Mycelium 3.0

CHEREL Valentin - DERRIEN Thomas - LE GOASTELLER Maiwenn
LESSIRARD Leo - MAUGER Arthur - OUVRARD Pierre - POURCHER Pierrick

Encadrants : PARLAVANTZAS Nikos - PAROL-GUARINO Volodia

Avec la participation de :
MOUREAU Julien

2023-2024

Figure 1 – Mycélium : projet de suivi environnemental

Table des matières

1 Introduction

1.1 Contexte

Après plusieurs années d'attente, la ligne B du métro de Rennes a enfin été inaugurée en septembre 2022, offrant ainsi une nouvelle solution de transport à environ 85 000 voyageurs quotidiens. Cette initiative a non seulement simplifié la vie des Rennais, mais a également permis de diminuer les déplacements en voiture de 15% dans le centre-ville [1]. Néanmoins, les huit années de travaux n'ont pas été sans impact sur l'environnement, notamment avec l'abattage de 1200 arbres. Afin de compenser ces dommages et de préserver l'écosystème local, la municipalité de Rennes a pris l'engagement de lancer des projets de renaturation visant à aménager environ 35 hectares au sein de la ville[2].

C'est dans ce contexte que le projet Mycélium a vu le jour en 2021, portant un objectif clairement défini : avoir un suivi de la renaturation de La Croix Verte, un espace végétalisé situé au cœur du campus de Beaulieu. Une représentation graphique de cette zone est illustrée dans la figure 2. En étroite collaboration avec le laboratoire des Géosciences de Rennes et sous la coordination de Laurent LONGUEVERGNE, ce projet vise à équiper cette zone de capteurs intelligents pour collecter des données cruciales à propos de la réaction de l'écosystème aux changements environnementaux et aux contraintes climatiques. Plus globalement, ce travail s'inscrit dans le projet de recherche **TERRA FORMA**[7], qui a pour objectif de maintenir constamment à jour une évaluation de nos ressources naturelles, telles que le sol, l'eau et la biodiversité.

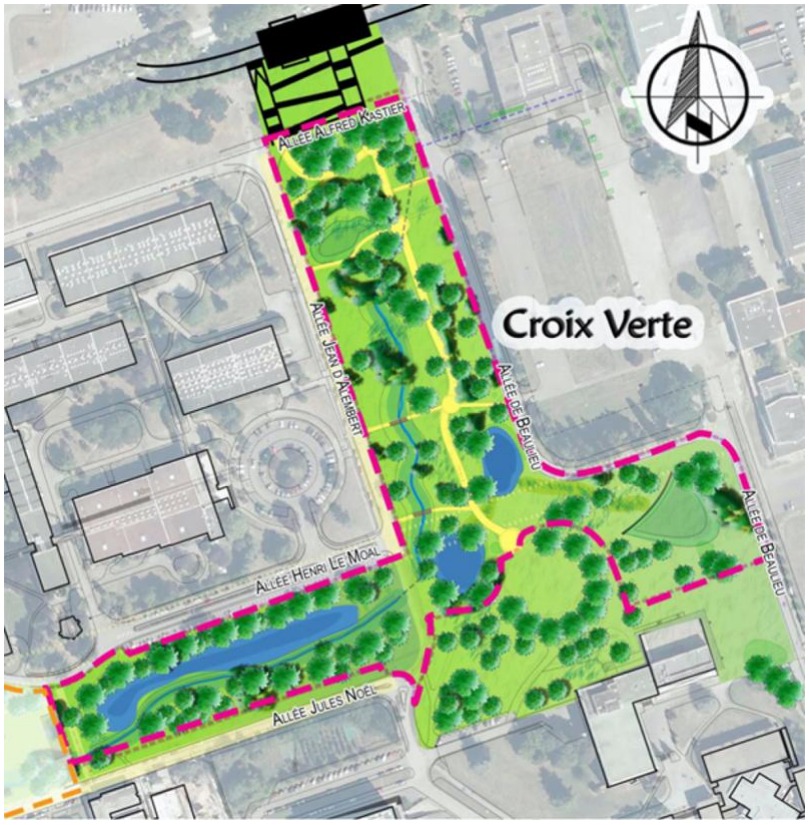

Figure 2 – Représentation cartographique de la Croix Verte

1.2 Versions antérieures

Mycélium 1.0, lancé en 2021, posa les bases d'un système de suivi basé sur un réseau de capteurs à faible consommation d'énergie, déployé sur le site de la Croix Verte. Cette première phase fut cruciale pour la création de l'architecture du projet et a conduit aux premiers essais.

Mycélium 2.0, avec l'objectif d'améliorer le système existant, a mis l'accent sur la gestion de données à grande échelle, l'amélioration des performances et apportait de nouvelles fonctionnalités.

1.3 Technologies utilisées

Le projet Mycélium repose sur une infrastructure combinant le Fog Computing et le Function as a Service (FaaS) afin d'optimiser la collecte, le traitement et la transmission des données provenant de plusieurs capteurs en temps réel.

Le **Fog**, ou **Fog Computing**, est un type d'infrastructure au sein duquel on trouve une couche intermédiaire entre les objets connectés et le cloud. À la différence du cloud computing traditionnel, qui centralise le stockage et le traitement des données sur des serveurs distants, le Fog délègue ces tâches à des nœuds plus proches du terrain. Ce pré-traitement local permet l'extraction des informations essentielles avant leur transmission vers le cloud, optimisant ainsi les calculs les plus urgents sur le Fog, tandis que les analyses plus approfondies sont déléguées au cloud.

Quant au **Function as a Service (FaaS)**, il s'agit d'un type de service basé sur le modèle Serverless, un modèle avec lequel les développeurs sont affranchis de la gestion de l'infrastructure sous-jacente. Ce modèle permet l'exécution de fonctions spécifiques en réponse à des événements. Pour implémenter cette architecture, nous avons opté pour le framework open-source OpenFaaS.

Combiner Fog Computing et FaaS a de nombreux avantages. Cela simplifie le développement, en permettant aux développeurs de se concentrer sur les fonctions directement plutôt que sur l'infrastructure. La latence ainsi que les coûts de transmissions sont réduits, grâce à la réalisation du pré-traitement local qui permet de réduire la quantité d'information à transmettre vers le cloud.

1.4 Objectifs de Mycélium 3.0

Cette évolution constante a conduit à la création de Mycélium 3.0 qui incarne la recherche d'optimisation, de durabilité et d'efficacité du système. Mycelium 3.0 est une refonte globale du projet, ayant pour objectif de simplifier l'architecture et de faciliter la prise en main ainsi que la maintenance du projet.

Dans un premier temps,la section 2.1 présente l'état de l'art lié au projet. Dans un second temps, la section 2.2 se penche sur les différentes fonctionnalités implémentées dans les versions antérieures du projet Mycélium. Ensuite, dans la section 3, l'architecture générale du projet est présentée. Enfin, les spécifications générales et fonctionnelles sont exposées dans la section 4.

2 Pré-Étude

2.1 État de l'art (Projets similaires)

Cette idée de suivre et d'étudier des données environnementales grâce à un réseau de capteurs est assez répandue. Il existe alors un grand nombre de projets ayant un but similaire au nôtre. Ces projets sont tous aussi divers que les différents environnements à étudier sont nombreux. Nous allons alors voir quelques-uns de ces projets, desquels le nôtre s'inspire.

2.1.1 Inspirations principales

Tout d'abord, il y a deux projets dont nous nous inspirons fortement car ils utilisent en grande partie les mêmes technologies que nous. Il s'agit de LivingFog Platform et ConnecSens. Des descriptions plus détaillées de ces deux projets peuvent être trouvées dans les rapports de pré-études des groupes précédents.

