

## PROJET MYCELIUM 2.0

Rapport Final

### Equipe projet

BOIZUMAULT Maïwenn  
JULIEN Léo-Paul  
TUMOINE Ivy

### Encadrant

PARLAVANTZAS Nikolaos

### Avec la participation de

LONGUEVERGNE Laurent et MOUREAU Julien  
- *Université de Rennes*

CAROFF Nicolas, RAOUL Thibaut, VILLE Grégoire  
- *INSA Rennes*

11 avril 2023



## Remerciements

Nous tenons tout d'abord à remercier **M. Nikolaos PARLAVANTZAS**, notre encadrant à l'INSA, pour son suivi, sa forte implication et ses nombreux conseils réguliers tout au long du projet.

Nous souhaitons également remercier nos encadrants extérieurs, travaillant à l'OSUR, pour leur aide. Merci à **M. Julien MOUREAU** pour sa présence régulière aux réunions et pour son expertise sur les différentes technologies. Merci aussi à **M. Laurent LONGUEVERGNE** de nous avoir proposé ce projet et donné le contexte de celui-ci.

Enfin, nous remercions tous les membres de l'équipe de Mycélium pour l'intérêt qu'ils ont porté en le bon déroulement du projet. Merci aux membres de Mycélium 2.0 - **Nicolas CAROFF, Thibaut RAOUL et Grégoire VILLE** - partis au second semestre étudier à l'étranger, pour leur investissement et leur présence, même à distance. Mais merci aussi à ceux de Mycélium 1.0, et en particulier à **Guillaume CHAUVEAU**, pour leur aide et leurs conseils techniques précieux, qui nous ont permis d'avancer.

# Table des matières

<b>1 Glossaire</b>	<b>3</b>
<b>2 Introduction</b>	<b>4</b>
<b>3 Mises à jour et difficultés rencontrées</b>	<b>5</b>
3.1 Les nouveaux capteurs . . . . .	5
3.1.1 FoxyFind . . . . .	5
3.1.2 Capteurs de l'OSUR . . . . .	5
3.2 Le cluster . . . . .	7
3.3 Le VPS . . . . .	7
3.4 Communication entre la gateway et le cluster . . . . .	8
3.5 Architecture logicielle . . . . .	9
<b>4 Etat de finalisation du projet</b>	<b>10</b>
4.1 Scénarios traités . . . . .	10
4.2 Scalabilité avec le cloud . . . . .	11
4.3 Architecture logicielle . . . . .	12
<b>5 Compte rendu des phases de test</b>	<b>14</b>
5.1 Tests des scénarios . . . . .	14
5.1.1 Normales Saisonnieres . . . . .	14
5.1.2 Orage . . . . .	14
5.1.3 FoxyFind . . . . .	14
5.2 Tests de l'architecture . . . . .	15
5.2.1 Ajout du chemin vers le VPS . . . . .	15
5.2.2 Ajout des nouvelles fonctions . . . . .	15
<b>6 Pistes pour la suite</b>	<b>16</b>
6.1 Réduction de la stack technique . . . . .	16
6.2 Changements matériels . . . . .	16
6.3 Améliorations logicielles . . . . .	17
<b>7 Conclusion</b>	<b>18</b>
<b>8 Annexes</b>	<b>19</b>
8.1 Annexe A : Bilan de planification . . . . .	19
8.2 Annexe B : Manuel de l'utilisateur . . . . .	20
8.2.1 Utilisation de l'environnement de test Garden . . . . .	20
8.2.2 Création d'une fonction OpenFaaS sur sa machine personnelle . . . . .	20
8.2.3 Création d'une fonction OpenFaas pour Mycélium . . . . .	20
8.2.4 Utilisastion du VPS . . . . .	20
8.2.5 Connexion au VPS . . . . .	21

# 1 Glossaire

- **ChirpStack** : logiciel open-source pour la gestion des appareils utilisant LoRaWAN ;
- **FaaS** : *Function as a Service*, modèle de déploiement de cloud computing en microservices serverless ;
- **Gateway** : passerelle de communication entre deux réseaux distincts ;
- **GlusterFS** : système de fichiers distribués open-source qui permet de combiner plusieurs serveurs de stockage en un seul système de fichiers virtuel, fournissant ainsi un stockage évolutif et résilient pour les applications d'entreprise ;
- **IoT** : *Internet des objets*, décrit le réseau d'objets physiques constitués de capteurs, de logiciels et d'autres technologies dans le but de connecter et d'échanger des données avec d'autres appareils et systèmes sur Internet ;
- **LoRaWAN** : protocole de communication basse consommation, longue portée et faible bande passante très utilisé pour l'IoT. Il se base sur le réseau LoRa (Long Range) ;
- **MQTT** : *Message Queuing Telemetry Transport*, protocole de messagerie publish-subscribe basé sur le protocole TCP/IP. Ce protocole très léger est énormément utilisé en IoT. Un broker est chargé de recevoir et de répartir les messages des clients ;
- **OpenFaaS** : logiciel open-source pour les fonctions serverless sur le cloud ;
- **Raspberry Pi** : ordinateur peu coûteux de la taille d'une carte de crédit, pouvant effectuer toutes les actions d'un ordinateur classique. Sa taille, son faible coût, les modules complémentaires et sa communauté en font un outil de choix pour les projets d'IoT ;
- **Scalabilité** : capacité d'un système à augmenter ou diminuer ses ressources en fonction de l'usage de ce système ;
- **Serverless** : modèle d'exécution pour les applications sans gestion de serveur par l'utilisateur, avec des coûts payés selon l'utilisation ;
- **VPS** : *Virtual Private Server*, machine virtuelle, créée sur un serveur physique et employant ses ressources pour offrir à ses utilisateurs les mêmes fonctionnalités qu'un serveur dédié.

## 2 Introduction

Dans un contexte actuel de crise climatique, la ville de Rennes s'est engagée à renaturer d'anciens espaces verts. Ainsi, le projet **Mycélium** se concentre sur le suivi de la **renaturation de la Croix Verte** (*figure 1*) située sur le campus de Beaulieu, face à l'INSA. Le projet Mycélium est réalisé en collaboration avec le laboratoire de Géosciences de Rennes et l'Observatoire des Sciences et de l'Univers de Rennes, l'OSUR [1]. Il vise à suivre au mieux l'évolution de la zone de la Croix Verte en y installant un maillage de capteurs et en traitant les informations issues de ces capteurs.

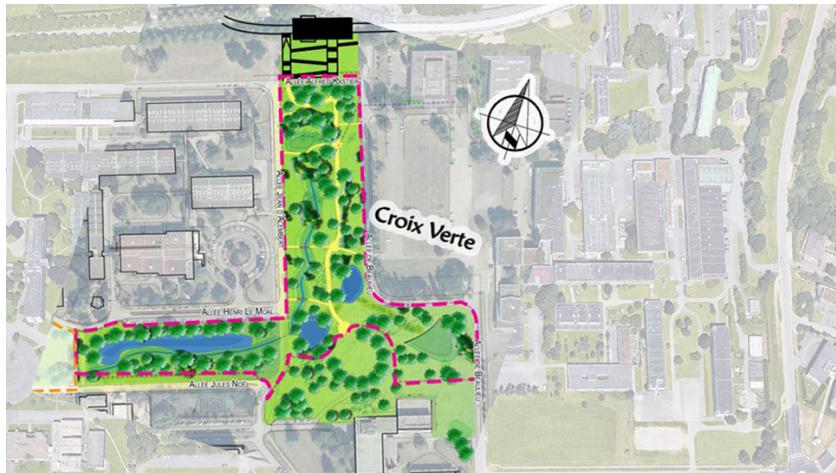


FIGURE 1 – La Croix Verte

A plus grande échelle, notre travail s'inscrit dans le projet de suivi environnemental national TERRA FORMA [2] coordonné par Laurent LONGUEVERGNE. TERRA FORMA vise à partager un état des lieux de nos ressources naturelles (capital sol, eau, biodiversité...) continuellement mis à jour.

