Millot Elisa
Dumas Mathieu

Rapport Technique SAE3.01

Sommaire :

1. Introduction

1.1 Contexte du Projet

- Dans le cadre de ce projet, nous avons pour mission de concevoir un objet connecté mesurant la qualité de l'air, en particulier la concentration de CO2. Ce projet s'inscrit dans une démarche de découverte et d'application pratique des technologies IoT (Internet of Things), en exploitant le réseau communautaire **The Things Network (TTN)** qui nous a préalablement été présenté en TD.
- En intégrant un capteur de CO2 précis avec une carte de développement compatible LoRaWAN, nous aurons l'opportunité de renforcer nos compétences en électronique, programmation embarquée et transmission de données sans fil. Ce projet nous pousse également à maîtriser l'intégration des données avec des outils modernes .
- Nous visons à créer un système fonctionnel et sécurisé, tout en proposant des affichages visibles et intuitifs grâce à des widgets graphiques, permettant une expérience utilisateur enrichissante..

1.2 Objectifs

Le projet vise à atteindre les objectifs suivants :

- Développer un système connecté fonctionnel pour mesurer la qualité de l'air et transmettre les données via LoRaWAN.
- Paramétrer le mode de communication **OTAA** sur TTN afin d'assurer une transmission sécurisée entre le capteur et le serveur cloud.
- Acquérir une maîtrise des outils IoT modernes en intégrant des protocoles sécurisés et en paramétrant efficacement les identifiants et clés.

1.3 Environnement de Développement

Pour mener à bien ce projet, nous utiliserons les outils et technologies suivants :

- **Matériel** :
  - **Carte Seeeduino LoRaWAN**, compatible avec l'environnement Arduino
  - **Carte arduino uno**, permettant de gérer notre maquette.
  - **Capteur de CO2 SCD30** de Sensirion, reconnu pour sa précision et sa polyvalence.
- **Environnement logiciel** :
  - **IDE Arduino**, pour programmer la carte Seeeduino et configurer la communication avec le capteur.

- **The Things Network (TTN)**, pour gérer la passerelle et transmettre les données en mode sécurisé.
- **Grafana** pour intégrer et visualiser les données via un dashboard.

2. Récupération et envoi sur TTN

2.1 Câblage de la maquette

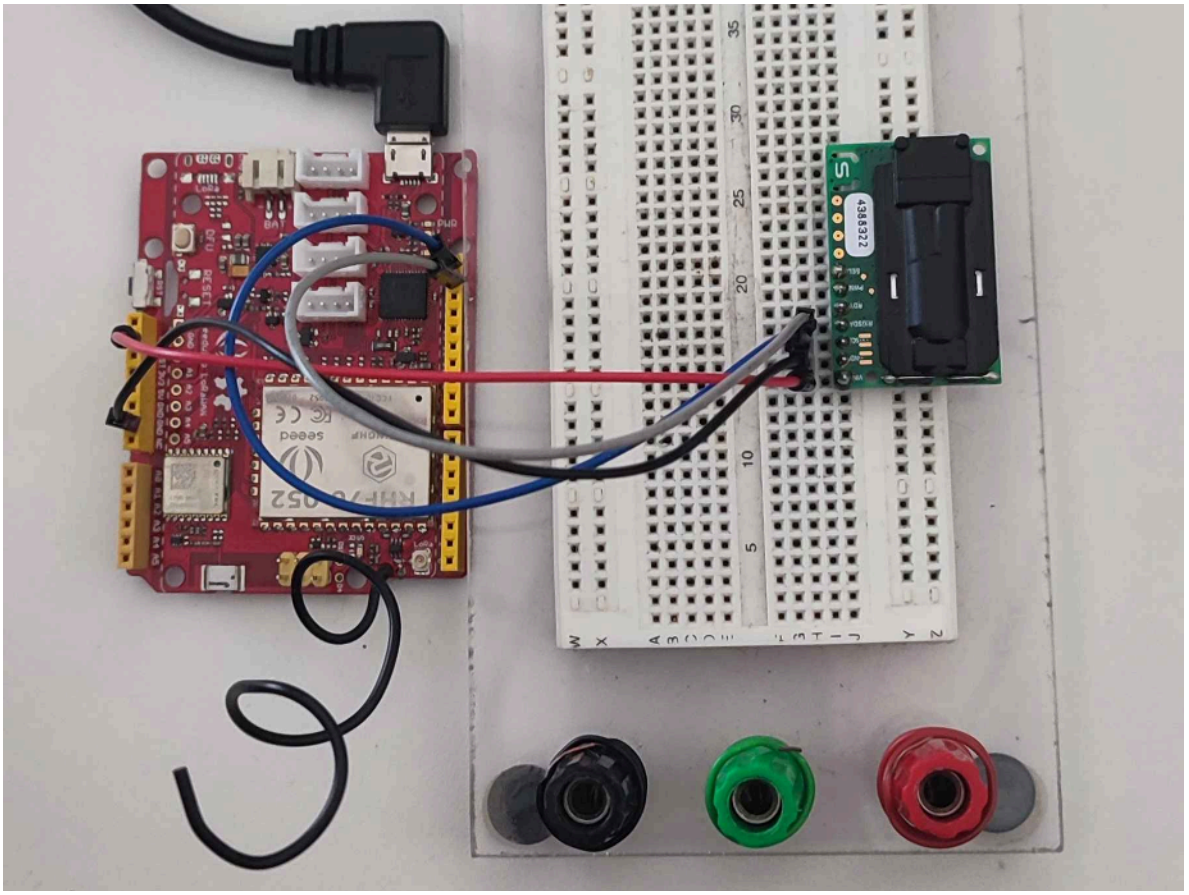
Nous nous sommes dans un premier temps intéressés à la manière dont nous allons cabler notre capteur à notre carte seeeduino.

En regardant dans la documentation, nous réussissons à trouver les PIN que nous allons connecter à notre cable.

