M5Stack CM4Stack Guide.

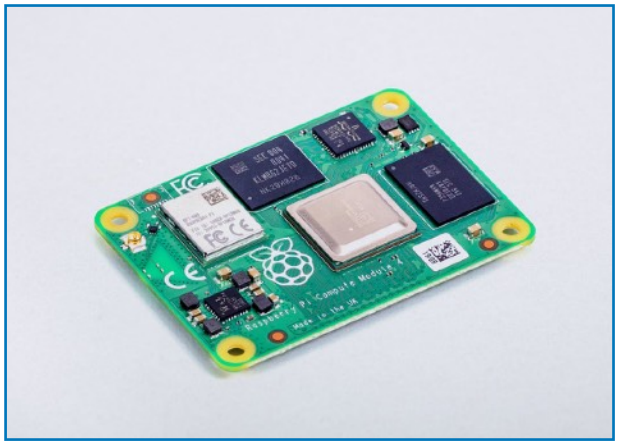
Table of Contents

Getting started with the CM4 Stack Development Kit.

Written by Adam Bryant © 2023

Introduction

The CM4 Stack Development is the latest of the controller line to be released but unlike regular M5Stack controllers, The CM4 Stack is not based on and ESP32 microcontroller but is built around the CM4 Module from Raspberry Pi Foundation.



Unlike the RP CM4, the CM4 Stack come prebuilt needing only a monitor, Keyboard and mouse in order to get started.

CM4 Stack Development Kit Features

From the back of the packaging, the features are as follows:

- CM4104032 module with the following features:
  - Quad Core Cortex A72 @1.5GHz,
  - 4GB Ram,
  - 32 GB eMMC,
  - 2.4GHz/5GHz WIFI and BLE,

- 2.0” IPS Colour LCD 240 X 320 pixel resolution,
- Capacitive multi-Touch,
- ATECC608B Crypto Chip,
- 2W Speaker.
-
- The CM4 Stack has the following connectors on the outside:
- 1 X Gigabit Ethernet,
- 2 X USB 3.2 (Type A)
- 1 X USB 2.0 OTG (type C)
- 1X HDMI Display,
- 2 X grove Ports (I2C and UART)

Power is provided via the included DC 12 V 3A adapter or via the USBC port using a 5V 3A adapter.

Power supply

The CM4 Stack can be powered through the barrel jack with a 12V @ 3Ah supply of through the USBC OTG port with a 5V @3Ah supply.

Storage

As mentioned above, the CM4 Stack has a 32GB eMMC chip soldered to the board and no SD Card slot meaning that in order to upgrade the storage you will need to use and external USB drive.

Opening the Packaging

When you receive your CM4 Stack development kit and open the box you will find the following items inside:

- The CM4 Stack Development Kit,
- 12V Power supply,
- Double sided information card,
- Pack of spare mounting fixings.

Exploring the Outside

When you Open the CM4 Stacks box, the first thing you will see is the 240X320px (2.0”) screen and on the bottom of the screen is a little red circle that is a dedicated touch zone for operating the the OS menu when a screen and keyboard is connected.

On the bottom you will find the Gigabit Ethernet port along with the 12V DC power jack which uses a plug with a 5.5mm OD 2.1mm ID jack.

On the left hand side you will find the Boot select switch and 2X USB A 3.2 ports.

On the top you will find the full sized HDMI outport for connecting external HDMI monitors and screens.

And on the right hand side you will find the USB C OTG, HY2.0 4P I2C Grove and HY2.0 4P UART Grove port

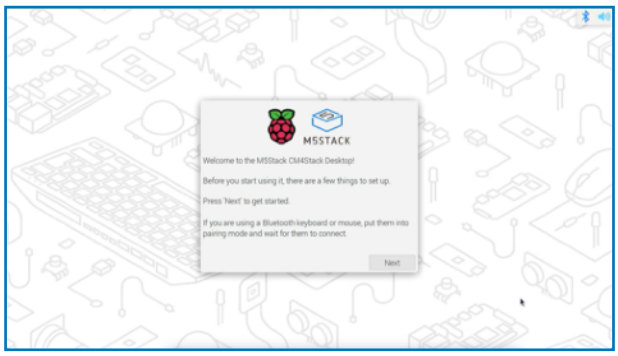
Underneath you will find the connectors for wall mounting or Din rail mounting and you will also see the extractor fan used to cool the CM4 module from over heating.

Connecting the Hardware

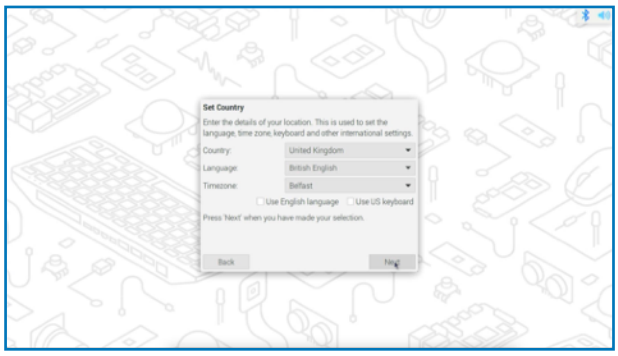
In order to get started all we need is to connect a Key board and mouse into the USB ports and an HDMI monitor into the HDMI port on the top of the CM4Stack development kit.

First Power Up

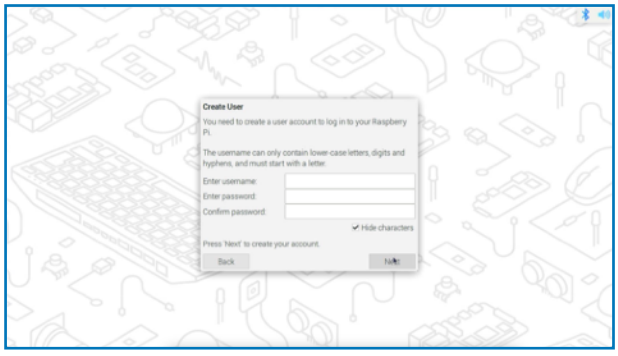
When you first power up the CM4 Stack Development kit you will presented with the introduction page.



All you need to do is have a read and then press the “Next” button to move on to the first of the configuration screens:



Here you set you Country of use, operating language, time zone and keyboard language. Once you have these set up to your needs, click the “Next” button to move on to the Username and Password configuration screen.



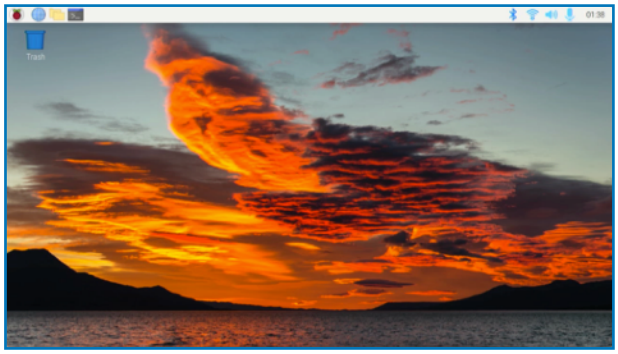
Make sure you fill these in and use a secure proper password and not a password that is likely to appear in any of the Top Passwords list available online as there have already been reports of hacked Raspberry Pi’s used to Robohack other online connected devices. Do not attempt to bypass the password boot system as logging in without a password will also put you CM4 Stack Development Kit at risk of hacking.

Once you have filled in the unique username and password, you can press the “Next” button to continue.
The next screen will ask you to change the default screen size to match the currently detected monitor size.

Press next to move on to the WIFI setup screen which shows that the WIFI version of the CM4 is actually installed and not an added option.

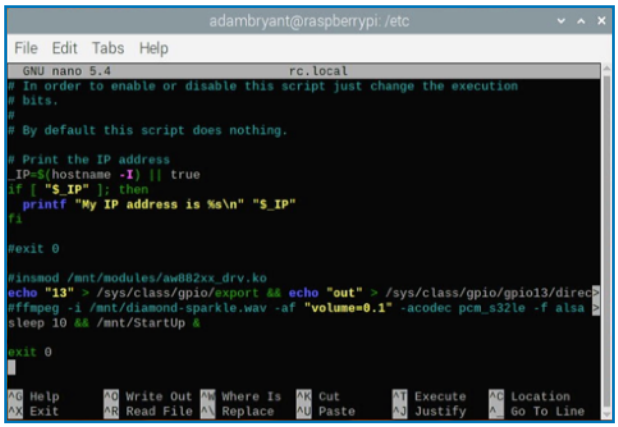
Once the Scan has completed, select your network, insert the password and click “Next” to continue.

The OS will attempt to look for updates and when complete will ask you to restart and then you will arrive in the Normal Raspbian desktop.



