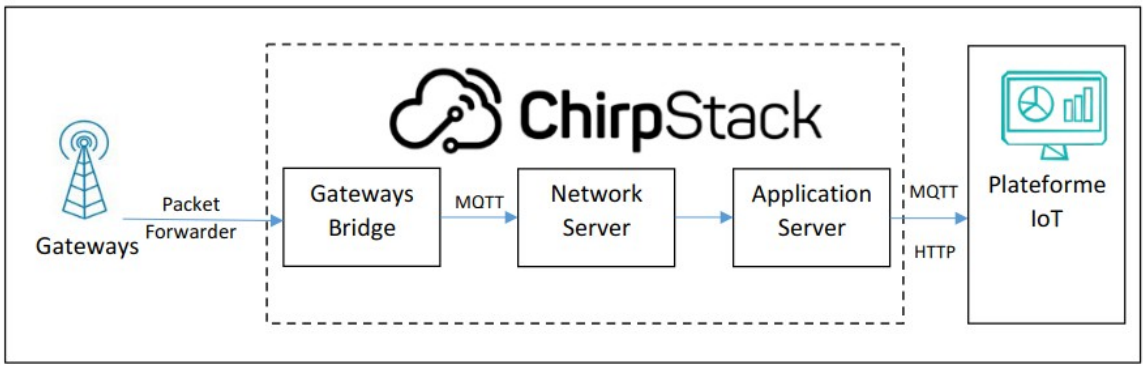
Serveur LoraWan ChirpStack

Sommaire :

I - Présentation

Pour pouvoir mettre en place cette version locale du serveur LoRaWAN (**https://www.chirpstack.io/project/guides/debian-ubuntu/**), il y a au moins 3 composants à installer, comme le montre l'architecture suivante :

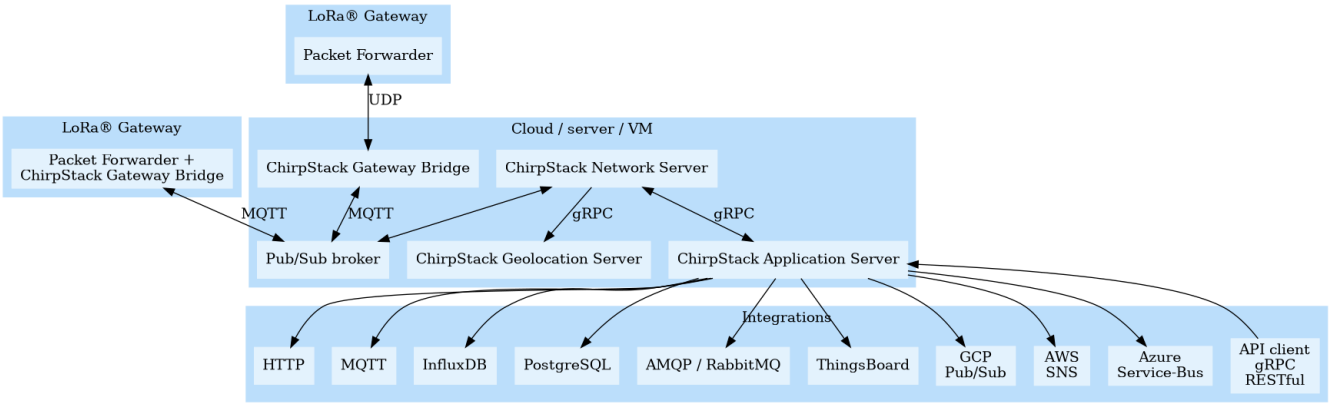


- « **ChirpStack Gateway Bridge** » : Ce service à la base du serveur va permettre de transformer nos paquets UDP (port 1700) provenant du « packet forwarder » en documents JSON. Ces fichiers seront envoyés, via MQTT par Mosquitto, directement sur le « ChirpStack Network Server » ;

- « **ChirpStack Network Server** » : C'est la composante qui représente le « network server » au sein du réseau. Il reçoit tous les documents provenant de la passerelle et réalise tout le traitement de ces informations ainsi que l'administration du réseau, par exemple quand un nouveau nœud veut rejoindre le réseau ou quand il faut vérifier si un nœud fait vraiment partie du réseau. C'est lui qui gère toutes les commandes MAC de manière générale. Si un message est jugé valide et conforme au protocole, il est alors transmis au « ChirpStack Application Server » ;

- « **ChirpStack Application Server** » : Ce service incarne le « application server » dans le réseau. En association avec le « ChirpStack Network Server », il offre la possibilité de gérer le réseau de manière interactive via une interface web. C'est sur cette interface que l'on crée des applications pour venir y ajouter des nœuds, choisir leur mode d'activation. Il est possible également d'avoir la liste de tous les paquets reçus pour un nœud en continu.

