The idea is to set up an IOT server in a Raspberry. The requirements for such a system are:

1. It must be able to receive, process and store MQTT messages,
2. It must provide a tool to program what to do with those messages
3. It must provide a way to store the data received.
4. In addition, It will also provide an interface to graphically show the stored data
5. And a way to access and manage the server components from internet.

The first thing I did was to select the different components, the limitations to this selection are: they must be open source, easy to manage and can run in a container in a Raspberry.

Another requirement was that every component could be set up as a container. I decided to install each single component in a separate container because it is an easy way to give independency to each subsystem. Each component runs in a separate container, so it is possible to start, stop, reconfigure, update re-install etc. without interfering the others. It also facilitates migrating from Raspberry to cloud. Containers are transportable easily (may be not directly) from one infrastructure to a quite different one. To simplify container usage, I will use **docker** as the container management software as it is widely implemented, easy to use and really robust.

With these requirements, and after some research, I selected the following ones:

1. **Mosquitto** as MQTT Broker. Mosquitto is widely used, stable, easy to configure and gives all the features I needed (and I likely will need in the future).
2. **Node-red**. This program is an easy way to link MQTT queues, devices and software components with a flow editor. Really friendly to use, scalable in functionality, and there are many blocks already developed but not limited to existing elements, as you can program easily your own block to adapt to your particular needs.
3. **Influxdb**. This database is incredible fast, simple and specially adapted to time series data. Has some features like policies and aggregations that are very useful. Furthermore, it is very easy to query, accessible with CURL and also there is a Python module that eases the integration of stored data with other applications.
4. **Grafana**. A powerful tool to visualize data. Preparing graphics to visualize stored data is a child's play. It provides many formats and timescales can be changed interactively. Also, it integrates smoothly with influxdb.
5. **Nginx**. Nginx is a very complete piece of software that can be used to many things related to route, balance and mask workload. In this case, I use it only as a reverse proxy, but could easily be scaled as a load balancer.
6. **Telegraf**. In addition, I installed telegraf to monitor the Raspberry itself. I store load data in influxdb and view it graphically with Grafana. Incredibly simple to install, configure and maintain.

I will also use **docker-compose** to simplify image management dependencies and parameters used to start each container.

So .... let's start.

# Installing Docker

First of all, as usual, be sure that your Raspberry is at latest level. As sometimes the process is long, I run update and upgrade in the same line with 'y' (yes) answered to all prompts:

**sudo apt-get update -y && sudo apt-get upgrade -y**

It is also advisable to set a fixed IP address; to do so, edit /etc/dhcpcd.conf and add the lines (for an ethernet interface):

interface eth0

static ip\_address=aa.bb.cc.dd.ee/24

static routers=rr.ss.tt.uu

static domain\_name\_servers=nn.mm.oo.pp qq.rr.ss.tt

and reboot

**sudo reboot now**

Now, install docker; first the necessary (transport) packages:

**sudo apt install -y apt-transport-https ca-certificates software-properties-common**

and then add docker DPG key

**curl -fsSL** [**https://download.docker.com/linux/debian/gpg**](https://download.docker.com/linux/debian/gpg) **| sudo apt-key add -**

and Docker repository

**echo "deb [arch=armhf]** [**https://download.docker.com/linux/debian**](https://download.docker.com/linux/debian) **\**

**$(lsb\_release -cs) stable" | \**

**sudo tee /etc/apt/sources.list.d/docker.list**

As there is a new repository, we need an update

**sudo apt update**

and install docker

**sudo apt install -y docker-ce**

Finally, add your own user (this is Pi as default user or the one you created if you did it) to docker group

**sudo usermod -aG docker MYUSERID**

Starting, stopping and updating docker images is tedious and sometime requires long commands. To simplify these tasks and to manage dependencies I recommend using **docker-compose**. It helps to control the behavior of your container using a plain text (yaml) file. Docker-compose is a tool written in python, so the easiest way to install it is with pip (it is the Python Package Management System), so we begin installing Python pip:

**sudo apt-get -y install python-pip**

once installed, we install docker-compose using pip:

**sudo pip install docker-compose**

Now, create a directory that will be used for all this stuff and a subdirectory for each of the components. I will call it /IOTServer, but you can choose any other name and parent directory:

**sudo mkdir /IOTServer**

**sudo chown MYUSERID:MYUSERID /IOTServer**

**cd /IOTServer**

**mkdir grafana**

**mkdir influxdb**

**mkdir mqtt**

**mkdir node-red**

**mkdir portainer**

**mkdir telegraf**

Copy the configuration file docker-compose.yaml



to /IOTServer directory. We are going to use it to download all the required images (this is one of the features of docker-compose):

**docker-compose pull**

This will last for a while, as all the container images are downloaded to the Raspberry. When pull process is finished, it is time to configure each component:

# Installing and configuring containers

## Influxdb

First, we have to create a configuration file. We do it by starting the container with 'config' option, we use -rm option in docker run to delete the container once it has finished.

**docker run --rm influxdb influxd config > /IOTServer/influxdb/influxdb.conf**

Now, start the influxdb for the first time to create a database ('admin' is a suggestion; pay attention to password and database name). When output stabilizes, and database creation has finished, you may stop it with control-C.

**docker run --rm -v /IOTServer/influxdb/influxdb.conf:/etc/influxdb/influxdb.conf -v /IOTServer/influxdb:/var/lib/influxdb -e INFLUXDB\_DB=mydatabasename -e INFLUXDB\_ADMIN\_USER=admin -e INFLUXDB\_ADMIN\_PASSWORD=the\_password influxdb -config /etc/influxdb/influxdb.conf /init-influxdb.sh**

To start influxdb:

**docker-compose up -d influxdb**

## Mosquitto

Create a file named /IOTServer/mqtt/conf/mosquitto.conf with the following contents:

persistence true

persistence\_location /mqtt/data/

allow\_anonymous true

#user mosquitto

# Port to use for the default listener.

port 1883

log\_dest stdout

#listener 9001

#protocol websockets

Mosquitto is now ready to start

### NODE-RED

start Node-red container in background:

**docker-compose up -d node-red**

Edit /IOTServer/node-red/settings.js and look for (if you don't have rw access give it to the directory) and uncomment this section:

// Securing Node-RED

// -----------------

// To password protect the Node-RED editor and admin API, the following

// property can be used. See <http://nodered.org/docs/security.html> for details.

adminAuth: {

type: "credentials",

users: [{

username: "user admin",

password: "a hash uninteligible",

permissions: "\*"

}]

}

To create a valid hash, do (put the password you want to use instead of 'your-password':

