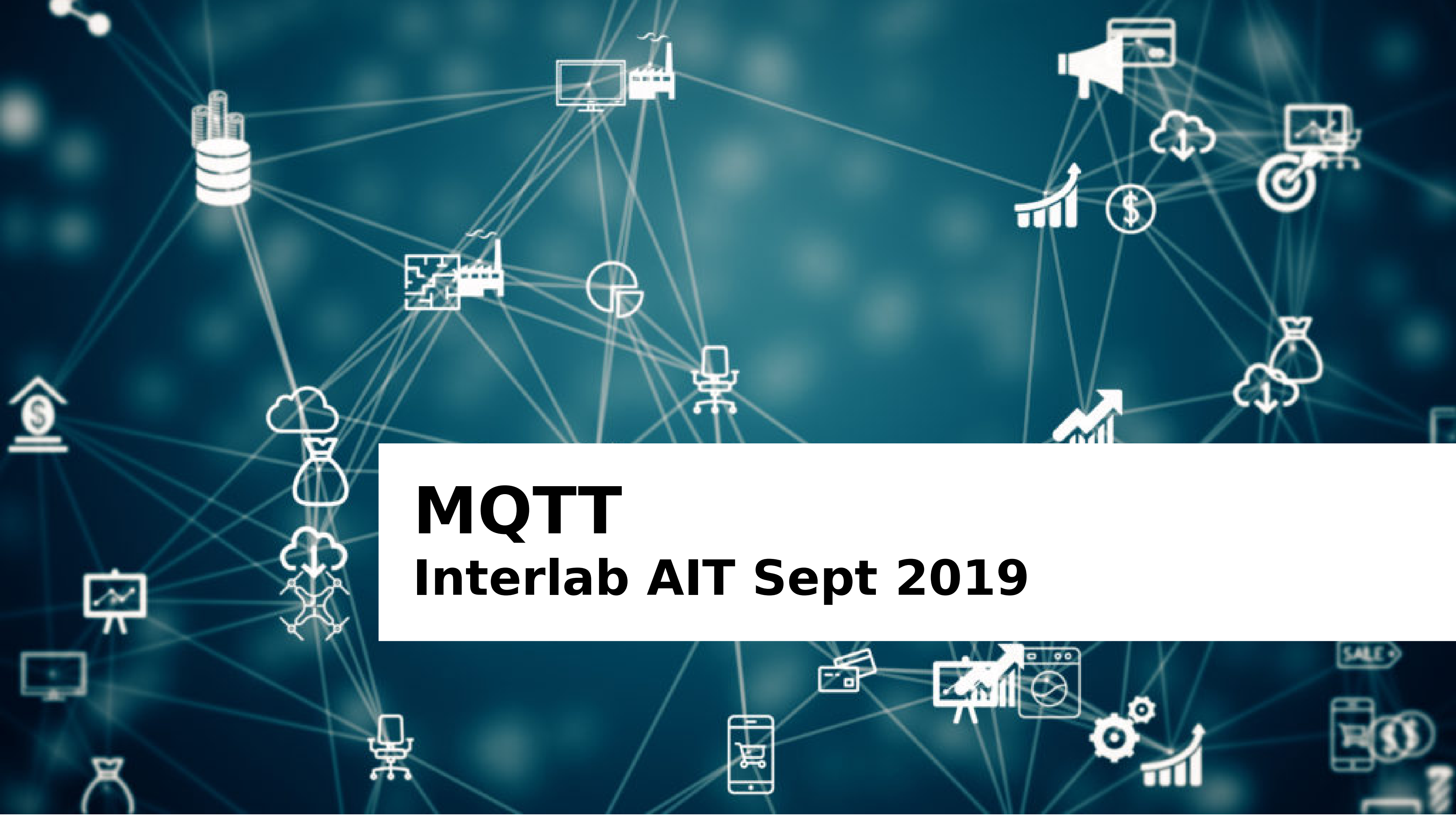
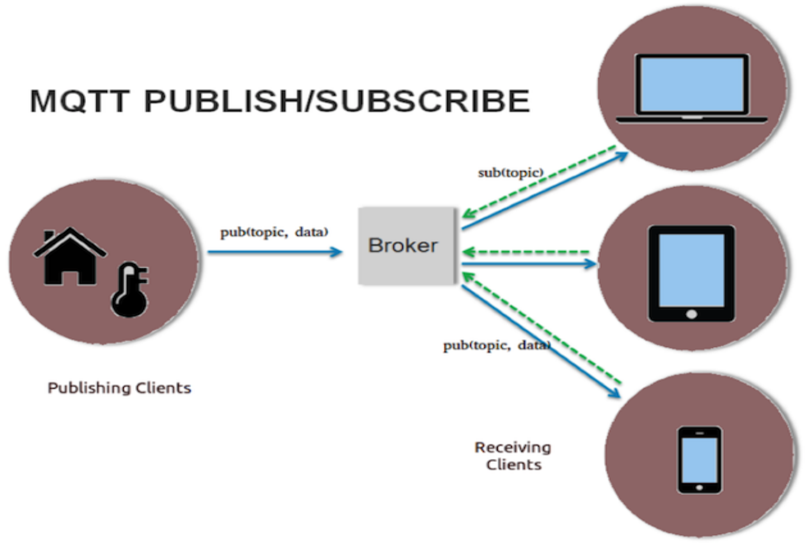


