

Signature

Encadrant ENIT : Mme Meriem KASSAR

Remerciements

Au terme de ce travail, nous adressons nos remerciements aux membres du jury d'avoir accepté d'évaluer notre projet de fin d'année.

Nous tenons à remercier vivement notre encadrante Mme. Meriem Kassar pour son suivi, ses remarques constructives, la qualité de l'encadrement ainsi que les précieux conseils qu'elle nous a prodigués avec bienveillance tout au long de ce travail.

Nous exprimons également toute notre reconnaissance à nos professeurs de l'ENIT, tant pour la formation de qualité offerte, que pour leur implication.

Finalement, nous tenons à exprimer nos sincères remerciements et notre profonde gratitude à toute personne qui, de près ou de loin, nous a apportés son aide lors de la réalisation de ce modeste projet.

Résumé

Avec la croissance rapide des applications IoT, le paradigme classique du Cloud Computing centralisé est confronté à plusieurs défis parmi lesquels une latence élevée, une faible capacité et des pannes de réseau. Pour faire face à ces défis et assurer une meilleure qualité de service, Le Fog Computing rapproche les ressources de calcul et de stockage des périphériques IoT où les données sont générées.

L'émergence de l'Internet des objets et l'utilisation du modèle Fog auront donc un impact majeur sur les normes de l'industrie, la sécurité et l'ensemble de l'écosystème informatique. Cependant, les développeurs sont confrontés à de nombreux obstacles lors du développement, de la création et de la mise à l'échelle des applications IoT. Docker est une technologie de virtualisation en conteneur open source qui peut aider à développer et à faire évoluer facilement les applications IoT.

Mots clés : Fog Computing, Smart Building, Microservices, DevOps

Abstract

With the rapid growth of IoT applications, the classic paradigm of centralized cloud computing is facing a number of challenges including high latency, low capacity, and network outages. To cope with these challenges and ensure a better quality of service, Fog Computing brings computing and storage resources closer to the IoT devices where data is generated.

The emergence of the Internet of Things and the use of the Fog model will therefore have a major impact on industry standards, security and the entire IT ecosystem. However, developers face many obstacles when developing, creating, and scaling IoT applications. Docker is an open source container virtualization technology that can help develop and scale IoT applications easily.

Mots clés : Fog Computing, Smart Building, Microservices, DevOps

Table des matières

Liste des figures	8
Liste des tableaux	10
Liste des acronymes	11
Introduction Générale	12
1 Présentation du projet	14
Introduction	14
1.1 Cadre du projet & Problématique	14
1.2 Objectifs du projet	15
1.3 Démarche méthodologique	16
Conclusion	16
2 État de l'Art	17
Introduction	17
2.1 Internet des Objets	17
2.1.1 Son architecture	17
2.1.2 Ses applications	18
2.1.3 Ses Défis	19

2.2	Fog Computing	19
2.2.1	Son architecture	20
2.2.2	Ses caractéristiques	21
2.2.3	Comparaison entre le Cloud et le Fog Computing	21
2.3	Smart Buildings & Fog Computing	22
2.4	DevOps & IoT	24
2.4.1	Avantages	24
2.4.2	Principales technologies offertes	24
2.5	Conteneurisation	25
2.6	Docker	27
2.7	Architecture de microservices	30
2.8	Raspberry Pi	31
	Conclusion	32
3	Conception du projet	33
	Introduction	33
3.1	Scénario d'application	33
3.2	Diagramme de séquences	34
3.3	Spécifications techniques	35
3.3.1	Docker Compose	35
3.3.2	Langage de programmation Python	36
3.3.3	Langage de programmation C	36
3.3.4	Base de données : MySQL, PHPMyAdmin	36
3.3.5	Serveur APACHE2	37
3.3.6	Framework de développement Web	37

3.3.6.1	Flask	37
3.3.6.2	Dash	37
3.4	Connectivité du matériel	38
3.4.1	Liaison Capteurs - Fog	39
3.4.2	Liaison Base de données - Fog	39
3.4.3	Liaison Terminaux - Fog	39
	Conclusion	39
4	Réalisation du projet	40
	Introduction	40
4.1	Environnement matériel	40
4.1.1	Carte Raspberry Pi 3 B+	40
4.1.2	Carte Arduino UNO	41
4.1.3	Capteurs	42
4.2	Mise en place de la solution	42
4.2.1	Câblage du matériel	42
4.2.2	Installation de Docker et Docker Compose	43
4.2.3	Création des scripts	43
4.2.4	Conteneurisation	44
4.2.5	Création de la base de données avec le fichier docker-compose	45
4.2.6	Visualisation des conteneurs	46
4.2.7	Du Fog vers le Dashboard	47
4.2.8	Du Fog vers le Cloud	48
4.3	Constatations & Perspectives	49
	Conclusion	49

TABLE DES MATIÈRES	8
--------------------	---

Conclusion Générale	49
---------------------	----

Bibliographie	50
---------------	----

Annexes	51
---------	----

A	52
---	----

B	54
---	----

Liste des figures

2.1	Architecture IoT	18
2.2	Modèle du Fog Computing [2]	19
2.3	Architecture du Fog Computing	20
2.4	Smart Building	22
2.5	Smart Building sans Fog Computing	23
2.6	Smart Building avec Fog Computing	23
2.7	Machine virtuelle vs Conteneurs	27
2.8	Architecture Docker	28
2.9	Taxonomie des termes et concepts Docker	30
2.10	Architecture de microservices	31
2.11	Raspberry Pi	32
3.1	Traitement des tâches dans un nœud Fog	34
3.2	Traitement des données dans le Fog	35
3.3	Interconnexion des composants	38
4.1	Carte Raspberry Pi	41
4.2	Carte Arduino Uno	41
4.3	Câblage du matériel	42

4.4	Installation Docker et Docker Compose	43
4.5	Résultat du test du scénario	44
4.6	Listes des conteneurs utilisés	45
4.7	Serveur de Base de données	45
4.8	Dashboard de conteneurs	46
4.9	Performances des conteneurs	46
4.10	Listes des images de conteneurs	47
4.11	Dashboard de visualisation des données	47
4.12	Plateforme Ubidots	48
4.13	Visualisation des Données sur le Cloud	48

Liste des tableaux

2.1	Comparaison entre le Cloud et le Fog Computing	21
2.2	Comparaison entre les conteneurs et les machines virtuelles [5]	26
2.3	Composants Docker [7]	29

Liste des acronymes

IoT Internet of Things

JSON JavaScript Object Notation

SQL Structured Query Language

UART Universal Asynchronous Receiver-Transmitter

Introduction Générale

L'Internet des objets, plus communément appelé IoT (Internet of Things) englobe un nombre illimité de périphériques et objets connectés pouvant atteindre 50 millions d'ici 2020. Une telle expansion de l'IoT et les applications correspondantes exigent la prise en charge de la mobilité, la distribution géographique, la reconnaissance de l'emplacement ainsi que le délai minimum d'analyse et de traitement des données.

Le Cloud Computing toujours considéré comme une technologie révolutionnaire ayant résolu de nombreux problèmes des modèles informatiques traditionnels n'est pas en mesure de répondre à ces exigences.

Un nouveau modèle, le Fog Computing, a été alors conçu afin de couvrir les failles du Cloud et répondre aux exigences des systèmes IoT. Ce dernier étend le concept de cloud computing à la périphérie du réseau le rendant ainsi idéal pour l'IoT et les applications nécessitant des interactions en temps réel.

Les dispositifs IoT ont un cycle de vie allant au-delà du déploiement, chacun étant essentiel au bon fonctionnement d'un écosystème vaste et connecté comprenant de nombreux autres dispositifs. L'utilisation d'une architecture basée sur des microservices conteneurisées permet de réduire la complexité d'intégration et de maintenabilité de ces dispositifs et d'améliorer leur performance.

Notre rapport comporte quatre chapitres. Le premier chapitre présente le contexte général de notre projet. Le deuxième chapitre introduit les différents concepts et notions sur lesquels est basé ce travail. Dans le troisième chapitre, nous présenterons le scénario de notre application ainsi que les différentes technologies utilisées lors de son développement. Enfin, le quatrième chapitre décrit les différentes étapes de réalisation de ce projet.

Chapitre 1

Présentation du projet

Introduction

Ce chapitre comporte une présentation du cadre de notre projet ainsi que nos objectifs et la démarche méthodologique que nous avons pris en considération lors du développement de ce travail.

1.1 Cadre du projet & Problématique

Les applications IoT ont dévoilé plusieurs limitations du Cloud Computing. En effet, le modèle du Cloud Computing ne convient pas à de nombreuses applications IoT qui nécessitent une latence prévisible et une interaction en temps réel entre les dispositifs. Le déploiement d'une nouvelle architecture est ainsi nécessaire pour pouvoir répondre à ces besoins. Le Fog Computing est une approche distribuée qui a été proposée pour relever les défis posés par l'IoT et résoudre les problèmes du modèle Cloud.

La virtualisation permet de garantir l'isolation, l'évolutivité des applications et l'indépendance du matériel, son utilisation pour déployer des applications IoT est ainsi souhaitable. La conteneurisation étant une solution de virtualisation plus légère que la machine virtuelle, elle convient mieux au modèle Fog Computing qui peut être composé de nœuds limités par les ressources.

Afin de réduire la complexité de la gestion du cycle de vie des applications IoT, l'utilisation d'une architecture basée sur les microservices s'avère être indispensable. Une telle architecture exige le choix d'une plateforme adéquate pouvant assurer une facilité de déploiement des services et garantir une bonne performance et une haute disponibilité. Il existe plusieurs technologies qui supportent les microservices. La solution la plus populaire est le Docker, une plateforme open source offrant beaucoup d'avantages.

Les nœuds Fog à déployer peuvent être des commutateurs, des routeurs ou des cartes. Après avoir étudié les caractéristiques de la carte Raspberry Pi, nous avons constaté que l'utilisation de cette composante en tant qu'un nano-ordinateur est faisable et offre plusieurs avantages.

Notre projet propose une solution IoT basée sur le modèle du Fog Computing et utilisant une architecture de microservices conteneurisés. Cette solution qui consiste à appliquer le concept du Smart Building sert à automatiser et optimiser l'utilisation des ressources permettant ainsi d'économiser l'énergie et réduire les coûts.

1.2 Objectifs du projet

La hausse du coût de l'énergie, les multiples prises de conscience environnementales ainsi que les réglementations croissantes relatives au développement durable sont autant de facteurs suscitant la mutation des bâtiments et donnant naissance au concept du Smart building. Ce type de bâtiment présente plusieurs avantages non seulement pour les usagers mais aussi pour l'environnement. Il offre un environnement confortable et réactif et permet d'éviter le gaspillage énergétique. Néanmoins, afin de pouvoir mettre en place une telle solution, on doit disposer d'une capacité de stockage et de calcul et d'une bande passante suffisante permettant la transmission en temps réel des données et le contrôle continu de l'état du bâtiment.

D'autre part, pour diminuer le trafic IoT détecté par les capteurs et les objets connectés et gagner en termes de bande passante et en temps de réponse, nous faisons appel au Fog Computing. Le système que nous proposons consiste à appliquer le concept du Smart building à notre école afin d'optimiser l'utilisation des ressources en utilisant des mini-ordinateurs tels que le modèle RaspBerry Pi. Nous cherchons à concevoir un "Dashboard"

(tableau de bord) permettant de suivre l'état du bâtiment en temps réel et automatiser la gestion d'équipements.

1.3 Démarche méthodologique

Afin de réaliser nos objectifs et réussir l'implémentation de notre application, nous avons procédé comme suit :

- Etudier les concepts IoT et Fog computing.
- Se familiariser avec la notion de conteneurisation et les architectures basées sur les microservices.
- Choisir un scénario d'application et proposer une architecture Fog adaptée à cette application.
- Déterminer les technologies nécessaires à la mise en place de ce scénario.
- Implémenter la solution choisie.
- Tester et valider la solution.

Conclusion

Dans ce chapitre, nous avons expliqué le cadre de notre projet et nous avons cité le concept du smart building que nous avons proposé comme solution pouvant améliorer la gestion des ressources dans notre école.

Chapitre 2

État de l'Art

Introduction

Dans ce chapitre, nous allons fournir les informations nécessaires à la compréhension des concepts de l'Internet des objets, du Fog computing et de conteneurisation.

Nous allons également présenter la technologie Raspberry Pi et ses caractéristiques ainsi que le concept du Smart Building.

2.1 Internet des Objets

Dans cette section, nous présentons l'architecture de l'IoT, ses applications et des défis.

2.1.1 Son architecture

L'Internet des objets souvent appelée IoT est un concept qui prend en compte la présence de nombreux objets, dotés d'identifiants uniques (UID) capables d'interagir les uns avec les autres, coopérer avec d'autres objets, transférer des données, créer de nouvelles applications ou services et atteindre des objectifs communs grâce à des connexions filaires et sans fil et à des systèmes d'adressage uniques [1].

La figure 2.1 représente une architecture IoT. L'architecture des objets Iot est divisées

comme suit : L'étape 1 d'une architecture IoT comprend les éléments en réseau, généralement des capteurs et des actionneurs sans fil. L'étape 2 comprend des systèmes d'agrégation de données de capteurs et une conversion de données analogique-numérique. À l'étape 3, les systèmes informatiques de périphérie effectuent un prétraitement des données avant leur transfert vers le centre de données ou le cloud. Enfin, à l'étape 4, les données sont analysées, gérées et stockées sur des systèmes de centre de données back-end traditionnels.

