

Notice de mise en œuvre d'une connexion MQTT et REST

Objet du document

Le but du document est de fournir une procédure afin de mettre en place deux maquettes vérifiant la connexion bidirectionnelle pour les protocoles MQTT et REST :

- Dans le sens montant (uplink) remonter et afficher sur un graphique la température et l'humidité.
- Dans le sens descendant (downlink) activer une LED depuis une interface web

La connexion MQTT

Voici ci-dessous le schéma de notre solution pour la connexion MQTT :

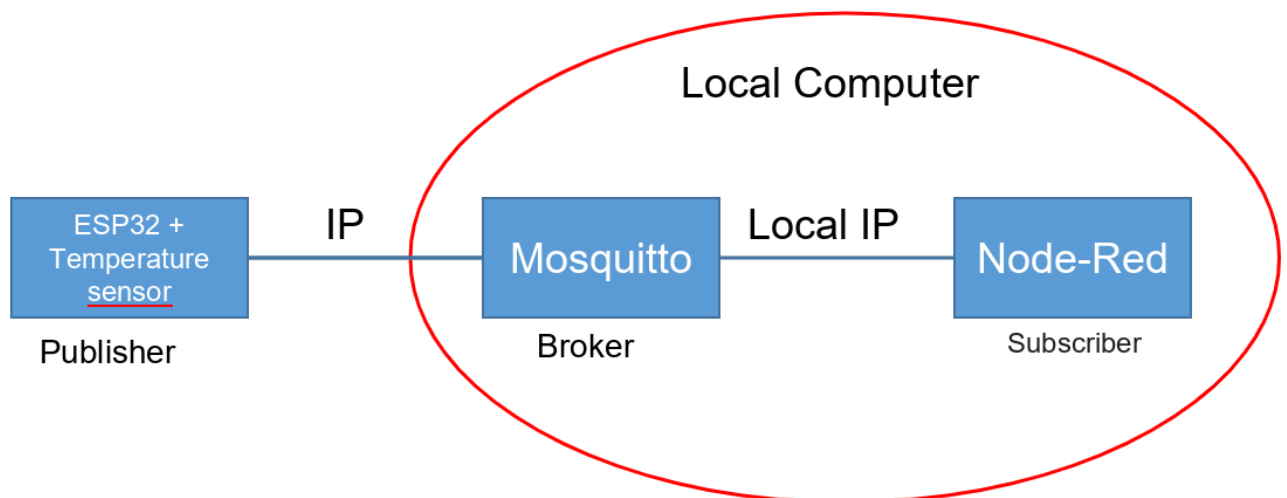


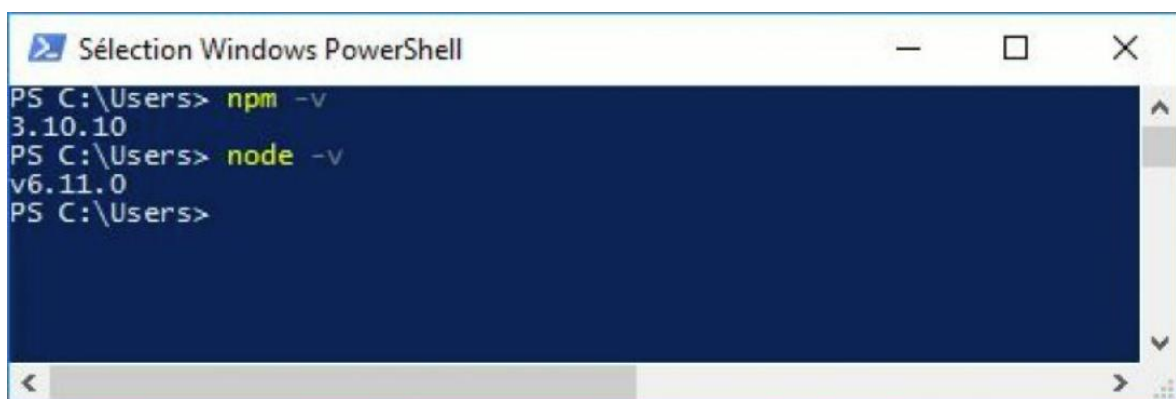
Figure 1 : Schéma de la connexion pour MQTT

