**Interfacing Sensors and Peripherals with ESP 32**

# Installing the ESP32 Board in Arduino IDE (Windows, Mac OS X, Linux)

There’s an add-on for the Arduino IDE that allows you to program the ESP32 using the Arduino IDE and its programming language. In this tutorial we’ll show you how to install the ESP32 board in Arduino IDE whether you’re using Windows, Mac OS X or Linux.

## **Prerequisites: Arduino IDE Installed**

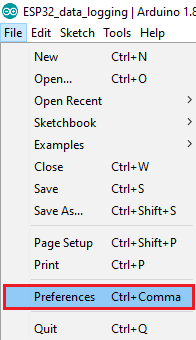
Before starting this installation procedure, make sure you have the latest version of the Arduino IDE installed in your computer. If you don’t, uninstall it and install it again. Otherwise, it may not work.

Having the latest Arduino IDE software installed from [arduino.cc/en/Main/Software](https://www.arduino.cc/en/Main/Software), continue with this tutorial.

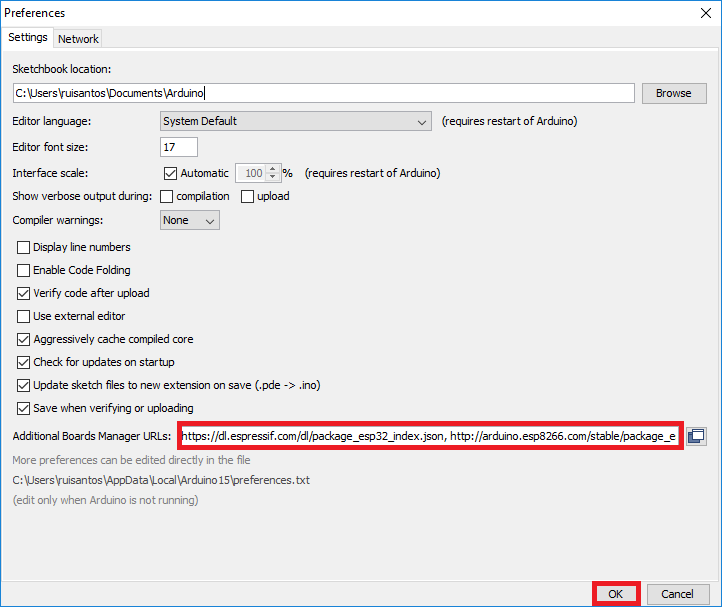
## **Installing ESP32 Add-on in Arduino IDE**

To install the ESP32 board in your Arduino IDE, follow these next instructions:

1. In your Arduino IDE, go to **File**> **Preferences**



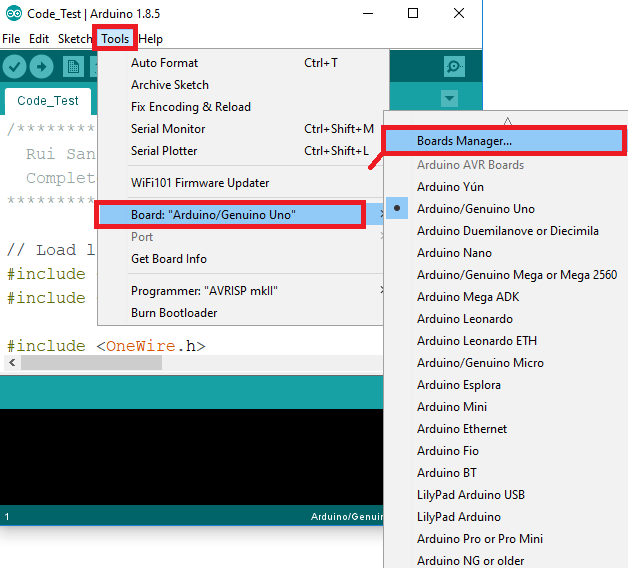
1. Enter **https://dl.espressif.com/dl/package\_esp32\_index.json** into the “Additional Board Manager URLs” field as shown in the figure below. Then, click the “OK” button:



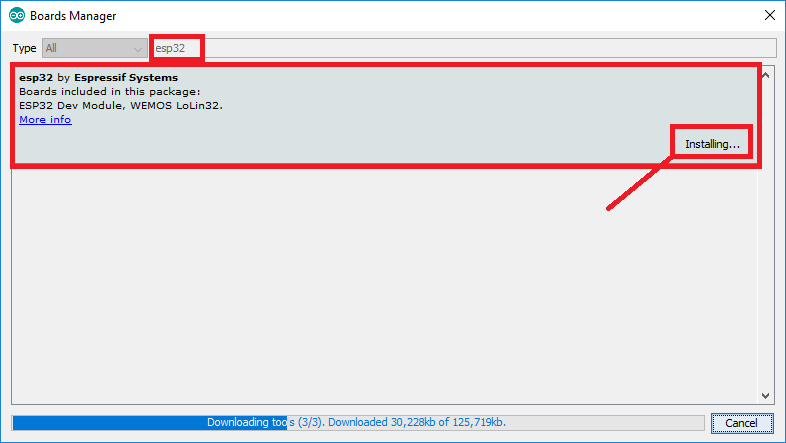
**Note:** if you already have the ESP8266 boards URL, you can separate the URLs with a comma as follows:

https://dl.espressif.com/dl/package\_esp32\_index.json, http://arduino.esp8266.com/stable/package\_esp8266com\_index.json

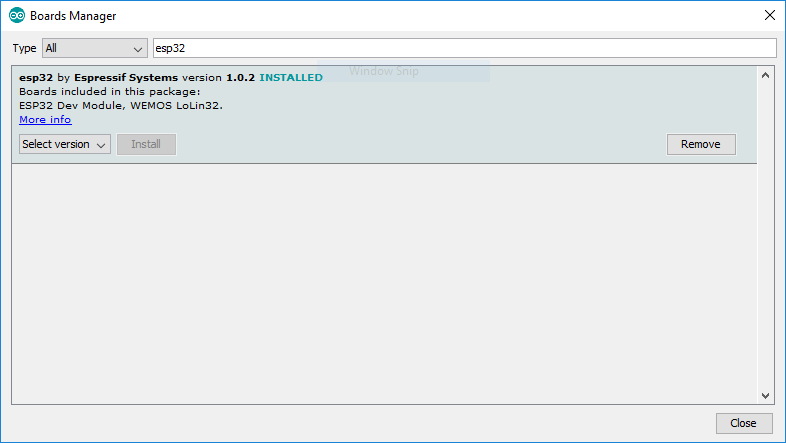
1. Open the Boards Manager. Go to **Tools** > **Board** > **Boards Manager…**



1. Search for **ESP32** and press install button for the “**ESP32 by Espressif Systems**“:



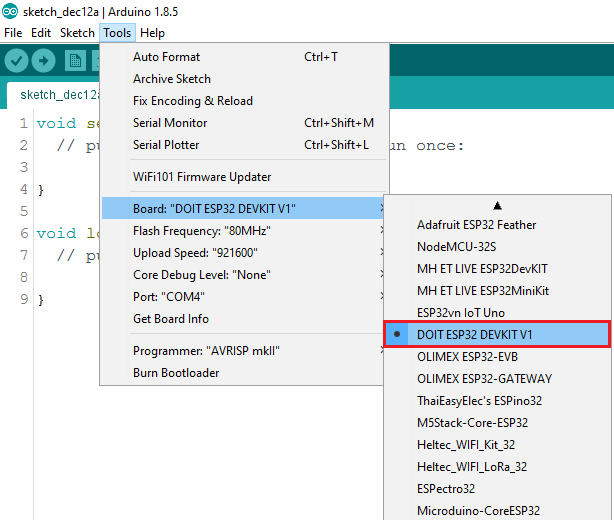
1. That’s it. It should be installed after a few seconds.



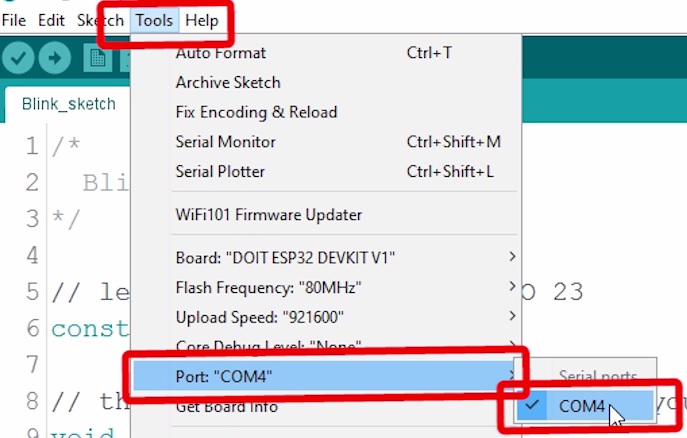
## **Testing the Installation**

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

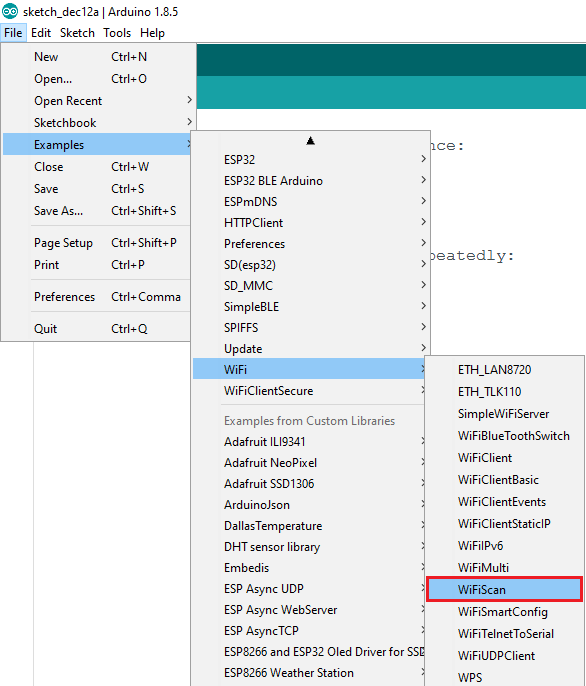
1. Select your Board in **Tools** > **Board** menu (in my case it’s the **DOIT ESP32 DEVKIT V1**)



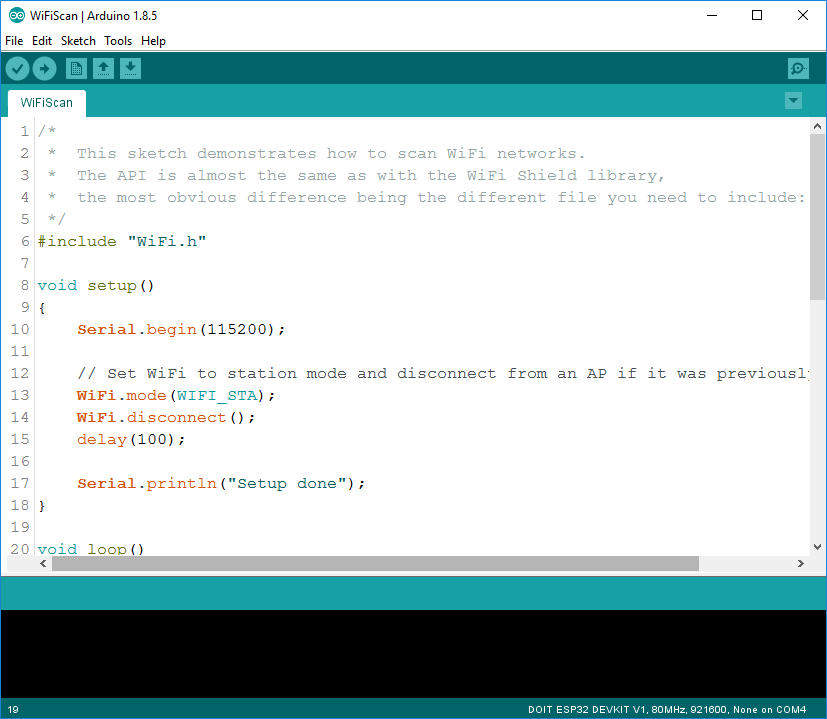
2. Select the Port (if you don’t see the COM Port in your Arduino IDE, you need to install the [CP210x USB to UART Bridge VCP Drivers](https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers)):



3. Open the following example under **File** > **Examples** > **WiFi (ESP32)** > **WiFiScan**



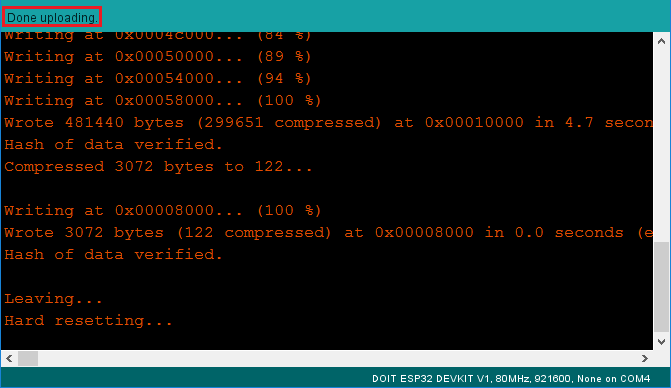
4. A new sketch opens in your Arduino IDE:



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

Arduino IDE upload WiFiScan sketch to ESP32

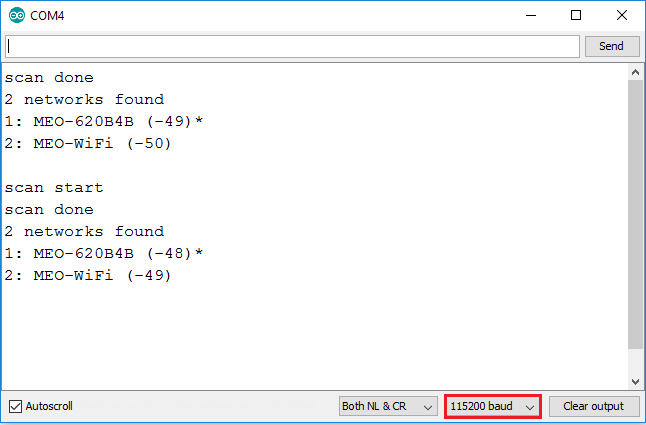
6. If everything went as expected, you should see a “**Done uploading.**” message.



7. Open the Arduino IDE Serial Monitor at a baud rate of 115200:

Open Arduino IDE Serial Monitor at baud rate 115200

8. Press the ESP32 on-board **Enable** button and you should see the networks available near your ESP32:

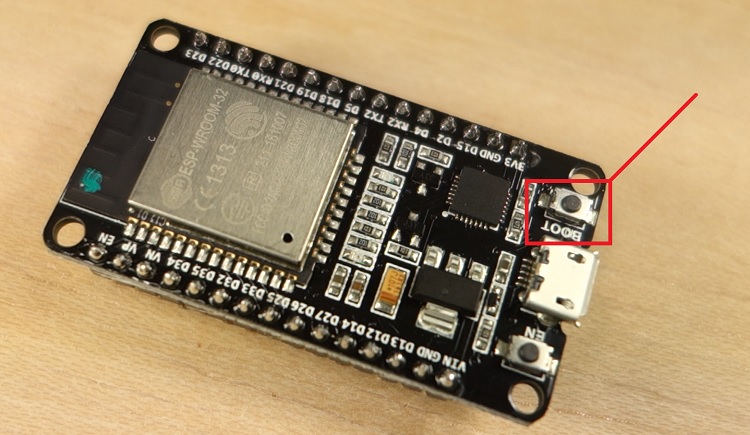


## **Troubleshooting**

If you try to upload a new sketch to your ESP32 and you get this error message “A fatal error occurred: Failed to connect to ESP32: Timed out… Connecting…“. It means that your ESP32 is not in flashing/uploading mode.

