

# *Intro to MQTT*

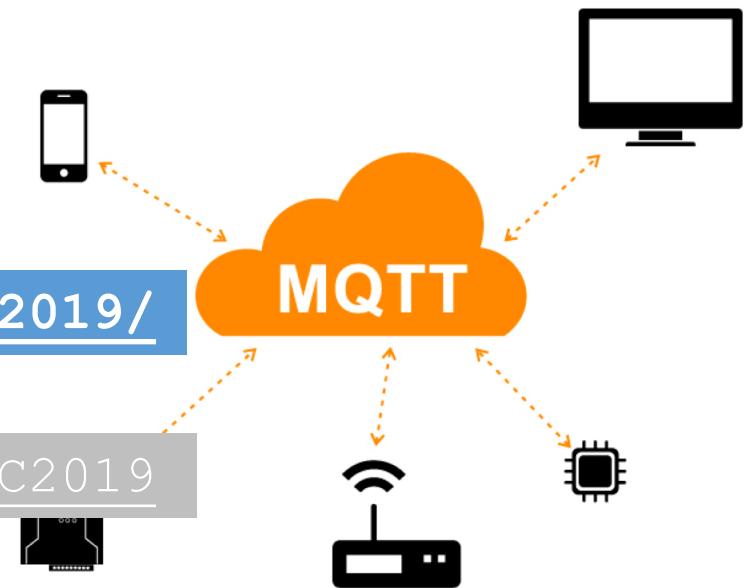
Pietro Manzoni

Universitat Politecnica de Valencia (UPV)  
Valencia - SPAIN

pmanzoni@disca.upv.es

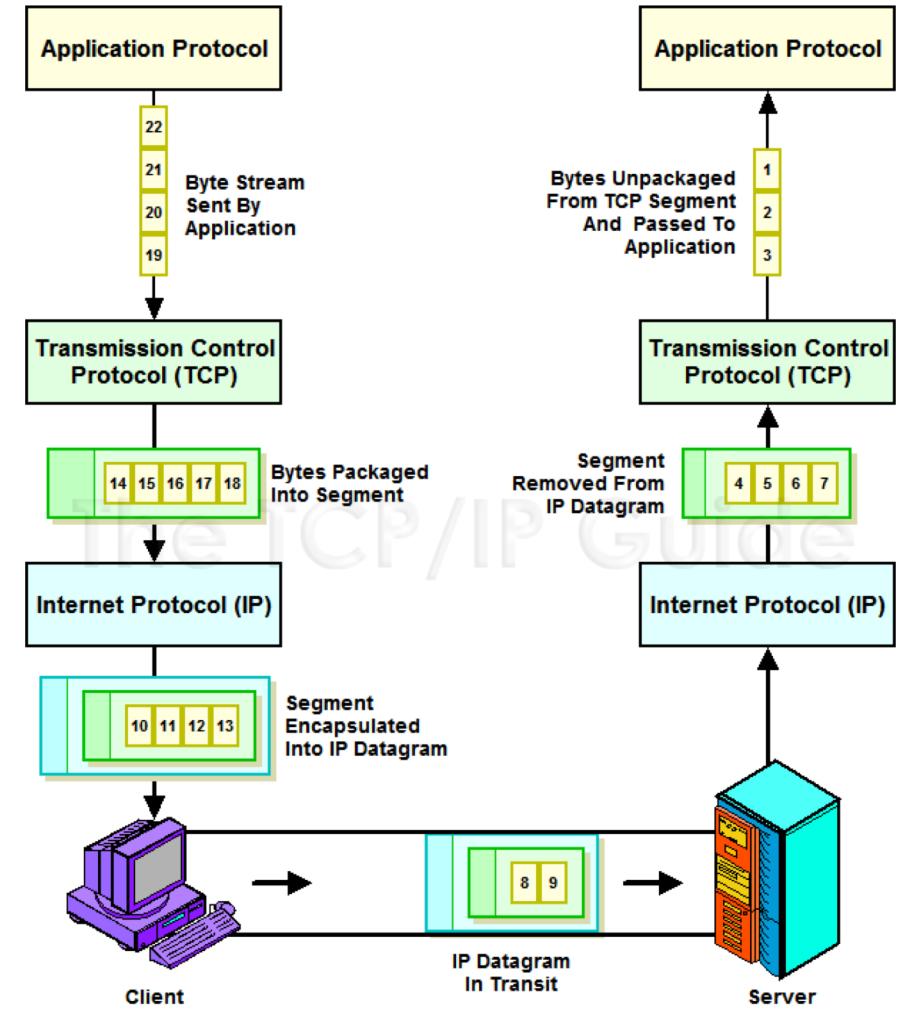
<https://pmanzoni.github.io/KIC2019/>

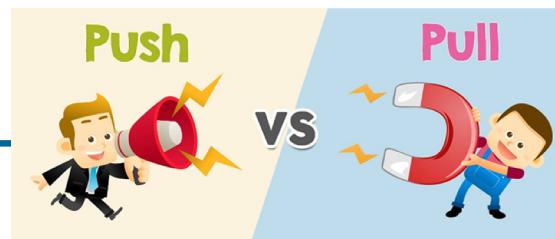
<https://github.com/pmanzoni/KIC2019>



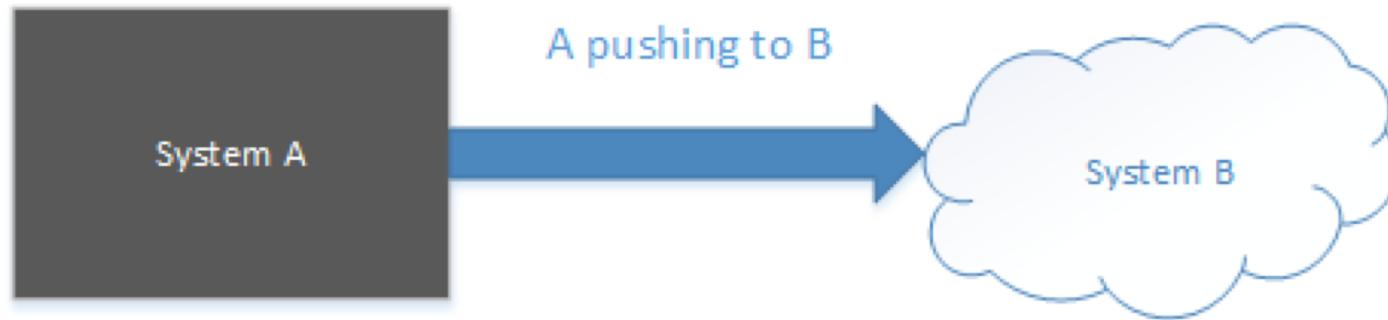
# From “byte streams” to “messages”

