M.TECH 1ST YEAR-IInd SEMESTER

IoT Applications Lab Manual

PG- EMBEDDED SYSTEMS

Department of Electronics and Communication Engineering

B.V.Raju Institute of Technology
Vishnupur, Narsapur, Medak.(Dt),Telangana

LIST OF EXPERIMENTS

1. Study of Node MCU, Raspberry-Pi and other boards.

2. Study of different operating systems for Raspberry-Pi. Understanding the process of OS installation on Raspberry-Pi.

3. Interfacing basic I/O components with Node MCU

4. Developing an local web server with Node MCU

5. Develop an application to control outputs using web socket server

6. To establish a Wi-Fi communication (HTTP) between two ESP8266 Node MCU boards to exchange data without the need to connect to the internet

7. Interface any Sensor with Node MCU and load the Sensor data into cloud

8. Study of Connectivity and configuration of Raspberry-Pi circuit with basic peripherals, LEDS. Understanding GPIO and its use in program.

9. Understanding the connectivity of Raspberry-Pi board circuit with temperature sensor. Write an application to read the environment temperature and load the data into cloud. If temperature crosses a threshold value, the application indicated user using LEDS

10. Understanding the connectivity of Raspberry-Pi circuit with IR sensor. Write an application to detect obstacle and notify user using LEDs.

11. Understanding and connectivity of Raspberry-Pi with camera. Write an application to capture image/Video over HTTP with Raspberry Pi Camera

12. Write an application using Raspberry-Pi to control the operation of stepper motor and controlling using it web.

13. To establish a communication between a Raspberry Pi running the Node-RED software and an ESP8266 using MQTT

14. Develop a simple home automation system with Node MCU/Raspberry Pi

1. Study of Node MCU, Raspberry-Pi and other boards.

NODE MCU Board:

NodeMCU is a low-cost open source IoT platform. It initially included firmware which runs on the ESP8266 Wi-Fi SoC from Espressif Systems, and hardware which was based on the ESP-12 module. Later, support for the ESP32 32-bit MCU was added.

NodeMCU is an open source firmware for which open source prototyping board designs are available. The name "NodeMCU" combines "node" and "MCU" (micro-controller unit). The term "NodeMCU" strictly speaking refers to the firmware rather than the associated development kits

Both the firmware and prototyping board designs are open source

The firmware uses the Lua scripting language. The firmware is based on the eLua project, and built on the Espressif Non-OS SDK for ESP8266. It uses many open source projects, such as lua-cjson and SPIFFS. Due to resource constraints, users need to select the modules relevant for their project and build a firmware tailored to their needs. Support for the 32-bit ESP32 has also been implemented. The **ESP8266** is a low-cost Wi-Fi microchip, with a full TCP/IP stack and microcontroller capability, produced by Espressif Systems in Shanghai, China. The chip first came to the attention of Western makers in August 2014 with the **ESP-01** module, made by a third-party manufacturer Ai-Thinker. This small module allows microcontrollers to connect to a Wi-Fi network and make simple TCP/IP connections using Hayes-style commands. However, at first, there was almost no English-language documentation on the chip and the commands it accepted. The very low price and the fact that there were very few external components on the module, which suggested that it could eventually be very inexpensive in volume, attracted many hackers to explore the module, the chip, and the software on it, as well as to translate the Chinese documentation

The **ESP8285** is an ESP8266 with 1 MB of built-in flash, allowing the building of single-chip devices capable of connecting to Wi-Fi. These microcontroller chips have been succeeded by the ESP32 family of devices, including the pin-compatible ESP32-C3.

The prototyping hardware typically used is a circuit board functioning as a dual in-line package (DIP) which integrates a USB controller with a smaller surface-mounted board containing the MCU and antenna. The choice of the DIP format allows for easy prototyping on breadboards. The design was initially based on the ESP-12 module of the ESP8266, which is a Wi-Fi SoC integrated with a Tensilica Xtensa LX106 core, widely used in IoT applications

There are two available versions of NodeMCU as version 0.9 & 1.0 where the version 0.9 contains **ESP-12** and version 1.0 contains **ESP-12E** where E stands for "Enhanced"

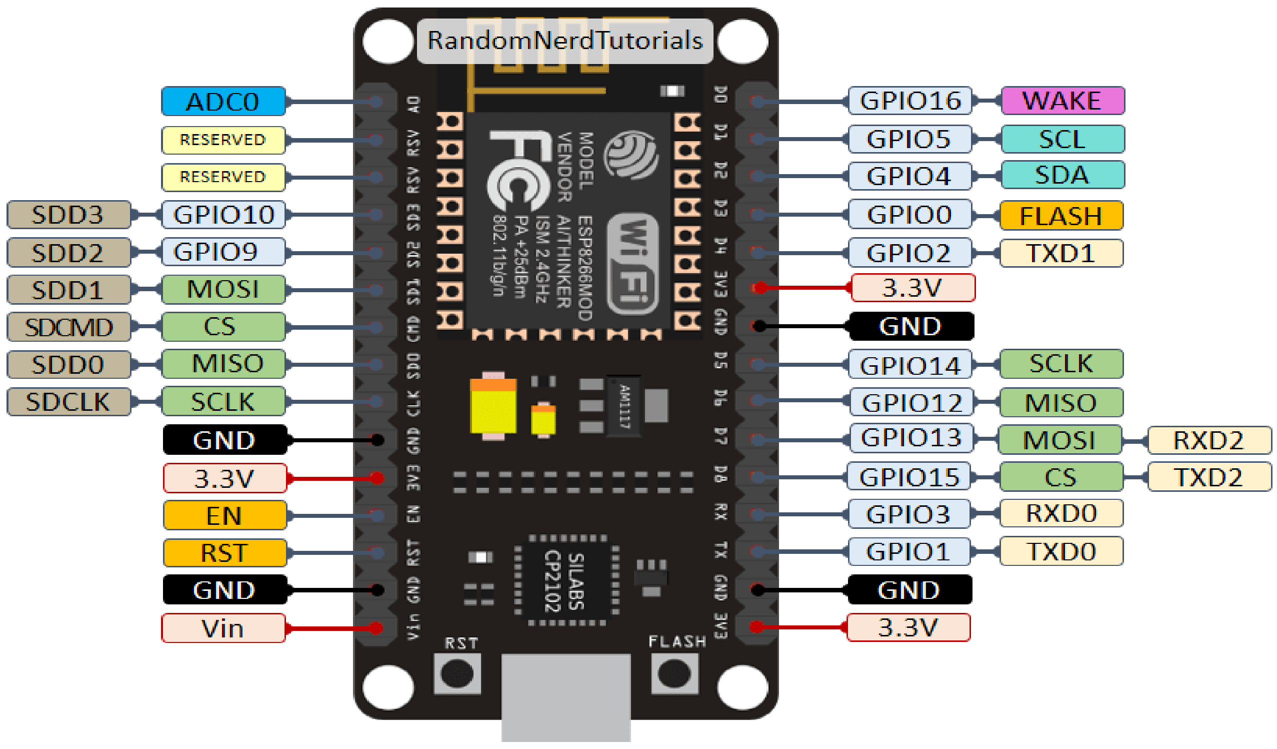


Fig: Node MCU Board Pin Layout

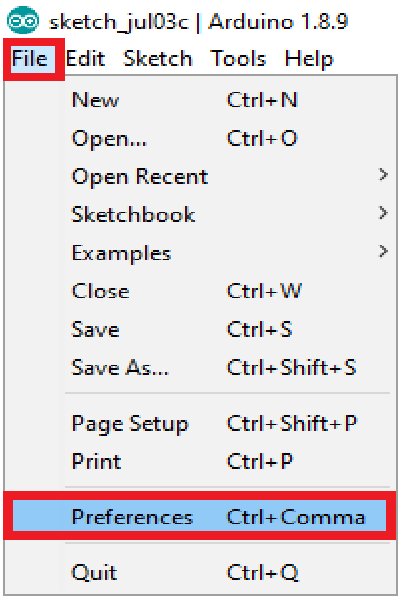
Specifications	NodeMCU v1.0	Arduino Pro Mini
MCU	32 bit Tensilica L106	8 bit ATmega328P
Frequency	80/160 MHz	16 MHz
Input-Output	17xDIO	14xDIO
ADC Pin	1x10 bit (1V)	6x10 Bit(3V3)
Operating Voltage	3.0~3.6V	3.0~3.6V
Program Memory	4MB	32kB
WiFi	IEEE 802.11 b/g/n	-

Table:Node MCU Board Specifications

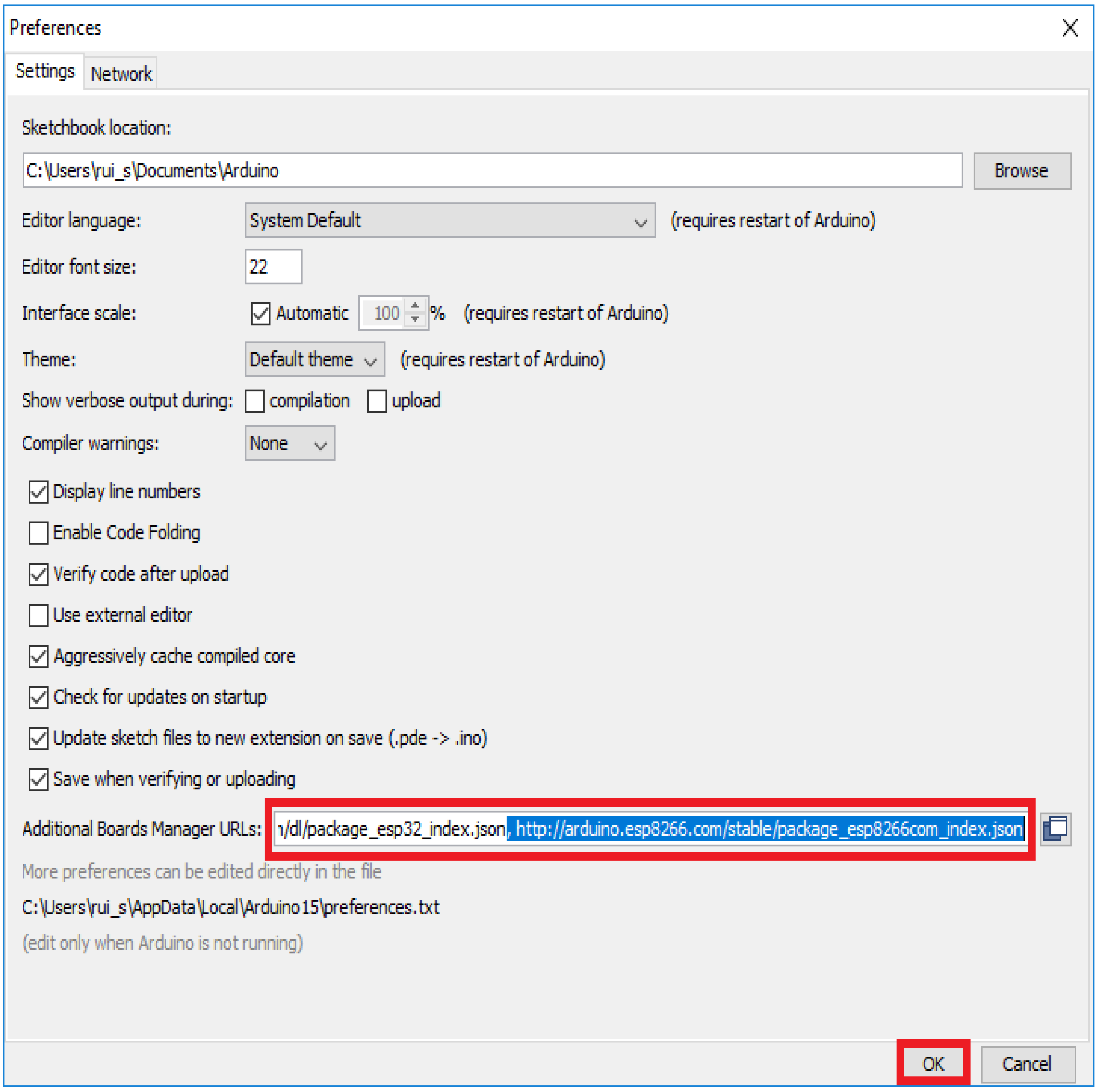
Install ESP8266 Add-on in Arduino IDE

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

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



2. Enter **http://arduino.esp8266.com/stable/package_esp8266com_index.json** into the “Additional Boards Manager URLs” field as shown in the figure below. Then, click the “OK” button:



Preferences

Settings | Network

Sketchbook location: C:\Users\rui_s\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 22

Interface scale: ☑ Automatic 100 % (requires restart of Arduino)

Theme: Default theme (requires restart of Arduino)

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☑ Display line numbers

☐ Enable Code Folding

☑ Verify code after upload

☐ Use external editor

☑ Aggressively cache compiled core

☑ Check for updates on startup

☑ Update sketch files to new extension on save (.pde -> .ino)

☑ Save when verifying or uploading

Additional Boards Manager URLs: n/dl/package_esp32_index.json, http://arduino.esp8266.com/stable/package_esp8266com_index.json

More preferences can be edited directly in the file

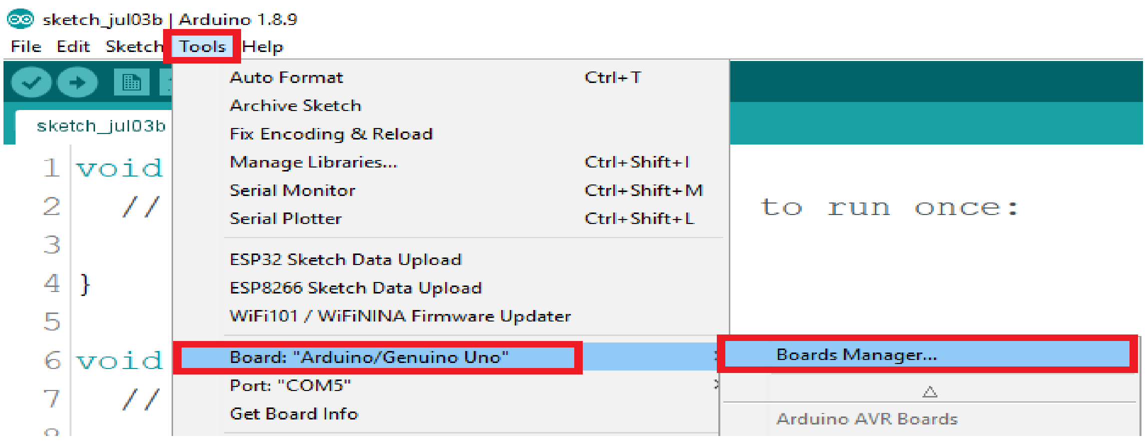
C:\Users\rui_s\AppData\Local\Arduino15\preferences.txt

(edit only when Arduino is not running)

Note: if you already have the ESP32 boards URL, you can separate the URLs with a comma as follows:

