ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

UNIVERSITÉ DU QUÉBEC

RAPPORT FINAL TECHNIQUE

PRÉSENTÉ À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
DANS LE CADRE DU COURS GTI795 - PROJET DE FIN D'ÉTUDES
EN GÉNIE DES TI

PFE06
SUIVI À DISTANCE D'ÉQUIPEMENTS DE CONTRÔLE DE PROCÉDÉS

Rajani, DEJEAN (DEJR06059900)
F. Ange-Christian, SILUE (SILF27039901)
Alan, THOMAS (THOA11089409)

DÉPARTEMENT DE GÉNIE LOGICIEL ET DES TI

Professeur-superviseur
ALAIN APRIL, PHD

MONTRÉAL, 26 AVRIL2023

REMERCIEMENTS

Nous tenons à remercier l'équipe de ChemBrains qui ont su nous accompagner durant ce Projet de Fin d'Études (PFE) et partager leur passion pour l'écologie avec leur solution de traitement des eaux, qui nous l'espérons, aura un impact positif dans le monde de demain.

Nous voulons aussi remercier Pr. Alain April et Pr. Mathieu Dupuis pour leurs conseils et l'aide apportée au projet. Parfois, il ne suffit que d'un coup de pouce dans la bonne direction.

RÉSUMÉ

PFE06 - Suivi à distance d'équipements de contrôle de procédés

Dans le cadre du Projet de Fin d'Études (PFE), on a travaillé avec une start-up du nom de ChemBrains dans le modèle d'affaire est basé sur le traitement des eaux usées en minimisant au maximum l'utilisation de produits chimique dans le processus de recyclage de l'eau. Le problème que ChemBrains a rencontré lors du développement de leurs solutions c'est de pouvoir trouver un moyen de faire du suivi des différentes installations qui seront installés à travers la région de Montréal. C'est dans cette problématique particulière qu'on a pu apporter nos connaissances et notre savoir-faire pour proposer une solution qui va pouvoir s'adapter aux besoins grandissant de Chembrains.

Actuellement, les Automates Programmables Industriels (APIs) ne disposent d'aucun moyen pour envoyer de l'information par internet et il n'y a aucune structure qui est capable de les récupérer et de le traiter. Il a donc fallu travailler sur plusieurs aspects de la solution qui sont :

- La connexion des équipements à distance
- La récupération et le traitement des données
- La création d'un tableau de bord pour afficher les données
- La mise en place d'un système d'alerte

Ces 4 grands axes ont permis de définir les limites du projet et de donner une direction à suivre. La solution proposée est donc constituée de la configuration des APIs pour l'envoi de données au travers d'un routeur utilisant une carte SIM et la création d'un serveur qui accueillera la pile technologique nécessaire au processus du traitement de la donnée. Le serveur est constitué de l'installation du courtier Mosquitto pour la gestion des envois des messages MQTT, de Node-RED pour le traitement des données, d'InfluxDB pour le stockage et la gestion et données et de Grafana pour l'affichage d'un tableau de bord et de la configuration des alertes.

L'avantage de la solution dans sa globalité, c'est qu'elle offre tout ce dont ChemBrains a besoin actuellement pour répondre à leur problématique et plus encore car qu'elle sera capable de suivre leur évolution en termes de taille et de besoin.

Mots-clés :
Recyclage, eaux usées, MQTT, API, Mosquitto, Node-RED, InfluxDB, Grafana

Liste des abréviations, sigles et acronymes

Expression	Définition
API	Acronyme de "Application Programming Interface". Il s'agit d'un ensemble de protocoles, d'outils et de règles qui permettent à différents logiciels de communiquer entre eux.
Backend	C’est la partie d'une application informatique qui se trouve du côté du serveur et qui est responsable de la gestion des données et de la logique métier de l'application.
Cloud	Le "cloud" (ou "informatique en nuage" en français) désigne un modèle de fourniture de services informatiques à la demande via Internet. Plutôt que de posséder et de gérer des ressources informatiques physiques telles que des serveurs, des disques de stockage et des réseaux, les utilisateurs peuvent accéder à des ressources informatiques virtuelles via Internet, en payant uniquement pour ce qu'ils utilisent.
Daemon	Un "daemon" (ou "démon" en français) est un programme informatique qui s'exécute en arrière-plan sur un système d'exploitation et qui effectue des tâches de manière autonome, sans intervention directe de l'utilisateur.
Docker	Une image Docker est un package qui contient tout le nécessaire pour exécuter une application, y compris le code, les bibliothèques, les dépendances et les fichiers de configuration.
Frontend	C’est la partie visible d'une application informatique, qui est destinée à l'interaction avec l'utilisateur final. Il est responsable de la présentation des données et de l'interface utilisateur de l'application.
IoT	IoT (Internet des objets) est un terme qui désigne la connectivité et l'échange de données entre des objets physiques et l'internet.
MQTT	MQTT (Message Queuing Telemetry Transport) est un protocole de communication de messagerie pour les objets connectés à faible consommation d'énergie et à bande passante limitée.
Open-source	Fait référence à un modèle de développement de logiciel dans lequel le code source est disponible publiquement et peut être librement utilisé, modifié et distribué par quiconque.

Payload	Le terme "payload" (ou "charge utile" en français) désigne les données ou les informations pertinentes qui sont transportées par un message ou un paquet de données.
PLC	PLC (Programmable Logic Controller), également connu sous le nom d'automate programmable industriel (API), est un ordinateur industriel programmable conçu pour contrôler et automatiser les processus de fabrication dans les usines et les installations industrielles.
Timestamp	Un "timestamp" (ou "horodatage" en français) est une représentation numérique d'une date et d'une heure spécifiques, souvent utilisée pour marquer le moment où un événement s'est produit.
Topic	Un "MQTT topic" (ou "sujet MQTT" en français) est une chaîne de caractères qui identifie un canal de communication pour les messages MQTT.

Table des matières

Listes des figures

Listes des tableaux

Chapitre I - Introduction

ChemBrains est une start-up qui fait partie du programme du Centech depuis l'été 2021, mais qui travaille sur des technologies de traitements des eaux usées depuis 2020. Elle offre une suite de technologie pour le traitement de certains types d'eaux comme l'eau potable, l'eau des piscines ou bien l'eau industrielle. Sa gamme de produits peut donc s'adapter au besoin du client en termes de taille et de règlementation. L'installation pour le traitement des eaux est donc installée directement chez le client, ce qui pose des défis en soi, car pour faire le suivi et le maintien de l'installation, il faut absolument qu'une personne soit sur place pour vérifier continuellement que tout fonctionne correctement.

