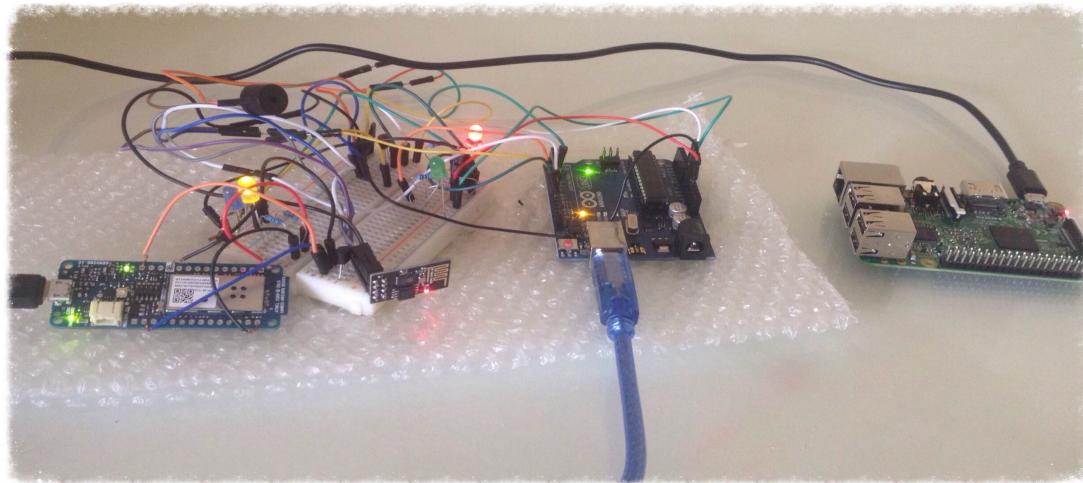


Chapitre 2

Mise en place de l'environnement



Introduction

Ce chapitre présentera la mise en œuvre de notre environnement internet des objets. Nous verrons en premier lieu les matériels et les logiciels utilisés dont la plateforme Cloud auquelle nous avons eu recours. Puis nous verrons le modèle architectural opté et son apport; ensuite nous présenterons le fonctionnement de nos objets connectés et de tout l'environnement en général pour finir par le nœud central et ses fonctionnalités.

2.1 Le matériel utilisé

Pour mettre en place notre environnement « Internet des objets », nous avons eu à notre disposition le hardware que nous allons présenter dans cette section.

2.1.1 Présentation du matériel

Afin de mettre en place notre environnement IoT, nous avons opté pour le matériel suivant:

Les microcontrôleurs :

Un microcontrôleur est un circuit intégré qui rassemble les éléments essentiels d'un ordinateur : processeur, mémoires (mémoire morte pour le programme, mémoire vive pour les données), unités périphériques et interfaces d'entrées-sorties. Les microcontrôleurs se caractérisent par un plus haut degré d'intégration, une plus faible consommation électrique, une vitesse de fonctionnement plus faible et un coût réduit par rapport aux microprocesseurs polyvalents utilisés dans les ordinateurs personnels.



Figure 2.1: Cartes Arduinos utilisées

Caractéristiques techniques des cartes Arduino [6] [7]

	Arduino UNO	Arduino MKR1000
Contrôleur	ATmega328	SAMD21 Cortex-M0+
Tension d'alimentation	7-12V	5V
Entrées/Sorties numériques	14	8
Entrées/Sorties analogiques	6 entrées	7 entrées - 1 sortie
Mémoire Flash	32 KB	256 KB
SRAM	2 KB	32 KB
Mémoire EEPROM	1 KB	Pas d'EEPROM
Fréquence d'horloge	16 MHz	48 MHz
Wi-Fi	Pas de wifi intégré	IEEE 802.11 b/g/n

Table 2.1: Caractéristiques des cartes Arduinos utilisées

La SRAM est la mémoire vive du composant. l'EEPROM est une mémoire morte et est capable de stocker des informations même lorsque la carte n'est plus alimentée. La mémoire Flash sert à stocker le programme qui est téléchargé dans le microcontrôleur.

Un module WiFi ESP8266

Figure 2.2: ESP8266-01

Vu que la carte Arduino Uno ne contient pas un module wi-fi intégré et que nous aurons besoin de la connecter à internet, nous allons la brancher avec le module WiFi ESP8266.

