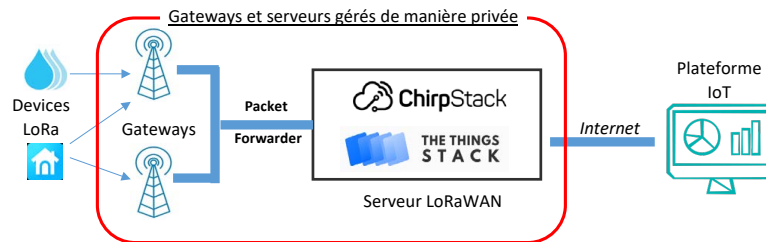


TP 3 SERVEUR LORAWAN PRIVÉ CHIRPSTACK

Objectifs :

- Installer et configurer un serveur Lorawan privé.
- Configurer la passerelle Lorawan.
- Enregistrer un device sur le serveur privé.

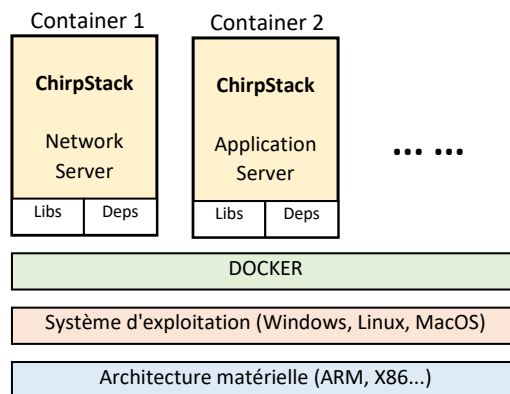


1. INSTALLATION DU SERVEUR CHIRPSTACK

1.1. Clonez le projet Chirpstack docker depuis son github

```
git clone https://github.com/chirpstack/chirpstack-docker.git
cd chirpstack-docker
```

1.2. Installation du projet avec docker et docker-compose



1.2.1. Saisissez la commande suivante qui permet la création des différents conteneurs constituant le serveur Lorawan.

```
docker-compose up
```

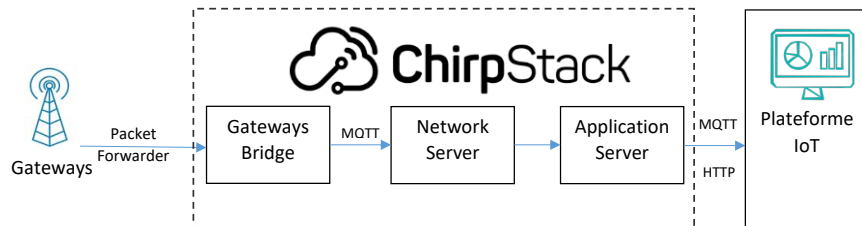
1.2.2. Saisissez dans un autre onglet du terminal la commande suivante pour vérifier les conteneurs créés :

```
docker ps
```

Parmi les conteneurs créés, Nous pouvons remarquer que le service Gateway Bridge écoute sur le port 1700/udp parce que nous utilisons le "Semtech UDP Packet forwarder". C'est le port sur lequel notre Gateway doit envoyer ses données. L'Application Server écoute sur le port 8080/tcp. C'est l'accès à l'interface web pour configurer le serveur LoRaWAN.

NOMS PORTS

```
chirpstack_chirpstack-application-server_1 0.0.0.0:8080->8080/tcp
chirpstack_chirpstack-network-server_1
chirpstack_chirpstack-gateway-bridge_1 0.0.0.0:1700->1700/udp
```



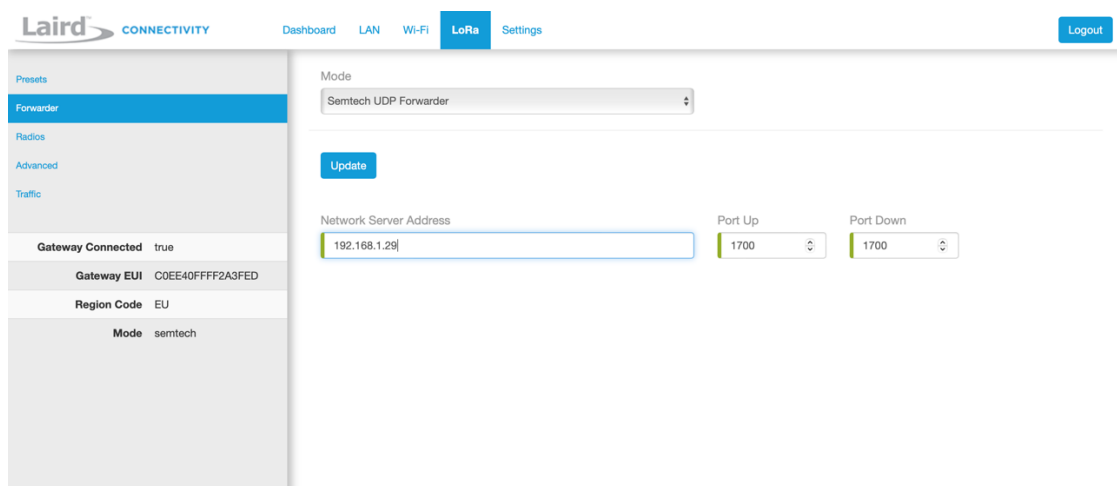
1.3. Importez les profils des principaux « device » depuis le répertoire chirpstack-docker