Le premier projet, **LivingFog Platform**[3], est une plateforme de Fog computing, qui utilise également un cluster de Raspberry Pi[4]. L'architecture de ce projet est illustrée dans la figure 3. Il est implanté en Espagne, à Valence, qui est donc plutôt un environnement portuaire. Les données qui y sont mesurées diffèrent donc de par leur nature puisqu'il y est question d'étudier principalement la qualité et la consommation de l'eau, mais aussi les vagues, le trafic, la météo, etc... Son but est principalement de démontrer qu'il est possible d'utiliser le Fog pour cet usage, plutôt qu'une architecture conventionnelle basée sur le cloud.

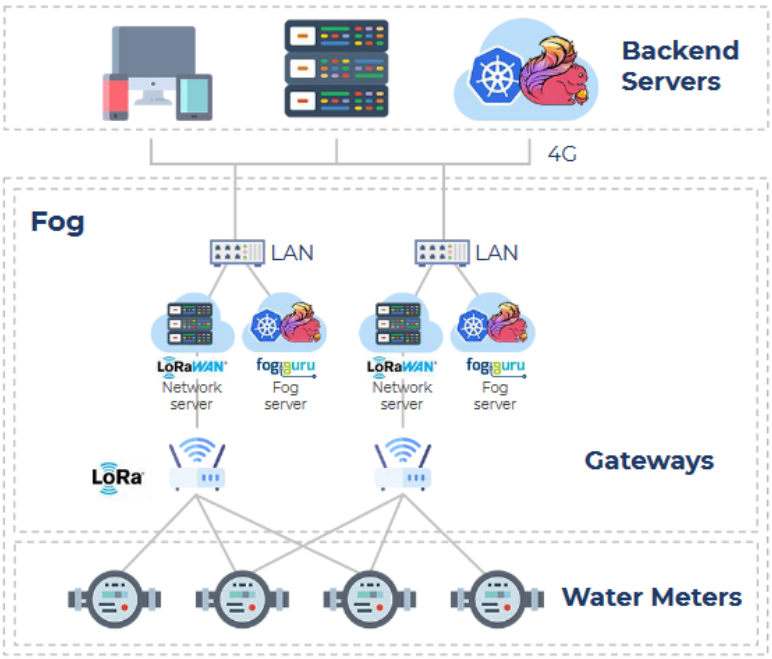

FIGURE 3 – Architecture de LivingFog pour l'étude de la consommation d'eau

Mycelium se distingue donc de ce projet par son contexte d'implantation, car dans notre cas, nous souhaitons notamment obtenir des données sur la météorologie et l'écosystème en général, sur le site de la Croix Verte.

Pour ce qui est du deuxième, notre projet s'inscrit directement dans le cadre de celui-ci. Il se nomme **ConnecSenS**[5], et son but est de réaliser des observations environnementales de précision en région Auvergne-Rhône-Alpes, notamment au niveau de la disponibilité des ressources en eau et de l'étude des sols. Son but principal est l'étude des changements climatiques et de leur impact sur les écosystèmes. Ce projet a en commun avec le nôtre son déploiement en milieu contraint. Il est donc important de raisonner l'énergie qu'utilise notre plateforme. Aussi, le nœud **SoLo**[6] que nous utilisons a été développé pour ce projet par des chercheurs de l'Université de Clermont-Ferrand. Vous pouvez trouver dans la figure 4 un schéma de l'architecture de ce projet.

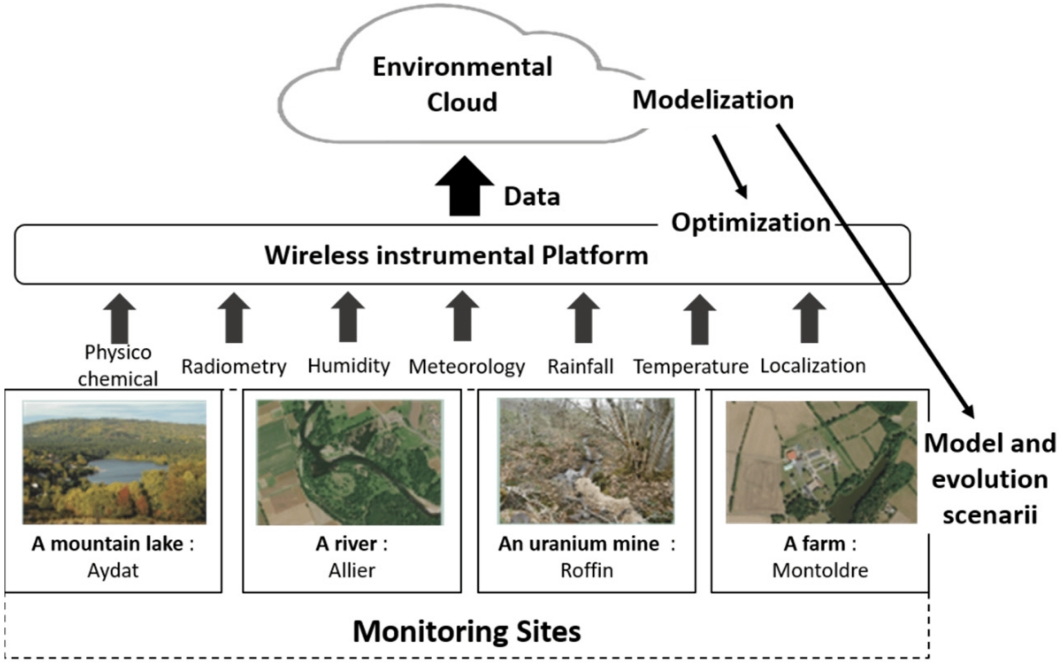

FIGURE 4 – Architecture de ConnecSenS

2.1.2 Autres projets de surveillance environnementale

Ensuite, nous pouvons parler d'autres projets, qui sont certes différents par l'architecture ou les technologies utilisées, mais qui poursuivent des buts similaires au nôtre. Il en existe de nombreux,

Nous pouvons déjà constater que beaucoup d'entre eux visent à étudier l'évolution de l'environnement aux changements climatiques, et son impact sur la faune et la flore, par l'intermédiaire de capteurs qui mesurent et véhiculent les données récoltées. Il s'agit alors d'une démarche "Data driven from IoT sensors". C'est également dans ce but que Mycelium s'inscrit.

Terra Forma[7] : Projet CNRS visant le déploiement d'un réseau de capteurs pour étudier les changements environnementaux dans des territoires témoins.

Econect[8] : Autre projet CNRS comportant des capteurs innovants et connectés pour étudier et surveiller la réponse des espèces sentinelles face aux changements environnementaux.

Il existe également de nombreux projets qui visent à comprendre plus précisément le fonctionnement de certains phénomènes naturels.

Volcano monitoring sensor network[9] : Benchmarking de l'utilisation du Fog en prenant l'exemple de l'analyse de relevés volcaniques.

Il existe également des projets qui réalisent de la surveillance environnementale dans le but de raisonner et rendre plus efficace l'usage de nos ressources en détectant des comportements anormaux afin d'éviter du gaspillage.

Potable Water Management[10] : Monitoring de la consommation d'eau à Valence dans le but de détecter des fuites ou tout autre problème menant à une consommation d'eau anormale. Ce projet est un cas d'usage de LivingFog Platform.

Ainsi, de nombreux projets similaires au nôtre existent. Ils s'inscrivent le plus souvent dans une démarche verte, qui semble être de plus en plus d'actualité. L'émergence croissante de ces initiatives souligne leur pertinence croissante.

2.2 Précédents projets Mycélium

Étant donné que nous reprenons un projet déjà existant, nous pouvons exposer les différentes fonctionnalités qui ont été implémentées par les deux groupes précédents.

