

IUT DE BÉTHUNE - UNIVERSITÉ D'ARTOIS  
LICENCE PRO RÉSEAUX INFORMATIQUES, MOBILITÉ, SÉCURITÉ



28 février 2021  
Projet tutoré M41

## Surveillance des points de mutualisations

HELETA Alexandre, FEUILLATRE Romain, DEVRIENDT Elie



# Table des matières

<b>Table des matières</b>	<b>1</b>	5.2 Chirpstack . . . . .	18
<b>1 Remerciements</b>	<b>2</b>	5.2.1 Configuration de chirpstack . .	18
<b>2 Résumé</b>	<b>3</b>	5.2.2 Configuration de gateway . .	18
<b>3 Introduction</b>	<b>4</b>	5.2.3 Configuration du capteur . . .	19
3.1 Cahier des charges . . . . .	5	5.2.4 Analyse du capteur . . . . .	20
3.2 Diagramme de Gantt . . . . .	5	5.2.5 Ecriture du code Javascript . .	22
<b>4 Présentation du Projet</b>	<b>6</b>	5.2.6 Analyse du résultat . . . . .	25
4.1 Présentation du matériel . . . . .	6	5.3 Supervision . . . . .	26
4.1.1 Contexte du projet . . . . .	6	5.3.1 Installation de InfluxDB . . .	26
4.1.2 Capteur Adeunis . . . . .	7	5.3.2 Configuration de base de In-	27
4.1.3 Paserelle LoraWAN . . . . .	8	fluxDB . . . . .	27
4.2 Présentation des logiciels . . . . .	9	5.3.3 Intégration de InfluxDB dans	28
4.2.1 Chirpstack.io . . . . .	9	le serveur Chirpstack . . . . .	28
4.2.2 IoTConfigurator . . . . .	10	5.3.4 Vérification de la communi-	29
4.2.3 PostgreSQL . . . . .	10	cation entre ChirpStack et	29
4.2.4 Grafana . . . . .	10	InfluxDB . . . . .	29
4.2.5 InfluxDB . . . . .	11	5.3.5 Instalation de Grafana . . . .	29
4.2.6 PuTTY . . . . .	11	5.3.6 Configuration de base de gra-	30
<b>5 Réalisation du projet</b>	<b>12</b>	fana . . . . .	30
5.1 Gateway . . . . .	12	5.3.7 Création du panels . . . . .	31
		<b>6 Bilan</b>	<b>33</b>
		6.1 Conclusion . . . . .	33
		<b>Glossaire</b>	<b>34</b>

# Partie 1

## Remerciements

L'accomplissement de notre projet a pu se réaliser grâce à différentes personnes. Nous tenons tout d'abord à remercier le corps enseignant, et plus particulièrement Monsieur Provolo et Monsieur Delattre qui nous ont permis de travailler sur ce projet et nous ont aidés tout au long du projet, ainsi que Madame Réant qui nous a aidés pour l'élaboration du rapport et de la soutenance.

# Partie 2

## Résumé

Depuis notre DUT Réseaux et Télécommunications, chacun des membres du projet avait déjà réalisé certains TPs sur la technologie LoRaWAN et était très intéressé par celui-ci mais n'avait pas eu l'occasion de réaliser de projet en lien. Étant également très intéressés par le monde des télécommunications et l'importance des points de mutualisation (PM), qui est un maillon de la chaîne vitale au réseau de distribution des opérateurs, nous étions assez préoccupés du fait que les opérateurs de télécommunications font appel à des sous-traitants afin d'assurer le raccordement des clients aux points de mutualisation et que ceci amène des problèmes car les PM restent ouverts très fréquemment. Actuellement, uniquement un système de signalement existe sur ce site : <https://dommages-reseaux.orange.fr/> et cela n'est clairement pas suffisant et met parfois plusieurs jours à solutionner le problème.

Le but de notre projet est de sécuriser ces PM à l'aide de la technologie LoRaWAN afin d'établir une traçabilité entre les sous-traitants et leurs accès aux PM. Nous allons fournir aux opérateurs la possibilité de voir quand quelqu'un a eu accès à un PM. et de signaler tout problème relatif à l'ouverture/fermeture le tout très rapidement .

# Partie 3

## Introduction

Ce rapport a été réalisé par trois étudiants de Licence professionnelle Réseaux Informatiques Mobilité Sécurité, dans le cadre d'un projet tutoré au lycée Malraux à Béthune.

Il s'agit de la surveillance des points de mutualisation via la technologie LoraWAN où l'on dispose de capteurs et d'une gateway LoRaWAN.

Ce projet avait pour objectif de réaliser un réseau de petite échelle afin de réaliser la surveillance d'un ou plusieurs PM. De ce fait, nous allons vous présenter la configuration de cette architecture LoraWAN et l'acheminement du projet jusqu'à la solution finale.

Dans ce projet, nous avons eu l'appui de Monsieur Provolo et Monsieur Delattre tout au long du projet, pour nous aider sur la partie technique.

Nous présenterons dans un premier temps notre cahier des charges et un diagramme de Gantt.

Dans une seconde partie, nous allons faire une présentation du matériel mis à disposition, puis nous présenterons les logiciels utilisés durant le projet.

Dans une troisième partie, nous allons montrer comment nous avons réalisé le projet, avec les tâches réalisées tout au long du projet, les solutions envisagées et retenues en termes de matériel et logiciel, et pour finir vous présenter le résultat.

Pour finir, on fera un bilan sur l'ensemble du projet.

## 3.1 Cahier des charges

- Contexte et présentation du projet
  - Contexte
    - Les opérateurs de télécommunications ont un besoin pour la sécurisation des points de mutualisation
  - Objectif
    - Mise en place d'un réseau LoRaWAN avec gateway et capteur dans chaque PM
- Besoins et contraintes liés au projet
  - Besoins fonctionnels
    - Mise en place de la gateway LoRaWAN
    - Communication entre la gateway et le capteur
    - Exploitation des données sous forme de trame
  - Contraintes du décodage de trame
    - traitement des données
  - Exigences
    - Etude de la documentation du capteur
    - Etude de la gateway LoRaWAN / installation et configuration
- Résultats attendus
  - Les données envoyées par le capteur doivent être claires et compréhensibles
  - Visualisation des informations sur une application dédiée
  - Être capable de déterminer si la porte est ouverte ou fermée et à quelle heure

## 3.2 Diagramme de Gantt

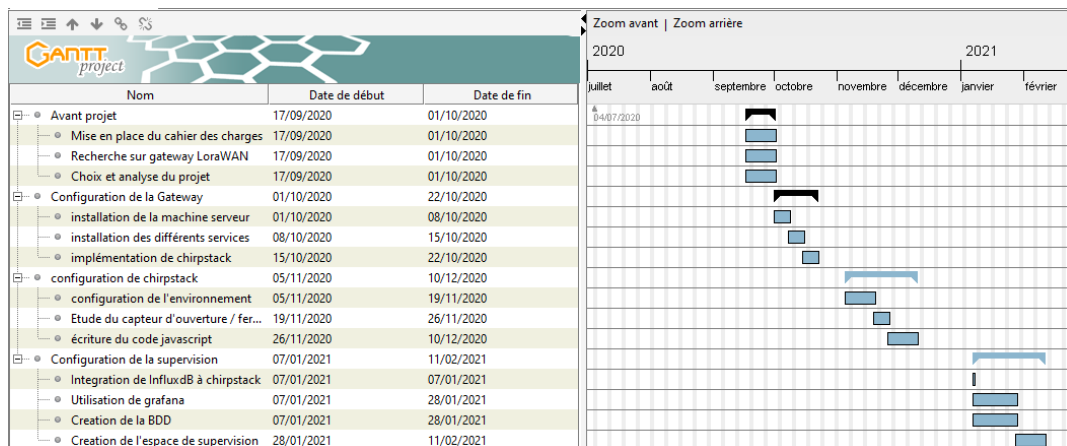


FIGURE 3.1 – Diagramme de Gantt

# Partie 4

## Présentation du Projet

### 4.1 Présentation du matériel

#### 4.1.1 Contexte du projet

La fibre est en train d'être câblée et installée partout dans les Hauts-de-France. Afin de pouvoir répondre aux futurs besoins des utilisateurs, les opérateurs de télécommunications font appel à des sous-traitants afin d'assurer le raccordement des clients aux points de mutualisation (PM). Ils ont cependant remarqué qu'énormément de PM rencontraient des problèmes récurrents : des portes restaient ouvertes, ce qui laissaient les câbles à la vue de tous. Des clients se retrouvent donc sans connexion Internet sans raison apparente, possible vandalisme et/ou négligence de la part des techniciens.

Le but de notre projet est d'empêcher les futures dégradations de ces PM en établissant une traçabilité entre les sous-traitants et leurs accès aux PM.

Nous allons fournir aux opérateurs la possibilité de voir chaque ouverture/fermeture d'un PM qui aura été équipé, et l'accès à une vue d'ensemble avec l'état d'ouverture du PM en temps réel.

Notre projet consiste donc à déployer une architecture LoRaWAN (réseau radio basse fréquence qui a pour avantage d'avoir une longue portée) . Un réseau LoRaWAN est composé de capteurs qui vont envoyer les informations à transmettre (comme des mesures, ou l'état de la batterie...) à la passerelle LoRaWAN. Celle-ci enverra l'information au serveur LoRaWAN qui analysera les données et les rendra lisibles avant de l'envoyer sur le serveur client.

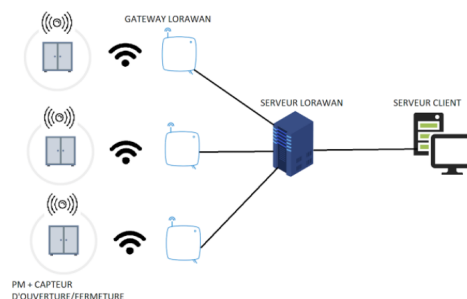


FIGURE 4.1 – Architecture LoraWAN

Avant même le déploiement du matériel, il faudra avoir configuré le matériel. Il faudra que les appareils soient sur le même réseau afin qu'ils puissent communiquer ensemble. Par exemple, la passerelle sera à ajouter dans notre réseau via le serveur ChirpStack pour qu'elle puisse transmettre les trames des capteurs. A la suite de quoi nous allons exploiter ces données qui seront sous forme de trame, et que nous allons devoir décoder. Nous allons donc sélectionner les données qui vont nous permettre d'identifier l'ouverture et la fermeture de la porte du point de mutualisation ainsi que les autres données jugées intéressantes. Il faudra un traitement des données, qui permettra d'envoyer celles-ci depuis le serveur ChirpStack vers une base de données. Afin de visualiser ces données, Grafana enverra des requêtes à la base de données afin de remplir les graphiques configurés. Par rapport au point de mutualisation voulu, l'application pourra alors voir si la porte est ouverte ou fermée. Un historique des ouvertures/fermetures avec l'heure de l'action pourra être configuré par exemple. Le serveur LoRaWAN et le serveur client pouvant être n'importe quel PC

### 4.1.2 Capteur Adeunis

Les fonctionnalités de ce capteur

- Alerter
  - Prévenir en cas
    - intrusion
    - d'ouverture ou de fermeture
    - de dépassement d'un certain nombre d'ouvertures/fermetures
- Mesurer
  - Compter le nombre d'événements sur une période

Ce capteur convient donc parfaitement à l'utilisation que l'on veut en faire, de plus il est à la fois compatible avec LoRaWAN et avec SigFox, qui sont deux technologies radio basses fréquences.