make import-lorawan-devices

2. CONFIGURATION DE LA PASSERELLE

2.1. Connectez-vous sur votre passerelle (l'adresse IP et les identifiants de connexion sont indiqués sur la passerelle).

2.2. Indiquez l'adresse IP de votre serveur Chirpstack et le port 1700 dans l'onglet LoRa/Forwarder de la passerelle.

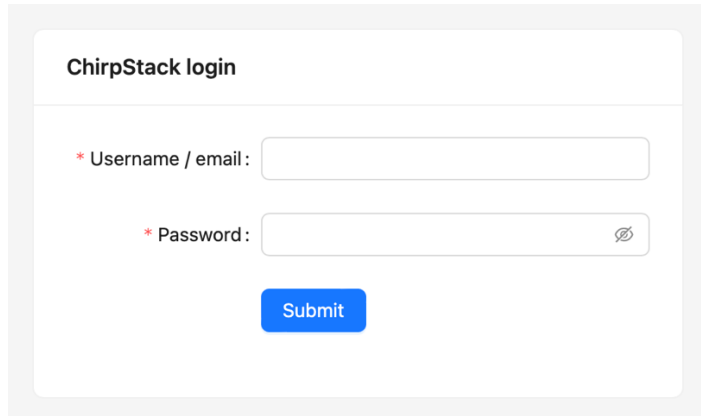


3. CONFIGURATION DE CHIRPSTACK

3.1. Configurez ChirpStack à partir de son interface web graphique disponible via le port 8080/tcp : <http://@IP-ChirpStack:8080>

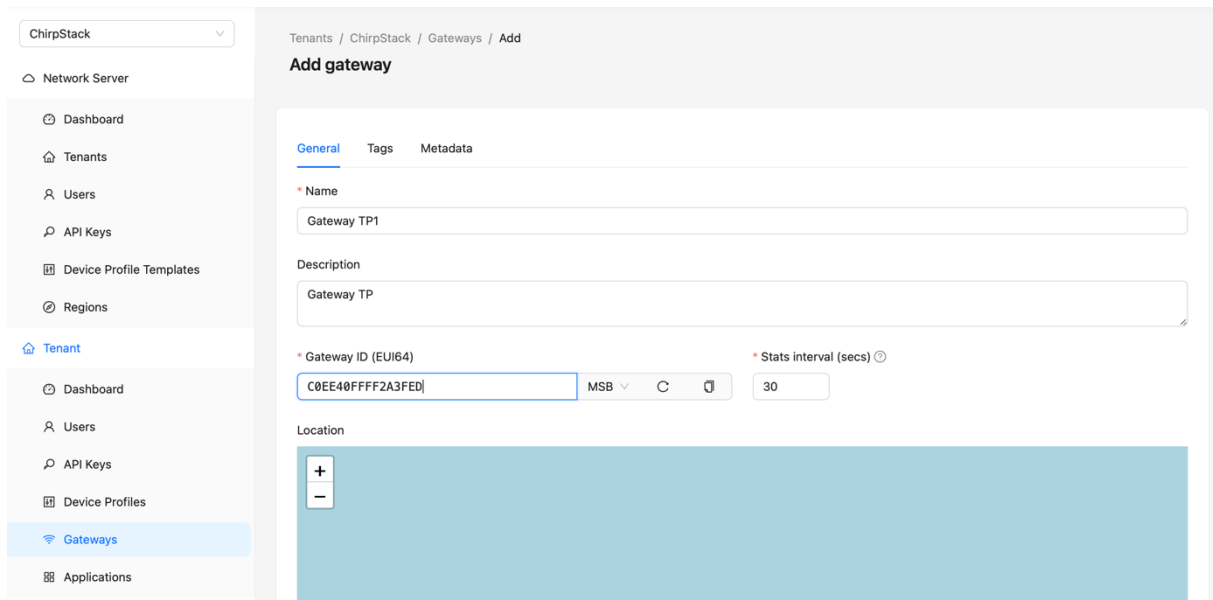
Les noms d'utilisateur et les mots de passe par défaut sont les suivants :

