RAPPORT FINAL
Mycelium

Guillaume CHAUVEAU, Pieyre IACONE, Florian DABAT

Élodie JUVÉ, Yifan TIAN, Victoria Maria VELOSO RODRIGUES, Haoying ZHANG

Encadrants : Nikolaos PARLAVANTZAS, Christian RAYMOND

Intervenants : Laurent ROYER, Laurent LONGUEVERGNE, Nicolas LAVENANT, Guillaume PIERRE

2021-2022

Résumé

Une équipe pluridisciplinaire nous sollicite pour faire un suivi environnemental d'un espace naturel à l'aide d'un réseau de capteurs intelligents. Pour ce faire, nous avons à disposition divers matériels et technologies, notamment pour la collecte et l'envoi des données et la mise en place d'une infrastructure *fog*.

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

Table des matières

1 Remerciements

Tout d'abord, nous souhaitons remercier nos encadrants INSA, Mr Nikolaos PARLAVANTZAS et Mr Christian RAYMOND, pour leur suivi et leurs conseils tout au long de l'année. Merci pour votre implication et pour l'aide apportée durant chaque réunion hebdomadaires.

Ensuite, nous remercions également nos encadrants extérieurs : Mr Laurent ROYER pour avoir répondu à nos questions à propos du fonctionnement du nœud SoLo, Mr Guillaume PIERRE pour les informations apportées concernant d'autres projets de suivi environnemental, Mr Laurent LONGUEVERGNE et Mr Nicolas LAVENANT pour leurs conseils et leurs idées de pistes d'améliorations.

Enfin, nous remercions les membres de l'équipe - Guillaume CHAUVEAU, Pieyre IACONE et Florian DABAT - partis en mobilité à l'étranger durant ce second semestre, pour leur intérêt à propos de l'état d'avancement de Mycelium et pour leurs conseils techniques apportés sur le *cluster*, et ce malgré la distance.

2 Introduction

Le projet Mycelium correspond au déploiement d'un réseau de capteurs intelligents sur la zone de la Croix-Verte dans le campus de l'université de Rennes 1. Le but est de réaliser le suivi environnemental de cette zone en renaturation, et d'analyser les différentes données fournies par les capteurs. Ce rapport permet de clôturer le travail fait par les membres de l'équipe ayant travaillé sur le projet Mycelium.

Premièrement nous présenterons les rectifications par rapport à ce qui était initialement prévu dans les rapports de spécification et de conception. Nous présenterons ensuite l'état de notre produit, ses fonctionnalités et ses améliorations. Puis nous détaillerons les tests effectués sur le produit pour vérifier que les différentes implémentations fonctionnent mais aussi ceux qui permettent de vérifier la qualité des données. Enfin, nous exposerons les pistes envisagées pour la suite du projet afin de donner des idées d'améliorations pour une éventuelle reprise du projet.

3 Mise à jour des spécifications et de la conception

Au fur et à mesure que nous développions les fonctionnalités prévues dans le cahier des charges, nous nous sommes rendu compte que la manière dont nous avions prévu de les faire n'était pas viable dans le cadre des demandes et du temps disponible pour le développement du projet. Dans cette partie, nous allons présenter des mises à jour des spécifications et de la conception du projet Mycelium.

3.1 Mise à jour des spécifications et de la conception concernant le nœud SoLo

Tout d'abord, l'équipement utilisé est un nœud SoLo équipé de quatre capteurs : un accéléromètre, un capteur « HumiTemp » pour mesurer la température et l'humidité, un capteur « Lumi » pour mesurer la luminosité et un capteur « Press » pour mesurer la pression. Toute la configuration de nœud, y compris le réseau, les alarmes, la date et la fréquence de mesure, est effectuée via un fichier JSON, comme illustré dans les rapports précédents. De plus, nous avons également un pluviomètre de type LUFFT WTB100 qui a été installé pour mettre en œuvre les scénarios du projet de conception.

Comme prévu, un système a été développé pour détecter les défaillances du nœud lors du démarrage, dans ce système une mesure de chaque capteur est prise et envoyée au *cluster* pour vérifier la valeur GPS, l'heure et le bon fonctionnement de tous les capteurs. Pour cela, nous avons créé une fonction SendAll(), qui est appelée à chaque redémarrage du système, cette fonction est capable d'initialiser tout le processus de mesure, depuis la création du fichier « .csv » pour le stockage, jusqu'à l'activation de chacun des capteurs disponibles 4.2.

Dans le rapport de conception, nous avons proposé un protocole du nouveau système d'alarmes qui permet de définir des scénarios avec plusieurs capteurs de manière générale. Cela nécessite de créer deux nouvelles classes et sera difficile à implémenter. En raison de la difficulté technique et le manque du temps, nous n'avons finalement pas eu temps de finir la construction de ce nouveau système d'alarmes. Néanmoins, nous avons réussi à implémenter deux fonctions embarquées pour réaliser deux scénarios que nous allons préciser dans la partie 4.2.2. Cela permet de compléter la chaîne de réactions lors qu'un scénario a lieu.

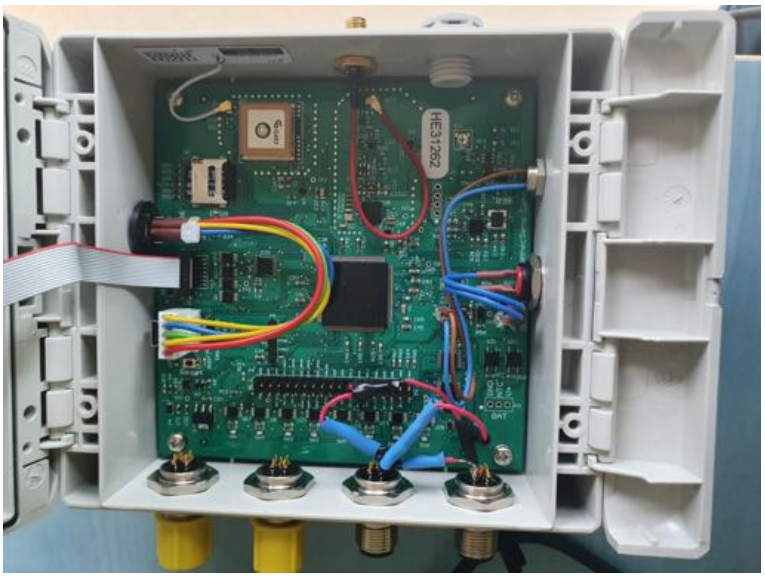

FIGURE 1 – Soudure faite pour le mise en place du pluviomètre

La version du nœud SoLo qui nous a été mise à disposition n'était pas adaptée pour utiliser un pluviomètre, il a donc fallu implémenter un circuit dans le système embarqué pour installer le capteur externe, comme le montre l'image 1. Pour cela, nous suivons le schéma de la figure 2, dans laquelle nous utilisons la ligne d'interruption « INT_1 » comme interface de communication entre le capteur et le nœud selon la documentation de câblage du ConnecSens [5]. Le rond avec les cinq points dans le

schéma correspond à la borne la plus droite dans la photo du nœud (FIGURE 1), nous allons brancher le pluviomètre à cette borne. Le nœud et le pluviomètre sont connectés par un point de soudure, la ligne vert dans le schéma, et la broche GND (terre), représenté par la ligne noir. En outre, pour utiliser le sous-système sans interruption dans le nœud SoLo, on soude également le point sérigraphié TP6, en série avec une résistance de 200K, pour éviter les faux positifs et réduire la consommation d'énergie, et en parallèle à la ligne « INT_1 », cette connexion est représentée dans le schéma par la ligne rouge.

