PPS Cold Box - RaspberryPi setup

José Miguel

July 2025

Contents

1 Introduction

This pdf is part of a folder containing other files; those files will come in handy throughout the tutorial. Please make sure to update the Raspberry Pi and have everything up to date with:

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

SSH should be enabled, and a static IP address is required. Furthermore, make sure that I^2C communication is enabled. Use "sudo raspi-config" and navigate to "Interfaces Options" -> "I2C".

Install all the necessary packages:

```
sudo apt install -y python3-pip python3-smbus python3-rpi.gpio python3-gpiozero \
i2c-tools python3-smbus2 python3-paho-mqtt python3-scipy
```

2 Server and Data Visualization

2.1 Setting up Docker

To start the installation process, download IoTStack using the following command, it should reboot once the download is complete:

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

If any prompt shows up to download certain packages, do accept. Once the Pi is back, navigate to the IoTStack folder, then run the menu script:

```
cd IOTstack/
./menu.sh
```

Use enter to select "Build Stack". Use the "up" and "down" arrows to navigate between packages. Press the space bar to select: Grafana, Influxdb, Mosquitto and NodeRed. The interface can glitch quite a bit, make sure to use the fullscreen mode. Furthermore, there are two Influxdb options; select Influxdb, not Influxdb2.

There will be a warning message upon selecting the NodeRed package, simply press the right arrow key to enter the options menu, hit "Select & build addons list" and accept the default. Navigate back to the packages menu.

Press "Enter" to begin build, navigate to "Docker commands" and select "Start stack"

After the setup is finished, run the command:

```
docker-compose ps
```

The output should be something like the following (Created and Status columns adjusted to when the containers were created):

Service	Image	Command	Created	Status	Ports
grafana	grafana/grafana	`/run.sh`	2 days ago	Up 20 hours (healthy)	0.0.0.0:3000→3000/tcp, [::]:3000→3000/tcp
influxdb	influxdb:1.8	`/entrypoint.sh influxd`	5 days ago	Up 20 hours (healthy)	0.0.0.0:8086→8086/tcp, [::]:8086→8886/tcp
mosquitto	iotstack-mosquitto	`/docker-entrypoint.sh`	5 days ago	Up 20 hours (healthy)	0.0.0.0:1883→1883/tcp, [::]:1883→1883/tcp
nodered	iotstack-nodered	`./entrypoint.sh`	5 days ago	Up 20 hours (healthy)	0.0.0.0:1880→1880/tcp, [::]:1889→1880/tcp

2.2 Setting up Influxdb

Inside the IOTstack folder, run the following command to enter the INfluxDB container:

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

Create a database named "sensor_data" and exit using:

```
CREATE DATABASE sensor_data
quit
```

2.3 Setting up Node-Red

Open Node-Red using a Web browser. If the set-up is being done via the Raspberry Pi, search:

```
http://localhost:1880
```

Otherwise, use:

```
http://<ip_address>:1880
```

Open the drop-down menu in the top right corner and select "Import". Click "Select a file to import" and import the file "flows.json" from the provided folder. A new flow should have been created. It should look something like this:

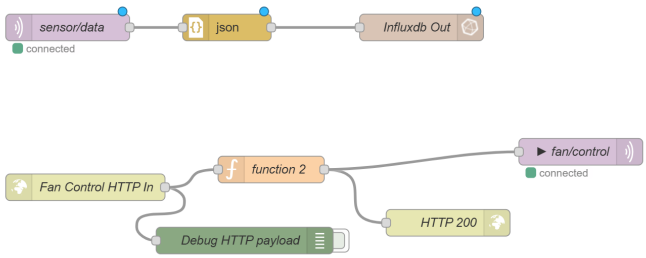

Figure 1: Node-Red flow configuration

Before hitting "Deploy", double click each node to edit. Make sure of the following:

MQTT In Node: `sensor/data`

- **Server:** MQTT Docker
  - **Name:** MQTT Docker
  - **Server:** `<ip_address>` **Port:** 1883
  - **Connect Automatically**
  - **Protocol:** MQTT V3.1.1
- **Topic:** `sensor/data`
- **Output format:** a parsed JSON object

