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. **Mqttdbs** is a ‘dockerized’ python program that reads from an MQTT brokers, stores the data in an influxdb database and, optionally, resends every record to another MQTT server
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 mqttdbs**

**mkdir influxdb**

**mkdir mqtt**

**mkdir portainer**

**mkdir telegraf**

**mkdir log**

Create a file docker-compose.yaml (that will hold docker-compose configuration) in /IOTServer directory with following contents:

*# Docker-compose.yaml*

*version: '3.6'*

*services:*

*grafana:*

*container\_name: grafana*

*image: grafana/grafana*

*ports:*

*- 3000:3000*

*volumes:*

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

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

*restart: on-failure*

*depends\_on:*

*- influxdb*

*influxdb:*

*container\_name: influxdb*

*image: influxdb:1.8*

*ports:*

*- 8086:8086*

*volumes:*

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

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

*#network\_mode: "host"*

*restart: on-failure*

*healthcheck:*

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

*interval: 60s*

*timeout: 1s*

*retries: 10*

*telegraf:*

*container\_name: telegraf*

*image: arm32v7/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*

*mosquitto:*

*container\_name: mosquitto*

*image: eclipse-mosquitto*

*user: "1001:1001"*

*ports:*

*- 1883:1883*

*volumes:*

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

*- /IOTServer/mqtt:/mosquitto/config:ro*

*- /IOTServer/mqtt:/mosquitto/data:rw*

*- /IOTServer/log:/mosquitto/log:rw*

*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: arm32v7/nginx*

*ports:*

*- 80:80*

*- 443:443*

*volumes:*

*- /IOTServer/nginx:/etc/nginx:rw*

*- /IOTServer/log:/config/log/nginx*

*- /IOTServer/nginx/www:/opt/www*

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

*mqttdbs:*

*container\_name: mqttdbs*

*image: churruscat/mqttdbs:latest*

*volumes:*

*- /IOTServer/mqttdbs:/etc/mqttdbs:rw*

*depends\_on:*

*- mosquitto*

*- influxdb*

*restart: on-failure*

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

Run pull to download latest version of all the containers configured in docker-compose.yaml:

**cd /IOTServer**

**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

(\*) These instructions refer to influxdb version 1.8. If you own a Raspberry 4, be aware to not install the latest version (see docker-compose.yaml, influxdb section)

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:1.8 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=iotdb -e INFLUXDB\_ADMIN\_USER=admin -e INFLUXDB\_ADMIN\_PASSWORD=my\_password influxdb:1.8 -config /etc/influxdb/influxdb.conf /init-influxdb.sh**

To start influxdb in background mode:

**docker-compose up -d influxdb**

## MQTTDBS

Once downloaded (pulled) mqttdbs image, to start it it is only needed to setup a configuration file. Copy the below configuration file to /IOTServer/mqttdbs/mqttdbs.conf file. Change parameters when needed. Take into account that if there is an address configured in [*mqtt\_broker\_send* ] section, every received record will be resent to that address. This feature can be used to set gateway mqtt servers. Source code is in github.

*# mqttdbs.conf*

*#*

*[mqtt\_broker\_read]*

*address=mosquitto*

*userid=*

*password=*

*port=1883*

*subscribe\_topic=#*

*[mqtt\_broker\_send]*

*address=*

*userid=*

*password=*

*port=1883*

*publishTopic=meteo/envia*

*[log\_level]*

*log\_level=info*

*[influxdb\_V1]*

*dbname=iotdb*

*address=influxdb*

*userid=*

*password=*

## mosquitto

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

*persistence true*

*persistence\_location /mqtt/data/*

*allow\_anonymous true*

*# Port to use for the default listener.*

*port 1883*

*log\_dest stdout*

*#listener 9001*

*#protocol websockets*

Mosquitto is now ready to start

## 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.

## 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

## 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.

**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 probable that you do not 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=**](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 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 will facilitate you to access your configured services by name using only port 443. Instead of accessing MyDOMAIN.duckdns.org:port (e.g. 9000 for portainer) you can set SERVICE.MYDOMAIN. duckdns.org (eg. Portainer instead SERVICE). 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;*

and add the default configuration file to /IOTServer/nginx/site-confs/myConfig.conf (attached)

*# Redirect all http traffic to https*

*server {*

*listen 80;*

*return 301 https://$host$request\_uri;*

*}*

*# ServiciosUpstream*

*upstream servicios {*

*server kmlserver:5000;*

*keepalive 32;*

*}*

*# GrafanaUpstream*

*upstream grafana {*

*server grafana:3000;*

*keepalive 32;*

*}*

*# Portainer Upstream*

*upstream portainer {*

*server portainer:9000;*

*keepalive 32;*

*}*

*#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# este es el raiz, que usare para GPS*

