

Nouveaux usages de l'Internet global

Rapport :

L'objectif commun de chacune des parties est de réaliser une petite station qui permet de renvoyer des mesures de température, d'humidité et de luminosité. Pour effectuer ces mesures, nous disposons d'un Arduino, d'un capteur de luminosité et d'un capteur DHT (température et humidité). Le capteur de luminosité fonctionne simplement en analogique, il renvoie une valeur qui augmente en fonction de l'intensité lumineuse. Le capteur DHT, connecté sur un port digital, fonctionne avec la bibliothèque **DHT-sensor-library** (1) et utilise ses propres fonctions pour récupérer les valeurs.

La première partie consiste à réaliser un serveur CoAP à l'aide de l'Arduino. CoAP est un protocole de la couche applicative du modèle OSI qui repose sur UDP. C'est un protocole qui est adapté pour l'Internet des objets notamment car il permet d'économiser de l'énergie en envoyant moins de donnée que d'autres protocoles plus conventionnel.

On a commencé par trouver une bibliothèque pour l'Arduino qui permettrait de traiter les requêtes CoAP et de renvoyer les valeurs que l'on souhaitait récupérer. Nous avons utilisé pour cela la bibliothèque **CoAP-simple-library** (2) et nous sommes parti de l'exemple fourni « coapserver ». La connexion entre l'Arduino et l'ordinateur se fait grâce à un câble Ethernet. On a donc aussi utilisé la bibliothèque Ethernet qui est directement implémentée dans Arduino. Pour utiliser la bibliothèque CoAP, il faut implémenter des fonctions de callback qui seront appelées en fonction du paramètre que le client aura donné. Ici, on veut pouvoir récupérer au choix la température, la luminosité ou bien l'humidité. On crée donc trois fonctions qui correspondent à ce que l'on peut envoyer.

Pour la luminosité, on lit ce que donne le capteur de luminosité sur la sortie analogique. En fonction de la valeur récupérée, on renvoie la chaîne de caractère « Jour », « Soir », ou « Nuit ». Pour la température et l'humidité, les deux fonctions de callback utilisent la bibliothèque DHT associée au capteur que l'on utilise. Elle renvoie directement une valeur en °C ou en %.



Fig : Schéma représentant les connexions entre les différents appareils utilisés lors de la première partie

La connexion Ethernet en elle-même est établie en donnant l'adresse MAC de l'interface associée à l'ordinateur et en ajoutant une adresse IP pour l'interface de l'Arduino. Il suffit alors de rajouter une adresse dans le même sous réseaux sur l'interface de l'ordinateur, ici 192.168.0.0/30. Une fois la connexion établie, il suffit de pouvoir envoyer des requêtes en CoAP depuis le PC. On utilise pour cela « coap-client » qui permet d'envoyer ces requêtes depuis un terminal.

```

info@i005pc22: ~
Fichier Édition Affichage Rechercher Terminal Aide

-f file          non-ASCII characters)
-m method        file to send with PUT/POST (use '-' for STDIN)
-N              request method (get|put|post|delete), default is 'get'
-o file          send NON-confirmable message
-p port          output received data to this file (use '-' for STDOUT)
-s duration      listen on specified port
-v num           subscribe for given duration [s]
-V num           verbosity level (default: 3)
-O num,text      add option num with contents text to request
-P addr[:port]   use proxy (automatically adds Proxy-Uri option to
                  request)
-T token         include specified token
-U              never include Uri-Host or Uri-Port options

examples:
coap-client -m get coap://[::1]/
coap-client -m get coap://[::1]/.well-known/core
coap-client -m get -T cafe coap://[::1]/time
echo 1000 | coap-client -m put -T cafe coap://[::1]/time -f -

info@i005pc22:~$ coap-client coap://
Dec 04 14:13:50 ERR invalid CoAP URI
info@i005pc22:~$ coap-client coap://192.168.0.1/temperatur
v:1 t:CON c:GET i:bf1b {} [ ]
^Z
[1]+  Arrêté          coap-client coap://192.168.0.1/temperatur
info@i005pc22:~$ coap-client coap://192.168.0.1/temperature
v:1 t:CON c:GET i:d751 {} [ ]
Temp: 25 °C
info@i005pc22:~$ coap-client coap://192.168.0.1/lumiere
v:1 t:CON c:GET i:89ea {} [ ]
Jour
info@i005pc22:~$ coap-client coap://192.168.0.1/humidite
v:1 t:CON c:GET i:22ff {} [ ]
Humidite: 36%
info@i005pc22:~$
  
