

Maxime MARTELLI  
Lounes ACHAB  
Matthieu BOULANGER  
Afshin KHALGHDOOST

# Internet Of Things



## ENCADRANTS

Cécile Braunstein  
François Pécheux  
Yann Douze

# Table des matières

<b>I. Présentation générale du projet.....</b>	<b>3</b>
A. Description du projet.....	3
B. Récupération des sources (Github).....	4
C. Utilisation client final .....	5
1. <i>MBED</i> .....	5
2. <i>Raspberry Pi</i> .....	5
3. <i>Site web</i> .....	5
4. <i>Sms</i> .....	7
<b>II. MBED .....</b>	<b>8</b>
A. Présentation de l'environnement.....	8
B. Montage électronique.....	9
1. <i>Microcontrôleur : Seeeduuino-Arch-Pro</i> .....	9
2. <i>Capteurs</i> .....	9
C. Configuration BLE.....	12
D. Architecture logicielle.....	13
E. Explications code.....	14
1. <i>Bibliothèques des composants</i> .....	14
2. <i>Fichier de configuration :</i> .....	15
3. <i>Code Principal</i> .....	16
<b>III. Raspberry Pi 2.....</b>	<b>23</b>
A. Présentation de l'environnement.....	23
B. Configuration du système.....	24
1. <i>BLE</i> .....	24
2. <i>Node</i> .....	24
3. <i>Npm</i> .....	24
4. <i>Lancement du programme</i> .....	24
C. Explications code.....	25
<b>IV. Bluemix.....</b>	<b>28</b>
A. Présentation de l'environnement.....	28
B. Internet of things.....	34
1. <i>Création d'un terminal et configuration du compte</i> .....	34
2. <i>Principe du service IoT Foundation</i> .....	35
C. Base de données DB2 .....	38
1. <i>Ouvrir la base de données</i> .....	41
2. <i>Modifier la base de données</i> .....	41
3. <i>Fermer la base de données</i> .....	42
D. Service sms Twilio .....	42
<b>V. Bibliographie.....</b>	<b>45</b>

# I. Présentation générale du projet

## A. Description du projet

Ce projet a pour but de visualiser sur un terminal, les mesures en temps réel des différents capteurs déployés.

Le projet peut être divisé en 4 parties :

- **Environnement matériel :** nous nous situons à l'amont de la chaîne globale. Elle contient tous les capteurs disponibles (capteur de température, luminosité, pression, etc.), la carte ARM et le BLE (Bluetooth Low-Energy). Les capteurs, qui sont reliés à la carte ARM, sont contrôlés et sollicités de façon périodique par le microcontrôleur. Toutes les valeurs mesurées par les capteurs seront enregistrées dans une variable globale. Par la suite, le BLE va lire dans cette variable globale puis émettre le résultat de la lecture vers le deuxième étage du projet : la Raspberry Pi.
- **Raspberry Pi 2 :** cette plateforme permet de créer un serveur pour pouvoir envoyer des données vers l'Internet. Elle va, via un petit module Bluetooth, réceptionner toutes les données envoyées par le BLE. Puis, après traitement des données, elle va envoyer toutes les données reçues via MQTT vers le service Bluemix dans le cloud.
- **Environnement Bluemix :** celui-ci permet de déployer des applications dans le cloud sans avoir à se soucier de la mémoire, de l'espace disque, de l'installation du serveur et de sa configuration. Il comprend deux applications et deux services qui sont Twilio et SQL Database. L'API SMS de Twilio permet de gérer les sms entre les téléphones et l'application. Ainsi, ce service est utilisé pour pouvoir afficher les dernières valeurs mesurées par les capteurs mais aussi de contrôler, par des mots-clés, la base de données. Le deuxième service, SQL Database, permet de stocker toutes les valeurs prises par les capteurs. Cette base est très utile pour l'utilisateur qui souhaite obtenir les dernières valeurs sur son smartphone. Il pourra également récupérer toutes les valeurs pour y effectuer différents traitements (moyenne, statistiques, etc.). La première application permet de gérer Twilio et la base de données, alors que la seconde application permet de récupérer les données envoyées en live par les différents terminaux, et d'afficher les données historiques stockées par le service IoT foundation.
- **Interface client :** cette dernière fournit deux possibilités pour interagir avec notre système.
  - **Si l'utilisateur dispose d'une connexion Internet :**  
Le lien suivant <http://visualisationappmax.eu-gb.mybluemix.net/> permet de visualiser les valeurs envoyées par n'importe quel terminal lié à notre organisation, en temps réel, ou en récupérant l'historique des valeurs. Ces valeurs sont directement intégrées dans un graphe qui peut prendre plusieurs formes. Cette fonctionnalité est très intéressante si l'utilisateur souhaite visualiser les tendances des différentes mesures sur de longues périodes de temps.
  - **Si l'utilisateur ne dispose pas d'une connexion Internet :**  
Il peut envoyer un sms au numéro suivant +33 6 44 60 71 38 des mots-clés spécifiques pour pouvoir par exemple interroger la base de données et retrouver les dernières valeurs mesurées par les capteurs. Plusieurs fonctionnalités ont été implémentées comme des systèmes d'alarme, des systèmes de contrôle de base de données, etc. Toutes ces fonctionnalités vous seront présentées dans la partie adéquat de ce tutoriel.

Voici donc le synoptique général de notre projet :

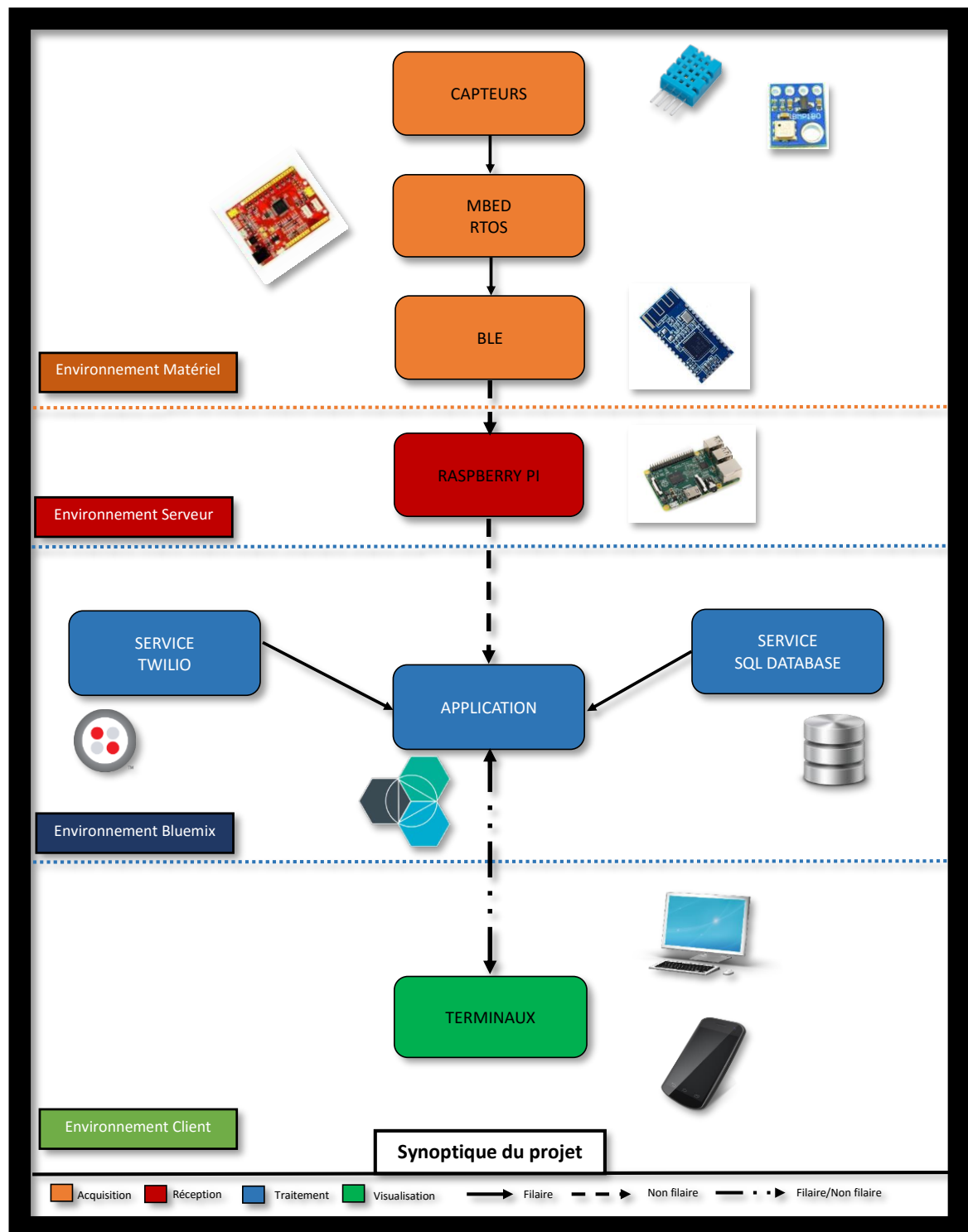


Figure 1: Synoptique général

## B. Récupération des sources (Github)

Il suffit donc de récupérer tous les fichiers présents à l'adresse GIT suivante :

<https://github.com/ALounes/E-Home>

## C. Utilisation client final

Cette partie ne parle pas du paramétrage complet de chaque plateforme, mais plutôt de l'utilisation par un client final.

### 1. MBED

Pour la MBED, il suffit juste de la brancher au secteur, et l'émetteur BLE enverra les données sous forme de notifications.

Le code est fourni à cette adresse suivante :

<https://developer.mbed.org/users/ALounes/code/IoT-Polytech-Upmc/>

### 2. Raspberry Pi

How-to :

1. Mettre le dossier « IOT\_Raspberry » sur le bureau
2. Ouvrir un terminal
3. Taper `cd ~`
4. Taper `cd Desktop/IOT_Raspberry`
5. Lancer le script `node.js` à l'aide de la commande « `sudo node ble.js` »

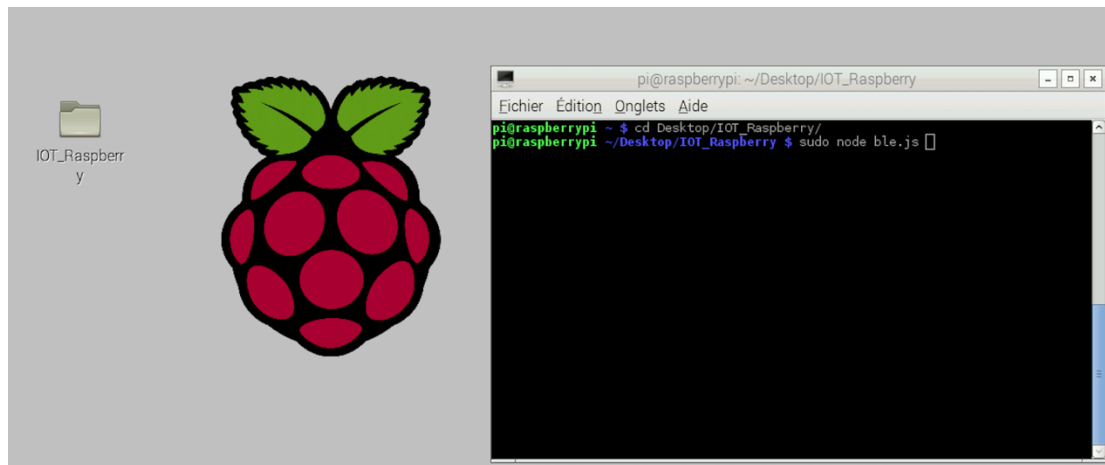


Figure 2: Lancement du script sur la Raspberry Pi

### 3. Site web

Lien : <http://visualisationappmax.eu-gb.mybluemix.net>

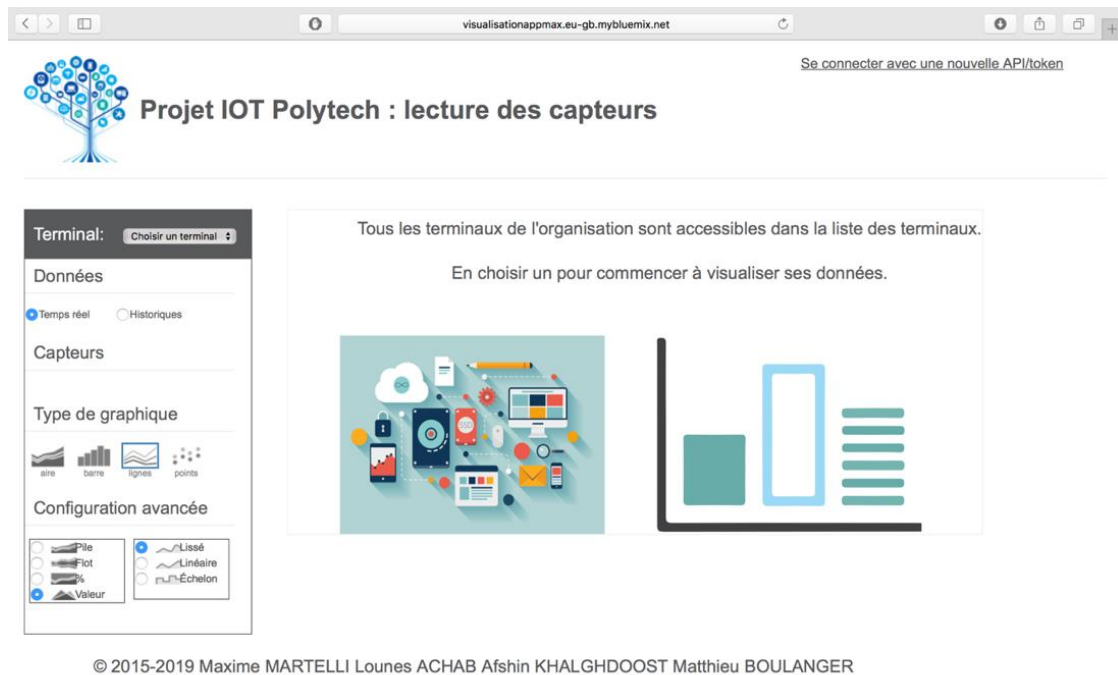


Figure 3: Page d'accueil du site de visualisation du projet

Ce site a été conçu afin de monitorer les données envoyées par une flotte de périphérique. Son utilisation est relativement simple.

