

Application IoT

Quentin Chalvin M2-SICOM



Problématique	3
Cahier des charges	4
Solution matérielle	5
Solution logicielle	5
Mise en place	6
Programme python	7
Programme Arduino (C)	11
TTN payload decoder	14
Suite influx data	16
Fonctionnalités optionnelles	19
Fichiers présents dans l'archive	21

Problématique

- - - - X

La loi du 12 juillet 2010, prescrit la surveillance de la qualité de l'air intérieure pour certains établissements

recevant du public et l'étiquetage des polluants volatils sur les produits de construction et d'ameublement.

A partir de ce texte, les décrets d'application et arrêtés étayent le dispositif et sont retranscrits dans le code de l'environnement.

Les valeurs guides pour les composés organiques volatiles (formaldéhyde et le benzène) sont fixées et codifiées.

Le projet doit permettre de surveiller les composants suivants :

- Le CO₂
 - Les COV
 - La température
 - L'humidité
-

Cahier des charges

- - - - X

Le projet devra contenir plusieurs fonctionnalités :

- ❖ Développer une interface de supervision et configuration **locale** qui permet :
 - D'accéder aux mesures directement issues des capteurs
 - D'accéder aux mesures "historisées" sur 24h à raison d'une mesure toutes les 15 minutes
 - De "monitorer" la liaison LoRa (Afficher le RSSI et le SNR)
 - De régler les seuils limites pour les polluants CO2 et COV et pour les paramètres climatiques Temp et Hum afin d'agir sur la ventilation
 - De localiser la station de mesure au sein du CERI (Salles de cours/TD/TP, amphis, administration).
- ❖ Développer une interface de supervision et configuration **distante** qui permet :
 - D'accéder aux mesures des capteurs issues du cloud TTN ou de la passerelle LoRa Multitech
 - D'accéder aux mesures "historisées" sur une période réglable (horodatage début et fin)
 - De "monitorer" la liaison LoRa (Afficher le RSSI et le SNR)
 - De régler les seuils limites pour les polluants CO2 et COV et pour les paramètres climatiques Temp et Hum afin d'agir sur la ventilation
 - De localiser la station de mesure au sein du CERI (Salles de cours/TD/TP, amphis, administration).
- ❖ (**Optionnel**) La station pourra être munie d'un écran local 7" permettant d'afficher en continue les différentes mesures et alerter visuellement le public en cas de dépassement des seuils. L'affichage pourra prendre la forme d'un "dashboard" avec des widgets significatifs, d'un design sobre et épuré mais visible entre 2 et 5 mètres.

Solution matérielle

- - - - X

Le projet est composé de plusieurs cartes électroniques :

- ❖ Une carte ThingsUno basée sur une Arduino Leonardo qui permet la communication des données vers une gateway TTN.
- ❖ Une carte Raspberry pi 3 avec un shield composée de 2 capteurs:
 - Un capteur de CO2/COV
 - Un capteur de température/humidité
- ❖ Un écran de 7 pouces connecté à la raspberry pour l'affichage local
- ❖ Une gateway LoraOne pour la communication vers le réseau TTN

Solution logicielle

- - - - X

Le projet se compose d'une multitude de logiciels :

- **Node-red** pour l'affichage local
- **Python** et diverses bibliothèques pour l'extraction de données des capteurs et l'envoi vers la carte ThingsUNO.
- **Arduino (C)** pour la récupération des données de la raspberry et pour la communication vers les gateways LoraWAN.
- Suite **influxdata** pour l'affichage distant:
 - ◆ Influxdb pour le stockage ;
 - ◆ Telegraf pour récupérer les données TTN et les stocker ;
 - ◆ Chronograf pour afficher les données ;
 - ◆ Kapacitor pour la gestion des alertes mail.

Mise en place

- - - - X

Dans un premier temps nous allons mettre en place la partie IoT du projet : installation des cartes et extraction des données.

