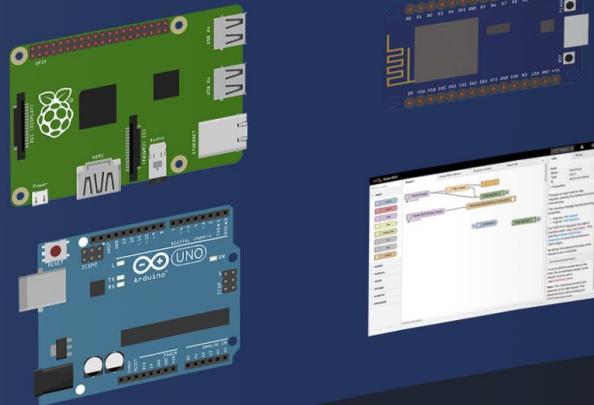


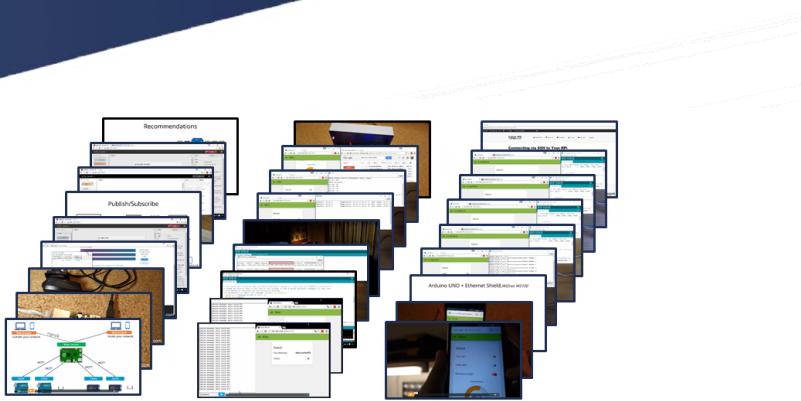
BUILD A HOME AUTOMATION SYSTEM FOR \$100

Learn Raspberry Pi, ESP8266, Arduino and Node-RED



This is a premium step-by-step course to get you building a real world home automation system using open-source hardware and software

Written by Rui Santos



Security Notice

This is the kind of thing I hate having to write about, but the evidence is clear: piracy for digital products is over all the internet.

For that reason I've taken certain steps to protect my 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 probably won't see anything different, since those strings are hidden in this PDF. I apologize for having to do that – but it means if someone were to share this eBook I know exactly who shared it and I can take further legal actions.

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

- <http://randomnerdtutorials.com/products>
- <https://rntlab.com>

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

What I really want to say is thank you for purchasing this eBook and I hope you have fun with it!

Disclaimer

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

The author (Rui 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 author (Rui Santos) earns a small commission from each purchase with that link. Please understand that the author has experience with all of these products, and he recommends them because they are useful, not because of the small commissions he makes if you decide to buy something. Please do not spend any money on these products unless you feel you need them.

Other Helpful Links:

- [Join Private Facebook Group](#)
- [Terms and Conditions](#)

About the Author

Hey There,

Thank you for purchasing my course "[Build a Home Automation System for \\$100](#)"!

I'm Rui Santos, founder of the [Random Nerd Tutorials blog](#), founder of [RNTLab.com](#) and author of [BeagleBone For Dummies](#).



If you're new to the world of Home Automation, this eBook is perfect for you! If you are already familiar with what Home Automation allows you to do, I'm sure you'll also learn something new.

This eBook contains the information you need to get up to speed quickly and start your own venture with the open-source hardware and software! Learn Raspberry Pi, ESP8266, Arduino and Node-RED.

Thanks for reading,

-Rui

P.S. If you would like the longer version of my story, you can find it over [here](#).

Join the Private Facebook Group

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

Inside that group you can ask questions and create discussions about everything related to ESP8266, Arduino, BeagleBone, Raspberry Pi, etc.

See it for yourself!

- Step #1: Go to -> <http://randomnerdtutorials.com/fb>
- Step #2: Click “Join Group”
- Step #3: I’ll approve your request within less than 24 hours

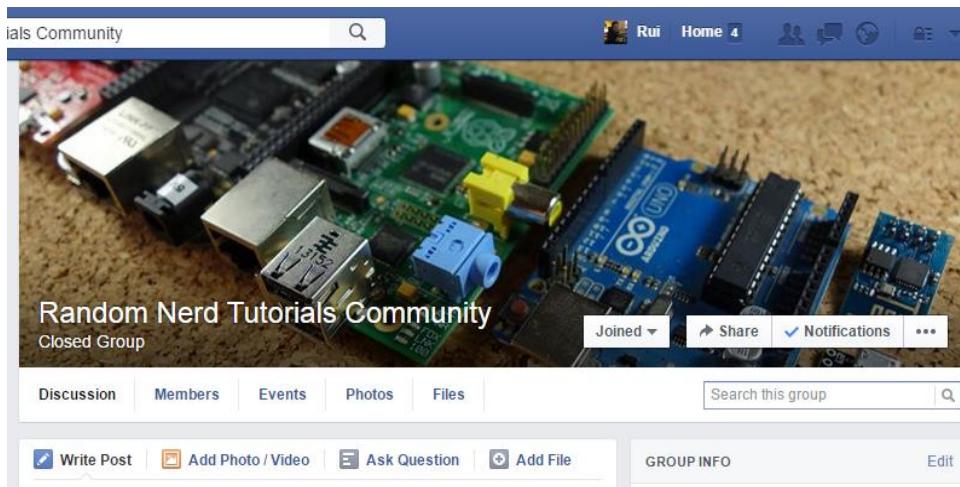


Table of Contents

I.	Security Notice.....	2
II.	Disclaimer	3
III.	About the Author.....	4
IV.	Join the Private Facebook Group	5
V.	Table of Contents	6
VI.	How to Watch the Videos.....	10
VII.	Getting Started with the Raspberry Pi.....	11
	Unit 1 - Course Overview	12
	Unit 2 - List of Components and Parts.....	15
	Unit 3 - Read This Before You Continue	19
VIII.	Installing the Operating System	20
	Unit 1 - Choosing and Downloading the Operating System	21
	Unit 2 - Installing Raspbian Lite in Your MicroSD Card	23
	Unit 3 - Booting Up Your Pi.....	31
	Unit 4 - Searching for Your Pi on Your Network.....	32
	Unit 5 - Connecting via SSH to Your RPi	35
IX.	Getting started with Node-RED.....	43
	Unit 1 - What's Node-RED?.....	44
	Unit 2 - Installing Node-RED	46
	Unit 3 - Node-RED overview	48
	Unit 4 - Controlling an LED with Node-RED	52
X.	Experimenting with MQTT	58
	Unit 1 - What is MQTT?.....	59
	Unit 2 - Installing Mosquitto Broker	64
	Unit 3 - Establishing an MQTT communication with Node-RED	66
XI.	Designing the Graphical User Interface	76
	Unit 1 - Installing Node-RED Dashboard	77

Unit 2 - Experimenting with Node-RED Dashboard.....	79
Unit 3 - Sketching Your Home Rooms.....	94
Unit 4 - Creating Tabs on Node-RED Dashboard for each Room	97
XII. Connecting the ESP8266 - Part 1	108
Unit 1 - Introducing the ESP8266.....	109
Unit 2 - How to Install the ESP8266 Board in Arduino IDE.....	113
Unit 3 - Testing the Installation.....	117
Unit 4 - Installing the PubSubClient Library.....	120
Unit 5 - Connecting the ESP8266 to the Node-RED Nodes.....	122
XIII. Connecting the ESP8266 - Part 2	137
Unit 1 - Controlling Outputs with ESP using MQTT.....	138
Unit 2 - Decoding RF Signals to Control Outlets.....	150
Unit 3 - Controlling Lamps and Outlets with ESP using MQTT	158
XIV. Connecting the ESP8266 - Part 3	169
Unit 1 - Reading the Temperature and Humidity.....	170
Unit 2 - Smoke and Gas Detector	187
Unit 3 - Motion Detector with Email Notification	207
Unit 4 - Storing Your Circuit in a Project Box Enclosure	228
Unit 5 - ESP8266 Final Demonstration.....	235
XV. Accessing Node-RED Dashboard From Anywhere in the World	238
Unit 1 - Accessing Node-RED Dashboard From Anywhere (it's encrypted and password protected).....	239
Unit 2 - Another Way of Making Node-RED Dashboard Accessible	246
XVI. Connecting the Arduino - Part 1	248
Unit 1 - Introducing the Arduino.....	249
Unit 2 - Installing the PubSubClient Library	254
Unit 3 - Connecting the Arduino to the Node-RED Nodes.....	256

Unit 4 - Controlling Outputs with Arduino using MQTT	269
XVII. Connecting the Arduino - Part 2	283
Unit 1 - Decoding RF Signals to Control Outlets.....	284
Unit 2 - Controlling Lamps and Outlets with Arduino using MQTT.....	293
Unit 3 - Plotting the Temperature in a Chart	304
Unit 4 - Reading the Light Intensity	316
Unit 5 - Triggering Outlets with Temperature and Luminosity	327
XVIII..... A	
dding Rules and Triggering Events	334
Unit 1 - Creating Master Switches or Modes	335
Unit 2 - Triggering Time-based Events.....	340
Unit 3 - Sending Notifications to All Your Mobile Devices	346
Unit 4 - Wrapping Up and Taking It Further.....	354
XIX. Extra #1 - Information that might be useful for this course	357
Unit 1 - How to Configure WiFi on Your Raspberry Pi.....	358
Unit 2 - Change the Time Zone on Raspberry Pi with Raspbian.....	362
Unit 3 - ESP-01 with Arduino IDE	364
Unit 4 - ESP-12E – Pinout Reference.....	371
Unit 5 - MQTT Authentication with Username and Password.....	373
Unit 6 - Exporting Node-RED Nodes	378
Unit 7 - Sending Linux Commands Through the Node- RED Dashboard	381
XX. Extra #2 - Getting Started with Linux	385
Unit 1 - Learning Basic Linux Commands	386
Unit 2 - Exploring the Linux File System	387
Unit 3 - Editing Files using the Terminal	391
Unit 4 - Managing Software on Your Raspberry Pi	395
Unit 5 - Changing the Raspberry Pi Settings	398

Unit 6 - Shutting Down and Rebooting.....	400
XXI. Download Other RNT Products	402

How to Watch the Videos

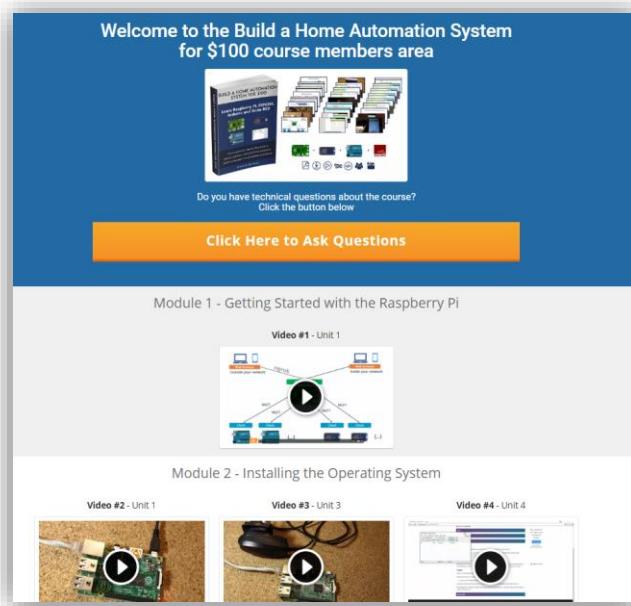
The Build a Home Automation System for \$100 course comes with video demonstrations and tutorials.

To watch the companion video tutorials, you have to go to this link:

<https://RNTLab.com/28hasvideos> and enter the password **87541457**

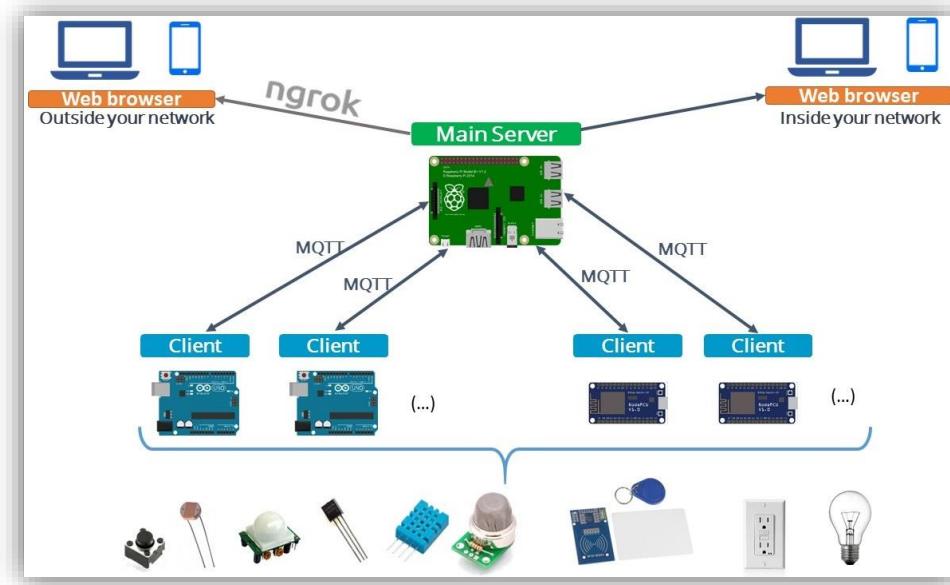
Each video is numbered, so you can easily find the video that you're looking for.

Click here to Watch the Videos
<https://RNTLab.com/28hasvideos>
Enter Password: 87541457

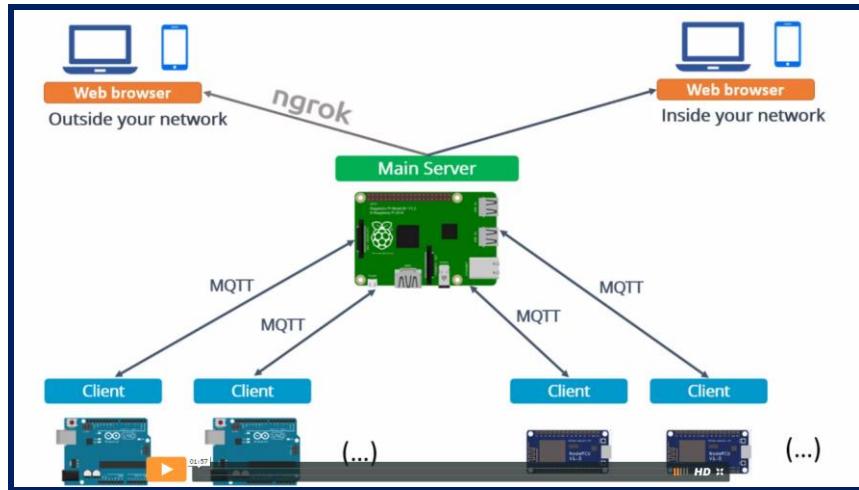


Module 1

Getting Started with the Raspberry Pi



Unit 1 - Course Overview



Video # 1 – <https://RNTLab.com/28hasvideos>

Hi there!

Welcome to the **Build a Home Automation System for \$100** course.

A hands on introductory course designed to teach you how to build a complete home automation system using open-source hardware and software.

This course is designed for people who find home automation or internet of things (IoT) subject interesting.

There's no previous knowledge required to complete the course. If I think that there is something extra that you need to learn during the course, I'll point you to the right resource.

The course contains video, image and text. So, it is straightforward to follow.

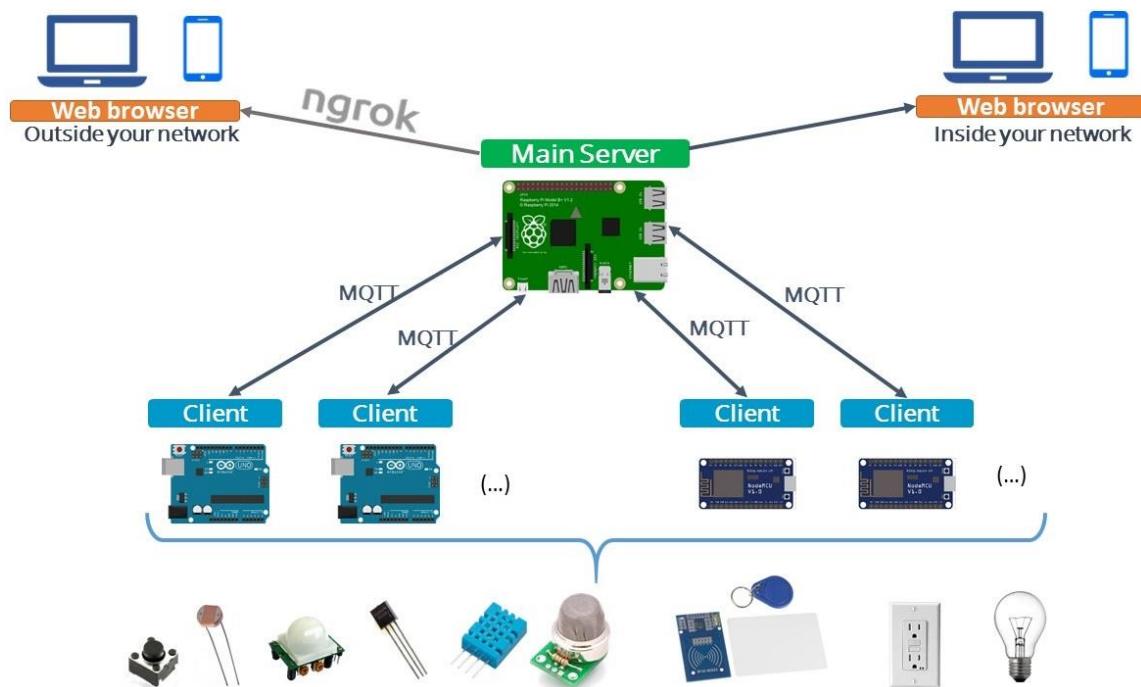
About Home Automation

Home automation can include the scheduling and automatic operation of lighting, heating, air conditioning, window blinds, security systems and more. Home automation may also allow vital home functions to be controlled remotely from anywhere in the world using any device with a browser that has an Internet connection.

Application Overview

By the end of the course, you'll have a home automation application running on your RPi that allows you to monitor and control various devices in your house.

Here's a quick overview of the application that we're going to build.



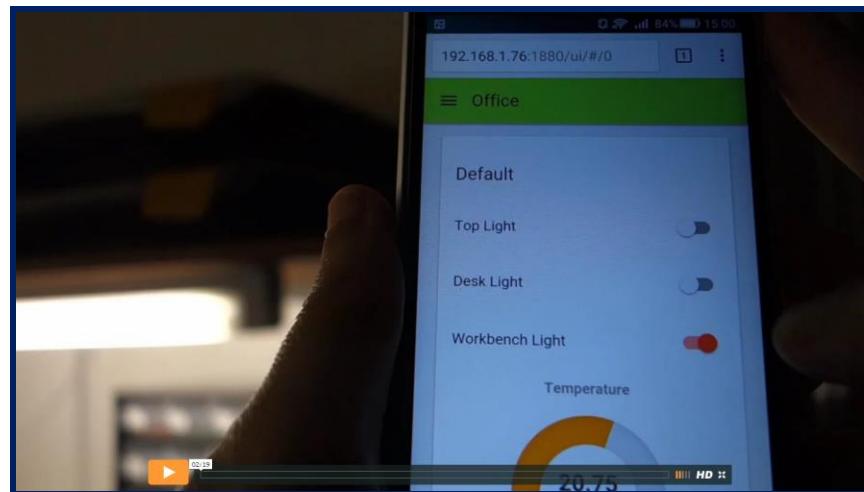
The brain of the operation is the Raspberry Pi. Any Raspberry Pi model B will work. You'll need to install some software in your Pi.

You'll learn how to connect the Arduino and the ESP8266 development boards to your Raspberry. They are going to talk via a protocol called MQTT.

You'll access your application from any web browser: whether it's using a laptop, a tablet or your smartphone. Your application will be accessible from anywhere.

Sneak Peek

Here's a sneak peek of how your project looks like by the end of Module 8:



Video # 19 - <https://RNTLab.com/28hasvideos>

Unit 2 - List of Components and Parts

The following list shows all the components and parts required to complete the Build a Home Automation System for \$100 course.

Don't worry, you don't need to buy all the components right away, because during the course I will mention the exact parts you need for each project.

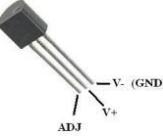
List of Components and Parts

Note: I recommend the Raspberry Pi 3 Model B, but you can use the Raspberry Pi 1/2 Model B or the Raspberry Pi Model B+.

Figure	Name	eBay
	Raspberry Pi 3 Model B - Recommended	http://ebay.to/1VdkRNx
	Raspberry Pi 2 Model B - Alternative	http://ebay.to/1NvgZ7b
	Raspberry Pi 1 Model B/B+ - Alternative	Don't buy this board, but you can use it if you already have one

	5V DC Power Supply	http://ebay.to/1M04pJ7
	MicroSD Card 8Gb Class 10	http://ebay.to/1Z1RduF
	Ethernet Cable	You probably already have this cable
	Breadboard	http://ebay.to/21bEojM
	Jumper Cables - Female to Male and Male to Male	http://ebay.to/1PXeaJz
	LEDs	http://ebay.to/20H2Oyy
	Resistors	http://ebay.to/1KsMYFP

	ESP8266-12E	http://ebay.to/1jU3piA
	433MHz Remote Controlled Sockets	Make sure you buy the right plug for your country and that operates at 433MHz (read the product description and look for the remote frequency specification) http://ebay.to/1qUAde6
	1x 433MHz Receiver and 2x 433MHz Transmitter	http://ebay.to/1TelzW8
	1x Buzzer	http://ebay.to/2bxCtp6
	Arduino UNO	http://ebay.to/1SQda0R
	DHT11 Temperature and Humidity Sensor	http://ebay.to/1bperHe
	PIR Motion Sensor	http://ebay.to/1Wqh9jL

	MQ-2 Gas Sensor	http://ebay.to/1SsN4ib
	Project Box Enclosure	http://ebay.to/1Tx2ctR
	Ethernet Shield (WIZnet W5100)	http://ebay.to/1WWpuhv
	LM335 Temperature Sensor	http://ebay.to/1TYCwUV
	Light-dependent Resistor (LDR)	http://ebay.to/22qj3Co

Unit 3 - Read This Before You Continue

I truly appreciate you taking the time to study this topic, and I hope that you will enjoy the Build a Home Automation System for \$100 course.

Problems during the course

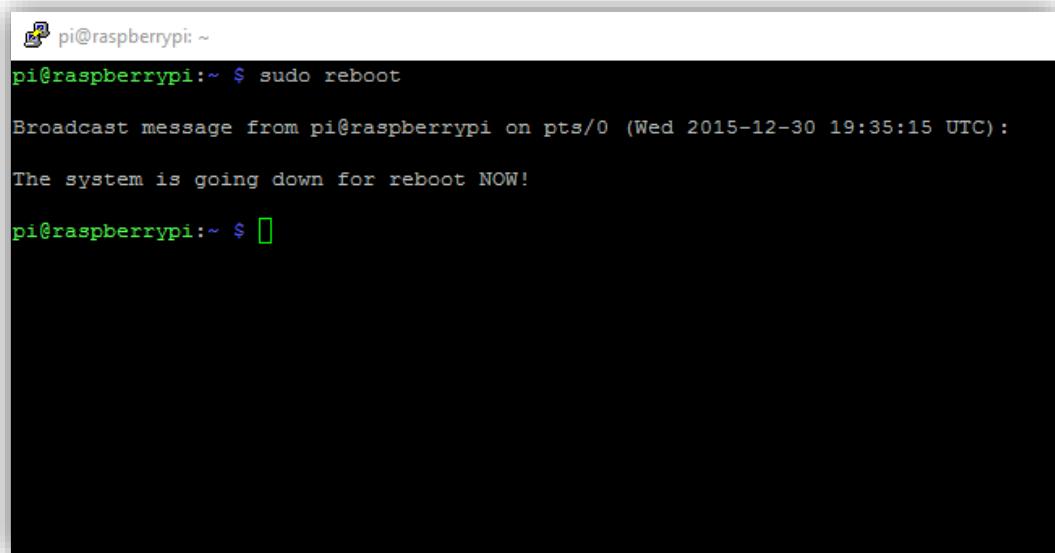
As you go through the course, it is likely that you will encounter some sort of technical problem.

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, you can always rely on the community to help you out. You can join the [Facebook Group](#) to get in touch.

Module 2

Installing the Operating System



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo reboot
Broadcast message from pi@raspberrypi on pts/0 (Wed 2015-12-30 19:35:15 UTC):
The system is going down for reboot NOW!
pi@raspberrypi:~ $ [
```

Unit 1 - Choosing and Downloading the Operating System



Video # 2 - <https://RNTLab.com/28hasvideos>

In this module, I'll show you how to download and prepare your Raspberry Pi with the latest version of the **Raspbian Lite** Operating System (OS).

The Raspberry Pi is a computer and like any other computer it needs an OS installed.

The Pi doesn't have built-in memory, so you'll need a microSD card to install your OS. I recommend using a microSD card class 10 with at least 8GB of memory.



If you go to the Raspberry Pi website and you open the downloads section: <https://www.raspberrypi.org/downloads>. You'll find all the official Operating Systems that you can download.

I recommend using **Raspbian**, because it is the most supported OS by the Raspberry Pi community. Throughout this course we're going to use **Raspbian Lite**.

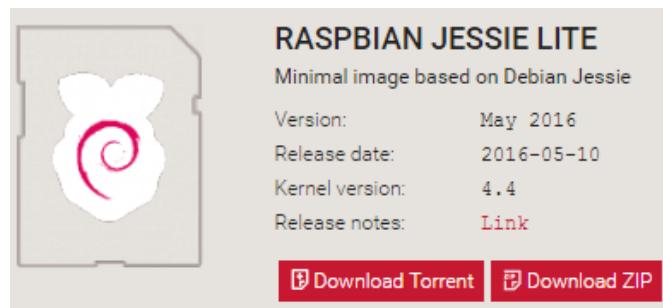


RASPBIAN

Why Raspbian Lite?

Because it is a lightweight version of the Raspbian and it doesn't have a graphical user interface installed. This means that it doesn't have any unnecessary software installed that we don't need for our projects, so this makes it the perfect solution for our home automation project.

Click the **Download ZIP** button to download the Raspbian Lite Operating System.



In the next Unit I'll show you how to prepare your microSD Card.

Unit 2 - Installing Raspbian Lite in Your MicroSD Card

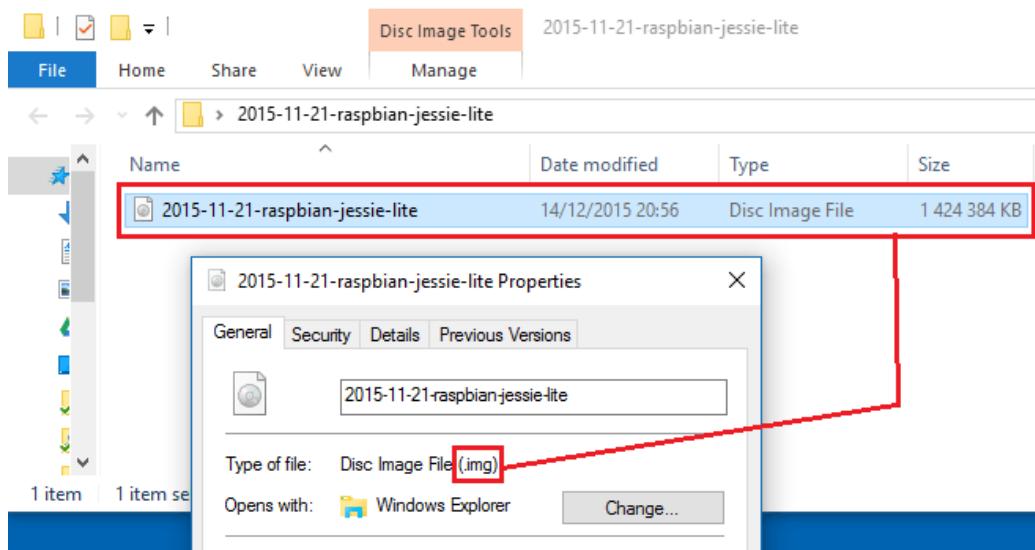
This Unit is divided into two sections: one for Windows and the other for Mac OS X/Linux.

Choose a title below to read the instructions for the Operating System that you have installed in your computer:

Read Windows Version

After downloading the Raspbian Lite OS, you should have a .zip file in your Downloads folder.

Unzip it and inside you'll find a .img file (as shown in the Figure below).



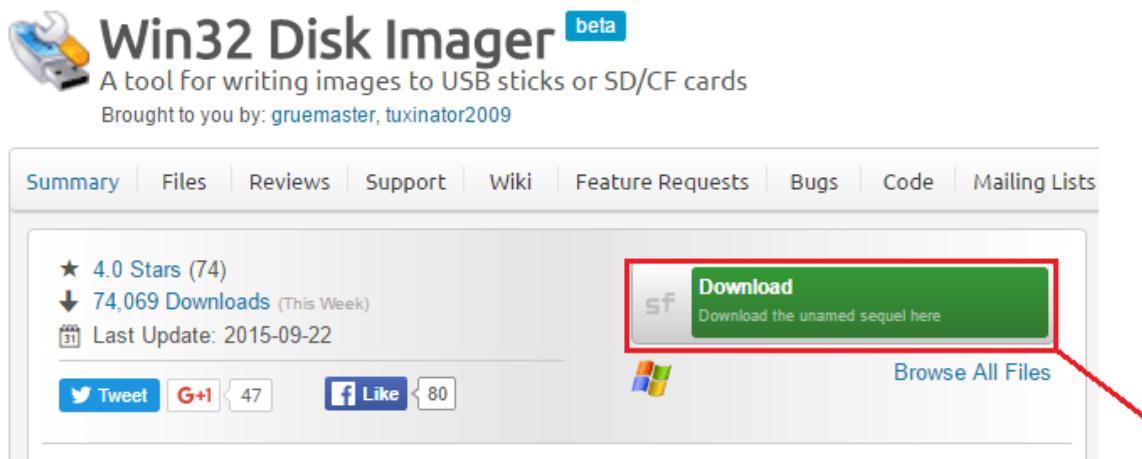
Downloading Win32 Disk Imager

To flash your microSD Card with a .img file on a Windows PC, it requires an application called **Win32 Disk Imager**, which is available for free download.

Follow these steps to install it:

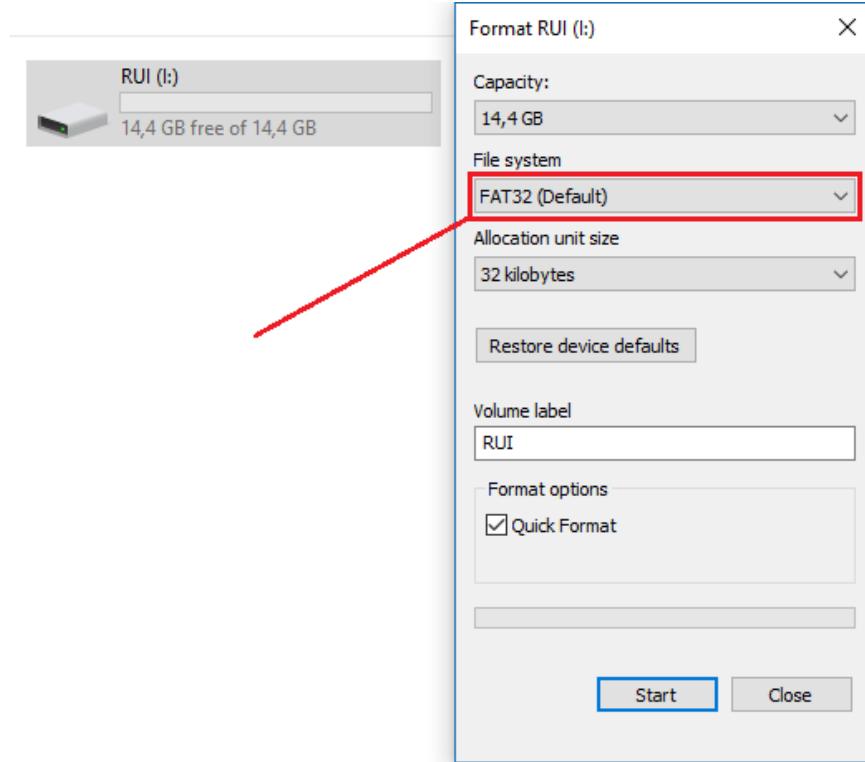
- 1) Go to the Win32 Disk Imager download page at

<http://sourceforge.net/projects/win32diskimager/>.



- 2) Click the Download button to retrieve the installer.
- 3) Run the Win32 Image Writer application installer. With Win32 Disk Imager installed, you're ready to write the .img file in your microSD card.

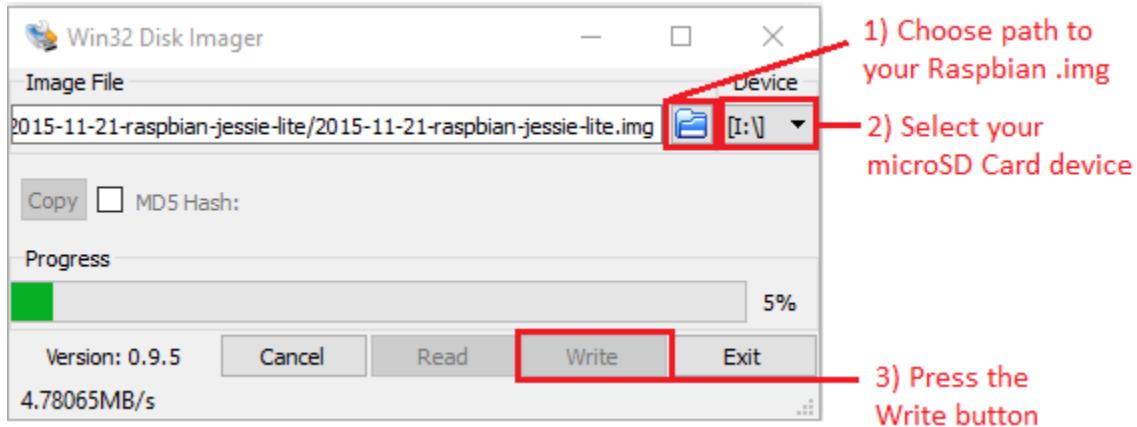
Connect your microSD card to your computer. Open its properties window and check if it is formatted in FAT32



Flashing your microSD card

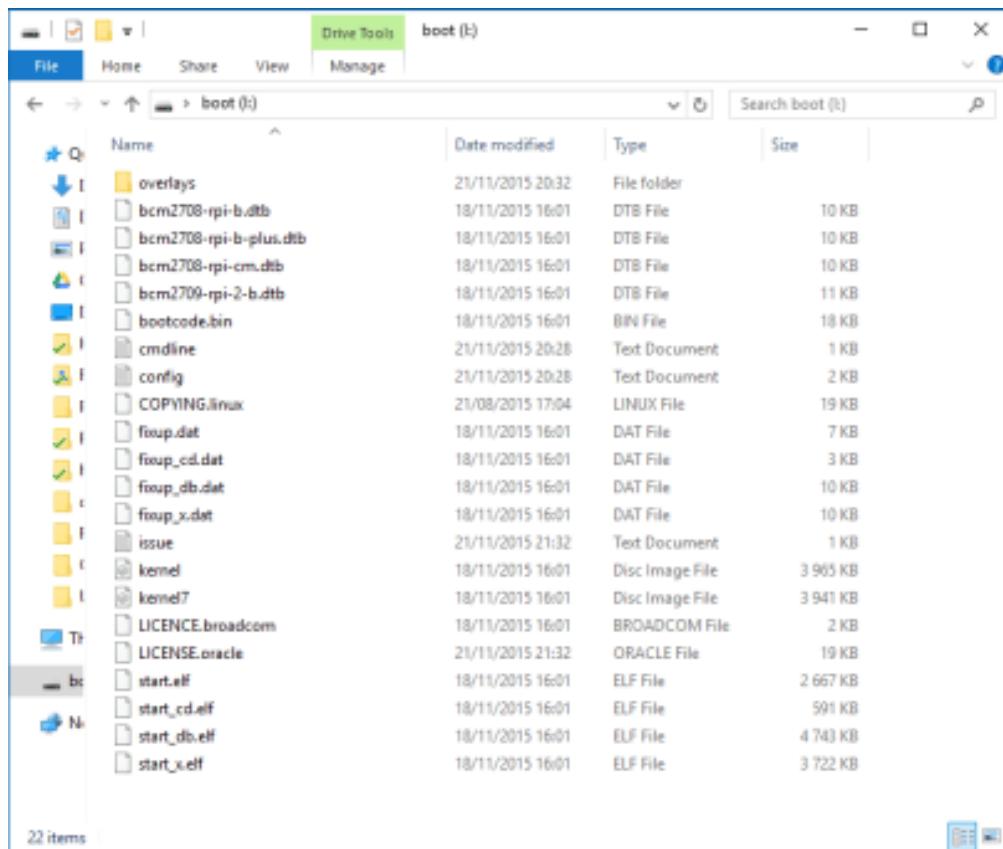
After opening Win32 Disk Imager, follow these steps:

- 1) Select your Raspbian Lite .img file.
- 2) Select your microSD card as the device. **This process erases and overwrites the selected device.** Be certain that you've selected the microSD card, and be certain that you have copies of any files that you need from the card. I can't stress this enough: **Be certain that the microSD card is the device you chose!**
- 3) Click Write to start writing the image to the microSD card. This process takes between 10 and 20 minutes, depending on your microSD card class number and your SD card writer's transfer speed.



Removing your microSD card

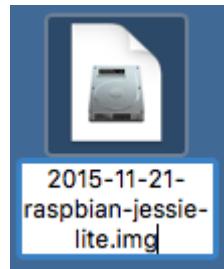
After the flashing process is finished, open your microSD card and you should see something similar to the image below:



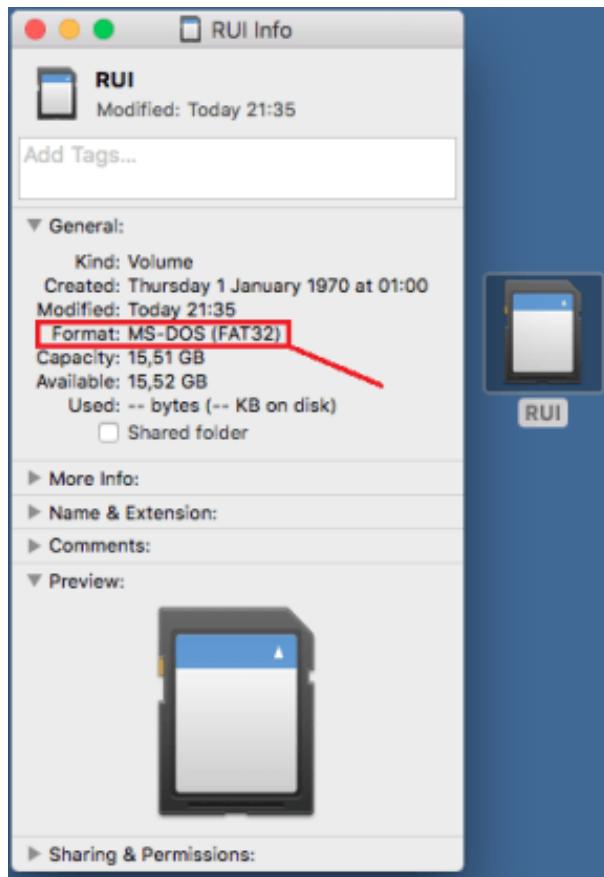
Raspbian Lite was successfully flashed into your microSD card!

Read Mac OS X/Linux Version

After downloading the Raspbian Lite OS, you should have a .zip file in your **Downloads** folder. Unzip it and inside you'll find a .img file (as shown in the figure below).



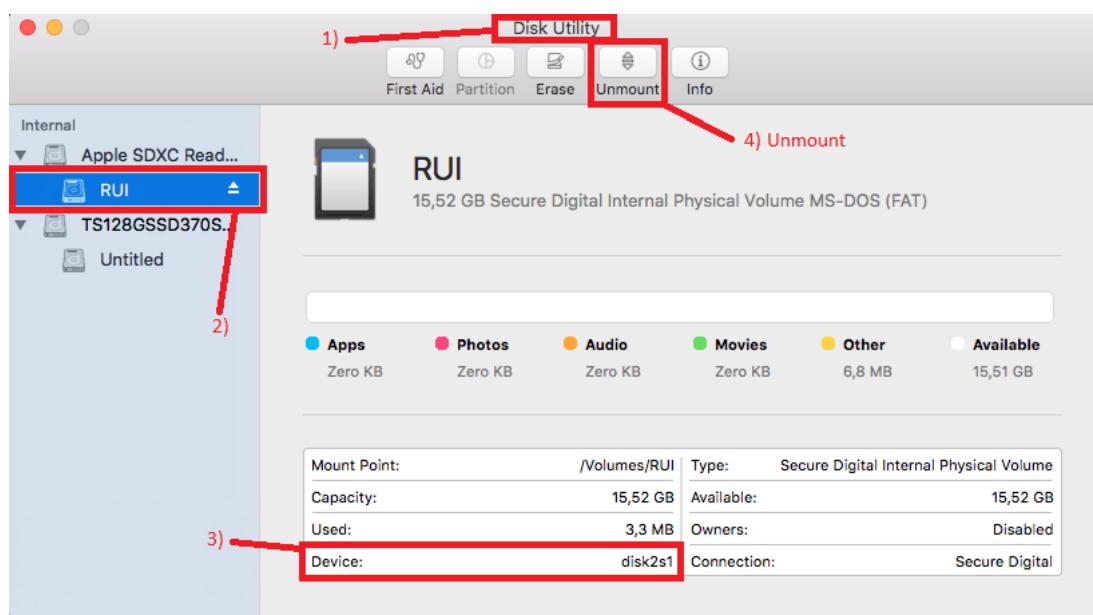
Connect your microSD card to your computer. Open its properties window and check if it is formatted in FAT32.



Opening the Disk Utility

Having your microSD card connected to your computer, follow these steps:

- 1) Open the Disk Utility.
- 2) Select the microSD card in which you want to install the Raspbian Lite OS.
- 3) Save the device name. In my case it is **disk2s1**, but it only matters the **disk2** part.
- 4) Then Unmount your microSD card.



Flashing your microSD card with the Terminal

Warning: This process erases and overwrites the selected device. Be certain that you've selected the microSD card, and be certain that you have copies of any files that you need from the card.

Open a Terminal window:

1) Start typing the following command:

```
$ sudo dd bs=1m if=path_to_your_raspbian_lite.img of=/dev/rdiskN
```

The highlighted red text in the preceding command will be unique to you. Make sure you replace the **N** with the right letter for your microSD card that you have found in the Disk Utility section, in my case is **2**.

Warning: I can't stress this enough: Be certain that the microSD card is the device you chose!

2) When you are 100% certain that you have entered the right command, press Enter/Return key to execute that command.

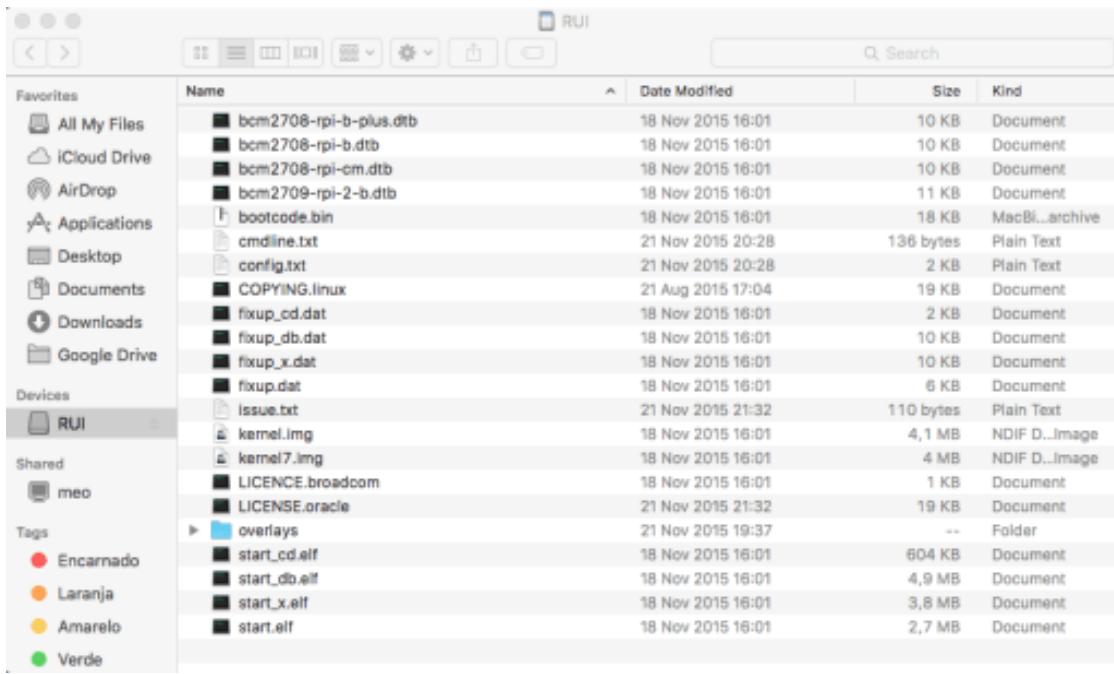
This process takes between 10 and 20 minutes, depending on your microSD card class number and your SD card writer's transfer speed.

Note: Your Terminal window will remain blank during the flashing process and you won't see anything changing, so be patient until the command finishes.



Removing your microSD card

After the flashing process is finished open your microSD and you should see something similar to the image below:



Raspbian Lite was successfully flashed into your microSD card! To remove your microSD card from your computer, type the following command:

```
$ sudo diskutil eject /dev/rdiskN
```

Replace the **N** with your microSD card disk number, in my case it's **2**. Press Enter/Return key. You can now remove your microSD card from your computer.

```
Rui -- bash -- 105:11
Utilizadores-MBP:~ Rui$ sudo dd bs=1m if=Desktop/2015-11-21-raspbian-jessie-lite.img of=/dev/rdisk2
1391+0 records in
1391+0 records out
1458569216 bytes transferred in 136.993359 secs (10647007 bytes/sec)
Utilizadores-MBP:~ Rui$ sudo diskutil eject /dev/rdisk2
Disk /dev/rdisk2 ejected
Utilizadores-MBP:~ Rui$
```

Unit 3 - Booting Up Your Pi



Video # 3 - <https://RNTLab.com/28hasvideos>

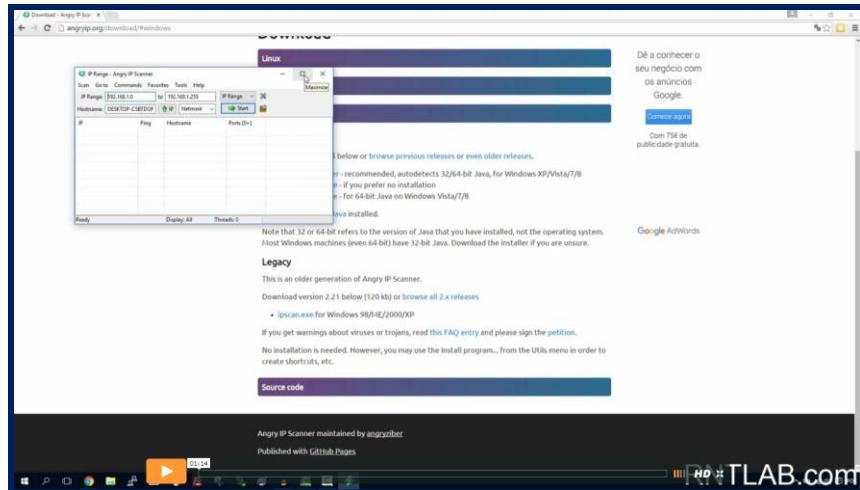
After installing Raspbian Lite in your microSD, it is time to boot up your RPi for the first time.

Follow these instructions:

- 1) Insert your microSD card in your Raspberry Pi
- 2) Connect an Ethernet cable from your Raspberry Pi to your router to ensure you have an internet connection
- 3) Connect your 5V DC power adapter to power your Pi

In the next couple of Units, I'll show you how to establish an SSH communication with your Raspberry Pi.

Unit 4 - Searching for Your Pi on Your Network



Video # 4 - <https://RNTLab.com/28hasvideos>

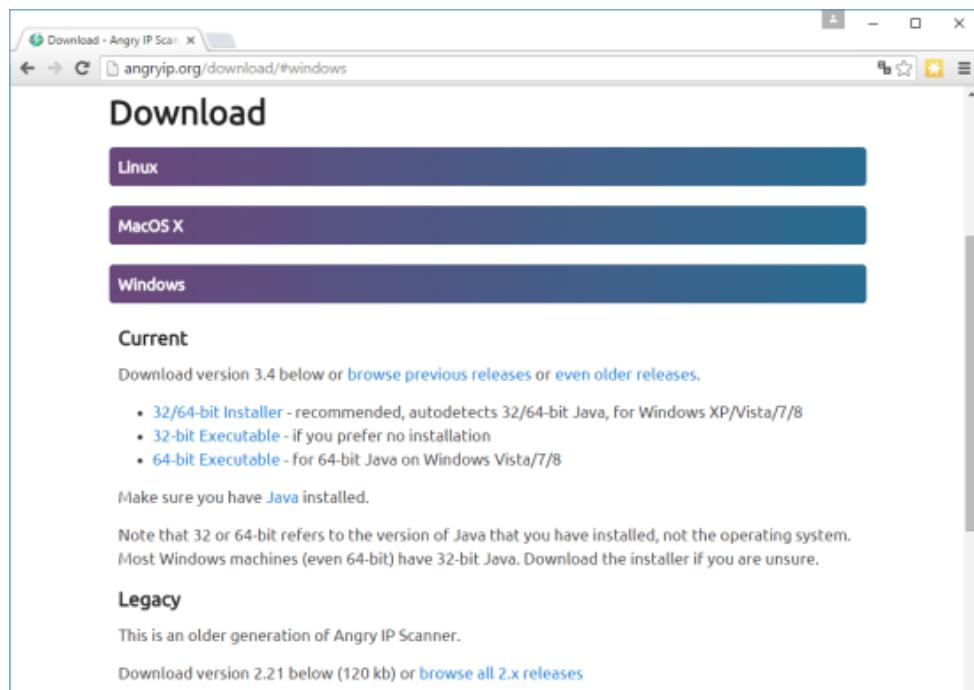
After booting up your Raspberry Pi, you need to find its IP Address. In order to find your RPi in your network, you need to install a software that scans your network for devices.

I'll use a software called **Angry IP Scanner**. It runs on Windows, Mac OS X or Linux.

Downloading Angry IP Scanner

Go to the downloads section of the Angry IP Scanner software: <http://angryip.org/download>.

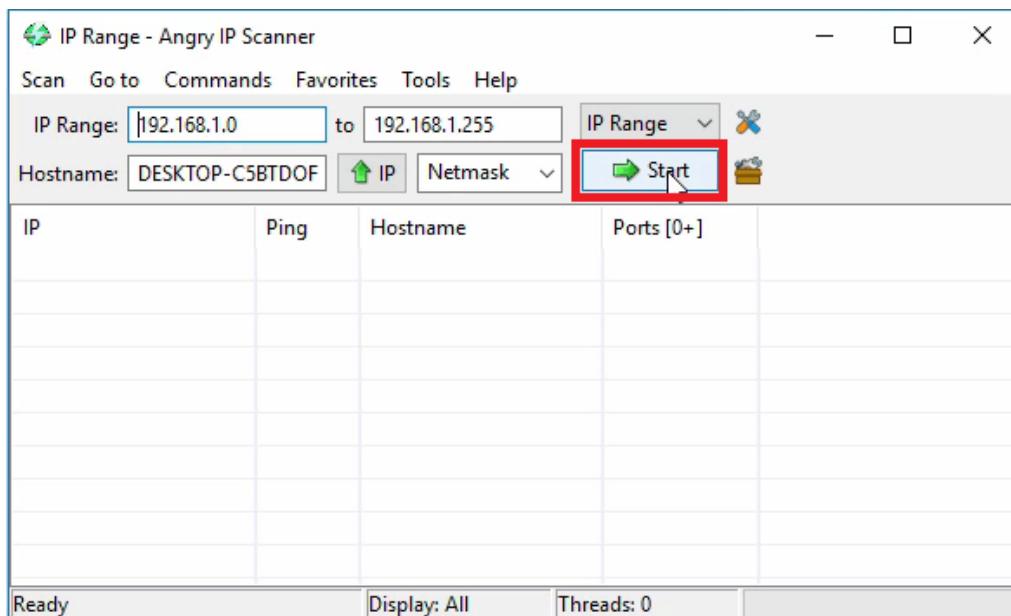
Select the right installation file for your computer. In my case, I'm using 64-bit Windows PC.



Note: Make sure you have [Java](#) installed.

Running Angry IP Scanner

Now, run the Angry IP Scanner. When you launch the software, it should automatically pick the IP Range for your network, so all you need to do is press **Start** and wait a few seconds.



Angry IP Scanner found my Pi on the local network with its default Hostname of "raspberrypi.local".

The screenshot shows the interface of the Angry IP Scanner software. At the top, there's a menu bar with Scan, Go to, Commands, Favorites, Tools, and Help. Below the menu is a toolbar with buttons for IP Range, Hostname, Start, and others. The main area is a table with columns: IP, Ping, Hostname, and Ports [0+]. The table lists several IP addresses and their corresponding hostnames and ping times. The row for the Raspberry Pi (IP 192.168.1.98, Hostname raspberrypi.local) has a red box around it, and a cursor is hovering over the IP column of this row.

IP	Ping	Hostname	Ports [0+]
192.168.1.66	0 ms	DESKTOP-C5BTDOF.lan	[n/s]
192.168.1.253	3 ms	MEO	[n/s]
192.168.1.98	268 ms	raspberrypi.local	[n/s]
192.168.1.64	500 ms	[n/a]	[n/s]
192.168.1.74	431 ms	[n/a]	[n/s]
192.168.1.254	1 ms	[n/a]	[n/s]
192.168.1.1	[n/a]	[n/s]	[n/s]
192.168.1.2	[n/a]	[n/s]	[n/s]
192.168.1.3	[n/a]	[n/s]	[n/s]
192.168.1.4	[n/a]	[n/s]	[n/s]

Save your IP address in a notepad (for example 192.168.1.98), because you'll need it later in this course.

Important: Your Raspberry Pi may have a different IP address depending on whether it's connected to your router through WiFi or Ethernet, and that address might even change from time to time. If you ever find yourself unable to connect via SSH, you can always double-check the IP address using the Angry IP software!

Unit 5 - Connecting via SSH to Your RPi

This Unit is divided into two sections: one for Windows and the other for Mac OS X/Linux.

Choose a title below to read the instructions for the Operating System that you have installed in your computer:

Read Windows Version

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. It has grown to be one of the most popular methods for communication between different devices.

Download PuTTY

If you use Windows, you need to download and install a free application called PuTTY. Here's how to install it:

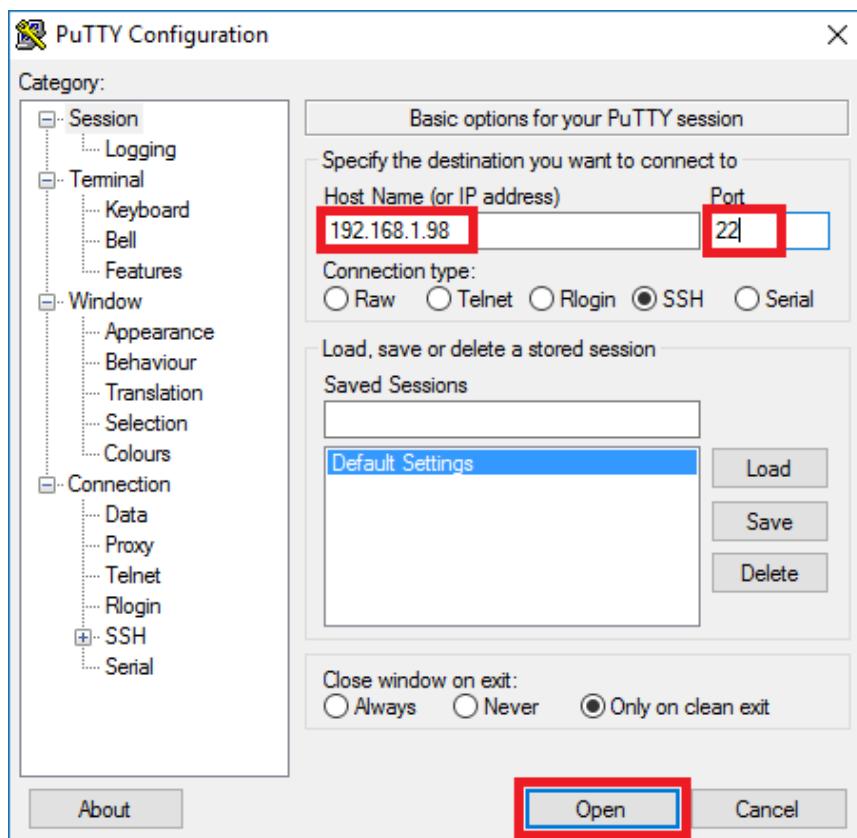
- 1) Open your web browser.
- 2) Go to www.putty.org.
- 3) Click the **putty.exe** file to download it.
- 4) Run the **putty.exe** file to install the software.

Opening PuTTY

With PuTTY installed, power up your Raspberry Pi and follow these steps:

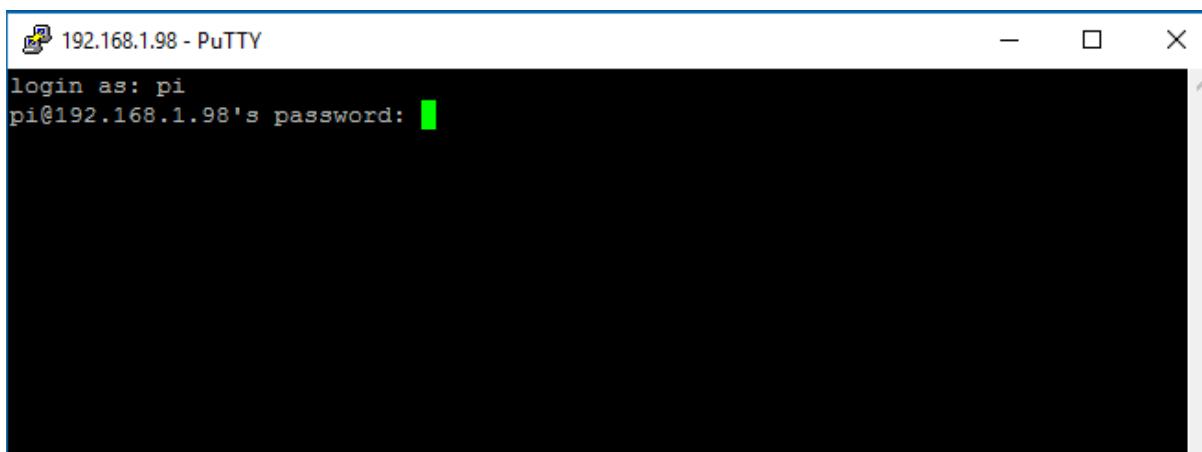
- 1) Open PuTTY.
- 2) In the PuTTY Configuration dialog box, select SSH as your connection type.
- 3) Type your IP address from the previous Unit, in my case is **192.168.1.98** as the host. The port needs to remain at the default number, which is **22**.

The dialog box should have the settings shown in the Figure below:



- 5) Click **Open**.
- 6) When you're asked to log in, type **pi** and press **Enter**.

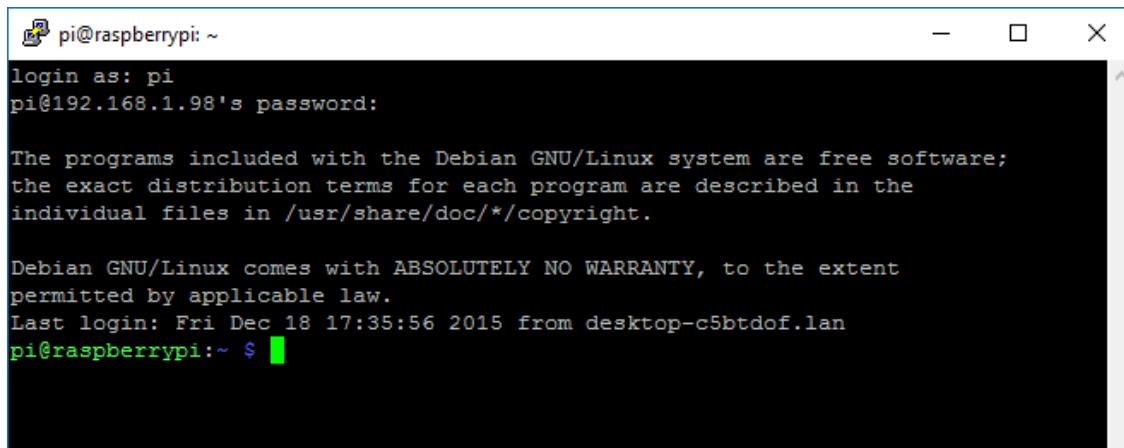
7) When you're asked to type a password, type **raspberry**.



A screenshot of a PuTTY terminal window titled "192.168.1.98 - PuTTY". The window shows a black terminal screen with white text. At the top, it says "login as: pi". Below that, it asks "pi@192.168.1.98's password:" followed by a redacted password field.

Note: Default settings for Raspbian Lite OS are: username = pi and password = raspberry

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.



A screenshot of a terminal window titled "pi@raspberrypi: ~". The window shows a black terminal screen with white text. It starts with "login as: pi" and "pi@192.168.1.98's password:". Below the password prompt, there is a block of text about Debian's free software license and warranty. At the bottom, it shows the last login information: "Last login: Fri Dec 18 17:35:56 2015 from desktop-c5btddof.lan" and the prompt "pi@raspberrypi:~ \$".

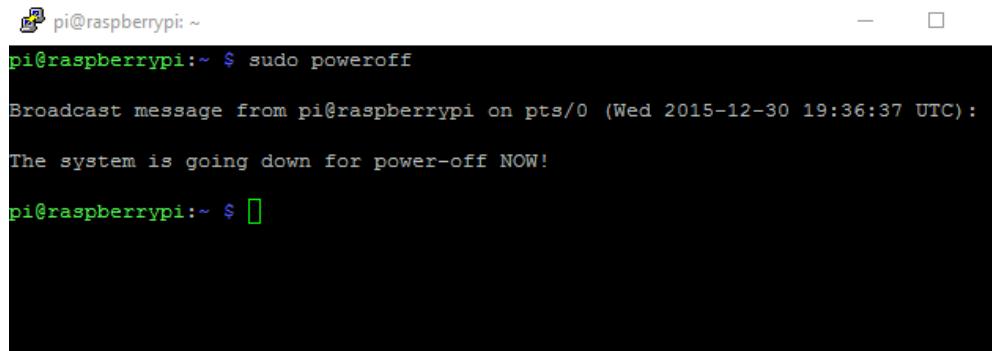
Now you have an SSH communication established with your Raspberry Pi. This will be useful to install software in your Pi, run your programs, create folders or files, etc...

Shutting Down and Rebooting

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

```
pi@raspberry:~ $ sudo poweroff
```

You see the following information after you use the shutdown command:



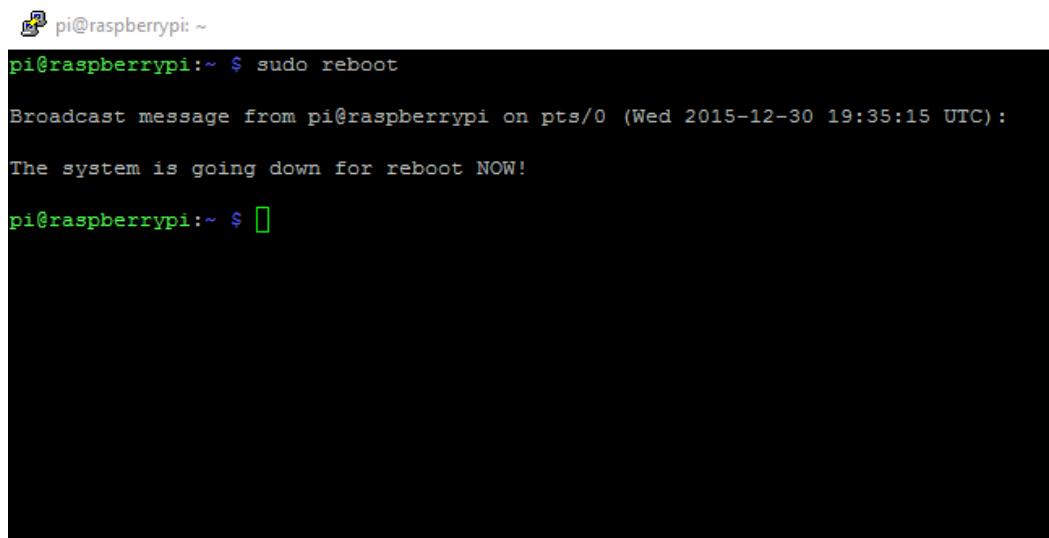
A screenshot of a terminal window on a Raspberry Pi. The window title bar shows a small icon of a computer monitor and the text "pi@raspberrypi: ~". The terminal itself has a black background with white text. It displays the command "sudo poweroff" being entered, followed by a broadcast message from the pi user on pts/0 at 19:36:37 UTC on Wednesday, December 30, 2015. The message states "The system is going down for power-off NOW!" and ends with a prompt for further input.

```
pi@raspberrypi:~ $ sudo poweroff
Broadcast message from pi@raspberrypi on pts/0 (Wed 2015-12-30 19:36:37 UTC):
The system is going down for power-off NOW!
pi@raspberrypi:~ $ 
```

To reboot, type this:

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

This is the result:



A screenshot of a terminal window on a Raspberry Pi, similar to the previous one. It shows the command "sudo reboot" being entered. A broadcast message follows, identical to the shutdown message: it's from the pi user on pts/0 at 19:35:15 UTC on Wednesday, December 30, 2015, stating "The system is going down for reboot NOW!" and ending with a prompt.

```
pi@raspberrypi:~ $ sudo reboot
Broadcast message from pi@raspberrypi on pts/0 (Wed 2015-12-30 19:35:15 UTC):
The system is going down for reboot NOW!
pi@raspberrypi:~ $ 
```

Read Mac OS X/Linux Version

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. It has grown to be one of the most popular methods for communication between different devices.

Establishing an SSH Communication

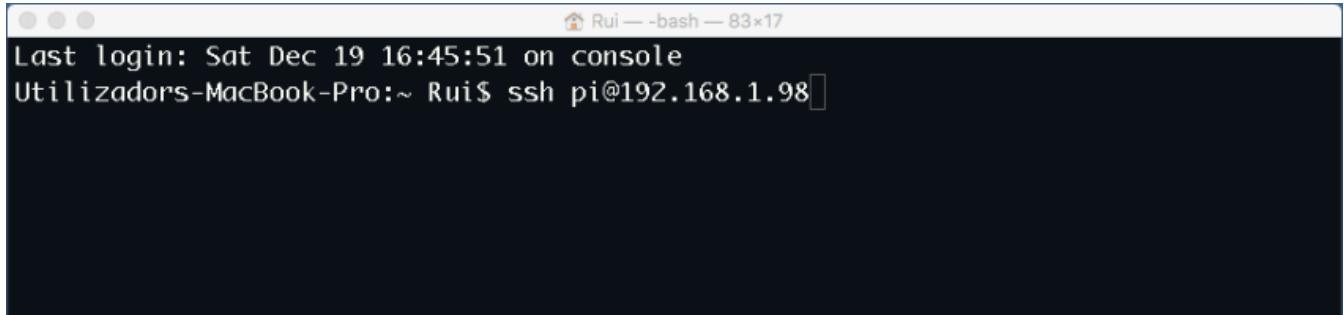
In Mac OS X and Linux, you can use the default Terminal window to establish an SSH communication, because SSH comes in all Unix-based OSes. Follow these steps:

- 1) Boot up your Raspberry Pi
- 2) Open a new Terminal window
- 3) Type the following command:

```
$ sudo ssh pi@ip_address
```

Make sure you replace the **ip_address** part with the real IP address of your Raspberry Pi that you found in the previous Unit. In my case, I'll run the following command:

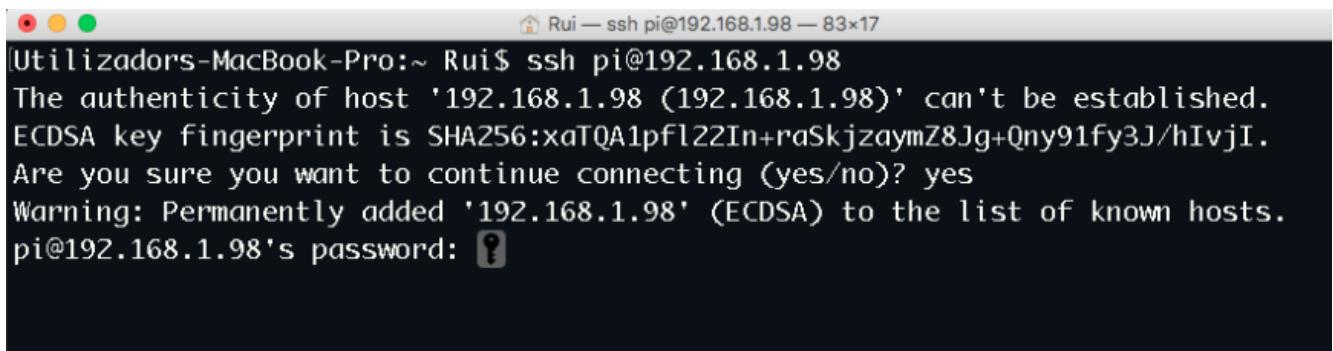
```
$ sudo ssh pi@192.168.1.98
```



```
Rui — -bash — 83x17
Last login: Sat Dec 19 16:45:51 on console
Utilizadores-MacBook-Pro:~ Rui$ ssh pi@192.168.1.98
```

- 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 **raspberry**, 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:xaTQA1pfI22In+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: ?
```

Note: Default settings for Raspbian Lite OS are: username = pi and password = raspberry

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
Utilizadors-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:xaTQA1pflZ2In+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-c5bt0f.lan
pi@raspberrypi:~ $
```

Now you have an SSH communication established with your Raspberry Pi. This will be useful to install software in your Pi, run your programs, create folders or files, etc...

Shutting Down and Rebooting

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

```
pi@raspberry:~ $ sudo poweroff
```

You see the following information after you use the shutdown command:

```
pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo poweroff
Broadcast message from pi@raspberrypi on pts/0 (Wed 2015-12-30 19:36:37 UTC):
The system is going down for power-off NOW!
pi@raspberrypi:~ $
```

To reboot, type this:

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

This is the result:

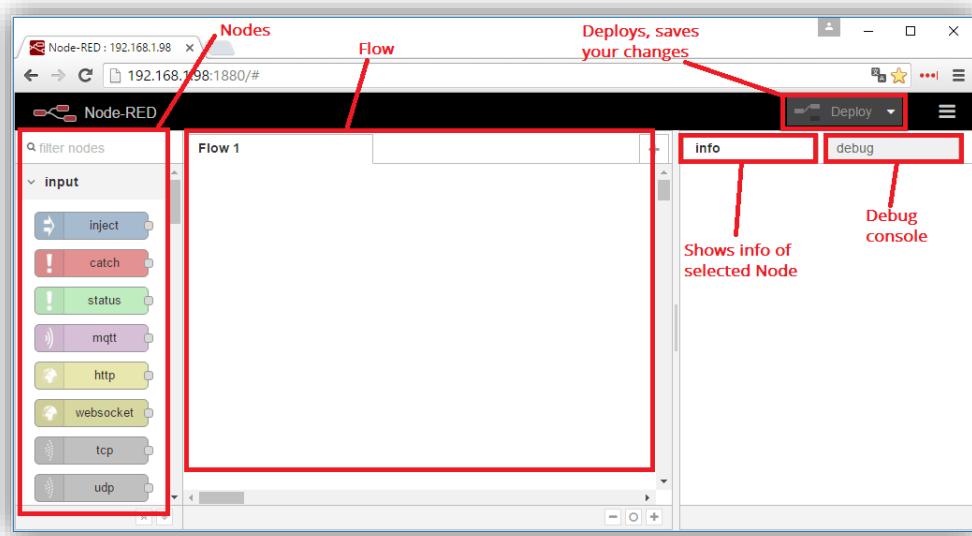
 pi@raspberrypi: ~
pi@raspberrypi:~ \$ sudo reboot

Broadcast message from pi@raspberrypi on pts/0 (Wed 2015-12-30 19:35:15 UTC):

The system is going down for reboot NOW!
pi@raspberrypi:~ \$

Module 3

Getting started with Node-RED

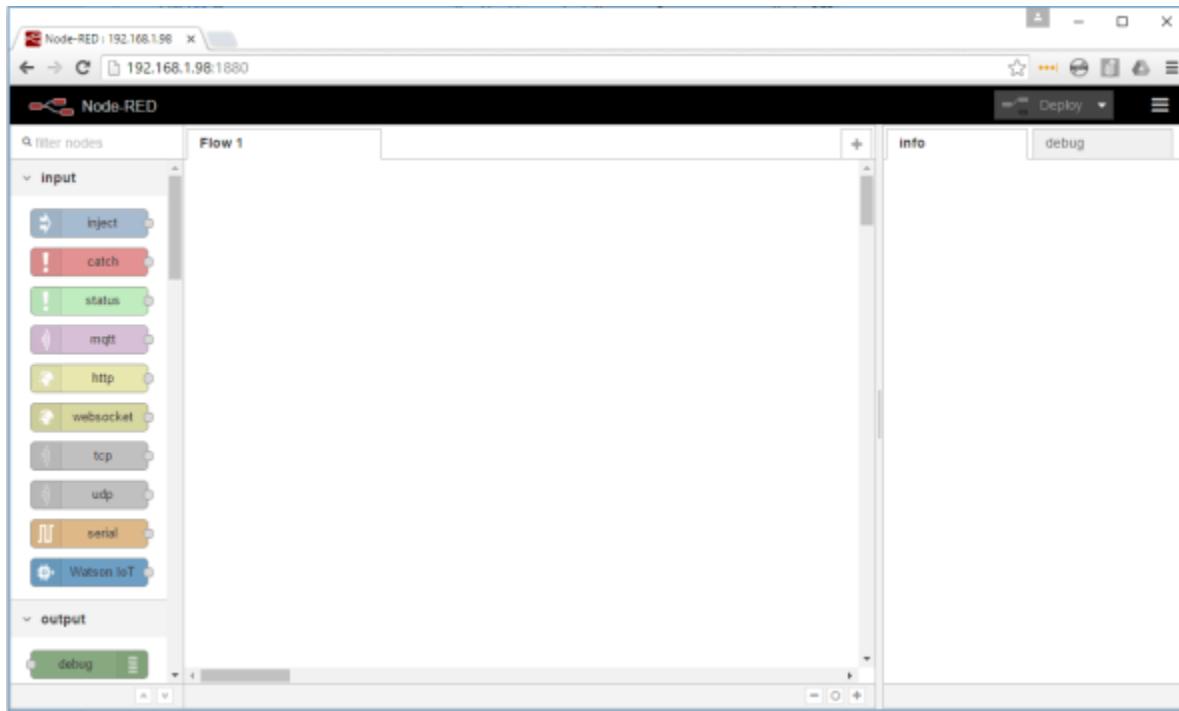


Unit 1 - What's Node-RED?

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

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

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



Why is Node-RED a great solution?

Node-RED is open source and developed by IBM.

The Raspberry Pi runs Node-RED perfectly.

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

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

What can you do with Node-RED?

Node-RED makes it easy to:

- Access your RPi GPIOs
- Establish an MQTT connection with other boards (Arduino, ESP8266, etc)
- Create a responsive graphical user interface for your projects
- Communicate with third-party services (IFTTT.com, Adafruit.io, Thing Speak, etc)
- Retrieve data from the web (weather forecast, stock prices, emails. etc)
- Create time triggered events
- Store and retrieve data from a database

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

Let's install it!

Unit 2 - Installing Node-RED

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

Having an SSH connection established with your Raspberry Pi, enter the following commands to update and upgrade your Pi packages:

```
pi@raspberry:~ $ sudo apt-get update && sudo apt-get upgrade
```

Then install Node-RED by typing:

```
pi@raspberry:~ $ sudo apt-get install nodered
```

You'll be prompted to confirm the installation, type Y and press Enter.

The installation should be completed after a couple of minutes.

Autostart Node-RED on boot

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

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

Now, restart your Pi so the autostart takes effect:

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

Testing the Installation

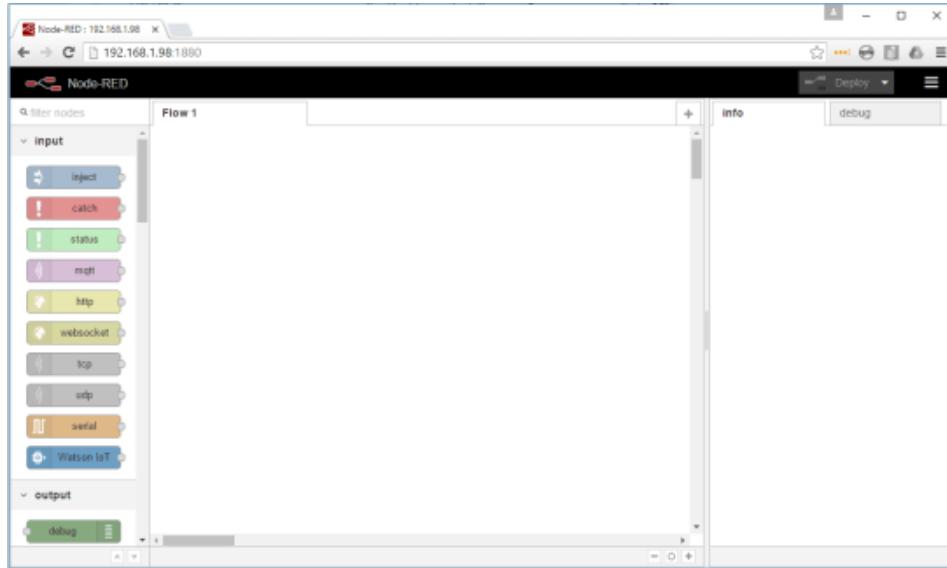
When your Pi is back on, you can test the installation by entering the IP address of your Pi in a web browser followed by the 1880 port number:

```
http://YOUR_RPi_IP_ADDRESS:1880
```

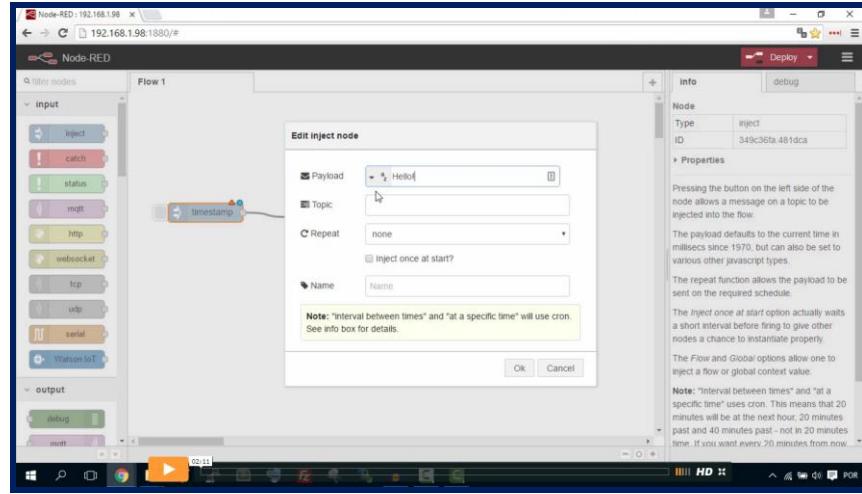
In my case is:

```
http://192.168.1.98:1880
```

A page like this loads:



Unit 3 - Node-RED overview



Video # 5 - <https://RNTLab.com/28hasvideos>

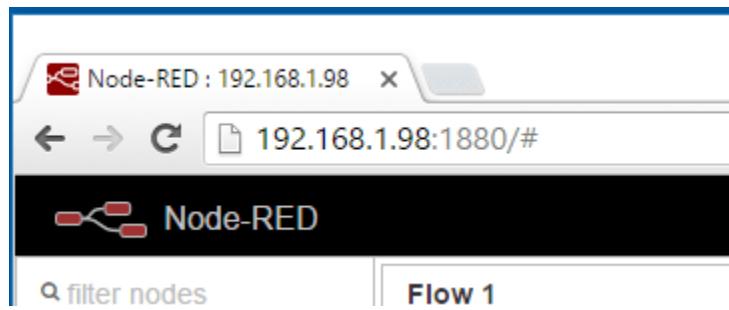
Let's take a look at the Node-RED visual interface.

In order to access the Node-RED program, type in your browser your Raspberry Pi IP address followed by the port number **1880**.

http://YOUR_RPi_IP_ADDRESS:1880

In my case is:

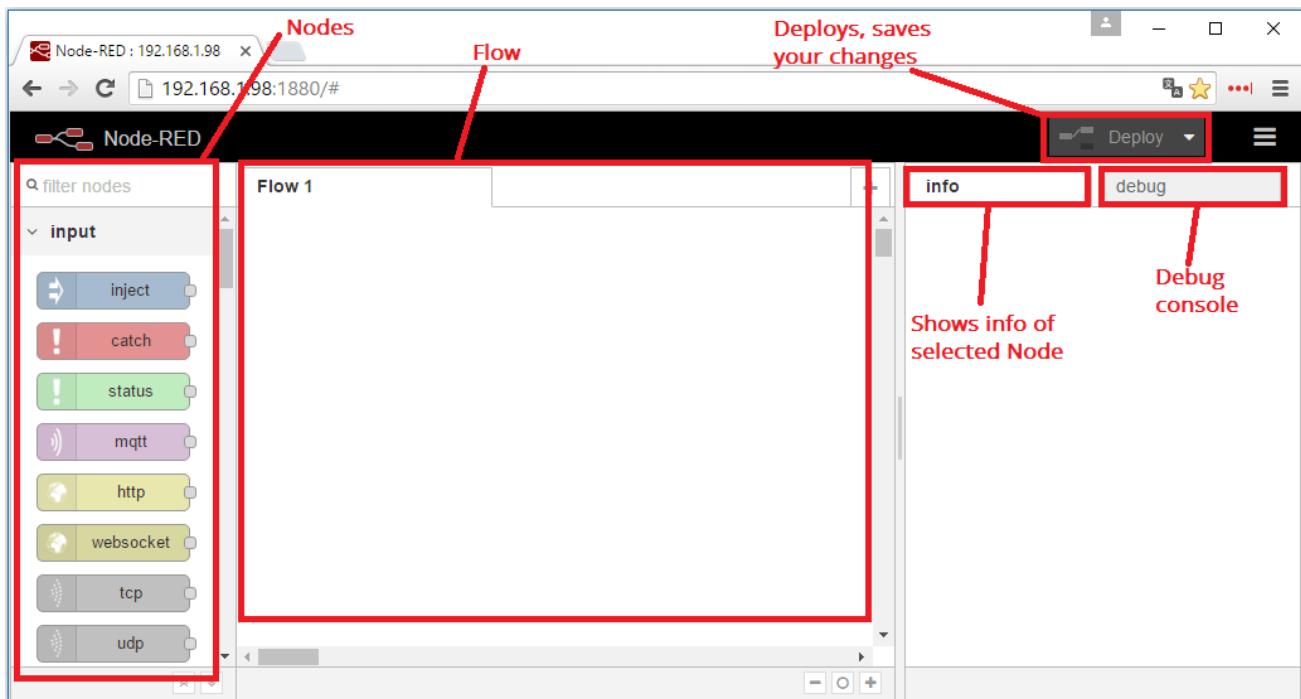
http://192.168.1.98:1880



Main sections

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

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



Creating a simple flow

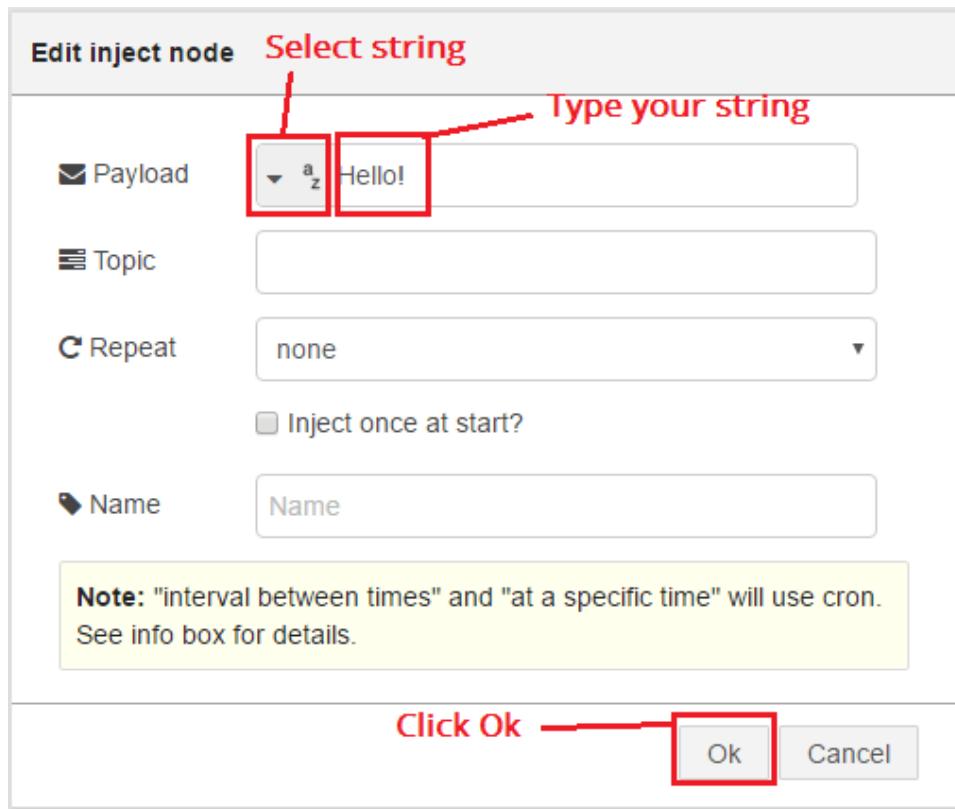
Let's test a simple example of a flow. Start by dragging an **Inject** node to your flow. Then, also drag a **Debug** node.



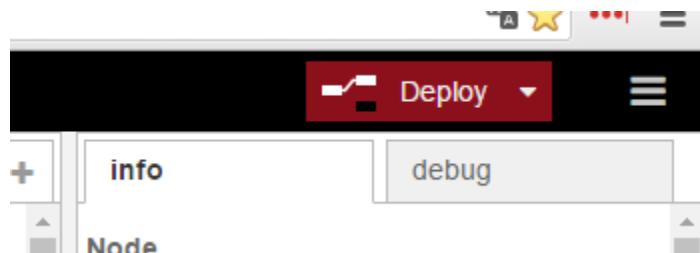
Connect your nodes together. You can drag them to confirm that they are connected.

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

Select **string** and type **Hello!**.



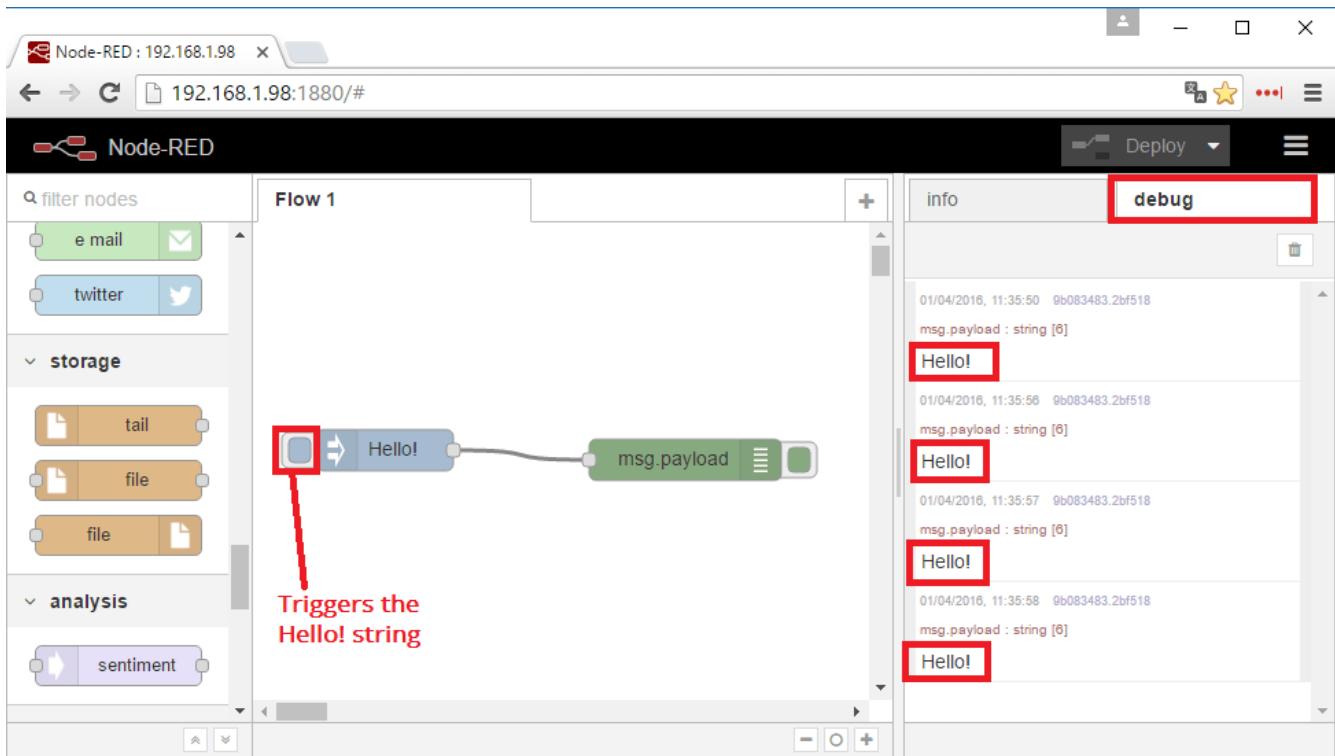
To save your application, you need to click the **Deploy** button on the top right corner.



Your application is saved.

Testing the flow

Let's test our simple flow. Open the debug window and click the Inject node to trigger the "Hello!" string.



As you can see, our message is being printed in the **debug** window. 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.

Go to the next Unit to learn how to control an LED with Node-RED.

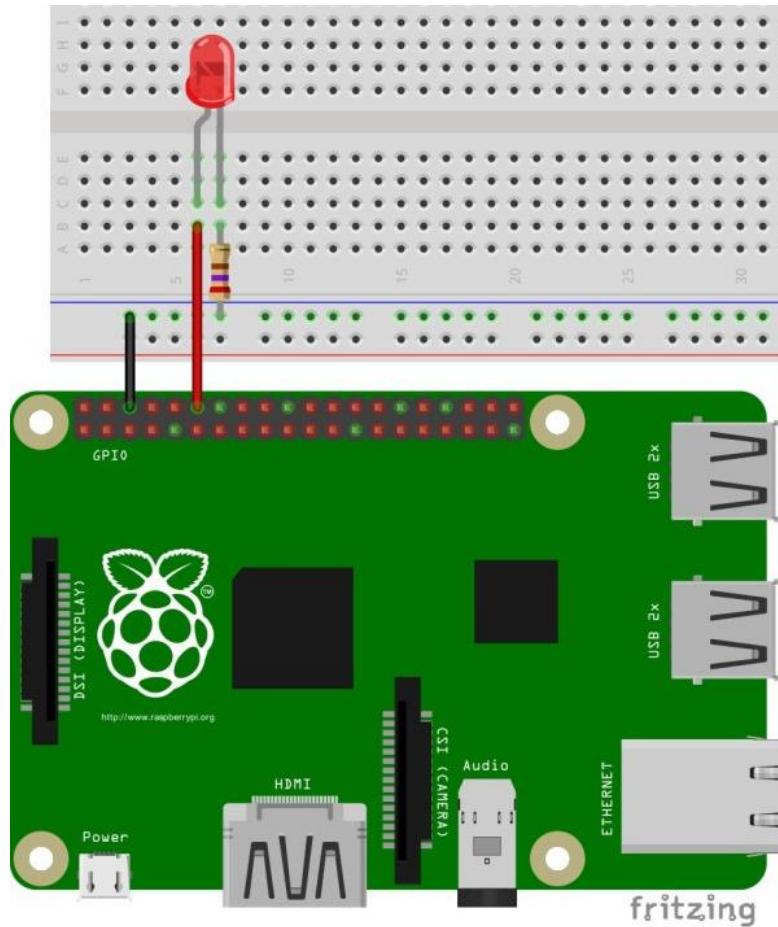
Unit 4 - Controlling an LED with Node-RED

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 with Node-RED.

Schematics

Here's the hardware required to complete this project:

- LED (5mm)
- 270Ω resistor
- Breadboard
- Wire cables

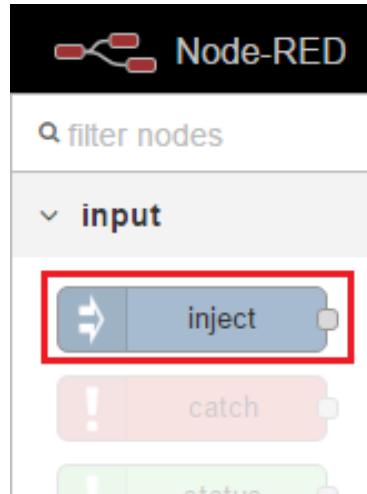


Wire your circuit by following the image above or these instructions:

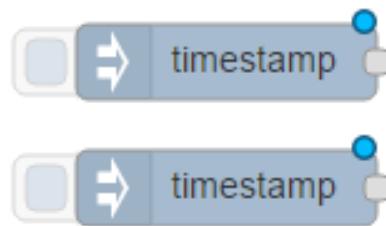
- Longest lead of the LED -> pin 12
- Shortest lead of the LED -> 270Ω resistor
- Resistor connected to GND
- Pin 6 is GND

Preparing your flow

Drag two **Inject** nodes into your flow.



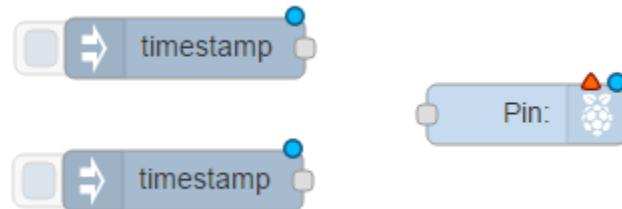
They will look like this in your flow:



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

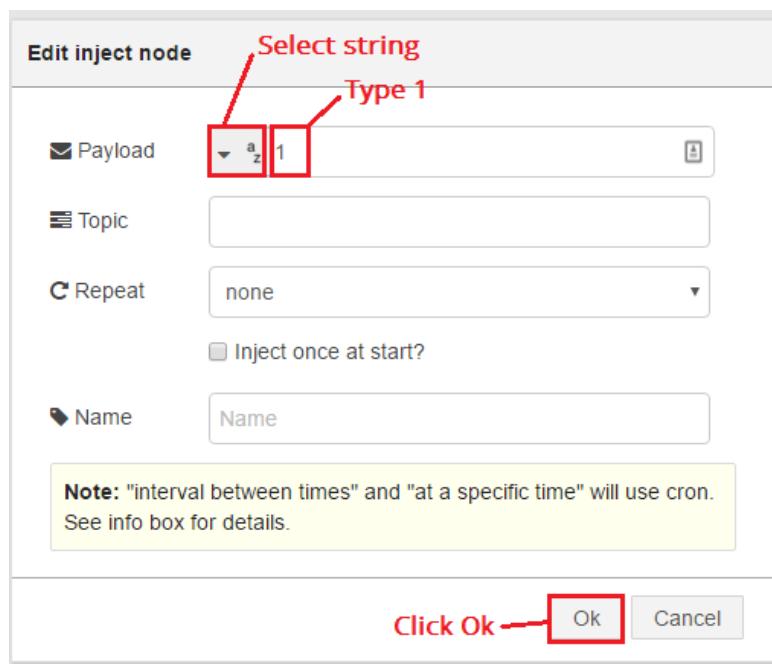


Here's what you should see:

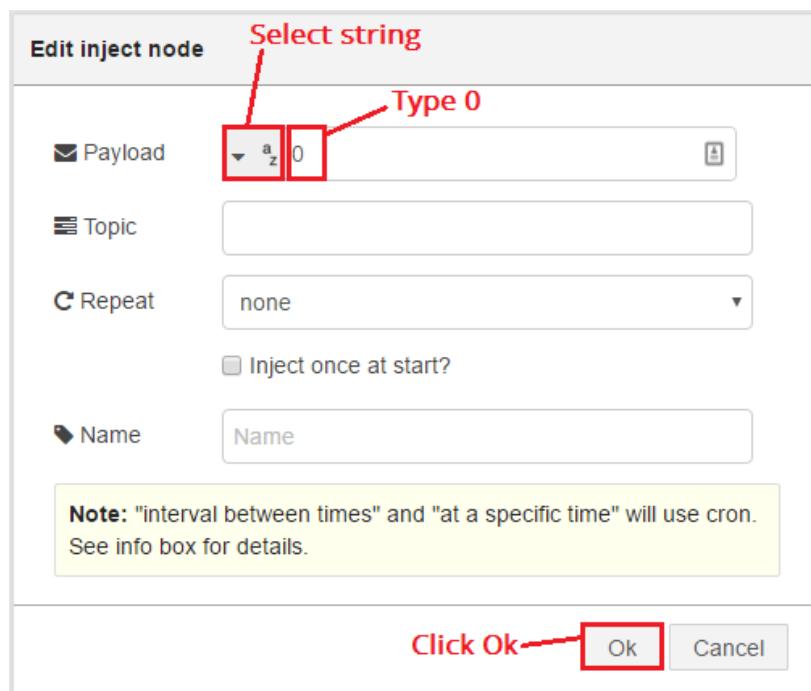


Double-click the first **Inject** node to edit its properties. Select **string** and type 1.

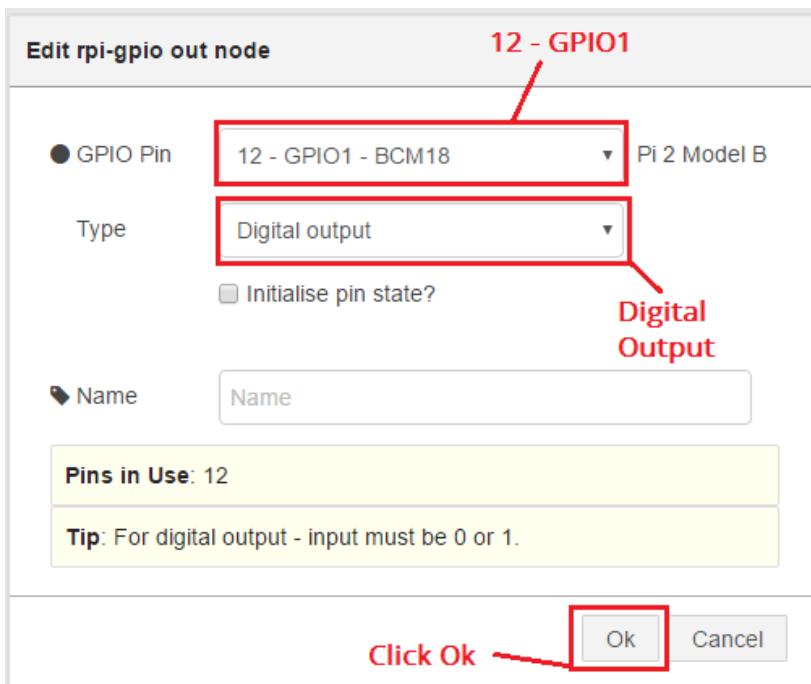
Press **Ok**.



Edit Inject node number two. Select **string** and type 0. Press **Ok**.

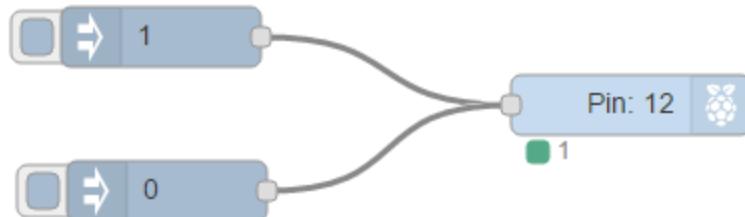


Lastly, edit the rpi-gpio out node. Set the type as a **digital output** and the GPIO1 – Pin 12 (that's where your LED is plugged in).

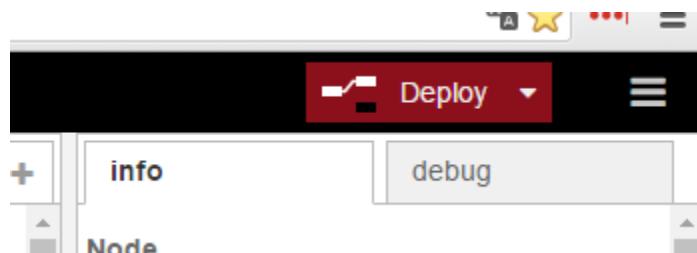


Deploying the application

After editing each node, your flow should look like this:



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

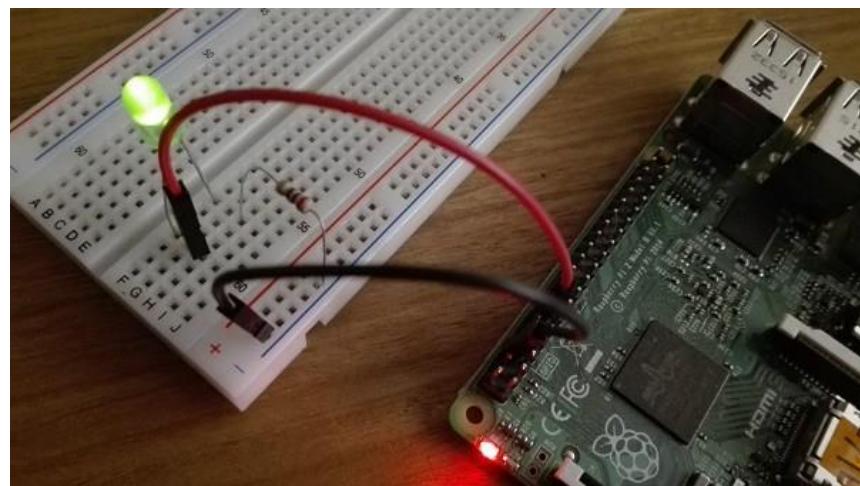


Testing the application

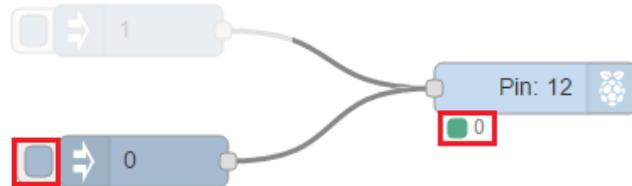
Now, press the square of the **Inject 1** node.



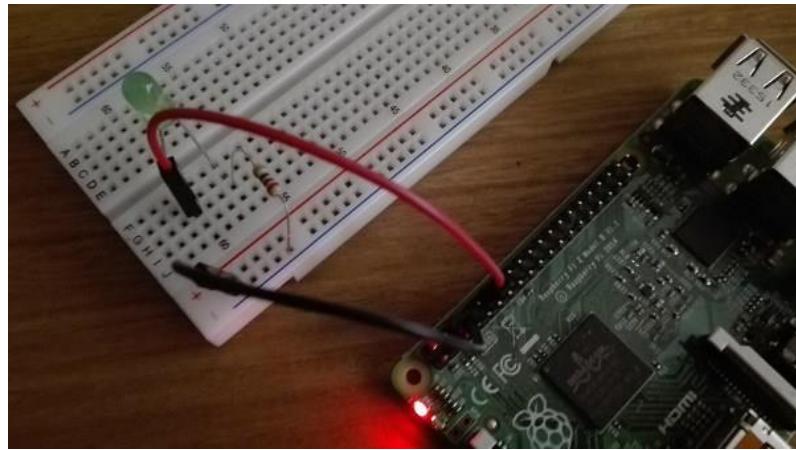
Your LED turns on:



And when you press the **Inject 0** node.



Your LED turns off:

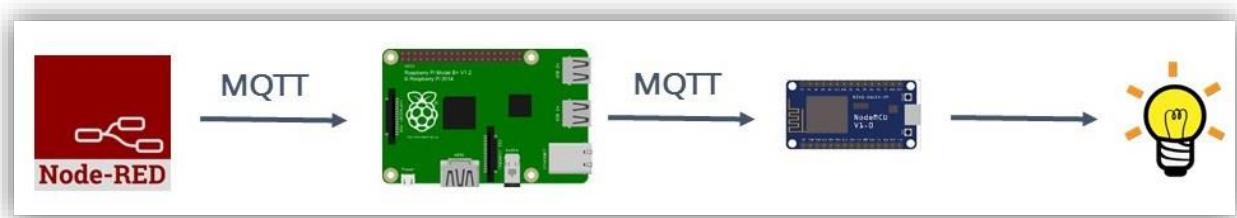


This shows how easy it is to access your Raspberry Pi GPIOs with Node-RED.

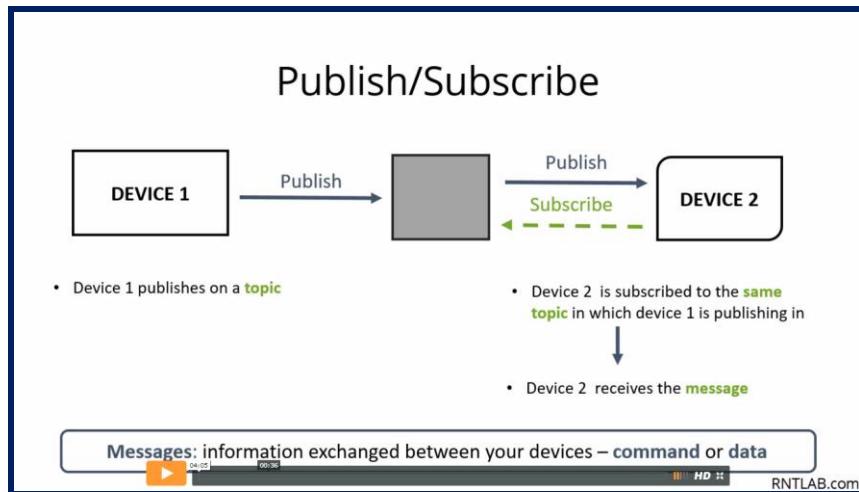
In the next Module, we're going to explore the MQTT protocol to integrate with your RPi and other popular development boards (Arduino and ESP8266).

Module 4

Experimenting with MQTT



Unit 1 - What is MQTT?



Video # 6 - <https://rntlab.com/28hasvideos>

Introducing MQTT

In this Unit, I'm going to introduce you to the MQTT protocol.

MQTT stands for MQ Telemetry Transport.

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

It makes it really easy to establish a communication between multiple devices.



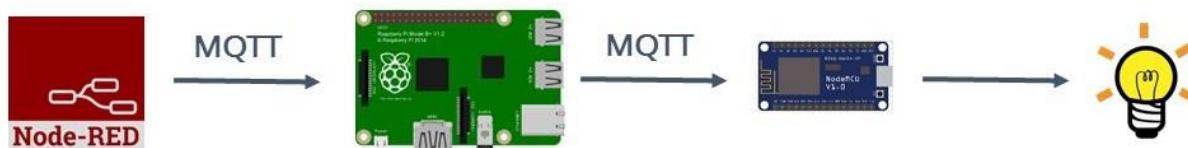
It is a simple messaging protocol, designed for constrained devices and with low-bandwidth.

So, it's the perfect solution for Internet of Things applications.

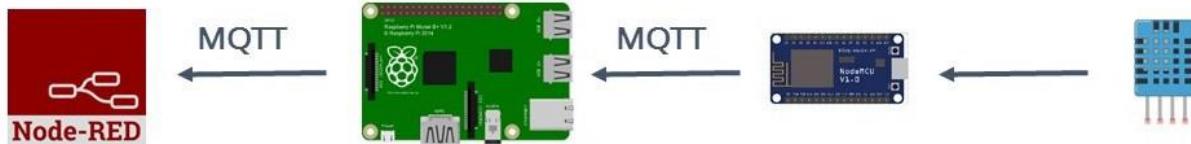
High Level Overview

Here's a quick high level overview of what MQTT allows you to do.

You can **send a command to control an output**:



Or you can **read data from a sensor and publish it**:



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 and subscribe system.

What does that mean?

This means that a device can publish messages to your devices. Or your device can subscribe to a particular topic to receive those messages.



For example **device 1 publishes** on a topic.

Device 2 is subscribed to the same topic as **device 1 is publishing in**.

So, **device 2 receives** the message.

MQTT – Messages

Messages are the information that you want to exchange between your devices. Whether it's a command or data.

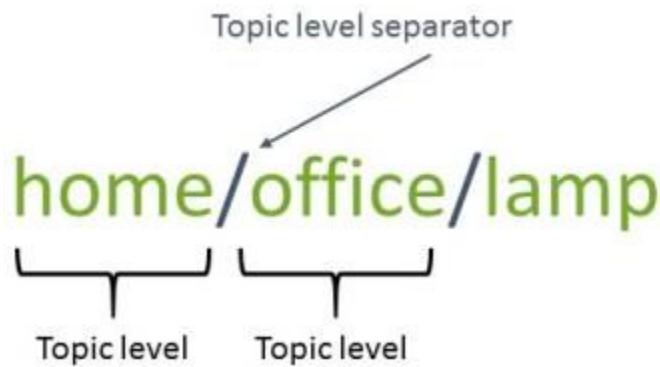
With the publish and subscribe system you can do pretty much anything you could ever wanted in your home automation projects.

MQTT – Topics

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

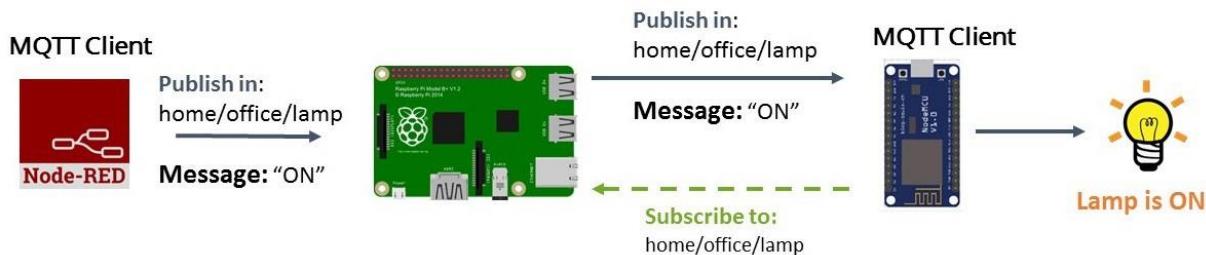
Topics are represented with strings separated by slashes. The slashes indicate the topic level.

Here's an example on how you would create a topic for a lamp in your home office:



For example if you would like to turn on a lamp in your home office you would publish a message to a topic using Node-RED saying “ON”.

Your ESP8266 would be subscribed to that same topic, so it would receive the “ON” message and finally turn on the lamp.



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

home/office/lamp



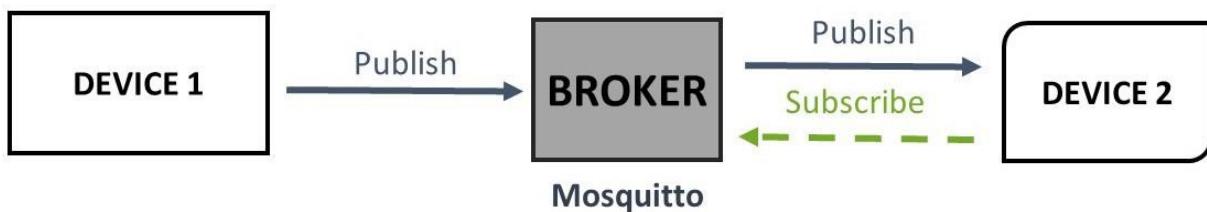
home/office/Lamp

MQTT – Broker

At last, you also need to be aware of the term broker.

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

There are several brokers you can use. We're going to use the [Mosquitto broker](#) which can be installed in the Raspberry Pi.



In the next Units, we will experiment with MQTT and Node-RED to see how everything works with practical examples.

Unit 2 - Installing Mosquitto Broker

Let's install the [Mosquitto](#) broker. The broker receives all messages, filters the messages, decide who is interested in it and then publishes the messages to all subscribed clients.



Updating the Repository

To update the repository you should first import the repository package signing key:

```
pi@raspberry:~ $ wget http://repo.mosquitto.org/debian/mosquitto-repo.gpg.key  
pi@raspberry:~ $ sudo apt-key add mosquitto-repo.gpg.key
```

Make the repository available to be installed with apt-get:

```
pi@raspberry:~ $ cd /etc/apt/sources.list.d/  
pi@raspberrypi:/etc/apt/sources.list.d $
```

Then, run the following command:

```
pi@raspberrypi:/etc/apt/sources.list.d $ sudo wget  
http://repo.mosquitto.org/debian/mosquitto-jessie.list
```

Go back to the root directory:

```
pi@raspberrypi:/etc/apt/sources.list.d $ cd  
pi@raspberry:~ $
```

Finally, update apt-get information:

```
pi@raspberry:~ $ sudo apt-get update
```

Installing

To install the Mosquitto broker enter the following command:

```
pi@raspberry:~ $ sudo apt-get install mosquitto
```

You'll have to type **Y** and press **Enter** to confirm the installation.

Testing

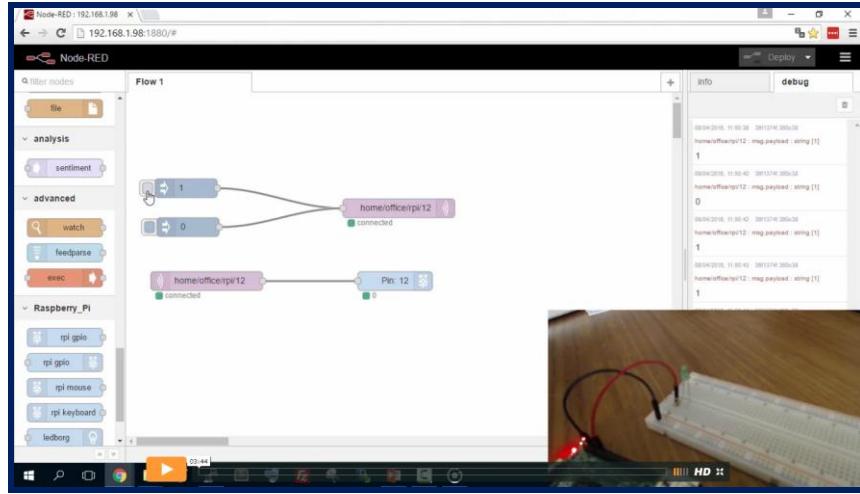
Send the command:

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

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

```
pi@raspberrypi:~ $ mosquitto -v  
1460043031: mosquitto version 1.4.8 (build date
```

Unit 3 - Establishing an MQTT communication with Node-RED



Video # 7 - <https://rntlab.com/28hasvideos>

In this Unit, we're going to establish an MQTT communication using Node-RED nodes.

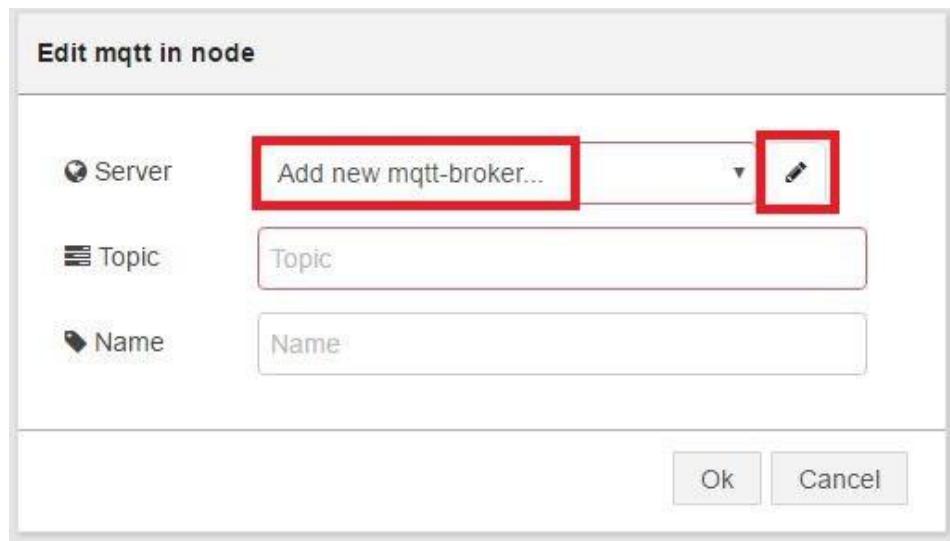
Creating the Publish Flow

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

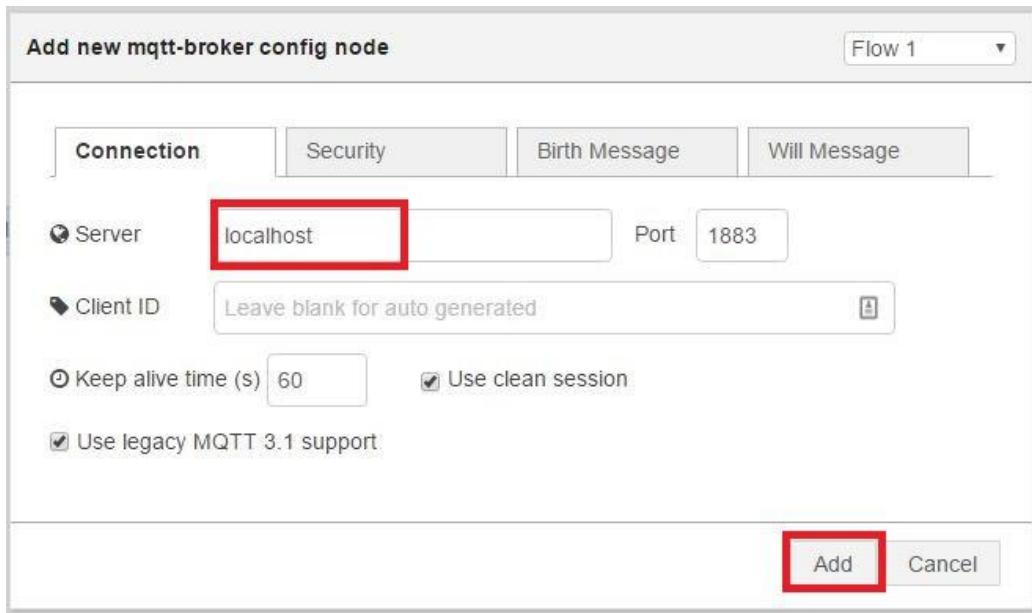


You need to connect Node-RED to your MQTT broker. Double-click the **MQTT output** node.

Click the Add new mqtt-broker option.



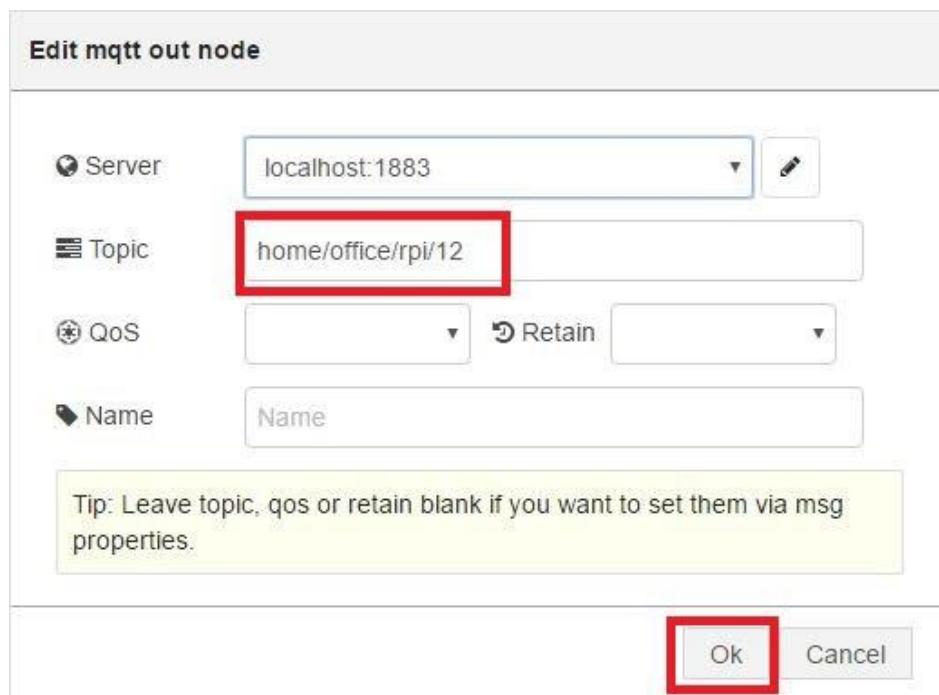
Type **localhost** in the server field and all the other settings are configured properly by default.



Press **Add** and the **MQTT output** node automatically connects to your broker.

Now, let's imagine you would like to control the Raspberry Pi pin 12, and your RPi is located in your home office.

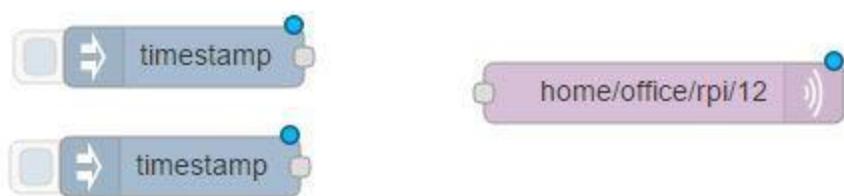
A good choice for a topic that controls pin 12 would be `home/office/rpi/12`.



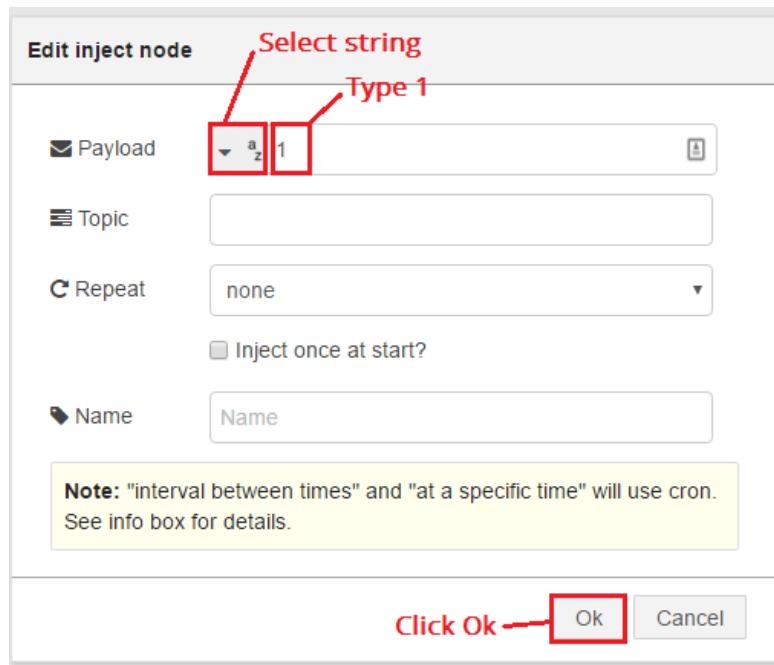
Press **Ok** to save your node.

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

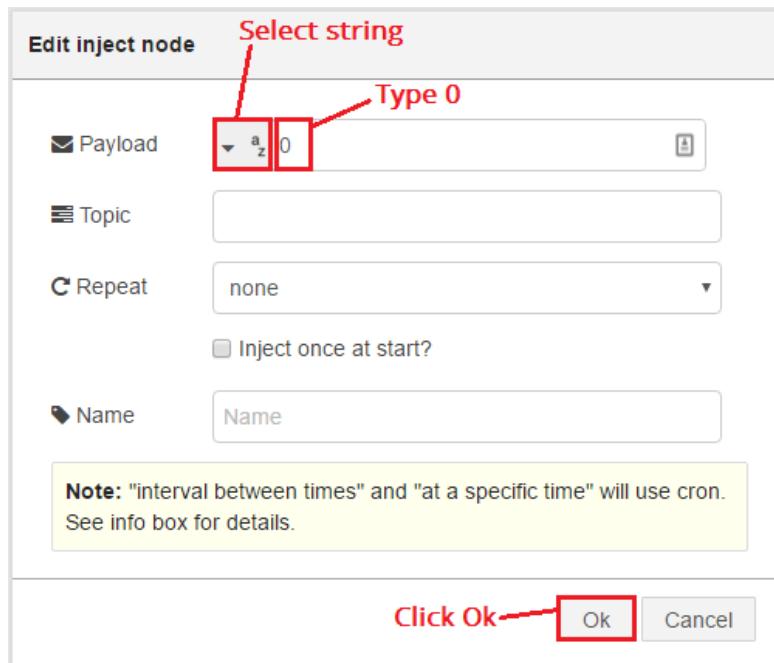
Drag two **Inject** nodes to your flow.



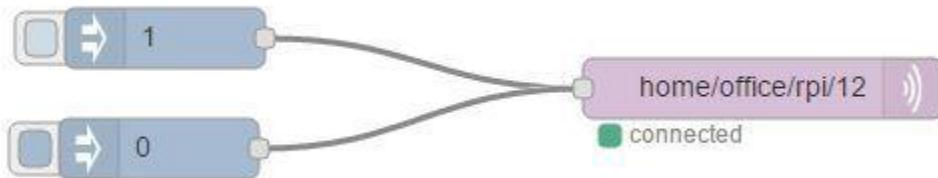
Double-click the first **Inject** node to edit its properties. Select **string** and type 1. Press Ok.



Edit Inject node number two. Select **string** and type 0. Press Ok.



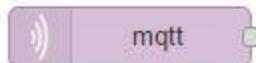
Connect both **Inject** nodes to the MQTT output node.



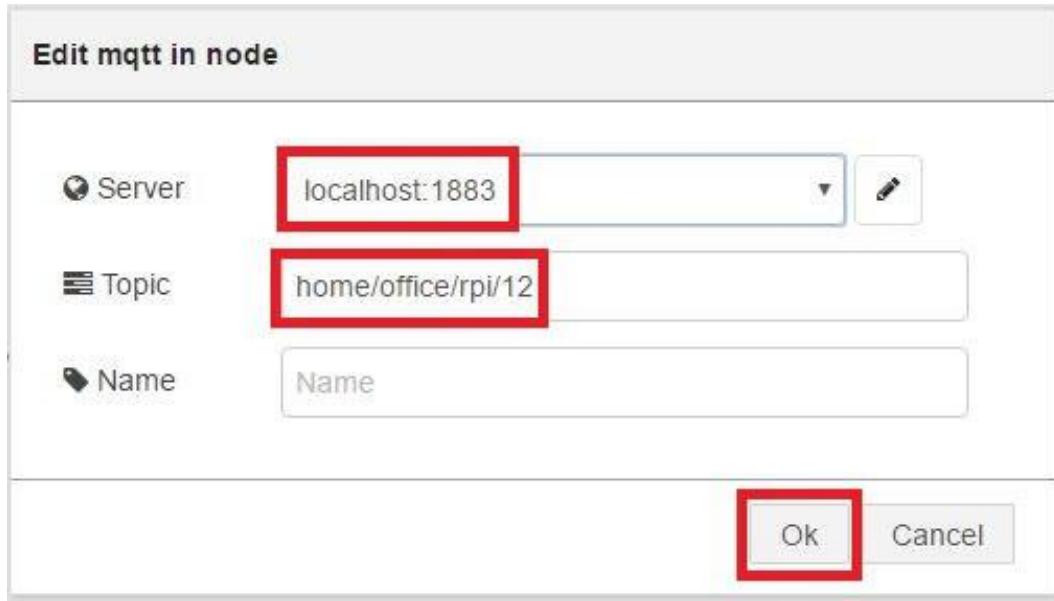
Creating the Subscribe Node

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

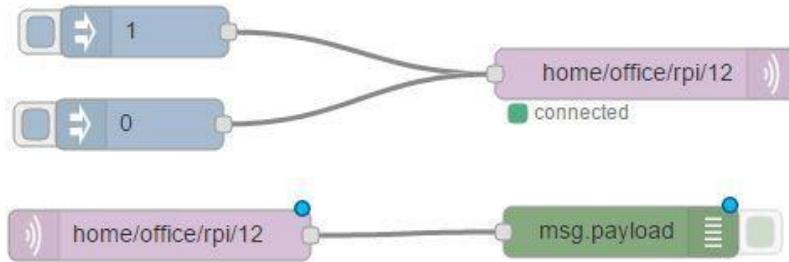
Drag an **MQTT input** node to the flow.



Select the MQTT broker. Type the exact topic created previously.

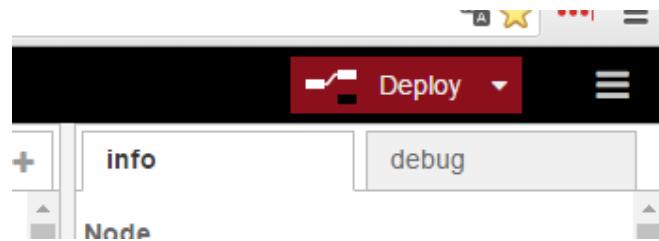


Drag a **Debug** node, so we can print the message received. Connect both nodes.



Testing the MQTT Connection

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



Open the **debug** window.

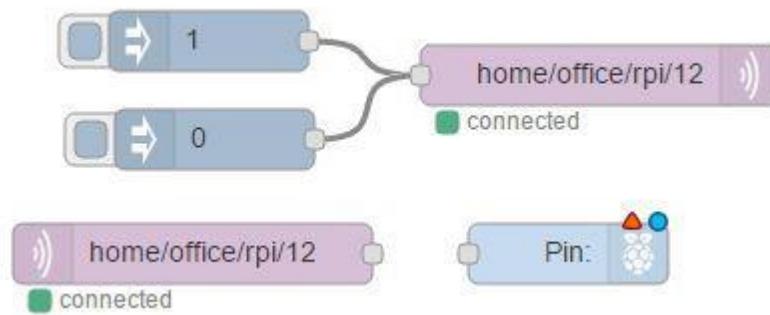
Trigger the **Inject** node with 0 and 1. As you can see the message is being received and printed in the **debug** window.



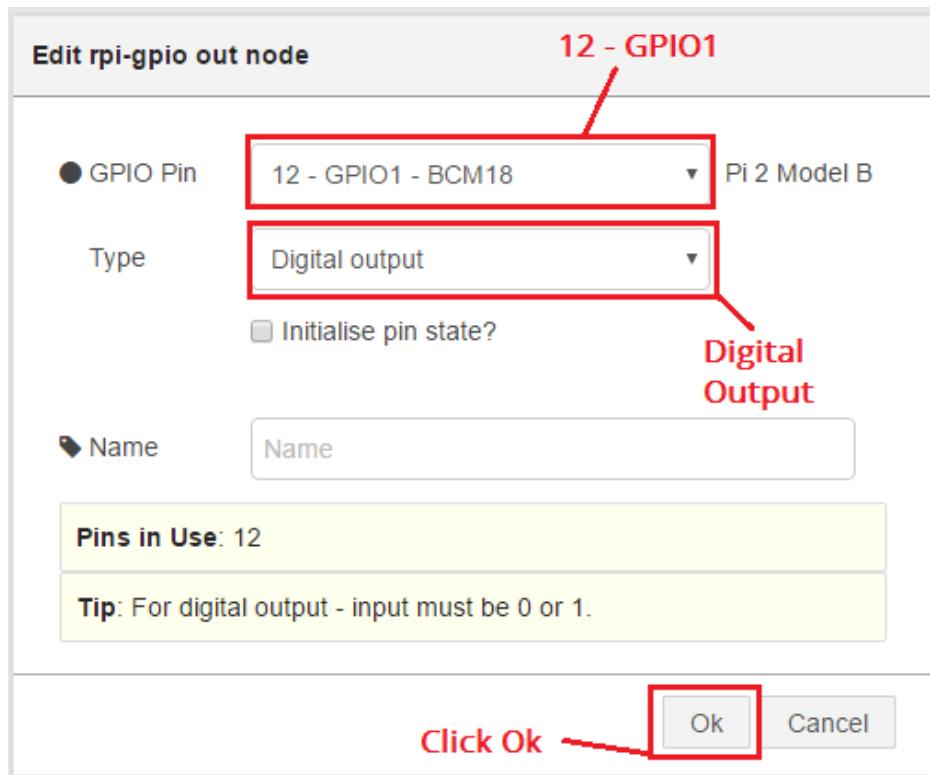
This is how you make a publish and subscribe system.

Connecting the rpi gpio out node

Now, delete the debug node. Drag a rpi-gpio out node.



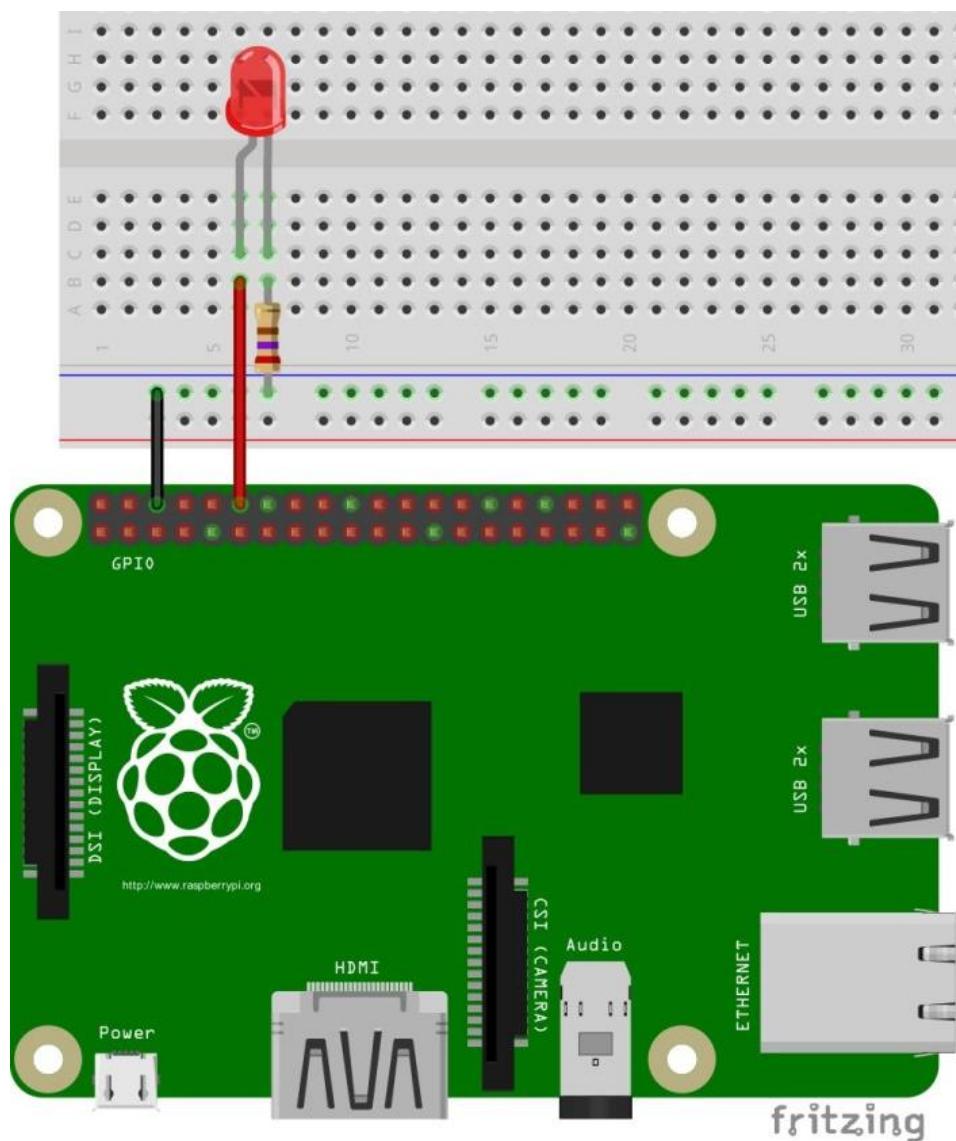
Lastly, edit the rpi-gpio out node. Set the type as a **Digital output** and the GPIO Pin 12 (that's where your LED will be connected to). Click Ok.



Schematics

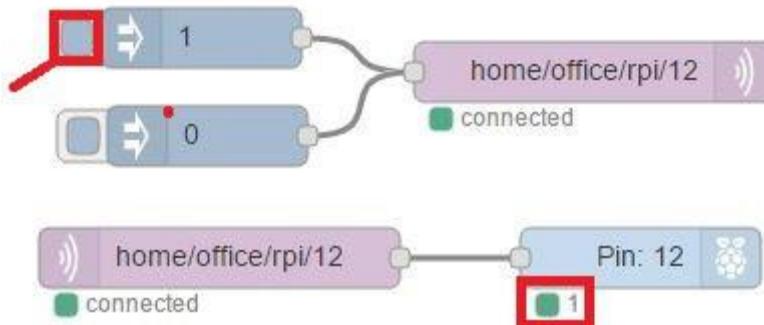
Here's the hardware required to complete this project and connect an LED to pin 12:

- LED (5mm)
- 270Ω resistor
- Breadboard
- Wire cables

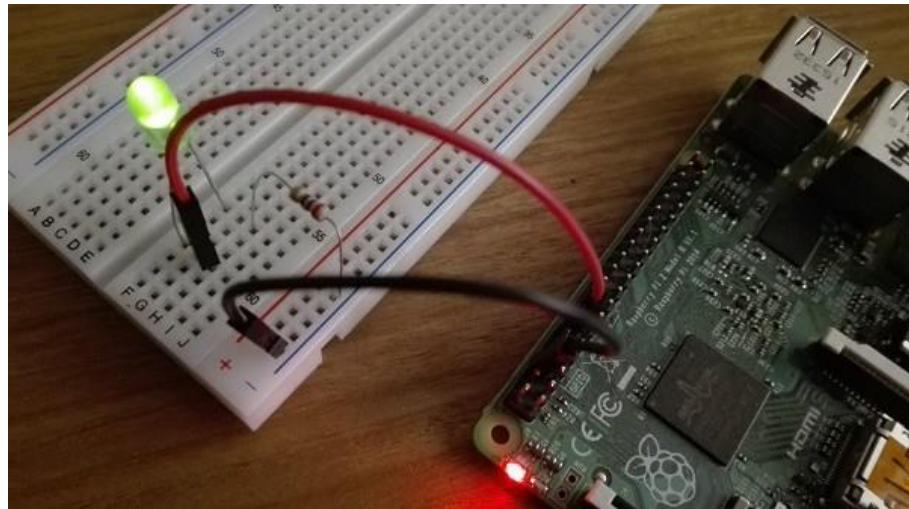


Deploy your application and now you can control the LED on and off.

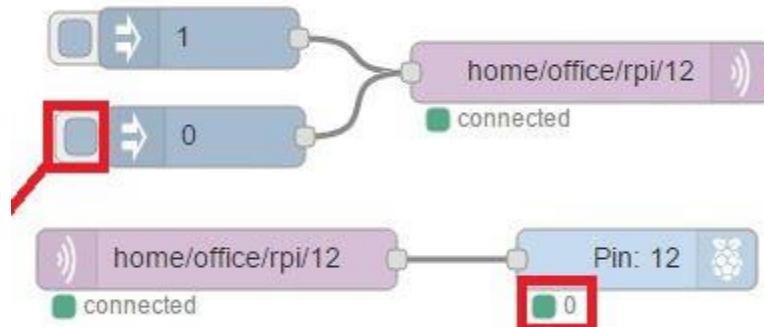
Now, press the square of the **Inject 1** node.



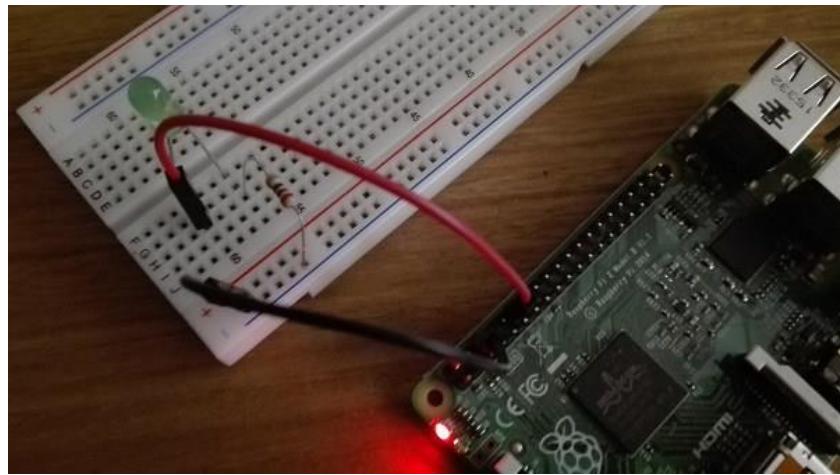
Your LED turns on:



And when you press the 0 Inject node.



Your LED turns off:



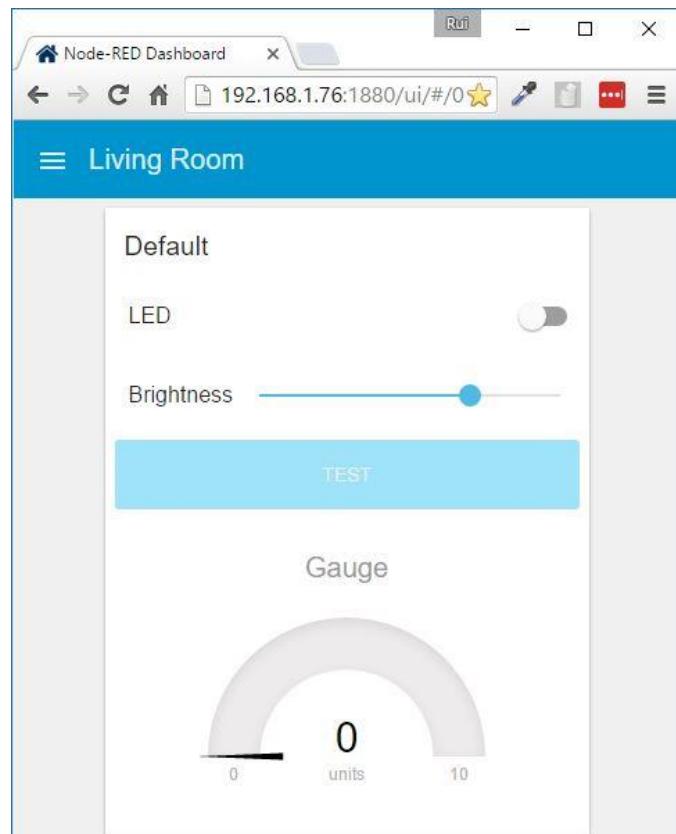
Keep in mind this is just a basic demonstration and these nodes can be connected to your Arduino and your ESP8266.

In the next Module, you're going to learn how to design the graphical user interface to control the GPIOs with the MQTT protocol.

In the future Modules, you'll learn how to connect the Arduino and the ESP8266 to your main server and control any output or read any sensor data.

Module 5

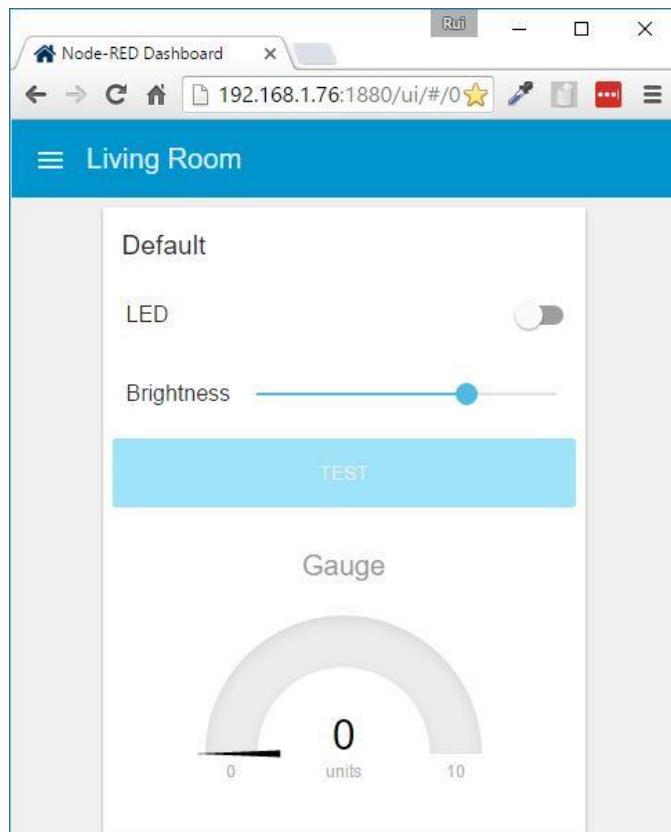
Designing the Graphical User Interface



Unit 1 - Installing Node-RED Dashboard

In this Module you're going to install a Node-RED package called **Node-RED Dashboard**.

This package allows you to add buttons, sliders, switches, charts, gauges and much more to your projects.



Here's two links that might be helpful to extend your knowledge about this package:

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

Installing Node-RED Dashboard

To install Node-RED Dashboard run these 5 commands:

```
pi@raspberry:~ $ sudo apt-get install npm  
pi@raspberry:~ $ sudo npm install -g npm@2.x  
pi@raspberry:~ $ hash -r  
pi@raspberry:~ $ cd ~/.node-red  
pi@raspberry:~/node-red $ sudo npm install node-red-dashboard
```

Rebooting your Raspberry Pi

Reboot your Raspberry Pi:

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

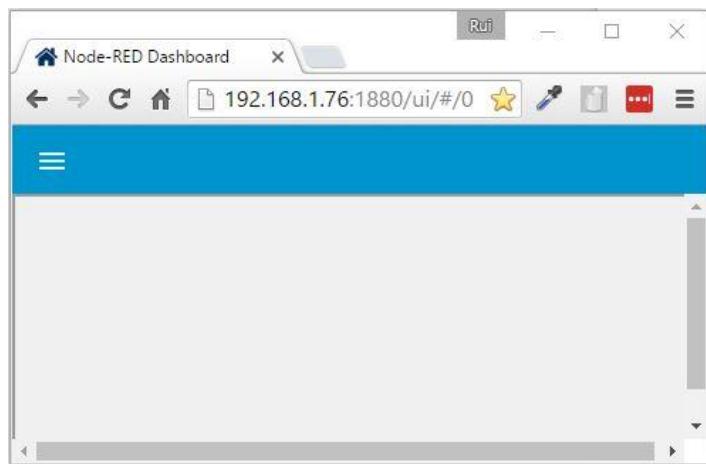
When your Pi is back on, you can open the UI by entering the RPi IP address in a web browser followed by :1880/ui as follows:

```
http://YOUR_RPi_IP_ADDRESS:1880/ui
```

In my case is:

```
http://192.168.1.98:1880/ui
```

A page like this loads:



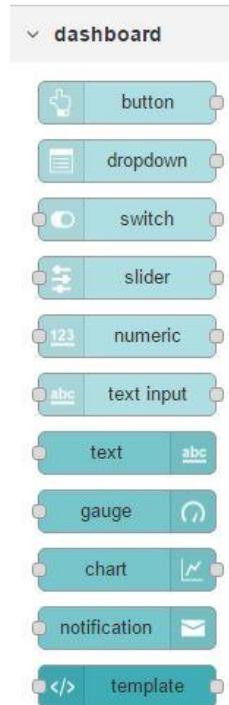
Unit 2 - Experimenting with Node-RED Dashboard

If you've followed the previous Unit, you have Node-RED Dashboard installed in your Raspberry Pi. Open two tabs in your web browser, one with Node-RED and the other with Node-RED Dashboard:

```
http://YOUR_RPi_IP_ADDRESS:1880  
http://YOUR_RPi_IP_ADDRESS:1880/ui
```

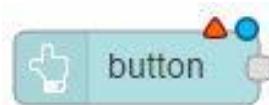
Now that you have Node-RED Dashboard installed, if you scroll down, you can see a new section of nodes called Dashboard.

At a first glance, you see that you have the option to create buttons, dropdowns, switches, sliders, text inputs and more...



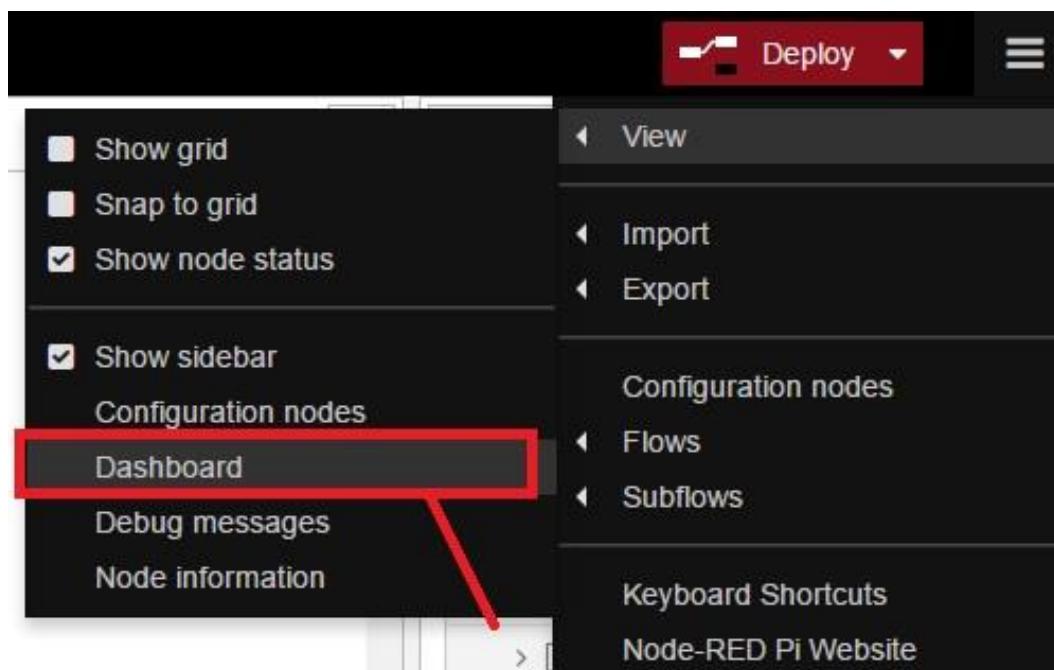
Button Node

Let's start by dragging a **Button** node to the flow.



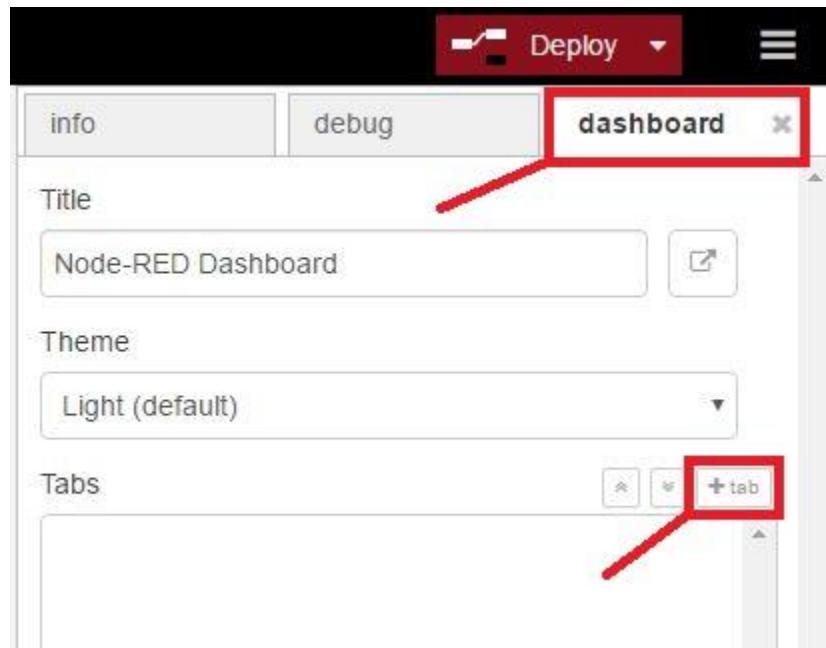
If you Deploy your application and go to the Dashboard page. You see that the Dashboard is completely empty at the moment.

In your Dashboard, you can create tabs. Tabs are particular useful to define your rooms or the different sections in your home. Open the tab manager window. Go to the top menu > View > Dashboard.

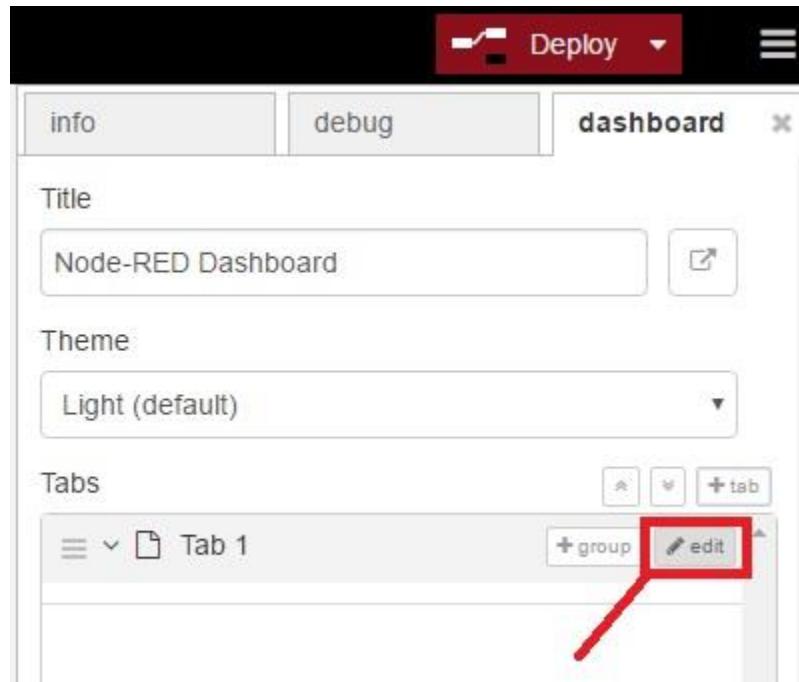


A new window called dashboard opens in the right side.

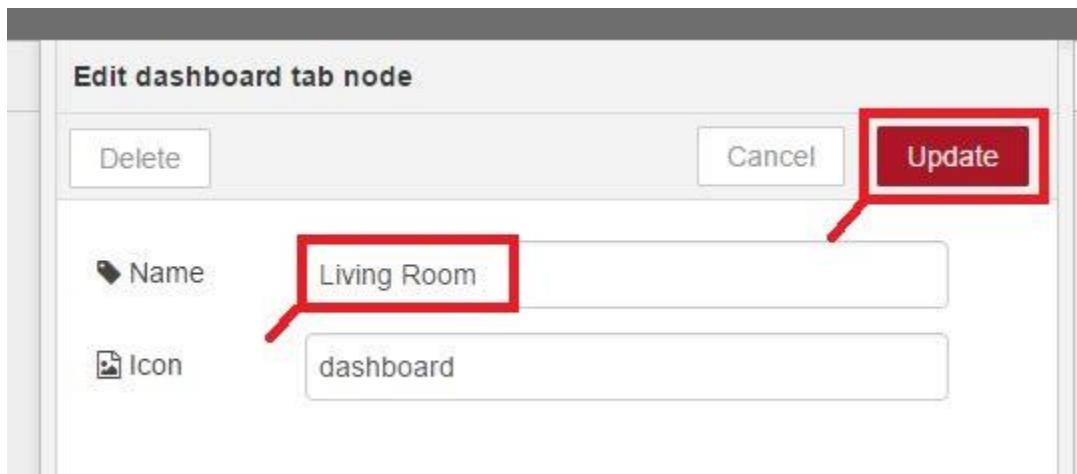
That's where you can create tabs, groups and organize the buttons of the interface. Click the **+tab** button.



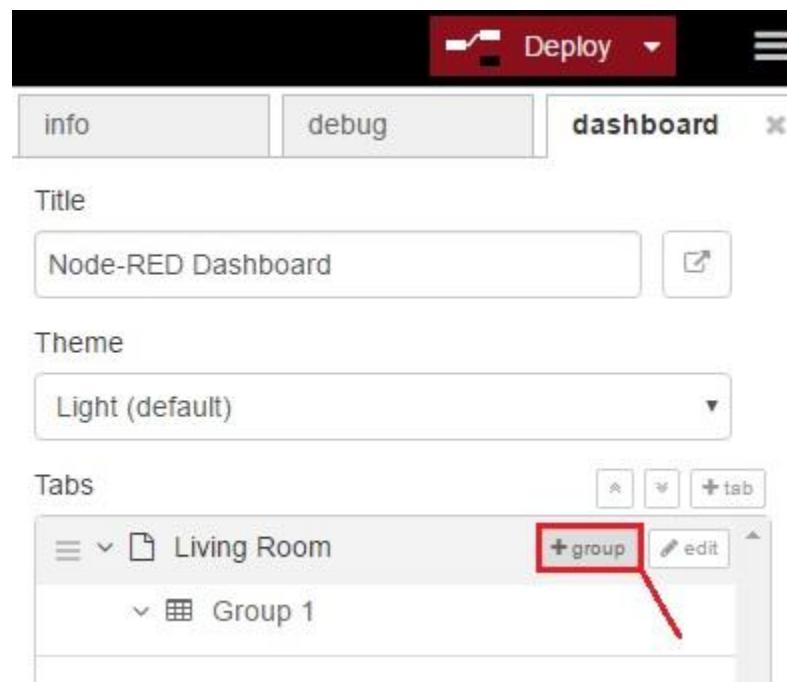
Then click the **edit** button.



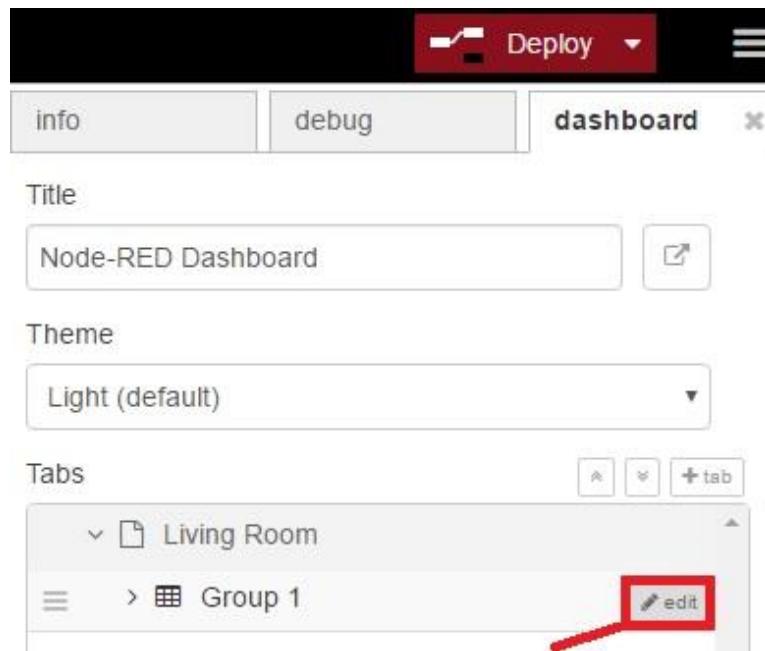
Call it **Living Room** and click the Update button.



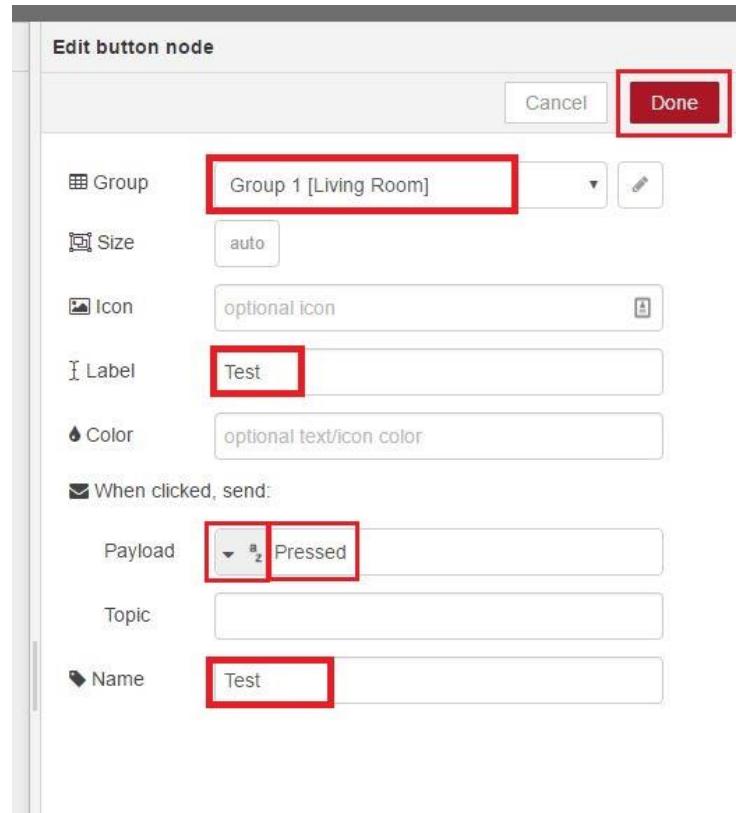
Finally, you have to set a group.



You can change the group name by clicking **edit**.



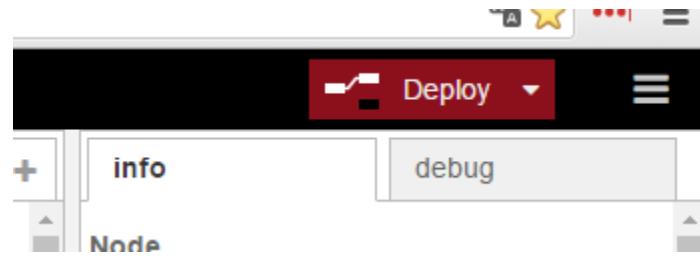
Double-click the **Button** node to edit its settings. Select the **Living Room** tab, call your button **Test** and make it print the message “**Pressed**” when it’s pressed.



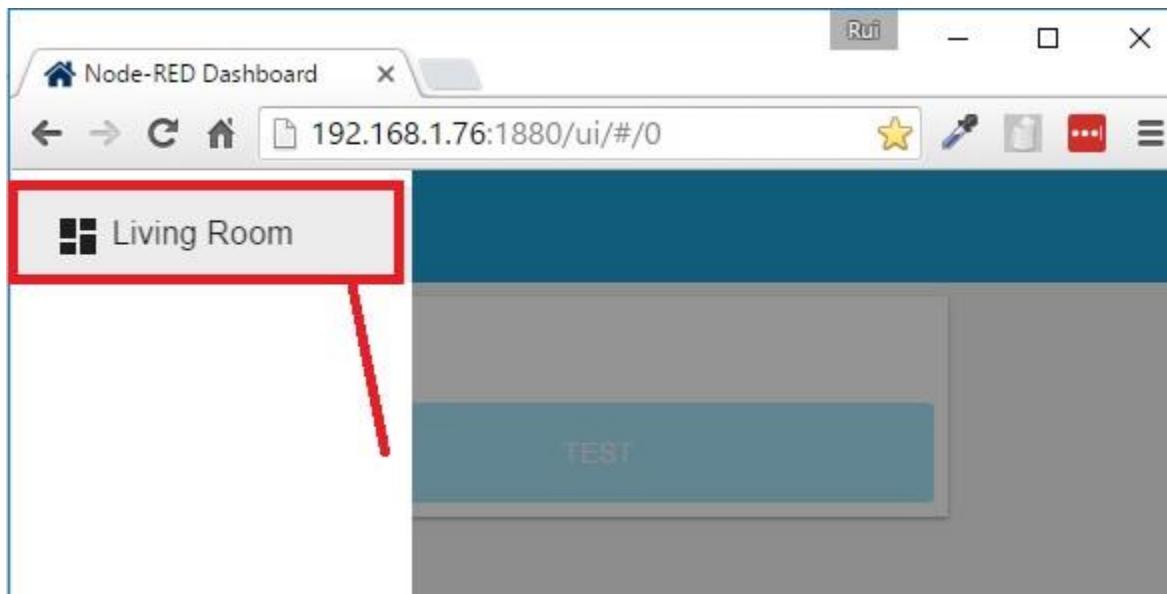
Connect a **Debug** node to your **Button** node:



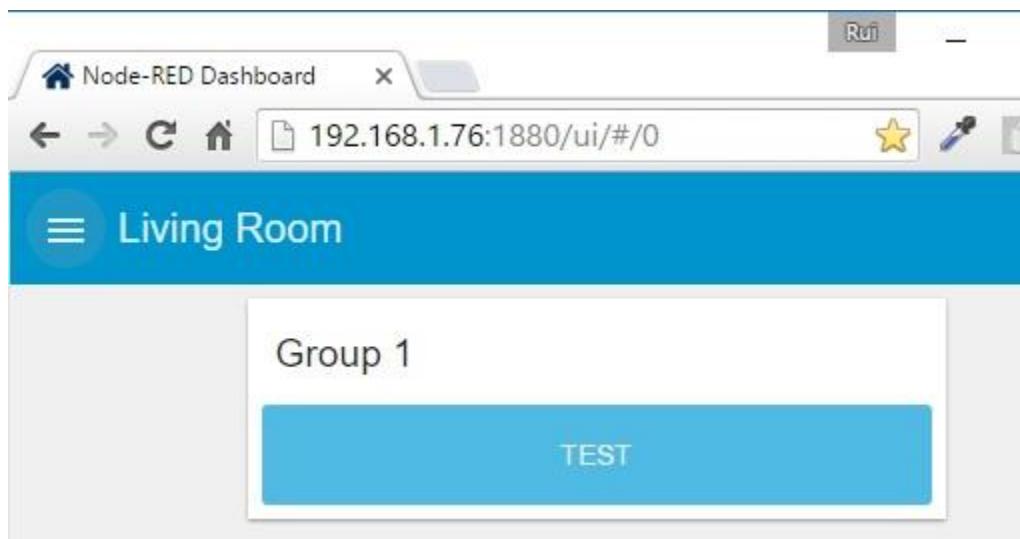
Click the **Deploy** button on the top-right corner to save your application.



Go to the dashboard page and you can see the **Living Room** tab on the left side.



Here's your **Test** button.



If you press it, it will print the message "**Pressed**" in your **debug** window.



You can use this mechanism to trigger events. You're going to learn how to do that in future Modules.

Delete those nodes and move on to the next example.

Switch and Slider Nodes

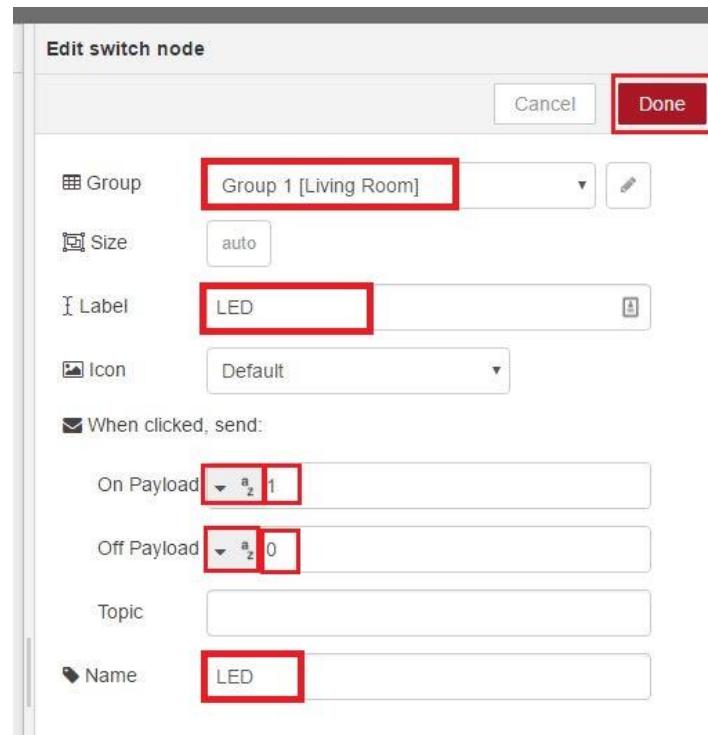
Add the **Switch** and **Slider** nodes to your flow.



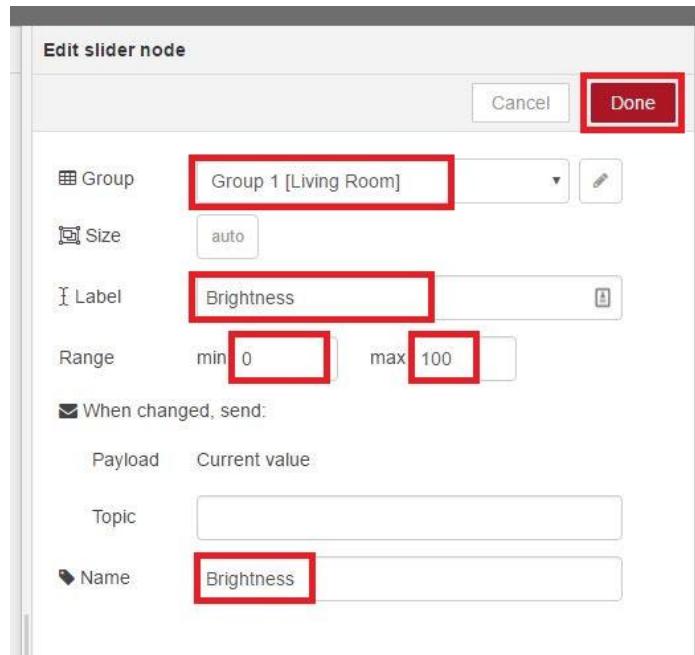
With these nodes you can control your Raspberry Pi GPIOs:

- **Switch** node: turns LED on and off
- **Slider** node: controls the LED brightness

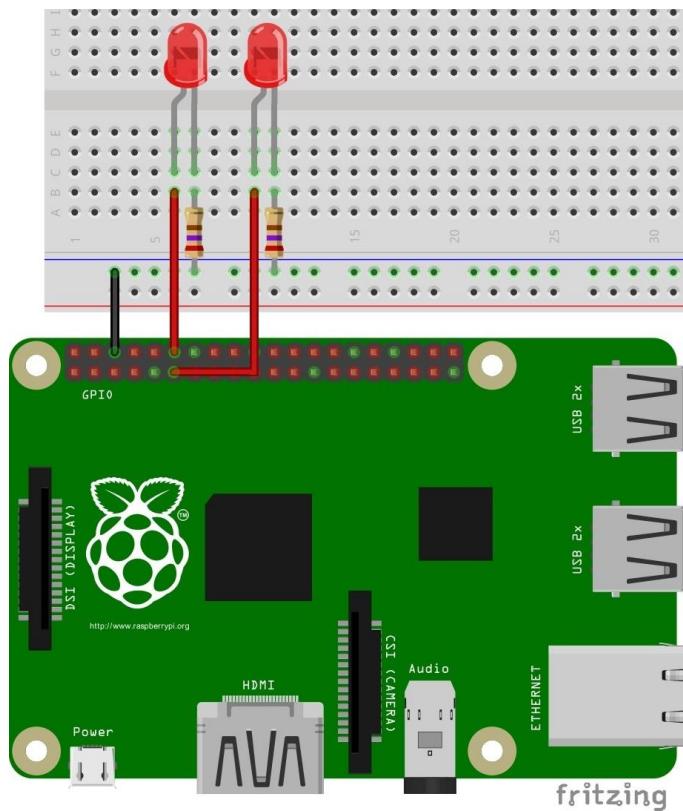
Double-click the **Switch** node and call it **LED**. Set the on value to **1** and the off value to **0**.



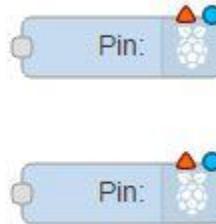
Double-click the **Slider** node and call it **Brightness**. The min should be **0** and the max should be **100**.



Follow the schematic to connect two LEDs to your Pi.



Go back to the Node-RED software and drag two **rpi-gpio out** nodes to your flow.



Assign **Pin 12** and select the type **Digital output**.

Edit rpi-gpio out node

● GPIO Pin ▾ Pi 2 Model B

Type ▾

Initialise pin state?

Name

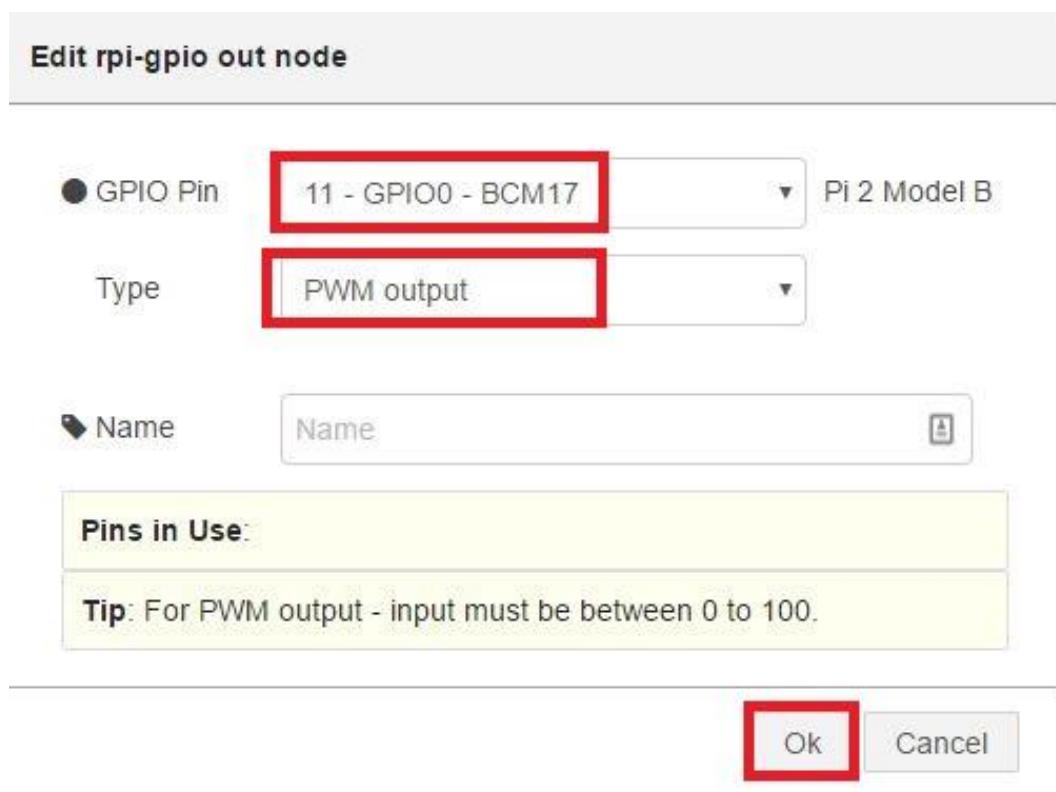
Pins in Use:

Tip: For digital output - input must be 0 or 1.

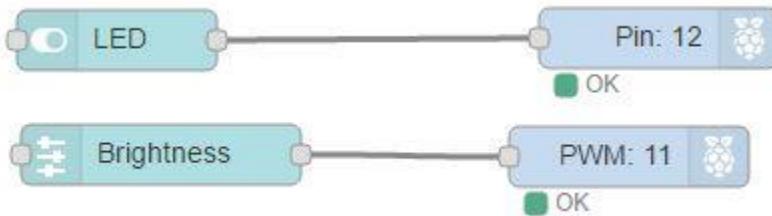
Ok **Cancel**

The "GPIO Pin" field and the "Type" field are highlighted with red boxes. The "Ok" button at the bottom right is also highlighted with a red box.

The other node assign **Pin 11** and select the type **PWM output**.

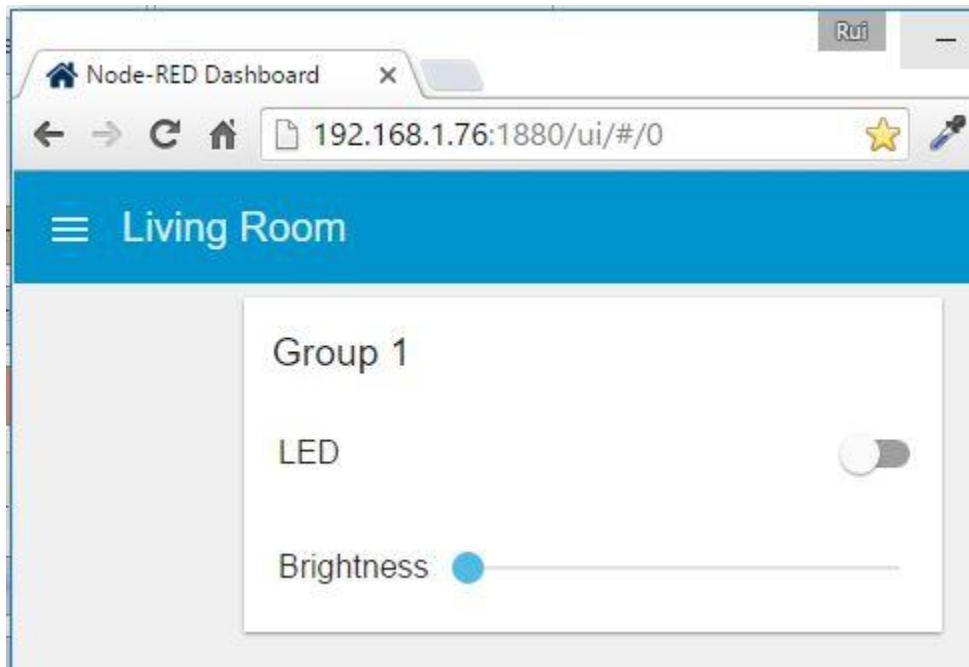


Deploy your application.

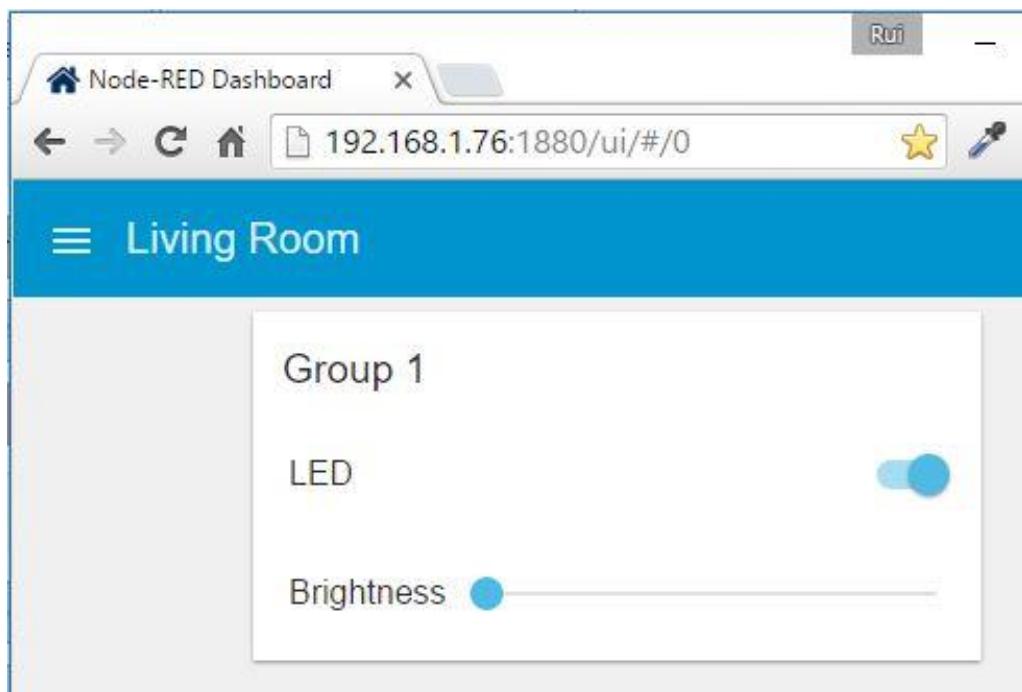


When you go to the dashboard page you can control the two LEDs.

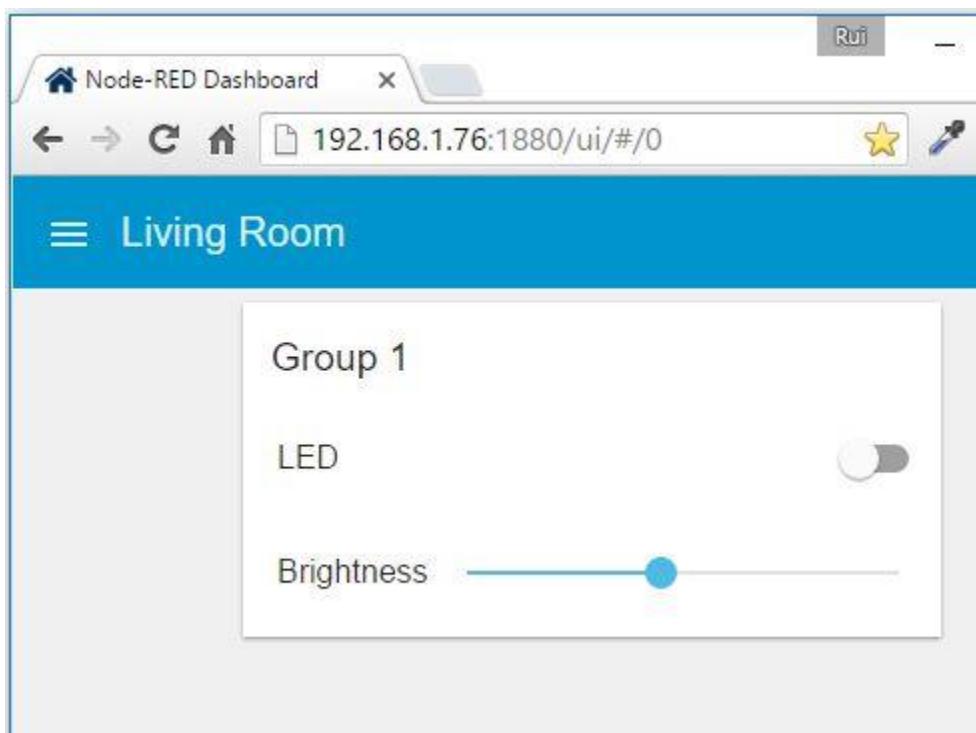
Pin 12 is currently turned off and if you press the switch:



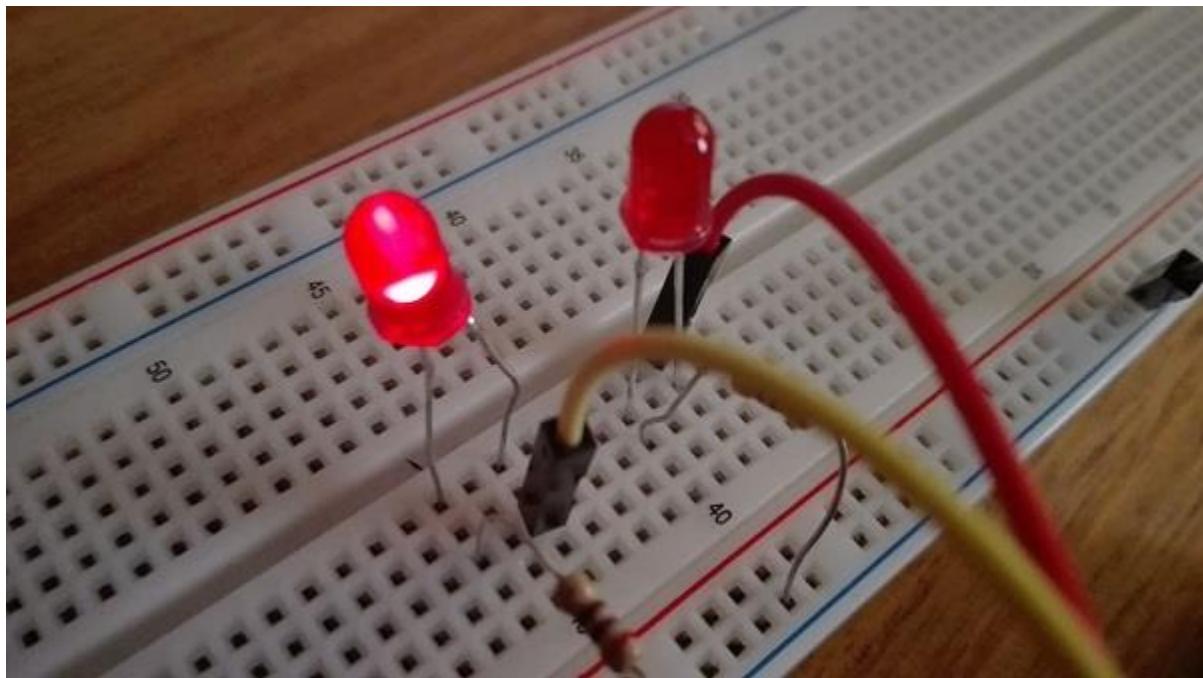
It turns your LED on.



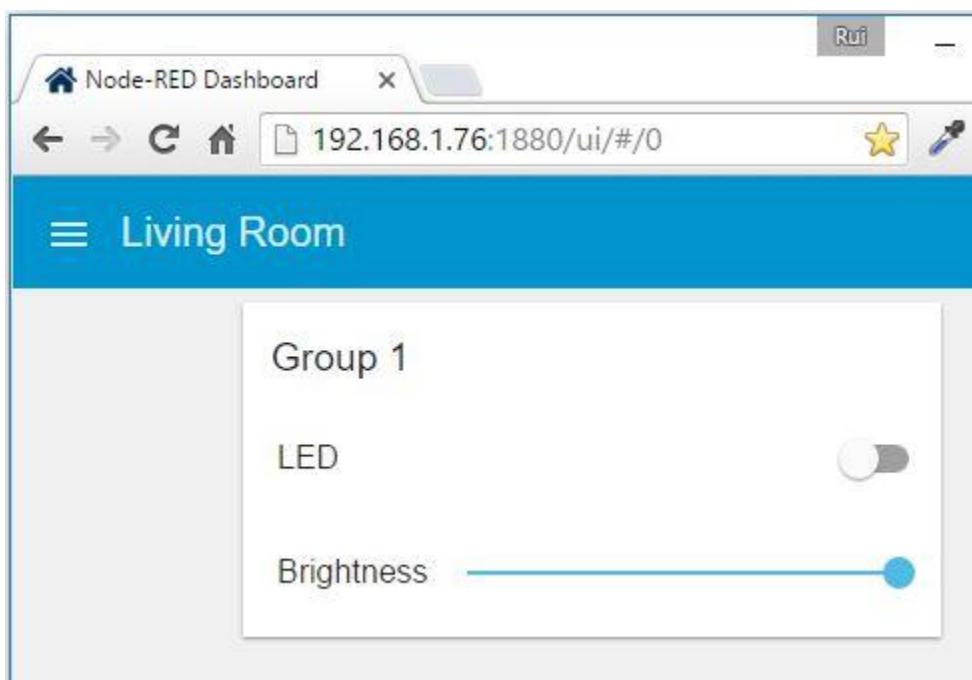
You can also control the LED brightness with the slider, if you move the slider.



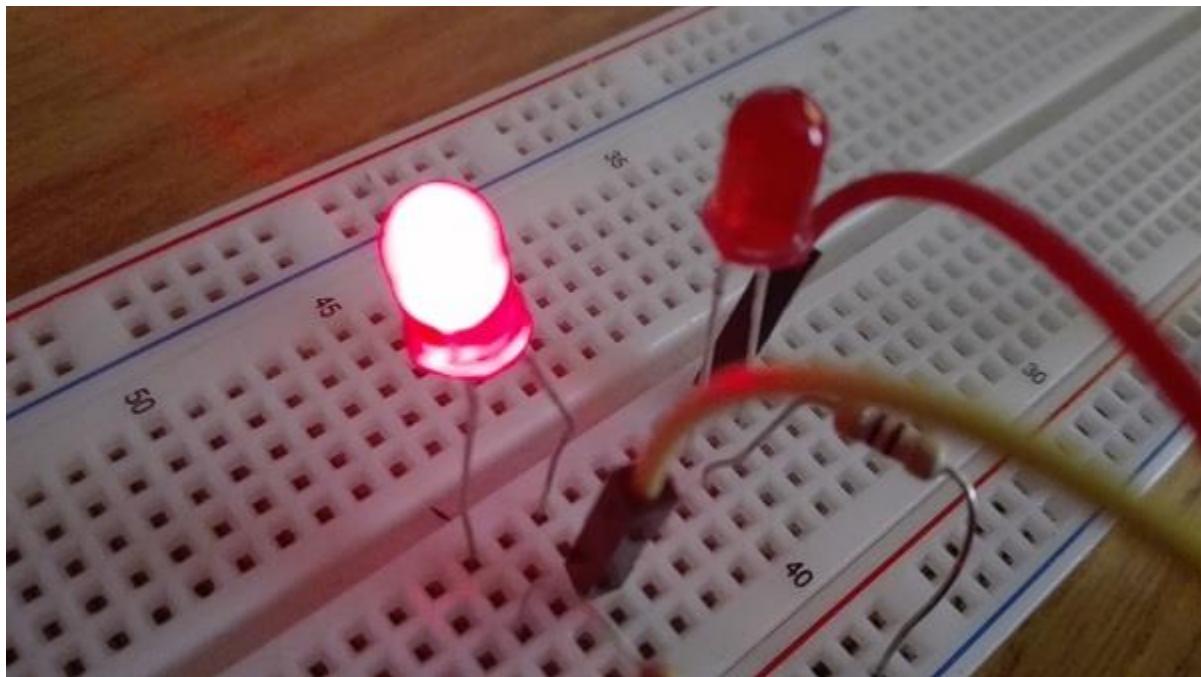
The brightness of your LED on Pin 11 changes (see previous figure):



And if you move the slider all the way up:

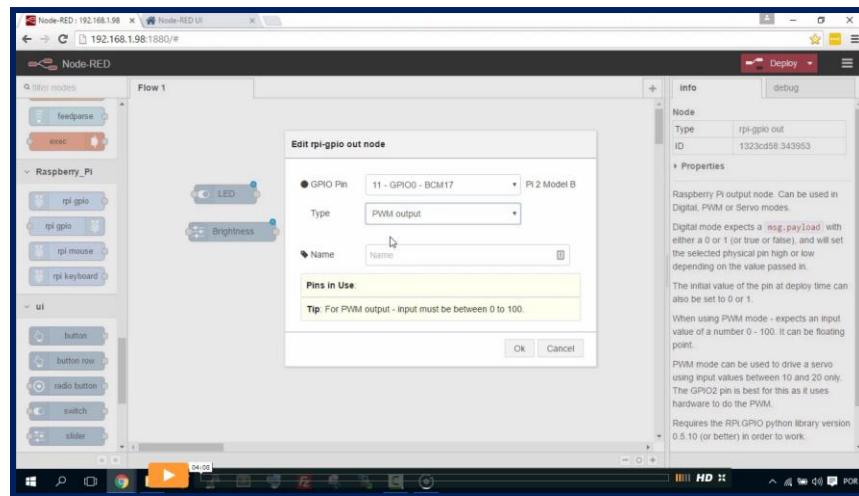


It turns your LED to its maximum brightness:



Keep in mind that you are able to access the user interface with any device that has a web browser.

Here's the video demonstration



Video # 8 - <https://rntlab.com/28hasvideos>

I encourage you to open the Node-RED Dashboard in your smartphone to control the LEDs with the switch and slider.

There are other Dashboard nodes that I didn't mention in this Unit, but you will use them in future Modules.

Feel free to experiment with the other Dashboard nodes in the meanwhile.

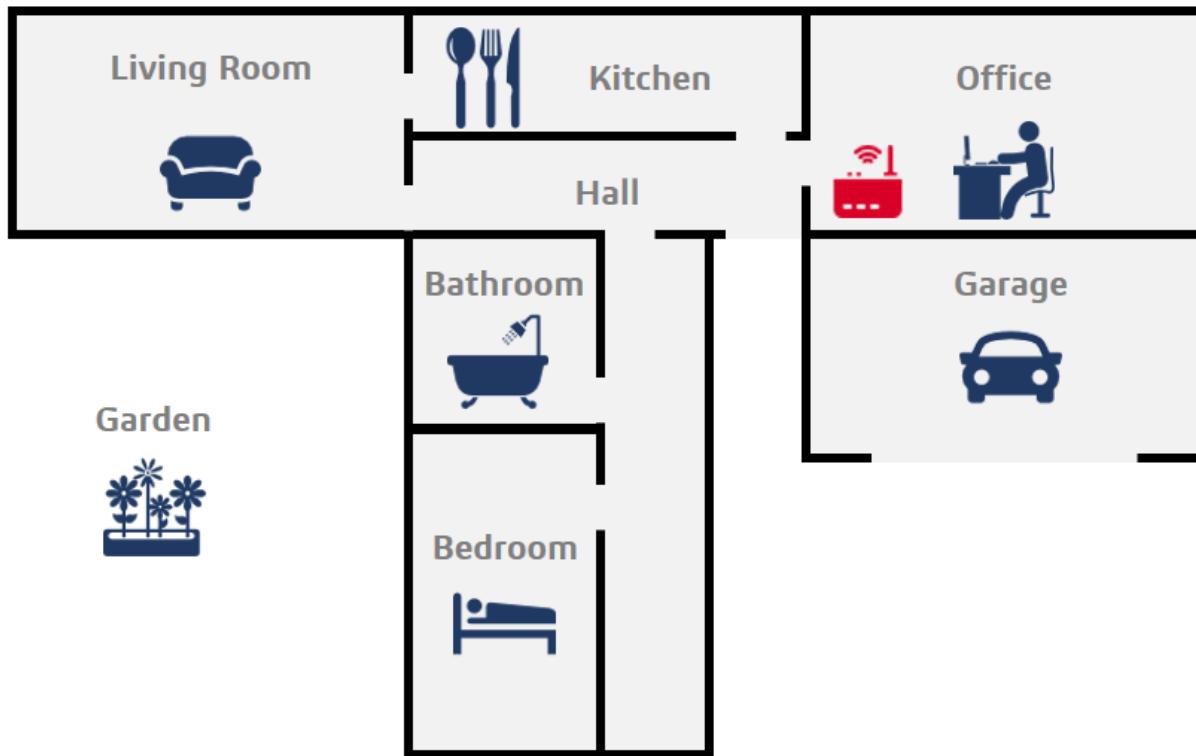
Unit 3 - Sketching Your Home Rooms

In this Unit, you will start planning which rooms of your home you would like to automate.

Rooms

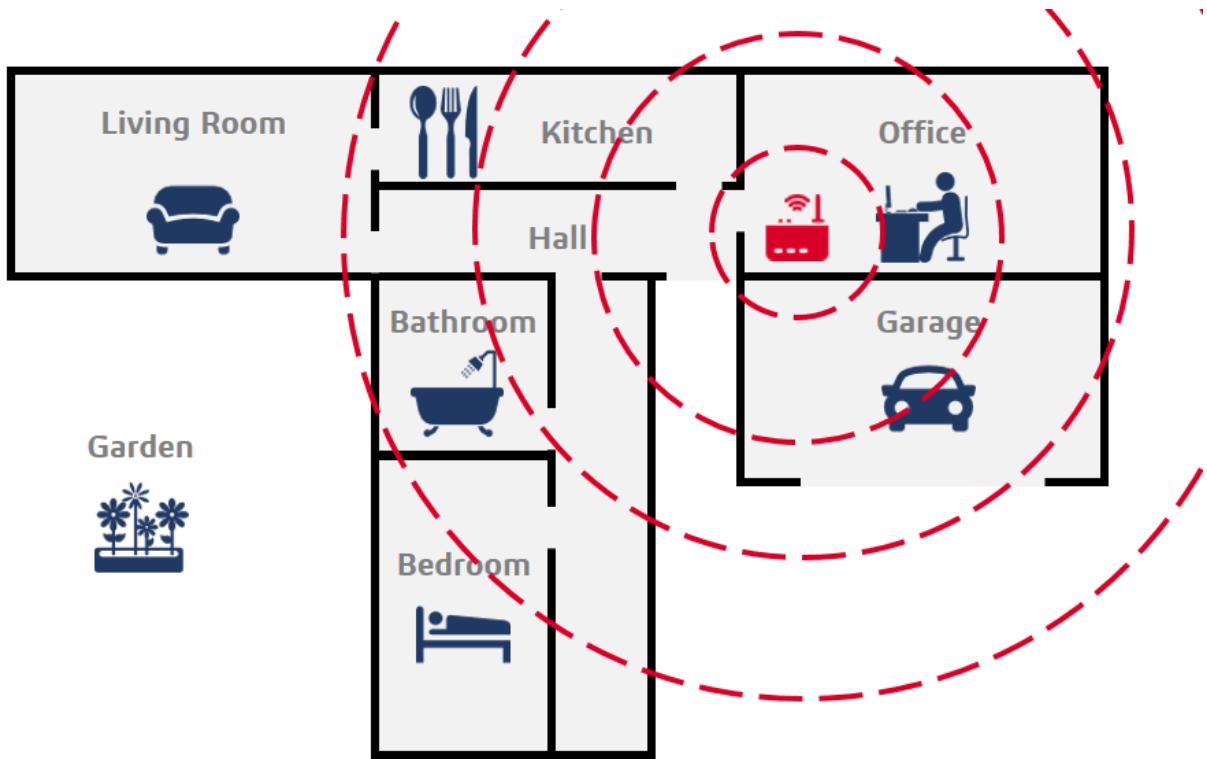
By now, you don't need to think about the final result, but it's important to at least have an idea on how your house is divided and which sections you would like to control and monitor.

Below there's a simplified sketch of my home.



You should draw something similar to represent your house and use it as a reference.

As you can see, my router is located almost in the center of my home. The location of your router is very important. You need to assure that all the devices can establish a stable wireless communication with your router.



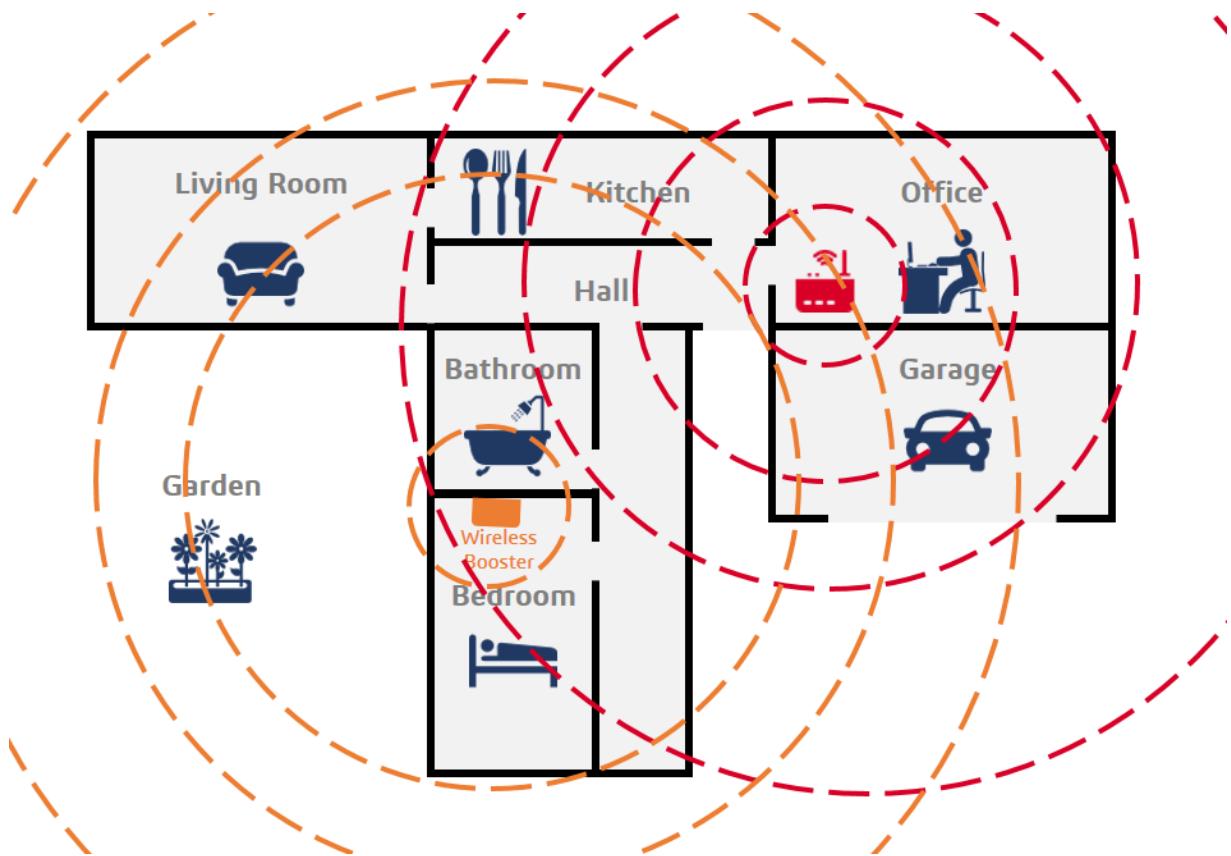
Wireless Booster

Some of the devices that you're going to add to your project may be too far away from the router and consequently won't establish a wireless communication or they will receive a weak signal....

There is a solution for this problem. You can buy a [wireless booster](#) to extend the wireless signal of your router.



For example, I need to have a [wireless booster](#) to extend the wireless signal to my bedroom and living room.



This is just something that you might want to start thinking about, but you don't have to decide or do anything about that at the moment.

Unit 4 - Creating Tabs on Node-RED Dashboard for each Room

This Unit shows how you can configure and organize the different tabs in the Dashboard. and customize the icons.

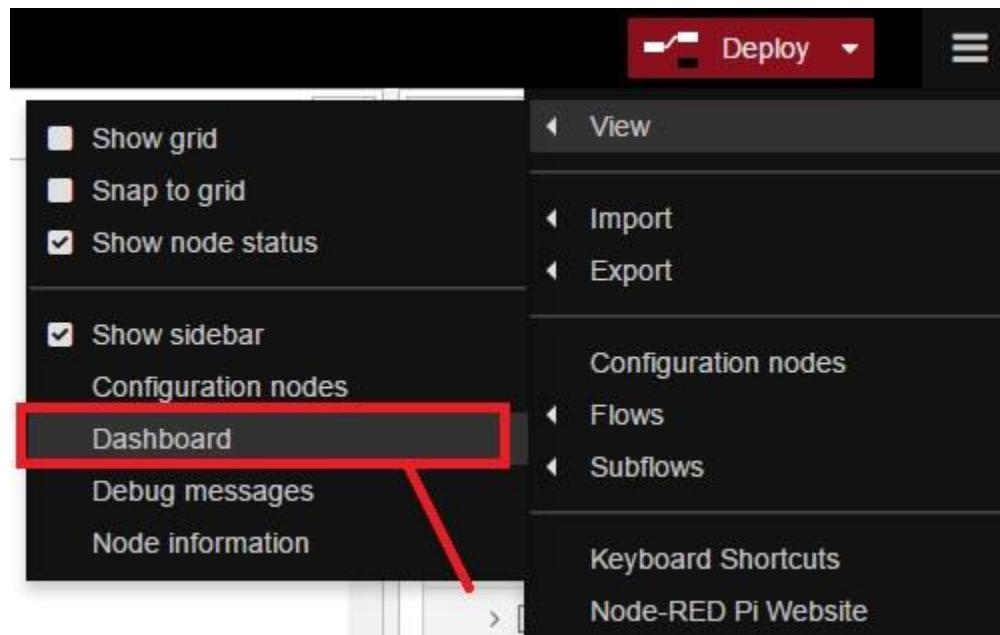
In my home I want to automate these rooms:

1. Main (This isn't a room, but it's the main tab)
2. Office
3. Living Room
4. Kitchen
5. Bedroom

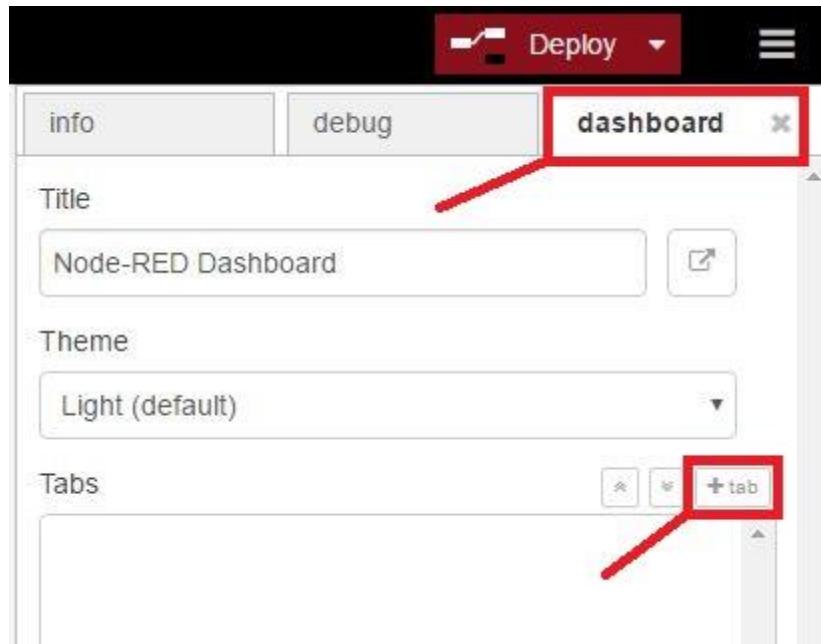
Drag 5 **Button** nodes to your flow:



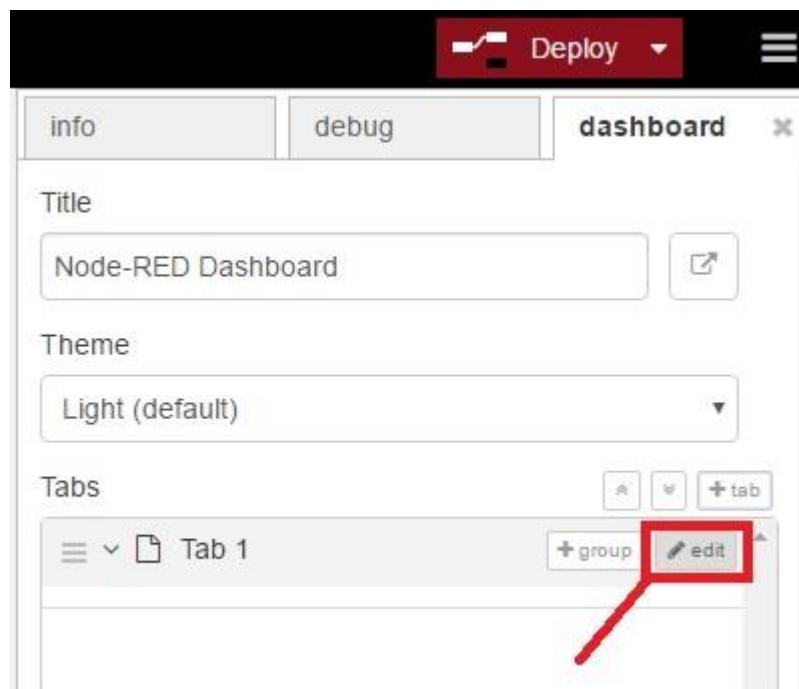
In your Dashboard, you can create tabs. Tabs are particular useful to define your rooms or the different sections in your home. Open the tab manager window. Go to the top menu > View > Dashboard.



A new window called dashboard opens in the right side. That's were you can create tabs, groups and organize the buttons of the interface. Click the +tab button.



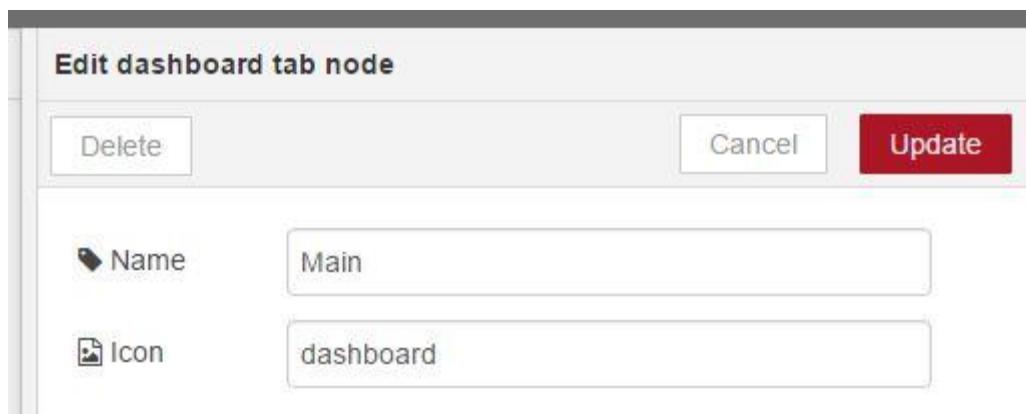
Then click the **edit** button.



In the **edit tab** window you can configure the tab name and the icon that appears in your screen:

- Name: whatever you want (it should describe your room)
- Icon: name it according to these icon names <https://klarsys.github.io/angular-material-icons>

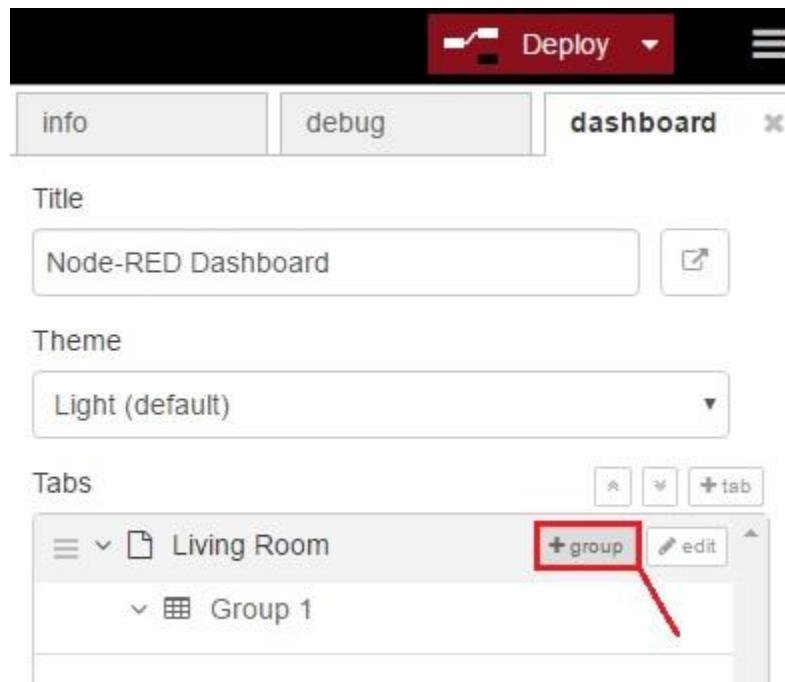
Set your tab with the name **Main** and icon **dashboard**.



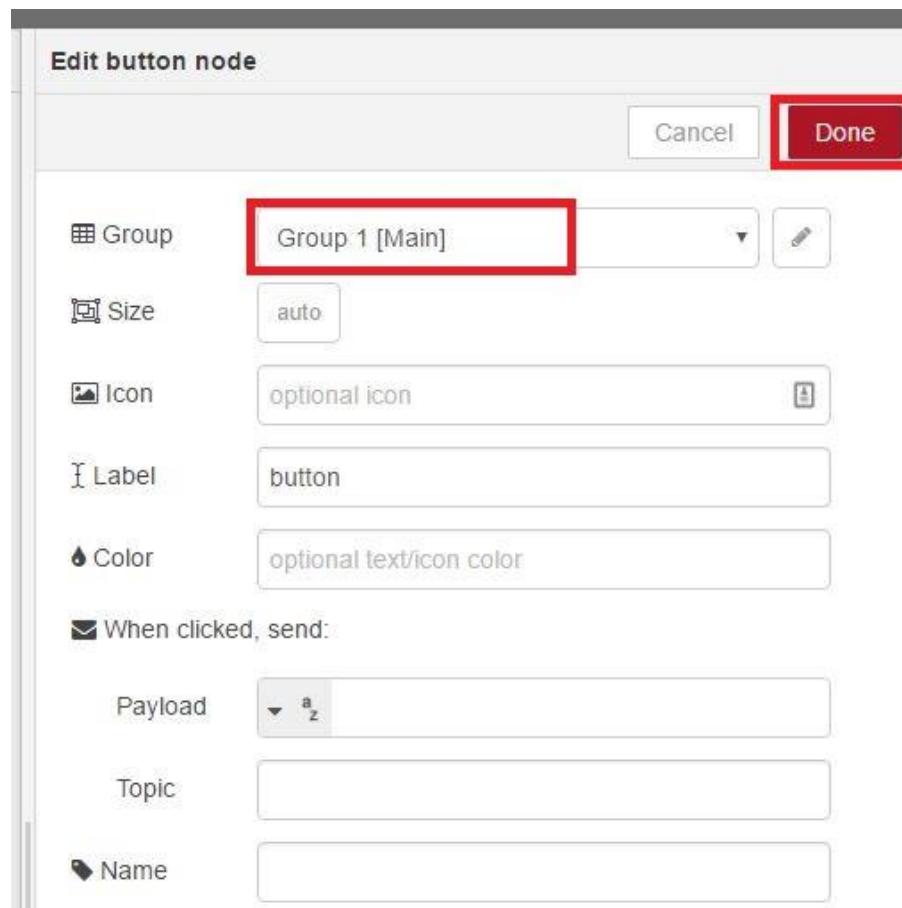
Here's the dashboard icon:



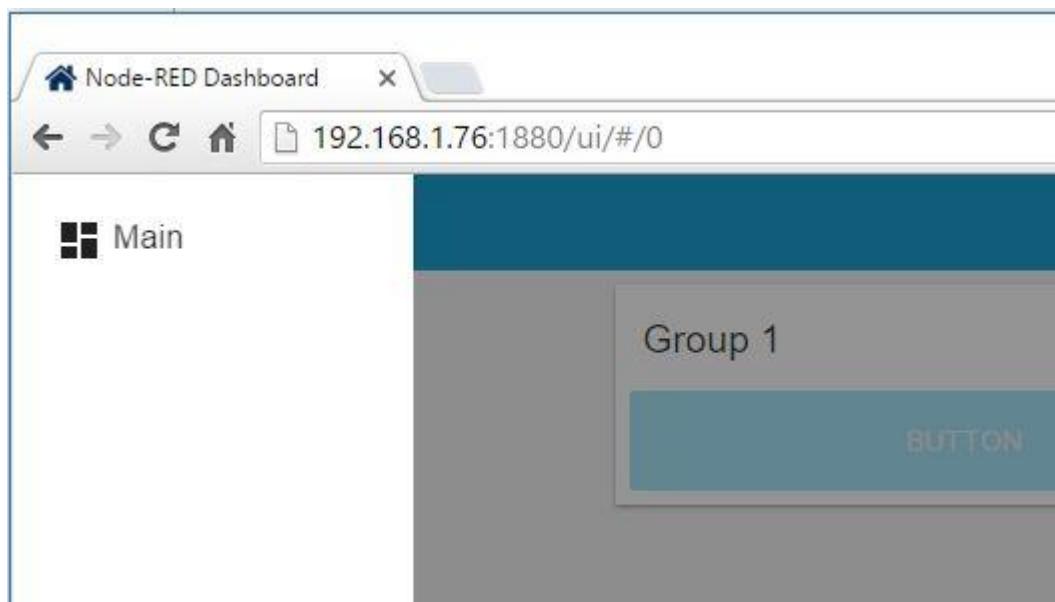
Press **+group** button.



Note: each tab should have a group assigned. Double-click the button node and select the Group 1 of the **Main** tab.



If you deploy your application and you go to the Dashboard page you can see your new tab along with its icon:



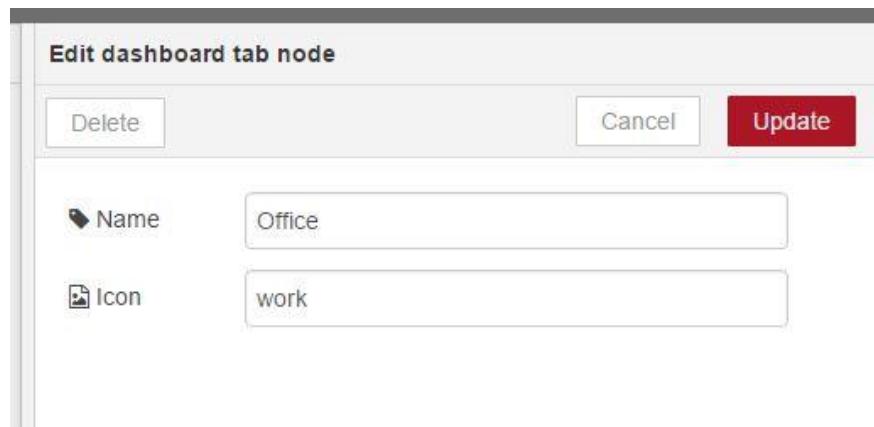
For the second tab follow these configurations:

Edit dashboard tab node

Delete Cancel Update

Name: Office

Icon: work



work icon for the **Office** tab:



Assign the button node to the **Office** tab:

Edit button node

Cancel Done

Group: Group 1 [Office]

Size: auto

Icon: optional icon

Label: button

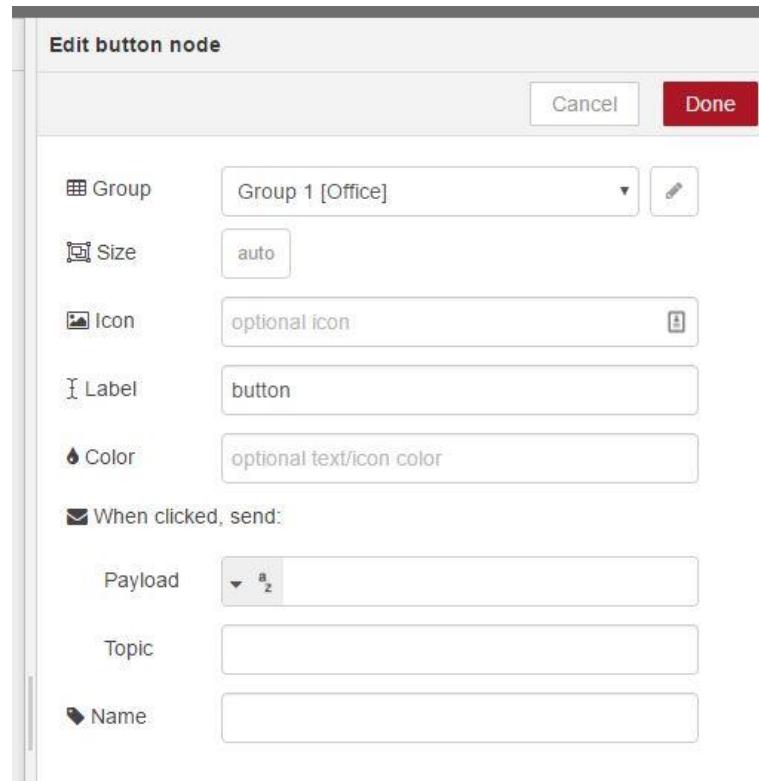
Color: optional text/icon color

When clicked, send:

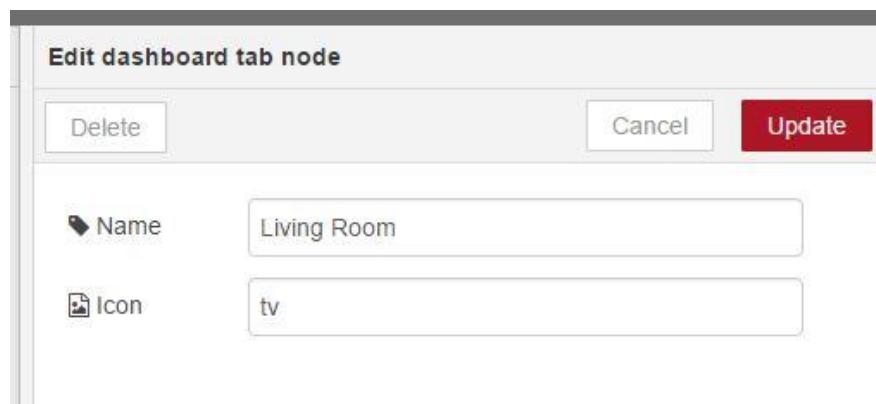
Payload: a_z

Topic:

Name:



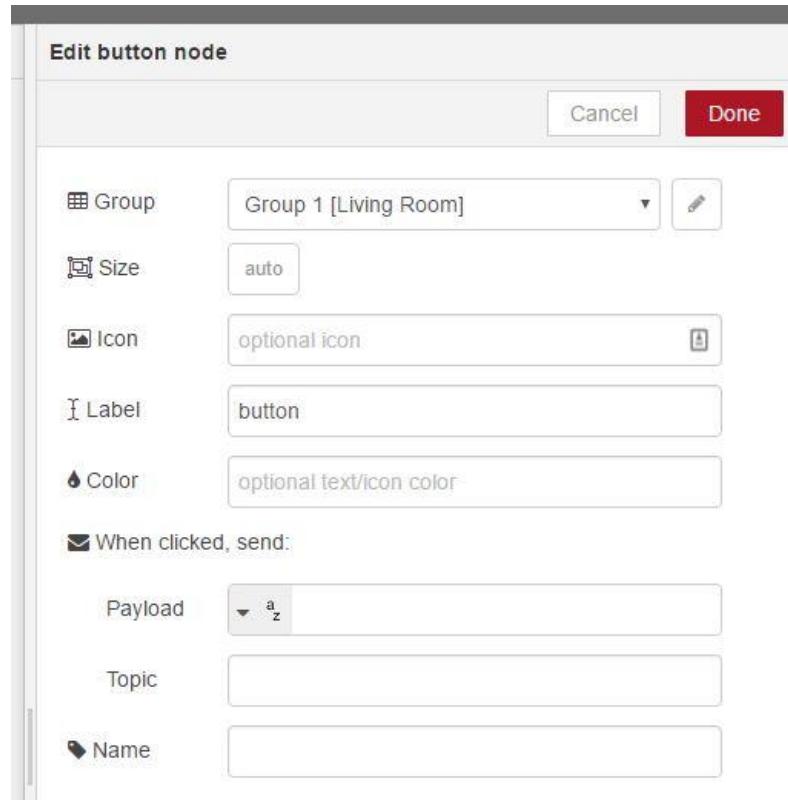
Here's the third tab:



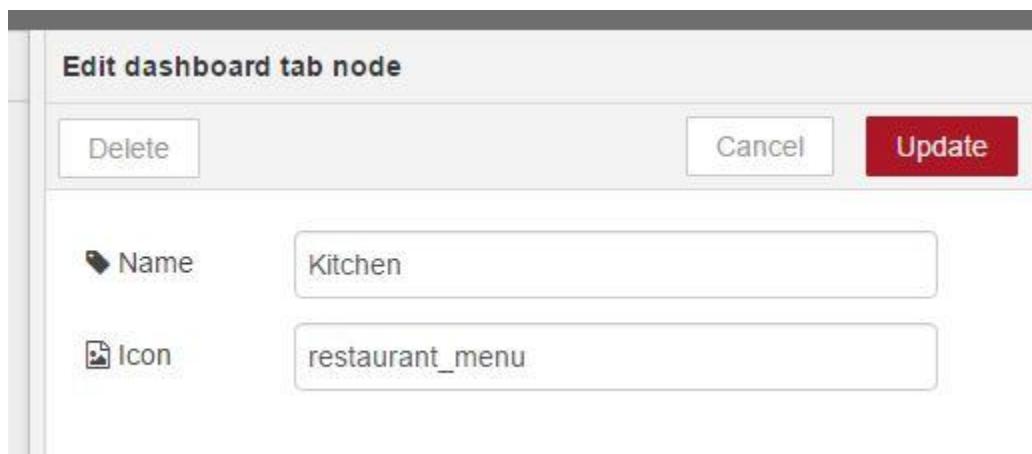
tv icon for the Living Room tab:



Assign the button node to the Living Room tab:



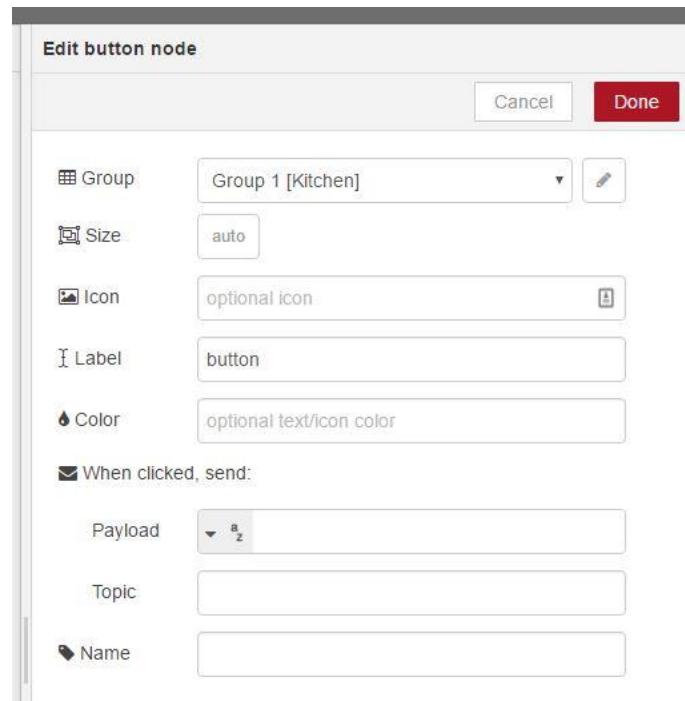
The **Kitchen** tab comes in fourth place:



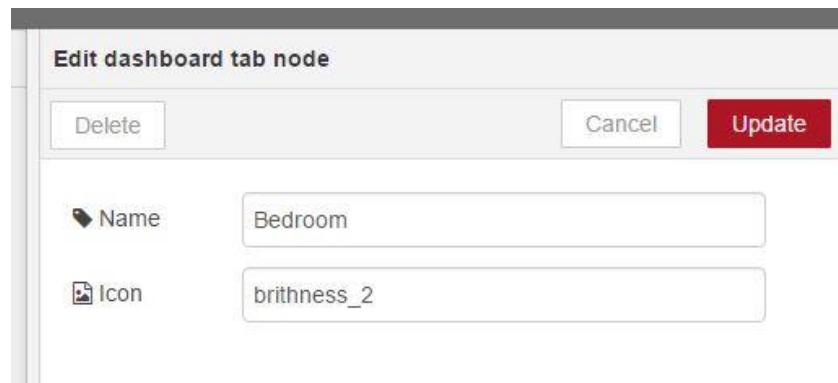
restaurant_menu for the Kitchen tab:



Assign the button node to the **Kitchen** tab:



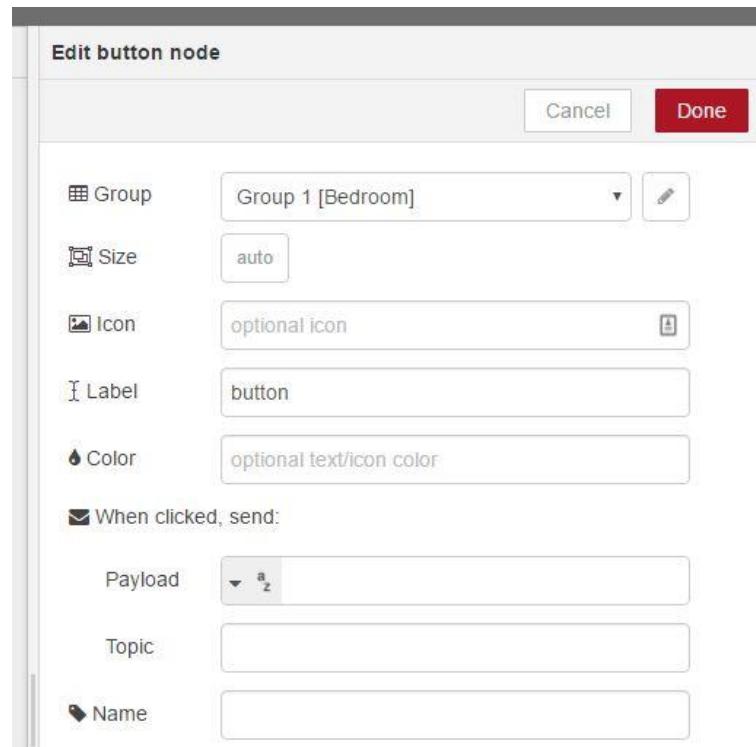
Finally, I've created the **Bedroom** tab:



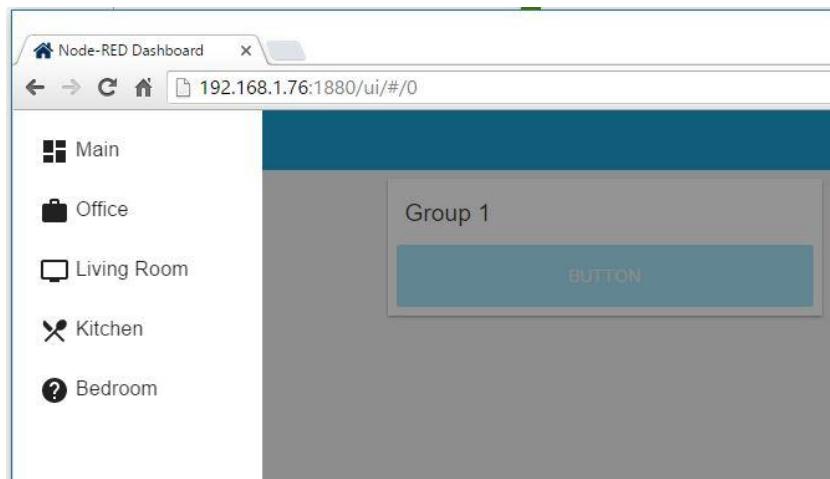
brithness_2 icon for the **Bedroom** tab:



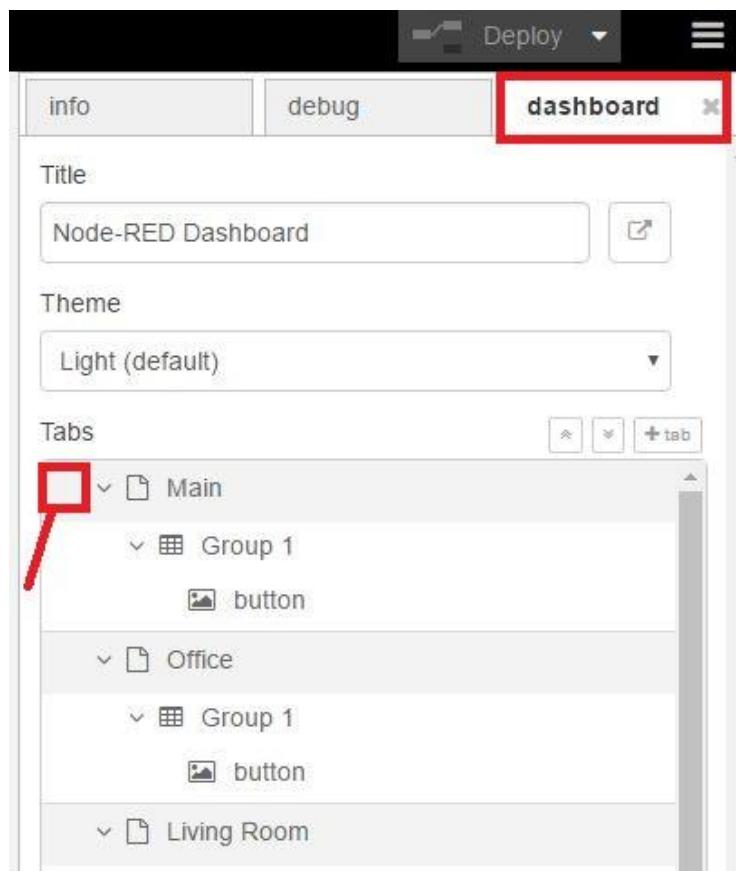
Assign the button node to the **Bedroom** tab:



If you deploy your application and you go the Dashboard page, you'll see the tabs organized:



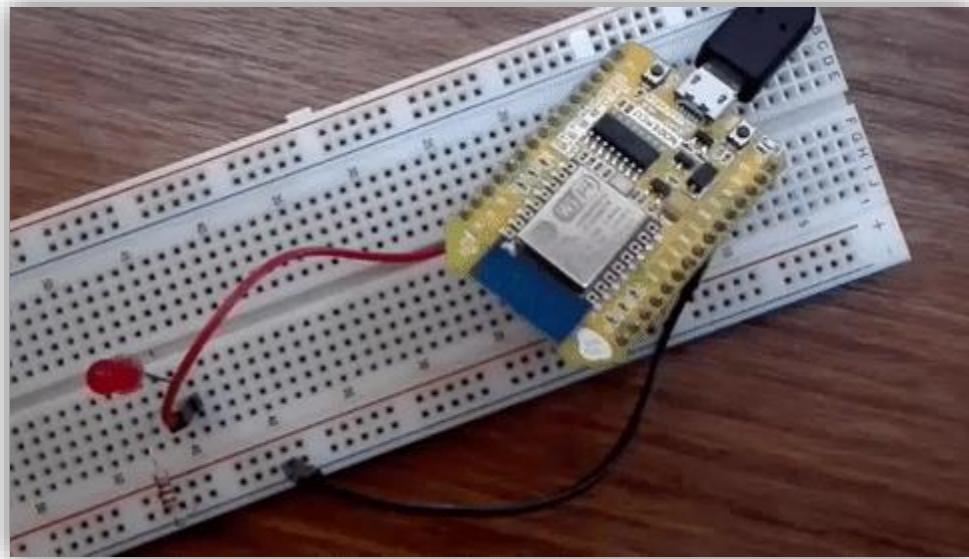
In the dashboard window within Node-RED software, you can organize the tabs with the drag-and-drop feature that is highlighted in the figure below:



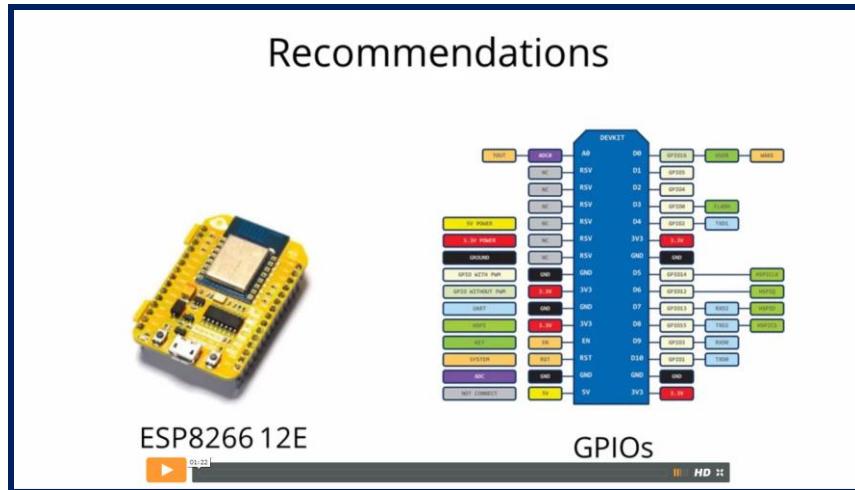
In the next couple of Modules, you'll start programming the ESP8266 and connect it to your Raspberry Pi to control outputs and monitor sensors.

Module 6

Connecting the ESP8266 - Part 1



Unit 1 - Introducing the ESP8266



Video # 9 - <https://rntlab.com/28hasvideos>

The ESP8266 is a low cost Wi-Fi module which is a great platform for any home automation project.

Comparing the ESP with other Wi-Fi modules in the market, this is definitely a great option for most “Internet of Things” projects! It’s easy to see why it’s so popular: it only costs a few dollars and can be integrated in advanced projects.

So what can you do with this low cost module?

You can create a web server, send HTTP requests, establish an MQTT communication, control outputs, read inputs and interrupts, send emails, post tweets, etc.

Here's a list of tutorials that I've already created that you might find useful:

- [Posting a Tweet with the ESP8266](#)
 - [How to Control Your ESP8266 From Anywhere in the World](#)
 - [Retrieving Bitcoin Price Using ESP8266 WiFi Module](#)
 - [ESP8266 Web Server Tutorial using NodeMCU](#)

- [Door Status Monitor using the ESP8266](#)

Let's take a look at the **ESP8266** specifications:

- 11 b/g/n protocol
- Wi-Fi Direct (P2P), soft-AP
- Integrated TCP/IP protocol stack
- Built-in low-power 32-bit CPU
- SDIO 2.0, SPI, UART

Finding Your **ESP8266**

The ESP8266 comes in a wide variety of versions (see the following figure).

Throughout this entire course we'll be using the ESP-12E NodeMCU kit.



This module is available through most electronics stores.

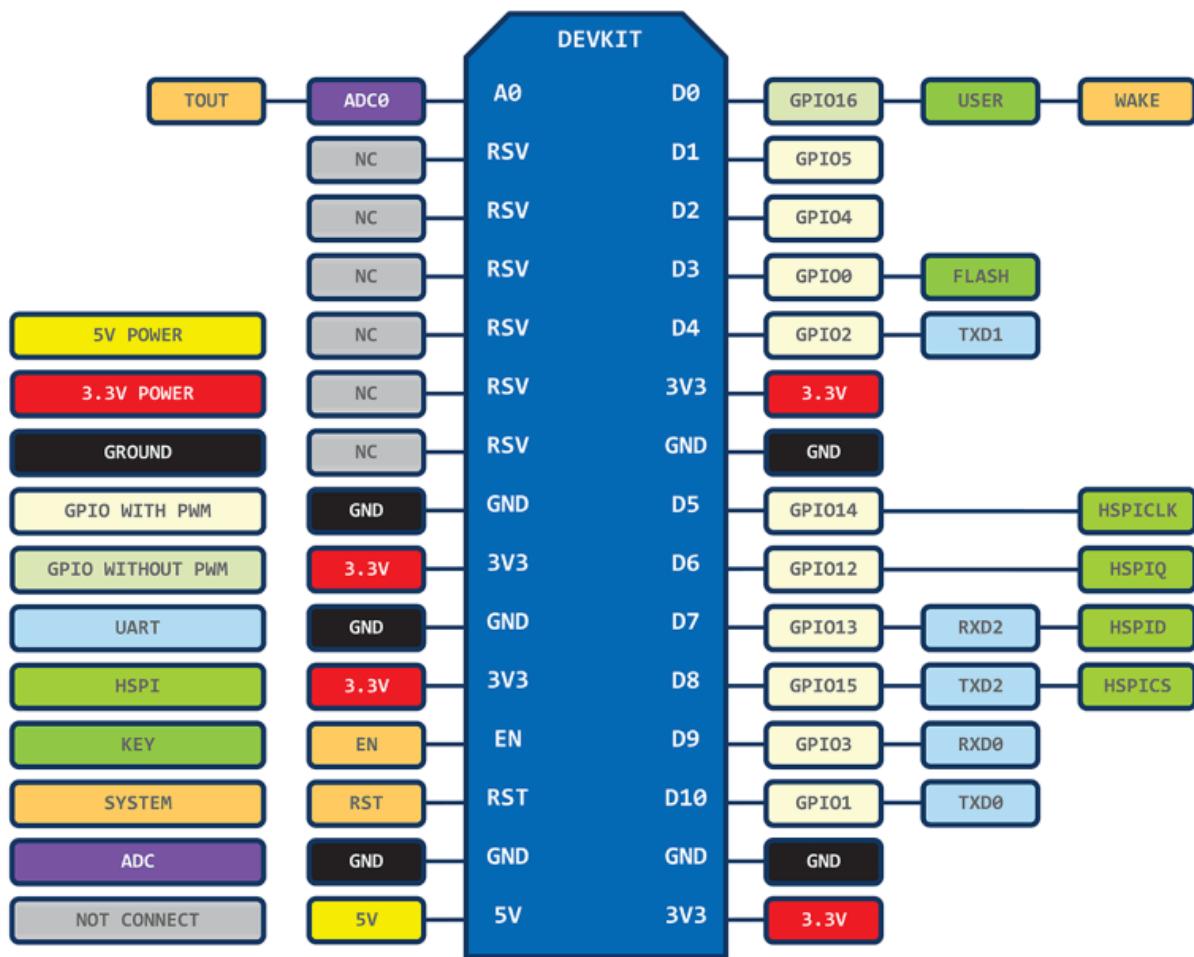
But still... The cheapest place to get one is eBay, you can purchase one ESP-12E for less than \$6. [Click here to buy this module on eBay](#).

The ESP826-12E that has built-in programmer is the best option right now. The built-in programmer makes it easy to prototype and upload your programs.



ESP-12E Pinout

The ESP-12E also comes with more GPIOs for your projects. Here's a quick overview of the pinout of your ESP-12E:



Programming the ESP8266 with Arduino IDE

There are a few ways of programming the ESP8266. In this course you'll program your ESP using the Arduino IDE software.



If you are already familiar with the ESP you probably can skip the next two Units, but it's still important that you at least take a look at them to ensure that you aren't missing anything.

Unit 2 - How to Install the ESP8266 Board in Arduino IDE

There are a variety of development environments that can be used to program the ESP8266.

The ESP8266 community created an add-on for the Arduino IDE that allows you to program the ESP8266 using the Arduino IDE and its programming language.

Downloading Arduino IDE

First, download the Arduino IDE to ensure that you have the latest software version (some older versions won't work), visit the following URL: <https://www.arduino.cc/en/Main/Software>.

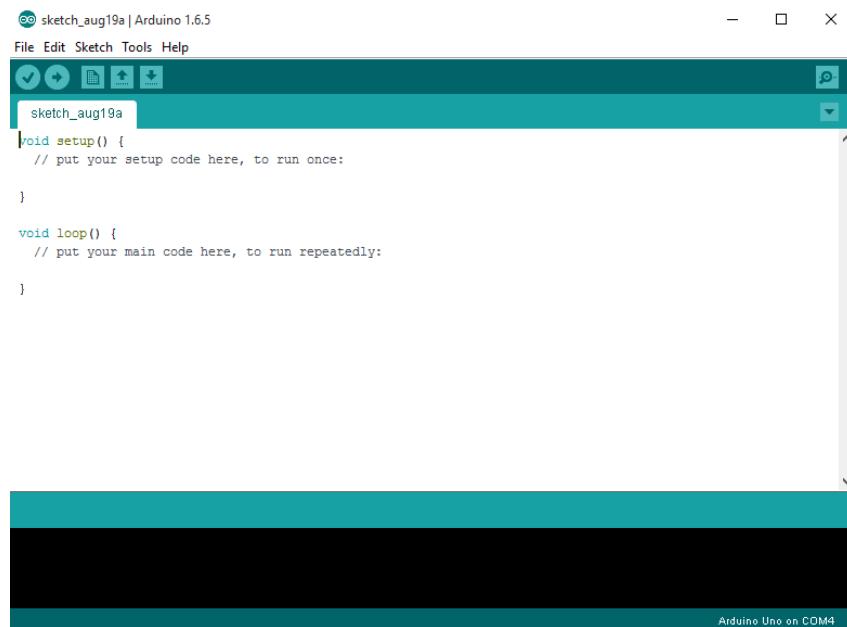
Then, select your operating system and download the latest software release of the Arduino IDE.

Installing Arduino IDE

Grab the file that you have just downloaded and open the Arduino IDE application file (see figure below).

Name	Date modified	Type
dist	13/08/2015 20:53	File folder
drivers	13/08/2015 20:53	File folder
examples	13/08/2015 20:54	File folder
hardware	13/08/2015 20:54	File folder
java	13/08/2015 20:57	File folder
lib	13/08/2015 20:59	File folder
libraries	11/09/2015 13:38	File folder
reference	13/08/2015 21:03	File folder
tools	13/08/2015 21:03	File folder
arduino	14/08/2015 17:42	Application

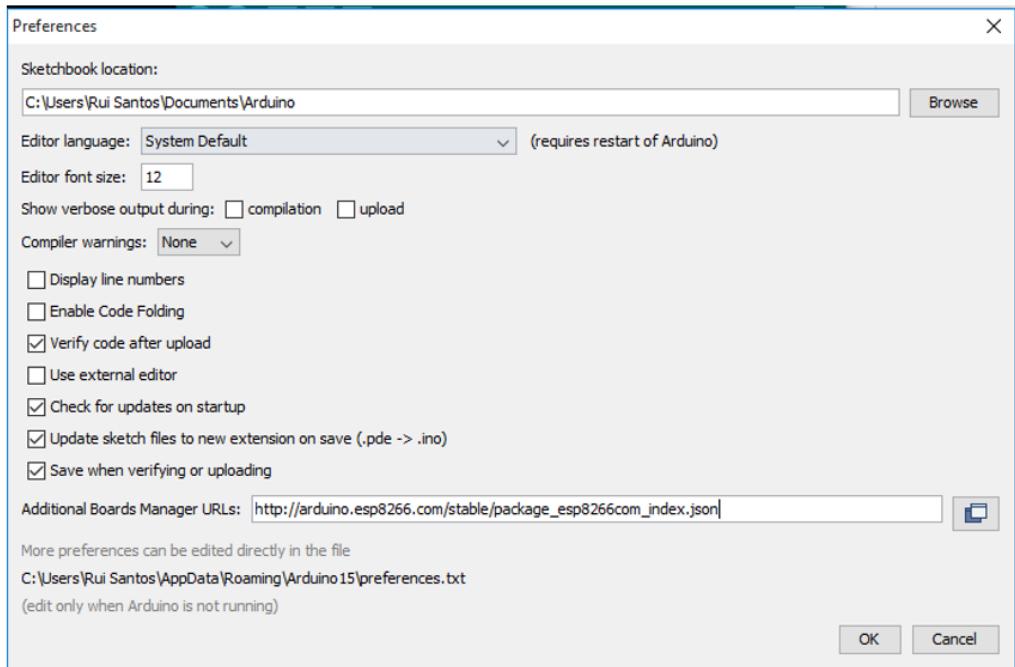
When the Arduino IDE first opens, this is what you should see:



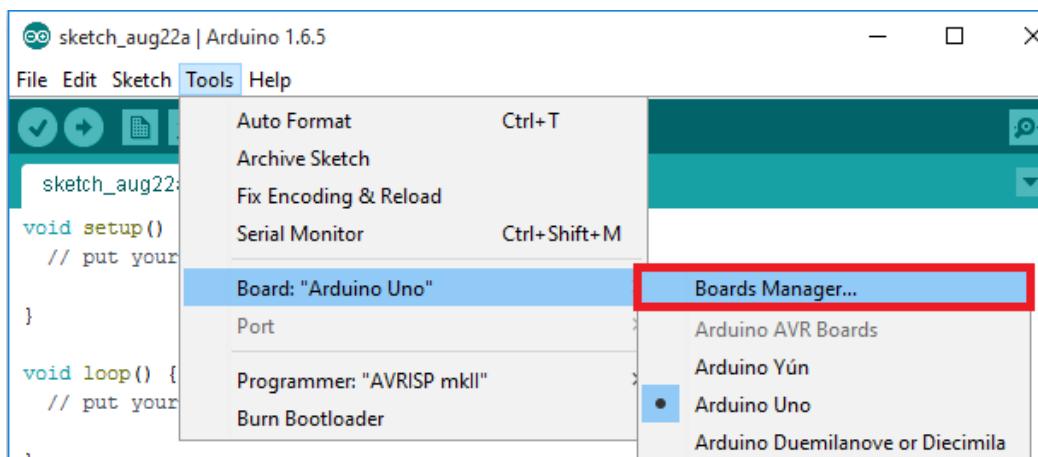
Installing the ESP8266 Board

To install the ESP8266 board in your Arduino IDE, follow these next instructions:

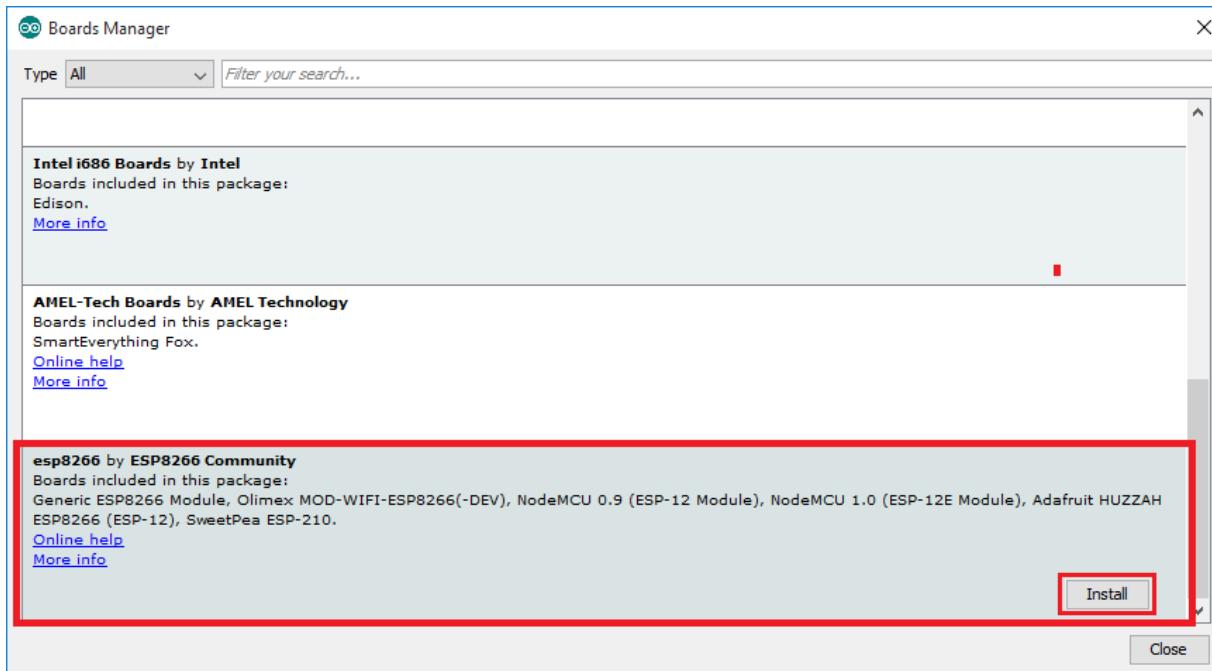
- 1) Open the preferences window from the Arduino IDE. Go to **File > Preferences**
- 2) Enter http://arduino.esp8266.com/stable/package_esp8266com_index.json into Additional Board Manager URLs field and click the “OK” button



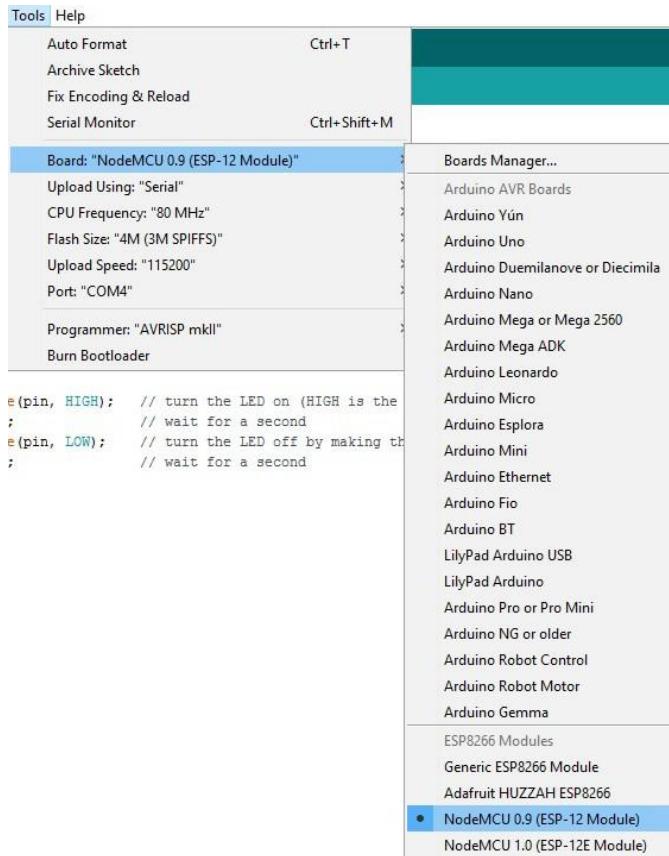
3) Open boards manager. Go to Tools > Board > Boards Manager...



4) Scroll down, select the ESP8266 board menu and install “esp8266 platform”



5) Choose your ESP8266 board from Tools > Board > NodeMCU 0.9(in my case)



6) Finally, close and re-open your Arduino IDE

Unit 3 - Testing the Installation

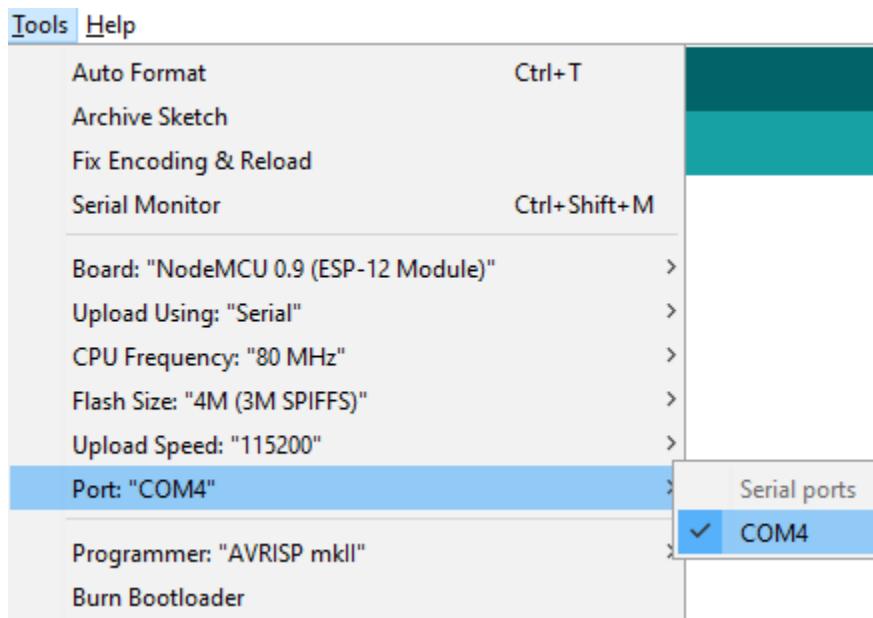
To test the ESP8266 add-on installation, let's see if we can blink an LED with the ESP8266 using the Arduino programming language.

Uploading Code

Connect a USB cable from your ESP to your computer to upload code.

Having your Arduino IDE open and the right board selected follow these next instructions:

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



- 2) Copy and paste the sketch below to your Arduino IDE

```
*****
Rui Santos
Complete project details at http://randomnerdtutorials.com
*****/
```

```

int pin = 2;

void setup() {
    // initialize GPIO 2 as an output.
    pinMode(pin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(pin, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000);           // wait for a second
    digitalWrite(pin, LOW); // turn the LED off by making the voltage LOW
    delay(1000);           // wait for a second
}

```

DOWNLOAD SOURCE CODE

https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/blink_led_esp8266.ino

- 3) Click the “Upload” button. You should see “Done Uploading” after a few seconds

```

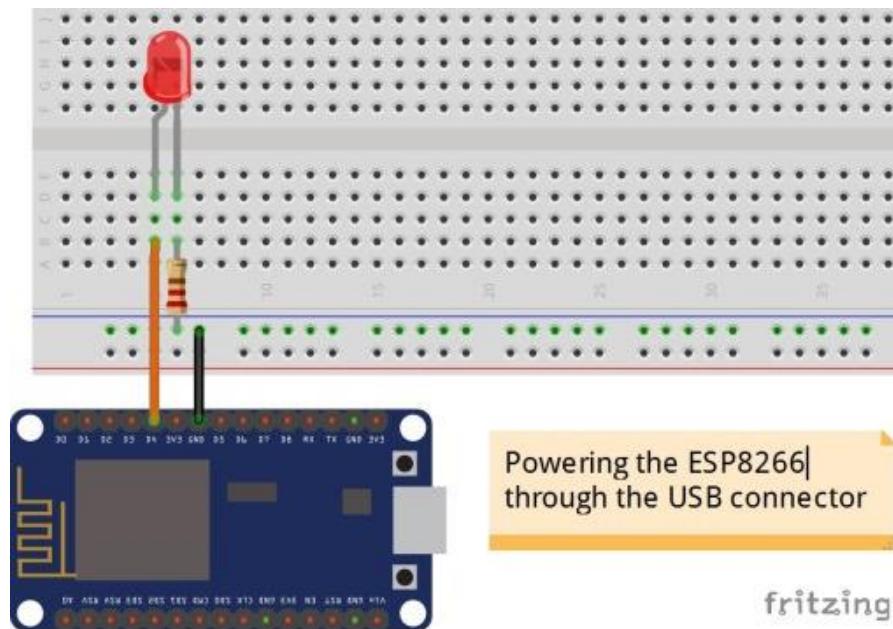
Done uploading.

Global variables use 33,050 bytes (40%) of dynamic memory, leaving 48,870 bytes for local variables. Maximum is 81,920 bytes.
Uploading 202992 bytes from C:\Users\RIU SAN-1\AppData\Local\Temp\builid201033229013242365.tmp\esp8266_blink.cpp.bin to flash at 0x00000000
.....
19

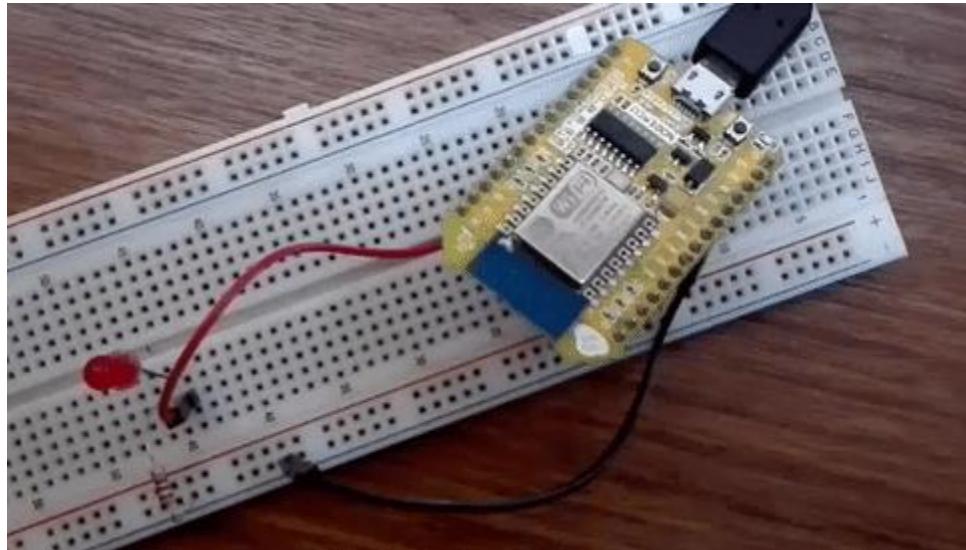
```

Demonstration

Connect an LED and a 220 Ohm resistor to your ESP D4 (GPIO 2).



Your LED should be blinking every 1 second.

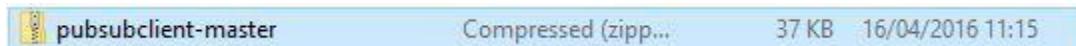


Unit 4 - Installing the PubSubClient Library

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

Installing the Library

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



- 2) Unzip the .zip folder and you should get pubsubclient-master folder



- 3) Rename your folder from **pubsubclient-master** to **pubsubclient**



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

Name	Date modified	Type	Size
Bridge	08/03/2016 17:07	File folder	
Esplora	20/05/2015 17:10	File folder	
Ethernet	09/03/2016 16:11	File folder	
Firmata	16/02/2016 01:32	File folder	
GSM	09/03/2016 16:11	File folder	
Keyboard	08/03/2016 17:05	File folder	
LiquidCrystal	09/03/2016 16:11	File folder	
Mouse	08/03/2016 17:04	File folder	
pubsubclient	16/04/2016 11:16	File folder	
Robot_Control	10/06/2015 15:00	File folder	
Robot_Motor	10/06/2015 15:00	File folder	
RobotIRremote	10/06/2015 15:00	File folder	
SD	09/03/2016 16:11	File folder	
Servo	09/03/2016 16:11	File folder	
SpacebrewYun	27/03/2015 15:01	File folder	
Stepper	09/03/2016 16:11	File folder	
Temboo	08/03/2016 15:58	File folder	
TFT	09/03/2016 16:11	File folder	
WiFi	09/03/2016 16:11	File folder	

5) Then, re-open your Arduino IDE

The library comes with a number of example sketches.

See **File >Examples > PubSubClient** within the Arduino IDE software.

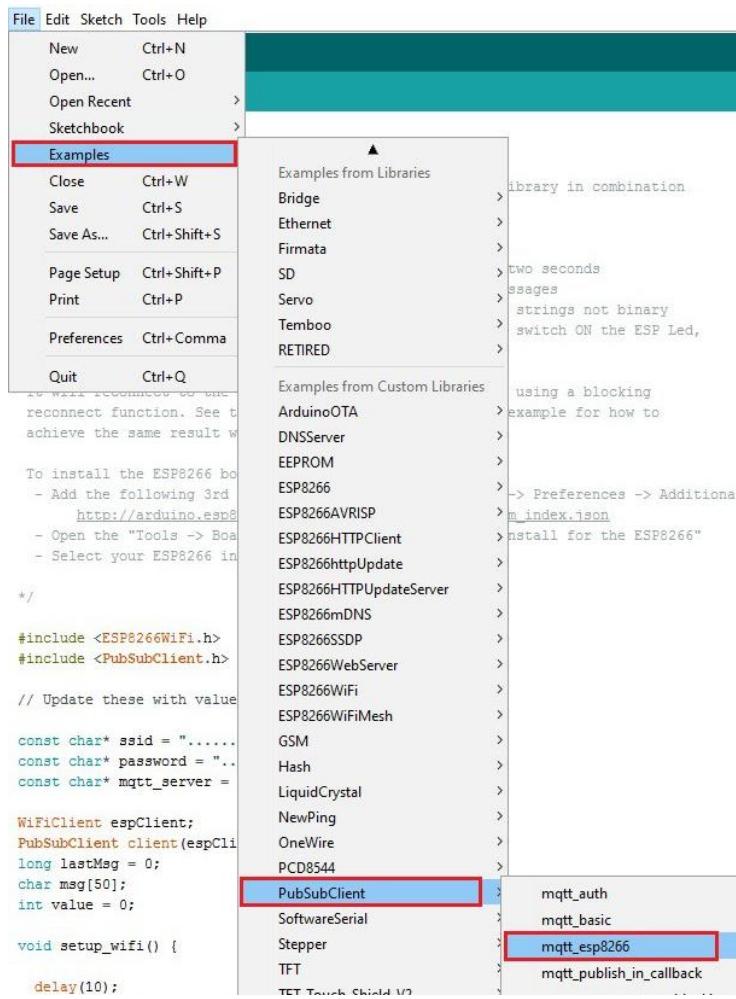
Unit 5 - Connecting the ESP8266 to the Node-RED Nodes

This Unit shows how you can publish messages and subscribe to a topic with MQTT using an ESP8266 and Node-RED.

These are the basic concepts that will allow you to turn on lights and monitor sensors (which we are going to cover in the next Module).

Writing Your ESP Sketch

Open the Arduino IDE. Go to **File > Examples > PubSubClient > mqtt_esp8266**.



A new sketch opens. That sketch publishes messages and subscribes to a topic with MQTT.

Understanding How the Sketch Works

First, it starts by loading the **ESP8266WiFi** library and the **PubSubClient** library.

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

Then, it configures the module with your own credentials (SSID, password and MQTT broker IP address).

You actually need to replace those 3 variables with your credentials, so that your ESP8266 can connect to your network and MQTT broker.

```
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
const char* mqtt_server = "YOUR_RPi_IP_Address";
```

It initializes the `espClient` and creates three variables.

```
WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;
```

setup

The `setup()` function sets the ESP8266 built-in LED as an OUTPUT and starts the serial communication at a baud rate of 115200.

The `setup()` also:

- calls the `setup_wifi()` function
- connects your ESP to the MQTT broker
- sets the `callback()` function

```
void setup() {
    pinMode(BUILTIN_LED, OUTPUT);      // Initialize the BUILTIN_LED pin as an output
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
}
```

setup wifi

The `setup_wifi()` function connects your ESP to your home router with the credentials that you have provided.

If the Wi-Fi connection is successful, your ESP IP address will be printed in the Arduino IDE serial monitor.

```
void setup_wifi() {

    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

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

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

loop

The `loop()` function checks if your ESP is connected to the MQTT broker. If your ESP isn't connected, it will try to connect/reconnect.

Then, we have a timer that every 2 seconds publishes the message saying "hello world" to a topic called `outTopic`.

```
void loop() {

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

    long now = millis();
    if (now - lastMsg > 2000) {
        lastMsg = now;
        ++value;
        sprintf (msg, 75, "hello world #%ld", value);
        Serial.print("Publish message: ");
        Serial.println(msg);
        client.publish("outTopic", msg);
    }
}
```

reconnect

If the ESP8266 loses connection with the MQTT broker, it calls the `reconnect()` function.

If for some reason your ESP can't establish an MQTT communication with your broker, your ESP keeps trying to reconnect every 5 seconds.

When the connection is finally established, it does the following:

- Publishes a message to the `outTopic` topic
- Subscribes to the `inTopic` topic

```

    void reconnect() {
        // Loop until we're reconnected
        while (!client.connected()) {
            Serial.print("Attempting MQTT connection...");
            // Attempt to connect
            if (client.connect("ESP8266Client")) {
                Serial.println("connected");
                // Once connected, publish an announcement...
                client.publish("outTopic", "hello world");
                // ... and resubscribe
                client.subscribe("inTopic");
            } else {
                Serial.print("failed, rc=");
                Serial.print(client.state());
                Serial.println(" try again in 5 seconds");
                // Wait 5 seconds before retrying
                delay(5000);
            }
        }
    }
}

```

callback

The `callback()` function is triggered every time that another device publishes a message to your ESP (in our example it will be the Raspberry Pi).