Une architecture plus complète montre la nécessité d'avoir aussi un broker MQTT pour relayer les paquets UDP récupérés par « **ChirpStack Gateway Bridge** » vers le « **ChirpStack Network Server** » :



- « **MQTT broker** » : Un autre service également essentiel est « **Mosquitto** » qui est un **broker MQTT**. Il implémente le protocole **MQTT** pour transmettre de l'information. Ce protocole-ci, basé sur **TCP/IP**, est un protocole de messagerie construit sur le modèle de **publier-s'abonner**. En d'autres termes, une entité a la possibilité de créer un sujet ou « **topic** » et de publier toutes sortes d'informations sur celui-ci tandis qu'une ou plusieurs autre(s) entité(s) ont l'opportunité de s'abonner à ce sujet et de recevoir toutes les informations qui en émanent. Ce principe est important puisqu'il est utilisé pour l'implémentation du serveur.

II - Installation sur Debian 10 « Buster »

II.1. Installation des différentes dépendances

On installe les paquets suivants :

apt install mosquitto mosquitto-clients redis-server redis-tools postgresql

II.2. Configuration de postgresql

On se connecte à la SBDD **postgresql** puis on cré les différentes bases et utilisateurs :

```
su postgres
psql
-- set up the users and the passwords
```

```
-- (note that it is important to use single quotes and a semicolon at the end!)
create role chirpstack_as with login password 'dbpassword';
create role chirpstack_ns with login password 'dbpassword';
-- create the database for the servers
create database chirpstack_as with owner chirpstack_as;
create database chirpstack_ns with owner chirpstack_ns;
-- change to the ChirpStack Application Server database
\c chirpstack_as
-- enable the pq_trgm and hstore extensions
-- (this is needed to facilitate the search feature)
create extension pg_trgm;
-- (this is needed to store additional k/v meta-data)
create extension hstore;
-- exit psql
\q
exit
```

II.3. Configuration des dépôts

On installe les paquets suivants :

```
apt install apt-transport-https dirmngr
```

Puis on modifie la liste des paquets et on met à jour cette liste :

```
apt-key adv --keyserver keyserver.ubuntu.com --recv-keys
1CE2AFD36DBCCA00
echo "deb https://artifacts.chirpstack.io/packages/3.x/deb stable main" | tee
/etc/apt/sources.list.d/chirpstack.list
apt update
```

II.4. Installation et lancement de chirpstack-gateway-bridge

On installe le paquet suivant :

```
apt install chirpstack-gateway-bridge
```

La configuration de ce paquet se fait dans le fichier **/etc/chirpstack-gateway-bridge/chirpstack-gateway-bridge.toml**.

On peut maintenant lancer le service correspondant :

```
systemctl start chirpstack-gateway-bridge
```

Activer ce service au boot de la machine :

```
systemctl enable chirpstack-gateway-bridge
```

Pour savoir si le service est bien lancé, on peut visualiser les logs :

```
journalctl -f -n 100 -u chirpstack-gateway-bridge
```

II.5. Installation, configuration et lancement de chirpstack-network-server

On installe le paquet suivant :

```
apt install chirpstack-network-server
```

La configuration de ce paquet se fait dans le fichier **/etc/chirpstack-network-server/chirpstack-network-server.toml**. Ce fichier est modifié de la façon suivante :

Dans la section [postgresql] :

dsn="postgres://chirpstack_ns:dbpassword@localhost/chirpstack_ns?sslmode=disable"

On peut maintenant lancer le service correspondant :
systemctl start chirpstack-network-server

Activer ce service au boot de la machine :
systemctl enable chirpstack-network-server

Pour savoir si le service est bien lancé, on peut visualiser les logs :
journalctl -f -n 100 -u chirpstack-network-server

II.6. Installation, configuration et lancement de chirpstack-application-server

On installe le paquet suivant :
apt install chirpstack-application-server

La configuration de ce paquet se fait dans le fichier **/etc/chirpstack-application-server/chirpstack-application-server.toml**. Ce fichier est modifié de la façon suivante :

Dans la section [postgresql] :
dsn="postgres://chirpstack_as:dbpassword@localhost/chirpstack_as?sslmode=disable"

Dans la section [application_server.external_api] :
jwt_secret="12345678"

Dans la section [metrics] :
timezone="Europe/Paris"

Note : Il faut remplacer le contenu du champ **jwt_secret** avec un code secret quelconque ou généré à l'aide de la commande **openssl rand -base64 32**.