There is a problem with the updates in that there is a driver that gets ignored and stops the fan and the status display for being shown on the onboard screen. As a walk around for these issues, it has been found that commenting out (add ing # symbol to the line beginning), of two lines in

rc.local fixes these issues (see screen grab for the lines.)



Clock Demo.

In the documents for the CM4 Stack is an example code which shows a very nice QT based clock on the built in screen

If you follow the instructions for installing the program, you will encounter an error. The error is due to packages that didn't get installed.
The instructions for installing the example is as follows:

```
sudo apt update
sudo apt install qtbase5-dev qt5-qmake
qtbase5-dev-tools qml
sudo apt install build-essential cmake
git clone https://github.com/
Forairaaaaa/CM4Stack_QtDemo.git
cd CM4Stack_QtDemo
mkdir build && cd build
cmake .. && make
```

However when you run the last command you get an error about files not found. This turns out to be an issue with the dependences and you need to run:

```
sudo apt-get install qtdeclarative5-
dev
```

Before running

```
cmake .. && make
```

To get the example to compile.

Once the example has finished compiling, to can run the code with:

```
./cm4QtDemo
```

Which will run it on the desktop but to make it appear on the screen of the CM4Stack you need to run:

```
export QT_QPA_PLATFORM=linuxfb:fb=/
dev/fb$(cat /proc/fb | grep fb_st7789v
| awk '{print $1}')
./cm4QtDemo
```

And you will see the clock running based on the current system time as shown in the photo on the left.

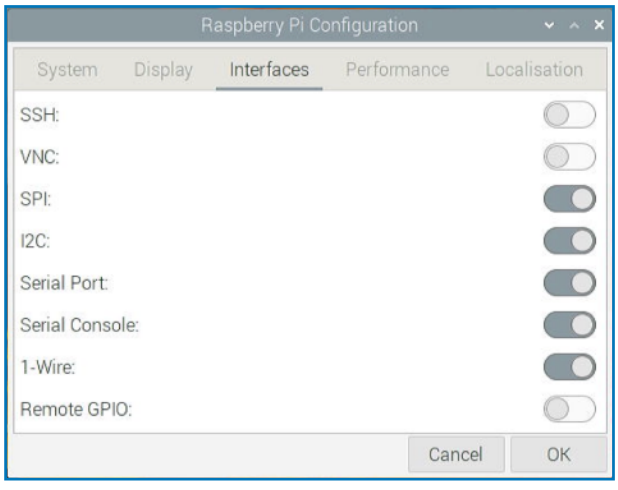
Connecting External Hardware via I2C.

While the CM4 Stack does not come with extra sensors, as mentioned earlier, the CM4 Stack as a “Grove” four pin port for I2C and UART communication (no analog port unfortunately).

In order to use an I2C device with the CM4 Stack you first need to connect a sensor to the I2C port with the four pin grove cable that came with the sensor (**while powered off !!**) and then power on the CM4 Stack.

For this example I will use the M5Stack ENVIII sensor available here M5Stack ENVII Unit.

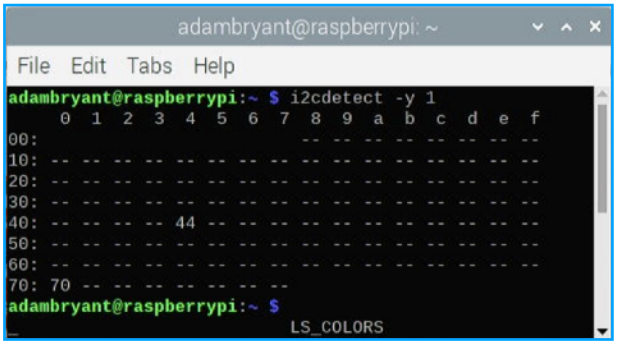
This is plugged into Port1 the I2C port.
Once the CM4Stack has booted up we need to check that the I2C services have been enabled in the os. To find this out, you need to open the Raspberry Pi configuration tool found in the Preferences menu.



Switch to the interfaces tab and check that I2C has been enabled as shown in the screen shot above. If I2C has been enabled, close this panel and then open the terminal and type:

```
I2cdetect -y 1
```

And hit return. This will show us a table with detected I2C address of devices currently connected to the I2C port.



In the screenshot we can see two address found 0x44 and 0x70. 0x44 is the address of the SHT30 is the temperature and humidity sensor while 0x77 is the address of the QMP6988 pressure sensor.

Next we need to install the libraries that python needs to access the SHT30 and the QMP6988 sensors in the ENVIII Unit using the following commands :

```
sudo pip3 install adafruit-
circuitpython-sht31d
```

Which copies and installs all the necessary files from the Adafruit archives need to access the SHT30’s temperature and humidity data.

Then to fetch the data which open Thonny and add the following configuration and imports into REPL(Shell) one line at a time:

```
import board
import busio
import adafruit_sht31d
i2c = busio.I2C(board.SCL, board.SDA)
sensor = adafruit_sht31d.SHT31D(i2c)
```

And then we can print the reading with:

```
print('Humidity:
{0}%'.format(sensor.relative_humidity))
```

Or

```
print('Temperature: {0}
C'.format(sensor.temperature))
```

Accessing the CM4Stack Development Kit in USB Mode.

From time to time you will need to access the CM4 Stacks eMMC memory in order to update and install a new operating system. In OSX this can be a pain.
Is this section I will show you how to get the CM4Stack to appear as a USB drive in OSX.

First you need to connect the CM4stack to a usb port via the Cm4 Stack's USB OTG port while holding the boot button in. If you do this correctly, the HDMI port will glow red from the boot LED hidden inside the case.

To get the CM4 Stack to appear as a USB drive, we need to install some software. This is easiest done using the Homebrew system.

If you haven't got Homebrew installed on your OSX machine type the following command in:

```
/bin/bash -c "$(curl -fsSL https://
raw.githubusercontent.com/Homebrew/
install/HEAD/install.sh)"
```

This command installs the required tools for home brew to work, you will need this to be installed as root so when asked for the system password, fill the password in as its requested by OSX and not the software installer.

Now that Homebrew is installed, We can install and compile the drivers needed for the CM4 Stack to appear as a drive. Jeff Geerling has written a good guide for it that you can find here:

https://www.jeffgeerling.com/blog/2020/how-flash-raspberry-pi-os-compute-module-4-emmc-usbboot

For the CM4Stack to appear we fist need to install LibUSB using:

```
brew install pkgconfig libusb
```

And when that has finished installing, clone the usbboot GitHub repository using:

```
git clone --depth=1 https://
github.com/raspberrypi/usbboot
```

After the repository has finished being copied, move into the directory with:

```
cd usbboot
```

And then to compile usb boot use:

```
make
```

When completed, USBBOOT can be run with:

```
sudo ./rpiboot
```

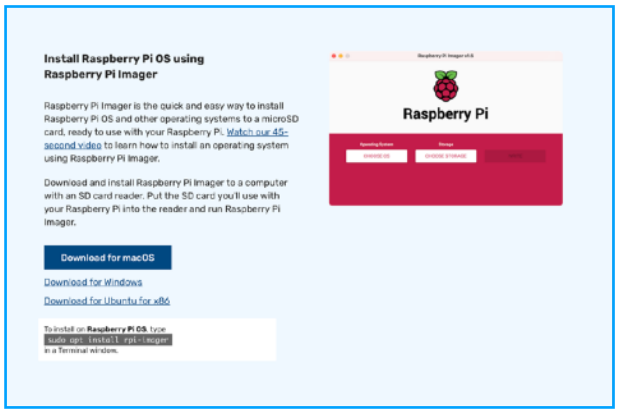
And then after a few seconds, the CM4Stack will appear as a USB drive.

Installing the new OS version

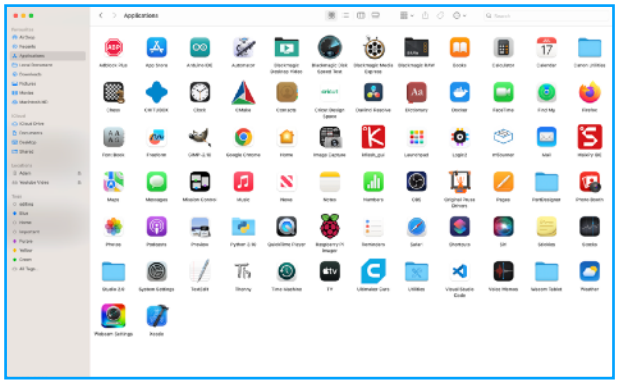
The first batch of CM4 Stack Development Kits came with the 32 bit version of RPI bullseye preinstalled. Shortly after, the 64 bit version was released and that is the version I will now install.

Go to : https://t.co/3mx4cW3hqa and download the latest version.
The files comes as a zipped disk image like most RPI distributions and so we will need to unzip the file file to reveal the disk image.

