SOMMAIRE

L'objectif de ce projet est de concevoir un capteur d'humidité et de température capable d'envoyer ses données vers une interface Grafana via le WiFi local.
Cette solution permet de centraliser la collecte et la visualisation des données environnementales sur un tableau de bord dynamique et accessible en temps réel.

Donc pour cela voici l'infrastructure mise en place :

1) Installer Docker & Compose sur Debian

Préparation du système

Nous commençons par mettre à jour Debian et installer les dépendances nécessaires à la configuration sécurisée des dépôts logiciels.
Ces prérequis garantissent la compatibilité et la stabilité du système avant l'installation de Docker.

```
apt update
apt -y install ca-certificates curl gnupg lsb-release
```

Ajout du dépôt officiel Docker

Nous ajoutons la clé GPG officielle et configurons le dépôt correspondant à notre version Debian (Bookworm, Bullseye, etc.).
Cette étape assure la fiabilité des paquets téléchargés et la mise à jour future du moteur Docker.

```
install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg \
 | gpg --dearmor -o /etc/apt/keyrings/docker.gpg
chmod a+r /etc/apt/keyrings/docker.gpg
```

Pour récupérer le code name Debian proprement (bookworm/bullseye…) nous faisons :

```
CODENAME=$( . /etc/os-release; echo $VERSION_CODENAME )
echo \
  "deb [arch=$(dpkg --print-architecture)
signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/debian $CODENAME stable" \
| tee /etc/apt/sources.list.d/docker.list > /dev/null
```

Installation et vérification

Nous installons Docker CE, son interface en ligne de commande et le plugin Docker Compose.

```
apt update
apt -y install docker-ce docker-ce-cli containerd.io
docker-buildx-plugin docker-compose-plugin
```

Une vérification des versions confirme que l'installation s'est déroulée correctement et que le service fonctionne.

```
docker --version
docker compose version
```

Cette capture confirme que Docker et Docker Compose sont installés et fonctionnels : Docker 28.5.0 (build 887030f) et Compose v2.39.4, indiquant une installation réussie.

```
root@debianVM:~# docker --version
docker compose version
Docker version 28.5.0, build 887030f
Docker Compose version v2.39.4
```

Mise en place du pare-feu (UFW)

Nous configurons un pare-feu simple afin de protéger les services exposés en autorisant uniquement les ports essentiels.
Les ports 22 (SSH), 3000 (Grafana) et 8086 (InfluxDB) sont ouverts pour garantir un accès contrôlé.

```
apt -y install ufw
ufw allow 22/tcp
ufw allow 3000/tcp   # Grafana
ufw allow 8086/tcp   # InfluxDB
ufw enable
ufw status
```

Cette étape renforce la sécurité du serveur en limitant les connexions aux seuls services nécessaires au fonctionnement de la stack IoT.

```
root@debianVM:~# ufw status
Status: active

To                         Action      From
--                         ------      ----
22/tcp                     ALLOW       Anywhere
3000/tcp                   ALLOW       Anywhere
8086/tcp                   ALLOW       Anywhere
22/tcp (v6)                ALLOW       Anywhere (v6)
3000/tcp (v6)              ALLOW       Anywhere (v6)
8086/tcp (v6)              ALLOW       Anywhere (v6)
```

La sortie ufw status montre un pare-feu actif autorisant 22, 3000 et 8086 en IPv4/IPv6, correspondant à SSH, Grafana et InfluxDB.

2) Déploiement d'InfluxDB et Grafana avec Docker Compose

Création de l'arborescence

Nous isolons les répertoires de stockage pour éviter la perte de données lors des redéploiements ou mises à jour.

```
mkdir -p /root/iot-stack/data/influxdb
mkdir -p /root/iot-stack/data/grafana
```

Fichier Docker Compose

Le fichier `docker-compose.yml` définit deux services complémentaires :

- ➔ InfluxDB, base de données orientée séries temporelles, pour stocker les mesures du capteur.
- ➔ Grafana, outil de visualisation, pour afficher les données collectées sous forme de graphiques interactifs.

```
services:
  influxdb:
    image: influxdb:2
    container_name: influxdb
    ports:
      - "8086:8086"
    volumes:
      - ./data/influxdb:/var/lib/influxdb2
    environment:
      - DOCKER_INFLUXDB_INIT_MODE=setup
      - DOCKER_INFLUXDB_INIT_USERNAME=admin
      - DOCKER_INFLUXDB_INIT_PASSWORD=admin12345
```

```
      - DOCKER_INFLUXDB_INIT_ORG=roran
      - DOCKER_INFLUXDB_INIT_BUCKET=sensors
      - DOCKER_INFLUXDB_INIT_ADMIN_TOKEN=super-secret-token
    restart: unless-stopped

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    ports:
      - "3000:3000"
    volumes:
      - ./data/grafana:/var/lib/grafana
    depends_on:
      - influxdb
    restart: unless-stopped
```