Avec ce capteur nous disposons d'un interrupteur reed dénommé ILS et d'un aimant de l'autre côté comme montré ci-dessous, et quand cette aimant vient ce collé avec l'interrupteur magnétique cela passe de ouvert à fermé.

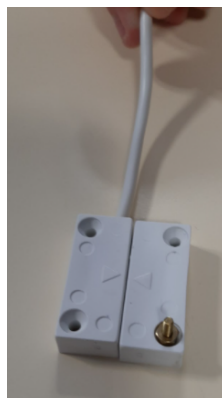


FIGURE 4.2 – Interrupteur reed



C'est dans la boîte grise qu'il y a la partie électronique qui va créer et transmettre la trame.



FIGURE 4.3 – Capteur Adeunis

### 4.1.3 Paserelle LoraWAN

La passerelle LoraWAN est capable de travailler sur 8 canaux, elle est performante. Nous n'avons pas dû en chercher car le lycée Malraux en possédait déjà une. Nous l'avons juste ajoutées sur notre réseau LoRaWAN.



FIGURE 4.4 – Paserelle LoraWAN

## 4.2 Présentation des logiciels

### 4.2.1 Chirpstack.io

ChirpStack.io fournit des composants open source pour les réseaux LoRaWAN. Ensemble, ils forment une solution prête à l'emploi comprenant une interface Web pour la gestion des appareils et des API pour l'intégration. Pour cela, nous avons besoin de 3 composants qui sont chirpstack-gateway-bridge, chirpstack-network-server, et chirpstack-application-server.

chirpstack-gateway-bridge est un service qui convertit les protocoles LoRa® Packet Forwarder en un format de données commun ChirpStack Network Server (JSON et Protobuf).

chirpstack-network-server est responsable de la déduplication des trames LoRaWAN reçues par les passerelles LoRa® et gère :

- L'authentification
- La couche mac LoRaWAN (et commandes mac)
- La communication avec le serveur d'applications ChirpStack
- La planification des trames de liaison descendante

chirpstack-application-server est la partie principale de notre projet. Il offre une interface Web où les utilisateurs, les organisations, les applications et les appareils LoraWan peuvent être gérés. Les données des appareils LoRaWAN peuvent être envoyées et reçues via MQTT, HTTP et être écrites directement dans un système de gestion de base de données tel que InfluxDB.

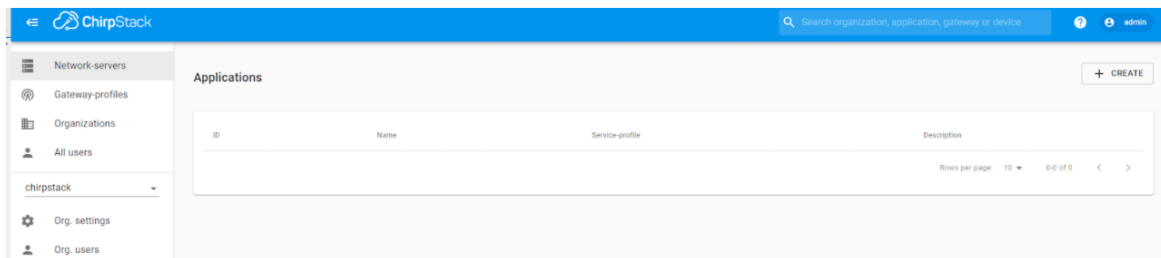


FIGURE 4.5 – Interface Web Chirpstack

## 4.2.2 IoTConfigurator

IoTConfigurator est une application qui permet de configurer les capteurs Adeunis. La configuration des capteurs se fait en local avec un câble microUSB. Elle peut se faire sur Android ou sur un ordinateur. IoTConfigurator va extraire automatiquement toutes les informations du produit connecté : référence, réseau, identifiant, versions. . . , et donner accès à la configuration applicative (modes de fonctionnement, périodicité, type de capteur. . . ) puis au réseau (réglages de l'acquittement, identifiants et clés. . . ). Il est possible de sauvegarder les configurations (export) et de les importer, sur le même type de produits pour gagner du temps. Durant notre projet, nous avons configuré le capteur sur un ordinateur avec Windows 10.

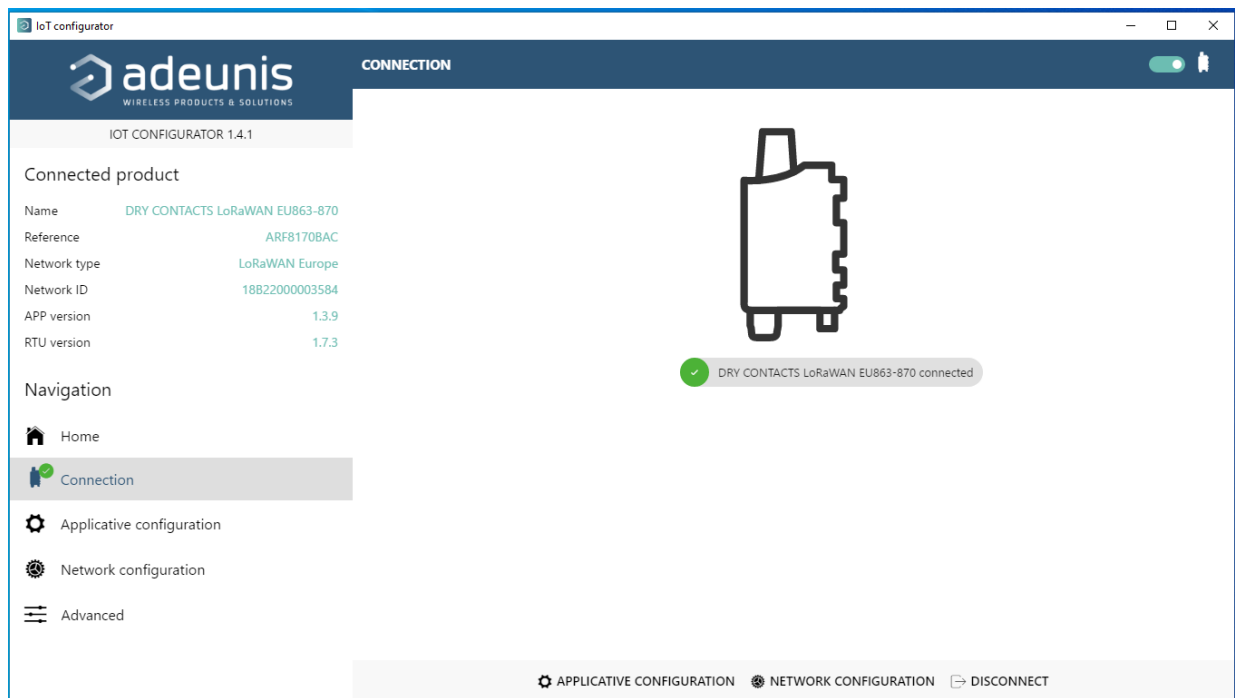


FIGURE 4.6 – IotConfigurator

## 4.2.3 PostgreSQL

PostgreSQL est un système de gestion de base de données utilisé par les composants LoraWan et qui est nécessaire au bon fonctionnement de ChirpStack. PostgreSQL est utilisé par ChirpStack Network Server et ChirpStack Application Server.

## 4.2.4 Grafana

Grafana est un logiciel qui permet la visualisation de données. Nous en avons besoin pour visualiser sous forme de tableau de bord ou graphique, la base de données InfluxDB que nous avons configurée sur notre serveur et intégrée via l'application serveur ChirpStack.io. Nous pourrions donc voir si une armoire est ouverte ou fermée ainsi que la date à laquelle l'événement s'est produit. Nous pourrions donc assurer la supervision (qui est le but même du logiciel) et la surveillance des armoires.

## 4.2.5 InfluxDB

Dans les différentes étapes, nous avons eu à renseigner la licence mais cela nous indique “An existing configuration was found”. Au début, on supposait que c’était bien la licence de cet IPBX mais après recherche, ça nous a généré une licence par défaut qui n’a aucune option supplémentaire.

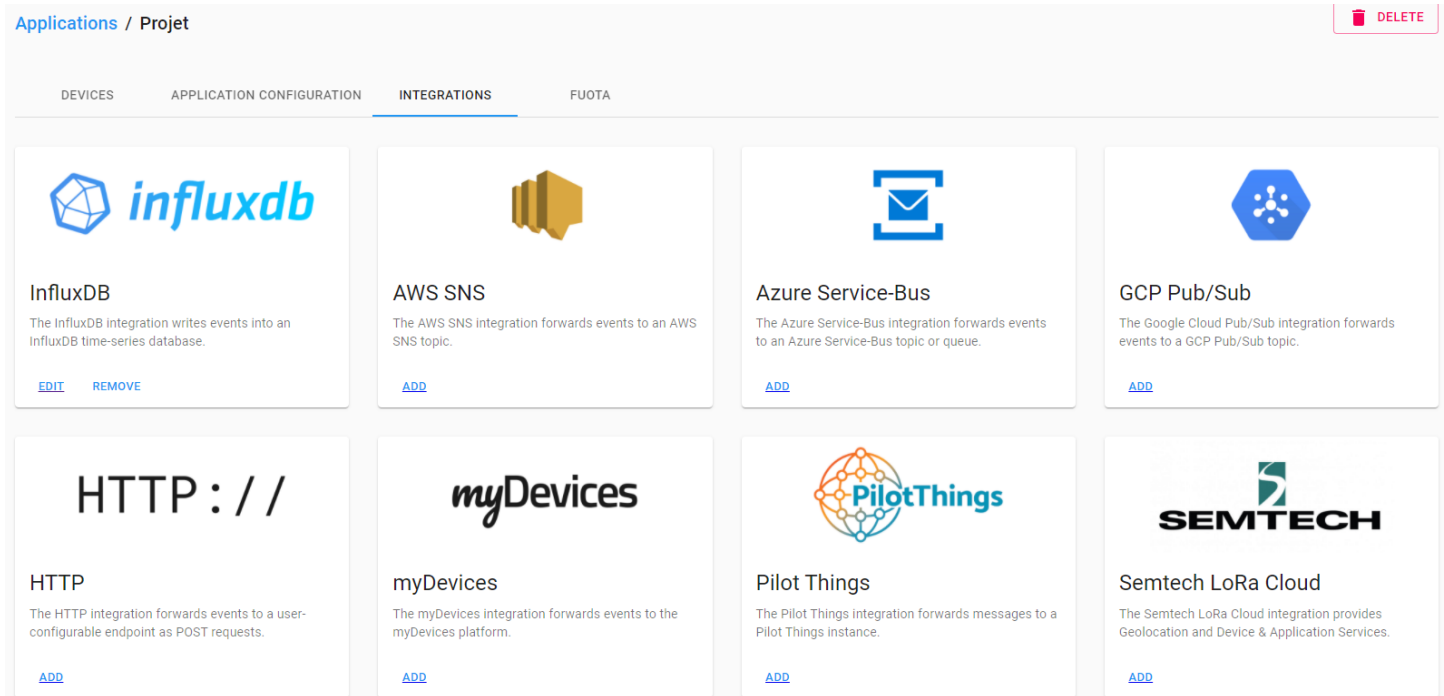


FIGURE 4.7 – Integration d’InfluxDB

## 4.2.6 PuTTY

PuTTY sera notre outil pour dialoguer avec le serveur. C’est une application qui est un émulateur de terminal, il va nous permettre principalement d’accéder en tant que client via le protocole SSH sur notre serveur. C’est sur ce serveur que nous allons installer tous les logiciels requis pour réaliser notre projet.

