MQTT

Core Concepts & Developments Setup

Getting started

MQTT Specification

Software

Use Cases

FAQ

Use Cases

MQTT is used in a large variety of use cases and industries.

Automotive

- HiveMQ: BMW Car-Sharing application relies on HiveMQ for reliable connectivity
- EMQ helps SAIC Volkswagen building IoV platform

Logistics

- Transportation & Logistics company cuts costs and improves asset tracking

Manufacturing

- Transforming Manufacturing Efficiency: The Power of MQTT in Industrial Solutions

MQTT

Getting started

MQTT Specification

Software

Use Cases

FAQ

Smart Home

- IBM Telemetry use case: Home energy monitoring and control
- IBM Telemetry use case: Home patient monitoring
- The eFon Technology's Smart Home security system trusts Bevywise MQTT solution

Energy

- EMQ helps IoT innovation in the petrochemical industry
- Energy company maximizing MQTT for control
- MQTT implementation on Celikler Holding's power plant monitoring

Consumer Products

- CASO Design creates smart kitchen appliances

Transportation

- Deploying IoT on Germany's DB Railway System
- Air France-KLM Group improves efficiency and passenger experience

Core Concepts

Broker: The central server that routes messages between clients (like a post office).

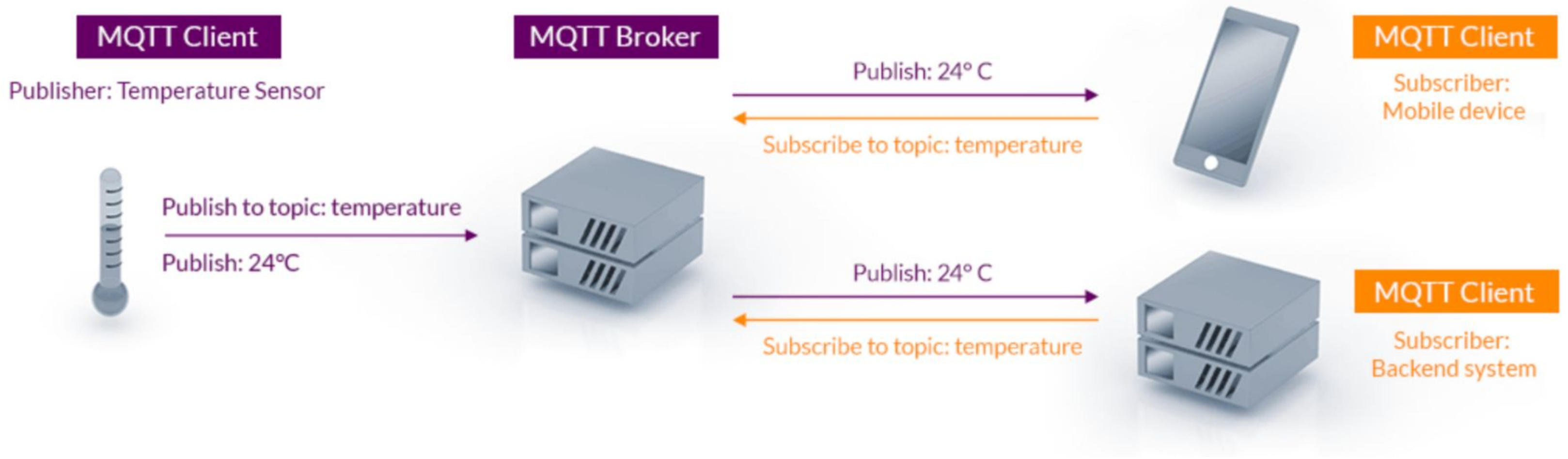
Client: Any device (sensor, app, microcontroller) that connects to the broker.

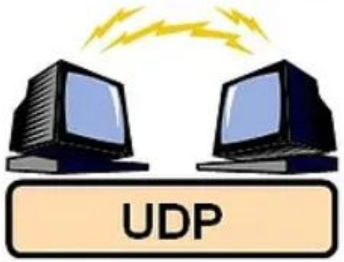
Topic: A UTF-8 string used to organize messages. Think of it like a "channel".

Publish: Sending a message to a topic.

Subscribe: Registering to receive messages on a topic.

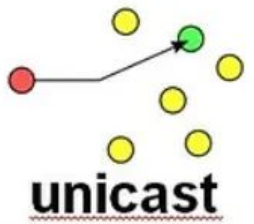
MQTT Publish / Subscribe Architecture

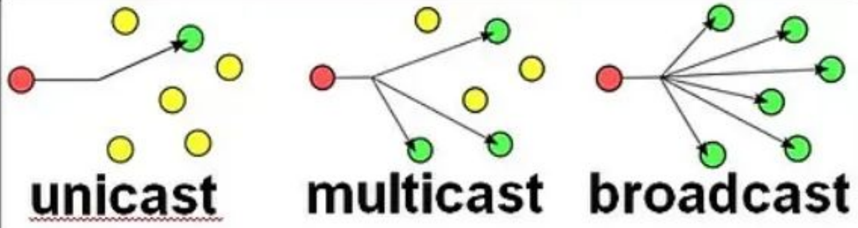

MQTT Client
Publisher: Temperature Sensor
Publish to topic: temperature
Publish: 24°C
MQTT Broker
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Backend system



- **Slower but reliable transfers**
- **Typical applications:**
  - Email
  - Web browsing

- **Fast but non-guaranteed transfers (“best effort”)**
- **Typical applications:**
  - VoIP
  - Music streaming

