

MQTT : MQ Telemetry/Transport Un protocole pour l'IoT

Ph. Truillet

Novembre 2025 – v. 2.0



1. MQTT : comment ça marche ?

MQTT (<https://mqtt.org>) est un protocole ouvert, simple, léger et facile à mettre en œuvre. Ce protocole est idéal pour répondre aux besoins suivants :

- l'utilisation d'une très faible bande passante,
- l'utilisation sur les réseaux sans fils,
- la consommation en énergie,
- la rapidité avec un temps de réponse supérieur aux autres standards du web actuel,
- la fiabilité,
- et l'usage de faibles ressources processeurs et de mémoire.

MQTT est un standard ISO (ISO/IEC PRF 20922:2016¹ pour sa version 3.11) et OASIS² (pour la version 5.0).

Au départ, **TT** (premier nom du protocole) a été développé par Andy Stanford-Clark (IBM) et Arlen Nipper (Eurotech, actuellement Cirrus Link) en 1999 pour le contrôle d'un pipeline dans le désert. L'objectif était d'avoir un protocole de bande passante efficace utilisant peu d'énergie à un moment où les périphériques ne pouvaient être connectés qu'au travers de satellites.

Le protocole MQTT utilise une architecture « *publish/subscribe* » en contraste avec le protocole HTTP et son architecture « *request/response* » (cf. Figure 1).

Le point central de la communication est le **broker MQTT** (qui par défaut utilise le port **1883** pour les connexions non chiffrées et le port **8883** pour celles chiffrées en SSL/TLS), en charge de relayer les messages des émetteurs vers les clients. Chaque client s'abonne via un message vers le broker : le « *topic* » (sorte d'information de routage pour le broker) qui permettra au broker de réémettre les messages reçus des producteurs de données vers les clients. Les clients et les producteurs n'ont ainsi pas à se connaître, ne communiquant qu'au travers des topics. Cette architecture permet des solutions multi-échelles.

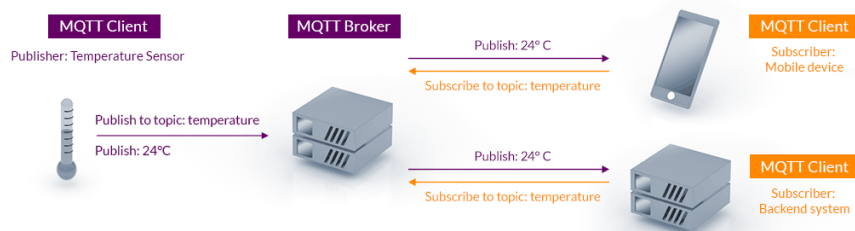


Figure 1 : Architecture MQTT «Publish/Subscribe» (tiré du site <https://mqtt.org>)

Chaque client MQTT a une connexion ouverte en permanence avec le broker. Si la connexion s'arrête, le broker bufférise les messages et les émet dès que la reconnexion avec le client est effectuée.

Un « **topic MQTT** » est une **chaîne de caractères** qui peut posséder une hiérarchie de niveaux séparés par le caractère « / ».

Par exemple, une information de température du salon pourrait être envoyée sur le topic « *maison/salon/temperature* » et la température de la cuisine sur « *maison/cuisine/temperature* ».

Nota : Les topics doivent au moins contenir un caractère (ne pas commencer par le caractère « \$ », réservé à un usage interne de MQTT), sont sensibles à la casse et le topic « / » est valide !

¹ <https://www.iso.org/standard/69466.html>

² <https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html>

Wildcards

- Le signe « + » est un caractère « *wildcard* » qui permet des valeurs arbitraires pour un niveau en particulier

Ex : « `maison/+/temperature` » permet de s'abonner aux températures de la cuisine et du salon

- le signe « # » pour plus d'un niveau (ce signe doit se trouver à la fin). Les messages envoyés peuvent être de toutes sortes mais ne peuvent excéder une taille de 256 Mo.

Ex : si un *publisher* s'abonne au topic « `maison/salon/#` », il recevra toutes les données provenant du salon. De même, s'il s'abonne au topic « `maison/#` », il collectera toutes les données des sondes de la maison.

2. Sécurité

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

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

3. QoS -Qualité de Service

MQTT intègre en natif la notion de **QoS**. En effet le *publisher* a la possibilité de définir la qualité de son message. Trois niveaux sont possibles :

- Un message de **QoS niveau 0** « *At most once* » sera délivré tout au plus une fois. Ce qui signifie que le message est envoyé sans garantie de réception, (le broker n'informe pas l'expéditeur qu'il l'a reçu et le message)
- Un message de **QoS niveau 1** « *At least once* » sera livré au moins une fois. Le client transmettra plusieurs fois s'il le faut jusqu'à ce que le Broker lui confirme qu'il a été transmis sur le réseau.
- Un message de **QoS niveau 2** « *exactly once* » sera obligatoirement sauvegardé par l'émetteur et le transmettra toujours tant que le récepteur ne confirme pas son envoi sur le réseau. La principale différence étant que l'émetteur utilise une phase de reconnaissance plus sophistiquée avec le broker pour éviter une duplication des messages (plus lent mais plus sûr).