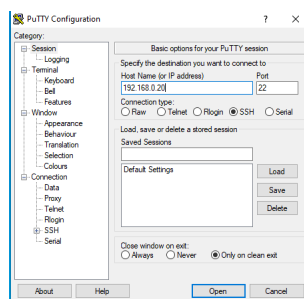


FIGURE 4.8 – PuTTY

# Partie 5

## Réalisation du projet

### 5.1 Gateway

Pour commencer, nous avons besoin d'installer notre Gateway qui est aussi notre serveur. Pour cela, nous avons utilisé l'outil de virtualisation VMWare dans le but de configurer notre machine. Nous utiliserons la plage d'adresse réservée aux machines utilisateurs du réseau du lycée Malraux, c'est-à-dire 192.168.0.0/24. L'adresse de notre machine sera 192.168.0.20.

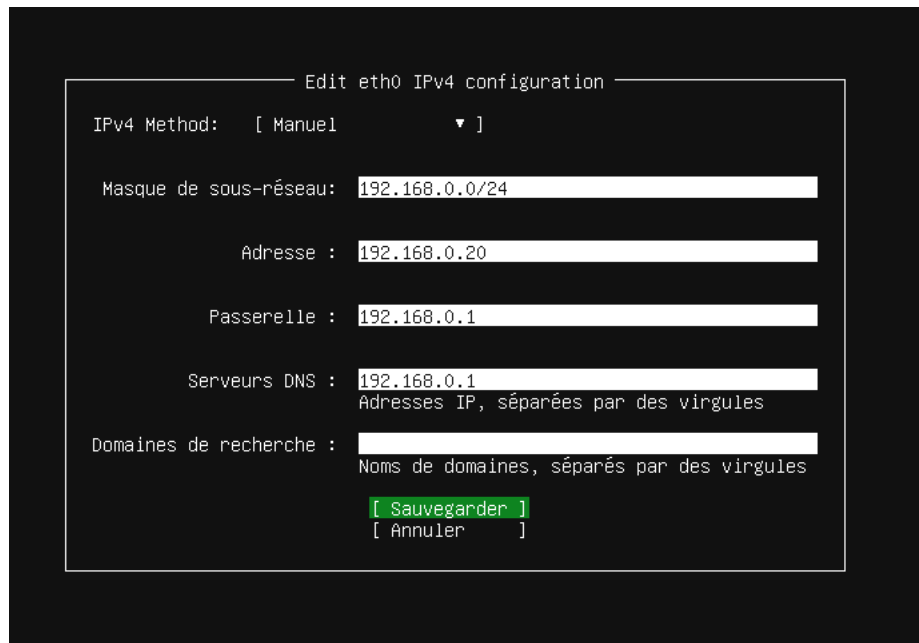


FIGURE 5.1 – Configuration des adresses

Par la suite, nous allons créer un utilisateur “rims” qui nous servira aussi à se connecter en SSH avec le logiciel PuTTY

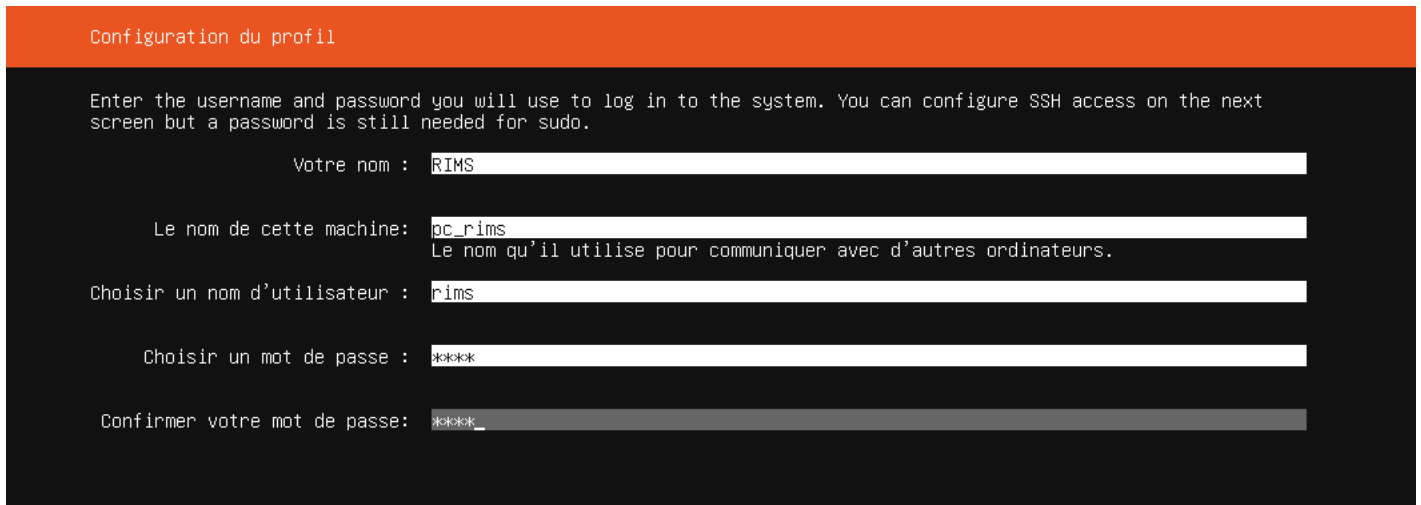


FIGURE 5.2 – Configuration de l'utilisateur

Pour pouvoir se connecter en SSH avec PuTTY, nous devons installer le paquet OpenSSH server. En effet, nous devons nous connecter en SSH afin de nous connecter plus facilement à la machine sans passer par VMWare et de terminer plus rapidement la configuration.

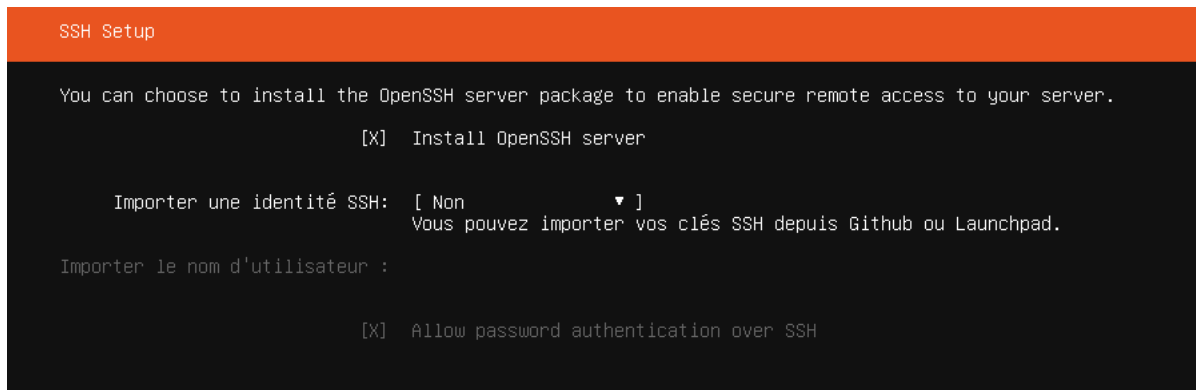


FIGURE 5.3 – SSH

Il faut aussi installer les paquets Mosquitto et PostgreSQL qui sont nécessaires au bon fonctionnement de la Gateway. En effet, Mosquitto est obligatoire pour pouvoir utiliser le protocole MQTT entre chirpstack-gateway-bridge, et chirpstack-network-server sinon ils ne communiqueront jamais.

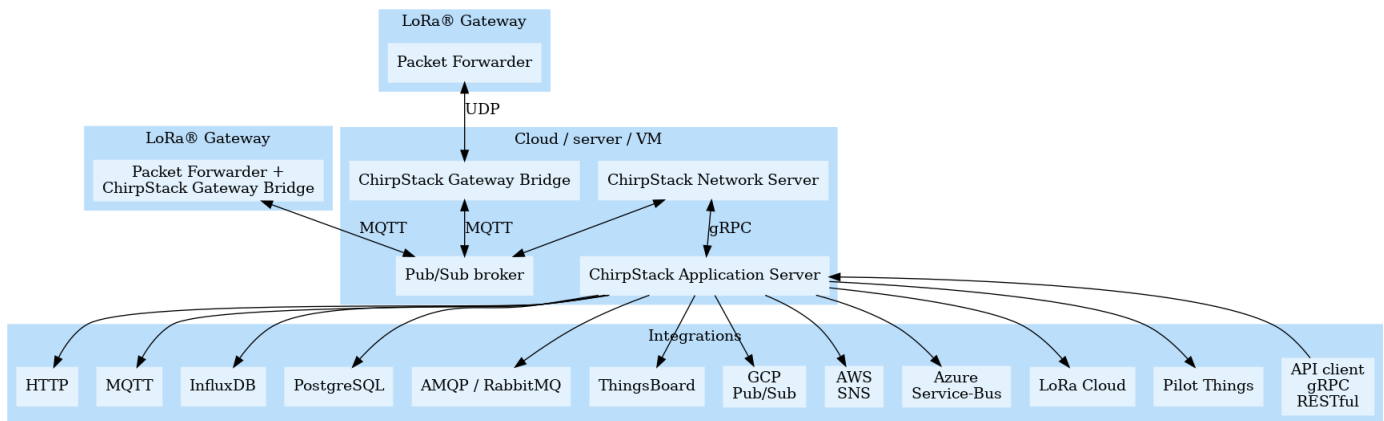


FIGURE 5.4 – Schema chirpstack

Les paquets de la Gateway sont transmis en UDP et passe par chirpstack-gateway-bridge puis utilise le protocole MQTT en passant par le modèle publish/subscribe qui dissocie l'éditeur (publisher) et le destinataire (subscriber) d'un message. Le broker va filtrer tous les messages entrants et les distribue en conséquence, et est connu par le publisher et le suscriber. Par la suite, le message est retransmis en MQTT jusqu'au chirpstack-network-server.

Modification du serveur DNS. Nous avons mis le serveur DNS de Google. Pour cela, nous sommes allés dans le répertoire, /etc/netplan et nous avons modifié le fichier 01-netcfg.yaml pour mettre le DNS de Google qui est 8.8.8.8

```

ServeurLorawanRIMS 192.168.0.20 sur SERVEUR2016 - Connexion à un ordinateur virtuel
Fichier Action Média Presse-papiers Affichage Aide
GNU nano 2.9.3 50-cloud-init

# This file is generated from information provided by
# the datasource. Changes to it will not persist across an instance.
# To disable cloud-init's network configuration capabilities, write a file
# /etc/cloud/cloud.cfg.d/99-disable-network-config.cfg with the following:
# network: {config: disabled}
network:
  ethernets:
    eth0:
      addresses:
        - 192.168.0.20/24
      gateway4: 192.168.0.1
      nameservers:
        addresses:
          - 8.8.8.8

version: 2
    
```

FIGURE 5.5 – Modification du serveur DNS

A partir de ce moment-là, nous pouvons utiliser PuTTY afin de se connecter en SSH, et ainsi terminer l'installation des services puis leurs configurations.

