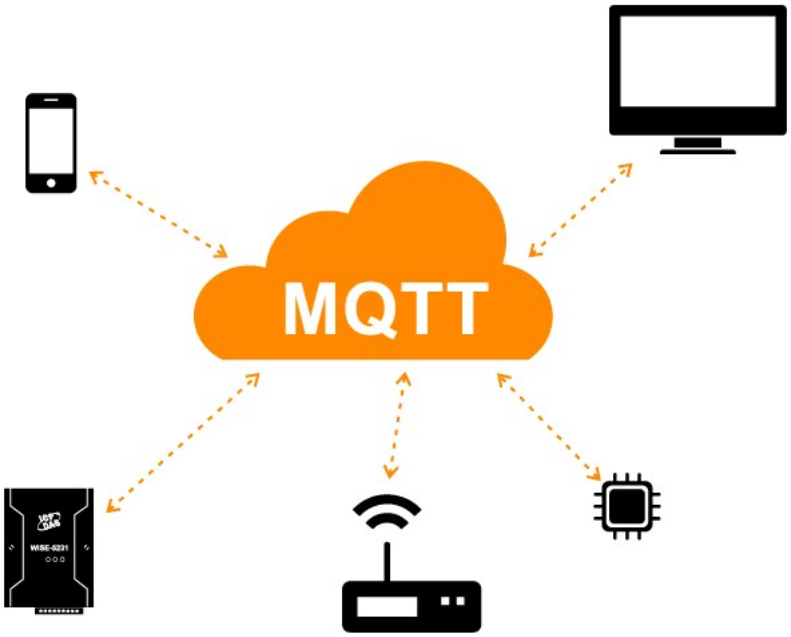
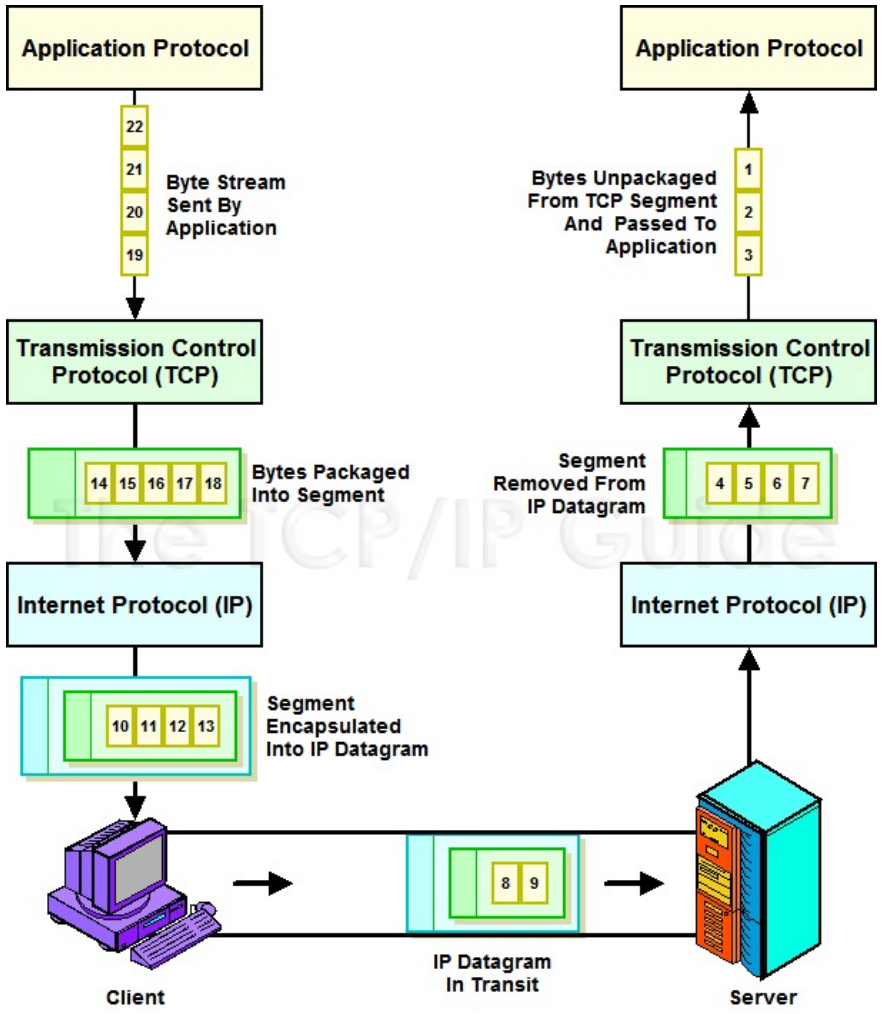
MQTT basics

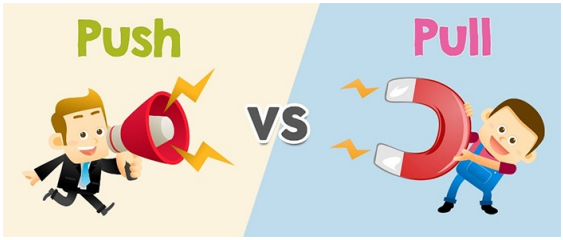


From "byte streams" to "messages"

