IoKey Technical Notice

Antoine Goulin

Angela Nana

Arnaud Fabre

February the 2nd

Abstract

The aim of this project is to design and implement a humidity and temperature measurement system specifically adapted to the windrow environment. Windrows, or piles of organic material such as hay, compost or agricultural waste, are subject to significant variations in humidity and temperature, which can have an impact on their quality, stability and combustible potential.

The proposed system uses state-of-the-art sensors to monitor conditions inside the windrows in real time. These sensors are designed to withstand harsh environmental conditions, such as high humidity, temperature fluctuations and the possible presence of corrosive substances

The data collected by the sensor is transmitted to a centralized processing unit , then visualized on a dashboard, where it is analyzed and presented in a user-friendly way. Advanced algorithms are used to detect trends, predict changes and warn of abnormal conditions, enabling proactive windrow management.

The system offers a number of benefits, including continuous, accurate monitoring of internal windrow conditions, helping to optimize management processes, reduce fire risks and improve end-product quality.

Contents

List of Figures

List of appendices

Introduction

This project aims at coming up with a solution to monitor temperature and moisture of a windrow (i.e. compost) in an industrial context.

What is a windrow? A windrow is a long industrial or agricultural pile of organic matter, such as hay, manure, or other biodegradable waste. The compost can be used as a fertiliser for agricultural purposes.

Figure 1: photo of windrows lined in the sun, rts.com

As shown on the above picture, a windrow is not simply left to dry in the sun, but also turned from time to time to encourage aeration and make the compost more porous. This fact will impact some of our design choices.

Customer needs

As previously mentioned, the goal of this project is to monitor the temperature and moisture of a mass of compost. The data must be sent over wireless (i.e. radio) connection and be available to the customer through an internet interface (or any other business intelligence platform). Moreover, a technician must access all data of a device through wireless connection whenever necessary. This implies several technical requirements, such as operating temperature and humidity ranges, measurement frequency, communication range, power consumption and lifetime, all of which are listed in the appendices (see Table 1: Functional Requirements).

Specific use case

The needs expressed above relate to the use case of a small business possessing 10 windrows, which are 100 metres long by 5 metres wide, and at most 5 metres apart from each other.

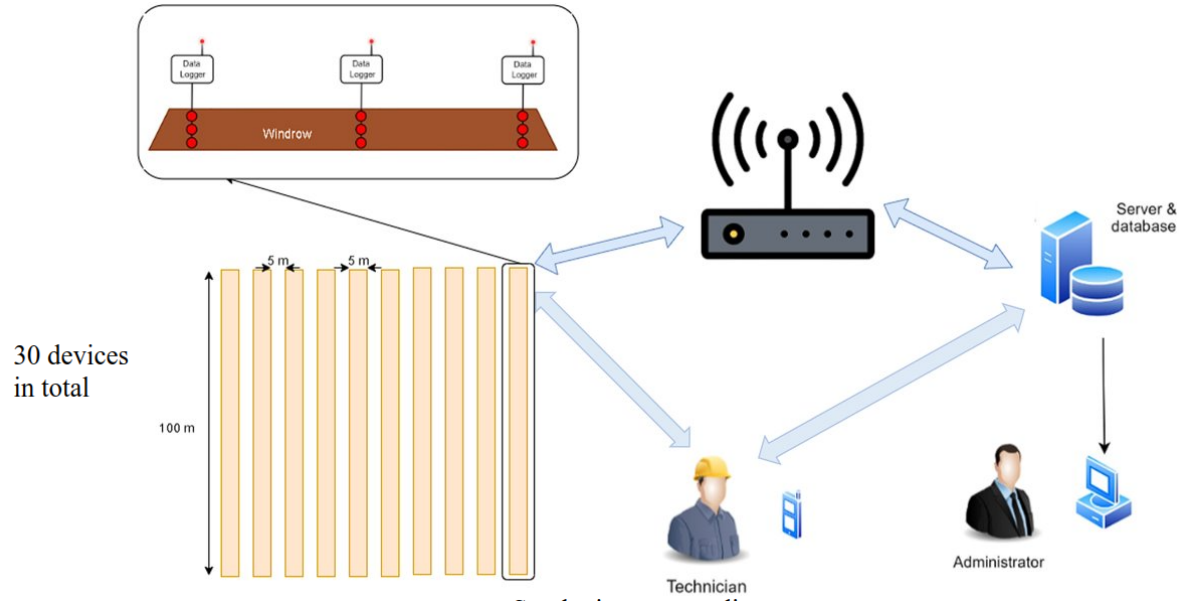


Figure 2: Use case representation

1 Functional description

1.1 Current prototype features

- Three-point temperature measurement:
  - 0 to +85 °C measurement range
  - ±1 °C accuracy
- Three-point moisture measurement:
  - 0 to 100 % measurement range
  - ±5 % accuracy
- Program cycle and power management
  - 1-hour wake-up period for measurements
  - button-press wake-up for BLE routine
  - deep sleep mode when inactive
- LoRaWAN data transmission:
  - 550 metres current maximum range
  - The Things Network data centralisation and parsing
- Bluetooth Low Energy communication
  - single-button-press wake-up to enable the device's BLE server and retrieve data on another device

2 Prototype architecture and wiring diagrams

2.1 Data concentrator architecture

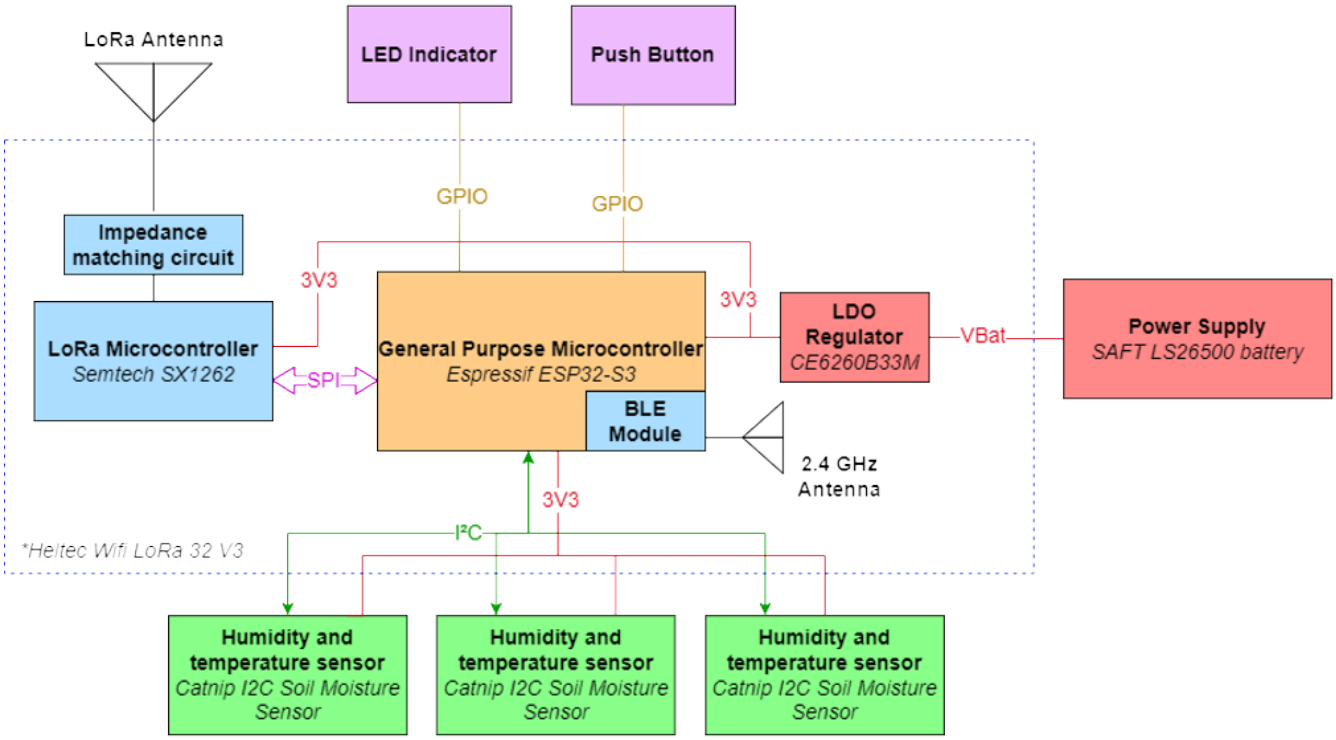


Figure 3: Data Concentrator architecture

LoRa module: This setup has an antenna designed for LoRa communication, and an impedance matching circuit to maintain the efficiency of the signal.

LoRa Microcontroller: A Semtech SX1262, is responsible for handling the LoRa communication, connected to the microcontroller through the SPI protocol.

main Microcontroller: An ESP32-S3 chip serves as the main device. It has a BLE (Bluetooth Low Energy) with it 2.4 GHz antenna.

Sensors: There are three identical sensors connected to the ESP32 via I2C. These sensors measure both humidity and temperature, and they are placed in the windraw.

LED and Button: There's an LED indicator and a push button connected to the ESP32 via GPIO pins. The button is likely used for connecting a smartphone in BLE.