Having the right board name and COM por selected, follow these steps:

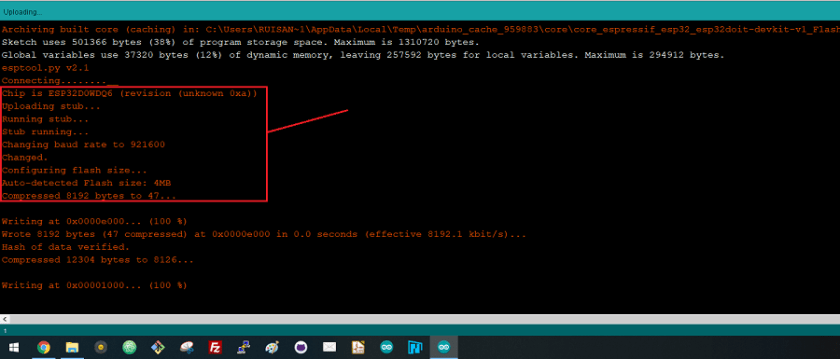
* Hold-down the “**BOOT**” button in your ESP32 board



* Press the “**Upload**” button in the Arduino IDE to upload your sketch:

Arduino IDE uploading new sketch to ESP32

* After you see the  “**Connecting….**” message in your Arduino IDE, release the finger from the “**BOOT**” button:



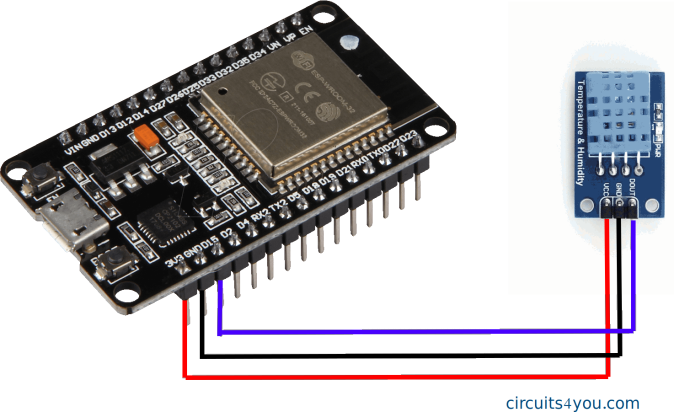
* After that, you should see the “**Done uploading**” message

That’s it. Your ESP32 should have the new sketch running. Press the “**ENABLE**” button to restart the ESP32 and run the new uploaded sketch.

**DHT11 Connections with ESP32**

The DHT11 (or DHT22 and similar) are cheap temperature and humidity sensors. The communicate with a ESP32 is over a single wire, but unfortunately it is not compatible with the 1-Wire protocol defined by Dallas Semiconductors.

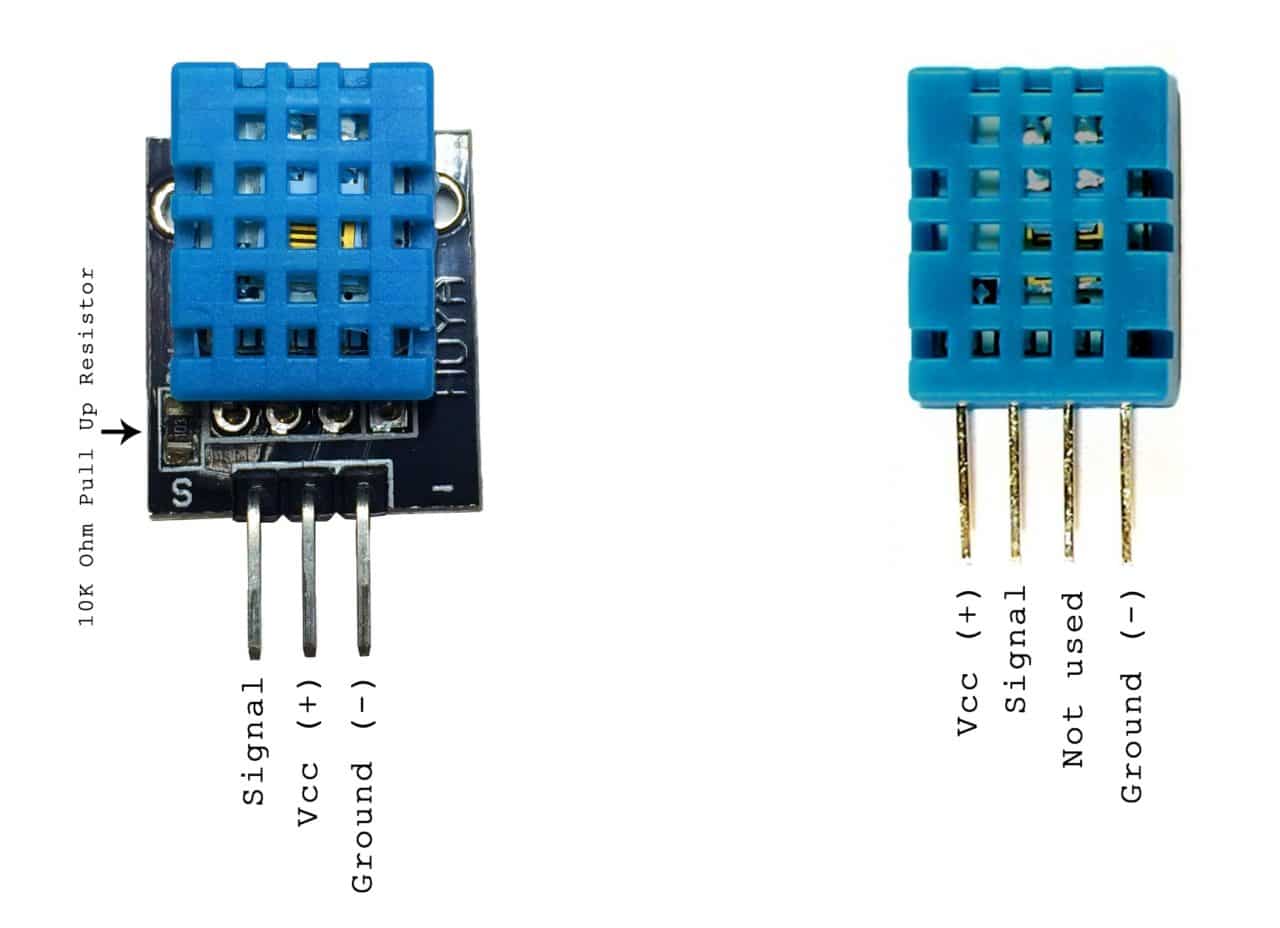
The electric connection to the ESP32 DevKit is very simple, as the DHT series can be powered direct with 3.3V. Only 3 wires are needed: VCC, GND and the data line. Data Line is connected to GPIO15 (D15).



### DHT11 Sensor Specifications

The DHT11 is a basic, ultra low-cost digital temperature and humidity sensor. It uses a capacitive humidity sensor and a thermistor to measure the surrounding air, and spits out a digital signal on the data pin (no analog input pins needed). Its fairly simple to use, but requires careful timing to grab data. The only real downside of this sensor is you can only get new data from it once every 2 seconds, so when using our library, sensor readings can be up to 2 seconds old. Compared to the DHT22, this sensor is less precise, less accurate and works in a smaller range of temperature/humidity, but its smaller and less expensive.

**If you are using module it comes with a 4.7K or 10K resistor. You need external 10K or 4.7K pull up on data pin to VCC.**



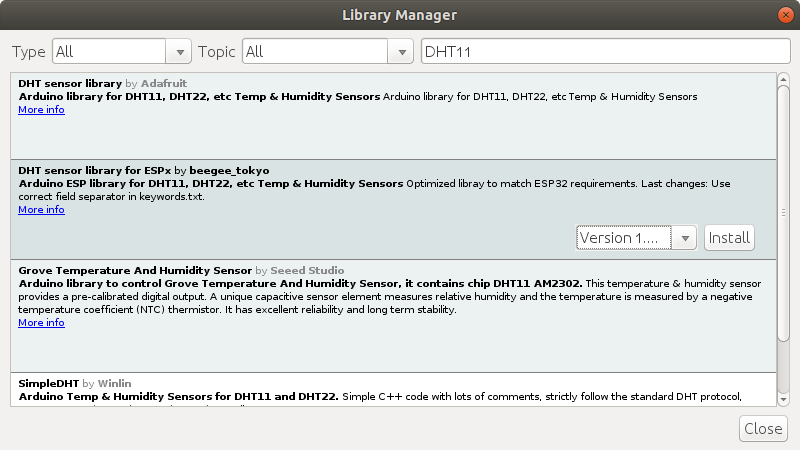
**DHT11 Specifications:**

* Low cost
* 3 to 5V power and I/O
* 2.5mA max current use during conversion (while requesting data)
* Good for 20-80% humidity readings with 5% accuracy
* Good for 0-50°C temperature readings ±2°C accuracy
* No more than 1 Hz sampling rate (once every second)
* Body size 15.5mm x 12mm x 5.5mm
* 4 pins with 0.1″ spacing

### ESP32 Program for DHT11 Sensor

Before moving directly on coding install required libraries.

Go to menu **Sketch >> Include Library >> Manage Library**

Install **DHT sensor library for ESPx by beegee** library. 

/\*

\* ESP32 NodeMCU DHT11 - Humidity Temperature Sensor Example

\* https://circuits4you.com

\*

\* References

\* https://circuits4you.com/2017/12/31/nodemcu-pinout/

\*

\*/

#include "DHTesp.h"

#define DHTpin 15    //D15 of ESP32 DevKit

DHTesp dht;

void setup()

{

  Serial.begin(115200);

  Serial.println();

  Serial.println("Status\tHumidity (%)\tTemperature (C)\t(F)\tHeatIndex (C)\t(F)");

  // Autodetect is not working reliable, don't use the following line

  // dht.setup(17);

  // use this instead:

  dht.setup(DHTpin, DHTesp::DHT11); //for DHT11 Connect DHT sensor to GPIO 17

  //dht.setup(DHTpin, DHTesp::DHT22); //for DHT22 Connect DHT sensor to GPIO 17

}

void loop()

{

  delay(dht.getMinimumSamplingPeriod());

  float humidity = dht.getHumidity();

  float temperature = dht.getTemperature();

  Serial.print(dht.getStatusString());

  Serial.print("\t");

  Serial.print(humidity, 1);

  Serial.print("\t\t");

  Serial.print(temperature, 1);

  Serial.print("\t\t");

  Serial.print(dht.toFahrenheit(temperature), 1);

  Serial.print("\t\t");

  Serial.print(dht.computeHeatIndex(temperature, humidity, false), 1);

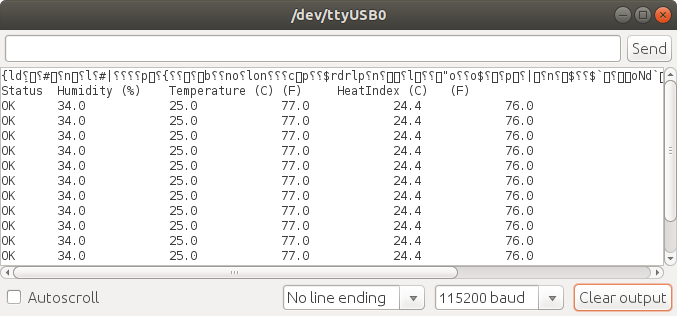
  Serial.print("\t\t");

  Serial.println(dht.computeHeatIndex(dht.toFahrenheit(temperature), humidity, true), 1);

}

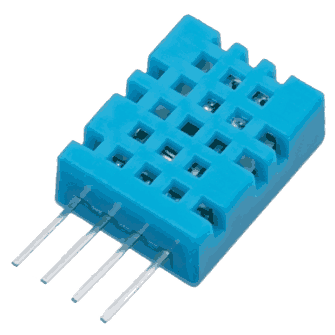
**Results**

After uploading program in ESP32 open serial monitor with 115200 baud rate.



**Interfacing Sensors with Pi**

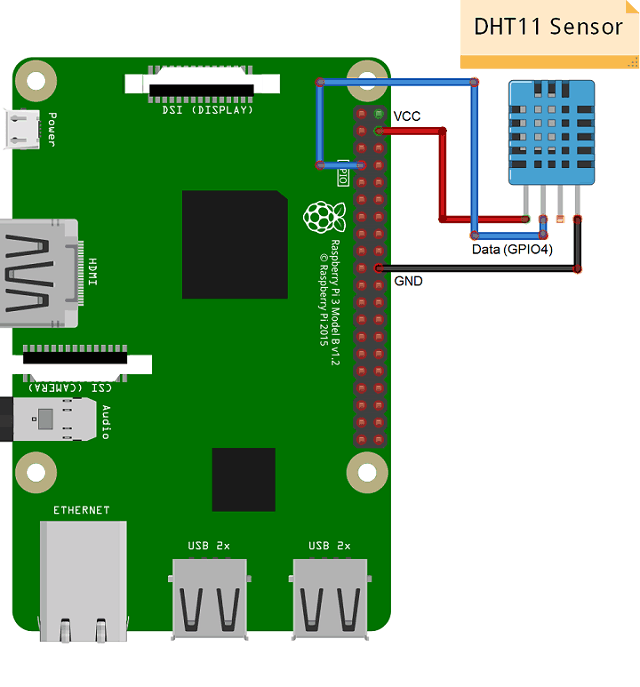
**DHT11 Interfacing with Raspberry Pi**



**DHT11 Sensor**

* DHT11 sensor measures and provides humidity and temperature values serially over a single wire.
* It can measure relative humidity in percentage (20 to 90% RH) and temperature in degree Celsius in the range of 0 to 50°C.
* It has 4 pins; one of which is used for data communication in serial form.
* Pulses of different TON and TOFF are decoded as logic 1 or logic 0 or start pulse or end of a frame.

# ****Interfacing diagram****



**DHT11 Interfacing with Raspberry Pi 3**

* Here, we are going to interface DHT11 sensor with Raspberry Pi 3 and display Humidity and Temperature on terminal.
* We will be using the DHT Sensor Python library by Adafruit from GitHub. The Adafruit Python DHT Sensor library is  created to read the Humidity and Temperature on raspberry Pi or Beaglebone Black.
* It is developed for DHT series sensors like DHT11, DHT22 or AM2302.
* Extract the library and install it in the same root directory of downloaded library by executing following command,

sudo python setup.py install

* Once the library and its dependencies has been installed, open the example sketch named simpletest from the library kept in examples folder.
* In this code, raspberry Pi reads Humidity and Temperature from DHT11 sensor and prints them on terminal. But, it read and display the value only once. So, here we made change in the program to print value continuously.

**Note:**

* Assign proper sensor type to the sensor variable in this library. Here, we are using DHT11 sensor.

sensor = Adafruit\_DHT.DHT11

* If anyone is using sensor DHT22 then we need to assign **Adafruit\_DHT.DHT22** to the sensor variable shown above.
* Also, comment out Beaglebone pin definition and uncomment pin declaration for Raspberry Pi.
* Then assign pin no. to which DHT sensor’s data pin is connected. Here, data out of DHT11 sensor is connected to GPIO4. As shown in above interfacing diagram.