Il faut donc installer le serveur Redis. Pour cela, nous devons nous mettre en root en SSH et installer le paquet avec la commande `apt-get install redis-server`.

Aucune configuration n'est requise pour `redis-server`.

```
root@localhost:/etc/netplan# apt install redis-server
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
```

FIGURE 5.6 – installation `redis-server`

Auparavant sur VMWare, nous avons installé PostgreSQL. Pour que ChirpStack.io fonctionne parfaitement, nous avons besoin de créer 2 bases de données qui sont : `chirpstack_ns` (`ns` = network-server) et `chirpstack_as` (`as` = application-server). Le nom d'utilisateur par défaut est `postgres`. Nous avons suivi la documentation fournie par ChirpStack.

```
root@localhost:/home/rims# sudo -u postgres psql
psql (10.14 (Ubuntu 10.14-0ubuntu0.18.04.1))
Type "help" for help.

postgres=# create role chirpstack_ns with login password 'rims' ;
CREATE ROLE
postgres=# create database chirpstack_ns with owner chirpstack_ns;
CREATE DATABASE
```

FIGURE 5.7 – Creation BDD

```
root@localhost:/home/rims# sudo -u postgres psql
psql (10.14 (Ubuntu 10.14-0ubuntu0.18.04.1))
Type "help" for help.

postgres=# create role chirpstack_as with login password 'rims';
CREATE ROLE
postgres=# create database chirpstack_as with owner chirpstack_as;
CREATE DATABASE
postgres=# \c chirpstack_as
You are now connected to database "chirpstack_as" as user "postgres".
chirpstack_as=# create extension pg_trgm;
CREATE EXTENSION
chirpstack_as=# create extension hstore;
CREATE EXTENSION
chirpstack_as=# \q
root@localhost:/home/rims#
```

FIGURE 5.8 – Configuration BDD

Ces 2 bases de données nous serviront dans les fichiers de configuration de `chirpstack-network-server` et de `chirpstack-application-server` que nous installerons juste après, avec la commande `apt-get install`.



Concrètement, nous avons créé 2 BDD qui ont pour mot de passe “rims” et pour utilisateur “postgres”, mais également activer les extensions `pg_trgm` et `hstore`.

Nous pouvons donc passer à la dernière étape, installer `chirpstack-gateway-bridge`, `chirpstack-network-server`, `chirpstack-application-server`.

Une fois les paquets installés, nous devons nous rendre dans les fichiers de configuration de `chirpstack-network-server` et `chirpstack-application-server`. En effet comme dit tout à l’heure, PostgreSQL utilise ChirpStack Network Server et ChirpStack Application Server.

Ces fichiers se trouvent dans `/etc/chirpstack-network-server/chirpstack-network-server.toml` et `/etc/chirpstack-application-server/chirpstack-application-server.toml`

Le fichier de conf se présente comme ceci :

```

GNU nano 2.9.3 chirpstack-application-server.toml Modified
# This configuration sets the required settings and configures an integration
# with a MQTT broker. Many options and defaults have been omitted for
# simplicity.
#
# See https://www.chirpstack.io/application-server/install/config/ for a full
# configuration example and documentation.

# PostgreSQL settings.
#
# Please note that PostgreSQL 9.5+ is required.
[postgresql]
# PostgreSQL dsn (e.g.: postgres://user:password@hostname/database?sslmode=disable).
#
# Besides using an URL (e.g. 'postgres://user:password@hostname/database?sslmode=disable')
# it is also possible to use the following format:
# 'user=chirpstack_as dbname=chirpstack_as sslmode=disable'.
#
# The following connection parameters are supported:
#
# * dbname - The name of the database to connect to
# * user - The user to sign in as
# * password - The user's password
# * host - The host to connect to. Values that start with / are for unix domain sockets. (default $
# * port - The port to bind to. (default is 5432)
# * sslmode - Whether or not to use SSL (default is require, this is not the default for libpq)
# * fallback_application_name - An application_name to fall back to if one isn't provided.
# * connect_timeout - Maximum wait for connection, in seconds. Zero or not specified means wait in$
# * sslcert - Cert file location. The file must contain PEM encoded data.
# * sslkey - Key file location. The file must contain PEM encoded data.
# * sslrootcert - The location of the root certificate file. The file must contain PEM encoded dat$
#
# Valid values for sslmode are:
#
# * disable - No SSL
# * require - Always SSL (skip verification)
# * verify-ca - Always SSL (verify that the certificate presented by the server was signed by a tr$
# * verify-full - Always SSL (verify that the certification presented by the server was signed by $
dsn="postgres://localhost/chirpstack_as?sslmode=disable"
[application_server.external_api]
jwt_secret="rims"
    
```

FIGURE 5.9 – Fichier de conf

Il faut configurer la partie `postgresql` et ajouter le nom de source de données (`dsn`) correspondant au fichier de conf (ici `chirpstack_as`) afin qu’il puisse se connecter à la base de données.

ça représente donc le `[nom d'utilisateur] ://[base de données] :[mot de passe]@[adresse de la machine]/[base de données]` On enregistre et on fait de même avec le `network-server` (`chirpstack_ns`)

Il ne reste plus qu’à démarrer et activer les services grâce à la commande `systemctl` et vérifier leur bon fonctionnement.

```

GNU nano 2.9.3  chirpstack-application-server/chirpstack-application-server.toml  Modified
# * fallback_application_name - An application_name to fall back to if one isn't provided.
# * connect_timeout - Maximum wait for connection, in seconds. Zero or not specified means wait in$
# * sslcert - Cert file location. The file must contain PEM encoded data.
# * sslkey - Key file location. The file must contain PEM encoded data.
# * sslrootcert - The location of the root certificate file. The file must contain PEM encoded dat$
#
# Valid values for sslmode are:
#
# * disable - No SSL
# * require - Always SSL (skip verification)
# * verify-ca - Always SSL (verify that the certificate presented by the server was signed by a tr$
# * verify-full - Always SSL (verify that the certification presented by the server was signed by $
dsn="postgres://chirpstack_as:rims@localhost/chirpstack_as?sslmode=disable"
    
```

FIGURE 5.10 – configuration

```

root@localhost:/etc# systemctl start chirpstack-application-server
root@localhost:/etc# systemctl enable chirpstack-application-server
root@localhost:/etc# systemctl status chirpstack-application-server
● chirpstack-application-server.service - ChirpStack Application Server
   Loaded: loaded (/lib/systemd/system/chirpstack-application-server.service; enabled; vendor pres
   Active: active (running) since Thu 2020-10-08 08:30:53 UTC; 17s ago
     Docs: https://www.chirpstack.io/
   Main PID: 8892 (chirpstack-appl)
    Tasks: 4 (limit: 1020)
   CGroup: /system.slice/chirpstack-application-server.service
           └─8892 /usr/bin/chirpstack-application-server

oct. 08 08:30:53 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:30
oct. 08 08:30:53 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:30
oct. 08 08:30:55 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:30
oct. 08 08:30:57 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:30
oct. 08 08:30:59 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:30
oct. 08 08:31:01 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:31
oct. 08 08:31:03 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:31
oct. 08 08:31:05 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:31
oct. 08 08:31:07 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:31
oct. 08 08:31:09 localhost.localdomain chirpstack-application-server[8892]: time="2020-10-08T08:31
lines 1-19/19 (END)
    
```

FIGURE 5.11 – Service application-server active

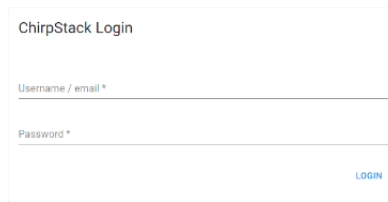
On peut donc voir que tout fonctionne et que nous pouvons avoir accès à l'interface web de ChirpStack.io.

## 5.2 Chirpstack

Après avoir réalisé l'installation des différents services nécessaires au bon fonctionnement de chirpstack, on peut maintenant se connecter à l'application chirpstack qui va nous permettre de gérer la gateway loRaWAN et également le capteur adeunis, le tout dans le but de pouvoir extraire les informations intéressantes.

### 5.2.1 Configuration de chirpstack

On doit se connecter à chirpstack dans un premier temps avec les logins définis précédemment.



ChirpStack Login

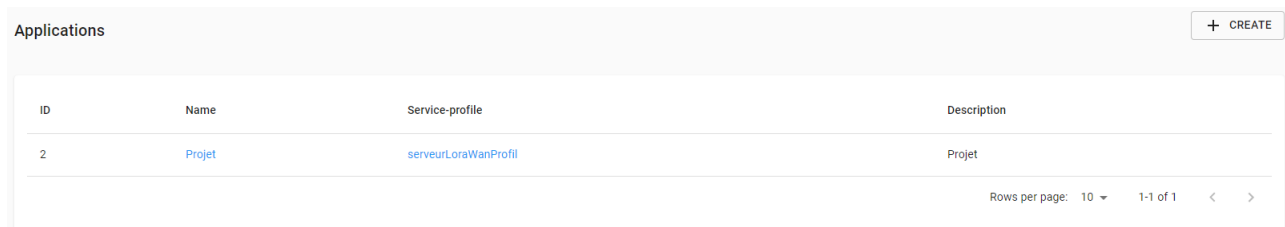
Username / email \*

Password \*

LOGIN

FIGURE 5.12 – Connexion à Chirpstack

Pour commencer la configuration, on va créer un “projet” qui sera notre base afin de lier la gateway et le capteur, ce projet va être rattaché à un service-profile qui est une fonctionnalité assez pratique qui va permettre d'associer l'utilisateur et le network server à ce projet.



ID	Name	Service-profile	Description
2	Projet	serveurLoraWanProfil	Projet

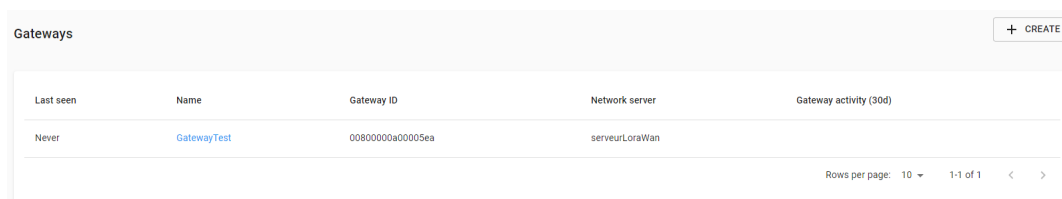
Rows per page: 10 ▾ 1-1 of 1 < >

FIGURE 5.13 – creation "projet"

### 5.2.2 Configuration de gateway

Pour configurer la gateway sur chirpstack, nous avons uniquement à lui donner un nom, préciser son ID, et l'associer au network server.

Après cela, chirpstack pourra recevoir les informations de la gateway.



