Mettre en oeuvre un client MQTT sur un EP8266 (ESP32) Feather Huzzah ou MKR1010

[Mise à jour le 26/2/2021]

Prérequis

Lire les généralités sur le protocole MQTT et installer un **broker Mosquitto sur un Raspberry Pi**. Pour cela, on pourra consulter la page "Installer un broker MQTT Mosquitto sur un Raspberry Pi"

Ressources

- Article *"Faites communiquer vos projets simplement avec MQTT"* sur Hackable Magazine n°26
- Tutoriel Node-Red & MQTT with Arduino ESP8266
- Exemples de code pour un ESP8266 sur Github
- Bibliothèque pubsubclient pour implémenter un client MQTT sur Arduino disponible sur Github
- Documentation des bibliothèques Arduino pour l'ESP8266 : ESP8266 Arduino Core
- Bibliothèque pour la résolution de nom : mDNS

Lectures connexes

- Wiki Raspberry Pi - Installer un broker MQTT Mosquitto sur un Raspberry Pi
- Wiki Raspberry Pi sous Linux - Créer un flux de données et une interface utilisateur avec Node-RED
- Wiki Réseau - Tester un broker Mosquitto avec MQTTlens
- Wiki Web - Créer un client MQTT (Websocket) avec Eclipse Paho
- Wiki Raspberry Pi sous Linux - Sauvegarder ses données dans une base TSDB (InfluxdB)

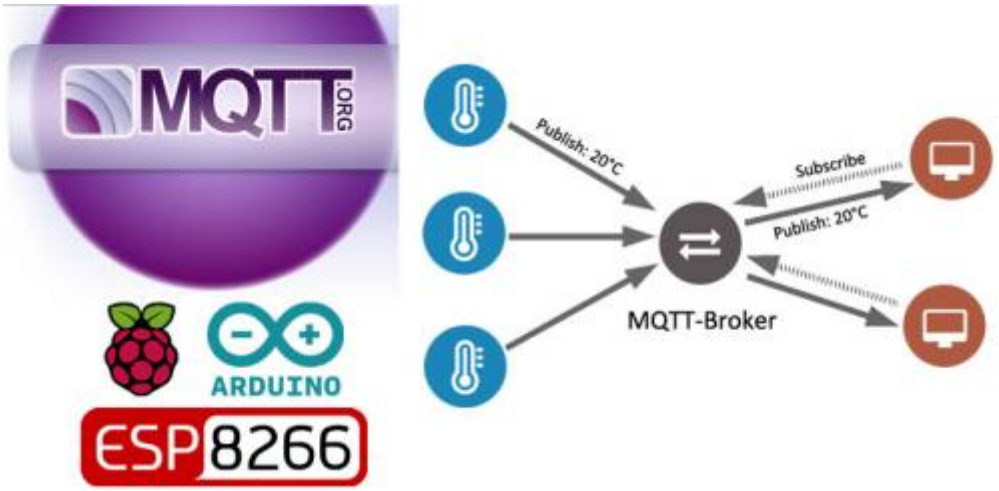
Mots-clés

client[1], **serveur** [2], **broker MQTT**[3], **subscriber**[4], **publisher**[5], **topic MQTT**[6], **payload**[7](charge utile), **joker**[8], sécurité, **QoS**[9].