#### How-to :

1. Rentrer sa clef API et son token en cliquant sur le lien en haut à droite
2. Choisir le terminal dans la liste
3. Sélectionner si l'on veut des données en temps réel ou consulter l'historique des valeurs
4. Finalement, paramétrer l'affichage des différentes courbes

#### Capteurs

- ✓ ☒ Mouvement
- ✓ ☒ Luminosité
- ✓ ☒ Pression
- ✓ ☒ Température2
- ✓ ☒ Humidité
- ✓ ☒ Température1

5. Afin de désactiver ou activer une courbe, cliquer sur sa check box.
6. Pour n'en sélectionner qu'une, cliquer sur le texte correspondant
7. Pour tous les sélectionner, double cliquer sur

#### 4. Sms

Nous avons configuré un serveur twilio. Ce dernier nous permet donc, en envoyant certains messages à un numéro prédéfini, d'interagir avec le système.

Le numéro actuel est le +33 6 44 60 71 38.

##### Commandes SMS :

##### Capteurs :

- Renvoie les dernières valeurs de capteurs

##### Alarme YY XX Z :

- Permet d'activer une alarme suivant la valeur d'un capteur
  - YY peut prendre les valeurs T1, H, P, T2, M, L (correspond à chaque capteur)
  - XX Valeur seuil
  - Z : + ou - . Paramètre le sens de la détection

Ainsi, « Alarme T1 30 + » permet d'activer une alarme si la température dépasse les trente degrés

##### Alarme M :

- Active la détection de mouvement

##### Etat :

- Renvoie les alarmes qui sont actives pour le moment

##### Désactiver YY :

- Désactive l'alarme liée au capteur YY

##### Désactiver tout :

- Désactive toutes les alarmes

##### Purge :

- Vide la base de donnée

## II. MBED

### A. Présentation de l'environnement

Dans cette section nous allons expliquer les différentes étapes de conception et de programmation de la partie « acquisition et transfert de données ». Cette dernière est constituée de 3 composantes : les capteurs, la carte ARM et l'émetteur BLE.

Ce tutoriel explicitera la partie développement sur la plateforme MBED. Les sources étant fournies, l'application est reproductible sous d'autres plateformes de développement (IDE Keil...). Par ailleurs, les sous-sections de ce tutoriel donneront toutes les informations liées au branchement des différents capteurs et émetteurs sur la carte ARM, ainsi que les explications liées à l'architecture logicielle du système.

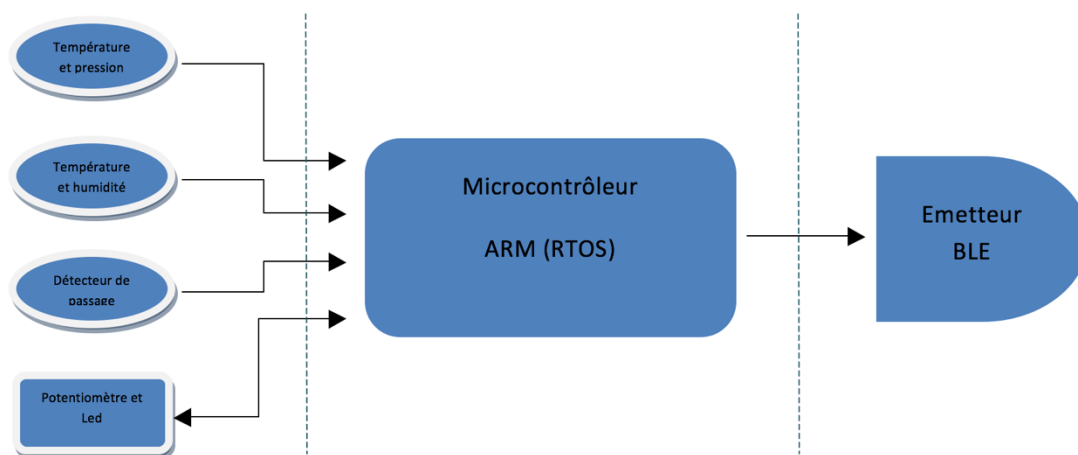


Figure 4: Structure du MBED

Le système est géré avec un RTOS sur la carte ARM. Ce dernier permet de faire du multithreading sur les différents modules électroniques utilisés, puis, avec une période constante et à l'aide du BLE, le système transmet les informations capteurs récoltées représentatif à l'instant « t » de l'état de l'environnement système.

Les informations récoltées sont les suivantes :

- Temperature 01
- Temperature 02
- Pourcentage d'humidité
- Pression atmosphérique
- Luminosité
- Détection de présence et type de mouvement

Par ailleurs, le système offre la possibilité de modifier manuellement la luminosité à l'aide d'un potentiomètre, et de contrôler l'allumage et l'extinction de 7 LED's qui peuvent être remplacées par des actionneurs ou par d'autres modules électroniques.



## B. Montage électronique

Dans cette partie nous allons expliquer quel composant électronique prendre et comment les brancher afin de reproduire notre application.

### Liste des modules électroniques :

#### 1. Microcontrôleur : Seeeduuino-Arch-Pro

Manipulation : alimenter la carte a l'aide d'un câble micro-usb.

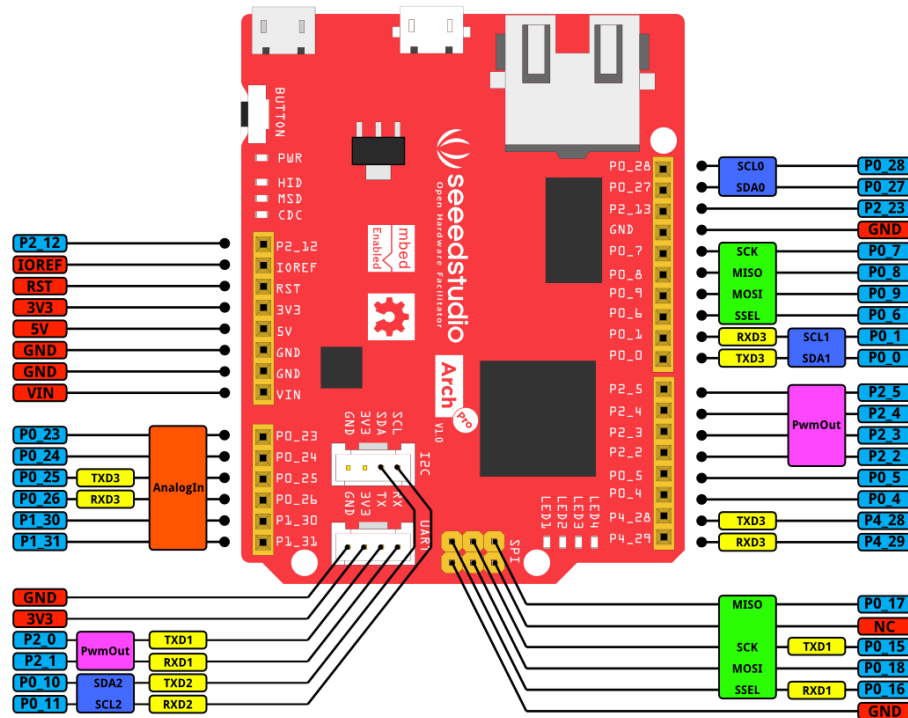


Figure 5: Pinout MBED

#### 2. Capteurs

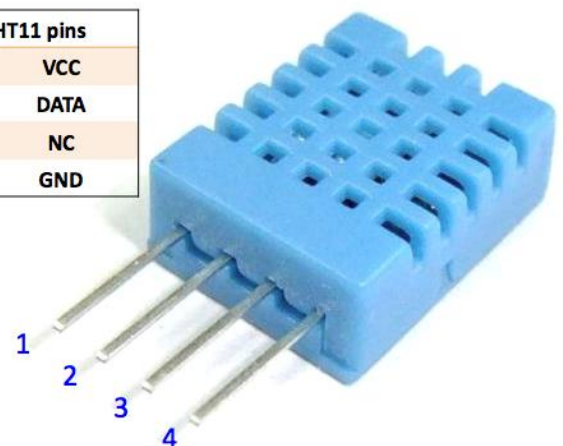
##### a) Capteur de température et d'humidité

Référence capteur : DHT11

- Pin\_01 ⇔ VCC ⇔ 5V (Arch-Pro)
- Pin\_02 ⇔ Data ⇔ P2\_12 (Arch-pro)
- Pin\_03 ⇔ non connecté
- Pin\_04 ⇔ GND ⇔ GND (Arch-pro)

Note : pour le bon fonctionnement du capteur il est nécessaire d'ajouter entre le Pin\_01(VCC) et le Pin\_02(Data) une résistance De pull-up de 10Koh.

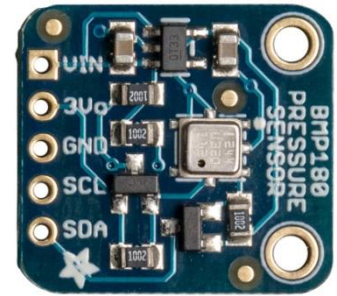
DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



b) Capteur de température et de pression

Référence capteur : BMP180

- Pin\_Vin ⇔ 5V (Arch-Pro)
- Pin\_3V0 ⇔ non connecté
- Pin\_GND ⇔ GND (Arch-Pro)
- Pin\_SCL ⇔ P0\_28 (Arch-Pro)
- Pin\_SDA ⇔ P0\_27 (Arch-Pro)

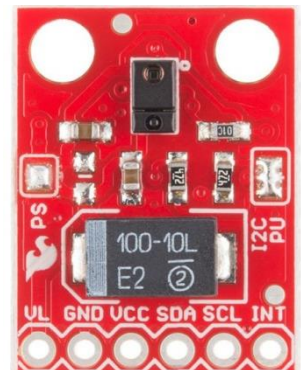


Note : le capteur utilise un bus I<sup>2</sup>C.

c) Capteur de présence (détection de mouvements) :

Référence Capteur : sparkfun 9960

- Pin\_VL ⇔ non connecté
- Pin\_GND ⇔ GND (Arch-Pro)
- Pin\_VCC ⇔ 3.3V (Arch-Pro)
- Pin\_SDA ⇔ P0\_27 (Arch-Pro)
- Pin\_SCL ⇔ P0\_28 (Arch-Pro)
- Pin\_INT ⇔ P0\_24 (Arch-Pro)



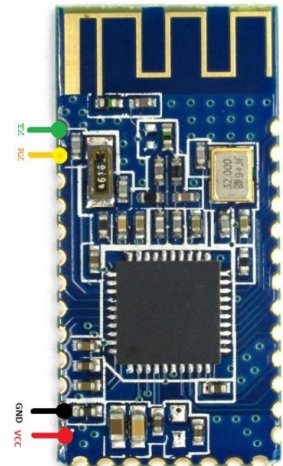
Note : le capteur utilise un bus I<sup>2</sup>C, et il est sur le même bus que le capteur BMP180, par ailleurs le PIN\_INT est un Pin d'interruption et est connecté à une entrée analogique.

d) Emetteur BLE

Référence BLE : HM-10

- Pin\_VCC ⇔ 3.3V (Arch-Pro)
- Pin\_GND ⇔ GND (Arch-Pro)
- Pin\_RX ⇔ P4\_28 (Arch-Pro)
- Pin\_TX ⇔ P4\_29 (Arch-Pro)

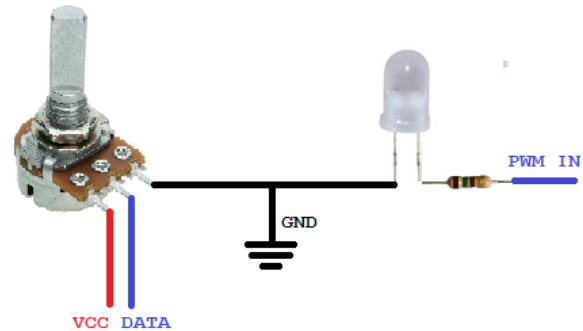
Note : le Pin\_VCC peut aussi être connecté a du 5V



e) *Potentiomètre et Led :*

Le but ici est de contrôler la luminosité de la LED avec un PWM, et cela proportionnellement à la valeur renvoyée par le potentiomètre.

- Pin\_VCC ⇔ 5V (Arch-Pro)
- Pin\_GND ⇔ GND (Arch-Pro)
- Pin\_DATA ⇔ P1\_31 (Arch-Pro)
- Pin\_PWM\_IN ⇔ P2\_5 (Arch-Pro)

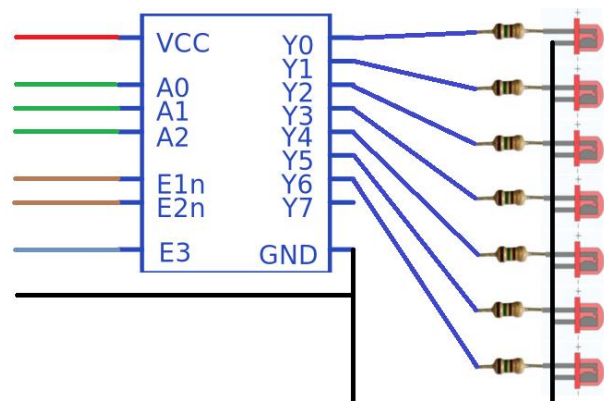


f) *Démultiplexeur et Led*

L'objectif est de pouvoir contrôler 7 LEDs avec uniquement 3 Pins du microcontrôleur à l'aide d'un multiplexeur.

Référence multiplexeur : CD74HC238E

- Pin\_VCC ⇔ 5V (Arch-Pro)
- Pin\_GND ⇔ GND (Arch-Pro)
- Pin\_A0 ⇔ P2\_13 (Arch-Pro)
- Pin\_A1 ⇔ P0\_1 (Arch-Pro)
- Pin\_A2 ⇔ P0\_2 (Arch-Pro)
- Pin\_E1n ⇔ GND (Arch-Pro)
- Pin\_E2n ⇔ GND (Arch-Pro)
- Pin\_E3 ⇔ VCC (Arch-Pro)



Note : les entrées du multiplexeur sont activées à l'état bas (0 Logique ).

## C. Configuration BLE

Ce tutoriel présente deux façons de configurer le BLE (HM-10) afin de pouvoir l'utiliser pour émettre des données.

