INTERNET OF THINGS

ASSIGNMENT - 2

PREPARED BY

Garv Baheti

(2020BTechCSE031)

SUBMITTED TO

Mr. Divanshu Jain

Department of Computer Science Engineering
Institute of Engineering and Technology
JK Lakshmipat University Jaipur

September 25, 2022

MQTT – MESSAGE QUEUING TELEMETRY TRANSPORT

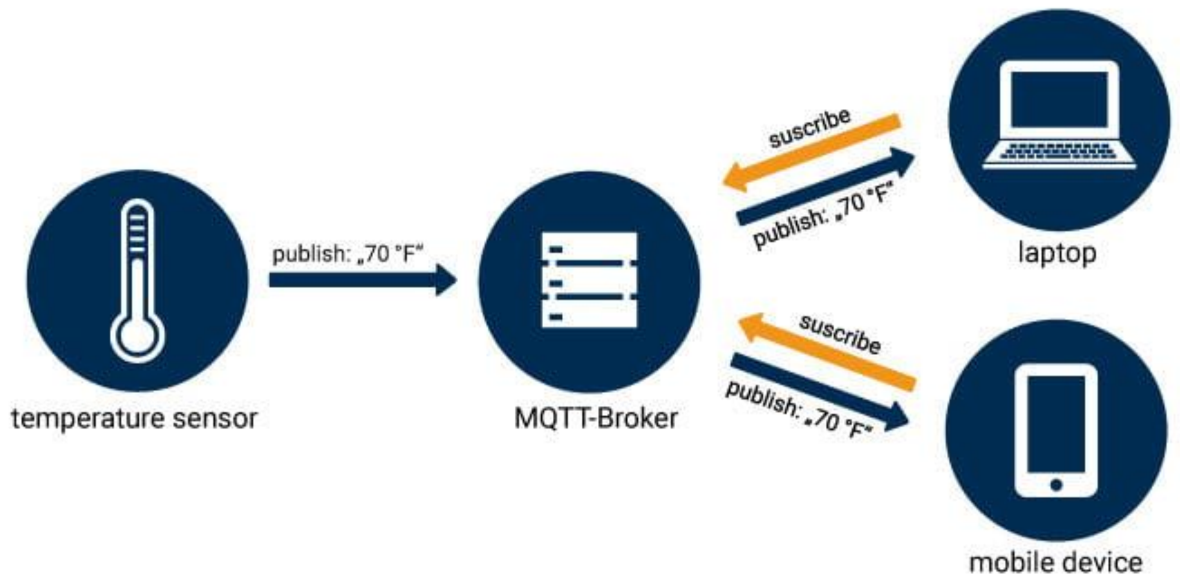
MQTT stands for Message Queuing Telemetry Transport. It is an extremely simple and lightweight messaging protocol (subscribe and publish) designed for limited devices and networks with high latency, low bandwidth, or unreliable networks. Its design principles are designed to reduce the network bandwidth and resource requirements of devices and ensure security of supply. In addition, these principles are advantageous for M2M (machine-to-machine) or IoT devices because battery performance and bandwidth are very important.

MQTT is a messaging protocol for restricted low-bandwidth networks and extremely high-latency IoT devices. Since Message Queuing Telemetry Transport is specialized for low-bandwidth, high-latency environments, it is an ideal **protocol** for machine-to-machine (M2M) communication.

MQTT works on the publisher / subscriber principle and is operated via a central broker. This means that the sender and receiver have no direct connection. The data sources report their data via a publish and all recipients with interest in certain messages ("marked by the topic") get the data delivered because they have registered as subscribers.

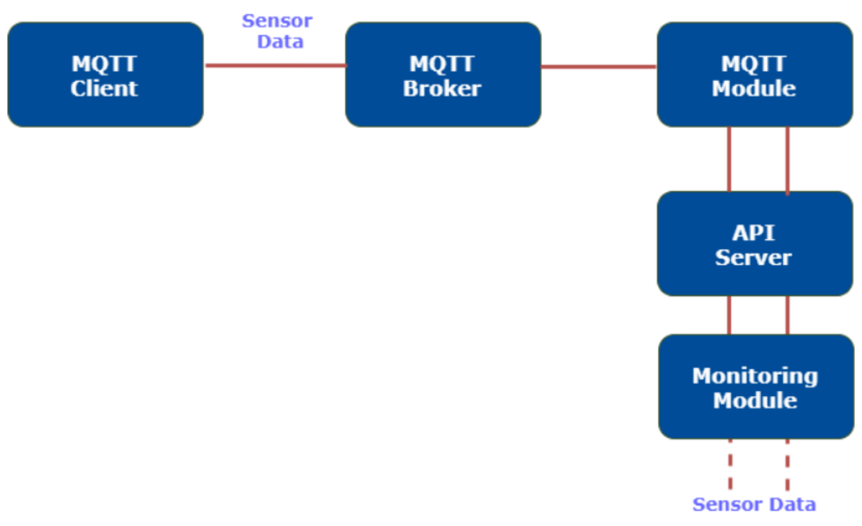
There are three parts of MQTT architecture –

- **MQTT Broker** – All messages passed from the client to the server should be sent via the broker.
- **MQTT Server** – The API acts as an MQTT server. The MQTT server will be responsible for publishing the data to the clients.
- **MQTT Client** – Any third-party client who wishes to subscribe to data published by API, is considered as an MQTT Client.
- The MQTT Client and the MQTT Server need to connect to the Broker in order to publish or subscribe messages.



