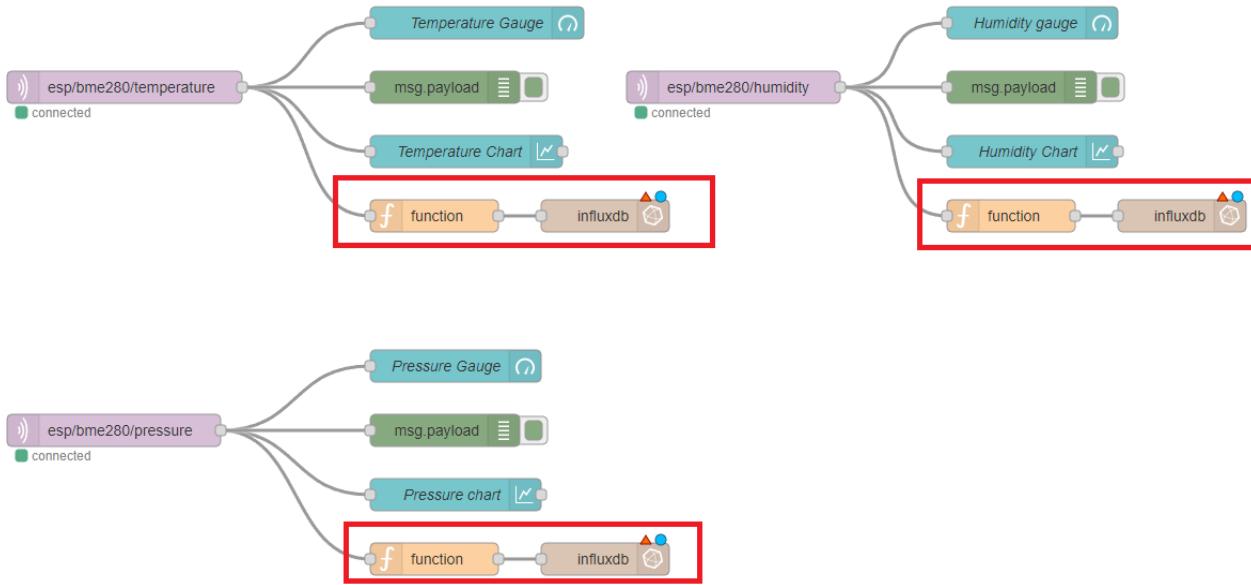
Here's what the flow looks like at the moment:



Then, wire an **influxdbout** node to each of the function nodes.

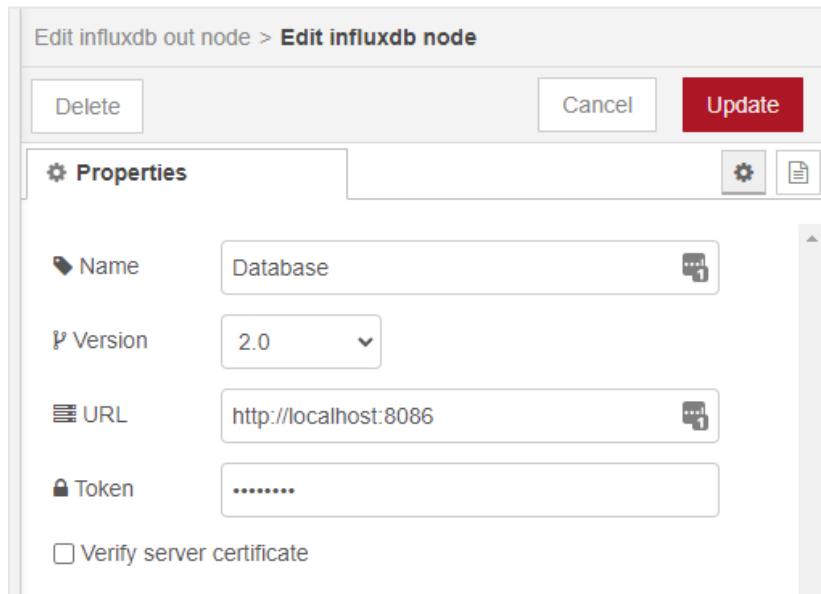


The flow will look as follows:



Configuring InfluxDB Nodes

Now, you need to edit the InfluxDB nodes. Double-click the InfluxDB node for the temperature. Then, on the Server option click on **Add new influxdb**.



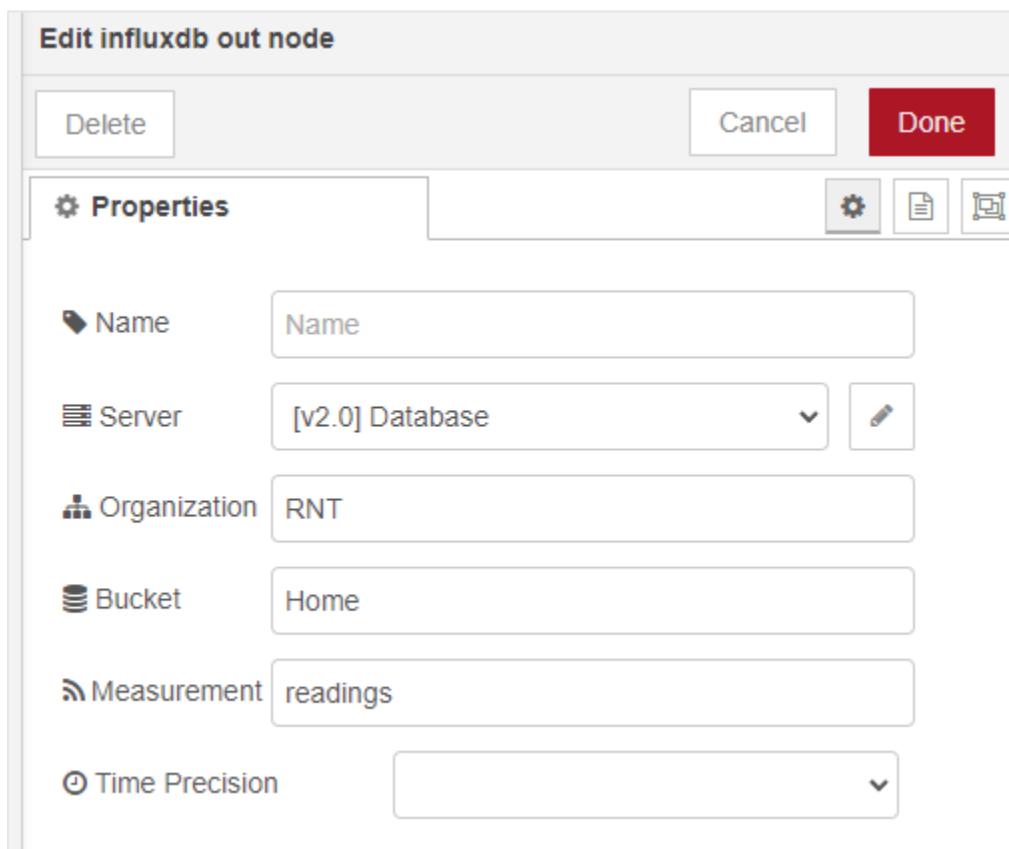
Select version 2.0 and insert the URL as shown in the picture above (InfluxDB is running on the localhost (Raspberry Pi) on port 8086:

<http://localhost:8086>

Insert the API token you got in the previous section on the **Token** field.

Finally, click **Update**.

Continue editing the InfluxDB out node. Insert the Organization name, bucket, and a name for the measurement—as we've mentioned previously the measurement will be called `readings`.



When you're finished, click on **Done**. Repeat the same process for the other **InfluxDB out** nodes.

If you select an **influxdbout** node and click on the **help** tab (right sidebar > book icon). You can see how to format `msg.payload` to pass the parameters we want. It mentions:

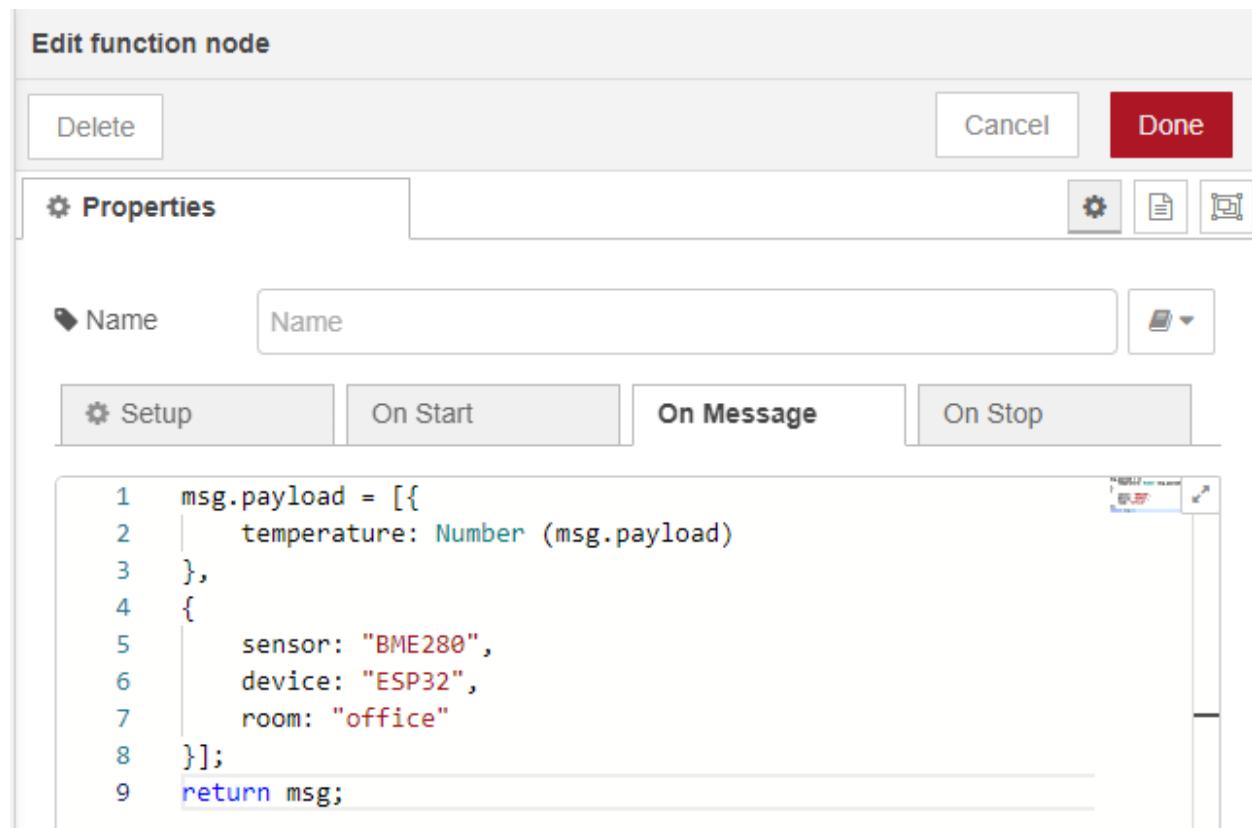
- “The fields and tags to write are in `msg.payload`.”

- “If `msg.payload` is an array containing two objects, the first object will be written as the set of named fields, the second is the set of named tags.”

Configuring the Function Nodes

Now, let’s edit the **function** nodes so that the `msg.payload` contains the field and tags. Double-click the function node for the temperature, and copy the following code to the **On Message** tab.

```
msg.payload = [
  temperature: Number (msg.payload)
],
{
  sensor: "BME280",
  device: "ESP32",
  room: "office"
];
return msg;
```



`temperature` is the field. `Number(msg.payload)` is the field value. `msg.payload` is the value received from the **MQTT In** node that comes in string format as mentioned previously,

Then, we have `sensor`, `device` and `room` that are the tags that we want to associate with that measurement. You can remove or add different tags that might make more sense for your application.

Repeat the same process for the other function nodes, but with different field values.

For the humidity function:

```
msg.payload = [{  
    humidity: Number(msg.payload)  
},  
{  
    sensor: "BME280",  
    device: "ESP32",  
    room: "office"  
}];  
return msg;
```

And finally, for the pressure function.

```
msg.payload = [{  
    pressure: Number(msg.payload)  
},  
{  
    sensor: "BME280",  
    device: "ESP32",  
    room: "office"  
}];  
return msg;
```

After making all the required changes. Deploy your application.



Alternatively, you can download the flow for this example on the following link. You'll need to edit the MQTT nodes with your credentials and the InfluxDB nodes with your own details.

- [Download Node-RED flow](#)

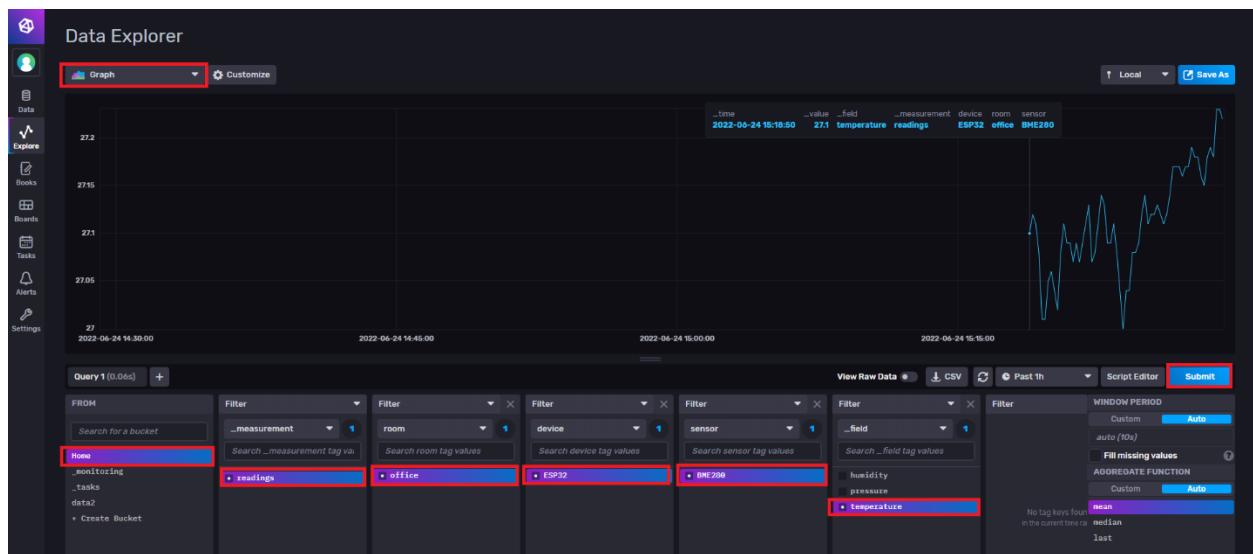
Visualizing your Data

To visualize your data, go to InfluxDB and click on the **Explore/Data Explorer** icon on the left sidebar.



Now, you can visualize your data. Start by selecting the bucket you want. Then, we need to add filters to select our data. Select the `readings` under the `_measurement` field, the **device**, **location**, and **sensor**. Then, you can select the temperature, humidity, or pressure values. You can also select all readings if you want to plot them all on the same chart.

After making the query, click on the **SUBMIT** button. This will display your data in your chosen format. In the upper left corner, you can select different ways to visualize the data. You can also click on the **CUSTOMIZE** button to change the color of the series and other options.

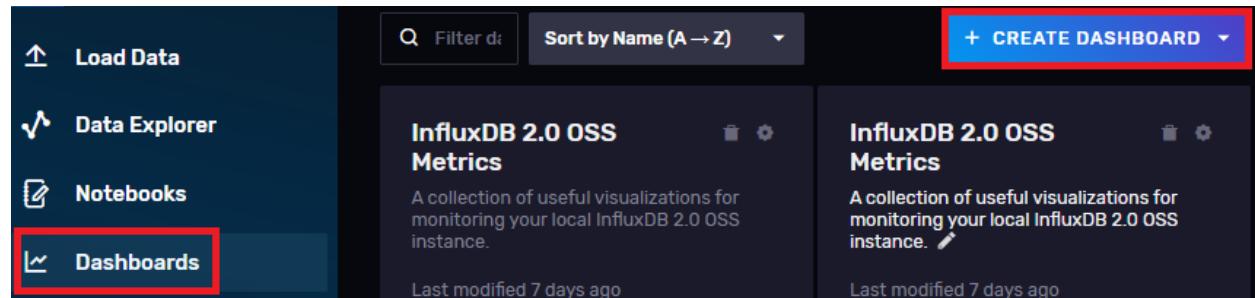


We recommend that you spend some time exploring the many options that the **Data Explorer** has to offer.

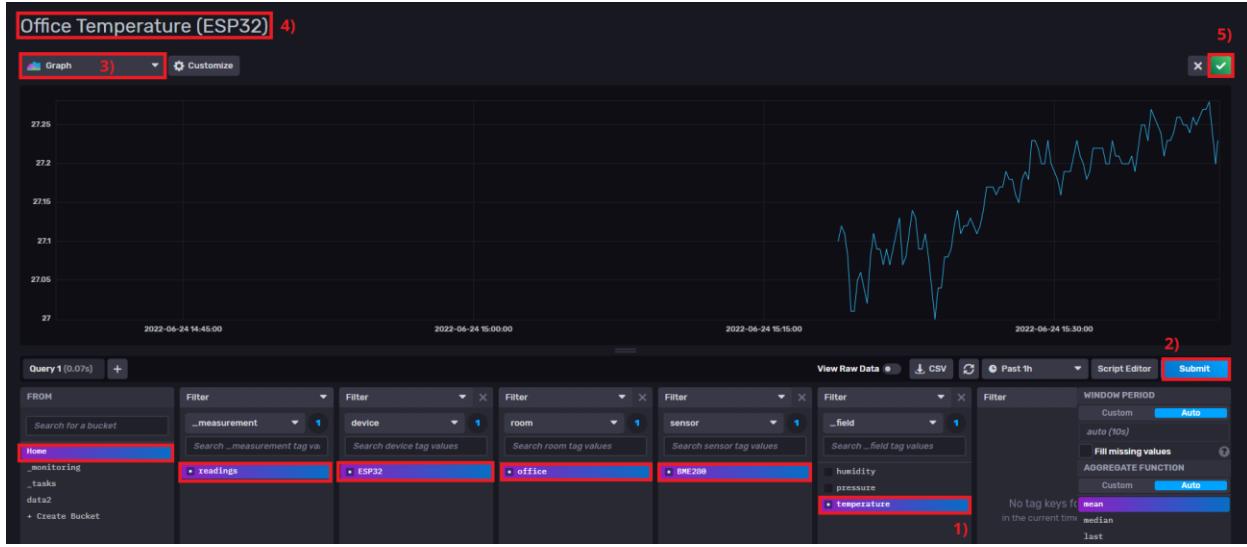
You can create a dashboard to show multiple data visualizations in different formats (gauges, histogram, single stat, etc.) or different data on the same page. For example, multiple charts to show temperature, humidity, and pressure, and boxes to show the current measurements.

Creating a Dashboard

Click on the **Boards/Dashboards** icon and then on **Create Dashboard > New dashboard**.

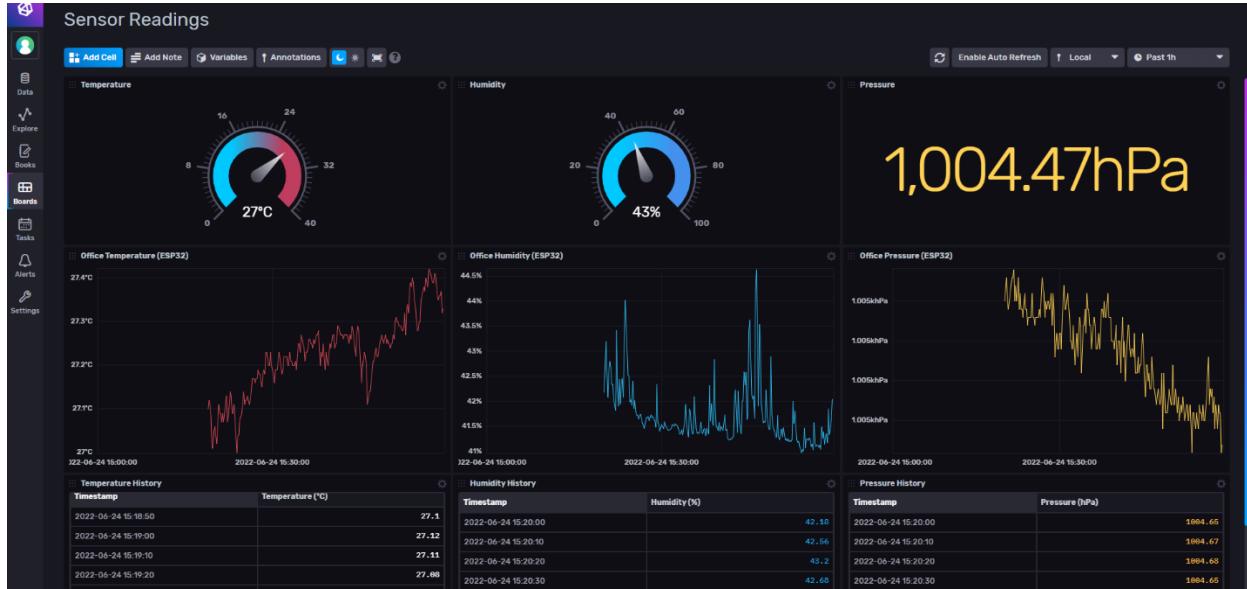


Add a cell. Make the query to get your data and select the visualization you want. Give a name to the cell, for example, `Office Temperature(ESP32)`. You can also click on the **Customize** button to customize the graph (we suggest selecting different colors for temperature, humidity, and pressure). Finally, click on the ✓ icon in the top right corner to add the visualization as a cell to your dashboard.



Repeat the same process for the other readings (humidity, and pressure). You can also add a single stat or gauge to show the current values of each reading. You can add a table if you want to check the detailed values. The interface is pretty straightforward to use and there are several menus to edit your data visualizations. We recommend exploring those menus to see what's available.

For example, I created the following dashboard using the data that's coming from the ESP.



You can move your cells to different positions and organize the dashboard in a way that makes sense for you. You can also customize the way the data is refreshed and how many data points you want to see. For more information about creating different types of charts, we recommend taking a look at the following link:

- <https://docs.influxdata.com/influxdb/cloud/visualize-data/visualization-types/graph/>

Wrapping Up

With this example, you learned how to send data from Node-RED to InfluxDB and how to query and visualize your data. At the moment, we're just displaying data from one ESP board. The idea is to have multiple boards in multiple rooms and several sensors, and then, organize all your data in InfluxDB. The tags will make it easy to sort, visualize and organize your data.

6.5 - Monitoring GPIO States on InfluxDB

Besides monitoring sensor readings, it may also be useful to have a record of the GPIO states on InfluxDB. You'll be able to check when a determined GPIO was on or off. This may be useful if you're controlling a lamp or any other appliance and you want to check when it was on or off.

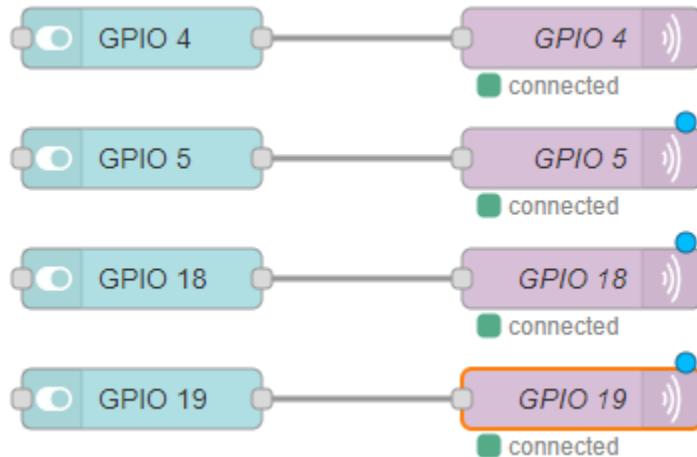
We assume that you have the flow created in project 5.4. That flow controls the ESP GPIOs via MQTT. You can download the flow on the following link (remember you need to edit the flow with your MQTT broker details):

- [Download Node-RED flow](#)

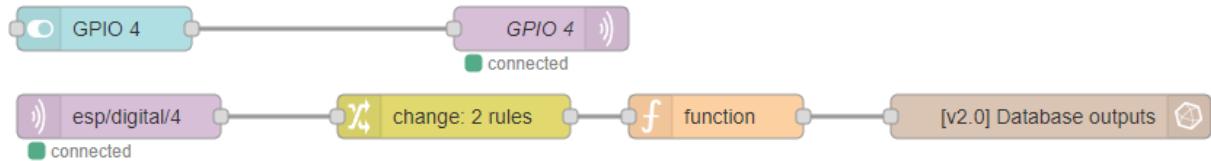
Project Overview

In this project, we'll add some nodes to the flow to save the GPIO state in InfluxDB whenever it changes.

This is the flow from Unit 5.4. It consists of switches that publish `true` or `false` messages on the topics that the ESP is subscribed to.

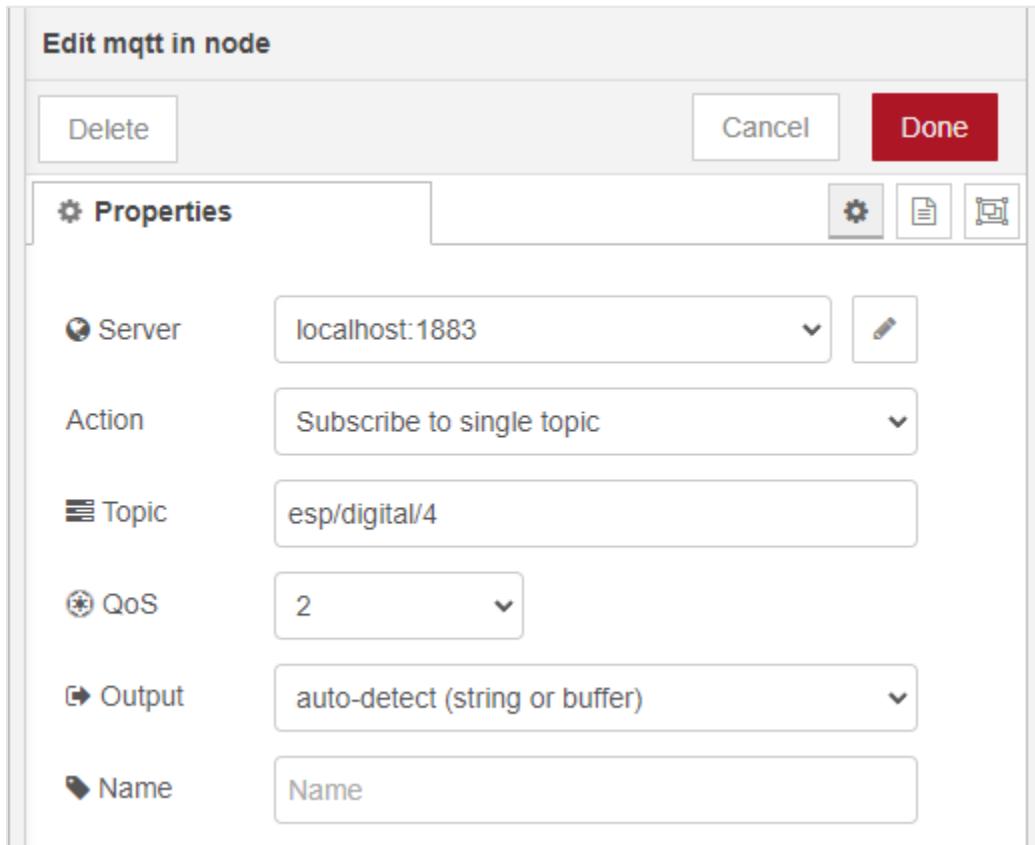


There are different ways to send the GPIO state to InfluxDB. We'll do it as shown in the following flow. The following picture shows an example for GPIO 4. It's the same for the other GPIOs— you just need to change the topic.



We added an **MQTT In** node that is subscribed to the topic that the switch is publishing in. This way, we receive on that node whenever there's a change in the state.

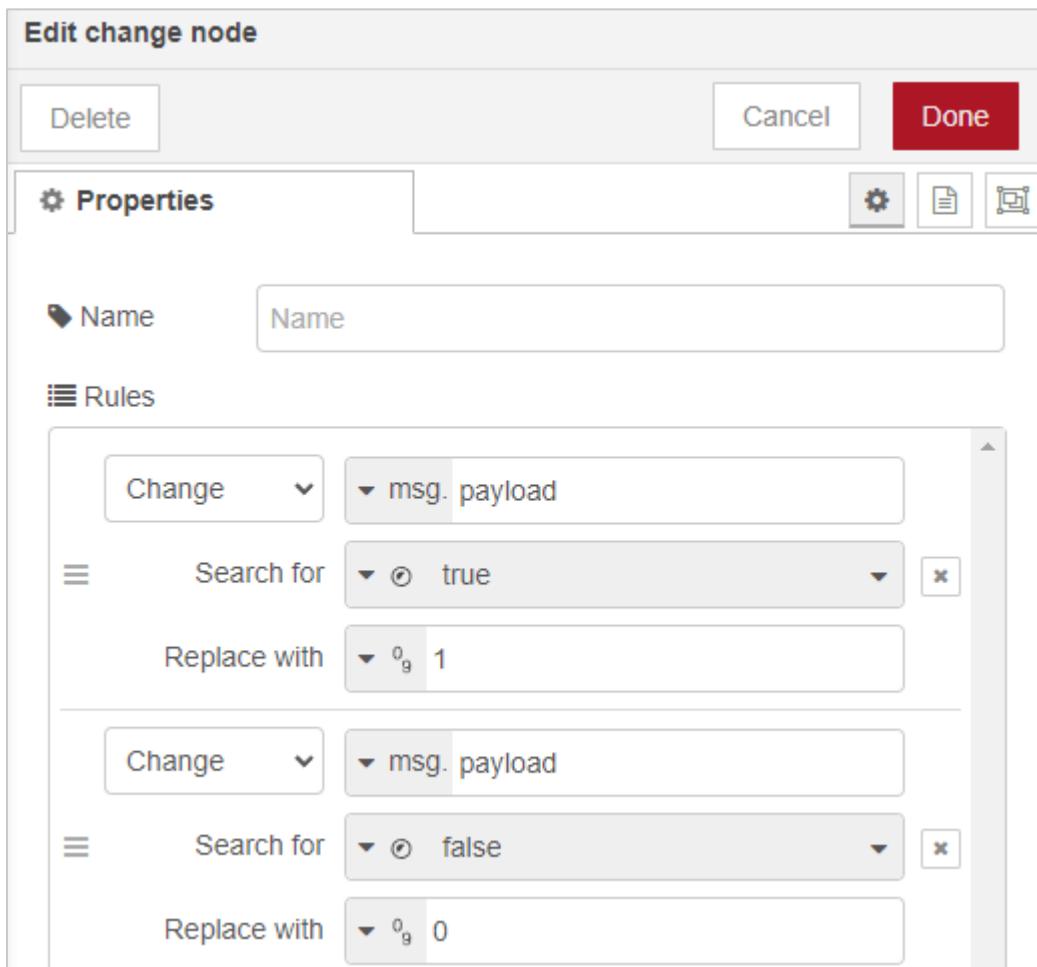
- 1) Add an **MQTT In** node to the flow. Edit its properties to subscribe to the `esp/digital/4` topic. Don't forget to select the MQTT host (`localhost:1883`).



The messages sent via MQTT are `true` or `false`. To use that information in InfluxDB on a chart, for example, it is better to have `1` or `0`. So, we can use a **change** node to do that.



2) Add a **change** node to the flow and wire it to the **MQTT In** node. Double-click to edit its properties. This node is pretty straightforward to use. You select what you want to change. In our case, we want to change the payload of the message `msg.payload`. And we want to replace `true` (boolean) with `1` (number) and `false` (boolean) with `0` (number). After making all the necessary changes click on **Done**.

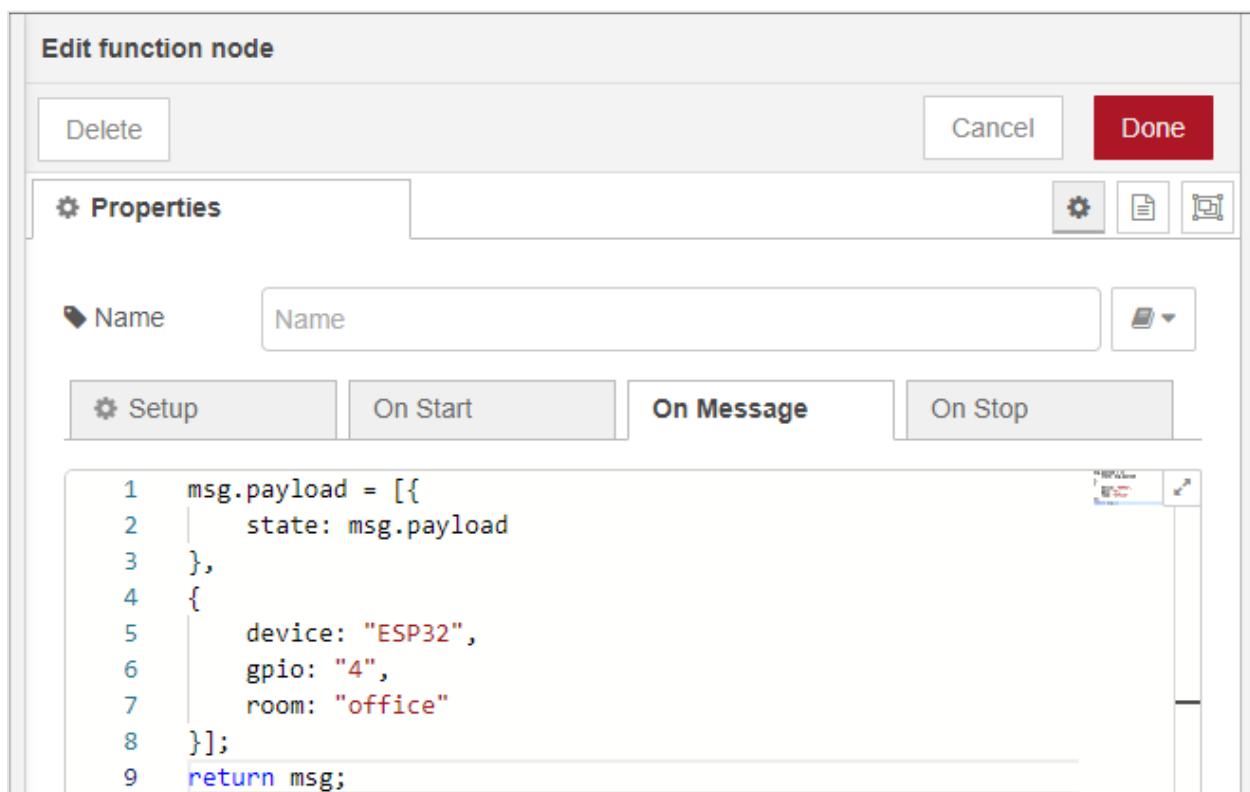


Now, we need to add a function node to prepare the fields and tags to send to InfluxDB (this was explained in the previous unit).

3) Drag a **function** node to the flow and wire it to the **change** node. Double-click to edit its properties. Copy the following to the function on the **On Message** tab.

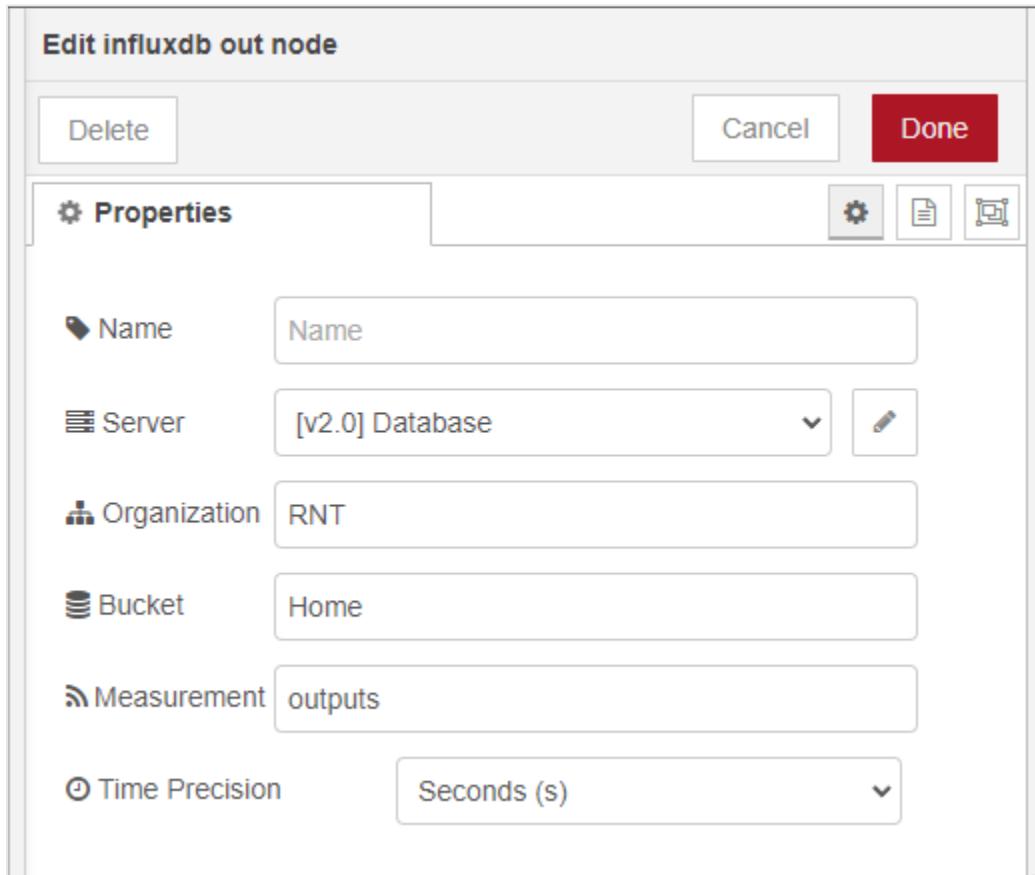
```
msg.payload = [{  
    state: msg.payload  
},  
{  
    device: "ESP32",  
    gpio: "4",  
    room: "office"  
}];  
return msg;
```

We're defining the field key as `state` because the field value is the actual GPIO state. Then, we add several tags `device`, `gpio`, and `room`. You may add more tags or remove them depending on your project, but you must keep the `gpio` tag so that you know which GPIO the state corresponds to. The `gpio` tag value must change depending on the GPIO you're controlling.



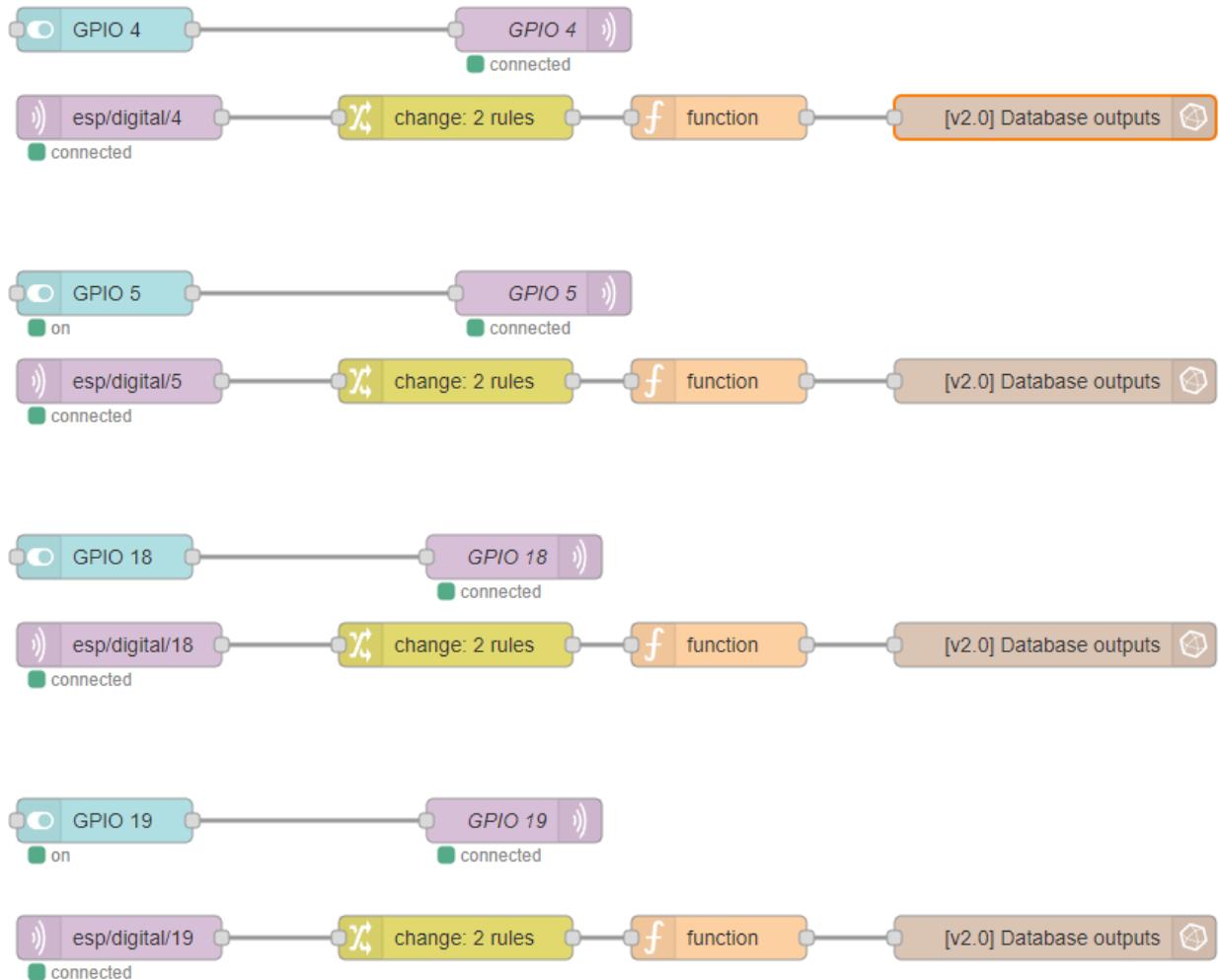
- 4) Add a an **influxdb out** node and wire it to the **function** node. Double-click on it to edit its properties.

We'll publish the GPIOs states on a new measurement called **outputs**. The bucket can be the same as the previous project, **Home**. Don't forget to add your Organization name and InfluxDB server details (they should be saved already if you've followed the previous unit). You can also change the time precision to seconds.



- 5) Repeat the same process but for the other GPIOs. Don't forget to edit the **MQTT In** and **function** nodes with the right GPIO number.

This is out the flow should look like after making all the changes.



The flow is ready. You can deploy your application.



Make sure all MQTT nodes show the **connected** green message after deploying.

Go to Node-RED UI and click on the switch buttons to turn the GPIOs on and off several times so that you have some data on InfluxDB about the GPIO states.

You can download the complete flow at the following link. Don't forget you'll need to change the MQTT broker details and the InfluxDB Server Details:

- [Download Node-RED Flow](#)

Visualizing your Data

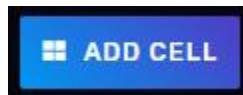
There are many different ways to visualize your data in InfluxDB. We'll show you three different ways to display the GPIO states data: table, chart, and current value.

The best way to have all your preferred visualizations ready is to create a dashboard. You can create a new dashboard or edit the dashboard you already have for the sensor readings.

Go to the **Boards/Dashboards** tab and create a new dashboard or edit the dashboard you already have. We'll use the same dashboard of the previous project.



Click on the **+Add Cell** button to add a new cell with a new visualization.



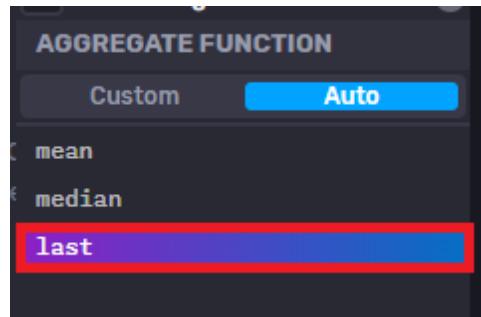
Table

We'll create a table to display the history of states of GPIO 5.

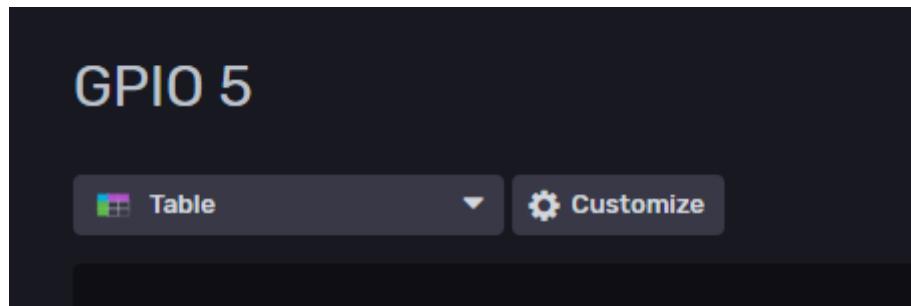
Make a query to get GPIO 5 states. For example:

- **bucket**: Home (bucket name)
- **_measurement**: outputs
- **device**: ESP32 or ESP8266 depending on the tag you used
- **room**: office (or another tag you used)
- **gpio**: 5
- **_field**: state

On the **Aggregate function** window (bottom right corner) select **last**. We don't want it to process mean or median with the GPIO states (it simply doesn't make sense).



On the upper left corner give a name to your cell, for example, `GPIO 5`, and select the **Table** option.



Finally, click on **Submit**. You'll get a similar table.

| _start | _stop | _time | _value | _field | _measu... | device | gpio | room |
|--------------|--------------|--------------|--------|--------|-----------|--------|------|--------|
| 2022-06-2... | 2022-06-2... | 2022-06-2... | 0 | state | outputs | ESP32 | 5 | office |
| 2022-06-2... | 2022-06-2... | 2022-06-2... | 1 | state | outputs | ESP32 | 5 | office |
| 2022-06-2... | 2022-06-2... | 2022-06-2... | 0 | state | outputs | ESP32 | 5 | office |
| 2022-06-2... | 2022-06-2... | 2022-06-2... | 1 | state | outputs | ESP32 | 5 | office |
| 2022-06-2... | 2022-06-2... | 2022-06-2... | 0 | state | outputs | ESP32 | 5 | office |
| 2022-06-2... | 2022-06-2... | 2022-06-2... | 0 | state | outputs | ESP32 | 5 | office |
| 2022-06-2... | 2022-06-2... | 2022-06-2... | 1 | state | outputs | ESP32 | 5 | office |

The previous table displays a lot of information that might not be much useful for our case. We just want to display the timestamp (`_timestamp` column) and the GPIO state (`_value` column). So, we can click on the **Customize** icon to edit our table.

Edit its properties as follows to only show the columns we want. You can also add different headers to the table (Time and State). Additionally, you can also add a different color to the table text or background or add a threshold with different colors (for example, the 0 is white and 1 is green).

The screenshot shows the 'Colorized Thresholds' settings interface. At the top, there is a button to 'Add a Threshold'. Below it, a 'Base' dropdown is set to 'White', and a 'Value is >=' input field contains the value '1'. To the right of the input field is a color picker set to 'Viridian'. A 'Colorization' section has two tabs: 'Background' (disabled) and 'Text' (selected, indicated by a blue background). Below this, the 'Column Settings' section lists eight table columns with their corresponding names: `_start`, `_stop`, `_time`, `_value`, `_field`, `_measurement`, `device`, `gpio`, and `room`.

When the table is looking good, you can click on the green thick icon on the top right corner to add the cell to the dashboard.



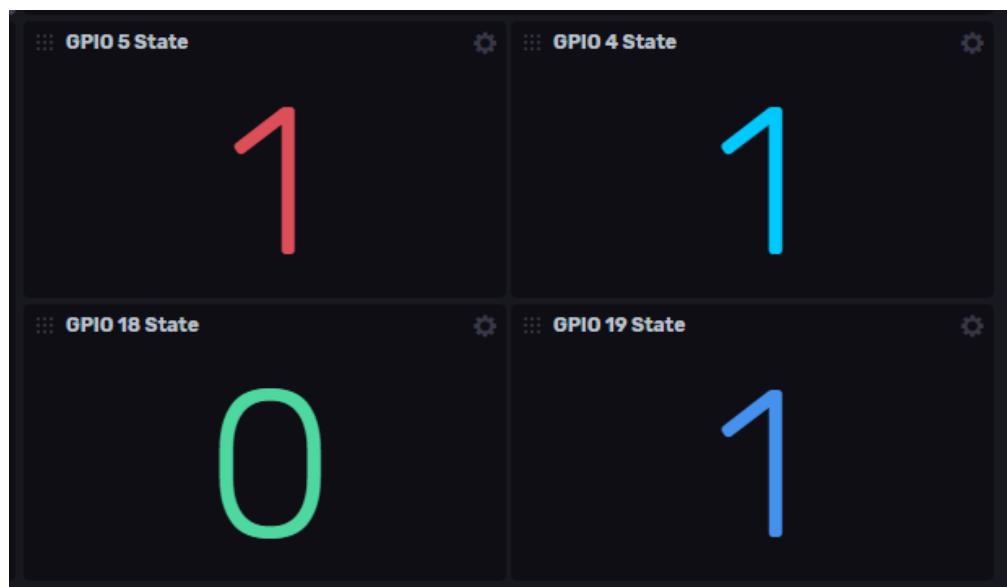
Now, your table should show up on the dashboard.

| Time | State |
|---------------------|-------|
| 2022-06-24 18:00:00 | 0 |
| 2022-06-25 00:24:00 | 1 |
| 2022-06-25 11:40:00 | 0 |
| 2022-06-25 11:44:00 | 1 |
| 2022-06-25 15:56:00 | 0 |
| 2022-06-25 16:00:00 | 0 |

Repeat the same process for the other GPIOs if you want.

Single Stat

It can also be useful to show the current GPIO state on the dashboard. For example, we create a cell for each GPIO to show its current state.



We'll show you how to do that for one GPIO. Then, repeat for the others.

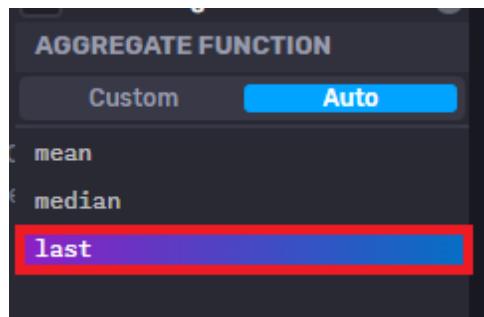
Click on the **+Add Cell** button to add a new cell with a new visualization.



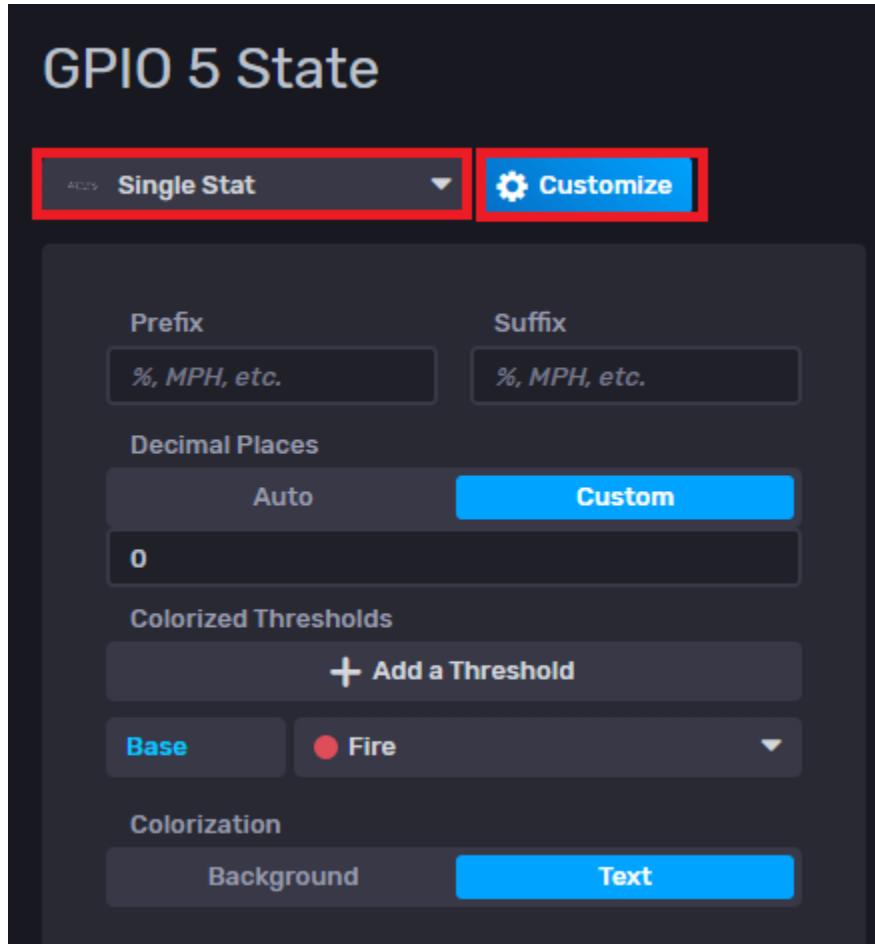
Make a query to get GPIO 5 states. For example:

- **bucket**: Home (bucket name)
- **_measurement**: outputs
- **device**: ESP32 or ESP8266 depending on the tag you used
- **room**: office (or another tag you used)
- **gpio**: 5
- **_field**: state

On the **Aggregate function** window (bottom right corner) select **last**. We don't want it to process mean or median with the GPIO states (it simply doesn't make sense).



On the upper left corner give a name to your cell, for example, `GPIO 5 State`, and select the **Single Stat** option. You can click on the **Customize** option to choose a different color for each GPIO or to add colorized thresholds.



When you're happy with your customization, you can click on the top right corner to add the cell to the dashboard.



Now, GPIO 5 state single stat should show up on the dashboard.



Repeat the same process for the other GPIOs.

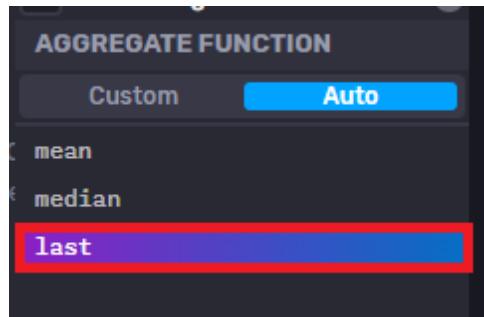
Graph

A graph (chart) is also a good way to visualize the GPIO states throughout time. You can create a chart with all GPIOs states (each GPIO corresponds to a different series) or a different chart for each GPIO.

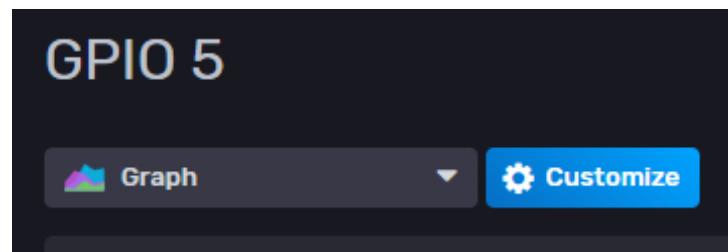
Make a query to get GPIO 5 states. For example:

- **bucket**: Home (bucket name)
- **_measurement**: outputs
- **device**: ESP32 or ESP8266 depending on the tag you used
- **room**: office (or other tag you used)
- **gpio**: 5
- **_field**: state

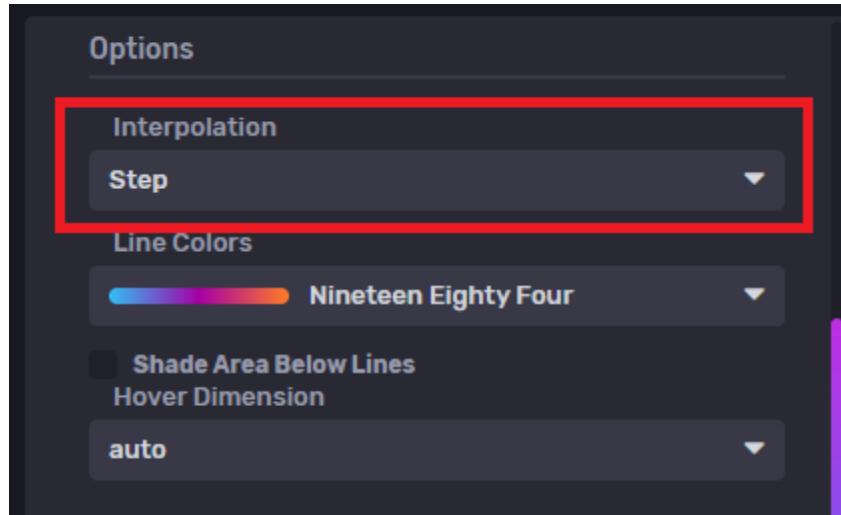
On the **Aggregate function** window (bottom right corner) select **last**. We don't want it to process mean or median with the GPIO states (it simply doesn't make sense).



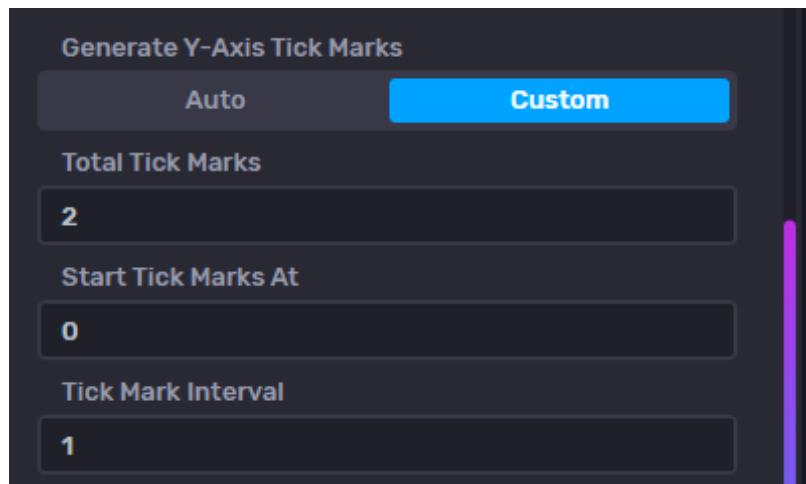
On the upper left corner give a name to your cell, for example, `GPIO 5`, and select the **graph** option.



Click on the **Customize** option to customize the chart. In this case, the state can only be 1 or 0, not any value in between. So, you must set the **Interpolation** option to **Step**.



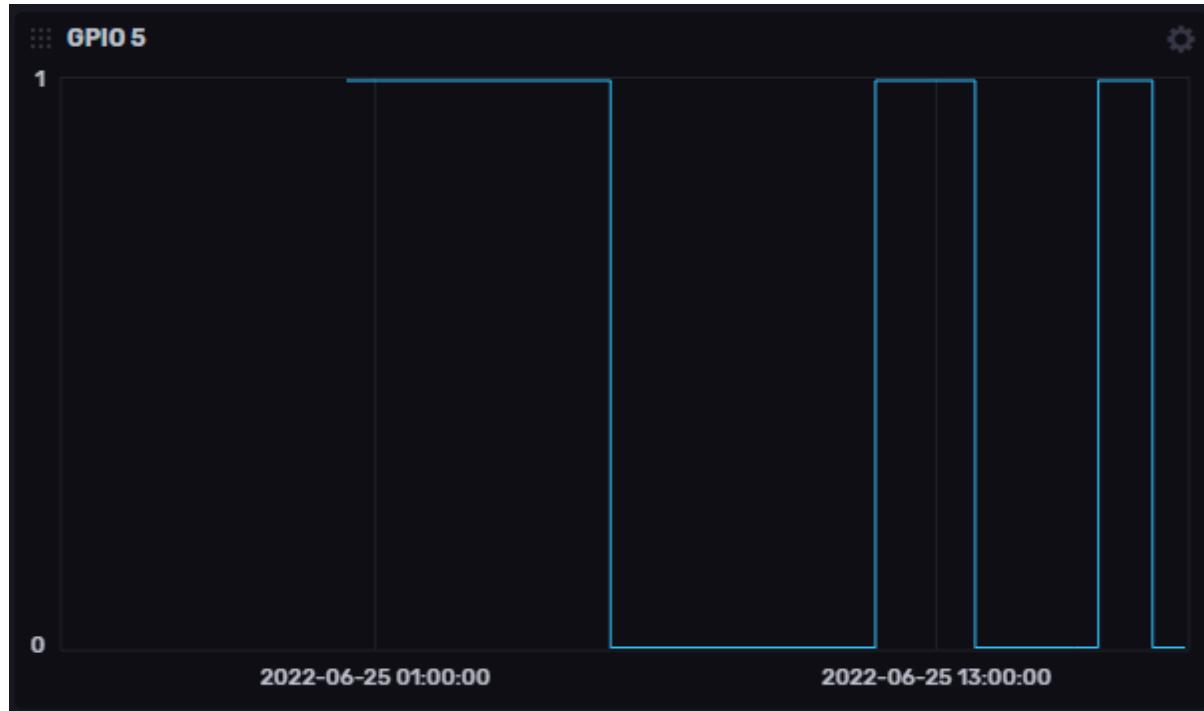
Additionally, the only values that make sense on the Y-axis range are either 0 or 1. So, let's edit the **Y-axis Tick Marks** as follows:



With those options, only 0 and 1 show up on the Y-axis marks. When you're happy with your customization, you can click on the top right corner to add the cell to the dashboard.

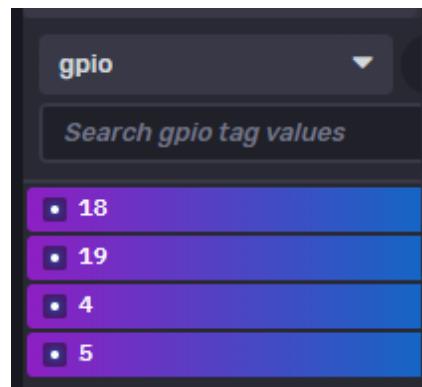


The chart will show up on your dashboard.



You can create charts for the other GPIOs, or edit that chart to plot multiple series.

To create charts with multiple series, select multiple gpio values when making the query.



Here's an example of a chart with multiple GPIOs:



In this scenario, it is a bit difficult to see the different series, but it can make sense if you have two GPIOs that always have opposite states.

Wrapping Up

In this unit, you learned how to save the GPIO states on InfluxDB, so that you can have a history of your GPIO states. That can be useful if the GPIOs are controlling a lamp, a pump, or other appliances and you want to check when they were on and off.

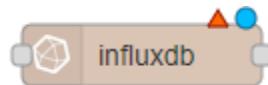
You also learned how to display your data using different visualizations. InfluxDB user interface is very intuitive to use and provides different customization options so that you can build an awesome dashboard to monitor your ESP32 and ESP8266 sensors and GPIOs. Please keep in mind that InfluxDB is a database, so it is not possible to change the GPIO states from there. You need to use Node-RED.

6.6 - Getting Data from InfluxDB

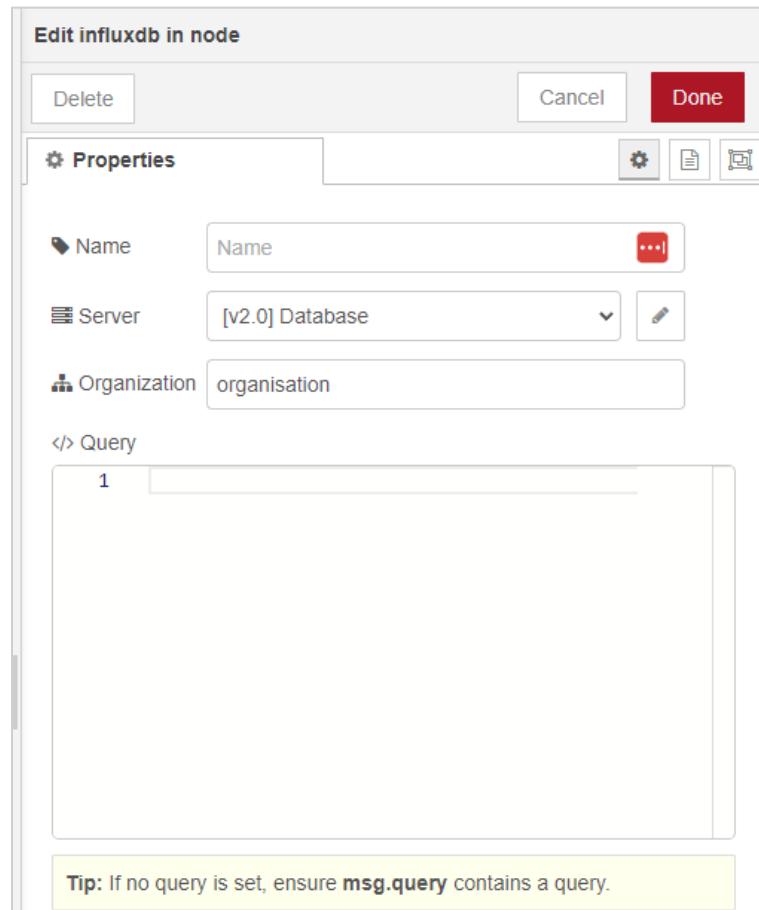
It's also possible to make queries to get InfluxDB data on Node-RED. We won't go into much detail about this subject, but we'll show you a simple example of how to get data from InfluxDB on Node-RED.

InfluxDB In Node

There's a node called **influxdb in** that allows you to get data from InfluxDB. Drag an **influxdb in** node to the flow.



Double-click the node. The following window should open.



The **Name** field is to give a name to the node. In the **Server** field, you should select the InfluxDB database you've configured previously. You should insert your InfluxDB organization name on the **Organization** field.

Finally, you should write a query on the **Query** field to request data.

If you check the node documentation it says the following:

"Allows basic queries to be made to an InfluxDB time series database. The query can be specified in the configuration property or using the property `msg.query`. The results will be returned in `msg.payload`."

Queries should be written in *Flux*.

Flux is InfluxData's functional data scripting language designed for querying, analyzing, and acting on data.

Flux Queries

Learning Flux scripting language is a subject for an entire course. However, you don't need to spend a lot of time learning how to make a simple query.