Mycélium 2.0 est la reprise du projet de l'année précédente Mycélium 1.0 [3]. Lors de ce projet, nos prédecesseurs ont déployé un premier système collectant des données sur la Croix Verte. Ce premier système était composé d'un noeud de capteurs basse consommation communiquant via le réseau LoRa, d'une gateway recevant les données récoltées et d'un cluster de Raspberry Pis [4], chargé de traiter ces données et d'alerter les utilisateurs finaux en cas de mesures exceptionnelles. Prenons un exemple : lorsque la température est inférieure à 0°C et que le taux d'humidité est important, l'utilisateur final est notifié de la forte probabilité de chutes de neige. Les critères de mesures exceptionnelles définissent ce que l'on nomme des "scénarios", l'exemple précédent correspondant au scénario neige. Les enjeux de ce projet étaient d'obtenir un système basse consommation et adapté à l'environnement afin qu'il puisse durer dans le temps. L'objectif de Mycélium 2.0 est alors d'**améliorer** et d'**étendre le système précédent**, notamment en y intégrant de **nouveaux capteurs** et de **nouveaux scénarios** de mesures exceptionnelles.

Ce rapport final vise à décrire tout d'abord les modifications réalisées relativement aux rapports de spécifications fonctionnelles et de conception, dues aux difficultés rencontrées. Ensuite, il sert à montrer l'état de finalisation du projet puis de faire un compte rendu des phases de tests pour valider nos réalisations. Enfin il permettra d'envisager des pistes pour une éventuelle future reprise de projet.

### 3 Mises à jour et difficultés rencontrées

Cette partie a pour but d'expliquer les différences entre l'état final du projet et l'état souhaité décrit dans les précédents rapports de spécifications fonctionnelles et de conception. Les modifications et difficultés rencontrées en lien avec l'ajout de nouveaux capteurs, le cluster, le VPS, la communication entre la gateway et le cluster et enfin l'architecture logicielle seront présentées.

#### 3.1 Les nouveaux capteurs

##### 3.1.1 FoxyFind

Le premier capteur ajouté au réseau installé sur la Croix Verte est le capteur FoxyFind, né du projet IoT du même nom qui a servi de "POC" (Prove Of Concept) à Mycélium 2.0. Il a donc été conçu de toute part par des étudiants de l'INSA. Il est constitué d'une caméra, couplée à un Raspberry Pi, comme représenté dans la *figure 2*. Après plusieurs efforts d'amélioration du modèle, on intègre ainsi à Mycélium des capteurs vidéo afin de détecter et de reconnaître la faune indigène à la Croix Verte et plus particulièrement un éventuel renard sur la zone.

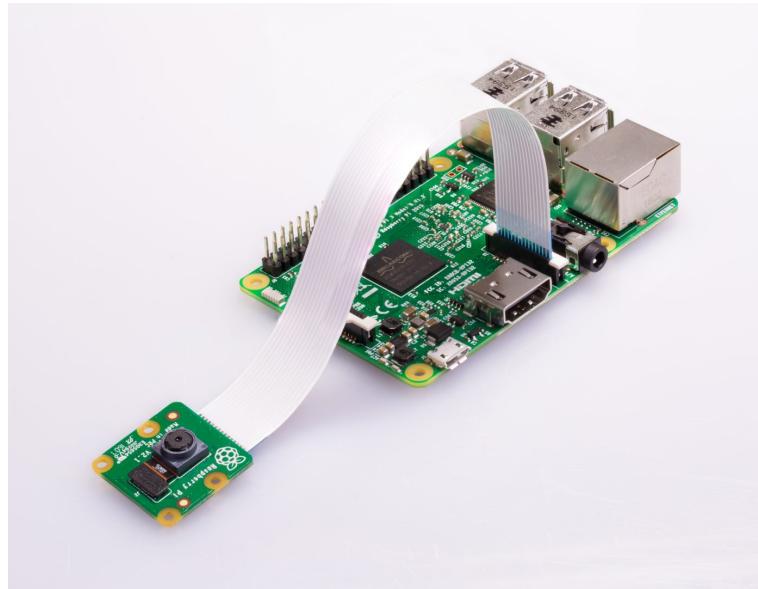


FIGURE 2 – FoxyFind : une caméra reliée à un Raspberry Pi

##### 3.1.2 Capteurs de l'OSUR

L'OSUR a également mis à notre disposition un certain nombre de capteurs. Ceux-ci consistent en un capteur multiparamètre, un débitmètre et de nouveaux noeuds SoLo : leur utilisation permet de mieux comprendre la zone de la Croix Verte. D'une part, ils permettent l'ajout de nouvelles mesures : le débitmètre et le capteur multiparamètre fournissent en effet des informations sur les cours d'eau de la Croix Verte, qui n'étaient pas relevées auparavant. D'autre part, ils améliorent la qualité des mesures : ajouter des noeuds SoLo permet de mieux couvrir la Croix Verte mais aussi d'avoir une meilleure confiance en les données récoltées en comparant celles des différents capteurs entre elles.

Concernant le capteur multiparamètre et le débitmètre illustrés en *figure 3*, le but était que l'OSUR se charge de déployer ces capteurs et de poster les mesures sur un serveur. Nous aurions alors eu accès à ces données en effectuant des requêtes HTTP auprès du serveur de manière régulière. Actuellement, les capteurs sont déployés sur la Croix Verte, à l'emplacement décrit en *figure 4* et l'OSUR dispose d'un serveur géré par l'Université de Rennes sur lequel les mesures sont envoyées. Malheureusement, cet état d'avancement n'a été atteint que très récemment et ce au prix de nombreux et longs échanges avec la DSI de l'université de Rennes. De ce fait, il est très peu probable que l'accès à ce serveur nous soit donné avant la fin du projet. Nous n'avons pas non plus eu la possibilité d'ajouter de nouveaux noeuds SoLo, à la fois à cause d'un manque de temps et d'un manque de communication sur ce sujet avec l'OSUR.