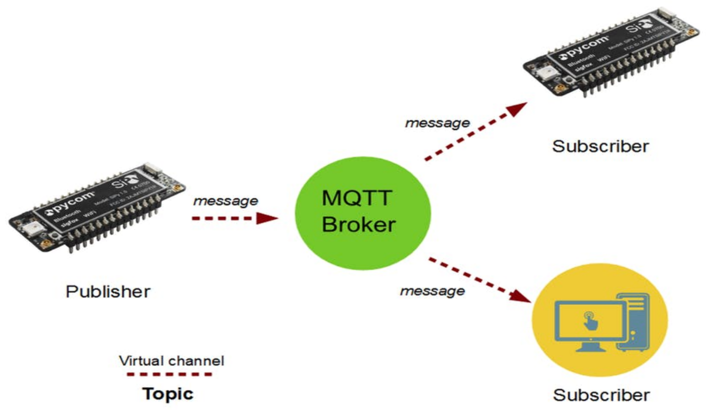
MQTT

Interlab AIT Sept 2019

MQTT / 1

MQTT (Message Queuing Telemetry Transport) is a **publish-subscribe**-based messaging protocol.
Works on top of **TCP/IP**.
A Message broker or server receives and redistributes





MQTT / 2 / topics

MQTT publishing and subscription is organized by topics:

e.g.
/house/light
Or
/greenhouse/temperature

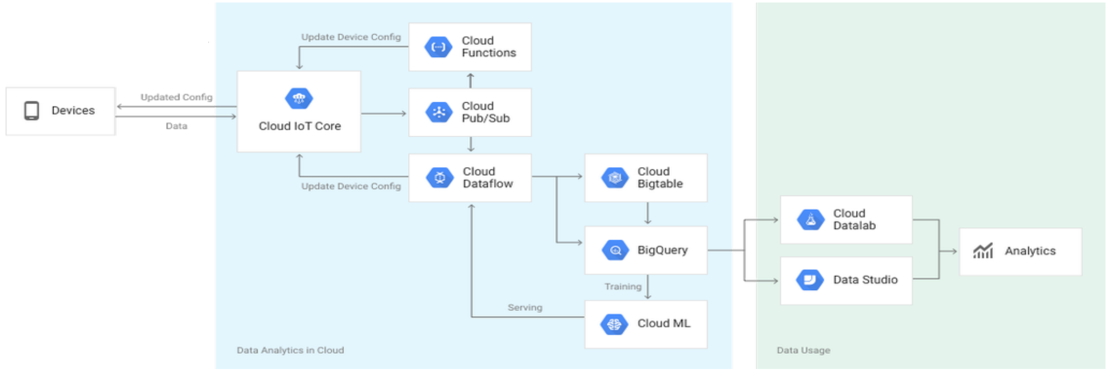
(case sensitive!)

Messages are in **free format,** however, often you will see e.g. json or xml messages.
Some service might restrict message formats.

MQTT / 3 / adoption

MQTT is a widely accepted and deployed standard in services/platforms such as

- Amazon AWS IoT
- Microsoft Azure
- Facebook Messenger (to an unknown degree)
- Google Cloud IoT Core is able to support data “from millions of globally dispersed devices.” Like similar services, Cloud IoT Core supports the standard MQTT and HTTP protocols for talking to devices.