La problématique qui se dessine est que pour voir les données recueillies par l'installation il faut absolument une personne sur place, mais ce n'est pas quelque chose de faisable lorsque les installations pourraient se faire à travers le Québec. De plus, il est difficilement possible de surveiller une machinerie 24 heures sur 24, car en termes d'organisation et de ressources humaines, cela revient à un certain coût. C'est de cette problématique qu'est né le Projet de Fin de Session (PFE), mettre en place une infrastructure technologique capable de recevoir ces données et de permettre le suivi de toutes les installations à distance.

De plus, la solution qui devra être adoptée devra remplir certaines conditions pour pouvoir être validée. La première est qu'étant donné la structure de la petite entreprise actuelle, la solution devra être dynamique en minimisant au maximum l'intervention d'une personne sur la solution développée, car il n'y a personne au sein de l'entreprise pouvant reprendre le projet en main une fois la passation faite. Deuxièmement, il faut utiliser des technologies open source, car la solution n'aura pas accès des fonds et le matériel sur place sera le matériel devant être utilisé pour la solution finale. Et finalement, la solution ne pourra pas utiliser la connexion internet des clients pour des raisons de sécurités et devra donc être indépendante.

Dans un premier temps, on va développer plus en détail le contexte du projet. Par la suite, on va parler de la démarche menant à la solution et des outils qui ont aidé à l'organisation du projet. Ensuite, on parlera des différents aspects de la solution. Et finalement, on parlera de la solution dans sa version actuelle ainsi que des difficultés rencontrées lors de son développement avec une liste de recommandation pour le futur.

Chapitre II - Contexte du projet

2.1 - Description du contexte

ChemBrains possède trois technologies de traitement des eaux usées dotées de sondes intégrées aux systèmes. On peut mesurer par exemple des valeurs pour le pH, la température, la turbidité, la pression, la conductivité, le potentiel d'oxydoréduction, etc. Ces données sont récoltées actuellement dans un API qui est la tête pensante du système.

Actuellement, ce système n'a aucun moyen de communication, les informations collectées sont internes au système et les alarmes ne peuvent être détectées que localement, car il n'y a qu'un signal visuel en cas de problème. Puisque ChemBrains a pour ambitions d'installer leurs installations à plusieurs endroits géographiquement, ils ont des problèmes pour intervenir facilement sur toutes les machines et voudraient pouvoir voir en temps réel les données qui sont collectées, recevoir des alarmes en cas de problème et générer des rapports.

Ils ont sous la main un vieux serveur, deux routeurs sans nom et des cartes SIM comme matériel disponible pour la réalisation du projet et le but est de faire fonctionner le tout avec l'équipement disponible sur le site.

2.2 - Analyse des besoins du client

Pour rappel, à la fin des 3 mois et demi qui sont utilisés pour le développement de la solution, ChemBrains compte mettre la solution en production directement une fois que la passation du projet sera terminée. Il faut donc que le projet soit fonctionnel dans la fenêtre de temps qui est disponible.

Caractéristique de la connexion API au serveur
Création d'identifiants pour les services
Connexion sécurisée
Caractéristique de la solution serveur
Stocker les données des capteurs
Créer des rapports
Afficher des tableaux de bord
Caracteristique de la solution
Solution dynamique
Solution avec peu d'entretien
Facile d'utilisation
Documenté avec des cas d'utilisation
Coût très faible

Tableau 1: Besoins du client

Pour les caractéristiques de la connexion entre l'API et le serveur, il faut que la connexion de l'API au serveur soit protégée par un chiffrement TLS et la connexion aux différents services doit être faite par l'intermédiaire de la combinaison d'un nom d'utilisateur et d'un mot de passe.

Pour les caractéristiques de la solution serveur, en plus de pouvoir stocker les données, il faut pouvoir les consulter facilement et créer des graphiques qui vont permettre de voir rapidement l'évolution des valeurs dans une plage donnée. Il faudrait aussi pouvoir extraire les données dans un format CSV dans le cas où le client de ChemBrains voudrait un rapport.

Et finalement pour les caractéristiques de la solution, il faut que la solution soit dynamique sur plusieurs niveaux. Il faut dans un premier temps que la solution soit dynamique, car ils ont 3 technologies et dans le futur, s'ils rajoutent d'autres méthodes de traitement des eaux, que notre solution s'adapte avec un peu ou pas de changement. Il faut dans un second temps que la solution ne soit pas regardante sur le type de données qui est envoyé, car la structure de celle-ci peut changer du tout au tout en fonction des technologies, mais aussi en fonction des versions de cette technologie. Il faut ensuite qu'à l'ajout d'une machine sur un site, que la connexion avec le serveur soit suffisante en termes de configuration, car n'ayant personne de technique en informatique au sein de l'entreprise, les modifications devant être apportées doivent être minimes. La solution retenue devra être aussi simple d'utilisation que possible, être accompagnée d'une aide visuelle sous la forme d'un document pour aider les utilisateurs de la solution et le coût de la solution devra rester aussi faible que possible.

2.3 - La problématique

En ayant maintenant connaissance des besoins du client, ChemBrains a besoin d'un projet clé en main qui sera développé en 3 mois et demi qui pourra stocker les données des capteurs envoyées par les APIs et générer des rapports et des graphiques. La solution devra être dynamique et s'autogérer pour qu'elle soit simple d'utilisation.

Chapitre III - Méthodologie de travail

3.1 - Fonctionnement de l'équipe et répartition des tâches

L'équipe est composée de 3 membres et on a choisi de travailler selon une méthode agile. On se rencontre une fois toutes les deux semaines le vendredi soir pour définir les objectifs du prochain sprint. Au besoin, on peut toujours contacter les membres de l'équipe ou proposer une réunion plus tôt si le besoin se faire ressentir. Donc, après le rendez-vous initial avec le client pour cerner ses besoins, on a planifié une solution préliminaire et on a commencé à répartir des tâches en fonction des envies et des compétences de chacun. Ange Christian était sur la partie réseautique pendant qu'Alan et Rajani travaillaient sur l'ensemble de la pile technologique. Afin de coordonner les rendez-vous et les communications entre le professeur ou le client, Alan a été nommé chef d'équipe.

