







E3020 Systèmes embarqués



CM13-PROTOCOLE MQTT

MQTT



Un protocole fait pour l'Internet des Objets




My  tells my  to open the garage and start my 

My  tells a  to dispatch a  to my location

My  tells my  that an intruder has entered

A  tells my  to tell my  that a package has arrived

My  tells my  that I am following my treatment plan

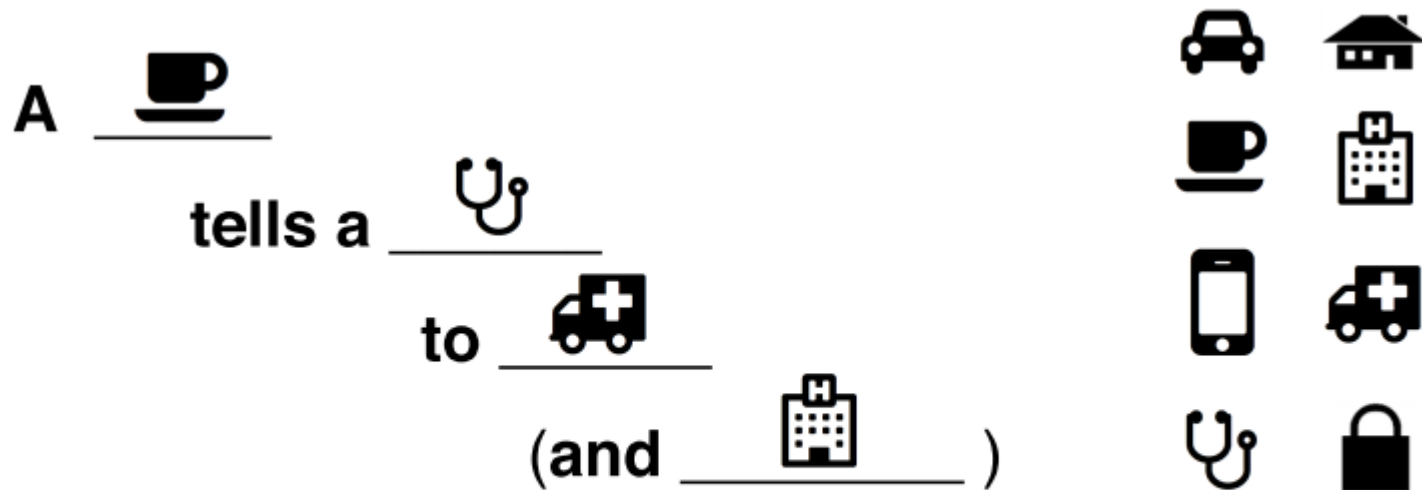
My  tells my  that they are too far from the 

Tout est possible

A _____
tells a _____
to _____
(and _____)



Un choix possible



My [connected coffee cup](#) tells my [doctor](#) to [send an ambulance](#) and [take me to the hospital](#) because I've had dangerous amounts of caffeine...

Les scénarios IoT

Tous les scénarios possibles font qu'il s'agit de nouveaux challenges :

1. Nécessite un modèle temps-réel
2. Publier des informations « one-to-many »
3. Écouter à des évènements s'ils ont lieux
4. Envoi de petit paquet de données issu de petit matériel électronique
5. Envoi des données « fiable » sur des réseaux « non-fiable »

MQTT

Est un protocole léger pour l'échanges de données

1. **Open** : il est standardisé et ouvert, avec plus de 40 implémentations clients
2. **Lightweight** : Léger, surcharge minimale, les clients font qqes (kb)
3. **Fiable** : il possède un outil (QoS) qui permet de fiabiliser la transmission
4. **Simple** : 3 fonctions principales : connecter, publier, s'abonner



MQTT - brokers

Les solutions possibles

Appliance

IBM MessageSight



1m connections
15m QoS 0 / sec
policies for security,
messaging, connection

[developer VM](#)

Commercial

Cloud

HiveMQ

IBM IoT Foundation

Eurotech EDC

Litmus Loop

Others

“Freemium”

Open Source

Mosquitto (C)

Mosca (Node.js)

Moquette (Java)

RSMB (C) [tiny]

Others

Eclipse Sandbox

iot.eclipse.org

Free

MQTT - caractéristiques

Désigné pour un trafic minimal et des « devices » conscits

[small header size](#)

PUBLISH	2-4 bytes
CONNECT	14 bytes

HTTP	0.1-1 KB
------	----------

[binary payload \(not text\)](#)

[small clients:](#) 30 KB (C), 100 KB (Java)

[minimal protocol exchanges](#)

MQTT has configurable keep alive
(2 byte PINGREQ / PINGRES)

Les concepts

- ❑ Le broker MQTT (parfois appelé serveur)
- ❑ le client :
 - ❑ Qui publie des topics
 - ❑ Qui s'abonne à des topics

Le broker MQTT

Il a pour fonction de contrôler les communications :

1. Réceptionne les infos des nœuds qui publient
2. Envoi les données vers les clients abonnés

Il s'occupe de transmettre les infos mais ne les stocke pas (sauf exception)

Si par exemple, il n'y a pas d'abonné, alors la donnée publiée est « perdue », sauf exception.

Le topic

Le topic est associé à la donnée et permet de la classer. Les rubriques sont traitées de manière hiérarchique en utilisant **une barre oblique (/)** comme séparateur, comme un système de fichiers. L'organisation est **libre** et est régie par la cohérence entre les différents clients.

Par exemple, plusieurs ordinateurs peuvent publier leurs informations sur la température du disque dur sous la rubrique suivante :

1. COMPUTER_NAME / Capteurs / température / HARDDRIVE_NAME

Les clients reçoivent les messages en créant des abonnements (souscriptions)

Les clients

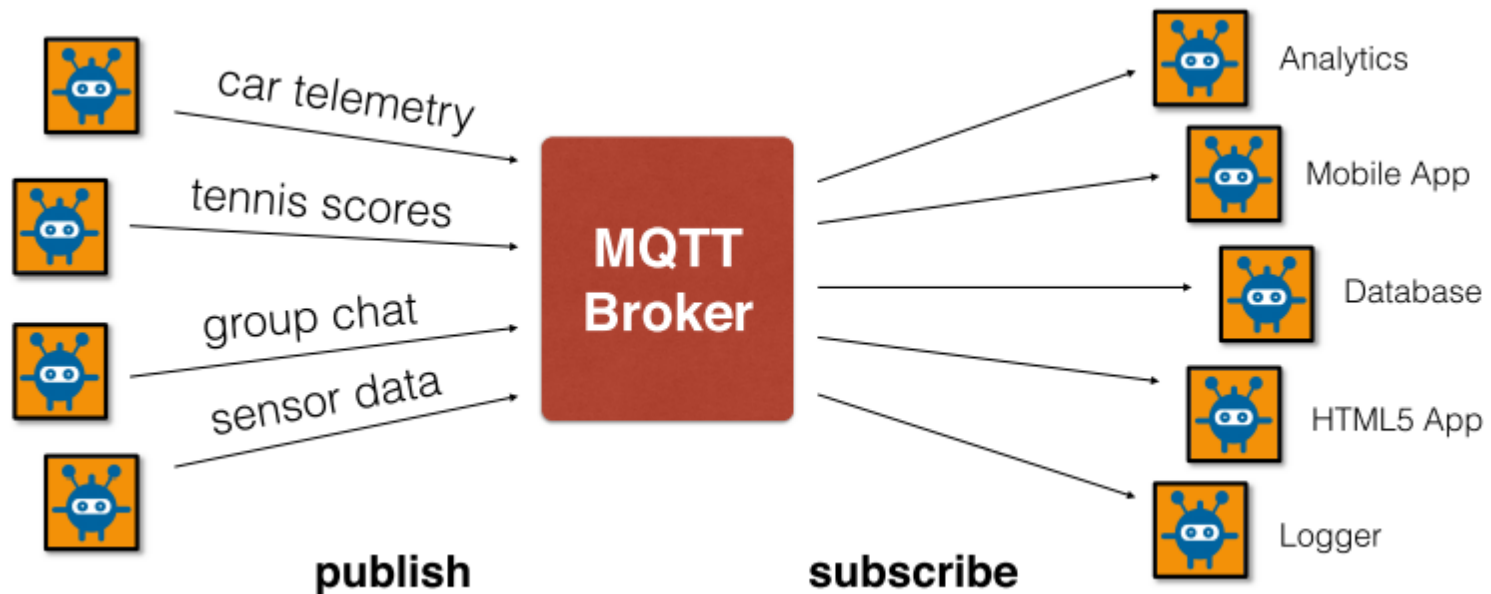
Ils doivent se faire connaître au moment de la connexion avec le broker.

Ils sont de deux types :

1. Soit « publisher », ils vont transmettre des données liées à un certain topic au broker
2. Soit « subscriber », ils sont abonnés à un(des) topics particuliers et ils reçoivent du broker toutes les nouvelles données publiées par les « publisher » correspondant au(x) topic(s) ciblés.