**docker exec -it node-red node -e "console.log(require('bcryptjs').hashSync(process.argv[1], 8));" your-password**

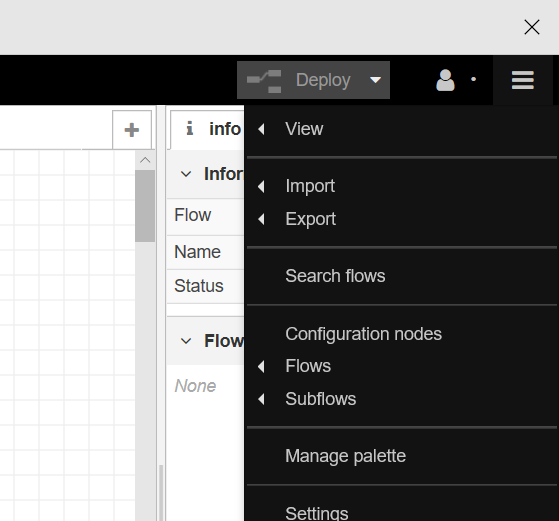
Copy and paste the result in password field of /IOTServer/node-redsettings.js

and restart node-red

**docker restart node-red**

with a Web browser, in <http://[My_IP_Address]:1880> node-red should ask for userid and password. Once connected you have an empty area to work with.

Remember that if you are going to insert data in influxd database, you will need to install the influxdb module: Go to manage palette,



then go to install tab and there look for influxdb and install it.

## PORTAINER

Portainer is a tool that helps you navigate, inspect …etc. through dockers with a web browser instead of using the command line. It should have been downloaded when you did docker-compose pull. Let’s start it for the first time by:

**docker run -d -p 9000:9000 -v /var/run/docker.sock:/var/run/docker.sock \**

**--restart always --name portainer portainer/portainer -H \**

**unix:///var/run/docker.sock**

Now Portainer is started and listening on port 9000. To navigate it just open a web browser on [**http://[My\_IP\_Address]:9000**](http://[My_IP_Address]:9000)

First time, you will be asked for a userID and password. The ones you put are set for the future.

## TELEGRAF

I use Telegraf to collect workload metrics and store them in an influxdb database. Find below a simplified (reduced) configuration file. Before using it, don't forget to make a copy of the original telegraf.conf and to substitute My\_IP\_Address with your own ip address in the file provided.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* falta telegraf.conf \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Congratulations! You have set up an IOT server**

... but, its only addressable from inside your network. To access it from internet, it's necessary to do some more things.

## Domain registering

As it is not probable that you have a static address in internet, you need to setup a Dynamic DNS. This is a server that publishes the name you register with the IP address that you provide and updates it as the address change (you also have to provide the updated address).

To accomplish this feature, I use duckdns (<https://www.duckdns.org>): it is free and works fine. Just connect to <https://www.duckdns.org> and register the domain name you want. Prior to registering the domain name, you have to log in. It is possible to do it with your Reddit, Github, Google ...etc. account. After you register your domain, you get a token (if you forget it, you can find it in <https://www.duckdns.org/install.jsp?tab=pi&domain=YOURDOMAIN> ) that will be needed in next steps. To tell duckdns your IPaddress, we will use a simple script. First, create the directory:

**cd /IOTServer**

**mkdir duckdns**

**cd duckdns**

and create the script publish\_ip.sh with a single line:

echo url="<https://www.duckdns.org/update?domains=MYDOMAIN&token=YOURTOKEN&ip=>" | curl -k -o /IOTServer/duckdns/publish\_ip.log -K -

make it executable and add it to crontab:

**chmod 700 publish\_ip.sh**

**crontab -e**

add to the end (run every 5 minutes)

\*/5 \* \* \* \* /IOTServer/duckdns/publish\_ip.sh >/dev/null 2>&1

You are telling duckdns what is your IP address, so it can find refer that address with the name you are providing.

## PORT FORWARDING

You probably have your raspberry behind a router that translates (NATs) your internal IP address into a public one. To make your Raspberry addressable, what we are going to do is to allow traffic HTTP and HTTPS to the Raspberry and then reroute it to the corresponding service using nginx. Port forwarding in your router is usually a simple task, you only have to configure on it to route 80 to [My\_IP\_Address:80] and 443 to [My\_IP\_Address:443].

To implement SSL, I use certbot and letsencrypt:

First of all, stop all services:

**docker-compose down**

Then download and install certbot:

**cd /IOTServer**

**wget** [**https://dl.eff.org/certbot-auto**](https://dl.eff.org/certbot-auto)

**chmod 755 certbot-auto**

**./certbot-auto certonly --standalone --preferred-challenges http-01 --email mymailname@mailserver -d MYDOMAIN.duckdns.org**

I can also set a certificate to every service:

**./certbot-auto certonly --standalone --preferred-challenges http-01 --email mymailname@mailserver -d nodered.MYDOMAIN.duckdns.org**

**./certbot-auto certonly --standalone --preferred-challenges http-01 --email mymailname@mailserver -d portainer.MYDOMAIN.duckdns.org**

**./certbot-auto certonly --standalone --preferred-challenges http-01 --email mymailname@mailserver -d grafana.MYDOMAIN.duckdns.org**

These certificates must be renewed monthly, so add a cron task:

\* \* 1 \* \* /IOTServer/renew\_cert.sh >/dev/null 2>&1

renew\_cert.sh is a simple script to stop nginx, renew the certificate and start nginx again.

edit renew\_cert.sh and add

#!/bin/bash

/usr/bin/docker stop nginx

/IOTServer/certbot-auto renew

/usr/bin/docker start nginx

and then

**chmod 700 renew\_cert.sh**

certificates are stored in

*Your certificate and chain have been saved at:*

*/etc/letsencrypt/live/SERVICE.MYDOMAIN.duckdns.org/fullchain.pem*

*Your key file has been saved at:*

*/etc/letsencrypt/live/SERVICE.MYDOMAIN.duckdns.org/privkey.pem*

## NGINX

We use nginx as reverse proxy, It was installed when running docker-compose pull. To configure it:

edit /IOTServer/nginx/nginx.conf and uncoment line 23:

*server\_names\_hash\_bucket\_size 64;*

Then copy the file default (attached)



to /IOTServer/nginx/site-confs/default (don't forget to change My\_IP\_Address with your own address)

## GRAFANA

Only need to create an empty configuration file

**touch /IOTServer/grafana/grafana.ini**

To start everything:

**docker-compose up -d**

Now you can access grafana at <http://[My_IP_Address]:3000>

first time it will set the admin user and password with the data you provide

## Autostart docker-compose

Now, it is necessary to start everything when Raspberry reboots. To do it we need to configure a service and to set it up.