On peut maintenant lancer le service correspondant :
systemctl start chirpstack-application-server

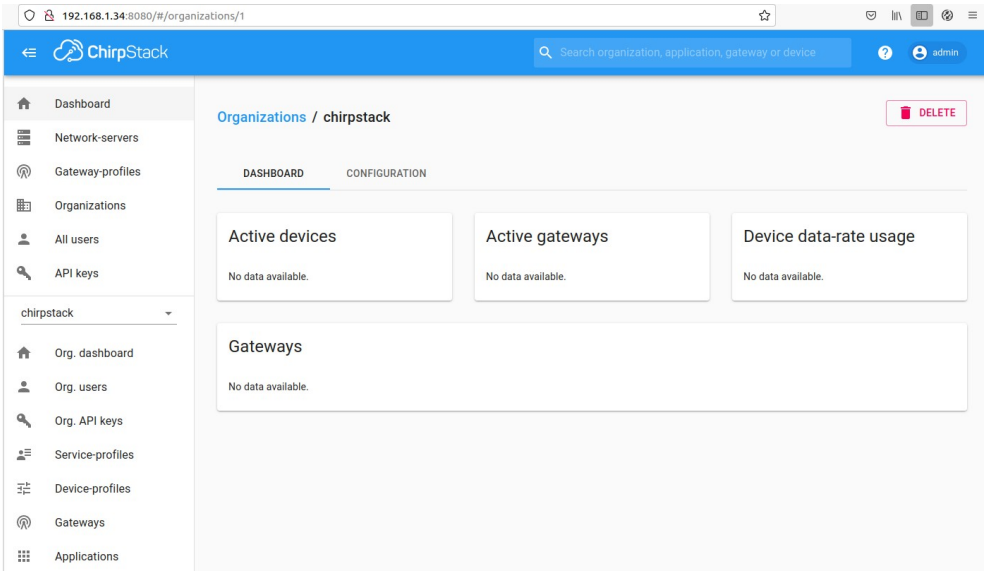
Activer ce service au boot de la machine :
systemctl enable chirpstack-application-server

Pour savoir si le service est bien lancé, on peut visualiser les logs :
journalctl -f -n 100 -u chirpstack-application-server

II.7. Test du fonctionnement

Ouvrir un navigateur avec l'URL suivante :

http ://ip_serveur:8080
login : admin / password : admin



Le port **8080** est celui configuré dans le fichier **/etc/chirpstack-application-server.toml**. Pour activer **TLS** (https) voir **https://github.com/brocaar/loraserver-certificates**.

Il faut commencer par créer un **network-servers** : **localhost:8000** et la passerelle doit envoyer les segments **UDP** sur le port **1700**.

II.8. Authentification MQTT

Afin d'authentifier les accès **MQTT**, nous allons créer un utilisateur nommé **lora** ayant le mot de passe **lora**. Pour cela il faut réaliser les actions suivantes :

Création de l'utilisateur sur Mosquitto :

mosquitto_passwd -c /etc/mosquitto/passwd lora

Rajouter dans le fichier **/etc/mosquitto/mosquitto.conf** :

allow_anonymous false
password_file /etc/mosquitto/passwd

Relancer le service mosquitto :

service mosquitto restart

Note : Pour supprimer un utilisateur sur Mosquitto : **mosquitto_passwd -D /etc/mosquitto/passwd user**.

Il faut modifier le fichier **/etc/chirpstack-gateway-bridge/chirpstack-gateway-bridge.toml** :

username= « lora »
password= « lora »

Puis relancer le service correspondant :

systemctl restart chirpstack-gateway-bridge

Pour savoir si le service est bien lancé, on peut visualiser les logs :

journalctl -f -n 100 -u chirpstack-gateway-bridge

Il faut modifier le fichier **/etc/chirpstack-network-server/chirpstack-network-server.toml** :

username= « lora »
password= « lora »

Puis relancer le service correspondant :
systemctl restart chirpstack-network-server

Pour savoir si le service est bien lancé, on peut visualiser les logs :
journalctl -f -n 100 -u chirpstack-network-server

Il faut modifier le fichier **/etc/chirpstack-application-server.toml** :
username= « lora »
password= « lora »