Lancement et vérification

Nous lançons les conteneurs avec :

```
cd /root/iot-stack
docker compose up -d
```

La commande docker compose up -d montre que la stack s'est lancée en arrière-plan : deux conteneurs créés, InfluxDB et Grafana, tous deux au statut Running.

```
root@debianVM:~/iot-stack# docker compose up -d
[+] Running 2/2
 ✔Container influxdb  Running                                                    0.0s
 ✔Container grafana   Running                                                    0.0s
```

Puis nous vérifions leur état :

```
docker compose ps
```

La commande docker compose ps confirme que deux conteneurs tournent: grafana et influxdb, en état Up, exposant respectivement 3000/tcp et 8086/tcp sur toutes les interfaces.

```
root@debianVM:~/iot-stack# docker compose ps
NAME       IMAGE                   COMMAND                  SERVICE    CREATED          STATUS          PORTS
grafana    grafana/grafana:latest  "/run.sh"                grafana    7 minutes ago    Up 2 minutes    0.0.0.0:3
000->3000/tcp, [::]:3000->3000/tcp
influxdb   influxdb:2              "/entrypoint.sh infl…"   influxdb   7 minutes ago    Up 7 minutes    0.0.0.0:8
086->8086/tcp, [::]:8086->8086/tcp
```

Nous pouvons accéder au interface via les adresses suivante :

- InfluxDB : http://192.168.1.21:8086
- Grafana : http://192.168.1.21:3000

Correction du problème de permission Grafana

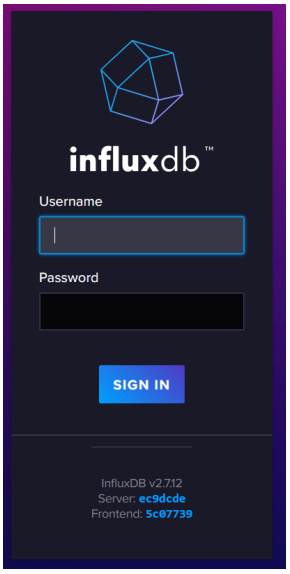
Lors du premier démarrage, Grafana rencontrait une erreur de droits sur son répertoire de données. Nous avons attribué la propriété au bon utilisateur (ID 472) afin de garantir l'écriture dans le volume.

```
cd /root/iot-stack
chown -R 472:472 data/grafana
docker compose restart grafana
```

3) Configuration d'InfluxDB et test d'écriture

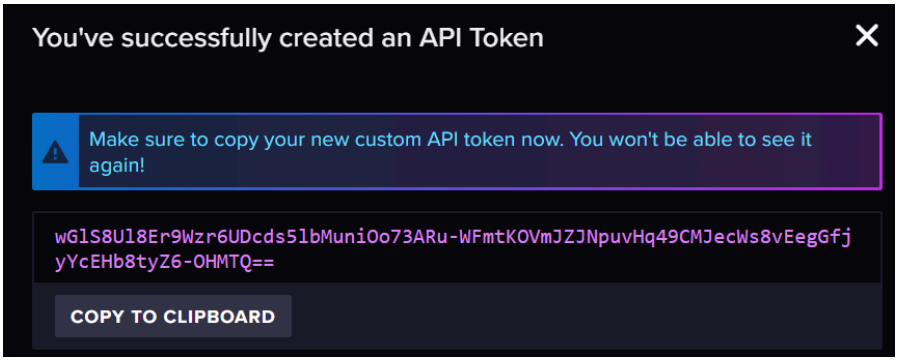
Connexion et génération du token

Nous nous connectons à InfluxDB via *http://192.168.1.21:8086* avec les identifiants administrateur qui sont *admin/admin12345*.



Nous créons ensuite un token d'écriture pour le bucket sensors, permettant à nos capteurs d'envoyer des données sans exposer le compte root.

Pour cela nous nous rendons dans Load Data → API Tokens → Generate Token → Write Token pour le bucket sensors (appelle-le pycom-write et note la valeur).