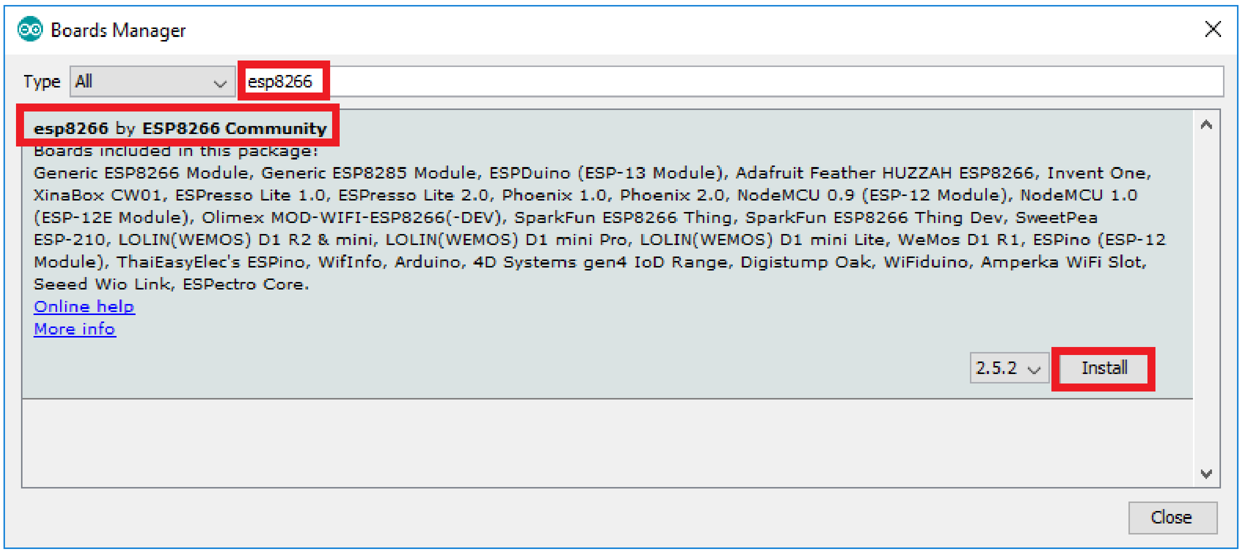
`https://dl.espressif.com/dl/package_esp32_index.json, http://arduino.esp8266.com/stable/package_esp8266com_index.json`

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



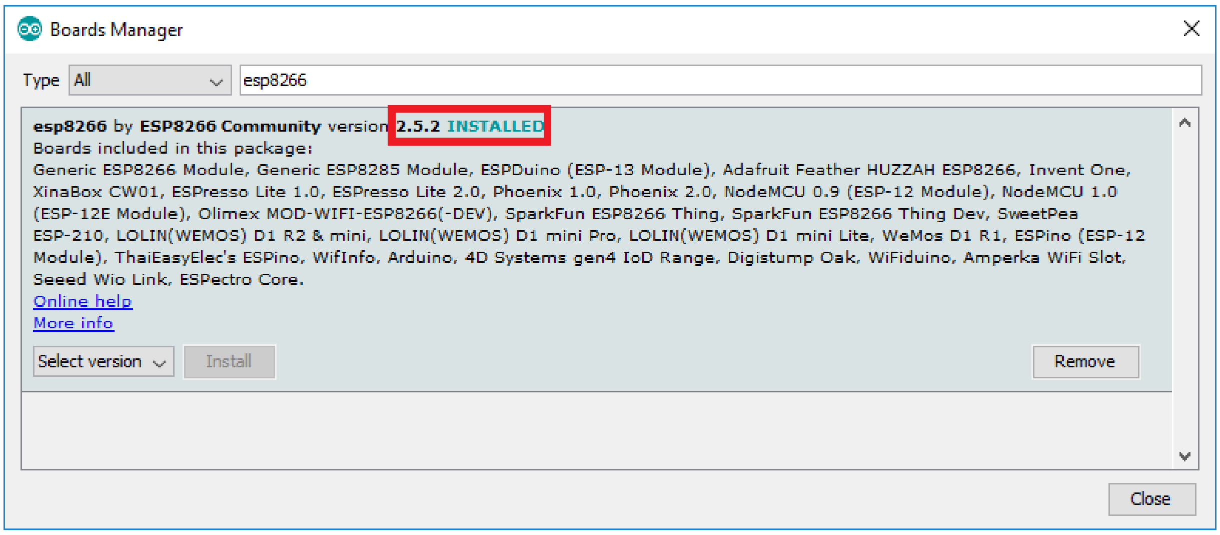
sketch_jul03b | Arduino 1.8.9
File Edit Sketch Tools Help
sketch_jul03b
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
ESP32 Sketch Data Upload
ESP8266 Sketch Data Upload
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino/Genuino Uno"
Port: "COM5"
Get Board Info
Boards Manager...
Arduino AVR Boards
void
//
}
void
//
to run once:

4. Search for **ESP8266** and press install button for the “**ESP8266 by ESP8266 Community**“:



esp8266 by **ESP8266 Community**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core.
Online help
More info
2.5.2

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

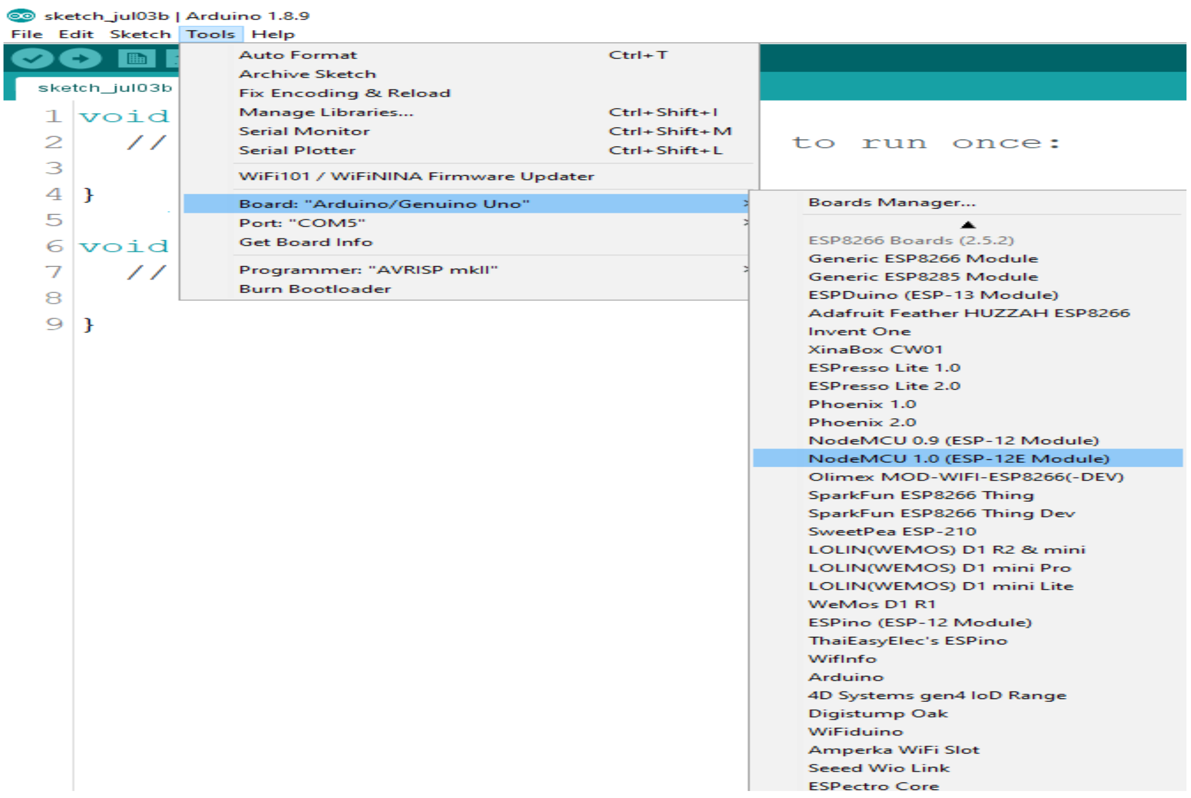


esp8266 by **ESP8266 Community** version **2.5.2 INSTALLED**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, Invent One, XinaBox CW01, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SweetPea ESP-210, LOLIN(WEMOS) D1 R2 & mini, LOLIN(WEMOS) D1 mini Pro, LOLIN(WEMOS) D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak, WiFiduino, Amperka WiFi Slot, Seeed Wio Link, ESPectro Core.
Online help
More info

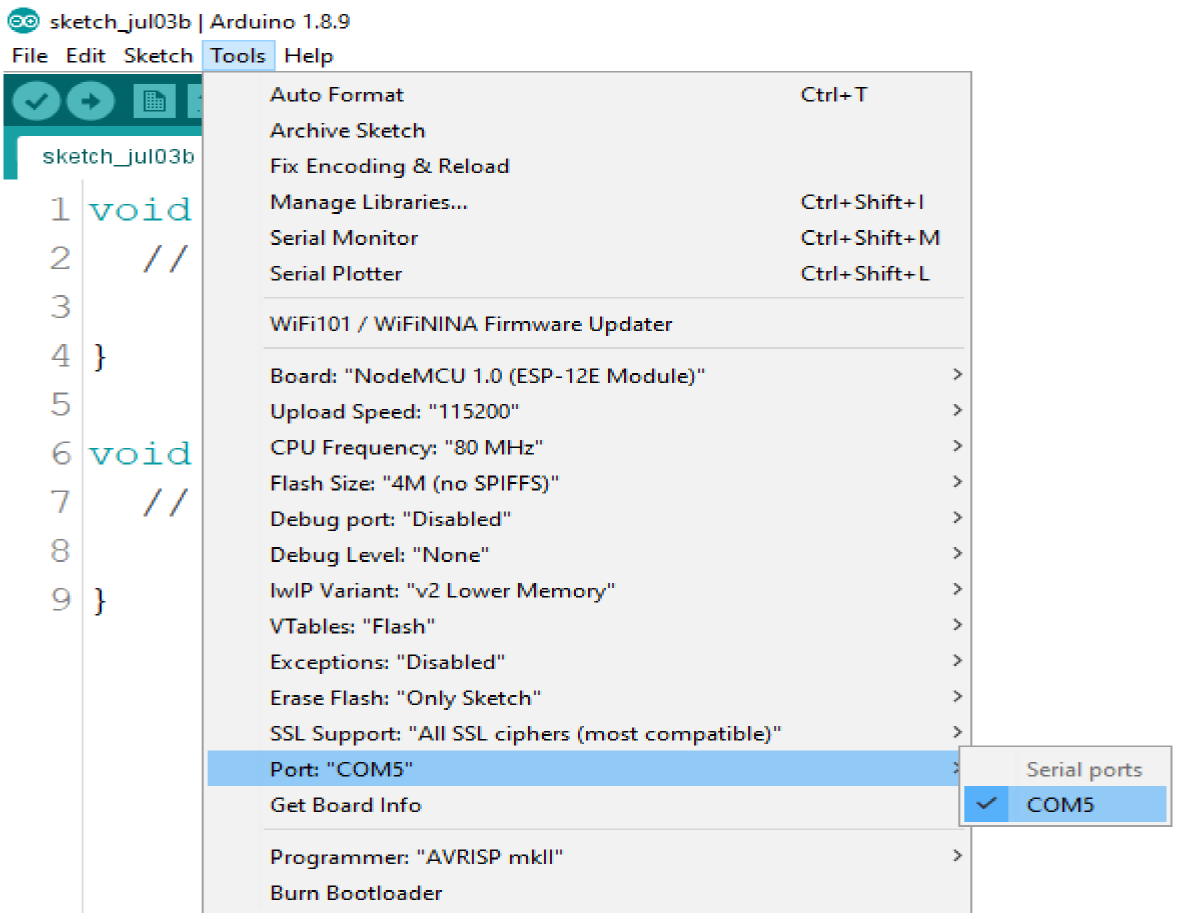
Uploading the Sketch

Uploading the Sketch to the ESP-12E

If you’re using an ESP-12E NodeMCU Kit, uploading the sketch is very simple, since it has built-in programmer. Plug your board to your computer. Make sure you have the right board selected:



You also need to select the Port:


sketch_jul03b | Arduino 1.8.9
File Edit Sketch Tools Help
sketch_jul03b
void
//
}
void
//
}
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater

Board: "NodeMCU 1.0 (ESP-12E Module)"
Upload Speed: "115200"
CPU Frequency: "80 MHz"
Flash Size: "4M (no SPIFFS)"
Debug port: "Disabled"
Debug Level: "None"
lwIP Variant: "v2 Lower Memory"
VTables: "Flash"
Exceptions: "Disabled"
Erase Flash: "Only Sketch"
SSL Support: "All SSL ciphers (most compatible)"
Port: "COM5"

Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
✓ COM5

2. Raspberry Pi Board

Raspberry Pi is a series of small single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom

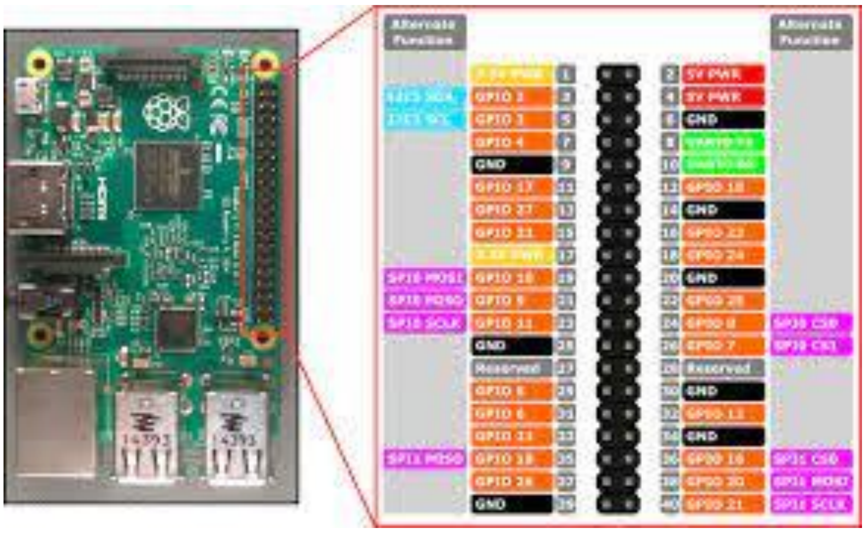


Fig: Raspberry Pi Pin Layout

There are several models of Raspberry Pi, and for most people the Raspberry Pi 3 Model B+ is the one to choose.

The Raspberry Pi 3 Model B+ is the newest, fastest, and easiest to use.

The Raspberry Pi Zero and Zero W are smaller and require less power, so they're useful for portable projects such as robots. It's generally easier to start a project with the Raspberry Pi 3, and to move to the Pi Zero when you have a working prototype that the smaller Pi would be useful for. If you want to buy a Raspberry Pi, head to rpf.io/products.

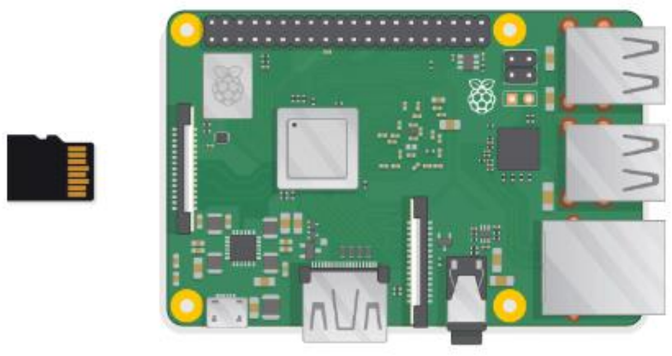
A power supply

To connect to a power socket, the Raspberry Pi has a micro USB port (the same found on many mobile phones).

A micro SD card

Your Raspberry Pi needs an SD card to store all its files and the Raspbian operating system.

You will need a micro SD card with a capacity of at least 8 GB.



Many sellers supply SD cards for Raspberry Pi that are already set up with Raspbian and ready to go.

A keyboard and a mouse

To start using your Raspberry, you will need a USB keyboard and a USB mouse.

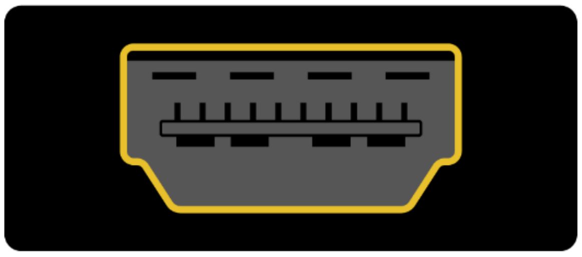
Once you've set your Pi up, you can use a Bluetooth keyboard and mouse, but you'll need a USB keyboard and mouse for setting up.