### **Technique 01 : Configuration à l'aide des commandes « AT »**

Il faut utiliser les commandes suivantes :

1. AT+RENEW : réinitialisation des paramètres du constructeur
2. AT+RESET : réinitialise l'HM-10
3. AT : test de la communication (attendre de recevoir AT+OK)
4. AT+MODE2 : mode émission et réception
5. AT+ADTY0 : rend le HM-10 connectable

Pour ce faire, il y a deux techniques. Soit utiliser le mini-programme fourni du nom de « configuration BLE » qui est flashé dans la carte ARM et configure le HM-10 avec le bus UART, soit utiliser un câble USB-FTDI avec l'interface série de ARDUINO, ou directement avec la console linux en écrivant et en lisant le fichier « /dev/ttyAMCx »

```
13  bluetooth.printf("AT+RENEW")
14  lectureReponseAT();
15  bluetooth.printf("AT+RESET")
16  lectureReponseAT();
17  bluetooth.printf("AT")
18  lectureReponseAT();
```

*Note : une commande AT envoyé par UART correspond à une chaîne de caractères.*

### **Technique 02 : Mise à jour du Firmware avec un Firmware configuré de base en émetteur.**

Pour ce faire, il faut aller sur le site web du constructeur ([www.jnhuamao.cn/](http://www.jnhuamao.cn/)) et télécharger le bon firmware, dans la section « download Rom ».

Une fois l'archive téléchargée, vous y trouverez 3 fichiers :

1. Readme.txt : qui contient les instructions d'installation
2. Fichier firmware avec l'extension .bin
3. Fichier de mise à jour avec l'extension .exe (HMSoft.exe)

Lancer l'exécutable HMSoft.exe et suivre les instructions du fichier Readme.txt, une fois le firmware mis à jour il est préconfiguré en mode émetteur, sur le service 0xffef.

*Note : la 2eme technique n'est utilisable que sur Windows.*

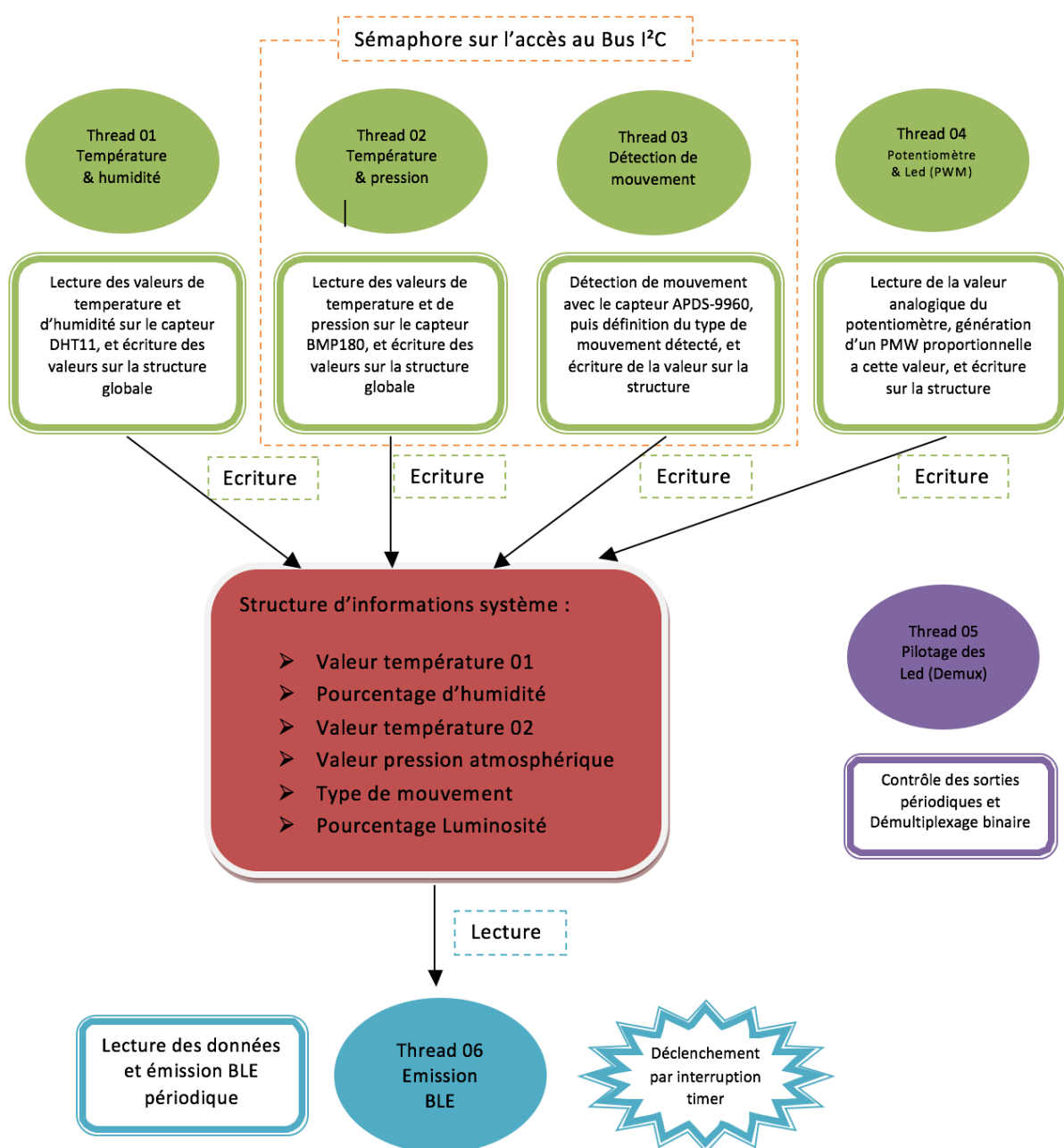
## D. Architecture logicielle

Dans cette partie nous allons expliciter la structure du code, donnant par là une vue globale du fonctionnement de l'architecture logiciel du système.

Le programme est relativement simple, il est basé sur la notion de multithreading, de façon à ce que chaque module électronique (capteur ou autre ...) soit contrôlé indépendamment des autres modules par un thread spécifique à celui-ci.

Par ailleurs chaque thread dispose d'une visibilité directe (variable globale) sur une structure de données représentant les données capteurs récoltés.

Le synopsis suivant offre une représentation graphique de cette architecture.



## E. Explications code

Dans cette section nous allons décrire le fonctionnement interne de chaque partie du programme et proposer de reconstruire le code de l'application, brique par brique, explicitant ainsi le fonctionnement interne de chaque brique à la volée.

### Généralités :

Tout d'abord il est important de noter que l'architecture de l'application peut être divisée en 3 sections :

1. Bibliothèques liées au composant (modules électroniques).
2. Fichier de configuration (définitions des Pin, des périodes timer, ...).
3. Code principal (Corps principal de notre programme).

### 1. Bibliothèques des composants

Comme présenté précédemment nous utilisons pour notre application plusieurs modules électroniques qui nécessitent pour fonctionner l'importation de bibliothèques.

**DHT11** : Capteur de température et d'humidité

- Importer la bibliothèque MBED : DHT11
- Inclure dans le main le fichier header : "DHT11.h"

**BMP180** : Capteur de température et de pression

- Importer la bibliothèque MBED : MBP180
- Inclure dans le main le fichier header : "BMP180.h"

**APDS-9960** : Capteur de mouvement et luminosité

- Importer la bibliothèque MBED : Sparkfun\_APDS9960
- Inclure dans le main le fichier header : "apds9960.h"

**PMW** : pour pouvoir générer et contrôler une sortie PWM

- Importer la bibliothèque MBED : SoftPWM
- Inclure dans le main le fichier header : "SoftPWM.h"

**RTOS** : pour pouvoir mettre MBED-RTOS sur la carte, et utiliser les fonctionnalités du RTOS

- Importer la bibliothèque MBED : mbed-rtos
- Inclure dans le main le fichier header : "rtos.h"

Par ailleurs il est important pour le bon fonctionnement de l'application de d'inclure 2 fichiers header :

- "mbed.h" : qui contient toutes les fonctions, types, adresses de l'environnement MBED.
- "configuration.h" : qui contient toutes les déclarations de constantes symboliques représentant les branchements des Pin sur la carte et différents temps et périodes pour les Timer, ainsi que la définition de la structure de données systèmes qui contiendra les valeurs des différents capteurs.

*Note : pour savoir comment faire pour importer des bibliothèques sur votre espace de développement MBED, merci de bien vouloir se référer au tutoriel sur l'utilisation de l'environnement MBED.*

## 2. Fichier de configuration :

Ce fichier comme son nom l'indique, est un fichier de configuration et permet entre autres de faire l'interfaçage entre le logiciel et le matériel. Autrement dit, ce fichier nous permet de faire la jointure entre les PINs des modules électroniques utilisés et les PINs de la carte principale (Seeduino-Arch-Pro), et cela sous forme de constantes symboliques (#define).

Par ailleurs, le fichier contient aussi la définition de la structure d'informations qui représente le conteneur des valeurs des différents capteurs, de plus il contient les déclarations des prototypes des fonctions utilisées dans le fichier principal.

*Note : si vous décidez de changer la carte principale Seeeduin-Arch-Pro par une autre carte ARM, alors nul besoin d'aller modifier le code principal de l'application, il vous suffira juste d'aller modifier les PIN déclarés dans le fichier configuration.h*

### **Exemple :**

Pour le capteur de température et d'humidité DHT11, il suffit d'aller dans le fichier configuration. Vous trouverez que le Pin auquel est relié le DHT11 sur la Seeeduino-Arch-Pro est le pin P2\_12. Il vous suffirait alors de modifier cette valeur pour y mettre le nouveau Pin correspondant à votre nouveau branchement sur la nouvelle carte (ex : P4\_21) sans n'avoir rien d'autre à modifier et votre application tournera parfaitement sur la nouvelle carte.

Code de base :

```
17 /*****  
18 /*****  
19 /*          THREAD TEMPERATURE ET HUMIDITE          */  
20 /*****  
21 /*****  
22  
23 #define PIN_TEMPERATURE_HUMIDITY_SENSOR      P2_12  
24 #define TIME_WAIT_MS_TEMPERATURE_HUMIDITY_SENSOR  3000  
25
```

Code configuré :

```
17 /*****  
18 /*****  
19 /*          THREAD TEMPERATURE ET HUMIDITE          */  
20 /*****  
21 /*****  
22  
23 #define PIN_TEMPERATURE_HUMIDITY_SENSOR      P4_21  
24 #define TIME_WAIT_MS_TEMPERATURE_HUMIDITY_SENSOR  3000  
25
```

Partant du même principe, ce fichier nous permet également de pouvoir, sans toucher le code principal, modifier les périodes, les Waits pour les threads, et autres valeurs disponibles dans le fichier.



Pour modifier la période d'émission du BLE :

```
61 /*****  
62 /*****  
63 /*                                BLE                                */  
64 /*****  
65 /*****  
66  
67 #define PIN_BLE_TX                P4_28  
68 #define PIN_BLE_RX                P4_29  
69 #define TIME_MS_PERIODE_BLE      5000  ← Periode d'émission BLE en Ms  
70
```

### 3. Code Principal

Le code principal est contenu dans le fichier Main.c, où toutes les définitions de fonction ainsi que leurs utilisations sont regroupées.

Comme vu précédemment l'architecture du programme est plutôt simple et s'organise autour de 6 threads, cette section se propose d'étudier la construction et le fonctionnement de ces threads l'un après l'autre :

#### Thread Température et humidité (DHT11):

Le fonctionnement de ce thread est simple, après avoir instancié un objet du type DHT11 avec le pin sur l'Arch-Pro correspondant au pin de branchement du Pin Data du composant, et cela en utilisant une des constantes symboliques déclarées dans le fichier de configuration :

```
65    DHT11 capteur(PIN_TEMPERATURE_HUMIDITY_SENSOR);
```

Une fois l'objet instancié, on rentre dans une boucle infinie :

```
68    while (true) {
```

Puis dans un premier temps il faudra demander à l'objet de calculer les valeurs de température et d'humidité avec la fonction readData() et de récupérer la valeur de retour de la fonction :

```
70        tmp = capteur.readData();
```

Puis tester s'il n'y a pas eu de problèmes en testant la valeur de retour de la fonction et on la compare à une constante symbolique OK de la bibliothèque DHT11, et afficher une erreur s'il y a un problème :

```
72        if (tmp != DHT11::OK) {  
73            printf("Error! %d\r\n", tmp);  
74        }
```

Dans le cas où il n'y aurait pas de problème, il suffira alors d'aller lire les champs de l'objet instancié correspondant aux valeurs souhaitées et mettre à jour les champs correspondant dans notre structure de données globale, et cela avec les fonctions définies dans l'objet (car champ privé, donc pas d'accès direct, pour plus d'informations sur la structure de l'objet, voir le fichier DHT11.c) :



```

76         inf.temperature_01 = capteur.readTemperature();
77         inf.humidite = capteur.readHumidity();

```

Une fois la lecture terminée on mettra le thread en attente pendant une durée définie avec la constante symbolique déclarée dans le fichier de configuration :

```

81         Thread::wait(TIME_WAIT_MS_TEMPERATURE_HUMIDITY_SENSOR);

```

**Note :** la période de sommeil (idle) du thread qui est représenté par la constante symbolique `TIME_WAIT_MS_TEMPERATURE_HUMIDITY_SENSOR` doit être supérieure à 2500 ms, qui correspond au temps minimum entre deux mesures du capteur DHT11 spécifié par la documentation du fournisseur.

