

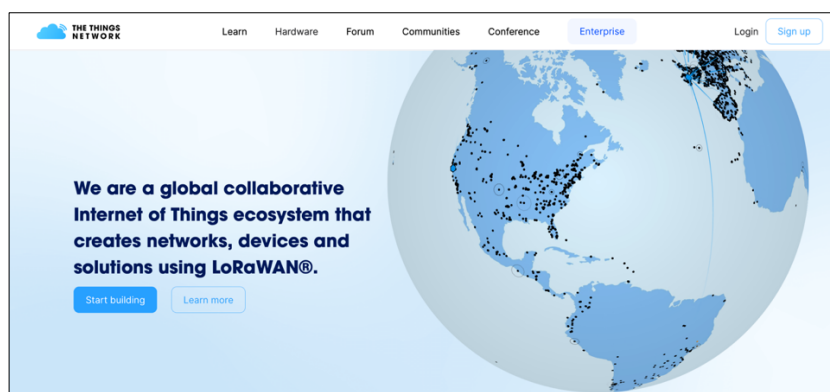
TP 1 CARTE ARDUINO MKR 1310 SUR LE RÉSEAU LORAWAN TTN

Objectifs :

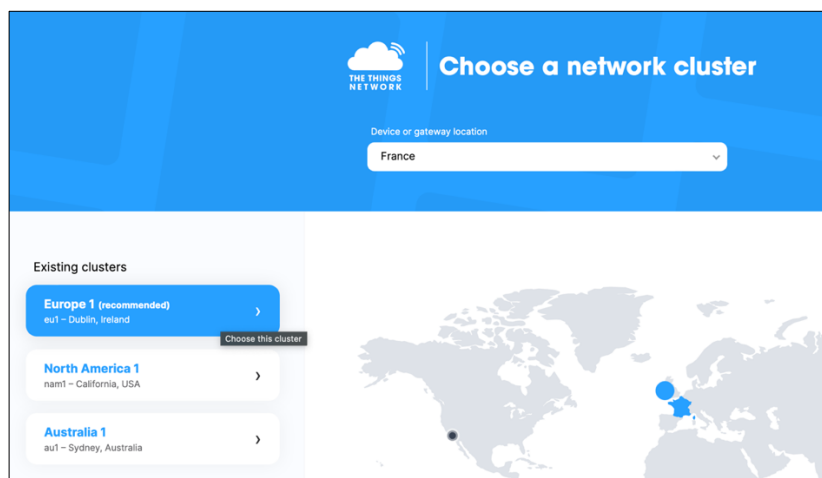
- Créer un compte d'accès au réseau LoRaWan TTN (TheThingsNetwork).
- Connecter la carte Arduino MKR1310 LoRa sur le réseau LoRaWan de TTN.
- Transmettre (uplink) et recevoir (downlink) des données vers et depuis le réseau TTN LoRaWan.
- Comprendre la classe A pour la réception des données (downlink).
- Uplink et downlink via MQTT
- Mesurer et transmettre la température et l'humidité à l'aide d'un capteur DHT22 et la carte Arduino MKR1310.
- Déployer un dashboard sur node-red.

Création du compte TTN :

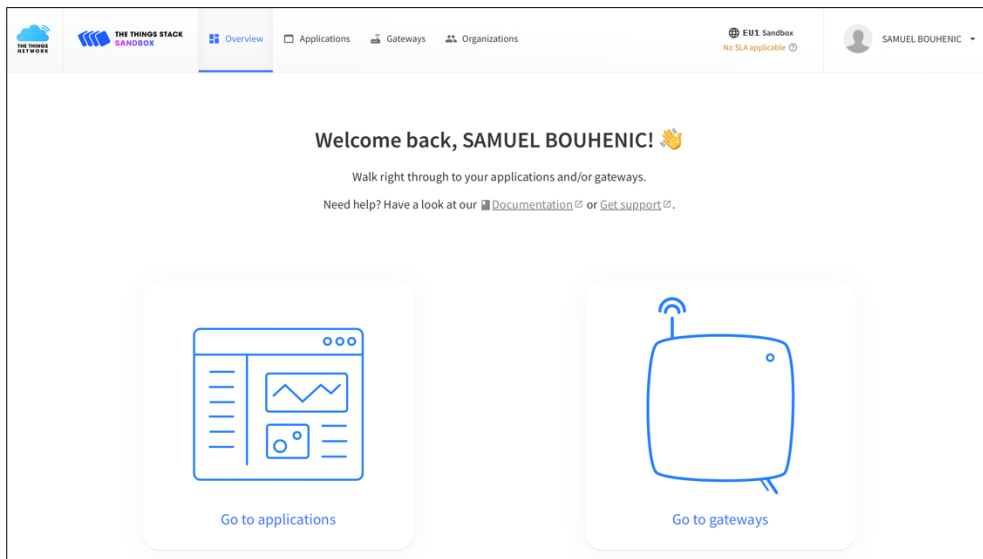
1. Connectez-vous sur le site <https://thethingsnetwork.org>.
2. Cliquez sur sign up et laissez-vous guider pour la création du compte TTN.