FIGURE 3 – Capteur multiparamètre et débitmètre de l'OSUR

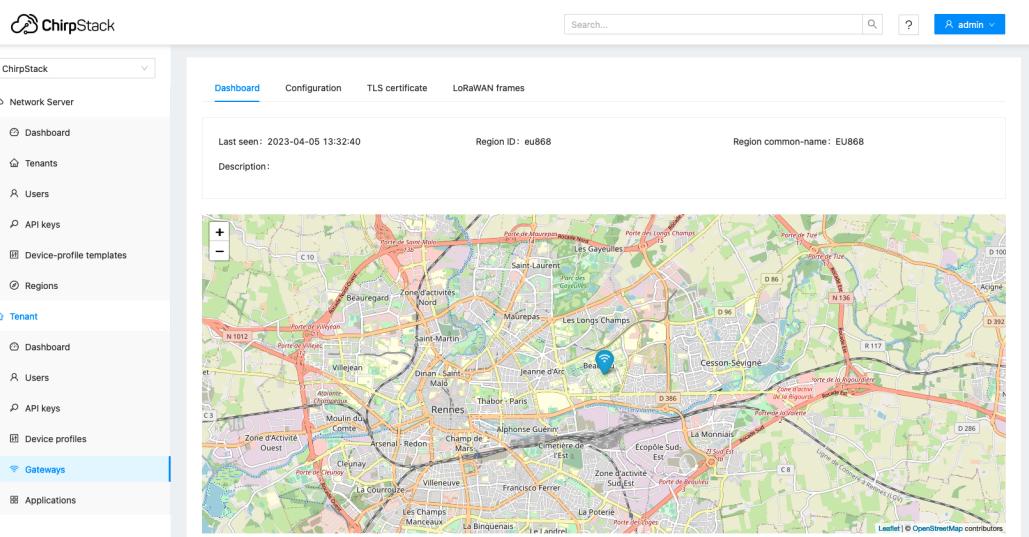


FIGURE 4 – Emplacement des capteurs de l'OSUR

### 3.2 Le cluster

Le cluster de Raspberry Pis est le noeud de calcul principal de Mycélium, c'est sur celui-ci que sont déployées toutes les applications de Mycélium permettant la sauvegarde et le traitement des données. Petit et peu énergivore, le choix de ce matériel, décrit en *figure 5* a été contraint par le cadre extérieur et le caractère autonome du projet.

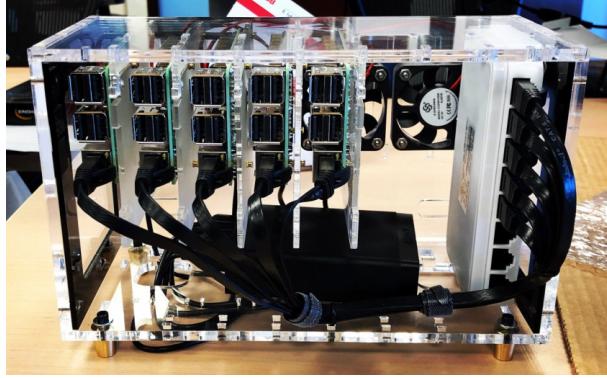


FIGURE 5 – Cluster de Mycélium

Dans l'état actuel des choses, le cluster est fonctionnel : il est capable de recevoir les paquets du noeud SoLo, d'enregistrer les données dans InfluxDB et de faire tourner les fonctions OpenFaaS.

Avant d'avoir pu être remis en fonction, il a cependant été source de difficultés tout au long du projet. Tout d'abord, suite à des travaux visant à moderniser les réseaux de l'INSA, le réseau Wi-Fi spécifique sur lequel se connectaient les Raspberry du cluster a été remplacé, obligeant la migration vers le nouveau réseau Wi-Fi insaIOT. Cela a nécessité de reprendre la configuration réseau et communiquer avec la DSI afin de déclarer les appareils et faire des demandes d'accès pour le cluster mais aussi pour nos ordinateurs personnels. De plus, il a fallu gérer certains problèmes de certificats expirés, en effectuant une fois de plus des demandes auprès de la DSI. Ces changements ont alors engendré de nouveaux problèmes et le cluster n'était toujours pas fonctionnel. Nous avons dû refaire toute la partie liée au déploiement des fonctionnalités du cluster. Cela concernait notamment la gestion de GlusterFS, qui permet de mettre en place un filesystem distribué entre toutes les Raspberry, et celle des accès aux repositories de GitLab. En effet, les fonctions OpenFaaS sont importées sur le cluster via des images Docker stockées sur un repository GitLab, malheureusement ce repository n'est plus accessible depuis le nouveau réseau insaIOT : un nouveau problème à résoudre avec la DSI est alors apparu.

En tout état de cause, bien qu'il soit finalement fonctionnel, le cluster est bien trop instable, rendant son utilisation trop hasardeuse pour réellement s'en servir.

### 3.3 Le VPS

Le VPS sert à décharger le cluster lorsque ce dernier n'est plus en mesure de traiter les données qui lui parviennent, ce qui est malheureusement fréquemment le cas. A l'origine, le VPS devait seulement servir de machine pour effectuer des tests sur le projet, mais au vu de la surcharge fréquente du cluster, son objectif a évolué. Cette évolution de fonction a entraîné une évolution des caractéristiques du VPS, avec l'ajout d'un cœur, de RAM, de stockage, et l'ouverture de certains ports.

La principale difficulté rencontrée a été le mode de communication avec le VPS. En effet, la communication au sein du cluster et entre la gateway et le cluster utilise le protocole MQTT. Or, plutôt que d'utiliser de la même manière le protocole MQTT, le VPS n'autorise la communication que par

le protocole MQTTS. Le protocole MQTTS est le pendant du protocole MQTT avec TLS : ce mode de communication demande des certificats signés du côté client et du côté broker. Il est alors impliqué que les communications sont bien plus sécurisées, mais c'est aussi évidemment plus difficile à mettre en place que le simple protocole MQTT. S'en sont alors une fois de plus suivis de longs échanges avec la DS1, ce qui a retardé l'avancée du projet.

Une autre difficulté non négligeable était que le VPS est très différent du cluster dans le sens où il n'est constitué que d'un seul noeud : on ne peut donc pas considérer le VPS comme un second cluster. Cela implique que le VPS ne peut pas être un "miroir" parfait du cluster. Puisqu'il ne s'agit que d'un seul noeud isolé, l'utilisation de Kubernetes n'est pas adaptée, or OpenFaas a besoin de Kubernetes pour fonctionner. Il a donc fallu complètement revoir l'architecture logicielle du VPS.

### 3.4 Communication entre la gateway et le cluster

Comme expliqué ci-dessus, la communication entre la gateway MultiConnect Conduit de Multitech illustrée en *figure 6* et le cluster est réalisée avec le protocole MQTT. On utilise en particulier le ChirpStack Gateway Bridge et ChirpStack afin d'observer les paquets transférés entre les deux modules.



FIGURE 6 – Gateway MultiConnect Conduit

Idéalement, il était souhaitable d'installer tous ces modules sur le cluster, afin de centraliser toute la configuration. Pour ce faire, la configuration a d'abord été modélisée sur une machine virtuelle avant de tenter de la lancer sur le cluster. Cela a permis de minimiser les erreurs de configuration finale, et de prévoir des éventuels problèmes. Une adresse IP statique a d'abord été configurée sur le poste de test, car le protocole DHCP est désactivé sur la gateway.

Cependant, il a été déduit que des difficultés seraient rencontrées lors d'une installation du bridge sur le cluster, lié au souci de mémoire et aux problèmes relatifs au cluster énoncés précédemment. Une alternative a alors été choisie : installer le ChirpStack Gateway Bridge directement sur la gateway. Pour cela, il a fallu établir une connection SSH avec la gateway, afin de modifier les fichiers de configuration. Or cette gateway n'étant pas connectée à Internet, il a fallu envoyer les nouveaux fichiers nécessaires pour les installations par SCP, car une connexion au Wi-Fi insaIOT aurait été plus complexe à mettre en place. ChirpStack est quant à lui installé sur le cluster, et ainsi les paquets transférés peuvent être suivis.