2.2.1 Mycélium 1.0

Le premier groupe s'est concentré sur la création de **scénarios**, qui sont des événements pouvant être observés à l'aide des capteurs. Dans cette première version du projet, sept ont été implémentés.

- Le scénario “Luminosité anormale : Intempéries” qui vérifie que la luminosité diminue lorsqu'il pleut.
- Le scénario “Luminosité anormale selon l'heure” qui vérifie que la luminosité est cohérente avec l'heure.
- Le scénario "Neige", qui regarde si le pluviomètre se remplit après que la température est descendue en dessous de 0°C.
- Le scénario "Températures extrêmes", qui notifie lorsque les températures sont extrêmement basses ou extrêmement élevées.
- Le scénario "Températures anormales selon l'historique de températures", qui vérifie que les températures n'augmentent pas ou ne diminuent pas trop vite.
- Le scénario "Vol et perte de matériel", qui notifie s'il y a un mouvement rapide du capteur, qui pourrait signifier un vol.
- Le scénario "Capteurs endommagés", qui notifie si les données mesurées sont aberrantes.

Les membres du groupe ont utilisé les fonctionnalités du capteur à leur disposition pour prendre des mesures à une certaine fréquence, les enregistrer, et envoyer des données aux capteurs. De plus, ils ont ajouté trois nouvelles fonctions pour ces scénarios. Deux de ces fonctions sont destinées à traiter certains des scénarios mentionnés précédemment, en déclenchant des actions lorsque certains seuils sont atteints. La troisième fonction sert à récupérer des informations au démarrage des capteurs.

Ils ont également déployé l'infrastructure du cluster Mycelium en utilisant plusieurs Raspberry Pi pour traiter les données envoyées par le nœud SoLo via le protocole LoRaWAN. Ils ont également créé des fonctions OpenFaaS[**?**] pour opérer directement sur ces données. Enfin, ils ont mis en place des fonctionnalités liées à la visualisation et au traitement des données, notamment l'envoi de notifications en cas d'événements climatiques, des tests de qualité, la comparaison avec l'historique des données et le stockage dans une base de données InfluxDB[12].

2.2.2 Mycélium 2.0

Le deuxième groupe a continué le projet, en consacrant d'abord un temps à l'appropriation du projet. Ils ont créé trois nouveaux scénarios :

- Le scénario “Normales saisonnières” qui notifie lorsque les températures détectées sortent des normales saisonnières.
- Le scénario “Orage” qui notifie la présence d'épisodes orageux.
- Le scénario “FoxyFind” qui notifie la présence d'animaux devant une caméra.

Les ressources du cluster disponibles étant moindres et rapidement consommées lors du fonctionnement, ils ont entrepris l'ajout d'un lien avec le cloud dans le but de déléguer une partie des calculs afin d'éviter la surcharge. Cependant, seules les fonctionnalités liées au scénario FoxyFind sont fonctionnelles sur le cloud, ce qui n'est pas totalement satisfaisant.

3 Architecture logicielle et matérielle existante

Dans cette partie, nous allons expliquer en quoi consiste l'architecture globale du projet de l'année précédente. Nous allons nous concentrer sur celle-ci car nous souhaitons partir de cette architecture pour l'améliorer. Cela permettra par la suite de montrer les modifications qu'on souhaiterait y apporter. Il y a plusieurs parties importantes dans l'architecture, nous évoquerons d'abord les capteurs et l'envoi de leurs enregistrements via le réseau LoRaWAN dans la section 3.1, puis la réception et le traitement des données sur un cluster de Raspberry Pi dans la section 3.2.

3.1 Capture et envoi des données via le réseau LoRaWAN

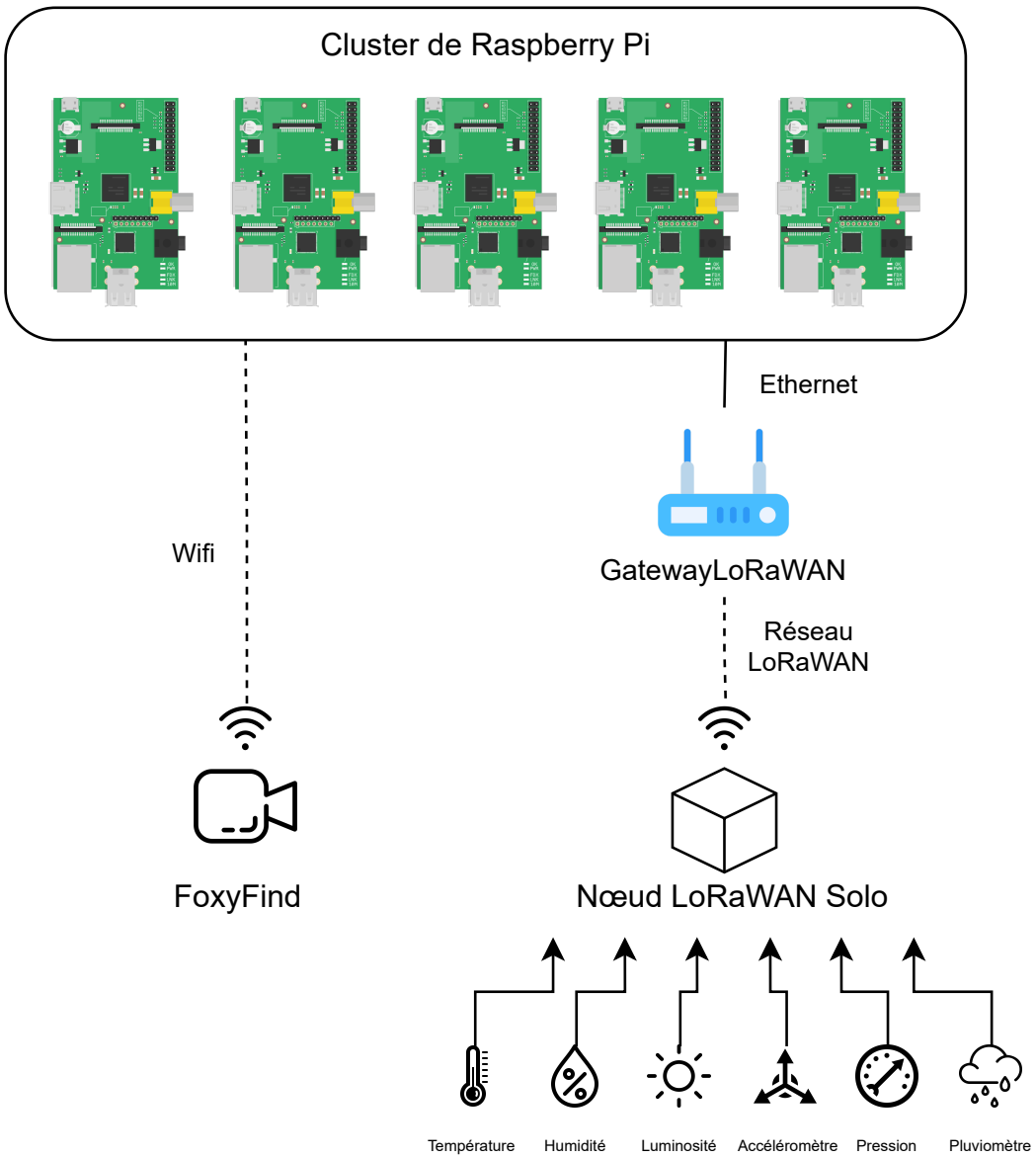

FIGURE 5 – Schéma de l'envoi des données entre les capteurs et le cluster

L'architecture de Mycélium intègre un réseau de capteurs déployés sur la Croix-Verte, conçu pour collecter diverses données environnementales. Les capteurs sont des **nœuds SoLo**. Ce sont des appareils autonomes qui communiquent via la technologie LoRaWAN sans faire partie d'un réseau de nœuds interconnectés. La méthode de communication utilisée est illustrée dans la figure 5. Le réseau LoRaWAN assure la transmission fiable et la très faible consommation des données provenant des capteurs sur le terrain.

