MSE | MASTER OF SCIENCE IN ENGINEERING

Hes·so
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences and Arts Western Switzerland

Master of Science HES-SO in Engineering
Av. de Provence 6
CH-1007 Lausanne

Master of Science HES-SO in Engineering

Orientation : Computer Science (CS)

IoT
Du besoin à l'industrialisation

Fait par

Jean Nanchen

Sous la direction de
Prof. Fabien, Vannel
Dans l'institut de l'hepia/inIT/CoRES

Sion, HES-SO//Master, 2023

Accepté par la HES-SO//Master (Suisse, Lausanne) sur proposition de

Prof. Fabien Vannel, conseiller du projet d'approfondissement

Sion, le 02.06.2023 2023

Prof. Fabien Vannel
Conseiller

1. Table des matières

1. Table des matières iv
2. Remerciements vii
3. Abréviations ix
4. Abstract x
5. Introduction 1
   5.1. Objectif de l'étude 1
6. Revue de la littérature et proposition de recherche 3
   6.1. État de l'art 3
   6.1.1. Les trackers GPS moto 3
   6.1.2. Les technologies de communications 4G pour l'IoT 3
   6.1.3. Les plateformes cloud pour l'IoT 4
7. Cahier des charges 4
   7.1. Objectifs 4
8. Hardware 5
   8.1. Architecture 5
9. Conception du software 7
   9.1. Système d'exploitation 7
   9.2. Librairies utilisées 7
   9.2.1. UBXLIB 7
   9.3. Architecture logicielle 9
   9.4. Gestion des capteurs 10
   9.4.1. Accéléromètre 10
   9.4.2. GPS 11
   9.5. Communication MQTT à travers LTE-M 12
   9.5.1. Connexion au broker MQTT 12
   9.5.2. Souscription 13
   9.5.3. Publication 13
10. Intégration Cloud 15
   10.1. Architecture Cloud 15
   10.2. Kit de développement AWS Cloud (CDK AWS) 16
   10.2.1. Custom ressources 16
   10.3. Provisioning 16
   10.3.1. Provisioning en masse (Bulk provisioning) 17
   10.3.2. Provisioning de flotte par demande (Fleet provisioning by request) 17
   10.3.3. Provisioning en temps réel (Just in time provisioning) 18
   10.3.4. Provisionnement d'un seul device (Single thing provisioning) 18

10.3.5. Choix du provisionnement 19
10.3.6. Création d'un Root CA 19
10.3.7. Création d'un certificat device 20
10.3.8. Provisionner le device IoT 20
10.3.9. Template pour le JITP 20
10.4. Mise à jour Over-The-Air (FOTA AWS Jobs) 21
10.4.1. Template 23
10.4.2. Création d'un Job avec la console de AWS 23
10.5. AWS Location Service 24
10.6. Plan de reprise après sinistre (Disaster recovery) 25
10.6.1. Régions AWS 25
10.6.2. Types de désastres 26
10.6.3. Objectif du point de récupération (RPO) 27
10.6.4. Objectif de délai de récupération (RTO) 27
10.6.5. Architecture choisie 27
10.7. Révocation d'un certificat 27
10.8. Coûts 27
10.8.1. Scénario 1 : utilisation intensive 27
10.8.2. Scénario 2 : utilisation fréquente 28
10.8.3. Scénario 3 : utilisation occasionnelle 29
10.8.4. Synthèse 30
11. Tests et validation 30
11.1. Test en condition réel 30
12. Conclusion 37
13. Travaux cités 39
14. Appendices **Erreur ! Signet non défini.**

2. Remerciements

Je tiens tout particulièrement à remercier le Prof. Fabien Vannel, conseiller du projet d'approfondissement, qui m'a suivi, conseillé et aidé durant ce projet d'approfondissement et répondu à mes innombrables questions.

Un énorme merci à Aurélien Héritier, qui m'a aidé à résoudre des problèmes avec mon module 4G.

3. Abréviations

AWS	Amazon Web Service
IoT	Internet Of Things
API	interface de programmation d'application
JITP	Just-In-Time Provisioning
GPS	Global Positioning System
FOTA	Fimrware Over The Air
LTE-M	Long-Term Evolution Machine Type Communication

4. Abstract

Dans cette étude, un démonstrateur a été réalisé permettant le suivi d'actifs visant à répondre aux diverses difficultés et problématiques lors d'un déploiement à grande échelle d'un produit IoT. Le démonstrateur établit une communication MQTT à travers une connexion 4G avec un service cloud et envoie sa position à celui-ci. Une partie de l'étude se penche sur les différents types de provisionnement fournis par le service Cloud AWS, tels que l'approvisionnement juste à temps ou encore le provisionnement de flotte. Elle aborde aussi différents outils pour piloter une flotte d'appareils (FOTA avec AWS IoT Jobs). Une partie de cette étude parle de récupération après désastre, permettant de rediriger la flotte IoT sur une autre région du service Cloud en cas de sinistre, cette partie a été partiellement implémentée et étudiée.

Mots clefs: IoT, provisioning, JITP, Disaster Recovery, UBXLIB, U-BLOX, LTE-M, 4G, NB-IoT, GPS, Tracker, AWS.

5. Introduction

L'Internet des objets (IoT) s'est considérablement installé dans divers secteurs tel que l'industrie, mais commence aussi à émerger dans tous les domaines. Cette émergence répond à un besoin crucial de tout pouvoir contrôler, récolter et traiter, cela permet de prendre des décisions et d'agir sur des données mesurées en temps réel.

Dans l'industrie, l'IoT permet la surveillance des machines de production en détectant d'éventuelle défaillance, effectuer des contrôles de qualités sur les produits fabriqués ou encore d'effectuer un suivi sur les actifs de l'entreprise. Ces cas d'applications permettent aux entreprises de réduire leurs coûts tout en gagnant du temps et en automatisant des processus fastidieux.

Dans les villes intelligentes, l'IoT permet d'améliorer la qualité de vie de ses citoyens par exemple avec une gestion des déchets intelligente, une gestion intelligente du trafic, une surveillance active de la qualité de l'air, etc.

5.1. Objectif de l'étude

Cette étude vise à examiner les stratégies à adopter et de l'architecture à mettre en place pour le déploiement d'une flotte d'appareil IoT, grâce à la réalisation d'un prototype permettant le suivi en temps réel d'une moto, avec une communication constante vers un service cloud.

Le provisionnement, qui représente la première relation de confiance entre l'appareil et le service cloud, est abordé dans cette étude. Il s'agit de comprendre comment l'appareil doit être enregistré et de quelle manière l'appareil va pouvoir s'intégrer à son réseau.

La gestion des appareils depuis le cloud a aussi est également traitée dans cette étude permettant d'interagir et d'effectuer des actions critiques à distance sur une flotte d'appareils IoT, comme par exemple effectuer une mise à jour à distance.

Un autre volet de cette étude parle de l'architecture cloud pour une flotte d'appareils tels que des tracker GPS moto et un plan de récupération après désastre a été élaboré permettant, lors d'un sinistre, la continuation de l'activité de la flotte d'appareils.

Enfin, une étude de coût de la partie cloud a été réalisée pour observer le coût global d'un déploiement d'une flotte.

6. Revue de la littérature et proposition de recherche

6.1. État de l'art

Ce chapitre s'intéresse aux différentes technologies de communications 4G disponible pour les appareils IoT, ainsi qu'un comparatif de différents services Cloud disponibles pour l'IoT. Une analyse de fonctionnement et financière d'un tracker GPS moto, similaire à celui développé lors de cette étude, permet d'étudier les fonctionnalités et les prix fournis par la concurrence.

6.1.1. Les trackers GPS moto

Les trackers GPS pour motos sont des dispositifs IoT permettant de les suivre en temps réels. Ces appareils sont généralement connectés à un service cloud et fournissent en temps réel sur la position du véhicule ainsi que son état. C'est une solution efficace contre le vol.

Le tracker GPS moto le plus connu est fabriqué par une entreprise française, Georide. Celui-ci se présente sous la forme d'un boitier que l'on peut placer sous la selle de sa moto et permet d'avertir son utilisateur en temps réel d'un mouvement suspect du véhicule et peut consulter sa position.

Ils contiennent différents modules. Un module GPS permet de déterminer l'emplacement du véhicule. Un module 4G permettant d'échanger des informations avec le cloud. Un gyroscope et un accéléromètre permettent de déterminer des mouvements suspects de la moto. Une batterie tampon de secours pour éviter tout cas de coupure intentionnelle de la batterie du véhicule. Une alarme en option peut être connectée pour permettre de retrouver le véhicule en sous-terrain.

Le traceur proposé par l'entreprise Géoride est affiché au prix de 369 euros avec une année d'abonnement gratuite, puis 49 euros par année [1]. L'entreprise fournit aussi de nombreux services comme une assistance après vol et une assistance en cas d'accident.

6.1.2. Les technologies de communications 4G pour l'IoT

LTE-M et NB-IoT sont des « Low Power Wide Area Network (LPWAN)» développé pour l'internet des objets en respectant un cahier des charges strict pour obtenir une faible consommation, une forte pénétration et des coûts réduits.

NB-IoT

NB-IoT est une technologie de communication radio dédiée à l'IoT. Cette technologie est basée sur le GSM et utilise le spectre existant des réseaux cellulaires. Cette technologie offre de nombreux avantages tels que :

- Une très faible consommation
- Une large portée : couverture dans des environnements difficiles
- Faible débit : dans l'ordre du kbit/s

Par contre, le NB-IoT n'est pas efficace si l'appareil est en mouvement. En effet, si l'appareil s'éloigne de l'antenne, il va augmenter la puissance d'émission pour rester connecter. Quand la connexion est perdue, l'appareil se connecte avec une autre antenne et doit recommencer son enregistrement depuis le début de ce qui cause des pertes et augmente la consommation d'énergie [2].

LTE-M

LTE-M est un type de LPWAN développé expressément pour l'IoT. Il a été créé par 3GPP qui est, selon Wikipédia, « une coopération entre organismes de normalisation en télécommunications [3]». Les grands avantages de LTE-M par rapport à NB-IoT sont :

- Grand débit : Mbit/s
- **Mobilité : LTE-M peut switcher entre les antennes disponibles et permet la connectivité d'un appareil en mouvement, ce qui n'est pas le cas du NB-IoT.**

- Faible consommation énergétique
- Voice over network

Le prototype qui doit être réalisé sera un tracker GPS et sera en mouvement. Le LTE-M sera alors privilégié.

6.1.3. Les plateformes cloud pour l'IoT

Les plateformes cloud offrent des services essentiels pour faciliter le déploiement et la gestion d'appareils IoT à grande échelle. Ces plateformes permettent d'utiliser des services à la demande sans avoir à construire une infrastructure dédiée. Elles offrent diverses fonctionnalités de sécurité, stockage de données, gestions de données, analyses de données, gestions des appareils, traitement de données, etc. Il existe plusieurs fournisseurs cloud qui offrent une solution pour des appareils IoT.

- AWS IoT Core (Amazon Web Services): AWS IoT Core est une plateforme permettant aux appareils IoT de pouvoir s'y connecter (HTTP, MQTT, WebSockets), échanger, et gérer des données. Cette plateforme fournit diverses fonctions telles que des fonctions de provisionning, des routages en utilisant des règles, connexions à d'autres services AWS.
- Azure IoT Hub (Microsoft Azure): Azure IoT Hub est aussi une plateforme permettant aux appareils IoT de pouvoir s'y connecter en HTTP, MQTT, AMQP Over WebSockets. Tout comme AWS il offre des fonctions de sécurité, de gestions des données et d'échanges de données.
- Google Cloud IoT Core (Google): Google Cloud IoT Core prend en charge des connexions HTTP et MQTT, il offre des fonctionnalités de traitement de données, de sécurité et permet l'intégration d'autres services Google Cloud.

Chaque plateforme offre des fonctionnalités essentielles à la gestion d'appareils IoT, mais diffère en termes de coûts, de flexibilité et d'évolutivité.

Figure 1 - Logo de Amazon Web Service / Google Cloud Platform / Azure

7. Cahier des charges

7.1. Objectifs

Les différents objectifs à démontrer grâce à ce démonstrateur sont :

- Connectivité : établir une connexion 4G LTE-M à l'aide d'un module 4G
- Positionnement : le tracker doit pouvoir récupérer sa position à l'aide d'un module GPS
- État du véhicule : Le tracker doit être capable de faire du suivi actif et passif en détectant l'état du véhicule (déplacement, immobile)
- Provisionnement : le provisionnement utilisé doit permettre une scalabilité de la flotte d'appareil
- Disaster Recovery : en cas de désastre, la flotte doit pouvoir avoir accès à une autre région du cloud
- Actions à distance : une solution de mise à jour à distance doit être étudiée et validée

8. Hardware

Le kit de développement choisi pour le développement de ce démonstrateur est le XPLR-IOT1 développé par la société U-BLOX. Il contient divers modules permettant de faire de la 4G LTE/M, 4G NB-IOT. Il est capable de déterminer sa position grâce à son module GPS. Ce kit de développement a été préféré au nRF9160 DK pour sa portabilité grâce à sa batterie intégrée et l'intégration native de différents capteurs et de boutons en façade facilitant l'interface utilisateur. De plus, il est entouré d'une coque protectrice rouge lui permettant une bonne robustesse. Il est livré de base avec un firmware contenant un bootloader et une partie applicative de démonstration. L'architecture complète de cet appareil est expliquée dans le chapitre 8.1.