Cela nous donnera donc ceci :

Sur le capteur SCD30	Sur la carte Seeeduino LoRaWAN
VDD	3.3V
GND	GND
TX/SCD	SCD
TX/SDA	SDA

Une fois notre maquette correctement connecté, cela nous donne ceci :



2.2 Création du code de récupération de données du capteur et d'envoi sur TTN.

Maintenant que nous avons réussi à brancher convenablement notre maquette, nous créons ensuite notre programme nous permettant de faire fonctionner tout ceci.

Dans un premier temps nous allons inclure nos librairie :

```
#include <Wire.h>
#include "Adafruit_SCD30.h"
#include <LoRaWan.h>
```

La librairie SCD30 nous permet d'utiliser le capteur et la librairie LoRaWan nous permet d'envoyer les données sur TTN.
Ensuite nous devons initialiser notre capteur. Pour cela nous faisons ceci :

```
    Serial.begin(9600);

    // Initialiser le capteur SCD30
    if (!scd30.begin()) {
```

```
        while (1) { delay(10); } // Boucle infinie si le capteur n'est
pas trouvé
    }
```

Cela nous permet d'afficher dans le moniteur série l'initialisation de notre capteur.
Ensuite, nous devons créer notre boucle de récupération des données. Nous avons donc ceci :

```
void loop(void) {
    // Vérifier si de nouvelles données sont disponibles sur le capteur
    if (scd30.dataReady()) {
        if (!scd30.read()) {
            return; // Arrêter si une erreur de lecture se produit
        }

        // Préparer les données pour l'envoi
        float co2 = scd30.CO2;
        float temperature = scd30.temperature;
        float humidity = scd30.relative_humidity;

        char data[64];
        snprintf(data, sizeof(data), "CO2=%.2f,Temp=%.2f,Hum=%.2f",
co2, temperature, humidity);
```

Maintenant, notre programme fonctionne correctement nous avons juste à réussir à les envoyer sur TTN.
Pour cela, après avoir créé notre compte, nous créons une application. Nous pensons bien à récuperer les clefs que nous renseignerons dans notre code comme ceci :

```
// Identifiants AppEUI, DevEUI et AppKey
char DevEUI[] = "70B3D57ED006CC5E";
char AppEUI[] = "8CF9572000000000";
char AppKey[] = "0EB94B96E2E0B7300B366C43FD49FDE7";
```

Cela permettra à la gateway qui va recevoir les messages de rediriger les données vers notre application étant donnée que c'est des identifiants unique.
Ensuite, dans notre programme, nous devons envoyer les données sur TTN. Pour cela, nous faisons ceci :

```
    // Initialiser LoRaWAN
    lora.init();
    lora.setId(NULL, DevEUI, AppEUI); // DevAddr est NULL pour OTAA
    lora.setKey(NULL, NULL, AppKey); // NwkSKey et AppSKey sont NULL
pour OTAA

    lora.setDeciveMode(LWOTAA);
```

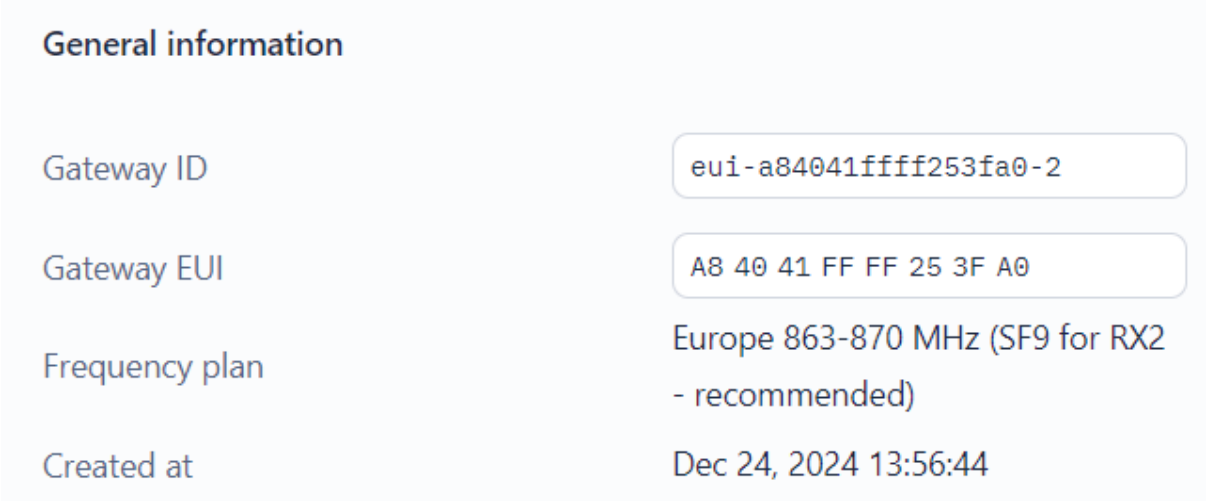
```
    lora.setDataRate(DR0, EU868);

    lora.setChannel(0, 868.1);
    lora.setChannel(1, 868.3);
    lora.setChannel(2, 868.5);

    lora.setReceiceWindowFirst(0, 868.1);
    lora.setReceiceWindowSecond(869.5, DR3);

    lora.setDutyCycle(false);
    lora.setJoinDutyCycle(false);
    lora.setPower(14);

    // Rejoindre le réseau LoRaWAN
    while (!lora.setOTAAJoin(JOIN)) {
        delay(10000); // Attendre 10 secondes avant de réessayer
```

Cette partie du programme permet d'initialiser lorawan .
Ensuite nous envoyons les données :

```
        // Envoyer les données via LoRaWAN
        lora.transferPacket((unsigned char*)data, strlen(data));
```

2.3 Configuration de la gateway

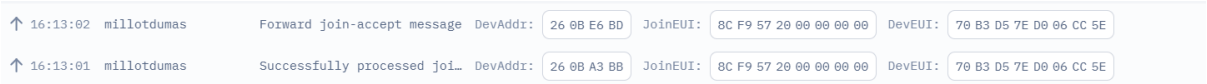
Pendant les vacances, nous avons pu demander un prêt de la gateway Lorawan. Cela nous à demandé une légère configuration supplémentaire étant donnée que celle-ci n'était pas inscrite dans l'organisation.
Nous l'avons donc ajouté avec les informations suivante :



Une fois ceci fait, celle-ci se montait et fonctionnait convenablement . Cela nous a donc permis de récolter un maximum de données pendant les vacances afin d'avoir un grand nombre d'informations dans notre base de données.