A TV or computer screen

To view the Raspbian desktop environment, you will need a screen and a cable to link the screen and the Pi. The screen can be a TV or a computer monitor. If the screen has built-in speakers, the Pi will be able to use these to play sound.

HDMI

The Raspberry Pi has a HDMI output port that is compatible with the HDMI port of most modern TVs and computer monitors. Many computer monitors may also have DVI or VGA ports.



Set up your SD card

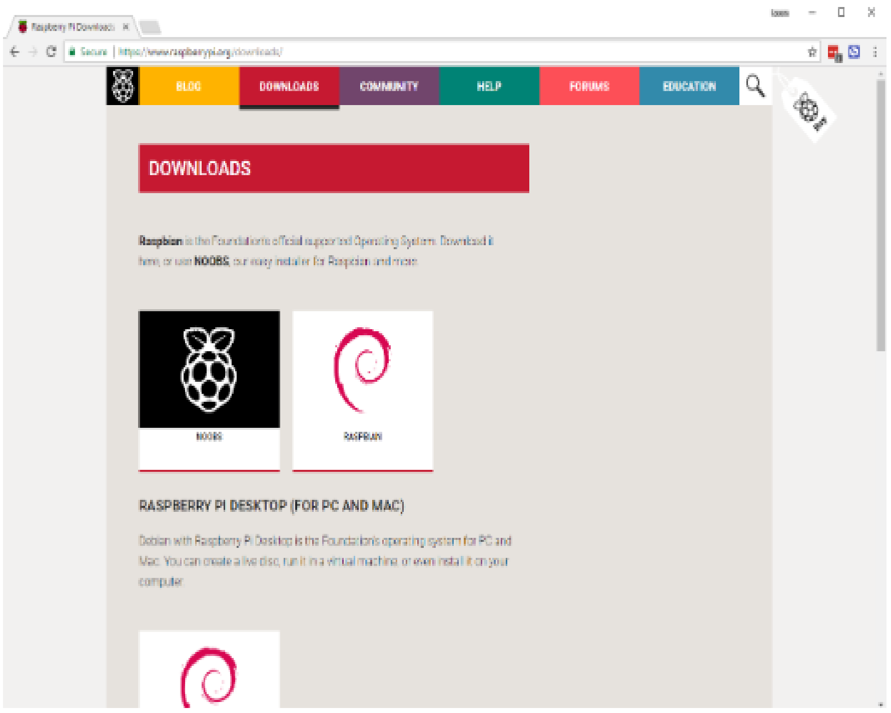
If you have an SD card that doesn't have the Raspbian operating system on it yet, or if you want to reset your Raspberry Pi, you can easily install Raspbian yourself. To do so, you need a computer that has an SD card port — most laptop and desktop computers have one.

The Raspbian operating system via NOOBS

Using the NOOBS software is the easiest way to install Raspbian on your SD card.

Download NOOBS

- Visit the Raspberry Pi downloads page.



DOWNLOADS

Raspbian is the Foundation's official supported Operating System. Download it here, or use **NOOBS**, our easy installer for Raspbian and more.

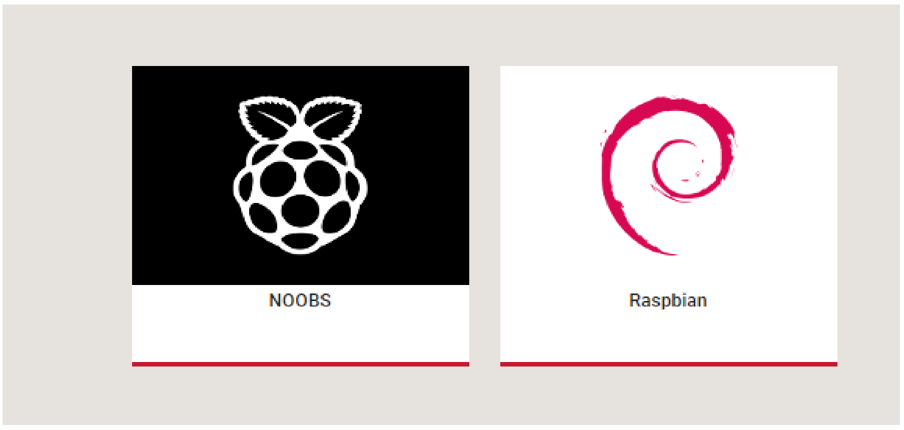
NOOBS

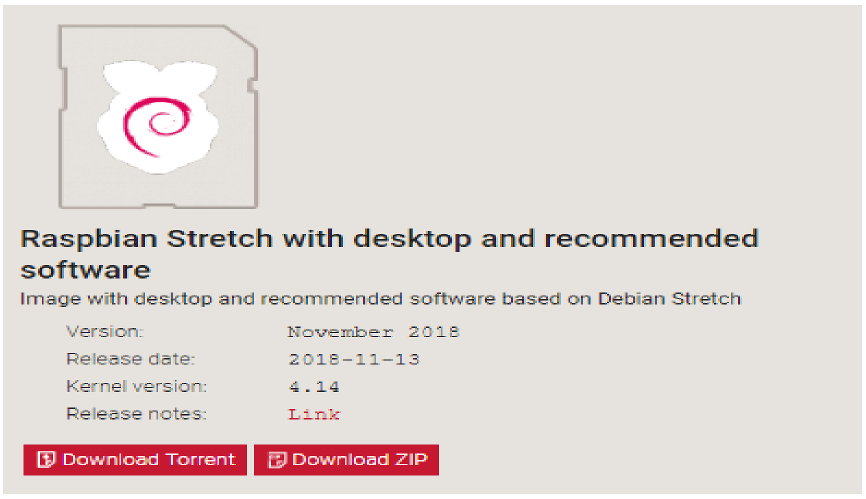
RASPBIAN

RASPBERRY PI DESKTOP (FOR PC AND MAC)

Debian with Raspberry Pi Desktop is the Foundation's operating system for PC and Mac. You can create a live disc, run it in a virtual machine, or even install it on your computer.

- You should see a box linking to the NOOBS or Raspbin files. Click on the box.





Raspbian Stretch with desktop and recommended software

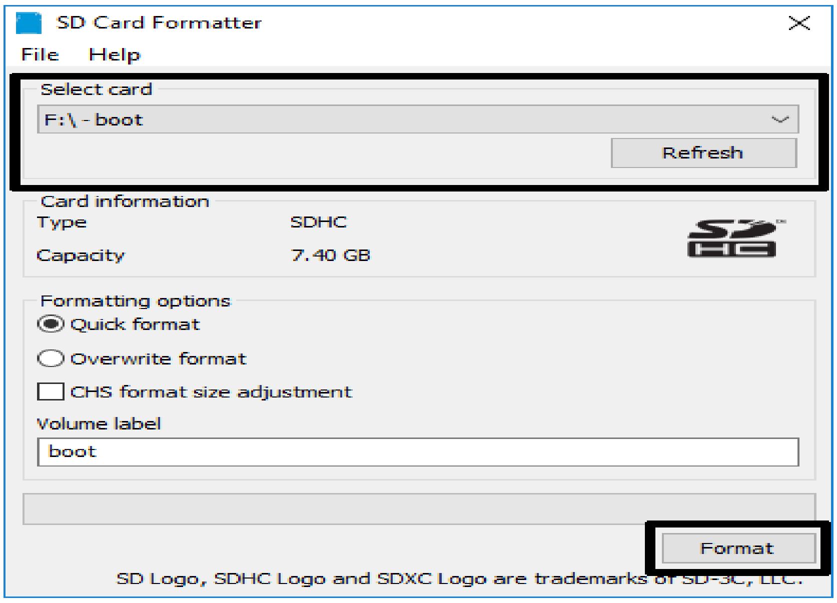
Image with desktop and recommended software based on Debian Stretch

Version:	November 2018
Release date:	2018-11-13
Kernel version:	4.14
Release notes:	Link

Format the SD card

Anything that's stored on the SD card will be overwritten during formatting. So if the SD card on which you want to install Raspbian currently has any files on it, e.g. from an older version of Raspbian, you may wish to back these files up first to not lose them permanently.

- Visit the SD Association's website and download SD Formatter for Windows or Mac.
- Follow the instructions to install the software.
- Insert your SD card into the computer or laptop's SD card slot.
- In SD Formatter, select your SD card, and the format the card.

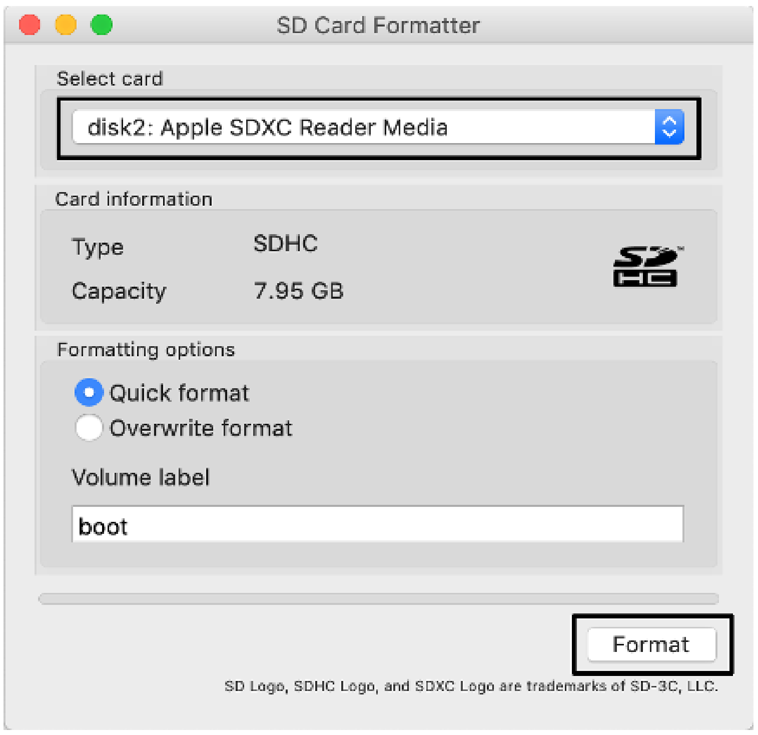

SD Card Formatter

Select card: F:\ - boot

Card information
Type SDHC
Capacity 7.40 GB

Formatting options
[✓ Quick format] Overwrite format
CHS format size adjustment
Volume label: boot

SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.


SD Card Formatter

Select card: disk2: Apple SDXC Reader Media

Card information
Type SDHC
Capacity 7.95 GB

Formatting options
[✓ Quick format] Overwrite format
Volume label: boot

SD Logo, SDHC Logo, and SDXC Logo are trademarks of SD-3C, LLC.

Extract NOOBS from the zip archive

Next, you will need to extract the files from the NOOBS zip archive you downloaded from the Raspberry Pi website.

- ➢ Find the downloaded archive — by default, it should be in your Downloads folder.
- ➢ Double-click on it to extract the files, and keep the resulting Explorer/Finder window open.

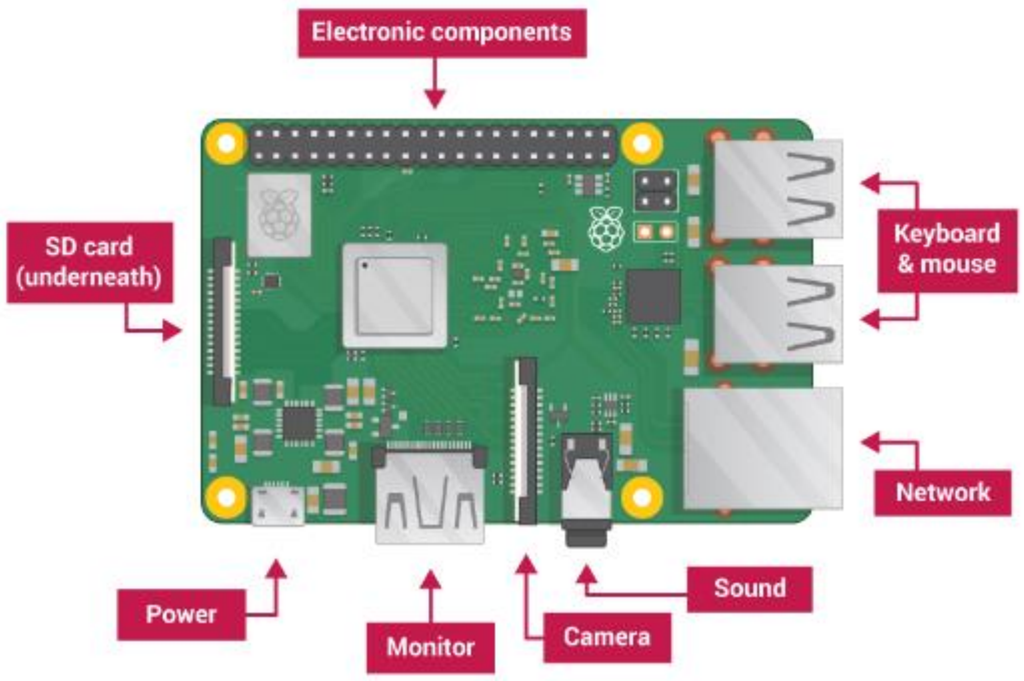
- ➢ Copy the files

- ➢ Now open another Explorer/Finder window and navigate to the SD card. It's best to position the two windows side by side.

- ➢ Select all the files in the NOOBS folder and drag them into the SD card window to copy them to the card.
- ➢ Once the files have all been copied over, you can eject the SD card.

Connect your Raspberry Pi

Let's get everything connected. It's important to do this in the right order, so that all your components are safe.



- ➢ Insert the SD card you've set up with Raspbian (via NOOBS) into the micro SD card slot at the underside of your Pi.
- ➢ Note: Lots of micro SD cards will come inside a larger adapter — you can slide the card out using the lip at the bottom.
- ➢ Find the USB cable for your mouse, and connect the mouse to a USB port on the Raspberry Pi (it doesn't matter which one).
- ➢ Connect the keyboard in the same way.
- ➢ Look at the HDMI port on the Raspberry Pi — notice that it has a large, flat side on top.
- ➢ Make sure your screen is plugged into a wall socket and turned on. Use a cable to connect the screen to the Pi's HDMI port — use an adapter if necessary.

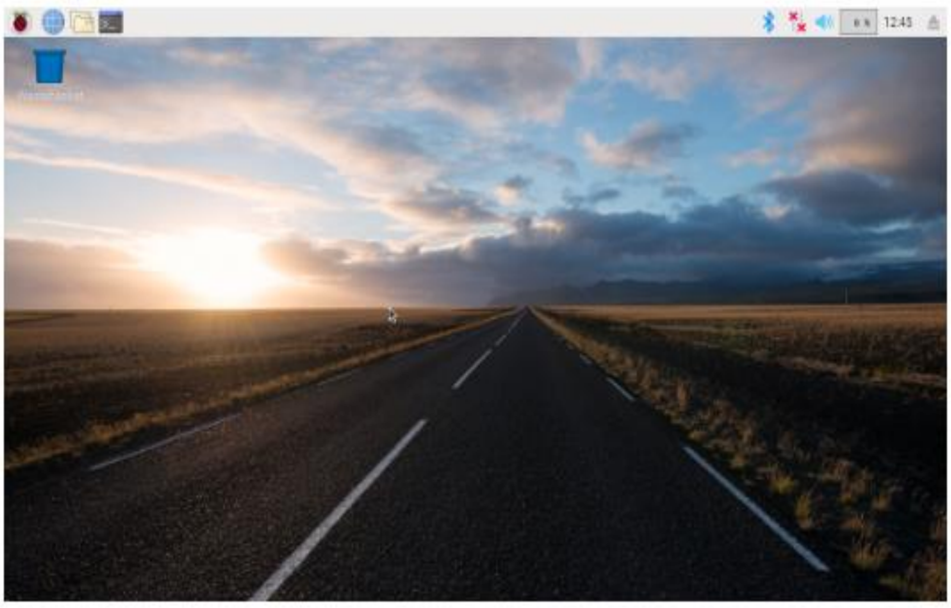
Note: nothing will display on the screen, because the Pi is not running yet.