Test d'écriture

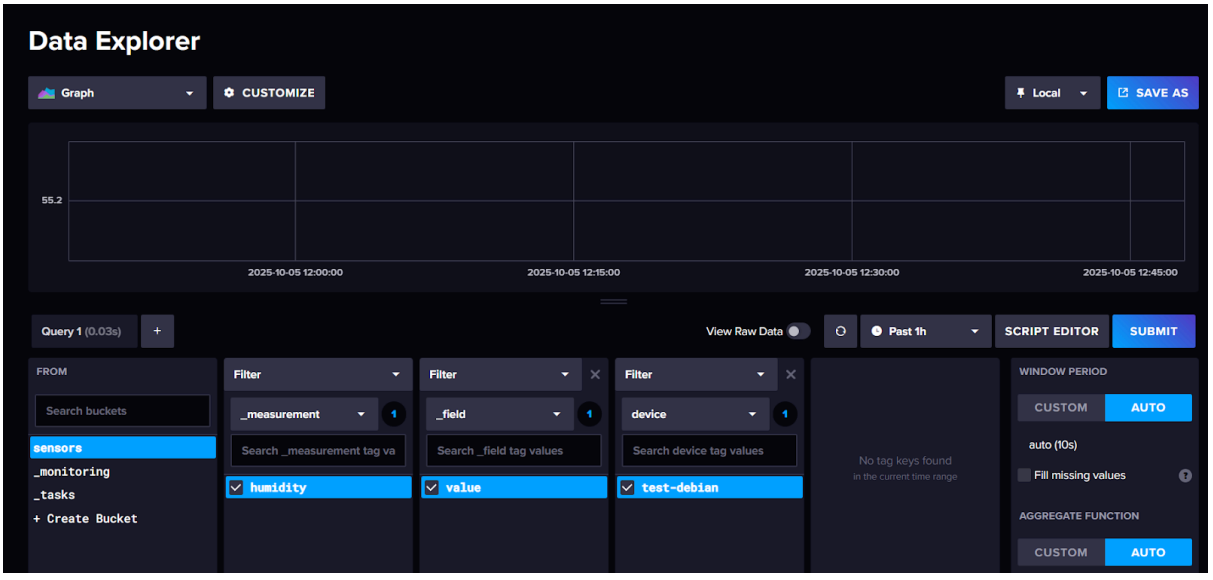
Un test de communication est effectué via la commande curl pour s'assurer que la base de données reçoit correctement les données.

```
curl -i -XPOST
"http://192.168.1.21:8086/api/v2/write?org=roran&bucket=sensors&precision=s" \
  -H "Authorization: Token
wGlS8Ul8Er9Wzr6UDcds5lbMuniOo73ARu-WFmtKOVmJZJNpuvHq49CMJecWs8vEegGfjyYcEHb8tyZ6
-OHMTQ==" \
  -H "Content-Type: text/plain" \
  --data-raw 'humidity,device=test-debian value=55.2'
```

Nous vérifions ensuite dans Data Explorer qu'une ligne de données apparaît dans le bucket sensors.

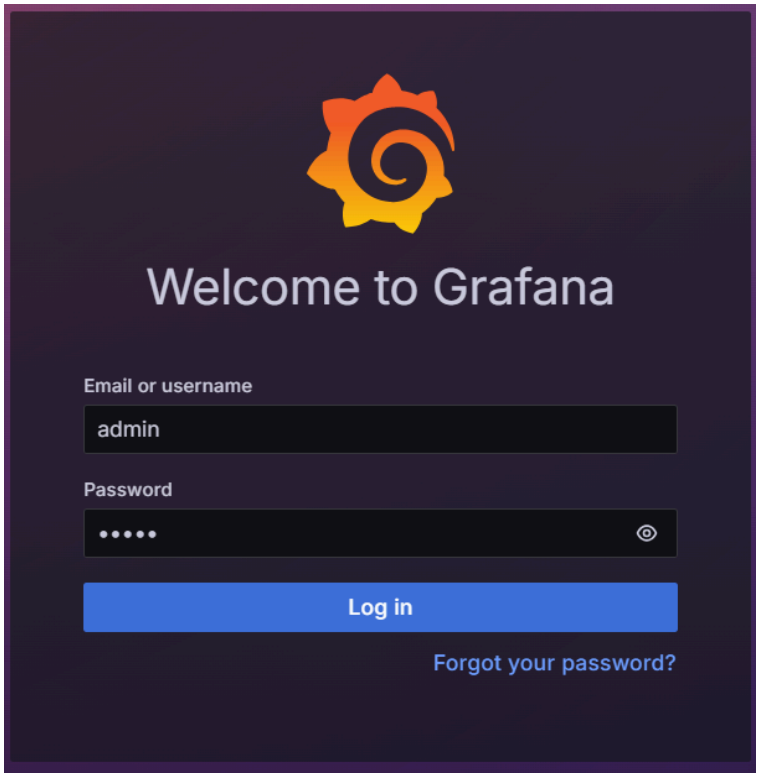
```
root@debianVM:~/iot-stack# curl -i -XPOST "http://192.168.1.21:8086/api/v2/write?org=roran&bucket=sensors&precis
ion=s" \
  -H "Authorization: Token wGlS8Ul8Er9Wzr6UDcds5lbMuniOo73ARu-WFmtKOVmJZJNpuvHq49CMJecWs8vEegGfjyYcEHb8tyZ6-OHMT
Q==" \
  -H "Content-Type: text/plain" \
  --data-raw 'humidity,device=test-debian value=55.2'
HTTP/1.1 204 No Content
X-Influxdb-Build: OSS
X-Influxdb-Version: v2.7.12
Date: Sun, 05 Oct 2025 08:40:54 GMT
```

Ce test valide la communication entre notre script et InfluxDB.



