Mise en place d'une mini « station météo » capable de relever température, humidité, pression, qualité de l'air, etc...

DeltaLab

Espace Maison Milon
2 Place E.Colongin
84600 Grillon

deltalabprototype.fr

Qu’est-ce que DeltaLab ?

DeltaLab est une association ‘loi 1901’ d’intérêt général, dont l’objectif est la création d’un espace dédié à l’innovation, à la production numérique au prototypage et à l’«expression artistique».

Le principe fondateur de l’associtation est **d’apprendre à faire soi-même, pour passer de l’dée à l’objet.**

Deltalab se spécialise en **Objets Connectés**, et est en train de créer un vaste «écosystème digital» entre Drôme et Vaucluse, pour répondre à des besoins non-couverts, mettre à disposition ressources et équipements pour usage professionnels et instaurer des partenariats avec les autres structures et initiatives numériques existantes.

Deltalab est aussi un **FabLab** (*Fabrication Laboratory / Laboratoire de Fabrication*), un tiers-lieu de type makerspace ou se trouve un atelier qui dispose de machines de fabrication comme des Imprimantes 3D ou des découpeuses Laser.

Deltalab se veut ouvert à tous publics : étudiants, professionnels, associations, inventeurs, designers,artistes, ...

Contexte de cette Documentation

Un des projets à moyen termes de DeltaLab est l’installation d’un réseau d’antennes LORIX (ou autres) pour créer un réseau LoRaWAN permettant la collecte et le traitement d’un grand nombre de données , et ce sur une grande zone s’étendant sur l’enclave des Papes et la Drôme Provencale.

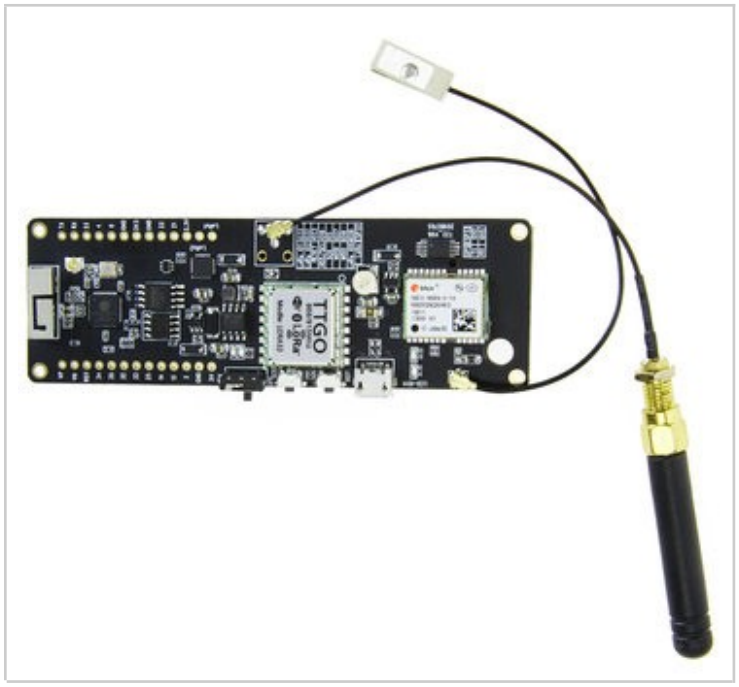
Cette documentation présente la mise en place d’une station météo centrée sur une carte arduino, utilisant le réseau LoRaWAN pour transmettre des données telles que la température, l’humidité, la qualité de l’air, la vitesse et la direction du vent, la quantité de pluie récente ou encore sa position GPS.

Table des matières

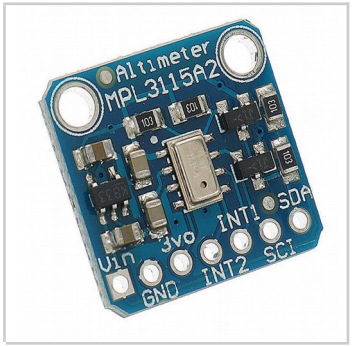
I - Introduction & Prérequis

Le but est de construire une station météo autonome grâce à une carte TTGO et quelques capteurs et appareils de mesure , et de récupérer les données sur un serveur Node-RED. Dans ce projet , les materiels utilisés sont :

