INSAT

Mobile Devices Ingénierie

Rapport de stage d'ingénieur

Mise en place d'un service de communication au standard MQTT hébergé au sein d'un environnement kubernetes

---

Génie Logiciel

- Ihsen Charfi
- Diplôme préparé : Ingénieur en Génie Logiciel
- Encadrant de stage en entreprise : Jeremy HEATER
- Responsable de stage à l'INSAT : Riadh ROBENNA
- Durée du stage : Du 13 mars 2018 au 31 août 2018
- Date de remise du rapport : Le 31 août 2018

Table des matières

Table des figures

Liste des tableaux

1 Introduction Générale

Paragraphe sur la télématique des voitures , la voiture connectée, la tendance des clouds et systèmes big data ...

La télématique est née de la convergence des progrès de l'informatique embarquée et des télécommunications. Dans le domaine des transports, elle désigne les dispositifs permettant la production, l'émission, la réception, le traitement et la représentation des données de manière automatisées.

2 Présentation du contexte du projet

2.1 Mobile Devices Ingénierie

2.1.1 Qui est MDI ?

Mobile Devices Ingénierie[1], un des leaders mondiaux de la télématique, est une entreprise française spécialisée dans les systèmes embarqués pour l'automobile. L'entreprise dont le siège est situé en région parisienne (à Villejuif) a depuis à peu près 15 ans, conçu et développé des boîtiers connectés aux véhicules par port OBD - On Board Diagnostics.
Actuellement, MDI comprend une cinquantaine d'employés répartis dans plusieurs pays dont la France, les États-Unis Irlande. Avec un chiffre d'affaire de plusieurs dizaines de millions d'euros et plus d'un million de boîtiers déployés dans le monde.

((((((((((((((((((((((((((((figure : carte du monde montrant les pays où se trouvent MdI (fr , USA , AS) et les pays ou les OBD sont déployés (les clients : (l'Afrique du sud,Autriche, Belgique, Bulgarie, Croatie, Chyprés, République Tchèque, Danemark, Estonie, ETats unies, Finlande, France, Allemagne, Grèce, Hongrie, Ireland, Italie, Lettonie, Lituanie, Luxembourg, Malte, Pays bas, Pologne, Portugal, Romanie, Slovaquie, Slovénie, Espagne, Suède, Suisse, Grande Bretagne))))))))))))))))))))))))))))))))

Fig.1: Les pays où circulent des véhicules ayant des OBD déployés

Les clients de Mobile Devices Ingénierie sont ainsi réparties partout dans l'Europe et les États Unies. En effet, ils occupent différents secteurs d'activités tels que :

- Des applications embarquées de gestion de flottes de taxis
- Des compagnies d'assurances
- Des compagnies de navigation et d'info trafic
- Des revendeurs qui adaptent les produits MDI à leurs services et qui disposent d'un accès à certains marchés de par leur histoire ou leur image de marque.

2.1.2 Domaine d'activité et Stratégie

L'objectif de l'entreprise est de devenir le leader mondial pour l'acquisition, le traitement, l'enrichissement et l'échange des données techniques de véhicules connectés.

Mobile Devices Ingénierie a pour secteur d'activité la télématique embarquée pour l'automobile. Concrètement, cela consiste à proposer des solutions techniques permettant l'échange d'informations entre un ou plusieurs systèmes de gestion centralisés et une flotte de véhicule qui y est rattachée et connectée en temps réel. Ces informations sont ensuite récupérées en temps réel et peuvent être envoyées vers les serveurs de l'entreprise ou des clients.

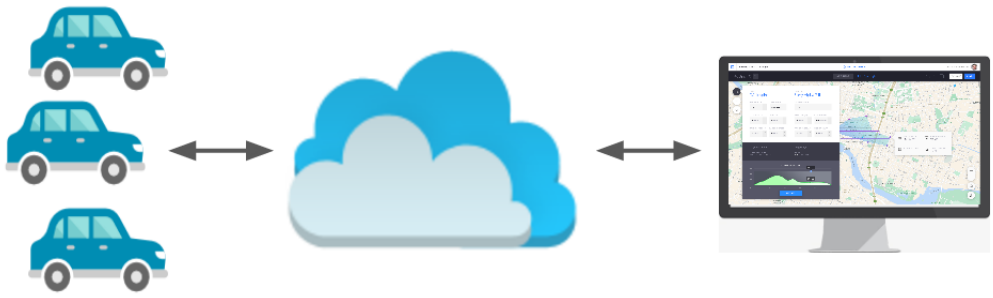

Fig.2: Schema représentant le circuit des données remontés

L'activité principale de Mobile Devices Ingénierie se résume à la mise en place de solutions techniques afin d'effectuer le suivi de véhicules et la gestion de flottes. Les produits de l'entreprise permettent de récupérer des données telles que la vitesse, la consommation, le type de conduite et même de prévenir des accidents. Ces mêmes données permettent ensuite, par exemple pour certaines entreprises, d'étudier la conduite des véhicules et leurs différents paramètres comme suivre un déplacement et/ou quantifier la consommation dans

le cas d'une gestion de flotte.
Ces données peuvent aussi servir à étudier le type de conduite dans le cas d'une compagnie d'assurance. La société compte aussi d'autres activités comme la commercialisation des boîtiers pouvant offrir l'aide à la navigation servant par exemple aux Taxis. Les entreprises clientes peuvent aussi reprogrammer une partie de leurs boîtiers afin de récupérer des données spécifiques et les transférer vers des serveurs distants.

Par la nature même de l'activité, l'entreprise apporte à ses clients des solutions innovantes permettant de concilier le développement économique et la préservation de l'environnement. En effet, ses solutions permettent de mettre en œuvre le contrôle de baisse de la consommation énergétique, de l'empreinte carbone des véhicules et du risque environnemental associé.

Dans le cadre de son développement durable, la protection de l'environnement est une préoccupation fondamentale pour MDI. L'éthique, l'équité et la diversité sont des facteurs de progrès qui contribuent à améliorer les résultats économiques ainsi que le climat social, éléments clefs d'un développement durable pour Mobile Devices Ingénierie.

2.1.3 Organisation et produits

L'entreprise suit un schéma d'organisation hiérarchique fonctionnelle composée d'une division classique du travail par fonctions. Elle est constituée d'un dirigeant et ensuite d'une équipe de collaborateurs ayant différentes fonctions : commerciales, comptables, financières, ressources humaines, productions, etc. Quant au côté technique, MDI articule plusieurs types d'équipes :

- L'équipe **Production** en charge de la production et la livraison des produits
- L'équipe **Core** en charge de « Morpheus » (le système d'exploitation applicatif du boîtier)
- L'équipe **Système** en charge du système d'exploitation linux
- L'équipe **Pre-Sales** en charge du support clients et accompagement des commerciaux
- L'équipe **Serveur** en charge du cloud, au sein de laquelle j'ai effectué mon stage

Ces équipes travaillent sur différents projets autour des produits hardwares et softwares comme le montre le tableau 1 de la page 5

Les produits Hardwares	Les produits Softwares
• munic.io • municMax • OBD Dongle	• Morpheus OS • CloudConnect • CloudNext

TABLE 1: Tableau des produits MDI

Les produits Hardwares sont intégrés dans les véhicules et récoltent des informations sur la vitesse, la position, le niveau de carburant, la pression des pneus, etc.. à un instant t donnée. En général nous appelons ces produits hardwares "les boîtiers" qui doivent être intégrés dans les véhicules et déployés dans le système cloud.