4) Configuration de Grafana et création des tableaux de bord

Nous nous connectons à Grafana via http://192.168.1.21:3000 (admin / admin).



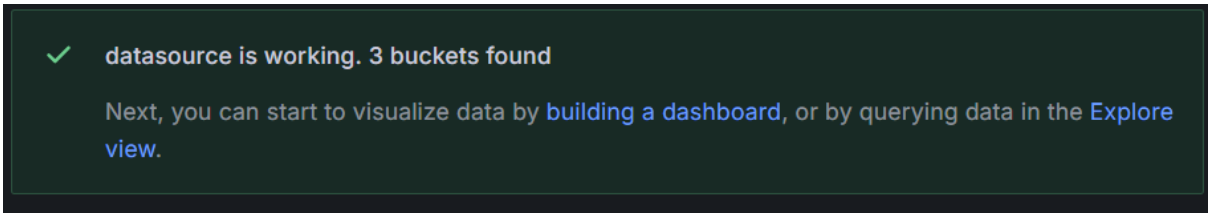
Dans **Connections** → **Data Sources**, nous ajoutons une source InfluxDB avec les paramètres suivants :

- URL : http://192.168.1.21:8086
- Organization : roran
- Token : (token généré précédemment)
- Bucket par défaut : sensors

Langage de requête : **FluxConnections** → **Data sources** → **Add data source** → **InfluxDB**

- Query language : Flux
- URL : http://192.168.1.21:8086
- Organization : roran
- Token:wGlS8Ul8Er9Wzr6UDcds5lbMuniOo73ARu-WFmtKOVmJZJNpuvHq49CMJecWs8vEegGfjyYcEHb8tyZ6-OHMTQ==
- Default Bucket : sensors

Une fois le test de connexion validé (indicateur vert), nous créons un dashboard comportant deux graphiques :



Cette visualisation facilite le suivi de l'environnement sans passer par des requêtes manuelles.

Création des tableaux de bord dans Grafana

Une fois la source de données InfluxDB configurée et testée avec succès, nous pouvons créer nos tableaux de bord pour visualiser les mesures reçues du capteur.

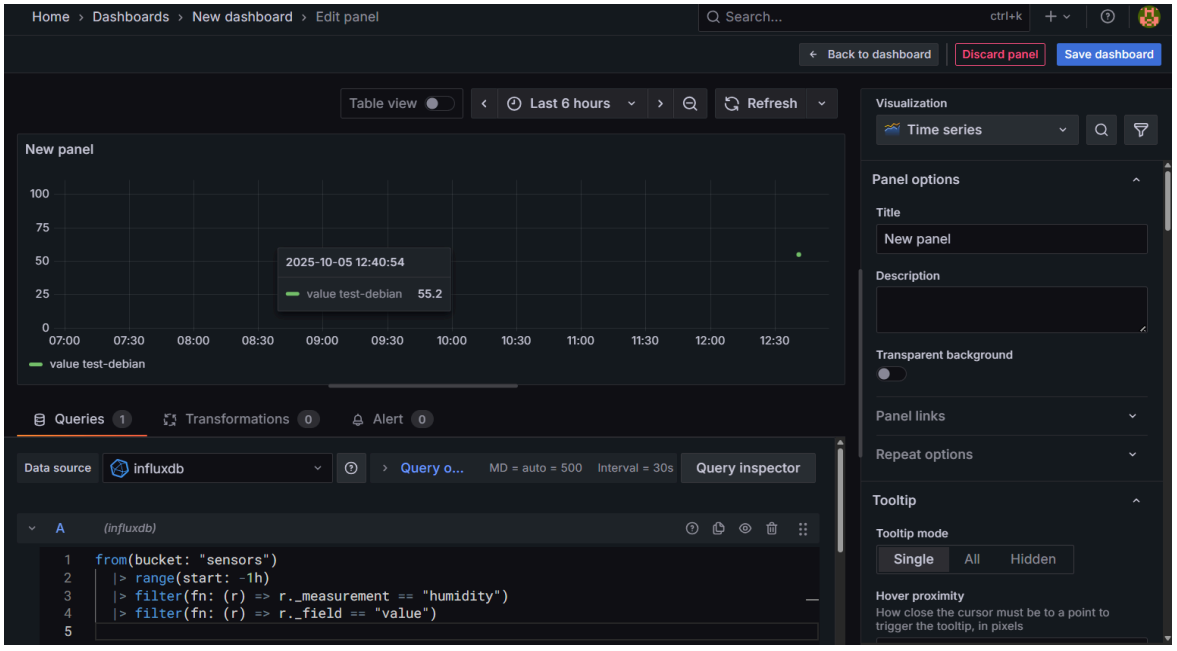
1. Accès à la création de dashboard

Dans l'interface Grafana, ouvre le menu latéral gauche, puis clique sur :
Create → Dashboard → Add visualization.
Cette section permet de créer des graphiques personnalisés à partir des données collectées par InfluxDB.