**Python Program**

#!/usr/bin/python

# Copyright (c) 2014 Adafruit Industries

# Author: Tony DiCola

# Permission is hereby granted, free of charge, to any person obtaining a copy

# of this software and associated documentation files (the "Software"), to deal

# in the Software without restriction, including without limitation the rights

# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

# copies of the Software, and to permit persons to whom the Software is

# furnished to do so, subject to the following conditions:

# The above copyright notice and this permission notice shall be included in all

# copies or substantial portions of the Software.

# THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

# IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

# FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

# AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

# LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

# OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

# SOFTWARE.

import Adafruit\_DHT

# Sensor should be set to Adafruit\_DHT.DHT11,

# Adafruit\_DHT.DHT22, or Adafruit\_DHT.AM2302.

sensor = Adafruit\_DHT.DHT11

# Example using a Beaglebone Black with DHT sensor

# connected to pin P8\_11.

#pin = 'P8\_11'

# Example using a Raspberry Pi with DHT sensor

# connected to GPIO4.

pin = 4

# Try to grab a sensor reading. Use the read\_retry method which will retry up

# to 15 times to get a sensor reading (waiting 2 seconds between each retry).

while True:

humidity, temperature = Adafruit\_DHT.read\_retry(sensor, pin)

print('Temp={0:0.1f}\*C Humidity={1:0.1f}%'.format(temperature, humidity))

'''

# Note that sometimes you won't get a reading and

# the results will be null (because Linux can't

# guarantee the timing of calls to read the sensor).

# If this happens try again!

if humidity is not None and temperature is not None:

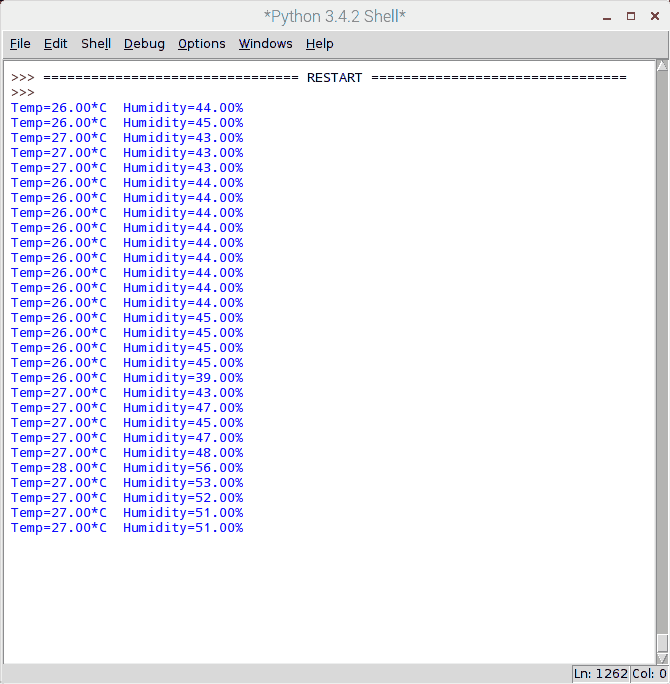
print('Temp={0:0.1f}\*C Humidity={1:0.1f}%'.format(temperature, humidity))

else:

print('Failed to get reading. Try again!')

'''

# ****DHT11 Output****



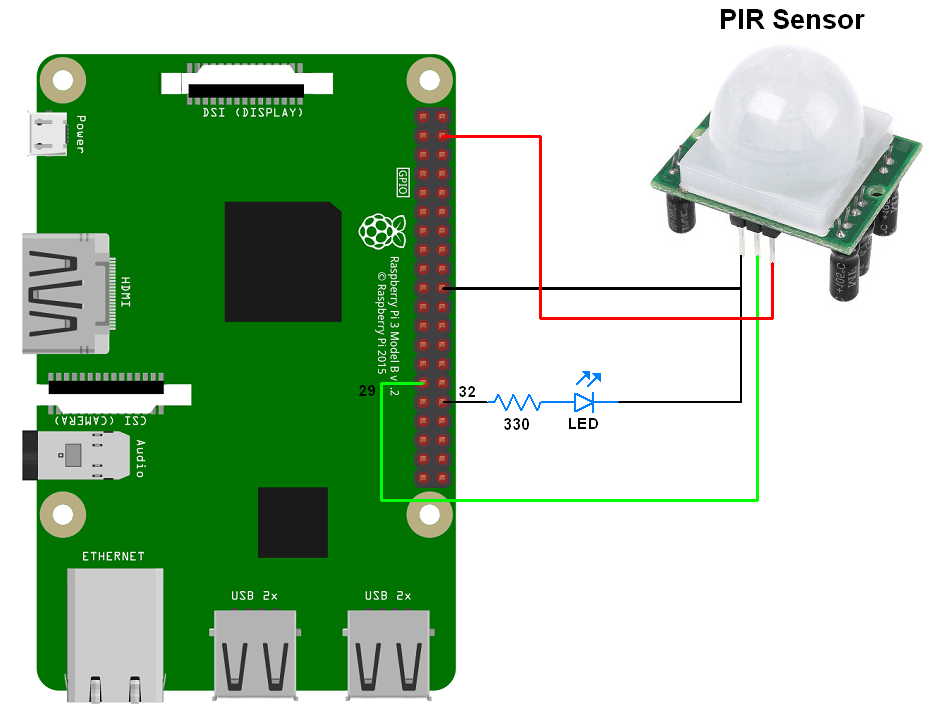
**PIR Motion Sensor**



**PIR Motion Detection Sensor**

* PIR sensor is used for detecting infrared heat radiations. This makes them useful in the detection of moving living objects that emit infrared heat radiations.
* The output (in terms of voltage) of PIR sensor is high when it senses motion; whereas it is low when there is no motion (stationary object or no object).
* PIR sensors are used in many applications like for room light control using human detection, human motion detection for security purpose at home, etc.

# ****Interfacing Diagram****



**PIR Motion Sensor Interfacing with Raspberry Pi**

Let’s interface PIR sensor with Raspberry Pi for motion detection.

* When motion is detected, PIR output goes HIGH which will be read by Raspberry Pi. So, we will turn on LED when motion is detected by PIR sensor.
* Here, LED is connected to GPIO12 (pin no. 32) whereas PIR output is connected to GPIO5 (pin no. 29). Let’s write a python based program to interface PIR motion sensor with Raspberry Pi.

# ****Python Program****

'''

Motion detection using PIR on raspberry Pi

http://www.electronicwings.com

'''

import RPi.GPIO as GPIO

PIR\_input = 29 #read PIR Output

LED = 32 #LED for signalling motion detected

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BOARD) #choose pin no. system

GPIO.setup(PIR\_input, GPIO.IN)

GPIO.setup(LED, GPIO.OUT)

GPIO.output(LED, GPIO.LOW)

while True:

#when motion detected turn on LED

if(GPIO.input(PIR\_input)):

GPIO.output(LED, GPIO.HIGH)

else:

GPIO.output(LED, GPIO.LOW)

**Actuator with Raspberry Pi with Serial Communication**

**Installation of LAMP**

From an economic point of view, you should know that web hosting services are not free and that you have to pay every month / year. Unlike the Raspberry who just need to a connection.  
In addition, by choosing Raspberry, you have the possibility to modify your services like you want (examples: the size of the disk, the hosting of Database, etc.), which is generally not the case with specialized hosts , Which often sell shared hosting with low configuration capacity.

**What is Apache?**

First, we will install Apache, which is the web server as such.  
When we speak of a web server, we often think about the machine, but this term also refers to the software that allows the machine to analyze user requests (in http form), and to return the file corresponding to the request (Or an error if the file isn’t found, or the query incorrectly formulated).  
As part of Apache, it’s software that we talk about.

At the moment, Apache is the most used web server, with about 60% market share. Apache has its own license, used by many other projects. In addition, the massive use of Apache (which has become the standard for web servers), coupled with its high popularity, has led to a tremendous abundance of documentation, courses, and other books dealing with its use, and his security

Whether it is for the Raspberry Pi and Raspbian, or for a more general-purpose machine, Apache is therefore a safe choice, and the skills you will be able to acquire on the subject will always be useful.

### Apache installation

Before installing the server, make sure we have an up-to-date machine. To do this we must have administrator rights, either because of the sudo command.

sudo apt update

sudo apt upgrade

sudo apt update

Once the Raspberry Pi is up to date, we will install the Apache server.

sudo apt install apache2

By the way, we’ll take advantage of it to give rights to the apache file that you can easily manage your sites. To do this, run the following commands:

sudo chown -R pi:www-data /var/www/html/

sudo chmod -R 770 /var/www/html/

### Check if Apache is working

Once the installation completed, we can test that Apache is working properly by going to the Raspberry address.  
To do this, it’s necessary to try to access to the Raspberry from port 80 (this port not being opened from the outside, it will have to do since the Raspberry itself). Do not worry, it’s very easy. Simply open the Raspberry web browser, and go to “http://127.0.0.1”. You should then get a page with a message like “It works! “And plenty of other text.  
If you do not already have a GUI on your Raspbian, or you use SSH to connect to your Raspberry, you can use the following command:

wget -O check\_apache.html http://127.0.0.1

This command will **s**ave the HTML code of the page in the file “check\_apache.html” in the current directory.  
So you only have to read the file with the command

cat ./check\_apache.html

If you see marked at a location in the code “It works! ” is that Apache is working.

Apache uses the directory “/var/www/html” as the root for your site. This means that when you call your Raspberry on port 80 (http), Apache looks for the file in “/var/www/html”.  
For example, if you call the address “http://127.0.0.1/example”, Apache will look for the “example” file in the “/var/www/html” directory.  
To add new files, sites, etc., you will need to add them to this directory.

You can now use your Raspberry to make a site in HTML, CSS and JavaScript, internally.  
However, you may want to quickly allow interactions between the site and the user. For example, to allow the user to register, etc. For this, you are going to need PHP.

### PHP installation on your Raspberry Pi

### What is PHP?

First of all, you should know that PHP is an interpreted language. And as in the case of servers, the acronym PHP can have several meanings. In fact, when we talk about PHP, we can talk about either the language or the interpreter.  
Here, when we talk about installing PHP, it means that we will install the interpreter, in order to use the language.

PHP (the language this time) is mainly used to make a site dynamic, that is to say that the user sends information to the server which returns the modified results according to this information. Conversely, a static site doesn’t adapt to information provided by a user. It’s saved as a file once for all, and will always deliver the same content.

PHP is free, and maintained by the PHP Foundation, as well as Zend Enterprise, and various other companies (it should be noted that Zend is also the author of the famous Zend PHP framework, widely used and recognized in the world of ” business).

It’s one of the most widely used programming languages, and it is even the most used for web programming, with about 79% market share.

Again, all the skills you can acquire, on the language, or on the installation and configuration of the interpreter, will always be useful. So we can only advise you to learn the PHP, which is really a wonderful language and too often underestimated.

### How to install PHP

We will again use the administrator to install PHP with the command line.

sudo apt install php php-mbstring

### Control if PHP is working

To know if PHP is working properly, it’s not very complicated, and **the method is quite similar to the one used for Apache.**

You will **first delete the file “index.html”** in the directory “/var/www/html”.

sudo rm /var/www/html/index.html

Then create an “index.php” file in this directory, with this command line

echo "<?php phpinfo ();?>" > /var/www/html/index.php

From there, **the operation is the same as for the Apache check.** You try to access your page, and you should have a result close to this image (if you do not have an interface, use the same method as before, and look for the words “PHP Version”).

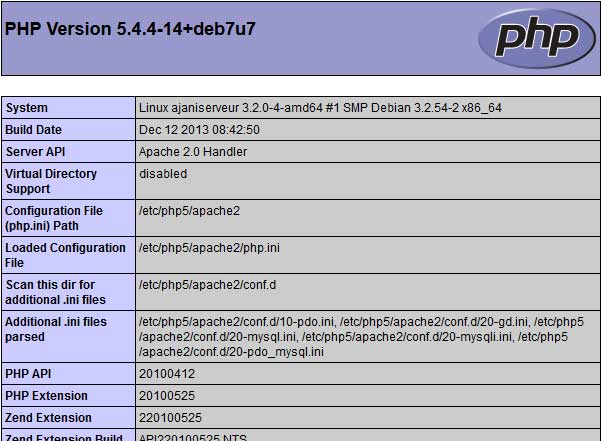
[](http://raspberry-pi.fr/wp-content/uploads/2014/03/phpinfo.jpg)

Table generated by the phpinfo command on a raspberry.

### A MySQL database for your server

### A DBMS what’s it ? Why MySQL ?

Now that we have set up PHP, you will probably **want to store information** for use in your sites. For this purpose, databases are most often used.  
We will therefore set up a DBMS (Database Management System), namely MySQL.

**MySQL is a free**, powerful, massively used DBMS (about 56% market share of free DBMS). Here again, MySQL is so essential to development, whatever the language, that you must absolutely learn and master it

### How to install MySQL

To do this, we will install mariadb-server **and** php-mysql (which will serve as a link between php and mysql)

sudo apt install mariadb-server php-mysql

### Verify that MySQL is working correctly

To check the operation of MySQL, this time **we will only use the command line.** To do this, we will simply connect via the command:

sudo mysql --user=root

We will no delete the default mysql root user and create a new mysql root user, because the default one can only be used with Linux root account, and so not available for the webserver and php scripts.

To do so, once your connect to MySQL, simply run thoses commands (replace password with the password you want) :

DROP USER 'root'@'localhost';

CREATE USER 'root'@'localhost' IDENTIFIED BY 'password';

GRANT ALL PRIVILEGES ON \*.\* TO 'root'@'localhost' WITH GRANT OPTION;

So you now have **a web server, connected to PHP and MySQL.** That’s all it takes.

(On your nexts connections, you will be able to connect to mysql without using sudo, with the command mysql --user=root --password=yourmysqlpassword).

### Add PHPMyAdmin

The installation of PHPMyAdmin is absolutly not necessary. **In this installation, we will not take care about any special security settings!**

The PHPMyAdmin installation is pretty quick and easy, we simply have to use the packet manager with this command :

sudo apt install phpmyadmin

PHPMyAdmin installation program will ask you few question. About the dbconfig-common part, choose to not use it (as we have already configure our database). About the server to configure PHPMyAdmin for, choose Apache. And the root password is the one you set for MySQL.

You should also enable **mysqli** extension using the above command :

sudo phpenmod mysqli

sudo /etc/init.d/apache2 restart

### Check that PHPMyAdmin is working properly

To check that PHPMyAdmin works, you will simply try to access it, using the address of your Raspberry followed by /phpmyadmin. For example, locally it will be http://127.0.0.1/phpmyadmin

If you still get an error, it could be because PHPMyAdmin has moved to another directory. In this case, try the command

sudo ln -s /usr/share/phpmyadmin /var/www/html/phpmyadmin

Now, we can access to PHPMyAdmin from Raspberry Pi’s browser, with the url : http://127.0.0.1/phpmyadmin

### Making a server accessible from the web

Your web server is ready. However, you probably cannot access it from the internet. Indeed, it would be necessary for that your modem to redirects the requests to your Raspberry, the good ports.  
To put these redirections in place, and even get a URL, you should look to DynDNS and port forwarding!

**Installing vsftpd**

In order to allow the transfer of files between your Mac and your Raspberry Pi, you'll need to

set up an FTP connection. To do so,

Enter this command to allow you to take full control of the web server's root directory (this assumes you've already installed apache):

sudo chown -R pi /var/www

Enter this command to install vsftpd

sudo apt-get install vsftpd

Now edit the configuration file for vsftpd to make it more secure and user friendly.

sudo nano /etc/vsftpd.conf

Check "anonymous\_enable=NO" to disable Anonymous FTP)

Ensure "local\_enable=YES" and "write\_enable=YES" are un-commented (to allow you to make changes to the FTP server).

Ensure "local\_umask=022" is not commented out (this allows files uploaded as the "pi" user for instance to be served (apache runs as root)

Add "force\_dot\_files=YES" to the bottom to enable files such as .htaccess

Use CTRL+X to save and exit

Restart the new FTP server

sudo service vsftpd restart

**Module 6: Introduction to IoT Cloud Platform**

**Cloud4RPi**

**Connecting an Raspberry Pi 4 to Cloud4RPi**

Follow the instructions below to connect your device to the [Cloud4RPi](https://cloud4rpi.io) control panel.

