Présentation du projet:

ChirpStack est un projet open source permettant la création de réseaux privés LoRaWAN. Le projet Campus IOT en est un bon exemple d'application. Notre objectif est d'améliorer la plateforme ChirpStack en ajoutant des fonctionnalités. Cela comprend l'ajout de nouvelles contributions, avec l'ajout de technologies Prometheus-Grafana et Kafka, mais également sur l'intégration de contribution datant de l'année dernière, portant sur la sécurisation des gateways et la visualisation de la position d'un device sur une carte interactive.

Organisation de l'équipe

Le travail à effectuer se divisait en trois tâches principales : les intégrations de l'année dernière, l'utilisation de Prometheus-Grafana pour faire du monitoring et l'intégration de Kafka. Nous avons prioriser la première tâche afin de garantir une homogénéité de notre code et d'éviter que de futurs étudiants, reprenant ce projet, ne doivent s'occuper de ce code déjà obsolète actuellement.

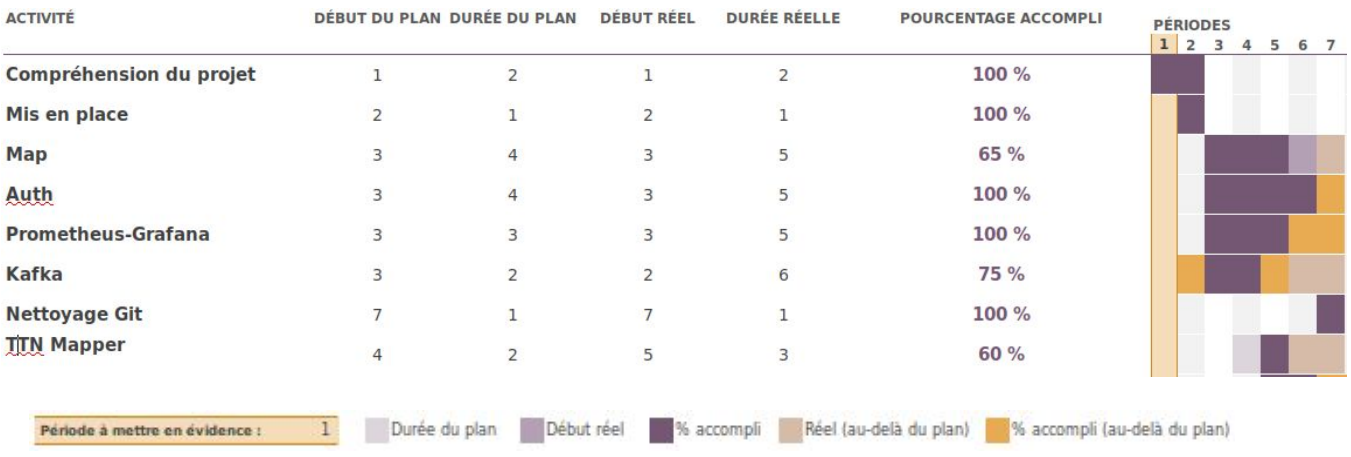
Nous nous sommes ainsi réparti pour que Mathieu et Hoël travaillent principalement sur les contributions de l'an dernier, Iheb et Mandresy se concentrent sur l'utilisation de Grafana et Jian essaye d'intégrer Kafka (avec de l'aide d'autres membres du groupe).

De plus, Julien n'a pu commencé à travailler qu'en cours de projet (à partir de la 4e semaine), il a donc pris en charge seul une nouvelle fonctionnalité basée sur l'ajout de TTN Mapper en tant que composant interne de ChirpStack.