2. Sélection de la source de données

Dans la fenêtre de configuration, nous sélectionnons la source InfluxDB que nous avons ajoutée précédemment.

Cela indique à Grafana où aller chercher les données pour générer les courbes.



Création du premier tableau de bord – Humidité

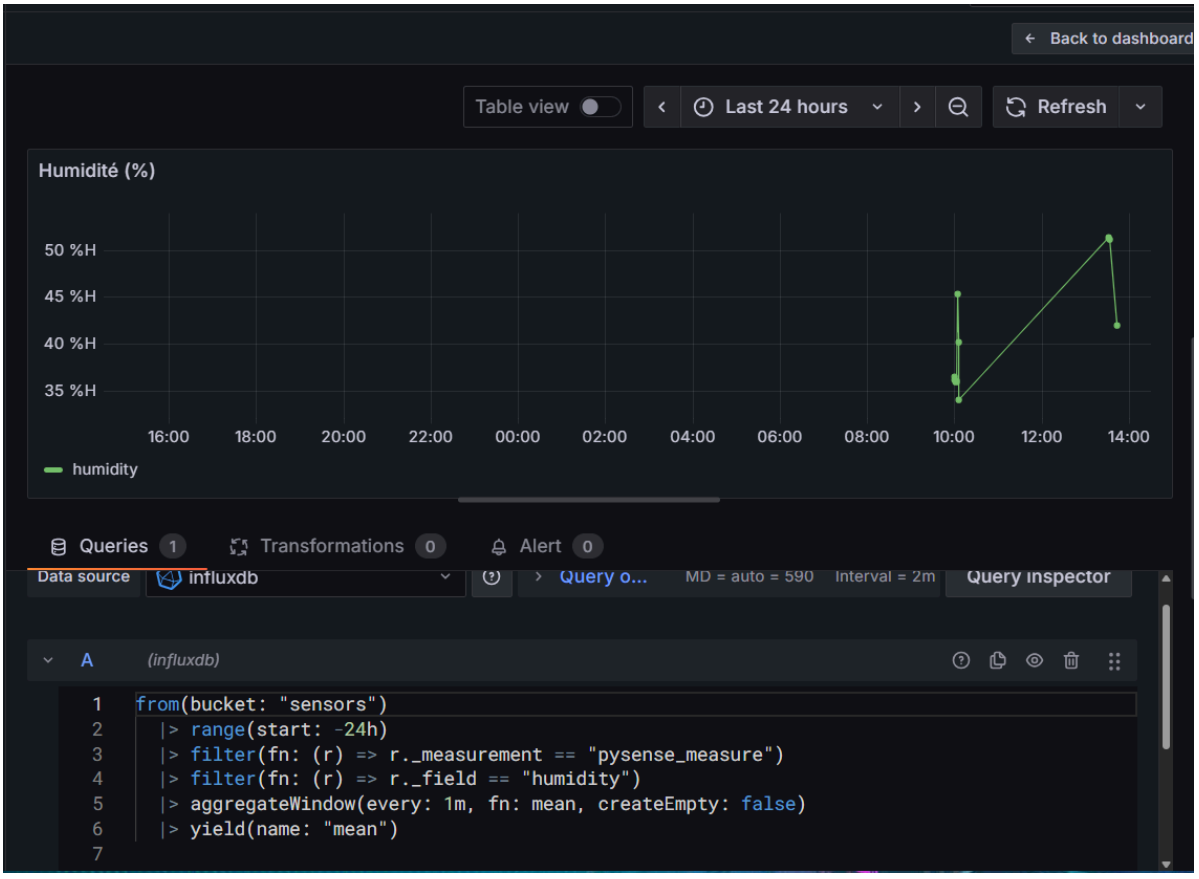
Cette visualisation affiche l'humidité provenant du capteur pysense_measure, agrégée par minute sur les dernières vingt-quatre heures pour lisser le bruit.

Nous utilisons aggregateWindow(mean, 1m) afin de réduire la variabilité instantanée et d'améliorer la lisibilité des courbes temporelles.