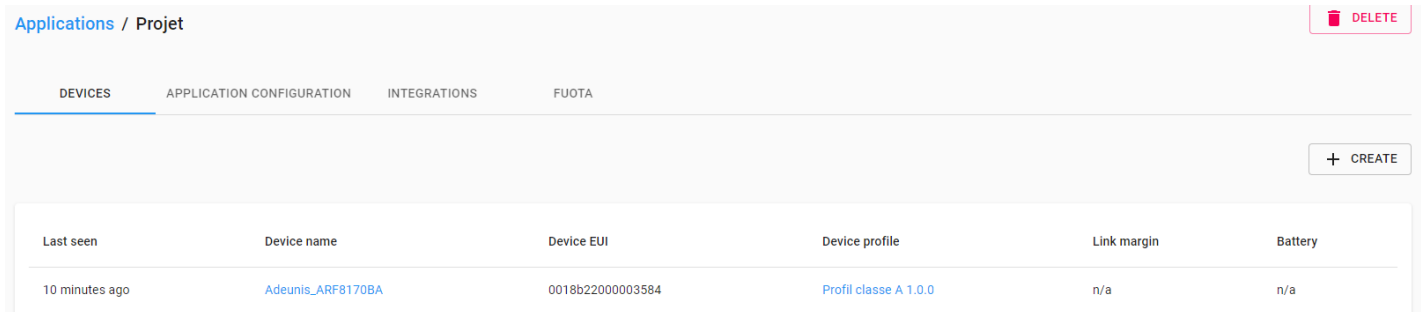
Last seen	Name	Gateway ID	Network server	Gateway activity (30d)
Never	GatewayTest	0080000a00005ea	serveurLoraWan	

Rows per page: 10 ▾ 1-1 of 1 < >

FIGURE 5.14 – Configuration gateway

### 5.2.3 Configuration du capteur

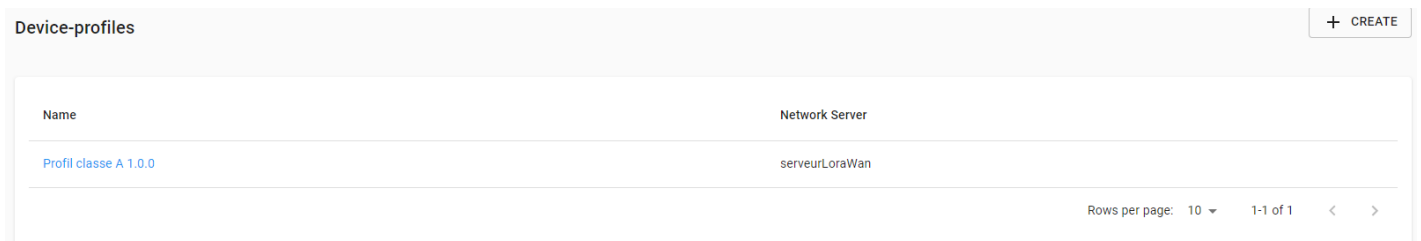
On passe maintenant à la création du device-profiles, première étape pour le capteur, qui est cette fois-ci fournie dans la documentation du capteur. Ici, c'est un classe A soit la classe la plus basique pour des objets connectés supportés par loRaWAN. Cette classe signifie simplement que l'objet connecté est capable de faire de la connection bi-directionnelle entre un appareil et une passerelle.



Last seen	Device name	Device EUI	Device profile	Link margin	Battery
10 minutes ago	<a href="#">Adeunis_ARF8170BA</a>	0018b22000003584	<a href="#">Profil classe A 1.0.0</a>	n/a	n/a

FIGURE 5.15 – Configuration capteur

Pour la configuration du capteur, il nous faut lui donner un nom, le lier au device-profile créé plus haut. Puis il faut notifier l'ID du capteur qui est noté sur le capteur généralement, l'ID étant l'information la plus importante car elle permet à la gateway de détecter le capteur et de communiquer avec.



Name	Network Server
<a href="#">Profil classe A 1.0.0</a>	serveurLoraWan

FIGURE 5.16 – Profil

Maintenant, le lien capteur-gateway fonctionne sur chirpstack, on peut donc passer à l'acquisition des données échangées entre le capteur et la gateway.

### 5.2.4 Analyse du capteur

Pour débiter l'analyse du capteur, nous avons une application mise à disposition "iot configurator" qui nous permet de gérer le capteur sous certains aspects.

Dans cette application, nous avons principalement découvert que le capteur dispose de 4 modes de configuration et qu'on peut les changer sur cette application.

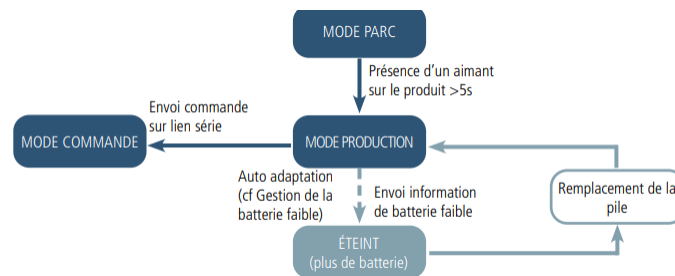


FIGURE 5.17 – Analyse du mode

- PARC qui est le mode sortie d'usine et qui permet une consommation minimale
- COMMANDE qui est un mode de configuration du capteur
- PRODUCTION le mode principal qui est quand le produit est utilisé
- BATTERIE FAIBLE quand le produit détecte n'a plus de batterie

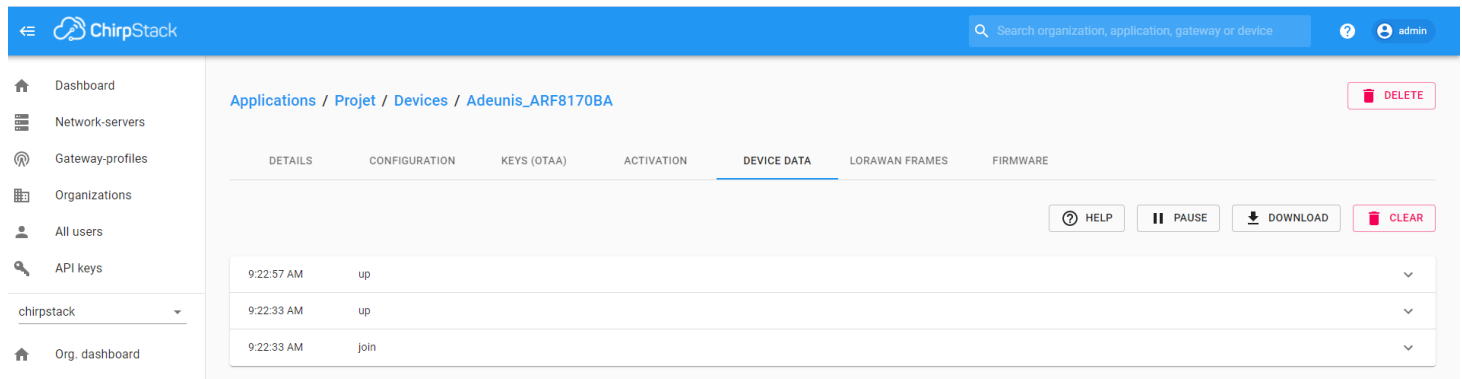
L'application "iot configurator" peut nous permettre également de changer les clés EUI et APP\_KEY et nous propose une synthèse des informations dont elle dispose sur le produit.

IOT CONFIGURATOR 1.4.1	
Connected product	
Name	DRY CONTACTS LoRaWAN EU863-870
Reference	ARF8170BAC
Network type	LoRaWAN Europe
Network ID	18B22000003584
APP version	1.3.9
RTU version	1.7.3

FIGURE 5.18 – IoT configurator

Afin de faire fonctionner le capteur, nous l'avons enlevé du mode PARC qui est le mode de sortie d'usine et nous avons sélectionné le mode PRODUCTION.

Maintenant, sur l'application chirpstack en ouvrant et fermant le capteur comme montré dans la partie présentation du matériel, nous obtenons bien des trames.

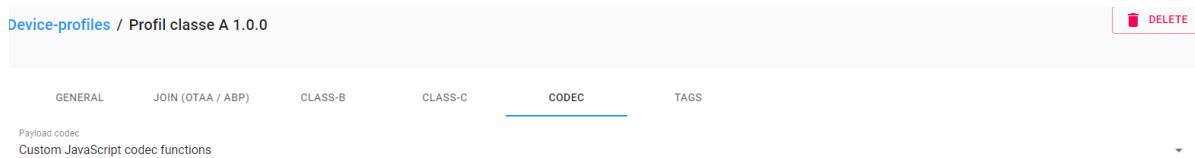


Time	Event	Action
9:22:57 AM	up	▼
9:22:33 AM	up	▼
9:22:33 AM	join	▼

FIGURE 5.19 – Trames

La première trame que l'on reçoit est l'initiation de la phase "join" c'est à dire que le capteur dit qu'il fonctionne correctement et qu'il est capable d'envoyer d'autres trames. La deuxième et troisième trames sont les trames les plus classiques que l'on peut trouver c'est à dire qu'à l'ouverture et à la fermeture du capteur, ce sont ces trames que l'on reçoit.

Malheureusement, quand on ouvre la trame, nous avons uniquement des informations non lisibles et surtout aucune information sur le fait que la porte est bien ouverte ou fermée ou à quoi correspondait la trame.



Device-profiles / Profil classe A 1.0.0

GENERAL JOIN (OTAA / ABP) CLASS-B CLASS-C **CODEC** TAGS

Payload codec  
Custom JavaScript codec functions

FIGURE 5.20 – Javascript

Mais heureusement, sur l'application chirpstack, nous avons un "codec" qui permet de faire du traitement de trame via javascript.

Nous devons donc écrire un code en javascript qui nous permet d'afficher toutes les informations utiles et exploitables à la sécurisation des PM.

## 5.2.5 Ecriture du code Javascript

Pour commencer, nous avons déjà regardé dans la documentation du capteur afin de mieux comprendre comment on pourrait trouver les informations d'ouverture et de fermeture du capteur.

Nous avons compris que la première information que la trame envoie est "quel type de trame ?"

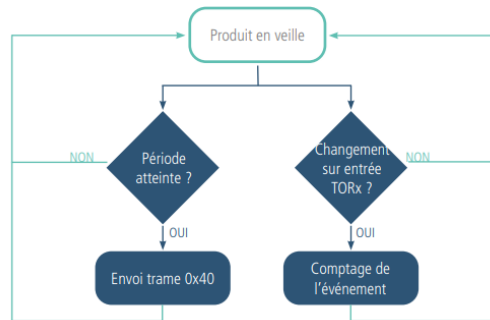


FIGURE 5.21 – Trame 0x40

Dans la documentation, on est capable de déterminer 4 types de trames :

- 0x40 Trame de donnée - Celle qu'on reçoit à chaque ouverture / fermeture
- 0x30 Trame de vie - On la reçoit périodiquement en fonction du mode de configuration
- 0x31 Trame de non réponse à une demande de valeur de registre
- 0x10 trame d'information sur la configuration - quand il y a un changement sur la configuration

La trame la plus importante pour nous est donc la trame de donnée soit 0x40.

C'est donc cette trame qu'on va analyser, dans le but de trouver comment on peut prendre l'information d'ouverture ou de fermeture du capteur.