3.2 - Méthodes de communication

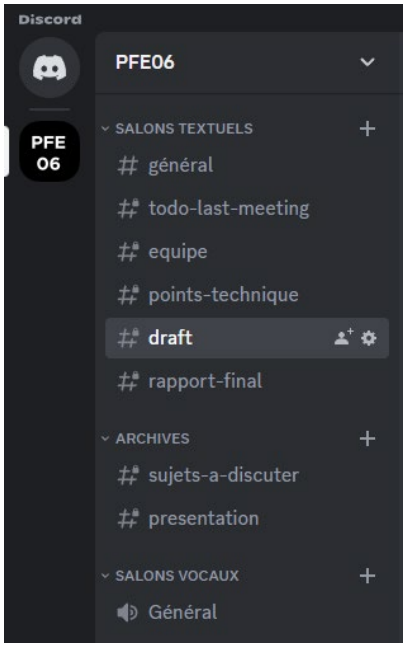

Figure 1: Vue du serveur Discord PFE06

Pour garder le contact, chaque membre de l'équipe a donné son numéro de téléphone en cas d'urgence. Les adresses courriel de l'école ont été partagées et un serveur Discord a été monté pour une utilisation plus journalière. Dans le serveur discord, les utilisateurs sont le client, le professeur et les membres de l'équipe. Pour des raisons évidentes, des rôles ont été créés pour permettre aux membres de l'équipe d'être administrateur du serveur Discord pour pouvoir créer des salons au besoin et de cacher la vue de certains salons au reste des utilisateurs. Il y a trois grandes sections, la section salon textuelle qui regroupe tous les points de discussion que les administrateurs ont accès alors qu'un utilisateur normal ne voit que le salon #général et le salon vocal. Dès que le besoin se fait sentir, un nouveau salon est créé puis supprimé si le contenu était réglé. Dans le cas où le contenu pourrait encore avoir de la valeur plus tard dans le projet, une section archives a été mise en place.

3.3 - Méthode de travail et outils utilisés

Plusieurs outils ont été testés pour le maintien de la connaissance durant le projet comme Obsidian, un document sur git, un fichier Word partagé, Evernote, etc. Finalement, pour des raisons de simplicité d'utilisation et de partage, il fallait prendre quelque chose qui soit assez facile pour mettre rapidement du contenu en ligne. On a donc choisi Notion pour avoir en place un environnement avec de la documentation, les articles intéressants ou bien

de la documentation sur l'architecture du projet qu'on pourra récupérer et mettre à jour rapidement. La page d'accueil ressemble à cela :

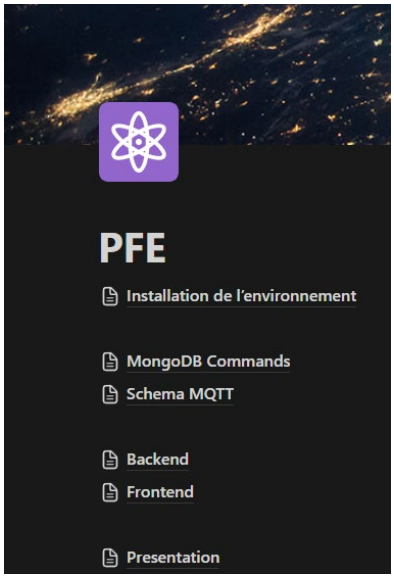

Figure 2: Page d'accueil du Notion de l'équipe

Lorsqu'on clique sur la page « Installation de l'environnement », on tombe sur :

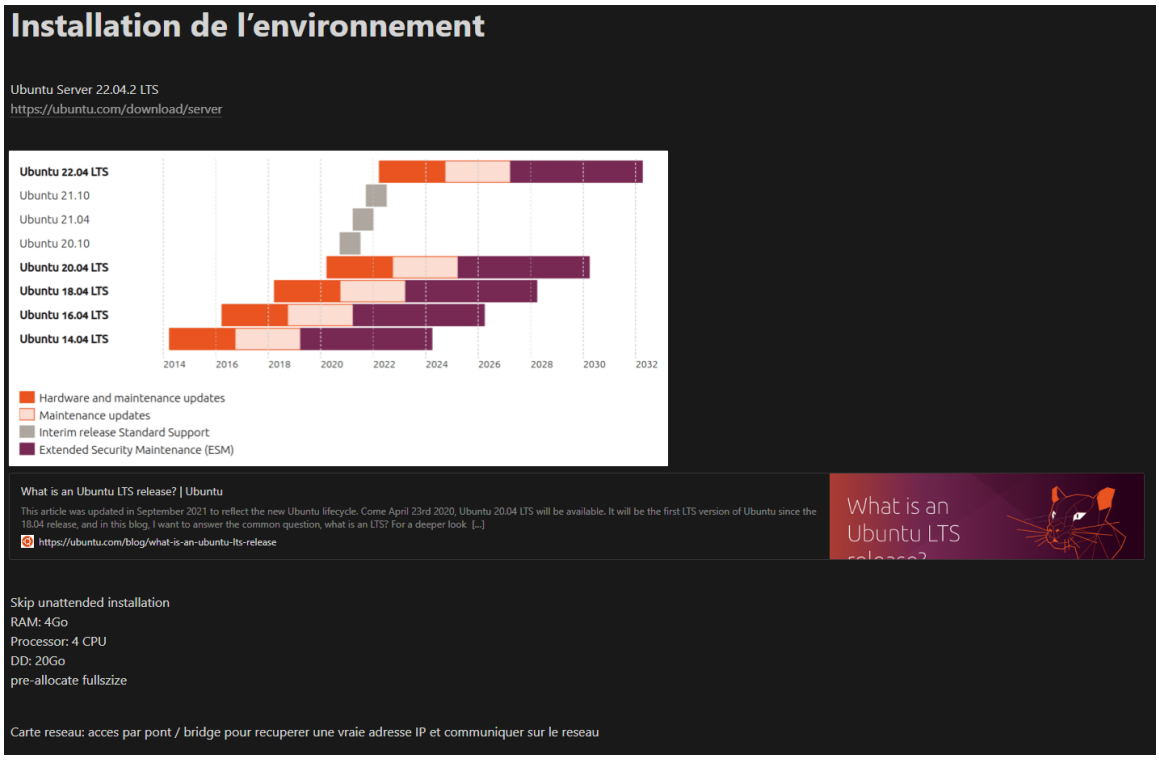

Figure 3: Page d'installation de l'environnement de Notion de l'équipe

On ne voit pas, bien entendu, la totalité de son contenu, mais on peut défiler sur la page pour retrouver les informations importantes que l'on a récupérées pour la création de l'environnement et des commandes utilisées. Cette base de connaissance commune a servi

à créer le fichier « Documentation technique » et ce document a été remis à ChemBrains pour les aider dans le maintien du projet et son utilisation.

De plus pour tout ce qui est en rapport avec le partage du travail, l'utilisation d'un dossier OneDrive et d'un dépôt git ont grandement facilité la tâche.

3.4 - Erreurs dans l'approche du travail d'équipe

On a mis en place des environnements pour partager notre travail, des endroits pour discuter, un endroit pour partager les connaissances, mais la dynamique n'était pas encore présente durant les premières semaines de la session. La rencontre avec le client était un peu difficile au niveau des disponibilités, personne ne se connaissait, le mandat n'était pas clair et changé d'une rencontre à une autre. Le début était un peu difficile et il a fallu prendre l'habitude d'utiliser les ressources qu'on s'était mises à disposition pour faciliter la coopération.