### **3.5 Architecture logicielle**

Afin de rendre compte de l'ajout de nouveaux capteurs ainsi que du VPS, il a fallu refaire l'architecture logicielle du projet. Le développement de la nouvelle architecture logicielle a été victime du manque de temps dû aux problèmes énoncés précédemment. L'architecture logicielle finale est alors moins travaillée que celle présentée dans le rapport de Conception, mais elle est fonctionnelle.

## 4 Etat de finalisation du projet

Après avoir expliqué les modifications à l'état initialement souhaité du projet, cette partie rend compte de son état final : y sont présentés les scénarios traités, la scalabilité avec le Cloud mis en place, et l'architecture logicielle actuelle.

### 4.1 Scénarios traités

Actuellement, tous les scénarios implémentés l'année dernière ont été remis en fonctionnement. À ces scénarios s'ajoutent ceux développés cette année : le scénario FoxyFind, le scénario Normales Saisonnières et le scénario Orage. Tous les scénarios sont implémentés à la fois sur le cluster et sur le VPS.

Le scénario **FoxyFind** permet de détecter et de reconnaître les animaux évoluant au sein de l'environnement de la Croix Verte. Il se déroule de la façon suivante : la caméra reliée à un Raspberry Pi détecte un changement significatif entre deux photos consécutives, la photo la plus récente est envoyée via MQTT au cluster. Grâce à une fonction OpenFaaS, ce dernier statue sur la présence d'un animal et envoie une notification Discord aux utilisateurs. Si jamais le cluster est saturé, il retransmet alors la photo via MQTTs au VPS qui se charge de détecter la présence d'un animal et d'envoyer une notification.

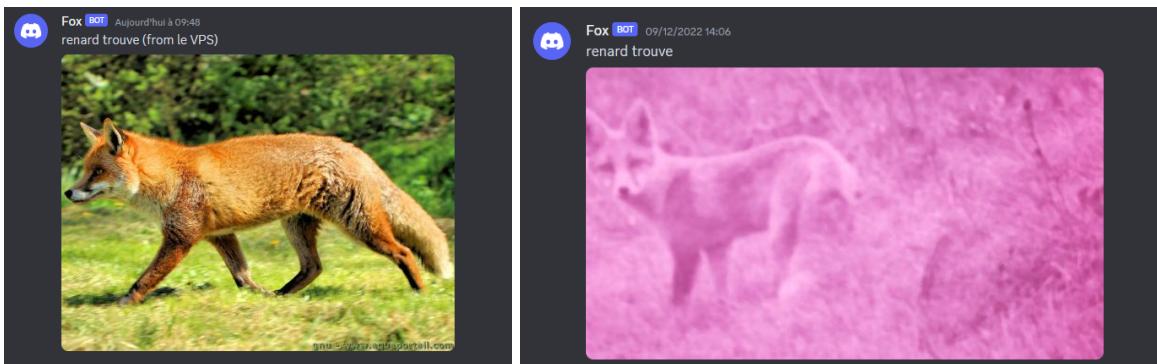


FIGURE 7 – Notifications Discord FoxyFind, en provenance du VPS et du cluster.

Le scénario **Normales Saisonnières** permet d'avertir l'utilisateur si la moyenne des températures de la journée sort de l'intervalle des températures normales pour la période courante. Ce scénario a été implémenté grâce à une fonction OpenFaaS et plus particulièrement grâce au *cron-connector* d'OpenFaaS qui permet d'appeler la fonction tous les jours.

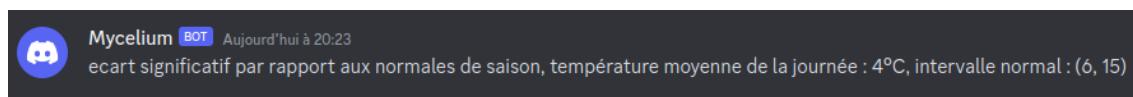


FIGURE 8 – Notification Discord pour le scénario "Normales Saisonnières".

Le scénario **Orage** permet d'avertir l'utilisateur si un épisode orageux se produit aux environs de la zone de la Croix-Verte. Tout comme les deux fonctions précédentes, celle-ci a été mise en place via une fonction OpenFaaS. N'ayant pas pu installer les capteurs de l'OSUR, le scénario ne repose que sur des capteurs déjà installés. Ce scénario repose sur une fonction OpenFaas dont le comportement est le même que pour les scénarios implantés l'année dernière : l'arrivée de données en provenance des capteurs entraîne ou non l'envoi de notifications en fonction de seuils prédéfinis.

## 4.2 Scalabilité avec le cloud

Actuellement, le cluster est très instable : en effet, sa ressource en CPU devient très vite problématique puisque cette ressource est déjà à moitié utilisée lorsque le cluster ne fait aucun traitement. Cette ressource est utilisée par les simples installations minimales nécessaires au fonctionnement du cluster dans la configuration souhaitée pour le projet. C'est pourquoi il a été décidé de développer de la **scalabilité dans le Cloud** : c'est la capacité de modifier la quantité de ressources informatiques selon les besoins pour justement répondre à l'évolution de la demande. C'est à ce besoin que répond le VPS.

Un miroir simplifié de l'architecture de Mycélium est déployé sur le VPS : une base de données InfluxDB, un broker MQTTs et une application Python. Ces trois éléments sont déployés dans des conteneurs Docker, eux-mêmes contenus dans un Docker Network, qui est un sous-réseau permettant aux conteneurs de communiquer entre eux, comme schématisé en *figure 9*. Le conteneur *InfluxDB* contient notre base de données. Le conteneur *MQTTs* contient un broker MQTT configuré pour n'accepter que des messages signés via TLS sur le port 8083. Le port 8083 du conteneur *MQTTs* est à cet effet redirigé vers le port 8083 du VPS afin de permettre la réception de messages ne provenant pas de clients internes au Docker Network. Lorsque le cluster dépasse les 70% d'utilisation de la ressource en CPU, la fonction *Controller* redirige les données vers le VPS en publiant les frames décodés sur le broker MQTTs du VPS, comme représenté en *figure 10*. Une application Python est déployée dans le conteneur *App* : elle contient les fonctions nécessaires aux scénarios, à l'enregistrement dans la base de données InfluxDB et à l'envoi de notifications Discord, et s'abonne au topic sur lequel le cluster a publié pour recevoir les frames décodés.