- Nom d'utilisateur : admin
- Mot de passe : admin



The image shows a login form titled "ChirpStack login". It contains two input fields: "Username / email" and "Password". Both fields are marked with a red asterisk. Below the fields is a blue "Submit" button.

3.2. Ajoutez les informations de la gateway (Add gateway), donnez lui un nom et complétez l'EUI gateway.



The image shows the "Add gateway" form in the ChirpStack web interface. The form is titled "Add gateway" and has three tabs: "General", "Tags", and "Metadata". The "General" tab is selected. The form contains the following fields:

- Name:** Gateway TP1
- Description:** Gateway TP
- Gateway ID (EUI64):** C0EE40FFFF2A3FED
- Stats interval (secs):** 30
- Location:** A large blue area with a "+" button in the top left corner.

4. CRÉATION DE NOTRE PREMIÈRE APPLICATION

4.1. Créez une application :

The screenshot shows the 'Add application' form in the ChirpStack interface. The left sidebar contains navigation links for Network Server, Tenant, and Applications. The main form has two tabs: 'General' and 'Tags'. Under the 'General' tab, there is a 'Name' field with the value 'Station météo' and a 'Description' field with the value 'station météo du lycée'. A 'Submit' button is located at the bottom of the form.

4.2. Créez un device profiles

This block contains three screenshots illustrating the process of creating a device profile in ChirpStack.

The first screenshot shows the 'Add device profile' form with the 'General' tab selected. The 'Name' field is empty, and the 'Description' field contains text about the Arduino MKR WAN 1310. The 'Region' dropdown is set to 'EU868'. A blue arrow points to the 'Select device-profile template' button.

The second screenshot is a modal window titled 'Select device-profile template'. It displays a list of templates with columns for Name, Region, and FW version. The 'Arduino MKR WAN 1310' template is selected, showing its region as 'EU868' and FW version as '1.2.3'.

The third screenshot shows the 'Add device profile' form with the 'General' tab selected. The 'Name' field is now filled with 'Arduino MKR WAN 1310'. The 'Region' dropdown is still set to 'EU868'. A blue arrow points to the 'Region' dropdown. A text box on the right says: 'On importe un template (on choisit ici un Arduino MKR1310 en fréquence)'. Another text box on the right says: 'Complétez le champ « configuration region » et vérifiez que l'OTAA est validé.'

The screenshot shows the 'Device profiles' table in the ChirpStack interface. The table has columns for Name, Region, MAC version, Revision, Supports OTAA, Supports Class-B, and Supports Class-C. The 'Arduino MKR WAN 1310' profile is listed with the following details:

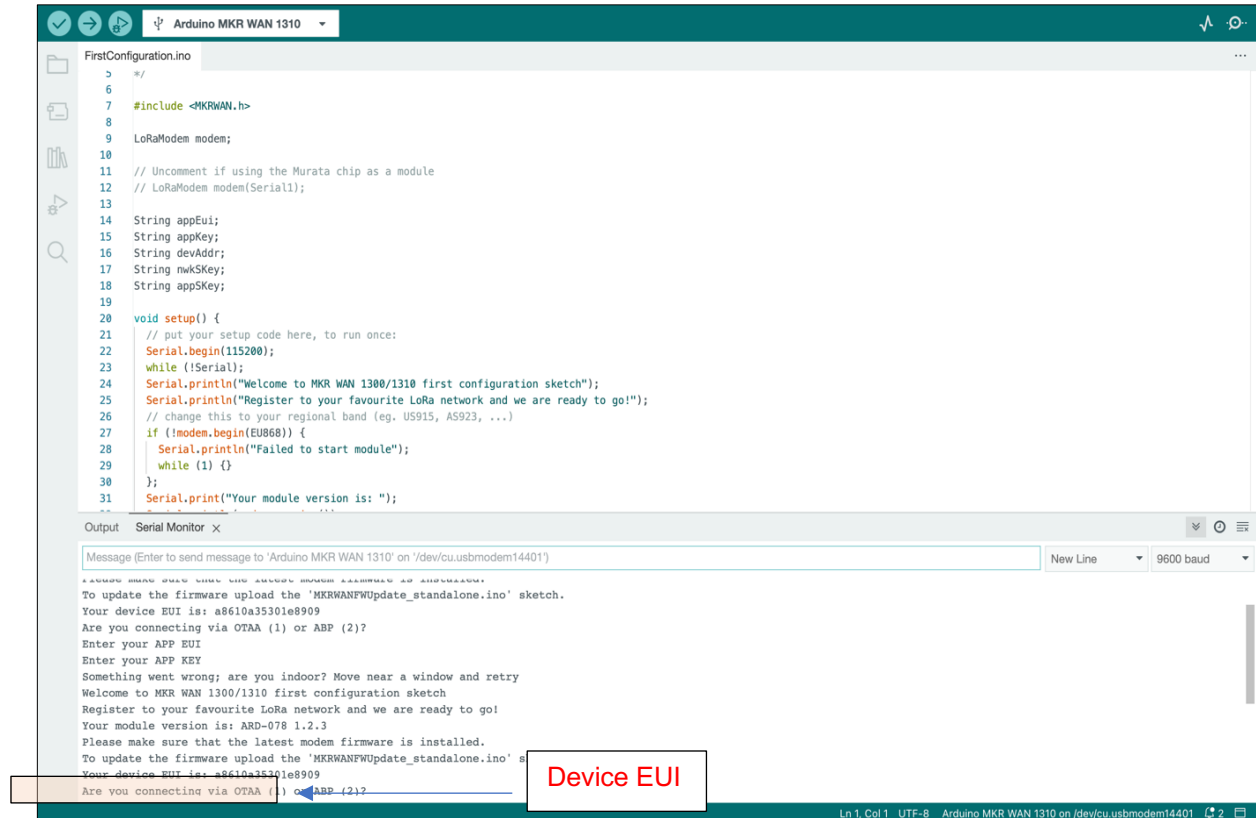
Name	Region	MAC version	Revision	Supports OTAA	Supports Class-B	Supports Class-C
Arduino MKR WAN 1310	EU868	LoRaWAN 1.0.2	A	yes	no	no

A text box on the right says: 'Résultat du profil créé après validation'.

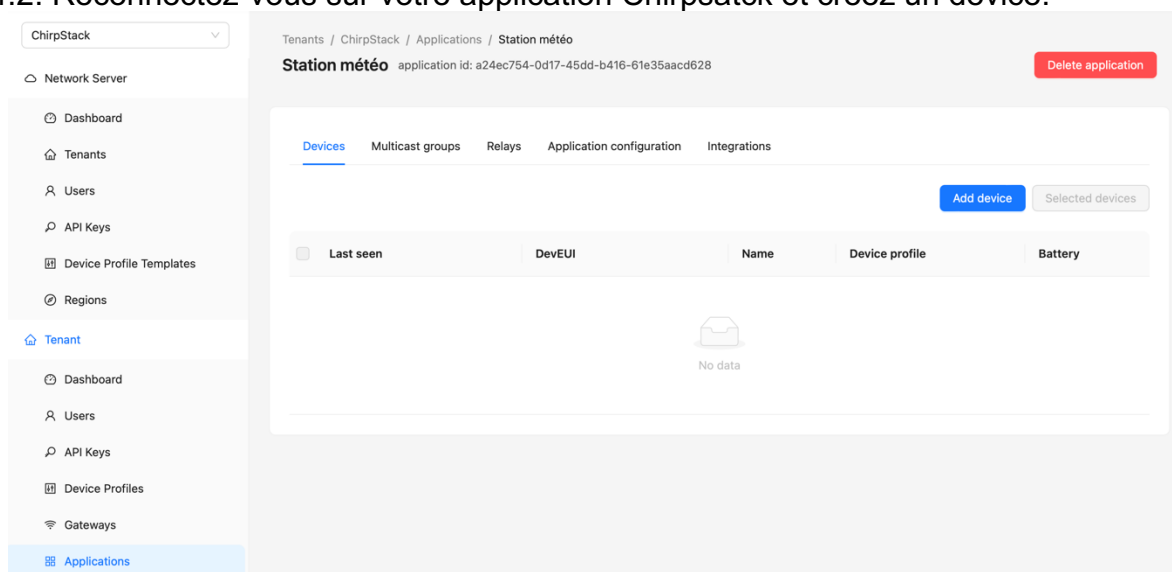
5. CRÉATION DU DEVICE

5.1. Importation du DEV EUI.

- 5.1.1. Connecter la carte Arduino MKR 1310 et lancer l'IDE Arduino.
- 5.1.2. Vérifier que la carte est reconnue par l'IDE (outils/board).
- 5.1.3. Vérifier que les ports.
- 5.1.4. Charger le sketch fichier/examples/MKRWAN/FirstConfiguration.
- 5.1.5. Téléverser le programme dans la carte et copier/coller votre device EUI.