Power Supply: The system is powered by a SAFT LS26500 battery. There's also an LDO (Low Dropout) regulator, which helps to provide a stable voltage to the microcontroller and sensors.

2.2 Data flux architecture

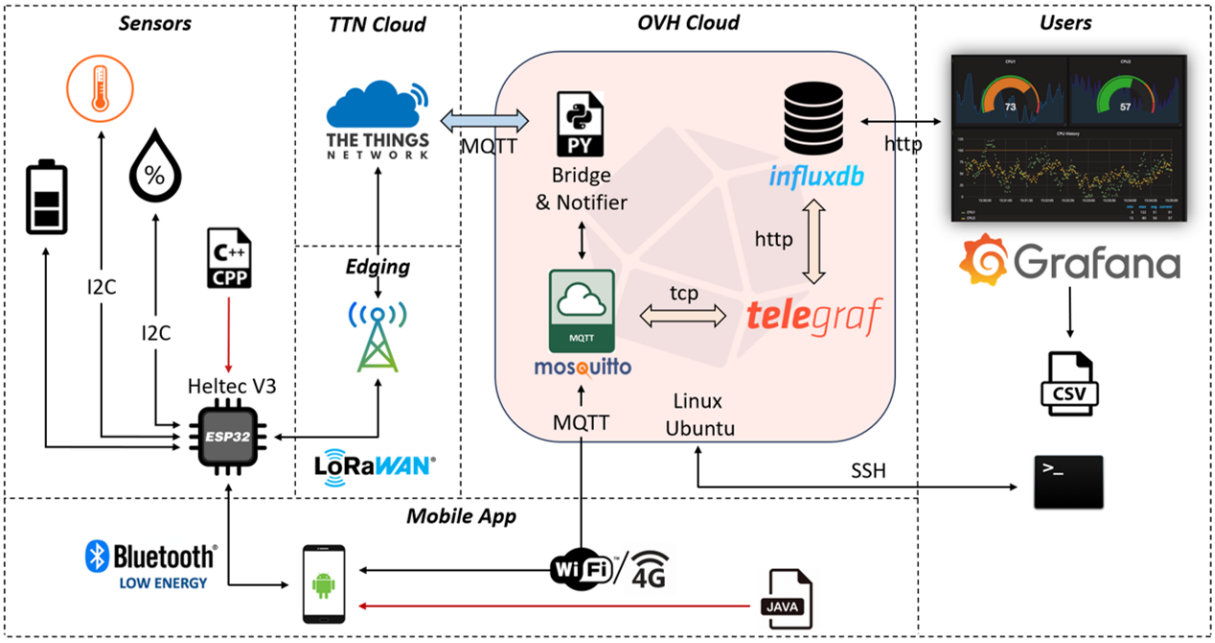


Figure 4: Flux architecture

Sensors: Various sensors are connected, they are interfaced with an ESP32 micro-controller via I2C protocol. These sensors include temperature and humidity.

Heltec V3 with ESP32: This is the micro-controller that collects data from the sensors and can communicate with other devices using LoRaWAN communication.

Mobile App (feature available in the industrial soltion): A mobile app can interact with the ESP32 via BLE, and can also send and receive data from the server.

TTN (The Things Network) Cloud: The data collected by the Heltec V3 microcontroller is sent to The Things Network, using MQTT protocol.

OVH Cloud: The data is then pushed to a server in the OVH Cloud, which is running a Linux Ubuntu operating system. Here, several processes occur:

A bridge and notifier script: It is establishing the connection, filtering data and handling notifications.re

Mosquitto: An MQTT broker receives the MQTT messages.

Telegraf: This is a plugin used for collecting and sending data to a database.

InfluxDB: A time-series database designed to handle high write and query loads.

Grafana: An interactive visualization web application where users can monitor the data in real-time.

2.3 Mechanical integration

The following mechanical representations were meant to give the user an idea of the final product's shape and mechanical features. It is not a proper finished model, nor a feature-complete implementation by any means.

The IoKey data concentrator is designed with easy integration into the windrow in mind, as well as ease-of-use. The spike at the end of its key shape allows the probe to get deep into the compost with as little resistance as possible. The tooth-like appendages circled in red boxes (see below picture) are where the senors are held, while the yellow box circled in green is the head of the device, which contains all electronics. The head is meant to stay out in the air to allow radio signals to air.

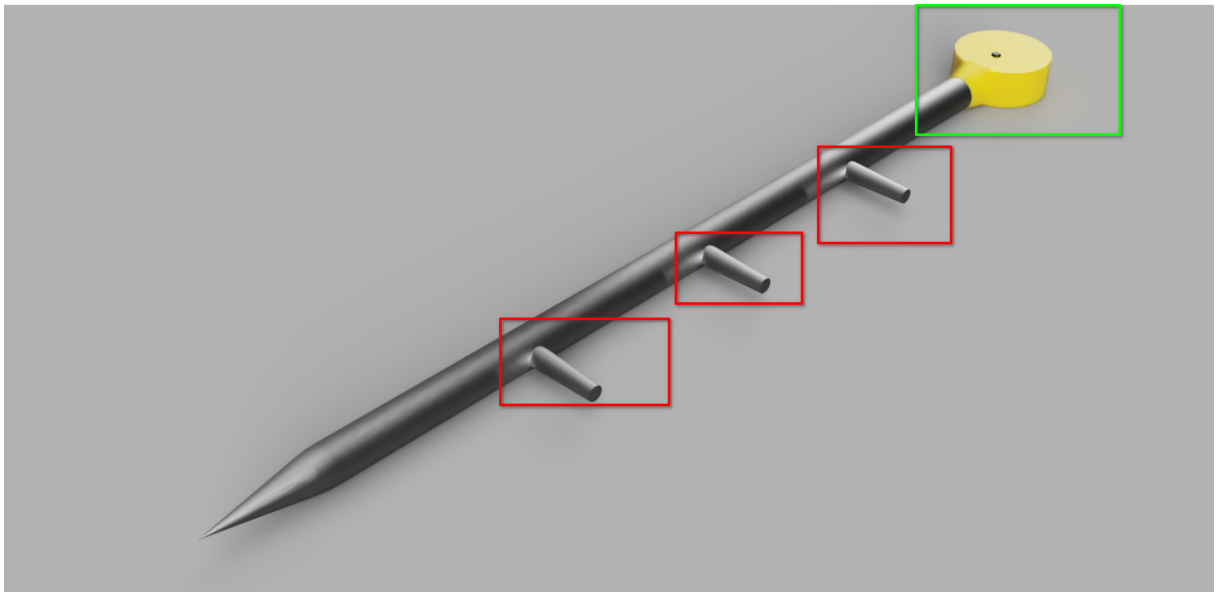


Figure 5: IoKey key-shaped probe graphical representation (non contractual)

The head part of the device features a wakeup button, which triggers a wakeup and BLE routine. An LED above the button is turned on while the BLE routine goes on.

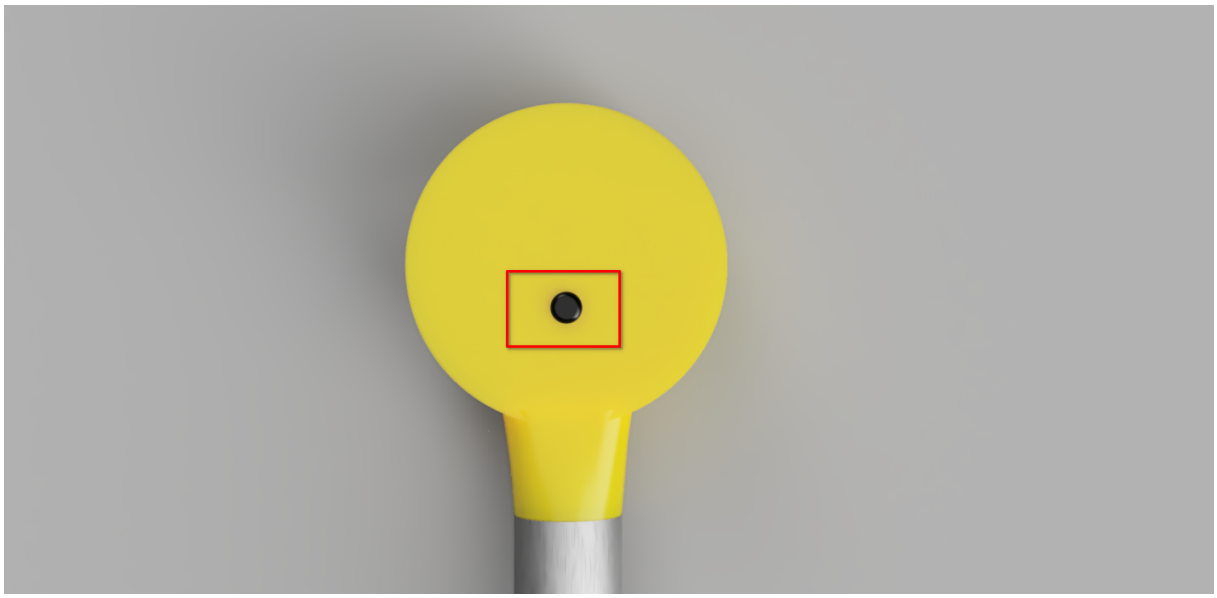


Figure 6: IoKey head portion graphical representation (non contractual)