Pour cette partie du projet nous utilisons un module ESP32 pour récupérer la valeur du capteur de température et l'envoyer sur un « broker ». Le broker que nous allons utiliser se nomme « Mosquitto » et est installé localement sur l'ordinateur qui nous servira de récepteur pour afficher notre dashboard. De plus afin de mettre en forme les données et de commander une LED nous utilisons Node-RED.

Étape 1 : installation des outils

Pour installer le broker Mosquitto nous avons installer et exécuter l'application que nous avons télécharger sur le site du fabricant : <https://mosquitto.org/download/>

Pour installer Node-RED nous avons installé Node.js que nous avons téléchargé sur le site constructeur : <https://nodejs.org/en/download/> . Pour vérifier que Node.js est bien installé on exécute les commande « **npm -v** » puis « **node -v** » dans le powershell windows qui doit nous indiquer la version des composants, exemple ci-dessous :



```
Sélection Windows PowerShell
PS C:\Users> npm -v
3.10.10
PS C:\Users> node -v
v6.11.0
PS C:\Users>
```

Figure 2 : terminal powershell après exécution des commandes "npm -v" et "node -v"

Une fois les composants bien installé, on installe Node-RED avec la commande suivante (à entrer dans le Powershell Windows) :

```
npm install -g --unsafe-perm node-red
```

Sur Windows 10, les fichiers sont installés dans le répertoire :

```
c:\Users\<Utilisateur>\AppData\Roaming\npm\node_modules
```

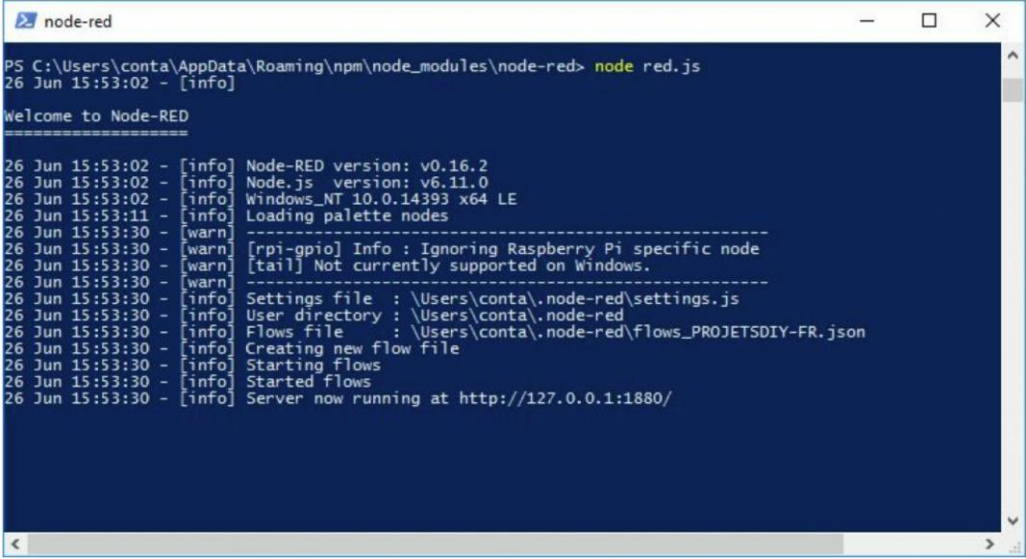
Pour lancer Node-RED il faut se placer dans le répertoire de l'installation, soit dans notre cs :

```
cd C:\Users\<Utilisateur>\AppData\Roaming\npm\node_modules\node-red
```