On a essayé de faire un suivi avec un tableau Kanban durant les premières semaines, mais qui n'a pas eu un grand succès.

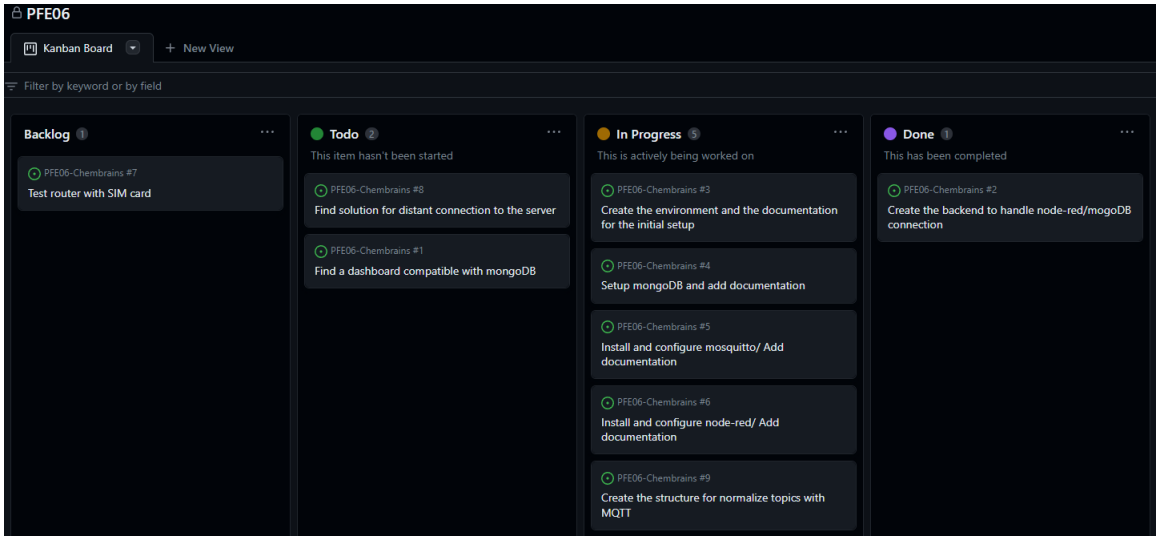

Figure 4: Vue du tableau Kanban de la semaine 4

Pour qu'un outil soit efficace, il faut qu'il soit adopté par la majorité et dans notre cas par la totalité de l'équipe. Le tableau Kanban a rapidement été mis de côté voyant que son taux d'adoption était faible. Ce qui n'est pas un problème, il faut juste trouver d'autres moyens pour remplacer le besoin auquel répondait ce tableau, c'est-à-dire avoir une vue d'ensemble des tâches accomplies, du travail en cours et du travail à effectuer. On a donc intégré à nos rendez-vous d'équipe une section pour la répartition des tâches où l'on mettait sur Discord ces informations.

Chapitre IV - Description de la solution et rôle des technologies

Voici un aperçu de la solution développé et qui est actuellement en production dans le local de ChemBrains. Dans cette partie, on va discuter du rôle des différentes technologies qui font partie de notre pile technologique.

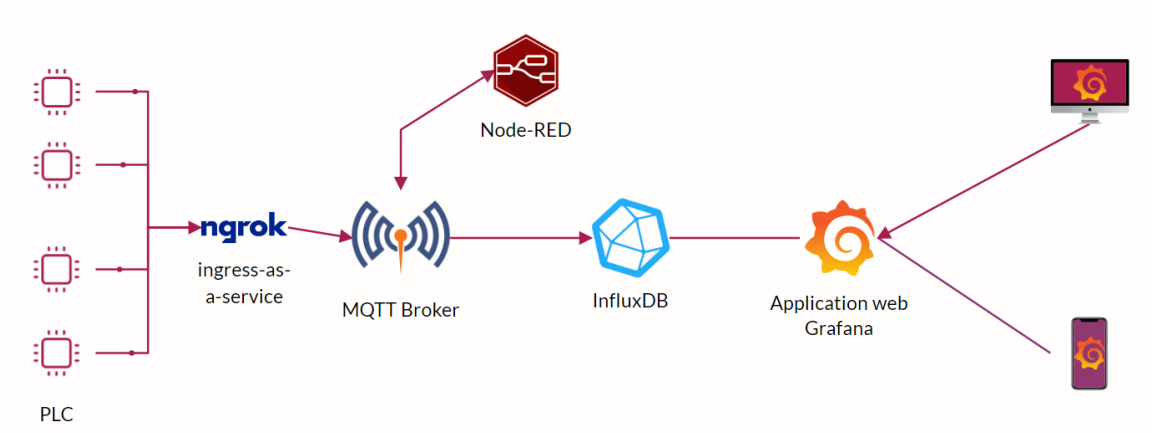

Figure 5: Représentation de la solution actuelle

4.1 - Serveur Ubuntu

Pour héberger notre solution, on a choisi d'utiliser la dernière version LTS (Long Time Support) d'Ubuntu. Elle obtiendra des mises à jour gratuites jusqu'en 2027. La version 22.04, appelée Jammy Jellyfish, permettra à l'entreprise d'ici là de mettre à jour son infrastructure et de porter potentiellement son installation dans un autre environnement s'il ne passe pas dans la version professionnelle pour profiter de l'ESM (Extended Security Maintenance).

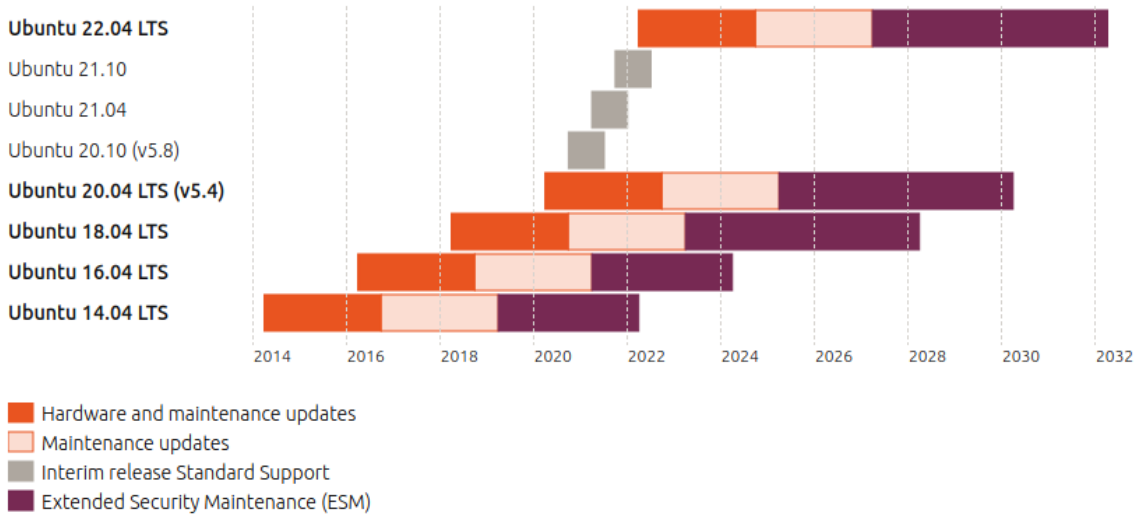

Figure 6: Référence du temps de maintien des versions LTS d'Ubuntu
https://ubuntu.com/blog/what-is-an-ubuntu-lts-release

Sur le serveur Ubuntu seront installés tous les services dont on aura besoin. Il y aura, Mosquitto, Ngrok, Node-RED, InfluxDB, Telegraf et Grafana. Ce qui nous permet de mettre en place tout un système de daemon qui se relance en cas de problème. Tout est automatisé pour qu'au lancement de l'ordinateur, tout fonctionne nativement.