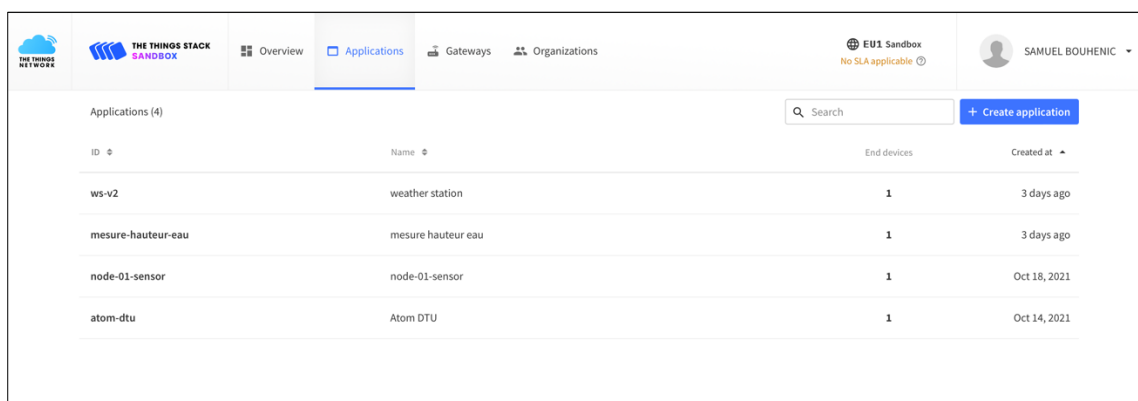
3. Sélectionner le cluster Europe :



4. Créer votre première application (pas besoin d'installer une passerelle, elle est déjà présente et configurée sur le réseau) :



5. Create application :



6. Compléter les champs (attention, pour l'application ID, il faut des minuscules)

The screenshot shows the 'Create application' form. It includes a title 'Create application', a brief explanation of applications, and a link to the 'Adding Applications' guide. The form has three main input fields: 'Application ID' (with a red asterisk), 'Application name', and 'Description'. Below the description field is a note: 'Optional application description; can also be used to save notes about the application'. At the bottom is a 'Create application' button.

Create application

Within applications, you can register and manage end devices and their network data. After setting up your device fleet, use one of our many integration options to pass relevant data to your external services.
Learn more in our guide on [Adding Applications](#).

Application ID *

my-new-application

Application name

My new application

Description

Description for my new application

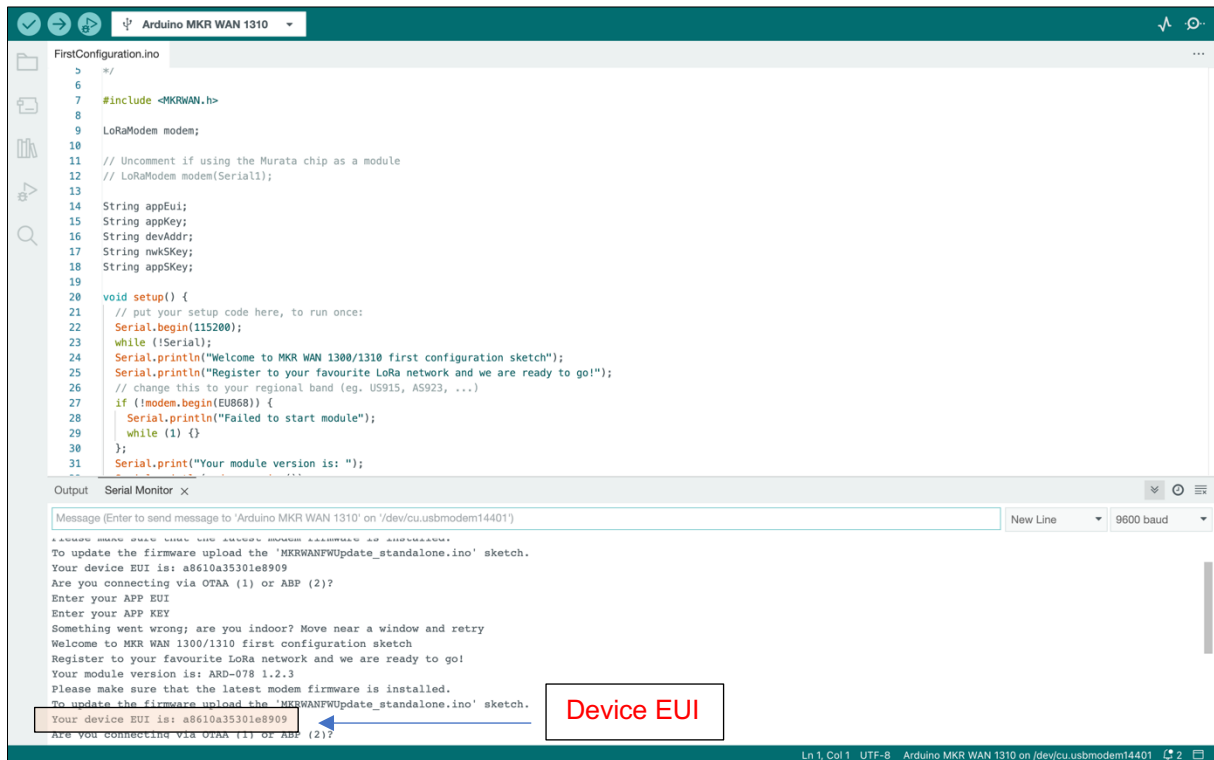
Optional application description; can also be used to save notes about the application

Create application

Connecter la carte Arduino MKR1310 LoRa sur le réseau LoRawan de TTN.

Préparation :

1. Connecter la carte Arduino MKR 1310 et lancer l'IDE Arduino.
2. Vérifier que la carte est reconnue par l'IDE (outils/board).
3. Vérifier que les ports.
4. Charger le sketch fichier/examples/MKRWAN/FirstConfiguration
5. Téléverser le programme dans la carte et copier/coller votre device EUI.