```

Fig : résultat obtenu en demandant certaines valeurs à l'aide du terminal

Pour la deuxième partie, nous avons comme objectif de mettre en place un serveur proxy à l'aide d'un Raspberry qui servirait d'intermédiaire entre l'Arduino et l'ordinateur. Nous devons conserver une connexion CoAP entre l'Arduino et le proxy mais établir une connexion HTTP entre l'ordinateur et le proxy. L'ordinateur enverra grâce à un navigateur web des requêtes en HTTP directement avec l'URL. Le proxy qui tournera sur le Raspberry traitera ces requêtes puis fera à son tour les requêtes à l'Arduino en CoAP. L'Arduino pourra alors envoyer les mesures faites à partir des requêtes initiales, et les informations feront le chemin inverse jusqu'à s'afficher sur une page web.

Nous devons déjà choisir une stratégie de connexion : nous devons en faire une en Wi-Fi et une en Ethernet. Utiliser le Wi-Fi entre l'Arduino et le Raspberry semble être le plus simple car les ordinateurs que l'on utilise n'ont pas de carte Wi-Fi. Il existe plusieurs bibliothèques Arduino qui permettent d'utiliser le Wi-Fi. Nous utiliserons dans la suite la bibliothèque **Wi-Fi101** (3) à chaque fois que nous utiliserons le Wi-Fi avec l'Arduino. Le point d'accès Wi-Fi sera celui d'un téléphone en partage de connexion.

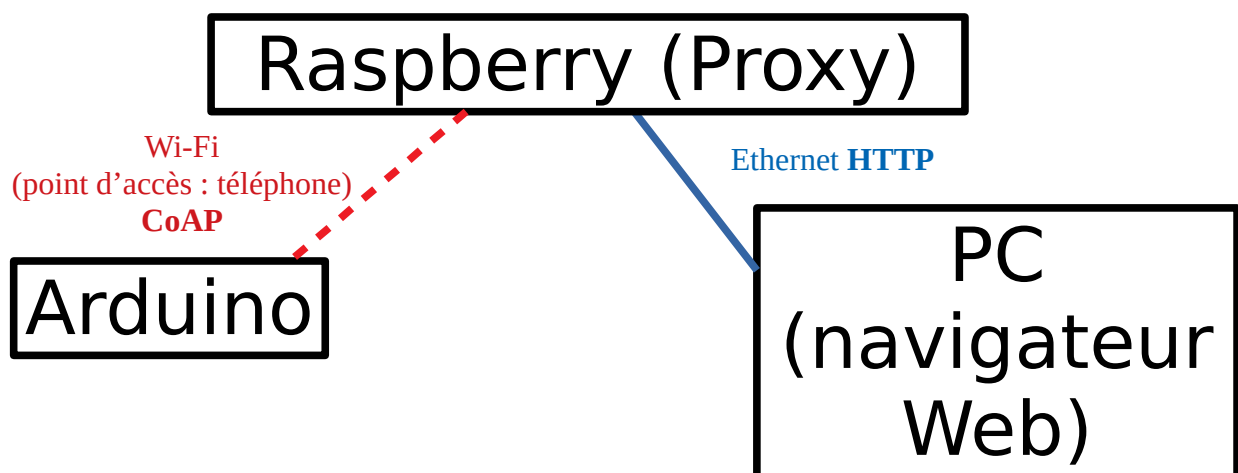


Fig : Schéma représentant la solution choisie pour les connexions entre les différents appareils utilisés lors de la deuxième partie

Même si le code Arduino reste le même dans le principe, il y a quand même plusieurs choses à changer. Notamment pour la connexion : ici, plus besoin de rentrer manuellement une adresse IP. Il suffit de donner le nom et le mot de passe d'un serveur Wi-Fi détectable. A part cela, le code utilisé est la fusion entre l'exemple de la bibliothèque Wi-Fi101 « Wi-FiWebServer » et le code de la partie ci-dessus.