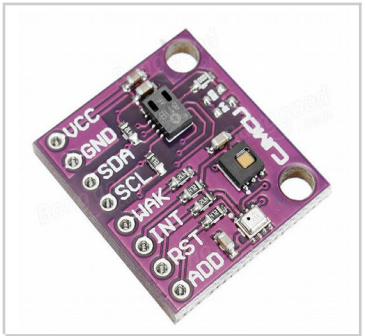
- **1 Carte TTGO Lora32 :**



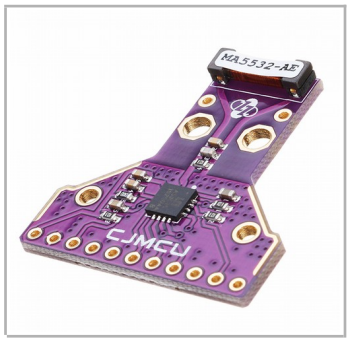
- **Capteur MPL3115A2**



- **Capteur CJMCU8128**



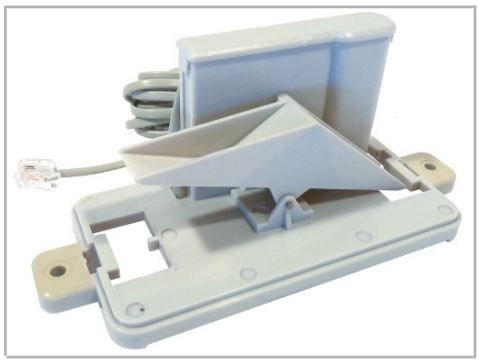
- **Capteur CJMCU3935**



- **Anémomètre**

- **Girouette**

- **Capteur de Pluie**



Les capteurs seront chainés en **I2C** sur la TTGO, qui transmettra ses données à un serveur de type Node-RED , via LoRa et The Things Network. Ils utiliseront les pins SCL et SDA de la TTGO.

La TTGO utilise une batterie de type Li-Ion à 3300 mA. On peut aussi y brancher un panneau solaire ou tout générateur d'énerggie , en utilisant un pin prévu pour recharger la batterie , ainsi que tout type de chargeur microUSB, grâce au port microUSB prévu à cet effet.

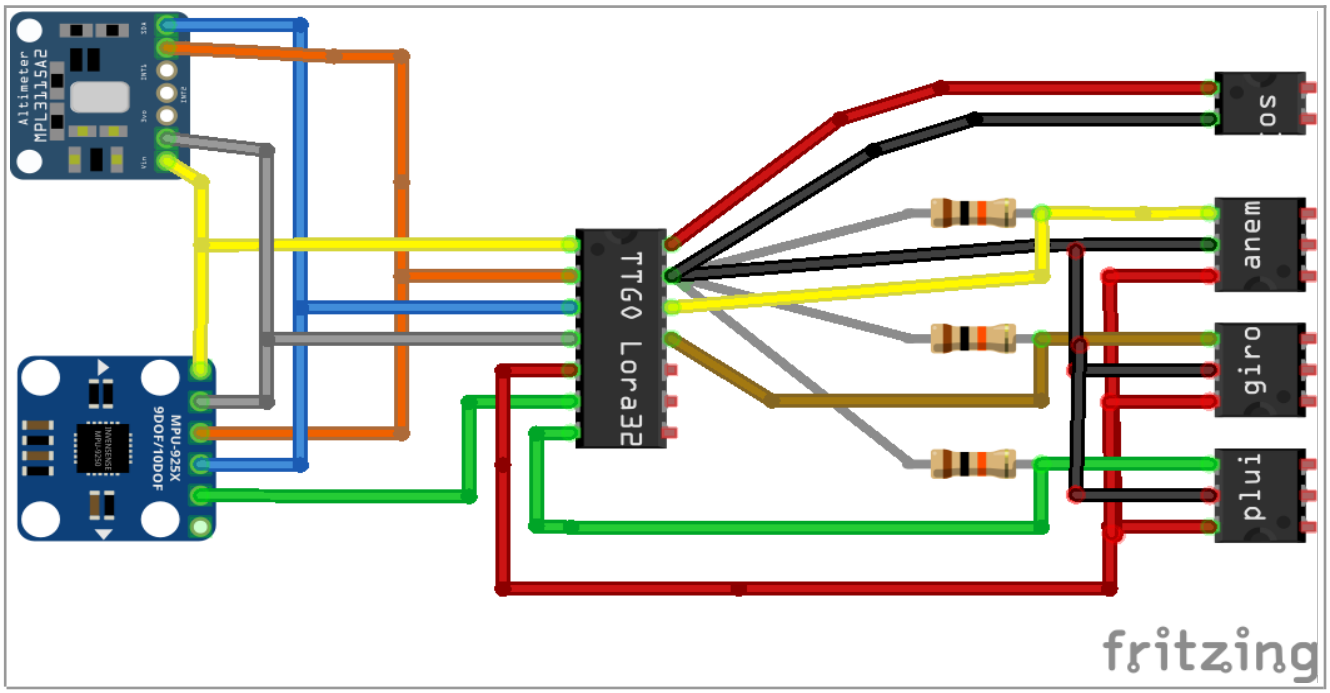
Les pins utiles de la TTGO sont :