### **Thread Température et pression (BMP180)**

Ici le principe est le même que pour le thread précédent à une exception qui est que le BMP180 communique avec la carte ARCH-PRO sur le même bus I<sup>2</sup>C que le APDS9960 et pour éviter des conflits sur le bus nous avons mis en place un sémaphore, son utilisation est simple, on demande la ressource avant de l'utiliser avec un Wait, et on relâche la ressource avec un release, il suffit alors de protéger toutes les zones critiques par ces deux fonctions comme suite :

```

140         two_slots.wait();
141         // Code critique
142         two_slots.release();

```

Pour ce qui est de l'utilisation du MPB on commence par instancier un objet avec le Pin correspondant défini par la constante symbolique définie dans le fichier de configuration comme suite

```

137         BMP180 bmp180(PIN_PRESURE_SENSOR_SDA, PIN_PRESURE_SENSOR_SCL);

```

Une fois l'objet instancié, on rentre dans une boucle infinie :

```

68         while (true) {

```

Puis, à l'aide d'une fonction de la bibliothèque du BMP180, on effectue la lecture de la valeur de la pression atmosphérique et de la température puis on écrit les valeurs sur les 2 variables passées par référence comme suite :

```

142         error = bmp180.readTP(&temp, &pressure, OVERSAMPLING_ULTRA_HIGH_RESOLUTION);

```

Puis il suffit de tester la valeur de la variable error pour savoir si l'opération c'est bien passé, et afficher un message d'erreur le cas échéant :

```

144         if (error)
145             printf("Error is %d\r\n\r\n", error);

```

Si la lecture c'est bien passé il suffira alors de mettre à jour les champs correspondant dans la structure des données globales que nous avons définies :

```

147         inf.temperature_02 = temp;
148         inf.pression = pressure;

```

Une fois les champs de la structure mis à jour, il suffira de faire passer le thread en mode attente (sommeil/idle) pour une période définie par la constante symbolique définie dans le fichier de configuration :

```

154     Thread::wait(TIME_WAIT_MS_PRESURE_SENSOR);

```

*Note: la section critique a protégée par le sémaphore ici, correspond à la lecture du capteur et doit donc être protégé :*

```

140     two_slots.wait();
141     error = bmp180.readTP(&temp, &pressure, OVERSAMPLING_ULTRA_HIGH_RESOLUTION);
142     two_slots.release();

```

### **Thread (Packfun APDS-9960)**

Ce thread a pour objectif de détecter tout mouvement se produisant devant le capteur APDS9960, puis de définir le type de ce mouvement et de mettre à jour le champ correspondant dans la structure de données globale définie :

{Droite, gauche, haut, bas, se rapproche, s'éloigne, autre, pas de mouvement}

Tout d'abord il faut créer un objet du type APDS puis l'initialiser comme ceci :

```

28 apds9960      sensor(PIN_MOUVEMENT_SENSOR_SDA, PIN_MOUVEMENT_SENSOR_SCL);
237     two_slots.wait();
238     while(!sensor.ginit(pc)){
239         printf("Something went wrong during APDS-9960 init\n\r");
240         Thread::wait(TIME_WAIT_MS_INITIALISATION_FAILURE);
241     }
242     printf("APDS-9960 initialization complete\n\r");
243
244     // Start running the APDS-9960 gesture sensor engine
245     while(!sensor.enableGestureSensor(true)){
246         printf("Something went wrong during gesture sensor init!\n\r");
247         Thread::wait(TIME_WAIT_MS_INITIALISATION_FAILURE);
248     }
249     printf("Gesture sensor is now running\n\r");
250     two_slots.release();

```

*Note: l'APDS étant sur le même bus I<sup>2</sup>C que le BMP180 l'accès à cette ressource (bus I<sup>2</sup>C) est projeté par un sémaphore.*

A chaque détection de mouvements le capteur génère une interruption. Donc la première étape est de créer l'objet interruption en variable globale :

```

29 InterruptIn    interrupt(PIN_MOUVEMENT_SENSOR_INTERRUPT);

```

Puis de lui associer une fonction de Callback qui correspond à l'action faite à chaque détection, ici on veut mettre à true un boolean qui nous servira de valeur de test pour le lancement du traitement

```

252     interrupt.fall(&trigger);
167 void trigger() {
168 //     pc.printf("triggered\n\r");
169     intFlag = true;
170 }

```

Puis il suffira de boucler à l'infini et de tester la valeur de notre boolean, quand celui-ci sera à vrai, il nous suffira de lancer la fonction de lecture du mouvement et de mettre à jour la structure de données globale :

```

256     if(intFlag) {
257         two_slots.wait();
258         printGesture(getGesture());
259         two_slots.release();
260         // Clean interrupt handler flag.
261         intFlag = false;
262     }

```

**Note :** la lecture étant sur le bus I<sup>2</sup>C ici aussi il est important de projeter l'accès à cette ressource avec le sémaphore.

Puis il suffira de mettre le thread en sommeil (idle) pour une durée définie par une constante symbolique du fichier de configuration :

```

264         // Do somethings else
265         wait_ms(TIME_WAIT_MS_MOUVEMENT_SENSOR);

```

## **Thread BLE**

Le thread est lancé périodiquement à l'aide d'un timer (la procédure est expliquée dans la section relative au programme principal main), et cela dans le but d'aller lire les différents champs de la structure globale, puis d'émettre ses valeurs avec un certain format qui sera explicité ci-dessous :

Tout d'abord il faut créer l'objet Bluetooth qui correspond à une liaison série (UART) avec l'Arch-pro, et cela avec les constantes symboliques définies dans le fichier de configuration :

```

30 Serial          bluetooth(PIN_BLE_TX, PIN_BLE_RX);

```

Puis il suffit d'aller écrire sur le bus série pour envoyer les données :

```

287     bluetooth.printf("%3d%3d%3d%6d%3d%1d" ,inf.temperature_01
288                      ,inf.humidite
289                      ,inf.temperature_02
290                      ,inf.pression
291                      ,inf.luminosite
292                      ,inf.mouvement);

```

Il est important de savoir que le BLE ne peut émettre des trames supérieures à 20 octets, et c'est pour cela qu'ici nous avons mis en place un format de données Ascii très simple pour l'émission :

- ⇒ Les 3 premiers octets correspondent à la température 01
- ⇒ Les 3 suivants à l'humidité
- ⇒ Les 3 suivants à la température 2
- ⇒ Les 6 suivants à la pression
- ⇒ Les 3 suivants à la luminosité
- ⇒ Le dernier octet au type de mouvement

Une fois l'émission terminée il convient de remettre la valeur du mouvement dans la structure globale de données à « 0 » ce qui correspond à pas de mouvement détecté

```
294     inf.mouvement = 0;
```

### **Thread Led et démultiplexage**

Ce thread a pour objectif de piloter un démultiplexeur 3 vers 8, afin de piloter des circuits ou des actionneurs ici on l'occurrence l'exemple est celui de 7 Led's

Dans un premier temps on crée un tableau unidimensionnel de taille 3 qui contiendra 3 Objets du type DigitalOut (sortie numérique GPIO) qui nous serviront pour le pilotage des LEDs, les entrées du multiplexeur sont activées à l'état bas ce qui revient, par ailleurs les Pin' sont initialisés avec des constantes symboliques définies dans le fichier de configuration :

```
92     DigitalOut tab[] = {PIN_MUX_P0, PIN_MUX_P1, PIN_MUX_P2};
```

Puis il suffit de faire varier les valeurs des différentes cases du tableau on lui affectant des valeurs :

```
97     //printf("000 \n\r");
98     tab[0] = LED_OFF; tab[1] = LED_OFF; tab[2] = LED_OFF;
```

- 111 => sortie 0 active
- 110 => sortie 1 active
- 101 => sortie 2 active
- 100 => sortie 3 active
- 011 => sortie 4 active
- 010 => sortie 5 active
- 001 => sortie 6 active
- 000 => sortie 7 active

**Note:** pour plus d'informations, voir la documentation du CD74HC238E

Ici pour notre exemple on met le thread en sommeil (idle) après chaque modification des valeurs sur les entrées du démultiplexeur :

```
99     Thread::wait( TIME_WAIT_BLINK_LED );
```

**Note:** la constante symbolique TME\_WAIT\_BLINK\_LED représente le temps en Ms de sommeil du thread et est configurable à partir du fichier de configuration.

### **Thread Potentiomètre Led et PWM**

Ce thread a pour but de contrôler la luminosité d'une LED en générant une sortie PWM proportionnelle à la valeur lue sur le potentiomètre connecté à l'Arch-Pro. Tout d'abord il faut instancier 2 objets l'un pour la sortie PWM et l'autre pour l'entrée analogique du potentiomètre :

```
305     AnalogIn pot(PIN_POTENTIOMETRE);
306     SoftPWM led = PIN_PWM_LED;
```

Puis il faut définir la période du PWM en Ms comme ceci :

```
308 led.period_ms(PWM_PERIODE_MS);
```

**Note :** *il existe d'autres fonctions à l'intérieur de la bibliothèque pour permettre d'initialiser l'objet PWM à l'instanciation ... (Merci de bien vouloir voir le fichier SoftPWM.c).*

Une fois l'initialisation terminant, il suffit d'aller lire en continue la valeur échantillonnée du potentiomètre et de l'appliquer au PWM :

```
310 while (true) {
311     if(pot.read() < PWM_VALUE_MIN){
312         led = PWM_LED_OFF;
313         inf.luminosite = 0;
314     }
315     else if (pot.read() > PWM_VALUE_MAX){
316         led = PWM_LED_ON;
317         inf.luminosite = 100;
318     }
319     else {
320         led = pot.read();
321         inf.luminosite = pot.read()*100;
322     }
323 }
```

**Note :** *pour des soucis pratiques, les valeurs, de luminosité comprise entre 100 et PWM\_VALUE\_MAX (ici préconfigurée dans le fichier de configuration à 95) est automatiquement mis à 100 sur la sortie du PWM, respectivement pour les valeurs comprises entre 0 et PWM\_VALUE\_MIN (ici à 5) sont automatiquement mise à 0, pour toutes les autres valeurs du potentiomètre (5 – 95) celles-ci sont directement recopiées sur la sortie du PWM. Par ailleurs à chaque modification de la valeur de la luminosité le champ correspondant dans la structure de données globale est lui aussi mis à jours par la même occasion.*

### **Le programme principal « Main » :**

Le rôle du « main » est de créer les threads de gestion des modules électronique et de les lancer, sachant que le Main lui-même est considéré comme un thread.

Pour lancer un thread il suffit d'instancier un objet Thread avec une adresse de fonction, et un paramètre de type void\* qui est facultatif ( ici il nous servira à passer le nom du thread en paramètre ), une fois instancier le thread est lancé automatiquement et est géré par l'ordonnanceur du RTOS, évidemment il existe plusieurs fonctions permettant d'agir sur le cycle de fonctionnement des threads, pour avoir plus d'informations merci de bien vouloir consulter le lien fourni dans la bibliographie sur MBED RTOS.

```
46 Thread t1(thread_temperature, (void *)"Thread temperature");
47 Thread t2(thread_pression , (void *)"Thread pression");
48 Thread t3(thread_presence , (void *)"Thread presence");
49 Thread t4(thread_led , (void *)"Thread Led");
```

Pour ce qui est des interruptions Timer le principe est le même sauf qu'ici le thread est lancé non par le main mais par un Timer, il suffit alors de créer un objet RtosTimer et de l'instancier avec une fonction de callBack qui ici correspond au thread

qu'on veut lancer, une fois l'instanciation terminée, il suffit de lancer le timer avec comme paramètre la valeur de la période des déclenchements des interruptions, ici la valeur est une constante symbolique définie dans le fichier de configuration, pour avoir plus d'informations merci de bien vouloir consulter le lien fourni dans la bibliographie sur MBED RTOS

```
51   RtosTimer BleSend(bleCallBack, osTimerPeriodic, (void *) "Ble emission");  
52   BleSend.start(TIME_MS_PERIODE_BLE);
```

Comme expliqué précédemment le « main » représente lui aussi un thread, qu'on exploite ici au lancement de la fonction de gestion de la luminosité de la LED avec le PWM et le potentiomètre. Il s'agit alors d'un simple appel de fonctions.

```
54   potAndPwm();
```

*Note : attention la fonction appelée doit être infinie (avoir une boucle infinie) sinon le thread main s'arrêtera de s'exécuter, arrêtant par là tous les threads lancés par ce dernier.*

### III. Raspberry Pi 2

#### A. Présentation de l'environnement

Dans notre projet, la Raspberry Pi sert de liaison entre notre RTOS et Bluemix. Son rôle va être de récupérer les données envoyées en BLE par le RTOS, de les traiter, et de les envoyer à Bluemix.

Bien que cette partie traite de la Raspberry, ce qui a été fait sur cette plateforme peut être implémenté sur n'importe quelle autre, à condition d'avoir une connexion Wifi et BLE.

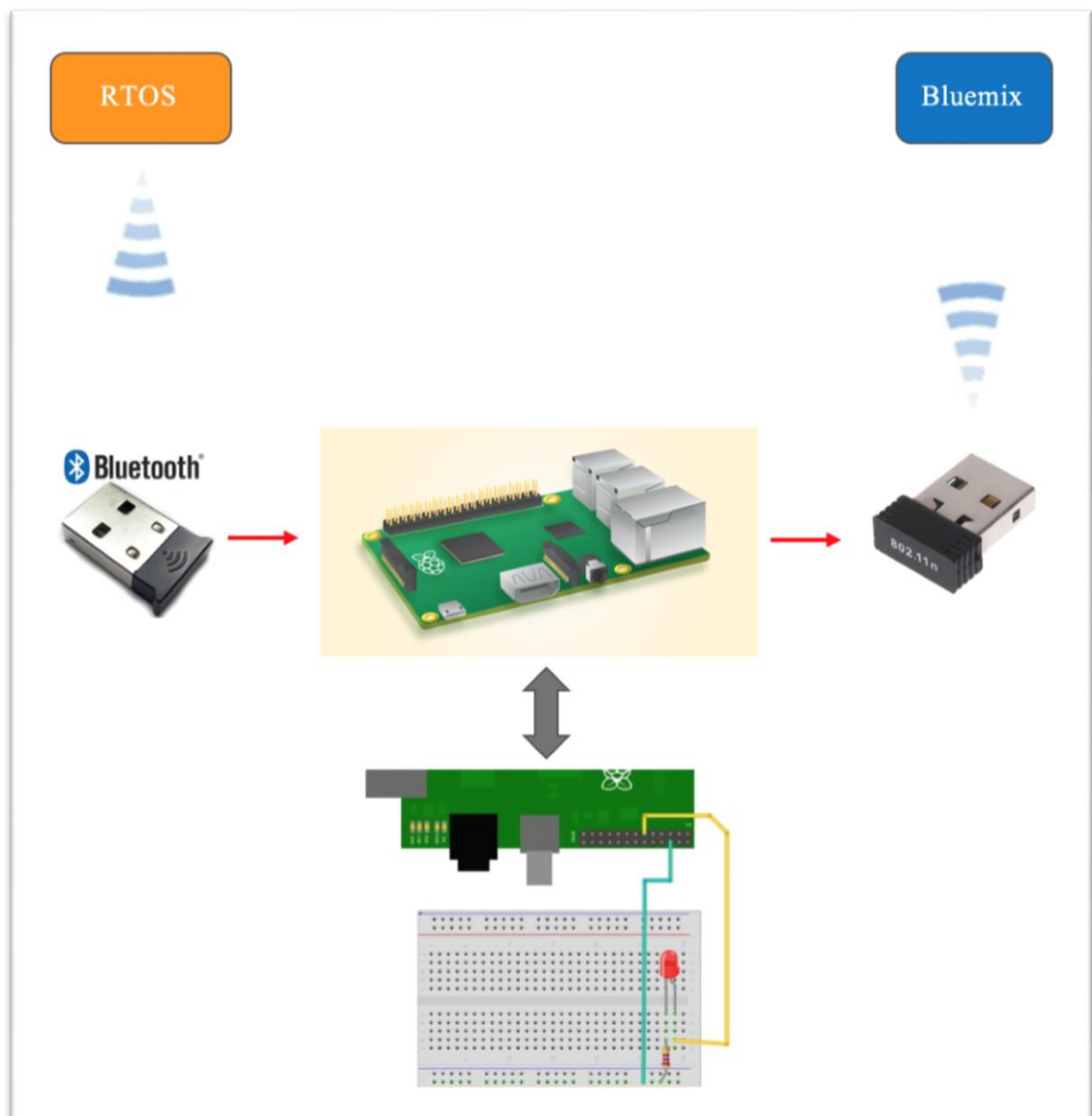


Figure 6: Connexions Raspberry Pi 2

Dès la réception des données envoyées par le RTOS, la Raspberry traite celles-ci, puis les renvoie au service Bluemix, tout en commandant le scintillement d'une LED à chaque envoi. Nous allons détailler maintenant la configuration de notre carte.