If you're serious about learning about this subject, we recommend starting on the following tutorials:

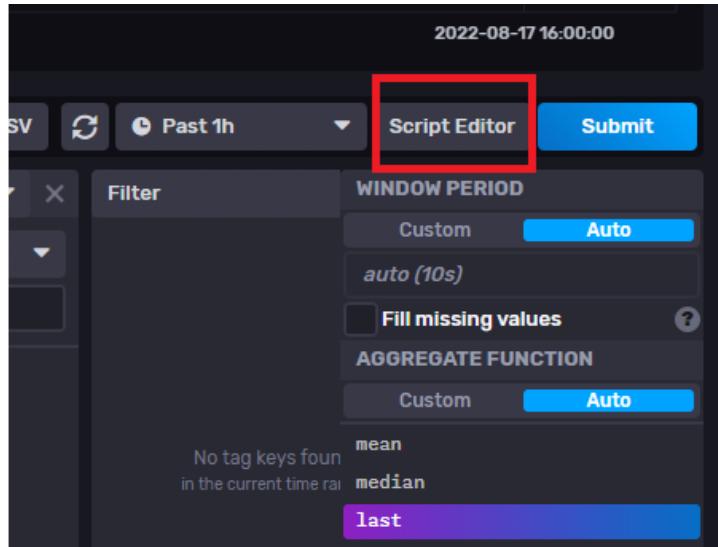
- [Get Started with Flux](#)
- [Flux Query Basics](#)

A Flux query does the following:

- Retrieves a specified amount of data from a source.
- Filters data based on time or column values.
- Processes and shapes data into expected results.
- Returns the result.

For example, go to your InfluxDB dashboard, click on **Explore** and make a query using the menus you learned before. For example, I'm querying GPIO 5 states.

There's a button next to the **Submit** button called **Script Editor**. Click on that button.



It will open flux Script Editor showing the flux query.

```
1 from(bucket: "Home")
2   |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
3   |> filter(fn: (r) => r["_measurement"] == "outputs")
4   |> filter(fn: (r) => r["device"] == "ESP32")
5   |> filter(fn: (r) => r["room"] == "office")
6   |> filter(fn: (r) => r["gpio"] == "5")
7   |> aggregateWindow(every: v.windowPeriod, fn: last, createEmpty: false)
8   |> yield(name: "last")
```

You can use that query as a starting point to write influx queries on Node-RED.

Use `from()` to retrieve data from a data source. In our case, we're storing our data in a bucket called `Home` (yours might be different).

So, our query starts with the following line:

```
from(bucket: "Home")
```

The pipe-forward `|>` operator sends the output of each function to the next function as an input.

The `range()` and `filter()` functions are used to filter data based on column values.

The `range()` is used to select a range of values based on time. In our query, it is using a variable `v` that refers to whatever we select on the InfluxDB dashboard.

```
|> range(start: v.timeRangeStart, stop: v.timeRangeStop)
```

However, we can't pass that line to Node-RED because that variable won't be defined.

We need to pass actual values to the `range()` function.

We can pass the `start` and `stop` parameters to the function, but only the `start` parameter is mandatory.

Values for `start` can be:

- Relative duration, for example: `-1h` (last hour), or `-15m` (last 15 minutes)
- Absolute time, for example: `2022-08-13T22:00:00Z`
- Unix timestamp in seconds, for example: `1660750796`

The `stop` parameter refers to the latest time we want to include in the results. If we don't pass anything, the default value is `now()`.

So, if we want to query data from the last hour, we can add another line to our query that will be as follows:

```
|> range(start: -1h)
```

Learn more about the [range\(\) function](#).

The lines that follow select the data we want taking into account the `measurement` name and the values of the `device`, `room`, and `gpio` tags.

```
|> filter(fn: (r) => r["_measurement"] == "outputs")
|> filter(fn: (r) => r["device"] == "ESP32")
|> filter(fn: (r) => r["room"] == "office")
|> filter(fn: (r) => r["gpio"] == "5")
```

For this example, you don't need the last two lines. So, our query will look as follows:

```
from(bucket: "Home")
|> range(start: -1h)
|> filter(fn: (r) => r["_measurement"] == "outputs")
|> filter(fn: (r) => r["device"] == "ESP32")
|> filter(fn: (r) => r["room"] == "office")
|> filter(fn: (r) => r["gpio"] == "5")
```

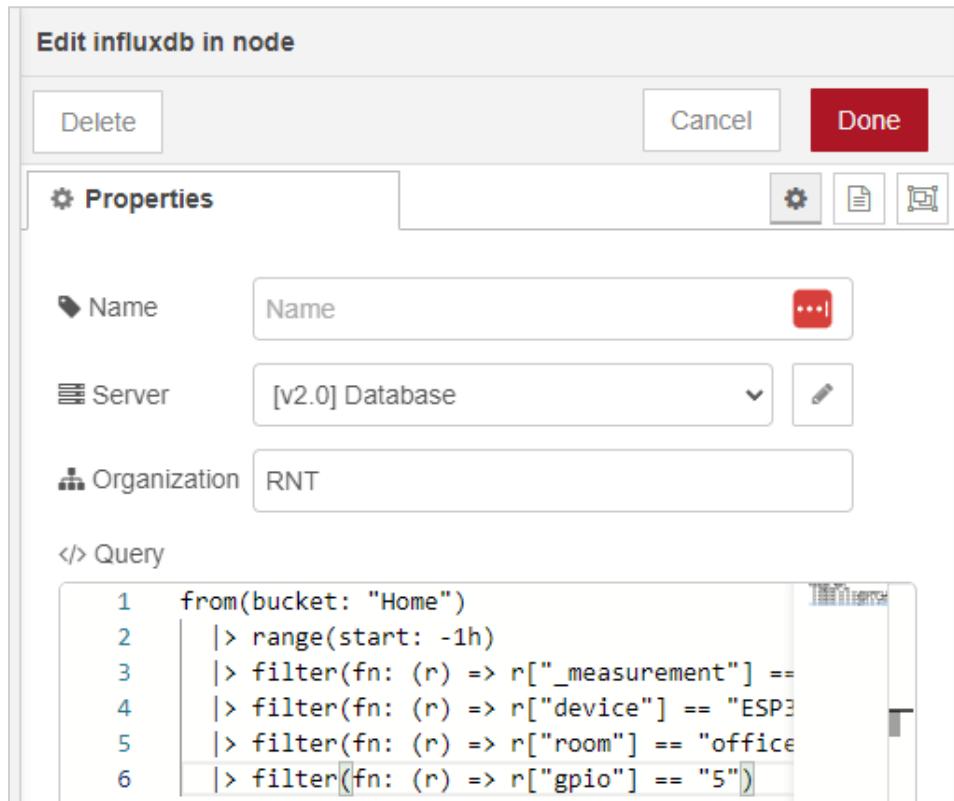
At this point, it is relatively easy to understand how to query data based on a time range, measurement values, and tags. If you need more advanced queries, we recommend exploring Flux documentation.

Creating the Flow

Go back to Node-RED and edit the **influxdb in** node with the query we created:

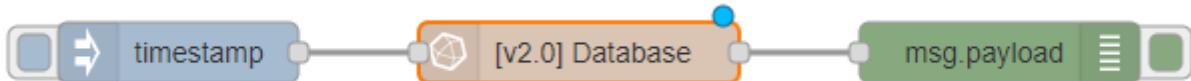
```
from(bucket: "Home")
|> range(start: -1h)
|> filter(fn: (r) => r["_measurement"] == "outputs")
|> filter(fn: (r) => r["device"] == "ESP32")
|> filter(fn: (r) => r["room"] == "office")
|> filter(fn: (r) => r["gpio"] == "5")
```

Don't forget to edit the **Organization** and **Server** fields with your details.



Click on **Done** when you're finished.

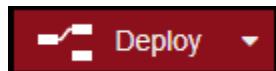
Next, connect an **inject** node to the input and a **debug** node to the output. So, the flow will look as follows:



The **inject** node is simply to trigger the query request. The **debug** node will print the results in `msg.payload`.

Demonstration

Deploy your application.



Open the debugging window (click on the bug icon at the top right corner).

Then, click on the square of the **inject (timestamp)** node to trigger the flow.

You'll receive an array with all the values that match your query.

```
msg.payload : array[9]
  ▶ [ object, object, object, object,
    object, object, object, object,
    object ]
```

In my case, I received an array with nine objects. You need to click on the arrows to expand the results.

The GPIO state value is saved on the `_value` property. You can then convert the output to a JSON object and handle the data you need (not covered in this eBook).

```
msg.payload : array[9]
  ▼ array[9]
    ▼ 0: object
      result: "last"
      table: 0
      _start: "2022-08-
17T14:22:02.254520734Z"
      _stop: "2022-08-
17T15:22:02.254520734Z"
      _time: "2022-08-17T14:25:17Z"
      _value: 0
      _field: "state"
      _measurement: "outputs"
      device: "ESP32"
      gpio: "5"
      room: "office"
    ▼ 1: object
      result: "last"
      table: 0
      _start: "2022-08-
17T14:22:02.254520734Z"
      _stop: "2022-08-
17T15:22:02.254520734Z"
      _time: "2022-08-17T14:26:21Z"
      value: 0
```

Wrapping Up

In this unit, we gave you a quick overview of how to request InfluxDB data on Node-RED using flux queries. You learned how to select data based on the bucket, time range, measurement, and tag values. If you need to make more complex queries, you'll need to learn more about flux, which we won't cover in this eBook. You can [start here](#) and then explore InfluxDB documentation.

6.7 - Deleting InfluxDB Data

In this unit, you'll learn how to delete data from InfluxDB.

Delete an InfluxDB Bucket

To delete a complete bucket, you can use the InfluxDB user interface. Click on **Data/Load Data** (on the left sidebar) and then select the **Buckets** tab.



You'll have a list of all your buckets.

When you hover your mouse over the buckets, a **Delete Bucket** button or a trash bin icon shows up (depending on the InfluxDB version). Click on that button to delete an entire bucket.

A screenshot of the InfluxDB Buckets list. The 'data2' bucket is highlighted with a red box. To its right, there is a 'Delete Bucket' button with a trash bin icon, also highlighted with a red box. Other buckets listed include '_monitoring', '_tasks', 'Home', and 'rasp-pi'. Each bucket has its retention period, ID, and 'Add a label' and 'Add Data' buttons.

Delete Specific Data Points

If you don't want to delete the entire bucket, but only some data points based on conditions, you can use the influx CLI (command line interface).

Influx CLI

Influx CLI is InfluxDB command line interface that contains commands to manage many aspects of InfluxDB, including buckets, organizations, users, tasks, etc. It should be automatically installed when you install InfluxDB.

Provide required authentication credentials

You can skip this section if you've followed Unit 6.3.

To avoid having to pass your InfluxDB host, API token, and organization with each command, you can store them in an influx CLI configuration (config). For that, you need to use the `influx config create` command and pass your information.

First, edit the following command with your information:

```
influx config create --config-name influx-config --host-url  
http://YOUR_RASPBERRY_PI_IP_ADDRESS:8086 --org <your-org> --  
token <your-auth-token> --active
```

Replace `YOUR_RASPBERRY_PI_IP_ADDRESS` with your Pi's IP address.

Replace `<your-org>` with your InfluxDB organization name

Replace the `<your-auth-token>` with the API token you got in the previous step.

For example, in my case, the command looks as follows:

```
influx config create --config-name influx-config --host-url  
http://192.168.1.106:8086 --org RNT --token v_od_mG--  
9_srf_OnaaODihPDX34suToP7XEH47v6x77CMxakZaoYHzF7Ec9mLT-  
CuXXXXXXXXXXXXXXXXXXXXvQCSSw== --active
```

Delete Data

We'll show you a few examples of how to delete data from InfluxDB. For more detailed information about this subject, we recommend taking a look at the following references:

- [Delete data using the influx CLI](#)
- [influx delete](#)

You can delete data from InfluxDB using InfluxDB commands on the Raspberry Pi terminal window. Open an SSH connection with your Pi if you haven't already.

Delete a Bucket

Besides deleting the bucket in the user interface as we've shown you previously, you can also delete a bucket using the influx CLI.

Use the `influx bucket delete` command and specify the bucket name and organization:

- `--n` to specify the bucket name
- `--o` to specify the organization

Here's the command to delete an entire bucket:

```
influx bucket delete -n <bucket-name> -o <org-name>
```

Replace `<bucket-name>` with the name of the bucket you want to delete and `<org-name>` with your organization name.

For example, the following command deletes the bucket with the name `data`, and my organization name is `RNT`.

```
influx bucket delete -n data -o RNT
```

Delete Specific Data Points

To delete data points, you use the `influx delete` command.

You use flags to specify which points you want to delete. You can check the [list of all flags here](#), but here's a list of the most relevant.

- `--bucket` specifies which bucket to delete data from.
- `--start` and `--stop` flags define the time range to delete data from—use date in [RFC3339 format](#) (for example `2020-01-01T00:00:00.00Z`).
- `--p`, predicative flag to include a delete predicate that identifies which point to delete (we'll show you a few examples later).

The `--bucket`, `--start`, and `--stop` flags are required.

Delete all points in a specified time range

The following command is an example of how to delete all data points in a bucket in a specified time range.

```
influx delete --bucket bucket-example-name \
  --start 2022-03-01T00:00:00Z \
  --stop 2022-08-14T00:00:00Z
```

Predicate Syntax

If you want to delete points by field, measurement or tags values, you need to use the predicate syntax.

"InfluxDB uses the *delete predicate* to evaluate the series keys of points in the time range specified in the delete request. Points with series keys that evaluate to true for the given predicate are deleted. Points with series keys that evaluate to false are preserved. A delete predicate is comprised of one or more predicate expressions. The left operand of the predicate expression is the column name. The right operand

is the column value. Operands are compared using comparison operators. Use logical operators to combine two or more predicate expressions." ([source](#))

Delete points by measurement

For example, to delete data points in the `outputs` measurement, you use the following predicate syntax.

```
_measurement="outputs"
```

So, the command to delete data between a predefined time range from the `outputs` measurement from the bucket `Home` would look as follows:

```
influx delete --bucket Home \
--start 2022-03-01T00:00:00Z \
--stop 2022-08-14T00:00:00Z \
--p '_measurement="outputs"'
```

Delete points by field

Similarly, you can delete data points by field. For example:

```
_field="temperature"
```

Then, the complete command:

```
influx delete --bucket Home \
--start 2022-03-01T00:00:00Z \
--stop 2022-08-14T00:00:00Z \
--p '_field="temperature"'
```

Delete points by tags

Finally, you can also delete data points by tag. Here's a predicative example:

```
board="ESP32" AND room="office"
```

An example of the complete command:

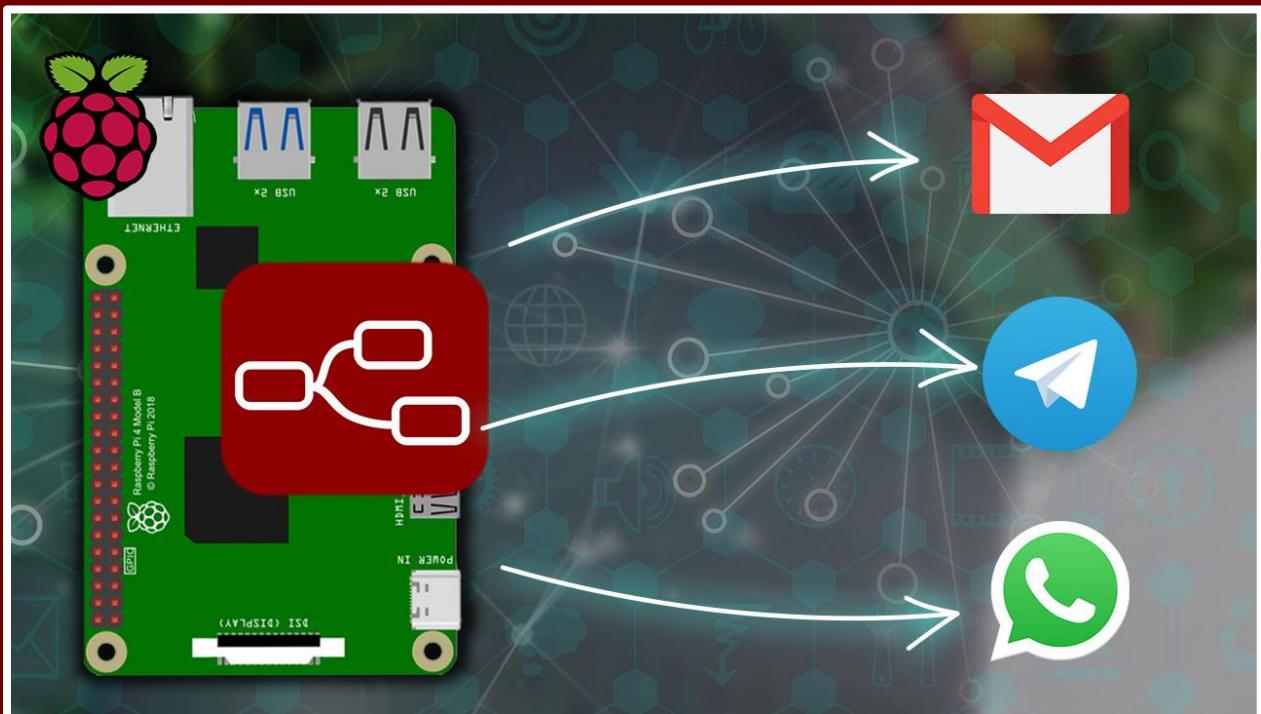
```
influx delete --bucket Home \
  --start 2022-03-01T00:00:00Z \
  --stop 2022-08-14T00:00:00Z \
  --p 'board="ESP32" AND room="office"'
```

Wrapping Up

This unit was a quick guide showing you different ways to delete data from InfluxDB. You learned how to delete complete buckets and how to delete specific data points. For more information about deleting data using InfluxDB, we recommend taking a look at the [documentation](#).

MODULE 7

Sending Notifications with Node-RED



In this Module, you'll learn how to send notifications with Node-RED. We'll show you how to send emails, Telegram messages, and WhatsApp messages. We'll also create a simple project to send the notification of your choice whenever motion is detected.

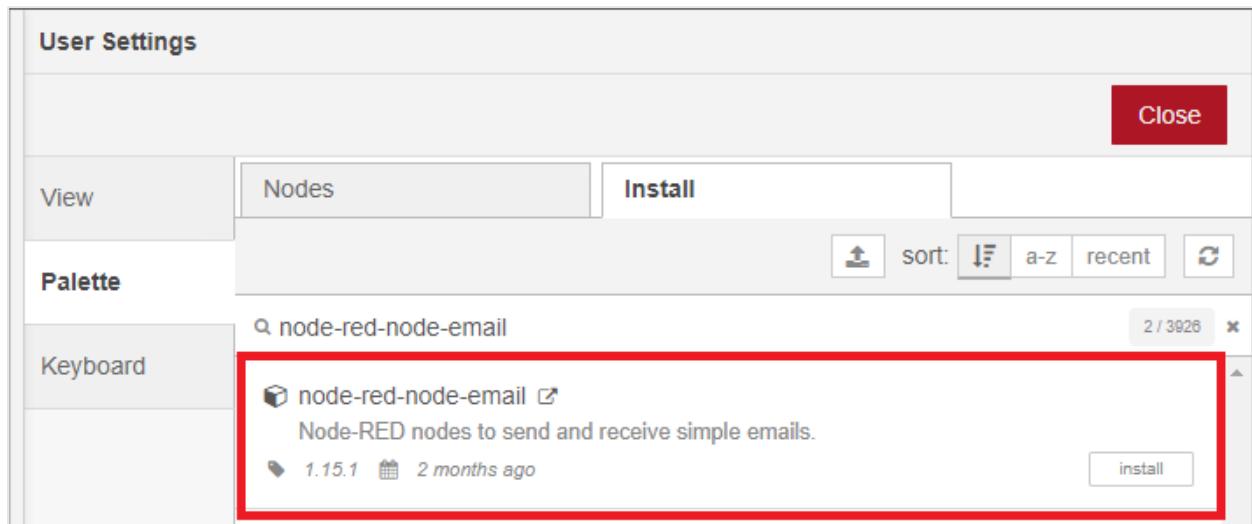
7.1 - Email Alerts with Node-RED

In this unit, you'll learn how to send emails with Node-RED. This can be useful to send notifications to your email if a sensor reading is above or below a certain threshold, to send readings periodically to your email, or to send a notification when motion is detected, for example.

Install Email Nodes

There are Node-RED Nodes you can install that make it easy to send emails using Node-RED.

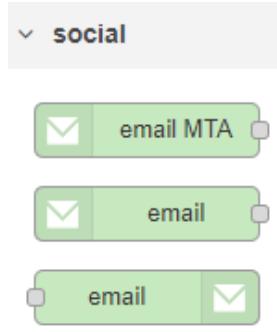
Go to **Menu > Manage Palette > Install**. Search for `node-red-node-email` and install those nodes.



For more information about the nodes, you can check the following link:

- <https://flows.nodered.org/node/node-red-node-email>

Now, you should have the email nodes on your palette.



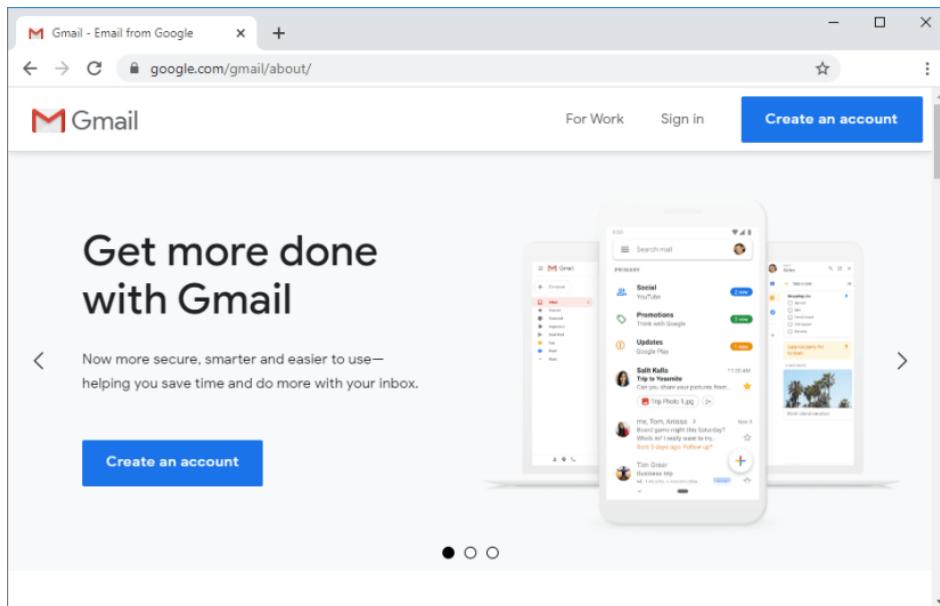
Sender Email

To send emails with Node-RED you need a sender email account. We don't recommend using your main email account to send emails using Node-RED. If something goes wrong in your flow or if by mistake you make too many requests, you can be banned or have your account temporarily disabled. So, we recommend creating a new email account to send emails or using a secondary email account.

We'll use a Gmail.com account to send the emails, but you can use any other email provider. The receiver email can be your personal email without any problem.

Create a Sender Email Account

To create a new Gmail account for sending emails with Node-RED, [go to this link](#).



Create an App Password

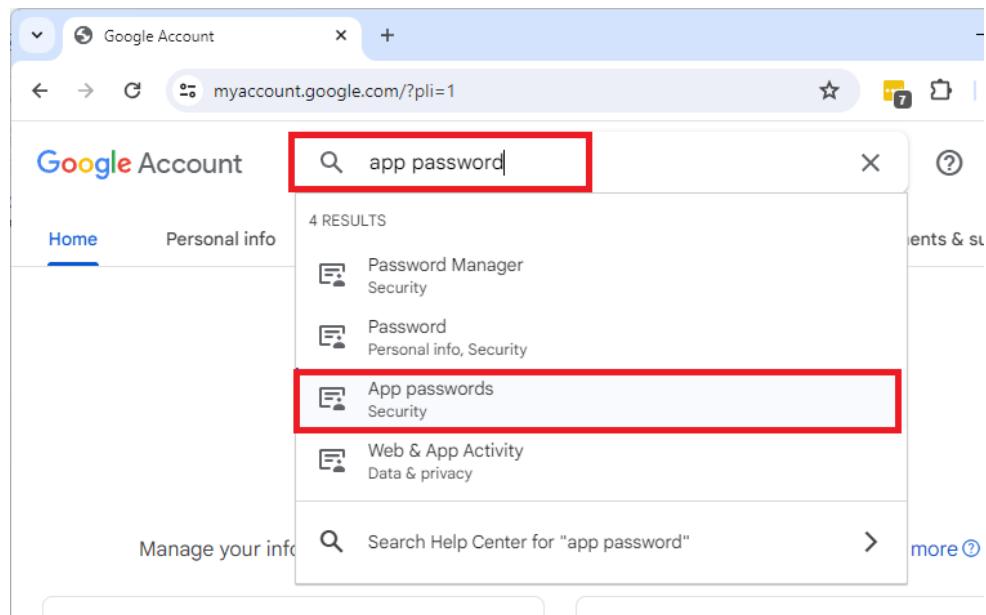
You need to create an app password so that Node-RED can send emails using your Gmail account. An App Password is a 16-digit passcode that gives a less secure app or device permission to access your Google Account. [Learn more about sign-in with app passwords here.](#)

An app password can only be used with accounts that have [2-step verification turned on.](#)

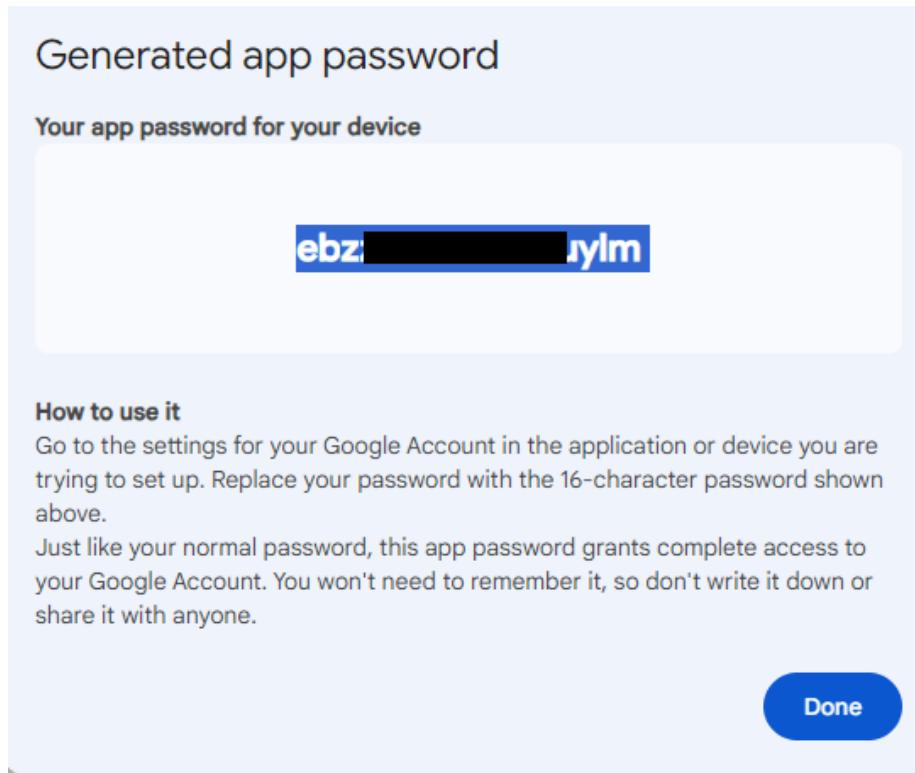
- 1) Open your [Google Account](#).
- 2) In the navigation panel, select **Security**.
- 3) Under "Signing in to Google", select **2-Step Verification > Get started**.
- 4) Follow the on-screen steps.

After enabling 2-step verification, you can create an app password.

- 1) Open your [Google Account](#).
- 2) In the search bar, search for App Password.
- 3) Click on the App Passwords menu.



- 4) Now, you can create a new app password to use with the Raspberry Pi. Give it a name, for example *Node-RED*.
- 5) Click **Create**. Your app password will show up. Copy it to a safe place because you'll need it later (even though it says you don't need to).



Now, you should have an app password that you'll use on Node-RED to send the emails.

| Your app passwords | | |
|--------------------|---------|-----------|
| Name | Created | Last used |
| Node-RED | 11:37 | - |

If you're using a different email provider, check what you need to do to allow less secure apps. You should be able to find the instructions with a quick google search "**your email provider name + app password**".

Gmail SMTP Server Settings

If you're using a Gmail account, these are the SMTP Server details:

- SMTP Server: smtp.gmail.com
- SMTP username: Complete Gmail address
- SMTP password: Your Gmail password
- SMTP port (TLS): 587
- SMTP port (SSL): 465
- SMTP TLS/SSL required: yes

Outlook SMTP Server Settings

For Outlook accounts, these are the SMTP Server settings:

- SMTP Server: smtp.office365.com
- SMTP Username: Complete Outlook email address
- SMTP Password: Your Outlook password
- SMTP Port: 587
- SMTP TLS/SSL Required: Yes

Live or Hotmail SMTP Server Settings

For Live or Hotmail accounts, these are the SMTP Server settings:

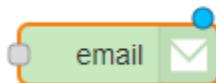
- SMTP Server: smtp.live.com
- SMTP Username: Complete Live/Hotmail email address
- SMTP Password: Your Windows Live Hotmail password

- SMTP Port: 587
- SMTP TLS/SSL Required: Yes

If you're using another email provider, you need to search for its SMTP Server settings. Now, you have everything ready to start sending emails with Node-RED.

Sending an Email with Node-RED

Drag an **email** node to the flow.



Drag an **inject** node to the flow and wire it to the email. The **inject** node will contain the email content.



To learn how we must configure the message for the **email** node, we can take a look at the email node documentation. Select the **email** node and then, click on the **help** icon in the top right corner.



You'll get information about how to use the node.

email

Sends the `msg.payload` as an email, with a subject of `msg.topic`.

The default message recipient can be configured in the node, if it is left blank it should be set using the `msg.to` property of the incoming message. If left blank you can also specify any or all of: `msg.cc`, `msg.bcc`, `msg.replyTo`, `msg.inReplyTo`, `msg.references`, `msg.headers`, or `msg.priority` properties.

You may optionally set `msg.from` in the payload which will override the `userid` default value.

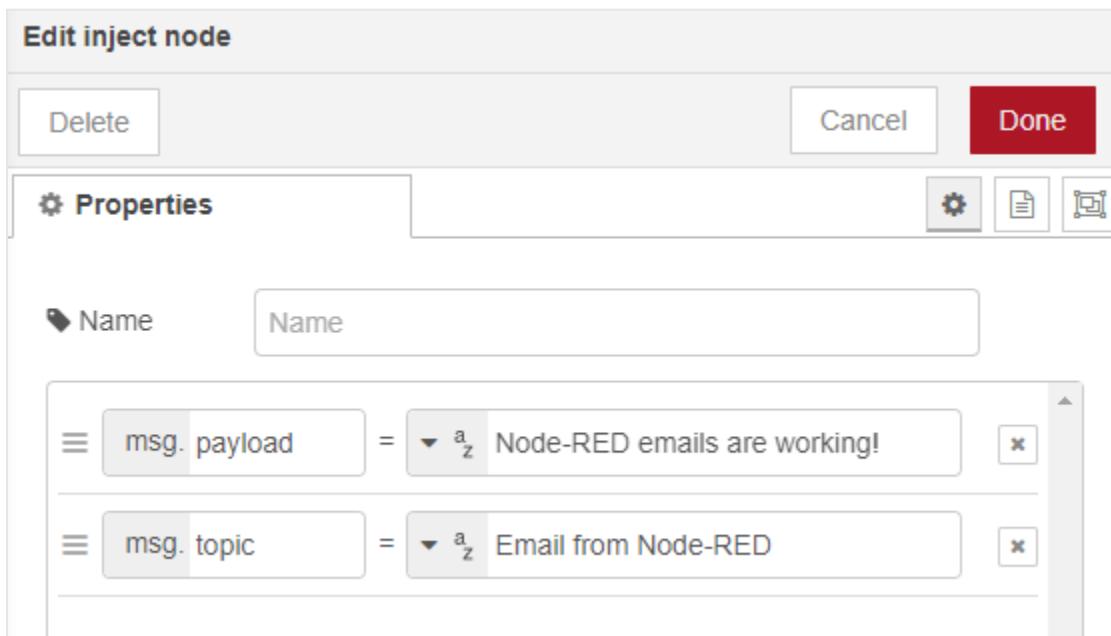
▼ Gmail users

If you are accessing Gmail you may need to either enable [an application password](#), or enable [less secure access](#) via your Google account settings.

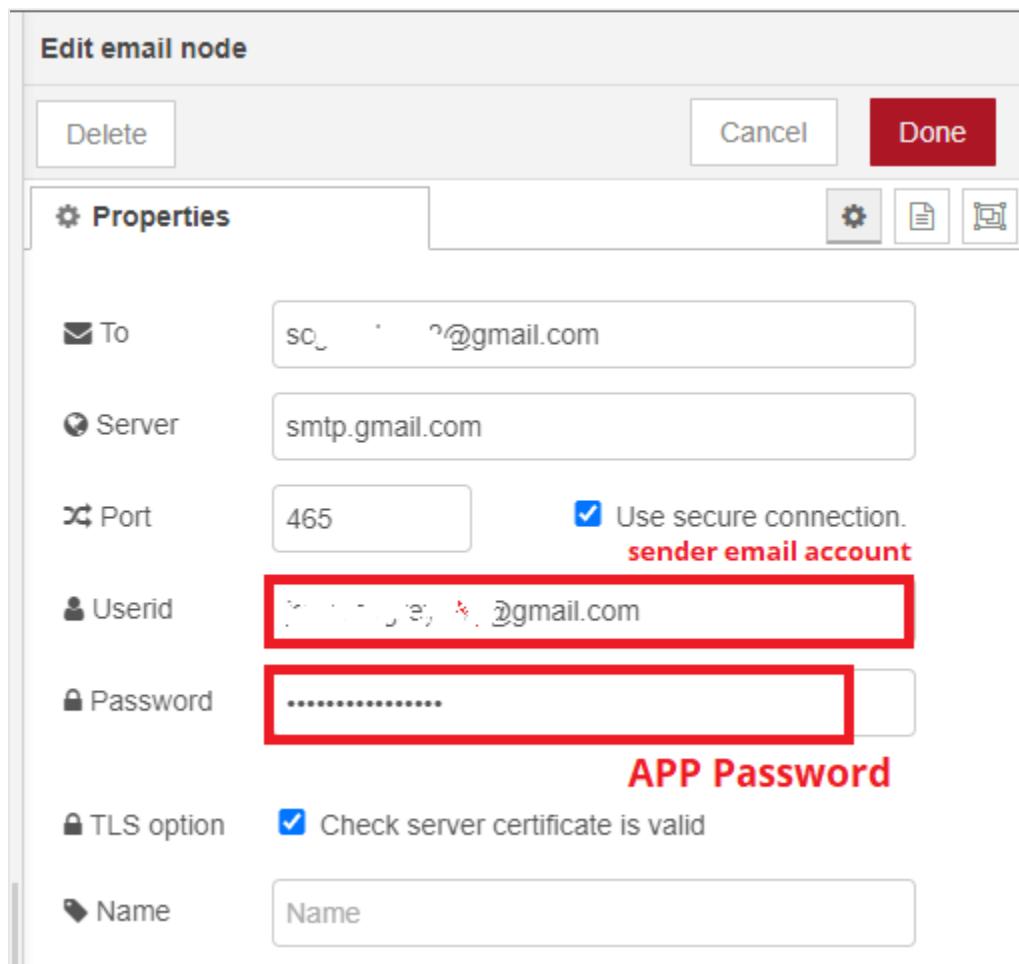
So, we must set the message content on `msg.payload` and the email subject on `msg.topic`. The recipient can be set in `msg.to` or in the email node (we'll set it in the email node).

Double-click the **inject** node to edit its properties.

Add the email content on `msg.payload`. You need to set it as a String. Then, add the email subject on the `msg.topic`. It also must be set as a String.



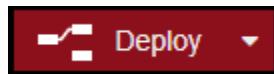
Now, double-click the email node to edit its properties.



When you're finished, click **Done**.

Testing the Flow

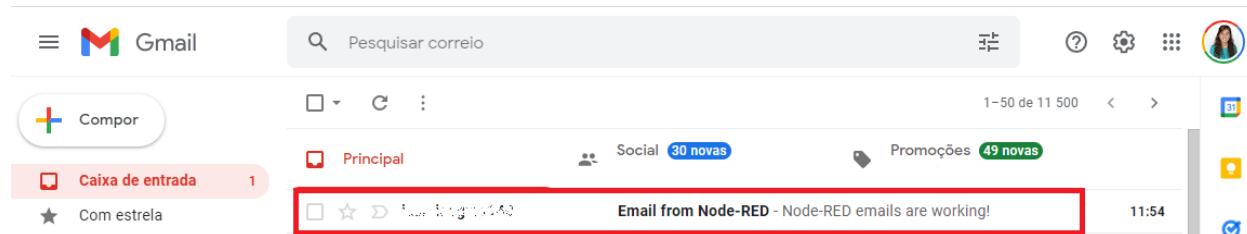
Now, deploy your application.



After deploying click on the left square on the **inject** node to trigger the **email** node.



If everything went as expected, you should have a new email on your recipient account.



And that's it!

Wrapping Up

As you can see, sending emails with Node-RED is very simple. In this unit, we covered the basics of how to send an email. The idea is to apply this feature in real-world scenarios. In upcoming units, you'll learn how to send emails when a certain condition is met (temperature above or below a threshold and when motion is detected).

7.2 - Telegram Messages with Node-RED

This unit is similar to the previous one but shows how to send messages to Telegram so that you can be notified on your smartphone. As long as you have access to the internet on your smartphone, you'll be notified no matter where you are.

Introducing Telegram

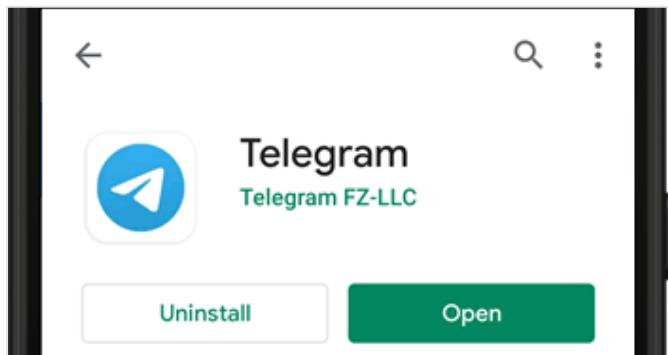
[Telegram](#) Messenger is a cloud-based instant messaging and voice over IP service. You can easily install it on your smartphone (Android and iPhone) or computer (PC, Mac, and Linux). It is free and without any ads. Telegram allows you to create bots that you can interact with.

"Bots are third-party applications that run inside Telegram. Users can interact with bots by sending them messages, commands and inline requests. You control your bots using HTTPS requests to Telegram Bot API".

Node-RED will interact with the Telegram bot to send messages to your Telegram account.

Creating a Telegram Bot

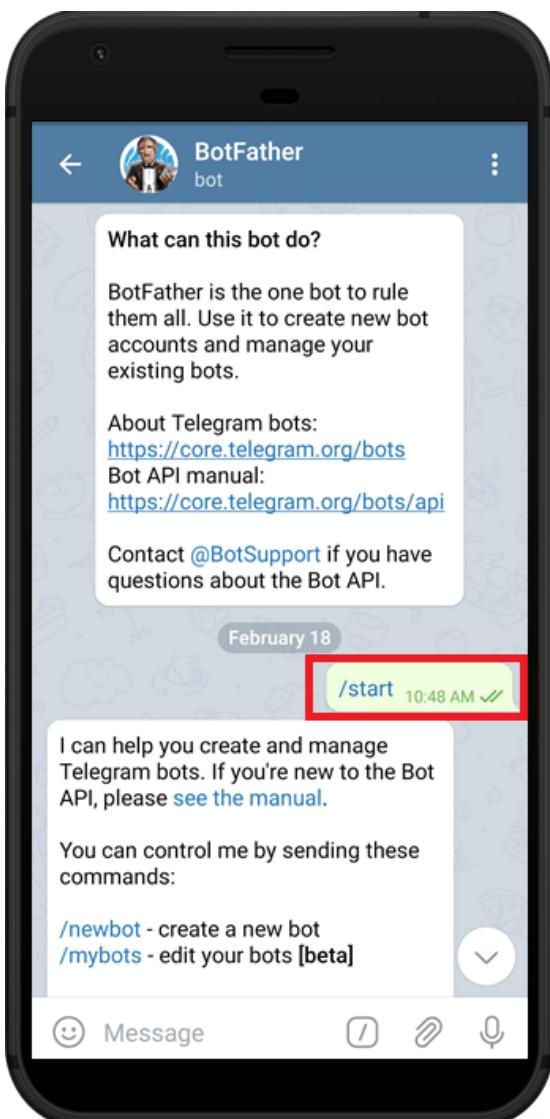
Go to Google Play or App Store, download, and install Telegram.



Open Telegram and follow the next steps to create a Telegram Bot. First, search for "botfather" and click the **BotFather** as shown below. Or open this link t.me/botfather on your smartphone.



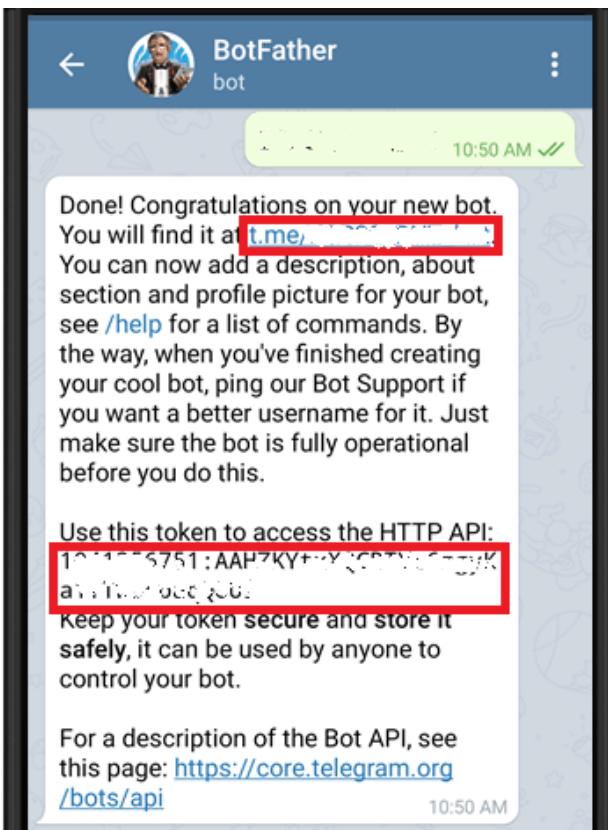
The following window should open and you'll be prompted to click the start button.



Type `/newbot` and follow the instructions to create your bot. Give it a name and username.



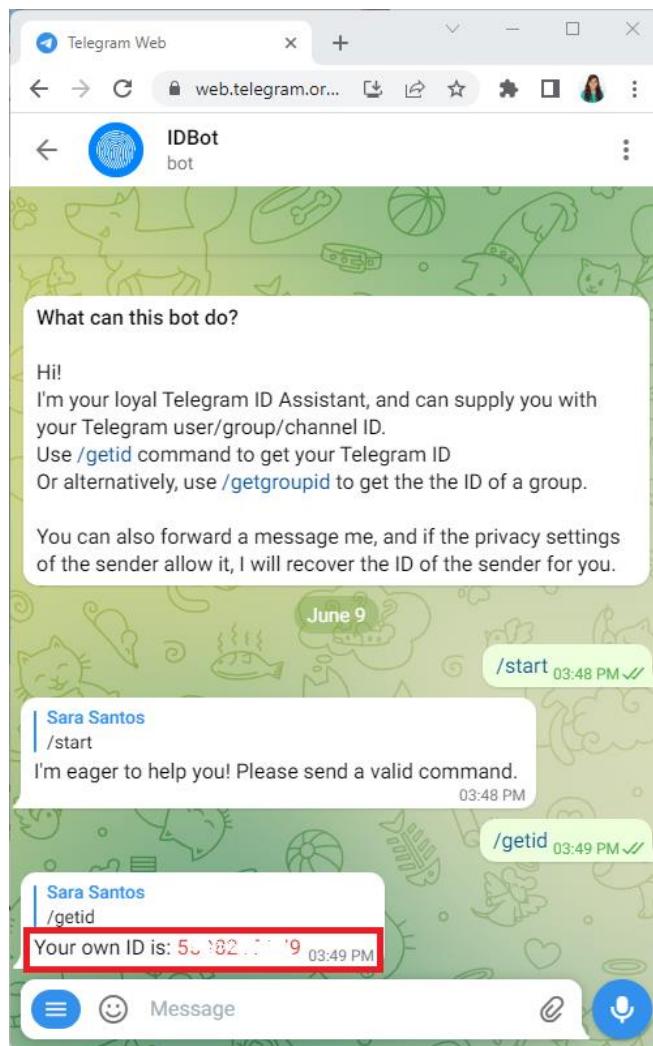
If your bot is successfully created, you'll receive a message with a link to access the bot and the bot token. Save the bot token because you'll need it so that Node-RED can interact with the bot.



To make it easy to copy the bot token to Node-RED, you can open the Telegram web app, copy the token from there and save it in a Notepad, for example. To open the Telegram web app on your computer, open the following link: <https://web.telegram.org/>

Get Your Telegram User ID

To set up the Node-RED nodes to send messages to your Telegram account, you need to get your Telegram User ID. In your Telegram account, search for "@myidbot" or open this link t.me/myidbot on your smartphone. Start a conversation with that bot and type /getid. You will get a reply back with your user ID. Save that user ID, because you'll need it later in this tutorial.



Start a Chat with Your Bot

This step is very important. Don't miss it. Otherwise, the project will not work.

You must send a message to your Telegram Bot from your Telegram account before it can send you messages.

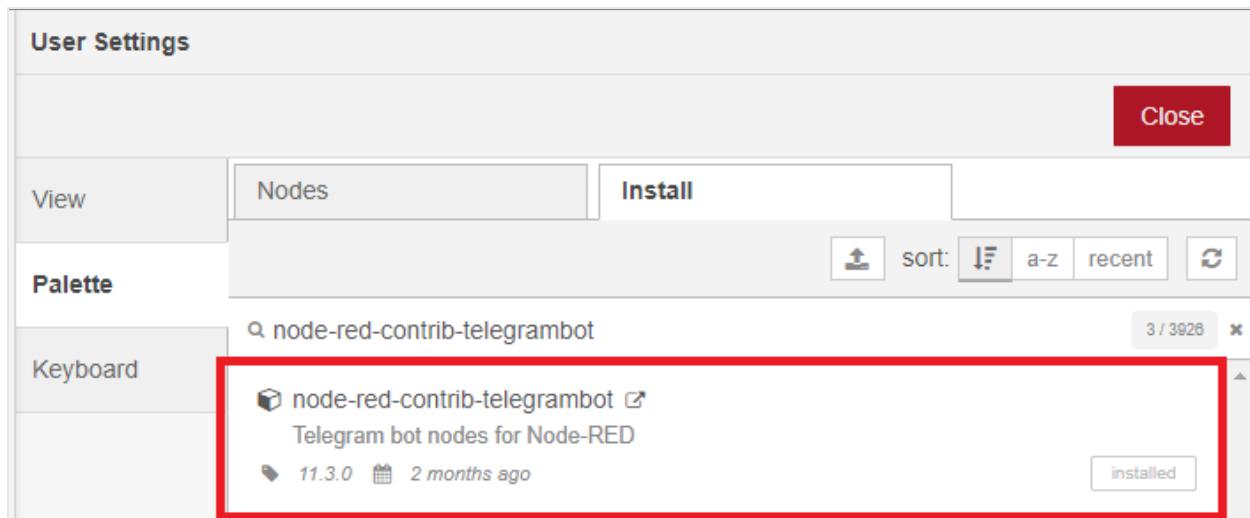
- 1) Go back to the chats tab, and in the search field, type the username of your bot.
- 2) Select your bot to start a conversation.
- 3) Click on the *Start* link that should appear at the bottom of the screen.

And that's it! You can proceed to the next section.

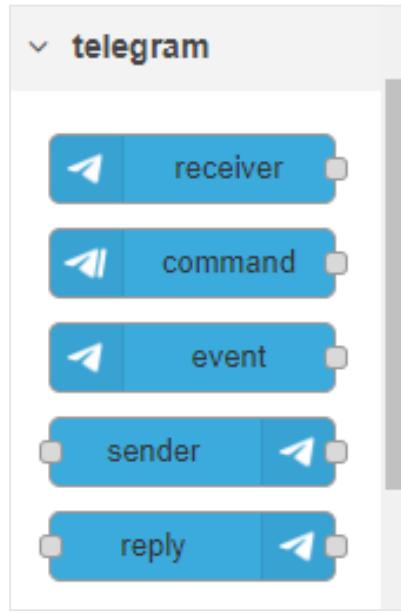
Installing Telegram Nodes

There are nodes you can install in Node-RED that make it easy to send messages using Telegram.

Go to **Menu > Manage Palette > Install** and search for `node-red-contrib-telegrambot`.

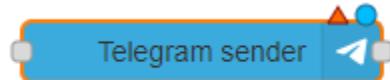


After installing, you should have new nodes on your palette.



Sending a Message to Telegram with Node-RED

Drag a **sender** node to the flow.



Double-click on the node to edit its properties. Click on the pencil icon to add your Telegram bot details.

On the **Bot-Name** field enter the username of the bot you created in the previous steps. Then, enter the corresponding token. You also need to add your chat ID to the **ChatIds** field.

Edit sender node > **Add new telegram bot config node**

Cancel **Add**

Properties

| | |
|----------|---|
| Bot-Name | Nc_127.0.0.1:5000 |
| Token | 5480521310:AAE3UJzjPfVqy4pFjD9SCUMLM2Cnq... [REDACTED] |

Tip: If you don't have a token yet, you can create a new one here: [@BotFather](#).

| | |
|---------|---|
| Users | (Optional list of authorized user names e.g.: hugo,sepp,egon) |
| Chatids | 5379612345 [REDACTED] |

| | |
|------------|--|
| Server URL | (Optional URL for proxying and testing e.g.: https://api.telegram.org) |
|------------|--|

| | |
|-------------|---------|
| Update Mode | Polling |
|-------------|---------|

When you're finished, click on **Add**. Then, click **Done** on the sender node.

Now, we need another node that will send the message content and properties to the node.



If you check the Telegram sender node documentation, it mentions:

"The `msg.payload` must be an object that contains a complete set of Telegram message properties, at a minimum these should contain:

- `content`: the message contents
- `type`: the type of message contents

- `chatId`: the chatId number or an array of chatIds “

Double-click on the **inject** node to edit its properties. Select JSON and click on the three dots to open a window to write JSON. Copy the following:

```
{
  "chatId": "XXXXXXXXXX",
  "type": "message",
  "content": "Node-RED is working"
}
```

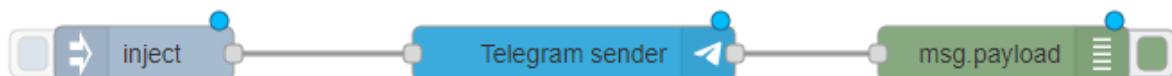
Replace the XXXXXXXXXX with your chat ID.



We've set the message to `Node-RED is working`, but you can change the message content.

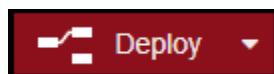
Then, click **Done**, and **Done** again to close the **inject** node.

Finally, drag a **debug** node and wire it to the **Telegram sender** node to check the output result.

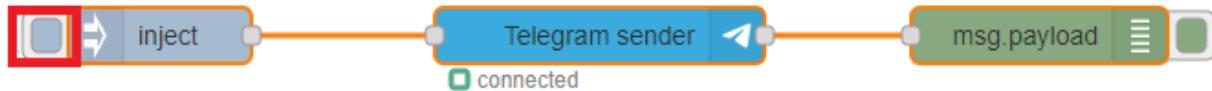


Testing the Flow

Deploy your application.



After deploying click on the left square on the **inject** node to trigger the **Telegram sender** node.



You should receive a message in your Telegram account from your bot with the content you've set on the node:



Congratulations! Now, you know how to send messages to Telegram using Node-RED. You can apply what you learned to your home automation system to send useful notification messages.

7.3 - WhatsApp Messages with Node-RED

Another option to receive notifications is using WhatsApp. In this Unit, you'll learn how to send messages to your WhatsApp account using Node-RED.

Introducing WhatsApp



"WhatsApp Messenger, or simply WhatsApp, is an internationally available American freeware, cross-platform centralized instant messaging and voice-over-IP service owned by Meta Platforms." It allows you to send messages using your phone's internet connection, so you can avoid SMS fees.

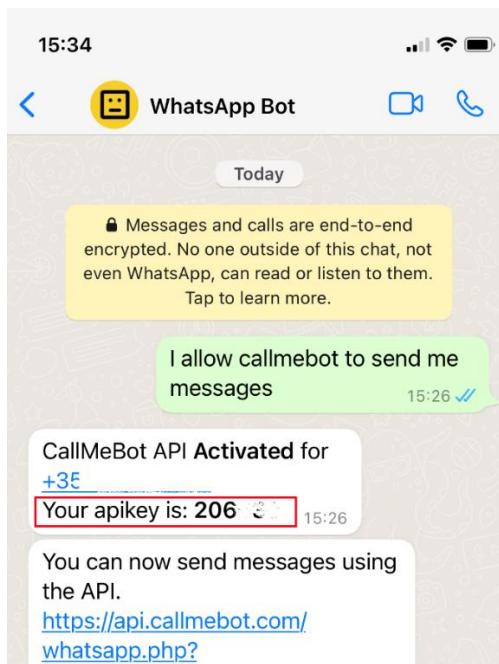
WhatsApp is free and is available for Android and iOS. Install WhatsApp on your smartphone if you don't have it already.

CallMeBot WhatsApp API

To send messages to your WhatsApp account using Node-Red, we'll use a Node-RED node that uses the callmebot service. You can learn more about callmebot on the following link: <https://www.callmebot.com/>. Basically, it works as a gateway that allows you to send a message to yourself, which is what you need for a notification system in your home automation setup.

Before starting using the node, you need to get the CallmeBot WhatsApp API key. Follow the next instructions ([check this link for the instructions on the official website](#)).

- 1) Add the phone number +34 621 062 163 to your Phone Contacts. (Name it as you wish)²;
- 2) Send this message "I allow callmebot to send me messages" to the new Contact created (using WhatsApp of course);
- 3) Wait until you receive the message "API Activated for your phone number. Your APIKEY is XXXXXX" from the bot.



Note: If you don't receive the API key in 2 minutes, please try again after 24hours or double-check the number.

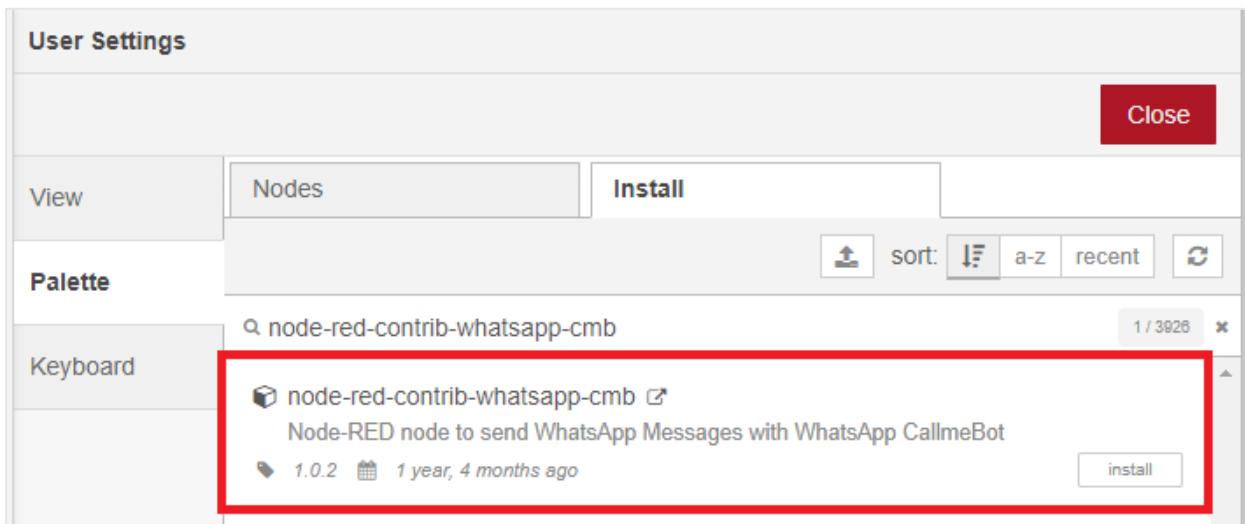
The WhatsApp message from the bot will contain the API key needed to send messages using the API.

² They may update the callmebot number at any moment, so we recommend double-checking the number at the official website: <https://www.callmebot.com/blog/free-api-whatsapp-messages/>

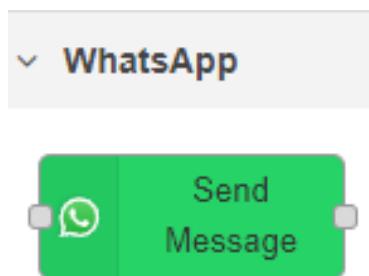
Installing WhatsApp Nodes

There are different nodes you can install in Node-RED that make it easy to send messages to your WhatsApp account. As we've mentioned previously, we'll use the nodes that use the CallmeBot service.

Go to **Menu > Manage Palette > Install** and search for `node-red-contrib-whatsapp-cmb`.



After installing, the WhatsApp node will show up on your palette.



Sending Messages to WhatsApp with Node-RED

Let's create a basic example that shows how to send messages to WhatsApp using the node you've just installed.

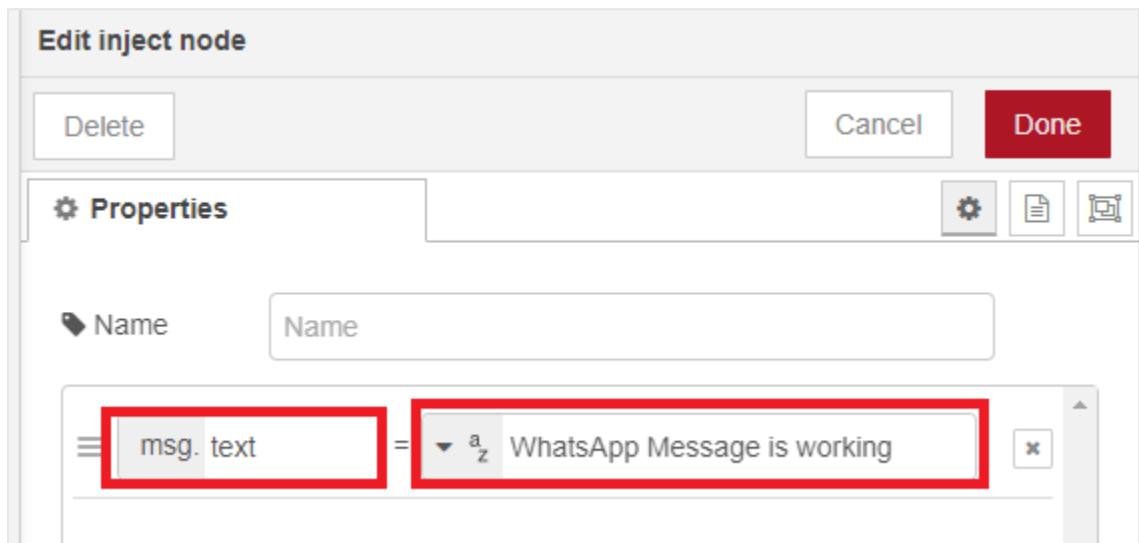
Drag a **Send Message WhatsApp** node, an **inject** node, and a **debug** node to the flow. Wire them as follows:



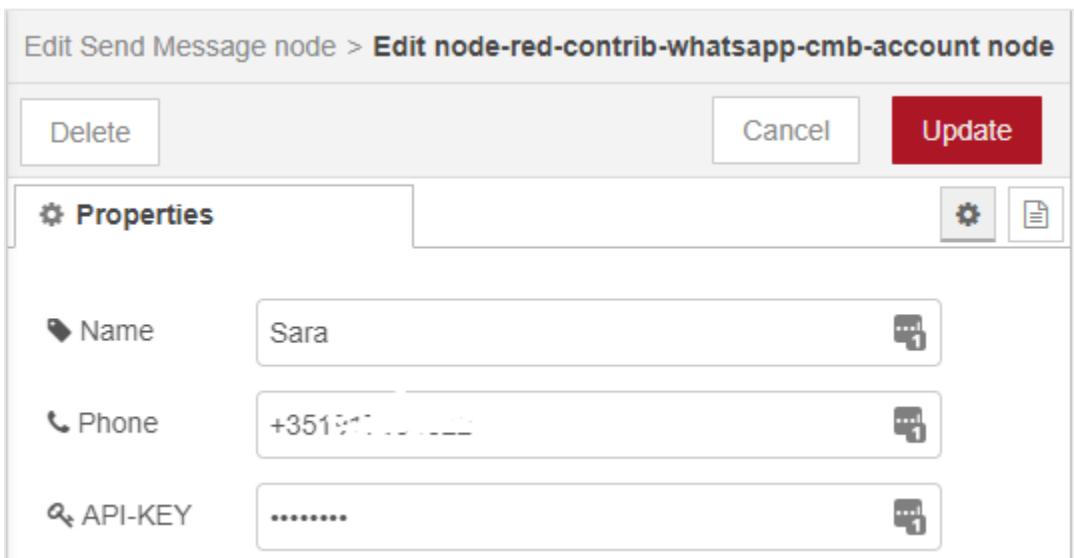
If you check the WhatsApp node documentation, it says:

"`msg.text` is used as the message you want to send. It must be a String. You can use it directly or with output from another node."

We'll set the message we want to send in `msg.text` in the **inject** node. Double-click on the **inject** node and edit its properties as follows (you can write any other message).

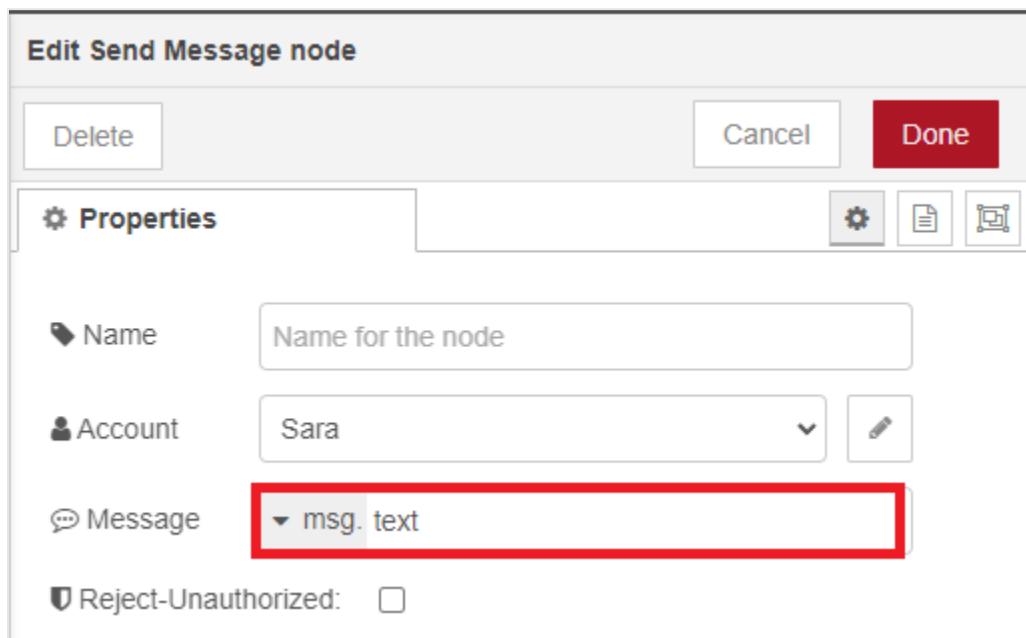


Double-click on the **Send Message** node. Click on the pencil icon to add a new WhatsApp account. Give it a name, insert your phone number (in international format) and the API key you received from the bot.



When you're done, click on **Update**.

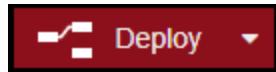
Then, change the Message field to `msg.text`.



Finally, click **Done**.

Testing the Flow

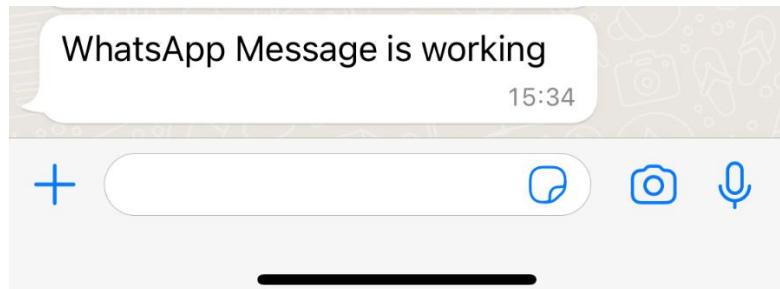
Deploy your application.



After deploying click on the left square on the **inject** node to trigger the **WhatsApp sender** node.