*#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*server {*

*#listen 80;*

*listen 443 ssl http2;*

*server\_name servicios.MYDOMAIN.es;*

*index index.html;*

*#auth\_basic "Username and Password Required";*

*#auth\_basic\_user\_file /etc/nginx/.htpasswd;*

*ssl\_certificate /etc/letsencrypt/live/servicios.MYDOMAIN.es/fullchain.pem;*

*ssl\_certificate\_key /etc/letsencrypt/live/servicios.myserver.es/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;*

*add\_header 'Access-Control-Allow-Origin' '\*' always;*

*location / {*

*auth\_basic "Username and Password Required";*

*auth\_basic\_user\_file /etc/nginx/.htpasswd;*

*index index.html;*

*#proxy\_set\_header 'Access-Control-Allow-Credentials' true;*

*}*

*location /gpstrack/ {*

*auth\_basic "Username and Password Required";*

*auth\_basic\_user\_file /etc/nginx/.htpasswd;*

*index gpstrack.html;*

*}*

*location /creakml/ {*

*proxy\_pass http://192.168.1.11:5000/creakml/;*

*auth\_basic "Username and Password Required";*

*auth\_basic\_user\_file /etc/nginx/.htpasswd;*

*#proxy\_hide\_header 'Access-Control-Allow-Origin';*

*proxy\_set\_header 'Access-Control-Allow-Origin' '\*';*

*}*

*location /subekml/ {*

*proxy\_pass http://192.168.1.11:5000/subekml/;*

*auth\_basic "Username and Password Required";*

*auth\_basic\_user\_file /etc/nginx/.htpasswd;*

*proxy\_set\_header 'Access-Control-Allow-Origin' '\*';*

*}*

*location /.well-known/acme-challenge/ {*

*index index.html;*

*}*

*}*

*#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*# Servicios de IOTServer (portainer, Grafana y nodered)*

*#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*server {*

*listen 443 ssl http2;*

*server\_name portainer.MYDOMAIN.es;*

*root /config/www;*

*index index.html index.htm index.php;*

*client\_max\_body\_size 0;*

*#SSL settings*

*ssl\_certificate /etc/letsencrypt/live/portainer.MYDOMAIN.es/fullchain.pem;*

*ssl\_certificate\_key /etc/letsencrypt/live/portainer.MYDOMAIN.es/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;*

*server\_name grafana.MYDOMAIN.es;*

*root /config/www/Dashboard;*

*index index.html index.htm index.php;*

*client\_max\_body\_size 0;*

*#SSL settings*

*ssl\_certificate /etc/letsencrypt/live/grafana.MYDOMAIN.es/fullchain.pem;*

*ssl\_certificate\_key /etc/letsencrypt/live/grafana.MYDOMAIN.es/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 "";*