2.4 Sensor wiring

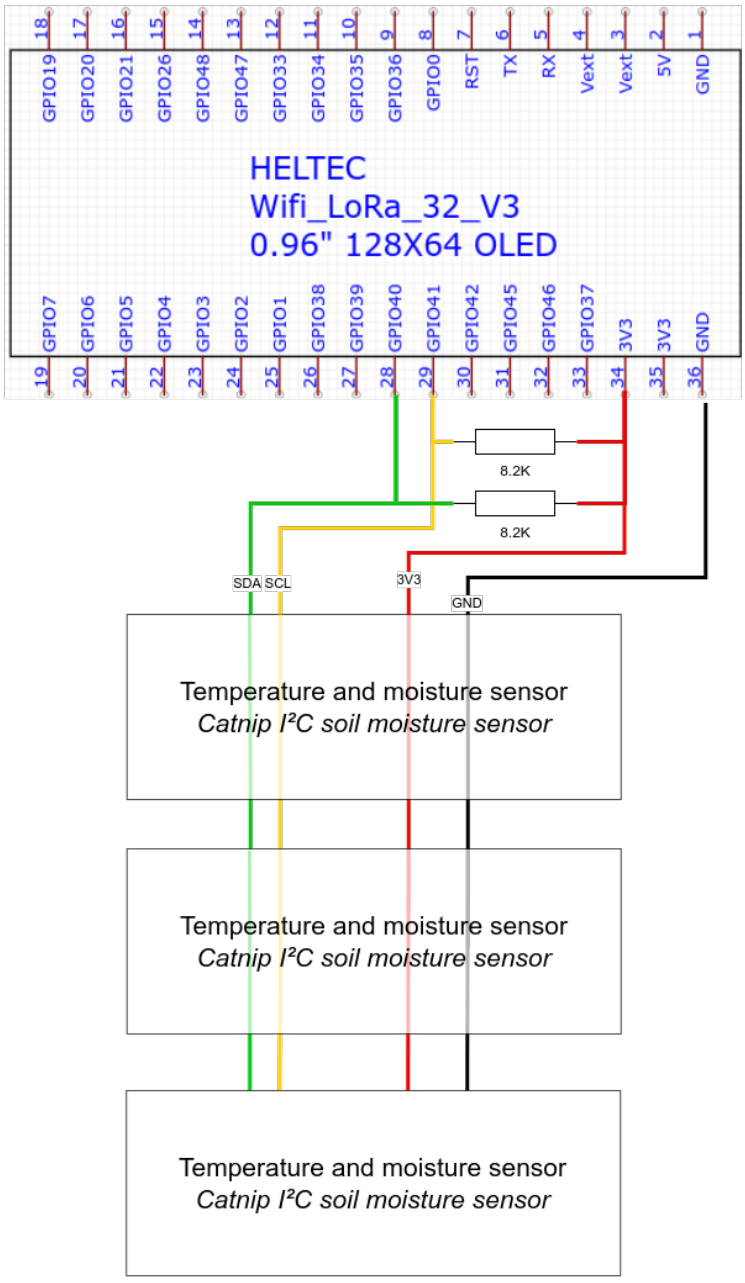


Figure 7: Catnip I²C soil moisture sensor wiring to Heltec WiFi LoRa 32 V3

3 Chosen components and services

This section is a list of all components of the IoKey device as well as the main reasons why they were chosen.

3.1 Micro-controller board

The first component on this list is a Heltec WiFi LoRa 32 V3 micro-controller board. It is made up of an ESP32-S3 micro-controller for general purpose actions, such as running the main program of the IoKey device, communicating over BLE and centralising the data sent by the three temperature and moisture sensors connected to it over I^2C. In addition, the board contains an SX1262 LoRa micro-controller and all relevant hardware adaptors, such as an impedence matching circuit, antenna connectors, etc. Finally the board is sold with a battery adapter and an antenna for LoRa transmission.

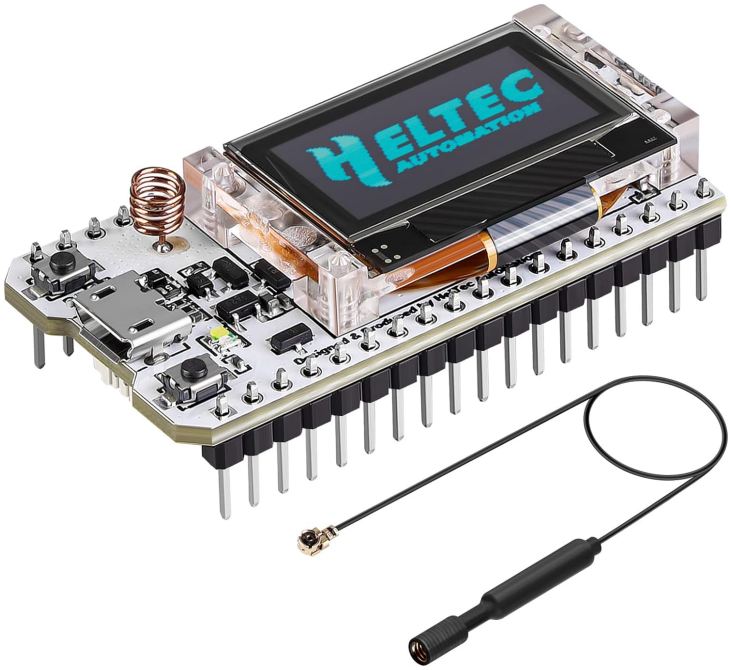


Figure 8: Heltec WiFi LoRa 32 V3

reddit.com

Main advantages:

- BLE and LoRa connectivity
- Integrated LoRa antenna
- Costs around $20
- Available with or without OLED display (useful for testing/debugging purposes)
- Deep sleep mode available for sub 10 µA current consumption
- Compatible with both 3.3V and 5V

Main drawbacks:

- Online pinout documentation erroneous or incomplete (fixed in libraries)
- Official library no longer maintained by Heltec (workarounds found)

3.2 Sensors

The sensor chosen for this prototype is the Catnip I²C soil moisture sensor. Given that the temperature range it covers is not enough for a real-life situation, this document covers how it has been programmed and used, but will also feature a list of sensors that better comply with the requirements. For more information on the decisions made during this project, please refer to the conclusion of this report.

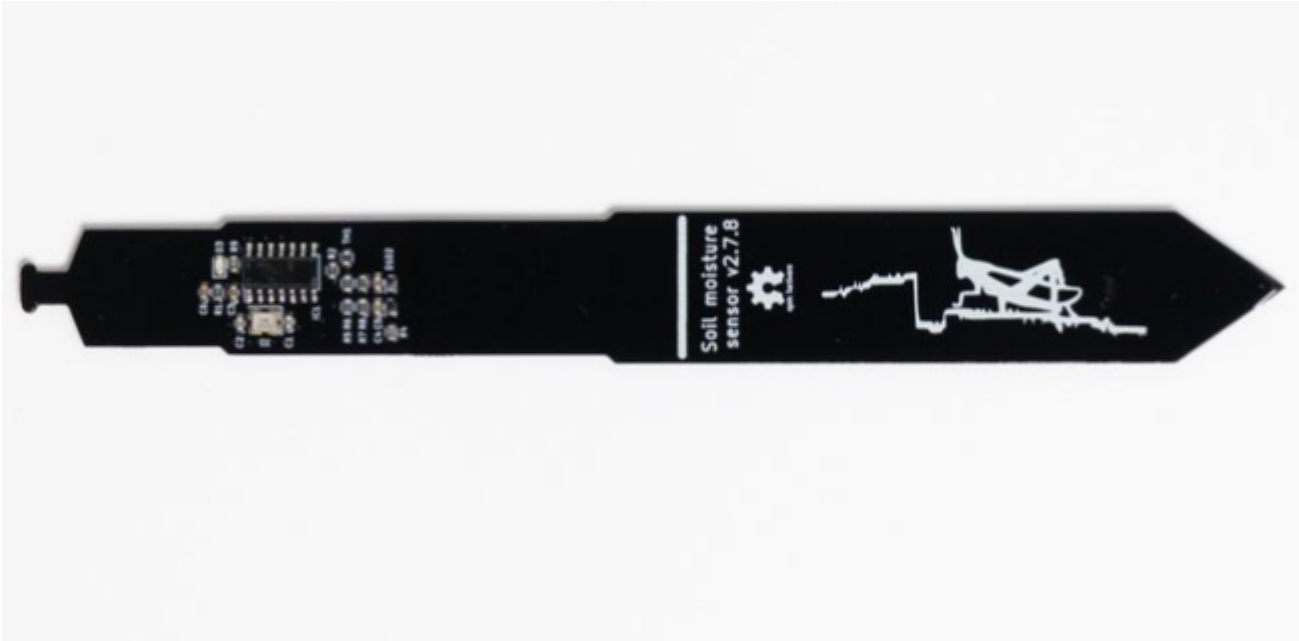


Figure 9: Catnip I²C soil moisture sensor

Features:

- Soil moisture sensing
- Temperature sensing
- I2C address change
- Deep sleep functionality
- Operating voltage range: 3.3V to 5V
- Operating temperature range: 0°C to 85°C
- Current consumption at 3.3V: 0.7mA when idle, 7.8mA when taking a measurement
- Unit price: approx $13

Drawbacks:

- Temperature range does not include negative temperatures.
- I²C is not meant for cables over one metre long, though workarounds exist, such as lowering the baud rate on the I²C interface, twisting and shielding all wires.