4.2 - MQTT et Mosquitto

MQTT est un protocole de messagerie de publication-abonnement. Il est très utilisé dans le milieu de l'IoT, car il a une faible empreinte énergétique, il utilise peu de bande passante et il est facile d'utilisation.

Le principe fondamental de MQTT est basé sur deux rôles clés : les publishers et les subscribers.

Un publisher est un dispositif ou une application qui envoie des messages à un broker MQTT. Les messages peuvent contenir des données, des commandes ou des informations de statut. Les publishers peuvent publier des messages sur un ou plusieurs topics (sujets), qui sont des canaux de communication thématiques. Les topics sont identifiés par des noms de chaînes de caractères et organisent les messages de manière hiérarchique.

D'un autre côté, un subscriber est un dispositif ou une application qui s'abonne à un ou plusieurs topics sur le broker MQTT. Lorsqu'un publisher publie un message sur un topic, le courtier MQTT transmet le message à tous les subscribers qui sont abonnés à ce topic. Les subscribers peuvent alors utiliser les données reçues pour déclencher des actions ou mettre à jour des informations.

MQTT fonctionne avec des topics et c'est comme cela qu'une structure est créée. Dans notre cas, on a 3 topics principaux :

- data/nomAPI/sensors
- data/nomAPI/io
- data/nomAPI/alerts

Ce que cela permet de faire, c'est que lorsqu'un API du nom de AquaPark publie avec MQTT des données, si ces données représentent les informations des capteurs, alors elle le publie sur le topic data/aquaPark/sensors. Du côté serveur, on peut s'abonner à tous les sujets qui nous intéressent en utilisant des jetons. Par exemple, sur le serveur on écoute sur le topic data/+/sensors. Le « + » sert ici de jeton et permet de s'inscrire dynamiquement à tous les topics quelque soit le nom de l'API. On parlait de l'importance de diminuer les actions requises pour modifier notre solution lorsque de nouvelles machines seront mises en place, MQTT permet d'apporter cette flexibilité.

On a donc installé Mosquitto qui est une version open source disponible sur Ubuntu qui remplit le rôle du courtier MQTT. Donc, le serveur est responsable de la réception et de l'envoi des messages.

4.3 - APIs

Les Automates Programmables Industriel (API) récoltent les données d'une installation et peuvent les envoyer sur internet avec le protocole MQTT. La manière dont la donnée est récoltée était la responsabilité de l'ingénieur électromécanique sur place, mais on a aidé à la configuration du protocole MQTT sur ces appareils. La configuration se fait à l'aide d'un module de l'API, pour que l'envoi fonctionne, il suffit de rajouter l'adresse de notre serveur, le port utilisé ainsi que de mettre en place la configuration de quelle donnée va dans quel topic.

4.4 - Twilio

Twilio est une entreprise spécialisée dans les télécommunications et elle offre des cartes SIM spécialisées pour l'IoT. Travaillant directement avec les fournisseurs locaux, elle possède une grande couverture sur le territoire. Twilio nous a été imposé et il a fallu activer les cartes SIM et les configurer pour qu'elle puisse envoyer des données.

4.5 - Ngrok

Ngork est la seule partie payante de notre solution. ChemBrains étant toujours au Centech, n'ayant pas de serveur, il n'y avait aucun moyen de communication certain entre un API et l'ordinateur qui héberge notre solution et c'est là que vient en jeu Ngrok. Son rôle est de créer un point fixe qui pourra être utilisé dans par les APIs pour contacter le serveur où qu'il soit et quelques soit son adresse IP. Ngrok fournit donc un point d'accès, mais aussi

une connexion sécurisée avec le protocole TLS. De plus, il est facile de le déployer dans l'environnement et permet le suivi du trafic.

4.6 - Node-RED

Node-RED est un outil visuel pour construire des applications spécialisées pour l'internet des objets, mais pas seulement. Sa philosophie est basée sur un développement orienté-évènement, c'est-à-dire que ce sont des évènements qui vont déclencher une suite d'action et dans notre cas ce sont les messages envoyés par les APIs via le protocole MQTT qu'on récupère et traite avant de les envoyer dans la base de données.

C'est une programmation basée sur des flux, ce sont des nœuds qui sont reliés par un ou plusieurs flux qui vont établir le traitement de la donnée. De plus, lorsqu'un changement est effectué, on peut pousser les changements en un seul clic et tous les nouveaux ajouts seront pris en compte immédiatement. Il est très facile de faire des itérations rapides et construire de nouveaux mécanismes.

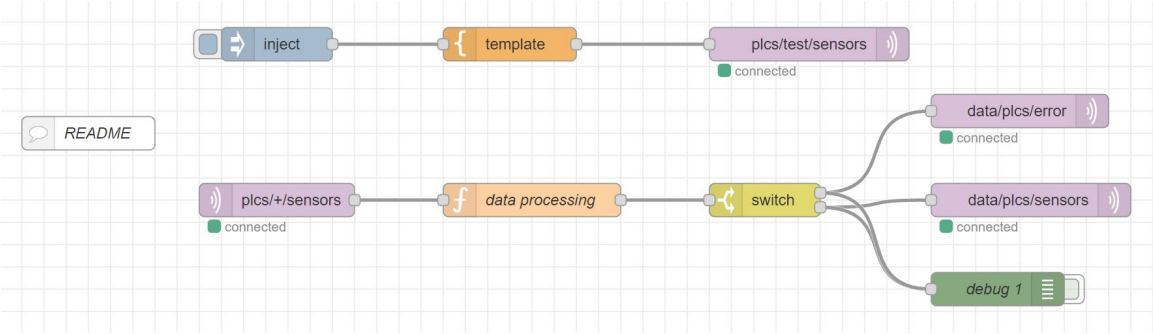

Figure 7: Exemple de notre solution sur Node-RED

Voici une partie de notre solution sur Node-RED. Il y a deux flux présentement, la première en haut qui commence par le nœud inject et la seconde en bas qui commence par le nœud mauve.

