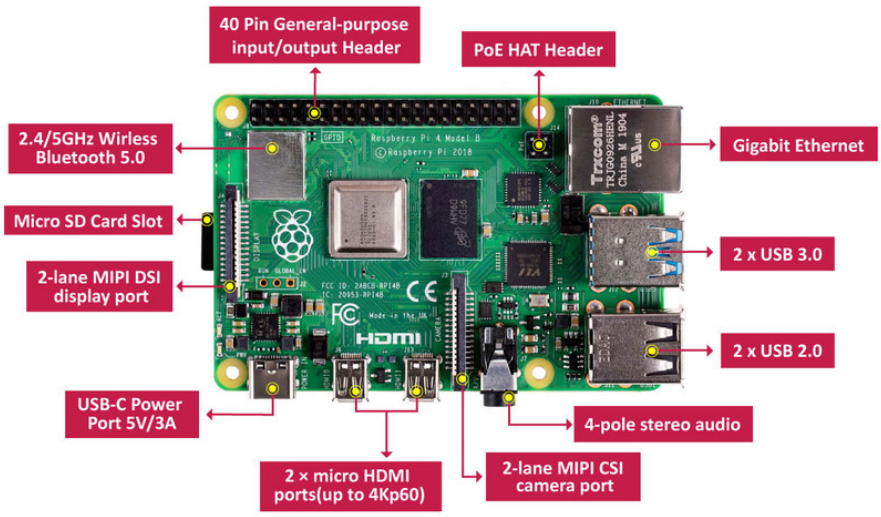
Raspberry Pi



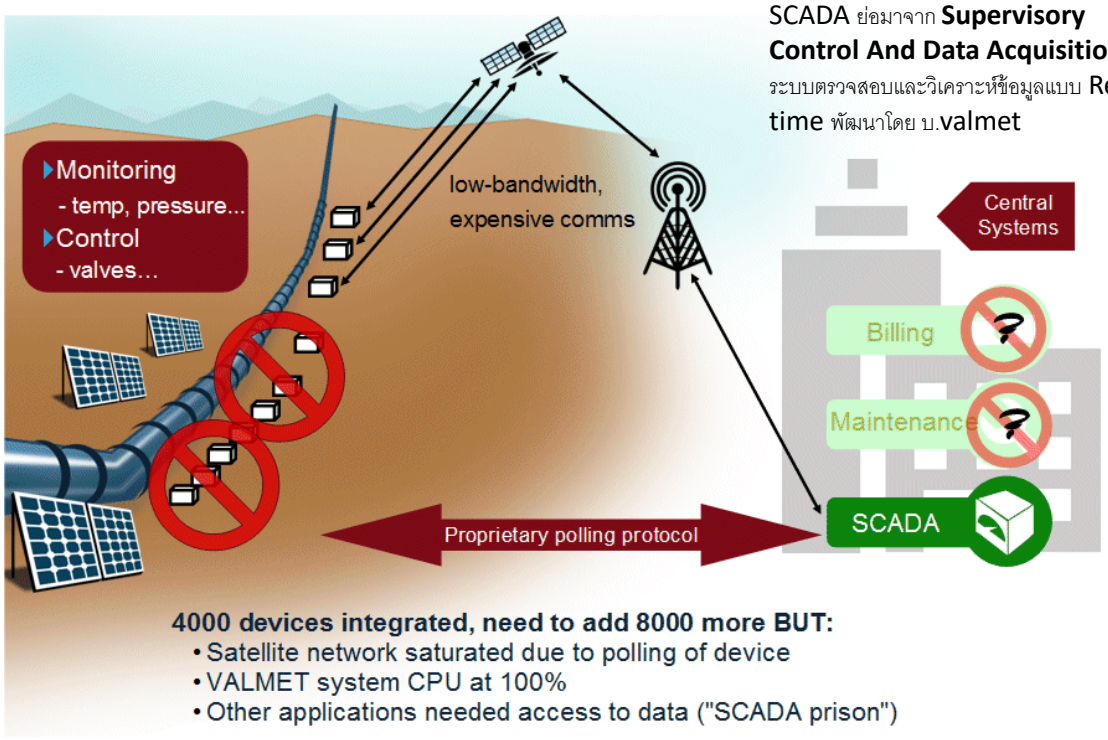
MQTT Protocol

- Invented by Andy Stanford Clark (IBM) and Arlen Nipper (Eurotech) in 1999
- Originally envisioned for use over Satellite link from an oil pipe line