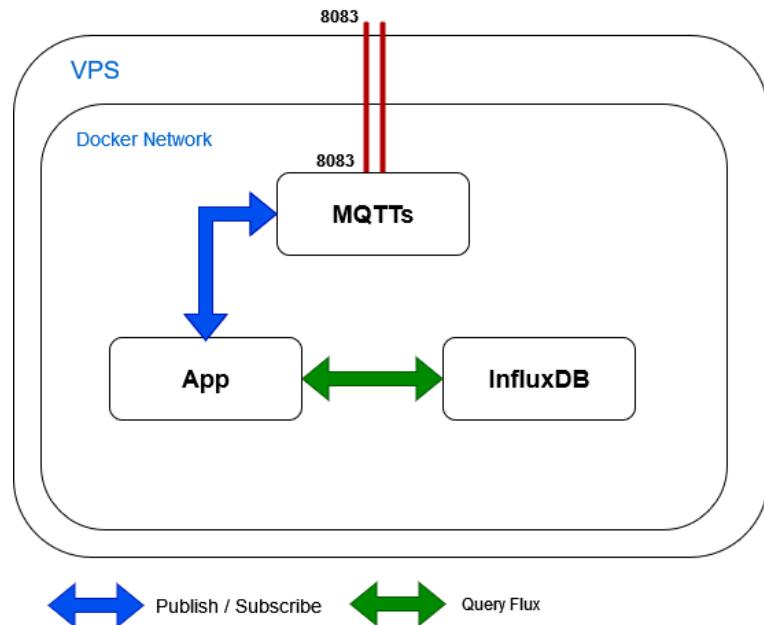


FIGURE 9 – Architecture du VPS

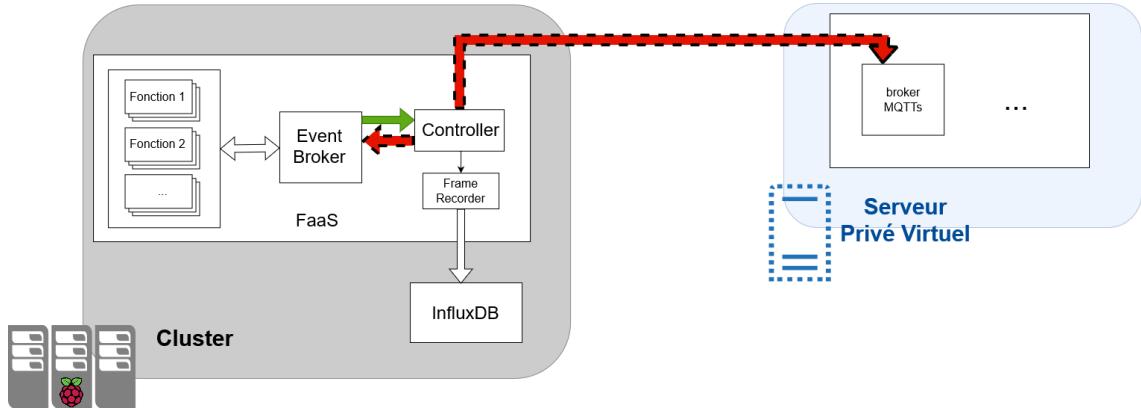


FIGURE 10 – Détail du transfert de charge entre le cluster et le VPS

### 4.3 Architecture logicielle

L’architecture logicielle de Mycélium 2.0 est illustrée en *figure 11*. On peut découper cette architecture en trois grandes parties en fonction du chemin suivi par les données.

La première partie concerne les données en provenance du Noeud SoLo et donc de ChirpStack : les données arrivent sur le controller via le topic *chirpstack/up*, elles sont ensuite décodées grâce à la fonction CNSSRF decoder, puis le controller décide de les envoyer soit sur le cluster soit sur le VPS. Ces données sont finalement reçues par d’autres fonctions permettant notamment de coder les scénarios et d’envoyer les notifications.

Le second chemin des données est celui de FoxyFind. Dans ce cas, les données en provenance du Raspberry Pi de FoxyFind arrivent directement sur la fonction FoxyFind qui se charge de décoder les données et de faire appel si besoin à la fonction notification.

Enfin la dernière partie concerne le scénario Normales Saisonnieres qui ne reçoit aucune donnée mais va les chercher directement dans InfluxDB. De nouveau, la fonction notification peut être appelée si les températures récupérées dans InfluxDB sortent des normales saisonnières.

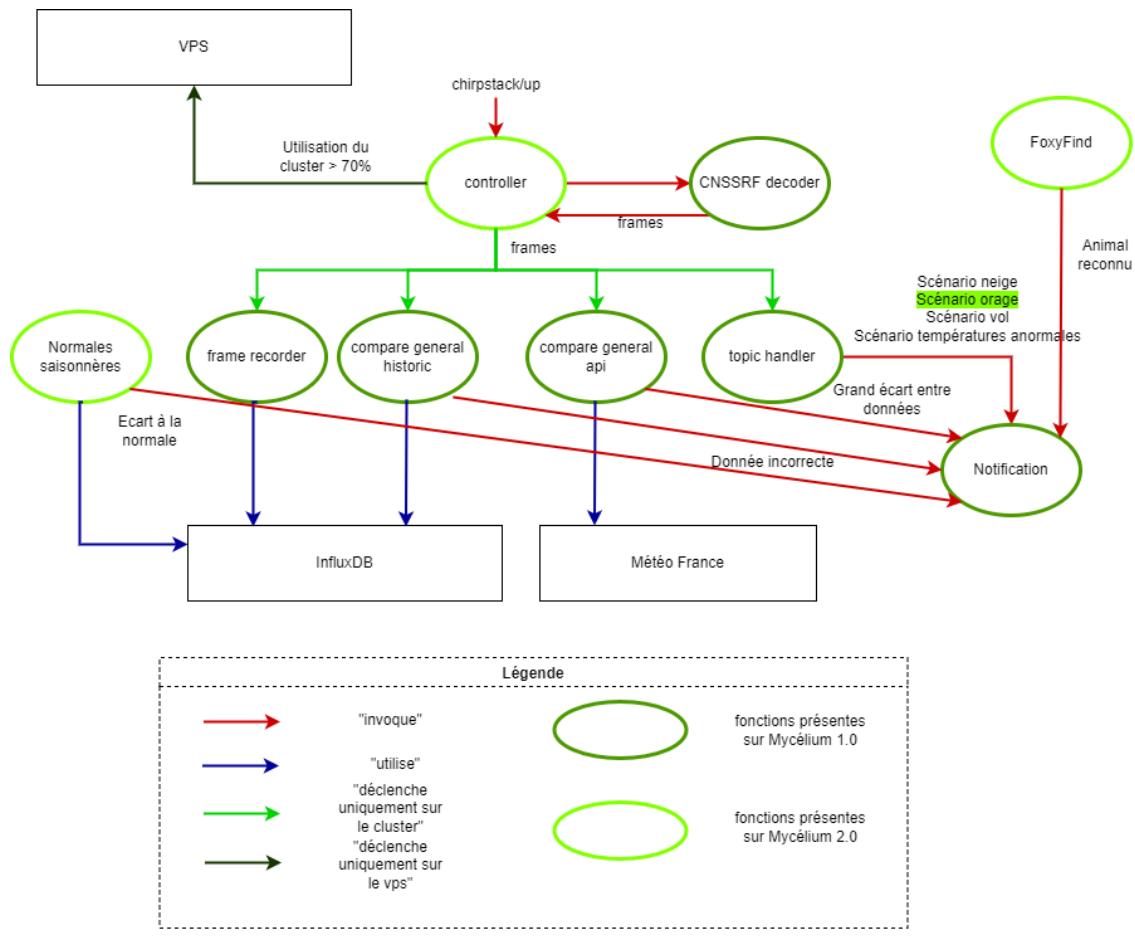


FIGURE 11 – Architecture logicielle de Mycélium 2.0

## 5 Compte rendu des phases de test

Chaque nouvelle fonctionnalité a été développée étape par étape. A chacune de ces étapes, des tests ont été effectués afin de vérifier le bon fonctionnement de la fonctionnalité. La dernière étape, consistant à intégrer la fonctionnalité dans l'ensemble existant, a également fait l'objet de tests de non-régression afin de vérifier que l'ajout de la dite fonctionnalité n'endommage pas celles déjà existantes.