To burn the image to the CM4Stacks eMMC memory we need to download the RPI imager program from https://www.raspberrypi.com/software/

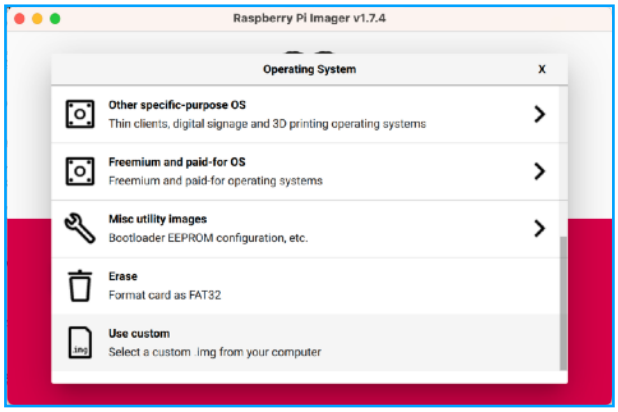


Click on the version for the OS you are using (in my case OSX) to download and extract to the applications folder.

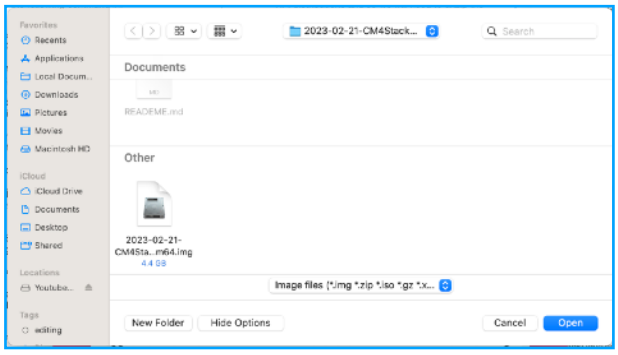


Double click (or single click depending on your OS) to open the imager.

Click on “Choose OS” and then scroll down to find “Use Custom”. This will allow us to select the disk image we just downloaded so that it can be burnt to the CM4 Stack’s eMMC.



This will open a file dialog that will allow us to look for the disk image.



Select the disk image and click open to return to the main screen showing that the disk image is now selected.

Now click on storage to open the available USB devices which in my case only shows the CM4Stack. Click on it to select it and click “Write to begin installing the new OS version.

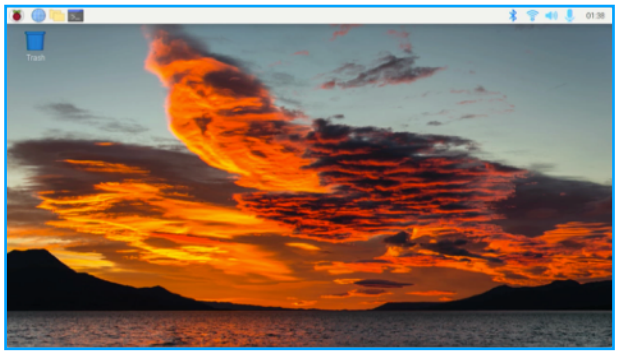
You may be asked for your password and permission to access certain folder on OSX but this is down to security setting within the OS put inlace to prevent unauthorised file and folder access.

Once the new OS has been written, you can reboot the CM4 Stack and you will now be in the new 64bit RPI os.

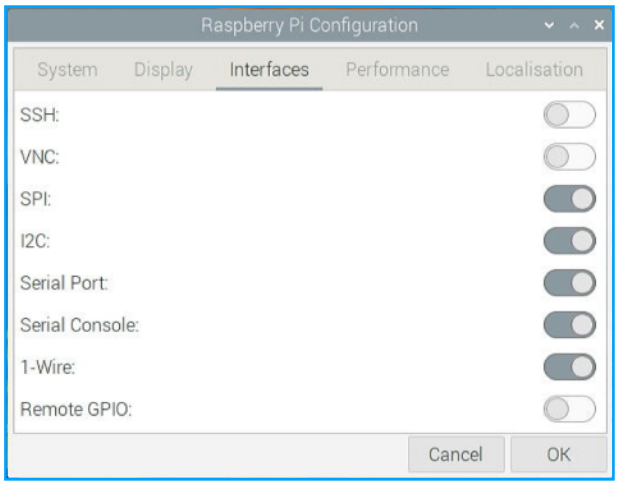
Setting up SSH remote access to the CM4Stack.

In a lot of cases it will not be possible to plug a keyboard mouse and monitor into the CM4 Stack in order to update the software of install new software. This is mode is known as “headless” mode. In order to access the CM4 Stack in this mode we need to configure SSH on the CM4 Stack which will allow us to connect to it from another computer on the network to update and install software.

In order to set up remote access to the CM4 Stack you will need to connect a USB Keyboard, mouse and a HDMI monitor and then boot into Raspbian.



Click on the Raspberry icon and go to Preferences > Raspberry Pi Configuration.

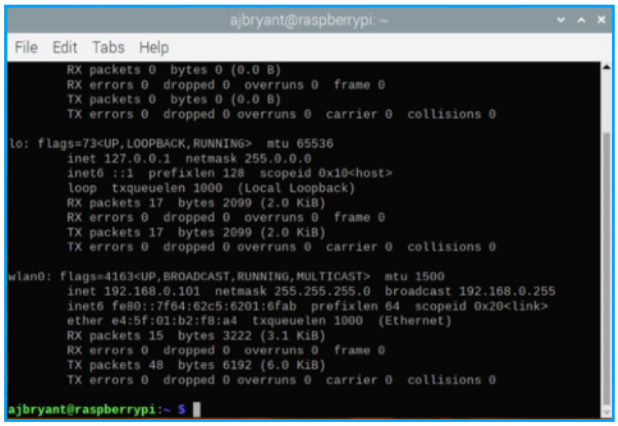


In this window, you will need to click on the slider to the right of SSH to enable SSH, Click on OK and then reboot the CM4 Stack.

Once the CM4 Stack as rebooted, the ip address should be shown on the CM4 Stack’s screen however, if it isn’t shown you will need to find the IP address. To get the IP address assigned to the CM4 Stack, Open the terminal and type:

```
Ifconfig
```

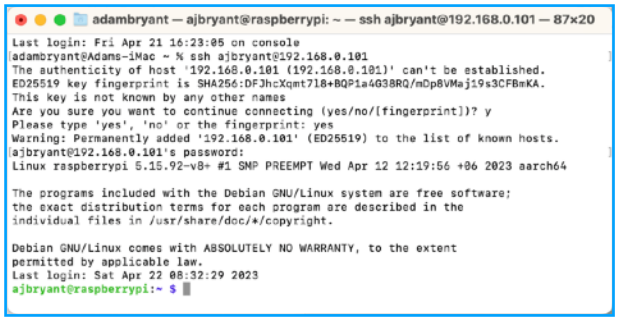
This command brings up a list of active network devices. Scroll down to wlan0 and you will see the currently assigned IP address.



Write down the IP address ready and open a terminal in another computer and type in:

ssh <yourname>@<cm4ipaddress>

Replace <yourname> and <cm4ipaddress> with the user name you gave the CM4 during initial boot and setup and the IP address that should be shown on the the CM4 Stacks screen. Type “yes” (y does not work) to the questions and you will now be connected to the CM4Stack remotely.



To close the SSH session you just need to type

```
exit
```

And then close the terminal window.

If you have issues of ssh keys having been changed, this is because the key files saved on the computer (not the CM4) are different. In OSX you need to find the folder <username>>SSH and delete all files in side. When you re try to connect with ssh you should be able to connect normally.

Shutting down the CM4Stack via SSH.

In order to safely power off the CM4Stack so that the files don't get corrupted there are two remote commands that are needed.

```
sudo shutdown -r now
```

Is used to reboot the CM4Stack while:

```
sudo shutdown now
```

Is used to shut down the CM4Stack.

Never remove the power from the CM4Stack without running these commands as there is a high chance of the OS stored on the CM4Stack's memory getting damaged and preventing the CM4Stack from booting.

IP Address (Internet Protocol Address)

The is a problem with using the previous steps to access the CM4 Stack and that is due to IP (Internet Protocol) address' are not permanently attached to a device on a network and that rebooting the CM4 Stack may result in a different address being assigned to the CM4Stack.