3.3 Power source

The operating temperature, moisture and up-time constraints require a power source that withstands harsh environmental conditions. A good power source that matches this project is to use a resistant battery meant for these special conditions, such as the SAFT LS-26500 single-use battery. Using a single-use battery might sound controversial, but using a LiPo or NiMH would not work given the environmental conditions, and potentially cause more environmental trouble than otherwise.
Main features:

- Voltage: 3.6V
- Capacity: 7.7 mAh
- Operating temperature range: -60 to +85 °C
- Unit price: between $20 and $30

Figure 10: SAFT LS-26500 battery

akkushop.de

3.4 LoRaWAN stack

The service chosen to handle the LoRaWAN data transfers is The Things Stack. The prototype uses a free subscription to The Things Network with access to only 10 devices, but the same company sells a professional service called the Things Industries. All paid industrial plans allow for a thousand devices and a pay-as-you-go system, an unlimited number of gateways, 99.9% uptime and a customer service.

Figure 11: The Things Network logo

homeandsmart.de

Main advantages:

- Stable
- Professional
- Well documented
- Formatting features included
- MQTT clients and webhooks available, among other forwarding solutions

Main drawbacks:

- Unclear options for free plans and many accounts potentially created
- Requires to have a gateway connected to TTN in range

3.5 LoRa gateway

In anticipation for scenarios where a client would be to far from a gateway to send their data over LoRaWAN, IoKey suggests adding a gateway to the offer. For the use case detailed in the Introduction of this document, the Heltec HT-M7603 could suffice. It is an indoor gateway, but its antenna can be wired outside so that the electronics stay safely inside a waterproofed hull.

Figure 12: Heltec HT-M7603 LoRaWAN gateway

heltec.org

Particular advantages:

- Easy to configure (web interface)
- MQTT available
- Compatible with LoRaWAN
- Affordable ($100)

3.6 Virtual Private Server

VPS provider

The VPS hosting IoKey's main data centralisation, database and visualisation tools is a virtual machine provided by OVH, a French VPS service.

Figure 13: OVH logo

i2coalition.com

- 2 virtual CPU cores (vCore)
- 2 gigabytes (GB) of RAM
- 40 GB of NVMe SSD storage
- Unlimited 500 Mbit/s bandwidth
- Operating System: Ubuntu 22

The price is changing frequently but is around $6 per month for such a machine. More enterprise-friendly plans exist with OVH, which makes scaling applications easier with a single provider. Furthermore, OVH has a hot-line for client services in case one encounters a problem with their machines.

Operating System

The operating system running on the VPS is Ubuntu 22. It has been chosen for its stability, popularity (i.e. good documentation and big community) and ease of use/configuration.

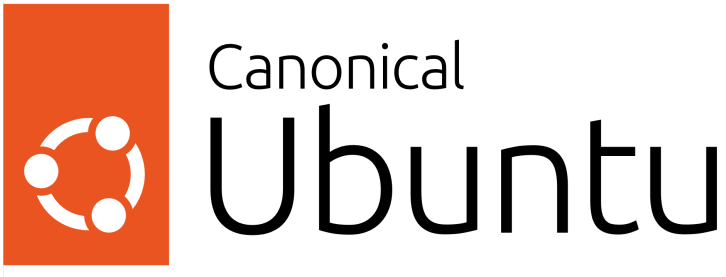


Figure 14: Ubuntu logo

assets.ubuntu.com

3.7 Data aggregation

Among all solutions that exist for data aggregation, Telegraf was one relevant software that fit our needs. It runs as a service on the VPS and gathers data from serveral different sources based on the plugins we install alongside its daemon. In this case, the LoRaWAN-transferred data is collected by a Python MQTT bridge (described in section 6), sent to an instance of the Mosquitto MQTT broker, and collected by Telegraf over TCP/IP.

Figure 15: Telegraf logo

il.wp.com

3.8 Database storage

The current database storage solution implemented into IoKey is managed by InfluxDB. The main reason for this choice is that it is natively compatible with Telegraf, and developped by the same company.

Figure 16: InfluxDB logo

logos-download.com

3.9 Data visualisation

Last, but not least, the data visualisation pages are built using Grafana

Figure 17: Grafana logo and visual

4 Data acquisition

The acquisition of the temperature and the humidity is handled by a library called `temperature_moisture_sensor.hpp` which provides a class called `TemperatureMoistureSensor` and includes the I2CSoilMoistureSensor library. The methods defined in the `TemperatureMoistureSensor` are sometimes really simple wrappings of the existing `I2CSoilMoistureSensor` methods. However, the original `I2CSoilMoistureSensor` class measures the sensor's capacitance instead of soil moisture. Therefore, the main purpose of the `TemperatureMoistureSensor` class is to have proper moisture and temperature values.

4.1 Initialisation

Class usage

To use the sensor, initialize it by creating a `TemperatureMoistureSensor` object with the address of the sensor on the I^2C bus as a parameter. In this example, a `SENSOR_ADDRESS` macro has been created for easy address remapping:

```
TemperatureMoistureSensor sensor(SENSOR_ADDRESS);
```

Each sensor must be initialised using the `begin()` function as follows:

```
top_sensor.begin();
middle_sensor.begin();

bottom_sensor.begin();
```

Constructor definition

The class contains a private pointer to an `I2CSoilMoistureSensor` object called `_sensor`. The address of the sensor on the I^2C bus must be passed to the constructor as a parameter:

```
TemperatureMoistureSensor::TemperatureMoistureSensor(uint8_t address) :
    _sensor(nullptr)
{
    _sensor = new I2CSoilMoistureSensor(address);
}
```

4.2 Measuring temperature

When retrieving the temperature value from the sensor, it must be stored in a variable of type `int8_t` to match the return of the function:

```
int16_t temperature = sensor.get_temperature();
```

Function definition

The function returns an `int8_t` with the value of temperature in °C. By default, the sensor is meant to measure the temperature multiplied by ten, which we account for before returning it.

```
int8_t TemperatureMoistureSensor::get_temperature()
{
    int temperature = _sensor->getTemperature();
    return (int8_t)(temperature / 10.0);
}
```

4.3 Measuring moisture content

When retrieving the moisture value from the sensor, it must be stored in an `unsigned int` variable of type as follows:

```
unsigned int moisture = sensor.get_moisture();
```

Function definition

The function returns an `unsigned int` with the value of the moisture in `%`. By default, the sensor is meant to measure the capacitance .

```
unsigned int TemperatureMoistureSensor::get_moisture()
{
    unsigned int capacitance = _sensor->getCapacitance();
    while (_sensor->isBusy())
        ;
    capacitance = _sensor->getCapacitance();
    return map(capacitance, MIN_CAPACITANCE_VALUE, MAX_CAPACITANCE_VALUE
   , 60, 100);
}
```

So, the map() function is used to convert the capacitance value read by the sensor from a scale to another (converting capacitance readings to moisture). In this case, the capacitance value measured by the sensor was about 250 at 60% RH (relative humidity) and 500 at 100% RH. The map function allows the code to return a value between 0 and 100% RH.

4.4 Putting a sensor to sleep

Putting the sensor in sleep mode can be achieved using the `sleep()` function, which wraps the `I2CSoilMoistureSensor::sleep()` function.

5 Putting the micro-controller to deep sleep

5.1 Introduction

During deep sleep, the CPU and Wi-Fi functions are inactive, but the Ultra Low Power (ULP) co-processor remains operational. Additionally, RTC memory remains powered on, enabling the main CPU to be awakened by external events or timers while minimizing power consumption.

5.2 Wake Up Sources

Deep sleep is enabled through the `sleep_functions.hpp` library. this library has a `go_to_sleep()` function.

```
void go_to_sleep()
{
    rtc_gpio_deinit(DEEP_SLEEP_WAKEUP_PIN);
    rtc_gpio_pulldown_en(DEEP_SLEEP_WAKEUP_PIN);
    esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP_SECONDS *
   MICROSECONDS_TO_SECONDS);
    esp_sleep_enable_ext0_wakeup(DEEP_SLEEP_WAKEUP_PIN, HIGH);
    esp_deep_sleep_start();
}
```

After putting the ESP32 into deep sleep mode, there are two ways to wake it up:

- By **timer**, waking up the ESP32 using a periods of time .

- By **external wake up**, for example a button.