A. Découverte

Objectif

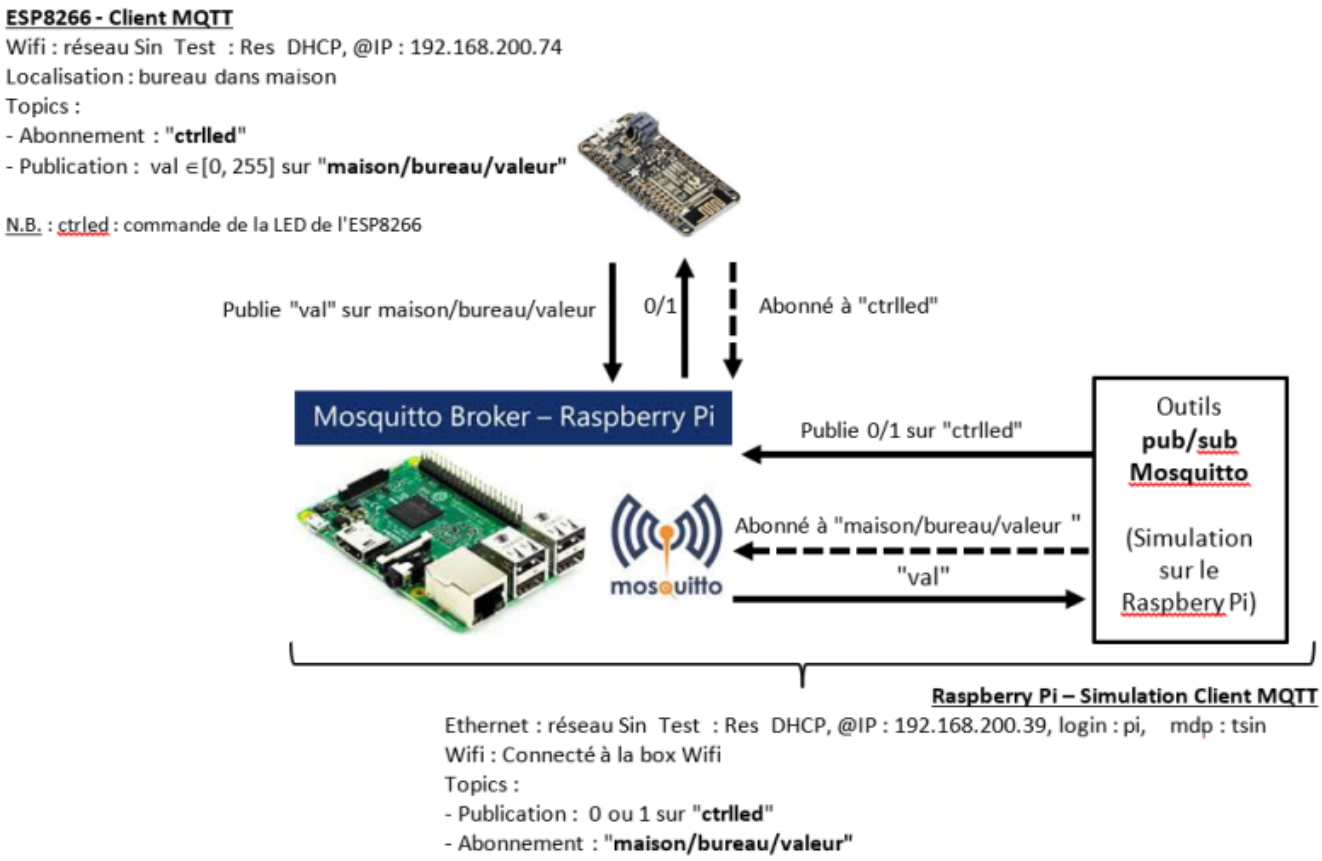
Se familiariser avec le protocole MQTT (Message Queuing Telemetry Transport), standard de l'IoT (Internet Of Things), en créant une communication entre deux clients MQTT, un ESP8266 et un outil de simulation via un broker Mosquitto installé sur un Raspberry Pi.
La préparation du Raspberry Pi est décrite sur la page "Installer un broker MQTT Mosquitto sur un Raspberry Pi".



Cahier des charges

- **Schéma**

Le schéma ci-dessous illustre le fonctionnement attendu :
- La LED embarquée de l'ESP8266 est commandée par un client MQTT simulé sur le Raspberry Pi.
- L'ESP8266 publie une valeur toute les 5s (simule un capteur).



- **Topics**
  - abonnement à “ctrlled”
  - publication de “maison/bureau/valeur”
- **Algorithme du programme ARD_ESP8266_MQTT.ino**

```
Algorithme PubSubClient_n
// Initialisations
Connexion au point d'accès wifi
Configuration du broker
Activation de la fonction de rappel chargée de traiter les messages reçus

Répéter (toujours)
   début
     // Maintenir la connexion et s'abonner (souscrire) à un ou plusieurs
topics
     si (non connecté au broker) alors
       se connecter puis s'abonner à un ou plusieurs topics
     fin si
     // Traiter les messages reçus
     si (un message a été reçu) alors
       la fonction de rappel traite le message
     fin si
     // Publier
     si (le temps d'attente entre deux publications est atteint) alors
       publier le ou les messages associer à leur topic
     fin si
```

```
fin
```

1. Sauvegarde des paramètres de connexion au réseau Wifi

Chaque client MQTT doit être identifiable et se connecter au réseau wifi. Pour cela il doit mémoriser le **ssid** et le **mot de passe** du réseau mais également fournir un **nom d'hôte**.