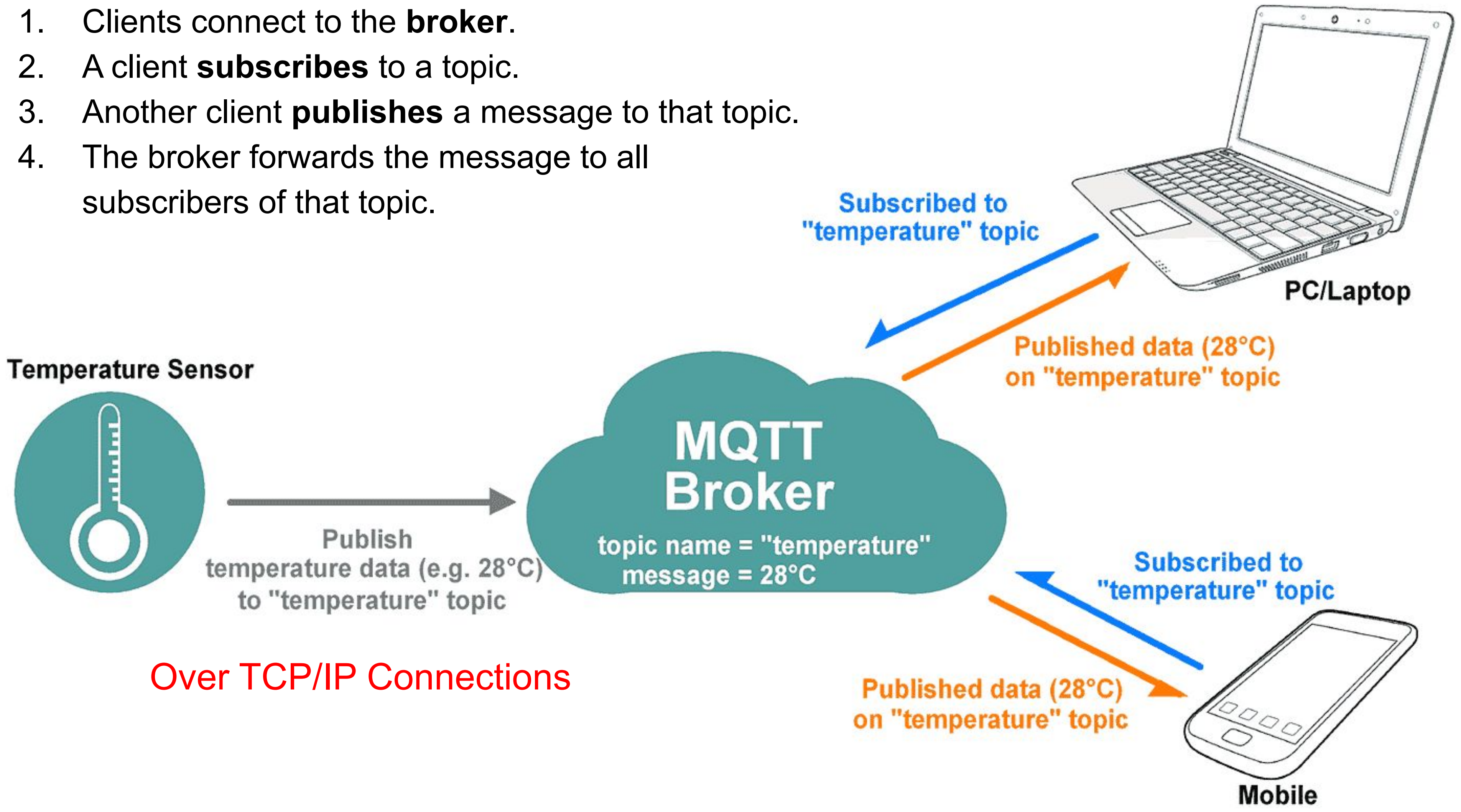




Socket Type	Protocol	Description	Common Use in MQTT
Stream Socket	TCP (SOCK_STREAM)	Reliable, ordered, connection-oriented communication.	✅ **Default for MQTT (port 1883)**
Datagram Socket	UDP (SOCK_DGRAM)	Unreliable, unordered, connectionless. Lightweight.	❌ Not used in MQTT; used in CoAP, DNS etc.
Raw Socket	IP-level access	Direct access to IP protocol. Used for low-level protocols like ICMP (ping).	🔧 Debugging tools, not for MQTT
WebSocket	HTTP over TCP	Layered socket enabling MQTT over web-friendly protocols.	✅ MQTT over WebSocket (port 9001)
Unix Domain Socket	Local IPC	Sockets for inter-process communication on the same host (file path instead of IP).	🔧 Mosquitto supports this for performance
TLS Socket	Secure TCP (SSL/TLS)	Encrypted version of TCP sockets.	✅ MQTT over TLS (port 8883) for security

Use Case	Socket Type	Notes
IoT device to cloud	TCP Socket	Standard MQTT over port 1883
Secure enterprise environment	TLS Socket	MQTT with certificates, port 8883
Web browser app dashboard	WebSocket	Use MQTT over WS on port 9001
Local app-to-broker on same host	Unix Domain Socket	Fastest; no TCP/IP stack involved

1. Clients connect to the **broker**.
2. A client **subscribes** to a topic.
3. Another client **publishes** a message to that topic.
4. The broker forwards the message to all subscribers of that topic.


Subscribed to "temperature" topic
PC/Laptop
Published data (28°C) on "temperature" topic
Temperature Sensor
MQTT Broker
topic name = "temperature"
message = 28°C
Publish temperature data (e.g. 28°C) to "temperature" topic
Subscribed to "temperature" topic
Published data (28°C) on "temperature" topic
Mobile

Over TCP/IP Connections

Security Levels of MQTT

Username/Password Authentication

TLS/SSL Encryption for secure communication

Access Control (topic-level restrictions)

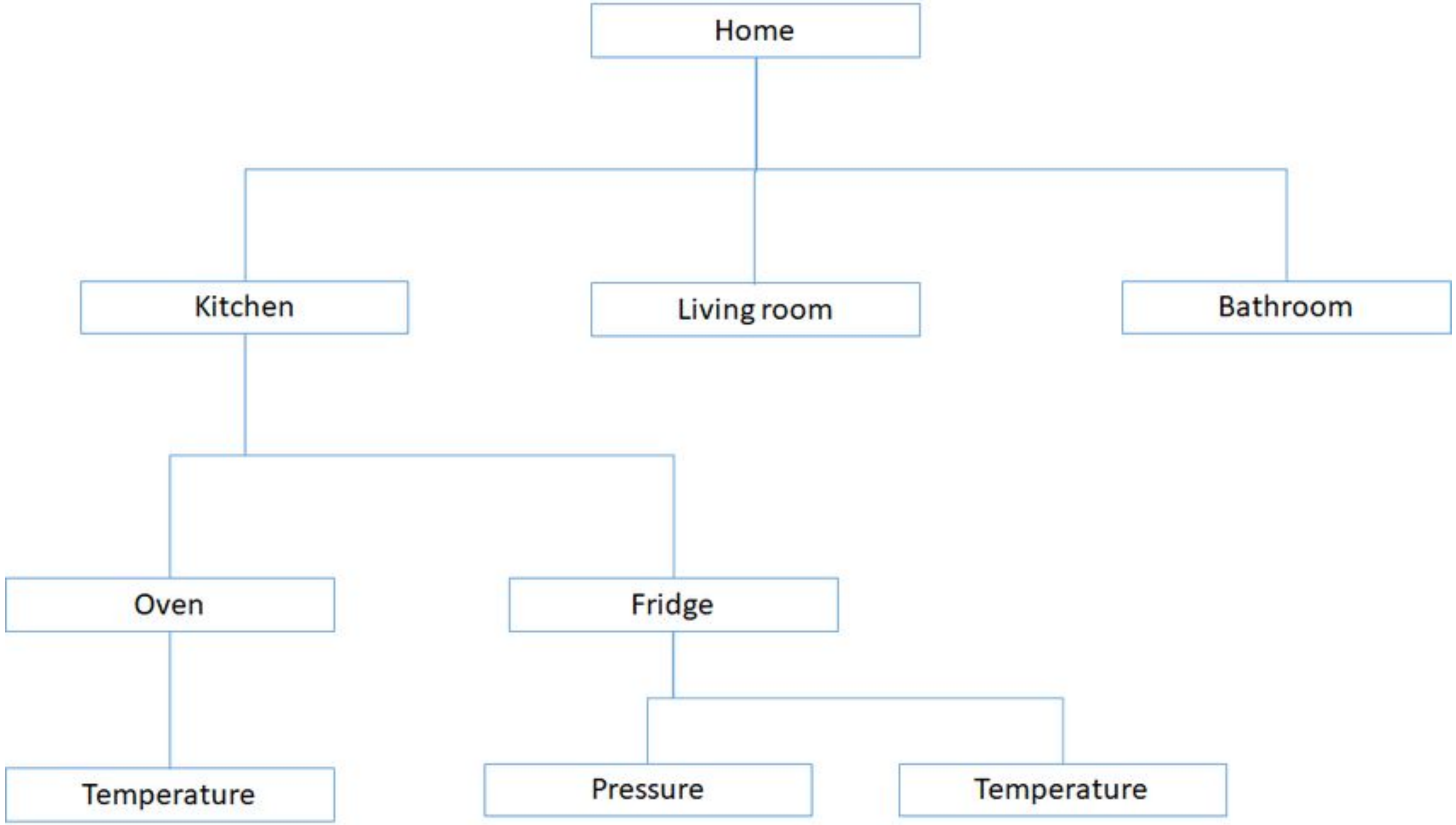
🧩 MQTT Topic Hierarchies and Wildcards

Topics are organized in a hierarchical format using `/`:

- `home/livingroom/temperature`

Wildcards:

- `+` (single-level): `home/+/temperature` matches `home/kitchen/temperature`
- `#` (multi-level): `home/#` matches `home/anything/here`


Home
Kitchen
Living room
Bathroom
Oven
Fridge
Temperature
Pressure
Temperature

1. Retained Messages

- Broker keeps the **last retained message** on a topic.
- New subscribers immediately receive it upon subscribing.