## B. Configuration du système

### 1. BLE

La première étape est donc de configurer la Raspberry afin de pouvoir récupérer avec succès les données émises en BLE par le RTOS. Cette étape est à faire avant toute chose, sinon les modules suivants ne pourront pas s'installer.

```
pi@raspberrypi ~/Desktop/IOT_Raspberry $ sudo apt-get install bluetooth bluez libbluetooth-dev libudev-dev libdbus-1-dev libglib2.0-dev libical-dev libreadline-dev
```

### 2. Node

On ajoute les dernières sources pour node js

```
pi@raspberrypi ~/Desktop/IOT_Raspberry $ curl -sL https://deb.nodesource.com/setup_0.12 | sudo -E bash -
```

Puis, on installe nodejs :

```
pi@raspberrypi ~/Desktop/IOT_Raspberry $ sudo apt-get install -y nodejs
```

Enfin, on crée un linkage afin d'éviter des crashes sur certaines versions de node.

```
pi@raspberrypi ~/Desktop/IOT_Raspberry $ sudo ln -s /user/bin/nodejs /usr/bin/node
```

### 3. Npm

L'avantage de l'outil npm est que nous pouvons préciser dans un fichier tous les modules node à installer.

Dans notre cas, nous aurons besoin de plusieurs modules :

- **Moment** : permet de récupérer l'heure à un format prédéfini
- **Mqtt** : permet de gérer les connexions MQTT
- **Ibmiof** : module créé par ibm afin de gérer les publications au cloud IBM
- **Onoff** : permet la gestion des GPIOs
- **Noble** : permet la gestion du bluetooth
- **Bluetooth-hci-socket** : permet l'utilisation de gatttool et hcitool avec node

L'installation de tout ces modules se fait en lançant la commande suivante :

```
pi@raspberrypi ~/Desktop/IOT_Raspberry $ npm install
```

### 4. Lancement du programme

Il suffit maintenant que de lancer le programme, à l'aide de la commande :

```
pi@raspberrypi ~/Desktop/IOT_Raspberry $ sudo node ble.js
```



## C. Explications code

Notre code a donc trois fonctions principales. La première est de scanner les périphériques bluetooth jusqu'à trouver celui qui nous intéresse, la seconde est de récupérer, traiter, puis envoyer les données au service Bluemix, et la dernière est de commander la LED.

Il faut, avant toute chose, configurer le serveur MQTT. Les paramètres de la connexion sont définis à l'aide des API et Token récupérés lors de la paramétrisation du terminal sur l'internet of things foundation. (Se référer à la section de paramétrage du service IOT foundation pour plus de détails).

```
var client = mqtt.connect('tls://yj5yyh.messaging.internetofthings.ibmcloud.com:8883',
{clientId:'d:yj5yyh:rasp2monitor:euro001',username:'use-token-auth',
password:'pq(jatiiAUtUylj6c&'}));

var rtosMac = '0017ea939da5';

var Gpio = require('onoff').Gpio, // Construction des objets GPIOs
led = new Gpio(17, 'out');        // Exportation du GPIO 17 en sortie
```

Le paquet « noble » permet d'exploiter les outils hcitool, hciconfig, ainsi que gatttool au sein de notre code nodejs.

Une fois la connexion établie, on lance le scan des périphériques bluetooth avoisinants.

```
// A l'allumage de l'adaptateur, on commence le scan
noble.on('stateChange', function(state) {
  if (state === 'poweredOn'){
    console.log('Scan des terminaux commencé');
    noble.startScanning();
  }
  else {
    noble.stopScanning();
  }
});
```

Le tronçon de code suivant peut se résumer ainsi :

1. On arrête le scan quand on a trouvé le périphérique voulu
2. On se connecte donc à ce périphérique
3. On accède au service voulu, puis aux caractéristiques de celui-ci.
4. On active les notifications, et on récupère à chaque réception des données la data envoyée par le RTOS. Après un formatage sous forme de chaîne, on lance la fonction de gestion de l'envoi à Bluemix

```

// Des qu'on découvre un périphérique, cette fonction est lancée
noble.on('discover', function(peripheral) {
  var macAdress = peripheral.uuid;
  // Si on a trouvé l'adresse du BLE, on arrête le scan, et on traite le périphérique
  if (macAdress === rtosMac) {
    noble.stopScanning();
    // On se connecte au périphérique
    peripheral.connect(function(error) {
      console.log('BLE trouvé, connecté ('+ macAdress+')');
      var service_capt = peripheral.advertisement.serviceUuids;
      console.log('Service uuid : '+ service_capt);
      peripheral.discoverServices(service_capt, function(error, services) {
        var deviceIndormationService = services[0];
        console.log('Connecté au service '+deviceIndormationService);
        deviceIndormationService.discoverCharacteristics(null, function(error,
characteristics) {
          var message = characteristics[0];
          console.log('Connecté au handle '+message);
          // Activation des notifications
          message.notify(true, function(error) {
            console.log('Notification activée');
          });

          // Se lance à la réception d'une donnée
          message.on('read', function(data, isNotification) {
            // On convertit la donnée et on lance la fonction d'envoi au service Bluemix
            var string_data = data.toString('utf8');
            console.log('Data reçue : '+ data);
            // Lance la gestion de l'envoi à Bluemix
            send_bluemix(string_data);
          });
        });
      });
    });
  }
});
})

```

Cette fonction va donc découper la chaîne réceptionnée et récupérer les informations des différents capteurs, puis les envoyer au format MQTT au service Bluemix initialisé plus haut. Aussi, à chaque envoi de données, une LED clignote pour ajouter une interaction utilisateur.

```

var send_bluemix = function(data) {

    var t1 = parseFloat(data.substring(0,3));
    var h = parseFloat(data.substring(3,6));
    var t2 = parseFloat(data.substring(6,9))/10;
    var p = parseFloat(data.substring(9,15))/100;
    var l = parseFloat(data.substring(15,18));
    var m = parseFloat(data.substring(18,19));

    var monval = { };
    monval.d = {Température1: t1, Humidité: h, Température2: t2, Pression: p,
Luminosité: l, Mouvement: m, status: "ok"};
    monval.ts = moment().add(1,'h').toISOString();

    var payload = JSON.stringify(monval);

    client.publish('iot-2/evt/itemsvc/fmt/json',payload);

    // Gère le scintillement de la LED à chaque envoi des données
    led.writeSync(1);

    var scint = setInterval(iv, 200);

    setTimeout(function () {
        clearInterval(scint); // Stop blinking
        led.writeSync(0); // Turn LED off.
    }, 2000);
}

var iv = function() {
    led.writeSync(led.readSync() ^ 1); // 1 = on, 0 = off
}

```

## **IV. Bluemix**

### **A. Présentation de l'environnement**

Bluemix est une plateforme cloud qui permet d'héberger des applications en ligne. Les serveurs d'IBM sont classés parmi les serveurs les plus puissants au monde. Ces serveurs sont principalement localisés aux États-Unis et en Angleterre. Bluemix intègre un ensemble de services allant de la base de donnée jusqu'à la création des applications mobiles, solution Internet of Things etc.

Ces services sont gratuits et payants, dépendant du degré d'utilisation et de la nature du besoin. Il faut rappeler que Bluemix supporte un certains nombres de langages de programmation tels que Java, Node.js, Go, PHP, Ruby. Cela permettra à un développeur de développer ses applications aussi bien pour le web que pour les plateformes mobiles.

Il existe aussi d'autres types de services que nous pouvons utiliser :

- Des runtimes, via CloudFoundry : c'est le service PaaS principal de Bluemix. Il fournit un environnement serveur déjà installé et configuré, que l'on peut personnaliser en ajoutant de nouveaux services.
- Des conteneurs, via Docker : c'est en quelque sorte une "couche" intermédiaire entre les runtimes et les machines virtuelles, qui vous permet de porter vos applications en ligne sans avoir à gérer le système d'exploitation.
- Des machines virtuelles, via OpenStack : c'est le plus bas niveau, où vous gérez directement le système d'exploitation. Ici, vous avez plus de contrôle mais il faut tout installer.

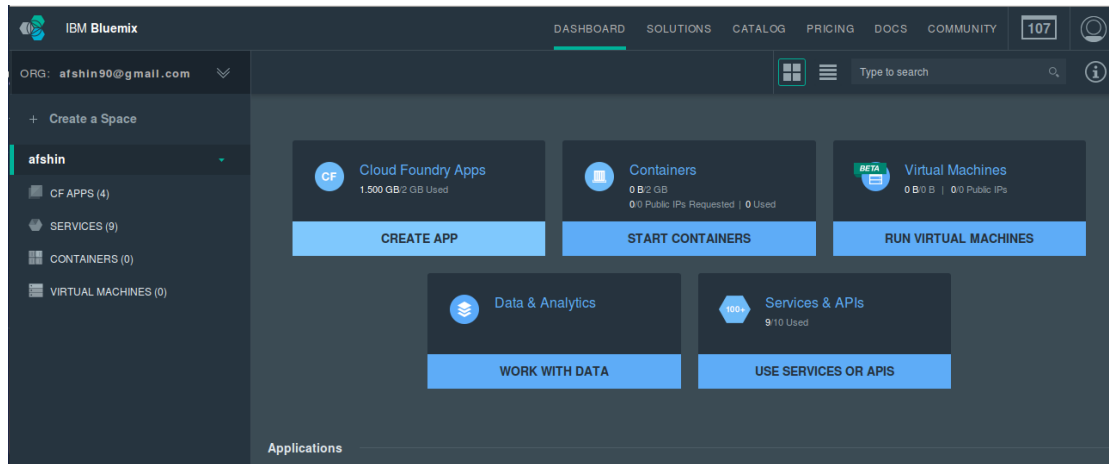
Nous allons maintenant expliquer comment bénéficier des services proposés.

Pour cela nous allons commencer par découvrir la procédure de création d'une application ; ensuite nous doterons celle-ci de différents services, comme le service Twilio, proposé par la plateforme Bluemix pour faciliter l'interaction entre la messagerie mobile et les applications Bluemix.

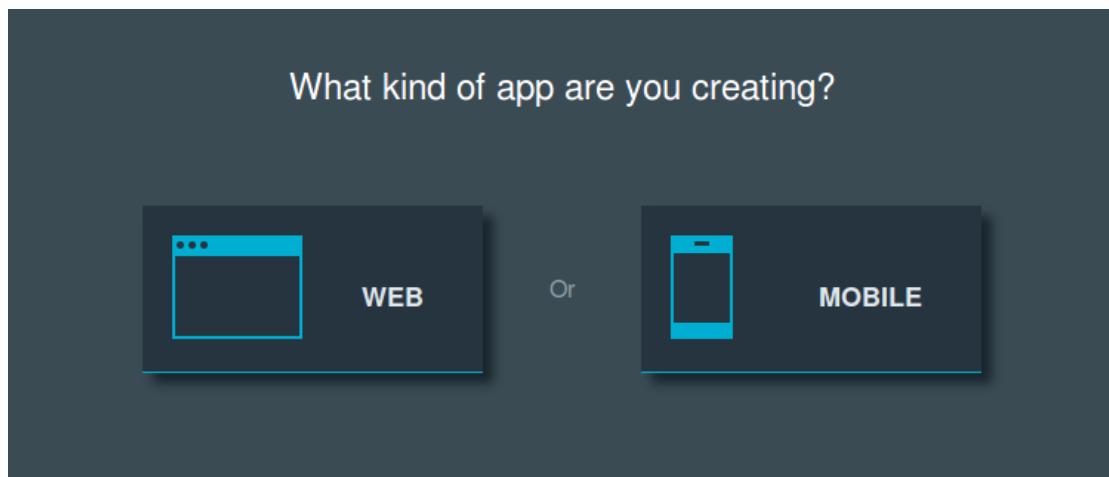
Remarque : l'utilisation du service Bluemix est payant. Néanmoins vous aurez un accès gratuit pendant 30 jours lors de votre première inscription.

## Création d'une application :

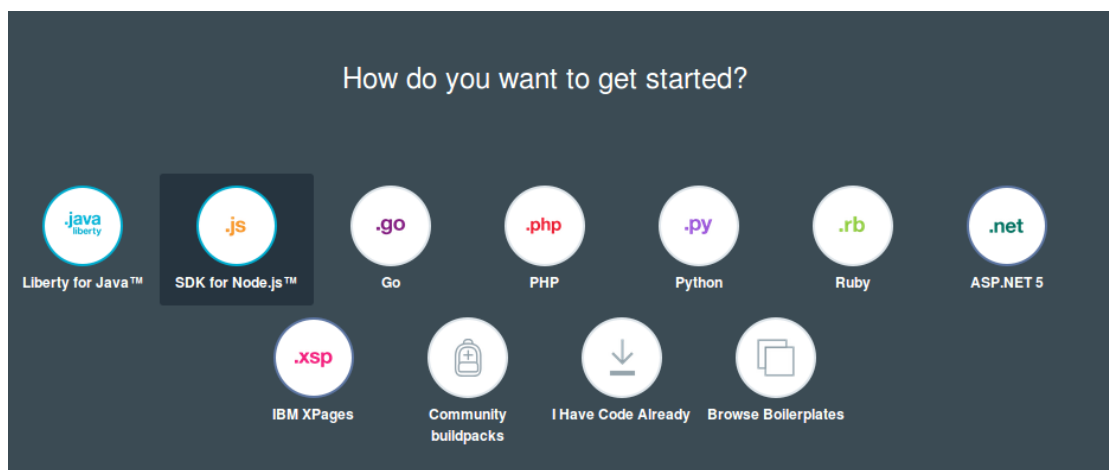
1. Se rendre sur le dashboard de Bluemix, puis cliquer sur CREATE APP dans l'espace intitulé Cloud Foundry Apps.