### 5.1 Tests des scénarios

Les phases de test des scénarios ont permis de développer de nouvelles fonctionnalités pas à pas afin de finalement les intégrer entièrement à Mycélium. Les scénarios déjà implémentés l'an dernier ont fait l'objet de tests d'intégration : nous avons directement testé le bon déroulement de ces scénarios dans des conditions réelles, avec la gateway et le cluster. Les tests des scénarios développés cette année vont être décrits dans les parties suivantes.

#### 5.1.1 Normales Saisonnieres

Le développement et les tests du scénario Normales Saisonnieres ont été effectués en plusieurs étapes. La première étape a été de créer un prototype très simple comprenant une base de données InfluxDB et une fonction Python chargée de récupérer les températures des dernières 24h dans la base, de vérifier que la moyenne des températures ne sort pas de l'intervalle des normales saisonnières, et d'envoyer une notification Discord si jamais ce n'est pas le cas. La seconde étape a été de développer le même prototype mais cette fois-ci avec OpenFaaS. L'ajout d'OpenFaaS a également permis d'utiliser le *cron-connector* d'OpenFaaS pour appeler la fonction chaque jour, et une fois que le tout était fonctionnel, le déploiement de la fonction sur le cluster n'a pas présenté de souci particulier. La version déployée sur le VPS est plus proche du premier prototype car elle ne passe pas par OpenFaaS, cependant elle est déployée sur un conteneur Docker, et l'appel régulier de la fonction est alors géré par la librairie *Schedule* de Python.

#### 5.1.2 Orage

Le développement et les tests du scénario Orage a suivi le même schéma de développement que le scénario précédent. Ainsi, la première étape a été de développer un prototype simple mais répondant au cahier des charges de la fonction. Tout d'abord, il a fallu créer un broker MQTT et une fonction Python s'abonnant à des topics de ce broker et capable d'envoyer une notification avec les données. Puis il a fallu ajouter les seuils à la fonction et réaliser des tests avec des données factices afin de simuler un épisode orageux. Le passage d'une fonction Python à une fonction OpenFaas a été plutôt simple : la fonction s'abonne aux topics dont elle a besoin et elle est appelée à chaque arrivée de messages sur les topics. Ainsi, le scénario "Orage" permet de mesurer un épisode orageux et de notifier les utilisateurs lorsqu'un épisode de ce type se produit.

#### 5.1.3 FoxyFind

Afin de valider le scénario FoxyFind, il a fallu valider de nombreux tests. En effet l'intégration du projet IoT au projet a été faite de façon empirique et à chaque étape, des tests de validation ont été menés. Tout d'abord, il a fallu valider le fait qu'OpenFaaS pouvait recevoir un flux de bytes sous forme d'une chaîne de caractère représentant une image depuis un unique topic et reconstruire l'image afin d'en faire une notification Discord. Pour cela, il a fallu déployer un MQTT-connector permettant de lier ce flux de bytes à un topic et de spécifier à la fonction OpenFaaS de récupérer uniquement les messages venant d'un topic spécifique. Puis la partie relative à la reconnaissance animale a été ajoutée avec Tensorflow qui est une librairie Python très utilisée en machine learning. Cette partie a été validée avec l'envoi d'une notification Discord et un dictionnaire de noms d'animaux afin de déterminer un pourcentage à partir duquel l'animal est susceptible d'être reconnu. Ainsi un seuil de 10% a été retenu

afin d'obtenir une reconnaissance correcte d'un animal.

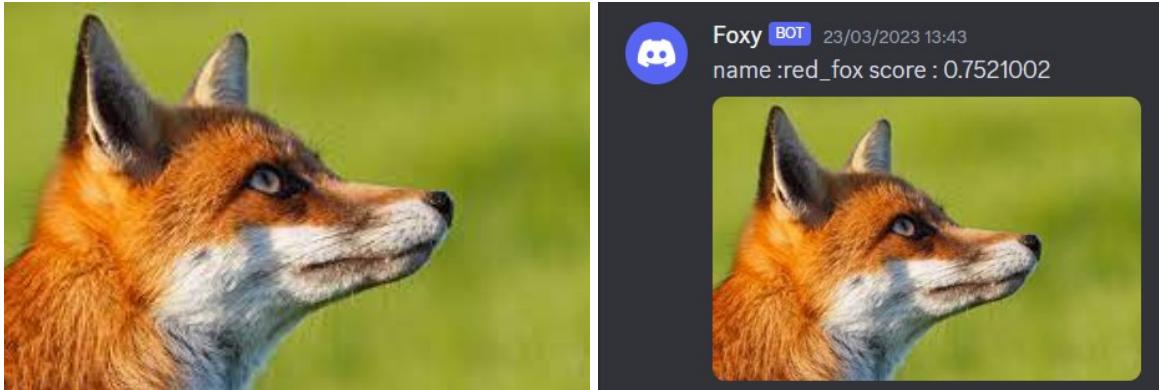


FIGURE 12 – Image d'origine de renard et la notification Discord associée.

Une fois ces deux premières étapes validées, il a été nécessaire d'intégrer le Raspberry Pi qui s'occupe de capturer en photo son environnement. Cette étape a été la plus rapide à faire et à valider comme la vérification et le paramétrage d'OpenFaaS et la capacité de la fonction d'envoi d'images à interagir avec OpenFaaS avaient été réalisés en amont.

Ainsi, à la fin de ces trois premières étapes, une fonction OpenFaaS de reconnaissance d'animaux et d'envoi de notifications Discord et une fonction de capture de photos tournant en continu sur un Raspberry Pi ont été liées. Il ne restait plus qu'à déployer la fonction OpenFaaS sur le cluster et le VPS à la fois. Les résultats de ce déploiement ont déjà été visualisés dans une partie précédente, à la *figure 7*. En outre afin de tenir compte de l'introduction d'OpenFaaS et d'éviter un phénomène d'encombrement sur toutes les composantes, il a été décidé de supprimer les photos une fois que celles-ci ont été traitées à la fois sur le Raspberry Pi et sur le cluster.

## 5.2 Tests de l'architecture

### 5.2.1 Ajout du chemin vers le VPS

Lors de l'ajout de nos modifications à l'architecture logicielle de Mycélium 1.0 et notamment l'ajout de la bifurcation vers le VPS en cas d'utilisation du cluster à plus de 70%, il a fallu tester si ce changement n'affectait pas le comportement initial de Mycélium. Pour cela, la fonction *controller* qui remplace la fonction *chipstack-handler*, permettant de déclencher toutes les fonctions déjà présentes dans Mycélium 1.0 à l'arrivée d'un frame de données, a été déployée. Puis les fonctions déjà présentes dans Mycélium 1.0 ont été invoquées avec comme scénario de test leur scénario propre afin de vérifier si le comportement avant modification de la fonction *chirpstack-handler* était le même après l'ajout de la fonction *controller*.

### 5.2.2 Ajout des nouvelles fonctions

Une fois le test de l'ajout du chemin du VPS fait, il a fallu vérifier si l'ajout des nouvelles fonctions n'engendrait pas de problèmes avec la nouvelle architecture. Ainsi leur ajout à l'architecture a été fait puis testé à la fois avec données fictives et celles transmises par le noeud SoLo.

