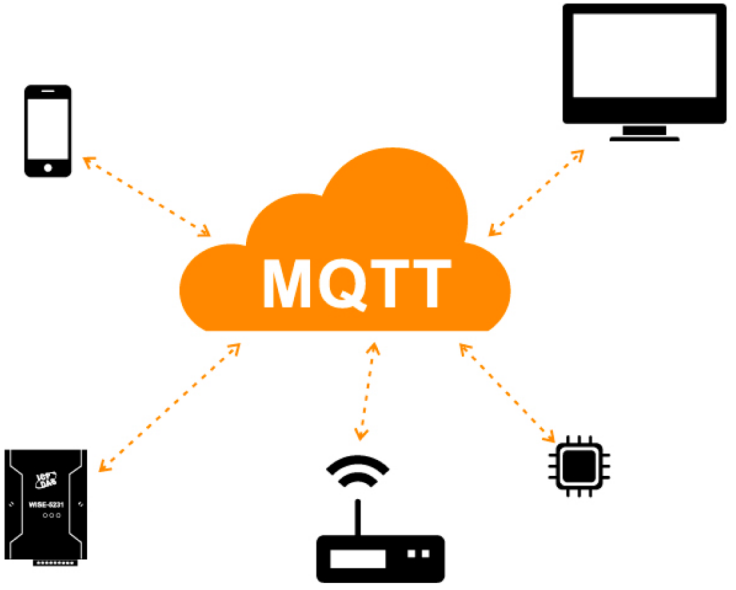
Intro to MQTT

Pietro Manzoni

Universitat Politecnica de Valencia (UPV)

Valencia - SPAIN

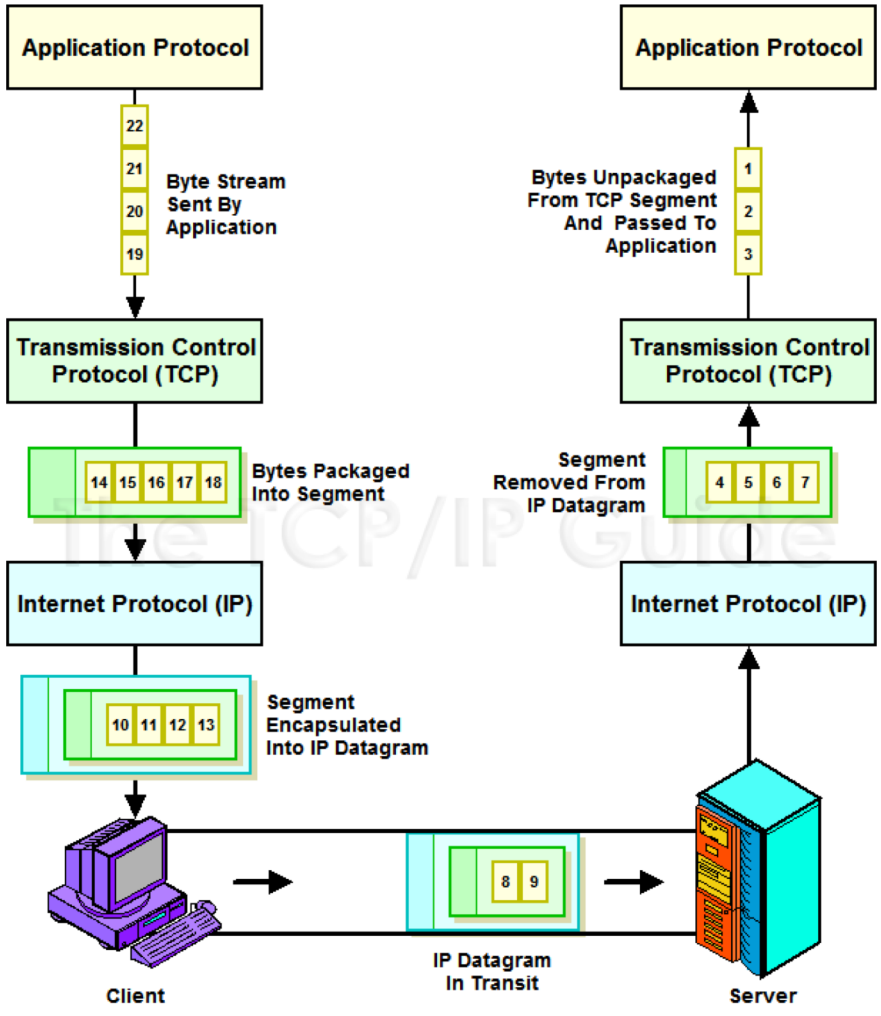
pmanzoni@disca.upv.es



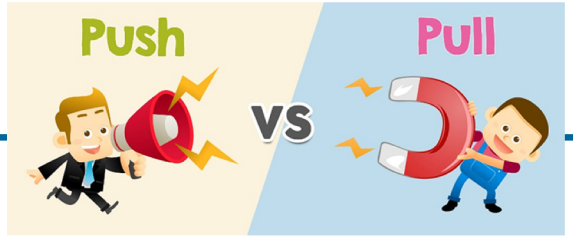
https://github.com/pmanzoni/KIC2019

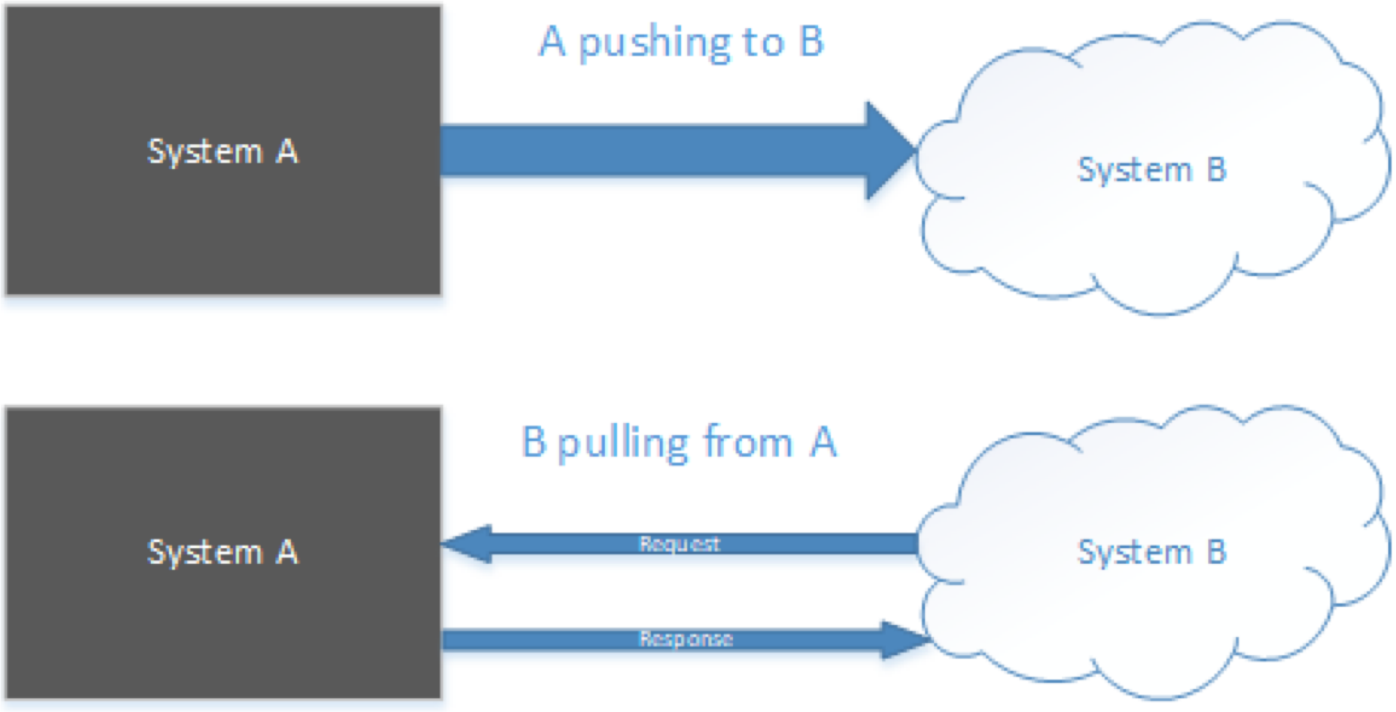
From "byte streams" to "messages"

- The "old" vision of data communication was based on **reliable byte streams**, i.e., TCP
- Nowadays **messages interchange** is becoming more common
  - E.g., Twitter, Whatsapp, Instagram, Snapchat, Facebook,…
- Actually is not that new…
  - emails: SMTP+MIME,
  - FTP,