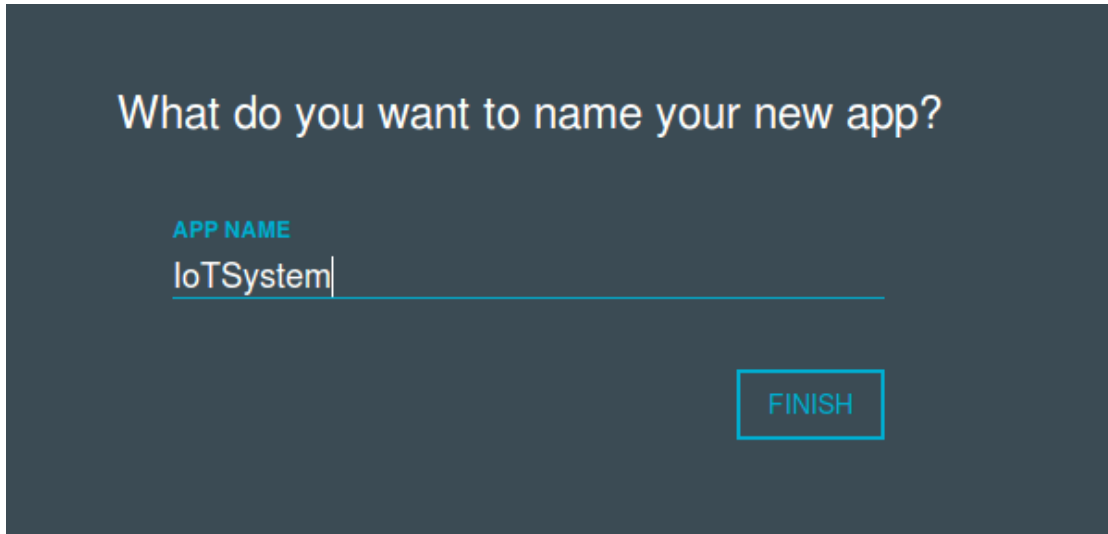
2. Dans cette étape vous devrez déterminer le type d'application que vous souhaitez développer. Nous choisissons ici une application WEB.



3. Puis, nous choisissons le langage de programmation pour notre application. Ici, ce sera du Node.js, outil très puissant permettant de créer une interface de communication aussi bien du côté serveur que du côté client. Sélectionner « SDK for Node.js » et continuer.



4. Puis, on nomme notre application

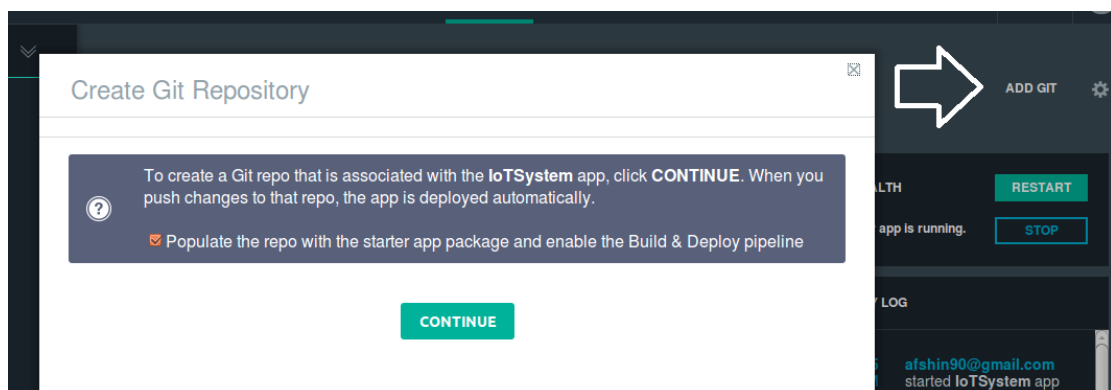


En l'espace de quelques secondes, notre application a été créée sur les clouds Bluemix.

#### **Utilisation des services de développement :**

Le service Bluemix vous donne la possibilité de développer votre application directement sur Cloud Foundry. Pour utiliser ces outils, suivre la procédure suivante :

1. Aller vers la page où se trouve l'application que vous venez de créer. En haut à droite se trouve un bouton ADD GIT. Appuyer sur ce bouton et ensuite sur continuer.



2. Apparaît maintenant une adresse github. Cliquer sur Edit Code. Ainsi vous accédez à l'environnement intégré de développement de Bluemix où un certain nombre d'outils seront à votre disposition.

The screenshot displays the IBM Bluemix DevOps Services interface. At the top, the application is named 'IoTSystem' with a Git URL: <https://hub.jazz.net/git/afshinkdoost90/IoTSystem>. The dashboard shows 1 instance, a memory quota of 256 MB, and 256.0 MB available. The 'APP HEALTH' section indicates 'Your app is running.' with 'RESTART' and 'STOP' buttons. The 'ACTIVITY LOG' shows recent events: 'started IoTSystem app' and 'updated IoTSystem app • changed routes'. Below the dashboard are buttons for 'ADD A SERVICE OR API' and 'BIND A SERVICE OR API'. The bottom section shows the code editor for 'app.js' with the following content:

```

1  /*eslint-env node*/
2
3  //-----
4  // node.js starter application for Bluemix
5  //-----
6
7  // This application uses express as its web server
8  // for more info, see: http://expressjs.com
9  var express = require('express');
10
11 // cfenv provides access to your Cloud Foundry environment
12 // for more info, see: https://www.npmjs.com/package/cfenv
13 var cfenv = require('cfenv');
14
15 // create a new express server
16 var app = express();
17
18 // serve the files out of ./public as our main files
19 app.use(express.static(__dirname + '/public'));
20
21 // get the app environment from Cloud Foundry
22 var appEnv = cfenv.getAppEnv();
23
24 // start server on the specified port and binding host
25 app.listen(appEnv.port, '0.0.0.0', function() {
26
27   // print a message when the server starts listening
28   console.log('server starting on ' + appEnv.url);
29 });
30


```

Si vous ne souhaitez pas utiliser cet outil de développement, vous avez parfaitement le choix de développer sur votre propre machine mais cela requiert un certain nombre d'outil supplémentaires, fournis une nouvelle fois par les outils de Cloud Foundry.

1. Appuyer de nouveau sur Start Coding.
2. Télécharger l'exécutable nommé CF de Bluemix qui permettra d'uploader tous vos fichiers depuis votre terminal machine vers votre espace cloud privé Bluemix.

## Deploying your app with the Cloud Foundry command line interface

You can use the Cloud Foundry command line interface to deploy and modify applications and service instances.


-  **Setup:** Before you begin, install the cf command line interface.



Restriction: The Cloud Foundry command line interface is not supported by Cygwin. Use the Cloud Foundry

3. Cliquer ensuite sur « Download Starter Code » qui contiendra tous vos fichiers.

After the cf command line interface is installed, you can get started:

- 1 Download your starter code.  

- 2 Extract the package to a new directory to set up your development environment.

Une fois que votre développement et vos tests sont terminés, vous pouvez les charger vers votre espace de stockage en procédant de la manière suivante :

```
cd your_new_directory
```

4. Connectez-vous à Bluemix en tapant :

```
cu api https://api.ng.bluemix.net
```

5. Ensuite , Identifiez-vous en tapant :

```
cf login -u votre_adresse_mail -o votre_adresse_mail -s  
nom-login
```

6. La dernière étape consiste à charger tous les fichiers vers votre espace à l'aide de la commande :

```
cf push nom_application
```

Il faut rappeler qu'une série de documents relatifs à l'utilisation des outils et de services de Bluemix vous sont proposés sous la rubrique DOCS. Pour approfondir le tutoriel et ajouter de nouvelles fonctionnalités, référez-vous à ces documentations.

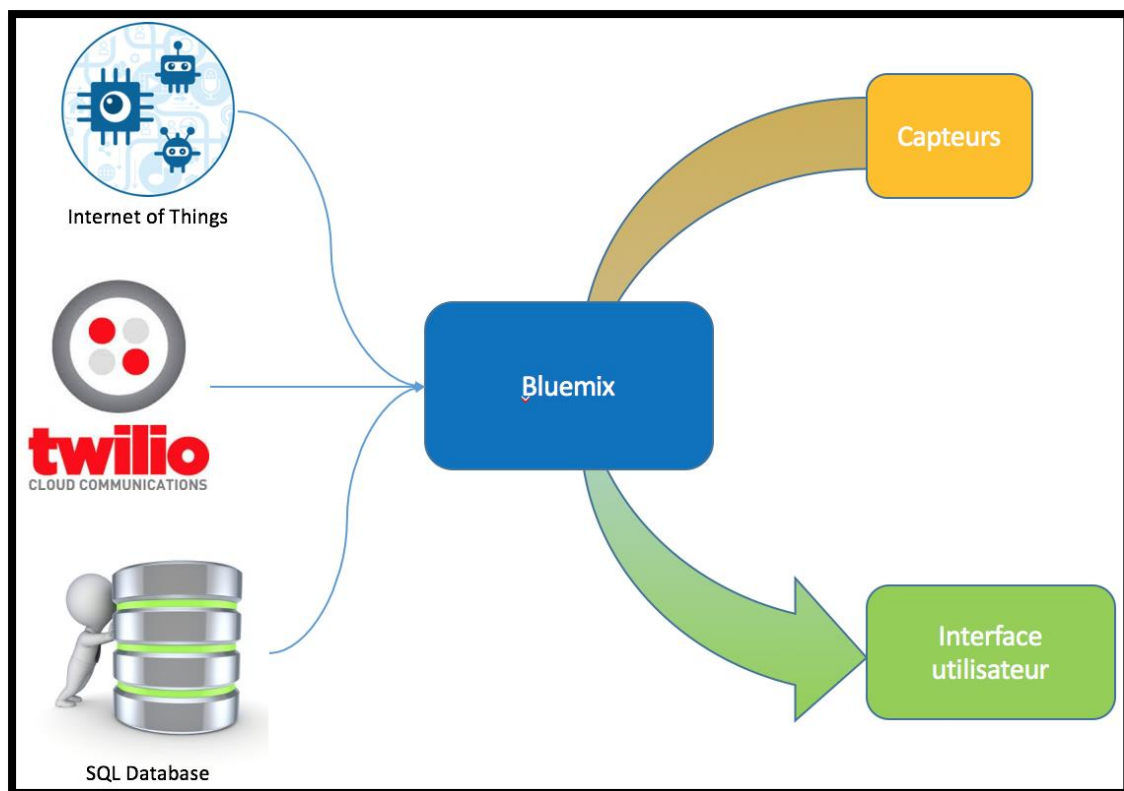
Autre élément à remarquer est l'option LOGS proposé dans votre menu principal à gauche qui vous permettra d'observer le terminal de Bluemix.

En effet, en tapant la commande suivante, on peut regarder les console.log produits par l'application pendant son exécution sur le cloud Bluemix

```
cf logs nom_application
```

Les services ajoutés à notre application sont illustrés dans la figure suivante, et détaillés dans les parties suivantes.





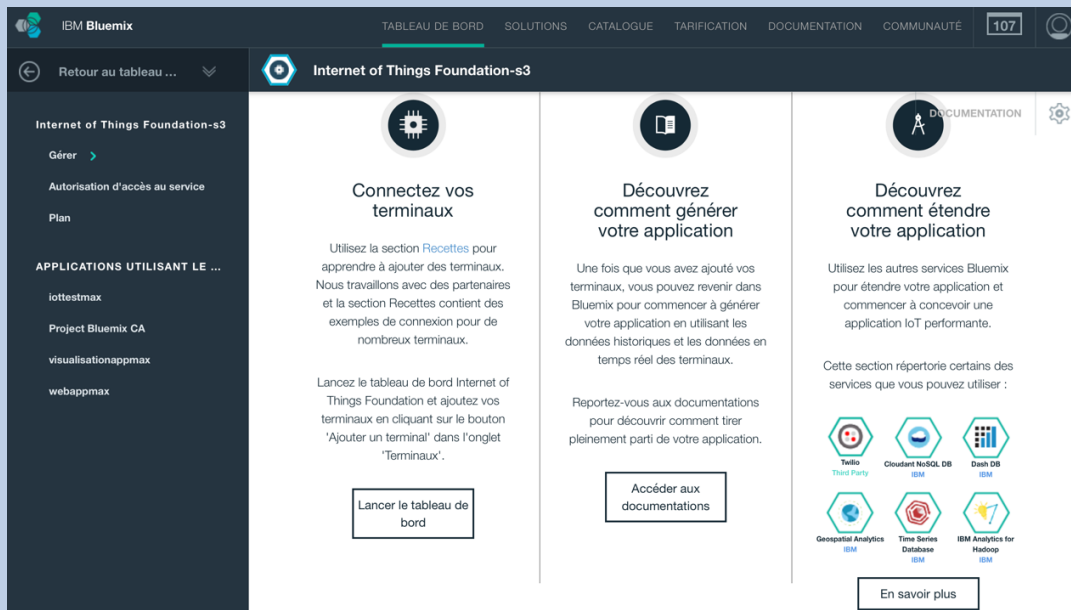
*Figure 7: Services Bluemix*

## B. Internet of things

### 1. Création d'un terminal et configuration du compte

How-to :

1. Ouvrir le Navigateur internet, puis se connecter à Bluemix.
2. Ouvrir le Tableau de Bord de Bluemix, puis ajouter le service « **IoT Foundation** ».
3. Sélectionner ce service, puis lancer le **tableau de bord**



4. Dans l'onglet « **Terminaux** », ajouter un terminal, puis créer un nouveau type de terminal. Le nom choisi est « **raspi2monitor** », et son identifiant est euro001

#### Ajouter un terminal

Sélection d'un type de terminal

Sélection d'un type de terminal

Or

Créer un type de terminal

5. Dans l'onglet « **Accès** », ajouter une clef d'API. ATTENTION, celle-ci permettra aux applications de se connecter, et le token ne pourra pas être obtenu plus tard. Il faut donc sauvegarder précieusement cette clef. Nous utiliserons toutes ces données pour connecter et authentifier nos applications.

## 2. Principe du service IoT Foundation

### Présentation générale :

L'IoT Foundation est un cloud auquel nos terminaux – quel qu'en soit le nombre ou leur simplicité – peuvent se connecter et communiquer à tout instant. La figure 8 montre le rôle de ce service dans notre réseau connecté.