*proxy\_set\_header Upgrade $http\_upgrade;*

*proxy\_set\_header Connection "upgrade";*

*}*

*}*

## 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]:port>

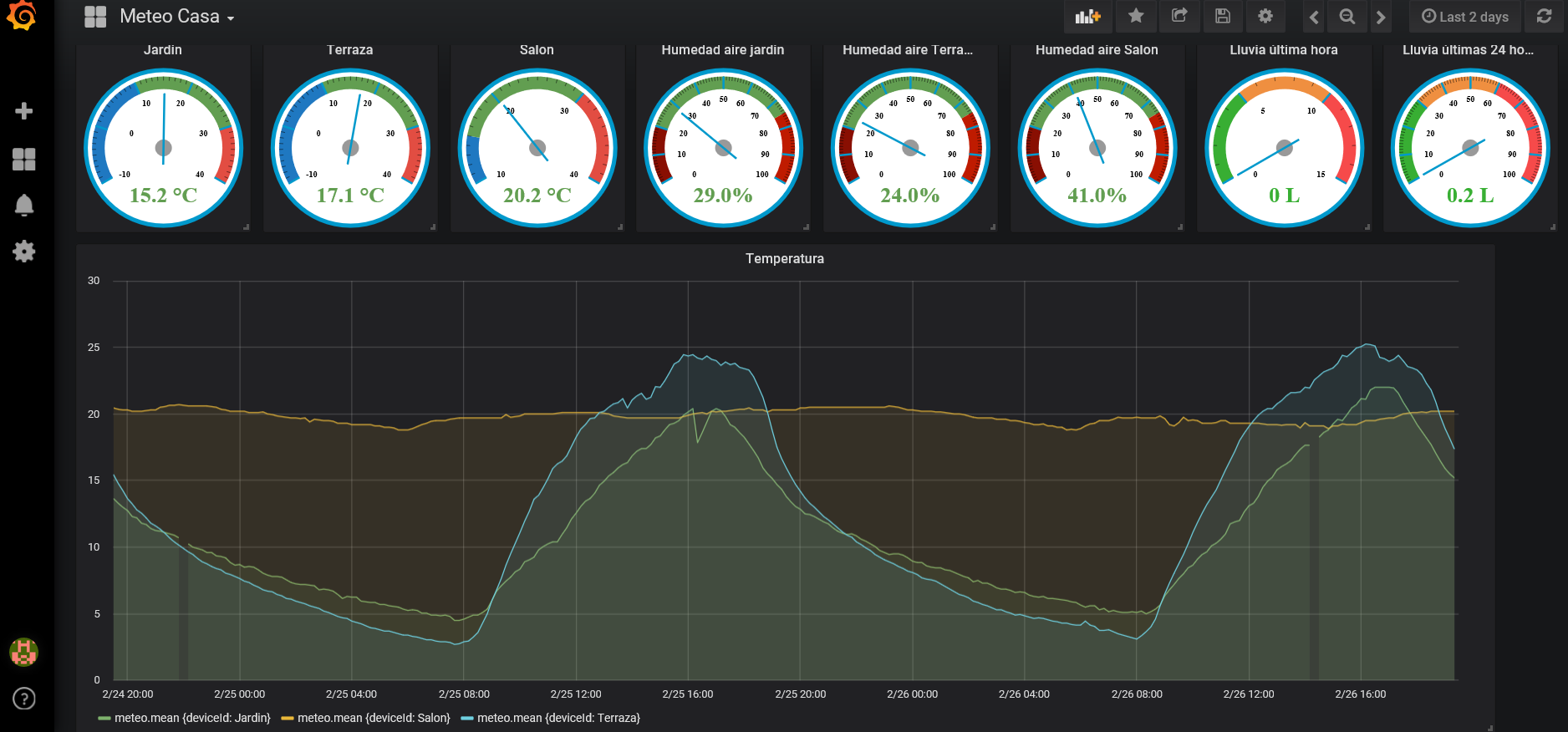
and navigating to <http://[My_IP_Address]:>9000 or <http://portainer.MYDOMAIN.duckdns.org>

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

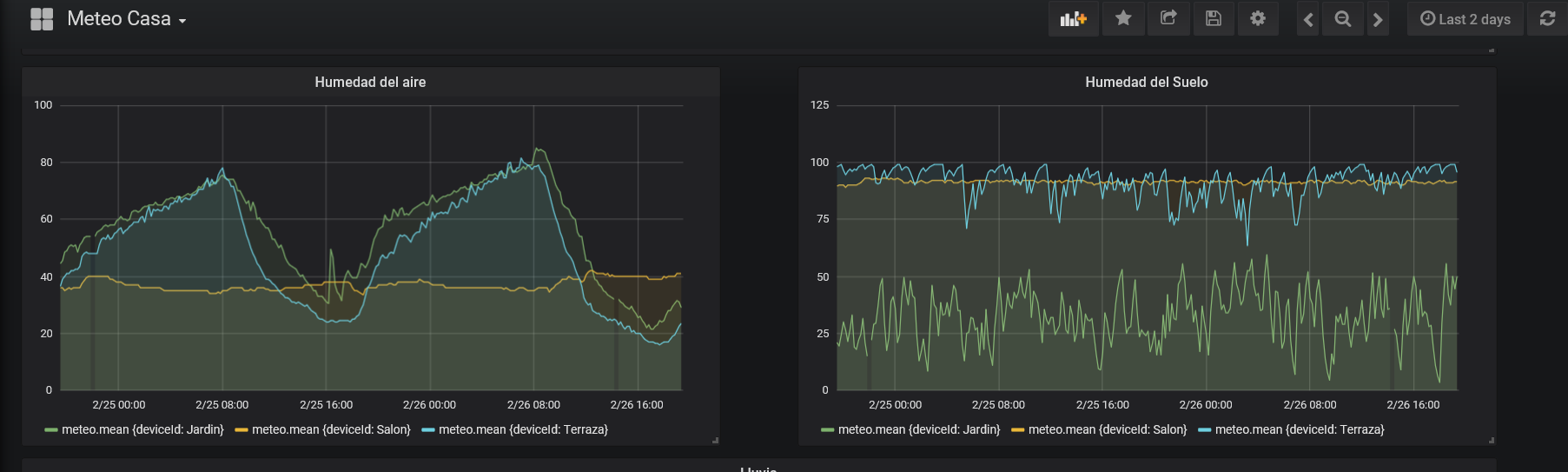
# Grafana customization

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

[can’t, please look for this (and for other) config files in MQTTSever directory

# Done!!!

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

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