- 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,

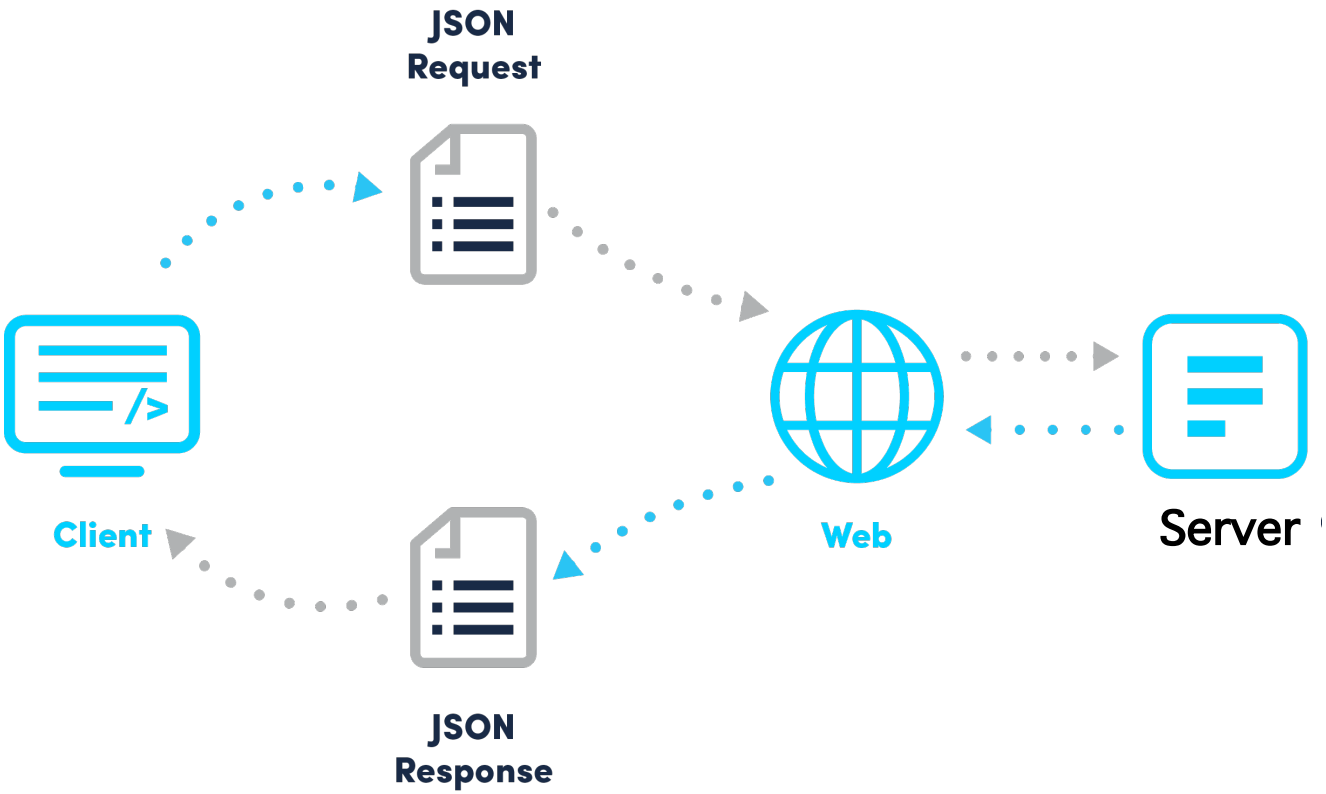


Ways to interchange “messages”



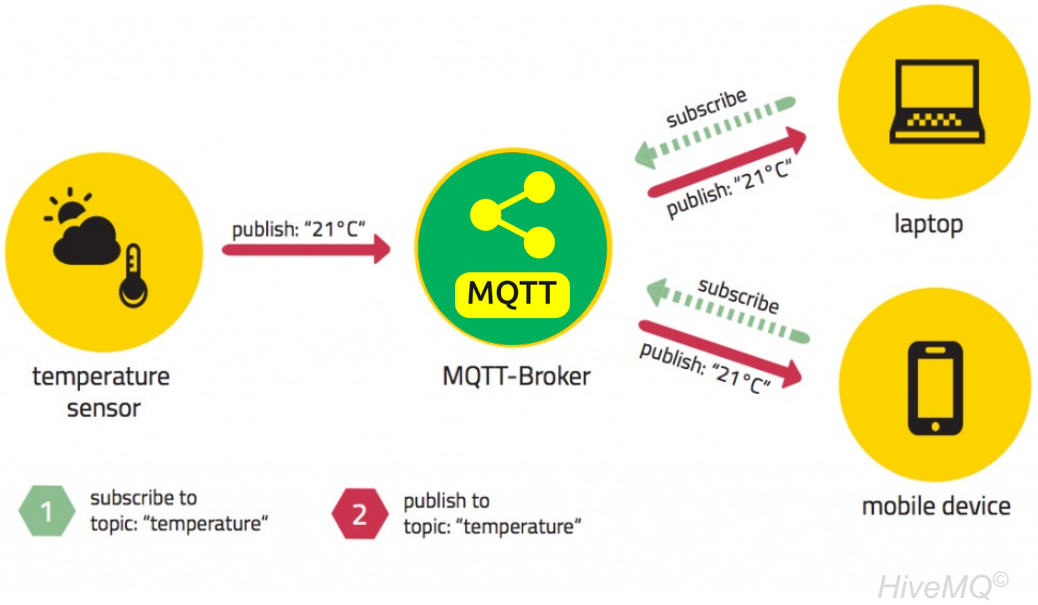
Request/response approach

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



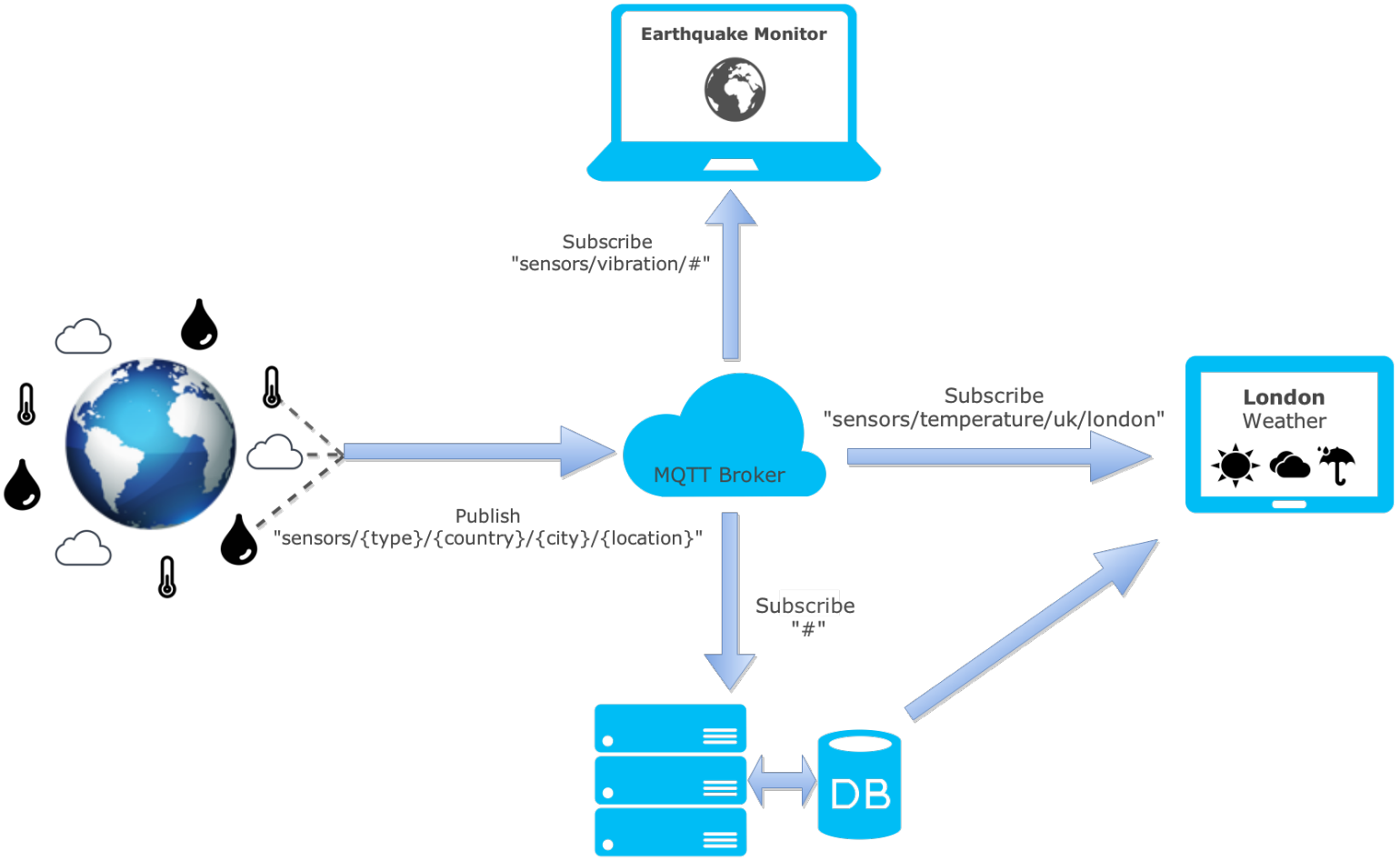
Pub/sub approach

- 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.