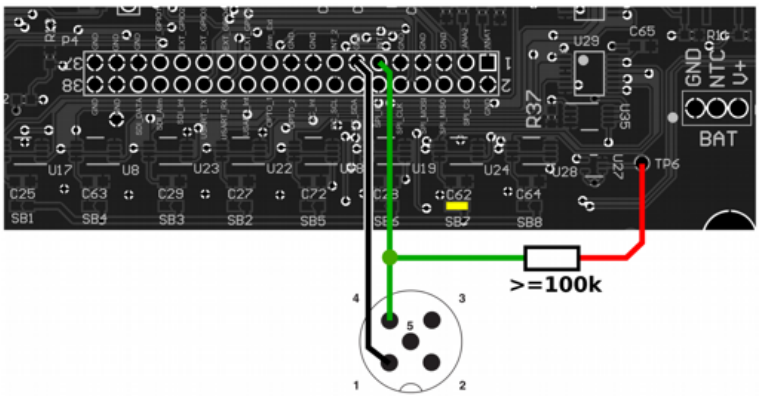

FIGURE 2 – Schéma de soudure du pluviomètre

Enfin, dans le rapport de spécification, nous avions prévu d'implémenter un lien descendant pour mettre en œuvre la communication allant du serveur vers le nœud SoLo. Mais comme cette fonctionnalité n'est jamais implémentée dans aucune version du *firmware*, nous aurions besoin d' une connaissance approfondie en systèmes embarqués. De plus, les autres fonctionnalités nous ont pris beaucoup plus de temps à réaliser, nous avons donc décidé de ne pas le mettre en place afin de prioriser les fonctions des scénarios planifiés.

3.2 Mise à jour des fonctions de traitement de données OpenFaaS

Pour le traitement des données capturées et envoyées par le nœud SoLo au niveau du *cluster*, nous déployons des fonctions OpenFaaS en Python sur celui-ci. Ces fonctions réalisent le décodage, le stockage et la classification des données (si les données sont envoyées dans des conditions normales ou en cas d'alarmes). Elles permettent également l'envoi des notifications afin d'informer les utilisateurs, c'est-à-dire soit d'un scénario qui a lieu en temps réel, soit de changements significatifs des données environnementales par rapport à l'heure précédente ou de divergences entre les données de surveillance et les données fournies par la station météorologique de Saint-Jacques de la Lande sur Rennes.

À la différence de ce qui est décrit dans le rapport de conception, nous avons généralisé les fonctions de traitement pour chaque scénario en une seule : *topic handler*. Lorsque de nouvelles données vont être reçues, elles seront toujours stockées dans InfluxDB grâce à la fonction *frame recorder*. Mais en même temps, leur exactitude sera vérifiée par la fonction *compare general api* en comparant avec les données météorologiques en temps réel retournées par l'API météo décrite dans le rapport de conception. Elles seront également comparées aux données précédentes pour déterminer s'il existe de grands écarts entre les données de périodes proches par la fonction *compare general historic*. En outre, la fonction *topic handler* permet de confirmer si les données correspondent à un scénario particulier, car le nœud SoLo envoie les données sans préciser selon quel événement il a déclenché l'envoi. Puis, si un scénario est trouvé, la fonction enverra une notification sur un channel Discord pour informer les utilisateurs : par exemple quand la température reçue est plus grande que 45 degrés, c'est-à-dire que l'exception de température est vraie, nous recevrons une notification avec une description de la situation (Scénario Température anormale) et la ou les données de déclenchement. La notification sera également envoyée lorsqu'il y a une grande différence entre les données observées et les données de l'API, ou lorsqu'il y a un écart important par rapport aux dernières données reçues.

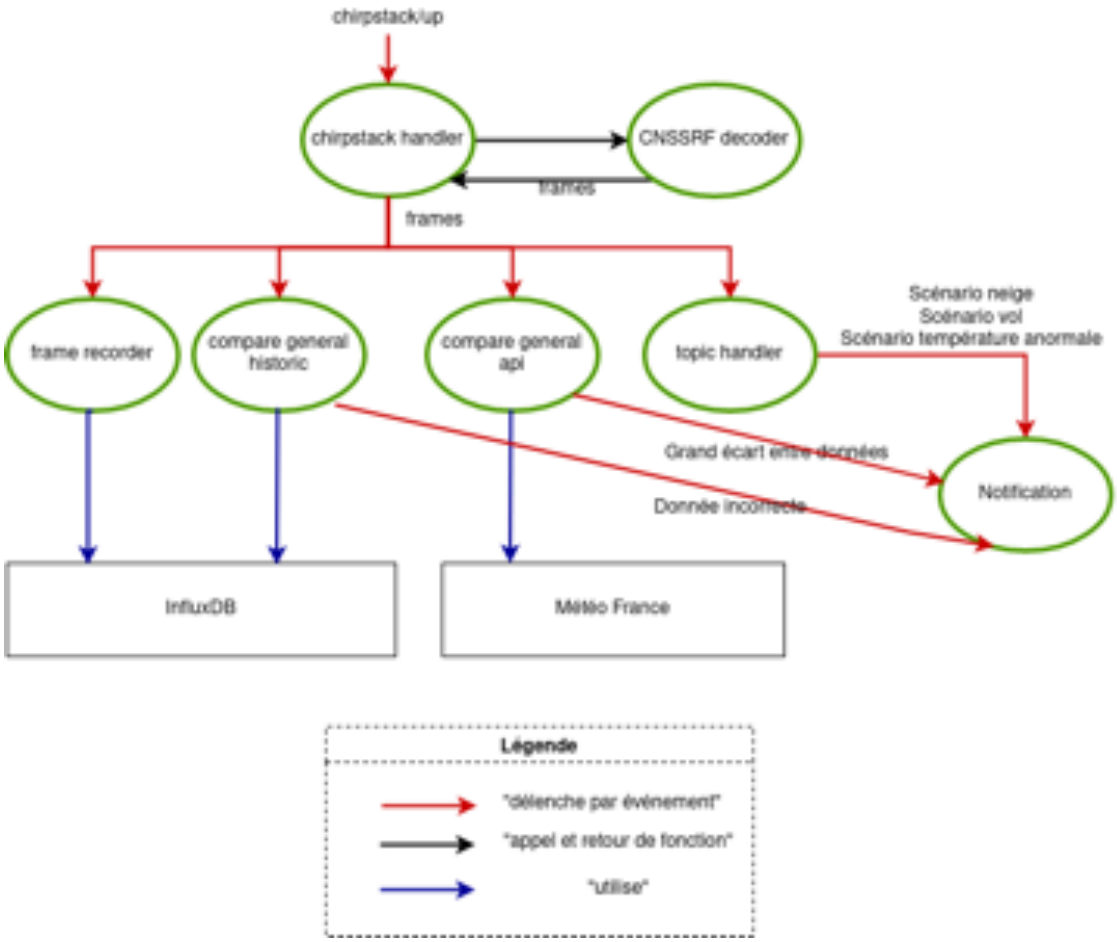

FIGURE 3 – Schéma mise à jour des relations entre les fonctions de traitement

Cette modification présente certains avantages, tels que la réduction de la répétition et de la redondance du code, et la diminution du temps de réaction pour les scénarios, mais elle ne vérifie pas la qualité des données avant d'envoyer la notification : il est donc possible que les données ne soient pas tout à fait correctes et que l'utilisateur n'en soit informé qu'après la réception de la notification des scénarios.

