SOMMAIRE

1	Introduction	2
2	Le fonctionnement de MQTT	2
3	MQTT et la QOS	3
4	Installer un broker MQTT sur Linux	3
5	Installer MQTT client	4
6	Utiliser MQTT en javascript	5
7	Pour aller plus loin : retour sur le QoS	5
8	Pour aller plus loin : le wildcard	7
9	Pour aller plus loin : Le "Last Will and Testament" (LWT)	8

1. Introduction :

MQTT, pour "Message Queuing Telemetry Transport", est un protocole open source de messagerie qui assure des communications non permanentes entre des appareils par le transport de leurs messages.

Il a été créé en 1999 par Andy Stanford-Clark, ingénieur chez IBM, et Arlen Nipper, chez EuroTech, principalement dans la communication M2M pour permettre à deux appareils utilisant des technologies différentes de communiquer.

Devenu une norme ISO en 2016, MQTT connectait déjà à cette date des millions d'appareils dans le monde entier, dans toutes sortes d'applications et d'industries.

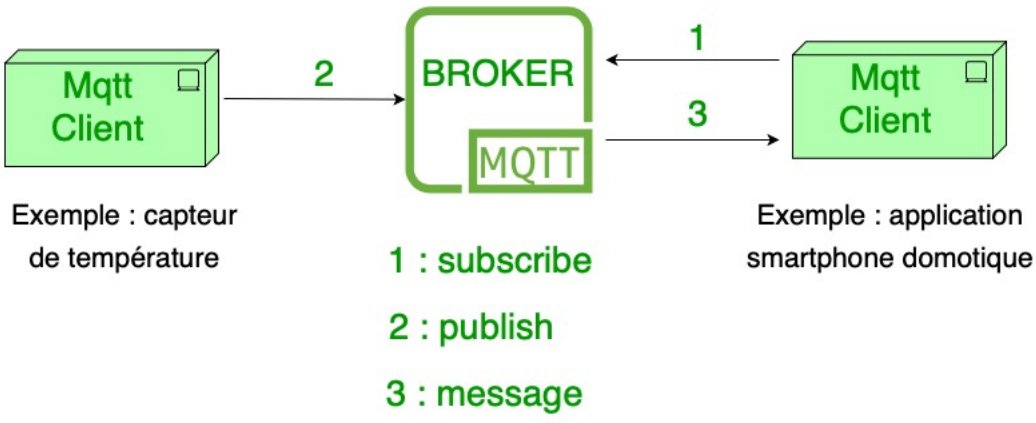
Les géants du web parmi lesquels AWS ou Microsoft utilisent MQTT pour remonter les données sur leur plateforme cloud.

2. Le fonctionnement de MQTT :

Le protocole MQTT est assez basique dans son fonctionnement, c'est d'ailleurs sa force. Il s'agit d'un protocole de messagerie machine to machine. Cela permet des échanges simples, rapides et fiables et c'est tout ce que l'on demande à des objets connectés.

Son fonctionnement est très logique. Quand on parle MQTT, on parle de connexion client-serveur avec abonnement. Concrètement les équipements connectés publient et/ou s'abonnent à un **topic** qui référence les messages et les communique aux abonnés.

Prenons un exemple concret, nous disposons d'un ensemble capteur de température connectée et d'une application smartphone domotique. Nous allons configurer le capteur en publication et notre application en tant qu'abonnée.



Le capteur de température va alors publier les valeurs mesurées dans un topic prédéfini "maison/chambre/temp" et notre application sera abonnée à ce topic "maison/chambre/temp" et consultera alors tous les messages (relevés de température) qui y seront publiés.