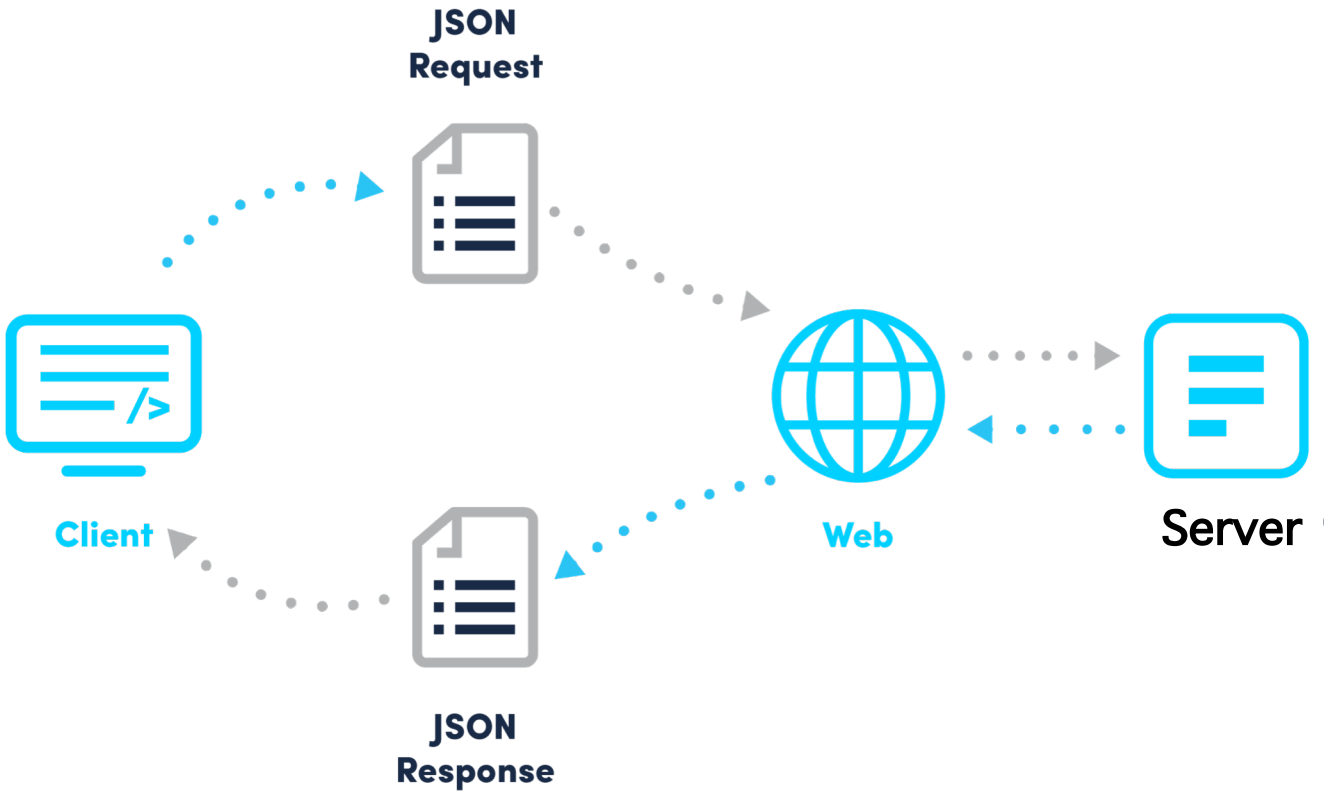
Ways to interchange “messages”





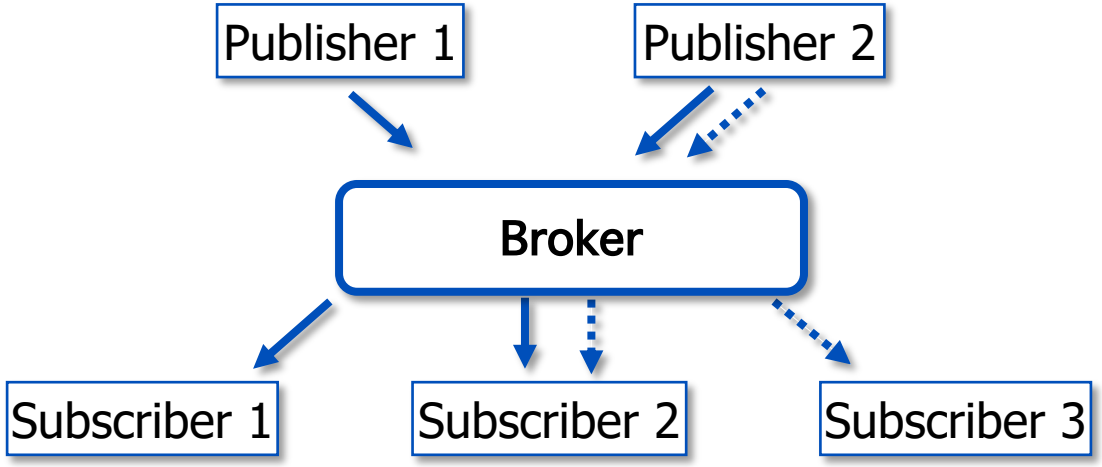
Request/response approach

- **REST**: Representational State Transfer
- Widely used; based on HTTP
- *Lighter version: **CoAP** (Constrained Application Protocol)*



Pub/sub approach

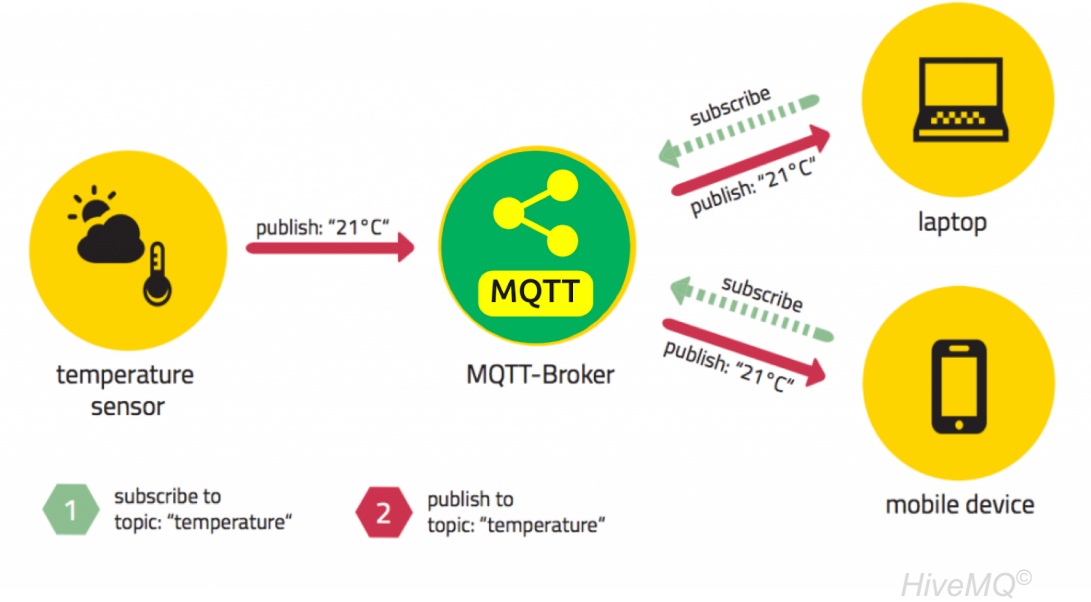
- Publish/Subscriber
  - aka: producer/consumer



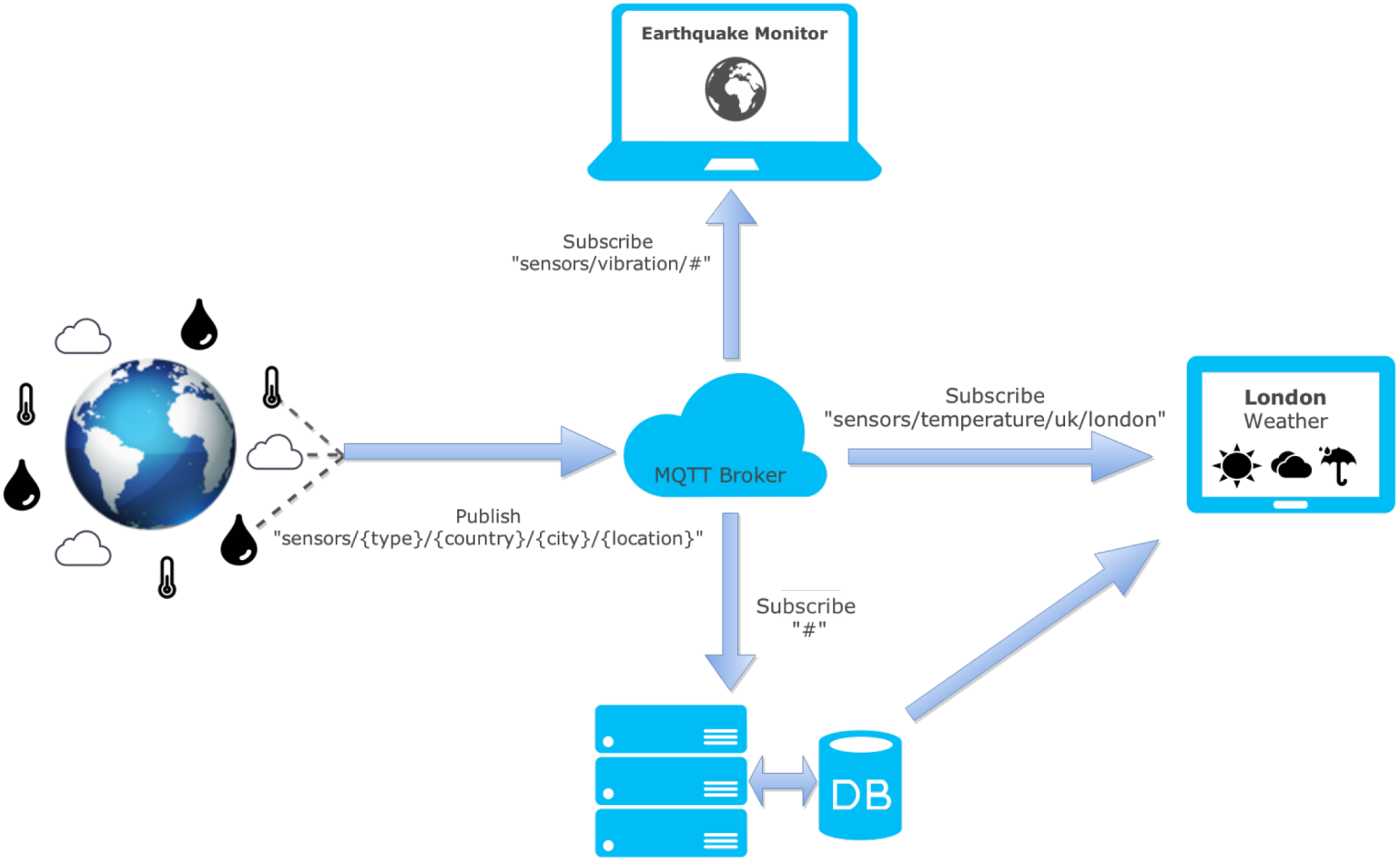
- Various protocols:
  - **MQTT**, AMQP, XMPP (was Jabber)
- Growing technique
  - E.g., https://cloud.google.com/iot/docs/how-tos/mqtt-bridge

Pub/sub approach

- Pub/Sub separate a client, who is sending a message about a specific **topic**, called publisher, from another client (or more clients), who is receiving the message, called subscriber.
- There is a third component, called broker, which is known by both the publisher and subscriber, which filters all incoming messages and distributes them accordingly.