Figure 2 - XPLR-IOT-1[1]

8.1. Architecture

Le XPLR-IOT-1 dispose d'une architecture complète pour réaliser différents « proof-of-concept » pour l'IoT. L'appareil peut communiquer en Wi-Fi, Bluetooth et LTE-M/NB-IoT. Il dispose d'un module GPS de la marque U-Blox dédiée, et dispose d'une batterie intégrée. Il est équipé de différents capteurs (température, humidité, pression, lumière ambiante, magnétomètre, gyroscope, accéléromètre et niveau de batterie).

Le composant mère de l'appareil est le NORA-B106. C'est un module de la série NORA-B1 fabriqué par U-BLOX. Il est doté d'un nrf5340 de chez Nordic Semiconductor qui est un ARM Cortex M33 dual core. Ce module est destiné pour la communication Bluetooth 5.2 ainsi que NFC.

D'autres modules sont connectés au NORA-B106 à l'aide de différents ports série, comme le SARA-R510S qui est un module développé par U-BLOX de la gamme SARA-R5 capable de communiquer en LTE-M/NB-IoT. Ce module contient une « secure element » pour pouvoir y stocker des éléments critiques comme des certificats. Un module GPS le MAX-M10S fabriqué lui aussi par U-BLOX est aussi relié sur un port série. Un module permettant la communication Wi-Fi, le NINA-W156 de chez U-BLOX, est aussi relié au NORA-B106.

Les différents capteurs (température, humidité, pression, lumière ambiante, magnétomètre, gyroscope, accéléromètre) sont reliés au NORA-B106 en I2C.

[1] Illustration de digikey : https://www.digikey.com/en/products/detail/u-blox/XPLR-IOT-1/16569699

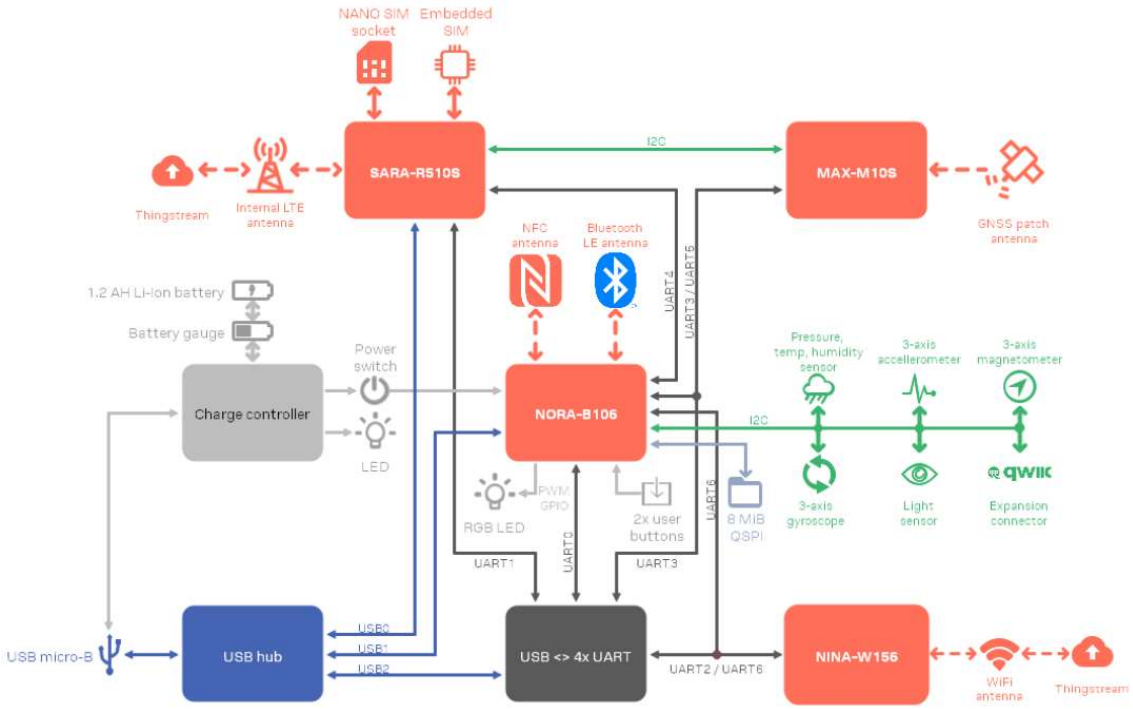

Figure 3 - XPLR-IOT-1 schéma bloc[2]

[2] Illustration de XPLR-IOT-1 User Guide : https://content.u-blox.com/sites/default/files/documents/XPLR-IOT-1_UserGuide_UBX-21035674.pdf

9. Conception du software

9.1. Système d'exploitation

La conception du software du tracker a été basé sur Zephyr OS qui est un système d'exploitation temps réel (RTOS) open source. Selon Wikipédia, ce RTOS a été spécialement « conçu pour les appareils aux ressources limitées, supportant plusieurs architectures [4]».

Il est utilisé dans les objets à fortes contraintes comme les objets IoT. Il a été conçu pour être aussi léger que possible grâce à sa grande modularité. Il est en effet possible d'inclure uniquement les composants du système d'exploitation nécessaire à l'applicatif, ce qui permet de réduire les ressources. Zephyr OS offre la possibilité de pouvoir gérer plusieurs tâches en même temps grâce à son support complet pour la programmation multitâche.

Figure 4 - Zephyr OS [4]

9.2. Librairies utilisées

9.2.1. UBXLIB

UBXLIB, une librairie développée par U-BLOX, a été utilisée pour faciliter la communication entre les différents modules U-BLOX et l'applicatif. Cette bibliothèque de haut niveau a été développée en C, permets donc depuis un hôte de communiquer et de configurer le périphérique attaché via commande AT. Elle propose donc une grande quantité d'API permettant un accès facile aux diverses fonctionnalités des modules U-BLOX :

- API basiques
  - Device : pour interagir avec un module
  - Network : pour se connecter à un réseau (GNSS, Cell, Wi-Fi, Bluetooth)
  - Sock : ouvrir des sockets sur un réseau
  - Security : gérer des éléments de sécurité (certificats)
  - MQTT : se connecter à un broker MQTT
  - HTTP : faire des requêtes HTTP
  - Localisation : obtenir une localisation
- API cellulaire
- API Bluetooth Low Energy
- API Wi-Fi
- API GNSS
- API Common Short Range
- API Client AT
- API UBX Protocol
- API Port

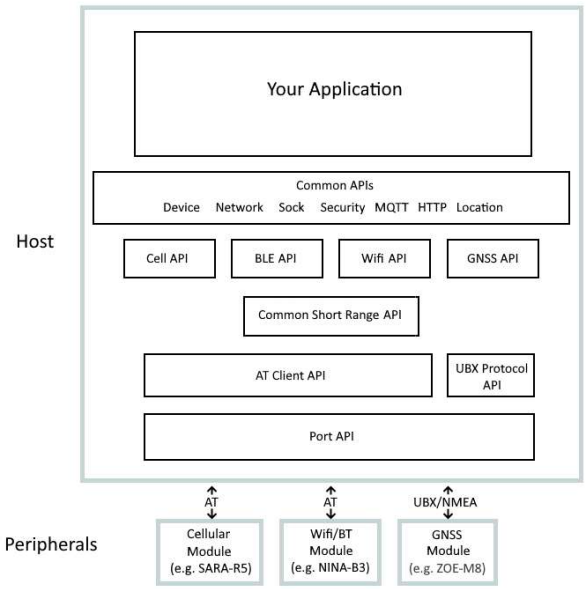

Figure 5 - UBXLIB API [5]

Compatibilité de UBXLIB avec d'autres OS

UBXLIB n'est pas compatible avec tous les systèmes d'exploitation et tous les processeurs hôtes comme illustré dans le Tableau 1. Pour le NORA-B106 intégrant un nRF53, la compatibilité est assurée avec Zephyr et non avec d'autres systèmes d'exploitation temps réel tels que FreeRTOS.

ubxlib host	NINA-W10	NINA-B40 series NINA-B30 series NINA-B1 series ANNA-B1 series	NORA-B1 series	C030 board	PC	PC
MCU	Espressif ESP32	Nordic nRF52	Nordic nRF53	ST-Micro STM32F4	x86 (win32)	x86 (32-bit Linux)
Toolchain	ESP-IDF Arduino-ESP32	GCC nRF Connect	nRF Connect	Cube	MSVC	Zephyr
RTOS/SDK	FreeRTOS	FreeRTOS Zephyr	Zephyr	FreeRTOS	Windows	Zephyr
APIs fournit par l'host avec les périphériques attachés	wifi ble device network sock	ble device network	ble device network	cell device network sock location*** TLS security	N/A	N/A

Tableau 1 - Hôtes compatibles avec UBXLIB [5]

9.3. Architecture logicielle

Le firmware embarqué dans le XPLR-IOT1 tourne sur 3 threads distincts, visible à la Figure 6. Le premier thread appelé « mqttThread » est responsable de la connexion au broker MQTT (AWS IoT Core) via le LTEM. Ce thread permet également la gestion d'une pseudo mise à jour du « Firmware Over The Air » (FOTA) à l'aide de Job AWS, voir chapitre 10.4.

Parallèlement à cela, un autre thread nommé « gpsThread » est dédié spécifiquement à la mise en place de la liaison GPS lors de l'initialisation de l'appareil.

L'ouverture de ces connexions par les threads sera exploitée par la fonction main du programme qui s'occupera de récupérer la localisation GPS et de l'envoyer au broker MQTT.

Cette architecture multi-threadée permet la parallélisation des tâches pour un démarrage à froid rapide.

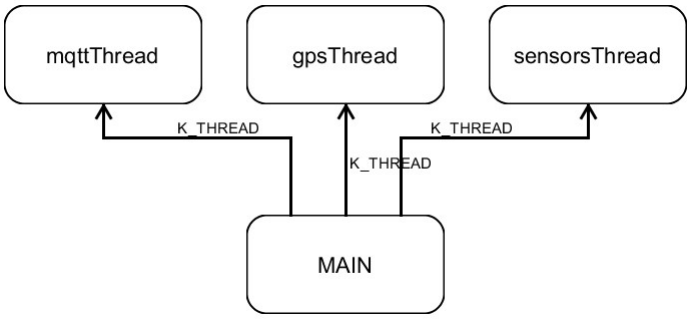

Figure 6 – Multithreading

La Figure 7 représente le diagramme de classe de l'architecture logicielle. On peut y observer les dépendances entre les classes. La classe « devicehandler.h » contenant un struct se nommant « trackerHandle_t » contient différents objets et pointeurs permettant de conserver le handle d'une connexion MQTT ou du module GPS. Il est nécessaire de créer un struct par connexion (un pour la connexion MQTT et un pour le module GPS). Cela permet d'effectuer du multithreading sur deux modules en même temps. Une classe « gps.h » a été implémentée avec diverses fonctions permettant par exemple d'activer le GPS et d'obtenir un signal GPS, de demander sa localisation actuelle et une autre fonction permet de mettre en forme le point GPS reçu en format char*. Une classe « mqtt.h » a été créée et permet de simplifier la connexion au broker MQTT et l'envoi à celui-ci à l'aide de fonctions basiques comme « connectToMqttBroker », « subscribeToTopic » ou encore « publishToMqttBroker ». Ces fonctions utilisent la structure « trackerHandle_t » qui contient la session en cours de la connexion MQTT. La fonction de subscribe au topic MQTT est appelée avec une structure nommée « mqttCbStruct ». Cette structure contient aussi un handle de la connexion MQTT et un pointeur sur le Topic à laquelle le device doit se souscrire.