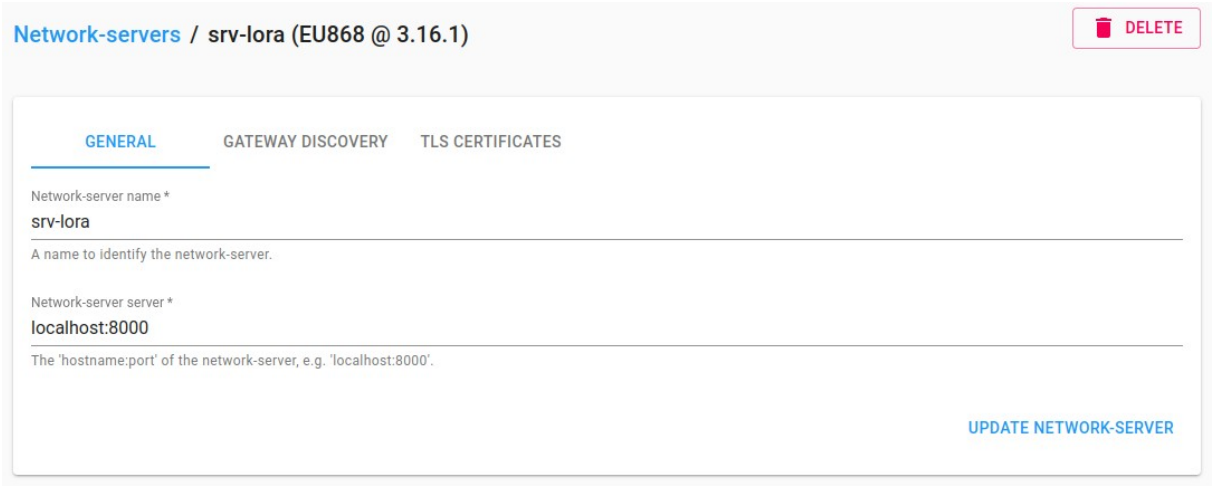
Puis relancer le service correspondant :
systemctl restart chirpstack-application-server

Pour savoir si le service est bien lancé, on peut visualiser les logs :
journalctl -f -n 100 -u chirpstack-application-server

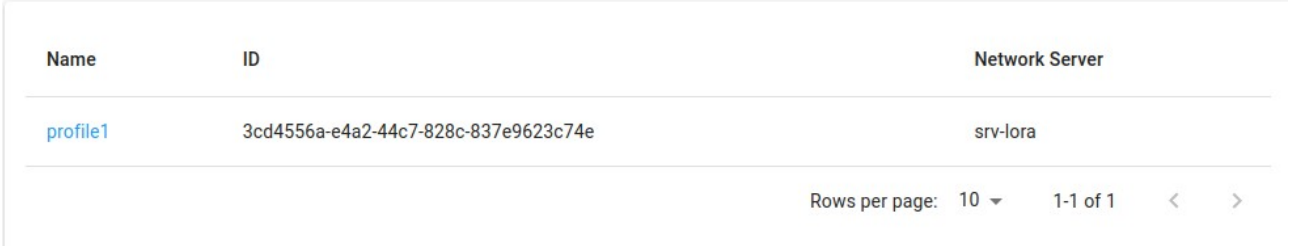
III - Utilisation de Chirpstack Application Server

III.1. Configuration initiale

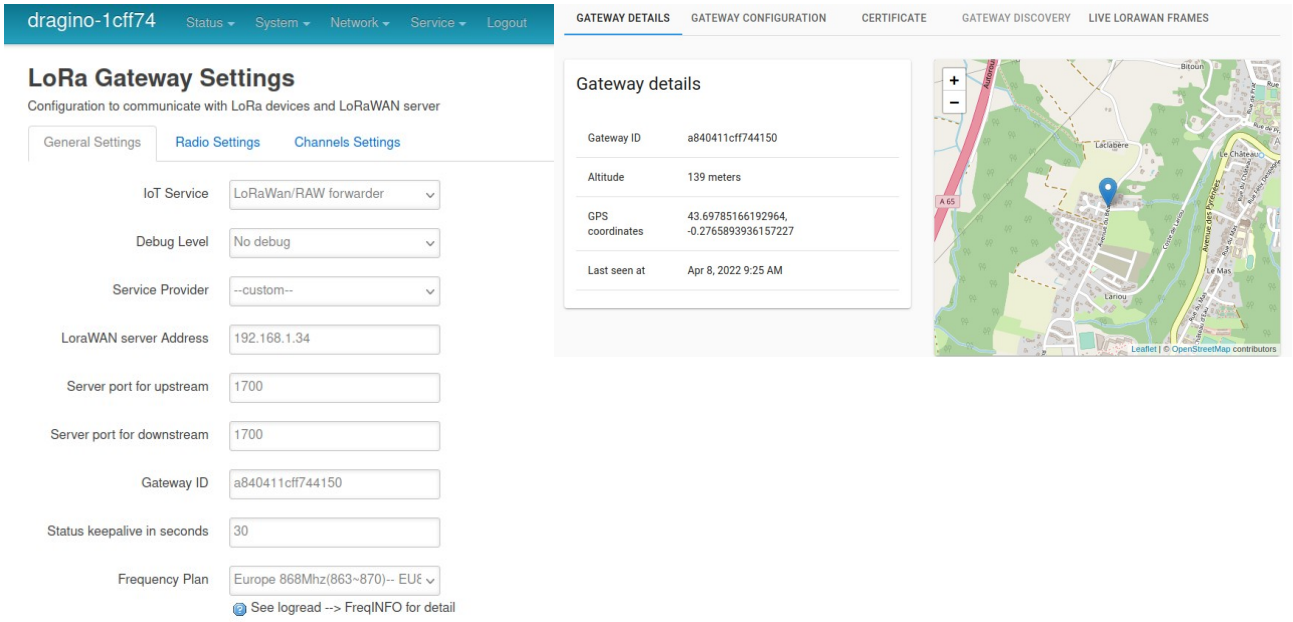
Il faut commencer par créer un **network-servers** : **localhost:8000** :



