MA_IoT

Supervision d'une ruche

2024-12-18

Adrien Rey & Lucien Jeandupeux
HES-SO Master
2024-2025
Rapport

Table des matières

1 Introduction

Ce projet est réalisé dans le cadre du cours MA_IoT du master MSE de l'HES-SO. Il a pour but de créer un prototype d'appareil pouvant superviser une ruche d'abeilles. Il doit être capable, à l'aide de capteurs, de monitorer la température, l'humidité, la position GPS et de détecter le mouvement de la ruche. Il doit aussi Les données sont envoyées à un serveur qui les stockera et les affichera sur une interface web. Les services fournis par AWS doivent être utilisés pour le stockage.

La communication entre l'appareil et le serveur se fait via le protocole LoraWan. Elle peut se faire soit un réseau *The Things Network* soit un réseau propriétaire (*AWS IoT LoRaWAN*). Pour les ruches à proximité d'une habitation, une solution par Wi-Fi est proposée à la place.

1.1 Objectifs

1. Un dashboard permettra de visualiser l'évolution des valeurs et de notifier les alarmes.
2. La communication par LoRa avec un cloud pourra se faire soit avec un réseau TTN soit avec un réseau propriétaire (AWS IoT LoRaWAN).
3. À distance, il devra être possible de bloquer les abeilles à l'intérieur.
4. Le déploiement de masse de ruches et la gestion multi-utilisateur devront être envisagés.
5. Pour les ruches à proximité d'une habitation, une solution par Wi-Fi sera proposée.
6. D'autres capteurs pourront être ajoutés pour remonter plus d'informations.

1.2 Rendu attendu

- La solution devra être déployée dans un cloud en utilisant des solutions de type SaaS
- Le système déployé devra être "scalable" avec la gestion de plusieurs ruches
- Une analyse des coûts devra être faite en imaginant 3 types de tailles : small (100 devices), medium (5000 devices), large (100'000 devices)
- Une étude sur une solution compatible "disaster recovery" est demandée

Tous les fichiers nécessaires pour le projet sont disponibles sur le dépôt GitHub suivant : https://github.com/luciennnnnnnn/MA_IoT.

2 Spécifications

Dans ce point, la description des besoins et des contraintes du projet est détaillée.

2.1 Hardware

2.1.1 Cartes

Deux versions de l'appareil sont demandées. La première version est destinée aux ruches éloignées d'une habitation et la communication se fera par LoRaWan. La seconde version est destinée aux ruches proches d'une habitation et la communication se fera par Wi-Fi. Les cartes suivantes ont été fournies pour permettre le développement de l'appareil.

Composant	***Numéro de série***
ESP32	113991114
Grove Shield pour Seeeduino XIAO	103020312

Composant	***Numéro de série***
Carte de développement Wio LoraWAN GPS Tracker	40790830

2.1.2 Capteurs

Ce device doit permettre de mesurer diverses valeurs importantes pour l'apiculteur. Les données suivantes ont été retenues pour ce projet et devront être envoyées à un serveur pour être traitées et affichées sur une interface web :

- **Température**: la température interne de la ruche est cruciale, car les abeilles maintiennent une température optimale (environ 34-36°C) pour le développement du couvain (larves). Une variation peut indiquer un problème (ex. absence de reine ou environnement défavorable).
- **Humidité** : l'humidité influe sur la santé des abeilles et la qualité du miel. Un taux trop élevé favorise les moisissures et nuit au miel, tandis qu'un taux trop faible peut perturber le développement des larves.
- **Position GPS** : le suivi GPS permet de localiser la ruche en cas de vol (un problème courant pour les apiculteurs).
- **En déplacement (bouge, statique)** : une ruche qui se déplace peut indiquer un vol ou un accident (chute, déplacement involontaire). Cette donnée va de pair avec la position GPS.
- **État de la porte (ouverte, fermée, en mouvement)** : comme la porte d'accès des abeilles doit être pilotable, il est important de savoir si elle est ouverte ou fermée.

Les capteurs suivants sont nécessaires pour mesurer ces valeurs :

Composant	***Numéro de série***
Capteur d'humidité et de température	101990644
Accéléromètre (en déplacement)	114020121
Capteur de distance (état porte)	101020532
Capteur switch (état porte)	102020143

La carte LoRaWAN intègre un capteur de température et d'humidité et un accéléromètre. Pour simplifier le développement, des capteurs externes communiquant en SPI vont quand même être utilisés. Cela facilitera le remplacement des capteurs en cas de panne et permet aussi de développer un software générique qui peut être utilisé avec les deux types de cartes (LoRaWAN et Wi-Fi). Seule la couche de communication sera différente. Pour interfacer tous ces capteurs, un hub I2C est utilisé.

Composant	***Numéro de série***
Hub I2C (6 ports)	103020272

Les cartes fournies ne disposent pas d'un module GPS permettant de récupérer la position de la ruche. Le module LoRaWAN intègre un module GPS, mais il nécessite de passer par une API pour récupérer les données. Pour ce prototype, les données de position GPS seront simulées

pour le module Wi-Fi. La position de la gateway LoRa utilisée pour la communication sera utilisée pour définir la position de la ruche avec la carte LoRaWAN.

2.1.3 Actionneurs

Pour piloter la porte de la ruche, un servomoteur est utilisé. Il permet d'ouvrir et de fermer la porte.

Composant	***Numéro de série***
Servo moteur	316010005

3 Architecture

Le système mis en place est représenté dans le schéma ci-dessous. Chaque section va être détaillée dans les points suivants. Les détails techniques sont eux détaillés dans les sections suivantes.

