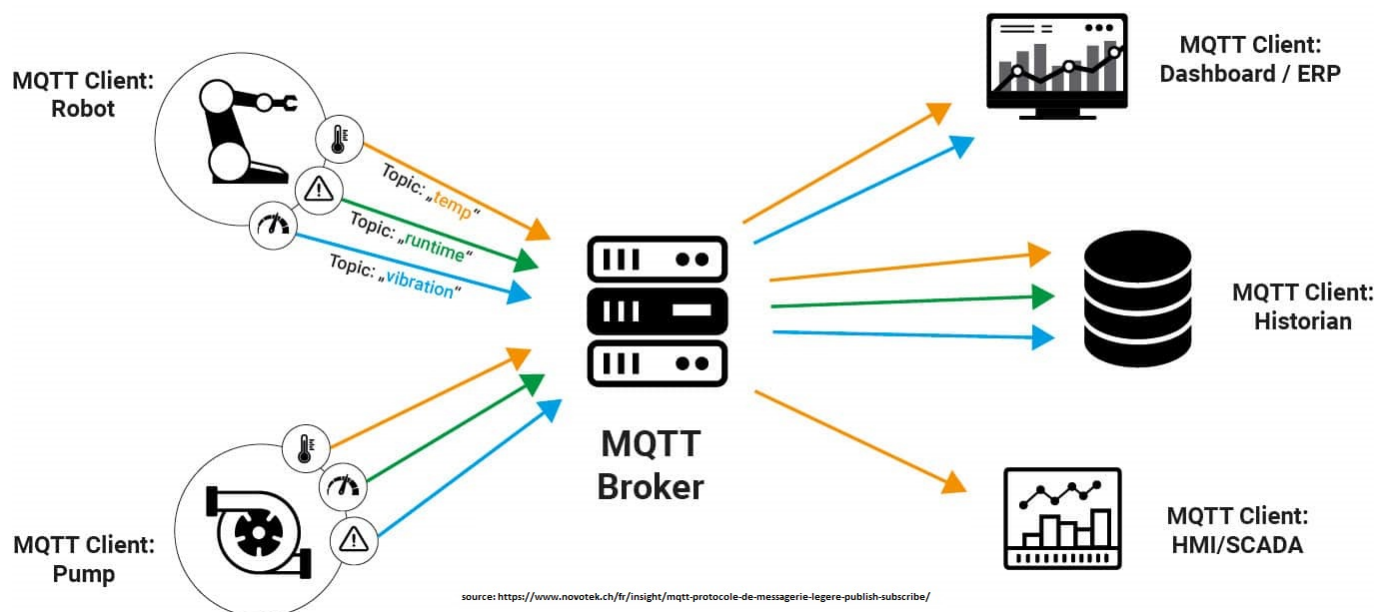


# MQTT

Le protocole MQTT (Message Queuing Telemetry Transport) est un protocole de communication léger conçu pour les systèmes nécessitant des échanges de données rapides et à faible bande passante, comme l'Internet des objets (IoT).

Développé dans les années 1990, MQTT repose sur une architecture de type "publish-subscribe", où les appareils communiquent via un serveur central appelé "broker".



Le protocole MQTT est particulièrement adapté aux environnements avec une bande passante limitée ou des connexions instables, car il consomme peu de ressources et peut fonctionner avec un minimum de données.

Dans ce modèle, les clients (appareils) ne communiquent pas directement entre eux. Ils se connectent au **broker** en TCP/IP et publient des messages sur des **topics** (sujets) spécifiques. D'autres clients peuvent alors s'abonner à ces topics pour recevoir les messages. Par exemple, un capteur de température peut publier des données sur le topic "moteur/température", et tous les appareils abonnés à ce topic recevront les mises à jour.

Les topics sont hiérarchiques, séparés par des barres obliques (/). Chaque niveau de la hiérarchie apporte des détails supplémentaires sur le sujet du message, ce qui permet d'organiser et de filtrer les informations.

Exemple: Prenons le topic `maison/salon/temperature`

`maison` est le premier niveau, qui indique un bâtiment spécifique. `salon` est le deuxième niveau, indiquant une pièce précise (ici, le salon). `temperature` est le troisième niveau, qui précise le type d'information (la température).