D'un point de vue matériel, c'est un boitier étanche contenant plusieurs capteurs (température, humidité, luminosité, accéléromètre, pression, pluviomètre). Ces capteurs sont paramétrables à l'aide de fichiers au format JSON. On peut configurer la fréquence d'envoi, la fréquence de prise de mesures, le réseau... Dans le cadre du projet, ces nœuds sont chargés de collecter des données environnementales puis de les envoyer vers une gateway installée dans les locaux de l'INSA.

Une **gateway LoRaWAN** est un dispositif qui agit comme une passerelle de communication entre les nœuds LoRaWAN et un réseau IP. Elle sert de relais, centralisant les données provenant des capteurs. Elle reçoit les données depuis les nœuds Solo et les renvoie en ethernet vers le cluster de Raspberry Pi.

L'utilisation de LoRaWAN présente un avantage significatif en matière d'économie d'énergie, ce qui en fait un choix approprié pour cette application. Cet avantage est dû au fait que LoRaWAN utilise une modulation de basse puissance pour la transmission des données. C'est une manière intelligente de transmettre des informations sans fil, en utilisant une technologie appelée la modulation LoRa[13]. Plutôt que de transmettre un signal à une fréquence fixe, la modulation LoRa fait varier la fréquence du signal au fil du temps. Cela crée une séquence de changements de fréquence appelée chirp. Les données sont ensuite encodées en utilisant ces chirps. Ceux-ci ont une caractéristique spéciale qui permet une portée plus étendue avec moins d'intermédiaires physiques, économisant ainsi de l'énergie.

Les données transitent ensuite depuis la gateway vers le cluster de traitement via une connexion Ethernet. Un autre capteur est aussi présent dans le projet. Il se nomme FoxyFind et a été ajouté l'an passé. Il permet de détecter le passage d'animaux devant une caméra. Ce capteur communique directement avec le cluster sans passer par la gateway en utilisant le réseau Wi-Fi.

3.2 Réception des données sur le cluster de Raspberry Pi

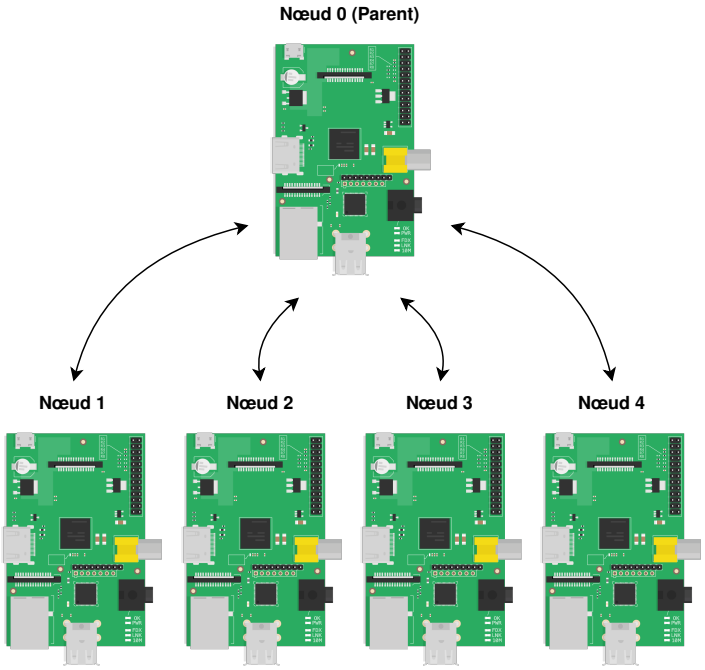

FIGURE 6 – Schéma général de l'organisation des Raspberry Pi au sein du cluster

Le cœur du système repose au sein d'un cluster de 5 Raspberry Pi situé dans les locaux de l'INSA. Un schéma de ce cluster est proposé dans la figure 6. L'objectif est de faire en sorte que l'on puisse utiliser ce cluster de façon simple, comme si on utilisait seulement une machine. Pour cela, on utilise Kubernetes.

Kubernetes[14] est une plateforme open-source de gestion de conteneurs. Un conteneur est un package logiciel qui regroupe le code d'une application avec les fichiers de configuration, les bibliothèques et les dépendances requises pour que l'application puisse s'exécuter. Cela permet de pouvoir s'affranchir des dysfonctionnements potentiels qui peuvent survenir lorsque l'on passe d'un environnement à l'autre. Ces problèmes surviennent généralement en raison de différences dans la configuration requise pour la bibliothèque sous-jacente et les autres dépendances. Kubernetes permet de déployer, de mettre à l'échelle et de gérer ces conteneurs. La plateforme organise automatiquement la distribution des conteneurs sur différents nœuds, assurant ainsi une répartition équilibrée de la charge de travail. En cas de défaillance d'un nœud, Kubernetes détecte et remplace automatiquement les conteneurs défaillants sur d'autres nœuds, garantissant ainsi une haute disponibilité. La gestion automatisée des ressources par Kubernetes ajuste dynamiquement l'allocation de ressources aux conteneurs en fonction de leurs besoins, optimisant ainsi l'utilisation des capacités disponibles. Un des Raspberry Pi sera vue comme un parent qui distribue les tâches aux nœuds enfants. Les applications essentielles sont alors déployées sur ce cluster, pour permettre la réception, le traitement, et la sauvegarde des données provenant des capteurs.

On utilise tout d'abord **Chirpstack**[15]. C'est un logiciel open-source pour la gestion des données des appareils utilisant LoRaWAN. Il est utilisé pour recevoir,

décoder et stocker les données transmises par les capteurs LoRaWAN. Ces données sont ensuite acheminées au sein du cluster via le protocole MQTT pour être traitées.

MQTT[16] (Message Queuing Telemetry Transport) est un protocole de communication léger pour l'IoT. Il permet aux appareils de publier et de s'abonner à des messages sur un réseau, facilitant la transmission de données en temps réel. Pour cela, on utilise des brokers. Un **broker** MQTT est un serveur ou une entité logicielle qui gère la communication avec un système publishers / subscriber. Il reçoit les messages publiés par certains appareils et les distribue aux appareils qui sont abonnés aux mêmes sujets (topics).

Donc dans le cluster, Chripstack va réceptionner les données provenant de la gateway et va ensuite les transférer dans le cluster grâce à un MQTT broker (voir 7). C'est ainsi que le cluster réceptionne les données.

3.3 Traitement des données

Comme expliqué précédemment dans la partie "Technologies utilisées", nous utilisons le FaaS pour le traitement de nos données, et plus précisément la plateforme OpenFaaS.

Les fonctions permettent d'effectuer divers traitements sur les données reçues et d'effectuer un retour sur ces traitements. Ces fonctions constituent le cœur du système de traitement des données. Elles ont deux rôles majeurs :

1. **Analyse des données :** Les fonctions OpenFaaS sont en mesure d'effectuer une analyse en temps réel des données capturées par les capteurs LoRaWAN. Par exemple, elles peuvent détecter des changements soudains dans les conditions météorologiques, comme des variations importantes de la température ou de la pression atmosphérique. De telles détections peuvent déclencher des alertes ou des mesures spécifiques, par exemple l'envoi de messages sur l'application de messagerie instantanée Discord.
2. **Stockage des données :** Les fonctions OpenFaaS sont aussi responsables du stockage des données traitées. Elles peuvent les archiver dans une base de données puisqu'elles sont sur le même cluster. Nous utiliserons pour cela InfluxDB, un système de gestion de base de données orienté séries temporelles. Cela permet de conserver un historique des données, qui peut être précieux pour l'analyse rétrospective ou la recherche de tendances sur une période prolongée. Pour visualiser les données, le logiciel de visualisation de données **Grafana**[17] est utilisé.