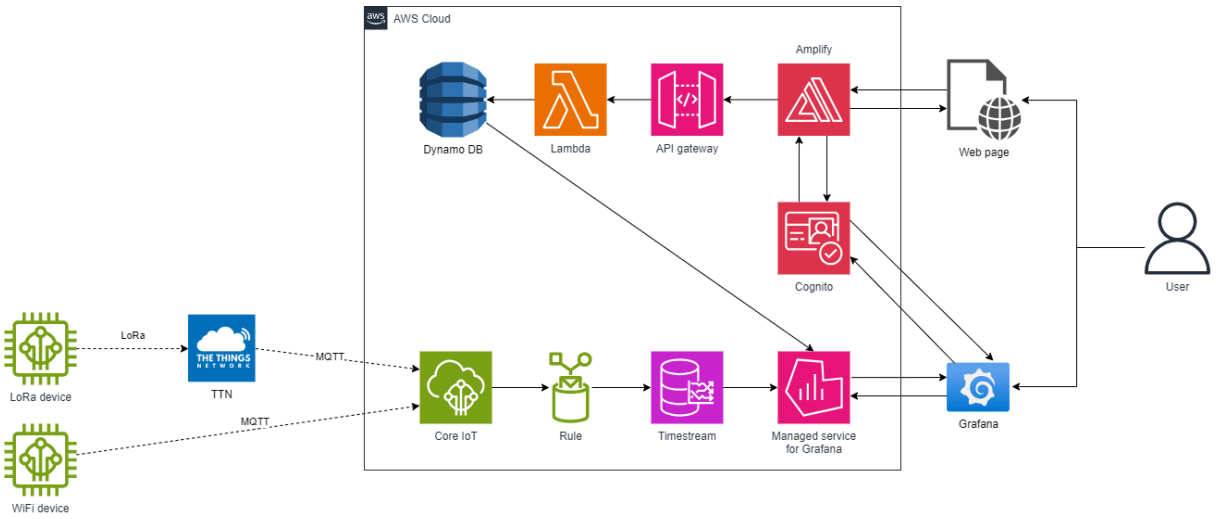

Fig. 1 — Architecture projet

3.1 Communication

La communication se fait entre un appareil et un serveur. Pour ce faire, *AWS Core Iot* est utilisé pour connecter les devices à l'environnement AWS. Un *things* par device est créé dans Core IoT.

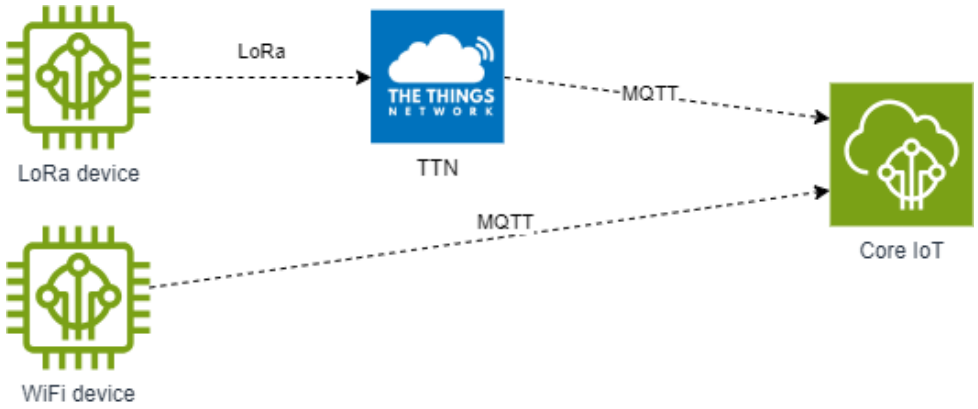

Fig. 2 — Communciation

Un appareil WiFi va directement envoyer ces données à *AWS Core IoT* via le protocole MQTT. Pour les appareils LoRaWAN, la communication se fait via un réseau LoRaWAN. Deux solutions étaient disponibles. La première est de passer par *The Things Network* (TTN) et la seconde est de passer par *AWS IoT LoRaWAN*. La première solution a été choisie.

Elle permet de simplifier le processus de mise en place de l'appareil pour l'utilisateur. Il lui suffit de placer l'appareil dans la ruche et de l'alimenter. Il n'a pas besoin d'acheter et d'installer une gateway LoRaWAN. Il existe deux points négatifs à cette solution. Les données sont envoyées à un serveur TTN. Ce serveur va ensuite envoyer les données à *AWS Core IoT*. Cela peut poser des problèmes de confidentialité des données. Le nombre de messages envoyés par mois est aussi limité. Pour un usage professionnel, il faudra payer pour augmenter ce nombre. De plus, la couverture fournie par TTN n'est pas garantie dans toutes les régions. Un autre réseau LoRaWAN pourrait être utilisé à la place, comme par exemple *Swisscom LPN*. Il permettrait de garantir une couverture en Suisse et de ne pas avoir besoin d'installer de gateway LoRaWAN, mais il engendrerait des coûts supplémentaires.

Le guide suivant a été utilisé pour connecter les devices à *AWS Core IoT* via TTN: Connecting The Things Network to AWS IoT

3.2 Stockage des données

Les données envoyées par les appareils sont stockées dans une base de données *Timestream*. Cela permet d'avoir une grande quantité de données stockées et de les traiter rapidement. Une règle *IoT Core* est utilisée pour envoyer les données reçues par *Core IoT* à *Timestream*.

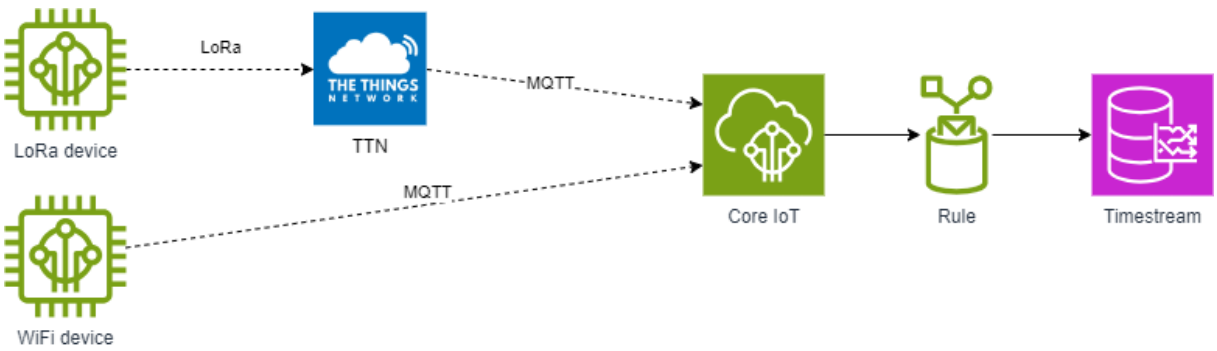

Fig. 3 — Stockage des données

La structure des données stockées dans *Timestream* et la règle *IoT Core* sont détaillées dans les points suivants.

3.3 Visualisation des données

Pour la visualisation des données, *Grafana* est utilisé. Il permet de créer des tableaux de bord pour afficher les données stockées dans *Timestream*. Cette solution est simple à mettre en place et permet de visualiser les données en temps réel. Cela convenait à ce projet, car le but n'était pas de coder une page web ou une application mobile. Si le produit devait être vendu, une solution plus personnalisée pourrait être envisagée, comme une page web ou alors une application mobile.

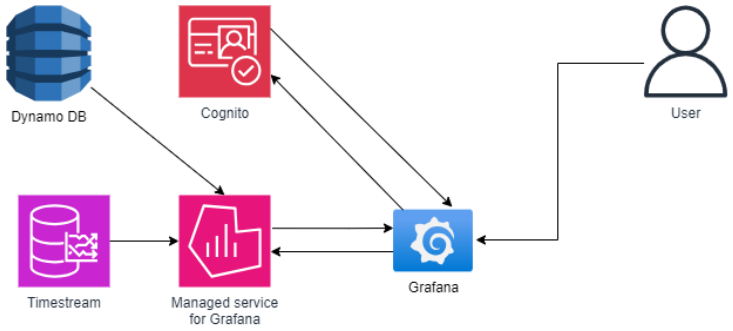

Fig. 4 — Visualisation des données

Pour ajouter une couche de sécurité, *Cognito* est utilisé pour gérer les utilisateurs. Chaque utilisateur doit s'authentifier pour accéder à l'interface de visualisation des données. Il peut alors seulement visualiser les données des appareils qu'il a enregistrés. Ces informations sont stockées dans *DynamoDB*.