Puis un **service profiles** associé au **network-servers** précédent :

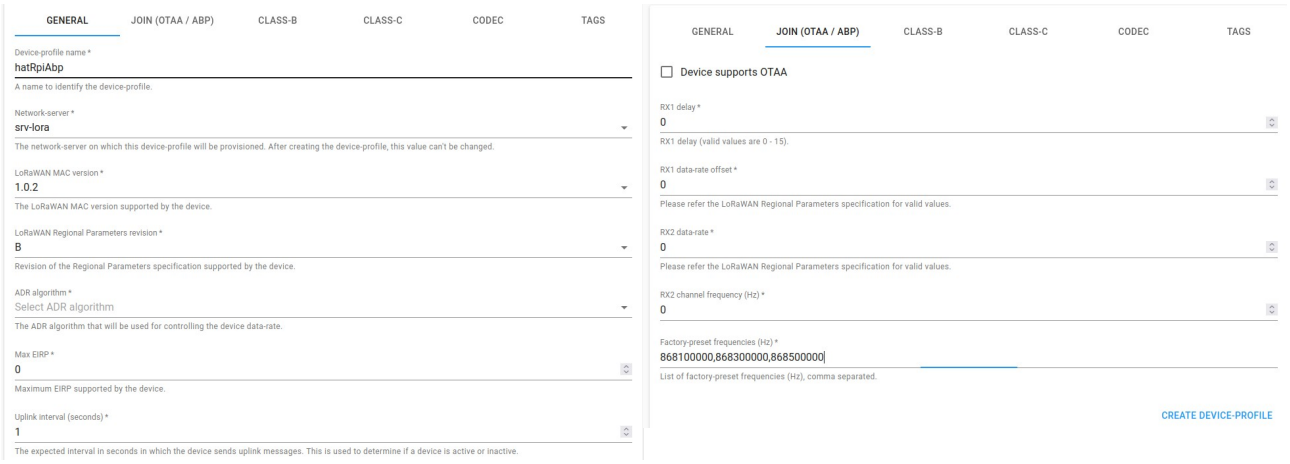


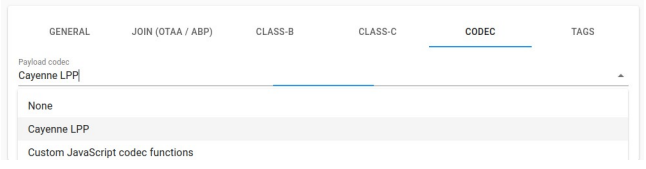
Puis il faut déclarer une ou plusieurs gateway. Nous allons utiliser une passerelle **Dragino LG308** comportant l'**ID** ci-dessous :



III.2. Déclaration d'un device ABP

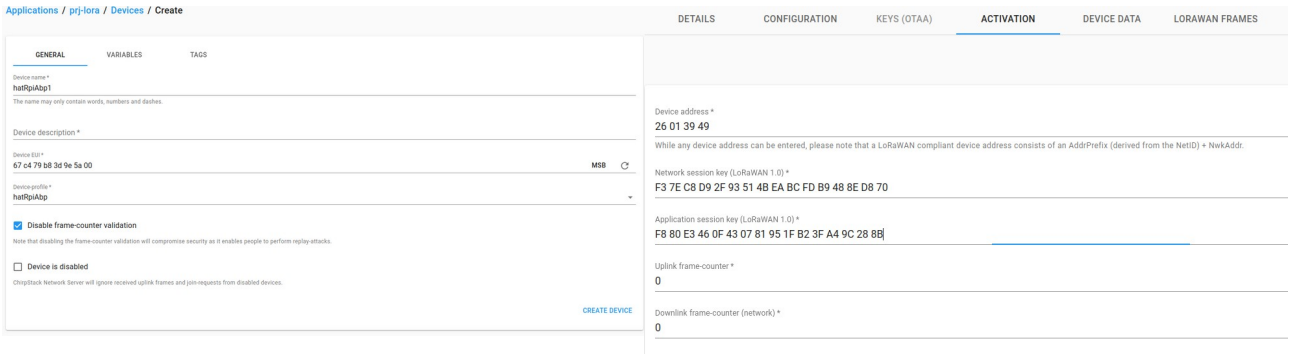
Il faut d'abord créer un **device profiles** de type **ABP** :





Remarque : On pourra choisir l'encodage des données (**Codec**) de type **Cayenne LPP** ou **Custom**.