Use case:

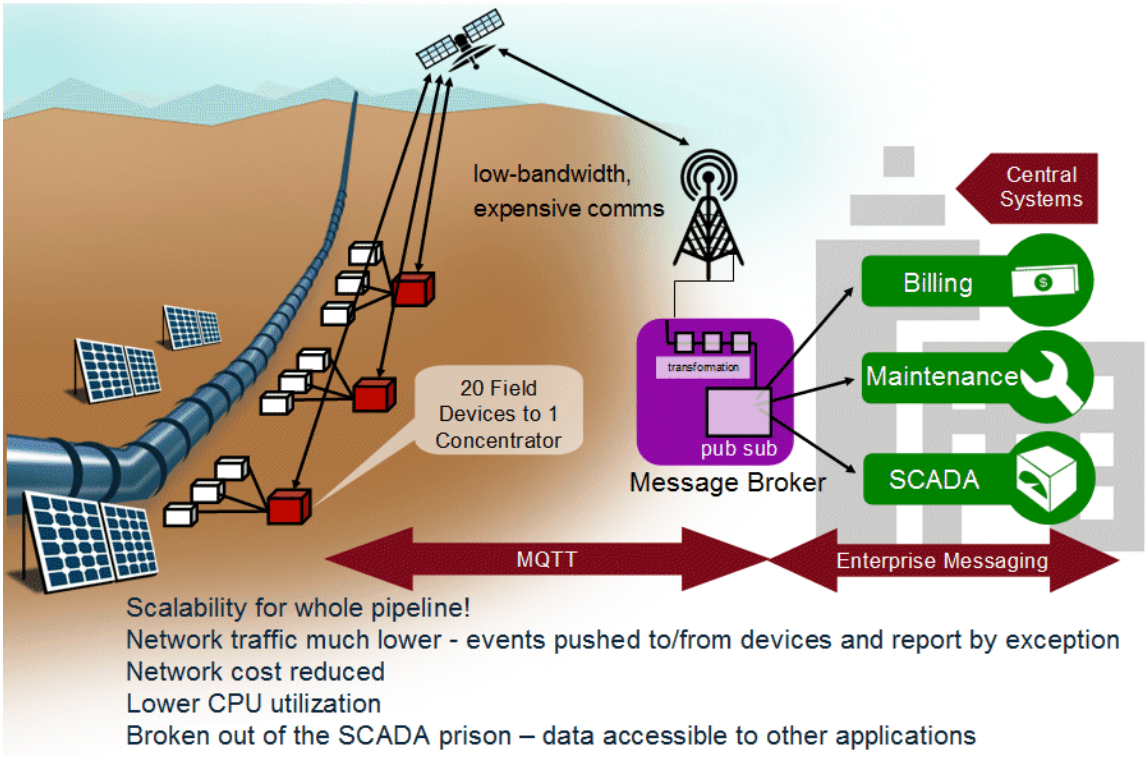
- 30.000 devices
- 17.000 km pipeline
- Remote monitoring
- Remote control
- Uses satellite links
- Bandwidth is **very** expensive

MQTT Protocol



SCADA ย่อมาจาก **Supervisory Control And Data Acquisition** เป็นระบบตรวจสอบและวิเคราะห์ข้อมูลแบบ Real-time พัฒนาโดย บ.valmet