Quant aux produits Softwares, il y a le Morpheus OS qui présente le système d'exploitations des boîtiers. D'autre part,les clouds CloudConnect et CloudNExt qui offrent un ensemble de services aux clients et qui sont développés et maintenus par l'équipe Serveur. On détaillera plus sur ces deux produits dans le prochain chapitre.

2.2 Présentation du projet

2.2.1 Contexte du projet

Mon sujet principal de stage est la conception et développement du nouveau composant du Cloud. Ce composant représente le point d'entrée des données au cloud. Il assure la communication entre le cloud et les boîtiers. Le composant actuel porte le nom de BS - acronyme de Binary Server. Il est intégré seulement dans CloudConnect.

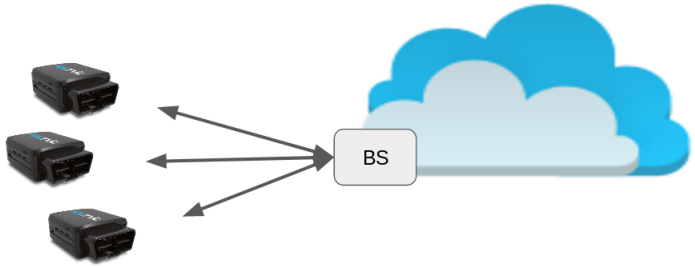

Fig.3: BS le seul point d'entrée des clouds

2.2.2 Problématique

Plusieurs facteurs rendent le développement d'un noouevau composant cloud une nécessité.
Dans l'architecture actuelle, les deux cloud sont totalement indépendants mais ils reçoivent les données de la même source BS malgré la différence entre les deux formats de messages.

D'autre part , pour une raison commercial, le nouveau protocole migre d'un protocole fait maison vers un protocole standardisé.
Un standard permet de bénéficier des librairies existantes sur le marché et donc de gagner en temps d'intégration puisque il' n y aura pas le besoin de faire du développement à zéro. MQTT n'étant pas propriétaire, les partenaires ou clients savent également comment celui-ci est conçu. Ce qui n'est pas le cas pour MD21 le protocole actuel fait maison donc cela permet de rassurer le client. Le "standard MQTT" permets également de laisser d'autres boîtiers se connecter à notre plate-forme plus facilement plutôt que d'implémenter notre protocole propriétaire dont ils n'ont même pas les sources.
Donc la possibilité de gagner la confiance de plus de clients et de s'ouvrir à des marchés potentiels.

2.2.3 Objectifs

Suite à cette problématique MDI a mis en place le projet MD30 pour mettre en place la solution. MD30 sera le nouveau protocole de communication OBD-cloud qui a comme objectifs :

- concevoir et réaliser un nouveau protocole de communication entre boîtier et cloud
- mettre en place un nouveau composant BS
- faire une étude de faisabilité et de performance de l'utilisation du protocole MQTT comme protocole de communication.
- Le support du format de données CloudNext par le nouveau BS

2.3 Méthodologies de travail

2.3.1 Cycle itératif

Le mode itératif est une méthodologie de développement différente des méthodes classiques(modèle en cascade, modèle en V). Elle tente de formaliser une méthode plus pragmatique et maniable que ces derniers.

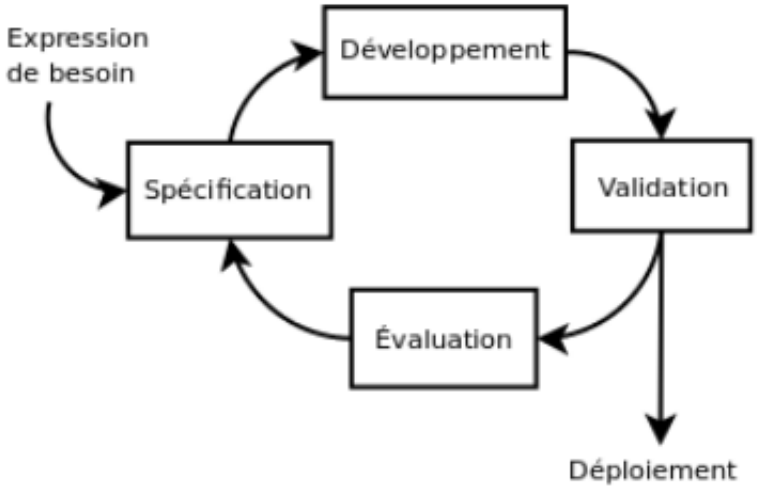

Fig.4: Le processus du cycle itératif

Cette méthode présente 6 étapes marquante :

- L'expression de besoin : où les exigences sont mis en place. L'idée reste que les informations en entrée peuvent être modifiées par la suite du processus itératif.
  Le cercle du processus itératif :
  — La spécification technique du besoin
  — Le développement : la période où on implémente la spec
  — La validation : une étape qui est marqué par les tests.C'est l'ensemble des tests qui permettent de s'assurer que le développement effectué correspond bien à ce qui était attendu.
  — L'évaluation : Cette étape sert à effectuer un retour sur les fonctionnalités laissés. Ceci sert comme des informations d'entrée pour un nouveau cycle.
- Déploiement : l'étape finale qui consiste à ce que les livrables validés sont livrés.

Ce type de cycle de développement est le plus souple des méthodologies : chaque itération permet de s'adapter à ce qui a été appris dans les itérations précédentes et le projet fini peut varier du besoin qui a été exprimé à l'origine.[2]

Vu la nature du projet qui part plutôt vers une étude de faisabilité avec des critères de performances, l'expression des besoins peuvent être redéfinies pendant ce temps. Il est préférable de suivre une méthodologie souple d'où le choix de telle méthodologie.

2.3.2 MD30 est un travail d'équipe suivi

le projet MD30 présente le nouveau protocole de communication entre les boîtiers OBD et le CLoud et donc le projet invoque les boîtiers ainsi que le point d entrée du cloud.

Le projet MD30 engage 5 personnes dans deux différentes parties :

- 2 personnes sur MD30 côté Serveur
- 3personnes sur MD30 côté Core

Même si le coeur de développement des deux parties est totalement indépendant l'un de l'autre. Les deux parties doivent rester synchroniser sur l'avancement de chaque partie afin d'avancer dans le projet en phase. C'est pour cela qu'une réunion hebdomadaire de synchronisation s'effectue chaque fin de semaine. Le suivi de ce projet se fait à travers la création des ticket en utilisant l'outil Asana.

