RAPPORT DE SPÉCIFICATION FONCTIONNELLE
Mycélium

Guillaume CHAUVEAU, Florian DABAT, Pieyre IACONE,

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

Figure 1 – Mycélium : projet de suivi environnemental

Table des matières

1 Introduction **3**

2 Présentation de l'architecture générale du projet **4**

3 Suivi environnemental de la Croix-Verte **5**
3.1 Fonctionnalités . . . 5
3.2 Précautions . . . 5
3.3 Scénarios . . . 5
3.3.1 Luminosité anormale : Intempéries . . . 5
3.3.2 Luminosité anormale selon l'heure . . . 6
3.3.3 Neige . . . 6
3.3.4 Températures extrêmes . . . 6
3.3.5 Températures anormales selon l'historique de températures . . . 6
3.3.6 Vol et perte de matériel . . . 7
3.3.7 Capteurs endommagés . . . 7

4 Nœud SoLo **7**
4.1 Aperçu du Nœud SoLo . . . 7
4.2 Fonctionnement . . . 8
4.3 Firmware . . . 9
4.4 Possibilités offertes par le nœud . . . 9
4.4.1 Les capteurs . . . 9
4.4.2 Les alarmes . . . 10
4.4.3 Les interruptions . . . 11
4.5 Améliorations envisagées . . . 11
4.5.1 Détection de dysfonctionnements . . . 11
4.5.2 Amélioration du système d'alarme . . . 11
4.5.3 Lien descendant . . . 12

5 Mycélium **12**
5.1 Prototype . . . 13
5.1.1 Gestion du réseau LoRaWAN . . . 13
5.1.2 Déploiement local . . . 14
5.1.3 Utilisation . . . 14
5.2 Orchestration de conteneurs . . . 15
5.3 Infrastructure as Code . . . 16
5.4 Serverless . . . 16
5.5 Framework de test . . . 17
5.6 Intégration et déploiement continus . . . 17
5.7 Sécurité . . . 17
5.7.1 LoRaWAN et le nœud SoLo . . . 18
5.7.2 Cluster . . . 18
5.7.3 GitLab . . . 19

6 Conclusion **19**

1 Introduction

Le projet Mycélium 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.

Pour ce faire, nous disposons de capteurs connectés à un agrégateur, un boîtier disposant d'un micro-contrôleur, et d'un module de communication LoRaWAN, appelé nœud SoLo. Conçu par le laboratoire de physique de Clermont [1], il a la possibilité d'utiliser d'autres capteurs tels qu'un pluviomètre. La collecte et le traitement des données mesurées sont effectués par une infrastructure logicielle polyvalente baptisée Mycélium, déployée sur un *cluster* de Raspberry Pi au département Informatique de l'INSA.

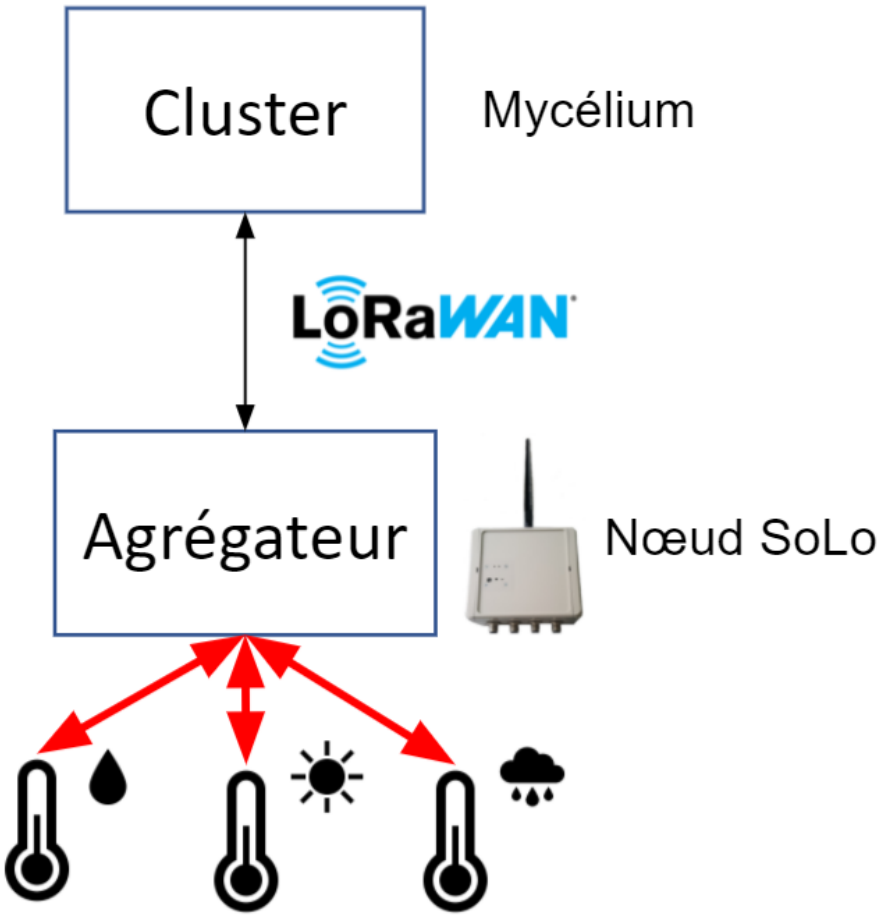

FIGURE 2 – Schéma du principe de fonctionnement, illustrant les communications entre les capteurs, le nœud et le *cluster*

Ce rapport présente les spécifications fonctionnelles du projet. Ce projet étant nouveau et ne partant d'aucun autre existant, nous avons réfléchi au cours des précédentes semaines à son architecture logicielle. Nous allons donc en présenter une première version dans ce rapport, en détaillant les différents modules et leurs interactions.

2 Présentation de l'architecture générale du projet

Le projet peut être découpé en trois parties : une partie concernant le nœud et ses capteurs, une autre se concentrant sur l'infrastructure Mycélium, et enfin une dernière portant sur l'utilisation de cette infrastructure pour le suivi de la zone de la Croix-Verte. Ces différentes parties reposent sur plusieurs sous-composantes du projet, chacune d'entre elles représentant une des fonctions importantes allant de la mesure des données à leur enregistrement.