Ci-dessous, c'est la documentation du capteur pour 0x40 mise à disposition par Adeunis

Offset (in byte)	Data	Description
0	0x40	Frame code
1	Status	<b>Erreur ! Source du renvoi introuvable.</b> Status byte
2-3	Channel 1 info	If configured in input mode: event counter
4-5	Channel 2 info	
6-7	Channel 3 info	
8-9	Channel 4 info	
10	Details	Define precisely the input/output state (ON/CLOSED : 1, OFF/OPEN : 0) <ul style="list-style-type: none"> <li>• &lt;0&gt; Channel1 current state</li> <li>• &lt;1&gt; Channel1 state when sending the previous frame</li> <li>• &lt;2&gt; Channel2 current state</li> <li>• &lt;3&gt; Channel2 state when sending the previous frame</li> <li>• &lt;4&gt; Channel3 current state</li> <li>• &lt;5&gt; Channel3 state when sending the previous frame</li> <li>• &lt;6&gt; Channel4 current state</li> <li>• &lt;7&gt; Channel4 state when sending the previous frame</li> </ul>
11-14	Timestamp	Only for LoRaWAN product with timestamping enabled. Timestamp of the last input/output state change in EPOCH 2013 format.

FIGURE 5.22 – explication 0x40

On peut déjà constater, à ce point de notre étude, qu'on peut récupérer énormément d'informations très utiles, tel que le statut sur le bytes[1] ou encore les informations du channel 1 sur le bytes[2] et bytes[3] qui est le seul channel qui est configurable sur l'application "iot configurator".

Dans le bytes[10] on trouve enfin notre information essentielle qui est l'ouverture ou la fermeture du capteur ici par "Channel1 current state". Vu qu'on utilise uniquement le channel , cela sera <0> et <1> qu'on va utiliser ici. On nous donne également la valeur précise soit quand l'état est fermé la valeur est de 1 et quand l'état est ouvert la valeur est 0, ce qui est très important pour notre code.

Nous avons également découvert qu'entre 0x40, 0x30, 0x31, 0x10, le premier byte qui est le statut reste toujours à la même position et dispose des mêmes champs pour n'importe quel type de trame. Donc nous avons écrit un code pour le bytes[1] qui est toujours le même puis un code pour les 4 types de trames différents

Dans ce code, nous avons au maximum essayer de prendre toutes les informations :



```
# function toHexString(bytes) {
return bytes.map(function(byte) {
return ("00" + (byte & 0xFF).toString(16)).slice(-2)
}).join("") }

function Decode(fPort, bytes) {
/* var tempObj = new Object();
tempObj.dataHex = toHexString(bytes);
tempObj.message = "Raw message bytes";
return tempObj;
*/
var trame = new Object();
trame.code = bytes[0];
//trame.lowbat1 = (bytes[1] >> 1)0x01;
if([(bytes[1] >> 1)0x01] == 0)
{trame.lowbat = "batterieok"}
elsetrame.lowbat = "batteriefaible";
//trame.panne = (bytes[1] >> 2)0x01;
if([(bytes[1] >> 2)0x01] == 0)
{trame.panne = "RAS"}
elsetrame.panne = "erreurmatrielle, SAV";
trame.compteur = (bytes[1] >> 5)0x03;
if(bytes[0] == 0x40)
{
trame.type = "tramedonnees";
trame.status = bytes[1];
trame.tor1Infos = bytes[2];
trame.tor1Infos2 = bytes[3];
trame.details = bytes[10];
trame.etatCourantTor1 = (bytes[10]0x01);
trame.etatPrecedentTor1 = (bytes[10]0x02) >> 1;
if(trame.etatCourantTor1 == 0){
trame.etat = "ouvert";}
else(trame.etat = "ferme");{
if(bytes[0] == 0x30){
trame.type = "tramedevie";}
if(bytes[0] == 0x31){
trame.type = "tramederponseunedemandevalueurderegistre";}
if(bytes[0] == 0x10){
trame.type = "tramed'informationsurlaconfiguration";}
returntrame;}
```

## 5.2.6 Analyse du résultat

Maintenant si on retourne sur le “codec” et qu’on ouvre et ferme avec le capteur, nous obtenons des informations qui sont utilisables et lisibles.

```
▼ objectJSON: {} 12 keys
  code: 64
  compteur: 1
  details: 15
  etat: "ferme"
  etatCourantTor1: 1
  etatPrecedentTor1: 1
  low_bat: "batterie ok"
  panne: "RAS"
  status: 32
  tor1Infos: 0
  tor1Infos2: 0
  type: "trame donnees"
tags: {} 0 keys
confirmedUplink: false
devAddr: "008bfa1f"
```

FIGURE 5.23 – detail trame

Par exemple ci-dessus, on peut dire que l’état est “fermé” et que c’est une trame de type “trame de donnée”, également que l’état précédent était 1 soit fermé, sachant que dans le mode “production” la période de renvoi du “up” est de 30 secondes, cela signifie que le capteur est fermé depuis 30 secondes. On reçoit des informations du bytes[1] également comme “batterie ok” et “RAS”

Maintenant que nous sommes capables d’exploiter les données et que ses données sont compréhensibles, nous devons être capables de proposer un support de visualisation des informations.

## 5.3 Supervision

La supervision est le fait de réaliser un suivi informatique des procédés de fabrication qui ont été automatisés. Dans notre cas, il s'agira d'effectuer la surveillance des points de mutualisation (PM) via des graphiques montrant l'état actuel des PM (ouverts ou fermés) ainsi que d'autres données paramétrables. Pour arriver à un tel résultat, nous avons besoin de données à afficher ainsi que d'un logiciel qui traitera ces données afin de créer de beaux graphiques.

Pour la partie base de données, nous avons choisi InfluxDB et pour le superviseur, nous avons choisi Grafana. Ces deux choix s'expliquent très facilement : en effet, ChirpStack Application Server nous permet d'envoyer et recevoir des charges utiles de périphériques (dans notre cas un capteur d'ouverture) via différents services dont l'intégration est possible (MQTT, InfluxDB...). C'est justement InfluxDB que nous avons choisi car l'intégration de la base de données est réalisée très facilement via l'interface graphique du serveur.

Pour le superviseur, le choix de Grafana s'est fait tout aussi naturellement : il est recommandé par ChirpStack et de plus, l'intégration d'une base de données InfluxDB est très facile. Comme les serveurs ChirpStack ont été créés pour permettre une intégration facile de ces outils (et d'autres outils que nous n'utilisons pas car nous n'en avons pas besoin), une fois l'installation réalisée, il suffira de configurer les éléments basiques afin que tout fonctionne bien ensemble.

Nous utilisons des configurations de base afin de pouvoir reproduire cette architecture de la manière la plus simple possible.

### 5.3.1 Installation de InfluxDB

Nous avons installé InfluxDB sur la machine où le serveur est installé : 192.168.0.20, via une connexion SSH distante, de cette manière nous avons le mot de passe administrateur et pouvons installer les logiciels nécessaires au projet.

```

root@pc_rims:/home/zims# apt-get install influxdb
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les NOUVEAUX paquets suivants seront installés :
  influxdb
0 mis à jour, 1 nouvellement installés, 0 à enlever et 112 non mis à jour.
Il est nécessaire de prendre 20813 ko dans les archives.
Après cette opération, 10,7 Mo d'espace disque supplémentaires seront utilisés.
Réception de :1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 influxdb
amd64 1.1.1+dfsg1-4 [20813 kB]
20813 ko réceptionnés en 0s (60648 ko/s)
Sélection du paquet influxdb précédemment désélectionné.
(Lecture de la base de données... 139025 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de ../influxdb_1.1.1+dfsg1-4_amd64.deb ...
Dépaquetage de influxdb (1.1.1+dfsg1-4) ...
Traitement des actions différées (« triggers ») pour ureadahead (0.100.0-20) ...
Paramétrage de influxdb (1.1.1+dfsg1-4) ...
Adding system user `influxdb' (UID 114) ...
Adding new user `influxdb' (UID 114) with group `nogroup' ...
Not creating home directory `/var/lib/influxdb'.
Adding group `influxdb' (GID 117) ...
Done.
Adding user `influxdb' to group `influxdb' ...
Adding user influxdb to group influxdb
Done.
    
```

FIGURE 5.24 – Configuration InfluxDB

Une fois le service influxdb-server installé, la commande `influxdb start` permet de le démarrer. Nous avons ensuite installé `influxdb-client` :

```

root@pc_rims:/home/rims# apt-get install influxdb-client
Lecture des listes de paquets... Fait
Construction de l'arbre des dépendances
Lecture des informations d'état... Fait
Les NOUVEAUX paquets suivants seront installés :
  influxdb-client
0 mis à jour, 1 nouvellement installés, 0 à enlever et 112 non mis à jour.
Il est nécessaire de prendre 10146 ko dans les archives.
Après cette opération, 30969 ko d'espace disque supplémentaires seront utilisés.
Réception de :1 http://archive.ubuntu.com/ubuntu bionic/universe amd64 influxdb-client amd64 1.1.1+dfsg1-4 [10146 kB]
10146 ko réceptionnés en 0s (30396 ko/s)
Sélection du paquet influxdb-client précédemment désélectionné.
(Lecture de la base de données... 139038 fichiers et répertoires déjà installés.)
Préparation du dépaquetage de ../influxdb-client_1.1.1+dfsg1-4_amd64.deb ...
Dépaquetage de influxdb-client (1.1.1+dfsg1-4) ...
Paramétrage de influxdb-client (1.1.1+dfsg1-4) ...
Traitement des actions différées (« triggers ») pour man-db (2.8.3-2ubuntu0.1) ...
root@pc_rims:/home/rims#
    
```

