010123131

Software Development Practice I

Handout #8

<rawat.s@eng.kmutnb.ac.th>

Last Update: 2024-08-05

The MQTT Protocol for IoT Applications

Expected Learning Outcomes

- **Setting up an MQTT broker on Linux:**
  - **Environment Setup**: Use an Ubuntu VM and an SBC.
  - **Installation**: Install and run a **Mosquitto MQTT broker** on Linux, either through a native installation or using a Docker image. .
- **MQTT client programming:**
  - **MQTT Clients:** Write **Python, C/C++ and Arduino code** for message publication / subscription via MQTT.
  - **Testing**: Use real **Arduino boards** and/or the **Wokwi simulator** to demonstrate and test Arduino sketches.

MQTT

- **MQTT = Message Queuing Telemetry Transport**
  - an open OASIS standard (since 2013) and an ISO recommendation (ISO/IEC 20922) – an the most commonly used messaging protocol for the Internet of Things (IoT).
  - a lightweight publish / subscribe messaging transport protocol for machine-to-machine (M2M) communication.
  - widely used for messaging and data exchange between IoT and Industrial IoT (IIoT) devices, such as embedded devices, sensors, industrial PLCs, etc.

MQTT

- The MQTT protocol is used to connect devices based on the publish / subscribe (pub/sub) pattern.
  - The sender (publisher) and the receiver (subscriber) communicate via topics.
  - The connection between them is handled by the MQTT broker, which filters and distributes incoming messages to the subscribers.
- Unlike HTTP’s request / response paradigm, MQTT is event-driven and the broker decouples the clients (publishers and subscribers) from each other.

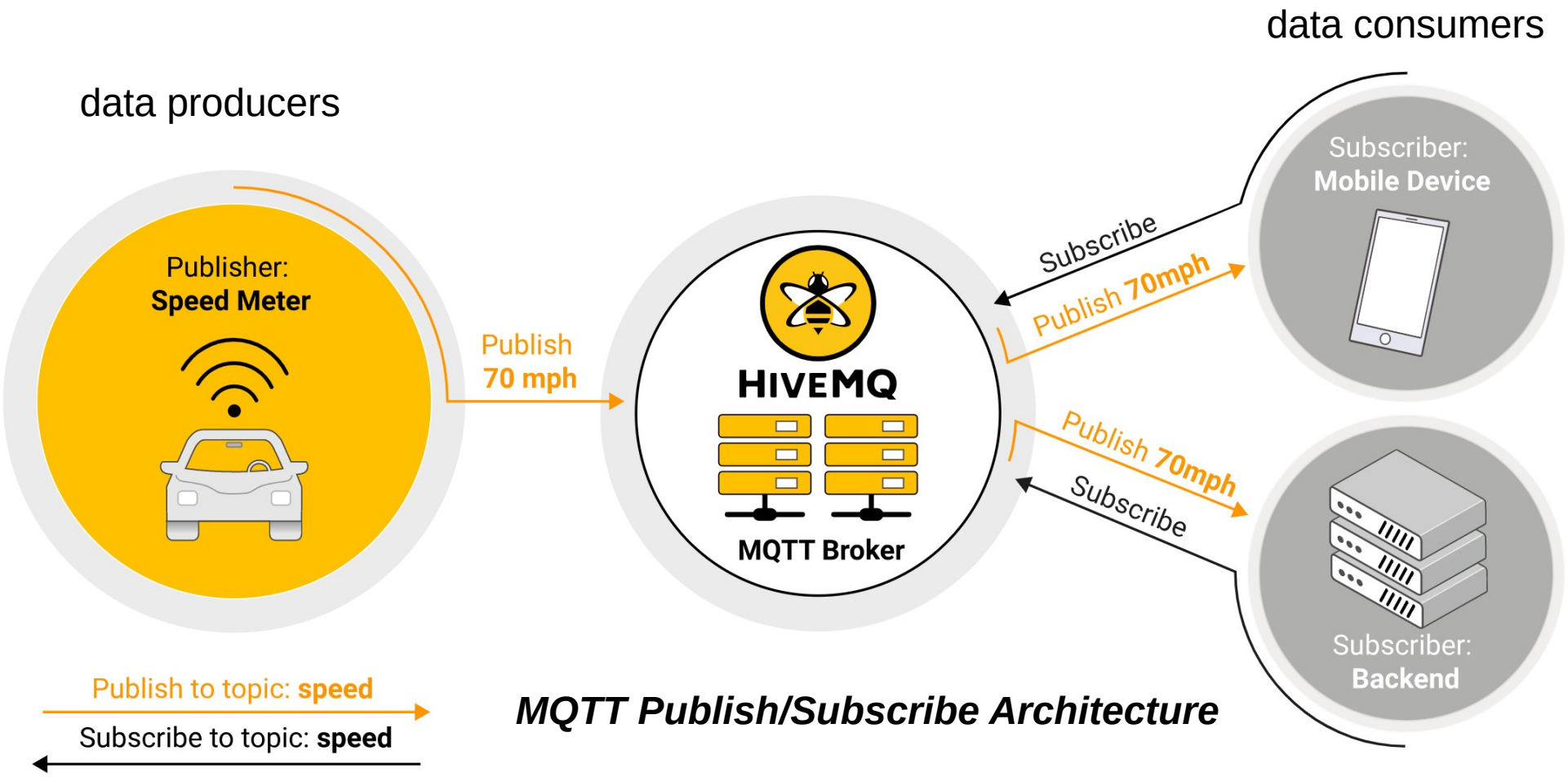


Image source: https://www.hivemq.com/blog/how-to-get-started-with-mqtt/

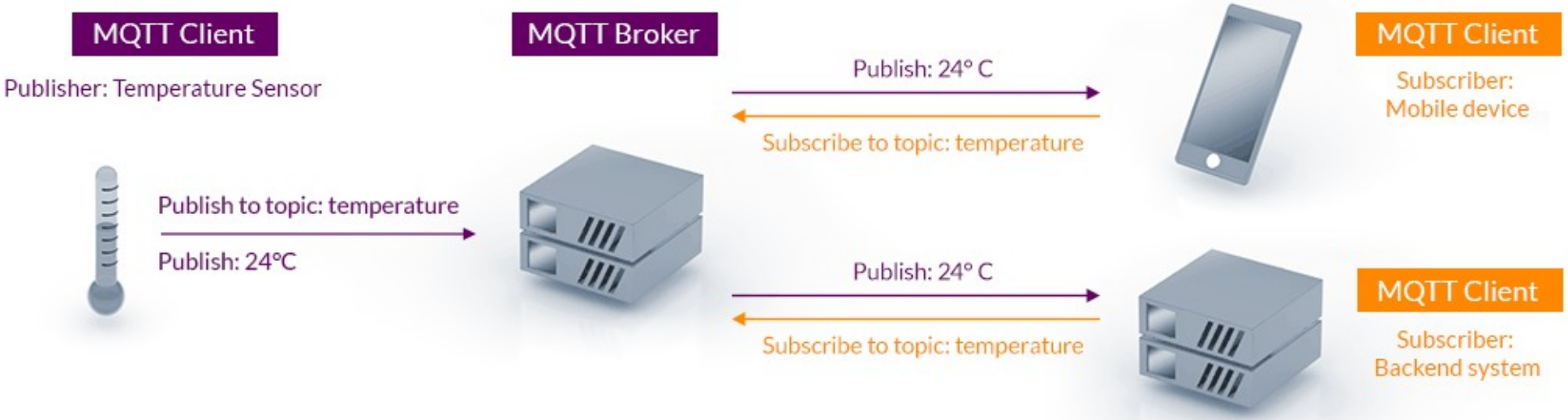


Image source: https://mqtt.org

MQTT Protocol Versions

- There are two versions of the specification: MQTT 3.1.1 and MQTT 5.
  - Most commercial MQTT brokers now support MQTT 5 but many of the IoT managed cloud services only support MQTT 3.1.1.
  - It is highly recommended to use version 5 for new IoT deployments due to the new features that focus on more robust systems and cloud native scalability.

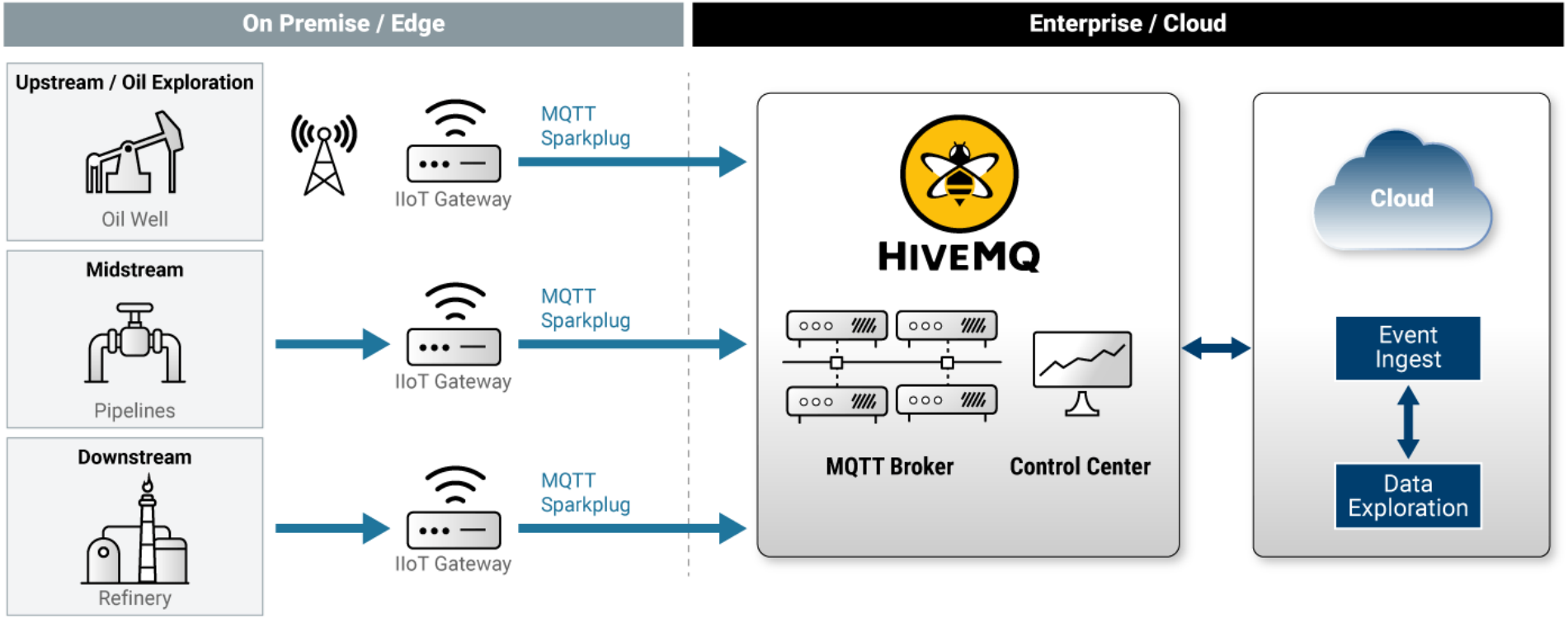
http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html
http://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html

Benefits of MQTT

- Lightweight and efficient to minimize resources required for the client and network bandwidth and supports:
  - Quality of Service (QoS) levels to support message reliability.
  - Persistent sessions or connections between device and server that reduces re-connection time required over unreliable networks.
  - Message encryption with SSL/TLS protocols (v1.3 / v1.2 / v1.1) and server / client authentication.
- Providing a good choice for wireless networks that experience varying levels of latency due to occasional bandwidth constraints or unreliable connections.

Examples of MQTT Use Cases

- Smart home systems
- Smart farming and agricultural / precision farming
- Smart metering and billing systems
- Gathering ambient or environmental sensor data
- Machine health data monitoring for preventive maintenance
- Remote asset management
- Remote performance monitoring



Oil & Gas Remote Asset Management

(Image source: https://www.hivemq.com/solutions/energy-solution-whitepaper/)

MQTT Architecture

- The MQTT broker is responsible for dispatching messages between senders and the receivers.
- A MQTT client publishes a message with a specific topic to the broker and other MQTT clients can subscribe to the topics they want to receive.
- The MQTT broker uses the topics and the subscriber list to dispatch messages to appropriate clients and is able to buffer messages that can’t be dispatched to MQTT clients that are not connected. This is very useful for situations where network connections are unreliable.
- The protocol supports 3 different types of QoS messages:
  0 - at most once, 1 - at least once, and 2 - exactly once.

MQTT Clients

- There are open source libraries for MQTT clients available in different computer languages.
  - **Eclipse Paho Library** (C, Python, …)
    - https://www.eclipse.org/paho/
  - **HiveMQ MQTT Client Library** (Java)
    - https://github.com/hivemq/hivemq-mqtt-client
  - **MQTT.js** and **Async-MQTT.js** (Node.js)
    - https://github.com/mqttjs
    - https://github.com/mqttjs/async-mqtt

Client	MQTT 3.1	MQTT 3.1.1	MQTT 5.0	LWT	SSL / TLS	Automatic Reconnect	Offline Buffering	Message Persistence	WebSocket Support	Standard MQTT Support	Blocking API	Non-Blocking API	High Availability
Java	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Python	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔	✔	✔	✖
JavaScript	✔	✔	✖	✔	✔	✔	✔	✔	✔	✖	✖	✔	✔
GoLang	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔
C	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
C++	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
Rust	✔	✔	✖	✔	✔	✔	✔	✔	✖	✔	✔	✔	✔
.Net (C#)	✔	✔	✖	✔	✔	✖	✖	✖	✖	✔	✖	✔	✖
Android Service	✔	✔	✖	✔	✔	✔	✔	✔	✔	✔	✖	✔	✔
Embedded C/C++	✔	✔	✖	✔	✔	✖	✖	✖	✖	✔	✔	✔	✖

Paho MQTT Client (v1.4) Comparison (Source: Eclipse.org)

MQTT Clients

- **Open Source Arduino Libraries**
  - **MQTT Library** (by Joel Gaehwiler)
    - https://www.arduino.cc/reference/en/libraries/mqtt/
  - **PubSubClient** (by Nick O'Leary)
    - https://github.com/knolleary/pubsubclient
  - **Async MQTT Library** (for ESP8266 / ESP32)
    - https://github.com/marvinroger/async-mqtt-client
  - **AsyncMQTT_Generic Library** (by Marvin Roger & Khoi Hoang)
    - https://github.com/khoih-prog/AsyncMQTT_Generic
  - **Adafruit MQTT Library**
    - https://github.com/adafruit/Adafruit_MQTT_Library

GUI-based MQTT Clients

- Examples of **GUI-based MQTT Client Apps**:
  - **MQTT Explorer (free, open source)**
    - https://github.com/thomasnordquist/MQTT-Explorer
    - https://mqtt-explorer.com/
  - **MQTTBox**
    - https://github.com/workswithweb/MQTTBox
  - **MQTT Web Client**
    - https://mqttx.app/

MQTT Brokers

- Examples of **open source MQTT brokers**:
  - **Eclipse Mosquitto**
    - https://github.com/eclipse/mosquitto
  - **HiveMQ Community Edition**
    - https://github.com/hivemq/hivemq-community-edition
  - ...