## Utilisation des Wildcards

MQTT permet d'utiliser des caractères génériques (wildcards) pour s'abonner à plusieurs topics en même temps :

- **+** pour un niveau : s'abonner à **maison/+temperature** signifie que l'on recevra tous les messages de température de toutes les pièces de la maison (maison/salon/temperature, maison/cuisine/temperature, etc.).
- **#** pour plusieurs niveaux : s'abonner à **maison/#** permet de recevoir tous les messages de la maison, peu importe la pièce et le type de données.

## Qualité de Service (QoS)

MQTT offre plusieurs fonctionnalités permettant d'assurer un service malgré des événements imprévus.

### QoS

MQTT offre trois niveaux de QoS en ce qui concerne la livraison des messages :

- QoS 0 : les messages sont envoyés sans garantie de réception.
- QoS 1 : les messages sont livrés au moins une fois.
- QoS 2 : les messages sont garantis d'être livrés une seule fois.

### Last Will and Testament (LWT)

Le Last Will and Testament (LWT) est un mécanisme conçu pour informer les autres appareils lorsque l'un des clients est déconnecté de manière inattendue.

Au moment de sa connexion au broker, le client MQTT peut définir un message LWT qui sera stocké par le broker jusqu'à ce que le client se déconnecte.

- Si le client se déconnecte correctement (en envoyant un message de fin de session, appelé DISCONNECT), le broker sait qu'il n'est plus actif et le message LWT n'est pas envoyé.
- Si le client perd sa connexion de façon imprévue (par exemple, en cas de panne de réseau ou de coupure de courant), le broker considère cela comme une déconnexion anormale et publie automatiquement le message LWT sur un topic défini.

### Keep-Alive

Le paramètre Keep-Alive est un intervalle de temps défini par le client MQTT au moment de sa connexion au broker. Ce paramètre indique la fréquence à laquelle le client doit envoyer des messages de type PINGREQ au broker pour signaler qu'il est toujours en ligne. En retour, le broker répond avec un message PINGRESP. Si le broker ne reçoit pas de PINGREQ dans l'intervalle défini, il considère que le client est déconnecté et peut déclencher le message LWT si un LWT est configuré.

### Session Persistante

MQTT offre la possibilité de maintenir une session persistante pour les clients qui se déconnectent temporairement. Lorsqu'un client utilise une session persistante, le broker garde en mémoire certains états importants, comme les abonnements et les messages non délivrés. Cela signifie que si le client se reconnecte plus tard, il pourra récupérer tous les messages publiés sur ses topics d'abonnement pendant sa déconnexion. Ce mécanisme est particulièrement utile pour les appareils qui peuvent se connecter de façon intermittente, comme les capteurs sur batterie.

### Retained Messages (Messages conservés)

Les retained messages permettent de conserver le dernier message publié sur un topic pour les nouveaux clients qui s'abonnent à ce topic après la publication. Si un capteur publie une donnée importante, comme une alerte, cette donnée restera disponible pour tout nouvel abonné, ce qui assure une certaine persistance des informations.

## Bufferisation

Lorsqu'un client perd sa connexion au broker, il peut décider de continuer à fonctionner normalement, ce qui va générer de nouveaux messages à émettre. Ces messages sont stockés localement dans une queue, ils seront envoyés dans le bon ordre au moment où le client se sera reconnecté au broker.

## NoLocal

Par défaut, un client qui envoie un message le reçoit aussi (s'il est abonné au topic correspondant). 'NoLocal' empêche le serveur MQTT de renvoyer un message au client qui l'a publié. Quand 'NoLocal' est activé (valeur 1), le serveur ne transmettra pas les messages au client qui en est l'origine.

```
var mqttSubscribeOptions = mqttFactory.CreateSubscribeOptionsBuilder()
    .WithTopicFilter(t => t
        .WithTopic("mon/topic")
        .WithNoLocal()) // Active l'option NoLocal
    .Build();

await mqttClient.SubscribeAsync(mqttSubscribeOptions, CancellationToken.None);
```

## Cas d'usage typique

Cette option est particulièrement utile dans les scénarios de pont (bridging) entre brokers MQTT, pour éviter des boucles infinies de messages lorsqu'un broker se connecte à un autre en tant que client...

Exemple concret :

```
Un client publie sur le topic sensors/temperature
Ce même client s'abonne à sensors/temperature avec NoLocal activé
Résultat: Le client ne recevra PAS ses propres messages, mais recevra bien ceux
des autres clients
```

Note importante: NoLocal est une fonctionnalité MQTT 5.0