- 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,



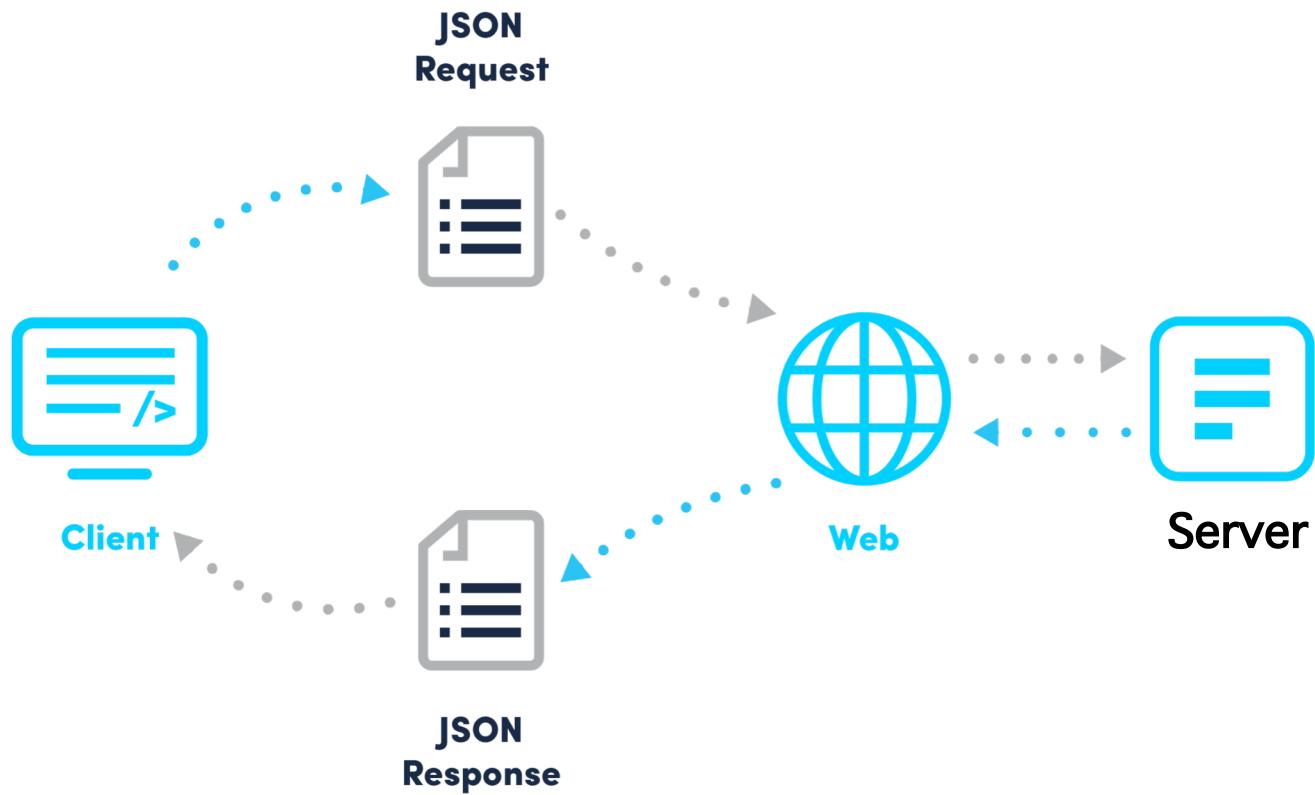


## Ways to interchange “messages”



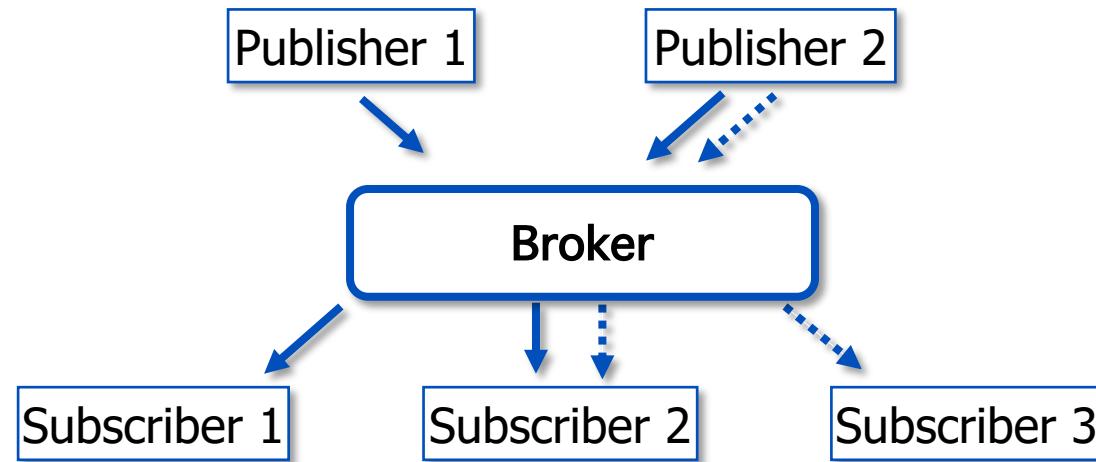
# Request/response approach

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



- Publish/Subscriber

- aka: producer/consumer