Public MQTT Brokers

- Examples of public MQTT brokers:
  - **Mosquitto Broker**
    - https://test.mosquitto.org/
  - **HiveMQ Broker**
    - http://broker.hivemq.com/
  - **EMQX Broker**
    - https://www.emqx.com/en/mqtt/public-mqtt5-broker

MQTT Sessions

- An MQTT session is divided into four stages: connection, authentication, communication and termination.
- A client starts by creating a TCP/IP connection to the broker by using either a standard port or a custom port defined by the broker's operators.
- During the communication phase, a client can perform publish, subscribe, unsubscribe and ping operations.
- When creating the connection, it is important to recognize that the server might continue an old (persistent) session if it is provided with a reused client identity.

Encryption and Authentication

- The standard ports are 1883 for non-encrypted communication and 8883 for encrypted communication.
  - using Secure Sockets Layer (SSL) / Transport Layer Security (TLS).
- During the SSL/TLS handshake, the client validates the server certificate and authenticates the server.
- The client may also provide a client certificate to the broker during the handshake. The broker can use this to authenticate the client.

MQTT Control Packets

Control Packet	Direction of Packet Flow	Description
CONNECT	Client to Broker	Connection request
CONNACK	Broker to Client	Connect acknowledgment
SUBSCRIBE	Client to Broker	Subscribe request
SUBACK	Broker to Client	Subscribe acknowledgment
UNSUBSCRIBE	Client to Broker	Unsubscribe request
UNSUBACK	Broker to Client	Unsubscribe acknowledgment
PINGREQ	Client to Broker	PING request
PINGRESP	Broker to Client	PING response

MQTT Control Packets

Control Packet	Direction of Packet Flow	Description
DISCONNECT	Bidirectional	Disconnect notification
PUBLISH	Bidirectional	Publish message
PUBACK	Bidirectional	Publish acknowledgment (QoS 1)
PUBREC	Bidirectional	Publish received (QoS 2)
PUBREL	Bidirectional	Publish released (QoS 2)
PUBCOMP	Bidirectional	Publish complete (QoS 2)
AUTH	Bidirectional	Authentication

MQTT Messages

- Each MQTT message consists of a fixed header (2 bytes), an optional variable header, a message payload that is limited to 256 megabytes of data (called *Binary Large Object* or BLOB) and a QoS level.

MQTT Packet Format

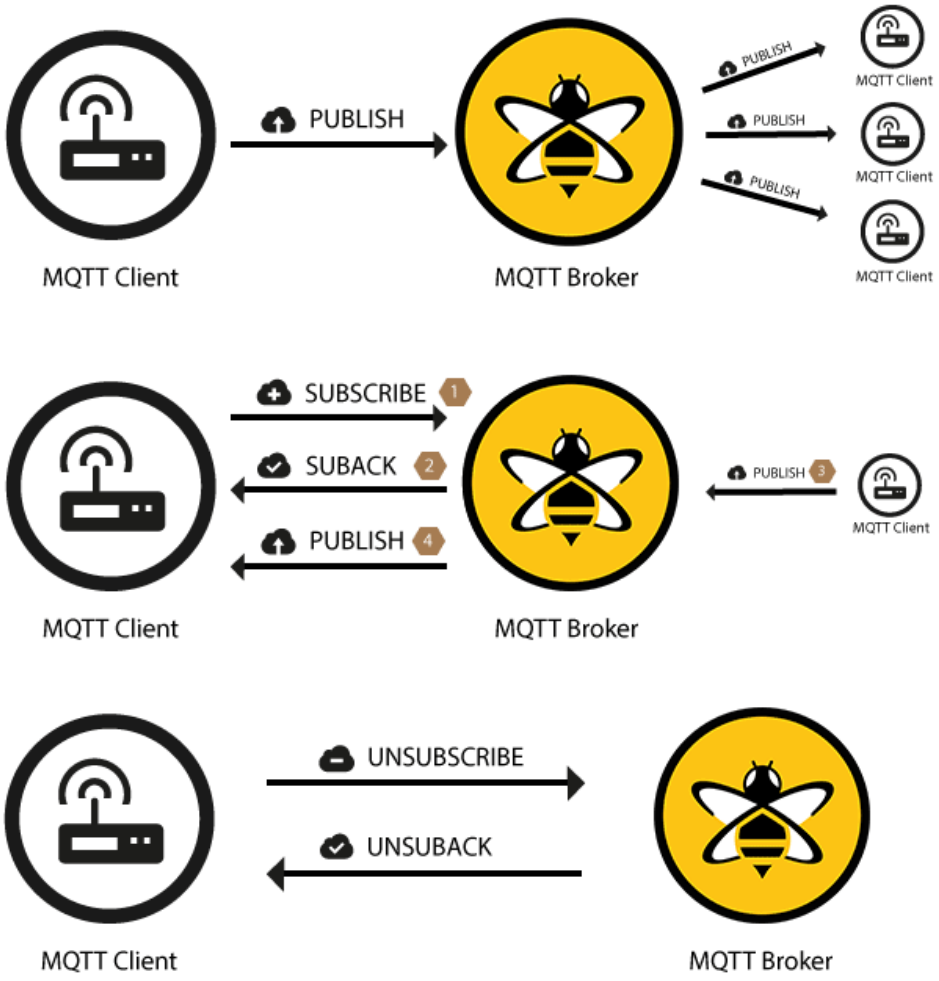
Fixed header
Variable header
Payload

Fixed Header

Bit	7	6	5	4	3	2	1	0
byte1	MQTT Packet type				Flags			
byte2...	Remaining Length							

Topic-based Message Routing

- Topic is a hierarchical structured string, like:
  - `chat/room/1`
  - `sensor/10/temperature`
  - `sensor/+/temperature`
  - `$SYS/broker/metrics/#`
- A forward slash (`/`) is used to separate levels within a topic tree and provide a hierarchical structure to the topic space.
- The number sign (#) is a wildcard for multi-level in a topic.
- The plus sign (+) is a wildcard for single-level.



MQTT publication and subscription (Image source: HiveMQ)

Example of an MQTT connection (QoS 0) with connect, publish / subscribe, and disconnect.

