installer MQTT bis --- A tester ----

Raspberry Pi - Installer un broker (serveur) MQTT Mosquitto

[Mise à jour le 1/8/2022] <html>En cours de rédaction</html>

- **Sources**
  - <html>Fundamentals of MQTT</html>
  - **Hackable Magazine n°26** : *"Faites communiquer vos projets simplement avec MQTT"*
  - Généralités sur <html>Wikipédia</html>
  - Site de référence <html>mqtt.org</html>
  - Eclipse <html>Mosquitto</html> An open source MQTT broker
  - Série d'articles sur <html>hivemq.com</html>
  - <html>Using MQTT Over WebSockets with Mosquitto</html>
- **Lectures connexes**
  - Wiki Arduino - Mettre en œuvre un client MQTT sur un EP8266 (ESP32) Feather Huzzah, MKR1010 ou Arduino Uno Wifi 2
  - Wiki Réseau - Tester un broker Mosquitto avec MQTTlens
  - Wiki Raspberry Pi sous Linux - Créer un flux de données et une interface utilisateur avec Node-RED
  - Wiki Web - Créer un client MQTT (Websockets) 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].

1. MQTT (généralités)

Pour répondre à la problématique du nombre grandissant d'objets connectés qui vont faire leur apparition sur la toile (selon une étude Gartner : près de 26 milliards d'objets connectés seront sur Internet d'ici 2020), l'IoT (Internet Of Things), s'est doté d'un nouveau standard : **MQTT** (**M**essage **Q**ueuing **T**elemetry **T**ransport).

Pourquoi MQTT et pas un autre ?

MQTT est ouvert, simple, léger et facile à mettre en œuvre. Il est idéal pour répondre aux besoins suivants :

- Particulièrement adapté pour utiliser une **très faible bande passante**,
- Idéal pour l'utilisation sur les **réseaux sans fil**,
- **Faible consommateur** en énergie,
- **Très rapide**, il permet un temps de réponse supérieur aux autres standards du web actuel,
- Permet une **forte fiabilité** si nécessaire,
- Nécessite **peu de ressources** processeurs et de mémoires.

1.1 Historique

Le protocole MQTT (MQ Telemetry Transport) trouve ses origines en 1999 dans les travaux de **Andy Standford-Clark** et **Arlen Nipper**, alors qu'ils travaillaient pour IBM au développement d'un protocole pour une utilisation industrielle de télémétrie en lien avec l'industrie pétrolière.

1.2 Principes

1.2.1 Organisation et communication

<callout type="primary" icon="true">**MQTT** est un **service de publication/abonnement** TCP/IP simple et extrêmement léger. Il fonctionne sur le principe **client/serveur**.</callout>

Le **serveur** ou **courtier**, nommé **broker**, va collecter des informations que les **éditeurs** (**publishers**) vont lui transmettre. Certaines informations collectées par le broker seront renvoyées à certains **abonnés** (**subscribers**) ayant préalablement fait la demande au broker. Un **client** peut être à la fois éditeur et abonné.

Le principe d'échange est très proche de celui de Twitter. Les messages sont envoyés par les éditeurs sur un **canal d'information** appelé **topic**. Ces messages peuvent être lus par les abonnés. Les topics peuvent avoir une hiérarchie qui permet de sélectionner finement les informations que l'on désire.

<callout type="primary" icon="true">Les **messages** envoyés par les éditeurs peuvent être de toute sorte, mais ne peuvent excéder une taille de **256 Mo** bien que dans les **mises en œuvre réelles**, le maximum soit de **2 à 4 Ko**.</callout>

Résumé

<callout type="success" icon="true">MQTT fonctionne sur **TCP/IP** et fait intervenir deux types d'acteurs : des **clients** (***subscriber***, ***publisher***) pouvant à la fois envoyer et recevoir des messages et un ***broker*** MQTT chargé de recevoir tous les messages et de les transmettre aux clients inscrits. Le principal travail du broker est de servir de relai. Pour cela, il maintient un répertoire de type "**qui veut quoi**" sous la forme de **sujets** ou ***topics***. </callout>

1.2.2 Les topics

<callout type="primary" icon="true">Un **topic** est une simple **chaine de caractères**, mais qui peut être **structurée hiérarchiquement**.
Exemple : **maison/salon/temperature**</callout>

Exemple
Le **topic** "**maison/salon/temperature**" communiquera la température du salon (la sonde de température présente dans le salon publiera régulièrement la température relevée sur ce topic).