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

An example



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

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

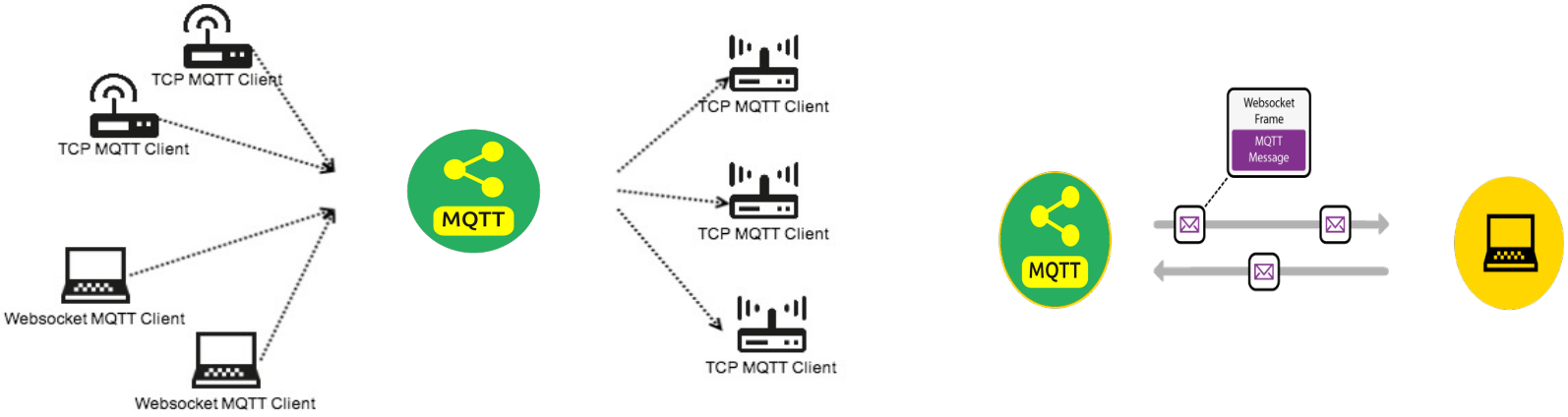
- **MQTT 3.1.1 is the "most widely used" version of the protocol.**
  - October 29th 2014: MQTT was officially approved as OASIS Standard.
  - Standard document here:
    - http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html

- 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…

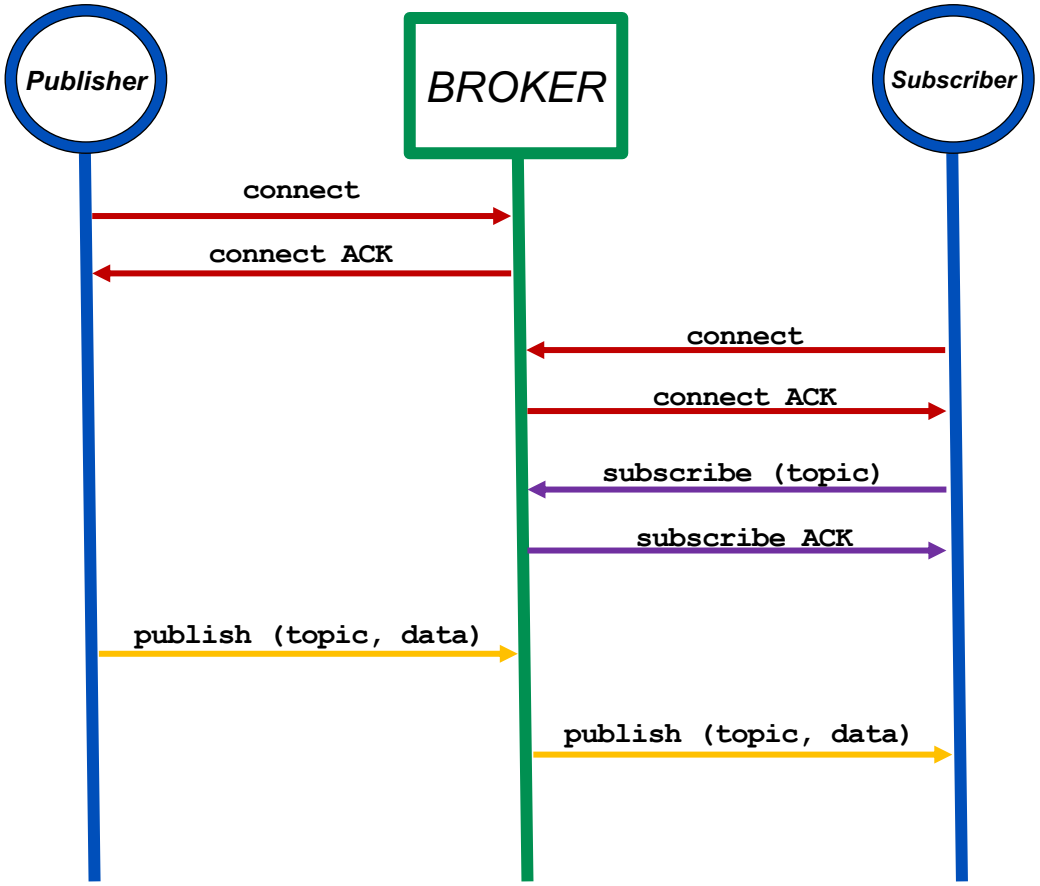
- mainly of TCP
  - There is also the closely related MQTT for Sensor Networks (MQTT-SN) where TCP is replaced by UDP → TCP stack is too complex for WSN

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



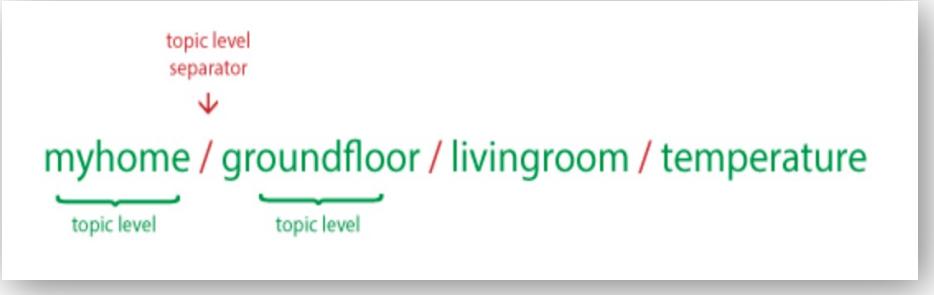
- Both, TCP & websockets can work on top of "Transport Layer Security (TLS)" (and its predecessor, Secure Sockets Layer (SSL))

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

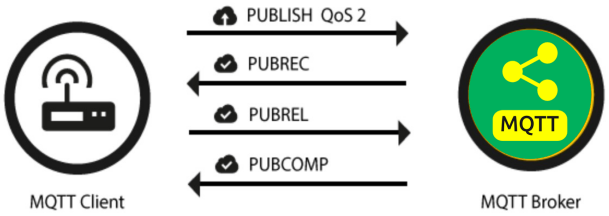
- $SYS/broker/clients/connected
- $SYS/broker/clients/disconnected
- $SYS/broker/clients/total
- $SYS/broker/messages/sent
- $SYS/broker/uptime

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.