FIGURE 5.25 – Configuration influxdb-server

### 5.3.2 Configuration de base de InfluxDB

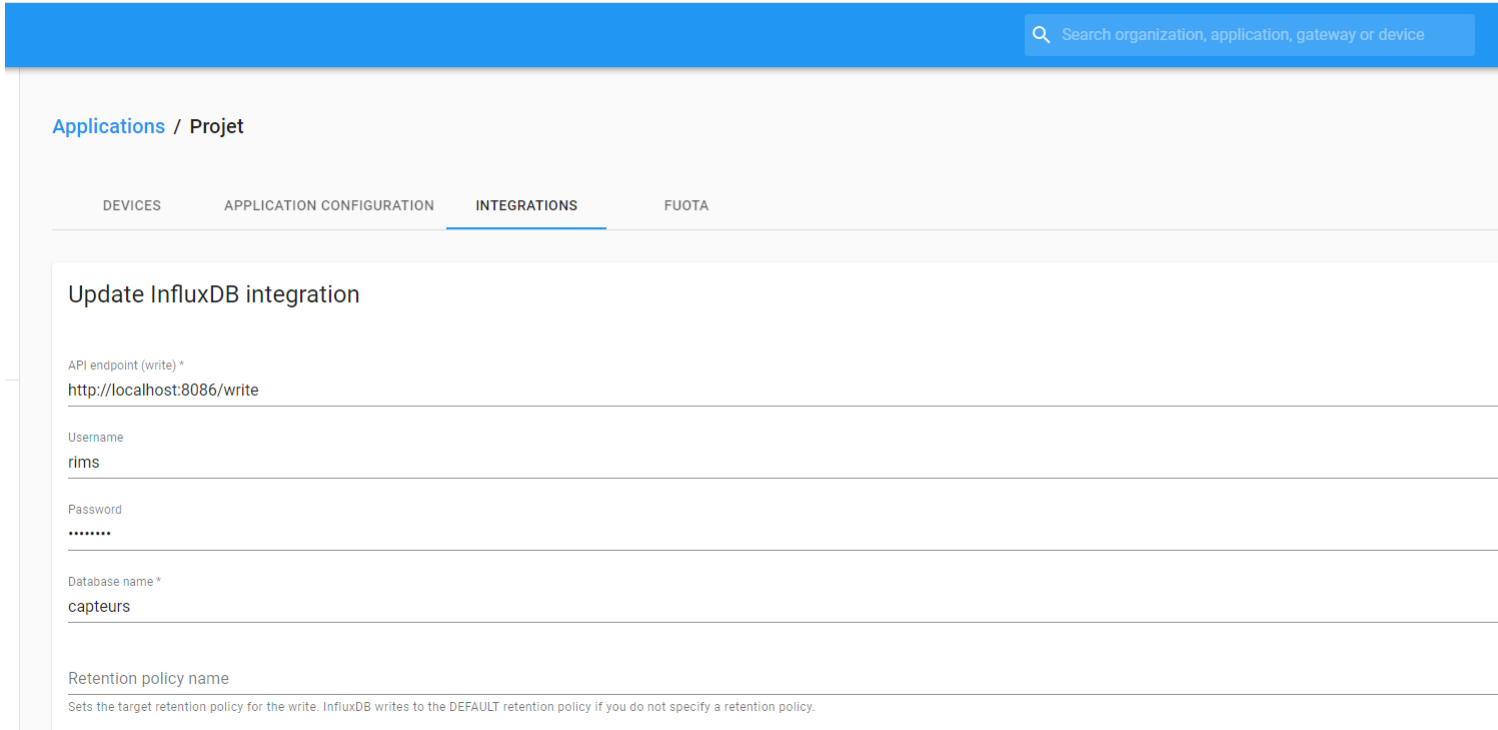
Nous avons créé une base de données “capteurs” dans laquelle le serveur ChirpStack enverra les données. Puis, nous avons créé un utilisateur afin de pouvoir gérer les bases de données (BDD) et les tables sans utiliser le compte admin. Pour cette raison, nous l’avons créé en lui donnant le maximum de droits. L’utilisateur se nomme “rims” :

```

> show databases
name: databases
name
----
_internal
> CREATE DATABASE capteurs
> show databases
name: databases
name
----
_internal
capteurs
> CREATE USER rims WITH PASSWORD 'progr00' WITH ALL PRIVILEGES
> exit
root@pc_rims:/home/rims# service influxdb restart
    
```

FIGURE 5.26 – Configuration BDD

### 5.3.3 Intégration de InfluxDB dans le serveur Chirpstack



The screenshot shows the ChirpStack web interface. At the top, there is a blue search bar with the text "Search organization, application, gateway or device". Below this, the breadcrumb "Applications / Projet" is visible. The main navigation menu includes "DEVICES", "APPLICATION CONFIGURATION", "INTEGRATIONS" (which is highlighted), and "FUOTA". The "Update InfluxDB integration" form contains the following fields:

- API endpoint (write) \***: `http://localhost:8086/write`
- Username**: `rims`
- Password**: `.....`
- Database name \***: `capteurs`
- Retention policy name**: (empty)

Below the retention policy name field, there is a small note: "Sets the target retention policy for the write. InfluxDB writes to the DEFAULT retention policy if you do not specify a retention policy."

FIGURE 5.27 – Integration InfluxDB

Sur l'interface graphique du serveur, une partie est dédiée aux Intégrations des différents services, parmi ceux-ci on retrouve InfluxDB. Il suffit alors de remplir le formulaire et si toutes les informations sont correctes la liaison entre InfluxDB et ChirpStack est fonctionnelle.

Comme chirpstack et influxDB sont installés sur la même machine, il est tout à fait normal que l'adresse de la BDD soit `http://localhost:8086/write`. Localhost désigne la boucle locale : le PC en lui-même (donc équivalent à 127.0.0.1), 8086 désigne le port d'écoute de InfluxDB, il s'agit du port par défaut mais c'est paramétrable si nous voulons le modifier. Il a fallu renseigner également l'utilisateur, ici "rims" ainsi que son mot de passe et pour finir, la BDD pour envoyer les données, donc "capteurs". Optionnellement, il est possible de rajouter un nom de politique de rétention ainsi que sa durée exprimée en secondes/minutes/heures.

### 5.3.4 Vérification de la communication entre ChirpStack et InfluxDB

Si jusqu'à maintenant le projet se déroulait plus ou moins sans accroc, cette partie-là nous a causé quelques soucis. En effet, InfluxDB était bel et bien installé, tout était bien configuré. L'intégration sur le serveur n'a posé aucun problème. Malgré cela, InfluxDB ne recevait rien, la base "capteurs" était vide. Nous avons mis plusieurs heures avant d'arriver à trouver le problème.

Nous avons d'abord commencé par désinstaller, purger, réinstaller InfluxDB, et refaire la configuration. Malheureusement ça n'a rien donné. Mais grâce à l'aide de notre professeur, nous avons pu déterminer le problème et trouver une solution.

En effet, nous avons essayé de voir si la BDD pouvait bel et bien recevoir des données. Nous avons donc inséré des données manuellement via une commande - curl et ainsi vérifier que le problème ne venait pas de la BDD en elle-même. Comme les données étaient bien consultables, notre BDD "capteurs" est totalement fonctionnelle et le problème ne venait pas de là. C'est donc le serveur Chirpstack qui avait un problème, il fallait ensuite découvrir si les trames ne s'envoyaient pas, ou si elles s'envoyaient mais n'étaient pas bien reçues. Pour cela, nous avons consulté les logs et ceux-ci nous ont confirmé que les données étaient bien envoyées.

Après avoir vérifié nos configurations une énième fois, nous nous sommes rendu compte que dans l'intégration d'InfluxDB, nous avons mis un nom de politique de rétention. Comme dit plus haut, étant donné qu'il s'agit d'un champ facultatif, nous avons essayé de le supprimer. L'essai a été concluant et à la suite de cela, les données étaient bien reçues dans la BDD "capteurs".

### 5.3.5 Installation de Grafana

Contrairement à InfluxDB, Grafana ne s'installe pas sur le serveur, mais sur n'importe quel PC dans le même réseau que le serveur. Il faudra en effet que Grafana puisse envoyer des requêtes à InfluxDB. Il suffit de télécharger l'exécutable sur leur site (<https://grafana.com/get/?plcmt=top-navcta=downloads>) et d'avoir les droits admin sur le PC pour pouvoir l'installer.

Une fois installé, il faut lancer Grafana et se connecter à l'interface graphique via l'URL

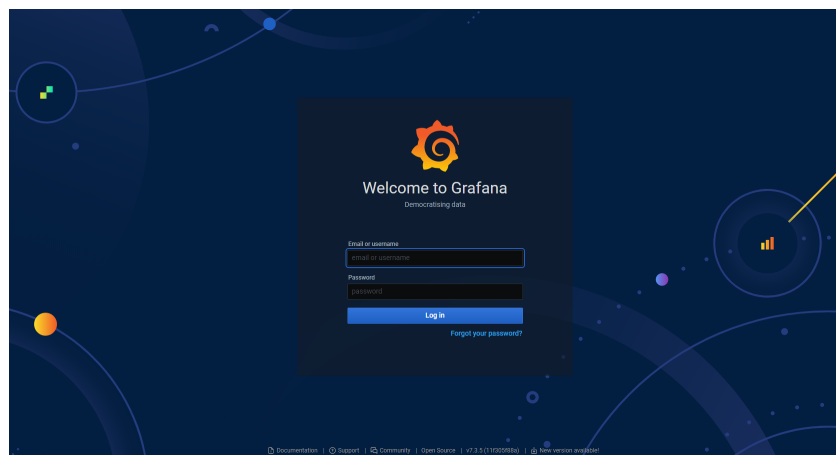


FIGURE 5.28 – Grafana

### 5.3.6 Configuration de base de grafana

Une fois connecté avec le compte admin (pour avoir tous les droits de création/suppression), on va se rendre dans la partie Intégrations, afin de relier Grafana avec notre base InfluxDB :

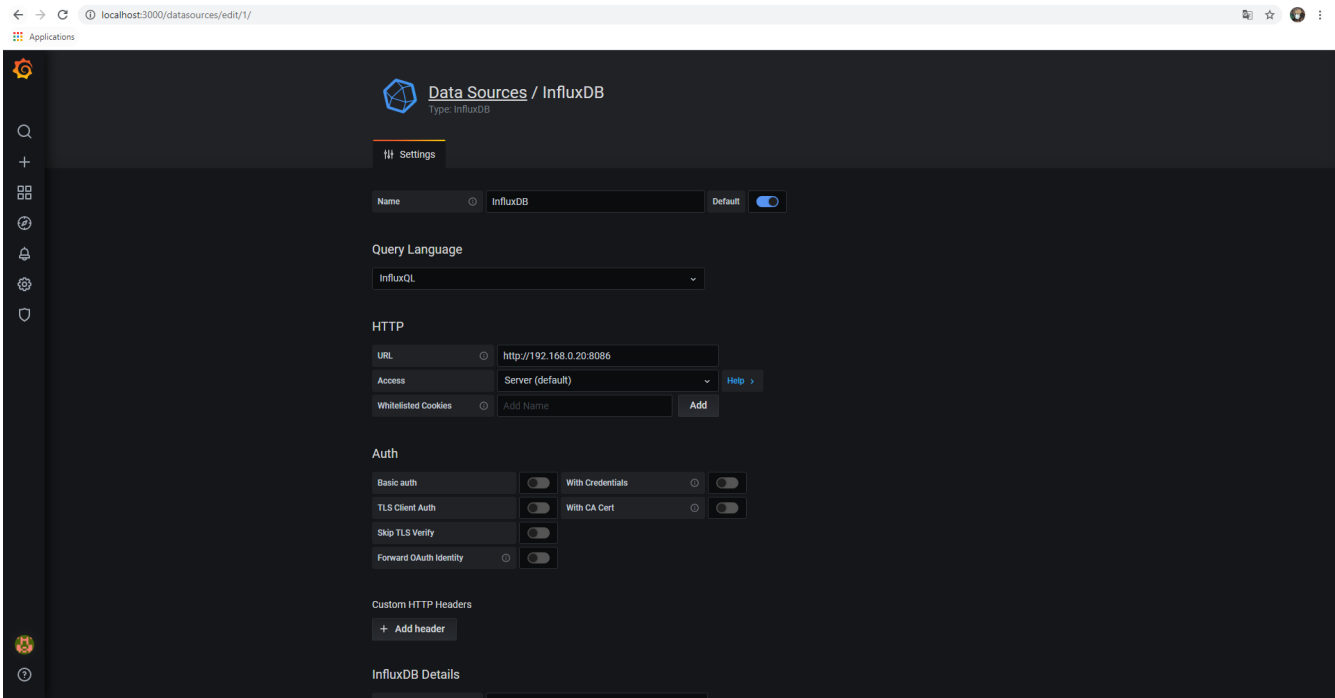


FIGURE 5.29 – Configuration Grafana

Ce genre d'interface nous rappelle beaucoup le serveur ChirpStack, très simple à configurer. On se limite à la configuration de base en ne remplissant que les champs obligatoires pour éviter le problème qu'on a eu précédemment.