On peut maintenant créer une application et un device associé :

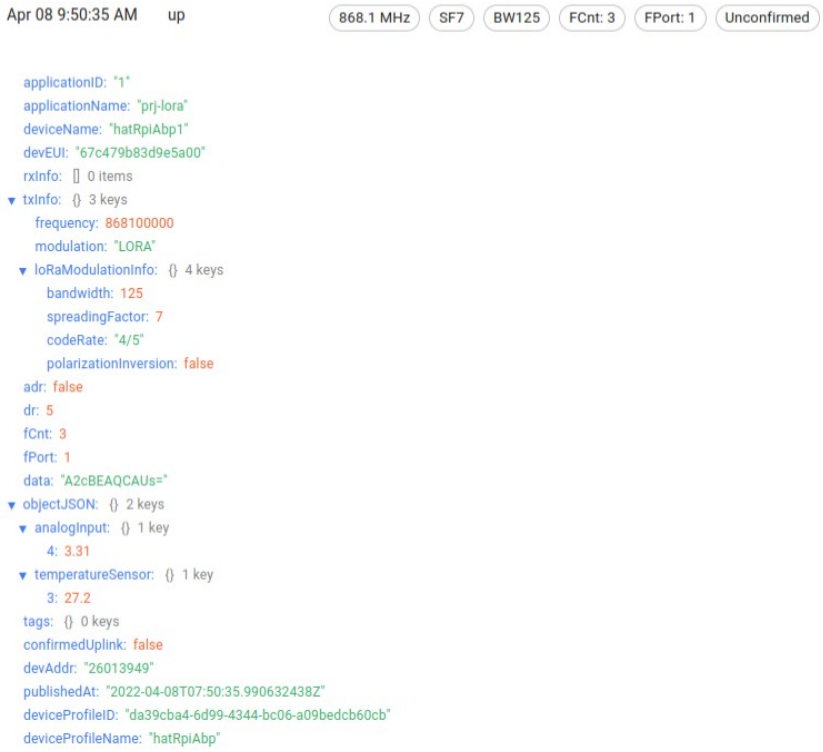


III.3. Déclaration d'un device OTAA

Même démarche en cochant **OTAA** lors de la création du **device profiles**.

Remarque : Il faut renseigner la champ **Network key** qui correspond au **Application key**.

III.4. Visualisation des trames

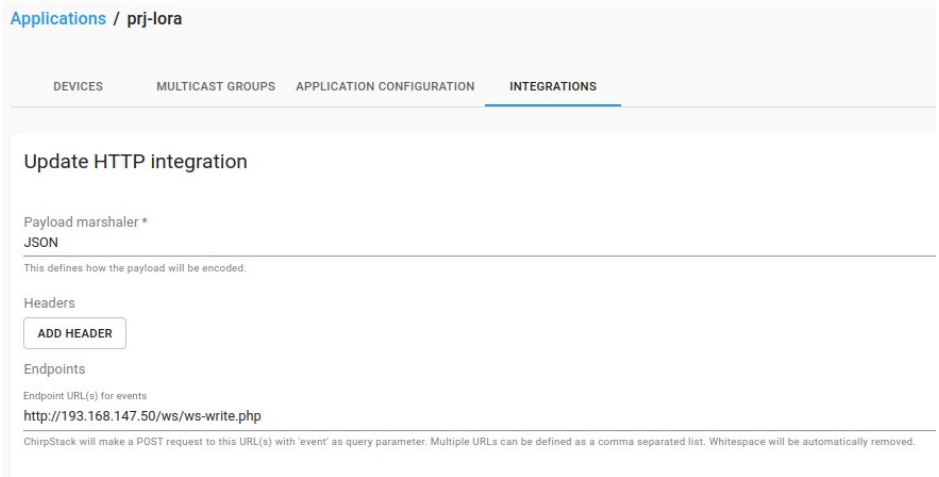


IV - Communication HTTP

IV.1. Intégration HTTP - Flux Uplink

L'intégration **HTTP** permet d'envoyer des données à un **Endpoint** au travers de **HTTP**. A chaque message reçu sur l'application, une requête **HTTP** est générée à destination de l'URL sélectionnée. Il faut créer une **intégration http** (méthode **POST** par défaut), choisir le format