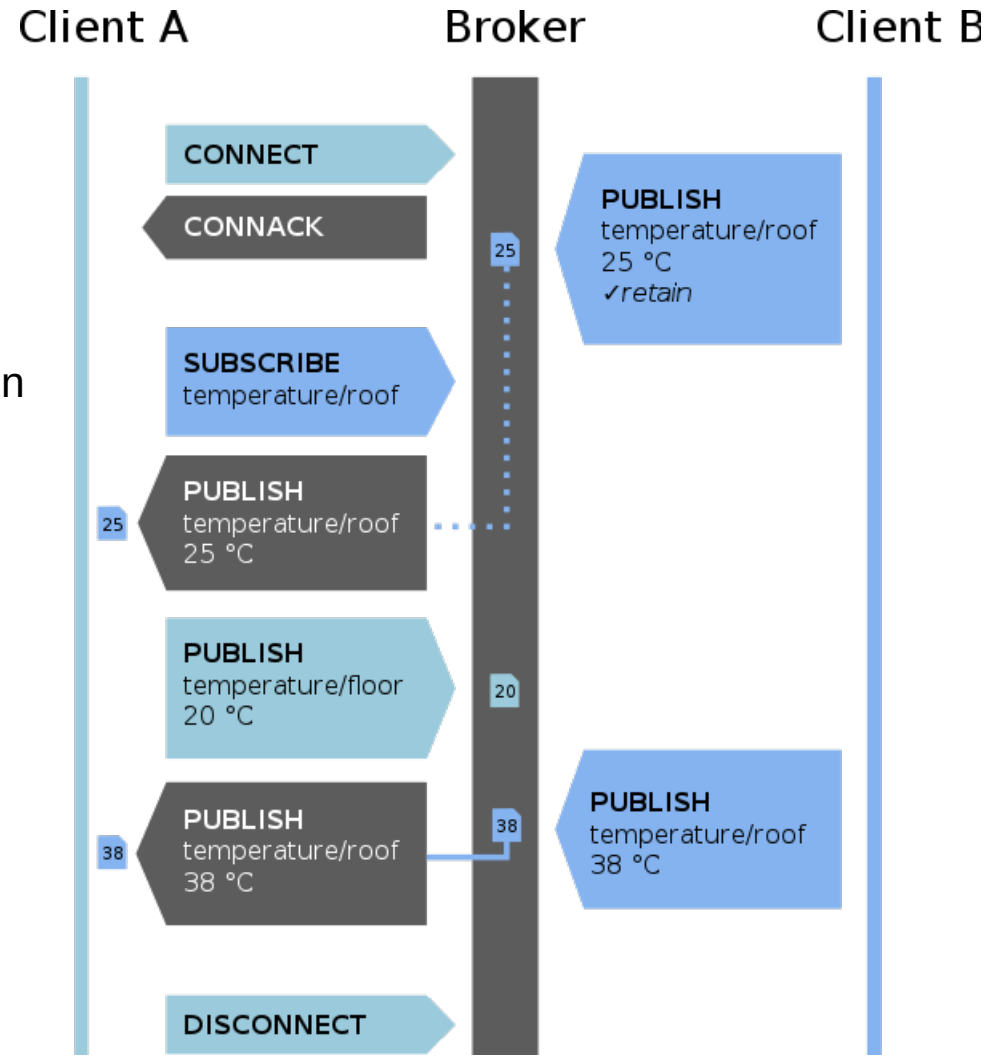


Image source; https://en.wikipedia.org/wiki/MQTT

QoS 0

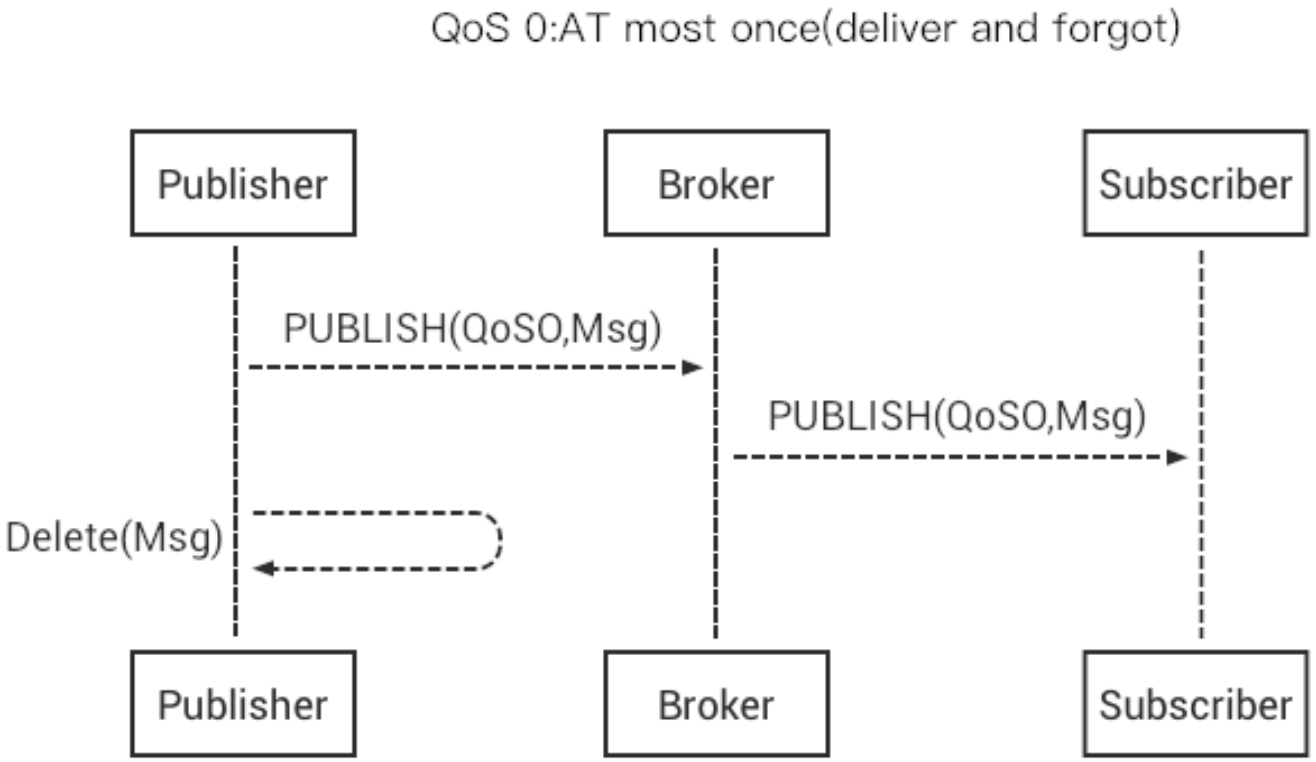


Image source: https://emqx-enterprise-docs-en.readthedocs.io/en/latest/mqtt.html

QoS 1

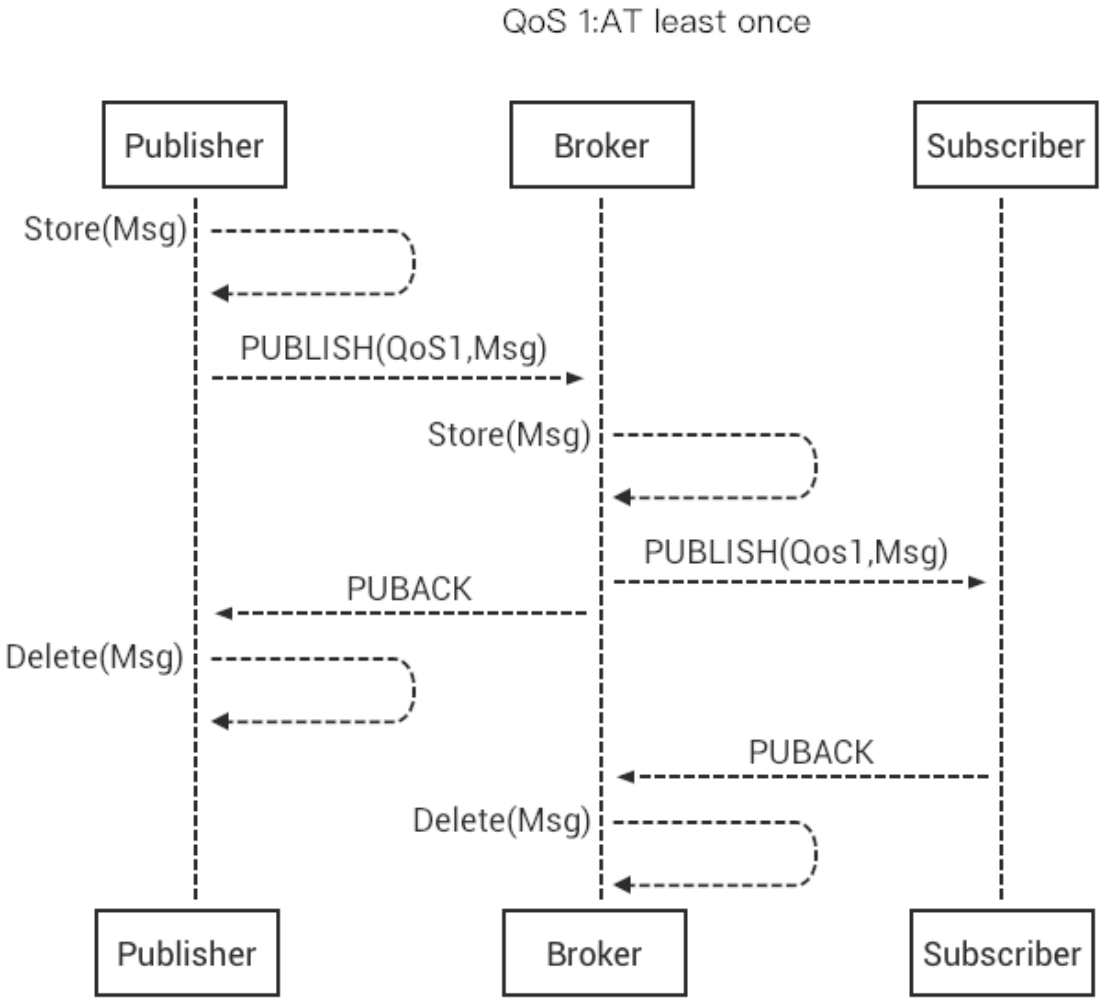


Image source: https://emqx-enterprise-docs-en.readthedocs.io/en/latest/mqtt.html

QoS 2

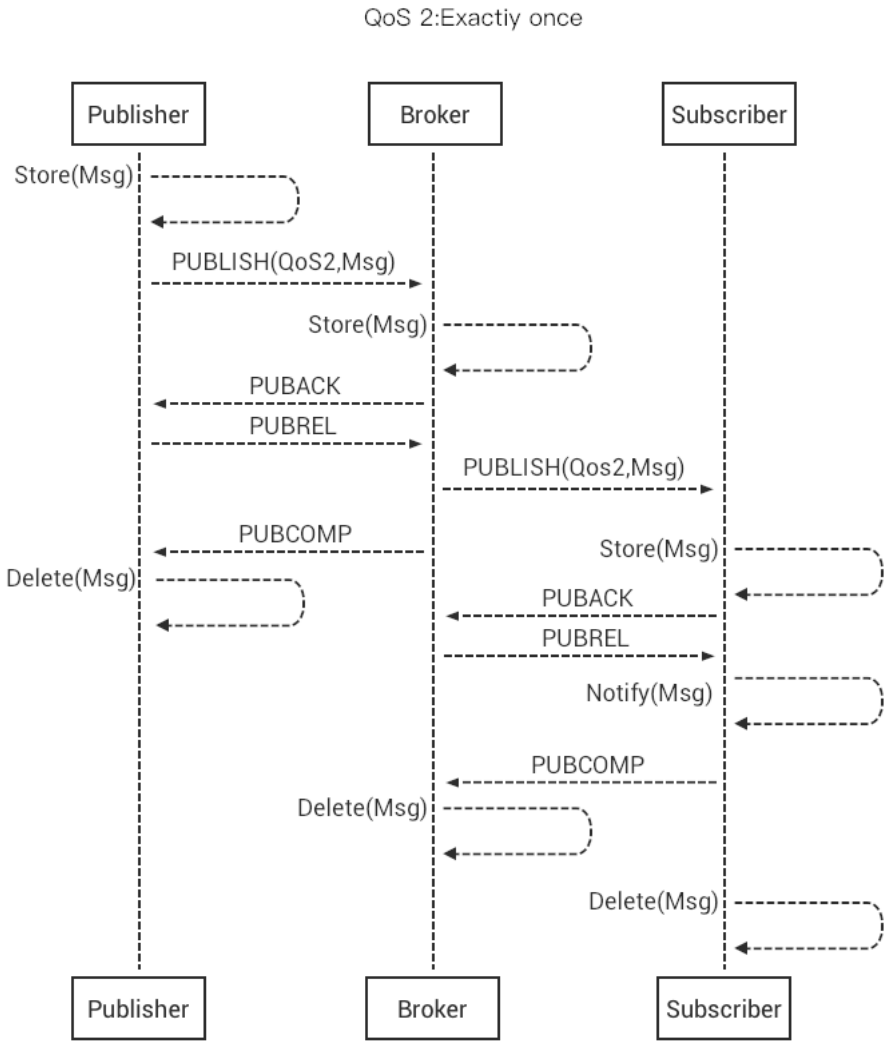


Image source: https://emqx-enterprise-docs-en.readthedocs.io/en/latest/mqtt.html

MQTT Ports

- Plain MQTT (default port: 1883)
- Plain MQTT with client authentication
  - Username / password protected
- MQTT over TLS (default port: 8883)
- MQTT over TLS with client certificate
- MQTT over WebSockets (default port: 9001)
- MQTT over WebSockets with TLS

TLS = Transport Layer Security

Mosquitto Servers for Testing

- **1883** : MQTT, unencrypted, unauthenticated
- **1884** : MQTT, unencrypted, authenticated
- **8883** : MQTT, encrypted, unauthenticated
- **8884** : MQTT, encrypted, client certificate required
- **8885** : MQTT, encrypted, authenticated
- **8886** : MQTT, encrypted, unauthenticated
- **8887** : MQTT, encrypted, server certificate deliberately expired
- **8080** : MQTT over WebSockets, unencrypted, unauthenticated
- **8081** : MQTT over WebSockets, encrypted, unauthenticated
- **8090** : MQTT over WebSockets, unencrypted, authenticated
- **8091** : MQTT over WebSockets, encrypted, authenticated

Source: https://test.mosquitto.org/

Installation of Mosquitto (MQTT) client package on Ubuntu:

```
$ sudo apt update
$ sudo apt install mosquitto-clients -y
```

Show the version of the Mosquitto client commands:

```
$ mosquitto_pub --version
mosquitto_pub version 2.0.11 running on libmosquitto 2.0.11.

$ mosquitto_sub --version
mosquitto_sub version 2.0.11 running on libmosquitto 2.0.11.
```

Subscribe messages for a topic at test.mosquitto.org using port 1883.

```
# command to subscribe to a specific topic
$ mosquitto_sub -h test.mosquitto.org -p 1883 -t 'test/1234/#'
```

Publish a message to test.mosquitto.org using port 1883.

```
# command to publish a message to a specific topic
$ mosquitto_pub -h test.mosquitto.org -p 1883 -t 'test/1234/msg' -m 'hello'
```

Run commands with the -d option

```
$ mosquitto_pub -h test.mosquitto.org -p 1883 -t 'test/1234/msg' -m 'hello' -d
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q0, r0, m1, 'test/1234/msg', ... (5 bytes))
Client (null) sending DISCONNECT
```

Online Client Certificate Generator

test.mosquitto.org/ssl/ https://test.mosquitto.org/ssl/

mosquitto

Generate a TLS client certificate for test.mosquitto.org

This page allows you to generate an x509 certificate suitable that will allow you to connect to the TLS enabled ports on test.mosquitto.org that require a client certificate, i.e. port 8884.

To use it, you will need to generate a PEM encoded Certificate Signing Request (CSR) and paste it into the form. After you submit the form, the certificate will be generated for you to download. The certificates are valid for 90 days.

Generate a CSR using the openssl utility

Generate a private key:

```
openssl genrsa -out client.key
```

Generate the CSR:

```
openssl req -out client.csr -key client.key -new
```

When you are generating the CSR, please do not use the default

Paste your CSR here


```
-----BEGIN CERTIFICATE REQUEST-----
MIIClDCCAXwCAQAwTzELMAkGA1UEBhMCVEgxEDAOBgNVBAgMB0Jhbmdrb2sxDzAN
BgNVBAoMBktNVVROQjEdMBsGA1UEAwwUaW90LWttdXRuYi5naXRodWIuaW8wggEi
MA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQC43vVrG15SID+j7CbyKGwfOP2g
T7kAcN+VSrGG23eI+6cDqxeAVoWU4pY9J9mbVl2fBwO4hQTa9fDRNw7FappzN8OC
a4zqsyR0OuXvaLRFOmHXyKZvez2KGJCpNtChPW9mPbC2mBTrUrnWynXeusbUh2cZ
Sfu9hEAk3cLG3yhuCW8Ma8ufaBelZGjweAZFDAiT4lmuJq0ehp6l81/6XXkRxLFz
IL6QL2Cpg/plJSOm7Z37xCop91lf2qNBatRUtVQa/ma2Jqr1oEPIckkJMxMAynrx
ksaH/ebdoAa+G3E71GlPQhzG8K1ccRwGkp9MbpxhqrTH2+zuYDk8KePNjITVAgMB
AAGgADANBgkqhkiG9w0BAQsFAAOCAQEAoO+E1vsknhpHHxnFdSVCsWvKtS5tjbU6
G5FA607HdPSTfXQls8pI/w/pgXKJ0/J0w3Vs0W6dUYXlz3JsA2HC1ARv7zpfih10
e9YaU8H5II90M3BlxnCJHjTZVK0ZNk84U4Ft8FjqR1GfNj/BIskfRKpktz7LADr7
NI5Qtkjdjly14xI0R0XiVwr3ifgZme7iYMV4rTx6ckj/MBHGz0nmgGRJUGuv8/3T
s0TlqwHOAddY6EhG954kF07cbW16mAiQbiMeS07Wnm6CaEEDT85BA32Xbzk7Wipx
OkRwuTQv7L0sZM/K6L2Ky6SiHw2kHD9aCZ2AatG+UtL+QKgvu3a/QQ==
```


Submit

Donate

Subscribe messages for a topic at test.mosquitto.org using port 8883 or 8884.

```
$ mosquitto_sub -h test.mosquitto.org -p 8883 -d \
   --cafile mosquitto.org.crt -t 'test/1234/#' -V mqttv5 -q 2
```

```
$ mosquitto_sub -h test.mosquitto.org -p 8884 -d \
 --cafile mosquitto.org.crt --cert client.crt --key client.key \
 -t 'test/1234/#' -V mqttv5 -q 2
```

Publish a message to test.mosquitto.org using port 8884.

```
$ mosquitto_pub -h test.mosquitto.org -p 8884 -d \
  --cafile mosquitto.org.crt --cert client.crt --key client.key \
  -t 'test/1234/msg' -m 'hello' -V mqttv5 -q 2
```

ubuntu@LENOVO-LAPTOP: ~/MQTT

```
ubuntu@LENOVO-LAPTOP:~/MQTT$ mosquitto_sub -h test.mosquitto.org -p 8884 -d \
>  --cafile mosquitto.org.crt --cert client.crt --key client.key \
>  -t 'test/1234/#' -V mqttv5 -q 2
Client (null) sending CONNECT
Client auto-EAB391AB-93F7-C4C6-863C-02BF2F0B4E83 received CONNACK (0)
Client auto-EAB391AB-93F7-C4C6-863C-02BF2F0B4E83 sending SUBSCRIBE (Mid: 1, Topic: test/1234/#, QoS: 2, Op
tions: 0x00)
Client auto-EAB391AB-93F7-C4C6-863C-02BF2F0B4E83 received SUBACK
Subscribed (mid: 1): 2
Client auto-EAB391AB-93F7-C4C6-863C-02BF2F0B4E83 received PUBLISH (d0, q2, r0, m1, 'test/1234/msg', ... (5
 bytes))
Client auto-EAB391AB-93F7-C4C6-863C-02BF2F0B4E83 sending PUBREC (m1, rc0)
Client auto-EAB391AB-93F7-C4C6-863C-02BF2F0B4E83 received PUBREL (Mid: 1)
Client auto-EAB391AB-93F7-C4C6-863C-02BF2F0B4E83 sending PUBCOMP (m1)
hello
```

```
ubuntu@LENOVO-LAPTOP:~/MQTT$ mosquitto_pub -h test.mosquitto.org -p 8884 -d \
>   --cafile mosquitto.org.crt --cert client.crt --key client.key \
>   -t 'test/1234/msg' -m 'hello' -V mqttv5 -q 2
Client (null) sending CONNECT
Client auto-72971754-98AA-CAF4-6A4B-FA1AD75C1DC0 received CONNACK (0)
Client auto-72971754-98AA-CAF4-6A4B-FA1AD75C1DC0 sending PUBLISH (d0, q2, r0, m1, 'test/1234/msg', ... (5
bytes))
Client auto-72971754-98AA-CAF4-6A4B-FA1AD75C1DC0 received PUBREC (Mid: 1)
Client auto-72971754-98AA-CAF4-6A4B-FA1AD75C1DC0 sending PUBREL (m1)
Client auto-72971754-98AA-CAF4-6A4B-FA1AD75C1DC0 received PUBCOMP (Mid: 1, RC:0)
Client auto-72971754-98AA-CAF4-6A4B-FA1AD75C1DC0 sending DISCONNECT
ubuntu@LENOVO-LAPTOP:~/MQTT$
```

[0] 0:bash* "LENOVO-LAPTOP" 00:23 12-Sep-22

Subscribe messages for a topic at broker.emqx.io using port 8883.

```
$ mosquitto_sub -h broker.emqx.io -p 8883 -d \
 --capath /etc/ssl/certs/ -t 'test/1234/#' -V mqttv5 -q 2
```

Publish a message to broker.emqx.io using port 8883.

```
$ mosquitto_pub -h broker.emqx.io -p 8883 -d \
  --capath /etc/ssl/certs/ -t 'test/1234/msg' -m 'hello' -V mqttv5 -q 2
```

Publish a message to test.mosquitto.org using port 8885.

```
$ mosquitto_pub -h test.mosquitto.org -p 8885 -d \
 --cafile mosquitto.org.crt --cert client.crt --key client.key \
 -t 'test/1234/msg' -m 'hello' -V mqttv5 -q 2 \
 -u 'rw' -P 'readwrite'
```

ubuntu@LENOVO-LAPTOP: ~/MQTT

```
>  --capath /etc/ssl/certs/ -t 'test/1234/#' -V mqttv5 -q 2
Client (null) sending CONNECT
Client MzA2ODQzODYwOTE0MTI2MzYxMDU4MzY3NTI2OTEyMDAwMDA received CONNACK (0)
Client MzA2ODQzODYwOTE0MTI2MzYxMDU4MzY3NTI2OTEyMDAwMDA sending SUBSCRIBE (Mid: 1, Topic: test/1234/#, QoS: 2, Options: 0x00)
Client MzA2ODQzODYwOTE0MTI2MzYxMDU4MzY3NTI2OTEyMDAwMDA received SUBACK
Subscribed (mid: 1): 2
Client MzA2ODQzODYwOTE0MTI2MzYxMDU4MzY3NTI2OTEyMDAwMDA received PUBLISH (d0, q2, r0, m1, 'test/1234/msg', ... (5 bytes))
Client MzA2ODQzODYwOTE0MTI2MzYxMDU4MzY3NTI2OTEyMDAwMDA sending PUBREC (m1, rc0)
Client MzA2ODQzODYwOTE0MTI2MzYxMDU4MzY3NTI2OTEyMDAwMDA received PUBREL (Mid: 1)
Client MzA2ODQzODYwOTE0MTI2MzYxMDU4MzY3NTI2OTEyMDAwMDA sending PUBCOMP (m1)
hello
```

```
ubuntu@LENOVO-LAPTOP:~/MQTT$ mosquitto_pub -h broker.emqx.io -p 8883 -d \
>   --capath /etc/ssl/certs/ -t 'test/1234/msg' -m 'hello' -V mqttv5 -q 2
Client (null) sending CONNECT
Client MzA2ODQzODYyODY1ODM4MDAwOTE3MDMzMzQwMjg3NzEzMjI received CONNACK (0)
Client MzA2ODQzODYyODY1ODM4MDAwOTE3MDMzMzQwMjg3NzEzMjI sending PUBLISH (d0, q2, r0, m1, 'test/1234/msg', ... (5 bytes))
Client MzA2ODQzODYyODY1ODM4MDAwOTE3MDMzMzQwMjg3NzEzMjI received PUBREC (Mid: 1)
Client MzA2ODQzODYyODY1ODM4MDAwOTE3MDMzMzQwMjg3NzEzMjI sending PUBREL (m1)
Client MzA2ODQzODYyODY1ODM4MDAwOTE3MDMzMzQwMjg3NzEzMjI received PUBCOMP (Mid: 1, RC:0)
Client MzA2ODQzODYyODY1ODM4MDAwOTE3MDMzMzQwMjg3NzEzMjI sending DISCONNECT
ubuntu@LENOVO-LAPTOP:~/MQTT$
```

[0] 0:bash* "LENOVO-LAPTOP" 15:38 17-Sep-22

Running Mosquitto MQTT broker Under Docker

- Mosquitto is an open-source message broker that implements the MQTT protocol.
- It is widely used for publish / subscribe messaging in a variety of applications.
- Mosquitto supports MQTT protocol v3.1/3.1.1 and 5.0.
- An official Eclipse Mosquitto Docker image is available on Docker Hub.

```
# Pull the latest Docker image for Eclipse Mosquito.
$ docker pull eclipse-mosquitto:latest
# Create a local Mosquitto directory.
$ mkdir -p ~/.mosquitto
# Create and edit the local configuration file.
$ nano ~/.mosquitto/mosquitto.conf
```

```
allow_anonymous true
listener 1883
listener 9001
protocol websockets

persistence true
persistence_location /mosquitto/data/
log_dest file /mosquitto/log/mosquitto.log
```

```
# Create and start a new container (named 'mosquitto')
# to run eclipse-mosquitto in detached mode.
$ docker run -d -p 1883:1883 -p 9001:9001 \
  --name="mosquitto" \
  -v ~/.mosquitto/mosquitto.conf:/mosquitto/config/mosquitto.conf \
  -v /mosquitto/data \
  -v /mosquitto/log \
  eclipse-mosquitto:latest
```

Note: There are 3 directories used for Mosquitto configuration, persistent storage and logs.