The 4 Stage IoT Solutions Architecture

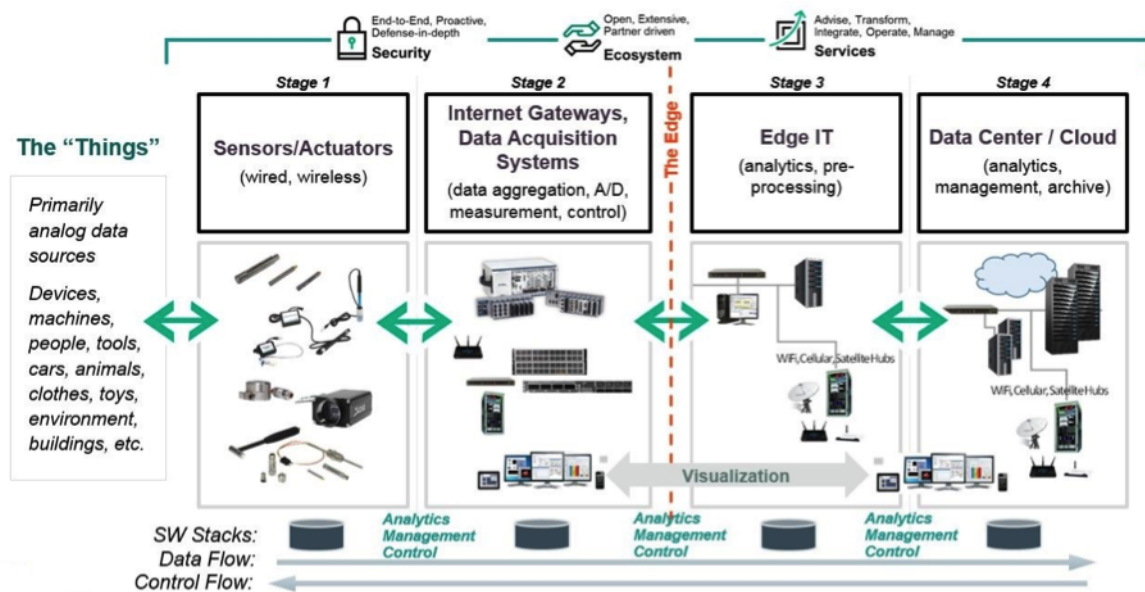


FIGURE 2.1 – Architecture IoT

2.1.2 Ses applications

Les applications de l'IoT sont de plus en plus nombreuses et couvrent tous les domaines de la vie quotidienne.

- Transport et Logistiques : Voitures connectées, voitures autonomes, etc.
- Santé : Équipements chirurgicaux et appareils médicaux connectés, etc.
- Environnement et Énergie : Bâtiments intelligents, systèmes d'alertes aux catastrophes, recyclage intelligent de l'eau, etc.
- Industrie : Usines Connectées, etc.

2.1.3 Ses Défis

En 2020, Le nombre d'appareils connectés devrait atteindre 50 milliards. Une telle explosion sera inévitablement catalyseur de la transformation de l'infrastructure Cloud. En effet, les centres de données traditionnels ont du mal à répondre aux exigences des systèmes IoT : L'émergence de l'IoT requiert la prise en charge de la mobilité et de la distribution géographique des objets connectés, l'énorme quantité de données transmises par les capteurs et les appareils connectés et du délai minimum de traitement de ces données. Le déploiement d'une nouvelle génération d'infrastructures est devenu ainsi crucial pour pouvoir répondre à ces besoins.

2.2 Fog Computing

Le Fog Computing est un modèle qui crée une infrastructure informatique distribuée plus proche de la périphérie du réseau permettant d'exécuter des tâches simples nécessitant une réponse rapide [2]. La figure 2.2 représente le modèle du Fog Computing.

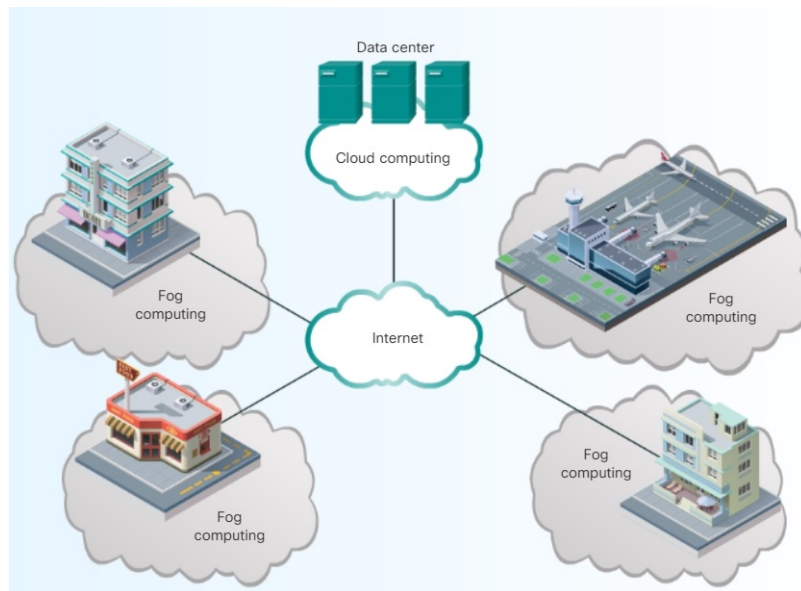


FIGURE 2.2 – Modèle du Fog Computing [2]

2.2.1 Son architecture

L'architecture du Fog computing est une architecture à trois niveaux [3] selon la Figure 2.3.

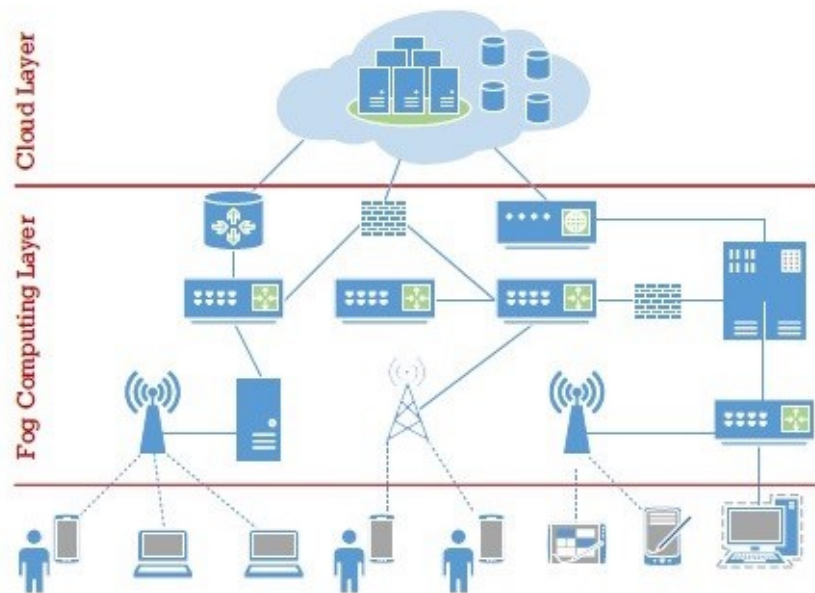


FIGURE 2.3 – Architecture du Fog Computing

- La couche Terminale : Comprend les différents dispositifs IoT (capteurs, téléphones portables, cartes ...) souvent appelés nœuds terminaux. Ces appareils largement distribués par topographie sont chargés de détecter les informations et les transmettre à la couche supérieure pour la manipulation et le stockage.
- La couche Fog : Composée de routeurs, commutateurs, stations de base, serveurs spécifiques... Ces nœuds (Fog Nodes) sont largement distribués entre les gadgets finaux et le Cloud. Les dispositifs d'extrémité peuvent s'associer aux nœuds du Fog pour acquérir des administrations. Ils ont la capacité de comprendre, transmettre et stocker les informations reçues. Ainsi, les applications sensibles à la latence peuvent être traitées dans la couche Fog.
- La couche Cloud : Dispose de plusieurs serveurs et périphériques de stockage hautement performants pour fournir divers services d'application et d'une capacité puissante de calcul et de stockage permettant de réaliser des analyses de calcul approfondies et de stocker en permanence l'énorme quantité de données qui y provient.

2.2.2 Ses caractéristiques

Le modèle du Fog Computing est doté d'un ensemble de caractéristiques qui permettent de couvrir les failles du modèle Cloud. Parmi ces caractéristiques :

- Faible latence et interactions temps-réel : Les nœuds Fog situés à la périphérie du réseau permettent de réduire la distance entre les périphériques finaux et les ressources Cloud. La latence, le temps d'attente et le temps d'exécution des tâches peuvent être considérablement réduits en raison de la prise de conscience par les nœuds du Fog de leur emplacement logique dans le contexte de tous les systèmes et des coûts de latence pour la communication avec les autres nœuds.
- Distribution géographique : Contrairement au Cloud, le Fog Computing se caractérise par une infrastructure distribuée et les applications Fog peuvent être déployées indépendamment de leurs emplacements.
- Sécurité : Les données sont stockées de façon sécurisée sur des infrastructures locales proche de l'utilisateur final. Les problèmes liés aux cybers menaces peuvent être détectés en temps réel, ce qui peut accélérer et faciliter leur résolution.

2.2.3 Comparaison entre le Cloud et le Fog Computing

Le tableau 2.1 présente une comparaison entre le modèle du Cloud et celui du Fog Computing :

TABLE 2.1 – Comparaison entre le Cloud et le Fog Computing

	Cloud Computing	Fog Computing
Latence	Elevée	Faible et prédictible
Infrastructure	Centralisée	Distribuée
Hétérogénéité	Faible	Elevée
Mobilité	Limitée	Mobilité supportée
Déploiement	Statique	Dynamique
Virtualisation	Lourde(Machines Virtuelles)	Légère (conteneurs)

2.3 Smart Buildings & Fog Computing

Les applications de l'IoT touchent divers domaines de la vie quotidienne et professionnelle. L'un des concepts qui a eu naissance grâce à l'IoT est le concept de smart Building.

Les bâtiments intelligents (Figure 2.4) exploitent une connectivité sans fil omniprésente, des capteurs et des technologies IoT pour communiquer et analyser des données utilisées pour contrôler et optimiser les systèmes de gestion de bâtiments. Une combinaison de solutions IoT est utilisée pour automatiser le contrôle d'accès, les systèmes de sécurité, l'éclairage et les systèmes CVC (chauffage, ventilation et climatisation). Ils offrent plus d'efficacité, de sécurité et de confort, tout en permettant des économies de coûts plus en phase avec les objectifs des propriétaires, des gestionnaires et des locataires.

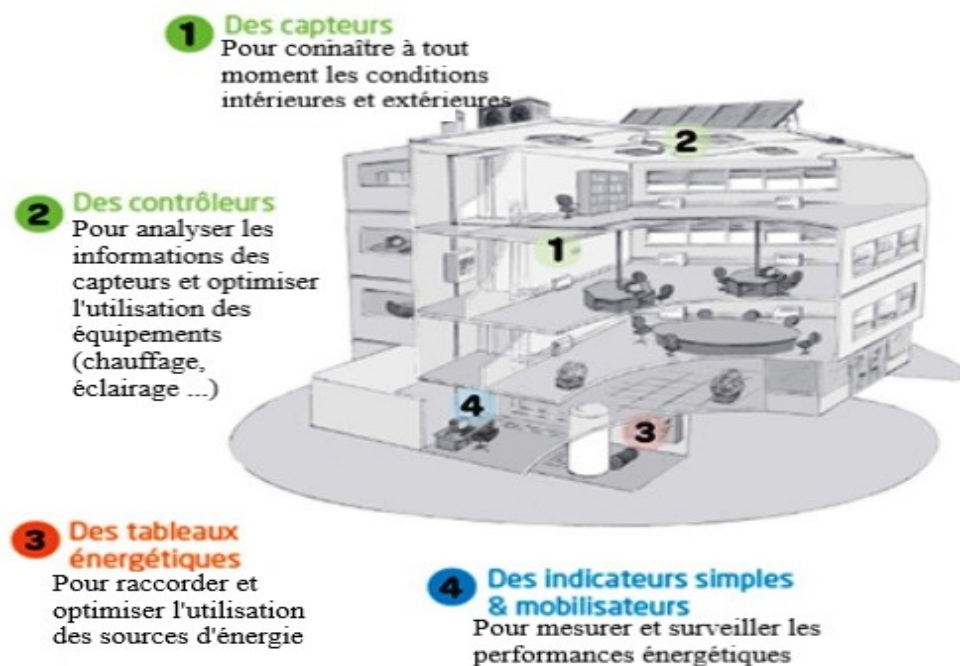


FIGURE 2.4 – Smart Building

C'est alors ici où la notion du fog computing va jouer un rôle très important dans les smart buildings. En effet, le Fog computing prend en charge les applications émergentes des objets IoT qui nécessitent le traitement en temps réel ou une latence prévisible, ce qui aide à gagner en termes de temps d'exécution des processus, de rapidité de réponse des objets et de portabilité. Les figures 2.5 et 2.6 suivantes mettent en évidence l'apport du fog computing dans ce type de contexte.