JSON et saisir l'URL du serveur vers lequel les données sont envoyées **uplink** :



Le format des données récupérées est le suivant :

{"applicationID":"1","applicationName":"prj-lora","deviceName":"HatRpiAbp-26013949","devEUI":"Z8R5uD2eWgA=","rxInfo":[],"txInfo":{"frequency":867300000,"modulation":"LORA","loRaModulationInfo":{"bandwidth":125,"spreadingFactor":7,"codeRate":"4/5","polarizationInversion":false}},"adr":false,"dr":5,"fCnt":287,"fPort":1,"data":"A2cBEAQCAUs=","objectJSON":"{\"analogInput\":{\"4\":3.31},\"temperatureSensor\":{\"3\":27.2}}","tags":{},"confirmedUplink":false,"devAddr":"JgE5SQ==","publishedAt":"2022-04-08T12:34:41.329812245Z","deviceProfileID":"c82a74b2-af87-4993-b597-86521afd12bd","deviceProfileName":"HatRpiAbp"}

On peut récupérer le **corps** de la requête (tout le flux json) à l'aide d'un script **php** appelé par l'URL sélectionnée, en utilisant la commande suivante :

$json = file_get_contents('php://input');

Par exemple pour stocker dans un fichier le flux json reçu :

```
<?php
header('Cache-Control: no-cache, must-revalidate');
header('Content-type: application/json');
date_default_timezone_set('Europe/Paris');
if($_SERVER['REQUEST_METHOD'] != 'POST')
        {
        header("HTTP/1.1 405 NOT ALLOWED");
        }
else {
        header("HTTP/1.1 200 OK");
        $json = file_get_contents('php://input');
        $f = fopen('/capteurs/chirpstack.txt', 'w+');
        if($f==false) echo "Erreur Ouverture Fichier";
        else {
                if (!fwrite($f, $json)) echo "Erreur Ecriture Fichier";
                fclose($f);
                }
   }
?>
```

Par exemple pour stocker dans un fichier l'objet json :

```
<?php
header('Cache-Control: no-cache, must-revalidate');
header('Content-type: application/json');
date_default_timezone_set('Europe/Paris');
if($_SERVER['REQUEST_METHOD'] != 'POST')
        {
        header("HTTP/1.1 405 NOT ALLOWED");
        }
else {
        header("HTTP/1.1 200 OK");
        $json = file_get_contents('php://input');
        $obj = json_decode($json, true) ;
        $json = $obj['objectJSON'];
        $f = fopen('/capteurs/chirpstack.txt', 'w+');
        if($f==false) echo "Erreur Ouverture Fichier";
        else {
                if (!fwrite($f, $json)) echo "Erreur Ecriture Fichier";
                fclose($f);
                }
   }
?>
```

Le format des données récupérées est le suivant :

{"analogInput":{"4":3.31},"temperatureSensor":{"3":27.2}}

IV.2. Flux Downlink

On peut envoyer un flux **Downlink** à un **device** via l'**API HTTP**. La commande suivante met en file d'attente le payload **"AQ=="** codé en base 64 (**0x01**) pour le device **DevEUI**. Il faudra générer depuis le menu **« API Keys »** une clé et on remplacera **<API TOKEN>** avec celle-ci :

curl -X POST --header 'Content-Type: application/json' --header 'Accept:

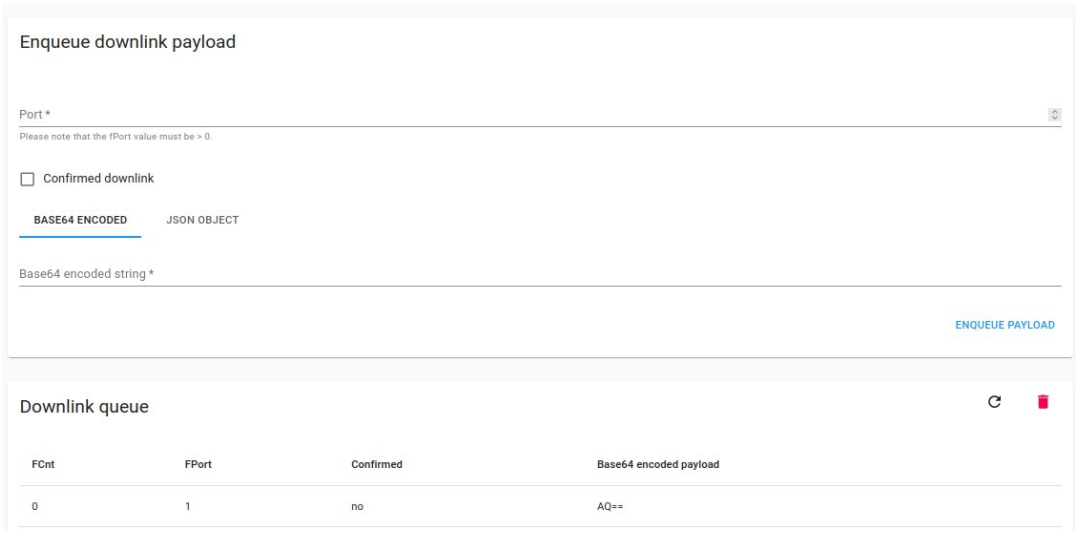
```
application/json' --header 'Grpc-Metadata-Authorization: Bearer <API TOKEN>' -d '{ \
  "deviceQueueItem": { \
    "confirmed": false, \
    "data": "AQ==", \
    "fPort": 1 \
  } \
}' 'http://localhost:8080/api/devices/<DevEUI>/queue'
```

Par exemple, pour le serveur **172.17.1.4** :

```
curl -X POST --header 'Content-Type: application/json' --header 'Accept: application/json' --header 'Grpc-Metadata-Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhcGIfa2V5X2lkIjoiMzVkOTUyYWMtMGUyMi00ODExLThhYzctMzM2OTlkMDIzMDk3IiwiYXVkIjoiYXMiLCJpc3MiOiJhcyIsIm5iZiI6MTY0OTg1NTQwMCwic3ViIjoiYXBpX2tleSJ9.yDToebXOVuKZVdM8t0F0IKdl2aoPKBYe4BtvMoO7y5c' -d '{"deviceQueueItem": {"confirmed": false,"data": "AQ==","fPort": 1}}' 'http://172.17.1.4:8080/api/devices/67c479b83d9e5a00/queue'
```

Réponse : {"fCnt":52}

On peut voir sur la page principale du **device** que le flux **Uplink** est mis en file d'attente :



V - Communication avec le broker MQTT

Le serveur **ChirpStack** joue le rôle de **broker MQTT**. Pour se connecter au **broker** de **ChirpStack**, il faudra fournir les informations suivantes :

- Adresse du broker : **ip_serveur:1883** ;
- Username : **lora** ;
- Password : **lora**.

Les événements seront reçus à l'aide d'un **topic** ayant la forme suivante :

application/[ApplicationID]/device/[DevEUI]/event/[EventType]

Les flux **Downlink** seront envoyés à l'aide d'un **topic** ayant la forme suivante :

application/[ApplicationID]/device/[DevEUI]/command/down

On peut interroger le broker présent sur le serveur Lorawan avec un client MQTT. Sur un poste Linux, on peut utiliser un client en ligne de commande. Pour cela, il faut installer le paquet **mosquitto-clients** :

apt-get install mosquitto-clients

Pour récupérer toutes les informations reçues par la passerelle :

mosquitto_sub -u lora -P lora -h ip_serveur -t "gateway/+/rx" -v

Pour **s'abonner** à tous les messages de l'application dont l'**applicationID** vaut **1** :

mosquitto_sub -u lora -P lora -h ip_serveur -t "application/1/#" -v

application/1/device/67c479b83d9e5a00/event/up
{"applicationID":"1","applicationName":"prj-
lora","deviceName":"hatRpiAbp1","deviceProfileName":"hatRpiAbp","devicePr
ofileID":"da39cba4-6d99-4344-bc06-
a09bedcb60cb","devEUI":"67c479b83d9e5a00","txInfo":
{"frequency":867100000,"dr":5},"adr":false,"fCnt":9,"fPort":1,"data":"A2cBEA
QCAUs=","object":{"analogInput":{"4":3.31},"temperatureSensor":
{"3":27.2}}}

Par exemple pour **s'abonner** à tous les messages de type **Uplink** du device ayant pour EUI **00bb063b41afa30e** et concernant l'application ayant pour **applicationID** la valeur **1** :

mosquitto_sub -h ip_serveur -t
'application/1/device/00bb063b41afa30e/event/up' -u lora -P lora

Par exemple pour **publier** un message de type **Downlink** au device ayant pour EUI **00bb063b41afa30e** et concernant l'application ayant pour **applicationID** la valeur **1** :

mosquitto_pub -h ip_serveur -t
'application/1/device/00bb063b41afa30e/command/down' -u lora -P lora -m
'{"confirmed":false,"fPort":1,"data":"AQ=="}'