Si besoin, il existe un système de fichier distribué entre toutes les différentes Rasbperry Pi, ce qui permet de partager des données peu importe où la fonction est en train de s'exécuter. Cela est rendu possible via **GlusterFS**[18]

Si les capacités de calculs ne sont pas suffisantes, les données sont envoyées sur le cloud, afin de ne pas surcharger le cluster. Concrètement, on peut envoyer les données vers les serveurs de l'INSA. Pour cela, on utilise ce que l'on appelle un VPS (Virtual Private Server) C'est une machine virtuelle, créée sur un serveur physique et employant ses ressources pour offrir à ses utilisateurs les mêmes fonctionnalités qu'un serveur dédié. Le VPS exécute des fonctions OpenFaaS.

3.4 Schéma bilan

Tous les éléments explicités précédemment constituent donc les éléments du projet Mycelium 2.0. Ces différents éléments communiquent dans une architecture complexe. Cette architecture est explicitée dans la figure 7.

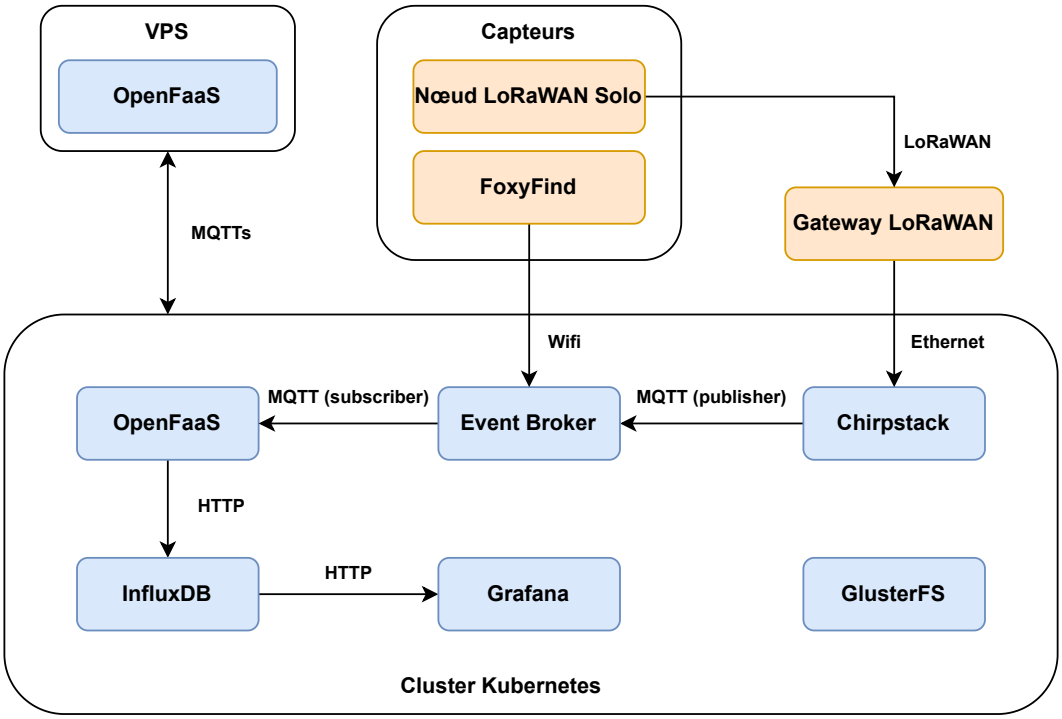

FIGURE 7 – Schéma de l'architecture globale du projet Mycelium 2.0

4 Spécifications générales et fonctionnelles

En utilisant différents éléments du projet Mycélium 2.0 (notamment des parties de l'architecture), nous allons réaliser notre projet en suivant plusieurs objectifs :

- Faciliter l'utilisation et l'installation de l'environnement pour que le développeur / utilisateur puisse s'en servir de manière claire et efficace (voir section 4.1)
- Simplifier le système afin de gagner en performance, en maintenabilité et réduire la consommation de ressources (voir section 4.2)
- Refaire le VPS qui n'existe plus, en l'adaptant selon nos besoins en utilisant les mêmes technologies que dans le cluster de Raspberry Pi (voir section 4.3)
- Autres améliorations que nous envisageons (voir section 4.4)

4.1 Facilitation de la prise en main du projet

4.1.1 Réalisation d'un guide

Nous nous sommes rapidement rendu compte que le projet était plutôt difficile à reprendre en main du fait de son architecture complexe ainsi que de certains choix d'implémentations et de technologies pris auparavant pouvant être améliorés.

Afin de jauger concrètement la difficulté de déploiement des différents éléments de l'architecture du projet, nous avons décidé de commencer directement à essayer de mettre en place et de faire fonctionner ceux-ci. Pour cela, nous avons pu nous aider de documents techniques créés par les deux anciens groupes dans les différents projets GitLab. De nombreux repository GitLab contiennent les différents éléments du projet, et ce n'est pas toujours facile de savoir où trouver les informations. En utilisant celles-ci, nous avons eu de nombreux problèmes techniques et n'avons pas réussi à ce jour à faire fonctionner les différents éléments, la documentation étant parfois trop peu détaillée pour nous aider.

Au niveau du cluster, nous avons décidé de ne pas directement l'utiliser pour éviter d'éventuels problèmes techniques liés au hardware et se focaliser sur l'utilisation des technologies utiles au projet. Pour ce faire, nous avons donc décidé de recréer une machine virtuelle pour émuler le comportement du cluster. Sur cette machine virtuelle, nous avons pu installer et configurer tous les logiciels nécessaires afin d'appeler des fonctions OpenFaaS. Nous avons aussi essayé de prendre en main le cluster directement en se connectant dessus directement en SSH. Cela a été compliqué car nous n'avions pas l'autorisation. Il a fallu faire un ticket à la DSI pour autoriser nos adresses MAC à se connecter.

Au niveau de la gateway LoRaWAN, nous avons également eu des difficultés à l'utiliser pour capter les signaux venant de l'extérieur. Ne disposant pas des mots de passes de connexion, nous avons dû la réinitialiser aux paramètres d'usine et tout reconfigurer, ce qui est assez complexe.

C'est en voyant toutes ces difficultés que nous nous sommes mis en tête de créer un guide pour le projet qui permettra d'aider les prochaines équipes et OSUR à prendre en main le projet. Ce guide contiendra :

- une liste de l'équipement matériel
- une liste des technologies utilisées
- un schéma de synthèse de l'architecture explicitant les liens entre les différentes technologies utilisées et leur place pour le fonctionnement du projet
- un guide d'installation expliquant comment et dans quelles versions installer les différentes technologies à utiliser
- un guide d'utilisation expliquant comment se servir du matériel et des technologies à notre disposition

Les équipes s'emparant du projet pourront ainsi, que ça soit pour l'améliorer ou pour juste pour l'utiliser, le comprendre et le déployer rapidement et modifier le guide à leur guise afin de garantir une bonne transmission de savoir.

4.1.2 Mise en place d'une Machine Virtuelle (VM)

Comme dit précédemment, l'élément central du projet est un cluster de 5 Raspberry Pi. Ce cluster se situe dans le département informatique de l'INSA de Rennes et pour s'y connecter, il faut réussir à se connecter en SSH, ce qui n'est pas des plus pratiques (il faut que nos adresses MAC soient autorisés par la DSI, que le cluster soit branché, etc...).