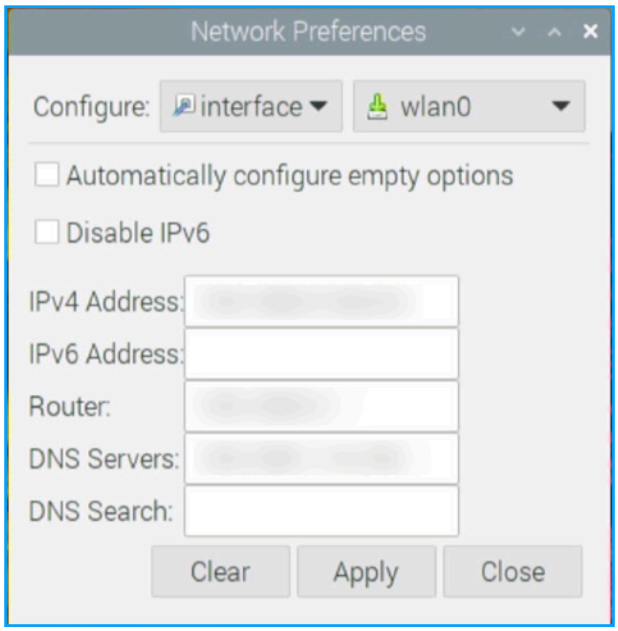
We can get around this issue by having the CM4Stack request that an IP address is permanently assigned to it. For this, we need to edit a file to add some settings.

I started by reading the guide on Tom's Hardware website found here: https://www.tomshardware.com/how-to/static-ip-raspberry-pi but discovered that some things had to be slightly differently for the CM4 Stack.

The first step in setting up a fixed IP address is to find out the current assigned IP address. This is normally displayed on the the CM4 Stacks load screen next to wlan0 for Wireless or eth0 for a wired network:

In the above image you can see that the CM4Stack is connected via wlan0 but I have blurred out the IP address for privacy.

There are times where this doesn't always show and so in this case you need to connect a screen, keyboard and mouse to the CM4 Stack and view the address as shows in the Rasbian OS. To find it in Raspbian you can look it up by right click on the wifi icon, click on wifi preferences and then select the wlan0 interface:

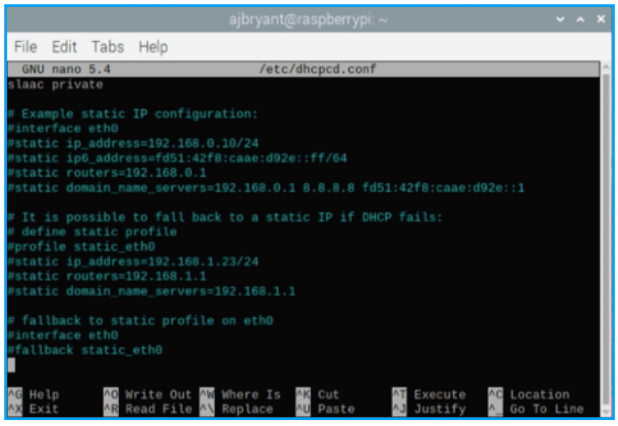


Which you can see has also been blurred out agin for privacy.
As well as the IP address, you will also see the Routers address and the DNS server address, make a note of all three address ready for editing a file.

While we have a Keyboard and mouse connected to the CM4 Stack, open a terminal and type:

```
Sudo nano /etc/dhcpcd.conf
```

And scroll to the bottom of the file:



Now add the following lines:

```
interface wlan0
static routers=[ROUTER IP]
static domain_name_servers=[DNS IP]
static ip_address=[STATIC IP ADDRESS
YOU WANT]/24
```

Note 1: It is worth noting that in the guide it shows an underscore between static and routers but this cause issues on my CM4 and I had to replace the underscore with a space.

Note 2: Do not forget to add the /24 after the IP address you want to be permanently set as this defines a subnet mask of 255.255.255.0!

Save and close the file and then reboot the CM4Stack to apply the change.

Installing an MQTT Broker.

As the CM4 stack is a full RPI micro computer, we can set it up as an MQTT broker to handle MQTT traffic sensors without the use of an online service. In the following steps I will show you how to install and configure the Mosquitto MQTT broker.
As always, when installing software on the CM4 Stack, first run:

```
Sudo apt update
```

Followed by

```
Sudo apt upgrade
```

To update the currently installed software and look for updates for available software. Once the updates have finished installing and the CM4 Stack and been restarted to apply any system updates, we can type in the following command to begin the Mosquitto install.

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

The above line consists of two commands. The first part of the command installs the Mosquitto broker and the second installs the Mosquitto clients used to communicate and test the broker on the CM4 Stack.

Once installed, we then need to set Mosquitto to autostart on boot. For that we need to use the following command to setup the service.

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

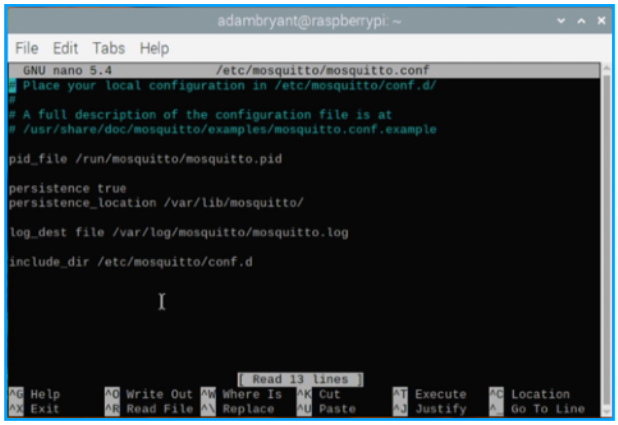
To test that the Mosquitto broker is working type:

```
mosquitto -v
```

Before we can communicate with Mosquitto, we need to set up an authentication file. You do that by typing

```
Sudo nano /etc/mosquitto/
mosquitto.conf
```

To open the config file in the Nano text editor.



Move to the end of the file and on a new line add the following lines of code and then save the file.

```
listener 1883
allow_anonymous true
```

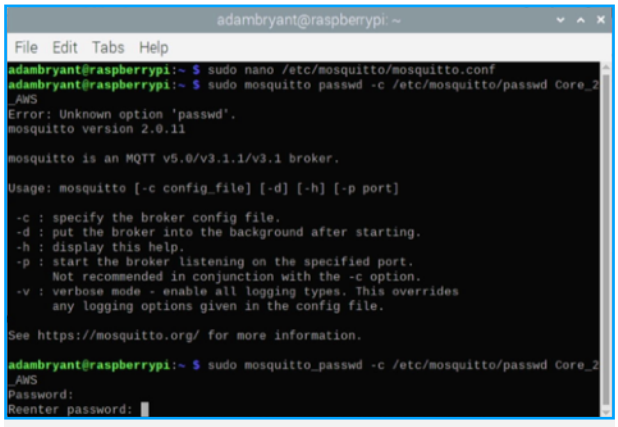
In order for the changes to the configuration to be used, Mosquitto need to be restarted with:

```
sudo systemctl restart mosquitto
```

Warning: the above is for testing the Mosquitto service and if others use the network the CM4Stack is connected to you will need to configure user and password access.

To add a user and password ignorer to secure the connection we first use:

```
sudo mosquitto_passwd -c /etc/
mosquitto/passwd Core_2_AWS
```



I'm using the user name Core_2_AWS because this is part of my IOT V2 book that is based around the Core2 AWS controller.

During the process, a file called passwd gets created and you will be asked to type in a password. Be careful when typing the password in as it is not shown on screen while you type it and so there is a large possibility to make a mistake.

Next type in the nano command again to reopen the file:

```
Sudo nano /etc/mosquitto/
mosquitto.conf
```

This time add:

```
per_listener_settings true
```

To the top of the file and:

```
password_file /etc/mosquitto/passwd
```

Under the last two lines added previously.
Save the file again and restart Mosquitto again with:

```
sudo systemctl restart mosquitto
```

To check that Mosquitto is working with the new changes run:

```
sudo systemctl status mosquitto
```

To test Mosquitto can receive messages, open a new shell/terminal and type:

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

This creates a new topic called test in which will hold our messages for testing.
Now open up another new shell/terminal and type:

```
mosquitto_sub -d -t testTopic -u
Core_2_AWS -P pass1245
```

Replacing Core_2_AWS with the username you set and replace the pass1245 with the password you set.
In order to send a message to the Mosquitto broker from within the CM4 Stack, type the following into the new window and you should see the message appear in the previously opened window.

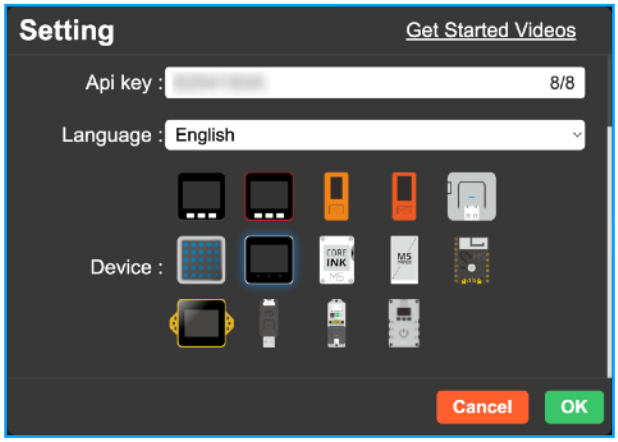
```
mosquitto_pub -d -t testTopic -m
"Hello world!"
```

Sending MQTT messages from M5Stack controllers to the MQTT broker.