4 État de finalisation du projet

Dans cette partie, nous allons présenter l'état final du projet Mycelium par quatre aspects : premièrement, nous expliquerons la fonctionnalité du nœud SoLo original ainsi que les détails sur le nouveau concept de scénario qui est introduit afin de rendre le nœud SoLo plus intelligent ; deuxièmement, nous expliquerons la fonctionnalité au niveau du cluster, en plus d'agir en tant que le serveur du projet, il fournit également de l'évolutivité et de la traçabilité ; et troisièmement, nous parlerons des fonctions OpenFaaS déployées sur le cluster qui sont responsables du traitement des données provenant du nœud SoLo. Le schéma 4 ci-dessous montre une chaîne de réactions complète lorsqu'un scénario a lieu afin de donner un aperçu général.

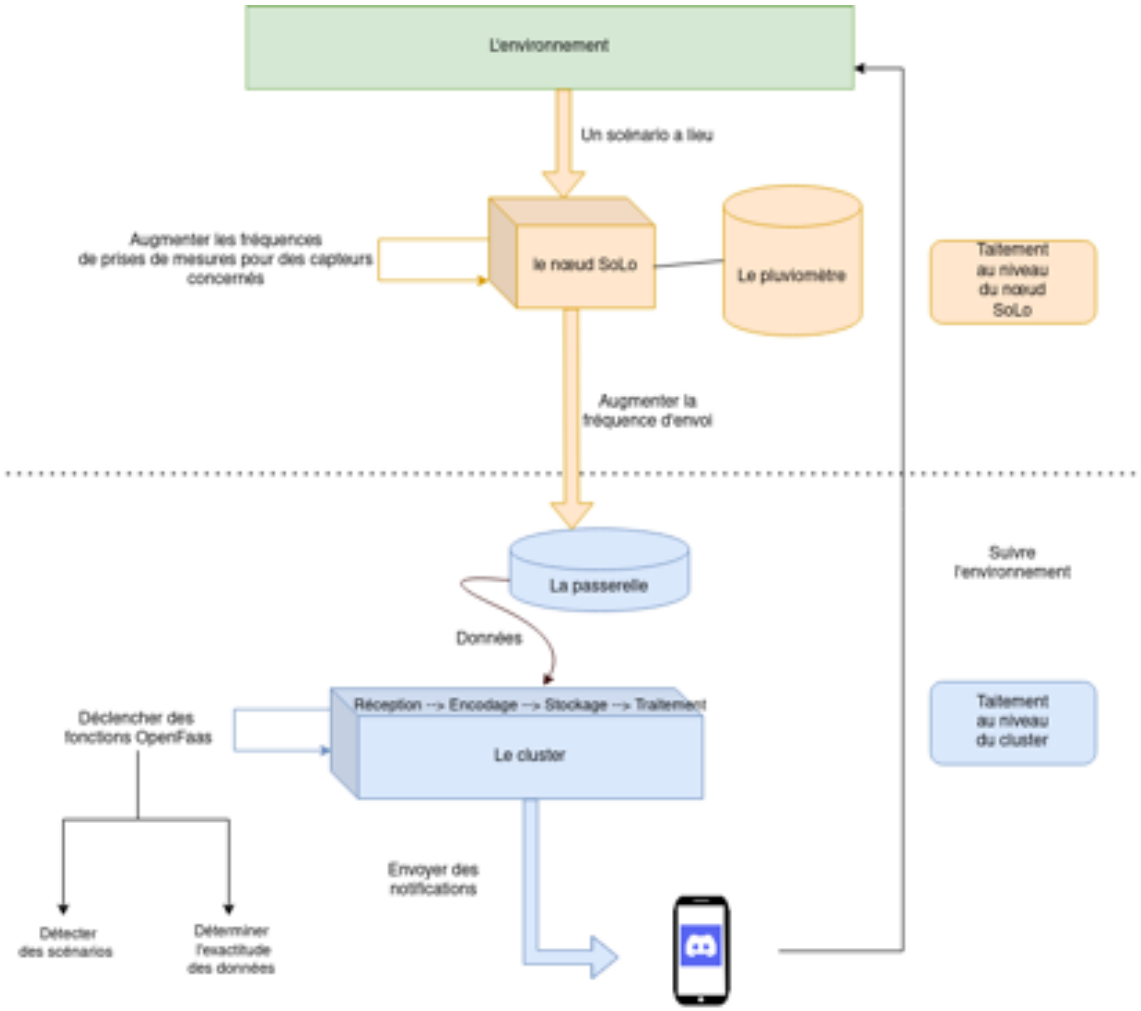

FIGURE 4 – Schéma de la chaîne de réactions en cas de scénarios

4.1 Scénarios traités

Un des objectifs de ce projet est de développer un système capable de réagir aux différents scénarios envisagés, ces scénarios correspondent à des situations inhabituelles qui nécessitent une analyse plus précise, comme des mesures anormales par rapport à un historique. Dans cette partie nous allons présenter les scénarios détectés et traités dans Mycelium.

Dans ce projet, les réactions sont traitées à deux niveaux différents, dans le *cluster* ou dans le nœud SoLo. Les scénarios vérifiés dans le nœud SoLo sont ceux dont la réponse est l'augmentation de la fréquence de mesure et d'envoi des données relevées par les capteurs correspondants. Et donc, la partie

de l'identification et du traitement des scénarios mis en œuvre dans le *cluster* sont ceux dont la réponse génère une notification.

Les scénarios implémentés dans le nœud SoLo sont neige et luminosité, un scénario à tester dans un environnement de développement. Le premier permet une observation plus détaillée de la température, de l'humidité et de la quantité de neige en millimètres, car lors de la détection d'une température basse et d'une humidité élevée, la fréquence de mesure des capteurs de température, d'humidité et de pluviomètre est augmentée. Le scénario Luminosité a été conçu comme un scénario pour tester les fonctions développées à l'aide des capteurs de luminosité et de température. Comme il n'était pas possible de tester des scénarios réels ou de les simuler avec précision lors de la phase de développement et de conception, nous avons créé un scénario utilisant des capteurs et des mesures plus faciles à manipuler. Les seuils de capteur dans chaque scénario sont expliqués dans la section 4.2.2.