3.4 Gestion des utilisateurs et appareils

Comme mentionné dans le point précédent, *Cognito* est utilisé pour gérer les utilisateurs. Il permet aux utilisateurs de créer un compte et de s'authentifier. Les clients peuvent alors visualiser les données de leurs appareils. Pour faire le lien entre les appareils et les utilisateurs, l'utilisateur doit renseigner l'ID de l'appareil lors de l'ajout de l'appareil à son compte.

Pour ce faire, une page web React permet, après s'être identifié, d'ajouter un numéro d'appareil à son compte. Cette page web est hébergée par *Amplify*.

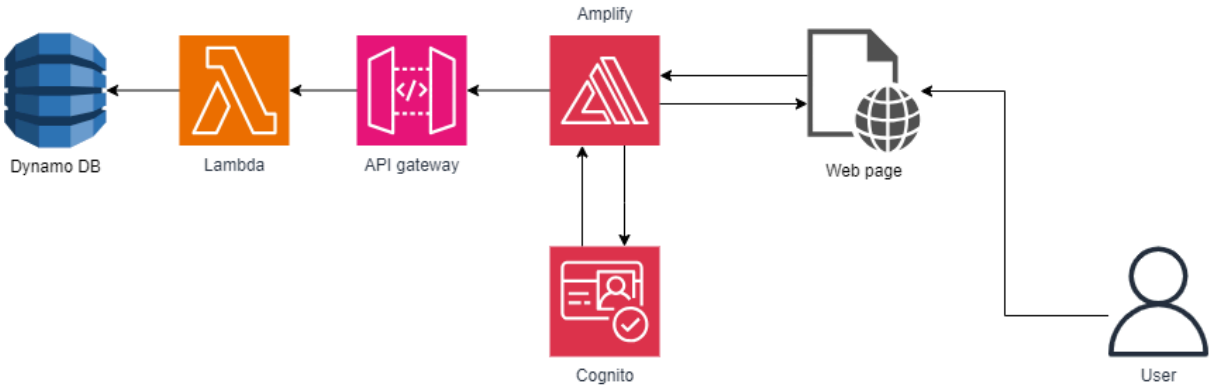

Fig. 5 — Gestion des utilisateurs et appareils

Pour ajouter un appareil dans la base de données *Dynamo DB*, un *Lambda* est utilisé. Il permet de vérifier si l'appareil existe déjà dans la base de données et de l'ajouter si ce n'est pas le cas. Ce *Lambda* est appelé par l'*API Gateway* qui met à disposition une méthode POST pour ajouter un appareil. Un autre *Lambda* est utilisé pour récupérer les appareils d'un utilisateur. Il est appelé par l'*API Gateway*, qui met à disposition une méthode GET pour récupérer les appareils.

4 Software

Cette section décrit les librairies logicielles nécessaires et les étapes de déploiement du projet HiveMonitoring pour les versions LoRaWAN et Wi-Fi. Ces deux systèmes partagent des bases communes, mais diffèrent dans la communication et le traitement des données.

4.1 Librairies nécessaires

Les librairies utilisées permettent de gérer les capteurs, la communication réseau (LoRaWAN ou Wi-Fi), et l'envoi des données vers le cloud AWS.

Librairie	**Description**	**Utilisation**
WiFi	Gestion de la connexion Wi-Fi.	Version Wi-Fi : Connexion au réseau.
WiFiClientSecure	Gestion des connexions sécurisées via TLS.	Connexion sécurisée à AWS IoT Core.
PubSubClient	Gestion du protocole MQTT pour l'envoi/réception des données.	Publication des données et souscription.
ArduinoJson	Sérialisation/désérialisation des données JSON.	Formatage des payloads JSON.
CayenneLPP	Format standard pour l'encodage binaire des données LoRaWAN.	Encodage des données version LoRaWAN.
LoRa	Gestion de la communication LoRa pour les appareils longue portée.	Transmission avec réseau TTN.

4.2 Paramètres Arduino

Régler la communication à 115'200Bauds

- Pour l'utilisation de la carte LoraWan

Aller dans Arduino -> file -> préférences -> Additional boards manager et coller l'URL qui suit : https://files.seeedstudio.com/arduino/package_seeeduino_boards_index.json

- Pour l'utilisation de la carte WiFi

Aller dans Arduino -> file -> préférences -> Additional boards manager et coller l'URL qui suit : https://dl.espressif.com/dl/package_esp32_index.json

4.3 Configuration générale du projet

a. Configuration Wi-Fi (version Wi-Fi uniquement) La version Wi-Fi utilise une connexion sécurisée TLS pour se connecter à AWS IoT Core. Voici les étapes :

1. Configurer les informations du réseau Wi-Fi : dans le fichier aws_config.cpp :

```
++
const char* ssid = "votre-SSID";            // Nom du réseau Wi-Fi
const char* password = "votre-mot-de-passe"; // Mot de passe
```

2. Ajouter les certificats AWS :

Les certificats TLS sont nécessaires pour établir une connexion sécurisée :

```
++
const char* certificate = R"EOF( ... )EOF";
const char* private_key = R"EOF( ... )EOF";
const char* root_ca = R"EOF( ... )EOF";
```

Ces certificats sont fournis par AWS IoT Core lors de l'enregistrement de l'appareil.

3. Connexion à AWS IoT Core : La fonction connectAWS() gère la connexion et l'abonnement aux topics MQTT :

```
++
void connectAWS() {
    while (!client.connected()) {
        if (client.connect(ThingName)) {
            client.subscribe(subscribeTopic);
        }
    }
}
```

4. Envoi des données en JSON :

Les données des capteurs sont insérées dans un document JSON via la fonction dataToSend() :

```
++
StaticJsonDocument<256> dataToSend() {
    StaticJsonDocument<256> doc;
    doc["is_moving"] = dernierEtatMouvement;
    doc["door"] = etatDistance;
    doc["temperature"] = temperature;
    doc["humidity"] = humidity * 100;
```

```
    doc["device_id"] = ThingName;
    return doc;
}
```

Les données sont ensuite publiées sur le topic MQTT configuré :

```
++
client.publish(publishTopic, payload);
```

b. Configuration LoRaWAN (version LoRa uniquement) La version LoRaWAN utilise le protocole LoRa pour transmettre des données via The Things Network (TTN) avant d'arriver sur AWS IoT Core.