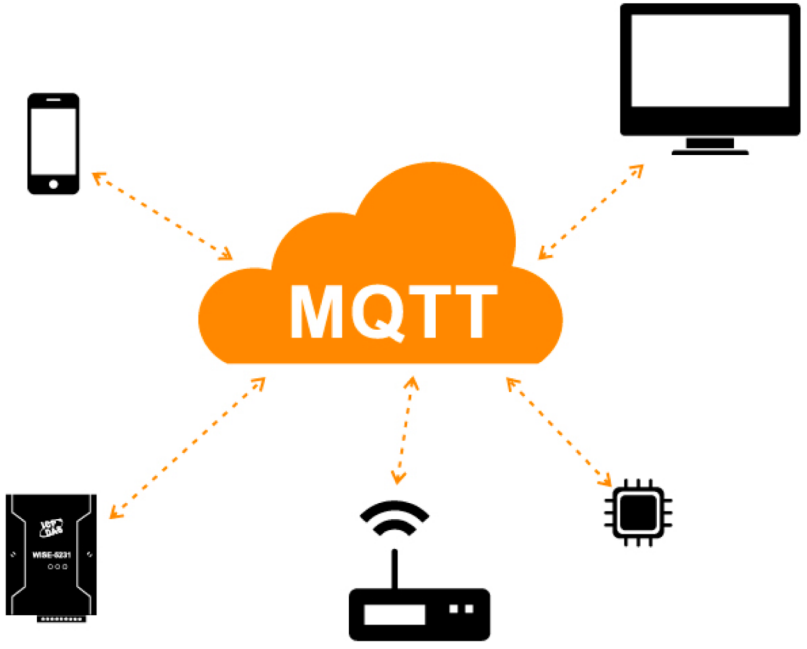
An example



Source: https://zoetrope.io/tech-blog/brief-practical-introduction-mqtt-protocol-and-its-application-iot

Intro to MQTT

- Fundamental concepts



Message Queuing Telemetry Transport

- A lightweight publish-subscribe protocol that can run on embedded devices and mobile platforms ➔ http://mqtt.org/
  - Low power usage.
  - Binary compressed headers
  - Maximum message size of 256MB
    - not really designed for sending large amounts of data
    - better at a high volume of low size messages.
- Documentation sources:
  - The MQTT community wiki:
    - `https://github.com/mqtt/mqtt.github.io/wiki`
  - A very good tutorial:
    - `http://www.hivemq.com/mqtt-essentials/`

Some details about versions

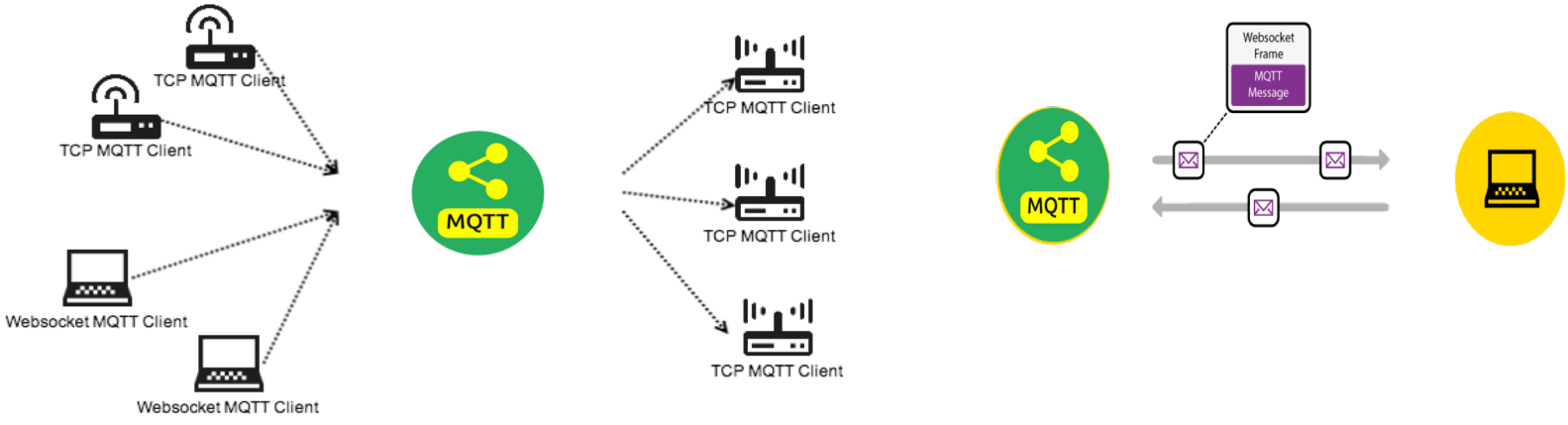
- **MQTT 3.1.1 is the current version of the protocol.**
  - Standard document here:
    - http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
  - October 29th 2014: MQTT was officially approved as OASIS Standard.
    - https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt

- MQTT v5.0 is the successor of MQTT 3.1.1
  - Current status: Committee Specification 02 (15 May 2018)
    - http://docs.oasis-open.org/mqtt/mqtt/v5.0/cs02/mqtt-v5.0-cs02.html
  - **Not backward compatible**; too many new things are introduced so existing implementations have to be revisited, for example:
    - **Enhancements for scalability** and large scale systems in respect to setups with 1000s and millions of devices.
    - **Improved error reporting** (Reason Code & Reason String)
    - Performance improvements and improved support for small clients
  - https://www.youtube.com/watch?time_continue=3&v=YIpesv_bJgU

MQTT works on top of…

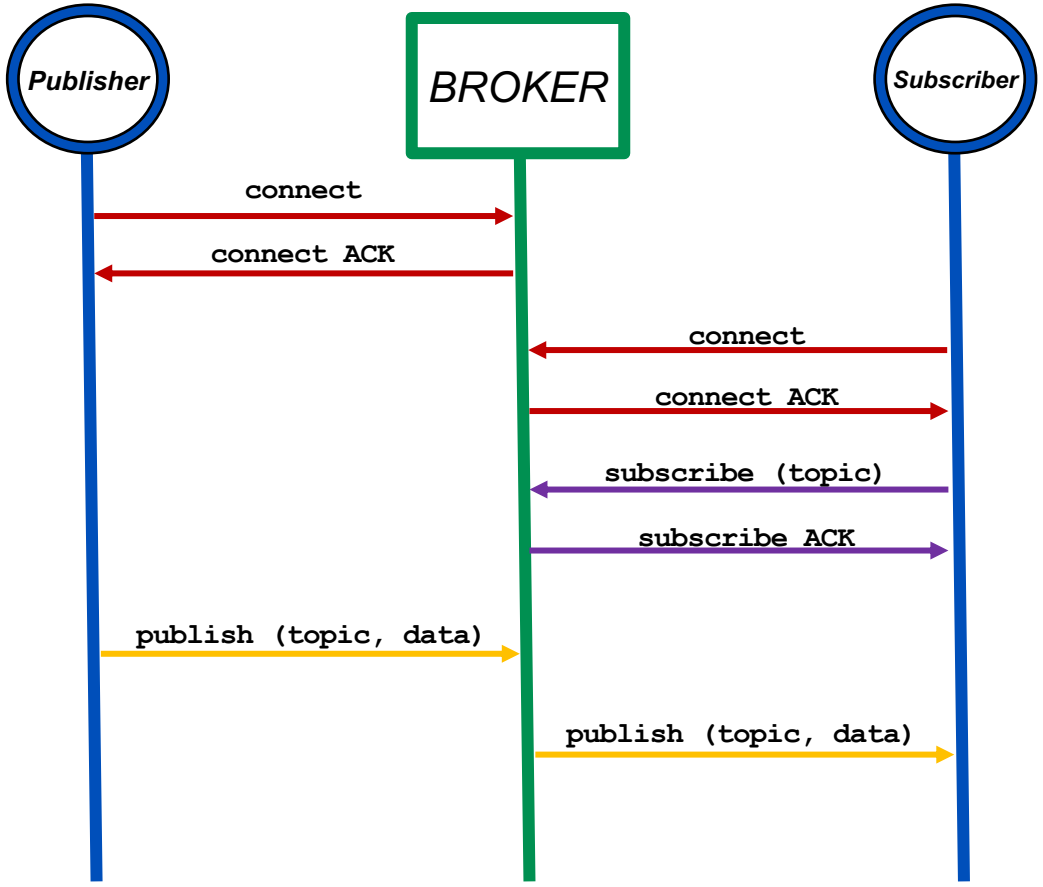
- mainly of TCP
  - There is also the closely related MQTT for Sensor Networks (MQTT-SN) where TCP is replaced by UDP → TCP stack is too complex for WSN

- websockets can be used, too!
  - Websockets allows you to receive MQTT data directly into a web browser.