**cd /etc/systemd/system**

**sudo nano docker-compose-opt.service**

add:

# /etc/systemd/system/docker-compose-opt.service

[Unit]

Description=Docker Compose Opt Service

Requires=docker.service

After=docker.service

[Service]

Type=oneshot

RemainAfterExit=yes

WorkingDirectory=/IOTServer

ExecStart=/usr/local/bin/docker-compose up -d

ExecStop=/usr/local/bin/docker-compose down

TimeoutStartSec=0

[Install]

WantedBy=multi-user.target

And we enable it

**sudo systemctl enable docker-compose-opt**

To test

**cd /IOTServer**

**docker-compose down**

**sudo reboot now**

on reboot, we check if everything is running:

**docker ps**

or navigating to <http://[My_IP_Address]:9000>

and navigating to <http://[My_IP_Address]:9000> or <http://MYSERVICE.MYDOMAIN.duckdns.org>

Now you have an IOTserver were to send data from your gauges, work with them and store if you want in an influxdb database.

# Configuring Node-red

Now you have your IOT server up and running, it is time to receive, transform and store the data that the meteo gauge is sending to it. As said at the beginning, node-red is a very friendly environment to program; there are some blocks especially useful when dealing with these kinds of projects, e.g.

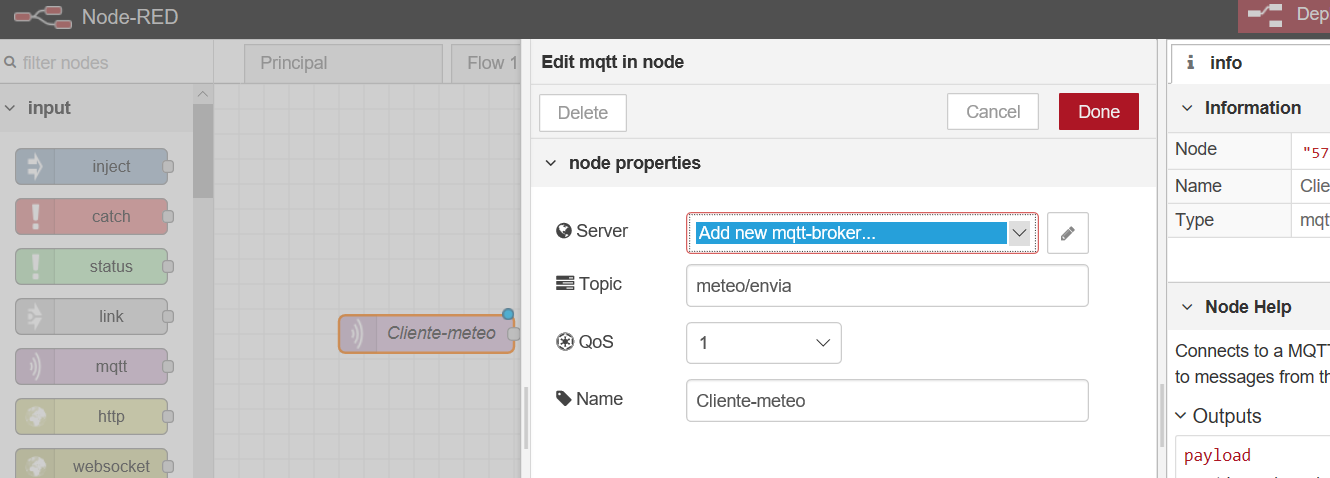
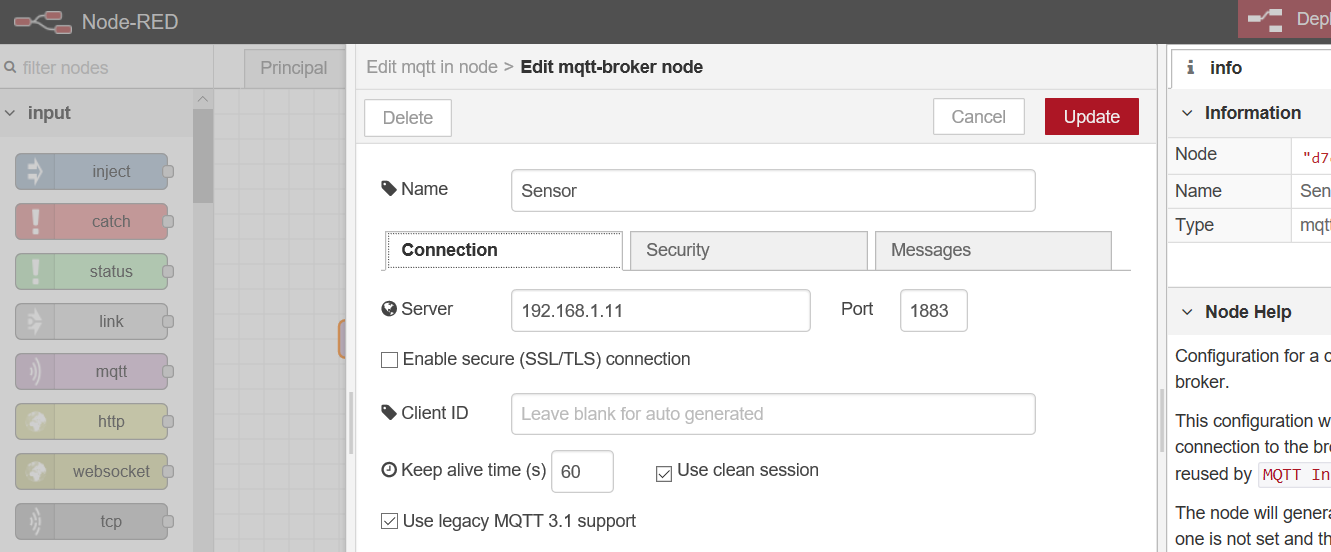
* **mqtt-in**. Connects to a MQTT broker and subscribes to messages from the specified topic.
* **JSON**. Converts between a JSON string and its JavaScript object representation, in either direction.
* **influxdb**. A simple output node to write values and tags to an influxdb measurement.

I will go through each of them to configure the required values.

To connect to your node-red server, open in a browser Go to <http://[My_IP_Address]:1880>. It will ask for userID and password, put the ones you defined when configuring node-red container. You will be presented a blank canvas, here is where we will define our flow.