## 6 Pistes pour la suite

Après avoir beaucoup appris de ce projet en travaillant dessus pendant toute une année, certains points d'amélioration possibles semblent importants à retenir pour la suite.

### 6.1 Réduction de la stack technique

La majeure source de difficulté et de freinage à l'avancée du projet s'est avérée être la pile de technologies nécessaires à la mise en place de l'écosystème de Mycélium. En effet, celle-ci est essentiellement imposée par l'utilisation du cluster de Raspberry Pis, OpenFaaS et Kubernetes, qui ne sont pas facilement compatibles. En complément, ces technologies ne sont pas abordés en cours en amont du projet, il est donc compliqué de tout appréhender dans son ensemble.

Le choix des Raspberry Pis est justifié par la petite taille et la basse consommation de celles-ci, permettant de rendre l'ensemble adaptable à l'environnement et plutôt autonome. Cependant, la capacité de stockage et de calcul ainsi que la quantité de celles mises à notre disposition sont insuffisantes par rapport à tout ce qui est déployé dessus. Rappelons que la moitié l'espace de stockage est de chaque Raspberry est déjà utilisé par tous les logiciels installés nécessaires au fonctionnement du projet, sans qu'elles ne tournent.

Parmi ces logiciels figure notamment Kubernetes, nécessaire pour orchestrer les différents noeuds du cluster, chacun constitué d'un Raspberry. Cependant, Kubernetes est extrêmement couteux en termes de calcul, notamment face à la capacité des Raspberry. Ainsi se présente une première incompatibilité, qui n'a pas pu être évitée en respectant la stack.

De plus, l'utilisation d'OpenFaaS est intéressante car elle permet de gérer le lancement de simples fonctions déclenché de certains événements. Cela permet aussi de découvrir une technologie récente, utile, et son fonctionnement. Cependant, on se rend rapidement compte que bien que les fonctions ne soient appelées qu'au besoin, d'autres fonctions intrinsèques comme le MQTT-connector doivent tourner en continu pour justement réveiller les fonctions FaaS souhaitées. Ainsi, il s'agit d'une nouvelle technologie à appréhender et comprendre, qui n'est peut-être pas si économique que l'on pourrait le penser.

Il paraît donc nécessaire de modifier la stack technique du projet, afin de la réduire et rendre le projet Mycélium plus accessible et plus robuste.

### 6.2 Changements matériels

Pour réduire la stack et rendre le projet moins complexe en terme de configuration, des changements matériels peuvent être effectués.

Comme vu précédemment, la capacité des Raspberry Pis est limitée, limitant donc rapidement le projet en lui-même. Il existe plusieurs solutions pour améliorer les capacités du cluster.

En restant dans le domaine des Raspberry Pis, l'on pourrait ajouter des noeuds, autrement dire ajouter des Raspberry au cluster. Cela permettrait d'ajouter un second noeud maître, qui pourrait prendre le relai du noeud maître actuel en cas de défaillance de celui-ci. Néanmoins, le même problème de stockage utilisé par la simple installation des logiciels persisterait. Il serait aussi possible de monter en gamme, en utilisant des Raspberry Pis plus performants. Cependant, pour ces deux solutions, la production de Raspberry Pis a été stoppée : ainsi, bien qu'il soit encore possible de s'en procurer, ils coûtent de plus en plus cher.

Alors, il peut être intéressant de se pencher sur des alternatives aux Raspberry Pis. On pourrait pour instance se tourner vers un format de mini-ordinateur, de petite taille, mais avec une meilleure capacité. En combinant les cinq Raspberry, on se rend d'ailleurs compte que le cluster consomme finalement presque autant qu'un Mac-Mini, à titre de simple exemple. Avec la flambée des prix des Raspberry Pis, le prix peut facilement être équivalent, et on augmenterait ainsi la capacité disponible. D'une manière plus artisanale, on pourrait aussi fabriquer un cluster à partir de cartes mères d'ordinateurs qui ne sont plus utilisés, mais en bon état. Cela nécessiterait en revanche beaucoup d'investissement et de compréhension de l'architecture et du système.

Par ailleurs, la gateway MultiConnect Conduit de Multitech utilisée est une gateway indoor, qui est pourtant utilisée ici pour un réseau de capteurs en extérieur. Bien que la portée en théorie soit d'une quinzaine de kilomètres, c'est loin d'être le cas en pratique. Il paraîtrait donc logique et utile d'utiliser une gateway adaptée à l'extérieur, comme le fait l'OSUR. De plus, en restant sur le kit MultiConnect, celui-ci contient en plus de la gateway un dispositif appelé MultiConnect mDot Box, illustré en *figure 13*. Ce dispositif contient plusieurs capteurs : un accéléromètre, un altimètre, ainsi que des capteurs de pression, de température et de lumière. Or, bien que cela corresponde à certains des capteurs installés dans la Croix Verte, nous n'utilisons pas du tout la mDot Box. Il serait donc intéressant, si la gateway ne peut être remplacée, d'utiliser ce boîtier fait pour fonctionner avec la gateway.



FIGURE 13 – Boîtier MultiConnect mDot Box

### 6.3 Améliorations logicielles

Du point de vue logiciel, des améliorations peuvent être envisagées tant sur le cluster que sur le VPS.

Le défaut principal de l'architecture logicielle du **cluster** est qu'elle n'est pas assez robuste à l'ajout de nouveaux capteurs. Afin de remédier à ce problème, deux solutions peuvent être envisagées. La première serait de suivre l'architecture proposée dans le rapport de conception. La seconde solution qui viendrait suppléer la première serait de définir une interface claire pour les capteurs. En effet, une fois décodées, les données provenant des capteurs ont des formats différents (objet JSON, image codée en base64...). Cela implique que les fonctions qui utilisent les données des capteurs s'adaptent à chaque format, entraînant alors une complexité problématique. Une interface normalisée à tous les capteurs permettrait d'éliminer cette complexité et de faciliter l'intégration de nouveaux capteurs pour la suite.

En ce qui concerne le **VPS**, un meilleur découpage de l'application tournant dans le conteneur *App* serait aussi souhaitable. Enfin, l'application bénéficierait ici encore d'une définition claire d'interface pour les capteurs.

## 7 Conclusion

Mycélium est un projet pluridisciplinaire, qui s'est révélé très enrichissant pour chacun des membres de l'équipe.

Sur le plan technique, il nous a réellement permis de monter en compétences sur beaucoup de technologies. Malgré les difficultés rencontrées et le temps passé à appréhender la large pile de technologies, ce projet nous a permis de découvrir de nombreux outils variés, pour certains innovateurs, et pour d'autres très utilisés dans beaucoup de projets informatiques.

Sur le plan personnel, nous avons pu évoluer avec des intervenants variés, dont certains n'étaient pas informaticiens. De plus, nous avons dû s'adapter à une équipe qui s'est réduit en cours d'année, et avons eu l'occasion de travailler avec des anciens élèves, participant donc à un véritable échange très instructif.

Mais la particularité de Mycélium repose sur sa dimension environnementale. Nous avons pu découvrir les normes météorologiques afin de récolter des données précises et cohérentes, mais aussi comprendre le fonctionnement et interpréter les résultats des capteurs, pour suivre l'impact sur l'environnement de l'être humain au quotidien. Ce projet nous a permis de voir, en tant que futur ingénieur, qu'il est toujours possible de mettre nos connaissances au service de la préservation environnementale, en travaillant de manière responsable, dans un objectif d'impact carbone positif.