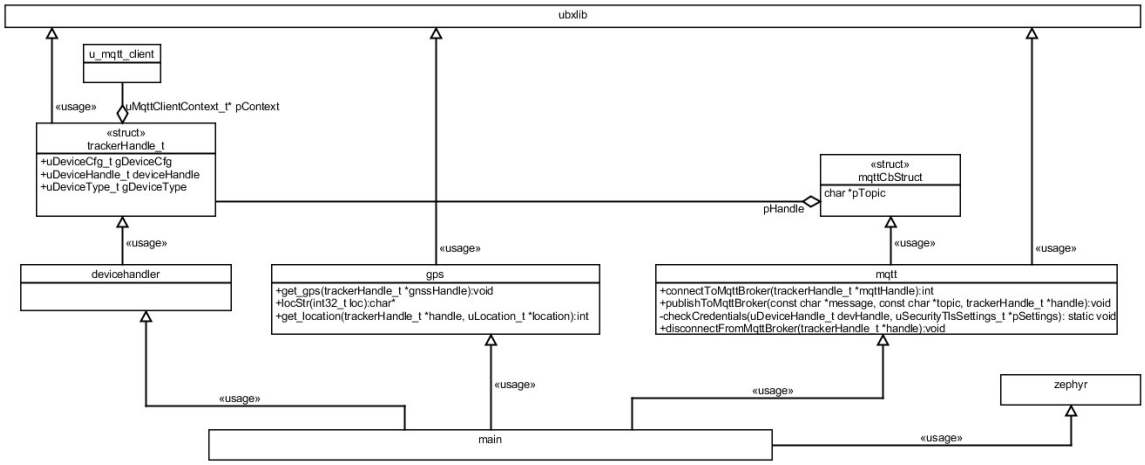

Figure 7 - Diagramme de classe

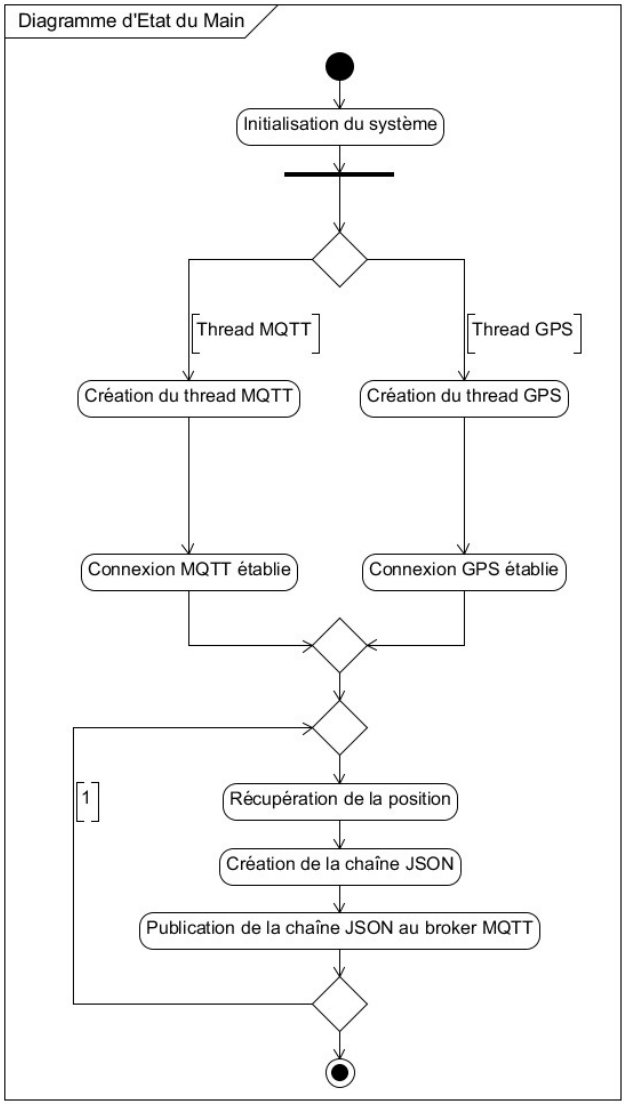

Figure 8 - Diagramme d'état du main.c

La Figure 8 illustre le diagramme d'état de la fonction main. On y observe le processus et les étapes que suit la fonction main. Au début, une routine d'initialisation est lancée, ensuite deux threads sont créés et établissent une connexion GPS et une connexion au broker MQTT. Ensuite, le programme tombe dans une boucle infinie et « trace » la position de l'appareil toutes les 30 secondes.

9.4. Gestion des capteurs

Dans ce chapitre est expliquée la façon dont la partie applicatif s'interface à travers de la librairie U-BLOX, UBXLIB, aux différents modules sur le kit de développement XPLR-IOT1.

9.4.1. Accéléromètre

L'état de la moto a été défini en deux états :

- En mouvement
- Immobile

La lecture de l'accéléromètre sur le bus I2C du kit de développement XPLR-IOT1 permet de déterminer son état. Zephyr permet de connecter ce capteur à une interruption à l'aide de la classe « sensor.h » et de la fonction « sensor_attr_set() ».

```
    gpLis2dhDev2 = DEVICE_DT_GET_ANY(st_lis2dh);
    struct sensor_value val_threshold;
    val_threshold.val1 = 9;
    val_threshold.val2 = 1000000;

    retval = sensor_attr_set(gpLis2dhDev2, SENSOR_CHAN_ACCEL_XYZ, SEN-
SOR_ATTR_SLOPE_TH, &val_threshold);
    retval = sensor_trigger_set(gpLis2dhDev2, &trig, trigger_handler);
```

Figure 9 – Callback de mouvement sur trigger_handler avec le capteur LIS2DH de ST

Un tableau avec l'historique des mouvements lors des 5 dernières minutes permet de déterminer si le véhicule s'est immobilisé ou s'il est toujours en mouvement.

9.4.2. GPS

Le module GPS permet de récupérer la position du véhicule. L'applicatif doit alors ouvrir une interface pour communiquer avec le module GPS externe dans le kit de développement XPLR-IOT1.

Ouverture de l'interface avec le module GPS

Le diagramme de séquence visible à la Figure 10 illustre le processus de récupération de la position GPS à l'aide du module GPS embarqué dans le XPLR-IOT1. La fonction reçoit en entrée la structure « tackerHandle_t » qui se comporte comme un gestionnaire de connexion. La fonction « uDeviceGetDefaults() » va être appelée pour récupérer les paramètres par défaut du dispositif GPS. La structure va ensuite ouvrir le module spécifique, en l'occurrence ici le module GPS à l'aide de la fonction « uDeviceOpen ». Un fragment « ALT » divise le flux du diagramme en deux en fonction du code d'erreur retourné. S'il n'y a pas d'erreur, la fonction « uNetworkInterfaceUp » est appelée pour établir la connexion avec le module GPS. Une fois la connexion établie avec le module, la fonction « uLocationGet » est appelée dans une boucle jusqu'à 4 fois pour obtenir les données de géolocalisation. Si le GPS ne parvient pas à fournir une localisation GPS correcte, l'interface avec le GPS est fermée avec « uNetworkInterfaceDown() » et « uDeviceClose() » et la LED témoin est mise en rouge. Dans le cas contraire, si le GPS parvient à obtenir une localisation GPS correcte, la LED devient verte, on retourne le « handle » du GPS.

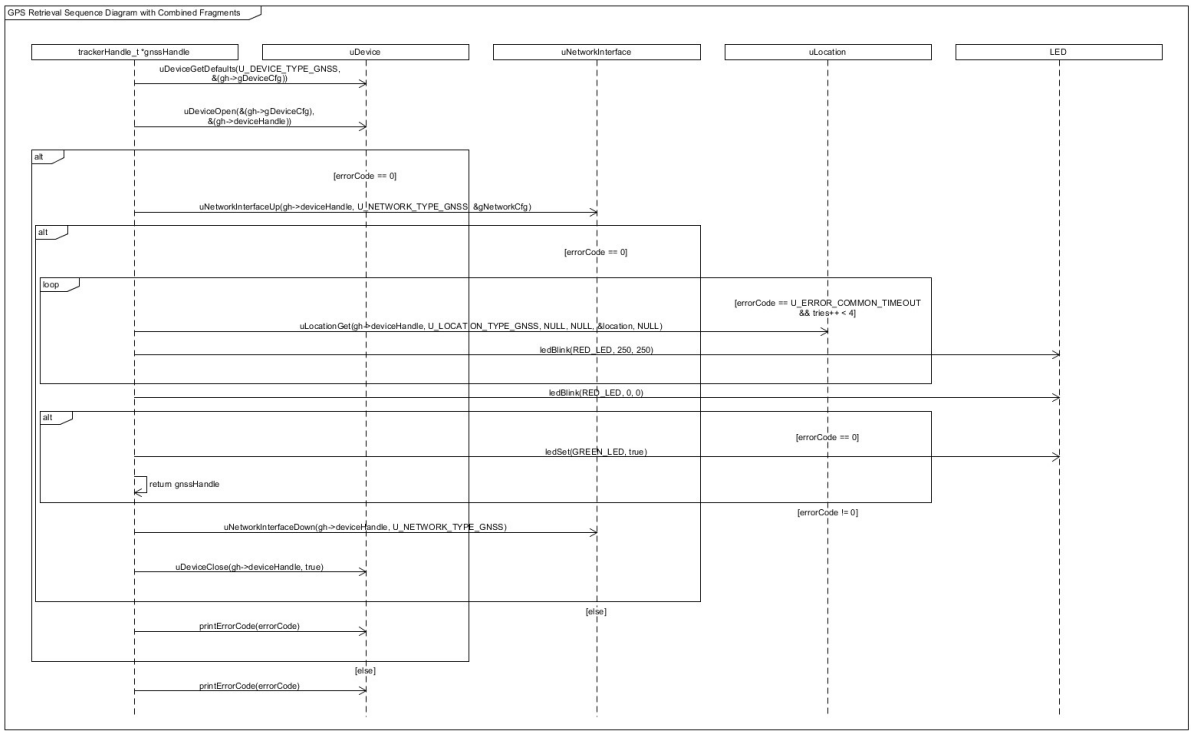

Figure 10 - Diagramme de séquence de la connexion GPS

9.5. Communication MQTT à travers LTE-M

La communication avec le cloud se fait par MQTT. Le device écrit sur des topics (par exemple sa position actuelle) et s'abonne à des topics pour être avertis d'une éventuelle mise à jour. Lors d'envoi fréquent de données, il n'est pas recommandé d'ouvrir et de fermer a chaque fois la connexion au broker, car chaque ouverture de session est couteuse en données transmises. Pour ce faire, il est recommandé d'ouvrir la session et le module SARA R5 s'occupe d'envoyer des heartbeat pour laisser la connexion ouverte. Les différents sous-chapitre de ce chapitre décrivent la procédure pour se connecter au broker MQTT de AWS, s'abonner à un topic, et publier sur un topic avec la librairie UBX-LIB.

9.5.1. Connexion au broker MQTT

Le diagramme de séquence, visible à la Figure 11, illustre le processus de mise en route du module SARA R5 pour effectuer une connexion au broker MQTT (connectToMqttBroker). Grâce à la structure « trackerHandle_t » (gestionnaire de connexion) passée en paramètre de la fonction la fonction appelle au début une fonction d'initialisation et ouvre un device spécifique avec la fonction « uDeviceOpen », en l'occurrence ici le module SARA R5. À partir de cet instant, un fragment « ALT » divise en deux le flux du diagramme en fonction de l'erreur retourné. Si le device s'est ouvert, le code continue en appelant la fonction « checkCredential » qui va charger les certificats dans le module. Ensuite la fonction « uNetworkInterfaceUp() » permet de mettre en ligne l'interface réseau pour un type de réseau spécifique, dans cette application il s'agit d'une connexion cellulaire. Dans cette lancée, des fonctions de la classe uMqttClient vont être appelées pour créer un client de connexion et ensuite ouvrir une connexion vers le broker MQTT. En cas d'erreur, la fonction retourne la valeur entière négative -1. En cas de réussite, la fonction retourne 0.

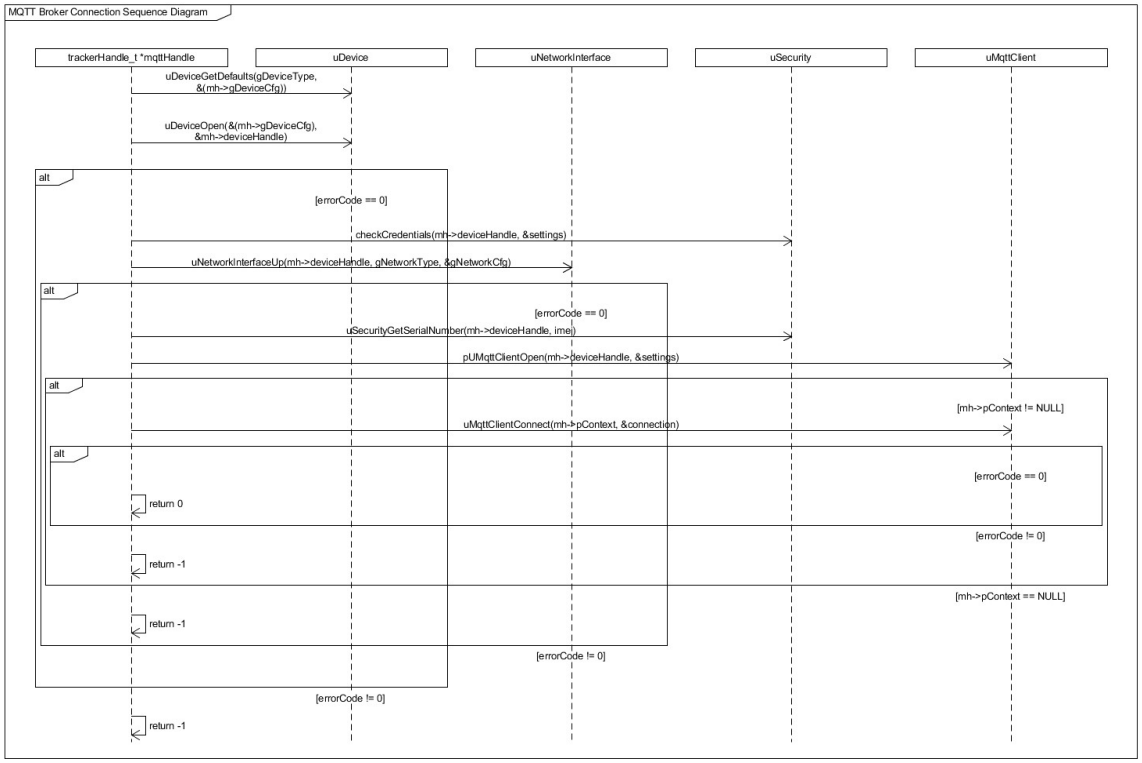

Figure 11 - Diagramme de séquence d'une ouverture de connexion MQTT à un broker

9.5.2. Souscription

Le diagramme de séquence visible à la Figure 12, démontre le processus d'abonnement à un topic MQTT à l'aide du module cellulaire SARA R5. La fonction prend en entrée la structure « mqttCbStruct » qui contient le handle de la connexion MQTT, le nom du topic ainsi que le callback qui sera appelé lors d'un message sur le topic souscrit. Grâce à ce « handle », la fonction « subscribeToTopic() » appelle des fonctions de la classe « uMqttClient », comme « uMqttClientOpenResetLastError » pour effectuer un reset sur les derniers messages d'erreurs. Un abonnement sur le topic est ensuite fait grâce à la fonction « uMqttClientSubscribe() ». Après la souscription, la fonction « uMqttClientSetMessageCallback() » est appelée pour définir une fonction de callback qui sera déclenchée lors de la réception de messages sur le topic auquel le device s'est abonné.