- `/mosquitto/config`
- `/mosquitto/data`
- `/mosquitto/log`

How to use Docker Compose for Mosquitto
see: https://github.com/sukesh-ak/setup-mosquitto-with-docker or https://cedalo.com/blog/mosquitto-docker-configuration-ultimate-guide/

Running Mosquitto MQTT Client Under Docker

```
# Create and run a container from the eclipse-mosquitto image
# and run the mosquitto_pub command inside the container.
# Remove the container when it exits.
$ docker run -it --rm eclipse-mosquitto \
  mosquitto_pub -d -h raspberrypi -p 1883 \
  -t test/topic -m "Hello Mosquitto!"
```

wokwi.com/projects/343045226095444562

WOKWI SAVE SHARE Docs

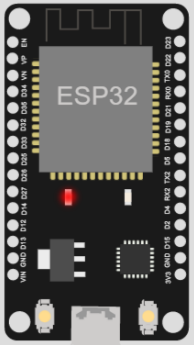
esp32_mqtt_client_demo.ino | diagram.json | libraries.txt | arduino_secrets.h

Library Manager

```
#include <WiFiClientSecure.h>  // changed from <WiFi.h>
#include <PubSubClient.h>
#include "arduino_secrets.h"

#define MQTT_BROKER    "test.mosquitto.org"
#define MQTT_PORT      (8883)

#define CLIENT_ID      "arduino_client"
#define SUB_TOPIC      "test/1234/#"
#define PUB_TOPIC      "test/1234/msg"
#define INTERVAL_MSEC  (5000)

WiFiClientSecure net;  // ESP32 WiFi client (Secure)
PubSubClient client(net);   // MQTT client
uint32_t last_pub_ts_msec = 0;

void connect() {
  // connect the WiFi network first (if not already connected)
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
  }
  Serial.print( "\n\nWiFi Connected: ");
  Serial.println( WiFi.localIP() ); // show the IP address

  // connect/reconnect the MQTT broker
  while ( !client.connect(CLIENT_ID, MQTT_USER, MQTT_PASS) ) {
    delay(1000);
  }
```

Simulation

00:21.632 56%



Arduino-ESP32 Simulation using WokWi Simulator

```
WiFi Connected: 10.10.0.2
Published: 'hello id=1'
Received: topic='test/1234/msg', payload='hello id=1', rtt=232 msec
Published: 'hello id=2'
Received: topic='test/1234/msg', payload='hello id=2', rtt=322 msec
Published: 'hello id=3'
Received: topic='test/1234/msg', payload='hello id=3', rtt=188 msec
```

esp32_mqtt_client_demo_btn.ino - Wokwi A

wokwi.com/projects/343132972237455956

WOKWI SAVE SHARE Docs

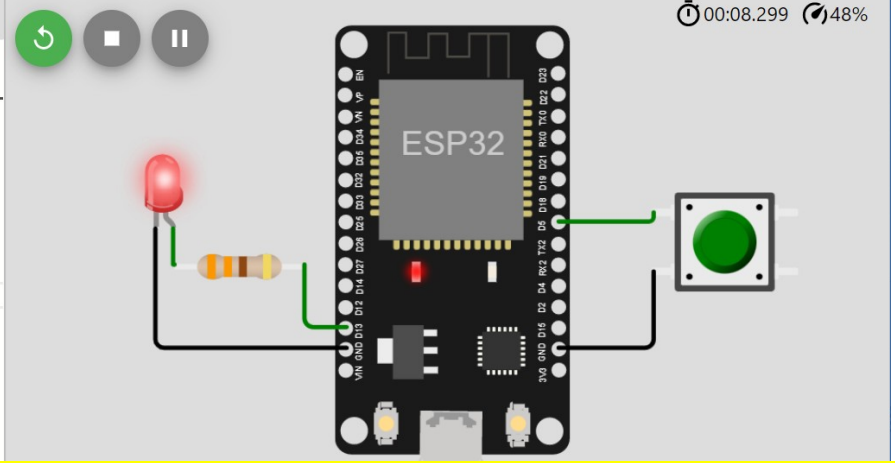
esp32_mqtt_client_demo.ino ● diagram.json libraries.txt arduino_secrets.h

Library Manager

```
#include <WiFi.h>
#include <WiFiClient.h>
#include <PubSubClient.h> // https://github.com/knolleary/pubsubclient/
#include "arduino_secrets.h"

#define BTN_PIN        (5)
#define LED_PIN        (13)

#define MQTT_BROKER    "test.mosquitto.org"
#define MQTT_PORT      (1883)
#define CLIENT_ID      "arduino_client"
#define SUB_TOPIC      "test/1234/#"
#define PUB_TOPIC      "test/1234/msg"

WiFiClient net;            // ESP32 WiFi client
PubSubClient client(net);  // MQTT client

void connect() {
  // connect the WiFi network first (if not already connected)
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
  }
  Serial.print( "\n\nWiFi Connected: ");
  Serial.println( WiFi.localIP() ); // show the IP address
  // connect/reconnect the MQTT broker
  while ( !client.connect(CLIENT_ID, MQTT_USER, MQTT_PASS) ) {
    delay(1000);
  }
```

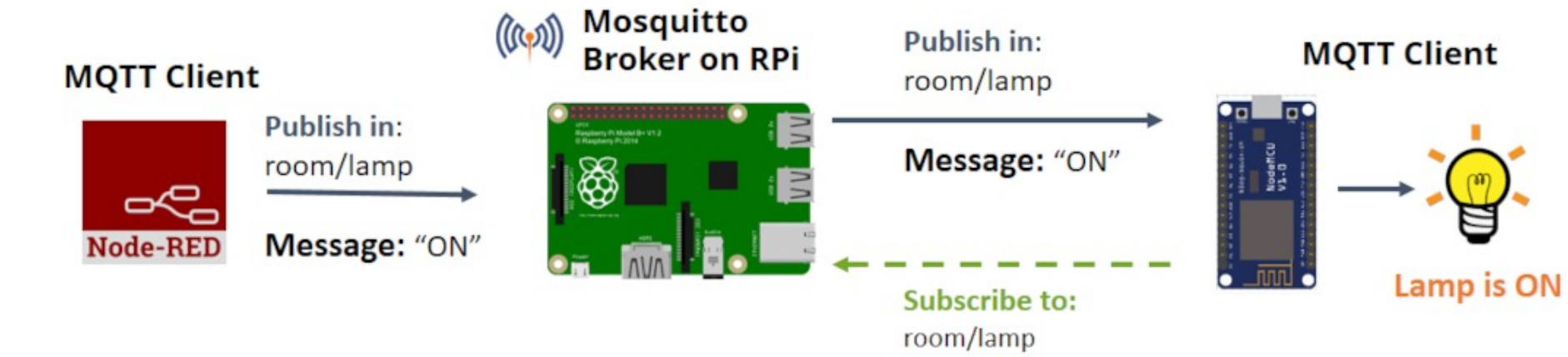
Simulation

00:08.299 48%



Arduino-ESP32 Simulation using WokWi Simulator

```
WiFi Connected: 10.10.0.2
MQTT connected.
Received: topic='test/1234/msg', payload='pressed'
Received: topic='test/1234/msg', payload='pressed'
Received: topic='test/1234/msg', payload='pressed'
```



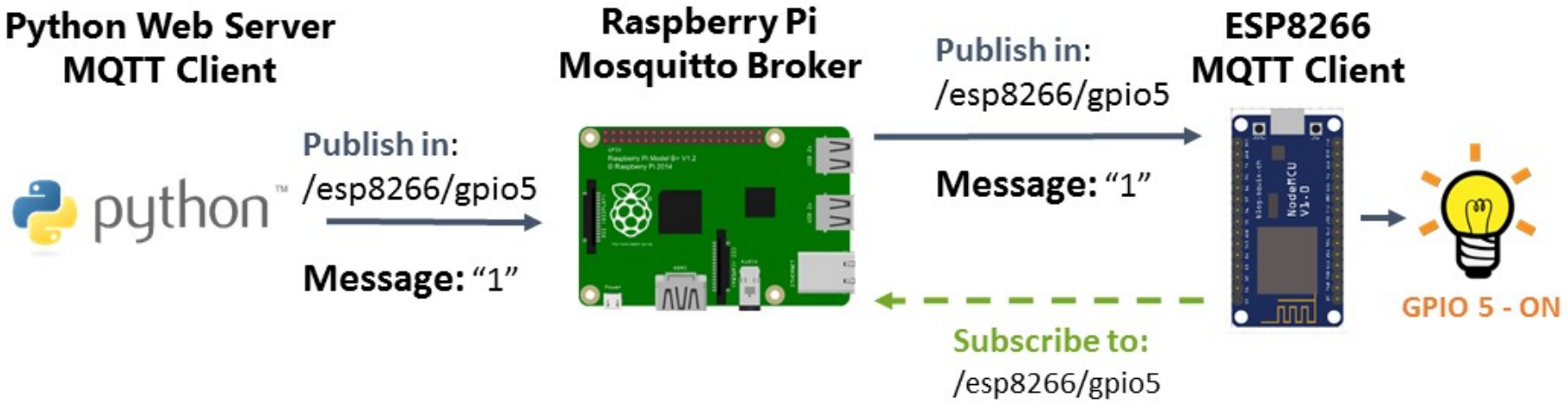
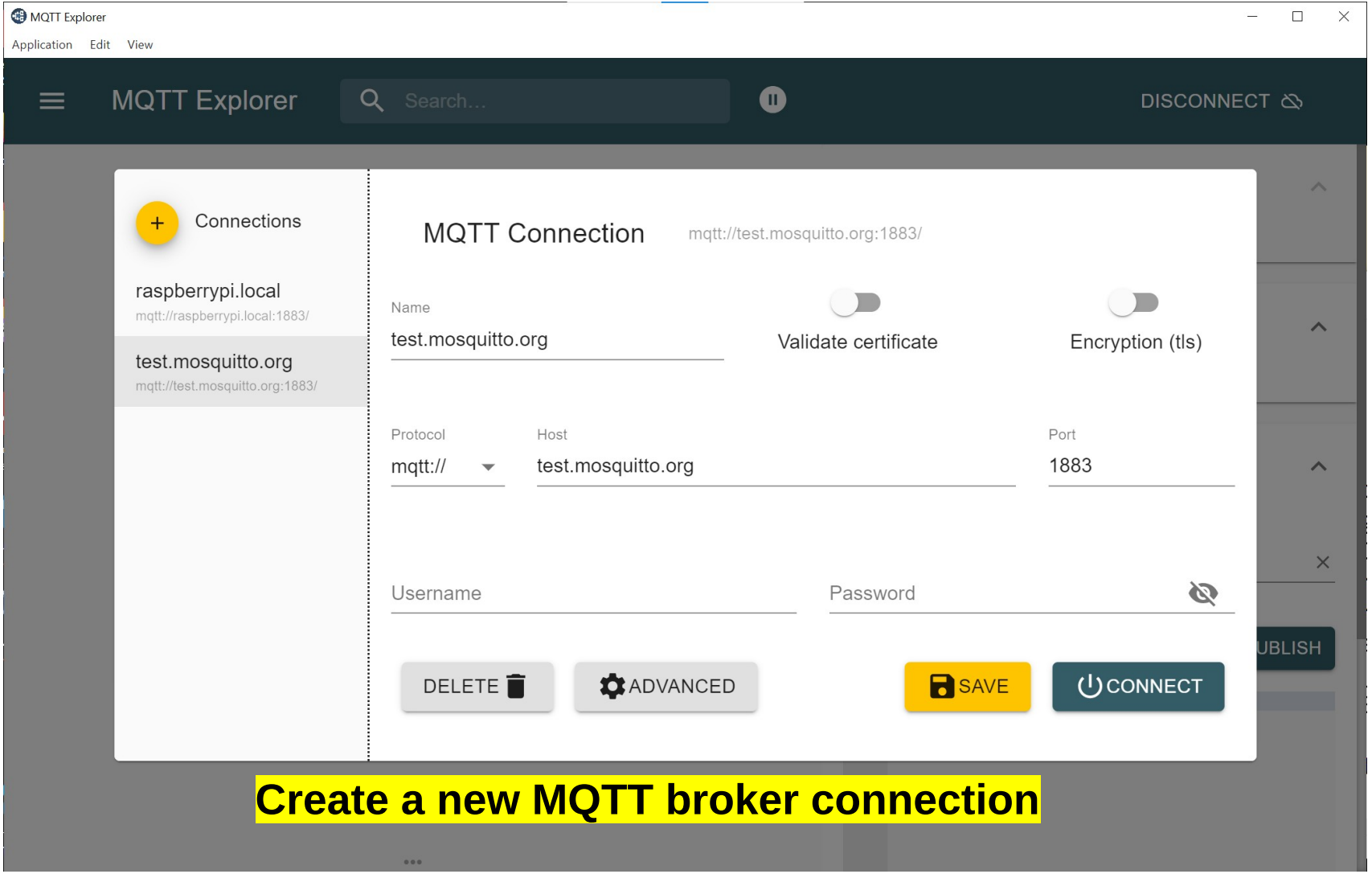
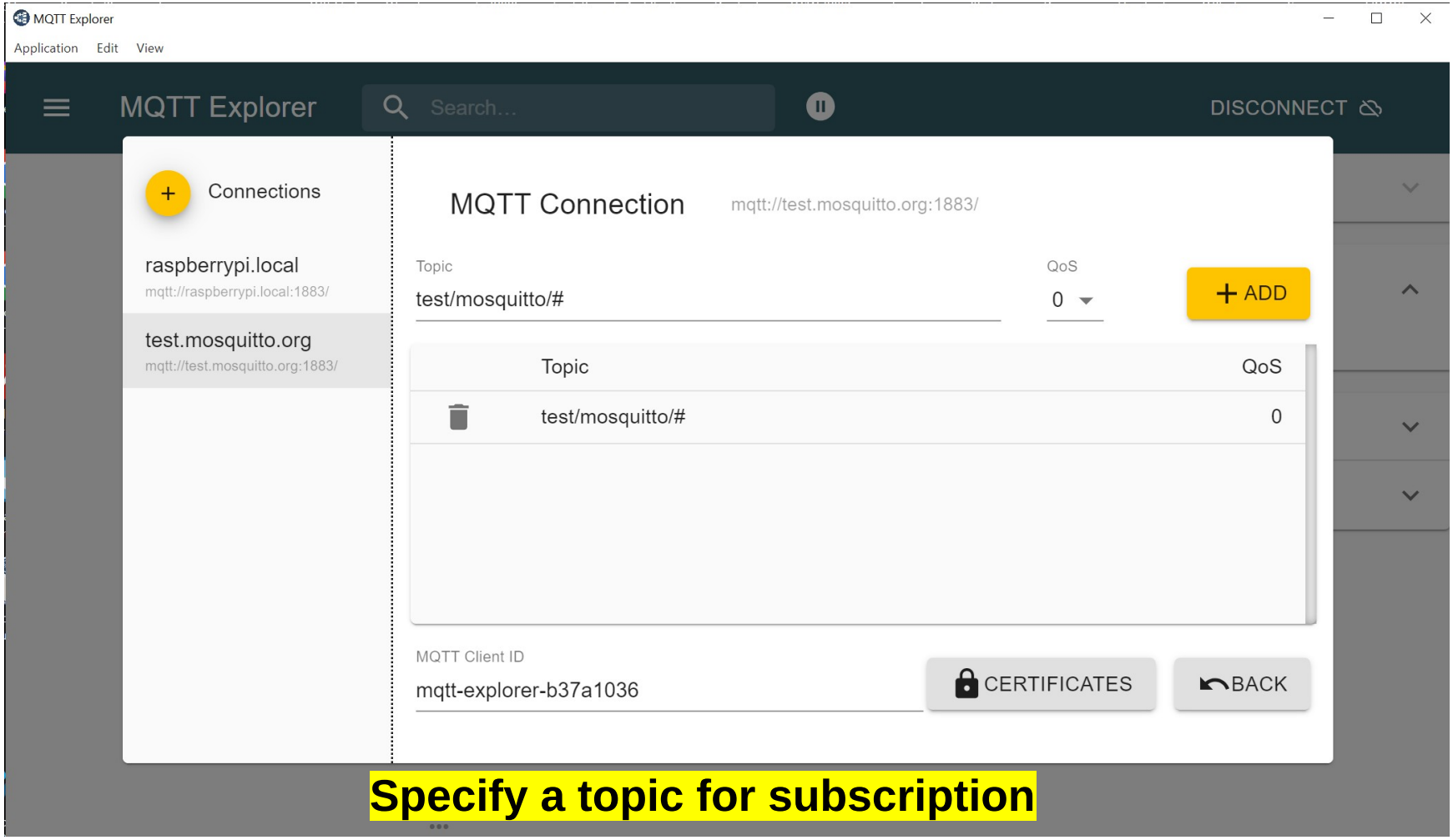