The screenshot shows the Arduino IDE interface. The top window displays the 'WiFiWebServer101' sketch, which includes code for connecting to a Wi-Fi network and sending a response. The serial monitor window, titled '/dev/ttyACM0 (Arduino/Genuino MKR1000)', shows the output of the sketch. The output includes the message 'Attempting to connect to SSID: ES_1134' and 'Starting endpoints...'. Below this, the serial monitor shows the IP address '192.168.43.58' and the status 'connected'. The bottom status bar indicates 'Téléversement terminé' (Upload finished) and 'Verify successful'.

```

WiFiWebServer101 | Arduino 1.8.7
Eichier Édition Croquis Outils Aide
WiFiWebServer101 arduino_secrets.h
}
else if(lum<=2)
  sprintf(resp
}
else{sprintf("Starting DHT sensor...
Starting endpoints...
if(packet.codeReady.
  Serial.printSSID: ES_1134
  coap.sendResIP Address: 192.168.43.58
}
connected
}

void setup() {
  //Initialize s
  Serial.begin(S
  while (!Serial
    ; // wait fo
}

// check for t
if (WiFi.statu Défilement automatique ☐ Show timestamp
  Serial.print
  // don't continue:
  while (true);
}

// attempt to connect to Wi-Fi network:
while (status != WL_CONNECTED) {
  Serial.print("Attempting to connect to SSID: ");
  Serial.println(ssid);
  // Connect to WPA/WPA2 network. Change this line if using open or WEP network:
  status = WiFi.begin(ssid, password);

  // wait 10 seconds for connection:
  delay(10000);
}

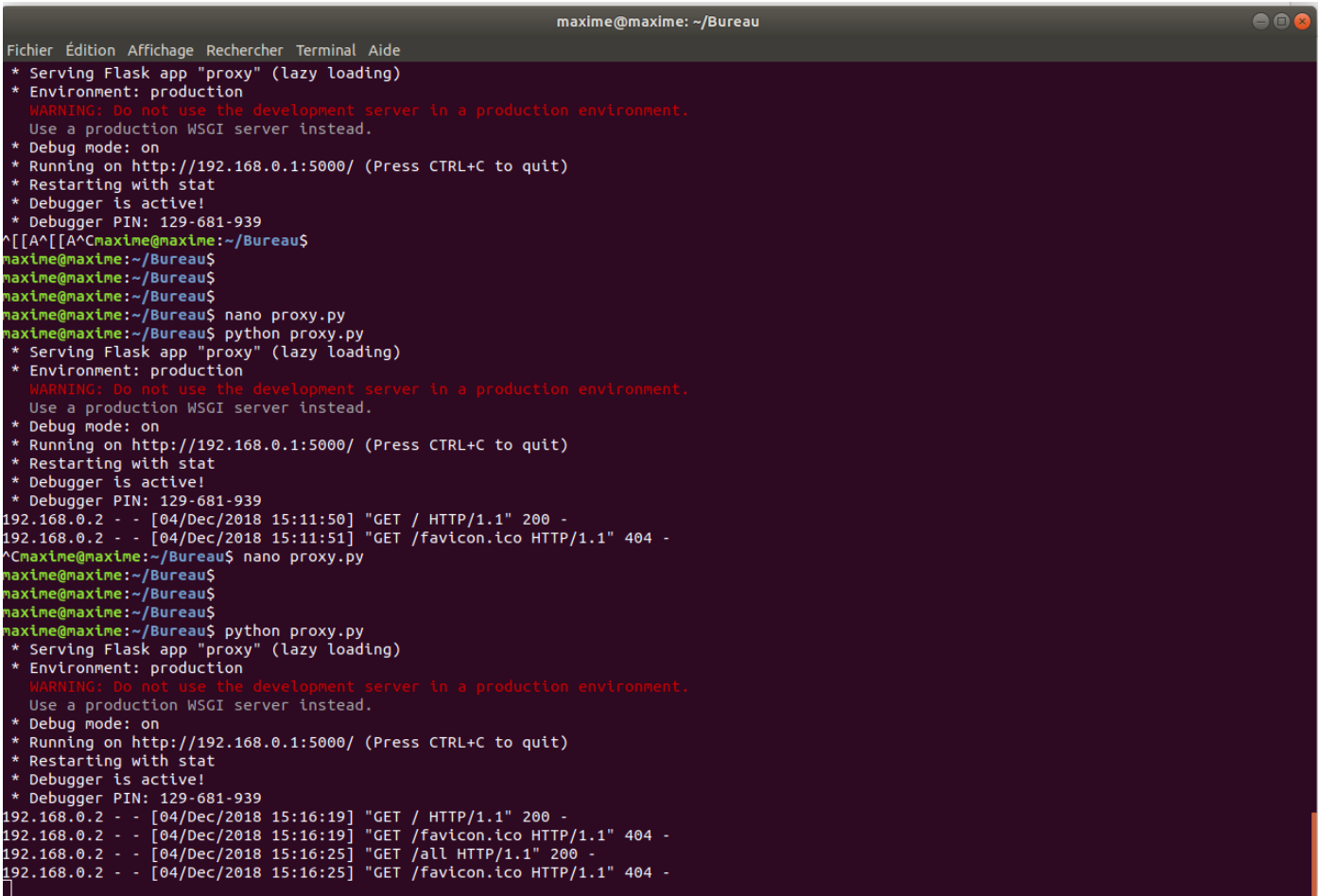
Téléversement terminé
Verify successful
done in 0.041 seconds
CPU reset.
144 Arduino/Genuino MKR1000 sur /dev/ttyACM0