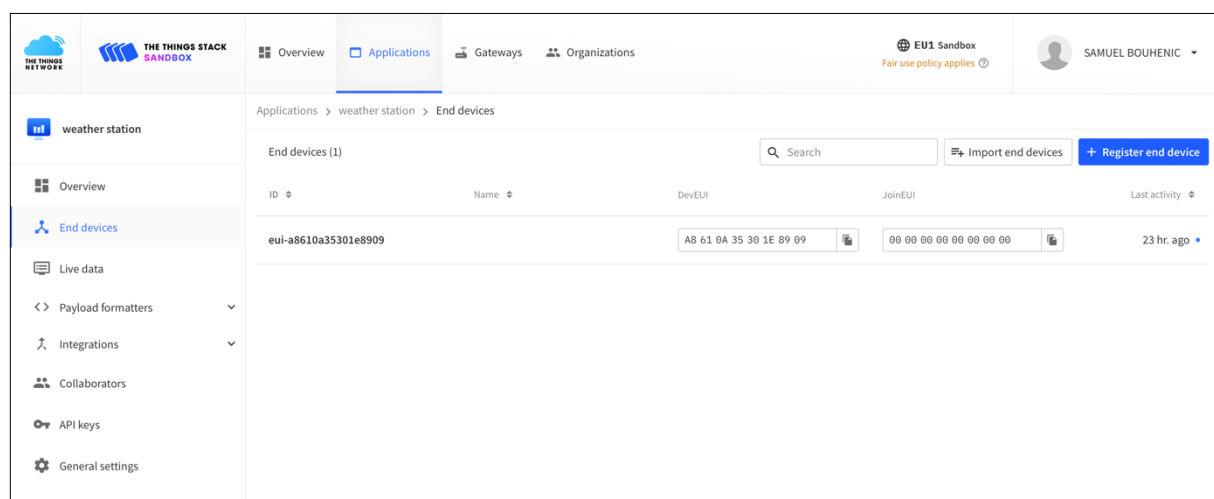


The screenshot shows the Arduino IDE interface. The top pane displays the 'FirstConfiguration.ino' sketch. The bottom pane shows the 'Serial Monitor' window with the following output:

```
Message (Enter to send message to 'Arduino MKR WAN 1310' on '/dev/cu.usbmodem14401')
To update the firmware upload the 'MKRWANFWUpdate_standalone.ino' sketch.
Your device EUI is: a8610a35301e8909
Are you connecting via OTAA (1) or ABP (2)?
Enter your APP EUI
Enter your APP KEY
Something went wrong; are you indoor? Move near a window and retry
Welcome to MKR WAN 1300/1310 first configuration sketch
Register to your favourite LoRa network and we are ready to go!
Your module version is: ARD-078 1.2.3
Please make sure that the latest modem firmware is installed.
To update the firmware upload the 'MKRWANFWUpdate_standalone.ino' sketch.
Your device EUI is: a8610a35301e8909
Are you connecting via OTAA (1) or ABP (2)?
```

A red box labeled "Device EUI" points to the line "Your device EUI is: a8610a35301e8909" in the Serial Monitor output.

6. Reconnectez-vous sur votre application TTN et enregistrez un end-device.



7. Choisissez la marque Arduino SA, le modèle MKR WAN, version hardware 1.0, version firmware 1.2.3 et profile EU_863_870

weather station

Overview

End devices

Live data

Payload formatters

Integrations

Collaborators

API keys

General settings

Register end device

Does your end device have a LoRaWAN® Device Identification QR Code? Scan it to speed up onboarding.

Scan end device QR code [Device registration help](#)


End device type

Input method

☒ Select the end device in the LoRaWAN Device Repository

☐ Enter end device specifics manually

End device brand Model Hardware Ver. Firmware Ver. Profile (Region)



Arduino MKR WAN 1310

LoRaWAN Specification 1.0.2, RP001 Regional Parameters 1.0.2, Over the air activation (OTAA), Class A

The Arduino MKR WAN 1310 is a development board that provides a practical and cost-effective solution to add LoRaWAN® connectivity for projects requiring long-range, low-power wireless communication. Sensors and actuators can be connected to the board through the analog, digital, UART, SPI, and I2C pins. The MKR WAN 1310 comes complete with an ATECC508 secure element, a battery charger, 2MByte SPI Flash, and power consumption as low as 104 uA.

[Product website](#)

Frequency plan

< Hide sidebar

8. On choisit le plan de fréquence (SF9 for RX2).

weather station

Overview

End devices

Live data

Payload formatters

Integrations

Collaborators

API keys

General settings


Register end device

Input method

☒ Select the end device in the LoRaWAN Device Repository

☐ Enter end device specifics manually

End device brand Model Hardware Ver. Firmware Ver. Profile (Region)



Arduino MKR WAN 1310

LoRaWAN Specification 1.0.2, RP001 Regional Parameters 1.0.2, Over the air activation (OTAA), Class A

The Arduino MKR WAN 1310 is a development board that provides a practical and cost-effective solution to add LoRaWAN® connectivity for projects requiring long-range, low-power wireless communication. Sensors and actuators can be connected to the board through the analog, digital, UART, SPI, and I2C pins. The MKR WAN 1310 comes complete with an ATECC508 secure element, a battery charger, 2MByte SPI Flash, and power consumption as low as 104 uA.

[Product website](#)

Frequency plan

JoinEUI [View glossary page](#)

To continue, please enter the JoinEUI of the end device so we can determine onboarding options

9. On complète le Join EUI que l'on appelle aussi l'App EUI : 00 00 00 00 00 00 00 00.
10. On complète de Dev EUI avec la valeur relevé sur l'IDE Arduino.