## **Prerequisites**

It is advisable to update your system before installing.

sudo apt update && sudo apt upgrade -y

Install/update the required packages.

sudo apt install git python3 python3-pip -y

You can use alternative methods to [install pip](https://pip.pypa.io/en/stable/installing.html).

Note

The Cloud4RPi library is compatible with Python 3.2+ and Python 2.7.9+ versions.

### Enable interfaces

* Run sudo raspi-config
* Open a section for configuring additional interfaces (Advanced Options or Interfacing Options | Configure connections to peripherals depending on the version).
* Enable I2C, 1-wire and other necessary interfaces.
* Choose <Finish>.
* Reboot the device with the sudo reboot command.

## **Getting the Cloud4RPi Client Library**

Install the library using your preferred Python version. The following command installs and integrates Cloud4RPi with your OS's default Python interpreter (usually Python 3):

sudo pip3 install cloud4rpi

If you are using Python 2, use the following command:

sudo python2 -m pip install cloud4rpi

Note

For information on how to work with several versions of Python installed

If you get an error while using **pip**, try to update the **setuptools** and **pip** packages.

sudo pip3 install --upgrade setuptools pip

## **Hacking Together some Code**

We have prepared sample code for several platforms in the [cloud4rpi-raspberrypi-python](https://github.com/cloud4rpi/cloud4rpi-raspberrypi-python) repository to demonstrate sending data to the Cloud.

Get Cloud4RPi examples for your device:

git clone https://github.com/cloud4rpi/cloud4rpi-raspberrypi-python.git && cd cloud4rpi-raspberrypi-python

Before running a sample ([control.py](https://github.com/cloud4rpi/cloud4rpi-raspberrypi-python/blob/master/control.py)), remember to replace the \_\_YOUR\_DEVICE\_TOKEN\_\_ string with your real device token. Use a text editor (for instance, nano) to replace \_\_YOUR\_DEVICE\_TOKEN\_\_ with the token displayed at the top of the device page on [cloud4rpi.io](https://cloud4rpi.io/). If you have no token yet, open the [Devices](https://cloud4rpi.io/devices) page, create a device using the **New Device** button in the top right corner, and use its token.

## **Running**

Execute the script with a Python interpreter, for example:

sudo python3 control.py

Note

If you have installed Cloud4RPi to a non-default Python, use the version with the Cloud4RPi library.

If the script output looks right, open the [Devices](https://cloud4rpi.io/devices) page to see if the device status has changed.

## **Installing as a Service**

You can use our service templates to facilitate service installation. Pass the path to your Cloud4RPi-enabled Python script to the [service\_install.sh](https://github.com/cloud4rpi/cloud4rpi-raspberrypi-python/blob/master/service_install.sh) script as a parameter.

chmod +x service\_install.sh

sudo ./service\_install.sh your\_script.py

Note

You need to replace 'your\_script.py' with the actual path to your service script.

**Connecting an ESP32 to Cloud4RPi**

**Getting Libraries and Examples**

1. Open **Arduino** and select your board in the **Tools** | **Board** menu. Add [ESP32](https://github.com/espressif/arduino-esp32/blob/master/docs/arduino-ide/boards_manager.md) support if it is not added yet.
2. Install the Cloud4RPi library from the **Library Manager**. To do this, open the **Sketch** | **Include Library** | **Manage Libraries** menu, enter cloud4rpi into the search field and install the **cloud4rpi-esp-arduino** package.
3. Install the dependencies: **ArduinoJson** and **PubSubClient** libraries.
4. Configure the installed libraries:
   1. Open the %HOMEPATH%\Documents\Arduino\libraries\PubSubClient\src\PubSubClient.h (~/Documents/Arduino/libraries/PubSubClient/src/PubSubClient.h on Mac) file with any text editor (for instance, [VS Code](https://code.visualstudio.com)).
   2. Add the following define directives at the beginning:

#define MQTT\_MAX\_PACKET\_SIZE 1024

#define MQTT\_MAX\_TRANSFER\_SIZE 128

[Optional] Open the %HOMEPATH%\Documents\Arduino\libraries\cloud4rpi-esp-arduino\src\Cloud4RPi.h (~/Documents/Arduino/libraries/cloud4rpi-esp-arduino/src/Cloud4RPi.h on Mac) file and add the #define CLOUD4RPI\_DEBUG 1 line at the beginning to enable verbose output.

**Opening Sample Code**

1. Use the **File** | **Examples** | **cloud4rpi-esp-arduino** menu item to open the sample code. Restart Arduino IDE if this item does not appear.
2. Select the **ESP32 Dev Module** board in **Tools** menu.
3. If the LED\_BUILTIN constant is not defined in your board's library, change this constant to the LED's pin number (for example, 2).
4. Replace the \_\_SSID\_\_ and \_\_PASSWORD\_\_ strings with your Wi-Fi network data.

**Connecting to Your Cloud4RPi Account**

1. [Log in to your Cloud4RPi account](https://cloud4rpi.io/signin) or [create a new one](https://cloud4rpi.io/register).
2. Copy [your device](https://cloud4rpi.io/devices)'s **Device Token**. You can create a device on the [Devices](https://cloud4rpi.io/devices) page and copy its **Device Token** if you do not have one.
3. Replace the \_\_YOUR\_DEVICE\_TOKEN\_\_ string with your device token.

## **Running**

Hit the Build button and wait until the compilation is completed.

Hit the Upload button.

If the upload fails, try to start the board in the flashing mode (connect the power supply with the GPIO0 pin connected to GND) and try uploading again.

Once flashing is complete, open Serial Monitor to monitor the device's status.

Notice that the device on the [Devices page](https://cloud4rpi.io/devices) went online and started sending data.

Go to the [Control Panels](https://cloud4rpi.io/control-panels/) page and add a new control panel.

Add a new **Switch** widget and bind it to the LED On variable.

Add a new **Text** widget and bind it to the State variable. Configure different colors for the **"IDLE"**, **"RING"** and **"BOOM!"** strings.

Add a new **Slider** widget and bind it to the DesiredTemp variable, set its minimum value to 10 and maximum value to 30.

Add a new **Gauge** widget and bind it to the DesiredTemp variable, set its minimum value to 10 and maximum value to 30.

## **Setting up your Raspberry Pi**

VNC Server is included with Raspbian but you still have to enable it.

**\*If you have an earlier version of Raspbian, or a different Linux distribution, you can**[**install**](https://www.realvnc.com/connect/download/vnc/raspberrypi/)**VNC Server yourself.**

First, run the following commands to make sure you have the latest version:

sudo apt-get update   
sudo apt-get install realvnc-vnc-server

If you’re already using an older version of VNC Server, restart it. If not, and you’re already booted into the graphical desktop, select **Menu > Preferences > Raspberry Pi Configuration > Interfaces** and make sure **VNC** is set to **Enabled**.

Alternatively, run the command sudo raspi-config, navigate to **Interfacing Options > VNC** and select **Yes**.

From now on, VNC Server will start automatically every time you boot your Raspberry Pi.

**\*By default, VNC Server remotes the graphical desktop running on your Raspberry Pi. However, if your Pi is headless (not plugged into a monitor) or not running a graphical desktop, VNC Server can still give you graphical remote access using a virtual desktop.**

Note you can also install VNC Viewer on your Raspberry Pi, in case you want to control a remote computer (or another Raspberry Pi!). To do this, use the Recommended Software program, or run the command sudo apt-get install realvnc-vnc-viewer.

### Getting connected to your Raspberry Pi

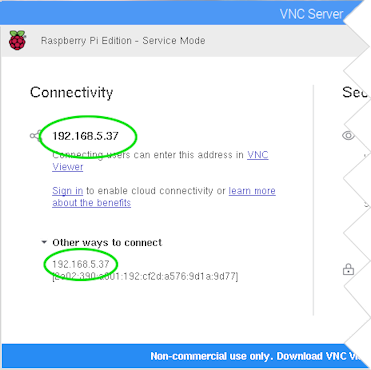
There are two ways to connect; you can use either or both. Please make sure you’ve downloaded [VNC Viewer app](https://www.realvnc.com/connect/download/viewer/) to computers or devices you want to control from.

### Establishing a direct connection

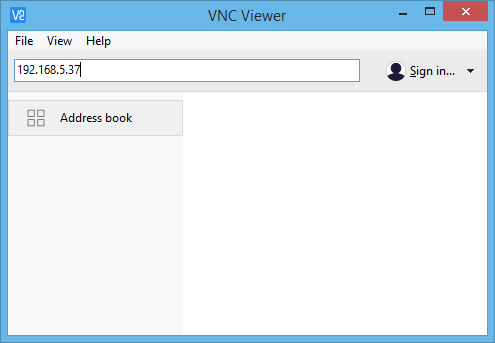
Direct connections are quick and simple providing you’re joined to the same private local network as your Raspberry Pi (for example, a wired or Wi-Fi network at home, school or in the office).

**\*If you’re connecting over the Internet, it’s much safer and more convenient to establish a cloud connection.**

1. On your Raspberry Pi, discover its private IP address by double-clicking the VNC Server icon on the taskbar and examining the status dialog:



1. On the device you will use to take control, run VNC Viewer and enter the IP address in the search bar:



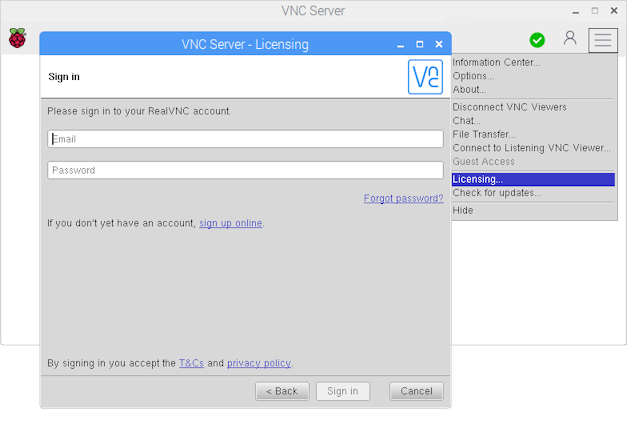
### Establishing a cloud connection

Cloud connections are convenient and encrypted end-to-end, and highly recommended for connections over the Internet. There’s no firewall or router reconfiguration, and you don’t need to know the IP address of your Raspberry Pi, or provide a static one.

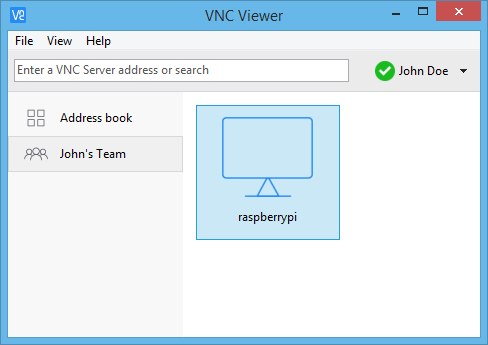
You’ll need a RealVNC account; it’s completely free to set up and only takes a few seconds. We’ll give you a special version of our Home subscription that enables both cloud and direct connectivity, and also in-session features such as system authentication, file transfer, printing and chat.

**\*You can apply your Home subscription to five Raspberry Pis and/or desktop computers in total. Please note you revert to the**[**standard feature set**](https://www.realvnc.com/pricing#comparison-table)**for Windows, Mac and Linux desktop computers.**

1. Sign up for a RealVNC account by entering your email address in [the box on this page](https://www.realvnc.com/raspberrypi/#sign-up), and following the instructions.
2. On your Raspberry Pi, select **Licensing** from the VNC Server status menu, choose **Sign in to your RealVNC account**, and enter your new account email and password:



1. On the device you will use to take control, run VNC Viewer and sign in using the same account credentials.
2. In VNC Viewer, a connection to your Raspberry Pi automatically appears under the name of your team. Simply tap or double-click to connect:



### Authenticating to VNC Server

To complete either a direct or cloud connection you must authenticate to VNC Server. Enter the user name and password you normally use to log on to your user account on the Raspberry Pi.

By default, these credentials are pi and raspberry, but hopefully you’ll have changed them to something more secure by now!

### Running directly rendered apps such as Minecraft remotely

VNC Server can remote the screen of Raspberry Pi apps that use a directly rendered overlay, such as Minecraft, the text console, the Pi camera module, and more.

To turn this feature on, open the VNC Server dialog, navigate to **Menu > Options > Troubleshooting**, and select **Enable direct capture mode**. On the device you will use to take control, run VNC Viewer and connect (if already connected, you’ll need to reconnect).

If you’ve turned direct capture mode on and mouse movements seem erratic when using Minecraft remotely, try pressing F8 to open the VNC Viewer shortcut menu and selecting **Relative Pointer Motion**.

If performance seems impaired, try:

1. On your Raspberry Pi, run sudo raspi-config, navigate to **Advanced options > Memory Split**, and ensure your GPU has at least 128MB.
2. Reduce your Raspberry Pi’s screen resolution.

## **Transferring files to and from your Raspberry Pi**

You can transfer files to and from your Raspberry Pi providing you’re connecting from VNC Viewer running on a Windows, Mac or Linux desktop computer.

* To transfer files to your Raspberry Pi, click the VNC Viewer VNC_Viewer_Toolbar_File_Transfer_Small.png toolbar button and follow the instructions.
* To transfer files from your Raspberry Pi, use VNC Viewer to open the VNC Server dialog remotely, select **Menu > File transfer**, and follow the instructions.

## **Printing to a local printer**

It can be really useful to print to a printer attached to your Windows, Mac or Linux computer if no printer is set up for your Raspberry Pi. To do this, first run the following command on your Raspberry Pi to install cups (the Common Unix Printing System):

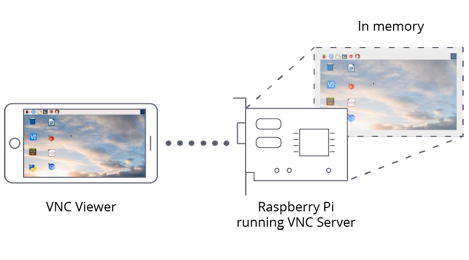
sudo apt-get install cups

Then, connect to your Pi using VNC Viewer and perform whatever the standard operation is for printing the file you want to print (for example, select a text editor’s **File > Print** menu option). VNC Server directs the output to VNC Viewer, and spools it to your local printer.

## **Creating and remoting a virtual desktop**

If your Raspberry Pi is headless (that is, not plugged into a monitor) or embedded in a robot, it’s unlikely to be running a graphical desktop.

VNC Server can run in Virtual Mode to create a resource-efficient virtual desktop on demand, giving you graphical remote access even when there is no actual desktop to remote. This virtual desktop exists only in your Raspberry Pi’s memory:



To do this:

1. On your Raspberry Pi, run the command vncserver. Make a note of the IP address/display number printed to the console, for example 192.167.5.149:1.
2. On the device you will use to take control, enter this information in VNC Viewer.

### Stopping a virtual desktop

A virtual desktop persists until you explicitly destroy it. Run the following command when you are sure it is no longer needed:

vncserver -kill :<display-number>

Note this command will terminate any current connections without warning to those users.

### Operating VNC Server at the command line

You can operate VNC Server exclusively at the command line or via SSH if you prefer.