JSON Node

- **Action:** Always convert to JavaScript object
- **Property:** msg. payload

InfluxDB Out Node

- **Measurement:** `environment`
- **Server:**
  - **Database:** `sensor_data`
  - **InfluxDB Version:** 1.x
  - **Host:** `<ip_address>`
  - **Port:** 8086

HTTP In Node: `/fan`

- **Method:** GET
- **URL:** `/fan`

Function Node:

- **On Message:**

```
// msg.req.query.duty contains the numeric duty from the URL
const duty = parseInt(msg.req.query.duty, 10);
if (!isNaN(duty) && duty >= 0 && duty <= 100) {
   msg.payload = duty;
} else {
    node.error("Invalid duty: " + msg.req.query.duty);
    msg.payload = 0;
}
return msg;
```

HTTP Response Node:

- **Status Code:** 200

MQTT Out Node:

- **Server:** Mosquitto
  - **Name:** Mosquitto
  - **Server:** `mosquitto` **Port:** 1883
  - **Connect Automatically**
  - **Protocol:** MQTT V3.1.1
- **Topic:** `fan/control`
- **QoS:** 0

Hit "Deploy", the MQTT in and out nodes should go from red: Disconnected to green: Connected in a few seconds.

2.3.1 Debugging:

Return to the Raspberry Pi terminal, navigate to the IOTstack folder, and run the following list of commands to check that data are being written to the database:

```
docker exec -it influxdb influx
USE sensor_data
show measurements
select * from environment
quit
```

If nothing shows, it means that the data is not reaching Influxdb. Run the following command to subscribe to the MQTT topic and receive the transmitted data:

```
docker exec -it mosquitto mosquitto_sub -v -t /sensor/data
```

If data appears, then the error is most likely due to node red. Start by checking the servers and the debug terminal (the third of the four square-button menu just below deploy) to identify the node at fault. Feel free to add a debug node to better visualize the issue.

2.4 Setting up Grafana

When entering Grafana for the first time, the username and password are "admin". Then it will prompt you to change the password.

2.4.1 Adding a Database

On the main menu, under "Welcome to Grafana", click "Add your first data source". Select Influxdb.
Under "HTTP", type the url:

```
http://<ip_address>:8086
```

Under InfluxDB Details, set the following:

- **Database:** `sensor_data`
- **HTTP Method:** GET
- **Min time interval:** 1s
- **Max series:** Choose accordingly

Click "Save & test", you should see the following message:

> datasource is working. 1 measurements found
> Next, you can start to visualize data by building a dashboard, or by querying data in the Explore view.

Select "building a dashboard".

2.4.2 Building a Dashboard

Click "Import a Dashboard", discard the current one, and import the JSON file named "Fan&Monitor".

To modify the dashboard, enter edit mode, drag the panels and change their sizes. To edit a specific panel, click the three dots and "Edit". The only relevant settings are the "Visualization" type (eg. Time Series vs Gauge), the Data Source (InfluxDB), the FROM (Default & "environment") and the SELECT field to select the dependent variable. Everything else is more a matter of aesthetics.

To control the fan, the panel should have the visualization type "Text", be in Markdown mode and the content should be the following (make sure to update the ip address):

```
### Fan Control
[Fan Off](http://<ip_address>:1880/fan?duty=0)
[Fan 25%](http://<ip_address>:1880/fan?duty=25)
[Fan 50%](http://<ip_address>:1880/fan?duty=50)
[Fan 75%](http://<ip_address>:1880/fan?duty=75)
[Fan 100%](http://<ip_address>:1880/fan?duty=100)
```

2.4.3 Debugging:

Make sure that the data source is configured correctly and contains data for Grafana to read (refer back to 2.3.1). A not-very-common bug is the uid of the influxdb datasource, which causes all panels to show a warning sign (this is accessable by opening the menu of any panel, hitting "Inspect" and selecting "Panel JSON"). The solution is to edit each panel individually and refresh it.