Les trois clients établissent une connexion TCP avec le broker. Les clients B et C souscrivent au topic temperature.	Le Client A publie sur le topic temperature une valeur de 22,5°. Le broker propage le message à tous les clients ayant préalablement souscrit au topic Temperature.

<callout type="primary" icon="true">Cette écriture hiérarchique permet à un abonné de souscrire à un ensemble de topics en utilisant des **caractères joker** (**+**, **#**).</callout>

Le caractère joker +
+ est le joker pour un **unique niveau hiérarchique**. Un client souscrivant à "**maison/+/temp**" recevra les messages adressés par d'autres clients aux topics :

- "maison/salon/temp"
- "maison/garage/temp"
- "maison/couloir/temp"

mais pas :

- "maison/salon/hum"
- "jardin/temp"

Le caractère joker #
Le **#** est un joker **multiniveau** s'utilisant toujours après un / et en dernier caractère. Il est destiné à remplacer n'importe quel niveau supérieur dans le topic.
"**maison/#**" correspondra aux topics :

- "maison/salon/temp"
- "maison/salon/hygro"
- "maison/rdc/salon/hum"

mais pas :

- "annexe/couloir/hum"
- "jardin/temp"

Le caractère joker $
Le joker **$** ne peut pas être utilisé pour publier. Il précède les topics concernant les **statistiques** internes du broker. Son utilisation est illustrée au paragraphe Le broker Mosquitto.

<callout type="warning" icon="true">Voir les **bonnes pratiques** d'écriture des topics sur
<html><a href="https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/"

target="_blank">hivemq.com</html></callout>

1.2.3 Sécurité

Les données IoT échangées peuvent s'avérer très critiques, c'est pourquoi il est possible de sécuriser les échanges à plusieurs niveaux :

- Transport en SSL/TLS,
- Authentification par certificats SSL/TLS,
- Authentification par login/mot de passe.

1.2.4 Qualité de service (QoS)

MQTT intègre en natif la notion de QoS. En effet, le publisher à la possibilité de définir la qualité de son message.

Trois niveaux sont possibles :

- Un message de QoS **niveau 0** « **Au plus une fois** ». Le niveau de QoS minimal est zéro. Ce niveau de service garantit une livraison au mieux. Il n'y a aucune garantie de livraison. Le destinataire n'accuse pas réception du message et le message n'est pas stocké ni retransmis par l'expéditeur. Le niveau de QoS 0 est souvent appelé "**fire and forget**". Ce niveau de service doit être utilisé si:
  - Internet est fiable.
  - La perte de message à petite échelle n'a pas d'importance.
  - Les messages doivent être livrés rapidement.

- Un message de QoS **niveau 1** « **Au moins une fois** ». Le niveau de qualité de service 1 garantit qu'un message est remis au moins une fois au destinataire. L'expéditeur stocke le message jusqu'à ce qu'il reçoive du destinataire un paquet PUBACK qui accuse réception du message. Il est possible qu'un message soit envoyé ou remis plusieurs fois. Le niveau 1 de QoS est plus lent que le niveau 0. Ce niveau de service doit être utilisé si:
  - Le client ou le courtier doit recevoir tous les messages.
  - Les messages en double peuvent être traités correctement.

<callout type="primary" icon="true">MQTT QoS niveau 1 est utilisé dans les courtiers MQTT commerciaux comme AWS IoT, Azure, etc.</callout>

- Un message de QoS **niveau 2** « **Exactement une fois** ». QoS 2 est le niveau de service le plus élevé dans MQTT. Ce niveau garantit que chaque message est reçu une seule fois par les destinataires prévus. QoS 2 est le niveau de qualité de service le plus sûr et le plus lent. Ce niveau de service doit être utilisé si:
  - Les messages peuvent être délivrés lentement.
  - La duplication des messages provoque des problèmes.

<callout type=“primary” icon=“true”>La plupart des courtiers MQTT commerciaux ne prennent pas en charge le niveau de QoS 2 car il est lent et consomme plus de ressources.</callout>

1.3 Structure d'un paquet MQTT

- **Source** : <html>Comprendre la structure des paquets du protocole MQTT</html>

Le format de paquet ou de message MQTT se compose d'un en- tête fixe de 2 octets (toujours présent) + en -tête de variable (pas toujours présent) + charge utile (pas toujours présent).

2. Le broker Mosquitto

Eclipse Mosquitto est un **courtier de messages** (**broker**) open source (sous licence EPL / EDL) qui implémente les versions 3.1 et 3.1.1 du protocole MQTT. Mosquitto est léger et convient à une utilisation sur tous les appareils, des ordinateurs monocarte basse consommation aux serveurs complets.