MQTT Protocol



MQTT Protocol

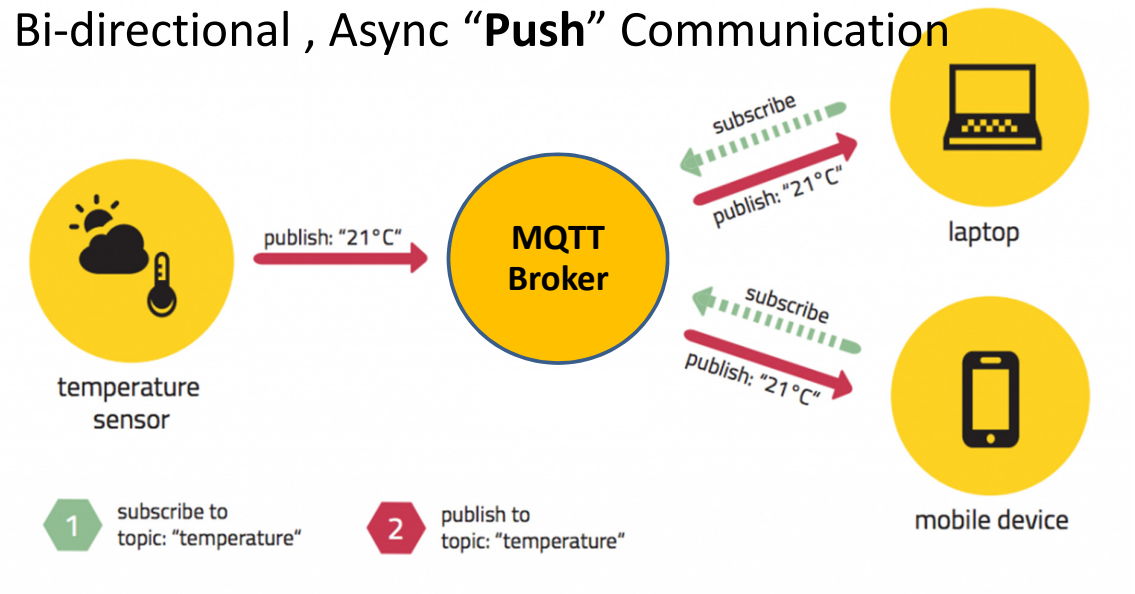
MQTT a lightweight protocol for IoT messaging

- **open** — open spec, standard — 40+ client implementations
- **lightweight** — minimal overhead — efficient format — tiny clients (kb)
- **reliable** — QoS for reliability on unreliable networks
- **simple** — 43-page spec — connect + publish + subscribe

Invented	Published	Eclipse M2M	Standard
EUROTECH IBM	MQTT	paho	OASIS
Late 1990s	Aug 2010	Nov 2011	Sep 2014

The publish/subscribe pattern

Bi-directional , Async "**Push**" Communication



MQTT Protocol

MQTT

simple to implement

Connect

Subscribe

Publish

Unsubscribe

Disconnect

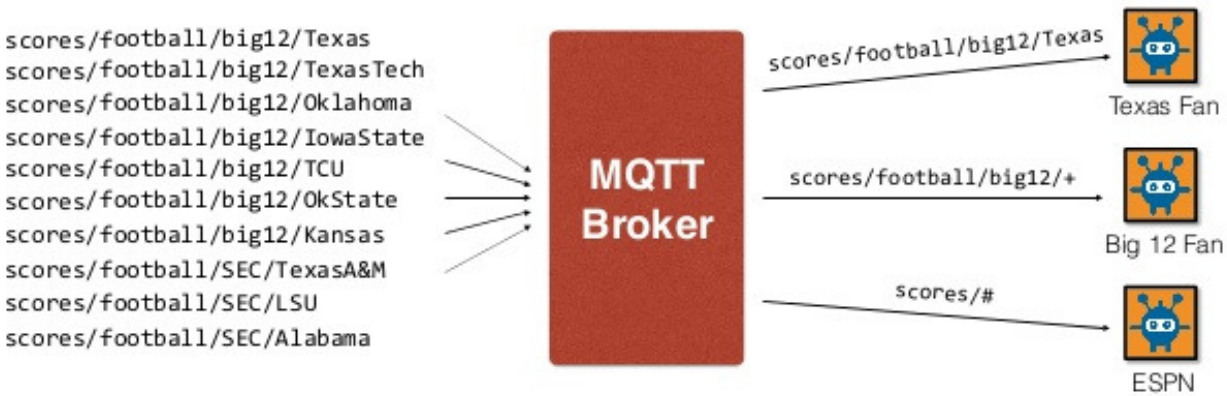
```
client = new Messaging.Client(hostname, port, clientId)
client.onMessageArrived = messageArrived;
client.onConnectionLost = connectionLost;
client.connect({ onSuccess: connectionSuccess });

function connectionSuccess() {
     client.subscribe("planets/earth");
     var msg = new Messaging.Message("Hello world!");
     msg.destinationName = "planets/earth";
     client.publish(msg);
}

function messageArrived(msg) {
     console.log(msg.payloadString);
     client.unsubscribe("planets/earth");
     client.disconnect();
}
```