Publisher et subscriber

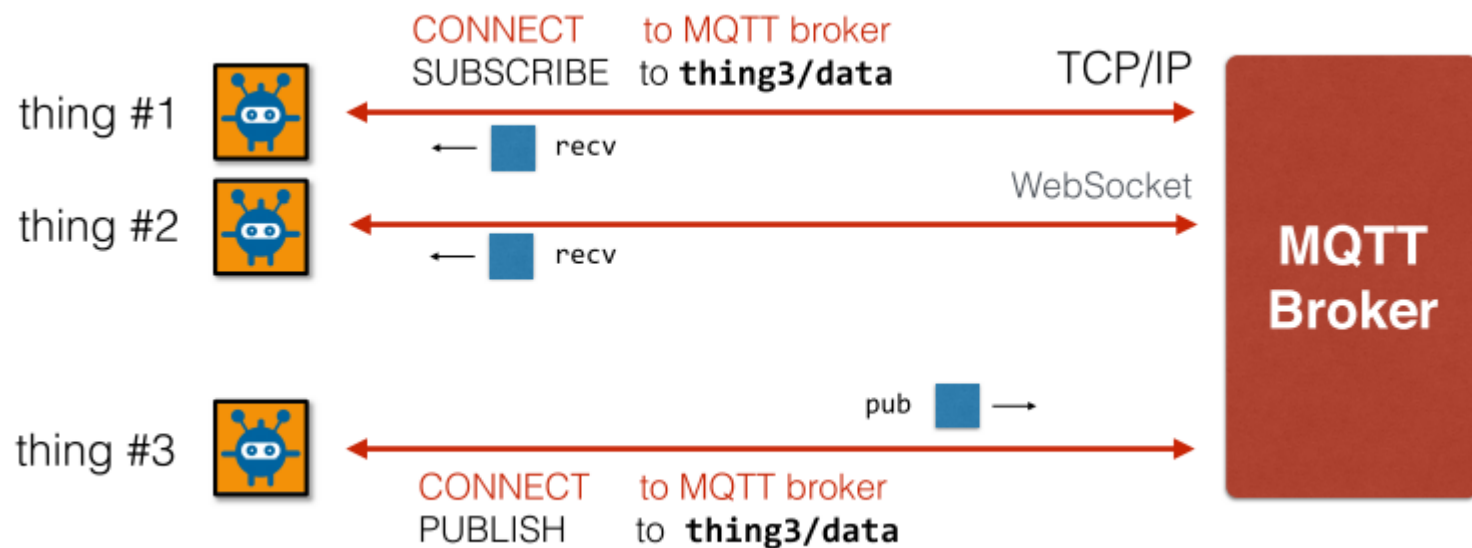
Les deux fonctionnement sont découplés



MQTT

La communication est Bi-directionnelle

Communication est asynchrone



MQTT

Protocole basé sur la communication TCP/IP

Les ports dédiés sont :

1. 1883 (MQTT)
2. 8830 (TSL) permet un encryptage de la communication mais

On privilégie l'encryptage des données à la source et ensuite le transfert des données.

MQTT - procédure

1. Le broker MQTT est lancé (écoute sur le port 1883)
2. Demande de connexion d'un client avec certaines options
 1. Nom/MdP
 2. Clean session
 3. Keep alive
3. Publier une donnée sur un topic avec des options :
 1. QoS (Quality of Service)
 2. Retained
4. S'abonner à un topic avec certaines options :
 1. QoS
 2. Wildcard

Demo (mqtt)

Broker Mosquitto sur PC

Plugin MQTTLens sur Chrome (client 1)

App MyMQTT sur Andoïd (client 2)

Réseau H2G2 => TheAnswerIs42

MQTT – contenu du topic

Il est « data agnostic »

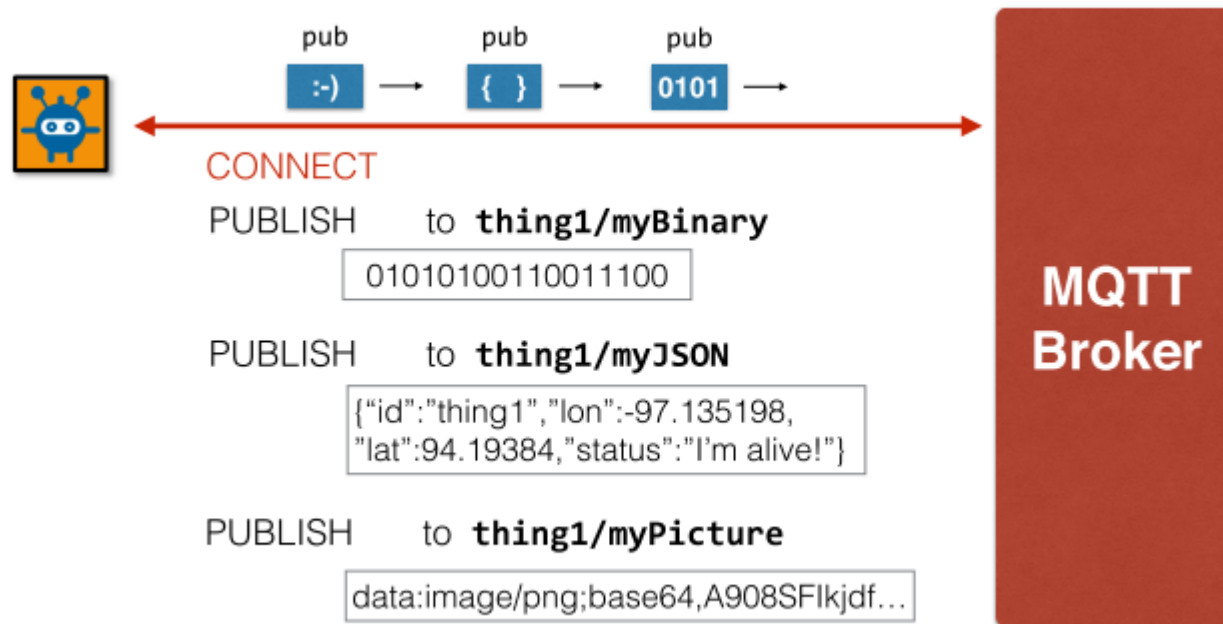
Après connexion, un client peut publier des messages

1. Chaque message MQTT doit contenir un topic, qui sera utilisé par le courtier pour transmettre le message aux clients intéressés
2. Le Payload dépend totalement du cas d'utilisation, est sans formatage et entièrement à charge de l'expéditeur.

Les différents cas typiques sont les messages : binaire, texte, XML, JSON voire image en base64.

MQTT – contenu du topic

N'a pas besoin de connaître le contenu pour une flexibilité du transfert



MQTT : Topic et abonnement

Un abonnement peut concerné un sujet unique, ou plusieurs, via des « Wildcards ».

Deux Wildcards (caractères génériques) sont disponibles :

Le « + » est utilisé comme joker pour un seul niveau de hiérarchie

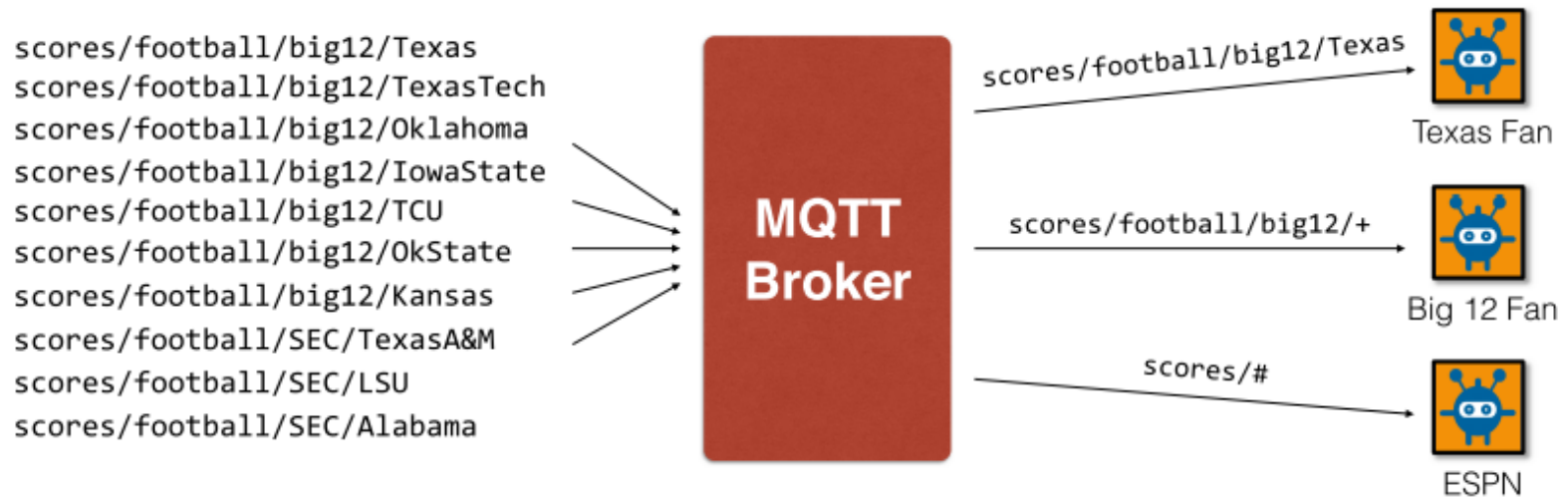
❑ Capteurs / + / température / +

Le « # » est utilisé comme générique pour tous les niveaux de hiérarchie inférieurs