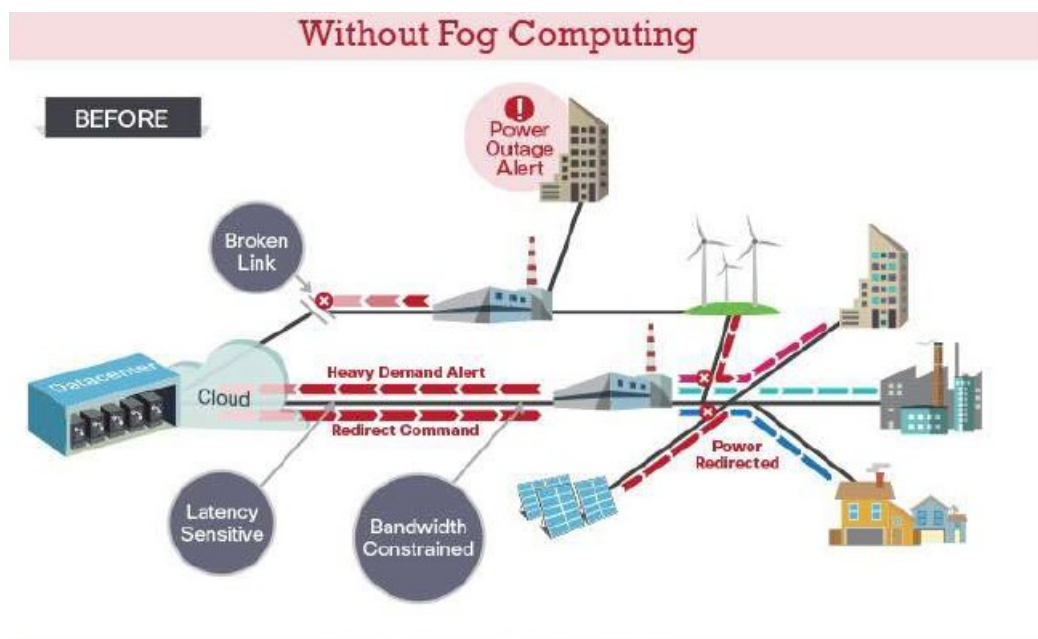


FIGURE 2.5 – Smart Building sans Fog Computing

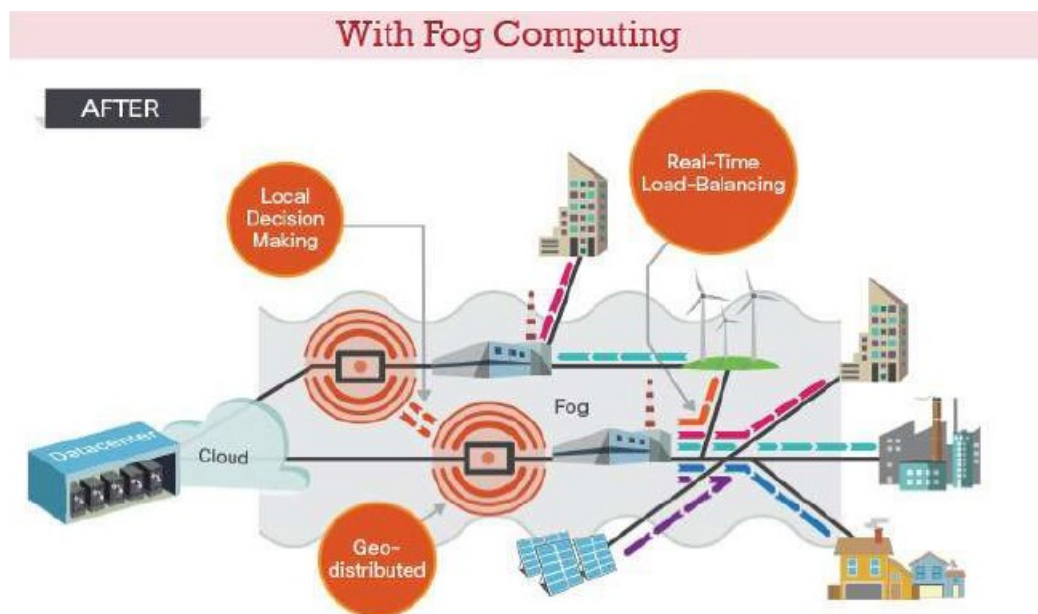


FIGURE 2.6 – Smart Building avec Fog Computing

2.4 DevOps & IoT

DevOps est la combinaison de philosophies, de pratiques et d'outils culturels qui augmente la capacité d'une organisation à fournir des applications et des services à haute vitesse : elle permet de faire évoluer et améliorer ses produits plus rapidement que les organisations utilisant les processus traditionnels de développement de logiciels et de gestion d'infrastructure. Cette vitesse permet aux organisations de mieux servir leurs clients et de rivaliser plus efficacement sur le marché.

Les périphériques IoT connectés ont un cycle de vie allant au-delà du déploiement, de la réparation et de la mise hors service. La connectivité offre la possibilité d'insuffler une innovation incrémentale tout au long du cycle de vie du système. DevOps place les organisations dans la meilleure position pour capitaliser sur cette opportunité.

Au fur et à mesure que les lignes entre la création du logiciel et son utilisation commencent à disparaître, Il en va de même pour les lignes organisationnelles qui séparent les développeurs de systèmes des opérateurs de systèmes. Le concept DevOps a fait ses preuves au niveau de l'entreprise et évolue vers des passerelles et des périphériques de pointe afin de relever les défis uniques du développement et de la gestion des systèmes IoT.

2.4.1 Avantages

- Livraison rapide : Augmenter la fréquence et le rythme des versions afin d'innover et améliorer le produit plus rapidement.
- Fiabilité : Assurer la qualité des mises à jour des applications et des modifications d'infrastructure afin de fournir des résultats fiables à un rythme plus rapide tout en maintenant une expérience positive pour les utilisateurs finaux.
- Sécurité : adopter un modèle DevOps sans sacrifier la sécurité en utilisant des stratégies de conformité automatisées, des contrôles détaillés et des techniques de gestion de la configuration.

2.4.2 Principales technologies offertes

DevOps révolutionne la façon dont les entreprises fournissent des applications sur le marché. Il associe le développement des logiciels et les opérations informatiques, ou les pro-

cessus et services utilisés par le personnel informatique ainsi que par ses clients internes et externes pour s'acquitter de leurs tâches. Ce processus implique un travail intensif. En conséquence, les ingénieurs de DevOps sont constamment à la recherche de nouveaux outils. Parmi outils déjà existants nous citons :

- Ansible : c'est un moteur d'automatisation qui permet aux administrateurs informatiques d'automatiser une partie de leurs tâches quotidiennes. Les entreprises qui utilisent Ansible bénéficient d'une rationalisation et d'une conformité accrues dans leurs environnements informatiques, ainsi que de l'innovation et de la collaboration entre leurs employés.
- Docker : le Docker s'intègre dans n'importe quel environnement et fournit une portabilité complète de la pile pour les applications. Le cadre est fourni avec des conteneurs certifiés que les entreprises peuvent utiliser pour créer des applications sécurisées.
- Git : c'est un outil open source qui aide les développeurs à gérer leurs projets avec rapidité et efficacité. L'une de ses caractéristiques de signature est le fait qu'il s'agit d'un modèle de branchement qui permet aux développeurs de créer plusieurs branches locales ou des pointeurs qui sont indépendants les uns des autres.

2.5 Conteneurisation

La conteneurisation est une approche de virtualisation qui permet d'utiliser le même noyau du système d'exploitation pour exécuter plusieurs processus isolés.

Cette technologie permet de pallier le problème de l'exécution des services logicielles dans divers environnements.

L'application, ses fichiers de configuration, ses dépendances ainsi que les binaires et les bibliothèques nécessaires sont empaquetés sous forme d'image de conteneur.

Elle peut être ensuite testée et déployée sous forme d'instance de cette image sur le système d'exploitation hôte [4].

Conteneurs vs Machine virtuelle

Comparés aux machines virtuelles qui utilisent un hyperviseur pour émuler le matériel permettant ainsi l'exécution en parallèle de plusieurs systèmes d'exploitation, les conteneurs sont beaucoup plus légers. En effet, un conteneur s'exécute de manière native sous le système d'exploitation et partage le noyau de la machine hôte avec d'autres conteneurs. Il exécute un processus discret mobilisant moins de ressources que les machines virtuelles. Le tableau 2.2 et la figure 2.7 expliquent la différence entre les machines virtuelles et les conteneurs :

TABLE 2.2 – Comparaison entre les conteneurs et les machines virtuelles [5]

	Machine virtuelle	Conteneur
Système d'exploitation	Les machines virtuelles permettent à plusieurs systèmes d'exploitation de s'exécuter simultanément sur un seul système matériel	Les conteneurs partagent le même noyau du système d'exploitation hôte
Isolation	Les fichiers et les bibliothèques ne peuvent pas être partagés entre les machines virtuelles invitées ou entre une machine virtuelle et le système hôte	Les sous répertoires peuvent être partagés d'une manière transparente
Stockage	Gigaoctet	Mégaoctet
Temps de démarrage	Rapide	Lent
Sécurité	Dépend de l'hyperviseur	Un contrôle d'accès est nécessaire

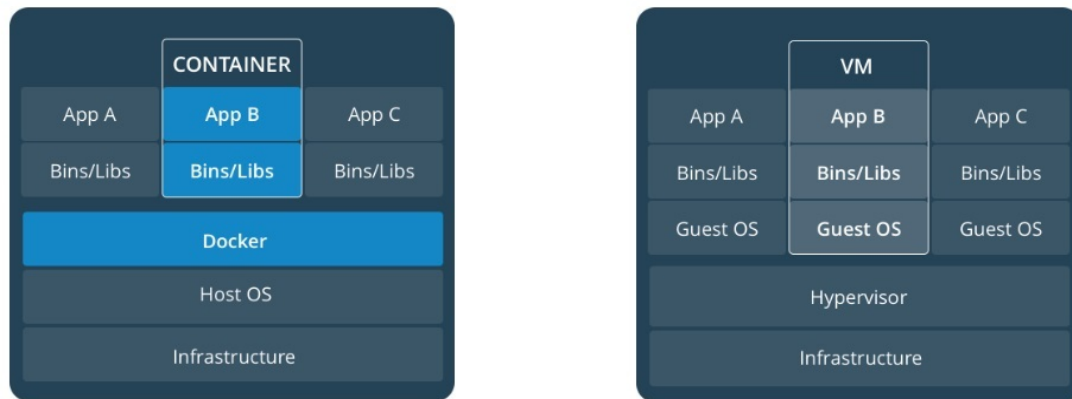


FIGURE 2.7 – Machine virtuelle vs Conteneurs

Afin de faciliter le déploiement des conteneurs, plusieurs plateformes de conteneurisation ont été conçues. Parmi les plateformes les plus populaires nous citons Docker qui sera présenté dans la section suivante.

2.6 Docker

Docker est une plateforme open source permettant le développement, le déploiement et l'exécution des applications avec des conteneurs. Cette technologie utilise le noyau Linux et des fonctions de ce noyau afin de séparer les processus pour qu'ils puissent s'exécuter de façon indépendante.

Docker est associé à un modèle de déploiement basé sur une image. Ce qui permet de partager une application ou un ensemble de services ainsi que leurs dépendances entre plusieurs environnements. Docker permet également d'automatiser et accélérer le déploiement des applications au sein d'un environnement de conteneurs ainsi que le contrôle et l'attribution des versions de ces applications [6].

La technologie Docker offre une multitude d'avantages parmi lesquels nous citons :

- Livraison cohérente et rapide des applications : Permet aux développeurs de travailler dans des environnements standardisés à l'aide de conteneurs locaux ce qui leur permet de rationaliser le cycle de vie de leurs applications.
- Déploiement et Mise à l'échelle réactifs des applications : La portabilité et la légèreté offertes par Docker facilitent la gestion dynamique des charges de travail, la

suppression et l'extension des applications en fonction des besoins des développeurs en temps quasi réel.

- Optimisation des ressources : Idéal pour les environnements à haute densité et pour les déploiements de petite et moyenne taille où on est limité par les ressources.

Architecture

L'architecture Docker est une architecture client-serveur. Le Docker Daemon se charge de la construction, de l'exécution et de la distribution des conteneurs Docker. Le Docker Client ainsi que le Docker Daemon peuvent s'exécuter sur le même système ou peuvent être connectés à distance. Ces derniers communiquent à l'aide d'une API REST via des sockets UNIX ou une interface réseau [7]. La figure 2.8 permet de mieux comprendre l'architecture Docker.

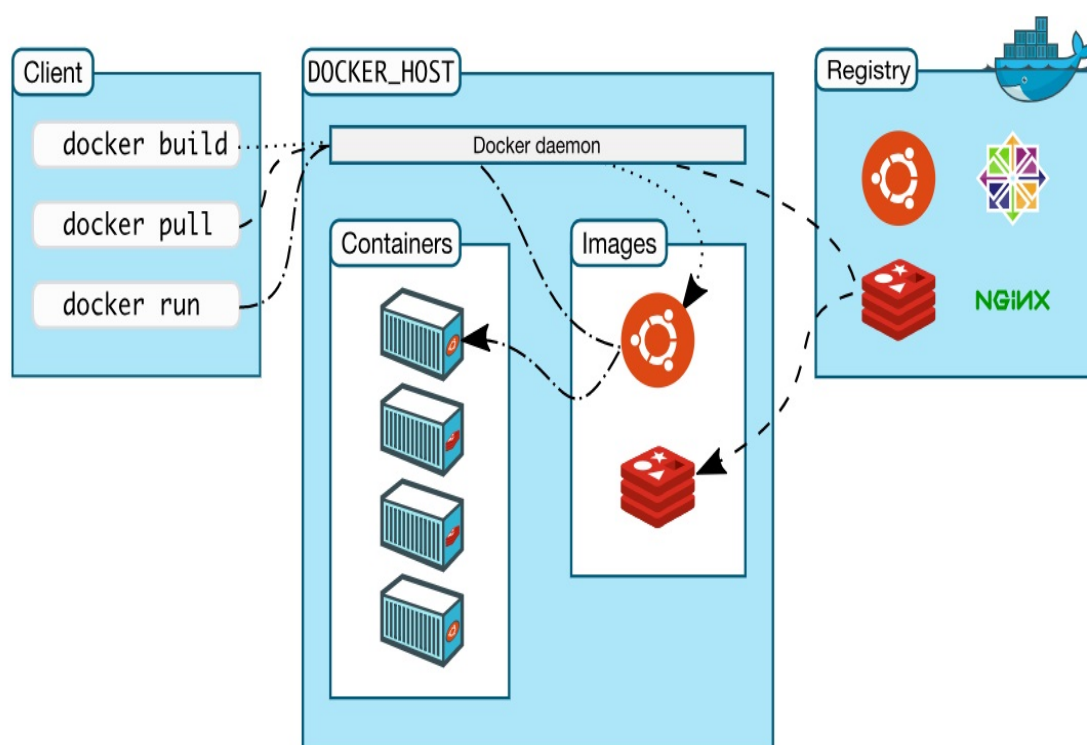


FIGURE 2.8 – Architecture Docker

Le tableau 2.3 regroupe les différents composants Docker.