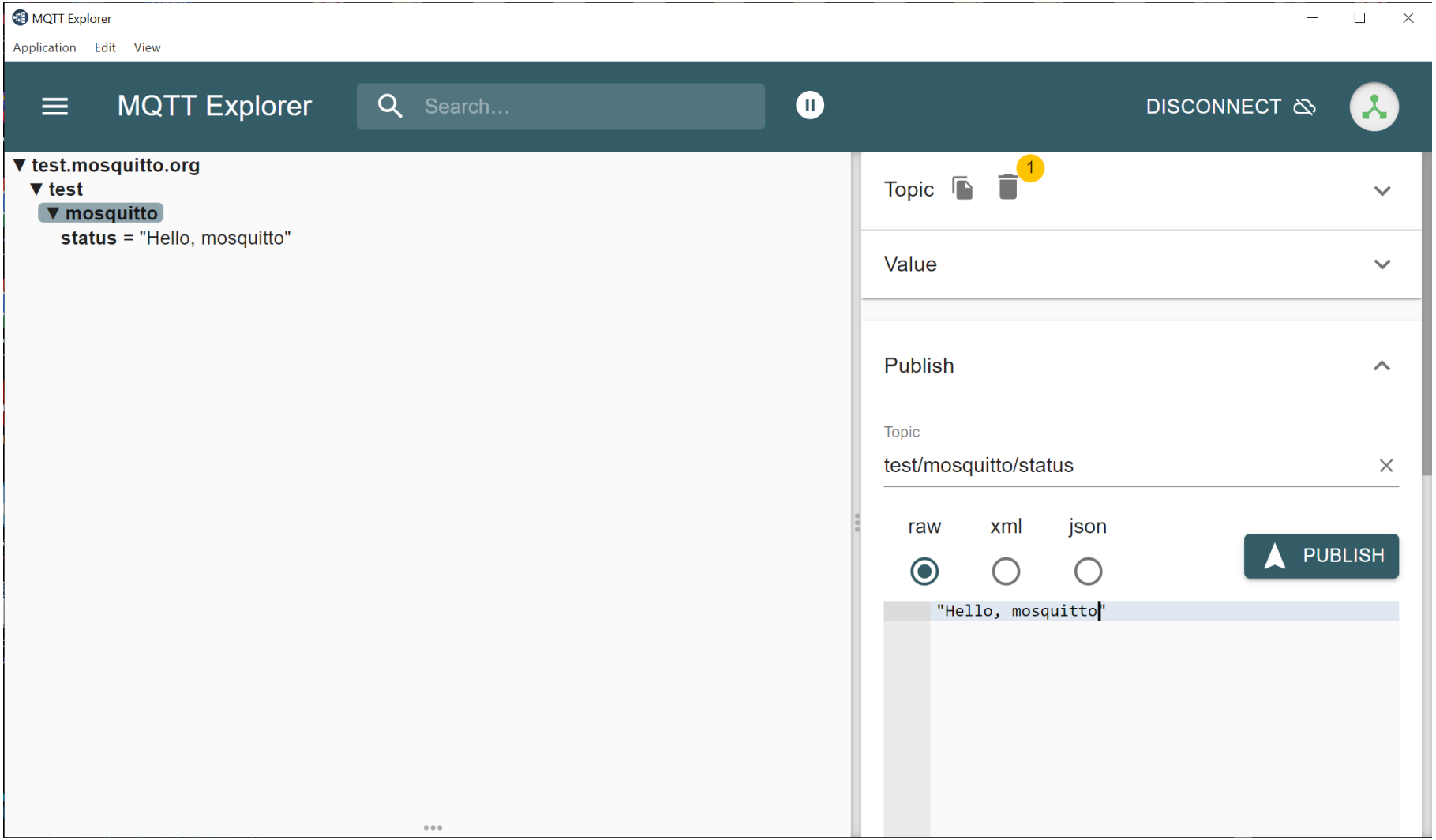
Image source: https://randomnerdtutorials.com/



Create a new MQTT broker connection

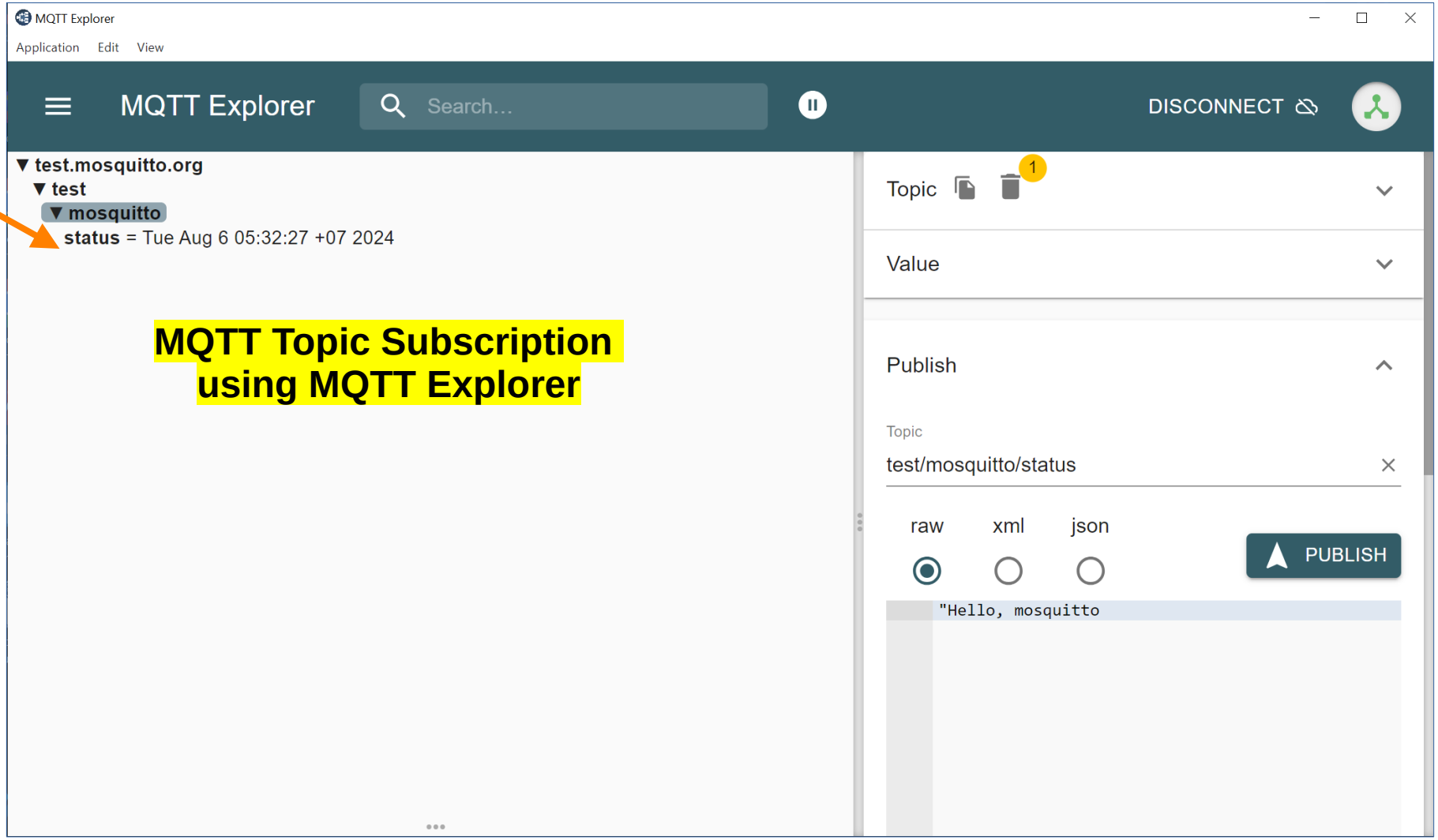


Specify a topic for subscription


MQTT Explorer
Application
Edit
View
MQTT Explorer
Search...
DISCONNECT
test.mosquitto.org
test
mosquitto
status = "Hello, mosquitto"
Topic
1
Value
Publish
Topic
test/mosquitto/status
raw
xml
json
PUBLISH
"Hello, mosquitto"

ubuntu@LENOVO-LAPTOP: ~

```
ubuntu@LENOVO-LAPTOP:~$ MESSAGE=$(date) |
>   mosquitto_pub -h test.mosquitto.org -p 1883 \
>   -t test/mosquitto/status -m "$MESSAGE" -q 2
ubuntu@LENOVO-LAPTOP:~$
ubuntu@LENOVO-LAPTOP:~$
```



MQTT Topic Subscription using MQTT Explorer

Arduino & MQTT

- Many **Arduino-compatible boards** have built-in WiFi capabilities, making them ideal for connecting to MQTT brokers.
- Examples of such boards include:
  - Espressif ESP32, ESP32-S2/S3, ESP32-C3/C6 SoC boards
  - Arduino Uno R4 and compatible boards
  - Raspberry Pi Pico W
- Arduino Libraries for MQTT clients are also available such as:
  - **Arduino PubSubClient library** (v2.8)

esp32_mqtt_demo | Arduino IDE 2.3.2

File Edit Sketch Tools Help

WEMOS LOLIN32 Lite

BOARDS MANAGER

Filter your search...

Type: All

0.0.1 REMOVE

esp32 by Espressif Systems

3.0.4 installed

Boards included in this package: Bee Motion S3, VALTRACK_V4_VTS_ESP32_C3,...

More info

3.0.4 REMOVE

esp8266 by ESP8266 Community

3.1.1 installed

Boards included in this package: DOIT ESP-Mx DevKit (ESP8285), ESPectro Core, Generic ESP82...

More info

3.1.1 REMOVE

esp32_mqtt_demo.ino arduino_secrets.h

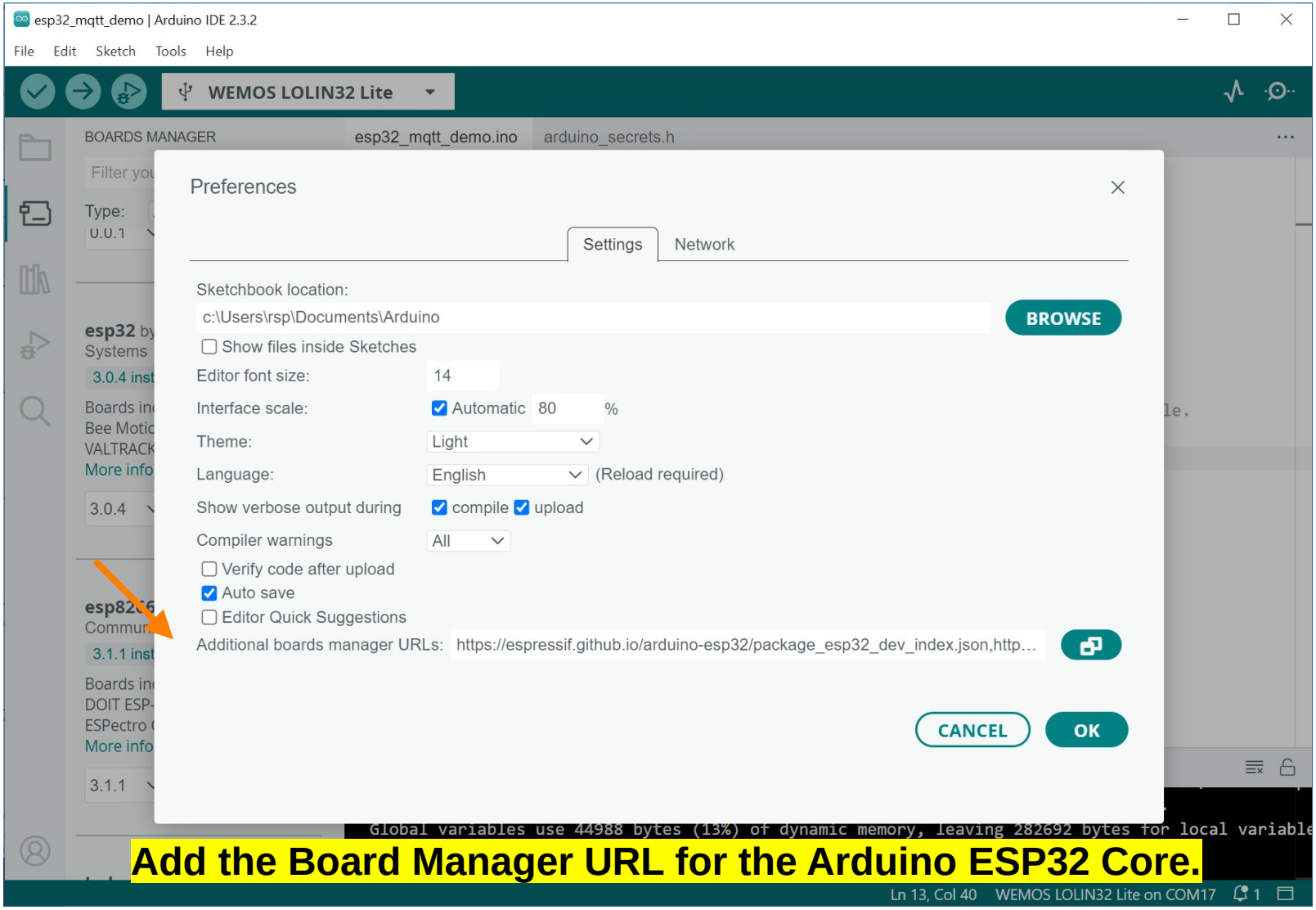
Make sure that you have installed the latest version of Arduino Core for ESP32 in the Arduino IDE 2.x.

```
// - Super-Mini ESP32C3
// - WeAct Studio ESP32-C3FH4 Mini Core
// Arduino-ESP32 Core v3.0.x
// Arduino PubSubClient library v2.8

#include <WiFi.h>
#include <PubSubClient.h>     // https://github.com/knolleary/pubsubclient/
#include "arduino_secrets.h"  // Set your WiFI SSID and password in this file.

#define WIFI_TX_POWER  WIFI_POWER_11dBm

#define MQTT_PORT      (1883) // no authentication
//#define MQTT_BROKER    "test.mosquitto.org"
#define MQTT_BROKER   "broker.hivemq.com"

#define CLIENT_ID      "ArduinoESP32Client"
#define SUB_TOPIC      "test/mosquitto/#"
#define PUB_TOPIC      "test/mosquitto/status"
#define INTERVAL_MSEC  (5000)

WiFiClient net;            // ESP32 WiFi client
PubSubClient client(net);  // MQTT client
```