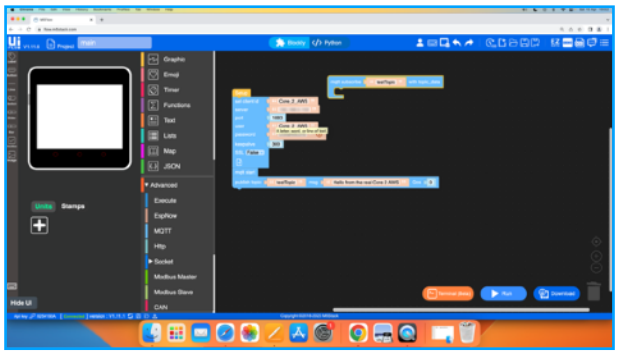
In order to test that the Core2 AWS can also communicate with the Mosquitto broker we have just set up on the CM4 Stack, we power on the Core two and connect to UIFlow 1: https://flow.m5stack.com/

Use UIFlow 1 for the moment because UIFlow 2 in only in open Alpha for testing with S3 based M5Stack controllers.

In UIFlow make sure your Core2 AWS is connected by going into the settings:

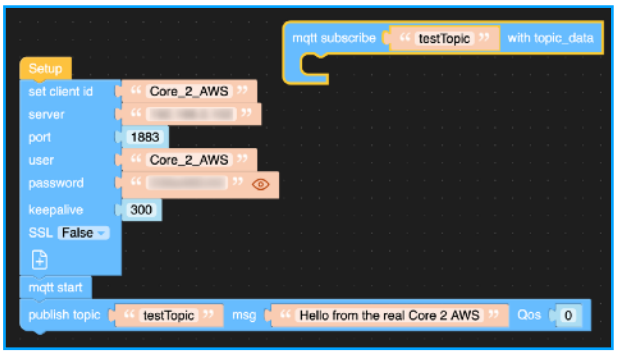


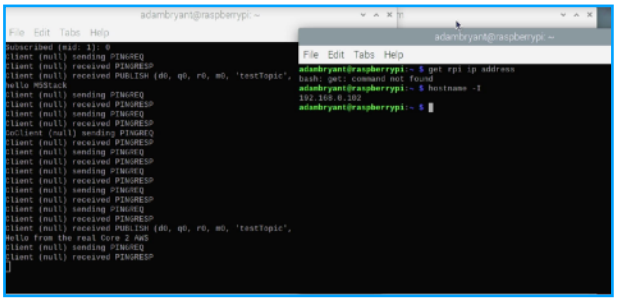
Click on the Icon of the Core 2 to select it and then type in you API key that gets shown on the Core 2's screen on startup. Click ok to close this panel and then look on the bottom left of uiflow:



You should see the API code printed in green. If the API code is red it means that the code is wrong or that a connection to the M5Stack server is not available.

If its API code is green, place the following code blocks as shown, insert you own username and password, click the run button and when you switch back to the CM4 Stack you should find a message appearing in the terminal.





Building the MQTT server with Mosquitto, Nodered, Grafana and InfluxDB.

One of the issues I discovered from configuring the CM4 Stack as a MQTT server using the previous steps is that a software update prevented certain versions of Nodered from installing. After spending over a week trying to solve the issue, I abandoned that method and had to wipe the CM4 Stacks eMMC and start again.

The Portaina method uses a package or container preconfigured with a set of of tools, libraries and programs for creating software but allows all programmers to have the same software environment where ever they may be.

I started building the server by following this guide:

https://learnembeddedsystems.co.uk/easy-raspberry-pi-iot-server

But in several sections I had to deviate because certain functions were not working for my setup.

As when installing new software on the CM4Stack, start with the usual:

```
Sudo apt update
```

Followed by

```
Sudo apt upgrade
```

And then when complete type in the follow:

```
curl -fsSL https://
raw.githubusercontent.com/SensorsIot/
IOTstack/master/install.sh | bash
```

This downloads the necessary tools needed to setup and install the server software. Once finished use:

```
sudo shutdown -r now
```

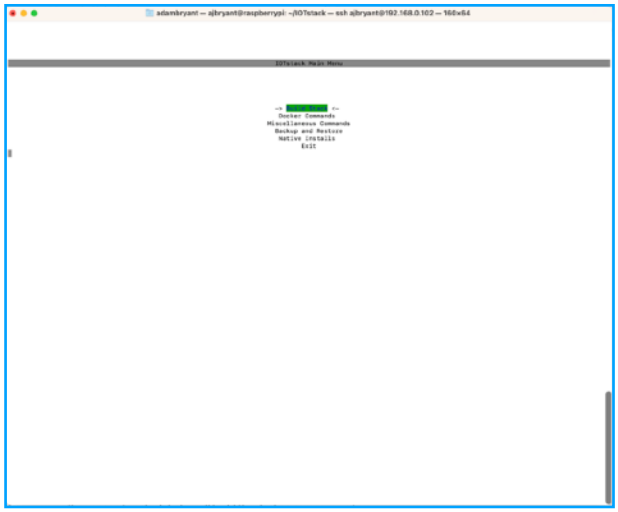
To reboot the CM4Stack.
After the reboot, move into the IOTstack folder with:

```
cd IOTstack/
```

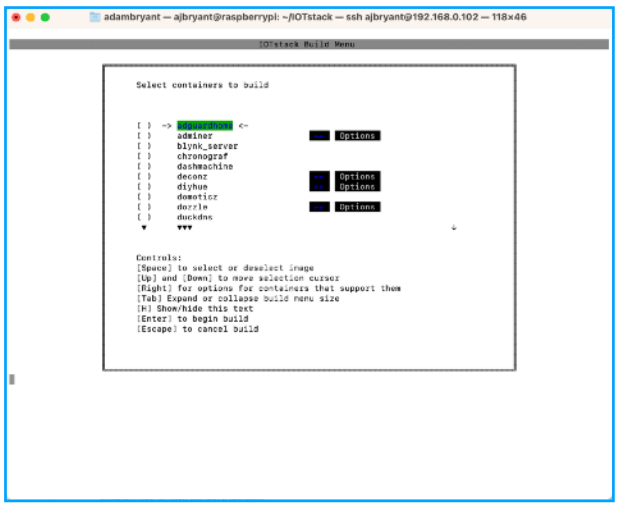
And run:

```
./menu.sh
```

To open the installer which is used to configure and install the necessary packages.



Hit the <Enter> key to enter the menu which allows selection of the packages to install.



Use the Arrow keys to move up and down the menu and hit the <space> button to select the following packages:

- Grafana
- InfluxDB
- Mosquitto
- Node-RED
- Portainer-CE

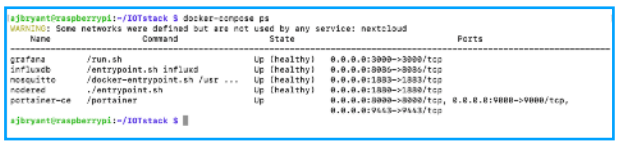
When you get to Node-Red, you will find a warning message, press the <right arrow> key to enter the options, select “build list” and when complete click on back to return to the selection screen. Once all the packages are selected hit <enter> to build the packages list and then go to “Docker Commands” and click on “Start Stack” to begin downloading and install ion of the docker images needed for the server.

When complete, we can go back to the menu and click on exit to leave the installer.

To check if everything is working, type in:

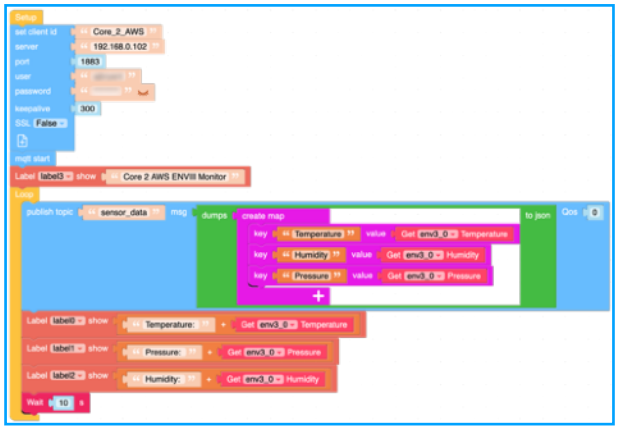
```
docker-compose ps
```

Which will show us the running containers.



Setting up a sensor and checking Mosquitto is receiving.