Le *cluster* analyse les données envoyées par le nœud SoLo et, dès qu'il identifie un scénario déclenché dans le nœud SoLo, ou l'un des autres scénarios implémentés, il envoie une notification au serveur Discord comme indiqué sur la figure 4. Les scénarios identifiés dans le *cluster* sont le vol, lorsqu'il y a une altération inattendue dans les mesures de l'accéléromètre, la température extrême, lorsque la température mesurée dépasse un seuil, et un grand changement consécutif de données, pour indiquer un changement soudain et inattendu dans l'une des mesures envoyées par le nœud SoLo. En plus de ces scénarios spécifiques, le *cluster* envoie également une notification lors de l'identification d'un des scénarios précédemment vérifié dans le nœud SoLo comme illustré par l'image 5.

FIGURE 5 – Notifications de scénario envoyées sur le serveur Discord

Pour conclure, les scénarios que nous traitons au niveau du nœud SoLo et du *cluster* sont les suivantes :

1. Scénario de luminosité , avec capteur « HumiTemp » et « Lumi »
2. Scénario de neige , avac capteur « HumiTemp »
3. Température anormale, avec capteur « HumiTemp »
4. Luminosité anormale , avec « Lumi »
5. Scénario Vol
6. Grand changement consécutif de données

Nous pouvons très bien ajouter des autres scénarios, mais ce sont les scénarios que nous avons surtout testés car les grandeurs sont relativement facile à faire varier.

4.2 Fonctionnalités du nœud SoLo

Le nœud SoLo a été conçu par des chercheurs du laboratoire de physique de l'Université de Clermont-Ferrand dans le cadre du projet ConnecSenS. Son objectif est de prendre des mesures de données environnementales afin de mieux appréhender le changement climatique de la planète dans une perspective de développement durable. Pour cette raison, l'équipe a créé le nœud SoLo, un appareil entièrement reconfigurable et capable de s'adapter à différents contextes [1]. Dans le cadre du projet Mycelium, notre

objectif est de modifier cet équipement pour l'adapter aux besoins de suivi climatique de la Croix-Verte. Il était donc nécessaire d'analyser les fonctions déjà implémentées du nœud SoLo et de ses capteurs afin de les intégrer dans les scénarios conçus et de développer les fonctionnalités supplémentaires nécessaires.

Dans les sections 4.2.1 et 4.2.2, nous énumérons tout d'abord les fonctionnalités précédemment implémentées et nous faisons un exposé des fonctionnalités développées pour le projet Mycelium.

4.2.1 Fonctionnalités déjà implémentées dans le nœud SoLo

Le nœud SoLo a déjà des fonctionnalités fondamentales suivantes :

1. Prendre des mesures selon une fréquence pour chaque capteur
2. Enregistrer des données dans la carte SD
3. Envoyer des données selon une fréquence pour tous les capteurs
4. Définir un seuil d'alarme et la fréquence de prendre des mesures en mode d'alarme pour chaque capteur
5. Définir une ou plusieurs interruptions pour chaque capteur

Une des idée du projet Mycelium est de rendre le nœud SoLo plus intelligent, qui peut réagir automatiquement aux plusieurs scénarios sans interférence des humains. En plus, il devrait être capable de faire signaler les comportements anormales de lui-même.

4.2.2 Nouvelles fonctionnalités implémentées dans le nœud SoLo

Nous avons donc ajouté des fonctionnalités suivantes :

1. Forcer les capteurs à effectuer un envoi lors du premier démarrage

   Cette fonctionnalité vise à détecter un dysfonctionnement du nœud SoLo. Quand il redémarre anormalement sans arrêt, nous pourrons le détecter immédiatement.

2. Ajouter deux fonctions embarquées pour les deux scénarios

   Le nœud SoLo peut prendre des données de 5 grandeurs : la température, l'humidité, l'accélération selon x, l'accélération selon y, l'accélération selon z, la luminosité, la quantité de la pluie, les données des grandeurs sont prises par 4 capteurs internes et 1 capteur externe. On peut définir une fréquence de prise de données pour chaque capteur et une fréquence d'envoi pour tous les capteurs. C'est possible de définir des alarmes en fixant des seuils pour des grandeurs dans la fiche de configuration, ces alarmes peuvent servent à nos scénarios, par exemple le scénario de la température anormale et le scénario de détection du vol. En revanche, pour des scénarios qui ont besoin de plusieurs seuils ou si on voulait d'implémenter plusieurs seuils pour un grandeur, les alarmes deviennent restreints.

   Pour implémenter plus de scénarios, nous avons créer deux fonctions embarquées dans le système du nœud SoLo. Une fonction correspond au scénario neige, l'autre correspond au scénario de luminosité. Quand la température est inférieur à 0 et quand l'humidité est supérieur à 88 %, le scénario neige va être déclenché. En conséquence, la fréquence de prise de mesure et la fréquence d'envoi vont augmenter pour bien suivre le scénario. En ce qui concerne du scénario de luminosité, on fixe un seuil pour la température et un autre pour la luminosité. Quand la température est supérieur à 20 degré et la luminosité est supérieur à 700 lux, on augmente les fréquences.

   Nous allons détailler les tests pour les fonctions embarquées de scénarios dans la partie 5.2 .

4.3 Fonctionnalités de *cluster* de Raspberry Pi et Mycelium Core

Comme décrit dans le rapport de conception, nous avons déployé Mycelium core, l'infrastructure du *cluster* qui permet la réception des données envoyées par la passerelle LoRaWAN, et le déclenchement des fonctions *serverless* qui réalisent le traitement des données sur le *cluster*.

De plus, avec l'utilitaire Mycelium-Garden, les développeurs peuvent coder, tester et mettre en place très facilement de nouvelles fonctions. Pendant la phase de codage, nous implémentons des fonctions par un script Python et nous prenons les modèles de modules de Garden[4] pour les fonctions, cela simplifie considérablement la difficulté de développement. En effet, en abstrayant la relation entre les fonctions et les autres déploiements, les développeurs de fonctions n'ont pas à se soucier des autres éléments du

cluster. Lors du mode de développement sur le *cluster* local pour tester la fonctionnalité des fonctions, Mycelium-Garden nous permet de visualiser les changements en temps réel quand on modifie le code de la fonction, il ne déploiera pas réellement la fonction avec OpenFaaS, il la déploiera comme un service normal dans le *cluster*, afin de mettre en place le partage de fichiers entre le code source et la fonction en direct. Donc si nous mettons à jour le code, nous n'avons pas à couper le mode de développement et relancer, la fonction actualisera automatiquement. Une fois que nous avons terminé le développement local, on confirme par des tests que la fonction marche bien. Le déploiement sur le *cluster* physique utilise aussi Mycelium-Garden, sous la condition d'avoir l'accès au cluster physique, par une seule commande et tout sera mis en service.

En ce qui concerne le suivi d'état du *cluster*, en utilisant Mycelium-Garden et GitLab Runner, les journaux d'exécution des pods sont accessibles via GitLab, cela permet d'accéder aux informations utiles en cas de panne mais l'intervention dans le *cluster* n'est pas possible. Toutefois, l'intégration de Grafana dans GitLab envisagée dans le rapport de conception n'a été pas mis en place avec succès. Par conséquent la visualisation sur Grafana des données collectées n'est accessible que sur le réseau «IOTINFO». En ce qui concerne l'intégration continue mentionnée dans le rapport de spécification, vu la complication d'assurer de la sécurité lorsqu'on lance la commande de Garden[4] sur GitLab, nous n'avons pas pu à l'implémenter.