❑ Capteurs / mypc / température / #

MQTT : Topic et abonnement

MQTT permet l'inscription à plusieurs topics via des « wildcard »



single level wildcard: +

multi-level wildcard: #

MQTT : Topic et abonnement

Je ne veux les sports que des équipes masculines :

Score/tennis/homme/roland

Score/tennis/femme/roland

Score/football/homme/D1

Score/footrball/femme/D1

=> Score/+/homme/#

Topics commençant par \$

In general you are totally free in naming your topics, but there is one exception. Each topic, which starts with a \$-symbol will be treated specially and is for example not part of the subscription when subscribing to `#`. These topics are reserved for internal statistics of the MQTT broker. Therefore it is not possible for clients to publish messages to these topics. At the moment there is no clear official standardization of topics that must be published by the broker. It is common practice to use `$SYS/` for all these information and a lot of brokers implement these, but in different formats. One suggestion on \$SYS-topics is in the [MQTT GitHub wiki](#) and here are some examples:


- `$SYS/broker/clients/connected`
- `$SYS/broker/clients/disconnected`
- `$SYS/broker/clients/total`
- `$SYS/broker/messages/sent`
- `$SYS/broker/uptime`

Topics – best practices

1. Don't use a leading forward slash : */myhome/groundfloor/livingroom*
2. Don't use spaces in a topic
3. Keep the topic short and concise
4. Use only ASCII characters, avoid non printable characters
5. Embed a unique identifier or the ClientId into the topic (*client1/status et client2/status.*)
6. Don't subscribe to #
7. Don't forget extensibility
8. Don't forget extensibility
9. Use specific topics, instead of general ones
: *myhome/livingroom/temperature, myhome/livingroom/brightness* and *myhome/livingroom/humidity*, instead of sending all values over *myhome/livingroom*.

MQTT – connect packet

MQTT CONNECT PACKET FORMAT

MQTT-Packet: CONNECT		
contains:		Example
clientId		"client-1"
cleanSession		true
username (optional)		"hans"
password (optional)		"letmein"
lastWillTopic (optional)		"/hans/will"
lastWillQos (optional)		2
lastWillMessage (optional)		"unexpected exit"
lastWillRetain (optional)		false
keepAlive		60

MQTT - sécurité

MQTT autorise une authentification client-broker via :

- Login et Password
- MAC ou token sont souvent utilisés

- ❑ Le port TCP/IP 1883 est utilisé
- ❑ Les communications passent en clair

MQTT CONNECT PACKET FORMAT

MQTT-Packet: CONNECT	
contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

MQTT - sécurité

Pour crypter la communication MQTT, la plupart des brokers utilisent TLS

1. Le port 8883 est réservé pour « MQTT over TLS »
2. Le cryptage induit une surcharge et une latence réseau

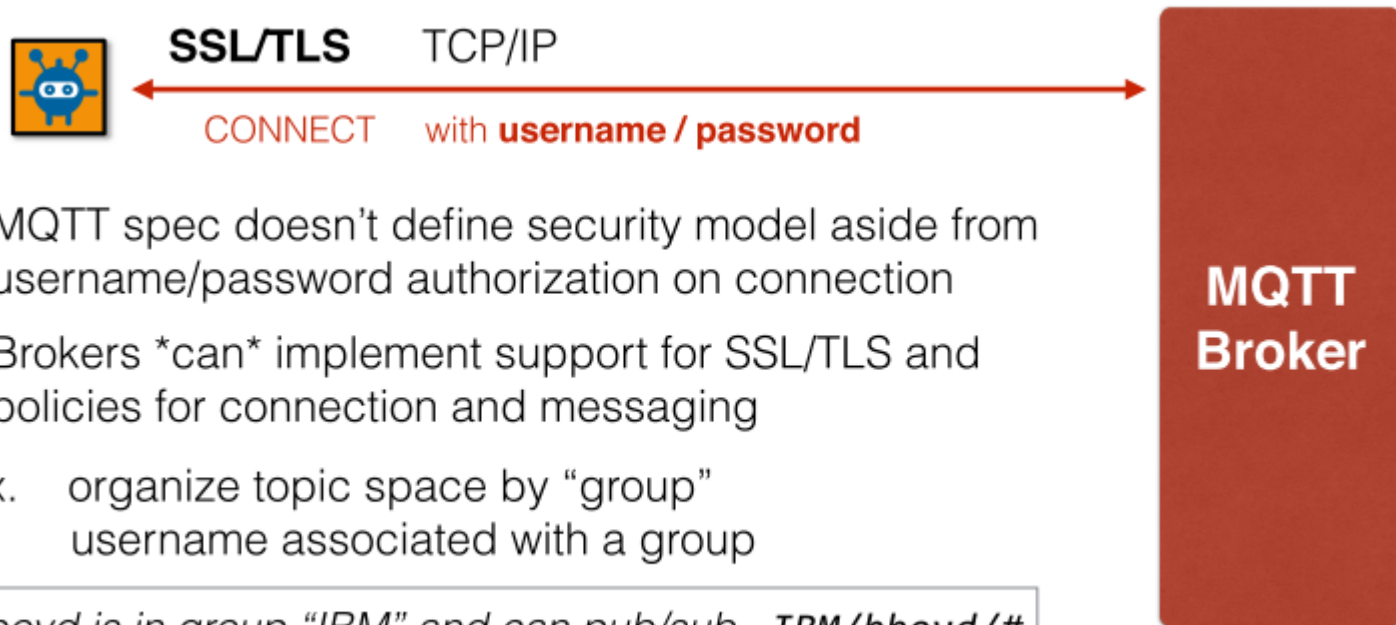
A l'heure actuelle, les objets utilisent des microcontrôleurs qui ne peuvent souvent fournir assez de puissance de calcul pour utiliser du TLS. On privilégie dans certains cas un encryptage de la donnée avant envoi.

MQTT CONNECT PACKET FORMAT

MQTT-Packet: CONNECT	
	
contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

MQTT - sécurité

sécurité



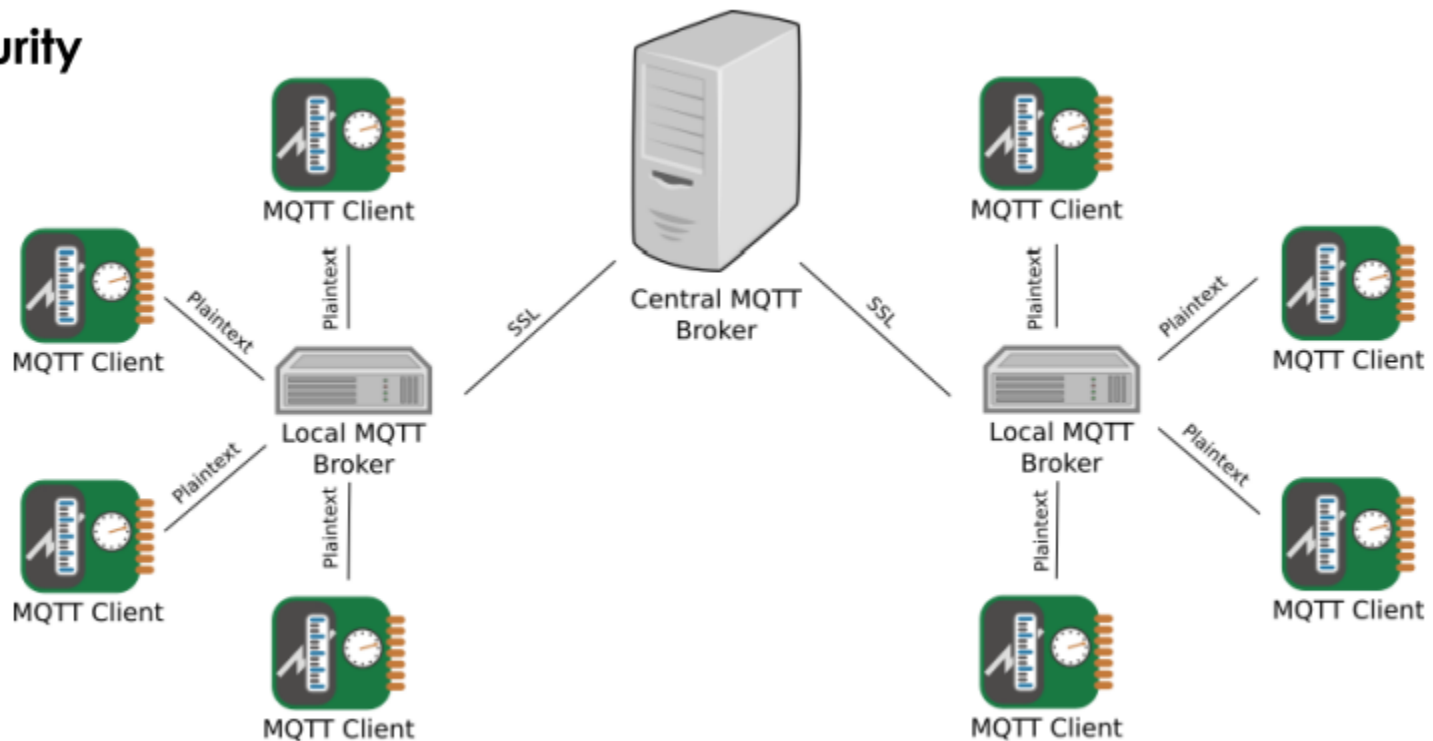
- MQTT spec doesn't define security model aside from username/password authorization on connection
- Brokers **can** implement support for SSL/TLS and policies for connection and messaging