```

Fig : connexion de l'Arduino au point d'accès Wi-Fi

Il faut maintenant mettre en place un proxy. Nous avons donc installé **UbuntuMate**, un système d'exploitation, sur le Raspberry. Utiliser un script Python est une solution relativement simple et efficace pour faire tourner un proxy. Pour configurer rapidement ce que l'on veut sur le Raspberry, nous nous connectons en SSH grâce à un câble Ethernet. Il faut installer plusieurs choses sur le Raspberry notamment Flask, une bibliothèque pour Python, qui permet de parser les URL que le proxy recevra.

On peut donc grâce à Flask, effectuer différentes actions dans le code Python en fonction de la requête HTTP reçu. On met en place cinq actions différentes : un test basique, une récupération de la température, de l'humidité et de la luminosité, et une qui récupère toutes les valeurs en une seule fois.



```
maxime@maxime: ~/Bureau
Fichier  Edition  Affichage  Rechercher  Terminal  Aide
* Serving Flask app "proxy" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://192.168.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 129-681-939
^[[A^[[A^Cmaxime@maxime:~/Bureau$
maxime@maxime:~/Bureau$
maxime@maxime:~/Bureau$
maxime@maxime:~/Bureau$ nano proxy.py
maxime@maxime:~/Bureau$ python proxy.py
* Serving Flask app "proxy" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://192.168.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 129-681-939
192.168.0.2 - - [04/Dec/2018 15:11:50] "GET / HTTP/1.1" 200 -
192.168.0.2 - - [04/Dec/2018 15:11:51] "GET /favicon.ico HTTP/1.1" 404 -
^Cmaxime@maxime:~/Bureau$ nano proxy.py
maxime@maxime:~/Bureau$
maxime@maxime:~/Bureau$
maxime@maxime:~/Bureau$ python proxy.py
* Serving Flask app "proxy" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://192.168.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 129-681-939
192.168.0.2 - - [04/Dec/2018 15:16:19] "GET / HTTP/1.1" 200 -
192.168.0.2 - - [04/Dec/2018 15:16:19] "GET /favicon.ico HTTP/1.1" 404 -
192.168.0.2 - - [04/Dec/2018 15:16:25] "GET /all HTTP/1.1" 200 -
192.168.0.2 - - [04/Dec/2018 15:16:25] "GET /favicon.ico HTTP/1.1" 404 -
```

Fig : démarrage du proxy (Flask) sur le Raspberry

Pour faire une requête, il suffit de rentrer dans la barre d'adresse du navigateur: <http://192.168.0.1:5000/> soit l'adresse IP de l'interface Ethernet du Raspberry suivi du numéro de port utilisé, et enfin le /<commande> que l'on souhaite : /, /temperature, /humidite, /luminosite ou /all.