La première en haut, sert à simuler à l'intérieur de Node-RED un flux de données que pourrait envoyer un API. En effet, si on regarde de plus près ce qu'elle fait, lorsqu'on va appuyer sur le bouton bleu, qui va être notre évènement finalement, elle sert simplement à lancer le flux.

Le prochain nœud template permet d'écrire le payload d'un message et de le transmettre au nœud suivant. À l'intérieur de ce message, on a actuellement une suite de nombre séparé par des virgules qui représente des nombres aléatoires que pourrait envoyer l'API et chacun de ces nombres représente la valeur d'un des capteurs du système.

Finalement, le dernier nœud est un nœud MQTT qui nous permet d'envoyer un message sur le topic plcs/test/sensors qui a été configuré au préalable pour se connecter à notre

serveur MQTT. Pour rappel, le topic plcs/test/sensors représente selon notre schéma qu'on a défini un API du nom de test et ce qu'il envoie ce sont les données de ces capteurs d'où le /sensors. Tout ce flux nous permet de tester sous différentes configurations les messages qu'on pourrait recevoir par MQTT en simulant l'envoi de données.

Maintenant la deuxième ligne de flux écoute en tout temps tout ce qui se passe sur le topic plcs/+/sensors. Ça tombe bien, au moment où l'on clique sur le bouton bleu du dessus, c'est exactement ce qui se passe. À la réception de ce message, le prochain nœud est un nœud fonction qui va le traiter.

Il se passe plein de chose dans le nœud fonction data processing. Mais avant de pouvoir continuer, il faut expliquer une petite subtilité. On parle depuis le début de flux, de message, de payload. La manière dont fonctionne Node-RED, c'est que ce sont des messages qui sont envoyés d'un nœud à l'autre. Le contenu de ce message qu'on va appeler payload est accessible dans le nœud fonction en tapant msg.payload. Mais rien ne nous empêche de créer un nouvel attribut. Dans notre cas, on l'a appelé statut et qui est accessible en tapant msg.statut. Vous allez comprendre bientôt pourquoi c'est utile et comment on s'en sert.

Donc dans le nœud fonction, premièrement, on vérifie que le message soit correctement formaté, car comme on va le voir dans la prochaine section, il est très important pour des raisons de véracités et aussi de performance d'avoir un bon format de données lorsqu'on veut les enregistrer dans InfluxDB. S'il y a une erreur qui est détectée, on rajoute à notre message la fameuse extension msg.statut.

Deuxièmement, après la vérification du format, on les formate pour construire le payload qui sera utilisé pour envoyer l'information dans la base de données. Par exemple l'un des traitements que nous faisons parfois, c'est qu'on reçoit des messages pour le topic des alertes et dans ce message des nombres. L'un des types de traitement que l'on fait c'est de transformer ces nombres dans son équivalent en bit et de l'entrer correctement dans la base de données, car le premier bit c'est celui de droite et non de gauche.

Maintenant on arrive au nœud switch et ce qu'elle regarde c'est le fameux msg.statut. En fonction du contenu de msg.statut, le message pourrait être envoyé sur le topic data/plcs/sensors ou bien data/plcs/error si dans le nœud, on a détecté une erreur. Donc ce nœud nous permet de filtrer et rediriger les messages dans la bonne catégorie.

4.7 - InfluxDB

InfluxDB est développé par InfluxData et c'est une base de données de séries chronologiques, c'est-à-dire que la base de données est optimisée pour stocker et faire des

requêtes avec comme argument principal un timestamp. C'est aussi une base de données NoSQL, c'est-à-dire qu'on peut travailler sur des données qui ne sont pas forcément structurées. Ce qui est parfait, car la compagnie peut encore évoluer sur sa manière d'acquérir les données. De plus, pour interagir avec la base de données, ils ont créé un langage de programmation qui s'appelle Flux qui est facile d'utilisation et très lisible.

Dans notre cas, on mesure avec des sondes des grandeurs tel que la pression, la turbidité, le pH, la température, etc. Mais ce qui rend ces informations importantes, ce qui leur donne du sens, c'est à quel moment ont les mesures. Prenez l'exemple de l'identité d'une personne, cette donnée n'a pas besoin de contexte de temps pour avoir de la valeur. Ce qui n'est pas notre cas. Savoir qu'il fait 27 °C et que le pH est de 8 ne rajoute pas de plus-values. Pour que notre donnée prenne tout son sens, il faut non seulement avoir cette notion temporelle, mais il faut aussi faire son suivi dans le temps (son évolution au cours des dernières minutes, dernières heures, derniers jours). C'est ça qui est important.

De plus, ChemBrains est amenée à évoluer rapidement sur sa manière de générer les données, il est impossible à l'heure actuelle de définir un schéma pour les données à l'épreuve du futur. Si jamais dans 1 mois, il faut changer la manière d'enregistrer les données qu'on reçoit, une base de données SQL classique ne nous donne pas facilement cette possibilité. NoSQL au contraire, nous offre la flexibilité de pouvoir adapter le schéma des données au gré de l'évolution de l'entreprise.

Par la suite, on va parler du format des données et de l'importance du formatage. On a dans Node-RED des nœuds fonctions qui formatent le message avant de les envoyer à la base de données. Ce format est appelé Line Protocol Syntax (LPS).

Figure 8: Schéma des données avec le protocole LPS

La première partie en bleue contient le nom de ce que l'on mesure en général, la seconde partie en orange contient les tags qui vont servir à filtrer les requêtes. La troisième partie contient les champs qui contiendront des valeurs, mais qui ne seront pas indexés et pour finir, la quatrième et dernière partie est le timestamp.

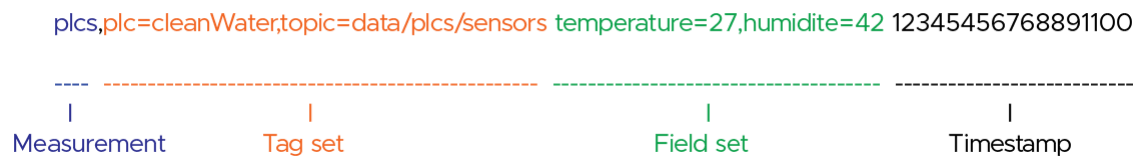

Figure 9: Exemple de message type reçu avec le protocole LPS

Un message type que l'on pourrait recevoir serait comme la figure ci-dessus: plcs suivi d'une virgule pour passer au tag, le nom du PLC qui est cleanWater puis le nom du topic qui est data/plcs/sensors. Suivi d'un espace pour annoncer les champs qui sont température et humidité. Suivi d'un dernier espace pour le timestamp.