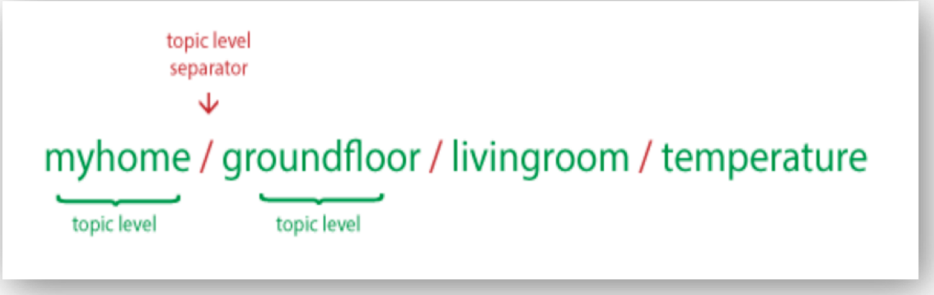
- Both, TCP & websockets can work on top of "Transport Layer Security (TLS)" (and its predecessor, Secure Sockets Layer (SSL))

Publish/subscribe interactions sequence



Topics

- MQTT Topics are structured in a hierarchy similar to folders and files in a file system using the forward slash (/) as a delimiter.
- Allow to create a user friendly and self descriptive **naming structures**



- Topic names are:
  - Case sensitive
  - use UTF-8 strings.
  - Must consist of at least one character to be valid.
- Except for the $SYS topic **there is no default or standard topic structure.**

Special $SYS/ topics

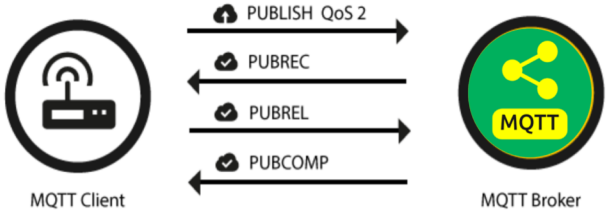
- $SYS/broker/clients/connected
- $SYS/broker/clients/disconnected
- $SYS/broker/clients/total
- $SYS/broker/messages/sent
- $SYS/broker/uptime

Topics wildcards

- Topic subscriptions can have wildcards. These enable nodes to subscribe to groups of topics that don't exist yet, allowing greater flexibility in the network's messaging structure.
  - '+' matches anything at a given tree level
  - '#' matches a whole sub-tree
- Examples:
  - Subscribing to topic `house/#` covers:
    - ✓ house/room1/main-light
    - ✓ house/room1/alarm
    - ✓ house/garage/main-light
    - ✓ house/main-door
  - Subscribing to topic `house/+/main-light` covers:
    - ✓ house/room1/main-light
    - ✓ house/room2/main-light
    - ✓ house/garage/main-light
  - but doesn't cover
    - ✓ house/room1/side-light
    - ✓ house/room2/side-light

Quality of Service (QoS)

- Messages are published with a Quality of Service (QoS) level, which specifies delivery requirements.
- A QoS 0 ("at most once") message is fire-and-forget.
  - For example, a notification from a doorbell may only matter when immediately delivered.
- With QoS 1 ("at least once"), the broker stores messages on disk and retries until clients have acknowledged their delivery.
  - (Possibly with duplicates.) It's usually worth ensuring error messages are delivered, even with a delay.
- QoS 2 ("exactly once") messages have a second acknowledgement round-trip, to ensure that non-idempotent messages can be delivered exactly once.



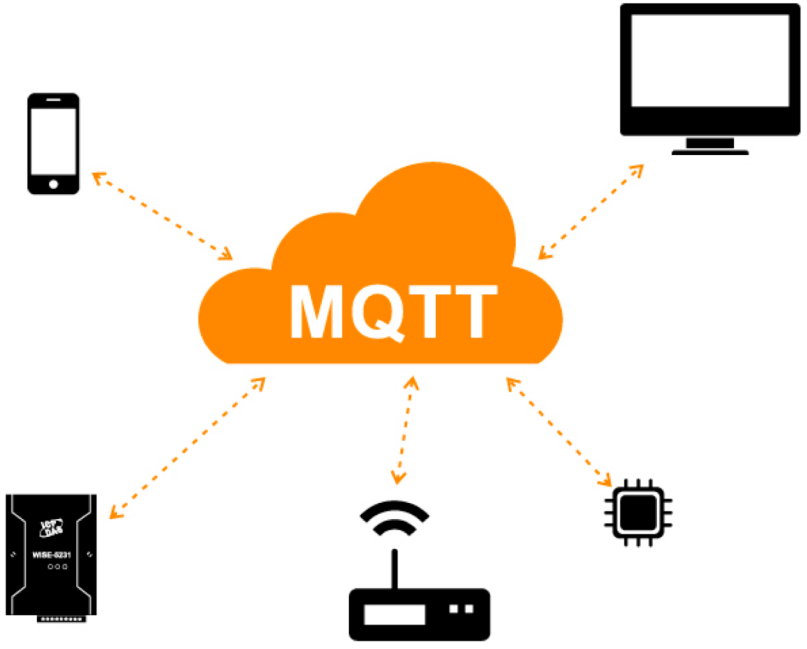
Retained Messages!!!

- A retained message is a normal MQTT message with the **retained flag set to true**. The broker will store the last retained message and the corresponding QoS for that topic
  - Each client that subscribes to a topic pattern, which matches the topic of the retained message, will receive the message immediately after subscribing.
  - For each topic **only one retained message** will be stored by the broker.

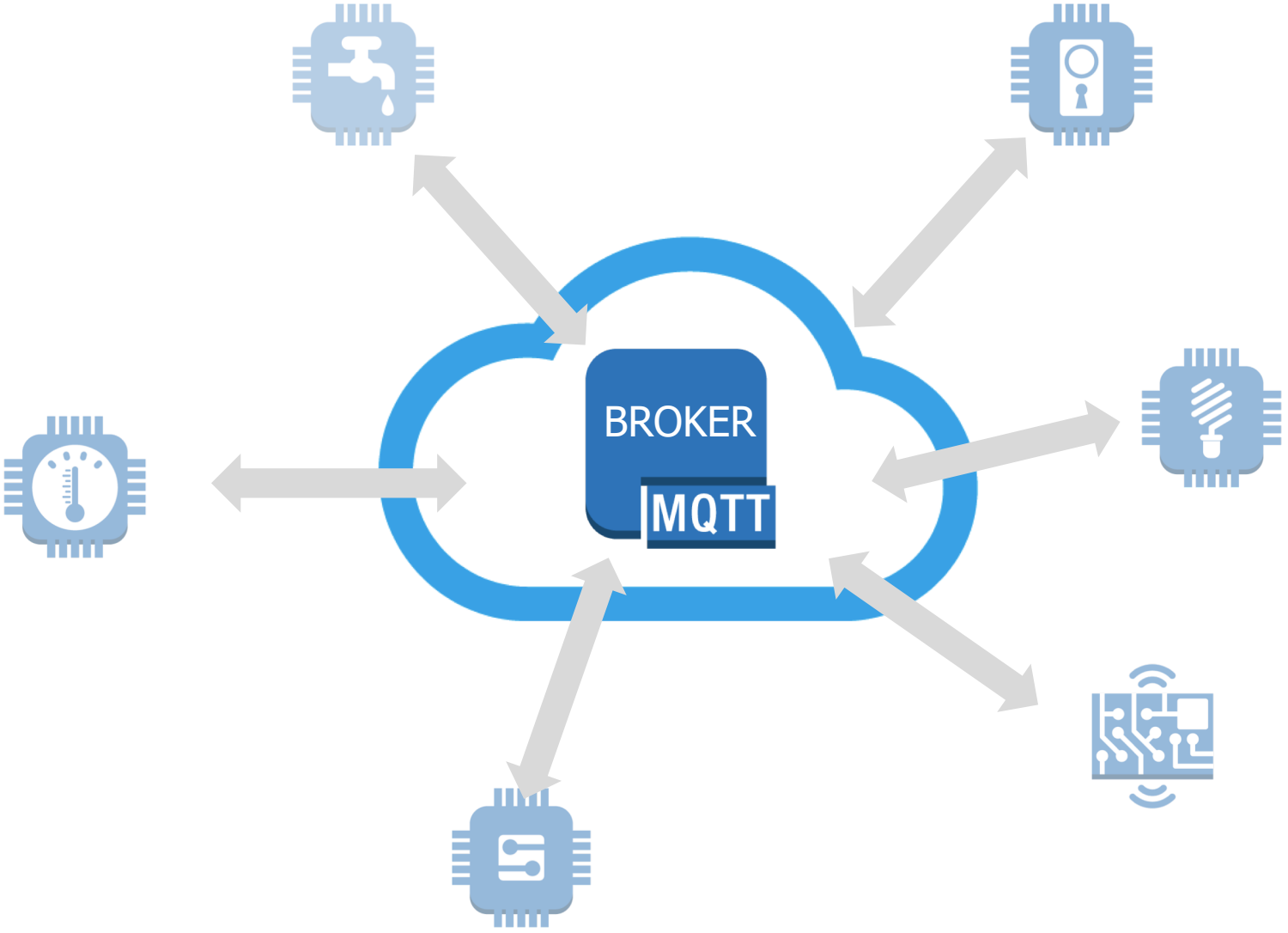
- Retained messages can help newly subscribed clients to get a status update immediately after subscribing to a topic and don't have to wait until a publishing clients send the next update.
  - In other words a retained message on a topic is the last known good value, because it doesn't have to be the last value, but it certainly is the last message with the retained flag set to true.