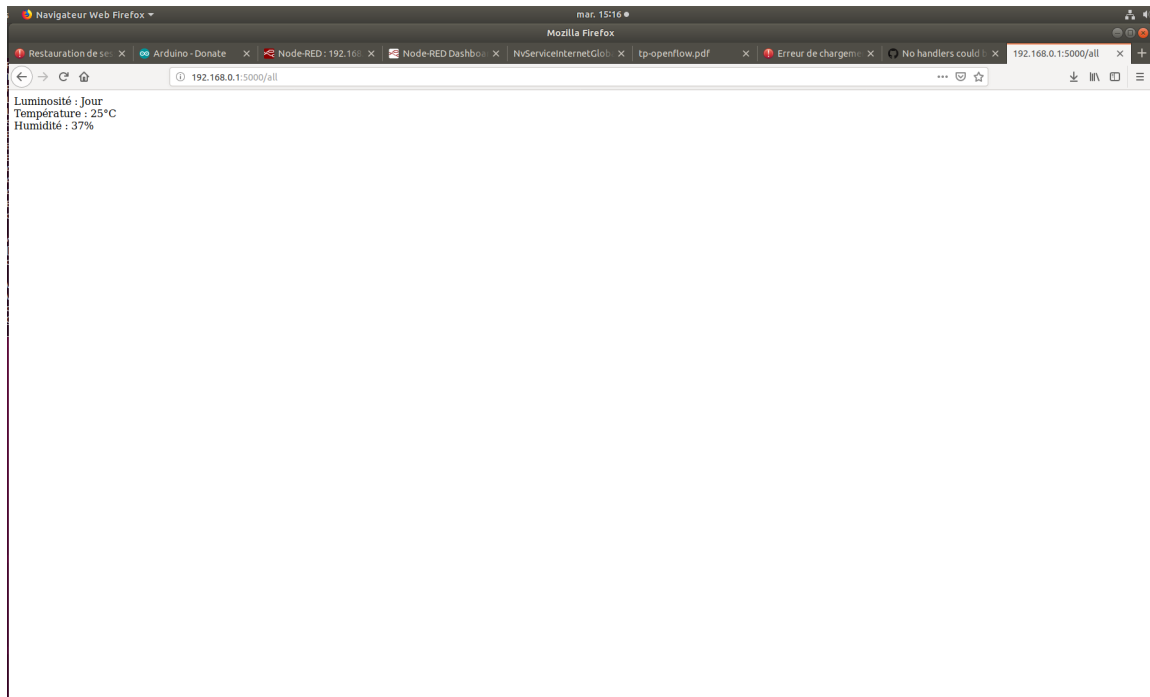


Fig : affichage du résultat de la demande « /all » dans le navigateur Web

Pour la dernière partie, l'objectif était d'utiliser le protocole MQTT pour transmettre les données à la place de CoAP. On garde cependant le même montage que la partie précédente : c'est-à-dire faire les requête depuis l'ordinateur vers l'Arduino en passant par le Raspberry et en gardant le même type de connexion. MQTT est différent de CoAP dans le sens où MQTT utilise un « broker » et un système de publication et de souscription.