Eclipse Paho JavaScript MQTT client

Subscribe

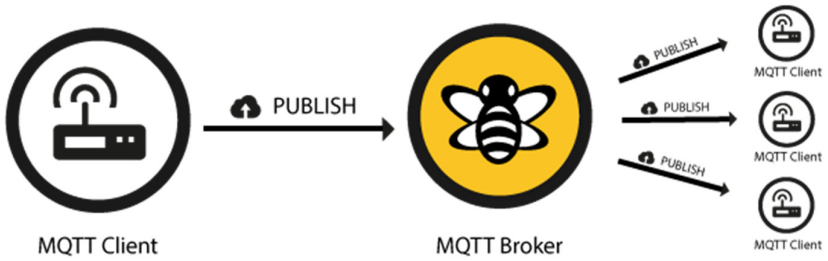
MQTT

allows wildcard subscriptions



single level wildcard: **+** multi-level wildcard: **#**

Publish



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

Publish

Topics
A topic is a UTF-8 string, which is used by the broker to filter messages for each connected client. A topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

QoS
A Quality of Service Level (QoS) for this message. The level (0,1 or 2) determines the guarantee of a message reaching the other end

Retain-Flag
This flag determines if the message will be saved by the broker for the specified topic as last known good value. New clients that subscribe to that topic will receive the last retained message on that topic instantly after subscribing. (ให้ Broker เก็บ Payload ล่าสุด ที่ส่งขึ้นไปไว้ด้วย ถ้ามีคนใหม่เข้ามา Subscribe ทีหลัง ก็จะได้รับข้อมูลนี้ด้วย)

Payload
This is the actual content of the message.

DUP flag
The duplicate flag indicates, that this message is a duplicate and is resent because the other end didn't acknowledge the original message. This is only relevant for QoS greater than 0

Packet Identifier
The packet identifier is a unique identifier between client and broker to identify a message in a message flow. This is only relevant for QoS greater than zero.

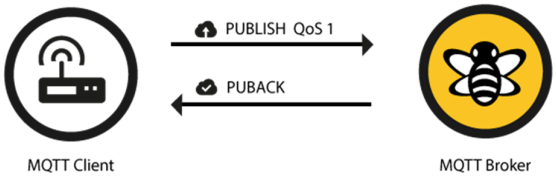
QoS

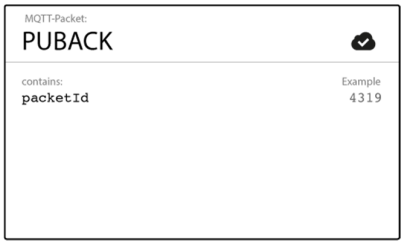
- **QoS 0 – at most once**

  The minimal level is zero and it guarantees a best effort delivery. A message won't be acknowledged by the receiver or stored and redelivered by the sender.

- **QoS 1 – at least once**

  it is guaranteed that a message will be delivered at least once to the receiver. The sender will store the message until it gets an acknowledgement in form of a PUBACK command message from the receiver.