Puis exécutez la commande commande :

```
node red.js
```

Si tout c'est déroulé correctement voici à quoi doit ressembler votre terminal Powershell Windows :



```

PS C:\Users\conta\AppData\Roaming\npm\node_modules\node-red> node red.js
26 Jun 15:53:02 - [info]
Welcome to Node-RED
=====
26 Jun 15:53:02 - [info] Node-RED version: v0.16.2
26 Jun 15:53:02 - [info] Node.js version: v6.11.0
26 Jun 15:53:02 - [info] Windows_NT 10.0.14393 x64 LE
26 Jun 15:53:11 - [info] Loading palette nodes
26 Jun 15:53:30 - [warn] -----
26 Jun 15:53:30 - [warn] [rpi-gpio] Info : Ignoring Raspberry Pi specific node
26 Jun 15:53:30 - [warn] [tail] Not currently supported on Windows.
26 Jun 15:53:30 - [warn] -----
26 Jun 15:53:30 - [info] Settings file : \Users\conta\.node-red\settings.js
26 Jun 15:53:30 - [info] User directory : \Users\conta\.node-red
26 Jun 15:53:30 - [info] Flows file : \Users\conta\.node-red\flows_PROJETSDIY-FR.json
26 Jun 15:53:30 - [info] Creating new flow file
26 Jun 15:53:30 - [info] Starting flows
26 Jun 15:53:30 - [info] Started flows
26 Jun 15:53:30 - [info] Server now running at http://127.0.0.1:1880/

```

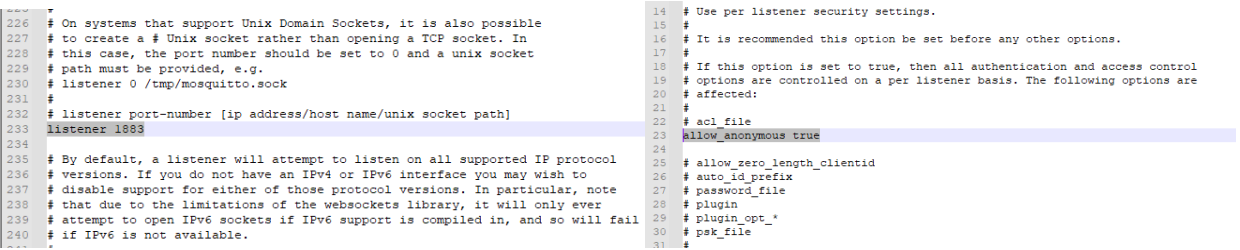
Figure 3 : Node-RED lancé dans un Powershell

Étape 2 : Configuration

Nous allons donc configurer l'ESP32, Mosquitto et Node-RED pour notre projet.

Premièrement nous allons configurer Mosquitto. Pour cela nous allons devoir modifier le fichier « mosquitto.conf » pour ajouter 2 lignes :

- « listener 1883 » : permet d'écouter sur le port 1883
- « allow_anonymous true » : permet d'accepter la connexion de n'importe qui



```

226 # On systems that support Unix Domain Sockets, it is also possible
227 # to create a # Unix socket rather than opening a TCP socket. In
228 # this case, the port number should be set to 0 and a unix socket
229 # path must be provided, e.g.
230 # listener 0 /tmp/mosquitto.sock
231 #
232 # listener port-number [ip address/host name/unix socket path]
233 listener 1883
234
235 # By default, a listener will attempt to listen on all supported IP protocol
236 # versions. If you do not have an IPv4 or IPv6 interface you may wish to
237 # disable support for either of those protocol versions. In particular, note
238 # that due to the limitations of the websockets library, it will only ever
239 # attempt to open IPv6 sockets if IPv6 support is compiled in, and so will fail
240 # if IPv6 is not available.
241
242 # Use per listener security settings.
243 # It is recommended this option be set before any other options.
244 #
245 # If this option is set to true, then all authentication and access control
246 # options are controlled on a per listener basis. The following options are
247 # affected:
248 #
249 # acl_file
250 allow_anonymous true
251 #
252 # allow_zero_length_clientid
253 # auto_id_prefix
254 # password_file
255 # plugin
256 # plugin_opt_*
257 # psk_file
258