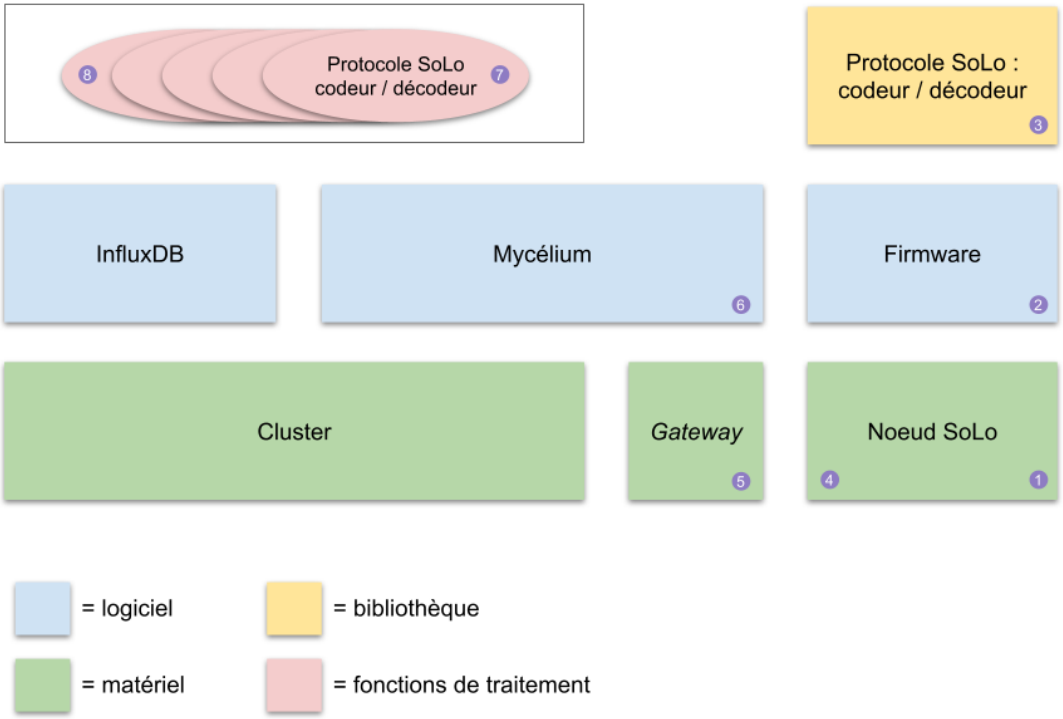

FIGURE 3 – Schéma présentant l'architecture générale du projet Mycélium

— **InfluxDB** : base de données de séries temporelles
— **Cluster** : unité de traitement centrale composée de cinq Raspberry Pi
— **Mycélium** : plateforme de gestion IoT
— **Gateway** : passerelle de protocole LoRaWAN, réseau de communication entre le nœud et Mycélium
— **Protocole SoLo** : format d'échange des données
— **Firmware** : code du nœud SoLo
— **Nœud SoLo** : boîtier composé de capteurs environnementaux : il sera déposé dans un endroit spécifique de la Croix-Verte pour y prendre des mesures

Les données sont transmises en suivant l'ordre indiqué par les numéros sur les blocs de la figure 3. Dans le cas d'une mesure de données classique, les différentes étapes par lesquelles ces données passent sont, dans l'ordre :

1. un capteur prend des mesures
2. le nœud décide d'envoyer des informations
3. les données à envoyer sont encodées avec le protocole SoLo, créé spécialement pour le nœud SoLo.
4. les données encodées sont envoyées avec LoRaWAN
5. la *gateway* LoRaWAN transmet les données encodées à Mycélium
6. Mycélium déclenche les fonctions de traitement

7. une des fonctions décode les données
8. une autre enregistre les informations dans InfluxDB

Dans un tel scénario, l'envoi des données peut avoir lieu plusieurs heures après leur mesure, selon la configuration du nœud, que nous verrons plus en détail par la suite (cf. partie 4).

Dans la suite du rapport, nous allons présenter plus en détail les trois grandes parties de l'architecture du projet Mycélium. Dans un premier temps, nous allons expliquer le fonctionnement de l'application qui servira au suivi de la Croix-Verte. Nous parlerons de sa conception basée sur des scénarios et du stockage des données. Nous verrons ensuite le rôle et le fonctionnement du nœud SoLo dans le projet. Nous détaillerons ainsi la manière dont doivent être paramétrés les capteurs afin de les rendre le plus autonome possible. Enfin nous présenterons le cœur du projet : Mycélium. Nous expliquerons comment à l'aide d'un *cluster* de Raspberry Pi et d'une *gateway* LoRaWAN, nous pouvons mettre en lien l'application de suivi de la Croix-Verte et le nœud SoLo.

3 Suivi environnemental de la Croix-Verte

L'objectif est de concevoir une application qui repose sur des scénarios pouvant déclencher des réactions. Elles sont définies par différents scénarios et correspondent à des événements qui ne sont pas censés se produire, qui nécessitent une analyse supplémentaire, ou qui sont inhabituels par rapport aux données précédemment mesurées et qu'il serait par conséquent intéressant d'observer avec une plus grande précision. Ces alertes pourront être utilisées à titre informatif pour détecter des événements, ou alors pour modifier la configuration du nœud, par exemple la fréquence d'échantillonage des capteurs.

3.1 Fonctionnalités

Le nœud recueille des données de tous ses capteurs à intervalle de temps régulier et prédéfini. Il doit être capable de réagir tout seul à une majorité d'alertes simples : par exemple, selon une certaine valeur, il doit pouvoir augmenter le rythme d'échantillonnage des autres capteurs. Les seuils seront définis à l'aide de mesures prises dans des stations météo proches disponibles en ligne [3], et pourront être modifiés dynamiquement afin de s'adapter aux changements selon les différentes périodes de l'année. Les nouveaux rythmes d'échantillonage pourront être réinitialisés lorsque les évènements sont terminés. Dans certains cas, la levée d'une alerte devra envoyer une notification au *cluster*.

À la fin de la journée, le nœud envoie toutes les données au *cluster* afin de les stocker et de les analyser. Cela est préférable à un envoi plus régulier car cette solution serait plus énergivore.

Il faut garder en tête qu'il est préférable de limiter les interactions entre le nœud et le *cluster*, car communiquer est coûteux en énergie, il est préférable de préserver la batterie du nœud.

On limite également les notifications envoyées au *cluster*. Elles auront principalement un caractère d'urgence.

3.2 Précautions

Certaines précautions doivent être prises lors de l'installation du nœud et des capteurs afin de ne pas fausser les données :

— Le nœud comporte un capteur de luminosité et ne doit donc pas être trop proche d'une source de lumière (sous un lampadaire par exemple)
— Le nœud doit être bien fixé pour que l'accéléromètre ne détecte pas de mouvements à chaque instant, ou il faudrait configurer des seuils de sensibilité prenant ces mouvements en compte.
— Le pluviomètre et le nœud ne doivent pas être sous un arbre pour ne pas fausser les mesures des capteurs de luminosité et du pluviomètre ; de manière générale, ils ne doivent pas être couverts par quoi que ce soit.

3.3 Scénarios

3.3.1 Luminosité anormale : Intempéries

Capteurs utilisés : luminosité et pluviomètre

Contexte : La luminosité est supposée diminuer lorsqu'il pleut. Il se peut que cela ne soit pas le cas : dans ce cas, le nœud réagit

Réaction : Si le pluviomètre indique qu'il pleut, et que la luminosité augmente, on déclenche une augmentation du rythme de prise de mesures de luminosité. On n'a pas besoin d'envoyer de notification au *cluster* car il n'y a pas de risque immédiat. Ce sont des données qui pourront être analysées plus tard, quand le nœud enverra toutes les données.

3.3.2 Luminosité anormale selon l'heure

Capteurs utilisés : luminosité et horloge

Contexte : Des données de luminosité vont être relevées de manière régulière, puis analysées quotidiennement par le *cluster*. Cela va permettre de détecter si le capteur de luminosité a mesuré des données anormalement élevées ou basses selon l'heure (ex : éclipse, objet recouvrant le capteur de luminosité, lampadaire ou lampe de poche...)

Réaction : Il n'y a pas de réactions immédiate suite à la prise de mesure de ces données, car le nœud n'effectue aucun traitement. On n'a donc pas besoin d'envoyer de notifications au *cluster*, cela peut attendre l'envoi des données. Cependant, lors de la réception journalière des données, un traitement sera effectué automatiquement sur les données par le *cluster*. Pour cela, il aura besoin des données trouvées sur internet pour la luminosité en fonction de l'heure de la journée et de la période de l'année.

3.3.3 Neige

Capteurs utilisés : température et pluviomètre

Informations complémentaires : historique des températures et de la pluviométrie

Contexte : La température est négative et des données des données de pluviométrie non nulles sont relevées dès que la température repasse au dessus de 0. On peut alors supposer que la pluie détectée était en réalité de la neige qui a fondu dans le pluviomètre après la hausse de température.

Réaction : Aucune décision n'est prise par le nœud, les données seront envoyées au *cluster* au rythme normal et tout le traitement y sera effectué.

3.3.4 Températures extrêmes

Capteurs utilisés : température

Contexte : Des températures très élevées ou très basses sont mesurées

Réaction : On accélère le rythme d'échantillonage du thermomètre afin de suivre plus précisément l'évolution de la température. On envoie également une notification au *cluster*, afin d'en informer l'utilisateur de l'application dans les plus brefs délais : en effet, si les données sont exactes et que ce n'est pas une erreur du capteur, cela pourrait être dangereux pour le matériel de rester exposé longtemps dans ces conditions.