1. Encodage des données : Les données sont encodées au format Cayenne LPP. Voici un exemple :

   ```
   ++
   CayenneLPP lpp;
   lpp.addTemperature(1, temperature);
   lpp.addRelativeHumidity(2, humidity);
   lpp.addDigitalInput(3, doorState);
   ```

2. Transmission des données : La fonction sendPayload() permet d'envoyer les données via LoRa :

   ```
   ++
   LoRa.beginPacket();
   LoRa.write(lpp.getBuffer(), lpp.getSize());
   LoRa.endPacket();
   ```

3. Connexion à TTN : Le Device EUI, App Key, et App EUI doivent être configurés pour authentifier l'appareil avec TTN.

4.4 Pinout

4.4.1 WiFi

Composant	Connexion
Hub I2C (6 ports)	Connecté aux broches I2C de la carte : SDA (GPIO 4) et SCL (GPIO 5)
Capteur de température/humidité (AHT20)	Connecté à l'un des ports I2C du hub
Capteur de distance (VL53L0X)	Connecté à l'un des ports I2C du hub
Accéléromètre (LIS3DHTR)	Connecté à l'un des ports I2C du hub

Composant	Connexion
Servo moteur (pour la porte)	Connecté à la broche GPIO 1 pour le signal PWM
Switch (état de la porte)	Connecté à la broche GPIO 0 pour la lecture de l'état du switch
Alimentation	Capteurs, hub I2C et switch alimentés via le 3.3V et GND de la carte microcontrôleur.

4.4.2 LoraWAN

Composant	Connexion
Hub I2C (6 ports)	Connecté aux broches I2C de la carte : SDA (GPIO 5) et SCL (GPIO 4)
Capteur de température/humidité (AHT20)	Connecté à l'un des ports I2C du hub
Capteur de distance (VL53L0X)	Connecté à l'un des ports I2C du hub
Accéléromètre (LIS3DHTR)	Connecté à l'un des ports I2C du hub
Servo moteur (pour la porte)	Connecté à la broche GPIO 13 pour le signal PWM
Switch (état de la porte)	Connecté à la broche GPIO 15 pour la lecture de l'état du switch
Alimentation	Capteurs, hub I2C et switch alimentés via le 3.3V et GND de la carte microcontrôleur.

4.5 Étapes de déploiement

4.5.1 Version Wi-Fi

1. Installer les librairies nécessaires :
   - Ajoutez les librairies via le gestionnaire de bibliothèque d'Arduino IDE.
2. Configurer AWS IoT Core :
   - Enregistrez votre appareil comme un « Thing ».
   - Téléchargez les certificats (Root CA, clé privée, certificat client).
3. Modifier aws_config.cpp :
   - Remplissez les informations Wi-Fi et les certificats AWS.
4. Compiler et téléverser :
   - Chargez le code sur votre carte ESP32.

5. Tester la connexion :
    - Vérifiez les messages MQTT publiés dans AWS IoT Core.

4.5.2 Version LoRaWAN

1. Configurer TTN :
   - Créez une application et enregistrez vos appareils.
   - Récupérez les informations Device EUI, App Key, et App EUI.
2. Configurer les paramètres LoRa :
   - Modifiez les identifiants dans le fichier .ino.
3. Ajouter la librairie CayenneLPP :
   - Installez cette librairie pour encoder les données.
4. Compiler et téléverser :
   - Téléchargez le code sur la carte compatible LoRa (e.g., Seeed Wio Tracker).
5. Tester l'envoi des données :
   - Surveillez les messages dans la console TTN.

4.6 Résumé des configurations

Version	Paramètres principaux	Services requis
Wi-Fi	SSID, mot de passe, certificats AWS, endpoint MQTT	AWS IoT Core, MQTT, TLS
LoRaWAN	Device EUI, App Key, App EUI, encodage Cayenne LPP	TTN, AWS IoT Core, MQTT

4.7 Outils recommandés pour le déploiement

- Arduino IDE : Développement et téléversement du code.
- Serial Monitor : Débogage local pour visualiser les messages envoyés.
- AWS IoT Core : Surveillance des messages MQTT pour valider la transmission.
- The Things Network : Suivi des messages pour la version LoRaWAN.

En suivant ces étapes, les deux versions peuvent être déployées de manière efficace pour la supervision des ruches.

4.8 État actuel

Les données remontent jusqu'au grafana avec une connexion Wifi ou une connexion via LoraWan. La communication des données de l'utilisateur jusqu'aux ruches ne peut actuellement que se faire depuis AWS IoT Core et uniquement sur Arduino. En effet, le système utilisant une interface grafana, prévu pour de l'affichage de graphique, il est plus compliqué de communiquer dans l'autre sens. Dans le cas idéal, il y aurait une application Web permettant de jouer avec tous les paramètres et communications.

4.8.1 Éteindre

En communiquant via AWS IoT Core, il est possible d'allumer et d'éteindre la ruche équipée d'un ESP32. Cependant, cette fonctionnalité, ajoutée tardivement, n'a pas été implémentée sur les cartes LoraWan car elle reste difficile à tester sans passerelle aux alentours.

Allumer :

- Il suffit d'envoyer **ON** en précisant l'appareil de réception (subscribe/« NOM DU DEVICE »/downlink)
- La ruche reprend dans son précédent mode

Éteindre :

- Il suffit d'envoyer **OFF** en précisant l'appareil de réception (subscribe/« NOM DU DEVICE »/downlink)
- La ruche continue d'envoyer des données au grafana mais ferme la porte et empêche son ouverture

4.8.2 Modes

Le bouton « B » de l'ESP32, permet de passer du mode manuel au mode automatique. Les fonctions sont aussi incluses dans la carte LoraWan mais la datasheet n'indique pas clairement la pin du bouton à utiliser. C'est pourquoi la carte LoraWan reste en mode automatique. Mode manuel :

- Permet d'ouvrir et fermer la porte avec le switch
- Ne permet pas d'ouvrir et fermer la porte avec la température

Mode automatique :

- Permet d'ouvrir et fermer la porte, si la température n'est pas bloquante
- Permet de fermer et ouvrir la porte en fonction de la température
  - Si la température est trop élevée, la ruche se ferme
  - Si la température redescend au-dessous d'un certain seuil, la ruche s'ouvre

4.8.3 Uplink

Les fonctionnalités de collecte et de remontée de données suivantes sont opérationnelles :

- ID ruche
- Température
- Humidité
- En déplacement : détection des états bouge ou statique.
- État de la porte : ouvert, fermé ou en mouvement.

Concernant la position GPS, les fonctions sont prévues, mais non encore implémentées complètement. Cependant, des positions GPS apparaissent déjà dans Grafana, indiquant que seule l'implémentation bas niveau reste à finaliser.