TABLE 2.3 – Composants Docker [7]

Composant	Rôle
Docker daemon	Répond aux requêtes de l'API Docker et gère les objets Docker. Peut communiquer avec d'autres Docker daemon pour gérer les services Docker.
Docker client	Agent principal utilisé par de nombreux utilisateurs Docker pour interagir avec Docker. Peut communiquer avec plusieurs daemons.
Docker registry	Permet de stocker des images Docker. Docker Hub est un registre public. Par défaut, Docker est configuré pour rechercher des images sur Docker Hub. On peut également gérer des registres privés. Les commandes docker pull ou docker run permettent d'extraire les images requises du registre configuré. La commande docker push permet de transférer une image est dans le registre configuré.
Objets Docker	Image : Modèle en lecture seule avec des instructions de création de conteneur Docker. Souvent elle est basée sur une autre image avec une personnalisation supplémentaire. Conteneur : Instance exécutable d'une image. Il est défini par son image ainsi que les options de configuration fournies lors de sa création ou son démarrage. Service : Permettent de redimensionner les conteneurs sur plusieurs Docker daemon formant ce qu'on appelle "swarm".
Dockerfile	Docker peut créer des images automatiquement suivant les instructions d'un Dockerfile. Il s'agit d'un document texte contenant les commandes nécessaires pour former une image.

La figure 2.9 illustre les liens entre les différents objets Docker.

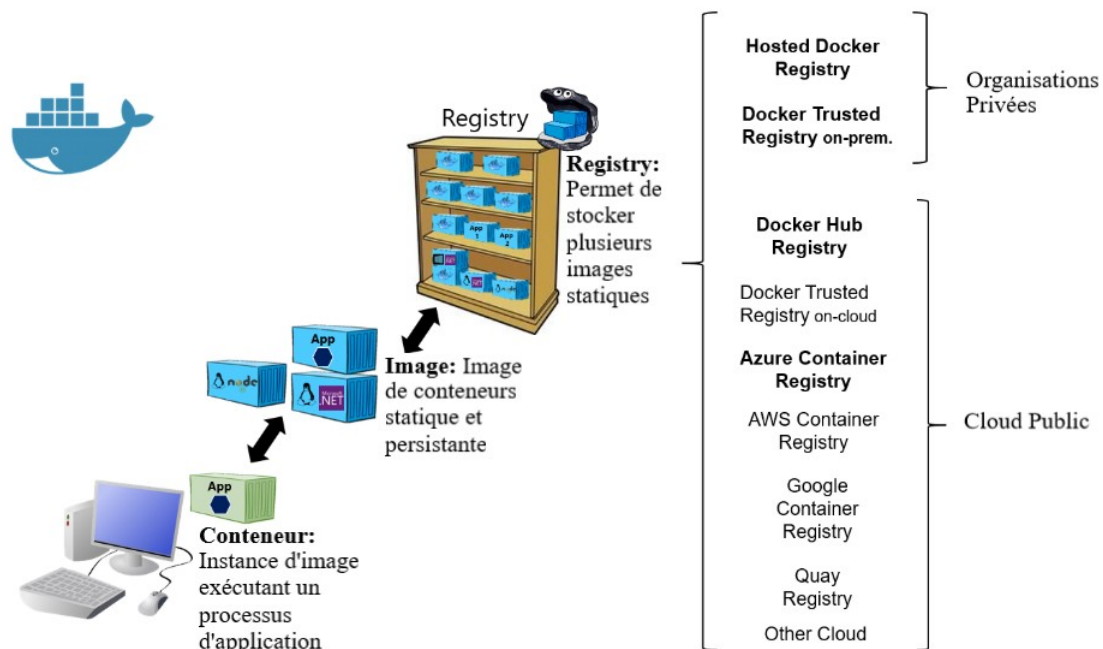


FIGURE 2.9 – Taxonomie des termes et concepts Docker

2.7 Architecture de microservices

L'architecture de microservices permet de diviser de grandes applications en des petits services qui peuvent interagir et communiquer les uns avec les autres. Chaque service comporte des éléments livrables indépendants. Chaque microservice peut être déployé, mis à niveau, mis à l'échelle ou remplacé séparément. La communication entre les microservices se fait à l'aide d'une connexion réseau via des appels HTTP (demande / réponse). Les sockets web, les files de messages ainsi que les appels publish / subscribe peuvent également être utilisés pour connecter des services autonomes. La figure 2.10 explique le principe de l'architecture de microservices.

L'architecture des microservices offre plusieurs avantages :

- Agilité : Les cycles de développement étant raccourcis l'architecture des microservices permet un déploiement d'application et des mises à jour plus agiles.

- Résilience : Dans le cas de défaillance d'un service, les autres services ne seront pas paralysés. Ce qui permet de garantir une bonne performance avec moins d'interruptions des services.
- Productivité : Les développeurs peuvent plus facilement appréhender, mettre à jour et améliorer les services, ce qui permet d'accélérer les cycles de développement en particulier lorsque ces derniers sont associés à des méthodologies de développement agiles.
- Evolutivité : Dans le cas où la demande de certains services augmente, il est possible de déployer sur plusieurs serveurs et infrastructures pour répondre à leurs besoins.

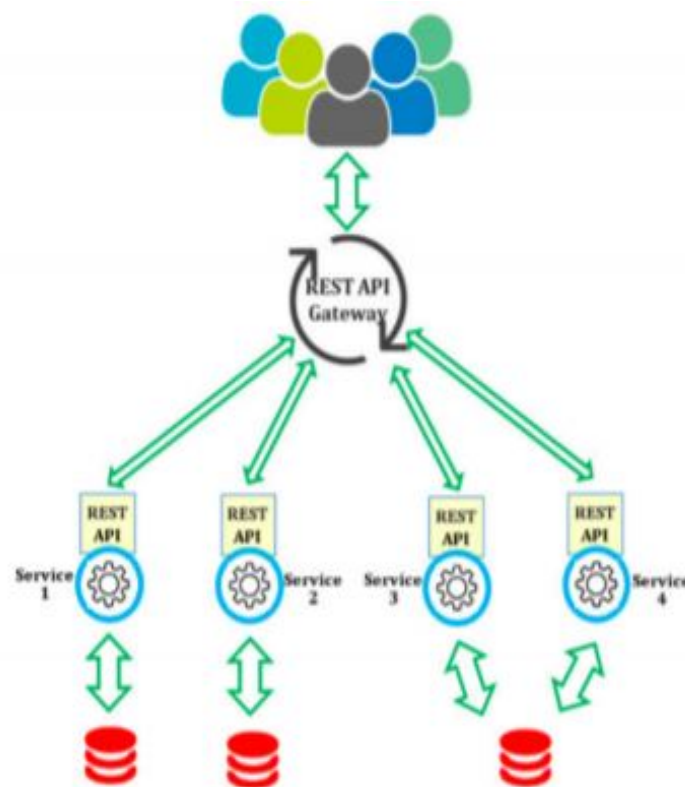


FIGURE 2.10 – Architecture de microservices

2.8 Raspberry Pi

Le Raspberry Pi (figure 2.11) est un ordinateur monocarte à faible coût qui a été créé en 2012 par David Braben dans le but de promouvoir l'étude de l'informatique et de sujets connexes dans les écoles et dans les pays en développement. Il est doté d'un processeur

basé sur ARM et fonctionnant sur une bande de fréquence qui va de 700 à 1,2 GHz et d'une mémoire pouvant atteindre 1 Go. Les principaux composants incluent des ports USB, des ports Ethernet et une carte SD. Le système d'exploitation utilisé pour faire fonctionner un Raspberry Pi est Raspbian, une distribution Linux basée sur Debian. En plus des composants précédemment cités, Raspberry Pi est équipé de nombreuses interfaces permettant d'interagir avec de petits appareils électroniques. Raspberry Pi utilise Python comme langage de programmation principal mais prend en charge d'autres langages de programmation tels que C / C ++, et Java [8].

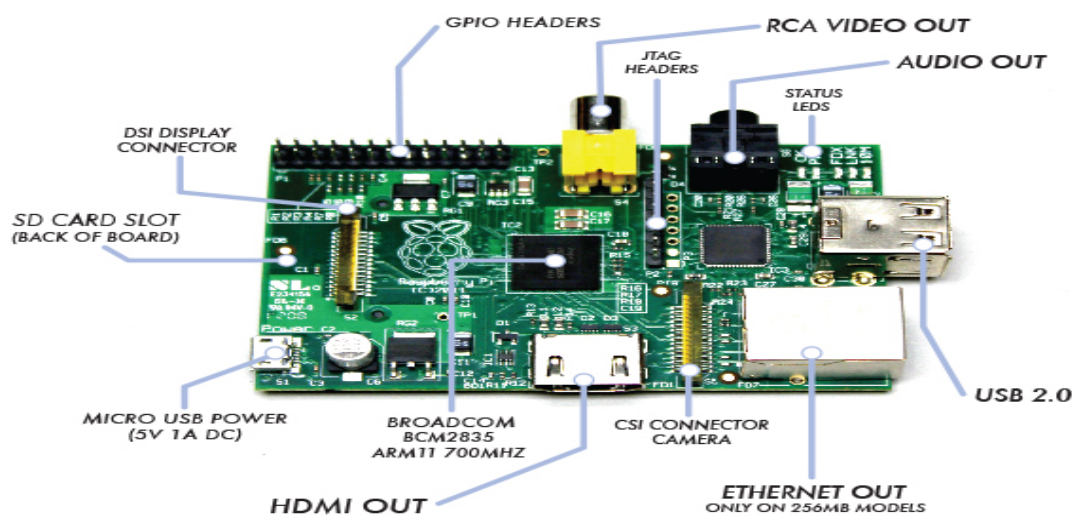


FIGURE 2.11 – Raspberry Pi

Conclusion

L'architecture Fog a réussi à couvrir les failles du modèle Cloud et surmonter les défis posés par les applications IoT. Pour des raisons de performance, d'intégration et de maintenabilité, l'utilisation des microservices s'avère être indispensable lors de la conception d'une architecture IoT. Après une étude approfondie des différents concepts détaillés précédemment nous avons pu concevoir notre propre application qui sera présentée dans le chapitre suivant.

Chapitre 3

Conception du projet

Introduction

Dans ce chapitre nous abordons la partie conception du projet dans laquelle nous allons détailler le scénario que nous avons choisi. Nous présentons ensuite le diagramme de séquence de ce scénario. Enfin, nous allons citer les différentes technologies utilisées lors du développement de ce projet et justifier le choix de ces technologies.

3.1 Scénario d'application

Il a été démontré que la présence et le comportement des occupants dans les bâtiments avaient un impact important sur la demande de chauffage, climatisation, la ventilation, la consommation d'énergie d'éclairage ainsi que sur les commandes du bâtiment. Un comportement inconscient et une utilisation irresponsable de l'énergie peut affecter la performance énergétique d'un bâtiment.

La solution que nous proposons consiste à appliquer le concept du smart building à notre école. Nous allons développer un Dashboard qui nous permet de suivre l'état du bâtiment (température, humidité, lumière) et d'automatiser la gestion des ressources.

Description du scénario

- Le nœud Fog (Raspberry Pi) reçoit les données détectées par les capteurs sous format JSON depuis une carte Arduino.

- Ces données sont stockées dans une base de données implémentée dans un conteneur à travers docker compose sur le Raspberry Pi.
- Elles sont ensuite simulées dans un Dashboard de contrôle implémenté dans plusieurs conteneurs sur le Raspberry Pi.
- Selon les valeurs des données collectées, le Raspberry Pi va décider l'action à déclencher (Allumer Led, Lancer La climatisation, Lancer le Chauffage. . .).
- L'utilisateur reçoit une notification indiquant l'action à faire. Les données détectées par les capteurs seront envoyées du Raspberry Pi vers le cloud pour visualiser l'état des capteurs sur la plateformes UBIDOTS.

3.2 Diagramme de séquences

Les figures 3.1 et 3.2 ci-dessous présentent les deux scénarios précédemment décrits.