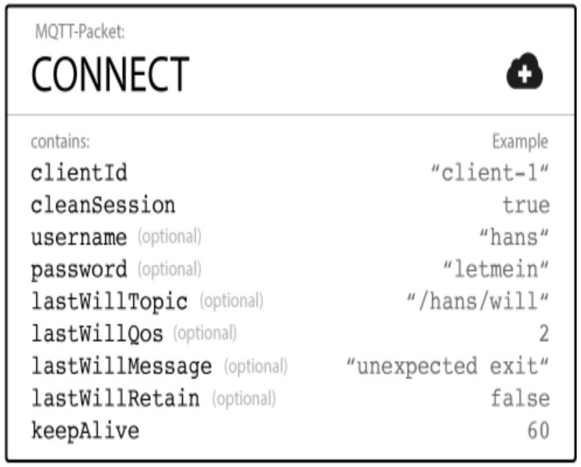
- Retained messages can help newly subscribed clients to get a status update immediately after subscribing to a topic and don't have to wait until a publishing clients send the next update.
  - In other words a retained message on a topic is the last known good value, because it doesn't have to be the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

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?

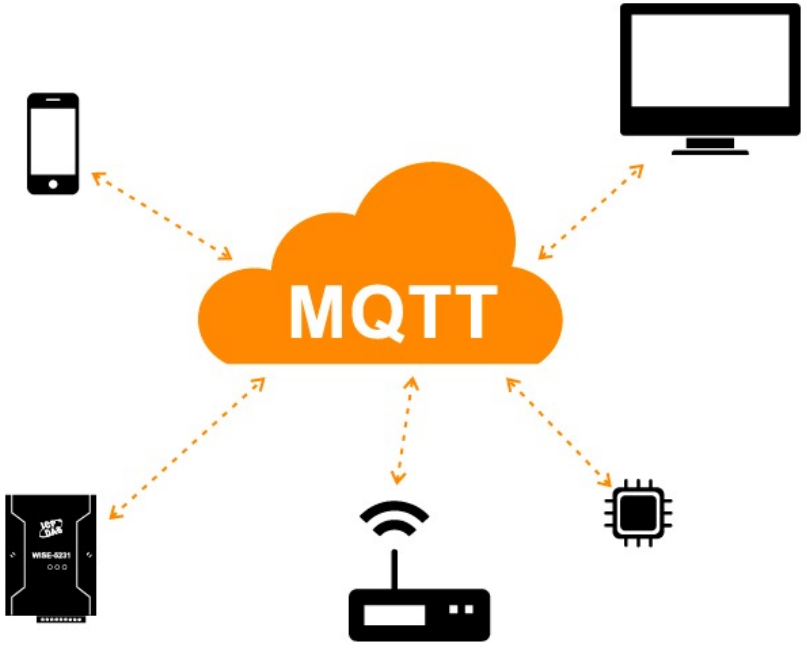
A few words on security

- 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.

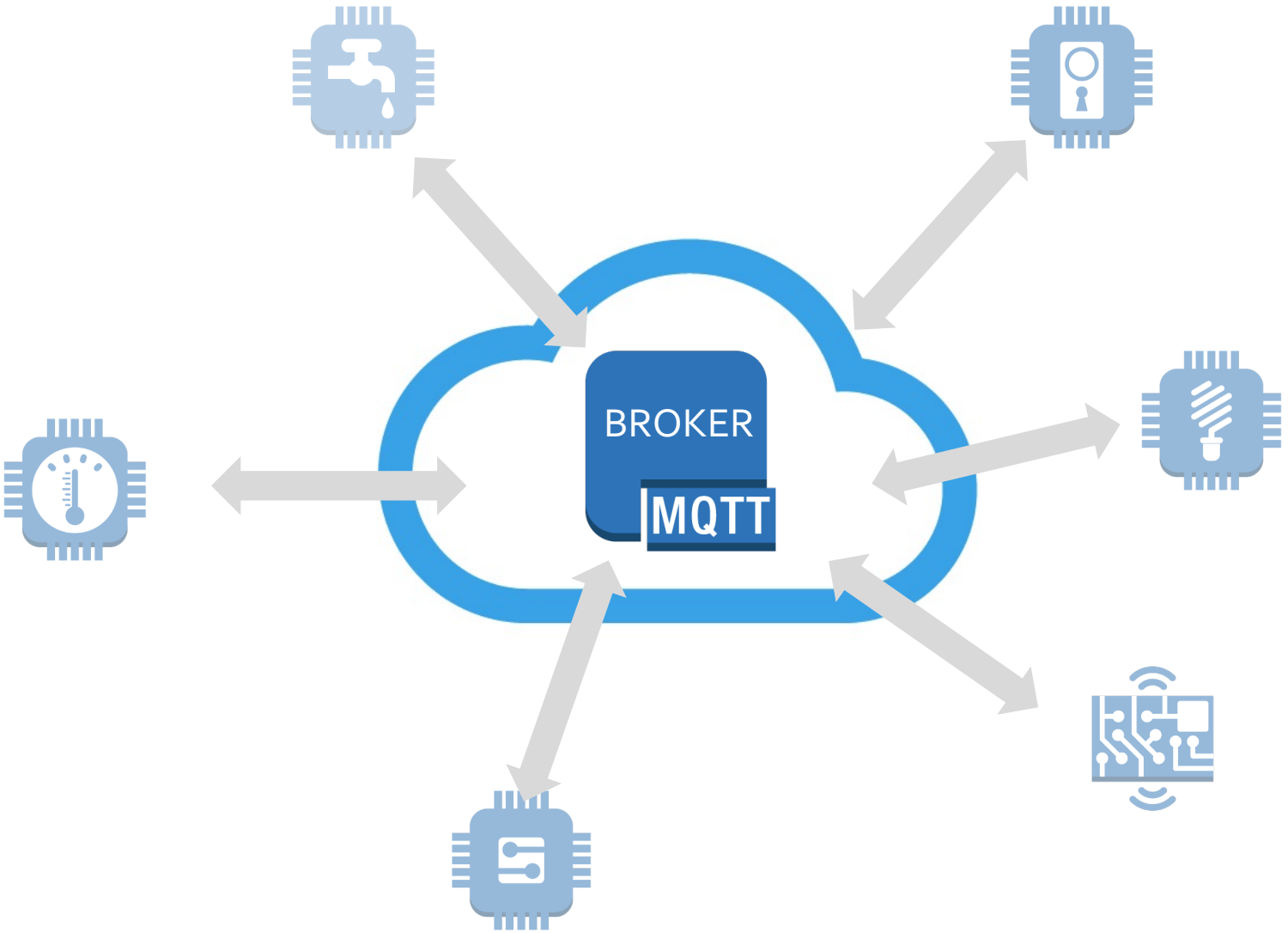
"It's not just you. We're all insecure in one way or another."