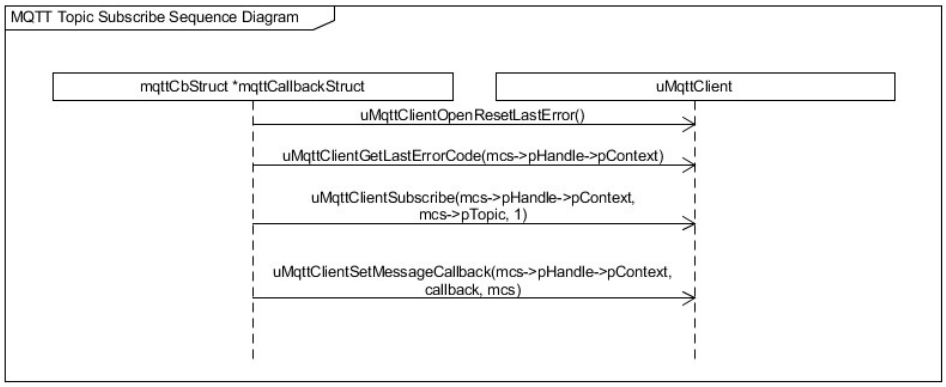

Figure 12 - Diagramme de séquence pour la souscription à un topic MQTT

9.5.3. Publication

Le diagramme de séquence à la Figure 13 illustre le processus d'une publication d'un message sur un topic défini grâce au module SARA R5. La fonction « puiblishToMqttBroker » prend en entrée un message, un topic et un

handle d'une connexion MQTT. Avec le « handle » de connexion, la fonction va appeler diverses fonctions de la librairie « uMqttClient » de U-BLOX. La première fonction appelée est « uMqttClientOpenResetLastError() » qui permet de mettre à zéro les erreurs reçues précédemment. Ensuite la fonction appelle « uMqttClientPublish() » qui prend en paramètre le contexte (fourni par le handle), le topic, le message, la longueur du message, le QOS ainsi qu'un paramètre de retain.

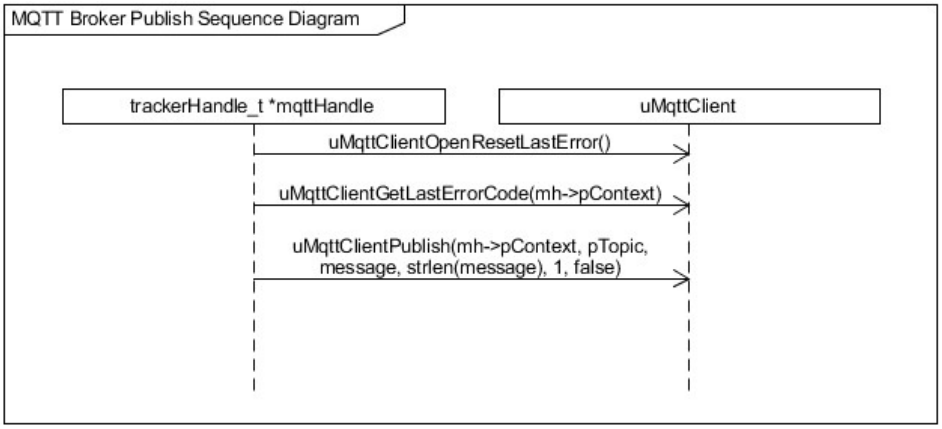

Figure 13 - Diagramme de séquence pour la publication sur un topic

10. Intégration Cloud

10.1. Architecture Cloud

L'architecture cloud pour une flotte de tracker est présentée à la Figure 14. Elle est structurée sur deux régions, une région primaire et une région secondaire. L'architecture de la région primaire est à moitié répliquée sur la seconde région pour permettre la création et la gestion facile d'un plan de reprise après sinistre.

La région primaire contient deux piles distinctes créées par CloudFormation (voir le chapitre 10.2). Une pile nommée « SAAS_STACK » contient un « template » de provisionnement qui permet aux appareils de pouvoir s'enregistrer juste à temps à AWS IoT Core. Diverses règles sont créées pour router les messages reçus / état de provisionnement des appareils vers une base de données DynamoDB et vers une Lambda « tackFunctiuon1-dev » de le deuxième pile « Amplify Stack ». Les données transférées à la base de données DynamoDB sont majoritairement des informations de provisionnement juste à temps comme le nom du device enregistré, son certificat, etc. Cette base de données est répliquée sur la seconde région AWS puis renvoie ses évènements à une lambda qui s'occupe de provisionner les appareils sur la seconde région.

Comme expliqué précédemment, les messages contenant la géolocalisation des trackers sont redirigés sur la Lambda « TrackFunction1-dev » de la seconde stack de la région primaire, nommée « Amplify Stack ». La stack précédente contient toute la partie applicative, comme une carte de AWS Location Service et un unique tracker AWS Location Service. La Lambda appelée pas une rule de AWS IoT Core va pusher les données de géolocalisation sur le tracker de AWS Location Service avec l'ID correspondant. L'utilisateur va pouvoir à l'aide d'un serveur Web afficher la carte et l'historique des suivis du tracker.

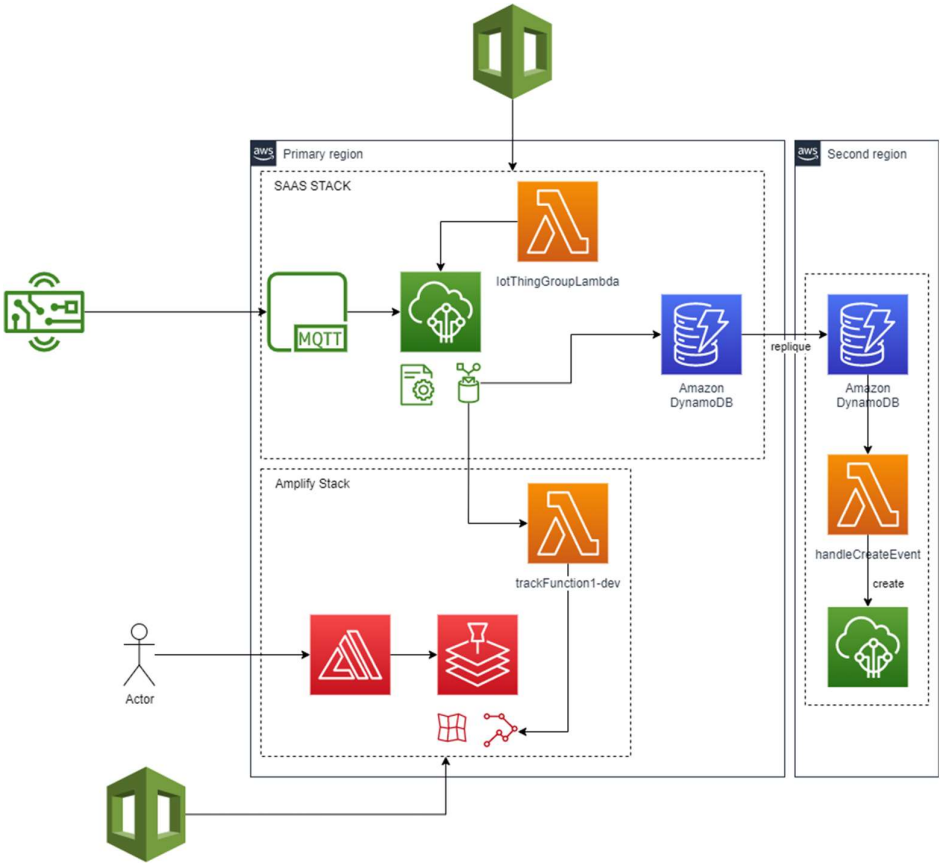

Figure 14 - Architecture cloud AWS du tracker

10.2. Kit de développement AWS Cloud (CDK AWS)

AWS CDK permet de créer des applications fiables, scalable et à faible coût dans le cloud. L'application est décrite à l'aide de langage de programmation familier tel que Python, Java, C#, TypeScript, JavaScript, Go et .NET. Ces applications peuvent être ensuite déployées à l'aide de AWS CloudFormation qui permet d'effectuer des déploiements de manière prévisible et répétée avec retour en arrière en cas d'erreur [6].

L'architecture a donc été déployée sur deux piles distinctes, une pile nommée « SaasStack » sur eu-central-1 et une seconde « SaasStackSecondary » sur eu-west-1. Le code des deux piles se trouve sur le repository à l'annexe I.

10.2.1. Custom ressources

La création de certains éléments n'est pas encore disponible avec la dernière version du CDK de AWS. Pour contourner cela, l'utilisation de « Custom ressources » fût nécessaire. Une « custom ressource » permet de prendre en paramètre une Lambda (voir le code visible à la Figure 15) qui sera exécutée lors de sa création et de sa suppression sous forme d'évènement. Cette Lambda doit informer AWS CloudFormation de l'état de la création/suppression de la ressource (« SUCCESS » ou « FAILED »). C'est la Lambda qui, grâce à boto3 qui est plus complet, qui s'occupe de gérer la création de l'élément.

```
# JITP
# Lambda iot_jitp_template
iot_jitp_template_lambda = lambda_.Function(
    self,
    "IotJitpTemplateLambda",
    code=lambda_.Code.from_asset("saas/iot_jitp_template"),
    handler="index.lambda_handler",
    runtime=lambda_.Runtime.PYTHON_3_8,

    # Add the policy to the role
    role=lambda_role,
)
iot_jitp_template_lambda.apply_removal_policy(cdk.RemovalPolicy.DESTROY)

# Create Provisioning template WITH CUSTOM RESOURCE
iot_jitp_template = cfn.CfnCustomResource(
    self,
    "IotJitpTemplate",
    service_token=iot_jitp_template_lambda.function_arn,
)
#iot_jitp_template.apply_removal_policy(cdk.RemovalPolicy.DESTROY)
iot_jitp_template.add_override("Properties.TemplateName", "SaasIotJitpTemplate")
iot_jitp_template.add_override("Properties.ProvisioningRoleArn", iot_provioning_role.role_arn)
jitp_body = open("saas/body.json", "r").read()
iot_jitp_template.add_override("Properties.TemplateBody", jitp_body)
# iot_jitp_template.add_override("Properties.CaCertificate", ca_certificate)
```

Figure 15 - Création d'une custom ressource qui lance une Lambda

10.3. Provisioning

Le provisionnement du device est la première relation de confiance entre l'appareil et AWS. Pendant ce processus, AWS enregistre l'appareil qui définit la lien entre AWS et le device. Les fichiers créés pour l'appareil sont :

- Un certificat X.509

- Une clef privée

- Un certificat racine Amazon Trust Service Root Certificate Authority (Root CA) ou un certificat racine signé par AWS

Quand un « thing » IoT est créé, un certificat X.509 doit lui être attribué. Ce certificat est utilisé pour authentifier le device. Le certificat est signé par le certificat racine AWS. Le certificat racine est signé par le certificat racine Amazon Trust Service Root Certificate Authority (Root CA). Le Root CA est un certificat racine signé par AWS. Chaque device à un certificat X.509 unique. Chaque certificat unique doit être rattaché à une unique IoT Policy (il n'est pas nécessaire d'avoir une policy par device).

Il existe de nombreuses possibilités pour provisionner les devices IoT en fonction des besoins spécifiques de chaque application. Dans ce chapitre toutes les possibilités de provisionnement ont été testées et comparées.

10.3.1. Provisioning en masse (Bulk provisioning)

La bulk registration permet de faire du provisionnement en masse d'une liste d'appareil. Cette méthode consiste à charger un fichier JSON contenant les informations des appareils à provisionner. Ce fichier est ensuite envoyé sur une base de données S3. AWS IoT Core va ensuite lire le fichier et provisionner les appareils. Cette méthode permet de provisionner une grande quantité d'appareils en une seule fois [7].

10.3.2. Provisioning de flotte par demande (Fleet provisioning by request)

Avec cette méthode, l'appareil possède un certificat de demande de connexion. Ce certificat possède une police ultra restrictive et ne peut publier uniquement dans des topics bien définis. Un administrateur ou un utilisateur de confiance doit préalablement autoriser la connexion de l'appareil. L'appareil se connecte à AWS IoT Core avec son certificat de demande de connexion et un certificat, un thing et une policy sont créés. L'appareil peut alors se connecter à AWS IoT Core avec son certificat X.509 qui lui a été attribué et retourné par AWS IoT Core [8].