11. On génère le l'AppKey.
12. Valider « Register end device ».
13. On retourne sur l'IDE Arduino.
14. Sélectionner la méthode OTAA (option 1).
15. À la question App EUI, on recopie 00 00 00 00 00 00 00 00 sans les espaces.
16. À la question App Key, on recopie la valeur générée sur TTN.

*Si tout se déroule correctement, « message sent correctly » apparaît.
Le device est maintenant enregistré sur le réseau TTN.*

*On peut vérifier sur TTN le message reçu. Il apparaît dans l'onglet « live data ».
Le payload reçu est en hexadécimal. On peut le décoder à l'aide d'un script JS sur l'onglet
« Payload Formatters », Le message reçu est : « **HeLoRA world!** ».*

Exemple de code JS :

Source : <https://github.com/bouhenic/FormationIOT/blob/main/Tp1TTN/decodePayloadPart1.js>

```
function Decoder(bytes, port) {
  var result = "";
  for (var i = 0; i < bytes.length; i++) {
    result += String.fromCharCode(parseInt(bytes[i]));
  }
  return { payload: result };
}
```

Transmettre (uplink) et recevoir (downlink) des données vers et depuis le réseau TTN LoRaWan.

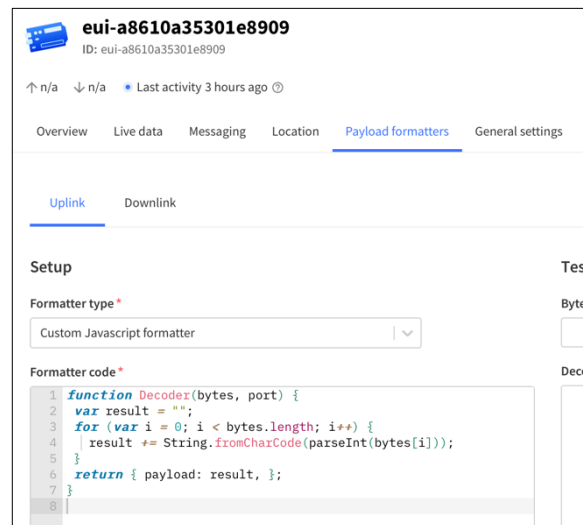
1. Ouvrir le sketch LoraSendAndReceive.ino

On constate qu'il est associé à un fichier `arduino_secrets.h`. Ce fichier contiendra l'APP EUI et l'APP KEY.

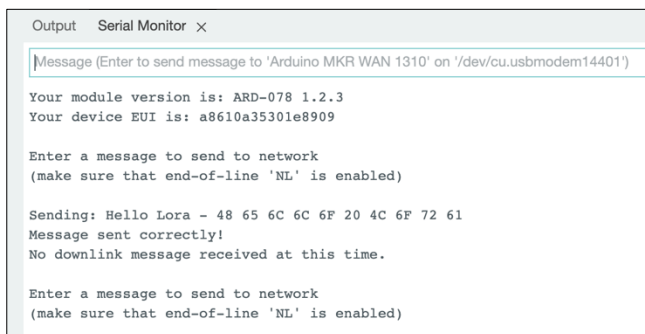
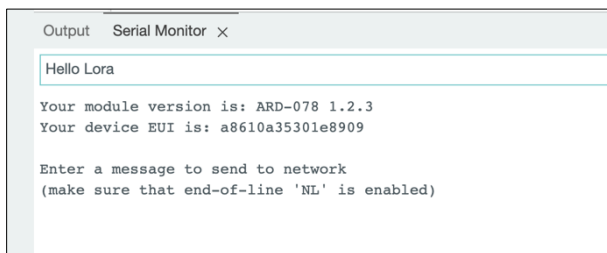
2. Recopier ces identifiants sur le fichier et téléverser le sketch :



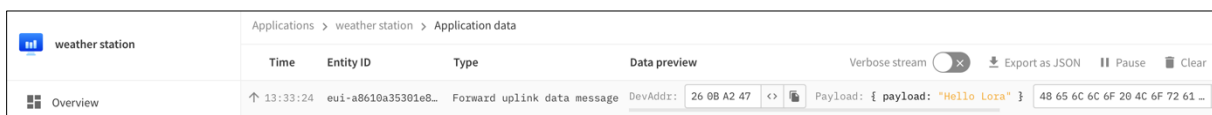
3. Dans l'onglet Payload formatters, compléter le code JS de décode du payload uplink.



4. Saisir un message à envoyer.



5. Dans l'onglet live data, on constate la réception du message



Comprendre la classe A pour la réception des données (downlink).

Ci-dessous l'espace pour soumettre un message en downlink sur TTN.

eui-a8610a35301e8909
ID: eui-a8610a35301e8909

↑ n/a ↓ n/a • Last activity 4 hours ago

Overview Live data **Messaging** Location Payload formatters General settings

Uplink **Downlink**

Schedule downlink

Insert Mode

☒ Replace downlink queue
☐ Push to downlink queue (append)

FPort*

1

Payload type

☒ Bytes ☐ JSON

Payload

AA

The desired payload bytes of the downlink message

☐ Confirmed downlink

Schedule downlink

1. Ici, on programme l'envoi de l'octet AA. On peut vérifier dans le « live data » le schedule du downlink.

Overview	Live data	Messaging	Location	Payload formatters	General settings
Time	Type	Data preview			
↓ 13:33:25	Schedule data downlink for...				