Timer wake up

The ESP32 can go into deep sleep mode, and then wake up at predefined periods of time

Specify the sleep time by converting the factor for micro seconds to seconds:

```
MICROSECONDS_TO_SECONDS 1000000ULL;
```

`TIME_TO_SLEEP_SECONDS` define a constant that allows to assign a value in seconds.

```
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP_SECONDS *
   MICROSECONDS_TO_SECONDS);
```

External Wake Up (ext0)

This wake up source allows to use a pin to wake up the ESP32. The ext0 wake up source option uses RTC GPIOs to wake up.

Firstly, reinitialize the pin even if it is used.

```
rtc_gpio_deinit(DEEP_SLEEP_WAKEUP_PIN);
```

Then, put the GPIO into pulldown to avoid bad signals

```
rtc_gpio_pulldown_en(DEEP_SLEEP_WAKEUP_PIN);
```

This function takes the pin as first argument (represented by `DEEP_SLEEP_WAKEUP_PIN` in the above example). Note that one must only use pins that are RTC GPIOs. The second argument, can be either `HIGH` or `LOW`. This represents the state of the pin that will trigger wake up.

```
esp_sleep_enable_ext0_wakeup(DEEP_SLEEP_WAKEUP_PIN, HIGH)
```

6 Data transmission

This part of the document provides a short description of the data transmission protocols used with the data collector. For the regular measurement routine, the data measured by the IoKey device is sent over LoRaWAN to The Things Network (TTN) for decoding and forwarding. This protocol relies on the LoRa technology, chosen for its power efficiency regarding small-sized data transmission and long range. When a technician has to interact with the device in close range - for example in a scenario where the LoRaWAN communication does not work - the data is transferred to their phone over Bluetooth Low Energy.

6.1 LoRaWAN protocol

The LoRaWAN protocol is an encrypted and secure protocol that handles data formatting and forwarding over the LoRa frequency bands (868MHz in europe for ISM band).

6.1.1 The Things Network

The current version of IoKey relies on The Things Network LoRaWAN stack for data forwarding and formatting. Upon logging into the TTN console, the administrator is greeted with an interface prompting them to open applications of gateways.

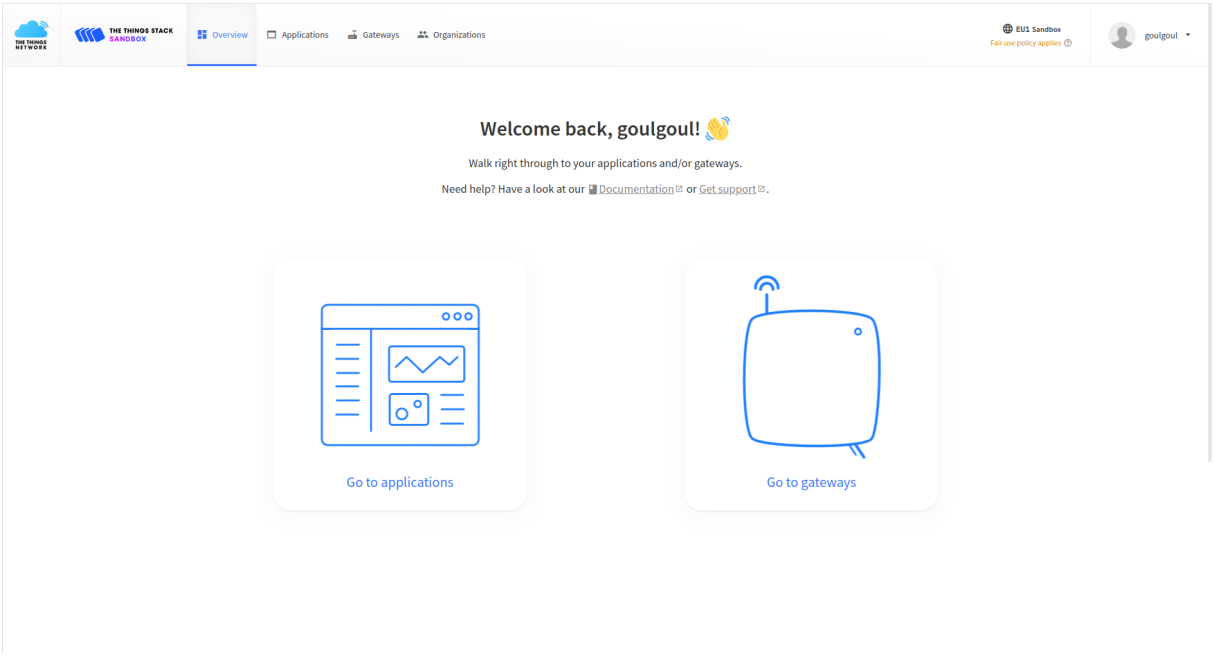


Figure 18: TTN console home

Clicking on the 'go to applications' section reveals a list of available applications. In the list, select 'iokey-application' to reveal a list of features. This is the menu that allows an administrator to see available devices, formatters and more. The interesting section here is 'Live data', for it is a console that allows users to see the frames and join requests sent by every device connected to said application.

6.1.2 Data formatting

After having been acquired, measurement data is serialised as a frame according to a specific format, detailed below. Note that all bytes of the frame must me unsigned integers and written on 8 bits, which implies further conversions on the server's side. Moreover, negative temperature values have to be converted to binary as a number greater than 127. Therefore, all readings received as bytes greater than 127 must be converted back to negative numbers on the Things Network payload formatter.

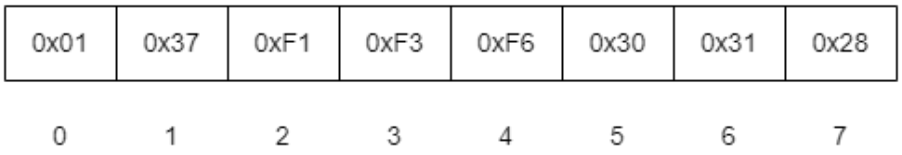


Figure 19: LoRaWAN data frame format
(above values are an example of real measurements given negative temperatures)

0. Device ID - this number is an ID set per device and per client, meant to change according to project scaling
1. Battery level - this value reflects the measured battery voltage on he ADC channel of the micro-controller board, multiplied by 10 to fit on an integer byte.
2. Topmost sensor temperature reading
3. Middle sensor temperature reading
4. Bottom sensor temperature reading
5. Topmost sensor moisture reading

6. Middle sensor moisture reading

7. Bottom sensor moisture reading

6.1.3 Javascript decoder

The following code block shows the javascript decoder used to sent frames into correct data readable from the Things Network MQTT broker.

```
function decodeUplink(input) {
    var data = input.bytes;

    if (data[2] > 127) {    temp1 = data[2] - 128   }
    if (data[3] > 127) {    temp2 = data[3] - 128   }
    if (data[4] > 127) {    temp3 = data[4] - 128   }

    var json = {
        datalogger: data[0],
        battery_level: data[1],
        temperature: {
            temperature_s1: temp1,
            temperature_s2: temp2,
            temperature_s3: temp3
        },
        humidity: {
            humidity_s1: data[5],
            humidity_s2: data[6],
            humidity_s3: data[7]
        }
    }

    return {
        data: json,
        warnings: [],
        errors: []
    };
}
```

6.2 Programming interfaces

6.2.1 LoRaWAN interface

The programming interface for LoRaWAN protocol functionalities is provided by the `LoRaWANManager` class, in the `lorawan_manager.hpp`. Here is a list of its methods, and how to use them.

Initialising LoRaWAN

The only instance of `LoRaWANManager` available is created at runtime. To use it, define a pointer to a `LoRaWANManager*` object and call the `LoRaWANManager::get_instance()` method to store the relevant reference:

```
LoRaWANManager *lorawan = LoRaWANManager::get_instance();
```

The LoRaWAN interface must be initialised using the `begin()` function, which takes three parameters: a device EUI, an application EUI and an application key. These parameters must be declared elsewhere as arrays of respectively 8, 8 and 16 bytes (unsigned char):

```
lorawan->begin(lorawan_dev_eui, lorawan_app_eui, lorawan_app_key);
```

Printing EUIs for debugging

For debugging purposes, the `print_parameters()` function is available. It prints the aforementioned parameters to the serial monitor when plugged in.

```
lorawan->print_parameters();
```

Joining a LoRaWAN network

Joining LoRaWAN network can be done by using the `join()` function provided that given credentials are correct.
This function returns an error code defined by the `lorawan_manager_error_t` enumeration (typedef), which means it can be compared to an error code for debugging applications.

```
if (lorawan->join() == JOINING_TIMEOUT)
{
    Serial.println("\tCould not join TTN, please try again later or
    closer to a gateway.");
    return;
}
```

Sending a data packet

The exact same conditions apply to the `send_data` function. The latter requires data array parameter, of type `uint8_t[]` and whose size must be set using `LORAWAN_FRAME_LENGTH` macro in the `global_config.h` file.

```
if (lorawan->send_data((const uint8_t *)data_to_send,
    LORAWAN_FRAME_LENGTH) == FRAME_NOT_SENT)
{
    Serial.println("\tCould not send data to TTN, please try again later
    .");
    return;
}
```

6.3 BLE server

So far, the BLE functionalities of the IoKey device are limited to transmission from the data concentrator to a BLE compatible device. No configuration of IoKey data concentrators can be done from a BLE device (such as a smartphone).