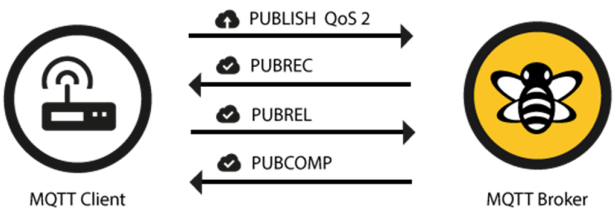




QoS

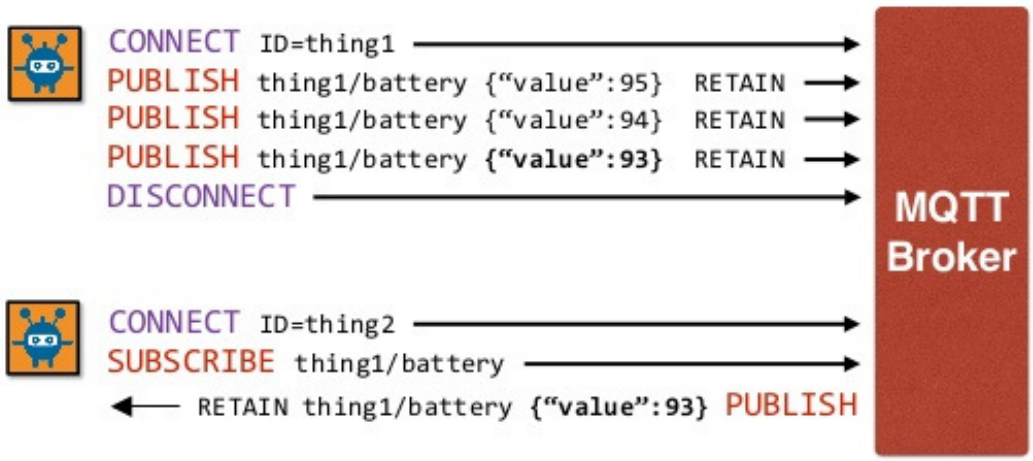
- **QoS 2 – Exactly once**

  The highest QoS is 2, it guarantees that each message is received only once by the counterpart. It is the safest and also the slowest quality of service level. The guarantee is provided by two flows there and back between sender and receiver.