Asana est un outil de gestion de projet web et mobile qui donne une visibilité sur .....
figure : What asana dashboard looks like

3 Étude de l'existant

Cette partie concerne le plus sur l'architecture globale des serveurs clouds de MDI, leurs évolutions au cours du temps ce qui éclaircira le besoin de la création du nouveau composant point d entrée.

3.1 Évolution des clouds MDI

Les boîtiers OBD ont la particularité de s'intégrer dans tout type de véhicule et permettent l'émission, la réception, le traitement et la représentation de données diverses comme l'état de la batterie du véhicule, les données standards, les données du propriétaire du véhicule ou des données des capteurs du boîtier même (localisation GPS, l'accéléromètre, le gyroscope, la consommation de la batterie ...)et un tas d'autres informations complémentaires. Ces données là ont besoin d'être traité pour pouvoir en sortir de l'information utile, d'où le besoin de services divers, regroupés dans un cloud.

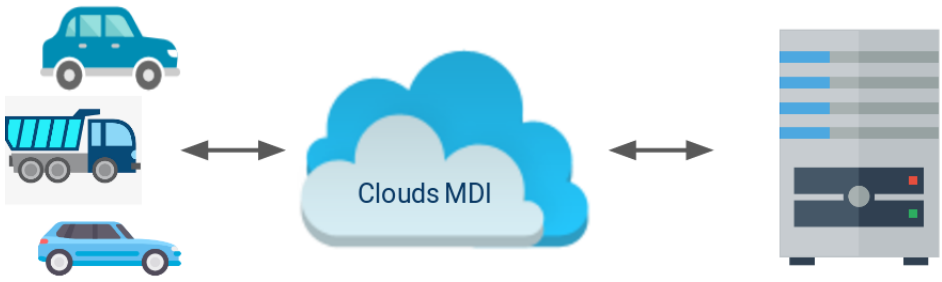

Fig.5: Communication avec le cloud

Le Cloud collecte les informations des véhicules récoltés par ces produits intégrés, traite et stocke ces derniers et communique avec les serveurs clients externes.

3.1.1 Le premier Cloud de MDI

Le CloudConnect - ou CC - est le premier cloud fait maison de MDI qui englobe un ensemble de services pour les clients. Créé en ..et hébergés dans des serveurs d'OVH, CC traite jusqu'à présent un nombre important du traitement des données. CloudConnect propose également un écosystème d'applications partenaires (gestion de flottes, ecodriving. . .) mais aussi des services connectés par l'intermédiaire de partenaires.

L'architecture globale de ce cloud est présenté par le schema de la figure 6 de la page 10.

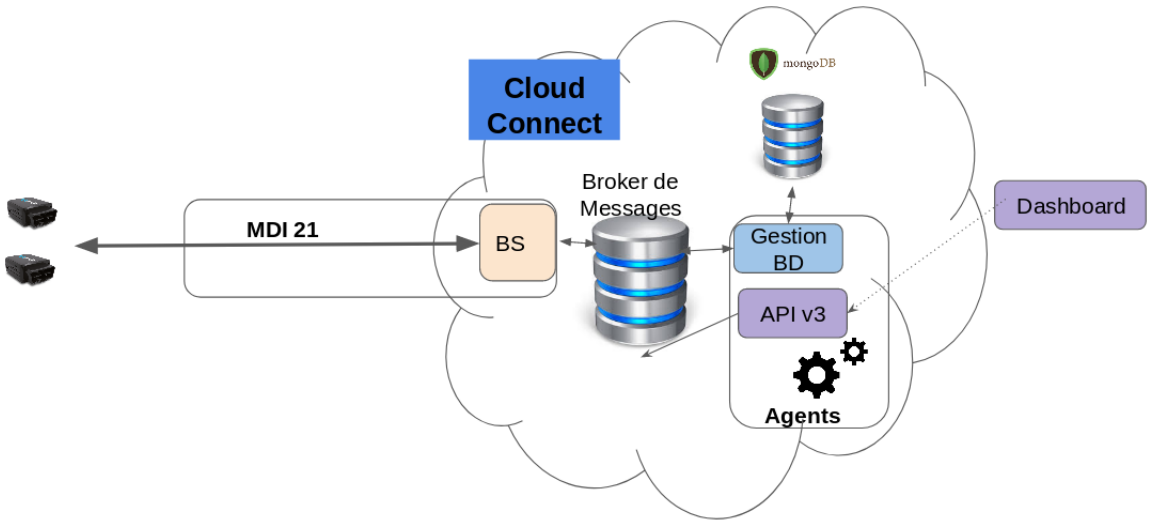

Fig.6: L'architecture de CloudConnect - CC

Pour des raisons de confidentialité ce schema a été simplifié. Néanmoins, il présente les principaux composants de CloudConnect. Parmi ces coposants on trouve :

- un **BS** qui est le composant d'échange de messages entre le cloud et les OBD. Il présente le point d'entrée du cloud et représente le composant en question. Il supporte MD21, le protocole de l'entreprise conçu pour la communication entre boîtiers et Cloud. Nous détaillerons bien ce composant à la fin de ce chapitre.
- un **broker de messge :** pour les systèmes Big Data. C'est un système distribué très répandu dans le domaine de BigData centralisant le flux de données provenant des différents systèmes de l'entreprise tout en gardant le découplage entre les systèmes producteurs de ceux consommateurs de données.
  Le point fort de cet outil est sa capacité de supporter des débits très importants que ce soit pour la publication ou la lecture des données. D'autres part, ce broker permet la consommation des données en temps réel et en mode batch, en persistant les données. Il est à noter que les données ne sont pas stockés dans ce système, ils sont là jusqu'à leurs consomattion. D'où le besoin d'une base de données.
- une **base de données NoSQL** orienté documents pour le stockage des données jusqu'à 3 mois.
- un ensemble de services qui sont accéder à partir de **API-V3** :

- un **dashboard** pour l'affichage des données et l acces visuel des services au clients

Il est bien de noter que l'équipe serveur de MDI veille à ce que sa politique de récupération des données respecte la nouvelle réglementation de GDPR[3].

3.1.2 De CC à CN

Bien que le CloudConnect paraît complet. Il présente quelques problématiques au niveau de :

- **Développement des agents :** Un Framework interne est utilisé pour simplifier le développement des différents agents en langage ruby. Ce framework fournit un ensemble de fonctions qui permettent, entre autres, d'accéder au courtier de messages en lecture et en écriture. Concernant les agents à état, le framework ne fournit pas de mécanismes pour simplifier la gestion de leurs états. La gestion de l'état est donc laissée aux développeurs.