Common commands for Raspbian Jessie (which is based on Debian 8, and uses systemd) are:

* To start VNC Server now: sudo systemctl start vncserver-x11-serviced.service
* To start VNC Server at next boot, and every subsequent boot: sudo systemctl enable vncserver-x11-serviced.service
* To stop VNC Server: sudo systemctl stop vncserver-x11-serviced.service
* To prevent VNC Server starting at boot: sudo systemctl disable vncserver-x11-serviced.service

For equivalent commands for Raspbian Wheezy (which is based on Debian 7, and uses initd), see [this page](https://help.realvnc.com/hc/en-us/articles/360002253218).

## **Troubleshooting VNC Server**

### Changing the Raspberry Pi’s screen resolution

You may want to do this if:

* Performance is impaired. A smaller screen resolution gives a more responsive experience.
* Your Raspberry Pi is headless (that is, not plugged into a monitor) and the default initial screen resolution is too small.

To change the resolution, run the command sudo raspi-config, navigate to **Advanced Options > Resolution**, and choose an option.

If this menu is not available, or you want more control, specify settings in the /boot/config.txt file:

| **Setting** | **Value** | **Explanation** |
| --- | --- | --- |
| hdmi\_force\_hotplug | 1 | Tells your Pi an HDMI display is attached. |
| hdmi\_ignore\_edid | 0xa5000080 | Ignores EDID/display data. |
| hdmi\_group | 2 | Defines the HDMI output group. |
| hdmi\_mode | 16 | Forces (for example) 1024x768 at 60Hz. |

See the [Raspberry Pi documentation](https://www.raspberrypi.org/documentation/configuration/config-txt/README.md) for more hdmi\_mode options, and information on /boot/config.txt in general. You will need to reboot your Raspberry Pi for any changes to take effect.

Note that settings you specify in this file override monitors you subsequently plug in (unless you revert hdmi\_force\_hotplug), so pick a ‘headless’ resolution compatible with your regular monitor.

### Specifying a screen resolution for a virtual desktop

If you run VNC Server in Virtual Mode to create a virtual desktop, you can specify the screen resolution (geometry) at start up, for example:

vncserver -randr=800x600

You can even specify multiple screen resolutions and cycle between them.

### Optimizing for Raspberry Pi Zero and Pi 1

If performance is impaired for direct connections to a Raspberry Pi Zero or Pi 1, try turning off encryption if you are sure your private local network is secure. This reduces CPU usage.

**\*You cannot turn off encryption for cloud connections.**

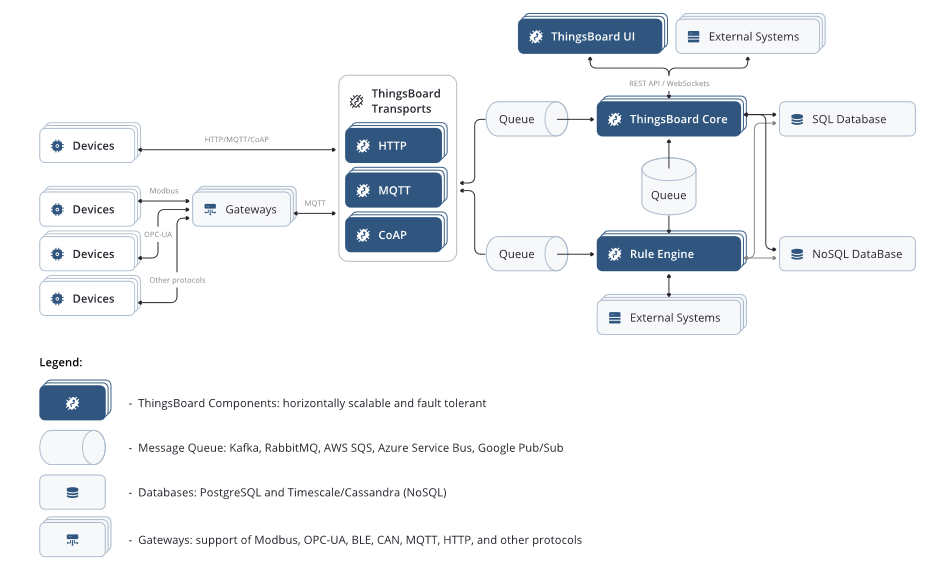
1. On your Raspberry Pi, open the VNC Server dialog and select **Menu > Options > Expert**.
2. Change the Encryption parameter to AlwaysOff.
3. Restart any existing connections.

If performance is still impaired, try reducing your Raspberry Pi’s screen resolution.

## **ThingsBoard services**

ThingsBoard is designed to be:

* **scalable**: horizontally scalable platform, build using leading open-source technologies.
* **fault-tolerant**: no single-point-of-failure, every node in the cluster is identical.
* **robust and efficient**: single server node can handle tens or even hundreds thousands of devices depending on use-case. ThingsBoard cluster can handle millions of devices.
* **durable**: never lose your data. ThingsBoard supports various queue implementations to provide extremely high message durability.
* **customizable**: adding new functionality is easy with customizable widgets and rule engine nodes.



**ThingsBoard Transports**

ThingsBoard provides [MQTT](https://thingsboard.io/docs/reference/mqtt-api/), [HTTP](https://thingsboard.io/docs/reference/http-api/) and [CoAP](https://thingsboard.io/docs/reference/coap-api/) based APIs that are available for your device applications/firmware. Each of the protocol APIs are provided by a separate server component and is part of ThingsBoard “Transport Layer”. MQTT Transport also provides [Gateway APIs](https://thingsboard.io/docs/reference/gateway-mqtt-api/) to be used by gateways that represent multiple connected devices and/or sensors.

Once the Transport receives the message from device, it is parsed and pushed to durable [Message Queue](https://thingsboard.io/docs/reference/#message-queues-are-awesome). The message delivery is acknowledged to device only after corresponding message is acknowledged by the message queue.

**ThingsBoard Core**

ThingsBoard Core is responsible for handling [REST API](https://thingsboard.io/docs/reference/rest-api/) calls and WebSocket [subscriptions](https://thingsboard.io/docs/user-guide/telemetry/#websocket-api). It is also responsible for storing up to date information about active device sessions and monitoring device [connectivity state](https://thingsboard.io/docs/user-guide/device-connectivity-status/). ThingsBoard Core uses Actor System under the hood to implement actors for main entities: tenants and devices. Platform nodes can join the cluster, where each node is responsible for certain partitions of the incoming messages.

**ThingsBoard Rule Engine**

ThingsBoard Rule Engine is the heart of the system and is responsible for processing incoming [messages](https://thingsboard.io/docs/user-guide/rule-engine-2-0/overview/#rule-engine-message). Rule Engine uses Actor System under the hood to implement actors for main entities: rule chains and rule nodes. Rule Engine nodes can join the cluster, where each node is responsible for certain partitions of the incoming messages.

Rule Engine subscribes to incoming data feed from queue(s) and acknowledge the message only once it is processed. There are multiple strategies available that control the order or message processing and the criteria of message acknowledgement. See [submit strategies](https://thingsboard.io/docs/user-guide/rule-engine-2-0/overview/#queue-submit-strategy) and [processing strategies](https://thingsboard.io/docs/user-guide/rule-engine-2-0/overview/#queue-processing-strategy) for more details.

ThingsBoard Rule Engine may operate in two modes: shared and isolated. In shared mode, rule engine process messages that belong to multiple tenants. In isolated mode Rule Engine may be configured to process messages for specific tenant only.

**ThingsBoard Web UI**

ThingsBoard provides a lightweight component written using Express.js framework to host static web ui content. Those components are completely stateless and no much configuration available. The static web UI contains application bundle. Once it is loaded, the application starts using the REST API and WebSockets API provided by ThingsBoard Core.

## **Message Queues are awesome!**

ThingsBoard supports multiple message queue implementations: Kafka, RabbitMQ, AWS SQS, Azure Service Bus and Google Pub/Sub. We plan to extend this list in the future. Using durable and scalable queues allow ThingsBoard to implement back-pressure and load balancing. Back-pressure is extremely important in case of peak loads.  
We provide “abstraction layer” over specific queue implementations and maintain two main concepts: topic and topic partition. One topic may have configurable number of partitions. Since most of the queue implementations does not support partitions, we use topic + “.” + partition pattern.

ThingsBoard message Producers determines which partition to use based on the hash of entity id. Thus, all messages for the same entity are always pushed to the same partition. ThingsBoard message Consumers coordinate using Zookeeper and use consistent-hash algorithm to determine list of partitions that each Consumer should subscribe to. While running in microservices mode, each service also has the dedicated “Notifications” topic based on the unique service id that has only one partition.

ThingsBoard uses following topics:

* **tb\_transport.api.requests**: to send generic API calls to check device credentials from Transport to ThingsBoard Core.
* **tb\_transport.api.responses**: to receive device credentials verification results from ThingsBoard Core to Transport.
* **tb\_core**: to push messages from Transport or Rule Engine to ThingsBoard Core. Messages include session lifecycle events, attribute and RPC subscriptions, etc.
* **tb\_rule\_engine**: to push messages from Transport or ThingsBoard Core to Rule Engine. Messages include incoming telemetry, device states, entity lifecycle events, etc.

**Note:** All topic properties including names and number of partitions are [configurable](https://thingsboard.io/docs/user-guide/install/config/) via thingsboard.yml or environment variables. We plan to enable configuration via UI in ThingsBoard 2.6 and/or 3.1.

**Note:** Starting version 2.5 we have switched from using [gRPC](https://grpc.io/) to [Message Queues](https://thingsboard.io/docs/reference/#message-queues-are-awesome) for all communication between ThingsBoard components. The main idea was to sacrifice small performance/latency penalties in favor of persistent and reliable message delivery and automatic load balancing.

## **On-premise vs cloud deployments**

ThingsBoard supports both on-premise and cloud deployments. With more then 5000 ThingsBoard servers running all over the world, ThingsBoard is running in production on AWS, Azure, GCE and private data centers. It is possible to launch ThingsBoard in the private network with no internet access at all.

## **Standalone vs cluster mode**

Platform is designed to be horizontally scalable and supports automatic discovery of new ThingsBoard servers (nodes). All ThingsBoard nodes inside cluster are identical and are sharing the load. Since all nodes are identical there is no “master” or “coordinator” processes and there is no single point of failure. The load balancer of your choice may forward request from devices, applications and users to all ThingsBoard nodes.

## **Monolithic vs microservices architecture**

Starting ThingsBoard v2.2, it is possible to run the platform as a monolithic application or as a set of microservices. Supporting both options requires some additional programming efforts, however, is critical due to back-ward compatibility with variety of existing installations.

Approximately 80% of the platform installations are still using monolithic mode due to minimum support efforts, knowledge and hardware resources to do the setup and low maintenance efforts.

However, if you do need high availability or would like to scale to millions of devices, then micro services is a way to go. There are also some challenges that are solved with micro services architecture and applicable for more complex deployments and usage scenarios. For example, running a multi-tenant deployments where one need more granular isolation to protect from:

* unpredictable load spikes;
* unpredictable rule chain misconfiguration;
* Single devices opening 1000s of concurrent connections due to firmware bugs; and many other cases.

Please follow the links listed below to learn more and choose the right architecture and deployment option:

* [**monolithic**](https://thingsboard.io/docs/reference/monolithic): Learn more about deployment, configuring and running ThingsBoard platform in a monolythic mode.
* [**microservices**](https://thingsboard.io/docs/reference/msa): Learn more about deployment, configuring and running ThingsBoard platform in a microservices mode.

## **SQL vs NoSQL vs Hybrid database approach**

ThingsBoard uses database to store [entities](https://thingsboard.io/docs/user-guide/entities-and-relations/) (devices, assets, customers, dashboards, etc) and [telemetry](https://thingsboard.io/docs/user-guide/telemetry/) data (attributes, timeseries sensor readings, statistics, events). Platform supports three database options at the moment:

* **SQL** - Stores all entities and telemetry in SQL database. ThingsBoard authors recommend to use PostgreSQL and this is the main SQL database that ThingsBoard supports. It is possible to use HSQLDB for local development purposes. **We do not recommend to use HSQLDB** for anything except running tests and launching dev instance that has minimum possible load.
* **NoSQL (Deprecated)** - Stores all entities and telemetry in NoSQL database. ThingsBoard authors recommend to use Cassandra and this is the only NoSQL database that ThingsBoard supports at the moment. Please note that this option is deprecated in favor of Hybrid approach due to many limitations of NoSQL for transactions and “joins” that are required to enable advanced search over IoT entities.
* **Hybrid (PostgreSQL + Cassandra)** - Stores all entities in PostgreSQL database and timeseries data in Cassandra database.
* **Hybrid (PostgreSQL + TimescaleDB)** - Stores all entities in PostgreSQL database and timeseries data in Timescale database.

It is possible to configure this options using **thingsboard.yml** file.

database:

ts\_max\_intervals: "${DATABASE\_TS\_MAX\_INTERVALS:700}" # Max number of DB queries generated by single API call to fetch telemetry records

entities:

type: "${DATABASE\_ENTITIES\_TYPE:sql}" # cassandra OR sql

ts:

type: "${DATABASE\_TS\_TYPE:sql}" # cassandra, sql, or timescale (for hybrid mode, DATABASE\_TS\_TYPE value should be cassandra, or timescale)

# note: timescale works only with postgreSQL database for DATABASE\_ENTITIES\_TYPE.

**Module 7: Application Development with Raspberry Pi 4**

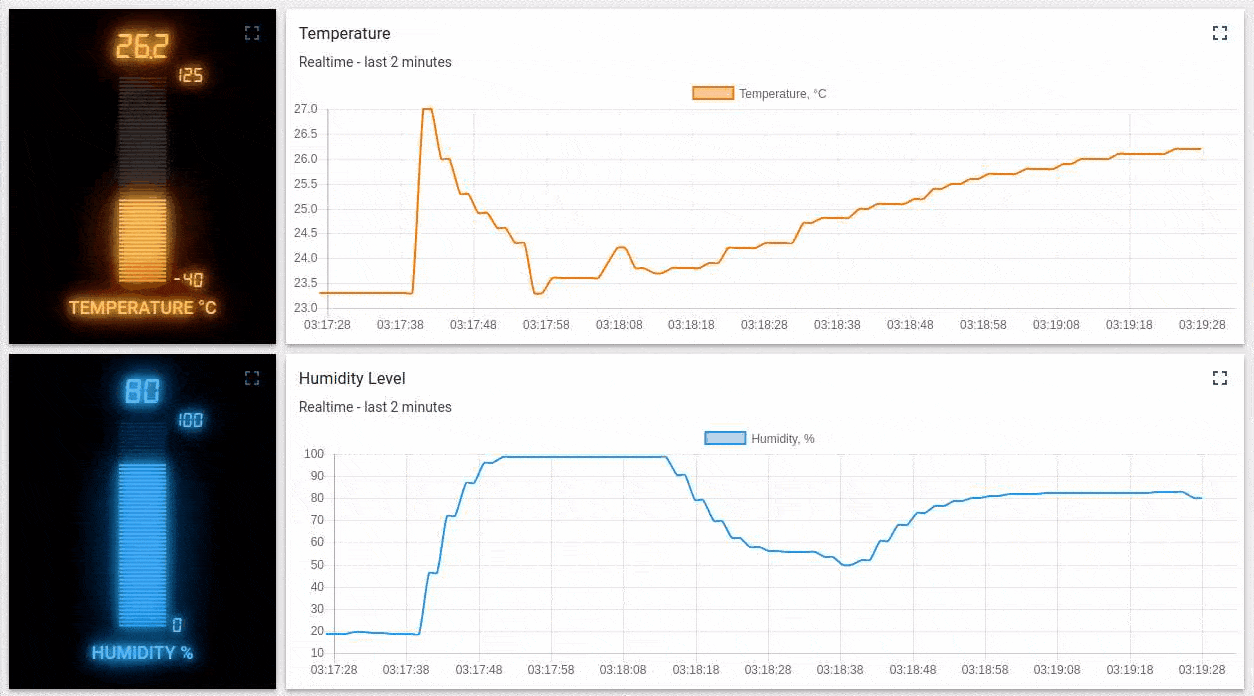
# Connecting Raspberry Pi 4 to Cloud Platforms

## **Introduction**

ThingsBoard is an open-source server-side platform that allows you to monitor and control IoT devices. It is free for both personal and commercial usage and you can deploy it anywhere.