4.4 Fonctionnalités des fonctions serverless du *cluster*

Plusieurs fonctions OpenFaaS sont déployées sur le *cluster*. On a tout d'abord celles qui permettent de décoder les trames actuelles et d'enregistrer leur contenu sur InfluxDB. On a également des fonctions qui permettent de comparer les données actuelles à celles sur Internet.

5 Compte-rendu des phases de tests

Dans cette partie, nous allons parler des différents tests que nous avons effectués sur le nœud SoLo, sur le *cluster* et sur des fonctions du *cluster*.

5.1 Tests généraux

Dans le but de nous mettre en conditions réelles, et de récolter des mesures caractéristiques de la saison, on a placé le nœud SoLo et le pluviomètre dans la Croix-Verte. On a choisi de les mettre dans des boqueteaux où les équipements sont isolés des passagers. La localisation exacte du matériel sur la Croix-Verte est présentée dans la figure 6.

FIGURE 6 – Carte indiquant la localisation du matériel sur la Croix-Verte

FIGURE 7 – Photos de l'installation du matériel sur la Croix-Verte

5.1.1 Tests sur la réaction du nœud SoLo

Nous avons laissé le nœud SoLo allumé pendant 16 heures et le nœud SoLo se réveillait tous les 20 secondes pour prendre des données (2880 fois) et envoyait des données tous les 60 secondes (960 fois). Dans le cas général, les fréquences sont fixées beaucoup moins haut avec une prise de données toutes les heures et un envoi de données toutes les 4 heures. Ce test nous permet de vérifier les bon fonctionnements du nœud SoLo y compris la capacité de la batterie et la capacité de la carte SD.

5.1.2 Tests sur la bonne réception des données sur le *cluster* via LoRaWAN

Le nœud SoLo n'arrive pas à rejoindre le réseau si nous le plaçons dans la Croix-Verte. En effet, en regardant dans les logs, il dépasse le temps de la réception de l'ACK (message de confirmation pour rejoindre au réseau) après avoir envoyé la requête de demande de la connexion. La communication LoRaWAN dépend des conditions de propagation du signal radio. Les conditions comme par exemple des bâtiments et leurs hauteurs et la végétation entre le nœud SoLo et la passerelle ont un impact sur la propagation du signal radio.

5.2 Tests sur les scénarios

Nous avons fait des tests sur deux des scénarios que nous avons présentés dans la partie 4.1. Considérant que le nœud SoLo est utilisé pour les mesures environnementales, lors des tests de scénarios, il a été nécessaire de manipuler l'environnement perçu par les capteurs. Ainsi, il a fallu utiliser quelques « tours » pour simuler les scénarios et tester nos fonctions embarquées. La variation de température se faisait en plaçant de le glaçon sur le nœud SoLo ou en le chauffant à l'aide d'un sèche-cheveux. Nous avons testé différents niveaux de luminosité en utilisant une lampe de poche pour éclairer l'environnement ou en masquant le capteur pour l'assombrir. Et nous avons simulé la pluie avec un vaporisateur pour tester le bon fonctionnement du pluviomètre. De cette façon, il a été possible de tester chacune des fonctions et scénarios implémentés.

Pour le scénario de luminosité, nous pouvons voir dans le schéma 8 que lors que nous sommes en cas de scénario, nous recevons plus de données qu'en cas normal. Si nous regardons plus en détail des *trames* que nous recevons (figure 11), nous constatons que surtout les fréquences du capteur « HumiTemp » et le capteur « Lumi » ont été augmentées pour avoir plus de données sur les grandeurs « airHumidity » , « temperature » et « illuminance ». Pour le reste de données, elles ne sont pas prioritaires dans ce scénario, donc nous gardons leurs fréquences par défaut pour ne pas dépasser la limite d'envoi des paquets de réseau.

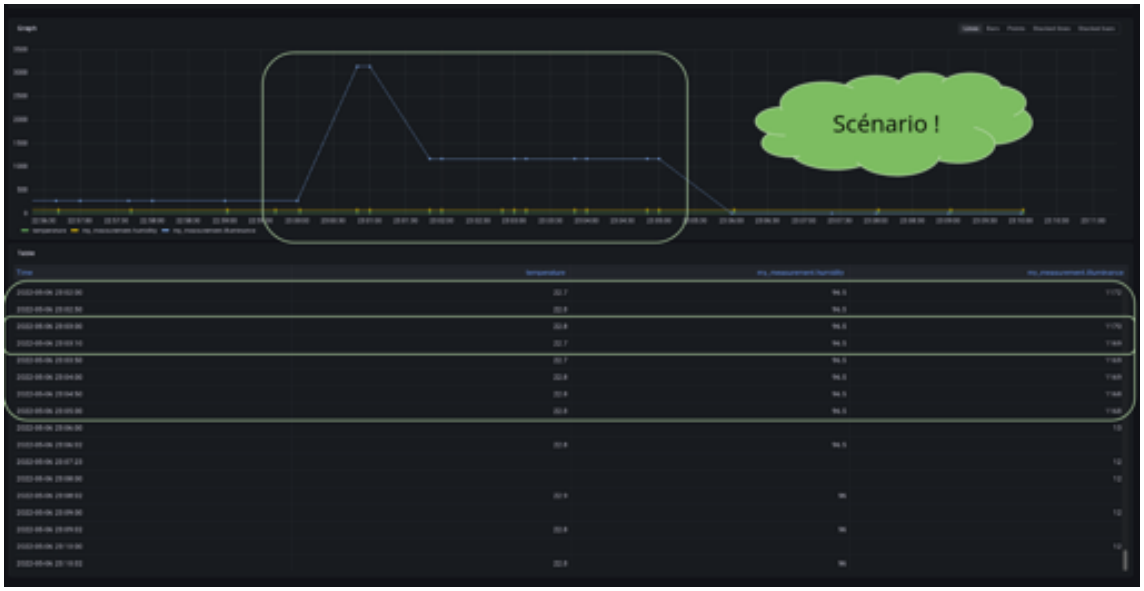

FIGURE 8 – Tests sur le scénario de luminosité

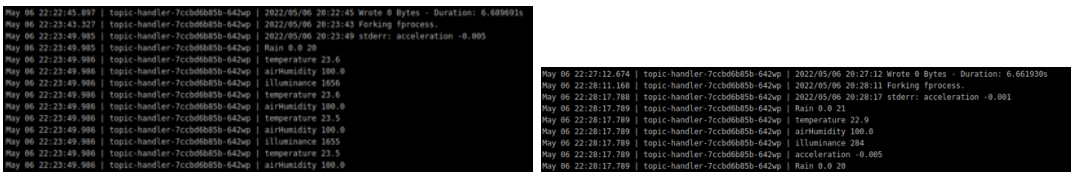

FIGURE 9 – Comparaison de la *trame* en cas de scénario et en cas normal

Pour le scénario neige, l'idée est la même que pour le scénario de luminosité. Nous avons modifié le seuil pour la température à 18 degré juste pour faire les tests (le seuil est normalement fixé à 0 degré). Nous pouvons regarder dans le schéma 10 que quand la valeur d'humidité (la ligne bleu claire) dépasse le seuil (la ligne rouge horizontale), nous avons plus de données pour la température et l'humidité.