Des capteurs de température

Figure 2.3: LM35

Pour détecter la température des environs où seront placés nos objets connectés, nous allons utiliser des capteurs de température LM35.

Des actionneurs

Comme actionneurs nous avons opté pour des LEDs (light-Emitting Diode) qui vont s'allumer suivant des règles que nous allons définir (une verte et une rouge) et une alarme TMB12A05.



Figure 2.4: Actionneurs utilisés

Un Raspberry Pi 3

Le Raspberry Pi est un nano-ordinateur mono-carte qui permet l'exécution de plusieurs variantes du système d'exploitation libre GNU/Linux et des logiciels compatibles. Il est fourni nu (carte mère seule, sans boîtier, alimentation, clavier, souris ni écran) dans l'objectif de diminuer les coûts et de permettre l'utilisation de matériel de récupération.

Le Raspberry Pi 3 contient entre autres un processeur 1.2GHz 64-bit quad-core ARMv8, un module wifi intégré 802.11n, le Bluetooth 4.1 et un Bluetooth de faible énergie (BLE Bluetooth Low Energy), 1 GB de RAM et 4 ports USB. Le système d'exploitation utilisé dans notre projet (et que nous devons installer/configurer avant de commencer notre projet [8]) est le Raspbian.

C'est dans la Raspberry que nous allons installer notre serveur local dont nous évoquerons la nature et les fonctions par la suite.



Figure 2.5: Raspberry Pi 3

Un Régulateur de tension

Pour relier le module Wifi ESP8266 à l'Arduino Uno, il est nécessaire d'utiliser un régulateur de tension qui va transformer la tension d'alimentation (5V) à 3,3V. Un régulateur de tension est un élément qui permet de stabiliser une tension à une valeur fixe, et qui est nécessaire pour les montages électroniques qui ont besoin d'une tension qui ne fluctue pas, ne serait-ce que peu.



Figure 2.6: AMS1117

Ci-dessous un tableau qui compare les nœuds étudiés et un iphone 6 afin d'illustrer leurs capacités et performances.

	Arduino UNO	Arduino MKR1000	Raspberry Pi	iPhone 6
RAM	2KB	32KB	1GB	1GB
CPU	ATmega328P 16 MHz architecture 8-bit	SAMD21 Cortex-M0+ 48MHz architecture 32-bit	Quad-core ARM cortex-A53 1.2 GHz architecture 64-bit	Dual-core ARM v8-A 1.4 GHz architecture 64-bit

Table 2.2: Comparaison UNO, MKR1000, RPi et iphone6

2.1.2 Montage des objets

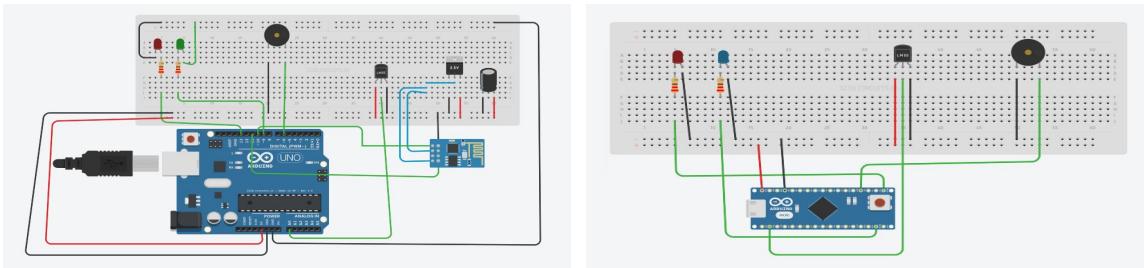


Figure 2.7: Objet basé sur l'Arduino UNO (gauche) et sur MKR1000 (droite)

Pour ces prototypes, les différents composants sont reliés à travers une platine d'expérimentation (les ports dans les extrémités supérieures et inférieures ont le même voltage horizontalement et dans les parties au milieu de la platine, les ports ont le même voltage verticalement).

Des résistances de 220Ω ont été utilisées pour éviter le survoltage des lampes (LED) pour qu'elles ne se grillent pas et un condensateur de $10\mu F$ dans le premier circuit pour stabiliser les tensions et les courants circulants dans tout le circuit.