- **Déploiement des agents :** les agents sont exécutés de manière parallèle. En d'autres termes, un agent peut avoir plusieurs instances déployées dans différentes machines. Les différentes instances d'un même agent partagent le traitement du flux des événements mais ne communiquent pas entre eux. Le déploiement ainsi que la maintenance d'un agent se traduisent par une mise-à-jour des fichiers de configuration des machines exécutant l'agent via le logiciel Chef.

Comme solution à ces problèmes, MDI a eu recours à la création d'un autre Cloud afin de revoir l'architecture logicielle et technique de CloudConnect afin de simplifier le développement des agents, leurs déploiements ainsi que leurs maintenances et de permettre à des développeurs externes de l'entreprise de coder leurs propres agents.
L'entreprise a créé alors sa propre infrastructure adaptée aux données de la télématique appelée « CloudNext » - ou CN dont l'objectif est de se diriger vers une infrastructure orientée containers qui rend plus facile l'intégration de solutions développées par les clients de l'entreprise et permet de diversifier ses activités.

L'architecture simplifié du CN est présenté par la figure 7 dans la page 12.

Dans le nouveau cloud on trouve d'autres composants nouveaux. Parmi eux on trouve :

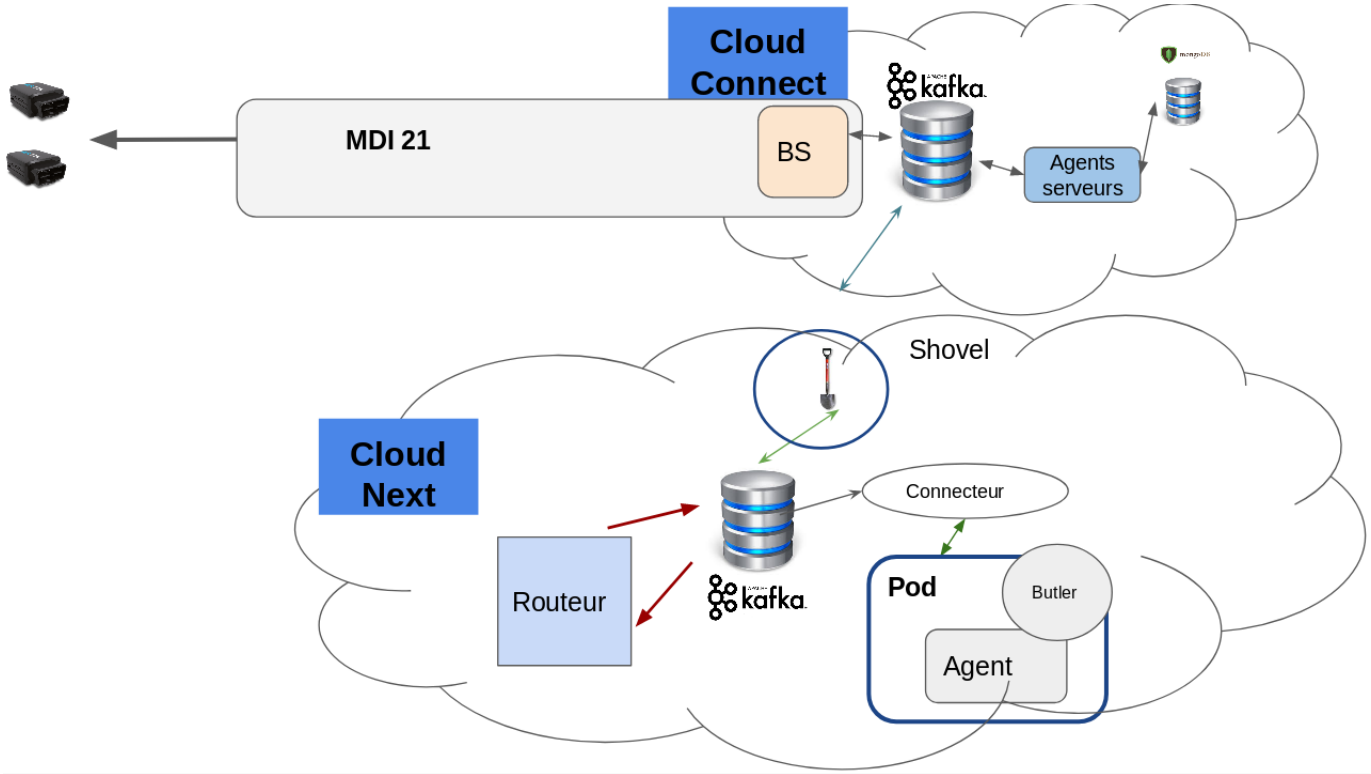

Fig.7: L'architecture du cloud avec CloudNext

- **Shovel :** Il traduit les informations du format "MessagePack" au format "Protobuf".
- **Routeur :** Il assigne chaque track à un agent.
- **Connecteur :** Il gère les états des stateful agents
- **Pod :** Les pods de Kubernetes
- **Butler :**
- **Agent :** Un service Cloud
- docker custom registery

Pour faciliter la communication entr les agents, CC migre vers un format de message de MsgPack vers ProtoBuffer

**********Expliquer différence entre MsgPack et Protocole Buffer *******

3.2 le BS MD21 de plus pres

Redis , Indigen ..., fonctionnalités

le BS est le service de communication. Il présente le point d'entrée des boîtiers au Cloud-Connect. Multiple instances sont lancés sur des serveurs host différents pour assurer la balance de la charge et éviter les *Single Point Of Failure*.
Cet agent est codé en Erlang, c'est un problème pour la garantie de la maintenance future du composant. Erlang perds de son intérêt pour 2 raisons : L'élasticité de la VM n'est plus utile avec Docker et Kubernetes Le chargement à chaud des nouvelles versions et la coexistence avec les anciennes version est aussi disponible avec Docker et Kubernetes

Le BS ecrit dans RQueue et le Shovel copie de RQueue vers Kafka. donc il n'écrit pas directement dans le broker de message de CloudNext.

Les informations de session sont dans redis (socket permanente). Les informations statiques (fields, channels, paramètres des accounts) sont récupérés depuis les apis (web services) - c'est une action rare. De plus les changements en DB sont notifiés au BS via Kafka.

3.2.1 Format de données

Les informations remontées par les boîtiers sont considérées comme des évènements. Ces événement peuvent contenir des tracks , des messages et/ou des présences.

- **Track :** informations géolocalisées envoyés depuis le boîtier au Cloud et qui sont les données remontés des capteurs traités par les boîtiers.

- **Message :** informations plus complété qui se dirige vers un channel précis, invoqué depuis le serveur vers le boîtier.

- **Présence :** état du boîtier, si il est connecté au serveur ou pas.