In this sketch the messages received will be printed in your serial monitor. Later, you are going to use this mechanism to turn on and off your lights.

```

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

    // Switch on the LED if an 1 was received as first character
    if ((char)payload[0] == '1') {
        digitalWrite(BUILTIN_LED, LOW);    // Turn the LED on (Note that LOW is the voltage level
        // but actually the LED is on; this is because
        // it is active low on the ESP-01)
    } else {
        digitalWrite(BUILTIN_LED, HIGH);   // Turn the LED off by making the voltage HIGH
    }
}

```

Uploading the Sketch

Finally, you can upload the full sketch to your ESP8266 (replace with your SSID, password and RPi IP address):

```
/*
Basic ESP8266 MQTT example
Forked from http://bit.ly/1qH7Gsf
*/

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

// Update these with values suitable for your network.

const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
const char* mqtt_server = "YOUR_RPi_IP_Address";

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;

void setup_wifi() {

    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

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

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

```

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

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

// Switch on the LED if an 1 was received as first character
if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW); // Turn the LED on (Note that LOW is the voltage level
    // but actually the LED is on; this is because
    // it is active low on the ESP-01)
} else {
    digitalWrite(BUILTIN_LED, HIGH); // Turn the LED off by making the voltage HIGH
}

}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection... ");
        // Attempt to connect
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            // Once connected, publish an announcement...
            client.publish("outTopic", "hello world");
            // ... and resubscribe
            client.subscribe("inTopic");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

```

```

}

}

void setup() {
    pinMode(BUILTIN_LED, OUTPUT); // Initialize the BUILTIN_LED pin as an output
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
}

void loop() {

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

    long now = millis();
    if (now - lastMsg > 2000) {
        lastMsg = now;
        ++value;
        sprintf(msg, 75, "hello world #%ld", value);
        Serial.print("Publish message: ");
        Serial.println(msg);
        client.publish("outTopic", msg);
    }
}

```

DOWNLOAD SOURCE CODE

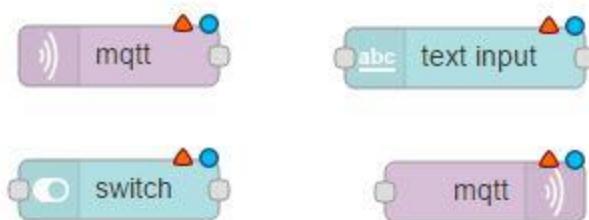
https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/mqtt_esp8266.ino

Creating Your Flow

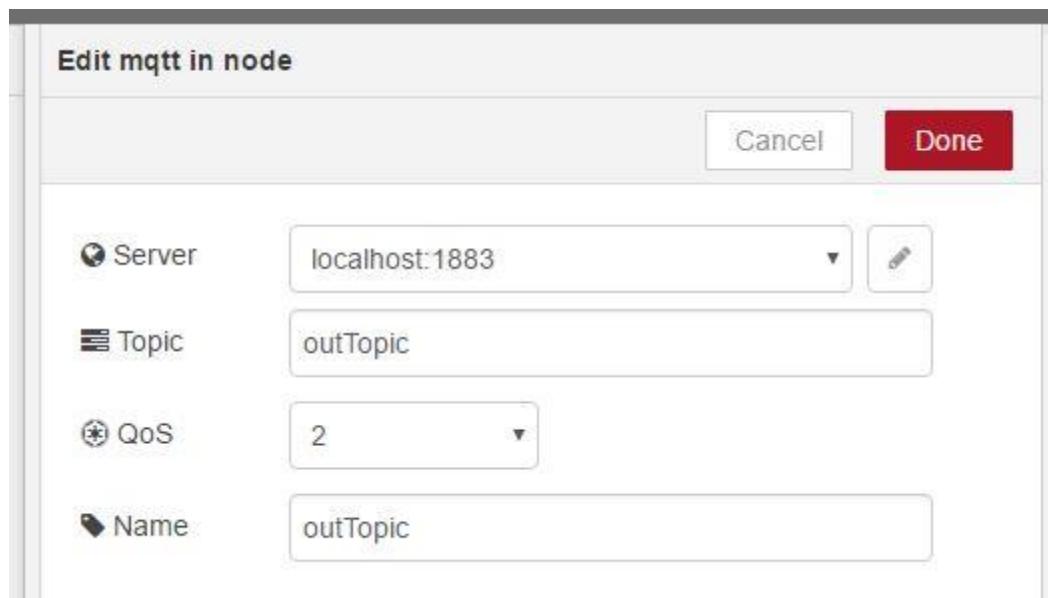
You have to create a Node-RED flow to subscribe to the **outTopic** topic and to publish messages to the **inTopic** topic.

Drag 4 nodes to your flow:

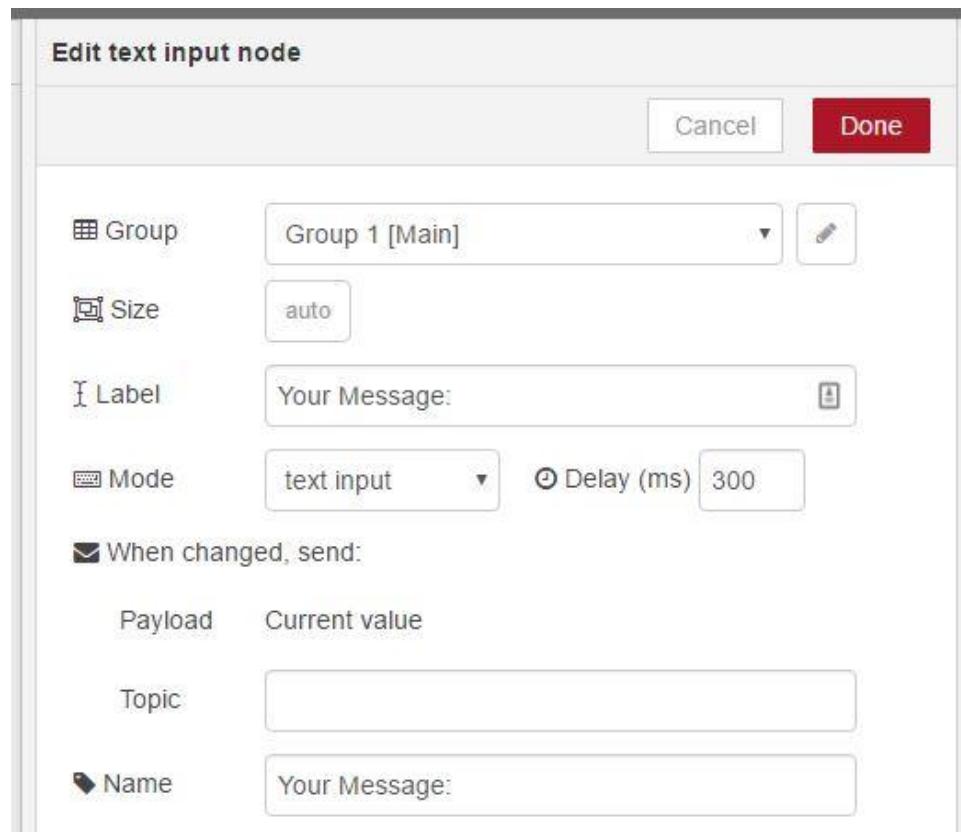
- mqtt in
- Text UI
- Switch UI
- mqtt out



Double-click the **mqtt out** node to edit its settings. Select the **localhost** option and type **outTopic**.

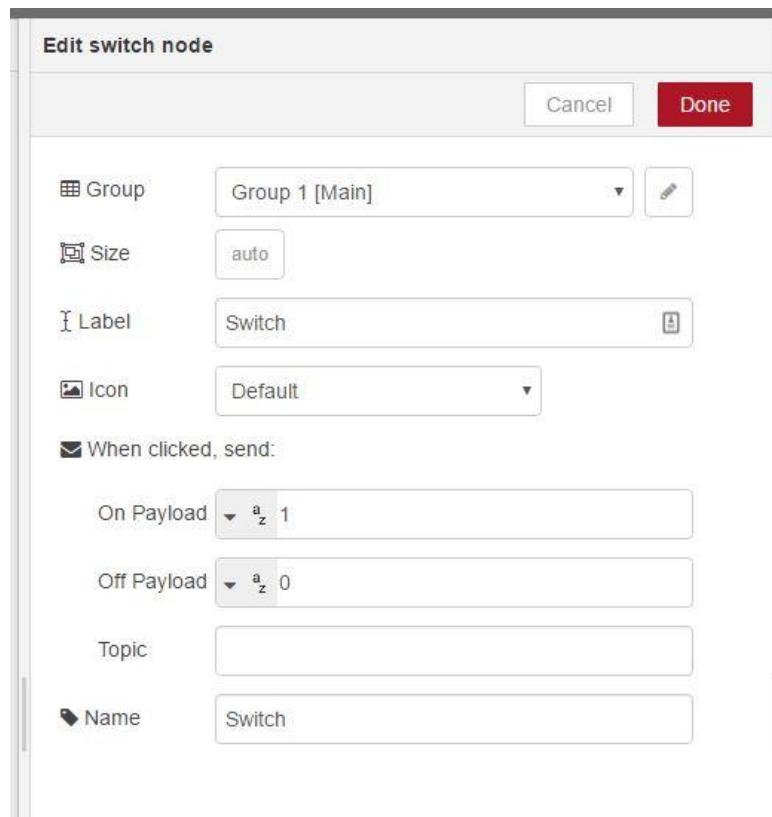


Double-click the **text input** node. Select the **Main** tab and type "Your Message:" in the name field.

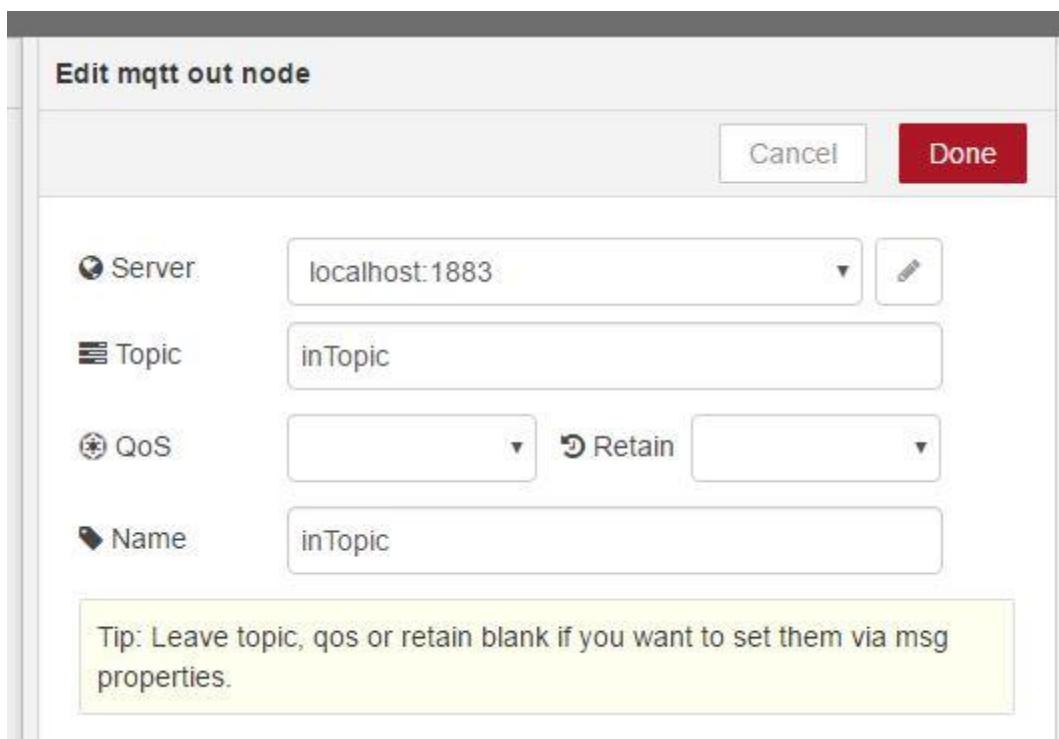


Edit the **switch** node. Select the **Main** tab and name it "**Switch**".

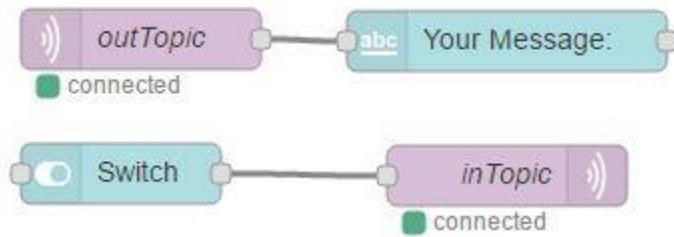
Set the On Value to "**1**" and the Off Value to "**0**".



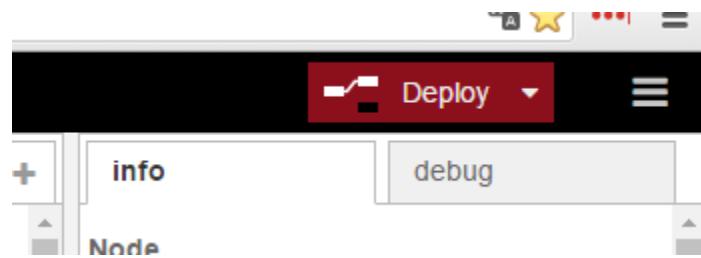
Double-click the **mqtt in** node to edit its settings. Select the **localhost** option and type **inTopic**.



This is how it should look like in the end:

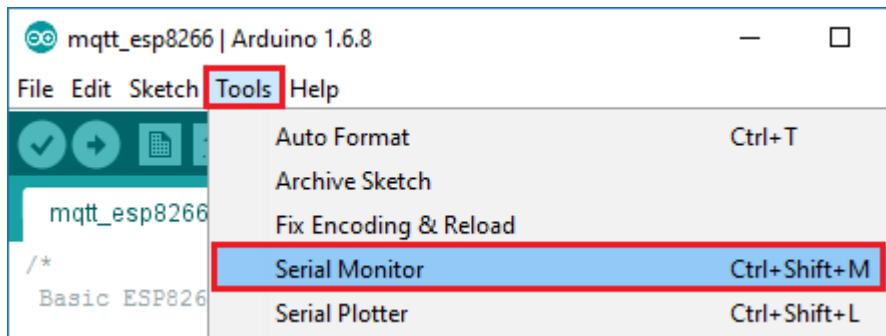


Click the Deploy button on the top-right corner to save your application.

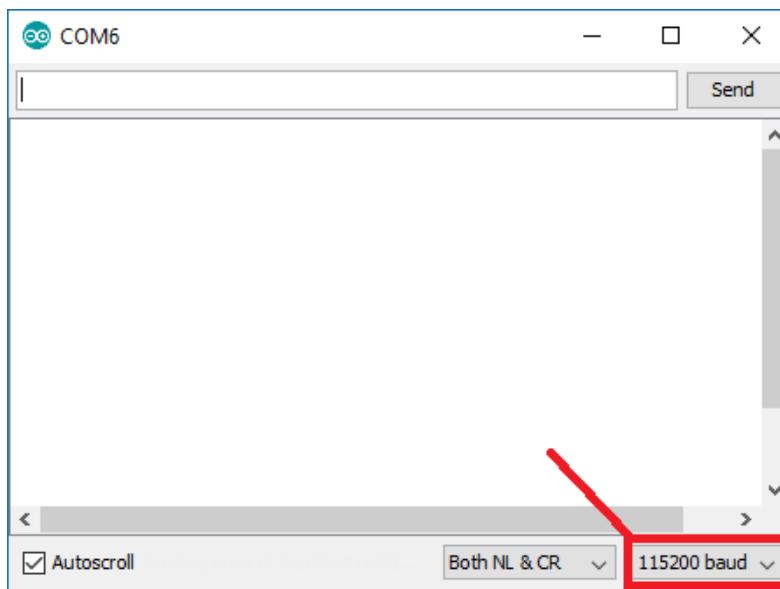


Opening the Node-RED Dashboard

Open the Arduino IDE serial monitor:



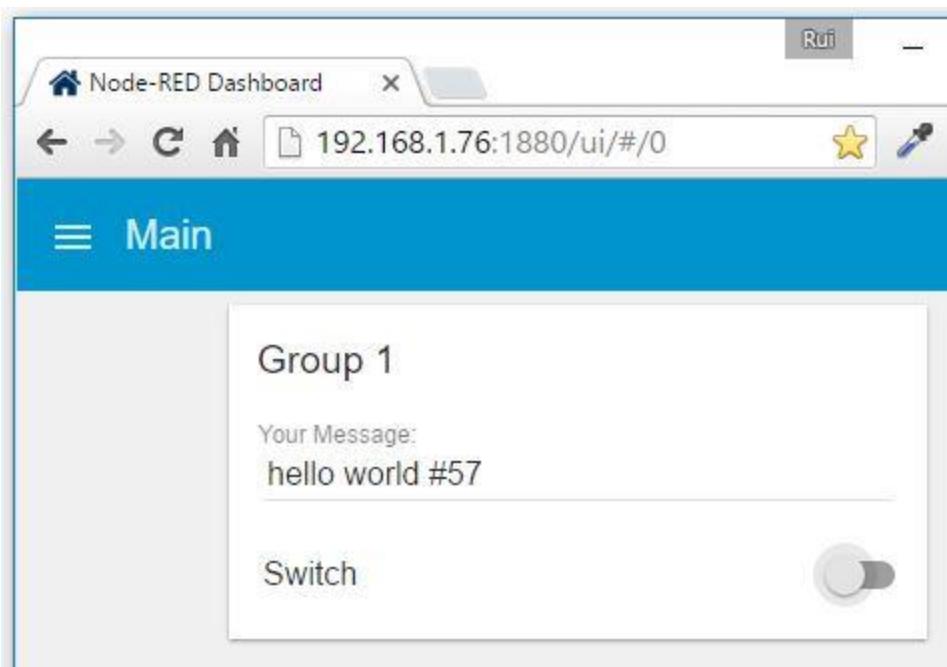
Set the baud rate to 115200:



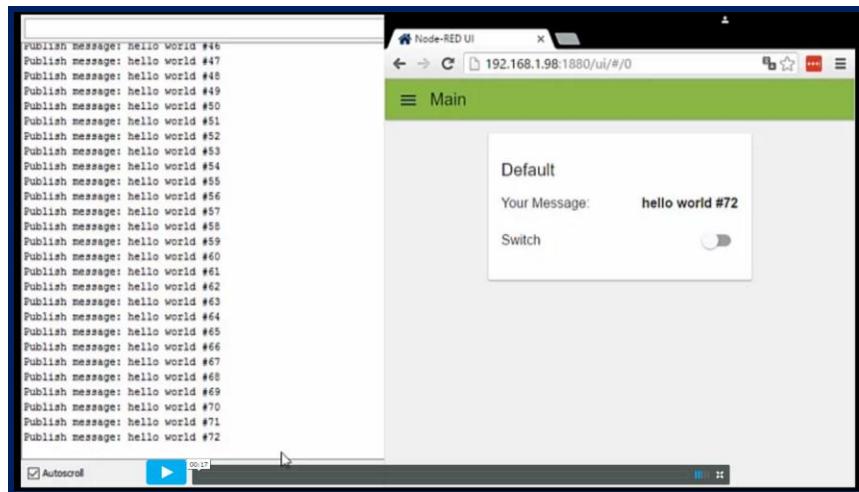
Open the Node-RED Dashboard.

http://YOUR_RPi_IP_ADDRESS:1880/ui

At this point, the Text ui node is subscribed to the **outTopic** topic, so it receives the message “hello world #125” from your ESP8266.

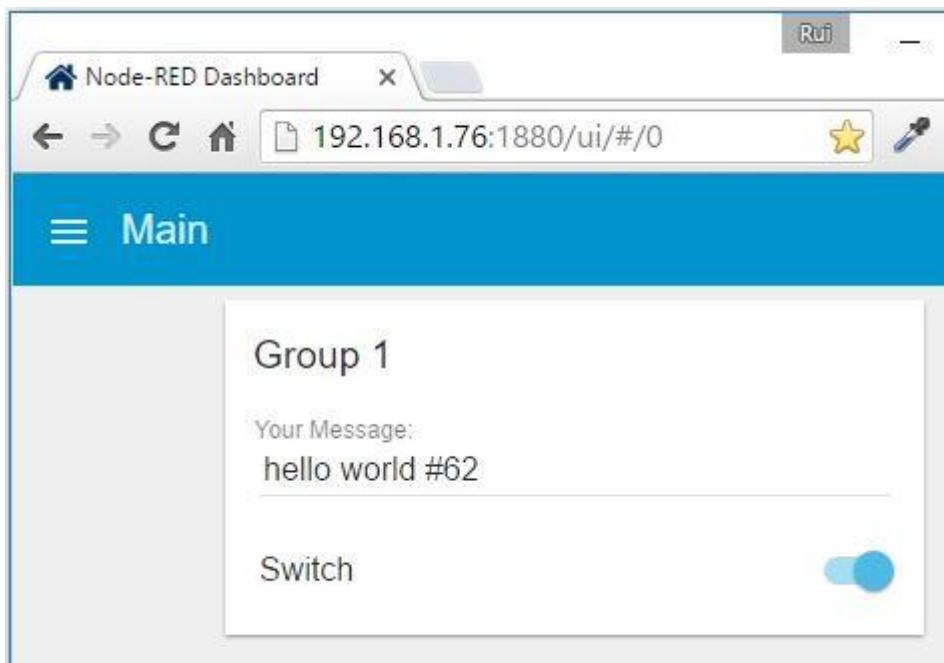


Watch the quick video demonstration of the Node-RED Dashboard receiving the messages from your ESP8266.

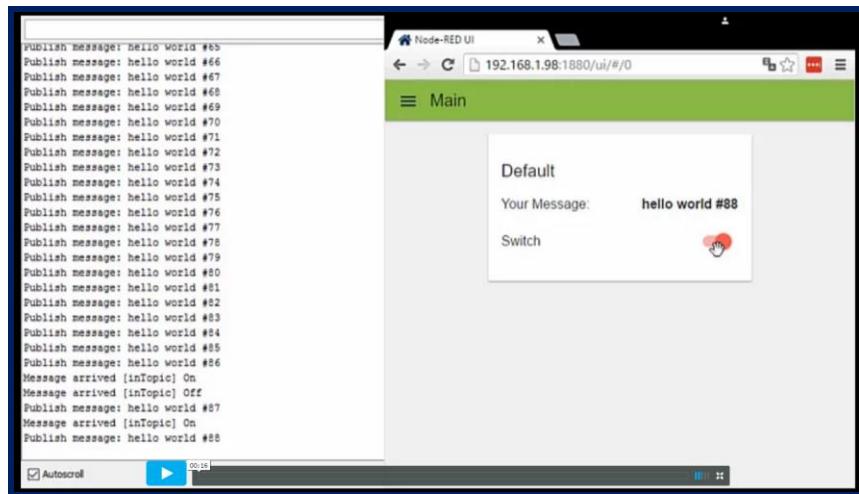


Video # 10 - <https://rntlab.com/28hasvideos>

If you press the Switch, you can publish messages to the **inTopic** topic that your ESP is subscribed to.



As you can see, the ESP8266 is receiving the commands **On** and **Off** when I press the Switch button.



Video # 11 - <https://rntlab.com/28hasvideos>

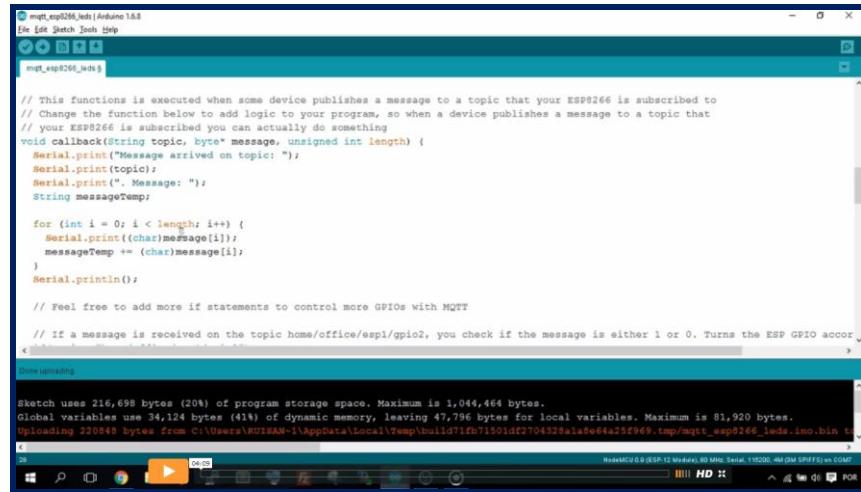
By now you have acquired all the basic concepts. In the next Module, the real home automation project begins. You'll start controlling real lamps and monitor sensors with your ESP8266.

Module 7

Connecting the ESP8266 - Part 2



Unit 1 - Controlling Outputs with ESP using MQTT



The screenshot shows the Arduino IDE interface with the sketch titled "mqtt_esp8266_leds.ino". The code is a MQTT callback function that prints received messages to the serial monitor. It includes comments explaining the logic for handling messages on specific topics. The serial monitor window shows the uploaded sketch's memory usage and the upload progress.

```
// This function is executed when some device publishes a message to a topic that your ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(String topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(": Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
}

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

// If a message is received on the topic home/office/esp1/gpio2, you check if the message is either 1 or 0. Turns the ESP GPIO accordingly

```

Video # 12 - <https://rntlab.com/28hasvideos>

In the previous Unit, I've showed how to subscribe and publish messages to a topic. You're going to use that mechanism to control two LEDs which will be replaced with real lights.

The sketch for this Unit is very similar to the example that you have just used, but it has a few important modifications.

I've also added comments to the sketch to explain what each function does, what you should and shouldn't change. I think it's easier to understand.

You can open this link or scroll down this page to see the complete sketch and follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/mqtt_esp8266_leds.ino

Understanding How the Sketch Works

First, you need to replace those 3 variables with your credentials, so that your ESP8266 can connect to your network and MQTT broker.

```
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";
const char* mqtt_server = "YOUR_RPi_IP_Address";
```

Define two variables that refer to D1 (GPIO 2) and D4 (GPIO 5) of your ESP.

```
// Connect an LED to each GPIO of your ESP8266
const int ledGPIO2 = 2;
const int ledGPIO5 = 5;
```

setup wifi

The `setup_wifi()` function connects your ESP8266 to your router and you don't need to modify anything.

```
void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.print("WiFi connected - ESP IP address: ");
    Serial.println(WiFi.localIP());
}
```

callback

Then you have the `callback()` function.

```
void callback(String topic, byte* message, unsigned int length) {
```

This function is executed when a device publishes a message to a topic that your ESP8266 is subscribed to.

You can change the preceding function to add logic to your programs. For example, when a device publishes a message to a topic that your ESP is subscribed, you can actually do something useful with that message.

These next lines of code turn the LEDs on and off. If a message is received on the topic `home/office/esp1/gpio2`, you check if the message is either 1 or 0.

Turns the ESP GPIO 2 according to the message.

```
if(topic=="home/office/esp1/gpio2"){
    Serial.print("Changing GPIO 2 to ");
    if(messageTemp == "1"){
        digitalWrite(ledGPIO2, HIGH);
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        digitalWrite(ledGPIO2, LOW);
        Serial.print("Off");
    }
}
```

reconnect

The function `reconnect()` reconnects your ESP to your MQTT broker.

```
void reconnect() {
```

It also subscribes to two topics

- `home/office/esp1/gpio52`
- `home/office/esp1/gpio2`

```
client.subscribe("home/office/esp1/gpio5");
client.subscribe("home/office/esp1/gpio2");
```

If you would like to control more GPIOs, you need to subscribe to more topics. Feel free to experiment with that.

setup

The `setup()` function sets your ESP GPIOs as OUTPUTs, starts the serial communication, connects your ESP to your router, sets your MQTT broker and sets the `callback()`.

```
void setup() {
    pinMode(ledGPIO2, OUTPUT);
    pinMode(ledGPIO5, OUTPUT);

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

loop

For this project, you don't need to change anything in the `loop()` function. The `loop()` function checks if your ESP is connected to the MQTT broker. If your ESP isn't connected, it will try to connect/reconnect.

```
void loop() {

    if (!client.connected()) {
        reconnect();
    }
    client.loop();
}
```

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your ESP (after changing the credentials and RPi IP address).

```
*****
```

All the resources for this project:

<https://rntlab.com/>

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

```
// Loading the ESP8266WiFi library and the PubSubClient library
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

// Change the credentials below, so your ESP8266 connects to your router
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// Change the variable to your Raspberry Pi IP address, so it connects to your MQTT broker
const char* mqtt_server = "YOUR_RPi_IP_Address";

// Initializes the espClient
WiFiClient espClient;
PubSubClient client(espClient);

// Connect an LED to each GPIO of your ESP8266
const int ledGPIO2 = 2;
const int ledGPIO5 = 5;

// Don't change the function below. This functions connects your ESP8266 to your router
void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
```

```

    Serial.print(".");
}

Serial.println("");
Serial.print("WiFi connected - ESP IP address: ");
Serial.println(WiFi.localIP());
}

// This functions is executed when some device publishes a message to a topic that your ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(String topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
}

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

// If a message is received on the topic home/office/esp1/gpio2, you check if the message is either 1 or 0. Turns the
// ESP GPIO according to the message
if(topic=="home/office/esp1/gpio2"){
    Serial.print("Changing GPIO 2 to ");
    if(messageTemp == "1"){
        digitalWrite(ledGPIO2, HIGH);
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        digitalWrite(ledGPIO2, LOW);
        Serial.print("Off");
    }
}
if(topic=="home/office/esp1/gpio5"){
    Serial.print("Changing GPIO 5 to ");
    if(messageTemp == "1"){
        digitalWrite(ledGPIO5, HIGH);
        Serial.print("On");
    }
}

```

```

        else if(messageTemp == "0"){
            digitalWrite(ledGPIO5, LOW);
            Serial.print("Off");
        }
    }
    Serial.println();
}

// This functions reconnects your ESP8266 to your MQTT broker
// Change the function below if you want to subscribe to more topics with your ESP8266
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("home/office/esp1/gpio5");
            client.subscribe("home/office/esp1/gpio2");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

// The setup function sets your ESP GPIOs to Outputs, starts the serial communication at a baud rate of 115200
// Sets your mqtt broker and sets the callback function
// The callback function is what receives messages and actually controls the LEDs
void setup() {
    pinMode(ledGPIO2, OUTPUT);
    pinMode(ledGPIO5, OUTPUT);

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

```

```

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

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

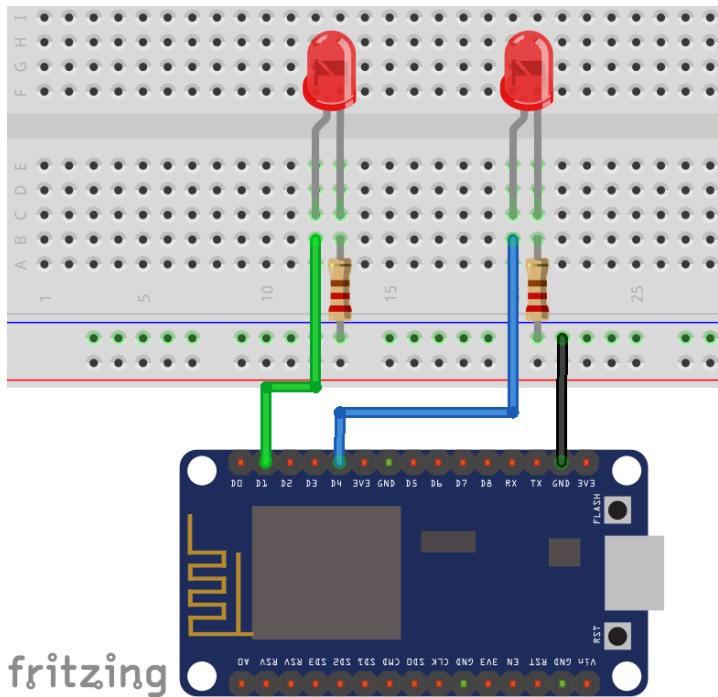
```

DOWNLOAD SOURCE CODE

https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/mqtt_esp8266_leds.ino

Schematics

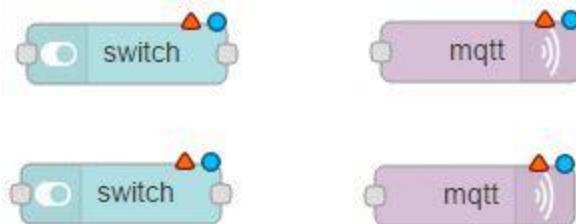
The schematics for this project are very straightforward. Simply connect two LEDs with two resistors to your ESP8266 as shown in the figure below.



Creating the Flow

Go to the Node-RED software and you should create a flow by following these next 7 steps:

1 – Drag 4 Nodes



2 – GPIO 2 Switch node

The screenshot shows the 'Edit switch node' dialog box. At the top right are 'Cancel' and 'Done' buttons. The main area contains the following settings:

- Group:** Group 1 [Main]
- Size:** auto
- Label:** GPIO 2
- Icon:** Default
- When clicked, send:**
 - On Payload:** 1
 - Off Payload:** 0
 - Topic:** (empty)
- Name:** GPIO 2

3 – GPIO 5 Switch node

Edit switch node

Cancel Done

Group Group 1 [Main] ▾

Size auto

Label GPIO 5

Icon Default ▾

When clicked, send:

On Payload ▾ 1

Off Payload ▾ 0

Topic

Name GPIO 5

4 – GPIO 2 MQTT out node

Edit mqtt out node

Cancel Done

Server localhost:1883 ▾

Topic home/office/esp1/gpio2

QoS ▾ Retain ▾

Name Name

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

5 – GPIO 2 MQTT out node

Edit mqtt out node

Cancel Done

Server: localhost:1883

Topic: home/office/esp1/gpio5

QoS: 0 Retain: checked

Name: Name

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

6 – Connect Nodes

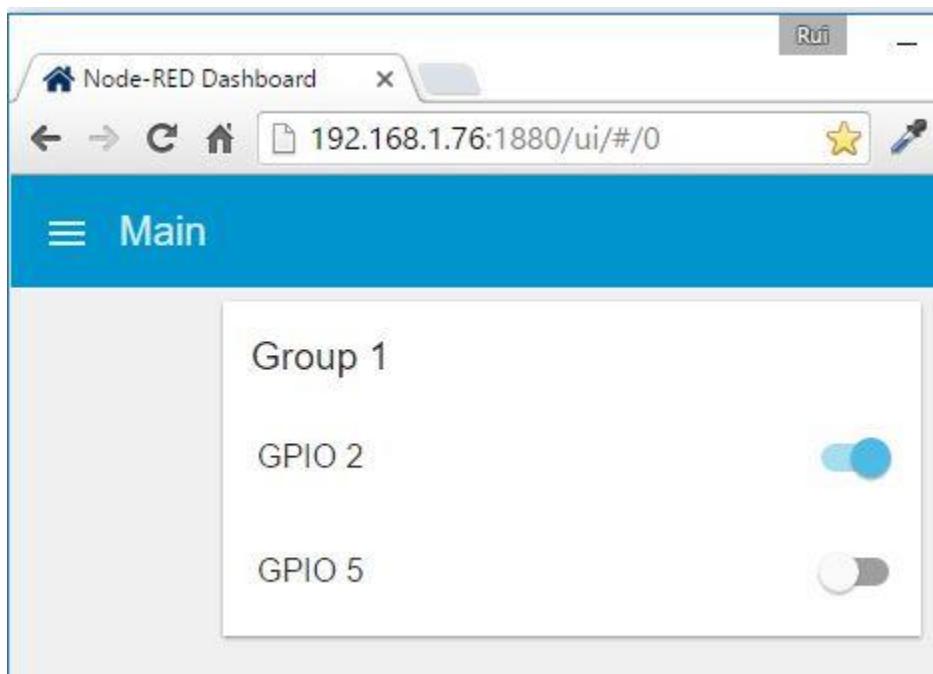


7 – Deploy your application

Deploy ▾

Open Node-RED Dashboard

You can control the LEDs by pressing the switch buttons.



In the next Unit, you're going to replace those LEDs with real lamps.

Unit 2 - Decoding RF Signals to Control Outlets

After so many Units the real fun begins. You are finally going to control real things and make useful stuff.

I've tried different methods of controlling the mains voltage, but some of the methods require:

1. **Experience** dealing with AC voltage
2. **Opening holes** in your wall/ceiling/switches
3. **Modifying** the electrical panel
4. **Knowing** the electrical rules for each country

It was very hard to come up with a solution that is safe and works for everyone.

Solution

I wanted to create a course that would work regardless of the country. The solution for this problem was using radio frequency (RF) controlled outlets.

Why? Using remote controlled outlets have 5 benefits:

1. **Fairly inexpensive**
2. **Easy to get**
3. **Works** with ESP8266 and Arduino
4. **Safe** to use
5. **Works** in any country

Parts Required

To complete this Unit you need:

- 1x Arduino UNO
- 1x 433MHz Receiver
- 1x Remote Controlled Outlets that operate at 433MHz

You can see the complete list of components and parts in Module 1.

Note: you need to buy remote controlled outlets that operate at a RF of 433MHz. They should say the operating RF either in the product page or in the label.

Example

Here's how they look:



There are also [E27 Lamp Bulb Holders](#) that are controlled with an RF remote. They should also operate at 433MHz and they work like the outlets.



You simply connect an E27 lamp to the E27 remote controlled holder.



Then, you attach this to the lamp holder that you want to control. You can turn the light on and off with your remote control:



Setting the RF Channels

I've set my remote control to the **I** position.



The outlets must be both on the **I** position. I've selected channels **3** and **4** for the outlets (you can use any channel you want).



If you plug them to an outlet, you should be able to control the remote controlled outlets with your remote control.

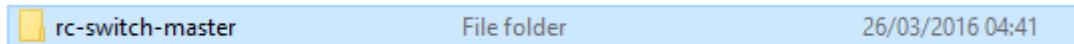
Installing the RC Switch Library

The [RC_Switch_library](#) provides an easy way of using your ESP8266, Arduino or Raspberry Pi to operate remote radio controlled devices. This will most likely work with all popular low cost power outlet sockets.

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



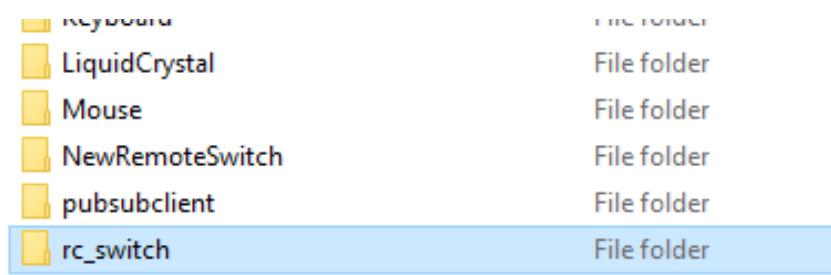
- 2) Unzip the .zip folder and you should get **rc-switch-master** folder



- 3) Rename your folder from **rc-switch-master** to **rc_switch**



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

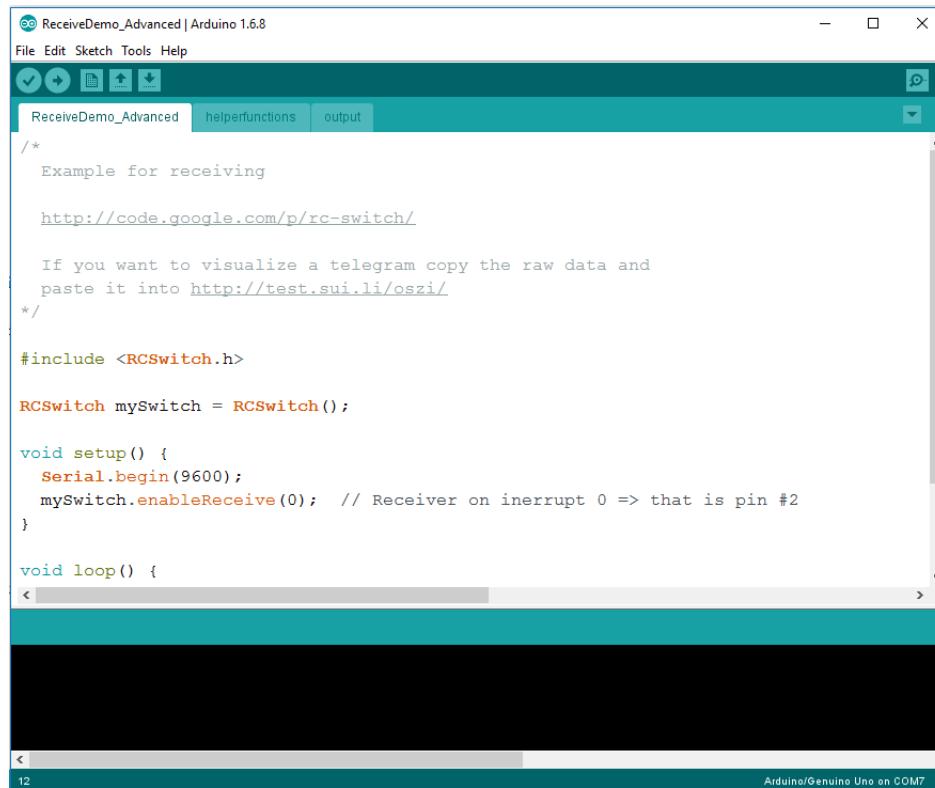


- 5) Then, re-open your Arduino IDE

Opening the Decoder Sketch

You need to decode the signals that your remote control sends, so that the ESP8266 can reproduce those signals and ultimately control the outlets.

The library comes with several sketch examples. Within the Arduino IDE software, you need to go to **File > Examples > RC_Switch > ReceiveDemo_Advanced**.



The screenshot shows the Arduino IDE interface with the title bar "ReceiveDemo_Advanced | Arduino 1.6.8". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for file operations. The main window displays the code for "ReceiveDemo_Advanced". The code is as follows:

```
/*  
 * Example for receiving  
  
 * http://code.google.com/p/rc-switch/  
  
 * If you want to visualize a telegram copy the raw data and  
 * paste it into http://test.sui.li/oszi/  
 */  
  
#include <RCswitch.h>  
  
RCswitch mySwitch = RCswitch();  
  
void setup() {  
    Serial.begin(9600);  
    mySwitch.enableReceive(0); // Receiver on interrupt 0 => that is pin #2  
}  
  
void loop() {
```

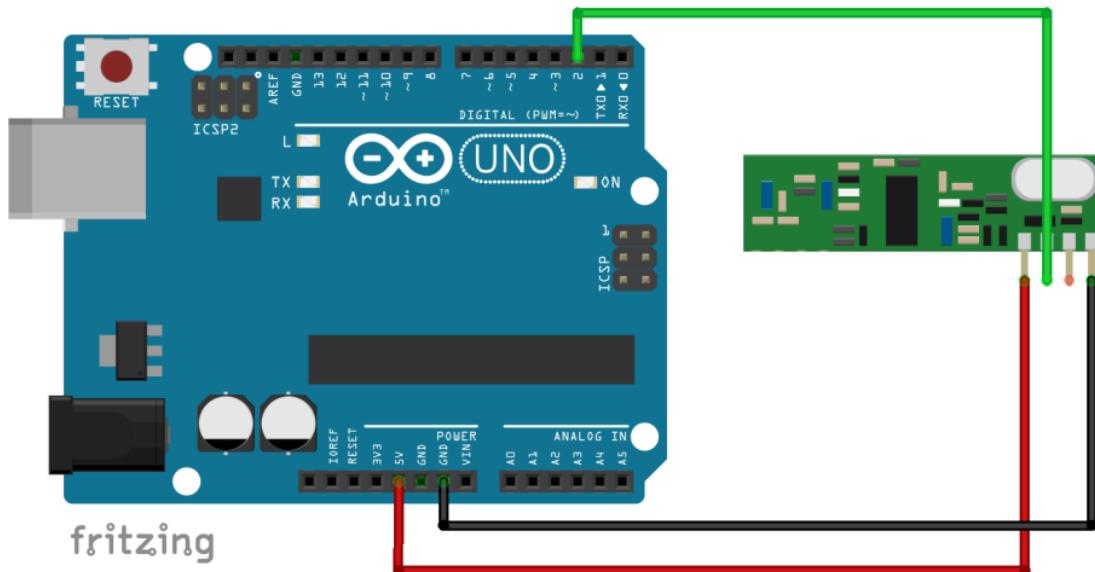
The code is partially visible, with the rest of the loop function obscured by a large black redaction box. At the bottom of the code editor, there is a status bar showing "12" and "Arduino/Genuino Uno on COM7".

Having an Arduino board connected to your computer follow these instructions:

1. Go to the **Tools** tab
2. Select **Arduino UNO** board
3. Select the **COM** port
4. Press the **Upload** button.

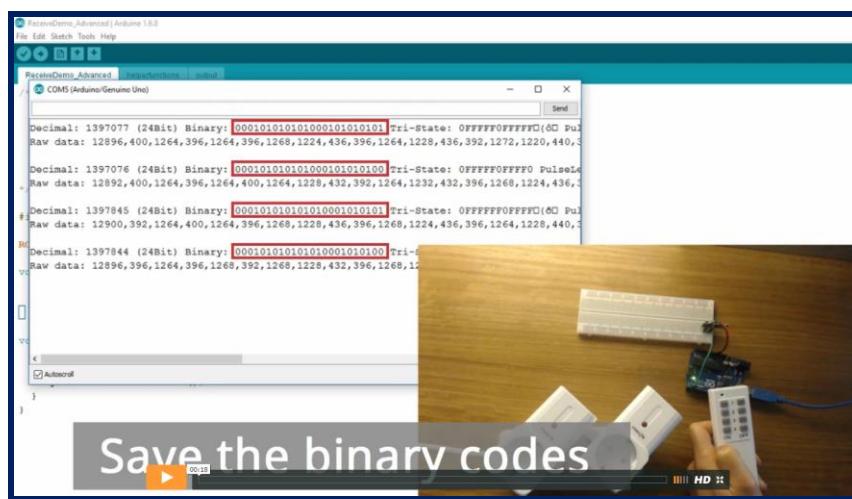
Schematics

After uploading the sketch, connect an 433MHz RF receiver to Digital Pin 2 of your Arduino UNO board:



Decoding the RF Signals

Open the Arduino IDE serial monitor and start pressing the buttons. As shown in the video demonstration below:



Video # 13 - <https://rntlab.com/28hasvideos>

After pressing each button one time, you can see the binary code for each button (it's highlighted in red):

```
COM5 (Arduino/Genuino Uno)

Decimal: 1397077 (24Bit) Binary: 0001010101000101010101 Tri-State: 0FFFFF0FFFF60 PulseLength: 416 microseconds Protocol: 1
Raw data: 12924,392,1272,396,1272,392,1272,1224,440,392,1276,1224,440,392,1272,1224,444,388,1276,1224,440,392,1276,1224,444,388,
Decimal: 1397076 (24Bit) Binary: 0001010101000101010100 Tri-State: 0FFFFF0FFFF0 PulseLength: 416 microseconds Protocol: 1
Raw data: 12924,392,1272,396,1268,396,1268,1228,430,388,1276,1224,440,392,1272,1224,440,396,1268,1228,440,392,1272,1224,440,392,
Decimal: 1397845 (24Bit) Binary: 0001010101000101010101 Tri-State: 0FFFFF0FFFF PulseLength: 416 microseconds Protocol: 1
Raw data: 12928,388,1276,392,1272,392,1272,1228,440,388,1276,1224,444,388,1276,1220,444,388,1276,1224,440,392,1272,1220,444,388,
Decimal: 1397844 (24Bit) Binary: 0001010101000101010000 Tri-State: 0FFFFF0FFF0u0 PulseLength: 416 microseconds Protocol: 1
Raw data: 12928,396,1268,396,1268,396,1268,1228,436,396,1272,1224,440,396,1268,1224,444,392,1272,1224,440,392,1276,1216,448,384,
```

Save your binary codes for each button press:

- **Button 3 ON = (24Bit) Binary:** 0001010101000101010101
- **Button 3 OFF = (24Bit) Binary:** 0001010101000101010100
- **Button 4 ON = (24Bit) Binary:** 0001010101010001010101
- **Button 4 OFF = (24Bit) Binary:** 0001010101010001010100

Save your Pulse Length: 416 Microseconds.

Save your Protocol: 1.

You'll need your binary codes, pulse length and protocol in the next Unit.

Unit 3 - Controlling Lamps and Outlets with ESP using MQTT

It's time to control the outlets with Node-RED Dashboard (watch the video at the end of the Unit to see this project in action).

Understanding How the Sketch Works

You can open the complete sketch in a new window here to follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_esp8266_sockets.ino

It's almost the exact same sketch that you've used to control the LEDs. Here's the main differences:

Preparing RC Switch

Loading the RC Switch library:

```
#include <RCSwitch.h>
```

This line creates the `mySwitch` variable:

```
RCSwitch mySwitch = RCSwitch();
```

In the `setup()` function, you set D0 (GPIO 16) as a transmitter:

```
mySwitch.enableTransmit(16);
```

Set your pulse length:

```
mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);
```

Set your protocol:

```
mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);
```

Subscribing to 2 topics

In the `reconnect()` function, you subscribe to 2 topics that identify where your ESP is located and which outlet you want to control:

```
client.subscribe("home/office/esp1/desk");
client.subscribe("home/office/esp1/workbench");
```

Important: I recommend that you follow this Unit with these exact same topics. However, the idea is to replace them later with topics that clearly identify where your ESP is located and the light you are trying to control.

Sending the binary codes

In the `callback()` function, you create 2 `if` statements for each outlet that send the binary code to turn the outlet on and off.

You have to replace the binary codes with your own:

```

if(topic=="home/office/esp1/desk"){
    Serial.print("Changing Desk Light to ");
    if(messageTemp == "1"){
        // Sends binary code to turn on Desk Light
        mySwitch.send("00010101010001010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        // Sends binary code to turn off Desk Light
        mySwitch.send("0001010101010001010100");
        Serial.print("Off");
    }
}
if(topic=="home/office/esp1/workbench"){
    Serial.print("Changing Workbench Light to ");
    if(messageTemp == "1"){
        // Sends binary code to turn on Workbench Light
        mySwitch.send("0001010101010001010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        // Sends binary code to turn off Workbench Light
        mySwitch.send("000101010101010001010100");
        Serial.print("Off");
    }
}

```

Important: in case you change the topics, you also have to change the if statements to match the topics that your ESP8266 is subscribed.

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your ESP (after changing the credentials, RPi IP address and binary codes of your remote control).

Note: make sure you select the **ESP board** in the **Tools** tab and the correct **COM port**.

All the resources for this project:

<https://rntlab.com/>

```

// Loading the ESP8266WiFi library and the PubSubClient library
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
// Loading the RCSSwitch library to control the outlets
#include <RCSSwitch.h>

// Change the credentials below, so your ESP8266 connects to your router
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// Change the variable to your Raspberry Pi IP address, so it connects to your MQTT broker
const char* mqtt_server = "YOUR_RPi_IP_Address";

// Initializes the espClient. You have to change the espClient name if you have multiple ESPs running in your home
automation system
WiFiClient espClient;
PubSubClient client(espClient);

// Initializes the RCSSwitch
RCSSwitch mySwitch = RCSSwitch();

// Don't change the function below. This functions connects your ESP8266 to your router
void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.print("WiFi connected - ESP IP address: ");
    Serial.println(WiFi.localIP());
}

// This functions is executed when some device publishes a message to a topic that your ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(String topic, byte* message, unsigned int length) {

```

```

Serial.print("Message arrived on topic: ");
Serial.print(topic);
Serial.print(". Message: ");
String messageTemp;

for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
}
Serial.println();

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

// If a message is received on the topic home/office/esp1/gpio2, you check if the message is either 1 or 0. Turns the
// ESP GPIO according to the message
if(topic=="home/office/esp1/desk"){
    Serial.print("Changing Desk Light to ");
    if(messageTemp == "1"){
        // Sends binary code to turn on Desk Light
        // BINARY CODE EXAMPLE. REPLACE WITH YOUR BINARY CODE
        mySwitch.send("000101010101000101010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        // Sends binary code to turn off Desk Light
        // BINARY CODE EXAMPLE. REPLACE WITH YOUR BINARY CODE
        mySwitch.send("0001010101010001010100");
        Serial.print("Off");
    }
}

if(topic=="home/office/esp1/workbench"){
    Serial.print("Changing Workbench Light to ");
    if(messageTemp == "1"){
        // Sends binary code to turn on Workbench Light
        // BINARY CODE EXAMPLE. REPLACE WITH YOUR BINARY CODE
        mySwitch.send("0001010101010001010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        // Sends binary code to turn off Workbench Light
        // BINARY CODE EXAMPLE. REPLACE WITH YOUR BINARY CODE
        mySwitch.send("0001010101010001010100");
        Serial.print("Off");
    }
}

```

```

        }
    }
    Serial.println();
}

// This functions reconnects your ESP8266 to your MQTT broker
// Change the function below if you want to subscribe to more topics with your ESP8266
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("home/office/esp1/desk");
            client.subscribe("home/office/esp1/workbench");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

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

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

    mySwitch.enableTransmit(16);

    // SET YOUR PULSE LENGTH
    mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);
}

```

```

// SET YOUR PROTOCOL (default is 1, will work for most outlets)
mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);

// Set number of transmission repetitions.
mySwitch.setRepeatTransmit(15);
}

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

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

```

DOWNLOAD SOURCE CODE

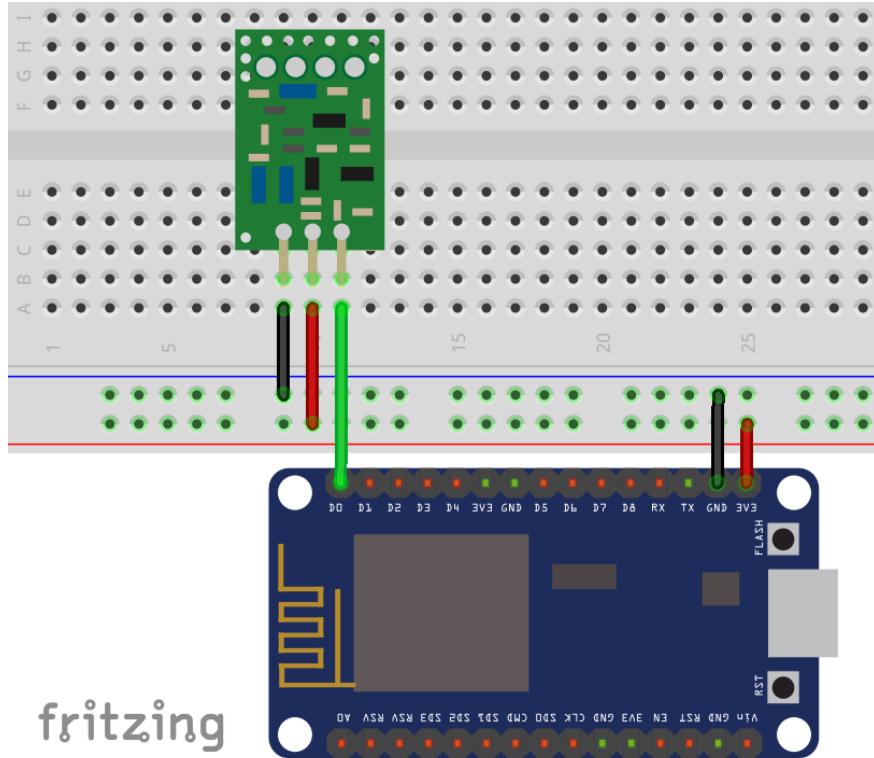
https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_esp8266_sockets.ino

Schematics

To complete this Unit you need:

- 1x ESP8266-12E
- 1x 433MHz Transmitter
- 1x Remote Controlled Outlets that operate at 433MHz

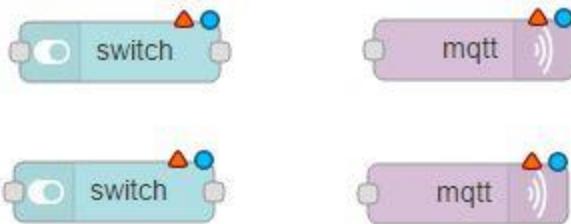
Here's the schematics:



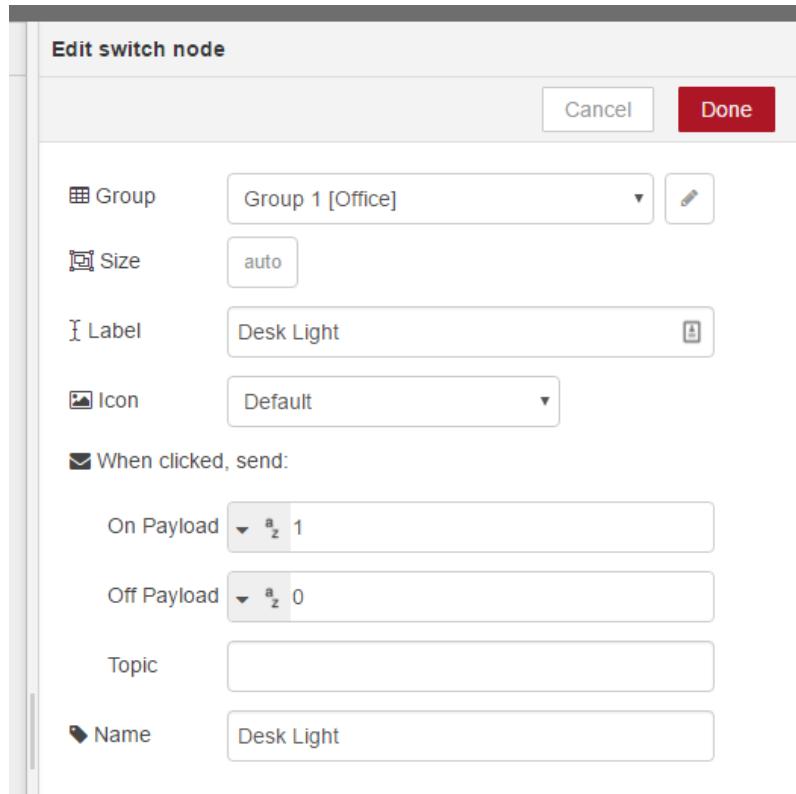
Creating the Flow

The flow is very simple and similar to one presented in an earlier Unit. Here's the 7 steps that you need to follow:

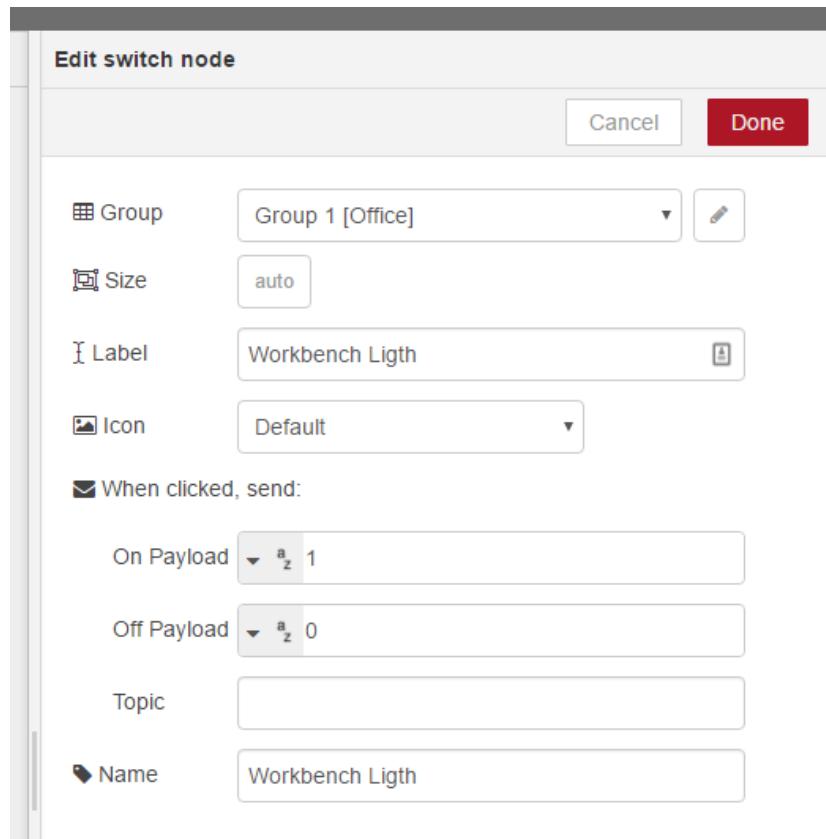
1 – Drag 4 Nodes



2 – Desk Switch node



3 – Workbench Switch node



4 – Desk MQTT out node

Edit mqtt out node

Cancel Done

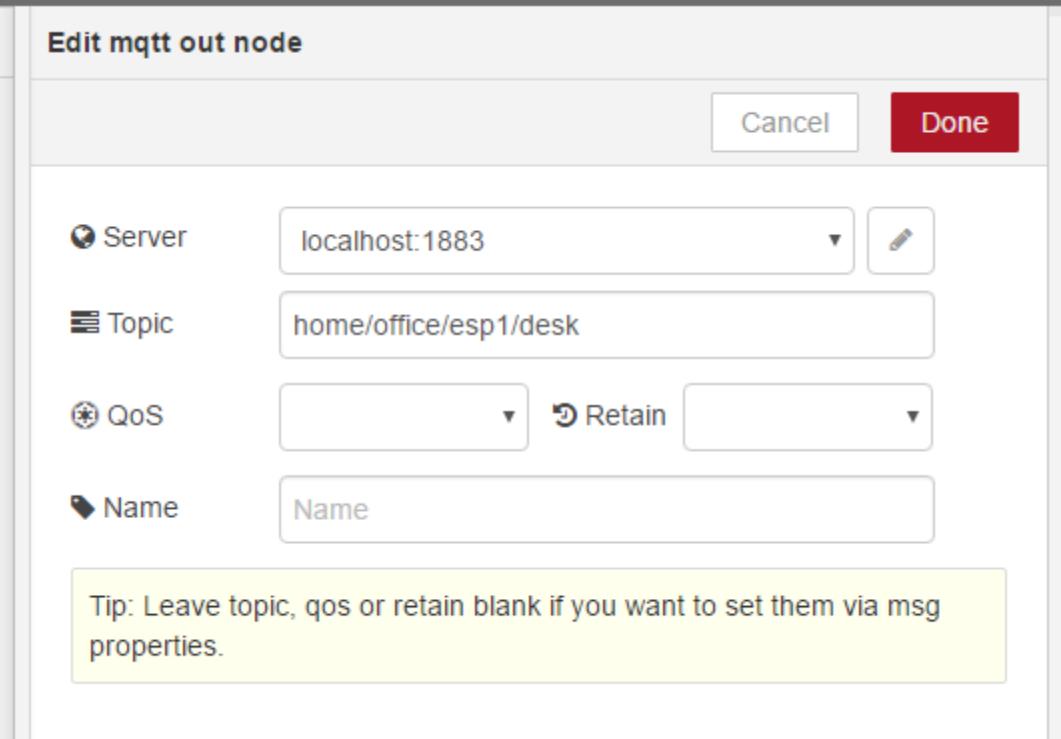
Server: localhost:1883

Topic: home/office/esp1/desk

QoS: [dropdown] Retain: [dropdown]

Name: Name

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



5 – Workbench MQTT out node

Edit mqtt out node

Cancel Done

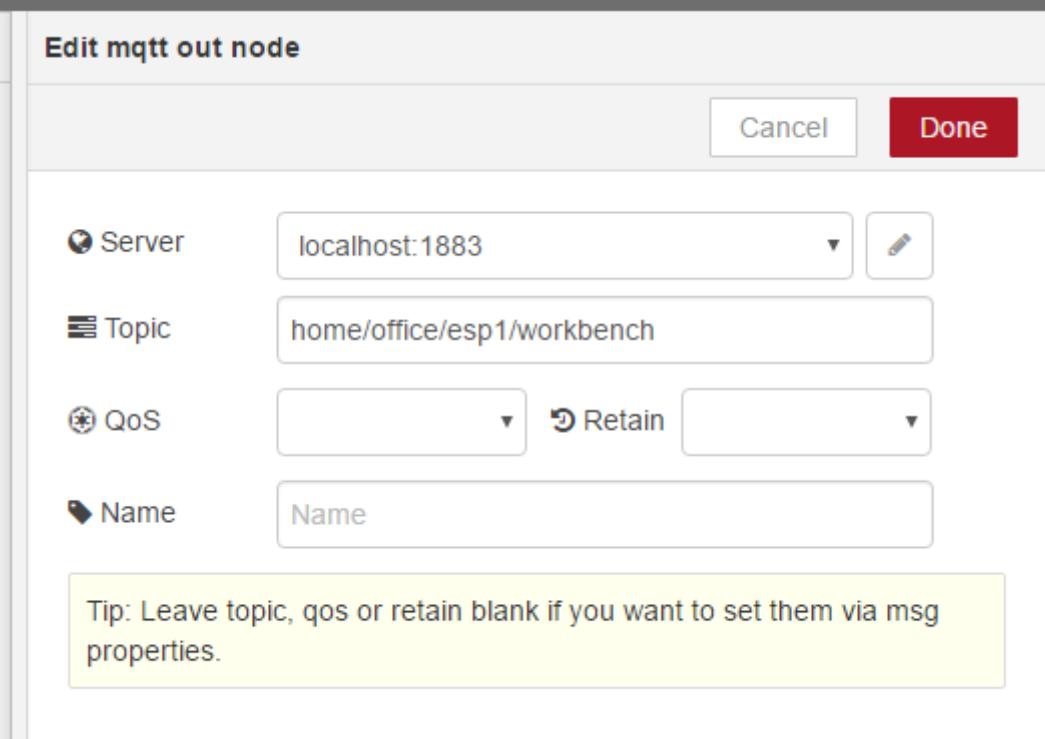
Server: localhost:1883

Topic: home/office/esp1/workbench

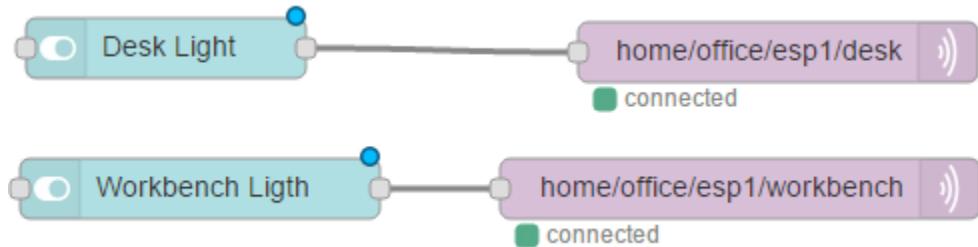
QoS: [dropdown] Retain: [dropdown]

Name: Name

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



6 – Connect Nodes

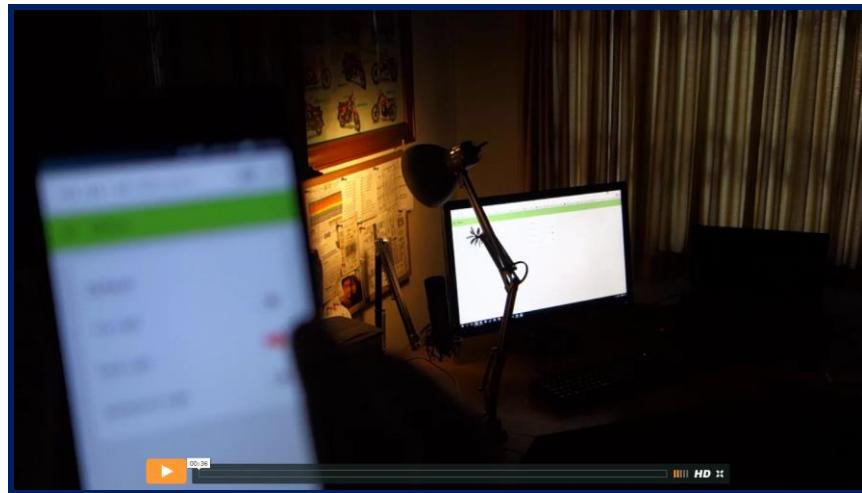


7 – Deploy your application



Watch the Video Demonstration

Finally, you can control the light and any outlet in your home with any device that has a web browser. Watch this video to see it in action:



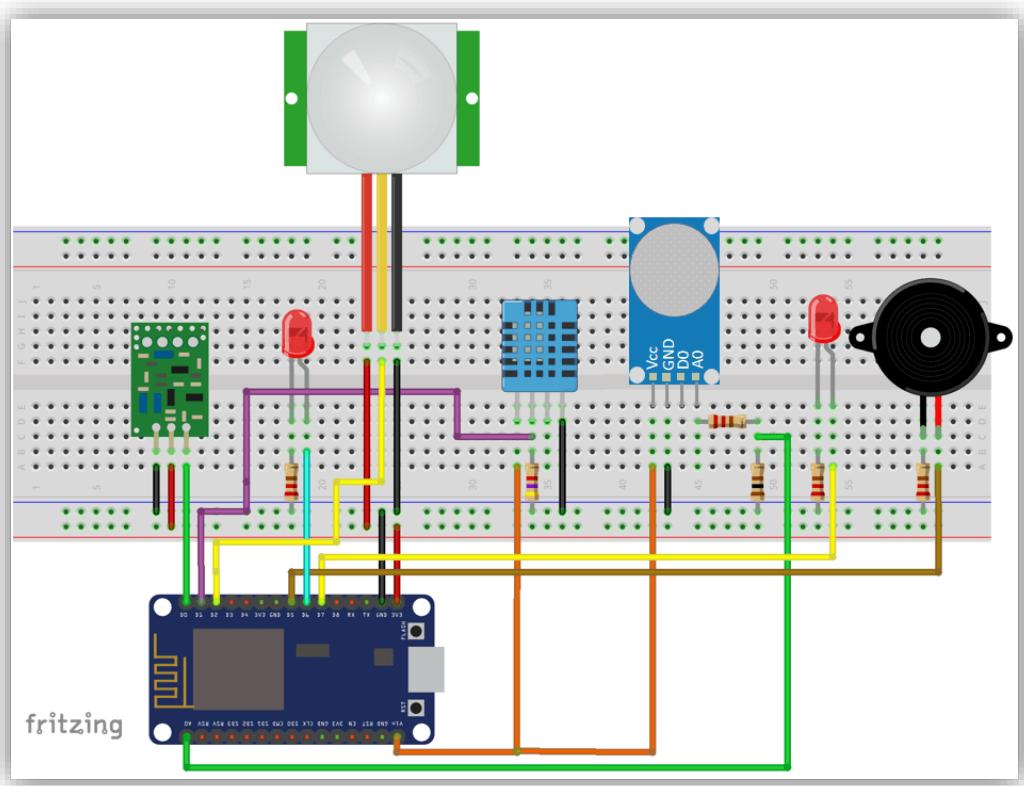
Video # 14 - <https://rntlab.com/28hasvideos>

In the next Module, you're going to add a motion sensor to detect movement and notify you via email.

You're also going to connect a gas sensor to detect flammable gases. Lastly, you're going to store everything inside a project box.

Module 8

Connecting the ESP8266 - Part 3

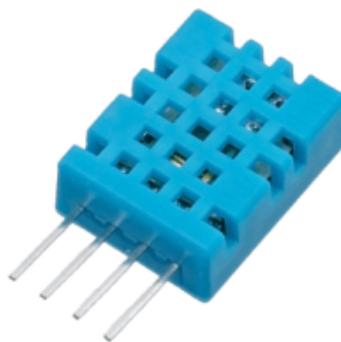


Unit 1 - Reading the Temperature and Humidity

(You should keep the sketch, circuit and nodes from the previous Unit and add this new sensor to your project)

This Unit shows you how to read temperature and humidity with an ESP8266 and how to publish those values to the Node-RED Dashboard.

Throughout this Unit you're going to use the DHT11 sensor.



Installing the DHT Sensor Library

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

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



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



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



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

arduino-1.6.8 > libraries	
Name	Type
Bridge	File folder
DHT	File folder
Esplora	File folder
Ethernet	File folder
Firmata	File folder
SSM	File folder

5) Then, re-open your Arduino IDE

Understanding How the Sketch Works

You can open the complete sketch in a new window here to follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_esp8266_temperature_humidity.ino

Preparing DHT

The following line loads the DHT sensor library:

```
#include "DHT.h"
```

You have to define the type of DHT sensor that you're using. In my case, I'm using the DHT11, but if you're using another version you can comment the DHT11 line and uncomment your sensor version.

```
// Uncomment one of the lines below for whatever DHT sensor type you're using!
#define DHTTYPE DHT11    // DHT 11
//#define DHTTYPE DHT21    // DHT 21 (AM2301)
//#define DHTTYPE DHT22    // DHT 22 (AM2302), AM2321
```

Then, you set the D1 (GPIO 5) as the pin `DHTPin` variable and you prepare the `dht` sensor variable:

```
// DHT Sensor
const int DHTPin = 5;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);
```

You need two variables to keep track of the time, so that you can create a timer that publishes new temperature and humidity measurements every 30 seconds.

```
// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;
```

setup

In the `setup()` function, you begin the `dht` sensor:

```
dht.begin();
```

loop

Finally, you have the `loop()` function that contains a timer that compares the time that has passed since the last sensor measurement.

If it has passed 30 seconds, it goes into that `if` statement.

```
now = millis();
// Publishes new temperature and humidity every 30 seconds
if (now - lastMeasure > 30000) {
    lastMeasure = now;
```

Note: I recommend you to only read the DHT sensor every 30 seconds or above, because the DHT sensor takes a long time to read the values and it might interfere with other actions that your ESP8266 is performing.

These first 3 lines of code read the humidity, temperature in Celsius and Fahrenheit. It also checks if the sensor reading was successful.

```
// Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
float h = dht.readHumidity();
// Read temperature as Celsius (the default)
float t = dht.readTemperature();
// Read temperature as Fahrenheit (isFahrenheit = true)
float f = dht.readTemperature(true);

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

The next snippet of code computes the sensor values and saves them in temporary variables (converts float to char).

Tip: you can comment the Celsius code and uncomment the Fahrenheit code, if you prefer.

```

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

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

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

```

Finally, you need to publish the temperature and humidity measurements to the following MQTT topics:

- `home/office/esp1/temperature`
- `home/office/esp1/humidity`

Later, you can change these topics to fit your rooms or devices.

```

// Publishes Temperature and Humidity values
client.publish("home/office/esp1/temperature", temperatureTemp);
client.publish("home/office/esp1/humidity", humidityTemp);

```

Lastly, you have these lines of code to print the temperature and humidity values in the Arduino IDE serial monitor.

Note: you can delete these lines after testing your project:

```

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

```

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your ESP (after changing the credentials, RPi IP address and binary codes of your remote control).

Note #1: Make sure you select the **ESP board** in the **Tools** tab and the correct **COM** port.

Note #2: You can comment/uncomment some lines if you're using another DHT sensor or if you're reading temperature in Fahrenheit.

```

*****
All the resources for this project:  

https://rntlab.com/  

*****  

#include <ESP8266WiFi.h>
#include <PubSubClient.h>
// Loading the RCSSwitch library to control the outlets
#include <RCSSwitch.h>
#include "DHT.h"  

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

```

```

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

// Change the credentials below, so your ESP8266 connects to your router
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// Change the variable to your Raspberry Pi IP address, so it connects to your MQTT broker
const char* mqtt_server = "YOUR_RPi_IP_Address";

// Initializes the espClient. You have to change the espClient name if you have multiple ESPs running in your home
automation system
WiFiClient espClient;
PubSubClient client(espClient);

RCSwitch mySwitch = RCSwitch();

// DHT Sensor
const int DHTPin = 5;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;

// Don't change the function below. This functions connects your ESP8266 to your router
void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.print("WiFi connected - ESP IP address: ");
    Serial.println(WiFi.localIP());
}

```

```

}

// This functions is executed when some device publishes a message to a topic that your ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(String topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
}

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

// If a message is received on the topic home/office/esp1/desk, you check if the message is either 1 or 0. Turns the
// Desk outlet according to the message
if(topic=="home/office/esp1/desk"){
    Serial.print("Changing Desk light to ");
    if(messageTemp == "1"){
        mySwitch.send("000101010101000101010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("000101010101000101010100");
        Serial.print("Off");
    }
}

if(topic=="home/office/esp1/workbench"){
    Serial.print("Changing Workbench light to ");
    if(messageTemp == "1"){
        mySwitch.send("0001010101010001010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("0001010101010001010100");
        Serial.print("Off");
    }
}
}

```

```

Serial.println();
}

// This functions reconnects your ESP8266 to your MQTT broker
// Change the function below if you want to subscribe to more topics with your ESP8266
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("home/office/esp1/desk");
            client.subscribe("home/office/esp1/workbench");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

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

    dht.begin();

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

    mySwitch.enableTransmit(16);

    // SET YOUR PULSE LENGTH
    mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);
}

```

```

// SET YOUR PROTOCOL (default is 1, will work for most outlets)
mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);

// Set number of transmission repetitions.
mySwitch.setRepeatTransmit(15);
}

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

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

now = millis();
// Publishes new temperature and humidity every 30 seconds
if (now - lastMeasure > 30000) {
    lastMeasure = now;
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    float h = dht.readHumidity();
    // Read temperature as Celsius (the default)
    float t = dht.readTemperature();
    // Read temperature as Fahrenheit (isFahrenheit = true)
    float f = dht.readTemperature(true);

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

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

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

```

```

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

// Publishes Temperature and Humidity values
client.publish("home/office/esp1/temperature", temperatureTemp);
client.publish("home/office/esp1/humidity", humidityTemp);

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

```

DOWNLOAD SOURCE CODE

https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_esp8266_temperature_humidity.ino

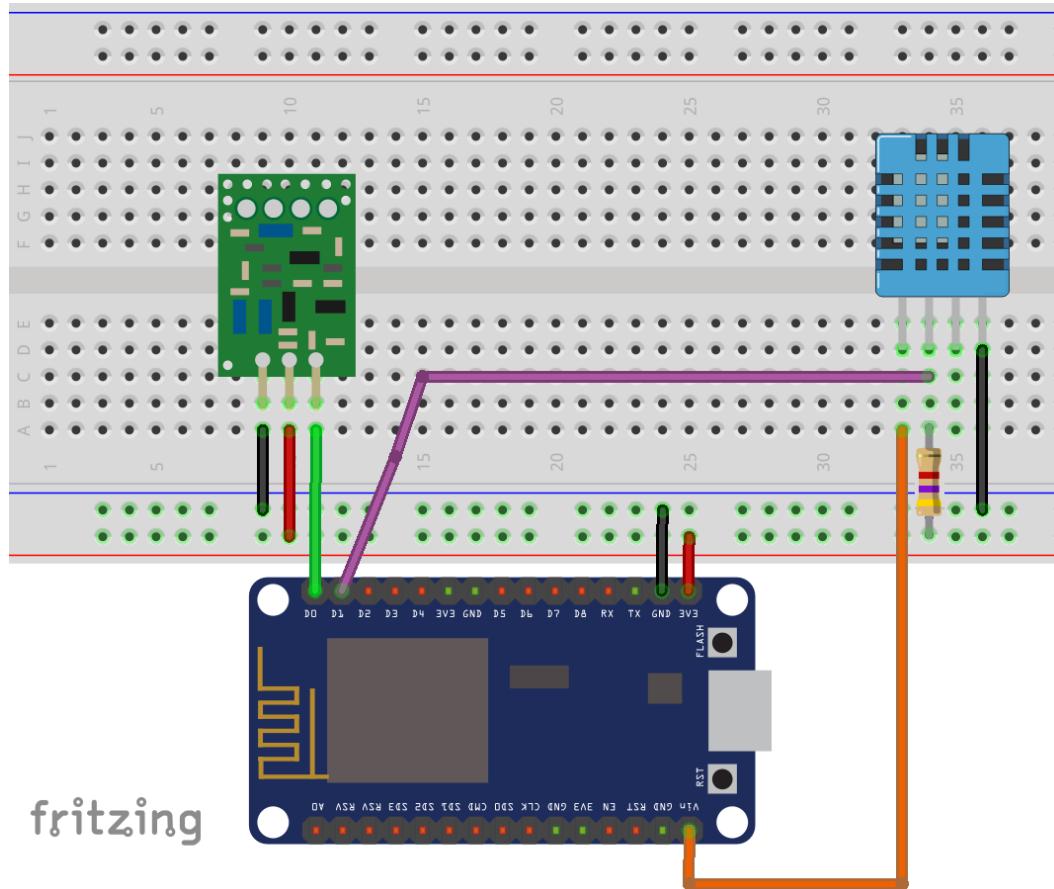
Schematics

To complete this Unit you need (highlighted you can see the new components):

- 1x ESP8266 12E
- 1x 433MHz Transmitter
- 1x Remote Controlled Outlets that operate at 433MHz
- 1x DHT11 Sensor
- 1x 4700 Ohm Resistor

Note: other DHT sensor types will also work with a small change in the code (as described in a preceding section).

Here's the schematics:

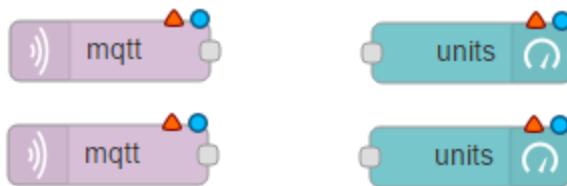


Important: The DHT sensor requires 5V to operate properly, so make sure you use the Vin pin from your ESP8266 that outputs 5V.

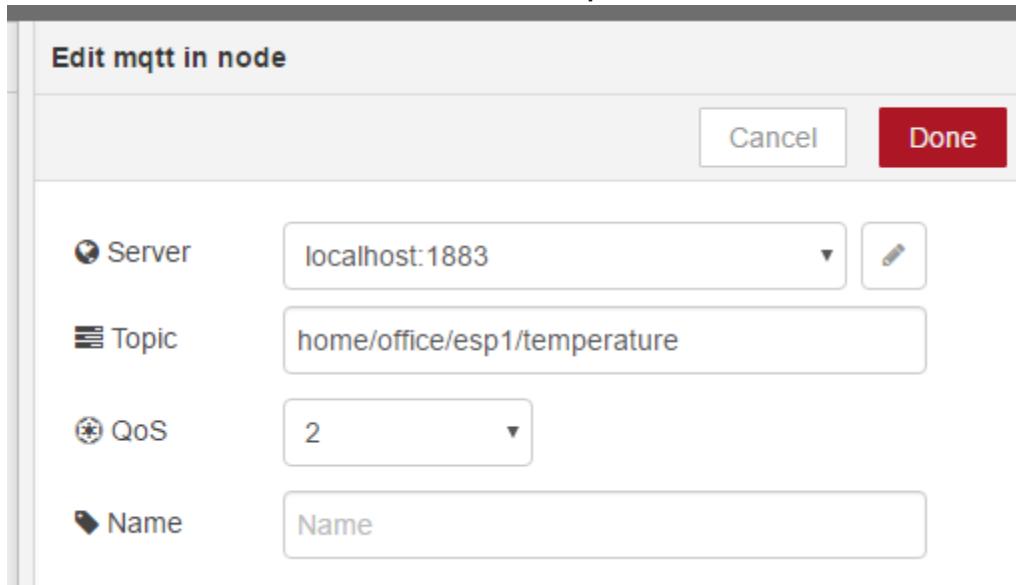
Creating the Flow

In this flow, you're going to use a node called **Gauge** to plot the temperature and humidity values. Follow these next 8 steps to create your flow:

1 – Drag 4 Nodes



2 – MQTT out Temperature



3 – MQTT out Humidity

Edit mqtt in node

Cancel Done

Server	localhost:1883	▼	
Topic	home/office/esp1/humidity		
QoS	2	▼	
Name	Name		

4 – Gauge Temperature for Celsius

Edit gauge node

Cancel Done

Group	Group 1 [Office]	▼	
Size	auto		
Type	Gauge	▼	
Title	Temperature		
Value format	{{value}}		
Label	units		
Range	min -10	max 40	
Name	Temperature		

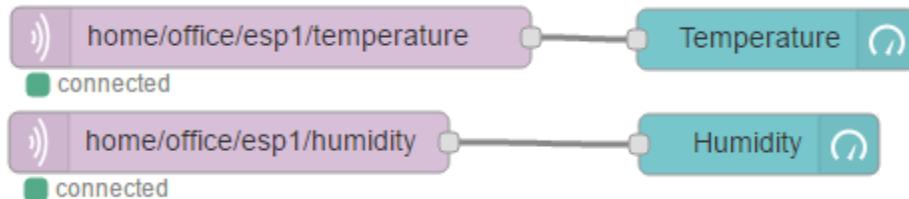
5 – Gauge Humidity (0 to 100%)

Edit gauge node

Cancel Done

Group	Group 1 [Office]	
Size	auto	
Type	Gauge	
Title	Humidity	
Value format	{{{value}}}	
Label	units	
Range	min 0	max 100
Name	Humidity	

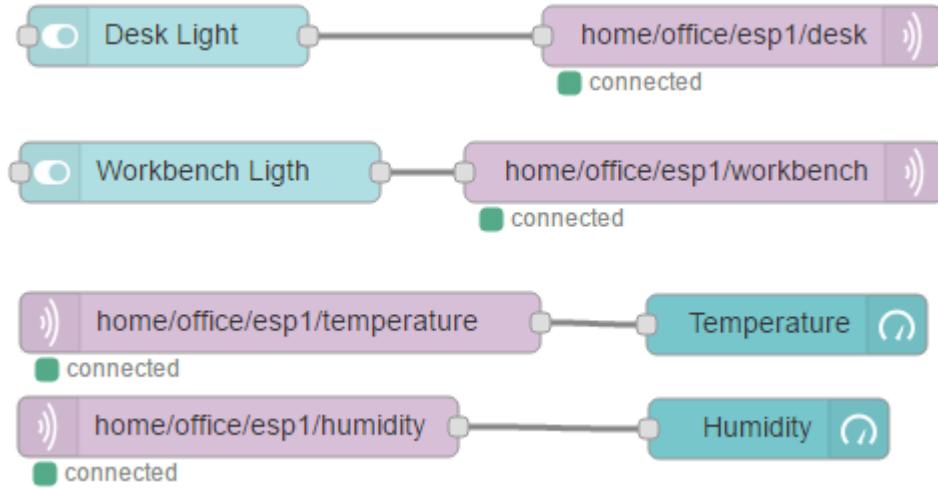
6 – Connecting Nodes



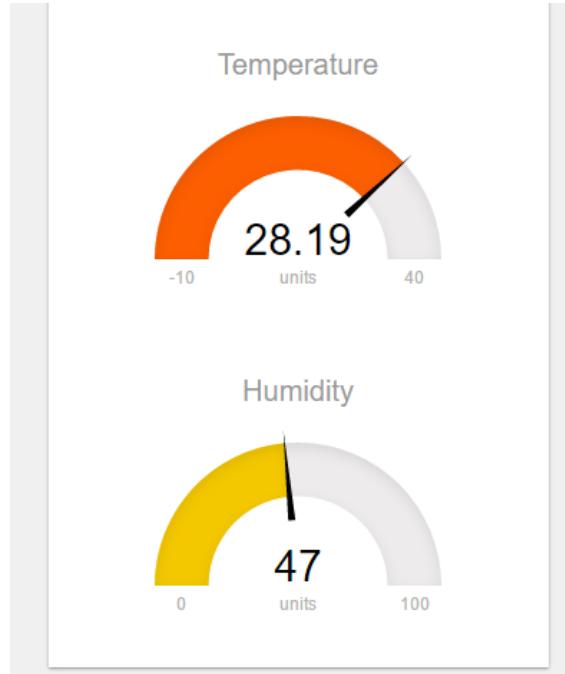
7 – Deploy your application



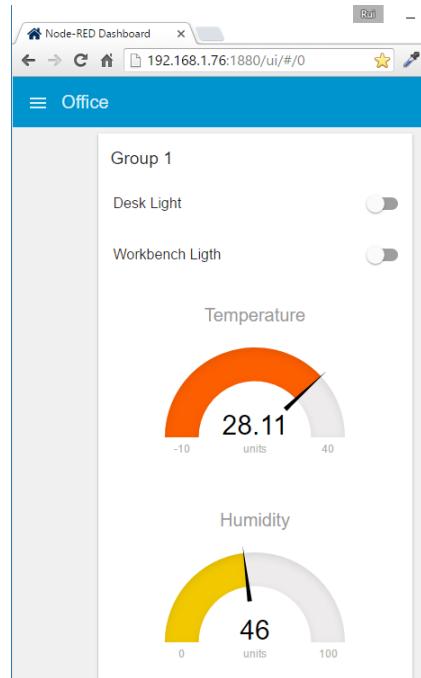
Final Flow



8 – Gauges

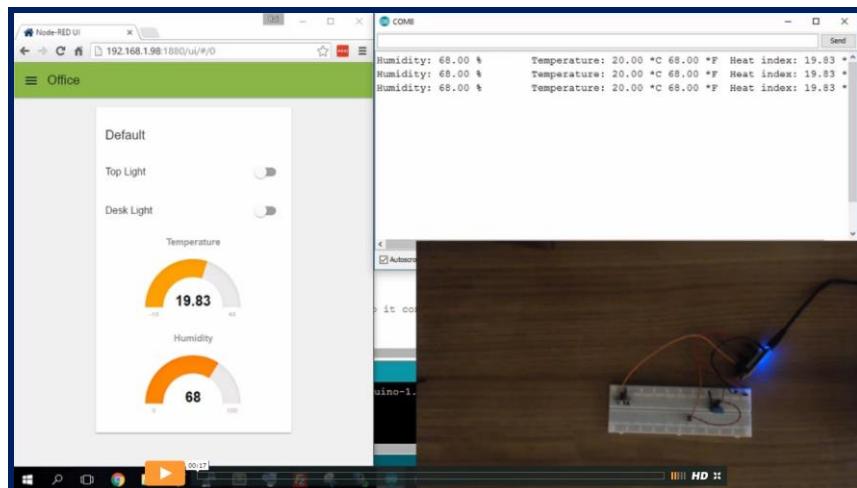


When you go to the Node-RED Dashboard tab, here's what you should see:



Watch the Video Demonstration

You can see your room temperature and humidity updated every 30 seconds in the Node-RED Dashboard. Watch this video to see it in action:



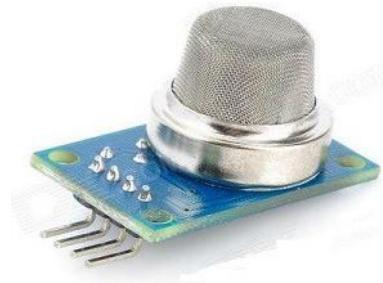
Video # 15 - <https://rntlab.com/28hasvideos>

Unit 2 - Smoke and Gas Detector

(You should keep the sketch, circuit and nodes from the previous Units and add this new sensor to your project)

This Unit shows you how to build a flammable gas detector that beeps when it detects gas and publishes a notification to the Node-RED Dashboard.

Throughout this Unit you're going to use the MQ-2 sensor.



Understanding How the Sketch Works

You can open the complete sketch in a new window here to follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_esp8266_smoke.ino

Preparing the variables

Assign the Analog pin A0 to the `smokePin` variable.

```
// Smoke Sensor  
int smokePin = A0;
```

Then, assign the pin D5 (GPIO 14) to the `buzzerPin` variable.

```
// Buzzer  
const int buzzerPin = 14;
```

You also need to define a threshold value for the smoke sensor, I'll be using `smokeThres=60` (you might need to change this value to adjust the smoke sensor sensitivity).

I've also created a control variable called `armSmoke` to arm/disarm the smoke sensor. The `smokeTriggered` variable is used to trigger the notification message to the Node-RED Dashboard just once.

```
// Smoke Threshold  
int smokeThres = 60;  
  
// Control Variables  
boolean armSmoke = false;  
boolean smokeTriggered = false;
```

The next variable is used to create a timer that keeps checking the smoke value every few milliseconds.

```
long lastSmokeCheck = 0;
```

setup

In the `setup()` function, you need to set the LED and buzzer as OUTPUTs and the smoke sensor is set as an INPUT.

```
pinMode(smokeLED, OUTPUT);  
pinMode(buzzerPin, OUTPUT);  
  
pinMode(smokePin, INPUT);
```

reconnect

The `reconnect()` function publishes two messages when the ESP connects to the broker:

```
// Once connected, publish an announcement...
client.publish("home/office/esp1/smoke/status", "Not Armed");
client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
```

You also have to be subscribed to the following topic:

- `home/office/esp1/smoke`

So you can send commands from the Node-RED Dashboard to arm/disarm the smoke sensor.

```
client.subscribe("home/office/esp1/smoke");
```

callback

Inside the `callback()` function, you receive commands from the Node-RED Dashboard to actually arm/disarm the smoke sensor.

```

if(topic=="home/office/esp1/smoke") {
    Serial.print("SMOKE SENSOR STATUS CHANGE");
    if(messageTemp == "1"){
        Serial.print("Smoke Sensor Armed");
        client.publish("home/office/esp1/smoke/status", "Armed");
        client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
        armSmoke = true;
        smokeTriggered = false;
        digitalWrite(smokeLED, HIGH);
    }
    else if(messageTemp == "0"){
        Serial.print("Smoke Sensor Not Armed");
        client.publish("home/office/esp1/smoke/status", "Not Armed");
        client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
        armSmoke = false;
        smokeTriggered = false;
        digitalWrite(smokeLED, LOW);
    }
}

```

loop

In the `loop()` function, your ESP keeps checking, every 200 milliseconds, the current value of the smoke sensor:

```

if (now - lastSmokeCheck > 200) {
    lastSmokeCheck = now;
}

```

If the `smokeValue` is greater than the `smokeThres` value and if the sensor is armed, it sends a warning message saying “SMOKE DETECTED” to the Node-RED Dashboard.