After a few seconds, you should receive a message in your WhatsApp account from the bot with the content you've set on the node:



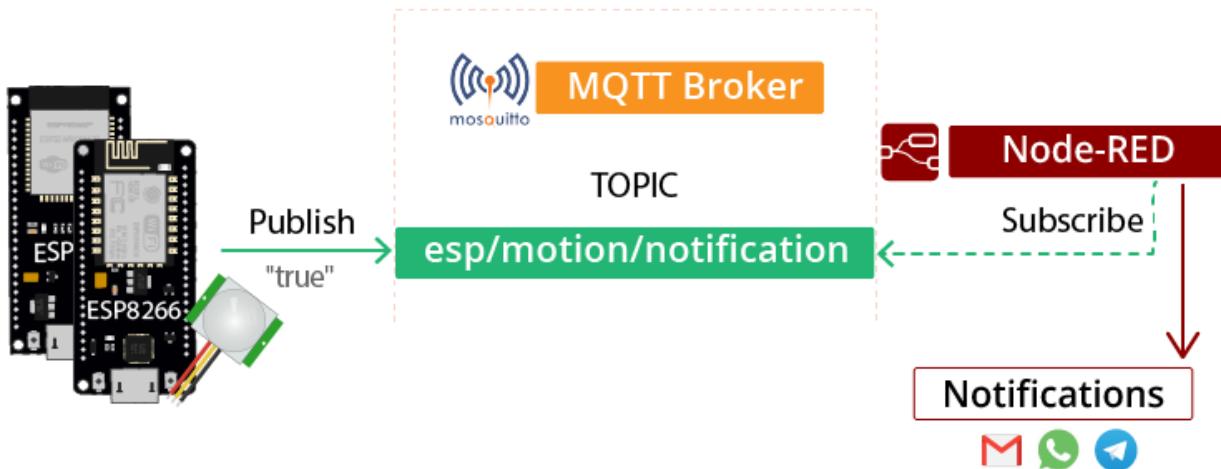
Congratulations! Now, you know how to send messages to your WhatsApp account using Node-RED and the callmebot service. You can apply what you learned to your home automation system to send useful notification messages.

7.4 - Motion Detector with Notifications

This unit shows you how to detect motion with an ESP32/ESP8266 and how to send a notification (email, Telegram message, or WhatsApp message).

Project Overview

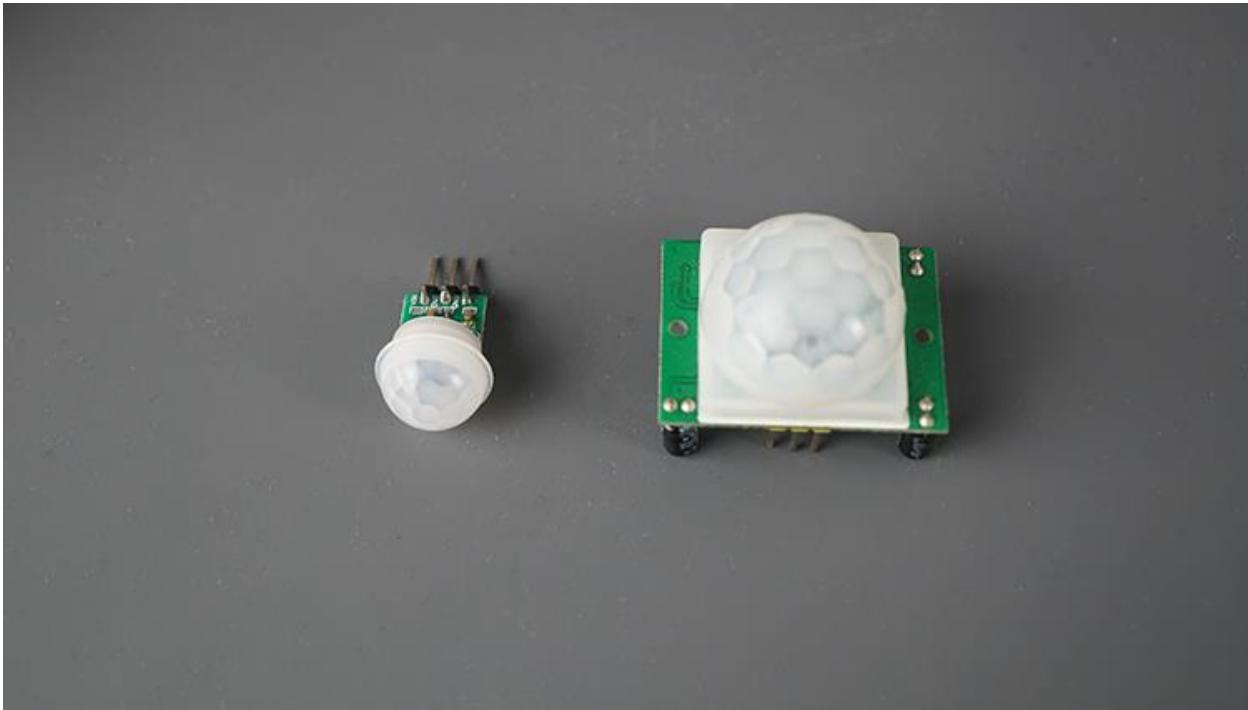
Here's a summary of what we'll build in this unit:



- The ESP32/ESP8266 is connected to a PIR motion sensor to detect motion;
- When motion is detected the ESP publishes a `true` message on the `esp/motion/notification` topic;
- We'll build a Node-RED flow that is subscribed to that same topic `esp/motion/notification`;
- When Node-RED receives the message on that topic, it sends a notification;
- In Node-RED UI, you can select which notification system you want to activate: WhatsApp messages, Telegram, or email; you can activate more than one or deactivate all notifications if not needed.

Detect Motion with the ESP32/ESP8266

To detect motion with the ESP32/ESP8266, we'll use a PIR motion sensor.



You can get one of the following PIR sensors:

- [Mini PIR motion sensor \(AM312\)](#) or [PIR motion sensor \(HC-SR501\)](#)

PIR sensors have a GND, VCC, and a data line. Connect the GND to the ESP GND, VCC to 3.3V, and the data line to an available GPIO. We'll use the following pins:

| | GND | VCC | Data |
|----------------|------------|------------|-------------|
| ESP32 | GND | 3.3V | GPIO26 |
| ESP8266 | GND | 3.3V | GPIO14 |

(You should keep the sketch, circuit, and nodes from the previous Units and add this new sensor to your project). In the case of the ESP8266, we'll remove the LED connected to GPIO 14 and add the PIR motion sensor.

Code

The code is slightly different for each board. Upload the correct code for the board you're using.

- [Download Sketch folder ESP32](#)
- [Download Sketch folder ESP8266](#)

We kept the code from previous units and added the lines of code to detect motion and publish to the `esp/motion/notification` topic.

ESP32 – Code

Here's the code for the ESP32. You need to insert your network credentials (SSID and password), the broker IP address, and the broker username and password. The lines where you need to insert your details are highlighted in light green. The new sections of code are highlighted in a light orange color.

```
#include <Arduino.h>
#include <WiFi.h>
extern "C" {
    #include "freertos/FreeRTOS.h"
    #include "freertos/timers.h"
}
#include <AsyncMqttClient.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// BROKER
#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106) */
#define MQTT_PORT 1883
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"

//MQTT Publish Topics
#define MQTT_PUB_TEMP "esp/bme280/temperature"
#define MQTT_PUB_HUM "esp/bme280/humidity"
#define MQTT_PUB_PRES "esp/bme280/pressure"
#define MQTT_PUB_MOTION "esp/motion/notification"
```

```

//MQTT Subscribe Topics
#define MQTT_SUB_DIGITAL "esp/digital/#"

// BME280 I2C
Adafruit_BME280 bme;

// Variables to hold sensor readings
float temp;
float hum;
float pres;

// Variables for motion
const int motionSensor = 26; // PIR Motion Sensor
bool motionDetected = false;

// Indicates when motion is detected
void IRAM_ATTR detectsMovement() {
    Serial.println("MOTION DETECTED!!!!");
    motionDetected = true;
}

AsyncMqttClient mqttClient;
TimerHandle_t mqttReconnectTimer;
TimerHandle_t wifiReconnectTimer;

unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

void WiFiEvent(WiFiEvent_t event) {
    Serial.printf("[WiFi-event] event: %d\n", event);
    switch(event) {
        case ARDUINO_EVENT_WIFI_STA_GOT_IP:
            Serial.println("WiFi connected");
            Serial.println("IP address: ");
            Serial.println(WiFi.localIP());
            connectToMqtt();
            break;
        case ARDUINO_EVENT_WIFI_STA_DISCONNECTED:
            Serial.println("WiFi lost connection");
            xTimerStop(mqttReconnectTimer, 0); // ensure we don't reconnect to MQTT
            while reconnecting to Wi-Fi
                xTimerStart(wifiReconnectTimer, 0);
                break;
    }
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
}

```

```

Serial.print("Session present: ");
Serial.println(sessionPresent);
//Subscribe to topics
// Subscribe to topic MQTT_SUB_DIGITAL when it connects to the broker
uint16_t packetIdSub1 = mqttClient.subscribe(MQTT_SUB_DIGITAL, 2);
Serial.print("Subscribing at QoS 2, packetId: ");
Serial.println(packetIdSub1);

}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        xTimerStart(mqttReconnectTimer, 0);
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message

    // Save the message in a variable
    String receivedMessage;
    for (int i = 0; i < len; i++) {
        Serial.println((char)payload[i]);
        receivedMessage += (char)payload[i];
    }
    // Save topic in a String variable
    String receivedTopic = String(topic);
    Serial.print("Received Topic: ");
    Serial.println(receivedTopic);

    // Check which GPIO we want to control
    int stringLen = receivedTopic.length();
    // Get the index of the last slash
    int lastSlash = receivedTopic.lastIndexOf("/");
    // Get the GPIO number (it's after the last slash "/")
    // esp/digital/GPIO
    String gpio = receivedTopic.substring(lastSlash+1, stringLen);
    Serial.print("DIGITAL GPIO: ");
    Serial.println(gpio);
    Serial.print("STATE: ");
    Serial.println(receivedMessage);
}

```

```

// Check if it is DIGITAL
if (receivedTopic.indexOf("digital") > 0) {
    //Set the specified GPIO as output
    pinMode(gpio.toInt(), OUTPUT);
    //Control the GPIO
    if (receivedMessage == "true"){
        digitalWrite(gpio.toInt(), HIGH);
    }
    else{
        digitalWrite(gpio.toInt(), LOW);
    }
}
Serial.println("Publish received.");
Serial.print(" topic: ");
Serial.println(topic);
Serial.print(" qos: ");
Serial.println(properties.qos);
Serial.print(" dup: ");
Serial.println(properties.dup);
Serial.print(" retain: ");
Serial.println(properties.retain);
Serial.print(" len: ");
Serial.println(len);
Serial.print(" index: ");
Serial.println(index);
Serial.print(" total: ");
Serial.println(total);
}

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();

    // PIR Motion Sensor mode INPUT_PULLUP
    pinMode(motionSensor, INPUT_PULLUP);
    // motionSensor pin as interrupt, assign interrupt function and set RISING mode
    attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);

    // Initialize BME280 sensor
    if (!bme.begin(0x76)) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
        while (1);
    }

    mqttReconnectTimer = xTimerCreate("mqttTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToMqtt));
    wifiReconnectTimer = xTimerCreate("wifiTimer", pdMS_TO_TICKS(2000), pdFALSE,
(void*)0, reinterpret_cast<TimerCallbackFunction_t>(connectToWifi));

    WiFi.onEvent(WiFiEvent);

    mqttClient.onConnect(onMqttConnect);
}

```

```

mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onSubscribe(onMqttSubscribe);
mqttClient.onUnsubscribe(onMqttUnsubscribe);
mqttClient.onMessage(onMqttMessage);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT);
mqttClient.setCredentials(BROKER_USER, BROKER_PASS);

connectToWifi();
}

void loop() {
    if(motionDetected){
        // Publish an MQTT message on topic esp/motion
        uint16_t packetIdPub4 = mqttClient.publish(MQTT_PUB_MOTION, 1, true,
"true");
        Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_MOTION, packetIdPub4);
        Serial.println("Motion Detected");
        motionDetected = false;
    }

    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 10 seconds)
    // it publishes a new MQTT message
    if (currentMillis - previousMillis >= interval) {
        // Save the last time a new reading was published
        previousMillis = currentMillis;

        // New BME280 sensor readings
        temp = bme.readTemperature();
        //temp = 1.8*bme.readTemperature() + 32;
        hum = bme.readHumidity();
        pres = bme.readPressure()/100.0F;

        // Publish an MQTT message on topic esp32/BME2800/temperature
        uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
        Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_TEMP, packetIdPub1);
        Serial.printf("Message: %.2f \n", temp);

        // Publish an MQTT message on topic esp32/BME2800/humidity
        uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(hum).c_str());
        Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ",
MQTT_PUB_HUM, packetIdPub2);
        Serial.printf("Message: %.2f \n", hum);

        // Publish an MQTT message on topic esp32/BME2800/pressure
        uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_PRES, 1, true,
String(pres).c_str());
        Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_PRES, packetIdPub3);
        Serial.printf("Message: %.3f \n", pres);
    }
}

```

ESP8266 – Code

Here's the code for the ESP8266. You need to insert your network credentials (SSID and password), the broker IP address, and the broker username and password. The lines where you need to insert your details are highlighted in light green. The new lines of code are highlighted in light orange.

```
#include <Arduino.h>

#include <ESP8266WiFi.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BME280.h>

#define WIFI_SSID "REPLACE_WITH_YOUR_SSID"
#define WIFI_PASSWORD "REPLACE_WITH_YOUR_PASSWORD"

// BROKER
#define MQTT_HOST IPAddress(XXX, XXX, X, XXX) // MQTT BROKER IP ADDRESS
/*for example:
#define MQTT_HOST IPAddress(192, 168, 1, 106) */
#define MQTT_PORT 1883
#define BROKER_USER "REPLACE_WITH_BROKER_USERNAME"
#define BROKER_PASS "REPLACE_WITH_BROKER_PASSWORD"

//MQTT Topics
#define MQTT_PUB_TEMP "esp/bme280/temperature"
#define MQTT_PUB_HUM "esp/bme280/humidity"
#define MQTT_PUB_PRES "esp/bme280/pressure"
#define MQTT_PUB_MOTION "esp/motion/notification"

//MQTT Subscribe Topics
#define MQTT_SUB_DIGITAL "esp8266/digital/#"

// BME280 I2C
Adafruit_BME280 bme;

// Variables to hold sensor readings
float temp;
float hum;
float pres;

// Variables for motion
const int motionSensor = 14; // PIR Motion Sensor
bool motionDetected = false;

// Indicates when motion is detected
void ICACHE_RAM_ATTR detectsMovement() {
    Serial.println("MOTION DETECTED!!!");
    motionDetected = true;
```

```

}

AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;

WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
Ticker wifiReconnectTimer;

unsigned long previousMillis = 0;      // Stores last time a message was published
const long interval = 10000;           // Interval at which to publish values

void connectToWifi() {
    Serial.println("Connecting to Wi-Fi...");
    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}

void connectToMqtt() {
    Serial.println("Connecting to MQTT...");
    mqttClient.connect();
}

void onWifiConnect(const WiFiEventStationModeGotIP& event) {
    Serial.println("Connected to Wi-Fi.");
    connectToMqtt();
}

void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
    Serial.println("Disconnected from Wi-Fi.");
    mqttReconnectTimer.detach(); // ensure we don't reconnect to MQTT while
reconnecting to Wi-Fi
    wifiReconnectTimer.once(2, connectToWifi);
}

void onMqttConnect(bool sessionPresent) {
    Serial.println("Connected to MQTT.");
    Serial.print("Session present: ");
    Serial.println(sessionPresent);
    //Subscribe to topics
    // Subscribe to topic MQTT_SUB_DIGITAL when it connects to the broker
    uint16_t packetIdSub1 = mqttClient.subscribe(MQTT_SUB_DIGITAL, 2);
    Serial.print("Subscribing at QoS 2, packetId: ");
    Serial.println(packetIdSub1);
}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
    Serial.println("Disconnected from MQTT.");

    if (WiFi.isConnected()) {
        mqttReconnectTimer.once(2, connectToMqtt);
    }
}

void onMqttSubscribe(uint16_t packetId, uint8_t qos) {
    Serial.println("Subscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
    Serial.print(" qos: ");
    Serial.println(qos);
}

```

```

}

void onMqttUnsubscribe(uint16_t packetId) {
    Serial.println("Unsubscribe acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void onMqttMessage(char* topic, char* payload, AsyncMqttClientMessageProperties properties, size_t len, size_t index, size_t total) {
    // Do whatever you want when you receive a message

    // Save the message in a variable
    String receivedMessage;
    for (int i = 0; i < len; i++) {
        Serial.println((char)payload[i]);
        receivedMessage += (char)payload[i];
    }
    // Save topic in a String variable
    String receivedTopic = String(topic);
    Serial.print("Received Topic: ");
    Serial.println(receivedTopic);

    // Check which GPIO we want to control
    int stringLen = receivedTopic.length();
    // Get the index of the last slash
    int lastSlash = receivedTopic.lastIndexOf("/");
    // Get the GPIO number (it's after the last slash "/")
    // esp/digital/GPIO
    String gpio = receivedTopic.substring(lastSlash+1, stringLen);
    Serial.print("DIGITAL GPIO: ");
    Serial.println(gpio);
    Serial.print("STATE: ");
    Serial.println(receivedMessage);

    // Check if it is DIGITAL
    if (receivedTopic.indexOf("digital") > 0) {
        //Set the specified GPIO as output
        pinMode(gpio.toInt(), OUTPUT);
        //Control the GPIO
        if (receivedMessage == "true"){
            digitalWrite(gpio.toInt(), HIGH);
        }
        else{
            digitalWrite(gpio.toInt(), LOW);
        }
    }
    Serial.println("Publish received.");
    Serial.print(" topic: ");
    Serial.println(topic);
    Serial.print(" qos: ");
    Serial.println(properties.qos);
    Serial.print(" dup: ");
    Serial.println(properties.dup);
    Serial.print(" retain: ");
    Serial.println(properties.retain);
    Serial.print(" len: ");
    Serial.println(len);
    Serial.print(" index: ");
}

```

```

    Serial.println(index);
    Serial.print(" total: ");
    Serial.println(total);
}

void onMqttPublish(uint16_t packetId) {
    Serial.println("Publish acknowledged.");
    Serial.print(" packetId: ");
    Serial.println(packetId);
}

void setup() {
    Serial.begin(115200);
    Serial.println();
    Serial.println();

    // PIR Motion Sensor mode INPUT_PULLUP
    pinMode(motionSensor, INPUT_PULLUP);
    // motionSensor pin as interrupt, assign interrupt function and set RISING mode
    attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);

    // Initialize BME280 sensor
    if (!bme.begin(0x76)) {
        Serial.println("Could not find a valid BME280 sensor, check wiring!");
        while (1);
    }

    wifiConnectHandler = WiFi.onStationModeGotIP(onWifiConnect);
    wifiDisconnectHandler = WiFi.onStationModeDisconnected(onWifiDisconnect);

    mqttClient.onConnect(onMqttConnect);
    mqttClient.onDisconnect(onMqttDisconnect);
    mqttClient.onSubscribe(onMqttSubscribe);
    mqttClient.onUnsubscribe(onMqttUnsubscribe);
    mqttClient.onMessage(onMqttMessage);
    mqttClient.onPublish(onMqttPublish);
    mqttClient.setServer(MQTT_HOST, MQTT_PORT);
    mqttClient.setCredentials(BROKER_USER, BROKER_PASS);

    connectToWifi();
}

void loop() {

    if(motionDetected) {
        // Publish an MQTT message on topic esp/motion
        uint16_t packetIdPub4 = mqttClient.publish(MQTT_PUB_MOTION, 1, true,
"true");
        Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_MOTION, packetIdPub4);
        Serial.println("Motion Detected");
        motionDetected = false;
    }

    unsigned long currentMillis = millis();
    // Every X number of seconds (interval = 10 seconds)
    // it publishes a new MQTT message
    if (currentMillis - previousMillis >= interval) {
}

```

```

// Save the last time a new reading was published
previousMillis = currentMillis;

// New BME280 sensor readings
temp = bme.readTemperature();
//temp = 1.8*bme.readTemperature() + 32;
hum = bme.readHumidity();
pres = bme.readPressure()/100.0F;

// Publish an MQTT message on topic esp32/BME2800/temperature
uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_TEMP, packetIdPub1);
Serial.printf("Message: %.2f \n", temp);

// Publish an MQTT message on topic esp32/BME2800/humidity
uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(hum).c_str());
Serial.printf("Publishing on topic %s at QoS 1, packetId %i: ",
MQTT_PUB_HUM, packetIdPub2);
Serial.printf("Message: %.2f \n", hum);

// Publish an MQTT message on topic esp32/BME2800/pressure
uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_PRES, 1, true,
String(pres).c_str());
Serial.printf("Publishing on topic %s at QoS 1, packetId: %i",
MQTT_PUB_PRES, packetIdPub3);
Serial.printf("Message: %.3f \n", pres);
}
}

```

How the Code Works

Continue reading to learn how the new sections of code work.

Motion Topic

We create a variable called `MQTT_PUB_MOTION` that holds the topic we want to publish to indicate whether motion was detected.

```
#define MQTT_PUB_MOTION "esp/motion/notification"
```

PIR Sensor Variables

Create a variable called `motionSensor` to refer to the PIR motion sensor. In the case of the ESP32, we're using GPIO 26. For the ESP8266 we're using GPIO14.

```
const int motionSensor = 26; // PIR Motion Sensor
```

I've also created a control variable called `motionDetected` to indicate whether motion was detected.

```
bool motionDetected = false;
```

setup()

In the `setup()`, we'll set the PIR motion sensor as an `INPUT_PULLUP`.

```
pinMode(motionSensor, INPUT_PULLUP);
```

To trigger an event with a PIR motion sensor, you use interrupts. Interrupts are useful for making things happen automatically in microcontroller programs and can help solve timing problems.

Interrupts

With interrupts, you don't need to constantly check the current value of a pin. With interrupts, when a change is detected, an event is triggered (a function is called).

To set an interrupt in the Arduino IDE, you use the `attachInterrupt()` function, which accepts as arguments: the GPIO pin, the name of the function to be executed, and the mode:

```
attachInterrupt(digitalPinToInterrupt(GPIO), function, mode);
```

GPIO

The first argument is a GPIO number. Normally, you should use `digitalPinToInterrupt(GPIO)` to set the actual GPIO as an interrupt pin.

Function to be triggered

The second argument of the `attachInterrupt()` function is the name of the function that will be called every time the interrupt is triggered. That function can also be called ISR (interrupt service routine). In our case, it's the `detectsMovement()`

function. ISRs should be placed in RAM and shouldn't include delays. The code to be executed should be as simple as possible.

```
attachInterrupt(digitalPinToInterruption(motionSensor), detectsMovement, RISING);
```

Mode

The third argument is the mode. There are 5 different modes:

- LOW: to trigger the interrupt whenever the pin is LOW;
- HIGH: to trigger the interrupt whenever the pin is HIGH;
- CHANGE: to trigger the interrupt whenever the pin changes value – for example from HIGH to LOW or LOW to HIGH;
- FALLING: for when the pin goes from HIGH to LOW;
- RISING: to trigger when the pin goes from LOW to HIGH.

For this example will be using the RISING mode, because when the PIR motion sensor detects motion, the GPIO it is connected to goes from LOW to HIGH.

detectsMovement() function

The `detectsMovement()` function will run when motion is detected. In our case, we simply set the `motionDetected` variable to `true` and print a message in the Serial Monitor. Then, in the `loop()`, we'll check the value of the `motionDetected` variable and act accordingly.

To place the function in RAM, we must use `IRAM_ATTR` before the function name (for the ESP32).

```
// Indicates when motion is detected
void IRAM_ATTR detectsMovement() {
    Serial.println("MOTION DETECTED!!!");
    motionDetected = true;
}
```

If you're using an ESP8266, use the prefix `ICACHE_RAM_ATTR`:

```
// Indicates when motion is detected
void ICACHE_RAM_ATTR detectsMovement() {
    Serial.println("MOTION DETECTED!!!");
    motionDetected = true;
}
```

loop()

In the `loop()`, we check the value of the `motionDetected` variable. If it is true, we add the line of code to publish a `true` message to the `esp/motion/notification` topic. Then, we set the `motionDetected` variable to `false` again so that we can detect motion later on.

```
if(motionDetected){
    // Publish an MQTT message on topic esp/motion
    uint16_t packetIdPub4 = mqttClient.publish(MQTT_PUB_MOTION, 1, true,
    "true");
    Serial.printf("Publishing on topic %s at QoS 1, packetId: %i", MQTT_PUB_MOTION,
    packetIdPub4);
    Serial.println("Motion Detected");
    motionDetected = false;
}
```

Uploading the Code

Upload the previous code to your ESP board. After uploading press the board RST button so that it starts running the code. Wave your hand over the PIR motion sensor and see if your board is detecting motion properly. When motion is detected a message is printed in the Serial monitor and a message is sent to the `esp/motion/notification` topic.

At the same time, the board is sending temperature, humidity, and pressure readings and listening to messages on the `esp/digital/#` topic.

The screenshot shows a 'Serial Monitor' window with the title 'Output' and 'Serial Monitor'. The text area contains the following log entries:

```
Publishing on topic esp/bme280/pressure at QoS 1, packetId: 4Message: 1000.000
Publish acknowledged.
packetId: 2
Publish acknowledged.
packetId: 3
Publish acknowledged.
packetId: 4
Publishing on topic esp/bme280/temperature at QoS 1, packetId: 5Message: 32.74
Publishing on topic esp/bme280/humidity at QoS 1, packetId: 6: Publish acknowledged.
packetId: 5
Message: 37.31
Publishing on topic esp/bme280/pressure at QoS 1, packetId: 7Message: 1000.646
Publish acknowledged.
packetId: 6
Publish acknowledged.
packetId: 7
MOTION DETECTED!!!
```

Creating the Node-RED Flow

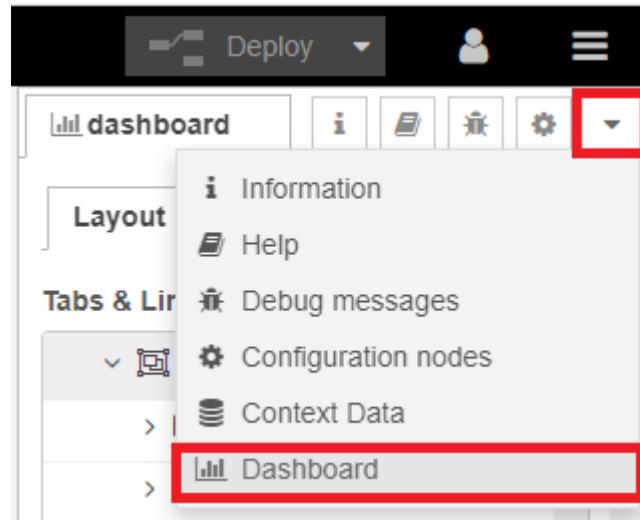
At this point, creating a Node-RED flow that subscribes to the `esp/motion/notification` topic and sends a notification should be easy after following previous units.

We'll show you how to trigger the three different types of notifications we've seen previously (WhatsApp messages, Telegram messages, and email). You can add all of them to your project, or just the one that makes more sense for you.

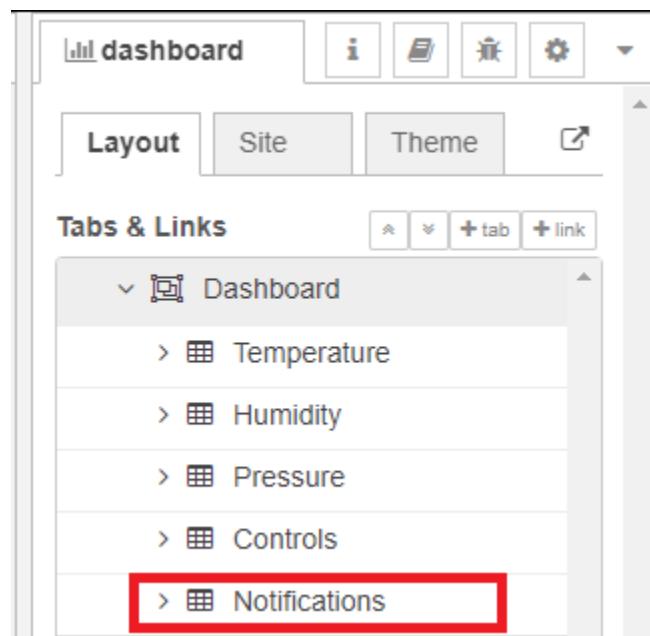
Creating a Dashboard Group

We want to add the option to enable and disable notifications. For example, when you're close to the PIR, it doesn't make sense to have the notifications activated. So, we'll add a **switch** node to each type of notification that you can activate and deactivate.

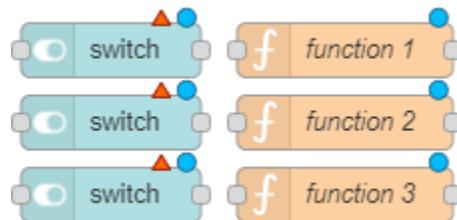
Go to the Dashboard tab and create a new group for the notifications switches.



We created a new group called notifications.

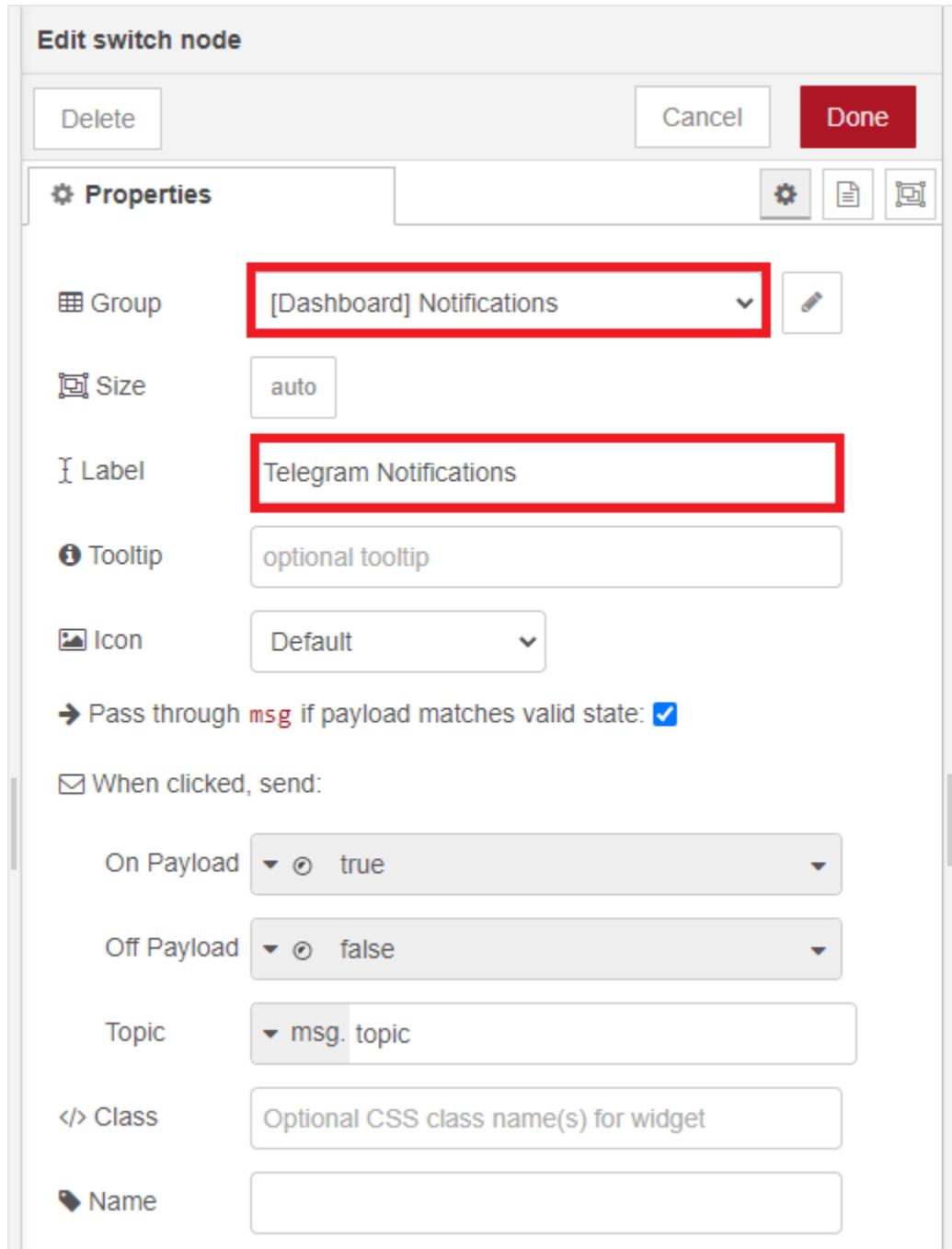


Drag three **switch** dashboard nodes and three **function** nodes to the flow. Connect each **switch** node to a **function** node.



Switch Nodes

Double-click on one of the switches. Add it to the **Notifications** group and give it a name. This will be the **Telegram Notifications** switch.



Similarly, edit the other **switch** nodes, but give them different names: **WhatsApp Notifications** and **Email Notifications**.

Edit switch node

Properties

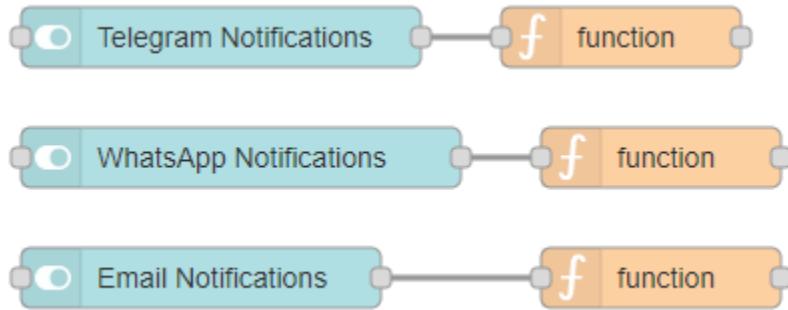
| | |
|---|---------------------------------------|
| Group | [Dashboard] Notifications |
| Size | auto |
| Label | Email Notifications |
| Tooltip | optional tooltip |
| Icon | Default |
| → Pass through <code>msg</code> if payload matches valid state: <input checked="" type="checkbox"/> | |
| <input type="checkbox"/> When clicked, send: | |
| On Payload | true |
| Off Payload | false |
| Topic | msg.topic |
| </> Class | Optional CSS class name(s) for widget |
| Name | |

Edit switch node

Properties

| | |
|---|---------------------------------------|
| Group | [Dashboard] Notifications |
| Size | auto |
| Label | WhatsApp Notifications |
| Tooltip | optional tooltip |
| Icon | Default |
| → Pass through <code>msg</code> if payload matches valid state: <input checked="" type="checkbox"/> | |
| <input type="checkbox"/> When clicked, send: | |
| On Payload | true |
| Off Payload | false |
| Topic | msg.topic |
| </> Class | Optional CSS class name(s) for widget |
| Name | |

This is what the flow looks like at the moment:



When the switch is on it sends a `msg.payload` with the `true` value, and when the switch is off it sends a `msg.payload` of `off` to the next node, in our case the **function** node.

Function Nodes

On each function node, we'll create a **global variable** to indicate if that notification must be enabled or disabled. The global variable can be accessed by any other node on the flow.

Context

This brings up to the table a new concept that's worth exploring, **context**.

Node-RED provides a way to store information that can be shared between different nodes without using the messages that pass through a flow. This is called '**context**'.

Context scopes

The 'scope' of a particular context value determines who it is shared with. There are three context scope levels:

- **Node** - only visible to the node that set the value
- **Flow** - visible to all nodes on the same flow (or tab in the editor)
- **Global** - visible to all nodes

The choice of scope for any particular value will depend on how it is being used.
[\(source\)](#)

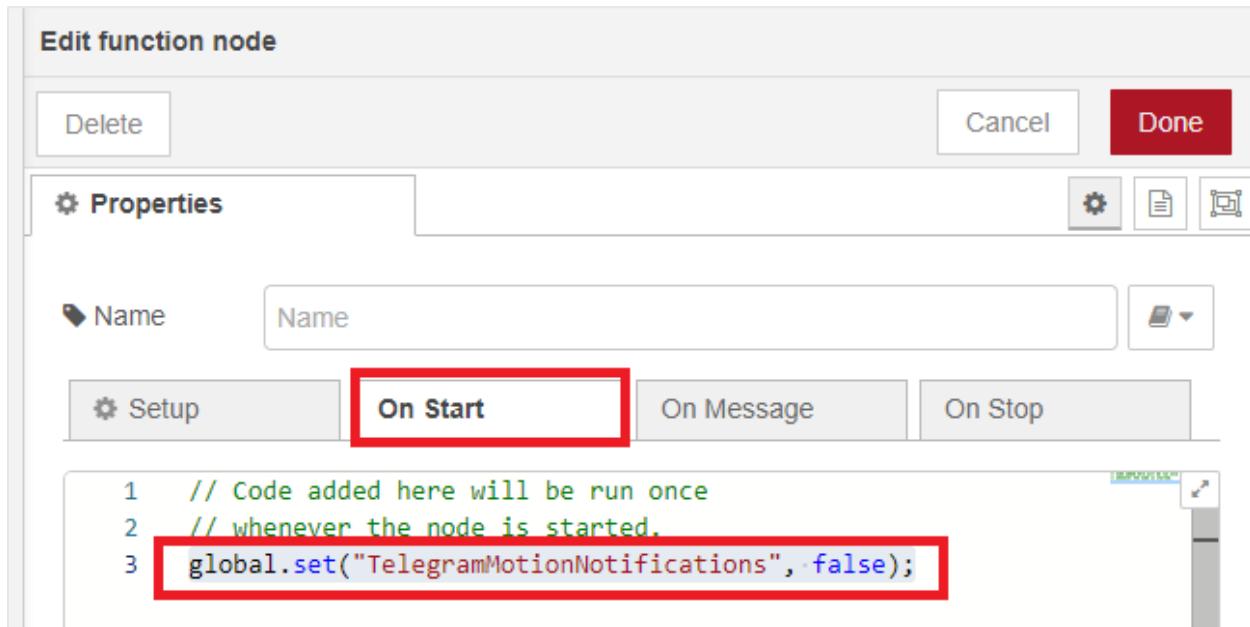
Double-click on the Telegram **function** node and select the **On Start** tab. Then, copy the following:

```
global.set("TelegramMotionNotifications", false);
```

The code that is added to the **On Start** tab will run once when the node starts (when the application is deployed). When the node starts, we want the notifications to be disabled by default. So, our global variable for the Telegram notifications should be **false**.

To create a global variable called `TelegramMotionNotifications` with the value `false`, use:

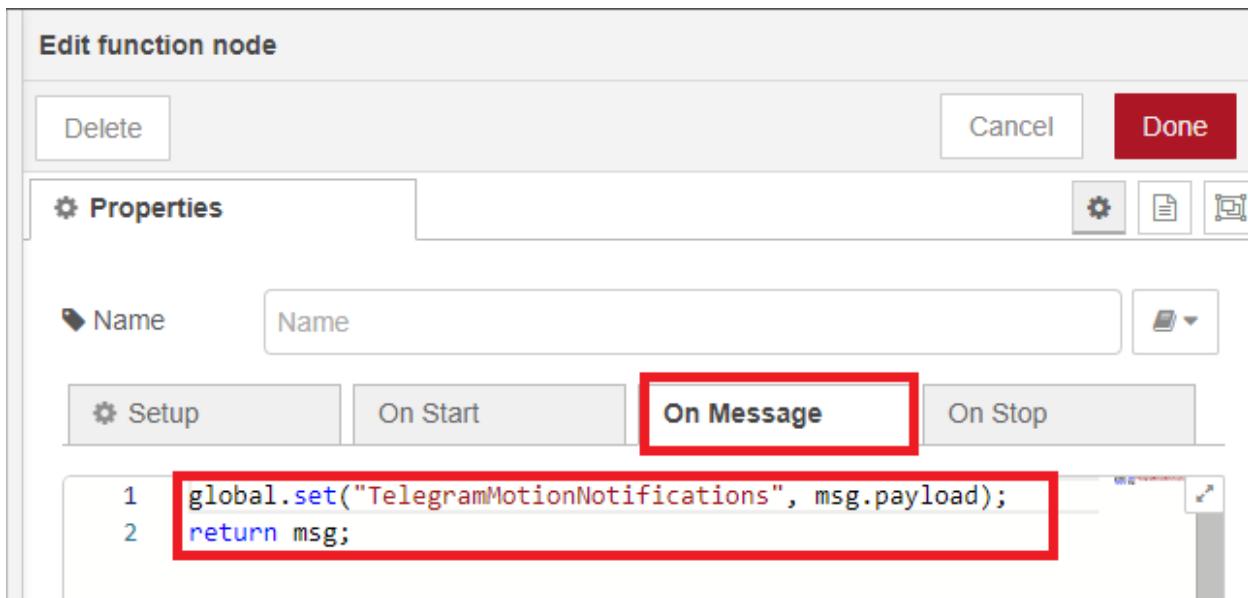
```
global.set("TelegramMotionNotifications", false);
```



When the application is deployed, the value of the global variable is `false`. However, after that, we want to change it whenever we change the state of the corresponding switch. So, now you need to edit the **On Message** tab to set the value of the `TelegramMotionNotifications` variable to the switch payload value.

Open the On Message tab and copy the following:

```
global.set("TelegramMotionNotifications", msg.payload);
return msg;
```



Similarly, edit the other function nodes, but use different names for the global variables.

We'll use: `WhatsAppMotionNotifications` and `emailMotionNotifications`.

Here's the function node for the WhatsApp notifications.

The image contains two screenshots of the 'Edit function node' dialog box. The top screenshot shows the 'On Start' tab selected (highlighted with a red box). The code editor contains:

```
1 // Code added here will be run once
2 // whenever the node is started.
3 global.set("WhatsAppMotionNotifications", false);
```

The bottom screenshot shows the 'On Message' tab selected (highlighted with a red box). The code editor contains:

```
1 global.set("WhatsAppMotionNotifications", msg.payload);
2 return msg;
```

And finally, here's the function node for the email notifications.

The screenshot shows a Node-RED node configuration window. At the top, there are four tabs: 'Setup', 'On Start' (which is highlighted with a red border), 'On Message', and 'On Stop'. The main area contains the following code:

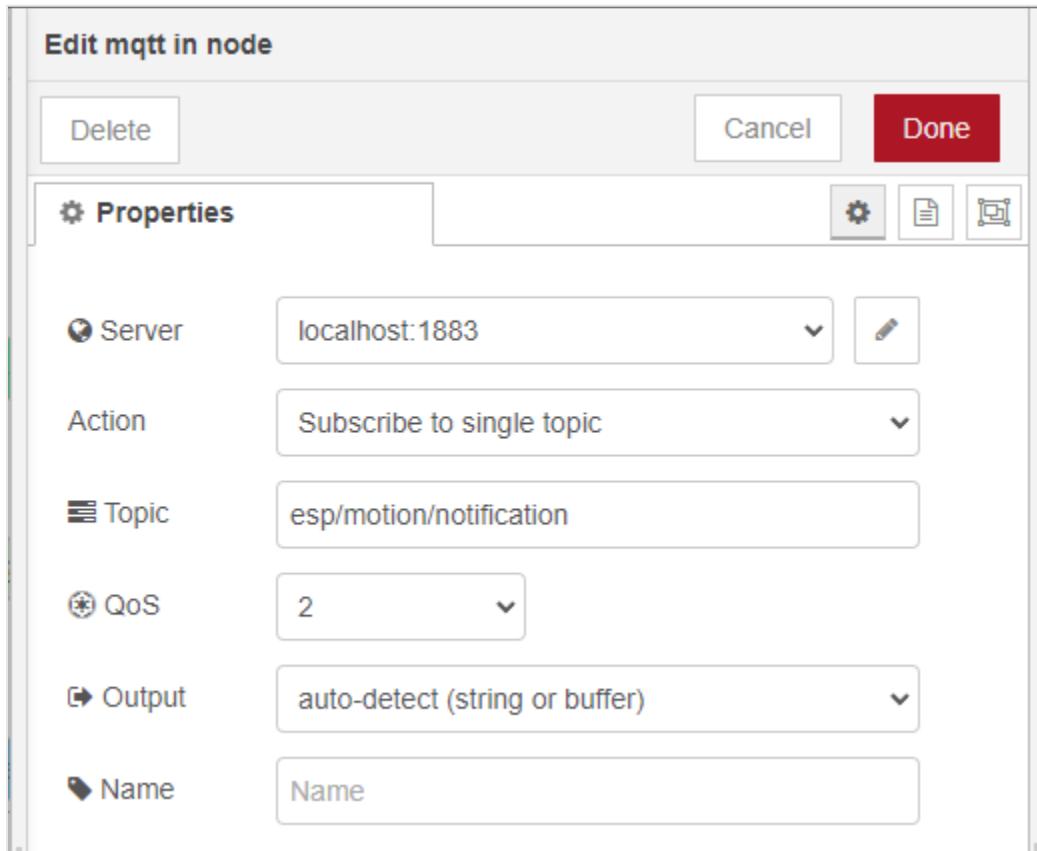
```
1 // Code added here will be run once
2 // whenever the node is started.
3 global.set("emailMotionNotifications", false);
```

The screenshot shows a Node-RED node configuration window. At the top, there are four tabs: 'Setup', 'On Start', 'On Message' (which is highlighted with a red border), and 'On Stop'. The main area contains the following code:

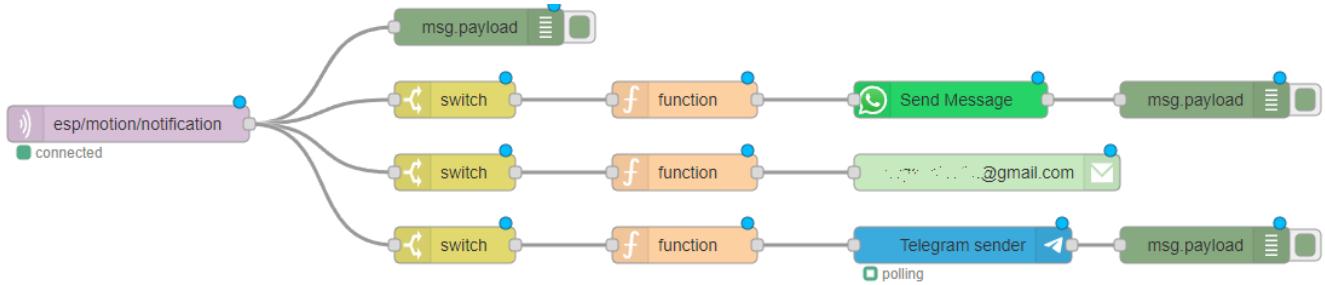
```
1 global.set("emailMotionNotifications", msg.payload);
2 return msg;
```

Subscribe to esp/motion/notification Topic

The ESP32/ESP8266 publishes a true message on the esp/motion/notification topic whenever it detects motion. To receive that message on Node-RED, we need to subscribe to that topic. Drag an MQTT In node to the flow and edit its properties to subscribe to the esp/motion/notification topic.



Then, add three **switch** (function) nodes, three **debug** nodes, three **function** nodes, one **WhatsApp send message** node, one **Telegram sender** node, and one **email sender** node. Wire the nodes to create the following flow:



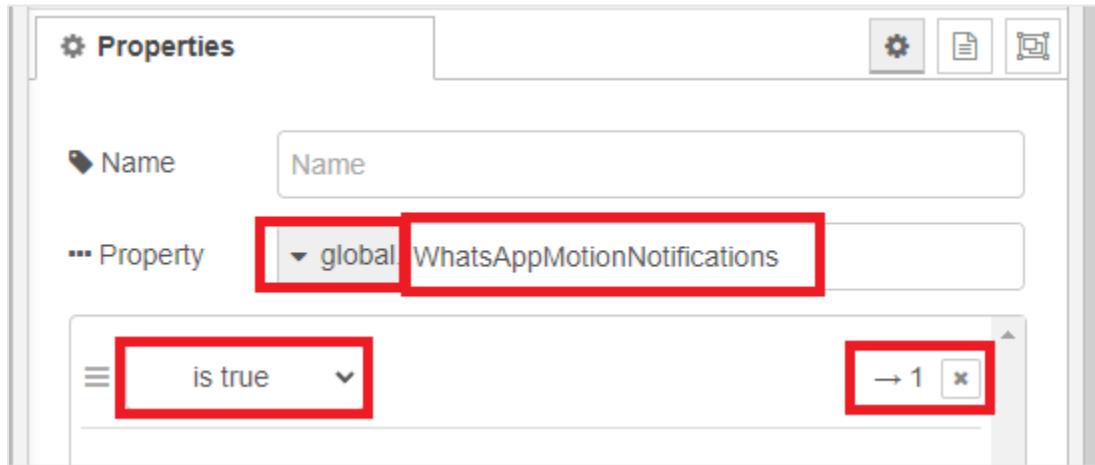
The **switch** (yellow) node allows you to pass the received message to different output nodes (or not pass the message at all) depending on the message value. On that switch node, we'll check whether the global variables for the notifications are true and only send the message to the next node if that's the case.

If you check the **switch** node documentation:

"When a message arrives, the node will evaluate each of the defined rules and forward the message to the corresponding outputs of any matching rules."

WhatsApp Messages

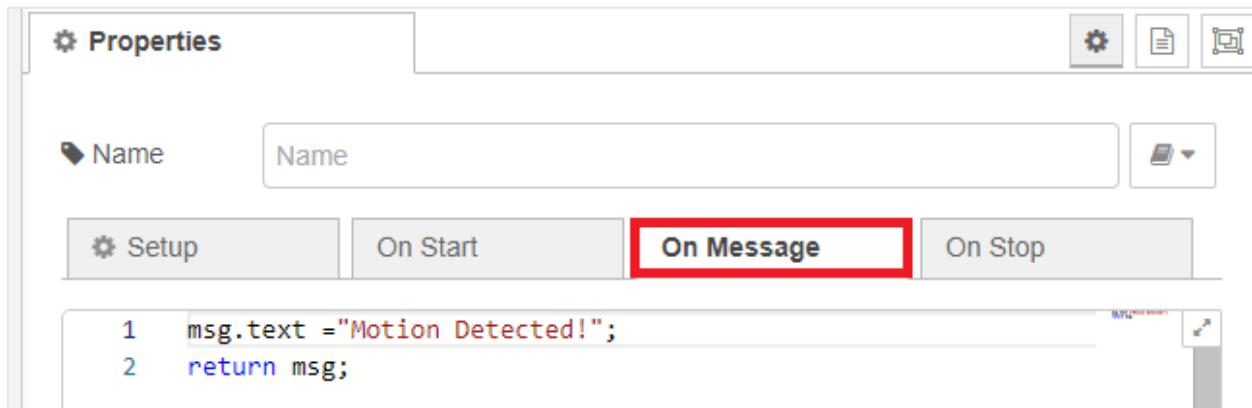
Edit the first **switch** node (for the WhatsApp messages) as follows:



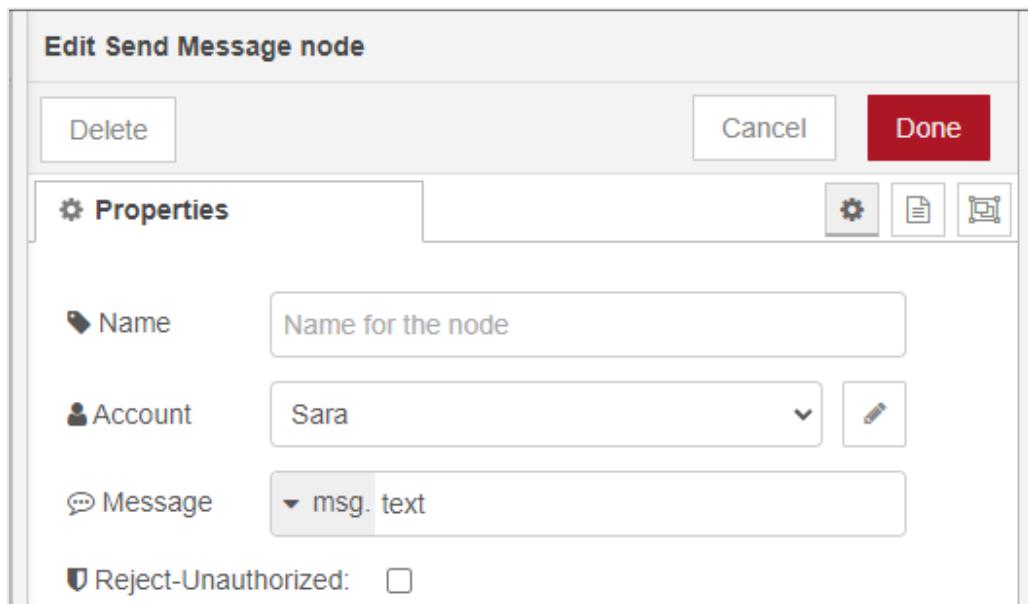
This means that if the global variable called `WhatsAppMotionNotifications` is true, it will forward the received message (the MQTT message) to the first output. If the `WhatsAppMotionNotifications` variable has any other value, the message won't pass to the following node.

Now, edit the WhatsApp **function** node. As we've seen previously, you must set the message we want to send on `msg.text`. So, copy the following code to the node:

```
msg.text = "Motion Detected!";
return msg;
```



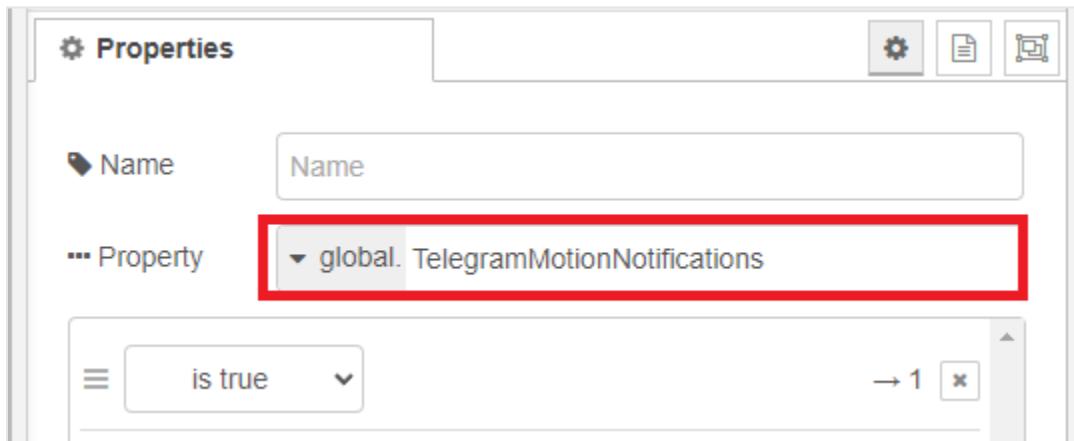
Next, edit the **Send Message** node to include your WhatsApp account (see Unit 7.3).



The WhatsApp notifications flow is ready.

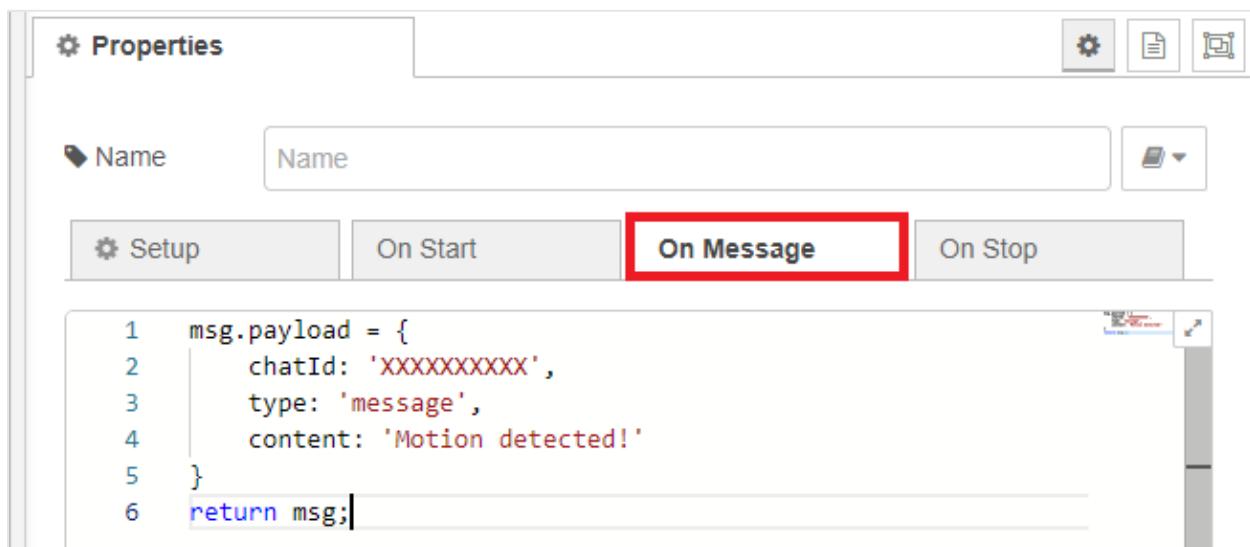
Telegram Messages

Proceed in a similar way for the Telegram notifications flow, but in this case, we want to check the value of the `TelegramMotionNotifications` variable. Check the figure below.

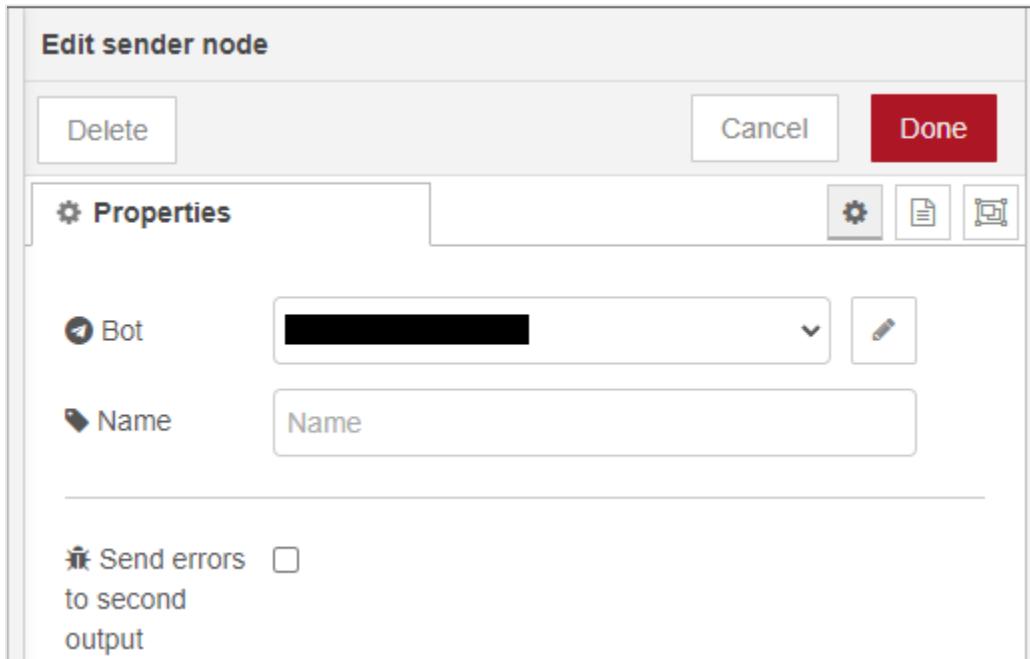


Edit the **function** node to prepare the message to be sent. Don't forget to insert your Telegram chat UID.

```
msg.payload = {
  chatId: 'XXXXXXXXXX',
  type: 'message',
  content: 'Motion detected!'
}
return msg;
```



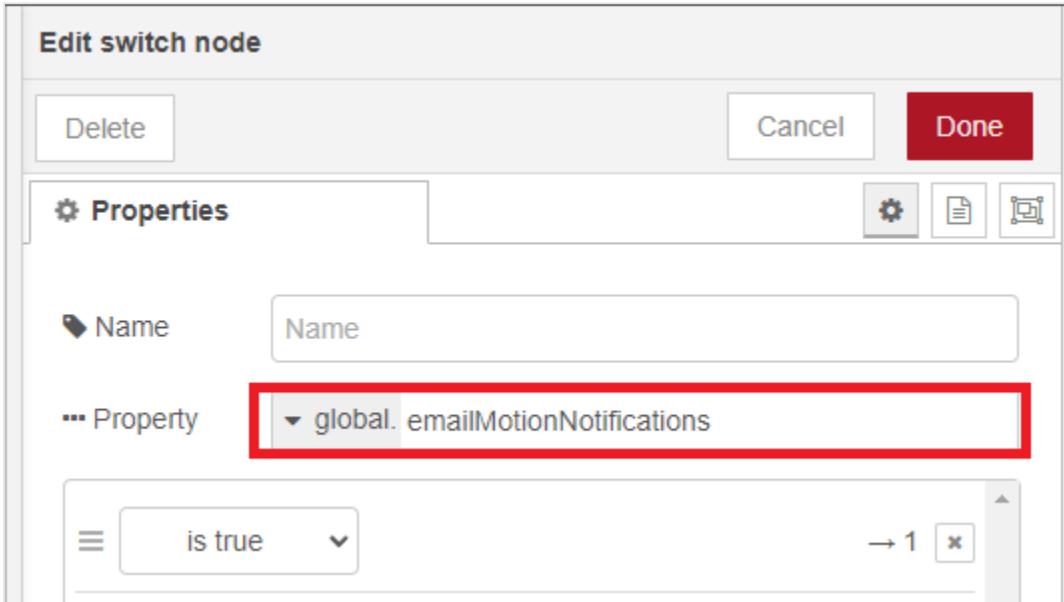
Finally, edit the **Telegram Sender** node with your Bot information (see Unit 7.2).



The Telegram flow is ready.

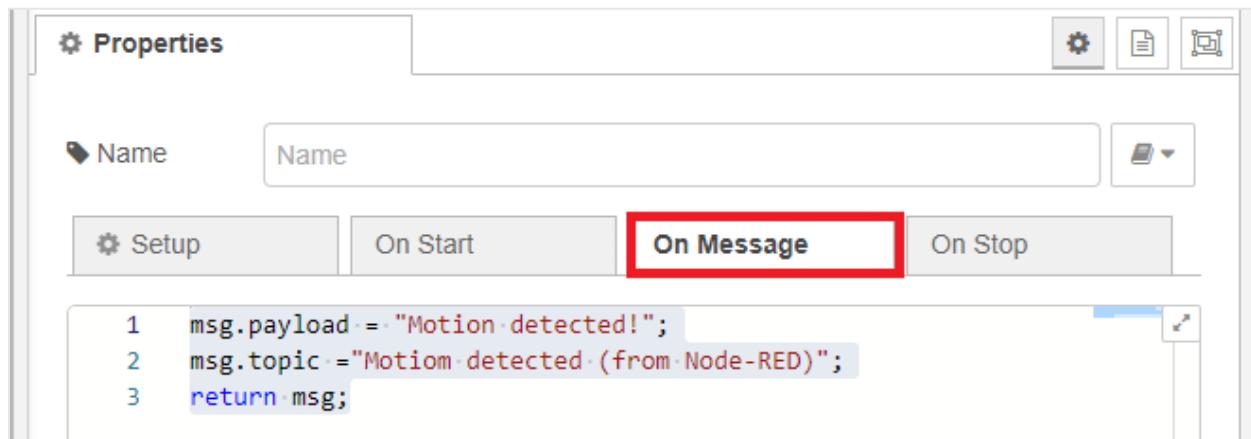
Email Messages

Finally, let's edit the nodes to send an email. Edit the **switch** node as in the previous examples, but with the `emailMotionNotifications` variable.

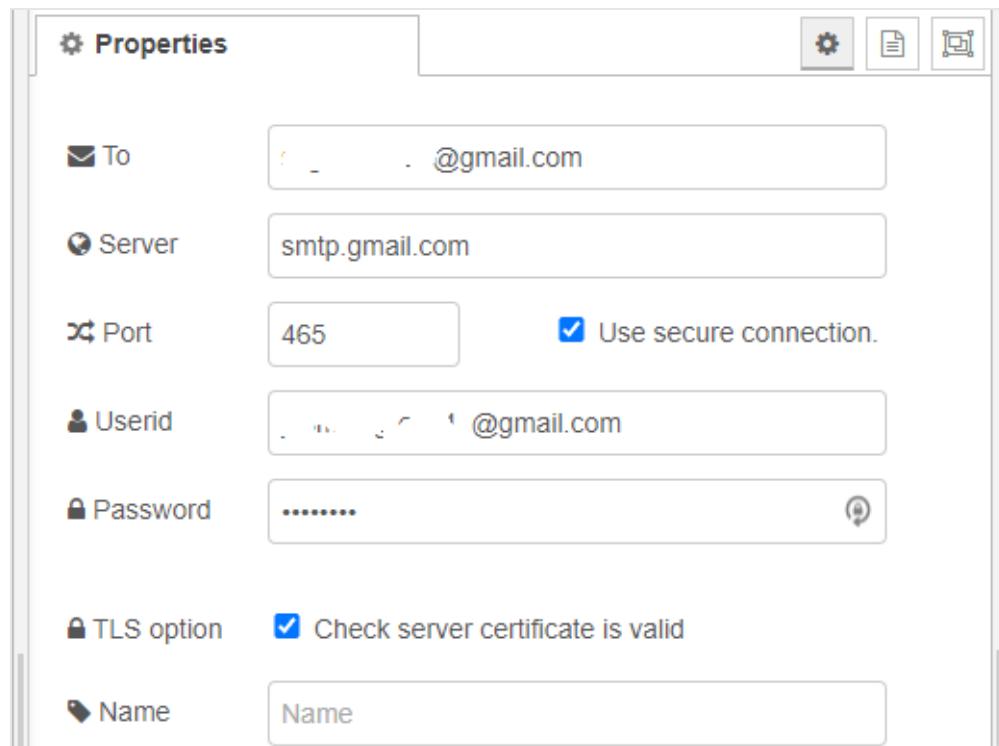


Now, edit the function node to set the email body (`msg.payload`) and email subject (`msg.topic`).

```
msg.payload = "Motion detected!";
msg.topic = "Motion detected (from Node-RED)";
return msg;
```



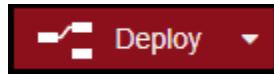
Finally, edit the email node with the recipient email, sender, and sender app password (see unit 7.1).



The email notifications flow is ready.

Testing the Flow

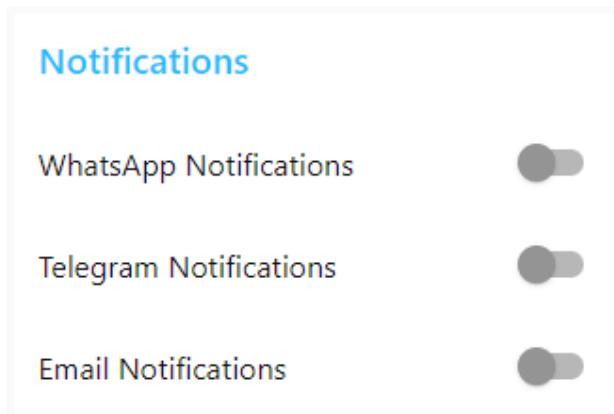
Deploy your application.