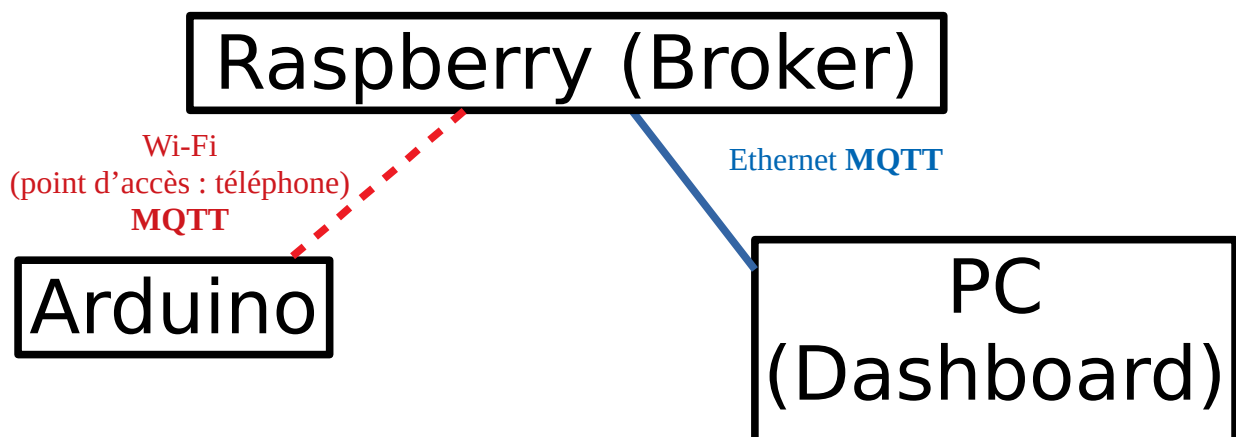
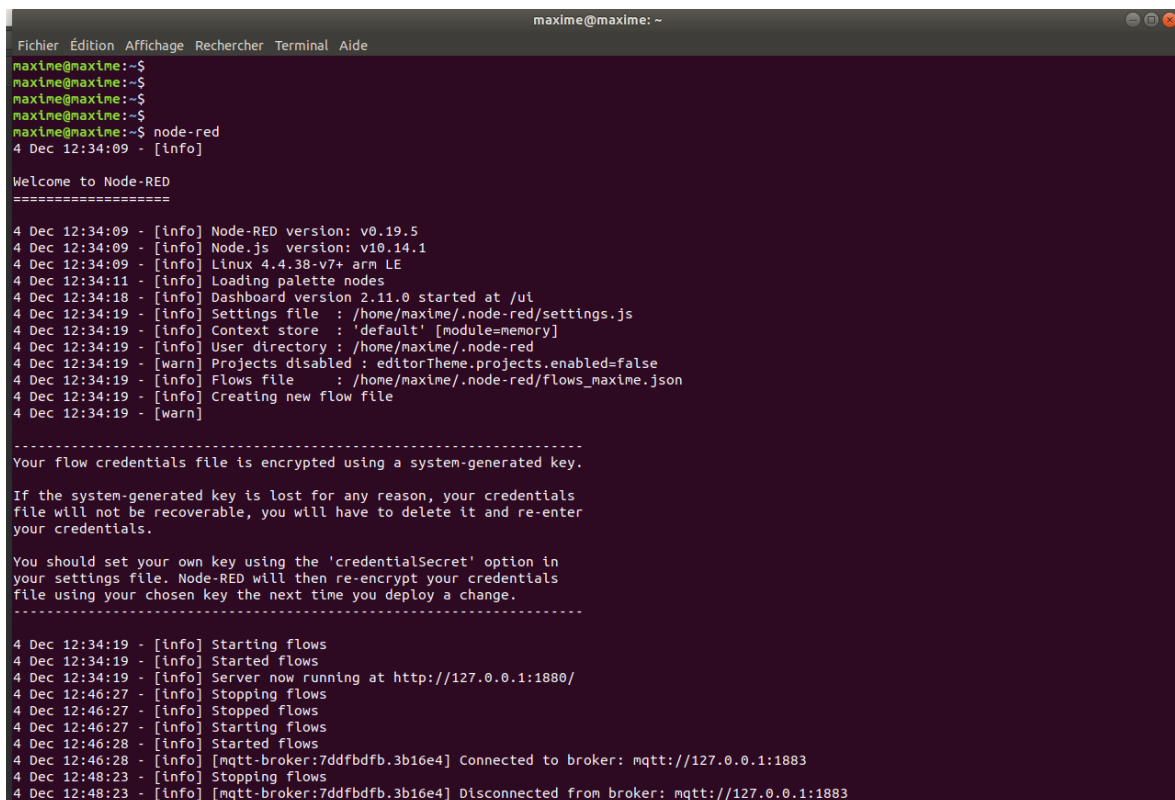


Fig : Schéma représentant la solution choisie pour les connexions entre les différents appareils utilisés lors de la troisième partie

A intervalle de temps régulier, l'Arduino va donc ici publier les données qu'il a mesuré vers le broker qui se trouve sur l'Arduino, et l'ordinateur pour recevoir ces données par le biais d'un abonnement aux topics dont il souhaite recevoir les informations.

Le broker installé sur le Raspberry est Mosquitto. Il suffit de l'installer et de le démarrer immédiatement. Le code Arduino est lui aussi encore modifié. On utilise cette fois-ci la bibliothèque **publishclient** (4) et la base de l'exemple « mqtt_esp8266 » que l'on trouve dans celle-ci. On réutilise les mêmes fonctionnalités pour le Wi-Fi que la partie prétendante. Le code utilise en plus de la fonctionnalité de publisher, celle de subscriber. On peut enlever toutes ces parties que ne nous sont pas nécessaire ici et changer certains paramètres comme le temps entre chaque publication de donnée : ici deux secondes pour pouvoir observer plus facilement les changements. On ajoute donc les trois fonctions que l'on utilise depuis le début qui correspondent à chaque valeurs et qui seront appelées à chaque boucle.