Dans ce test, nous pouvons très bien visualiser que l'augmentation des fréquences est faite si et seulement si les deux grandeurs dépassent tous les deux à leur seuils. Pour rappel, nous avons défini le scénario neige pour le test par une température strictement inférieur à 18 degré et une humidité strictement supérieur à 88 %. Donc nous avons bien vérifié que même si les valeurs de la température est toujours dans la bande d'alarme, l'alarme n'est déclenchée que quand la valeur de l'humidité dépasse aussi son seuil.

FIGURE 10 – Tests sur le scénario neige

5.3 Tests du *cluster*

En tant que serveur du projet, le *cluster* doit répondre aux exigences : d'abord une bonne réception des données au niveau du Chirpstack qui peut démontrer le déploiement réussi de Mycelium Core. Ensuite un stockage correcte des données reçues, il permet non seulement de prouver la stabilité des volumes persistants qui sont actuellement déployées sur *cluster*, mais aussi de nous donner une occassion d'observer l'usage de la mémoire du *cluster* afin de déterminer si les matériaux actuel (quatre nœuds de travail Raspberry pi pour les tâches de stockage, chacun équipe une carte mémoire de 30G) peut être capable pour le stockage de données à long terme. De plus un traitement correctes des données qui sera présenté dans la partie suivante 5.4, l'extensibilité des fonctions de traitement est aussi nécessaire, cela implique le déploiement de nouvelles fonctions OpenFaaS pour pouvoir traiter de nouvelles situations, par exemple l'implémentation de nouveaux scénarios au niveau du nœud SoLo. Nous avons donc testé sur *cluster* pour les trois aspects : la réception des données ; le status de stockage et le déploiement de nouvelles fonctions.

5.3.1 Tests en tant que serveur de LoRaWAN (la réception des données sur Chirpstack)

Le journal de nœud SoLo contient l'état de chaque envoi, et l'information fourni par l'interface Web du Chipstack sur le *cluster* nous preuve la correspondance entre les envois et les réceptions. Lorsque le nœud SoLo indique que les données ont été envoyées avec succès, nous pouvons retrouver sur Chirpstack les informations de réception correspondantes, mais si le nœud SoLo ne parvient pas à envoyer les données, le Chirpstack n'a pas d'informations sur la réception. À noter que le fuseau horaire du nœud SoLo est UTC+0, donc l'heure dans son journal a un décalage de deux heures.

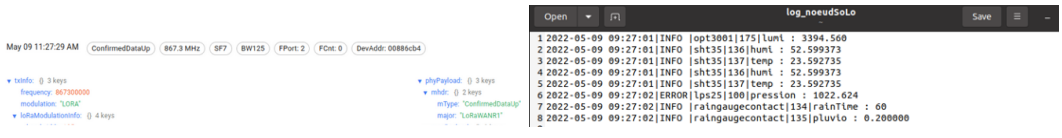

FIGURE 11 – Correspondance entre l'envoi du nSoLo et la réception de Chirpstack

5.3.2 Test du stockage des données

Les données stockées dans InfluxDB correspondent toutes aux journaux des envois du nœud SoLo.

Nous avons testé le redémarrage par commande et le redémarrage après une coupe de courant du *cluster* avec aucune perte dans la base de données, ce qui prouve la stabilité du stockage.

Nous avons jusqu'à présent 8000 données stockées dans InfluxDB du *cluster* qui occupent 4k de mémoire, donc avec la fréquence envisagée dans le rapport de conception, c'est-à-dire que le nœud SoLo mesure toutes les heures et envoie les données mesurées toutes les quatre heures, dans le cas normal, 24 données seront envoyées pendant une journée, même dans le cas de scénarios où la fréquence de mesure et d'envoi du nœud SoLo augmente, le stockage à long terme peut toujours être garanti.

5.3.3 Déploiement des fonctions OpenFaaS

— Visualisations des données dans Grafana : des tests sur le bon fonctionnement du décodage et de l'enregistrement des données dans InfluxDB ont été réalisés. On voit bien que les fonctions OpenFaaS pour le décodage et l'enregistrement des données dans InluxDB sont bien déployées et fonctionnent sur le serveur de bout en bout, car on visualise leur résultat via Grafana

FIGURE 12 – Une capture d'écran des données qu'on reçoit sur Grafana

— Déploiement des fonctions de test sur la qualité des données : Dans le cadre de certains scénario, des fonctions sont implémentées pour vérifier de la qualité des données. Par exemple la fonction *compare-general-api* permet de comparer cahque mesure de température et d'humidité reçue dans les trames avec les données à la date correspondante dans l'API.

5.4 Tests sur les fonctions de traitement

Lors du développement des fonctions de traitement, des tests ont été effectué pour vérifier leur bon fonctionnement. En effet, nous voulions être sûrs de leur bon fonctionnement avant de les déployer sur le *cluster*.

Pour cela, nous avons utilisé l'outil *minukube*[2] qui permet un déploiement local de *Kubernetes*[3] dans le but de simuler un *cluster* virtuel sur l'ordinateur du développeur. Nous avons également voulu simuler l'envoi de faux paquets de données sur ce *cluster* local. Pour cela nous avons implémenté le module *chirpstack-simulator*. Ce module est codé en GO et permet de recevoir une frame contenant 4 fausses trames de données prises à 30 minutes d'intervalle les unes des autres.

On peut ainsi bien vérifier le bon déclenchement et le bonne exécution de ces fonctions de plusieurs manières :

— Vérifier les logs : ils permettent d'afficher les erreurs éventuelles et les résultats des *print* pour les fonctions serverless. Ces logs sont disponibles dans l'interface de Garden[4] ou sur Gitlab.

— Vérifier la bonne réception d'une notification sur Discord : si un scénario s'est déclenché suite à l'envoi de données, nous pourrons le voir ainsi.

— Utiliser l'interface d'OpenFaaS, afin d'invoquer les fonctions serverless en cours d'implémentations. Si une erreur est détectée, l'interface renvoie une erreur 500 : il faut donc se référer aux logs comme indiqué dans le premier point.

Toutes ces démarches et pratiques de tests ont été utilisées tout au long du développement des fonctions de traitement.

Des tests ont ensuite été réalisés à plus large échelle une fois les fonctions de traitement déployées sur le *cluster* : en recevant des *frames* contenant des données réelles, on a pu vérifier le bon déclenchement et la bonne exécution des fonctions implémentées.

6 Pistes pour la suite

Mycelium est un projet à fort potentiel d'amélioration, il peut être adapté pour répondre aux exigences des différents environnements de suivi climatique. Dans cette partie, nous allons présenter quelques-unes de ces améliorations possibles.

6.1 Améliorations techniques

Il pourrait être intéressant de mettre en place un lien descendant au niveau du nœud SoLo : pour l'instant, la communication entre le nœud SoLo et le *cluster* ne se fait que dans un sens. Mais il serait intéressant que le *cluster* puisse envoyer des ordres au nœud SoLo, afin d'obtenir des données supplémentaires selon les besoins.