2. Last Will and Testament (LWT)

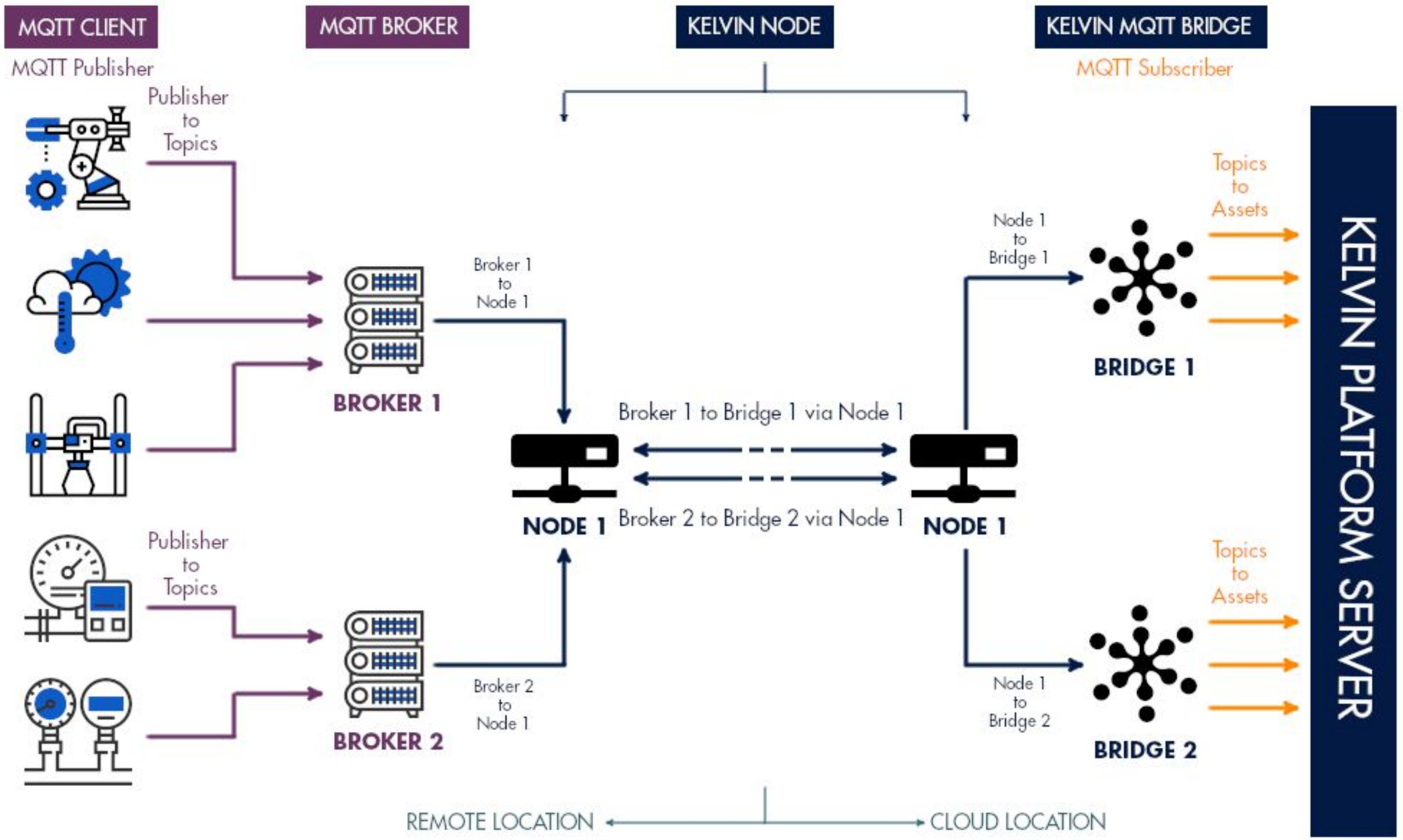
- A message defined at connection time to be sent by the broker **if the client disconnects unexpectedly**.

3. Persistent Sessions

- Allows clients to resume subscriptions and message queues **after reconnecting**.

4. Bridging Brokers

- You can connect multiple MQTT brokers together to share messages across networks or geographic locations.



MQTT CLIENT
MQTT Publisher
MQTT BROKER
KELVIN NODE
KELVIN MQTT BRIDGE
MQTT Subscriber
Publisher to Topics
BROKER 1
Broker 1 to Node 1
Broker 1 to Bridge 1 via Node 1
Broker 2 to Bridge 2 via Node 1
NODE 1
NODE 1
Node 1 to Bridge 1
Topics to Assets
BRIDGE 1
Publisher to Topics
BROKER 2
Broker 2 to Node 1
Node 1 to Bridge 2
Topics to Assets
BRIDGE 2
KELVIN PLATFORM SERVER
REMOTE LOCATION
CLOUD LOCATION

Tools

Brokers: Mosquitto, EMQX, HiveMQ, VerneMQ

Clients: MQTT.fx, MQTT Explorer, mosquitto_pub/sub, Node-RED, custom code (Python `paho-mqtt`, C++, etc.)

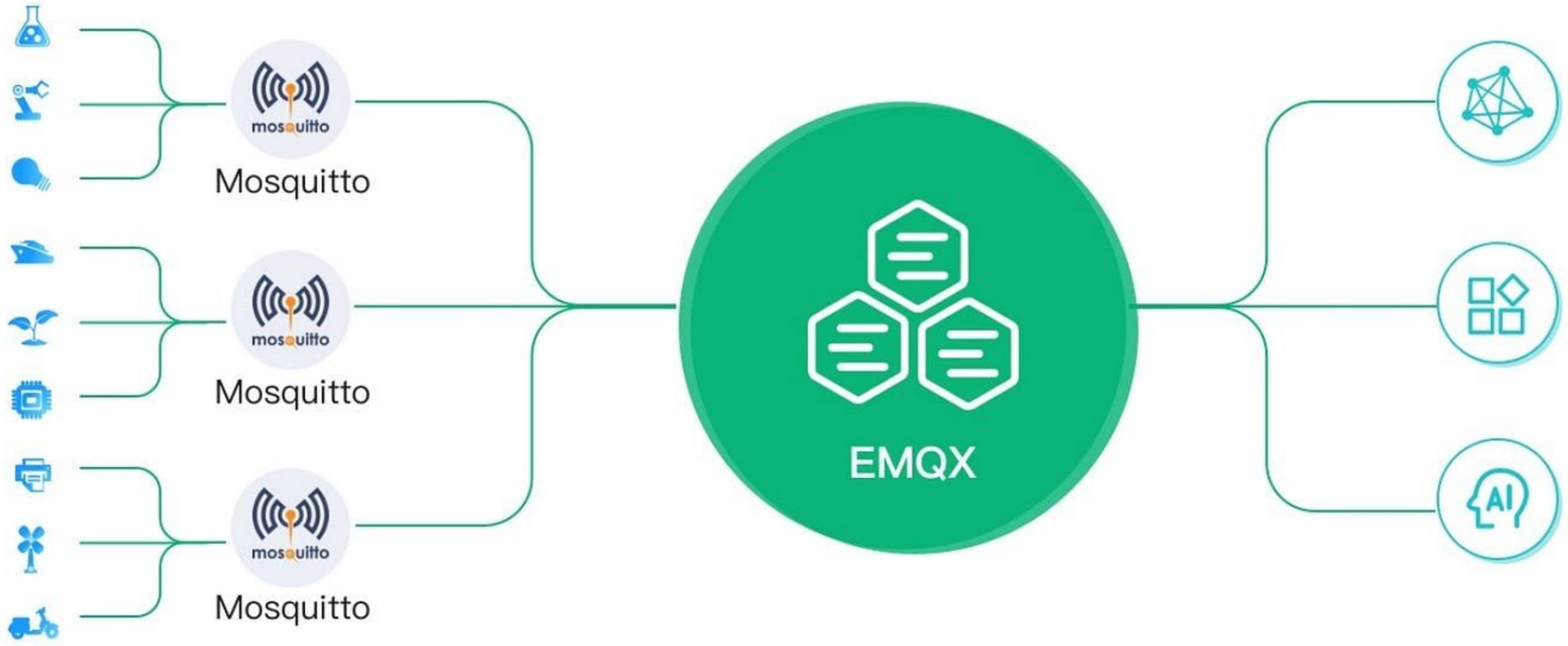
Command Line Tools:

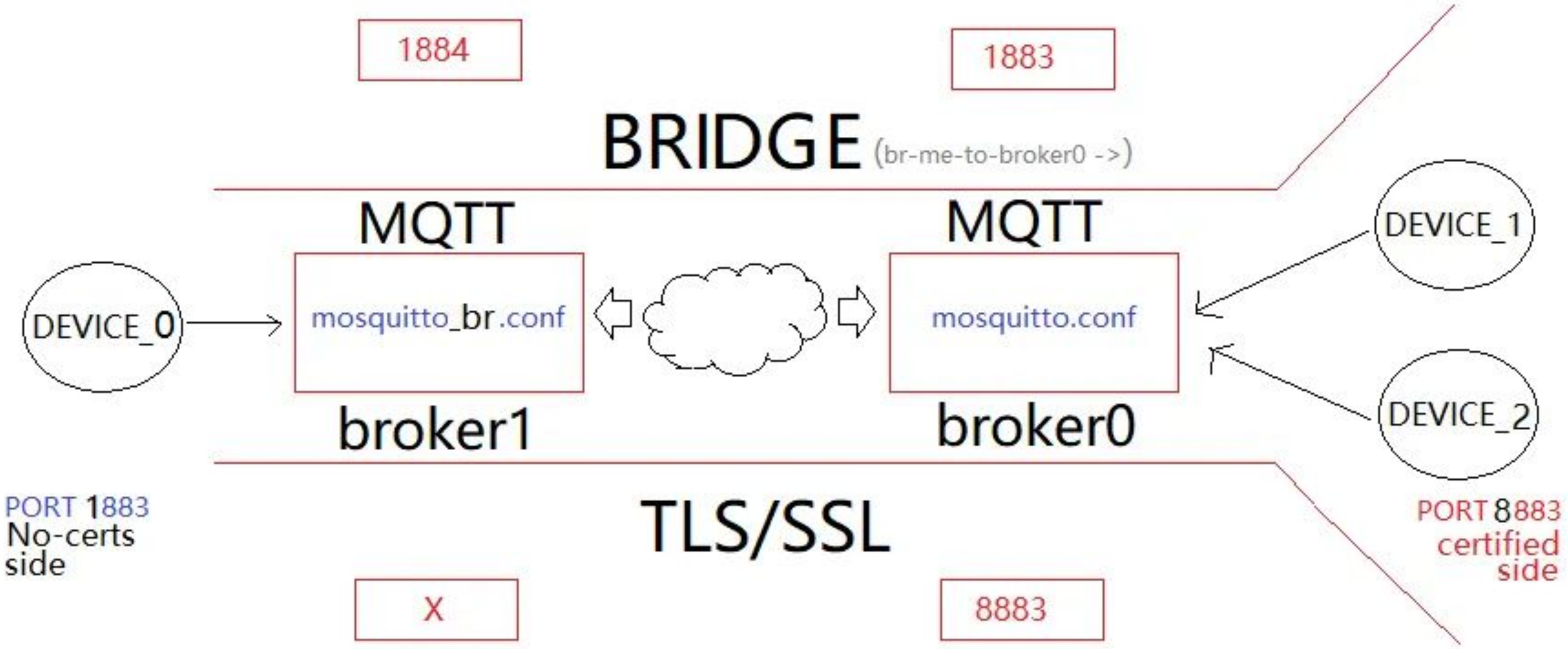
```
bot@bot-x:~$ mosquitto_ctrl    mosquitto_pub     mosquitto_sub   mosquitto_passwd  mosquitto_rr
```

Types of MQTT broker

MQTT message broker type comparison:

Type	Description
Open-source MQTT broker	• Often available at minimal or no cost, these brokers offer the flexibility to modify the code to suit your specific requirements. • They are typically maintained by open-source communities, and are ideal for testing, prototyping, personal projects and small to medium-scale applications. • Examples: Mosquitto
Cloud MQTT broker	• An online MQTT broker hosted and run by a cloud service provider that manages infrastructure, providing scalability and reducing maintenance efforts. • These brokers are ideal for large-scale IoT deployments. • Examples: AWS IoT Core and Azure IoT Hub.
On-premises (local, self-hosted) MQTT broker	• Installed and hosted directly on an individual's or organisation's servers/infrastructure for complete control over broker environment, data security, and configuration. • Local MQTT broker setups are ideal for organizations that must meet stringent regulatory requirements and maintain full control over their data. • Examples: Pro Edition for Mosquitto
Enterprise MQTT broker	• Commercial-grade MQTT broker that offers rich features and robust support for mission-critical IoT applications. • Examples: Pro Edition for Mosquitto


mosquitto
Mosquitto
mosquitto
Mosquitto
mosquitto
Mosquitto
EMQX
AI

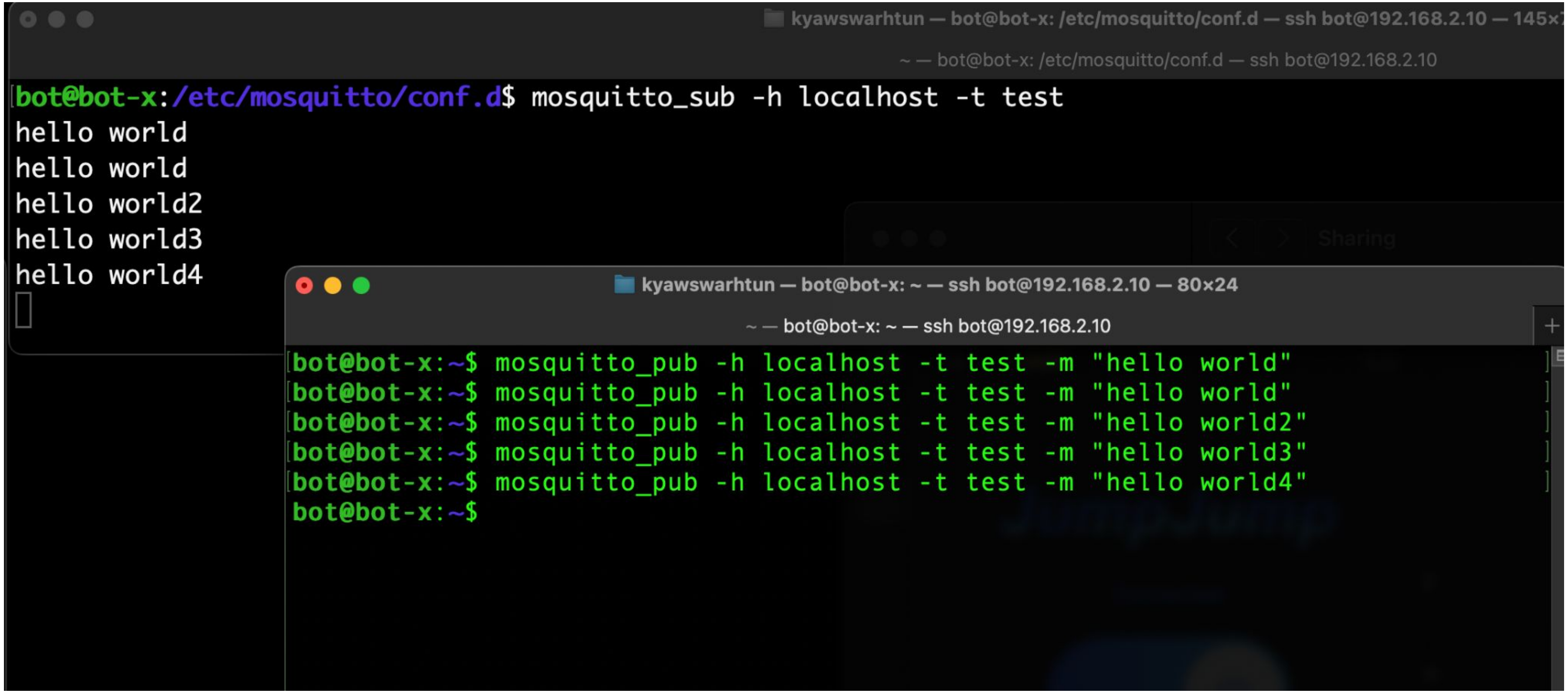

1884
1883
BRIDGE (br-me-to-broker0 ->)
MQTT
MQTT
DEVICE_1
DEVICE_0
mosquitto_br.conf
mosquitto.conf
broker1
broker0
DEVICE_2
PORT 1883
No-certs
side
TLS/SSL
PORT 8883
certified
side
X
8883

Installation Broker on Linux System

- # sudo apt install **mosquitto mosquitto-clients**
- # sudo systemctl status mosquitto
- # sudo systemctl enable mosquitto
- # sudo systemctl start mosquitto

- Configure Broker ??