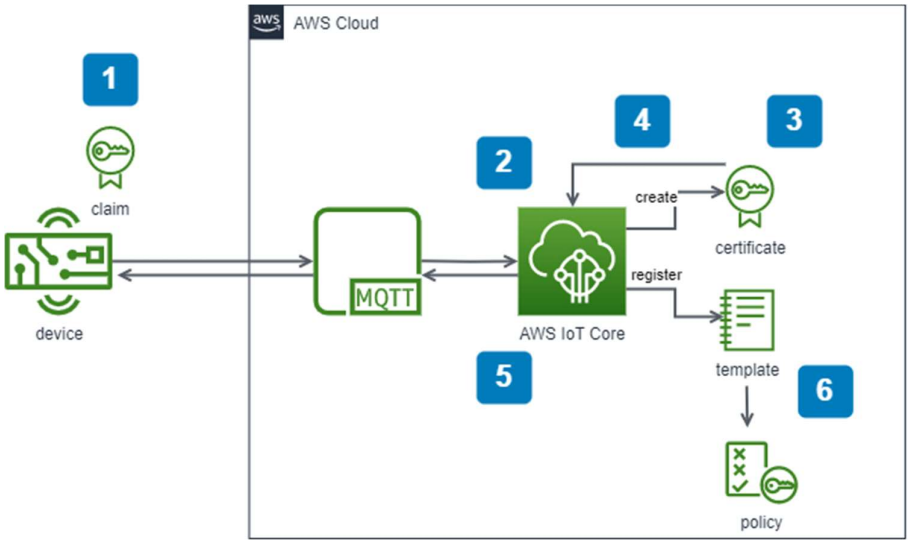

Figure 16 - Provisionnement de flotte par demande

Voici les étapes pas-à-pas de la Figure 16 :

1. Le device est préconfiguré avec un certificat initial, appelé « claim», qui dispose d'une politique de sécurité très restrictive. Ce certificat est utilisé pour établir une première connexion sécurisée avec AWS IoT Core via le protocole MQTT.
2. Une fois connecté, le dispositif fait une demande à AWS IoT Core pour la création d'un nouveau certificat, qui sera utilisé pour sa communication à long terme avec le service.
3. Le certificat est généré par IoT Core.
4. Une fois le certificat généré par IoT Core, il est retourné au device via MQTT.
5. Une fois le certificat reçu, le device change de certificat et rétabli une connexion sécurisée avec IoT Core pour faire une demande de registration.
6. Finalement, le device est enregistré à l'aide d'un template qui permet de définir des attribut à celui-ci et son comportement (police).

10.3.3. Provisioning en temps réel (Just in time provisioning)

L'approvisionnement JITP est une méthode sécurisée, facile et évolutive pour provisionner de nombreux devices. Cette façon de faire permet de provisionner les devices lors de leurs premières connexions. Pour ce faire, il faut créer un certificat CA signé par AWS, activer la registration automatique des devices et associer un template de registration au certificat. Ce template contient diverses informations telles que :

- Le pays
- L'organisation
- L'unité organisationnelle
- Le nom du device
- Le nom de l'état
- Le numéro de série
- L'identifiant du device

Le certificat X.509 contient ces informations. Lors de la première connexion, le certificat racine et le certificat X.509 sont utilisés pour se connecter à AWS IoT Core. Lors de cette première connexion, la communication sera rompue et retournera une erreur. AWS IoT Core va ensuite créer un thing avec les informations du certificat X.509, attacher le certificat X.509 au thing et créer une policy par défaut. Le device peut alors se reconnecter à AWS IoT Core à l'aide du certificat X.509. Cette méthode permet de provisionner des devices sans connaître à l'avance leurs fonctionnalités. Elle permet également de créer des devices en masse [8].

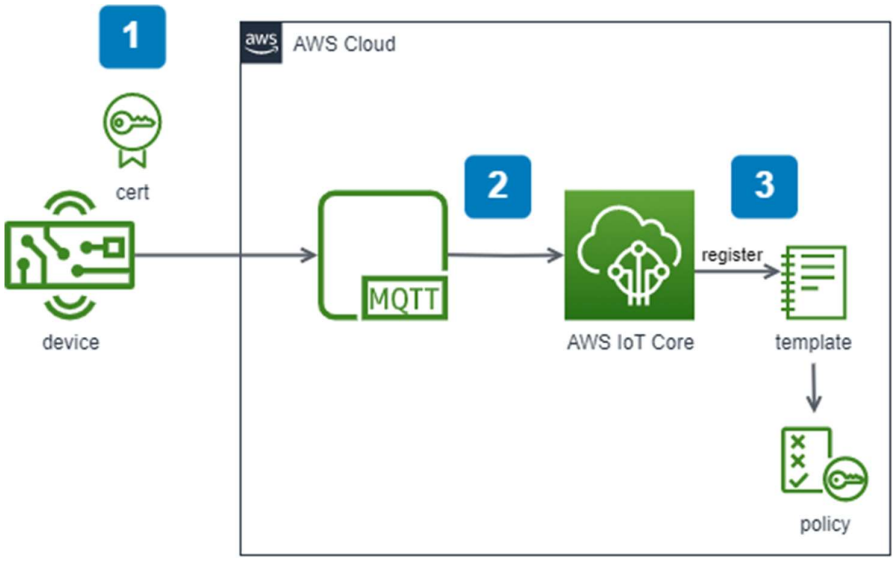

Figure 17 - Provisionnement en temps réel

Voici les étapes pas-à-pas de la Figure 17 :

1. Le device est préconfiguré avec un certificat qui a été signé par une CA connue de AWS.
2. Le device tente de se connecter une première fois, mais ne pourra pas se connecter au broker, car la connexion sera rompue volontairement.
3. AWS IoT Core enregistre le certificat et registre le device si le certificat a bien été signé par une autorité de certification connue. Le device est registré selon un template défini qui permet de définir son comportement comme les autorisations que peut avoir le device. Le device peut désormais communiquer normalement.

10.3.4. Provisionnement d'un seul device (Single thing provisioning)

Le single thing provisioning est une méthode utilisée lors des phases de tests / développements. Elle consiste à créer un thing sur AWS IoT Core depuis la console ou depuis l'API de AWS. Le certificat est téléchargé sur la machine locale puis flashé sur le device.

10.3.5. Choix du provisionnement

Le Tableau 2 compare les différentes méthodes de provisionnement. Dans le cadre d'un tracker GPS moto, la scalabilité du provisionnement est une priorité absolue. Le choix d'un déploiement avec un provisionnement tel que le « Single thing provisioning » doit être de ce fait écarté. Le « fleet provisioning » ou le « JITP » sont des choix judicieux, car la quantité de device produits n'a pas besoin d'être connu en avance, ce qui n'est pas le cas du « Bulk provisioning ».

Le « Fleet provisioning » se distingue au fait de sa facilité de déploiement. En effet, avec ce provisionnement, tout est automatisé avec AWS IoT Core. Avant leurs déploiement, tous les devices possèdent le même certificat et lors de leurs premières connexions AWS IoT Core s'occupe de leur créer un autre certificat unique, ce qui rend le processus de déploiement simple et sûr.

Malgré la modularité et les avantages du « Fleet provisioning », le provisionnement juste à temps ou « JITP » a été préféré. La principale raison de ce choix est liée aux exigences de récupérations en cas de sinistre (voir le chapitre 10.6). Pour garantir une résilience maximale, le tracker doit être capable de se connecter à différentes régions de AWS IoT Core avec le même certificat. Le certificat délivré lors d'un provisionnement par flotte («Fleet provisioning ») est signé par le CA spécifique de la région AWS, ce qui rend impossible la connexion sur une autre région avec le même certificat.

En revanche, avec le « JITP », il est possible d'utiliser sa propre autorité de certification (CA). Il est donc possible de rendre valide un certificat sur plusieurs régions AWS, qui, en cas de désastre, offre une architecture cloud bien plus résiliente.

Nom du provisionnement	Scalabilité	Facilité d'implémentation	Remarques
Single thing provisioning	Faible	Facile	Idéal pour les petits projet. Impossible d'utiliser ce provisionnement en cas de gros déploiement
Bulk provisioning	Élevée	Moyenne	Crée de nombreux dispositifs en une seule opération.
JITP	Très élevée	Difficile	Automatise le processus de provisionnement. Nécessaire de créer des certificats uniques avant le déploiement et de les flasher sur les devices.
Fleet provisioning	Extrêmement élevée	Difficile	Automatise entièrement le processus de provisionnement et de la création des certificats unique par appareils.

Tableau 2 - Comparatifs entre les différents moyen de provisionnement fournit par AWS

10.3.6. Création d'un Root CA

Les commandes suivantes ont été repris d'une page AWS expliquant comment faire du JITP [9].

Pour créer un Root CA il faut tout d'abord créer une clef privée [9]:

```
$ openssl genrsa -out deviceRootCA.key 2048
```

Un fichier de configuration peut-être crée en créant [9] :

```
$ vi deviceRootCA_openssl.conf
```

Le fichier de configuration doit contenir ces différents paramètres [9] :

```
[ req ]
distinguished_name      = req_distinguished_name
extensions              = v3_ca
req_extensions          = v3_ca
```

```
[ v3_ca ]
basicConstraints         = CA:TRUE

[ req_distinguished_name ]
countryName              = Country Name (2 letter code)
countryName_default      = IN
countryName_min          = 2
countryName_max          = 2
organizationName         = Organization Name (eg, company)
organizationName_default = AMZ
```

Une demande de signature de certificat (CSR) peut être ensuite créée comme ceci [9]:

```
$ openssl req -new -sha256 -key deviceRootCA.key -nodes -out deviceRootCA.csr -
config deviceRootCA_openssl.conf
```

Le certificat racine peut ensuite être crée comme suit [9] :

```
openssl x509 -req -days 3650 -extfile deviceRootCA_openssl.conf -extensions v3_ca
-in deviceRootCA.csr -signkey deviceRootCA.key -out deviceRootCA.pem
```

Ce certificat doit ensuite être enregistré sur AWS IoT Core et ajouté au template de provisionnement juste à temps.

10.3.7. Création d'un certificat device

Ce chapitre aborde la création d'un certificat pour un appareil nécessaire pour s'identifier auprès de AWS. Pour ce faire une liste d'étape a été créée (reprise d'un exemple de AWS [9]) :

Il faut tout d'abord créer la clef privée [9] :

```
$ openssl genrsa -out deviceCert.key 2048
```

Ensuite une demande de signature de certificat d'appareil doit être créée [9]:

```
$ openssl req -new -key deviceCert.key -out deviceCert.csr
```

Le "Common Name" correspond au nom de l'appareil IoT [9] :

```
Common Name (eg. server FQDN or YOUR name) []: DemoThing
```

Le certificat peut ensuite être créé [9] :

```
$ openssl x509 -req -in deviceCert.csr -CA deviceRootCA.pem -CAkey
deviceRootCA.key -CAcreateserial -out deviceCert.crt -days 365 -sha256
```

10.3.8. Provisionner le device IoT

Pour provisionner le device IoT avec son certificat, il existe plusieurs façons d'opérer. La première serait de flasher le certificat sur une flash externe qui serait ensuite soudée au PCB. Lors de la première initialisation de l'appareil, le certificat est téléchargé par le MCU puis transféré au SARA R5. Le MCU s'occupe ensuite de supprimer le certificat de la flash externe. La deuxième solution est de donner le certificat à travers le port série du MCU.

10.3.9. Template pour le JITP

Le template pour le modèle d'enregistrement juste à temps contient différents paramètres qui doivent être récupérés dans le certificat du device qui doit être provisionné. C'est depuis celui-ci que le « Common Name »

est attribué au ThingName et que le groupe de l'objet lui est attribué. Il contient aussi le nom de la police qui sera attribué aux objets.

```
{
  "Parameters": {
    "AWS::IoT::Certificate::CommonName": {
      "Type": "String"
    },
    "AWS::IoT::Certificate::Id": {
      "Type": "String"
    }
  },
  "Resources": {
    "thing": {
      "Type": "AWS::IoT::Thing",
      "Properties": {
        "ThingName": {
          "Ref": "AWS::IoT::Certificate::CommonName"
        },
        "ThingGroups": [
          "MyIotThingGroup"
        ]
      },
      "OverrideSettings": {
        "AttributePayload": "MERGE",
        "ThingTypeName": "REPLACE",
        "ThingGroups": "DO_NOTHING"
      }
    },
    "certificate": {
      "Type": "AWS::IoT::Certificate",
      "Properties": {
        "CertificateId": {
          "Ref": "AWS::IoT::Certificate::Id"
        },
        "Status": "ACTIVE"
      }
    },
    "policy": {
      "Type": "AWS::IoT::Policy",
      "Properties": {
        "PolicyName": "SaasIotPolicy"
      }
    }
  }
}
```

Figure 18 - Template pour le JITP

10.4. Mise à jour Over-The-Air (FOTA AWS Jobs)

La mise à jour d'un appareil IoT est une fonctionnalité critique. Elle permet de résoudre des problèmes firmware et d'y apporter de nouvelles fonctionnalités et des améliorations sans avoir à rappeler physiquement les appareils. Une approche de mise à jour à distance a été étudiée durant ce projet d'approfondissement, testée, mais non réellement implémentée. La mise à jour du firmware à distance a été pensée autour des AWS IoT Jobs qui permets un contrôle à distance d'une flotte. Ce contrôle ou « job » peut ordonner à une flotte d'appareils divers comportements spécifiques. Par exemple le téléchargement d'un firmware sur une base de données S3 AWS, un reboot à distance, la rotation de certificats ou bien d'effectuer une gestion d'erreurs distante.

Nordic semiconducteurs fournissent un digramme séquentiel détaillé expliquant le fonctionnement d'une mise à jour à travers le cloud visible sur le diagramme séquentiel visible à la Figure 19.