Gestion de projet

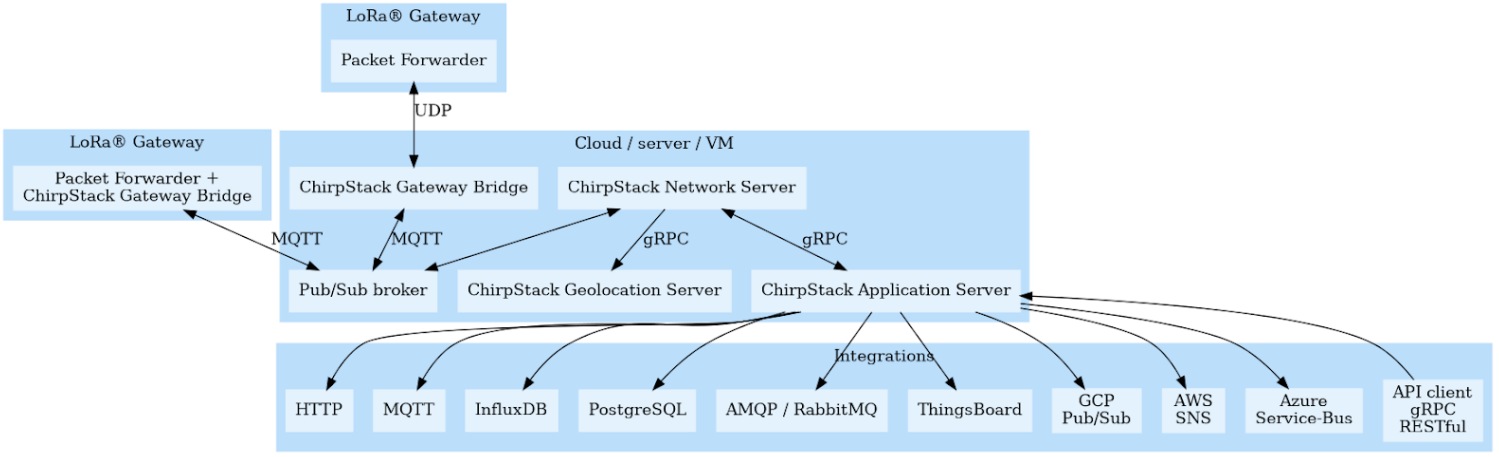
Voici un diagramme montrant la planification du projet

ACTIVITÉ	DÉBUT DU PLAN	DURÉE DU PLAN	DÉBUT RÉEL	DURÉE RÉELLE	POURCENTAGE ACCOMPLI
Compréhension du projet	1	2	1	2	**100 %**
Mis en place	2	1	2	1	**100 %**
Map	3	4	3	5	**65 %**
Auth	3	4	3	5	**100 %**
Prometheus-Grafana	3	3	3	5	**100 %**
Kafka	3	2	2	6	**75 %**
Nettoyage Git	7	1	7	1	**100 %**
TTN Mapper	4	2	5	3	**60 %**



Technologies et outils utilisés

L'architecture de Chirpstack comporte plusieurs composants logiciels résumés dans le schéma ci-dessous qui provient de la documentation de Chirpstack:



Ces composants sont majoritairement codés en Go et en JavaScript. L'API est générée à partir des fichiers de configuration gRPC et Protobuf, et ceci en plusieurs languages (Go, Python, JavaScript/TypeScript et Rust). Notre travail a concerné tous les composants de ChirpStack sauf le gateway bridge.

Prometheus:

Prometheus est un logiciel open source permettant d'enregistrer, en temps réel, des métriques dans une base de données. Le langage PromQL permet ensuite de faire des requêtes vers cette base de données. Plusieurs métriques Prometheus sont incluses dans les composants de ChirpStack mais pas Prometheus lui-même, encourageant à sa future intégration.

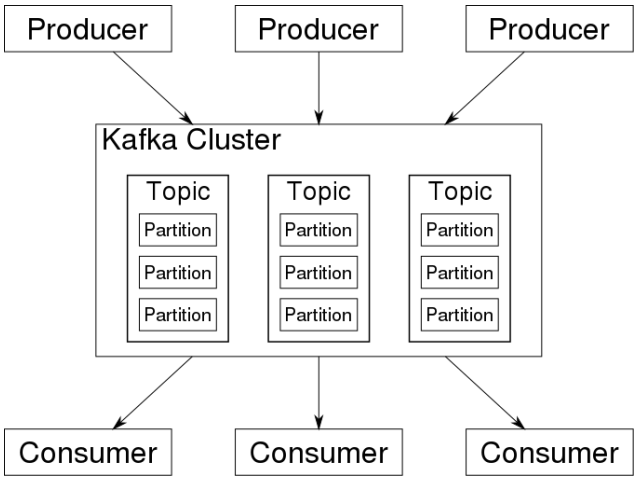
Grafana:

Grafana est un logiciel open source permettant de créer facilement des dashboards personnalisables. Il est spécialisé dans la visualisation de données temporelles, qu'il peut extraire depuis de nombreuses sources ou bases de données. Il est également basé sur une API HTTP. L'association Prometheus-Grafana est une architecture standard pour une utilisation de monitoring d'application.