3.3.5 Températures anormales selon l'historique de températures

Capteurs utilisés : température

Contexte : Des données de température sont prises, et elles vont être analysées lors de la réception des données journalière.

Réaction : Il n'est pas nécessaire d'envoyer un message au *cluster*, cela peut attendre si les températures ne sont pas extrêmes. Cependant, lorsqu'on recevra les données, celles-ci seront analysées par le *cluster* afin de vérifier si les données pour la température ne sont pas anormales : par exemple si on a un écart de plus de 3°C avec la précédente, si on a des températures anormales selon la saison, ou selon l'heure de la journée. Si le *cluster* relève une de ces incohérences, il envoie une notification pour l'utilisateur. Par exemple, on passe de -8°C à 32°C : anormal !

Pour cela, le *cluster* aura besoin d'éléments de comparaison : il utilise l'historique des données de température qu'il a reçu ou provenant d'internet.

3.3.6 Vol et perte de matériel

Capteurs utilisés : accéléromètre + GPS

Contexte : L'accéléromètre doit envoyer des valeurs à 0 m/s^2 (le nœud n'est pas censé se déplacer).

Réaction : Dès que ces valeurs dépassent le seuil de 1m/s (moins de 1G = 9,80665 m/s), on prend les mesures suivantes : augmenter le rythme de mesures de l'accéléromètre, puis allumer le GPS et éteindre les autres capteurs. On envoie donc une notification au *cluster* il faut réagir vite : en effet, un message pour prévenir l'utilisateur permet d'avoir accès aux informations avant que le nœud ne soit trop loin.

Si ça ne bouge plus, on stoppe le GPS et on remet à un rythme normal le rythme des mesures.

3.3.7 Capteurs endommagés

Capteurs utilisés : tous

Contexte : Si des données absurdes sont envoyées et ne correspondent à aucun scénario, on peut supposer que le capteur est endommagé.

Réaction : Notification au *cluster* "Aller vérifier l'état du matériel"

Dans le cas où plusieurs nœuds seraient déployés, le *cluster* pourrait utiliser les informations envoyées par tous les nœuds afin de voir si les données définies comme absurdes sont dues à un problème de capteur (un seul nœud envoie des données aberrantes), ou s'il s'agit d'un évènement non pris en compte. Dans le cas contraire, le *cluster* pourrait détecter ces valeurs aberrantes en comparant les données reçues avec des données d'autres stations proches disponibles sur internet.

4 Nœud SoLo

Pour être capable de prendre en compte les scénarios expliqués précédemment, il nous faut d'abord réaliser un état des lieux du nœud SoLo pour comprendre les possibilités qu'il offre et ses limites. Pour ce faire, nous commencerons par le présenter brièvement, nous verrons ensuite son fonctionnement, son *firmware*, ses fonctionnalités puis nous terminerons par les améliorations envisagées.

4.1 Aperçu du Nœud SoLo

Le nœud SoLo (figure 4) est un dispositif équipé de capteurs conçu pour réaliser des mesures de l'environnement et transmettre des données. Il est capable d'adapter son comportement à certains évènements (expliqués dans la partie 4.4.2), dispose d'une autonomie conséquente (idéal pour des déploiements longs) et d'une grande couverture réseau.

FIGURE 4 – Photo du nœud SoLo

4.2 Fonctionnement

Après une rapide présentation du dispositif, nous allons voir son fonctionnement plus en détail. La figure 5 présente son schéma fonctionnel dont nous expliciterons les différentes parties.

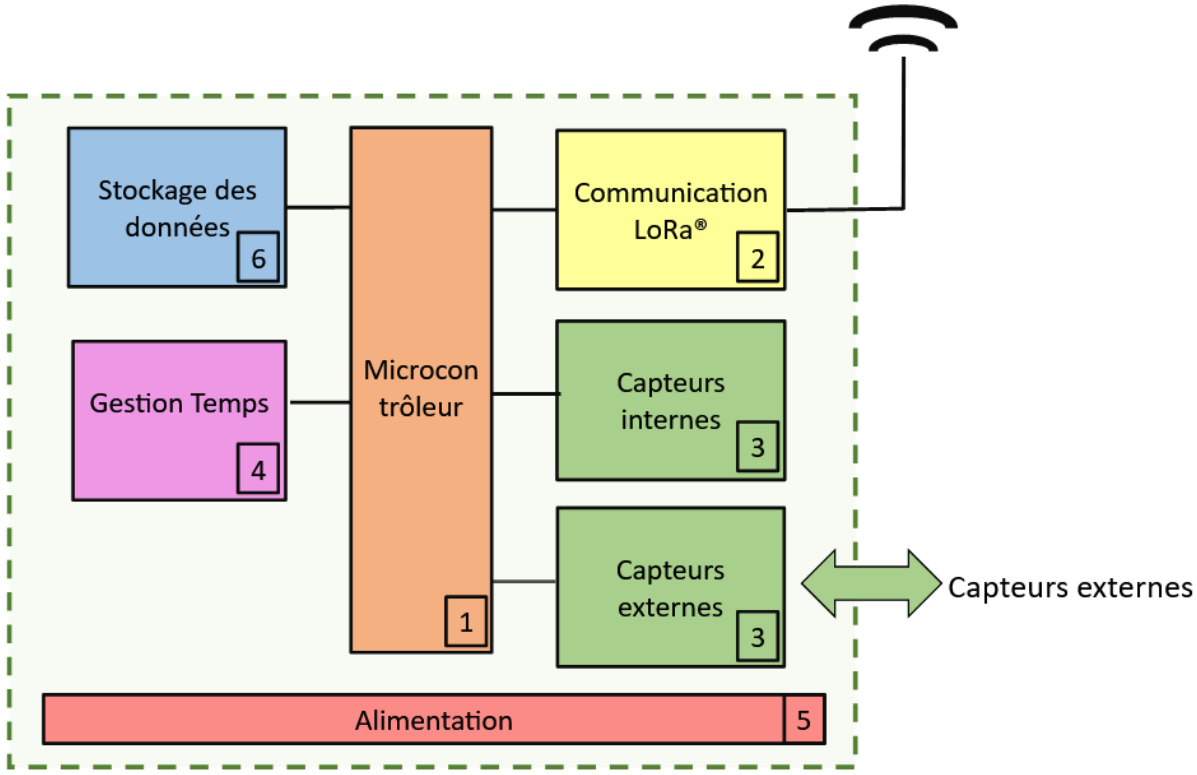

FIGURE 5 – Schéma fonctionnel du nœud SoLo

1. Le microcontrôleur rassemble les éléments essentiels d'un système embarqué : processeur, mémoires et périphériques d'entrée-sortie. Il s'agit ici d'un microcontrôleur STM32. L'utilisation d'un microcontrôleur permet de gagner en place (par rapport aux systèmes utilisant des microprocesseurs) et de diminuer la consommation électrique.
2. Les communications réseau sont assurées par un émetteur-récepteur utilisant la technologie LoRa et le protocole de communication LoRaWAN. Ce protocole est conçu pour envoyer une petite quantité de données sur une longue distance (12 kilomètres dans un cas optimal) le tout en consommant peu d'énergie. Le protocole étant assez limitant sur la taille des données, il est nécessaire de compresser les messages à envoyer, d'où l'utilisation du protocole SoLo.
3. On trouve deux types de capteurs dans le nœud : des capteurs internes et des capteurs externes. Le nœud est déjà équipé d'un capteur de température et d'humidité, d'un capteur de luminosité, d'un accéléromètre et d'un capteur de pression. Comme capteur externe, un pluviomètre nous a été fourni. Leur but est d'effectuer des mesures périodiquement, mesures qui seront ensuite envoyées selon une période d'émission définie. Il est aussi possible de modifier ces périodes selon certains événements comme détaillé en partie 4.4.2.
4. La gestion du temps se fait à l'aide d'un GPS et d'une horloge Real-time clock, alias RTC, (présente sur le microcontrôleur). Le GPS va servir à géolocaliser le nœud et à récupérer l'heure exacte pour mettre à jour l'horloge RTC.
5. Le nœud est alimenté par une batterie de 8800mAh. Son autonomie va dépendre de la fréquence d'activation du GPS, la fréquence d'envoi des données, la fréquence de mesure des capteurs et l'activation (ou non) des capteurs internes. Le nœud consomme peu d'énergie, c'est pour cette raison qu'il peut rester plusieurs mois sur le terrain sans rechargement. Pour économiser de l'énergie, lorsque qu'aucune mesure ne doit être prise, il est dans un état de sommeil profond.