Source: https://techcrunch.com/2018/02/21/googles-cloud-iot-core-is-now-generally-available

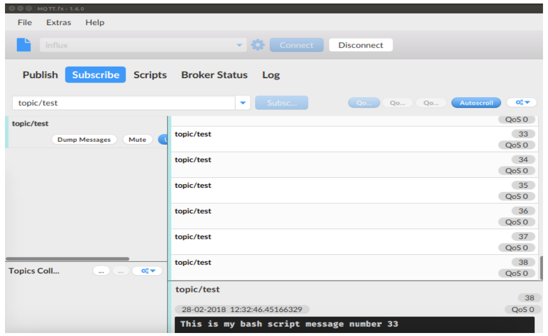
MQTT / 4 / usage

MQTT clients exist for a wide variety of platforms and languages:
https://github.com/mqtt/mqtt.github.io/wiki/software?id=software
https://www.hivemq.com/blog/seven-best-mqtt-client-tools

e.g
MQTT.fx
(available for Win/MacOSX/Linux, http://www.jensd.de/apps/mqttfx/, free)
mqtt-spy
(based on Java 8, http://kamilfb.github.io/mqtt-spy/ , OpenSource)
mosquitto_tools
(commandline, for Win/MacOSX/Linux, https://mosquitto.org/download/ , OpenSource)

And of course for

Arduino (C)
Raspberry (== Debian Linux)
Pycom (micropython)



Connection: influx
Subscribed topic: topic/test

```
topic/test                                  QoS 0
topic/test                                  33  QoS 0
topic/test                                  34  QoS 0
topic/test                                  35  QoS 0
topic/test                                  36  QoS 0
topic/test                                  37  QoS 0
topic/test                                  38  QoS 0

topic/test                                  38
28-02-2018 12:32:46.45166329                QoS 0
This is my bash script message number 33
```

MQTT / 5 / servers (aka brokers)

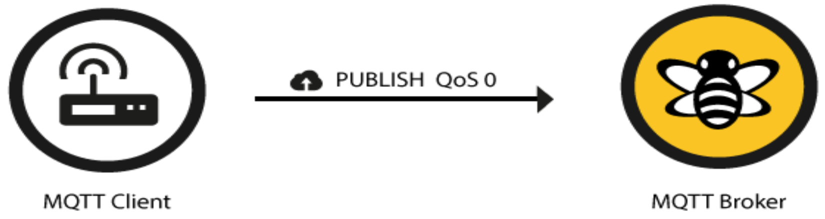
A wide choice of servers is available, many of them open source:
https://github.com/mqtt/mqtt.github.io/wiki/servers

influx.itu.dk runs a public mosquitto server.

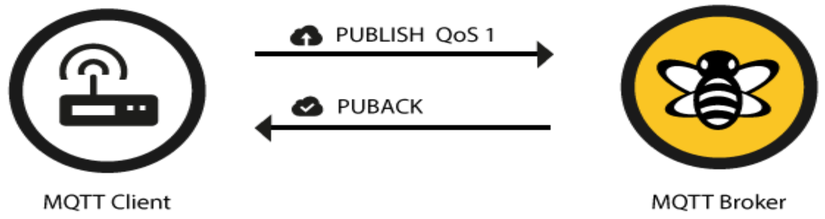
MQTT / 6 / QoS

MQTT implements 3 levels of QoS:

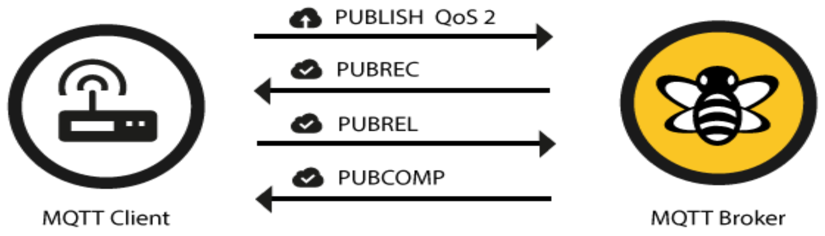
At most once (0)
Just send it



At least once (1)
Send and confirm



Exactly once (2)
Send, confirm, reference, stop.



MQTT / 7 / Last Will

MQTT Last Will and Testament

The Last Will and Testament (LWT) feature is used in MQTT to notify other clients about an ungracefully disconnected client. Each client can specify its last will message (a normal MQTT message with topic, retained flag, QoS and payload) when connecting to a broker. The broker will store the message until it detects that the client has disconnected ungracefully, and in tat case, send it out to subscribers.