Par défaut, **MQTT utilise la QoS de niveau 0.**

4. Exercices de chauffe

4.0 Préambule

Pour pouvoir fonctionner, vous allez devoir utiliser un broker MQTT. Vous pouvez facilement en installer un en allant sur le site <https://shiftr.io> et télécharger la **Desktop App** (disponible sous Linux, Windows et MacOS).

Nota : si cela ne fonctionne pas, vous pouvez vous rendre sur le site : <https://shiftr.io/try>

Dézippez et lancez le *DesktopApp* shiftr.io : un broker MQTT est lancé sr votre machine, prêt à recevoir vos messages ! Vous devriez voir une fenêtre s'ouvrir (cf. Figure 1)

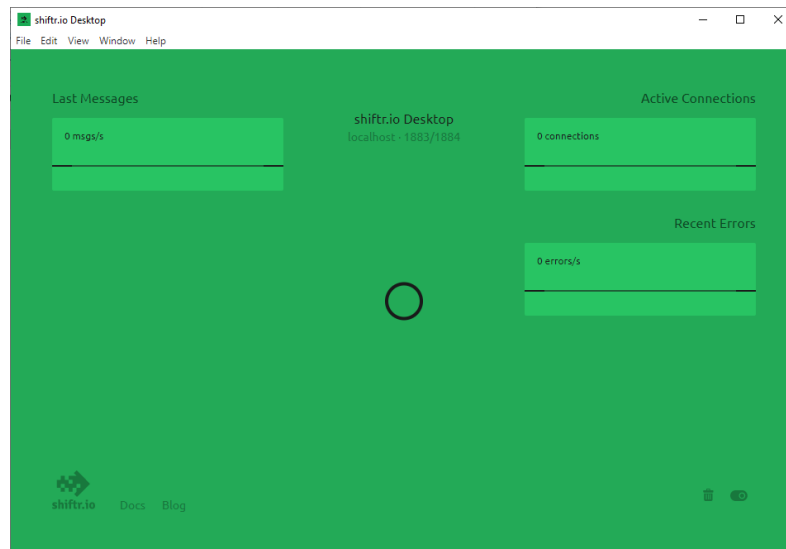



Figure 2 : DesktopApp – shiftr.io

4.1 Un premier exemple avec Processing.org

Ouvrez Processing, installez la librairie MQTT (Menu « **Outils** » | « **Ajouter un outil** », onglet « **Libraries** » MQTT).


MQTT | MQTT library for Processing based on the Eclips... Joel Gaehwiler

Ouvrez l'exemple installé avec la librairie MQTT (Menu « **Fichier** » | « **Exemples...** », | « **Contributed Libraries** » | « **MQTT** », exemple *PublishSubscribe*.

Remplacez l'url utilisée dans l'instruction `client.connect` ligne 20

« `mqtt://try:try@broker.shiftr.io` » par « `mqtt://localhost` »

Lancez le sketch. Appuyez sur la base espace dans le sketch Processing.org et visualisez le résultat sur le DesktopApp (cf. Figure 3)

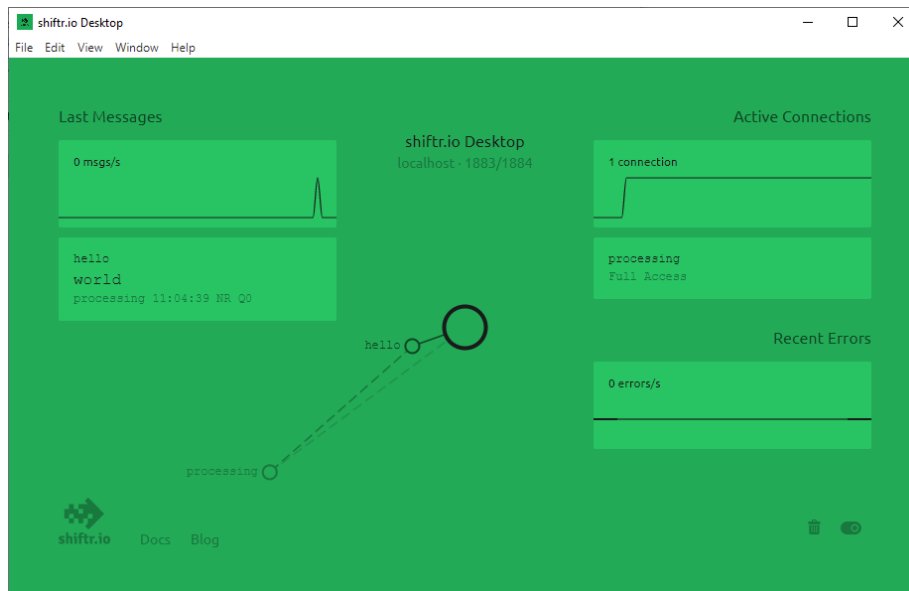


Figure 3 : Visualisation des messages MQTT “Publish/Subscribe”

