Raspberry Pi - Installer un broker (serveur) MQTT Mosquitto

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

- **Sources**
  - Fundamentals of MQTT
  - **Hackable Magazine n°26** : *"Faites communiquer vos projets simplement avec MQTT"*
  - Généralités sur Wikipédia
  - Site de référence mqtt.org
  - Eclipse Mosquitto An open source MQTT broker
  - Série d'articles sur hivemq.com
  - Using MQTT Over WebSockets with Mosquitto
- **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

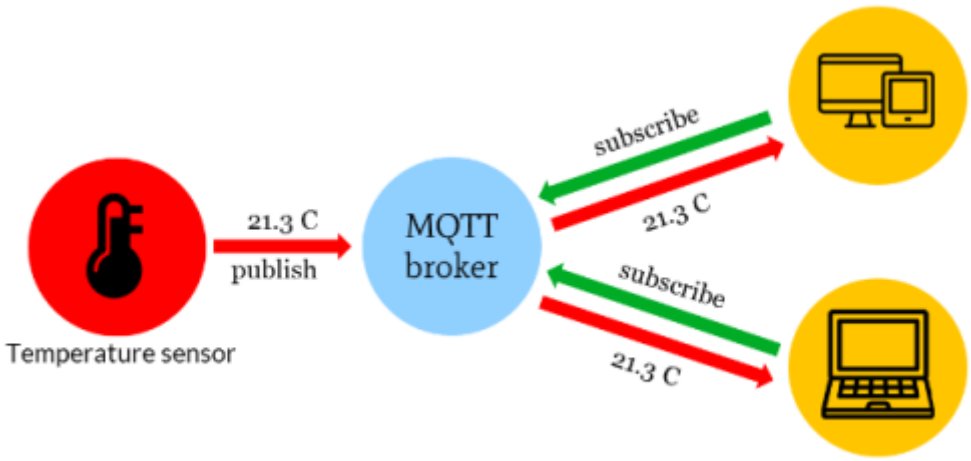
Le protocole MQTT (MQ Telemetry Transport) trouve ses origines en 1999 dans les travaux de **Andy Standford-Clark** et **Arlen Nipper**, alors qu'ils travaillaient pour IBM au développement d'un protocole pour une utilisation industrielle de télémétrie en lien avec l'industrie pétrolière.

1.2 Principes

1.2.1 Organisation et communication

> **MQTT** est un **service de publication/abonnement** TCP/IP simple et extrêmement léger. Il fonctionne sur le principe **client/serveur**.

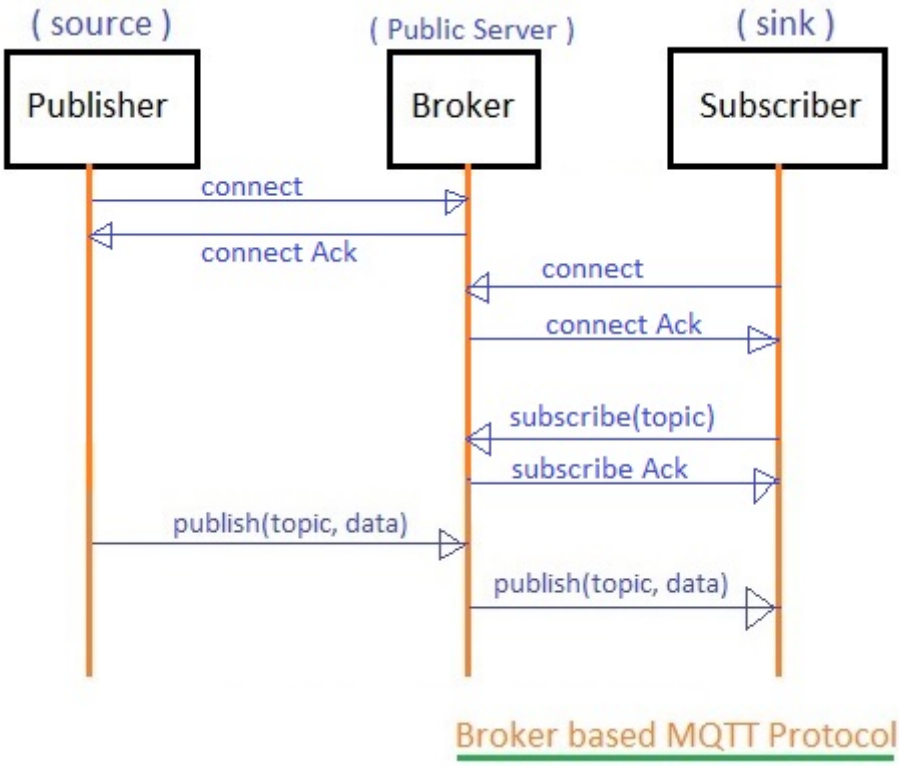
Le **serveur** ou **courtier**, nommé **broker**, va collecter des informations que les **éditeurs** (**publishers**) vont lui transmettre. Certaines informations collectées par le broker seront renvoyées à certains **abonnés** (**subscribers**) ayant préalablement fait la demande au broker. Un **client**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