Intro to MQTT

- ✓ Brokers and clients



Creating a broker



Available MQTT brokers

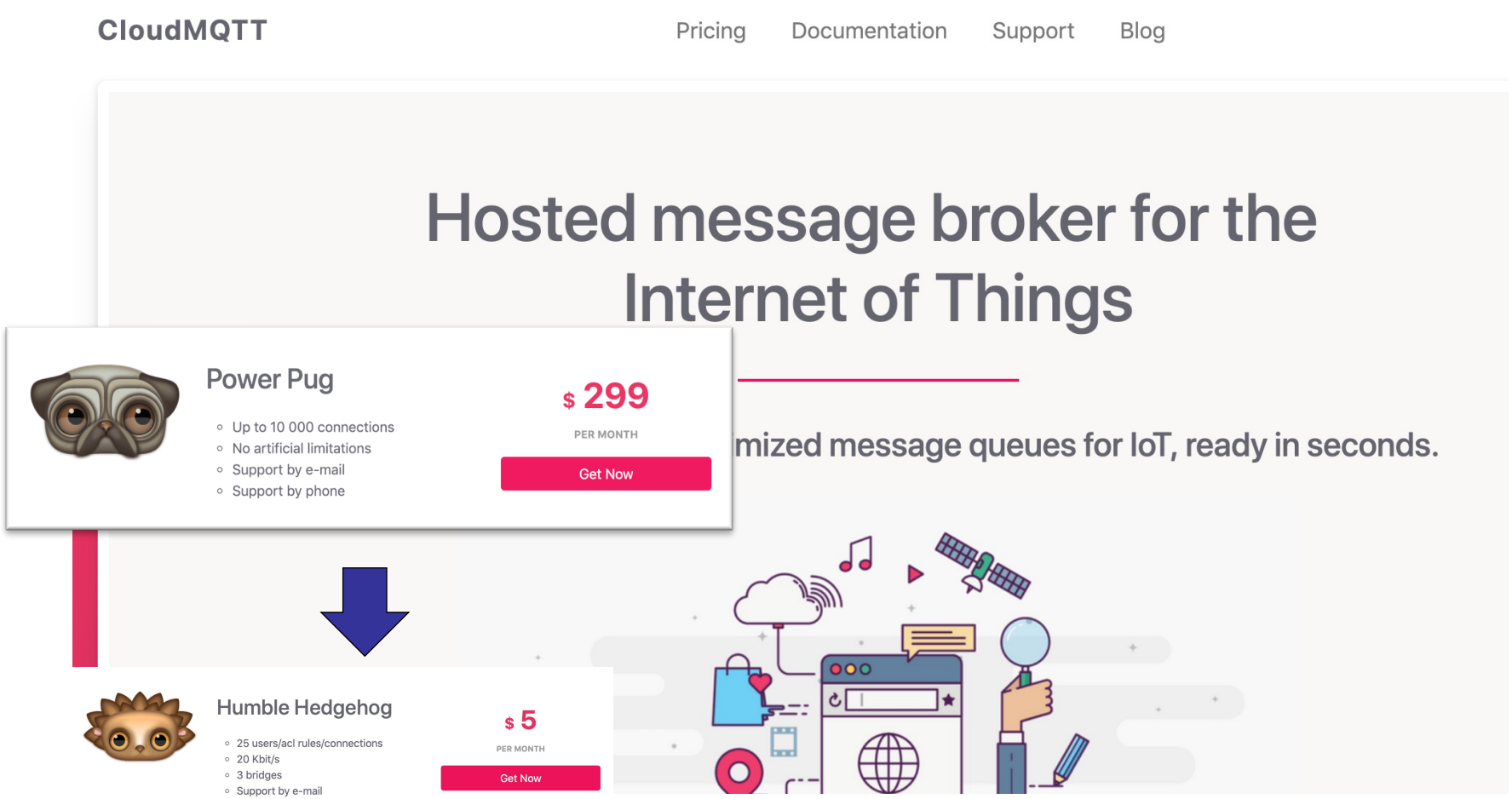
- The most widely used are:
  - http://mosquitto.org/
    - man page: https://mosquitto.org/man/mosquitto-8.html
  - http://www.hivemq.com/
    - The standard trial version only supports 25 connections.