2.2 Le logiciel utilisé

Pour mettre en place notre environnement « Internet des objets », nous avons eu recours aux éléments logiciels suivants :

L'IDE Arduino

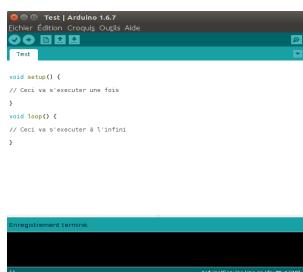


Figure 2.8: IDE d'Arduino

C'est l'environnement de programmation qui va nous permettre d'écrire les codes (voir ANNEXES 1 et 2) que nous allons téléverser dans les cartes Arduino.

Dans la section *void setup()* les opérations sont exécutées une seule fois, par contre dans la section *void loop()* les opérations sont répétées infiniment, tant que l'objet est branché à une source d'alimentation.

Dans l'onglet *Outils* puis *type de carte*, on choisit l'Arduino UNO ou l'Arduino MKR1000 selon le besoin.

La plateforme ThingSpeak

ThingSpeak est une plateforme open-source d'internet des objets qui permet de collecter des données de capteurs dans le Cloud et de développer des applications IoT. Elle va nous permettre de visualiser l'évolution de nos données et contient plusieurs applications qui permettent de réagir lorsque le comportement de nos données atteint une certaine condition, comme par exemple envoyer un e-mail ou un tweet pour alerter le concerné lorsque la température dépasse un certain seuil.

Dans la figure 2.9, on observe une courbe représentant les valeurs de température reçues. Pour ce faire, et après s'être inscrit sur la plateforme, nous créons un canal pour la communication entre le serveur

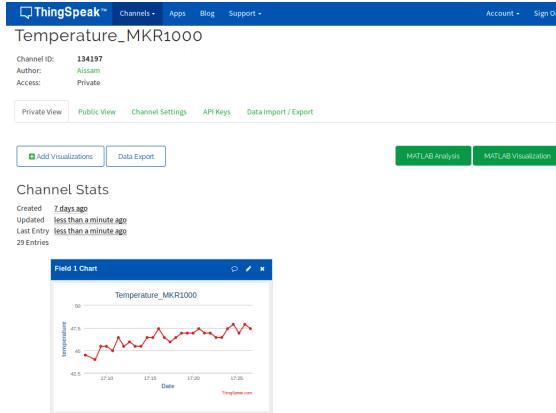


Figure 2.9: Visualisation sur ThingSpeak

de Thingspeak et notre objet IoT. Une clé nous est offerte (API Key) pour l'inclure dans notre code (voir ANNEXES 1 & 2) pour qu'on puisse transférer nos données de l'objet IoT à travers ce canal.

Serveur Web Apache

Nos communications se font avec le protocole HTTP. Sur notre nœud central Raspberry Pi, utilisé comme serveur local, nous y avons installé Apache 2 comme serveur web. Nous détaillerons le rôle de ce serveur par la suite, dans la partie 2.5.

Twilio SMS

www.twilio.com est une plateforme qui offre des APIs pour des appels vocaux, envois de messages SMS et bien d'autres fonctionnalités. Nous l'avons utilisé dans notre environnement pour l'envoi d'SMS. Ci-dessous un exemple de message reçus.



Figure 2.10: Message Twilio

2.3 Architecture hybride

Pour assurer une haute disponibilité de notre service, nous avons opté pour architecture hybride comme illustré dans la figure 2.11. Les objets IoT basés sur les Arduino envoient les données de température au Cloud ThingSpeak mais aussi et surtout envoie ces données au nœud central Raspberry Pi.

Comme Thingspeak est une plateforme en ligne qui n'est pas sous notre contrôle, si jamais il y a un dysfonctionnement de la plateforme nous aurons tout de même les données stockées et gérées en local.