Le processus d'une mise à jour à distance peut être décrit comme suit :

1. **Création du job pour une flotte d'appareil :** Un job est créé dans la console AWS IoT. Il est basé sur un template qui contient les informations à fournir à l'appareil, permettant de décrire la nature du job et les informations nécessaires pour effectuer le mise à jour tel que l'URL du fichier de firmware hébergé sur une S3 de AWS.
2. **Notification du job :** AWS notifie dans le topic « $aws/things/<thingName>/jobs/notify-next » existence d'un nouveau job avec ses informations sous forme d'un JSON.
3. **Acceptation du job :** Le device reçoit la notification et décide de l'accepter ou non.
4. **Exécution du job :** Le device exécute le job.
5. **Mise à jour de l'état du job :** Le device tient au courant AWS Job de l'état du job (IN_PROGRESS, FAILED, SUCCEEDED).

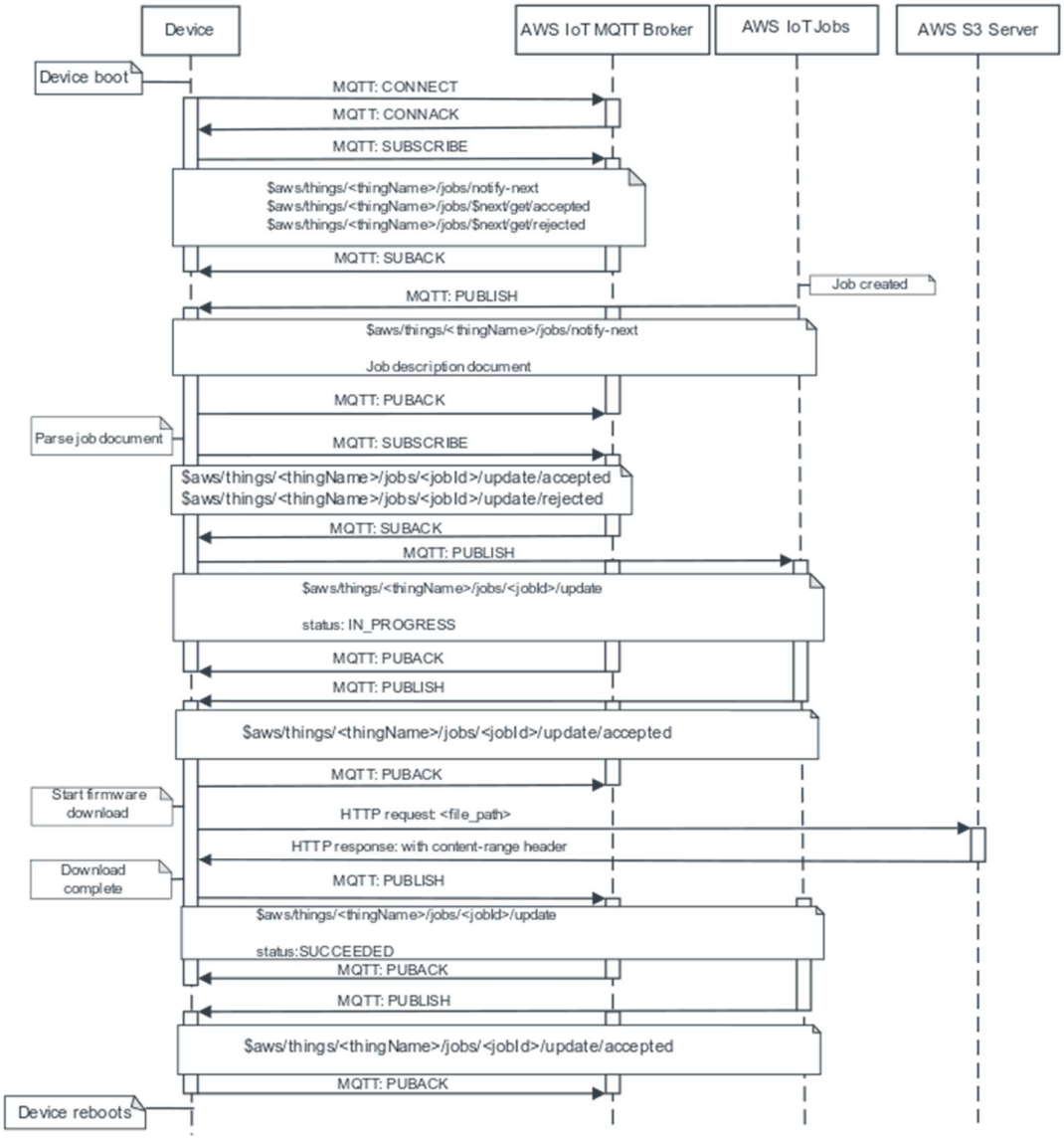

Figure 19 - AWS Firmware Over-the-Air[3]

[3] Illustration de Nordic Semiconductor :
https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/nrf/libraries/networking/aws_fota.html

10.4.1. Template

Lors de la création du job, un paramètre se nommant « jobDocument » contient une structure JSON définissant les paramètres de l'action à distance à effectuer. Cette structure contient l'opération à effectuer, la version, la taille du document à télécharger, et le lien du document à télécharger. Le lien du document, stocké sur une S3, peut être remplacé par une URL temporaire présignée grâce à ***« ''document'' : ''${aws:iot:s3-presigned-url:https://s3.<REGION>.amazonaws.com/<S3NAME>/<FILENAMEINS3>'' »***. La durée de visibilité de cette URL peut être paramétrée de 1 minute à 12 heures lors de la création du Job. Malheureusement, l'URL présignée générée est de longueur bien supérieure à 1024 bytes, la taille maximale autorisée par le module SARA R5. Il n'est donc pas possible d'utiliser directement cette URL. Une solution pour pallier ce problème est d'utiliser l'API de bit.ly pour obtenir une URL plus courte. Durant ce projet, l'implémentation d'un raccourcisseur d'URL a été réalisée, mais rapidement écartée, faute de temps. Deux faux firmwares (fichiers textes contenant un string) ont été hébergés sur un serveur Infomaniak :

1. www.bambinobar.ch/blue.txt
2. www.bambinobar.ch/green.txt

```
{
    "operation": "fwupdate",
    "version": "v1.0.2",
    "size": 1024,
    "document": "https://bambinobar.ch/green.txt"
}
```

Figure 20 - Template pour FOTA (green)

```
{
    "operation": "fwupdate",
    "version": "v1.0.2",
    "size": 1024,
    "document": "https://bambinobar.ch/blue.txt"
}
```

Figure 21 - Template pour FOTA (blue)

10.4.2. Création d'un Job avec la console de AWS

Lors de la création d'un job depuis la console AWS, un nom doit lui être attribué. Ensuite, le job doit déterminer une cible comme par exemple un device ou un groupe ainsi que le fichier d'une tâche. Ce fichier est en réalité le template réalisé au chapitre 10.4.1. L'attribution de ces deux paramètre est illustré à la Figure 22.

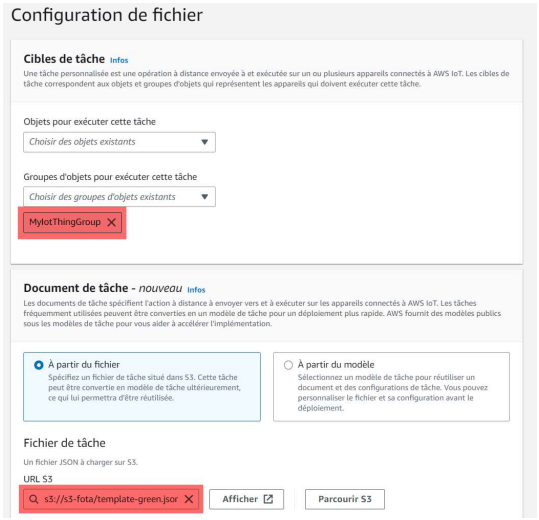

Figure 22 - Attribution d'un job au groupe "MyIotThingGroup" avec la tâche template-green.json

10.5. AWS Location Service

La partie de l'architecture contenant l'asset de suivi (en mode « distance based » ce qui permet d'éviter de sauvegarder plusieurs points au même endroit) et la carte de AWS Location Service fait partie de la stack Amplify (voir chaptire 10.1). Elle a été reprise d'un exemple de AWS [10] démontrant des fonctionnalités de AWS Location Service. Ce service fourni par AWS permet la gestion et l'intégration facile de carte ainsi que le stockage de points de géolocalisation dans un asset de tracking. L'exemple proposé par Amazon permet à des utilisateurs autorisés de pouvoir consulter sur une carte l'historique de position d'un tracker à l'aide de points reliés entre eux (voir un exemple de trajet Aigle-Saint-Léonard à la Figure 24). Cette Stack est créé à l'aide de Amplify qui permet d'authentifier l'utilisateur grâce à un backend.

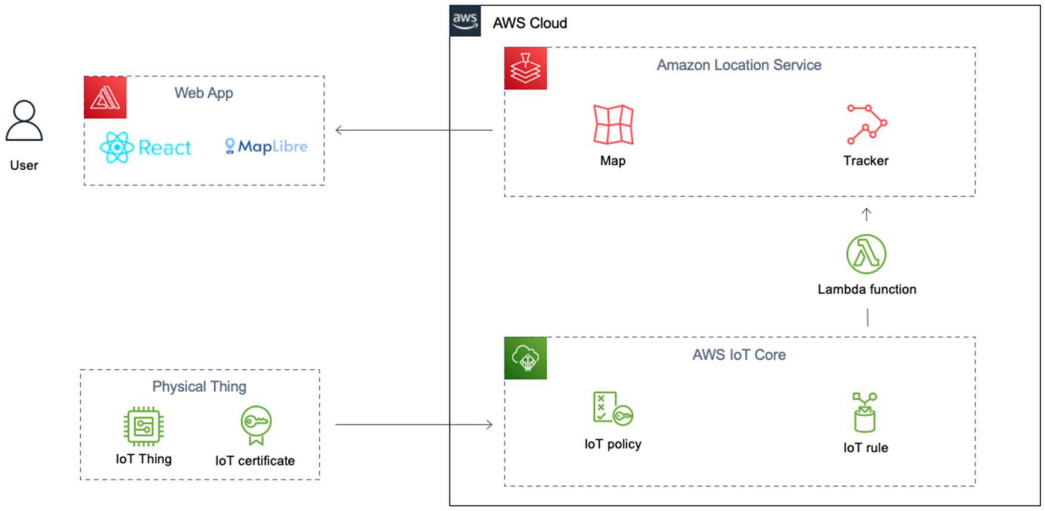

Figure 23 - Stack Amplify[4]

[4] Illustration tirée du repository d'exemple AWS Location Service : https://github.com/aws-samples/amazon-location-samples/tree/main/maplibre-js-react-iot-asset-tracking

Figure 24 - Affichage d'un tracé Aigle-Saint-Léonard réalisé en train

10.6. Plan de reprise après sinistre (Disaster recovery)

Un plan de reprise après sinistre est une architecture permettant d'implémenter des procédures pour limiter la perte de données et de restaurer un service après un sinistre. La Figure 25 illustre la résilience d'un service. Elle y illustre l'architecture d'un système résilient, doté d'un plan de reprise après sinistre et mécanisme de haute disponibilité. En cas d'évènement catastrophique, le plan de reprise après sinistre est activé pour rétablir le service efficacement. Le mécanisme de haute disponibilité quant à lui permet de prévenir toute interruption de service en assurant un fonctionnement continu. [11]

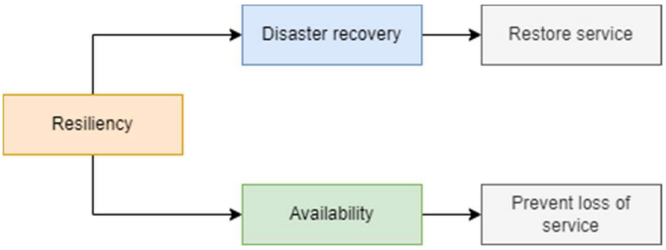

Figure 25 - Resilience d'un service

10.6.1. Régions AWS

La Figure 26 illustre les différentes régions de AWS tout autour du globe. À l'intérieur de chacune de ces régions se trouvent des « zones de disponibilités. Leurs nombres varient en fonction des régions. Elles sont physiquement séparées d'environ une centaine de kilomètres pour pouvoir permettre la réplication de donnée entre les zones et de se protéger entre elles d'un désastre. [11]

Figure 26 - AWS Régions[5]

10.6.2. Types de désastres

Un désastre peut être :

1. Naturel : tremblement de terre, inondation, éruption volcanique
2. Technique : connexion internet, alimentation
3. Humaine : suppression d'une table

Chacun de ces désastres à un impact géographique, il peut être local, régional, à l'échelle d'un pays ou même globale. [11]

[5] Illustration prise de AWS : https://aws.amazon.com/fr/about-aws/global-infrastructure/?p=ngi&loc=1

10.6.3. Objectif du point de récupération (RPO)

Le « Recovery Point Objective » est un concept crucial d'un plan de reprise après sinistre. Il correspond au temps maximal de donnée qu'une organisation puisse se permettre de perdre lors d'un sinistre. Le PRO indique donc l'âge maximal des fichiers qui doivent être récupérés lors d'une sauvegarde. Pour exemple, si le RPO est défini à 5 heures, cela signifie qu'une sauvegarde doit être effectuée toutes les 5 heures. En cas de sinistre l'entreprise doit être prête à accepter la perte des données des 5 dernières heures. Un faible RPO à un cout bien plus élevé qu'un faible RPO. [11]

10.6.4. Objectif de délai de récupération (RTO)

Le « Recovery Time Objective » est un autre concept crucial lors d'un plan de reprise après sinistre. Le RPO définit le délai acceptable pour restaurer le service. Il s'agit donc du temps que peut se permettre une entreprise d'être hors ligne sans conséquences majeures. Si le RTO est fixé à 2 heures, l'entreprise doit être capable de revenir en ligne et d'opérer normalement en moins de 2 heures. [11]

10.6.5. Architecture choisie

L'architecture choisie est une architecture hybride. Elle consiste à l'établissement d'une seconde région qui reste en mode passif et ne s'active qu'en cas de sinistre. Au fur à mesure du provisionnement de l'entièreté de la flotte sur la région principale, le certificat de chaque tracker est ajouté à la région secondaire. Cela permet en cas de sinistre un changement rapide et presque sans interruption grâce à une sauvegarde.