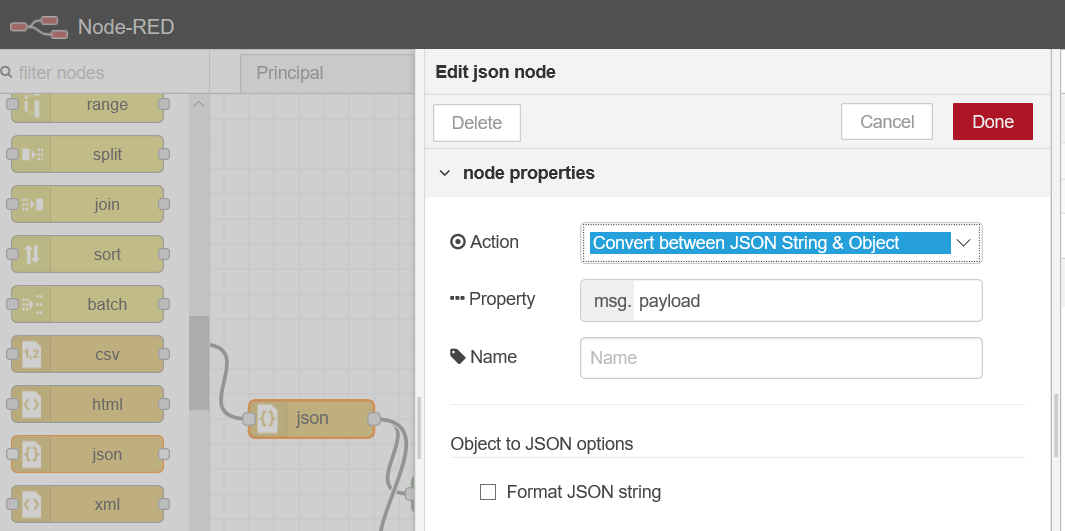
## Block mqtt in

We will begin with “mqtt in” block. This block will:

1.  **Connect to a mqtt broker and subscribe to the message topic that our client is using**. To subscribe , drag mqtt in node to the canvas and double click on it. In “Topic” write the topic that theclient is using (meteo/envia in this case), and a Nme for the node. This name is to represent the node so can be anyone. Regarding QoS, I recommend using ‘2’ that means that one and only one message will arrive . Next step is to
2. **Define a mqtt broker to connect to**. In the previous window, click on the pencil in the combo “Server”. Probably it will say something like ”add a new mqtt-broker”. You will get another window, where we will define the broker to connect. In “name”, put any name that has some meaning, e.g. “Sensor”. In Server, you have to put the local address of the mqtt broker. In this case, it will be the local address of the Raspberry. Por must be 1883, unless you changed it when configuring mosquito (not likely and not recommended). Check “use clean session”. For entry configurations, we will keep “Client ID” blank, so the node will generate a random ID, leave SSL/TLS unchecked and do not configure Security nor Messages tabs. Once you are familiar with mqtt, it will be time to secure connections.

After saving and deploying (there is a button in the upper right are of the main node-red window) if mosquito is running you will see a “connected” and a green square below mqtt node.

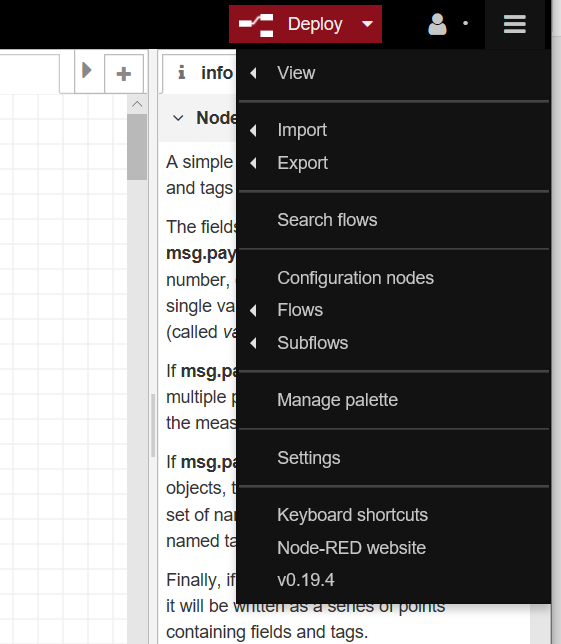
## Block JSON

This is a very simple one. It will convert a JSON string into a JSON object. Drag a JSON node for “function” area to the canvas and click on it.

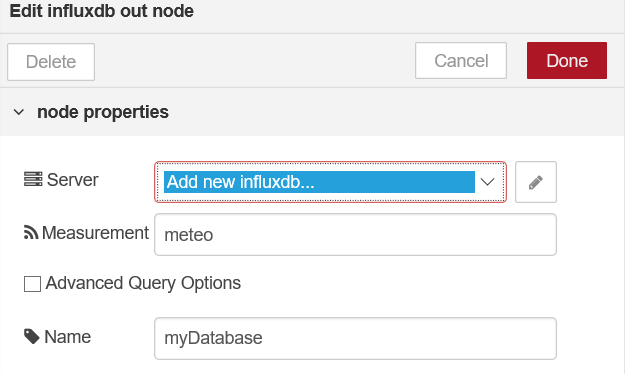
In “action”, select “convert between JSON String &object”. The property to convert is msg.payload (the message the client is sending), keep unchecked “Format JSON String” checkbox.

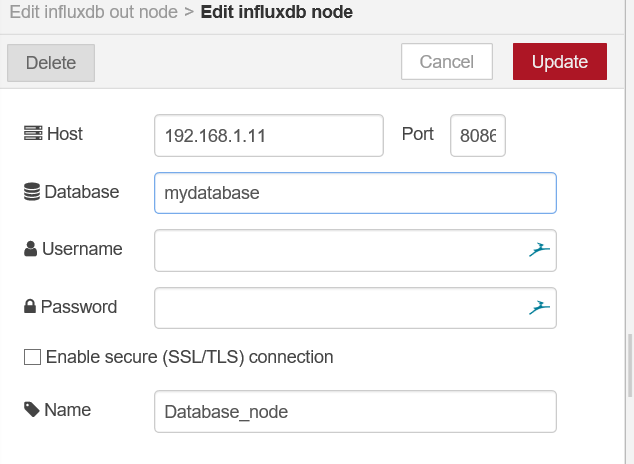
## Block influxdb

We will now configure a node to insert in a database the data sent that was received in a message stored in msg.payload and then formatted to a JSON object (and passed again in msg.payload within node-red).

First, you will probably need to install influxdb node. To do it, go to “manage palette” (in dropdown menu in upper right side of the window) and there, in “install” tab, look for influxdb. Once found, install it.