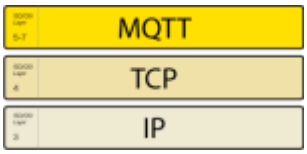
Schematic data flow from sensor (machine) to devise (machine)

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.

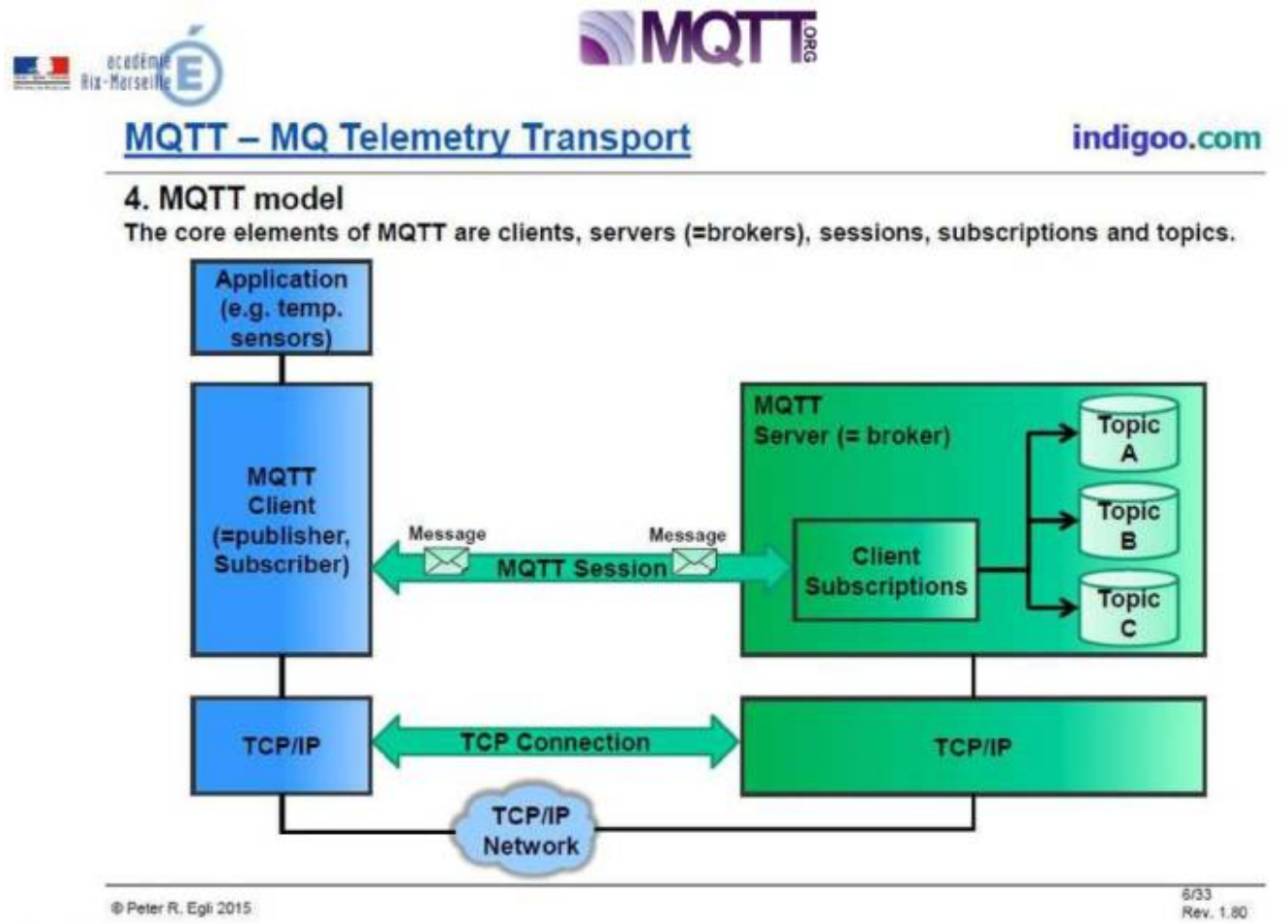
Les **messages** envoyés par les éditeurs peuvent être de toute sorte, mais ne peuvent excéder une taille de **256 Mo** bien que dans les **mises en œuvre réelles**, le maximum soit de **2 à 4 Ko**.