- **2** : Pin digital , pour l'anémomètre
- **4** : Pin analogique pour le capteur de Pluie
- **13** : Pin analogique pour la girouette
- **21** : Pin **SDA** , pour les capteurs
- **22** : Pin **SCL**, pour les capteurs
- **3,3V** : Sortie d'alimentation en 3 Volts, pour les capteurs
- **GND** : Ground
- **5v** : Entrée /Sortie d'alimentation en 3,3Volts pour le panneau solaire.

Ce projet est réalisé sous arduino 1.8.12

II - Présentation du Circuit

Circuit



TTGO	MPL3115A2	CJMCU8218	Anémomètre	Girouette	pluie	Couleur	solaire
3.3V	VCC	VCC	Power	Power	Power	Rouge	-
GND	GND	GND	GND	GND	GND	Noir	GND
22	SCL	SCL	-	-	-	Jaune	-
21	SDA	SDA	-	-	-	Bleu	-
2	-	-	Data	-	-	Vert	-
4	-	-	-	-	Data	Vert	-
13	-	-	-	Data	-	Vert	-
5V	-	-	-	-	-	Rouge	Power

L'anémomètre, la girouette et le capteur de pluie doivent passer par une résistance. Celles utilisées ici ont une résistance de 10KΩ

III - Le code Arduino

- Chaque capteur possède sa librairie, ses fonctions et variables, et les exemples de codes donnés dans les librairies seront souvent suffisants

- La TTGO enverra les données des capteurs via LoRaWAN, pour simuler une station éloignée nécéssitant un protocole longue distance, mais il est bien sur possible de changer ça pour du MQTT ou n'importe quel autre protocole de communication.

- **Librairies utilisées :**
  - **Adafruit Unifed Sensor Library - version 1.1.2**
  - **Adafruit BMP280 Library - version 2.0.1**
  - **Adafruit CCS811 Library - version 1.0.2**
  - **Adafruit Si7021 Library - version 1.2.3**
  - **Adafruit MPL3115A2 Library - version 1.2.2**
  - **MCCI LoRaWAN LMIC Library - version 3.0**
  - **TinyGPS++ - Version 1.2.5**