3.2.2 Transmission boitier-serveur

La transmission se fait en utilisant le protocole MD21. MD21 est une logique in-house de séquence ainsi qu'un encodage décrit en ASN.1
Ce protocole fonctionne en TCP et en UDP. L'UDP n'est que rarement utilisé étant donné

que les opérateurs téléphonique ne le favorise pas, voire rendent son utilisation impossible correctement. L'utilisation d'UDP nécessite en parallèle un lien TCP.
Le lien est à l'initiative du boîtier,c'est à dire que le boîtier qui doit se connecter en premier au serveur en passant par le BS La sécurité du lien est demandée par le boîtier au serveur et supporte les modèles récents (TLS 1.2).
Une limitation actuelle, c'est que les transaction ne sont pas identifiées et que le serveur ne peut donc pas indiquer au boîtier un index unique du dernier track traite. De ce fait, le boîtier re-envoi tous les track qui n'ont pas été reconnu et on a des rejeu de données.

4 itération 1 : nouveau BS au standard MQTT

MDI, étant une entreprise innovante en télématique, doit toujours suivre et avancer les nouvelles technologies. Une de ces technologies est le protocole de communication standard des objets connectés : le MQTT.

...

4.1 Descriptif du protocole :Le standard MQTT, l'essentiel à savoir

D'après la spécification officielle MQTT [4], le standard MQTT est un protocole de transport de message Client/serveur sous le pattern de Publication/Souscription " Publish/ Subscribe". C'est un protocole léger , ouvert, simple et désigné d'être facile à implémenter. Ces caractéristiques le rend idéal à être utilisé dans plusieurs situations, surtout pour la communication Machine to MMachine M2M et pour l'internet des objets IoT. Des contextes où la faible empreinte du code est requise ainsi qu'une bonne bande passante.

Qu'offre MQTT ?

- L'utilisation du pattern de message " Publication/Souscription" qui offre des distributions un-à-plusieurs (one-to-many) et le découplage de ces deux parties.
- 3 Qualités de service QoS pour la distribution des messages .
- Un échange de protocole minimisé pour réduire le trafic du réseau
- Un mécanisme pour notifier les parties intéressées lors d'une déconnexion anormal

En cas de besoin de plus de détails sur le standard, consultez la partie Annexes. Elle comporte des éclaircissements sur le standard MQTT.

En quoi MQTT sera meilleur ?

Pour répondre à une telle question il faut comparer entre MQTT et le protocole de MD21 comme le montre le tableau 2 de la page 16.

Cette compariaosn montre les différences entre les deux protocoles en mettant en valeur les points considérés forts de MQTT par rapport à MD21. Ces points sont autour la logique du protocole adapté, leurs protocoles de base et le dégrée de couplage avec les consommateurs ou la partie traitement dans notre cas.

	MQTT	MD21
Logique	standardisée	fait maison
Protocole basé sur	TCP/IP	TCP - UDP
Couplage entre BS et consommateurs	Broker découplé des consommateurs	couplage fort

Table 2: Tableau de comparaison : MQTT vs MD21

4.2 Spécifications techniques & Implémentation

Le Point d'entrée du cloud ou BS se charge de deux principales fonctionnalités.

- **Transfert de données :** où ils gèrent les connexions avec les boîtiers.
- **Traitement de données :** encodage/décodage des données de messages selon le format de données du cloud.

Puisque le standard repose sur le découplage entre le publisher et subscriber, la conception doit prendre en compte un broker MQTT pour assurer cette notion. La conception est présenté par la figure suivante ..

********* Insertion d une figure de la conception **********

Le BS aura deux parties importantes qui sont le transfert des données et l'encodage pour le broker.

4.2.1 Recherche & état de l'art

Il existe quelque large implémentation de MQTT comme Facebook Messenger par example, mais aussi de nombreux outils monétisés et d'autres open source. Il y a aussi un projet Eclipse active, "Paho", qui offre une implémentation scalable , open source pour différents langage de programmation comme Java , C, C++ , Python , JavaScript , C# et Go lang.

Il existe plusieurs implémentations qui sont réunis dans cette référence [5] publié par un des membbre de la communauté de MQTT et qui les comparent selon plusieurs critères. Pour effectuer un bon choix satisfaisant de toute ce s propositions, il faut bien demander les critères de l'entreprise en premier lieu sur lesquels on se base.

A la recherche d'un broker MQTT !

Comme la stratégie de l entreprise est de réduire les coûts, nous éliminons les choix d'outils payants.

D'autre part, le support de QoS 2 n'est pas exigé vue que l'échange de message en QoS 2 est trés couteux. Tandis que le QoS 1 est indispensable pour profiter du système des ack.
Un autre critère est la charge des connexions du broker. L'un des objectifs de MD30 est d'augmenter le support en charge du nouveau BS ce qui revient dans ce cas à étudier la charge du broker MQTT en question. Ceci peut être effectué de deux manières :

— Augmenter le support en charge du Broker par rapport au BS de MD21, sachant que le support du BS actuel est de ...
— Assurer la scalabilité du broker ce qui revient à choisir dés le début un broker scalable.

En se basant sur ces critères, nous éliminons déjà beaucoup de choix pour rester avec les implémentations des projets open source qui ont implémenté la QoS 1 ainsi qu'une solution pour la sclalabilité du broker. En plus il faut que le projet soit mis à jour avec une communauté qui le supporte. Nous nous retrouvons alors avec 2 choix qui répondent à nos critères :

— HmQ : Broker MQTT open source développé en Go lang.
— Mosca : Broker MQTT open source déeloppé en Node.JS.

Afin de déterminer lequel de ces brokers sera choisi il faut bien vérifier qu il répondent aux attentes en effectuons quelques tests. Pour ce faire nous avons intégrés les brokers dans la chaine de transfert des données jusqu' à lentrée dans le cloud - c'est à dire le push dans kafka.

4.2.2 Développement effectué

L'intégration du broker dans la chiane de push qui est composé par :
. Traitement de données : le(s) subscriber(s) qui lie le Track et lance les processus d'encodage.
le(s) publishers qui envoie le Message et publie au broker
Dans cette partie , il faut mentionner le dilemme des topics et leurs créations pour savoir le nombre des sub / pub .. et le problème de clustering qu l engendre . => le couplage lâche n'est pas toujours la bonne solution surtout dans notre cas : les instances doivent être dépendantes .

. L'échange avec Kafka : Les clients kafka : = un producteur kafka et consommateur
il faut tester les différents particularités du standard MQTT.

— le wildcard :
— le support de lastwill :
— le support de QoS 1

— le support

Tous ceci ont été vérifié en résumant dans le tableau suivant :

4.3 Tests & Performance

Virtualisation avec Docker

À l'aide de conteneurs, on trouve tout ce qui est nécessaire pour faire exécuter un logiciel est emballé dans des conteneurs isolés. Contrairement aux machines virtuelles, les conteneurs ne regroupent pas un système d'exploitation complet : seules les bibliothèques et les paramètres requis pour que le logiciel fonctionne sont nécessaires.
Cela permet d'avoir des systèmes autonomes, légers et garantit que les logiciels fonctionneront toujours de la même manière, quel que soit l'endroit où ils sont déployés.