Lorsqu'on effectue une recherche grâce à l'interface graphique ou bien via Flux, on fait essentiellement la même chose. On sélectionne le *bucket*, on coche ce que l'on veut mesurer, le nom du PLC, ainsi que le topic et on peut sélectionner les champs qui nous intéresse.

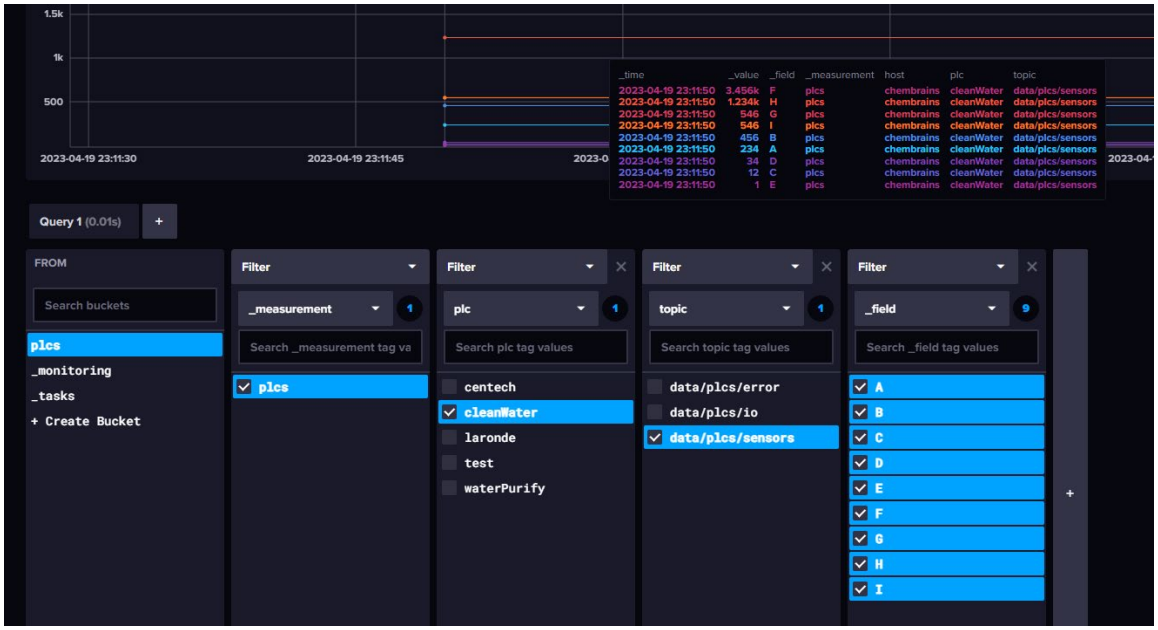

Figure 10: Exemple d'une requête de données sur InfluxDB

Maintenant qu'est ce qui peut arriver si le format n'est pas bien pensé. On peut tomber sur des problèmes de cardinalités. On mesure la cardinalité par le calcul suivant :

$$\text{Mesures} \times \text{CombinaisonTagsPossible}$$

Par exemple, si on utilise 1 mesure qui est plcs, qu'on a 4 noms d'APIs différents et trois topics différents possibles. La cardinalité de cette série serait de 1*4*3 = 12.

Mesures	Tags	
	Nom PLC	Topic
plcs	cleanWater	data/plcs/sensors
	reflectWater	data/plcs/io
	trustClean	data/plcs/alerts
	waterScape	
1	4	3

Figure 11: Exemple de calcul de cardinalité

Ce qui arrive parfois c'est de rajouter des mauvais tags ce qui a pour effet d'augmenter la cardinalité de notre série drastiquement. La cardinalité de cette série si l'on rajoutait comme tag Température serait 1*4*3*100 = 1200. Et encore, on a pris le cas où la température n'était pas continue et elle est comprise entre 1 et 100.

Mesures	Tags		
	Nom PLC	Topic	Temperature
plcs	cleanWater	data/plcs/sensors	1
	reflectWater	data/plcs/io	2
	trustClean	data/plcs/alerts	...
	waterScape		99
			100
1	4	3	100

Figure 12: Exemple de calcul de cardinalité avec un mauvais tag

Une cardinalité trop élevée impacte négativement les performances d'InfluxDB. Température ne devrait pas être dans les tags, mais plutôt dans la section des champs. Il n'y a pas de nombre miracle à ne pas dépasser, ce sont juste de bonnes pratiques à avoir.

4.8 - Telegraf

Telegraf est le module qu'utilise InfluxDB pour récupérer de l'information facilement. Nous utilisons son module MQTT pour configurer et pouvoir récupérer les données qui sont envoyées par Node-RED. Il existe des dizaines et des dizaines de modules qui permettent de configurer rapidement un agent dans le rôle seront de récupérer les données.

4.9 - Grafana

Grafana est un logiciel open source qui permet de créer des graphiques à partir de données qui seront récupérés sur InfluxDB. En faisant des requêtes avec Flux, on peut traiter et récupérer toutes les informations qu'on récolte.