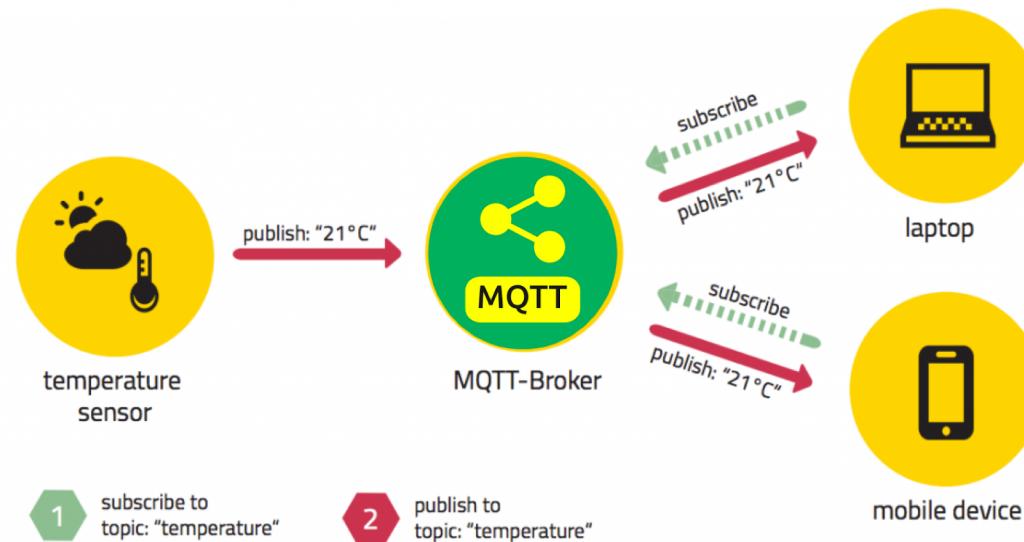
- Various protocols:

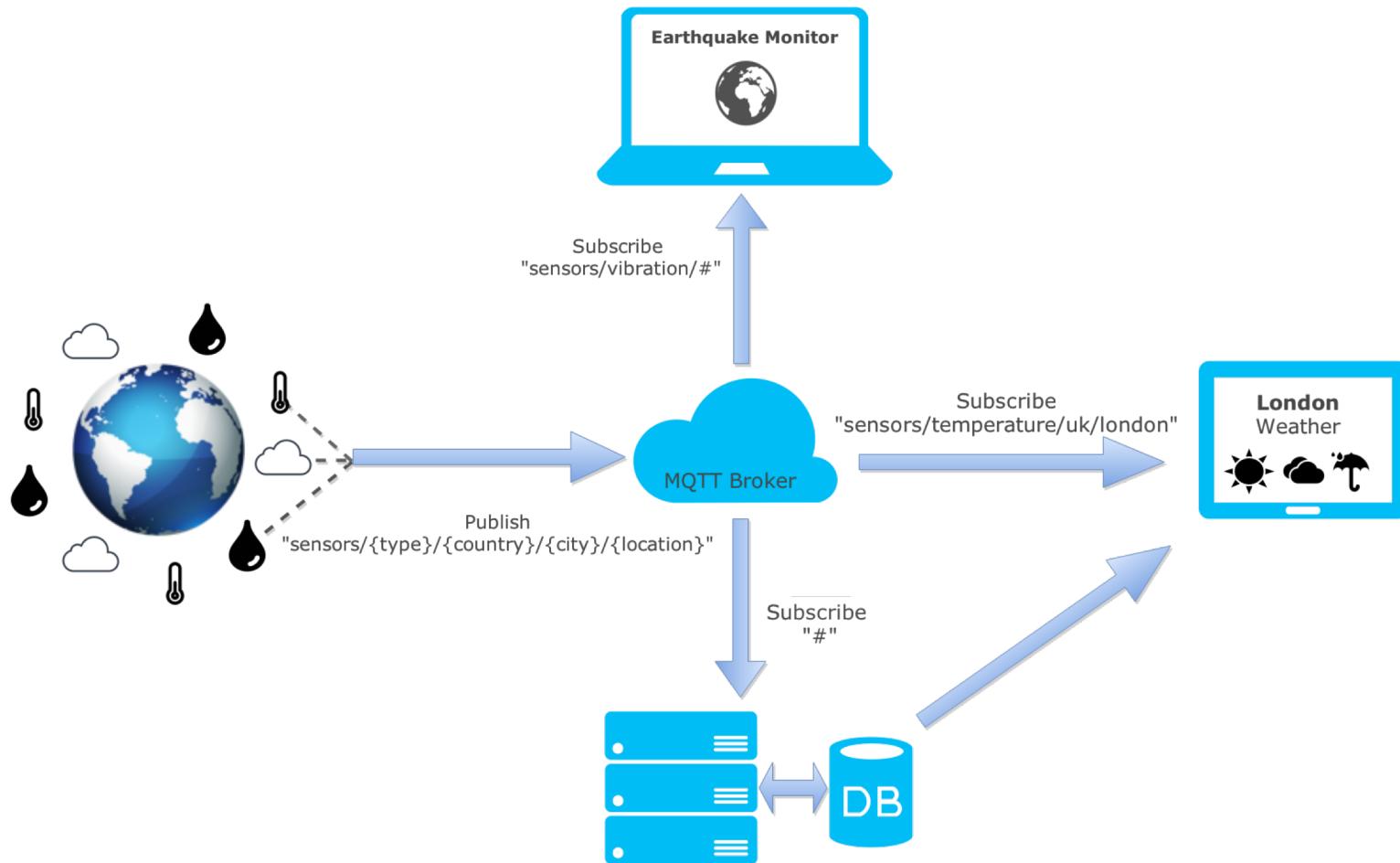
- MQTT, AMQP, XMPP (was Jabber)

- Growing technique

- E.g., <https://cloud.google.com/iot/docs/how-tos/mqtt-bridge>

- Pub/Sub separates 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.