2. Pour déclencher le flux downlink vers le device, il faut que celui-ci émettent un message (uplink).

```
Sending: test downlink - 74 65 73 74 20 64 6F 77 6E 6C 69 6E 6B
Message sent correctly!
No downlink message received at this time.

Enter a message to send to network
(make sure that end-of-line 'NL' is enabled)

Sending: test downlink2 - 74 65 73 74 20 64 6F 77 6E 6C 69 6E 6B 32
Message sent correctly!
Received: AA
```

On s'aperçoit que nous sommes obligé d'envoyer 2 messages uplink pour recevoir le flux downlink. En modifiant le code, en augmentant le délai après le uplink (on choisit arbitrairement 10s), on résout le problème.

```

LoraSendAndReceive.ino  arduino_secrets.h
64   modem.beginPacket();
65   modem.print(msg);
66   err = modem.endPacket(true);
67   if (err > 0) {
68     Serial.println("Message sent correctly!");
69   } else {
70     Serial.println("Error sending message :(");
71     Serial.println("(you may send a limited amount of messages per minute, depending on the signal strength");
72     Serial.println("it may vary from 1 message every couple of seconds to 1 message every minute)");
73   }
74   delay(10000);
75   if (!modem.available()) {
76     Serial.println("No downlink message received at this time.");
77     return;
78   }
79   char rcv[64];
80   int i = 0;
81   while (modem.available()) {
82     rcv[i++] = (char)modem.read();
83   }
84   Serial.print("Received: ");
85   for (unsigned int j = 0; j < i; j++) {
86     Serial.print(rcv[j] >> 4, HEX);
87     Serial.print(rcv[j] & 0xF, HEX);
88     Serial.print(" ");
89   }
90   Serial.println();

```

Uplink et downlink via MQTT :

1. Dans Integrations/MQTT, on peut relever certaines informations pour écrire la commande subscribe mqtt :

The screenshot shows the 'MQTT' configuration page in the TTS interface. The left sidebar lists various integration options, with 'MQTT' selected. The main content area is titled 'MQTT' and provides an overview of the protocol. Below this, there are sections for 'Further resources' (linking to MQTT server and official website), 'Connection information', and 'MQTT server host' details. The 'MQTT server host' section includes fields for 'Public address' and 'Public TLS address', both set to 'eu1.cloud.thethings.network:1883'. The 'Connection credentials' section shows a 'Username' field set to 'ws-v2@ttn' and a 'Password' field with a 'Generate new API key' button.

Le domaine du broker mqtt : *eu1.cloud.thethings.network*
 On voit ici qu'il peut être utilisé en mqtt (port 1883) ou mqtts (port 8883).
 On peut relever le username : ici *ws-v2@ttn*
 On peut générer une API key qui sera le password mqtt

Le topic est de la forme : v3/id-application/devices/id-device/up

On sait que la commande mosquitto mqtt est de la forme : mosquitto_sub -h **domaine-serveur** -p **port** -u **username** -P **password** -t '**topic**'

Dans mon cas, cela donne :

- Domaine-serveur : eu1.cloud.thethings.network
- Port : 8883
- Username : ws-v2@ttn
- Password :
NNSXS.F2YZU33TCK4H2GCSS25XTCP2YU3PHABGQ46DGRQ.SXGHUHBPRZWN5YRUW5I6D2LSDBS0D5XPJK7XKFNUVNOKS5B4MBQQ
- Topic : v3/ws-v2@ttn/devices/eui-a8610a35301e8909/up

```
mosquitto_sub -h eu1.cloud.thethings.network -p 8883 -u ws-v2@ttn -P NNSXS.F2YZU33TCK4H2GCSS25XTCP2YU3PHABGQ46DGRQ.SXGHUHBPRZWN5YRUW5I6D2LSDBS0D5XPJK7XKFNUVNOKS5B4MBQQ -t 'v3/ws-v2@ttn/devices/eui-a8610a35301e8909/up'
```

réponse du broker :

```
{
  "end_device_ids": {
    "device_id": "eui-a8610a35301e8909",
    "application_ids": {
      "application_id": "ws-v2"
    },
    "dev_eui": "A8610A35301E8909",
    "join_eui": "0000000000000000",
    "dev_addr": "2608CD17",
    "correlation_ids": {
      "gs": "uplink:01HK7QB3RVF1EB4XABFS8YNN"
    },
    "received_at": "2024-01-03T13:13:56.806101208Z",
    "uplink_message": {
      "session_key_id": "AyzPVeh00J+Z66srkhFDQ==",
      "f_port": 2,
      "f_cnt": 4,
      "frm_payload": "dGVzdF9tb3NxdWI0dG9fc3Vi",
      "decoded_payload": {
        "payload": "test_mosquitto_sub",
        "rx_metadata": {
          "gateway_ids": {
            "gateway_id": "eui-00e40ffff2a3fed"
          },
          "eui": "C0EE40FFFF2A3FED",
          "timestamp": "4223200131",
          "rssi": -43,
          "channel_rssi": -43,
          "snr": 10,
          "location": {
            "latitude": 49.76058425609115,
            "longitude": 0.36482810873796906,
            "source": "SOURCE_REGISTRY"
          },
          "uplink_token": "CiIKIAoUZVpLWmWzU0MGZmZmYyYTNmZWQSCMDUQP/Kj/tEIPK490PgwlLLvVrAYQ3rqJngIgu0/f0vT0Aq==",
          "channel_index": 3,
          "received_at": "2024-01-03T13:13:56.599940446Z"
        },
        "settings": {
          "data_rate": {
            "lorawan": {
              "bandwidth": 125000,
              "spreading_factor": 7,
              "coding_rate": "4/5"
            },
            "frequency": "867100000",
            "timestamp": "4223200131",
            "received_at": "2024-01-03T13:13:56.601034262Z",
            "confirmed": true,
            "consumed_airtime": "0.071936s",
            "version_ids": {
              "brand_id": "arduino",
              "model_id": "mkr-wan-1310",
              "hardware_version": "1.0",
              "firmware_version": "1.2.3",
              "band_id": "EU_863_870"
            },
            "network_ids": {
              "net_id": "000013",
              "ns_id": "EC656E0000000181",
              "tenant_id": "ttn",
              "cluster_id": "eu1",
              "cluster_address": "eu1.cloud.thethings.network"
            }
          }
        }
      }
    }
  }
}
```