Ensuite, pour faire fonctionner le nœud SoLo et le pluviomètre dans la Croix-Verte, en mettant la passerelle et le cluster dans le bâtiment INFO, il faudra faire des tests de couverture du réseau LoRaWAN supplémentaires à l'aide du mode de travail : « campaignRange ». Ce dernier est un mode de cartographie des portées LoRaWAN sur le terrain. Dans ce mode, le nœud SoLo réalise des synchronisations GPS en permanence, toutes les cinq secondes environ, pour obtenir l'heure et surtout sa position GPS.

De plus, nous pourrions utiliser le *cloud*. En effet, quand nous commenceront à avoir de très grandes quantités de données, une grande quantité de ressources de calcul sera nécessaire.

Dans le sujet du projet, il était question d'utiliser des algorithmes d'apprentissage pour effectuer des prévisions de données. Cette idée a été mise de côté par manque de temps, mais il serait en effet très intéressant à mettre en place

Enfin, s'il est possible de se procurer du matériel en plus, on pourrait ajouter plusieurs nœuds SoLo qui pourraient communiquer entre eux. Nous pourrions également mettre en relation de nouveau aggregateurs, par exemple un capteur de concentration de CO2.

6.2 Amélioration environnementales

Les scénarios actuels ne permettent pas une analyse poussée des événement climatiques qui peuvent survenir. Ils sont pour l'instant très instrumentaux, c'est-à-dire très centrés autour du matériel : ainsi beaucoup ont pour but d'envoyer une alerte si la sécurité de celui-ci est menacées ou si les données semblent incorrectes. Les exemples ci-dessous sont des idées de traitement à mettre en place pour un meilleur suivi environnemental.

On pourrait implémenter plus de scénarios basés sur l'historique : par exemple, un scénario qui compare les données météorologiques reçues du nœud SoLo à celles de l'année précédente, à la même date, afin de suivre l'évolution année par année selon les saisons. Cela nous permettrait de comprendre la zone analysée sur le long terme.

Nous pourrions également faire davantage attention aux normes sur la prise de mesure : par exemple, les données de températures seront complètement différentes selon si le noeud SoLo est placé au soleil ou à l'ombre. Il faut donc non seulement bien fixer le matériel, mais aussi le placer à un endroit stratégique. C'est pour cela que nous avons choisit de na pas placer le pluviomètre sous un arbre. Mais pour être plus précis et respecter les normes de prise de mesure météo, le pluviomètre devrait dans l'idéal être placé au moins à 2m de hauteur.

Il serait aussi intéressant d'expérimenter différentes bases de données météorologiques, car il en existe beaucoup sur Internet : certaines sont certifiées, d'autres non. Il pourrait être intéressant d'exploiter celles qui proviennent de particuliers par exemple, qui seraient certes moins aux normes mais qui seraient encore plus proche de la Croix-Verte que celle de la station météo que nous utilisons actuellement.

Nous pourrions mettre également en place des scénarios basés sur des prévisions. Ils pourraient déclencher une alerte lorsque les données prélevées contredisent le modèle.

7 Conclusion

Mycelium est un projet qui mélange plusieurs disciplines. Il a tout d'abord permis une montée en compétences techniques des membres, que ce soit en réseau ou encore à propos des fonctions *serverless*. Nous avons également pu mieux appréhender des outils comme Kubernetes et InfluxDB, qui sont souvent très utilisés dans le monde de l'entreprise pour de nombreux projets. Mais la particularité de ce projet est sa dimension environnementale : nous avons beaucoup appris à propos des normes météorologiques pour pouvoir récolter des données de qualité, mais surtout à interpréter les résultats des capteurs afin d'aller au delà des chiffres pour déterminer s'ils représentent un danger pour l'environnement.

Malgré la réduction de l'équipe dans la phase de mise en œuvre du projet et les changements dans la proposition d'amélioration du système, nous avons réussi à faire de Mycélium un système de surveillance du climat intelligent, fonctionnel et adaptable à différents contextes. Les fonctionnalités développées et l'infrastructure mise en place serviront de base à un futur groupe travaillant sur Mycélium.

Pour conclure, nous apprécions de travailler sur un projet avec un potentiel d'impact environnemental positif. Mycélium a donc été un projet très enrichissant, tant techniquement qu'environnementalement, pour toute l'équipe.

Références

[1] Projet ConnecSenS : voir la documentation de ce projet ici

[2] Minikube : outil facilitant l'exécution locale de Kubernetes. Il permet l'exécution d'un *cluster* Kubernetes à nœud unique dans une machine virtuelle (VM), sur une seule machine physique. Minikube est très utile pour les tests et le développement. Voici le guide d'installation

[3] Kubernetes : moteur d'orchestration de conteneur open source permettant d'automatiser le déploiement, la mise à l'échelle et la gestion des applications conteneurisées. Pour voir la documentation, cliquer ici

[4] Garden : outil qui peut s'exécuter via l'intégration continue. Il permet de codifier une description complète de la pile, y compris la façon dont elle est construite, déployée et testée : cela rend les *workflows* reproductibles et portables. Pour voir la documentation, cliquer ici

[5] Câblage et configuration d'un nœud ConnecSenS pour y interfacer un pluviomètre à auget LUFFT WTB100

[6] L'aide mémoire des commandes «kubectl»ici

[7] Le jeton GitLab La documentation de GitLab

[8] L'architecture de Gluster : Une brique est l'unité de base de stockage dans GlusterFS La documentation de GlusterFS

A Annexe 1 : Bilan de planification

Dans le rapport de planification est défini l'organisation du projet : il était prévu une certaine répartition des tâches entre les différentes personnes encore présentes au S8 :

- les tâches liées au nœud SoLo : elles ont pour but de le rendre plus autonome et de le faire réagir aux différents scénarios météorologiques qui peuvent survenir.
- les tâches liées au *cluster* : elles permettent de maintenir la stabilité de *cluster* et du réseau LoraWAN, ainsi que le déploiement automatique du *cluster*
- les tâches liées aux fonctions de traitement : elles permettent l'envoi de notifications personnalisées selon l'évènement climatique, le test de la qualité des données et la comparaison avec un historique.

Les objectifs fixés ont été atteints pour la plupart. Cependant, à cause de soucis techniques au niveau du *cluster* et de sa stabilité, et d'une prise en main des fonctions OpenFaaS plus longue que prévue, nous n'avons pas pu réaliser certaines tâches au moment où nous les avions prévues.

Beaucoup de tâches qui étaient prévues tout au long du semestre ont donc dû être réalisées lors des dernières semaines. Par exemple, le déploiement et test d'un premier scénario qui fonctionne de bout en bout, du déclenchement d'un envoi d'urgence du nœud SoLo à la réception d'une notification sur Discord, a été réalisé plus tard que prévu.

Le temps de travail a donc été beaucoup plus conséquent sur la fin pour respecter les délais imposés.

Au niveau du nœud SoLo, l'implémention du nouveau système d'alarmes n'a pas touché à la fin. Nous avons commencé par réfléchir une réalisation complexe mais qui est générale pour ajouter toutes les alarmes possibles. Mais nous avons sous-estimé la complexité d'implémentation de cette fonctionnalité. Nous aurions dû essayer de mettre en œuvre un scénario pour bien estimer la quantité de travail demandée.