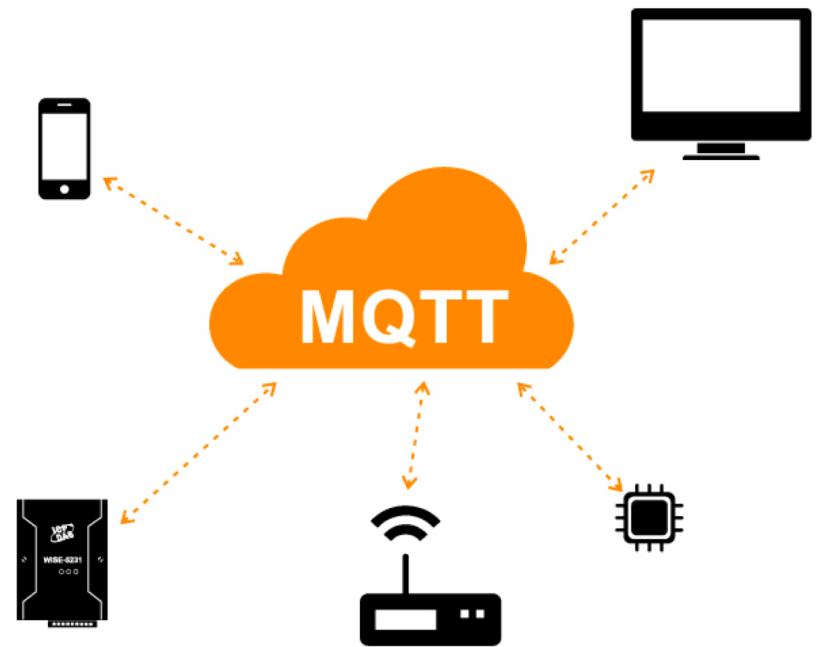
HiveMQ<sup>®</sup>



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

# *Intro to MQTT*

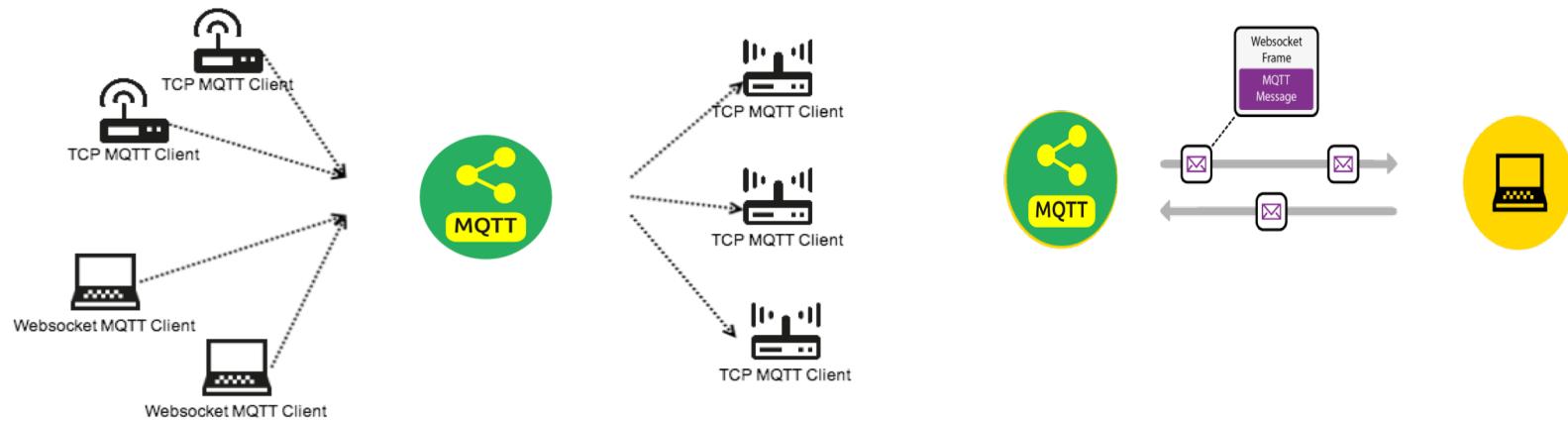
- Fundamental concepts



- 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/>

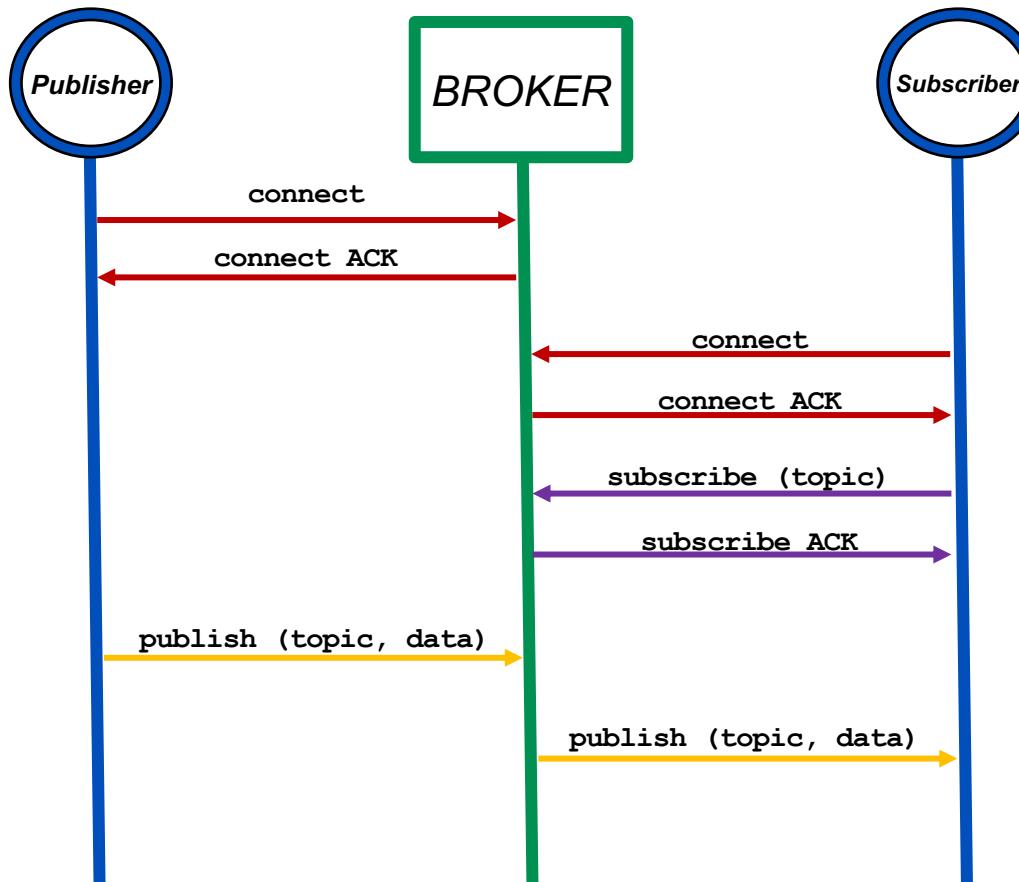
- **MQTT 3.1.1 is the current version of the protocol.**
  - Standard document here:
    - <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html>
  - October 29th 2014: MQTT was officially approved as OASIS Standard.
    - [https://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=mqtt](https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=mqtt)
- 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](https://www.youtube.com/watch?time_continue=3&v=YIpesv_bJgU)