Publish / Subscribe Testing (With mqtt_cli Client)



kyawswarhtun — bot@bot-x: /etc/mosquitto/conf.d — ssh bot@192.168.2.10 — 145×
~ — bot@bot-x: /etc/mosquitto/conf.d — ssh bot@192.168.2.10

```
bot@bot-x:/etc/mosquitto/conf.d$ mosquitto_sub -h localhost -t test
hello world
hello world
hello world2
hello world3
hello world4
```

kyawswarhtun — bot@bot-x: ~ — ssh bot@192.168.2.10 — 80×24
~ — bot@bot-x: ~ — ssh bot@192.168.2.10

```
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world2"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world3"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world4"
bot@bot-x:~$
```

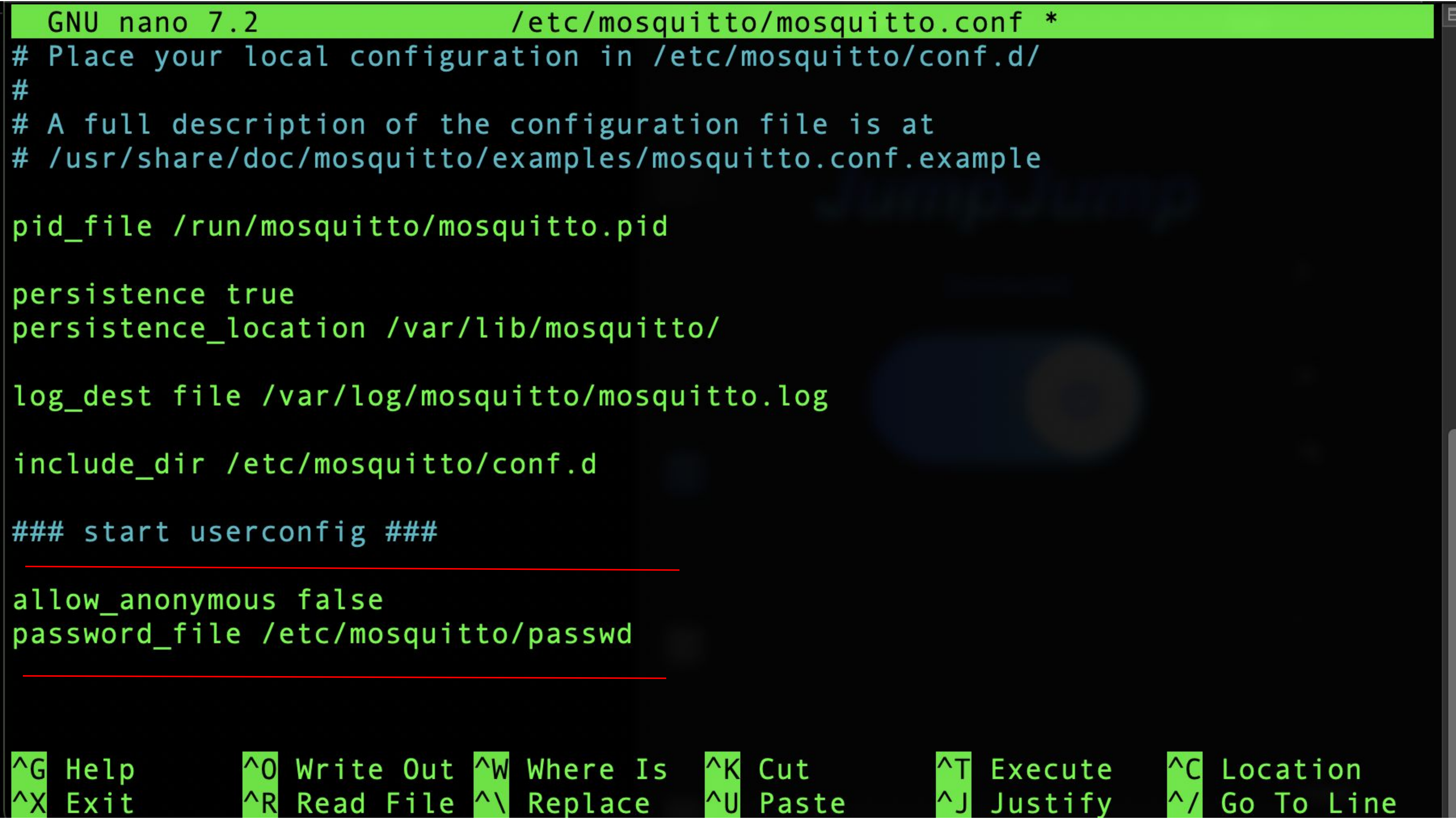
Configuration on Local & Remote Host

```
bot@bot-x:/etc/mosquitto$ ls
aclfile.example  ca_certificates  certs  conf.d  mosquitto.conf  pskfile.example  pwfile.example
```

Set Auth (username, password)

```
bot@bot-x:~$ sudo mosquitto_passwd -c /etc/mosquitto/passwd bot
Password:
Reenter password:

bot@bot-x:~$ cat /etc/mosquitto/passwd
bot:$7$101$aQP+Yw1np7NbyAIb$fgDivKODODB/4Ikn7YyjP/v+R1276uyx4K4FLe/BVIa4IHOAUeFvQyBvxoO
sx3zH7W80jImBXIz3lJa0KPUWvQ==
```



```
  GNU nano 7.2                /etc/mosquitto/mosquitto.conf *
# Place your local configuration in /etc/mosquitto/conf.d/
#
# A full description of the configuration file is at
# /usr/share/doc/mosquitto/examples/mosquitto.conf.example

pid_file /run/mosquitto/mosquitto.pid

persistence true
persistence_location /var/lib/mosquitto/

log_dest file /var/log/mosquitto/mosquitto.log

include_dir /etc/mosquitto/conf.d

### start userconfig ###

allow_anonymous false
password_file /etc/mosquitto/passwd



^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File ^\ Replace   ^U Paste     ^J Justify   ^/ Go To Line
```

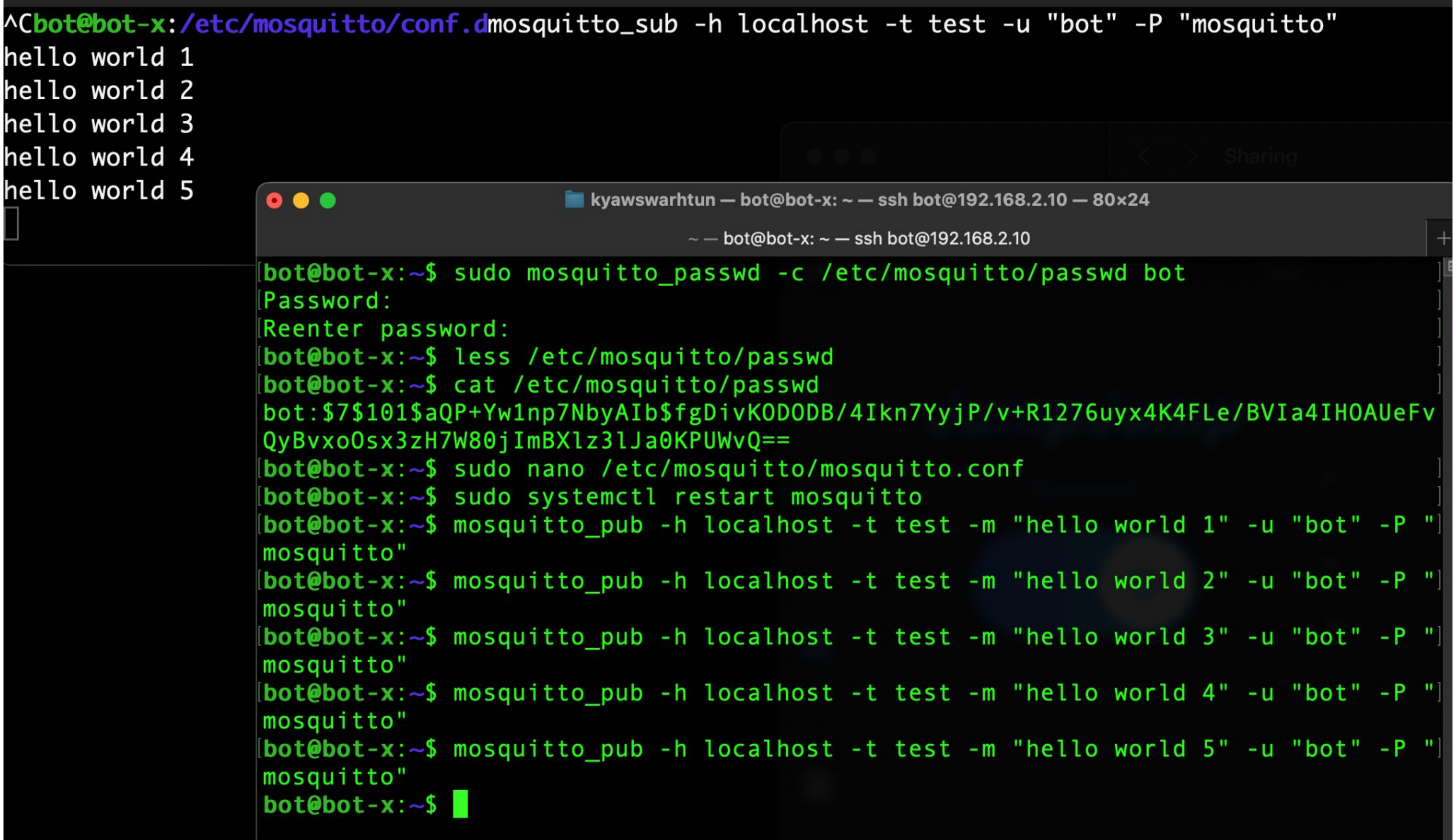
Testing With New Configuration

Reload Configuration..

```
# sudo systemctl restart mosquitto
```

```
$ mosquitto_sub -h localhost -t test -u "bot" -P "mosquitto"

$ mosquitto_pub -h localhost -t test -m "hello world 1" -u "bot" -P "mosquitto"
```



```
^Cbot@bot-x:/etc/mosquitto/conf.dmosquitto_sub -h localhost -t test -u "bot" -P "mosquitto"
hello world 1
hello world 2
hello world 3
hello world 4
hello world 5
```