2.4 Preuve de fonctionnement

Une fois que nous avons réalisé ceci, il nous reste simplement à vérifier que les données soient bien envoyées sur notre application TTN.
Pour cela, nous nous connectons à notre compte et nous regardons le "Live Data" de notre application.
Nous voyons donc que nous avons bien l'établissement de la connexion sur TTN :



Suite à cela, nous recevons donc toutes les 15/20 secondes environ les données de notre capteur.

Celles-ci sont hachée en base 64 :

Et lorsque nous le décodons, nous avons bien les bonnes informations :

2.5 Conclusion :

Suite à cette partie, nous avons donc bien configuré notre objet connecté afin de pouvoir envoyer les données sur le things network. Il reste cependant les ajouts post-envoi à configurer afin de pouvoir réaliser un objet utilisable et installable dans une habitation par exemple.

3. Récupération des données sur TTN

3.1 Développement du programme de récupération

Pour pouvoir récupérer les données de TTN, nous avons dû configurer plusieurs choses. Tout d'abord, nous récupérons ces données via un serveur mqtt au fur et à mesure pour ensuite décoder les données et les afficher sur notre invite de commande.

Pour cela, nous implémentons dans un premier temps nos librairies

```
import paho.mqtt.client as mqtt
import base64
```

Ensuite, nous configurons les informations TTN de notre serveur MQTT afin de pouvoir nous connecter à notre serveur et récupérer les données.

```
# Configuration
app_id = "dumasmillot"
tenant_id = "ttn"
access_key =
"NNSXS.PSFSPYG56GVZZOC3XHKHT5RLQAUKR25LFOVTBDA.H7JER7JYTBA6K5DFEZL5VQEX
33EAXMZGU2MVLFHM2V2IASXZXERQ"
```

Nous avons donc notre clef d'accès à notre serveur MQTT, le nom de notre application et le site que nous rejoignons.
Maintenant, nous devons mettre en place les retours sur l'invite de commande me permettant de vérifier que tout fonctionne.
Pour cela nous configurons un callback lorsque la connexion avec notre serveur est établie :

```
# Callback lorsque la connexion est établie
def on_connect(client, userdata, flags, rc):
    print("Connecté avec le code de résultat " + str(rc))
    client.subscribe(f"v3/{app_id}@{tenant_id}/devices/+/up")
```

Ensuite, nous configurons une nouvelle fonction qui envoie un message dans le moniteur série à chaque fois qu'un message est reçu :

```
# Callback lorsque le message est reçu
def on_message(client, userdata, msg):
    payload = msg.payload.decode()
    payload_dict = json.loads(payload)
    frm_payload = payload_dict['uplink_message']['frm_payload']
    decoded_payload = base64.b64decode(frm_payload).decode('utf-8')
    print(f"Message déchiffré: {decoded_payload}")
```

Cela permet d'obtenir les informations de TTN, les convertir en un format json, le load pour ensuite récupérer une donnée précise. Cette donnée que nous allons récupérer est à la suite de [frm_payload] c'est-à -dire la charge utile que nous transmettons.
Nous la décodons ensuite étant donné que cette information est encodée en base64.
Ensuite, nous affichons sur notre invite de commande.

Enfin, nous implémentons le tout dans notre client que nous connectons à notre serveur comme ceci :

```
client = mqtt.Client(protocol=mqtt.MQTTv311)
client.username_pw_set(f"{app_id}@{tenant_id}", access_key)
client.on_connect = on_connect
client.on_message = on_message

client.tls_set()
client.connect("eu1.cloud.thethings.network", 8883, 60)
```

Nous mentionnons donc que nous utilisons le protocole mqtt, puis les informations de connexion. Enfin, nous nous connectons sur l'adresse de notre serveur sur le port 8883 et nous récupérons les données.

3.2 Preuve de fonctionnement

Maintenant que nous avons fait le code, nous vérifions que cela fonctionne correctement

```
c:\Users\elisa\OneDrive\Bureau\BUT R&T\Semestre 3\SAE3.01\Traiteur.py:122:
  client = mqtt.Client(protocol=mqtt.MQTTv311)
Connecté avec le code de résultat 0
Message déchiffré: CO2=1659.45,Temp=23.25,Hum=40.69
```

Nous voyons ici que nous avons correctement développé la récupération des données sur TTN. Nous décodons donc bien notre message et l'affichons correctement sur invite de commande.

4. Graphique (google sheet + grafana)

4.1 Envoi des données dans un google sheet

Afin d'implémenter quelques fonctionnalités pour notre projet, nous avons décidé d'envoyer les données que nous avons récupérées préalablement sur un google sheet.
Pour cela, nous devons mettre en place plusieurs chose à commencer par l'implémentation des librairies :

```
import re
from google.oauth2 import service_account
from googleapiclient.discovery import build
```

Ces librairies pythons vont nous permettre de configurer la liaison avec notre google sheet et les envoyer les données sur celui-ci.

Ensuite nous devons configurer quelques informations concernant notre google sheet tel que son ID et la plage de nous allons devoir modifier.

```
spreadsheet_id = "1rS_3Hy9_SjsPAMTUE_iDa1YemXlTRHZvrPcgsWhxW0Y"
range_name = "Feuille 1!A:D"
```

Ici, nous avons donc notre ID du google sheet que nous obtenons ici :

```
docs.google.com/spreadsheets/d/1rS_3Hy9_SjsPAMTUE_iDa1YemXlTRHZvrPcgsWhxW0Y/edit?gid=0#gid=0
```

Nous mettons ensuite la plage que nous allons modifier en ajoutant au fur et à mesure les données de TTN.