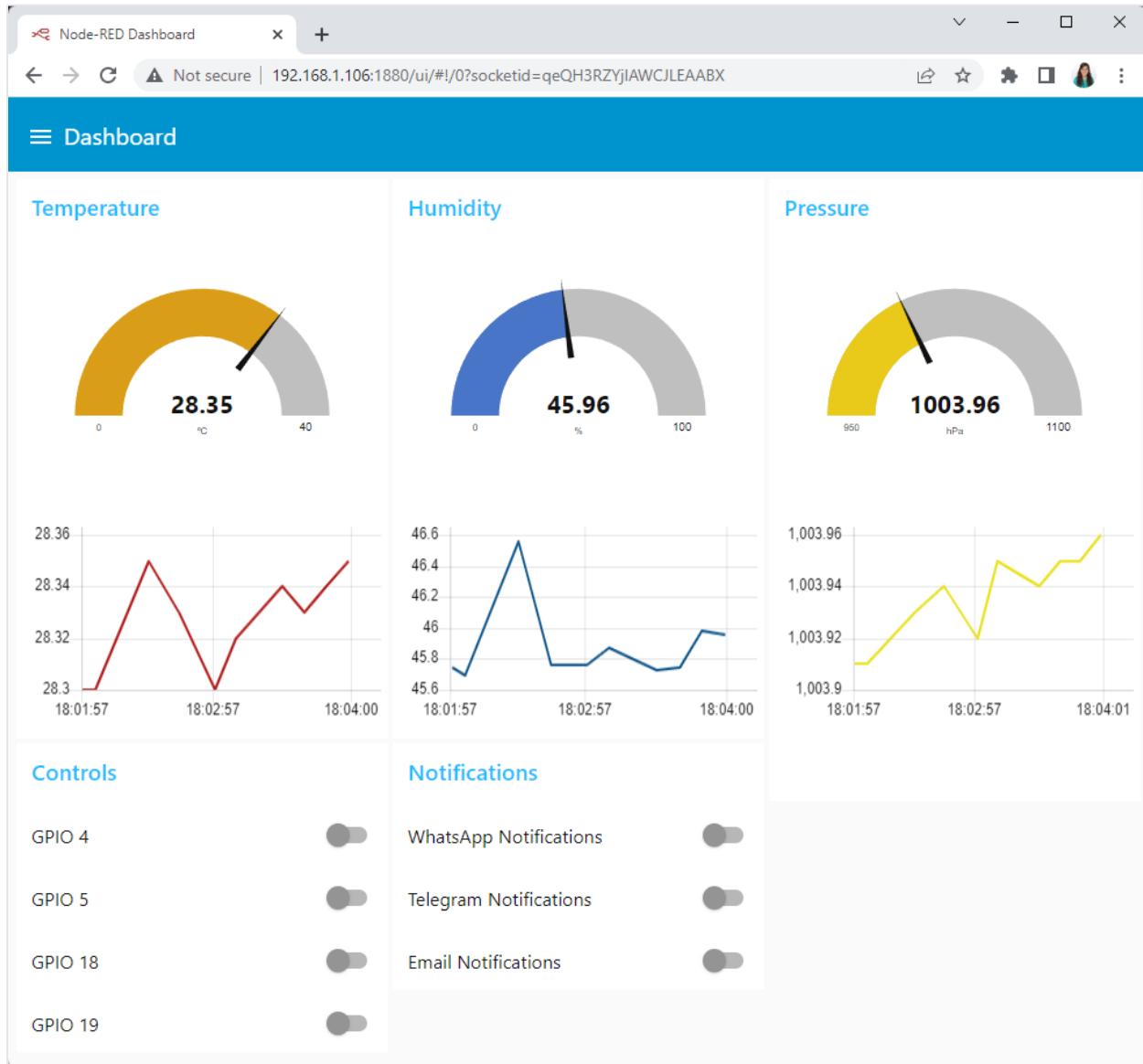
After deploying, go to Node-RED UI:

`http://Your_RPi_IP_address:1880/ui`

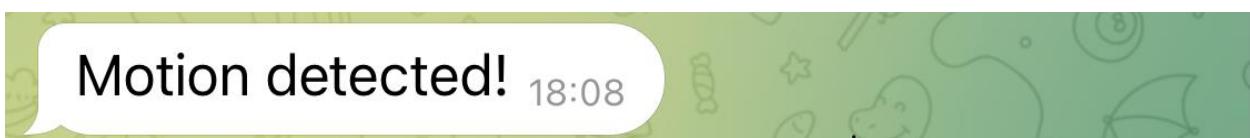
There should be a group with the notification switches.

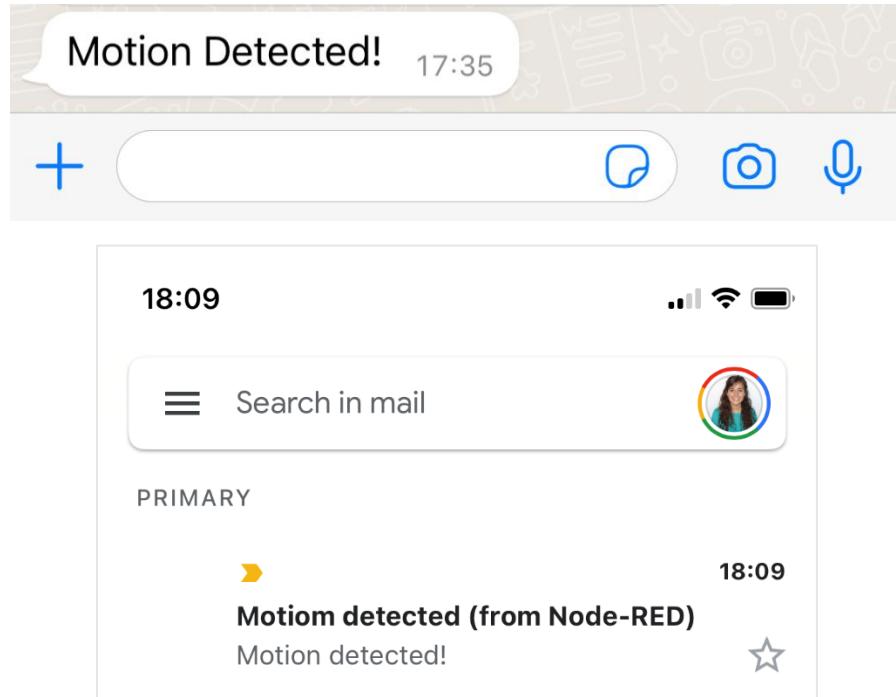


If you've added the new nodes to the flows we created on previous units, you should have a dashboard as shown in the following picture:



Activate the switches and wave your hand in front of the PIR motion sensor and check that you receive the messages (WhatsApp, Telegram, and/or email).

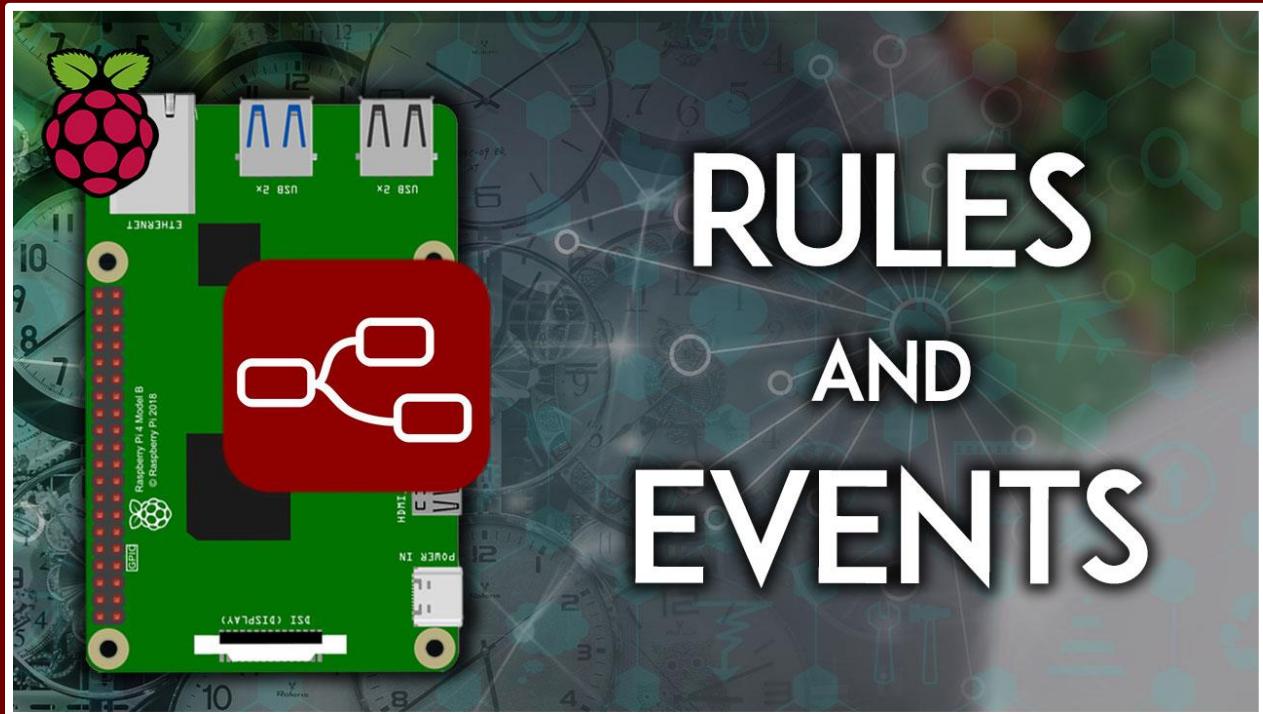




Then, you can turn off the switches and check that you do not receive any messages. If that's the case, everything is working properly. Congratulations! You've just created a motion detector with a notification system.

MODULE 8

Adding Rules and Triggering Events



In this Module, you'll learn how to automatically set all GPIOs to a defined state using master switches and modes. You'll also learn how to trigger events when something happens (a notification, a threshold value, etc.) Finally, you'll learn how to set timers and schedule events.

8.1 - Creating Master Switches or Modes

In any home automation system, it's useful to create specific modes tailored for your home.

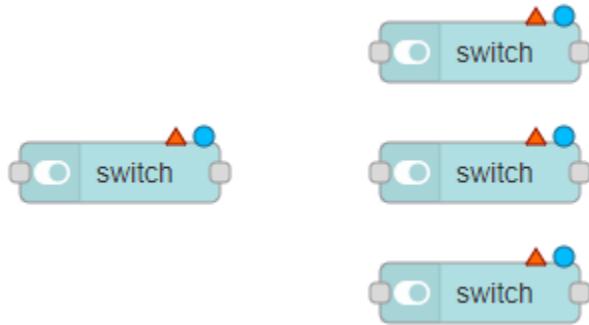
Master Switch

For instance, it's handy to press a button that turns on/off all the lights in your home.

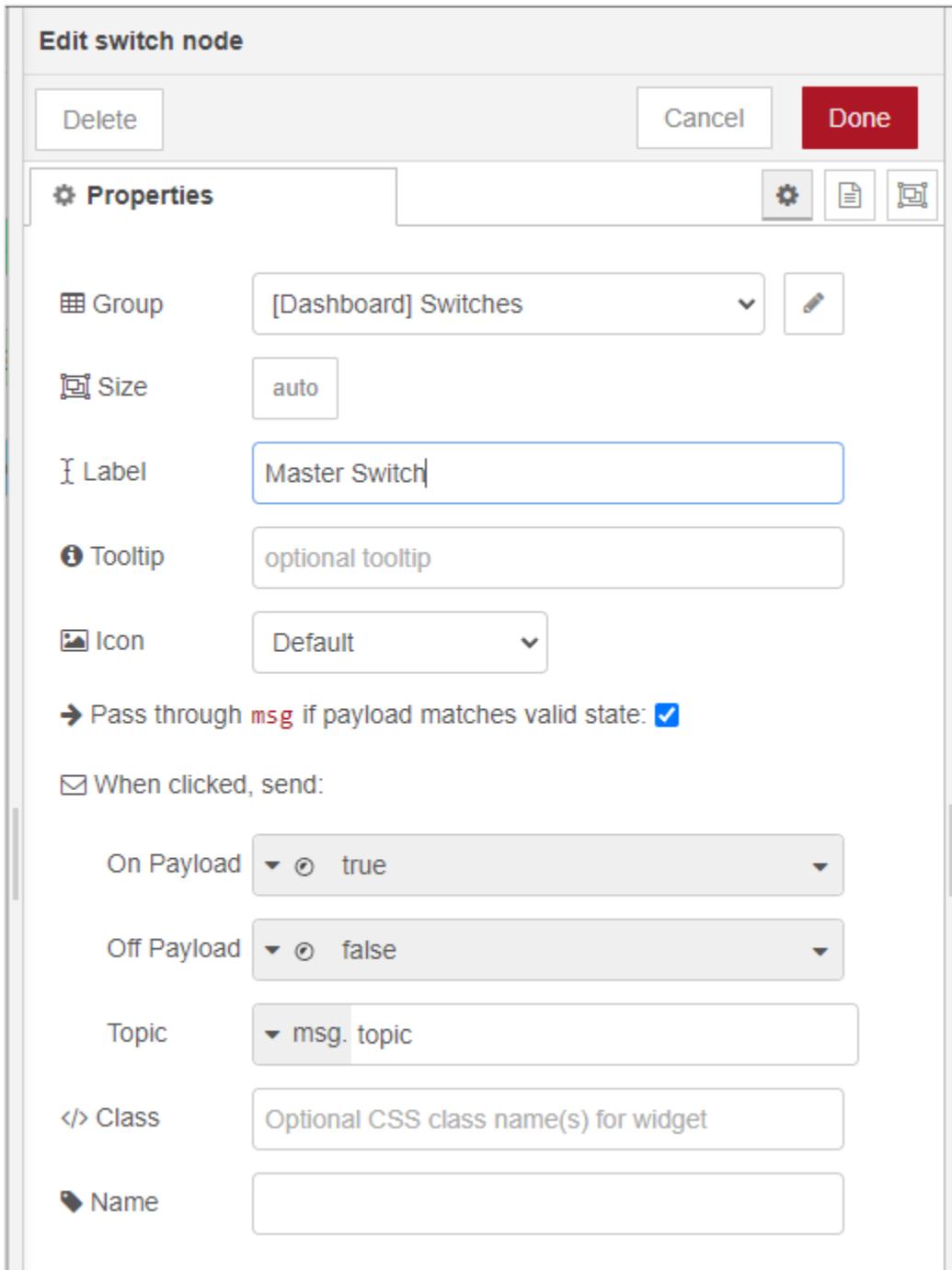
With Node-RED this can be accomplished with what I call a master switch. We'll show you an example of how you can implement a master switch.

Creating the Flow

First drag 4 **switch** nodes to your flow:

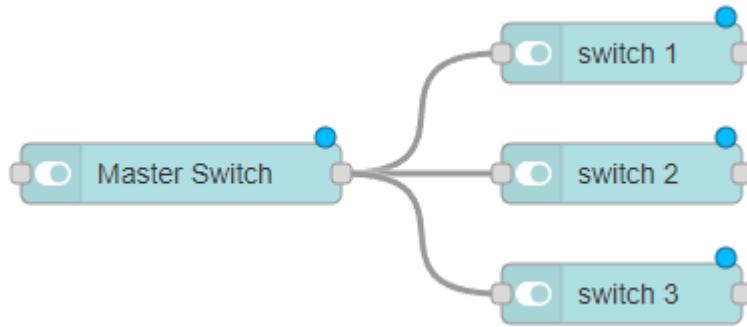


Open the configuration for one of the **switches** and call it **Master Switch**. Assign it to an existing dashboard group or a new group.



Now, edit the other switches by giving them different label names: `switch 1`, `switch 2`, and `switch 3`. Don't forget to assign the same dashboard group of the master switch.

Finally, connect your Master Switch to all the other **switch** nodes that you want to control, as shown in the following figure:



Testing the Master Switch

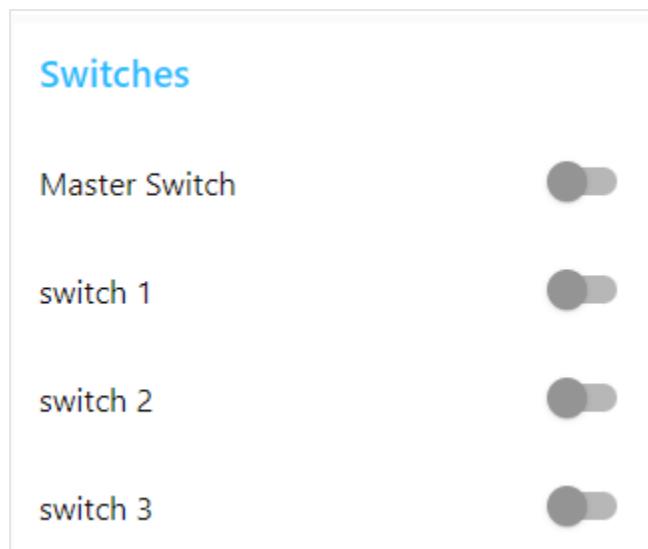
Deploy your application.



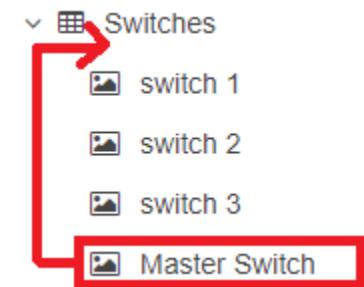
After deploying, go to Node-RED UI:

```
http://Your_RPi_IP_address:1880/ui
```

Open the Node-RED UI tab. There should be a group with your switches.

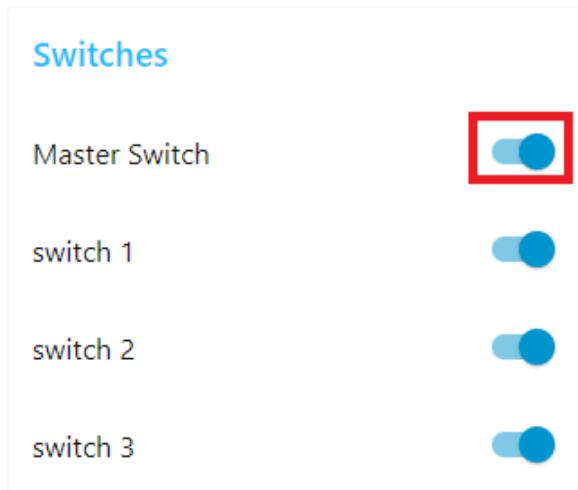


If the switches don't show up in the right order (the Master Switch first), go to the **dashboard** tab, open the group that contains the switches, and you can rearrange the order of the switches. Drag the Master Switch widget to the beginning. Then, redeploy your application again.



If you press the **Master Switch**, you can instantly control all the other **switches**.

If you press on, it turns on all the other **switches**.



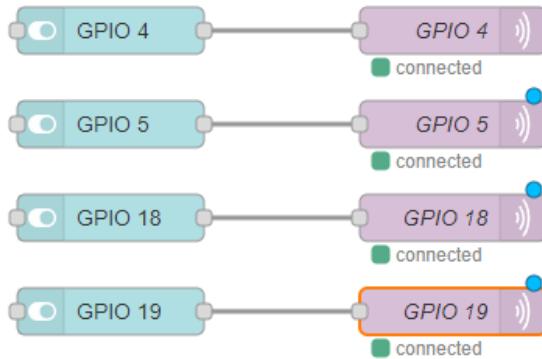
When you press the **Master Switch** off, it turns off all the other **switches**.

Note: you can also control each **switch** individually.

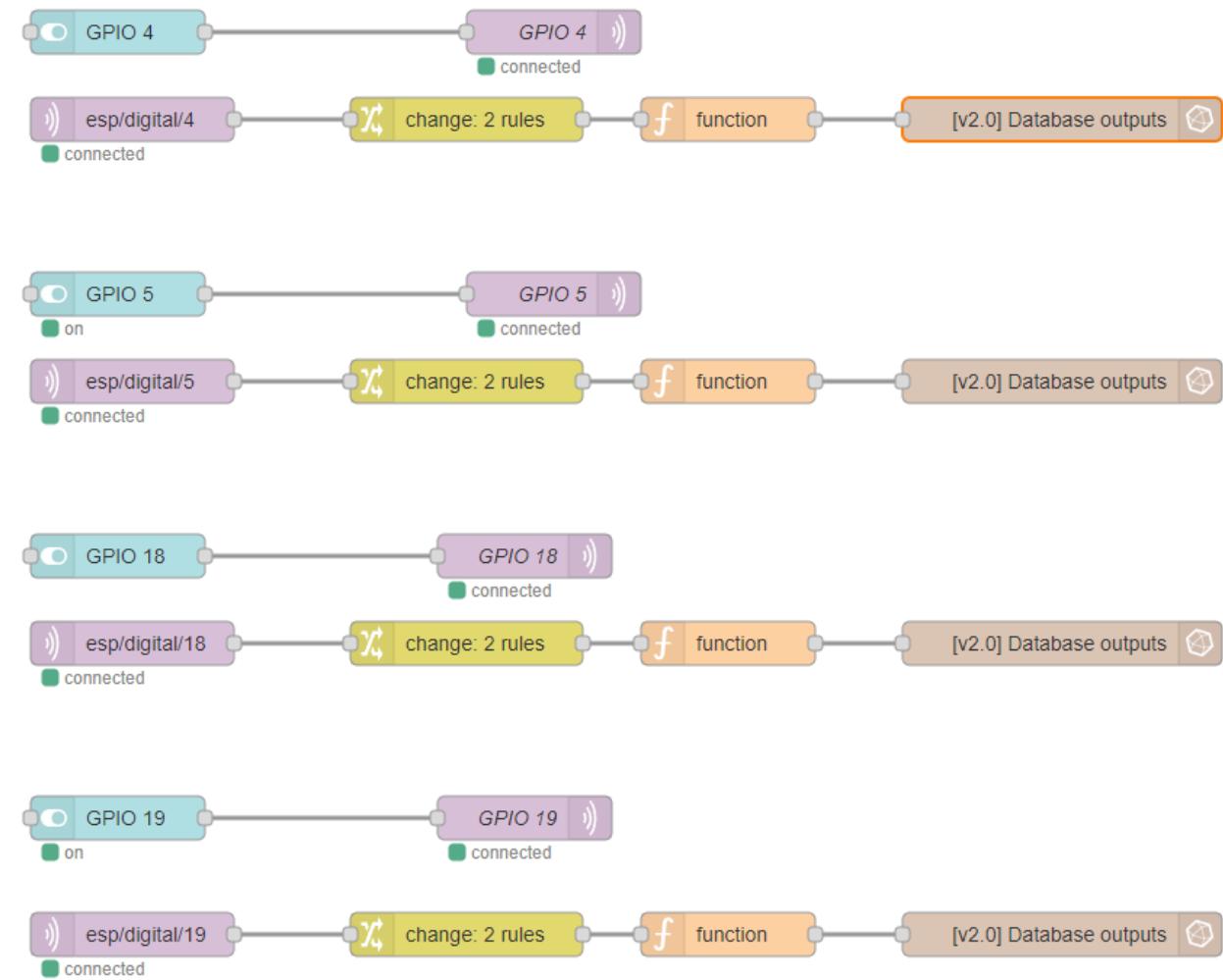
Applying the Master Switch to GPIO Controls

You can apply this idea to the GPIO controls you already have from Unit 5.4.

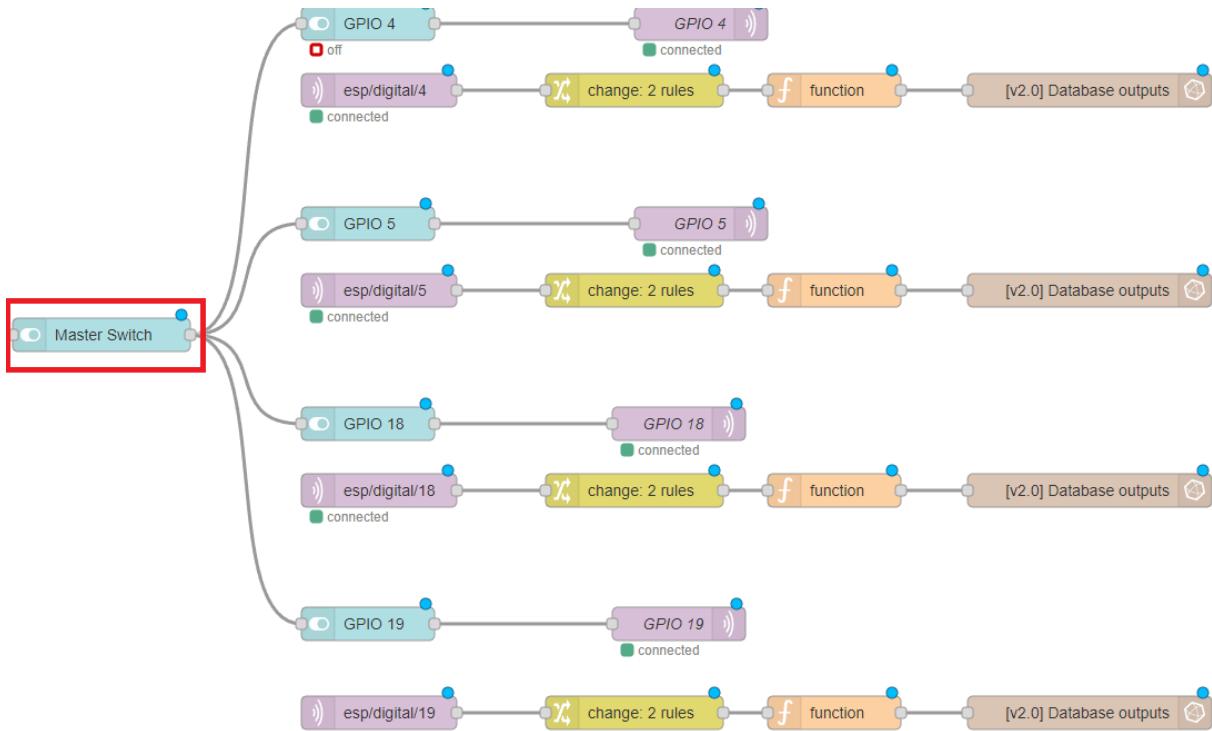
From that unit, you have the following flow:



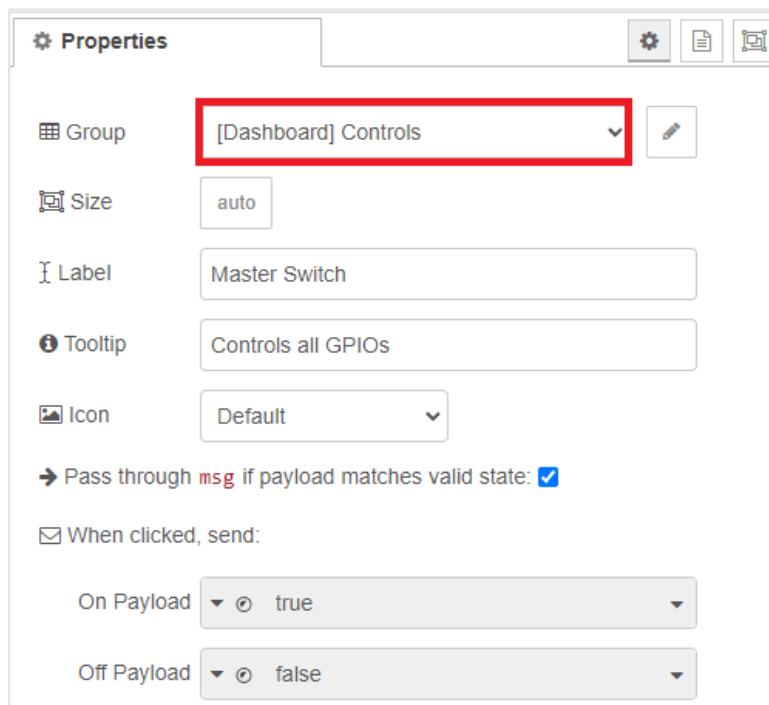
Or the following if you've followed the InfluxDB units:



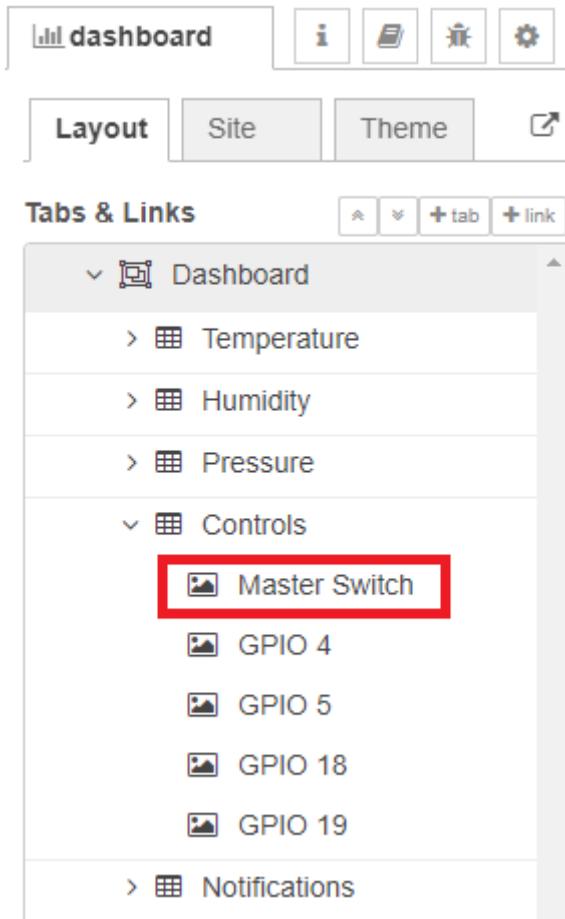
To apply the Master Switch to the GPIO Controls, you just need to drag a new dashboard **switch** node to the flow and wire it to the GPIO switches.



Don't forget to assign the same dashboard group to the GPIOs and to the Master Switch. In our case, it's the **Controls** group.



You may need to rearrange the dashboard layout if you want the Master Switch to show up at first.



Deploy your application.



After deploying, go to Node-RED UI:

```
http://Your_RPi_IP_address:1880/ui
```

Open the Node-RED UI tab. There should be a group with your controls and Master Switch. You can control all GPIOs simultaneously using the **Master Switch**.

| Controls | | Controls | |
|---------------|-------------------------------------|---------------|--------------------------|
| Master Switch | <input checked="" type="checkbox"/> | Master Switch | <input type="checkbox"/> |
| GPIO 4 | <input checked="" type="checkbox"/> | GPIO 4 | <input type="checkbox"/> |
| GPIO 5 | <input checked="" type="checkbox"/> | GPIO 5 | <input type="checkbox"/> |
| GPIO 18 | <input checked="" type="checkbox"/> | GPIO 18 | <input type="checkbox"/> |
| GPIO 19 | <input checked="" type="checkbox"/> | GPIO 19 | <input type="checkbox"/> |

Try the Master Switch and check that it is working as expected. You can still control each GPIO individually.

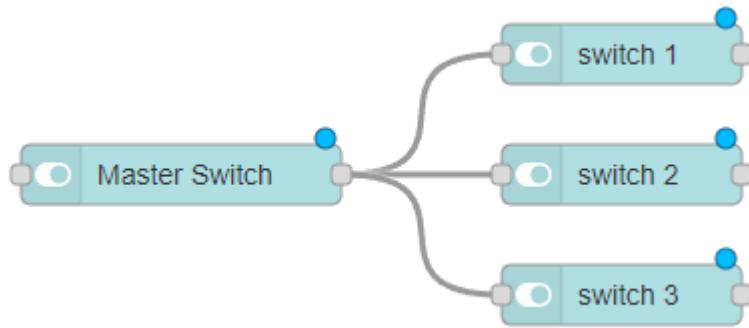
Creating Other Modes

This basic concept can be extended to create what I call modes, for example:

- Morning mode: opens your window blinds and disarms the motion sensor;
- Evening mode: turns on the living room light, kitchen light, and your office light;
- Night/sleep mode: turns off all lights and arms the motion sensor;
- Away mode: turns off all lights and arms the motion sensor.

The Master Switch created previously either turns all GPIOs on or all GPIOs off. What if you want to create a mode that turns some of them on and some of them off? Let's find out how to do that.

Let's go back to the flow created previously in this Unit.



We'll modify it to create a **Night Mode** switch.

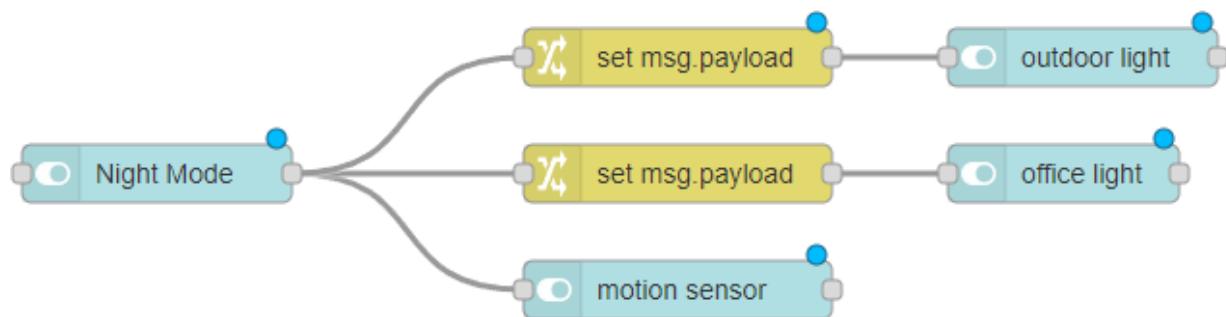
Give different names to the switches to mimic a real-world scenario. For example:

- Master Switch → Night mode
- Switch 1 → outdoor light
- Switch 2 → office light
- Switch 3 → motion sensor

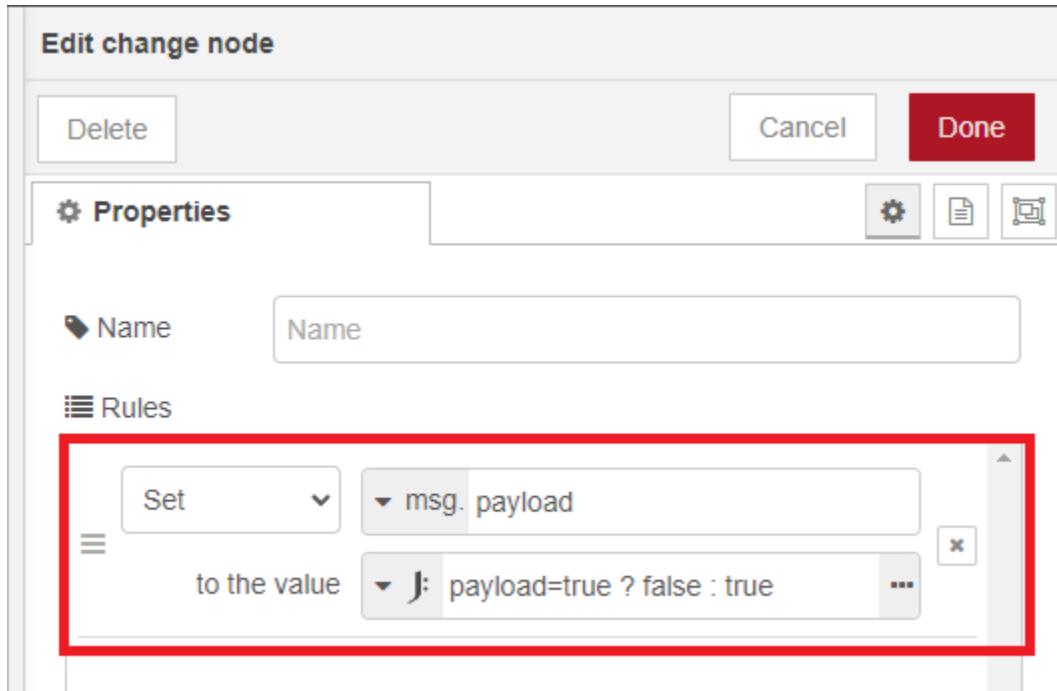
When the Night Mode Switch is on, we want the following states:

- Outdoor light → OFF
- Office light → OFF
- Motion sensor → ON

Drag two **change** nodes and wire them as shown in the following flow.



Double-click on the **change** node to edit its properties (**set msg.payload**) as shown below.



- **set msg.payload**
- to the value: `J: payload=true ? false : true`

Basically, the expression `payload=true ? false : true` means that if the payload is `true`, then the output will be `false`, otherwise it will be `true`.

Double-click on the other **change** node and edit its properties the same way.

Testing the Flow

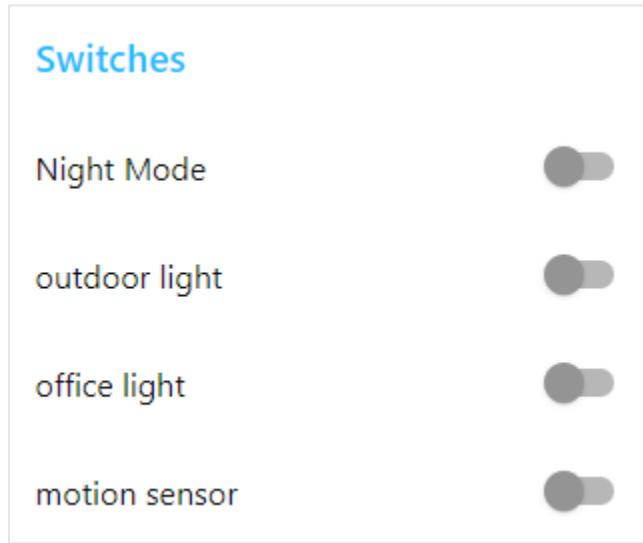
When you're done **Deploy** your application.



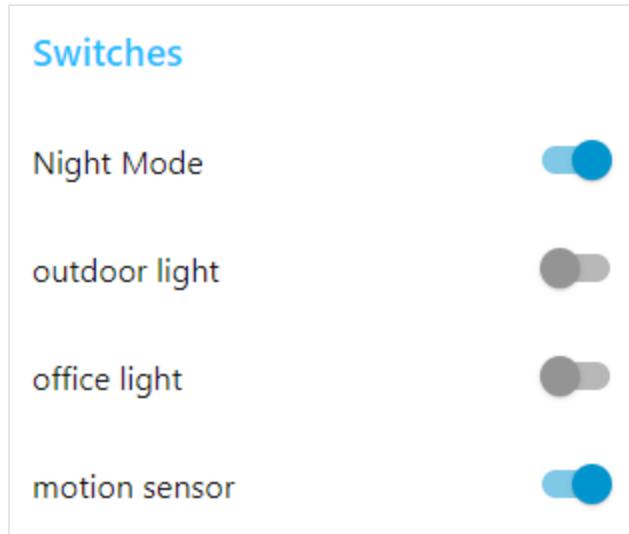
After deploying, go to Node-RED UI:

http://Your_RPi_IP_address:1880/ui

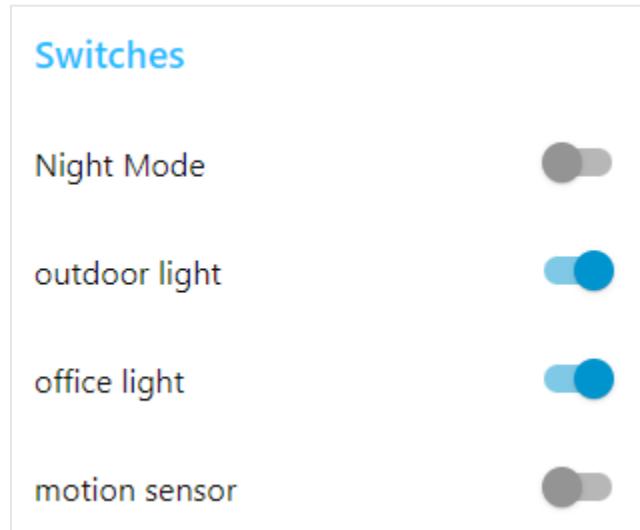
Open the Node-RED UI tab. There should be a group with the night Mode switch.



Click on the **Night Mode** switch. The lights will remain off and the motion sensor will turn on.



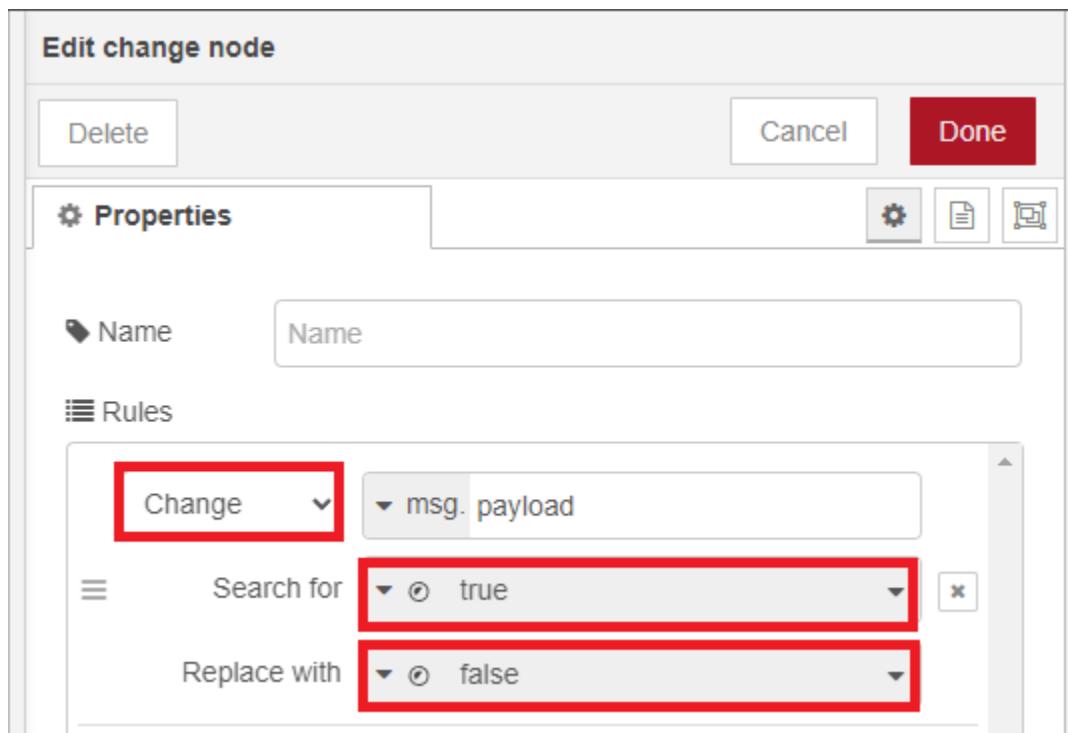
If you turn off the light switch, the lights will turn on and the motion sensor will turn off.



Of course, you can and should change this logic depending on the scenario.

Creating Other Modes (part 2)

For example, imagine you want to turn off the lights when you enable the night mode, but you want them to remain off when you deactivate the night mode. To do that, you can edit the **change** node differently.



- Change `msg.payload`
- Search for: **boolean true**
- Replace with: **boolean false**

Here's what this node does: it searches for a boolean `true` value in the received payload message. If it finds it, it changes the payload message to `false`.

Note: you can add more rules to the switch node. However, it runs all rules sequentially. It doesn't stop after the first match, which I think it's a little counterintuitive.

Edit the other change node in the same way.

Testing the Flow

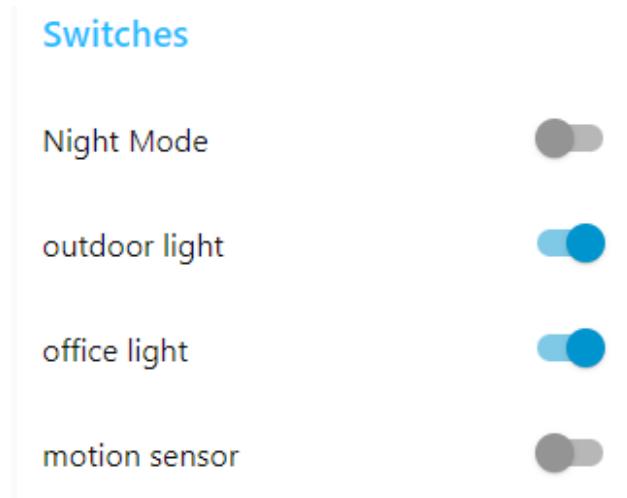
Deploy your application.



After deploying, go to Node-RED UI:

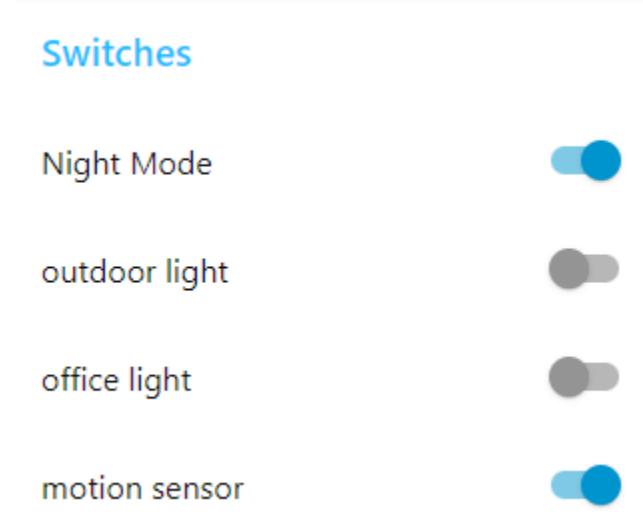
```
http://Your_RPi_IP_address:1880/ui
```

Let's see how it behaves. Start by turning the lights on individually.

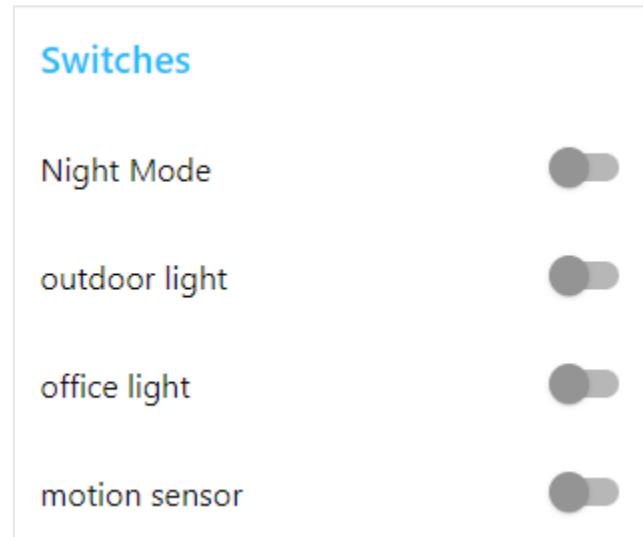


Then, activate and deactivate the night mode.

When you activate the night mode, the lights go off, and the motion sensor goes on.



But, this time, when you deactivate the night mode, the motion sensor goes off and the lights remain off.



As you can see, there are endless possibilities to create different modes.

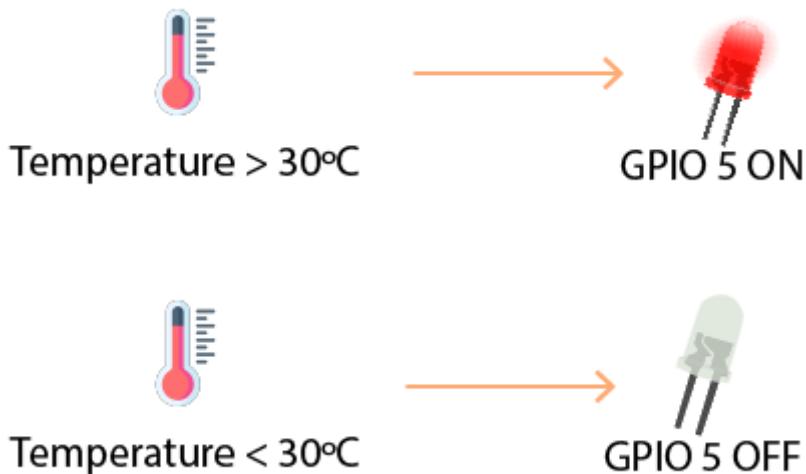
In the next units, you're going to learn how you can trigger these modes or any Node-RED node automatically based on threshold values or time-based events.

8.2 - Triggering Events Based on Threshold Values

This unit shows you how to trigger events based on threshold values. For example, if a sensor reading goes below or above a certain threshold value, it automatically turns your light on or off or sends you a notification.

Project Overview

Keeping the flows that get temperature, humidity, and pressure from previous units, we'll turn on/off a GPIO based on a temperature threshold value.

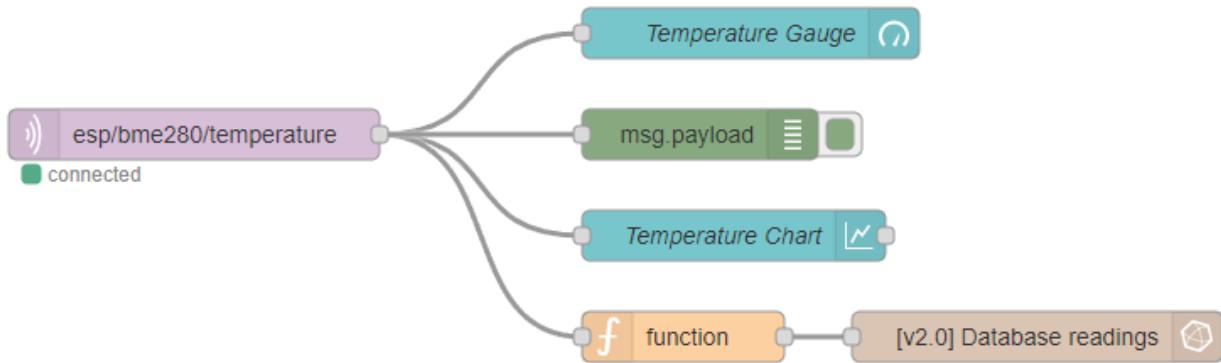


- When the temperature is above 30°C, we'll turn GPIO 5 on;
- When the temperature is below 30°C, We'll turn GPIO 5 off.

You can create a similar logic for the humidity, pressure, or other sensor readings you're using.

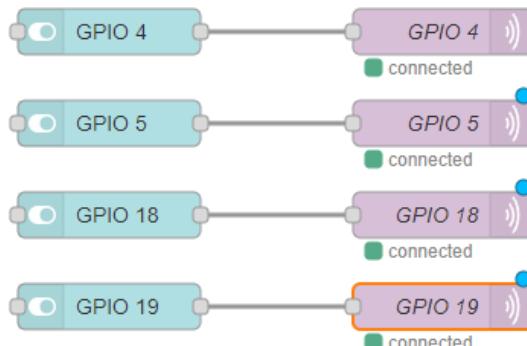
Creating the Flow

If you've followed previous units, you should have a flow that gets temperature readings via MQTT and displays them on a gauge and chart, and saves them on InfluxDB.

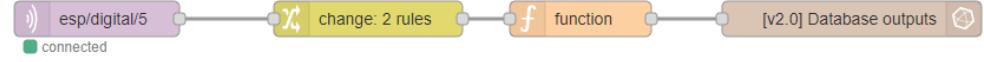
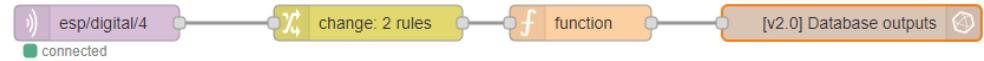


You should also have one of the following flows to control the GPIOs created in previous units.

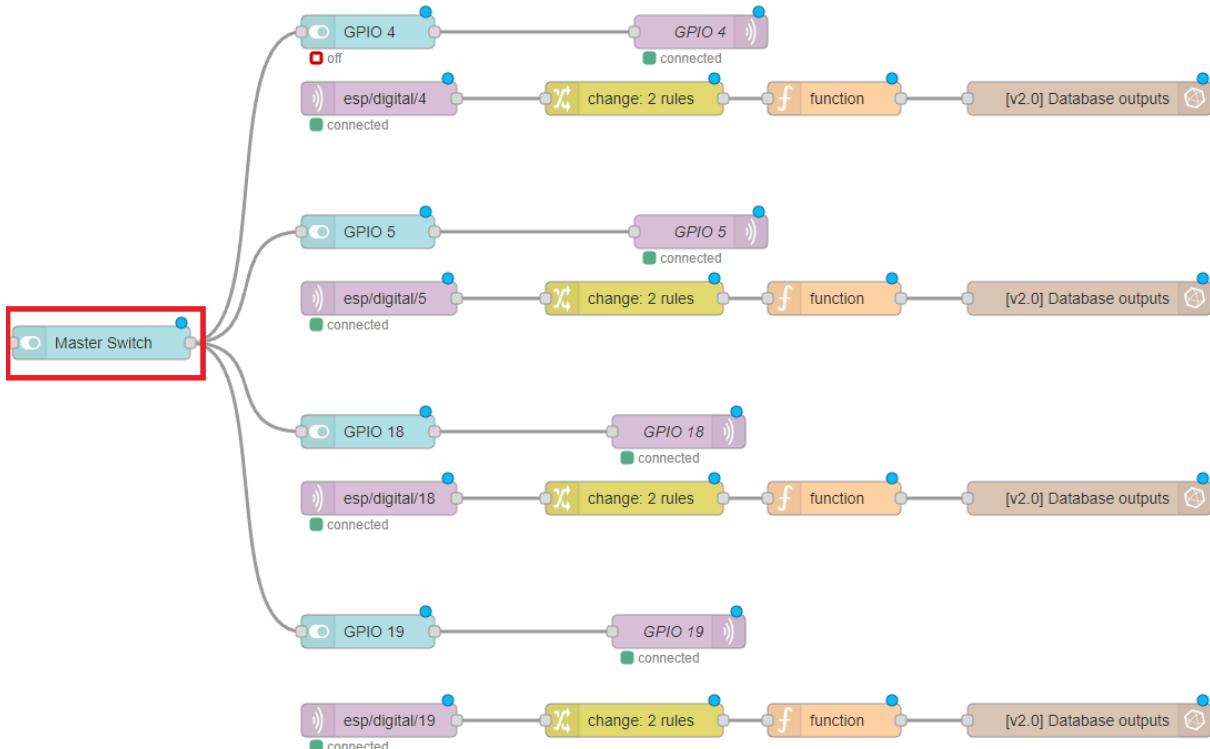
- 1) Basic GPIO control with a single switch:



- 2) GPIO control with InfluxDB integration:

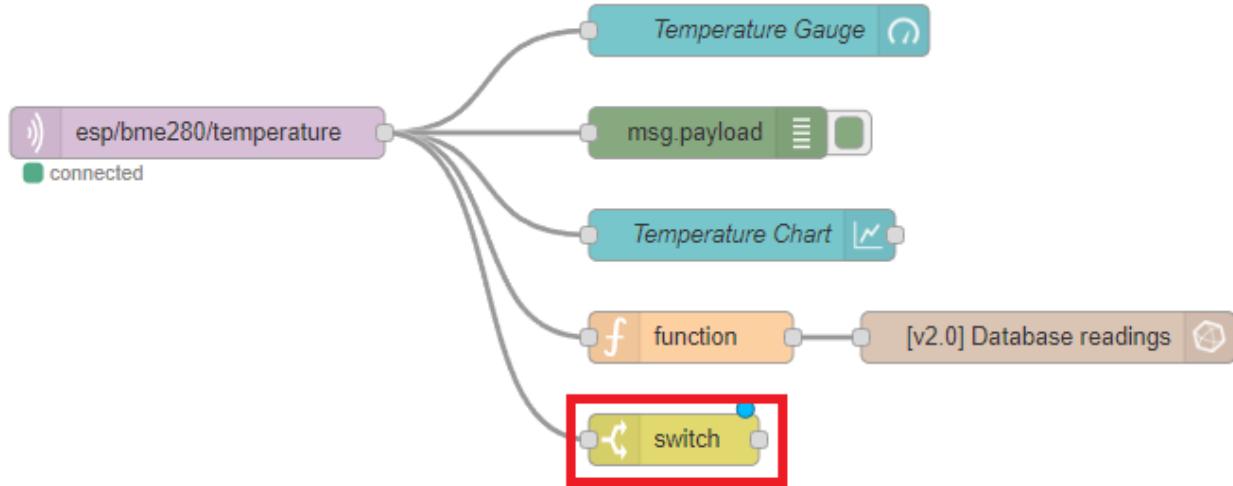


3) Or GPIO control with InfluxDB and Master Switch:



Now, we're going to add some nodes to turn on/off GPIO 5 depending on the temperature value. To do that, we can use a **switch** node and two **change** nodes.

Drag a **switch (function)** node to the flow and wire it to the **MQTT In** node that gets the temperature—take a look at the picture below.



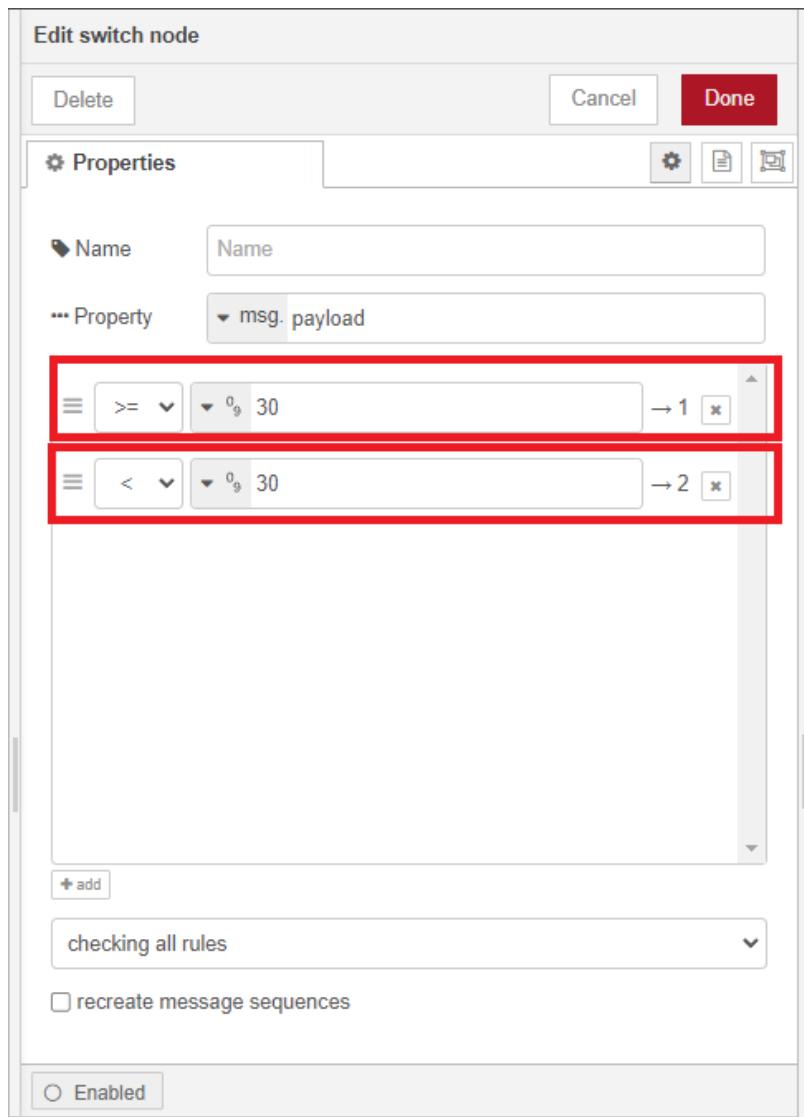
The switch nodes allow us to route messages based on their values. If you read the documentation:

"When a message arrives, the node will evaluate each of the defined rules and forward the message to the corresponding outputs of any matching rules.

Optionally, the node can be set to stop evaluating rules once it finds one that matches. The rules can be evaluated against an individual message property, a flow or global context property, environment variable or the result of a JSON data expression."

Double-click the **switch** node to edit its properties. Then, click on the **+add** button to add another rule. We'll have two rules:

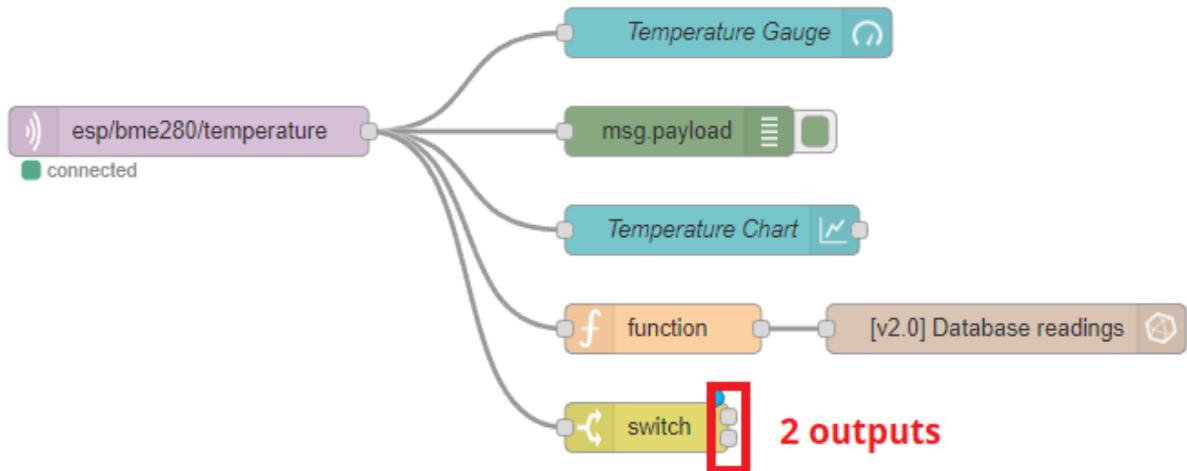
- Temperature above 30°C → turn GPIO off (first output)
- Temperature below 30°C → turn GPIO on (second output)



You can add more rules later on if you want to run specific tasks when the temperature is between different ranges.

When you're finished, click **Done**.

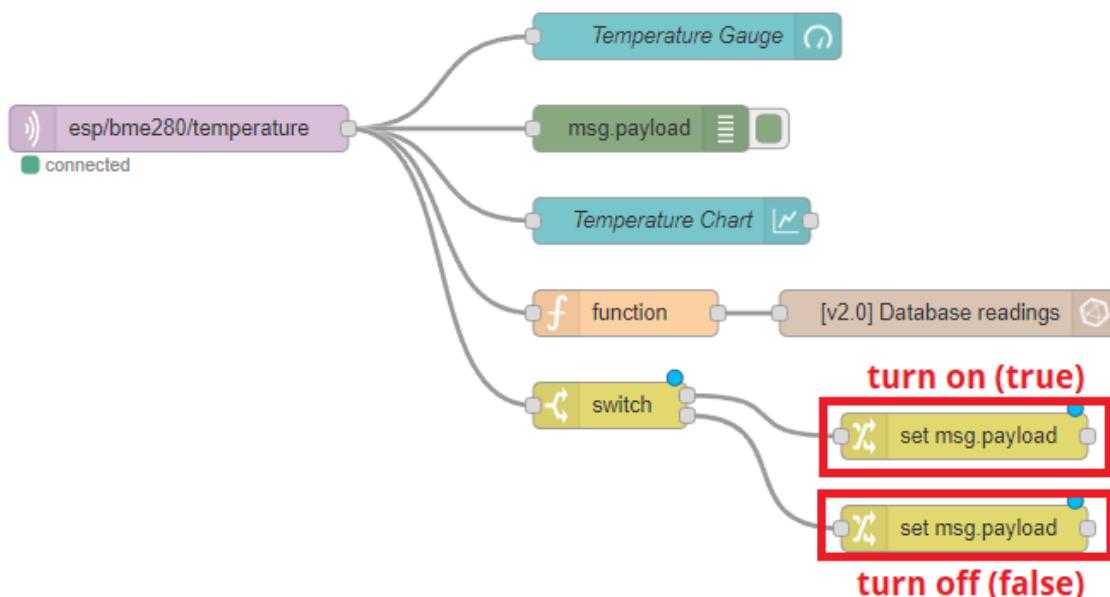
Now, you'll see that the switch node has two outputs.



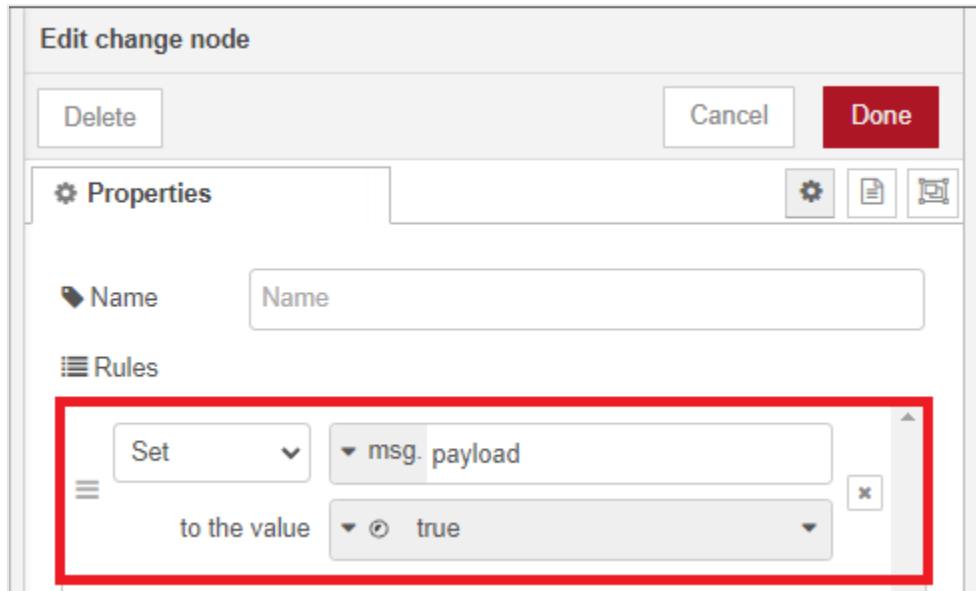
- The message will go through the first output when the temperature is above or equal to 30°C;
- The message will go through the second output when the temperature is below 30°C.