Kafka:

Apache Kafka est une plateforme opensource de diffusion de messages, permettant d'émettre des flux de données. Il se base sur une architecture

producteur-consommateur organisée en topics où on enregistre des messages. Il permet des manipulations en temps réel sur les flux grâce à un système unifié d'accès, gérant les demandes des consommateurs. Voici un schéma illustrant son fonctionnement général:



(Source de l'image: Wikipedia)

Gitkraken:

Gitkraken est un client graphique pour le logiciel Git qui facilite l'utilisation efficace et fiable de ce dernier et permet d'effectuer la plupart des opérations sans ligne de commande. GitKraken rend facile la visualisation de l'historique git, ce qui devient nécessaire quand on travaille sur un grand projet.

Les intégrations:

- De l'année dernière
    - MQTT authentification des gateways

Cette fonctionnalité avait pour but d'apporter une nouvelle possibilité d'identification des gateways, en les authentifiant avec une clé MQTT. Elle utilisait le plugin Mosquitto-go-auth, réalisé par Iegomez pour l'architecture de Chirpstack. Nous nous sommes aperçu ultérieurement que la plupart du code ne venait pas du groupe d'étudiants de l'an dernier et donc que cette fonctionnalité correspondait plus à un cas d'utilisation du plugin qu'à une nouvelle proposition d'authentification.

Nos difficultés principales sur cette partie provenaient des changements à la base de données, c'est à dire des migrations Postgresql que nous avons ajouté

et qui ont été compliquées à intégrer. En outre une mise à jour de l'image docker de Postgresql a eu lieu durant le projet et nous a obligé à modifier les fichiers de configuration. Nous avons aussi fait face à un problème important de persistance des modifications sur les formulaires, qui a requis de désactiver le cache et de recompiler l'API. Malheureusement la compilation des images docker pour l'API et l'application étaient particulièrement longue et a considérablement ralenti notre avancement. Enfin ce temps de compilation important a rendu compliqué les changements que nous avons apportés, notamment la décision de rendre l'authentification par clé MQTT facultative, afin d'assurer la rétrocompatibilité.

Nous avons finalisé cette fonctionnalité et créé une pull request sur les composants modifiés. Celle-ci n'a pas été acceptée par Brocaar qui l'a considéré trop spécifique et complexifiante pour son architecture, qui vise à offrir une solution simple et prête à l'emploi. Cependant cette fonctionnalité est intéressante et propose un bon exemple d'implémentation du plugin Mosquitto-go-auth, nous l'avons donc mise à disposition de la communauté via le forum Chirpstack. Brocaar a également encouragé la communauté à trouver une façon de généraliser cette fonctionnalité.

- Carte affichant la position d'un device

Cette contribution avait pour objectif d'afficher la position d'un device sur une carte Leaflet, en utilisant ses coordonnées GPS ou un algorithme codé par les étudiants de l'année dernière. Nous avons donc commencé par intégrer leur travail sur la version la plus récente de ChirpStack. Nous avons découvert qu'ils avaient également codé un nouveau backend de géolocalisation, nommée "Willy", afin de pouvoir passer outre la clé de souscription du backend Collos.

Ensuite, il a fallu tester cette partie, en récupérant la position émise par un device et en vérifiant qu'elle s'affichait correctement sur la carte. Nous avons rencontré de nombreuses difficultés et perdu un temps considérable pour faire fonctionner le matériel adéquat (mauvaise compréhension du fonctionnement au départ et fonctionnement assez discontinu).