4.8.4 Downlink Allumer/Éteindre la ruche

En l'absence d'un site web dédié à l'application et sachant que Grafana est principalement conçu pour une utilisation uplink, le downlink est actuellement limité à la communication entre AWS IoT Core et l'Arduino. Les fonctionnalités disponibles sont les suivantes :

- Allumer/Éteindre la ruche (ON/OFF).

Lorsque la ruche est éteinte, les données continuent à remonter jusqu'au grafana mais les contrôles de la porte sont désactivés et la porte se ferme.

Procédure d'utilisation

1. Accéder à AWS IoT Core :
2. Connectez-vous à la console AWS.

3. Accédez à la section AWS IoT Core depuis le tableau de bord.
4. Utilisez le format suivant pour le topic MQTT

```
subscribe/"NOM DU DEVICE"/downlink
```

5. Envoyer une commande Downlink :

Créez un message MQTT destiné à l'Arduino. Le message doit spécifier les commandes suivantes :

- Pour allumer/éteindre la ruche :

```
ON  // Pour allumer
```

```
OFF // Pour éteindre
```

Exemple :

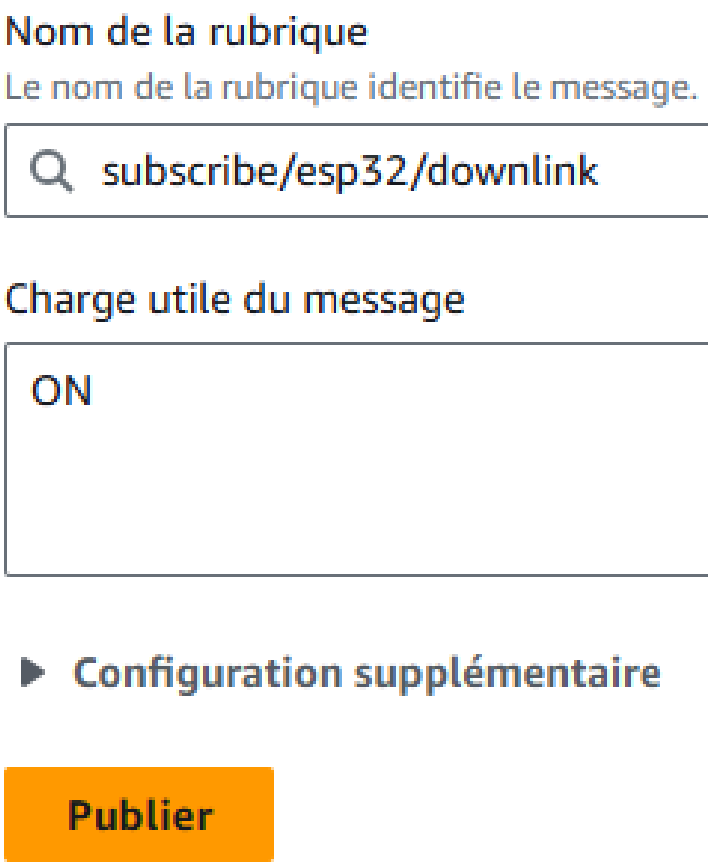

Fig. 6 — Exemple de downlink

Cette fonctionnalité, uniquement testée par Wifi, n'est pas implémentée pour LoraWan.

5 The Things Network

Une seule application est utilisée pour tous les devices, toujours dans le but de faciliter la mise en service de nouveaux appareils. Les devices sont identifiés par leur EUI. L'utilisateur n'a pas besoin de créer une application ou d'avoir un compte *TTN*, il doit seulement connaître l'ID de son appareil (EUI) pour visualiser les données.

Les données envoyées par les devices sont stockées dans *TTN* et envoyées à *Core IoT* via MQTT. Les données envoyées par les devices sont encodées en *Cayenne LPP*. Le formateur de données *CayenneLPP* fourni par *TTN* est utilisé pour décoder les données.

L'application est liée à *AWS IoT Core* en utilisant un template *AWS CloudFormation* existant (Guide de déploiement). Une fois, cette liaison faite, lorsqu'un nouvel appareil est ajouté à *TTN*, il est automatiquement ajouté à *Core IoT*. Cela permet de simplifier la gestion des devices et de ne pas avoir à ajouter manuellement les devices à *Core IoT*.

6 Cloud

6.1 Core IoT

Les things ajoutées dans *Core IoT* sont les appareils qui envoient les données. Ils sont identifiables à l'aide de leur ID. Les données envoyées sont envoyées via MQTT.

6.2 Payload TTN

Le payload envoyé par les devices LoRaWAN est encodé en *Cayenne LPP*. C'est un format de payload standardisé pour les devices LoRaWAN. Il permet de définir des types de données et de les encoder de manière standardisée. Les données envoyées par les devices sont encodées en binaire et envoyées à *TTN*. *TTN* se charge de décoder les données et de les envoyer à *Core IoT* via MQTT.

Les données importantes et utiles pour le traitement sont les suivantes:

```
  "device_id": "wio-tracker-1110-arey",     // ID de l'appareil
  "dev_eui": "2CF7F1F061900067"            // EUI de l'appareil (ID utilisé
pour l'identification)
  "temperature_4": 0                        // Température
  "relative_humidity_3": 0                  // Humidité
  "digital_in_2": 0                         // Etat porte
  "digital_in_1": {"x": 0.0,"y": 0.0,"z": 0.0}  // Accéléromètre (mouvement)
  "received_at": "2024-11-25T16:32:03.740026534Z"     // Date de réception
```

Une règle *IoT Core* est utilisée pour envoyer les données reçues par *Core IoT* à *Timestream*. Elle permet de récupérer les données utiles listées ci-dessus.

```
SELECT uplink_message.decoded_payload.accelerometer_1 AS is_moving,
uplink_message.decoded_payload.digital_in_2 AS door_state,
uplink_message.decoded_payload.relative_humidity_3 AS relative_humidity,
uplink_message.decoded_payload.temperature_4 AS temperature,
FROM 'lorawan/+/uplink'
```

Cette règle permet de récupérer les données de tous les appareils lorawan sans distinction de l'utilisateur.

6.3 Payload Wi-Fi

Les données envoyées par les appareils utilisant la version Wi-Fi sont encapsulées dans un document JSON. Ce format est léger, lisible, et idéal pour la transmission via MQTT. Chaque appareil collecte les données des capteurs et les organise dans un document JSON avant de les publier sur le serveur MQTT.