```

if (now - lastSmokeCheck > 200) {
    lastSmokeCheck = now;
    int smokeValue = analogRead(smokePin);
    if (smokeValue > smokeThres && armSmoke) {
        Serial.print("Pin A0: ");
        Serial.println(smokeValue);
        tone(buzzerPin, 1000, 200);
        if(!smokeTriggered) {
            Serial.println("SMOKE DETECTED!!!");
            smokeTriggered = true;
            client.publish("home/office/esp1/smoke/notification", "SMOKE DETECTED");
        }
    }
}

```

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your ESP (after changing the credentials, RPi IP address and binary codes of your remote control).

Note #1: Make sure you select the **ESP board** in the **Tools** tab and the correct **COM** port.

Note #2: You can change the variable `smokeThres` to adjust the sensitivity of your gas sensor (or you might need to rotate the potentiometer on the back of the sensor).

All the resources for this project:

<https://rntlab.com/>

```

// Loading the required libraries
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <RCSwitch.h>
#include "DHT.h"

```

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

```

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

// Change the credentials below, so your ESP8266 connects to your router
const char* ssid = "YOUR_SSID";
const char* password = "YOUR_PASSWORD";

// Change the variable to your Raspberry Pi IP address, so it connects to your MQTT broker
const char* mqtt_server = "YOUR_RPi_IP_Address";

// Initializes the espClient. You have to change the espClient name if you have multiple ESPs running in your home
automation system
WiFiClient espClient;
PubSubClient client(espClient);

RCSwitch mySwitch = RCSwitch();

// Smoke Sensor
int smokePin = A0;
// DHT Sensor
const int DHTPin = 5;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

// Smoke Threshold
int smokeThres = 60;

// Control Variables
boolean armSmoke = false;
boolean smokeTriggered = false;

// Status LEDs
const int smokeLED = 13;

// Buzzer
const int buzzerPin = 14;

// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;
long lastSmokeCheck = 0;

```

```

// Don't change the function below. This functions connects your ESP8266 to your router
void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.print("WiFi connected - ESP IP address: ");
    Serial.println(WiFi.localIP());
}

// This functions is executed when some device publishes a message to a topic that your ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(String topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
}

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

// If a message is received on the topic home/office/esp1/desk, you check if the message is either 1 or 0. Turns the
// Desk outlet according to the message
if(topic=="home/office/esp1/desk"){
    Serial.print("Changing Desk light to ");
    if(messageTemp == "1"){
        mySwitch.send("00010101010001010101");
        Serial.print("On");
    }
}

```

```

else if(messageTemp == "0"){
    mySwitch.send("0001010101010001010100");
    Serial.print("Off");
}
}

if(topic=="home/office/esp1/workbench"){
    Serial.print("Changing Workbench light to ");
    if(messageTemp == "1"){
        mySwitch.send("0001010101010001010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("0001010101010001010100");
        Serial.print("Off");
    }
}

if(topic=="home/office/esp1/smoke"){
    Serial.print("SMOKE SENSOR STATUS CHANGE");
    if(messageTemp == "1"){
        Serial.print("Smoke Sensor Armed");
        client.publish("home/office/esp1/smoke/status", "Armed");
        client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
        armSmoke = true;
        smokeTriggered = false;
        digitalWrite(smokeLED, HIGH);
    }
    else if(messageTemp == "0"){
        Serial.print("Smoke Sensor Not Armed");
        client.publish("home/office/esp1/smoke/status", "Not Armed");
        client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
        armSmoke = false;
        smokeTriggered = false;
        digitalWrite(smokeLED, LOW);
    }
}

Serial.println();
}

// This functions reconnects your ESP8266 to your MQTT broker
// Change the function below if you want to subscribe to more topics with your ESP8266
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {

```

```

Serial.print("Attempting MQTT connection...");
// Attempt to connect
if (client.connect("ESP8266Client")) {
    Serial.println("connected");
    // Once connected, publish an announcement...
    client.publish("home/office/esp1/smoke/status", "Not Armed");
    client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
    // Subscribe or resubscribe to a topic
    // You can subscribe to more topics (to control more LEDs in this example)
    client.subscribe("home/office/esp1/desk");
    client.subscribe("home/office/esp1/workbench");
    client.subscribe("home/office/esp1/smoke");
} else {
    Serial.print("failed, rc=");
    Serial.print(client.state());
    Serial.println(" try again in 5 seconds");
    // Wait 5 seconds before retrying
    delay(5000);
}
}

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

    pinMode(smokePin, INPUT);

    dht.begin();

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

    mySwitch.enableTransmit(16);

    // SET YOUR PULSE LENGTH
    mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);
}

```

```

// SET YOUR PROTOCOL (default is 1, will work for most outlets)
mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);

// Set number of transmission repetitions.
mySwitch.setRepeatTransmit(15);
}

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

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

now = millis();
// Publishes new temperature and humidity every 30 seconds
if (now - lastMeasure > 30000) {
    lastMeasure = now;
    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    float h = dht.readHumidity();
    // Read temperature as Celsius (the default)
    float t = dht.readTemperature();
    // Read temperature as Fahrenheit (isFahrenheit = true)
    float f = dht.readTemperature(true);

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

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

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

```

```

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

// Publishes Temperature and Humidity values
client.publish("home/office/esp1/temperature", temperatureTemp);
client.publish("home/office/esp1/humidity", humidityTemp);

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

// Checks smoke
if (now - lastSmokeCheck > 200) {
    lastSmokeCheck = now;
    int smokeValue = analogRead(smokePin);
    if (smokeValue > smokeThres && armSmoke){
        Serial.print("Pin A0: ");
        Serial.println(smokeValue);
        tone(buzzerPin, 1000, 200);
        if(!smokeTriggered){
            Serial.println("SMOKE DETECTED!!!");
            smokeTriggered = true;
            client.publish("home/office/esp1/smoke/notification", "SMOKE DETECTED");
        }
    }
}
}
}

```

DOWNLOAD SOURCE CODE

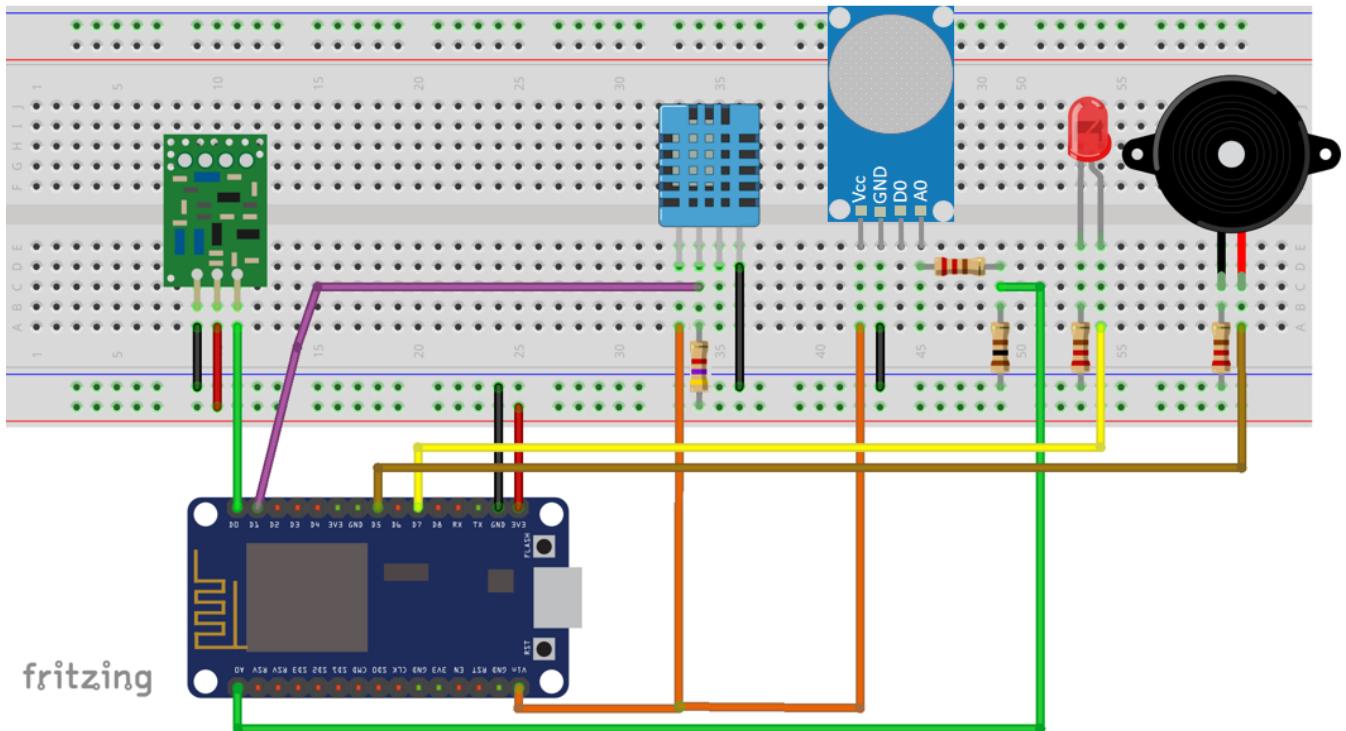
https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_esp8266_smoke.ino

Schematics

To complete this Unit you need:

- 1x ESP8266 12E
- 1x 433MHz Transmitter
- 1x Remote Controlled Outlets that operate at 433MHz
- 1x DHT11 Sensor
- 1x 4700 Ohm Resistor
- 1x MQ-2 Gas Sensor
- 1x Buzzer
- 1x LED
- 3x 220 Ohm Resistors
- 1x 100 Ohm Resistor

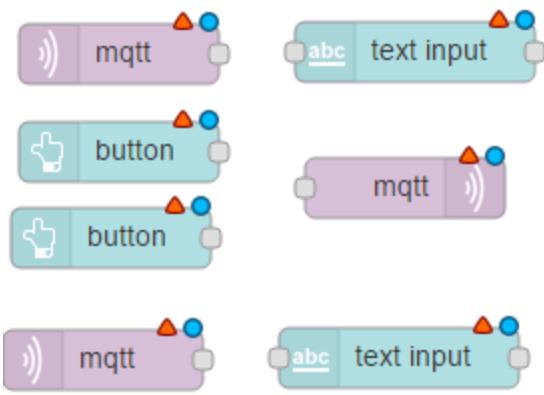
Important: The gas sensor requires 5V to operate properly, so make sure you use the Vin pin from your ESP8266 that outputs 5V.



Creating the Flow

The flow is very easy and similar to the previous Unit. Here's the 11 steps that you need to follow:

1 – Drag 6 Nodes



2 – MQTT in subscribes to status

Edit mqtt in node

Cancel Done

Server	localhost:1883	
Topic	home/office/esp1/smoke/status	
QoS	2	
Name	Name	

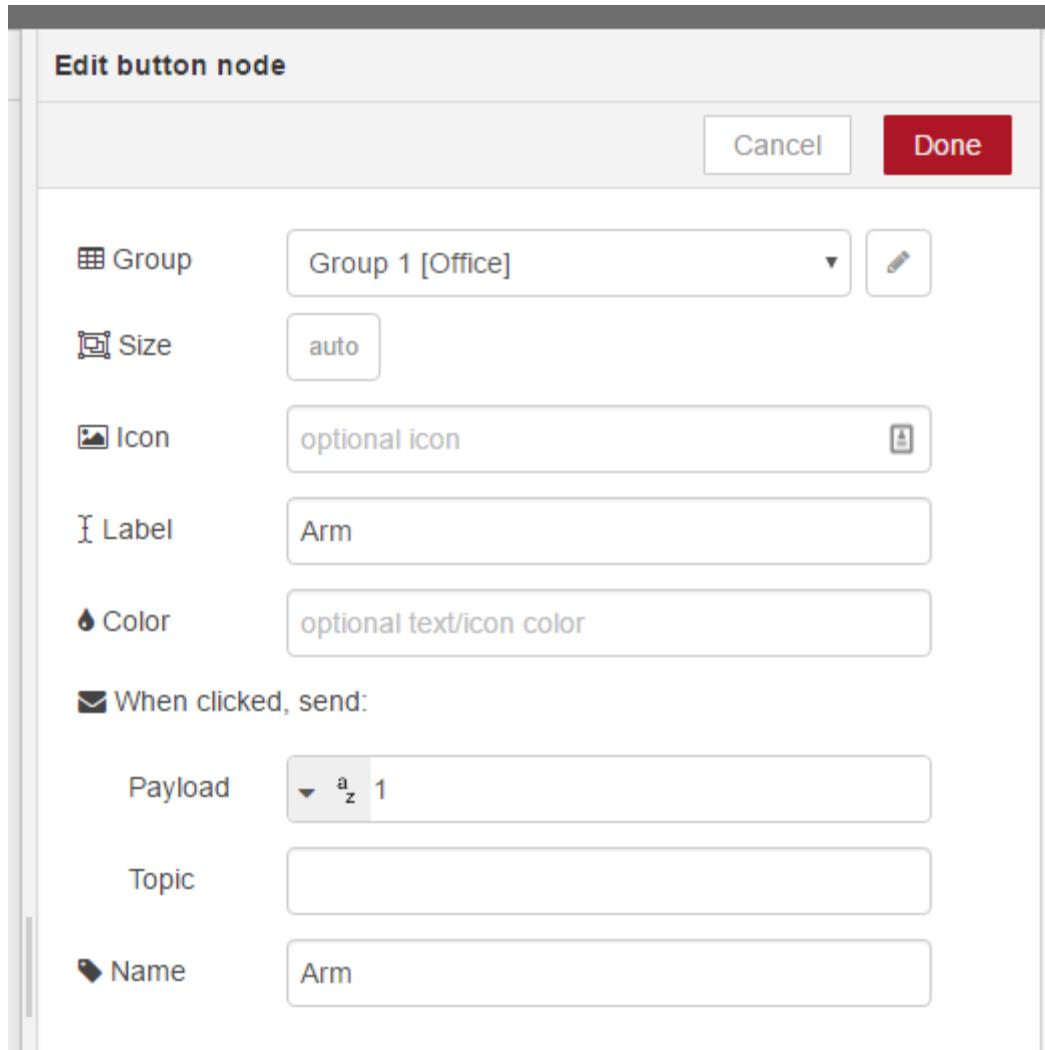
3 – Displays current status

Edit text input node

Cancel Done

Group	Group 1 [Office]	
Size	auto	
Label	Your smoke sensor is:	
Mode	text input	Delay (ms) 300
When changed, send:		
Payload	Current value	
Topic		
Name	Your smoke sensor is:	

4 – Button to arm smoke sensor



5 – Button to disarm smoke sensor

Edit button node

Cancel Done

Group: Group 1 [Office]

Size: auto

Icon: optional icon

Label: Disarm

Color: optional text/icon color

When clicked, send:

Payload: ▾ 0

Topic:

Name: Disarm

6 – MQTT out sends arm or disarm command to ESP8266

Edit mqtt out node

Cancel Done

Server: localhost:1883

Topic: home/office/esp1/smoke

QoS: Retain:

Name: home/office/esp1/smoke

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

7 – MQTT in subscribes to notification

Edit mqtt in node

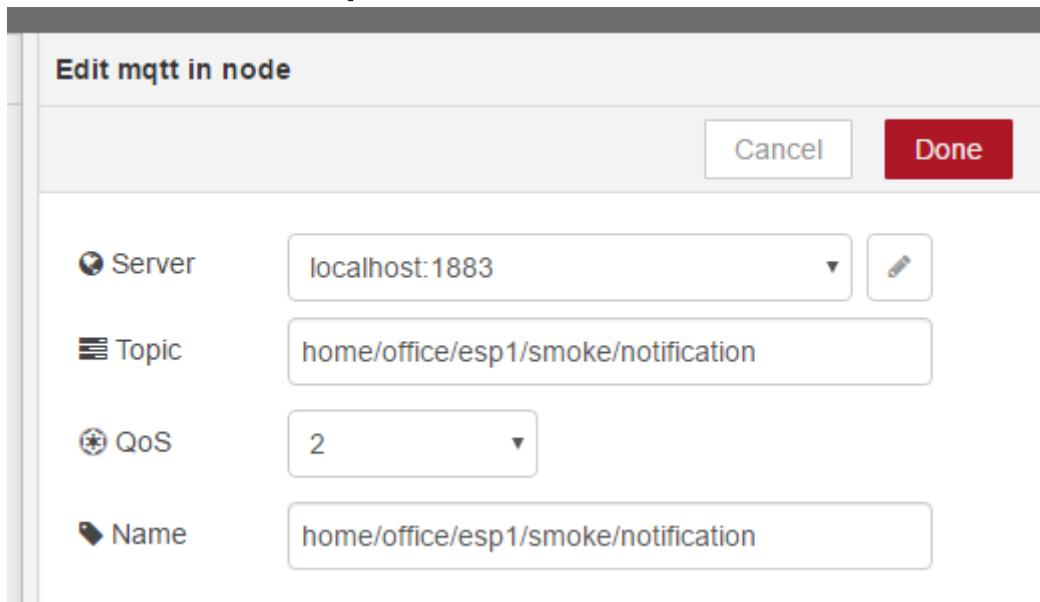
Cancel Done

Server: localhost:1883

Topic: home/office/esp1/smoke/notification

QoS: 2

Name: home/office/esp1/smoke/notification



8 – Displays notification status

Edit text input node

Cancel Done

Group: Group 1 [Office]

Size: auto

Label: Notifications

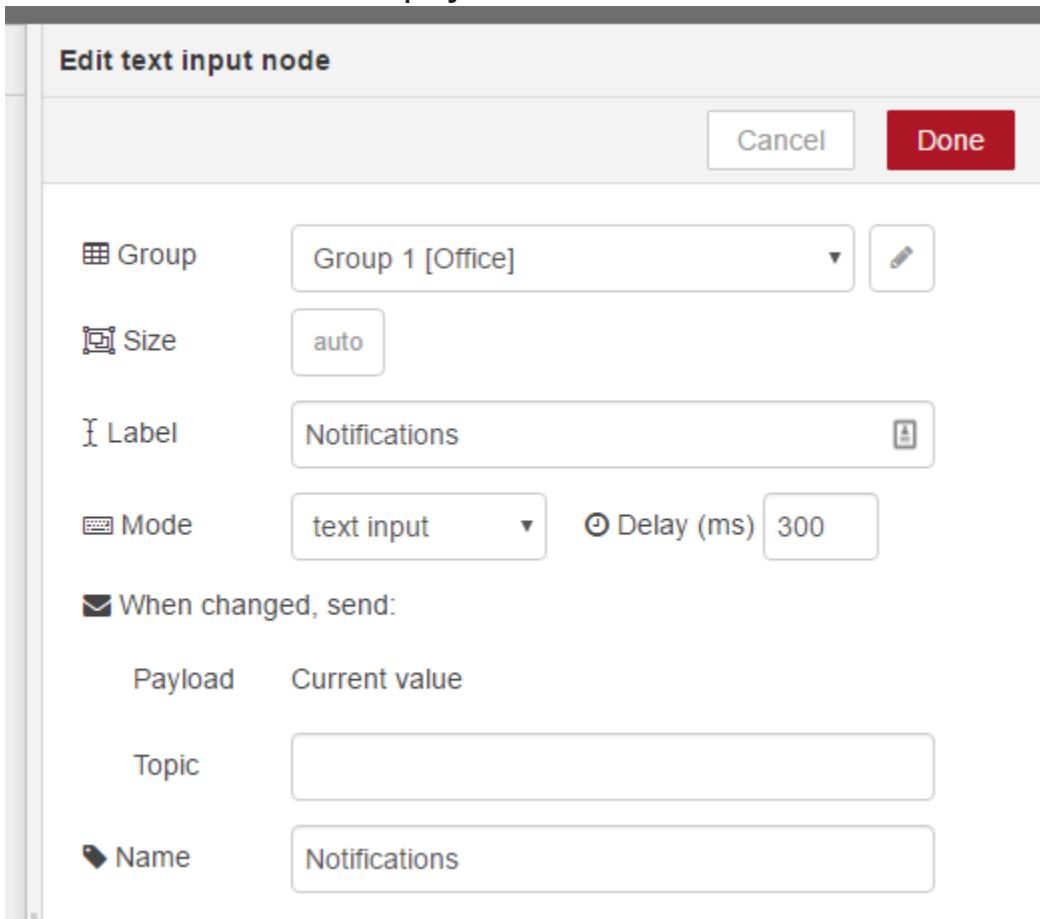
Mode: text input Delay (ms): 300

When changed, send:

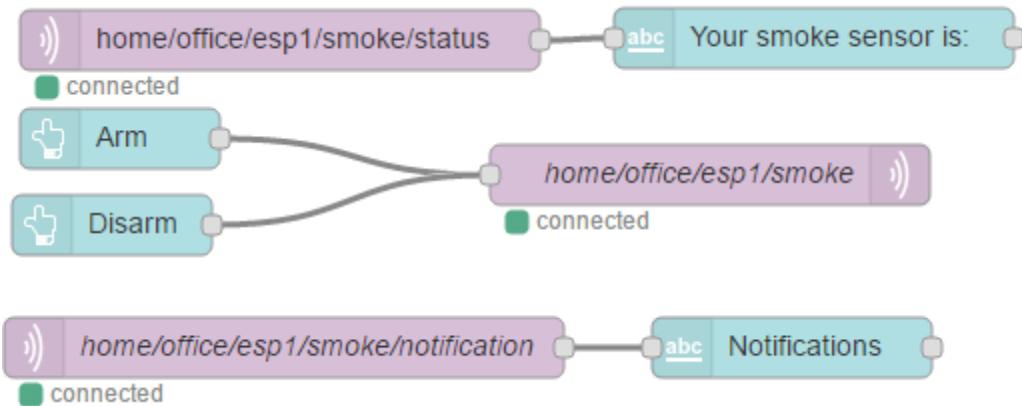
Payload: Current value

Topic:

Name: Notifications



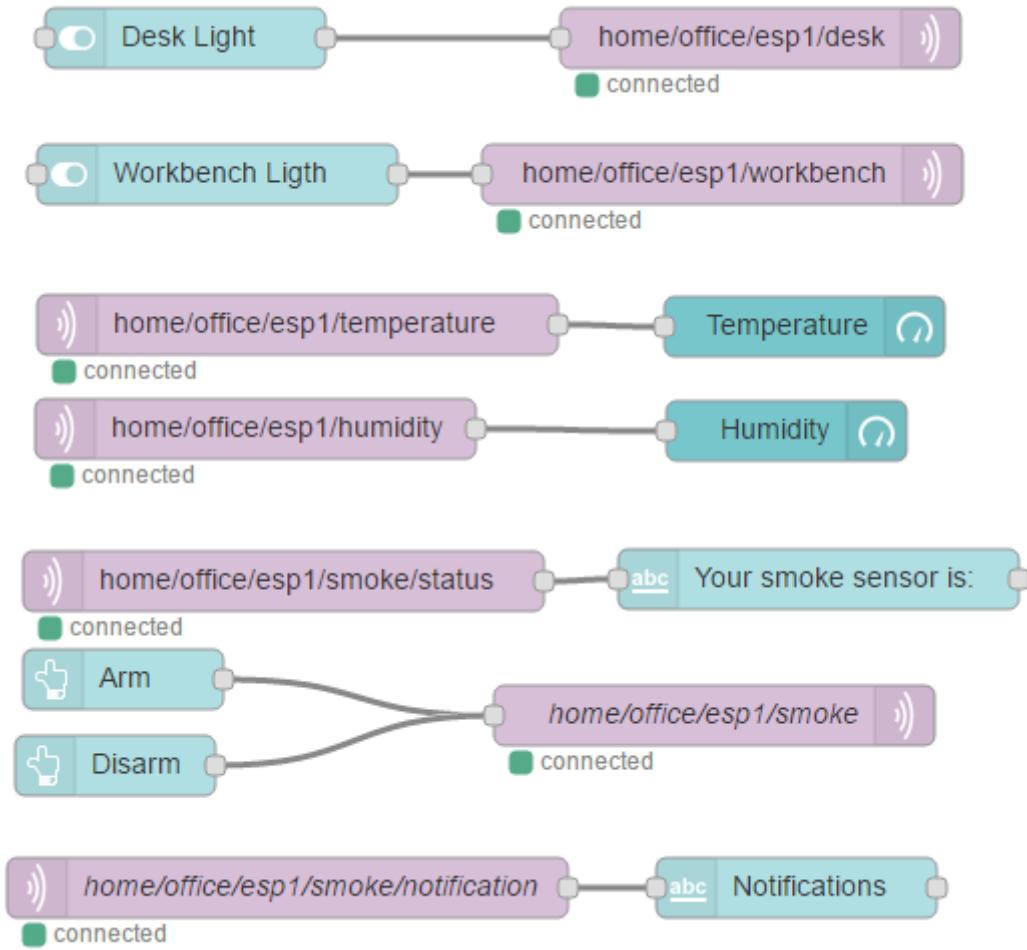
9 – Connecting nodes



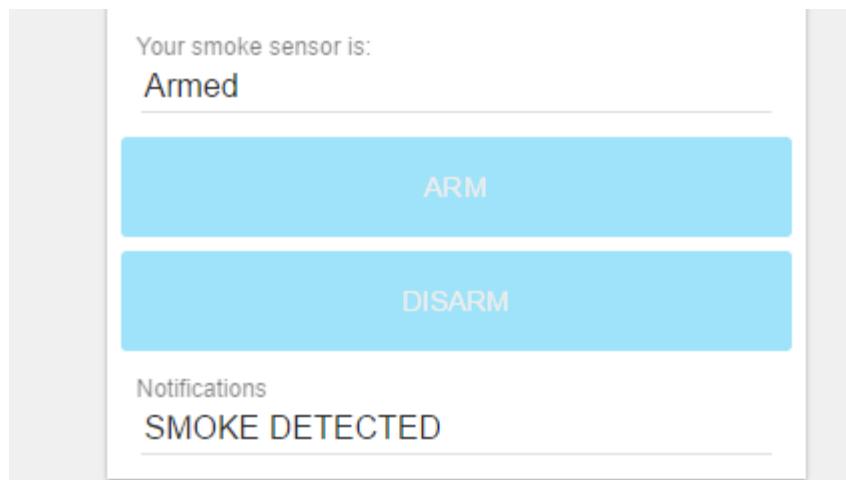
10 – Deploy your application



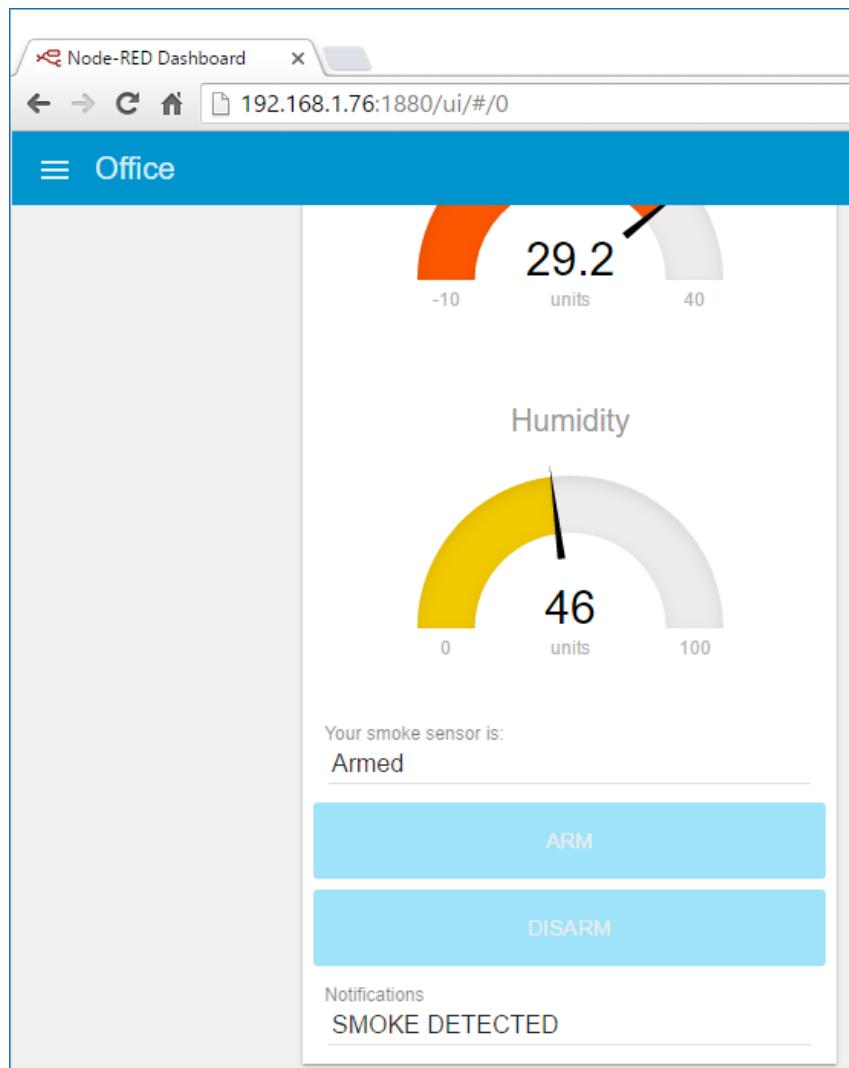
Final Flow



11 – Arm or Disarm smoke sensor and current status

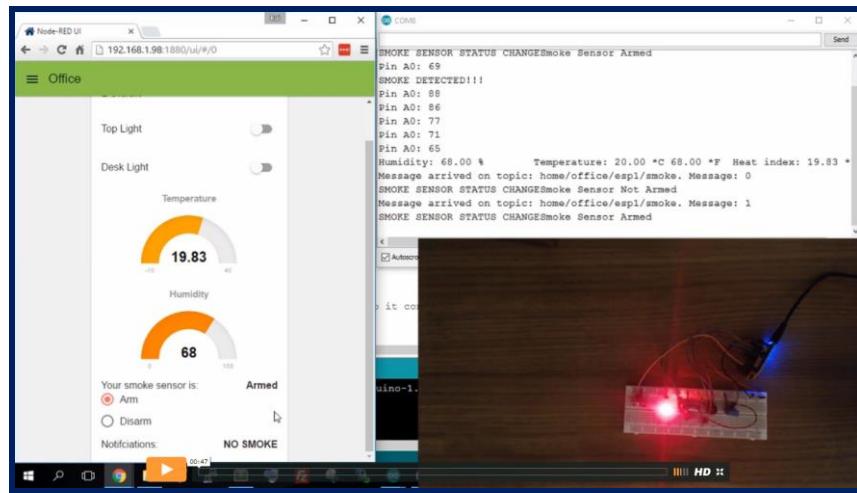


When you go to the Node-RED Dashboard page, here's what you should see:



Watch the Video Demonstration

If there's any flammable gas in your room, your gas sensor will detect it and you'll be notified with a message (you can arm/disarm the sensor). Watch this video to see it in action:



Video # 16 - <https://rntlab.com/28hasvideos>

Unit 3 - Motion Detector with Email Notification

(You should keep the sketch, circuit and nodes from the previous Units and add this new sensor to your project)

This Unit shows you how to detect motion with an ESP8266 and how to send an email notification.

Throughout this Unit you're going to use the PIR Motion sensor. You can learn more about this sensor in my [blog post](#).



Understanding How the Sketch Works

You can open the complete sketch in a new window here to follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_esp8266_final.ino

Preparing the variables

Assign the D2 (GPIO 4) to the `motionSensor` variable.

```
const int motionSensor = 4;
```

I've also created a control variable called `armMotion` to arm/disarm the motion sensor. The `motionTriggered` control variable is used to trigger the notification message to the Node-RED Dashboard just once.

```
boolean armMotion = false;
boolean motionTriggered = false;
```

setup

In the `setup()` function, you need to set the LED as an OUTPUT and the motion sensor is set as an INTERRUPT that is activated in RISING mode.

```
pinMode(motionLED, OUTPUT);

attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);
```

reconnect

The `reconnect()` function publishes two messages when the ESP connects to the broker:

```
client.publish("home/office/esp1/motion/status", "Not Armed");

client.publish("home/office/esp1/motion/notification", "NO MOTION");
```

You also have to be subscribed to the following topic:

- `home/office/esp1/motion`

So you can send commands from the Node-RED Dashboard to arm/disarm the motion sensor.

```
client.subscribe("home/office/esp1/motion");
```

detectsMovement

The `detectsMovement()` function sends a message “MOTION DETECTED” to the Node-RED when the sensor is armed and triggered. That message is displayed in the Node-RED Dashboard and you’ll also receive an email notification.

```
void detectsMovement() {
    if (armMotion && !motionTriggered) {
        Serial.println("MOTION DETECTED!!!!");
        motionTriggered = true;
        client.publish("home/office/esp1/motion/notification", "MOTION DETECTED");
    }
}
```

callback

Inside the `callback()` function, you receive commands from the Node-RED Dashboard to actually arm/disarm the motion sensor. It also changes the LED to on and off to indicated the current status of the sensor.

```
if(topic=="home/office/esp1/motion"){
    Serial.print("MOTION SENSOR STATUS CHANGE");
    if(messageTemp == "1"){
        Serial.print("Motion Sensor Armed");
        client.publish("home/office/esp1/motion/status", "Armed");
        client.publish("home/office/esp1/motion/notification", "NO MOTION");
        armMotion = true;
        motionTriggered = false;
        digitalWrite(motionLED, HIGH);
    }
    else if(messageTemp == "0"){
        Serial.print("Motion Sensor Not Armed");
        client.publish("home/office/esp1/motion/status", "Not Armed");
        client.publish("home/office/esp1/motion/notification", "NO MOTION");
        armMotion=false;
        motionTriggered = false;
        digitalWrite(motionLED, LOW);
    }
}
```

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your ESP (after changing the credentials, RPi IP address and binary codes of your remote control).

Note: Make sure you select the ESP board in the Tools tab and the correct COM port.

```
*****  
  
All the resources for this project:  
https://rntlab.com/  
  
*****/  
  
// Loading the required libraries  
#include <ESP8266WiFi.h>  
#include <PubSubClient.h>  
#include <RCSwitch.h>  
#include "DHT.h"  
  
// Uncomment one of the lines below for whatever DHT sensor type you're using!  
#define DHTTYPE DHT11 // DHT 11  
//#define DHTTYPE DHT21 // DHT 21 (AM2301)  
//#define DHTTYPE DHT22 // DHT 22 (AM2302), AM2321  
  
// Change the credentials below, so your ESP8266 connects to your router  
const char* ssid = "YOUR_SSID";  
const char* password = "YOUR_PASSWORD";  
  
// Change the variable to your Raspberry Pi IP address, so it connects to your MQTT broker  
const char* mqtt_server = "YOUR_RPi_IP_Address";  
  
// Initializes the espClient. You should change the espClient name if you have multiple ESPs running in your home  
automation system  
WiFiClient espClient;  
PubSubClient client(espClient);  
  
RCSwitch mySwitch = RCSwitch();  
  
// Smoke Sensor  
int smokePin = A0;
```

```

// DHT Sensor
const int DHTPin = 5;

// Initialize DHT sensor.
DHT dht(DHTPin, DHTTYPE);

// Smoke Threshold
int smokeThres = 60;

// Control Variables
boolean armMotion = false;
boolean armSmoke = false;
boolean smokeTriggered = false;
boolean motionTriggered = false;

// PIR Motion Sensor
const int motionSensor = 4;

// Status LEDs
const int smokeLED = 13;
const int motionLED = 12;

// Buzzer
const int buzzerPin = 14;

// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;
long lastSmokeCheck = 0;

// Don't change the function below. This functions connects your ESP8266 to your router
void setup_wifi() {
    delay(10);
    // We start by connecting to a WiFi network
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
}

```

```

Serial.print("WiFi connected - ESP IP address: ");
Serial.println(WiFi.localIP());
}

// This functions is executed when some device publishes a message to a topic that your ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(String topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
}

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

// If a message is received on the topic home/office/esp1/desk, you check if the message is either 1 or 0. Turns the
// Desk outlet according to the message
if(topic=="home/office/esp1/desk"){
    Serial.print("Changing Desk light to ");
    if(messageTemp == "1"){
        mySwitch.send("0001010101000101010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("00010101010001010100");
        Serial.print("Off");
    }
}
if(topic=="home/office/esp1/workbench"){
    Serial.print("Changing Workbench light to ");
    if(messageTemp == "1"){
        mySwitch.send("0001010101010001010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("0001010101010001010100");
        Serial.print("Off");
    }
}

```

```

    }
}

if(topic=="home/office/esp1/smoke"){
    Serial.print("SMOKE SENSOR STATUS CHANGE");
    if(messageTemp == "1"){
        Serial.print("Smoke Sensor Armed");
        client.publish("home/office/esp1/smoke/status", "Armed");
        client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
        armSmoke = true;
        smokeTriggered = false;
        digitalWrite(smokeLED, HIGH);
    }
    else if(messageTemp == "0"){
        Serial.print("Smoke Sensor Not Armed");
        client.publish("home/office/esp1/smoke/status", "Not Armed");
        client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
        armSmoke = false;
        smokeTriggered = false;
        digitalWrite(smokeLED, LOW);
    }
}
if(topic=="home/office/esp1/motion"){
    Serial.print("MOTION SENSOR STATUS CHANGE");
    if(messageTemp == "1"){
        Serial.print("Motion Sensor Armed");
        client.publish("home/office/esp1/motion/status", "Armed");
        client.publish("home/office/esp1/motion/notification", "NO MOTION");
        armMotion = true;
        motionTriggered = false;
        digitalWrite(motionLED, HIGH);
    }
    else if(messageTemp == "0"){
        Serial.print("Motion Sensor Not Armed");
        client.publish("home/office/esp1/motion/status", "Not Armed");
        client.publish("home/office/esp1/motion/notification", "NO MOTION");
        armMotion=false;
        motionTriggered = false;
        digitalWrite(motionLED, LOW);
    }
}
Serial.println();
}

```

```

// This functions reconnects your ESP8266 to your MQTT broker
// Change the function below if you want to subscribe to more topics with your ESP8266
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        /*
         YOU MIGHT NEED TO CHANGE THIS LINE, IF YOU'RE HAVING PROBLEMS WITH MQTT
         MULTIPLE CONNECTIONS
        To change the ESP device ID, you will have to give a new name to the ESP8266.
        Here's how it looks:
        if (client.connect("ESP8266Client")) {
        You can do it like this:
        if (client.connect("ESP1_Office")) {
        Then, for the other ESP:
        if (client.connect("ESP2_Garage")) {
        That should solve your MQTT multiple connections problem
        */
        if (client.connect("ESP8266Client")) {
            Serial.println("connected");
            // Once connected, publish an announcement...
            client.publish("home/office/esp1/smoke/status", "Not Armed");
            client.publish("home/office/esp1/motion/status", "Not Armed");
            client.publish("home/office/esp1/smoke/notification", "NO SMOKE");
            client.publish("home/office/esp1/motion/notification", "NO MOTION");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("home/office/esp1/desk");
            client.subscribe("home/office/esp1/workbench");
            client.subscribe("home/office/esp1/smoke");
            client.subscribe("home/office/esp1/motion");

        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

```

```

// Checks motion
void detectsMovement() {
    if (armMotion && !motionTriggered) {
        Serial.println("MOTION DETECTED!!!");
        motionTriggered = true;
        client.publish("home/office/esp1/motion/notification", "MOTION DETECTED");
    }
}

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

    pinMode(smokePin, INPUT);
    attachInterrupt(digitalPinToInterrupt(motionSensor), detectsMovement, RISING);
    dht.begin();

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

    mySwitch.enableTransmit(16);

    // SET YOUR PULSE LENGTH
    mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);

    // SET YOUR PROTOCOL (default is 1, will work for most outlets)
    mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);

    // Set number of transmission repetitions.
    mySwitch.setRepeatTransmit(15);
}

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

    if (!client.connected()) {

```

```

reconnect();
}

if(!client.loop())
    client.connect("ESP8266Client");

now = millis();

// Publishes new temperature and humidity every 30 seconds
if (now - lastMeasure > 30000) {
    lastMeasure = now;

    // Sensor readings may also be up to 2 seconds 'old' (its a very slow sensor)
    float h = dht.readHumidity();

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

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

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

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

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

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

    // Publishes Temperature and Humidity values
    client.publish("home/office/esp1/temperature", temperatureTemp);
    client.publish("home/office/esp1/humidity", humidityTemp);

    Serial.print("Humidity: ");
    Serial.print(h);
    Serial.print(" %\t Temperature: ");
    Serial.print(t);
}

```

```

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

// Checks smoke
if (now - lastSmokeCheck > 200) {
    lastSmokeCheck = now;
    int smokeValue = analogRead(smokePin);
    if (smokeValue > smokeThres && armSmoke){
        Serial.print("Pin A0: ");
        Serial.println(smokeValue);
        tone(buzzerPin, 1000, 200);
        if(!smokeTriggered){
            Serial.println("SMOKE DETECTED!!!");
            smokeTriggered = true;
            client.publish("home/office/esp1/smoke/notification", "SMOKE DETECTED");
        }
    }
}
}
}

```

DOWNLOAD SOURCE CODE

https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_esp8266_final.ino

Schematics

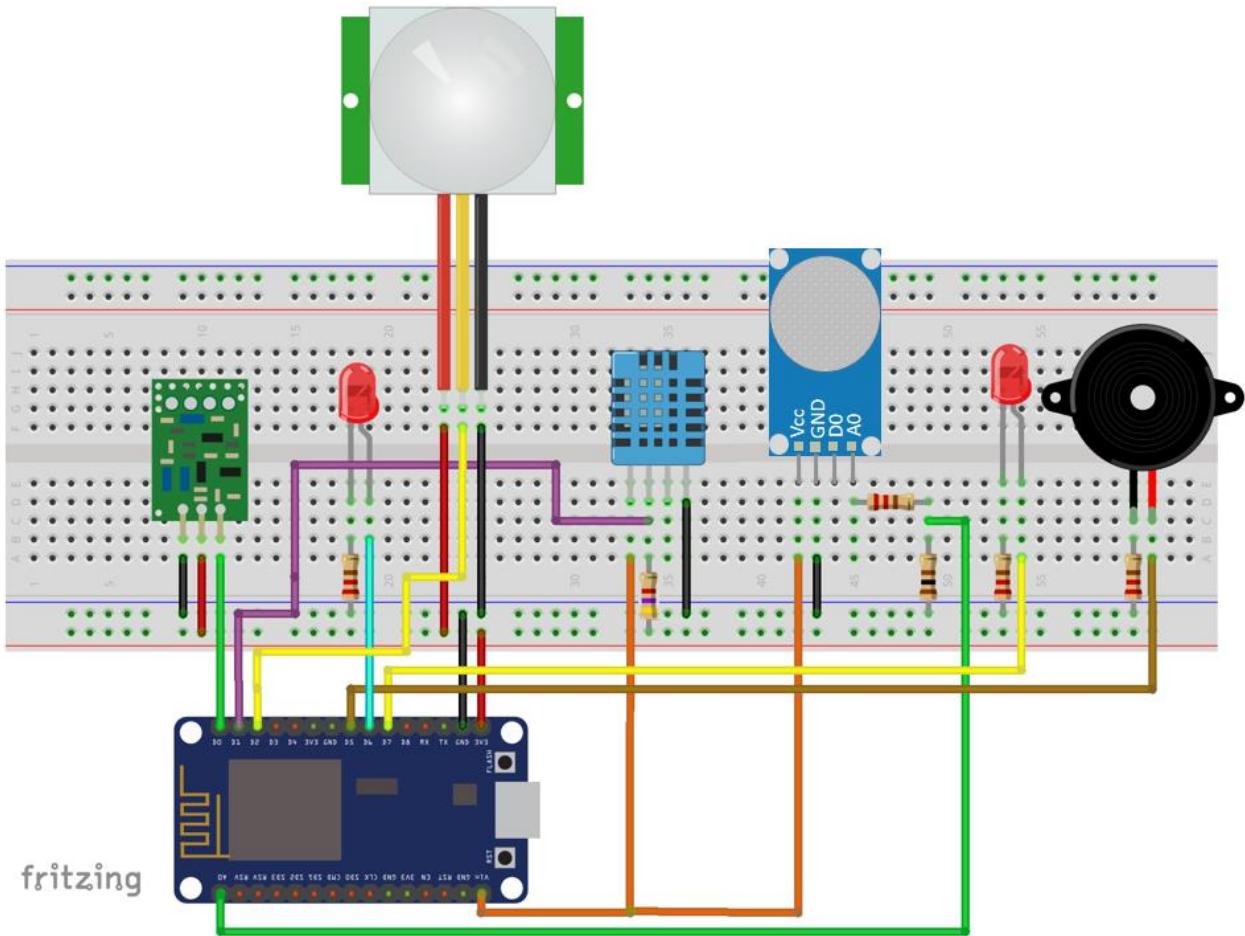
To complete this Unit you need:

- 1x ESP8266 12E
- 1x 433MHz Transmitter
- 1x Remote Controlled Outlets that operate at 433MHz
- 1x DHT11 Sensor
- 1x 4700 Ohm Resistor

- 1x MQ-2 Gas Sensor
- 1x Buzzer
- 1x LED
- 3x 220 Ohm Resistors
- 1x 100 Ohm Resistor
- 1x PIR Motion Sensor
- 1x LED
- 1x 220 Ohm Resistor

Important: The PIR motion sensor by default operates at 5V, you can modify the sensor to operate at 3.3V. [Click here to learn](#) how to change the voltage from 5V to 3.3V, otherwise you need to use the `Vin` pin from your ESP8266 that outputs 5V.

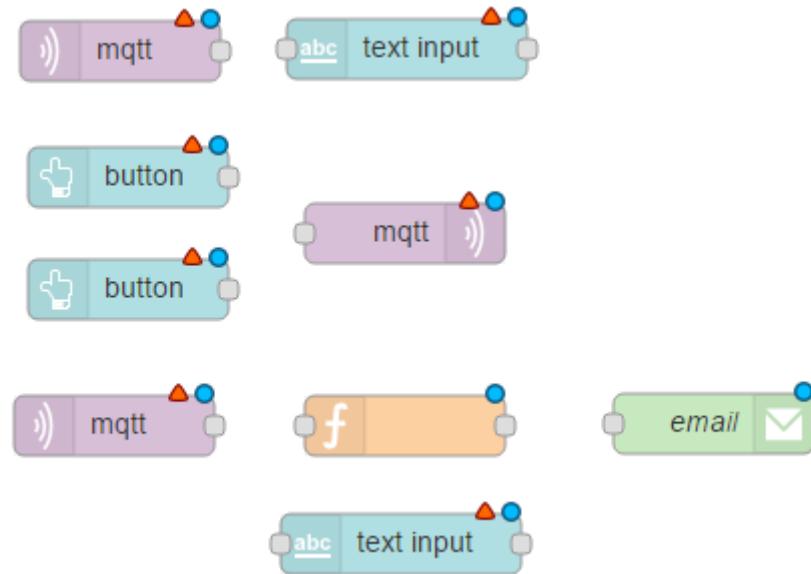
Here's the schematics powering the PIR motion sensor at 3.3V:



Creating the Flow

The flow is very easy and similar to the previous Unit. Here's the 13 steps that you need to follow:

1 – Drag 8 nodes



2 – MQTT in subscribed to status

Edit mqtt in node

Cancel
Done

	Server	localhost:1883	
	Topic	home/office/esp1/motion/status	
	QoS	2	
	Name	home/office/esp1/motion/status	

3 – Displays sensor status (arm or not armed)

Edit text input node

Cancel Done

Group: Group 1 [Office]

Size: auto

Label: Your motion sensor is:

Mode: text input Delay (ms) 300

When changed, send:

Payload: Current value

Topic:

Name: Your motion sensor is:

4 – Button to arm smoke sensor

Edit button node

Cancel Done

Group: Group 1 [Office]

Size: auto

Icon: optional icon

Label: Arm

Color: optional text/icon color

When clicked, send:

Payload:

Topic:

Name: Arm

5 – Button to disarm smoke sensor

Edit button node

Cancel Done

Group: Group 1 [Office]

Size: auto

Icon: optional icon

Label: Disarm

Color: optional text/icon color

When clicked, send:

Payload: 0

Topic:

Name: Disarm

6 – MQTT out sends arm or disarm command to ESP8266

Edit mqtt out node

Cancel Done

Server: localhost:1883

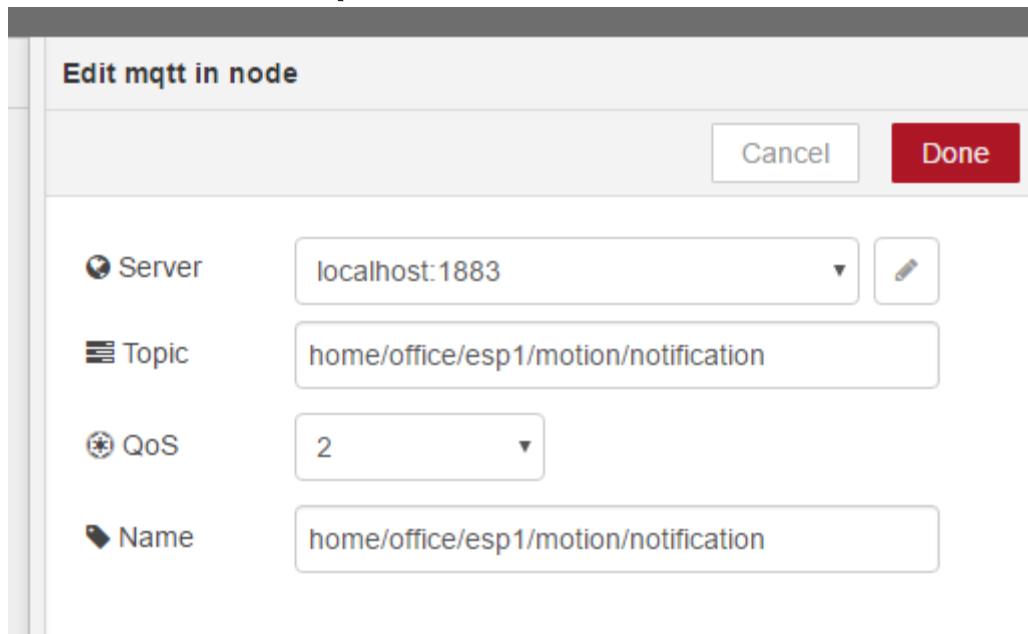
Topic: home/office/esp1/motion

QoS: Retain

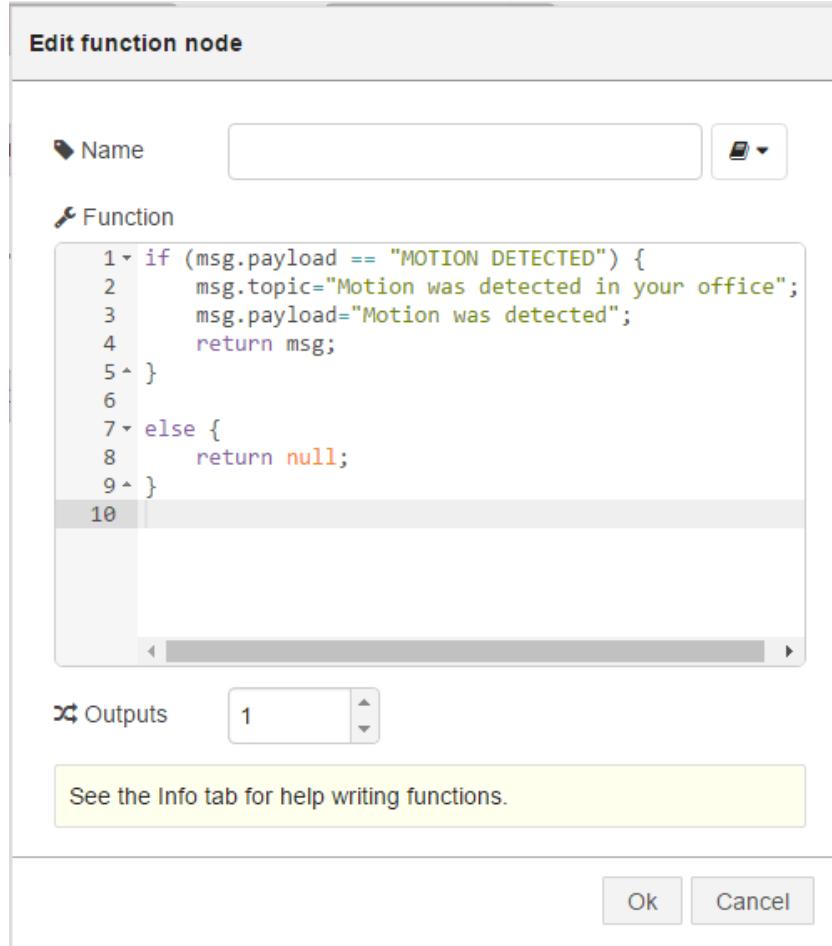
Name: home/office/esp1/motion

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

7 – MQTT in subscribed to notification



8 – Function node to create the Email title and Message



9 – Email notification settings

Edit e-mail node

To	hello@ruisantos.me
Server	smtp.gmail.com
Port	465
Userid	randomnerdtutorials
Password
Name	Email Notification

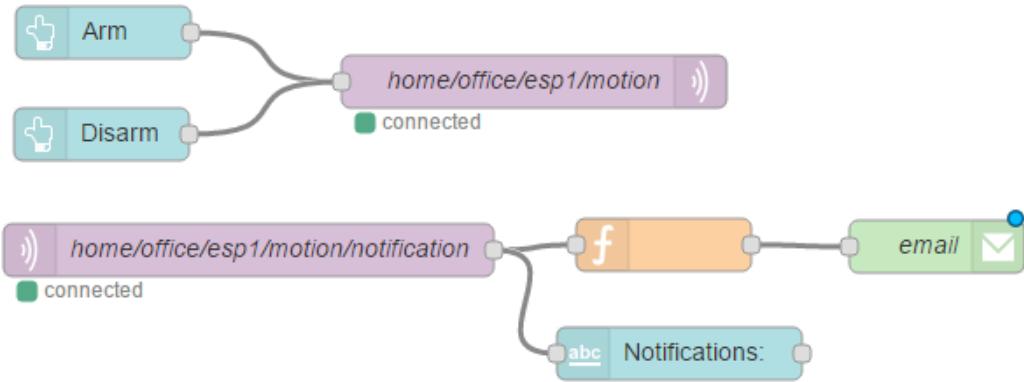
Ok Cancel

10 – Displays notifications from ESP8266

Edit text input node

Cancel Done	
Group	Group 1 [Office]
Size	auto
Label	Notifications:
Mode	text input
Delay (ms)	300
When changed, send:	
Payload	Current value
Topic	
Name	Notifications:

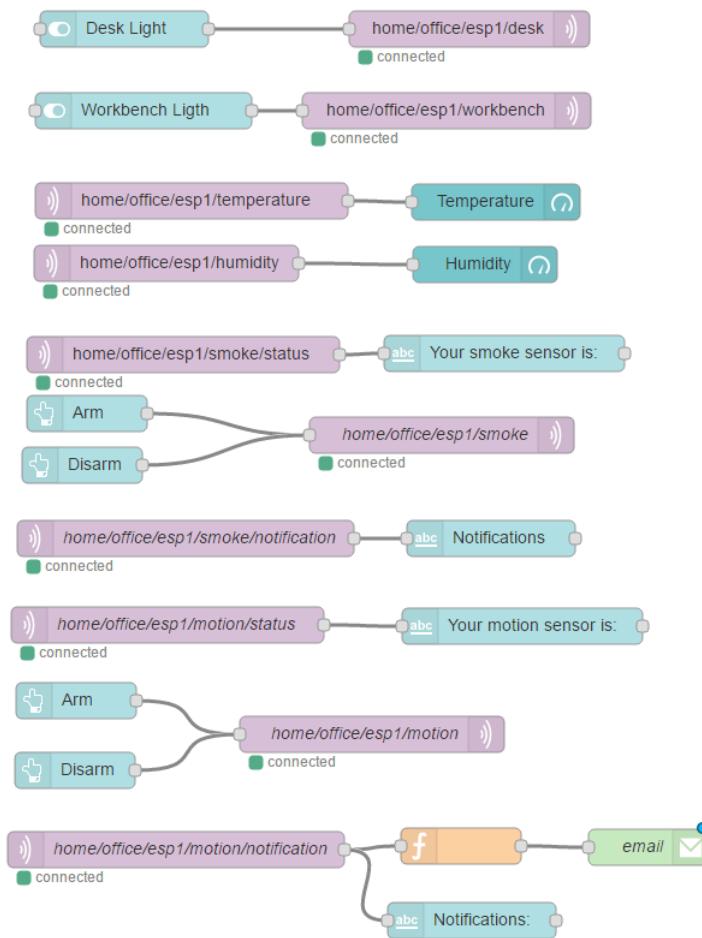
11 – Connecting nodes



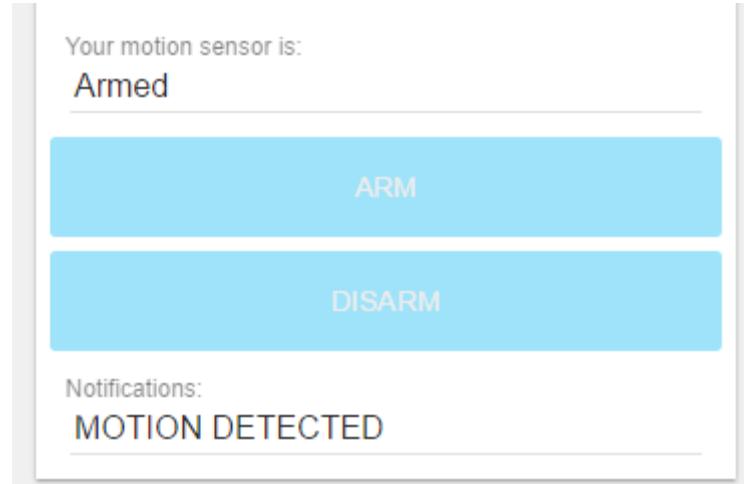
12 – Deploy your application



Final Flow



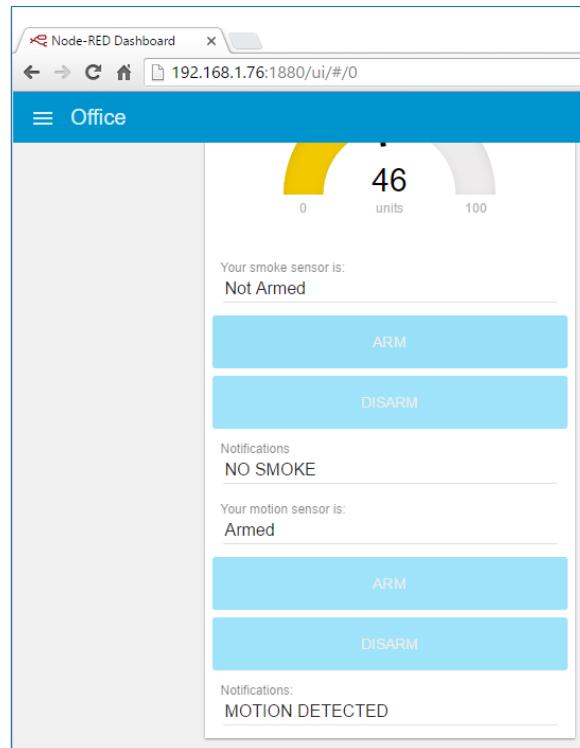
13 – Arm or Disarm smoke sensor and current status



Important: If you use Gmail to send emails, you need to activate the “Allowing Less Secure Apps” feature in your account by [following this tutorial](#).

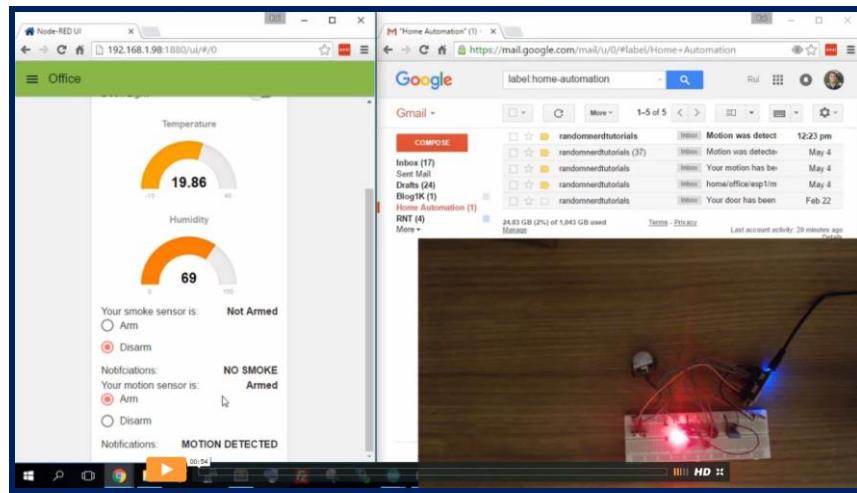
Note: If you use another email client, you can search for “your email client name” + SMTP and you should find the settings to configure the Node-RED **Email** node.

When you go to the Node-RED Dashboard page, here's what you should see:



Watch the Video Demonstration

Now, you can detect motion with your ESP8266. If your ESP detects any motion in your room, you'll receive an email notification (you can arm/disarm the sensor). Watch this video to see it in action:



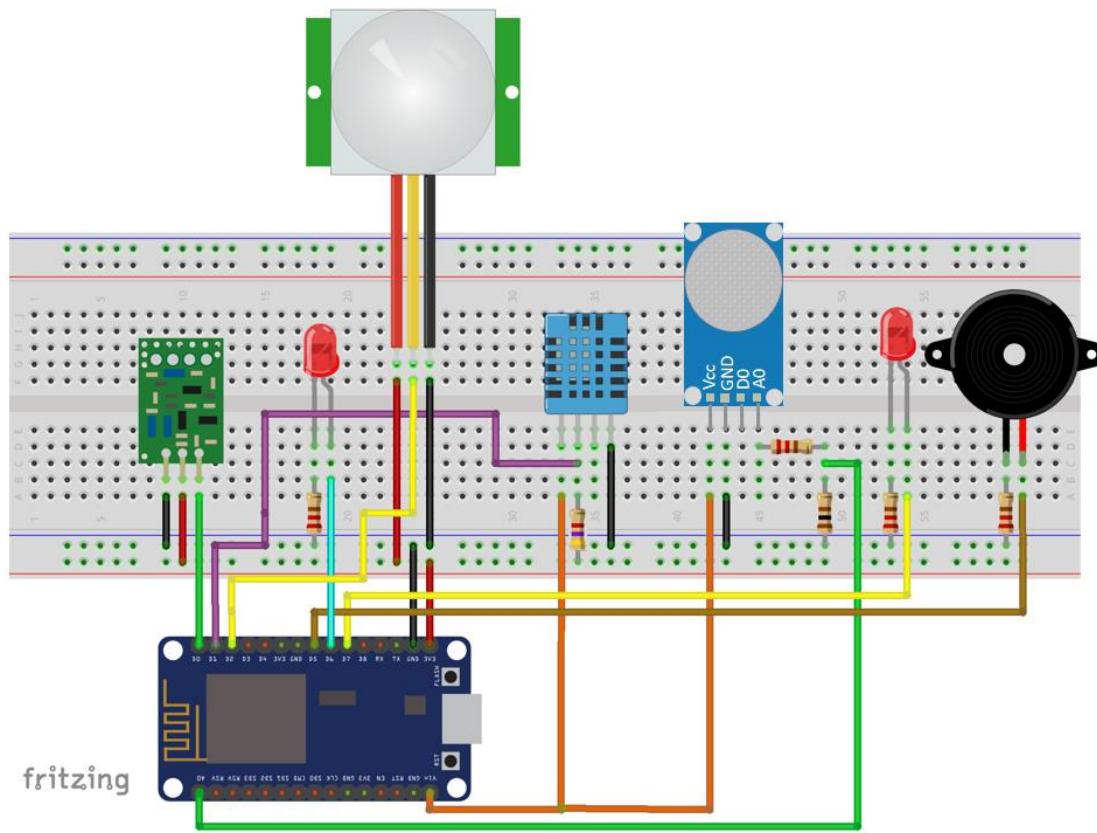
Video # 17 - <https://rntlab.com/28hasvideos>

Unit 4 - Storing Your Circuit in a Project Box Enclosure



Video # 18 - <https://rntlab.com/28hasvideos>

After completing the previous Units, you should have this circuit on a breadboard.



The ESP can:

1. Control outlets
2. Read the temperature and humidity
3. Detect smoke and motion

Circuits on a breadboard are temporary and if you're not careful with your circuit, a wire disconnects very easily and your project stops working.

In my opinion, I think you should take it a step further and make this circuit permanent on a stripboard.

Parts Required

You simply need to gather a couple of components. Here's the complete list:

- 1x ESP8266 12E
- 1x 433MHz Transmitter
- 1x Remote Controlled Outlets (433MHz)
- 1x DHT11 Sensor
- 1x 4700 Ohm Resistor
- 1x MQ-2 Gas Sensor
- 1x Buzzer
- 2x LED
- 2x LED Holders
- 4x 220 Ohm Resistors
- 1x 100 Ohm Resistor
- 1x PIR Motion Sensor
- 1x Stripboard

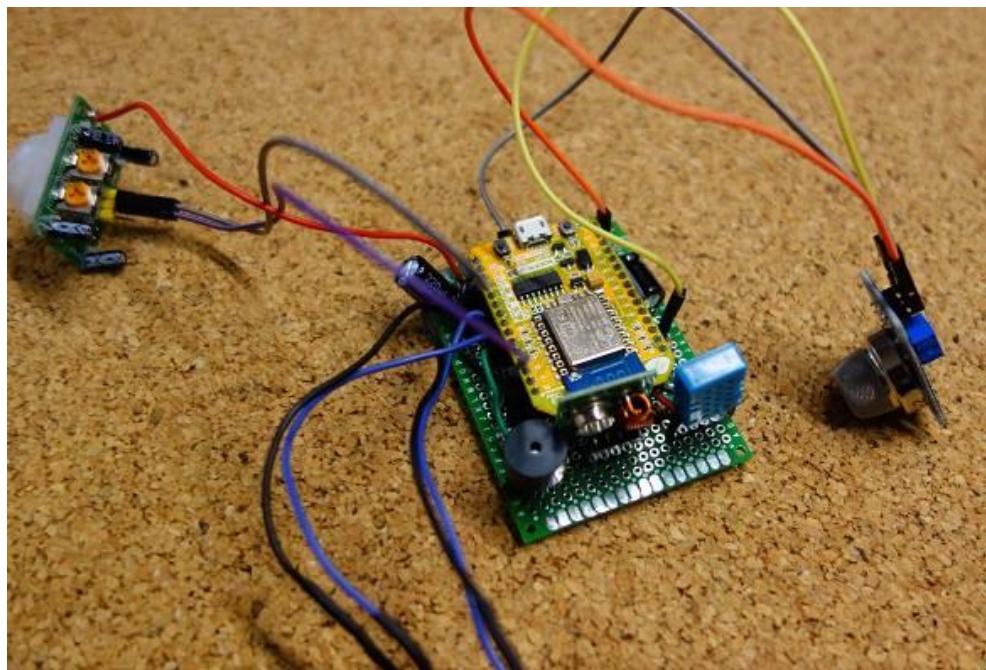
- 1x Project Box
- 2x 220 Microfarad Electrolytic Capacitors

Then, you solder each component in a stripboard according to the breadboard diagram shown earlier.

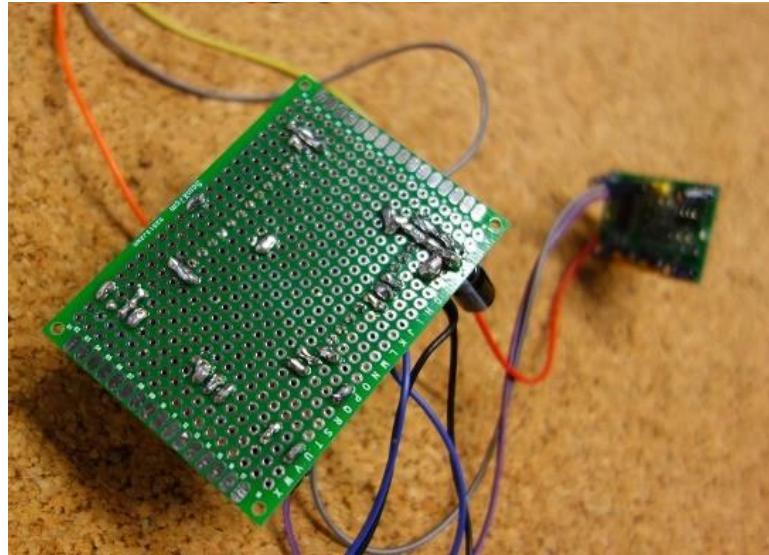
Note: The soldering skill is outside the course scope, so I will not show you how to solder.

Your Custom Circuit

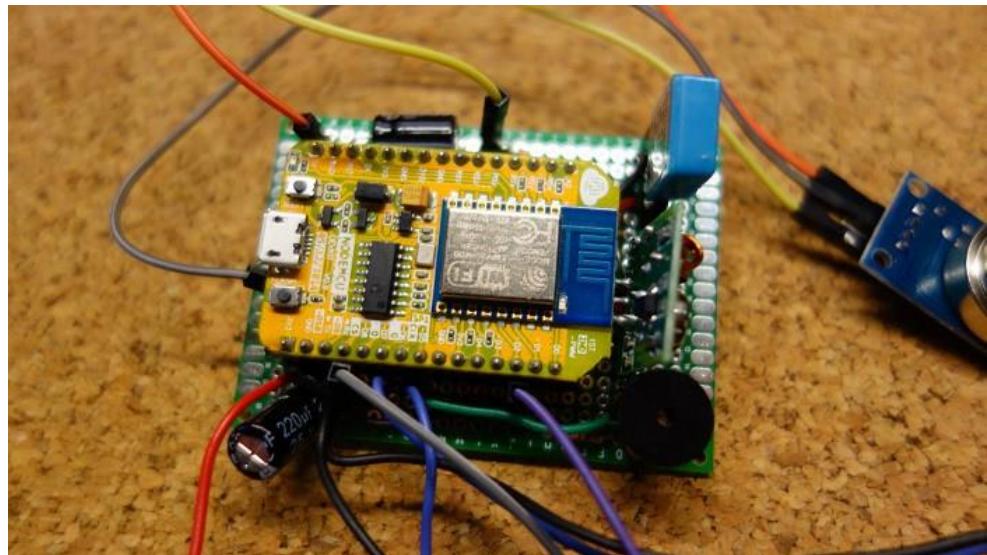
After soldering, you should have a very similar circuit.



Circuit viewed from the bottom:



Note: I've added two electrolytic capacitors in the 3.3V-GND rails and the other in the 5V-GND rails.



This prevents your ESP from having power issues when connected to a USB port (by the way, these are 220 Microfarad capacitors).

Project Box Enclosure

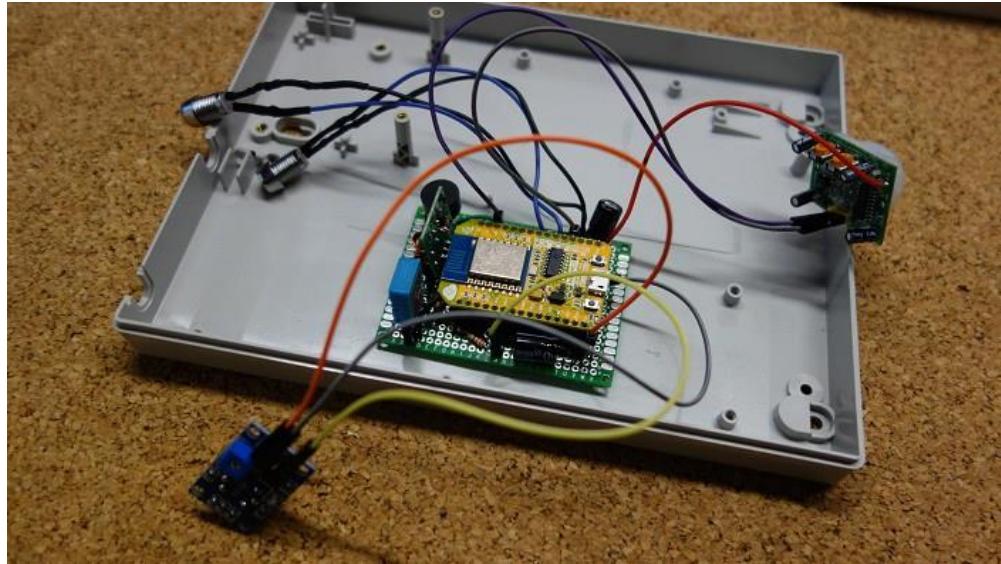
Finally, I've bought a project box enclosure to store my final circuit.



I've drilled 4 holes. Two for the LEDs, one for the motion sensor and the other for the smoke sensor.



My project enclosure is quite big, but you can get a smaller one. I bought this enclosure, because I also plan store my Raspberry Pi in that same enclosure.



The ESP is going to be powered through the Raspberry Pi USB port, however you can also use a 5V power adapter and plug your ESP directly to an outlet.



You might need to drill some extra holes in the enclosure to have some airflow.

You can assemble everything inside the project box, install the screws and it's done.



In the next Unit, you're going to see the final demonstration to quickly recap all the features and see the project in action.



Unit 5 - ESP8266 Final Demonstration

So, let's quickly recap what we have accomplished so far.

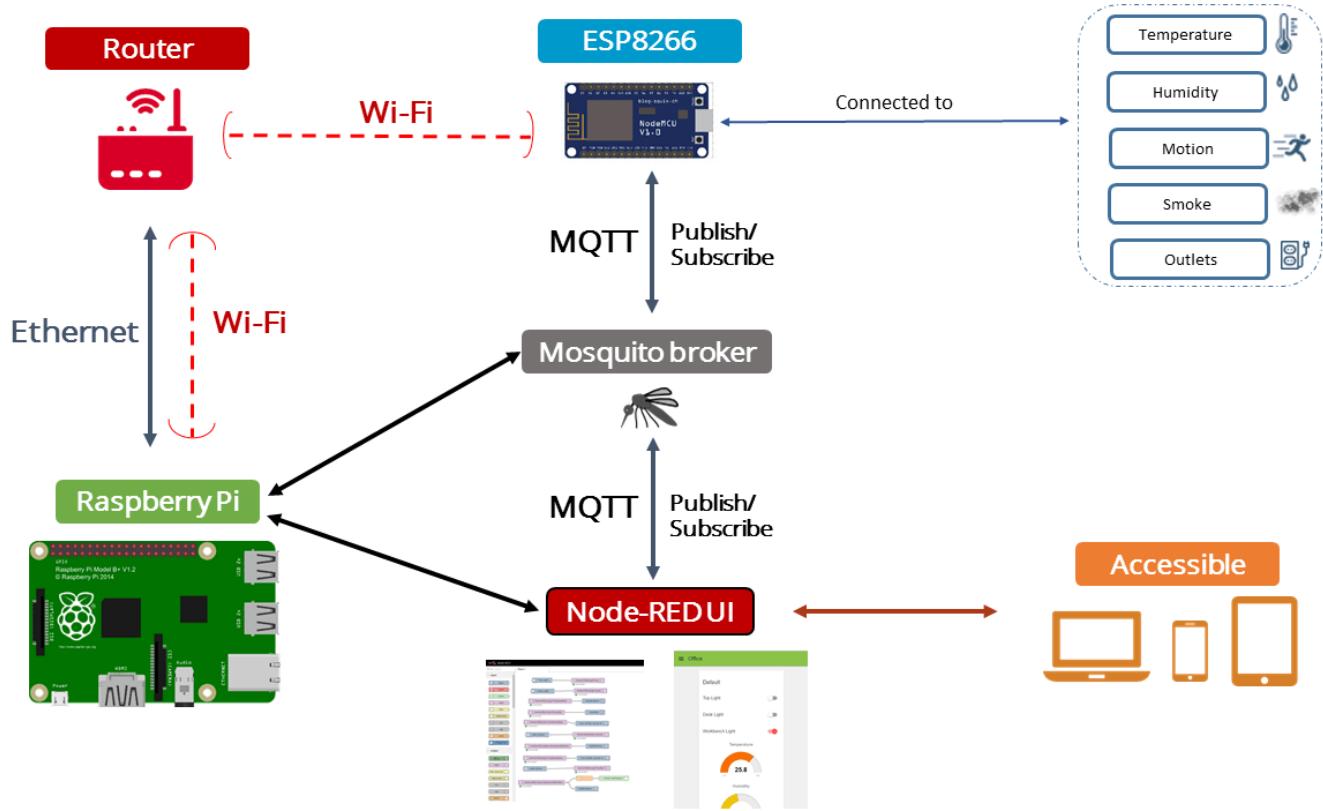
Inside the project box, we have a Raspberry Pi hosting the Mosquitto Broker and the Node-RED software.

We also have an ESP8266 being powered through the Raspberry Pi USB port.



Diagram

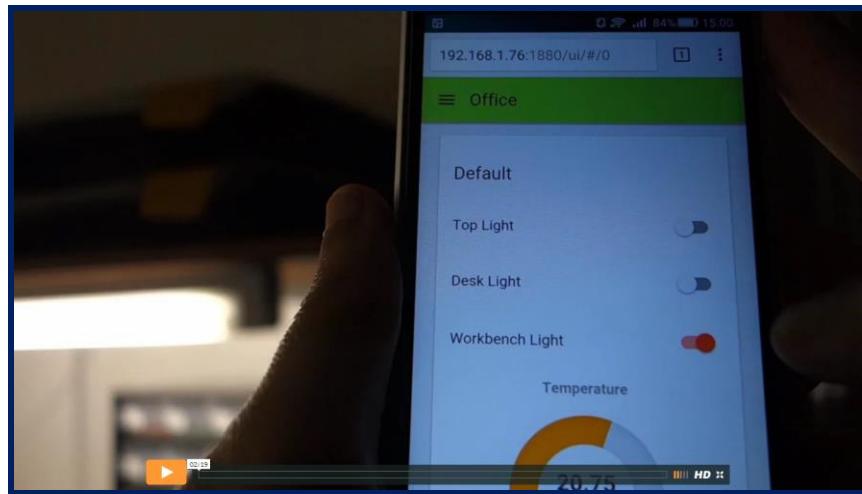
Here's a diagram that illustrates how everything is connected:



The ESP is able to:

1. Control outlets
2. Read the temperature and humidity
3. Check the room for smoke
4. Detect motion and send email notifications

Watch the ESP8266 Final Video Demonstration

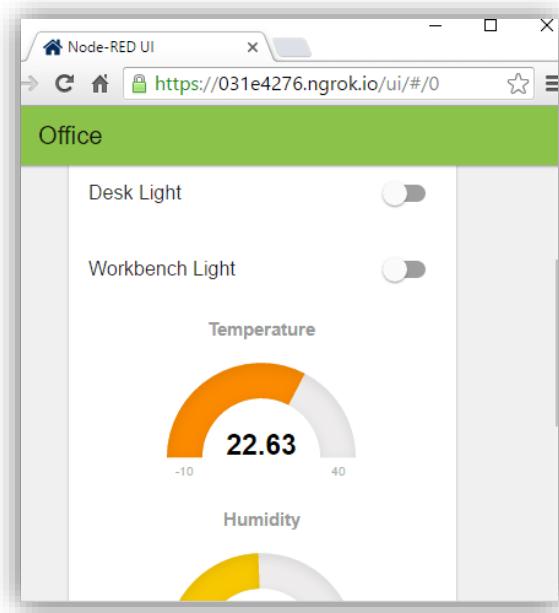


Video # 19 - <https://rntlab.com/28hasvideos>

This project can be extended to other rooms in your home and that's what I encourage you to do next.

Module 9

Accessing Node-RED Dashboard From Anywhere in the World



Unit 1 - Accessing Node-RED Dashboard From Anywhere (it's encrypted and password protected)

In this Unit you're going to make your Node-RED Dashboard accessible from anywhere in the world.

At this moment, the Node-RED Dashboard is only accessible when you are connected to your network.

Luckily, in just a few minutes you'll be able to control and monitor your home from anywhere.

Introducing ngrok

You are going to use a free service called **ngrok** to create an encrypted password protected tunnel to your Raspberry Pi.

Go to <https://ngrok.com> to create your account. Click the green “Sign up” button:

 Home Download Docs Product FAQ Login

Secure tunnels to localhost

“I want to expose a local server behind a NAT or firewall to the internet.”

[Download](#)

[Sign up](#)

Enter your details in the fields.

Sign up

Your Name

Your Email

Confirm Email

Password

After creating your account, login and go to the “Auth” tab to find Your Tunnel Authtoken.

Copy your unique Your Tunnel Authtoken to a safe place (you'll need them later in this Unit).

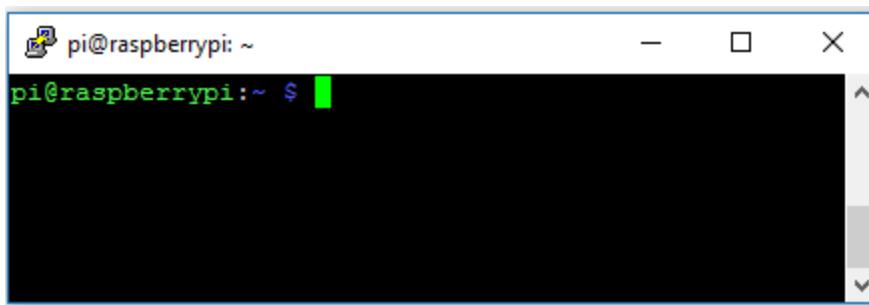
The screenshot shows the ngrok dashboard interface. At the top, there's a navigation bar with links: Dashboard (highlighted in orange), Download, Docs, Product, and FAQ. Below the navigation bar, there's a horizontal menu with links: Get Started, Status, Reserved, Auth (highlighted with a red box and a red arrow pointing to it), Team, Admin, and Billing. The main content area is titled "Your Tunnel Authtoken" and displays a text input field containing the token: 3V1MfHcNMD9rhBif8TRs_2whamY91tqX4. To the right of the input field is a "Copy" button with a clipboard icon. Below the input field, a note says "You only need to do this one time." At the bottom, there's a code input field containing the command: ./ngrok authtoken 3V1MfHcNMD9rhBif8TRs_2whamY91tqX4.

My Tunnel Authtoken is:

```
3V1MfHcNMD9rhBifzf8TRs_2whamY91tqX4
```

Installing ngrok in the RPi

Having an SSH communication established with your Raspberry Pi:



Send the following command to download ngrok:

```
pi@raspberry:~ $ sudo wget  
https://dl.ngrok.com/ngrok_2.0.19_linux_arm.zip
```

Unzip the ngrok software using:

```
pi@raspberry:~ $ unzip ngrok_2.0.19_linux_arm.zip
```

Running ngrok

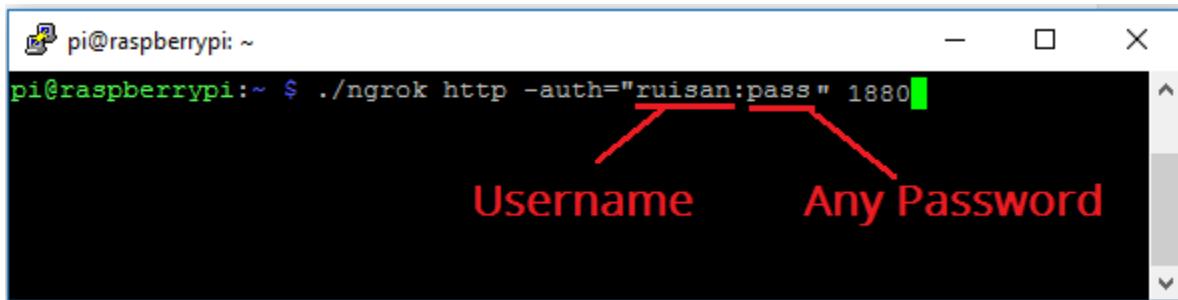
Enter the following command to authenticate your ngrok account (replace the red highlighted text with your own Tunnel Authtoken):

```
pi@raspberry:~ $ ./ngrok authtoken  
3V1MfHcNMD9rhBifzf8TRs_2whamY91tqX4
```

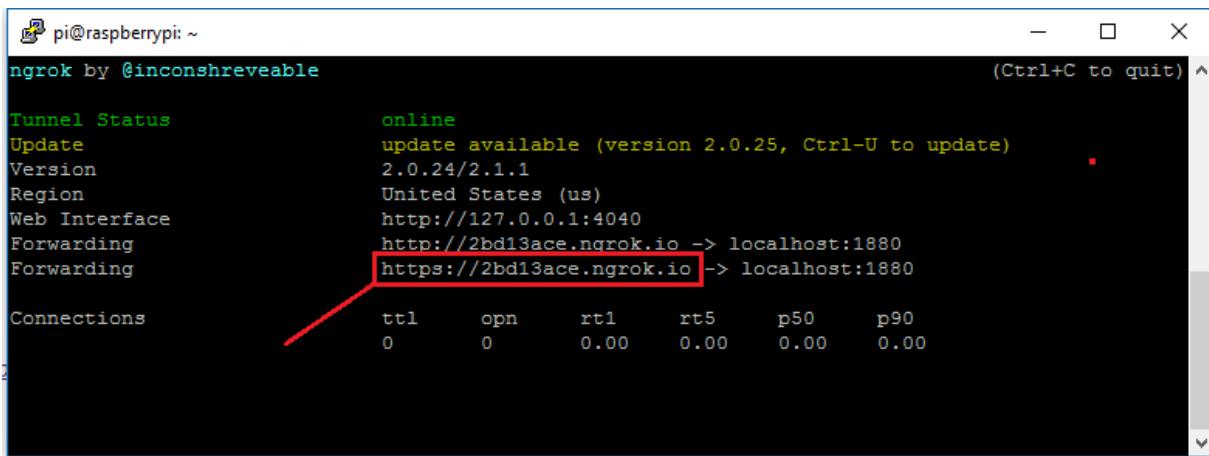
In your terminal window, enter the following command and replace the red text with your desired credentials:

```
pi@raspberrypi:~ $ ./ngrok http -auth="ruisan:pass" 1880
```

When you try to access the UI, you'll be asked to enter a username (**ruisan**) and a password (**pass**).



When you run the previous command, a new window that looks like this appears:



Copy your unique link (it is highlighted in the preceding figure):

```
https://2bd13ace.ngrok.io
```

Important: do not use the http link, because that link is not encrypted:

```
http://http://2bd13ace.ngrok.io
```

Important: you should always use the https link:

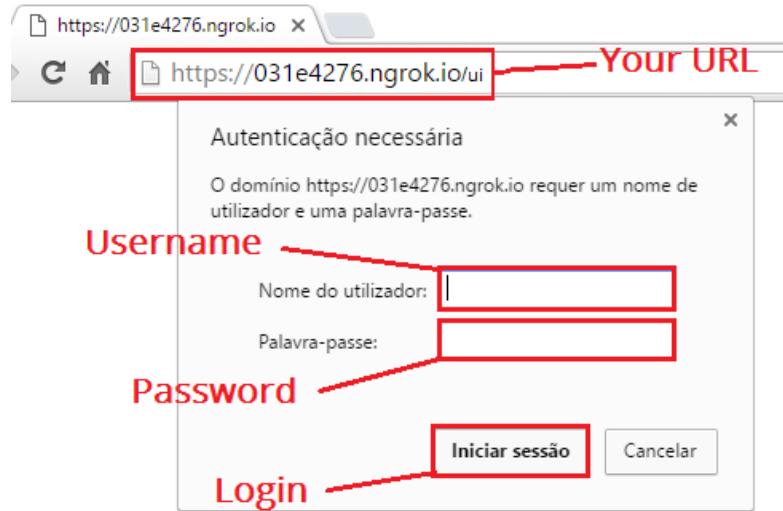
<https://2bd13ace.ngrok.io>

Accessing Your UI from Anywhere

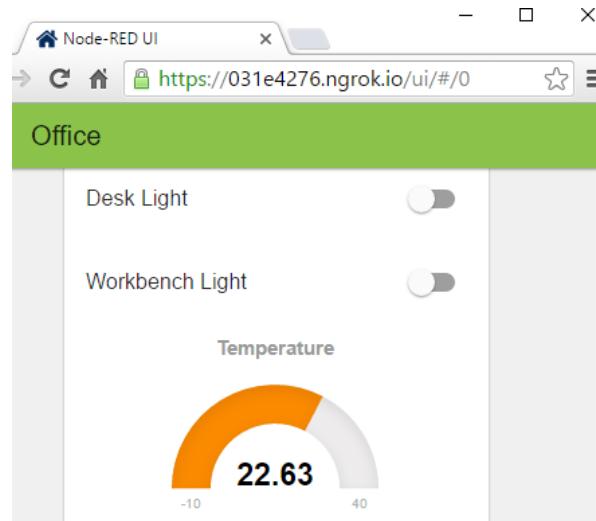
If everything runs smoothly, you can open the Node-RED Dashboard from any web browser in the world. By entering the following URL in any browser:

<https://2bd13ace.ngrok.io/ui>

You'll be asked to enter a username (**ruisan**) and password (**pass**):

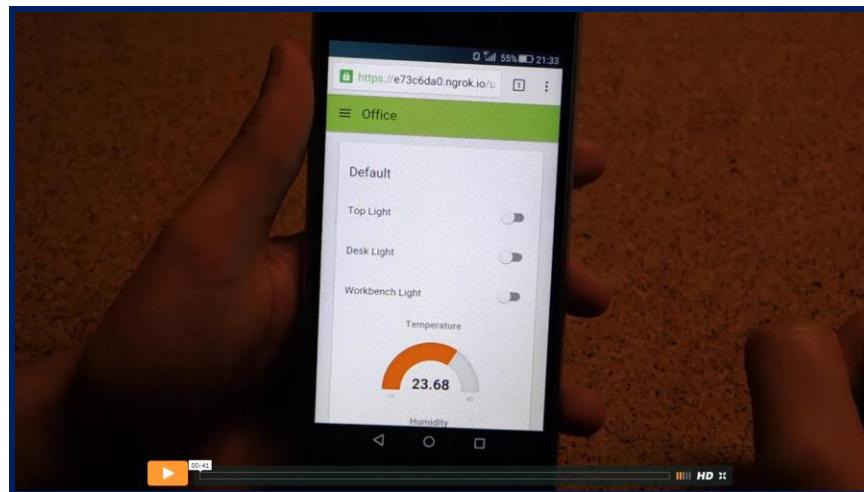


After entering the correct credentials for your tunnel, your Node-RED Dashboard loads:



Now, you have full control over your home!

Watch this video demonstration



Video # 20 - <https://rntlab.com/28hasvideos>

Important: You can close the SSH window, but you can't quit the ngrok software, otherwise your tunnel stops.

Note #1: you can reserve a custom subdomain name, so you'll always have the exact same link to access your UI (but that's a paid option in ngrok software).

Note #2: you'll be asked to enter your username and password to open your UI for the first time. If you want to "logout":

1. You'll have to clear the cache in your web browser
2. Or you can use the incognito window in your web browser, so it automatically logs off from the UI when you close the web browser

Unit 2 - Another Way of Making Node-RED Dashboard Accessible

If you've followed the previous Unit, you don't need to do anything else. You can skip/ignore this Unit.

I personally think that the method implemented in the previous Unit (using ngrok) is the best option to make Node-RED Dashboard accessible from anywhere.

However, I think it's important to share the most popular method of making the Raspberry Pi accessible from anywhere.

Port Forwarding Your RPi

Doing port forwarding is a bit tricky and it's different for everyone, because every router has a different configurations menu.

Gus from Pi My Life Up blog did an excellent job documenting how to do port forwarding with a Raspberry Pi.

He also created a video that goes throughout the whole process (you can watch the video tutorial below):

```
Raspberry Pi Port Forwarding & Dynamic DNS pi@raspberrypi: ~
noip2.c:1826:13: warning: variable 'x' set but not used [-Wunused-but-set-variable]
noip2.c: In function 'hosts':
noip2.c:1838:20: warning: variable 'y' set but not used [-Wunused-but-set-variable]
if [ ! -d /usr/local/bin ]; then mkdir -p /usr/local/bin;fi
if [ ! -d /usr/local/etc ]; then mkdir -p /usr/local/etc;fi
cp noip2 /usr/local/bin/noip2
/usr/local/bin/noip2 -C -c /tmp/no-ip2.conf

Auto configuration for Linux client of no-ip.com.