Le protocole MQTT fournit une méthode légère pour effectuer la messagerie en utilisant un modèle de publication / abonnement. Cela le rend approprié pour la **messagerie Internet of Things**, par exemple avec des capteurs de faible puissance ou des appareils mobiles tels que des téléphones, des ordinateurs intégrés ou des microcontrôleurs.

Le projet Mosquitto fournit également une bibliothèque C pour l’implémentation des clients MQTT, ainsi que les très populaires clients MQTT **mosquitto_pub** et **mosquitto_sub**.

Mosquitto fait partie de la <html>Fondation Eclipse</html> et est un projet de <html>iot.eclipse.org</html>.

3. Installation et mise en oeuvre basique

3.1 Sur un Raspberry Pi

3.1.1 Installation

- **Mise à jour**

*.bash

```
sudo apt update && sudo apt upgrade -y
```

- **Installation du broker**

*.bash

```bash
sudo apt install mosquitto -y
```

- **Installation des utilitaires** en ligne de commande pour tester le broker

*.bash

```bash
sudo apt install mosquitto-clients -y
```

- **Affichage de la version installée**

*.bash

```bash
mosquitto_sub -v -h localhost -t '$SYS/broker/version'

# Exemple de résultat attendu
$SYS/broker/version mosquitto version 2.0.11
```

- **-v**, **–verbose** : message imprimé sous la forme de sujet.
- **-h**, **–host** : Spécifie l'hôte auquel se connecter. La valeur par défaut est localhost.
- **-t**, **–topic** : le sujet MQTT auquel on s'abonne.

<callout type=“info” icon=“true”>On remarque que l'outil **mosquitto_sub** ne rend pas la main et reste connecté au broker (carré noir). C'est le principe même du fonctionnement de MQTT lors d'un abonnement à un topic, rester à l'écoute. Pour se déconnecter, entrer le raccourci **CTRL-C**.</callout>

- **Quelques topics spécifiques**

“$SYS/broker/clients/connected”	Le nombre de clients connectés au broker.
“$SYS/broker/clients/maximum”	Le nombre maximum de clients connectés ayant été atteint.
“$SYS/broker/messages/received”	Le nombre total de messages reçus depuis que le broker a été démarré.
“$SYS/broker/uptime”	Le nombre de secondes écoulées depuis le démarrage.
“$SYS/broker/version”	La version du broker.

3.1.2 Arrêt, redémarrage

<callout type=“tip” icon=“true”>Le broker est installé en tant que **service**. Pour l'arrêter ou le redémarrer, utiliser les commandes suivantes :</callout>

*.bash

```bash
sudo systemctl stop mosquitto.service
```

```
sudo systemctl start mosquitto.service
```

3.1.3 Tests

<callout type="info" icon="true">Le paquet **mosquitto-clients** fournit deux commandes, <html>mosquitto_sub</html> pour une souscription et <html>mosquitto_pub</html> pour une publication.</callout>

3.1.3.1 Test sur le RaspBerry Pi (localhost)

Pour tester le bon fonctionnement du broker, nous allons publier le message (payload) "*Bonjour*" sur le canal d'information (topic) *test/val* à l'aide d'un client **mosquitto_pub**. Ce message sera reçu par un client **mosquitto_sub** abonné à *test/val*.

- **Abonnement**

*.bash

```
mosquitto_sub -v -h localhost -t test/val
```

- **Publication**

*.bash

```
mosquitto_pub -h localhost -t test/val -m "Bonjour"
```

- **-v**, **–verbose** : message imprimé sous la forme de sujet.
- **-h**, **–host** : Spécifie l'hôte auquel se connecter. La valeur par défaut est localhost.
- **-t**, **–topic** : le sujet MQTT auquel on s'abonne.
- **-m**, **–message** : Envoie un seul message à partir de la ligne de commande.

Résultat attendu

3.1.3.2 Test sur le réseau local

- **Ressource** : <html>Page de manuel de moustique.conf</html>

<callout icon="fa fa-hand-stop-o" color="red" title="STOP">A partir de la **version 2 de Mosquitto** seule la **connexion sur le réseau local** est acceptée par le broker.
Pour effectuer le test précédent entre un courtier situé sur une machine (PC, Raspberry Pi,etc.) et un

éditeur/abonné situé sur une autre machine (PC, smartphone, etc.) via un réseau local, il est nécessaire de modifier le fichier de configuration **mosquitto.conf** situé dans **/etc/mosquitto/**.
Pour cela :
- Ouvrir le fichier mosquitto.conf
- Ajouter les interfaces à l'aide de l'option **listener** </callout>

*.bash

```
# Ouvrir le fichier mosquitto.conf
sudo nano /etc/mosquitto/mosquitto.conf
```

Exemple : connexion au broker situé sur un Raspberry Pi, sur l'hôte local et sur le réseau local via les interfaces Ethernet et wifi.