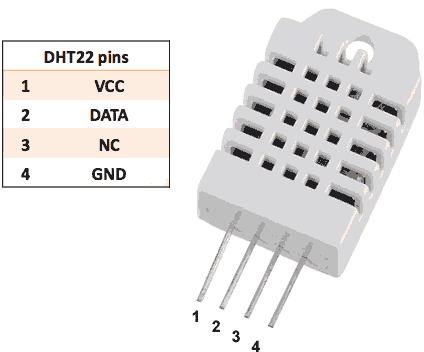
This sample application performs collection of temperature and humidity values produced by [DHT22 sensor](https://www.adafruit.com/product/385) and further visualization on the real-time web dashboard. Collected data is pushed via MQTT to ThingsBoard server for storage and visualization. The purpose of this application is to demonstrate ThingsBoard [data collection API](https://thingsboard.io/docs/user-guide/telemetry/) and [visualization capabilities](https://thingsboard.io/docs/user-guide/visualization/).

The DHT22 sensor is connected to [Raspberry Pi](https://en.wikipedia.org/wiki/Raspberry_Pi). Raspberry Pi offers a complete and self-contained Wi-Fi networking solution. Raspberry Pi push data to ThingsBoard server via MQTT protocol by using [paho mqtt](https://eclipse.org/paho/clients/python/) python library. Data is visualized using built-in customizable dashboard. The application that is running on Raspberry Pi is written in Python which is quite simple and easy to understand.

Once you complete this sample/tutorial, you will see your sensor data on the following dashboard.

## **List of hardware and pinouts**

* Raspberry Pi 4
* [DHT22 sensor](https://www.aliexpress.com/item/1pcs-DHT22-digital-temperature-and-humidity-sensor-Temperature-and-humidity-module-AM2302-replace-SHT11-SHT15/32316036161.html?spm=2114.03010208.3.49.aZvfaG&ws_ab_test=searchweb0_0,searchweb201602_2_10065_10068_10084_10083_10080_10082_10081_10060_10061_10062_10056_10055_10054_10059_10099_10078_10079_10093_426_10073_10103_10102_10096_10052_10050_10051,searchweb201603_6&btsid=28d9ee9a-283a-4e97-af7b-a7e530490916)



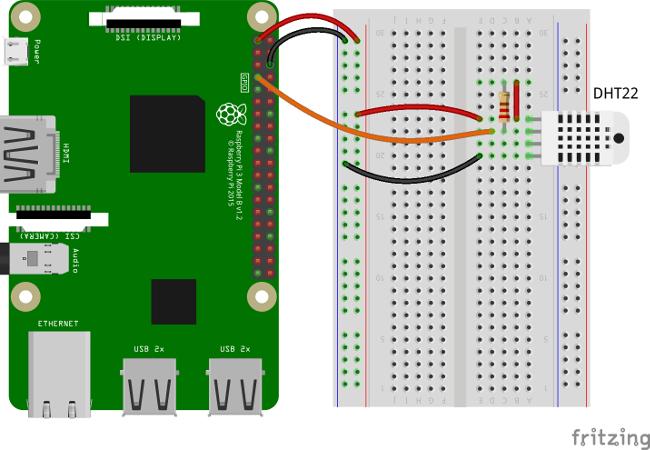
* Resistor (between 4.7K and 10K)
* Breadboard
* 2 female-to-female jumper wires
* 10 female-to-male jumper wires
* 3 male-to-male jumper wire

## **Wiring schemes**

| **DHT-22 Pin** | **Raspberry Pi Pin** |
| --- | --- |
| DHT-22 Data | Raspberry Pi GPIO 4 |
| DHT-22 VCC | Raspberry Pi 3.3V |
| DHT-22 GND (-) | Raspberry Pi GND |

Finally, place a resistor (between 4.7K and 10K) between pin number 1 and 2 of the DHT sensor.

The following picture summarizes the connections for this project:



## **ThingsBoard configuration**

**Note** ThingsBoard configuration steps are necessary only in case of **local ThingsBoard installation**.

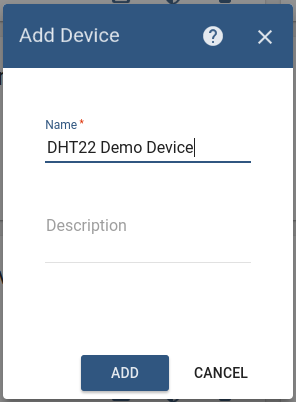
### Provision your device

This step contains instructions that are necessary to connect your device to ThingsBoard.

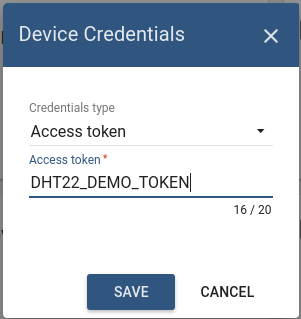
Open ThingsBoard Web UI (http://localhost:8080) in browser and login as tenant administrator

* login: tenant@thingsboard.org
* password: tenant

Goto “Devices” section. Click “+” button and create a device with the name “DHT22 Demo Device”.



Once device created, open its details and click “Manage credentials”. Copy auto-generated access token from the “Access token” field. Please save this device token. It will be referred to later as **$ACCESS\_TOKEN**.



Click “Copy Device ID” in device details to copy your device id to the clipboard. Paste your device id to some place, this value will be used in further steps.

### Provision your dashboard

### Programming the Raspberry Pi

### MQTT library installation

The following command will install MQTT Python library:

sudo pip install paho-mqtt

### Adafruit DHT library installation

Install python-dev package:

sudo apt-get install python-dev

Downloading and install the Adafruit DHT library:

git clone https://github.com/adafruit/Adafruit\_Python\_DHT.git

cd Adafruit\_Python\_DHT

sudo python setup.py install

### Application source code

Our application consists of a single python script that is well documented. You will need to modify **THINGSBOARD\_HOST** constant to match your ThingsBoard server installation IP address or hostname.

The value of **ACCESS\_TOKEN** constant corresponds to sample DHT22 demo device.

* [mqtt-dht22.py](https://thingsboard.io/docs/samples/raspberry/temperature/#mqtt-dht22)

| [**resources/mqtt-dht22.py**](https://raw.githubusercontent.com/thingsboard/thingsboard.github.io/master/docs/samples/raspberry/resources/mqtt-dht22.py) |
| --- |
| import os  import time  import sys  import Adafruit\_DHT as dht  import paho.mqtt.client as mqtt  import json  THINGSBOARD\_HOST = 'demo.thingsboard.io'  ACCESS\_TOKEN = 'DHT22\_DEMO\_TOKEN'  # Data capture and upload interval in seconds. Less interval will eventually hang the DHT22.  INTERVAL=2  sensor\_data = {'temperature': 0, 'humidity': 0}  next\_reading = time.time()  client = mqtt.Client()  # Set access token  client.username\_pw\_set(ACCESS\_TOKEN)  # Connect to ThingsBoard using default MQTT port and 60 seconds keepalive interval  client.connect(THINGSBOARD\_HOST, 1883, 60)  client.loop\_start()  try:  while True:  humidity,temperature = dht.read\_retry(dht.DHT22, 4)  humidity = round(humidity, 2)  temperature = round(temperature, 2)  print(u"Temperature: {:g}\u00b0C, Humidity: {:g}%".format(temperature, humidity))  sensor\_data['temperature'] = temperature  sensor\_data['humidity'] = humidity  # Sending humidity and temperature data to ThingsBoard  client.publish('v1/devices/me/telemetry', json.dumps(sensor\_data), 1)  next\_reading += INTERVAL  sleep\_time = next\_reading-time.time()  if sleep\_time > 0:  time.sleep(sleep\_time)  except KeyboardInterrupt:  pass  client.loop\_stop()  client.disconnect() |

### Running the application

This simple command will launch the application:

python mqtt-dht22.py

## Data visualization

Finally, open ThingsBoard Web UI. You can access this dashboard by logging in as a tenant administrator.

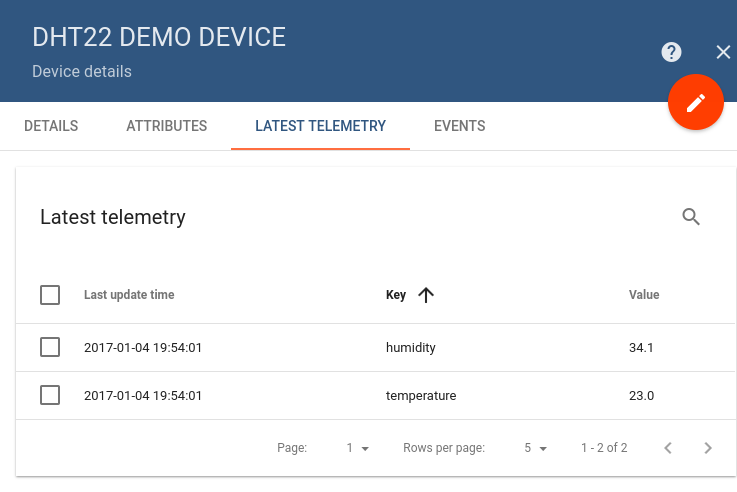
In case of local installation:

* login: tenant@thingsboard.org
* password: tenant

In case of live-demo server:

* login: your live-demo username (email)
* password: your live-demo password

Go to **“Devices”** section and locate **“DHT22 Demo Device”**, open device details and switch to **“Latest telemetry”** tab. If all is configured correctly you should be able to see latest values of “temperature” and “humidity” in the table.



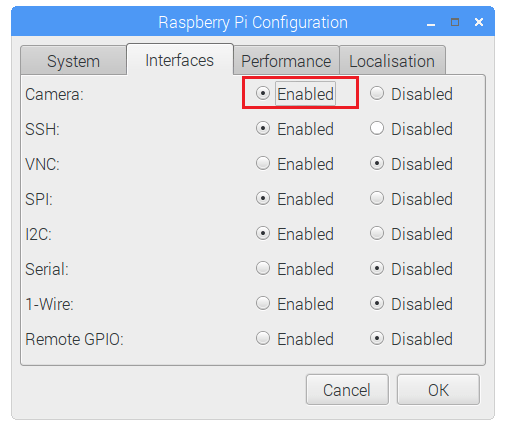
After, open **“Dashboards”** section then locate and open **“DHT22: Temperature & Humidity Demo Dashboard”**. As a result you will see two digital gauges and two time-series charts displaying temperature and humidity level (similar to dashboard image in the introduction).

# 

# Raspberry Pi 4 as a Video Streaming Server (LAN / Remote Network)

**Enable the Raspberry Pi Camera Module**

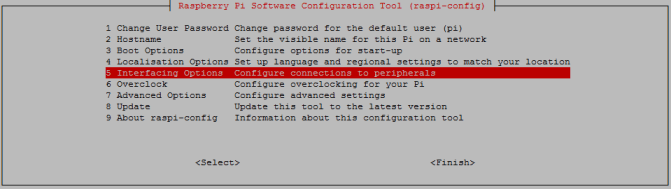
If you’re using the [Raspberry Pi Camera Module](https://makeradvisor.com/tools/raspberry-pi-camera-v2-module/), you need to enable the camera software in your Raspberry Pi in order to use it. In the Desktop environment, go to the **Raspberry Pi Configuration** window under the **Preferences** menu, open the **Interfaces** tab and enable the **Camera** as shown in figure below.



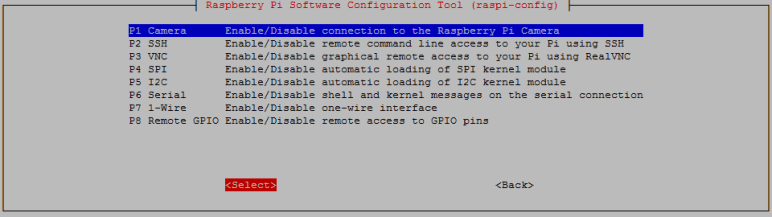
Or, in the **Terminal** window, type the following command:

pi@raspberry:~ $ **sudo raspi-config**

You should see the Raspberry Pi software configuration tool. Select the **Interfacing Options**:



Enable the camera and reboot your Pi:

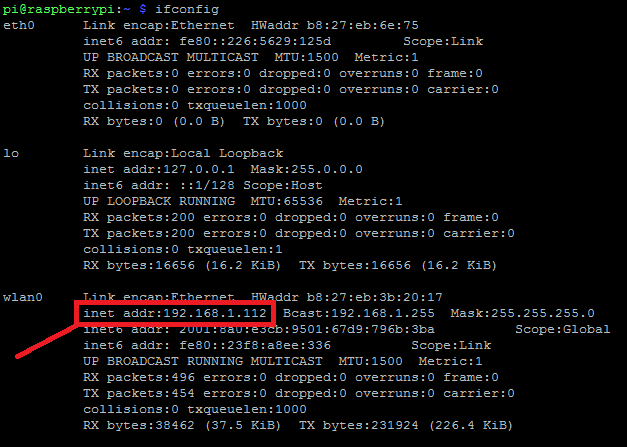


**Find the Raspberry Pi IP address**

To access your video streaming web server, you need to know your Raspberry Pi IP address. For that, use the following command:

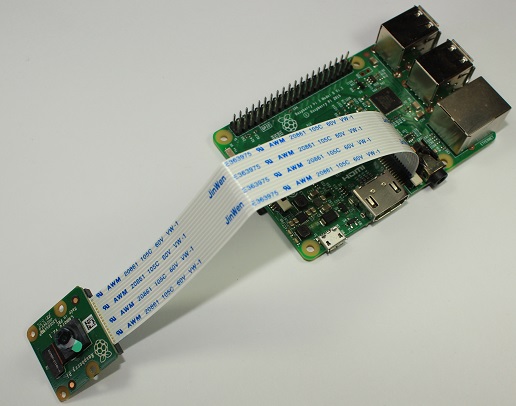
pi@raspberry:~ $ **ifconfig**

You’ll be given a bunch of information, including your Raspberry Pi IP address. In my case, the RPi IP address is **192.168.1.112**.



**Connect the camera**

Connecting the Raspberry Pi Camera Module is easy. With the Pi shutdown, connect the camera to the Pi CSI port as shown in the following figure. Make sure the camera is connected in the right orientation with the ribbon blue letters facing up as shown in the next figure.



**Writing the script**

The script for video streaming is shown below. You can find this script at the official PiCamera package [documentation](https://picamera.readthedocs.io/en/latest/recipes2.html).