Pour la réception des données du côté de l'ordinateur, il faut maintenant utiliser un dashboard à la place de texte en HTML dans une simple page web. Pour cela, on peut utiliser Node-Red qui permet de traiter les flux de donnée que l'on reçoit. En l'installant sur le Raspberry, il suffit de l'exécuter pour que depuis l'ordinateur, toutes les configurations nécessaires soient être faites.



```
maxime@maxime: ~  
Fichier Edition Affichage Rechercher Terminal Aide  
maxime@maxime:~$  
maxime@maxime:~$  
maxime@maxime:~$  
maxime@maxime:~$  
maxime@maxime:~$ node-red  
4 Dec 12:34:09 - [info]  
  
Welcome to Node-RED  
=====
```

```
4 Dec 12:34:09 - [info] Node-RED version: v0.19.5  
4 Dec 12:34:09 - [info] Node.js version: v10.14.1  
4 Dec 12:34:09 - [info] Linux 4.4.38-v7+ arm LE  
4 Dec 12:34:11 - [info] Loading palette nodes  
4 Dec 12:34:18 - [info] Dashboard version 2.11.0 started at /ui  
4 Dec 12:34:19 - [info] Settings file : /home/maxime/.node-red/settings.js  
4 Dec 12:34:19 - [info] Context store : 'default' [module=memory]  
4 Dec 12:34:19 - [info] User directory : /home/maxime/.node-red  
4 Dec 12:34:19 - [warn] Projects disabled : editorTheme.projects.enabled=false  
4 Dec 12:34:19 - [info] Flows file : /home/maxime/.node-red/flows_maxime.json  
4 Dec 12:34:19 - [info] Creating new flow file  
4 Dec 12:34:19 - [warn]
```

```
-----  
Your flow credentials file is encrypted using a system-generated key.  
  
If the system-generated key is lost for any reason, your credentials  
file will not be recoverable, you will have to delete it and re-enter  
your credentials.  
  
You should set your own key using the 'credentialSecret' option in  
your settings file. Node-RED will then re-encrypt your credentials  
file using your chosen key the next time you deploy a change.  
-----
```

```
4 Dec 12:34:19 - [info] Starting flows  
4 Dec 12:34:19 - [info] Started flows  
4 Dec 12:34:19 - [info] Server now running at http://127.0.0.1:1880/  
4 Dec 12:46:27 - [info] Stopping flows  
4 Dec 12:46:27 - [info] Stopped flows  
4 Dec 12:46:27 - [info] Starting flows  
4 Dec 12:46:28 - [info] Started flows  
4 Dec 12:46:28 - [info] [mqtt-broker:7ddfbdfb.3b16e4] Connected to broker: mqtt://127.0.0.1:1883  
4 Dec 12:48:23 - [info] Stopping flows  
4 Dec 12:48:23 - [info] [mqtt-broker:7ddfbdfb.3b16e4] Disconnected from broker: mqtt://127.0.0.1:1883
```

Fig : Lancement de Node-Red à partir du Raspberry

On peut alors ouvrir une interface graphique dans un navigateur qui nous permet de manipuler les flux provenant de chacun des topics. L'interface est intuitive et on peut, en installant les bon plug-in, utiliser une fonctionnalité qui permet d'afficher des jauges :