C'est donc pour des raisons pratiques que nous avons décidé de construire un cluster de machine virtuelle. Nous aurons donc plusieurs machines virtuelles indépendantes (le parent et un enfant pour commencer) qui pourront communiquer comme le ferait les Raspberry. Ces machines virtuelles reprennent le système d'exploitation et les logiciels des machines du cluster afin d'avoir un environnement de travail que chacun peut avoir sur son ordinateur personnel et identique au cluster réel.

Ce cluster de machine virtuelle va également nous permettre de réaliser les modifications de l'architecture pour simplifier le système sans risquer de faire régresser la version actuelle du cluster.

Pour réaliser ce cluster, nous avons décidé d'utiliser le logiciel de virtualisation open source QEMU[19] (Quick EMUlator) qui permet donc d'exécuter des machines virtuelles sur un système hôte. QEMU prend en charge la virtualisation matérielle et logicielle, ce qui va nous permettre d'obtenir des performances proches du système original. Cela nous permettra par ailleurs un développement accéléré, car on pourra avoir de meilleures performances que le support original (dans notre projet, les Raspberry ont 1 Go de RAM chacun, ce qui ne permet pas une grande rapidité d'exécution des programmes).

4.2 Simplification du système

Actuellement, le projet existant consomme énormément de ressources, si bien que pour le projet Mycélium2.0, l'équipe précédente a dû rajouter un transfert de données vers le cloud afin d'augmenter la puissance de calcul du cluster en cas de débordement d'actions à réaliser. Un de nos objectifs est donc de simplifier le système (architecture du cluster et fonctions) afin que cela nécessite moins de ressources, dans le but de pouvoir diminuer au maximum l'utilisation du cloud. Cela permettrait de s'inscrire dans une dynamique plus écologique.

Pour réaliser cet objectif, nous avons différentes pistes :

4.2.1 Remplacement de MQTT par NATS

Pour que Chirpstack envoie les données qu'il a reçues à OpenFaaS, MQTT était utilisé. On souhaite remplacer MQTT par NATS[20] (Nanoseconds.io Asynchronous Transport System). NATS est un système de messagerie open source conçu pour permettre la communication entre des applications distribuées. Il a été développé pour offrir une messagerie légère, rapide et fiable, adaptée aux architectures de microservices et aux systèmes distribués.

On souhaite effectuer ce remplacement car NATS est plus performant que MQTT en raison de sa conception légère. En effet, NATS est optimisé pour des débits de messages avec une latence minimale, ce qui correspond à notre projet car on souhaite que OpenFaaS fournisse le plus rapidement possible les données afin de les traiter.

De plus, on souhaite uniquement transférer des messages de Chripstack à OpenFaaS, donc tout l'écosystème de MQTT permettant le traitement des données ne nous sera pas utile. Remplacer MQTT par NATS pour le transfert de données nous permettra de diminuer la complexité et d'augmenter les performances. Tout cela est valable également pour la communication entre deux fonctions OpenFaaS, qui se faisait là aussi par MQTT et que l'on fera à présent avec NATS.

4.2.2 Remplacement de Kubernetes par K3S

Dans l'ancien projet, il avait été choisi d'utiliser Kubernetes (K8S). Nous allons plutôt utiliser K3S[21] qui est largement plus léger pour des fonctionnalités identiques dans le cadre de notre projet. Cela permettra d'alléger la taille de l'architecture sur le cluster. K3S comporte de nombreux autres avantages par rapport à K8S. Ces avantages sont listés dans la figure 1

Caractéristiques	K3S	Kubernetes (K8S)
Installation	Installation simplifiée et rapide, adaptée aux environnements avec des ressources limitées.	Installation plus complexe, nécessitant plus de ressources et de configuration.
Configuration	Configuration par défaut optimisée pour une utilisation simplifiée, ce qui réduit la complexité.	Flexibilité accrue, mais nécessite une configuration plus détaillée pour s'adapter aux besoins spécifiques.
Gestion des nœuds	Gestion simplifiée, avec une approche plus automatisée pour rejoindre et quitter le cluster.	Gestion plus avancée des nœuds, mais cela peut être plus complexe à mettre en œuvre et à maintenir.
Utilisation des ressources	Efficace en termes d'utilisation des ressources, adapté aux environnements avec des contraintes.	Consomme davantage de ressources en raison de sa conception plus robuste et de sa gestion étendue.

TABLE 1 – Comparaison entre K3S et Kubernetes (K8S)

4.2.3 Remplacement des fonctions Python en fonctions Go

Les fonctions OpenFaaS sont actuellement écrites en Python. Nous voudrions accélérer la vitesse d'exécution des fonctions et alléger le système. Pour cela, nous voulons remplacer le langage utilisé pour aller vers du **Go**[22].

Go est souvent plus rapide que Python pour les opérations de bas niveau. Le code produit est également plus facilement maintenable, du fait du typage et des erreurs qui peuvent être détectées directement à la compilation, là où ce n'est pas possible avec Python sans utiliser de bibliothèques tierces. De plus, Go consomme généralement moins de mémoire que Python, car Python a besoin d'une runtime pour inférer les types puisque c'est un langage dynamique. Grâce à l'usage d'une architecture serveless nous limitons le lancement de cette runtime et son sa consommation. Mais puisque nous cherchons à réduire autant que possible l'usage des ressources, il est approprié d'éliminer ce problème en utilisant un langage typé statiquement. Ce langage possède ces différents avantages sans pour autant être plus dur à maîtriser. C'est un avantage sur un appareil tel que le cluster dont nous disposons, car il est souhaitable de réduire au maximum le temps d'exécution et l'utilisation de mémoire afin d'éviter une potentielle surcharge. De plus, le langage Go compile en un binaire autonome, facilitant le déploiement et l'exécution sur différentes plates-formes, y compris les systèmes embarqués comme les Raspberry Pi.

4.2.4 Remplacement des notifications Discord par un flux RSS

Nous pensons à supprimer le robot Discord qui envoie des notifications à chaque événement important pour plusieurs raisons. En premier lieu, nous pensons qu'il n'est pas souhaitable de forcer les utilisateurs à utiliser un réseau social pour utiliser le projet. De plus, cela permettra de simplifier le système de ce côté afin d'éviter de devoir gérer la communication avec un site extérieur au projet. Cela permettra de se concentrer sur le déploiement de l'architecture globale.

Pour notifier l'utilisateur, on souhaiterait plutôt mettre en place un flux RSS qui permettrait de visualiser ces notifications plus simplement. Un flux RSS (Really Simple Syndication) est un format de données utilisé pour diffuser des informations mises à jour régulièrement. Techniquement, un flux RSS est un fichier XML structuré. Les utilisateurs peuvent s'abonner à ce flux à l'aide d'un lecteur RSS ou d'une application, ce qui leur permet de recevoir automatiquement les nouvelles mises à jour dès qu'elles sont publiées.

Ce système est préférable pour plusieurs raisons. Cela permet une facilité de suivi, car les flux RSS permettent une surveillance simplifiée des événements. Les utilisateurs peuvent s'abonner et recevoir automatiquement les mises à jour dans leur lecteur RSS favori, sans devoir être connectés à Discord. Cela peut aussi se faire sur plusieurs plateformes différentes. Ce changement donne aussi une plus grande liberté de personnalisation, car les utilisateurs peuvent choisir de recevoir uniquement les notifications qui les intéressent en s'abonnant aux flux RSS pertinents, offrant ainsi une expérience plus personnalisée par rapport aux notifications Discord plus génériques.

4.3 Amélioration du cloud

L'équipe précédente a eu pour objectif de mettre en place une interface Cloud. Pour cela, ils ont utilisé un VPS hébergé par la DSI de l'INSA Rennes afin de prendre en charge les calculs effectués par le capteur en cas de sur-utilisation de celui-ci. S'ils ont effectivement mis en place ledit VPS, il se trouve que son fonctionnement n'est aujourd'hui pas satisfaisant, car il ne fonctionnait réellement que pour FoxyFind. Nous avons donc décidé de réorganiser l'organisation du cloud, afin que cela fonctionne plus efficacement, en ayant également pour objectif de réduire la consommation énergétique.