ex. organize topic space by "group"
username associated with a group

bboyd is in group "IBM" and can pub/sub IBM/bboyd/#

MQTT - topologie

Security



MQTT – publish packet

MQTT PUBLISH PACKET FORMAT

MQTT-Packet:

PUBLISH



contains:	Example
packetId (always 0 for qos 0)	4314
topicName	"topic/1"
qos	1
retainFlag	false
payload	"temperature:32.5"
dupFlag	false



MQTT - QoS

QoS définit la pertinence de la livraison d'un message

1. Chaque objet peut souscrire à un sujet avec un QoS de 0 à 2
2. Chaque objet peut publier sur un sujet avec un QoS de 0 à 2

Mais l'envoi d'un message avec un certain niveau QoS ne nécessite pas le même niveau pour la réception, le souscripteur utilise lui aussi son QoS

MQTT - QoS

L'envoi d'un message avec un niveau QoS ne nécessite pas le même niveau pour la réception, le souscripteur utilise lui aussi son QoS, cependant la qualité de la transmission sera celle du niveau le plus bas :

1. Si un message est publié en QoS 2 et qu'un client est abonné en QoS 0, le message sera fourni à ce client en QoS 0.
2. Pour un deuxième exemple, si un client est abonné en QoS 2 et qu'un message est publié en QoS 0, le client le recevra en QoS 0

MQTT - QoS

MQTT définit trois niveaux de qualité de service (QoS)

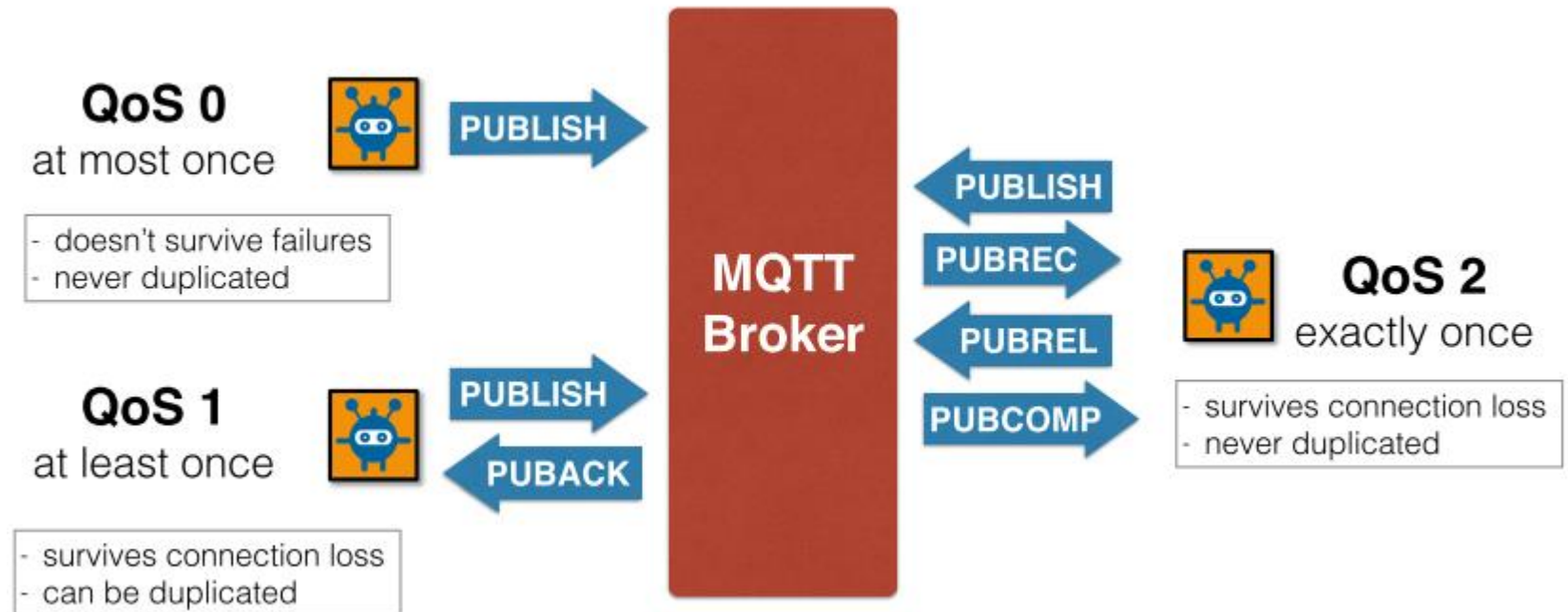
Les niveaux plus fiables impliquent une latence et une bande passante plus élevées

- 0.** Livre une fois (ou plus), sans confirmation
- 1.** Livre au moins une fois, avec confirmation requise
- 2.** Livre une fois en utilisant une « handshake » de confirmation

Il faut choisir le bon niveau en fonction de l'application!

MQTT - QoS

Différent niveaux de validation de transmission



MQTT – retained message

Tous les messages peuvent être configurés pour être conservés. Le courtier conservera le dernier message du topic. Et ce même après l'avoir envoyé aux abonnés déjà connectés.

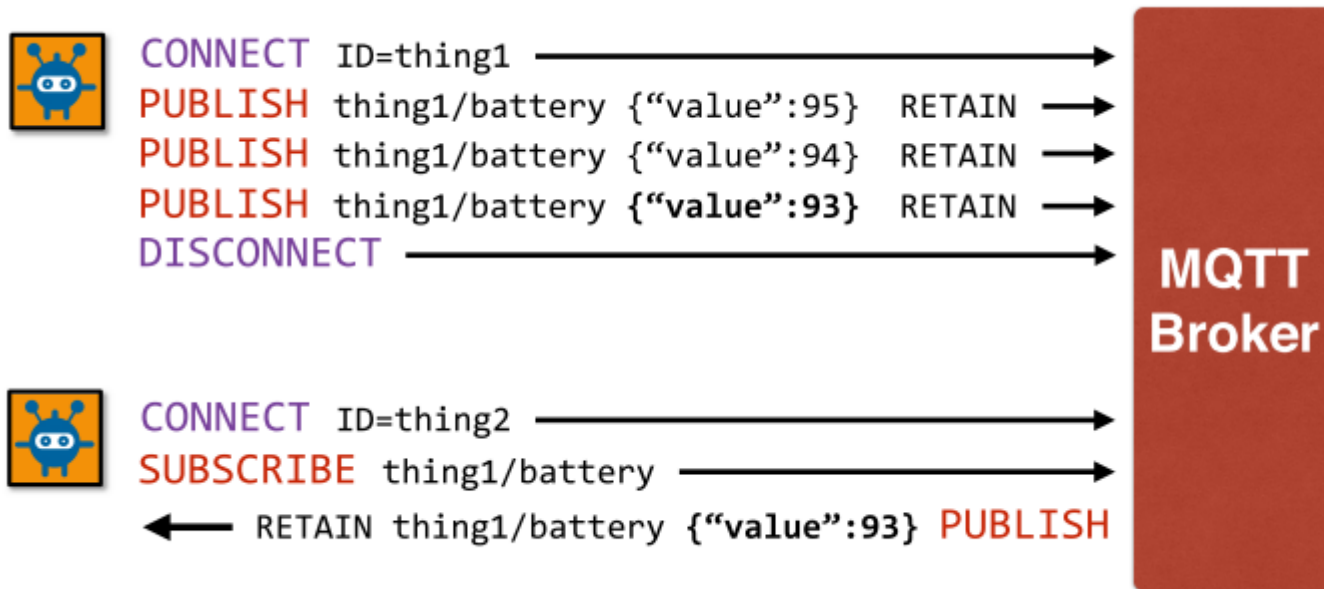
Si un nouvel abonnement est effectué, le message sera envoyé au client

Ceci est utile en tant que mécanisme de « dernière valeur connue »

- ❑ Si un topic (sans retain) est rarement mis à jour, un nouvel abonné devra attendre pour recevoir un message
- ❑ Avec retain, le client recevra une mise à jour instantanée

MQTT – retained message

Retained messages for last value caching



MQTT – publish packet

1. Packet Identifier : Unique id in a flow (QoS 1,2)
2. Topic Name : Structured string
3. QoS : Level definition 0,1,2
4. Retain-Flag : Saved by the broker
5. Payload : data-agnostic
6. DUP flag : Copie renvoyée d'un message à cause de non réception d'un ACK (QoS 1,2)

MQTT PUBLISH PACKET FORMAT

MQTT-Packet:	
PUBLISH	
	
contains:	Example
packetId (always 0 for qos 0)	4314
topicName	"topic/1"
qos	1
retainFlag	false
payload	"temperature:32.5"
dupFlag	false

MQTT – subscribe packet

1. Packet Identifier : Unique id in a flow (QoS 1,2)
2. QoS : Level definition 0,1,2
3. Topic Name : Structured string
4. ...