Docker Docker est une technologie, un "runtime" pour les conteneurs. Il est aussi une plate-forme pour le développement, l'expédition et l'exécution d'applications. Il permet de séparer les applications de l'infrastructure afin de livrer rapidement un logiciel.

Avec Docker, on gère l'infrastructure de la même façon qu'on gère les applications. En profitant des méthodologies de Docker pour l'expédition, le test et le déploiement rapide du code, on peut réduire considérablement le délai entre l'écriture du code et l'exécuter en production.

Comment docker construit ces conteneurs isolés ?

— **Isolation du système de fichiers** : chaque conteneur s'exécute dans un système de fichiers racine complètement distinct.
— **Isolation des ressources** : les ressources système comme CPU et mémoire peuvent être attribuées différemment à chaque conteneur. Ceci présente la principale motivation pour tester les performances de chaque composant.
— **Isolation de réseau** : chaque conteneur de processus s'exécute dans son propre espace de noms de réseau, avec une interface virtuelle et une adresse IP propre.

la figure 8 montre l'optimisation en ressources qu'offre la virtualisation avec la technologie des conteneurs par rapport à la technologie de machine virtuelle et hyperviseur.

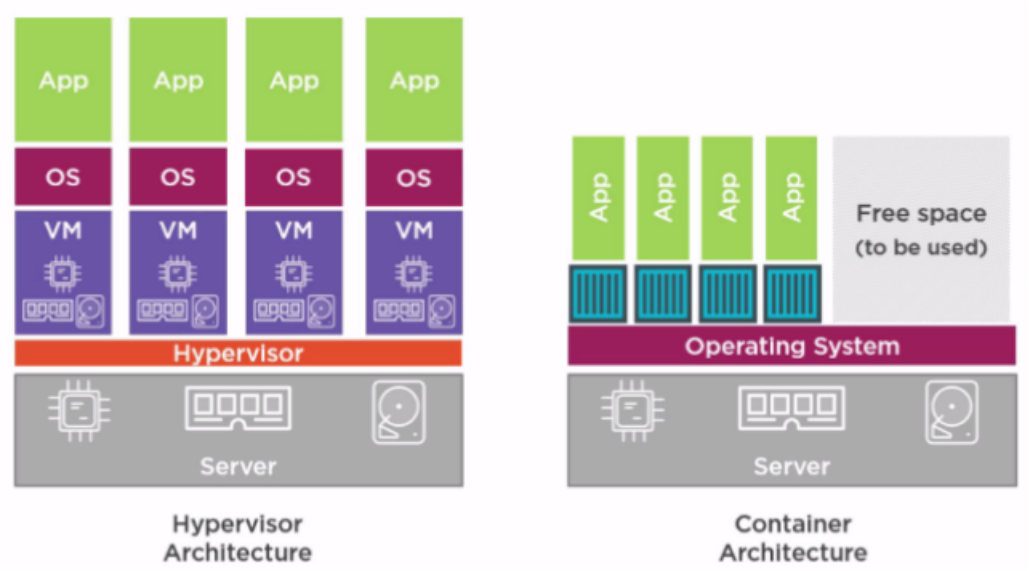

Fig.8: L'optimisation de docker par rapport à la machine

4.3.1 Tests

L'infrastructure est de devops .. plusieurs clusters local / DEV / inté / prod , pour nous on s intéresse aux deux premières étapes.

Le but de ces tests et de tester la performance du cluster et son support de charge.

1er déploiement de test : le clustering avec HAProxy :
Communication entre les conteneurs en local avec minikube. HAproxy est un load balancer pour les conteneurs docker Monitoring avec haproxy : outil intégré de visualiser le nbr de cnx , les instances ups **2eme déploiement de test : le clustering avec Kubernetes**
Toutes les services et composants de CLoudNExt sont orchestrés par k8s. Kubernetes un orchestrateur de conteneurs qui ... un cluster de dev sur des machines réels hébergés à OVH. l infrastructure est développé et maintenu par l'équipe serveur. On trouve les composants de base toujours déployé et pour les nouveaux agents on les intégre en passant par helm.

Le but de ces tests et de tester la performance du cluster et son support de charge.
********** les graphes de cnx **********

Nombre de cnx par broker , par

********** les graphes de percentile **********
K8s traite la résilience

Le monitoring se fait par InfluxDB/Grafana. Les composants remontent des métriques influxdb qui sont visibles par Grafana.

4.3.2 Evaluation

Après avoir fini le développement et les tests de la première itération. On évalue le système par rapport à nos exigences globaux du système. Néanmoins lors de cette évaluation on est face à plusieurs problématiques qu il faut prendre en considération.

- Le coût de communication sera élevé, car ça sera des paquets tcp dans des paquets MQTT
- L'ajout du maintien d'un nouveau broker avec sa BD : ceci ajoute de la complexité à gérer.
- La manière de consommer depuis le broker :
  - **un seul consommateur global :**
    Avantages : Lire tout du broker, Écrire tous au clients sans avoir besoin de partager la consommation sur plusieurs consommateurs. D'un autre coté, avec une telle manière de consommation on assure que tous les messages sont toujours dans le broker jusqu'à leurs consommation lorsque la persistance est assuré par le broker MQTT
    Inconvénients : Problème de *Single Point Of Failure* car si le consommateur tombe il y aura un blocage sur toute la chaîne. Cette résilience peut être traité par Kubernetes, mais elle entraîne une latence considérable si elle est récurrente.
  - **plusieurs consomatteurs connectés selon une hiérarchie de topics :**
    Avantages : Théoriquement c'est la solution standard du protocole MQTT d'avoir plusieurs consommateurs par broker et donc la plus adéquate à utiliser.
    Inconvénients : Pour les clients il y a pas d'hiérarchie clair pour les clients. D'autre part, il y a une différence importante de nombre d asset par client chez MD21 la diversité des clients de MDI fait que il y a des individus qui ne possèdent que qlq boîtiers en revanche d'autres clients comme constructeurs d automobile possèdent des milliers. De point de vue scalabilité cela fera un problème vue que le nombre de consommateurs augmentera selon les clients. Ou bien au contraire qu'un seul client augmente sa commande que finalement un seul thread consommateur n'accepte plus de connexions.
  - **plusieurs consomatteurs connectés aux instances, un consommateur par instance de broker** :

Avantages : un des avantages

Inconvénients : C'est le couplage fort entre le broker et son consommateurs et donc c 'est contre.

Différentes manières pour effectuer la consommation :

Pour chaque manière on étudie les avantages et les inconvénients ...