6. Le stockage se fait à l'aide d'une carte microSD de 8 Go. Ses objectifs sont de stocker les données récoltées, les *logs*, le fichier de configuration ainsi que de conserver les mesures en cas de défaillance réseau pour un envoi ultérieur. La configuration se fait à l'aide d'un fichier JSON. Les sections intéressantes du fichier sont décrites dans la partie 4.4.

Maintenant que nous avons vu le fonctionnement du nœud, nous pouvons nous pencher sur les différentes versions de son *firmware* et leurs particularités.

4.3 Firmware

Il existe deux versions principales du *firmware* : la version 1.4 (la version de notre nœud) et la version 2.1. La version 1.4 est la dernière version stable disponible. Dans celle-ci, lorsque le nœud ne prend pas de mesure, il est dans un sommeil profond. Il ne se réveillera qu'à partir du moment où il y a des mesures à réaliser ou des données à envoyer. Le nœud peut aussi être réveillé par une interruption (partie 4.4.3).

Quant à la version 2.1, elle vient complètement changer le comportement du nœud, qui peut maintenant être réveillé directement par un signal provenant d'un des capteurs. Cette version est cependant encore instable : elle a été développée spécialement pour un capteur de radon (capteur mesurant la radioactivité) et n'a pas été testée pour les autres capteurs.

4.4 Possibilités offertes par le nœud

Le fonctionnement du nœud est configurable via un fichier. Ceci nous permet de l'adapter à nos besoins, aux conditions dans lesquelles nous voulons le déployer. Pour chaque fonctionnalité ci-dessous, nous détaillerons aussi les possibilités offertes par le fichier de configuration.

4.4.1 Les capteurs

La configuration des capteurs se fait avec le champ *sensors* auquel on associe une liste de capteurs. Chaque élément de la liste correspond à la configuration d'un capteur. Le champ *name* va servir à identifier le capteur et *type* va indiquer au nœud le type du capteur (selon une liste de type prédéfinie). Parmi les champs qui nous seront très utiles pour le projet, on trouve *periodSec*, *periodMn*, *periodHr* et *periodDay* qui permettent de choisir la période de mesure du capteur.

```
"sensors": [{
    "name": "Press",
    "type": "LPS25",
    "periodMn": 10,
    "interruptChannel":"LPS25_INT",
    "sendOnInterrupt": true,
    "alarm": {
        "sendOnAlarmSet": true,
        "lowThresholdHPa":  900,
        "highThresholdHPa": 1100
    }
}, {
    "name": "Lumi",
    "type": "OPT3001",
    "periodMn": 5,
    "interruptChannels": ["OPT3001_INT", "INT1_INT"],
    "sendOnInterrupt": false
}, {
    "name": "Pluvio1",
    "type": "RainGaugeContact",
    "periodHr": 1,
    "tickInterrupt":  "INT2_INT",
    "tickDebounceMs": 500,
    "rainMMPerTick":  0.2,
    "interruptChannel": "OPTO1_INT",
    "sendOnInterrupt":  true,
    "alarm": {
        "sendOnAlarmSet":     true,
        "sendOnAlarmCleared": true,
        "periodSec":          60,
        "thresholdSetMMPerMinute":  10.0,
        "thresholdClearMMPerMinute": 7.0
    }
}],
```

FIGURE 6 – Exemple de configuration des capteurs

Comme montré sur la figure 6, le capteur de pression (qui correspond au type *LPS25*) est programmé pour prendre des mesures toutes les 10 minutes, le capteur de luminosité (type *OPT3001*) quant à lui prendra des mesures toutes les 5 minutes, etc.

Enfin, d'autres concepts nous seront utiles pour être capable de répondre aux scénarios définis, notamment les alarmes et les interruptions, qui sont expliqués dans les parties suivantes.

4.4.2 Les alarmes

Les capteurs sont capables de produire des alarmes. En cas d'alarme, il est possible de changer le comportement du capteur concerné, et même celui du nœud. Une alarme peut forcer une mesure et une émission, mais aussi augmenter les fréquences de mesures et d'émissions. Une alarme va généralement se déclencher lorsque la dernière mesure effectuée va dépasser un seuil défini par l'utilisateur et restera en mode alarme jusqu'à ce qu'une mesure repasse sous ce seuil. De fait, les champs pour définir les seuils d'alarme sont spécifiques à chaque capteur (et donc les champs associés dans le fichier de configuration aussi).

La configuration de ces alarmes se fait à l'aide d'une section nommée *alarm* comme montré dans la figure 7.

```
"alarm": {
    "sendOnAlarmSet":      true,
    "sendOnAlarmCleared": true,
    "periodSec":           60,
    "thresholdSetMMPerMinute":  10.0,
    "thresholdClearMMPerMinute": 7.0
}
,
```

FIGURE 7 – Exemple de configuration d'une alarme

L'exemple choisi présente la configuration des alarmes du pluviomètre (identifié par le nom *Pluvio1*). Si nous suivons le fichier :

"sendOnAlarmSet" : true

Le nœud forcera l'émission des données récoltées le plus récemment lorsque le capteur passera en mode alarme.

"sendOnAlarmCleared" : true

Le nœud forcera aussi l'émission des dernières données mesurées lorsque le capteur sortira du mode alarme.

"periodSec" : 60

Lorsque ce capteur est en mode d'alarme, il effectuera alors des mesures toutes les minutes contre toutes les heures habituellement (figure 6).

"thresholdSetMMPerMinute" : 10.0

L'alarme sera déclenchée si la quantité de pluie (autrement dit la «vitesse des précipitations») par minute dépasse les 10 millimètres.

"thresholdClearMMPerMinute" : 7.0

Définir un seuil de déclenchement implique obligatoirement de définir un seuil qui arrêtera le mode alarme. Ici, l'alarme sera arrêtée si la quantité de pluie par minute diminue à 7 millimètres.

Sur la figure 7, les paramètres sont propres au pluviomètre. On pourrait citer le capteur de pression où l'on doit définir un seuil de pression, le capteur d'humidité et de température avec ses seuils d'humidité et de température, etc. Cependant, ces paramètres ne sont pas forcément des seuils : l'accéléromètre peut passer en mode alarme à la détection d'un simple mouvement (indiqué par le champ *motionDetection* ayant un booléen comme valeur).

En plus de modifier le comportement d'un capteur, les alarmes permettent aussi de modifier celui du nœud et de l'obliger à augmenter la fréquence d'émission des données.

Dans le fichier de configuration, il faut modifier les paramètres *periodWhenAlarmSec*, *periodWhenAlarmMn*, *periodWhenAlarmHr* ou *periodWhenAlarmDay* qui se situent dans la partie dédiée à la configuration réseau. Ces paramètres ont comme valeur par défaut 0 qui indique qu'il n'y a pas de période d'envoi spécifique lorsqu'un capteur est en alarme.

Pour revenir à nos scénarios (partie 3.3), on se rend compte qu'il serait possible de faire réagir le nœud aux évènements définis de manière automatique. Notre travail consisterait à définir les seuils d'alarme puis de modifier les fréquences de mesures et d'émissions comme convenu, de trouver les paramètres adaptés à la situation.

Cependant, le système d'alarme possède une limite : il est seulement possible d'augmenter la fréquence de mesure du capteur en alarme et d'augmenter la fréquence d'envoi du nœud. Nous ne pouvons pas faire réagir indépendamment d'autres capteurs à une alarme. Une solution à ce problème consisterait à utiliser des interruptions.