kyawswarhtun — bot@bot-x: ~ — ssh bot@192.168.2.10 — 80×24

~ — bot@bot-x: ~ — ssh bot@192.168.2.10

```
bot@bot-x:~$ sudo mosquitto_passwd -c /etc/mosquitto/passwd bot
Password:
Reenter password:
bot@bot-x:~$ less /etc/mosquitto/passwd
bot@bot-x:~$ cat /etc/mosquitto/passwd
bot:$7$101$aQP+Yw1np7NbyAIb$fgDivKODODB/4Ikn7YyjP/v+R1276uyx4K4FLe/BVIa4IHOAUeFv
QyBvxoOsx3zH7W80jImBXlz3lJa0KPUWvQ==
bot@bot-x:~$ sudo nano /etc/mosquitto/mosquitto.conf
bot@bot-x:~$ sudo systemctl restart mosquitto
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 1" -u "bot" -P "
mosquitto"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 2" -u "bot" -P "
mosquitto"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 3" -u "bot" -P "
mosquitto"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 4" -u "bot" -P "
mosquitto"
bot@bot-x:~$ mosquitto_pub -h localhost -t test -m "hello world 5" -u "bot" -P "
mosquitto"
bot@bot-x:~$ 
```

QoS defines the reliability of message delivery:

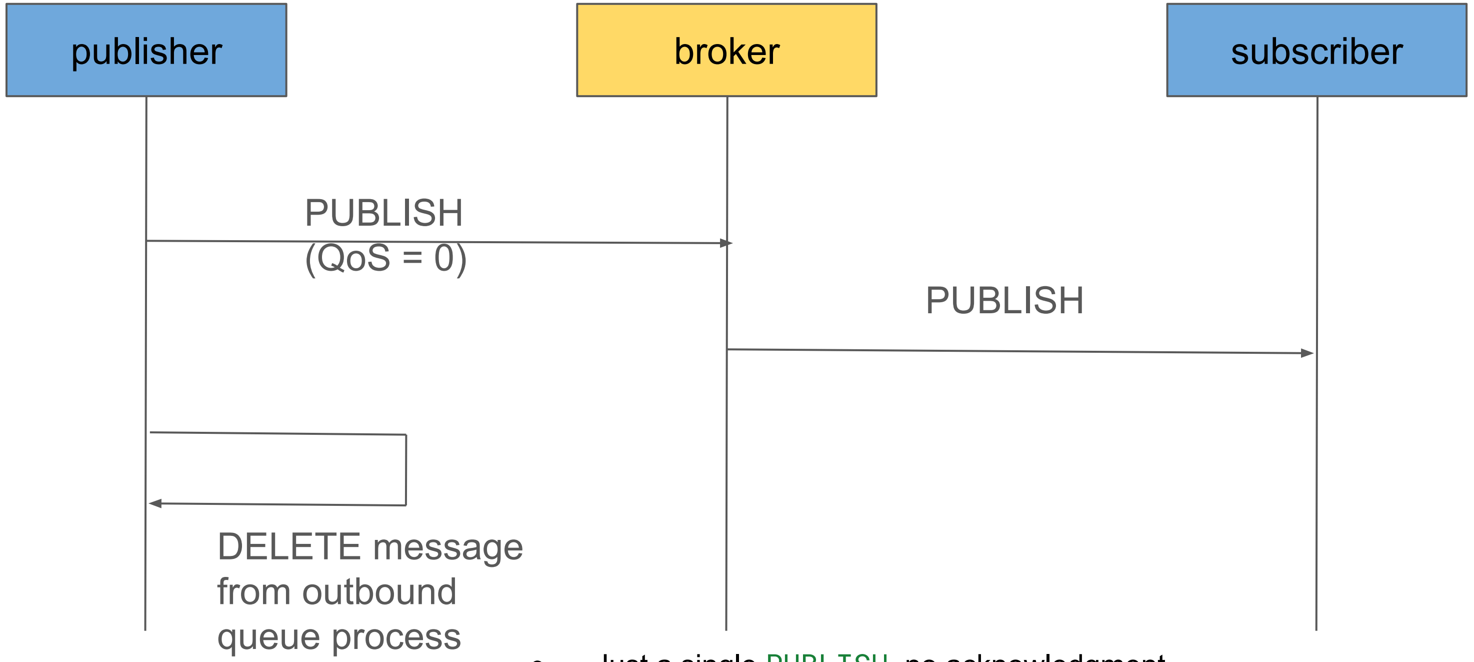
QoS 0: when we prefer that the message will arrive at most once; the message will be received or it won't, there isn't a chance of a duplicate; at most once; fire and forget; the most unreliable transfer mode.

QoS 1: when we want the message to arrive at least once but don’t care if it arrives twice (or more); at least once;

QoS 2: when we want the message to arrive exactly once. A higher QOS value means a slower transfer; exactly once.

MQTT : Quality Of Service

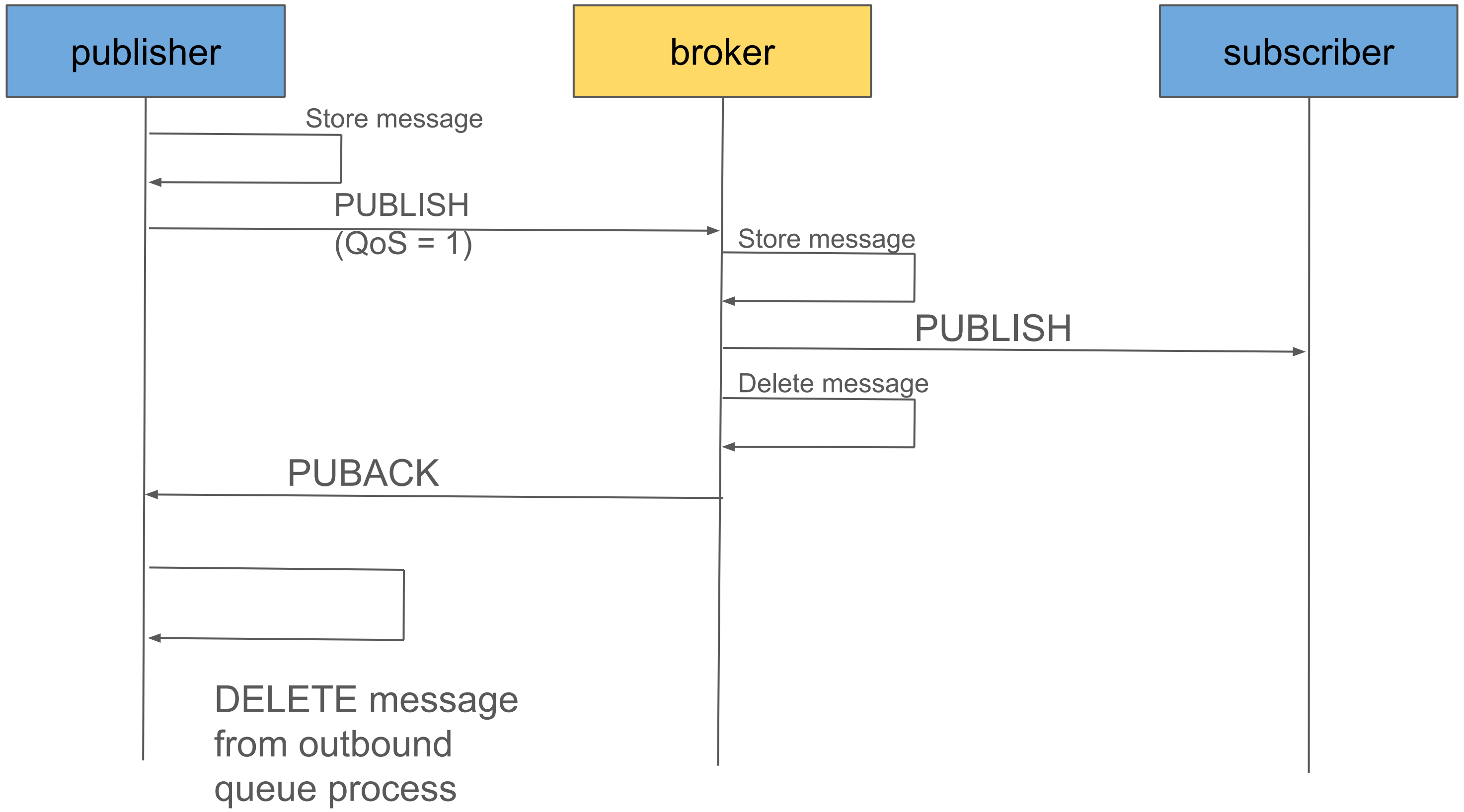
QoS 0 : At Most Once (fire and forget)