- **CODE :**

```
#include <SD.h>
#include <SimpleTimer.h>
#include <Wire.h>
#include <HardwareSerial.h>
#include <TinyGPS++.h>
#include <arduino_lmic.h>
#include "lmic.h"
#include "hal/hal.h"
#include "arduino_lmic_hal_configuration.h"
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BMP280.h>
#include <Adafruit_CCS811.h>
#include <Adafruit_Si7021.h>
#include <Adafruit_MPL3115A2.h>
#include <AsyncDelay.h>
#include <SoftWire.h>
#include <AS3935.h>

#define PRESSIONZERO_HPA (1013.25)
#ifdef JTD
#include <DisableJTAG.h>
#endif
#define Pin_anemo 2
#define Pin_rain 4
#define Pin_giro 13
```

```
#define SCK     5    // GPIO5  -- SX1278's SCK
#define MISO    19   // GPIO19 -- SX1278's MISnO
#define MOSI    27   // GPIO27 -- SX1278's MOSI
#define SS      18   // GPIO18 -- SX1278's CS
#define RST     14   // GPIO14 -- SX1278's RESET
#define DI0 26 // GPIO26 -- SX1278's IRQ(Interrupt Request)
#define GPS_RX 12
#define GPS_TX 15

Adafruit_CCS811 ccs;
Adafruit_BMP280 bmp;
Adafruit_BMP280 cjbmp;
Adafruit_Si7021 si;
Adafruit_MPL3115A2 mpl = Adafruit_MPL3115A2();
TinyGPSPlus gps;
HardwareSerial GPSSerial(1);

float latitude;
float longitude;
int rotations;
float pluie = 0.279; //pluie par switch, env 0.3mm
float total = 0.0; // pluie totale
SimpleTimer timer_heure(3600000); // 1 heure = 3600000 millisecondes
SimpleTimer timer_jour(86400000); // 1 jour = 24 heures = 86400000 secondes
float pluie_heure = 0.0;
float pluie_jour = 0.0;
int ContactBounceTime = 10;
float dist = 0.0;
float rais = 0.0;
bool ok = true;
int direction;
int offset;
int taille = 0;
int id = 0;
int giro_value;      //Valeur brute de la girouette
double Direction;       //Convertie en degré (entre 0 et 360)
double off_direction;    //Convertie avec l'offset
static osjob_t envoi;
const unsigned TX_INTERVAL = 20;  //intervalle entre deux mesures / envois sur ttn

typedef union {        // données envoyées par les capteurs I2C - toutes les 20 minutes
  float f[10];
  unsigned char bytes[40];
} donnees_capt;
donnees_capt datas;

// L'EUI de votre application ttn , en format lsb
static const u1_t PROGMEM APPEUI[8] = { 0xB4, 0xC4, 0x02, 0xD0, 0x7E, 0xD5, 0xB3, 0x70 };
void os_getArtEui (u1_t* buf) {  memcpy_P(buf, APPEUI, 8);   }
```

```
// L'EUI de votre device , en format lsb
static const u1_t PROGMEM DEVEUI[8] = { 0x9B, 0x65, 0x36, 0xEF, 0x6D, 0xEE, 0x61, 0x00 };
void os_getDevEui (u1_t* buf) {  memcpy_P(buf, DEVEUI, 8);  }

//L'application key , en format msb
tatic const u1_t PROGMEM APPKEY[16] = { 0xCF, 0x28, 0xD3, 0x19, 0x4C, 0xCB, 0x44, 0xC4, 0xDA,
0x5A, 0x66, 0x98, 0xE0, 0x99, 0x70, 0x87 };
void os_getDevKey (u1_t* buf) {  memcpy_P(buf, APPKEY, 16);   }

//mappage des pins
const lmic_pinmap lmic_pins = {
  .nss = 18,
  .rxtx = LMIC_UNUSED_PIN,
  .rst = 14,
  .dio = {26, 33, 32},           };

//récupération des coordonnées gps
void get_coords () {
        while (GPSSerial.available()) {   gps.encode(GPSSerial.read());  }
        latitude  = gps.location.lat();
        longitude = gps.location.lng();
        Serial.print(gps.location.lat());
        datas.f[id] = latitude;
        id++;
        datas.f[id] = longitude;
         id++;
}

//Récupère les données du capteur MPL 3115 A2
void mesure_mpl(){
mpl.begin();
    datas.f[id] = float(mpl.getTemperature());
    id++;
    datas.f[id] = float(mpl.getPressure()*1,7) ;
    id++;
   datas.f[id] = float(mpl.getAltitude() / 140);
    id++;
}


//Récupère les données du capteur BMP280 du CJMCU-8128
void mesure_cjbmp(){
    datas.f[id]=cjbmp.readTemperature();
    id++;
    datas.f[id]=cjbmp.readPressure()/ 100.0F;
    id++;
   datas.f[id]=cjbmp.readAltitude(PRESSIONZERO_HPA);
    id++;
}
```

```
//Récupère les données du capteur SI7010
void mesure_si(){
  datas.f[id]=si.readTemperature()*0.80;
  id++;
  datas.f[id]=si.readHumidity();
  id++;
}


//Récupère les données du capteur CCS811 du CJMCU-8128
void mesure_ccs(){
    if(ccs.available()){
        if(!ccs.readData()){
            datas.f[id]=ccs.geteCO2();
            id++;
            datas.f[id]=ccs.getTVOC();
            id++;
        }
    }
}


//anémomètre : détection d'une rotation
void isr_rotation(){
     if((millis() - ContactBounceTime) > 5 ) {
        rotations++;
        ContactBounceTime = millis();
     }
}


//vitesse du vent
void getVitesse(){
    float vitesse = rotations * (2.25/TX_INTERVAL); //donne la vitesse en miles per hour
    datas.f[id] = vitesse / 1.609 ; // conversion en km/h
    id++;
    rotations = 0; //réinitialisation des rotations
}


// Mesure de la direction du vent
void getDirection(){
    giro_value = analogRead(Pin_giro);
    Serial.print(giro_value);
    Direction = getSens(giro_value);
    off_direction = Direction + Offset_giro;
}
```

```
// Traduction de la direction en sens cardinaux
double getSens(int direction){
   if((direction >700 && direction <900 )|| (direction >1750 && direction < 1900 )){
      Serial.print(" N");
      return 0.0;
   }else if (direction >2200 && direction < 2600){
      Serial.print(" NE");
      return 45.0;
   }else if (direction >3800 && direction < 3950){
      Serial.print(" E");
      return 90.0;
   }else if (direction >3200 && direction < 3450){
      Serial.print(" SE");
      return 135.0;
   }else if (direction >2800 && direction < 3050){
      Serial.print(" S");
      return 180.0;
    }else if (direction >1400 && direction < 1500){
      Serial.print(" SO");
      return 225.0;
    }else if ((direction >150 && direction < 250) || (direction >1500 && direction < 1650)){
      Serial.print(" O");
      return 270.0;
    }else if (direction >1650 && direction < 1750){
      Serial.print(" NO");
      return 335.0;
   }else{
      Serial.print(" ERR");
      return 0.0;
   }
}

// Pluie : détection d'un basculement du capteur
void rain(){
        int reading = digitalRead(Pin_rain);
        if((millis() - ContactBounceTime) > 5){
                if(timer_jour.isReady()){
                        timer_jour.reset();
                        pluie_jour = 0.0;
                if(timer_heure.isReady()){
                        timer_heure.reset();
                        pluie_jour += pluie_heure;
                        total = pluie_heure;
                        pluie_heure = 0.0;
                }else{
                        pluie_heure += pluie;
                }
                delay(150);
        }
}
```