Une fois le formulaire rempli, on a cette image qui apparaît en bas de la page pour nous confirmer que Grafana arrive bien à questionner InfluxDB :



FIGURE 5.30 – Data source is working

### 5.3.7 Création du panels

Les panels sont simplement une représentation graphique des données. Grafana va interroger InfluxDB afin de récupérer les données que nous souhaitons. Ces “questions” envoyées par Grafana sont des requêtes SQL, pour cela il y a un éditeur de requêtes. La première étape est de choisir le type de graphique souhaité, puis l’éditeur de requête sera disponible et nous pourrons écrire notre requête.

Dans l’exemple ci-dessous, on désire connaître l’état actuel du capteur : savoir s’il est en position ouverte ou fermée.

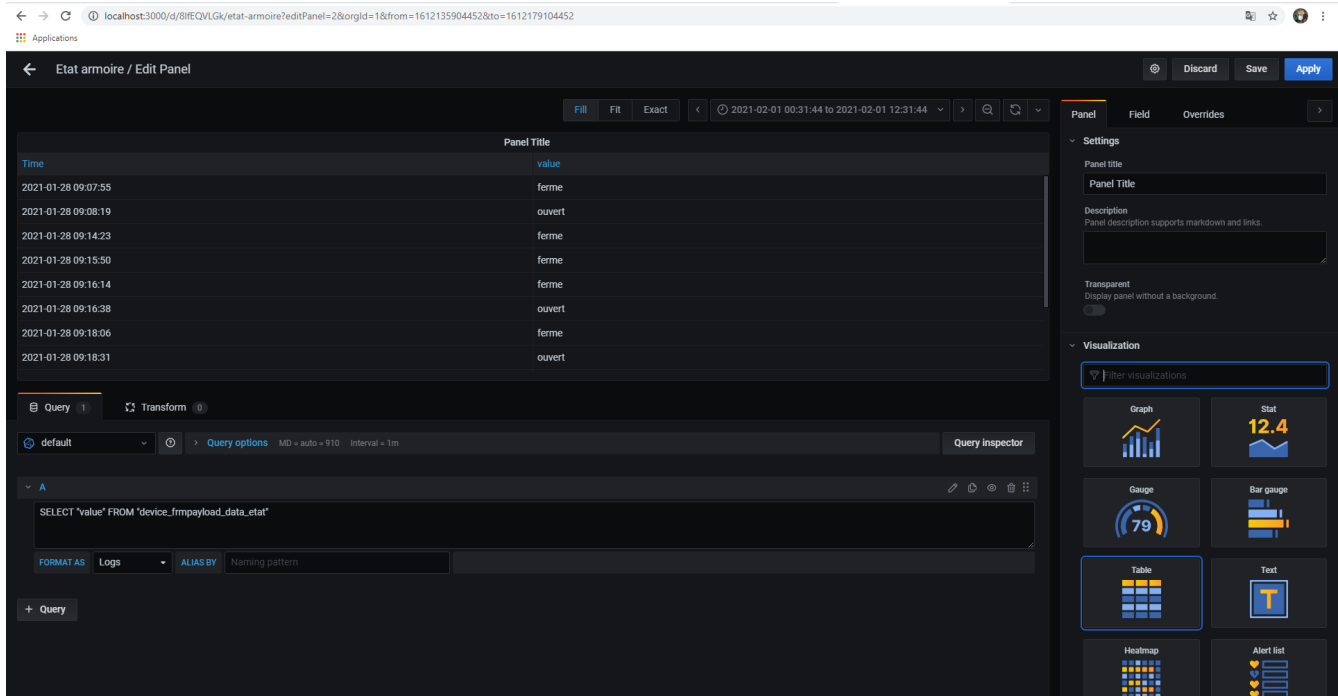


FIGURE 5.31 – Panel

Dans le codec sur chirpstack, nous exploitons la variable “\_frmpayload\_data\_etat” qui contient cette information.

On se retrouve donc avec un tableau à 2 colonnes, la première indique l’heure et la date, et la deuxième indique l’état du capteur. Grafana est très intuitif et permet une grande personnalisation.



Voilà ce que donne cette requête lorsqu'on interroge directement InfluxDB en ligne de commandes :

```
> SELECT "value" FROM "device_fmppayload_data_etat"
name: device_fmppayload_data_etat
time                value
----                -
1611821275000000000  ferme
1611821299000000000  ouvert
1611821663000000000  ferme
1611821750000000000  ferme
1611821774000000000  ferme
1611821798000000000  ouvert
1611821886000000000  ferme
1611821911000000000  ouvert
1611821940000000000  ferme
1611821964000000000  ouvert
1611822015000000000  ferme
1611822040000000000  ouvert
1611823049000000000  ouvert
1611823083000000000  ferme
1611823177000000000  ouvert
1611823205000000000  ferme
1611823233000000000  ouvert
1611823267000000000  ferme
1611823292000000000  ouvert
```

FIGURE 5.32 – Resultat requete en ligne de commande

En partant de ce panel, on peut partir de cette requête SQL dans le but d'avoir juste le dernier état. Pour plus de lisibilité, on pourrait également faire en sorte que ça apparaisse en vert si le capteur est fermé, et rouge en cas d'ouverture. Pouvoir gérer tous ces paramètres en mode graphique est simple et intuitif et permet à n'importe qui de pouvoir configurer Grafana pour ses besoins, rien qu'en cherchant un peu sur internet.

Après avoir regardé les différents graphiques disponibles, nous avons réussi à créer un graphique représentant l'état du capteur au cours du temps. On peut facilement imaginer ce genre de panel sur un écran de monitoring.

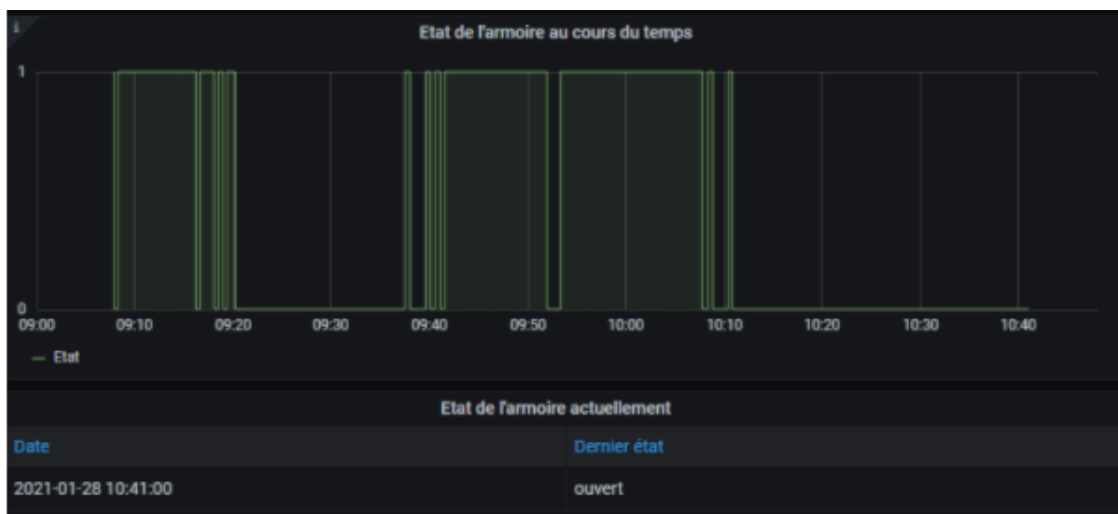


FIGURE 5.33 – Panel final

# Partie 6

## Bilan

### 6.1 Conclusion

Nous avons tous les trois réalisé individuellement des projets lors du DUT R&T. Enrichis de cette expérience, l'ensemble de notre projet, qui a duré de septembre 2020 à février 2021, s'est bien déroulé. La force de notre groupe est qu'à la base, nous avons tous les trois choisi ce sujet car il nous intéressait. En effet, pouvoir travailler sur une technologie LoraWan et son matériel (capteur, gateway) est une réelle chance car cette technologie est en pleine expansion et s'inscrit parfaitement dans la lignée du développement des villes intelligentes (avec une régulation autonome de la consommation d'énergie, des routes produisant de l'énergie lorsqu'on roule dessus). Bien qu'apparue assez récemment, la technologie LoraWan et plus généralement les technologies radio basses fréquences sont amenées à se développer rapidement. Dès la présentation du sujet, nous avons individuellement décidé de choisir celui-ci, c'est une des raisons pour laquelle ce projet s'est déroulé sans grosse difficulté et dans une bonne ambiance de travail. Ce projet mêlait à la fois des compétences que nous avons développées auparavant, comme l'installation et la configuration de serveurs, et également de nouvelles compétences, la principale nouveauté du projet concernant les trames LoraWan. Une grosse différence entre le projet de DUT et celui-ci est que cette année le projet était encadré par deux enseignants du lycée Malraux, qui nous ont accompagné et aidé activement lorsqu'un point posait problème et qu'on ne trouvait pas la solution de nous-même.

Nous avons eu à réaliser un travail de documentation afin d'assimiler le fonctionnement de la technologie LoraWan et plus particulièrement comment interpréter les données que le capteur envoie, comment faire pour visualiser ces données. Comme l'installation/configuration de technologie Linux s'est déroulée sans souci, nous avons pu nous focaliser sur ce point. L'inconvénient majeur du projet était que le matériel restait au lycée, nous devons donc optimiser nos séances de quatre heures pour réaliser nos tests/installations et faire le travail de recherche chez nous. Nous avons gagné en autonomie et nous avons apprivoisé une technologie dont nous ne connaissions que vaguement la théorie. Le fait de pouvoir compter les uns sur les autres nous a beaucoup aidé aussi. Bien que tous les trois sortis d'un DUT R&T, comme nous venions chacun d'un établissement différent (Béthune, Blois et Poitiers), nous avons chacun un domaine de compétence différent donc nous sommes une équipe où l'on se complétait bien.

Nous avons tous les trois apprécié de réaliser ce projet, et c'est cette collaboration qui a permis de remplir les objectifs du projet.

# Glossaire

- BDD** Une base de donnée est un outil de stockage d'information structuré sous forme de table par exemple. .... 16, 27
- DNS** Domain name service, va permettre l'action de traduire un nom de domaine en adresse IP. . . 14
- LoRaWAN** Protoicole de télécommunication bas debit, très répendu pour l'internet des objets, qui permet à une multitude d'object connecté de communiqué. .... 3, 4
- PM** Un point de mutualisation est le premier élément qui va permettre de faire le lien avec le noeud de raccordemen t optique de l'opérateur télécom. .... 3, 4, 6, 21, 26
- PuTTY** Logiciel de prise en main qui regroupe les connexion SSH, Telnet, Rlogin et du Serial. . . . 11, 13, 15
- SSH** Protocole de prise en main à distance sécurisé qui vise à remplacer le TELNET, Rlogin. . 11, 13, 26
- trame** Un bloc d'information qui va véhiculé au travers d'un support physique. .... 22