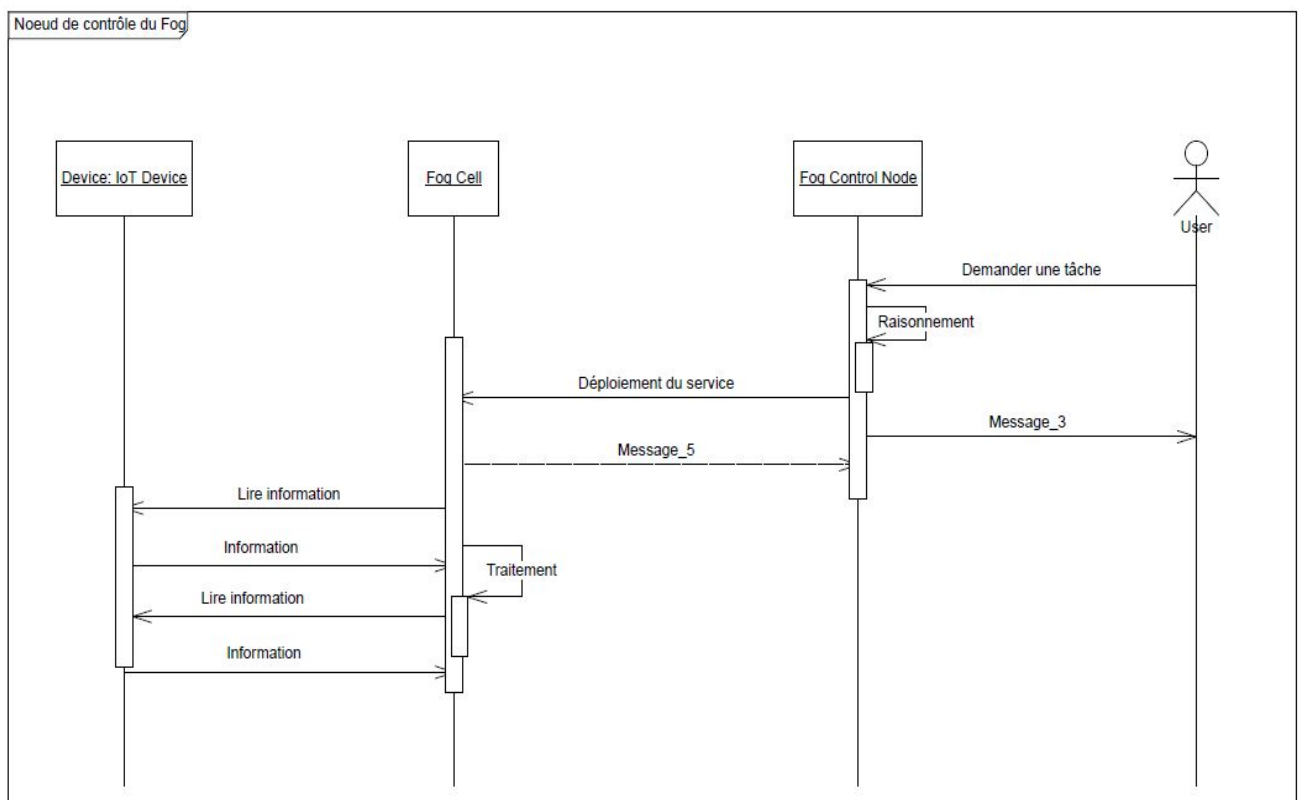


FIGURE 3.1 – Traitement des tâches dans un nœud Fog

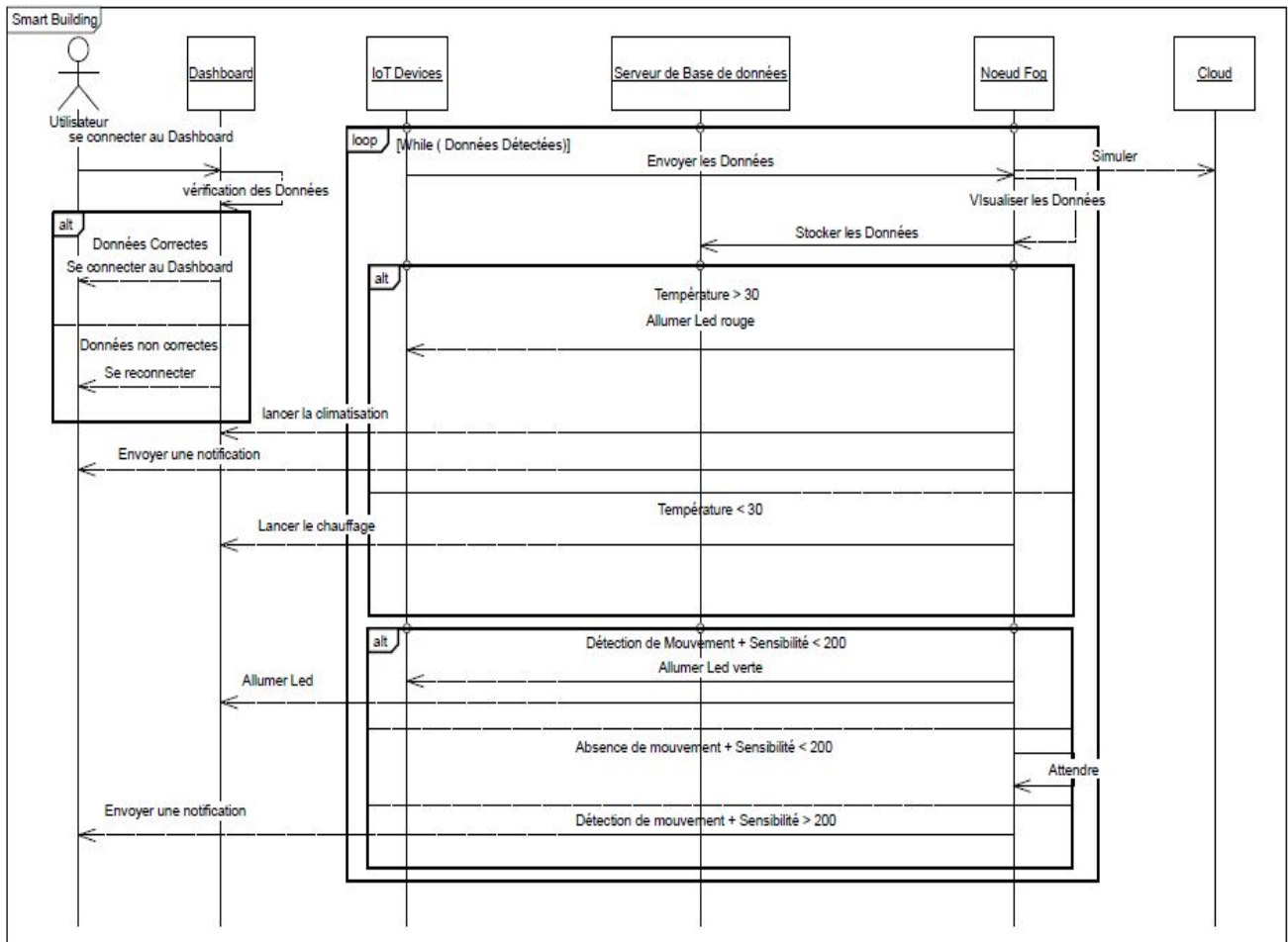


FIGURE 3.2 – Traitement des données dans le Fog

3.3 Spécifications techniques

Dans cette section, nous énumérons toutes les composantes logicielles nécessaires pour implémenter notre projet.

3.3.1 Docker Compose

Docker Compose est un outil qui permet de définir et d'exécuter des applications Docker à conteneurs multiples. Un fichier YAML est utilisé pour configurer les services de l'application. Ensuite, avec une seule commande, on peut créer et démarrer tous les services

qui ont été configurés à la fois [9].

Utiliser Compose est un processus à trois étapes :

1. On définit l'environnement de l'application avec un fichier Docker afin qu'il puisse être reproduit n'importe où.
2. On définit les services qui composent l'application dans un fichier `docker-compose.yml` pour qu'ils puissent être exécutés ensemble dans un environnement isolé.
3. Enfin on exécute la commande `docker-compose up` et Compose démarre et exécute l'intégralité de l'application.

3.3.2 Langage de programmation Python

Python est un langage de programmation de haut niveau largement utilisé.

Ce langage polyvalent fonctionne sur différentes plateformes (Linux, Raspberry Pi, ...) et utilise une approche de programmation modulaire et orientée objet.

Utilisant une syntaxe simple combinée à des types de données évolués, ce langage permet aux développeurs d'écrire des programmes à la fois compacts et lisibles. Python rend alors la compréhension de la structure du programme plus facile à apprendre que d'autres langages de programmation populaires.

3.3.3 Langage de programmation C

Il s'agit d'un Langage structuré à usage général conçu pour traiter les tâches d'un programme en les mettant dans des blocs. Il permet de produire des programmes efficaces et de générer des codes compacts et rapides.

3.3.4 Base de données : MySQL, PHPMyAdmin

PHPMyAdmin est un logiciel gratuit écrit en PHP et destiné à la gestion de l'administration de MySQL sur le Web. Ce logiciel prend en charge un large éventail d'opérations sur MySQL. Les opérations fréquemment utilisées (la gestion des bases de données, des tables, des colonnes, des relations, des index, des utilisateurs, des autorisations ...) peuvent être effectuées à l'aide d'une interface utilisateur, tout en permettant d'exécuter directement toute instruction SQL.

3.3.5 Serveur APACHE2

Apache Web Server permet de créer des serveurs Web pouvant héberger un ou plusieurs sites Web basés sur HTTP. Les fonctionnalités notables incluent la possibilité de prendre en charge plusieurs langages de programmation, les scripts côté serveur, un mécanisme d'authentification et le support de base de données..

Apache est largement utilisé par les sociétés d'hébergement Web dans le but de fournir un hébergement partagé. Par défaut, Apache Web Server prend en charge et distingue différents hôtes résidant sur le même ordinateur.

3.3.6 Framework de développement Web

3.3.6.1 Flask

Flask est un framework open source de développement web en python. Flask fournit des outils, des bibliothèques et des technologies permettant de créer une application Web. Il ne comporte aucune couche d'abstraction de base de données, aucune validation de formulaire ni tout autre composant pour lequel des bibliothèques tierces préexistantes fournissent des fonctions communes. Cependant, Flask prend en charge des extensions pouvant ajouter des fonctionnalités d'application comme si elles étaient implémentées dans Flask même. Des extensions existent pour les mappeurs relationnels-objet, la validation de formulaire, la gestion du téléchargement, diverses technologies d'authentification ouverte et plusieurs outils communs liés au framework [10].

3.3.6.2 Dash

Dash est un framework Python productif pour la construction d'applications Web.

Dash est idéal pour créer des applications de visualisation de données avec des interfaces utilisateur hautement personnalisées en Python.

3.4 Connectivité du matériel

Notre application inclut plusieurs périphériques et terminaux. Dans cette section, nous allons expliquer la communication entre les différents composants : le nœud Fog : Raspberry Pi, les capteurs, le serveur de Base de données et les terminaux.

la figure 3.3 explique la connectivité entre les différentes parties de notre système.

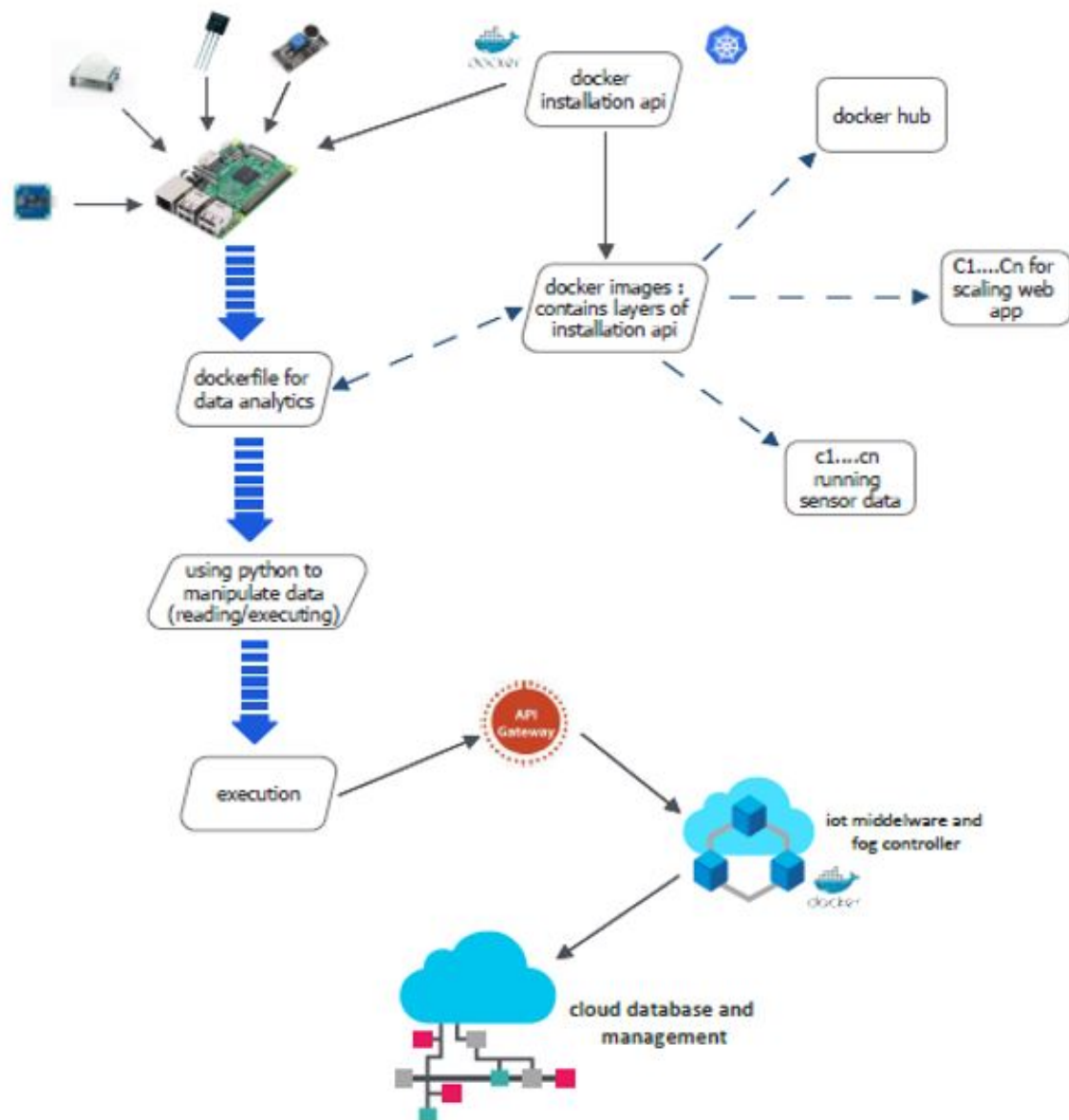


FIGURE 3.3 – Interconnexion des composants

3.4.1 Liaison Capteurs - Fog

Les capteurs dont nous disposons étant des capteurs analogiques ne sont pas directement utilisable sur le Raspberry Pi car ils délivrent une tension/un courant (donc analogique) et nécessite la mise en œuvre d'une carte de mesure avec convertisseur analogique numérique. Nous utilisons donc une carte Arduino qui peut être connectée à toute sorte de capteurs afin de transmettre les données détectées au Raspberry.

Communication entre Arduino et Raspberry Pi

La communication entre la carte Arduino et le nœud Fog sera assurée par une liaison série utilisant le protocole UART (Universal Asynchronous Receiver Transmitter), l'une des techniques de communication série les plus simples et les plus couramment utilisées. La fonction de UART est de convertir les données entrantes et sortantes en flux binaire série.

3.4.2 Liaison Base de données - Fog

Les capteurs envoient leurs données de la carte Arduino vers le Raspberry Pi sur laquelle nous avons conçu une base de donnée à travers un serveur Apache2 installé depuis le Docker Hub avec le fichier docker compose.

3.4.3 Liaison Terminaux - Fog

Les terminaux sont généralement des ordinateurs ou des smartphones. Cette liaison peut être utilisée dans deux cas. L'administrateur a besoin de savoir l'état en cours de chaque capteur sans qu'il se déplace et sera donc capable de les manipuler à travers un Dashboard implémenté sur le Raspberry Pi.