4.4.3 Les interruptions

Une interruption correspond à la détection d'un événement. C'est un concept similaire à celui des alarmes, seulement, ici, elle est gérée au niveau matériel. Il existe une liste d'interruptions prédéfinies et utilisables, on en trouve notamment qui sont levées lorsque qu'un capteur passe en mode alarme. Ceci va nous servir pour déclencher des mesures de capteurs et une émission de données au lancement d'une interruption. Sur la figure 8, un exemple de configuration du capteur de pression avec la prise en compte des interruptions.

```
"name": "Press",
"type": "LPS25",
"periodMn": 10,
"interruptChannel":"LPS25_INT",
"sendOnInterrupt": true,
```

FIGURE 8 – Exemple de configuration du capteur de pression

On peut notamment voir le paramètre *interruptChannel*, qui va servir à renseigner l'interruption qui va déclencher une mesure. Si l'on veut définir plusieurs interruptions capables de provoquer une mesure du capteur, on utilise *interruptChannels* qui prend une liste d'interruptions. Le paramètre *sendOnInterrupt* sert à indiquer si l'on veut envoyer les données après une mesure causée par une interruption. Cette fonctionnalité nous sera très utile : lorsqu'un capteur passera en mode alarme, ceci déclenchera une interruption qui pourra forcer la mesure d'autres capteurs.

4.5 Améliorations envisagées

En réalisant un inventaire des possibilités offertes par le nœud, nous avons aussi pu voir ses limites. Cette partie vient présenter les modifications du nœud que nous effectuerons pour être capable de répondre aux scénarios définis.

4.5.1 Détection de dysfonctionnements

Afin de surveiller des pannes éventuelles lors du démarrage du nœud, un envoi de données vers le *cluster* sera demandé pour tous les capteurs. Ceci permettrait de vérifier la valeur du GPS, de l'heure ainsi que de compter le nombre de capteurs démarrés pour détecter le dysfonctionnement d'un ou plusieurs capteurs sans attendre une période d'envoi de plus.

4.5.2 Amélioration du système d'alarme

La deuxième amélioration consiste en la création d'un système d'alarme plus flexible. Pour ce nouveau système, il faudra définir une condition (par exemple, le dépassement des seuils de plusieurs capteurs) et une réaction engendrée (modification de la fréquence de mesure, par exemple). Cette modification du comportement serait effective tant que la condition d'entrée est vraie. Le champ ajouté dans le fichier de configuration pourrait ressembler à la figure 9.

```
"alarms": [{
        "conditions": {
            "Lumi.HighLux": 30.0},
            " Pluvio1.thresholdHigh": 10.0},
        "reactions":[{
        "name":          "Lumi",
        "sendOnAlarmSet":    true
        },{
        "name":          "Pluvio1",
        "sendOnAlarmSet":    true
        }]
    },
    {
        "conditions":{
            "TempHumi.temperatureLowDegC": 0.0},
            "Pluvio1.thresholdHigh": 7.0},
        "reactions":[{
        "name":          "Pluvio1",
        "sendOnAlarmSet":    true
        }]
    }
    }]
```

FIGURE 9 – Proposition de nouveaux champs de configuration à ajouter

La première alarme correspondrait au scénario d'intempéries, avec une vitesse de précipitations élevée mais une basse luminosité. La deuxième correspondrait à celui de la neige, avec une température inférieure à 0 et une vitesse de précipitations relativement élevée.

4.5.3 Lien descendant

L'autre amélioration du nœud que nous ferons est de rendre possible une communication allant du serveur vers le nœud, d'où le terme lien descendant. Ceci nous permettrait de changer le comportement à distance, par exemple en prévision d'un événement (qui ne peut pas forcément se traduire par des alarmes) ou alors si le moment nous semble opportun à une augmentation des fréquences de mesures. Cette fonctionnalité servirait à modifier la configuration des alarmes à distance, puis à mettre à jour le fichier de configuration.

Cette fonctionnalité n'est implémentée dans aucune version du *firmware*, il faut donc tout d'abord choisir la version la plus appropriée pour la développer. Elle implique aussi de se pencher sur le protocole LoRaWAN et sur une manière de réveiller le nœud pour qu'il puisse recevoir les données.

5 Mycélium

Afin de recevoir et de traiter les informations transmises par les nœuds, il est nécessaire de créer une infrastructure logicielle adéquate. Elle doit répondre à différents critères :

- Haute disponibilité : la surveillance de la Croix-Verte doit se faire 24h sur 24. Le système ne peut être interrompu, au risque de rater des informations cruciales.
- Scalabilité : Le système sera déployé sur un *cluster* de Raspberry Pis, et non sur une seule machine. La charge de travail doit pouvoir être distribuée.
- Sécurité : Le système sera exposé non seulement sur internet, mais le matériel sur le terrain ainsi que la communication radio représentent aussi une surface d'attaque.

Les composants principaux de Mycélium sont le fournisseur de réseau LoRaWAN, qui est responsable de la gestion de notre réseau LoRaWAN privé, ainsi qu'un pont vers la partie métier, qui traite les données réceptionnées.

Dans cette partie, nous étudierons rapidement le fonctionnement d'une première version de notre infrastructure (Mycélium), puis les évolutions que nous allons réaliser pour répondre aux critères de disponibilité et de scalabilité. Nous nous intéresserons ensuite aux questions de sécurité.

5.1 Prototype

Afin de tester le nœud SoLo et la *gateway* LoRaWAN qui nous ont été fourni, nous avons mis en place une première infrastructure simplifiée. Elle nous a permis d'effectuer des mesures avec les capteurs intégrés au nœud (température, humidité, luminosité, pression atmosphérique et accéléromètre), et de les transmettre périodiquement via LoRaWAN, en utilisant la *gateway* connectée directement à un ordinateur personnel. Elle est composée des éléments suivants (figure 10) :

— Fournisseur de réseau LoRaWAN : ChirpStack. . Il est configuré pour interagir avec la *gateway*, authentifier le nœud et permet la transmission des données entre ce dernier et le reste du système.
— Serveur de traitement. Cette application réceptionne les données du nœud transmises par ChirpStack, les décode et les enregistre dans une base de données.
— Base de donnée : InfluxDB. Cette base de données est spécialisée dans les séries temporelles. Elle stocke ici les mesures effectuées par les capteurs internes du nœud, en fonction du temps.

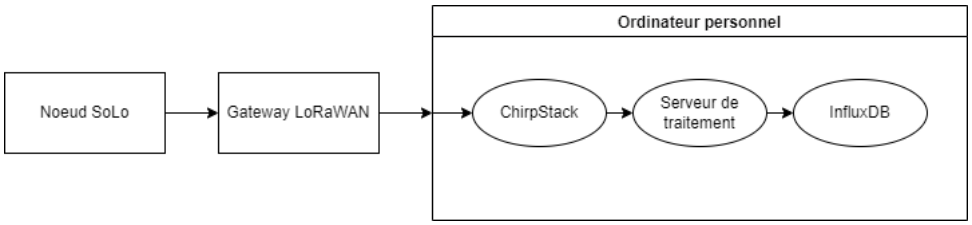

FIGURE 10 – Éléments du prototype

5.1.1 Gestion du réseau LoRaWAN

La gestion d'un réseau LoRaWAN est standardisée. Trois éléments interagissent pour amener les informations reçues par la ou les *gateways* aux applications destinataires (figure 11) :

— le Network Server : déduplique les trames reçues dans le cas où plusieurs *gateways* reçoivent le même envoi, envoie les commandes réseau bas niveau (couche MAC) ;
— le Join Server : gère l'inventaire des appareils du réseau pour authentifier et (dé)chiffrer les trames ;
— l'Application Server : sert d'intermédiaire entre le réseau et les applications, en redirigeant les informations aux bons destinataires.