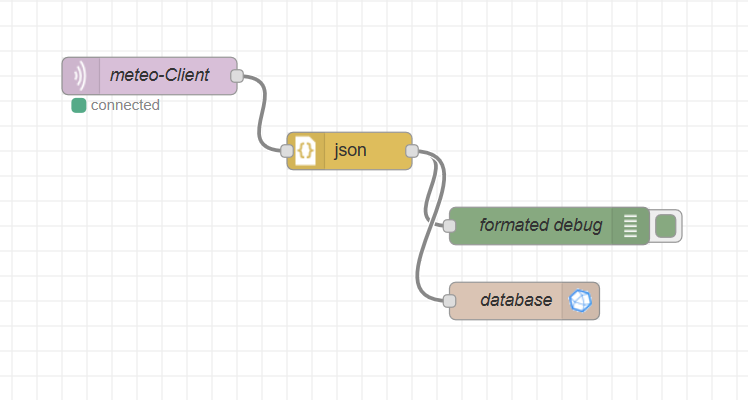
Once installed, you can select t for the left menu. As with previous nodes, drag in into the canvas and double click on it.

There is a server to set ( I will cover it soon), a measurement name (this is an influxdb terminology, it is somewhat like a table, where data is to be stored) and a node name, that is not relevant in configuration terms. Leave “Advanced Query options” unchecked and click on the pencil in “Server” (probably you will see there \* Add new influxdb\* ).

When clicked, you will get another window where to configure the database. In this window you must set the host, with the Raspberry IP address and the database name you defined when configuring influxdb.

## Summary

Find below these three blocks configured in node-red. Meteo-client represents the connection with the MQTT Broker. To check how received messages are being formatted, I configured a debug node (‘formatted debug’) which output goes to the debug window.

[{"id":"54133c00.4e5ce4","type":"tab","label":"Flow 2","disabled":false,"info":""},{"id":"5364ed36.6c7294","type":"mqtt in","z":"54133c00.4e5ce4","name":"meteo-Client","topic":"meteo/envia","qos":"1","broker":"d7c0ce0a.414658","x":170,"y":200,"wires":[["711d918b.3ee22"]]},{"id":"fed2a0b5.cd474","type":"debug","z":"54133c00.4e5ce4","name":"formated debug","active":true,"tosidebar":true,"console":false,"tostatus":false,"complete":"payload","x":490,"y":320,"wires":[]},{"id":"d3ae7cf.1bb068","type":"influxdb out","z":"54133c00.4e5ce4","influxdb":"10f94ce.9562bb3","name":"database","measurement":"meteo","precision":"","retentionPolicy":"","x":470,"y":380,"wires":[]},{"id":"711d918b.3ee22","type":"json","z":"54133c00.4e5ce4","name":"","property":"payload","action":"","pretty":false,"x":330,"y":260,"wires":[["fed2a0b5.cd474","d3ae7cf.1bb068"]]},{"id":"d7c0ce0a.414658","type":"mqtt-broker","z":"","name":"Sensor","broker":"192.168.1.11","port":"1883","clientid":"","usetls":false,"compatmode":true,"keepalive":"60","cleansession":true,"birthTopic":"","birthQos":"1","birthPayload":"{}","closeTopic":"","closeQos":"0","closePayload":"","willTopic":"","willQos":"0","willPayload":""},{"id":"10f94ce.9562bb3","type":"influxdb","z":"","hostname":"192.168.1.11","port":"8086","protocol":"http","database":"yourdatabase","name":"yourdatabase","usetls":false,"tls":""}]

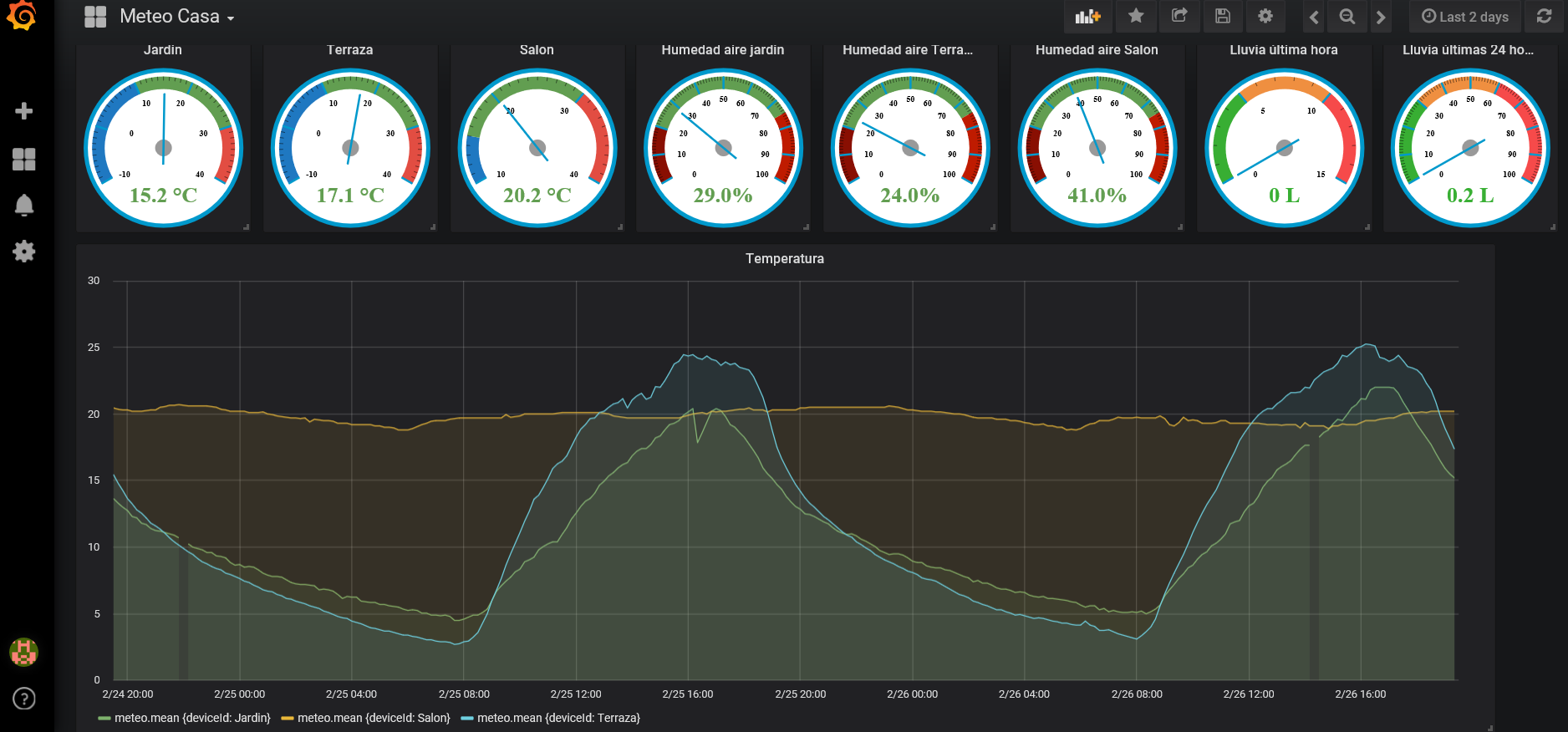
You might need to change database name (and password) and also the broker IP address.