- **Espressif**

En sauvegardant ces paramètres de connexion dans la mémoire **EEPROM émulée** de chaque ESP8266 (ESP32), on se libère de cette tâche et on fait en sorte qu'ils ne soient pas affectés par la suite lors de la mise à jour du croquis.

Le croquis ci-dessous stocke les paramètres de connexion dans une structure, les enregistre dans l'EEPROM émulée de l'ESP8266 (ESP32) et les relit pour les afficher dans le moniteur série. Dès son exécution, ces données seront stockées indépendamment des enregistrements de croquis qui suivront.

Ce **croquis** n'est donc **à exécuter** qu'**une fois** sur chaque carte destinée à être programmée en client MQTT.

ARD_ESP_infosClientMQTT.cpp

```
#include <EEPROM.h>

// Structure
struct EEconf
{
  char ssid[32];
  char password[64];
  char myhostname[32];
};

void setup()
{
  // Initialisation des paramètres de connexion
  EEconf myconf = {              // A remplacer par :
                   "monSSID",    // le SSID du réseau,
                   "motdepasse", // le mot de passe du réseau
                   "nomhote"};   // le nom à donner au client MQTT

  // Variable pour la relecture
  EEconf readconf;
```

```
  Serial.begin(115200);

  // Initialisation EEPROM
  EEPROM.begin(sizeof(myconf));
  // Enregistrement
  EEPROM.put(0,myconf);
  EEPROM.commit();

  // Relecture et affichage
  EEPROM.get(0,readconf);
  Serial.println("\n\n\n");
  Serial.println(readconf.ssid);
  Serial.println(readconf.password);
  Serial.println(readconf.myhostname);
}

void loop()
{
}
```

Télécharger le fichier *ARD_ESP_infosClientMQTT.ino* pour ESP32, ESP8266 (intégré à un projet **VSCode**).

- **MKR1010** : le SSID, le mdp du réseau et le nom de la carte sont placé dans un fichier de configuration.

2. Conception du client MQTT (template)

- **Source** : bibliothèque PubSubClient de Nick O'Leary[10] sur Github.

Le client MQTT décrit dans ce paragraphe repose sur la bibliothèque **PubSubClient**. Celle-ci fournit un client pour faire des messages de publication / abonnement simples avec un serveur prenant en charge MQTT.

- Le client utilise **MQTT 3.1.1** par défaut. Il peut être modifié pour utiliser MQTT 3.1.
- Il ne peut **publier** que des messages **QoS 0**. Il peut s'abonner à QoS 0 ou QoS 1.
- La taille maximale du message (configurable), y compris l'en-tête, est de **256 octets** par défaut.
- L'intervalle keepalive (configurable) est défini sur **15 secondes** par défaut.

Ce client cible un **ESP8266**. Moyennant quelques modifications, il peut être adapté à un **ESP32** ou à une carte Arduino **MKR1010**. Le code adapté à ces différentes cartes est téléchargeable à la fin de ce paragraphe.

a. Préparer l'IDE

Télécharger et installer les bibliothèques **ESP8266mDNS** et **PubSubClient** à l'aide du gestionnaire de bibliothèques de l'IDE.

b. Inclure les bibliothèques

Le croquis débute par l'inclusion des bibliothèques utilisées dans la communication :

- ESP8266WiFi : connexion de la carte ESP8266 au réseau wifi
- ESP8266mDNS : résolution des noms des cartes ESP8266 et Raspberry Pi (fonctionne avec le service Bonjour sous Windows)
- EEPROM : émule une EEPROM dans l'ESP8266
- PubSubClient : permet d'implémenter un client MQTT

biblio.cpp

```
// MQTT : implémentation d'un client MQTT
#include <PubSubClient.h>
// Connexion au wifi
#include <ESP8266WiFi.h>
// mDNS pour la résolution des noms des hôtes
#include <ESP8266mDNS.h>
//  EEPROM : émule une EEPROM dans l'ESP8266
#include <EEPROM.h>
// Paramètres de connexion au broker
#include "parametres.h"
```