6.3.1 Services list

The data sent over BLE is organised into two services: one for measurement data and one for device status information, such as the battery voltage and potential other information in the future. All characteristics of the following services are set to read-only mode.

Device information

Currently, this service contains one characteristic for the battery voltage and a descriptor for the volt unit. So far, however, the current version of the code is not displaying this information correctly. There could be an issue with data length or descriptor compatibility, but no further modifications were tried on the code. Please note that the battery voltage value, so far, is not measured for real, but simulated.

Measurement data

This service contains six characteristics and six relevant descriptors. The first three characteristics are reserved for temperatures displayed in degrees Celsius, from the topmost sensor to the bottom-most one. The remaining characteristics display the soil moisture as humidity in percents.

6.3.2 Programming interface

All actions related to the BLE features are handled by the `BLEManager` class, provided by the `ble_manager.hpp` file.

Initialisation

To initialise the class, one must first get a handle of its instance. To do so, assign the return of the `BLEManager::get_instance()` method to a pointer of type `BLEManager*`.

Starting the server

To start the server, call the `begin()` method on the pointer created as a handle. The required parameters of this function are a server name, that can be chosen at will, and the data to display. The server name string must be in the `const char*` format. The default program provided as an example of IoKey application has a `BLE_SERVER_NAME` macro that is used for easy configuration. The data to send is a bytes array (`const uint8_t*`) following the same convention as the LoRaWAN frame. Only the device ID will not be sent over BLE, which can be changed my editing the `ble_manager.hpp` library. Should one edit the library code, they should note that all values transmitted over BLE have to be multiplied by 100 to be rendered correctly (at least for temperature and humidity).

Fetching the current connection status

The `BLEManager` class contains a `is_connected()` method that returns the current status of the device as a boolean. It can be useful to check this status periodically to determine when to move on to the next task in the code. The only place in the code where the value of `__ble_connected` must be changed is inside the two `onConnect()` and `onDisconnect()` callback functions, called as the user connects and disconnects from the BLE server.

Stopping the BLE server

In the example code, stopping the BLE server is done automatically as the user disconnects, given the micro-controller is forced to go into deep sleep mode. However, the initial purpose of the `onDisconnect()` method was to stop the BLE server automatically, as the user disconnected, but internally to the `BLEManager` class. This is still to be done by editing the library.

7 Server

Sections

- Instance
- Security
- Services installation
- Services configuration
- Data visualization
- Maintenance and monitoring

7.1 Security

To enhance the security of your system, it's recommended to update your package lists and upgrade the packages by running the following commands:

```
sudo apt update
sudo apt upgrade
```

7.1.1 Changing SSH Port

To configure SSH to use a non-standard port, edit the SSH configuration file and replace the default port number with one between '49152' and '65535'. Open the SSH configuration file in a text editor, such as nano:

```
sudo nano /etc/ssh/sshd_config
```

Look for the following lines or equivalents:

```
# What ports, IPs, and protocols we listen for
Port 22
```

Replace the number '22' with a port number between '49152' and '65535' that is not already use by the system. You can check its availability using the following command in another terminal:

```
sudo lsof -i :[your_port]
```

Save and exit the configuration file. If the ‘Port‘ line is commented out with a ‘#‘ or other symbol and restart the SSH service to apply the changes:

```
sudo systemctl restart sshd
```

To verify, do not close your session, just try a new ssh connection to your server.

```
ssh username@your_VPS_IPv4 -p [new port]
```

7.1.2 Adding a New User

To add a new user to your system, use the following command:

```
sudo adduser [user_name]
```

Follow the prompts to set the password, name, and other information for the new user. This user will be able to connect via SSH using the specified credentials. Once logged in as the new user, you can switch to the root user for operations requiring root permissions by typing:

```
su root
```

Enter the password when prompted, and the active session will switch to the root user.

7.1.3 Disabling root server access

The root user, which has the highest level of access to an operating system, is created by default on GNU/Linux systems. It’s advised against and even dangerous to leave your VPS accessible only as root, as this account can perform irreversibly damaging operations.

It’s recommended to disable direct root user access via SSH. Ensure you have created another user before proceeding with the steps below. Edit the SSH configuration file as previously described:

```
sudo nano /etc/ssh/sshd_config
```

Find the following section:

```
# Authentication:
LoginGraceTime 120
PermitRootLogin yes
StrictModes yes
```

Change ‘yes‘ to ‘no‘ on the ‘PermitRootLogin‘ line. To apply this change, you must restart the SSH service:

```
sudo systemctl restart sshd
```

Following this, any connections to your server via the root user will be disabled.

7.1.4 Installing Fail2ban

Fail2ban is an intrusion prevention framework that aims to block IP addresses from which bots or attackers attempt to penetrate your system. This package is recommended, or even essential in some cases, to protect your server from Brute Force or Denial of Service attacks. To install Fail2ban, use the following command:

```
sudo apt install fail2ban
```

7.1.5 Configuring Fail2ban

You can customize Fail2ban's configuration files to protect services exposed to the public Internet from repeated login attempts. As recommended by Fail2ban, create a local configuration file for your services by copying the "jail" file:

```
sudo cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
```

Then, open the file with a text editor:

```
sudo nano /etc/fail2ban/jail.local
```

Pay attention to the information at the top of the file, especially the comments under '[DEFAULT]'. The '[DEFAULT]' settings are global and will apply to all services set to be enabled in this file. It's important to note that global settings will only be considered if there are no different values defined in the service sections (JAILS) further down in the file. For example, these lines under '[DEFAULT]':

```
bantime = 10m
maxretry = 5
enabled = false
```

This means that an IP address from which a host attempts to connect will be blocked for ten minutes after the fifth unsuccessful login attempt. Furthermore, all settings specified by '[DEFAULT]' and in the following sections remain disabled unless the line 'enabled = true' is added for a service (listed below '#JAILS'). For example, having the following lines in the '[sshd]' section will enable restrictions only for the OpenSSH service:

```
[sshd]
enabled = true
port = ssh
filter = sshd
maxretry = 3
findtime = 5m
bantime = 30m
```

In this example, if an SSH login attempt fails three times within five minutes, the IP ban period will be 30 minutes. You can replace "ssh" with the actual port number if you have changed it. The best approach is to enable Fail2ban only for services that are actually running on the server. Any custom setting added under '#JAILS' will then take precedence over the default values. After completing your modifications, save the file and close the editor. Restart the service to ensure it runs with the applied customizations:

```
sudo service fail2ban restart
```

7.1.6 Uncomplicated Firewall

To secure your system by closing non-essential and unused incoming ports, you can use UFW (Uncomplicated Firewall) with a strict security approach. Here's a summarized guide: 1. Set the Default Incoming Policy to Deny: This step makes UFW block all incoming traffic by default, effectively closing all ports unless explicitly opened by a specific rule.

```
sudo ufw default deny incoming
```

2. Allow Traffic on Essential Ports: After setting the default policy to deny all incoming connections, open only the ports needed for your services. For common services like SSH (port 22), HTTP (port 80), and HTTPS (port 443), allow these ports:

```
sudo ufw allow ssh
sudo ufw allow http
sudo ufw allow https
```

For services on non-standard ports, specify the port number, like so for port 1883 (defaultport for mosquitto) and 3000 (default port for grafana) and 8086 (default port for influxdb):

```
sudo ufw allow 1883
sudo ufw allow 3000
```

```
sudo ufw allow 8086
```

3. Verify Your Rules: You can check that your rules are set up correctly. Use the following command to display the current rules:

```
sudo ufw status verbose
```

4. Enable UFW: If UFW is not already active, enable it to apply your configuration:

```
sudo ufw enable
```

5. Testing: With UFW activated and new rules in place, test your setup from an external network to ensure that services are accessible on allowed ports and that other ports remain inaccessible.

7.2 Services Installation

7.2.1 InfluxDB

Purpose: InfluxDB is a time series database designed to handle high write and query loads. It is an ideal datastore for time-series data generated from sensors, IoT devices, metrics, applications, and other sources. InfluxDB is optimized for fast, high-availability storage and retrieval of time series data in fields such as operations monitoring, application metrics, IoT sensor data, and real-time analytics.

Installation: Add the InfluxDB repository to your system:

```
wget -qO- https://repos.influxdata.com/influxdb.key | sudo apt-key add -
echo "deb https://repos.influxdata.com/ubuntu $(lsb_release -cs) stable"
    | sudo tee /etc/apt/sources.list.d/influxdb.list
```

Update your package list and install InfluxDB:

```
sudo apt-get update && sudo apt-get install influxdb
```

Start and enable InfluxDB to run on system boot:

```
sudo systemctl start influxdb
sudo systemctl enable influxdb
```

7.2.2 Telegraf

Purpose: Telegraf is a plugin-driven server agent for collecting and sending metrics and events from databases, systems, and IoT sensors. Telegraf is designed to minimize memory

usage and support plugins to collect data from a variety of sources, which it can then write into a variety of destinations, including InfluxDB.