Meteo-client represents the connection with the MQTT Broker. To check how messages are being formatted, I configured a debug node (‘formatted debug’) which output goes to the debug window.

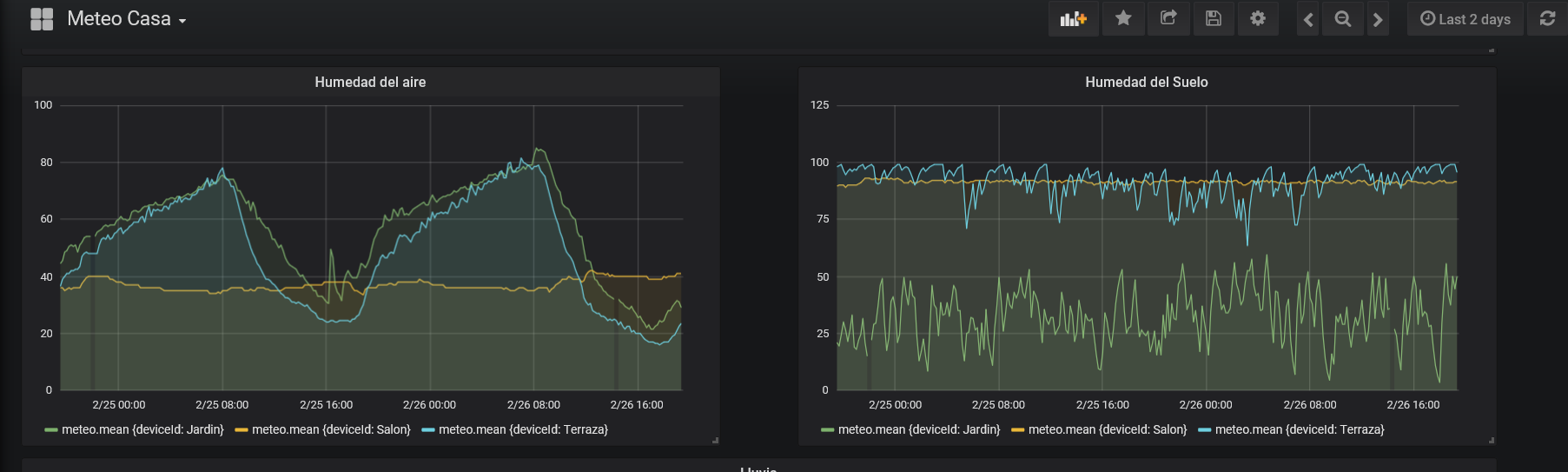
# Grafana customization

As data is stored in influxdb database, it can be visualized with Grafana. Grafana is easy to customize so you can have personalized charts, there is plenty of documentation and shared charts. To have some starting point, my main charts for meteo gauges are the following ones (I have three gauges, only one of them has rain gauge):

General dashboard and temperatures



Air humidity and soil moisture



Rain (15 min and accumulated)



### In JSON format:

The JSON object that represents this dashboard is very long, should you want to load it, here it is attached



# Done!!!

After this tutorial, you should have a meteo station working. If you have comments, issues or whtever, do not hesitate contacting me

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

### file docker-compose.yaml

# Change My\_IP\_Address with your own address

version: '2.1'

# Change My\_IP\_Address with your own address

services:

influxdb:

container\_name: influxdb

image: influxdb

ports:

- 8086:8086

- 8088:8088

volumes:

- /IOTServer/influxdb/influxdb.conf:/etc/influxdb/influxdb.conf:ro

- /IOTServer/influxdb:/var/lib/influxdb

restart: on-failure

healthcheck:

test: ["CMD", "curl", "-sI", "<http://127.0.0.1:8086/ping>"]

interval: 30s

timeout: 1s

retries: 24

mosquitto:

container\_name: mosquitto

image: churruscat/mosquitto-rpi

user: "1001:1001"

ports:

- 1883:1883

volumes:

- /etc/localtime:/etc/localtime:ro

- /IOTServer/mqtt:/mqtt/config:ro

- /IOTServer/mqtt:/mqtt/data

restart: on-failure

portainer:

container\_name: portainer

image: portainer/portainer

ports:

- 9000:9000

volumes:

- /IOTServer/portainer:/data:rw

- /var/run/docker.sock:/var/run/docker.sock

restart: on-failure

nginx:

container\_name: nginx

image: lsioarmhf/nginx-armhf

ports:

- 80:80

- 443:443

volumes:

- /IOTServer/nginx:/config:rw

- /etc/letsencrypt:/etc/letsencrypt:ro

node-red:

container\_name: node-red

image: nodered/node-red-docker:rpi-v8

environment:

- http\_proxy=<http://My_IP_Address:1880>

- https\_proxy=<http://My_IP_Address:1880>

ports:

- 1880:1880

volumes:

- /IOTServer/node-red:/data

- /etc/localtime:/etc/localtime:ro

- /etc/timezone:/etc/timezone:ro

restart: on-failure

depends\_on:

mosquitto:

condition: service\_started

healthcheck:

test: ["CMD", "curl", "-f", "<http://127.0.0.1:1880>"]

interval: 30s

timeout: 10s

retries: 5

telegraf:

container\_name: telegraf

image: telegraf

volumes:

- /IOTServer/telegraf/telegraf.conf:/etc/telegraf/telegraf.conf:ro

- /var/run/docker.sock:/var/run/docker.sock:ro

- /sys:/rootfs/sys:ro

- /proc:/rootfs/proc:ro

- /etc:/rootfs/etc:ro

depends\_on:

influxdb:

condition: service\_started

grafana:

container\_name: grafana

image: churruscat/grafana-rpi

ports:

- 3000:3000

volumes:

- /IOTServer/grafana:/etc/grafana:rw

- /IOTServer/grafana:/var/lib/grafana:rw

restart: on-failure

depends\_on:

influxdb:

condition: service\_healthy

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

file /IOTServer/nginx/nginx/site-confs/default

# Change MYDOMAIN by your domain nam

# Redirect all http traffic to https

server {

listen 80;

return 301 <https://$host$request_uri>;

}

# GrafanaUpstream

upstream grafana {

server My\_IP\_Address:3000;

keepalive 32;