- 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



- 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.**

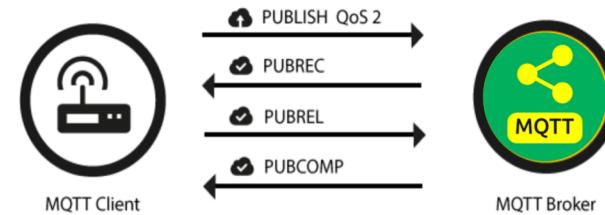


Special \$SYS/ topics

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

- 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

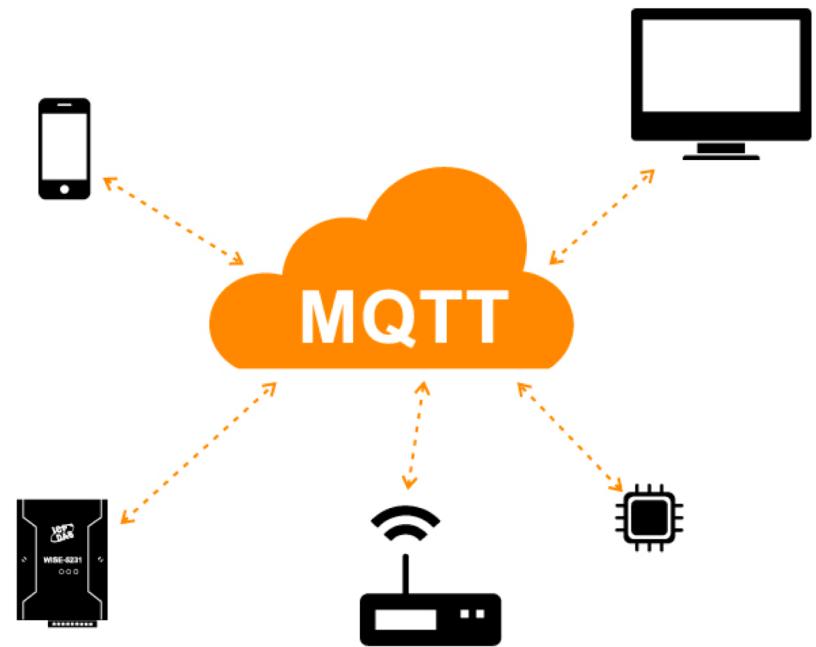
- Messages are published with a **Quality of Service (QoS)** level, which specifies delivery requirements.
- A **QoS 0 ("at most once")** message is fire-and-forget.
  - For example, a notification from a doorbell may only matter when immediately delivered.
- With **QoS 1 ("at least once")**, the broker 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.



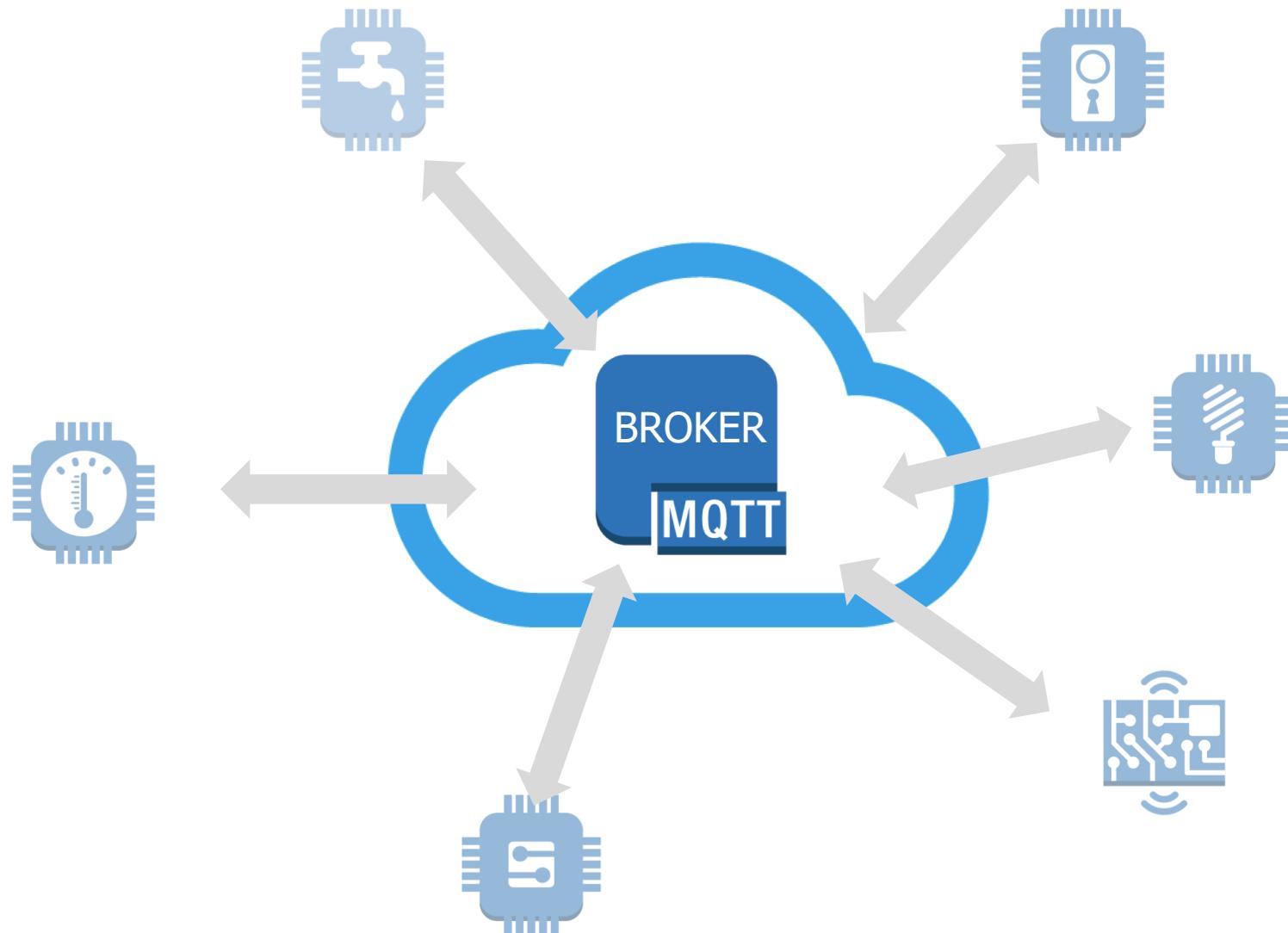
- A retained message is a normal MQTT message with the **retained flag set to true**. The broker will store the last retained message and the corresponding QoS for that topic
  - Each client that subscribes to a topic pattern, which matches the topic of the retained message, will receive the message 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