Installation:

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

Start the Telegraf service:

```
sudo systemctl start telegraf
```

Enable Telegraf to start on boot:

```
sudo systemctl enable telegraf
```

7.2.3 Grafana

Purpose: Grafana is an open-source platform for monitoring and observability. It allows you to query, visualize, alert on, and understand your metrics no matter where they are stored. You can create, explore, and share dashboards with your team and foster a data-driven culture. It's most commonly used for time series analytics but also supports other data visualizations.

Installation: Add the Grafana APT repository:

```
sudo apt-get install -y software-properties-common
sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable
    main"
wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -
```

Update your package list and install Grafana:

```
sudo apt-get update && sudo apt-get install Grafana
```

Start and enable Grafana:

```
sudo systemctl start grafana-server
sudo systemctl enable grafana-server
```

7.2.4 Mosquitto

Purpose: Mosquitto is an open-source message broker that implements the MQTT protocol. MQTT is a lightweight, publish-subscribe network protocol that transports messages between devices. Mosquitto is suitable for use in all situations from connected home applications to large-scale messaging in enterprise architectures.

Installation:

```
sudo apt-get update && sudo apt-get install mosquitto mosquitto-clients
```

Start the Mosquitto service:

```
sudo systemctl start mosquito
```

Enable Mosquitto to start on boot:

```
sudo systemctl enable mosquito
```

7.3 Services Configuration

Configuration files for InfluxDB, Telegraf, and Grafana are located in their respective '/etc' directories.

7.3.1 Influxdb

Influxdb is configurable in the /etc/influxdb/influxdb.conf file

```
 GNU nano 7.2                                    /etc/influxdb/influxdb.conf

[http]
  # Determines whether HTTP endpoint is enabled.
  enabled = true

  # The bind address used by the HTTP service.
  bind-address = ":8086"

  # Determines whether user authentication is enabled over HTTP/HTTPS.
  # auth-enabled = false

  # The default realm sent back when issuing a basic auth challenge.
  # realm = "InfluxDB"

  # Determines whether HTTP request logging is enabled.
  log-enabled = true

  # Determines whether the HTTP write request logs should be suppressed when the log is enabled.
  # suppress-write-log = false
```

Figure 20: Influxdb config file

7.3.2 Telegraf

Telegraf is configurable in the /etc/telegraf/telegraf.conf file Influxdb is configurable in the /etc/influxdb/influxdb.conf file

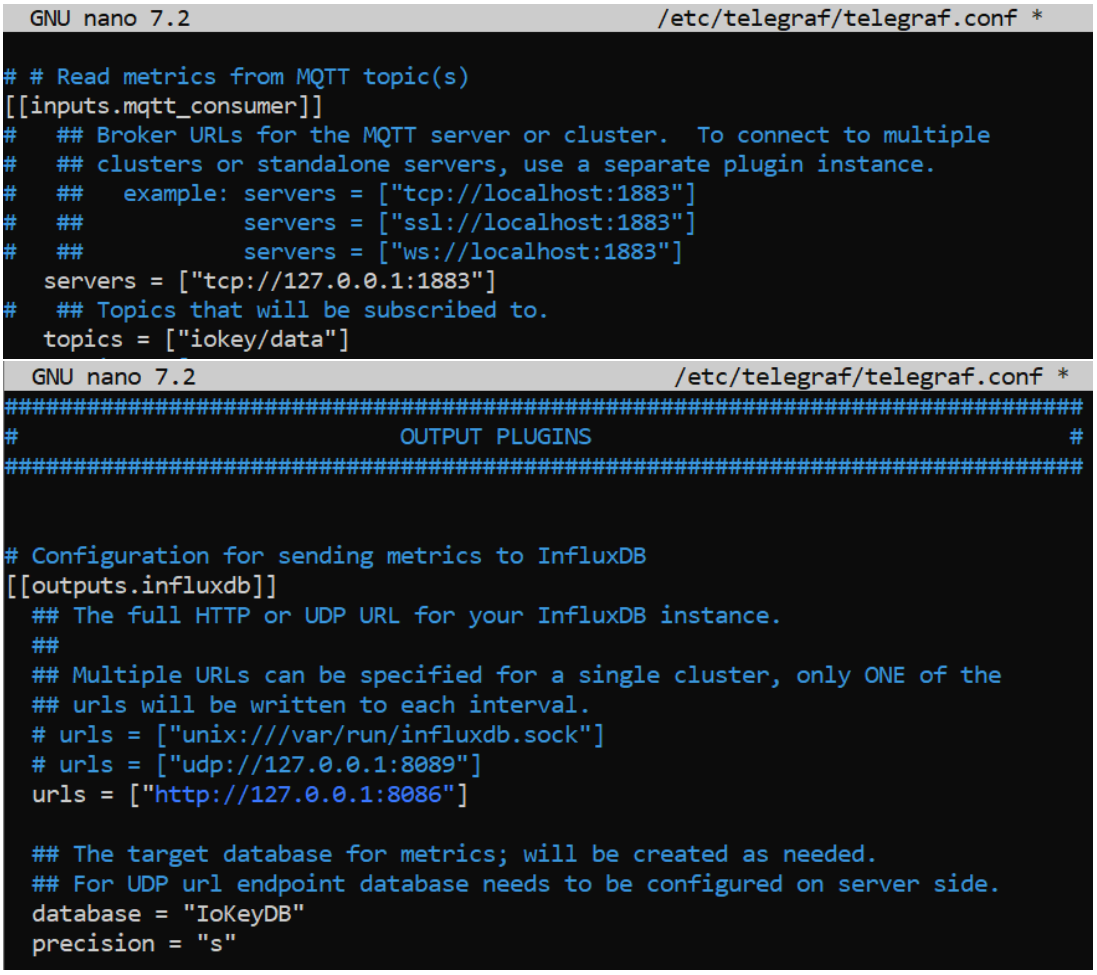


Figure 21: Telegraf config file (Input first, output last)

7.3.3 Grafana

The configuration of Grafana can be done from the graphical interface that the plugin offers. The only command added is the permission for anonymous viewing.

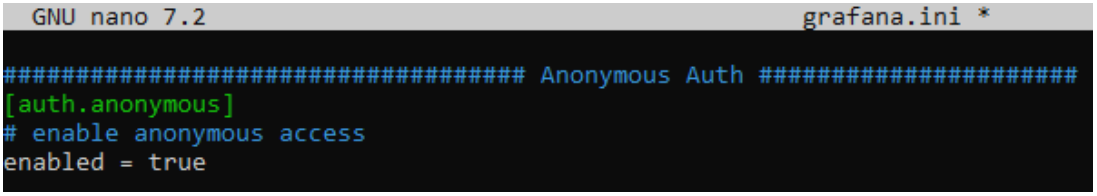


Figure 22: Grafana config file

7.4 MQTT Bridge (Optional)

The MQTT bridge allow the MQTT connection to be established, in our case MQTT broker from TTN to the VPS. This python script is not only used to establish a connection but also to filter the payload by removing additional information added by TTN. If the payload has the right format, the data will be then stored into influxdb. This python script needs to be run in a virtual environment due to the requirement of the paho-mqtt lib. Furthermore, logs are available with this the bridge, allowing you to debug and track the data. The bridge is automatically launched at the reboot of the server via a ksh script link to the crontab. (cf. python script)

7.5 Visualization & Dashboard

Grafana dashboard can be created by first logging in to the Grafana web interface. Click on the the left sidebar and select "Dashboard" to create a new dashboard. Then, click "Add new panel" to add your first panel to the dashboard. In the panel editor, select the data source you've configured (in our case IoKeyDB) and construct your query to display the data. Here is the official documentation of Grafana to create dashboards: Create a dashboard — Grafana documentation

7.6 Maintenance and Operability

Here is some basic but essential commands to manage the services mosquitto, telegraf, influxdb and grafana: status, and restart:

```
systemctl status [service]
```

```
systemctl restart [service]
```

Logs consultaion via journalctl:

```
journalctl -u [service] -n [number_of_lines]
```

8 Testing and Validation

To assess the performance and responsiveness of the windraw's monitoring system, an arbitrary test was conducted using a sensor of the probe. The objective of this test is to evaluate how accurately and quickly the sensor can detect changes.

8.1 End-to-end testing

This test is arbitrary conducted for the humidity of a sensor probe.

The following frame shows measurement data taken by the sensors. The red square shows all three moisture measurements, with the first one coming from a sensor wrapped in a wet piece of tissue to simulate soil moisture.

```
WAKE_UP (0)

DATA_SERIALIZATION (1)
  Data from measurements:
  (id Vb t1 t2 t3 m1 m2 m3)
    1 48 24 24 23 98 60 61 ^M

DATA_UPLOAD (2)
  Device EUI: 2232330088882
  Application EUI: 45434F4C45494F45
  Application key: CE2376272482788C
  Attempting to join TTN...^M
  Attempting to send data...^M
  Successfully sent data, going to deep sleep.^M

DEEP_SLEEP (4)
```

Figure 23: Data concentrator logs

```
2024-01-27 13:20:33,425 - INFO - Connection successfully established between Node: eu1.cloud.thethings.network:1883 and Node: 127.0.0.1:1883
2024-01-27 13:20:33,425 - INFO - << MQTT bridge operational >>
```

Figure 24: Bridge logs - Connection

To make sure that the connection between The Things Network (TTN) and the Virtual Private Server (VPS) works well, the first need is to check the bridge and it connection.

```
2024-01-27 16:38:26,772 - INFO - Message broadcasted: {"datalogger": 1, "battery_level": 48, "temperature": {"temperature_s1": 24,
"temperature_s2": 24, "temperature_s3": 23}, "humidity": {"humidity_s1": 98, "humidity_s2": 60, "humidity_s3": 61}}
```

Figure 25: Bridge logs - Payload

The sensor data, or payload, gets into the database in the right format and with the correct values needed for proper processing.