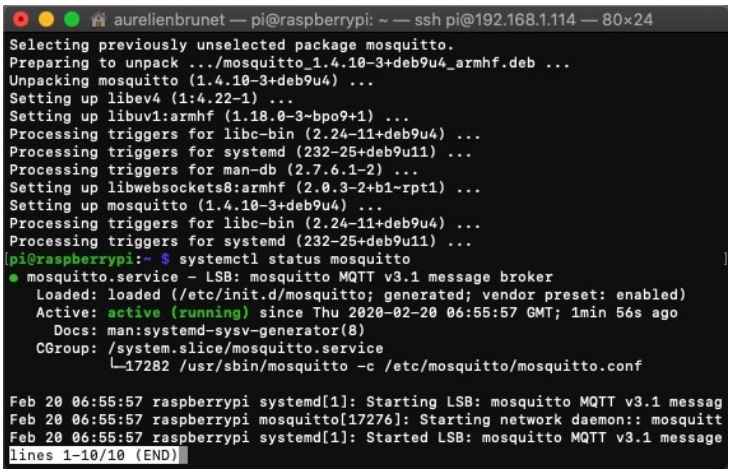
3. MQTT et la QOS :

La QoS représente la Qualité of Service. Il existe plusieurs niveaux de qualité de service avec MQTT Allant du néant (niveau 0), un message est publié sans garantie de sa bonne réception, au niveau 2 aussi appelé le service garantie, qui lui se charge de publier le message en deux parties qui s'assure et garantit, que le message sera correctement délivré de façon unique, quel que soit le nombre de tentatives.

Bien entendu, plus vous montez en niveau de QoS, plus vous chargez les trames.

4. Installer un broker MQTT sur une machine LINUX :

```
sudo apt-get install mosquitto
```



Puis vérifiez que tout est opérationnel avec la commande suivante.

```
systemctl status mosquitto
```

Le service peut être redémarré avec :

```
sudo systemctl restart mosquitto
```

On peut modifier le fichier de configuration avec :

```
Sudo nano /etc/mosquitto/mosquitto.conf
```

On peut faire en sorte qu'il faille un login/mot de passe pour pouvoir se connecter à Mosquitto.

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

La commande mosquitto_passwd permet d'ajouter un utilisateur dans le fichier de mot de passe de Mosquitto. Une fois la commande saisie, l'utilitaire demande le mot passe.

Le drapeau -c permet de créer le fichier passwd. Ce dernier sera écrasé s'il existe déjà.

Par la suite, il est possible de lister les utilisateurs en affichant le contenu du fichier à l'aide de la commande more /etc/mosquitto/passwd.

Il faut ensuite modifier le fichier mosquitto.conf en ajoutant à la fin :

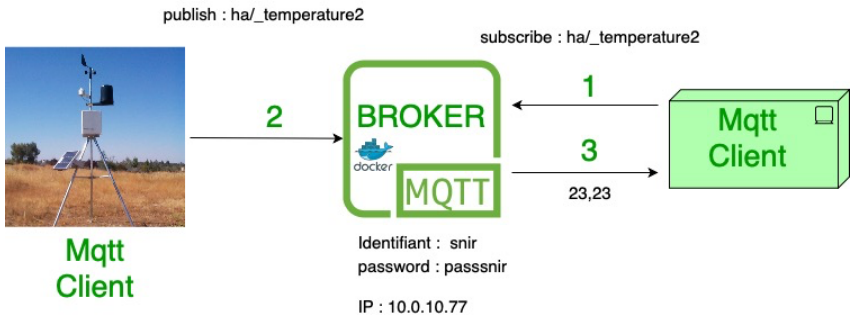
```
allow_anonymous false
password_file /etc/mosquitto/passwd
```

5. Installer MQTT client :

```
Sudo snap install mosquitto
```

Maintenant, testons notre broker.

Un broker MQTT est présent sur le réseau SNIR à l'adresse 10.0.10.77. c'est un conteneur docker.