Synthèse : Cette première itération nous a prouvé la limite de MQTT pour le projet MD30. Elle sert comme une étude sur la faisabilité d'intégration du standard dans le projet. La question qui se pose est , quelles sont les fonctionnalités MQTT que MD30 exige par rapport aux diverses fonctionnalités offertes par MQTT ? $=>$ ils sont peu : Ce que MD30 exige de tout cela n'est que le mécanisme des acks qui sont traités par les headers des messages MQTT. Donc ajouter des bytes de plus comme étant un header dans le nouveau format de message du boîtier permet ce mécanisme. $=>$ une implémentation fait-maison de ça fera l'affaire.

Conclusion

Après avoir fait une évaluation de la première itération et faire l étude de coût et performances de MQTT, nous nous rendons compte que ce dernier est tellement riche en exigences divergente des exigences de MD30 et donc l'adopter tel qu il comme le protocole de communication de MDI n'était pas la bonne solution.

5 Itération2 : nouveau BS englobant TCP serveur

5.1 Spec

Cette partie repose sur la modification des exigences et la nouvelle Conception

5.1.1 L'évolution des exigences

Après avoir fait l'évaluation des résultats de performance de la 1ere itération du projet, on a redéfinit les spécifications du projet. Le tableau suivant montre la différence des exigences entre les deux itérations :

Les changements des exigences			
	Itération1	Itération2	Avantage de cette évolution
Protocole de communication	MQTT	TCP avec le support de MQTT	Moins coûteux par rapport aux paquets échangés, tout en gardant la possibilité de connecter le boîtier à un serveur MQTT
Architecture du BS	Un cluster de brokers MQTT avec des subscribers des tracks et publishers d'events.	Un cluster de TCP serveur qui gèrent les connexions, traitent les message échangés et envoient/reçivent les msg à/de kafka.	Moins coûteux par rapport aux instances à gérer. Plus facile à gérer un seul composant au BS. Pas de complexité à dispatcher les subscribers quand on monte de charge.

Table 3: Tableau de comparaison des exigences du projet

La spécification technique peut être divisée en deux parties :

- La spec technique du protocole
- La spec ajoutée par MDI pour adapter le protocole au besoin

L'une des majeurs modifications est d'abandonner l'intégration du standard MQTT tel qu il est . On ne garde que le Header de MQTT comme le Header des messages échangés avec le nouveau tcp serveur. Ceci a pour but de garder la compatibilité avec MQTT pour satisfaire les besoins commerciaux.

5.1.2 Architecture de l'itération 2

L'architecture du nouveau BS est conçue d'une façon à ce qu il présente une solution aux problèmes rencontrés par l'architecture de la 1ere itération. La figure ci dessous montre les composants de la nouvelle architecture :

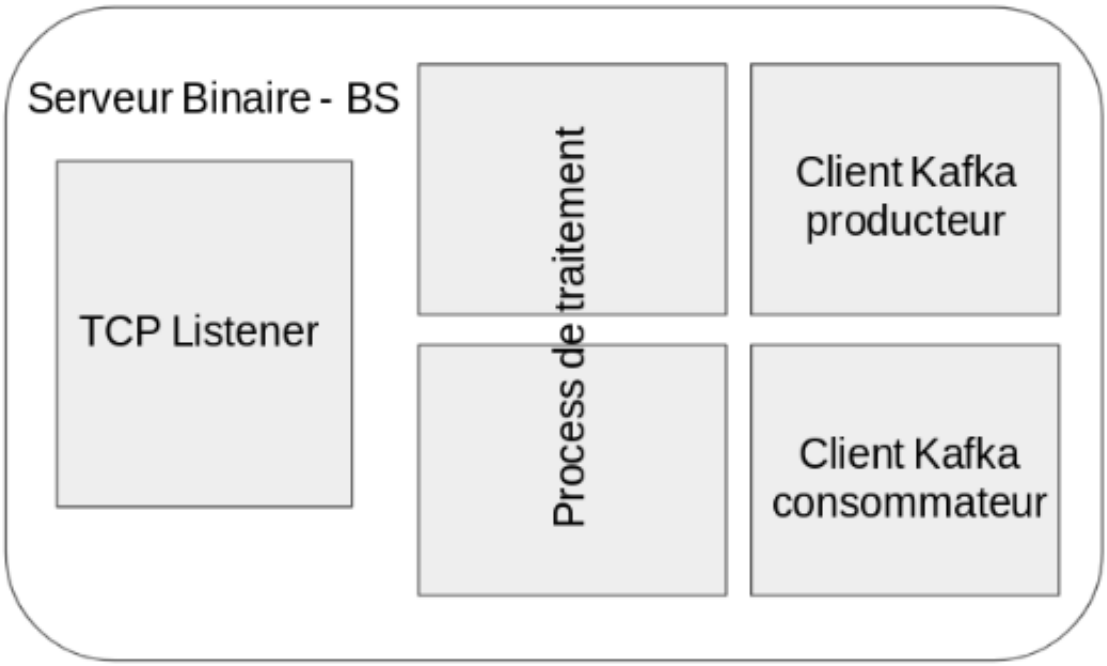

Fig.9: La conception de l'architecture du BS de l'itération 2

Le BS repose sur 3 grands composants : En premier temps, le TCP serveur qui gère la connexion avec le boîtier. En deuxième temps,le traitement des données doit être gérer en parallèle. Ce traitement assure l'encodage du protobuf boîtier au protobuf cloud et vis-versa. D'autre part, deux clients kafka, un producteur qui va publier les messages encodés et un autre consommateur qui va recevoir les messages d event.

5.2 Implémentation

Le nouveau BS lance les diverses traitements dans des traitements en parallèle. L'implémentation de ces processus parallèles s'effectuent grâce aux goroutines du langage go, ce qui englobe toutes les fonctionnalités du BS dans un même composant unifié.

5.2.1 Control plane

Concernant le flux de données ou Control plane, la communication entre ces processus se fait comme le présente la figure 10.