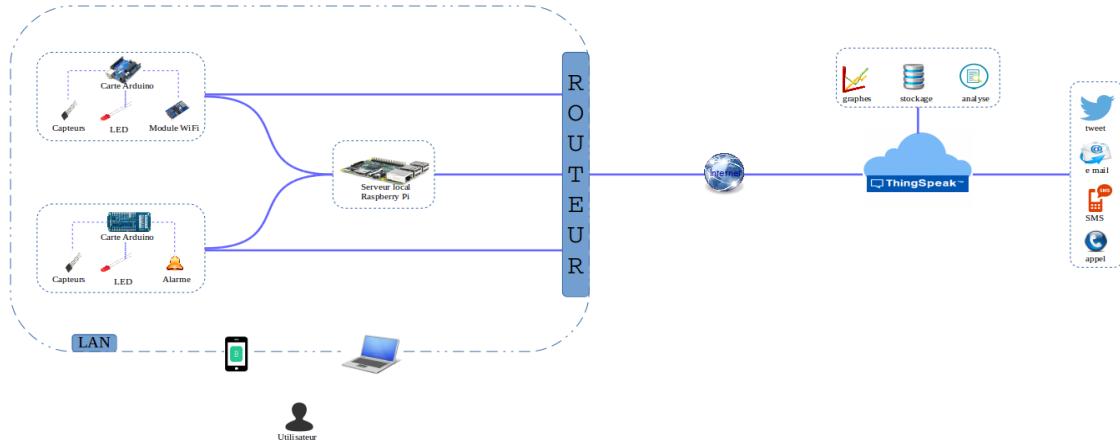


Figure 2.11: Architecture de notre environnement

Comme application web, on considère le site/plateforme ThingSpeak où on visualise le comportement des données de température. Nous pouvons utiliser pas mal d'applications mobile qui peuvent contrôler des Arduinos (ex. Blynk), ou encore l'application ThingView sur android pour visualiser les données sur Thingspeak.

2.4 Algorithme tournant sur les objets

Dans cette section, nous présentons le fonctionnement des objets, et de tout l'environnement en général, dû au téléchargement des codes Arduino (voir annexes).

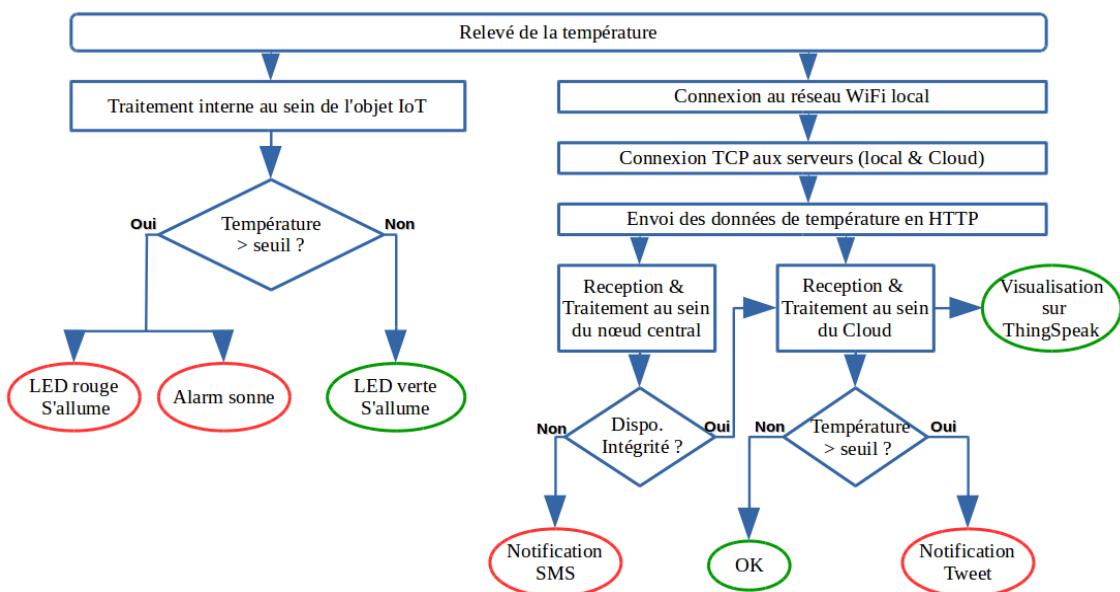


Figure 2.12: Organigramme de l'algorithme de l'environnement IoT

On peut identifier trois sections dans notre système où il y a des traitements et des actions:

- Au niveau des objets: il y a un traitement interne pour déterminer si la valeur de la température dépasse le seuil fixé; et si c'est le cas l'alarme sonne et la lampe rouge s'allume.
- Au niveau du nœud central: où on collecte les données, il y figure aussi des fonctions de vérification de disponibilité, si les données sont toujours envoyées (voir ANNEXE 7,) et d'intégrité, si les valeurs sont crédibles (voir ANNEXE 6). Si ce n'est pas le cas, on lance l'envoi de messages SMS vers l'utilisateur pour l'informer.
- Au niveau du Cloud: sur ThingSpeak nous avons configuré aussi l'action d'un envoi de Tweets vers le compte dédié à notre environnement IoT, le responsable sera mentionné dans le Tweet pour qu'il reçoive une notification l'informant que la température dépasse le seuil fixé. La figure qui suit montre un exemple : `@aissam_out` est mon compte Twitter et `iot_account` est le compte que nous avons précisé à ThingSpeak (ThingTweet) dans <https://thingspeak.com/apps> pour qu'il soit utilisé comme compte expéditeur des tweets.