Les principales étapes incluent :

Construction des données : Les données des capteurs sont collectées et insérées dans un document JSON. Sérialisation et envoi : Ce document est sérialisé en une chaîne de caractères JSON avant d'être envoyé au topic MQTT défini. Structure des données envoyées Le payload JSON envoyé par les appareils Wi-Fi contient les éléments suivants :

```
{
    "device_id": "hive-wifi-001",       // ID unique de l'appareil
    "is_moving": false,                 // État de mouvement (booléen)
    "door": 0,                          // État de la porte (0 = fermée, 1
= ouverte)
    "temperature": 25.6,                // Température (en °C)
    "humidity": 45.5                    // Humidité (en %)
}
```

Description des champs :

- device_id : Identifie de manière unique l'appareil dans le système. Sert à associer les données aux utilisateurs et aux ruches correspondantes.
- is_moving : Indique si la ruche est en mouvement. 1 : Statique, 2 : En mouvement.
- door : Indique l'état de la porte. 1 : Fermée, 2 : En Mouvement, 3 : Ouverte.
- temperature : Température mesurée à l'intérieur de la ruche en degrés Celsius. Une température trop élevée peut femrer les portes si le mode automatique est activé.
- humidity : Taux d'humidité mesuré dans la ruche, exprimé en pourcentage. Une humidité trop basse ou trop élevée peut indiquer un problème.

Une règle *IoT Core* est utilisée pour envoyer les données reçues par *Core IoT* à *Timestream*. Elle permet de récupérer les données utiles listées ci-dessus.

```
SELECT is_moving AS is_moving, door AS door, temperature AS temperature,
humidity AS humidity, FROM 'publish/+/uplink'
```

Cette règle permet de récupérer les données de tous les appareils wifi sans distinction de l'utilisateur.

6.4 Différences entre les payloads LoraWAN et WiFi

	LoRaWAN	**Wi-Fi**
Format de données	Cayenne LPP	JSON
Transmission	Encodé en binaire	Chaîne JSON
Protocole	LoRaWAN + MQTT via TTN	MQTT directement
Scénarios d'usage	Longue portée, faible consommation	Courte portée, connexion stable

6.5 Timestream

Une base de données *Timestream* est utilisée pour stocker les données. C'est une base de données de séries temporelles qui permet de stocker et de traiter des données temporelles.

Une base de données est créée pour le système (hivesData). Les données sont stockées dans des tables. Une seule table est créée pour tous les devices (table). Cela permet d'avoir un système plus simple à gérer. Il suffit de gérer les droits et requêtes du côté des utilisateurs. La règle pour insérer des données reste la même pour tous les devices et les requêtes aussi (hormis le filtrage des données par device).

Les données sont stockées dans des colonnes. Les colonnes sont les données envoyées par les devices. Les données sont stockées en fonction de leur timestamp sous la forme suivante.

deviceID	***measure_name***	***time***	***measure_value::bigint***
2CF7F1F061900067	temperature	2024-12-08 14:44:20.264000000	23

Les données peuvent donc être récupérées en fonction de leur timestamp, de l'ID de l'appareil et du type de mesure. Cela permet de récupérer les données d'un seul device ou de tous les devices en fonction des droits de l'utilisateur.

6.6 Cognito

Pour la partie gestion des utilisateurs, *Cognito* est utilisé. Il permet de gérer les utilisateurs et de les authentifier. Ce service vous permet de gérer des identités et des accès des clients (IAM) dans vos applications web et mobiles. Un user pool est utilisé pour regrouper tous les clients utilisant le service.

6.7 DynamoDB

Un des problèmes à résoudre lors de ce projet fut de pouvoir lier facilement l'utilisateur à son ou ses appareils. Le but étant de minimiser les manipulations requises par le client pour faciliter l'expérience utilisateur de ce dernier.

Pour ce faire, la solution retenue est de stocker les devices de chaque utilisateur dans une base de données *DynamoDB*. Comme pour *Timestream*, une seule table est utilisée pour stocker les devices de tous les utilisateurs (devicesIDTable). Les données sont stockées sous la forme suivante.

Partition key (UserID)	***Sort key (DeviceID)***
jean.michel@gmail.com	2CF7F1F061900067

Les données sont donc stockées en fonction de l'ID de l'utilisateur et de l'ID de l'appareil. Cela permet de récupérer les appareils d'un utilisateur en fonction de son ID.

Pour écrire ou lire des données dans *DynamoDB*, des *Lambda* sont utilisés. Ils sont détaillés dans le point suivant.

6.8 Lambda

Deux *Lambda* sont utilisés dans ce projet : un pour ajouter un appareil à la base de données *DynamoDB* et un pour récupérer les appareils d'un utilisateur.

6.8.1 deviceRegistrationLambda

Ce *Lambda* est utilisé pour ajouter un appareil à la base de données *DynamoDB*. Il est appelé par l'*API Gateway*, qui met à disposition une méthode POST pour ajouter un appareil. Il est donc trigger par l'*API Gateway*. Le *Lambda* vérifie si la paire identifiant utilisateur/identifiant appareil existe déjà. Si ce n'est pas le cas, il l'ajoute.

6.8.2 getDevices

Ce *lambda* est utilisé pour récupérer les appareils d'un utilisateur. Il est appelé par l'*API Gateway*, qui met à disposition une méthode GET pour récupérer les appareils. Le *Lambda* récupère les appareils en fonction de l'ID de l'utilisateur passé en paramètre de la requête.

Le service utilisé dans grafana pour appeler la méthode GET (Infinity Data Source) a un bug pour le traitement des données reçu avec la version 10.4 de *Grafana*. La réponse JSON de la méthode *Lambda* a donc été simpifée en retirant la structure habituelle (statusCode et body) pour éviter de devoir traiter le JSON dans grafana.

6.9 API Gateway

Pour permettre aux utilisateurs d'ajouter des appareils et de récupérer les appareils, une *API Gateway* est utilisée. Elle met à disposition deux méthodes : une pour ajouter un appareil et une pour récupérer les appareils.

6.9.1 POST /deviceRegistrationLambda

La méthode POST passe l'ID de l'utilisateur et l'ID de l'appareil à ajouter dans le body de la requête sous la forme :

```
{
    "userId": "jean.michel@gmail.com",
    "deviceId": "2CF7F1F061900067"
}
```

Pour que le contenu du body soit accessible dans le *Lambda*, il faut ajouter un *mapping template* dans le *Integration request* de l'*API Gateway*.

```
{
    "body" : $input.json('$')
}
```

6.9.2 GET /getDevices

La méthode GET passe l'ID de l'utilisateur en paramètre de la requête sous la forme suivante :

```
GET .../getDevices?userID=jean.michel@gmail.com
```

Elle retourne ensuite les appareils de l'utilisateur sous la forme suivante :

```
[
    "2CF7F1F061900067",
    "2CF7F1F061900068"
]
```

6.10 Amplify

Pour permettre aux utilisateurs de créer un compte *Cognito*, de s'authentifier et d'ajouter des appareils, une page web est utilisée (https://env.d2alk2iwy901pr.amplifyapp.com). La page web a été développée en utilisant React. Amplify est utilisé pour héberger cette page web. Elle intègre *Cognito* pour l'authentification et l'*API Gateway* pour les appels *Lambda*. Cette page web existe car il n'était pas possible d'implémenter facilement ces fonctionnalités dans *Grafana*.