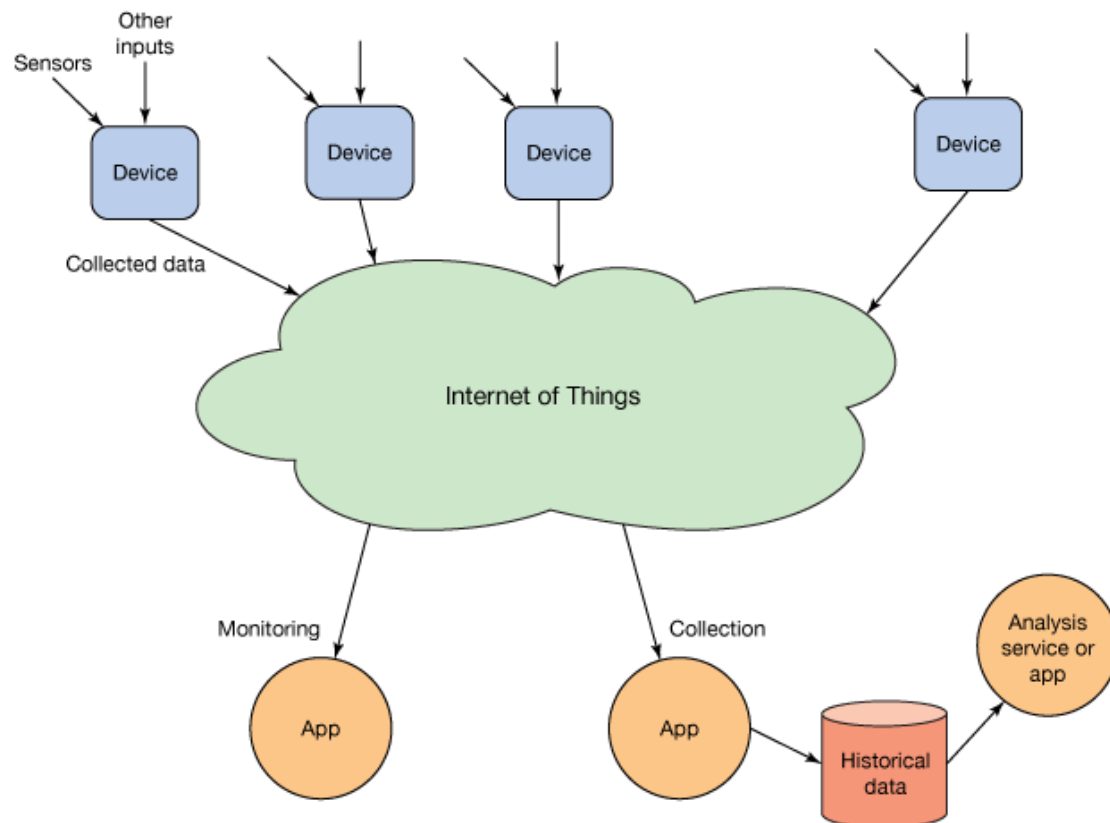


Figure 8: Iot Foundation

Ce service offre par exemple :

- Une multitude de proxys, brokers, et serveurs qui sont connectables via internet. Si notre terminal est connecté à internet, il peut toujours communiquer avec l'IoT foundation.
- Pour les terminaux qui envoient des données, on dispose d'un stockage de données pratiquement illimité, que l'on peut ensuite analyser et traiter avec d'autres services Bluemix
- Une authentification et un accès sécurisé pour chaque application et terminal.
- L'accès rapide à l'écosystème Bluemix
- Des outils et templates facilitant la création rapide d'application de monitoring, de contrôle à distance pour l'internet des objets.

L'IoT foundation consiste en un réseau MQTT sécurisé, augmenté de services pouvant être accédé via HTTP (en utilisant REST). L'avantage de MQTT est la simplicité de la sémantique de publication/abonnement des messages échangés via TCP. L'avantage indéniable de ce protocole est donc sa simplicité.

### Connexion au réseau MQTT de l'IOT Foundation :

MQTT est construit sur le protocole TCP. MQTT peut utiliser la connexion TCP ou TLS. L'utilisation avec TLS se fait comme suit :

```
tls://nom_organisation.messaging.internetofthings.ibmcloud.com:8883
```

TLS a la particularité d'implémenter une encryption end-to-end pour les connexions. adds end-to-end encryption to your connections. Les données transmises entre les terminaux et la fondation IoT sont cryptés de la même manière que pour les connections HTTPS.

### Espace des topics de l'IoT Foundation :

En MQTT générique, un client (que cela soit un terminal ou une application) peut publier ou s'abonner à n'importe quel topic sans restriction. Un topic est juste une chaine de texte à n'importe quel format. Chaque message publié à un topic est envoyé à tous ses abonnés.

La fondation IoT impose une restriction des espaces des topics, différent entre les applications et les terminaux, afin de sécuriser les terminaux face aux fuites de données, et d'éviter un terminal d'accéder aux données d'un autre terminal. Quand un terminal publie ou s'abonne à un topic MQTT, il a accès a un espace différent que celui des applications.

On peut voir dans la figure 9 les capacités asymétriques entre les terminaux et les applications.

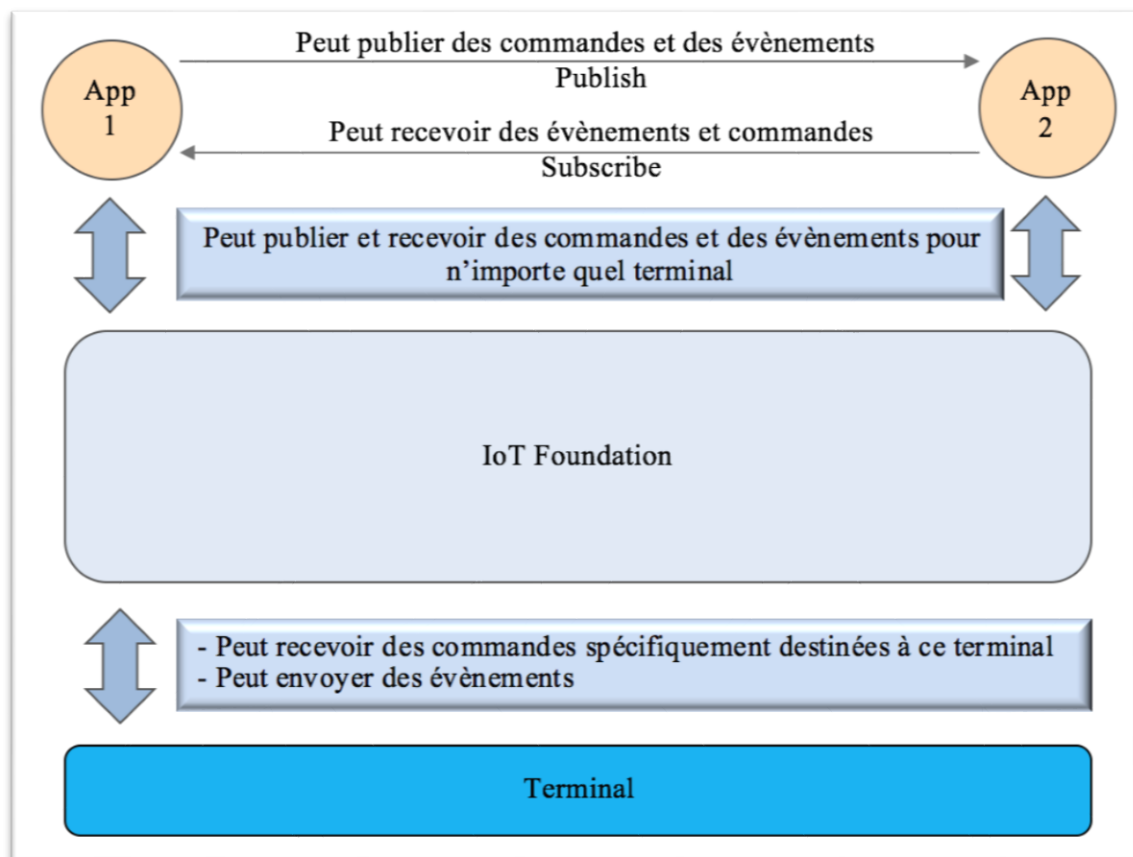


Figure 9: Différentes possibilités applications/terminaux

En effet, les applications authentifiées peuvent envoyer des commandes ou recevoir des événements de n'importe quelle application ou terminal de l'organisation, alors qu'un terminal authentifié peut uniquement s'abonner aux commandes lui étant destinées, et envoyer des événements en son nom.

Dans l'IoT Foundation, la différenciation des commandes se fait en utilisant des filtres. Par exemple, un topic pour qu'une application s'abonne au terminal de type rasp2monitor et d'identifiant euro001 sera le suivant :

```
iot-2/type/rasp2monitor/id/euro001/evt/itemsvc/fmt/json
```

Ici, le type et l'id du terminal est explicitement statué. L'IoT foundation permet aux applications d'utiliser des jokers ('+') pour s'abonner aux événements issus de multiples terminaux. Ainsi, la commande suivante permet de s'abonner aux abonnements originaires de tous les terminaux.

```
iot-2/type/+id/+evt/itemsvc/fmt/json
```

Une commande destinée à un terminal, envoyé depuis une application, peut ressembler à :

```
iot-2/type/rasp2monitor/id/euro001/cmd/operation/fmt/json
```

Pour la publication d'un événement depuis un terminal, il n'est pas nécessaire de spécifier l'id ou le type du terminal, vu que l'IoT Foundation est déjà au courant de ceux-ci.

Ainsi, la publication d'un événement se fera simplement ainsi :

```
iot-2/evt/itemsvc/fmt/json
```

Et l'abonnement à une commande destinée à ce terminal :

```
iot-2/cmd/operations/fmt/json
```

### Format des messages de la fondation IoT :

En MQTT, les messages envoyés à un topic peuvent être des chaînes sous n'importe quel format. Avec la fondation IoT, nous pouvons faire de même, mais un format est préférable, exposé ci-après.

L'encodage se fait au format UTF-8, de type JSON, avec deux champs principaux. Le premier regroupera les données, et le second le timestamp. Un exemple de format sera donc :

```
{ "d":{ "capteur1":23, "capteur2":"1015"}, "ts": "2015-12-01T03:41:50.341Z" }
```

### Authentification du client :

La Raspberry Pi est utilisée comme un client MQTT, qui s'authentifie au service IoT foundation via un token récupéré lors de la création des APIs.

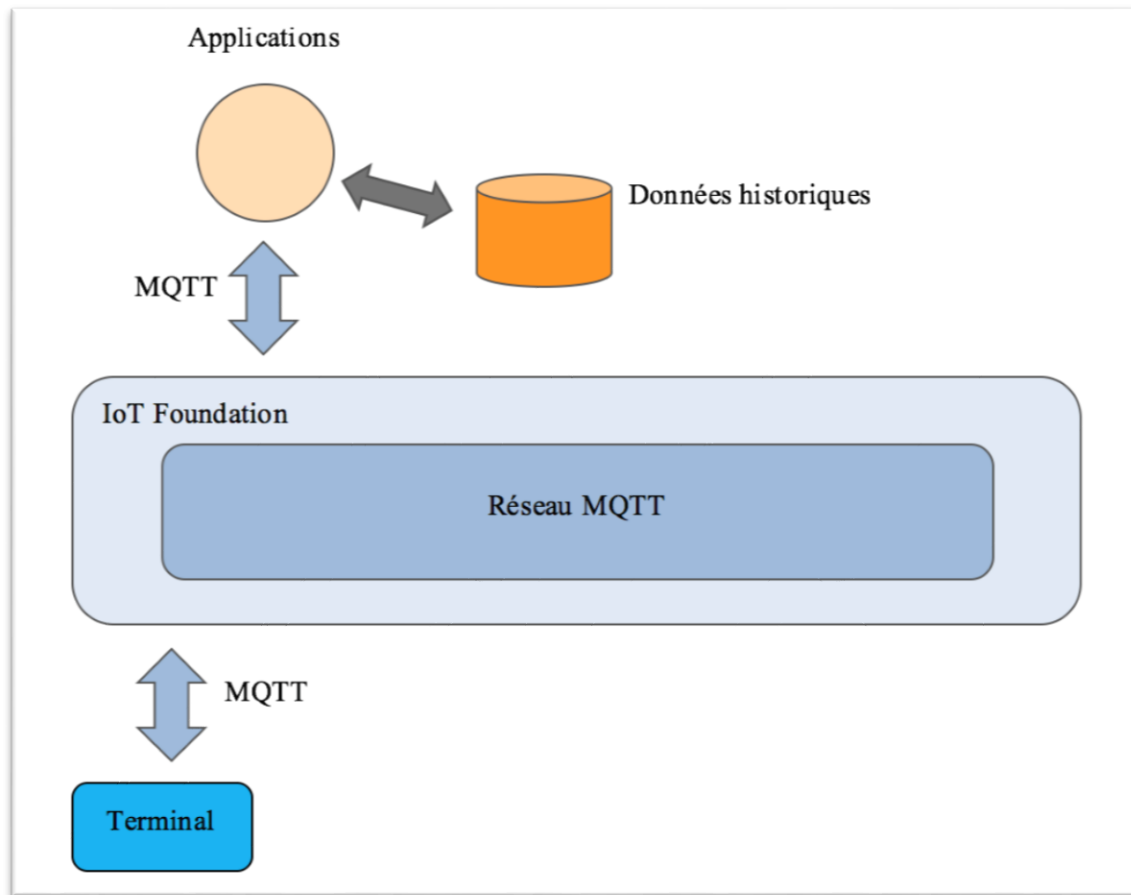


Figure 10: Résumé de l'IoT foundation

### **C. Base de données DB2**

Nous avons utilisé DB2 qui est l'un des systèmes de gestion de base de données propriétaire d'IBM.

Par l'intermédiaire du service SQL Database proposé gratuitement sous certaines conditions, nous avons implémenté les tables nécessaires au bon fonctionnement du projet. Ces tables permettent de stocker toutes les valeurs mesurées par les capteurs.

Dans un premier temps, il faut implémenter toutes les tables qui nous seront nécessaires. Dans notre cas, nous avons besoin de 3 tables : CAPTEUR, TYPE\_CAPTEUR et MESURE. Ces dernières sont totalement liées.

La première table, CAPTEUR, permet d'enregistrer tous les capteurs déployés dans notre projet. Elle est constituée d'un numéro d'identifiant et d'un nom.

### Création de la table CAPTEUR :

```
CREATE TABLE CAPTEUR (
    ID_CAPTEURS INT NOT NULL GENERATED ALWAYS AS IDENTITY
(START WITH 1 INCREMENT BY 1),
    NOM_CAPTEUR VARCHAR (30),
    PRIMARY KEY(ID_CAPTEUR)
);
```

La deuxième table, TYPE\_CAPTEUR, permet de préciser pour chaque capteur préalablement enregistré, sa nature. Cette table est constituée d'un numéro d'identifiant, d'un numéro de capteur prédéfini et d'un nom de fonction.