Next we need a device to test the connection to the CM4 Stacks server works. The MQTT test code I used on the Core2 AWS to read and transmit values has been built in UIFlow and is as follows:

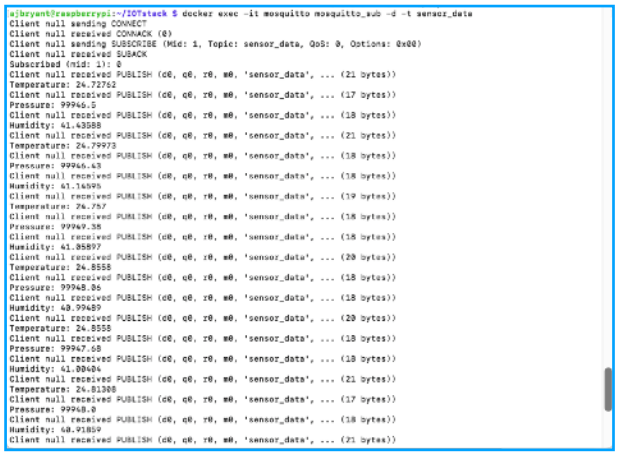


And uses the same ENVII sensor I used previously. In the above image I have blurred out the username and password for security reasons.

Run a test MQTT client over the ssh terminal with:

```
docker exec -it mosquitto
mosquitto_sub -d -t sensor_data
```

And you will see the screen filling with readings.



The above image show that the Core2 AWS is connected to the CM4 Stacks MQTT server and data is being received.

The next step is to create the database that will store the readings. Close the shell window to terminate the shell session, open a new window and re connect to the CM4Stack over SSH.

Open Influxdb using:

```
docker exec -it influxdb influx
```

And type the following to create the new database:

```
CREATE DATABASE sensor_data
```

Once the database is created, we can exit Influxdb by typing:

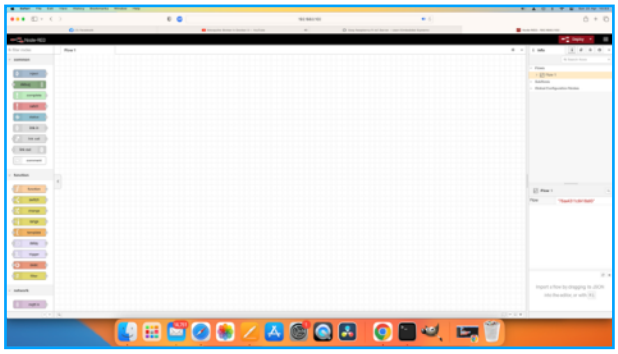
```
quit
```

And can now move on to configuring node red.

Nodered setup.

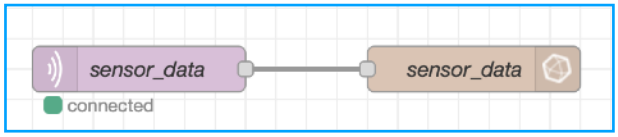
To load Nodered you need to open a web browser, type in the CM4Stacks IP address (shown on the front screen) but add :1880 to the end:

http://192.168.0.102:1880 (For example)

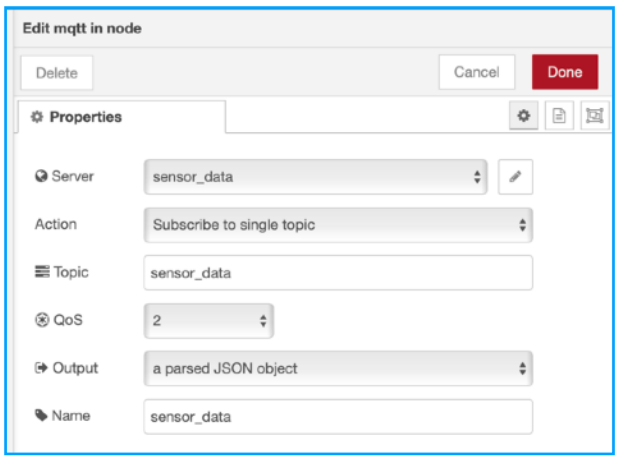


Nodered has open on a blank screen in my case because I have already tried accessing node red before.

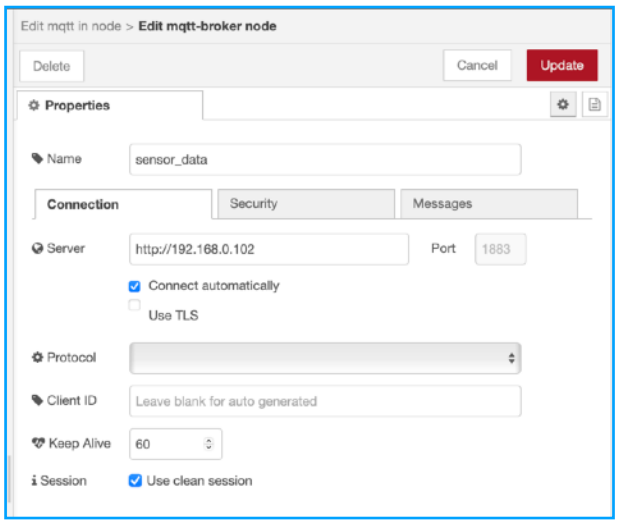
Add an “MQTT In” block and an “Influxdb out” block and connect as shown:



To configure the blocks, double click on the MQTT In block to open the configuration options:



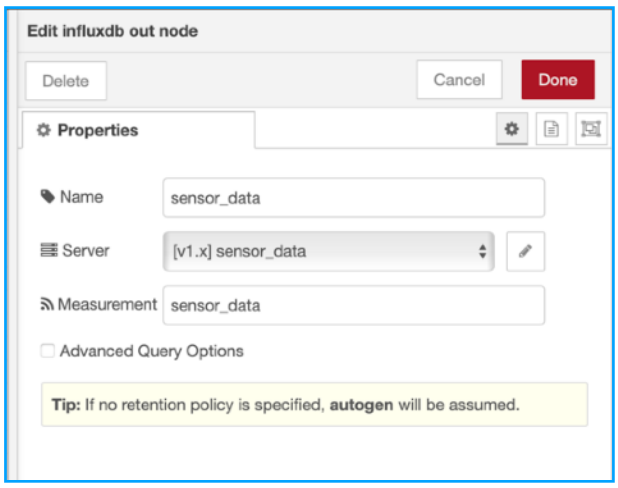
Click on the pencil icon to open the server settings:



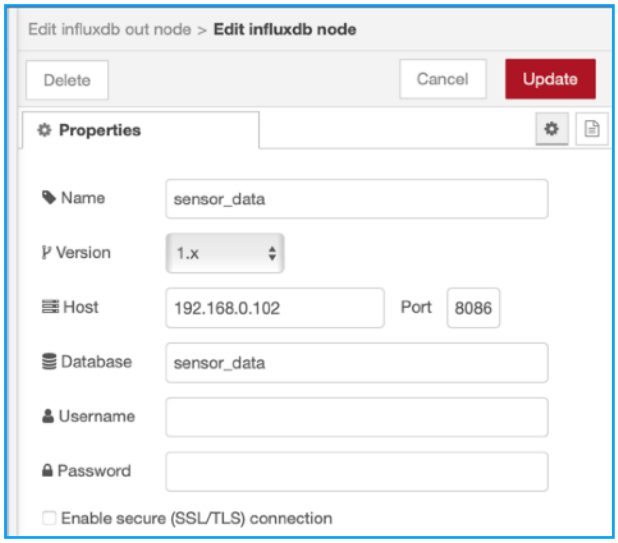
Type in a name for the server (I used the sensor_data topic for the name) set the IP address to the CM4 stack, and leave the port number as 1883.

Click on “Update” to close and apply the settings and return to the previous panel, select the newly added server from the dropdown list if not automatically set. Make sure the other options are set as shown above and click “Done” to finish configuring the MQTT block.

Next, double click on the influxdb out block to open its settings:



Click on the pencil icon to open the server setting window:



Type in the CM4 Stacks IP address again and leave the port number as shown, type in the sensor_data database name we created earlier, give the server a name and then click the “update” button to close and apply these settings and return to the previous window.
Make sure that the “Measurement” and “Name” box’s are filled with the database name and click “Done” to close the settings and that is all the configuration complete. Click on the “Deploy” button at the top of the screen and everything should now work.