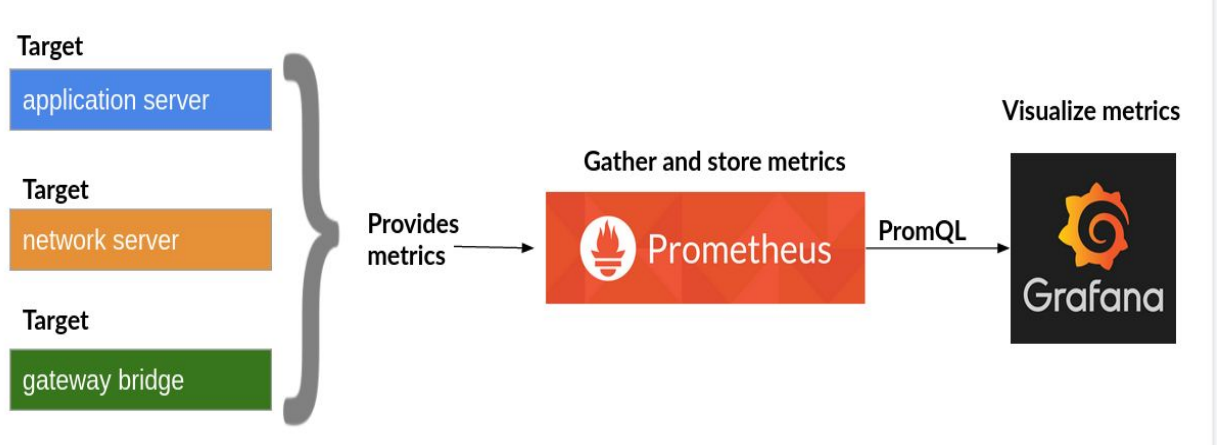
Nous avons cependant pu déterminer que cette contribution n'était pas fonctionnelle et ce pour deux raisons : une mauvaise gestion de la carte, c'est à dire une erreur au niveau de l'UI si la position du device est inconnue (corrigée depuis) et l'algorithme de géolocalisation censé déterminer la position du device ne semble jamais être appelé. De plus, nous ne sommes pas convaincu par l'intérêt de cette fonctionnalité car un grand nombre de devices comportent déjà un composant GPS et connaissent donc leur propre position. Il suffirait alors de récupérer celle-ci dans les messages envoyés par le device et de l'enregistrer dans la base de données pour ensuite l'afficher sur la carte. Enfin nous savons

qu'un groupe d'Ensimag travaille sur un algorithme de géolocalisation, qui sera sûrement plus élaboré que celui réalisé l'année dernière.

En conclusion, cette contribution pourrait être une base intéressante pour ajouter une carte de visualisation sur chaque device à partir d'un algorithme de géolocalisation ou des coordonnées transmises par le device. Cependant celle-ci n'a pas été codée avec rigueur, il paraît donc plus simple de recommencer cette fonctionnalité de zéro avec le travail effectué par le groupe d'Ensimag.

- Prometheus-Grafana

Voici le workflow que nous souhaitions réaliser sur Prometheus et Grafana:



Prometheus a besoin que l'on expose les métriques à surveiller en créant des targets sur les composants cibles. Comme expliqué précédemment, chaque composant de Chirpstack comporte déjà un ensemble de métriques définies qui sont prêtes à être récupérées. La liste des targets disponibles est fournie à Prometheus grâce au fichier de configuration prometheus.yml.

Sur une instance locale de Chirpstack, on peut trouver les métriques fournies par toutes les targets à l'adresse http://localhost:9090 (par défaut), ainsi que les détails et les requêtes PromQL qui sont déjà prêtes à l'exécution.

Pour le dashboard Grafana que l'on trouve sur http://localhost:3000 avec une instance locale de Chirpstack, on peut définir Prometheus comme source des données par défaut. On peut ensuite créer un dashboard et ajouter les requêtes PromQL dans les champs nécessaires. On obtient alors un dashboard qui contient la visualisation graphique des métriques en temps réel.

Nous avons également ouvert une pull request pour cette fonctionnalité mais elle n'a pas non plus été accepté car de nouveau, elle répondait à un besoin de niche et risquait d'augmenter la complexité générale du projet. Cependant nous

l'avons aussi partagée sur le forum ChirpStack pour les personnes qui seraient intéressée par l'utilisation de Grafana pour du monitoring d'application.