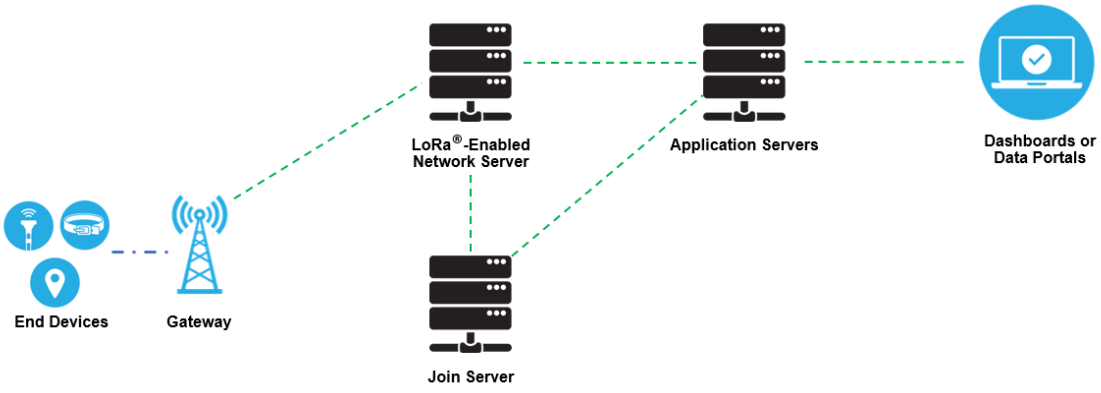

FIGURE 11 – Éléments de gestion d'un réseau LoRaWAN

ChirpStack est une implémentation open-source et gratuite de la spécification LoRaWAN [4]. L'Application Server et le Join Server y sont combinés en un seul logiciel pour simplifier l'utilisation via

l'interface web de configuration. C'est dans cette interface que l'on renseigne les *gateways* et appareils du réseau, ainsi que les applications auxquelles envoyer les données reçues. Le Network Server (NS) et l'Application Server (AS), qui a donc aussi le rôle de Join Server, sont distribués avec des images Docker, et nécessitent des bases de données PostgreSQL et Redis pour fonctionner.

5.1.2 Déploiement local

Pour ce prototype, nous déployons l'infrastructure sur un ordinateur personnel avec Docker : quatre conteneurs pour ChirpStack (NS, AS, PostgreSQL et Redis), un pour notre base de données InfluxDB et un pour un serveur HTTP recevant les informations transmises (figure 12). Un fichier Docker Compose a été créé pour faciliter le déploiement. ChirpStack est configuré manuellement via l'interface web, pour indiquer les identifiants de la *gateway* et du nœud, ainsi que l'adresse du serveur HTTP.

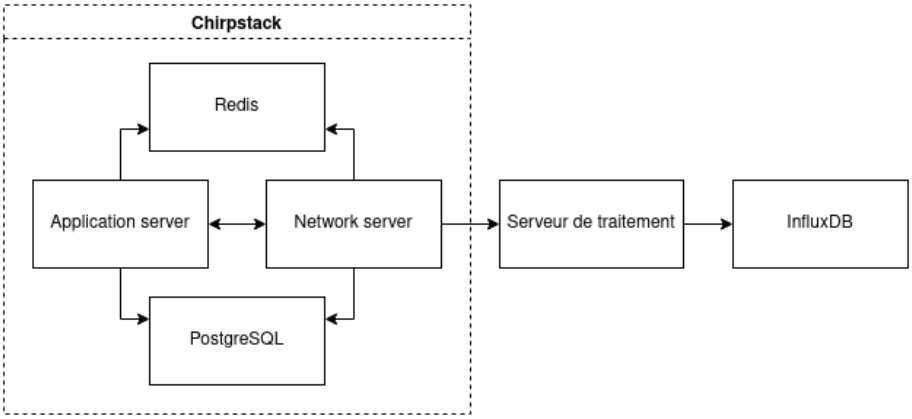

FIGURE 12 – Conteneurs du prototype

5.1.3 Utilisation

ChirpStack envoie une requête POST dès que des données sont reçues. Le serveur peut alors décoder les informations et les enregistrer dans la base de données. L'utilisateur peut se connecter à l'interface web d'InfluxDB pour y générer des graphiques (figure 13).

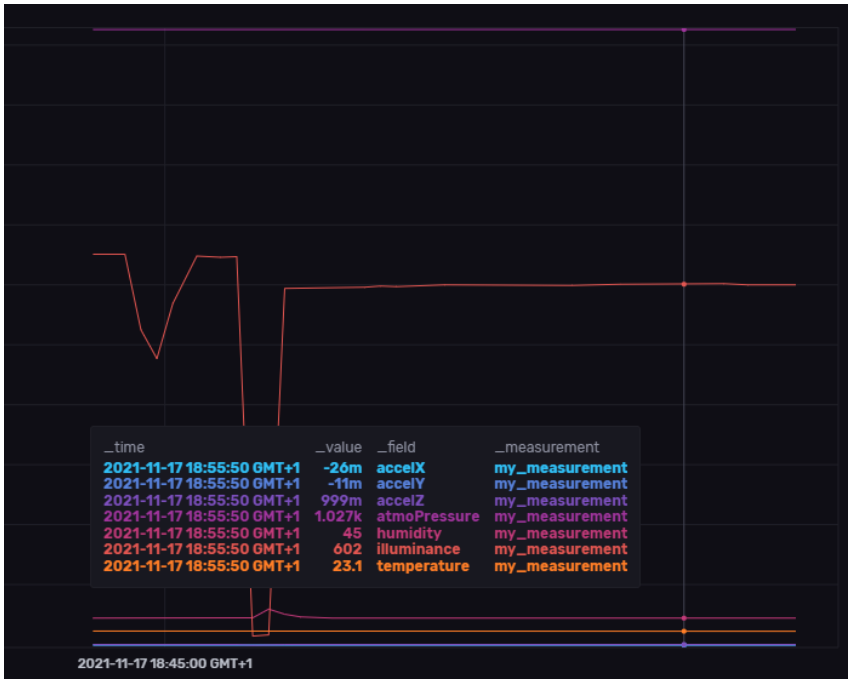

FIGURE 13 – Affichage des données par InfluxDB

Dans ce prototype, Mycélium consiste uniquement en un fournisseur de réseau LoRaWAN. Il présente les problèmes suivants :

- Il nécessite des interventions manuelles pour son déploiement (configuration de ChirpStack), des erreurs humaines peuvent se produire et ainsi provoquer des pannes du système.
- Il est difficile de distribuer des conteneurs sur plusieurs ordinateurs en utilisant seulement Docker.
- L'application prend la forme d'un serveur monolithique conçu spécifiquement pour Mycélium. Les développeurs en charge de ce projet doivent donc connaître son fonctionnement en profondeur.

Afin de pallier ces problèmes, de nouveaux éléments vont être ajoutés à Mycélium.

5.2 Orchestration de conteneurs

Afin de distribuer la charge de travail, c'est-à-dire les conteneurs nécessaires au fonctionnement de notre système, nous allons utiliser Kubernetes[6]. Ce logiciel permet d'orchestrer le déploiement de conteneurs sur plusieurs ordinateurs, appelés nœuds de travail (*worker nodes*). Ces nœuds, dans notre cas des Raspberry Pi, communiquent avec ce qui est appelé le plan de contrôle (*control plane*), en charge de la coordination. L'ensemble de ces éléments forme un *cluster* Kubernetes. Le plan de contrôle reçoit les instructions pour déployer les différentes applications via des fichiers de configuration, et distribue automatiquement les conteneurs sur les nœuds de travail. Il permet aussi au système de rester opérationnel en redéployant en cas de panne. Les différents logiciels nécessaires au fonctionnement de Mycélium auront leur configuration Kubernetes associée pour leur déploiement.