C'est au format JSON :

On retrouve le payload codé en base 64 et son décodage :

- frm_payload: dGVzdF9tb3NxdWI0dG9fc3Vi
- payload : test_mosquitto_sub

Pour publier une valeur vers le broker, la commande mosquitto a la forme :

mosquitto_pub -h **domaine-serveur** -p **port** -u **username** -P **password** -t '**topic**' -m '{"downlinks": [{"f_port": 1, "frm_payload": "**message-en-base64**"}]}'

Dans notre cas

```
mosquitto_pub -h eu1.cloud.thethings.network -p 8883 -u ws-v2@ttn -P NNSXS.F2YZU33TCK4H2GCSS25XTCP2YU3PHABGQ46DGRQ.SXGHUHBPRZWN5YRUW5I6D2LSDBS0D5XPJK7XKFNUVNOKS5B4MBQQ -t 'v3/ws-v2@ttn/devices/eui-a8610a35301e8909/down/replace' -m '{"downlinks": [{"f_port": 1, "frm_payload": "AQ=="}]}'
```

AQ== en base 64 donne 01 en hexa.

Côté device, on reçoit 01 après émission d'un uplink :

```
Enter a message to send to network
(make sure that end-of-line 'NL' is enabled)

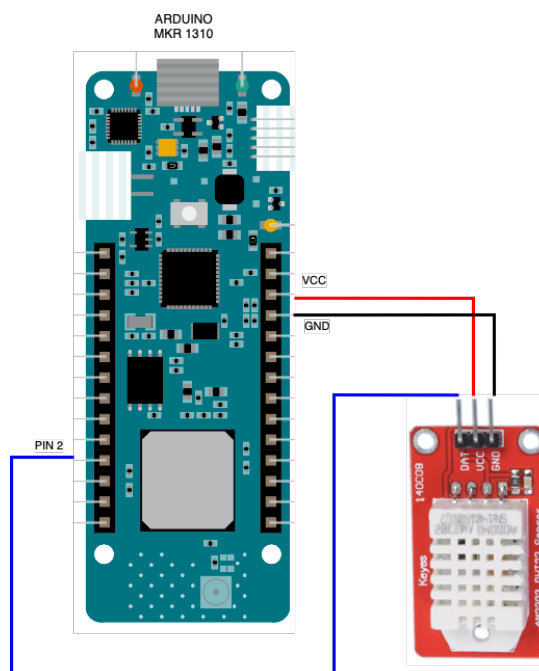
Sending: test mosquitto_pub - 74 65 73 74 20 6D 6F 73 71 75 69 74 74 6F 5F 70 75 62
Message sent correctly!
Received: 01
```

Pour s'assurer du transfert, on peut imposer la priorité en rajoutant `priority :highest`
Ce qui donne :

```
mosquitto_pub -h eu1.cloud.thethings.network -p 8883 -u ws-v2@ttn -P
NNSXS.F2YZU33TCK4H2GCSS25XTCP2YU3PHABGQ46DGRQ.SXGHUHBZPNW5YRUW5I6D2
LSDBSOD5XPJK7XKFNUVN0KS5B4MBQ -t 'v3/ws-v2@ttn/devices/eui-
a8610a35301e8909/down/replace' -m '{"downlinks":[{"f_port":
1,"frm_payload":"AQ==","priority": "HIGHEST"}]}'
```

Il existe différents niveaux de priorité : LOW, NORMAL, HIGH et HIGHEST

Mesurer et transmettre la température et l'humidité à l'aide d'un capteur DHT22 et la carte Arduino MKR1310.



Sketch :

Source : <https://github.com/bouhenic/FormationIOT/blob/main/Tp1TTN/sketchLoraDht22.ino>

```
#include <MKRWAN.h>
#include "DHT.h"

LoRaModem modem;

// Définissez les informations de votre réseau LoRaWAN
const char *appEui = "0000000000000000";
const char *appKey = "5C527757EDB0B87DD9767568125889F0"; //indiquer ici votre appkey
```

```

#define DHTPIN 2          // Pin où le DHT22 est connecté
#define DHTTYPE DHT22    // Utilisation du DHT22
DHT dht(DHTPIN, DHTTYPE);

void setup() {
    Serial.begin(115200);
    while (!Serial);

    dht.begin();
    if (!modem.begin(EU868)) {
        Serial.println("Échec de l'initialisation du modem");
        while (1);
    }
    if (!modem.joinOTAA(appEui, appKey)) {
        Serial.println("Échec de la connexion");
        while (1);
    }
    Serial.println("Connecté au réseau LoRaWAN");
}

void loop() {
    float humidity = dht.readHumidity();
    float temperature = dht.readTemperature();
    if (isnan(humidity) || isnan(temperature)) {
        Serial.println("Échec de la lecture du capteur DHT");
        return;
    }
    // Convertir les valeurs en entiers pour simplifier l'envoi
    uint16_t h = humidity * 100;      // Par exemple, 55.12% devient 5512
    uint16_t t = (temperature + 100) * 100; // Par exemple, 24.76°C devient 2476 (avec un
    // décalage de +100 pour gérer les températures négatives)

    // Créer un tableau de 4 octets pour envoyer les données
    byte payload[4];
    payload[0] = h >> 8;
    payload[1] = h & 0xFF;
    payload[2] = t >> 8;
    payload[3] = t & 0xFF;

    modem.beginPacket();
    modem.write(payload, 4);
    if (modem.endPacket(true) > 0) {
        Serial.println("Message envoyé avec succès");
    } else {
        Serial.println("Erreur lors de l'envoi du message");
    }

    delay(240000); // Attendre 4 minutes avant le prochain envoi
}