Influxdb

We can test that influxdb is receiving our data by going back the the shell window and typing the following commands:

```
docker exec -it influxdb influx

USE sensor_data

show measurements

select * from sensor_data
```

Which should now show data being written to the database:

```
ajbryant@raspberrypi:~ $ docker exec -it influxdb influx
Connected to http://localhost:8086 version 1.8.10
InfluxDB shell version: 1.8.10
> USE sensor_data
Using database sensor_data
> show measurements
name: measurements
name
----
Temperature
sensor_data
> select * from sensor_data
name: sensor_data
time                Humidity Pressure Temperature
----                -------- -------- -----------
1682245247260474736 39.45678 99947.81 24.33242
1682245257905071876 39.16381 99946.94 24.47662
1682245268612290116 39.35302 99943.25 24.53269
1682245279289831928 39.10887 99943.49 24.3885
1682245290013088840 38.5565  99945.43 24.37515
1682245300859653982 38.72587 99949.81 24.44457
1682245311507907215 39.33928 99946.31 24.47662
1682245322256130528 38.77165 99945.18 24.41787
1682245333023983419 38.96391 99943.38 24.31907
1682245343448959087 38.91814 99944.88 24.33242
1682245354145046393 38.82963 99943.74 24.26299
1682245364823033020 39.28435 99948.19 24.31907
1682245375516433804 39.08598 99943.44 24.41787
1682245386199235814 38.5977  99943.11 24.34577
1682245396889206061 38.68315 99947.88 24.41787
1682245407585502349 38.87694 99945.61 24.33242
1682245418260898011 38.40544 99944.31 24.33242
1682245428949569342 38.41764 99947.88 24.40185
> quit
ajbryant@raspberrypi:~ $
```

With this all working, we can now type:

```
quit
```

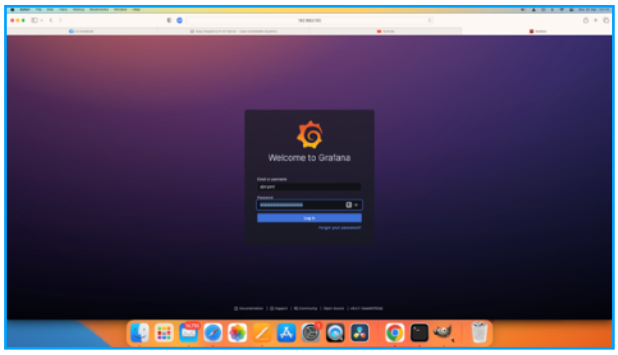
To exit out of influxdb and move on to Grafana.

Grafana

Grafana is an open source web based tool used for collecting and display the data being captured by devices connected to the MQTT server. With Grafana we can separate out and graph the individual sensor data recorded on the ENVIII and display on a clean web page with next to no programming.

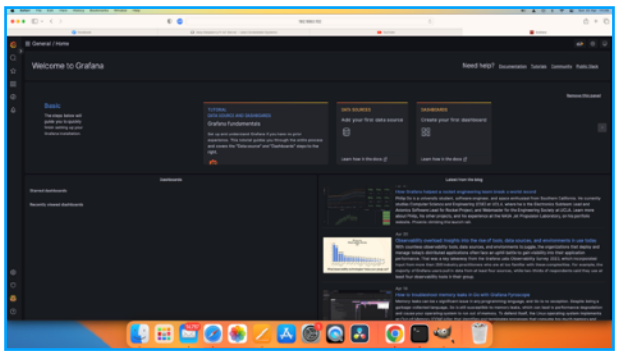
In the next few paragraphs, I will show you the steps need to build the display graphs shown in the screen shot above from the data getting written to the infuxdb database.

To access Grafana hosted on the CM4 Stack you need to type in the CM4 Stack's IP address shown ion the front of the CM4 Stack followed by :3000 which will load up the Grafana from page asking you to log in.

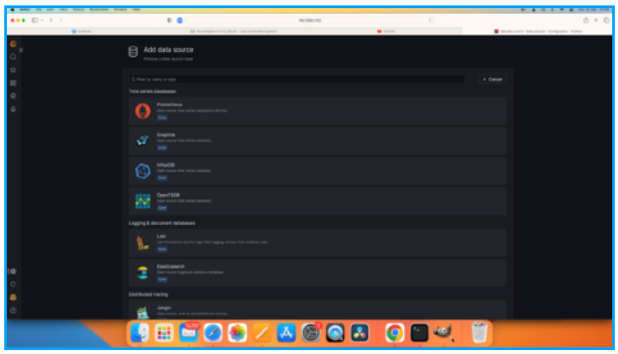


The default user name and password for Grafana is admin and admin (lowercase) however, when you log in for the first time, you will be requested for a new password.

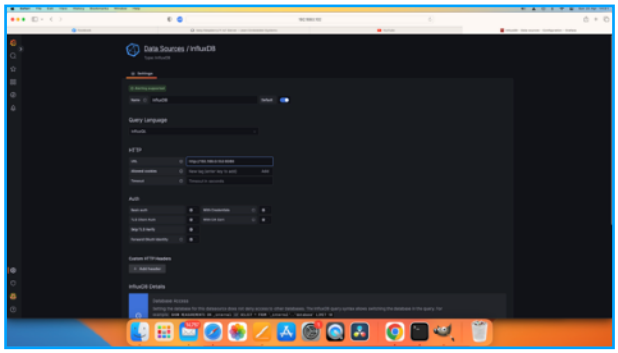
Once the new password is set and logged in, you will be taken to Grafana's main screen:



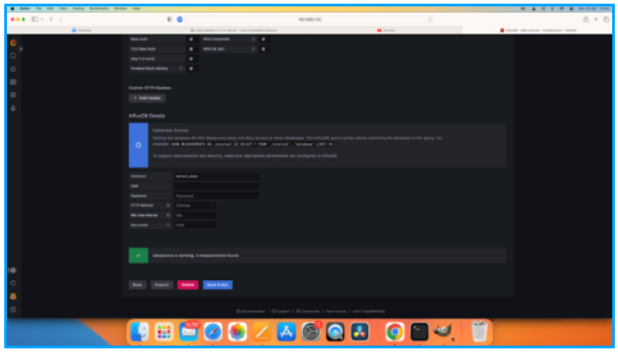
Before we can display an dat being recorded, we first need to select the data source to read from. Click on the Data Sources panel to add a data source:



As we are using an Influxdb database, click on Influxdb to add the data source and move on to the configuration panel.



Type in the CM4 Stack's IP address followed by :8086 and then scroll down to the bottom of the page:

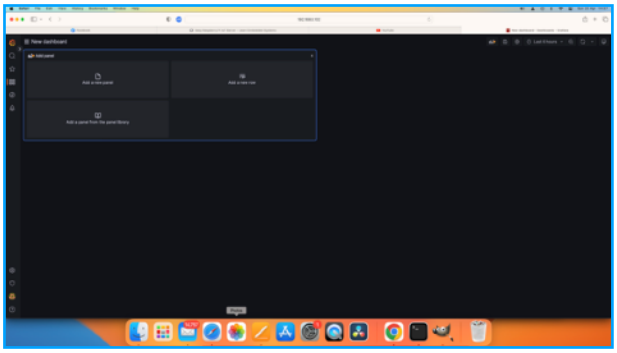


Type in the name of the influxdb database we created (in my case sensor_data) and then click on the "Save & Test" button.
If you get a green block with a message saying:

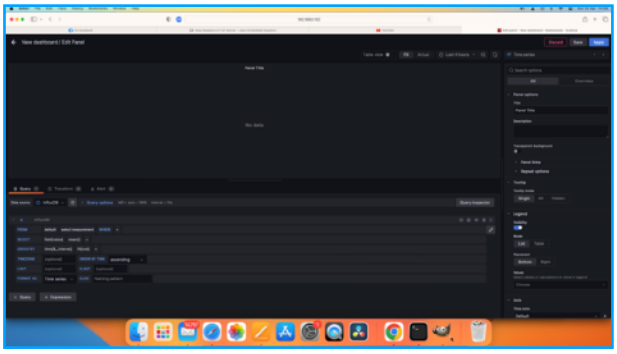
```
datasource is working. 2 measurements found
```

It mean the configuration is correct and we can more on to creating the dashboard to display the readings.

Click on the round orange logo to return to the main page and you should see that the data source panel is now saying complete. Click on the Dashboard panel to go to the dashboard creation screen:

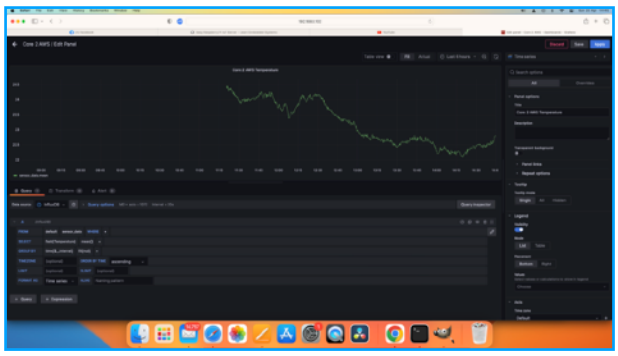


Click on the "Add New Panel" to create a a new panel and you will be taken to the panel setup screen.



On the right of the page change "Panel Title" to "Core 2 AWS Temperature " then in the bottom click on "select measurement" and set to "sensor_data" which is the name of the influxdb database.