## 8 Annexes

### 8.1 Annexe A : Bilan de planification

Au mois de décembre, un rapport de planification a été écrit. Dans celui-ci était définie une planification ainsi qu'une répartition entre les membres de l'équipe encore présents au Semestre 8. Ainsi étaient prévues :

- les tâches liées à la création de nouvelles fonctions OpenFaaS correspondant à de nouveaux scénarios ;
- la tâche liée à l'intégration du nouveau flux vidéo ;
- la tâche liée à l'ajout de nouveaux capteurs de l'OSUR ;
- la tâche liée au développement du projet dans le Cloud grâce au VPS.

Dans l'ensemble, les tâches prévues ci-dessus ont été réalisées. Néanmoins, l'ajout des capteurs n'a pas pu se faire en cause notamment les raisons évoquées dans la section 3.1.2. Cependant, la réalisation de ces tâches a été plus longue que prévue avec un certain nombre de difficultés rencontrées qui ont été énumérées dans le rapport. S'ajoute à cela la décision d'avancer la fin du projet de deux semaines par rapport à la planification. Ainsi, vers la fin du projet, une charge conséquente de travail a été consentie par l'ensemble des personnes encore présentes en deuxième moitié d'année afin de réaliser les tâches planifiées. Cela comportait notamment l'écriture de ce rapport et la présentation finale du projet ainsi que les démonstrations à préparer lors de la dernière semaine de projet. Le déploiement du scénario Normales Saisonniers a par exemple été réalisé dans les deux dernières semaines du projet et le scénario Orage est toujours en cours de déploiement.

## 8.2 Annexe B : Manuel de l'utilisateur

### 8.2.1 Utilisation de l'environnement de test Garden

Les installations requises avant de créer l'environnement de test :

- Cloner le projet `./Croix-Verte/Croix-Verte` disponible sur le GitLab Mycélium 1.0, puis ouvrir un terminal dans le répertoire courant du projet : `cd ./Croix-Verte`;
- Se référer au *Readme* de Croix-Verte pour l'installation de **Kubernetes** et **Minikube**.

Une fois ces installations faites, lancer les commandes suivantes :

- `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

### 8.2.2 Création d'une fonction OpenFaaS sur sa machine personnelle

Les installations requises avant de d'installer d'OpenFaaS sont les suivantes :

- installer **Minikube** et **Kubernetes** : se référer au *Readme* dans le dossier `/Croix-Verte/Croix-Verte` du projet Mycélium 1.0 ;
- installer **Docker** : se référer au *Readme* dans le dossier `/Mycelium` du projet Mycélium 2.0.

Installer alors **OpenFaaS** et créer sa première fonction :

- installer OpenFaaS : se référer au *Readme* dans le dossier `/Mycelium` du projet Mycélium 2.0.

Les commandes utiles pour créer et modifier sa fonction OpenFaaS sont les suivantes :

- `faas-cli template store pull python3-flask` : récupérer les templates de fonctions d'OpenFaaS ;
- `faas-cli login` : se connecter à OpenFaaS-CLI ;
- `faas-cli new` : créer une nouvelle fonction avec un template choisi ;
- `faas-cli build` : permet de construire la fonction sur sa machine locale ;
- `faas-cli push` : permet de pousser la fonction dans son environnement Docker ;
- `faas-cli deploy` : permet de déployer la fonction ;
- `faas-cli invoke` : permet d'invoquer la fonction afin de la tester ;
- `docker login` : se connecter à Docker.

### 8.2.3 Création d'une fonction OpenFaas pour Mycélium

Une fois la fonction développée sur PC personnel, elle peut être déployée sur Mycélium de la façon suivante :

- cloner le dépôt `Croix-Verte/Croix-Verte` ;
- créer un nouveau dossier portant le nom de la fonction ;
- dans ce dossier, mettre le code dans une fonction `handler.py` et le nom des librairies nécessaires dans un fichier `requirements.txt` ;
- toujours dans le même dossier, créer un fichier `garden.yml` comme indiqué dans le *Readme* du dépôt `Croix-Verte/Croix-Verte` ;
- redéployer le projet avec `garden dev` ou `garden deploy`.

### 8.2.4 Utilisastion du VPS

Pour utiliser le VPS suivre les indications suivantes ou se référer au *Readme* de `Croix-Verte/VPS`

- demander le renouvellement du VPS à la DSI ;
- se connecter au VPS via `ssh` (depuis le réseau insaIOT ou bien depuis les machines du département)
- cloner le dépôt `Croix-Verte/VPS` sur le VPS.
- lancer le script python `relai.py`

- sur une autre session *ssh* lancer la commande suivante : *Docker Compose up*

Pour ajouter des fonctions au VPS se référer au *Readme*

#### 8.2.5 Connexion au VPS

Pré-requis à la connexion au VPS :

- demander le renouvellement du VPS à la DSI ;
- se référer au *Readme* dans le dossier /Mycelium du projet Mycélium 2.0.

Communiquer avec le VPS :

- se connecter au VPS (pour se connecter en SSH, il faut être sur une session Windows sur un ordinateur du département, sinon il faut faire la démarche de connexion à distance) ;
- récupérer les certificats du VPS dans le dossier /etc/mosquitto/certs/ et les mettre sur sa machine dans le dossier du même nom ;
- installer mosquitto et mosquitto-clients avec la commande *sudo apt install* ;
- taper les commandes permettant de communiquer entre le VPS et votre machine listées dans le *Readme* de la section correspondant à cette communication ;

Les commandes suivantes sont utiles afin de communiquer avec le VPS :

- *sudo mosquitto -c* : permet de lancer le broker mosquitto sur le VPS ;
- *mosquitto\_sub* : permet de souscrire à un topic sur le VPS ;
- *mosquitto\_pub* : permet de publier depuis sa machine vers le VPS.

## Table des figures

1	La Croix Verte . . . . .	4
2	FoxyFind : une caméra reliée à un Raspberry Pi . . . . .	5
3	Capteur multiparamètre et débitmètre de l'OSUR . . . . .	6
4	Emplacement des capteurs de l'OSUR . . . . .	6
5	Cluster de Mycélium . . . . .	7
6	Gateway MultiConnect Conduit . . . . .	8
7	Notifications Discord FoxyFind, en provenance du VPS et du cluster . . . . .	10
8	Notification Discord pour le scénario "Normales Saisonnières" . . . . .	10
9	Architecture du VPS . . . . .	11
10	Détail du transfert de charge entre le cluster et le VPS . . . . .	12
11	Architecture logicielle de Mycélium 2.0 . . . . .	13
12	Image d'origine de renard et la notification Discord associée . . . . .	15
13	Boîtier MultiConnect mDot Box . . . . .	17

## Références

- [1] “Site osur.” [osur.univ-rennes.fr](http://osur.univ-rennes.fr).
- [2] L. LAHMAR, “TERRA FORMA : un nouveau paradigme pour l’observation des territoires.” [www.insu.cnrs.fr](http://www.insu.cnrs.fr).
- [3] G. CHAUVEAU, P. IACONE, F. DABAT, E. JUVÉ, Y. TIAN, V. M. ELOSO RODRIGUES, and H. ZHANG, “RAPPORT FINAL MYCELIUM.”
- [4] “What is a raspberry pi ?” <https://www.raspberrypi.org>.