Voici notre Requête Flux :

```
from(bucket: "sensors")
  |> range(start: -24h)
  |> filter(fn: (r) => r._measurement == "pysense_measure")
  |> filter(fn: (r) => r._field == "humidity")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Cette requête sélectionne les valeurs d'humidité reçues durant la dernière heure pour les afficher en temps réel.



Elle récupère les points d'humidité du capteur ciblé, puis calcule une moyenne glissante par minute pour fournir une tendance fiable et exploitable.

Création du second tableau de bord – Température

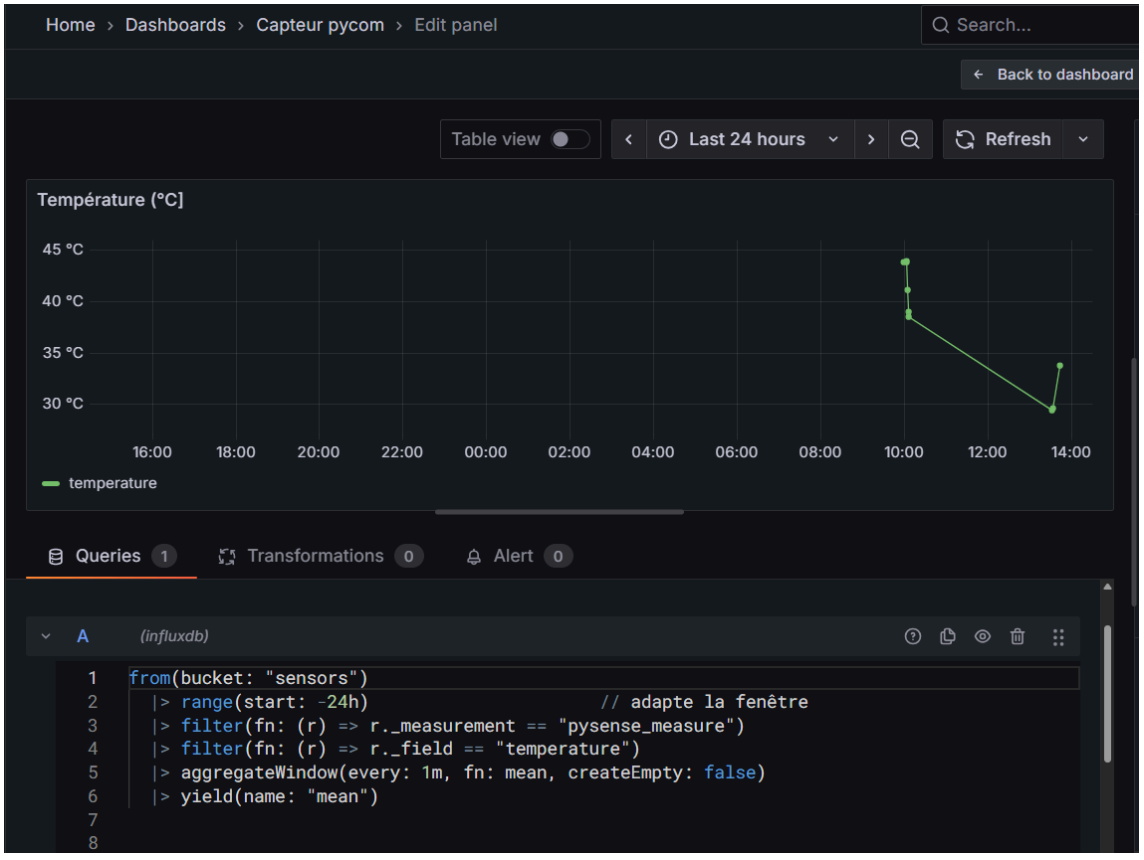
Cette visualisation affiche la température du même capteur sur vingt-quatre heures, avec la même fenêtre d'agrégation d'une minute pour rester parfaitement comparable.

Conserver une fenêtre identique entre métriques facilite la corrélation visuelle et évite des décalages d'échantillonnage pénibles à interpréter.

Requête Flux :

```
from(bucket: "sensors")
  |> range(start: -24h)                          // adapte la fenêtre
  |> filter(fn: (r) => r._measurement == "pysense_measure")
  |> filter(fn: (r) => r._field == "temperature")
  |> aggregateWindow(every: 1m, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Cette requête sélectionne les valeurs d'humidité reçues durant la dernière heure pour les afficher en temps réel.



Elle isole les mesures de température du bon appareil et applique la même agrégation, garantissant une comparaison cohérente avec la série d'humidité.

Voici les deux dashboard créer :
Nous visualisons l'humidité et la température sur vingt-quatre heures, agrégées par minute, afin de lisser le bruit et comparer les tendances entre capteurs équivalents.

Dans le pycom nous avons implémenté les librairies et le code présent en annexe afin de faire fonctionner le tout.