c. Déclarer les constantes et les variables globales

init.cpp

```
// ----------------------------------------------------------------------
-------------
// Broker
// ----------------------------------------------------------------------
-------------
// Nom (mDNS) de la machine sur laquelle est installé le broker et port
// Ne pas modifier, RENSEIGNER les paramètres situés dans parametres.h
const char *mqtt_server = BROKER;
uint16_t mqtt_PORT = MQTTPORT; // Port TCP sur lequel le broker écoute
// (par défaut pour brokers MQTT)
// ----------------------------------------------------------------------
-------------
// Topics
// ----------------------------------------------------------------------
```

```
-------------
const char inTopicLed[] = "ctrl_led"; // #### A adapter ####
const char outTopicVal[] = "valeur";  // #### A adapter ####
// ----------------------------------------------------------------------
-------------
// Structure pour la configuration de la connexion au réseau wifi
struct EEconf
{ // Les champs sont remplis avec les données stockées dans l'EEPROM
(émulée)
    // par le croquis infoClientMQTT_ESP8266.ino
    char ssid[32];       // SSID du réseau. Exemple : SynBoxLAN,
    char password[64];   // Mot de passe du réseau. Exemple : 12345678
    char myhostname[32]; // Nom donné au client MQTT. Exemple :
ESP8266_1
} readconf;
// Objet pour la connexion au réseau wifi
WiFiClient espClient;
// Objet pour la connexion au broker MQTT (Publisher/Subscriber)
PubSubClient mqttClient(espClient);
// Intervalle de temps séparant la publication des topics
const long interval = 6000; // #### A adapter ####
// Permet de calculer l'intervalle de temps depuis la dernière
// publication des topics
unsigned long previousMillis = 0;
// Valeur à publier
byte val = 0;
```

d. Se connecter au point d'accès Wifi (fonction)

Ressource : Les modes de fonctionnement du Wifi (802.11 ou Wi-Fi)

Pour simplifier l'écriture du champ **setup()**, la connexion au réseau wifi et la configuration mDNS sont placées dans une fonction. On suppose que les données de configuration ont été mémorisées dans l'ESP8266 avec le croquis *infosClientMqtt_ESP8266.ino*.

connect.cpp

```
void setup_wifi()
{
    // mode station
    WiFi.mode(WIFI_STA);
    Serial.println();
    Serial.print("Tentative de connexion à ");
    Serial.println(readconf.ssid);
    // Connexion Wifi
    WiFi.begin(readconf.ssid, readconf.password);
    while (WiFi.status() != WL_CONNECTED)
    {
```

```
        delay(5000);
        Serial.print(".");
    }
    // Affichage
    Serial.println("");
    Serial.println("Connexion Wifi ok");
    Serial.println("Adresse IP: ");
    Serial.println(WiFi.localIP());
    // Configuration mDNS
    WiFi.hostname(readconf.myhostname);
    if (!MDNS.begin(readconf.myhostname))
    {
        Serial.println("Erreur de configuration mDNS !");
    }
    else
    {
        Serial.println("Répondeur mDNS démarré");
        Serial.println(readconf.myhostname);
    }
}
```

e. Traiter les messages reçus (fonction)

Le principe de fonctionnement de la bibliothèque **PubSubClient** repose sur un mécanisme de **callback** (fonction de rappel[11]).

Dans le cas présent, la bibliothèque demande que l'on implémente cette fonction de rappel. Elle sera appelée en cas de réception d'un message sur n'importe lequel des topics souscrits. **Le prototype de cette fonction, le nombre et le type des arguments qui lui sont passés est imposé par la bibliothèque**.

recept.cpp

```
void callback(char *topic, byte *payload, unsigned int length) // topic
: chaîne de caractères décrivant le topic
{                                                               //
payload : message associé au topic
                                                                //
length : taille des données du message
    // Afffichage dans la console pour debug
    Serial.print("Message [");
    Serial.print(topic);
    Serial.print("] ");
    // Affichage du message
    for (int i = 0; i < length; i++)
    {
        Serial.print((char)payload[i]);
    }
```

```
    Serial.println("");
    // '1' est-il le premier caractère du message ?
    // Traitement
    if ((char)payload[0] == '1') // Récupération du premier caractère du payload
    {
        // Oui led=on
        digitalWrite(BUILTIN_LED, LOW);  // Led de la carte active à l'état bas !
    }
    else
    {
        // Non led=off
        digitalWrite(BUILTIN_LED, HIGH);
    }
}
```

f. Maintenir la connexion au broker MQTT et s'abonner au(x) topic(s) (fonction)