- 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>

- 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

CloudMQTT

Pricing

Documentation

Support

Blog

## Hosted message broker for the Internet of Things



Power Pug

- Up to 10 000 connections
- No artificial limitations
- Support by e-mail
- Support by phone

\$ 299

PER MONTH

Get Now

mized message queues for IoT, ready in seconds.



Cute Cat

- 5 users/acl rules/connections
- 10 Kbit/s

FREE

Get Now



<https://flespi.com/mqtt-broker>

**MQTT broker**

Fast, secure, and free public MQTT broker with MQTT 5.0 support, private namespace, WSS, ACLs, and rich API.

- flespi MQTT broker architecure
- MQTT as a remote distributed storage system
- MQTT as the foundation for event-driven web-application design

Also check out [MQTT Board](#) - our MQTT 5.0 client tool for debugging and testing.

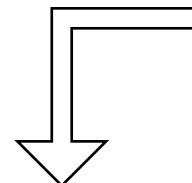
# Cloud based brokers: flespi

## Terms of use

Free	\$0/mo
MQTT	
100 active MQTT sessions	

The screenshot shows the flespi v3.7.0 mobile application interface. On the left is a sidebar with user info (Pietro manzoni@ieee.org), tokens, MQTT (selected), and links to MQTT Board, Toolbox, and MQTT Broker API. The main area has tabs for 'Subscriber' and 'Publisher'. The 'Subscriber' tab shows a topic '#', QoS levels (0, 1, 2), and checkboxes for 'No local' and 'Retain as Published'. The 'Publisher' tab shows a topic 'my/topic', message content '{"hello": "world"}', and QoS levels (0, 1, 2). Both tabs have sections for 'Options', 'Properties', and 'Unsubscribe properties'.

<https://flespi.com/mqtt-api>



## flespi MQTT broker connection details

- **Host** — [mqqt.flespi.io](https://mqqt.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](#).

I1RKMMIUJpp1QoSgAQ8MvDUJWNNJ9R2HIJgijo1S1gt5rajaeIOaiaKWwlHt2z1z

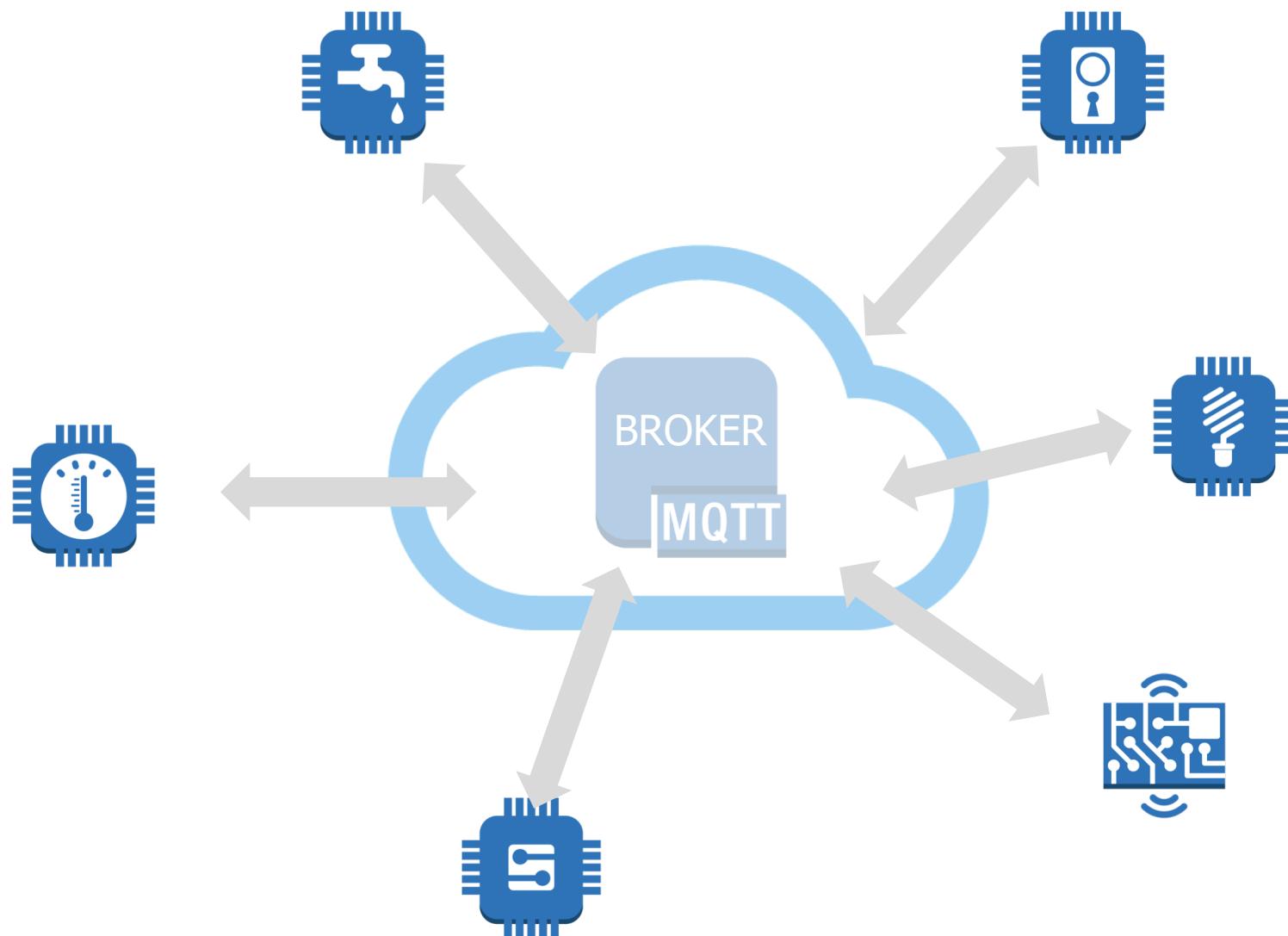
## ○ TCP based:

- <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

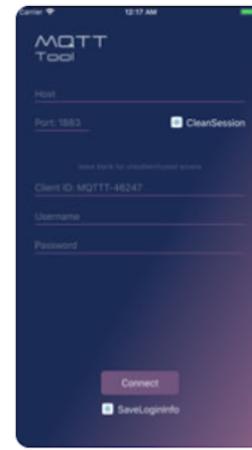
## ○ [https://github.com/mqtt/mqtt.github.io/wiki/public\\_brokers](https://github.com/mqtt/mqtt.github.io/wiki/public_brokers)



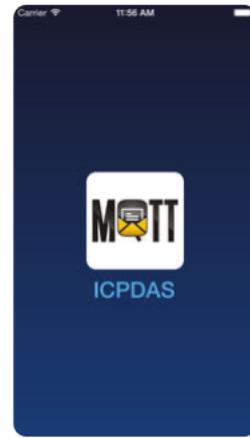
- The Mosquitto broker comes with a couple of useful commands to quickly publish and subscribe to some topic.
- Their basic syntax is the following.
  - `mosquitto_sub -h HOSTNAME -t TOPIC`
  - `mosquitto_pub -h HOSTNAME -t TOPIC -m MSG`
- More information can be found:
  - [https://mosquitto.org/man/mosquitto\\_sub-1.html](https://mosquitto.org/man/mosquitto_sub-1.html)
  - [https://mosquitto.org/man/mosquitto\\_pub-1.html](https://mosquitto.org/man/mosquitto_pub-1.html)