Now, we need to change the message to `true` or `false` and forward it to the **switch** node that controls GPIO 5.

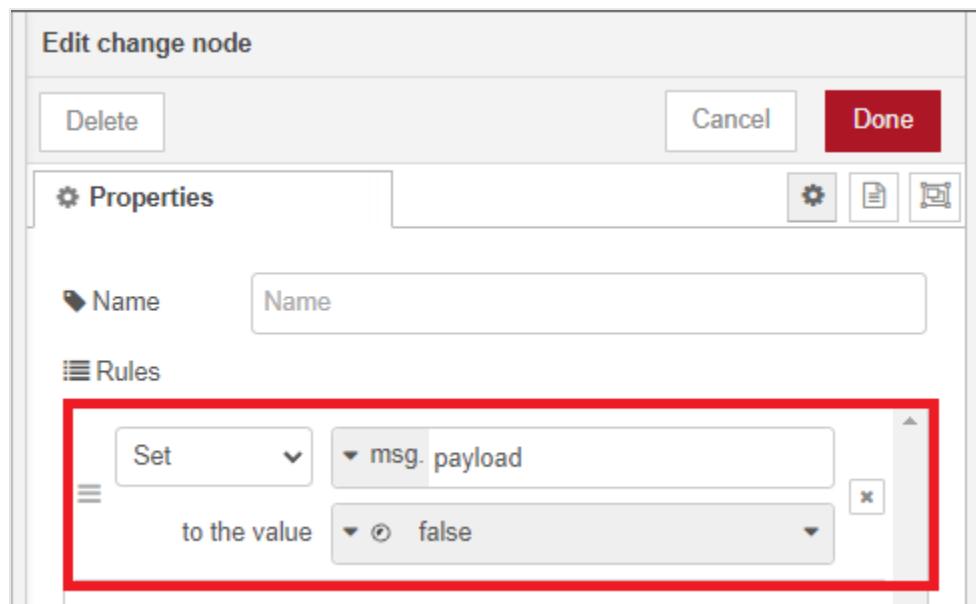
Add two **change** nodes to the flow and wire one to each of the outputs. The first one will send a `true` message to turn the GPIO on, and the second one will send a `false` message to turn the GPIO off.



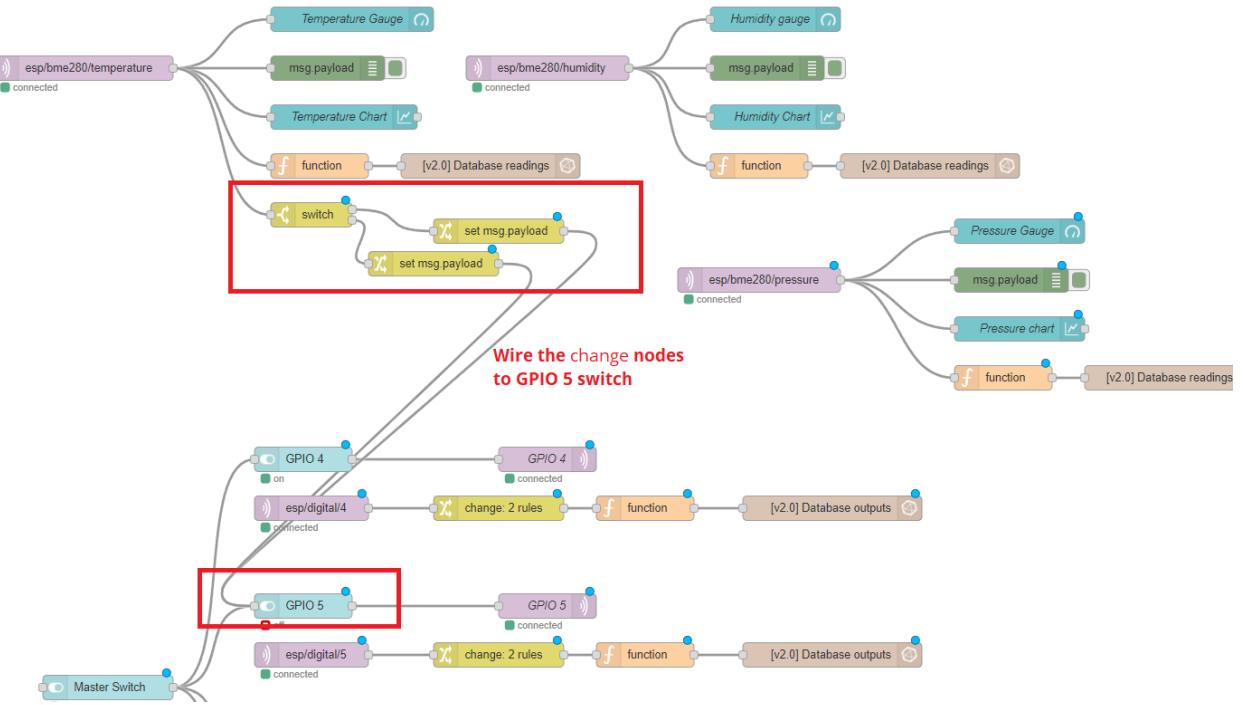
Double-click on the first change node, and edit its properties as follows:



Similarly, edit the other switch node, but this time, change the value to `false`.

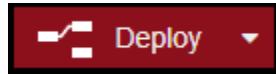


Finally, wire the two **change** nodes to the GPIO **switch** you want to control, in this case, GPIO 5.



Testing the Flow

Deploy your application.



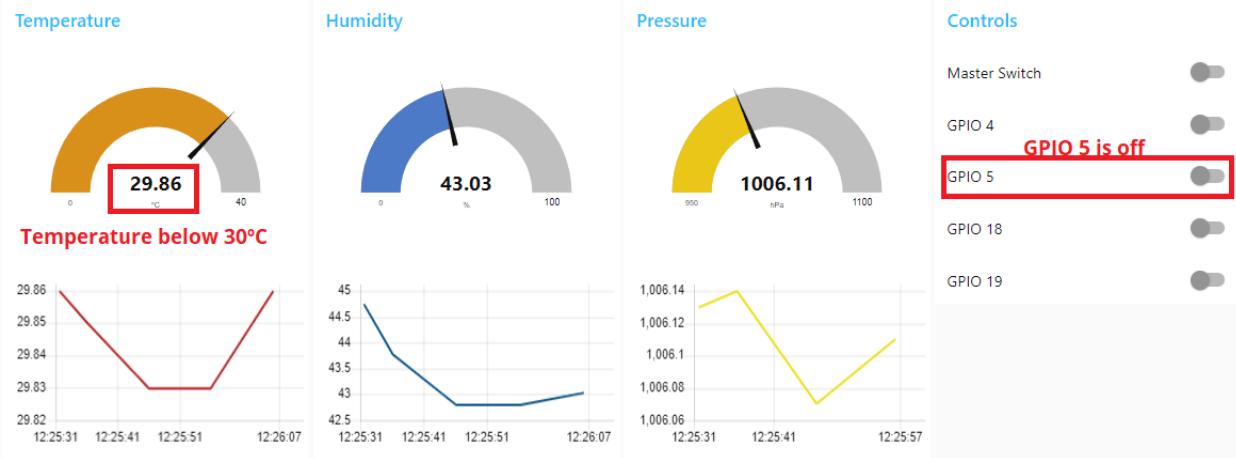
After deploying, go to Node-RED UI:

```
http://Your_RPi_IP_address:1880/ui
```

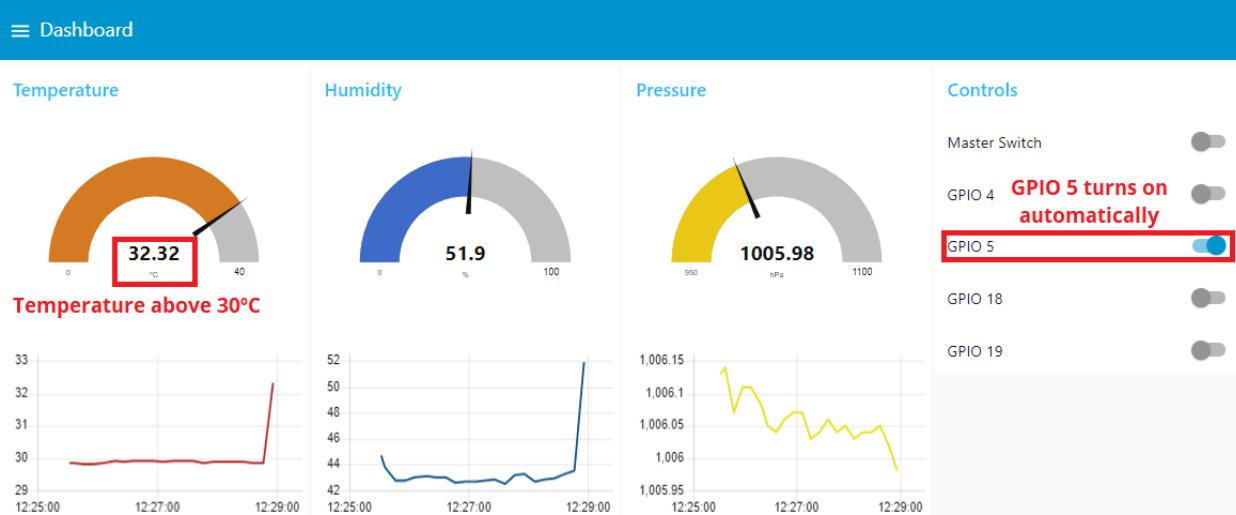
Now, increase/decrease the temperature around the temperature sensor, and see GPIO 5 turning on and off automatically, depending on the temperature value.

At the moment, the temperature is below 30°C, and GPIO 5 is off.

≡ Dashboard



If I touch the temperature sensor to increase its temperature above 30°C, GPIO 5 will automatically turn on.



Wrapping Up

In this unit, you learned how to use the **switch** and **change** nodes to trigger something depending on the temperature reading. You can create similar conditions for other sensor readings and turn on/off GPIOs that make sense for your case scenario. For example:

- Turn on/off lights when luminosity (light sensor) is above or below a certain threshold;
- Turn on/off heating when the temperature is within a determined range of values;
- Turn on/off irrigation when soil moisture is above/below a certain threshold;
- Turn on/off air purifier when air quality is above/below a certain threshold;
- Send notifications when a sensor reading is out of a predefined range;
- Log sensor readings only when these are within a reasonable range of values to exclude outliers;
- And many more applications.

8.3 - Time-based Events

Having a great home automation system means that you should be able to make something happen without having to touch a single button.

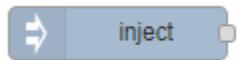
There are certain routines in your home that can be automated. Turning on/off your lights at a certain time, arming/disarming your motion sensor, sending notifications with sensor readings twice a day, etc...

In this unit, you're going to create events that can occur on a certain day at a specific time. You can add time-based triggers to your home automation system to meet your specific needs.

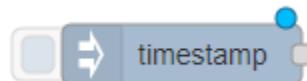
Repeated Events

The **Inject** node has a great feature that we haven't covered yet. It's called "Repeat" and you can set the **Inject** node to be automatically triggered "at a specific time".

Let's take a look at the "Repeat" feature. Drag an **inject** node to the flow:

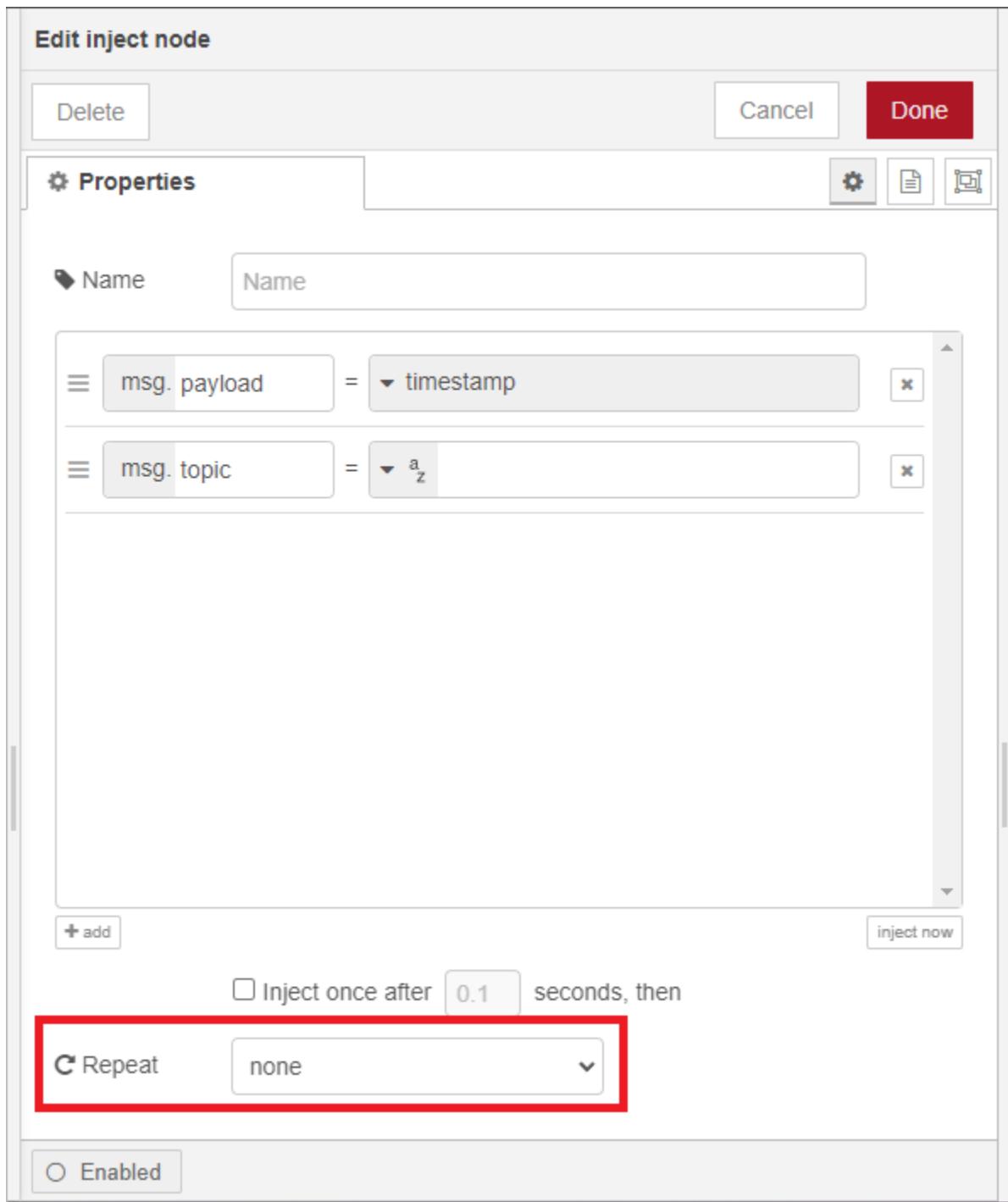


Its name will change to **timestamp** once it's placed on the flow.

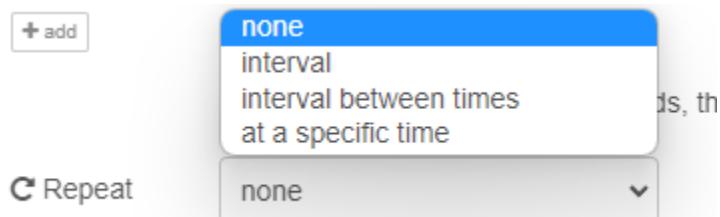


Double-click on the node to explore its properties.

At the bottom, there is a **Repeat** feature that is set to **none** by default.



If you click on the **Repeat** drop-down menu, you'll see several options:



- **Interval:** it will trigger every X seconds or minutes or hours. This can be useful, for example, to trigger notifications with sensor readings every two hours.

C Repeat

interval

every 1 seconds

- **Interval between times:** it will trigger every X minutes between a determined period. This is useful for example, if you want to trigger notifications with sensor readings but only during the day and excluding weekends.

C Repeat

interval between times

every 60 minutes

between 08:00 and 22:00

on Monday Tuesday Wednesday
 Thursday Friday Saturday
 Sunday

- **At a specific time:** select a specific time and days of the week to trigger an event.

C Repeat

at a specific time

at 12:00

on Monday Tuesday Wednesday
 Thursday Friday Saturday
 Sunday

Project Overview

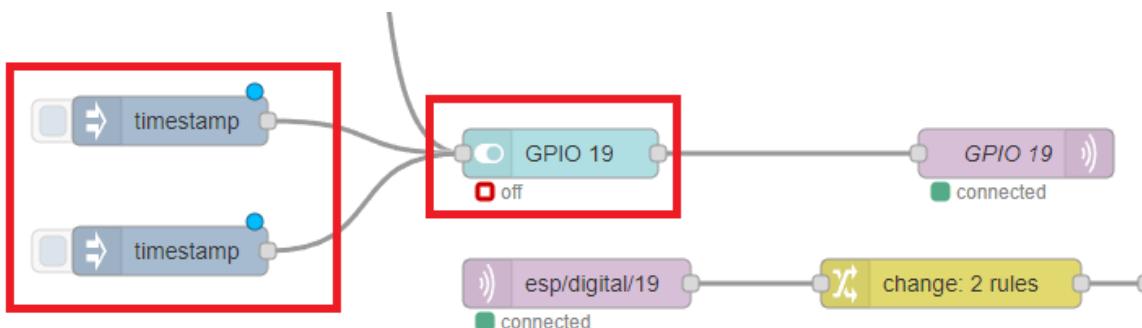
To show you how you can use the **inject** nodes **Repeat** property on your flow, we'll create the following automation:

- GPIO 19 turns on every day at 19:00 (7:00PM);
- GPIO 19 turns off every day at 8:00 (8:00 AM).

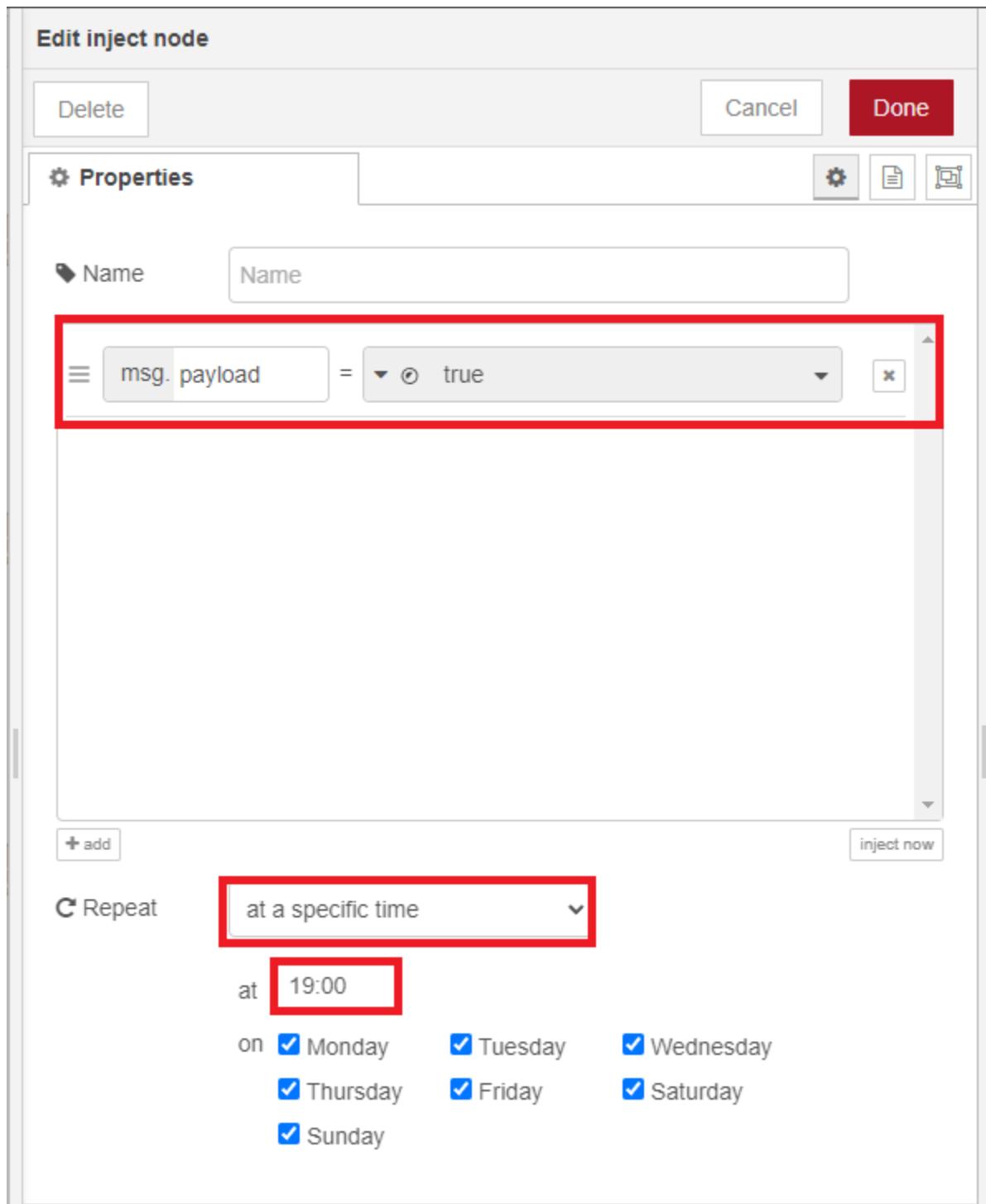
You should have a flow with a **switch** for GPIO19 from previous units.

Creating Your Flow

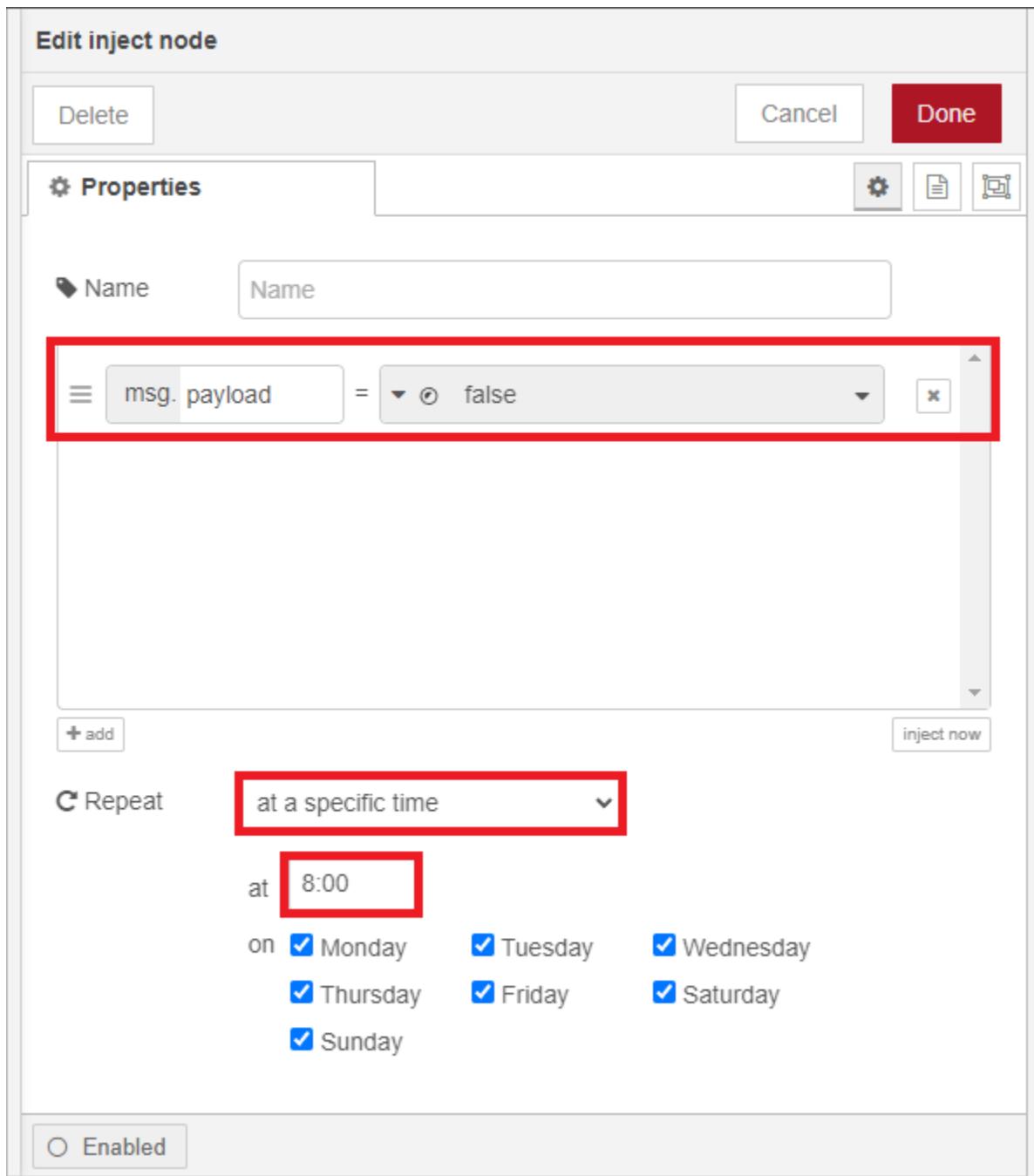
Drag 2 **Inject** nodes to the flow and wire them to GPIO 19 **switch**.



Let's edit the first inject node to turn GPIO 19 on (send a **true** boolean message) every day at 19:00.

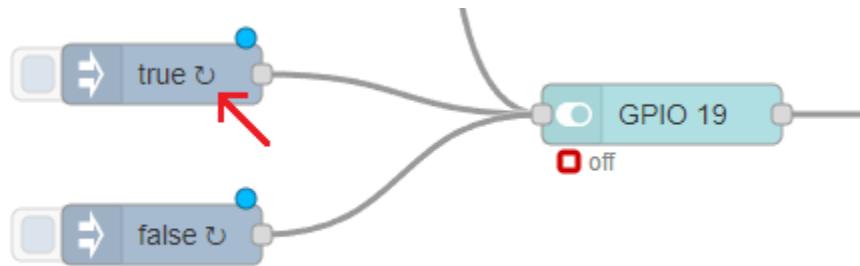


Now, edit the other **inject** node to send a **false** boolean message every day at 8:00.



When you're finished, click on **Done**.

When you use the **Repeat** property, the **inject** nodes will appear with a looping arrow on the flow.



Testing the Flow

Deploy your application.



GPIO 19 will turn on at 8:00 AM and turn off at 7:00 PM. If you want to see if it is working as expected, but it's still a long time to 8:00 AM or 7:00 PM, change the time in your nodes.

GPIO 19 should turn on and off at the specified times.

This same concept can be applied to:

- Control outlets at a certain day and hour;
- Arm or disarm sensors at a certain hour every day;
- Trigger notifications or reminders;
- Control master switches;
- Anything that is time-based.

The **inject** node **Repeat** property works great for many scenarios, but if you want even more customization options, there's a node called **Big Timer** with many more options.

8.4 - Time-based Events with Big Timer

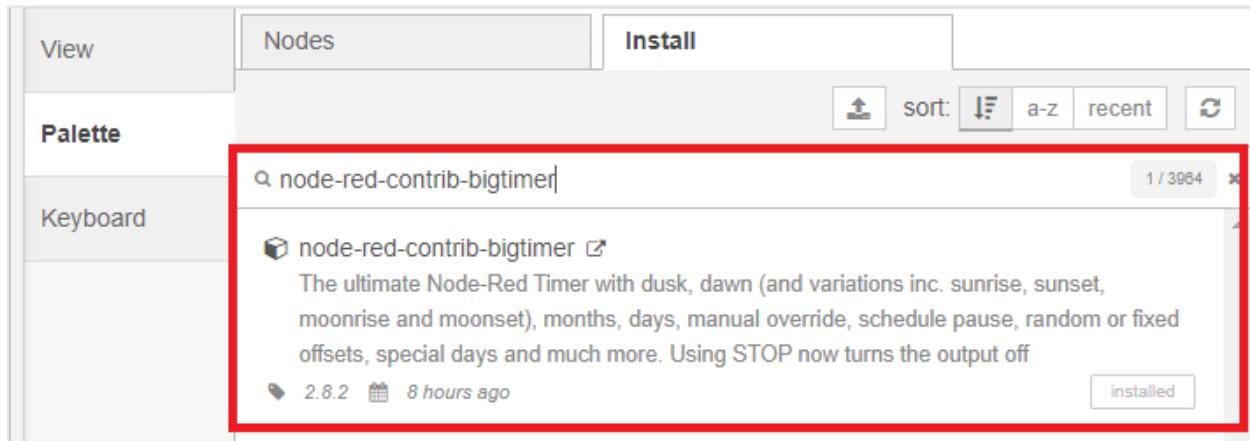
The Big Timer node for Node-RED provides a huge range of timing facilities that will take your automation to the next level. For example, you don't want your outside lights to turn on at 6 PM, you want to turn them on when it gets dark. The Big Timer node can use your latitude and longitude to estimate sunset and sunrise times and turn on/off the light accordingly.

Installing the Big Timer Node

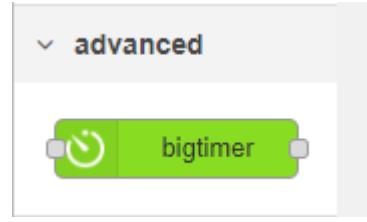
The Big Timer Node created by Peter Scargill is a powerful and easy-to-use timing node for Node-RED. We recommend taking a look at the Big Timer page on Peter Scargill's blog:

- [Big Timer page](#)

Installing the Big Timer node is very straightforward. Go to the main menu on the top right corner and click on **Manage Palette**. Then, click on the **Install** tab and search for `node-red-contrib-bigtimer`. Install that node.



A new node called **bigtimer** will show up on your palette.



Drag the **bigtimer** node to the flow. You'll notice it comes with an input and three outputs.



The documentation clearly explains what each output consists of.

"BigTimer has 3 outputs, the first of which triggers when there is a change of state and presents one of two messages (for, for example, MQTT or other control mechanism), the second output has a topic of "status" and contains a simple 1 or 0 every minute in the payload and also has additional outputs reflecting the status message in msg.state and message time and others. The third output presents a message which could be used for speech or debugging."

The input can be used to override the schedule properties. You can check the valid input commands on the [documentation page](#).

Double-click on the **BigTimer** node to explore its properties.

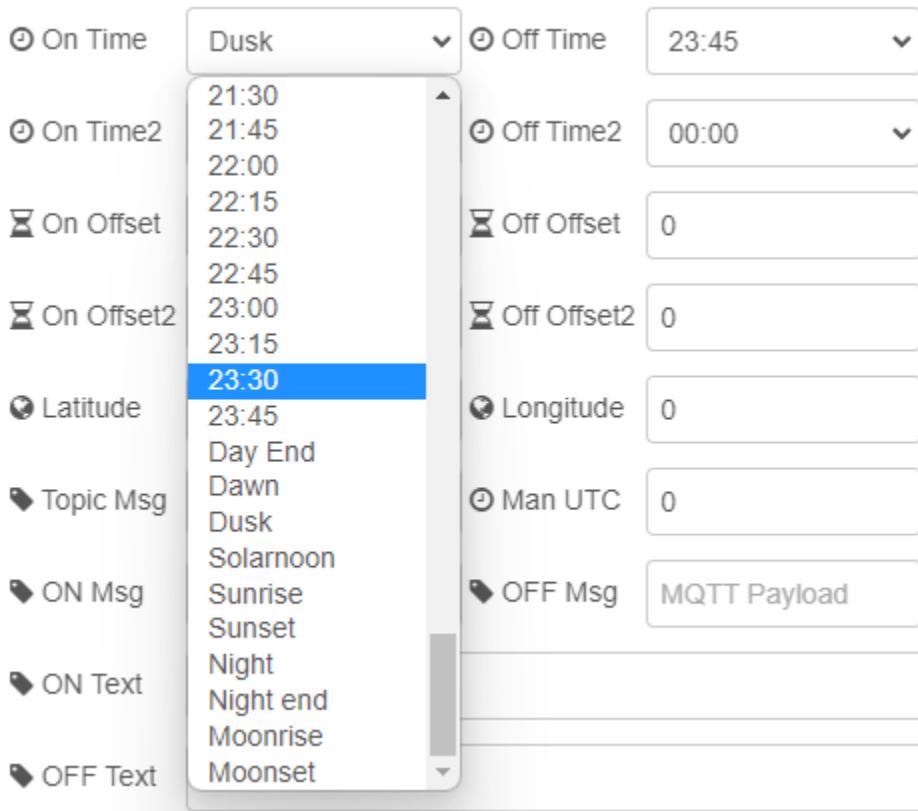
Edit bigtimer node

Delete
Cancel
Done

Properties

| | | | |
|--|--------------|---|--------------|
| ◆ Name | Big Timer | | |
| ◆ Comment | | | |
| <input type="radio"/> On Time | Dusk | <input type="radio"/> Off Time | 23:45 |
| <input type="radio"/> On Time2 | 00:00 | <input type="radio"/> Off Time2 | 00:00 |
| <input checked="" type="checkbox"/> On Offset | 0 | <input checked="" type="checkbox"/> Off Offset | 0 |
| <input checked="" type="checkbox"/> On Offset2 | 0 | <input checked="" type="checkbox"/> Off Offset2 | 0 |
| <input type="radio"/> Latitude | 0 | <input type="radio"/> Longitude | 0 |
| ◆ Topic Msg | MQTT Topic | <input type="radio"/> Man UTC | 0 |
| ◆ ON Msg | MQTT Payload | ◆ OFF Msg | MQTT Payload |
| ◆ ON Text | Text | | |
| ◆ OFF Text | Text | | |
| <input checked="" type="checkbox"/> Timeout | 1440 | | |

You have the option to set two **On Time** schedules and two **Off Time** schedules. You can select specific hours, or you can select: day end, dawn, dusk, solarnoon, sunrise, sunset, night, night end, moonrise, and moonset.



You can adjust the time using the **Offset** fields to set a trigger, for example, 15 minutes before sunrise.

The first output of the big timer can be a message for an MQTT node or for any other purpose. If you'll output the message for an MQTT node, you can fill the MQTT topic and **On Msg** and **Off Msg** fields.

For precise sunrise, sunset, etc, you should include your latitude and longitude location.

There's also the option to include up to 12 special days of the year (for example, Christmas), and up to 6 special weekdays of the month.

Include special days of the year (optional) day=1-31 month=1-12

| | | | |
|---------------------------------|---|-----------------------------------|---|
| <input type="checkbox"/> Day 1 | 0 | <input type="checkbox"/> Month 1 | 0 |
| <input type="checkbox"/> Day 2 | 0 | <input type="checkbox"/> Month 2 | 0 |
| <input type="checkbox"/> Day 3 | 0 | <input type="checkbox"/> Month 3 | 0 |
| <input type="checkbox"/> Day 4 | 0 | <input type="checkbox"/> Month 4 | 0 |
| <input type="checkbox"/> Day 5 | 0 | <input type="checkbox"/> Month 5 | 0 |
| <input type="checkbox"/> Day 6 | 0 | <input type="checkbox"/> Month 6 | 0 |
| <input type="checkbox"/> Day 7 | 0 | <input type="checkbox"/> Month 7 | 0 |
| <input type="checkbox"/> Day 8 | 0 | <input type="checkbox"/> Month 8 | 0 |
| <input type="checkbox"/> Day 9 | 0 | <input type="checkbox"/> Month 9 | 0 |
| <input type="checkbox"/> Day 10 | 0 | <input type="checkbox"/> Month 10 | 0 |
| <input type="checkbox"/> Day 11 | 0 | <input type="checkbox"/> Month 11 | 0 |
| <input type="checkbox"/> Day 12 | 0 | <input type="checkbox"/> Month 12 | 0 |

Similarly, you can exclude special days of the year and special days of the month.

Include special weekdays of the month (optional) Sun=1 etc. Week=1-5

| | | | |
|-------|---|--------|---|
| Day 1 | 0 | Week 1 | 0 |
| Day 2 | 0 | Week 2 | 0 |
| Day 3 | 0 | Week 3 | 0 |
| Day 4 | 0 | Week 4 | 0 |
| Day 5 | 0 | Week 5 | 0 |
| Day 6 | 0 | Week 6 | 0 |

Exclude special days of the year (optional) day=1-31 month=1-12

| | | | |
|-------|---|---------|---|
| Day 1 | 0 | Month 1 | 0 |
| Day 2 | 0 | Month 2 | 0 |
| Day 3 | 0 | Month 3 | 0 |
| Day 4 | 0 | Month 4 | 0 |
| Day 5 | 0 | Month 5 | 0 |
| Day 6 | 0 | Month 6 | 0 |
| Day 7 | 0 | Month 7 | 0 |
| Day 8 | 0 | Month 8 | 0 |

Finally, you also have all these other options to include/exclude specific months or days of the week, exclude even or odd days, etc.

Exclude special weekdays of the month (optional) Sun=1 etc. Week=1-5

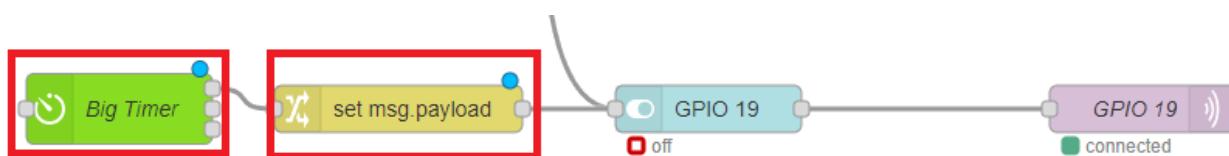
| | | | |
|---------|---|----------|---|
| 📅 Day 1 | 0 | 📅 Week 1 | 0 |
| 📅 Day 2 | 0 | 📅 Week 2 | 0 |
| 📅 Day 3 | 0 | 📅 Week 3 | 0 |
| 📅 Day 4 | 0 | 📅 Week 4 | 0 |
| 📅 Day 5 | 0 | 📅 Week 5 | 0 |
| 📅 Day 6 | 0 | 📅 Week 6 | 0 |

- Sun Mon Tue Wed Thu Fri Sat
 Jan Feb Mar Apr May Jun
 Jul Aug Sep Oct Nov Dec
- Suspend schedule Randomise all offsets
 Randomise On 1 Randomise Off 1
 Randomise On 2 Randomise Off 2
 BAN Odd days BAN Even days
 Repeat output Output at startup

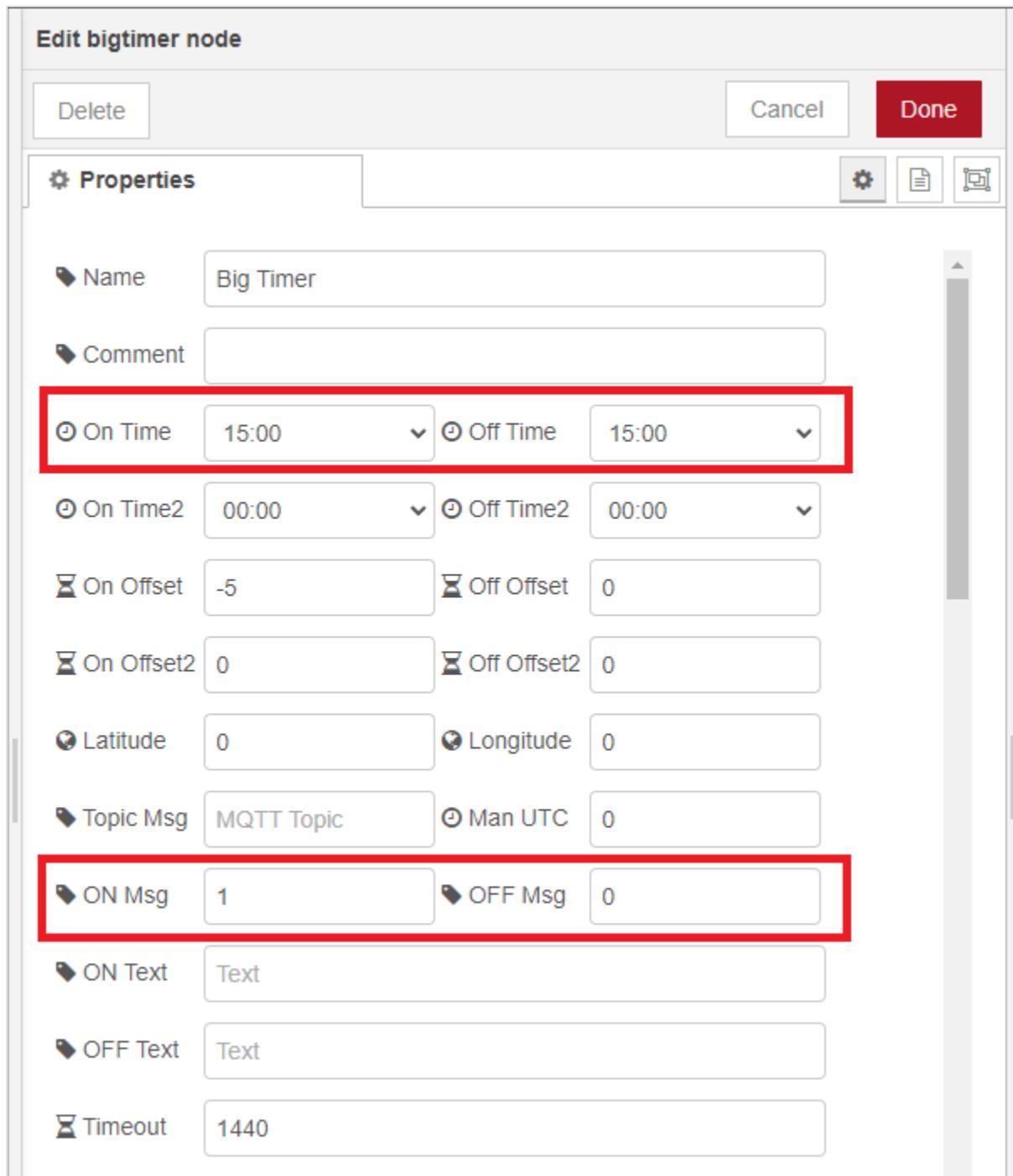
Example Flow

For you to have an idea of how to use the **Big Timer** to control events, we'll create a simple flow that will turn GPIO 19 on and off at specific times. You can then, apply the concept to other tasks you want to schedule. We also recommend taking a look at the [official Big Timer page](#) to explore all the awesome automation you can do with this node.

Drag a **Big Timer** node to the flow. Add a **change** node and wire it to the first output of the **Big Timer** node. Then, wire the change node to GPIO 19 **switch**.



Double-click on the **Big Timer** node to edit its properties. Fill the **On Time** and **Off Time** fields with the time you want to turn GPIO 19 on and off. For demonstration purposes, we recommend selecting something close to the current time so that you can easily see if it is working as expected. Use the offset times if needed.

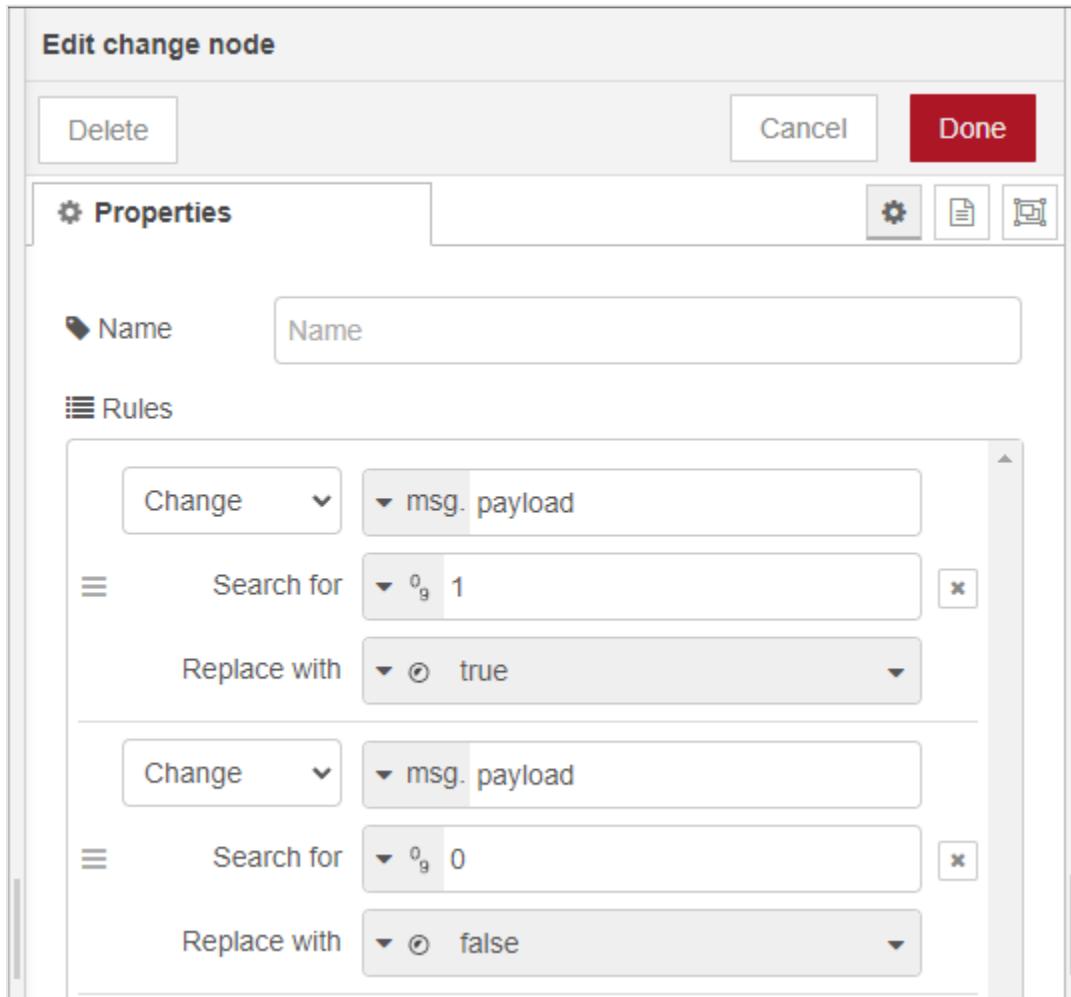


The on message payload and off message payload should be filled on the **On Msg** and **Off Msg** fields. The **On Text** and **Off Text** fields are optional.

With the current options, the node will send a payload of **1** when it reaches the **On Time**, and a payload of **0** when it reaches the **off time**. You can then, try the other options to create more complex schedules.

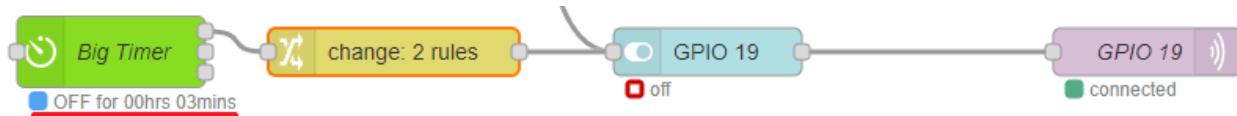
To control the switch, we must send a **true** to turn the switch on, or a **false** to turn the switch off.

We can use a **change** node to replace the **0** with a **false**, and the **1** with a **true**. Double-click on the **change** node and change its properties as shown in the following picture.



When you're done deploy your application.

After deploying you'll see a timer below the **Big Timer** node showing how long it will be on or off.



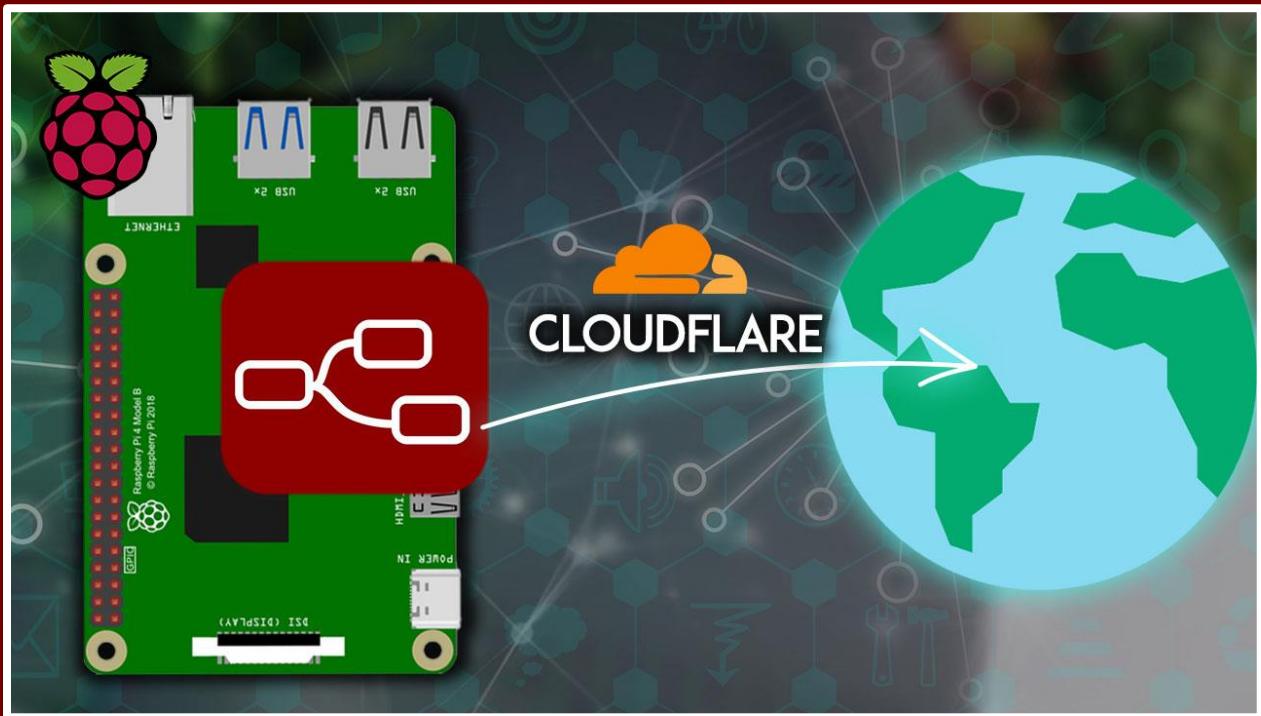
If everything is working as expected GPIO 19 should turn on and off at the times you designated.

Wrapping Up

This was just a simple example showing you how to use Big Timer. There are much more features you can use. You can also use the input to override the current on/off state, enable manual/auto mode, and stop the schedule. You'll find lots of information and use case examples [here](#).

MODULE 9

Accessing Your Local System from Anywhere



This Module explains how to set up a Cloudflare tunnel to access your Node-RED home automation system and InfluxDB monitoring dashboards from anywhere.

9.1 - Accessing Your System from Anywhere

At the moment, to control and monitor your devices using Node-RED or InfluxDB, you need a browser connected to your local network. The system is not accessible outside your house.

There are many different ways to access your local web server from anywhere. One of the most “popular” options include opening a port on your router. I don’t recommend this method, unless you know exactly what you’re doing and you understand the risks of having an open port to your local network and how to avoid security issues.

Cloudflare Tunnel

Luckily, nowadays there are secure and reliable solutions that are easy to set up, like setting up a Cloudflare Tunnel. Learn more about Cloudflare Tunnel on the following link: <https://www.cloudflare.com/products/tunnel/>

“Cloudflare Tunnel is tunneling software that lets you quickly secure and encrypt application traffic to any type of infrastructure, so you can hide your web server IP addresses, block direct attacks, and get back to delivering great applications.”

Here’s how it works:

“The Tunnel daemon creates an encrypted tunnel between your origin web server and Cloudflare’s nearest data center, all without opening any public inbound ports. After locking down all origin server ports and protocols using your firewall, any requests on HTTP/S ports are dropped, including volumetric DDoS attacks. Data breach attempts — such as snooping of data in transit or brute force login attacks — are blocked entirely.”

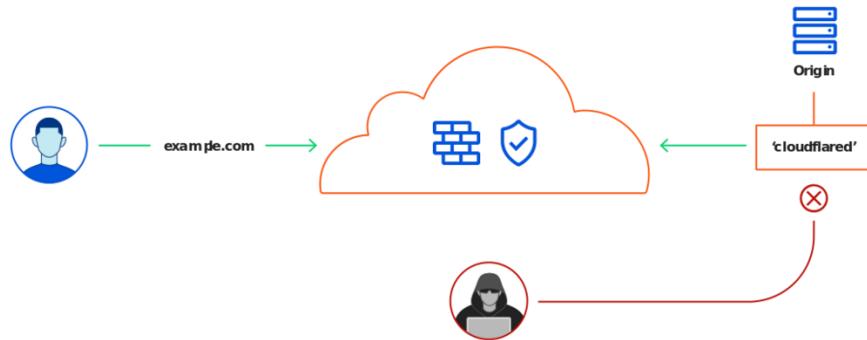


Image source: <https://www.cloudflare.com/products/tunnel/>

"Tunnel allows you to quickly deploy infrastructure in a Zero Trust environment, so all requests to your resources first pass through Cloudflare's robust security filters."

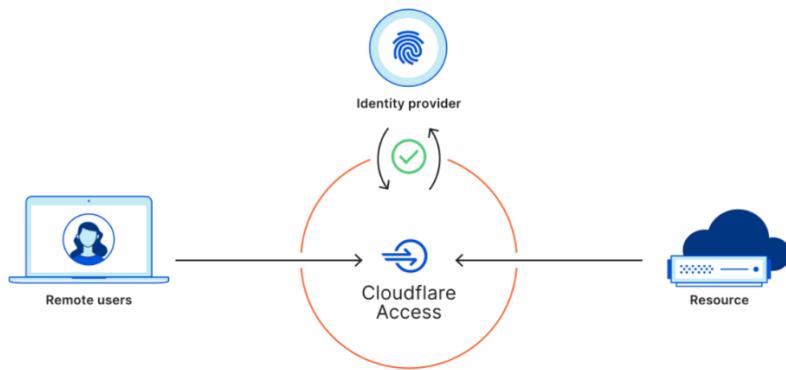
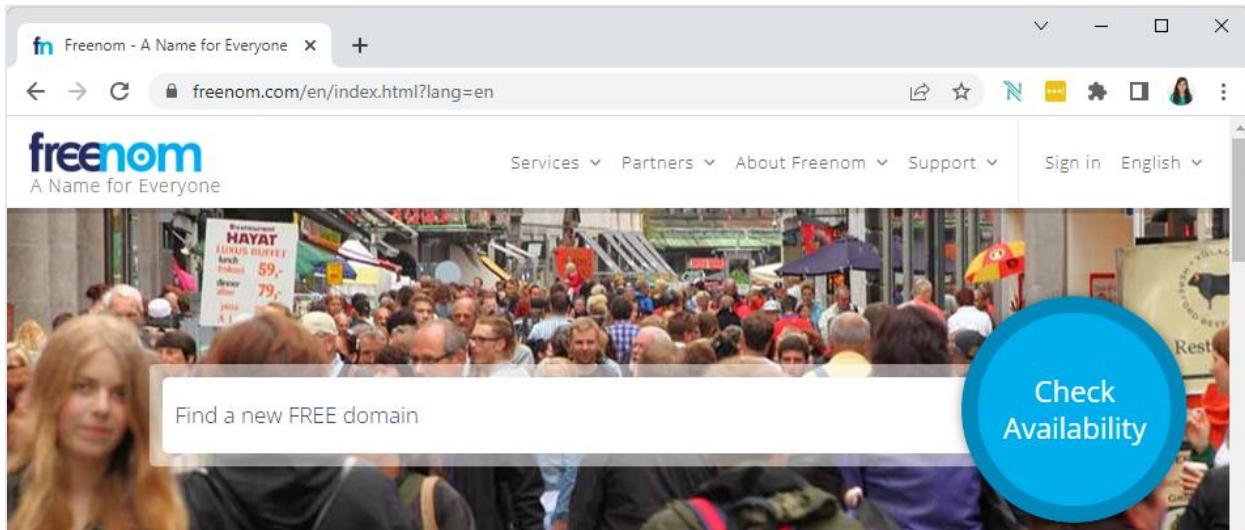


Image source: <https://www.cloudflare.com/products/tunnel/>

Getting a Domain Name

Cloudflare Tunnel creates a tunnel to your local web server so that you can access it from anywhere using a domain name. So, to create the tunnel you need to get a domain name.

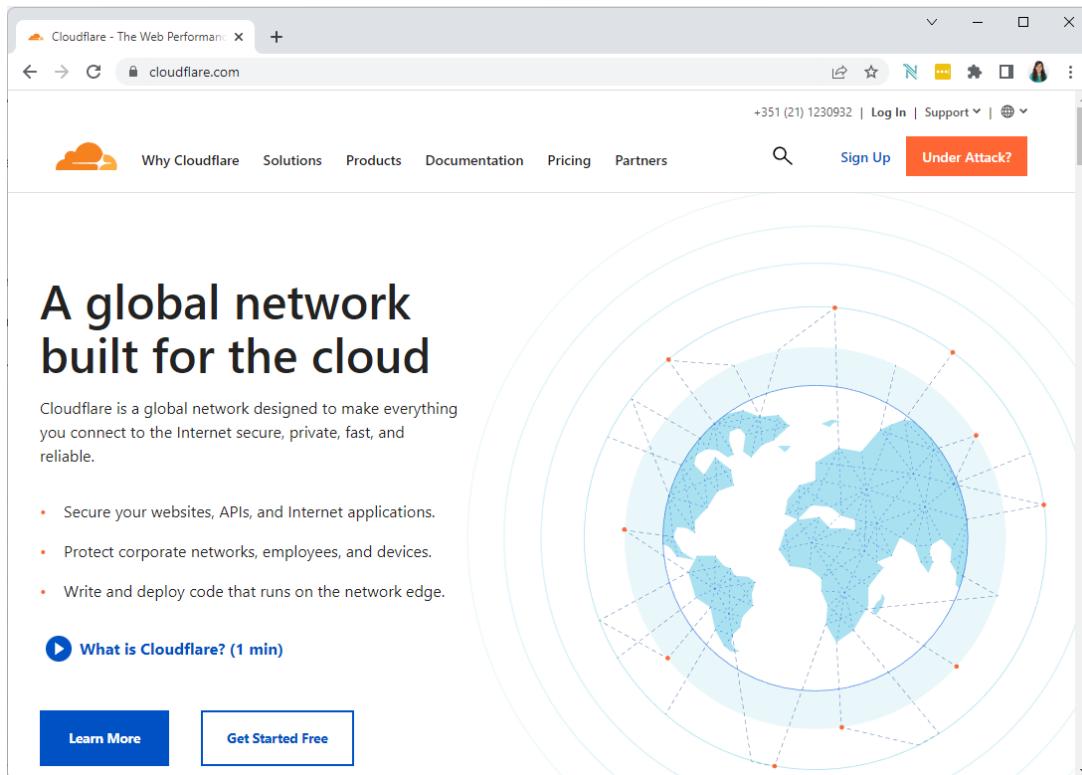
We'll use a domain name from [GoDaddy](#), but any domain name provider should work ([like Namecheap for example](#)). You can also grab a free domain name at [freenom.com](#). You can search and register a free domain name with several extensions (.tk, .ml, .ga, .cf, .gq).



Please note that at the time of updating this eBook, freenom is no longer accepting new members, but that can change in the future.

Creating a Cloudflare Tunnel

Go to <https://www.cloudflare.com/> and create a free account.



Click on **+Add site** or go to <https://dash.cloudflare.com/> and enter your domain name on the field. Ours is `smarthomecourse.com`. Then, click on **Add site**.

We're using a domain from GoDaddy, but if you're using Namecheap you can follow the [official instructions here](#).

The screenshot shows the Cloudflare dashboard at <https://dash.cloudflare.com/add-site>. On the left, there's a sidebar with options like Websites, Domain Registration, and Analytics. The main area has a heading "Accelerate and protect your site with Cloudflare". Below it, there's a text input field labeled "Enter your site (example.com):" containing "smarthomecourse.com". To the right of the input field is a blue "Add site" button, which is also highlighted with a red box. In the top right corner of the main area, there's a "Support" dropdown and a language setting for "English (US)". A purple sidebar on the right says "Register a new domain with Cloudflare" with a "New" button.

Choose the **Free plan** at the bottom of the page. Then, click on **Continue**.

The screenshot shows the "Select a plan" page. On the left, there's a sidebar with various service options like Overview, Analytics, DNS, Email, SSL/TLS, Security, Access, Speed, Caching, Workers, Rules, Network, Traffic, Custom Pages, Apps, Scrape Shield, Zaraz, and Web3. The main area has three plan options: "Pro", "Business", and "Enterprise". Each plan has a price, billing information, and a list of core features. At the bottom left, there's a section titled "Not sure where to start? Get started for free." with a "Free" plan option showing "\$0". At the bottom center, there's a "Continue" button, which is highlighted with a red box. A red box also surrounds the entire bottom section of the page.

After that, click on the **Continue** button again (you don't need to fill any of those fields the moment).

The screenshot shows the Cloudflare dashboard for the domain `smarthomecourse.com`. The left sidebar contains navigation links: Overview, Analytics, DNS, Email (Beta), SSL/TLS, Security, Access, Speed, Caching, Workers, Rules, and Network. The main content area is titled "Add more DNS records for smarthomecourse.com" and includes instructions for adding proxy and non-proxy DNS records. It features a table for managing existing DNS records with columns for Type, Name, Content, Proxy status, TTL, and Actions. A "Continue" button is at the bottom.

At this moment, you need to change your nameservers. You must proceed as explained on the Cloudflare instructions. You need to login into your domain name provider, find the DNS management settings and replace the referred nameservers with Cloudflare's nameservers.

The screenshot shows the Cloudflare dashboard for the domain 'smarthomecourse.com'. The left sidebar contains various management options like Overview, Analytics, DNS, Email, SSL/TLS, Security, Access, Speed, Caching, Workers, Rules, Network, Traffic, Custom Pages, Apps, Scrape Shield, Zaraz, and Web3. The main content area is titled 'Change your nameservers' and includes a note about pointing to Cloudflare's nameservers. It provides a step-by-step guide: 1. Log in to your GoDaddy account, 2. Remove the following nameservers (ns51.domaincontrol.com, ns52.domaincontrol.com), 3. Add Cloudflare's nameservers (sean.ns.cloudflare.com, shaz.ns.cloudflare.com), and 4. Save your changes. A note states that registrars can take up to 24 hours to process updates. A 'Done, check nameservers' button is at the bottom.

For example, in GoDaddy the *Edit nameservers* menu looks as follows.

The screenshot shows the 'Edit nameservers' interface in GoDaddy. The title is 'Edit nameservers' and the sub-section is 'Introduce my own nameservers'. A warning message says 'Changing nameservers is risky and can cause your website to no longer be publicly visible.' Below are two input fields containing 'sean.ns.cloudflare.com' and 'shaz.ns.cloudflare.com'. At the bottom, there is a link '+ Add name server' and three buttons: 'Cancel', 'Go back', and a prominent black 'Save' button.

Save the changes.

You need to wait 15 minutes or more for the domain nameserver to propagate. On the Cloudflare dashboard click on **Done, check nameservers**.

When it's done, under the **Access** menu (left sidebar), click on launch the **Zero Trust Dashboard**.

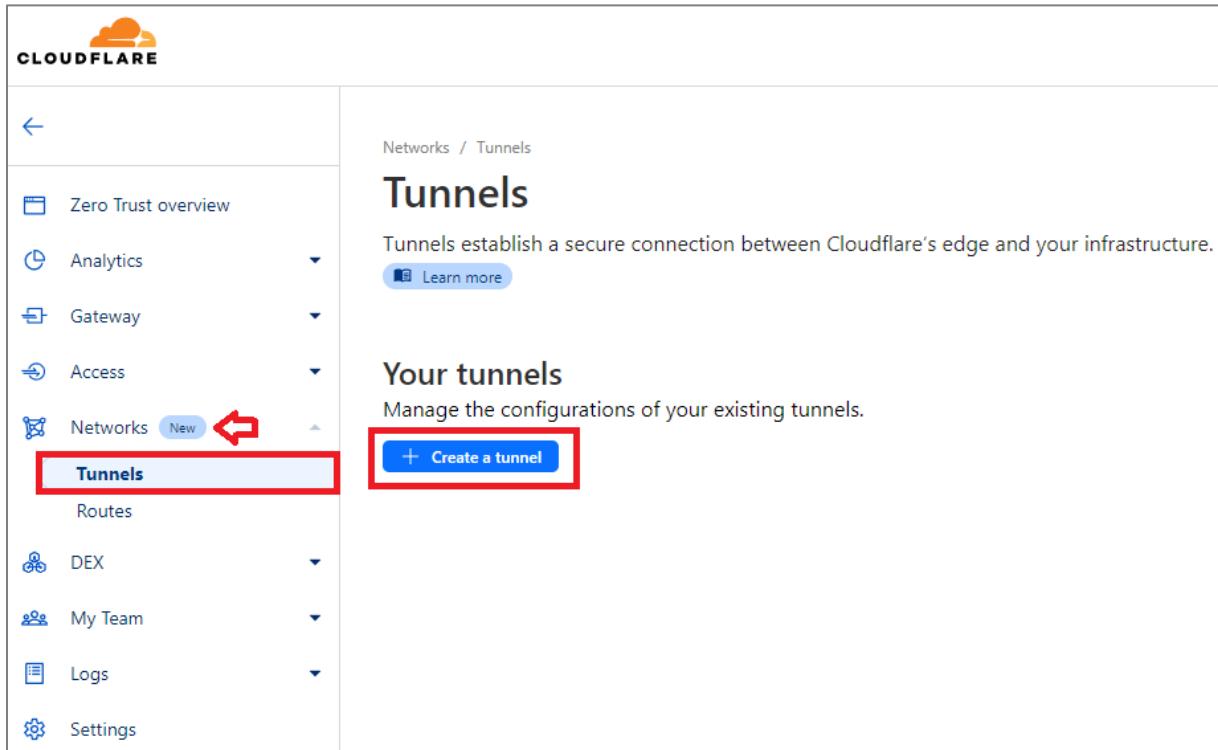
The screenshot shows the Cloudflare Access dashboard. The left sidebar has a red box around the 'Access' item. The main content area has a message: 'Configuration for Access has moved to Cloudflare Zero Trust as of May 25, 2022. This page has been deprecated. All existing functionality, as well as new features, is available in Zero Trust.' A blue button labeled 'Launch Zero Trust' is visible. The top bar shows the domain 'smarthomecourse.com' is active, starred, and on a free plan.