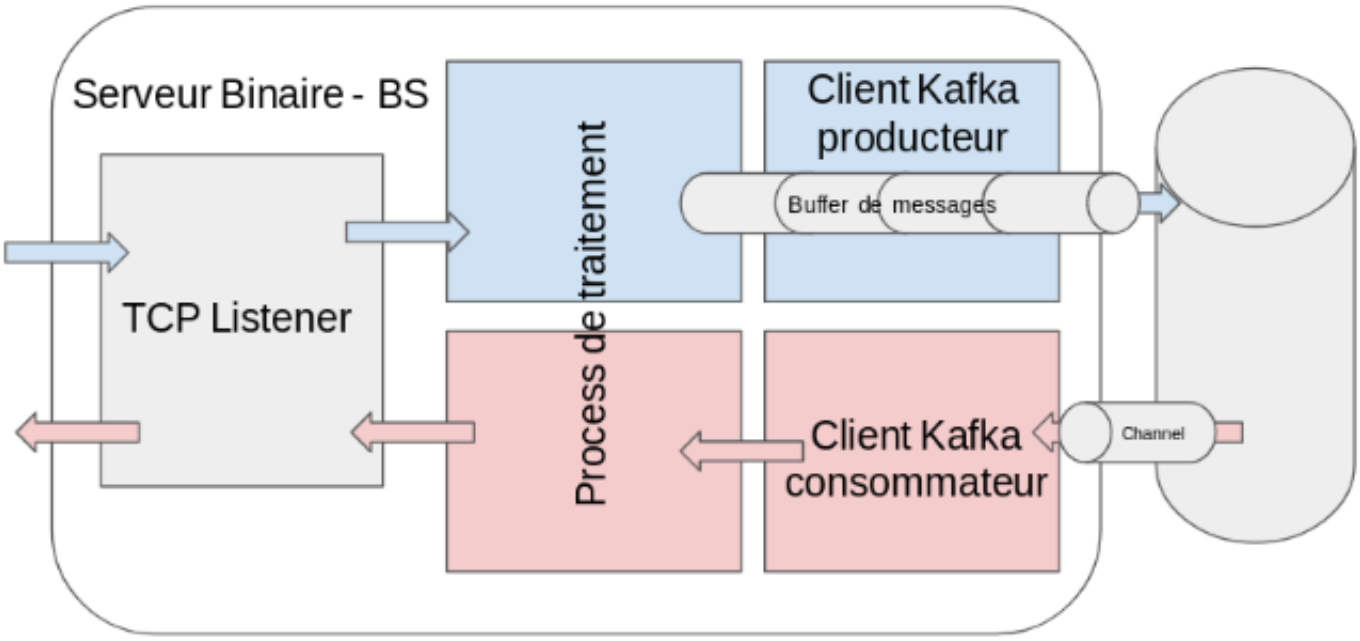

Fig.10: Le flux de données du BS de l'itération 2

- **Le boitier envoie un track pour le cloud :**
  La première connexion se fait avec le TCP Listener. Comme son nom l'indique, ce composant écoute, vérifie et maintient la connexion avec les boîtiers. Il est considéré comme le Générateur. Après avoir vérifier la connexion avec un boîtier donné, il envoie le message reçu au processus de traitement dans un channel.
  Ce dernier décode le format du message et l'encode en format des données du cloud. Ensuite il le met dans un buffer des messages prêts à la publication kafka. Puis le client kafka envoie tous les messages du buffer d'une manière périodique de quelques secondes.
  L'envoi des messages d'une manière périodique minimise l'accès écriture dans la BD ce qui augmente l'efficacité du traitement global.

- **Le cloud envoie un event au boîtier :**
  Le service qui a déclenché l'event va publier dans le broker des messages le message

dans un topic spécifique. Le client consommateur est toujours en écoute sur ce topic, il reçoit le message et l'envoie directement au traitement. Le message s'encode avec le format protobuf du boîtier est maintenant prêt à être envoyer au boîtier. Le TCP listener s'occupe alors de son envoi à son boîtier.
Il est à noter que l'implémentation de cet envoi par rapport à plusieurs instances sera plus complexe.

5.2.2 Go un language différent

Paragraphe sur le language Go : Go est un jeune language de programmation système fascinant. Ce language compilé hérite des idées des paradigmes de programmation impérative et fonctionnelle. Il définit des concepts et des règles qui assurent à l'utilisateur des binaires .

Plusieurs notions de Go ont été implémenté dans le projet MD30. Parmi ces notions et Patterns :

— les Goroutines : Une fonction indépendante en cours d'exécution qui est lancé par une instruction go. Cette goroutine possède sa propre call stack, qui se développe et se réduit comme requiers. Une goroutine est faible en coût. donc le fait de lancer des centaines voire des milliers n'est pas couteux. Une goroutine n'est pas un thread ... Dans un programme go on peu trouver un seul thread avec plusieurs goroutines.

— la gestion Data race : ****** développer un peu sur ça *****

— Go pattern de concurrence : Pipeline : Les primitives de concurrence de Go facilite la construction des data streaming qui prouve l'efficacité d utilisation des I/O avec multiple CPUs. Il n existe pas une définition formelle de pipeline. c'est une parmi plusieurs type de programme concurrentiel. D'une façon générale , une pipeline est une série d'étapes connectés par une "channel", chaque étape est un groupe de goroutines qui exécute la même fonction. Chaque étape les goroutines :
reçoivent une valeur de upstream depuis les channels de lectures effectuent un certain traitement de fonctions sur cette valeur qui produit de nouvelle valeur envoient les nouvelles valeurs downstream pour les channels d'écriture

5.3 Test

Les tests unitaires ... k8s ...

5.4 Amélioration à effectuer

Data plane et l intégration du bon protobuf..

6 Conclusion

Une conclusion globale sur tous le projet, l expérience ...
Apprendre Go était une très belle expérience. La référence principale pour moi était .... m'a servi de référence pour comprendre des concepts sur le fonctionnement interne du language. Mais ça m'a aidé à comprendre un certain nombre de concepts comme le parallélisme.

Il faut aussi dire que le bagage fourni par les cours de compilation, d'algorithmique et de complexité à INSAT était d'une utilité exceptionnelle, surtout que programmer en Go repose sur les notions fondamentales de la programmation.,

Mais la piste principale pour l'apprentissage reste la pratique. Et avec le projet MD30 j'ai eu la chance d'approfondire des recherches, apprendre des concepts et des patterns, et tester, dans un cycle qui a duré 5 mois et demi.

7 Perspectives

Le développement du projet demande encore du travail à fournir et des " features" à développer.

Références

[1] *www.mobile-devices.com*. Site officiel de Mobile Devices Ingénierie.

[2] *www.geek-directeur-technique.org*. Documentation technique sur le cycle itératif.

[3] *www.gdpr-info.eu*. Documentation officielle de la réglementation générale sur la protection des données.

[4] *docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html*. Site officiel de la spécification de MQTT v3.1.1.

[5] *www.github.com/mqtt/mqtt.github.io/wiki*. Documentation technique sur MQTT fait par la communauté MQTT à github.

Glossaire

ASN.1 Abstract Syntaxe Notation One: Syntxe de grammaire utilisé pour décrire les protocoles de communication.. 13

BS Binary Server: service de communication entre Cloud et boîtiers. 5, 6, 10, 13, 14, 17

CC CloudConnect. I, III, 9–12

CN CloudNext. I, 11

GDPR General Data Protection Regulation - Règlement Général sur la protection des données[3] . 11

Kubernetes orchestrateur de conteneurs docker. 19, 20

MD21 Mobile Device 21 : le protocole de communication actuel de MDI. I, 6, 10, 13, 15–17, 20

MD30 Mobile Devices 30 : le nom du nouveau protocol de communication . 6, 17, 25, 27

MDI Mobile Devices Ingénierie. I, 2, 4, 6, 9, 11, 15, 20, 22

OBD ON Board Diagnostics - les boîtiers connectés . 5, 6, 9

Annexes Dans cette annexe on aura : Des détails sur MQTT : - le pattern publish subscribe. - le 3 QoS -