# MQTT clients: iOS

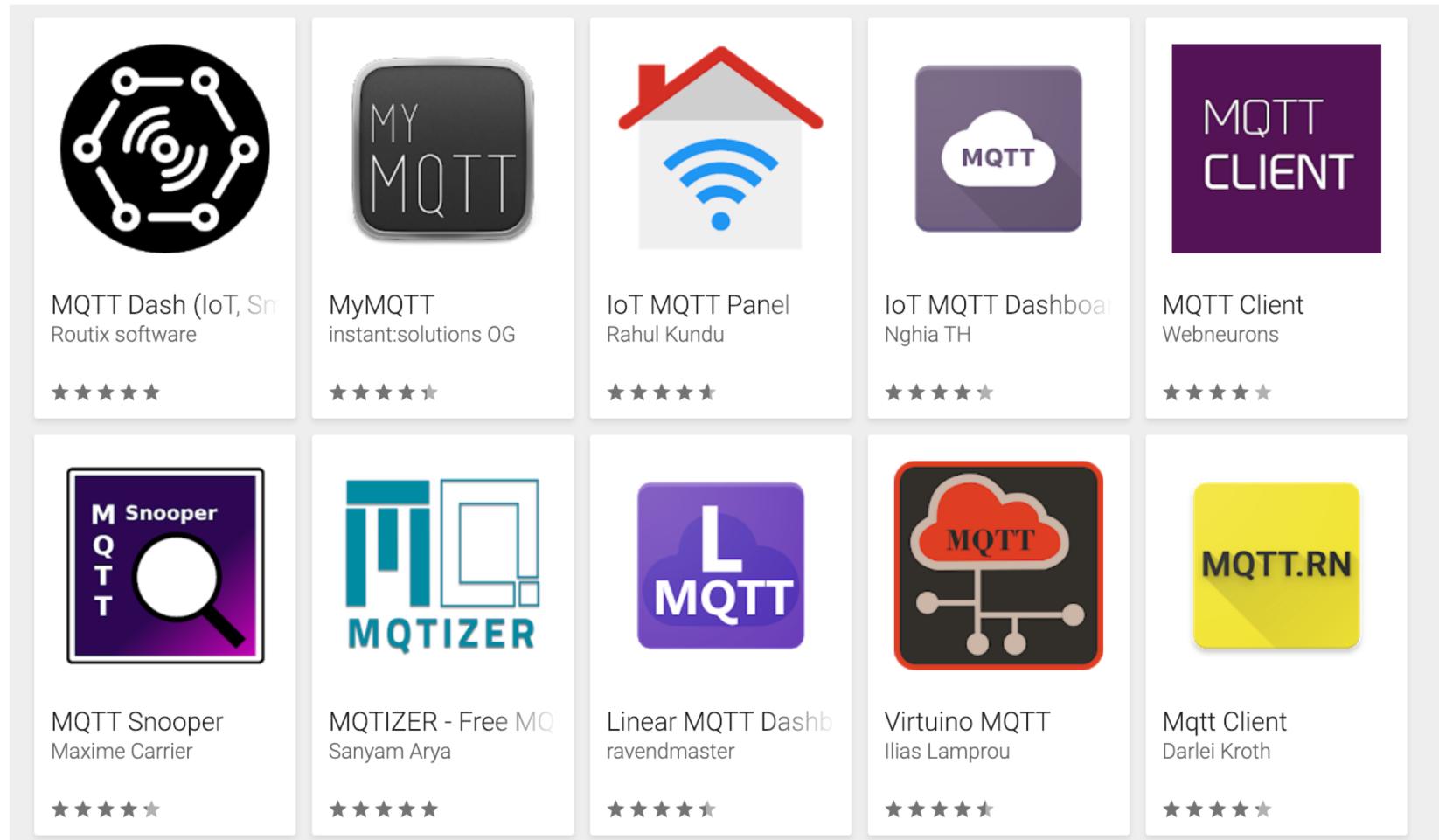
Mqttt  
Utilidades



ICPDAS MQ...  
Utilidades



# MQTT clients: Android



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

### MQTT over WebSockets

This is a very early/incomplete/broken example of MQTT over Websockets for test.mosquitto.org. Play around with the buttons below, but don't be surprised if it breaks or isn't very pretty. If you want to develop your own websockets/mqtt app, use the url ws://test.mosquitto.org/mqtt , use subprotocol "mqtt" (preferred) or "mqqtv3.1" (legacy) and binary data. Then just treat the websocket as a normal socket connection and read/write MQTT packets.

#### Usage

Click Connect, then use the Publish and/or Subscribe buttons. You should see text appear below. If you've got another mqtt client available, try subscribe to a topic here then use your other client to send a message to that topic.

#### Broker

[Connect](#) [Disconnect](#)

#### Publish

Topic:

Payload:

[Publish](#)

Subscribe



Websockets Client Showcase

#### Connection

Host: broker.mqttdashboard.com

Port: 8000

ClientID: clientId-EVU0qAkr8g

[Connect](#)

Username:

Password:

Keep Alive: 60

Clean Session:

Last-Will Topic:

Last-Will QoS: 0

Last-Will Retain:

Last-Will Message:

[Publish](#)

[Subscriptions](#)

[Messages](#)

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

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

MQTT on Websocket sample

message clear

Connect / Disconnect

[connect](#) [disconnect](#)

MQTT broker on websocket

Address: ws://broker.hivemq.com:8000/mqtt

Subscribe / Unsubscribe

Topic: mitsuruog

[subscribe](#) [unsubscribe](#)

Publish

Topic: mitsuruog

<https://pmanzoni.github.io/KIC2019/>

## / IoT Labs

### // Session 1: Tuesday, June 11th, 2019

- Intro to IoT
- Intro to MQTT
- Lab 1: MQTT using a web client



### // Session 2: Wednesday, June 12th, 2019

- Intro to TTN and LoRaWAN
- LAB 2: Installing a gateway,
- LAB 3: sending sensors data with LoRaWAN via TTN to Ubidots

### // Session 3: Friday, June 14th, 2019

- LAB 4: InfluxDB, Telegraf and Graphana

# Some more exercises: “pull” data

Broker  **mosquitto**

IP: *indicated in class*  
port: 9001

Client

1) periodically (every 5sec) **pushes** the temperature using topic:  
sensehat/temperature

2) subscribed to:  
peek/temperature used to pull data with topic: sensehat/temperature  
peek/pressure used to pull data with topic: sensehat/pressure  
peek/humidity used to pull data with topic: sensehat/pressure

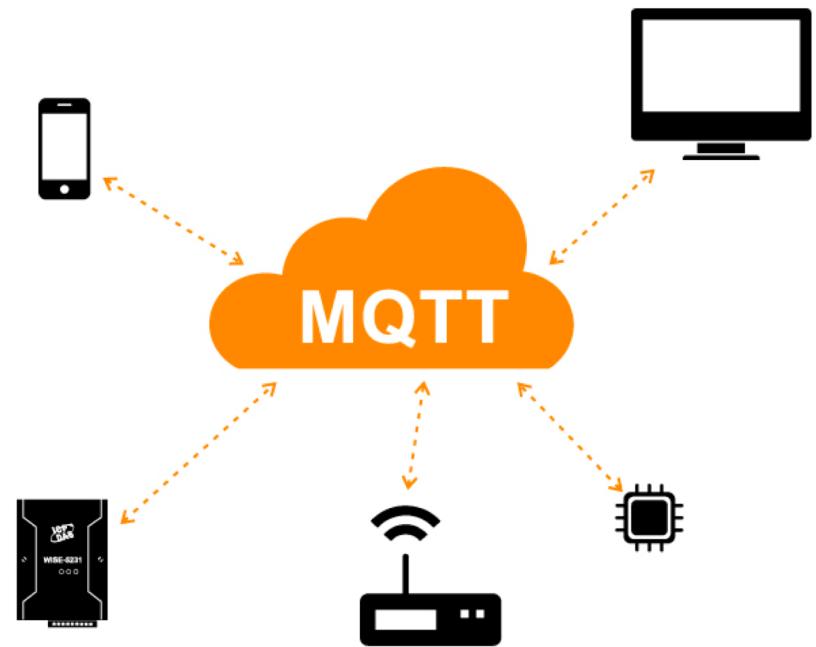
show/sun  
show/cloud  
show/rain  
show/message

<https://www.raspberrypi.org/products/sense-hat/>



# *Intro to MQTT*

- Some final details



- 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.

- 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.

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

WHEN?

- 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

- 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 #

Why?

- 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.



## 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

# CONCLUSIONS: 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.
  - “last will & testament” so all apps know immediately if a client disconnects ungracefully,
  - “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
  - E.g., <http://stephendnicholas.com/posts/power-profiling-mqtt-vs-https>

# CONCLUSIONS: disadvantages of MQTT

- 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

A large word cloud centered around the words "Thank You". The words are in various colors (red, orange, yellow, green, blue) and sizes, representing different languages. Some of the visible words include "Thank You" in English, "Merci" in French, "Gracias" in Spanish, "Danke" in German, "Grazie" in Italian, "Chokrane" in Kazakh, "Xièxie" in Chinese, "Motashakkeram" in Korean, and many others from various cultures and languages.