Create a new file called **rpi\_camera\_surveillance\_system.py**:

pi@raspberrypi:~ $ **nano rpi\_camera\_surveillance\_system.py**

Copy the following code to your newly created file:

# Web streaming example

# Source code from the official PiCamera package

# http://picamera.readthedocs.io/en/latest/recipes2.html#web-streaming

import io

import picamera

import logging

import socketserver

from threading import Condition

from http import server

PAGE="""\

<html>

<head>

<title>Raspberry Pi - Surveillance Camera</title>

</head>

<body>

<center><h1>Raspberry Pi - Surveillance Camera</h1></center>

<center><img src="stream.mjpg" width="640" height="480"></center>

</body>

</html>

"""

class StreamingOutput(object):

def \_\_init\_\_(self):

self.frame = None

self.buffer = io.BytesIO()

self.condition = Condition()

def write(self, buf):

if buf.startswith(b'\xff\xd8'):

# New frame, copy the existing buffer's content and notify all

# clients it's available

self.buffer.truncate()

with self.condition:

self.frame = self.buffer.getvalue()

self.condition.notify\_all()

self.buffer.seek(0)

return self.buffer.write(buf)

class StreamingHandler(server.BaseHTTPRequestHandler):

def do\_GET(self):

if self.path == '/':

self.send\_response(301)

self.send\_header('Location', '/index.html')

self.end\_headers()

elif self.path == '/index.html':

content = PAGE.encode('utf-8')

self.send\_response(200)

self.send\_header('Content-Type', 'text/html')

self.send\_header('Content-Length', len(content))

self.end\_headers()

self.wfile.write(content)

elif self.path == '/stream.mjpg':

self.send\_response(200)

self.send\_header('Age', 0)

self.send\_header('Cache-Control', 'no-cache, private')

self.send\_header('Pragma', 'no-cache')

self.send\_header('Content-Type', 'multipart/x-mixed-replace; boundary=FRAME')

self.end\_headers()

try:

while True:

with output.condition:

output.condition.wait()

frame = output.frame

self.wfile.write(b'--FRAME\r\n')

self.send\_header('Content-Type', 'image/jpeg')

self.send\_header('Content-Length', len(frame))

self.end\_headers()

self.wfile.write(frame)

self.wfile.write(b'\r\n')

except Exception as e:

logging.warning(

'Removed streaming client %s: %s',

self.client\_address, str(e))

else:

self.send\_error(404)

self.end\_headers()

class StreamingServer(socketserver.ThreadingMixIn, server.HTTPServer):

allow\_reuse\_address = True

daemon\_threads = True

with picamera.PiCamera(resolution='640x480', framerate=24) as camera:

output = StreamingOutput()

#Uncomment the next line to change your Pi's Camera rotation (in degrees)

#camera.rotation = 90

camera.start\_recording(output, format='mjpeg')

try:

address = ('', 8000)

server = StreamingServer(address, StreamingHandler)

server.serve\_forever()

finally:

camera.stop\_recording()

To save your file press Ctrl+X, type Y and Enter.

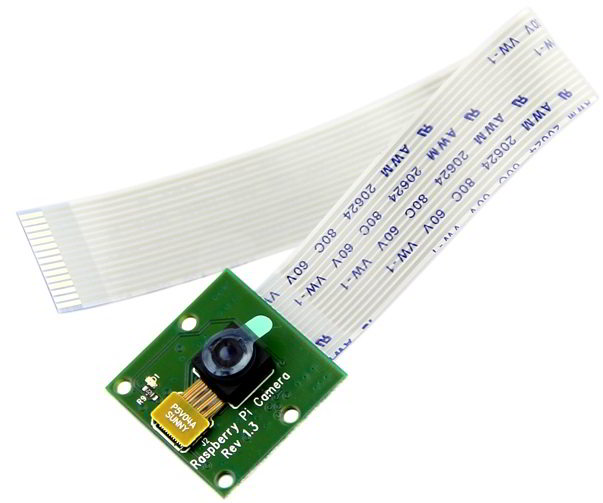
**Accessing the video streaming**

After writing the scrip, you can run it using Python 3. Run the next command:

pi@raspberrypi:~ $ **python3 rpi\_camera\_surveillance\_system.py**

Once the script is running, you can access your video streaming web server at: http://**<Your\_Pi\_IP\_Address>**:8000. Replace with your own Raspberry Pi IP address, in my case **http://192.168.1.112:8000**.

You can access the video streaming through any device that has a browser and is connected to the same network that your Pi.



**Pi Camera Module (v1.3)**

Pi Camera module is a camera which can be used to take pictures and high definition video.

Raspberry Pi Board has CSI (Camera Serial Interface) interface to which we can attach PiCamera module directly.

This Pi Camera module can attach to the Raspberry Pi’s CSI port using 15-pin ribbon cable.

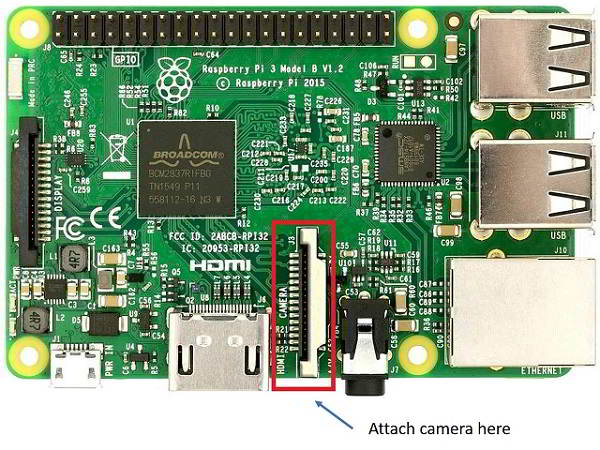
# ****Features of Pi Camera****

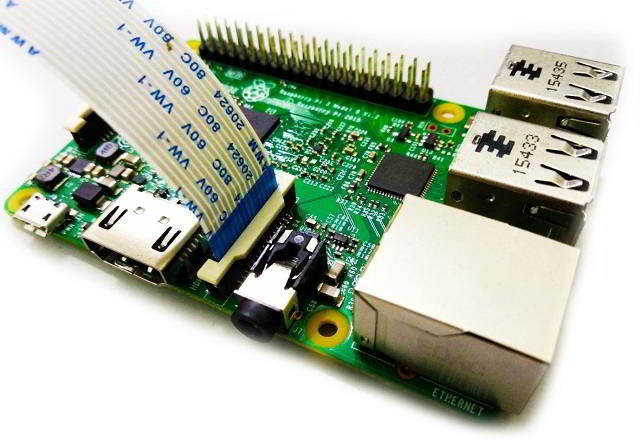
Here, we have used Pi camera v1.3. Its features are listed below,

* Resolution – 5 MP
* HD Video recording –     1080p @30fps, 720p @60fps, 960p @45fps and so on.
* It Can capture wide, still (motionless) images of resolution 2592x1944 pixels
* CSI Interface enabled.

# ****How to attach Pi Camera to Raspberry Pi?****

Connect Pi Camera to CSI interface of Raspberry Pi board as shown below,





Now, we can use Pi Camera for capturing images and videos using Raspberry Pi.

Before using Pi Camera, we need to enable camera for its working.

# ****How to Enable Camera functionality on Raspberry Pi****

For enabling camera in Raspberry Pi, open raspberry pi configuration using following command,

sudo raspi-config

then select **Interfacing options** in which select **camera** option to enable its functionality.

reboot Raspberry Pi.

Now we can access camera on Raspberry Pi.

Now we can capture images and videos using Pi Camera on Raspberry Pi.

# ****Example****

Capture images and save it to the specified directory.

We can capture images using Python. Here, we will write a Python program to capture images using Pi Camera on Raspberry Pi.

Here, we have used picamera package(library) which provides different classes for Raspberry Pi. Out of which we are mainly interested in PiCamera class which is for camera module.

**Python Program for Image Capture**

'''

capture images on Raspberry Pi using Pi Camera

http://www.electronicwings.com

'''

import picamera

from time import sleep

#create object for PiCamera class

camera = picamera.PiCamera()

#set resolution

camera.resolution = (1024, 768)

camera.brightness = 60

camera.start\_preview()

#add text on image

camera.annotate\_text = 'Hi Pi User'

sleep(5)

#store image

camera.capture('image1.jpeg')

camera.stop\_preview()

**Functions Used**

To use picamerapython based library we have to include it in our program as given below

import picamera

Thispicameralibrary hasPiCamera class for camera module. So, we have to create object for **PiCamera** class.

**PiCamera Class**

To use Pi Camera in Python on Raspberry Pi, we can use PiCamera class which has different APIs for camera functionality. We need to create object for PiCamera class.

**E.g.** Camera = picamera.PiCamera()

The above PiCamera class has different member variables and functions which we can access by simply inserting a dot (.) in between object name and member name.

**E.g.** Camera.resolution = (1080, 648)

**capture()**

It is used to capture images using Pi Camera.

**E.g.** Camera.capture(“/home/pi/image.jpeg”)

The capture() function has different parameters which we can pass for different operations like resize, format, use\_video\_port, etc.

**E.g.** Camera.capture(“/home/pi/image.jpeg”, resize=(720, 480))

**resolution= (width,height)**

It sets the resolution of camera at which image captures, video records and preview will display. The resolution can be specified as **(width, height)** tuple, as a string formatted **WIDTHxHEIGHT**, or as a string containing commonly recognised display resolution name e.g. “HD”, “VGA”, “1080p”, etc.

**E.g.**

                Camera.resolution = (720, 480)

Camera.resolution = “720 x 480”

Camera.resolution = “720p”

Camera.resolution = “HD”

**Annotate\_text = “Text”**

It is used to add text on image, video, etc.

**E.g.** Camera.annotate\_text = “Hi Pi User”

**start\_preview()**

It displays the preview overlay of default or specified resolution.

Example Camera.start\_preview()

**stop\_preview()**

It is used to close the preview overlay.

**E.g.** Camera.stop\_preview()

**Python Program for Video Recording**

'''

Record video on Raspberry Pi using pi Camera

http://www.electronicwings.com

'''

import picamera

from time import sleep

camera = picamera.PiCamera()

camera.resolution = (640, 480)

print()

#start recording using pi camera

camera.start\_recording("/home/pi/demo.h264")

#wait for video to record

camera.wait\_recording(20)

#stop recording

camera.stop\_recording()

camera.close()

print("video recording stopped")

**Functions used**

We have to create object for PiCamera class. Here, we have create object as **camera.**

**start\_recording()**

It is used to start video recordingand store it.

**E.g.** Camera.start\_recording(‘demo.h264’)

It records video named demo of h264 format.

**wait\_recording(timeout)**

Wait on video encoder for specified timeout seconds.

**E.g.** Camera.wait\_recording(60)

**stop\_recording()**

It is used to stop video recording.

E.g.Camera.stop\_recording()

**Play Recorded Video**

To open video, we can use omxplayer by using following command,

 omxplayer video\_name

**Ultrasonic**

For many (outdoor) projects a distance measurement is necessary or advantageous. These small modules are available starting at 1-2 bucks and can measure the distance up to 4-5 meters by ultrasound and are suprisingly accurate.

**Hardware**

* HC-SR04 Module
* Resistors: 330Ω and 470Ω
* Jumper wire

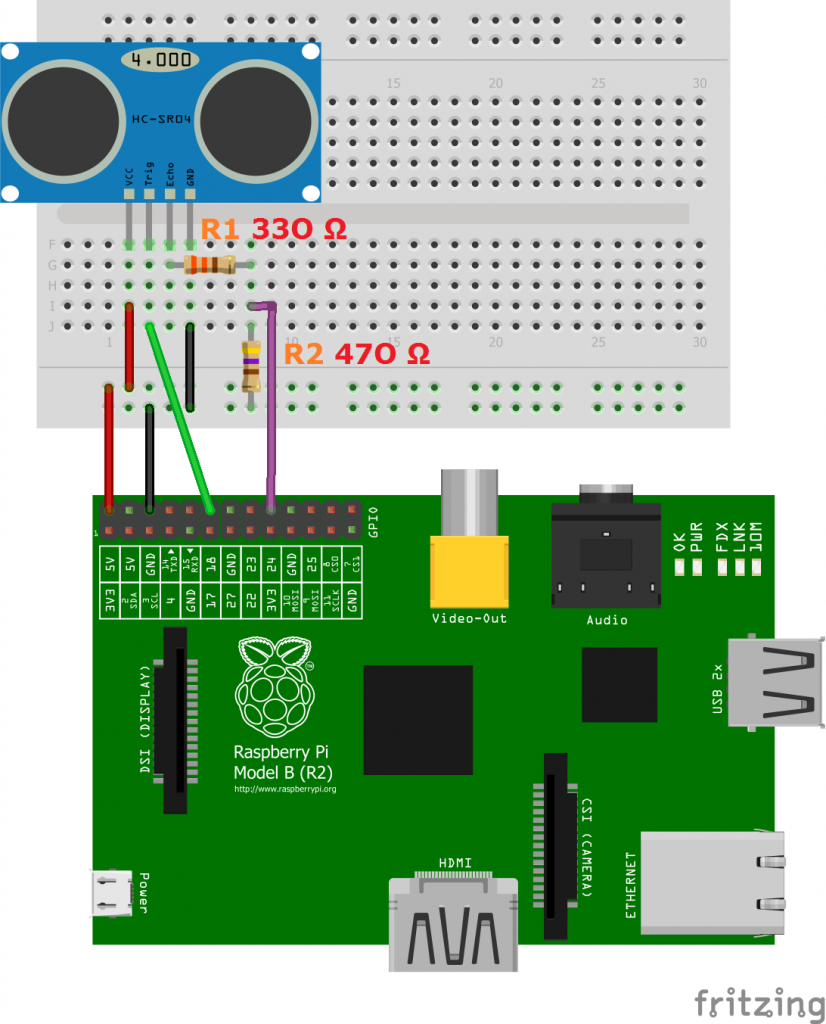
**Wiring**

There are four pins on the ultrasound module that are connected to the Raspberry:

* VCC to Pin 2 (VCC)
* GND to Pin 6 (GND)
* TRIG to Pin 12 (GPIO18)
* connect the 330Ω resistor to ECHO.  On its end you connect it to Pin 18 (GPIO24) and through a 470Ω resistor you connect it also to Pin6 (GND).

We do this because the GPIO pins only tolerate maximal 3.3V. The connection to GND is to have a obvious signal on GPIO24. If no pulse is sent, the signal is 0 (through the connection with GND), else it is 1. If there would be no connection to GND, the input would be undefined if no signal is sent (randomly 0 or 1), so ambiguous.

Here is the structure as a circuit diagram:

[](https://tutorials-raspberrypi.de/wp-content/uploads/2014/05/ultraschall_Steckplatine.png)

**Script for controlling**

First of all, the Python GPIO library should be installed

To use the module, we create a new script

sudo nano ultrasonic\_distance.py

with the following content:

#Libraries

import RPi.GPIO as GPIO

import time

#GPIO Mode (BOARD / BCM)

GPIO.setmode(GPIO.BCM)

#set GPIO Pins

GPIO\_TRIGGER = 18

GPIO\_ECHO = 24

#set GPIO direction (IN / OUT)

GPIO.setup(GPIO\_TRIGGER, GPIO.OUT)

GPIO.setup(GPIO\_ECHO, GPIO.IN)

def distance():

    # set Trigger to HIGH

    GPIO.output(GPIO\_TRIGGER, True)

    # set Trigger after 0.01ms to LOW

    time.sleep(0.00001)

    GPIO.output(GPIO\_TRIGGER, False)

    StartTime = time.time()

    StopTime = time.time()

    # save StartTime

    while GPIO.input(GPIO\_ECHO) == 0:

        StartTime = time.time()

    # save time of arrival

    while GPIO.input(GPIO\_ECHO) == 1:

        StopTime = time.time()

    # time difference between start and arrival

    TimeElapsed = StopTime - StartTime

    # multiply with the sonic speed (34300 cm/s)

    # and divide by 2, because there and back

    distance = (TimeElapsed \* 34300) / 2

    return distance

if \_\_name\_\_ == '\_\_main\_\_':

    try:

        while True:

            dist = distance()

            print ("Measured Distance = %.1f cm" % dist)

            time.sleep(1)

        # Reset by pressing CTRL + C

    except KeyboardInterrupt:

        print("Measurement stopped by User")

        GPIO.cleanup()

After that we run:

sudo python ultrasonic\_distance.py

So every second, the distance will be measured until the script is cancelled by pressing CTRL + C.  
That‘s it. You can use it many fields, but who still want to measure larger distances would have to rely on laser measuring devices, which, however, are much more expensive.