Intro to MQTT

- Brokers and clients



Creating a broker



Available MQTT brokers

- The most widely used are:
  - http://mosquitto.org/
    - man page: https://mosquitto.org/man/mosquitto-8.html
  - http://www.hivemq.com/
    - The standard trial version only supports 25 connections.

- And also:
  - https://www.rabbitmq.com/mqtt.html
  - http://activemq.apache.org/mqtt.html

- A quite complete list can be found here:
  - https://github.com/mqtt/mqtt.github.io/wiki/servers

Installing Mosquitto on a Raspberry Pi

- It takes only a few seconds to install a Mosquitto broker on a Raspberry. You need to execute the following steps:

```
sudo apt-get update
sudo apt-get install mosquitto mosquitto-clients
```

- Installation guidelines with websockets

```
https://gist.github.com/smoofit/dafa493aec8d41ea057370dbfde3f3fc
```

- Managing the broker:
  - To start and stop its execution use:

    ```
    sudo /etc/init.d/mosquitto start/stop
    ```

  - Verbose mode:

    ```
    sudo mosquitto –v
    ```

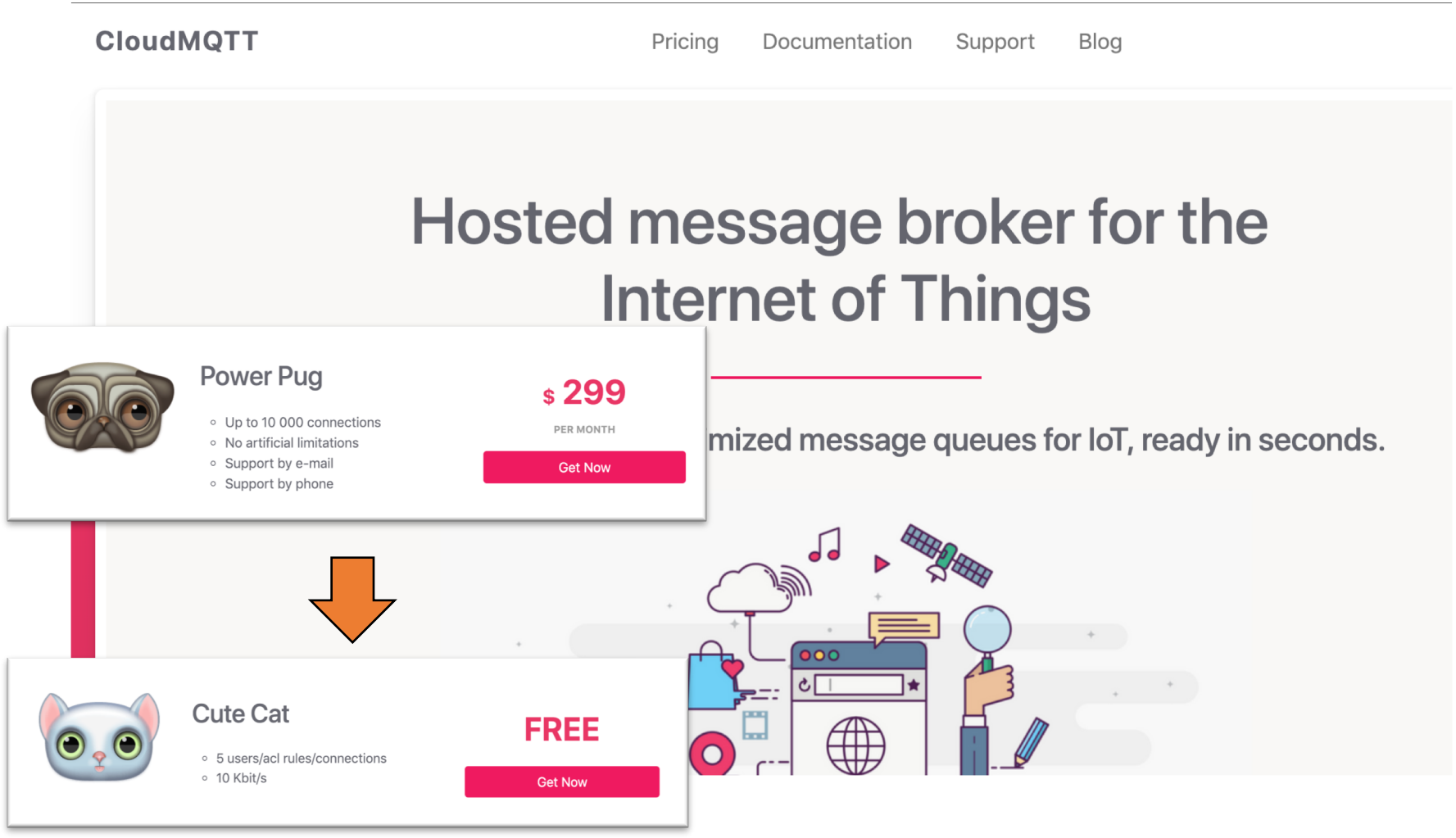
  - To check if the broker is running you can use the command:

    ```
    sudo netstat -tanlp | grep 1883
    ```

    - *note: "-tanlp" stands for: tcp, all, numeric, listening, program*

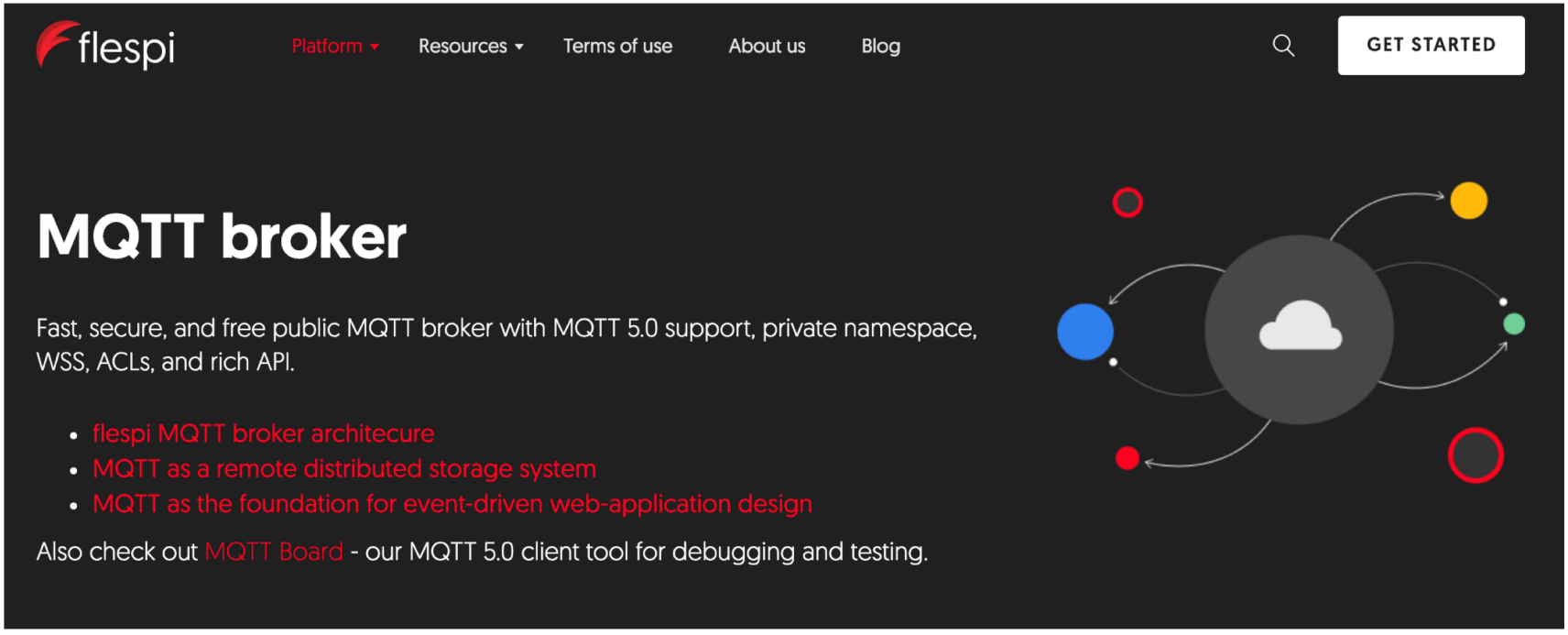
Cloud based MQTT brokers: CloudMQTT

https://www.cloudmqtt.com/ ➔ based on Mosquitto



Cloud based brokers: flespi

https://flespi.com/mqtt-broker



Cloud based brokers: flespi

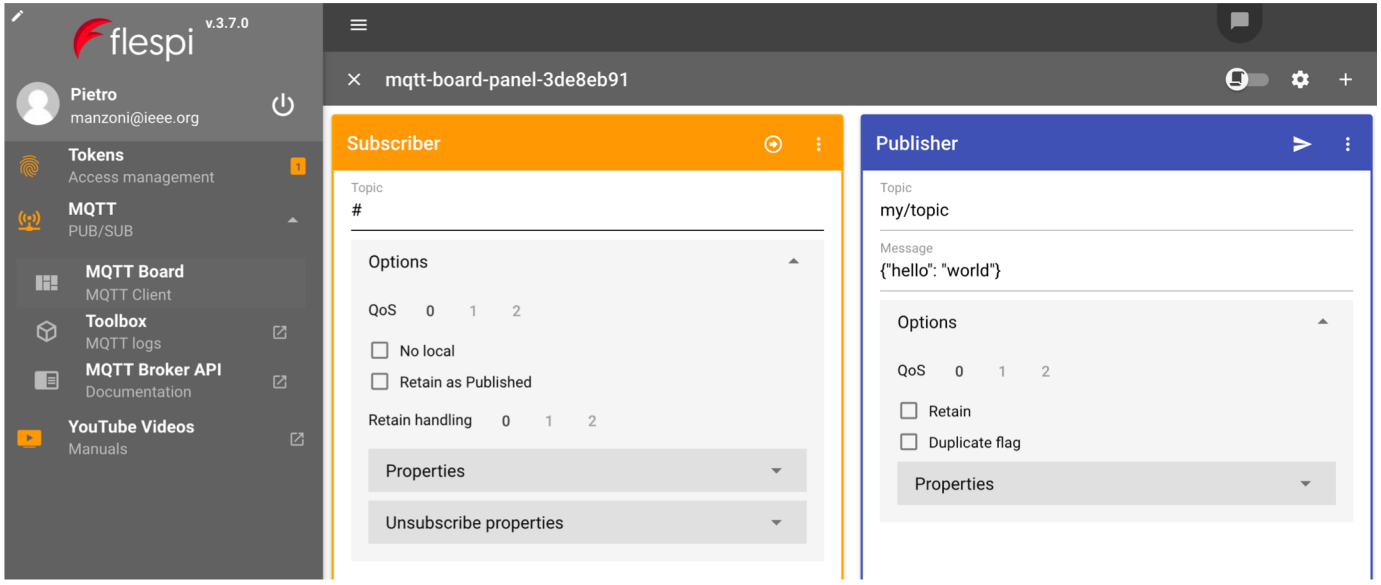
Terms of use

Free $0/mo

MQTT

100 active MQTT sessions

https://flespi.io/#/panel/mqttboard



https://flespi.com/mqtt-api

flespi MQTT broker connection details

- **Host** — **mqtt.flespi.io**.
  Port — **8883 (SSL)** or **1883 (non-SSL)**; for MQTT over WebSockets: **443 (SSL)** or **80 (non-SSL)**.
- **Authorization** — use a flespi platform token as MQTT session username; no password.
- **Client ID** — use any unique identifier within your flespi user session.
- **Topic** — you can publish messages to any topic except **flespi/**.
- **ACL** — both **flespi/** and **MQTT pub/sub** restrictions determined by the token.

```
I1RKMMIUJppLdlQoSgAQ8MvJPyNV9R2HIJgijO1S1gt5rajaeIOaiaKWwlHt2z1z
```

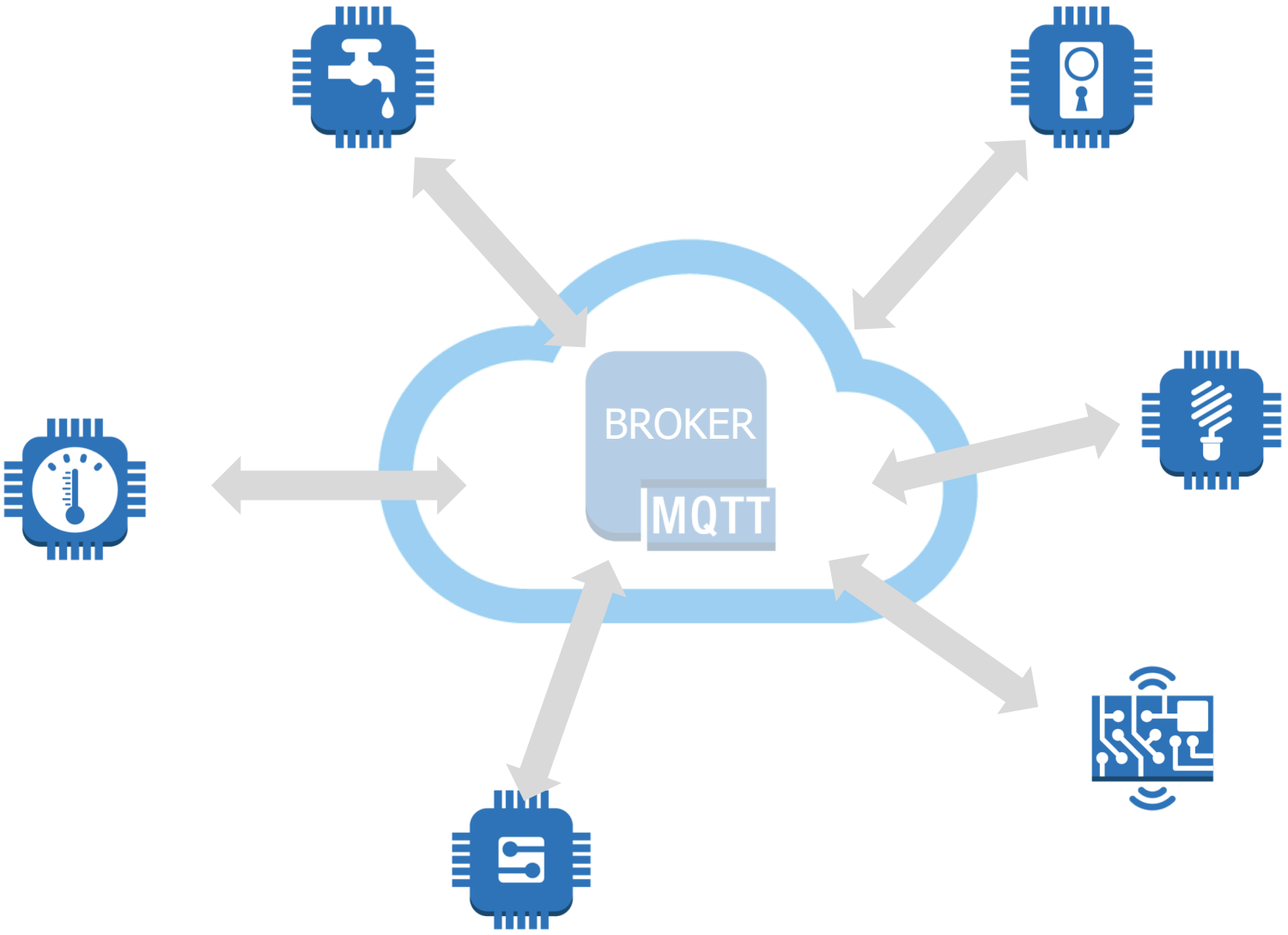
Open brokers ("Sandboxes")

- TCP based:
  - https://iot.eclipse.org/getting-started/#sandboxes
    - Hostname: **iot.eclipse.org**
  - http://test.mosquitto.org/
    - Hostname: **test.mosquitto.org**
  - https://www.hivemq.com/mqtt-demo/
    - Hostname: **broker.hivemq.com**
    - http://www.mqtt-dashboard.com/
  - Ports:
    - standard: 1883
    - encrypted: 8883 *(TLS v1.2, v1.1 or v1.0 with x509 certificates)*

- Websockets based:
  - broker.mqttdashboard.com port: 8000
  - test.mosquitto.org port: 8080
  - broker.hivemq.com port: 8000

- https://github.com/mqtt/mqtt.github.io/wiki/public_brokers

Creating clients



Clients for testing

- The Mosquitto broker comes with a couple of useful commands to quickly publish and subscribe to some topic.

- Their basic syntax is the following.
  - `mosquitto_sub -h HOSTNAME -t TOPIC`
  - `mosquitto_pub -h HOSTNAME -t TOPIC -m MSG`

- More information can be found:
  - https://mosquitto.org/man/mosquitto_sub-1.html
  - https://mosquitto.org/man/mosquitto_pub-1.html