Choose the **free plan** for the Cloudflare Zero Trust.

You'll be redirected to the Cloudflare Zero Trust dashboard:

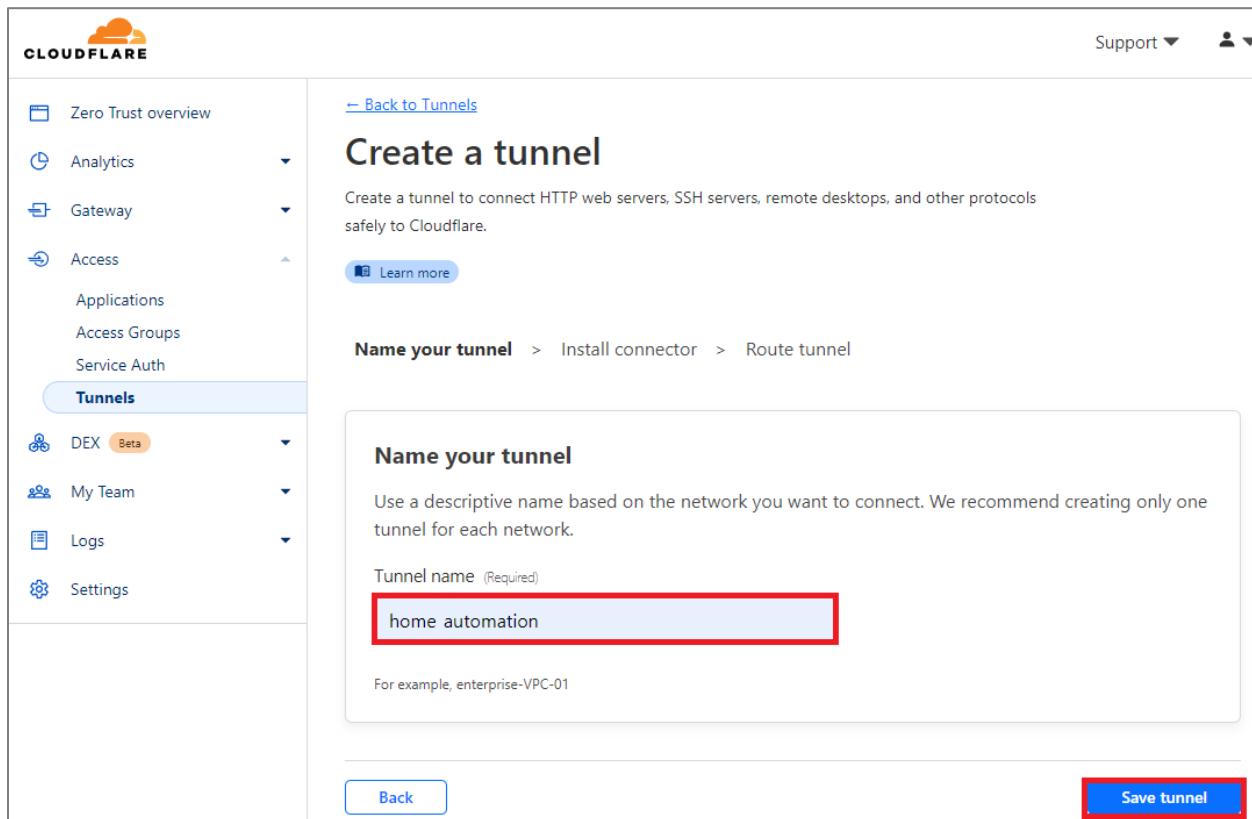
- <https://one.dash.cloudflare.com>

Then, under the **Networks** menu, select **Tunnels** and click on **Create a tunnel**.



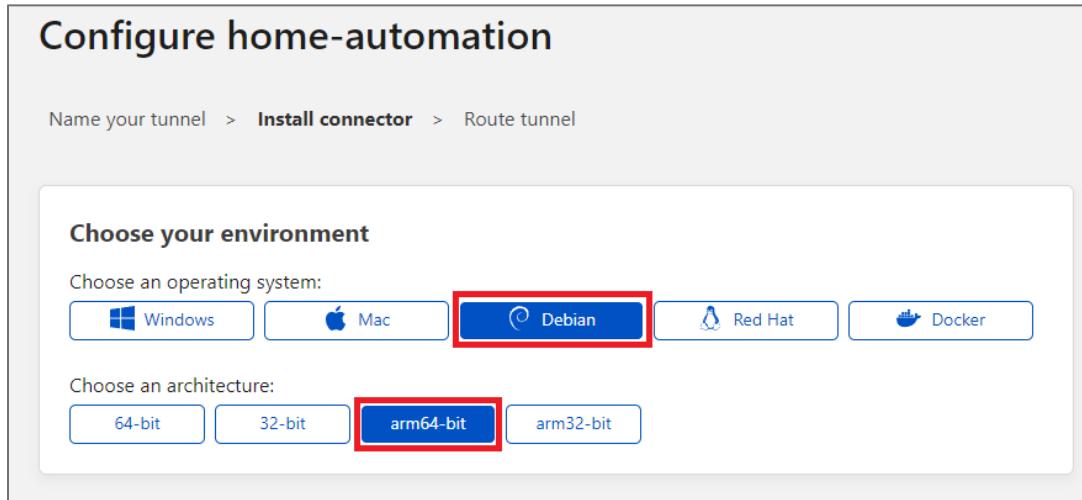
The screenshot shows the Cloudflare interface. On the left, there's a sidebar with various menu items: Zero Trust overview, Analytics, Gateway, Access, Networks (which has a 'New' button and a red arrow pointing to it), Tunnels (highlighted with a red box), Routes, DEX, My Team, Logs, and Settings. The main content area is titled 'Tunnels' and contains the sub-section 'Your tunnels'. It says 'Manage the configurations of your existing tunnels.' and features a blue 'Create a tunnel' button. The entire 'Your tunnels' section is also highlighted with a red box.

Give a name to your tunnel, for example *home automation*, and click **Save tunnel**.



The screenshot shows the 'Create a tunnel' page. The left sidebar is identical to the previous one, with 'Tunnels' selected. The main area has a heading 'Create a tunnel' and a sub-heading 'Name your tunnel'. It says 'Create a tunnel to connect HTTP web servers, SSH servers, remote desktops, and other protocols safely to Cloudflare.' Below this is a 'Learn more' link. A breadcrumb navigation shows 'Name your tunnel > Install connector > Route tunnel'. The 'Name your tunnel' section contains a 'Tunnel name (Required)' field with the value 'home automation' (which is also highlighted with a red box). Below the field is a placeholder 'For example, enterprise-VPC-01'. At the bottom right is a blue 'Save tunnel' button.

Then, if you're using a Raspberry Pi, select **Debian** and **arm64-bit architecture**.

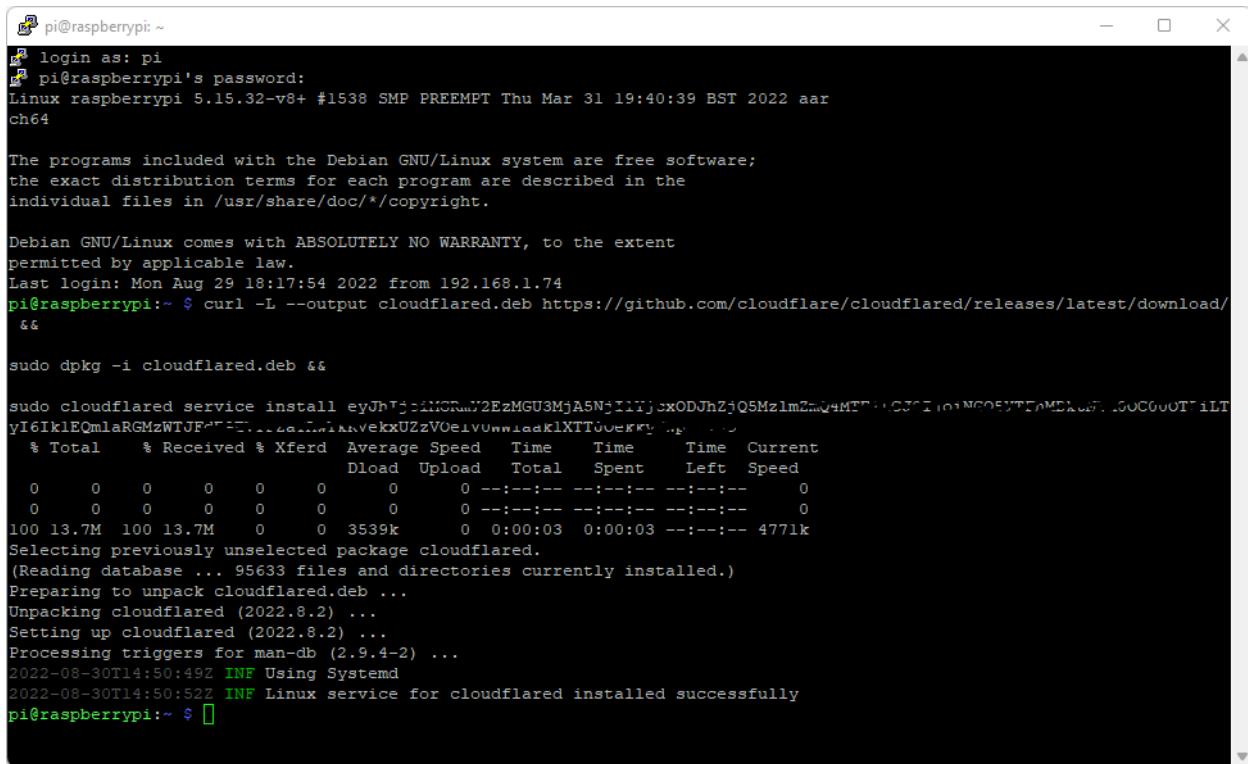


Scroll down and copy the whole command at the left.

The screenshot shows the 'Install and run a connector' section of the Cloudflare Tunnel setup. It provides instructions for connecting to Cloudflare using a terminal command. Two sections are shown: one for users who don't have cloudfaired installed, and one for users who already have it installed. The command in the first section is highlighted with a red box.

```
curl -L --output cloudfaired.deb https://github.com/cloudflare/cloudfaired/releases/latest/download/cloudfaired-linux-arm64.deb &&
sudo dpkg -i cloudfaired.deb &&
$ sudo cloudflared service install
```

Run that command on your Raspberry Pi PuTTY terminal window.



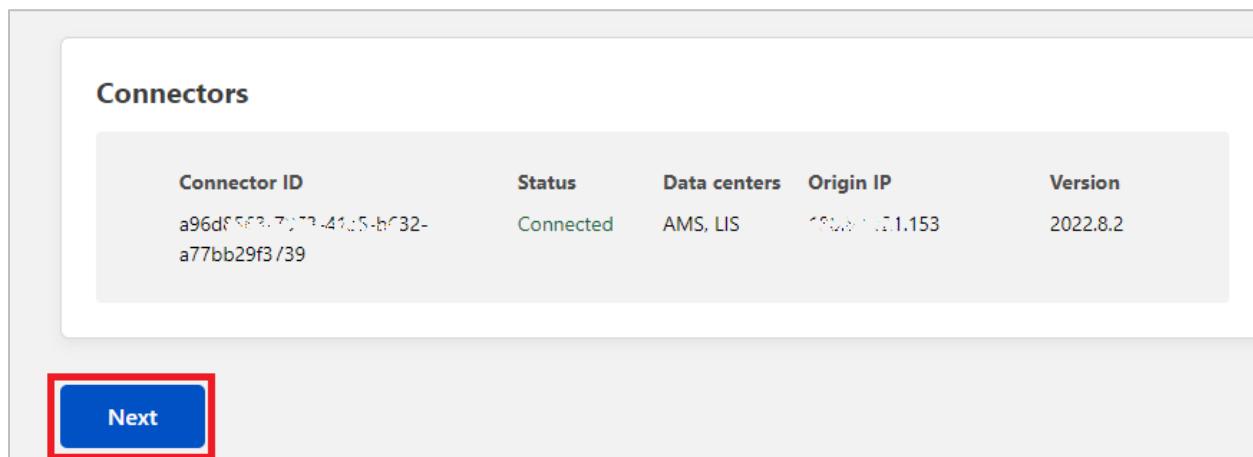
```
pi@raspberrypi: ~
pi login as: pi
pi@raspberrypi's password:
Linux raspberrypi 5.15.32-v8+ #1538 SMP PREEMPT Thu Mar 31 19:40:39 BST 2022 aarch64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Aug 29 18:17:54 2022 from 192.168.1.74
pi@raspberrypi:~ $ curl -L --output cloudfaired.deb https://github.com/cloudflare/cloudflared/releases/latest/download/
cloudfaired.deb
pi@raspberrypi:~ $ sudo dpkg -i cloudfaired.deb &&
pi@raspberrypi:~ $ sudo cloudfaired service install
pi@raspberrypi:~ $ cat /var/log/syslog
pi@raspberrypi:~ $
```

You should get a success message "*Linux service for cloudfaired installed successfully*".

Go back to Zero Trust dashboard, scroll down and click Next.



The screenshot shows the "Connectors" section of the Zero Trust dashboard. It displays a table with the following data:

| Connector ID | Status | Data centers | Origin IP | Version |
|--------------------------------------|-----------|--------------|------------------|----------|
| a96d89c3-7173-41c5-bf32-a77bb29f3/39 | Connected | AMS, LIS | 192.168.1.11.153 | 2022.8.2 |

A blue "Next" button is highlighted with a red box at the bottom left of the connector card.

Now, fill in the details to access Node-RED on your domain name as shown below (but use your domain name instead).

- **Subdomain:** nodered
- **Domain:** your registered domain name
- **Path:** (leave it blank)
- **Service:** HTTP, localhost:1880

Route Traffic for home automation

Name your tunnel > Install connector > **Route tunnel**

Public Hostnames **Private Networks**

Edit public hostname for home automation

Public hostname

| | | |
|------------------|--------------------------|-----------------|
| Subdomain | Domain (Required) | Path |
| nodered | smarthomecourse.com | (optional) path |

Service

| | |
|------------------------|-----------------------|
| Type (Required) | URL (Required) |
| HTTP | localhost:1880 |

For example, https://localhost:8001

[Additional application settings ▶](#)

[Back](#) **Save tunnel**

Click on **Save tunnel**.

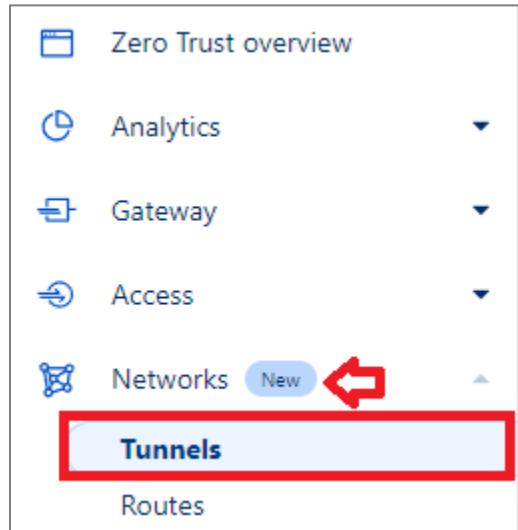
With these settings you'll be accessing Node-RED on the following:

`https://nodered.<your-domain-name>`

For example, in my case:

`https://nodered.smarthomecourse.com`

In the **Network**, open the **Tunnels** menu:



In the **Tunnels** menu, you'll see a list of the available tunnels.

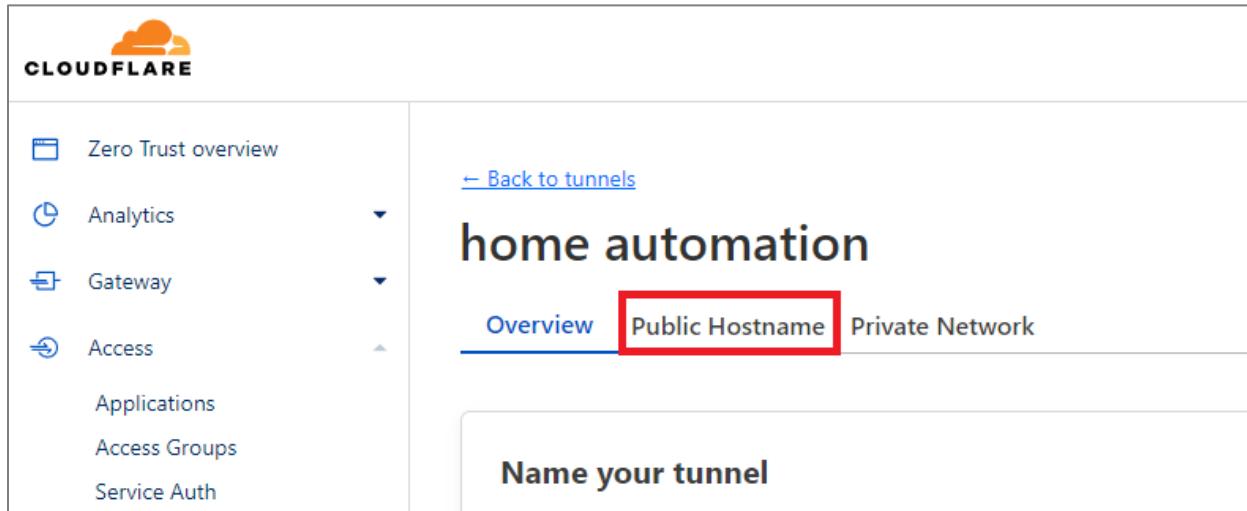
The screenshot shows the 'Tunnels' page under 'Access / Tunnels'. The title is 'Tunnels'. A sub-header says 'Tunnels establish a secure connection between Cloudflare's edge and your infrastructure.' with a 'Learn more' link. Below is a section titled 'Your tunnels' with a sub-header 'Manage the configurations of your existing tunnels.' and a 'Create a tunnel' button. A search bar is also present. A table lists one tunnel:

| Tunnel name | Tunnel ID | Status | Routes | Uptime | Created |
|-----------------|-------------------------------------|---------|---------------------|--------|---------------|
| home automation | 4c171d00-1042-4911-98c3-41111c1328b | HEALTHY | nodered.smarthomeco | 2 days | July 12, 2023 |

Now, you need to configure your tunnel to add a subdomain to access InfluxDB. On the right corner click on **Configure**.

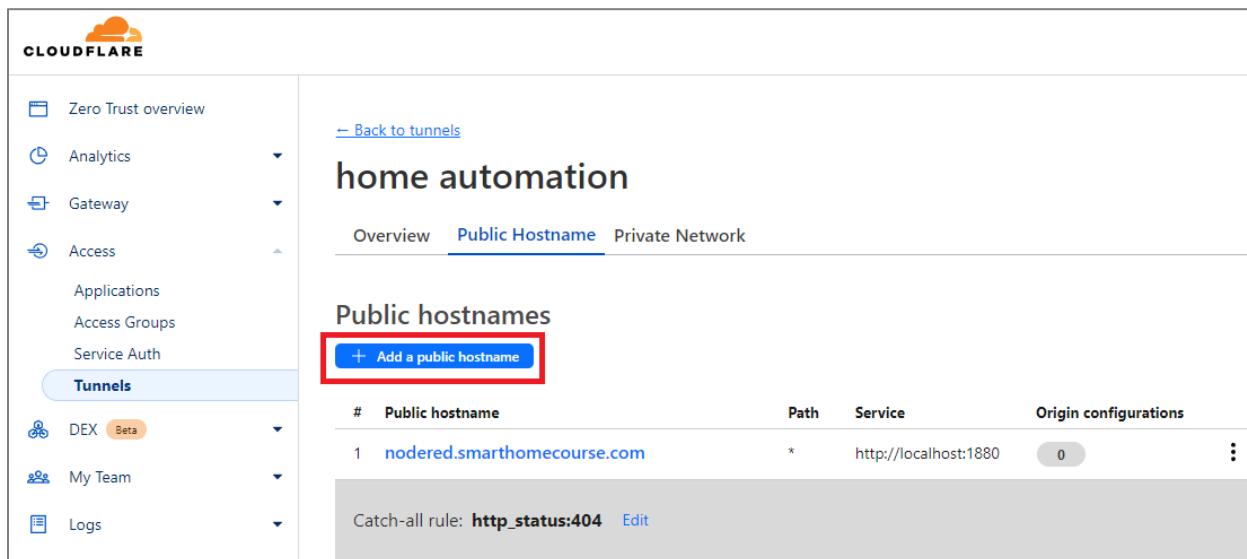
The screenshot shows a modal window for configuring a tunnel. It includes a search bar at the top. The main area displays tunnel details: Status (228b, HEALTHY), Routes (nodered.smarthomeco), Uptime (2 days), and Created (July 12, 2023). On the right side, there is a vertical ellipsis menu with two options: 'Configure' (highlighted with a red box) and 'Delete'.

The following menu will open. Make sure you select the **Public Hostname** tab.



Screenshot of the Cloudflare Tunnel Overview page for a tunnel named "home automation". The left sidebar shows navigation links like Zero Trust overview, Analytics, Gateway, Access, Applications, Access Groups, and Service Auth. The main content area has a "Back to tunnels" link, the tunnel name "home automation", and three tabs: Overview (selected), Public Hostname (highlighted with a red box), and Private Network. Below the tabs is a section titled "Name your tunnel".

Click **Add a public hostname**.



Screenshot of the Cloudflare Public Hostnames page for the "home automation" tunnel. The left sidebar includes a "Tunnels" section which is highlighted with a blue background. The main content area shows the "Public hostnames" section with a "Add a public hostname" button (highlighted with a red box). A table lists one public hostname entry: "nodered.smarthomecourse.com" with service "http://localhost:1880".

| # | Public hostname | Path | Service | Origin configurations |
|---|-----------------------------|------|-----------------------|-----------------------|
| 1 | nodered.smarthomecourse.com | * | http://localhost:1880 | 0 |

Then fill in the details as follows:

- **Subdomain:** influxdb
- **Domain:** your registered domain name
- **Path:** (leave it blank)
- **Service:** HTTP, localhost:8086

Edit public hostname for home-automation

Public hostname (Required)

| | | |
|------------------|--------------------------|---------------------|
| Subdomain | Domain (Required) | Path |
| influxdb | . | smarthomecourse.com |
| | | / (optional) path |

Service (Required)

| | | | |
|------|---|----|----------------|
| HTTP | : | // | localhost:8086 |
|------|---|----|----------------|

For example, <https://localhost:8001>

[Additional application settings ▶](#)

Save hostname

Finally, click on **Save hostname**.

With these settings you'll be accessing InfluxDB on the following:

`https://influxdb.<your-domain-name>`

For example, in my case:

`https://influxdb.smarthomecourse.com`

Wait a couple of minutes for the changes to take effect.

Now, you can access InfluxDB and Node-RED on your domain name from anywhere.

The image displays two separate browser windows side-by-side. Both windows have their URLs highlighted with a red box.

Node-RED: The URL is `nodered.smarthomecourse.com`. The page features a large red background with a white Node-RED logo icon (two nodes connected by wires) and the text "Node-RED". On the right, there is a login form with fields for "Username" and "Password", and a "Login" button.

InfluxDB: The URL is `influxdb.smarthomecourse.com/signin`. The page has a dark purple background with a light purple cube logo. It features the "influxdb" logo. Below the logo is a login form with fields for "Username" and "Password", and a "SIGN IN" button.

MODULE 10

Creating a Cloud Server



If you don't have a Raspberry Pi to follow along, you can create your system on the cloud. We provide instructions about installing Node-RED, InfluxDB, and Mosquitto broker on Digital Ocean (hosting).

10.1 - Introduction (Digital Ocean)

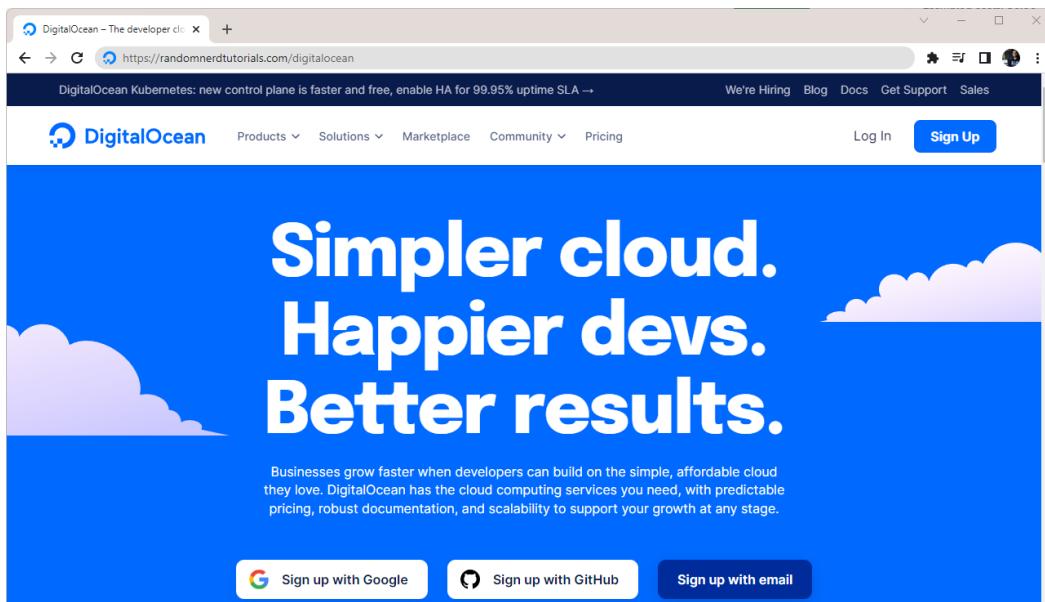
To run your system on the cloud, you need to use a hosting service that allows you to have access to the command line and install any software that you need. I recommend using [Digital Ocean](#) that offers an Ubuntu server that you can manage through a command line.

I've been using it since 2015 and I personally recommend it, but you can use any other hosting service. Any hosting service that offers a Linux Ubuntu VM with full console access should work. If you don't have a hosting account, I recommend [signing up for Digital Ocean](#).

When you sign up for Digital Ocean, you can try it for 60 days (they give you free credits to test the platform). You need to go to this link in order to claim the free credits: <https://randomnerdtutorials.com/digitalocean>

Creating Digital Ocean Account

To create a Digital Ocean Account, go to [Digital Ocean](#) and sign up using one of the available options.



On the **Projects** tab, click on your name.

The screenshot shows the DigitalOcean interface. On the left, a sidebar titled 'PROJECTS' has 'Rui Santos' selected. The main area displays the 'Rui Santos' project details, including a profile icon, the project name, and a 'DEFAULT' status. Below this are tabs for 'Resources', 'Activity', and 'Settings'. Under 'Resources', there's a section for 'DROPLETS' showing four entries, each with a green dot and a blue water droplet icon. There's also a section for 'DOMAINS' which is currently empty. At the top right, there's a 'Create' button, a help icon, a notification bell with 45 notifications, and a 'My Team' section.

Then, click on **Create > Droplets**.

This screenshot shows a dropdown menu from the 'Create' button. The 'Droplets' option is highlighted with a red box. Other options listed include 'Kubernetes', 'Apps', 'Functions', 'Databases', 'Volumes', 'Spaces', and 'Domains/DNS'. A 'NEW' badge is visible next to the Functions option.

- Droplets**
Create cloud servers
- Kubernetes
Create Kubernetes clusters
- Apps
Deploy your code
- Functions
Create Cloud Functions **NEW**
- Databases
Create database clusters
- Volumes
Add storage to Droplets
- Spaces
Store and serve static assets
- Domains/DNS
Route your existing domains

Then, select the following options:

- **Distributions:** Ubuntu
- **Choose a plan:** Shared CPU Basic—we recommend choosing the \$6/month option (the \$4 plan will also work, but might be a bit slow).

Choose an image [?](#)

[Distributions](#) Marketplace Snapshots Backups Custom images

| | | | | | |
|---|---|--|--|--|---|
|  Ubuntu 22.04 x64 ▼ |  RancherOS Select version ▼ |  Fedora Select version ▼ |  Debian Select version ▼ |  CentOS Select version ▼ |  Rocky Linux Select version ▼ |
|---|---|--|--|--|---|

Choose a plan [Help me choose ↗](#)

| SHARED CPU | | DEDICATED CPU | | | | | | | | | | | | | | | |
|--|--|---|--|--|---|------------------------|------------------------|-------------------------|-------------------------|-------------------------|-------------------------|---|--|---|--|--|---|
| Basic | | General Purpose | CPU-Optimized | Memory-Optimized | Storage-Optimized | | | | | | | | | | | | |
| <p>Basic virtual machines with a mix of memory and compute resources. Best for small projects that can handle variable levels of CPU performance, like blogs, web apps and dev/test environments.</p> <p>CPU options: <input checked="" type="radio"/> Regular with SSD <input type="radio"/> Premium Intel with NVMe SSD NEW <input type="radio"/> Premium AMD with NVMe SSD NEW</p> <table border="1"><tbody><tr><td>\$4/mo \$0.006/hour</td><td>\$6/mo \$0.009/hour</td><td>\$12/mo \$0.018/hour</td><td>\$18/mo \$0.027/hour</td><td>\$24/mo \$0.036/hour</td><td>\$48/mo \$0.071/hour</td></tr><tr><td>512 MB / 1 CPU 10 GB SSD Disk 500 GB transfer</td><td>1 GB / 1 CPU 25 GB SSD Disk 1000 GB transfer</td><td>2 GB / 1 CPU 50 GB SSD Disk 2 TB transfer</td><td>2 GB / 2 CPUs 60 GB SSD Disk 3 TB transfer</td><td>4 GB / 2 CPUs 80 GB SSD Disk 4 TB transfer</td><td>8 GB / 4 CPUs 160 GB SSD Disk 5 TB transfer</td></tr></tbody></table> <p>Show all plans</p> | | | | | | \$4/mo \$0.006/hour | \$6/mo \$0.009/hour | \$12/mo \$0.018/hour | \$18/mo \$0.027/hour | \$24/mo \$0.036/hour | \$48/mo \$0.071/hour | 512 MB / 1 CPU 10 GB SSD Disk 500 GB transfer | 1 GB / 1 CPU 25 GB SSD Disk 1000 GB transfer | 2 GB / 1 CPU 50 GB SSD Disk 2 TB transfer | 2 GB / 2 CPUs 60 GB SSD Disk 3 TB transfer | 4 GB / 2 CPUs 80 GB SSD Disk 4 TB transfer | 8 GB / 4 CPUs 160 GB SSD Disk 5 TB transfer |
| \$4/mo \$0.006/hour | \$6/mo \$0.009/hour | \$12/mo \$0.018/hour | \$18/mo \$0.027/hour | \$24/mo \$0.036/hour | \$48/mo \$0.071/hour | | | | | | | | | | | | |
| 512 MB / 1 CPU 10 GB SSD Disk 500 GB transfer | 1 GB / 1 CPU 25 GB SSD Disk 1000 GB transfer | 2 GB / 1 CPU 50 GB SSD Disk 2 TB transfer | 2 GB / 2 CPUs 60 GB SSD Disk 3 TB transfer | 4 GB / 2 CPUs 80 GB SSD Disk 4 TB transfer | 8 GB / 4 CPUs 160 GB SSD Disk 5 TB transfer | | | | | | | | | | | | |

Choose a datacenter region—choose the closest to your location.

Choose a datacenter region

| | | | | | |
|---|--|--|--|---|--|
|  New York 1 2 3 |  San Francisco 1 2 3 |  Amsterdam 2 3 |  Singapore 1 |  London 1 |  Frankfurt 1 |
|  Toronto 1 |  Bangalore 1 | | | | |

For the Authentication, select **Password** and create a password—**you need to remember it later.**

Authentication ?

SSH keys
A more secure authentication method

Password
Create a root password to access Droplet (less secure)

Create root password *

PASSWORD REQUIREMENTS

- ✓ Must be at least 8 characters long
- ✓ Must contain 1 uppercase letter (cannot be first or last character)
- ✓ Must contain 1 number
- ✓ Cannot end in a number or special character

Please store your password securely. You will not be sent an email containing the Droplet's details or password.

Then, you can select any additional options you think might be useful for your project.

Select additional options ?

| | | |
|--|---|---|
| <input type="checkbox"/> Enable backups RECOMMENDED | A system-level backup is taken once a week, and each backup is retained for 4 weeks. | \$1.20/mo (per Droplet) 20% of the Droplet price |
| <input type="checkbox"/> Monitoring | Enables additional Droplet metrics collection, monitoring, and alerting. | FREE |
| <input type="checkbox"/> IPv6 | Enables public IPv6 networking. | FREE |
| <input type="checkbox"/> User data | Enter user data when you create a Droplet to perform tasks or run scripts as the root user during a Droplet's first boot. | FREE |

Finally, choose a hostname for your droplet and finally **Create Droplet**.

Finalize and create

How many Droplets?

Deploy multiple Droplets with the same configuration.

| | | |
|---|-----------|---|
| - | 1 Droplet | + |
|---|-----------|---|

Choose a hostname

Give your Droplets an identifying name you will remember them by. Your Droplet name can only contain alphanumeric characters, dashes, and periods.

Add tags

Use tags to organize and relate resources. Tags may contain letters, numbers, colons, dashes, and underscores.

Select Project

Assign Droplets to a project

Create Droplet

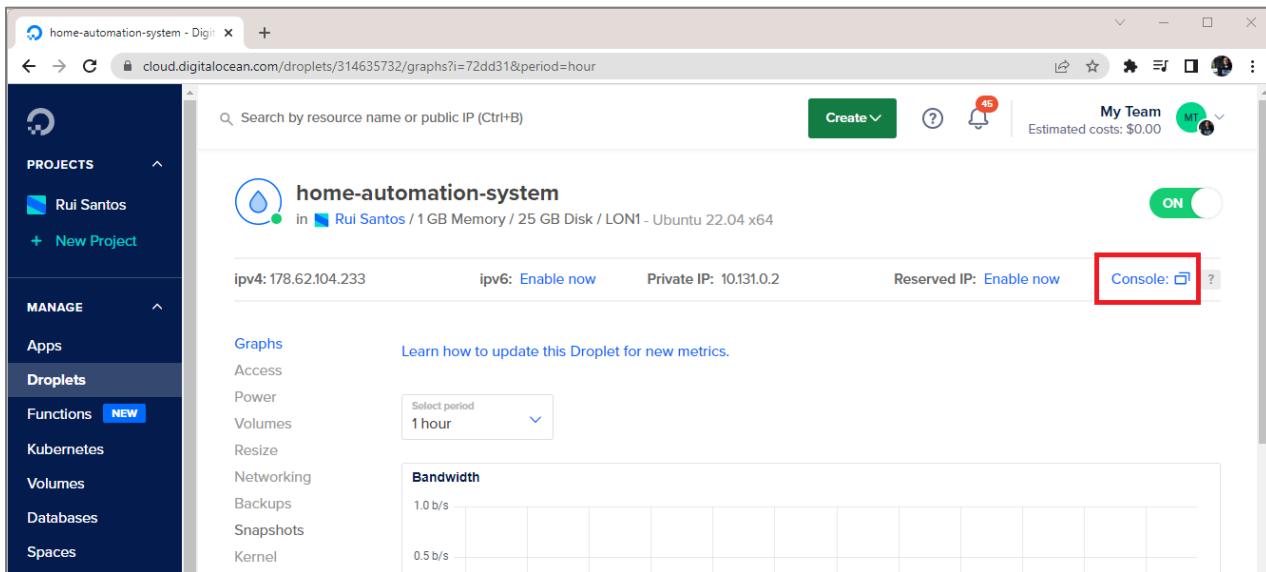
Now, if you click on the **Droplets** tab, your newly created droplet should be there.

The screenshot shows the DigitalOcean dashboard interface. On the left, there's a sidebar with 'PROJECTS' (containing 'Rui Santos' and '+ New Project'), 'MANAGE' (containing 'Apps', 'Droplets' - which is currently selected and highlighted in blue, 'Functions' with a 'NEW' button, 'Kubernetes', 'Volumes', 'Databases', and 'Spaces'). The main area is titled 'Droplets' and contains a table with one row:

| Name | IP Address | Created | Tags |
|---|----------------|---------------|------|
| home-automation-system 1 GB / 25 GB Disk / LON1 - Ubuntu 22.04 x64 | 178.62.104.233 | 4 minutes ago | |

At the top right, there are buttons for 'Create' (with a dropdown), help, and notifications (with a red badge showing 45). There are also search bars for resources and droplets.

If you click on the droplet name, you'll be presented with the following:



At the top right corner, there's a **Console** link. If you click there, it will open a new console/terminal window where you can type Linux commands to install software or run commands the same way you do on your Raspberry Pi via SSH.

The username is `root`, and the password is the one you created when you set up the droplet.

```
home-automation-system - DigitalOcean Droplet Web Console - Google Chrome
cloud.digitalocean.com/droplets/314635732/terminal/ui/
Welcome to Ubuntu 22.04 LTS (GNU/Linux 5.15.0-41-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information as of Tue Aug 30 10:19:41 UTC 2022

 System load:  0.52099609375   Users logged in:      0
 Usage of /:   6.3% of 24.05GB  IPv4 address for eth0: 178.62.104.233
 Memory usage: 22%
 Swap usage:  0%
 Processes:    96               IPv4 address for eth0: 10.16.0.6
                                IPv4 address for eth1: 10.131.0.2

0 updates can be applied immediately.

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

root@home-automation-system:~# 
```

10.2 - Installing Node-RED on Digital Ocean

In this section, you'll learn how to install Node-RED on your Digital Ocean droplet.

Open the console on your droplet, as we explained in the previous unit. Then, follow the next steps.

Installing Node-RED

Run the following command to install Node-RED:

```
bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

It will ask you "*are you really sure you want to install as root?*". Type **y** and hit **Enter**.

"*Are you really sure?*". Type **y** and hit **Enter**.

Then, you'll be asked: "*Would you like to install Pi-specific nodes?*" Press **y** and **Enter**. It will take a few minutes to install Node-RED.

Configuring Node-RED Settings

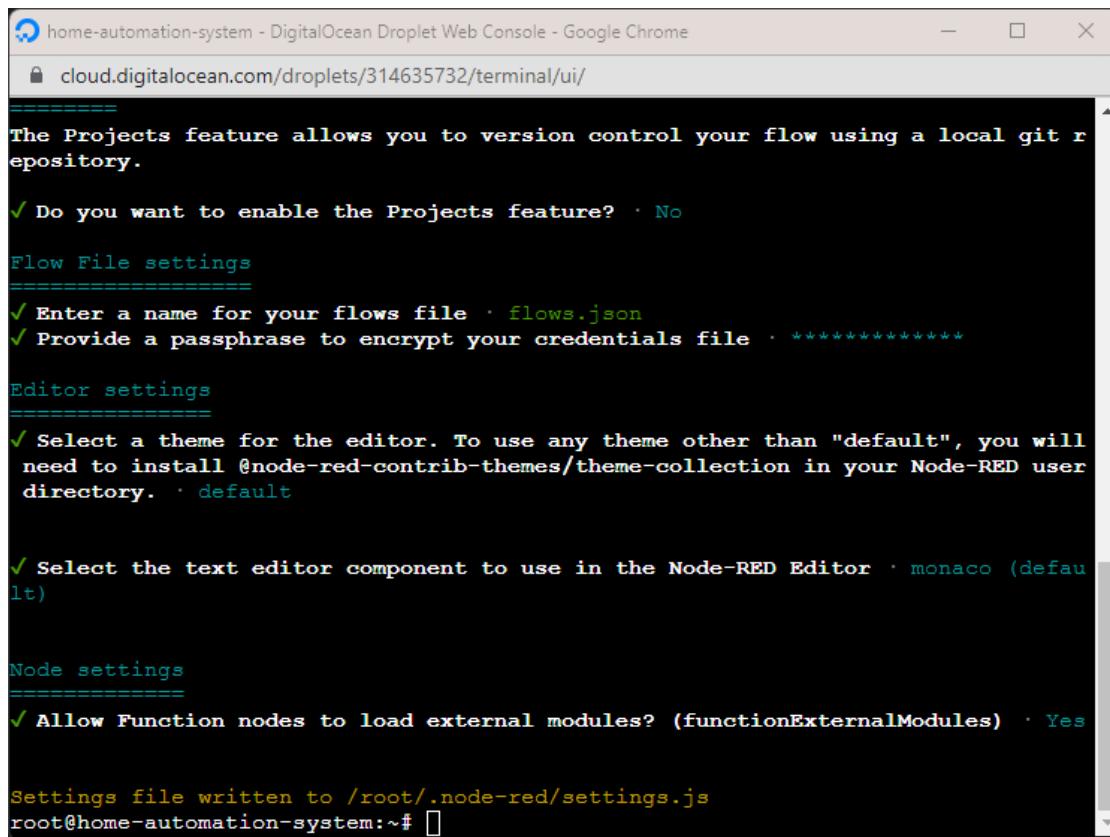
After installing, it is recommended to configure initial options and settings. Run the following command:

```
node-red admin init
```

- Press **Enter** to create a Node-RED Settings file on `/home/pi/.node-red/settings.js` (keep the default `settings.js` file location)
- Do you want to set up user security? **Yes**.
- Enter a username and press **Enter** ([you need to remember it later](#)).
- Enter a password and press **Enter** ([you need to remember it later](#)).

- Then, you need to define user permissions. We'll set full access, make sure the **full access** option is highlighted in blue and press **Enter**.
- You can add other users with different permissions if you want. We'll just create one user for now. You can always add other users later.
- Do you want to enable the Projects feature? **No**.
- Enter a name for your flows file. Press **Enter** to select the default name `flows.json`.
- Provide a passphrase to encrypt your credentials file. Learn more about what is a [passphrase](#).
- Select a theme for the editor. Simply press **Enter** to select default.
- Press **Enter** again to select the default text editor.
- Allow Function nodes to load external modules? **Yes**.

Node-RED configuration was successful. All settings are saved on `settings.js`.



The screenshot shows a terminal window titled "home-automation-system - DigitalOcean Droplet Web Console - Google Chrome" with the URL "cloud.digitalocean.com/droplets/314635732/terminal/ui/". The terminal output displays the configuration steps for Node-RED:

```

The Projects feature allows you to version control your flow using a local git repository.

✓ Do you want to enable the Projects feature? · No

Flow File settings
=====
✓ Enter a name for your flows file · flows.json
✓ Provide a passphrase to encrypt your credentials file · *****

Editor settings
=====
✓ Select a theme for the editor. To use any theme other than "default", you will
need to install @node-red-contrib-themes/theme-collection in your Node-RED user
directory. · default

✓ Select the text editor component to use in the Node-RED Editor · monaco (defau
lt)

Node settings
=====
✓ Allow Function nodes to load external modules? (functionExternalModules) · Yes

Settings file written to /root/.node-red/settings.js
root@home-automation-system:~# []

```

Start Node-RED

Run the following command to start Node-RED:

```
node-red-start
```

Autostart Node-RED on boot

To automatically run Node-RED when the server boots up, you need to enter the following command (press **CTRL + C** to stop the previous task before entering the new command).

```
sudo systemctl enable nodered.service
```

This means that as long as your server is up, Node-RED will be running.

Now, restart your server so the autostart takes effect.

```
sudo reboot
```

Note: If, later on, you want to disable autostart on boot, you can run:
`sudo systemctl disable nodered.service`

For more information about the installation process, check the [official documentation](#).

Access Node-RED

Wait a few seconds while everything loads.

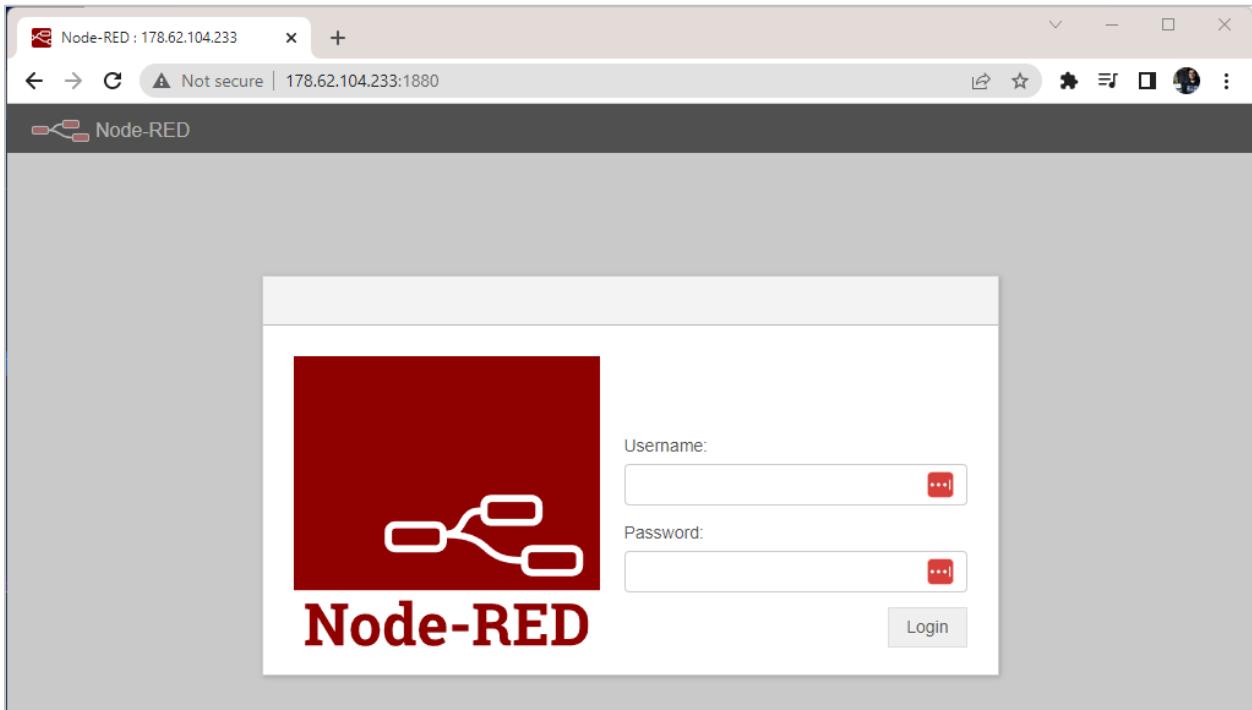
Node-RED runs on port 1880. To access Node-RED open a browser on your computer and type your server IP address followed by :1880.

```
http://<YOUR-Digital-Ocean-Droplet-IP-Address>:1880
```

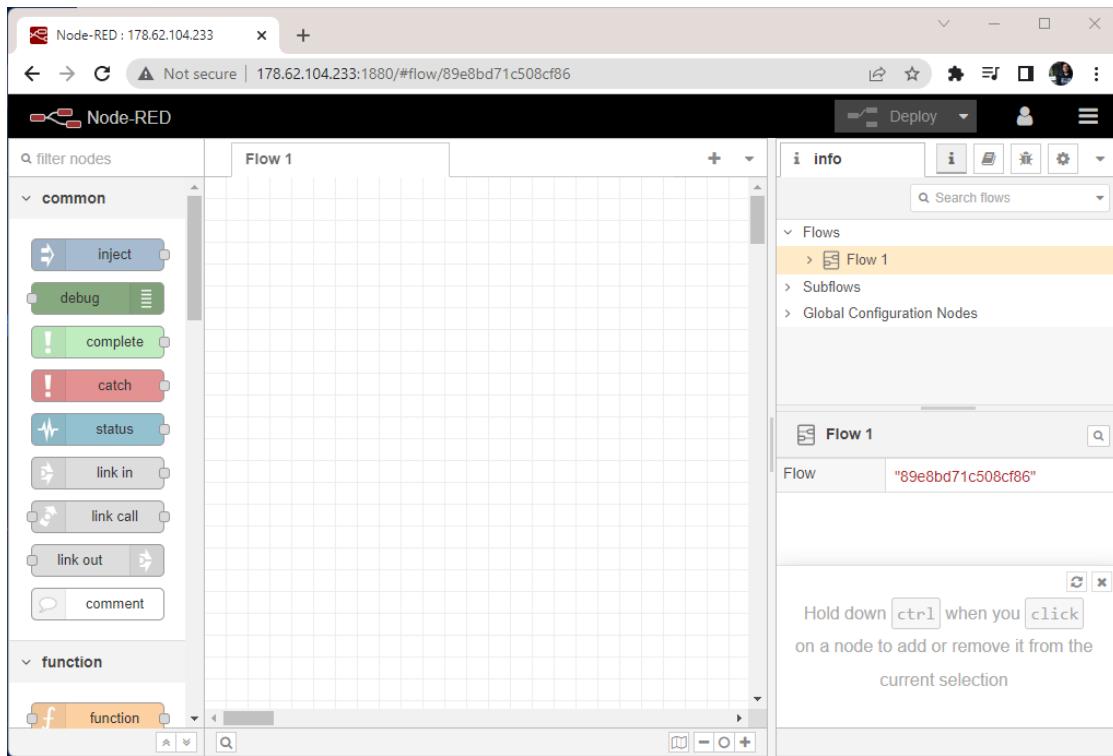
For example, in my case:

```
http://178.62.83.231:1880
```

The Node-RED login page should load. Insert your Node-RED username and password that you've defined when configuring the Node-RED settings.



Now, you have access to Node-RED. You can start building your flows on the cloud.



Secure Node-RED Dashboard with Username and Password

As we've seen previously, you can create a dashboard (UI) using Node-RED dashboard nodes. We explain how to install those nodes in Unit 2.4. Node-RED editor is protected using username and password, but not the user interface. After creating a user interface, if you want to protect it with username and password, follow the next steps.

- 1) Open the Digital Ocean console for your droplet and run the following command to edit the *settings.js* file.

```
nano ~/.node-red/settings.js
```

- 2) Scroll down through the file using the arrow keys until you find the section that mentions.

```
/* To password protect the Node-RED editor and admin API, the  
following property can be used...*/
```

- 3) Copy the username and password combination with **CTRL+C**.

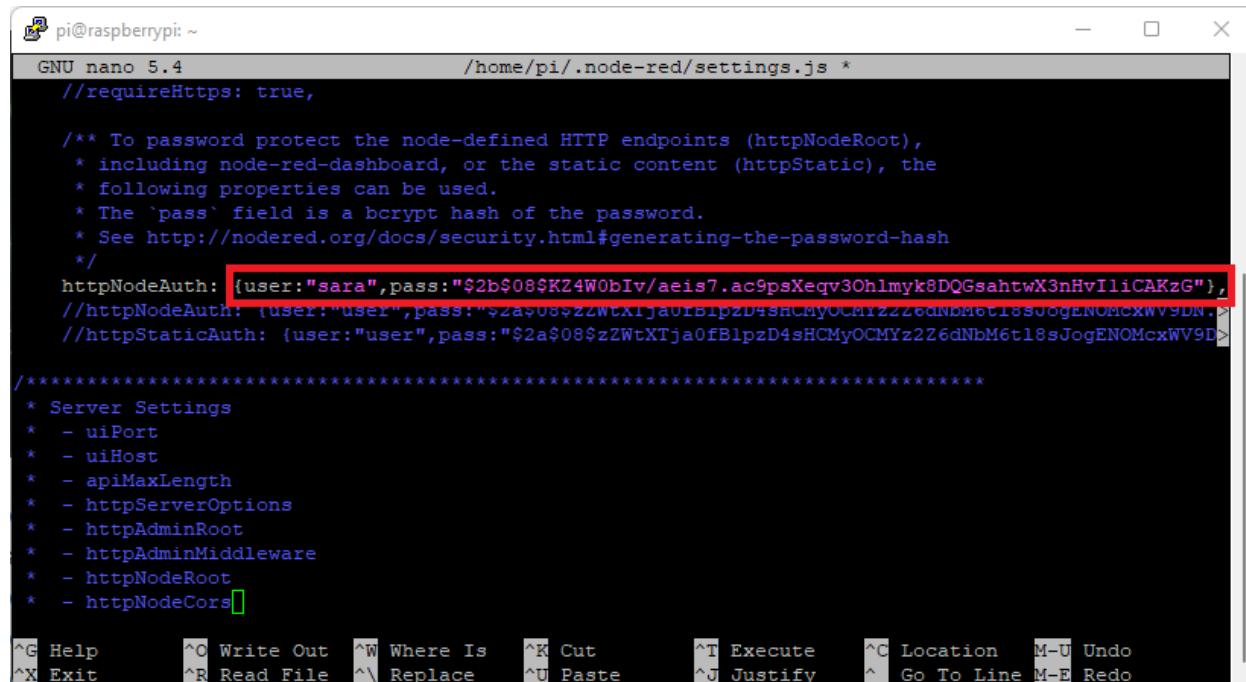
```
//nodesDir: '/home/nol/.node-red/nodes',  
  
*****  
* Security  
* - adminAuth  
* - https  
* - httpsRefreshInterval  
* - requireHttps  
* - httpNodeAuth  
* - httpStaticAuth  
*****  
  
/** To password protect the Node-RED editor and admin API, the following  
* property can be used. See http://nodered.org/docs/security.html for details.  
*/  
  
adminAuth: {  
    "type": "credentials",  
    "users": [  
        {  
            "username": "sara",  
            "password": "$2b$08$KZ4W0bIv/aeis7.ac9psXeqv3Ohlmyk8DQGsahtwX3nHvIliCAKzG",  
            "permissions": "*"  
        }  
    ]  
}  
  
^G Help      ^O Write Out    ^W Where Is     ^K Cut          ^T Execute      ^C Location  
^X Exit      ^R Read File    ^\ Replace     ^U Paste         ^J Justify      ^_ Go To Line
```

4) Continue scrolling down the file until you find the following section:

```
/** To password protect the node-defined HTTP endpoints
(httpNodeRoot),
* including node-red-dashboard, or the static content
(httpStatic), the
* following properties can be used.
```

5) Edit the `httpNodeAuth` with the user and password you copied previously. In my case, the like will look as follows:

```
httpNodeAuth: {user:"sara",pass:"$2b$08$KZ4W0bIv/aeis7.ac9psXeqv3
Ohlmyk8DQGsahtwX3nHvI1iCAKzG"},
```



```
pi@raspberrypi: ~
GNU nano 5.4          /home/pi/.node-red/settings.js *
/requiresHttps: true,
/** To password protect the node-defined HTTP endpoints (httpNodeRoot),
 * including node-red-dashboard, or the static content (httpStatic), the
 * following properties can be used.
 * The `pass` field is a bcrypt hash of the password.
 * See http://nodered.org/docs/security.html#generating-the-password-hash
 */
httpNodeAuth: {user:"sara",pass:"$2b$08$KZ4W0bIv/aeis7.ac9psXeqv3Ohlmyk8DQGsahtwX3nHvI1iCAKzG"},  
//httpNodeAuth: {user:"user",pass:"$2a$08$z2WtXTja0fBlpzD4sHCMYz2Z6dNbM6tl8sJogENOMcxWV9D"  
//httpStaticAuth: {user:"user",pass:"$2a$08$z2WtXTja0fBlpzD4sHCMYz2Z6dNbM6tl8sJogENOMcxWV9D"  

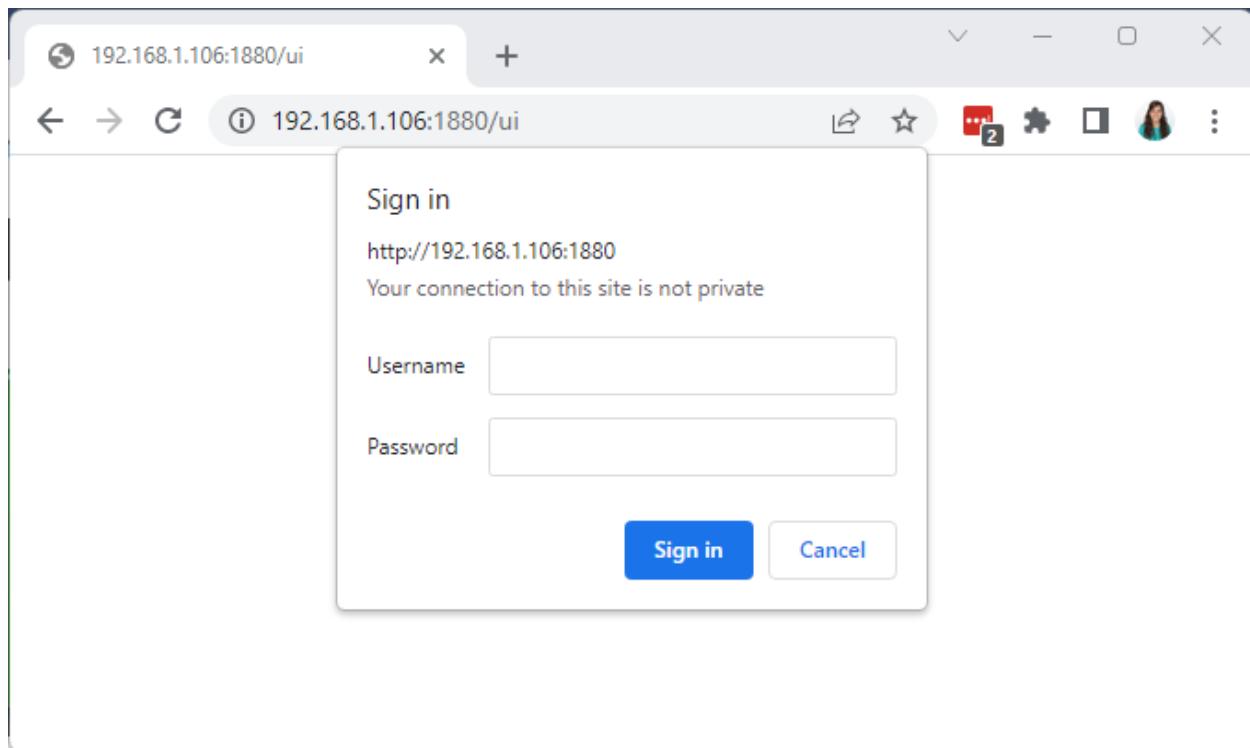
*****  
* Server Settings  
* - uiPort  
* - uiHost  
* - apiMaxLength  
* - httpServerOptions  
* - httpAdminRoot  
* - httpAdminMiddleware  
* - httpNodeRoot  
* - httpNodeCors[]  

^G Help      ^O Write Out   ^W Where Is    ^K Cut        ^T Execute     ^C Location   M-U Undo  
^X Exit      ^R Read File   ^\ Replace    ^U Paste      ^J Justify    ^^ Go To Line M-E Redo
```

- 6) After making the required changes, press **CTRL+X**, then **Y** and **Enter** to save the changes.
7) Reboot your server.

```
sudo reboot
```

Now, the next time you try to access Node-RED dashboard you need to insert a username and password. The user/password combination is the same you use to access Node-RED editor.



10.3 - Mosquitto Cloud MQTT Broker

With the console open on your Digital Ocean droplet, follow the next instructions to install Mosquitto MQTT broker.

Installing Mosquitto MQTT Broker

- 1) Run the following command to upgrade and update your system:

```
sudo apt update && sudo apt upgrade
```

- 2) When asked, press **Y** and **Enter**. It will take some time to update and upgrade.
- 3) To install the Mosquitto Broker enter these next command:

```
sudo apt install -y mosquitto mosquitto-clients
```

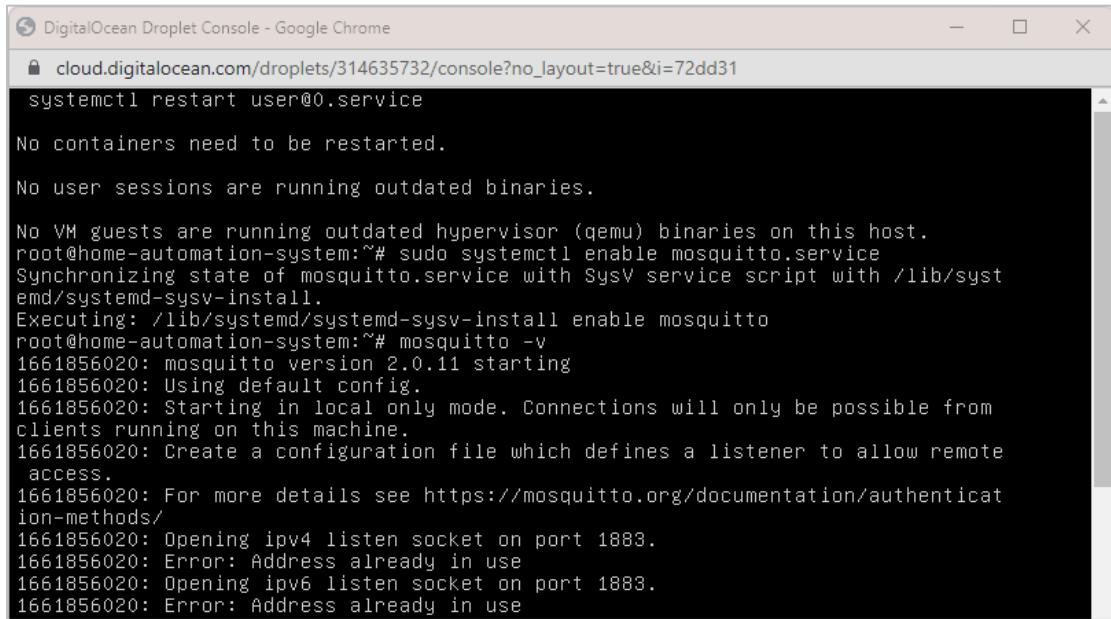
- 4) To make Mosquitto auto start when the server boots, you need to run the following command:

```
sudo systemctl enable mosquitto.service
```

- 5) Now, test the installation by running the following command:

```
mosquitto -v
```

This returns the Mosquitto version that is currently running on your server. It will be 2.0.11 or above.



```
DigitalOcean Droplet Console - Google Chrome
cloud.digitalocean.com/droplets/314635732/console?no_layout=true&i=72dd31
systemctl restart user@0.service
No containers need to be restarted.
No user sessions are running outdated binaries.

No VM guests are running outdated hypervisor (qemu) binaries on this host.
root@home-automation-system:~# sudo systemctl enable mosquitto.service
Synchronizing state of mosquitto.service with SysV service script with /lib/systemd/systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable mosquitto
root@home-automation-system:~# mosquitto -v
1661856020: mosquitto version 2.0.11 starting
1661856020: Using default config.
1661856020: Starting in local only mode. Connections will only be possible from clients running on this machine.
1661856020: Create a configuration file which defines a listener to allow remote access.
1661856020: For more details see https://mosquitto.org/documentation/authentication-methods/
1661856020: Opening ipv4 listen socket on port 1883.
1661856020: Error: Address already in use
1661856020: Opening ipv6 listen socket on port 1883.
1661856020: Error: Address already in use
```

You can ignore the error message “Error: Address already in use”.

Enable Remote Access/ Authentication

To enable remote access so that we can communicate with IoT devices, we need to edit/create a configuration file.

We'll add authentication with user and password.

- 1) Run the following command, but replace `YOUR_USERNAME` with the username you want to use:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd YOUR_USERNAME
```

I'll be using the MQTT user `sara`, so I run the command as follows:

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd sara
```

When you run the preceding command with the desired username, you'll be asked to enter a password. No characters will be displayed while you enter the password. Enter the password and memorize the user/pass combination, you'll need it later in your projects to make a connection with the broker.

This previous command creates a password file called `passwd` on the `/etc/mosquitto` directory.

Now, we need to edit the mosquitto configuration file so that it only allows authentication with the username and password we've defined.

- 2) Set the correct permissions in the `passwd` file:

```
sudo chown mosquitto /etc/mosquitto/passwd
```

- 3) Run the following command to edit the configuration file:

```
sudo nano /etc/mosquitto/mosquitto.conf
```

- 4) Add the following line at the top of the file (make sure it is at the top of the file, otherwise it won't work):

```
per_listener_settings true
```

- 5) Add the following three lines to allow connection for authenticated users and tell Mosquitto where the username/password file is located.

```
allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd
```

Your configuration file will look as follows (the new lines are in bold):

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

per_listener_settings true

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d
allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd
```

```
GNU nano 6.2          /etc/mosquitto/mosquitto.conf *
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

per_listener_settings true
pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

allow_anonymous false
listener 1883
password_file /etc/mosquitto/passwd
```

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location
^X Exit ^R Read File ^Y Replace ^U Paste ^J Justify ^/ Go To Line

- 5) Press **CTRL-X**, then **Y**, and finally press **Enter** to exit and save the changes.
- 6) Restart Mosquitto for the changes to take effect.

```
sudo systemctl restart mosquitto
```

- 7) Wait a few seconds. Then, to check if Mosquitto is running, you can type the following command:

```
sudo systemctl status mosquitto
```

```
pi@raspberrypi:~ $ sudo mosquitto_passwd -c /etc/mosquitto/passwd sara
Password:
Reenter password:
pi@raspberrypi:~ $ sudo nano /etc/mosquitto/mosquitto.conf
pi@raspberrypi:~ $ sudo systemctl restart mosquitto
pi@raspberrypi:~ $ sudo systemctl status mosquitto
● mosquitto.service - Mosquitto MQTT Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor pre>
   Active: active (running) since Mon 2022-08-29 18:22:55 WEST; 27s ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
  Process: 13791 ExecStartPre=/bin/mkdir -m 740 -p /var/log/mosquitto (code=exit>
  Process: 13792 ExecStartPre=/bin/chown mosquitto /var/log/mosquitto (code=exit>
  Process: 13793 ExecStartPre=/bin/mkdir -m 740 -p /run/mosquitto (code=exit>
  Process: 13794 ExecStartPre=/bin/chown mosquitto /run/mosquitto (code=exit>
 Main PID: 13795 (mosquitto)
   Tasks: 1 (limit: 4164)
     CPU: 81ms
    CGroup: /system.slice/mosquitto.service
              └─13795 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Aug 29 18:22:55 raspberrypi systemd[1]: Starting Mosquitto MQTT Broker...
Aug 29 18:22:55 raspberrypi systemd[1]: Started Mosquitto MQTT Broker.
lines 1-17/17 (END)
```

Now, you have Mosquitto MQTT broker installed on the cloud with authentication with username and password enabled.

On your ESP32/ESP8266 Arduino code, on the MQTT Host, you should use your droplet IP address.

10.4 - InfluxDB on Digital Ocean

You can also install InfluxDB on the cloud so that you can access your data from anywhere.

Follow the next instructions to install InfluxDB on the cloud.