- If you want to connect the Pi to the internet via Ethernet, use an Ethernet cable to connect the Ethernet port on the Raspberry Pi to an Ethernet socket on the wall or on your internet router. You don’t need to do this if you’ll be using WiFi or if you don’t want to connect to the internet.
- Sound will come from your screen if it has speakers or you can connect headphones or speakers to the audio jack if you have them.

Start up your Raspberry Pi

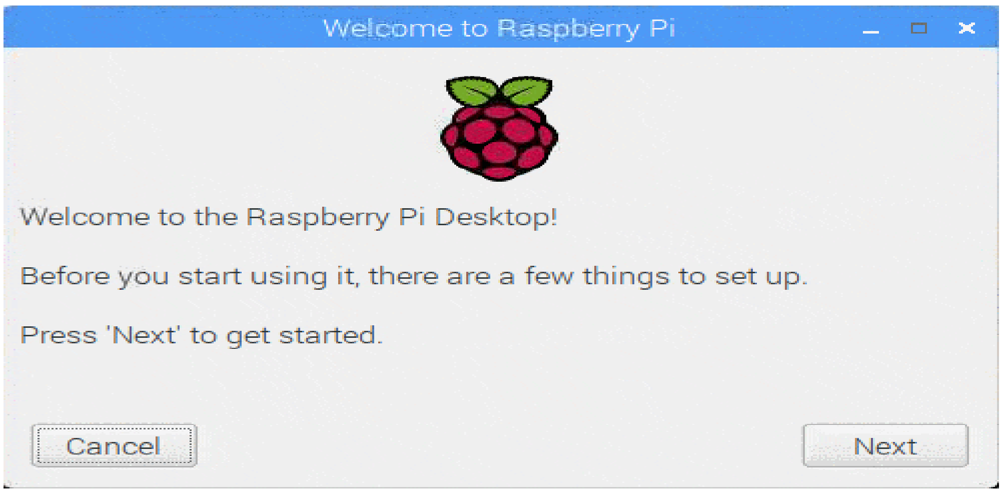
Your Raspberry Pi doesn’t have a power switch: as soon as you connect it to a power outlet, it will turn on. Notice that the Pi’s micro USB power port has a longer flat side on top.

- Plug a micro USB power supply into a socket and connect it to you Pi’s power port.
- You should see a red LED light up on the Raspberry Pi, which indicates that the Pi is connected to power. As it starts up (this is also called booting), you will see raspberries appear in the top left-hand of your screen.



Finish the setup

When you start your Raspberry Pi for the first time, the Welcome to Raspberry Pi application will pop up and guide you through the initial setup.

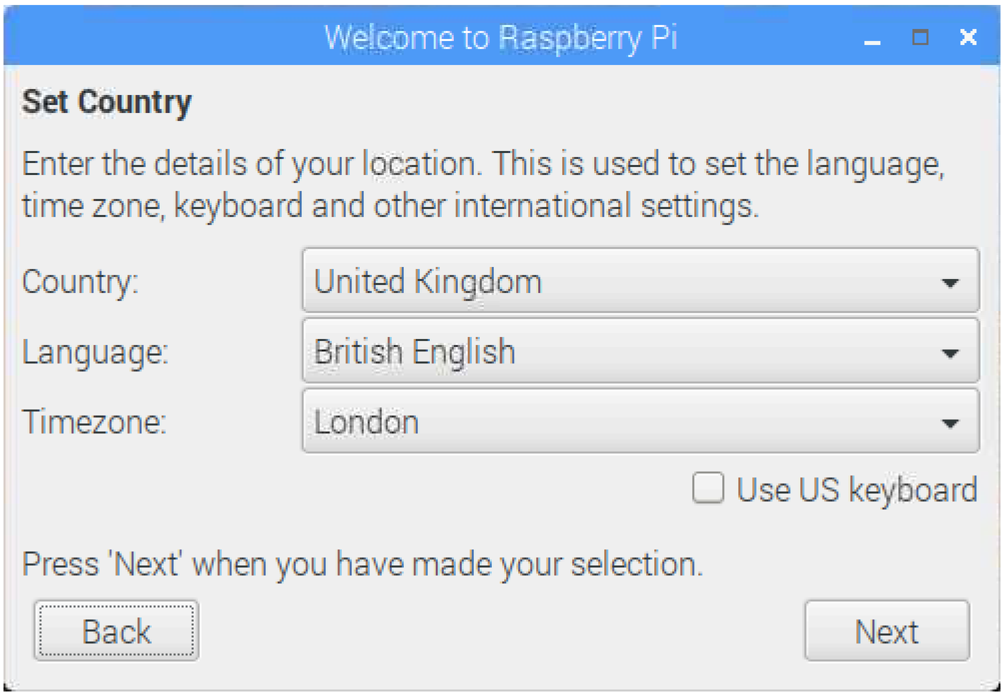


Welcome to the Raspberry Pi Desktop!

Before you start using it, there are a few things to set up.

Press 'Next' to get started.

- Click Next to start the setup.
- Set your Country, Language, and Timezone, then click Nextagain.


Welcome to Raspberry Pi

Set Country

Enter the details of your location. This is used to set the language, time zone, keyboard and other international settings.

Country: United Kingdom

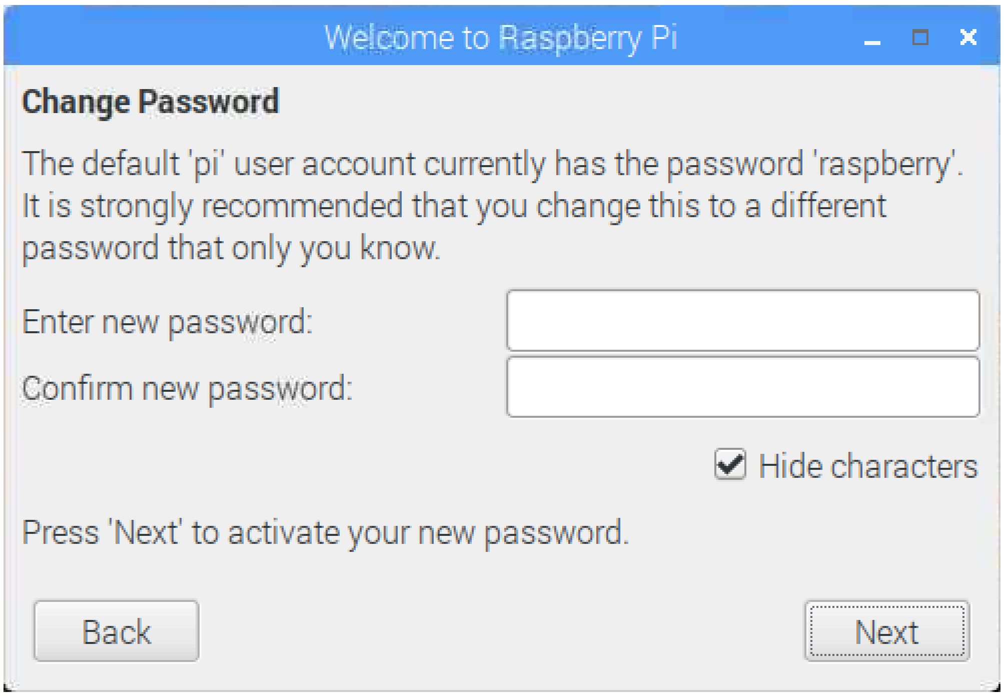
Language: British English

Timezone: London

☐ Use US keyboard

Press 'Next' when you have made your selection.

Enter a new password for your Raspberry Pi and click next.


Welcome to Raspberry Pi

Change Password

The default 'pi' user account currently has the password 'raspberry'. It is strongly recommended that you change this to a different password that only you know.

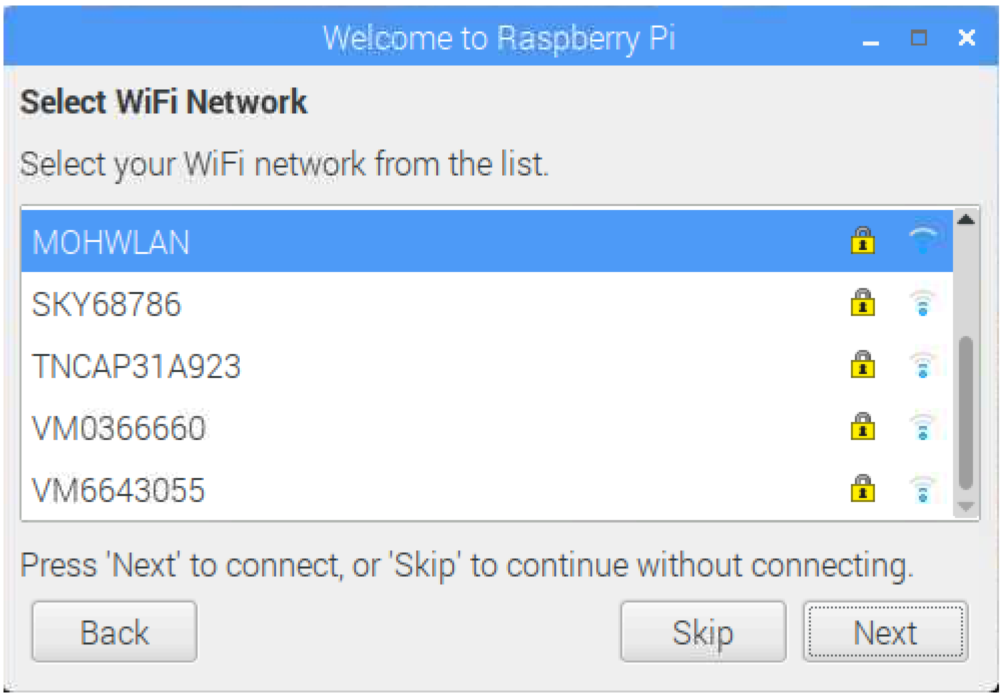
Enter new password:

Confirm new password:

✓ Hide characters

Press 'Next' to activate your new password.

- Connect to your WiFi network by selecting its name, entering the password, and clicking Next.


Welcome to Raspberry Pi

Select WiFi Network

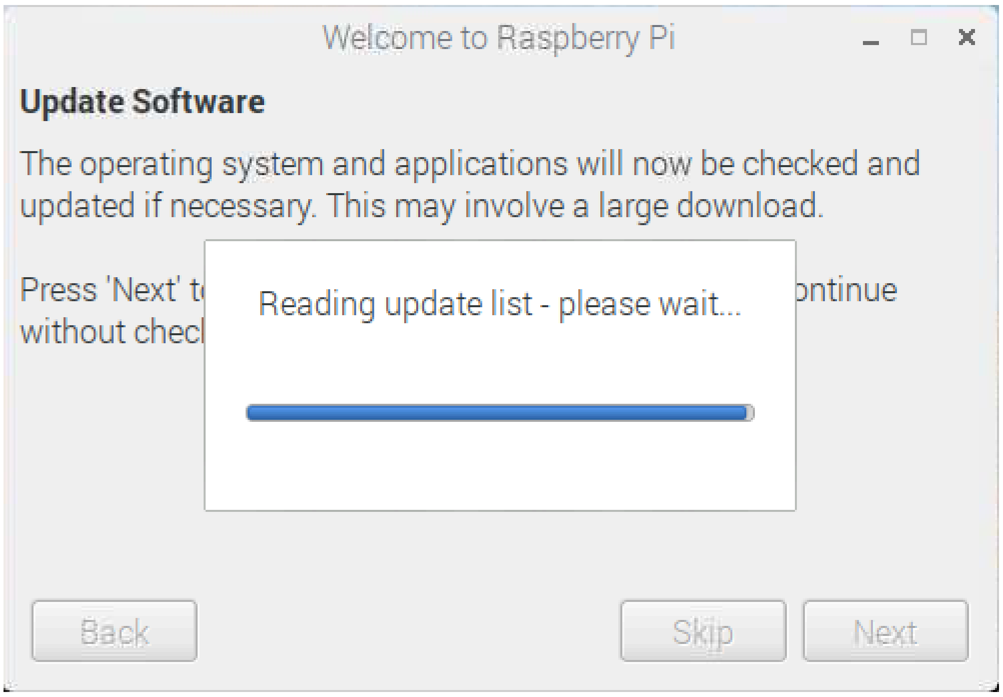
Select your WiFi network from the list.

MOHWLAN
SKY68786
TNCAP31A923
VM0366660
VM6643055

Press 'Next' to connect, or 'Skip' to continue without connecting.

Note: if your Raspberry Pi model doesn't have wireless connectivity, you won't see this screen.

- Click Next let the wizard check for updates to Raspbian and install them (this might take a little while).


Welcome to Raspberry Pi

Update Software

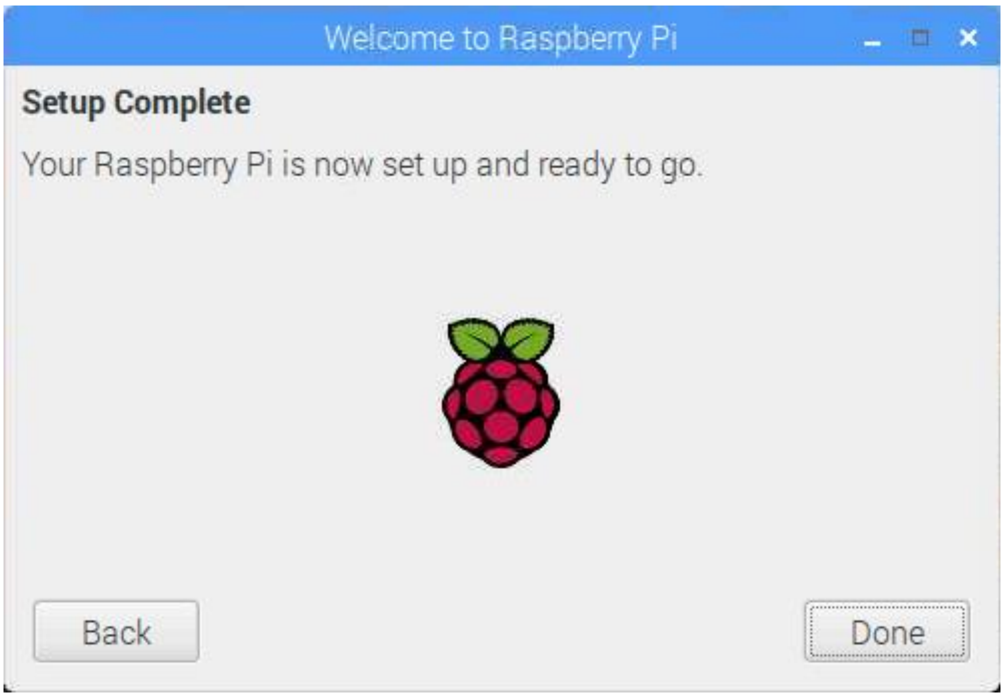
The operating system and applications will now be checked and updated if necessary. This may involve a large download.

Press 'Next' t ontinue without checl

Reading update list - please wait...

- Click Done or Reboot to finish the setup.