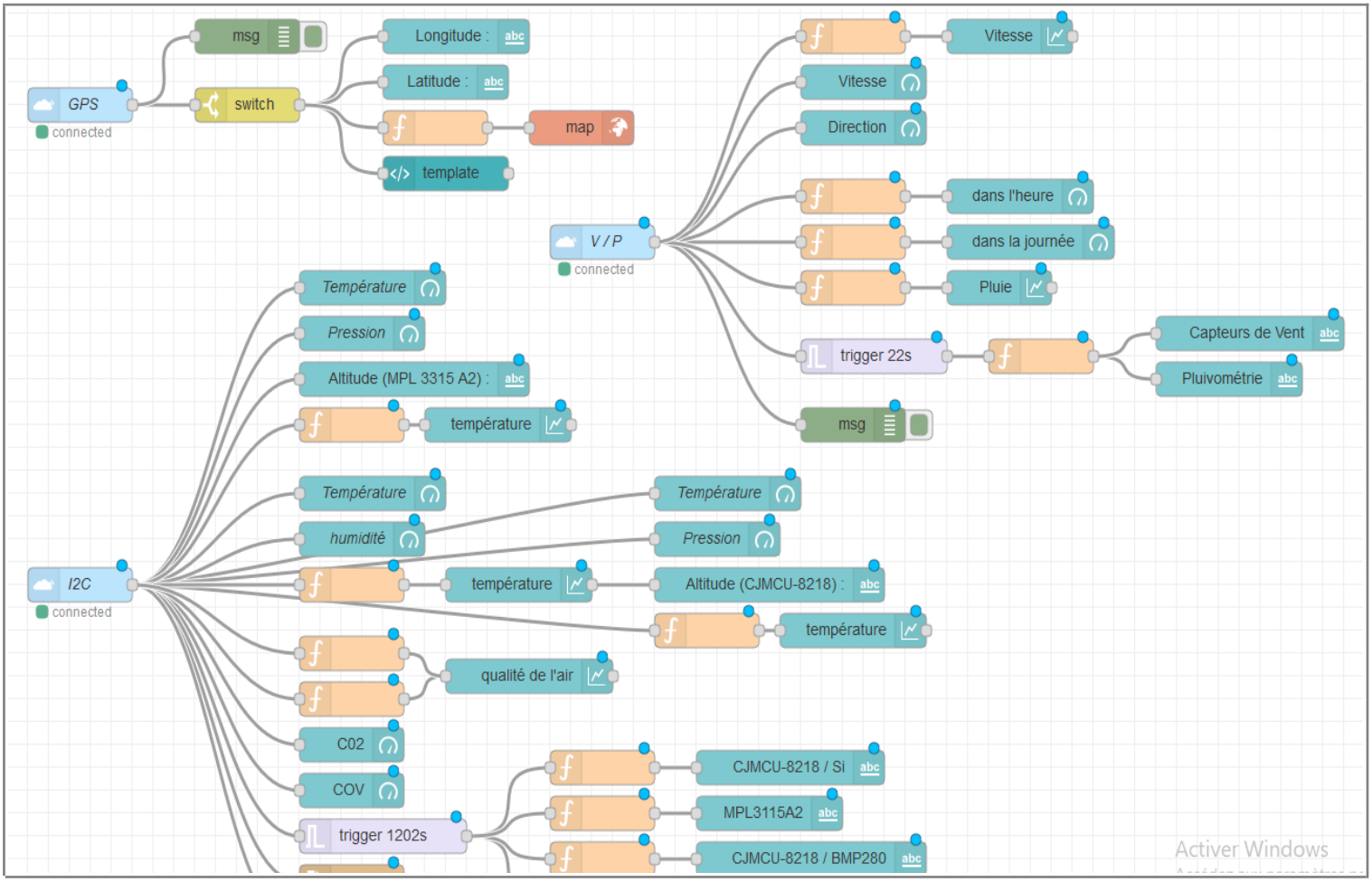
```
//gestion des évènements TTN
void onEvent (ev_t ev) {
    switch (ev) {
        case EV_JOINED:
            Serial.println(F("EV_JOINED"));
            LMIC_setLinkCheckMode(0);
        break;
        case EV_JOIN_FAILED:
            Serial.println(F("EV_JOIN_FAILED"));
        break;
        case EV_TXCOMPLETE:
            Serial.println(F("EV_TXCOMPLETE (includes waiting for RX windows)"));
            os_setTimedCallback(&envoi, os_getTime() + sec2osticks(TX_INTERVAL), do_send);
        break;
        default:
            Serial.println(F("Unknown event"));
        break;
    }
}

//Envoi des données
void do_send(osjob_t* j) {
    detachInterrupt(Pin_rain);
    detachInterrupt(Pin_anemo);
    if (LMIC.opmode & OP_TXRXPEND) {
        Serial.println(F("OP_TXRXPEND, not sending"));
    } else {
        if(timer == 0 ) {  // msg à l'allumage : toutes les données
            taille = 17;
            id = 0;
            datas.f = new float[taille];
            datas.bytes = new unsigned char[taille*4];
            get_coords();
            mesure_mpl();
            mesure_cjbmp();
            mesure_si();
            mesure_ccs();
            getVitesse();
            getDirection();
            datas.f[id] = off_direction;
            id++;
            datas.f[id] = pluie_heure;
            id++;
            datas.f[id] = pluie_jour;
            id++;
            datas.f[id] = total;
            timer+=20;
            LMIC_setTxData2(1, datas.bytes, 68 , 0);
```

```
      } else if (timer == 86400 ) {   //GPS : 1 x par jour ; remise du timer à zero
            taille = 2;
            id = 0;
            datas.f = new float[taille];
            datas.bytes = new unsigned char[taille*4];
            get_coords();
            timer = 0;
            LMIC_setTxData2(1, datas.bytes, 8 , 0);

      } else if(timer % 1200 == 0){    // capteurs (temp, hum,…) : toutes les 20 minutes
            taille = 10;
            id = 0;
            datas.f =new float[taille];
            datas.bytes = new unsigned char[taille*4];