Pour continuer, nous devons configurer le système d'authentification à notre Google Sheet. Pour cela, nous créons un fichier credentials.json (dont la structure est trouvable sur internet) et nous implémentons à notre code comme ceci :

```
# Authentification Google Sheets
SCOPES = ['https://www.googleapis.com/auth/spreadsheets']
SERVICE_ACCOUNT_FILE = 'credentials.json'

credentials = service_account.Credentials.from_service_account_file(
    SERVICE_ACCOUNT_FILE, scopes=SCOPES)
service = build('sheets', 'v4', credentials=credentials)
sheet = service.spreadsheets()
```

Une fois que nous avons réalisé ceci, nous sommes connectés à notre application, nous pouvons mettre en place les fonctions d'envoie sur le google sheet. Dans un premier temps, nous allons faire en sorte que si le google sheet est vide (la première ligne) cela écrivent les en-tête. Pour cela, un script existe aussi sur github et nous l'adaptons pour qu'il fonctionne sur notre application :

```
# Fonction pour ajouter l'en-tête si la feuille est vide
def add_header_if_empty():
    result = sheet.values().get(spreadsheetId=spreadsheet_id,
range=range_name).execute()
    values = result.get('values', [])
    if not values:
        header = [["Horodatage", "CO2 (ppm)", "Température (°C)",
"Humidité (%)"]]
        body = {
            'values': header
        }
        sheet.values().append(
            spreadsheetId=spreadsheet_id,
            range=range_name,
```

```
            valueInputOption="RAW",
            body=body
        ).execute()
        print("En-tête ajouté à la feuille Google Sheets.")
```

Ensuite, nous mettons en place le système d'envoie des données sur notre google sheet :

```
        values = [[timestamp, co2, temp, hum]]
        body = {
            'values': values
        }
        result = sheet.values().append(
            spreadsheetId=spreadsheet_id,
            range=range_name,
            valueInputOption="RAW",
            body=body
        ).execute()
        print(f"{result.get('updates').get('updatedCells')} cellules
mises à jour.")
```

ici, nous utilisons nos valeurs que nous envoyons ensuite sur notre site. Nous récupérons ensuite grâce à une requête le nombre de cellule mise à jour que nous affichons sur notre invite de commande.

Grâce à cela nous réussissons à envoyer les données sur un google sheet.

4.2 Démonstration de fonctionnement

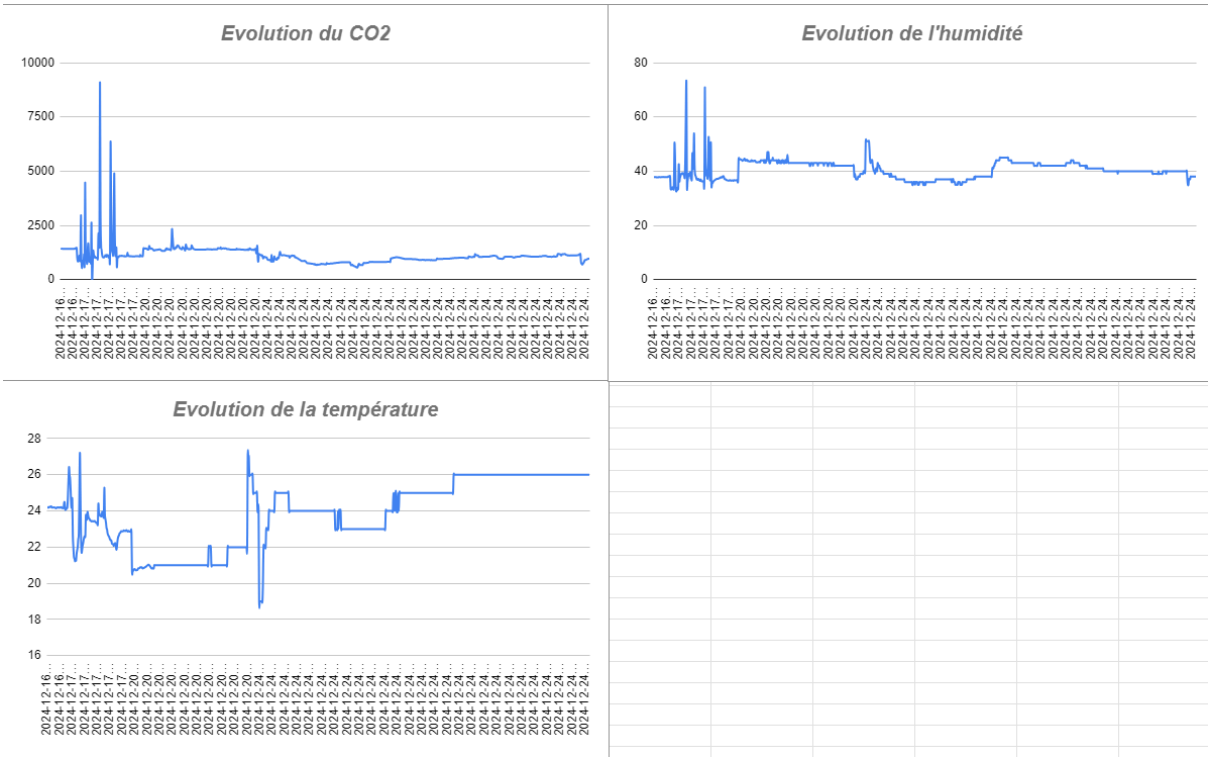
Maintenant , il nous reste à lancer notre code et vérifier si l'envoie fonctionne correctement. Nous avons donc ceci sur notre invite de commande

```
CO2: 1659.0, Temp: 23.0, Hum: 41.0
4 cellules mises à jour.
```

Nous voyons donc que nous avons réussi à mettre à jour des cellules. Reste à voir si cela fonctionne donc correctement :

Horodatage	CO2 (ppm)	Température (°C	Humidité (%)
2024-12-16 11:23:29	1438	24,17	37,85
2024-12-16 11:23:43	1430,79	24,19	37,84
2024-12-16 11:23:57	1428,82	24,22	37,8

On voit donc que les envoie de données fonctionnent bien sur notre google sheet. De plus, afin de rendre cet ajout utile, j'ai mis en place des graphiques permettant d'avoir un visuel en temps réel sur les données que nous récupérons.