Please enter the login/email string for no-ip.com pimylifeup@outlook.com
Please enter the password for user 'pimylifeup@outlook.com' ****
Only one host [pimylifeup.ddns.net] is registered to this account.
It will be used.
Please enter an update interval:[30] 30
Do you wish to run something at successful update?[N] (y/N) n

New configuration file '/tmp/no-ip2.conf' created.

mv /tmp/no-ip2.conf /usr/local/etc/no-ip2.conf
rpi@pi:~$ 3:57 / 5:59 pi:/usr/local/src/noip-2.1.9-1#
```

Extra - <https://rntlab.com/28hasvideos>

You can go to this link for all the resources mentioned: <https://pimylifeup.com/raspberry-pi-port-forwarding> and follow the instructions in that tutorial.

Important: keep in mind that you want to expose port number 1880 (instead of port 80).

Note: There is a great website that shows the different configurations of port forwarding for each router. Go to this website <http://portforward.com> and search for your router model (or simply type in Google your “router name” + “port forwarding”).

Module 10

Connecting the Arduino - Part 1



Unit 1 - Introducing the Arduino



Video # 21 - <https://rntlab.com/28hasvideos>

If you've followed the previous Modules, you know how easy it is to use the ESP8266 with your home automation system.

About the Arduino

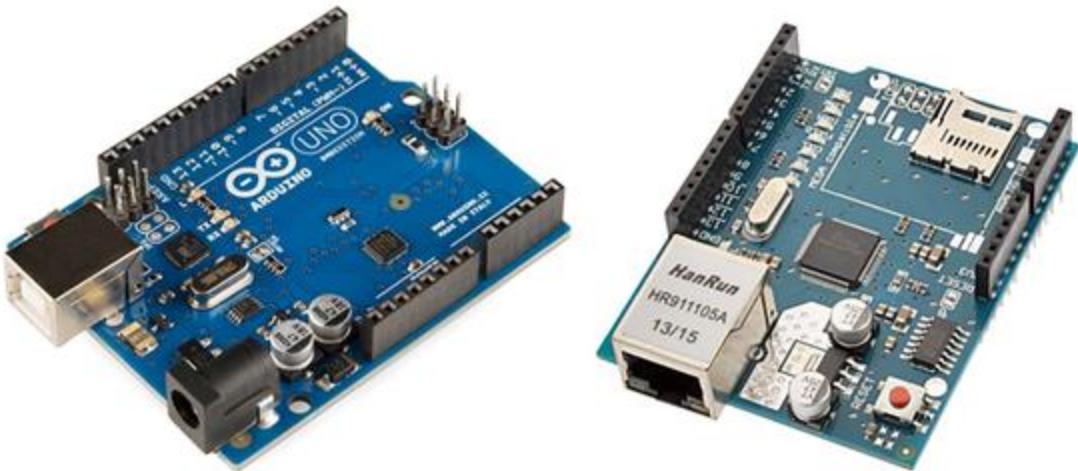
However, in this Module I would like to introduce you to the Arduino board and how you can integrate an Arduino in your home automation system.

Attaching an Ethernet Shield to Your Arduino

You're going to attach an Ethernet shield to your Arduino board.

The Ethernet shield when attached to your Arduino board, connects it to your network and you can establish an MQTT connection with Node-RED.

For this course I recommend that you get an Ethernet shield with a chip called WIZnet W5100. It's very cheap if you get them on eBay.



So what can you do with an Arduino?

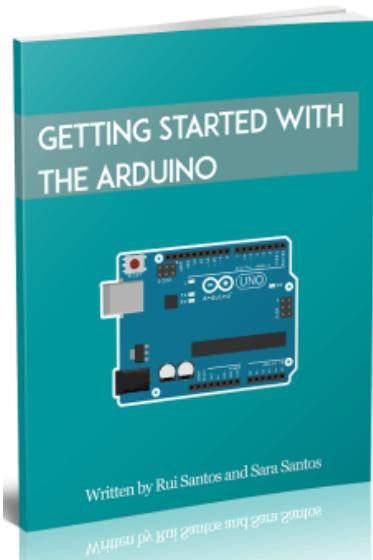
The Arduino is a very versatile board and allows you to:

- Attach various sensors
- Access 6 analog pins
- Establish an MQTT communication
- Trigger actions
- Save sensor data

Read the Arduino eBook

I've also prepared a PDF eBook called Getting Started with the Arduino and you can download it from the Toolbox.

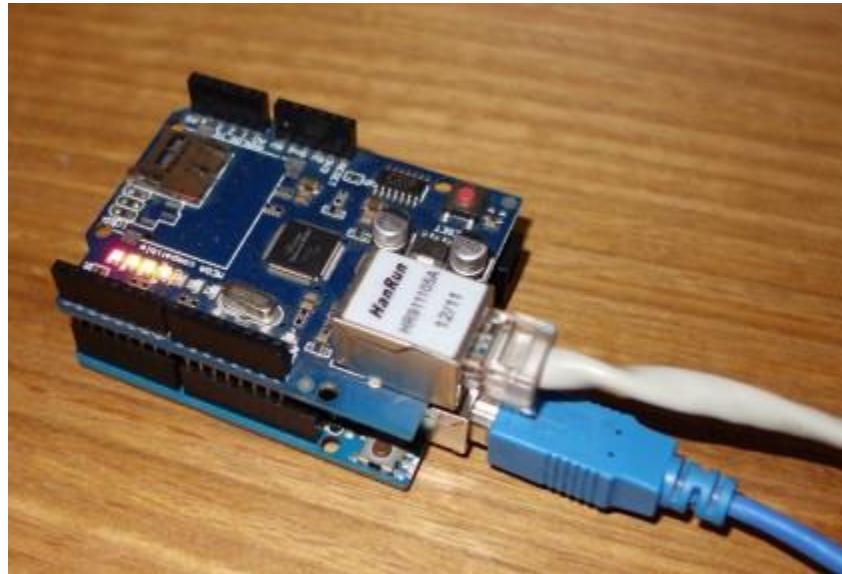
That eBook only covers the Arduino basic concepts, but it will be helpful if you've never used an Arduino before.



Mounting an Ethernet Shield

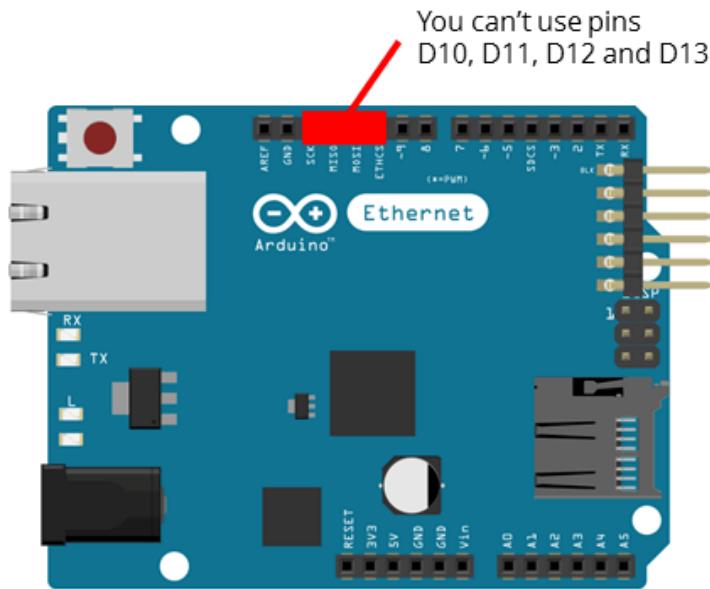
You can easily mount an Ethernet shield to an Arduino board. Then, you can power your Arduino through the USB cable.

Note: you must connect an Ethernet cable from your router to your Ethernet shield as shown in the following figure.



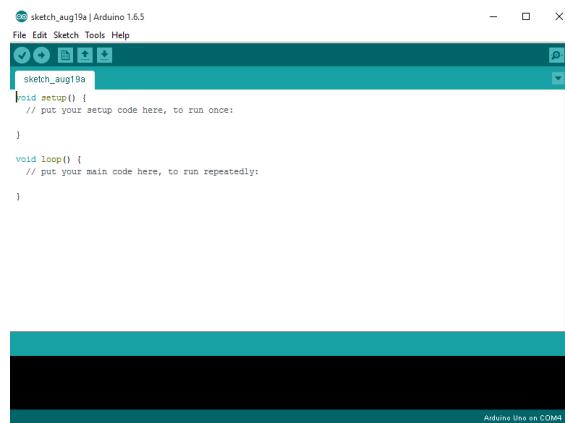
Pin Usage

When the Arduino is connected to an Ethernet shield, you can't use Digital pins from 10 to 13, because they are being used in order to establish a communication between the Arduino and the Ethernet shield.



Installing the Arduino IDE

In this course, you've already programmed your ESP using the Arduino IDE software. So, you should have the Arduino IDE installed in your computer.



If you have already followed the ESP8266 Modules, the next three Units are fairly similar, but it's still important that you go through them to ensure that you aren't missing anything.

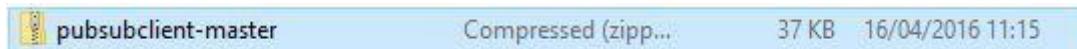
Unit 2 - Installing the PubSubClient Library

If you have completed the previous Modules you already have the PubSubClient library installed in the Arduino IDE, so you can skip this Unit.

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

Installing the Library

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



- 2) Unzip the .zip folder and you should get pubsubclient-master folder



- 3) Rename your folder from ~~pubsubclient-master~~ [pubsubclient-ma](#) to pubsubclient



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

Name	Date modified	Type	Size
Bridge	08/03/2016 17:07	File folder	
Esplora	20/05/2015 17:10	File folder	
Ethernet	09/03/2016 16:11	File folder	
Firmata	16/02/2016 01:32	File folder	
GSM	09/03/2016 16:11	File folder	
Keyboard	08/03/2016 17:05	File folder	
LiquidCrystal	09/03/2016 16:11	File folder	
Mouse	08/03/2016 17:04	File folder	
pubsubclient	16/04/2016 11:16	File folder	
Robot_Control	10/06/2015 15:00	File folder	
Robot_Motor	10/06/2015 15:00	File folder	
RobotIRremote	10/06/2015 15:00	File folder	
SD	09/03/2016 16:11	File folder	
Servo	09/03/2016 16:11	File folder	
SpacebrewYun	27/03/2015 15:01	File folder	
Stepper	09/03/2016 16:11	File folder	
Temboo	08/03/2016 15:58	File folder	
TFT	09/03/2016 16:11	File folder	
WiFi	09/03/2016 16:11	File folder	

5) Then, re-open your Arduino IDE

The library comes with a number of example sketches.

See **File> Examples > PubSubClient** within the Arduino IDE software.

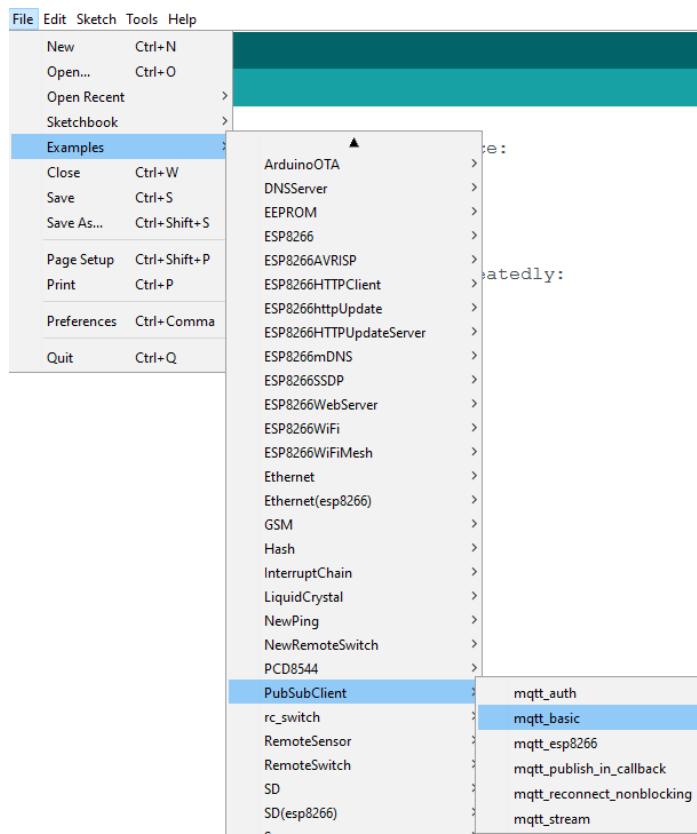
Unit 3 - Connecting the Arduino to the Node-RED Nodes

This Unit shows how you can publish messages and subscribe to a topic with MQTT using an Arduino with an Ethernet shield.

These are the basic concepts that will allow you to turn on your lights and monitor your sensors (which we are going to cover in the next Module).

Writing Your Arduino Sketch

Open the Arduino IDE. Go to **File > Examples > PubSubClient> mqtt_basic.**



A new sketch opens. That sketch publishes a message and subscribes to a topic with MQTT.

Understanding How the Sketch Works

First, it starts by loading the **SPI** library, the **Ethernet** library and the **PubSubClient** library.

```
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>
```

It initializes the `ethClient` and creates a client.

```
EthernetClient ethClient;
PubSubClient client(ethClient);
```

Configuring your network

Then, it configures the Arduino with your network details.

```
// Update these with values suitable for your network.
byte mac[] = { 0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };
IPAddress ip(192, 168, 1, 99);
IPAddress server(192, 168, 1, 76);
```

Important: You actually need to replace those two variables with appropriate values that are suitable for your network, otherwise your Arduino will not establish a connection with your network neither with your MQTT broker.

Replace the following line with an IP that is available and suitable for your network:

```
 IPAddress ip(X, X, X, X);
```

In my case, my IP range is 192.168.1.X and with the software [Angry IP Scanner](#) I know that the IP 192.168.1.99 is available in my network, because it doesn't have any active device in my network with that exact same IP address:

```
IPAddress ip(192, 168, 1, 99);
```

Replace the next line with your Raspberry Pi IP Address:

```
IPAddress server(X, X, X, X);
```

My RPi IP address is 192.168.1.76, so it becomes:

```
IPAddress server(192, 168, 1, 76);
```

Note: Your IP range is very likely to be different from mine, but if you already know the RPi IP address you know the range that you are working with.

callback

The `callback()` function is triggered every time another device publishes a messages to your Arduino (in our example it will be the Raspberry Pi through the Node-RED software).

At the moment, it only prints that message in your serial monitor. Later, you are actually going to use this mechanism to turn on/off your lights.

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

reconnect

If the Arduino loses connection with the MQTT broker it executes the `reconnect()` function.

If for some reason your Arduino can't establish an MQTT communication with your broker, your Arduino will keep trying to reconnect every 5 seconds.

When the connection is established, it does two things:

- Publishes a message to the `outTopic` topic
- Subscribes to the `inTopic` topic

```
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("arduinoClient")) {
            Serial.println("connected");
            // Once connected, publish an announcement...
            client.publish("outTopic", "hello world");
            // ... and resubscribe
            client.subscribe("inTopic");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}
```

setup

The `setup()` function starts the serial communication at a baud rate of 57600.

The `setup()` also:

- connects your Arduino to the MQTT broker
- sets the `callback()` function
- prepares the Ethernet

```
void setup()
{
    Serial.begin(57600);

    client.setServer(server, 1883);
    client.setCallback(callback);

    Ethernet.begin(mac, ip);
    // Allow the hardware to sort itself out
    delay(1500);
}
```

loop

The `loop()` function only checks if your Arduino is connected to the MQTT broker. If your Arduino isn't connected, it tries to connect/reconnect.

```
void loop()
{
    if (!client.connected()) {
        reconnect();
    }
    client.loop();
}
```

Uploading the Sketch

Finally you can upload the full sketch to your Arduino (replace with the appropriate IP and RPi IP address for your network):

```

*****  

All the resources for this project:  

https://rntlab.com/  

****/  

#include <SPI.h>  

#include <Ethernet.h>  

#include <PubSubClient.h>  

// Update these with values suitable for your network.  

byte mac[] = { 0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };  

IPAddress ip(192, 168, 1, 99);  

IPAddress server(192, 168, 1, 76);  

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

    Serial.print("Message arrived [");  

    Serial.print(topic);  

    Serial.print("] ");  

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

        Serial.print((char)payload[i]);  

    }  

    Serial.println();  

}  

EthernetClient ethClient;  

PubSubClient client(ethClient);  

void reconnect() {  

    // Loop until we're reconnected  

    while (!client.connected()) {  

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

        // Attempt to connect  

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

            Serial.println("connected");  

            // Once connected, publish an announcement...  

            client.publish("outTopic","hello world");  

            // ... and resubscribe  

            client.subscribe("inTopic");  

        } else {  

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

}

```

```

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

}

}

void setup()
{
Serial.begin(57600);

client.setServer(server, 1883);
client.setCallback(callback);

Ethernet.begin(mac, ip);
// Allow the hardware to sort itself out
delay(1500);
}

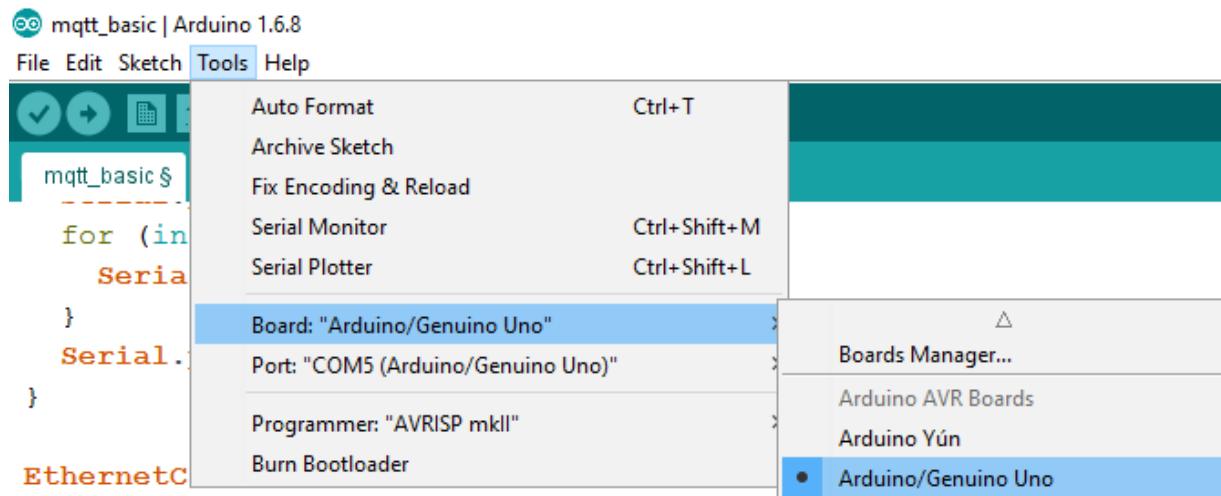
void loop()
{
if (!client.connected()) {
  reconnect();
}
client.loop();
}

```

DOWNLOAD SOURCE CODE

[https://github.com/RuiSantosdotme/Home-Automation-
Course/blob/master/code/mqtt_arduino.ino](https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/mqtt_arduino.ino)

In order to upload the sketch, you have to select the **Arduino/Genuino** board under **Tools > Board** menu:

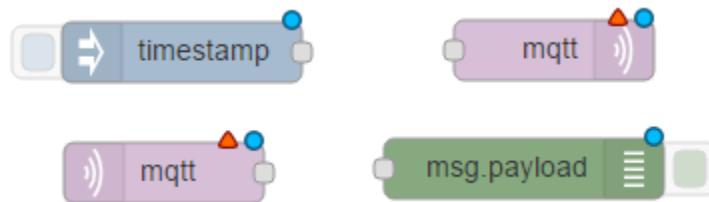


Creating Your Flow

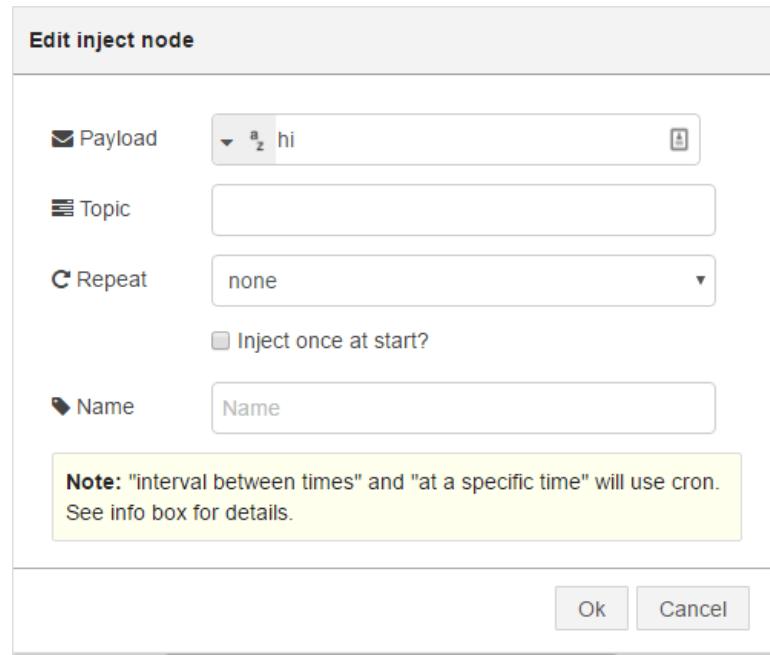
Lastly, you have to create a Node-RED flow to subscribe to the `outTopic` topic and to publish messages to the `inTopic` topic.

Let's start by dragging 4 nodes to your flow:

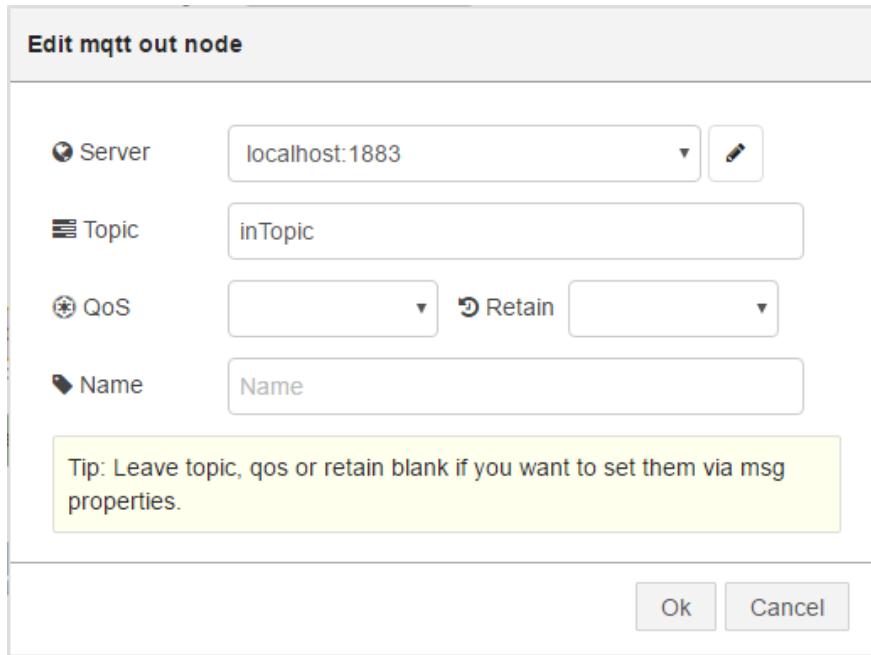
- Inject node
- mqtt out
- mqtt in
- Debug node



Edit the **Inject** node. Select the **String** payload and type any message in the payload field, for example "hi".



Double-click the **mqtt in** node to edit its settings. Select the **localhost** option and type **inTopic**.



Double-click the **mqtt out** node to edit its settings. Select the **localhost** option and type **outTopic**.

Edit mqtt in node

Server	localhost:1883	▼	
Topic	outTopic		
Name	Name		

Ok Cancel

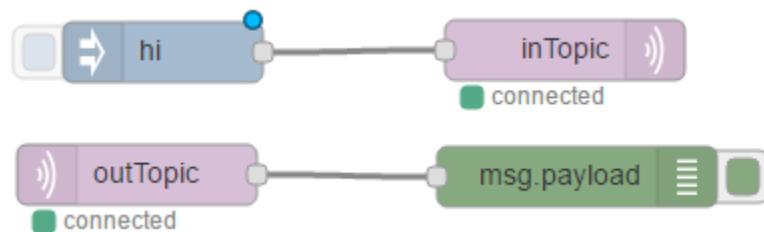
All the settings for the **Debug** node are configured properly by default.

Edit debug node

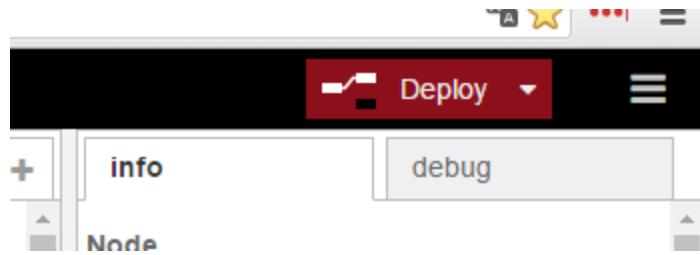
Output	message property	▼
	msg. payload	
to	debug tab	▼
Name	Name	

Ok Cancel

This is how it should look like in the end:

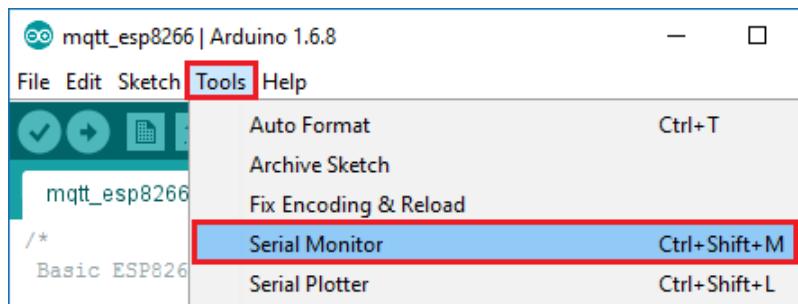


Click the **Deploy** button on the top-right corner to save your application.

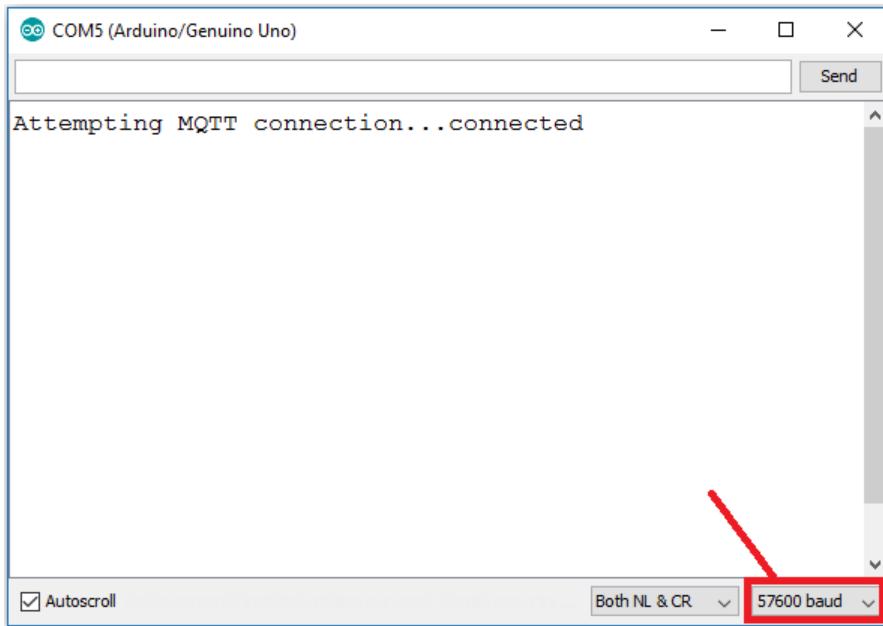


Opening the Node-RED

Open the Arduino IDE serial monitor:



Set the baud rate to 57600:



Open the Node-RED software.

http://YOUR_RPi_IP_ADDRESS:1880

outTopic

At this point, the Debug node is subscribed to the **outTopic** topic, so receives the message “hello world” from your Arduino on boot up.

Click the reset button of your Arduino board (see figure below), because the “hello world” message is only published once on boot up.



Open the debug window in the Node-RED software and you should see the “hello world” message there.

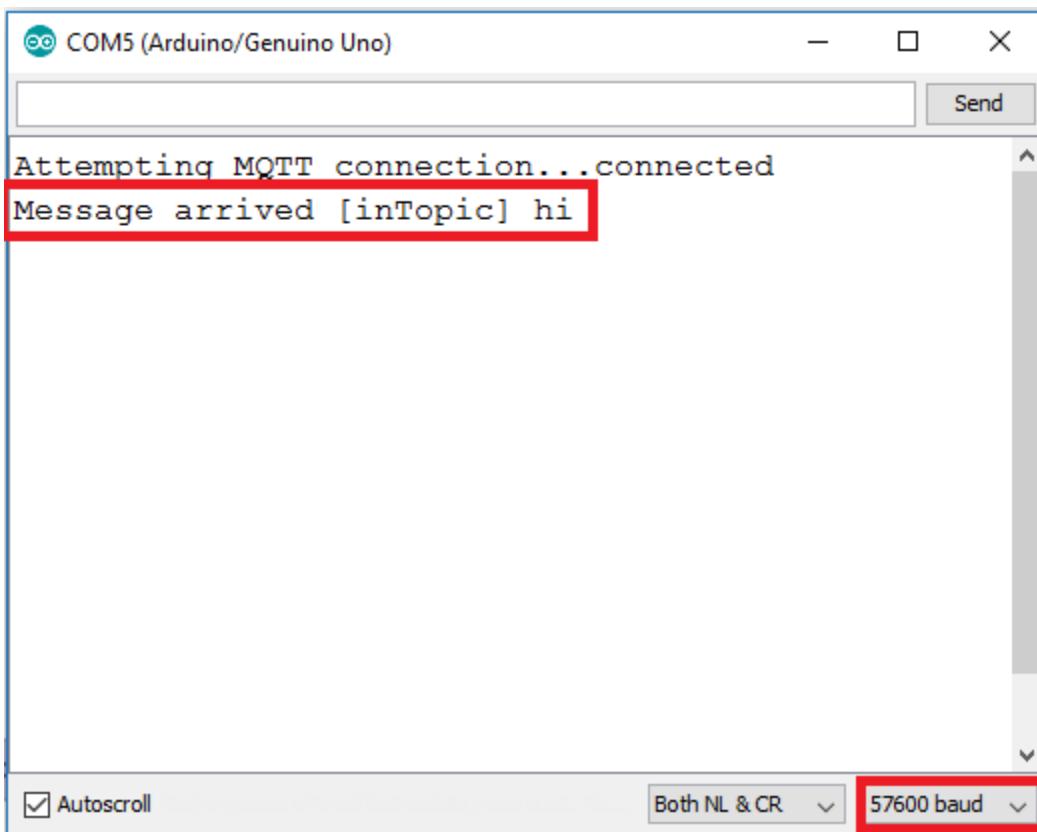


inTopic

If you trigger the **Inject** node, you can publish messages to the **inTopic** topic.



Since your Arduino is subscribed to the **inTopic** topic, it receives the "hi" message and prints it in the serial monitor every time you trigger the **Inject** node.



By now, you have acquired all the basic MQTT concepts with the Arduino (Publish/Subscribe). In the next Module, you'll start controlling real lamps and monitor sensors with your Arduino.

Unit 4 - Controlling Outputs with Arduino using MQTT

In the previous Unit, I've demonstrated how to subscribe and publish messages to a topic. Now, you're going to use that mechanism to control two LEDs, which later will be replaced with real lights.

The sketch presented in this Unit shows you how to control an LED on/off or adjust the brightness through the Node-RED Dashboard.

I've also added additional comments to the sketch to explain what each function does and what you should and shouldn't change.

You can open this link or scroll down this page to see the complete sketch and follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_arduino_leds.ino

Understanding How the Sketch Works

First, it starts by loading the **SPI** library, the **Ethernet** library and the **PubSubClient** library.

```
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>
```

Configuring your network

Then, it configures the Arduino with your network details.

Important: You actually need to replace those two variables with appropriate values that are suitable for your network, otherwise your Arduino will not establish a connection with your network neither with your MQTT broker.

Replace the following line with an IP that is available and suitable for your network:

```
IPAddress ip(X, X, X, X);
```

In my case, my IP values range is 192.168.1.X and with the software [Angry IP Scanner](#) I know that the IP 192.168.1.99 is available in my network, because it doesn't have any device in my network with that exact same IP address, so this works:

```
IPAddress ip(192, 168, 1, 99);
```

Replace that line with your Raspberry Pi IP Address:

```
IPAddress server(X, X, X, X);
```

My RPi IP address is 192.168.1.76, so it becomes:

```
IPAddress server(192, 168, 1, 76);
```

The MAC address can remain the same:

```
byte mac[] = { 0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };
```

Creating variables

It initializes the `ethClient` and creates a client.

```
EthernetClient ethClient;
PubSubClient client(ethClient);
```

Defines two variables that refer to Digital pin 6 and Digital pin 7 of your Arduino.

```
const int ledPin6 = 6;
const int ledPin7 = 7;
```

callback

Here, you have the `callback()` function.

```
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
    // Feel free to add more if statements to control more Pins with

    // If a message is received on the topic home/livingroom/arduino
    // It's a value between 0 and 255 to adjust the LED brightness
    if(String(topic)=="home/livingroom/arduino/ledPin6"){
        Serial.print("Changing Digital Pin 6 Brithness to ");
        Serial.print(messageTemp);
        analogWrite(ledPin6, messageTemp.toInt());
    }
    // If a message is received on the topic home/livingroom/arduino
    //you check if the message is either 1 or 0. Turns the Arduino D
    if(String(topic)=="home/livingroom/arduino/ledPin7"){
        Serial.print("Changing Digital Pin 7 to ");
        if(messageTemp == "1"){
            digitalWrite(ledPin7, HIGH);
            Serial.print("On");
        }
        else if(messageTemp == "0"){
            digitalWrite(ledPin7, LOW);
            Serial.print("Off");
        }
    }
    Serial.println();
}
```

This function is executed when a device publishes a message to a topic that your Arduino is subscribed to.

You can change the function above to add logic to your programs, so when a device publishes a message to a topic that your Arduino is subscribed to, you can actually do something useful with that message.

These are the lines of code that actually change the brightness of your LED. If a message is received on the topic **home/livingroom/arduino/ledPin6**, you convert the message to an int value to change the brightness of the LED.

```
if(String(topic)=="home/livingroom/arduino/ledPin6") {
    Serial.print("Changing Digital Pin 6 Brightness to ");
    Serial.print(messageTemp);
    analogWrite(ledPin6, messageTemp.toInt());
}
```

The next snippet of code turn the LED on/off. If a message is received on the topic **home/livingroom/arduino/ledPin7**, you check if the message is either 1 or 0.

Then, it turns the Arduino Digital pin 7 on/off according to the message.

```
if(String(topic)=="home/livingroom/arduino/ledPin7") {
    Serial.print("Changing Digital Pin 7 to ");
    if(messageTemp == "1"){
        digitalWrite(ledPin7, HIGH);
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        digitalWrite(ledPin7, LOW);
        Serial.print("Off");
    }
}
```

reconnect

The `reconnect()` function connects/reconnects your Arduino to your MQTT broker.

```
void reconnect() {
```

You can change this function if you want your Arduino subscribed to more topics.

```
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("arduinoClient")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs)
            client.subscribe("home/livingroom/arduino/ledPin6");
            client.subscribe("home/livingroom/arduino/ledPin7");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}
```

In this case, I'm subscribed to only two topics `home/livingroom/arduino/ledPin6` and `home/livingroom/arduino/ledPin7`.

You can add more topics, if you would like to control more LEDs. Feel free to experiment with that.

setup

The `setup()` function:

- sets your Arduino pins as OUTPUTs
- starts the serial communication at a baud rate of 57600
- sets your mqtt broker
- sets the `callback()` function
- starts the Ethernet communication

```

void setup()
{
    pinMode(ledPin6, OUTPUT);
    pinMode(ledPin7, OUTPUT);

    Serial.begin(57600);

    client.setServer(server, 1883);
    client.setCallback(callback);

    Ethernet.begin(mac, ip);
    // Allow the hardware to sort itself out
    delay(1500);
}

```

loop

For this project, you don't need to change anything in the `loop()` function. Basically it keeps checking that your Arduino is connected to your broker, otherwise it tries to reconnect.

```

void loop()
{
    if (!client.connected()) {
        reconnect();
    }
    client.loop();
}

```

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your Arduino board (after adding the appropriate IP and RPi IP address for your network).

```
*****
```

All the resources for this project:

<https://rntlab.com/>

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

```

// Loading the libraries
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>

// This MAC address can remain the same
byte mac[] = { 0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };

/*
 * IMPORTANT!
 * YOU MUST UPDATE THESE NEXT TWO VARIABLES WITH VALUES SUITABLE TO YOUR
NETWORK!
*
*/

```

// Replace with an IP that is suitable for your network. I could use any IP in this range: 192.168.1.X
// I've used a tool <http://angryip.org/> to see an available IP address and I ended up using 192.168.1.99

```

IPAddress ip(192, 168, 1, 99);

// Replace with your Raspberry Pi IP Address. In my case, the RPi IP Address is 192.168.1.76
IPAddress server(192, 168, 1, 76);

// Initializes the clients
EthernetClient ethClient;
PubSubClient client(ethClient);

// Connect two LEDs to your Arduino. One to pin 6 and the other to pin 7
const int ledPin6 = 6;
const int ledPin7 = 7;

// This function is executed when some device publishes a message to a topic that your Arduino is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your Arduino is subscribed you can actually do something
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

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

```

```

Serial.println();
// Feel free to add more if statements to control more Pins with MQTT

// If a message is received on the topic home/livingroom/arduino/ledPin6
// It's a value between 0 and 255 to adjust the LED brightness
if(String(topic)=="home/livingroom/arduino/ledPin6"){
    Serial.print("Changing Digital Pin 6 Brithness to ");
    Serial.print(messageTemp);
    analogWrite(ledPin6, messageTemp.toInt());
}

// If a message is received on the topic home/livingroom/arduino/ledPin7,
//you check if the message is either 1 or 0. Turns the Arduino Digital Pin according to the message
if(String(topic)=="home/livingroom/arduino/ledPin7"){
    Serial.print("Changing Digital Pin 7 to ");
    if(messageTemp == "1"){
        digitalWrite(ledPin7, HIGH);
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        digitalWrite(ledPin7, LOW);
        Serial.print("Off");
    }
}
Serial.println();
}

```

```

// This functions reconnects your Arduino to your MQTT broker
// Change the function below if you want to subscribe to more topics with your Arduino
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("arduinoClient")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("home/livingroom/arduino/ledPin6");
            client.subscribe("home/livingroom/arduino/ledPin7");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
        }
    }
}

```

```

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

}

}

// The setup function sets your Arduino pins as Outputs, starts the serial communication at a baud rate of 57600
// Sets your mqtt broker and sets the callback function. The callback function is what receives messages and
// actually controls the LEDs. Finally starts the Ethernet communication
void setup()
{
pinMode(ledPin6, OUTPUT);
pinMode(ledPin7, OUTPUT);

Serial.begin(57600);

client.setServer(server, 1883);
client.setCallback(callback);

Ethernet.begin(mac, ip);
// Allow the hardware to sort itself out
delay(1500);
}

// For this project, you don't need to change anything in the loop function.
// Basically it ensures that you Arduino is connected to your broker
void loop()
{
if (!client.connected()) {
reconnect();
}
if(!client.loop())
client.connect("ethClient");
}

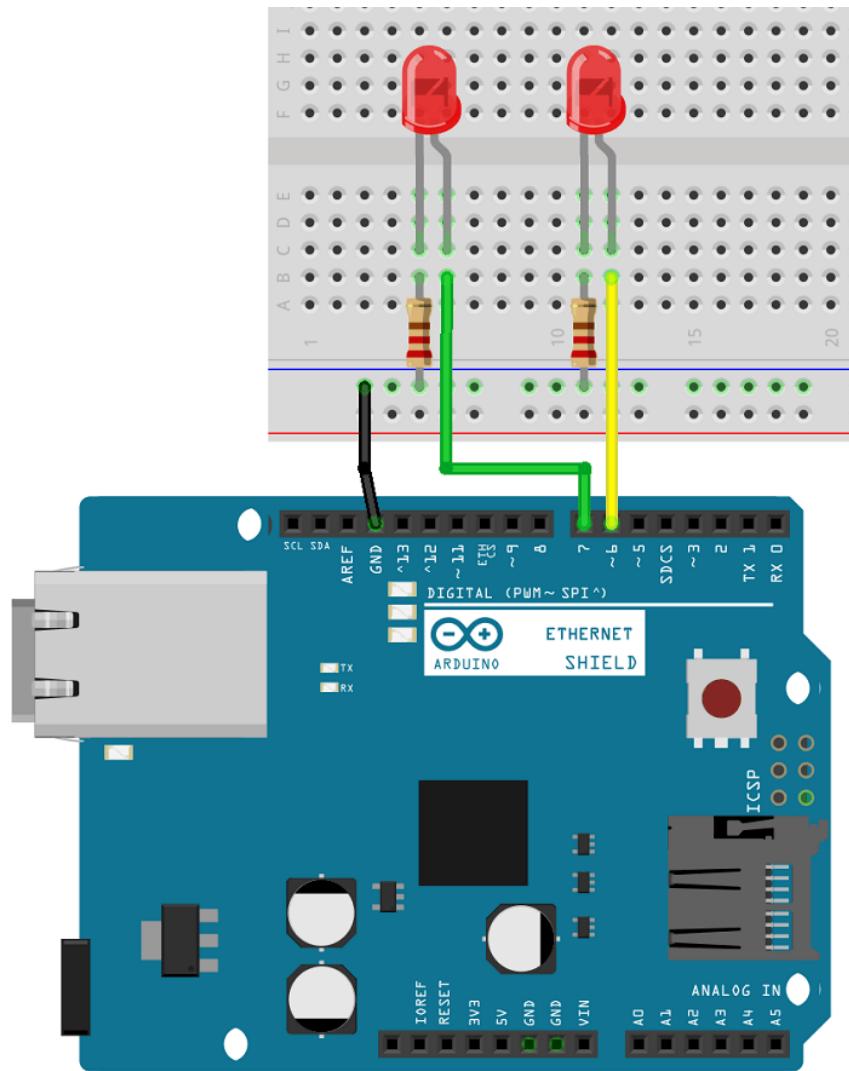
```

DOWNLOAD SOURCE CODE

https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_arduino_leds.ino

Schematics

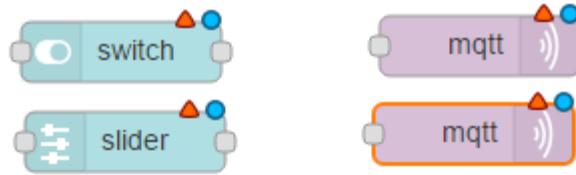
The schematics for this project are very straightforward. Simply connect two LEDs with two 220 Ohm resistors to your Arduino as shown in the figure below.



Creating the Flow

Go to the Node-RED software and you should create a flow by following these next 7 steps:

1 – Drag 4 nodes



2 – Switch digital pin 7

Edit switch node

Cancel Done

Group: Group 1 [Living Room]

Size: auto

Label: Switch

Icon: Default

When clicked, send:

On Payload: true

Off Payload: false

Topic:

Name: Switch

This screenshot shows the configuration dialog for a 'switch' node. It includes settings for grouping, size, label, icon, and a 'When clicked, send:' section. The 'On Payload' field is set to 'true' and the 'Off Payload' field is set to 'false'. The 'Name' field is also labeled 'Switch'.

3 – MQTT out node publishes a message 0 or 1

Edit mqtt out node

Cancel Done

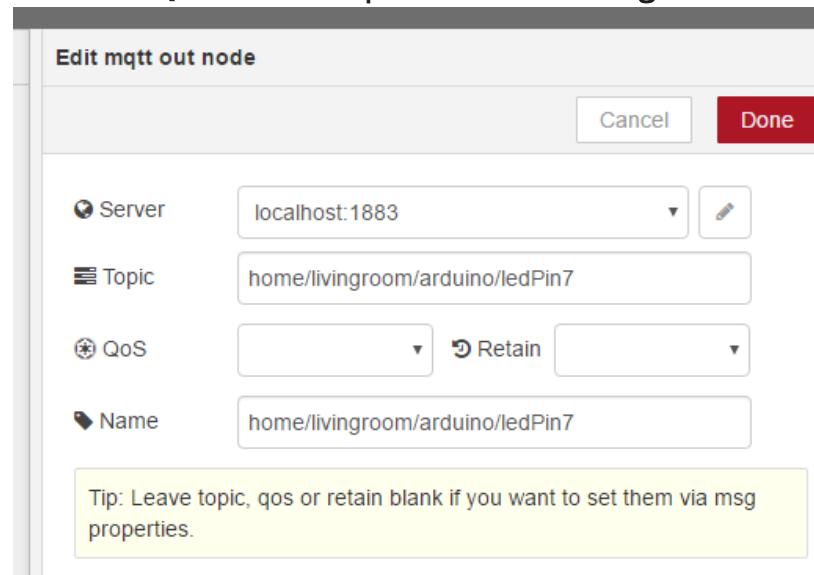
Server: localhost:1883

Topic: home/livingroom/arduino/ledPin7

QoS: Retain

Name: home/livingroom/arduino/ledPin7

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



4 – Slider digital pin 6

Edit slider node

Cancel Done

Group: Group 1 [Living Room]

Size: auto

Label: Slider

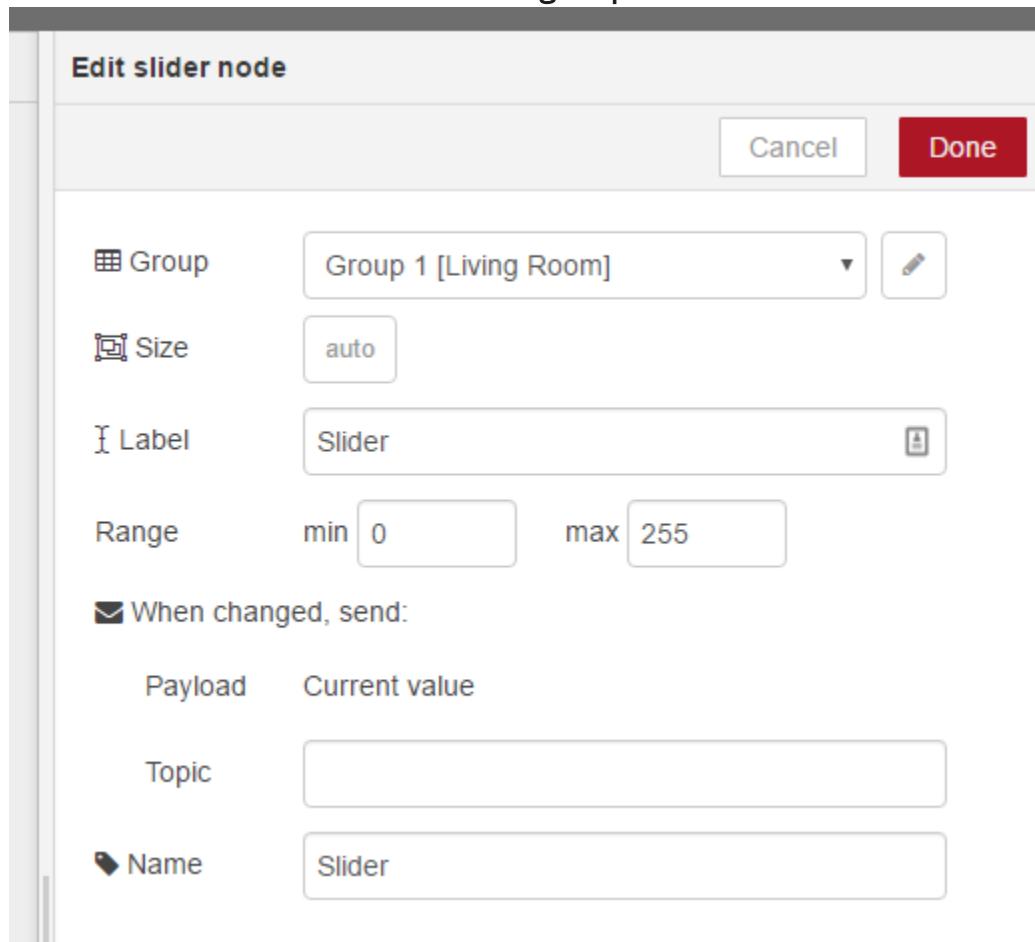
Range: min 0 max 255

When changed, send:

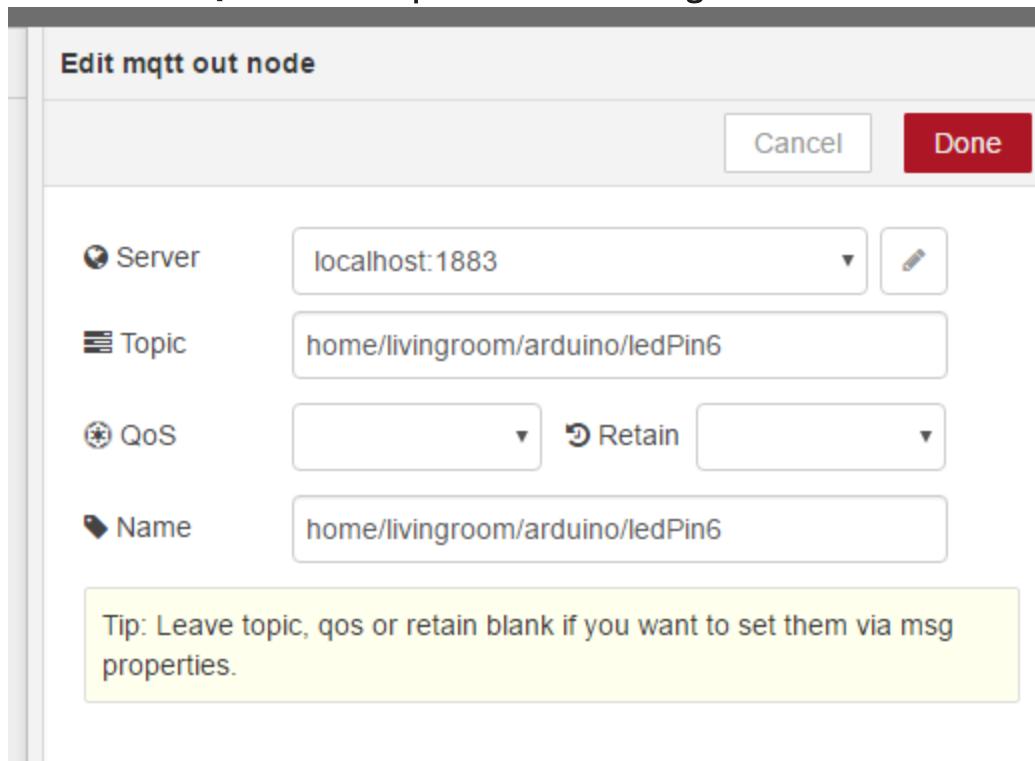
Payload: Current value

Topic:

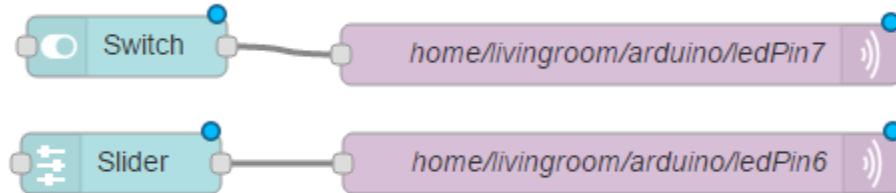
Name: Slider



5 – MQTT out node publishes a message from 0 to 255



6 – Final flow

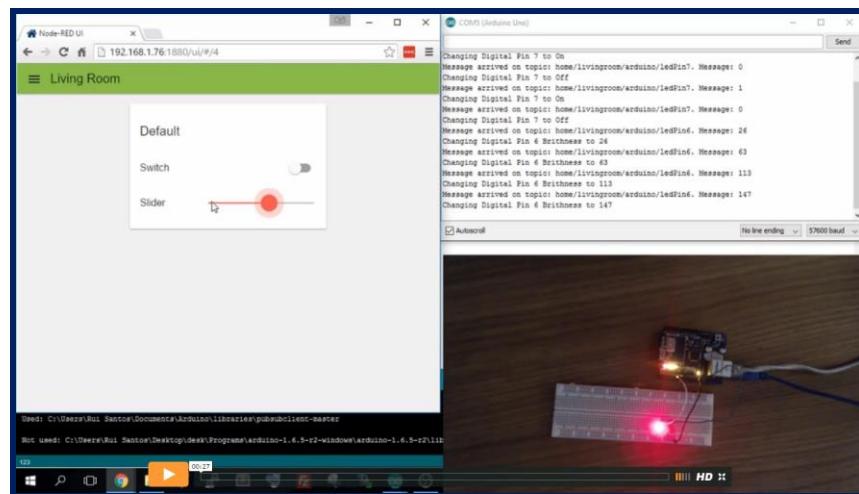
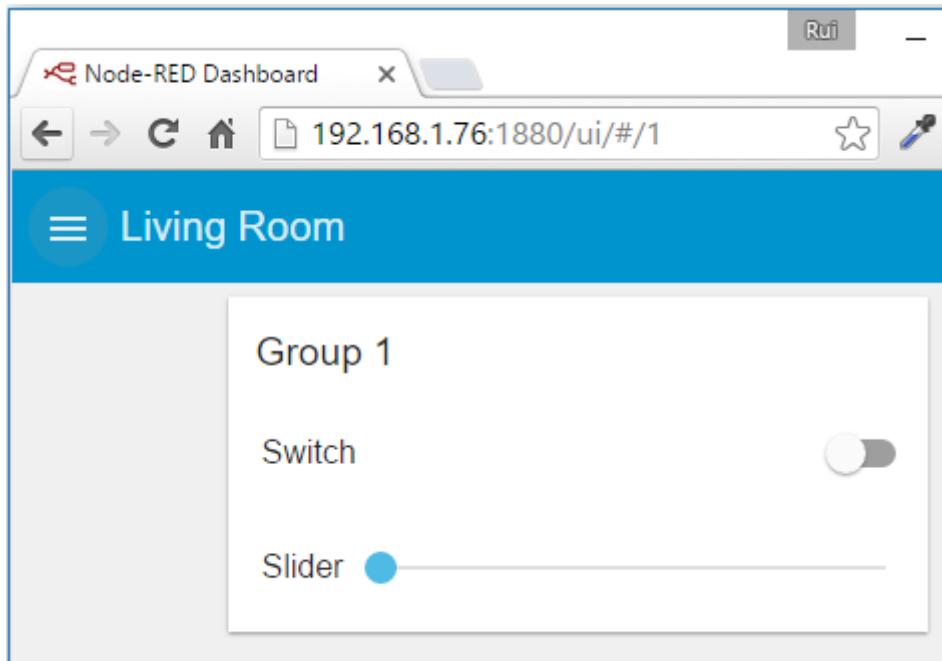


7 – Deploy your application



Opening the Node-RED Dashboard

You can control the LED on/off by pressing the Switch button and adjust the LED brightness with the Slider (for a demonstration watch the video below).

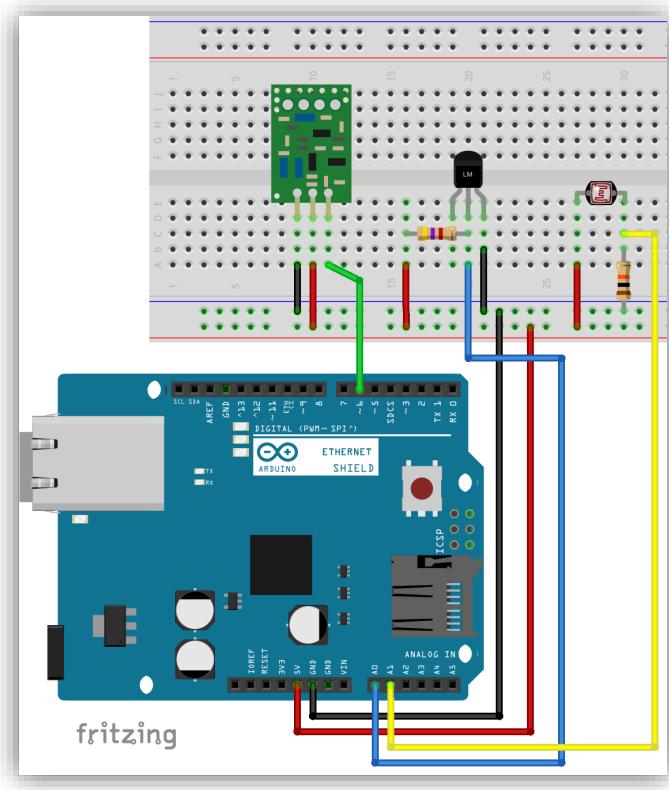


Video # 22 - <https://rntlab.com/28hasvideos>

In the next Module, you're going to replace those LEDs with outlets

Module 11

Connecting the Arduino - Part 2



Unit 1 - Decoding RF Signals to Control Outlets

If you've followed all previous Modules, this Unit is repeated and it's very likely that you already know how it works. You can go through this Unit again to decode the binary codes from another set of RF controlled outlets that are going to be controlled with your Arduino.

You are finally going to control real things with your Arduino and make useful stuff.

I've tried different methods of controlling the mains voltage, but some of the methods require:

1. Experience dealing with AC voltage
2. Opening holes in your wall/ceiling/switches
3. Modifying the electrical panel
4. Knowing the electrical rules for each country

It was very hard to come up with a solution that is safe and works for everyone.

Solution

I wanted to create a course that would work regardless of the country. The solution for this problem was using radio frequency (RF) controlled outlets.

Why? Using these remote controlled outlets have 5 benefits:

1. Fairly inexpensive
2. Easy to get
3. Works with ESP8266 and Arduino
4. Safe to use

5. Works in any country

Parts Required

To complete this Unit you need:

- 1x Arduino UNO
- 1x 433MHz Receiver
- 1x Remote Controlled Outlets that operate at 433MHz

You can see the [complete list of components and parts](#) in Module 1.

Note: you need to buy remote controlled outlets that operate at a RF of 433MHz. They should say the operating RF either in the product page or in the label.

Example

Here's how they look:



There are also [E27 Lamp Bulb Holders](#) that are controlled with RF remote. They should also operate at 433MHz and they work exactly like the outlets.



You simply connect an E27 lamp to the E27 remote controlled holder.



Then, you attach this to the lamp holder that you want to control. You can turn the light on and off with your remote control:



Setting the RF Channels

I set my remote control to the I position.



The outlets must be both on the I position. I've selected channels 3 and 4 for the outlets (you can use any channel you want).



If you plug them to an outlet, you should be able to control the remote controlled outlets with your remote control.

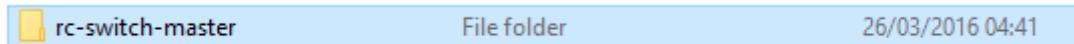
Installing the RC Switch Library

The [RC_Switch_library](#) provides an easy way of using your ESP8266, Arduino or Raspberry Pi to operate remote radio controlled devices. This will most likely work with all popular low cost power outlet sockets.

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



- 2) Unzip the .zip folder and you should get **rc-switch-master** folder



- 3) Rename your folder from **rc-switch-master** to **rc_switch**



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

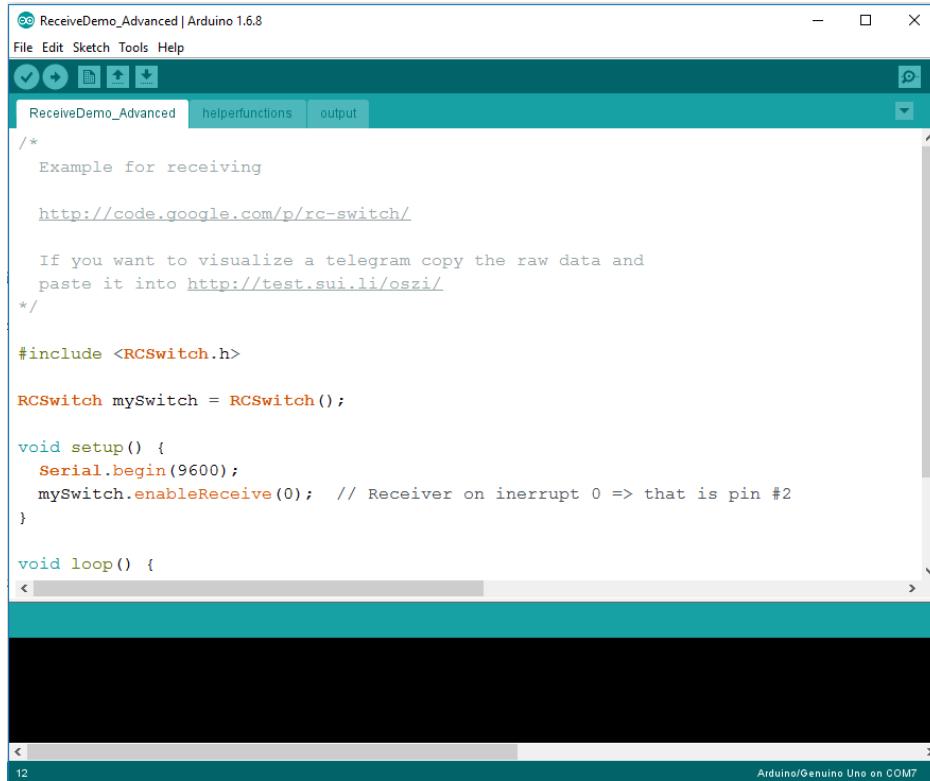
Name	Type
Bridge	File folder
Esplora	File folder
Ethernet	File folder
Firmata	File folder
GSM	File folder
InterruptChain	File folder
Keyboard	File folder
LiquidCrystal	File folder
Mouse	File folder
NewRemoteSwitch	File folder
pubsubclient	File folder
rc_switch	File folder

5) Then, re-open your Arduino IDE

Opening the Decoder Sketch

You need to decode the signals that your remote control sends, so that the Arduino can reproduce those signals and ultimately control the outlets.

The library comes with several sketch examples. Within the Arduino IDE software, you need to go to **File > Examples > RC_Switch > ReceiveDemo_Advanced**.



```
ReceiveDemo_Advanced | Arduino 1.6.8
File Edit Sketch Tools Help
ReceiveDemo_Advanced helperfunctions output
/*
Example for receiving

http://code.google.com/p/rc-switch/

If you want to visualize a telegram copy the raw data and
paste it into http://test.sui.li/oszi/
*/
#include <RCswitch.h>

RCswitch mySwitch = RCswitch();

void setup() {
  Serial.begin(9600);
  mySwitch.enableReceive(0); // Receiver on interrupt 0 => that is pin #2
}

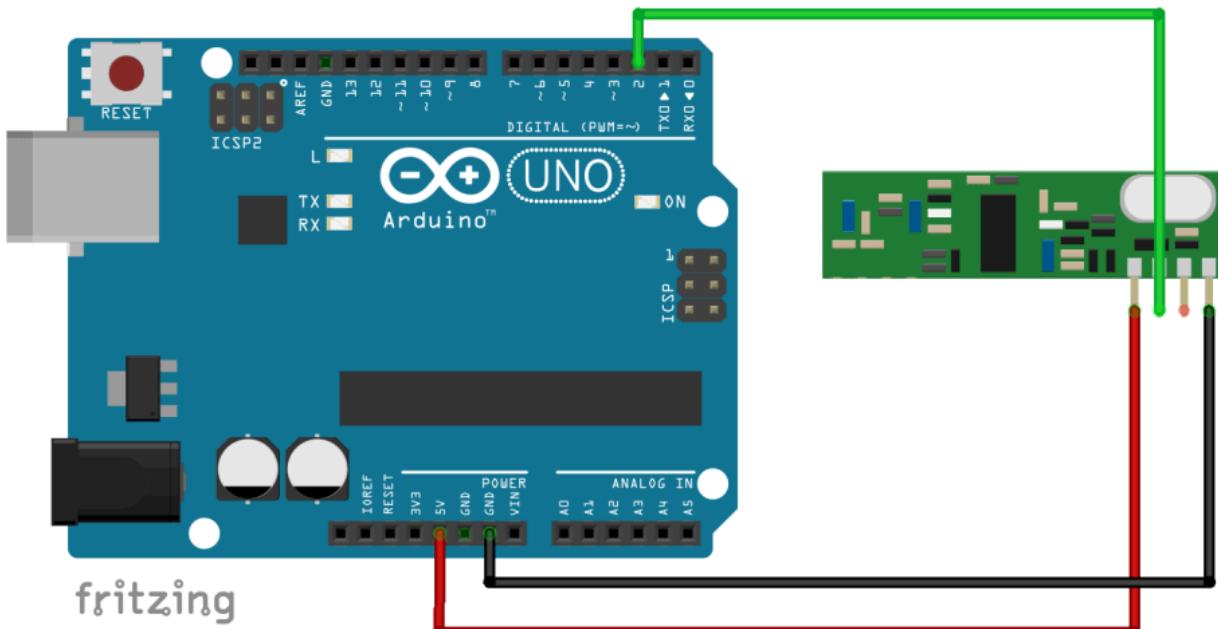
void loop() {
< >
12
Arduino/Genuine Uno on COM7
```

Having an Arduino board connected to your computer follow these instructions:

1. Go to the **Tools** tab
2. Select **Arduino UNO** board
3. Select the **COM** port
4. Press the **Upload** button

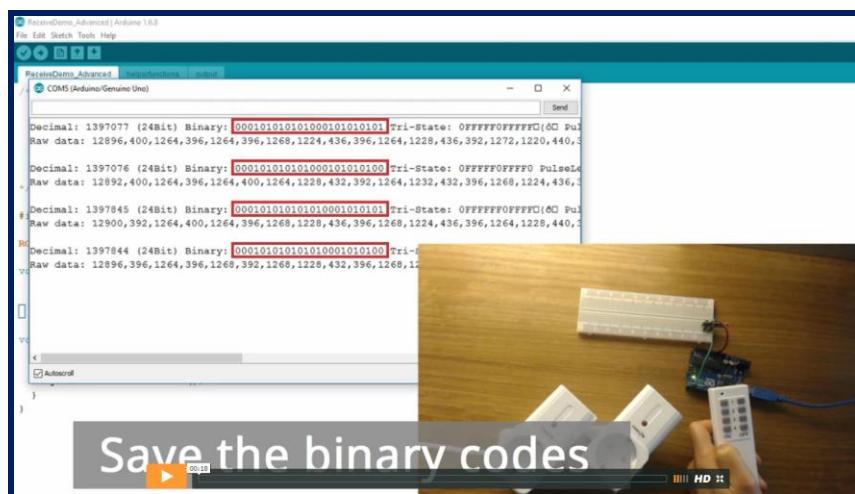
Schematics

After the sketch is uploaded connect an 433MHz RF receiver to Digital Pin 2 of your Arduino UNO board:



Decoding the RF Signals

Open the Arduino IDE serial monitor and start pressing the buttons. As shown in the video demonstration below:



Video # 23 - <https://rntlab.com/28hasvideos>

After pressing each button one time, you can see the binary code for each button (it's highlighted in red):

```
Decimal: 1397077 (24Bit) Binary: 000101010101000101010101 Tri-State: OFFFFF0FFFFF0 PulseLength: 416 microseconds Protocol: 1  
Raw data: 12924,392,1272,396,1272,392,1272,1224,440,392,1276,1224,440,392,1272,1224,444,388,1276,1224,440,392,1276,1224,444,388,  
  
Decimal: 1397076 (24Bit) Binary: 000101010101000101010100 Tri-State: OFFFFF0FFFF0 PulseLength: 416 microseconds Protocol: 1  
Raw data: 12924,392,1272,396,1268,396,1268,1228,436,388,1276,1224,440,392,1272,1224,440,396,1268,1228,440,392,1272,1224,440,392,  
  
Decimal: 1397845 (24Bit) Binary: 000101010101000101010101 Tri-State: OFFFFFF0FFFF PulseLength: 416 microseconds Protocol: 1  
Raw data: 12928,388,1276,392,1272,392,1272,1228,440,388,1276,1224,444,388,1276,1220,444,388,1276,1224,440,392,1272,1220,444,388,  
  
Decimal: 1397844 (24Bit) Binary: 000101010101000101010100 Tri-State: OFFFFF0FFF0u PulseLength: 416 microseconds Protocol: 1  
Raw data: 12928,396,1268,396,1268,396,1268,1228,436,396,1272,1224,440,396,1268,1224,444,392,1272,1224,440,392,1276,1216,448,384,
```

Save your binary codes for each button press:

- **Button 3 ON** = (24Bit) Binary: 000101010101000101010101
 - **Button 3 OFF** = (24Bit) Binary: 000101010101000101010100
 - **Button 4 ON** = (24Bit) Binary: 000101010101010001010101
 - **Button 4 OFF** = (24Bit) Binary: 000101010101010001010100

Save your Pulse Length: 416 Microseconds.

Save your Protocol: 1.

You'll need your binary codes, pulse length and protocol in the next Unit.

Unit 2 - Controlling Lamps and Outlets with Arduino using MQTT

This Unit shows you how to control outlets with your Arduino through the Node-RED Dashboard.

You can open this link or scroll down this page to see the complete sketch and follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_arduino_outlets.ino

Understanding How the Sketch Works

Let's see how the new snippets of code work.

Preparing RC Switch

Loads the RCSwitch library:

```
#include <RCSwitch.h>
```

Creates the mySwitch variable:

```
RCSwitch mySwitch = RCSwitch();
```

In the `setup()` function you set Digital pin 6 as a transmitter:

```
mySwitch.enableTransmit(6);
```

Set your pulse length:

```
mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);
```

Set your protocol:

```
mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);
```

Subscribing to 2 topics

In the `reconnect()` function, you subscribe to 2 topics that identify where your Arduino is located and which outlet you would like to control:

```
client.subscribe("home/livingroom/arduino/outlet1");
client.subscribe("home/livingroom/arduino/outlet2");
```

Important: I recommend that you first try to follow this Unit with these exact same topics. However, the idea is to replace them later with topics that clearly identify where your Arduino is located.

Sending the binary codes

In the `callback()` function, you create 2 `if` statements for each outlet that send the binary code to turn the outlet on/off.

You have to replace those binary codes with your own:

```

// If a message is received on the topic home/livingroom/arduino/outletX,
// you check if the message is either 1 or 0. Turns the outlet according to the message
if(String(topic)=="home/livingroom/arduino/outlet1"){
    Serial.print("Changing outlet 1 to ");
    if(messageTemp == "1"){
        mySwitch.send("000101010101000101010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("000101010101000101010100");
        Serial.print("Off");
    }
}
if(String(topic)=="home/livingroom/arduino/outlet2"){
    Serial.print("Changing outlet 2 to ");
    if(messageTemp == "1"){
        mySwitch.send("0001010101010001010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("0001010101010001010100");
        Serial.print("Off");
    }
}

```

Important: in case you change the topics, you also have to change the if statements to match the topics that your Arduino is subscribed.

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your Arduino board (after adding the appropriate IP, the RPi IP address and the binary codes for your outlets).

```

*****
All the resources for this project:  

https://rntlab.com/
*****  

// Loading the libraries  

#include <SPI.h>

```

```

#include <Ethernet.h>
#include <PubSubClient.h>
#include <RCSwitch.h>

// This MAC address can remain the same
byte mac[] = { 0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };

/*
 * IMPORTANT!
 * YOU MUST UPDATE THESE NEXT TWO VARIABLES WITH VALUES SUITABLE TO YOUR
NETWORK!
*
*/
// Replace with an IP that is suitable for your network. I could use any IP in this range: 192.168.1.X
// I've used a tool http://angryip.org/ to see an available IP address and I ended up using 192.168.1.99
IPAddress ip(192, 168, 1, 99);

// Replace with your Raspberry Pi IP Address. In my case, the RPi IP Address is 192.168.1.76
IPAddress server(192, 168, 1, 76);

// Initializes the ethClient. You have to change the ethClient name if you have multiple ESPs running in your home
automation system
EthernetClient ethClient;
PubSubClient client(ethClient);

RCSwitch mySwitch = RCSwitch();

// This function is executed when some device publishes a message to a topic that your ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
}

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

```

```

// If a message is received on the topic home/livingroom/arduino/outletX,
// you check if the message is either 1 or 0. Turns the outlet according to the message
if(String(topic)=="home/livingroom/arduino/outlet1"){
    Serial.print("Changing outlet 1 to ");
    if(messageTemp == "1"){
        mySwitch.send("000101010101000101010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("000101010101000101010100");
        Serial.print("Off");
    }
}
if(String(topic)=="home/livingroom/arduino/outlet2"){
    Serial.print("Changing outlet 2 to ");
    if(messageTemp == "1"){
        mySwitch.send("000101010101000101010101");
        Serial.print("On");
    }
    else if(messageTemp == "0"){
        mySwitch.send("000101010101000101010100");
        Serial.print("Off");
    }
}
Serial.println();
}

```

```

// This functions reconnects your Arduino to your MQTT broker
// Change the function below if you want to subscribe to more topics with your Arduino
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("arduinoClient")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("home/livingroom/arduino/outlet1");
            client.subscribe("home/livingroom/arduino/outlet2");
        } else {

```

```

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

}

}

// The setup function sets your Arduino pins as a transmitter, starts the serial communication at a baud rate of 57600
// Sets your mqtt broker and sets the callback function. The callback function is what receives messages and
// actually controls the LEDs. Finally starts the Ethernet communication
void setup()
{
mySwitch.enableTransmit(6);

// SET YOUR PULSE LENGTH
mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);

// SET YOUR PROTOCOL (default is 1, will work for most outlets)
mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);

// Set number of transmission repetitions.
mySwitch.setRepeatTransmit(15);
Serial.begin(57600);

client.setServer(server, 1883);
client.setCallback(callback);

Ethernet.begin(mac, ip);
// Allow the hardware to sort itself out
delay(1500);
}

// For this project, you don't need to change anything in the loop function.
// Basically it ensures that your Arduino is connected to your broker
void loop()
{
if (!client.connected()) {
  reconnect();
}
if(!client.loop())

```

```
    client.connect("ethClient");  
}
```

[DOWNLOAD](#) [SOURCE](#) [CODE](#)

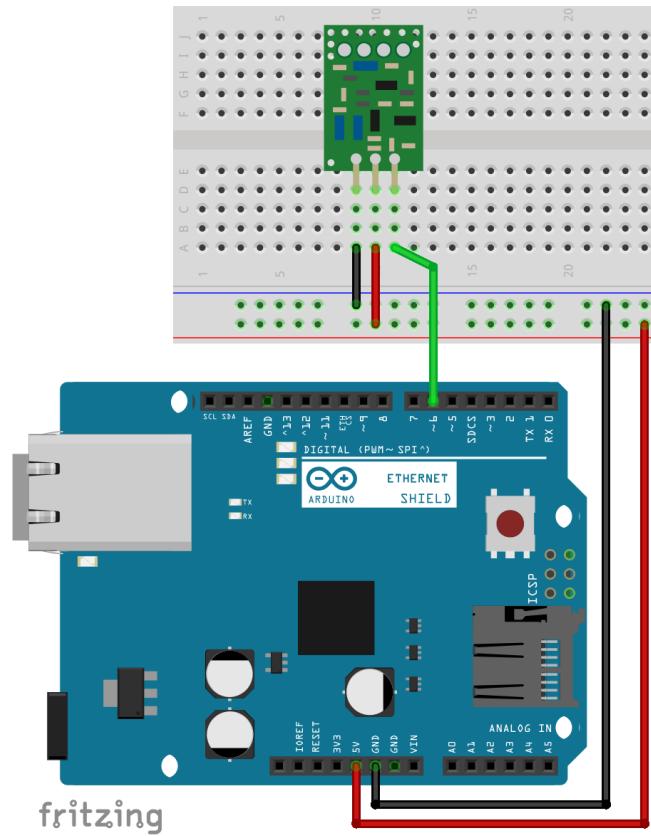
https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_arduino_outlets.ino

Schematics

To complete this Unit you need:

- 1x Arduino UNO
 - 1x Ethernet Shield (WIZnet W5100)
 - 1x 433MHz Transmitter

Here's the schematics:



Creating the Flow

In this flow, you're going to add two node **Switches** to control the outlets. Follow these next 8 steps to create your flow:

1 – Drag 4 nodes



2 – Configure Switch node

The screenshot shows the 'Edit switch node' configuration dialog. At the top right are 'Cancel' and 'Done' buttons. The configuration fields are as follows:

- Group:** Group 1 [Living Room] (dropdown menu with edit icon)
- Size:** auto
- Label:** Outlet #1 (with edit icon)
- Icon:** Default (dropdown menu)
- When clicked, send:**
 - On Payload:** a_z 1 (dropdown menu)
 - Off Payload:** a_z 0 (dropdown menu)
 - Topic:** (empty input field)
- Name:** Outlet #1 (input field)

3 – MQTT out node

Edit mqtt out node

Cancel Done

Server localhost:1883

Topic home/livingroom/arduino/outlet1

QoS Retain

Name home/livingroom/arduino/outlet1

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

4 – Configure Switch node

Edit switch node

Cancel Done

Group Group 1 [Living Room]

Size auto

Label Outlet #2

Icon Default

When clicked, send:

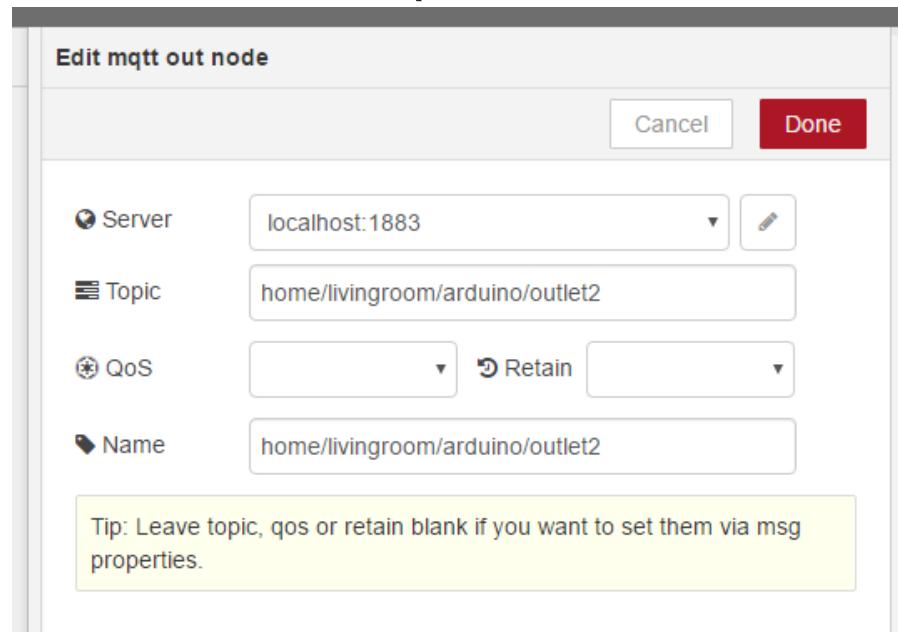
On Payload

Off Payload

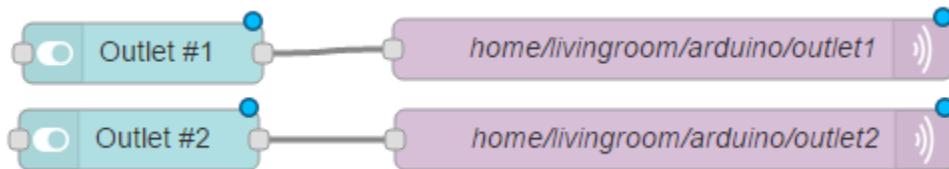
Topic

Name Outlet #2

5 – MQTT out node



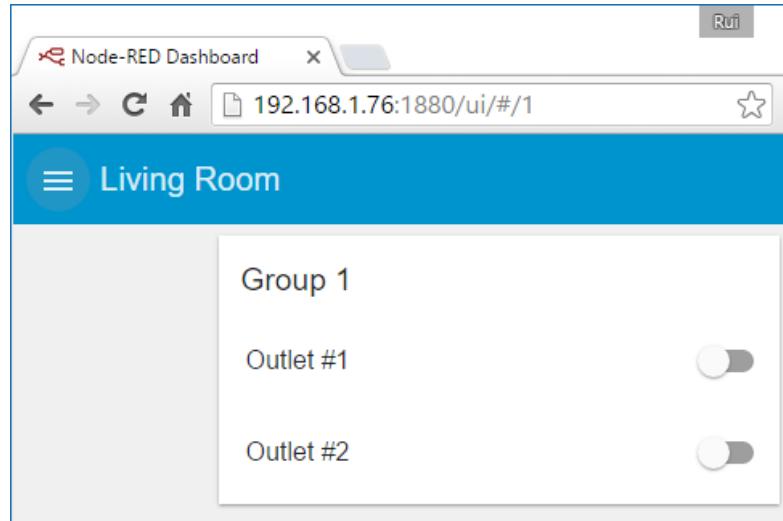
6 – Final Flow



7 – Deploy your application

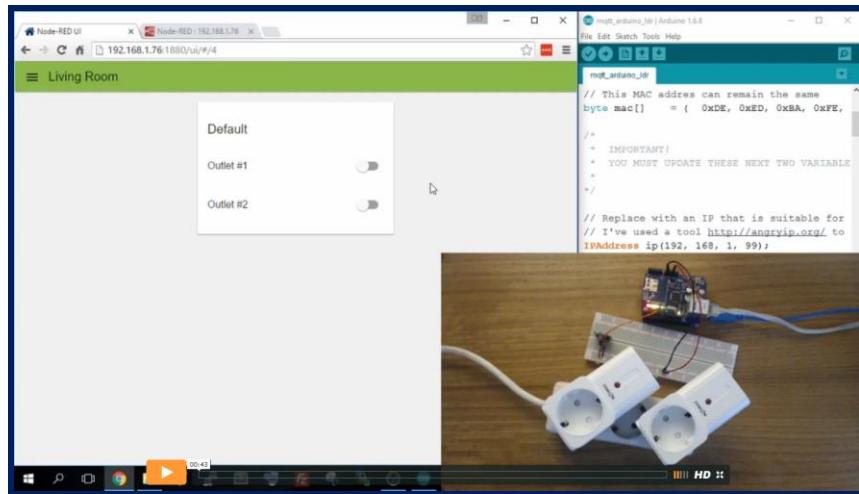


8 – Dashboard



Watch the Video Demonstration

Finally, you can control any outlet in your home with any device that has a web browser. Watch this video to see it in action:



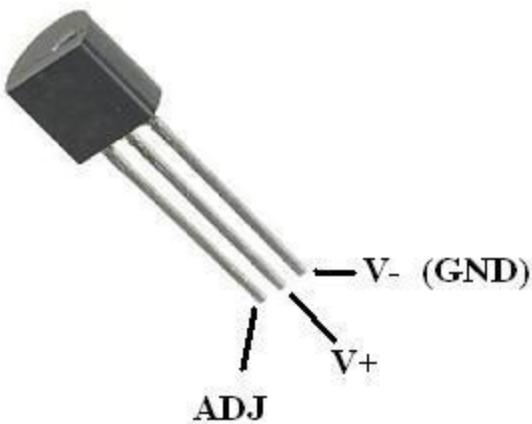
Video # 24 - <https://rntlab.com/28hasvideos>

Unit 3 - Plotting the Temperature in a Chart

(You should keep the sketch, circuit and nodes from the previous Unit and add this new sensor to your project)

This Unit shows you how to read temperature values with an Arduino and how to publish them to a chart in the Node-RED Dashboard.

Throughout this Unit, you're going to use the LM335 sensor, but if you don't have this exact sensor you can use another temperature sensor with some minor code changes.



Understanding How the Sketch Works

You can open this link or scroll down this page to see the complete sketch and follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_arduino_temperature.ino

Let's see how the new snippets of code work.

Preparing the variables

Assign the Analog pin A0 to the `tempAnalogPin` variable.

```
// Defines the analog temperature pin for the LM335 temperature sensor
int tempAnalogPin = A0;
```

The next variable is used to create a timer that checks the temperature value every few milliseconds.

```
// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;
```

loop

The `loop()` function has a timer that compares the time that has passed to the last sensor measurement.

If it has passed 10 seconds, it goes into that `if` statement.

```
now = millis();
// Publishes new temperature and humidity every 10 seconds
if (now - lastMeasure > 10000) {
    lastMeasure = now;
```

These 5 lines of code read the value of the sensor pin, convert the value to millivolts and convert millivolts to degrees Kelvin.

```
// Reads analog pin and computes temperature values in Kelvin
int rawVoltage= analogRead(tempAnalogPin);
float millivolts= (rawVoltage/1024.0) * 5000;
float kelvin= (millivolts/10);
Serial.print(kelvin);
Serial.println(" degrees Kelvin");
```

The next snippet of code converts degrees Kelvin to Celsius and saves the temperature in a temporary variable.

Tip: you can comment the Celsius code and uncomment the Fahrenheit code.

```
// Converts temperature in Kelvin to Celsius, you can comment  
// and uncomment the lines below if you prefer Fahrenheit temperature  
float celsius = kelvin - 273.15;  
Serial.print(celsius);  
Serial.println(" degrees Celsius");  
static char temperatureTemp[6];  
dtostrf(celsius, 6, 2, temperatureTemp);  
  
// Uncomment to convert temperature in Kelvin to Fahrenheit  
/*  
float fahrenheit = (((kelvin - 273.15) * 9)/5 +32);  
Serial.print(fahrenheit);  
Serial.println(" degrees Fahrenheit");  
static char temperatureTemp[6];  
dtostrf(fahrenheit, 6, 2, temperatureTemp);*/
```

Finally, you need to publish the new temperature reading in Celsius to the Node-RED Dashboard chart:

- `home/livingroom/arduino/temperature`

```
// Publishes a new Temperature value  
client.publish("home/livingroom/arduino/temperature", temperatureTemp);
```

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your Arduino board (after adding the appropriate IP, the RPi IP address and the binary codes for your outlets)..