            mesure_mpl();
            mesure_cjbmp();
            mesure_si();
            mesure_ccs();
            timer+=20;
            LMIC_setTxData2(1, datas.bytes, 40 , 0);

      } else {   // vent & pluie : toutes les 20 secondes
            taille = 5;
            id = 0;
            datas.f = new float[taille];
            datas.bytes = new unsigned char[taille*4];
            getVitesse();
            getDirection();
            datas.f[1] = off_direction;
            datas.f[2] = pluie_heure;
            datas.f[3] = pluie_jour;
            datas.f[4] = total;
            timer+=20;
            LMIC_setTxData2(1, datas.bytes, 20 , 0);
      }
  if( !mpl.begin() ){ Serial.println("MPL3115A2 pas Ok"); }
  if( !cjbmp.begin(0x76) ){ Serial.println("BMP du CJMCU pas Ok"); }
  if( !si.begin() ){ Serial.println("Si7021 pas Ok"); }
  if( !ccs.begin() ){ Serial.println("CCS811 pas Ok"); }
  attachInterrupt(Pin_rain,rain,FALLING);
  attachInterrupt(digitalPinToInterrupt(Pin_anemo), isr_rotation, FALLING);
}
```

```
//Paramétrage initial
void setup() {
        Serial.begin(115200);
        Serial.println("test des capteurs");
        GPSSerial.begin(9600, SERIAL_8N1, GPS_RX, GPS_TX);
        if( !mpl.begin() ){ Serial.println("MPL3115A2 pas Ok");  }
        if( !cjbmp.begin(0x76) ){ Serial.println("BMP du CJMCU pas Ok");  }
        if( !si.begin() ){Serial.println("Si7021 pas Ok");  }
        if( !ccs.begin() ){Serial.println("CCS811 pas Ok");  }

        as3935.initialise(21,22 , 0x03, 3, false, NULL);
        as3935.start();
        as3935.setNoiseFloor(0);
        attachInterrupt(21, foudre , RISING);

        pinMode(Pin_anemo, INPUT);
        attachInterrupt(digitalPinToInterrupt(Pin_anemo), isr_rotation, FALLING);

        pinMode(Pin_rain, INPUT_PULLUP);
        attachInterrupt(Pin_rain,rain,FALLING);

        SPI.begin(SCK,MISO,MOSI,SS);
        os_init();
        LMIC_reset();
        LMIC_setClockError(MAX_CLOCK_ERROR * 1/100);
        do_send(&envoi);
}


//boucle
void loop() {
  os_runloop_once();
}
```