4.2. Reconnectez-vous sur votre application Chirpsatck et créez un device.



5.2. Donnez un nom au device, choisissez le device profile créer précédemment, recopiez le device EUI précédent et complétez le join EUI (App EUI avec la valeur 00 00 00 00 00 00 00 00)

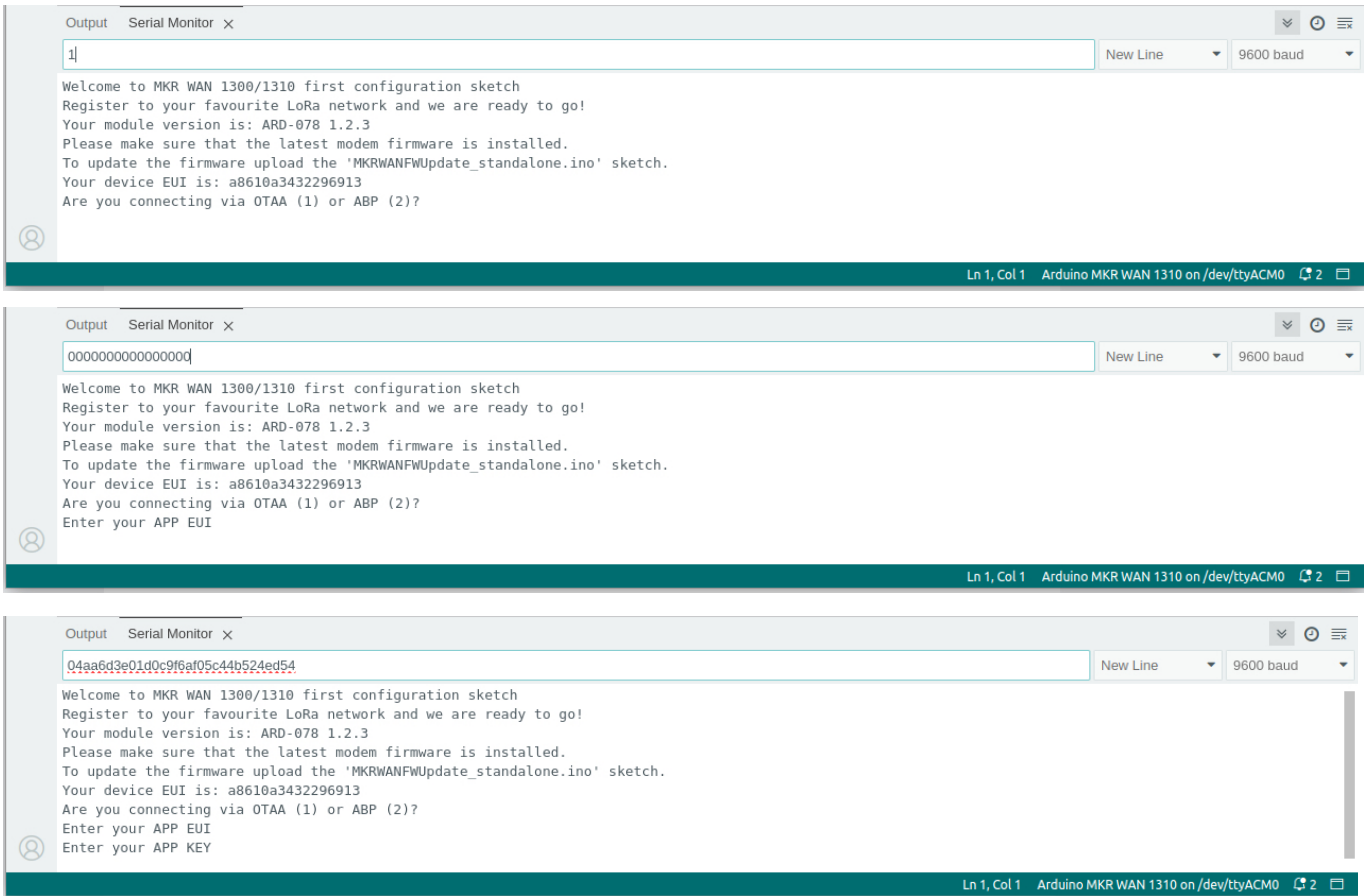
The screenshot shows the 'Add device' form in the ChirpStack interface. The breadcrumb trail is 'Tenants / ChirpStack / Applications / Station météo / Add device'. The form has three tabs: 'Device' (selected), 'Tags', and 'Variables'. Fields include: 'Name' (MKR1310), 'Description' (MKR1310 Temp et Hum), 'Device EUI (EUI64)' (a8610a3233428e03), 'Join EUI (EUI64)' (0000000000000000), and 'Device profile' (Arduino MKR WAN 1310). There are also toggle switches for 'Device is disabled' and 'Disable frame-counter validation'. A 'Submit' button is at the bottom.