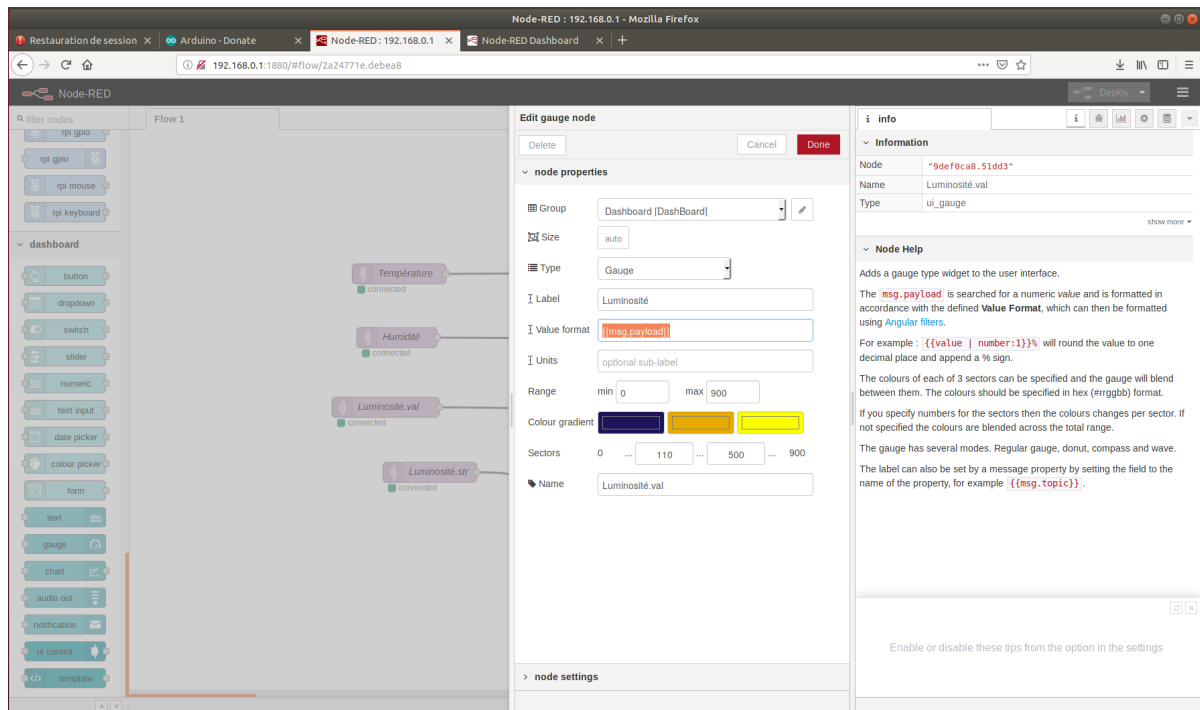


Fig : Configuration du dashboard depuis le PC

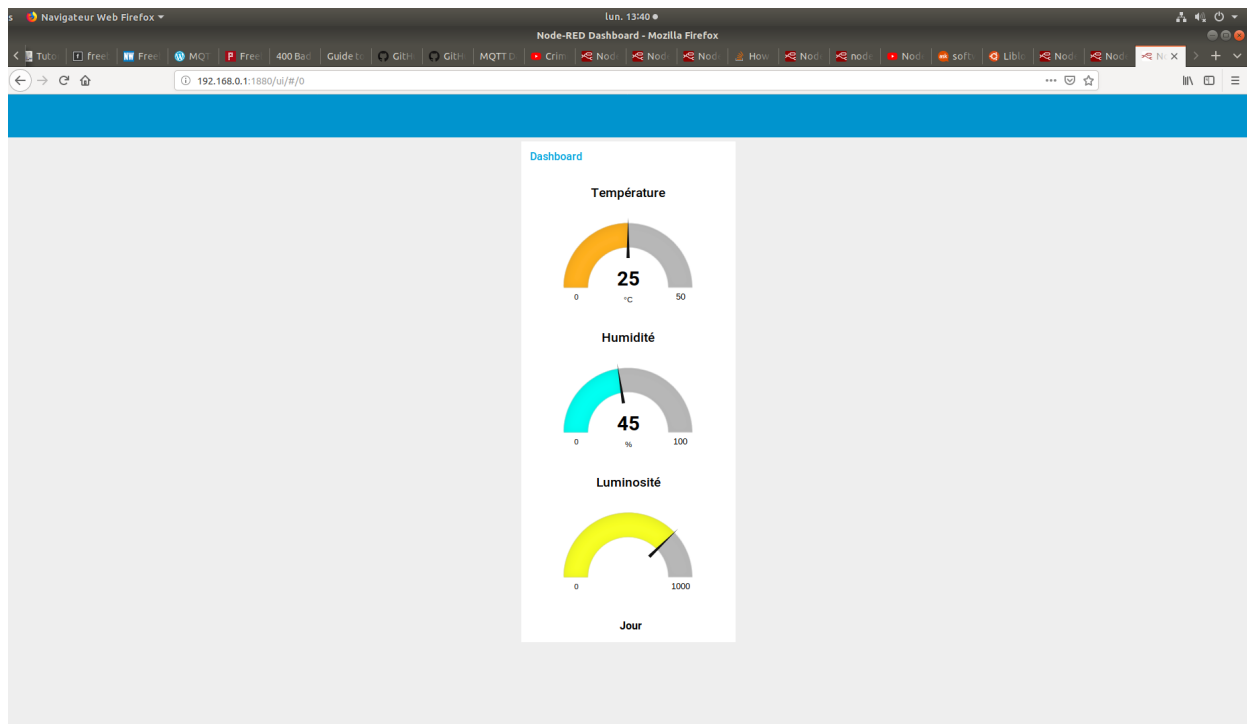


Fig : Le dashboard fonctionnel

Sources :

- (1) <https://github.com/adafruit/DHT-sensor-library/blob/master/DHT.h>
- (2) <https://github.com/hirotakaster/CoAP-simple-library>
- (3) <https://github.com/arduino-libraries/WiFi101/blob/master/src/WiFi101.h>
- (4) <https://github.com/knolleary/pubsubclient>