IV - Le serveur Node-RED

- **Flow :**



Les capteurs reliés à la TTGO lui envoient leur relevés. La TTGO les transmet à The Things Network (TTN) via LoRaWAN. TTN décode le message et transmet le message décodé à Node-RED. Node-RED reçoit les messages dans la node correspondante selon le type de capteur, grâce à un filtrage par champs. (**nodes GPS , I2C et V/P - *ttn uplink***)

Chaque capteur renvoie plusieurs données. Chaque donnée est représentée dans une jauge, rangée par capteur. Les champs Longitude, Latitude et Altitude sont eux des champs textes. Chaque jauge ou champs prend juste en entrée un champs du message reçu. (**nodes de dashboard).**
Les données de températures et de qualité de l'air (CO2 , COV) sont également représentées sous forme des graphes linéaires, permettant de voir la variation de ces valeurs sur une durée donnée. Les nodes function permettent de mettre en forme le message d'entrée.

Code d'une function :

```
return {
        topic : "COV",
        payload : msg.payload.ccs_tvoc.toFixed(2)
};
```

Une **worldmap** est utilisée pour représenter la position indiquée par le gps. La node **function** prépare les données nécéssaires à l'ajout d'un point worldmap. La node **template** permet d'inclure la worldmap dans le dashboard. Elle est aussi accessible indépendamment avec un chemin du type [ip-serveur]/map.