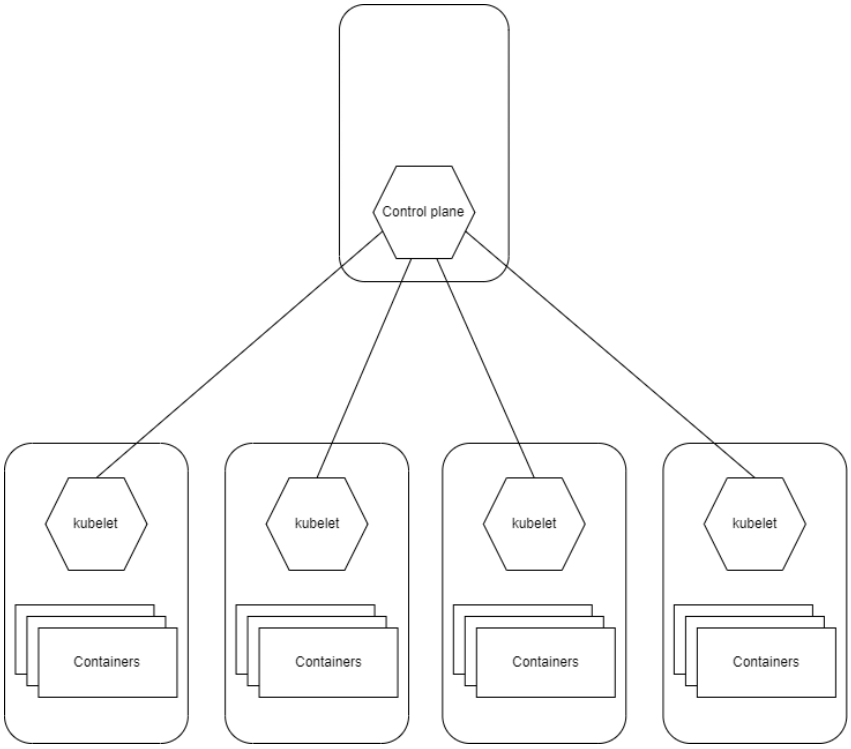

FIGURE 14 – Illustration du *cluster* Kubernetes

La figure 14 présente de façon simplifiée les éléments du *cluster* Kubernetes. On y voit cinq rectangles arrondis (les Raspberry). L'un d'entre eux contient le plan de contrôle, il s'agit du nœud maître du *cluster*. Les autres contiennent *kubelet*, un composant de Kubernetes qui reçoit les instructions du plan de contrôle pour déployer des conteneurs : ce sont les nœuds de travail.

5.3 Infrastructure as Code

Avec Kubernetes, la mise en œuvre et la maintenance des applications se font par des fichiers YAML décrivant la configuration d'état souhaitée. Ces fichiers sont appliqués par une ligne de commande, Kubernetes gère ensuite tout le processus. Les fichiers de configuration de Kubernetes sont versionnés avec Git, leur traçabilité et le retour en arrière sont assurés. C'est ce qu'on appelle l'Infrastructure as Code (« infrastructure en tant que code ») : il n'est plus nécessaire d'installer manuellement les logiciels, ce qui réduit le risque d'erreur humaine. Cependant, certains logiciels ne peuvent pas être configurés via Kubernetes une fois installés. ChirpStack requiert la liste des *gateways* et appareils LoRaWAN, ainsi que d'autres éléments clés, mais ne propose pas de moyen de les définir via un fichier. Afin de réduire le risque d'erreur et d'automatiser le déploiement, un outil capable d'interagir avec l'API de ChirpStack pour configurer le logiciel à partir d'un fichier a été créé.

5.4 Serverless

L'architecture monolithique actuelle du prototype apporte une série d'inconvénients. L'ensemble du système est développé comme une seule application, en charge du décodage, des traitements et de la sauvegarde des données reçues. Ainsi, le logiciel devient de plus en plus complexe au fur et à mesure de son développement, les mises à jour nécessitent un redémarrage de tout le système, et les développeurs en charge ont besoin de connaître tout le système en profondeur, ce qui entrave la répartition des tâches au sein de l'équipe. Pour résoudre ce problème, nous allons séparer les tâches d'application avec une architecture en microservices. Chaque morceau de l'application prend la forme d'une fonction développée séparément, une façon d'implémenter les microservices appelée serverless[7]. Dans le cas de la Croix-Verte, le décodage et l'enregistrement dans InfluxDB sont deux des fonctions du système. La communication entre ces fonctions et leur exécution est abstraite au développeur, qui peut se concentrer entièrement

sur le code métier. L'exécution d'une fonction est provoquée par l'arrivée de nouvelles données dans ChirpStack, et cette fonction peut elle-même en déclencher d'autres. Les fonctions sont exécutées dans un conteneur géré par Kubernetes, qui va automatiquement distribuer la charge sur le *cluster*. Plusieurs instances d'une même fonction peuvent s'exécuter en parallèle en cas de forte demande. Un logiciel supplémentaire, OpenFaaS [8], va être chargé de la gestion de ces fonctions.

5.5 Framework de test

Mycélium est un système complexe, de nombreux éléments interagissent entre eux pour faire circuler et traiter les informations. Tester ces éléments individuellement est important, mais tester l'ensemble du système l'est d'autant plus. Pour effectuer ces tests, nous allons créer un ensemble d'outils permettant de simuler les données envoyées par un nœud SoLo, de vérifier le comportement de chaque élément de la chaîne de traitement, jusqu'au données renvoyées au nœud simulé. Ce framework permettra de spécifier les données (mesures, notifications...) que le nœud virtuel devra envoyer. En utilisant l'API de ChirpStack, il fera entrer les données encodées dans le réseau LoRaWAN comme si elles venaient d'être reçues par la *gateway*. Ce principe est déjà utilisé par le projet chirpstack-simulator sur lequel nous nous basons (le projet original est trop limité pour notre usage) [9]. Les tests de la chaîne de traitement seront effectués en interceptant l'exécution des fonctions *serverless*. Les données sortant des différentes fonctions peuvent être simplement comparées. Le nœud virtuel peut recevoir des données, simulant le flux descendant. Ici aussi, les données reçues peuvent être comparées à ce qui est attendu. La figure 15 illustre le concept général.

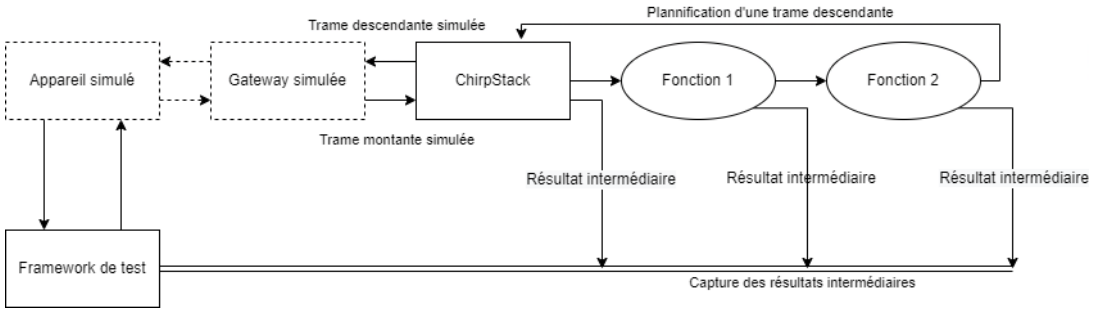

FIGURE 15 – Déroulement d'un test d'intégration

Ces tests d'intégration permettront de décrire les scénarios établis précédemment, et de vérifier que le code présent sur le *cluster* ainsi que l'infrastructure dont il dépend fonctionnent correctement.

5.6 Intégration et déploiement continus

L'ensemble du projet est entièrement déployable et configurable sans intervention manuelle grâce aux mécanismes que nous avons présentés précédemment. Cette particularité nous donne la possibilité de déployer automatiquement, en utilisant des pipelines GitLab CI [10]. Nous collaborons avec la Direction des Systèmes d'Information de l'INSA pour permettre à notre *cluster* d'être accessible au GitLab de l'INSA. Une fois l'accès opérationnel, GitLab sera en mesure d'exécuter des instructions sur le *cluster*, nous permettant de tester et déployer chaque nouvelle version de façon continue, sans que nous ayons à intervenir sur le *cluster* directement. Nous serons aussi capables de surveiller le statut du système directement dans GitLab.[11]

5.7 Sécurité

Dans cette partie nous verrons quels sont les points de faiblesse du projet en terme de sécurité, et comment nous comptons les renforcer.