La page de login permet de créer un compte et de se logger.

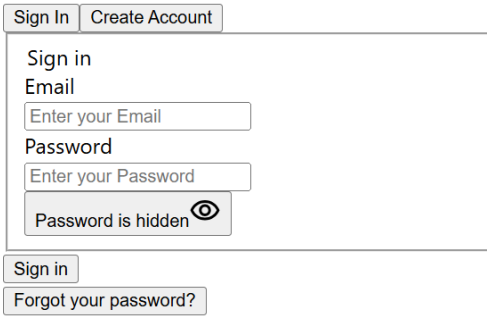

Fig. 7 — Login page

Une fois connecté, l'utilisateur peut ajouter un appareil en renseignant l'ID de l'appareil. Il peut aussi voir les devices déjà ajoutés à son compte.

Fig. 8 — Page d'enregistrement d'un appareil

Le design de la page web est simple. L'accent n'a pas été mis sur ce point durant ce projet. Si le produit devait être vendu, les fonctionnalités de cette page web seraient intégrées dans la page web ou l'application de visualisation.

6.11 Grafana

Pour la visualisation des données, *Amazon Managed Grafana* est utilisé. Il permet de créer des tableaux de bord pour afficher les données stockées dans des bases de données. Cette solution est simple à mettre en place et permet de visualiser les données en temps réel. L'ID du device à afficher est sélectionnable par l'utilisateur. Une requête est faite à *DynamoDB* pour récupérer les devices associés à l'utilisateur.

Fig. 9 — Visualisation des données

Le dashboard est composé de plusieurs graphiques. Chaque graphique affiche une donnée différente. Les données sont récupérées de *Timestream* et affichées en temps réel grâce à des requêtes *SQL*. L'exemple ci-dessous permet de récupérer les données pour le graphique de température. Dans cette requête, les données des devices sélectionnées sont récupérées. Les autres requêtes sont similaires et permettent de récupérer les données pour les autres mesures.

```
SELECT measure_value::bigint as Temperature, time
FROM "hivesData"."table"
WHERE measure_name = 'temperature' AND deviceID = '$devicesID'
ORDER BY time
```

Des alertes sont aussi configurées pour notifier les utilisateurs en cas de problème. Par exemple, si la température dépasse une certaine valeur, une alerte est envoyée à l'utilisateur. Les alertes sont configurées directement dans *Grafana* avec des conditions sur les données.

Cognito est utilisé pour gérer les utilisateurs. Malheureusement, *Amazon Managed Grafana* ne permet pas encore d'utiliser ce service d'authentification comme source d'authentification. Dans notre projet, nous avons donc utilisé un utilisateur par défaut pour la connexion à *Grafana* (compte AWS personnel). L'utilisateur renseigne ensuite son ID de device. Une requête est alors faite à la base de données *DynamoDB* pour récupérer les devices associés à l'utilisateur.

Grafana convenait à ce projet, car le but n'était pas de coder une page web ou une application mobile. Si le produit devait être vendu, une solution plus personnalisée pourrait être envisagée, comme une page web ou alors une application mobile. Il faudra alors utiliser *Cognito* comme source d'authentification à ce service. Ainsi, l'utilisateur n'aura plus besoin de renseigner son identifiant.

7 Scaling

La solution mise en place est scalable. Elle permet de gérer un grand nombre d'appareils et d'utilisateurs. Les services utilisés sont *serverless* et *managed*. Cela permet de ne pas avoir à gérer l'infrastructure et de laisser *AWS* gérer la montée en charge.

7.1 Appareils

Le software développé est générique. Il faut toutefois modifier les informations spécifiques aux appareils (EUI, AppKey, ...). Cette partie demande beaucoup de travail, mais il pourrait être fait en partie en créant un script qui va lire un fichier JSON de configuration et modifier le software.

Pour l'ajout de nouveaux appareils, il faut les ajouter à *TTN* ou à *Core IoT* pour les appareils Wi-Fi. Pour *TTN*, il est possible d'ajouter plusieurs appareils en même temps en utilisant un fichier JSON ou CSV (Adding devices in bulk). Il est aussi possible d'ajouter des appareils en utilisant un fichier et un provisioning template dans *IoT Core* pour les appareils Wi-Fi (Provisioning templates).

Les valeurs mesurées par des nouveaux appareils sont automatiquement ajoutées à *Timestream* comme les règles utilisées sont génériques.

7.2 Utilisateurs

La gestion des utilisateurs est gérée par *Cognito*. Il est possible de gérer un grand nombre d'utilisateurs. *Cognito* permet de gérer des millions d'utilisateurs. Il est donc possible de gérer un grand nombre d'utilisateurs sans problème.

La gestion des appareils est gérée par *DynamoDB*. Il est possible de gérer un grand nombre d'appareils sans problème. *DynamoDB* est scalable et permet de gérer jusqu'à des millions de requêtes par seconde.

Les utilisateurs peuvent ajouter un grand nombre d'appareils à leur compte. Il n'y a pas de limite au nombre d'appareils qu'un utilisateur peut ajouter à son compte. La visualisation permet d'afficher plusieurs appareils séparément ou simultanément. Un appareil peut aussi être ajouté à plusieurs comptes. Cela permet à plusieurs utilisateurs de visualiser les données d'un même appareil.

7.3 Coûts

Les coûts de la solution dépendent du nombre d'appareils et d'utilisateurs. Les coûts ont été avec le calculateur de coûts d'*AWS* (AWS Pricing Calculator).

Ils ont été calculés pour une fréquence d'envoi de 1 message chaque 10 minutes pour chaque appareil (4320 messages/mois), 10 visites par jours sur la visualisation et un nombre d'utilisateurs égal au nombre d'appareils. Les coûts sont détaillés dans le tableau suivant.

	100 devices	***5'000 devices***	***100'000 devices***
Coût total par mois en USD	1,80	58,44	1'653,14
Coût par appareil par mois en USD	0,018	0,012	0,016

Les coûts sont en principe dégressifs du nombre d'appareils et d'utilisateurs. Pour 100′000 appareils et utilisateurs, il par appareil est un peu plus élevé, car le service *Cognito* est payant seulement à partir de 10′000 utilisateurs. Pour 100′000, il faut payer un prix fixe par utilisateur (0,0055 USD).