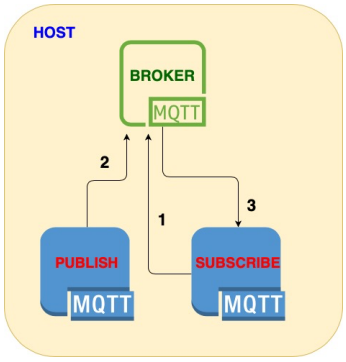
La station météo y publie ses mesures sur différents topics.
Température : ha/_temperature2
Humidité : ha/humidity
Pression atmosphérique : ha/pressure

Identifiant/mot de passe : snir/passsnir

Pour s'abonner et recevoir les mesures de température :

```
mosquitto_sub -h 10.0.10.77 -u 'snir' -P 'passsnir' -t 'ha/_temperature2'
```

Pour publier un message sur un broker, test depuis votre machine à la fois broker et cliente.



Lancer une fenêtre terminale et saisir :

```
mosquitto_sub -h localhost -t topic/test
```

Dans une autre fenêtre terminale, saisir :

```
mosquitto_pub -h localhost -t topic/test -m "Voici un test"
```

Vous devez recevoir votre message

6. Utiliser MQTT en Javascript :

```
const mqtt = require('mqtt');

const server = 'mqtt://10.0.10.77:1883';
const options = {
    clientId: 'bouhenic',
    username: 'snir',
    password: 'passsnir'
};

const client = mqtt.connect(server, options);

client.on('connect', () => {
    console.log('Connected to MQTT broker');
    client.subscribe('ha/_temperature2', { qos: 0 });
});

client.on('message', (topic, message) => {
    console.log(`${topic} ${message.toString()}`);
});

client.on('error', (err) => {
    console.error('Connection error:', err);
    client.end();
});
```

7. Pour aller plus loin : retour sur le QoS

Le QoS (Quality of Service) est un élément clé du protocole MQTT qui détermine comment les messages sont livrés dans diverses conditions. Il existe trois niveaux de QoS : 0, 1 et 2. Chacun offre un niveau de garantie de livraison différent.

QoS 0 : Au plus une fois (At Most Once)

Description :

- Lorsque le niveau QoS est défini sur 0, cela signifie que le message est livré au destinataire au plus une fois.
- Il n'y a aucune garantie de livraison, et il n'y a pas d'accusé de réception envoyé par le destinataire.

- Le message est envoyé une seule fois, et aucune autre tentative n'est faite pour s'assurer que le destinataire l'a reçu.
- C'est le niveau de QoS le plus rapide car il n'implique qu'une seule transmission, mais il est également le moins fiable.

Scénarios d'utilisation :

- QoS 0 est approprié pour les situations où la perte occasionnelle de messages est acceptable. Par exemple, pour les données de capteurs qui sont fréquemment mises à jour, comme la température ou l'humidité, où la perte d'une lecture n'est pas critique.
- Il est également utile pour les situations où la latence est une préoccupation majeure et où la garantie de livraison n'est pas essentielle.

Avantages :

- Faible latence : Comme il n'y a pas de processus d'accusé de réception, la transmission est rapide.
- Moins de bande passante : Seul le message initial est envoyé, sans transmissions supplémentaires pour les accusés de réception.

Inconvénients :

- Pas de garantie de livraison : Les messages peuvent être perdus en cas de problèmes de réseau ou d'autres interruptions.
- Pas d'accusé de réception : L'expéditeur ne sait pas si le message a été reçu ou non.

QoS 1 : Au moins une fois (At Least Once)

Le niveau QoS 1 garantit que le message est livré au destinataire au moins une fois, mais il est possible qu'il soit livré plusieurs fois en cas de problèmes de réseau ou d'autres interruptions.

Voici comment cela fonctionne :

- PUBLISH : Le client émetteur envoie un message avec un identifiant de paquet unique.
- PUBACK : Le client récepteur reçoit le message et envoie un accusé de réception (PUBACK) avec le même identifiant de paquet pour informer l'émetteur qu'il a reçu le message.
- Si le client émetteur ne reçoit pas le PUBACK dans un délai spécifié, il peut décider de renvoyer le message, d'où la possibilité de recevoir le message plusieurs fois.

Pourquoi utiliser QoS 1 ?