MQTT-Packet:	
SUBSCRIBE	
contains:	Example
packetId	4312
qos1	1
topic1	"topic/1"
qos2	0
topic2	"topic/2"
...	...

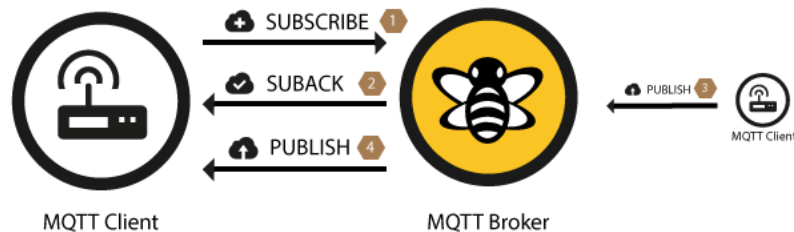
Packet Identifier


The packet identifier is a unique identifier between client and broker to identify a message in a message flow. Setting this MQTT internal identifier is the responsibility of the client library and/or the broker. The Packet Identifier becomes available for reuse after the Client has processed the corresponding acknowledgement packet.

Source : <https://www.hivemq.com/blog/mqtt-essentials-part-4-mqtt-publish-subscribe-unsubscribe>

MQTT – suback packet

Retourne un code par abonnement à un topic. Si pas permis, code 128



MQTT-Packet: **SUBACK** 

contains:

packetId

returnCode 1 (one returnCode for each

returnCode 2 topic from SUBSCRIBE,

... in the same order)

...

Example
4313
2
0
...

Return Code	Return Code Response
0	Success – Maximum QoS 0
1	Success – Maximum QoS 1
2	Success – Maximum QoS 2
128	Failure

MQTT Packet format

Ressource : <https://docs.solace.com/MQTT-311-Prtl-Conformance-Spec/MQTT%20Control%20Packet%20format.htm>

MQTT – Clean session

Un client doit (re)souscrire aux sujets à chaque connexion, ce qui est fastidieux pour les clients légers/réguliers et/ou dans un environnement instable

Lors de la connexion un client peut définir l'option "clean session"

1. Si l'option est False : la connexion est traitée comme étant durable, les souscriptions (et les messages QoS 1 & 2) resteront stockées jusqu'à la prochaine connexion
2. Si l'option est True: les abonnements seront supprimés à la déconnexion

Ressource : <https://www.hivemq.com/blog/mqtt-essentials-part-7-persistent-session-queuing-messages>

MQTT – Clean session

So what will be stored in the session?

- ❑ Existence of a session, even if there are no subscriptions
- ❑ All subscriptions
- ❑ All messages in a [Quality of Service \(QoS\) 1 or 2](#) flow, which are not confirmed by the client
- ❑ All new QoS 1 or 2 messages, which the client missed while it was offline
- ❑ All received QoS 2 messages, which are not yet confirmed to the client
- ❑ That means even if the client is offline all the above will be stored by the broker and are available right after the client reconnects.

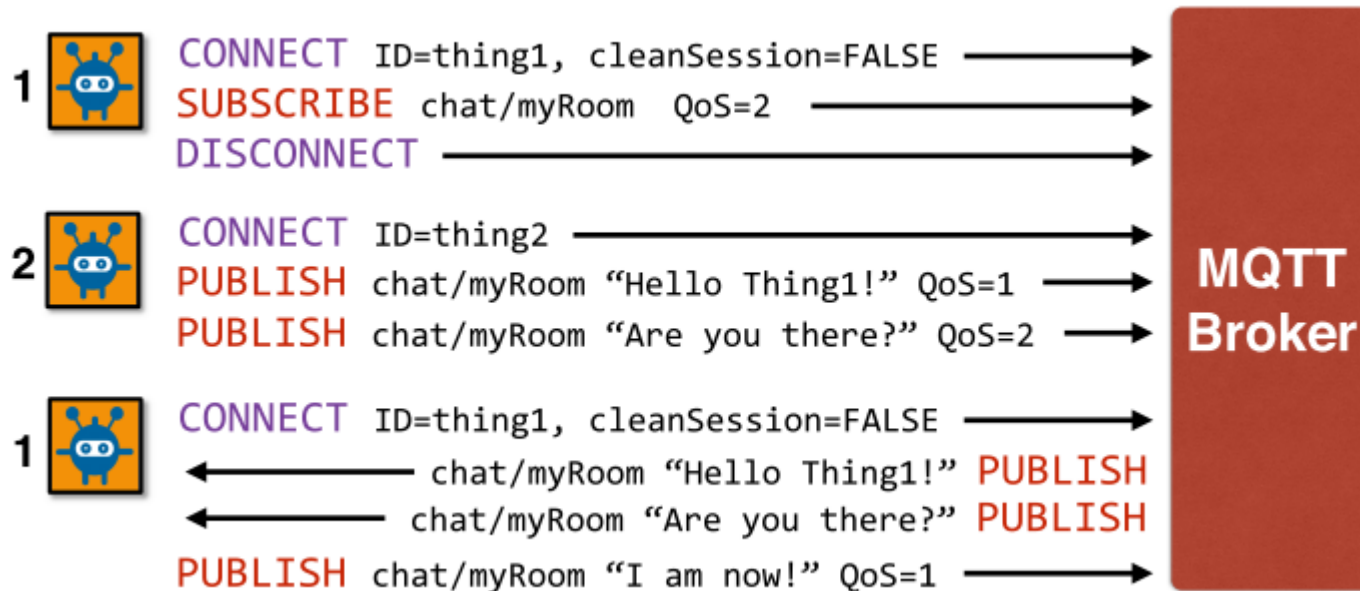
MQTT – Clean session

How long are messages stored on the broker ?

*A often asked question is how long is a session stored on the broker. The easy answer is until the clients comes back online and receives the message. **But what happens if a client does not come online for a long time?** The constraint for storing messages is often the memory limit of the operating system. There is no standard way on what to do in this scenario*

MQTT – Clean session

Client id and clean session for session state



MQTT – keep alive

Les connexions semi-ouvertes ne sont pas utilisées dans le modèle M2M car consommatrices de ressources.

MQTT fournit la fonctionnalité de maintien en vie

1. Elle s'assure que la connexion est toujours viable entre Broker et client
2. Le client spécifie un intervalle de temps maximale prouvant sa présence lors de la connexion au Broker
3. L'intervalle définit la période de temps la plus longue possible que le courtier et le client utilisent sans envoyer de message

MQTT – keep alive

Si le client n'émet aucun message utile durant la période définie, il doit envoyer un paquet PINGREQ au courtier pour confirmer sa disponibilité et s'assurer que le courtier est toujours disponible

1. Le courtier déconnecte un client qui n'émet pas de PINGREQ ou tout autre message
2. De même, le client ferme la connexion si la réponse (PINGRESP) du courtier n'est pas reçue dans un délai raisonnable

MQTT – Last will and testament

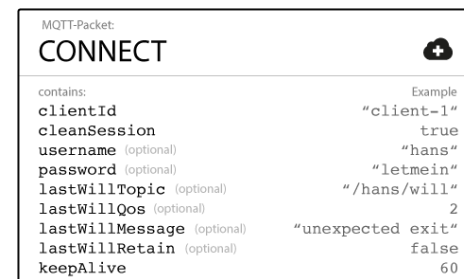
Surtout utilisé dans les scénarios où la connexion réseau est très peu fiable (typ. Perte de connexion dû au lieu, la batterie, perturbation..).