Conclusion

Dans ce chapitre nous avons présenté le scénario de notre application et nous avons spécifié les différentes technologies que nous avons utilisées lors du développement de notre solution.

Chapitre 4

Réalisation du projet

Introduction

Dans ce chapitre nous allons décrire les différentes étapes d'implémentation de notre solution.

4.1 Environnement matériel

4.1.1 Carte Raspberry Pi 3 B+

Ce Modèle est doté d'un processeur quad core 64 bits fonctionnant à 1,4 GHz, de réseaux locaux sans fil allant de 2,4 GHz jusqu'à 5 GHz, de Bluetooth 4.2 / BLE et d'Ethernet rapide. Elle admet plusieurs spécifications :

- Broadcom BCM2837B0, SoC 64 bits Cortex-A53 (ARMv8) à 1,4 GHz.
- Tête GPIO étendue à 40 broches.
- Port Micro SD pour charger votre système d'exploitation et stocker des données.
- Ethernet Gigabit sur USB 2.0 (débit maximal 300 Mbps).
- Entrée d'alimentation CC 5V / 2,5A



FIGURE 4.1 – Carte Raspberry Pi

4.1.2 Carte Arduino UNO

Arduino consiste à la fois en une carte de circuit physique programmable souvent appelée microcontrôleur et en un logiciel ou IDE (environnement de développement intégré) utilisé pour écrire et télécharger du code informatique sur la carte physique.

La plate-forme Arduino est devenue très populaire parmi les débutants en électronique. Contrairement à la plupart des cartes de circuit programmables précédentes, l'Arduino n'a pas besoin de matériel séparé (appelé programmeur) pour charger le nouveau code sur la carte, on peut simplement utiliser un câble USB. De plus, l'IDE Arduino utilise une version simplifiée du C ++, ce qui facilite l'apprentissage de la programmation.

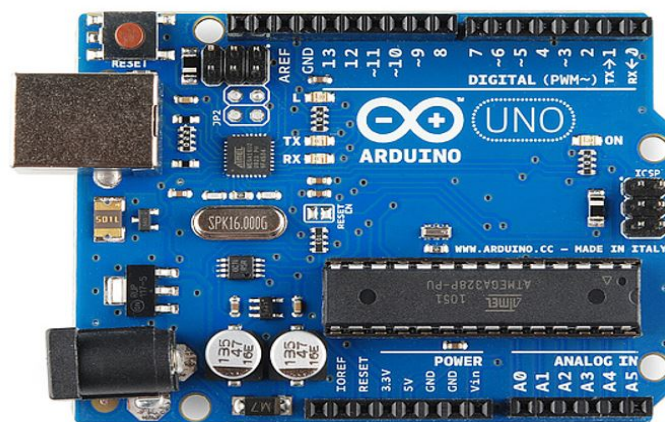


FIGURE 4.2 – Carte Arduino Uno

4.1.3 Capteurs

Afin d'acquérir des données, nous avons utilisé plusieurs capteurs dans le but de modéliser le scénario décrit le chapitre précédent :

- Photorésistance (photocell) : Permet de détecter la présence de lumière. Sa résistance est proportionnelle à la quantité de lumière qu'il reçoit.
- Capteur de mouvement PIR (PIR sensor) : Les capteurs PIR peuvent détecter des mouvements dans un rayon de 5 mètres grâce aux radiations infrarouges.
- Capteur de température et d'humidité (DHT11) : Permet la détection de la température et l'humidité d'une salle
- Un buzzer : Permet de renvoyer du son selon la fréquence donnée.

4.2 Mise en place de la solution

4.2.1 Câblage du matériel

Afin de garantir le bon fonctionnement de notre système, nous devons s'assurer que tout le matériel soit connecté. Le Raspberry Pi doit être lié à l'ordinateur via un câble Ethernet et la connexion se fait par un simple SSH. Ensuite, nous devons connecter notre Raspberry Pi à la carte Arduino. Ces deux cartes communiquent ensemble à l'aide d'une liaison UART. Les figures 4.3 montre le câblage entre les différents composants.

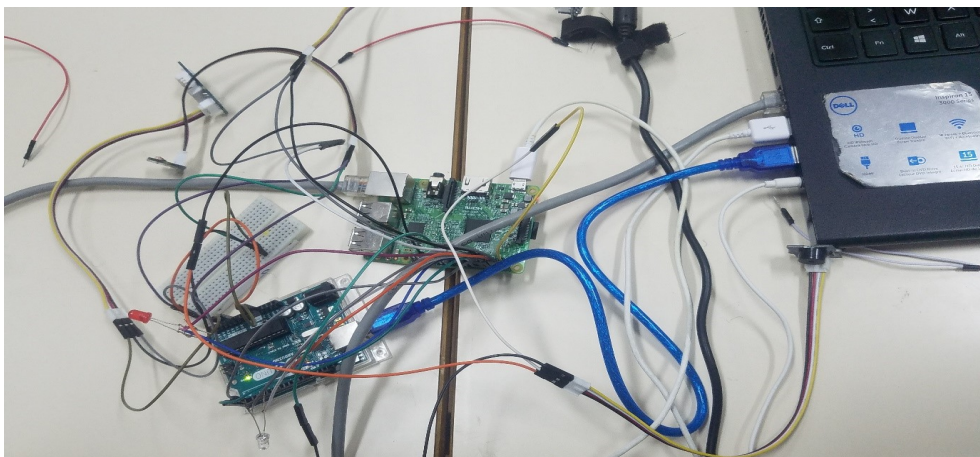
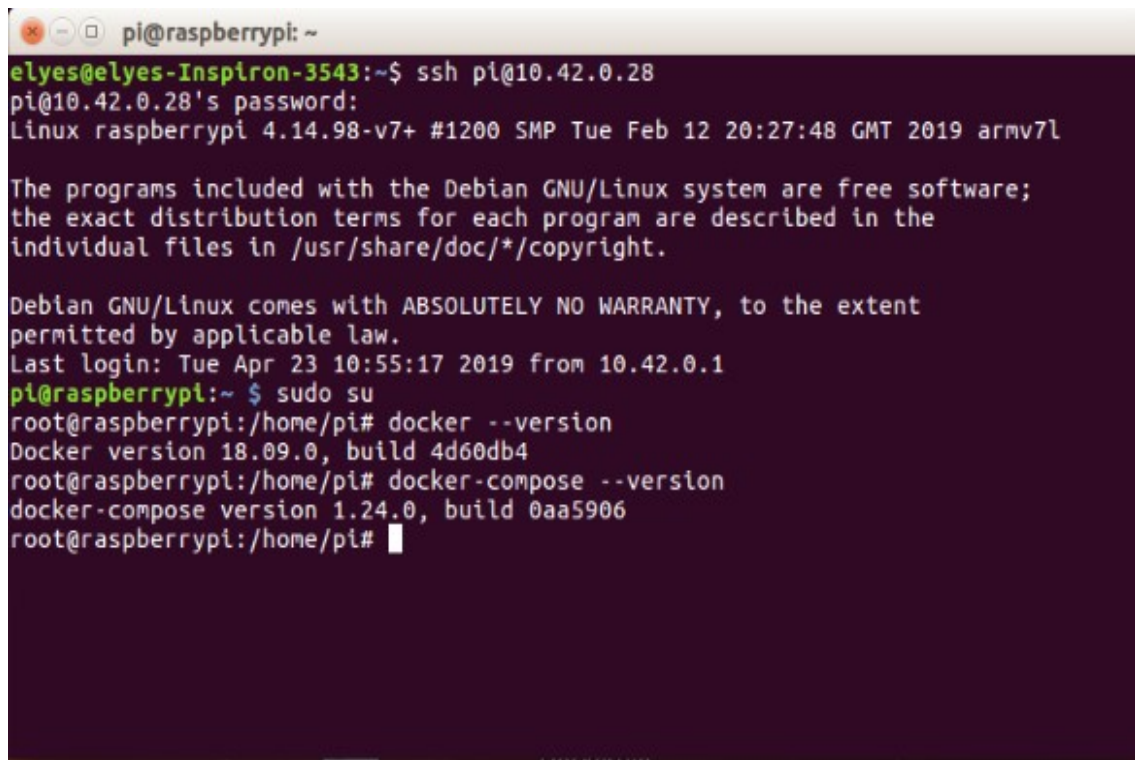


FIGURE 4.3 – Câblage du matériel

4.2.2 Installation de Docker et Docker Compose

Le traitement devant se faire sur Le noeud Fog, Docker ainsi que Docker Compose doivent donc être installés sur le Raspberry. La figure 4.4 représente l'une des étapes d'installation de ces deux outils :



```
pi@raspberrypi: ~  
elyes@elyes-Inspiron-3543:~$ ssh pi@10.42.0.28  
pi@10.42.0.28's password:  
Linux raspberrypi 4.14.98-v7+ #1200 SMP Tue Feb 12 20:27:48 GMT 2019 armv7l  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Apr 23 10:55:17 2019 from 10.42.0.1  
pi@raspberrypi:~ $ sudo su  
root@raspberrypi:/home/pi# docker --version  
Docker version 18.09.0, build 4d60db4  
root@raspberrypi:/home/pi# docker-compose --version  
docker-compose version 1.24.0, build 0aa5906  
root@raspberrypi:/home/pi#
```

FIGURE 4.4 – Installation Docker et Docker Compose

Nous pouvons constater que la version la plus récente de Docker et de Docker Compose sont installés avec succès.

4.2.3 Création des scripts

Les capteurs doivent détecter des données et les envoyer au Raspberry. Ceci est assuré par un code écrit en langage C sous Arduino.

Notre solution permet de mesurer la température, l'humidité, la sensibilité de la lumière et détecter la présence d'un mouvement. Dans le cas où la température dépasse une valeur seuil que nous avons fixée, une Led rouge s'allume et un Buzzer se met en action. Si de plus il y aura détection de mouvement dans la salle et la sensibilité de lumière mesurée

est inférieur à une valeur minimale fixée, la Led verte s’allume. Pour avoir un bon contrôle de tout les cas possible nous avons ecrit un script en langage python (voir annexe B) sous le Raspberry Pi. C’est le principe du fog. La figure 4.5 représente des données détectées par les capteurs.

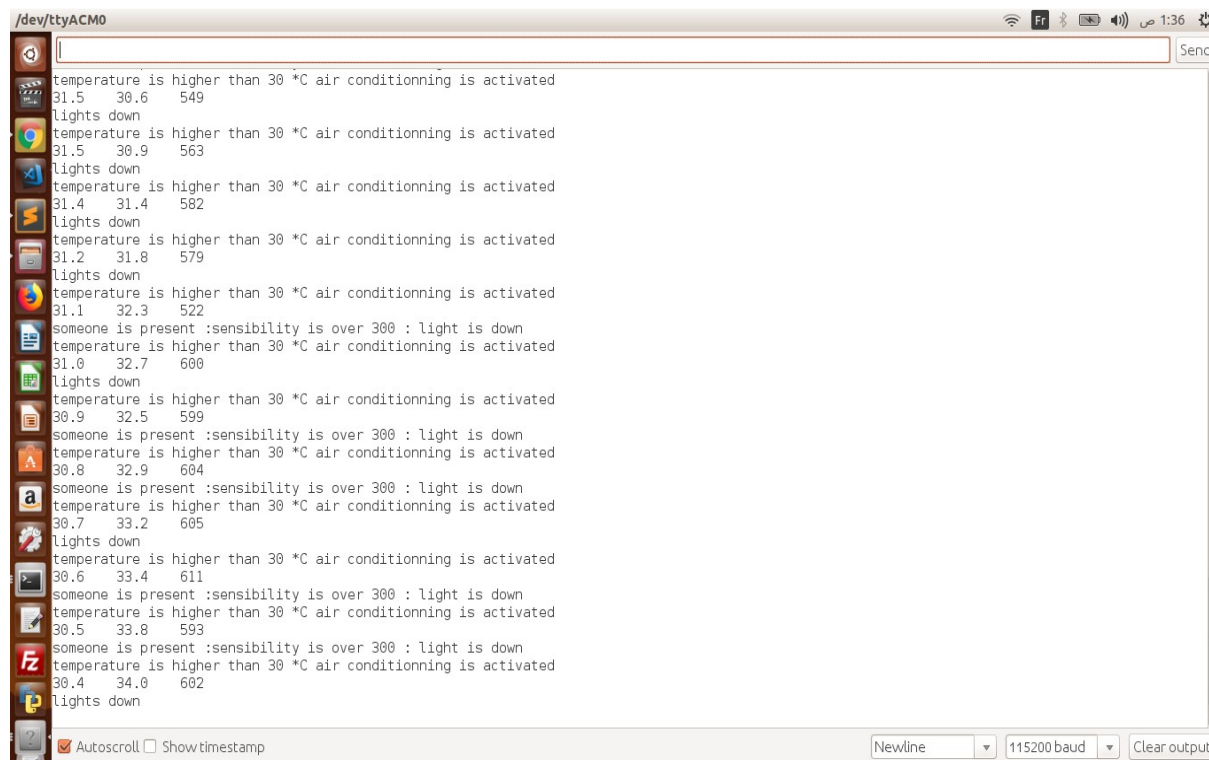


FIGURE 4.5 – Résultat du test du scénario

4.2.4 Conteneurisation

Dans cette étape nous avons utilisé les performances offertes par Docker pour conteneuriser les scripts, les serveurs et notre Dashboard. Avec ce service nous aurons une communication rapide entre les conteneurs qui sont totalement isolés les uns des autres, cette isolation est trop importante dans des applications à l’échelle des entreprises pour garantir que si une fonctionnalité va tomber en panne l’application restera fonctionnelle et ceci est dû à l’utilisation de plusieurs conteneurs pour un même package de l’application. La figure 4.6 présente quelques conteneurs que nous avons créés avec Docker Compose.


```

pi@raspberrypi: ~
4e8a144c71b3  phpmyadmin/phpmyadmin:4.8  "/run.sh supervisord..."  29 hours ago  Restarting (1) 24 seconds ago
phpmyadmin
8dcaecda6c34  portainer/portainer        "/portainer -H unix:..."  33 hours ago  Up 17 hours  0.0.0.0:9000->9000/
tcp  pi_portainer_1
root@raspberrypi:/home/pi# docker ps -a
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
38c127427cc4   187e                              "/usr/bin/entry.sh p..."  19 hours ago  Exited (1) 19 hours ago
c3559a7b5723   hypriot/rpi-mysql             "/entrypoint.sh mysql..."  19 hours ago  Exited (0) 17 hours ago
new_db_1
87ac9044f4df   dc67ade9f22c                  "/bin/sh -c 'sudo ap..."  24 hours ago  Exited (100) 24 hours ago
zen_torvalds
3c33c1498d6a   tobi312/rpi-apache2:latest     "/entrypoint.sh /usr..."  29 hours ago  Exited (255) 17 hours ago  10.42.0.28:9090->4
5/tcp, 0.0.0.0:32771->80/tcp, 0.0.0.0:32770->443/tcp  apache
9acc6311a590   hypriot/rpi-mysql             "/entrypoint.sh mysql..."  29 hours ago  Up 17 hours  3306/tcp
mysql
4e8a144c71b3   phpmyadmin/phpmyadmin:4.8     "/run.sh supervisord..."  29 hours ago  Restarting (1) 44 seconds ago
phpmyadmin
9f099298463f   ghost:latest                  "docker-entrypoint.s..."  33 hours ago  Exited (128) 29 hours ago
ghost-portainer
8dcaecda6c34   portainer/portainer          "/portainer -H unix:..."  33 hours ago  Up 17 hours  0.0.0.0:9000->9000
pi_portainer_1
root@raspberrypi:/home/pi#

```

FIGURE 4.6 – Listes des conteneurs utilisés

4.2.5 Création de la base de données avec le fichier docker-compose

Pour stocker les données détectées par les capteurs nous devons nous assurer qu'ils sont transférées vers la base de données. Nous avons utilisé un serveur apache2 sur lequel nous avons implémenté PHPMYAdmin avec la base de données MySQL (voir annexe A).