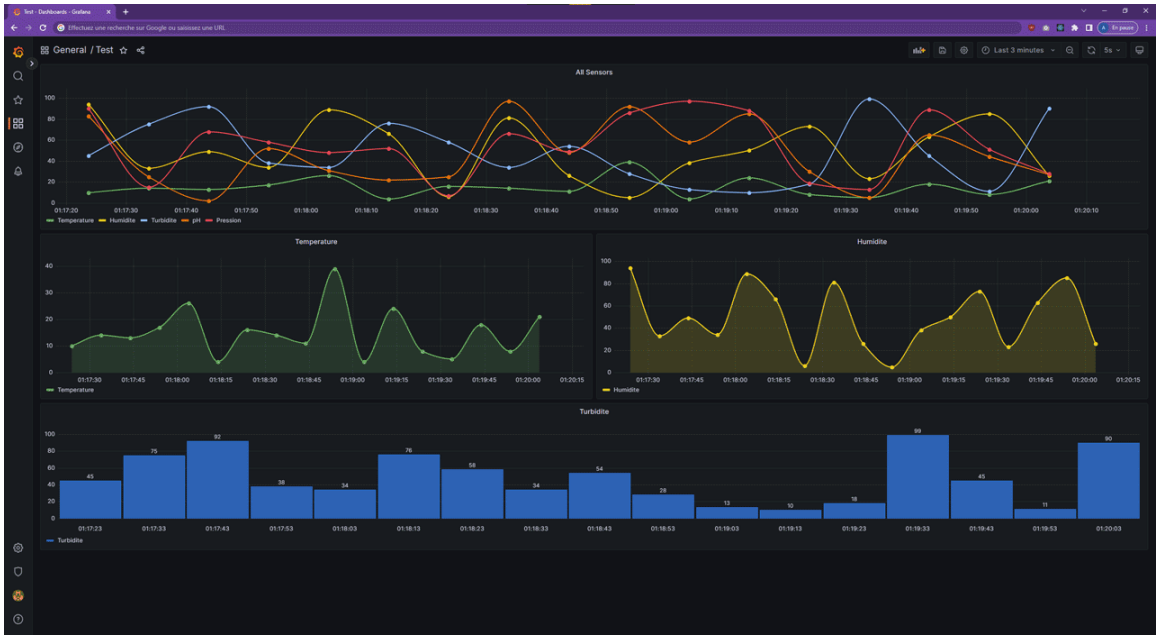

Figure 13: Exemple de tableau de bord généré avec Grafana

Mais Grafana ne nous sert pas qu'à l'affichage, cet outil est aussi utilisé pour générer des rapports sous le format CSV ainsi que l'envoi des alertes. Un topic est réservé aux alarmes et quand on en reçoit une, c'est une suite de 1 et 0. Donc dans Grafana, au moment où l'on reçoit un 1 dans une alerte, cela veut dire pour nous qu'il y a un problème. Une alerte est levée et un courriel ainsi qu'une notification sur Telegram sont envoyés dans la minute.

On utilise Grafana afin de répondre au besoin de ChemBrains qui nécessite que leurs systèmes envoient des alertes en fonction de leurs états.
les concepts clés pour comprendre les alertes dans Grafana :

1. Règles d'alertes : Les règles d'alertes sont des conditions définies par l'utilisateur qui doivent être remplies pour qu'une alerte soit déclenchée. Ces règles peuvent être basées sur des seuils de valeurs, des modèles de comportement, des anomalies, etc.
2. Notifications : Les notifications sont des actions qui sont déclenchées lorsqu'une alerte est déclenchée. Ces actions peuvent inclure l'envoi de notifications par e-mail, Slack, PagerDuty, etc.
3. État d'alerte : L'état d'alerte indique si une alerte est active ou inactive. Les alertes peuvent être activées ou désactivées manuellement ou automatiquement.
4. Évaluations : Les évaluations sont des cycles de vérification des conditions d'alerte qui sont effectués à des intervalles réguliers. Les évaluations sont utilisées pour surveiller les métriques et déclencher des alertes si nécessaire.

Chapitre V - Résultat final et travaux futurs

5.1 - Rétrospective sur la solution

Tout le système a été installé chez le client et il a commencé à la prendre en main. L'équipe est contente du résultat obtenu avec les restrictions du projet et du nombre de personnes travaillant dessus. Il y a eu un gros effort pour acquérir en début de session la connaissance nécessaire pour monter une solution qui répondrait à la problématique de ChemBrains, car aucun membre de l'équipe n'avait touché à une base de données NoSQL et encore moins une base de données de séries chronologiques ou monter un projet avec l'IoT au cœur.

5.2 - Recommandation pour le maintien du projet et pour le futur

Beaucoup de documentation a été écrite sur le projet dans le document « Documentation technique », mais tous les choix pris lors de ce projet n'ont pas été notés ou clairement explicités, il a fallu faire un compromis entre justifier tous les choix de design et avancer le projet assez rapidement pour que le client puisse s'en servir à la fin du projet. Par exemple, pourquoi on utilise 3 topics et pourquoi avons-nous choisi spécifiquement ce format pour le protocole MQTT.

La solution actuellement n'est pas parfaite, l'installation est sur un ordinateur qui se trouve au Centech et si jamais il y a des problèmes d'électricités ou d'internet, on ne reçoit plus de données et on ne peut plus recevoir d'alertes. De plus, ChemBrains a un nom de domaine qui pourrait être utilisé dans le futur à la place de Ngrok. Il y aurait donc plein d'avantages à mettre la solution dans le cloud pour des raisons de continuités de service et de sécurité.

Pour le maintien du projet, il faudrait continuer à documenter la solution, si jamais il y a de nouveaux cas d'études qui se présentent, il faudrait incrémenter sur le document « Documentation technique ».

Chapitre VI - Conclusion

En conclusion, le projet de ChemBrains a permis à l'équipe de toucher à une multitude de nouvelles technologies et solutions pour aboutir au résultat final. Pour résoudre la problématique du suivi à distance d'équipements de contrôle, on a mis en place une pile technologique qui permettrait d'y répondre efficacement. On utilise le protocole MQTT pour tous les avantages qu'il offre dans le domaine de l'IoT. Les données sont envoyées avec l'aide de Ngrok sur le courtier MQTT Mosquitto du serveur qui l'envoi à Node-RED pour le traitement et l'enrichissement. Par la suite, Node-RED republie ces données pour que InfluxDB les récupère avec l'aide de Telegraf. Grafana pendant ce temps, fait des requêtes à InfluxDB pour générer des rapports et en cas de problèmes, envoie des alertes sur des adresses courriel et un message texte sur Telegram.

Actuellement, le travail pourrait être poussé plus loin en passant la solution dans le cloud et en mettant la pile technologique dans un conteneur comme Docker. On pourrait aussi profiter du nom de domaine de ChemBrains pour faciliter le transport et mettre en place notre propre sécurité TLS.

References

Davies, R. (2020, April 17). *What is an Ubuntu LTS release?*
Récupéré sur Ubuntu: https://ubuntu.com/blog/what-is-an-ubuntu-lts-release

Documentation Mosquitto. (s.d.).
Récupéré sur mosquitto.org: https://mosquitto.org/documentation/

Get started with InfluxDB OSS 2.7. (s.d.).
Récupéré sur docs.influxdata.com: https://docs.influxdata.com/influxdb/v2.7/

Grafana documentation. (s.d.).
Récupéré sur grafana.com: https://grafana.com/docs/grafana/latest/

How do I install Guest Additions in a VirtualBox VM? (2011, 01 22).
Récupéré sur askubuntu.com: https://askubuntu.com/questions/22743/how-do-i-install-guest-additions-in-a-virtualbox-vm

MongoDB Documentation. (s.d.).
Récupéré sur mongodb.com: https://www.mongodb.com/docs/

Mongoose - Getting Started. (s.d.).
Récupéré sur mongoosejs.com: https://mongoosejs.com/docs/

NanoDano. (2016, 09 16). *Creating Systemd Service Files*.
Récupéré sur devdungeon.com: https://www.devdungeon.com/content/creating-systemd-servicefiles

ngrok Platform Overview. (s.d.).
Récupéré sur ngrok.com: https://ngrok.com/docs/

Node-RED. (s.d.). *Running on Raspberry Pi*.
Récupéré sur nodered.org: https://nodered.org/docs/getting-started/raspberrypi

Telegraf 1.9 documentation. (s.d.).
Récupéré sur docs.influxdata.com: https://docs.influxdata.com/telegraf/v1.9/