```

Une fois la configuration éditée il faut lancer l'application Mosquitto. Pour cela il faut :

1. Lancer un terminal PowerShell
2. Se déplacer dans le dossier où est installé Mosquitto et lancer la commande « ./mosquitto » pour lancer l'application `PS C:\Users\jerem> ./mosquitto`
3. On vérifie que l'application est bien lancée en utilisant la commande « ./mosquitto -v », la version de mosquitto doit alors apparaître.

Ensuite nous allons configurer l'ESP32 pour notre application. Pour cela nous avons récupéré un code sur le site : <https://randomnerdtutorials.com/esp32-mqtt-publish-subscribe-arduino-ide/> que nous avons modifié (le code final est en annexe du document).

Dans le header du code nous retrouvons les variables à modifier pour notre utilisation (nos commentaires sont surlignés en jaune sur l'extrait de code ci-dessous) :

```

#define WIFI_SSID "pc-jeremy"    On indique le nom du réseau wifi que nous utilisons
#define WIFI_PASSWORD "jeremyleplusbg"    On indique le mot de passe du wifi utilisé

// Raspberry Pi Mosquitto MQTT Broker
#define MQTT_HOST IPAddress(192, 168, 137, 94)    On indique l'adresse IP du broker
utilisé, dans notre cas un pc en local
// For a cloud MQTT broker, type the domain name
// #define MQTT_HOST "example.com"
#define MQTT_PORT 1883    On indique le port d'écoute du broker

// Temperature MQTT Topics
#define MQTT_PUB_TEMP "esp32/dht/temperature"    On définit le nom du « topics » à
publier sur le broker
#define MQTT_PUB_HUM "esp32/dht/humidity"    On définit le nom du « topics » à
publier sur le broker

// Digital pin connected to the DHT sensor
#define DHTPIN 4    On définit le numéros de pin sur laquelle est codé l'information
envoyé par le capteur

// Uncomment whatever DHT sensor type you're using
#define DHTTYPE DHT11    // DHT 11    On dé-commente la ligne correspondant à notre
capteur, dans notre cas le DHT11
// #define DHTTYPE DHT22    // DHT 22    (AM2302), AM2321
// #define DHTTYPE DHT21    // DHT 21    (AM2301)

// Initialize DHT sensor
DHT dht(DHTPIN, DHTTYPE);

// Variables to hold sensor readings
float temp;    On déclare le nom des variables que l'on veut publier en global
float hum;

void callback(char* topic, byte* payload, unsigned int length) {

    Serial.print("Message arrived in topic: ");
    Serial.println(topic);

    Serial.print("Message:");
    for (int i = 0; i < length; i++) {
        Serial.print((char)payload[i]);
        if (i == 0) {    On teste si le message envoyé contient la lettre «t» qui compose
le mot «true», si oui on allume la LED sinon on l'éteint
            //Serial.println("boucle if");
            if (payload[i] == 't') {
                digitalWrite(led, HIGH);
            }
            else digitalWrite(led, LOW);
        }
    }
}

```

ATTENTION :

L'ESP32 ne peut se connecter que sur des réseaux wifi 2.4GHz et ne peut pas se connecter sur des réseaux 5GHz

Enfin pour la configuration de Node-RED nous avons appliqué la configuration suivante :