Run the following commands sequentially to install InfluxDB on your server.

```
wget -q https://repos.influxdata.com/influxdata-archive_compatible.key
```

```
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c
influxdata-archive_compatible.key' | sha256sum -c && cat influxdata-
archive_compatible.key | gpg --dearmor | sudo tee
/etc/apt/trusted.gpg.d/influxdata-archive_compatible.gpg > /dev/null
```

```
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compatible.gpg]
https://repos.influxdata.com/debian stable main' | sudo tee
/etc/apt/sources.list.d/influxdata.list
```

```
sudo apt-get update && sudo apt-get install influxdb2
```

When prompted, press **Y** and hit **Enter**.

Start InfluxDB by running the following command:

```
sudo service influxdb start
```

Check if InfluxDB is running:

```
sudo service influxdb status
```

The instructions are not working?

If your instructions are not working it might be due to an update on the installation instructions. If that's the case, we recommend that you go to the following link and follow the official instructions: <https://www.influxdata.com/downloads/>

InfluxDB 2.x Open Source Time Series Database

InfluxDB is an open source time series database. It has everything you need from a time series platform in a single binary – a multi-tenant dashboarding tools, background processing and monitoring agent. All this makes deployment and setup a breeze and easier to secure.

The InfluxDB Platform also includes APIs, tools, and an ecosystem that includes 10 client and server libraries, Telegraf plugins, visualization Studio, and data sources integrations with Google Bigtable, BigQuery, and more.

Version

InfluxDB v2.6.1

Platform

Ubuntu & Debian

```
# influxdata-archive_compat.key GPG fingerprint:  
#      9D53 9D90 D332 8DC7 D6C8 D3B9 D8FF 8E1F 7DF8 B07E  
wget -q https://repos.influxdata.com/influxdata-archive_compat.key  
echo '393e8779c89ac8d958f81f942f9ad7fb82a25e133faddaf92e15b16e6ac9ce4c influxdata-archive_compat.key' | sha256sum -c && cat influxd  
echo 'deb [signed-by=/etc/apt/trusted.gpg.d/influxdata-archive_compat.gpg] https://repos.influxdata.com/debian stable main' | sudo  
sudo apt-get update && sudo apt-get install influxdb
```

[Documentation](#) | [Release Notes](#) | [Register Download](#)

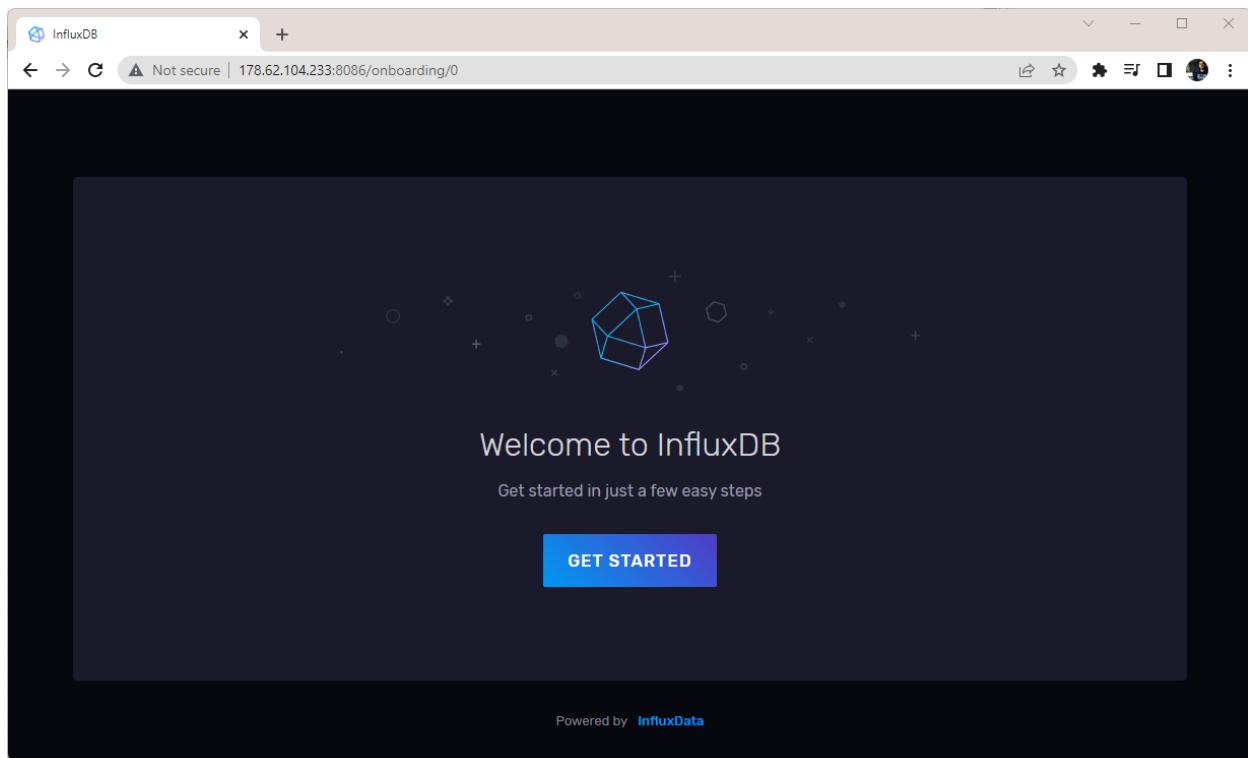
Make sure you select Ubuntu & Debian. Then, simply copy the commands provided.

Now, you can access InfluxDB on your server IP address, port 8086.

`http://<YOUR-Digital-Ocean-Droplet-IP-Address>:8086`

For example, in my case:

`http://178.62.83.231:8086`



Now, you have InfluxDB up and running on the cloud. To interface it with Node-RED (on the cloud or locally), you just need to use the cloud server IP address.

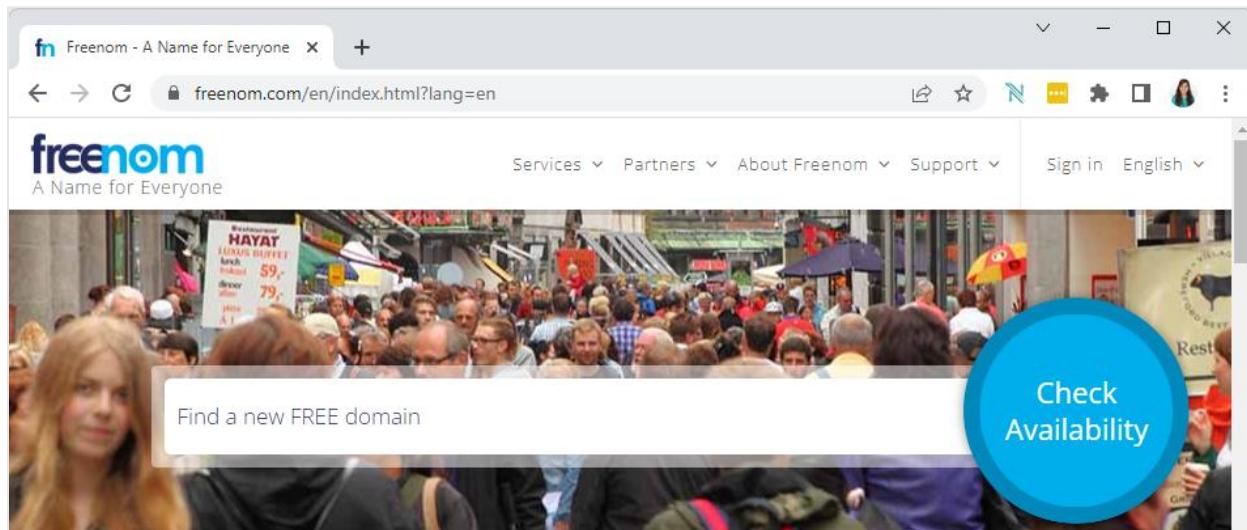
10.5 - Setting up a Custom Domain with Cloudflare (HTTPS)

There's an easy way to add HTTPS to your cloud server using Cloudflare Tunnel services (for free). To do that you need a domain name.

Getting a Domain Name

Cloudflare Tunnel creates a secure tunnel to your server so that you can access it from anywhere securely using a domain name. To create the tunnel, you need to get a domain name.

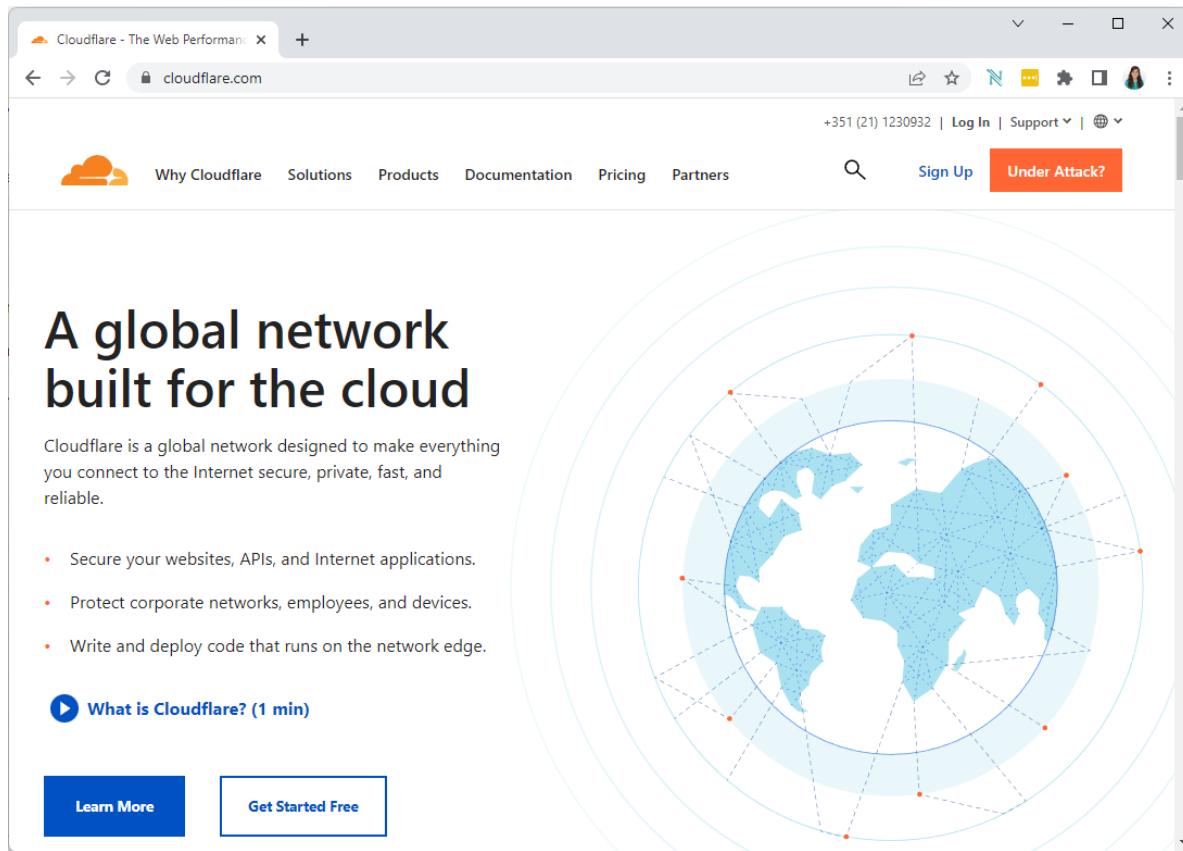
We'll use a domain name from [GoDaddy](#), but any domain name provider should work ([like Namecheap for example](#)). You can also grab a free domain name at [freenom.com](#). You can search and register a free domain name with several extensions (.tk, .ml, .ga, .cf, .gq).



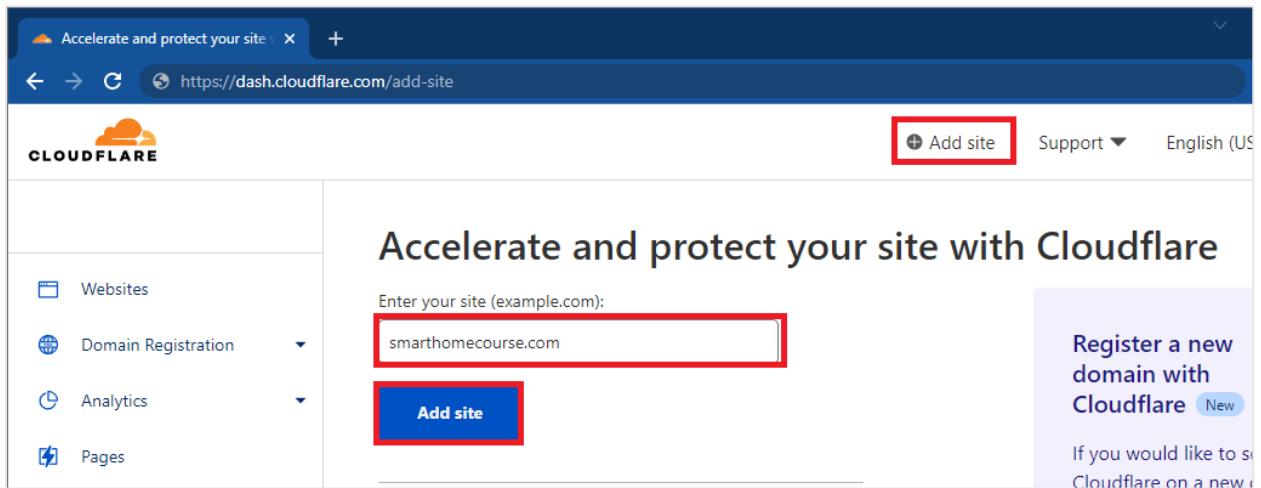
Please note that at the time of updating this eBook, freenom is no longer accepting new members, but that can change in the future.

Creating a Cloudflare Tunnel

Go to <https://www.cloudflare.com/> and create an account.



Click on **+Add site** or go to <https://dash.cloudflare.com/> and enter your domain name on the field. Ours is `smarthomecourse.com`. Then, click on **Add site**.

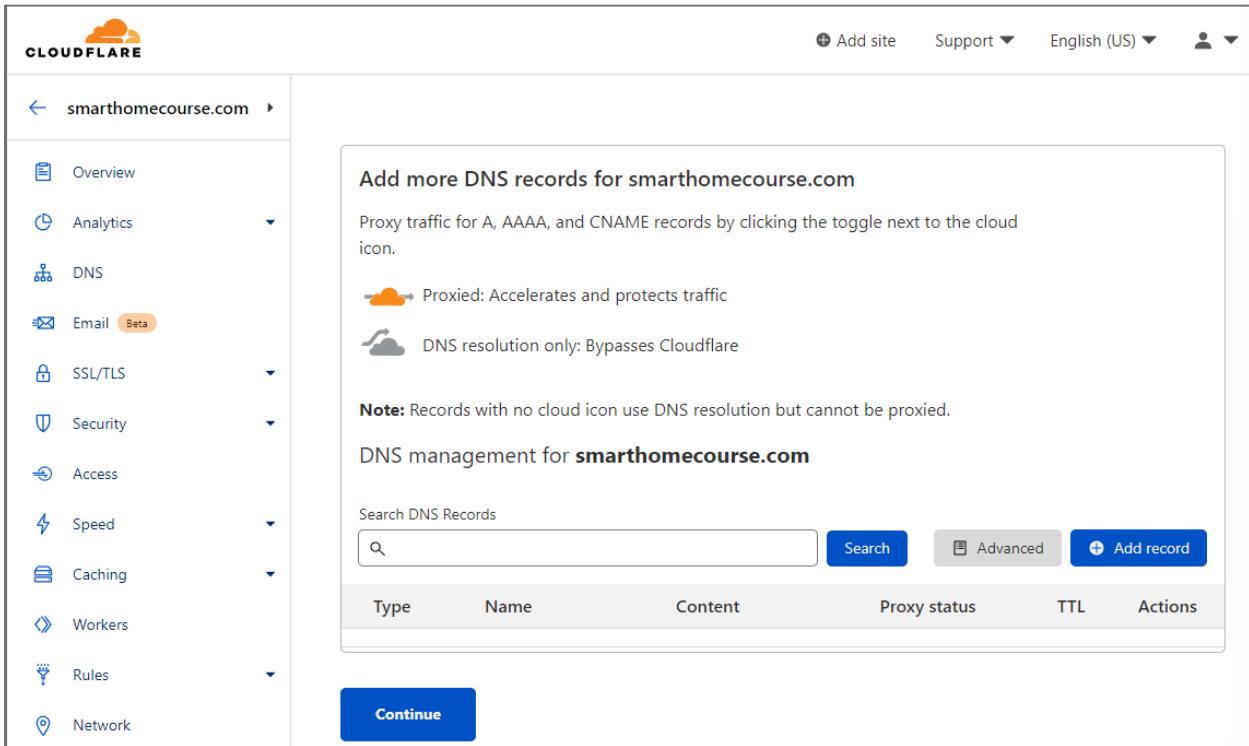


We're using a domain from GoDaddy, but if you're using Namecheap you can follow the [official instructions here](#).

Choose the **Free plan** at the bottom of the page. Then, click on **Continue**.

The screenshot shows the Cloudflare dashboard for the domain smarthomecourse.com. On the left, there's a sidebar with various services like Overview, Analytics, DNS, Email (Beta), SSL/TLS, Security, Access, Speed, Caching, Workers, Rules, Network, Traffic, Custom Pages, Apps, Scrape Shield, Zaraz (Beta), and Web3 (New). The main content area is titled "Select a plan". It shows three plans: Pro (\$20/month), Business (\$200/month), and Enterprise. A red box highlights the "Free" plan option, which is described as being built on a global network and ideal for personal or hobby projects. Below the plans, there's a section for "Not sure where to start? Get started for free." with a "Free \$0" button. A blue "Continue" button is also highlighted with a red box.

After that, click on the **Continue** button again (you don't need to fill any of those fields the moment).



The screenshot shows the Cloudflare dashboard for the domain `smarthomecourse.com`. The left sidebar contains navigation links: Overview, Analytics, DNS (selected), Email (Beta), SSL/TLS, Security, Access, Speed, Caching, Workers, Rules, and Network. The main content area is titled "Add more DNS records for `smarthomecourse.com`". It explains that proxy traffic for A, AAAA, and CNAME records can be managed by clicking a toggle next to the cloud icon. Two options are shown: "Proxied: Accelerates and protects traffic" (indicated by a cloud icon with a lightning bolt) and "DNS resolution only: Bypasses Cloudflare" (indicated by a cloud icon without a lightning bolt). A note states: "Records with no cloud icon use DNS resolution but cannot be proxied." Below this is a section titled "DNS management for `smarthomecourse.com`" with a search bar and buttons for "Search", "Advanced", and "Add record". A table header is visible with columns: Type, Name, Content, Proxy status, TTL, and Actions. At the bottom is a "Continue" button.

At this moment, you need to change your nameservers. You must proceed as explained on the Cloudflare instructions. You need to login into your domain name provider, find the DNS management settings and replace the referred nameservers with Cloudflare's nameservers.

The screenshot shows the Cloudflare dashboard for the domain 'smarthomecourse.com'. The left sidebar contains various management options like Overview, Analytics, DNS, Email, SSL/TLS, Security, Access, Speed, Caching, Workers, Rules, Network, Traffic, Custom Pages, Apps, Scrape Shield, Zaraz, and Web3. The main content area is titled 'Change your nameservers' and includes a note about pointing to Cloudflare's nameservers. It provides a step-by-step guide: 1. Log in to your GoDaddy account, 2. Remove the following nameservers (ns51.domaincontrol.com, ns52.domaincontrol.com), 3. Add Cloudflare's nameservers (sean.ns.cloudflare.com, shaz.ns.cloudflare.com), and 4. Save your changes. A note states that registrars can take up to 24 hours to process updates. A 'Done, check nameservers' button is at the bottom.

For example, in GoDaddy the edit nameservers menu look as follows.

The screenshot shows the 'Edit nameservers' interface in GoDaddy. It features a heading 'Edit nameservers' and a sub-section 'Introduce my own nameservers'. A warning message states: 'Changing nameservers is risky and can cause your website to no longer be publicly visible.' Below this, two nameserver entries are listed: 'sean.ns.cloudflare.com' and 'shaz.ns.cloudflare.com'. There is a link '+ Add name server' for adding more. At the bottom are three buttons: 'Cancel', 'Go back', and a prominent black 'Save' button.

Save the changes.

You need to wait 15 minutes or more for the domain nameserver to propagate. On the Cloudflare dashboard click on **Done, check nameservers**.

When it's done, under the **Access** menu (left sidebar), click on launch the **Zero Trust Dashboard**.

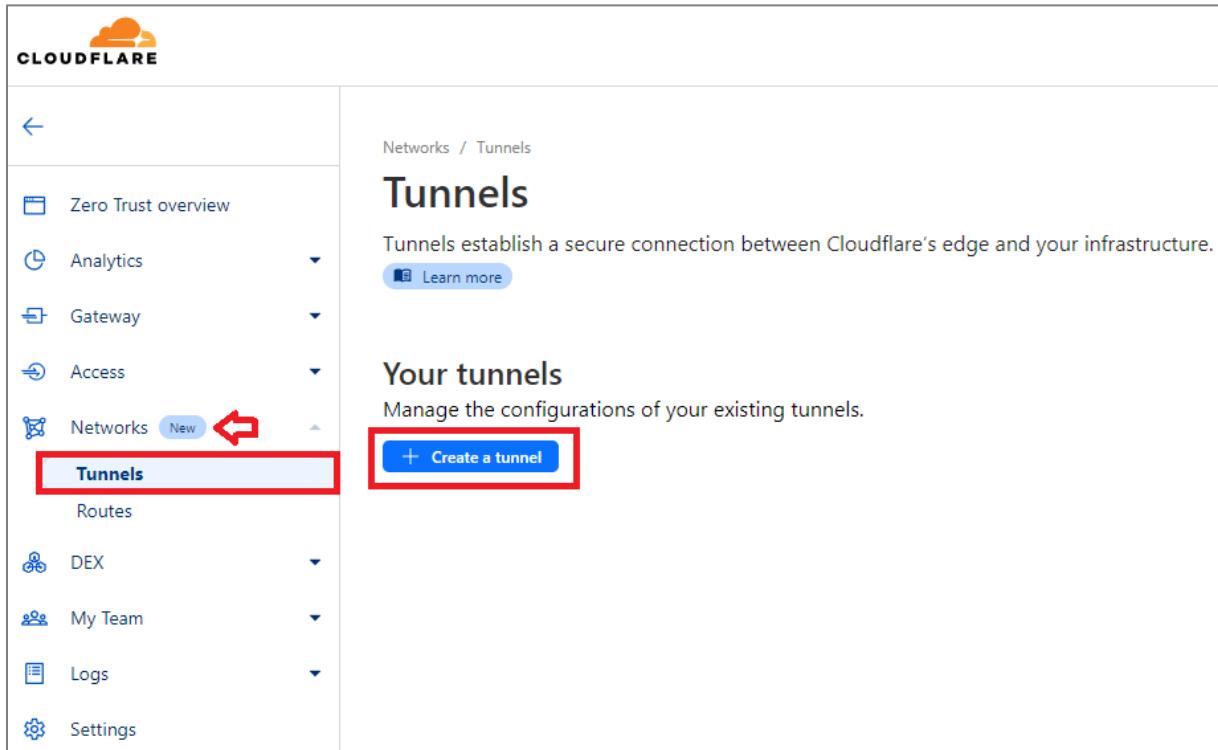
The screenshot shows the Cloudflare Access dashboard. The left sidebar has a red box around the 'Access' item. The main content area has a notice: 'Configuration for Access has moved to Cloudflare Zero Trust as of May 25, 2022. This page has been deprecated. All existing functionality, as well as new features, is available in Zero Trust.' A blue button labeled 'Launch Zero Trust' is visible. The top bar shows the domain 'smarthomecourse.com' is active, starred, and on a free plan.

Choose the **free plan** for the Cloudflare Zero Trust.

You'll be redirected to the Cloudflare Zero Trust dashboard:

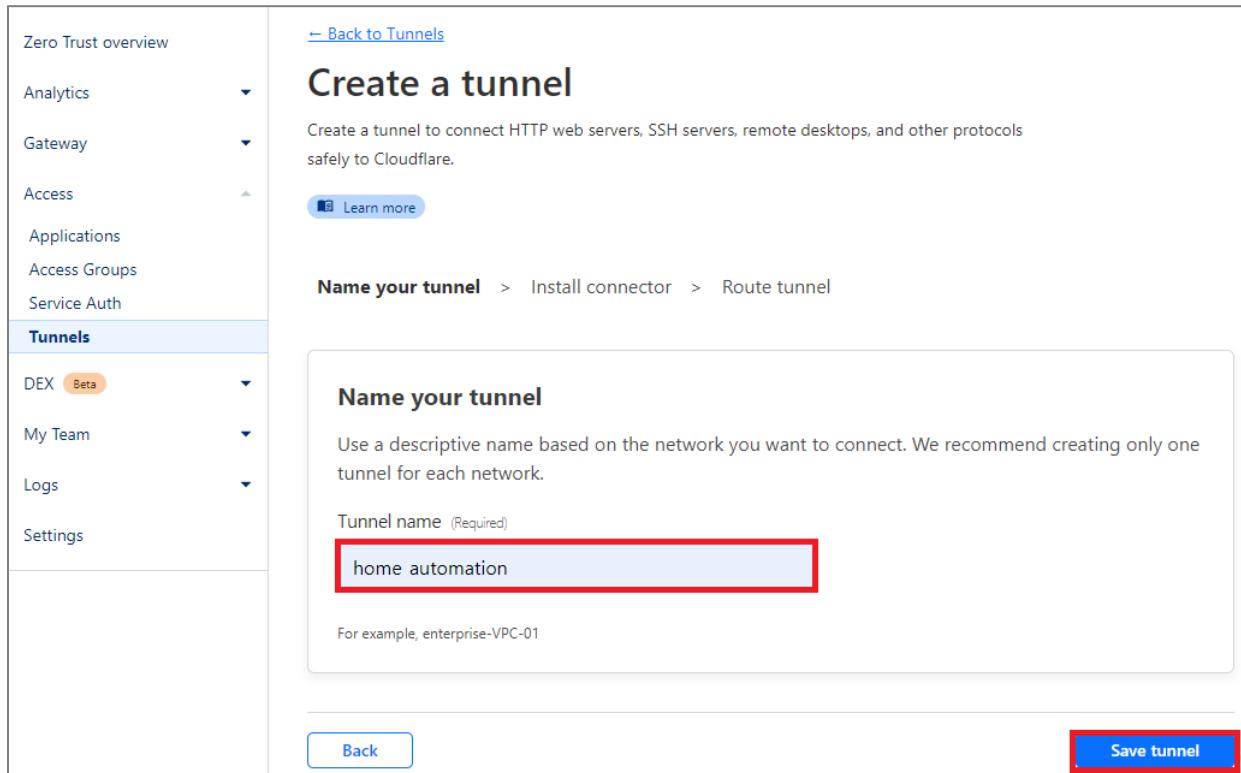
- <https://one.dash.cloudflare.com>

Then, under the **Networks** menu, select **Tunnels** and click on **Create a tunnel**.



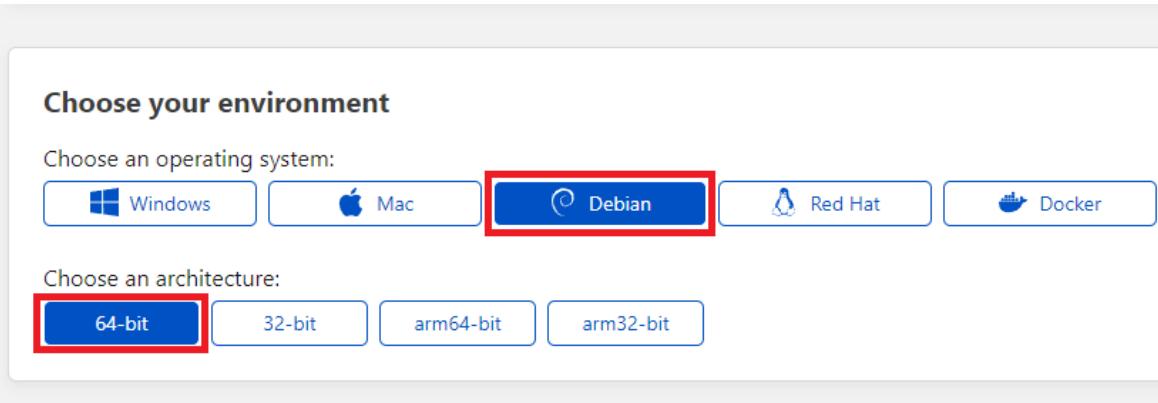
The screenshot shows the Cloudflare interface. On the left, there's a sidebar with various menu items: Zero Trust overview, Analytics, Gateway, Access, Networks (which has a 'New' button and a red arrow pointing to it), Tunnels (highlighted with a red box), Routes, DEX, My Team, Logs, and Settings. The main content area is titled 'Tunnels' and contains the sub-section 'Your tunnels'. It says 'Manage the configurations of your existing tunnels.' and features a blue 'Create a tunnel' button. The 'Create a tunnel' button is also highlighted with a red box.

Give a name to your tunnel, for example *home automation*, and click **Save tunnel**.



The screenshot shows the 'Create a tunnel' page. On the left, there's a sidebar with: Zero Trust overview, Analytics, Gateway, Access, Applications, Access Groups, Service Auth, **Tunnels** (selected and highlighted with a blue background), DEX (Beta), My Team, Logs, and Settings. The main content area has a 'Back to Tunnels' link. The title is 'Create a tunnel' with a sub-instruction 'Create a tunnel to connect HTTP web servers, SSH servers, remote desktops, and other protocols safely to Cloudflare.' Below this is a 'Learn more' button. A navigation bar at the bottom says 'Name your tunnel > Install connector > Route tunnel'. The 'Name your tunnel' section contains a 'Tunnel name (Required)' input field which has 'home automation' typed into it. A note below says 'For example, enterprise-VPC-01'. At the bottom are 'Back' and 'Save tunnel' buttons, with the 'Save tunnel' button highlighted with a red box.

Then, select **Debian** and **64-bit architecture**.



Scroll down and copy the whole command at the left.

A screenshot of the Cloudflare Zero Trust dashboard under the "Settings" tab. On the left, there's a sidebar with "My Team", "Logs", and "Settings". The "Settings" section is active. To its right is a panel titled "Install and run a connector". It contains instructions: "To connect your tunnel to Cloudflare, copy-paste one of the following commands into a terminal window. Remotely managed tunnels require that you install clouflare 2022.03.04 or later." Below this is a warning box: "Store your token carefully. This command includes a sensitive token that allows the connector to run. Anyone with access to this token will be able to run the tunnel." Two code snippets are provided: one for users without clouflare installed and one for users who already have it installed. The "curl" command in the first snippet is highlighted with a red box. At the bottom of the panel is a link "View Frequently Asked Questions".

Run that command on the console of your digital ocean droplet.

You should get a success message "*Linux service for clouflare installed successfully*".

Go back to Zero Trust dashboard, scroll down and click Next.

Connectors

| Connector ID | Status | Data centers | Origin IP | Version |
|-------------------------------------|-----------|--------------|---------------|----------|
| a96d8b03-7a70-41a5-b632-a77b51ad705 | Connected | AMS, LIS | 10.0.2.10.1.3 | 2022.8.2 |

Next

Now, fill in the details to access Node-RED on your domain name as shown below (but use your domain name instead).

- **Subdomain:** nodered
- **Domain:** your registered domain name
- **Path:** (leave it blank)
- **Service:** HTTP, localhost:1880

Route Traffic for home automation

Name your tunnel > Install connector > **Route tunnel**

Public Hostnames

Edit public hostname for home automation

Public hostname

Subdomain **Domain (Required)** **Path**

Service

Type (Required) **URL (Required)**

For example, <https://localhost:8001>

[Additional application settings ▶](#)

Back **Save tunnel**

Click on **Save tunnel**.

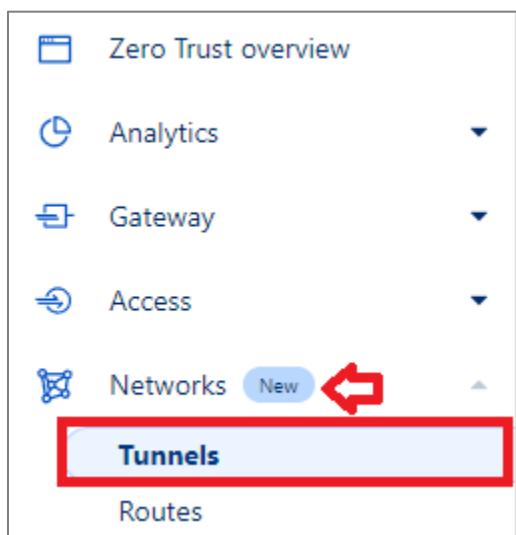
With these settings you'll be accessing Node-RED on the following:

`https://nodered.<your-domain-name>`

For example, in my case:

`https://nodered.smarthomecourse.com`

In the **Network**, open the **Tunnels** menu:



You'll see a list of the available tunnels.

The screenshot shows the 'Tunnels' page under 'Access / Tunnels'. The title is 'Tunnels'. A sub-header says 'Tunnels establish a secure connection between Cloudflare's edge and your infrastructure.' There is a 'Learn more' link. Below is a section titled 'Your tunnels' with the sub-instruction 'Manage the configurations of your existing tunnels.' A blue button '+ Create a tunnel' is visible. To the right is a search bar. The main area is a table with the following data:

| Tunnel name ↑ | Tunnel ID | Status | Routes | Uptime | Created | ⋮ |
|-----------------|-------------------------------------|---------|------------------------|--------|---------------|---|
| home automation | 4C171A0010m2...91L99L0...1111N1328b | HEALTHY | nodered.smarthomeco... | 2 days | July 12, 2023 | ⋮ |

Now, you need to configure your tunnel to add a subdomain to access InfluxDB. On the right corner click on **Configure**.

| | | | | | Search |
|--------|---------|---------------------|---------|---------------|---|
| Status | Routes | Uptime | Created | | |
| 228b | HEALTHY | nodered.smarthomeco | 2 days | July 12, 2023 | Configure Delete |

The following menu will open. Make sure you select the **Public Hostname** tab.

The screenshot shows the Cloudflare Tunnel Overview page for a tunnel named "home automation". The left sidebar includes links for Zero Trust overview, Analytics, Gateway, Access, Applications, Access Groups, and Service Auth. The main content area has tabs for Overview, Public Hostname (which is highlighted with a red box), and Private Network. Below the tabs, there is a section titled "Name your tunnel".

Click **Add a public hostname**.

The screenshot shows the Cloudflare Tunnel Public Hostnames page. The left sidebar now includes a "Tunnels" section under the "Access" category. The main content area shows a table of "Public hostnames" with one entry: "nodered.smarthomecourse.com". A blue button labeled "+ Add a public hostname" is highlighted with a red box. Below the table, a note says "Catch-all rule: http_status:404 Edit".

Then fill in the details as follows:

- **Subdomain:** influxdb
- **Domain:** your registered domain name
- **Path:** (leave it blank)
- **Service:** HTTP, localhost:8086

Edit public hostname for home-automation

Public hostname (Required)

| | | |
|-----------|---------------------|-----------------|
| Subdomain | Domain (Required) | Path |
| influxdb | smarthomecourse.com | (optional) path |

Service (Required)

| | | | |
|------|---|----|----------------|
| HTTP | : | // | localhost:8086 |
|------|---|----|----------------|

For example, https://localhost:8001

Additional application settings ▶

Save hostname

Finally, click on **Save hostname**.

With these settings you'll be accessing InfluxDB on the following:

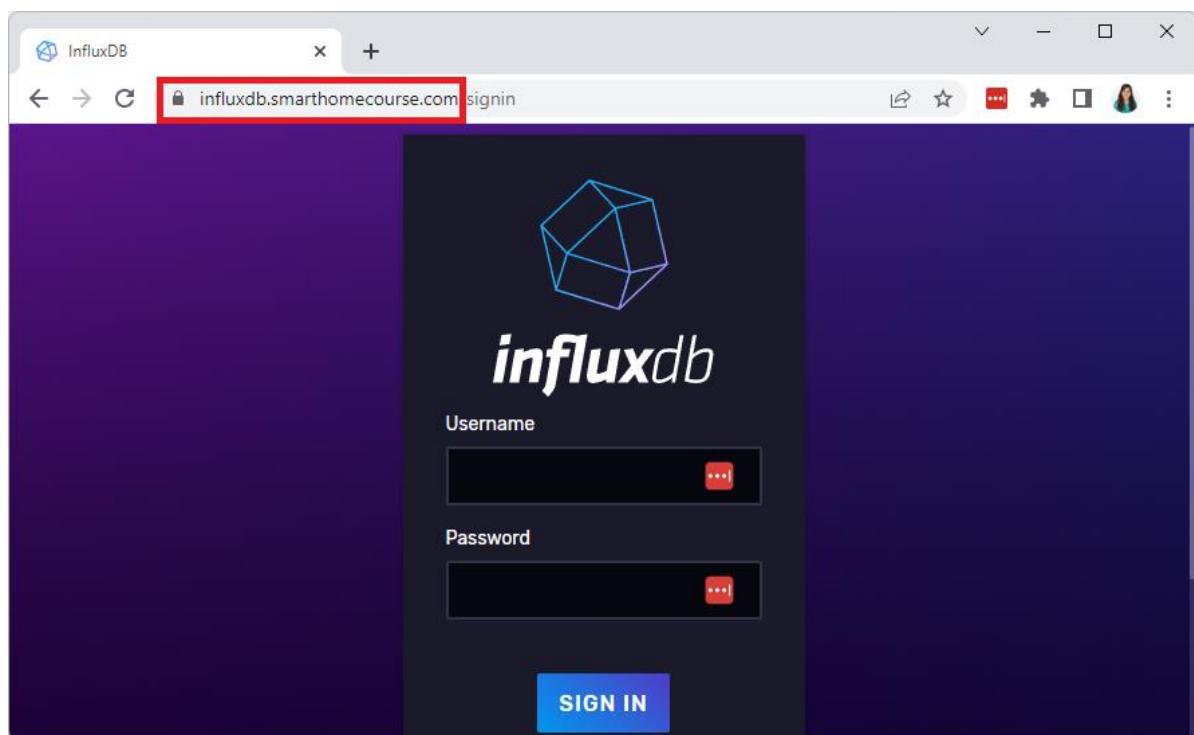
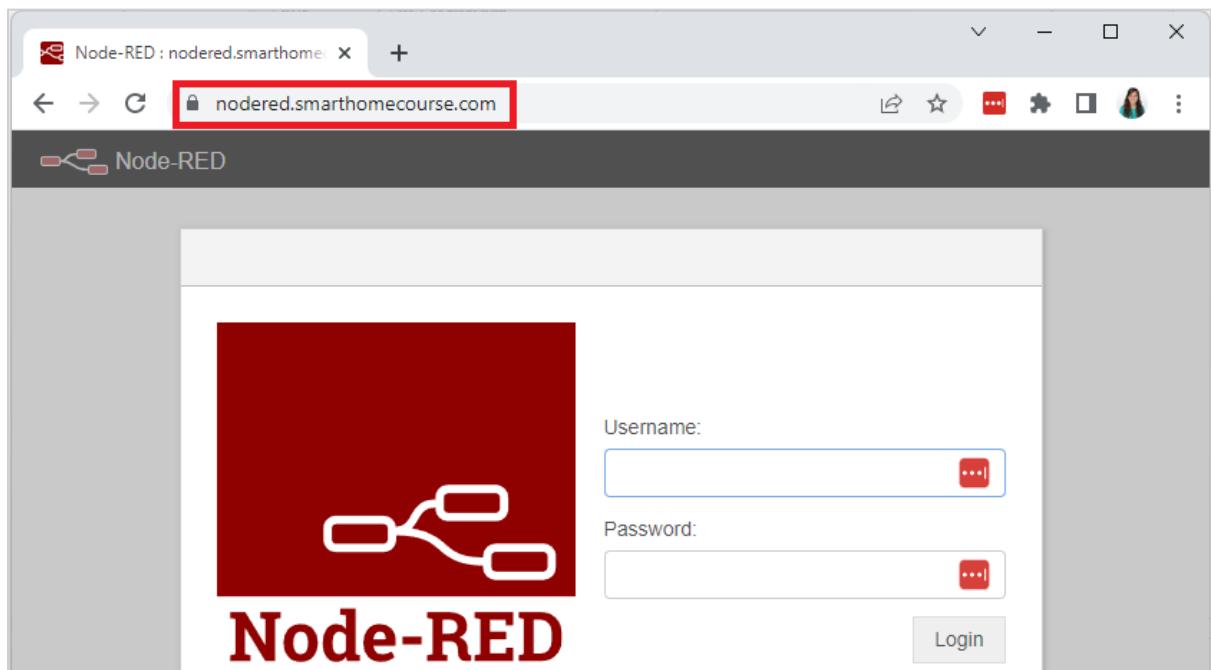
https://influxdb.<your-domain-name>

For example, in my case:

https://influxdb.smarthomecourse.com

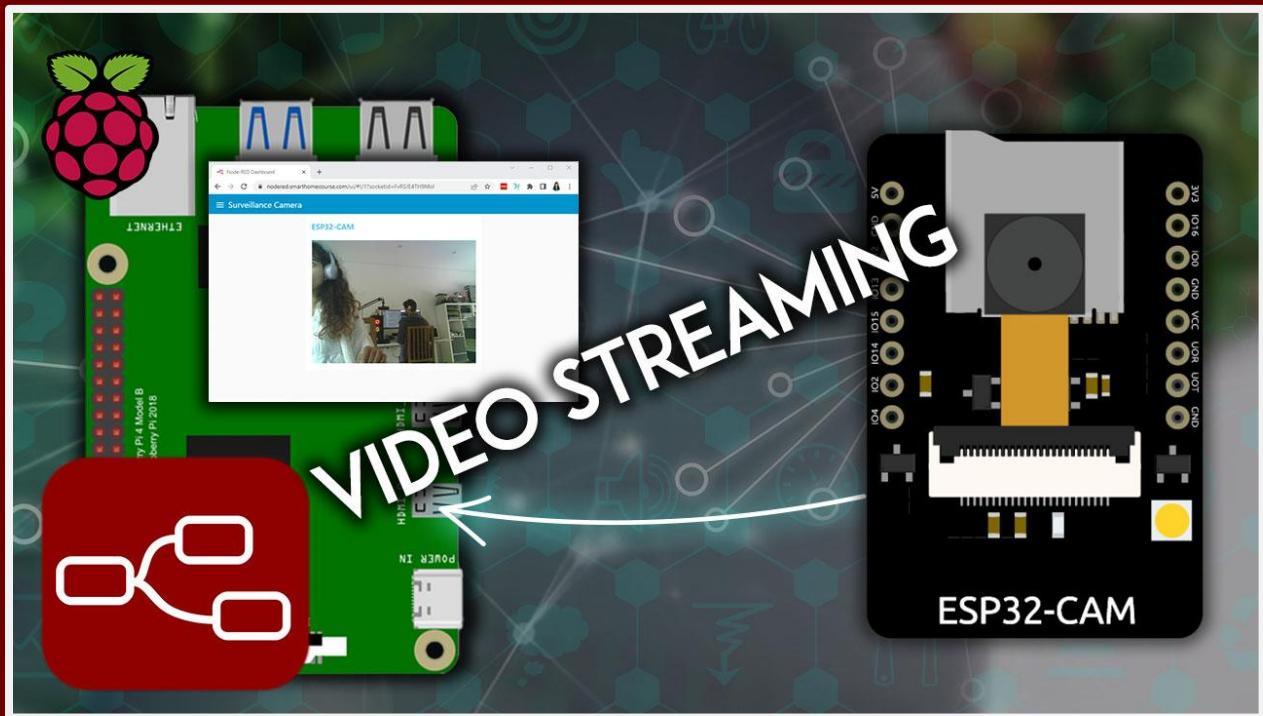
Wait a couple of minutes for the changes to take effect.

Now, you can access InfluxDB and Node-RED on your domain name from anywhere.



MODULE 11

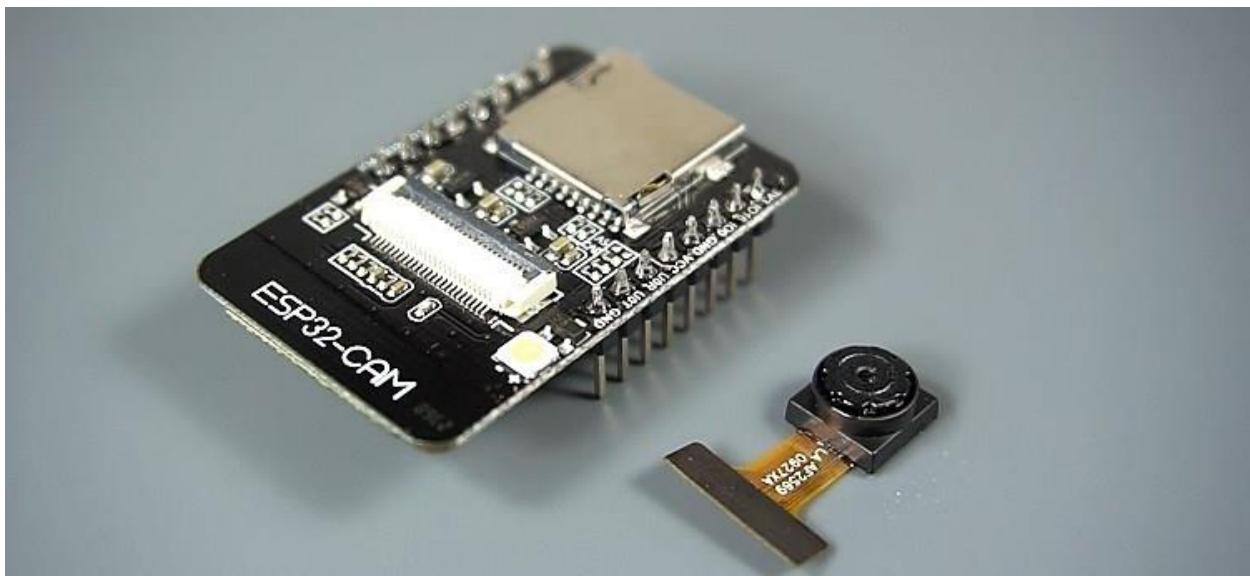
Setting up a Surveillance Camera



In this section, you'll learn how to add an ESP32-CAM surveillance camera to your Node-RED home automation system. The ESP32-CAM is a development board with an ESP32-S chip, an OV2640 camera, microSD card slot and several GPIOs to connect peripherals.

11.1 – Introducing the ESP32-CAM

This Unit is a quick introduction to the ESP32-CAM. To learn more about this board, you can check our [ESP32-CAM Projects eBook](#) or all our [ESP32-CAM projects here](#).



ESP32-CAM Introduction

The ESP32-CAM is a development board with an ESP32-S chip, an OV2640 camera, several GPIOs to connect peripherals, and a microSD card slot that can be useful to store images taken with the camera or to store files to serve to clients (web server).

It allows you to set up a video streaming web server, build a surveillance camera to integrate with your home automation system, do face detection and recognition, and many other applications in the Smart Home and Internet of Things (IoT) fields.

The ESP32-CAM is *just* another ESP32 development board with its own features:

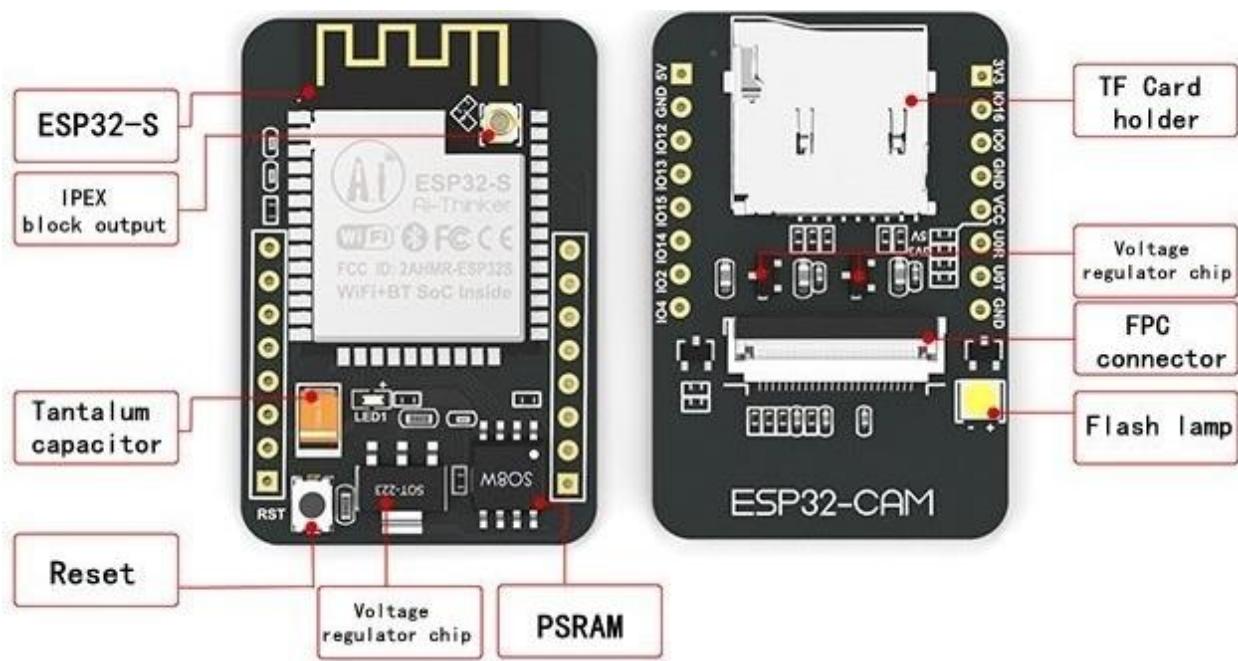
- OV2640 camera;
- No USB-to-UART interface;
- Reset button (labeled as RST or EN);
- 10 accessible GPIOs;

- 4MB PSRAM;
- MicroSD card interface.

Like with any regular ESP32 development board, you can program the ESP32-CAM using Arduino IDE and the Arduino core for the ESP32.

ESP32-CAM Features

The ESP32-CAM (AI-Thinker) has most of the specs of a regular ESP32 development board, but it has its own tweaks and features.



To keep in mind about the ESP32-CAM features:

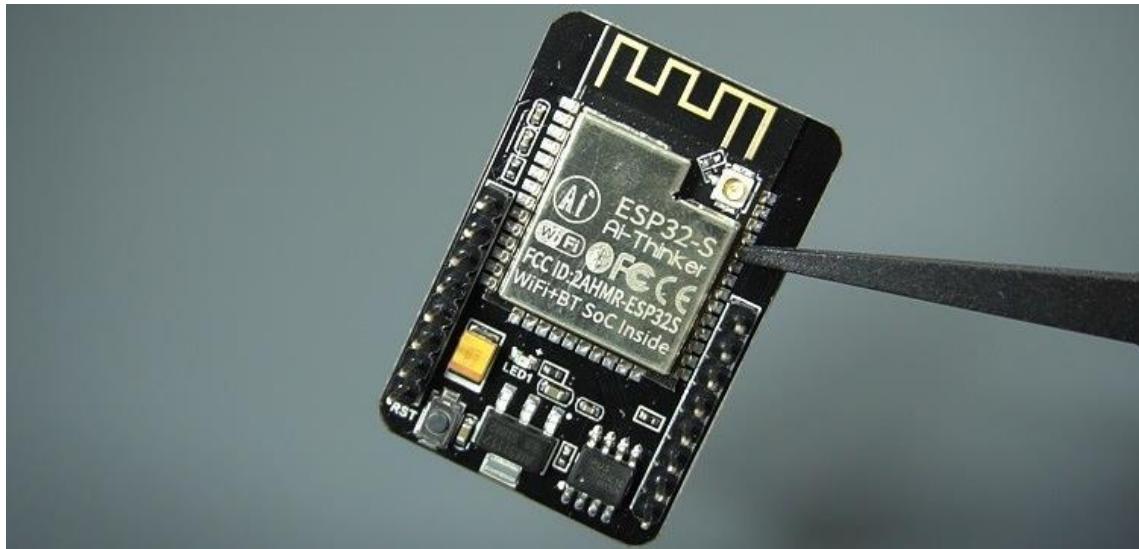
- The ESP32-CAM comes with an **OV2640 camera**. There are camera probes sold separately with longer flex cables and with a fish-eye lens. You connect the camera probes to the FPC connector.
- The **TF card holder** supports a microSD card interface. The ESP32-CAM datasheet mentions that it only supports microSD cards up to 4GB. However, we've experimented with microSD cards with other sizes (8GB and 16GB), and these worked well. Probably, they mean that it only supports writing up to 4GB.

(but we haven't tested this yet—you need a lot of photos with the ESP32-CAM to occupy 4GB).

- **No USB-to-UART interface:** this means that you can't connect the ESP32-CAM directly to your computer using a USB cable. You need to use a TTL to USB converter (FTDI programmer) or an ESP32-CAM MB programmer. There are other ESP32 camera boards (not ESP32-CAM AI-Thinker) that come with a USB connector.
- **On-board RESET button:** press this button to reset (restart) the ESP32-CAM.
- **On-board antenna and IPEX connector for external antenna:** the ESP32-CAM board comes with an on-board antenna and an IPEX connector that allows you to use an external antenna to improve Wi-Fi communication.
- **On-board LED (flashlight):** the ESP32-CAM comes with a very bright on-board LED. It can be handy to add some light to your pictures or video streaming.
- **Accessible GPIOs:** the ESP32-CAM comes with 10 accessible GPIOs. However, not all GPIOs can be used. For example, GPIO 0 is used to put the board in flashing mode, so you can't use it for other tasks. Additionally, some pins are being used either by the camera or by the microSD card. So, you need to be careful with which GPIOs you'll use.
- The ESP32-CAM has **PSRAM:** PSRAM is used for buffering images from the camera into video streaming or other tasks and allows you to use higher quality in your pictures without crashing the ESP32.

Buying an ESP32-CAM

Several different boards feature an ESP32 chip and a camera. One of the most popular and cheapest ESP32 boards with a camera is the ESP32-S AI-Thinker and that's the one we'll use.



There are other similar ESP32-CAM boards that look precisely like this one but don't have the "AI-THINKER" label. These should be perfectly suited too.

If you don't have an ESP32-CAM yet, we recommend getting one that comes with a programmer board. The vendors usually refer to the programmer board as **ESP32-CAM-MB programmer**.



Besides the AI-THINKER ESP32-CAM, other ESP32 camera development boards can be used with the code we'll provide in this module.

You'll find a brief description of some of the most popular ESP32 camera development boards on the following article:

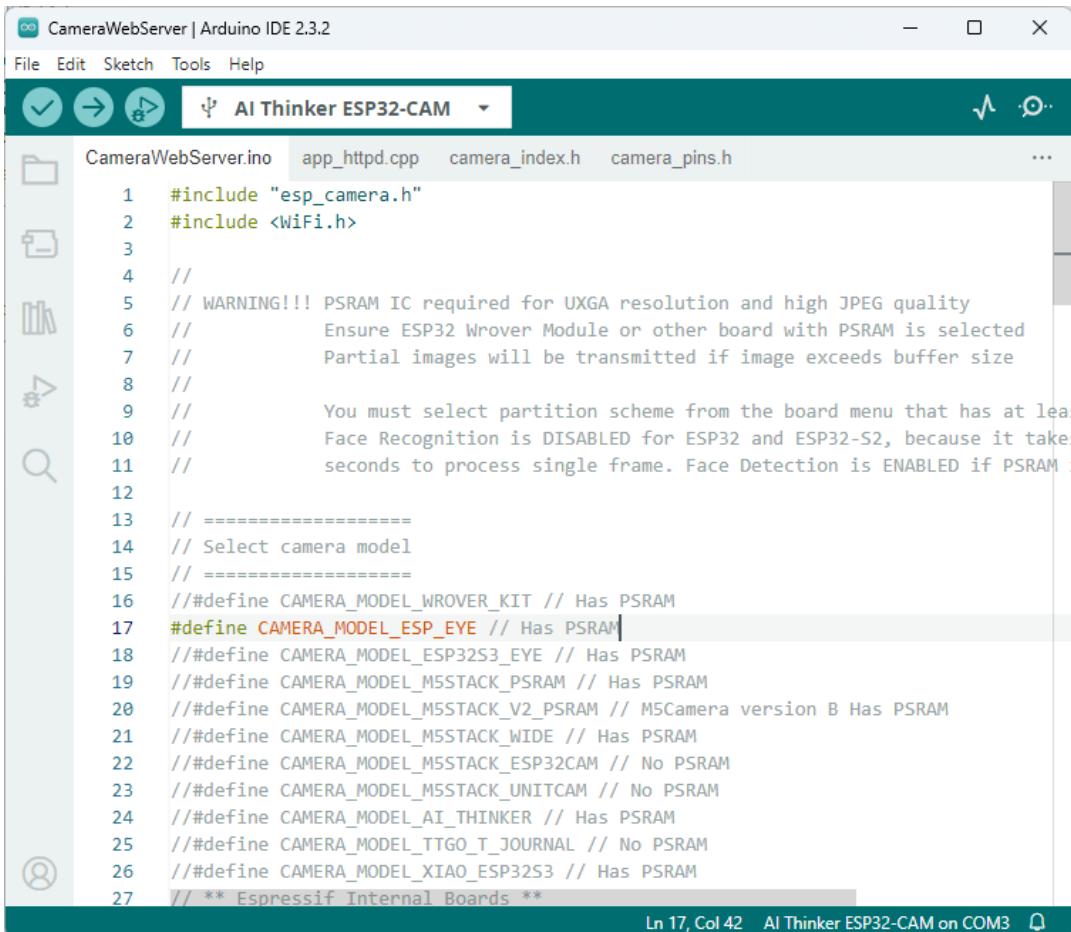
- [ESP32 Camera Dev Boards Review and Comparison \(Best ESP32-CAM\)](#)

11.2 – ESP32-CAM Video Streaming

In this unit, you'll upload a code to build an ESP32-CAM video streaming server. The video will be streaming on a specific URL and port. Then, you'll be able to embed this video streaming on Node-RED by referring to its URL.

Video Streaming Code

To turn the ESP32-CAM into a video streaming web server, we'll use an example available in the Arduino IDE *examples* folder. Open your Arduino IDE and go to **File > Examples > ESP32 > Camera > CameraWebServer**. Several tabs will open with the code.



The screenshot shows the Arduino IDE interface with the title bar "CameraWebServer | Arduino IDE 2.3.2". The menu bar includes File, Edit, Sketch, Tools, and Help. The toolbar has icons for Save, Run, Stop, and Refresh. The tab bar shows "AI Thinker ESP32-CAM" with a dropdown arrow. The code editor displays the "CameraWebServer.ino" file. The code is as follows:

```
1 #include "esp_camera.h"
2 #include <WiFi.h>
3
4 //
5 // WARNING!!! PSRAM IC required for UXGA resolution and high JPEG quality
6 // Ensure ESP32 Wrover Module or other board with PSRAM is selected
7 // Partial images will be transmitted if image exceeds buffer size
8 //
9 // You must select partition scheme from the board menu that has at least
10 // Face Recognition is DISABLED for ESP32 and ESP32-S2, because it takes
11 // seconds to process single frame. Face Detection is ENABLED if PSRAM is
12 //
13 // =====
14 // Select camera model
15 // =====
16 //#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
17 #define CAMERA_MODEL_ESP_EYE // Has PSRAM
18 //#define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
19 //#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
20 //#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B Has PSRAM
21 //#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
22 //#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
23 //#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
24 //#define CAMERA_MODEL_AI_THINKER // Has PSRAM
25 //#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM
26 //#define CAMERA_MODEL_XIAO_ESP32S3 // Has PSRAM
27 // ** Espressif Internal Boards **
```

The status bar at the bottom right shows "Ln 17, Col 42 AI Thinker ESP32-CAM on COM3".

You need to select your camera model on the code. Comment the `CAMERA_MODEL_ESP_EYE` and uncommend `CAMERA_MODEL_AI_THINKER` (or whatever model you're using)—see the picture below.

```
10 // Face Recognition is DISABLED for ESP32 and ESP32-S2, because
11 // seconds to process single frame. Face Detection is ENABLED in
12 //
13 // =====
14 // Select camera model
15 // =====
16 //#define CAMERA_MODEL_WROVER_KIT // Has PSRAM
17 //#define CAMERA_MODEL_ESP_EYE // Has PSRAM
18 //#define CAMERA_MODEL_ESP32S3_EYE // Has PSRAM
19 //#define CAMERA_MODEL_M5STACK_PSRAM // Has PSRAM
20 //#define CAMERA_MODEL_M5STACK_V2_PSRAM // M5Camera version B Has PSRAM
21 //#define CAMERA_MODEL_M5STACK_WIDE // Has PSRAM
22 //#define CAMERA_MODEL_M5STACK_ESP32CAM // No PSRAM
23 //#define CAMERA_MODEL_M5STACK_UNITCAM // No PSRAM
24 #define CAMERA_MODEL_AI_THINKER // Has PSRAM
25 //#define CAMERA_MODEL_TTGO_T_JOURNAL // No PSRAM
26 // ** Espressif Internal Boards **
27 //#define CAMERA_MODEL_ESP32_CAM_BOARD
28 //#define CAMERA_MODEL_ESP32S2_CAM_BOARD
29 //#define CAMERA_MODEL_ESP32S3_CAM_LCD
```

Scroll a bit more, and you'll find a section where you need to insert your network credentials.

```
// =====
// Enter your WiFi credentials
// =====
const char* ssid = "*****";
const char* password = "*****";
```

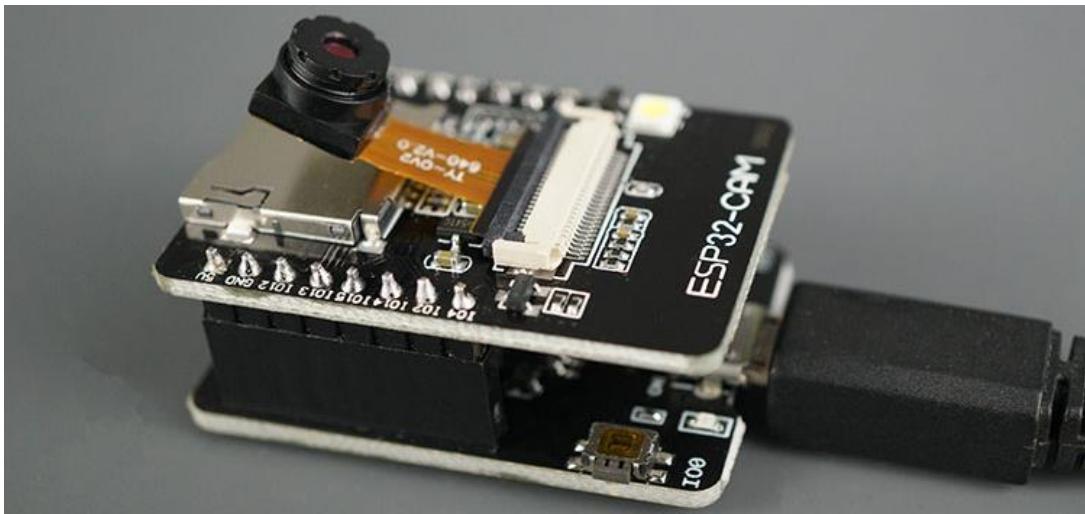
After inserting your network credentials you're ready to upload the code to your board.

Upload Code to the ESP32-CAM

To upload code to the ESP32-CAM you can use an ESP32-CAM MB programmer, or an FTDI programmer.

ESP32-CAM MB Programmer

If you're using an MB programmer, simply connect the MB programmer to the ESP32-CAM as shown in the following image.



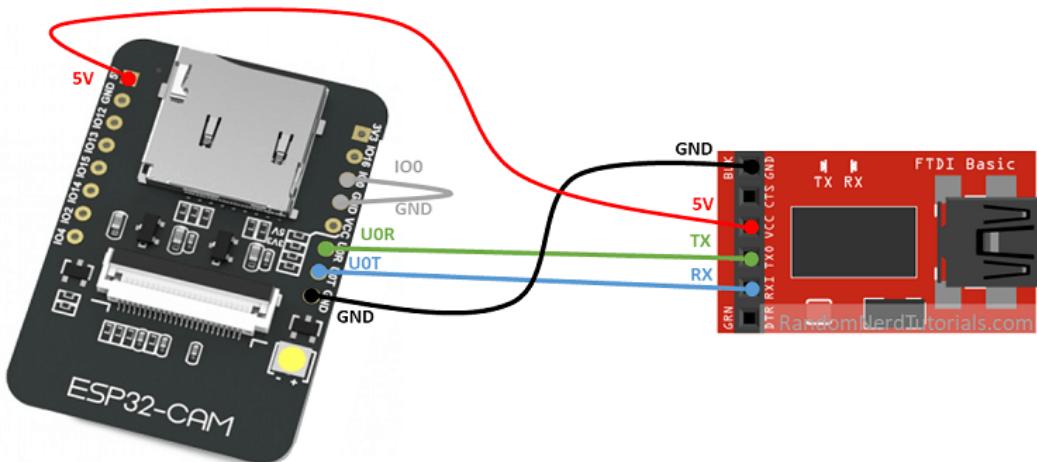
Then, connect the board to your computer using a USB cable.

For more details about uploading code using the ESP32-CAM MB programmer, take a look at the following article:

- [Upload Code to ESP32-CAM AI-Thinker using ESP32-CAM-MB USB Programmer](#)

FTDI Programmer

If you're using an FTDI programmer, you need to make the following connections:



Important: GPIO 0 needs to be connected to GND so that you're able to upload code.

| ESP32-CAM | FTDI Programmer |
|-----------|-----------------|
| GND | GND |
| 5V | VCC (5V) |
| U0R | TX |
| U0T | RX |
| GPIO 0 | GND |

Then, connect the FTDI programmer to your computer.

For more information about programming the ESP32-CAM using an FTDI programmer, take a look at the following article:

- [How to Program / Upload Code to ESP32-CAM AI-Thinker \(FTDI programmer\)](#)

Uploading the Code

To upload code to the ESP32-CAM using Arduino IDE, follow the next steps:

1. Go to **Tools > Board** and select **AI-Thinker ESP32-CAM**.
2. Go to **Tools > Port** and select the COM port the ESP32-CAM is connected to.
3. Then, click the **Upload** button in your Arduino IDE.