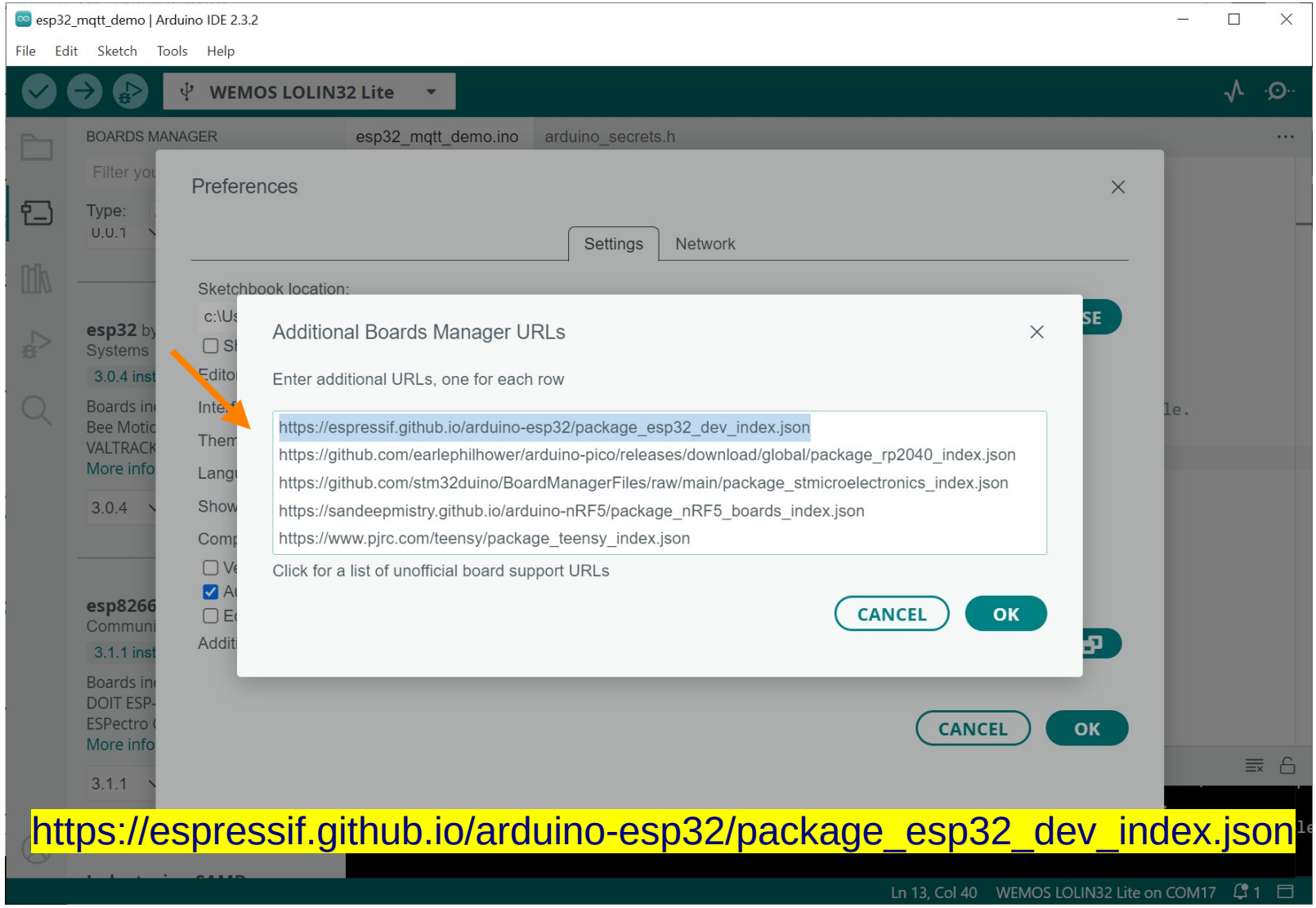
Output

```
Sketch uses 921269 bytes (70%) of program storage space. Maximum is 1310720 bytes.
Global variables use 44988 bytes (13%) of dynamic memory, leaving 282692 bytes for local variable
```

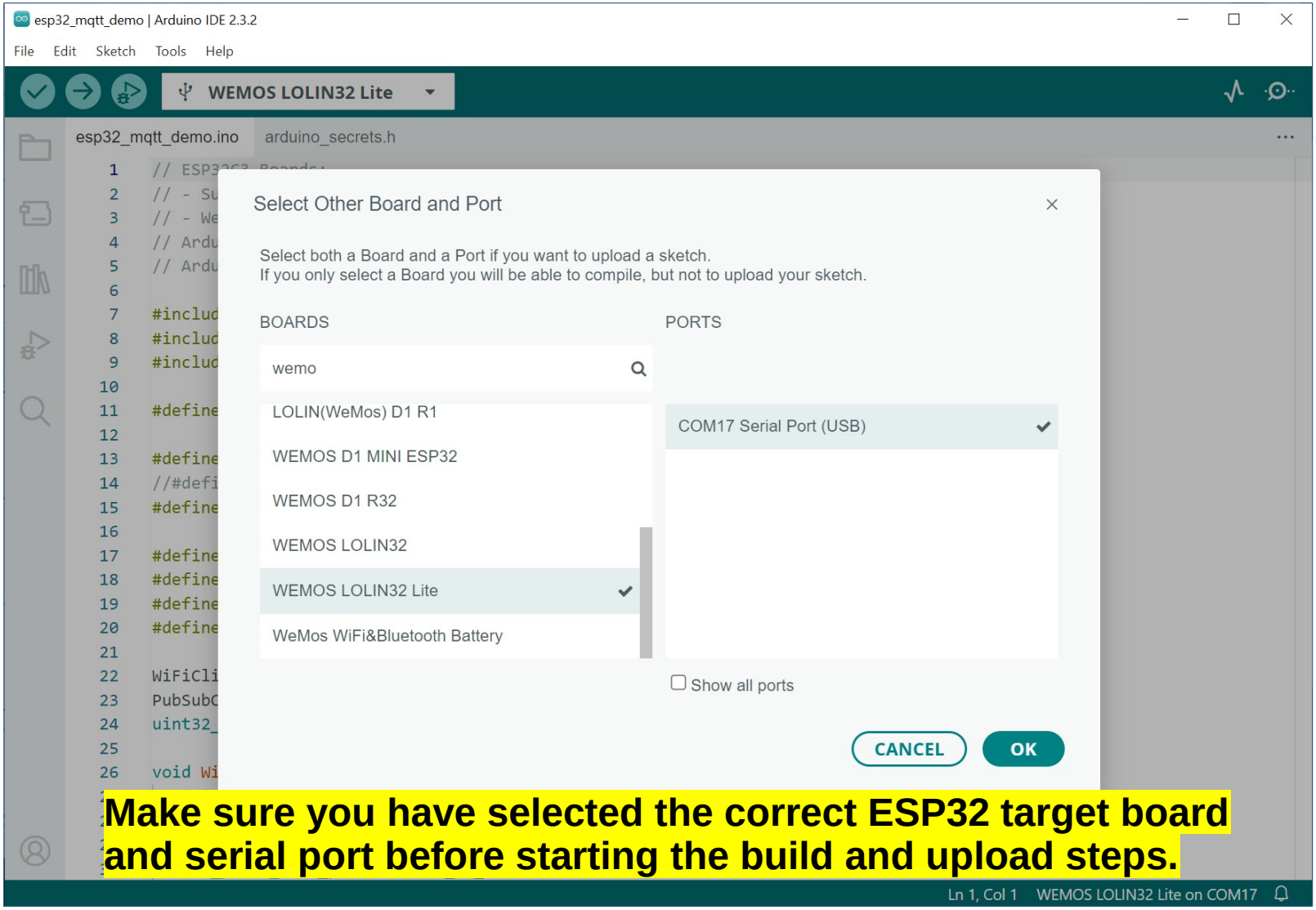
Ln 8, Col 1 WEMOS LOLIN32 Lite on COM17 1



Add the Board Manager URL for the Arduino ESP32 Core.



https://espressif.github.io/arduino-esp32/package_esp32_dev_index.json



Make sure you have selected the correct ESP32 target board and serial port before starting the build and upload steps.

Arduino-ESP32-MQTT-Client Demo

esp32_mqtt_demo | Arduino IDE 2.3.2

File Edit Sketch Tools Help

WEMOS LOLIN32 Lite

esp32_mqtt_demo.ino arduino_secrets.h

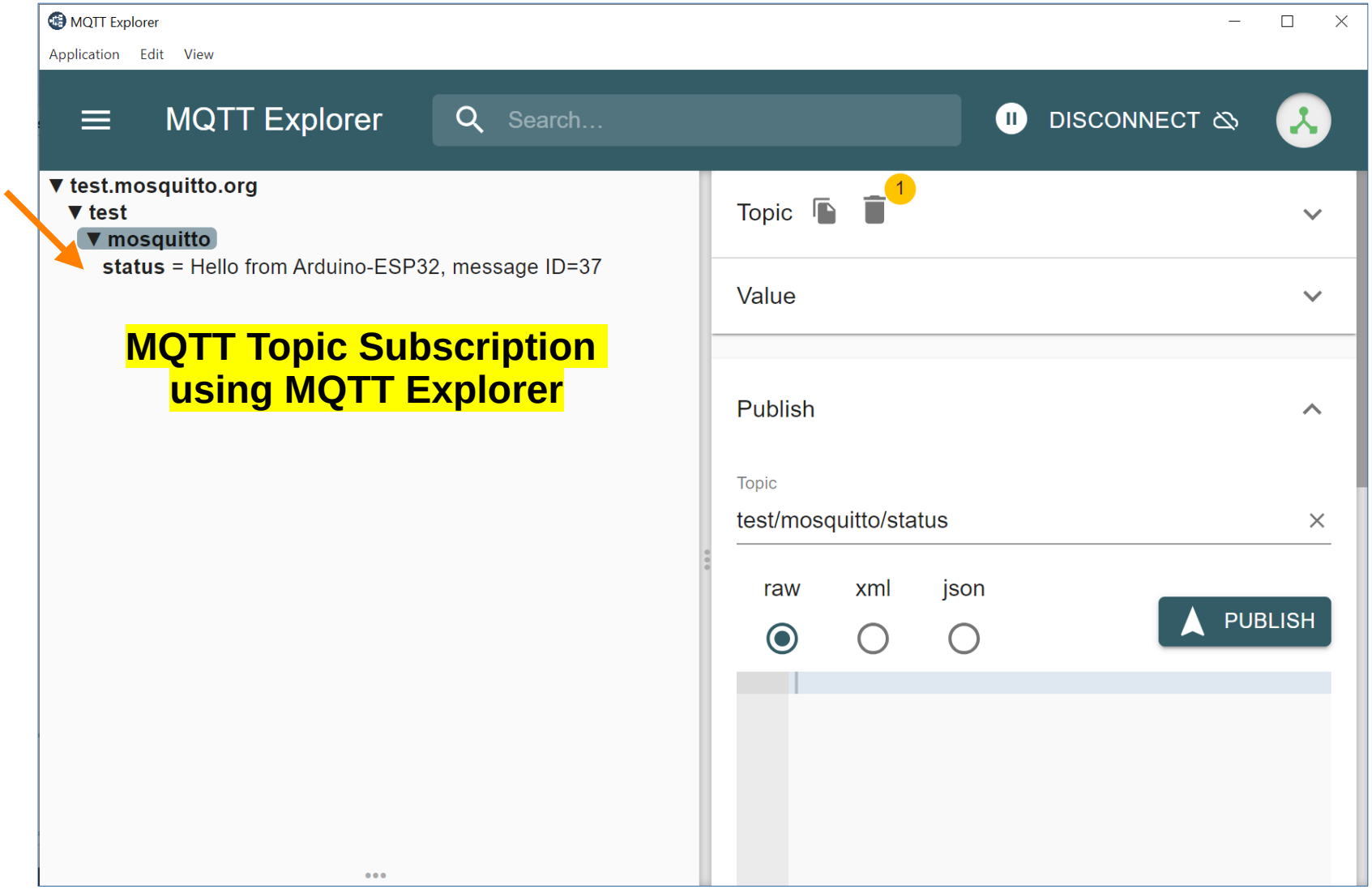
```
// Arduino-ESP32 Core v3.0.x
// Arduino PubSubClient library v2.8

#include <WiFi.h>
#include <PubSubClient.h>     // https://github.com/knolleary/pubsubclient/
#include "arduino_secrets.h"  // Set your WiFI SSID and password in this file.

#define WIFI_TX_POWER  WIFI_POWER_11dBm

#define MQTT_PORT      (1883) // no authentication
#define MQTT_BROKER    "test.mosquitto.org"
//#define MQTT_BROKER   "broker.hivemq.com"

#define CLIENT_ID      "ArduinoESP32Client"
#define SUB_TOPIC      "test/mosquitto/#"
#define PUB_TOPIC      "test/mosquitto/status"
#define INTERVAL_MSEC  (5000)

WiFiClient net;            // ESP32 WiFi client
PubSubClient client(net);  // MQTT client
```

Output Serial Monitor

Message (Enter to send message to 'WEMOS LOLIN32 Lite' on 'COM17') Both NL & CR 115200 baud

```
Published: Hello from Arduino-ESP32, message ID=26

Message received:
  Topic:   'test/mosquitto/status'
  Payload: 'Hello from Arduino-ESP32, message ID=26', rtt=62400 msec
  RTT:      376 [msec]
```

Ln 29, Col 48 WEMOS LOLIN32 Lite on COM17 2


MQTT Explorer
Application
Edit
View
MQTT Explorer
Search...
DISCONNECT
test.mosquitto.org
test
mosquitto
status = Hello from Arduino-ESP32, message ID=37
MQTT Topic Subscription using MQTT Explorer
Topic
1
Value
Publish
Topic
test/mosquitto/status
raw
xml
json
PUBLISH