2 possibilités :

1. soit déconnexion désirée :



2. Soit la déconnexion est non-désirée



MQTT – Last will and testament

Stratégie :

1. Lorsqu'un client se connecte il publie sur son statut : ***client1/status*** with the payload "***online***".
2. Au moment de la connexion, le client à envoyer un LWT sur le même topic "offline" en mode retained

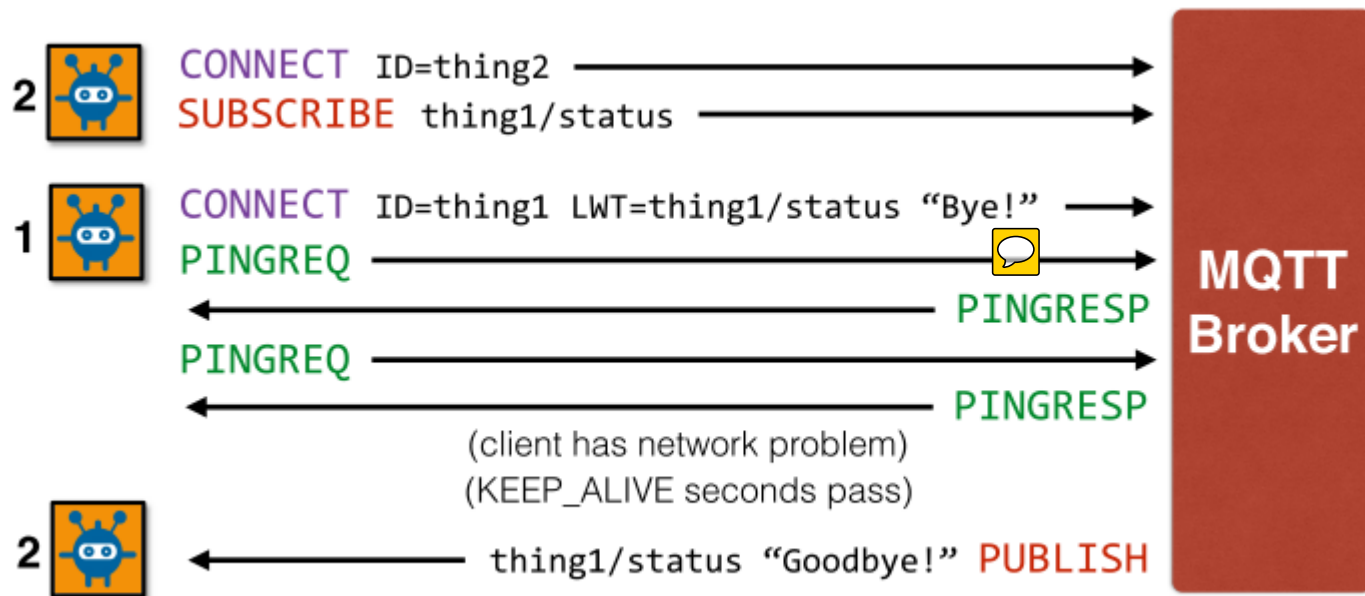
MQTT - sécurité

MQTT CONNECT PACKET FORMAT

MQTT-Packet:	
CONNECT	
	
contains:	Example
clientId	"client-1"
cleanSession	true
username (optional)	"hans"
password (optional)	"letmein"
lastWillTopic (optional)	"/hans/will"
lastWillQos (optional)	2
lastWillMessage (optional)	"unexpected exit"
lastWillRetain (optional)	false
keepAlive	60