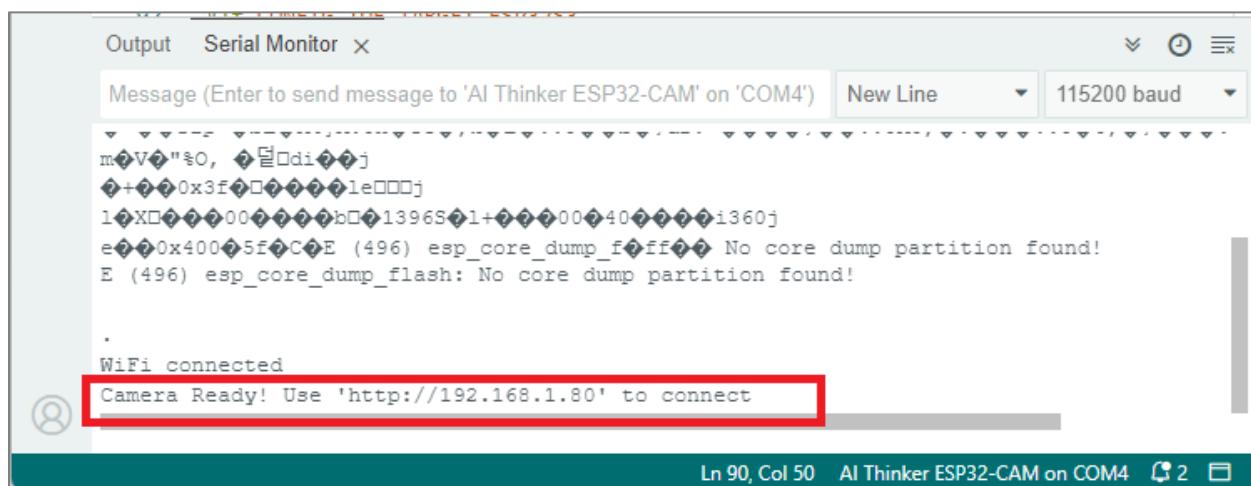
4. If you're using an FTDI programmer, you need to press the ESP32-CAM on-board RST button when you start to see some dots on the debugging window. If you're using an MB programmer, the code will upload automatically.

```
esptool.py v2.6-beta1
Serial port COM10
Connecting.....
```

5. After a few seconds, the code should be successfully uploaded to your board.
6. If you're using an FTDI programmer, remove GPIO 0 from GND after the "Done uploading message".

Getting the Video Streaming Server Address

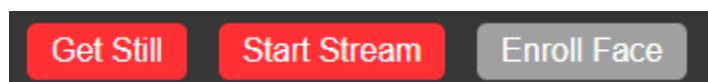
Open the Serial Monitor at a baud rate of 115200. Press the ESP32-CAM reset button. It will show the board streaming server IP address.



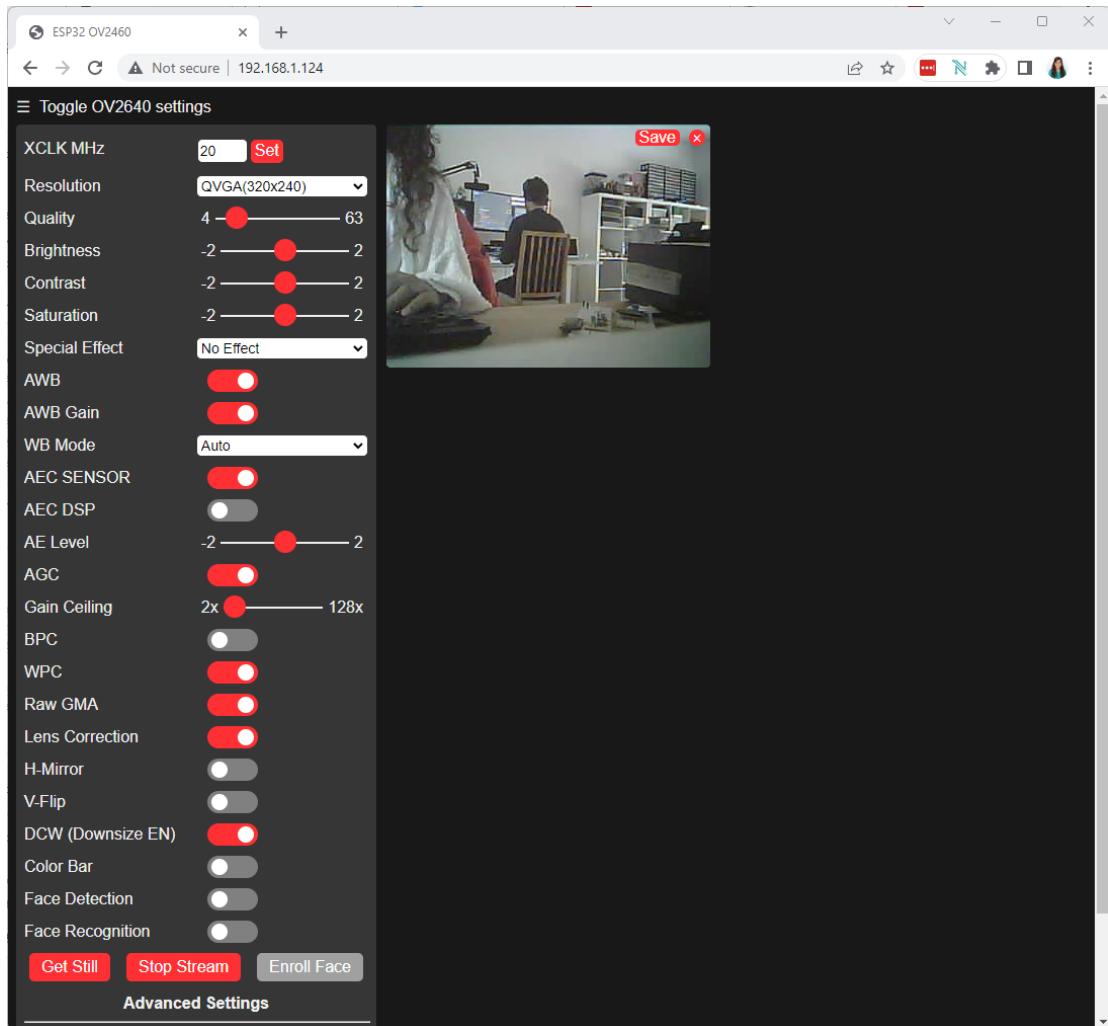
You may get a *No core dump partition found* warning. You can ignore it.

Open your browser and go to that IP address.

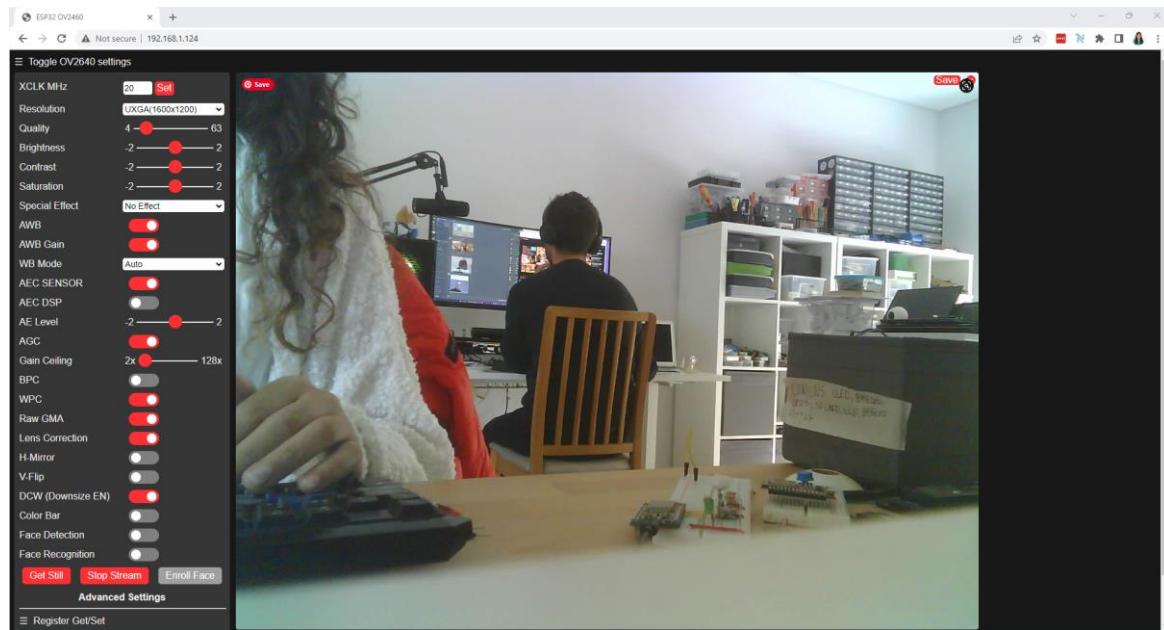
You'll see a web page with a left sidebar where you can adjust the camera image settings. Click on the red button at the bottom **Start Stream**.



The camera will start streaming. You can play with the controls to see which settings will give you a better picture.



Usually, changing the resolution to UXGA will give a much better picture.



If you right click on the image and click on *Open image in a new tab*, you'll be redirected to the following URL:

```
http://ESP32-CAM-IP-ADDRESS:81/stream
```

For example, in my case:

```
http://192.168.1.124:81/stream
```

This means the actual streaming source is on that URL. Save that URL because you'll need it later in the next Unit.

Close those web browser windows and proceed to the next section.

Important: the ESP32-CAM can only serve video streaming to one client at a time, so it is very important that you close the web browser window before proceeding. Otherwise, you won't see any video streaming on Node-RED later on.

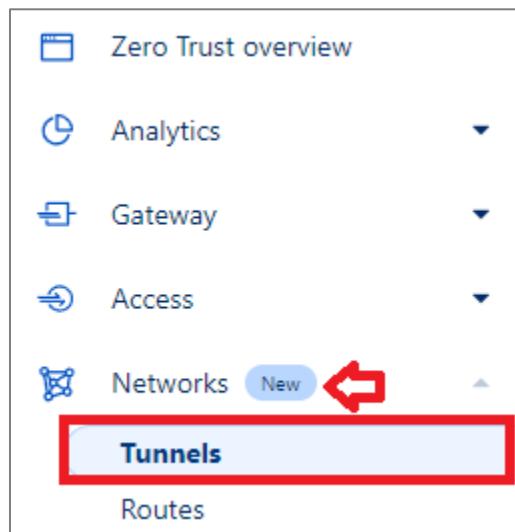
11.3 – Surveillance Camera on Node-RED

In the previous Unit, you got the ESP32-CAM streaming video on a specific URL. Now, we'll embed that URL on Node-RED UI so that you can have a surveillance camera on your home automation dashboard.

If you're using Cloudflare with your Node-RED system, you'll need to create a new tunnel to the video streaming URL. If you're just using Node-RED locally, without Cloudflare, you don't need to do that, so you can skip the following section.

Creating a Tunnel to the Video Streaming URL

On your Cloudflare account, go to your **Cloudflare Zero Trust** dashboard. You can open this link <https://one.dash.cloudflare.com/>, then on the left sidebar click on **Network**, open the **Tunnels** menu:



You should see a list of your current tunnels. Configure your tunnel.

The screenshot shows the 'Tunnels' section of the Cloudflare dashboard. It lists one tunnel named 'home automation'. The tunnel details are as follows:

| Tunnel name | Tunnel ID | Status | Routes | Uptime | Created |
|-----------------|--------------------------------------|---------|---------------------|--------|---------------|
| home automation | 4c07f300-10e2-4811-98c0-4f11143f228b | HEALTHY | nodered.smarthomeco | 2 days | July 12, 2023 |

Actions available for this tunnel include a blue 'Configure' button and a red 'Delete' button. A red box highlights the 'Configure' button.

The following menu will open. Make sure you select the **Public Hostname** tab.

The screenshot shows the 'home automation' tunnel overview page. The navigation bar on the left includes 'Zero Trust overview', 'Analytics', 'Gateway', and 'Access'. The main content area shows the tunnel name 'home automation' and three tabs: 'Overview' (selected), 'Public Hostname' (highlighted with a red box), and 'Private Network'.

Click on **Add a public hostname** to add a new tunnel to the video streaming.

The screenshot shows the 'Public hostnames' section of the 'home automation' tunnel configuration. It displays two entries:

| # | Public hostname | Path | Service | Origin configurations |
|---|------------------------------|------|-----------------------|-----------------------|
| 1 | nodered.smarthomecourse.com | * | http://localhost:1880 | 0 |
| 2 | influxdb.smarthomecourse.com | * | http://localhost:8086 | 0 |

A red box highlights the '+ Add a public hostname' button at the top of the list.

The following window will open.

Edit public hostname for home-automation

Public hostname

Subdomain **Domain** (Required) **Path** (optional) path

Service

Type (Required) **URL** (Required)

For example, https://localhost:8001

Additional application settings ▶

Save hostname

Fill in the details as shown:

- **Subdomain:** we'll call it `esp32cam`, but you can give it any other name. For example, if you intend to add more cameras in the future, you can call this `camera1`, or `camerakitchen`—you get the idea.
- **Domain:** your domain name
- **Path:** leave it empty
- **Type:** HTTP
- **URL:** your ESP32-CAM IP address followed by `:81`. For example, in my case, it is `192.168.1.124:81`

Click on **Save hostname**.

Now, the new tunnel will show up on your list of tunnels.

| # | Public hostname | Path | Service | Origin configurations | ⋮ |
|---|--|------|--------------------------------------|-----------------------|---|
| 1 | nodered.smarthomecourse.com | * | <code>http://localhost:1880</code> | 0 | ⋮ |
| 2 | influxdb.smarthomecourse.com | * | <code>http://localhost:8086</code> | 0 | ⋮ |
| 3 | esp32cam.smarthomecourse.com | * | <code>http://192.168.1.124:81</code> | 0 | ⋮ |

Embed the Video Streaming in Node-RED Dashboard

To embed the video streaming in Node-RED dashboard, we'll use a dashboard node called **template**. This node allows you to add custom HTML templates to your dashboard. We'll simply use this node to embed the video streaming.



Drag a template node to the flow. We're using the **dashboard template** node and **not the function template** node. Double-click on the **template** node to edit its properties.



We'll create a new dashboard tab and group to place the video streaming, but you can add it to an existing dashboard tab and group. We created a new tab called **Surveillance Camera** with a group called **ESP32-CAM**.

Make sure the **Template type** is set to **Widget in group**.

Enter the following HTML on the **Template** section. You need to replace the **<ESP32-CAM-STREAMING-URL>** with your video streaming URL.

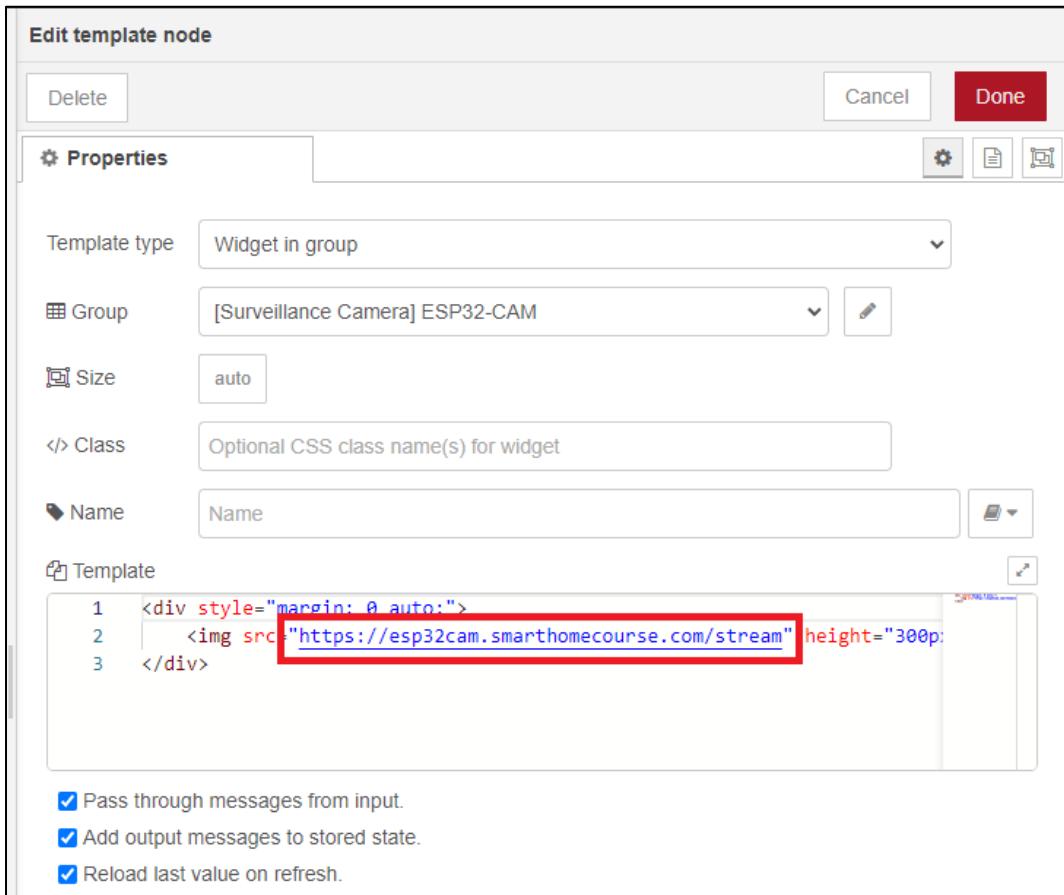
```
<div style="margin: 0 auto;">
  <img src=<ESP32-CAM-STREAMING-URL>/stream" height="300px">
</div>
```

If you're using Cloudflare, it corresponds to the tunnel you created previously. For example, in my case:

```
<div style="margin: 0 auto;">
  
</div>
```

If you're just running Node-RED locally, the URL should be the ESP32-CAM IP address followed by :81. For example, in my case, it would be:

```
<div style="margin: 0 auto;">
  
</div>
```



If you want to rotate the image 90°, copy the following HTML instead:

```
<div style="margin: 0 auto;">
  <img src=<ESP32-CAM-STREAMING-URL>/stream" height = "400px" style="transform:rotate(90deg);">
</div>
```

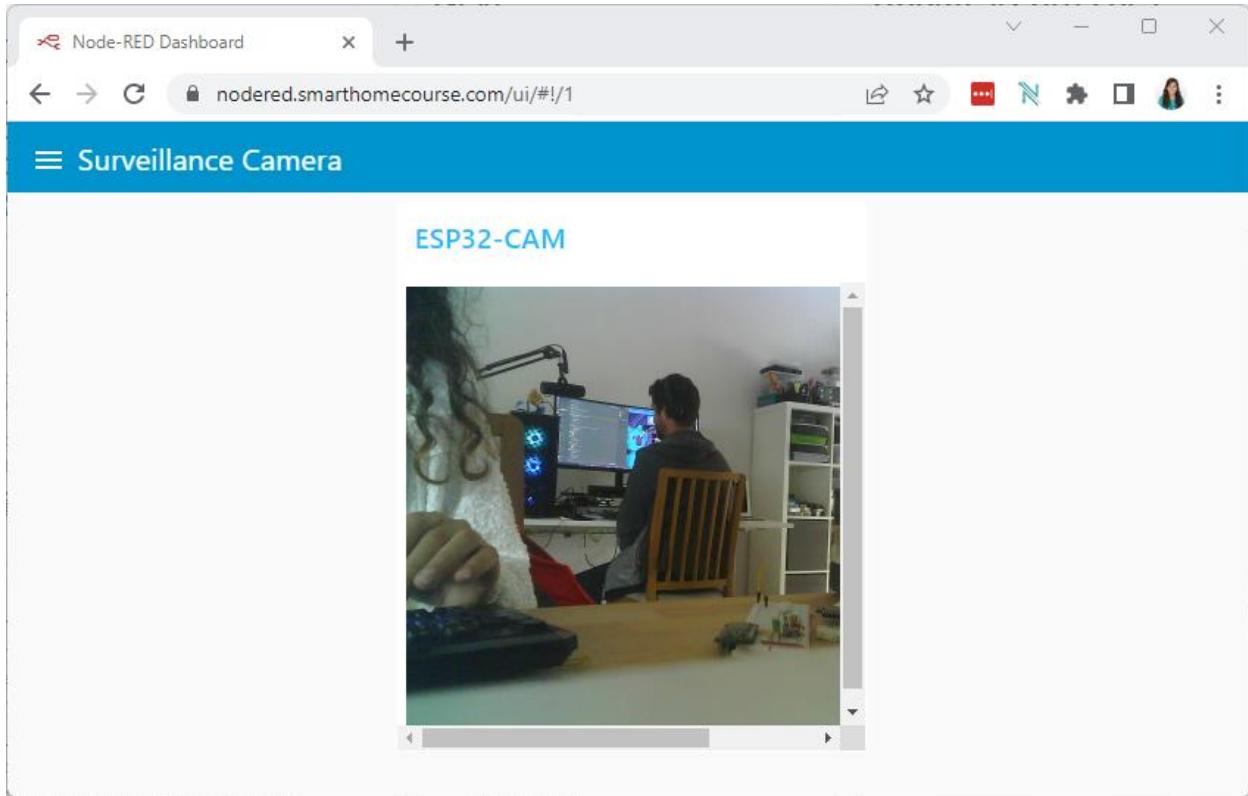
When you're finished, click **Done**.

Next, deploy your app.

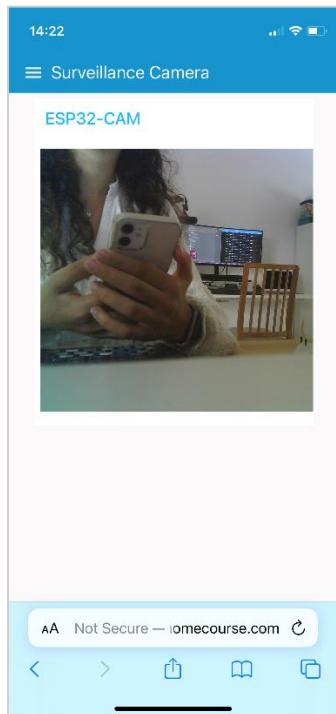


Go to your Node-RED UI and select the tab where you placed the template widget.

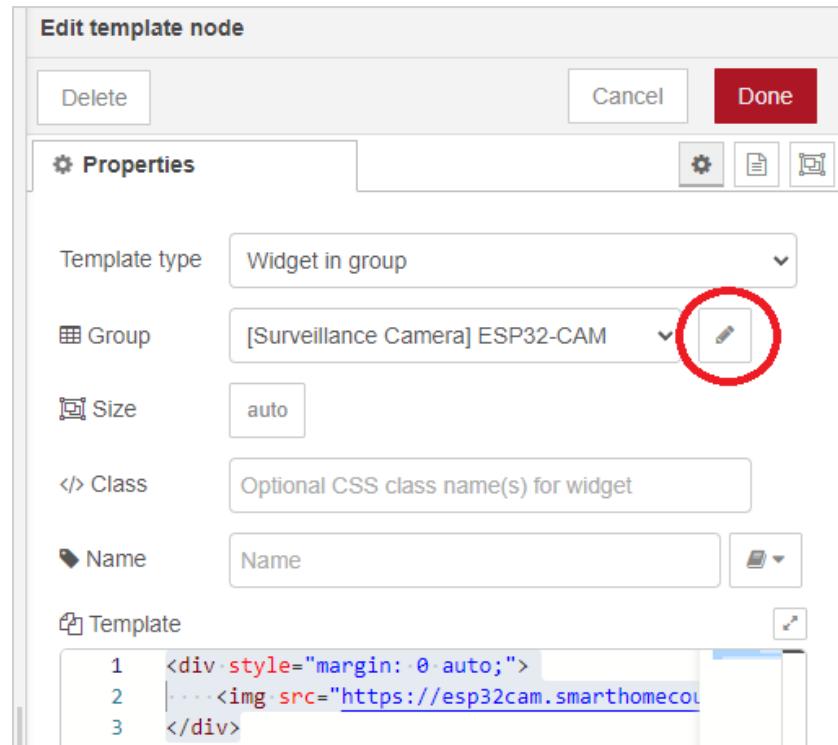
You'll get access to the video streaming.



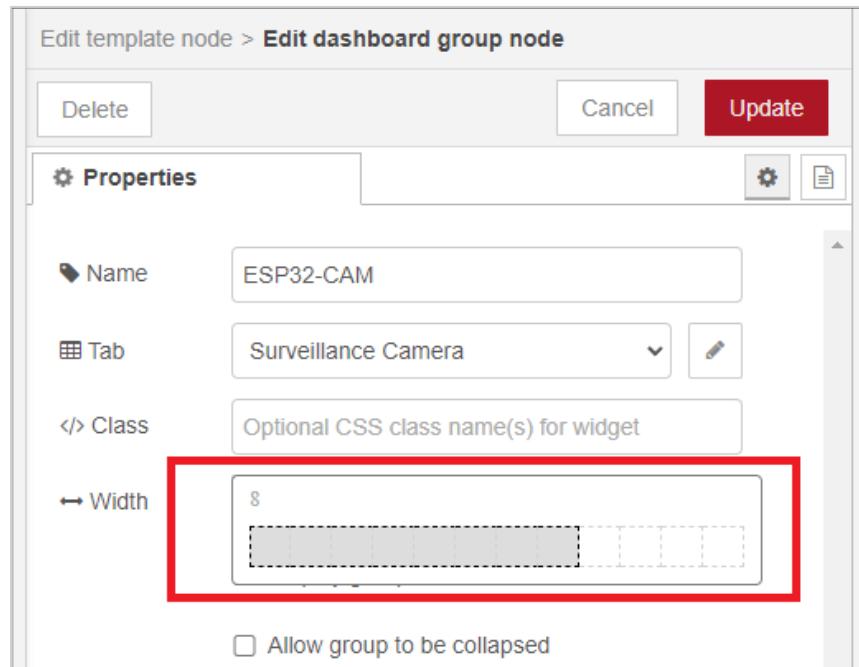
The video streaming image looks a bit small on a computer web browser. But, it's just the perfect size if you want to access it on your smartphone.



If you'll use Node-RED dashboard mainly on your computer, you can adjust the widget size. For example, double-click on the **template** node. Then, click on the **Group** pencil icon to edit the group properties.



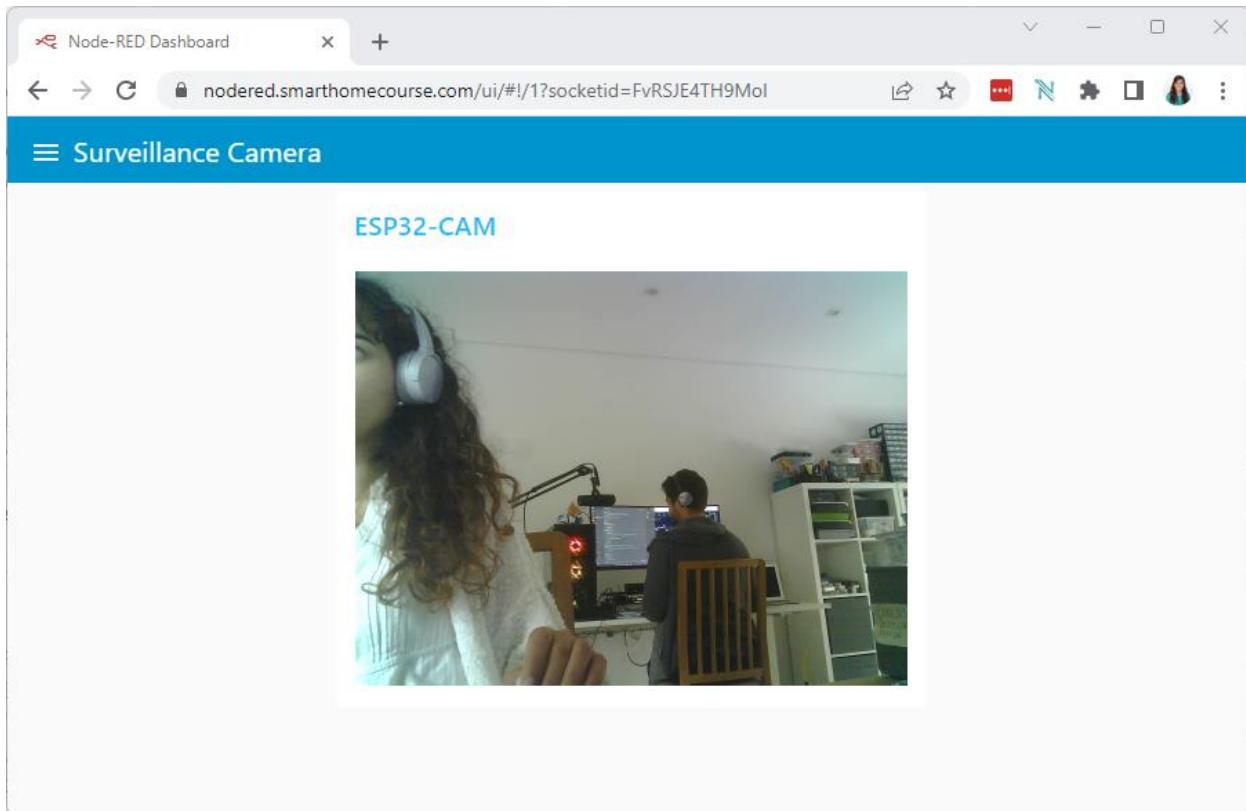
Edit the widget width to, for example, 8.



Click on **Update**, and then **Done**. Finally, deploy your app.



This is how the video streaming looks like now on Node-RED UI.



Important: the ESP32-CAM can only serve video streaming to one client at a time, so it is very important that you close the web browser window before proceeding. Otherwise, you won't see any video streaming on Node-RED later on.

Wrapping Up

In this module, you learned how to add an ESP32-CAM surveillance camera to your Node-RED home automation system. You can repeat the same procedure to add more cameras.

APPENDIX

Linux Commands and More...



This section provides additional information that might be useful like executing commands on the Raspberry Pi/server using Node-RED and a quick guide with Linux commands.

Sending Linux Commands Through the Node-RED UI

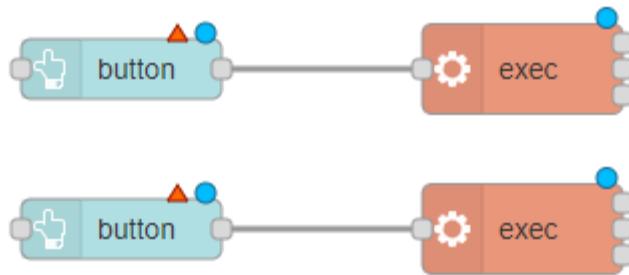
Sometimes you don't want to open the terminal and establish an SSH communication to power off your Raspberry Pi.

So, it would be useful to send the `poweroff` and the `reboot` commands through the Node-RED UI with a button press. That's exactly what you are going to learn in this extra Unit.

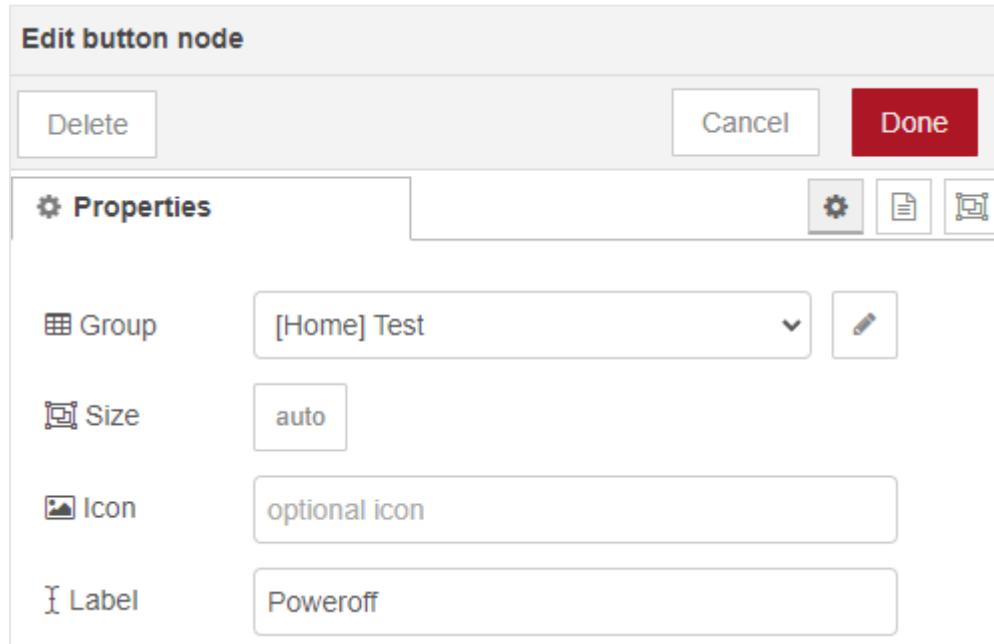
Creating the Flow

Follow the next steps to create a flow:

- 1) Drag two **button** nodes and two **exec** nodes to the flow and wire them as follows.

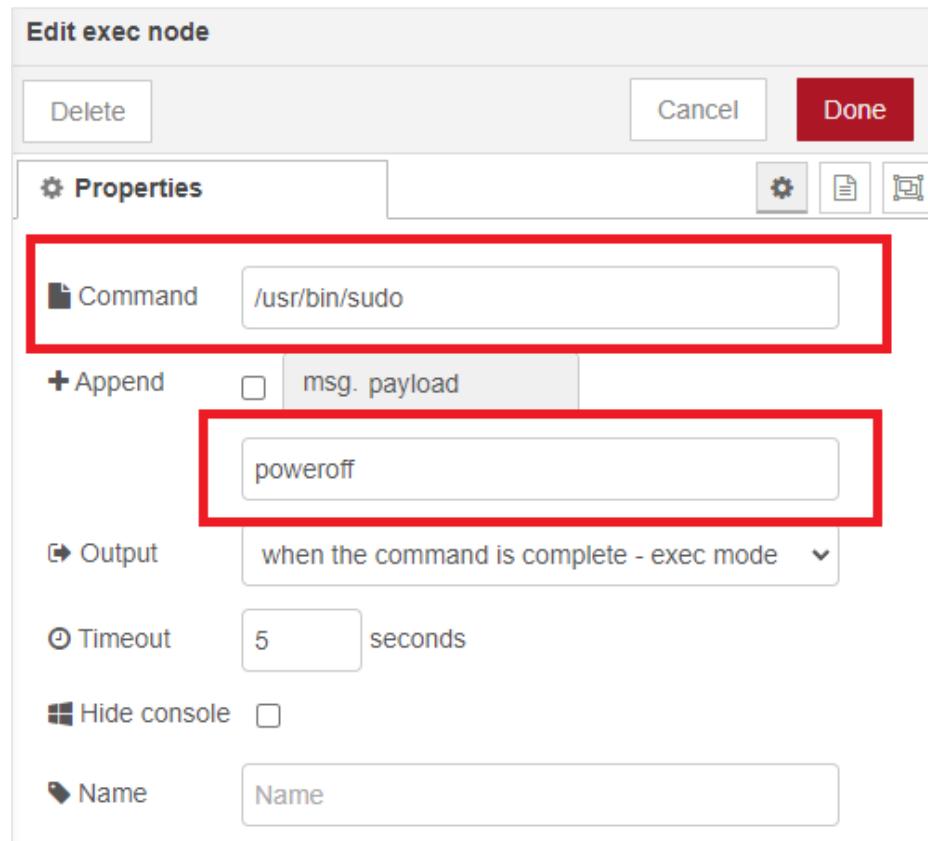


- 2) Double-click on the first **button** node to edit its properties. Choose a place on your Dashboard for the button and give it a name `Poweroff`. You can edit other button properties as you wish like the color, background, etc.

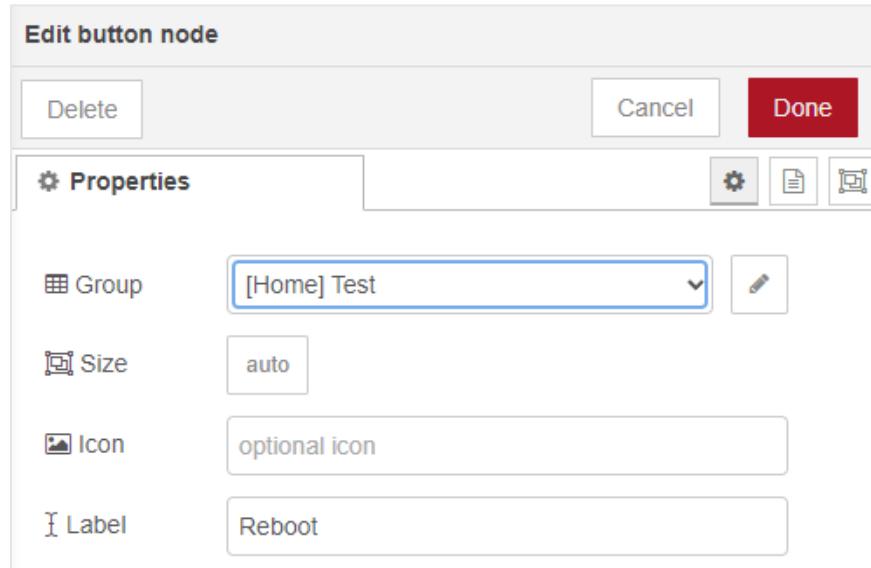


3) Edit the corresponding **exec** node as follows.

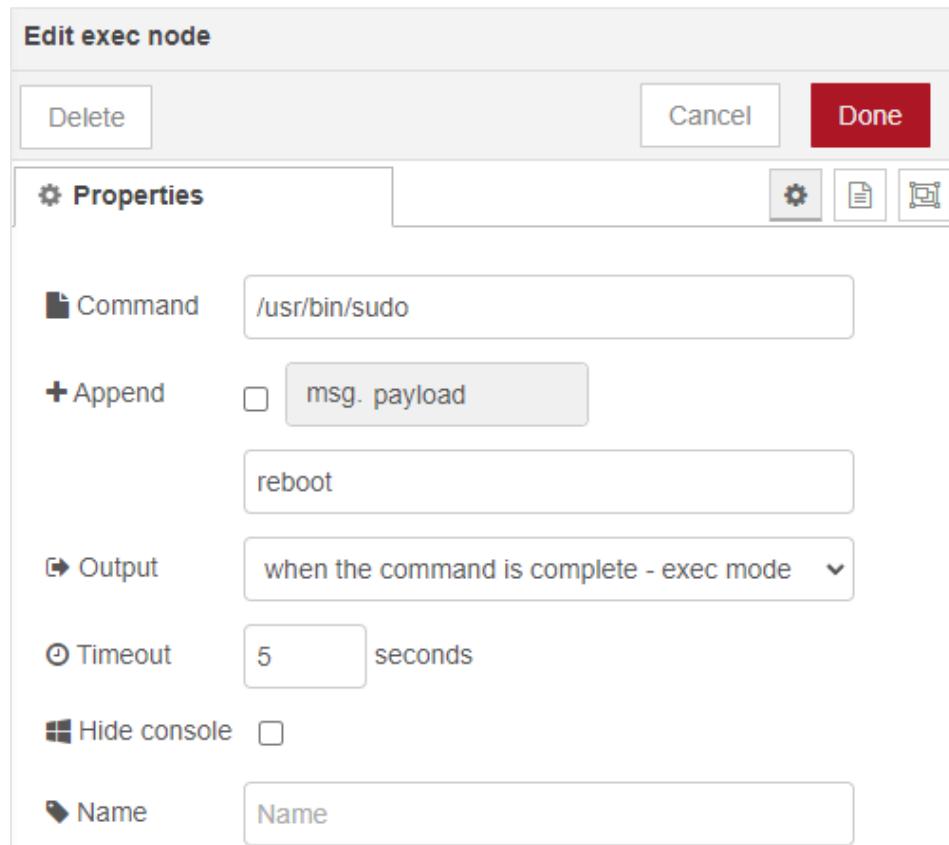
- Command: /usr/bin/sudo
- Append: poweroff



4) Edit the other **button** node to give it a name: Reboot.

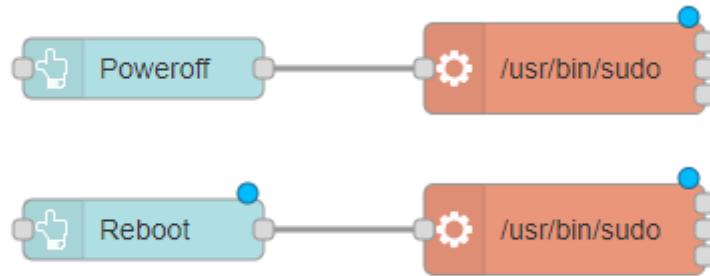


5) Then, edit the corresponding **exec** node to run the **reboot** command.



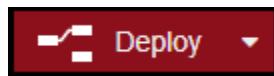
- Command: /usr/bin/sudo
- Append: reboot

The final flow looks like this.

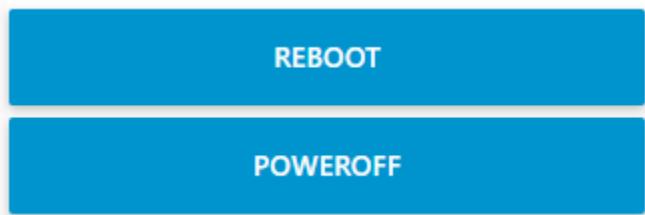


Demonstration

Deploy your application.



Go to Node-RED UI. The buttons to reboot and shut down the Raspberry Pi should be visible on the UI.



If you press the first button, the Raspberry Pi reboots. If you click the Poweroff button, the Raspberry Pi powers off safely.

You can apply this method to send almost any other Linux command to your Raspberry Pi through the UI.

Learning Basic Linux Commands

A big part of using a Raspberry Pi is also using the terminal. The terminal is something that a lot of people try to avoid because they feel like it is a bit hard to use.

But it doesn't need to be that way, because, in reality, we can break it down to just a few basic commands that you need to know to do everything. After you learn these commands you'll hopefully feel comfortable using the terminal.

In Module 1, we've shown how to establish an SSH communication with your RPi. Having that connection still on, you can now install software on your Pi remotely, create files or folders and run any scripts directly from your computer.

Follow the next few sections that will teach you how to use the command line in a few easy steps. You'll be comfortable in no time. You also don't need to know all these commands by heart, you can always access this appendix as a reference to remind you how to do something.

Exploring the Linux File System

It's time to play around with the command line.

For starters, type `pwd`, which means print working directory:

```
pwd
```

The output is `/home/pi`. Forward slashes are always used to indicate folders and files within other folders. In this case, the current working directory is `pi`, which is inside `home`, which is inside the root of the file system. Here, `pi` is the username with which you are logged in (it might be different if your user has a different name).

Note: The commands in Linux are case-sensitive, which means that `PWD`, `PwD`, `pwd`, and any other variations are completely different from `pwd`. The same holds for

all other commands and any code written in the programming languages addressed in this course.

Navigating the file system

The most frequent commands you will use are `ls` (list) and `cd` (change directory). They are used for listing the contents of a directory and moving from one directory to another.

When you first open the terminal, it will open up in your `home` folder (as you've seen with the `pwd` command). You can display exactly what kind of files or folders are in the working directory with `ls`:

```
ls
```

Want to create a new folder? Use `mkdir` followed by the name you want to give the folder:

```
mkdir NewFolder
```

You can use the `ls` command again to check if the folder was created.

To navigate, we'll be using the `cd` command, followed by the location you want to move to. This can be done like so:

```
cd NewFolder
```

This moved you to the `NewFolder` directory that you just created. So, you'll get the following before typing any command:

```
pi@raspberry:~/NewFolder $
```

Here's one trick you can use so you don't have to remember the exact name of the path – the command line or terminal will try to autocomplete the phrase if you press the Tab key while something is only partially typed. Try the `cd` command again (use `cd ..` to move up one directory):`cd`

```
pi@raspberry:~/NewFolder $ cd ..  
pi@raspberry:~ $ ls  
NewFolder
```

Now start writing your cd command again...

```
pi@raspberry:~ $ cd NewF
```

... by pressing **Tab** when you've only written 'NewF' It will autocomplete the file path:

```
pi@raspberry:~ $ cd NewFolder
```

Finally, there are some quick commands you can use to manipulate files. Create a new file with the **touch** command:

```
pi@raspberry:~/NewFolder $ touch NewFile.txt  
pi@raspberry:~/NewFolder $ ls  
NewFile.txt
```

Individual files can be copied using the command **cp**, followed by the file name and you can also use this to rename files by doing:

```
pi@raspberry:~/NewFolder $ cp NewFile.txt OtherFile.txt  
pi@raspberry:~/NewFolder $ ls  
NewFile.txt OtherFile.txt
```

The original file can then be deleted by using the **rm** command followed by the file name:

```
pi@raspberry:~/NewFolder $ rm NewFile.txt  
pi@raspberry:~/NewFolder $ ls  
OtherFile.txt
```

You can move files using the **mv** command:

```
pi@raspberry:~/NewFolder $ mv OtherFile.txt /home/pi  
pi@raspberry:~/NewFolder $ cd ..  
pi@raspberry:~ $ ls  
NewFolder OtherFile.txt
```

There's a lot more you can do with the command line, but these are the very basics. As you use Linux more and more, you'll be confronted with tasks that need the command line, and through this process, you'll learn just how much can be accomplished when you work using the command line to manipulate files.

Editing Files using the Terminal

Nano is an easy-to-use text editor that is installed by default in Raspberry Pi OS distribution and many other Linux distributions.

Using Nano

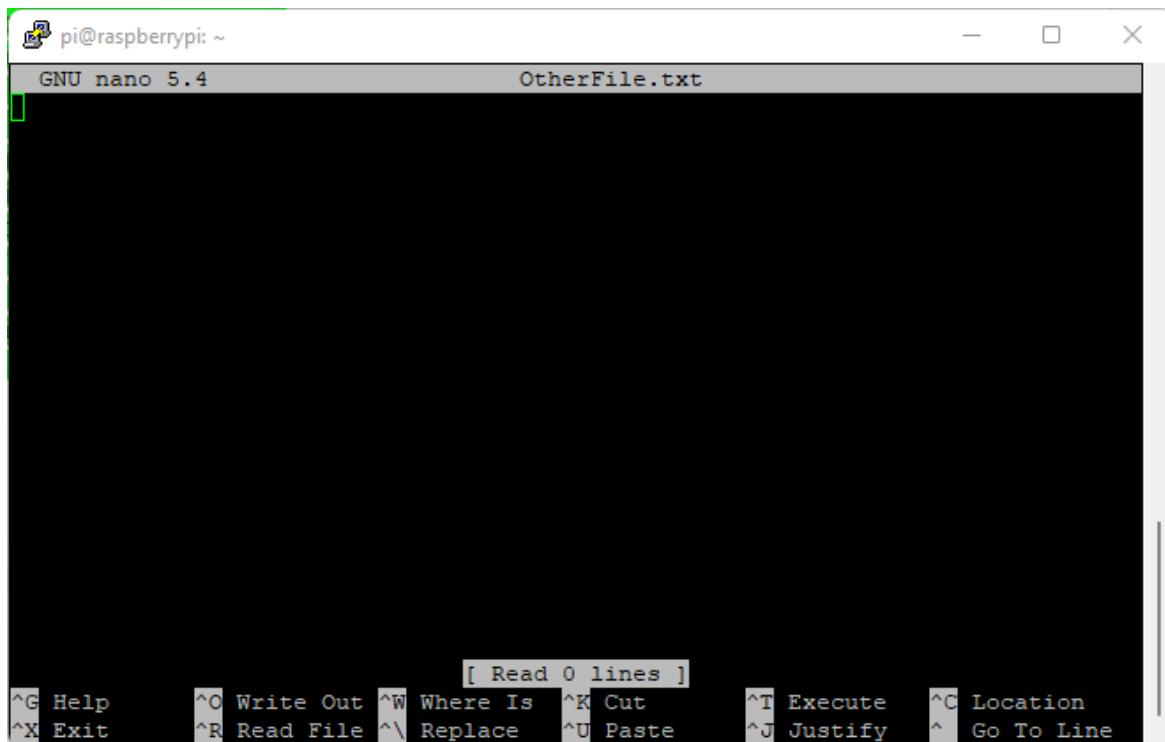
You can run nano by just typing in `nano` at the command prompt.

You can use the following commands to edit the `OtherFile.txt` created previously:

```
pi@raspberrypi:~ $ cd  
pi@raspberrypi:~ $ nano OtherFile.txt
```

Nano will follow the path and open that file if it exists. If it does not exist, it'll start a new buffer with that file name in that directory.

Let's take a look at the default nano screen:

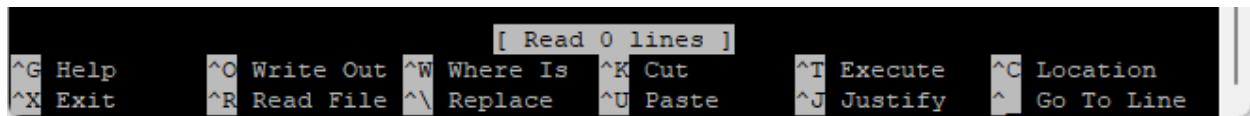


At the top row, you'll see the name of the program version number, the name and extension of the file you're editing, and whether the file has been modified since it was last saved.

Note: If you have a new file that isn't saved yet, you'll see "New Buffer."

Next, you'll see the contents of your file.

Lastly, the final two rows at the bottom are the shortcut lines (as shown below).



Shortcuts

Program functions are referred to as "shortcuts" in nano, such as saving, quitting, searching, etc. The most common ones are listed at the bottom of the screen (as shown in the preceding Figure), but there are many more that aren't.

Warning: nano does not use the Shift key in shortcuts. All shortcuts use lowercase letters and unmodified number keys, so Ctrl+G is NOT Ctrl+Shift+G.

Press **Ctrl+G** to bring up the Help menu and scroll down with your arrow keys to see a list of valid shortcuts.

When you're done looking at the list, hit **Ctrl+X** to exit Help menu.

Now let's say you're working on your text file and you want to save it and exit nano. This is executed by pressing **Ctrl+X**.

The screenshot shows a terminal window titled "pi@raspberrypi: ~". Inside, a nano editor window is open with the title "GNU nano 5.4" and the file name "OtherFile.txt *". The text "Just add some text here..." is displayed in the editor area. At the bottom, a menu bar lists various keyboard shortcuts: ^G Help, ^O Write Out, ^W Where Is, ^K Cut, ^T Execute, ^X Exit, ^R Read File, ^\ Replace, and ^U Paste.

Nano will ask you if you want to save the changes, you can type:

- **Y**, then **Enter** – to save all your changes
- **N**, then **Enter**- to cancel any changes

This is a very brief tutorial that shows how to edit a file and save it using the *nano* program.

The *nano* is way more powerful and has a lot of shortcuts that you can use to your advantage, but those go beyond what you need to know to complete this course. You can always refer to the official documentation or use the built-in *Help* menu.

Managing Software on Your Raspberry Pi

When you know your way around the command line, downloading and installing new software on a computer or device running the Linux OS is quite easy.

The software comes in what are called packages — software programs that can be downloaded from the Internet and installed simply by typing a command in the prompt.

To download and install these packages, you normally use a package manager, which downloads and installs not only the software you requested but also all other required software, known as dependencies.

The Raspbian distribution uses a package manager called `apt`.

To manage your software, you need the authorization of the administrator, also known as the superuser. To do so, type `sudo` (superuser do) before a command.

Updating and Upgrading

First and foremost, you have to update the list of available package versions that your package manager is aware of. (The package manager keeps such a list in Raspberry's file system.) Type the following command:

```
pi@raspberry:~ $ sudo apt-get update
```

You need to be connected to the Internet for this command to work. Text scrolls by after you type the command, giving information about the newest listings.

Next, you should update the software, which you can achieve by commanding `apt` to upgrade. This command upgrades all the packages you've installed to their most recent versions:

```
pi@raspberry:~ $ sudo apt-get upgrade
```

In terms of wording, the difference between updating and upgrading is subtle, but what they do is quite different (even though they're usually done together).

`sudo apt-get update` updates the list of available package versions but doesn't install or upgrade any of them, whereas `sudo apt-get upgrade` updates the packages themselves, checking the list to do so. For that reason, you should always run `update` before `upgrade`.

Installing software

To install a package for which you already know the name, you have to type the following command:

```
pi@raspberry:~ $ sudo apt-get install <desired application name>
```

Running software

To run programs directly from the prompt, simply type their names, as shown in the following command:

```
pi@raspberry:~ $ python
```

This opens the python interpreter.

Removing software

To remove software from your RPi, you resort once more to the `apt` package manager. Here's an example:

```
pi@raspberry:~ $ sudo apt-get remove <desired application name>
```

This command, however, leaves behind files that are somehow related to the software, such as configuration files and logs. If you don't intend to use those files in any way, you can remove everything by using `purge`:

```
pi@raspberry:~ $ sudo apt-get purge <desired application name>
```

Do not remove any package that you didn't install yourself unless you're certain that you know what it's for. It may be a necessary package that comes with the Linux OS, and removing it may lead to a system crash.

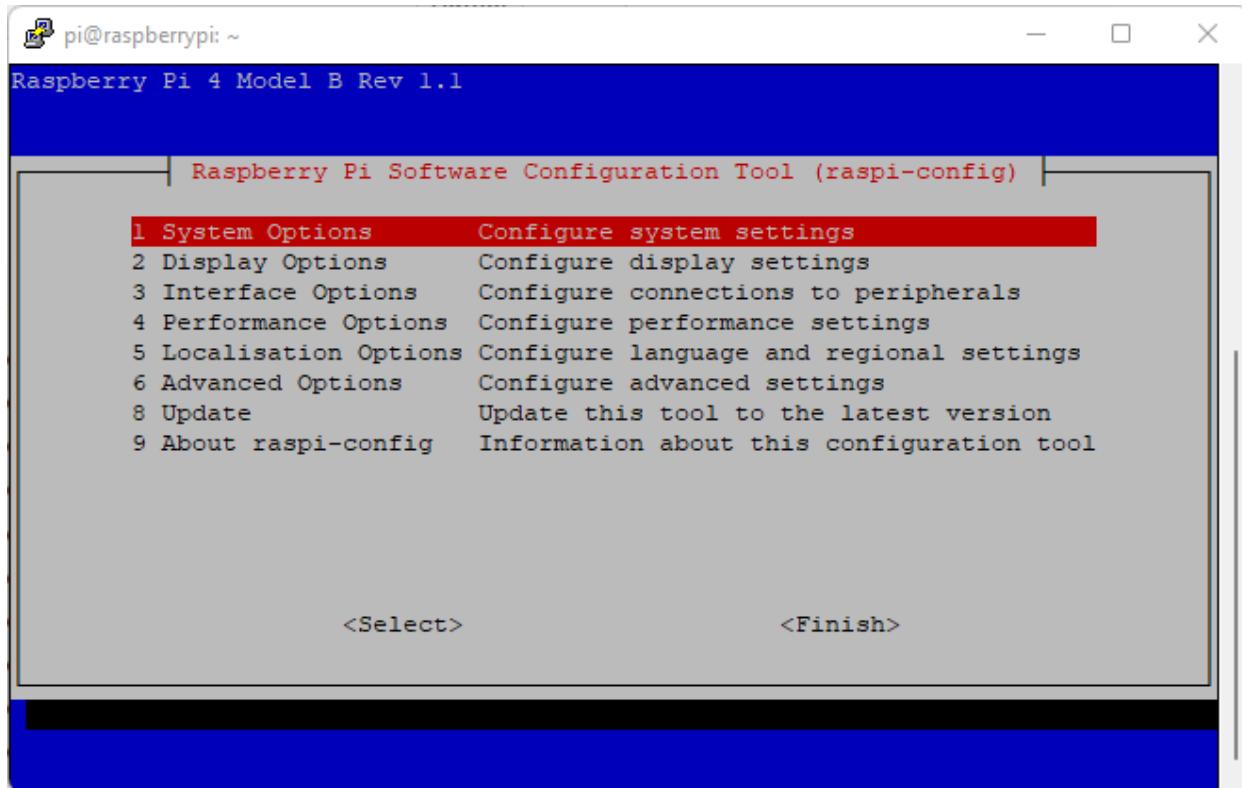
Changing the Raspberry Pi Settings

To change the Raspberry Pi configurations you can use a tool written by Alex Bradbury. To open the configuration tool, simply run the following from the command line:

```
pi@raspberry:~ $ sudo raspi-config
```

The `sudo` is required, because you will be changing files that you do not own as the `pi` user.

You should see a blue screen with options in a grey box in the center:



raspi-config aims to provide the functionality to make the most common configuration changes. keep in mind that some options require a reboot to take

effect. If you changed any of those, raspi-config will ask if you wish to reboot now when you select the <**Finish**> button.

It has the following options available:

1. System Options
2. Display Options
3. Interface Options
4. Performance Options
5. Localisation Options
6. Advanced Options
7. Update
8. About rasp-config

Shutting Down and Rebooting

There are better ways to shut down and reboot your Raspberry Pi than simply unplugging it. Unplugging your RPi is the equivalent of shutting down your computer by pressing the power button or even removing its power source, which can lead to file corruption.

To shut down your Raspberry Pi, simply type this command on the command line:

```
sudo poweroff
```

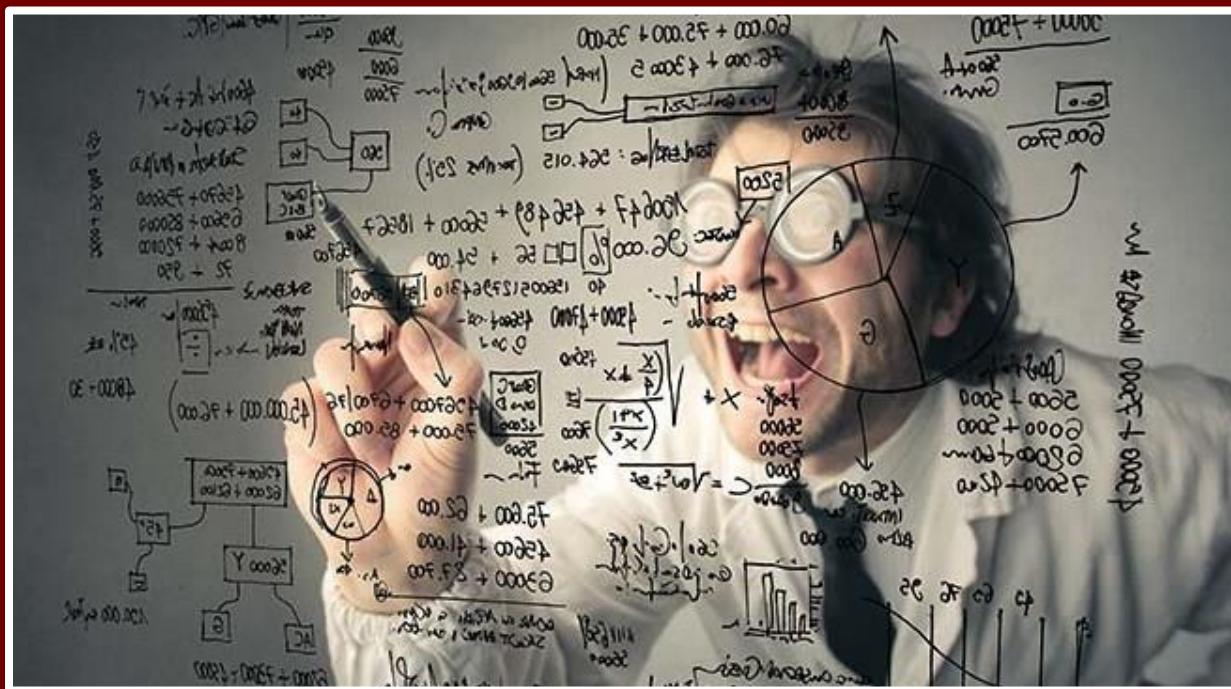
To reboot, type this:

```
sudo reboot
```

You need to log in again through SSH after rebooting.

WRAPPING UP

Congratulations for
completing this course!



If you've followed all modules presented throughout this eBook, you have all the foundations to build your own Home Automation system using Node-RED, MQTT, Raspberry Pi, ESP32, and ESP8266 boards. You also know how to monitor your system and save data on InfluxDB.

Congratulations for completing this course!

Here's a quick summary of what you learned with this training:

- Use the Raspberry Pi, the ESP32, and ESP8266 boards;
- Use Node-RED, Node-RED UI, and InfluxDB;
- Establish an MQTT connection with multiple devices in your network;
- Publish and subscribe with the ESP32/ESP8266 and Node-RED;
- Display your data in gauges and charts;
- Create a user interface using Node-RED dashboard to control the ESP outputs;
- Save data in InfluxDB;
- Send notifications using Node-RED (email, WhatsApp, and Telegram);
- Build a motion detection system with notifications;
- Trigger events based on sensor data;
- Schedule events and create time-based events;
- Put your home automation system on the cloud (access from anywhere).

We hope you had fun following along with this training and that you've learned something new. Now, we encourage you to add multiple ESP32 and/or ESP8266 boards around your home and build up on the snippets of code presented in this eBook to fit your projects and specific needs.

If you have something you would like to share (your projects, suggestions, feedback) use the [Private Forum](#), [Facebook group](#), or our [Support form](#).

Good luck with all your projects,

Sara Santos and Rui Santos

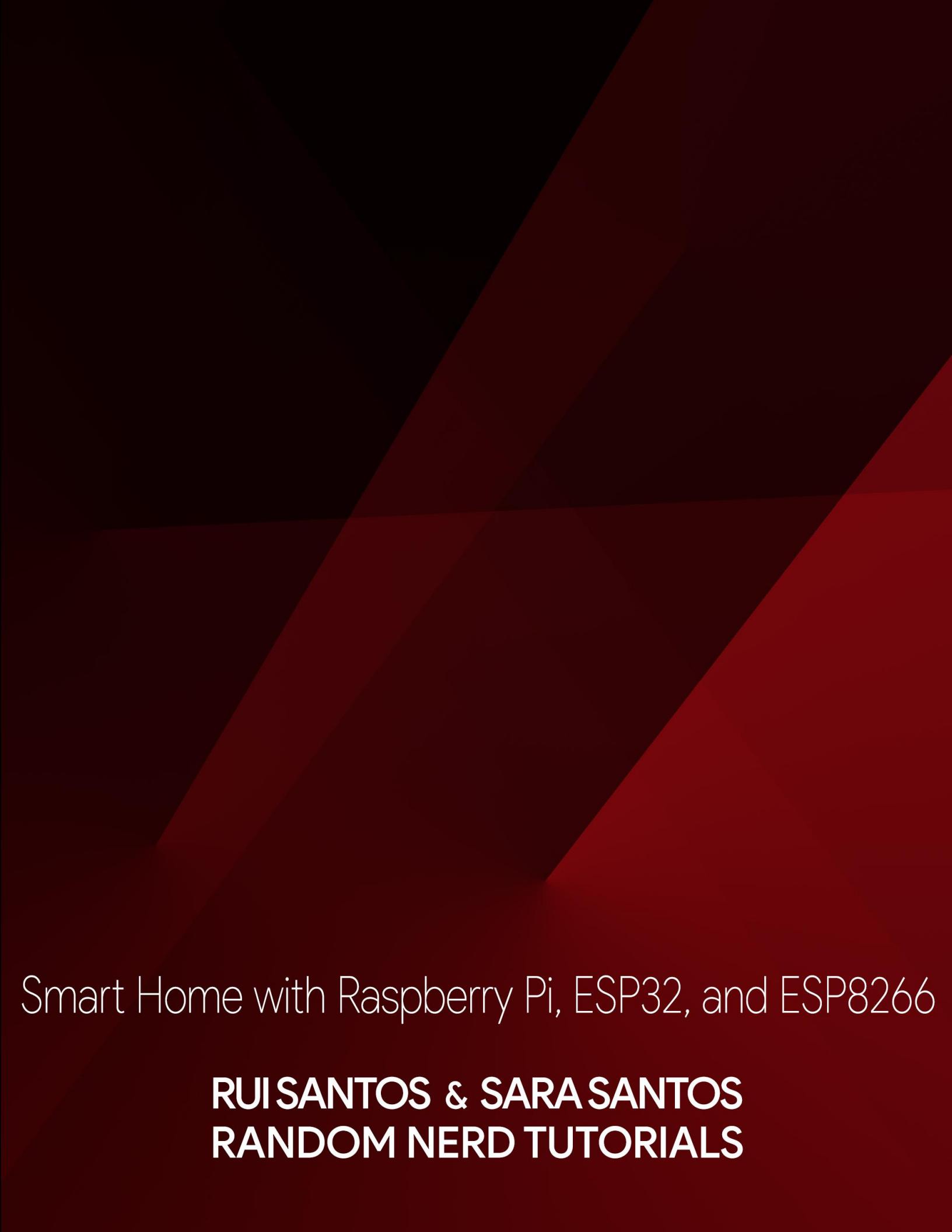
Other RNT Courses/eBooks

[Random Nerd Tutorials](#) is an online resource with electronics projects, tutorials, and reviews. Currently, Random Nerd Tutorials has more than [700 free blog posts](#) with complete tutorials using open-source hardware that anyone can read, remix and apply to their projects.

To keep free tutorials coming, there is also paid content or what we like to call "premium content". To support Random Nerd Tutorials, you can [download premium content here](#). If you enjoyed this eBook, make sure you [check all our courses and resources](#).

Here's a list of all the courses/eBooks available to download at Random Nerd Tutorials:

- [Learn ESP32 with Arduino IDE](#) (Bestseller)
- [Learn Raspberry Pi Pico/Pico W with MicroPython](#)
- [Build Web Servers using ESP32 and ESP8266](#)
- [Firebase Web App with the ESP32 and ESP8266](#)
- [Build ESP32-CAM Projects using Arduino IDE](#)
- [Home Automation Using ESP8266](#)
- [MicroPython Programming with ESP32/ESP8266](#)
- [20 Easy Raspberry Pi Projects](#) (available on Amazon)
- [Arduino Step-by-step Projects Course](#)
- [Android Apps for Arduino with MIT App Inventor 2](#)
- [Electronics For Beginners eBook](#)



Smart Home with Raspberry Pi, ESP32, and ESP8266

**RUI SANTOS & SARA SANTOS
RANDOM NERD TUTORIALS**