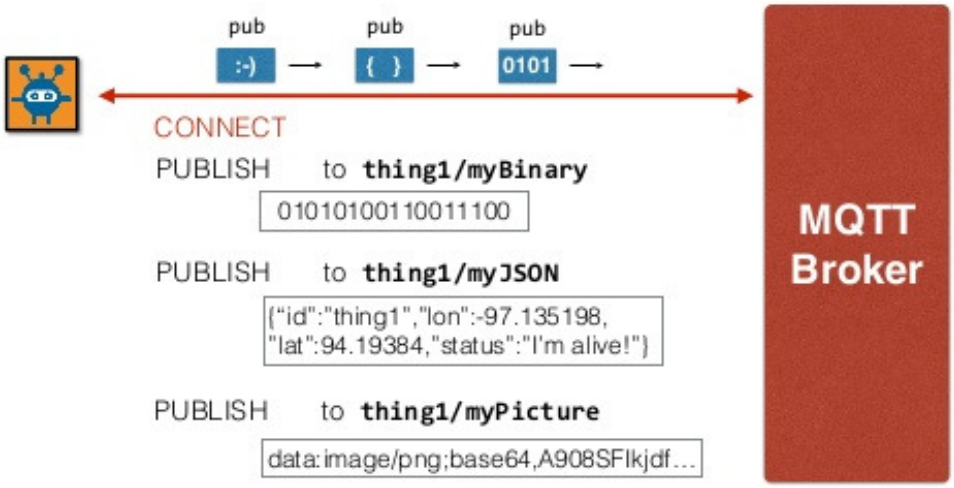
Retain messages

MQTT retained messages for last value caching

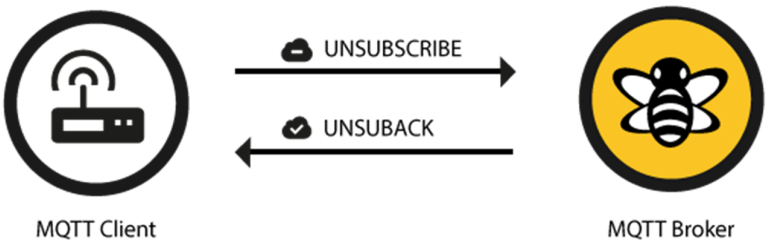


Payload

MQTT agnostic payload for flexible delivery



Unsubscribe



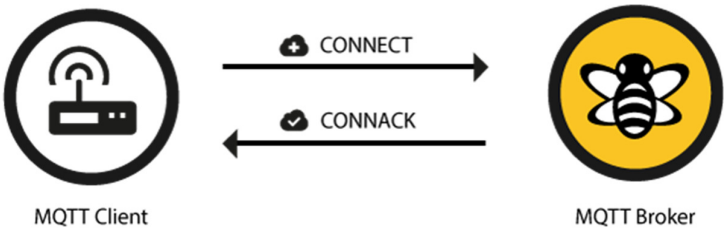
MQTT-Packet:
UNSUBSCRIBE

contains:	Example
`packetId`	`4315`
`topic1` } (list of topics)	`"topic/1"`
`topic2`	`"topic/2"`
`...`	`...`

MQTT-Packet:
UNSUBACK

contains:	Example
`packetId`	`4316`

Connect & Disconnect



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"`
`keepAlive`	`60`

MQTT-Packet:
CONNACK

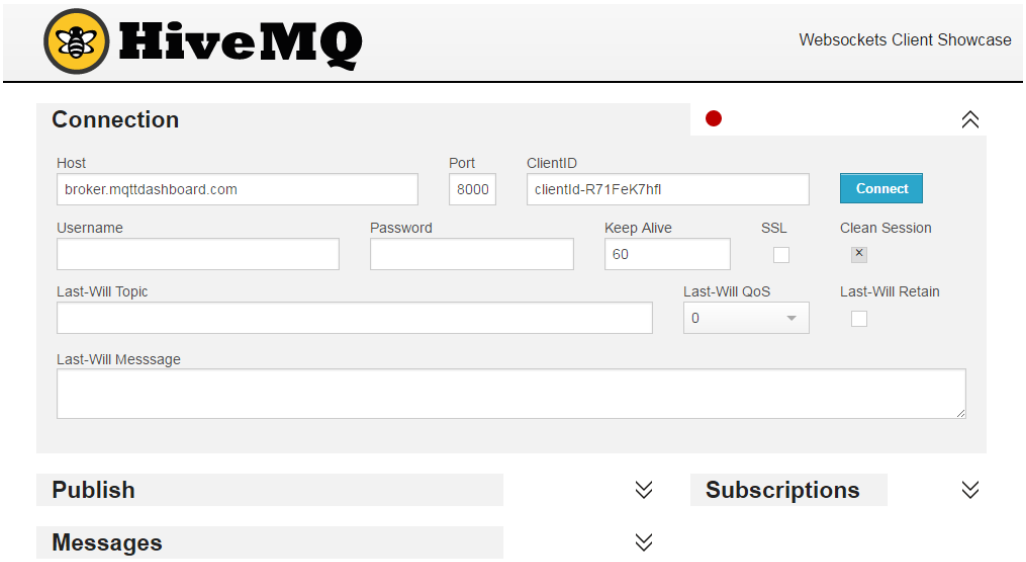
contains:	Example
`sessionPresent`	`true`
`returnCode`	`0`