**Module 8: Node-Red Visualization**

# Node-RED Concepts

**Node**

A Node is the basic building block of a flow.

Nodes are triggered by either receiving a message from the previous node in a flow, or by waiting for some external event, such as an incoming HTTP request, a timer or GPIO hardware change. They process that message, or event, and then may send a message to the next nodes in the flow.

A node can have at most one input port and as many output ports as it requires.

**Configuration node**

A Configuration (config) Node is a special type of node that holds reusable configuration that can be shared by regular nodes in a flow.

For example, the MQTT In and Out nodes use an MQTT Broker config node to represent a shared connection to an MQTT broker.

Config nodes do not appear in the main workspace, but can be seen by opening the Configuration nodes sidebar.

**Flow**

A Flow is represented as a tab within the editor workspace and is the main way to organise nodes.

The term “flow” is also used to informally describe a single set of connected nodes. So a flow (tab) can contain multiple flows (sets of connected nodes).

**Context**

Context is a way to store information that can be shared between nodes without using the messages that pass through a flow.

There are three types of context;

* Node - only visible to the node that set the value
* Flow - visible to all nodes on the same flow (or tab in the editor)
* Global - visible to all nodes

By default, Node-RED uses an in-memory Context store so values do not get saved across restarts. It can be configured to use a file-system based store to make the values persistent. It is also possible to plug-in alternative storage plugins.

**Message**

Messages are what pass between the nodes in a flow. They are plain JavaScript objects that can have any set of properties. They are often referred to as msg within the editor.

By convention, they have a payload property containing the most useful information.

**Subflow**

A Subflow is a collection of nodes that are collapsed into a single node in the workspace.

They can be used to reduce some visual complexity of a flow, or to package up a group of nodes as a reusable component used in multiple places.

**Wire**

Wires connect the nodes and represent how messages pass through the flow.

**Palette**

The Palette is on the left of the editor and lists of the nodes that are available to use in flows.

Extra nodes can be installed into the palette using either the command-line or the Palette Manager.

**Workspace**

The Workspace is the main area where flows are developed by dragging nodes from the palette and wiring them together.

The workspace has a row of tabs along the top; one for each flow and any subflows that have been opened.

**Sidebar**

The sidebar contains panels that provide a number of useful tools within the editor. These include panels to view more information and help about a node, to view debug message and to view the flow’s configuration nodes.

# Running Node-RED locally

### Prerequisites

To install Node-RED locally you will need a [supported version of Node.js](https://nodered.org/docs/faq/node-versions). If you are on a Raspberry Pi or any Debian-based operating system, including Ubuntu.

### Installing with npm

To install Node-RED you can use the npm command that comes with node.js:

sudo npm install -g --unsafe-perm node-red

If you are using Windows, do not start the command with sudo. More information about installing Node-RED on Windows can be found [here](https://nodered.org/docs/getting-started/windows).

This command will install Node-RED as a global module along with its dependencies.

You can confirm it has succeeded if the end of the command output looks similar to:

+ node-red@1.1.0

added 332 packages from 341 contributors in 18.494s

found 0 vulnerabilities

### Installing with docker

To run in Docker in its simplest form just run:

docker run -it -p 1880:1880 --name mynodered nodered/node-red

For more detailed information see our [docker](https://nodered.org/docs/getting-started/docker) guide.

### Installing with snap

If your OS supports [Snap](https://snapcraft.io/docs/core/install) you can install Node-RED with:

sudo snap install node-red

When installed as a Snap package, it will run in a secure container that does not have access to some extra facilities that may be needed for you to use, such as:

* access to main system storage. Can only read/write to local home directories.
* gcc - needed to compile any binary components of nodes you want to install
* git - needed if you want to use the Projects feature
* direct access to gpio hardware
* access to any external commands your flows want to use with the Exec node (for example).

You can run it in “classic” mode which reduces the container security but then does provide wider access.

### Running

Once installed as a global module you can use the node-red command to start Node-RED in your terminal. You can use Ctrl-C or close the terminal window to stop Node-RED.

$ node-red

Welcome to Node-RED

===================

30 Jun 23:43:39 - [info] Node-RED version: v1.1.0

30 Jun 23:43:39 - [info] Node.js version: v10.21.0

30 Jun 23:43:39 - [info] Darwin 18.7.0 x64 LE

30 Jun 23:43:39 - [info] Loading palette nodes

30 Jun 23:43:44 - [warn] rpi-gpio : Raspberry Pi specific node set inactive

30 Jun 23:43:44 - [info] Settings file : /Users/nol/.node-red/settings.js

30 Jun 23:43:44 - [info] HTTP Static : /Users/nol/node-red/web

30 Jun 23:43:44 - [info] Context store : 'default' [module=localfilesystem]

30 Jun 23:43:44 - [info] User directory : /Users/nol/.node-red

30 Jun 23:43:44 - [warn] Projects disabled : set editorTheme.projects.enabled=true to enable

30 Jun 23:43:44 - [info] Creating new flows file : flows\_noltop.json

30 Jun 23:43:44 - [info] Starting flows

30 Jun 23:43:44 - [info] Started flows

30 Jun 23:43:44 - [info] Server now running at http://127.0.0.1:1880/red/

You can then access the Node-RED editor by pointing your browser at <http://localhost:1880>.

The log output provides you various pieces of information:

* The versions of Node-RED and Node.js
* Any errors hit when it tried to load the palette nodes
* The location of your Settings file and User Directory
* The name of the flows file it is using.

Node-RED uses flows\_<hostname>.json as the default flows file. You can change this by providing the flow file name as argument to the node-red [command](https://nodered.org/docs/getting-started/local#command-line-usage).

### Command-line Usage

Node-RED can be started using the command node-red. This command can take various arguments:

node-red [-v] [-?] [--settings settings.js] [--userDir DIR]

[--port PORT] [--title TITLE] [--safe] [flows.json|projectName]

[-D X=Y|@file]

| **Option** | **Description** |
| --- | --- |
| -p, --port PORT | Sets the TCP port the runtime listens on. Default: 1880 |
| --safe | Starts Node-RED without starting the flows. This allows you to open the flows in the editor and make changes without the flows running. When you deploy your changes, the flows are then started. |
| -s, --settings FILE | Sets the settings file to use. Default: settings.js in userDir |
| --title TITLE | Set process window title |
| -u, --userDir DIR | Sets the user directory to use. Default: ~/.node-red |
| -v | Enables verbose output |
| -D X=Y|@file | [Override individual settings](https://nodered.org/docs/getting-started/local#override-individual-settings) |
| -?, --help | Shows command-line usage help and exits |
| flows.json|projectName | If the Projects feature is not enabled, this sets the flow file you want to work with. If the Projects feature is enabled, this identifies which project should be started. |

Node-RED uses flows\_<hostname>.json as the default flows file. If the computer you are running on may change its hostname, then you should ensure you provide a static file name; either as a command-line argument or using the flowsFile option in your [settings file](https://nodered.org/docs/user-guide/runtime/settings-file).

#### Override individual settings

Since Node-RED 1.1.0

You can override individual settings on the command-line using the -D (or --define) option.

For example, to change the logging level you can use:

-D logging.console.level=trace

You can also provide the custom settings as a file:

-D @./custom-settings.txt

The file should contain a list of the settings to override:

logging.console.level=trace

logging.console.audit=true

### Passing arguments to the underlying Node.js process

There are occasions when it is necessary to pass arguments to the underlying Node.js process. For example, when running on devices like the Raspberry Pi or BeagleBone Black that have a constrained amount of memory.

To do this, you must use the node-red-pi start script in place of node-red. Note: this script is not available on Windows.

Alternatively, if are running Node-RED using the node command, you must provide arguments for the node process before specifying red.js and the arguments you want passed to Node-RED itself.

The following two commands show these two approaches:

node-red-pi --max-old-space-size=128 --userDir /home/user/node-red-data/

node --max-old-space-size=128 red.js --userDir /home/user/node-red-data/

### Upgrading Node-RED

If you installed Node-RED using the Pi script, you can rerun it to upgrade. The script is available [here](https://nodered.org/docs/hardware/raspberrypi).

If you have installed Node-RED as a global npm package, you can upgrade to the latest version with the following command:

sudo npm install -g --unsafe-perm node-red

If you are using Windows, do not start the command with sudo.

**Running on Raspberry Pi**

**Prerequisites**

If you are using Raspbian, then you must have Raspbian Jessie as a minimum version. Raspbian Buster is the currently supported version.

**Installing and Upgrading Node-RED**

We provide a script to install Node.js, npm and Node-RED onto a Raspberry Pi. The script can also be used to upgrade an existing install when a new release is available.

Running the following command will download and run the script. If you want to review the contents of the script first, you can view it [here](https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered).

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

This script will work on any **Debian-based** operating system, including **Ubuntu** and **Diet-Pi**. You may need to run sudo apt install build-essential git first to ensure npm is able to build any binary modules it needs to install.

This script will:

* remove the pre-packaged version of Node-RED and Node.js if they are present
* install the current Node.js LTS release using the [NodeSource](https://github.com/nodesource/distributions/blob/master/README.md). If it detects Node.js is already installed from NodeSource, it will ensure it is at least Node 8, but otherwise leave it alone
* install the latest version of Node-RED using npm
* optionally install a collection of useful Pi-specific nodes
* setup Node-RED to run as a service and provide a set of commands to work with the service

Node-RED has also been packaged for the Raspbian repositories and appears in their list of 'Recommended Software'. This allows it to be installed using apt-get install nodered and includes the Raspbian-packaged version of Node.js, but *does not* include npm.

While using these packages is convenient at first, we **strongly recommend** using our install script above instead.

**Running locally**

As with [running Node-RED locally](http://nodered.org/docs/getting-started/local), you can use the node-red command to run Node-RED in a terminal. It can then be stopped by pressing Ctrl-C or by closing the terminal window.

Due to the limited memory of the Raspberry Pi, you will need to start Node-RED with an additional argument to tell the underlying Node.js process to free up unused memory sooner than it would otherwise.

To do this, you should use the alternative node-red-pi command and pass in the max-old-space-size argument.

node-red-pi --max-old-space-size=256

**Running as a service**

The install script for the Pi also sets it up to run as a service. This means it can run in the background and be enabled to automatically start on boot.

The following commands are provided to work with the service:

* node-red-start - this starts the Node-RED service and displays its log output. Pressing Ctrl-C or closing the window does *not* stop the service; it keeps running in the background
* node-red-stop - this stops the Node-RED service
* node-red-restart - this stops and restarts the Node-RED service
* node-red-log - this displays the log output of the service

You can also start the Node-RED service on the Raspbian Desktop by selecting the Menu -> Programming -> Node-RED menu option.

**Autostart on boot**

If you want Node-RED to run when the Pi is turned on, or re-booted, you can enable the service to autostart by running the command:

sudo systemctl enable nodered.service

To disable the service, run the command:

sudo systemctl disable nodered.service

**Opening the editor**

Once Node-RED is running you can access the editor in a browser.

If you are using the browser on the Pi desktop, you can open the address: <http://localhost:1880>.

We recommend using a browser outside of the PI and pointing it at Node-RED running on the Pi. However you can use the built in browser and if so we recommend Chromium or Firefox-ESR and *not* Epiphany

When browsing from another machine you should use the hostname or IP-address of the Pi: http://<hostname>:1880. You can find the IP address by running hostname -I on the Pi.

**Creating your first flow**

**Overview**

This tutorial introduces the Node-RED editor and creates a flow that demonstrates the Inject, Debug and Function nodes.

**1. Access the editor**

With Node-RED [running](https://nodered.org/docs/getting-started), open the editor in a web browser.

If you are using a browser on the same computer that is running Node-RED, you can access it with the url: <http://localhost:1880>.

If you are using a browser on another computer, you will need to use the ip address of the computer running Node-RED: http://<ip-address>:1880.

**2. Add an Inject node**

The Inject node allows you to inject messages into a flow, either by clicking the button on the node, or setting a time interval between injects.

Drag one onto the [workspace](https://nodered.org/docs/user-guide/editor/workspace/) from the [palette](https://nodered.org/docs/user-guide/editor/palette/).

Select the newly added Inject node to see information about its properties and a description of what it does in the [Information sidebar pane](https://nodered.org/docs/user-guide/editor/sidebar/info).

**3. Add a Debug node**

The Debug node causes any message to be displayed in the [Debug sidebar](https://nodered.org/docs/user-guide/editor/sidebar/debug). By default, it just displays the payload of the message, but it is possible to display the entire message object.

**4. Wire the two together**

Connect the Inject and Debug nodes together by [dragging between](https://nodered.org/docs/user-guide/editor/workspace/wires) the output port of one to the input port of the other.

**5. Deploy**

At this point, the nodes only exist in the editor and must be deployed to the server.

Click the Deploy button.

With the Debug sidebar tab selected, click the Inject button. You should see numbers appear in the sidebar. By default, the Inject node uses the number of milliseconds since January 1st, 1970 as its payload.

**6. Add a Function node**

The Function node allows you to pass each message though a JavaScript function.

Delete the existing wire (select it and press delete on the keyboard).

Wire a Function node in between the Inject and Debug nodes.

Double-click on the Function node to bring up the edit dialog. Copy the following code into the function field:

// Create a Date object from the payload

var date = new Date(msg.payload);

// Change the payload to be a formatted Date string

msg.payload = date.toString();

// Return the message so it can be sent on

return msg;

Click Done to close the edit dialog and then click the deploy button.

Now when you click the Inject button, the messages in the sidebar will now be formatted is readable timestamps.

# Creating your second flow

### Overview

This tutorial builds on the [first tutorial](https://nodered.org/docs/tutorials/first-flow) to make a flow that starts to bring in data from external sources to do something useful locally.

The flow will:

* Retrieve information from a website at a regular interval
* Convert that information into a useful form
* Display the result in the Debug sidebar

### 1. Add an Inject node

In the [previous tutorial](https://nodered.org/docs/tutorials/first-flow), the Inject node was used to trigger the flow when its button was clicked. For this tutorial, the Inject node will be configured to trigger the flow at a regular interval.

Drag an Inject node onto the workspace from the palette.

Double click the node to bring up the edit dialog. Set the repeat interval to every 5 minutes.

Click Done to close the dialog.

### 2. Add an HTTP Request node

The HTTP Request node can be used to retrieve a web-page when triggered.

After adding one to the workspace, edit it to set the URL property to:

https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/significant\_month.csv

Then click Done to close the dialog.

This URL is a feed of significant earthquakes in the last month from the US Geological Survey web site. The site offers a number of [other options](https://earthquake.usgs.gov/earthquakes/feed/v1.0/csv.php) that you may want to play around with after completing this tutorial.

### 3. Add a CSV node

Add a CSV node and edit its properties. Enable option for ‘First row contains column names’.

Then click Done to close.

### 4. Add a Debug node

Add a Debug node to the output.

### 5. Wire them all together

Add wires connecting:

* The Inject node output to the HTTP Request node input.
* The HTTP Request node output to the CSV node input.
* The CSV node output to the Debug node input.

### 7. Add a Switch node

Add a Switch node to the workspace. Edit its properties and configure it to check the property msg.payload.mag with a test of >= change it to test on a number and the value 7. Click Done to close.

Add a second wire from the CSV node to this Switch node.

### 8. Add a Change node

Add a Change node, wired to the output of the Switch node. Configure it to set msg.payload to the string PANIC!.

### 9. Add a Debug node

Wire a new Debug node to the output of the Change node

### 10. Deploy

Deploy the flow to the runtime by clicking the Deploy button.