Le système d'exploitation pour la raspberry est Raspberry Pi OS trouvable sur le site officiel :

<https://www.raspberrypi.org/downloads/>

Il est nécessaire d'installer l'ide d'arduino disponible sur leur site officicel : <https://www.arduino.cc/en/software>

Les paquets pour Python3 et Node-red sont aussi nécessaires :

```
sudo apt install python3  
sudo apt-get -y install python3-pip
```

et

```
bash <(curl -sL  
https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)
```

Le flow node red est disponible dans l'archive. Il sera nécessaire d'installer les plugins suivants :

- node-red-contrib-ui-led
- node-red-dashboard

Programme python

- - - - X

Le programme python permet de récupérer les valeurs des capteurs, de les traiter et de les enregistrer dans un fichier json.

Il est nécessaire d'installer les paquets suivants pour le bon fonctionnement du programme :

```
import time
import board
import adafruit_dht
import json
import busio
import serial
from statistics import mean
import sys
```

<https://learn.adafruit.com/dht-humidity-sensing-on-raspberry-pi-with-gdocs-logging/python-setup>

https://circuitpython.readthedocs.io/en/3.x/shared-bindings/busio/_init_.html

La fonction “computation” fait X relevés des différentes sondes et calcule la moyenne.

```
def computation():
    loop = 2
    i = 0
    temp = [0] * loop
    hum = [0] * loop
    co2 = [0] * loop
    cov = [0] * loop
    status = 0
    tmp = bytearray(9)
    while i < loop:

        try:
            i2c.readfrom_into(90, tmp)
            co2[i] = tmp[0] * 256 + tmp[1]
            status = tmp[2] * 256
            cov[i] = tmp[7] * 256 + tmp[8]
        except Exception as e:
            print("error" + str(e))
            time.sleep(2)

        try:
            temperature_c = dhtDevice.temperature
            humidity = dhtDevice.humidity
            temp[i] = temperature_c
            hum[i] = humidity
            i += 1

        except RuntimeError as error:
            print(error.args[0])
            time.sleep(2.0)
            continue
        except Exception as error:
            dhtDevice.exit()
            raise error
        time.sleep(2.0)

    return [temp, hum, co2, cov, status]
```


La fonction “to_json” permet d’écrire les données précédemment relevées, celles-ci sont d’abord arrondies à 1 chiffre après la virgule et multipliées par 10 pour obtenir un entier. Cette fonction facilite en effet la transmission de données pour le réseau TTN, le tout étant enregistré dans un fichier json avec l’architecture suivante :

```
def to_json(temp, hum, co2, cov, status):
    data_dict = ""

    with open('data.json') as json_data:
        data_dict = json.load(json_data)
        data_dict["dht22"]["temp"] = int(round(temp, 1) * 10)
        data_dict["dht22"]["hum"] = int(round(hum, 1) * 10)
        data_dict["iaq"]["co2"] = int(round(co2, 1) * 10)
        data_dict["iaq"]["cov"] = int(round(cov, 1) * 10)
        data_dict["iaq"]["status"] = status
        json_data.close()

    with open('data.json', 'w') as outfile:
        json.dump(data_dict, outfile, indent = 4)
        outfile.close()
```

Enfin, la fonction “main” initialise les variables pour lire les données et les envoie vers le port serial (depuis le fichier json vers la carte ThingsUNO). Les données sont envoyées tant que la carte ThingsUNO ne nous répond pas avec un accusé de réception. Les valeurs sont envoyées au réseau TTN toutes les 15 minutes.

```
if __name__ == "__main__":
    dhtDevice = adafruit_dht.DHT22(board.D8, use_pulseio=False)
    i2c = busio.I2C(board.SCL, board.SDA)
    temp, hum , co2, cov, status = computation()
    to_json(mean(temp), mean(hum), mean(co2), mean(cov), status)
    with open('data.json') as json_data:
        data_dict = json.load(json_data)
        payload = "data;"
        payload += str(data_dict["dht22"]["hum"]) + ";" +
str(data_dict["dht22"]["temp"]) + ";" + str(data_dict["iaq"]["co2"]) + ";"
+ str(data_dict["iaq"]["cov"]) + ";" + str(data_dict["iaq"]["status"])
        print(payload)
        if len(sys.argv) > 1 and sys.argv[1] == "ttn":
            with serial.Serial("/dev/ttyACM0", 9600, timeout=1) as arduino:
                time.sleep(0.1) #wait for serial to open
                if arduino.isOpen():
                    print("{} connected!".format(arduino.port))
                    done = True
                    try:
                        while done:
                            arduino.write(str.encode(payload))
                            while arduino.inWaiting()==0: pass
                            if arduino.inWaiting()>0:
                                answer = arduino.readline()

                                while b"ok\r\n" not in answer:
                                    arduino.write(str.encode(payload))
                                    answer = arduino.readline()
                                    time.sleep(1)
                                done = False
                                arduino.flushInput()
                    except KeyboardInterrupt:
                        print("KeyboardInterrupt has been caught.")
```

Programme Arduino (C)

- - - - X

Le programme arduino permet l'envoi de données vers le réseau TTN.