}

# Portainer Upstream

upstream portainer {

server My\_IP\_Address:9000;

keepalive 32;

}

# Node-RED Upstream

upstream node-red {

server My\_IP\_Address:1880;

keepalive 32;

}

server {

listen 443 ssl http2;

root /config/www;

index index.html index.htm index.php;

server\_name portainer.MYDOMAIN.duckdns.org;

client\_max\_body\_size 0;

ssl\_certificate /etc/letsencrypt/live/portainer.MYDOMAIN.duckdns.org/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/portainer.MYDOMAIN.duckdns.org/privkey.pem;

ssl\_protocols TLSv1.1 TLSv1.2;

ssl\_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";

add\_header Strict-Transport-Security "max-age=31536000; includeSubdomains";

ssl\_prefer\_server\_ciphers on;

location / {

proxy\_pass <http://portainer/>;

proxy\_http\_version 1.1;

proxy\_set\_header Connection "";

}

location /api/websocket/ {

proxy\_set\_header Upgrade $http\_upgrade;

proxy\_set\_header Connection "upgrade";

proxy\_http\_version 1.1;

proxy\_pass <http://portainer/api/websocket/>;

}

}

server {

listen 443 ssl http2;

root /config/www;

index index.html index.htm index.php;

server\_name node-red.MYDOMAIN.duckdns.org;

client\_max\_body\_size 0;

ssl\_certificate /etc/letsencrypt/live/node-red.MYDOMAIN.duckdns.org/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/node-red.MYDOMAIN.duckdns.org/privkey.pem;

ssl\_protocols TLSv1.1 TLSv1.2;

ssl\_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";

add\_header Strict-Transport-Security "max-age=31536000; includeSubdomains";

ssl\_prefer\_server\_ciphers on;

location / {

proxy\_pass <http://nodered/>;

proxy\_http\_version 1.1;

proxy\_set\_header Connection "";

}

}

server {

listen 443 ssl http2 default\_server;

server\_name MYDOMAIN.duckdns.org;

root /config/www/Dashboard;

index index.html index.htm index.php;

client\_max\_body\_size 0;

#SSL settings

ssl\_certificate /etc/letsencrypt/live/MYDOMAIN.duckdns.org/fullchain.pem;

ssl\_certificate\_key /etc/letsencrypt/live/MYDOMAIN.duckdns.org/privkey.pem;

ssl\_protocols TLSv1.1 TLSv1.2;

ssl\_ciphers "EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";

add\_header Strict-Transport-Security "max-age=31536000; includeSubdomains";

ssl\_prefer\_server\_ciphers on;

location / {

proxy\_pass <http://grafana/>;

proxy\_http\_version 1.1;

proxy\_set\_header Connection "";

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* telegraf.conf \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#Simplified (reduced) version of telegraf.conf

# Telegraf Configuration

#

# Telegraf is entirely plugin driven. All metrics are gathered from the

# declared inputs, and sent to the declared outputs.

#

# Plugins must be declared in here to be active.

# To deactivate a plugin, comment out the name and any variables.

#

# Use 'telegraf -config telegraf.conf -test' to see what metrics a config

# file would generate.

#

# Environment variables can be used anywhere in this config file, simply prepend

# them with $. For strings the variable must be within quotes (ie, "$STR\_VAR"),

# for numbers and booleans they should be plain (ie, $INT\_VAR, $BOOL\_VAR)

# Global tags can be specified here in key="value" format.

[global\_tags]

# dc = "us-east-1" # will tag all metrics with dc=us-east-1

# rack = "1a"

## Environment variables can be used as tags, and throughout the config file

# user = "$USER"

# Configuration for telegraf agent

[agent]

## Default data collection interval for all inputs

interval = "10s"

## Rounds collection interval to 'interval'

## ie, if interval="10s" then always collect on :00, :10, :20, etc.

round\_interval = true

## Telegraf will send metrics to outputs in batches of at most

## metric\_batch\_size metrics.

## This controls the size of writes that Telegraf sends to output plugins.

metric\_batch\_size = 1000

## For failed writes, telegraf will cache metric\_buffer\_limit metrics for each

## output, and will flush this buffer on a successful write. Oldest metrics

## are dropped first when this buffer fills.

## This buffer only fills when writes fail to output plugin(s).

metric\_buffer\_limit = 10000

## Collection jitter is used to jitter the collection by a random amount.

## Each plugin will sleep for a random time within jitter before collecting.

## This can be used to avoid many plugins querying things like sysfs at the

## same time, which can have a measurable effect on the system.

collection\_jitter = "0s"

## Default flushing interval for all outputs. You shouldn't set this below

## interval. Maximum flush\_interval will be flush\_interval + flush\_jitter

flush\_interval = "10s"

## Jitter the flush interval by a random amount. This is primarily to avoid

## large write spikes for users running a large number of telegraf instances.

## ie, a jitter of 5s and interval 10s means flushes will happen every 10-15s

flush\_jitter = "0s"

## By default or when set to "0s", precision will be set to the same

## timestamp order as the collection interval, with the maximum being 1s.

## ie, when interval = "10s", precision will be "1s"

## when interval = "250ms", precision will be "1ms"

## Precision will NOT be used for service inputs. It is up to each individual

## service input to set the timestamp at the appropriate precision.

## Valid time units are "ns", "us" (or "µs"), "ms", "s".

precision = ""

## Logging configuration:

## Run telegraf with debug log messages.

debug = false

## Run telegraf in quiet mode (error log messages only).

quiet = false

## Specify the log file name. The empty string means to log to stderr.

logfile = ""

## Override default hostname, if empty use os.Hostname()

hostname = ""

## If set to true, do no set the "host" tag in the telegraf agent.

omit\_hostname = false

###############################################################################

# OUTPUT PLUGINS #

###############################################################################

# Configuration for sending metrics to InfluxDB

[[outputs.influxdb]]

## The full HTTP or UDP URL for your InfluxDB instance.

##

## Multiple URLs can be specified for a single cluster, only ONE of the

## urls will be written to each interval.

# urls = ["unix:///var/run/influxdb.sock"]

# urls = ["udp://127.0.0.1:8089"]

urls = ["<http://My_IP_Address:8086>"]

## The target database for metrics; will be created as needed.

database = "telegraf"