```
*****
```

All the resources for this project:

<https://rntlab.com/>

*****/

// Loading the libraries

```
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>
#include <RCSwitch.h>
```

// This MAC address can remain the same

```
byte mac[] = { 0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };
```

/*

* IMPORTANT!

* YOU MUST UPDATE THESE NEXT TWO VARIABLES WITH VALUES SUITABLE TO YOUR
NETWORK!

*

*/

// Replace with an IP that is suitable for your network. I could use any IP in this range: 192.168.1.X

// I've used a tool <http://angryip.org/> to see an available IP address and I ended up using 192.168.1.99

```
IPAddress ip(192, 168, 1, 99);
```

// Replace with your Raspberry Pi IP Address. In my case, the RPi IP Address is 192.168.1.76

```
IPAddress server(192, 168, 1, 76);
```

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

```
EthernetClient ethClient;
```

```
PubSubClient client(ethClient);
```

```
RCSwitch mySwitch = RCSwitch();
```

// Defines the analog temperature pin for the LM335 temperature sensor

```
int tempAnalogPin = A0;
```

// Timers auxiliar variables

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

```
long lastMeasure = 0;
```

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

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

```

// your ESP8266 is subscribed you can actually do something
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
    // Feel free to add more if statements to control more outlets with MQTT

    // If a message is received on the topic home/livingroom/arduino/outletX,
    // you check if the message is either 1 or 0. Turns the outlet according to the message
    if(String(topic)=="home/livingroom/arduino/outlet1"){
        Serial.print("Changing outlet 1 to ");
        if(messageTemp == "1"){
            mySwitch.send("000101010101000101010101");
            Serial.print("On");
        }
        else if(messageTemp == "0"){
            mySwitch.send("0001010101010001010100");
            Serial.print("Off");
        }
    }
    if(String(topic)=="home/livingroom/arduino/outlet2"){
        Serial.print("Changing outlet 2 to ");
        if(messageTemp == "1"){
            mySwitch.send("0001010101010001010101");
            Serial.print("On");
        }
        else if(messageTemp == "0"){
            mySwitch.send("0001010101010001010100");
            Serial.print("Off");
        }
    }
    Serial.println();
}

// This functions reconnects your Arduino to your MQTT broker

```

```

// Change the function below if you want to subscribe to more topics with your Arduino
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection... ");
        // Attempt to connect
        if (client.connect("arduinoClient")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("home/livingroom/arduino/outlet1");
            client.subscribe("home/livingroom/arduino/outlet2");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

// The setup function sets your Arduino pins as a transmitter, starts the serial communication at a baud rate of 57600
// Sets your mqtt broker and sets the callback function. The callback function is what receives messages and
// actually controls the LEDs. Finally starts the Ethernet communication
void setup()
{
    mySwitch.enableTransmit(6);
    // SET YOUR PULSE LENGTH
    mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);

    // SET YOUR PROTOCOL (default is 1, will work for most outlets)
    mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);

    // Set number of transmission repetitions.
    mySwitch.setRepeatTransmit(15);

    Serial.begin(57600);

    client.setServer(server, 1883);
    client.setCallback(callback);

    Ethernet.begin(mac, ip);
}

```

```

// Allow the hardware to sort itself out
delay(1500);
}

// For this project, you don't need to change anything in the loop function.
// Basically it ensures that your Arduino is connected to your broker
void loop(){
if (!client.connected()) {
  reconnect();
}
if(!client.loop())
  client.connect("ethClient");
now = millis();
// Publishes new temperature and humidity every 10 seconds
if (now - lastMeasure > 10000) {
  lastMeasure = now;

  // Reads analog pin and computes temperature values in Kelvin
  int rawVoltage= analogRead(tempAnalogPin);
  float millivolts= (rawVoltage/1024.0) * 5000;
  float kelvin= (millivolts/10);
  Serial.print(kelvin);
  Serial.println(" degrees Kelvin");

  // Converts temperature in Kelvin to Celsius, you can comment
  // and uncomment the lines below if you prefer Fahrenheit temperature
  float celsius = kelvin - 273.15;
  Serial.print(celsius);
  Serial.println(" degrees Celsius");
  static char temperatureTemp[7];
  dtostrf(celsius, 6, 2, temperatureTemp);

  // Uncomment to convert temperature in Kelvin to Fahrenheit
  /*
  float fahrenheit = (((kelvin - 273.15) * 9)/5 +32);
  Serial.print(fahrenheit);
  Serial.println(" degrees Fahrenheit");
  static char temperatureTemp[7];
  dtostrf(fahrenheit, 6, 2, temperatureTemp);*/

  // Publishes a new Temperature value
  client.publish("home/livingroom/arduino/temperature", temperatureTemp);
}

```

```
}
```

DOWNLOAD SOURCE CODE

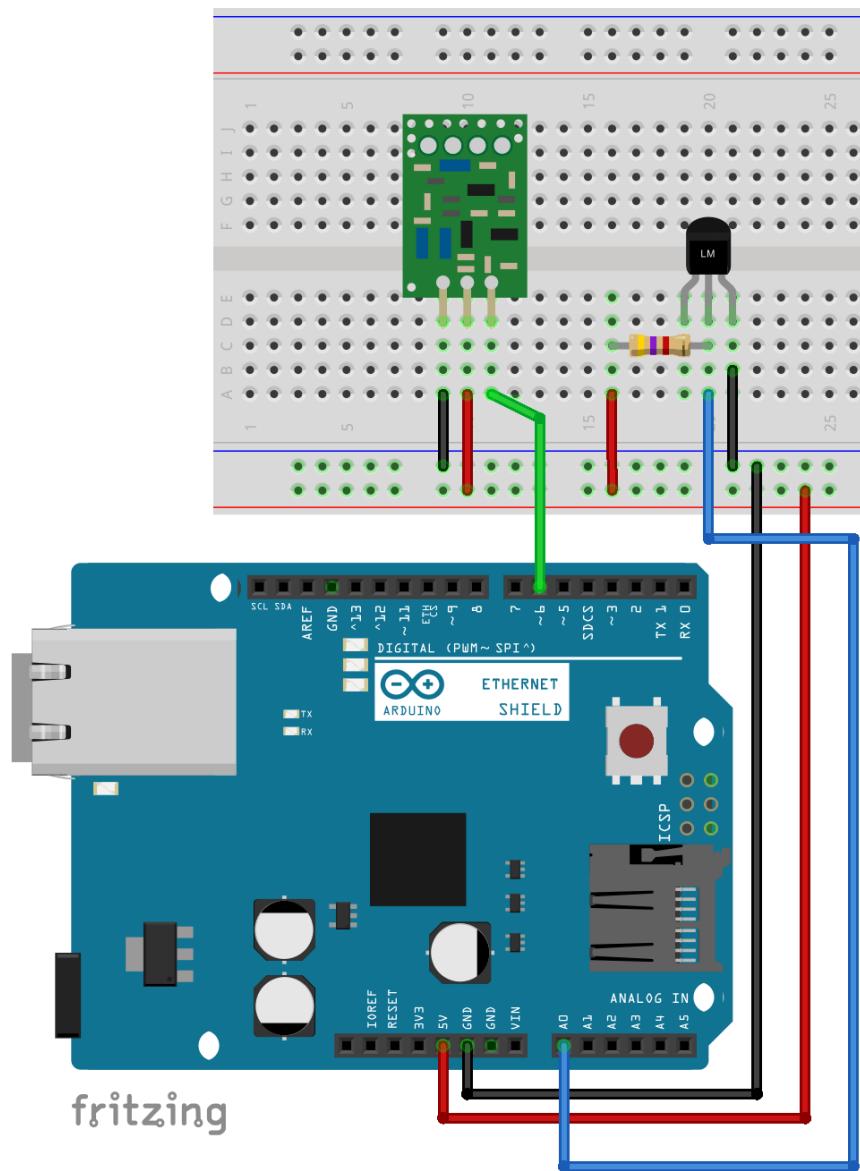
https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_arduino_temperature.ino

Schematics

To complete this Unit you need:

- 1x Arduino UNO
- 1x Ethernet Shield (WIZnet W5100)
- 1x 433MHz Transmitter
- 1x LM335 Temperature Sensor
- 1x 4700 Ohm Resistor

Here's the schematics:



Creating the Flow

In this flow, you're going to add a node **Chart** to plot the temperature values. Follow these next 6 steps to create your flow:

1 – Drag 2 nodes



2 – MQTT in node to receive temperature

Edit mqtt in node

Cancel Done

Server	localhost:1883	
Topic	home/livingroom/arduino/temperature	
QoS	2	
Name	home/livingroom/arduino/temperature	

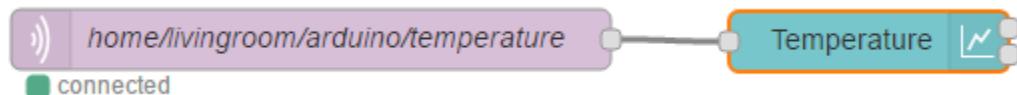
3 – Configure the Chart node

Edit chart node

Cancel Done

Group	Group 1 [Living Room]	
Size	auto	
Label	Temperature	
X-axis	last 1	hours
Y-axis	min	max
Interpolate	linear	
Blank label	display this text before valid data arrives	
Name	Temperature	

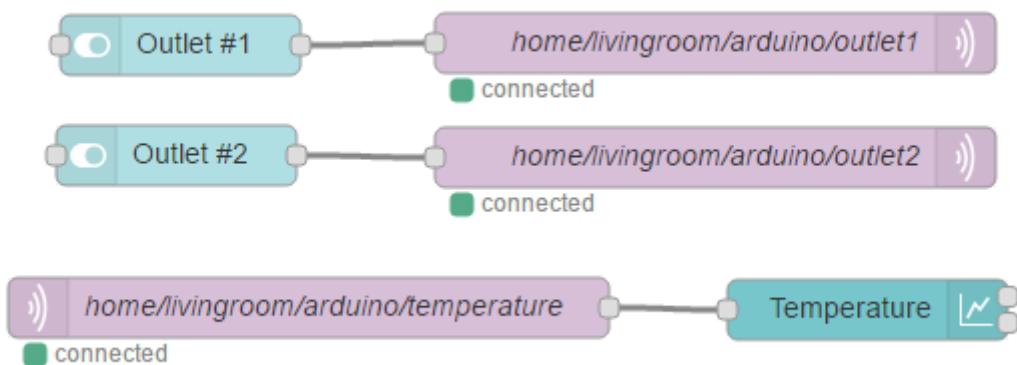
4 – Connect the nodes



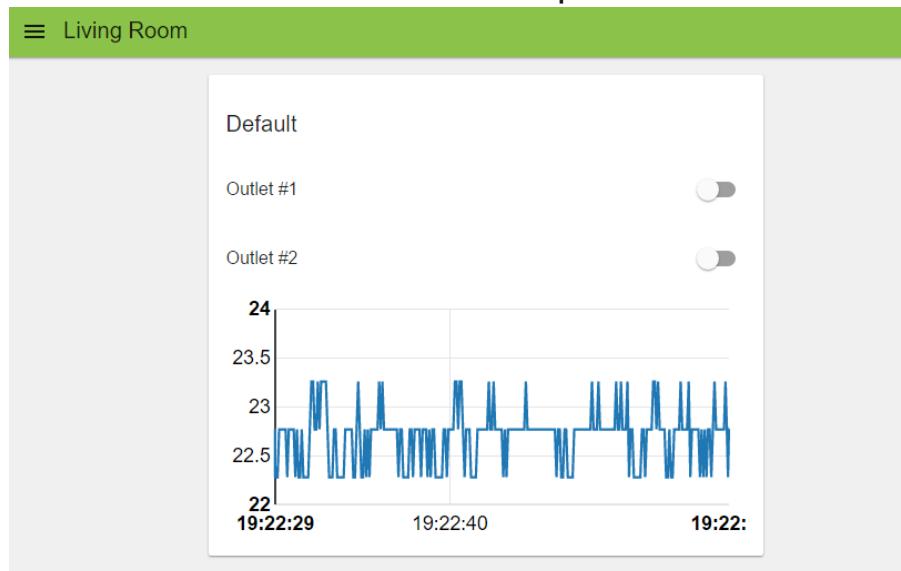
5 – Deploy your application



Final Flow

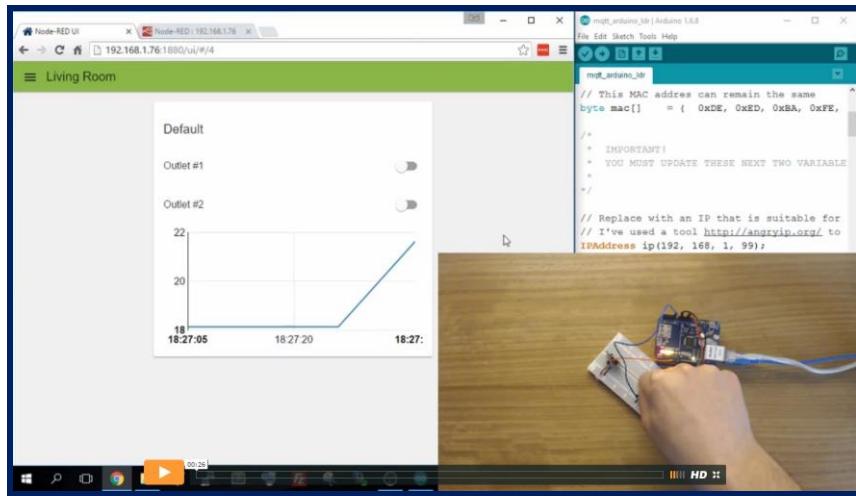


6 – Dashboard with the temperature Chart



Watch the Video Demonstration

Now, you can see the temperature in your room displayed in a Chart. Watch this video to see it in action:



Video # 25 - <https://rntlab.com/28hasvideos>

Unit 4 - Reading the Light Intensity

(You should keep the sketch, circuit and nodes from the previous Unit and add this new sensor to your project)

This Unit shows you how to use a light-dependent resistor to measure the light sensitivity in your room with an Arduino and how to publish those values to a gauge.

A light-dependent resistor (LDR) is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity.



Understanding How the Sketch Works

You can open this link or scroll down this page to see the complete sketch and follow along:

- https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_arduino_ldr.ino

Let's see how the new snippets of code work.

Preparing the variables

Assign the Analog pin A1 to the `LDRAnalogPin` variable.

```
// Defines the analog pin for the LDR
int LDRAnalogPin = A1;
```

This following variable is used to create a timer that checks the LDR value every few milliseconds.

```
long lastLDRCheck = 0;
```

loop

In the `loop()` function you keep checking every 1000 milliseconds the current value of the LDR sensor:

```
if ((now - lastLDRCheck) > 1000) {
```

If 1 second has passed since the last check, it reads the LDR value and publishes the value to the Node-RED Dashboard.

```
if ((now - lastLDRCheck) > 1000) {
    lastLDRCheck = now;
    int LDRValue = analogRead(LDRAnalogPin);
    Serial.print("LDR Value: ");
    Serial.println(LDRValue);

    static char LDRTemp[6];
    dtostrf(LDRValue, 5, 0, LDRTemp);

    // Publishes a new LDR value
    client.publish("home/livingroom/arduino/ldr", LDRTemp);
}
```

Uploading the Final Sketch

You can copy the final sketch to your Arduino IDE and upload it to your Arduino board (after adding the appropriate IP, the RPi IP address and the binary codes for your outlets).

```

*****  

All the resources for this project:  

https://rntlab.com/  

*****/  

// Loading the libraries  

#include <SPI.h>  

#include <Ethernet.h>  

#include <PubSubClient.h>  

#include <RCSwitch.h>  

// This MAC address can remain the same  

byte mac[] = { 0xDE, 0xED, 0xBA, 0xFE, 0xFE, 0xED };  

/*
 * IMPORTANT!
 * YOU MUST UPDATE THESE NEXT TWO VARIABLES WITH VALUES SUITABLE TO YOUR
NETWORK!  

*  

*/
  

// Replace with an IP that is suitable for your network. I could use any IP in this range: 192.168.1.X  

// I've used a tool http://angryip.org/ to see an available IP address and I ended up using 192.168.1.99  

IPAddress ip(192, 168, 1, 99);  

// Replace with your Raspberry Pi IP Address. In my case, the RPi IP Address is 192.168.1.76  

IPAddress server(192, 168, 1, 76);  

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

EthernetClient ethClient;  

PubSubClient client(ethClient);  

RCSwitch mySwitch = RCSwitch();  

// Defines the analog temperature pin for the LM335 temperature sensor  

int tempAnalogPin = A0;  

// Defines the analog pin for the LDR  

int LDRAnalogPin = A1;

```

```

// Timers auxiliar variables
long now = millis();
long lastMeasure = 0;
long lastLDRCheck = 0;

// This functions is executed when some device publishes a message to a topic that your ESP8266 is subscribed to
// Change the function below to add logic to your program, so when a device publishes a message to a topic that
// your ESP8266 is subscribed you can actually do something
void callback(char* topic, byte* message, unsigned int length) {
    Serial.print("Message arrived on topic: ");
    Serial.print(topic);
    Serial.print(". Message: ");
    String messageTemp;

    for (int i = 0; i < length; i++) {
        Serial.print((char)message[i]);
        messageTemp += (char)message[i];
    }
    Serial.println();
    // Feel free to add more if statements to control more outlets with MQTT

    // If a message is received on the topic home/livingroom/arduino/outletX,
    // you check if the message is either 1 or 0. Turns the outlet according to the message
    if(String(topic)=="home/livingroom/arduino/outlet1"){
        Serial.print("Changing outlet 1 to ");
        if(messageTemp == "1"){
            mySwitch.send("00010101010001010101");
            Serial.print("On");
        }
        else if(messageTemp == "0"){
            mySwitch.send("00010101010001010100");
            Serial.print("Off");
        }
    }
    if(String(topic)=="home/livingroom/arduino/outlet2"){
        Serial.print("Changing outlet 2 to ");
        if(messageTemp == "1"){
            mySwitch.send("00010101010100010101");
            Serial.print("On");
        }
        else if(messageTemp == "0"){
            mySwitch.send("00010101010100010100");
            Serial.print("Off");
        }
    }
}

```

```

    }
}

Serial.println();
}

// This functions reconnects your Arduino to your MQTT broker
// Change the function below if you want to subscribe to more topics with your Arduino
void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("arduinoClient")) {
            Serial.println("connected");
            // Subscribe or resubscribe to a topic
            // You can subscribe to more topics (to control more LEDs in this example)
            client.subscribe("home/livingroom/arduino/outlet1");
            client.subscribe("home/livingroom/arduino/outlet2");
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

// The setup function sets your Arduino pins as a transmitter, starts the serial communication at a baud rate of 57600
// Sets your mqtt broker and sets the callback function. The callback function is what receives messages and
// actually controls the LEDs. Finally starts the Ethernet communication
void setup()
{
    mySwitch.enableTransmit(6);

    // SET YOUR PULSE LENGTH
    mySwitch.setPulseLength(REPLACE_WITH_YOUR_PULSE_LENGTH);

    // SET YOUR PROTOCOL (default is 1, will work for most outlets)
    mySwitch.setProtocol(REPLACE_WITH_YOUR_PROTOCOL);

    // Set number of transmission repetitions.
}

```

```

mySwitch.setRepeatTransmit(15);

Serial.begin(57600);

client.setServer(server, 1883);
client.setCallback(callback);

Ethernet.begin(mac, ip);
// Allow the hardware to sort itself out
delay(1500);
}

// For this project, you don't need to change anything in the loop function.
// Basically it ensures that your Arduino is connected to your broker
void loop(){
if (!client.connected()) {
  reconnect();
}
if(!client.loop())
  client.connect("ethClient");

now = millis();
// Publishes new temperature and humidity every 10 seconds
if (now - lastMeasure > 10000) {
  lastMeasure = now;

  // Reads analog pin and computes temperature values in Kelvin
  int rawVoltage= analogRead(tempAnalogPin);
  float millivolts= (rawVoltage/1024.0) * 5000;
  float kelvin= (millivolts/10);
  Serial.print(kelvin);
  Serial.println(" degrees Kelvin");

  // Converts temperature in Kelvin to Celsius, you can comment
  // and uncomment the lines below if you prefer Fahrenheit temperature
  float celsius = kelvin - 273.15;
  Serial.print(celsius);
  Serial.println(" degrees Celsius");
  static char temperatureTemp[6];
  dtostrf(celsius, 6, 2, temperatureTemp);

  // Uncomment to convert temperature in Kelvin to Fahrenheit
  /*

```

```

float fahrenheit = (((kelvin - 273.15) * 9)/5 +32);
Serial.print(fahrenheit);
Serial.println(" degrees Fahrenheit");
static char temperatureTemp[7];
dtostrf(fahrenheit, 6, 2, temperatureTemp);/*

// Publishes a new Temperature value
client.publish("home/livingroom/arduino/temperature", temperatureTemp);
}

// Publishes new LDR value every 1000 milliseconds
if ((now - lastLDRCheck) > 1000) {
    lastLDRCheck = now;
    int LDRValue = analogRead(LDRAalogPin);
    Serial.print("LDR Value: ");
    Serial.println(LDRValue);

    static char LDRTemp[7];
    dtostrf(LDRValue, 5, 0, LDRTemp);

    // Publishes a new LDR value
    client.publish("home/livingroom/arduino/ldr", LDRTemp);
}
}

```

DOWNLOAD SOURCE CODE

https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/v2/mqtt_arduino_ldr.ino

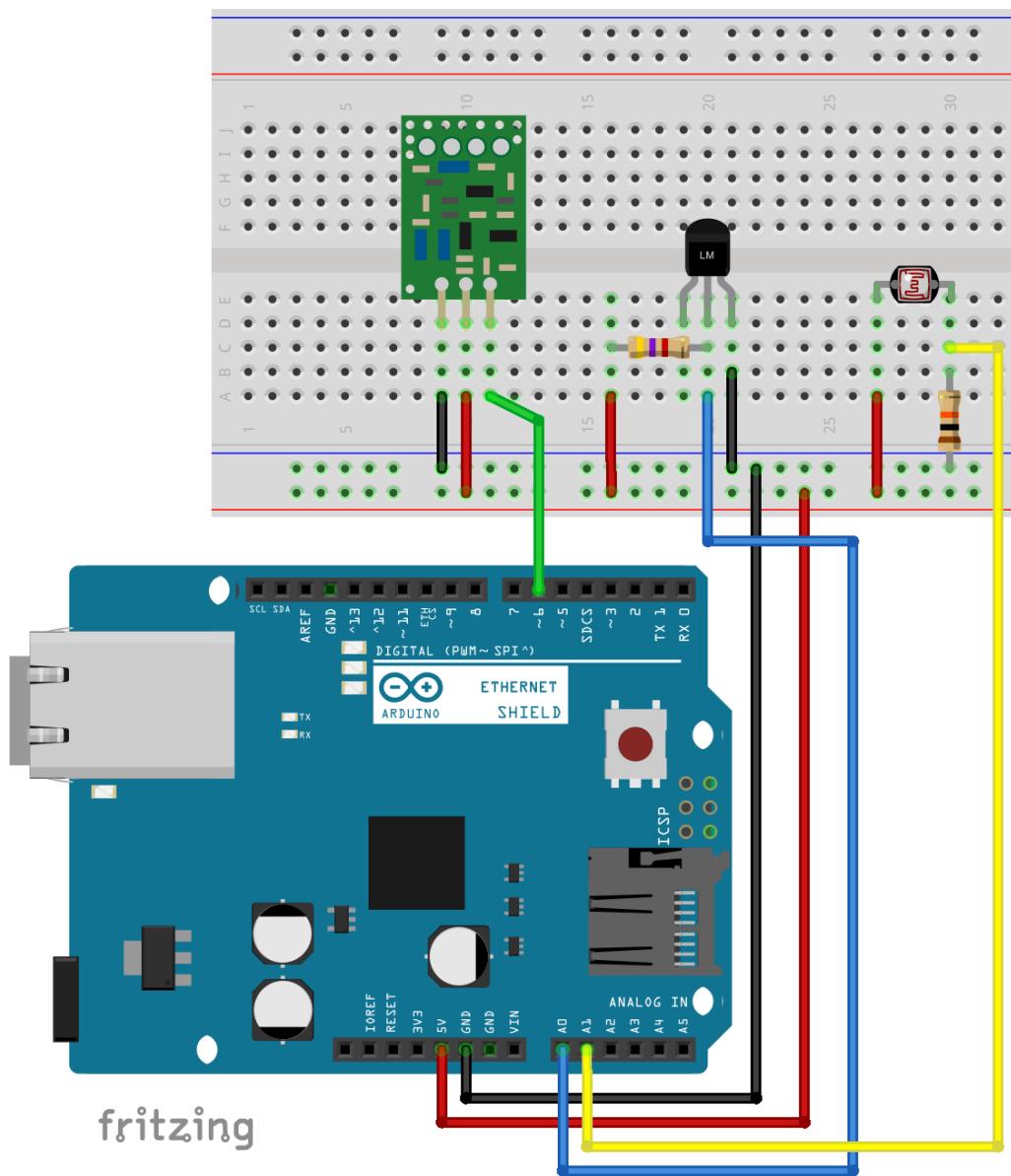
Schematics

To complete this Unit you need:

- 1x Arduino UNO
- 1x Ethernet Shield (WIZnet W5100)
- 1x 433MHz Transmitter
- 1x LM335 Temperature Sensor
- 1x 4700 Ohm Resistor

- 1x LDR
- 1x 10k Ohm Resistor

Here's the schematics:



Creating the Flow

In this flow, you're going to add a Gauge to display the light intensity value in the Node-RED Dashboard. Follow these next 6 steps to create your flow:

1 – Drag 2 nodes



2 – Edit the MQTT in node

Edit mqtt in node

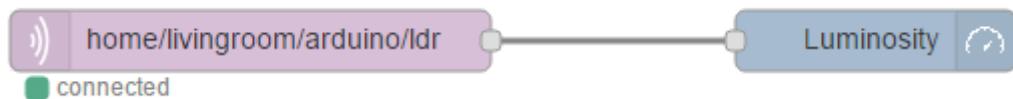
Server	localhost:1883	
Topic	home/livingroom/arduino/ldr	
Name	Name	
Ok Cancel		

3 – Configure the Gauge node

Edit ui_gauge node

Tab	Living Room	
Name	Luminosity	
Group		Order 4
Template	{{value}}	
Min	0	
Max	1000	
Ok Cancel		

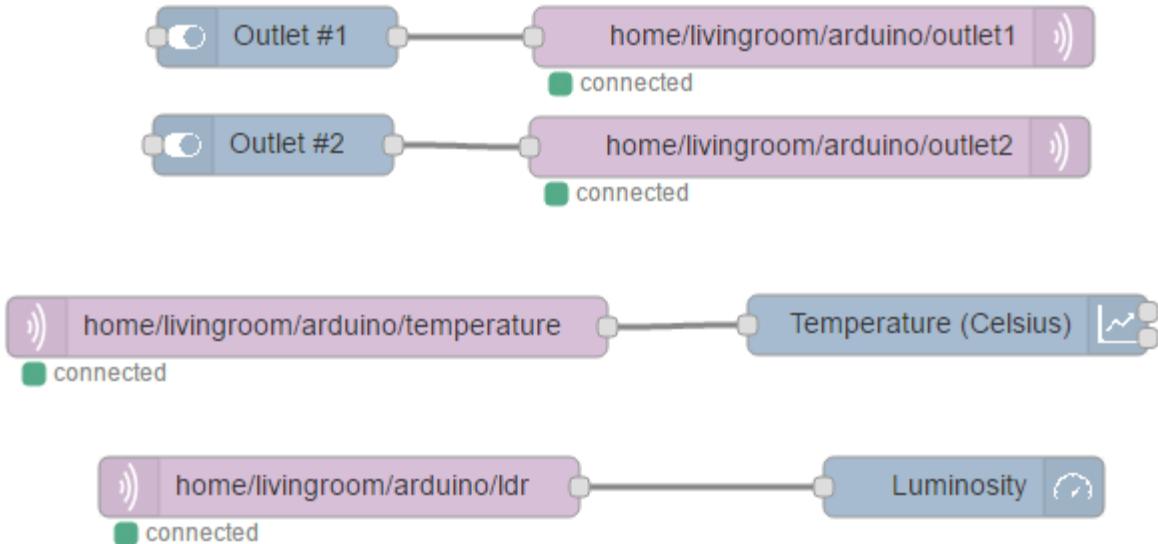
4 – Connect the nodes



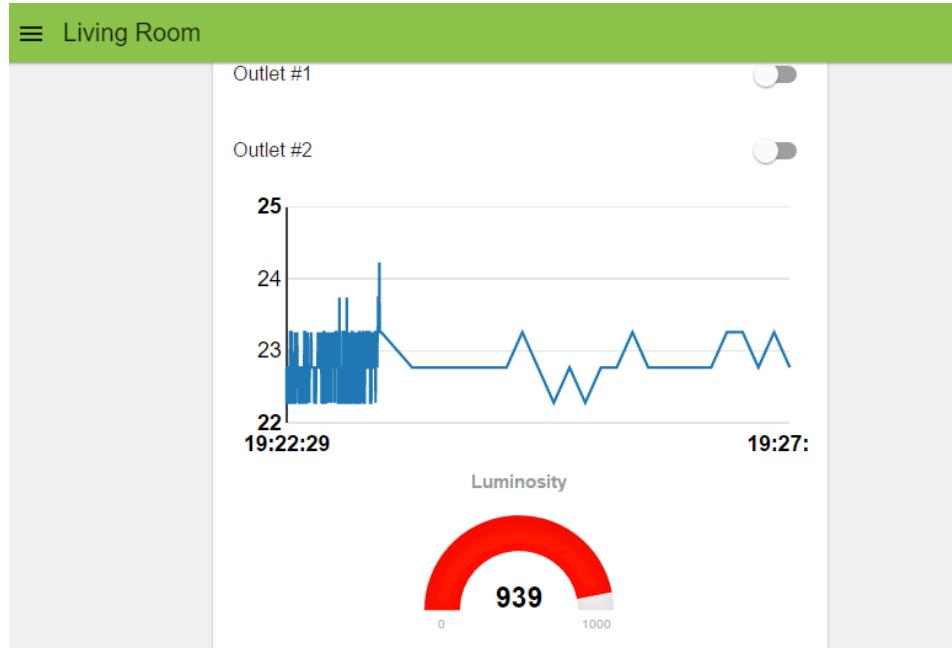
5 – Deploy your application



Final Flow

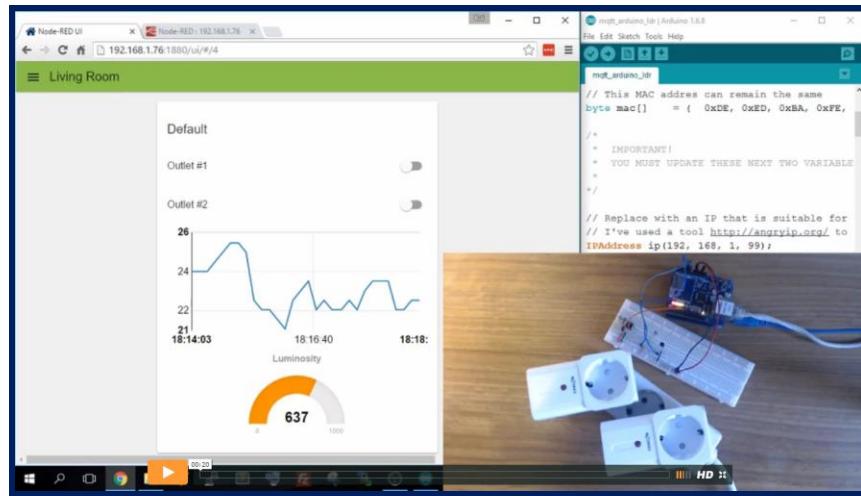


6 – Dashboard



Watch the Video Demonstration

Now, you can check the light intensity in your room displayed in a Gauge. Watch this video to see it in action:



Video # 26 - <https://rntlab.com/28hasvideos>

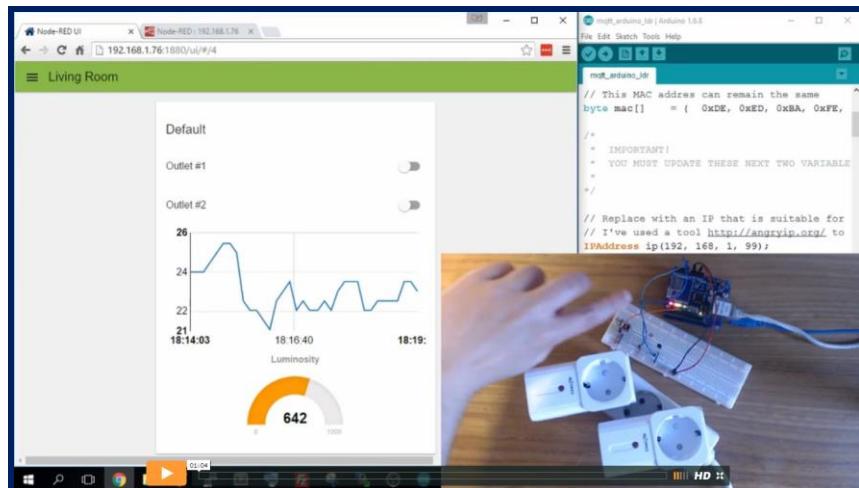
Unit 5 - Triggering Outlets with Temperature and Luminosity

(You should keep the sketch, circuit and nodes from the previous Unit and simply add this new nodes to your flow)

This Unit shows you how to trigger your outlets based on sensor values that you define. If a sensor reading goes below a certain threshold value, it automatically turns your light on or off.

Watch the Video Demonstration

Watch this next video to see the outlets being triggered according to the light intensity in your room:

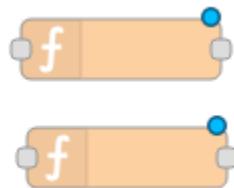


Video # 27 - <https://rntlab.com/28hasvideos>

Creating the Flow

In this flow, you're going to turn your outlets on/off according to a threshold value.

Start by dragging two Function nodes:



Temperature trigger

Double-click one of the Function nodes:

Edit function node

Name Temperature Trigger 

Function

```
1 if (parseFloat(msg.payload) < 15) {  
2     msg.payload = 1;  
3     return msg;  
4 }  
5  
6 else {  
7     msg.payload = 0;  
8     return msg;  
9 }
```

Outputs 1 

See the Info tab for help writing functions.

Ok **Cancel**

This screenshot shows the 'Edit function node' dialog box. It contains fields for 'Name' (set to 'Temperature Trigger'), a code editor for the 'Function' logic, and settings for 'Outputs'. The function code itself is a simple conditional statement that checks if the payload is less than 15. If true, it sets the payload to 1; otherwise, it sets it to 0. A note at the bottom suggests checking the 'Info' tab for help writing functions. At the bottom right are 'Ok' and 'Cancel' buttons.

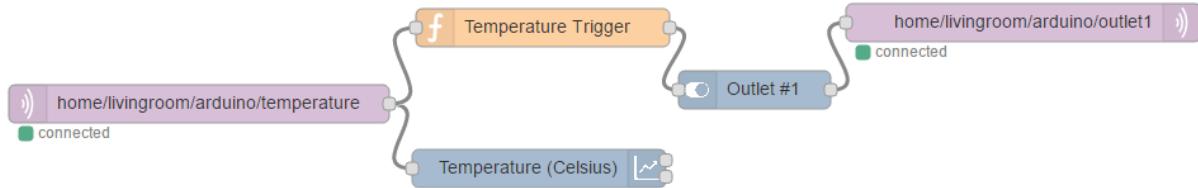
Add the following code in the function field (the value highlighted in red is the temperature threshold value):

```
if (parseFloat(msg.payload) < 15) {  
    msg.payload = 1;  
    return msg;  
}  
else {  
    msg.payload = 0;  
    return msg;  
}
```

If the temperature reading goes under **15** degrees, it will automatically turn the outlet on.

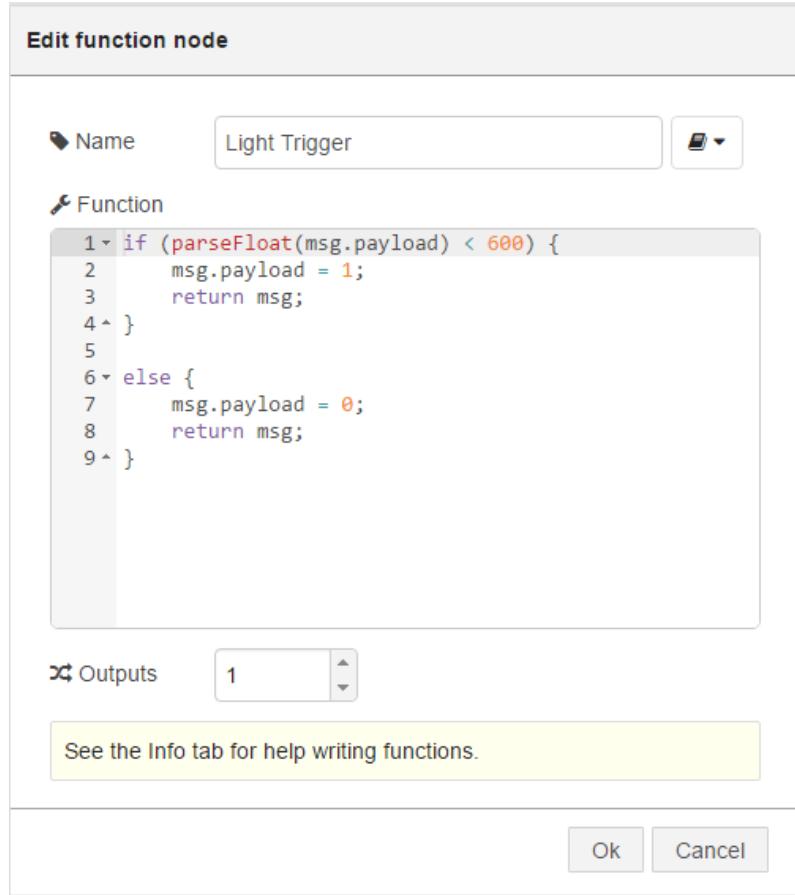
You can attach a heater to that outlet that will be on for as long as the temperature in your room is under **15** degrees.

Connect your nodes as follows:



Light trigger

Double-click the other **Function** node:



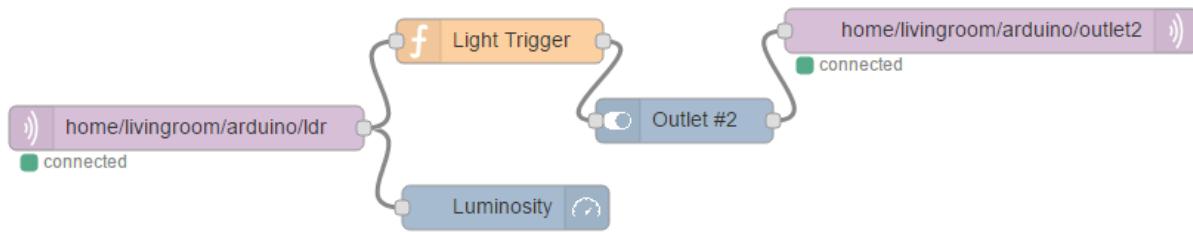
Add the following code in the function field (the value highlighted in red is the light threshold value):

```
if (parseFloat(msg.payload) < 600) {  
    msg.payload = 1;  
    return msg;  
}  
else {  
    msg.payload = 0;  
    return msg;  
}
```

If the light intensity is below **600**, it will automatically turn the outlet on.

You can attach a lamp to the outlet, that will be on when there's no light in your room.

Connect your nodes as shown in the next figure:

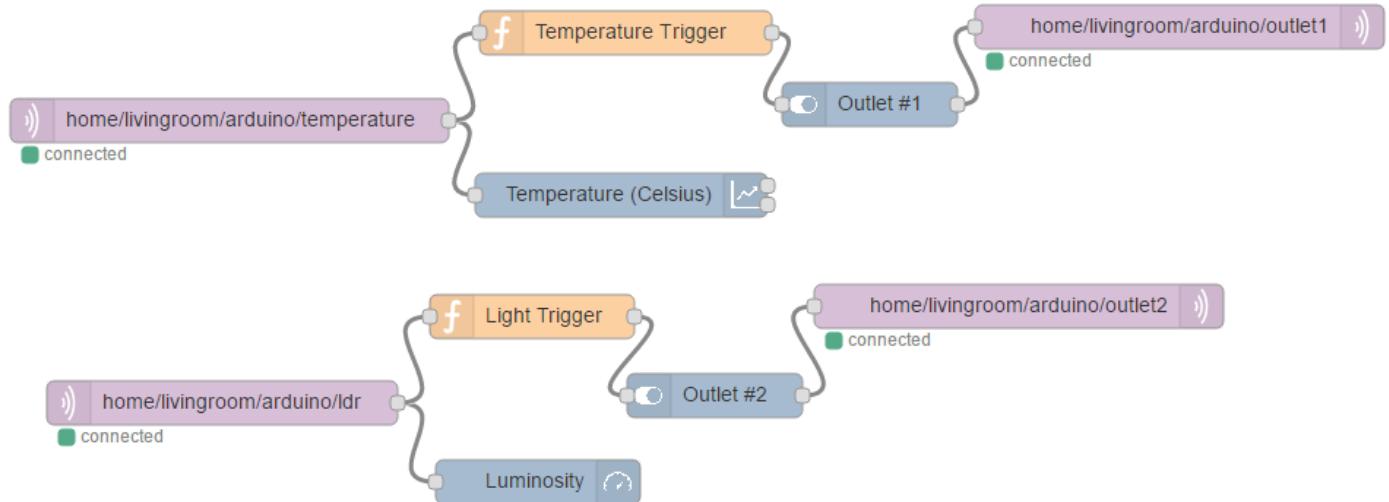


Deploying your application

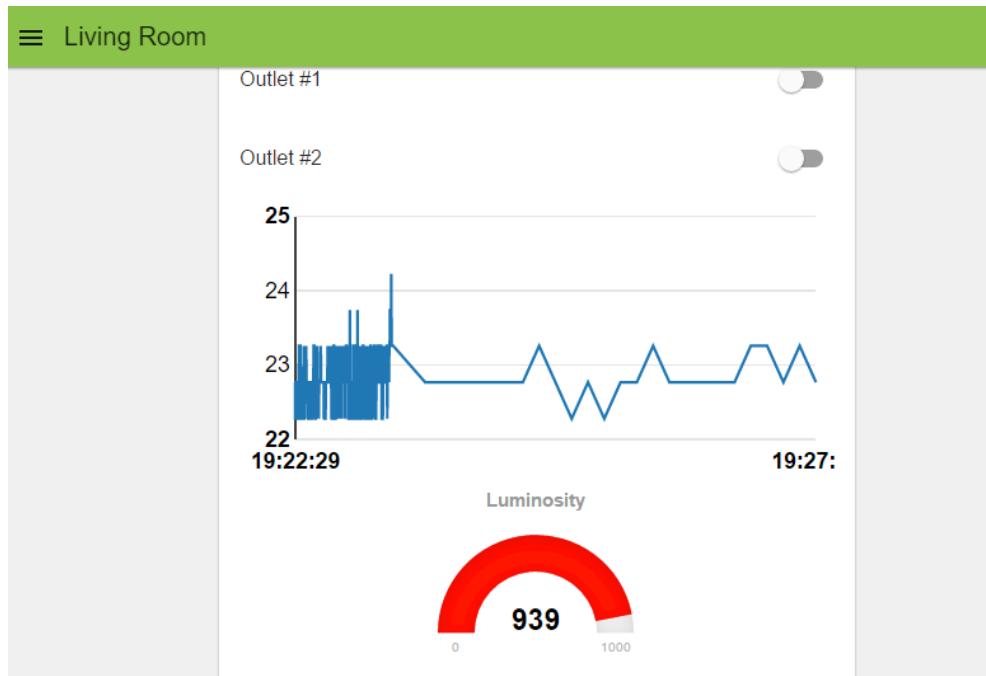
Click the Deploy button to save your application.



Here's how the final flow looks:

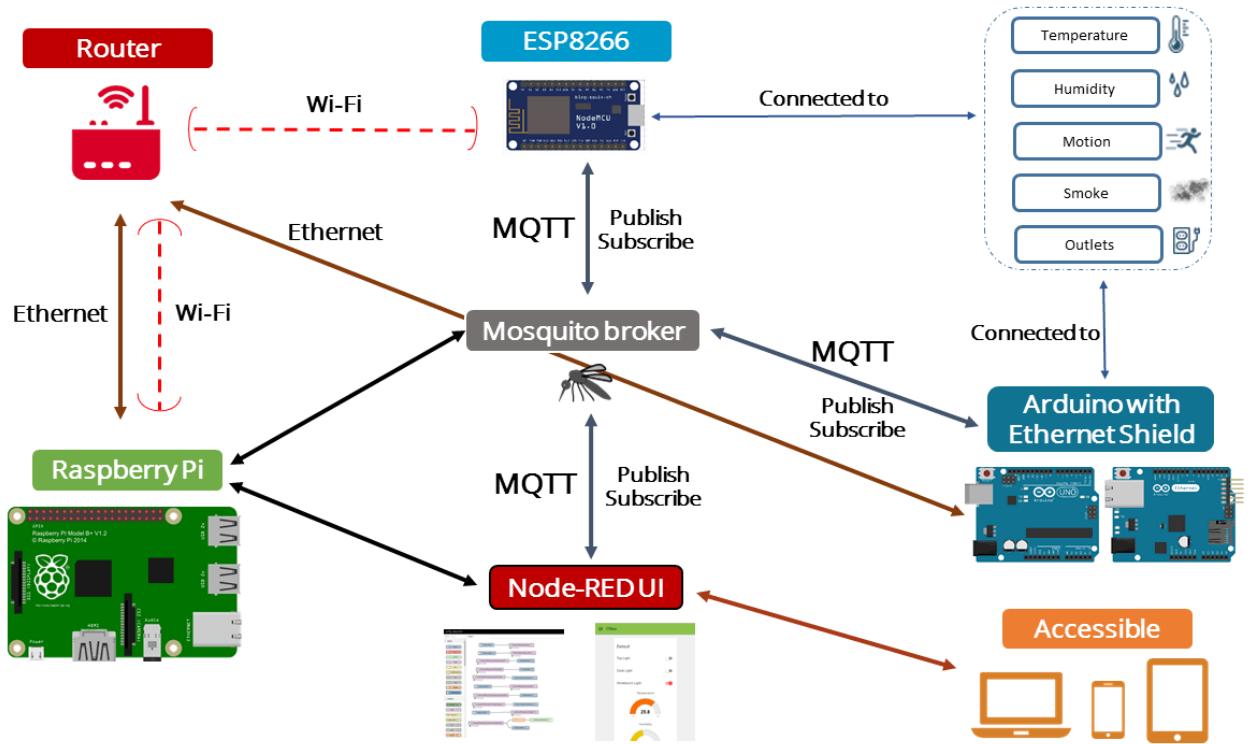


The Node-RED Dashboard keeps the same look from the previous Unit:



Final Diagram

To sum up, here's a diagram that illustrates how everything is connected:



The Arduino is able to:

- Control outlets
- Read the room temperature and plot those readings in a chart
- Read the light intensity
- Trigger events based on certain sensor readings

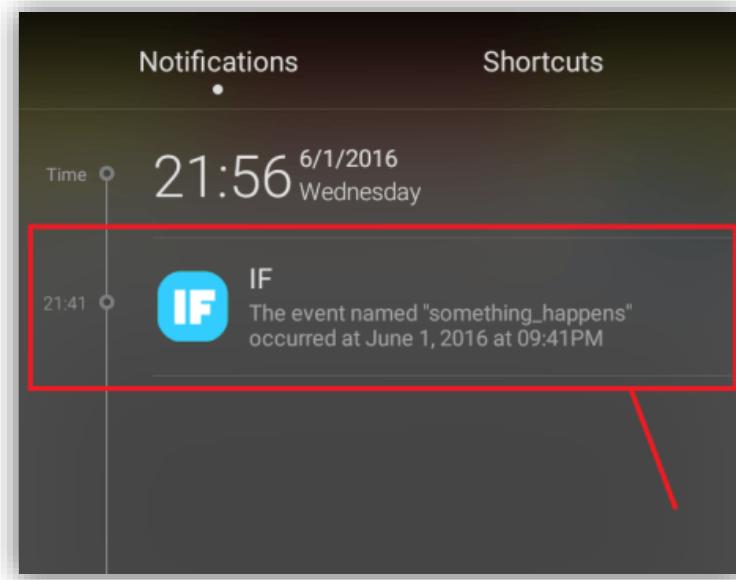
The ESP8266 is able to:

- Control outlets
- Read the temperature and humidity
- Check the room for smoke
- Detect motion and send email notifications

This course demonstrates how to add one Arduino and one ESP8266 to your Home Automation system. However, you can add as many as you want and extend this project to other rooms in your home.

Module 12

Adding Rules and Triggering Events



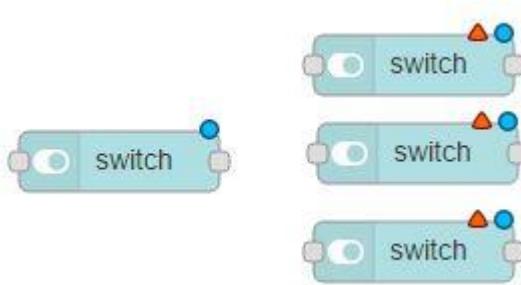
Unit 1 - Creating Master Switches or Modes

In any home automation system, it's useful to create specific modes tailored for your home. For instance, it's handy to press a button that turns on/off all the lights in your home.

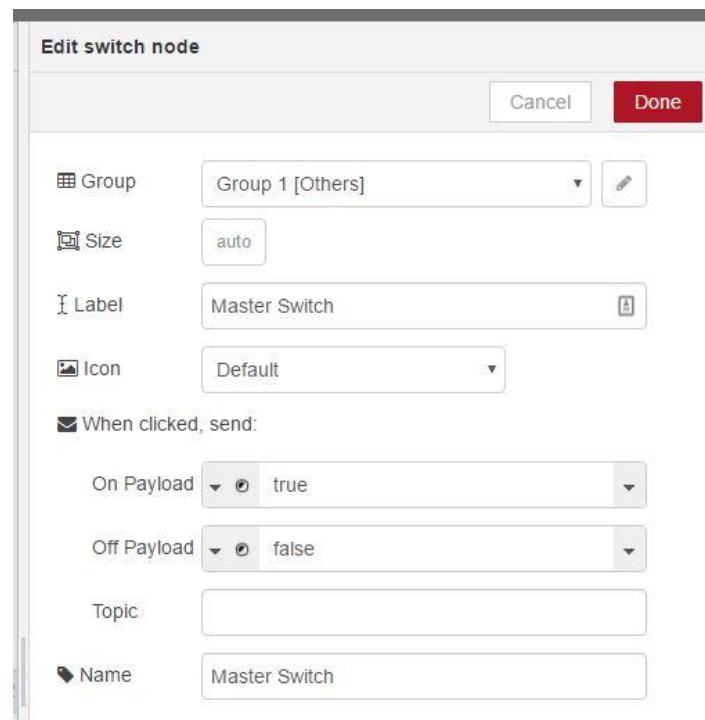
With Node-RED this can be accomplished with what I call a master switch.

Creating Your Flow

First drag 4 **Switch** nodes to your flow:

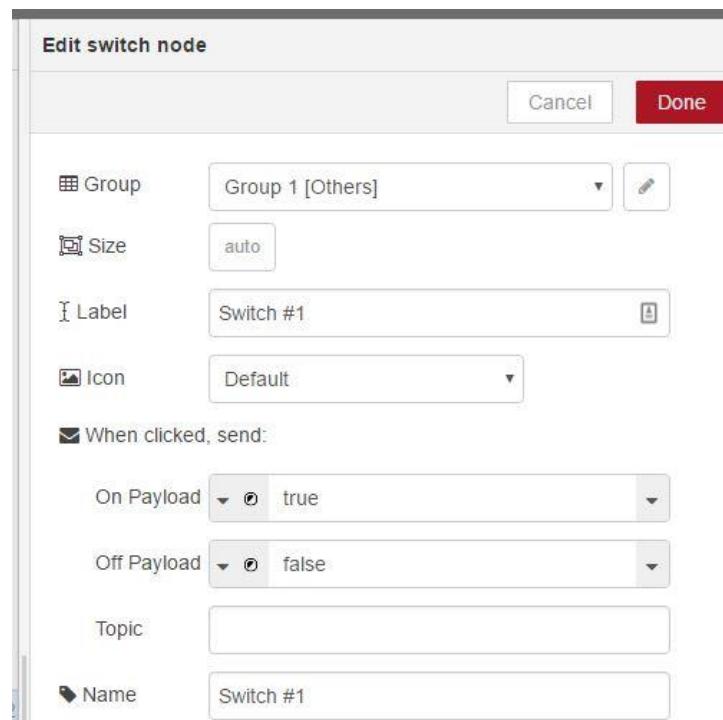


Open the configuration for one of the **Switches** and call it **Master Switch**.

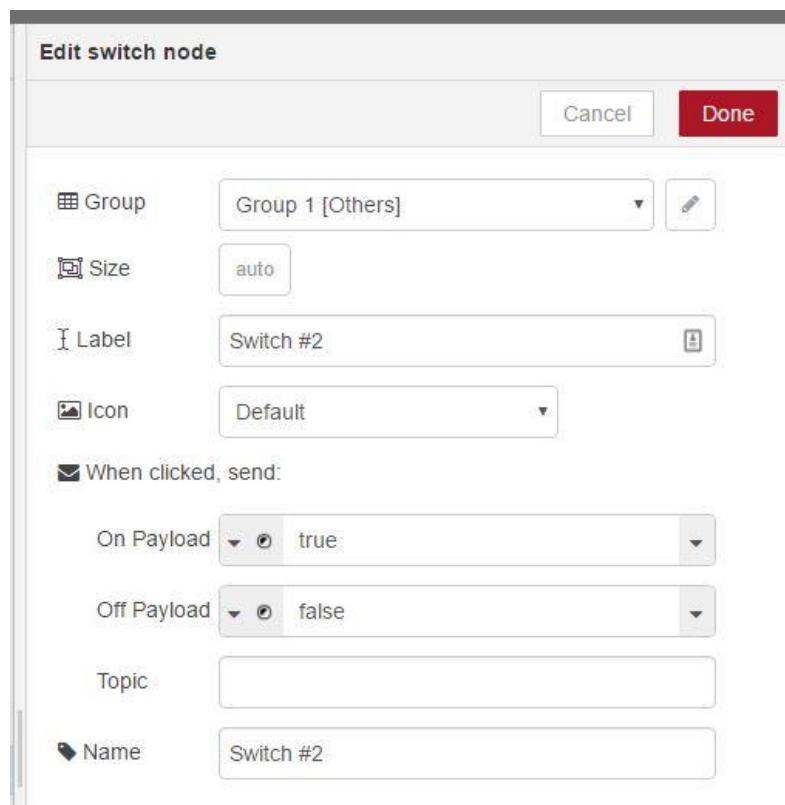


In this example, I'll be controlling 3 switches. So, I've created 3 other test **Switches** with the following configurations.

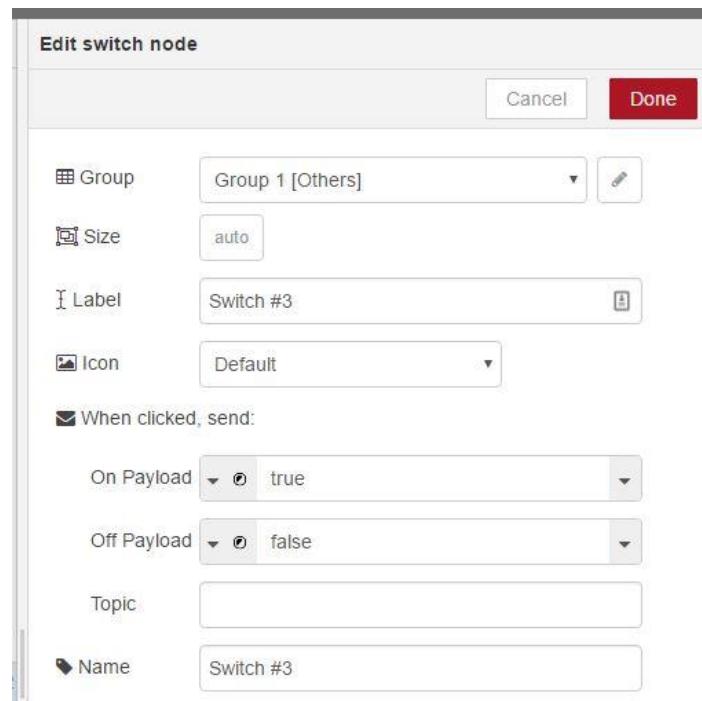
For **Switch #1**:



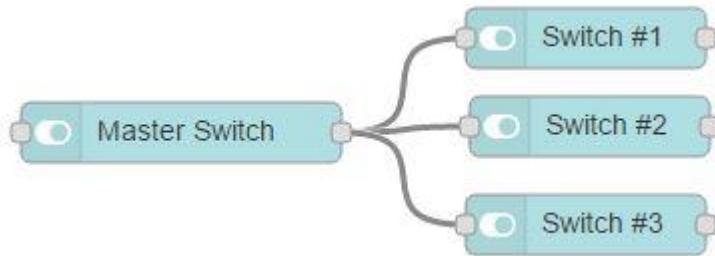
For Switch #2:



For Switch #3:



Finally, you connect your Master Switch to all the other **Switch** nodes that you want to control, as shown in the following figure:

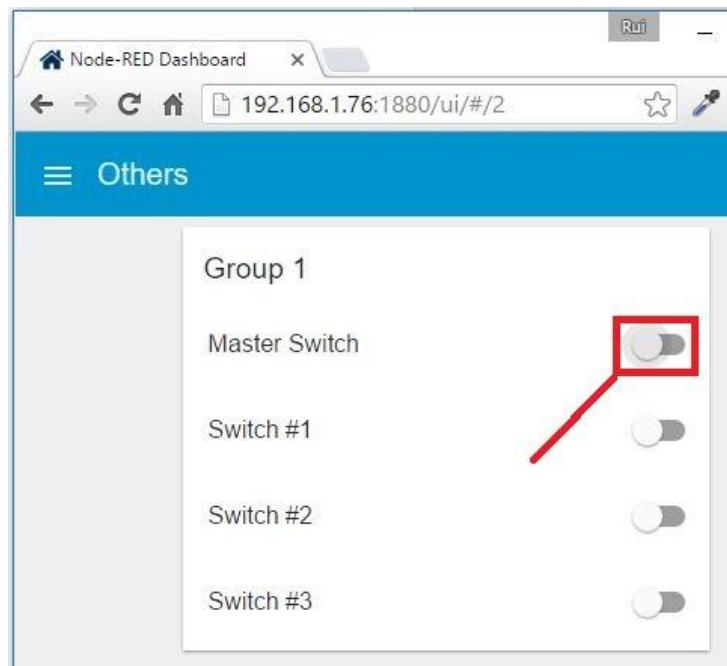


Deploy your application:

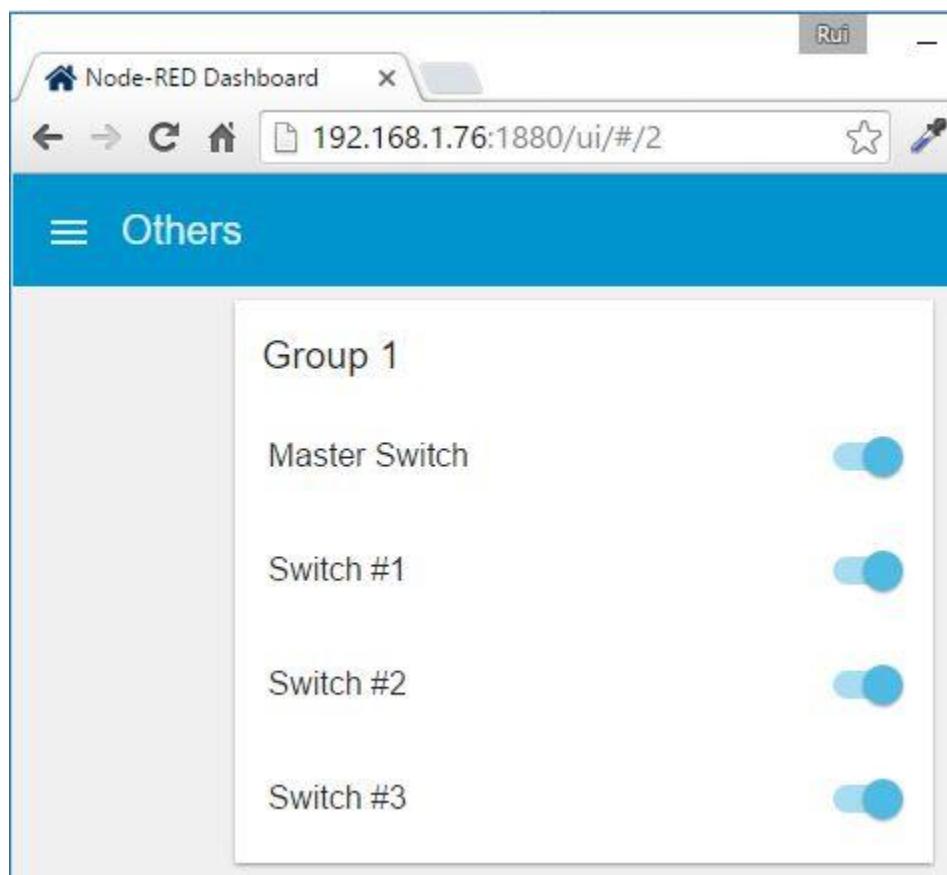


Testing the Master Switch

Open the Node-RED Dashboard tab. 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 control each **Switch** individually.

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 mode: turns off all lights and arms the motion sensor
- Away mode: turns off all lights and arms the motion sensor

In the next Unit, you're going to learn how you can trigger these modes or any Node-RED node automatically with time-based events.

Unit 2 - Triggering 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 could be automated. Turning on/off your lights at a certain time, arming/disarming your motion sensor, etc...

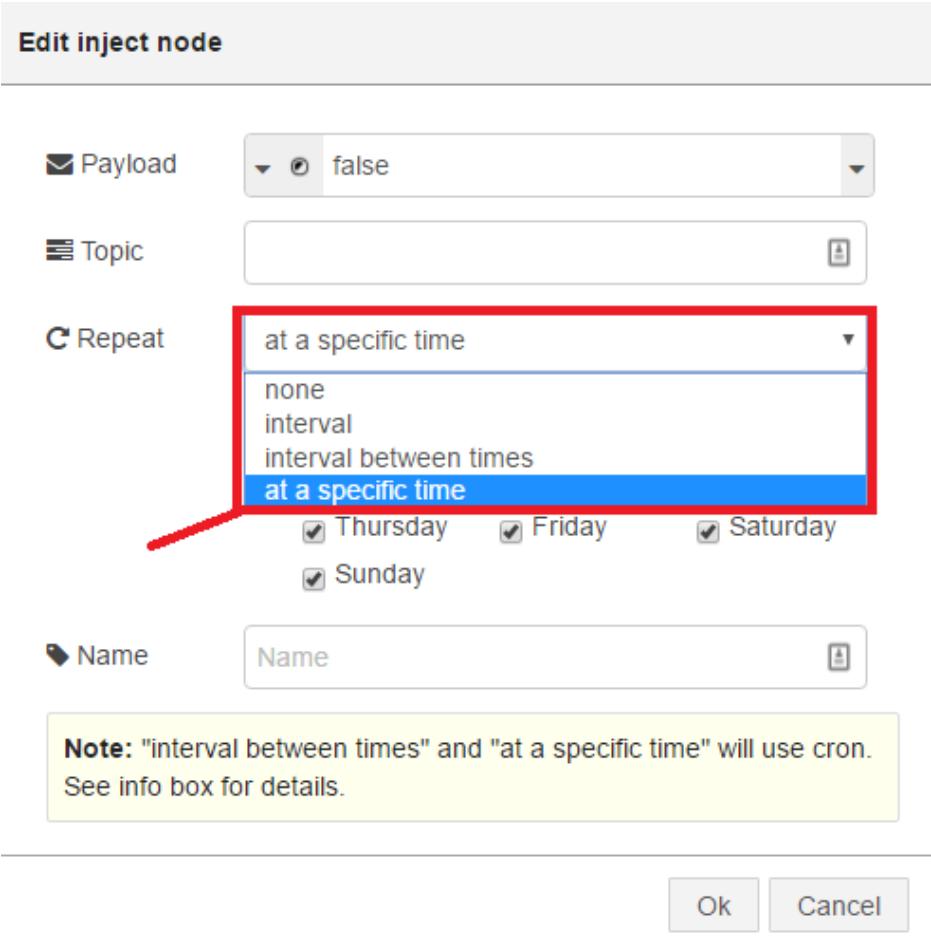
In this Unit, you're going to create events that can occur in a certain day at a specific time. You should add time-based triggers to your home automation system to meet your specific needs.

Creating Your Flow

In this flow, drag 3 nodes: 2 **Inject** nodes and 1 **Switch** node.



The **Inject** node has a great feature that I haven't covered. It's called "Repeat" and you can set the **Inject** node to be automatically triggered "at a specific time".



Warning: you should have your Raspberry Pi time zone properly configured. Read the extra Unit called: "Change the Time Zone on Raspberry Pi with Raspbian".

In my case, I'll be setting the **Inject** node to trigger the **false** value everyday at 19:00 (that's 7PM).

Edit inject node

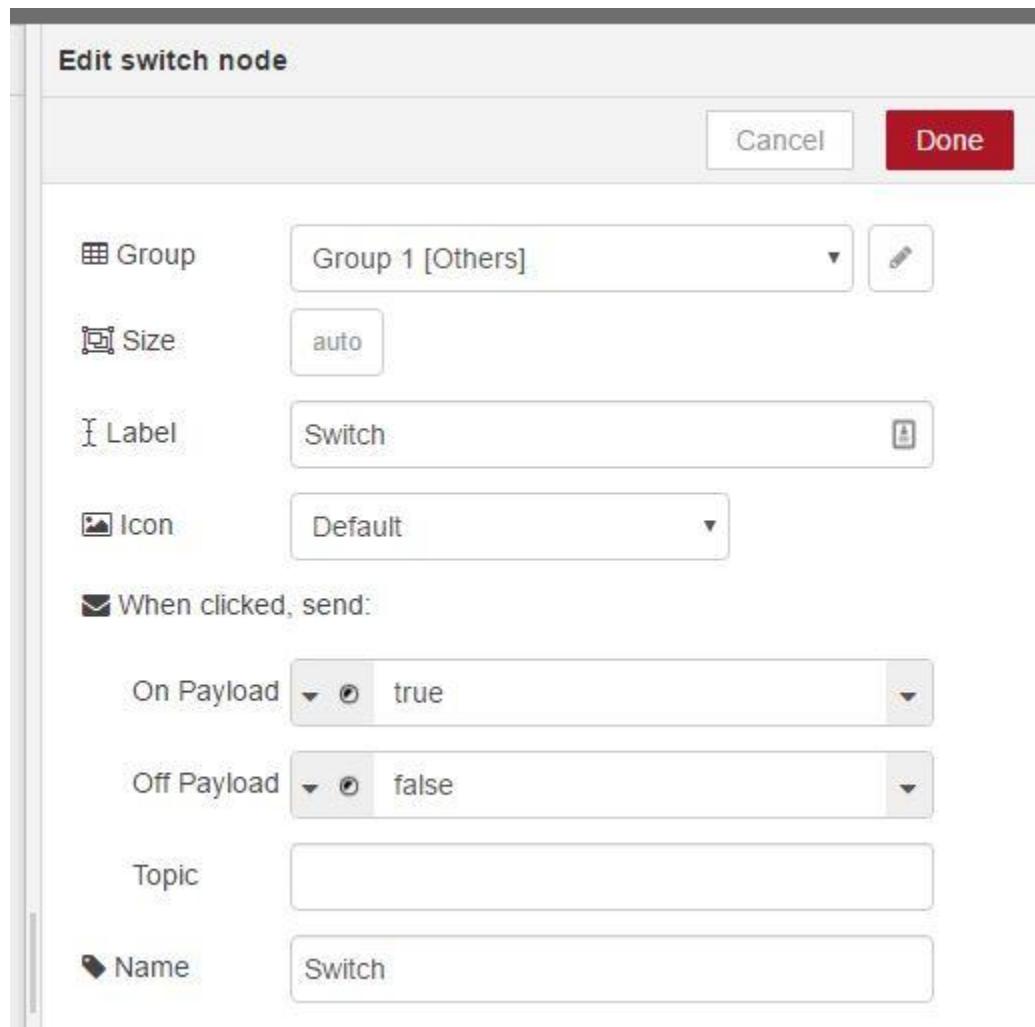
✉ Payload	<input type="radio"/> false
Topic	<input type="text"/>
⌚ Repeat	at a specific time
at <input type="text" value="19:00"/>	
on <input checked="" type="checkbox"/> Monday <input checked="" type="checkbox"/> Tuesday <input checked="" type="checkbox"/> Wednesday	
<input checked="" type="checkbox"/> Thursday <input checked="" type="checkbox"/> Friday <input checked="" type="checkbox"/> Saturday	
<input checked="" type="checkbox"/> Sunday	
🏷 Name	<input type="text" value="Name"/>
Note: "interval between times" and "at a specific time" will use cron. See info box for details.	
<input type="button" value="Ok"/> <input type="button" value="Cancel"/>	

For the other **Inject** node, it will trigger the **true** value everyday at 8:00 (that's 8AM).

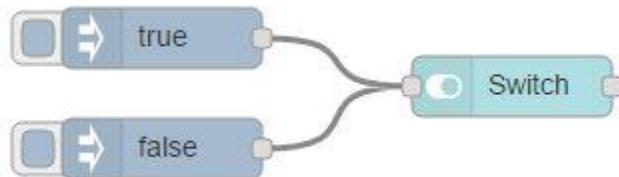
Edit inject node

✉ Payload	<input type="radio"/> true
Topic	<input type="text"/>
⌚ Repeat	at a specific time
at <input type="text" value="8:00"/>	
on <input checked="" type="checkbox"/> Monday <input checked="" type="checkbox"/> Tuesday <input checked="" type="checkbox"/> Wednesday	
<input checked="" type="checkbox"/> Thursday <input checked="" type="checkbox"/> Friday <input checked="" type="checkbox"/> Saturday	
<input checked="" type="checkbox"/> Sunday	
🏷 Name	<input type="text" value="Name"/>
Note: "interval between times" and "at a specific time" will use cron. See info box for details.	
<input type="button" value="Ok"/> <input type="button" value="Cancel"/>	

Finally, you add a Switch node to a Node-RED Dashboard tab:



Connect your 3 nodes as shown in the following figure:

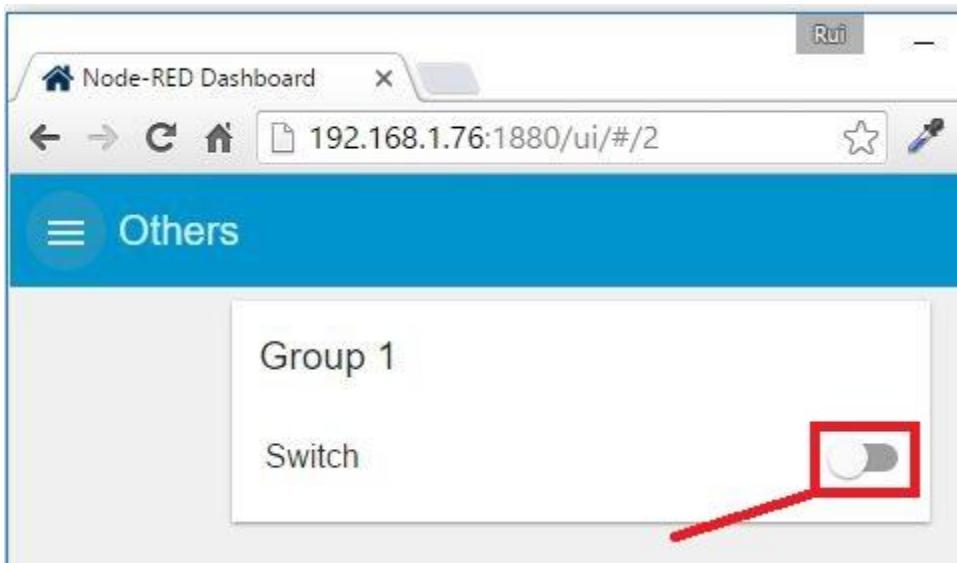


Deploy your application.

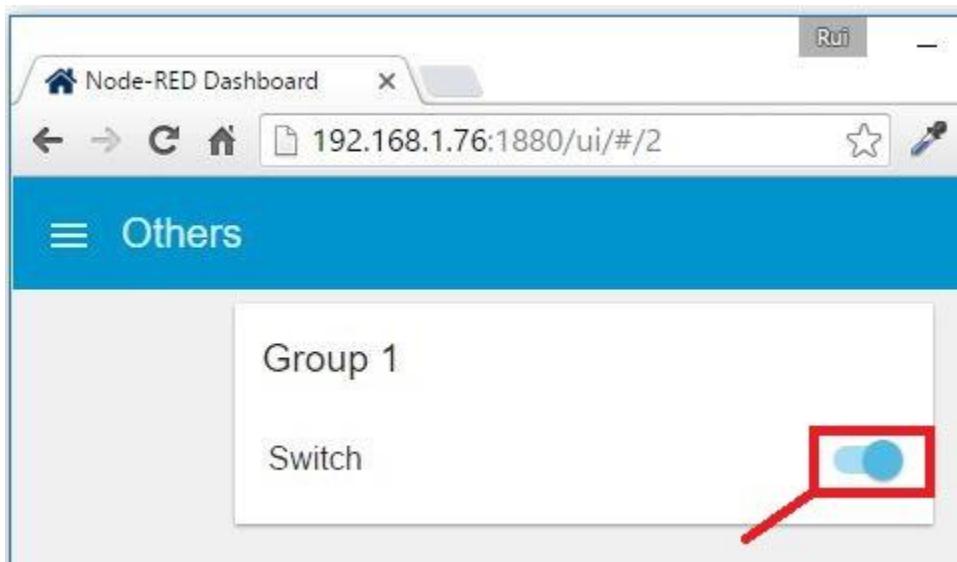


Testing the Flow

The **Switch** node starts off by default. When it reaches 19:00 (that's 7PM).



It will automatically turn on your **Switch**.



Your **Switch** will be automatically turned off at 8:00 (8AM).

This exact same concept can be applied to:

- Control outlets at a certain day and hour
- Arm or disarm sensors
- Trigger notifications or reminders
- Control master switches
- Anything that is time-based

Unit 3 - Sending Notifications to All Your Mobile Devices

In this Unit you're going to send notifications to all your mobile devices with Node-RED.

In order to accomplish this task you have to sign up for a free service called IFTTT which stands for "If This Then That".

IFTTT is a platform that gives you creative control over dozens of products and apps.

You can make apps work together. For example, when you send a request to IFTTT, it triggers a recipe that sends an email alert or notification.

Creating Your IFTTT Account

Creating an account on IFTTT is free!

Go the official site: <https://ifttt.com> and click the "Sign Up" button in the middle of the page.

Complete the form with your information (see figure below) and create your account.



Create your free account

You're only seconds away from doing more with the products you love.

Your Email

Choose a Password

Create account

After creating your account, follow their getting started tutorial by clicking the word "this". Then, click the "Continue" button that appears on your screen a few more times to complete their introductory tutorial.

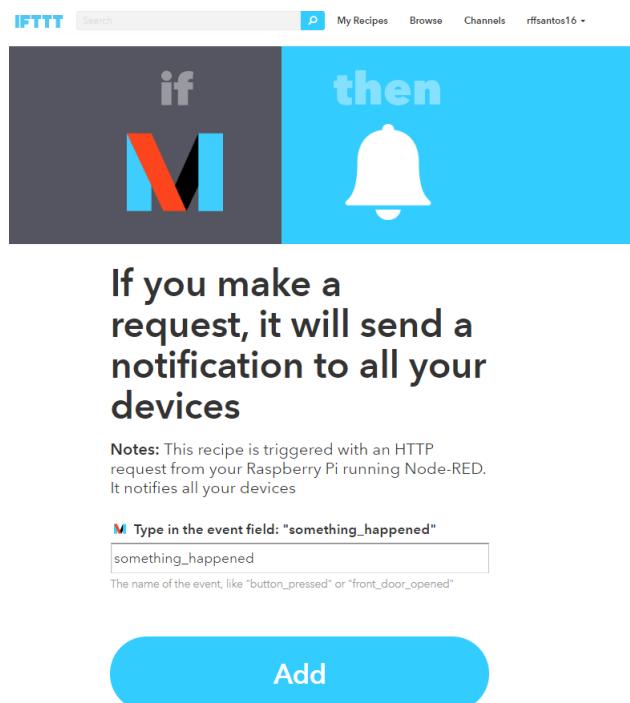
I've created a recipe that integrates perfectly in this project and you can use it.

While logged in at IFTTT, open the URL below to use my recipe instantly:

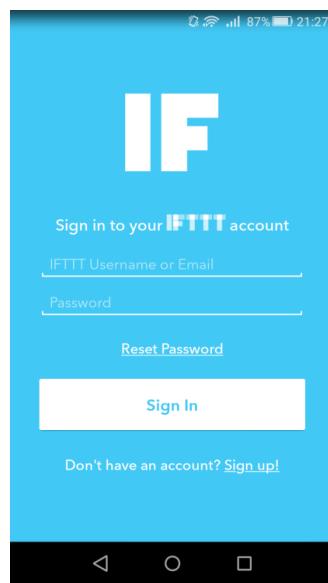
<https://ifttt.com/recipes/425179-if-you-make-a-request-it-will-send-a-notification-to-all-your-devices>

Next, scroll down that page and follow these instructions to make it work for you:

1) Connect your account to the Maker Channel. A new page loads (see figure below) when you finish connecting your account to the Channel.



2) Go to the [Google Play Store](#) or [App Store](#), install the IFTTT app in your smartphone and sign in with your account.



Testing Your Recipe

Now, go to this URL: <https://ifttt.com/maker>. Copy your secret key to a safe place (you'll need them later in this project).

In my example, my secret key is: **b6eDdHYblEv2Sy32qLwe**

The screenshot shows the IFTTT website with the 'Maker Channel' selected. At the top, there are navigation links for 'My Recipes', 'Browse', and 'Channels'. Below the navigation, the 'Maker Channel' is displayed with a large 'M' logo. To the right, it shows '2 Personal Recipes' and '1 Published Recipe'. A descriptive text block explains the Maker Channel's purpose: connecting IFTTT to DIY projects via webhooks. It also links to hackster.io. Below this, a 'Connected as:' section lists a single connection. Underneath, there are two buttons: 'Reconnect Channel' (in a blue box) and 'Disconnect'. To the right, a link to 'How to Trigger Events' is shown, along with the secret key: **b6eDdHYblEv2Sy32qLwe**.

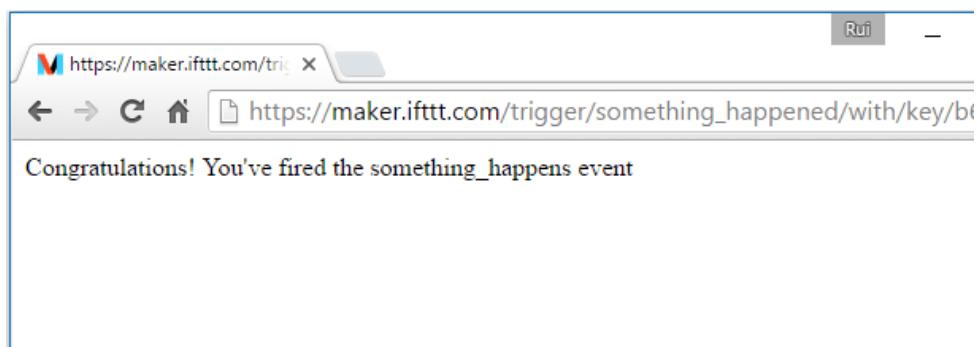
Let's test if your request is working properly. Replace YOUR_API_KEY from the following URL:

https://maker.ifttt.com/trigger/something_happened/with/key/YOUR_API_KEY

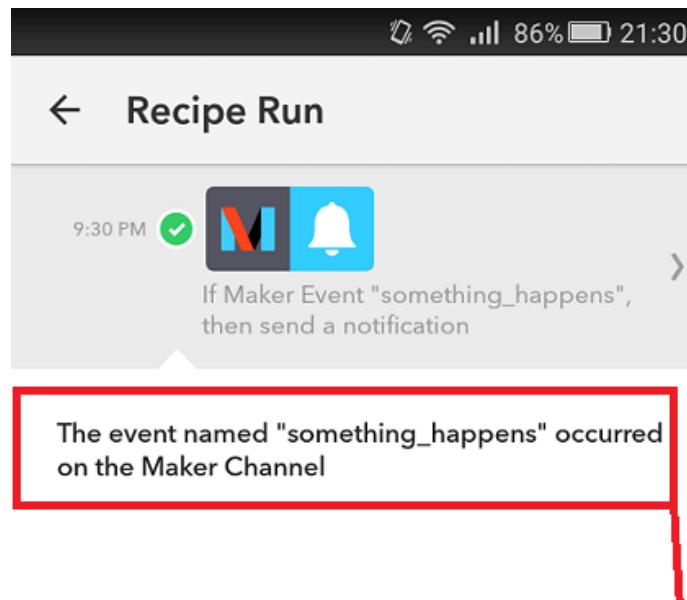
With your **API KEY**:

https://maker.ifttt.com/trigger/something_happened/with/key/b6eDdHYblEv2Sy32qLwe

Open your URL with your API KEY in your browser.

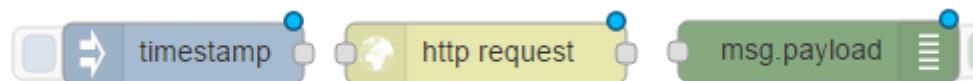


You should see something similar to the preceding figure. Then, go to your smartphone and a new notification should be there.

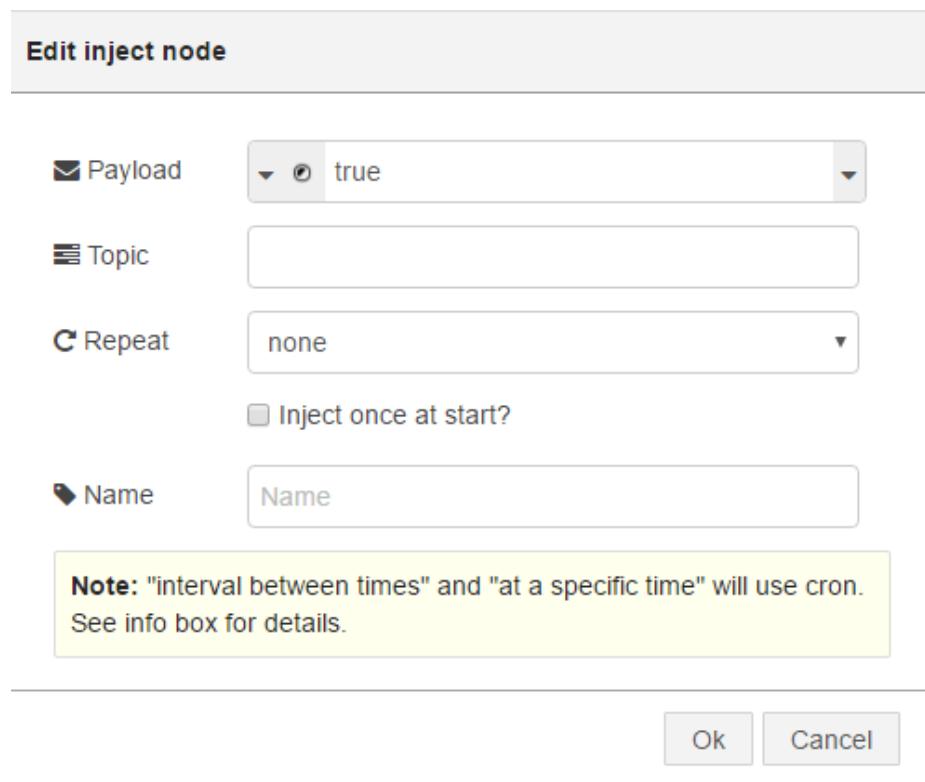


Creating Your Flow

In this flow, you're going to create a request to trigger the IFTTT notification with Node-RED. Drag 3 nodes to your flow: **Inject node**, **HTTP request node** and **Debug node**.

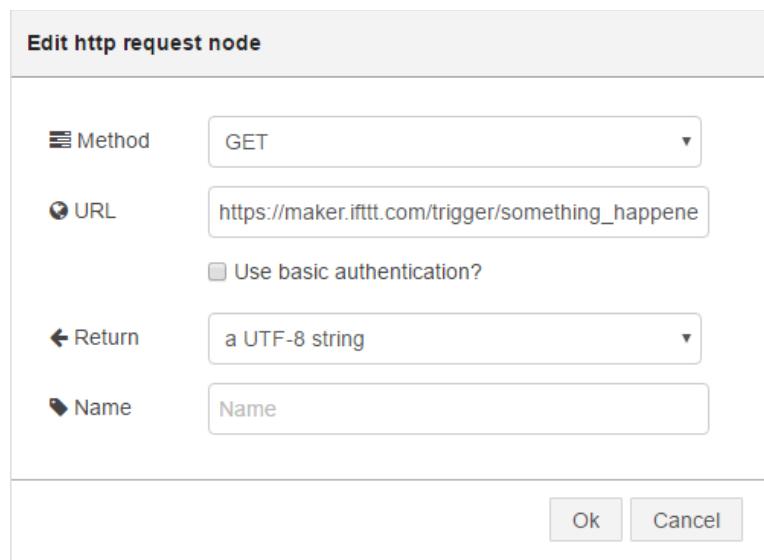


Double-click the **Inject** node and follow these instructions:

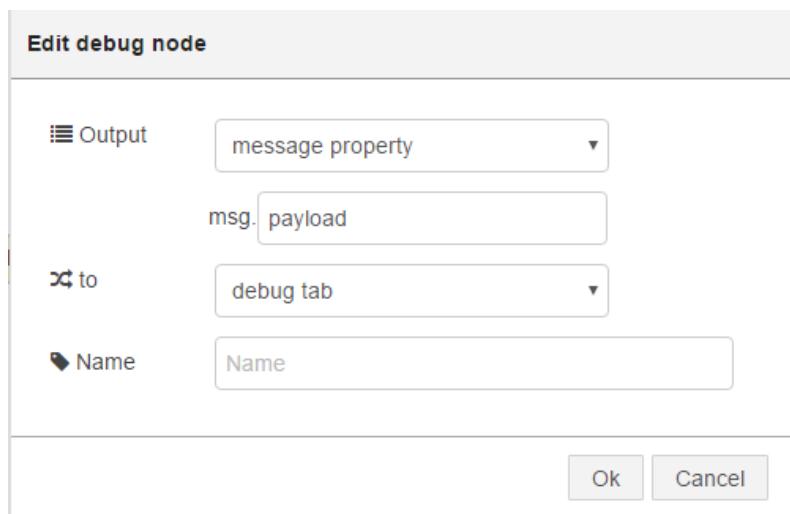


Edit the **HTTP request** node, select the **GET** in the method field and in the **URL** field enter (replace with your actual API key):

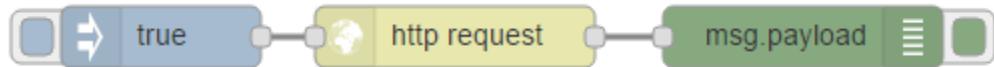
https://maker.ifttt.com/trigger/something_happened/with/key/YOUR_API_KEY



You can leave the default settings for the **Debug** node.



Connect the three nodes as shown in the following figure:

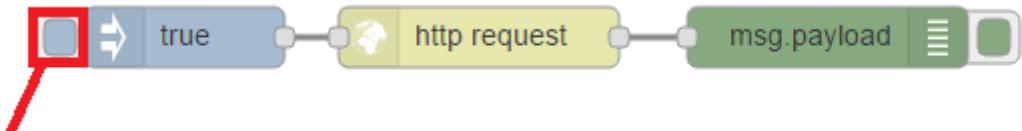


Deploy your application.



Triggering IFTTT Notifications

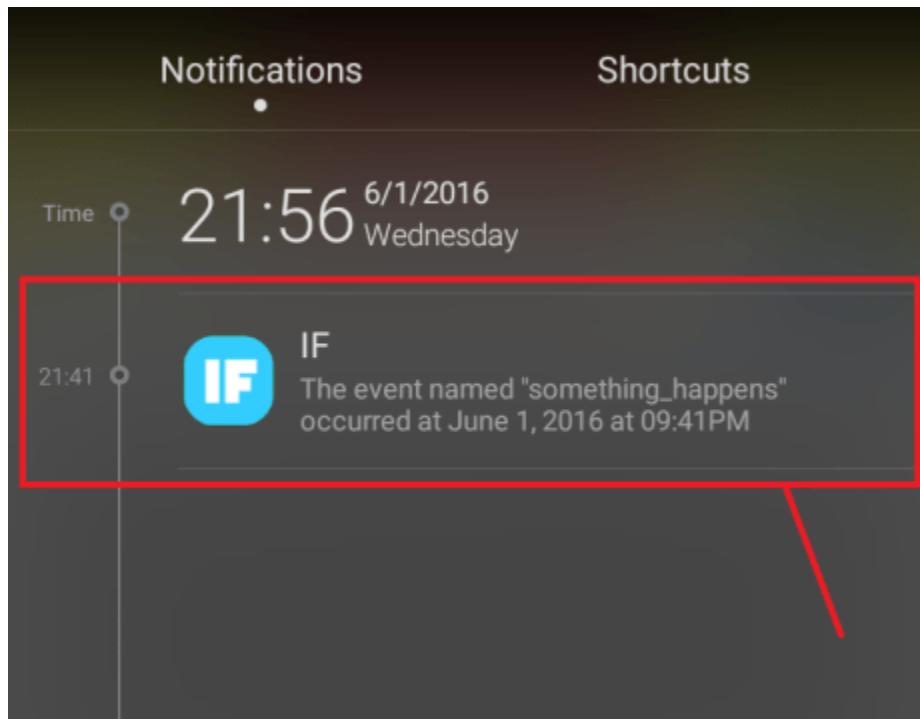
Press the square of the **Inject** node to trigger the flow:



In the **debug** window, you should see a message saying "Congratulations! You've fired...".



Your smartphone instantly receives a notification with the event name and when it happened:



This basic concept can be easily extended and applied to any sensor that you're monitoring in your home automation system. When something happens, you make a request to the IFTTT service to send a notification.

Unit 4 - Wrapping Up and Taking It Further

Congratulations for completing the course!

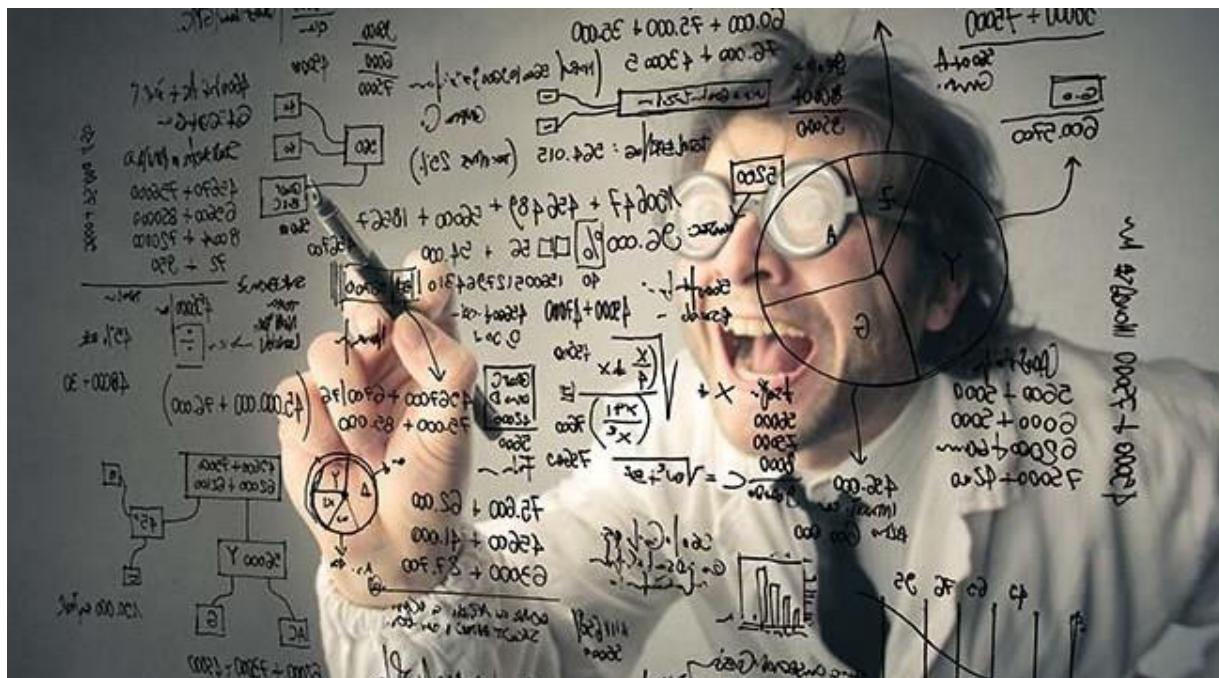
If you've followed all Modules presented throughout this course, you now have the knowledge to build your Home Automation system using the Raspberry Pi, ESP8266 and Arduino boards.

Wrapping Up

Let's see the main things that you've accomplished. You know how to:

- Use Raspberry Pi, ESP8266 and Arduino board
- Use Node-RED, Node-RED Dashboard and Linux commands
- Establish an MQTT connection with multiple devices in your network
- Control any device, outlet or lamp
- Read sensor data (temperature, humidity, luminosity and more)
- Trigger events based on sensor data
- Display your data in gauges and charts
- Build a smoke detection system
- Make a user interface password protected and accessible from anywhere in the world
- Build a motion detection system with email notifications
- Create time-based events
- Set modes for your home

It's Your Time to Experiment



Now, I encourage you to add multiple ESP8266s and Arduinos around your home and build up on the snippets of code presented in this course to fit your own projects and specific needs.

I honestly feel like we've covered all the important subjects, but there's always room for improvement and here's a handful of ideas:

- Surveillance camera
- Controlling TV with Arduino
- Voice commands recognition
- RFID door lock system
- RGB LED strip light controller
- Garage door controller
- Control window blinds

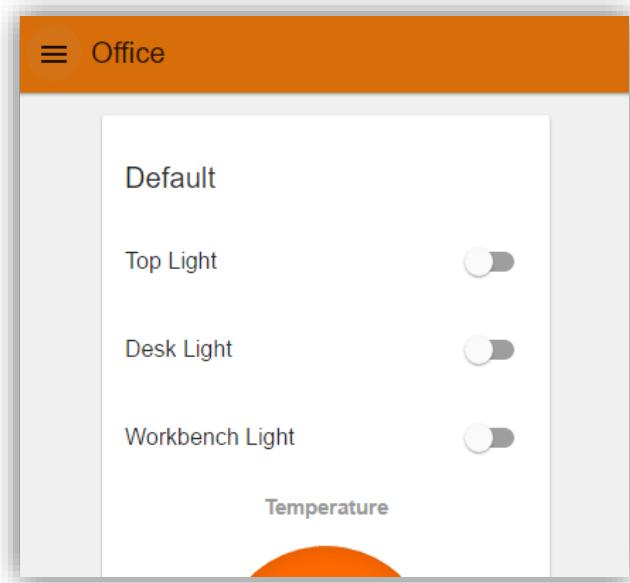
I hope you had fun following all these projects! If you have something you would like to share, let me know in the Facebook group ([join the Facebook group here](#)).

Good luck with all your projects,

-Rui Santos

Module 13

Extra #1 - Information that might be useful for this course

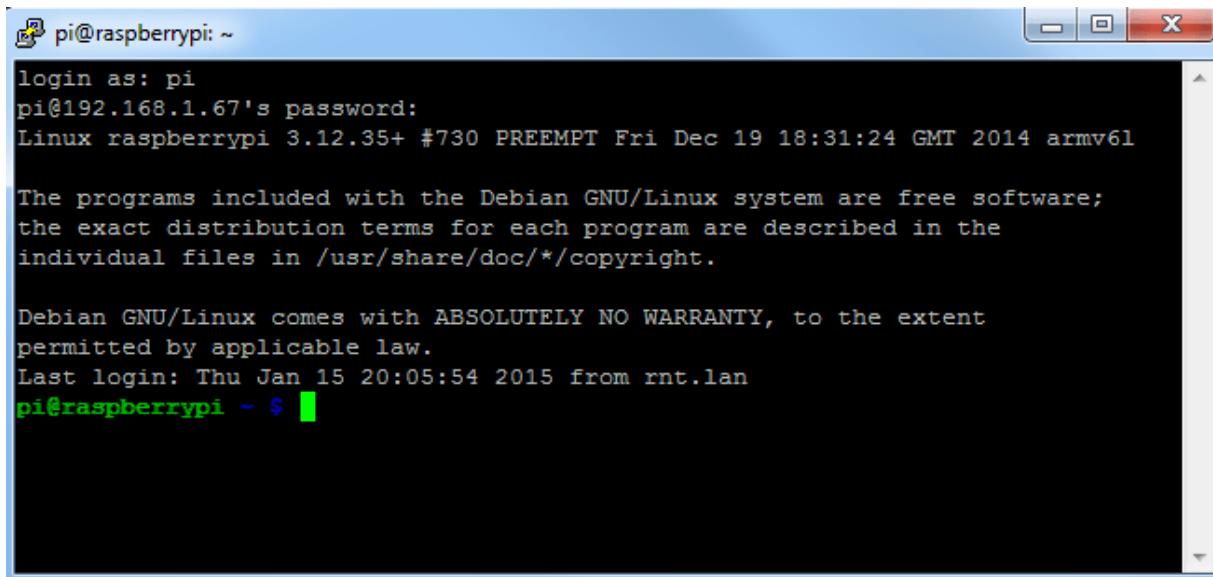


Unit 1 - How to Configure WiFi on Your Raspberry Pi

Here's how to configure WiFi on your Raspberry Pi through the command line.

1) Establish an SSH communication with your Pi

Boot your Raspberry Pi with the WiFi adapter plugged in. You also need to connect Ethernet cable connection to ensure that you can open an SSH client like PuTTY to establish an SSH communication.