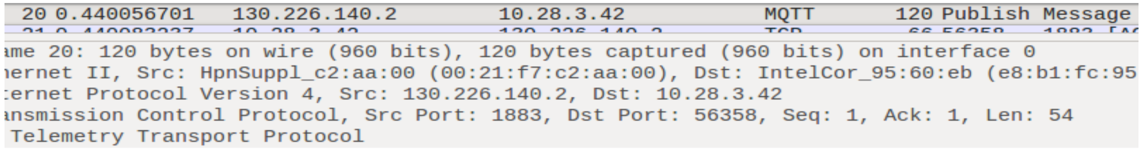
Why is this important?

In a lean communications protocol, dependent devices and services need to know about and have some chance to react to the disappearance of devices.

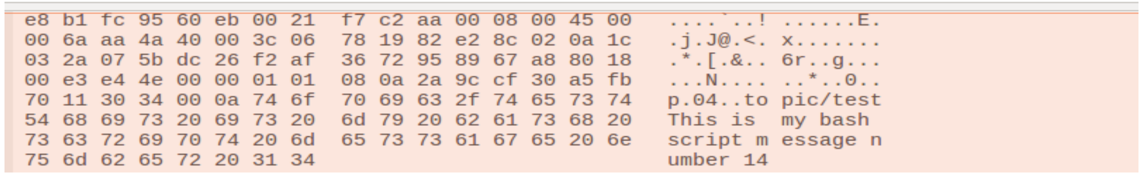
MQTT / 8 / security

MQTT may be used encrypted or unencrypted - here is mosquittos standard ports:

- 1883 : MQTT, unencrypted
- 8883 : MQTT, encrypted
- 8884 : MQTT, encrypted, client certificate required
- 8080 : MQTT over WebSockets, unencrypted
- 8081 : MQTT over WebSockets, encrypted



```
 20 0.440056701    130.226.140.2         10.28.3.42            MQTT        120 Publish Message
 21 0.440083227    10.28.3.42            130.226.140.2         TCP          66 56358 → 1883 [AC
ame 20: 120 bytes on wire (960 bits), 120 bytes captured (960 bits) on interface 0
hernet II, Src: HpnSuppl_c2:aa:00 (00:21:f7:c2:aa:00), Dst: IntelCor_95:60:eb (e8:b1:fc:95
ernet Protocol Version 4, Src: 130.226.140.2, Dst: 10.28.3.42
ansmission Control Protocol, Src Port: 1883, Dst Port: 56358, Seq: 1, Ack: 1, Len: 54
 Telemetry Transport Protocol
```



```
 e8 b1 fc 95 60 eb 00 21  f7 c2 aa 00 08 00 45 00   ....`..! ......E.
 00 6a aa 4a 40 00 3c 06  78 19 82 e2 8c 02 0a 1c   .j.J@.<. x.......
 03 2a 07 5b dc 26 f2 af  36 72 95 89 67 a8 80 18   .*.[.&.. 6r..g...
 00 e3 e4 4e 00 00 01 01  08 0a 2a 9c cf 30 a5 fb   ...N.... ..*..0..
 70 11 30 34 00 0a 74 6f  70 69 63 2f 74 65 73 74   p.04..to pic/test
 54 68 69 73 20 69 73 20  6d 79 20 62 61 73 68 20   This is  my bash 
 73 63 72 69 70 74 20 6d  65 73 73 61 67 65 20 6e   script m essage n
 75 6d 62 65 72 20 31 34                            umber 14
```

MQTT / 9 / security

MQTT with TLS/SSL works very much the same as https (which we are familiar with from web usage).
Example with letsencrypt certificates:

```
/etc/mosquitto/conf.d/ssl.conf
listener 8883
certfile /etc/letsencrypt/live/influx.itu.dk/cert.pem
cafile /etc/letsencrypt/live/influx.itu.dk/chain.pem
keyfile /etc/letsencrypt/live/influx.itu.dk/privkey.pem
```

A mosquitto_pub client would publish like this:

```
mosquitto_pub -h influx.itu.dk -p 8883 --capath
/etc/ssl/certs/ -t topic/test -m "encrypted msg"
```

MQTT / 10 / security

MQTT with TLS/SSL works very much the same as https.

In addition to “web style” TLS/SSL, specific client certificates can be demanded.

Username / password protection is also available.

MQTT / 11 / security note

Note:

If devices publish encrypted data,
but the broker allows subscribers to listen unencrypted,
data will be readable on the network!