Le but est de s'en servir pour soulager le cluster de manière générale. Cela permettrait ainsi de déplacer l'exécution des fonctions sur le cloud lorsque cela est nécessaire. On va pour cela uniquement utiliser le VPS lorsque le cluster est saturé. Actuellement, les fonctions sont renvoyées vers le cloud dès que le cluster dépasse 70% d'utilisation du CPU. Ce n'est peut-être pas forcément optimal. Nous essayerons de voir s'il est possible d'optimiser ce système afin d'effectuer le plus de calcul possible directement sur le cluster.

Pour certaines fonctions trop gourmandes, l'exécution sera effectuée par défaut sur le cloud. C'est le cas pour les fonctions liées à FoxyFind. Cela permettra d'éviter de saturer le cluster avec des traitements bien trop complexes.

Sur ce VPS, les données seront stockées temporairement dans une base de données SQLite ou InfluxDB.

1. **SQLite**[23] permettrait une mise en place rapide, ainsi qu'utilisation des ressources et une complexité moindre. SQLite est bien adapté aux petits projets avec des besoins de stockage limités. Cependant, si une plus grande capacité de stockage est nécessaire, nous pourrions basculer vers une autre base de donnée SQL plus adapté à de plus grands volumes.
2. **InfluxDB** propose une meilleure gestion des données temporelles et une meilleure performance. C'est aussi une solution plus simple car c'est la base de donnée que l'on souhaite utiliser dans le cluster.

Nous partons plutôt vers InfluxDB car cette base de données est conçue pour gérer les données temporelles, et c'est précisément ce que l'on souhaite stocker. En fin de compte, ce choix n'est pas si déterminant, car la base de données choisie n'impacte pas les performances du cluster.

Au niveau de l'envoi des données, nous essayerons de diminuer au maximum le nombre de données envoyées sur le cloud. Dans cette optique, les données seront tout d'abord agrégées sur le cluster, afin d'envoyer seulement les données nécessaires. Par exemple, sur une journée, il n'est pas forcément nécessaire d'envoyer la température mesurée toutes les 10 minutes. Dans cet exemple, on peut se contenter d'envoyer seulement des statistiques sur les températures relevées, comme la température moyenne par exemple.

Cette interface cloud servira également à accéder à une interface graphique permettant de visualiser les dernières données via Grafana. Nous pourrons également utiliser Jupyter[24] afin de traiter les données et réaliser des statistiques à plus large échelle (à l'aide de fonctions Python).

4.4 Améliorations diverses

Nous souhaitons aussi mettre en place plusieurs améliorations mineures.

Au niveau des capteurs, nous souhaitons mettre en place un système d'identification des capteurs afin de pouvoir identifier la position géographique des différentes données collectées, cela nous faciliterait la vie pour retrouver un capteur défaillant, cela pourrait également nous permettre de pouvoir plus facilement les retrouver en cas de vol ou de perte. Pour cela, nous utiliserons les capteurs GPS déjà présent dans les capteurs que nous possédons.

Une autre amélioration que nous souhaitons mettre en place concerne GlusterFS, un logiciel utilisé pour fournir un stockage distribué et redondant dans un cluster. Notre solution nécessite les nœuds enfants du cluster seulement pour des actions axées sur le calcul ou l'exécution de fonctions. Une conséquence de ceci est que GlusterFS n'a pas d'utilité et consomme donc des ressources inutilement. Nous avons donc décidé de nous en séparer.

Une autre fonctionnalité que nous pourrions ajouter serait liée aux capteurs. En effet, un capteur LoRa ne peut être relié qu'à une gateway. Or les capteurs appartiennent à l'OSUR. Nous ne pourrions donc en avoir que quelques-uns. Nous n'aurions donc des données que d'une partie des capteurs de la croix-verte et non la totalité. Pour pallier cela, nous pourrions récupérer les données des autres capteurs de l'OSUR, qu'ils ont stockées dans une base de données, ainsi que des données externes (par exemple l'API de météo France), afin d'avoir un ensemble de données bien plus grand.

Pour finir, nous voudrions implémenter une interface utilisateur afin de pouvoir observer les ressources en temps réel du cluster (cpu, ram, consommation...), les fonctions en cours, l'historique des fonctions lancées, celles en cours, etc... Pour cela, nous pourrions utiliser OpenTelemetry et l'intégrer à Grafana.

4.5 Résumé

Lors de ce projet, nous souhaitons donc grandement modifier la stack, pour tenter de l'améliorer. Un tableau montrant l'évolution des technologies utilisées lors

du projet peut être trouvé en figure 2.

Pour MQTT et NATS, nous attendons de faire marcher les fonctions pour réellement se positionner.

	Mycelium 1.0	Mycelium 2.0	Mycelium 3.0
Kubernetes (K8S)	X	X	
K3S			X
MQTT	X	X	
NATS			X
GlusterFS	X	X	
Fonctions Python	X	X	
Fonctions Go			X
VPS		X	X
Notifications (Discord, RSS, Telegram...)		X	X
Intégrations avec OSUR			X

TABLE 2 – Tableau de l'évolution de certaines technologies entre les différentes versions de Mycelium

Il y a donc de nombreuses modifications à faire dans le projet. Nous avons donc décidé de les classer par ordre de priorité afin de savoir dans quel ordre nous allons implémenter ces modifications. Nous avons mis cela dans un tableau de priorité des taches. 3

Taches	**Priorité**
Chirpstack	1
Cluster de VM	1
Guides	1
Identification des capteurs	2
MQTT -> NATS	2
Python -> Go	2
VPS	2
Discord -> FluxRSS	3
Visualisation des données (Cloud)	3
Intégrations données OSUR	4
Suppression de GlusterFS	4
Interface Utilisateur de Monitoring	5
Configuration automatique des capteurs	6
FoxyFind	7
Jupyter	7

TABLE 3 – Priorité des taches

Maintenant que nous avons vu les modifications que nous voulons apporter à l'architecture, la figure 8 présente un schéma résumant la nouvelle architecture du projet. Les éléments que nous allons supprimer sont en rouge, tandis que ceux que nous allons ajouter sont en vert.