```
pi@raspberrypi: ~
login as: pi
pi@192.168.1.67's password:
Linux raspberrypi 3.12.35+ #730 PREEMPT Fri Dec 19 18:31:24 GMT 2014 armv6l

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.

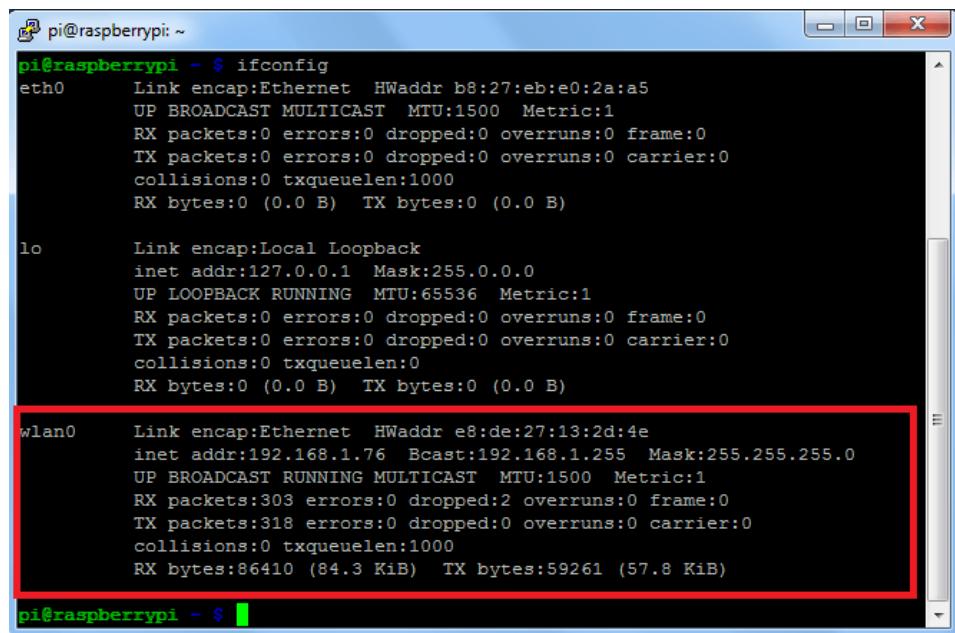
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Jan 15 20:05:54 2015 from rnt.lan
pi@raspberrypi ~ $
```

2) Checking if your RPi recognizes your WiFi adapter

There are several ways to check if your WiFi adapter has been recognized. You can type:

```
pi@raspberry:~ $ ifconfig
```

And your wireless adapter named as `wlan0` should appear as shown in the figure below.



```
pi@raspberrypi ~ $ ifconfig
eth0      Link encap:Ethernet HWaddr b8:27:eb:e0:2a:a5
          UP BROADCAST MULTICAST MTU:1500 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

lo       Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          UP LOOPBACK RUNNING MTU:65536 Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

wlan0    Link encap:Ethernet HWaddr e8:de:27:13:2d:4e
          inet addr:192.168.1.76 Bcast:192.168.1.255 Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
          RX packets:303 errors:0 dropped:2 overruns:0 frame:0
          TX packets:318 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:86410 (84.3 KiB)  TX bytes:59261 (57.8 KiB)

pi@raspberrypi ~ $
```

If you don't see your WiFi adapter listed, you might have to install drivers for your particular WiFi adapter. I'm using the [TL-WN725N](#) and by default Raspbian doesn't support my WiFi adapter. So I've followed this [thread](#) to install my drivers.

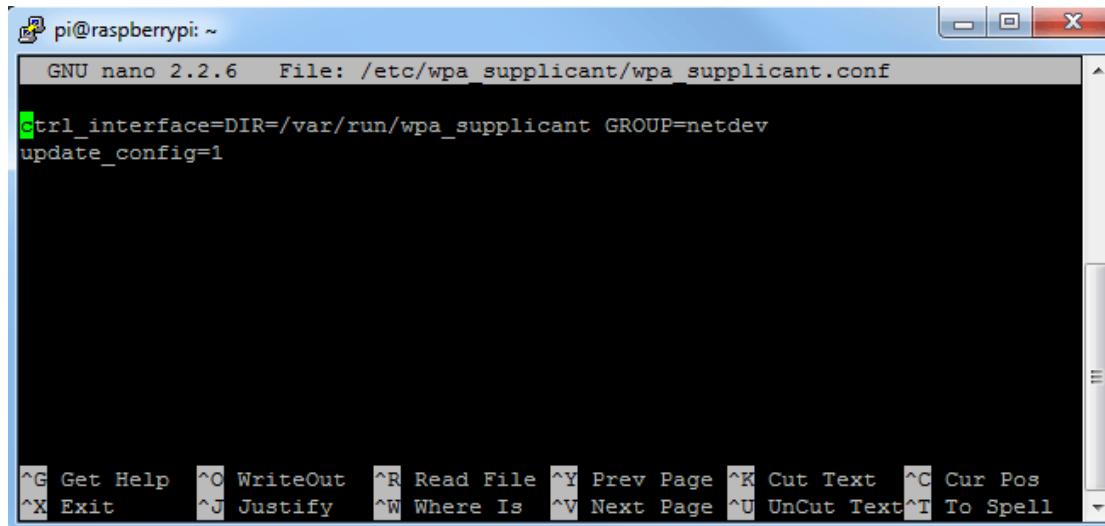
If you don't have a WiFi adapter yet, I highly recommend purchasing the [Edimax EW 7811UN](#). This is a good option, because Raspbian comes with its drivers installed out of the box, that ensures that your RPi recognizes that WiFi adapter.



3) Opening configuration file

Type the following command to open your configuration file:

```
pi@raspberry:~ $ sudo nano /etc/wpa_supplicant/wpa_supplicant.conf
```



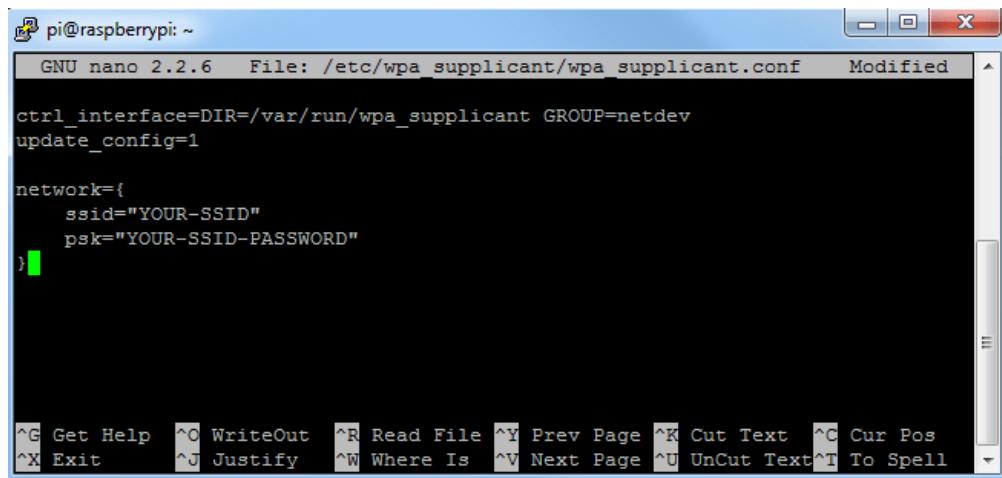
The screenshot shows a terminal window titled "pi@raspberrypi: ~". Inside the window, the nano text editor is displaying the contents of the file "/etc/wpa_supplicant/wpa_supplicant.conf". The file currently contains the following configuration:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
```

At the bottom of the terminal window, there is a menu bar with various keyboard shortcuts for navigating and editing the file.

4) Adding your network details

Go to the bottom of your configuration file wpa_supplicant.conf and add your network details as shown below. Replace “YOUR-SSID” and “YOUR-SSID-PASSWORD” with the details of your WiFi connection.



The screenshot shows the same terminal window as before, but now it displays additional configuration at the bottom of the file. The "network" section has been added and contains the following entries:

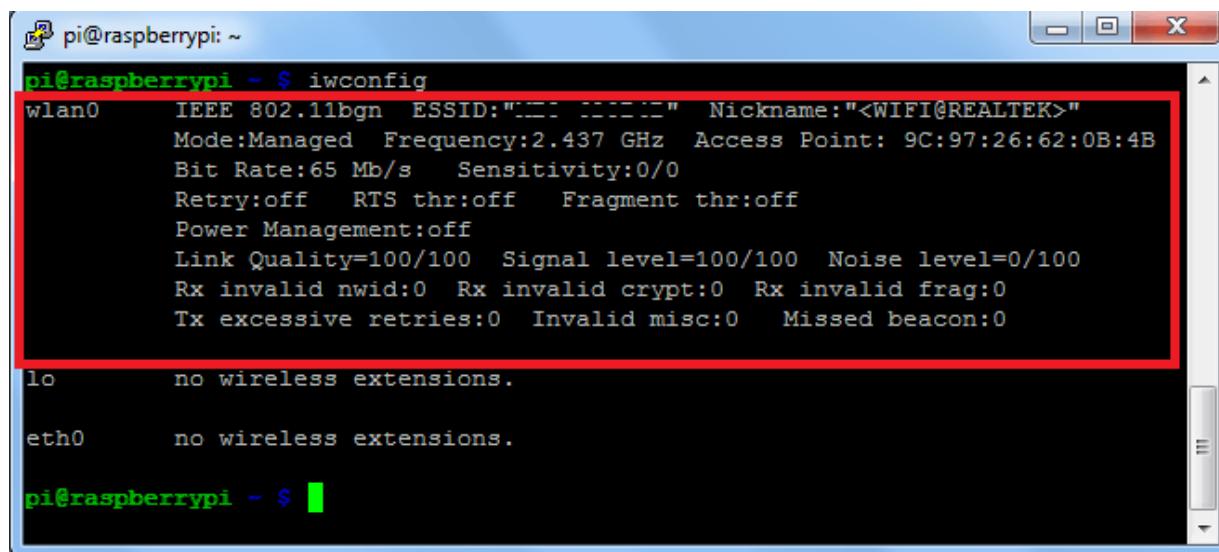
```
network={
    ssid="YOUR-SSID"
    psk="YOUR-SSID-PASSWORD"
}
```

The word "Modified" is visible in the title bar of the terminal window, indicating that changes have been made to the file.

Now save your file by pressing **Ctrl+x** then **y**, then finally press **Enter**.

5) Testing your connection

You can check the status of the wireless connection using `ifconfig` (to see if `wlan0` has acquired an IP address) and `iwconfig` to check which network the wireless adapter is using.



```
pi@raspberrypi: ~
pi@raspberrypi - $ iwconfig
wlan0      IEEE 802.11bgn  ESSID:"[REDACTED]"  Nickname:<WIFI@REALTEK>
            Mode:Managed  Frequency:2.437 GHz  Access Point: 9C:97:26:62:0B:4B
            Bit Rate:65 Mb/s  Sensitivity:0/0
            Retry:off  RTS thr:off  Fragment thr:off
            Power Management:off
            Link Quality=100/100  Signal level=100/100  Noise level=0/100
            Rx invalid nwid:0  Rx invalid crypt:0  Rx invalid frag:0
            Tx excessive retries:0  Invalid misc:0  Missed beacon:0

lo        no wireless extensions.

eth0      no wireless extensions.

pi@raspberrypi - $
```

Unit 2 - Change the Time Zone on Raspberry Pi with Raspbian

You can quickly change the time zone of your Raspberry Pi with one command.

First, you have to check if your time zone is already correct. Use the date command:

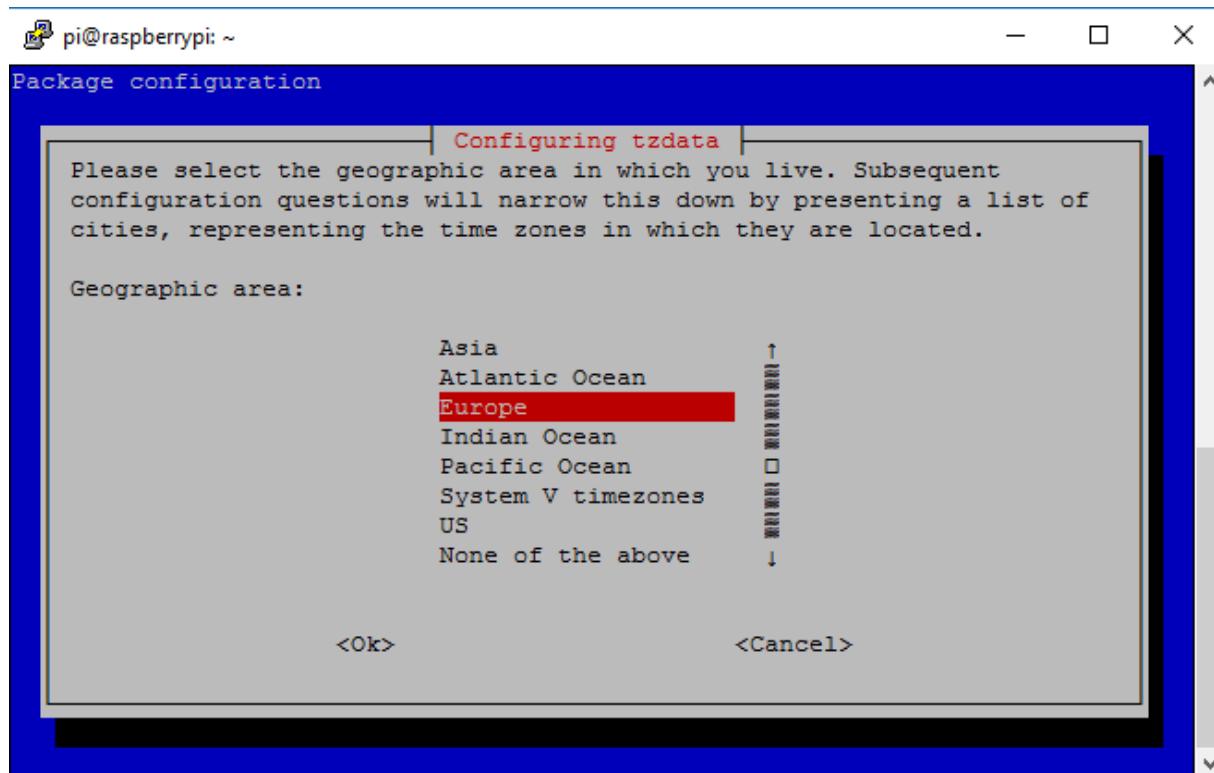
```
pi@raspberrypi:~ $ date  
Mon 30 May 09:54:40 UTC 2016
```

If the hour is correct, you don't have to do anything else. In my case, I need to change the time zone from UTC to WEST.

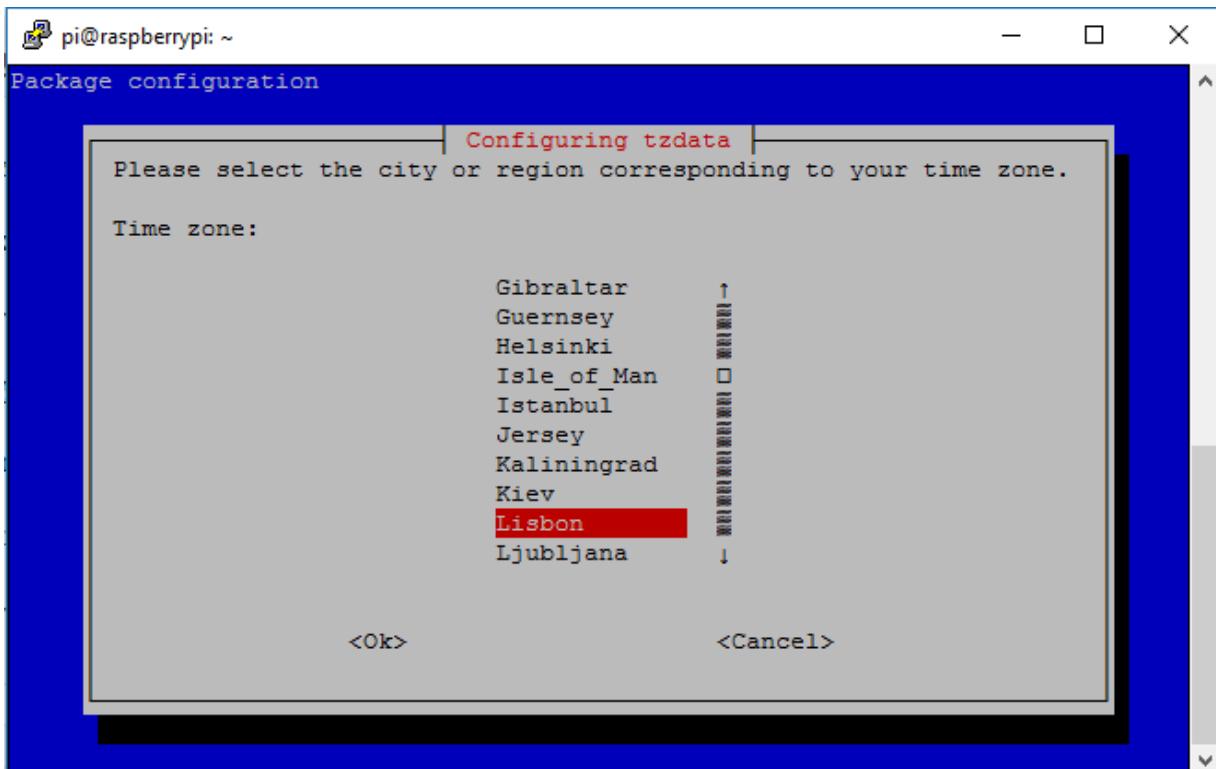
Run this command:

```
pi@raspberrypi:~ $ sudo dpkg-reconfigure tzdata
```

Select your geographic area (it's Europe for me) and click "Ok":



Select the national capital or state where you are located (I live in Portugal, so it's Lisbon for me). Click "Ok".



Now, your RPi should be set for the correct time zone.

Enter the date command again and it should output the right hour for your location.

```
pi@raspberrypi:~ $ sudo dpkg-reconfigure tzdata
Current default time zone: 'Europe/Lisbon'
Local time is now:      Mon May 30 10:58:43 WEST 2016.
Universal Time is now: Mon May 30 09:58:43 UTC 2016.

pi@raspberrypi:~ $ date
Mon 30 May 10:58:49 WEST 2016
```

Reboot your RPi to ensure that Node-RED updates the time zone.

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

Unit 3 - ESP-01 with Arduino IDE

This Extra Unit shows:

1. How to configure the ESP-01 with the Arduino IDE
2. How to upload code with FTDI programmer to the ESP-01

There are a variety of development environments that can be used to program the ESP8266.

The ESP8266 community created an add-on for the Arduino IDE that allows you to program the ESP8266 using the Arduino IDE and its programming language.

Downloading Arduino IDE

First download the Arduino IDE to ensure that you have the latest software version (some older versions won't work), visit the following URL: <https://www.arduino.cc/en/Main/Software>.

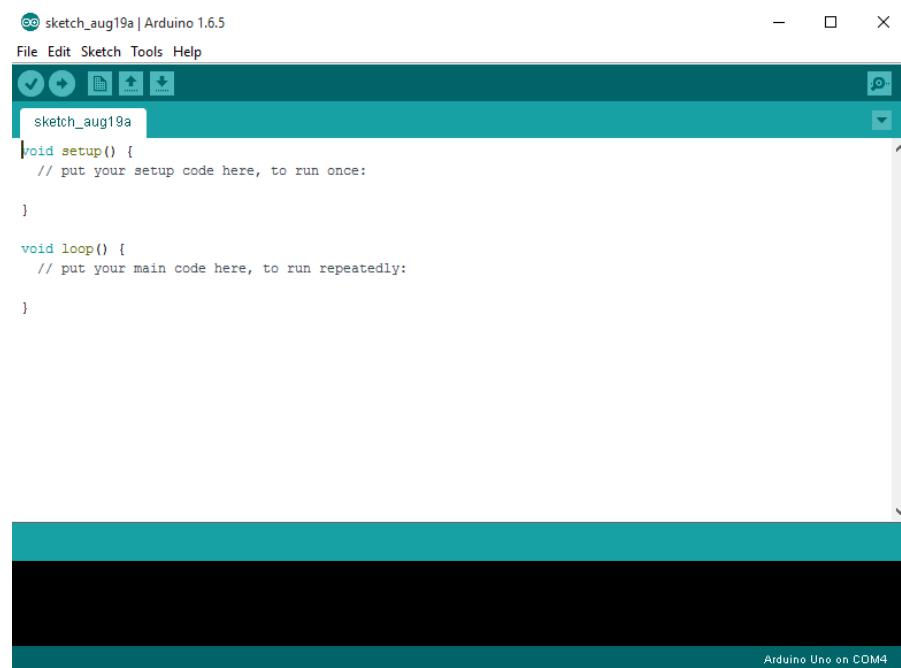
Then select your operating system and download the latest software release of the Arduino IDE.

Installing Arduino IDE

Grab the file that you have just downloaded and open the Arduino IDE application file (see Figure below).

Name	Date modified	Type
dist	13/08/2015 20:53	File folder
drivers	13/08/2015 20:53	File folder
examples	13/08/2015 20:54	File folder
hardware	13/08/2015 20:54	File folder
java	13/08/2015 20:57	File folder
lib	13/08/2015 20:59	File folder
libraries	11/09/2015 13:38	File folder
reference	13/08/2015 21:03	File folder
tools	13/08/2015 21:03	File folder
arduino	14/08/2015 17:42	Application

When the Arduino IDE first opens, this is what you should see:

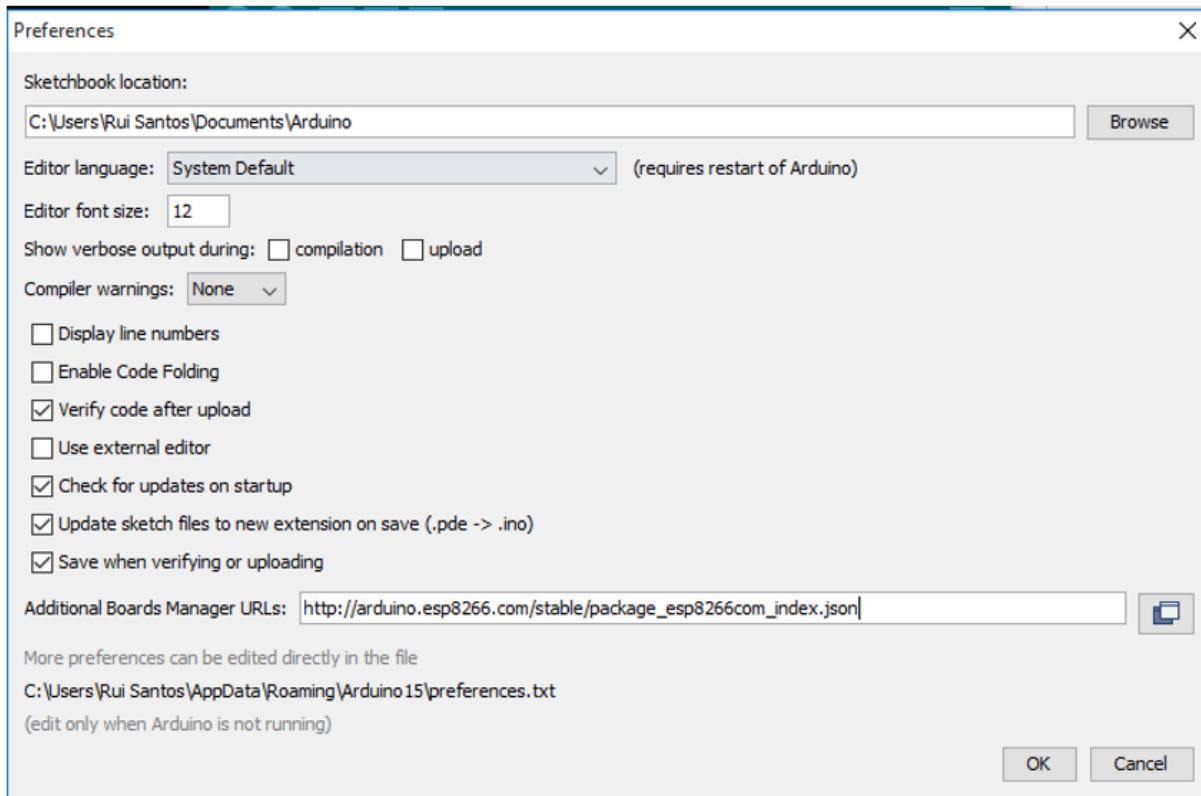


Installing the ESP8266 Board

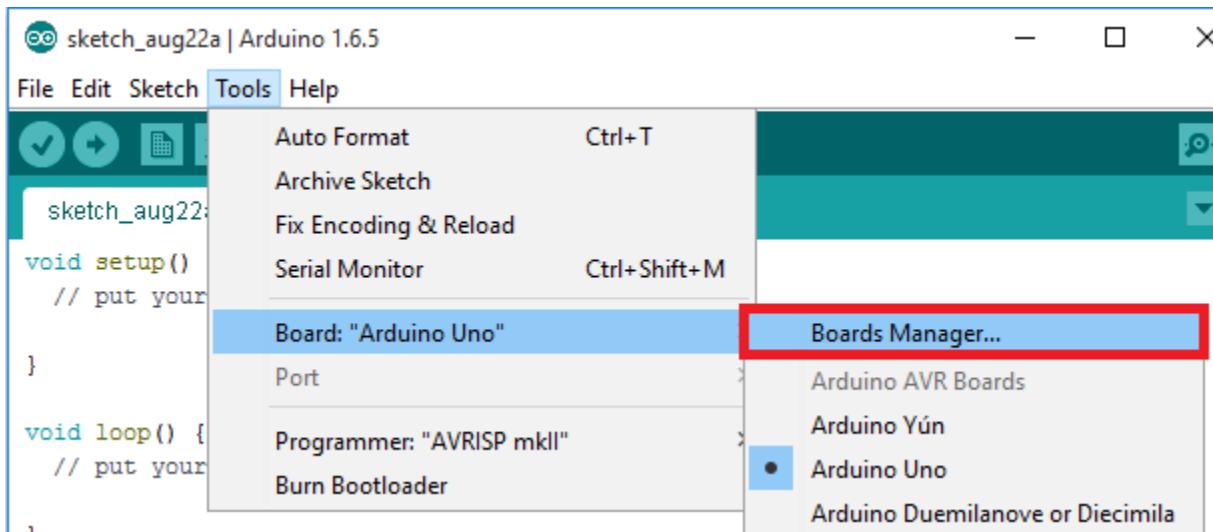
To install the ESP8266 board in your Arduino IDE, follow these next instructions:

- 1) Open the preferences window from the Arduino IDE. Go to **File > Preferences**

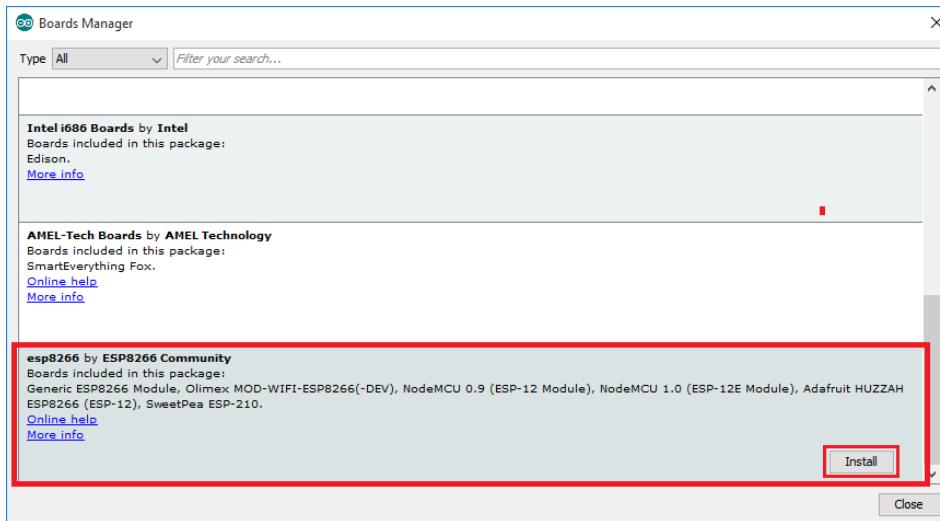
- 2) Enter http://arduino.esp8266.com/stable/package_esp8266com_index.json into **Additional Board Manager URLs** field and click the “OK” button



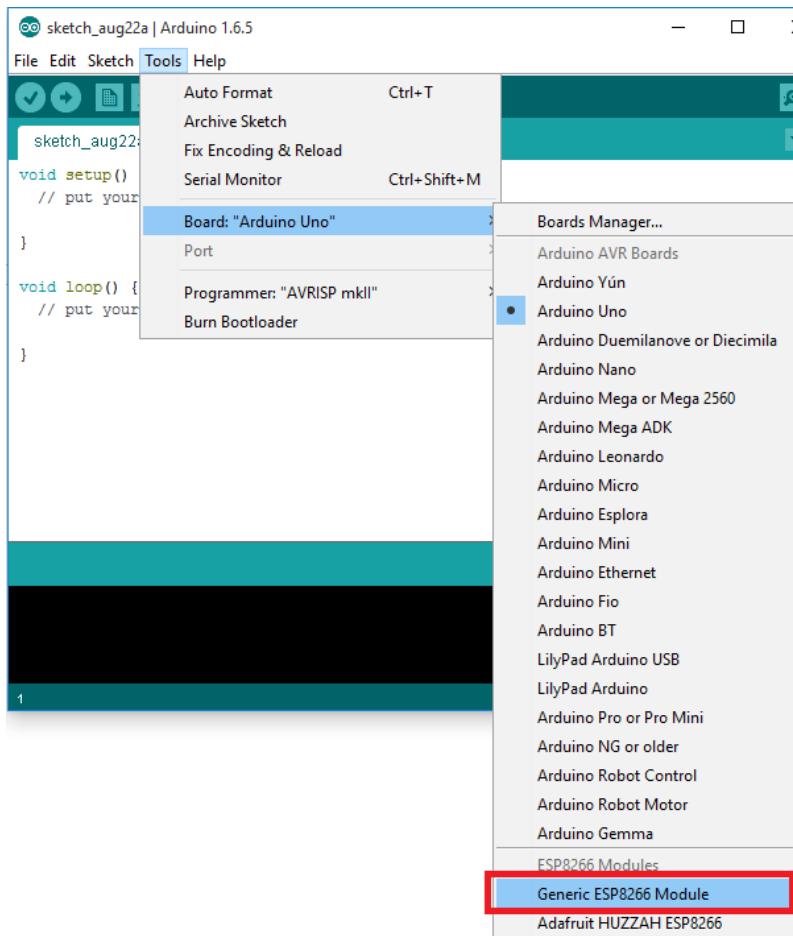
3) Open boards manager. Go to Tools > Board > Boards Manager...



4) Scroll down, select the ESP8266 board menu and install “esp8266 platform”



5) Choose your ESP8266 board from Tools > Board > Generic ESP8266 Module



6) Finally, re-open your Arduino IDE

Testing the Installation

To test the ESP8266 add-on installation, let's see if we can blink an LED with the ESP8266 using the Arduino programming language.

Parts List:

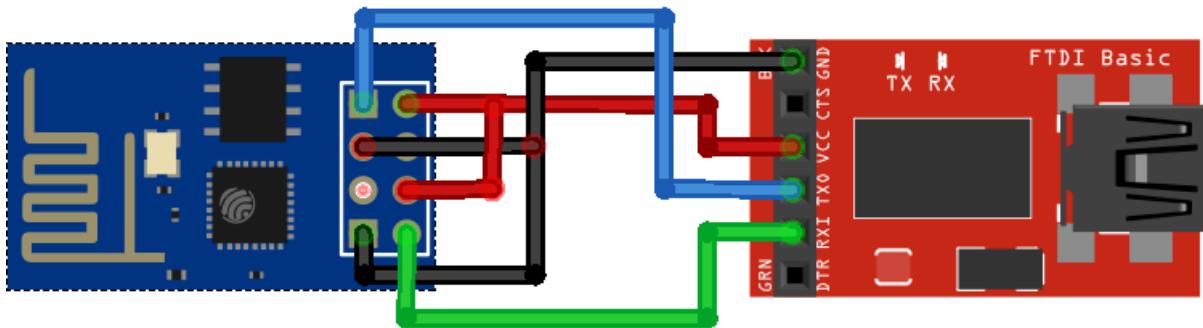
Here's the hardware that you need to complete this project:

- [Click here to get an ESP8266 on eBay](#) for less than \$4
- [Click here to get an FTDI programmer on eBay](#)



How to Upload a Sketch

To upload code to your ESP follow these schematics:



fritzing

Then upload the sketch below to your ESP using the Arduino IDE. You should see "Done Uploading" after a few seconds.

```
*****
Rui Santos
Complete project details at http://randomnerdtutorials.com
*****


int pin = 2;

void setup() {
    // initialize GPIO 2 as an output.
    pinMode(pin, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
    digitalWrite(pin, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(1000);           // wait for a second
    digitalWrite(pin, LOW); // turn the LED off by making the voltage LOW
    delay(1000);           // wait for a second
}
```

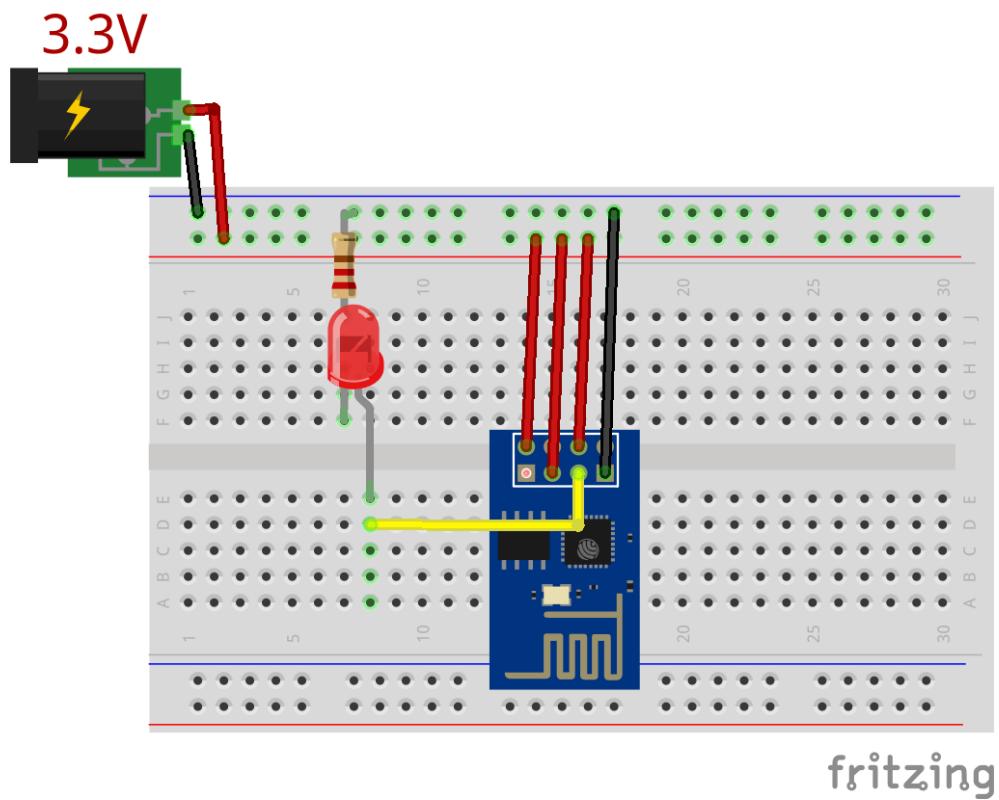
DOWNLOAD SOURCE CODE

https://github.com/RuiSantosdotme/Home-Automation-Course/blob/master/code/blink_led_esp8266.ino

Note: You have to select your FTDI's port number under the Tools > Port menu of the Arduino IDE.

Demonstration

Now assemble this simple circuit and add an LED to your ESP. Your LED should be blinking every 1 second.



Now you can continue the Home Automation using Raspberry Pi course with the ESP-01:

1. Installing the PubSubClient Library
2. Connecting the ESP8266 to the Node-RED Nodes

Unit 4 - ESP-12E – Pinout Reference

This page provides a quick pinout reference for ESP8266-12E that has built-in programmer.

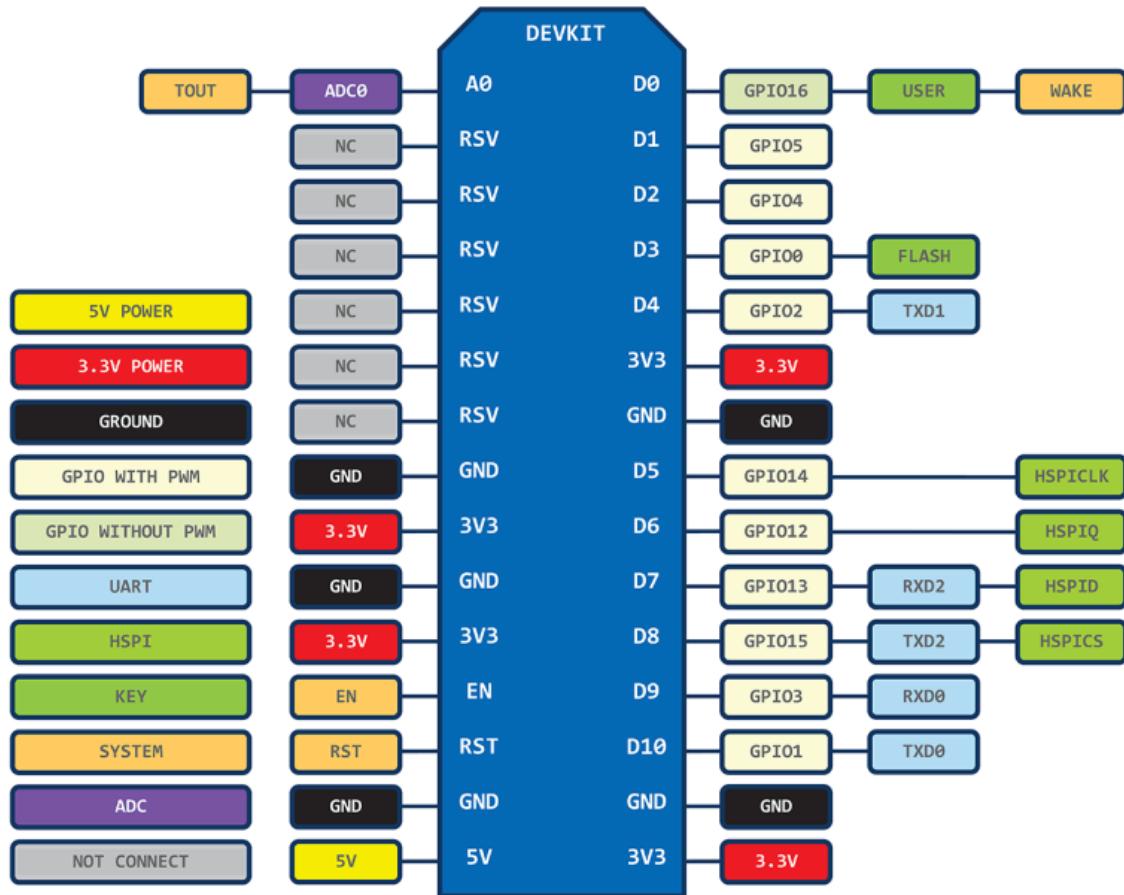


ESP-12E Pinout

Use this table below for a quick reference on how to assign the Arduino GPIOs in the Arduino code (first column):

IO index (Arduino code)	ESP8266 GPIO	ESP-12E
0	GPIO 0	D3
1	GPIO 1	D10
2	GPIO 2	D4
3	GPIO 3	D9
4	GPIO 4	D2
5	GPIO 5	D1
12	GPIO 12	D6
13	GPIO 13	D7
14	GPIO 14	D5
15	GPIO 15	D8
16	GPIO 16	D0
A0	ADC 0	A0

Here's the location for each pin in the actual board:



Unit 5 - MQTT Authentication with Username and Password

In this Unit, I'm going to show you how to setup an authentication mechanism with MQTT.

I will use basic authentication with username and password. You could potentially setup a full encrypted secured connection with TLS, but it is a bit more complicated and it's not the purpose of this tutorial.

Configuring Mosquitto

Before running this command, replace the username part with the actual username that you desire:

```
pi@raspberry:~ $ sudo mosquitto_passwd -c  
/etc/mosquitto/username
```

I'll be using **rui** as my username:

```
pi@raspberry:~ $ sudo mosquitto_passwd -c  
/etc/mosquitto/username rui
```

Then you will be prompted to enter a password twice. That will be the password required to connect to the server with the username that you just chose.

Run this command to configure the permissions to write in that directory:

```
pi@raspberry:~ $ sudo sh -c 'zcat  
/usr/share/doc/mosquitto/examples/mosquitto.conf.gz >  
/etc/mosquitto/conf.d/mosquitto.conf'
```

Then edit Mosquitto configuration file:

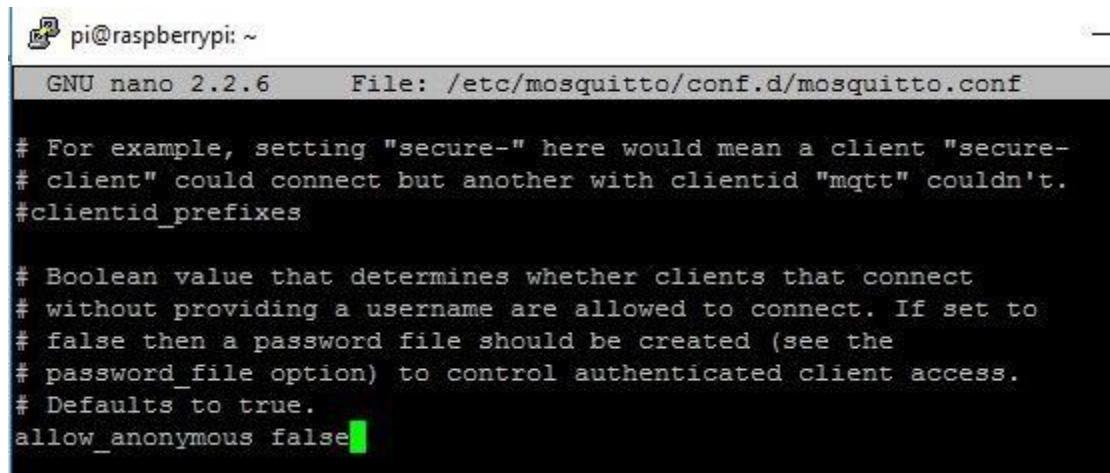
```
pi@raspberry:~ $ sudo nano /etc/mosquitto/conf.d/mosquitto.conf
```

Scroll and find the line containing “#allow_anonymous”:

1. remove the # to uncomment it
2. set the parameter to false

This how it should look like:

```
allow_anonymous false
```



```
pi@raspberrypi: ~
GNU nano 2.2.6      File: /etc/mosquitto/conf.d/mosquitto.conf

# For example, setting "secure-" here would mean a client "secure-
# client" could connect but another with clientid "mqtt" couldn't.
#clientid_prefixes

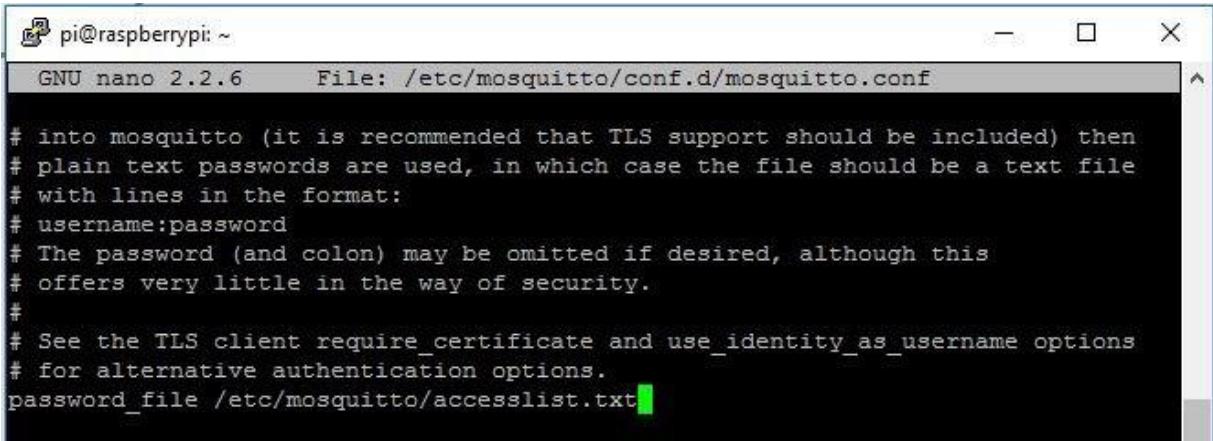
# Boolean value that determines whether clients that connect
# without providing a username are allowed to connect. If set to
# false then a password file should be created (see the
# password_file option) to control authenticated client access.
# Defaults to true.
allow_anonymous false
```

Lastly find the line containing “#password_file”:

1. remove the # to uncomment it
2. set the parameter to the path of the password file previously created

The line becomes:

```
password_file /etc/mosquitto/accesslist.txt
```



```
pi@raspberrypi: ~
GNU nano 2.2.6      File: /etc/mosquitto/conf.d/mosquitto.conf

# into mosquitto (it is recommended that TLS support should be included) then
# plain text passwords are used, in which case the file should be a text file
# with lines in the format:
# username:password
# The password (and colon) may be omitted if desired, although this
# offers very little in the way of security.
#
# See the TLS client require_certificate and use_identity_as_username options
# for alternative authentication options.
password_file /etc/mosquitto/accesslist.txt
```

Now the server is properly configured, restart the service and you're up to go:

```
pi@raspberry:~ $ sudo service mosquitto restart
```

Editing the MQTT Broker

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.

Click the Add new mqtt-broker option.

Edit mqtt in node

Server	Add new mqtt-broker...	
Topic	Topic	
Name	Name	

Ok Cancel

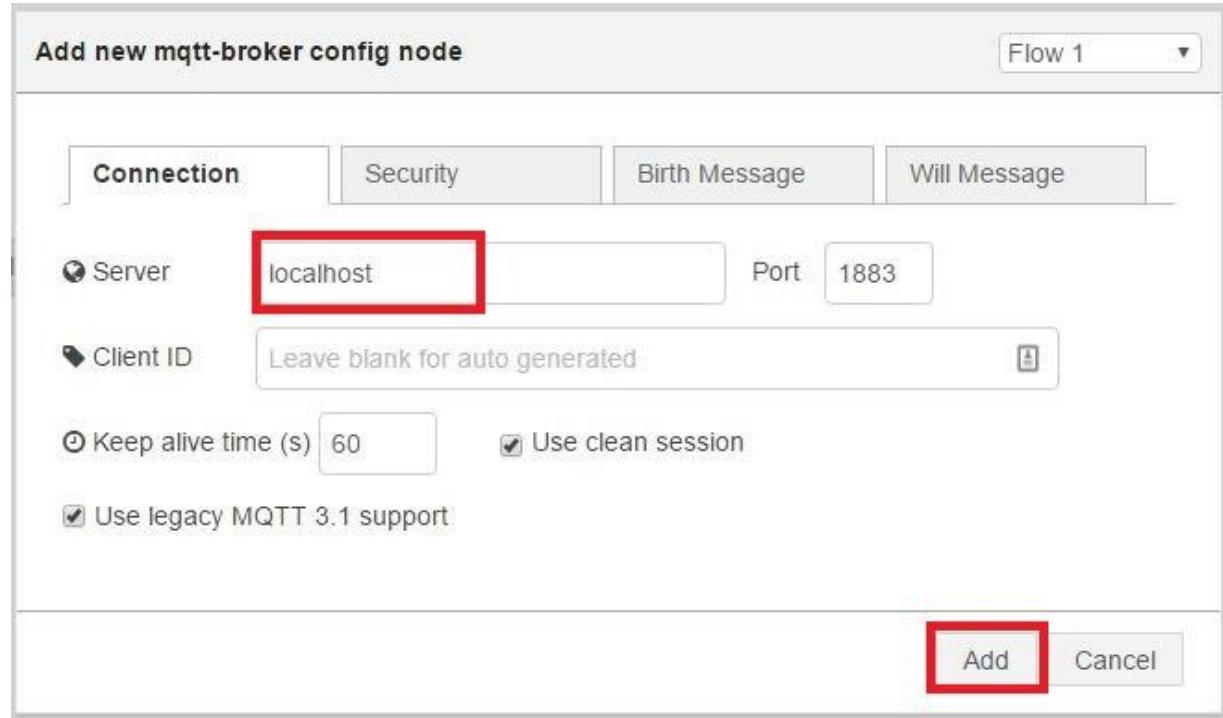
Go to the **Security** tab, Enter your username and password.

Edit mqtt-broker config node * Flow 1

Connection	Security	Birth Message	Will Message
Username	rui		
Password		
<input type="checkbox"/> Enable secure (SSL/TLS) connection			
<input checked="" type="checkbox"/> Verify server certificate			

2 nodes use this config Delete Update Cancel

Then go to the Connection tab, type **localhost** in the server field and I'll the other settings are configured properly by default.



Press **Add** and the MQTT output node automatically connects to your broker.

Now only devices with username and password can connect to your MQTT broker.

Unit 6 - Exporting Node-RED Nodes

This Unit shows how to export your Node-RED Nodes.

This is useful if you need to:

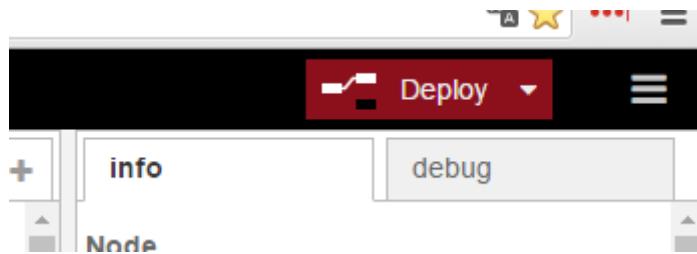
- **Backup** your Node-RED flow
- **Move** your flow to another Raspberry Pi (or machine)
- **Share** your Node-RED project with others

Example

Imagine that you had the following nodes in your flow:



You would need to click the deploy button on the top-right corner to save your application.

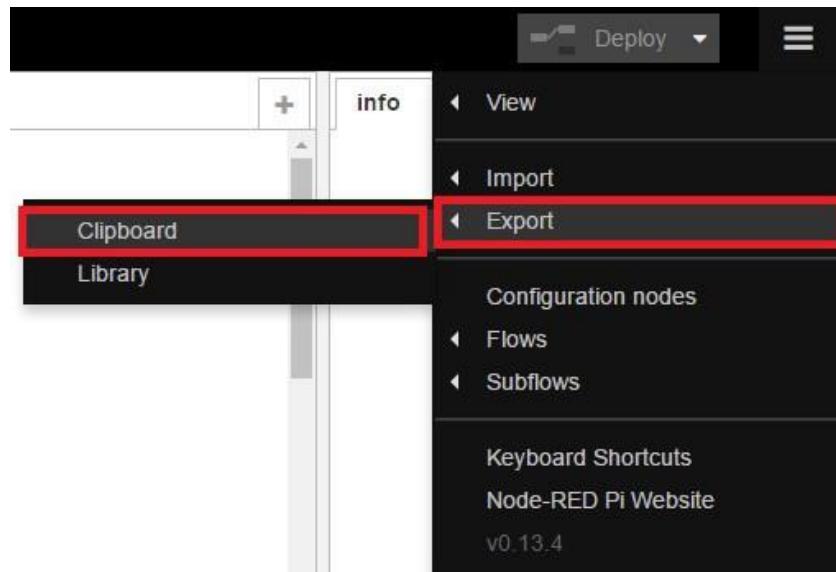


Then you would select with your mouse all the nodes that you wish to export and they would be highlighted in orange (as shown below):

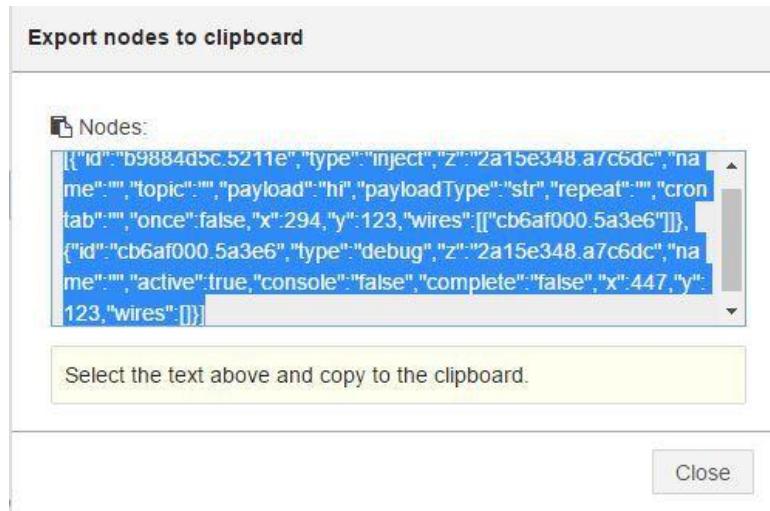


Exporting

Now with the nodes highlighted, you open the top Menu, go to Export and select Clipboard.



A new window opens. Copy the text that appears and save it.



This is how the text looks for the flow demonstrated in this example:

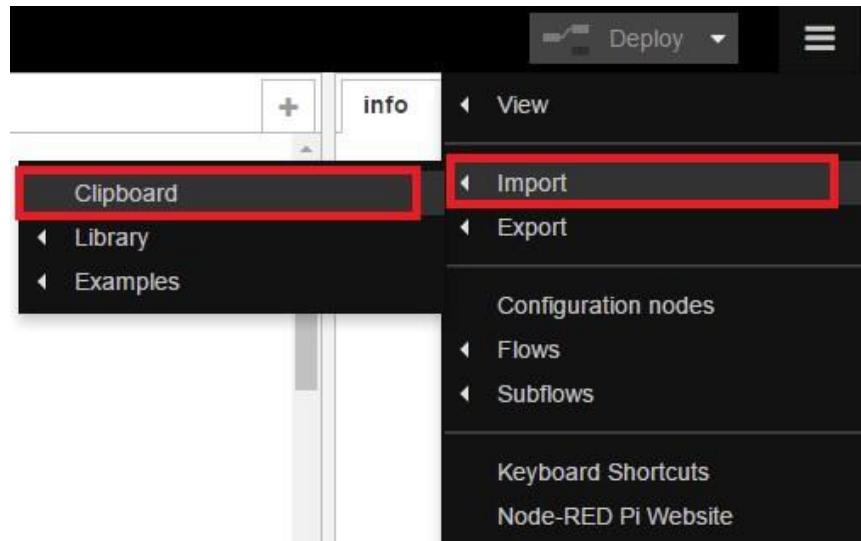
```
[{"id": "b9884d5c.5211e", "type": "inject", "z": "2a15e348.a7c6dc", "name": "", "topic": "", "payload": "hi", "payloadType": "str", "repeat": "", "cronTab": "", "once": false, "x": 294, "y": 123, "wires": [{"cb6af000.5a3e6"}]}, {"id": "cb6af000.5a3e6", "type": "debug", "z": "2a15e348.a7c6dc", "name": "", "active": true, "console": "false", "complete": "false", "x": 447, "y": 123, "wires": []}]
```

```
"name": "", "topic": "", "payload": "hi", "payloadType": "str", "repeat": "", "crontab": "", "once": false, "x": 294, "y": 123, "wires": [ ["cb6af000.5a3e6"] ], {"id": "cb6af000.5a3e6", "type": "debug", "z": "2a15e348.a7c6dc", "name": "", "active": true, "console": "false", "complete": false, "x": 447, "y": 123, "wires": [] }]
```

Importing

Now you could go to another Raspberry Pi or a machine that has Node-RED installed and you could simply import your flow.

You only have to go to the **Import** menu and paste your nodes in text format (as shown below).



Important: If you have previously installed extra nodes (for example Node-RED Dashboard), you'll also have to install those nodes in your new machine, otherwise the Import process will not work.

Unit 7 - Sending Linux Commands Through the Node-RED Dashboard

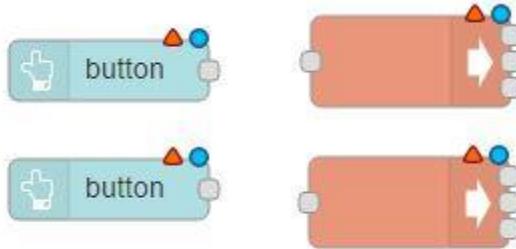
Sometimes you don't want to open the terminal and establish an SSH communication to power off your Raspberry Pi.

So, It would be useful to send the `poweroff` and the `reboot` commands through the Node-RED Dashboard with a button press. That's exactly what you are going to learn in this extra Unit.

Creating the Flow

Follow these 7 steps to create a flow:

1 – Drag Nodes



2 – Poweroff Button in Config tab

Edit button node

Cancel Done

Group: Group 1 [Config]

Size: auto

Icon: optional icon

Label: Poweroff

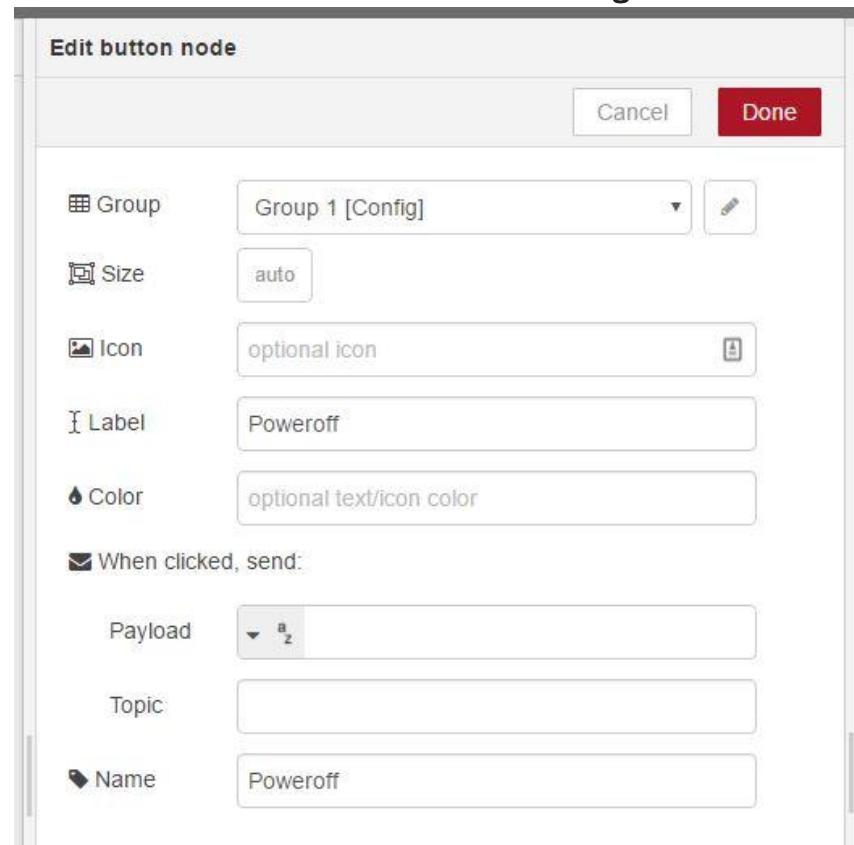
Color: optional text/icon color

When clicked, send:

Payload:

Topic:

Name: Poweroff



3 – poweroff Command

Edit exec node

Cancel Done

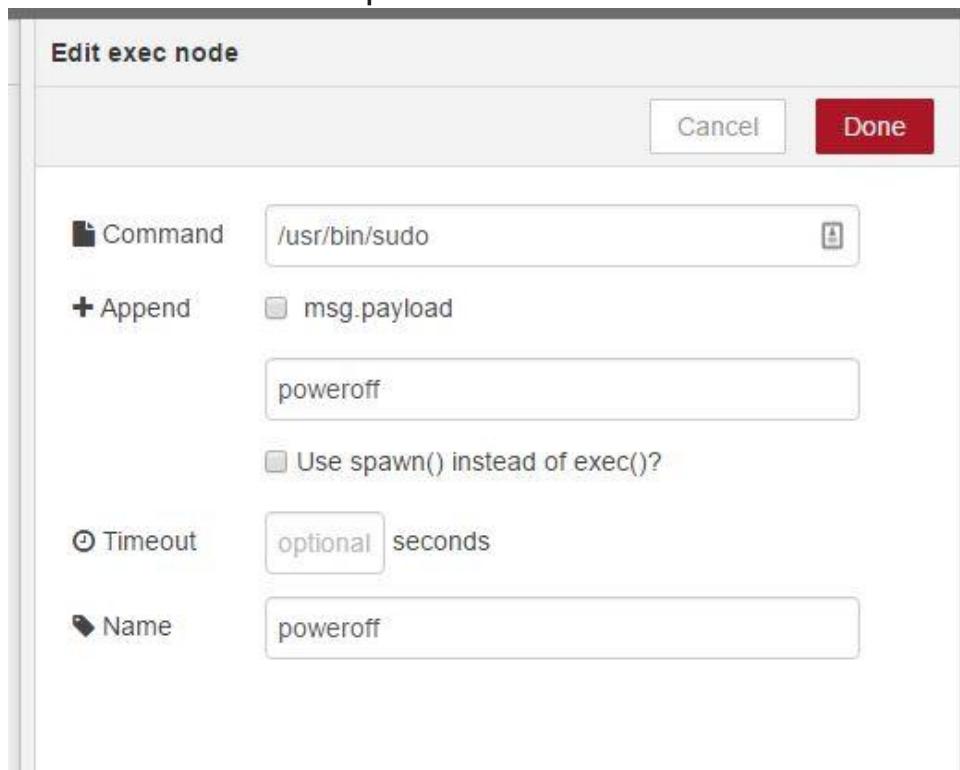
Command: /usr/bin/sudo

Append: msg.payload
poweroff

Use spawn() instead of exec()?

Timeout: optional seconds

Name: poweroff



4 – Reboot Button in Config tab

Edit button node

Cancel Done

Group: Group 1 [Config]

Size: auto

Icon: optional icon

Label: Reboot

Color: optional text/icon color

When clicked, send:

Payload:

Topic:

Name: Reboot

5 – reboot Command

Edit exec node

Cancel Done

Command: /usr/bin/sudo

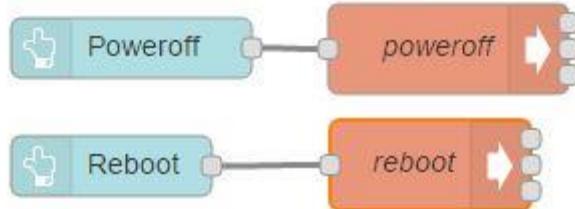
Append: msg.payload
reboot

Use spawn() instead of exec()?

Timeout: optional seconds

Name: reboot

6 – Final Flow



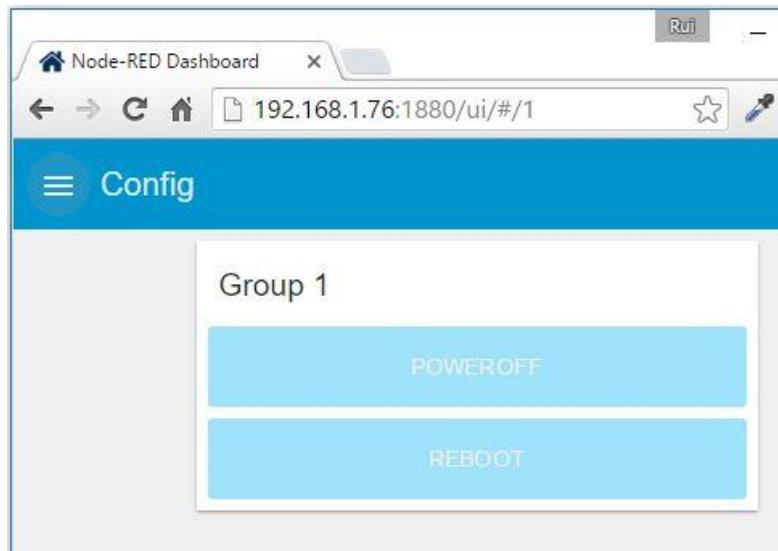
7 – Deploy Your Application



Open Node-RED Dashboard

When you go to the **Config** tab in the UI, it shows two new buttons.

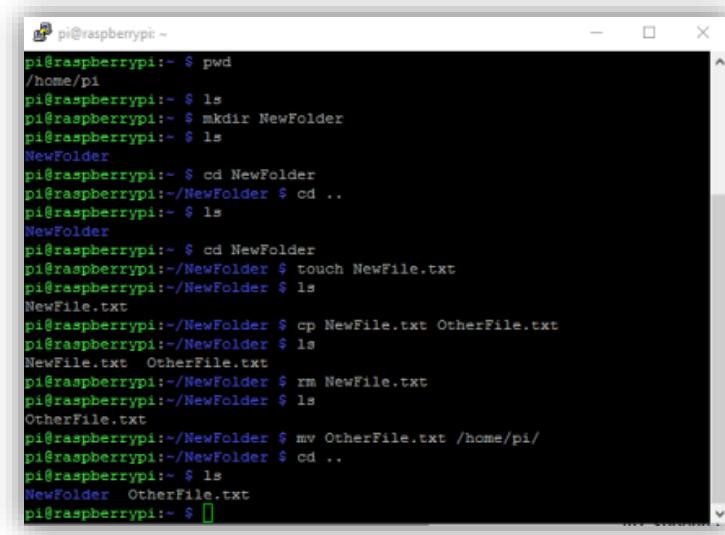
If you press the first button your Raspberry Pi immediately powers off safely and when you click the second button the RPi reboots.



You can apply this method to send almost any other Linux command to your Raspberry Pi through the UI.

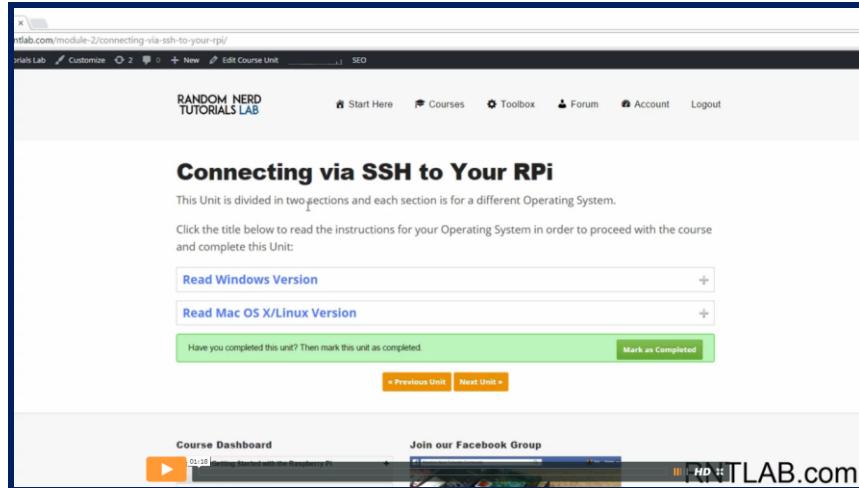
Module 14

Extra #2 - Getting Started with Linux



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $ ls
pi@raspberrypi:~ $ mkdir NewFolder
pi@raspberrypi:~ $ ls
NewFolder
pi@raspberrypi:~ $ cd NewFolder
pi@raspberrypi:~/NewFolder $ cd ..
pi@raspberrypi:~ $ ls
NewFolder
pi@raspberrypi:~ $ cd NewFolder
pi@raspberrypi:~/NewFolder $ touch NewFile.txt
pi@raspberrypi:~/NewFolder $ ls
NewFile.txt
pi@raspberrypi:~/NewFolder $ cp NewFile.txt OtherFile.txt
pi@raspberrypi:~/NewFolder $ ls
NewFile.txt OtherFile.txt
pi@raspberrypi:~/NewFolder $ rm NewFile.txt
pi@raspberrypi:~/NewFolder $ ls
OtherFile.txt
pi@raspberrypi:~/NewFolder $ mv OtherFile.txt /home/pi/
pi@raspberrypi:~/NewFolder $ cd ..
pi@raspberrypi:~ $ ls
NewFolder OtherFile.txt
pi@raspberrypi:~ $ [ ]
```

Unit 1 - Learning Basic Linux Commands



Video # 28 - <https://rntlab.com/28hasvideos>

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.

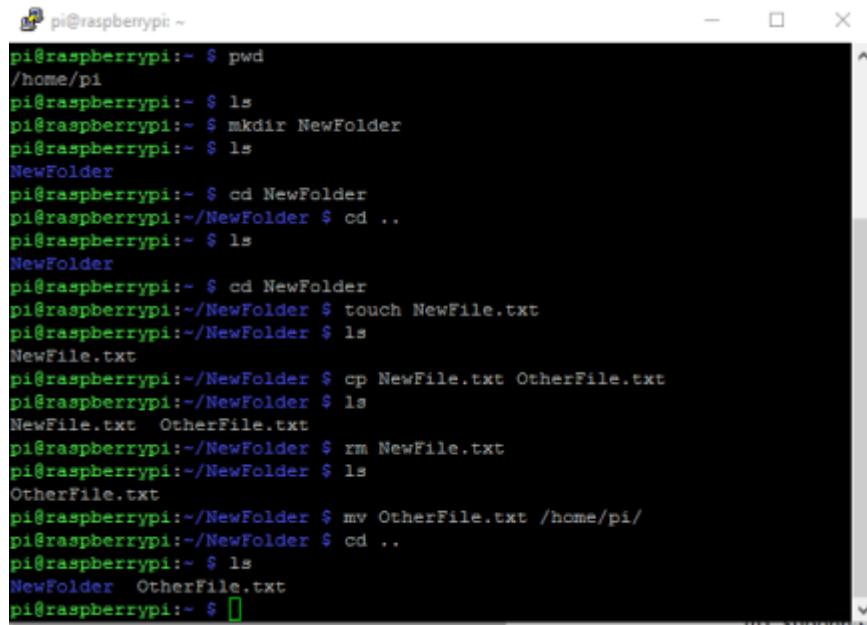
But it doesn't need to be that way, because in reality we can break it down to just a few basic commands that you need to know to do everything. After you learn these commands you'll hopefully feel really comfortable using the terminal.

In Module 2, I'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 Units 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 Module as a reference to remind how to do something.

Important: If there are commands that you need to run to do something, I'll post them in that Module or Unit so you don't get lost.

Unit 2 - Exploring the Linux File System



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ pwd
/home/pi
pi@raspberrypi:~ $ ls
pi@raspberrypi:~ $ mkdir NewFolder
pi@raspberrypi:~ $ ls
NewFolder
pi@raspberrypi:~ $ cd NewFolder
pi@raspberrypi:~/NewFolder $ cd ..
pi@raspberrypi:~ $ ls
NewFolder
pi@raspberrypi:~ $ cd NewFolder
pi@raspberrypi:~/NewFolder $ touch NewFile.txt
pi@raspberrypi:~/NewFolder $ ls
NewFile.txt
pi@raspberrypi:~/NewFolder $ cp NewFile.txt OtherFile.txt
pi@raspberrypi:~/NewFolder $ ls
NewFile.txt OtherFile.txt
pi@raspberrypi:~/NewFolder $ rm NewFile.txt
pi@raspberrypi:~/NewFolder $ ls
OtherFile.txt
pi@raspberrypi:~/NewFolder $ mv OtherFile.txt /home/pi/
pi@raspberrypi:~/NewFolder $ cd ..
pi@raspberrypi:~ $ ls
NewFolder OtherFile.txt
pi@raspberrypi:~ $ [ ]
```

It's time to play around with the command line.

For starters, type `pwd`, which means print working directory:

```
pi@raspberry:~ $ pwd
/home/pi
```

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.

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 true for all other commands and for 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 working directory with `ls`:

```
pi@raspberry:~ $ ls
```

Right now your directory is empty, so you won't see anything when you try to list your files and folders. Want to create a new folder? Use `mkdir` followed by the name you want to give the folder:

```
pi@raspberry:~ $ mkdir NewFolder  
pi@raspberry:~ $ ls  
NewFolder
```

To navigate, we'll be using the `cd` command, followed by the location you want to move to. This can be done like so:

```
pi@raspberry:~ $ cd NewFolder  
pi@raspberry:~/NewFolder $
```

This moved you to the `NewFolder` directory that you just created.

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):

```
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:~/NewFolder $ 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.

Unit 3 - Editing Files using the Terminal

Nano is an easy to use text editor that is installed by default in Raspbian distribution and many other Linux distributions.

Using Nano

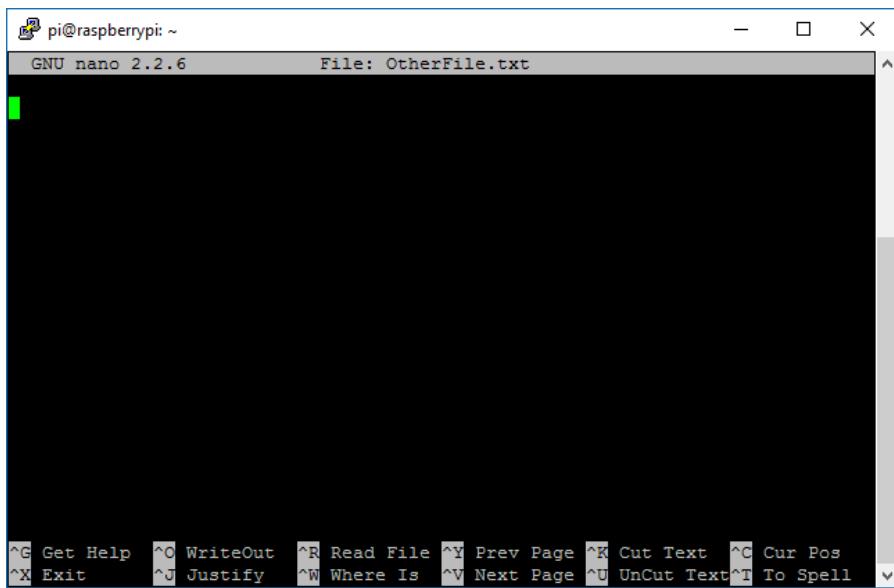
You can run nano by just typing in `nano` at the command prompt.

You can use the following commands to edit the `OtherFile.txt` created in the previous Unit:

```
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:

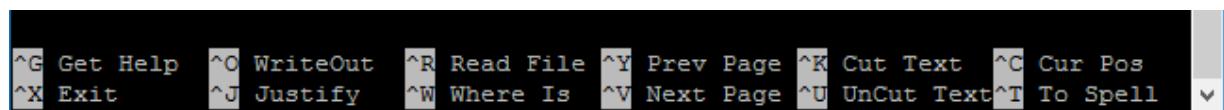


At the top row, you'll see the name of the program version number, the name and extension of the file you're editing, and whether the file has been modified since it was last saved.

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.

pi@raspberrypi: ~

GNU nano 2.2.6

Main nano help text

The nano editor is designed to emulate the functionality and ease-of-use of the UW Pico text editor. There are four main sections of this help text: the first section describes the main editor window, the second section describes the status line, the third section describes the command-line interface, and the fourth section describes the keyboard shortcuts.

The notation for shortcuts is as follows: Control-key sequences are noted with a caret (^) symbol and can be entered either by using the Control key alone or by pressing the Meta (M-) symbol and then the key. Keystrokes are noted with a caret (^) symbol followed by a key or sequence of keys. Alternative keys are shown in parentheses.

^G	(F1)	Display this help text
^X	(F2)	Close the current file buffer / Exit from nano
^O	(F3)	Write the current file to disk
^J	(F4)	Justify the current paragraph
^R	(F5)	Insert another file into the current one
^W	(F6)	Search for a string or a regular expression
^Y	(F7)	Go to previous screen
^V	(F8)	Go to next screen
^K	(F9)	Cut the current line and store it in the cutbuffer
^U	(F10)	Paste from the cutbuffer into the current line
^C	(F11)	Display the position of the cursor
^T	(F12)	Invoke the spell checker, if available
M-\	(M-)	Go to the first line of the file
M-/	(M-?)	Go to the last line of the file
^_	(F13)	(M-G) Go to line and column number
^\\	(F14)	(M-R) Replace a string or a regular expression
^^	(F15)	(M-A) Mark text at the cursor position
M-W	(F16)	Repeat last search
M-^	(M-6)	Copy the current line and store it in the cutbuffer
M-}		Indent the current line
M-{		Unindent the current line
^F		Go forward one character
^B		Go back one character
^Space		Go forward one word
M-Space		Go back one word
^P		Go to previous line
^N		Go to next line
^A		Go to beginning of current line
^E		Go to end of current line
M-((M-9)	Go to beginning of paragraph; then of previous paragraph
M-)	(M-0)	Go just beyond end of paragraph; then of next paragraph
M-]		Go to the matching bracket
M--	(M-_)	Scroll up one line without scrolling the cursor
M+-	(M-=)	Scroll down one line without scrolling the cursor
M-<	(M-,,)	Switch to the previous file buffer
M->	(M-.)	Switch to the next file buffer
M-V		Insert the next keystroke verbatim
^I		Insert a tab at the cursor position
^M		Insert a newline at the cursor position
^D		Delete the character under the cursor
^H		Delete the character to the left of the cursor

^Y Prev Page **^P** Prev Line **^X** Exit
^Y Next Page **^N** Next Line

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.

pi@raspberrypi: ~

GNU nano 2.2.6 File: OtherFile.txt Modified

Just add your text here...

^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U Uncut Text ^T To Spell

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

pi@raspberrypi: ~

GNU nano 2.2.6 File: OtherFile.txt Modified

Just add your text here...

Save modified buffer (ANSWERING "No" WILL DESTROY CHANGES) ?

Y Yes
N No ^C Cancel

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.

Unit 4 - Managing Software on Your Raspberry Pi



```
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: Wed Dec 30 19:36:28 2015 from desktop-c5btdof.lan  
pi@raspberrypi:~ $ python  
Python 2.7.9 (default, Mar  8 2015, 00:52:26)  
[GCC 4.9.2] on linux2  
Type "help", "copyright", "credits" or "license" for more information.  
>>> █
```

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 and straightforward.

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, whom you already know 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 the 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 that we are going to explore in the next Module.

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 absolutely 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.

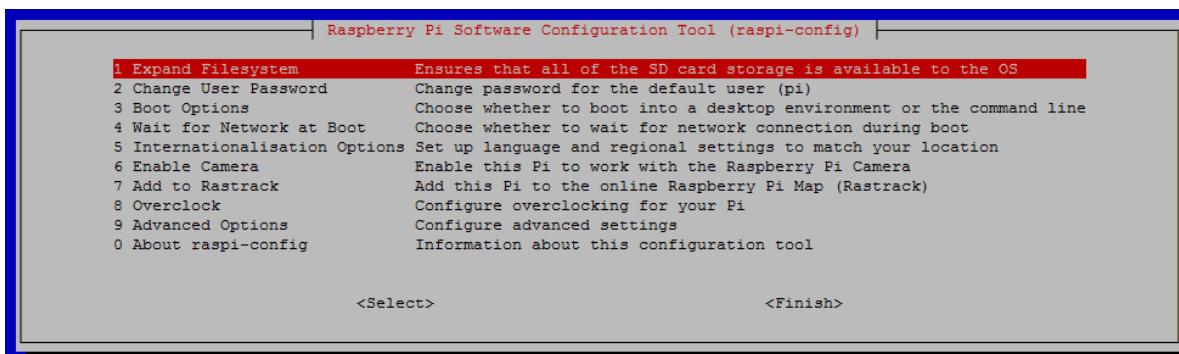
Unit 5 - Changing the Raspberry Pi Settings

To change the 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. Expand Filesystem
2. Change User Password
3. Boot Options
4. Wait for Network at Boot

5. Internationalisation Options
6. Enable Camera
7. Add to Rastrack
8. Overclock
9. Advanced Options
10. About raspi-config

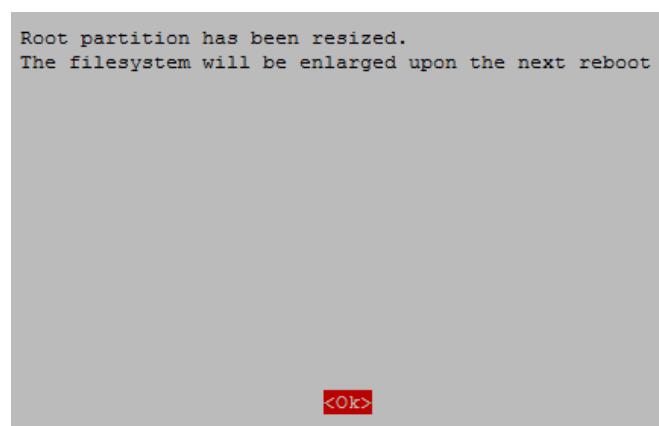
Most configurations are pretty self-explanatory and for this course you only need to change one setting (as shown in the next section).

Expanding your file system

I recommend expanding your file system.

Choosing option 1 from the raspi-config menu will expand your installation to fill the rest of the microSD card, giving you more space to use for files.

Note: you will need to reboot the Raspberry Pi to make this available. Note there is no confirmation; selecting the option begins the partition expansion immediately (as shown in the Figure below).



Right now you don't need to change anything else.

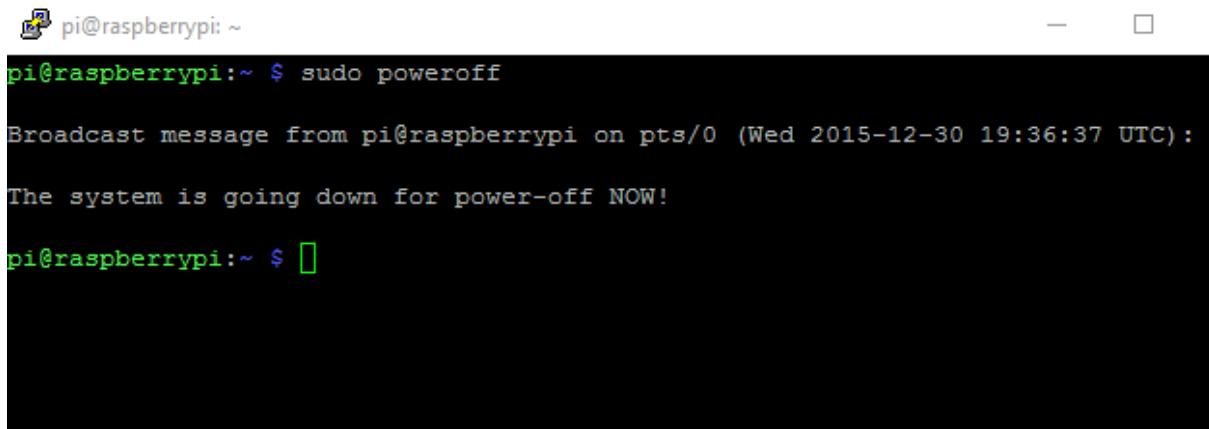
Unit 6 - Shutting Down and Rebooting

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:

```
pi@raspberry:~ $ sudo poweroff
```

You see the following information after you use the shutdown command:



A screenshot of a terminal window on a Raspberry Pi. The window title is 'pi@raspberrypi: ~'. The terminal shows the command 'sudo poweroff' being run, followed by a broadcast message from the pi user, and a confirmation that the system is going down for power-off. The terminal then ends with a prompt '\$'.

```
pi@raspberrypi:~ $ sudo poweroff
Broadcast message from pi@raspberrypi on pts/0 (Wed 2015-12-30 19:36:37 UTC):
The system is going down for power-off NOW!
pi@raspberrypi:~ $
```

To reboot, type this:

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

This is the result:

 pi@raspberrypi: ~
pi@raspberrypi:~ \$ sudo reboot
Broadcast message from pi@raspberrypi on pts/0 (Wed 2015-12-30 19:35:15 UTC):
The system is going down for reboot NOW!
pi@raspberrypi:~ \$

You need to log in again through SSH after rebooting.

Download Other RNT Products

[Random Nerd Tutorials](#) is an online resource with electronics projects, tutorials and reviews.

Creating and posting new projects takes a lot of time. At this moment, Random Nerd Tutorials has nearly 100 free blog posts with complete tutorials using open-source hardware that anyone can read, remix and apply to their own projects:
<http://randomnerdtutorials.com>

To keep free tutorials coming, there's also paid content or as I 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 the others](#).

Thanks for taking the time to read my work!

Good luck with all your projects,

-Rui Santos

[P.S. Click here for more Courses and eBooks like this one.](#)

[Click here to Download other Courses and eBooks](#)

<http://randomnerdtutorials.com/products>