5.7.1 LoRaWAN et le nœud SoLo

LoRaWAN est un protocole sécurisé par des clés AES 128 bits[12], stockées dans le fichier de configuration du nœud SoLo. Lorsqu'un appareil se connecte au réseau LoRaWAN (*join request*), il est d'abord authentifié par le Join Server. Des clés de session temporaires sont créées pour les échanges de données qui auront lieu par la suite (figure 16).

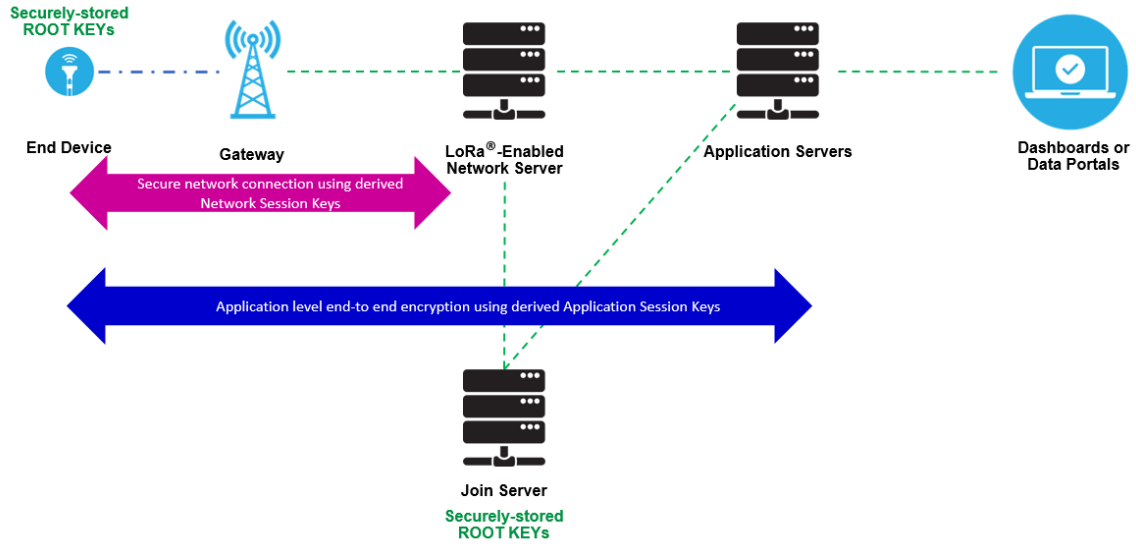

FIGURE 16 – Sécurisation d'une transmission LoRaWAN

Les clés réseau sont lisibles en clair sur le stockage du nœud. Leur sécurité dépend de celle du boîtier physique. Le risque est de recevoir des données incorrectes si quelqu'un reprogramme le nœud ou utilise les clés pour envoyer des données falsifiées. L'impact de cette faille est assez limité.

5.7.2 Cluster

En automatisant le déploiement et la configuration de l'ensemble des applications sur le *cluster*, il est possible de réduire les accès administrateur au minimum. Les Raspberry communiquent entre eux et avec la *gateway* via un réseau local accessible uniquement en y étant connecté physiquement. Les communications avec l'extérieur (via un réseau WiFi) sont limitées à l'API Kubernetes pour contrôler les charges de travail sur le *cluster* et quelques services informatifs uniquement (comme Grafana pour afficher les mesures). Un accès SSH n'est pas requis pour notre système. Le point sensible est donc l'API Kubernetes, qui est sécurisée via un certificat et un token. Puisque nous pratiquons le déploiement continu et que la configuration est entièrement statique, seulement GitLab a besoin de ces informations, aucune personne n'a directement des droits d'administration sur n'importe quel aspect du *cluster*. Il est même envisageable de restreindre l'accès à l'API Kubernetes au GitLab de l'INSA, sans l'exposer à internet.

Concernant le réseau local du *cluster*, les différents logiciels sécurisent leurs communications en TLS mutuel (mTLS) [13] : deux logiciels communiquants s'authentifient l'un-l'autre avec des certificats TLS. Pour intervenir sur la *gateway* ou un Raspberry, une connexion SSH à l'aide d'une clé peut être effectuée. Ces clés doivent être gardées précieusement, mais restent inutilisables sans accès physique au *cluster*.

Les communications réseau entre les différents conteneurs sur le *cluster* Kubenetes sont autorisées individuellement, et des identifiants aléatoires ajoutent une couche de complexité supplémentaire en cas d'intrusion. Comme indiqué précédemment, seulement certains conteneurs auront un port exposé à internet.

5.7.3 GitLab

Le contrôle du *cluster* Kubernetes est entre les mains de l'instance GitLab de l'INSA. Il est de la responsabilité des membres du projet de sécuriser l'accès à leur compte, et de ne pas déclencher le déploiement de code dangereux. La sécurité des identifiants de connexion au *cluster* dépend de celle de GitLab, gérée par la Direction des Systèmes d'Information de l'INSA.

6 Conclusion

Au coeur du projet, Mycélium permet via un réseau LoRaWAN de recevoir les données de capteurs et de déclencher différentes fonctions de traitement. Celles-ci analysent, stockent et réagissent en fonction de ce qui est reçu et de scénarios établis en amont. Les mesures sont effectuées par les nœuds SoLo. Ces nœuds sont autonomes quant à la gestion de leurs capteurs, et envoient leurs mesures via LoRaWAN à Mycélium. Ils sont aussi capables de réagir d'eux-mêmes aux mesures, et leur configuration est modifiable à distance.

Références

[1] Site du laboratoire de physique de Clermont traitant du projet ConnecSenS : http://www.lpc-clermont.in2p3.fr/spip.php?article583

[2] Le Gall, Alain-Hervé ; Longuevergne, Laurent. Avec SMARTOBS, le campus de Beaulieu va devenir un espace expérimental in situ alliant nouvelles technologies et sciences de l'environnement : https://osur.univ-rennes1.fr/news/avec-smartobs-le-campus-de-beaulieu-va-devenir-un-espace-experimental-in-situ.html

[3] https://www.wunderground.com/hourly/fr/cesson-sévigné/LFRN

[4] What is LoRaWAN® Specification https://lora-alliance.org/about-lorawan/

[5] Tankov, Vladislav ; Valchuk, Dmitriy ; Golubev, Yaroslav ; Bryksin, Timofey ; Infrastructure in Code : Towards Developer-Friendly Cloud Applications, 2021

[6] Qu'est-ce-que Kubernetes ? La documentation de Kubernetes qui donne une vue d'ensemble de Kubernetes : https://kubernetes.io/fr/docs/concepts/overview/what-is-kubernetes/

[7] Serverless ou FAAS (Function As A Service), une révolution ? : https://www.theodo.fr/digital-et-strategie/serverless-faas-une-revolution

[8] OpenFaaS, Serverless Functions, Made Simple : https://www.openfaas.com/

[9] Projet chirpstack-simulator : https://github.com/brocaar/chirpstack-simulator

[10] GitLab documentation, CI/CD Pipelines : https://docs.gitlab.com/ee/ci/pipelines/

[11] GitLab documentation, GitLab Kubernetes Agent, un composant actif en *cluster* permettant de connecter les *clusters* Kubernetes à GitLab : https://docs.gitlab.com/ee/user/clusters/agent/index.

[12] AES : Advanced Encryption Standard (norme de chiffrement avancé) : https://fr.wikipedia.org/wiki/Advanced_Encryption_Standard

[13] L'authentification mTLS (Mutual Transport Layer Security) : https://cloud.ibm.com/docs/cis?topic=cis-mtls-features&locale=fr