5.3. Après soumission du formulaire précédent, la fenêtre de création de l'application key apparaît, générez cette clé comme ci-dessous :

The screenshot shows the 'OTAA keys' generation screen for device 'MKR1310'. The breadcrumb trail is 'Tenants / ChirpStack / Applications / Station météo / Devices / MKR1310'. The device EUI is 'a8610a3233428e03'. The 'OTAA keys' tab is selected, showing an 'Application key' field with the value '04aa6d43e01d0c9f6af05c44b524ed54'. A 'Submit' button is present. A callout box with the text 'Générer l'AppKey' has an arrow pointing to the 'C' (copy) icon next to the application key field.

5.4. Copiez cette clé dans le presse-papier et validez la création du device avec submit .

5.5. Depuis l'IDE Arduino, choisissez la connexion via OTAA (1), recopier l'app EUI, l'app key copiée précédemment dans le presse-papier.



5.6. On peut vérifier les requêtes de connexion OTAA sur l'onglet LoRaWAN frames ;

Dashboard

Configuration

OTAA keys

Activation

Queue

Events

LoRaWAN frames

Download

2024-02-14 09:33:19

UnconfirmedDataDown

DevAddr: 005791db

DevEUI: a8610a3233428e03

Gateway ID: c0ee40ffff2a3fed

2024-02-14 09:33:19

ConfirmedDataUp

DevAddr: 005791db

DevEUI: a8610a3233428e03

2024-02-14 09:33:06

JoinAccept

DevEUI: a8610a3233428e03

Gateway ID: c0ee40ffff2a3fed


2024-02-14 09:33:06

JoinRequest

DevEUI: a8610a3233428e03

5.7. Vérifiez le message reçu dans l'onglet Events.

Dashboard Configuration OTAA keys Activation Queue **Events** LoRaWAN frames

2024-02-14 10:24:17  DR: 0 Data: 48654c6f524120776f726c6421 FCnt: 0 FPort: 3

```
deduplicationId: "d5bce951-0798-46d5-8aa0-b6b6b2366e9f"
time: "2024-02-14T09:24:17.086452021+00:00"
▼ deviceInfo: {} 10 keys
  tenantId: "52f14cd4-c6f1-4fbd-8f87-4025e1d49242"
  tenantName: "ChirpStack"
  applicationId: "a24ec754-0d17-45dd-b416-61e35aacd628"
  applicationName: "Station météo"
  deviceProfileId: "c9e620d9-23ef-4bfa-90e6-548592b027cc"
  deviceProfileName: "Arduino MKR WAN 1310"
  deviceName: "MKR1310"
  devEui: "a8610a3233428e03"
  deviceClassEnabled: "CLASS_A"
  tags: {} 0 keys
  devAddr: "00c07f42"
  adr: true
  dr: 0
  fCnt: 0
  fPort: 3
  confirmed: true
  data: "SGVMb1JBIHdvcmxklQ=="
▼ object: {} 1 key
  ledState: null
▼ rxInfo: [] 1 item
▼ 0: {} 10 keys
  gatewayId: "c0ee40ffff2a3fed"
  uplinkId: 38391
  nsTime: "2024-02-14T09:24:16.865183285+00:00"
  rssi: -49
  snr: 8.2
  channel: 7
  location: {} 0 keys
  context: "FShuJA=="
▼ metadata: {} 2 keys
  region_config_id: "eu868"
  region_common_name: "EU868"
  crcStatus: "CRC_OK"
▼ txInfo: {} 2 keys
```

C'est du Base64.
Converti en caractère, on
reçoit : *HeLoRA world!*

6. RECEVOIR DES DONNÉES SUR LE SERVEUR CHIRPSTACK

6.1. 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.