- 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

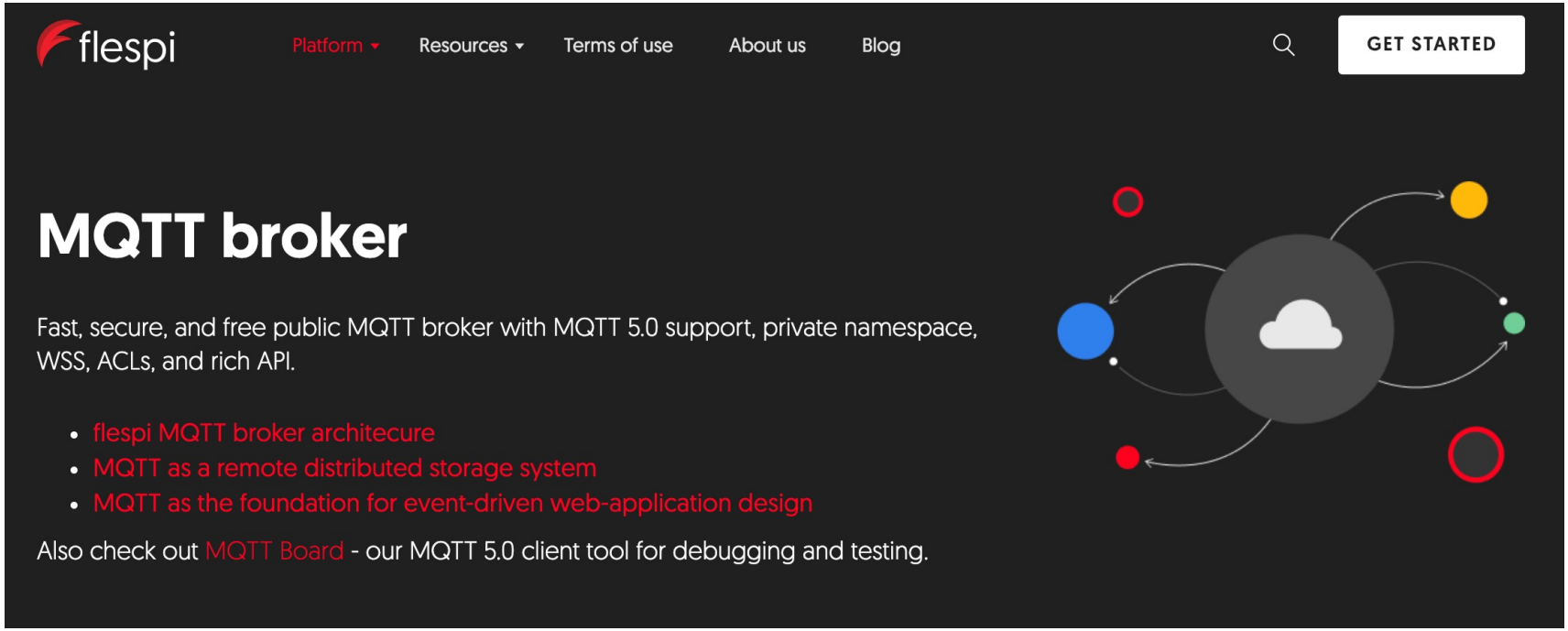
Cloud based MQTT brokers: CloudMQTT

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



Cloud based brokers: flespi

https://flespi.com/mqtt-broker



Cloud based brokers: flespi

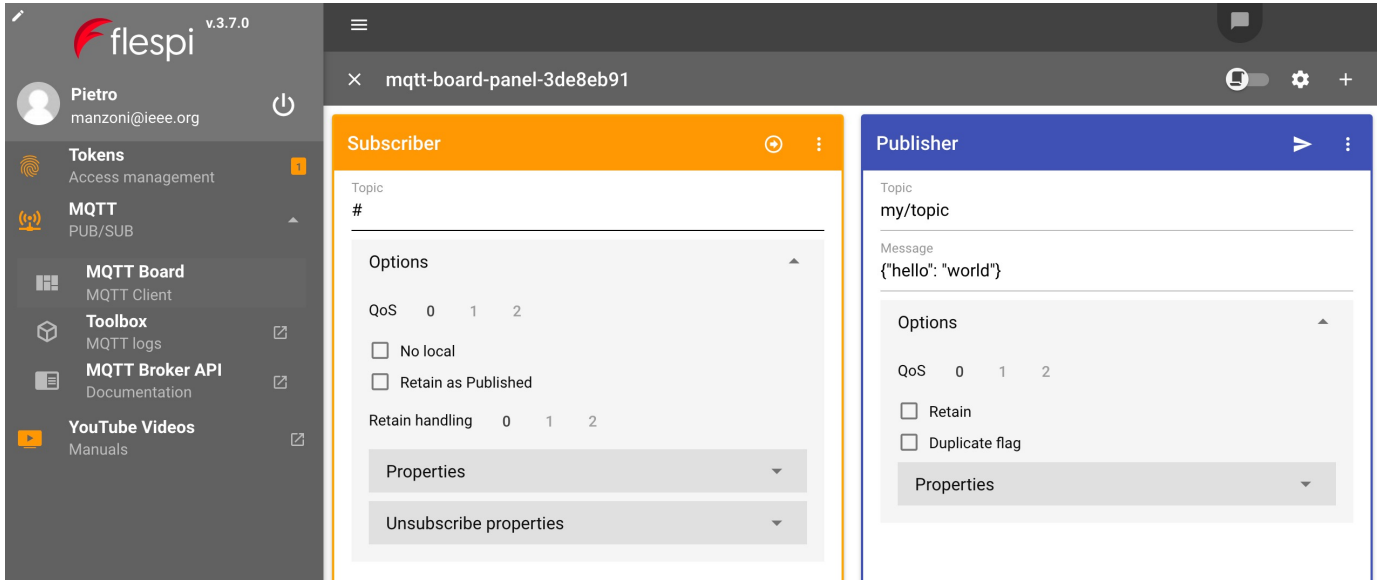
Terms of use

Free $0/mo

MQTT

100 active MQTT sessions

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



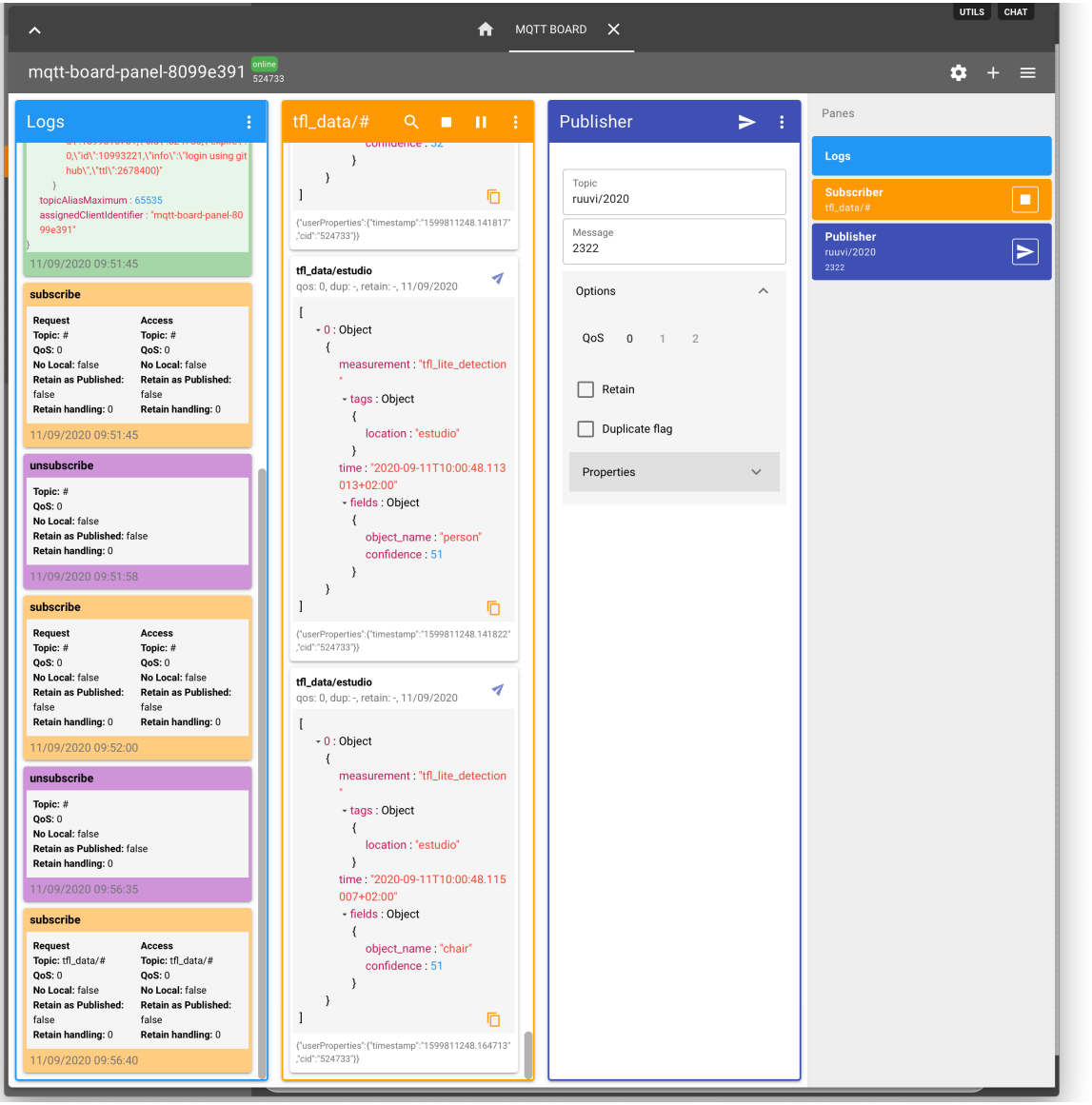
https://flespi.com/mqtt-api

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.

```
I1RKMMIUJpplQoSgAQ8MvDUJWNNJ9R2HIJgijO1S1gt5rajaeIOaiaKWwlHt2z1z
```