Exercice :

- Ecrivez maintenant à partir de cet exemple un programme Processing.org qui génère des valeurs aléatoires de température toutes les secondes et les envoie au broker MQTT (**vous devez déterminer le topic**). Chaque instance lancée sera considérée comme une « pièce » de la maison)
- Ecrivez ensuite un programme qui devra s'abonner à l'ensemble des températures émises par chacune des instances et afficher la moyenne générale des températures de la « maison » (générée donc par chacune des pièces) et les moyennes par « pièce ».

4.2 Python (Paho)

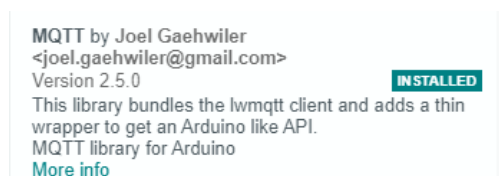
MQTT est **multi-langage** et on peut utiliser Python comme langage support (intéressant pour un Raspberry Pi par exemple). Pour utiliser MQTT sous Python, allez voir la documentation ici : <https://pypi.python.org/pypi/paho-mqtt>

- Ecrivez le même programme émetteur de données aléatoires en Python.

4.3 Arduino

De la même manière, il est possible de générer des messages MQTT depuis un périphérique compatible arduino connecté à TCP/IP (via un réseau filaire ou en WiFi)

Il faut dans un premier temps récupérer la librairie MQTT en faisant « **Croquis | Importer une bibliothèque | Gérer les bibliothèques** » et chercher la bibliothèque **MQTT** (Arduino 1.8.7 et ultérieur)



Vous trouverez dans la section « *Exemples* » du code permettant d'émettre des données depuis arduino directement sur MQTT.

Nota : S'il manque des périphériques arduino de type ESP8266 ou ESP32, vous pourrez peut-être sauter cette étape.

Vous trouverez un exemple de code pour **ESP8266** ici →

https://github.com/truillet/upssitech/blob/master/SRI/3A/ID/TP/Code/UPSSITECH_staton_meteo.zip

Nota2 : le code utilise un NodeMCU (ESP8266) et un capteur d'humidité et de température DHT22

4.4 Un broker MQTT

Le broker MQTT le plus connu et le plus utilisé reste **Mosquitto** (géré par la fondation **eclipse**)



Vous pouvez télécharger **Mosquitto** ici : <https://mosquitto.org>

Vous avez aussi accès au broker MQTT « UPSSITECH » en ligne à l'adresse IP **mqtt.upssitech.fr** (les ports 1883 et 9001 [websockets] sont disponibles avec les identifiants *login* : upssitech/ *password* : 2011) pour vos tests.

5. Bureau d'Etudes (par groupe de 5 à 7) : un superviseur de capteurs réels et virtuels

Il s'agit maintenant de créer, de gérer et superviser un réseau de capteurs à partir de données « **réelles locales** » (température, pression, luminosité, taux d'humidité, ...) et à distance (en utilisant par exemple des données récupérées via une **API REST**). Pour ce faire :

1. créez un client MQTT qui récupère les données de la météo locale via l'API **OpenWeatherMap** et les envoie sur le broker.
2. créer des clients MQTT capables d'envoyer sur le broker le nombre de « *VélÔToulouse* » disponibles dans les stations les plus proches de l'UPSSITECH (**API JCDecaux**) et de l'heure de passage des prochains bus à la station « *Sports Universitaires* » près du U3 (**API Tisseo**)
3. créez des clients MQTT à partir de micro-contrôleurs compatibles **arduino** (ESP) et de capteurs de luminosité, température (ou autre) ... et envoyez les valeurs vers le broker MQTT.
4. écrivez un client tiers capable d'agréger les valeurs des différents capteurs positionnés dans l'environnement (ex : toutes les températures d'une pièce) et de renvoyer périodiquement ces valeurs sur le broker (vous déterminerez les topics à utiliser).
5. proposez enfin un *dashboard* de vos données en javascript, Python, ... (et MQTT ou MQTT over websocket) permettant de suivre l'évolution des données dans le temps et déclencher des alertes à l'utilisateur.

Exemples d'API REST intéressantes :

- **OpenWeatherMap**, <https://openweathermap.org/api>
- **Tisseo**,
<https://data.toulouse-metropole.fr/explore/dataset/api-temps-reel-tisseo/api>
- **JCDecaux**, <https://developer.jcdecaux.com>