Note: you will only need to reboot if that's necessary to complete an update.



Setup Complete

Your Raspberry Pi is now set up and ready to go.

3. Interfacing basic I/O components with Node MCU

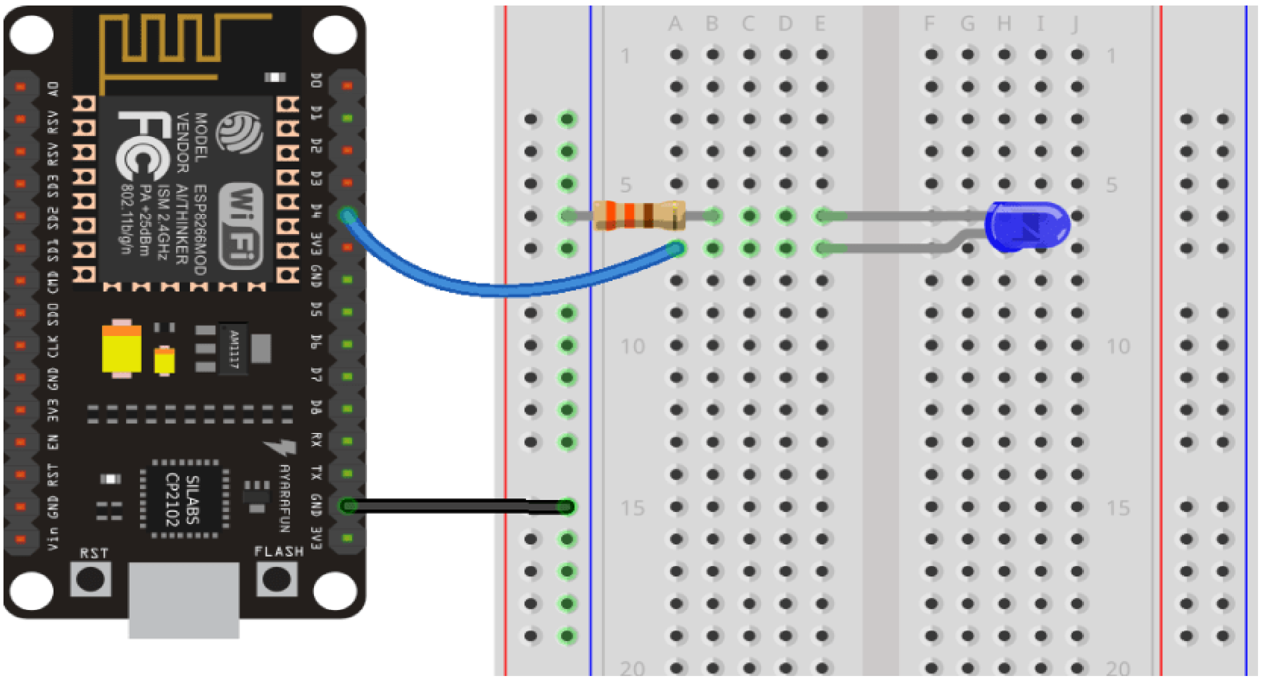
Hardware Required:

- ESP8266 Wi-Fi Development Board
- LED
- 330 Ohm resistor
- Breadboard
- Jumper wires

Code:

```
int pin = 2;
void setup() {
  // initialize GPIO 2 as an output.
  pinMode(pin, OUTPUT);
}
// the loop function runs over and over again forever
void loop() {
  digitalWrite(pin, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);               // wait for a second
  digitalWrite(pin, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);               // wait for a second
}
```

Hardware Connections:

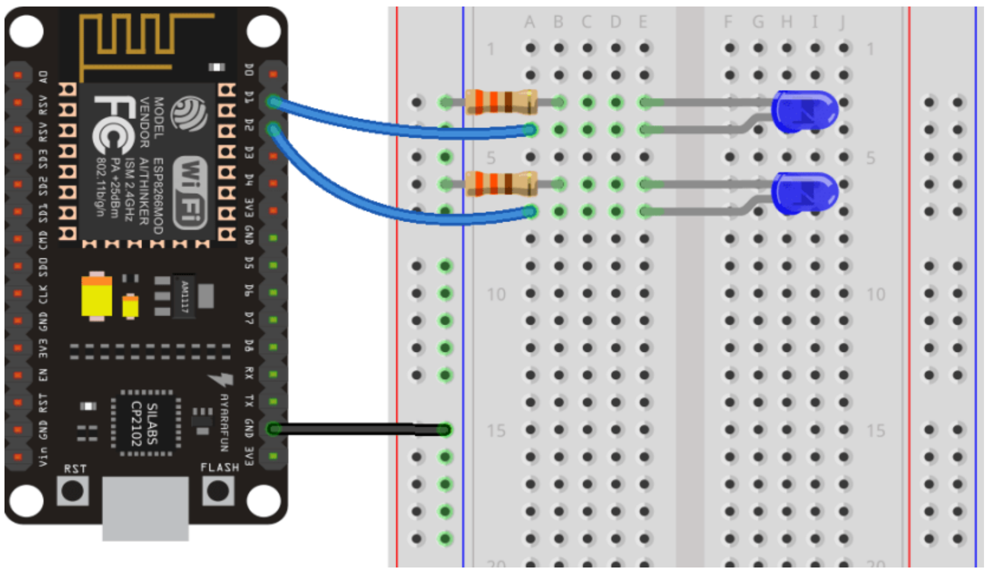


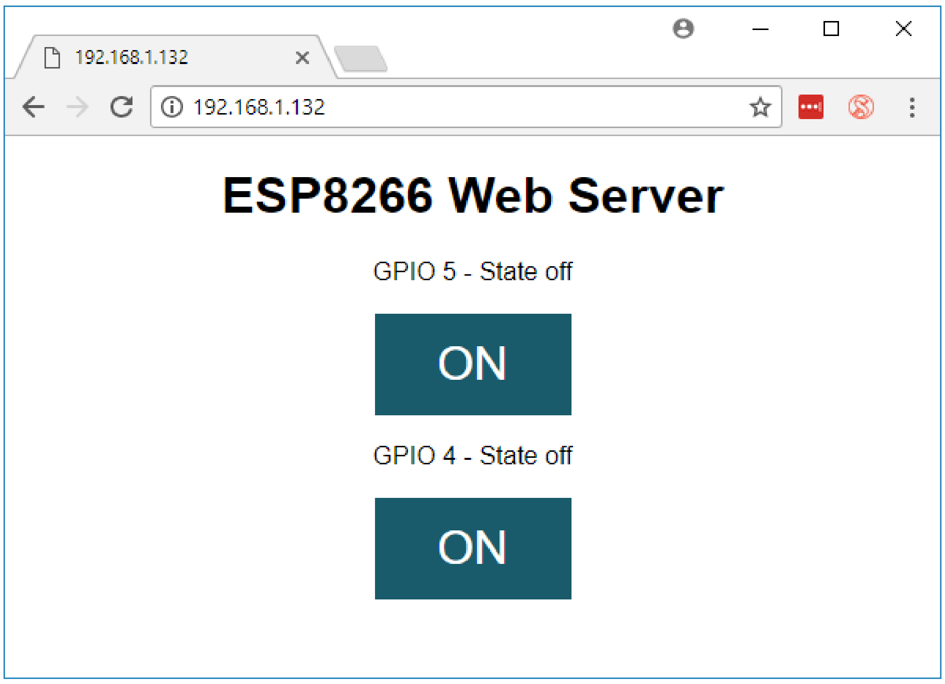
4. Developing an local web server with Node MCU

CODE:

```
// Load Wi-Fi library
#include <ESP8266WiFi.h>

// Replace with your network credentials
const char* ssid     = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output5State = "off";
String output4State = "off";

// Assign output variables to GPIO pins
const int output5 = 5;
const int output4 = 4;

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
  pinMode(output5, OUTPUT);
  pinMode(output4, OUTPUT);
  // Set outputs to LOW
  digitalWrite(output5, LOW);
  digitalWrite(output4, LOW);

  // Connect to Wi-Fi network with SSID and password
  Serial.print("Connecting to ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  // Print local IP address and start web server
  Serial.println("");
  Serial.println("WiFi connected.");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
  server.begin();
}

void loop(){
  WiFiClient client = server.available();   // Listen for incoming clients

  if (client) {                             // If a new client connects,
    Serial.println("New Client.");          // print a message out in the
serial port
    String currentLine = "";                // make a String to hold incoming
data from the client
    currentTime = millis();
    previousTime = currentTime;
    while (client.connected() && currentTime - previousTime <= timeoutTime) {
// loop while the client's connected
      currentTime = millis();
      if (client.available()) {             // if there's bytes to read from
the client,
        char c = client.read();             // read a byte, then
        Serial.write(c);                    // print it out the serial monitor
        header += c;
        if (c == '\n') {                    // if the byte is a newline
character
          // if the current line is blank, you got two newline characters in a
row.
          // that's the end of the client HTTP request, so send a response:
          if (currentLine.length() == 0) {
            // HTTP headers always start with a response code (e.g. HTTP/1.1
200 OK)
            // and a content-type so the client knows what's coming, then a
blank line:
            client.println("HTTP/1.1 200 OK");
            client.println("Content-type:text/html");
            client.println("Connection: close");
            client.println();

            // turns the GPIOs on and off
            if (header.indexOf("GET /5/on") >= 0) {
              Serial.println("GPIO 5 on");
              output5State = "on";
              digitalWrite(output5, HIGH);
            } else if (header.indexOf("GET /5/off") >= 0) {
              Serial.println("GPIO 5 off");
              output5State = "off";
              digitalWrite(output5, LOW);
            } else if (header.indexOf("GET /4/on") >= 0) {
              Serial.println("GPIO 4 on");
              output4State = "on";
              digitalWrite(output4, HIGH);
            } else if (header.indexOf("GET /4/off") >= 0) {
              Serial.println("GPIO 4 off");
              output4State = "off";
              digitalWrite(output4, LOW);
            }

            // Display the HTML web page
            client.println("<!DOCTYPE html><html>");
            client.println("<head><meta name=\"viewport\"
content=\"width=device-width, initial-scale=1\">");
            client.println("<link rel=\"icon\" href=\"data:,\">");
            // CSS to style the on/off buttons
            // Feel free to change the background-color and font-size
attributes to fit your preferences
            client.println("<style>html { font-family: Helvetica; display:
inline-block; margin: 0px auto; text-align: center;}");
            client.println(".button { background-color: #195B6A; border: none;
color: white; padding: 16px 40px;");
            client.println("text-decoration: none; font-size: 30px; margin:
2px; cursor: pointer;}");
            client.println(".button2 {background-color:
#77878A;}</style></head>");

            // Web Page Heading
            client.println("<body><h1>ESP8266 Web Server</h1>");

            // Display current state, and ON/OFF buttons for GPIO 5
            client.println("<p>GPIO 5 - State " + output5State + "</p>");
            // If the output5State is off, it displays the ON button
            if (output5State=="off") {
              client.println("<p><a href=\"/5/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/5/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }

            // Display current state, and ON/OFF buttons for GPIO 4
            client.println("<p>GPIO 4 - State " + output4State + "</p>");
            // If the output4State is off, it displays the ON button
            if (output4State=="off") {
              client.println("<p><a href=\"/4/on\"><button
class=\"button\">ON</button></a></p>");
            } else {
              client.println("<p><a href=\"/4/off\"><button class=\"button
button2\">OFF</button></a></p>");
            }
            client.println("</body></html>");

            // The HTTP response ends with another blank line
            client.println();
            // Break out of the while loop
            break;
          } else { // if you got a newline, then clear currentLine
            currentLine = "";
          }
        } else if (c != '\r') {  // if you got anything else but a carriage
return character,
          currentLine += c;      // add it to the end of the currentLine
        }
      }
    }
    // Clear the header variable
    header = "";
    // Close the connection
    client.stop();
    Serial.println("Client disconnected.");
    Serial.println("");
  }
}
```

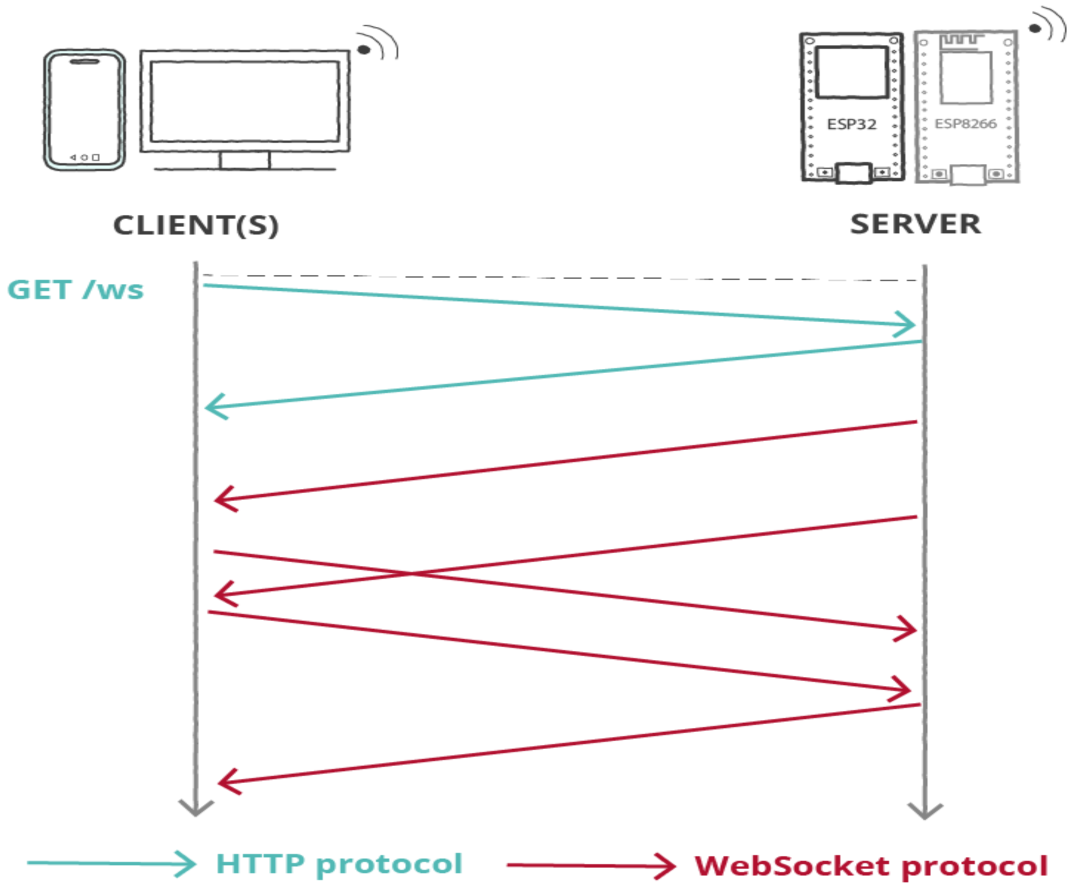
Schematic:




192.168.1.132
192.168.1.132
ESP8266 Web Server
GPIO 5 - State off
ON
GPIO 4 - State off
ON

5. Develop an application to control outputs using web socket server

A WebSocket is a persistent connection between a client and a server that allows bidirectional communication between both parties using a TCP connection. This means you can send data from the client to the server and from the server to the client at any given time.



The client establishes a WebSocket connection with the server through a process known as WebSocket handshake. The handshake starts with an HTTP request/response, allowing servers to handle HTTP connections as well as WebSocket connections on the same port. Once the connection is established, the client and the server can send WebSocket data in full duplex mode.

Using the WebSockets protocol, the server (ESP8266 board) can send information to the client or to all clients without being requested. This also allows us to send information to the web browser when a change occurs.

This change can be something that happened on the web page (you clicked a button) or something that happened on the ESP8266 side like pressing a physical button on a circuit.