MQTT – keep alive

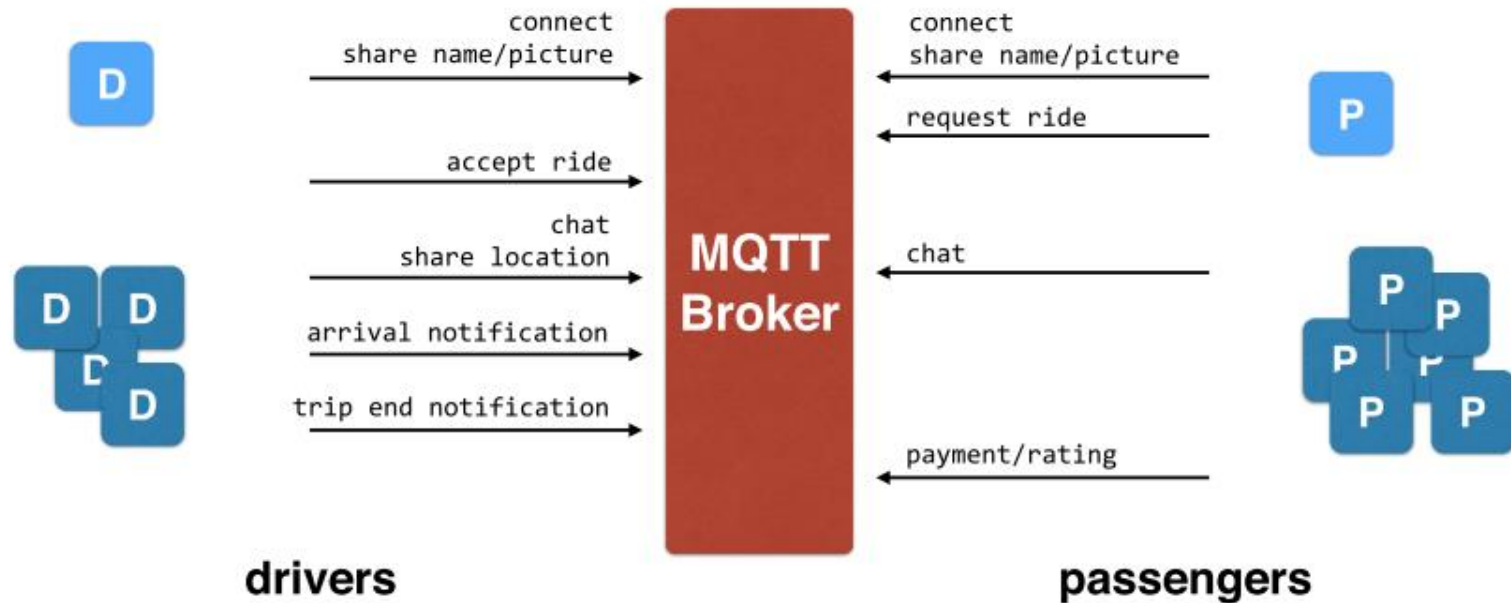
Last will and testament for presence



MQTT - exemple

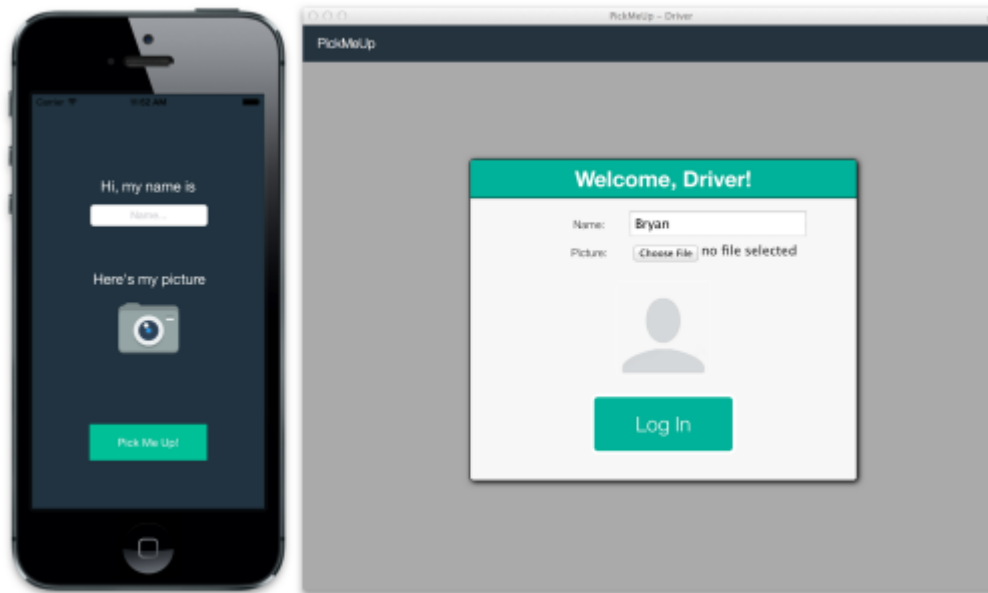
PickMeUp

Flow



MQTT – exemple

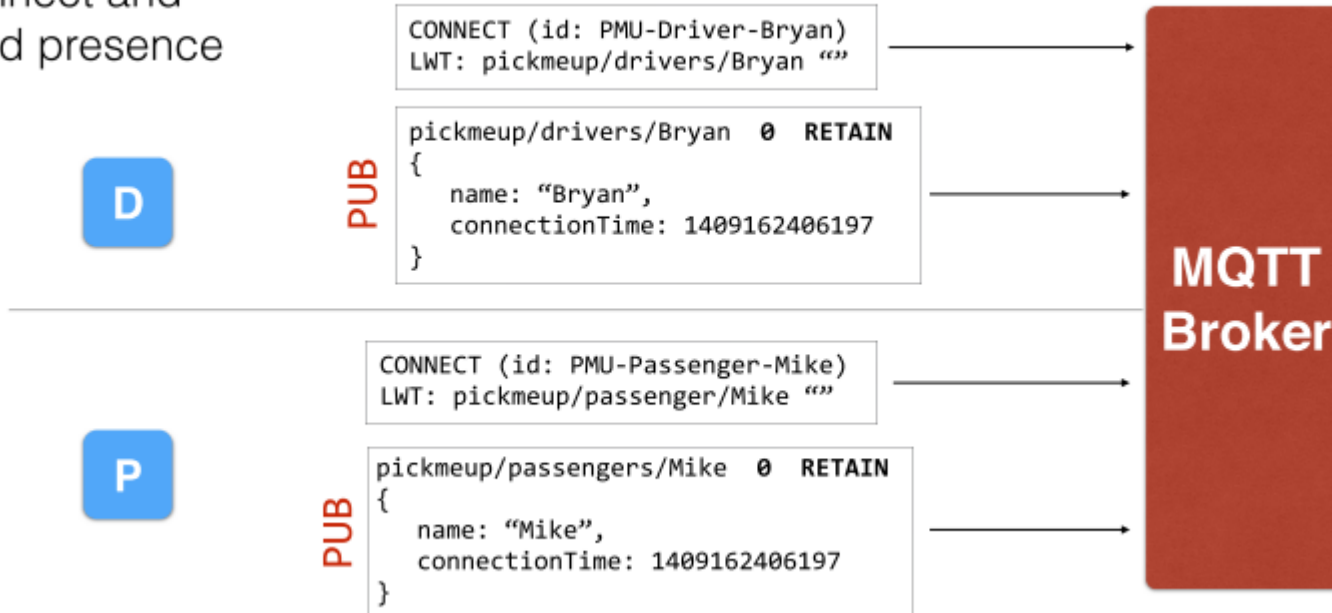
phase 1 : connexion



MQTT – exemple

phase 1 : connexion

Connect and
send presence



MQTT – exemple

phase 1 : connexion

Send picture,
subscribe to
inbox



PUB

```
pickmeup/drivers/Bryan/picture 0 RETAIN
{
  url: "data:image/png;base64,A198cf9013..."
}
```

SUB

```
pickmeup/drivers/Bryan/inbox 2
```

Send picture,
subscribe to
inbox



PUB

```
pickmeup/passengers/Mike/picture 0 RETAIN
{
  url: "data:image/png;base64,F87r19ZKa90..."
}
```

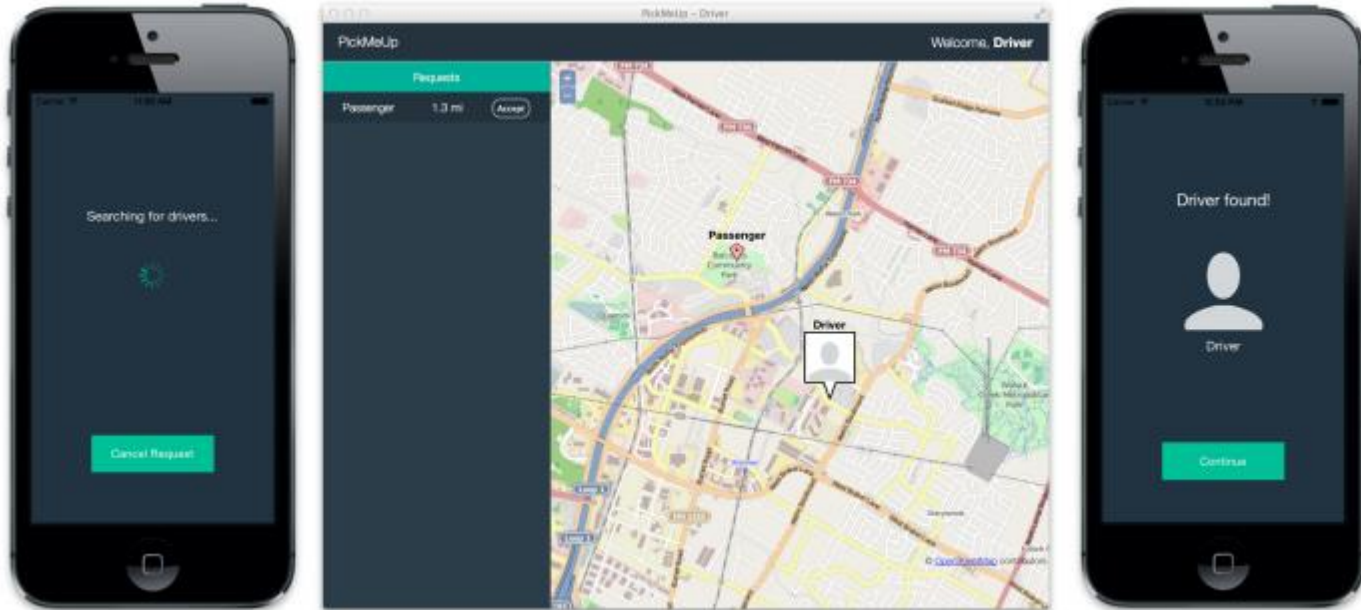
SUB

```
pickmeup/passengers/Mike/inbox 2
```

MQTT
Broker

MQTT – example

phase 2 : matching



MQTT – example

phase 2 : matching

Subscribe to
requests, accept
request

D

SUB

```
pickmeup/requests/+ 0
```

```
pickmeup/passengers/Mike/inbox 1
{
  type: "accept",
  driverId: "Bryan",
  lon: <lon>, lat: <lat>
}
```

PUB

```
pickmeup/requests/Mike 0 RETAIN ""
```

Send request,
subscribe to
driver

P

PUB

```
pickmeup/requests/Mike 1 RETAIN
{
  name: "Mike", lon: <lon>, lat: <lat>
}
```

SUB

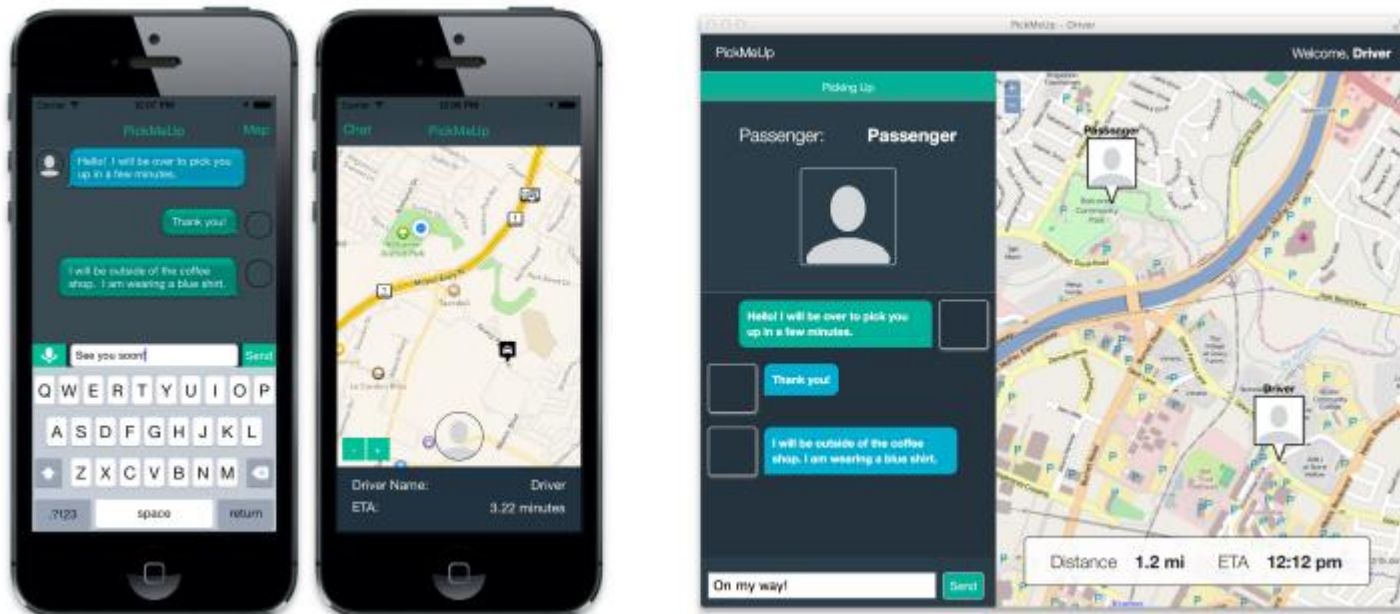
```
pickmeup/drivers/Bryan 0
```

```
pickmeup/drivers/Bryan/picture 0
```

MQTT
Broker

MQTT – exemple

phase 3 : approche du véhicule



MQTT – exemple

phase 3 : approche du véhicule

Subscribe to
passenger data
chat to driver

Publish
driver location
chat to passenger



Driver

SUB

```
pickmeup/passengers/Mike 0
```

```
pickmeup/passengers/Mike/picture 0
```

```
pickmeup/passengers/Mike/location 0
```

```
pickmeup/drivers/Bryan/chat 0
```