Pour simplifier l'écriture du champ **loop()**, la connexion au broker et la souscription aux messages destinés à contrôler la LED embarquée sont placées dans une fonction. Tant que la connexion est maintenue avec le broker cet abonnement reste actif et l'on peut recevoir des messages. Dans le cas contraire, il est nécessaire de se reconnecter et de se réabonner au topic. Dès que cet abonnement est réalisé, l'arrivée d'un message est automatiquement pris en charge par la bibliothèque et la fonction callback est appelée en passant le topic en argument ainsi que la donnée associée (payload) et sa longueur.

connect.cpp

```
void reconnect()
{
        // Non, on se connecte
        if (!mqttClient.connect(readconf.myhostname))
        {
            Serial.print("Erreur de connexion MQTT, rc=");
            Serial.println(mqttClient.state());
            delay(5000);
            continue;
        }
        Serial.println("Connexion serveur MQTT ok");
        // Connecté ! On s'abonne au topic (ici "ctrlled")
        mqttClient.subscribe(inTopicLed); // #### A adapter ####
    }
}
```

g. Programme principal

setup.cpp

```
// Initialisation
void setup()
{
    // Configuration de la broche connectée à la LED de la carte
    pinMode(BUILTIN_LED, OUTPUT);
    // Configuration du moniteur série
    Serial.begin(115200);
    // Lecture des paramètres sauvegardés par
ARD_ESP_SauveInfosClientMqtt.ino
    EEPROM.begin(sizeof(readconf));
    EEPROM.get(0, readconf);
    // Connexion au Wifi
    setup_wifi();
    // Configuration du broker
    // MQTT_PORT = 1883 (port TCP par défaut des brokers MQTT)
    mqttClient.setServer(mqtt_server, mqtt_PORT);
    // Activation de la fonction callback
    mqttClient.setCallback(callback);
}
```

Les messages envoyés par le protocole MQTT sont des chaînes de caractères, tout comme les topics. Des tableaux sont utilisés pour les stocker. Dans le cas présent, le topic est construit avec le nom d'hôte de l'ESP8266

La gestion de MQTT par la bibliothèque se fait par des appels répétés à ***client.loop()***. C'est pour cette raison que nous ne pouvons pas gérer le temps d'attente entre deux transmissions avec un *delay()* car cela bloquerait l'exécution du croquis, empêchant la réception des messages sur les topics auquel on a souscrit.

loop.cpp

```
void loop()
{
    // Tableau pour la conversion de la valeur "val" à transmettre
    char msg[16];
    // Tableau pour le topic
    char topic[64];

    // Sommes-nous connectés au broker MQTT ?
    if (!mqttClient.connected())
    {
        // Non alors
        reconnect(); // maintenir la connexion et s'abonner à un ou
```

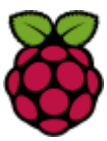
```
plusieurs topics
    }
    // Oui, maintien de la connexion avec le broker
    // Interrogation du broker : ne doit pas être bloquée !!! pour
traiter les messages reçus
    mqttClient.loop(); // Déclencle la fonction callback si message
reçu

    // Temporisation "non blocante" avant l'envoi d'un ou plusieurs
topics
    long currentMillis = millis();
    if (currentMillis - previousMillis > interval) // si temporisation
> interval en ms
    {                                              // alors on publie
        // On mémorise la "date" à laquelle le message a été envoyé
        previousMillis = currentMillis;
        val++; // incrémentation de la valeur à transmettre
        // Construction du message
        sprintf(msg, "%hu", val);
        // Construction du topic avec le nom d'hôte
        // Exemple "<myhostname>/<valeur>" #### A adapter ####
        sprintf(topic, "%s/%s", readconf.myhostname, outTopicVal);
        // Publication du message sur le topic
        mqttClient.publish(topic, msg);
    }
}
```

h. Téléchargement des templates

Le code complet (MàJ le 26/02/2021), intégré à un projet **VSCode** est accessible à partir des liens ci-dessous.