B Annexe 2 : Manuel utilisateur

B.1 Pour créer un environnement de test pour développer des fonctions de traitement

Les démarches à effectuer la première fois :
- Cloner le projet ./Croix-Verte/Croix-Verte disponible sur GitLab, puis ouvrir un terminal dans le repertoire courant du projet : *cd ./Croix-Verte*
- Se référer au *Readme* du projet pour le reste de l'installation : Kubernetes et Minikube.

Les commandes à lancer à chaque fois qu'on veut tester :
- minikube start : permet de lancer un *cluster* Kubernetes local
- garden dev : permet de déployer l'environnement sur le *cluster* local : cette commande peut prendre un peu de temps

Les interfaces utiles une fois Garden[4] déployé grâce à la commande *garden dev* :
- Garden[4] : cela permet d'accéder aux logs pour voir les bugs.
- Grafana : cela permet de visualiser en temps réel les données qu'on a ajouté.

Les ports de ces différentes interfaces sont disponibles dans les commentaires et le readme du code *./croix-verte/croix-verte*.

Pour simuler l'envoi d'une trame via *chirpstack-simulator* :
- Cloner le projet ./mycelium/chirpstack-simulator disponible sur GitLab, puis ouvrir un terminal dans le répertoire courant du projet : *cd ./chirpstack-simulator*
- Lancer le script main.go : *go run main.go*

Pour ajouter des données manuellement dans InfluxDB :
- une fois la commande *garden dev* lancée, ouvrir un second terminal dans le dossier local *./croix-verte*
- taper la commande suivante : *influx -host localhost -port 8086 -username admin -password admin -precision rfc3339* afin de se connecter à Influx-DB.
- taper la commande suivante : *USE croix_ verte* pour se connecter à la base de données du projet.
- on peut maintenant rentrer des données. Par exemple, taper *INSERT my_ measurement humidity=30,temperature=25,atmoPressure=50,illuminance=33,accelX=1,accelY=1,accelZ=1* . Cela permet de rentrer une trame ayant pour valeur d'humidité 30, une température de 25... cette données sera ajouter à l'heure courante du système
- pour rentrer une données à une date différente : tout d'abord il faut convertir la date via https ://www.unixtimestamp.com/ par exemple puis ajouter le nombre obtenu (auquel on rajoute 9 chiffres random, par exemple 9 zéros) à la fin de la commande INSERT. Par exemple : *INSERT my_ measurement humidity=30,temperature=25,atmoPressure=50,illuminance=33,accelX=1, accelY=1,accelZ=1 1115234226000000000.*

B.2 Pour maintenir le *cluster*

Accès au *cluster* :
- Se référer au *Readme* du projet ./Croix-Verte/infrastructure/cluster disponible sur GitLab.

Connexion et vérification de l'état par l'outil en ligne de commande «kubectl»[6] :
- Si Minikube est déjà installé, il faut basculer entre l'environnement de Minikube qui se trouve sur le local et celui de *cluster* physique en utilisant la commande ;
  export KUBECONFIG=$(pwd)/kube_ nikos, à noter que pwd est le chemin du dossier .kube, si Minikube est déjà installé le chemin par défaut sera /.kube, et «kube_nikos» est le fichier utilisé pour configurer l'accès au *cluster* qui peut être acquis à l'étape précédente ;
- Pour vérifier les états de tous les pods, services et déploiements, nous pouvons utiliser la commande *kubectl get all -A* ;
- Visualisation des détails de chaque nœud de Raspberry Pi sera fait par *kubectl top node* ;

— Affichage de l'état verbeux de la ressource : *kubectl describe nomDeRessource -n namespaceDeRessource* ;
— Le journal plus détaillé affichera via la commande *kubectl logs nomDeRessource -n namespaceDeRessource*, ceux qui appartiennent aux pods sont tous disponibles sur GitLab : ./Croix-Verte/Croix-Verte/Monitor/Logs ;
— Pour éviter toute confusion entre l'environnement local et l'environnement physique, il faut quitter de l'environnement du *cluster* phisique après la fin de l'opération par la commande *unset KUBECONFIG*.

Déploiement de nouvelles fonctions sur *cluster* :
— Créer un jeton GitLab[7] pour obtenir un accès en écriture à la registre d'images, se référer au *Readme* du projet ./Croix-Verte/Croix-Verte, la partie de *Deploying to production* ;
— Connecter à la registre d'images par la commande *docker login gitlab.insa-rennes.fr :5050* ;
— Connecter au *cluster* par la commende *export KUBECONFIG=$(pwd)/kube_nikos* qui a été expliquée ci-dessus ;
— Sous le répertoire ./Croix-Verte/Croix-Verte, utiliser ; *garden –env=nikos deploy* pour le déploiement, puisque cette commande déploie le code actuellement affiché, vérifiez bien d'être sur la bonne branche et d'être sur le bon commit.

Connexion et vérification de l'état des volumes par une connexion SSH :
— Si le *cluster* a de l'exception de données, par exemple si les données dans Grafana disparaissent, il faut connecter et intervenir sur le *cluster* en utilisant la clé SSH générée dans la partie «Accès au *cluster*» par la command *ssh -i .ssh/nikos nikos@adresseIPDuNœud*. Les adresses IP du nœud de Raspberry Pi peuvent être retrouvées dans le fichier ./Croix-Verte/Infrastructure/cluster/inventory_eth.ini pour la connexion via la câble Ethernet et le fichier ./Croix-Verte/Infrastructure/cluster/inventory_wlan.ini pour la connexion via le WiFi «IOTINFO» ;
— Pour vérifier l'état de tout les volume, il faut aller dans le répertoire /mnt sous le *cluster*, utiliser la commande *sudo gluster volume status* Si la majeure partie des briques[8] sont hors ligne pour un volume, cela provoquera des anomalies dans ses données. À des fins de débogage, se référer au *Readme* du projet ./Croix-Verte/Croix-Verte, la partie de «Troubleshooting».

B.3 Pour configurer et mise à jour le nœud SoLo

Connexion et configuration du Nœud SoLo :
— Assurer que le nœud SoLo est éteinte et la connecter à un ordinateur via USB ;
— Supprimer le fichier *log* et les fichiers «.csv» pour ne pas surcharger la mémoire ;
— Définir la configuration dans le fichier «config.json» ;
— Définir la date en utilisant le fuseau horaire UTC.

Mise à jour du firmware du Nœud SoLo :
— Cloner le projet ./ConnecSens/Boitier disponible sur GitLab ;
— Utiliser une IDE propre pour les systèmes embarqués comme Attolic Truestudio ;
— Modifier le firmware qui se trouve dans le dossier firmware_boitier/software/embedded/application ;
— Assurer que le nœud SoLo est éteint et le connecter à un ordinateur via USB ;
— Compiler le projet et copier le fichier application.bin qui est dans le dossier firmware_boitier/software/embedded/application/debug ;
— Coller le fichier sur la carte SD et renommer le fichier comme «FIRMWARE.BIN» ;
— Suivre les étapes de configuration.