Les graphiques que nous avons ci-dessus affichent plusieurs demains de données. Nous voyons ici que les graphiques fonctionnent correctement même avec plus de 50k données.

4.3 Mise en place de grafana

Ensuite, afin de mettre en place un système de graphique plus adéquat et aussi plus adapté, nous avons décidé de mettre en place un grafana. Pour cela, j'ai dû adapter le programme python afin qu'il envoie les données sur une base de données nommée InfluxDB. Ensuite, je devais configurer la base de données pour pouvoir récupérer les données se trouvant dessus et ainsi les faire afficher sur le grafana.

Pour cela, nous allons dans un premier temps voir la configuration de notre base de données InfluxDB.
J'ai donc dû configurer un bucket. En gros c'est une table dans la base de données :

Ensuite, j'ai mis en place une API me permettant d'avoir un lien entre ma base de données et mon programme python :

Il ne me restait donc plus qu'à envoyer les données de mon programme python sur cette base de données :

```
# Configuration InfluxDB Cloud
influxdb_url = "https://us-east-1-1.aws.cloud2.influxdata.com"
influxdb_token = "fmq7Uh0ponKfjS9E-V8kqEQnaoS8bNfVovWOJEy3zIwRjwM33E9pBFKCGReZl24g6rGJnVG6mwolTMkRCunufA=="
influxdb_org = "a954b6606f5da4d0"
influxdb_bucket = "SAE302"

client_influx = InfluxDBClient(url=influxdb_url, token=influxdb_token)
write_api = client_influx.write_api(write_options=SYNCHRONOUS)
```

Je configure donc dans un premier temps les information de ma base de donnée à savoir le bucket, le lien de InfluxDB, le token de mon API et de la même manière que le google sheet, l'identifiant de ma base de donnée que l'on trouve de cette manière

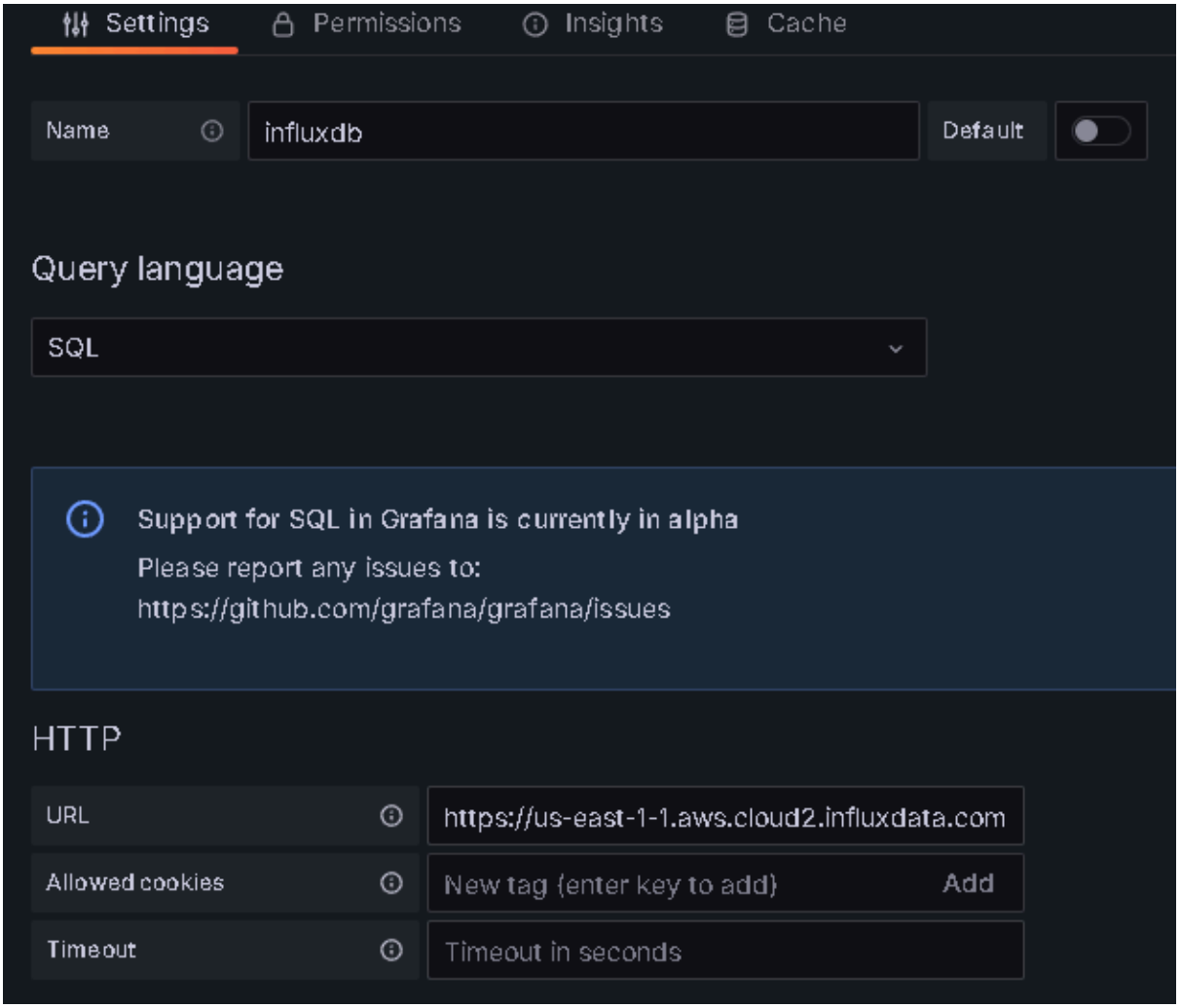
Ensuite, il ne reste plus qu'à coder l'envoie sur InfluxDB comme ceci :