- pour un ESP8266 : lien
- pour un ESP32 : lien
- pour un MKR1010 : lien

3. Essais et validation du croquis



La vérification du fonctionnement des croquis peut se faire dans une console, sur un **Raspberry Pi**, à l'aide des outils logiciels **mosquitto_pub** et **mosquitto_sub**. Au préalable, il est nécessaire d'avoir installé un broker MQTT conformément aux indications de la page "Installer un broker MQTT Mosquitto sur un Raspberry Pi". Voir le paragraphe "Sécurité" ci-dessous si un mot de passe a été défini sur le Raspberry Pi. Cette vérification peut également se faire après avoir installé l'extension MQTTlens dans Chrome.

3.1 Publication d'un message sur un topic

La commande de la LED de la carte ESP866 peut se faire comme ci-dessous :

- **Sans sécurité**

cmd1a.bash

```
mosquitto_pub -h localhost -t ctrlled -m 0
mosquitto_pub -h localhost -t ctrlled -m 1
```

- **Avec sécurité**

cmd1b.bash

```
mosquitto_pub -h localhost -u "sondes" -P "mot2passe" -t ctrlled -m 0
mosquitto_pub -h localhost -u "sondes" -P "mot2passe" -t ctrlled -m 1
```

3.2 Abonnement à un topic

L'affichage dans une console sur le Raspberry Pi de la valeur envoyée par l'ESP8266 toutes les 5s peut se faire comme ci-dessous :

- **Sans sécurité**

cmd2a.bash

```
mosquitto_sub -h localhost -t maison/bureau/valeur
```

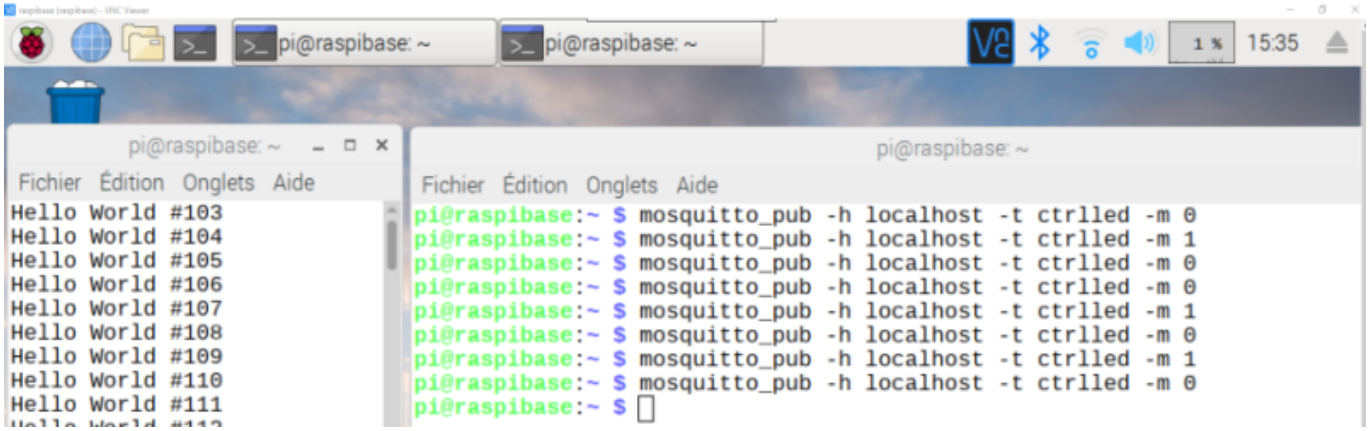
- **Avec sécurité**

cmd2b.bash

```
mosquitto_sub -h localhost -u "sondes" -P "mot2passe" -t
maison/bureau/valeur
```

> **N.B.**
> **-h** : permet de spécifier l'hôte sur lequel fonctionne le broker (ici un Raspberry Pi <nomRaspi>.local).
> **-t** : permet de spécifier le topic (publication, abonnement).
> **-m** : permet de spécifier le message (payload).
> **-u** : permet de spécifier l'utilisateur.
> **-P** : permet de spécifier le mot de passe associé à l'utilisateur.