Création de la table TYPE\_CAPTEUR :

```
CREATE TABLE TYPE_CAPTEUR (
    ID_TYPE_CAPTEUR INT NOT NULL GENERATED ALWAYS AS
IDENTITY (START WITH 1 INCREMENT BY 1),
    CAPTEUR INT NOT NULL,
    TYPE VARCHAR (30),
    PRIMARY KEY(ID_TYPE_CAPTEUR),
    CONSTRAINT contrainte FOREIGN KEY (CAPTEUR) REFERENCES
USER11069.CAPTEURS (ID_CAPTEUR)
);
```

Enfin, la troisième et dernière table, MESURE, permet d'enregistrer l'ensemble des valeurs mesurées par l'ensemble de nos capteurs. Cette table est formée d'un numéro d'identifiant, d'un numéro correspondant au type de capteur, d'un champ valeur et d'un horodatage.

Création de la table MESURES :

```
CREATE TABLE MESURES (
    ID_MESURE INT NOT NULL GENERATED ALWAYS AS IDENTITY
(START WITH 1 INCREMENT BY 1),
    TYPE_CAPTEUR INT NOT NULL,
    VALEUR VARCHAR (30),
    TS TIMESTAMP NOT NULL WITH DEFAULT CURRENT TIMESTAMP,
    PRIMARY KEY(ID_MESURE),
    CONSTRAINT contrainte2 FOREIGN KEY (TYPE_CAPTEUR)
REFERENCES USER11069.CAPTEURS (ID_TYPE_CAPTEUR)
);
```

Voici un schéma récapitulatif qui permet de montrer le lien entre les différentes tables.

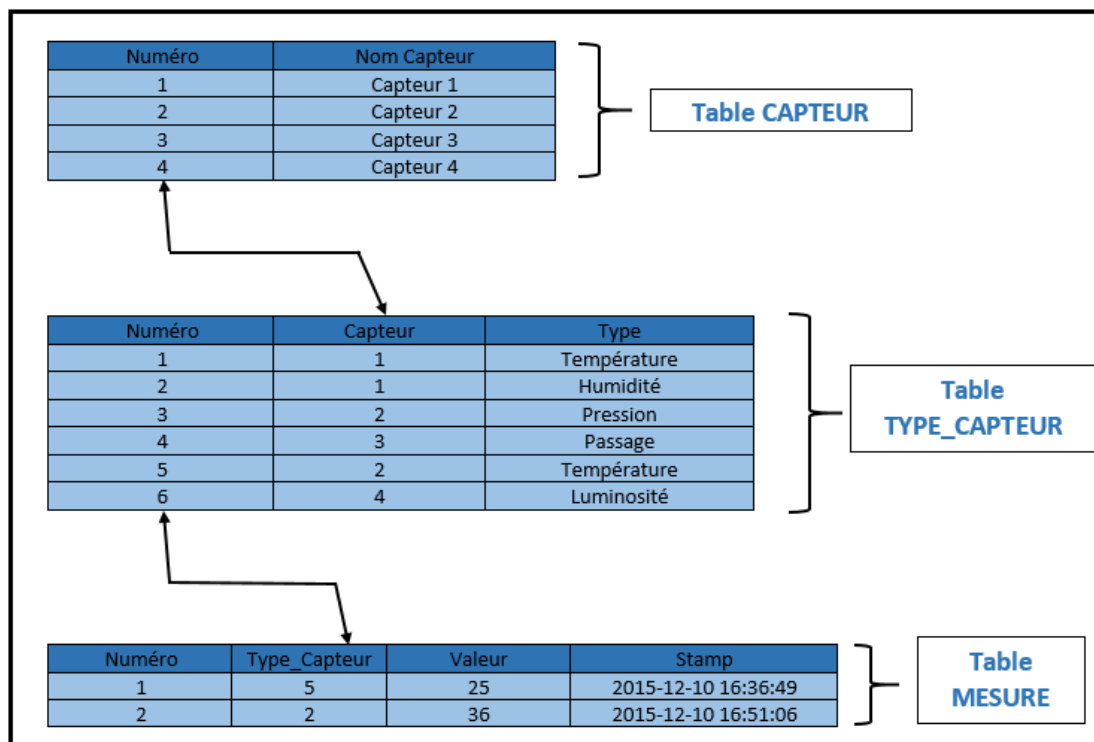


Figure 11: agencement des tables

Nous avons choisi ce type d'agencement de table pour pouvoir, par la suite, manipuler de façon optimale toutes les données insérées. Par exemple, il est facile de ne récupérer que les mesures qui correspondent à tel capteur ou bien de ne vouloir utiliser que les mesures de température par l'ensemble de nos capteurs de température.

Enfin, en regardant la première mesure de notre table 'MESURE', on peut voir que notre mesure est accompagnée du chiffre 5 qui correspond au type capteur. Grâce à une table de référence (qui permet de comprendre plus rapidement l'agencement des 3 tables), nous pouvons voir que cette mesure est du type « Température » prise par le capteur 2.

Valeur « Type_Capteur »	Numéro du capteur	Nature de la mesure
1	1	Température
2	1	Humidité
3	2	Pression Atmosphérique
4	3	Passage
5	2	Température
6	4	Luminosité

Figure 12: table de référence

Normalement, cette base est gérée par le système client-base de données par l'intermédiaire du service de sms, Twilio. L'utilisateur peut, en effet, à partir d'un échange de sms, lire ou modifier la base de données. Ces fonctionnalités vous seront présentées dans la section « Twilio » dans ce tutoriel.



Mais il se peut que, pour une raison ou une autre, vous soyez contraints à modifier la base de données depuis la plateforme Bluemix. Voici les étapes pour pouvoir modifier cette base de données :

Note : toutes les sections de code qui suivent doivent être situées dans le fichier « app.js ».

## 1. Ouvrir la base de données

```
ibmdbContext.open("DATABASE=SQLDB;HOSTNAME=75.126.155.153; UID=user11069;
PWD=gFge18eRNESG", function(err){
    if (err) {
        console.log(err);
        return;
    }
    // ... Suite du code
});
```

Les arguments qui se trouvent dans la fonction « ibmdbContext.open » permettent d'identifier le service souhaité. En effet, grâce au nom de la base, du numéro « Hostname », du numéro de l'utilisateur l'ayant créé et du mot de passe, vous pouvez vous connecter à la base et donc l'ouvrir. A partir de ce moment, vous pouvez effectuer tous les changements nécessaires pour la base.

## 2. Modifier la base de données

Vous pouvez effectuer plusieurs types de modification dans votre base comme, ajouter un champ, supprimer une table, etc.

```
ibmdbContext.query("LA REQUETE QUE VOUS SOUHAITEZ EFFECTUER",
function(err){
    if (err)
    {
        console.log("Interdit de rentrer dans cette boucle");
        console.log(err);
        return;
    }
});
```

A partir de ce bout de code, vous pouvez apporter des modifications dans cette base. Pour cela, il suffit de reprendre des commandes DB2.

Voici quelques exemples de commandes existantes :

ACTION	COMMANDE
Supprimer les valeurs d'une table	DELETRE FROM <i>nom_base</i>
Insérer une valeur dans une table	INSERT INTO <i>nom_base</i> ( <i>nom_champ</i> ) VALUES ( <i>valeur</i> )
Créer une table	CREATE TABLE <i>nom_table</i> ( <i>paramètres_table</i> );

### 3. Fermer la base de données

```
ibmndb.close(function (err) {  
    if (err) {  
        return console.log(err);  
    }  
    console.log('Base de données fermée');  
});
```

La base de données est désormais fermée. Plus aucune requête ne peut être effectuée sur notre base. Il faudra la ré-ouvrir si vous souhaitez apporter d'autres modifications.

*Note : à noter que vous pouvez effectuer plusieurs « actions » sur cette base de données dans le fichier « app.js » si ces derniers se situent entre le code correspondant à l'ouverture de la base de données et à sa fermeture.*

## D. Service sms Twilio

Dans cette partie nous nous intéresserons au service permettant l'envoi et la réception des messages via le module GSM d'un téléphone portable. Il faut savoir que l'objectif de cette partie est de proposer à l'utilisateur un outil sophistiqué qui permettra de suivre en temps réel, l'évolution du système (valeurs de différents capteurs). Cette fonctionnalité donne une grande valeur ajoutée à notre système dans la mesure où l'utilisateur se trouve à l'extérieur d'un lieu où le système est installé et fait de l'acquisition continue des données et les communique au serveur. Pour cela nous utiliserons le système Twilio qui donne la possibilité d'envoyer et de recevoir des messages textuels via le GSM.

Le service Twilio est un cloud de communication basé à San Francisco en Californie. Ses outils de développement proposés en ligne donnent la possibilité aux utilisateurs d'émettre et de recevoir des appels téléphoniques, des messages textuels. Pour bien accéder à ces services, Twilio utilise les protocoles HTTPS.

Nous allons voir dans la suite comment utiliser ce service et l'intégrer dans notre système.

Twilio fait partie des services proposées par Bluemix. Pour cela, nous allons voir comment ajouter ce service à notre application serveur déployée sur la plateforme de Bluemix.

### Inscription à Twilio :

La première étape incontournable avant de commencer quoique soit est de s'inscrire sur le service Twilio. L'inscription à ce service est essentielle. Sans cette étape, nous ne pourrions utiliser ce service dans notre application Bluemix. Après l'inscription, deux informations importantes seront à communiquer plus tard au service Bluemix au moment de son ajout à celle-ci.

1. Pour cela, rendez-vous sur le site <https://www.twilio.com/>.
2. Aller sur la page d'inscription en appuyant sur le bouton Sign Up. Des informations telles que le nom et prénom, l'adresse de mail ainsi que le mot de passe sont nécessaires pour la création de votre compte Twilio.



The image shows the Twilio sign-up form. It has a heading "It's fast and free to get started." followed by input fields for "First Name", "Last Name", "Company Name (optional)", and "Email". Below these are two fields for "Choose a password" and "Password, again". There are two dropdown menus: "What are you building?" and "Choose your language", both with "Please select..." as the placeholder. At the bottom is a red "Get Started" button. To the right of the button is the text "By clicking the button, you agree to our [legal policies](#)." Below the button is the text "Already have an account? [Login](#)".

3. Après avoir validé le mail de confirmation qui vous a été envoyé, rendez-vous dans votre espace utilisateur. Dans la page d'accueil, appuyer sur le bouton « Show API Credentials ». Veuillez noter ces informations quelque part, vous en aurez besoin plus tard pour le service Bluemix comme cela a été expliqué précédemment.

## Get Started with Phone Numbers

The image is a screenshot of the Twilio "API Credentials" page. At the top right, there is a link "Hide API Credentials" with a small icon to its left. Below this, the heading "API Credentials" is followed by the text "To use the Twilio API you will need your AccountSid and Auth Token." Below this text is a table with two columns: "ACCOUNT SID" and "AUTH TOKEN". The "ACCOUNT SID" value is "AC39bb08818107af9f9fc1463dc669ab2b" and the "AUTH TOKEN" value is "71075ec024be7540aa6d2773db0ab4ef". There are two large black arrows pointing to the "AUTH TOKEN" value, one from the left and one from the right.

4. Dans la rubrique « Products », cliquer sur la catégorie « PHONE NUMBERS ».
5. Appuyer ensuite sur le bouton « Buy a number ». Cette étape vous permettra de vous munir d'un numéro de téléphone. Il faut savoir que Twilio vous proposera des numéros compatibles service appel et sms. Faites très attention à ne pas choisir un numéro qui n'est pas capable d'émettre et de recevoir des appels.
6. Choisissez votre pays, appuyer sur le bouton sms et cliquer sur le bouton « Search ». Une liste de numéros disponibles vous sera proposée, veuillez en choisir un et appuyer sur le bouton « Buy ». Il ne faut pas penser que le numéro est payant. Le premier numéro est gratuit donc il suffit de cliquer sur le bouton Buy pour finaliser l'attribution d'un numéro.

## **Configuration de Twilio :**

Il faut maintenant procéder à la configuration du service Twilio. Pour cela regardons un peu plus en détails le fonctionnement du service de messagerie de Twilio.

Quand un message est envoyé au numéro fourni à Twilio, celui-ci envoie une requête http vers un URL. Cette requête permettra de récupérer l'action définie à cette adresse et la renverra vers le numéro qui lui a envoyé un message. Ainsi vous allez devoir vous diriger vers la page qui spécifiera cette adresse URL.

Pour cela :

1. Appuyer sur le bouton PHONE NUMBER dans la rubrique PRODUCTS comme précédemment.
2. Cliquer ensuite sur votre numéro de téléphone. En appuyant sur ce bouton, vous serez dirigés vers la page où vous préciserez l'adresse URL de réponse.
3. Cochez la case TwiML App et cliquer sur Create a new TwiML App afin de définir une application TwiML App.
4. Dans le champ Request URL dans la partie Messaging, mettez l'adresse de l'URL où sera mis en ligne une application qui récupèrera la requête et fera le traitement adapté selon la nature de la requête reçue. Vous choisirez le mode HTTP POST.
5. Appuyer sur « save ».
6. Revenir dans la page PRODUCTS -> PHONE NUMBERS
7. Cliquer sur votre numéro de téléphone Twilio, sélectionner la TwiML App déjà créée.

## **Ajouter et configurer le service Twilio dans Bluemix :**

Une fois cette configuration préalable faite, nous rajoutons ce service à notre application Bluemix. Ainsi le fichier manifest.yml se mettra à jour automatiquement.

1. Aller sur le Dashboard de Bluemix et cliquer sur « ajouter un service ».
2. Parmi tous les services proposés, sélectionner Twilio et appuyer sur le bouton create.
3. Paramétrer le nom si voulu, et entrer les ACCOUNT SID et AUTH TOKEN qui nous ont été communiqués dans la table d'API Credentials. (cf plus haut).

Une fois que l'application a été créée et ajoutée à la plateforme Bluemix, nous allons devoir modifier l'application node.js dans laquelle nous définirons le comportement de notre application en cas de sollicitation de l'URL par une requête HTTP.

## V. Bibliographie

<https://www.npmjs.com/> :

- Documentation sur tous les packages node

<https://openclassrooms.com/courses/des-applications-ultra-rapides-avec-node-js>

- Prise en main de Nodejs

<http://support.connectblue.com/display/PRODBTSPA/Bluetooth+Low+Energy+Serial+Port+Adapter+-+Getting+Started>

- Instructions et éclaircissements sur le BLE

<https://developer.ibm.com/iotfoundation/recipes/api-documentation/>

- Documentation sur Bluemix et MQTT