```

Source : <https://github.com/bouhenic/FormationIOT/blob/main/Tp1TTN/decodePayloadPart2.js>

Sur le live data, au niveau de l'application TTN :

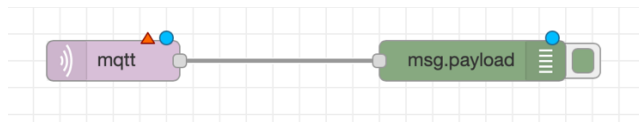
Depuis mosquitto :

[illegible]

```
decoded_payload":{"humidity":42.5,"temperature":27.099999999999994}
```

Déploiement d'un dashboard sur NodeRed

On commence par tester l'abonnement mqtt. On utilise un nœud mqtt in et un nœud debug :



Paramétrage du nœud mqtt in :

Edit mqtt in node > Edit mqtt-broker node

Delete Cancel Update

Properties

Name mqtt client node red

Connection Security Messages

Server eu1.cloud.thethings.network Port 8883

☒ Enable secure (SSL/TLS) connection

TLS Configuration TLS configuration

Client ID Leave blank for auto generated

☐ Keep alive time (s) 60 ☒ Use clean session

☐ Use legacy MQTT 3.1 support

Edit mqtt in node > Edit mqtt-broker node > Edit tls-config node

Delete Cancel Update

Properties

☐ Use key and certificates from local files

Certificate Upload

Private Key Upload

Passphrase private key passphrase (optional)

CA Certificate Upload

☒ Verify server certificate

Server Name for use with SNI

Name

Dans le champ server, on indique le domaine du broker TTN. On choisit le port 8883 pour être en mqtts, on coche Enable secure (SSL/TLS) connection. On édite la TLS configuration et on coche « Verify server certificate ».

Edit mqtt in node > Edit mqtt-broker node

Delete Cancel Update

Properties

Name mqtt client node red

Connection Security Messages

Username ws-v2@ttn

Password

Edit mqtt in node

Delete Cancel Done

Properties

Server mqtt client node red

Topic v3/ws-v2@ttn/devices/eui-a8610a35301e8909/uc

QoS 0

Output auto-detect (string or buffer)

Name

Dans l'onglet « security », on indique le username et le password. Dans l'onglet de propriété principal, on n'oublie pas d'indiquer le topic.

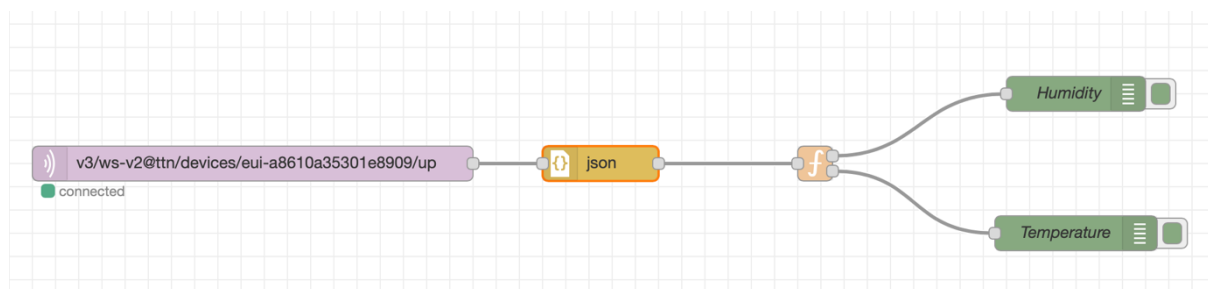
Dans l'onglet « debug », on pourra tester la réception dès que le capteur transmet son uplink sur TTN.

04/01/2024 à 13:40:48 node: c0b52c1e.f3c39

v3/ws-v2@ttn/devices/eui-a8610a35301e8909/up : msg.payload : string[1399]

```
{"end_device_ids":{"device_id":"eui-a8610a35301e8909","application_ids":{"application_id":"ws-v2"},"dev_eui":"A8610A35301E8909","join_eui":"0000000000000000","dev_addr":"260B7B58"},["gs:uplink:01HKA7V4CGBN1KAY68ZM2Z60K0"],"received_at":"2024-01-04T12:40:48.479572062Z","uplink_message":{"session_key_id":"AYzUXIf+IVnDl8W45XXCPw==","f_port":2,"f_cnt":9,"frm_payload":"Ezgw{"humidity":49,"temperature":25.2},"rx_metadata":[{"gateway_ids":{"gateway_id":"eui-c0ee40ffff2a3fed","eui":"C0EE40FFFF2A3FED"},"timestamp":607457779,"rssi":-36,"channel":{"latitude":49.76050425609115,"longitude":0.36482810873796906,"source":"SOURCE_REGISTERED"},"settings":{"data_rate":{"lora":{"bandwidth":125000,"spreading_factor":7,"coding_rate":"4/5"}}},"frequency":"868500000"
```

On constate que le résultat est un objet Json. On va devoir le convertir en objet javascript pour pouvoir traiter le résultat. C'est le rôle du nœud json.



On choisit pour le nœud Json l'action « Always convert to javascript object ».