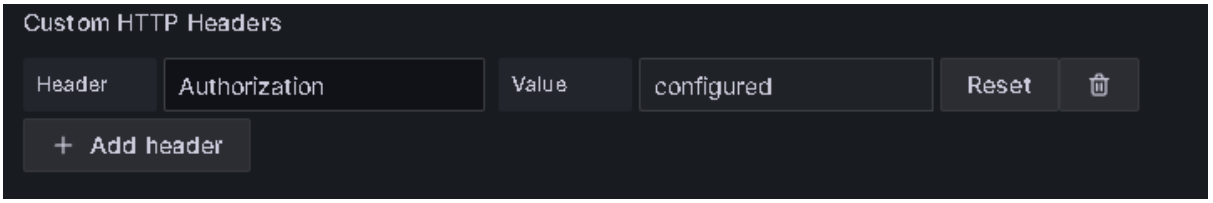
```
        # Envoyer les données à InfluxDB Cloud
        print(f"Envoi des données à InfluxDB: CO2={co2}, Temp={temp}, Hum={hum}")
        point = Point("donnees_capteur") \
            .tag("appareil", "capteur") \
            .field("co2", co2) \
            .field("temperature", temp) \
            .field("humidite", hum) \
            .time(datetime.utcnow(), WritePrecision.NS)
        write_api.write(bucket=influxdb_bucket, org=influxdb_org, record=point)
        print("Données envoyées à InfluxDB.")
    else:
        print("Format de message incorrect")
```

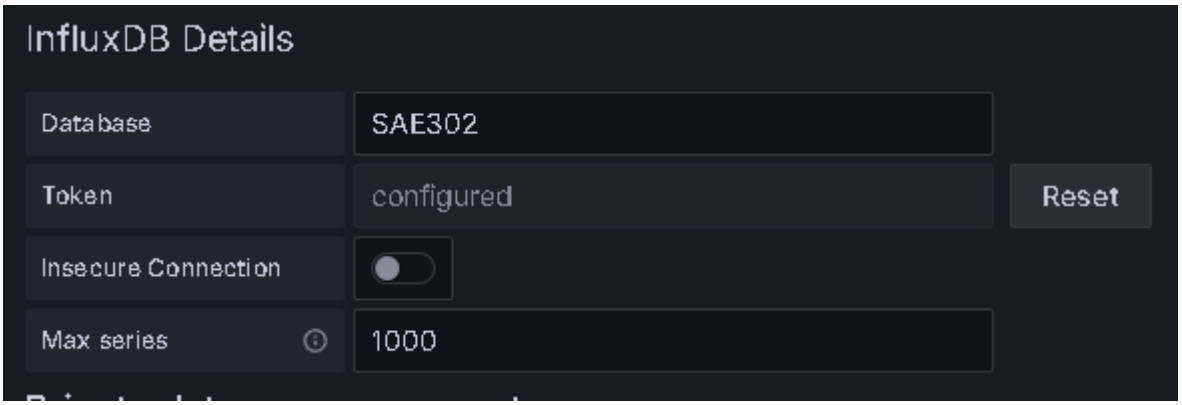
Nous avons donc un retour sur notre invite de commande de ce qu'on va envoyer. On mentionne aussi quel type de données on va envoyer pour que ça se mette sur la bonne table pour ensuite y incrémenter les données dans des champs. Nous rajoutons notre horodateur pour ensuite les envoyer sur InfluxDB. Si le format de donnée n'est pas bon, nous avons un retour pour expliquer le problème sinon nous avons un message indiquant que les données sont bien envoyé sur InfluxDB.

Maintenant que la configuration est bonne et que nous recevons les données sur InfluxDB, nous pouvons configurer notre grafana. Pour plus de facilité, nous avons configuré grafana cloud nous permettant d'avoir un visuel de n'importe ou sur nos graphiques.

Nous devons donc dans un premier temps ajouter une nouvelle source de donnée :