10.7. Révocation d'un certificat

Si un utilisateur souhaite résilier son abonnement, ou cesse de payer celui-ci, une révocation du certificat permet de suspendre temporairement le compte de celui-ci. Il est possible d'effectuer automatiquement cette action à l'aide de Boto3.

10.8. Coûts

Les coûts de cette architecture ont été déterminés à l'aide de trois profils types de motard :

- Utilisations tous les jours de l'année : 8 heures par jours d'active tracking (chaque 30 secondes) 16 heures par jours en mode repos (tracking chaque).
- Utilisation fréquente :Utilisation pendant 8 mois chaque année tous les jours, 4 heures par jour d'active tracking (chaque 30 secondes), 20 heures par jours en mode repos.
- Utilisation occasionnelle : 1 heure de voyage par jour, 8 mois par année.

10.8.1. Scénario 1 : utilisation intensive

Pour le scénario d'une utilisation intensive et quotidienne, on obtient le nombre de positions à envoyer au broker par jour :

$$nPosSentDay = 8 * \frac{3600}{30} + 16 * \frac{60}{30} = 976$$

Dans ce scénario l'utilisateur consulte une fois par jour l'historique de son tracé. Le nombre de points lus peut-être donc estimé comme suivant :

$$nReadPosDay = 8 * \frac{3600}{30} = 960$$

Région	Service	Frais mensuel / 10'000 devices	Frais mensuel / 1 device	Details
Europe (eu-central-1)	Maps	973.33	0.09	Mappage des mosaïques de carte extraites (Vectoriel ou Raster) (800000 par jour)
Europe (eu-central-1)	Tracking	29443.33	2.94	Positions lues (9600000 par jour), Positions écrites (9760000 par jour), Appareils supprimés (0 millier par mois)
Europe (eu-central-1)	AWS Lambda	12.12	0.001212	Architecture (x86), Architecture (x86), Mode d'appel (Mise en tampon), Nombre de requêtes (9760000 par mois), Volume de magasin éphémère alloué (512 Mo)
Europe (eu-central-1)	MQTT	514.04	0.0514	Nombre d'appareils (MQTT) (10000), Taille moyenne de chaque message (2 Ko), Taille moyenne de chaque enregistrement (1 Ko), Taille moyenne de chaque enregistrement (1 Ko), Nombre moyen d'actions exécutées par règle (1), Taille moyenne de chaque message (1 Ko), Nombre de messages pour un appareil (30256), Nombre de règles déclenchées (30256)
Europe (eu-central-1)	AWS IoT Device Management	36.12	0.003612	Taille moyenne de chaque mise à jour d'index (1 Ko), Nombre d'actions à distance (10000), Nombre d'objets enregistrés (1000)
Europe (eu-central-1)	DynamoDB on-demand capacity	(7817.42)	0.7817	Classe de table (Standard), Taille moyenne d'élément (tous les attributs) (10 Ko), Taille du stockage des données (100 Go)
		TOTAL / mois / 10000 devices	TOTAL / mois / 1 device	
		39796.36 USD	**3.879 USD**	

Tableau 3 - Coûts du scénario 1 pour 10'000 devices [12]

10.8.2. Scénario 2 : utilisation fréquente

Pour le scénario d'une utilisation fréquente (4 heures par jours de voyage) et quotidienne, on obtient le nombre de positions à envoyer au broker par jours :

$$nPosSentDay = 4 * \frac{3600}{30} + 20 * \frac{60}{30} = 520$$

Si l'utilisateur ne voyage uniquement 8 mois par année on peut obtenir une moyenne d'envoi par jour :

$$nPosSentDayAvg = nPosSentDay * \frac{8}{12} = 346$$

Dans ce scénario l'utilisateur consulte une fois par jour l'historique de son tracé. Le nombre de points lus peut-être donc estimé comme suivant :

$$nReadPosDay = 4 * \frac{3600}{30} * \frac{8}{12} = 320$$

Région	Service	Frais mensuel / 10'000 devices	Frais mensuel / 1 device	Details
Europe (eu-central-1)	Maps	973.33	0.09	Mappage des mosaïques de carte extraites (Vectoriel ou Raster) (800000 par jour)
Europe (eu-central-1)	Tracking	10128.75	1.01	Positions lues (3200000 par jour), Positions écrites (3460000 par jour), Appareils supprimés (0 millier par mois)
Europe (eu-central-1)	AWS Lambda	4.29	0.000429	Architecture (x86), Architecture (x86), Mode d'appel (Mise en tampon), Nombre de requêtes (3460000 par mois), Volume de magasin éphémère alloué (512 Mo)
Europe (eu-central-1)	MQTT	209.38	0.0209	Nombre d'appareils (MQTT) (10000), Taille moyenne de chaque message (2 Ko), Taille moyenne de chaque enregistrement (1 Ko), Taille moyenne de chaque enregistrement (1 Ko), Nombre moyen d'actions exécutées par règle (1), Taille moyenne de chaque message (1 Ko), Nombre de messages pour un appareil (10726), Nombre de règles déclenchées (10726)
Europe (eu-central-1)	AWS IoT Device Management	36.12	0.003612	Taille moyenne de chaque mise à jour d'index (1 Ko), Nombre d'actions à distance (10000), Nombre d'objets enregistrés (1000)
Europe (eu-central-1)	DynamoDB on-demand capacity	(2791.09)	0.279	Classe de table (Standard), Taille moyenne d'élément (tous les attributs) (10 Ko), Taille du stockage des données (100 Go)
		TOTAL / mois / 10000 devices	TOTAL / mois / 1 device	
		14'142.96 USD	**1.41 USD**	

Tableau 4 - Coûts pour le scénario 2 pour 10'000 devices [12]

10.8.3. Scénario 3 : utilisation occasionnelle

Pour le scénario d'une utilisation fréquente (1 heures par jours de voyage) et quotidienne, on obtient le nombre de positions à envoyer au broker par jour :

$$nPosSentDay = 1 * \frac{3600}{30} + 23 * \frac{60}{30} = 166$$

Si l'utilisateur ne voyage uniquement 8 mois par année on peut obtenir une moyenne d'envoi par jour :

$$nPosSentDayAvg = nPosSentDay * \frac{8}{12} = 110$$

Dans ce scénario l'utilisateur consulte une fois par jour l'historique de son tracé. Le nombre de points lus peut-être donc estimé comme suivant :

$$nReadPosDay = 1 * \frac{3600}{30} * \frac{8}{12} = 96$$

Région	Service	Frais mensuel / 10'000 devices	Frais mensuel / 1 device	Details
Europe (eu-central-1)	Maps	973.33	0.09	Mappage des mosaïques de cartes extraites (Vectoriel ou Raster) (800000 par jour)
Europe (eu-central-1)	Tracking	3132.92	0.31	Positions lues (960000 par jour), Positions écrites (1100000 par jour), Appareils supprimés (0 millier par mois)
Europe (eu-central-1)	AWS Lambda	1.37	0.000137	Architecture (x86), Architecture (x86), Mode d'appel (Mise en tampon), Nombre de requêtes (1100000 par mois), Volume de magasin éphémère alloué (512 Mo)
Europe (eu-central-1)	MQTT	95.25	0.009525	Nombre d'appareils (MQTT) (10000), Taille moyenne de chaque message (2 Ko), Taille moyenne de chaque enregistrement (1 Ko), Taille moyenne de chaque enregistrement (1 Ko), Nombre moyen d'actions exécutées par règle (1), Taille moyenne de chaque message (1 Ko), Nombre de messages pour un appareil (10726), Nombre de règles déclenchées (10726)
Europe (eu-central-1)	AWS IoT Device Management	36.12	0.003612	Taille moyenne de chaque mise à jour d'index (1 Ko), Nombre d'actions à distance (10000), Nombre d'objets enregistrés (1000)
Europe (eu-central-1)	DynamoDB on-demand capacity	(1696.39)	0.169	Classe de table (Standard), Taille moyenne d'élément (tous les attributs) (10 Ko), Taille du stockage des données (100 Go)
		TOTAL / mois / 10000 devices	TOTAL / mois / 1 device	
		4'935.39 USD	**0.5935 USD**	

Tableau 5 - Coûts pour le scénario 3 pour 10'000 devices [12]

10.8.4. Synthèse

Selon les différents scénarios, on peut observer que les frais maximum pour un motard roulant tous les jours seraient d'environ 3.9 USD par mois. Cependant, la grande majorité des motards n'utilise que rarement leurs véhicules et principalement uniquement en été. Ce qui permet de penser qu'une grande majorité des utilisateurs tomberait dans le scénario 2 ou le scénario 3, soit de 60ct à 1.4 USD par mois.

De plus il faut savoir que ces estimations sont pessimistes, car elle estime qu'à l'arrêt une position est ajoutée au tracker chaque 30 minutes ce qui n'est réellement pas le cas, car l'ajout d'une position au tracker utilise une approche de « distance based », ce qui permet de ne pas ajouter de multiples positions au même endroit.

11. Tests et validation

11.1. Test en condition réel

Dans ce chapitre, des captures d'écrans de la communication entre le processeur applicatif et les modules environnants permettent de prouver le bon fonctionnement du démonstrateur.

La Figure 27 montre l'envoi des certificats au module SARA R5. L'envoi de ce certificat ne se produit qu'une seule fois pour envoyer les certificats dans le secure element du module 4G.

La Figure 28 et Figure 29 démontrent le fonctionnement de la fausse mise à jour du firmware à distance grâce a AWS IoT Jobs. On peut y observer la réception du message sur le topic concerné, sa lecture ainsi que l'ouverture d'une connexion HTTP pour récupérer le firmware qui se trouve au lien : bambinobar.ch/blue.txt. Le socket va ensuite télécharger le fichier et allumer la LED en bleu après avoir lu le contenu du fichier texte.

La Figure 30, montre la connexion au broker MQTT via le module SARA R5.

La Figure 31 illustre le callback appelé lorsque le module détecte un mouvement. Une fois le callback appelé, le module passe en mode tracking actif (toutes les 30 secondes).

La Figure 32 démontre le bon fonctionnement de la communication du processeur avec le module GPS lorsque le GPS cherche a verouiller les satellites.

La Figure 33 montre la lecture de la position GPS et l'envoi de la position GPS au topic MQTT.

```
MYPX5xXvXDgFOKVnTX24/cUZH4eqyE6kDz8=-----END CERTIFICATE-----Waiting
 for thread to finish
Waiting for thread to finish

+USECMNG: 0,1,"aws_certificate_pem","c991a70d16d1e5558953d795dd23336
e"

OK
retval 1 : 0AT+USECMNG=0,2,"aws_private_key_pem",1679
>Waiting for thread to finish
-----BEGIN RSA PRIVATE KEY-----MIIEpAIBAAKCAQEAuuJmiEjtI9rrtlRPzsLjh
MEwiN1xWrULQGaSXtTXGc2cm45Txi/hlzPsc24l5ybPMu7GfWY7O520U+/B2mCmP+m3A
APnWao33Tlc1TK3H223Ykf3dxMZmSgC2QI/KNj8WGn6iL/O35cn9gL8Tq8o90S3CrXs6
gCh6TNG+zbYOb882qWg7rvhKsklzZMQM2zuiLAi64xvknzyceoGrBfU/WxQEl8oRZHoW
ezW1dYeQqfLivv1i4GfhI8dSeQCAeEhKyYoNOQTj5SwKJPpNygLiGi69jd30u8XtiG3I
mrFItOvRcmhsEKCzugjwRksucQmMCoZMB5Tk2DHx8/mE6ERuwIDAQABAoIBAFeIQ04CG
lB48X5s7xSA4+ACPKFeVwGNWr/HSzdvqXBEhd22RDbHpnoTr+RA6y/hVZsdi4qgn43Ex
/moirumGThGV5iCO5Ib0B6mB300S+FEmmhGHPqAz7SUsQDDtKGFQGyI9vemH1dKjUUFE
Pgpwp5OaQ1zOIxBz5RSmCSCA7THt6x6SJguiPVfy8G0T/7gFy+9AIYfhU/QWo6hhhHIo
+IwOyG3eLzhUX9WBJ79klwg3CTEouOGkJDPd7dNGic2hAqYkEai9lUJN+sK1+qgfobI2
Y8jG8D0GClreW4zfmWHbiEuiu7XfmbWTOvBGDCgiQIV346s+zNi/SZBVThF4cECgYEA9
DP6w5hLNWarFbIaZL890Vq68XNPvUHG/Y5yvhLw9IkixS6nWvlF1q0YDbzyZctBA2KpA
1QAR8OY3u0eDHFlZVn6c5vIg0Jzz7KBEAyAJElOuQRmTWyNEGOqYWMttZg4a+mH1tyrd
M66S1ahzGSDhE3hl+7tJEwCT83CECH+X9UCgYEAw+mR76ot8VI439trtk1rcaWdyhig8
jzIBIJbJ+DipJV6mmszvreQVf1dZIKe2wM3nAJr1csTOQBpQy0pn/6ye46/Q3MeH9A8D
odDmFZJWmi9Q/8OH1uCT6qBL4DQvN5KPGqhGWBYshN9Al1n7m3/xJ9fpILOOZ38L/oRE
3H6A08CgYBsl5zAU73F/mS/3hZcjKpZqYYODp/74f8yC6E8osGhjEX49zkWfFixEQHsQ
gRsfcjoT3wm7bi+MN2O7FR7ZgWX3z4/IJw2ulszlgZMc7Kb8DlgjvNLJKVKD3fZnLfb0
9frLMrqIfDgnXk8PUbv1Bajw9o4rNepV1+Dy1SrxeIPDQKBgQC8j8S8rklF6eN/4osLj
fyKB+OVLptKS6IgZO8jtNmvm4fDtMa7G6a6fQA0y1vmZHRTnij4P1sjE/g8he14ZwaW0
c/ZNaWa7SvAuAQhTXuQnxr7pTOMjuEfmHZ6qDtAbZt6Bw9Dg24bB5LcgdaUsB7qupKtn
pCq+BpQexmQY7VFFQKBgQDIb6dhmuTnHWCLgwBmrD7VXqKb99dTKps755SyUcUjLPoOu
az4zLGbnGLXpWOw7ggDKjnbf5voL71VgGmw44/+3R4ay8TdaHHAmjDQZLC3Y1Q7CQ6TQ
jGKot/HjTdeiJQQZgIuSOsn7aoQVEf3r+3E8ahD9o0OzWBHAnZ5+cZsRQ=-------END
RSA PRIVATE KEY-----Waiting for thread to finish

+USECMNG: 0,2,"aws_private_key_pem","1f9811356a3c575dc8e03aa924fe549
3"

OK
retval: 0AT+USECMNG=0,0,"aws_root_ca_pem",1171
```