MQTT-Packet:
DISCONNECT

no payload

การทดลองที่ 1. Public MQTT Broker

- เข้า Website ที่ URL
  http://www.hivemq.com/demos/websocket-client/



การติดตั้ง **Library** สำหรับภาษา **Python** ให้ใช้งานโปรโตคอล **MQTT** ได้

พิมพ์คำสั่งในการติดตั้งดังนี้

```
$sudo apt-get update
$sudo apt-get install python-pip
$pip install paho-mqtt
```

pip is a package management system used to install and manage software packages written in Python.

pip is a recursive acronym that can stand for either "Pip Installs Packages" or "Pip Installs Python"

ตัวอย่าง MQTT Publish โดยภาษา python

- พิมพ์คำสั่ง **$nano pub.py**
- เขียนโปรแกรม python ดังนี้

```
import  paho.mqtt.client as mqtt
import  time

mqttc = mqtt.Client()
mqttc.connect("broker.mqttdashboard.com", 1883)

while True:
        mqttc.publish("test/pub", "Hello")
        time.sleep(2)
```

- Run โปรแกรมโดยใช้คำสั่ง **$python pub.py**

ตัวอย่าง MQTT Subscribe โดยภาษา python

- พิพม์คำสั่ง **$nano sub.py**
- เขียนโปรแกรม python ดังนี้

```
import  paho.mqtt.client as mqtt

def on_connect(client, userdata, flags_dict, rc):
        print("Connected with result code "+str(rc))
        client.subscribe("test/sub")

def on_message(client, userdata, msg):
        print(msg.topic+" "+str(msg.payload))

client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

client.connect("broker.mqttdashboard.com", 1883)
client.loop_forever()
```

- Run โปรแกรมโดยใช้คำสั่ง **$python sub.py**

แบบฝึกหัดที่ 1.

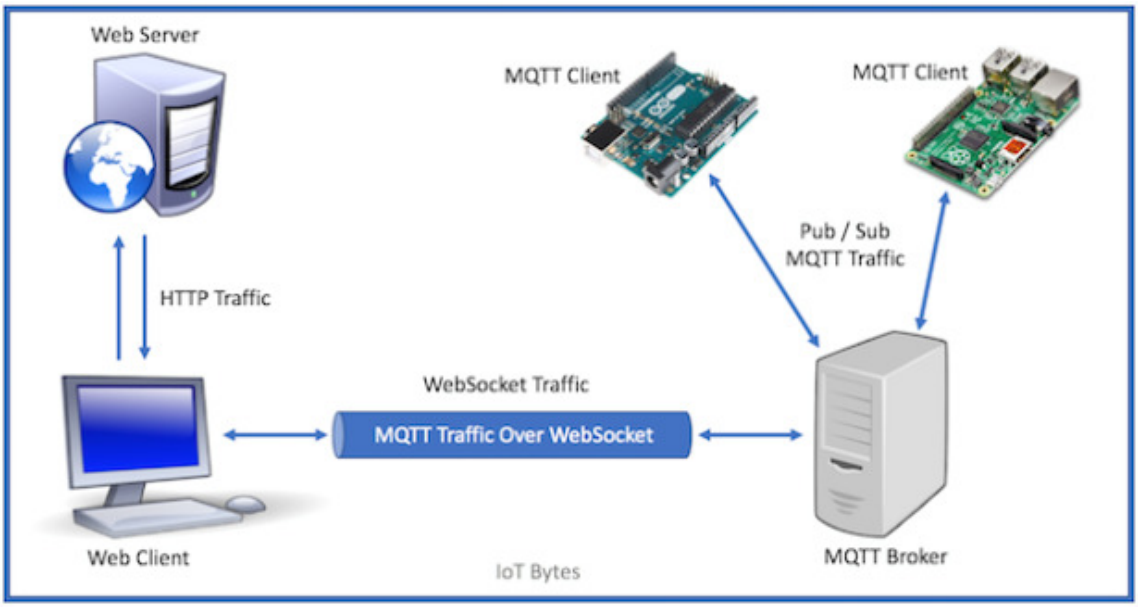
แบบฝึกหัด Raspberry Pi Publish ไปที่ Broker