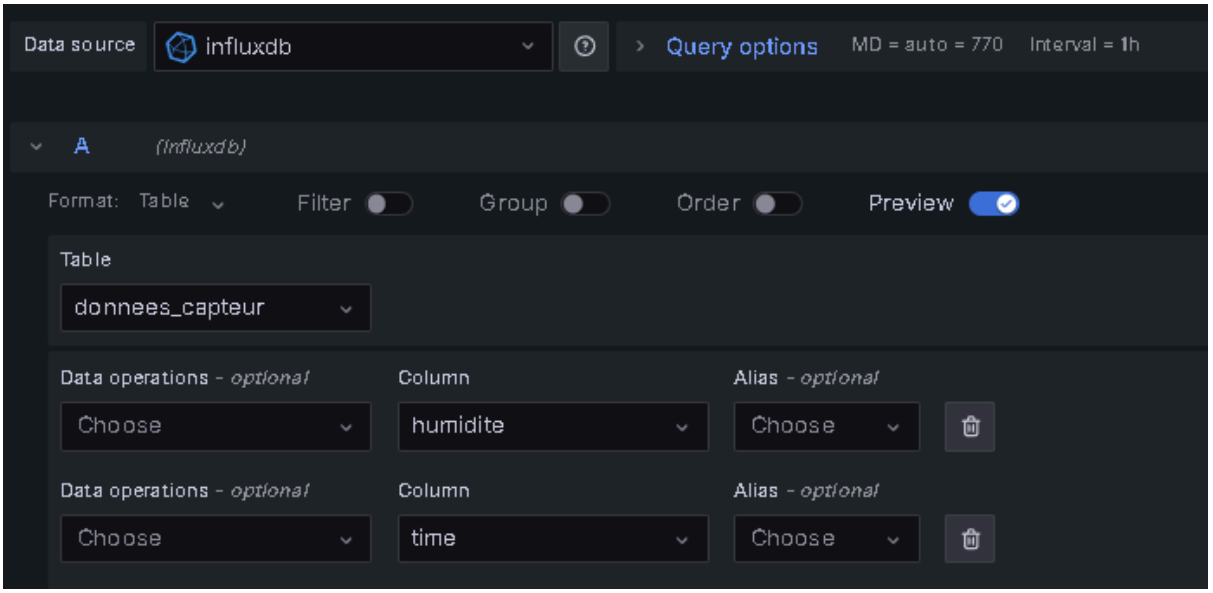






Nous configurons donc le nom, le type de base de données (SQL) le lien de celle-ci, la base de données (le bucket) puis enfin de token.

Il ne nous reste plus qu'à créer un graphique dans un nouveau dashboard comme ceci :



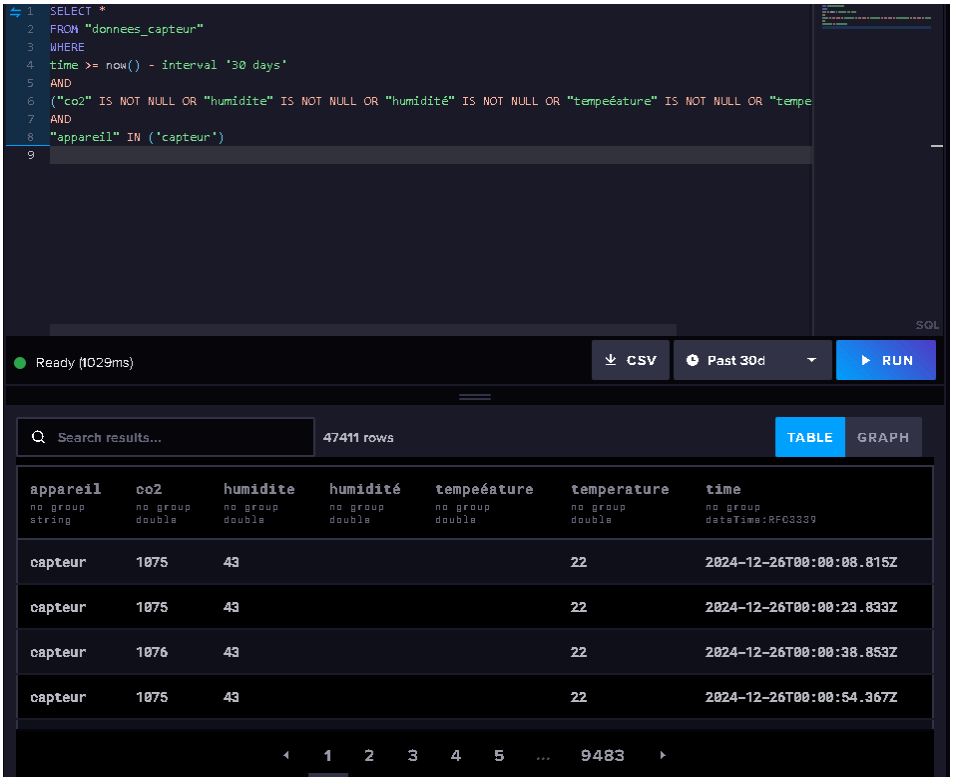
Nous sélectionnons les données que nous voulons afficher.

4.4 Preuve de fonctionnement

Pour montrer que cela fonctionne, nous allons afficher chaque étape de l'envoie de nos données pour prouver que cela fonctionne correctement.
Dans un premier temps sur le code python :

```
Envoi des données à InfluxDB: CO2=1659.0, Temp=23.0, Hum=41.0
Données envoyées à InfluxDB.
```

Nous voyons ici que les informations semblent bien s'envoyer. C'est pour cela que nous allons le vérifier sur InfluxDB :



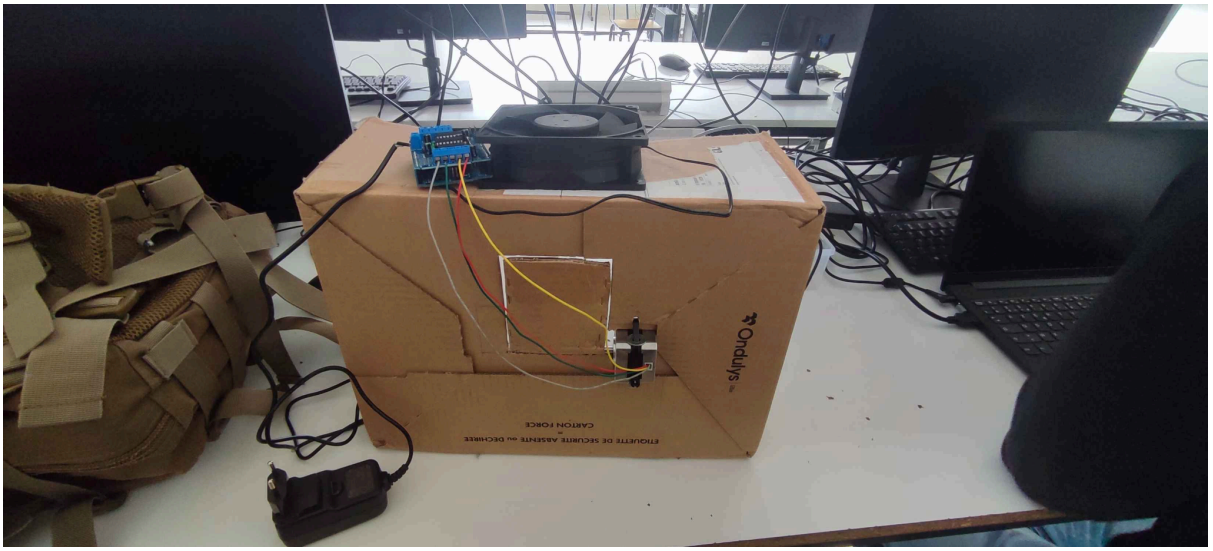
Ici, nous voyons bien que les données sont correctement envoyées sur InfluxDB. Il ne reste plus qu'à montrer le grafana :

Nous voyons donc bien nos graphiques ainsi qu'il moyen de la teneur en CO2 relevée. Etant donné que la plupart des informations ont été récupérées chez moi, nous remarquons que la teneur en CO2 est plutôt basse.