If the VPS services are well-configured, the grafana dashboard updates, allowing the client to focus on different metrics, including humidity levels in this test.

Figure 26: Humidity monitoring

As expected with the previous measurements, humidity peaks at 98%. After that, the humidity sensor is dried, leading to a significant drop in the humidity level measured. This drop is about one-third from the initial peak, showing the system's ability to detect and respond to changes in humidity quickly and accurately.

Conclusion

Achievements

in this project, certain functions have been implemented such as :

- Successfully implemented temperature and humidity data acquisition in deep-sleep mode to optimize power consumption.
- Established seamless LoRaWAN communication for remote data transmission.
- Integrated with The Things Network (TTN) for efficient data management and routing.
- Completed installation and configuration of Mosquitto, Telegraf, InfluxDB, and Grafana for comprehensive data storage and analysis.
- Visualized data effectively on Grafana for insightful analysis and monitoring.

However, there are still other functions to be carried out for this project to be completely finished, and these functions are as follows ::

- Development of an Android application to interface with the sensor and provide user-friendly interaction.
- Implementation of notification system to alert users of important events.
- Fine-tuning of sensor calibration to ensure accurate data readings.

Perspectives

In view of everything that this project involves, the prospects envisaged are as follows:

Identifying Interrupt-capable Sensors:

- .Identification and selection of interrupt-capable sensors to streamline data acquisition when windrows begin to heat up, even while asleep and enhance efficiency.

Development of Mobile App for BLE and Calibration:

- Creation of a mobile application compatible with Bluetooth Low Energy (BLE) technology to enable seamless communication with the microcontroller

- Integration of calibration features within the mobile app to facilitate sensor calibration for accurate readings.

Implementation of Smartphone Alerts in the App:

- Integration of real-time alerting functionalities within the mobile app to notify users of important events or critical sensor readings.

Appendices

Personal Reviews

Angela

Throughout this project, I was confronted with a number of challenges that I hadn't encountered before, and also had to deal with the deadline and all that goes with it. This project allowed me to deepen my knowledge, given that we were supervised by professors and also engineers from the Altyor group, who were true guides throughout the project, and also through the numerous research projects carried out to perfect this project. It also enabled me to improve my skills. I enjoyed working as part of a team, because we helped each other out. As the saying goes, one can get to the destination , but two can get to the destination faster.

Arnaud

Through my involvement in this IoT project, I had the opportunity to apply and expand my skills in server administration, scripting, and database management, with a particular focus on real-time data visualization. This experience allowed me to strengthen my abilities in needs analysis and solution optimization, thanks to a rigorous approach and the selection of right technologies. I also appreciated receiving feedback, help, and valuable advice of the engineers from Altyor, which enhanced my understanding and approach to the project. This project strengthened my ability to work in synergy with my team, strategically dividing tasks to achieve our common goals. The combination of these elements was crucial for the project's success and significantly enriched my technical skills and my ability to collaborate effectively.

Antoine

Although this project has taught me a lot about radio communications, cloud services and other technical aspects of IoT, I think that the richest aspect of it was the industrial part that taught me the most. Working with the support of Altyor gave us a variety of invaluable points of view on economical decisions, production environment mindsets, as well as project management approaches we initially tended to stride away from, at times. However, as much as the matter has already been discussed orally, I would like to insist on

the importance of allocating school time to the project, especially in the earliest stages. The lack thereof, added to the swift pace at which modules went by and the sheer amount of work we had to hand in made all progress painstaking to achieve. Moreover, meeting with Altyor with little to no progress to discuss in the early stages of the project felt like an embarrassing waste of their time.

Work distribution

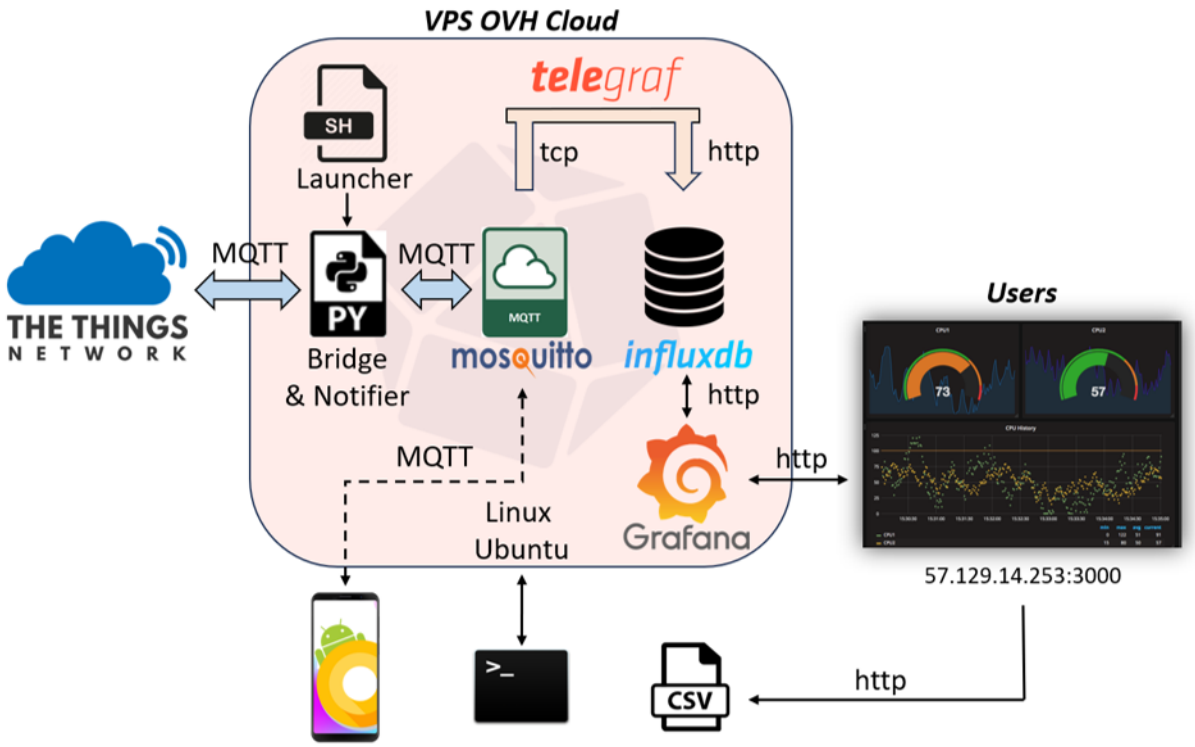
Work distribution	Achievement	Angela Nana	Antoine Goulin	Arnaud Fabre
Acquisition of temperature and humidity	100%	8h		
Program logic	100%	12h		
Deep-Sleep	100%	11h		
Code refactoring	100%		10h	
LoRaWAN communication	100%		14h	
BLE connection	70%		10h (No app)	
Server & Security	75%		2h	6h (some measures unimplemented)
Installation & configuration	100%			10h
Bridge & filter & launcher	100%			12h
Visualization	100%			10h
Smartphone notifier	25%			2h (No app for the moment)
TOTAL		31h	36h	40h

Figure 27: Estimated time spent on the mains tasks

Functional Requirements

ID	Function title	Sub-functions	Criteria	Levels
MF 1	Measuring parameters of the windrow's content	Mesuring temperature	Measurement range	-20 à 90 °C
			Accuracy	± 1 °C
			Number of sensors	3
			Distance between sensors	20 cm depth-wise
		Measuring humidity (moisture)	Measurement range	0 - 100 %
			Accuracy	± 5 %
			Number of sensors	3
			Distance between sensors	20 cm depth-wise
MF 2	Transmitting data	Sending data wirelessly	Transmission range	500 m
CF 1	Lifetime (in use)	Lasting long without charging	Lifetime	5 years
CF 2	Environmental constraints	Withstanding compost moisure	Moisture level	60 - 70 %
		Withstanding weather	Waterproofing	IP67
		Resisting compost's density	Operating temperature range Probe-end stiffness	-20 to 90 °C to be determined
		Resisting chemical reactions	Compost pH	Neutral (6.5 to 8)
EF 1	Receiving data	Receiving data from multiple devices	Nr of simultaneous devices	30
EF 2	Managing all collected data	Storing data	Storage + redundancy space	Several GB (1 JSON = 1 kB)
		Visualising data	Interface simplicity	Clear graphs, simple gauges
EF 3	Configuring the device	Setting measurement frequency		
		Allow calibration		
ECF 1	Being robust	Accepting a large number of simultaneous requests	Use case	One server for multiple clients
		Being secure	Potential attacks	DoS, ransomwares

Server architecture details



Grafana dashboard view