Flespi dashboard



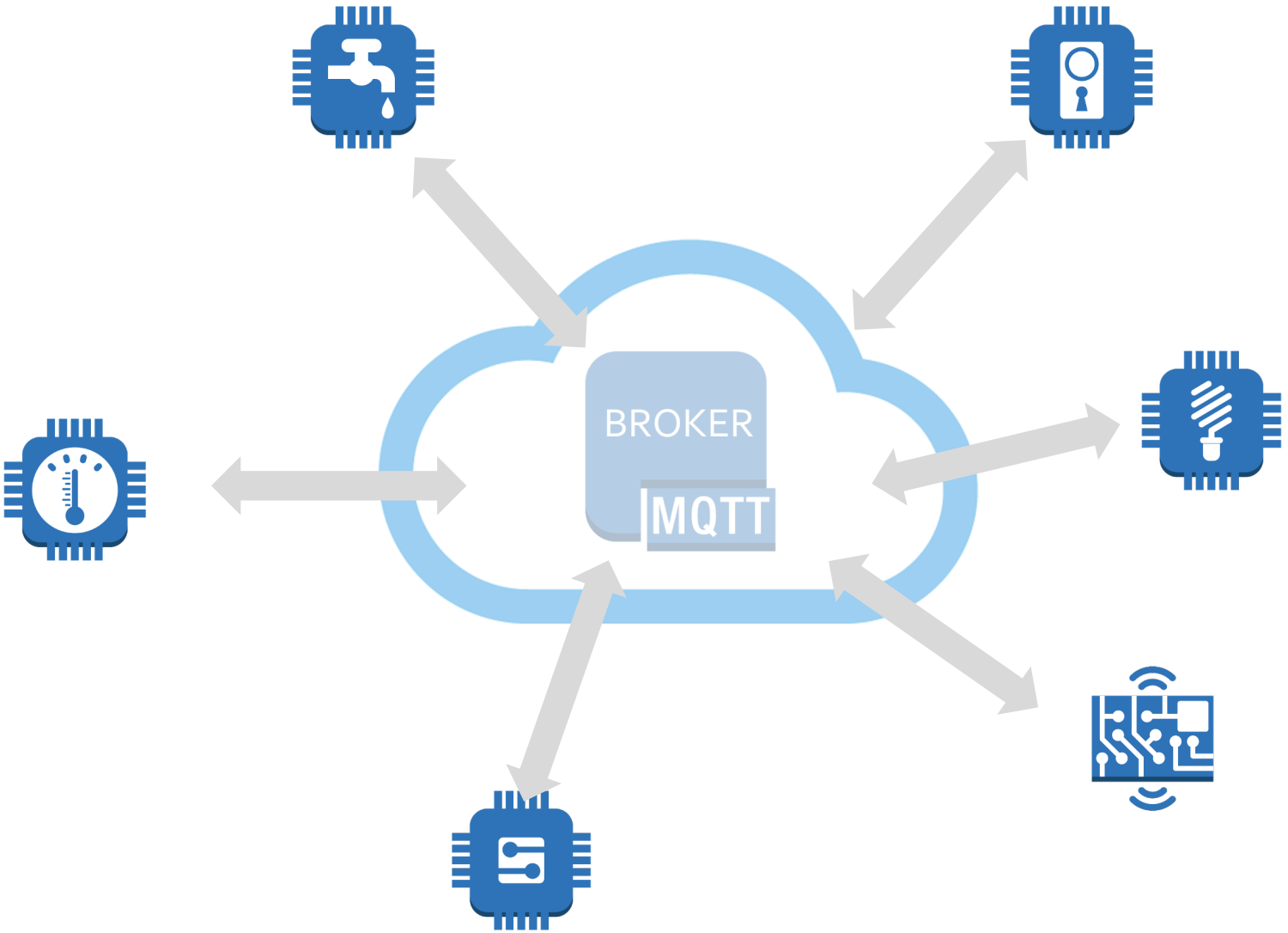
Open brokers ("Sandboxes")

- TCP based:
  - https://iot.eclipse.org/projects/sandboxes/
    - Hostname: **mqtt.eclipseprojects.io**
  - http://test.mosquitto.org/
    - Hostname: **test.mosquitto.org**
  - https://www.hivemq.com/public-mqtt-broker/
    - Hostname: **broker.hivemq.com**
    - dashboard: 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



MQTT clients: iOS

MQTTool 4+

Brent Petit

Designed for iPad

4.6 • 23 Ratings

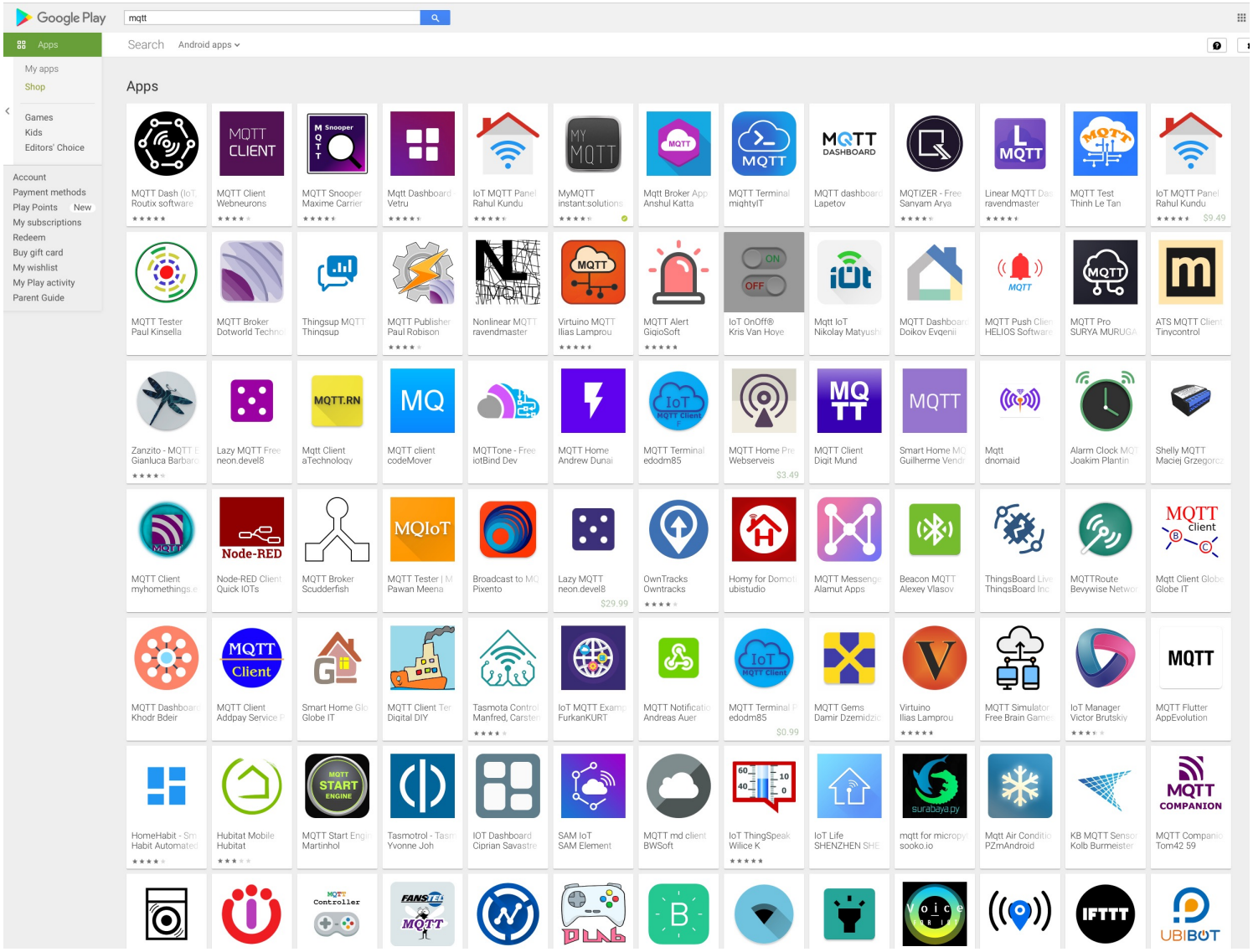
Free

View in Mac App Store ↗

ICPDAS MQ...