8 Disaster recovery

Dans notre projet, nous stockons divers types de données : les données mesurées par les appareils, les ID des appareils et les identifiants et mots de passe des utilisateurs. Si les données mesurées sont perdues, cela impliquerait qu'il n'y a plus d'historique. Cependant, le système pourrait continuer à fonctionner. Si les ID des appareils des utilisateurs sont perdus, il serait toujours possible de faire réenregistrer par les utilisateurs leurs appareils. Si les identifiants et mots de passe des utilisateurs sont perdus, il serait possible pour les utilisateurs de se réinscrire. Pour les autres services, un disaster pourrait être plus problématique. Pour éviter ce type de problème, il est possible d'anticiper et de mettre en place des stratégies. *AWS* propose plusieurs solutions pour le disaster recovery.

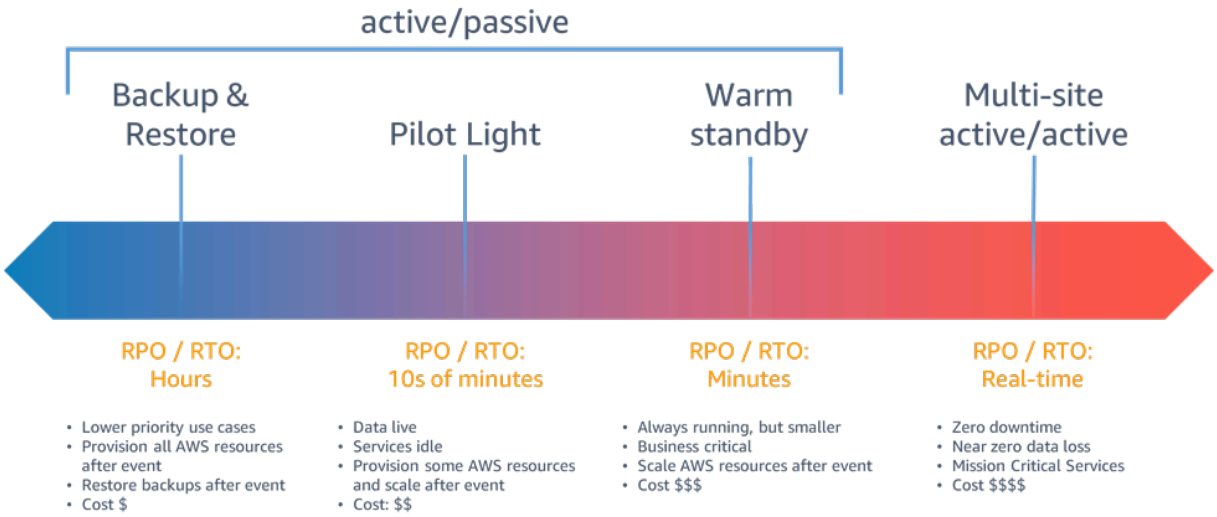

Fig. 10 — Statégies AWS de disaster recovery

La première est de faire des backups réguliers des données. Les backups sont stockés dans une autre *AWS Region* pour garantir la disponibilité des données en cas de problème. Cela protège les données contre des problèmes liés à une *Region* (*Backup and restore*). Cela ne protège que les données stockées dans *Timestream* et *DynamoDB*.

Une autre approche est l'approche pilote. Les données d'une région vers une autre et une copie de l'infrastructure de charge de travail principale sont provisionnées. Les ressources nécessaires à la prise en charge de la réplication et de la sauvegarde des données, telles que les bases de données, sont toujours actives. D'autres éléments, tels que les serveurs d'applications, sont chargés avec le code et les configurations d'application, mais sont « désactivés » et ne sont utilisés seulement en cas de problème (*Pilot light*).

Il existe deux autres solutions plus complexes (*Warm standby* et *Multi-site active-active*). Elles permettent de garantir une disponibilité maximale des données et de l'infrastructure. Mais ces solutions sont plus coûteuses, plus complexes à mettre en place et disproportionnées pour notre projet. Il est acceptable de perdre quelques données et d'avoir un temps d'arrêt de quelques minutes, voire d'heures, en cas de problème.

S'il y a un problème global avec *AWS*, cela est plus problématique, comme nous sommes dépendants d'*AWS* pour tous nos services. Il faudrait entreprendre les tâches suivantes :

- Mettre en place un broker MQTT pour les appareils.
- Mettre en place un serveur d'authentification pour les utilisateurs.
- Mettre en place de nouvelles bases de données pour les données mesurées et les ID des appareils.
- Mettre en place une API pour accéder à la base de données des appareils.
- Héberger une page web pour l'enregistrement des appareils.
- Héberger une page web ou un Grafana pour la visualisation des données.

Il faudrait en soi refaire le projet complet sans l'aide de *AWS*. Cela serait un travail conséquent qui prendrait énormément de temps et le système ne serait donc pas disponible pendant un certain temps.

Pour éviter d'arriver dans cette situation, il faut mettre en place toutes les solutions de disaster recovery proposées par *AWS* et faire des backups réguliers des données. Il faut aussi tester régulièrement ces solutions pour s'assurer qu'elles fonctionnent en cas de problème. Toutes ces solutions ont un coût et il faut donc évaluer si le coût de ces solutions est justifié par rapport aux risques encourus.

9 Conclusion

Ce projet de supervision des ruches a permis de développer un prototype fonctionnel qui répond aux besoins essentiels des apiculteurs. Grâce à l'intégration des capteurs de température, d'humidité, de mouvement et d'état de la porte, ainsi qu'à l'implémentation des communications LoRaWAN et Wi-Fi, le système permet de surveiller l'état des ruches en temps réel. L'utilisation des services AWS, tels que IoT Core, Timestream et Grafana, a offert une architecture scalable, résiliente et performante pour le traitement, le stockage et la visualisation des données.

9.1 Principaux résultats obtenus

Collecte et remontée de données : les données critiques des capteurs sont envoyées régulièrement et visualisées via un tableau de bord Grafana. Gestion utilisateur : l'intégration d'Amazon Cognito et de DynamoDB permet d'assurer une gestion efficace des utilisateurs et de leurs appareils. Contrôle de la ruche : bien que limité à la version Wi-Fi, le système permet le contrôle de la porte via des commandes Downlink. Le projet est conçu pour être déployé à grande échelle, avec des coûts raisonnables pour différentes tailles de déploiement (small, medium, large). La solution implémente également des principes de disaster recovery, garantissant la continuité des services et la protection des données en cas de panne.

9.2 Perspectives d'amélioration

Plusieurs améliorations peuvent être envisagées pour renforcer ce prototype :

Finalisation de l'intégration du GPS pour les versions LoRaWAN et Wi-Fi. Amélioration des interfaces utilisateurs avec une page web ou une application mobile plus ergonomique. Optimisation de l'architecture pour une meilleure intégration des commandes Downlink via LoRaWAN. Renforcement des stratégies de disaster recovery pour une résilience accrue. Ce projet démontre la pertinence des technologies IoT et cloud dans le domaine de l'apiculture. Il ouvre la voie à des solutions innovantes pour améliorer la gestion des ruches et répondre aux défis modernes des apiculteurs.