Figure 27 - Chargement des certificats dans le SARA-R5

MQTT Callback called

Topic: $aws/things/fleet2/jobs/notify-next

retval: 0AT+UMQTTC=6,1

+UMQTTC: 6,1,313,35,"$aws/things/fleet2/jobs/notify-next",278,"{"timestamp":1685713283,"execution":{"jobId":"FOTABLUUUU","status":"QUEUED","queuedAt":1685713282,"lastUpdatedAt":1685713282,"versionNumber":1,"executionNumber":1,"jobDocument":{"operation":"fwupdate","version":"v1.0.2","size":1024,"document":"https://bambinobar.ch/blue.txt"}}}"

OK

Success parsing

Status and operation correct

Topic to subscribe: $aws/things/fleet2/jobs/FOTABLUUUU/update/accepted

Subscribing to topic $aws/things/fleet2/jobs/FOTABLUUUU/update/accepted

AT+UMQTTER

+UMQTTER: 13,0

OK

AT+UMQTTC=4,1,"$aws/things/fleet2/jobs/FOTABLUUUU/update/accepted"

OK

+UUMQTTC: 4,1,1,"$aws/things/fleet2/jobs/FOTABLUUUU/update/accepted"

Device is moving, so 30 seconds sending

Setting callback

Callback struct: $aws/things/fleet2/jobs/FOTABLUUUU/update/accepted

Topic to publish: $aws/things/fleet2/jobs/FOTABLUUUU/update

String to publish: {"status":"IN_PROGRESS"}

Publishing message to topic $aws/things/fleet2/jobs/FOTABLUUUU/update

Message: {"status":"IN_PROGRESS"}

AT+UMQTTER

+UMQTTER: 13,0

OK

AT+UMQTTC=9,1,0,"$aws/things/fleet2/jobs/FOTABLUUUU/update",24

>{"status":"IN_PROGRESS"}

Figure 28 - Callback lors d'un FOTA

```
AT+UHTTP=0

OK

AT+UHTTP=0,1,"bambinobar.ch"

OK

AT+UHTTP=0,4,0

OK

AT+UHTTP=0,7,30

OK

AT+UHTTP=0,6,0

OK

AT+UHTTP=0,5,80

OK

HTTP Client opened

/blue.txt

AT+UHTTPC=0,1,"/blue.txt","ubxlibhttp_0"

OK

+UUHTTPCR: 0,1,1

Status: 200

Read buffer: blue

Closing HTTP Client

AT+ULSTFILE=0

+ULSTFILE: "ubxlibhttp_0","ubxlibhttp_1","ubxlibhttp_2"

OK

AT+UDELFILE="ubxlibhttp_0"

OK

HTTP Client closed

AT+UMQTTER

+UMQTTER: 13,0

OK

AT+UMQTTC=9,1,0,"$aws/things/fleet2/jobs/FOTABLUUUU/update",22

>{"status":"SUCCEEDED"}

Received message: {"timestamp":1685713310}
```

Figure 29 - Récupération via HTTP d'un faux firmware

```
AT+USECPRF=0,10,"a16q5rw5yauh4k-ats.iot.eu-central-1.amazonaws.com"

OK
Broker name: a16q5rw5yauh4k-ats.iot.eu-central-1.amazonaws.com:8883
AT+UMQTT=2,"a16q5rw5yauh4k-ats.iot.eu-central-1.amazonaws.com",8883

OK
AT+UMQTT=0,"351457831220931"
Waiting for thread to finish

OK
AT+UMQTT=11,1,0

OK
U_CELL_MQTT: trying to connect...
AT+UMQTTC=1

OK
U_CELL_MQTT: waiting for response for up to 30 second(s)...

+UUMQTTC: 1,1
U_CELL_MQTT: connected after 3 second(s).
AT+UMQTT=10

+UMQTT: 10,0,10

OK
Connection to MQTT broker success
Subscribing to topic $aws/things/fleet2/jobs/notify-next
AT+UMQTTER

+UMQTTER: 13,0

OK
AT+UMQTTC=4,1,"$aws/things/fleet2/jobs/notify-next"

OK

+UUMQTTC: 4,1,1,"$aws/things/fleet2/jobs/notify-next"
Setting callback
Callback struct: $aws/things/fleet2/jobs/notify-next
```

Figure 30 - Connexion au broker MQTT AWS

```
Raw values: X = 0, Y = -1, Z = -10
Triggered
Accel: X = 0.107 g, Y = -0.176 g, Z = -1.130 g
Raw values: X = 1, Y = -1, Z = -11
Triggered
Accel: X = 0.107 g, Y = -0.176 g, Z = -1.130 g
Raw values: X = 1, Y = -1, Z = -11
Triggered
Accel: X = 0.103 g, Y = -0.203 g, Z = -1.061 g
Raw values: X = 1, Y = -2, Z = -10
Triggered
Accel: X = 0.019 g, Y = -0.153 g, Z = -1.046 g
Raw values: X = 0, Y = -1, Z = -10
Triggered
Accel: X = 0.019 g, Y = -0.153 g, Z = -1.046 g
Raw values: X = 0, Y = -1, Z = -10
Triggered
Accel: X = 0.023 g, Y = -0.157 g, Z = -1.050 g
Raw values: X = 0, Y = -1, Z = -10
Triggered
Accel: X = 0.084 g, Y = -0.180 g, Z = -1.210 g
Raw values: X = 0, Y = -1, Z = -12
Triggered
Accel: X = 0.084 g, Y = -0.180 g, Z = -1.210 g
Raw values: X = 0, Y = -1, Z = -12
Triggered
Accel: X = 0.138 g, Y = -0.184 g, Z = -1.187 g
Raw values: X = 1, Y = -1, Z = -11
_
```

Figure 31 - Détection de mouvements

```
 00 00 00 00 00 00 00 00 00 00 00 00 20 4e 00 00 80 a8 12 01 0f 2
7 00 00 d2 7a 5b 21 00 00 00 00 00 00 00 00 [body 92 byte(s)].
U_GNSS_POS: UTC time = 1685712776.
U_GNSS: sent command b5 62 01 07 00 00 08 19.
Waiting for thread to finish
U_GNSS: decoded UBX response 0x01 0x07: 78 59 a8 1c e7 07 06 02 0d 2
0 39 f3 ff ff ff ff e0 fe ff ff 00 00 e4 00 00 00 00 00 00 00 00 00
00 00 00 00 98 bd ff ff ff ff ff ff 00 76 84 df 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 20 4e 00 00 80 a8 12 01 0f 2
7 00 00 d2 7a 5b 21 00 00 00 00 00 00 00 00 [body 92 byte(s)].
U_GNSS_POS: UTC time = 1685712777.
U_GNSS: sent command b5 62 01 07 00 00 08 19.
Waiting for thread to finish
U_GNSS: decoded UBX response 0x01 0x07: 60 5d a8 1c e7 07 06 02 0d 2
0 3a f3 ff ff ff ff 3a ff ff ff 00 00 e4 00 00 00 00 00 00 00 00 00
00 00 00 00 98 bd ff ff ff ff ff ff 00 76 84 df 00 00 00 00 00 00 00
 00 00 00 00 00 00 00 00 00 00 00 00 20 4e 00 00 80 a8 12 01 0f 2
7 00 00 d2 7a 5b 21 00 00 00 00 00 00 00 00 [body 92 byte(s)].
U_GNSS_POS: UTC time = 1685712778.
U_GNSS: sent command b5 62 01 07 00 00 08 19.
```

Figure 32 - Réponses du module GPS lors de son verrouillage

```
5712781000,"location":{"lat":46.2543134,"long":7.4228291}}}
Publishing message to topic iot/trackedAssets
Message: {"payload":{"deviceId":"thing123","timestamp":1685712781000
,"location":{"lat":46.2543134,"long":7.4228291}}}
AT+UMQTTER

+UMQTTER: 13,0

OK
AT+UMQTTC=9,1,0,"iot/trackedAssets",108

>{"payload":{"deviceId":"thing123","timestamp":1685712781000,"locati
on":{"lat":46.2543134,"long":7.4228291}}}
OK

+UUMQTTC: 9,1
AT

OK
retval: 0Device is moving, so 30 seconds sending
```

Figure 33 - Envoi de la position GPS du device

12. Conclusion

Durant cette étude, un démonstrateur permettant le suivi d'actif a été créé à partir d'un kit de développement nommé XPLR-IOT1. Celui-ci à la possibilité de pouvoir s'auto provisionner grâce au provisionnement juste à temps fournit par AWS IoT Core. Le démonstrateur peut envoyer sa position, récupérée grâce à un module GPS externe, au broker MQTT de AWS IoT Core via une connexion 4G LTE-M. Le device peut aussi recevoir l'ordre de télécharger un fichier de la part de l'outil AWS IoT Jobs pour simuler une mise à jour à distance. Le fichier téléchargé pour ce démonstrateur contient la couleur d'une LED à allumer. Une esquisse d'architecture de reprise après désastre a été implémentée permettant le transfert des certificats des appareils de la région primaire à la région secondaire. Un serveur web, basé sur l'outil Amplifly a été déployé sur AWS permettant de visualiser la trace de l'appareil en mouvement. L'appareil est capable aussi de déterminer s'il est en mouvement ou non-grâce à un accéléromètre.

Le cahier des charges a donc été pleinement rempli et tous les domaines ont pu être étudiés. Malheureusement, un manque de temps ne m'a pas permis de terminer l'implémentation totale de la partie reprise après désastre.

Sion, 2 juin 2023,

Jean Nanchen

13. Travaux cités

[1] [1] [2] Géoride. [En ligne]. Available: https://georide.fr/.

[3] [2] [4] M. Bosson, «NB-IoT vs LTE-M: A comparison of the two IoT technologies,» onomondo, [En ligne]. Available: https://onomondo.com/blog/nb-iot-vs-lte-m-a-comparison-of-the-two-iot-technology-standards/#:~:text=NB%2DIoT%20can%20be%20deployed,to%20be%20available%20on%205G..

[5] [3] [6] Wikipedia, «3rd Generation Partnership Project,» [En ligne]. Available: https://fr.wikipedia.org/wiki/3rd_Generation_Partnership_Project.

[7] [4] [8] «Zephyr (système d'exploitation),» 18 05 2023. [En ligne]. Available: https://fr.wikipedia.org/wiki/Zephyr_(syst%C3%A8me_d%27exploitation).

[9] [5] [10] u-blox, «ubxlib,» [En ligne]. Available: https://github.com/u-blox/ubxlib. [Accès le 18 05 2023].

[11] [6] [12] AWS, «What is the AWS CDK?,» [En ligne]. Available: https://docs.aws.amazon.com/cdk/v2/guide/home.html. [Accès le 19 05 2023].

[13] [7] [14] AWS, «Provisioning devices that have device certificates,» [En ligne]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/provision-w-cert.html.

[15] [8] [16] AWS, «Provisioning devices that don't have device certificates using fleet provisioning,» [En ligne]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/provision-wo-cert.html.

[17] [9] [18] AWS, «Comment puis-je utiliser JITP avec AWS IoT Core ?,» [En ligne]. Available: https://repost.aws/fr/knowledge-center/aws-iot-core-jitp-setup.

[19] [10] [20] AWS, «Location Service Examples,» [En ligne]. Available: https://github.com/aws-samples/amazon-location-samples/tree/main/maplibre-js-react-iot-asset-tracking.

[21] [11] [22] AWS, «Disaster Recovery of Workloads on AWS | AWS Events,» [En ligne]. Available: https://www.youtube.com/watch?v=cJZw5mrxryA.

[23] [12] [24] AWS, «Calculateur de tarification AWS,» [En ligne]. Available: https://calculator.aws/.

14. Annexes

Annexe I

https://github.com/73jn/pa-23