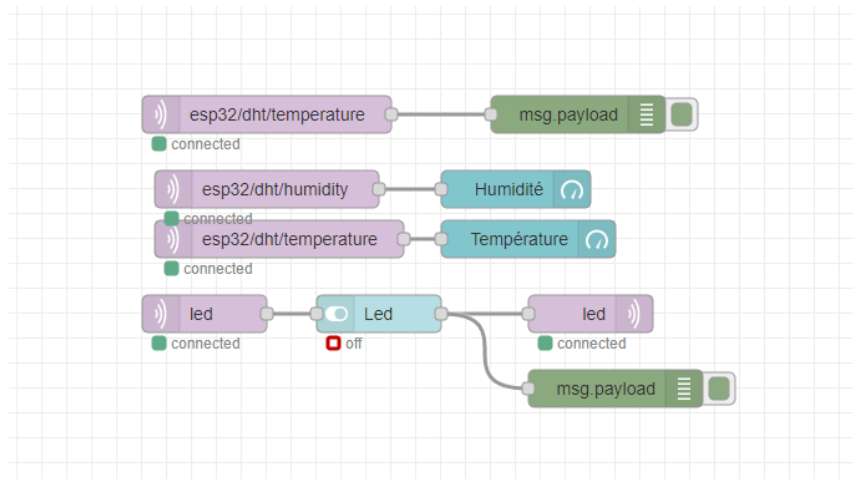
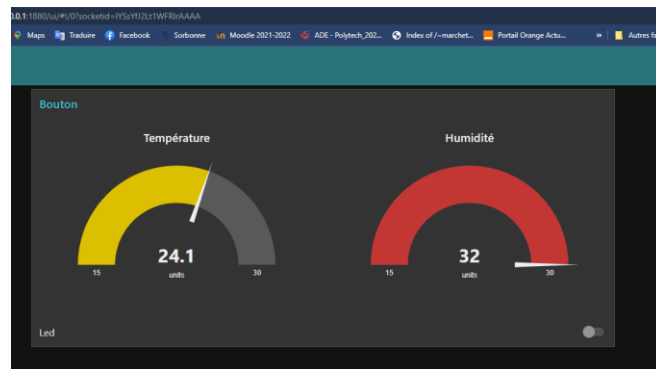


Figure 4 : Configuration de Node-RED



La connexion REST

Voici ci-dessous le schéma de notre solution pour la connexion REST :

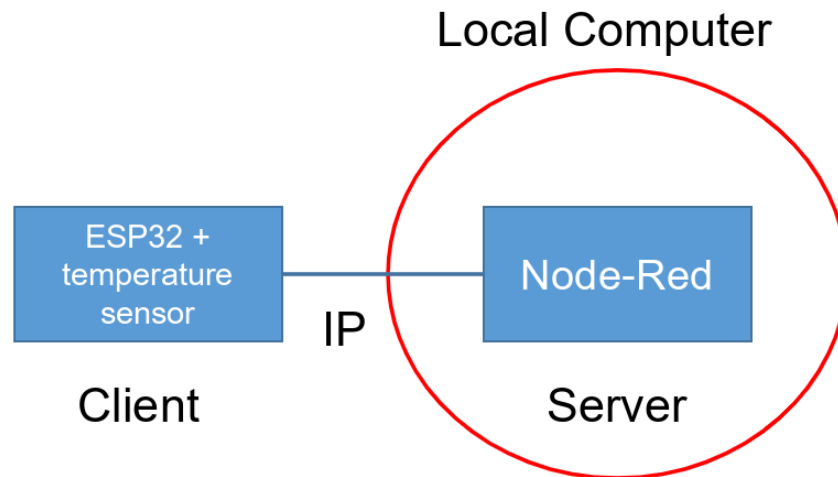


Figure 5 : Schéma de la connexion pour REST

REST est un ensemble de principes architecturaux adapté aux besoins des services web et applications mobiles légers. La mise en place de ces recommandations est laissée à l'appréciation des développeurs.