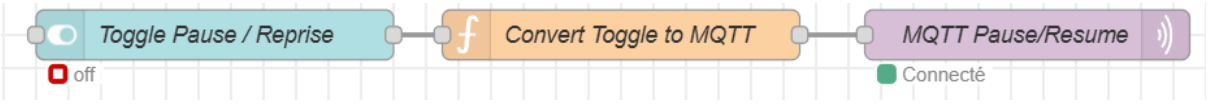
Pour faire le lien entre mosquitto et influxdb. Nous avons choisi d'utiliser un script python. Pour cela il est nécessaire d'utiliser le venv de python. Nous utilisons donc les deux commande suivante :

```
source ~/pysense_env/bin/activate
python ~/mon_script.py
```

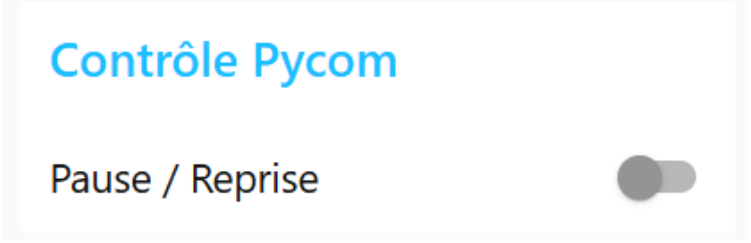
Le contenu de mon_script.py est disponible en annexe.

Contrôle à distance

Pour le contrôle à distance nous avons choisis d'utiliser Node-Red qui permet via des graphiques de créer des interfaces permettant d'agir sur le pycom et ces capteurs.



Ce genre de graphe peut être généré grâce au JSON présent en annexe. De plus cela permet donc de mettre en place une interface à l'adresse http://10.182.123.14:1880/ui. Cette interface permet donc de mettre en pause le programme ou de le reprendre.



Le chemin est donc :


```
root@debianVM:~# npm install -g --unsafe-perm node-red
(##########⧗⧗⧗⧗⧗⧗⧗⧗⧗) ⧗ idealTree:lib: timing idealTree:#root Completed in 6458ms
```

Cette commande installe globalement Node-RED avec les permissions administrateur nécessaires pour héberger l'interface de contrôle MQTT qui nous permet de piloter notre capteur Pycom à distance.

Annexe :

JSON pour Node Red

```
[
    {
        "id": "tab-pycom-pause",
        "type": "tab",
        "label": "Pycom Pause/Resume",
        "disabled": false,
        "info": ""
    },
    {
        "id": "ui-switch-pycom",
        "type": "ui_switch",
        "z": "tab-pycom-pause",
        "name": "Toggle Pause / Reprise",
        "label": "Pause / Reprise",
        "tooltip": "",
        "group": "group-pycom",
        "order": 1,
        "width": 0,
        "height": 0,
        "passthru": true,
        "decouple": "false",
        "topic": "commande/pysense1",
        "style": "",
        "onvalue": true,
        "onvalueType": "bool",
        "onicon": "",
        "oncolor": "orange",
        "offvalue": false,
        "offvalueType": "bool",
        "officon": "",
        "offcolor": "green",
        "x": 150,
        "y": 100,
        "wires": [["function-toggle-to-mqtt"]]
    },
    {
        "id": "function-toggle-to-mqtt",
        "type": "function",
        "z": "tab-pycom-pause",
        "name": "Convert Toggle to MQTT",
        "func": "// true = pause, false =
resume\nmsg.payload = (msg.payload === true) ?
\"pause\" : \"resume\";\nreturn msg;",
        "outputs": 1,
        "noerr": 0,
        "initialize": "",
        "finalize": "",
        "libs": [],
        "x": 400,
        "y": 100,
        "wires": [["mqtt-out-pycom"]]
    },
    {
        "id": "mqtt-out-pycom",
        "type": "mqtt out",
        "z": "tab-pycom-pause",
        "name": "MQTT Pause/Resume",
        "topic": "commande/pysense1",
        "qos": "",
        "retain": "",
        "respTopic": "",
        "contentType": "",
        "userProps": "",
        "correlationData": "",
        "messageType": "utf8",
        "x": 650,
        "y": 100,
        "wires": [],
        "broker": "mqtt-broker-pycom"
    },
    {
        "id": "group-pycom",
        "type": "ui_group",
        "name": "Contrôle Pycom",
        "tab": "tab-dashboard",
        "order": 1,
        "width": "6",
        "collapse": false
    },
    {
        "id": "tab-dashboard",
        "type": "ui_tab",
        "name": "Dashboard",
        "icon": "dashboard",
        "order": 1,
        "disabled": false,
        "hidden": false
    },
    {
        "id": "mqtt-broker-pycom",
        "type": "mqtt-broker",
        "name": "Mosquitto",
        "broker": "10.182.123.14",
        "port": "1883",
        "clientid": "nodered-pycom",
        "usetls": false,
        "protocolVersion": "4",
        "keepalive": "60",
        "cleansession": true,
        "birthTopic": "",
        "birthQos": "0",
        "birthPayload": "",
        "closeTopic": "",
        "closeQos": "0",
        "closePayload": "",
        "willTopic": "",
        "willQos": "0",
        "willPayload": ""
    }
]
```