Le programme est en partie fourni par TTN :

<https://www.thethingsnetwork.org/docs/devices/arduino/usage.html>

La première partie du programme sert de paramétrage, elle comprend entre autres les bibliothèques nécessaires, les clés pour l'activation OTAA, le type de capteur et la fréquence de communication européenne :

```
#include <TheThingsNetwork.h>
#include <DHT.h>

const char *appEui = "70B3D57ED003638A";
const char *appKey = "34042FE95BFF6584CBA04406BBA2223D";

#define loraSerial Serial1
#define debugSerial Serial

#define freqPlan TTN_FP_EU868

#define DHTPIN 2

#define DHTTYPE DHT22

String msg = "";
DHT dht(DHTPIN, DHTTYPE);
TheThingsNetwork ttn(loraSerial, debugSerial, freqPlan);
```

La partie Setup permet de rejoindre le réseau TTN via une gateway :

```
void setup(){
  loraSerial.begin(57600);
  debugSerial.begin(9600);

  // Wait a maximum of 10s for Serial Monitor
  while (!debugSerial && millis() < 10000);

  //debugSerial.println("-- STATUS");
  ttn.showStatus();

  //debugSerial.println("-- JOIN");
  ttn.join(appEui, appKey);

  dht.begin();
}
```

Cette fonction permet de lire l'entrée du port serial :

```
void readSerialPort() {
  if (Serial.available()) {
    delay(10);
    while (Serial.available() > 0) {
      msg += (char)Serial.read();
    }
    Serial.flush();
  }
}
```

Cette fonction permet d'envoyer l'accusé de réception des données à la raspberry :

```
void sendData() {
  Serial.println("ok");
}
```

Cette fonction permet de couper les valeurs reçues dans un message :

```
String getValue(String data, char separator, int index){
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

Cette partie du code permet de récupérer les données lues sur le port serial et de former un payload compréhensible pour le réseaux TTN qui contient 10 mots de 2 octets (8 bits) :

```
void loop(){
    //debugSerial.println("-- LOOP");
    msg = "";
    readSerialPort();
    String test = getValue(msg, ';', 0);

    if (test == "data") {
        sendData();
        delay(10);
        uint16_t hum = getValue(msg, ';', 1).toInt();
        uint16_t temp = getValue(msg, ';', 2).toInt();
        uint16_t co2 = getValue(msg, ';', 3).toInt();
        uint16_t cov = getValue(msg, ';', 4).toInt();
        uint16_t sta = getValue(msg, ';', 5).toInt();

        // Split both words (16 bits) into 2 bytes of 8
        byte payload[10];
        payload[0] = highByte(hum);
        payload[1] = lowByte(hum);
        payload[2] = highByte(temp);
        payload[3] = lowByte(temp);
        payload[4] = highByte(co2);
        payload[5] = lowByte(co2);
        payload[6] = highByte(cov);
        payload[7] = lowByte(cov);
        payload[8] = highByte(sta);
        payload[9] = lowByte(sta);

        ttn.sendBytes(payload, sizeof(payload));
    }
    delay(20000);
}
```

TTN payload decoder

Cette fonction est présente dans le cloud TTN, elle permet de retranscrire des données brutes en un format json :