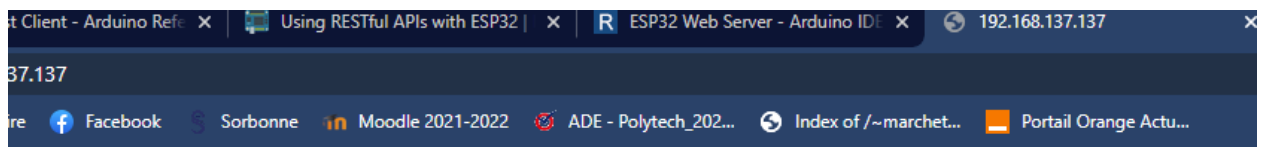
L'envoi d'une requête de données à une API REST se fait généralement par le protocole HTTP (Hypertext Transfer Protocol). À la réception de la requête, les API développées selon les principes REST (appelées API ou services web RESTful) peuvent renvoyer des messages dans différents formats : HTML, XML, texte brut et JSON. Le format JSON (JavaScript Object Notation) est le plus utilisé pour les messages, car, en plus d'être léger, il est lisible par tous les langages de programmation (en dépit de son nom) ainsi que par les humains.

Pour cette partie du projet nous utilisons un module ESP32 pour récupérer la valeur du capteur de température et l'envoyer directement au serveur Node-Red. Node-Red nous servira à gérer notre connexion côté serveur, commander une LED qui se trouve sur l'ESP32, ainsi qu'à mettre en forme et afficher notre dashboard.


Pour faire un envoi de donnée via le protocole REST, on utilise un serveur Web directement sur la carte. La configuration de celui-ci nécessite que de connecter la carte à un réseau Wi-Fi. Cette étape se fait de la même manière que pour le protocole MQTT. Dans le code (sous arduino), on réalise ensuite une page HTML qui actualisera en permanence les valeurs de température et d'humidité à l'aide d'un lecteur analogique du capteur DHT11. On obtient donc un code comme ci-dessous :

```
const char index_html[] PROGRAM = R"rawliteral(  
<!DOCTYPE HTML><html>  
<head>  
  <meta name="viewport" content="width=device-width, initial-scale=1">  
  <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/all.css" integrity="sha384-fnmOCqbTlWIl8LjTjo7mOStyKNC6pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="anonymous">  
  <style>  
    html {  
      font-family: Arial;  
      display: inline-block;  
      margin: 0px auto;  
      text-align: center;  
    }  
    h2 { font-size: 3.0rem; }  
    p { font-size: 3.0rem; }  
    .units { font-size: 1.2rem; }  
    .dht-labels {  
      font-size: 1.5rem;  
      vertical-align: middle;  
      padding-bottom: 15px;  
    }  
  </style>  
</head>  
<body>
```

Ainsi, on se connectant en local à la carte ESP32, via un navigateur Web, on obtient la page Web suivante :



Serveur ESP32 DHT

 Température 26,70 °C

 Humidité **39.00** %

L'avantage du format HTML est qu'il est possible d'inclure tout type d'image, de logos... afin d'avoir une interface très personnalisée et agréable à lire.

Afin de pouvoir agir sur un élément de la carte, une Led par exemple, on met en place un bouton sur l'interface Web qui renverra une valeur TRUE/FALSE. En récupérant cette valeur dans Arduino, il est possible de commander la Led voulu. On obtient l'interface suivante :



Ainsi, on arrive à récupérer les valeurs de la carte et à interagir avec celle-ci via un navigateur Web tout en utilisant le protocole REST.

Conclusion :

Ce projet nous a permis de mettre en place une communication en utilisant deux protocoles différents. Cela permet de mieux comprendre la puissance de l'IIoT. Il existe un grand nombre de protocoles différents pour l'IIoT, MQTT est très utilisé et permet à plusieurs appareils de communiquer ensemble très facilement. En revanche, on a vu qu'il est assez complexe à mettre en place. Le protocole REST est plus limité mais il est extrêmement simple à implémenter.