Tests réalisés entre un RaspBerry pi et un smartphone

Consulter la page Wiki Réseau - Test d'un broker Mosquitto avec MyMQTT (Android App) pour la mise en oeuvre du test.

3.2. Sous Windows

Voir ce <html>lien</html> pour installer le broker Mosquitto sous Windows,

3.3 Sur un NAS Synology

- **Source** : <html>Installer le broker MQTT Mosquitto sur NAS Synology (DSM 6.2+) avec Docker</html>

Le fichier mosquitto.conf se situe dans le dossier **/usr/local/mosquitto/var**. Se connecter en **ssh**. Ouvrir **mosquitto.conf** avec nano et le compléter comme ci-dessous :

mosquitto.conf

```
# Write process id to a file.
protocol websockets # A ajouter
pid_file /var/packages/mosquitto/target/var/mosquitto.pid

# ================================================================
# Default listener
# ================================================================
# Port to use for the default listener.
```

```
port 1883
listener 9001 # A ajouter
```

4 Sécurité

<callout type="info" icon="true">Cette partie ne sera pas exploitée lors du développement sur un réseau local dans la salle de classe. A prendre en compte si l'accès au broker se fait via Internet.</callout>

4.1 Authentification

Ce paragraphe illustre la mise en sécurité de l'installation étudiée dans la partie "Découverte" de la page Mise en oeuvre d'un client MQTT sur un ESP8266 feather Huzzah. La mise en sécurité de cette installation passe par la mise en place d'une authentification. Les clients MQTT doivent s'authentifier avec un **identifiant / mot de passe**. La mise en place de cette authentification doit se faire côté Mosquitto (traitée ci-dessous) et côté client (voir Mise en oeuvre d'un client MQTT sur un ESP8266 feather Huzzah)

Fichier de configuration de Mosquitto
<callout type="warning" icon="true">Par défaut, le fichier de configuration **mosquitto.conf** d'un Raspberry Pi, situé dans **/etc/mosquitto/**, contient :</callout>

mosquitto.conf

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /var/run/mosquitto.pid
persistence true
persistence_location /var/lib/mosquitto/
log_dest file /var/log/mosquitto/mosquitto.log
include_dir /etc/mosquitto/conf.d
# -----------------------------------------
# A ajouter à partir de la version 2
# -----------------------------------------
listener localhost # actif par défaut mais à ajouter
listener @IP1      # si ajout d'@IP(s)
listener @IP2
# etc.
```

- ***mosquitto.pid*** : le fichier contenant le numéro de processus du démon Mosquitto permettant la gestion du fonctionnement en arrière-plan (mode serveur)
- ***persistence*** : la directive permettant au serveur de conserver l'état des connexions, des

abonnements et des messages dans un fichier enregistré sur le disque. Ceci permet au serveur de recharger ces informations en cas de redémarrage.
- ***persistence_location*** : l'emplacement où doivent être stockées ses informations.
- ***log_dest file*** : le chemin complet vers le fichier contenant le journal d'activité du serveur.
- ***include_dir*** : le répertoire contenant d'autres fichiers de configuration à prendre en compte.

<callout type="warning" icon="true">Pour installer l'authentification côté Mosquitto, il n'est pas nécessaire de modifier le fichier *mosquitto.conf*. Il suffit de sauvegarder les éléments de configuration supplémentaires dans le répertoire ciblé par ***include_dir*** (ici /etc/mosquitto/conf.d) sous la forme de fichiers.</callout>

Étape 1. Création d'un fichier de mots de passe
Pour créer un fichier de mots de passe, le paquet mosquitto fournit l'outil ***mosquitto_passwd***. Entrer la commande ci-dessous :

*.bash

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd sondes
```

Le Raspberry pi demande un mot de passe. Entrer ***mot2passe***.

- **-c** crée le fichier
- **sondes** est l'identifiant

En supprimant -c de la commande ci-dessus, il est possible :

- de changer le mot de passe d'un identifiant,
- d'ajouter une autre entrée au fichier en spécifiant un nouvel identifiant.