3 Adjustments to the main script

The only things that might need adjustment are the ADC channels being read and the interval between data acquisitions. For the latter, simply change the variable SAMPLE_INTERVAL at the top of the script. To adjust the ADC channels, look for the initialization of the ADC128D818 class within the sensor_loop() function and change the list of channels (0-7):

```
def sensor_loop():

    #Thread target: read sensors at fixed intervals,
    #build JSON payload, and publish via MQTT.

    sensor = BME280()
    adc = ADC128D818(channels=(6, 7))
    next_time = time.time()
    bme_errors = 0

(...)
```

Nevertheless, the script has a nice amount of commenting to allow the user to understand and adapt it as they see fit.

NOTE: Everytime the script is updated after the implementation of the Daemon (next section) the latter should be updated by running:

```
sudo systemctl daemon-reload
sudo systemctl start fanmonitor
```

4 Setting up the Daemon

Saved a unit file using [test]

```
sudo nano /etc/systemd/system/fanmonitor.service
```

with the following content:

```
[Unit]
Description=Fan Monitor Script
After=network.target

[Service]
ExecStart=/usr/bin/python3 /home/pi/pps_cold_box.py
WorkingDirectory=/home/pi
StandardOutput=inherit
StandardError=inherit
Restart=always
User=pi

[Install]
WantedBy=multi-user.target
```

Adjust the following settings if need be:

ExecStart: Full path to the Python interpreter and then to the script.
User=pi & Working Directory: Run under the pi user
Reload, Enable, and Start the Service:

```
sudo systemctl daemon-reload
sudo systemctl enable fanmonitor
sudo systemctl start fanmonitor
```

Check that it is active (running):

```
sudo systemctl status fanmonitor
```

Tail live logs to ensure sensor publishes and fan-control messages are received:

```
sudo journalctl -u fanmonitor -f
```

5 Access Influxdb as a pandas dataframe

A secondary script titled "influx_reader.py" was also created as an option to facilitate the extraction and analysis of data. It was created to be versatile; depending on the data and data that need to be extracted, as well as its format. To use it, install the following packages:

```
sudo apt install python3-pandas
sudo apt install python3-influxdb
sudo apt install python3-tabulate
sudo apt install python3-pyarrow
```

By default, the program prints a table with the data requested, but there are options to select JSON, CSV or Feather formats, simply adjust the user configuration at the beginning of the sript:

```
# -- USER CONFIG ----------------------------------------
HOST = "localhost"
PORT = 8086
DATABASE = "sensor_data"
MEASUREMENT = "environment"

EXPORT_CSV = False # set True to extract as CSV
EXPORT_JSON = False # set True to extract as JSON
EXPORT_FEATHER = False # set True to extract as Feather

CSV_OUTFILE = "output.csv"
JSON_OUTFILE = "output.json"
FEATHER_OUTFILE= "output.feather"
#-------------------------------------------------------
```

It is also quite straightforward to adjust the type of data extracted by changing the influx query, for example:

```
# Connect & fetch
client = InfluxDBClient(host=HOST, port=PORT, database=DATABASE)

q = (
  f'SELECT time, temperature, humidity FROM "{MEASUREMENT}" '
  f"WHERE time >= '{start}' AND time <= '{end}'"
)
```

References

[1] Bosch Sensortec GmbH. (2024). *BME280 Data sheet* (Document revision 1.24, BST-BME280-DS001-24). Retrieved July 16, 2025, from `https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bme280/BST-BME280-DS001-24.pdf`

[2] Texas Instruments Inc. (2015). *ADC128D818 12-Bit, 8-Channel, ADC System Monitor With Temperature Sensor, Internal–External Reference, and I2C Interface* (SNAS483F–FEBRUARY 2010–REVISED AUGUST 2015). Retrieved July 16, 2025, from `https://www.ti.com/product/ADC128D818`

[3] TE Connectivity Measurement Specialties. (2020). *PTFD102A1A0 PT1000, 2.0 × 5.0 A Industrial Temperature Sensor* (Version 2 10/2020). Retrieved July 16, 2025, from `https://pt.mouser.com/ProductDetail/Measurement-Specialties/PTFD102A1A0?qs=bvCpHSZmQjl6myCYnXHj%2FA%3D%3D`