Next click on the "field Value" box and set it to "temperature"and you should see the graph above populated with readings.



Click on “Save” and you will be returned to the dash board with a new graph.

Add two more panels for the Pressure and humidity (don’t forget to click “save” and “apply”) And you now will have a nice no code graphical display of the reading being sent and organised by the MQTT server hosted on the CM4 Stack Development Kit.

Controlling the screen.

A big shout out to Lous Llamas @LuisLlamas on twitter for giving me the directions to get started on programming the screen as I had forgotten all I knew about the frame buffer.

The CM4Stack Development kit has an ST7789V2 2.0” screen on the front with a resolution of 240X320 pixels set up in portrait mode.
There are many ways to control and display things on the built in screen but for ease of use, in the following step I will use the Python programming language with the Pygame python module.

In order to use the Pygame Python module you will first need to install Pygame if its not already installed.

To install Pygame on the CM4Stack type:

```
Sudo pip install pygame
```

And hit return. If everything worked then you will see the following message:

```
ajbryant@raspberrypi:~ $ sudo pip
install pygame
Looking in indexes: https://pypi.org/
simple, https://www.piwheels.org/
simple
Requirement already satisfied: pygame
in /usr/lib/python3/dist-packages
(1.9.6)
```

Confirming that Pygame is now installed. Next step is to create a work folder. In the root folder shown when you first SSH into the CM4Stack, create a folder called CM4Stack with:

```
mkdir CM4Stack
```

Enter the folder with

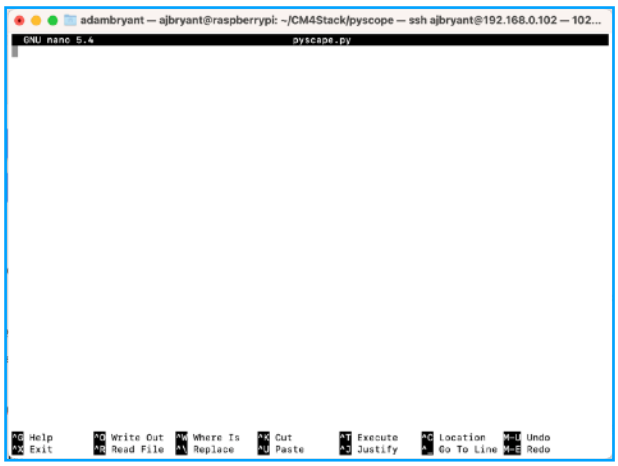
```
cd CM4Stack
```

Changing the Background Colour.

In order to start working with the screen I will make a base class called cm4base.py

```
Sudo nano cm4base.py
```

And the nano editor will appear with a blank screen.



Next copy the following code from below:

```
import os
import pygame
import time
import random

class cm4base :
    screen = None;

    def __init__(self):
        disp_no = os.getenv("DISPLAY")
        if disp_no:
            print ("I'm running under
X display = {0}".format(disp_no))

        # Check which frame buffer
drivers are available
        # Start with fbcon since
directfb hangs with composite output
        drivers =
['fb_st7789v','fbcon', 'directfb',
'svgalib']
        found = False
        for driver in drivers:
            # Make sure that
SDL_VIDEODRIVER is set
            if not
os.getenv('SDL_VIDEODRIVER'):

os.putenv('SDL_VIDEODRIVER', driver)
            try:
                pygame.display.init()
            except pygame.error:
                print ('Driver: {0}
failed.'.format(driver))
                continue
            found = True
            break

        if not found:
            raise Exception('No
suitable video driver found!')

        size =
(pygame.display.Info().current_w,
pygame.display.Info().current_h)
        print ("Framebuffer size: %d x
%d" % (size[0], size[1]))
        self.screen =
pygame.display.set_mode(size,
pygame.FULLSCREEN)
        # Clear the screen to start
        self.screen.fill((0, 0, 0))
        # Initialise font support
        pygame.font.init()
        # Render the screen
        pygame.display.update()

    def __del__(self):
        "Destructor to make sure
pygame shuts down, etc."

    def test(self):
        # Fill the screen with blue
(0, 0, 255)
        blue = (0, 0, 255)
        self.screen.fill(blue)
        # Update the display
        pygame.display.update()
```

Press Control+X to close the Nano editor but type Y to save the text to the buffer and then hit return to save the buffer to the file.

Nano will no close and you will return to the terminal. All I have done so far is to create the base handle class for writing to the ST7789V2 screen. In order to change the screen colour we need to add some code for a runnable program.

Re open cm4base, scroll to the end and add the following:

```
cm4screen = cm4base()
cm4screen.test()
time.sleep(5)
```

The program here consists of only three lines that create an instance of the cm4base class, calls the test function and then ends after five seconds. Save and close the file again to return to the terminal and then to run the cm4base code use:

```
sudo python cm4base.py
```

Hit return and you will see the screen clear and then turn red before clearing again.

Most of the code is just used for configuring the frame buffer ready for writing and in order to change the colour we just alter the line in the function `def test`.

```
def test(self):
        # Fill the screen with red
(255, 0, 0)
        red = (255, 0, 0)
        self.screen.fill(red)
        # Update the display
        pygame.display.update()
```

Just change the line:

```
red = (255, 0, 0)
```

To

```
blue = (0, 0, 255)
```

And the following line to:

```
self.screen.fill(blue)
```

It is worth noting that when pyscope.py finishes running the old screen doesn't get displayed. In order to restore the default screen we need to reboot the CM4Stack with:

```
sudo shutdown -r now
```

In order to restart the CM4 Demo screen you need to issue the following from the CM4's root folder:

```
/usr/local/m5stack/demo /dev/fb$(cat /
proc/fb | grep fb_st7789v | awk
'{print $1}') 2>&1 >> /dev/null &
```

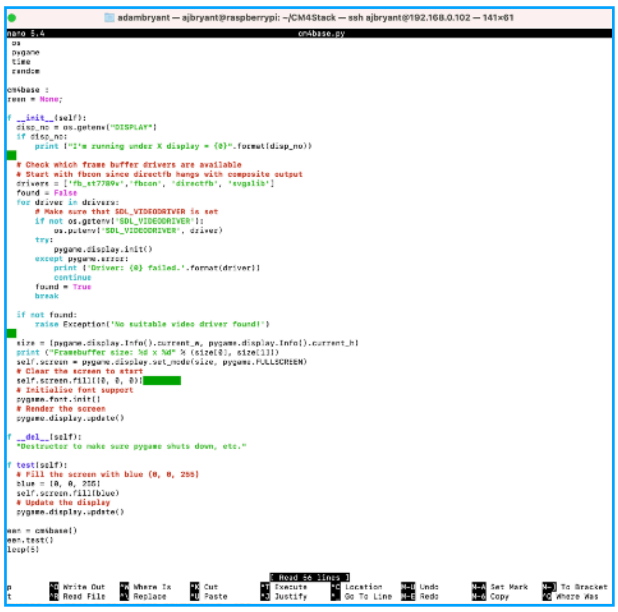
While this restarts the demo, we need to find a way of running this at the end of our program.

Displaying text on the screen.

In this step I will show you how to display text on the screen and for this I will be using the "Hello World" example modified to work with the code we have created so far. Reopen the pyscope.py file with:

```
Sudo nano cm4base.py
```

Which should still be full of code from the last step.



Replace :

```
cm4screen = cm4base()
cm4screen.test()
time.sleep(5)
```

With the following code:

```
cm4screen = cm4base()
# Get a refernce to the system font,
size 30
font = pygame.font.Font(None, 30)
# Render some white text (pyScope 0.1)
onto text_surface
text_surface = font.render('Hello
World (%s)' % "0.1",
True, (255, 255, 255))  # White text
# Blit the text at 10, 0
cm4screen.screen.blit(text_surface,
(10, 0))
# Update the display
pygame.display.update()

# Wait 10 seconds
time.sleep(10)
```

Save and close the file again and when you rerun:

```
sudo python cm4base.py
```

You will see Hello World printed on the screen.

Auto reload the Demo from code.

After consolation with the Raspberry pi forum, It turns out that in order to restore the CM4 Stacks built in demo screen from within the python code, all you need to do is add the following code as the last line of the program:

```
os.system("/usr/local/m5stack/demo /
dev/fb$(cat /proc/fb | grep fb_st7789v
| awk '{print $1}') 2>&1 >> /dev/null
&")
```

DataSheets

- CM4-datasheet
- CM4IO-datasheet
- ST7789V2
- AW88298
- MP8759
- AW32901
- SY8003
- ME1502
- BM8563
- ATECC608B

This file is located at http://ajbryant.co.uk/Guides/WIP_CM4StackDevelopmentKit.pdf

Index

Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

M5Stack:

K127 K127-US K127-EU