Figure 2.13: Notification par Tweet

On me notifie qu'une haute température a été détectée !

2.5 Le nœud central

Comme nous l'avons déjà cité, dans la Raspberry -le noeud le plus performant- nous avons installé Apache 2 qui va nous servir comme serveur local. Ce serveur aura pour fonction de stocker les données de température en local mais aussi de faire des traitements à savoir informer sur l'intégrité des données ou vérifier si les objets connectés envoient encore des données: qu'il n'y a pas des périodes d'indisponibilité.

Nous allons placer le dossier nommé `iot` qui contiendra les fichiers qui exécuteront ces différentes tâches dans le dossier racine de Apache : `/var/www/html`.

```
pi@raspberrypi:/var/www/html/iot
pi@raspberrypi:/var/www/html $ ls
index.php  iot
pi@raspberrypi:/var/www/html $ cd iot
pi@raspberrypi:/var/www/html/iot $ ls
code.php  code.php.save  code2.php  data.txt  data2.txt  things.py
pi@raspberrypi:/var/www/html/iot $
```

Figure 2.14: Dossier `/var/www/html/iot`

`code.php / code2.php`

`code.php` et `code2.php` sont deux fichiers en langage `php` (voir ANNEXE 4) qui se chargent de recevoir les données de température envoyées respectivement par les objets basés sur l'Arduino UNO et sur l'Arduino MKR1000 et de les stocker dans les fichiers textes `data.txt` et `data2.txt`.

`things.py`

C'est un fichier écrit en langage `python` (voir ANNEXE 5) qui fait de multiples tâches très importantes. Il récupère les dernières données stockées dans les fichiers `data.txt` et `data2.txt`, calcule la moyenne entre

les deux valeurs et envoie le résultat à ThingSpeak. Donc sur le Cloud de ce dernier, on peut lire les données transmises par les objets et aussi la valeur envoyée par noeud central. Ainsi d'après la lecture des trois courbes, on peut avoir une idée sur la température moyenne qui règne dans la salle où sont placés les objets.

Le fichier *things.py* doit être lancé au démarrage du Raspberry Pi. Pour ce faire, voici ce qu'il fallait faire :

- Rendre le script exécutable: `sudo chmod +x /home/pi/things.py`
- Editer le fichier */etc/rc.local* : `sudo nano /etc/rc.local`
- Ajouter avant *exit 0* : `python /home/pi/things.py`

Le fichier *things.py* fait aussi appel à deux autres fichiers python *msg.py* et *dispo.py*.

msg.py

Ce fichier (voir ANNEXE 6) a pour fonction d'envoyer des messages d'alertes à travers le site www.twilio.com. Nous avons fait appel aux bibliothèques appropriées pour utiliser l'API de twilio afin d'envoyer des SMS au responsable chargé de contrôler la température pour lui indiquer si la température envoyée par chacun des objets a dépassé le seuil (ici fixée à 60) et si la différence entre les deux températures dépasse une certaine valeur(ici 5), car cela indique un problème d'intégrité: l'un des capteurs donne une valeur très élevée par rapport à l'autre, c'est possible qu'elle soit erronée.

dispo.py

Ce fichier (voir ANNEXE 7) a pour rôle de vérifier la disponibilité du service. Il doit envoyer des SMS pour signaler s'il y a un des Arduinos qui n'envoie aucune donnée. Pour ce faire, il lit le nombre de lignes de chacun des fichiers *data.txt* et *data2.txt*; si après une ou plusieurs exécutions du fichier *things.py* le nombre de lignes de l'un des fichiers reste constant, cela signifie que l'Arduino correspondant n'envoie plus de données.

Conclusion

Dans ce chapitre, nous avons illustré l'élaboration de notre système internet des objets. Nous avons présenté le matériel et le logiciel utilisé, le montage des circuits constituant les objets, la plateforme Cloud ThingSpeak, le noeud central et ses différents rôles. Dans le chapitre qui suit, nous allons mener une analyse des risques détaillée sur notre environnement IoT en se basant sur la démarche de la méthode EBIOS.