- อ่านค่าจาก ปุ่มกด
- ถ้ากดปุ่ม ให้ Publish ไป Broker ว่า ON

แบบฝึกหัด Raspberry Pi ทำการ Subscribe จาก Broker

- รอรับค่าจาก Broker
- ถ้ารับได้คำว่า ON ให้ LED ติด
- ถ้ารับได้คำว่า OFF ให้ LED ดับ

Web socket

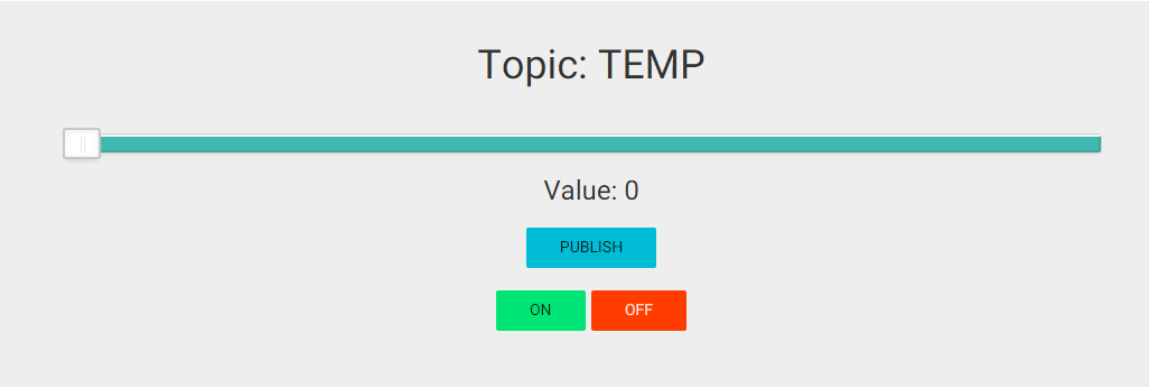


Control via Websocket

ไฟล์ที่เตรียมไว้ให้ web.rar ให้ทำการ unzip และเปิดไฟล์

control.html

ไฟล์ตัวอย่างที่ให้นี้ คือ ทำหน้าที่ Publish ไปยัง Topic ที่กำหนด
เมื่อกดปุ่ม ON จะ publish ค่า on ไปยัง Broker
เมื่อกดปุ่ม OFF จะ publish ค่า off ไปยัง Broker



แบบฝึกหัดที่ 2. Control via Websocket

1. แก้ไขไฟล์ control.html ดังนี้

```
var MQTTbroker = ‘ Hive MQ';        // broker
var MQTTport = 8000;                // port ของ Webseocket ที่เปิดไว้
var MQTTsubTopic = 'temp';             //แก้ชื่อ topic ได้ตามใจชอบ
```

2. ทำการ Upload ไฟล์ ไปยัง web hosting (หรือ run ที่ client)

3. เขียนโปรแกรม ที่ Raspi เพื่อ Subscribe ค่าที่ส่งมาจาก Webpage
   เมื่อกดปุ่ม ON ให้ LED ติด เมื่อกดปุ่ม OFF ให้ LED ดับ
   เมื่อเลื่อน Slide bar ให้ LED หรี่สว่างตามค่า

แบบฝึกหัดที่ 3. Realtime chart

1. แก้ไขไฟล์ mqttchart_realtime.html ดังนี้

```
var MQTTbroker = ‘ Hive MQ';        // broker
var MQTTport = 8000;                // port ของ Webseocket ที่เปิดไว้
var MQTTsubTopic = 'temp';            //แก้ชื่อ topic ได้ตามใจชอบ
```

2. ทำการ Upload ไฟล์ ไปยัง web hosting (หรือ run ที่ client)

3. เขียนโปรแกรม ที่ Raspi เพื่อ Publish ค่า ส่งไปแสดงผลที่ Webpage
   อ่านค่าจากตัวต้านทานปรับค่าได้ แล้ว publish ไปแสดงออกเป็นกราฟ ที่ webpage