5. Traitement de la donnée (ventilo + moteur)

5.1 Ajout de la fonctionnalité du ventilateur et du moteur pas à pas

Pour finir notre projet, nous avons décidé d'ajouter un système intéressant permettant de montrer que les données sont récupérées mais aussi de les appliquer dans la vie réelle. Nous avons, grâce au matériel que nous avons reçu pour le CSAW, mis en place un système faisant tourner le ventilateur et ouvrant une fenêtre avec un moteur pas à pas. Pour cela, nous avons dans un premier temps fait une maquette



Ces composants sont tous connectés à un shield arduino (délié électriquement car nous branchons une prise 12V) qui est pluggé sur une carte arduino et reçoit des données.

Ensuite, nous avons créé un code nous permettant de faire tourner les composants en fonction des données que nous envoyons sur le moniteur série.

```
#include <AFMotor.h>

AF_DCMotor fan(1);

AF_Stepper stepper(200, 2);

int co2 = 0;
float temperature = 0.0;
float humidity = 0.0;


int stepperPosition = 0;
const int eighthTurnSteps = 25;
bool fanRunning = false;
bool stepperAtEighthTurn = false;

void setup() {
  Serial.begin(9600);
  Serial.println("Adafruit Motorshield v2 - Motor control");

  stepper.setSpeed(30);
}

void loop() {
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');
    parseData(data);
    controlStepperMotor();
    controlFan();
  }
}
```

```
void parseData(String data) {
  int firstSemicolon = data.indexOf(';');
  int secondSemicolon = data.indexOf(';', firstSemicolon + 1);

  co2 = data.substring(0, firstSemicolon).toInt();
  temperature = data.substring(firstSemicolon + 1, secondSemicolon).toFloat();
  humidity = data.substring(secondSemicolon + 1).toFloat();

  Serial.print("CO2: ");
  Serial.print(co2);
  Serial.print(" ppm, Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C, Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
}

void controlStepperMotor() {
  if (co2 > 1500 && !stepperAtEighthTurn) {
    stepper.step(eighthTurnSteps, FORWARD, SINGLE);
    stepperAtEighthTurn = true;
    stepperPosition = eighthTurnSteps;
  } else if (co2 < 1450 && stepperAtEighthTurn) {
    stepper.step(eighthTurnSteps, BACKWARD, SINGLE);
    stepperAtEighthTurn = false;
    stepperPosition = 0;
  }
}

void controlFan() {
  if (temperature > 26 && !fanRunning) {
    fan.setSpeed(255);
    fan.run(FORWARD);
    fanRunning = true;
  } else if (temperature < 22.0 && fanRunning) {
    fan.run(RELEASE);
    fanRunning = false;
  }
}
```

Ce code fait donc tourner le ventilateur si la température est supérieure à 26°C et ouvre la fenêtre si la quantité de CO2 dans la salle est supérieure à 1450 ppm.
Les données sont reçues sur un moniteur série mais pour cela, il reste à adapter notre code afin de pouvoir envoyer les données sur le moniteur série.
Pour cela, grâce au CSAW, j'ai appris à envoyer des données directement dans un moniteur série via un code python grâce à des fonctions.
Nous ajoutons donc ceci sur notre code :

```
import serial
```

Nous ajoutons notre librairie puis nous actualisons notre moniteur série :

```
# Configuration du port série
ser = serial.Serial('COM3', 9600)
```

Suite à cela, nous créons une fonction envoyant les données sur le moniteur série :

```
def send_and_read_data(data):
    ser.write(data.encode())
    print("Données envoyées:", data)
    time.sleep(5)
    if ser.in_waiting > 0:
       response = ser.read(ser.in_waiting).decode()
       print("Réponse reçue:", response)
    else:
        print("Aucune réponse reçue")
```

Suite à cela, nous avons donc ajouté toutes les fonctions nous permettant de faire fonctionner ceci.
Cependant, nous ne pouvons prouver le fonctionnement qu'en face à face et pas par des photos qui ne prouvent rien.
Cependant, le fonctionnement est donc correct et nous avons pu en conclure que nos salles de classe ont une teneur en CO2 beaucoup trop élevée. En effet, notre fenêtre était toujours ouverte si nous configurions le maximum réglementaire qui est de 1250 ppm.
Nous avons donc dû adapter notre code afin que cette fenêtre s'ouvre et se ferme.

6. Conclusion

Le but principal était de créer un objet connecté envoyant les données d'un capteur sur TTN.

Grâce au travail effectué, il a été possible de :

- Envoyer les données sur TTN
- Recevoir les données de TTN
- Traiter les données pour en faire des graphiques
- Traiter les données pour faire tourner des composants simulant des éléments domotiques.

Analyse personnelle :

Cette SAÉ nous a permis d'augmenter significativement nos compétences. Étant donné que nous n'avions pas de compétence concernant la création d'objet connecté , nous nous

sommes formé et avons découvert bon nous de nouvelles choses nous permettant de devenir encore plus polyvalent.

7. Annexes

Code source :

- Code arduino envoyant les données
- Code arduino faisant fonctionner ntore maquette
- Code python réalisant tout le post-envoi