MQTT Topic Subscription using Mosquitto Client for Ubuntu

ubuntu@LENOVO-LAPTOP: ~

```
ubuntu@LENOVO-LAPTOP:~$ mosquitto_sub -h test.mosquitto.org -p 1883 -t test/mosquitto/status

Hello from Arduino-ESP32, message ID=1
Hello from Arduino-ESP32, message ID=2
Hello from Arduino-ESP32, message ID=3
Hello from Arduino-ESP32, message ID=4
```

Code Listing

esp32_mqtt_demo | Arduino IDE 2.3.2

File Edit Sketch Tools Help

WEMOS LOLIN32 Lite

esp32_mqtt_demo.ino arduino_secrets.h

```
#include <WiFi.h>
#include <PubSubClient.h>     // https://github.com/knolleary/pubsubclient/
#include "arduino_secrets.h"  // Set your WiFI SSID and password in this file.

#define WIFI_TX_POWER  WIFI_POWER_11dBm
#define MQTT_PORT      (1883) // no authentication
#define MQTT_BROKER    "test.mosquitto.org"
//#define MQTT_BROKER   "broker.hivemq.com"

#define CLIENT_ID      "ArduinoESP32Client"
#define SUB_TOPIC      "test/mosquitto/#"
#define PUB_TOPIC      "test/mosquitto/status"
#define INTERVAL_MSEC  (5000)

WiFiClient net;                // ESP32 WiFi client
PubSubClient client(net);      // MQTT client
uint32_t last_pub_ts_msec = 0; // used to keep the last message publish timestamp

void WiFi_connect() {
  WiFi.mode( WIFI_STA );   // WiFi Station mode
  WiFi.disconnect();
  // WiFi Tx power to 11dBm (see: WiFi/WiFiGeneric.h)
  wifi_power_t tx_power = WIFI_TX_POWER;
  WiFi.setTxPower( tx_power );
  WiFi.begin( WIFI_SSID, WIFI_PASS ); // start the WiFi client
  // connect the WiFi network first (if not already connected)
  while (WiFi.status() != WL_CONNECTED) { delay(1000); }
  Serial.println( "\n\nWiFi Connected: ");
  // Show the IP address assigned by DHCP.
  Serial.println( String ("IP address: " ) + WiFi.localIP().toString() );
}
```

Output

Ln 40, Col 1 WEMOS LOLIN32 Lite on COM17 2

Code Listing

esp32_mqtt_demo | Arduino IDE 2.3.2

File Edit Sketch Tools Help

WEMOS LOLIN32 Lite

esp32_mqtt_demo.ino arduino_secrets.h

```
#include <WiFi.h>
#include <PubSubClient.h>     // https://github.com/knolleary/pubsubclient/
#include "arduino_secrets.h"  // Set your WiFI SSID and password in this file.

#define WIFI_TX_POWER  WIFI_POWER_11dBm
#define MQTT_PORT      (1883) // no authentication
#define MQTT_BROKER    "test.mosquitto.org"
//#define MQTT_BROKER   "broker.hivemq.com"

#define CLIENT_ID      "ArduinoESP32Client"
#define SUB_TOPIC      "test/mosquitto/#"
#define PUB_TOPIC      "test/mosquitto/status"
#define INTERVAL_MSEC  (5000)

WiFiClient net;                // ESP32 WiFi client
PubSubClient client(net);      // MQTT client
uint32_t last_pub_ts_msec = 0; // used to keep the last message publish timestamp

void WiFi_connect() {
  WiFi.mode( WIFI_STA );   // WiFi Station mode
  WiFi.disconnect();
  wifi_power_t tx_power = WIFI_TX_POWER;
  WiFi.setTxPower( tx_power ); // Set WiFi Tx power (see: WiFi/WiFiGeneric.h)
  WiFi.begin( WIFI_SSID, WIFI_PASS ); // start the WiFi client
  // connect the WiFi network first (if not already connected)
  while (WiFi.status() != WL_CONNECTED) { delay(1000); }
  Serial.println( "\n\nWiFi Connected: ");
  // Show the IP address assigned by DHCP.
  Serial.println( String ("IP address: " ) + WiFi.localIP().toString() );
}
```

Output

Ln 38, Col 2 WEMOS LOLIN32 Lite on COM17

esp32_mqtt_demo | Arduino IDE 2.3.2

File Edit Sketch Tools Help

Code Listing

WEMOS LOLIN32 Lite

esp32_mqtt_demo.ino arduino_secrets.h

```
void MQTT_connect() {
  // Connect/reconnect the MQTT broker.
  while ( !client.connect(CLIENT_ID, MQTT_USER, MQTT_PASS) ) {
    delay(1000);
  }
  Serial.println( String("MQTT broker connected: ") + MQTT_BROKER + "port " + MQTT_PORT);
  client.subscribe( SUB_TOPIC );
}

// This is the callback function for the incoming MQTT message.
void onMessageReceived( char *topic, byte *payload, unsigned int len ) {
  uint32_t now_msec = millis(); // Get the timestamp for message reception.
  ((char *)payload)[len] = '\0'; // Make a NULL-terminated string.
  Serial.println( "Message received: ");
  Serial.printf( "  Topic:   '%s'\n", topic );
  Serial.printf( "  Payload: '%s', rtt=%lu msec\n", (char *)payload );
  Serial.printf( "  RTT:      %lu [msec]\n", now_msec-last_pub_ts_msec );
  Serial.flush();
}

void setup() {
  Serial.begin( 115200 ); // Initialize the Serial port
  Serial.println("\nArduino-ESP32 MQTT Client Demo...");
  WiFi_connect(); // Connect to the WiFi network.
  client.setServer( MQTT_BROKER, MQTT_PORT ); // Initialize the MQTT broker.
  client.setCallback( onMessageReceived );  // Set the callback function for MQTT events.
  client.setBufferSize( 1024 ); // Set buffer size for MQTT connection.
  Serial.println("Connecting to MQTT broker...");  // Connect to the MQTT broker.
  MQTT_connect(); // Try to connect to the MQTT broker.
}
```

Output

Ln 82, Col 75 WEMOS LOLIN32 Lite on COM17

esp32_mqtt_demo | Arduino IDE 2.3.2

File Edit Sketch Tools Help

Code Listing

WEMOS LOLIN32 Lite

esp32_mqtt_demo.ino arduino_secrets.h

```
void setup() {
  Serial.begin( 115200 ); // Initialize the Serial port
  Serial.println("\nArduino-ESP32 MQTT Client Demo...");
  WiFi_connect(); // Connect to the WiFi network.
  client.setServer( MQTT_BROKER, MQTT_PORT ); // Initialize the MQTT broker.
  client.setCallback( onMessageReceived );  // Set the callback function for MQTT events.
  client.setBufferSize( 1024 ); // Set buffer size for MQTT connection.
  Serial.println("Connecting to MQTT broker...");  // Connect to the MQTT broker.
  MQTT_connect(); // Try to connect to the MQTT broker.
}

void loop() {
  static uint32_t msg_cnt = 0; // published message count
  static char msg[64]; // MQTT message buffer
  if ( !client.connected() ) {
    MQTT_connect(); // Reconnect the MQTT broker if disconnected.
  }
  client.loop();   // Process the MQTT event (non-blocking call).
  if ( millis() - last_pub_ts_msec >= INTERVAL_MSEC ) {
    last_pub_ts_msec = millis(); // Update the message publish timestamp.
    // Define and publish the next message.
    sprintf( msg, "Hello from Arduino-ESP32, message ID=%lu", ++msg_cnt );
    client.publish( PUB_TOPIC, msg ); // Srnd the message.
    Serial.println( String("Published: ") + msg + "\n" );
    Serial.flush();
  }
}
```

Output

Building sketch Ln 86, Col 2 WEMOS LOLIN32 Lite on COM17 1

Code Listing

esp32_mqtt_demo - arduino_secrets.h | Arduino IDE 2.3.2

File Edit Sketch Tools Help

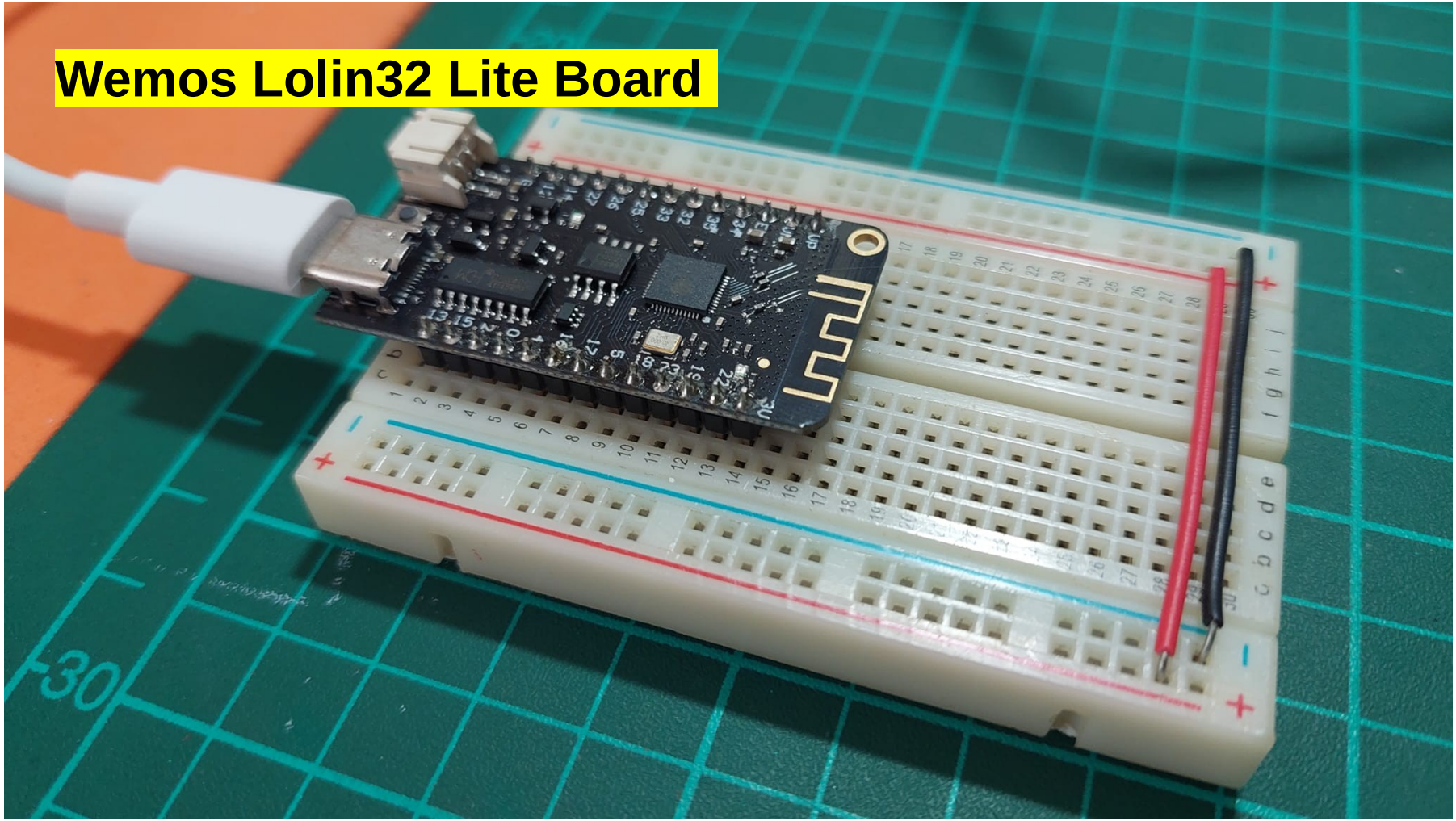
WEMOS LOLIN32 Lite

esp32_mqtt_demo.ino arduino_secrets.h

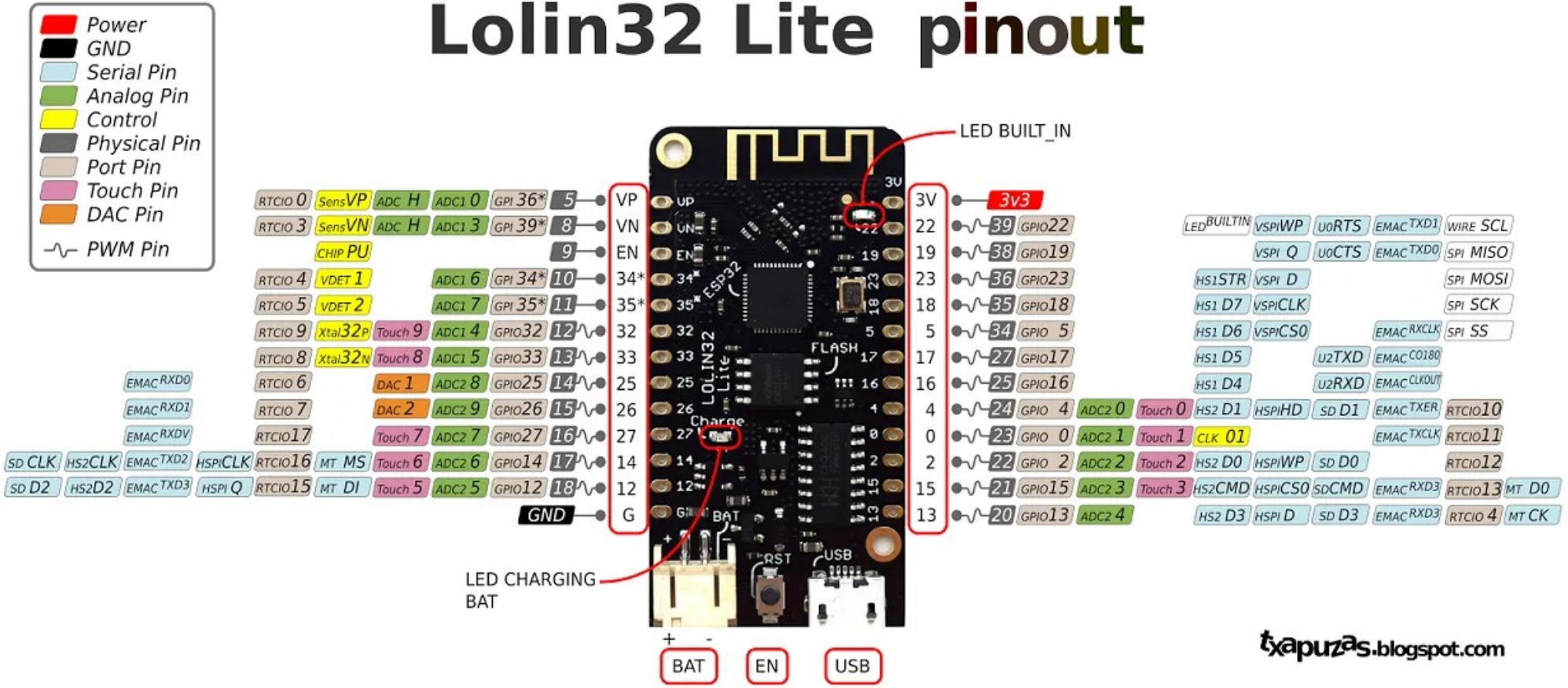
```
1  const char WIFI_SSID[] = "YOUR_WIFI_SSID";
2  const char WIFI_PASS[] = "YOUR_WIFI_PASSWORD";
3  const char MQTT_USER[] = ""; // use an empty string for anonymous user login
4  const char MQTT_PASS[] = "";
5
```