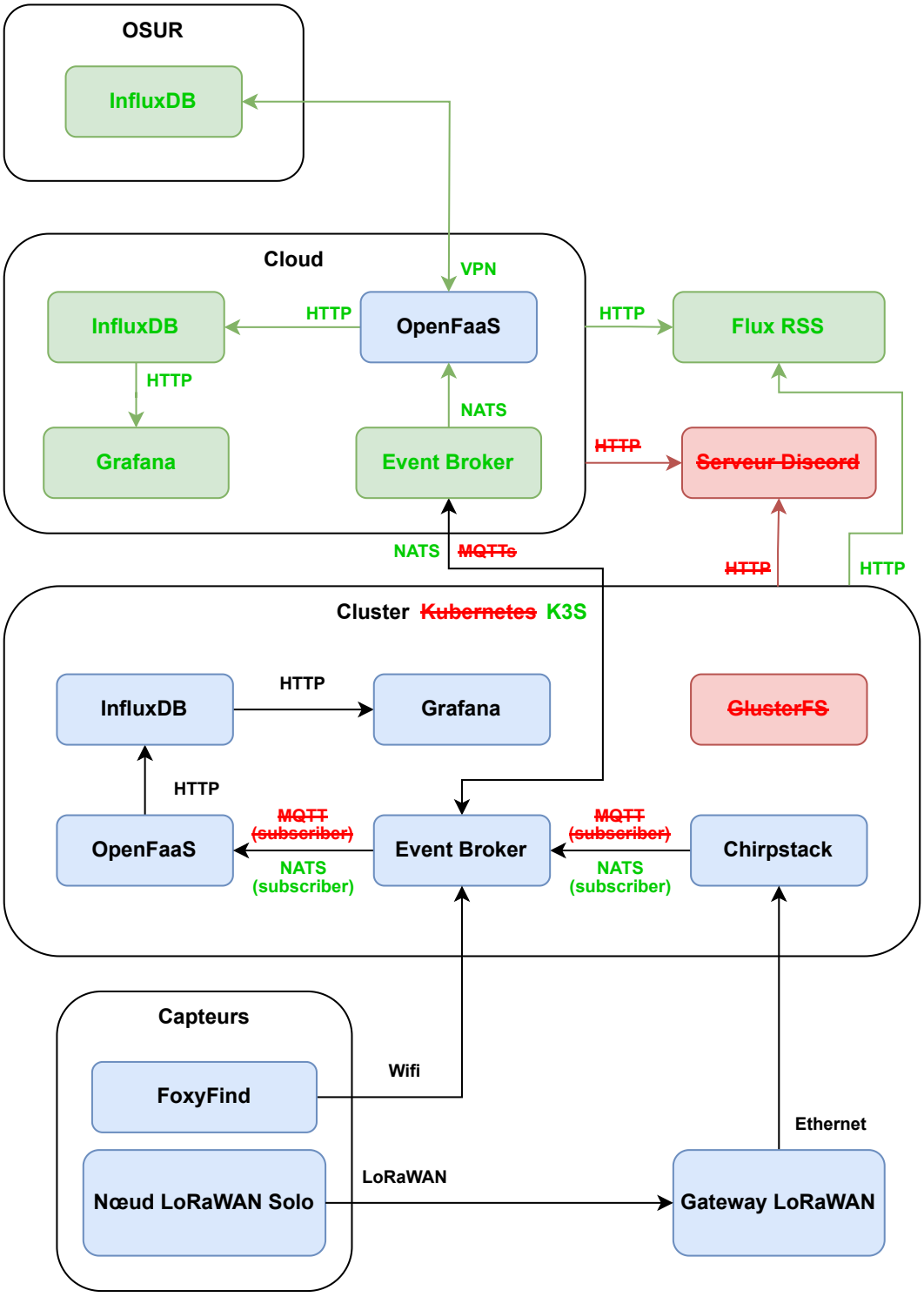

FIGURE 8 – Schéma de comparaison des architectures des projets Mycelium 2.0 et Mycelium 3.0 (vert = ajout, rouge = suppression)

5 Conclusion

Ce document a présenté le contexte du projet Mycélium V3, en exposant l'état actuel des technologies utilisées, les versions antérieures, ainsi que leurs limitations. Nous avons également détaillé l'architecture des versions précédentes pour mettre en évidence les domaines nécessitant des améliorations. En outre, nous avons fourni une vue d'ensemble des modifications que nous prévoyons d'apporter au projet en termes de spécifications générales et fonctionnelles.

L'objectif global de notre projet est de simplifier l'architecture afin de réduire la consommation des ressources du cluster et diminuer la complexité de toute l'infrastructure logicielle. Pour concrétiser cette démarche, nous commencerons par nous concentrer sur la simplification de l'architecture, en préservant les fonctionnalités essentielles (collecte de données, scénarios). Une fois cette étape franchie, notre attention se portera sur l'intégration d'autres fonctionnalités, comme FoxyFind (qu'il faut terminer d'implémenter dans le système) par exemple.

Nous avons également rencontré des difficultés lors de la prise en main du projet, c'est pourquoi nous avons prévu de fournir une documentation précise et détaillée. Ceci permettra de faciliter la reprise du projet dans les années à venir.

Ainsi, la refonte complète de l'architecture de Mycélium marque une amélioration majeure par rapport à la version de 2022 et s'inscrit dans une démarche environnementale durable.

Table des figures

6 Bibliographie

Références

[1] Article France 3 Region : Ligne b du métro à Rennes. Un an après son ouverture, quel bilan https://france3-regions.francetvinfo.fr/bretagne/ille-et-vilaine/rennes/transports-ligne-b-du-metro-a-rennes-un-an-apres-son-ouverture-quel-bilan-2842178.html

[2] Page Wikipédia de la ligne B du métro de Rennes https://fr.wikipedia.org/wiki/Ligne_B_du_m%C3%A9tro_de_Rennes

[3] Site de LivingFog Platform. http://www.fogguru.eu/living-lab/

[4] Site de Raspberry Pi. https://www.raspberrypi.com/

[5] Moiroux-Arvis, L. ; Royer, L. ; Sarramia, D. ; De Sousa, G. ; Claude, A. ; Latour, D. ; Roussel, E. ; Voldoire, O. ; Chardon, P. ; Vandaële, R. ; et al. ConnecSenS, a Versatile IoT Platform for Environment Monitoring : Bring Water to Cloud. Sensors 2023, 23, 2896. https://www.mdpi.com/1424-8220/23/6/2896

[6] I-Site Clermont Auvergne, CNRS. SoLo : nœud communicant LoRaWAN https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwjg_qWMvL-CAxWgcKQEHQtrDp0QFnoECBYQAQ&url=https%3A%2F%2Findico.in2p3.fr%2Fevent%2F23490%2Fcontributions%2F91608%2Fattachments%2F62480%2F85614%2FPr%25C3%25A9sentation_logiciel_SoLo_fev21.pptx&usg=AOvVaw0tkdzpCRGQGBHEvQ2WC1do&opi=89978449

[7] Article CNRS. 2022. Terra Forma : déployer un réseau dense de capteurs pour comprendre et répondre aux défis environnementaux https://www.occitanie-ouest.cnrs.fr/fr/cnrsinfo/terra-forma-deployer-un-reseau-dense-de-capteurs-pour-comprendre-et-repondre-aux-de

[8] Article CNRS. 2021. Econect : Un consortium de neuf laboratoires et entreprises étudie les changements globaux grâce à des espèces “sentinelles” https://www.occitanie-ouest.cnrs.fr/fr/cnrsinfo/un-consortium-de-neuf-laboratoires-et-entreprises-etudie-les-changements-globaux-gr

[9] Tobias Pfandzelter, David Bermbach. 2023. Towards a Benchmark for Fog Data Processing http://arxiv.org/abs/2304.09026

[10] Hamidreza Arkian, Dimitrios Giouroukis, Paulo Souza Junior, Guillaume Pierre. Potable Water Management with integrated Fog computing and LoRaWAN technologies. IEEE IoT Newsletter, 2020, pp.1-3. hal-02513467 https://inria.hal.science/hal-02513467/file/main.pdf

[11] Site d’OpenFaaS. https://www.openfaas.com/

[12] Site d’InfluxDB. https://www.influxdata.com/

[13] L. Vangelista, "Frequency Shift Chirp Modulation : The LoRa Modulation," in IEEE Signal Processing Letters, vol. 24, no. 12, pp. 1818-1821, Dec. 2017, doi : 10.1109/LSP.2017.2762960.

[14] Site de Kubernetes. https://kubernetes.io/fr/

[15] Site de Chirpstack. https://www.chirpstack.io/

[16] Site de MQTT. https://mqtt.org/

[17] Site de Grafana. https://grafana.com/

[18] Site de Gluster. https://www.gluster.org/

[19] Site de QEMU. https://www.qemu.org/

[20] Site de NATS. https://nats.io/
[21] Site de K3S. https://k3s.io/
[22] Site du langage Go. https://go.dev/
[23] Site de SQLite. https://www.sqlite.org/index.html
[24] Site de Jupyter. https://jupyter.org/