Étape 2. Création d'un fichier d'authentification auth.conf

- Entrer la commande ci-dessous :

*.bash

```
sudo touch /etc/mosquitto/conf.d/auth.conf
```

- Ouvrir le fichier avec nano.

*.bash

```
sudo nano  /etc/mosquitto/conf.d/auth.conf
```

- Ajouter le code ci-dessous

*.bash

```
password_file /etc/mosquitto/passwd
```

```
allow_anonymous false # Connexions sans mot de passe non autorisées

# password_file permet de spécifier le fichier de mots de passe à utiliser
# allow-anonymous autorise (**true**) ou non (**false**) les connexions anonymes (sans mot de passe)
```

- Redémarrer le serveur

*.bash

```
sudo systemctl restart mosquitto.service
```

Étape 3. Tests
Les tests ci-dessous sont à réaliser lorsque le croquis ***clientmqttesp8266.ino*** décrit à la page Mise en oeuvre d'un client MQTT sur un ESP8266 feather Huzzah a été modifié pour assurer l'authentification de la connexion.

- **Publication d'un message sur un topic avec une connexion sécurisée**

La commande de la LED de la carte ESP8266 peut se faire comme ci-dessous :

*.bash

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

ou

*.bash

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

- **Abonnement à un topic avec une connexion authentifiée**

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 :

*.bash

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

- *Exemple de résultat*

4.2 Chiffrage de la connexion

<html>A faire</html>

5 QoS

<callout type="primary" icon="true">Mosquitto implémente les trois qualités de service.</callout>

6. MQTT sur WebSockets avec Mosquitto

6.1 Pourquoi utiliser MQTT sur Websockets ?

<callout type="warning" icon="true">MQTT sur Websockets vous permet de **recevoir des données MQTT directement dans un navigateur Web**.</callout>

Le navigateur Web peut devenir l'INTERFACE pour afficher les données MQTT. Le support JavaScript de MQTT Websocket pour les navigateurs Web est fourni par le **client JavaScript**.

6.2 MQTT sur Websockets vs MQTT.

6.2.1 Présentation

Dans le cas de MQTT sur Websockets, la connexion websockets constitue un canal externe pour le protocole MQTT. Le courtier MQTT place le paquet MQTT dans un paquet websockets et l'envoie au client. Le client extrait le paquet MQTT du paquet websockets puis le traite comme un paquet MQTT normal.

<callout type="primary" icon="true">La **version 1.5.7** du **broker Mosquitto** pour Raspberry Pi OS est **compatible avec les Websockets**. Il faut configurer le fichier mosquitto.conf pour que la communication s'établisse entre le broker et un client.</callout>

6.2.2 Configuration du fichier mosquitto.conf

MQTT sur Websockets utilise généralement le port **9001** mais il n'est pas fixé.

- **Compléter** le fichier **mosquitto.conf** comme ci-dessous

mosquitto.conf

```
# Place your local configuration in /etc/mosquitto/conf.d/
#
```

```
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example
port 1883      # A ajouter
listener 9001  # A ajouter
protocol websockets # A ajouter
pid_file /var/run/mosquitto.pid
```

Cela crée un **écouteur supplémentaire** à l'aide de websockets et du port 9001.

- **Redémarrer** le serveur

*.bash

```
sudo systemctl restart mosquitto.service
```

6.2.3 Test de Websocket

Pour tester les websockets, nous avons besoin d'un client prenant en charge les websockets. Nous utiliserons le **client Javascript Paho** décrit sur la page Créer un client MQTT avec Eclipse Paho.

7. Outils

7.1 MyMQTT : un client pour Androïd

En installant un **client pour Androïd** tel que **MyMQTT** sur un smartphone, on pourra facilement vérifier la capacité du broker à recevoir ou à émettre des messages.

- **Fonctionnalités**
  - Connection à un courtier MQTT v3.1 (facultatif avec nom d'utilisateur et mot de passe)
  - Abonnement à des topics
  - Publication de messages
  - Enregistrement des messages

Consulter la page Wiki Réseau - Test d'un broker Mosquitto avec MyMQTT (Android App) pour sa mise en oeuvre.

7.2 MQTTlens : un client pour navigateur

MQTTlens est une application Google Chrome, qui se connecte à un courtier MQTT et peut s'abonner et publier sur des sujets MQTT.

Consulter la page Wiki Réseau - Test d'un broker Mosquitto avec MQTTlens pour sa mise en oeuvre.

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.