Contene de mon_script.py

```
import json
import time
from paho.mqtt import client as mqtt
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS

# --- CONFIGURATION MQTT ---
MQTT_BROKER = "10.182.123.14"   # IP de ton broker Mosquitto
MQTT_PORT = 1883
MQTT_TOPIC = "capteurs/+"        # s'abonner à tous les sous-topics

# --- CONFIGURATION INFLUXDB ---
INFLUX_URL = "http://localhost:8086"
INFLUX_TOKEN =
"wGlS8Ul8Er9Wzr6UDcds5lbMuniOo73ARu-WFmtKOVmJZJNpuvHq49CMJecWs8vEegGfjyYcEHb8tyZ6-OHMTQ=
="      # remplace par ton token réel
INFLUX_ORG = "roran"            # ton organisation
INFLUX_BUCKET = "sensors"        # ton bucket

# --- Connexion InfluxDB ---
client = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN, org=INFLUX_ORG)
write_api = client.write_api(write_options=SYNCHRONOUS)

# --- Fonction callback MQTT ---
def on_message(client, userdata, msg):
    payload = msg.payload.decode()
    print(f"Message MQTT reçu: {msg.topic} -> {payload}")  # affichage debug
    try:
        value = float(payload)  # convertir le float
        point = Point("pysense_measure")

        # Déterminer le champ à écrire selon le topic
        if "temperature" in msg.topic:
            point = point.field("temperature", value)
        elif "humidity" in msg.topic:
            point = point.field("humidity", value)
        else:
            print("Topic inconnu, ignoré")
            return
        write_api.write(bucket=INFLUX_BUCKET, org=INFLUX_ORG, record=point)
        print(f"Écrit dans InfluxDB: {msg.topic} -> {value}")
    except Exception as e:
        print(f"Erreur traitement message: {e}")

# --- Connexion MQTT ---
mqtt_client = mqtt.Client(client_id="pysense_bridge")
mqtt_client.on_message = on_message
mqtt_client.connect(MQTT_BROKER, MQTT_PORT)
mqtt_client.subscribe(MQTT_TOPIC)

print("Script démarré, en attente de messages MQTT...")
mqtt_client.loop_forever()
```

Programme Pysense :

```
from network import WLAN
from mqtt import MQTTClient
import pycom
import time
from SI7006A20 import SI7006A20
from pycoproc_1 import Pycoproc

# --- Paramètres ---
BROKER = "10.182.123.14"
TOPIC_DATA_TEMP = "capteurs/temperature"
TOPIC_DATA_HUM = "capteurs/humidity"
TOPIC_CMD = "commande/pysense1"

# --- État du système ---
paused = False

# --- Callback MQTT ---
def sub_cb(topic, msg):
    global paused
    msg = msg.decode('utf-8')
    print("Message reçu sur", topic, ":", msg)

    if msg == "pause":
        paused = True
        print("=== PAUSE ACTIVÉE ===")
    elif msg == "resume":
        paused = False
        pycom.rgbled(0x002000)  # vert fixe
        print("=== REPRISE ===")

# --- Connexion WiFi ---
pycom.heartbeat(False)
wlan = WLAN(mode=WLAN.STA)
wlan.connect(ssid="Spokecraft's Galaxy A34 5G", auth=(WLAN.WPA2, "Marquez73"))
while not wlan.isconnected():
    pycom.rgbled(0x101010)
    time.sleep(1)
pycom.rgbled(0x002000)
print("Connected:", wlan.ifconfig())

# --- MQTT ---
client = MQTTClient("pysense1", BROKER, port=1883)
client.set_callback(sub_cb)
client.connect()
client.subscribe(TOPIC_CMD)

# --- Capteur Pysense ---
py = Pycoproc(Pycoproc.PYSENSE)
si = SI7006A20(py)

# --- Boucle principale ---
while True:
    client.check_msg()  # Vérifie si un message arrive

    if paused:
        # Clignotement orange faible
        pycom.rgbled(0x201000)
        time.sleep(0.5)
        pycom.rgbled(0x000000)
        time.sleep(0.5)
        continue  # saute l'envoi de données

    humidity = si.humidity()
    temperature = si.temperature()

    print("Humidité : {:.2f}%".format(humidity))
    print("Température : {:.2f}°C".format(temperature))

    client.publish(TOPIC_DATA_HUM, str(humidity))
    client.publish(TOPIC_DATA_TEMP, str(temperature))
    time.sleep(10)
```