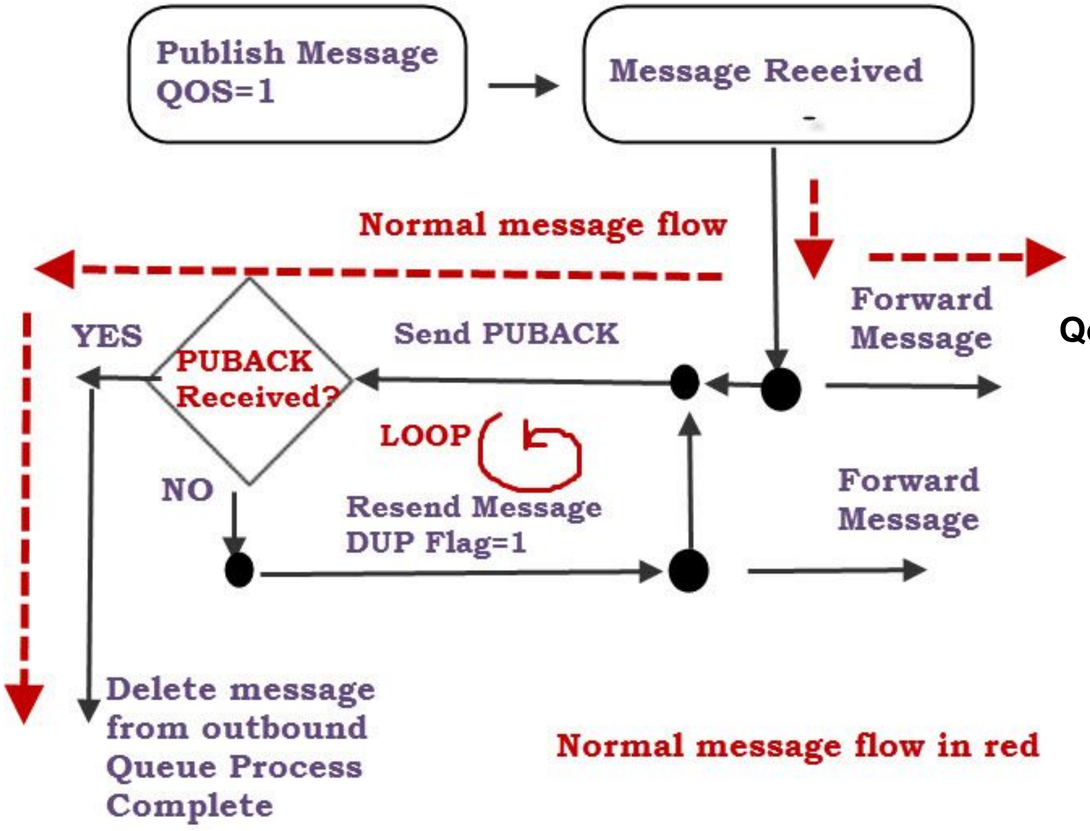


- Just a single `PUBLISH`, no acknowledgment.