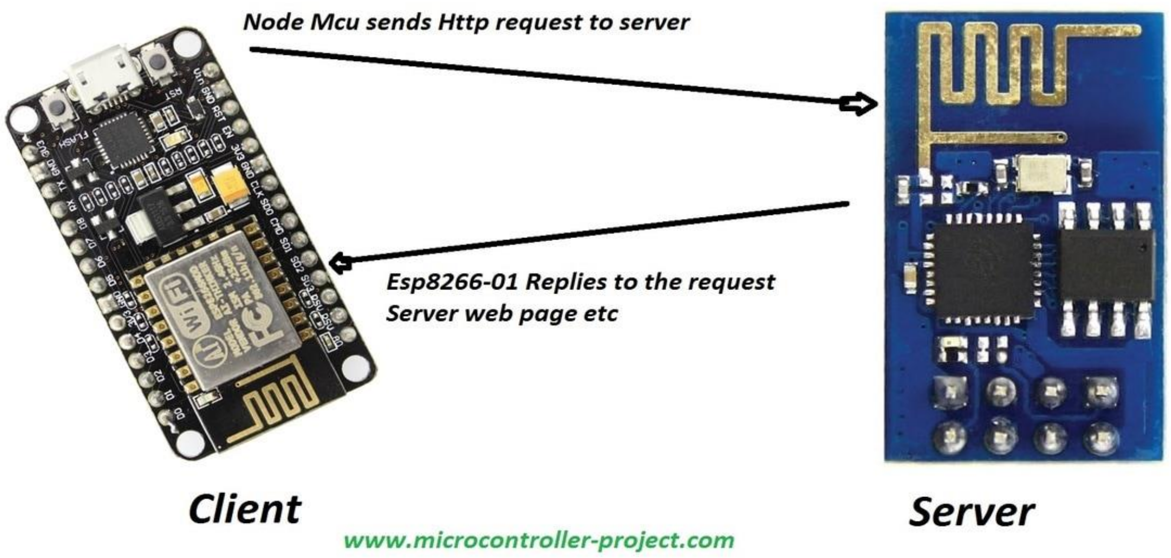
```
// Import required libraries
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

bool ledState = 0;
const int ledPin = 2;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);
AsyncWebSocket ws("/ws");
```

```
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <title>ESP Web Server</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="icon" href="data:,">
  <style>
  html {
    font-family: Arial, Helvetica, sans-serif;
    text-align: center;
  }
  h1 {
    font-size: 1.8rem;
    color: white;
  }
  h2{
    font-size: 1.5rem;
    font-weight: bold;
    color: #143642;
  }
  .topnav {
    overflow: hidden;
    background-color: #143642;
  }
  body {
    margin: 0;
  }
  .content {
    padding: 30px;
    max-width: 600px;
    margin: 0 auto;
  }
  .card {
    background-color: #F8F7F9;;
    box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
    padding-top:10px;
    padding-bottom:20px;
  }
  .button {
    padding: 15px 50px;
    font-size: 24px;
    text-align: center;
    outline: none;
    color: #fff;
    background-color: #0f8b8d;
    border: none;
    border-radius: 5px;
    -webkit-touch-callout: none;
    -webkit-user-select: none;
    -khtml-user-select: none;
    -moz-user-select: none;
    -ms-user-select: none;
    user-select: none;
    -webkit-tap-highlight-color: rgba(0,0,0,0);
   }
   /*.button:hover {background-color: #0f8b8d}*/
   .button:active {
     background-color: #0f8b8d;
     box-shadow: 2 2px #CDCDCD;
     transform: translateY(2px);
   }
   .state {
     font-size: 1.5rem;
     color:#8c8c8c;
     font-weight: bold;
   }
  </style>
<title>ESP Web Server</title>
```

```
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="icon" href="data:,">
</head>
<body>
  <div class="topnav">
    <h1>ESP WebSocket Server</h1>
  </div>
  <div class="content">
    <div class="card">
      <h2>Output - GPIO 2</h2>
      <p class="state">state: <span id="state">%STATE%</span></p>
      <p><button id="button" class="button">Toggle</button></p>
    </div>
  </div>
<script>
  var gateway = `ws://${window.location.hostname}/ws`;
  var websocket;
  window.addEventListener('load', onLoad);
  function initWebSocket() {
    console.log('Trying to open a WebSocket connection...');
    websocket = new WebSocket(gateway);
    websocket.onopen    = onOpen;
    websocket.onclose   = onClose;
    websocket.onmessage = onMessage; // <-- add this line
  }
  function onOpen(event) {
    console.log('Connection opened');
  }
  function onClose(event) {
    console.log('Connection closed');
    setTimeout(initWebSocket, 2000);
  }
  function onMessage(event) {
    var state;
    if (event.data == "1"){
      state = "ON";
    }
    else{
      state = "OFF";
    }
    document.getElementById('state').innerHTML = state;
  }
  function onLoad(event) {
    initWebSocket();
    initButton();
  }
  function initButton() {
    document.getElementById('button').addEventListener('click', toggle);
  }
  function toggle(){
    websocket.send('toggle');
  }
</script>
</body>
</html>
)rawliteral";

void notifyClients() {
  ws.textAll(String(ledState));
}

void handleWebSocketMessage(void *arg, uint8_t *data, size_t len) {
  AwsFrameInfo *info = (AwsFrameInfo*)arg;
  if (info->final && info->index == 0 && info->len == len && info->opcode ==
WS_TEXT) {
    data[len] = 0;
    if (strcmp((char*)data, "toggle") == 0) {
      ledState = !ledState;
      notifyClients();
    }
  }
}

void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,
AwsEventType type,
             void *arg, uint8_t *data, size_t len) {
    switch (type) {
      case WS_EVT_CONNECT:
        Serial.printf("WebSocket client #%u connected from %s\n", client-
>id(), client->remoteIP().toString().c_str());
        break;
      case WS_EVT_DISCONNECT:
        Serial.printf("WebSocket client #%u disconnected\n", client->id());
        break;
      case WS_EVT_DATA:
        handleWebSocketMessage(arg, data, len);
        break;
      case WS_EVT_PONG:
      case WS_EVT_ERROR:
        break;
  }
}

void initWebSocket() {
  ws.onEvent(onEvent);
  server.addHandler(&ws);
}

String processor(const String& var){
  Serial.println(var);
  if(var == "STATE"){
    if (ledState){
      return "ON";
    }
    else{
      return "OFF";
    }
  }
  return String();
}

void setup(){
  // Serial port for debugging purposes
  Serial.begin(115200);

  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW);

  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi..");
  }

  // Print ESP Local IP Address
  Serial.println(WiFi.localIP());

  initWebSocket();

  // Route for root / web page
  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
    request->send_P(200, "text/html", index_html, processor);
  });

  // Start server
  server.begin();
}

void loop() {
  ws.cleanupClients();
  digitalWrite(ledPin, ledState);
}
```

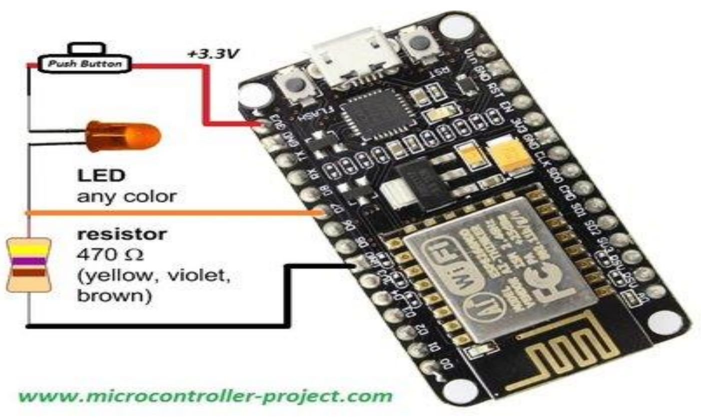
6.To establish a Wi-Fi communication (HTTP) between two ESP8266 Node MCU boards to exchange data without the need to connect to the internet

1. Send a HTTP request from client to server and the server upon receiving the particular request will perform the desired action, manipulate its Inputs Outputs etc. Consider this method as you are sending an HTTP request(From esp8266) containing data to a website(Server-Host esp8266) and the website than displays the data on a web page(Esp8266 Manipulate I/O Pins).


Node Mcu sends Http request to server
Esp8266-01 Replies to the request
Server web page etc
Client
Server
www.microcontroller-project.com

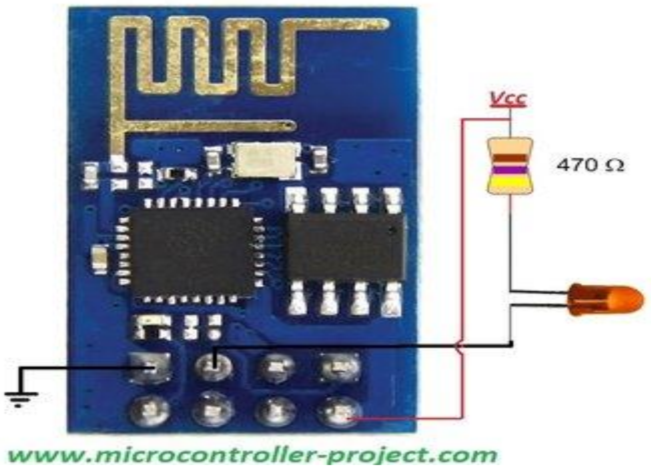
Esp8266 Client side Circuit Diagram

At client side NodeMCU we connected a push button with Pin#D7. Pin#D7 is used as input pin. A push button is connected in series with a led and a resistor. Resistor is grounded and led is at active high side with push button in series. Active high side is supplied +3.3 volt from the NodeMCU lua module. Resistor is also grounded with the NodeMCU module ground. Pin#D7 normally remains at low level. But whenever push button is pressed the led lights up and the D7 logic level changes to high.
What i want is whenever the push button(At client) is pressed the led connected at the server side(esp8266-01) toggles. So the circuit at the right side is made to achieve this logic. NodeMCU is powered through PC USB plug. You can remove the Led from the circuit is there are power issues but it worked for me and i have no issue with the same configuration.



Esp8266 Server Side Circuit Diagram

At sever side led is connected to Pin#2 of esp8266-01 module. Esp8266-01 has only two GPIO’s(General Purpose Input Output Pins) GPIO-0 and GPIO-2. I choose GPIO-2 because GPIO-1 is used during uploading code. A 470 ohm resistor is connected in series to led. Circuit is powered through the same power that we apply to Vcc of esp8266 module. Esp8266 is a 3.3 volt chip do not power it with more than 4 volts. The Vcc of resistor is connected to Vcc of esp8266.



Server Code:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

const char* ssid = "ESP_D54736";  // SSID of esp8266
//const char* password = "123";   //
bool toggle=0;                    //Variable to switch on and off the solenoid
ESP8266WebServer server(80);      //Specify port for TCP connection

void handleRoot() {
  toggle=!toggle;                 //Toggling Led Variable
    digitalWrite(2,toggle);       //Toggle Led
        String s = "\r\n\r\n  <!DOCTYPE HTML>\r\n<html><h1>Esp8266
Communication</h1> ";
        s += "<p>Success!!!</html>\r\n\r\n";
        server.send(200,"text/html",s);      //Reply to the client
}

void setup() {
  delay(200);                             //Stable Wifi
  Serial.begin(115200);                   //Set Baud Rate
  pinMode(2, OUTPUT);                     //Led/Solenoid at pin 2
  WiFi.softAP(ssid);//, password);        //In Access Point Mode

  IPAddress myIP = WiFi.softAPIP();       //Check the IP assigned. Put this Ip
in the client host.
  Serial.print("AP IP address: ");
  Serial.println(myIP);           //Print the esp8266-01 IP(Client must also be
on the save IP series)
  server.on("/Led", handleRoot);             //Checking client connection
  server.begin();                         // Start the server
  Serial.println("Server started");
}

void loop() {
  // Check if a client has connected. On first connection switch on the
Solenoid on next switch off.
```

```
  server.handleClient();
}
```

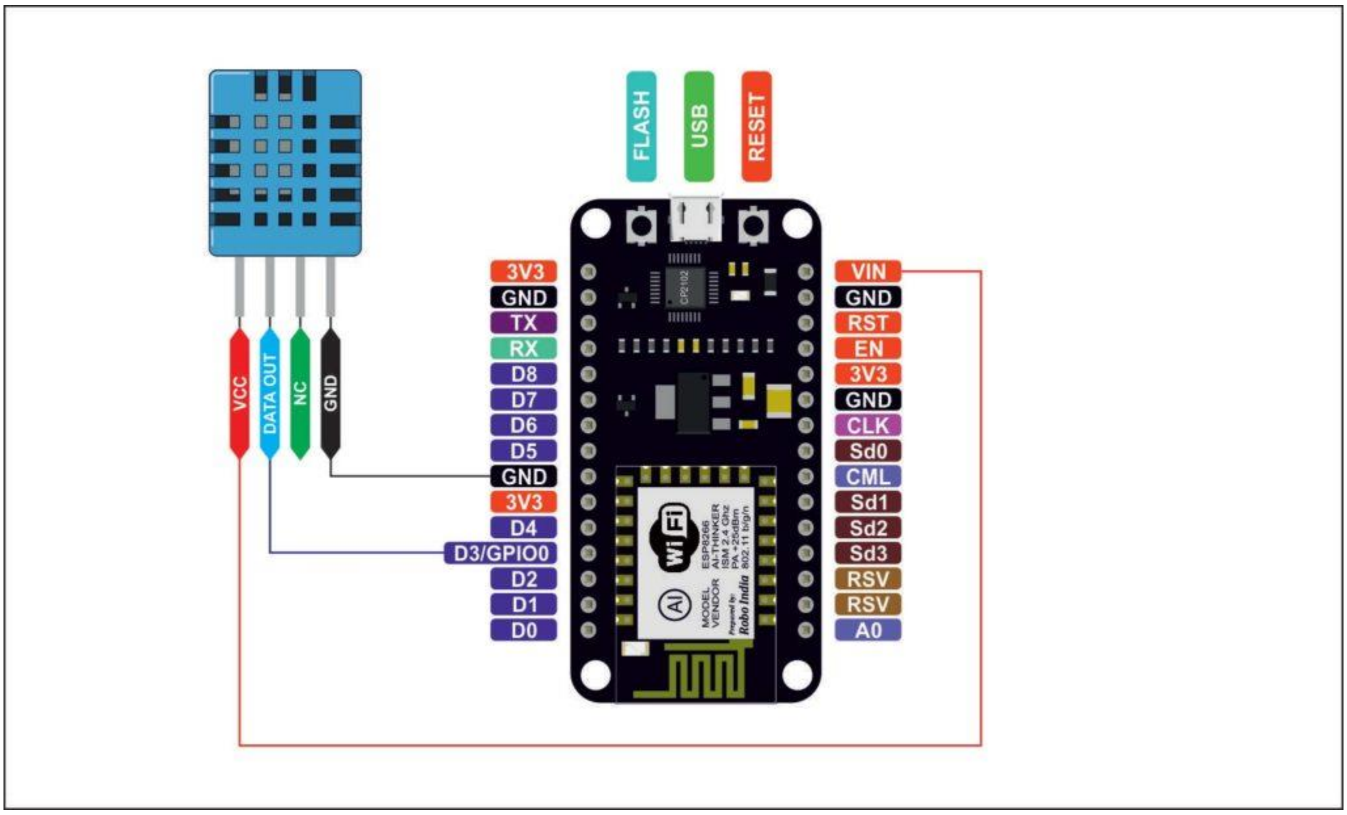
Client Code:

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
//Ip of the Host(Our Case esp8266-01 as server. Its the ip of the esp8266-01
as Access point)
const char* host = "192.168.4.1";

void setup() {
  Serial.begin(115200);          //Baud Rate for Communication
  delay(10);                     //Baud rate prper initialization
  pinMode(13,INPUT);             //Pin D7 on NodeMcu Lua. Button to switch on
and off the solenoid.
  WiFi.mode(WIFI_STA);           //NodeMcu esp12E in station mode
  WiFi.begin("ESP_D54736");      //Connect to this SSID. In our case esp-01
SSID.

  while (WiFi.status() != WL_CONNECTED) {      //Wait for getting IP assigned
by Access Point/ DHCP.
  //Our case  esp-01 as Access point will assign IP to nodemcu esp12E.
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());             //Check out the Ip assigned by
the esp12E
}

void loop() {

         if(digitalRead(13)==1){    //Check out if Pin D7/13 is high. If high
activate
                                    //the led connected to esp-01. On next
press deactivate it.
              Serial.print("connecting to ");
              Serial.println(host);
              // Use WiFiClient class to create TCP connections
              WiFiClient client;
              const int httpPort = 80;
                if (!client.connect("192.168.4.1", httpPort)) {
                  Serial.println("connection failed");
                  return;
                   }
              //Request to server to activate the led
              client.print(String("GET ") +"/Led"+" HTTP/1.1\r\n" +
                           "Host: " + host + "\r\n" +
                           "Connection: close\r\n\r\n");
              delay(10);
              // Read all the lines of the reply from server and print them to
Serial Monitor etc
              while(client.available()){
                String line = client.readStringUntil('\r');
                Serial.print(line);
              }
              //Close the Connection. Automatically
              Serial.println();
              Serial.println("closing connection");
            }//End if

}//End Loop
```