- Fiabilité : QoS 1 est un bon compromis entre performance et fiabilité. Il garantit que le message est livré, mais sans le coût supplémentaire du processus à quatre voies de QoS 2.
- Duplication : L'inconvénient est que le message peut être livré plusieurs fois si le PUBACK est perdu ou si le client émetteur renvoie le message. Les clients doivent être conçus pour gérer les messages en double. Par exemple, si un message commande à un appareil de s'allumer, recevoir le message deux fois n'est pas un problème. Cependant, si un message incrémente une valeur ou déclenche une transaction, la duplication pourrait causer des problèmes.

- Utilisation courante : QoS 1 est souvent utilisé dans les scénarios où la garantie de livraison est essentielle, mais où la duplication potentielle des messages n'est pas un problème majeur.

QoS 2 : Exactement une fois (Exactly Once)

Le niveau QoS 2 est le niveau de garantie le plus élevé et assure que le message est livré exactement une fois. C'est le niveau de QoS le plus sûr, mais aussi le plus lent, car il nécessite plusieurs échanges de messages pour garantir la livraison.

Voici comment cela fonctionne :

- PUBLISH : Le client émetteur envoie un message avec un identifiant de paquet unique.
- PUBREC : Le client récepteur reçoit le message et envoie un accusé de réception (PUBREC) avec le même identifiant de paquet pour informer l'émetteur qu'il a reçu le message.
- PUBREL : Le client émetteur reçoit le PUBREC et envoie un message PUBREL pour informer le récepteur qu'il peut libérer les ressources associées au message.
- PUBCOMP : Le client récepteur reçoit le PUBREL et envoie un message PUBCOMP pour confirmer la fin de l'échange.

Ce processus à quatre voies garantit que le message est livré exactement une fois, même si l'une des parties est déconnectée ou rencontre d'autres problèmes pendant la transmission. Si l'une des étapes échoue, le processus est répété jusqu'à ce que la livraison soit garantie.

Pourquoi utiliser QoS 2 ?

- Fiabilité : Si la garantie que chaque message est reçu exactement une fois est essentielle pour votre application, alors QoS 2 est le bon choix. Par exemple, si vous contrôlez un appareil ou effectuez une transaction où la réception de messages en double pourrait causer des problèmes, QoS 2 est recommandé.
- Coût : Cependant, cette fiabilité a un coût en termes de bande passante et de temps de traitement. Le processus à quatre voies nécessite plus de messages et donc plus de bande passante. De plus, chaque étape nécessite du temps, ce qui peut introduire un léger retard.
- Considérations : Il est important de noter que, bien que QoS 2 garantisse la livraison exactement une fois, il ne garantit pas l'ordre de livraison. Si l'ordre des messages est crucial, d'autres mécanismes doivent être mis en place.

8. Pour aller plus loin : le wildcard

Dans le contexte de MQTT, un "wildcard" (ou caractère générique) est un caractère spécial utilisé dans les topics pour s'abonner à plusieurs topics à la fois. MQTT propose deux wildcards : + et #.

- + (Wildcard à un niveau) :

Le caractère + peut remplacer un seul niveau de hiérarchie dans un topic.
Par exemple, si vous vous abonnez au topic maison/+/lumière, vous recevrez des messages de maison/salon/lumière, maison/cuisine/lumière, etc., mais pas de maison/salon/lumière/plafond.

- # (Wildcard multi-niveaux) :

Le caractère # peut remplacer plusieurs niveaux de hiérarchie dans un topic.
Il doit toujours être utilisé à la fin d'un topic.
Par exemple, si vous vous abonnez au topic maison/#, vous recevrez des messages de tous les topics qui commencent par maison/, comme maison/salon/lumière, maison/cuisine/thermostat, maison/jardin/arrosage, etc.

Exemples d'utilisation des wildcards :

- S'abonner à tous les capteurs d'une maison : maison/capteurs/#
- S'abonner à toutes les lumières de la maison : maison/+/lumière
- S'abonner à tous les événements d'une pièce spécifique : maison/salon/#

Points à retenir :

- Les wildcards ne peuvent être utilisés que lors de l'abonnement et non lors de la publication.
- L'utilisation judicieuse des wildcards peut grandement simplifier la logique d'abonnement, mais il est essentiel de comprendre leur comportement pour éviter des abonnements non désirés.