```
function Decoder(bytes, port) {
  var decoded = {};

  var hum = bytes[0] * 256 + bytes[1];
  var temp = bytes[2] * 256 + bytes[3];
  var co2 = bytes[4] * 256 + bytes[5];
  var cov = bytes[6] * 256 + bytes[7];
  var status = bytes[8] * 256 + bytes[9];

  decoded.humidity = hum / 10;
  decoded.temperature = temp / 10;
  decoded.co2 = co2 / 10;
  decoded.cov = cov / 10;
  decoded.status = status

  return decoded;
}
```

Exemple sur TTN :

Payload

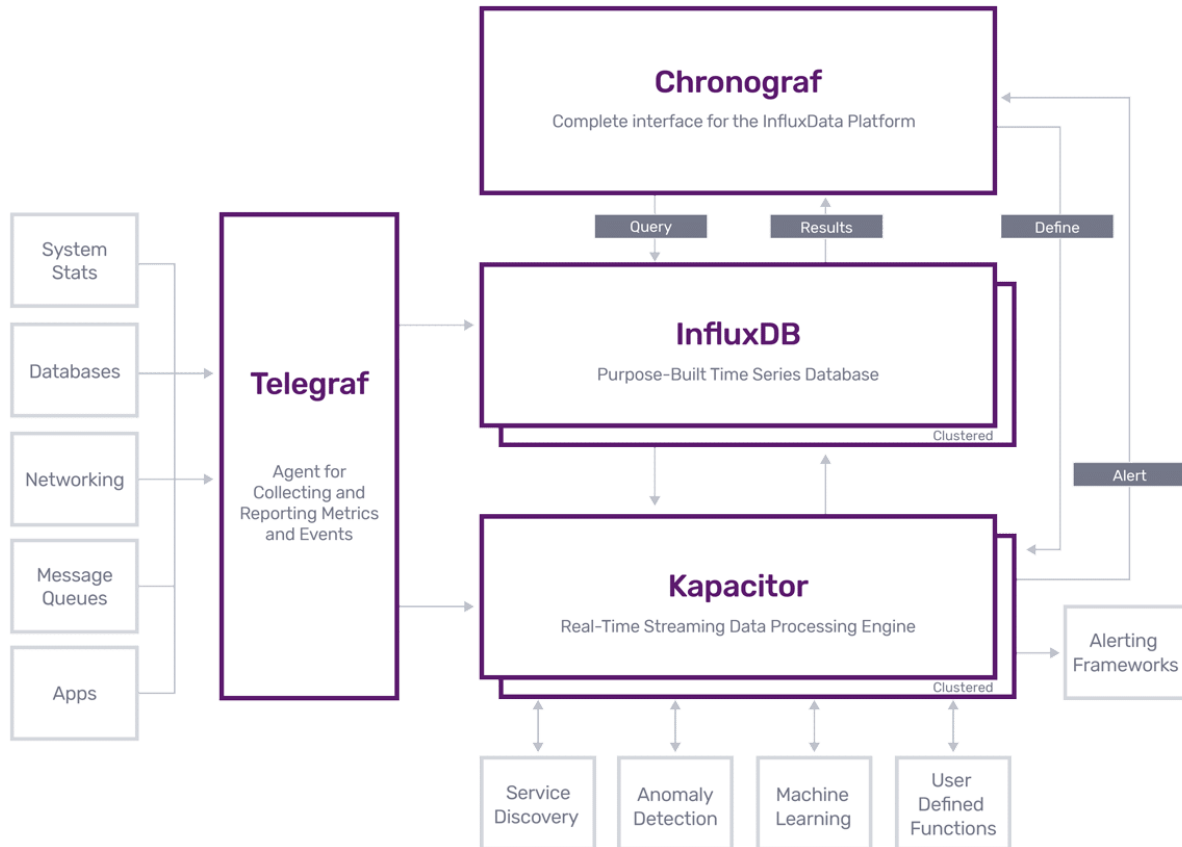
00 F0 00 FF 0F F0 04 FF 00 00

```
{
  "co2": 408,
  "cov": 127.9,
  "humidity": 24,
  "status": 0,
  "temperature": 25.5
}
```

Suite influx data

- - - - X

Pour la mise en place du serveur distant, j'ai utilisé la suite entière de influxdata, en voici le schéma fonctionnel :



Tout d'abord, le module Telegraf va permettre d'alimenter la base influxdb avec les données TTN.

Ensuite, le module Chronograf permet l'affichage, sous forme de dashboard, des données contenues dans la base influxdb.

Enfin, le module Kapacitor permet la gestion des alertes.

J'ai intégré la stack influxdb dans une machine virtuelle opérationnelle qui vous sera fournie. (login: influx, mdp: influx)

Pour mettre en place la stack, j'ai utilisé une machine virtuelle sous Ubuntu serveur. Dans un premier temps, il faut installer les modules disponibles ici : <https://portal.influxdata.com/downloads/>

Influxdb et Telegraf