MQTT : Quality Of Service

QoS 1 : At Least Once





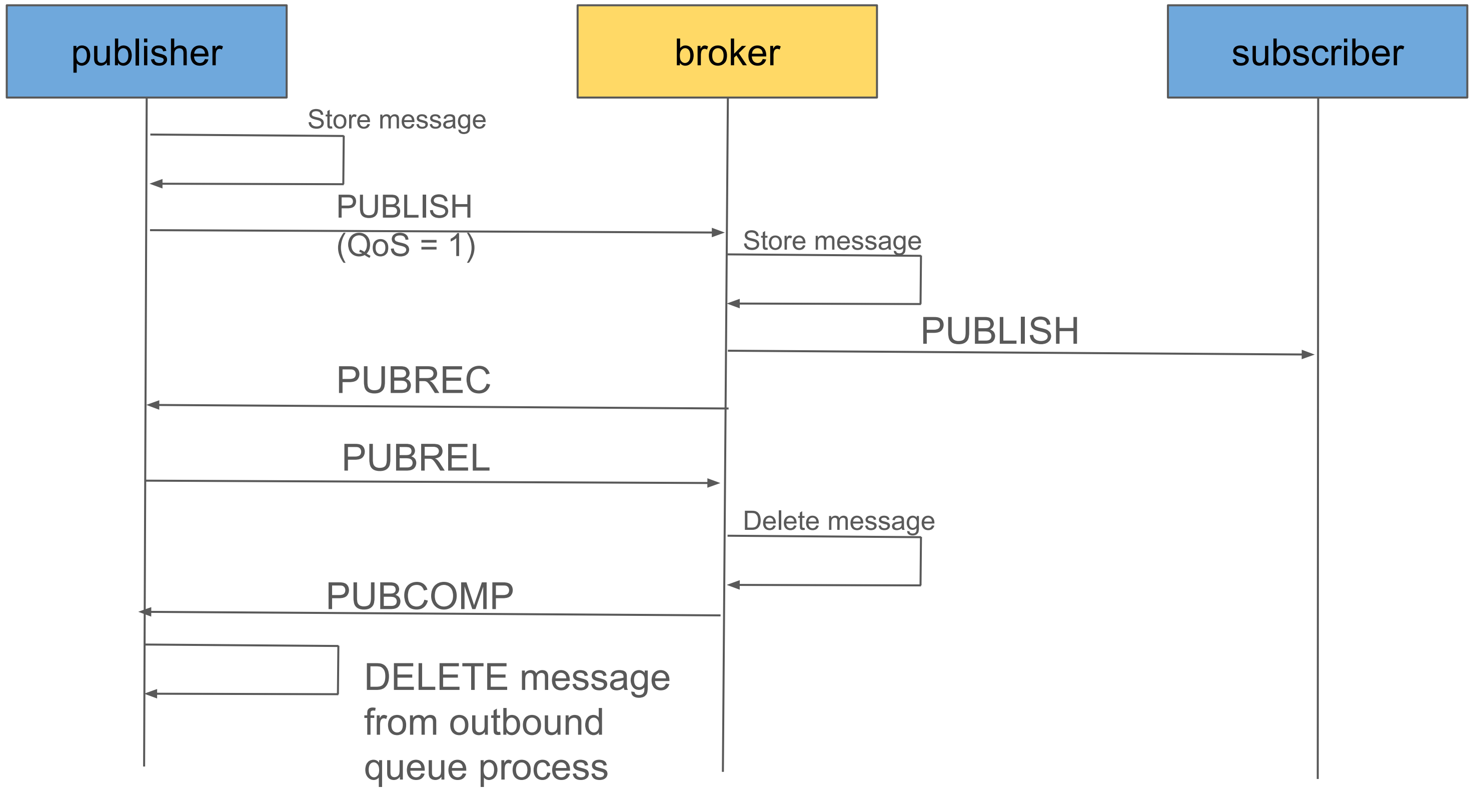
MQTT QOS 1 Message Flow Diagram

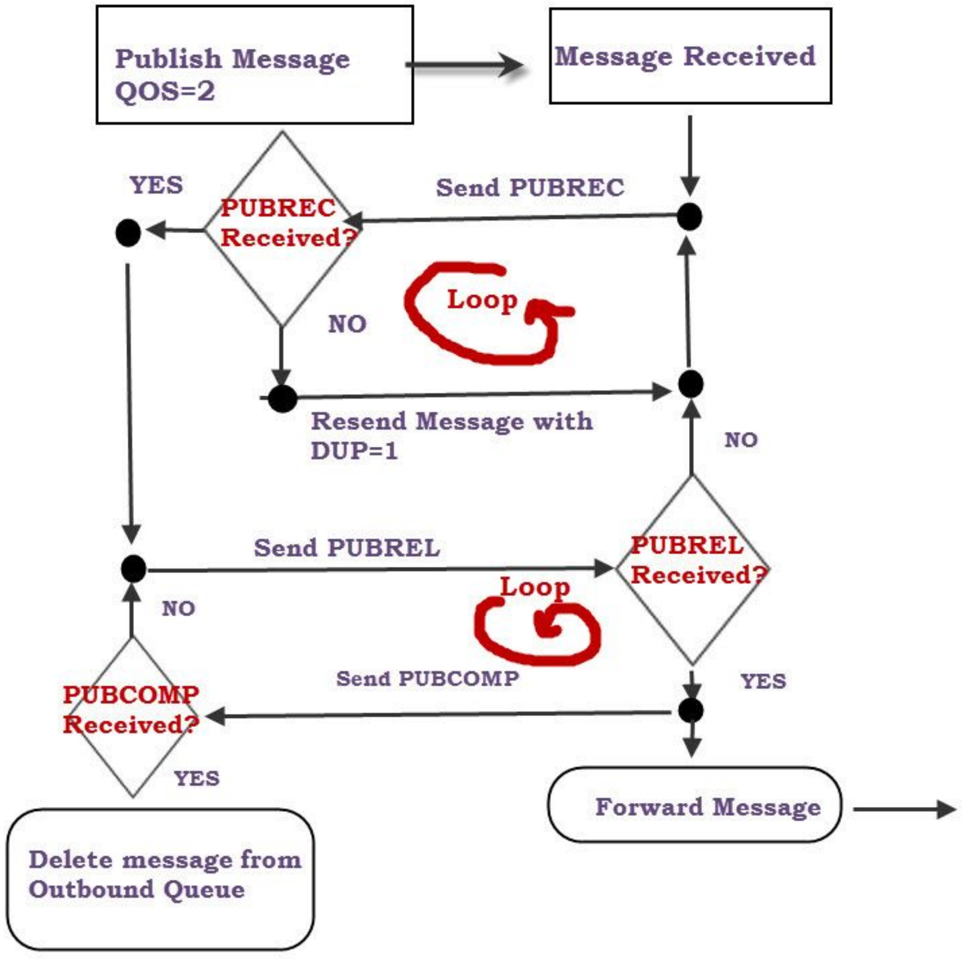
QoS 1:

- `PUBLISH` → wait for `PUBACK`.

MQTT : Quality Of Service

QoS 2 : Exactly Once





MQTT QOS 2 Message Flow Diagram

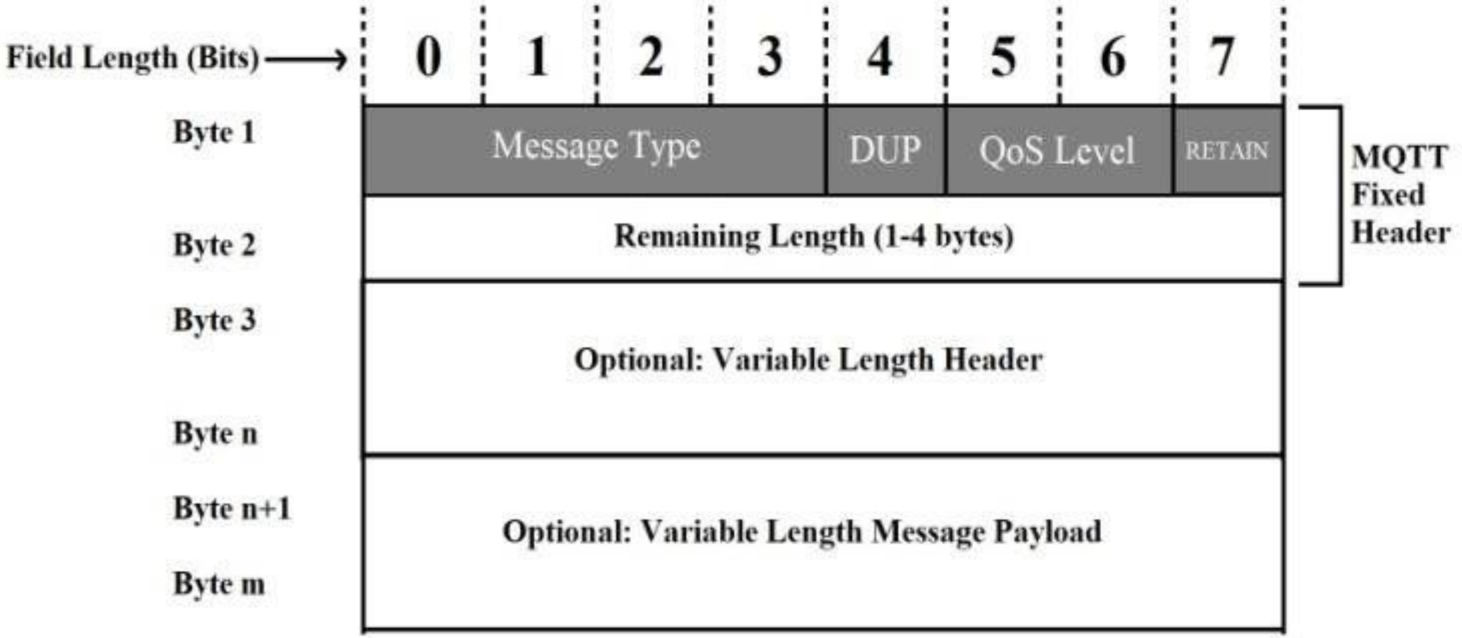
QoS 2:

- Four-way handshake:
  - `PUBLISH → PUBREC → PUBREL → PUBCOMP`
- Guarantees **no duplicate delivery**.

MQTT Packet Flow (Core Protocol Packets)

Packet	Purpose
`CONNECT`	Client to broker, starts a session
`CONNACK`	Broker to client, acknowledges connection
`PUBLISH`	Send a message to a topic
`PUBACK`	QoS 1 message acknowledgment
`PUBREC`	QoS 2 – Message received (part 1)
`PUBREL`	QoS 2 – Message released (part 2)
`PUBCOMP`	QoS 2 – Completion confirmation
`SUBSCRIBE`	Client subscribes to a topic
`SUBACK`	Subscription acknowledgment
`UNSUBSCRIBE`	Client unsubscribes
`UNSUBACK`	Acknowledge unsubscribe
`PINGREQ`	Client heartbeat check
`PINGRESP`	Broker heartbeat response
`DISCONNECT`	Graceful session close

Packet	Direction	Description	QoS Impact
`CONNECT`	Client → Broker	Start session	-
`CONNACK`	Broker → Client	Connection accepted/denied	-
`PUBLISH`	Client ↔ Broker	Publish a message to a topic	✅
`PUBACK`	Broker → Client	Acknowledge QoS 1 message	QoS 1
`PUBREC`	Broker → Client	Received QoS 2 message	QoS 2
`PUBREL`	Client → Broker	Release QoS 2 message	QoS 2
`PUBCOMP`	Broker → Client	Complete QoS 2	QoS 2
`SUBSCRIBE`	Client → Broker	Subscribe to a topic	-
`SUBACK`	Broker → Client	Acknowledge subscription	-
`PINGREQ`	Client → Broker	Keepalive	-
`PINGRESP`	Broker → Client	Response to keepalive	-
`DISCONNECT`	Client → Broker	Graceful shutdown	-



Message Type: PUBLISH, PUBACK, PUBREC, PUBREL, PUBCOMP
Qos: 0/1/2
DUP: DUPLICATE FLAG
RETAIN: Set to ON to store last known value

Persistent Configuration of Broker (bridge extensions)

MQTT is dynamic by design – topics are not “pre-declared” like in AMQP (e.g., RabbitMQ).

But you **can enforce topic-level control** and simulate structure.

For More information Please reference pdf (in https://github.com/geommax/mqtt.stm.io)

- ACL file
- Password file (or) TLS
- QoS configuration
- Payload configuration

Thank You

CLIENT NAME **XX.XX.XX**

Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam *nonummy* nibh euismod tincidunt ut laoreet dolore magna aliquam erat *volutpat.*