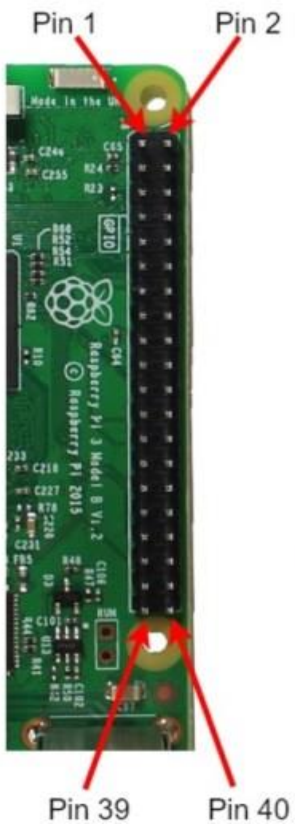
7.Interface any Sensor with Node MCU and load the Sensor data into cloud

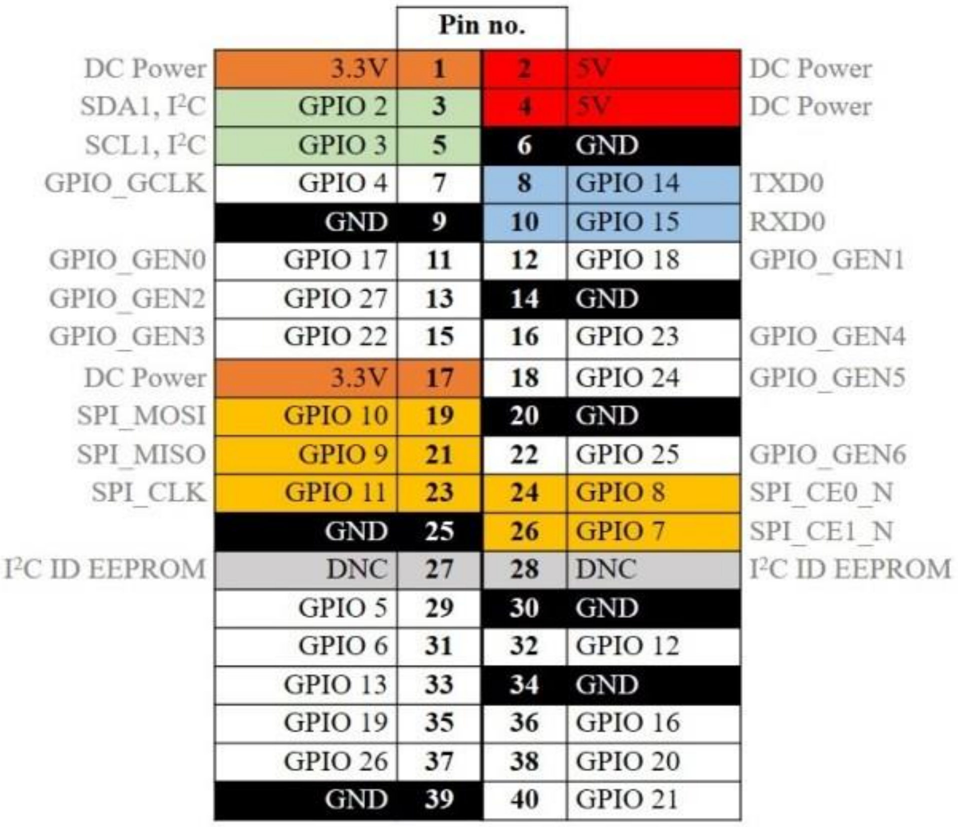
```
#include <DHT.h>  // Including library for dht

#include <ESP8266WiFi.h>

String apiKey = "Your API of thingsspeak";     //  Enter your Write API key
from ThingSpeak

const char *ssid =  "Your wifi Network name";     // replace with your wifi
ssid and wpa2 key
const char *pass =  "Network password";
const char* server = "api.thingspeak.com";

#define DHTPIN 0          //pin where the dht11 is connected

DHT dht(DHTPIN, DHT11);

WiFiClient client;

void setup()
{
       Serial.begin(115200);
       delay(10);
       dht.begin();

       Serial.println("Connecting to ");
       Serial.println(ssid);


       WiFi.begin(ssid, pass);

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

}

void loop()
{

      float h = dht.readHumidity();
      float t = dht.readTemperature();

              if (isnan(h) || isnan(t))
                 {
                     Serial.println("Failed to read from DHT sensor!");
                      return;
                 }

                         if (client.connect(server,80))   //
"184.106.153.149" or api.thingspeak.com
                      {

                             String postStr = apiKey;
                             postStr +="&field1=";
                             postStr += String(t);
                             postStr +="&field2=";
                             postStr += String(h);
                             postStr += "\r\n\r\n";

                             client.print("POST /update HTTP/1.1\n");
                             client.print("Host: api.thingspeak.com\n");
                             client.print("Connection: close\n");
                             client.print("X-THINGSPEAKAPIKEY: "+apiKey+"\n");
                             client.print("Content-Type: application/x-www-
form-urlencoded\n");
                             client.print("Content-Length: ");
                             client.print(postStr.length());
                             client.print("\n\n");
                             client.print(postStr);

                             Serial.print("Temperature: ");
                             Serial.print(t);
                             Serial.print(" degrees Celcius, Humidity: ");
                             Serial.print(h);
                             Serial.println("%. Send to Thingspeak.");
                        }
          client.stop();

          Serial.println("Waiting...");

  // thingspeak needs minimum 15 sec delay between updates, i've set it to 30
seconds
  delay(10000);
}
```

Circuit Layout :



8. Study of Connectivity and configuration of Raspberry-Pi circuit with basic peripherals, LEDS. Understanding GPIO and its use in program.



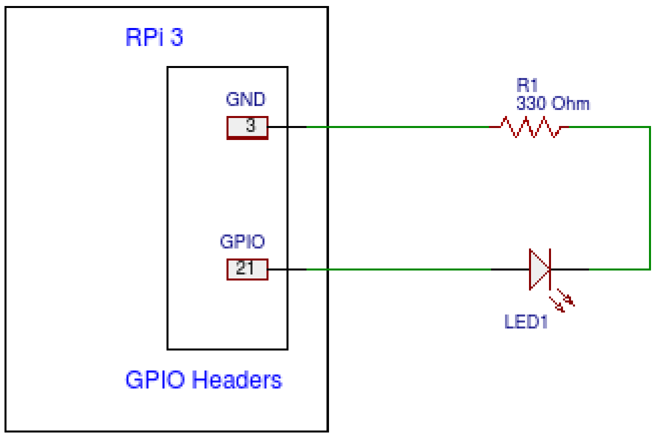


		Pin no.			
DC Power	3.3V	1	2	5V	DC Power
SDA1, I²C	GPIO 2	3	4	5V	DC Power
SCL1, I²C	GPIO 3	5	6	GND	
GPIO_GCLK	GPIO 4	7	8	GPIO 14	TXD0
	GND	9	10	GPIO 15	RXD0
GPIO_GEN0	GPIO 17	11	12	GPIO 18	GPIO_GEN1
GPIO_GEN2	GPIO 27	13	14	GND	
GPIO_GEN3	GPIO 22	15	16	GPIO 23	GPIO_GEN4
DC Power	3.3V	17	18	GPIO 24	GPIO_GEN5
SPI_MOSI	GPIO 10	19	20	GND	
SPI_MISO	GPIO 9	21	22	GPIO 25	GPIO_GEN6
SPI_CLK	GPIO 11	23	24	GPIO 8	SPI_CE0_N
	GND	25	26	GPIO 7	SPI_CE1_N
I²C ID EEPROM	DNC	27	28	DNC	I²C ID EEPROM
	GPIO 5	29	30	GND	
	GPIO 6	31	32	GPIO 12	
	GPIO 13	33	34	GND	
	GPIO 19	35	36	GPIO 16	
	GPIO 26	37	38	GPIO 20	
	GND	39	40	GPIO 21	

Code:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(21,GPIO.OUT)
print "LED on"
GPIO.output(21,GPIO.HIGH)
time.sleep(10)
print "LED off"
GPIO.output(21,GPIO.LOW)
```

Hardware Schematic:



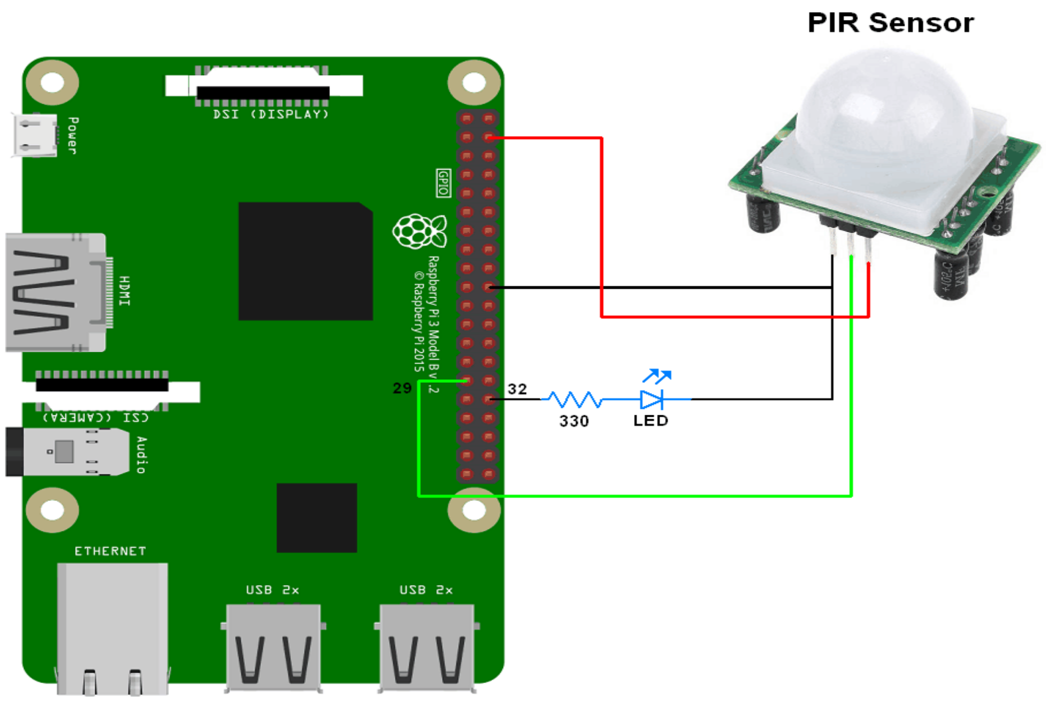
9. Understanding the connectivity of Raspberry-Pi board circuit with temperature sensor. Write an application to read the environment temperature and load the data

```
import thingspeak
import time
import Adafruit_DHT

channel_id = 206897 # PUT CHANNEL ID HERE
write_key  = '24GJQV17H7H4XGJ5' # PUT YOUR WRITE KEY HERE
read_key   = '9EZ7E0918UVVAGAY' # PUT YOUR READ KEY HERE
pin = 4
sensor = Adafruit_DHT.DHT22

def measure(channel):
    try:
        humidity, temperature = Adafruit_DHT.read_retry(sensor, pin)
        # write
        response = channel.update({'field1': temperature, 'field2': humidity})

        # read
        read = channel.get({})
        print("Read:", read)

    except:
        print("connection failed")


if __name__ == "__main__":
    channel = thingspeak.Channel(id=channel_id, write_key=write_key, api_key=read_key)
    while True:
        measure(channel)
        # free account has an api limit of 15sec
        time.sleep(15)
```

10. Understanding the connectivity of Raspberry-Pi circuit with IR sensor. Write an application to detect obstacle and notify user using LEDs.

```
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)
```

Hardware Schematic:



11. Understanding and connectivity of Raspberry-Pi with camera. Write an application to capture image/Video over HTTP with Raspberry Pi Camera

Python Program for Image Capture

```
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()
```

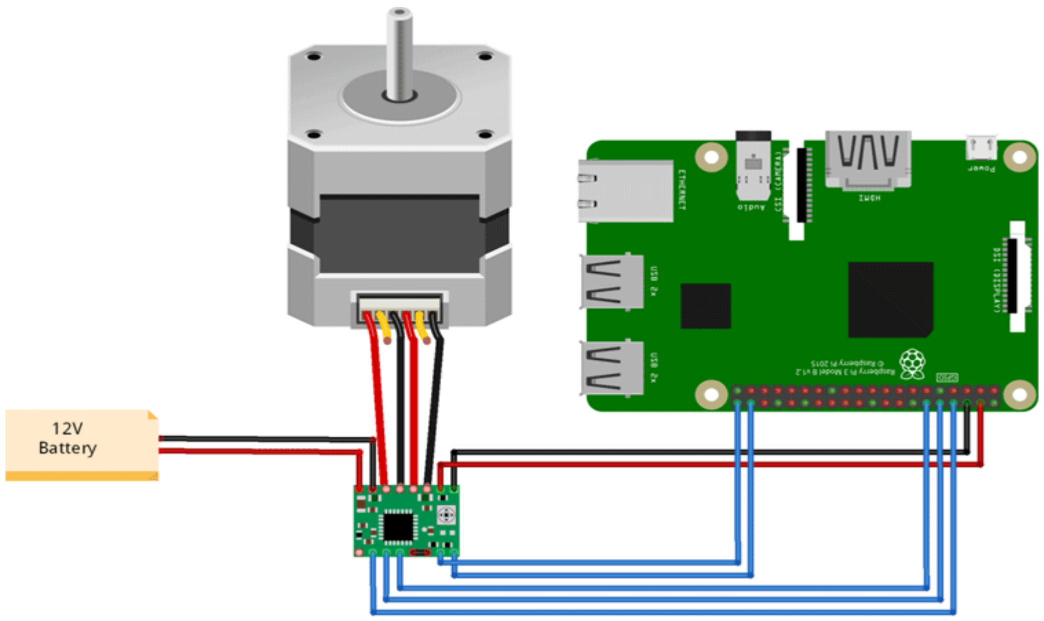
Python Program for Video Recording

```
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")
```

12. Write an application using Raspberry-Pi to control the operation of stepper motor and controlling using it web.