- Kafka

L'intégration de Kafka se fait en 3 étapes. Premièrement, l'ajout d'un backend Kafka qui permet de créer un producteur. Deuxièmement, la modification de l'interface utilisateur pour pouvoir ajouter Kafka dans la liste des intégrations. Cette modification implique aussi la dernière étape qui est la modification de l'API de Chirpstack pour ajouter les fonctions d'intégration de Kafka.

Le backend de Kafka ressemble fortement au backend HTTP. Nous avons donc rajouté un fichier kafka.go qui correspond au backend de Kafka. Il importe la librairie "sarama" qui fournit une API de Golang afin de permettre au producteur d'envoyer les messages au serveur Kafka. Quand un utilisateur crée une intégration Kafka sur son instance de Chirpstack, il doit fournir l'adresse du serveur Kafka. Si l'instance de Chirsptack est locale, le serveur Kafka se situera généralement sur localhost:9092 (configurable).

L'utilisateur doit créer un consommateur qui s'abonne aux topics Kafka afin de recevoir les messages de celui-ci. Nous avons utilisé aussi apache nifi en tant que consommateur pour effectuer des tests fonctionnels.

Cette fonctionnalité n'a pas été terminée par manque de temps. L'intégration semble se faire correctement au niveau de l'API mais l'application ne voit pas certaines fonctions pourtant existantes.

- TTN Mapper

Comme son nom l'indique, TTN Mapper est une application pouvant être intégrée à une source de données issue de TTN de manière à visualiser ces données sur une carte. L'idée est donc de récupérer le flux MQTT de données issues du serveur Chirpstack et, via un bridge Node-RED, de les envoyer à TTN Mapper. Cependant, TTN Mapper ne reconnaît qu'un format particulier de données, c'est pourquoi il faut les transformer (grâce à une configuration Node-RED) avant de les envoyer au service.

Dans un premier temps donc, quelques commandes shell nous permettent de récupérer les flots de données qui sont transmises au serveur via une passerelle.

L'outil Node-RED est ensuite utilisé pour créer le bridge entre MQTT et TTN ; grâce à une disposition de noeuds précise et une configuration spécifique, on peut aisément recueillir les flots de données et les transmettre à TTN. Cependant, comme expliqué précédemment, il est important de modifier le format de ces données. Un noeud intégrant une fonction JavaScript permet alors de filtrer les données pour ne récupérer que celles émises par un objet en particulier et de construire le JSON qui sera alors conduit vers TTN. Plusieurs clés sont ajoutées, d'autres sont supprimées ou renommées pour que le service soit au final capable d'interpréter ces données.

Vient donc ensuite la troisième étape, celle de la récupération et la reconnaissance des données par TTN Mapper. Bien que cette partie n'ait pas pu être réalisée en local, grâce à une instance privée, comme demandé, elle a partiellement été faite en ligne via la console TTN et l'intégration de TTN Mapper à cette dernière. Les flots de données formatés y sont visualisés et décodés grâce à une fonction appropriée, cependant le service ne semble pas les reconnaître pleinement, probablement à cause d'une erreur lors de leur formatage.

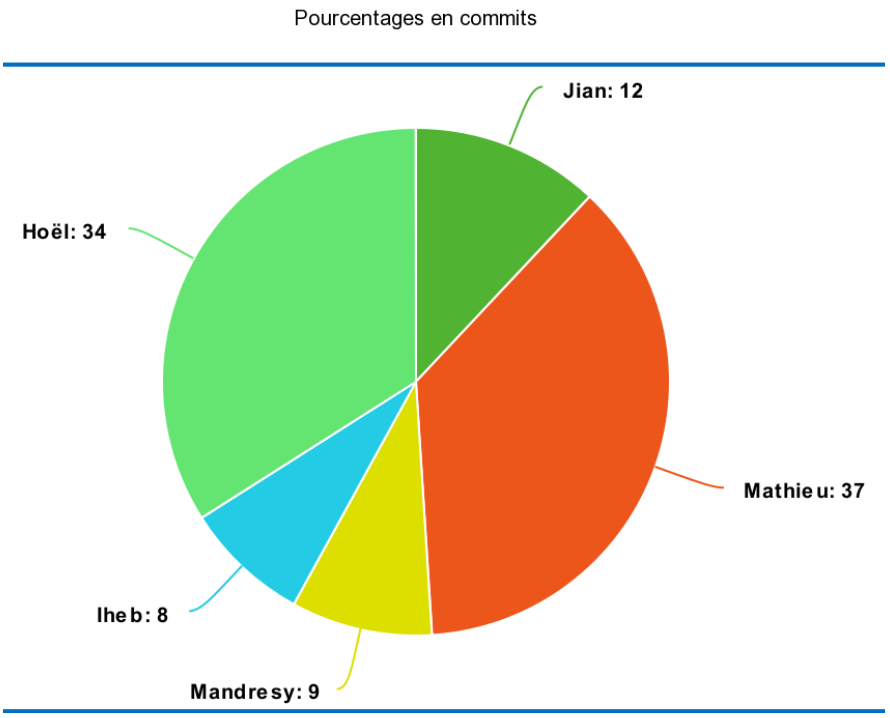
Comme énoncé, la création d'une instance locale de TTN Mapper n'a pas été finalisée. En plus de difficultés d'installation sur le serveur, l'accès à l'adresse locale permettant la visualisation des données sur la carte n'a pas été possible.