Résumé



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***.

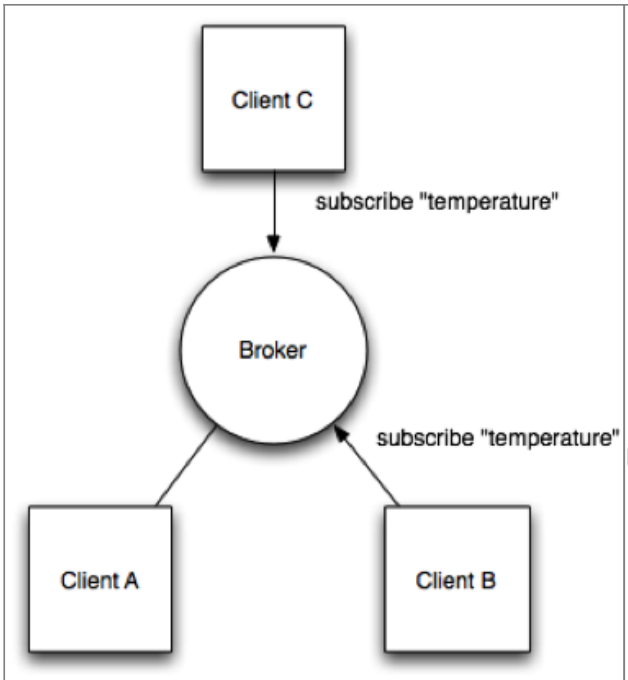


19

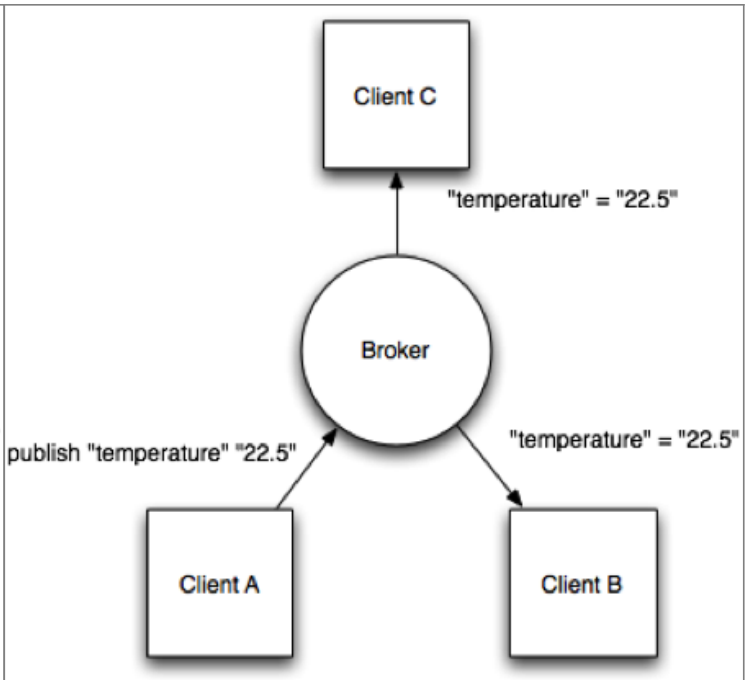
1.2.2 Les topics

> Un **topic** est une simple **chaine de caractères**, mais qui peut être **structurée hiérarchiquement**.
> Exemple : **maison/salon/temperature**

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.

> Cette écriture hiérarchique permet à un abonné de souscrire à un ensemble de topics en utilisant des **caractères joker** (**+**, **#**).

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.

> Voir les **bonnes pratiques** d'écriture des topics sur hivemq.com

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)

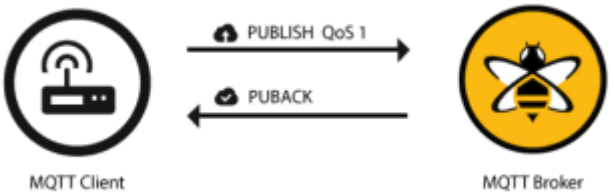
MQTT intègre en natif la notion de QoS. En effet, le publisher à la possibilité de définir la qualité de son message.

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.