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



6.3. Saisir un message à envoyer.

Output Serial Monitor x

Hello Lora

Your module version is: ARD-078 1.2.3
Your device EUI is: a8610a35301e8909

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

Output Serial Monitor x

Message (Enter to send message to 'Arduino MKR WAN 1310' on '/dev/cu.usbmodem14401')

Your module version is: ARD-078 1.2.3
Your device EUI is: a8610a35301e8909

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

Sending: Hello Lora - 48 65 6C 6C 6F 20 4C 6F 72 61
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)

6.4. Dans l'onglet Events, on constate la réception du message

Dashboard Configuration OTAA keys Activation Queue **Events** LoRaWAN frames

2024-02-14 10:47:07 up DR: 0 Data: 48656c6c6f204c6f7261 FCnt: 0 FPort: 2

fPort: 2
confirmed: true
data: "SGVsbG8gTG9yYQ=="

Message en base64 qui
donne Hello Lora

7. TRANSMETTRE DES DONNÉES VERS LE SERVEUR CHIRPSTACK

7.1. On programme l'envoi de données depuis l'onglet Queue.

Dashboard Configuration OTAA keys Activation **Queue** Events LoRaWAN frames

Enqueue

Confirmed: ☐ FPort: 1 Is encrypted: ☐

HEX BASE64 JSON

AA

Enqueue

7.2. On peut vérifier la programmation de l'envoi du message ci-dessous :

ID	Pending	Encrypted	Frame-counter	Confirmed	FPort	Data (HEX)
3fbd6778-c507-41f1-b745-57307396b775	no	no		no	1	aa

7.3. La transmission sera déclenchée lorsque le serveur aura reçu un message, on transmet donc depuis le device arduino.

Après 10 secondes, le message est reçu sur le device.

```
Message (Enter to send message to 'Arduino MKR WAN 1310' on '/dev
```

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

Sending: test - 74 65 73 74
Message sent correctly!
Received: AA
```

8. UPLINK ET DOWNLINK VIA MQTT

UPLINK :

8.1. Depuis le terminal d'un PC, générez la commande `mqtt subscribe` avec `mosquitto`. Elle est de la forme :

```
mosquitto_sub -h ip_du_serveur_lorawan -t  
"application/id_application/device/dev_eui/event/up"
```

example :

```
mosquitto_sub -h 192.168.1.29 -t "application/a24ec754-0d17-45dd-b416-61e35aacd628/device/a8610a3233428e03/event/up"
```

8.2. Depuis le device Arduino, générez un flux uplink en émettant un message.

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

Sending: Hello Lora - 48 65 6C 6C 6F 20 4C 6F 72 61
Message sent correctly!
```

8.3. Vérifiez le message reçu, il ressemble au message suivant :

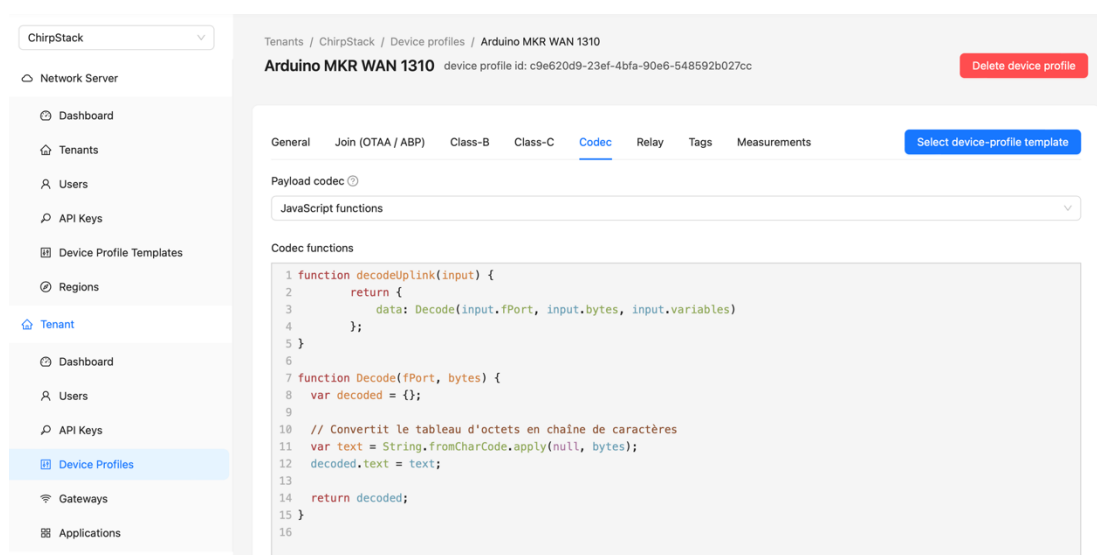
```
4fbd-8f87-4025e1d49242", "tenantName": "ChirpStack", "applicationId": "a24ec754-0d17-45dd-b416-61e35aacd628", "applicationName": "Station météo", "deviceProfileId": "c9e620d9-23ef-4bfa-90e6-548592b027cc", "deviceProfileName": "Arduino MKR WAN 1310", "deviceName": "MKR1310", "devFui": "a8610a3233428e03", "deviceClassEnabled": "CLASS_A", "tags": {}, "devAddr": "00504025", "adr": true, "dr": 0, "fCnt": 0, "fPort": 2, "confirmed": true, "data": "SGVsbG8gTG9yYQ==", "rxInfo": [{"gatewayId": "c0ee40ffff2a3fed", "uplinkId": 48272, "nsTime": "2024-02-15T15:14:50.102679928+00:00", "rssi": -28, "snr": 9.8, "rfChain": 1, "location": {}, "context": "GKyjFA==", "metadata": {"region_config_id": "eu868", "region_common_name": "EU868"}, "crcStatus": "CRC_OK"}], "txInfo": {"frequency": 868100000, "modulation": {"lora": {"bandwidth": 125000, "spreadingFactor": 12, "codeRate": "CR_4_5"}}}}
```