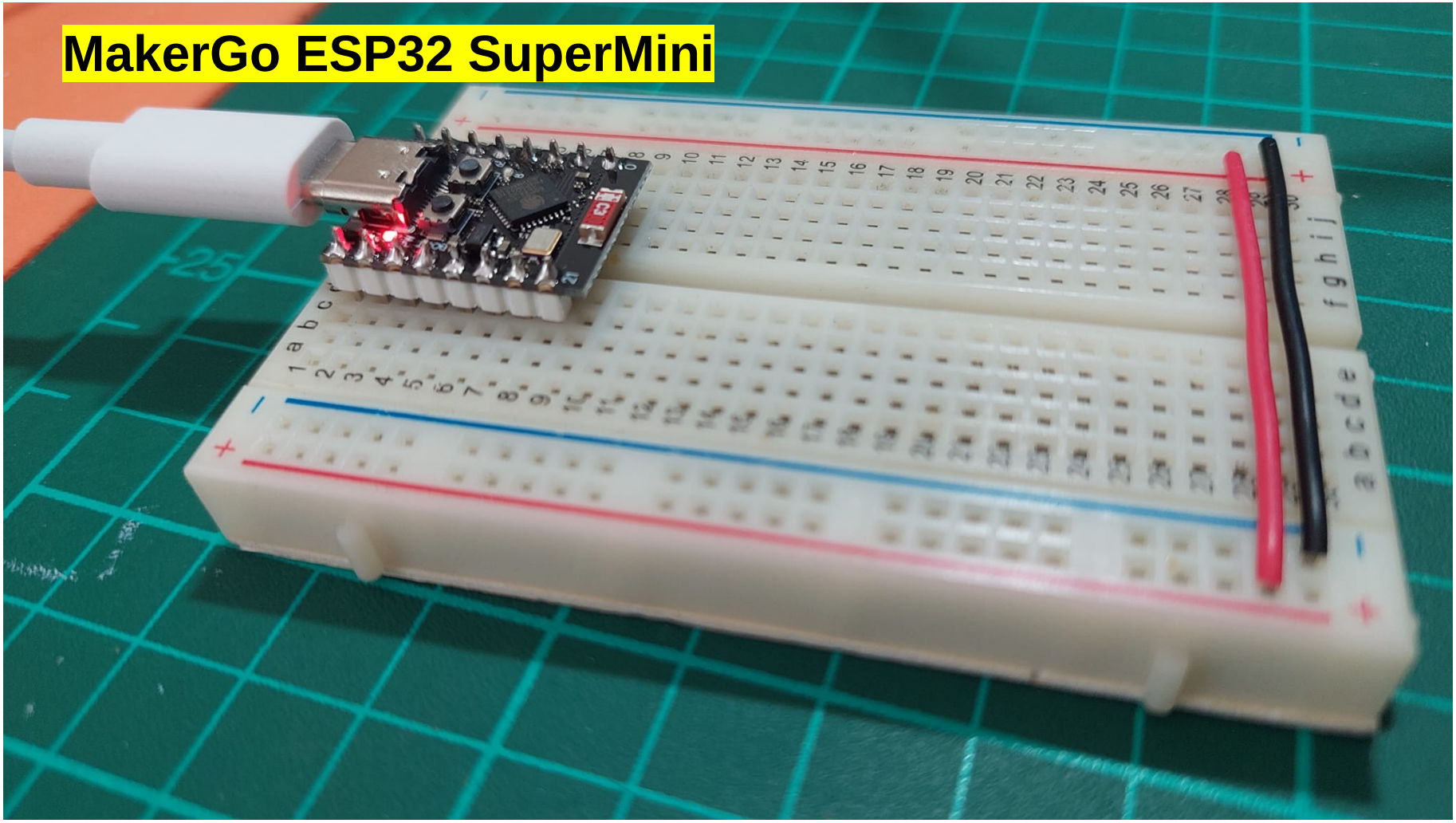
Output

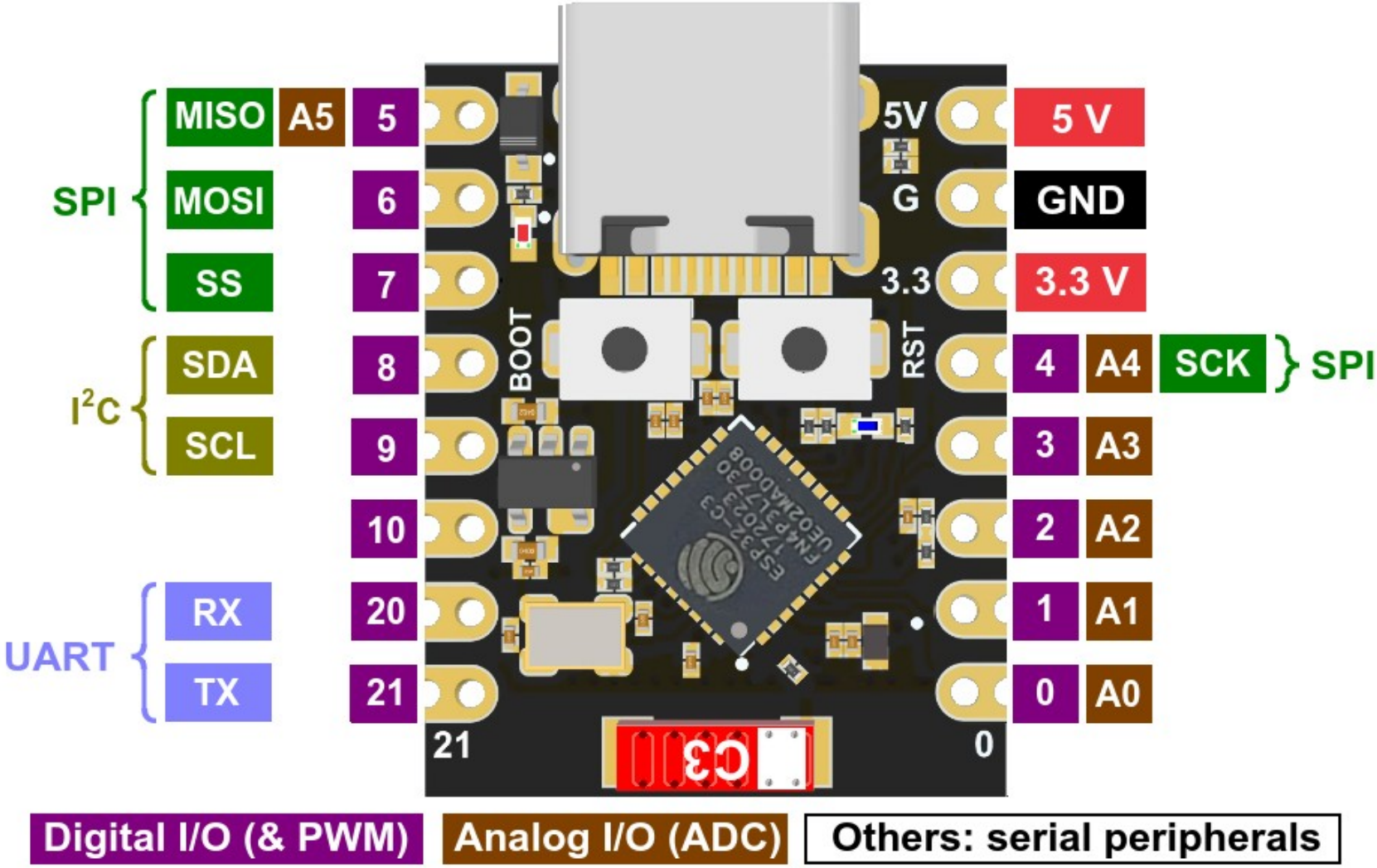
Ln 5, Col 1 WEMOS LOLIN32 Lite on COM17


Wemos Lolin32 Lite Board

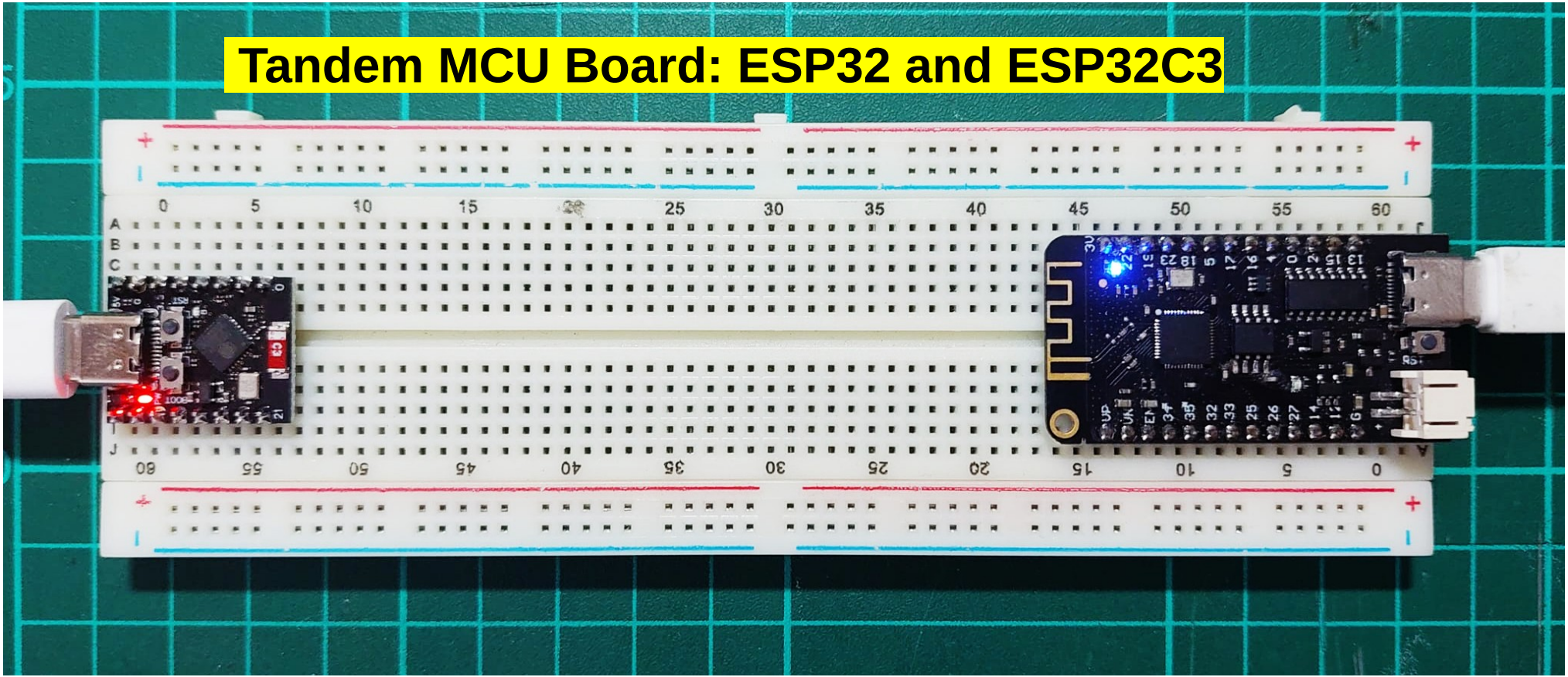
Lolin32 Lite pinout

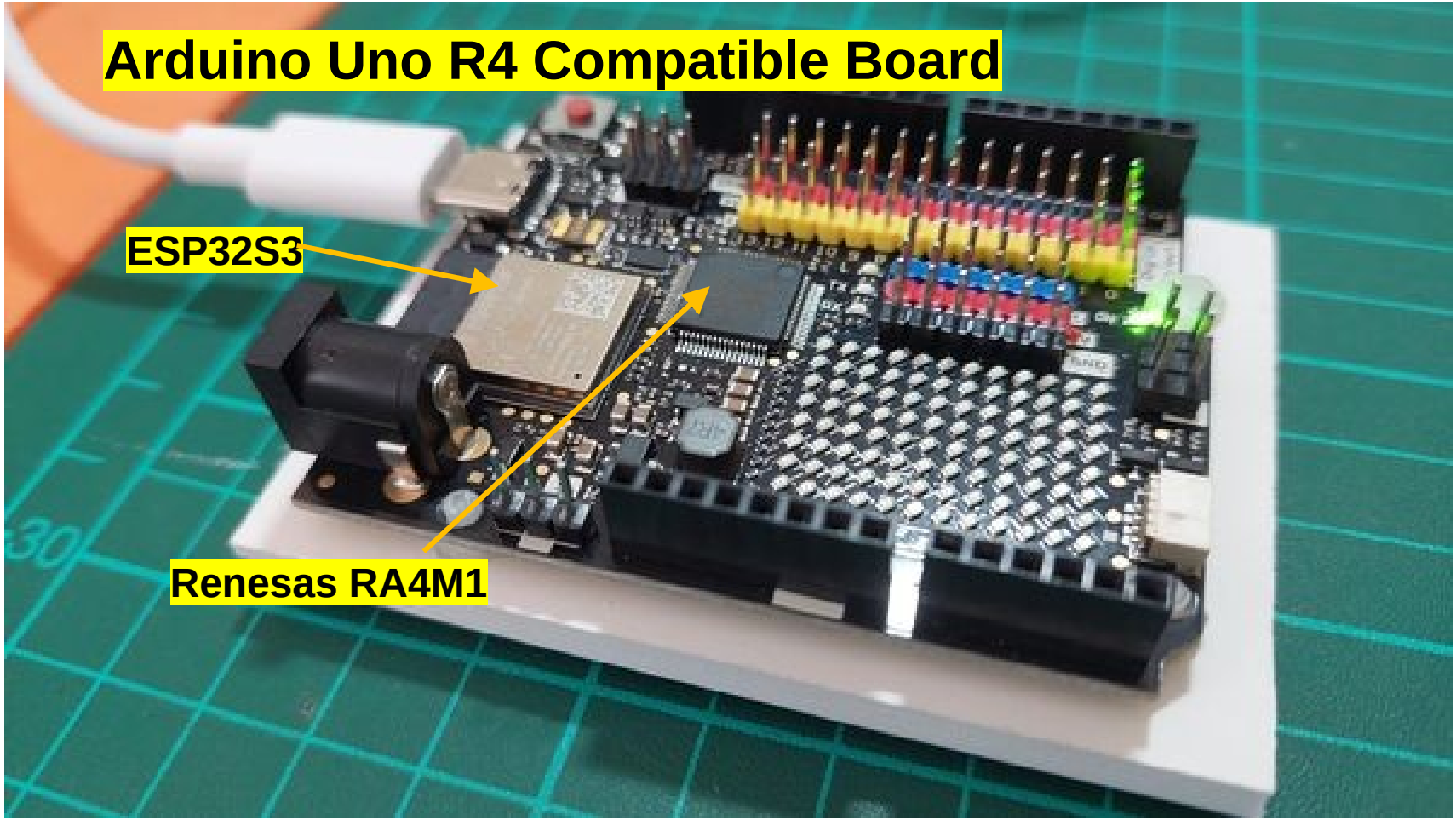



MakerGo ESP32 SuperMini


SPI
MISO
A5
5
MOSI
6
SS
7
I²C
SDA
8
SCL
9
10
UART
RX
20
TX
21
21
BOOT
5V
G
3.3
RST
0
5 V
GND
3.3 V
4
A4
SCK
SPI
3
A3
2
A2
1
A1
0
A0
C3
Digital I/O (& PWM)
Analog I/O (ADC)
Others: serial peripherals

Tandem MCU Board: ESP32 and ESP32C3




Arduino Uno R4 Compatible Board
ESP32S3
Renesas RA4M1

Raspberry Pi Pico-W Board