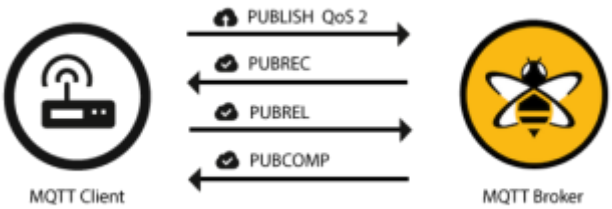
- Un message de QoS **niveau 1** « **Au moins une fois** ». Le niveau de qualité de service 1 garantit qu'un message est remis au moins une fois au destinataire. L'expéditeur stocke le message jusqu'à ce qu'il reçoive du destinataire un paquet PUBACK qui accuse réception du message. Il est possible qu'un 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.



> MQTT QoS niveau 1 est utilisé dans les courtiers MQTT commerciaux comme AWS IoT, Azure, etc.

- 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.

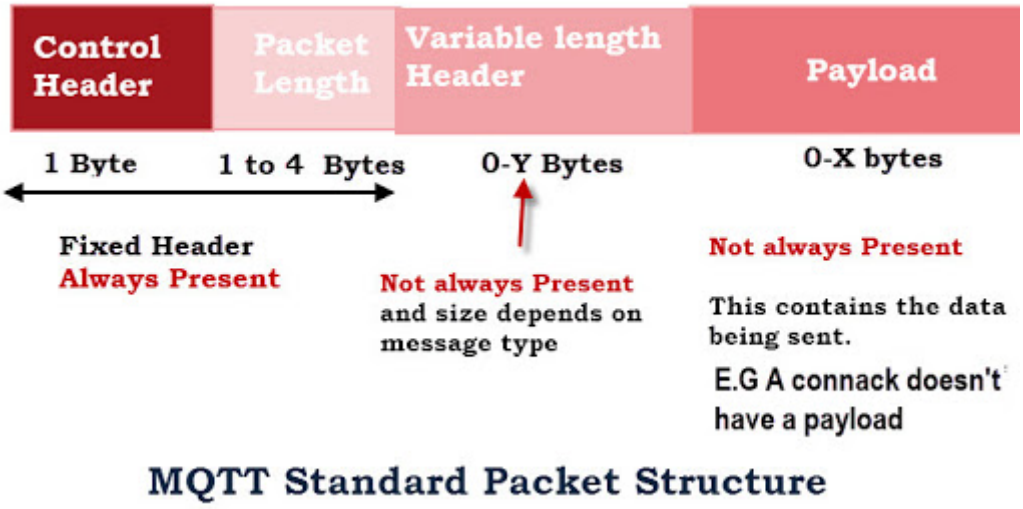


> La plupart des courtiers MQTT commerciaux ne prennent pas en charge le niveau de QoS 2 car il est lent et consomme plus de ressources.

1.3 Structure d'un paquet MQTT

- **Source** : Comprendre la structure des paquets du protocole MQTT

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).



MQTT Standard Packet Structure

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 Fondation Eclipse et est un projet de iot.eclipse.org.

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

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

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

*.bash

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

- **Affichage de la version installée**

*.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.

> 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**.

- **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

> Le broker est installé en tant que **service**. Pour l'arrêter ou le redémarrer, utiliser les commandes suivantes :

*.bash

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

3.1.3 Tests

> Le paquet **mosquitto-clients** fournit deux commandes, mosquitto_sub pour une souscription et mosquitto_pub pour une publication.

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
```

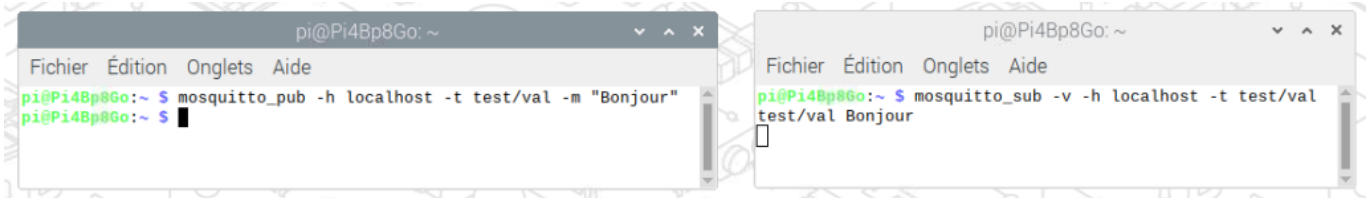
- **Publication**

*.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** : Page de manuel de moustique.conf