Les données reçues sont en base64 : SGVsbG8gTG9yYQ== qui donne Hello Lora.

8.4. On peut décoder le message et le transférer au format json en ajoutant un script js comme ci-dessous :

Source :

<https://github.com/bouhenic/FormationIOT/blob/main/Tp2Chirpstack/DecodePayload.js>



8.5 Transmettez de nouveau un message depuis le device Arduino :

Sending: Hello Lora JS - 48 65 6C 6C 6F 20 4C 6F 72 61 20 4A 53
Message sent correctly!

8.5. Le message reçu est le suivant :

```
{ "deduplicationId": "f08a4c8e-d2e8-41c7-b3b2-16be5ccf80b2", "time": "2024-02-15T15:39:10.889674134+00:00", "deviceInfo": { "tenantId": "52f14cd4-c6f1-4fbd-8f87-4025e1d49242", "tenantName": "ChirpStack", "applicationId": "a24ec754-0d17-45dd-b416-61e35aacd628", "applicationName": "Station météo", "deviceProfileId": "c9e620d9-23ef-4bfa-90e6-548592b027cc", "deviceProfileName": "Arduino MKR WAN 1310", "deviceName": "MKR1310", "devFui": "a8610a3233428e03", "deviceClassEnabled": "CLASS_A", "tags": {}, "devAddr": "01ef35da", "adr": true, "dr": 0, "fCnt": 0, "fPort": 2, "confirmed": true, "data": "SGVsbG8gTG9yYSBKUw==", "object": { "text": "Hello Lora JS" }, "rxInfo": [{"gatewayId": "c0ee40ffff2a3fed", "uplinkId": 53028, "nsTime": "2024-02-15T15:39:10.669807126+00:00", "rssi": -25, "snr": 9.2, "channel": 7, "location": {}, "context": "b7vWLA==", "metadata": {"region_config_id": "eu868", "region_common_name": "EU868"}, "crcStatus": "CRC_OK"}], "txInfo": { "frequency": 867900000, "modulation": { "lora": { "bandwidth": 125000, "spreadingFactor": 12, "codeRate": "CR_4_5" } } } }
```

On voit maintenant les données en base 64 et au format json.

DOWNLINK :

- 8.6. Depuis le terminal d'un PC, publier un message vers le device. La commande mosquitto est de la forme :

```
mosquitto_pub -h IP -t "application/a24ec754-0d17-45dd-b416-61e35aacd628/device/a8610a3233428e03/command/down" -m '{"devEui": "a8610a3233428e03", "confirmed": false, "fPort": 10, "data": "AQ=="}'
```

Exemple :

```
mosquitto_pub -h 192.168.1.29 -t "application/a24ec754-0d17-45dd-b416-61e35aacd628/device/a8610a3233428e03/command/down" -m '{"devEui": "a8610a3233428e03", "confirmed": false, "fPort": 10, "data": "AQ=="}'
```