Edit json node

DeleteCancelDone

⚙ Properties

⦿ ActionAlways convert to JavaScript Object

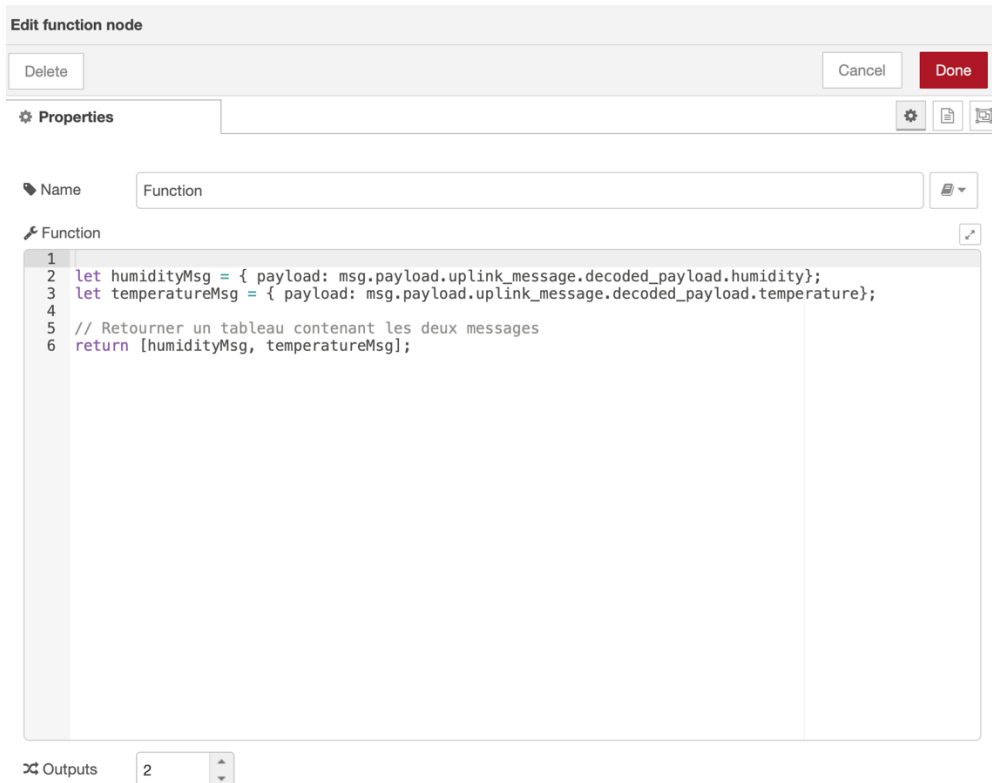
⋮ Propertymsg. payload

🏷 NameName

Le nœud fonction va nous permettre la sélection des propriétés « humidity » et « temperature ». Ces propriétés sont présentes dans l'objet js « decoded payload » qui se trouve dans l'objet js « uplink_message ».

Ci-joint le code JS pour récupérer les 2 grandeurs. Celles-ci seront redirigées vers 2 sorties. On indique donc 2 outputs. HumidityMsg sera redirigée vers la première sortie et temperatureMsg vers la deuxième sortie.

Source : <https://github.com/bouhenic/FormationIOT/blob/main/Tp1TTN/functionNodeRed.js>



Dans « debug », on doit recevoir les 2 valeurs :

04/01/2024 à 13:36:48 node: Humidity

msg.payload : number

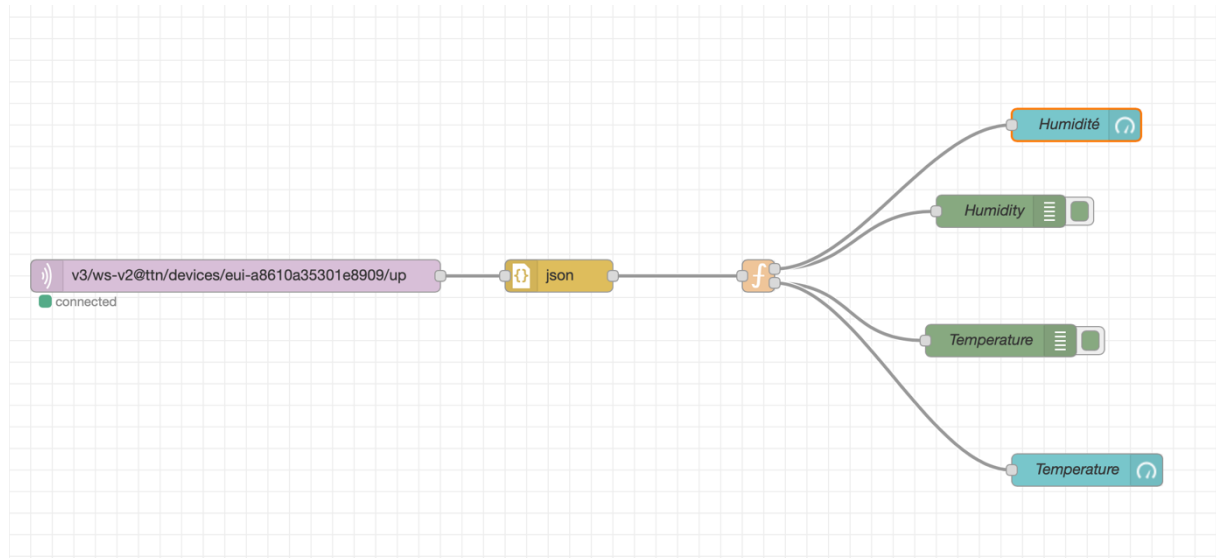
49

04/01/2024 à 13:36:48 node: Temperature

msg.payload : number

25.3

On rajoute maintenant 2 nœuds Gauge de type dashboard.



On obtient le résultat suivant :