MQTT clients: iOS

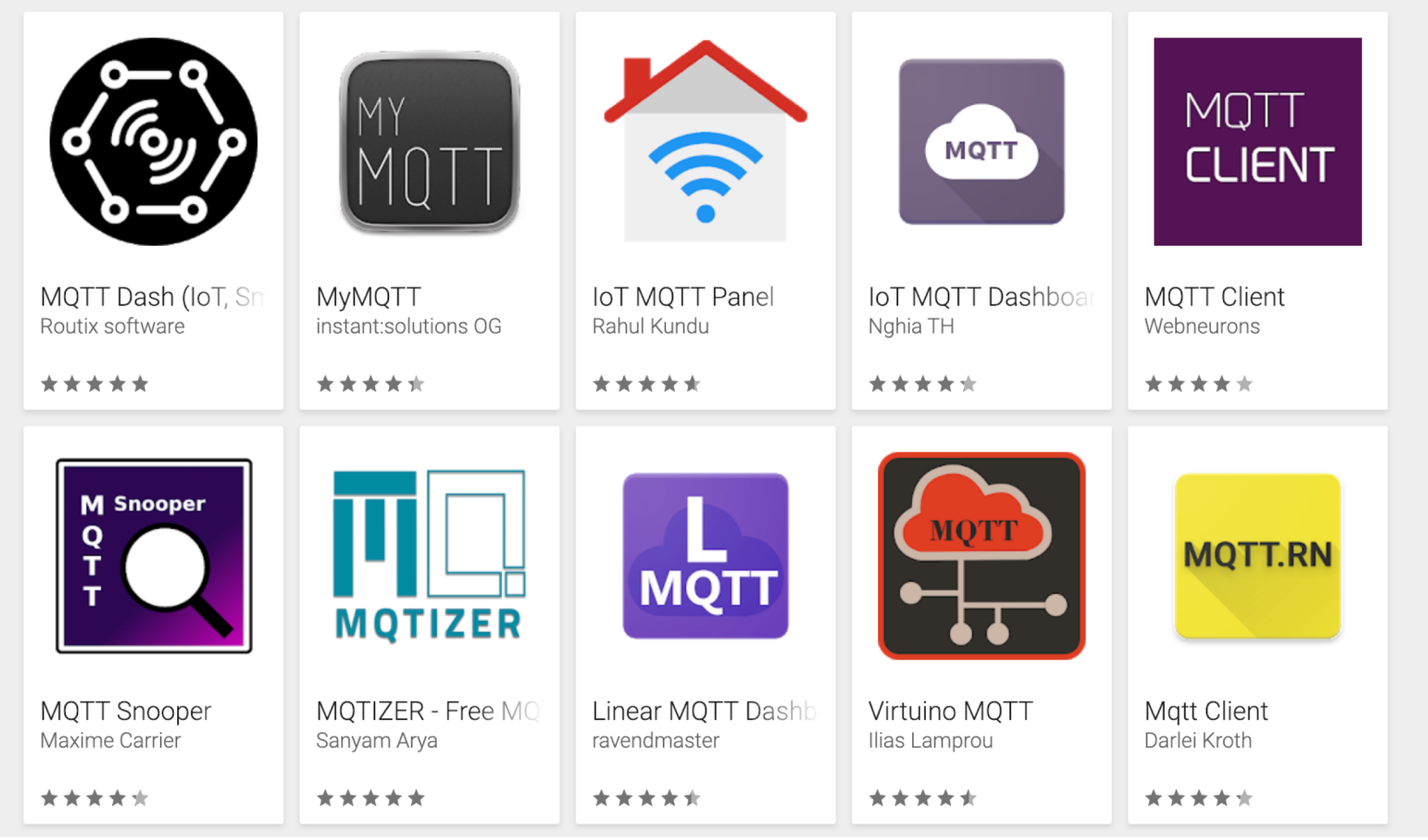
Mqttt

Utilidades

ICPDAS MQ...

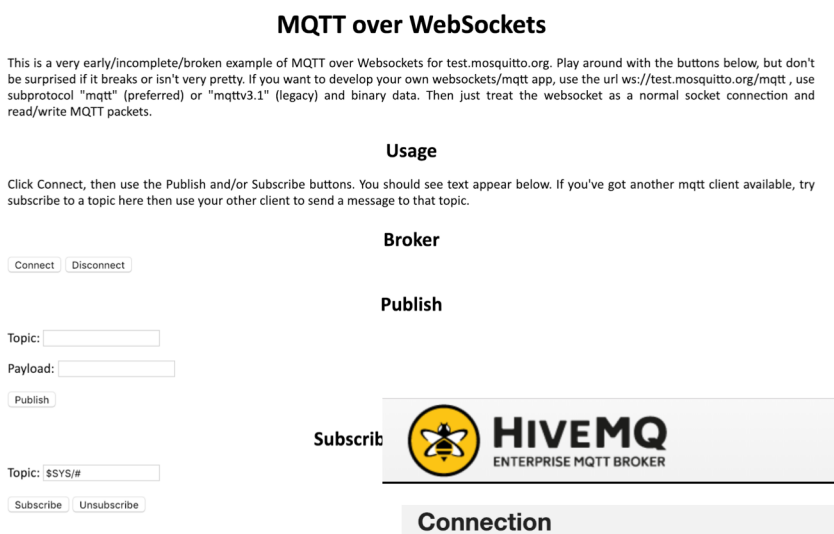
Utilidades

MQTT clients: Android

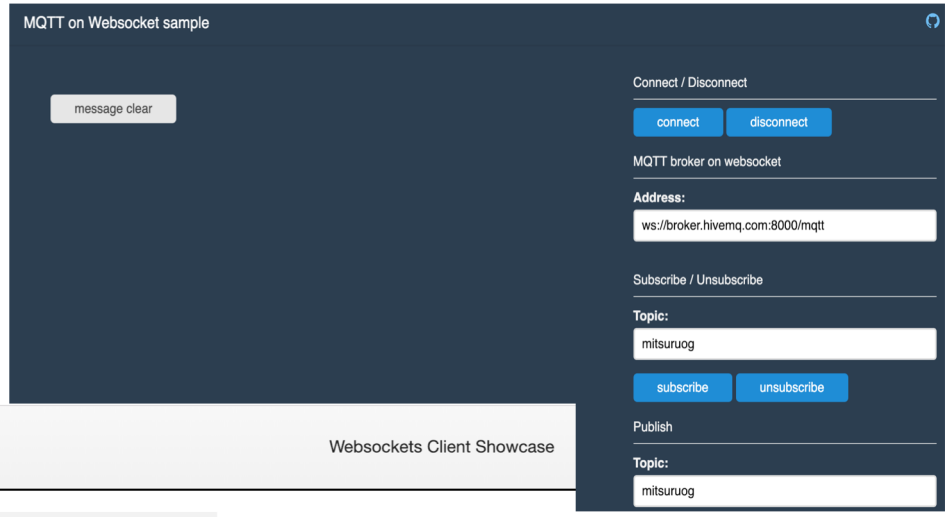


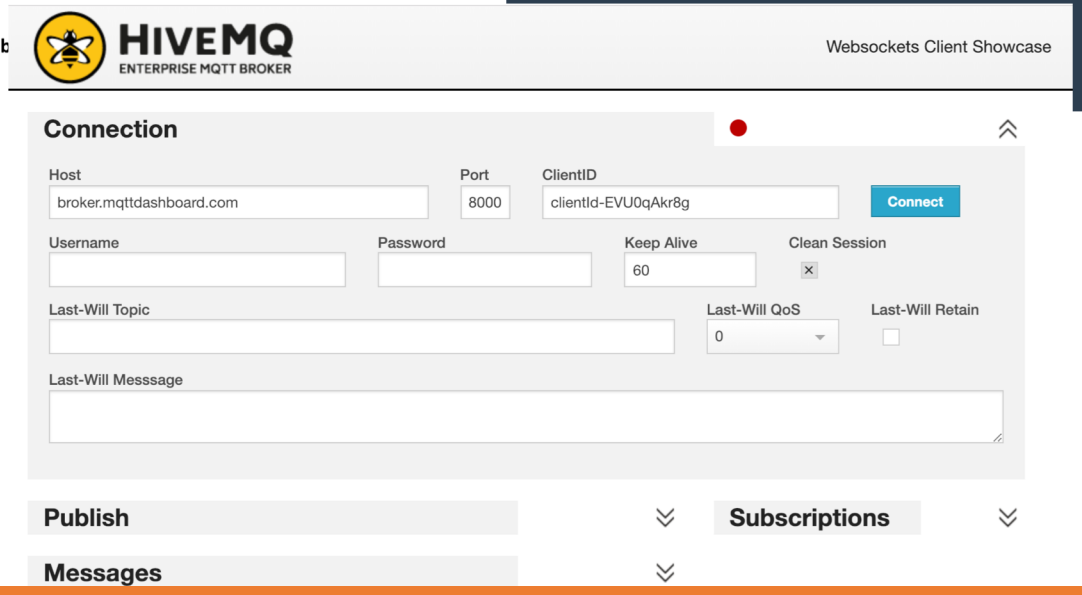
MQTT websocket clients

http://test.mosquitto.org/ws.html



http://mitsuruog.github.io/what-mqtt/

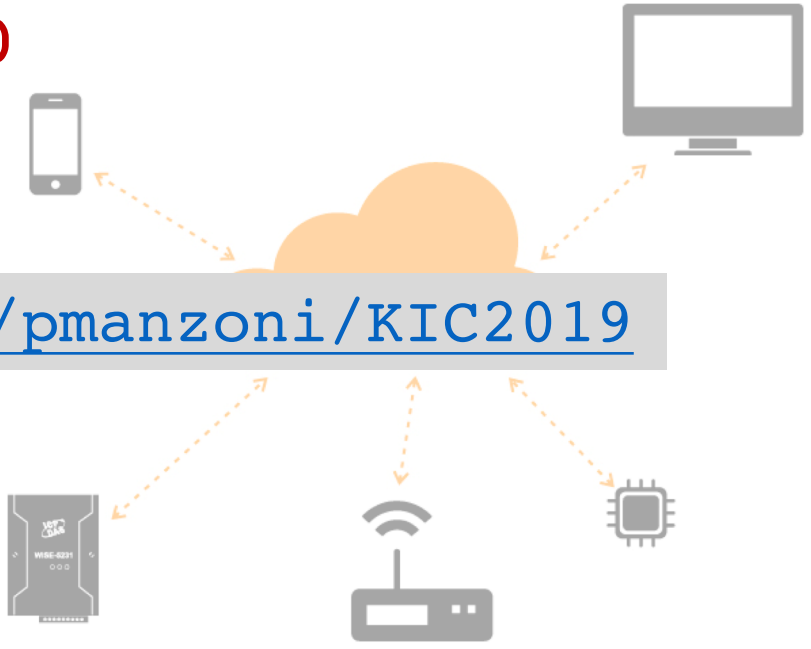




http://www.hivemq.com/demos/websocket-client/

Intro to MQTT

- Time for some exercise: **Lab 0**



https://github.com/pmanzoni/KIC2019

Intro to MQTT

- More time for some demo/exercise: **`Lab 0.1`**

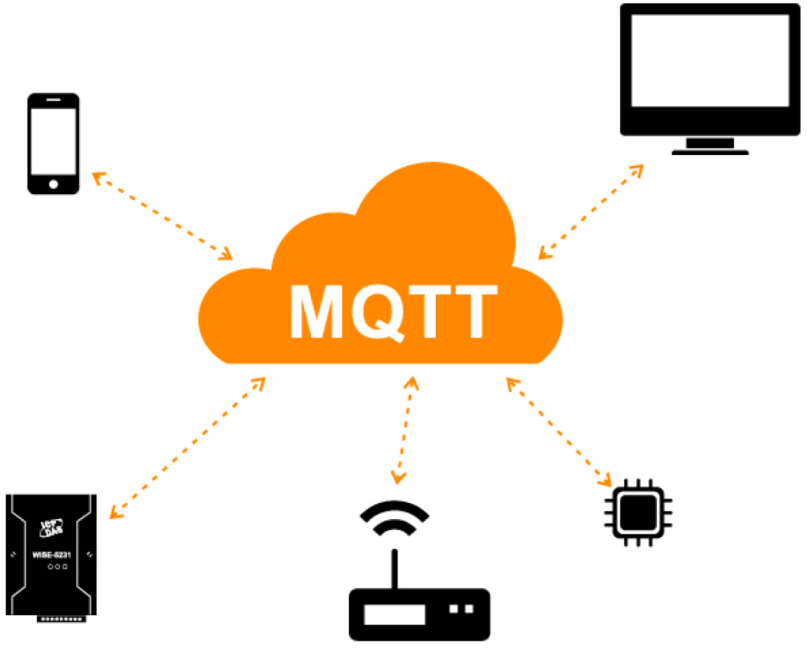
Broker address:

`192.168.1.121`
`port: 9001`

`https://www.raspberrypi.org/products/sense-hat/`

Intro to MQTT

- Some final details

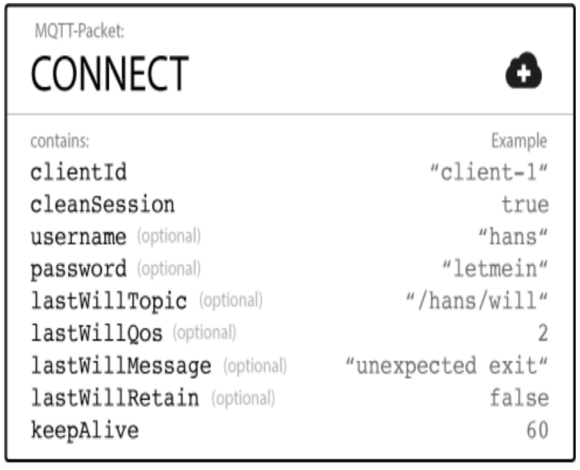


MQTT Keep alive

- The keep alive functionality assures that the connection is still open and both broker and client are connected to one another.
- The client specifies a time interval in seconds and communicates it to the broker during the establishment of the connection.
  - The interval is the longest possible period of time which broker and client can endure without sending a message.
  - If the broker doesn't receive a PINGREQ or any other packet from a particular client, it will close the connection and send out the last will and testament message (if the client had specified one).
- Good to Know
  - The MQTT client is responsible of setting the right keep alive value.
  - The maximum keep alive is 18h 12min 15 sec.
  - If the keep alive interval is set to **0**, the keep alive mechanism is deactivated.

"Will" message

- When clients connect, they can specify an optional "will" message, to be delivered if they are unexpectedly disconnected from the network.
  - (In the absence of other activity, a 2-byte ping message is sent to clients at a configurable interval.)
- This "last will and testament" can be used to notify other parts of the system that a node has gone down.



WHEN?

Persistent session

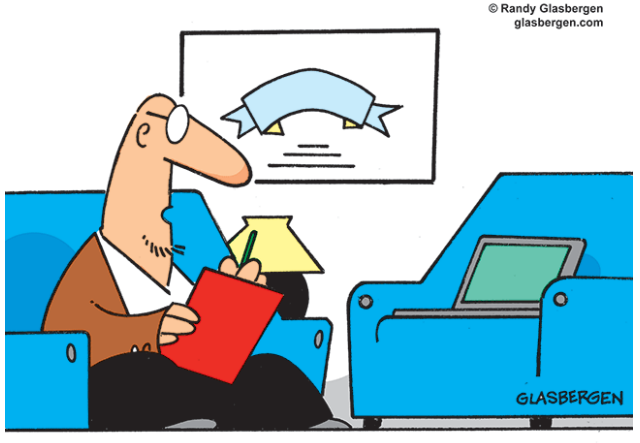
- A persistent session saves all information relevant for the client on the broker. The session is identified by the clientId provided by the client on connection establishment
- So what will be stored in the session?
  - Existence of a session, even if there are no subscriptions
  - All subscriptions
  - 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.
- Persistent session on the client side
  - Similar to the broker, each MQTT client must store a persistent session too. So when a client requests the server to hold session data, it also has the responsibility to hold some information by itself:
  - All messages in a QoS 1 or 2 flow, which are not confirmed by the broker
  - All received QoS 2 messages, which are not yet confirmed to the broker

Topics best practices

- First of all:
  - Don't use a leading forward slash
  - Don't use spaces in a topic
  - Use only ASCII characters, avoid non printable characters
- Then, try to..
  - Keep the topic short and concise
  - Use specific topics, instead of general ones
  - Don't forget extensibility
- Finally, be careful and don't subscribe to #

A few words on security

- MQTT has the option for Transport Layer Security (TLS) encryption.
- MQTT also provides username/password authentication with the broker.
  - Note that the password is transmitted in clear text. Thus, be sure to use TLS encryption if you are using authentication.



"It's not just you. We're all insecure in one way or another."

Smart homes can be easily hacked via unsecured MQTT servers

https://www.helpnetsecurity.com/2018/08/20/unsecured-mqtt-servers/

In fact, by using the Shodan IoT search engine, Avast researchers found over 49,000 MQTT servers exposed on the Internet and, of these, nearly 33,000 servers have no password protection, allowing attackers to access them and all the messages flowing through it.

TOTAL RESULTS

49,197

TOP COUNTRIES

China	12,151
United States	8,257
Germany	3,092
Korea, Republic of	2,003
Hong Kong	2,002

TOTAL RESULTS

32,888

TOP COUNTRIES

China	8,446
United States	4,733
Germany	1,719
Hong Kong	1,614
Taiwan	1,565

Advantages of MQTT

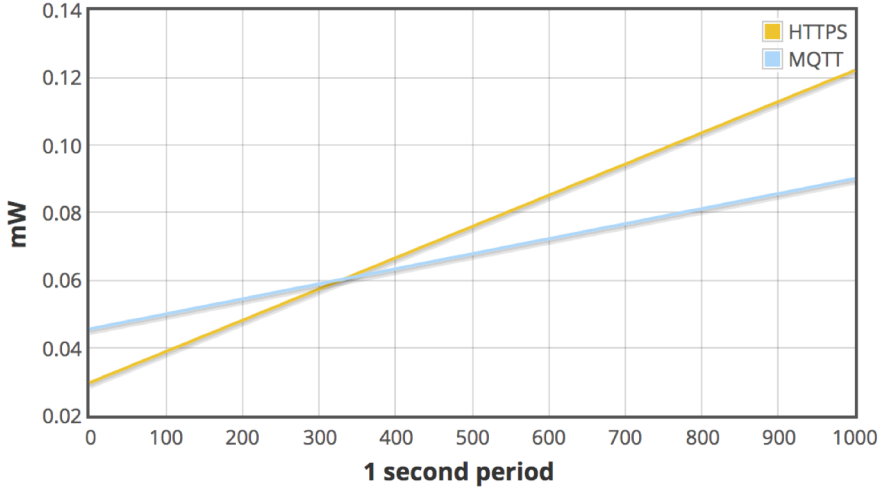
- **Push based**: no need to continuously look for updates
- It has built-in function useful for reliable behavior in an unreliable or intermittently connected wireless environments.
  1. "last will & testament" so all apps know immediately if a client disconnects ungracefully,
  2. "retained message" so any user re-connecting immediately gets the very latest information, etc.
- Useful for one-to-many, many-to-many applications
- Small memory footprint protocol, with reduced use of battery

Energy usage: some number

amount of power taken to establish the initial connection to the server:

% Battery Used			
3G		Wifi	
HTTPS	MQTT	HTTPS	MQTT
0.02972	0.04563	**0.00228**	0.00276

3G – 240s Keep Alive – % Battery Used Creating and Maintaining a Connection



cost of 'maintaining' that connection (in % Battery / Hour):

	% Battery / Hour			
	3G		Wifi	
Keep Alive (Seconds)	HTTPS	MQTT	HTTPS	MQTT
60	1.11553	**0.72465**	0.15839	**0.01055**
120	0.48697	**0.32041**	0.08774	**0.00478**
240	0.33277	**0.16027**	0.02897	**0.00230**
480	0.08263	**0.07991**	0.00824	**0.00112**

you'd save ~4.1% battery per day just by using MQTT over HTTPS to maintain an open stable connection.

http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https

MQTT disadvantages

- If the broker fails…
- Does not define a standard client API, so application developers have to select the best fit.
- Does not include many features that are common in Enterprise Messaging Systems like:
  - expiration, timestamp, priority, custom message headers, …
- Does not have a point-to-point (aka queues) messaging pattern
  - Point to Point or One to One means that there can be more than one consumer listening on a queue but only one of them will be get the message
- Maximum message size 256MB


Thank
You
Merci
Gracias
TerimaKasih
Hvala
Arigatô
Spacibo
Danke
KamSahHamnida
Mèsi
Xièxie
Chokrane
Grazie
aDank
Takk
ThintKo
Buznyg
Blagodaram
Rahmat
Matondo
Mercé
Taiku
Dakujem
Gràcies
Faleminderit
Efharisto
NajisTuke
Grandmerce
Tanemirt
Tenki
Obrigado
Salamat
Saha
Multumesc
CámOn
Gracie
GratiasAgo
KopKhunKha
Motashakkeram
Sagolun
Tak
Dankie
TänanVäga
Ačiū
GratiasAgimus
Dziakuju
Kiitos
Mauruuru
Chnorakaloutioun
Bayarlalaa
Nirringrazzjak
TapadhLeat
Köszönöm
Murakoze
WaadMahadsantahay
BarakAllahuFiik
Grazzi
Barkal
Aabhar
Dhanyavaad
KhobChaiDeu
Gratzias