9. Pour aller plus loin : Le "Last Will and Testament" (LWT)

Le "Last Will and Testament" (LWT), ou "Testament et Dernières Volontés" en français, est une fonctionnalité du protocole MQTT qui permet à un client de spécifier un message qui sera envoyé par le broker en son nom, dans le cas où il se déconnecterait inopinément.

C'est une manière pour les autres clients abonnés de savoir qu'un client particulier n'est plus disponible ou a rencontré un problème.

Voici comment cela fonctionne :

- Définition du LWT : Lorsqu'un client MQTT se connecte à un broker, il peut spécifier son "testament" : un topic, un message, un QoS et un drapeau "retain".
- Détection de la déconnexion : Si le broker détecte que le client s'est déconnecté sans envoyer de message de déconnexion propre (par exemple, en raison d'une perte de connexion ou d'un crash du client), le broker considère que le client est déconnecté de manière inattendue.
- Publication du LWT : Lorsque cela se produit, le broker publie automatiquement le message spécifié par le client sur le topic défini, avec le QoS et le drapeau "retain" indiqués.

Exemple d'utilisation :

Imaginons un capteur de température connecté via MQTT. Lorsqu'il se connecte au broker, il peut définir son LWT pour publier le message "Offline" sur le topic "capteur/etat". Si le capteur perd sa connexion ou s'éteint soudainement, le broker publiera le message "Offline" sur le topic "capteur/etat", informant ainsi les autres clients que le capteur n'est plus en ligne.

Pourquoi c'est utile ?

- Détection rapide des défaillances : Les autres clients ou systèmes peuvent rapidement détecter qu'un appareil ou un service est défaillant ou hors ligne.
- Automatisation : En fonction du message LWT, des actions automatisées peuvent être déclenchées, comme l'envoi d'alertes ou la mise en marche d'un système de secours.

- Gestion centralisée : Plutôt que de compter sur chaque client pour signaler son état, le broker centralise cette logique, garantissant une détection fiable des déconnexions inattendues.

Manipulation :

Étape 1 : Connexion avec un LWT

1 Pour configurer un LWT lors de la connexion, vous utiliserez le client mosquitto_sub (puisque vous voulez vous abonner à un topic et attendre les messages). Lors de la connexion, vous spécifierez le testament avec les options -t, -m et -q pour le topic, le message et le QoS, respectivement.

```
mosquitto_sub –h localhost –t some/topic –q 1 ––will–topic
"testament/topic" ––will–payload "Client unexpectedly disconnected"
––will–qos 1
```

- -h localhost : spécifie l'adresse du broker (dans cet exemple, le broker est sur la même machine).
- -t some/topic : le topic auquel vous souhaitez vous abonner.
- -q 1 : le niveau de QoS pour l'abonnement.
- --will-topic "testament/topic" : le topic sur lequel le message du testament sera publié si le client se déconnecte inopinément.
- --will-payload "Client unexpectedly disconnected" : le message à publier en cas de déconnexion inattendue.
- --will-qos 1 : le niveau de QoS pour le message du testament.

Étape 2 : Tester le LWT

1 Dans un autre terminal, abonnez-vous au topic du testament pour écouter le message en cas de déconnexion inattendue du premier client :

```
mosquitto_sub –h localhost –t testament/topic –q 1
```

2 Fermez maintenant le premier terminal (celui avec le client ayant le LWT) de manière inattendue, par exemple en utilisant CTRL+C ou en fermant simplement la fenêtre.

3 Dans le second terminal (celui abonné au topic du testament), vous devriez voir le message "Client unexpectedly disconnected", indiquant que le LWT a été déclenché et que le message du testament a été publié par le broker.

Cela confirme que le LWT fonctionne comme prévu : en cas de déconnexion inattendue du client, le broker publie le message du testament sur le topic spécifié, informant ainsi les autres clients de la déconnexion.