## If true, no CREATE DATABASE queries will be sent. Set to true when using

## Telegraf with a user without permissions to create databases or when the

## database already exists.

skip\_database\_creation = false

# skip\_database\_creation = true

## Name of existing retention policy to write to. Empty string writes to

## the default retention policy.

# retention\_policy = ""

## Write consistency (clusters only), can be: "any", "one", "quorum", "all"

# write\_consistency = "any"

## Timeout for HTTP messages.

# timeout = "5s"

## HTTP Basic Auth

# username = "telegraf"

# password = "metricsmetricsmetricsmetrics"

## HTTP User-Agent

# user\_agent = "telegraf"

## UDP payload size is the maximum packet size to send.

# udp\_payload = 512

## Optional SSL Config

# ssl\_ca = "/etc/telegraf/ca.pem"

# ssl\_cert = "/etc/telegraf/cert.pem"

# ssl\_key = "/etc/telegraf/key.pem"

## Use SSL but skip chain & host verification

# insecure\_skip\_verify = false

## HTTP Proxy override, if unset values the standard proxy environment

## variables are consulted to determine which proxy, if any, should be used.

# http\_proxy = "<http://corporate.proxy:3128>"

## Additional HTTP headers

# http\_headers = {"X-Special-Header" = "Special-Value"}

## HTTP Content-Encoding for write request body, can be set to "gzip" to

## compress body or "identity" to apply no encoding.

# content\_encoding = "identity"

## When true, Telegraf will output unsigned integers as unsigned values,

## i.e.: "42u". You will need a version of InfluxDB supporting unsigned

## integer values. Enabling this option will result in field type errors if

## existing data has been written.

# influx\_uint\_support = false

###############################################################################

# PROCESSOR PLUGINS #

###############################################################################

###############################################################################

# AGGREGATOR PLUGINS #

###############################################################################

###############################################################################

# INPUT PLUGINS #

###############################################################################

# Read metrics about cpu usage

[[inputs.cpu]]

## Whether to report per-cpu stats or not

percpu = true

## Whether to report total system cpu stats or not

totalcpu = true

## If true, collect raw CPU time metrics.

collect\_cpu\_time = false

## If true, compute and report the sum of all non-idle CPU states.

report\_active = false

# Read metrics about disk usage by mount point

[[inputs.disk]]

## By default stats will be gathered for all mount points.

## Set mount\_points will restrict the stats to only the specified mount points.

# mount\_points = ["/"]

## Ignore mount points by filesystem type.

ignore\_fs = ["tmpfs", "devtmpfs", "devfs"]

# Read metrics about disk IO by device

[[inputs.diskio]]

## By default, telegraf will gather stats for all devices including

## disk partitions.

## Setting devices will restrict the stats to the specified devices.

# devices = ["sda", "sdb", "vd\*"]

## Uncomment the following line if you need disk serial numbers.

# skip\_serial\_number = false

#

## On systems which support it, device metadata can be added in the form of

## tags.

## Currently only Linux is supported via udev properties. You can view

## available properties for a device by running:

## 'udevadm info -q property -n /dev/sda'

# device\_tags = ["ID\_FS\_TYPE", "ID\_FS\_USAGE"]

#

## Using the same metadata source as device\_tags, you can also customize the

## name of the device via templates.

## The 'name\_templates' parameter is a list of templates to try and apply to

## the device. The template may contain variables in the form of '$PROPERTY' or

## '${PROPERTY}'. The first template which does not contain any variables not

## present for the device is used as the device name tag.

## The typical use case is for LVM volumes, to get the VG/LV name instead of

## the near-meaningless DM-0 name.

# name\_templates = ["$ID\_FS\_LABEL","$DM\_VG\_NAME/$DM\_LV\_NAME"]

# Get kernel statistics from /proc/stat

[[inputs.kernel]]

# no configuration

# Read metrics about memory usage

[[inputs.mem]]

# no configuration

# Get the number of processes and group them by status

[[inputs.processes]]

# no configuration

# Read metrics about swap memory usage

[[inputs.swap]]

# no configuration

# Read metrics about system load & uptime

[[inputs.system]]

# no configuration

# # Read metrics about docker containers

[[inputs.docker]]

# ## Docker Endpoint

# ## To use TCP, set endpoint = "tcp://[ip]:[port]"

# ## To use environment variables (ie, docker-machine), set endpoint = "ENV"

endpoint = "unix:///var/run/docker.sock"

# endpoint = "/var/run/docker.sock"

#

# ## Set to true to collect Swarm metrics(desired\_replicas, running\_replicas)

# gather\_services = false

#

# ## Only collect metrics for these containers, collect all if empty

container\_names = []

#

# ## Containers to include and exclude. Globs accepted.

# ## Note that an empty array for both will include all containers

container\_name\_include = []

container\_name\_exclude = []

#

# ## Container states to include and exclude. Globs accepted.

# ## When empty only containers in the "running" state will be captured.

container\_state\_include = []

# # container\_state\_exclude = []

#

# ## Timeout for docker list, info, and stats commands

# timeout = "5s"

#

# ## Whether to report for each container per-device blkio (8:0, 8:1...) and

# ## network (eth0, eth1, ...) stats or not

perdevice = false

# ## Whether to report for each container total blkio and network stats or not

total = true

# ## Which environment variables should we use as a tag

# ##tag\_env = ["JAVA\_HOME", "HEAP\_SIZE"]

#

# ## docker labels to include and exclude as tags. Globs accepted.

# ## Note that an empty array for both will include all labels as tags

docker\_label\_include = []

docker\_label\_exclude = []

#

# ## Optional SSL Config

# # ssl\_ca = "/etc/telegraf/ca.pem"

# # ssl\_cert = "/etc/telegraf/cert.pem"

# # ssl\_key = "/etc/telegraf/key.pem"

# ## Use SSL but skip chain & host verification

# # insecure\_skip\_verify = false

# # Read metrics about network interface usage

[[inputs.net]]

# ## By default, telegraf gathers stats from any up interface (excluding loopback)

# ## Setting interfaces will tell it to gather these explicit interfaces,

# ## regardless of status.

# ##

# # interfaces = ["eth0"]

# ##

# ## On linux systems telegraf also collects protocol stats.

# ## Setting ignore\_protocol\_stats to true will skip reporting of protocol metrics.

# ##

# # ignore\_protocol\_stats = false

# ##

# # Read TCP metrics such as established, time wait and sockets counts.

[[inputs.netstat]]

# # no configuration

# # Collect kernel snmp counters and network interface statistics

[[inputs.nstat]]

# ## file paths for proc files. If empty default paths will be used:

# ## /proc/net/netstat, /proc/net/snmp, /proc/net/snmp6

# ## These can also be overridden with env variables, see README.

# proc\_net\_netstat = "/proc/net/netstat"

# proc\_net\_snmp = "/proc/net/snmp"

# proc\_net\_snmp6 = "/proc/net/snmp6"

# ## dump metrics with 0 values too

# dump\_zeros = true

###############################################################################

# SERVICE INPUT PLUGINS #

###############################################################################