Résultat attendu



4. Sécurité

Si un fichier de mots de passe a été défini sur le broker Mosquitto comme cela est indiqué au paragraphe “Sécurité” de la page "Installer un broker MQTT Mosquitto sur un Raspberry Pi" alors, lorsque cette sécurité est mise en place, le broker n'est plus accessible sans transmettre un identifiant et un mot de passe.

```
pi@raspibase:~ $ mosquitto_sub -h localhost -t maison/+/valeur
Connection Refused: not authorised.
Connection Refused: not authorised.
```

- Il faut modifier la commande *client.connect()* de la fonction *reconnect()* dans le croquis **ARD_ESP8266_MQTT.ino** comme ci-dessous :

cmd3.bash

```
client.connect(readconf.myhostname, "sondes","mot2passe")
```

- Pour simplifier les tests, il est également possible d'**autoriser les connexions anonymes** en ouvrant le fichier d'authentification *auth.conf* afin de modifier *allow_anonymous* comme ci-dessous.

*.bash

```
sudo nano  /etc/mosquitto/conf.d/auth.conf
allow_anonymous true # Connexions sans mot de passe autorisées
```

5. QoS

La qualité de service (QDS) ou quality of service (QoS) est la capacité à véhiculer dans de bonnes conditions un type de trafic donné. Voir "Installer un broker MQTT Mosquitto sur un Raspberry Pi" pour plus d'informations.

La bibliothèque Arduino **PubSubClient** ne gère que le **QoS de niveau 0** !

Résumé pour un ESP8266

- La gestion des **paramètres de connexion** est simplifiée en les sauvegardant dans l'**EEPROM** émulée à l'aide du fichier *ARD_ESP_infosClientMQTT.ino*. Ces paramètres sont à configurer comme dans l'exemple ci-dessous.

*.cpp

```
  // Initialisation des paramètres de connexion
  EEconf myconf = {
      // A remplacer par :
      "SynBoxRTest", // - le SSID du réseau. Exemple : SynBoxRTest,
      "12345678",    // - le mot de passe du réseau. Exemple : 12345678
      "chambre2"     // - le nom à donner au client MQTT. Exemple :
bureau.
                     // Ce nom peut être repris dans les topics.
  };
```

- Si mDNS est installé, **configurer le nom du broker** et **créer les topics** dans le fichier *clientmqttesp8266.ino* comme dans les exemples ci-dessous.

*.cpp

```
// Nom de la machine sur laquelle est installé le broker (mDNS)
const char *mqtt_server = "RPi3bp2.local"; // #### A adapter ####
```

- La **publication** des topics est placée dans la boucle **loop**

*.cpp

```
// Extrait
// Construction du topic avec le nom d'hôte
// Topic "maison/<myhostname>/valeur" #### A adapter ####
sprintf(topic, "maison/%s/valeur", readconf.myhostname);
```

- La **réponse** à un **abonnement** est placée dans une **fonction de rappel** nommée *callback*.

*.cpp

```
// Gestion des topics
void callback(char *topic, byte *payload, unsigned int length)
```

```
{
// A compléter
}
```

B. Pour aller plus loin

Projet Domotique HG (MàJ le 26/2/21)

- Description du projet ici
- Code des cartes ArduinoESP8266, ESP32 et MKR1010 ici
- Code Node-Red ici

1)

Dans un réseau informatique, un client est le **logiciel** qui envoie des demandes à un serveur.

2)

Un serveur informatique est un dispositif informatique (matériel ou logiciel) qui **offre des services**, à un ou plusieurs clients.

3)

Serveur ou **courtier** des messages. Il se charge de les aiguiller vers les différents clients qui se sont abonnés.

4)

Abonné à un ou plusieurs topics.

5)

Editeur de messages.

6)

Sujet ou canal d'information. Dans MQTT, le mot topic fait référence à une chaîne UTF-8 que le courtier utilise pour filtrer les messages des clients.

7)

Les messages possèdent un payload, c'est à dire, une propriété contenant les informations les plus utiles.

8)

Caractère générique utilisé dans le mécanisme de filtrage des messages.

9)

La qualité de service (QDS) ou **quality of service** (QoS) est la capacité à véhiculer dans de bonnes conditions un type de trafic donné.

10)

IBM, concepteur de Node-RED

11)

Une fonction de rappel est passée en paramètre d'une autre fonction et est appelée automatiquement dans certaines situations.