Métriques logicielles

Les seuls langages que nous avons utilisé dans ce projet sont le Go et le Javascript.

Nous avons estimé le temps ingénieur de notre projet à 1060h en tout parmi les 6 personnes du projet, en comptant tous les créneaux où nous avons travaillé à 5 et en ajoutant en plus le temps de travail de Julien. Nous avons décidé également de calculer le coût salarial de notre projet en nous basant sur une moyenne de salaire d'ingénieur IoT junior. Nous avons trouvé un salaire horaire moyen de 15.96€/h brut soit 12.29€/h net; ce qui fait donc un coût salarial total de 16 918€ en brut ou 13 027€ en net.

Voici la répartition des commits dans le groupe en pourcentage (Julien étant exclu car ayant travailler indépendamment et sur un temps différent).



Conclusion

A la fin de ce projet nous avons pu faire deux pull-requests sur les répertoires ChirpStack, concernant le travail du groupe de l'année dernière sur l'authentification des gateways par des clés MQTT et la visualisation de métriques Prometheus avec un dashboard Grafana. Nous avons également ouvert trois issues, afin d'échanger principalement sur les fonctionnalités plutôt que sur leur implémentation et être à l'écoute des attentes de la communauté.

Ce travail nous a permis d'avoir une première expérience de projet longue durée avec un aspect open source, où notre travail consistait à participer au développement d'un projet existant. Malgré le refus de nos pull request, notre travail a pu être partagé à la communauté et nous avons eu l'occasion d'avoir des discussions très intéressantes sur la problématique d'intégration de fonctionnalités dans un projet open source. De plus, l'analyse que nous avons fait des modèles de conception de ChirpStack pourra nous aider à développer nos propres projets à l'avenir. Nous avons également pu aborder de nouvelles technologies, comme l'utilisation de Docker dans un environnement de développement et la manipulations d'objets LoRa (devices et gateways).

Nous avons également acquis de l'expérience pour nos soft skills, au niveau du travail d'équipe et de l'organisation d'un projet.

Annexes

Liens des pull-requests, issues et contribution au forum :

Pull request Grafana : https://github.com/brocaar/chirpstack-docker/pull/30
Pull request MQTT : https://github.com/brocaar/chirpstack-docker/pull/31
https://github.com/brocaar/chirpstack-api/pull/12

https://github.com/brocaar/chirpstack-application-server/pull/442
Issue Grafana :
https://github.com/brocaar/chirpstack-network-server/issues/468
Issue Map :
https://github.com/brocaar/chirpstack-application-server/issues/439
Issue MQTT :
https://github.com/brocaar/chirpstack-application-server/issues/427

Forum MQTT :
https://forum.chirpstack.io/t/mqtt-authentification-for-gateways/7787
Forum Grafana :
https://forum.chirpstack.io/t/printing-of-prometheus-metrics-by-grafana-dashboard/7777

Bibliographie :

- Schéma Kafka :
  - https://commons.wikimedia.org/wiki/File:Overview_of_Apache_Kafka.svg?uselang=fr
- Schéma Chirpstack :
  - https://www.chirpstack.io/overview/architecture/
- Forum Chirpstack :
  - https://forum.chirpstack.io

Screenshots Grafana :