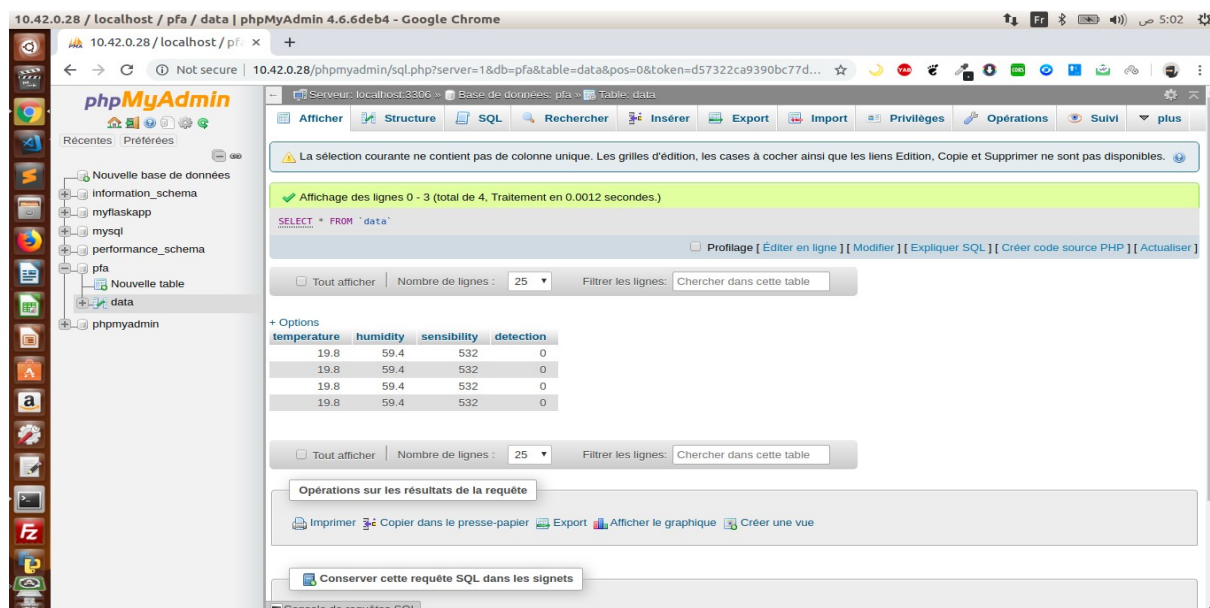


FIGURE 4.7 – Serveur de Base de données

4.2.6 Visualisation des conteneurs

La manipulation des conteneurs avec le terminal Linux semble un peu complexe. Afin de faciliter cette tâche, nous avons choisi d'utiliser un Dashboard qui permet de générer des conteneurs et visualiser leurs performances. Chaque conteneur est défini par son nom, son adresse IP, son image et ses caractéristiques. Les figures 4.8, 4.9 et 4.10 représentent notre Dashboard.

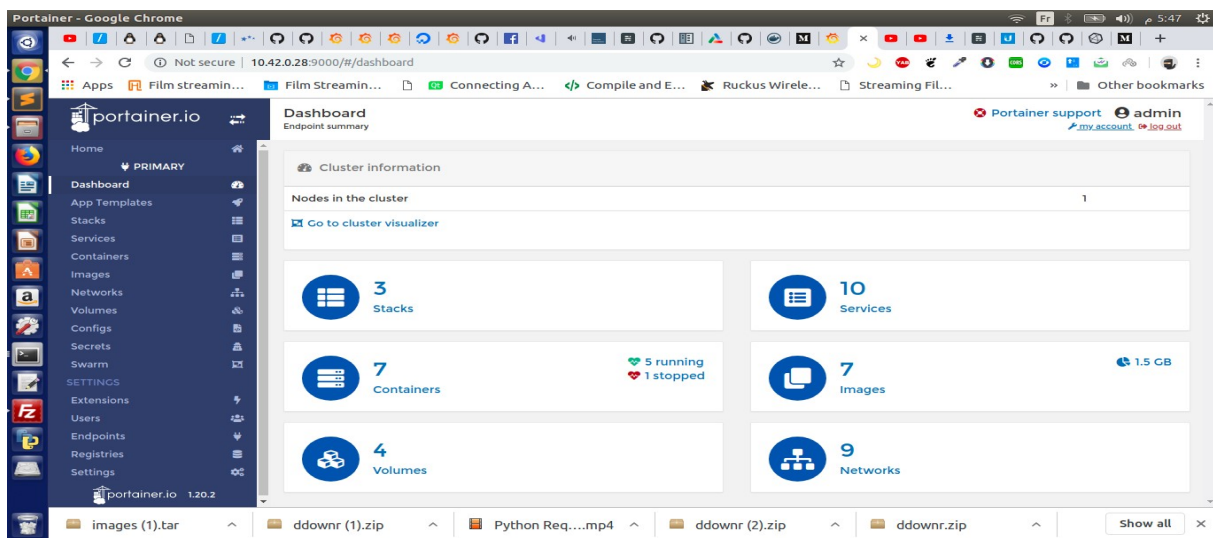


FIGURE 4.8 – Dashboard de conteneurs

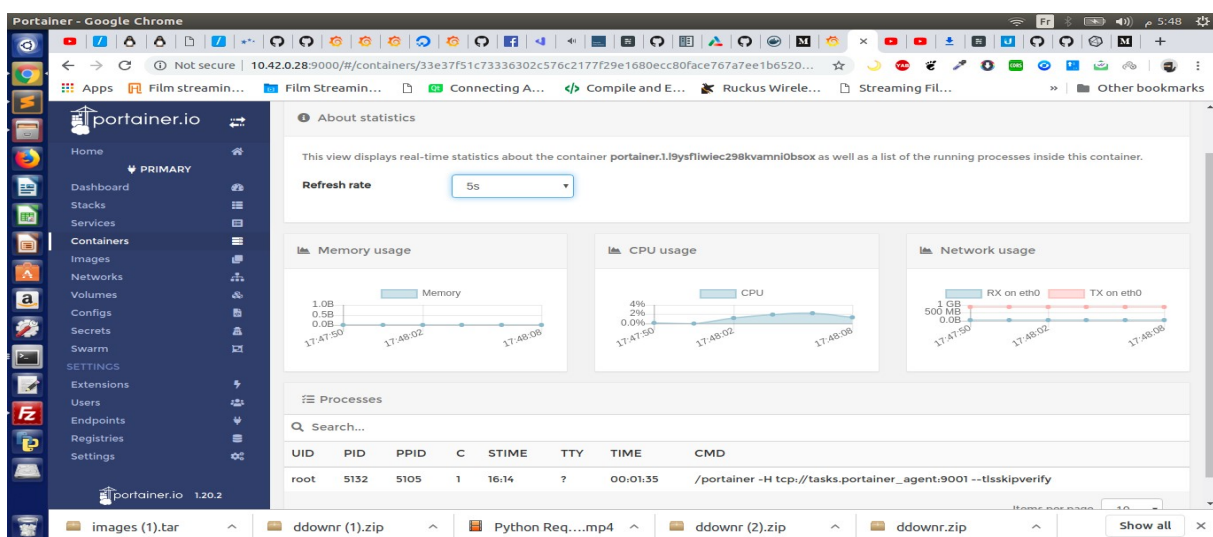


FIGURE 4.9 – Performances des conteneurs

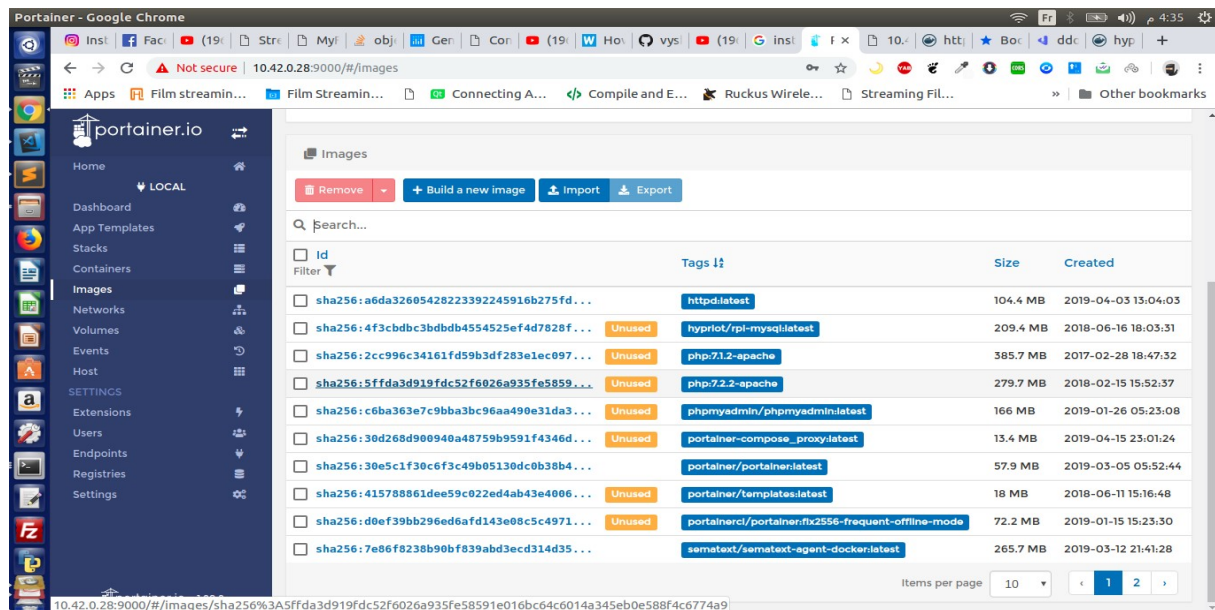


FIGURE 4.10 – Listes des images de conteneurs

4.2.7 Du Fog vers le Dashboard

Afin de visualiser les données détectées et suivre l'état des capteurs en temps réel, nous avons développé un Dashboard (voir Figure 4.11) en utilisant le langage Python, Flask, Bootstrap et Javascript.

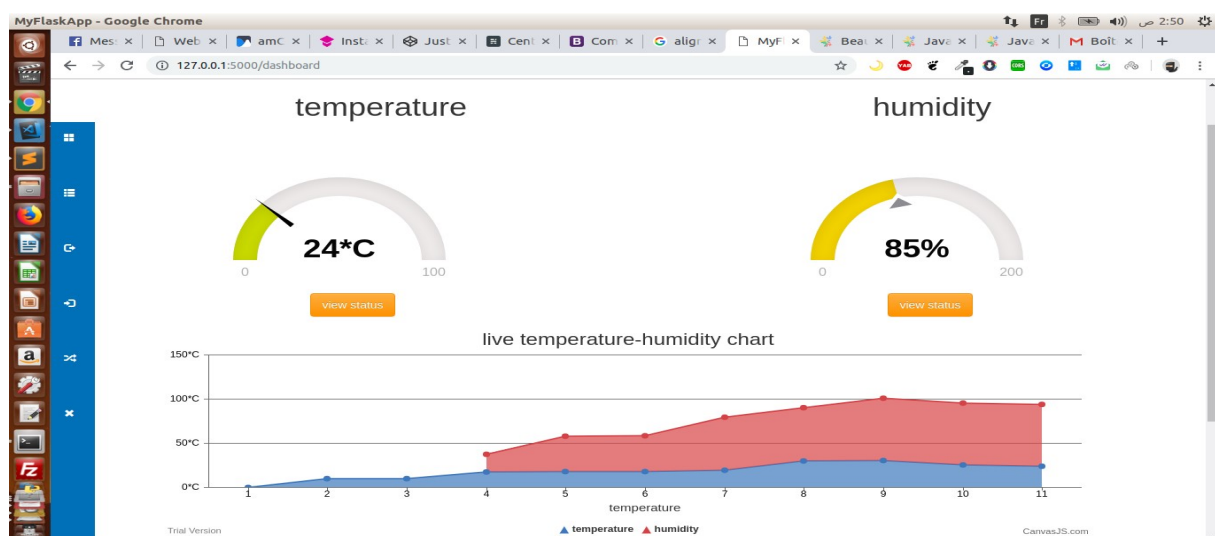


FIGURE 4.11 – Dashboard de visualisation des données

4.2.8 Du Fog vers le Cloud

Pour avoir un contrôle efficace des états des capteurs et pouvoir traiter et gérer le flux d'informations qui y provient, on doit disposer de serveurs ayant une haute puissance de calcul. Dans notre cas, nous nous intéressons à la latence. Pour cette raison, tout le traitement se fait dans le nœud Fog c'est à dire sur le Raspberry. Le résultat final sera déployé dans le cloud. Nous avons choisi d'utiliser la plateforme Ubidots.