PUB

```
pickmeup/passengers/Mike/chat 0
{
  format: "text", data: "On my way!"
  or
  format: "data:audio/wav;base64",
  data: "18bwagh0AH30913n..."
}
```

```
pickmeup/drivers/Bryan/location 0 RETAIN
{
  lon: <lon>, lat: <lat>
}
```

**MQTT
Broker**

MQTT – exemple

phase 3 : approche du véhicule

Subscribe to
driver location
chat to passenger

SUB

```
pickmeup/drivers/Bryan/location 0
```

```
pickmeup/drivers/Bryan/chat 0
```

Publish
chat to driver



Passenger

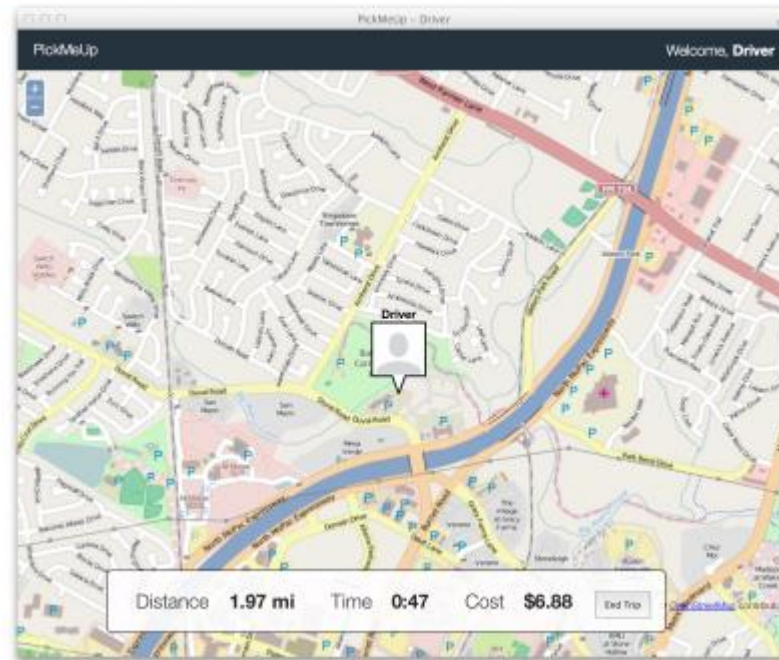
PUB

```
pickmeup/drivers/Bryan/chat 0
{
  format: "text", data: "On my way!"
  or
  format: "data:audio/wav;base64",
  data: "18bwagh0AH30913n..."
}
```

**MQTT
Broker**

MQTT – exemple

phase 4 : conduite



MQTT – exemple

phase 4 : conduite

Publish
trip start notification
trip end notification



Driver

PUB

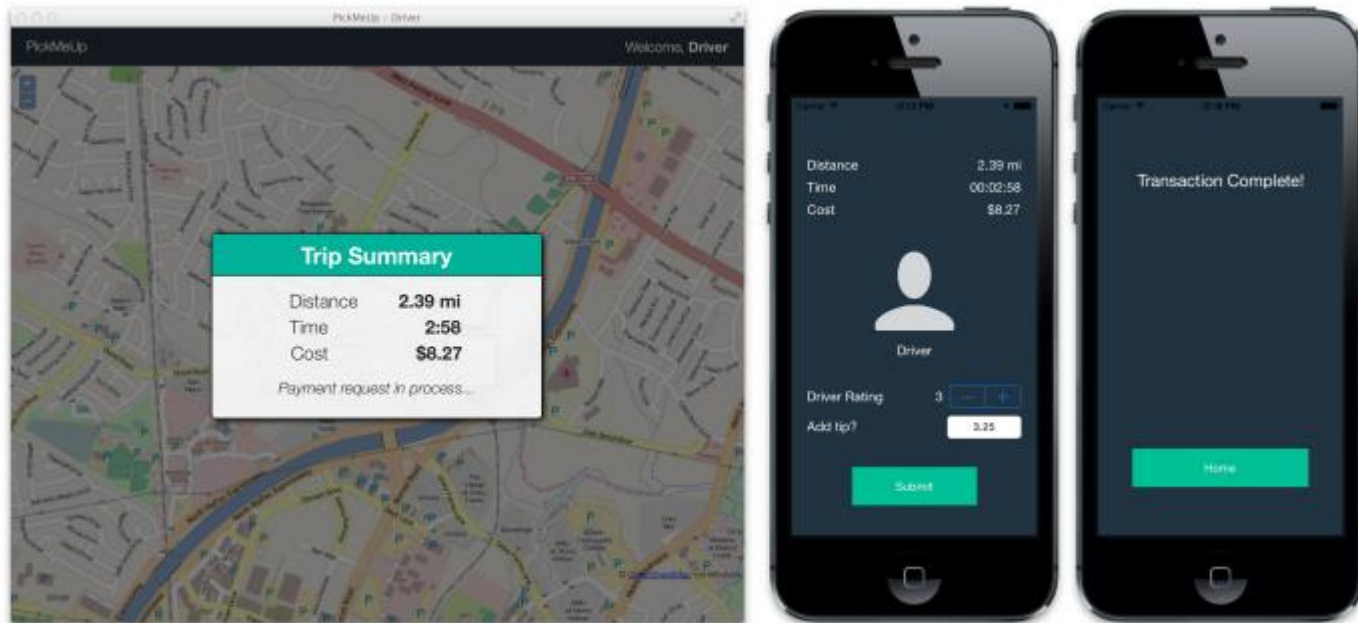
```
pickmeup/passengers/Mike/inbox 2
{
  type: "tripStart"
}
```

```
pickmeup/passengers/Mike/inbox 2
{
  type: "tripEnd",
  distance: 2.39,    // miles
  time: 178,         // minutes
  cost: 8.27         // dollars
}
```

**MQTT
Broker**

MQTT – example

phase 5 : paiement



MQTT – exemple

phase 5 : paiement

Publish rating and
payment



PUB

```
pickmeup/payments 2
{
  driverId: "Bryan",
  passengerId: "Mike",
  cost: 8.27,
  rating: 3,
  tip: 3.25
}
```

Subscribe to
payments, publish
when processed



Backend

SUB

```
pickmeup/payments 2
```

```
pickmeup/passengers/Mike/inbox 2
{
  type: "tripProcessed",
  tip: 3.25, rating: 3
}
```

PUB

```
pickmeup/drivers/Bryan/inbox 2
{
  type: "tripProcessed",
  tip: 3.25, rating: 3
}
```

**MQTT
Broker**

MQTT – exemple

Idées :

- Publish a retained “presence message” on connect, use last will and testament (LWT) to clear
- Use retained messages if you want late-joining subscribers to get data instantly (ex. driver position, requests)
- Set up a topic space friendly to wildcards (ex. <app>/<type>/<id>/<field>)
- QoS 0 = information updates, chat (things we can lose)
- QoS 1 = requests, request accepts (important, but client can handle dups)
- QoS 2 = inbox messages, payment (important, duplicates problematic)

Ressources :

- MQTT home MQTT.org
- Eclipse Paho MQTT clients eclipse.org/paho
- Mosquitto broker mosquitto.org
- IBM MessageSight ibmdw.net/messaging/messagesight
- IBM IoT Foundation internetofthings.ibmcloud.com
- MQTT demos m2m.demos.ibm.com
- IBM Messaging Github github.com/ibm-messaging (coming soon)
- IBM Redbook + PickMeUp github.com/ibm-messaging/mqtt-PickMeUp

Ressources

<https://learn.adafruit.com/diy-esp8266-home-security-with-lua-and-mqtt/configuring-mqtt-on-the-raspberry-pi>

<http://test.mosquitto.org/>

<https://mosquitto.org/>

MQTT client :

<https://www.hivemq.com/blog/seven-best-mqtt-client-tools>

App MyMQTT

Forum

<https://stackoverflow.com/questions/39678982/difference-between-port-and-listener-in-mqtt>

Getting started

<https://www.hivemq.com/blog/how-to-get-started-with-mqtt>

<http://www.steves-internet-guide.com/mqtt/>

https://www.ibm.com/support/knowledgecenter/en/SS9D84_1.0.0/com.ibm.mm.tc.doc/tc10140_.htm

<http://www.steves-internet-guide.com/into-mqtt-python-client/>

Ressources

Video pour utilisation de Paho python

<https://youtu.be/QAaXNt0oqSI>

https://www.youtube.com/watch?time_continue=61&v=2aHV2Fn0I60

Installation pour le prochain cours

1. Installer un broker MQTT, typiquement mosquitto
 1. Pour windows :
 1. <https://sivatechworld.wordpress.com/2015/06/11/step-by-step-installing-and-configuring-mosquitto-with-windows-7/>
 2. <http://www.steves-internet-guide.com/install-mosquitto-broker/>
 2. Faire bien attention à ce que le dll et openssl
2. Installer MQTTLens sur chrome
3. Télécharger un client MQTT sur votre smartphone

Faites un test en lançant Mosquitto, vérifier qu'il est lancé (netstat -an)

Publiez et écoutez avec un client

MQTT Clients

Python client pour BBB?