Code de la function :

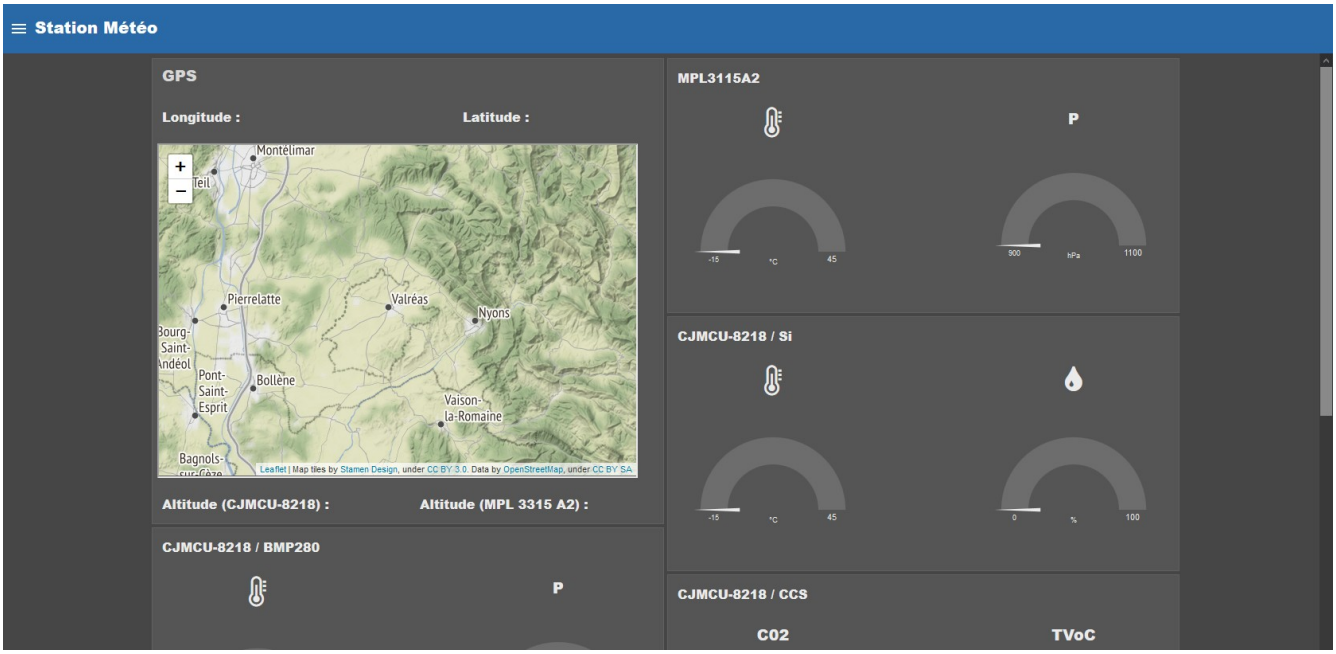
```
return {
        payload : {
                name:"station",      //nom du point
                lat:msg.lat,         //latitude
                lon:msg.lon,         //longitude
                icon:"fa-hand-o-down",   //icone utilisée – ici une main pointant en bas
                iconColor:"#55BBBB" // la couleur de l'icone – ici cyan
        }
}
```

Les nodes **trigger** permettent de gérer un témoin de fonctionnement des capteurs. Quand elles reçoivent un message, elles le transmettent puis lancent un timer selon le type de capteur (22 ou 1202 secondes), après lequel elle transmet un message différent. Les champs « textes » contiennent une icône qui change de couleur (rouge ou vert) selon le message reçu. Les nodes **function** permettent de gérer les conditions selon lesquelles l'icone doit être rouge ou verte.

Code d'une function :

```
var color = "green";
if(msg == "0" || (msg.payload.mpl_tmp == 0.0 && msg.payload.mpl_press == 0.0 )) {
        color = "red";
}

return {
        payload:{
                icon:"fa-square",      // icône 'font-awesome' de carré plein
                col:color
        }
};
```

dashboard :



Pour obtenir ce résultat , mettez le **theme** à **dark** (sombre) , et choisissez une taille de 6x4 pour chaque **gauge** , une taille de **13 x 8** pour chaque **graphe** , et une taille de **13x8** pour la **worldmap**. Chaque **group** a une largeur de **15**.