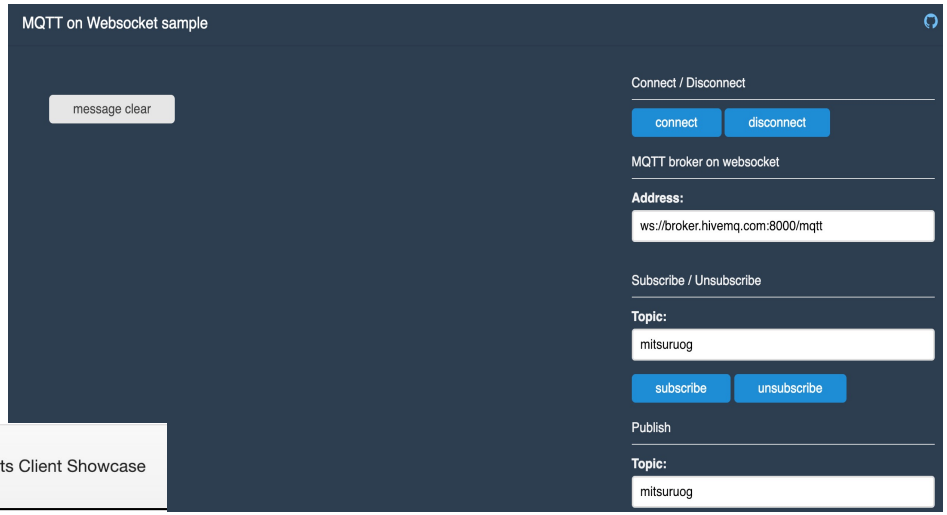
Utilidades

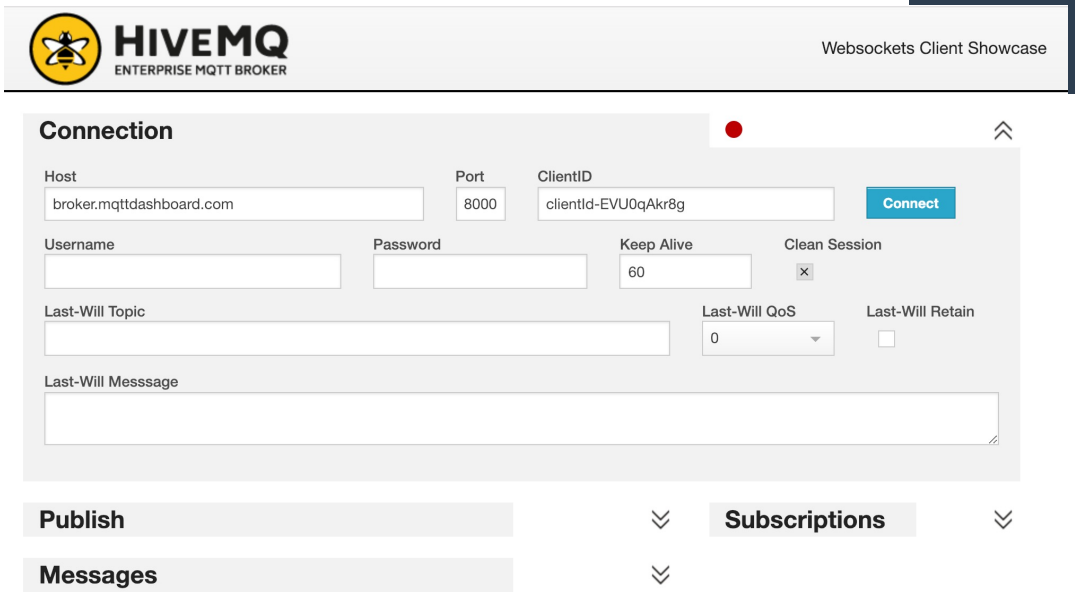
MQTT clients: Android



MQTT websocket clients

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





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

http://mqtt-explorer.com

Python example 1: the simplest subscriber

```
# File: sisub.py

import paho.mqtt.client as mqtt

THE_BROKER = "test.mosquitto.org"
THE_TOPIC = "$SYS/#"
C_ID = ""

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
  print("Connected to ", client._host, "port: ", client._port)
  print("Flags: ", flags, "return code: ", rc)

  # Subscribing in on_connect() means that if we lose the connection and
  # reconnect then subscriptions will be renewed.
  client.subscribe(THE_TOPIC, qos=0)

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
  print(msg.topic+" "+str(msg.payload))

client = mqtt.Client(client_id=C_ID, clean_session=True, userdata=None, protocol=mqtt.MQTTv311, transport="tcp")

client.on_connect = on_connect
client.on_message = on_message

client.username_pw_set(None, password=None)
client.connect(THE_BROKER, port=1883, keepalive=60)

client.loop_forever()
```

Python example 1: output

paho-code:pietro$ python example1.py

('Connected to ', 'test.mosquitto.org', 'port: ', 1883)

('Flags: ', {'session present': 0}, 'return code: ', 0)

$SYS/broker/connection/ks.ral.me.rnic/state 0

$SYS/broker/connection/Salem2.Public_Bridge/state 1

$SYS/broker/connection/RPi_MQTT_GESRV.bridgeTestMosquittoOrg/state 1

$SYS/broker/connection/br-john-jane/state 1

$SYS/broker/connection/cell_controller.bridge-01/state 1

$SYS/broker/connection/OpenWrt1504793280.test-mosquitto-org/state 1

$SYS/broker/connection/(none).test/state 0

$SYS/broker/connection/jrojoo-All-Series.test-mqtt-org/state 0

$SYS/broker/connection/archer.hive-archer/state 1

$SYS/broker/connection/MD-FelipeCoutto.test_mosquitto/state 1

$SYS/broker/connection/raspberrypi.snr-mqtt-bridge/state 0

$SYS/broker/connection/LAPTOP-TCMO862P.bridge-test-GSR01/state 0

…

Python example 2: very basic periodic producer

```
# File: sipub.py

import random
import time

import paho.mqtt.client as mqtt

THE_BROKER = "test.mosquitto.org"
THE_TOPIC = "PMtest/rndvalue"
CLIENT_ID = ""

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
  print("Connected to ", client._host, "port: ", client._port)
  print("Flags: ", flags, "returned code: ", rc)

# The callback for when a message is published.
def on_publish(client, userdata, mid):
  print("sipub: msg published (mid={})".format(mid))


client = mqtt.Client(client_id=CLIENT_ID,
            clean_session=True,
            userdata=None,
            protocol=mqtt.MQTTv311,
            transport="tcp")

client.on_connect = on_connect
client.on_publish = on_publish

client.username_pw_set(None, password=None)
client.connect(THE_BROKER, port=1883, keepalive=60)
```

```
client.loop_start()

while True:

    msg_to_be_sent = random.randint(0, 100)
    client.publish(THE_TOPIC,
                   payload=msg_to_be_sent,
                   qos=0,
                   retain=False)

    time.sleep(5)

client.loop_stop()
```

Generates a new data every 5 secs

Python example 2: very basic periodic producer

- Output obtained with a modified version of Example1.

```
('Connected to ', 'test.mosquitto.org', 'port: ', 1883)
('Flags: ', {'session present': 0}, 'return code: ', 0)
PMtest/rndvalue 56
PMtest/rndvalue 25
PMtest/rndvalue 43
PMtest/rndvalue 67
PMtest/rndvalue 0
PMtest/rndvalue 44
…
```

- **Which parts of that code had to be modified?**

```
# File: sisub.py

import paho.mqtt.client as mqtt

THE_BROKER = "test.mosquitto.org"
THE_TOPIC = "$SYS/#"
C_ID = ""

# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected to ", client._host, "port: ", client._port)
    print("Flags: ", flags, "return code: ", rc)

    # Subscribing in on_connect() means that if we lose the connection and
    # reconnect then subscriptions will be renewed.
    client.subscribe(THE_TOPIC, qos=0)

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))

client = mqtt.Client(client_id=C_ID, clean_session=True, userdata=None,
protocol=mqtt.MQTTv311, transport="tcp")

client.on_connect = on_connect
client.on_message = on_message

client.username_pw_set(None, password=None)
client.connect(THE_BROKER, port=1883, keepalive=60)

client.loop_forever()
```