```
sudo apt-get update && sudo apt-get install apt-transport-https
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -

sudo apt-get update && sudo apt-get install influxdb
sudo systemctl unmask influxdb.service
sudo systemctl start influxdb

sudo apt-get update && sudo apt-get install telegraf
sudo systemctl start telegraf
```

Chronograf :

```
wget https://dl.influxdata.com/chronograf/releases/chronograf_1.8.8_amd64.deb
sudo dpkg -i chronograf_1.8.8_amd64.deb
sudo systemctl start telegraf
```

Kapacitor s'installe directement dans l'ihm du module de Chronographe.

Le fichier de configuration de Telegraf est le suivant :
 /etc/telegraf/telegraf.conf,
 Voici la configuration nécessaire (adaptez les champs à votre infrastructure !) :

```
[[outputs.influxdb]]
urls = ["http://localhost:8086"]
database = "ttnadb"
username = ""
password = ""

[[inputs.mqtt_consumer]]
servers = ["tcp://eu.thethings.network:1883"]
qos = 0
connection_timeout = "30s"
topics = [
  "app_air_quality_chalvin_ceri_2020/devices/air_quality_sensors/up" ]
client_id = ""
username = "app_air_quality_chalvin_ceri_2020"
password = "ttn-account-v2.Af3u0eUoI9CdSsuR37ZltxSF0swhoftBFfxlQvYG3Hg"
data_format = "json"
```

Pour l'installation de Chronograf et de Kapacitor, il suffit de suivre les instructions sur l'ihm <http://<ip>:8888/>

Tous les paramétrages comme les dashboards et les alertes se font sur l'ihm avec des requêtes influxQL.

Fonctionnalités optionnelles

- - - - X

Pour cette fonctionnalité j'ai choisi d'implémenter mon propre dashboard.

Tout d'abord, j'ai développé un script python qui permet de créer un serveur web, j'ai également utilisé la base de données json pour afficher les derniers relevés des données.

Le dashboard est développé en html/css avec le framework Bootstrap.

Ensuite, j'ai mis en place différents services pour permettre le démarrage automatique du serveur web ainsi que de l'affichage web en plein écran.

Voici le service qui active le serveur web au démarrage :

/lib/systemd/system/pythonhttp.service

```
[Unit]
Description=Auto start python web server
After=multi-user.target
Conflicts=getty@tty1.service

[Service]
Type=simple
WorkingDirectory=/home/pi/Desktop/local_dash/
ExecStart=/usr/bin/python3 server.py
StandardInput=tty-force

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

Il faut ensuite activer le service avec les commandes :

```
sudo systemctl daemon-reload
sudo systemctl enable pythonhttp.service
sudo systemctl start pythonhttp.service
```

Pour que le navigateur s'ouvre tout seul il est nécessaire d'ajouter au fichier `/etc/xdg/lxsession/LXDE-pi/autostart` la commande suivante :

```
/usr/bin/chromium-browser --kiosk --disable-restore-session-state  
--start-fullscreen http://localhost:8888
```

Mon dashboard affiche les 4 relevées des sondes ainsi qu'un rapport sur l'état de la qualité de l'air. Selon les résultats, l'ihm affiche des recommandations pour améliorer la qualité de l'air.

Fichiers présents dans l'archive

- - - - X

- **crontab.txt** => fichier d'automatisation pour lancer les programmes à différentes intervalles de temps
- **autostartihm.txt** => commande pour lancer l'ihm locale automatiquement au démarrage de la raspberry
- **data.json** => bdd locale pour l'ihm
- **flow.json** => flow node red
- **ttn_send.ino** => script C pour l'envoi de données sur le réseau TTN
- **sonde.py** => script python pour l'extraction des données des sondes
- **local_dash** => dossier contenant toutes les sources pour l'ihm locale
- **influx.ova** => machine virtuelle qui contient l'os Ubuntu server avec la stack influxdb (lien drive car trop lourd pour l'ent)
https://drive.google.com/file/d/1J8MB3A1M3NTS_4GhHzU2hIPJgTluNUJd/view?usp=sharing