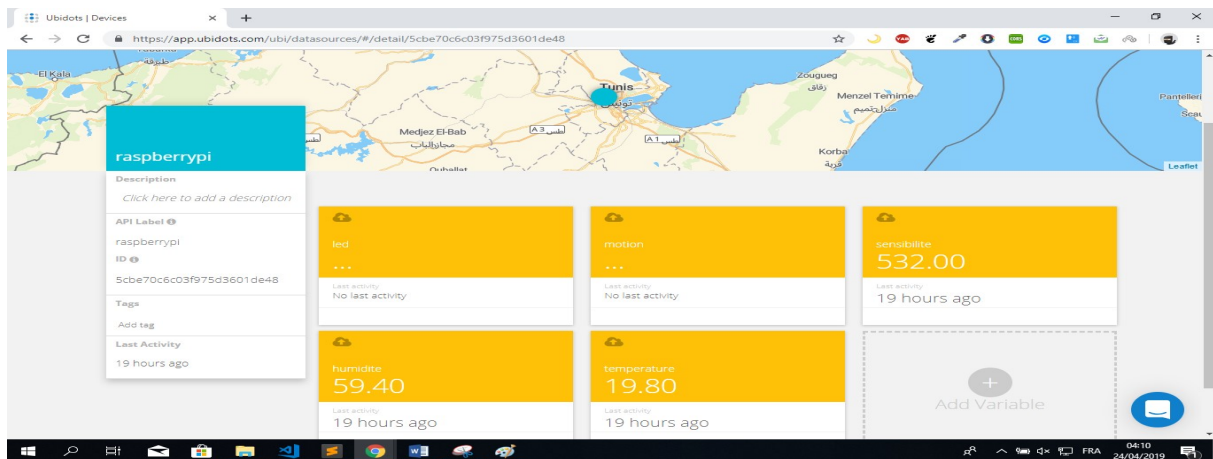


FIGURE 4.12 – Plateforme Ubidots

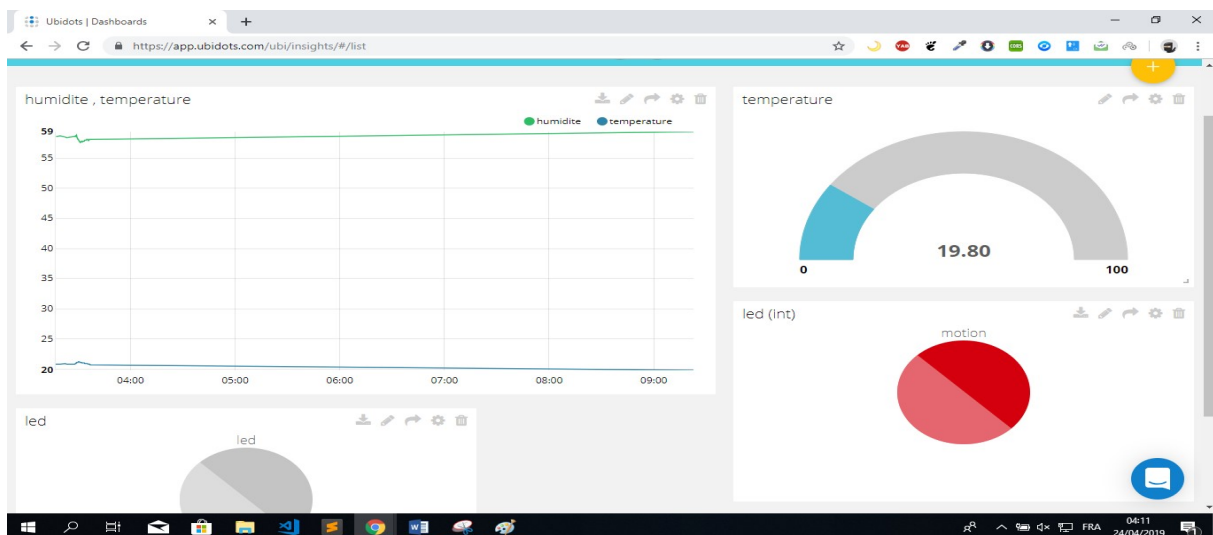


FIGURE 4.13 – Visualisation des Données sur le Cloud

4.3 Constatations & Perspectives

Malgré les contraintes de temps et les difficultés techniques que nous avons rencontrées, notre application peut être considérée efficace et remplit les objectifs que nous avons fixés. Les principaux problèmes que nous avons rencontrés lors de l'implémentation de notre solution se résument dans le mapping des ports et la difficulté d'installer Docker Compose sur le Raspberry Pi qui n'est pas facile à configurer et nécessite la présence d'un ensemble de packages devant être préalablement installés.

Le Dashboard doit permettre à l'utilisateur de visualiser les données en temps réel et contrôler l'état des capteurs ce qui rend son développement un peu complexe.

Enfin, l'utilisation de la carte Raspberry Pi nous a permis de mettre en place une architecture Fog Computing réelle. Mais elle présente certaines faiblesses parmi lesquelles ses capacités limitées.

Plusieurs perspectives s'offrent à ce projet. En utilisant plusieurs Raspberry nous pourrions créer un Cluster qui va nous permettre de détecter et traiter des données à partir d'un réseau de capteurs. Nous pourrions également développer un Dashboard d'administration qui permet à un agent administrateur de contrôler et gérer l'état des capteurs à distance. Enfin, nous pourrions utiliser le Cloud pour effectuer des traitements plus avancés.

Conclusion

Dans ce chapitre nous avons expliqué le fonctionnement de notre système et présenté les difficultés techniques que nous avons rencontrées.

Conclusion Générale

Le présent document est une présentation du travail réalisé dans le cadre du projet de fin d'année qui a pour objectif de déployer un service Fog à base de conteneurs.

Ce projet s'est révélé très enrichissant et instructif. En effet, il nous a permis d'appliquer nos connaissances en informatique et en télécommunications et de développer notre expérience de travail en équipe.

Dans ce rapport, nous avons commencé par présenter le cadre général de notre projet dans le premier chapitre ainsi que nos objectifs et la démarche que nous avons suivie pour réaliser ce travail. Dans le deuxième chapitre, nous avons expliqué les différents concepts et notions sur lesquels se base notre travail. Nous avons ensuite détaillé le scénario de notre application et justifié nos choix technologiques dans le troisième chapitre. Enfin, nous avons présenté une description détaillée des différentes étapes d'implémentation de notre solution.

Récemment, la perspective des environnements IoT intelligents a considérablement évolué. Ce type d'applications se caractérise par une qualité de service volatile en raison de conditions changeantes telles que la charge de travail et la mobilité des périphériques IoT. Le Fog Computing est un paradigme prometteur permettant de traiter de tels scénarios difficiles.

Bibliographie

- [1] Keyur K Patel, Sunil M Patel, "Internet of Things-IOT :Definition, Characteristics,Architecture,Enabling Technologies,Application and Future Challenges", International Journal of Engineering Science and Computing, pages 6122-6131, May 2016.
- [2] [https : //static - course - assets.s3.amazonaws.com/IoE11/FR/index.html](https://static-courses-assets.s3.amazonaws.com/IoE11/FR/index.html), consulté le 10/04/2019.
- [3] Ramaiah Challa, K. Kiran Kumar, "A brief exploration of fog computing architecture,technologies and challenges", International Journal of Engineering & Technology, pages 96-98, 2018
- [4] [https ://www.docker.com/resources/what-container](https://www.docker.com/resources/what-container), consulté le 01/03/2019
- [5] Rajdeep Dua, A Reddy Raja, Dharmesh Kakadia, "Virtualization vs Containerization to support PaaS", IEEE International Conference on Cloud Engineering, 2014
- [6] [https ://www.redhat.com/fr/topics/containers/what-is-docker](https://www.redhat.com/fr/topics/containers/what-is-docker), consulté le 01/03/2019
- [7] [https ://docs.docker.com/engine/docker-overview/](https://docs.docker.com/engine/docker-overview/), consulté le 01/03/2019
- [8] Vladimir Vujović, Mirjana Maksimović, "Raspberry Pi as a Wireless Sensor Node : Performances and Constraints", MIPRO 2014, pages 26-30, May 2014
- [9] [https ://docs.docker.com/compose/overview/](https://docs.docker.com/compose/overview/), consulté le 18/04/2019
- [10] [http ://flask.pocoo.org/docs/1.0/](http://flask.pocoo.org/docs/1.0/), consulté le 18/04/2019

Annexe A

```
version: "3"

services:
  apache:
    image: tobi312/rpi-apache2
    container_name: 'apache'
    restart: 'always'
    ports:
      - "90:90"
      - "443:443"
    volumes:
      - ./www:/var/www/html
      - ./php/php.ini:/usr/local/etc/php/php.ini
      - ./sites-enabled:/etc/apache2/sites-enabled
      - apache-logs:/var/log/apache2
  mysql:
    image: hypriot/rpi-mysql
    container_name: 'mysql'
    restart: 'always'
    volumes:
      - mysql-data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: pewdipie
      MYSQL_DATABASE: pfa
      MYSQL_USER: user
      MYSQL_PASSWORD: pewdipie
  phpmyadmin:
    image: phpmyadmin/phpmyadmin:4.8
    container_name: 'phpmyadmin'
    environment:
      PMA_HOST: mysql
      PMA_PORT: 3306
    ports:
      - '8080:90'
volumes:
  apache-logs:
  mysql-data:
```

```
version: '3'

services:
  cadvisor:
    image: budry/cadvisor-arm
    volumes:
      - /:/rootfs:ro
      - /var/run:/var/run:rw
      - /sys:/sys
      - /var/lib/docker:/var/lib/docker:ro
      - /dev/disk:/dev/disk:ro
    ports:
      - 9090:9090

    services:
  apache:
    image: tobi312/rpi-apache2
    container_name: 'apache'
    command: -H unix:///var/run/docker.sock
    restart: 'always'
    ports:
      - "9090:9090"
    volumes:
      - ./www:/var/www/html
      - ./php/php.ini:/usr/local/etc/php/php.ini
      - ./sites-enabled:/etc/apache2/sites-enabled
      - apache-logs:/var/log/apache2

volumes:
  apache-logs:
```

Annexe B

```
3 //temp
4 #include "DHT.h"
5 #define PIR_MOTION 2
6 #define DHTPIN A0
7 #define DHTTYPE DHT22
8 ///lcd
9 #include <TM1637Display.h>
10 #include<ArduinoJson.h>
11 //int sensor = 2;           // the pin that the sensor is attached to
12 int state = LOW;           // by default, no motion detected
13 int val = 0;
14 int analogValue = 0;
15 int lightSensorPin = A1;
16
17 int ledred=6;
18 int ledgreen=5;
19
20
21 const int buzzer = 10;
22 DHT dht(DHTPIN, DHTTYPE);
23
24 const int CLK = 9; //Set the CLK pin connection to the display
25 const int DIO = 8;
26 TM1637Display display(CLK, DIO);
27
28 void setup() {
29     // put your setup code here, to run once:
30     Serial.begin(115200);
31     display.setBrightness(0x0a);
32     pinMode(PIR_MOTION, INPUT);
33     pinMode(buzzer, OUTPUT);
34     pinMode(ledgreen,OUTPUT);
35     pinMode(ledred,OUTPUT);
36
37
38 }
```

```
39 void loop() {
40     float hum = dht.readHumidity();
41     float temp = dht.readTemperature();
42     val = digitalRead(PIR_MOTION);
43     monitor();
44
45     if(Serial.available()>0){
46         String a=Serial.readString();
47
48         if(a=="2"){
49             if (ispeople()==true && analogValue > 300){
50
51                 Serial.print("someone is present :");
52                 Serial.println("sensibility is over 300 : light is down");
53                 digitalWrite(ledgreen, LOW);
54                 //delay(2000);
55             }
56         else if( ispeople()== true && analogValue <200){
57             Serial.print("someone is present :");
58             Serial.println("sensibility is below 200 :led on");
59             digitalWrite(ledgreen, HIGH);
60             delay(2000);
61             digitalWrite(ledgreen, LOW);
62
63
64         }
65         else if (ispeople()== false && analogValue <200){
66             Serial.print("no one is present ");
67             Serial.println("light is always down ");
68             digitalWrite(ledgreen, LOW);
69             delay(1000);
70
71         }
72         else {if(ispeople()== false && analogValue >300)
73         { Serial.println("lights down");
74             digitalWrite(ledgreen, LOW);
75             delay(1000);
76         }
```

```
80 else {
81   if(a=="1"){
82     if (temp > 30)
83     {Serial.print("temperature is higher than 30 *C " );
84       Serial.println("air conditionning is activated");
85       digitalWrite(ledred, HIGH);
86       tone(buzzer, 400,1000);
87       delay(1000);
88     }
89   }
90   else {if (temp < 30)
91     {
92       digitalWrite(ledred, LOW);
93       noTone(buzzer);
94       delay(1000);
95     } }
96   }
97 }
98 //else {Serial.println("error");}
99 }
100 }
101
102
103 void monitor(){
104   float hum1 = dht.readHumidity();
105   float temp1 = dht.readTemperature();
106   analogValue = analogRead(lightSensorPin);
107   Serial.print(temp1,1);
108   Serial.print("\t");
109   Serial.print(hum1,1);
110   Serial.print("\t");
111   Serial.print(analogValue);
112   Serial.println("\t");
113
114 }
```



```
115
116 boolean ispeople(){
117
118     int state=digitalRead(PIR_MOTION);
119     if(state == HIGH) return true;
120     else return false;
121
122 }
123
```