```
import RPi.GPIO as GPIO
from RpiMotorLib import RpiMotorLib

#define GPIO pins
GPIO_pins = (14, 15, 18) # Microstep Resolution MS1-MS3 -> GPIO Pin
direction= 20       # Direction -> GPIO Pin
step = 21      # Step -> GPIO Pin

# Declare an named instance of class pass GPIO pins numbers
mymotortest = RpiMotorLib.A4988Nema(direction, step, GPIO_pins, "A4988")

app = Flask(__name__)


#HTML Code

TPL = '''
<html>
     <img src="https://iotdesignpro.com/sites/default/files/Iot%20Design%20Pro%20Logo_0.png" alt="">
    <head><title>Web Page Controlled Stepper</title></head>
    <body>
    <h2> Web Page to Control Stepper</h2>
        <form method="POST" action="test">
            <h5> Use the slider to rotate Stepper Clockwise & Counter-Clockwise  </h5>
            <p>Slider   <input type="range" min="1" max="20" name="slider" /> </p>
            <input type="submit" value="submit" />
        </form>
    </body>
</html>


'''

@app.route("/")
def home():

    return render_template_string(TPL)

@app.route("/test", methods=["POST"])
def test():
    # Get slider Values
    slider = request.form["slider"]
    print(int(slider))

    if (int(slider)>10):
       mymotortest.motor_go(True, "Full" , 600,int(slider)*.0004, False, .05)
       print("Rotating Clockwise")

    if (int(slider)<10):
       mymotortest.motor_go(False, "Full" , 600,int(slider)*.001, False, .05)
       print("Rotating Anti-Clockwise")


    return render_template_string(TPL)

# Run the app on the local development server
if __name__ == "__main__":
    app.run()
```

Hardware Schematic:

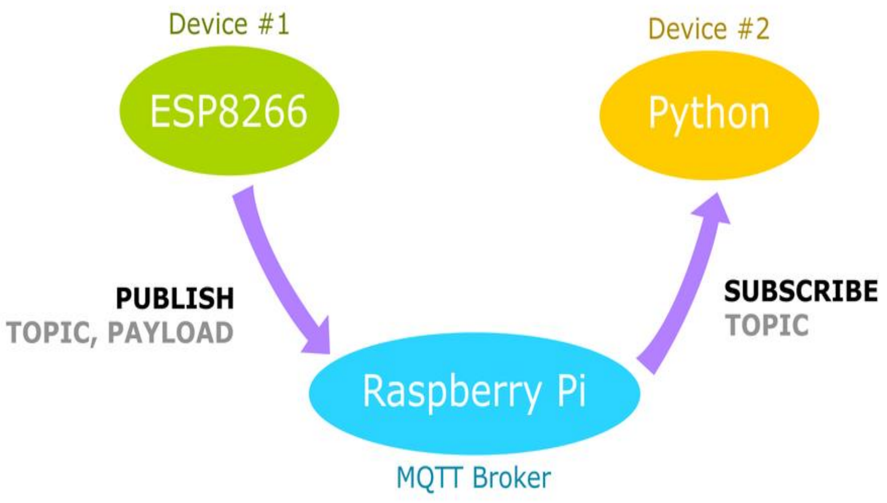


13.To establish a communication between a Raspberry Pi running the Node-RED software and an ESP8266 using MQTT

Equipment Required

1 x Raspberry Pi, connected to a local network (running Jessie)
1 x ESP8266 Module (Adafruit HUZZAH)
1 x Breadboard
3 x Jumper Wires (Male-to-Male)
1 x Pushbutton
1 x 10k Ohm Resistor (Brown-Black-Orange colour code)

Step 1: What Is MQTT?



MQTT, or MQ Telemetry Transport, is a messaging protocol which allows multiple devices to talk to each other. Currently, it is a popular protocol for the Internet of Things, although it has been used for other purposes - for example, Facebook Messenger. Interestingly MQTT was invented in 1999 - meaning it's as old as me!

MQTT is based around the idea that devices can publish or subscribe to topics. So, for example. If Device #1 has recorded the temperature from one of its sensors, it can publish a message which contains the temperature value it recorded, to a topic (e.g. "Temperature"). This message is sent to an MQTT Broker, which you can think of as a switch/router on a local area network. Once the MQTT Broker has received the message, it will send it to any devices (in this case, Device #2) which are subscribed to the same topic.

In this project, we will be publishing to a topic using an ESP8266, and creating a Python script that will subscribe to this same topic, via a Raspberry Pi which will act as the MQTT Broker. The great thing about MQTT is that it is lightweight, so it perfect for running on small microcontrollers such as an ESP8266, but it is also widely available - so we can run it on a Python script as well.

Step 2: Installing the MQTT Broker on the Raspberry Pi

To setup our MQTT system, we need a broker, as explained in the previous step. For the Raspberry Pi, we will be using the "Mosquitto" MQTT broker. Before we install this, it is always best to update our Raspberry Pi.

```
sudo apt-get update
sudo apt-get upgrade
```

Once you've done this, install mosquitto and then the mosquitto-clients packages.

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

When you've finished installing these two packages, we are going to need to configure the broker. The mosquitto broker's configuration file is located at /etc/mosquitto/mosquitto.conf, so open this with your favourite text editor. If you don't have a favourite text editor or don't know how to use any of the command line editors, I'll be using nano so you can follow along:

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

At the bottom of this file, you should see the line:

```
include_dir /etc/mosquitto/conf.d
```

Delete this line. Add the following lines to the bottom of the file.

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

By typing those lines, we've told mosquitto that we don't want anyone connecting to our broker who doesn't supply a valid username and password (we'll get on to set these in a second) and that we want mosquitto to listen for messages on port number 1883.

If you don't want the broker to require a username and password, don't include the first two lines that we added (i.e. allow_anonymous... and password_file...). If you have done this, then skip to rebooting the Raspberry Pi.

Now close (and save) that file. If you are following along with the nano example, press CTRL+X, and type Y when prompted.

Because we've just told mosquitto that users trying to use the MQTT broker need to be authenticated, we now need to tell mosquitto what the username and password are! So, type the following command - replacing username with the username that you would like - then enter the password you would like when prompted (Note: if, when editing the configuration file, you specified a different password_file path, replace the path below with the one you used).

```
sudo mosquitto_passwd -c /etc/mosquitto/pwfile username
```

As we've just changed the mosquitto configuration file, we should reboot the Raspberry Pi.

```
sudo reboot
```

Once the Raspberry Pi has finished rebooting, you should have a fully functioning MQTT broker!

Step 3: Testing the Broker

Once you've installed mosquitto on the Raspberry Pi, you can give it a quick test - just to make sure everything is working correctly. For this purpose, there are two commands that we can use on the command line. mosquitto_pub and mosquitto_sub. In this step, I will guide you through using each of these to test our broker.

In order to test the broker, you will need to open two command line windows. If you are using Putty or another SSH client, this is as simple as opening another SSH window and logging in as usual. If you are accessing your Pi from a UNIX terminal, this is exactly the same. If you are using the Raspberry Pi directly, you will need to open two terminal windows in the GUI mode (the command startxcan be used to start the GUI).

Now that you have opened two windows, we can get started on the testing. In one of the two terminals, type the following command, replacing username and password with the ones you setup in the previous step.

```
mosquitto_sub -d -u username -P password -t test
```

If you decided not to set a username and password in the previous step, then from now on, ignore the -u and -P flags in the commands. So, as an example, the mosquitto_sub command would now be:

```
mosquitto_sub -d -t test
```

The mosquitto_sub command will subscribe to a topic, and display any messages that are sent to the specified topic in the terminal window. Here, -d means debug mode, so all messages and activity will be output on the screen. -u and -P should be self-explanatory. Finally, -t is the name of the topic we want to subscribe to - in this case, "test".

Next, in the other terminal window, we are going to try and publish a message to the "test" topic. Type the following, remembering again to change username and password:

```
mosquitto_pub -d -u username -P password -t test -m "Hello, World!"
```

When you press enter, you should see your message "Hello, World!" appear in the first terminal window we used (to subscribe).

Step 4: Programming the ESP8266

Now we will begin to program the ESP8266, but before we can start, you will need to install the following libraries in the Arduino Library manager (Sketch->Include Libraries->Manage Libraries)

Bounce2
PubSubClient

Once you've installed those libraries, you will be able to run the code I've included in this Instructable (MQTT_Publish.zip). I've made sure to comment it so that you can understand what each section is doing, and this should hopefully enable you to adapt it to your needs.

Remember to change the constants at the top of the code so that your ESP8266 can connect to your WiFi network and your MQTT Broker (the Raspberry Pi).

```
"""
Python MQTT Subscription client
Thomas Varnish (https://github.com/tvarnish), (https://www.instructables.com/member/Tango172)
Written for my Instructable - "How to use MQTT with the Raspberry Pi and ESP8266"
"""
import paho.mqtt.client as mqtt

# Don't forget to change the variables for the MQTT broker!
mqtt_username = "MQTT Username"
mqtt_password = "MQTT Password"
mqtt_topic = "Your Topic"
mqtt_broker_ip = "Your Broker IP"

client = mqtt.Client()
# Set the username and password for the MQTT client
client.username_pw_set(mqtt_username, mqtt_password)

# These functions handle what happens when the MQTT client connects
# to the broker, and what happens then the topic receives a message
def on_connect(client, userdata, flags, rc):
    # rc is the error code returned when connecting to the broker
    print "Connected!", str(rc)

    # Once the client has connected to the broker, subscribe to the topic
    client.subscribe(mqtt_topic)

def on_message(client, userdata, msg):
    # This function is called everytime the topic is published to.
    # If you want to check each message, and do something depending on
    # the content, the code to do this should be run in this function

    print "Topic: ", msg.topic + "\nMessage: " + str(msg.payload)

    # The message itself is stored in the msg variable
    # and details about who sent it are stored in userdata

# Here, we are telling the client which functions are to be run
# on connecting, and on receiving a message
client.on_connect = on_connect
client.on_message = on_message

# Once everything has been set up, we can (finally) connect to the broker
# 1883 is the listener port that the MQTT broker is using
client.connect(mqtt_broker_ip, 1883)

# Once we have told the client to connect, let the client object run itself
client.loop_forever()
client.disconnect()
```

```
/*
 * ESP8266 (Adafruit HUZZAH) Mosquitto MQTT Publish Example
 * Thomas Varnish (https://github.com/tvarnish), (https://www.instructables.com/member/Tango172)
 * Made as part of my MQTT Instructable - "How to use MQTT with the Raspberry Pi and ESP8266"
 */
#include <Bounce2.h> // Used for "debouncing" the pushbutton
#include <ESP8266WiFi.h> // Enables the ESP8266 to connect to the local network (via WiFi)
#include <PubSubClient.h> // Allows us to connect to, and publish to the MQTT broker

const int ledPin = 0; // This code uses the built-in led for visual feedback that the button has been pressed
const int buttonPin = 13; // Connect your button to pin #13

// WiFi
// Make sure to update this for your own WiFi network!
const char* ssid = "Your SSID";
const char* wifi_password = "Your WiFi Password";

// MQTT
// Make sure to update this for your own MQTT Broker!
const char* mqtt_server = "MQTT Broker IP Address";
const char* mqtt_topic = "Your Topic";
const char* mqtt_username = "MQTT Username";
const char* mqtt_password = "MQTT Password";
// The client id identifies the ESP8266 device. Think of it a bit like a hostname (Or just a name, like Greg).
const char* clientID = "Client ID";

// Initialise the Pushbutton Bouncer object
Bounce bouncer = Bounce();
```

```
// Initialise the WiFi and MQTT Client objects
WiFiClient wifiClient;
PubSubClient client(mqtt_server, 1883, wifiClient); // 1883 is the listener port for the Broker

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);

  // Switch the on-board LED off to start with
  digitalWrite(ledPin, HIGH);

  // Setup pushbutton Bouncer object
  bouncer.attach(buttonPin);
  bouncer.interval(5);

  // Begin Serial on 115200
  // Remember to choose the correct Baudrate on the Serial monitor!
  // This is just for debugging purposes
  Serial.begin(115200);

  Serial.print("Connecting to ");
  Serial.println(ssid);

  // Connect to the WiFi
  WiFi.begin(ssid, wifi_password);

  // Wait until the connection has been confirmed before continuing
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  // Debugging - Output the IP Address of the ESP8266
  Serial.println("WiFi connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  // Connect to MQTT Broker
  // client.connect returns a boolean value to let us know if the connection was successful.
  // If the connection is failing, make sure you are using the correct MQTT Username and Password (Setup
Earlier in the Instructable)
  if (client.connect(clientID, mqtt_username, mqtt_password)) {
    Serial.println("Connected to MQTT Broker!");
  }
  else {
    Serial.println("Connection to MQTT Broker failed...");
  }

}

void loop() {
  // Update button state
  // This needs to be called so that the Bouncer object can check if the button has been pressed
  bouncer.update();

  if (bouncer.rose()) {
    // Turn light on when button is pressed down
    // (i.e. if the state of the button rose from 0 to 1 (not pressed to pressed))
    digitalWrite(ledPin, LOW);

    // PUBLISH to the MQTT Broker (topic = mqtt_topic, defined at the beginning)
    // Here, "Button pressed!" is the Payload, but this could be changed to a sensor reading, for example.
    if (client.publish(mqtt_topic, "Button pressed!")) {
      Serial.println("Button pushed and message sent!");
    }
    // Again, client.publish will return a boolean value depending on whether it succeded or not.
    // If the message failed to send, we will try again, as the connection may have broken.
    else {
      Serial.println("Message failed to send. Reconnecting to MQTT Broker and trying again");
      client.connect(clientID, mqtt_username, mqtt_password);
      delay(10); // This delay ensures that client.publish doesn't clash with the client.connect call
      client.publish(mqtt_topic, "Button pressed!");
    }
  }
  else if (bouncer.fell()) {
    // Turn light off when button is released
    // i.e. if state goes from high (1) to low (0) (pressed to not pressed)
    digitalWrite(ledPin, HIGH);
  }
}
```

14. Develop a simple home automation system with Node MCU/Raspberry Pi

```
#include "arduino_secrets.h"
/*
  Sketch generated by the Arduino IoT Cloud
  https://create.arduino.cc/cloud/things/86fc6944-21a5-4447-a980-125dc1b15276

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and updated when changes are made to the Thing

  bool light1;
  bool light2;
  bool light3;
  bool light4;

  Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this sketch.
*/

#include "thingProperties.h"

void setup() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  pinMode (19, OUTPUT);
  pinMode (21, OUTPUT);
  pinMode (22, OUTPUT);
  pinMode (23, OUTPUT);

  digitalWrite (19, HIGH);
  digitalWrite (21, HIGH);
  digitalWrite (22, HIGH);
  digitalWrite (23, HIGH);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  /*
     The following function allows you to obtain more information
     related to the state of network and IoT Cloud connection and errors
     the higher number the more granular information you'll get.
     The default is 0 (only errors).
     Maximum is 4
 */
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void loop() {
  ArduinoCloud.update();
  // Your code here


}

void onLight1Change()
{
  if (light1==1)
  {
    digitalWrite (19, LOW);
  }
  else
  {
    digitalWrite (19, HIGH);
  }
}

void onLight2Change()
{
  if (light2==1)
  {
    digitalWrite (21, LOW);
  }
  else
  {
    digitalWrite (21, HIGH);
  }
}

void onLight3Change()
{
  if (light3==1)
  {
    digitalWrite (22, LOW);
  }
  else
  {
    digitalWrite (22, HIGH);
  }
}

void onLight4Change()
{
  if (light4==1)
  {
    digitalWrite (23, LOW);
  }
  else
  {
    digitalWrite (23, HIGH);
  }
}
```