Suppose our API is sending sensor data to get more ideas on MQTT. API gathers the sensor data through the Monitoring module, and the MQTT module publishes the data to provide different channels. On the successful connection of external client to the MQTT module of the API, the client would receive sensor data on the subscribed channel.

Below diagram shows the flow of data from the API Module to the External clients.



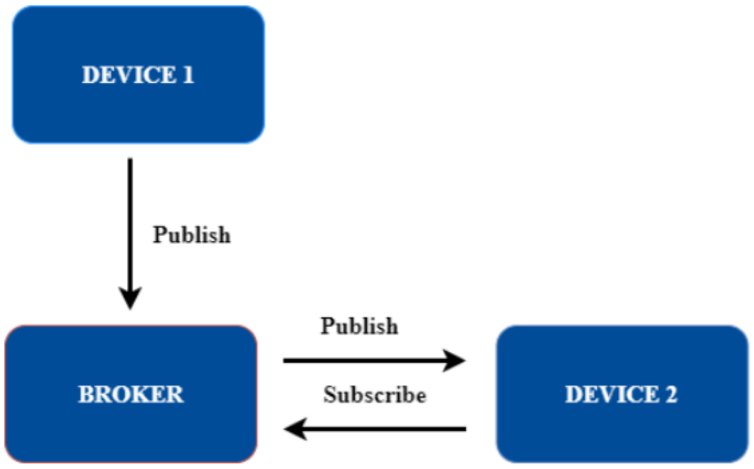
MQTT Broker – EMQTT:

EMQTT (Erlang MQTT Broker) is a massively scalable and cluster able MQTT V3.1/V3.1.1 broker, written in Erlang/OTP.

Main responsibilities of a Broker are-

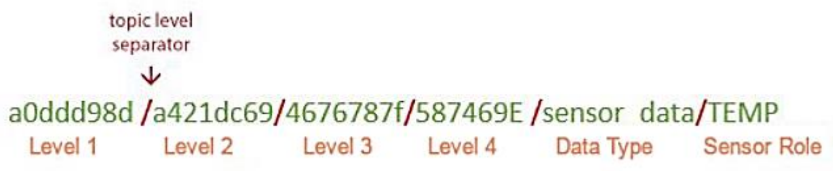
- Receive all messages
- Filter messages
- Decide which are interested clients
- Publish messages to all the subscribed clients

All messages published are passed through the broker. The broker generates the Client ID and Message ID, maintains the message queue, and publishes the message.



MQTT Topics:

A topic is a string(UTF-8). Using this string, Broker filters messages for all connected clients. One topic may consist of one or more topic levels. Forward slash(topic level separator) is used for separating each topic level.



When API starts, the Monitoring API will monitor the sensor data and publish it in a combination of topics. The third party client can subscribe to any of those topics, based on the requirement.

The topics are framed in such a way that it provides options for the user to subscribe at level 1, level 2, level 3, level 4, or individual sensor level data. While subscribing to each level of sensor data, the client needs to specify the hierarchy of the IDs. For e.g. to subscribe to level 4 sensor data, the client needs to specify level 1 id/ level 2 id/ level 3 id/ level 4 id.

The user can subscribe to any type of sensor by specifying the sensor role as the last part of the topic. If the user doesn't specify the role, the client will be subscribed to all types of sensors on that particular level.

The user can also specify the sensor id that they wish to subscribe to. In that case, they need to specify the whole hierarchy of the sensor, starting from project id and ending with sensor id.

FEATURES SUPPORTED BY MQTT:

1. Authentication:

EMQTT provides authentication of every user who intends to publish or subscribe to particular data. The user id and password is stored in the API database, into a separate collection called *'mqtt*

While connecting to EMQTT broker, we provide the username name and password, and the MQTT Broker will validate the credentials based on the values present in the database.

2. Access Control:

EMQTT determines which user is allowed to access which topics. This information is stored in MongoDB under the table *'mqtt_acl'*

By default, all users are allowed to access all topics by specifying '#' as the allowed topic to publish and subscribe for all users.

3. QoS:

The Quality of Service (QoS) level is the Quality transfer of messages which ensures the delivery of messages between sending body & receiving body. There are 3 QoS levels in MQTT:

- ***At most once***(0) –The message is delivered at most once, or it is not delivered at all.
- ***At least once***(1) – The message is always delivered at least once.
- ***Exactly once***(2) – The message is always delivered exactly once.

4. Last Will Message:

MQTT uses the Last Will & Testament(LWT) mechanism to notify ungraceful disconnection of a client to other clients. In this mechanism, when a client is connecting to a broker, each client specifies its last will message which is a normal MQTT message with QoS, topic, retained flag & payload. This message is stored by the Broker until it detects that the client has disconnected ungracefully.

5. Retain Message:

MQTT also has a feature of Message Retention. It is done by setting TRUE to retain the flag. It then retained the last message & QoS for the topic. When a client subscribes to a topic, the broker matches the topic with a retained message. Clients will receive messages immediately if the topic and the retained message are matched. Brokers only store one retained message for each topic.

6. Duplicate Message:

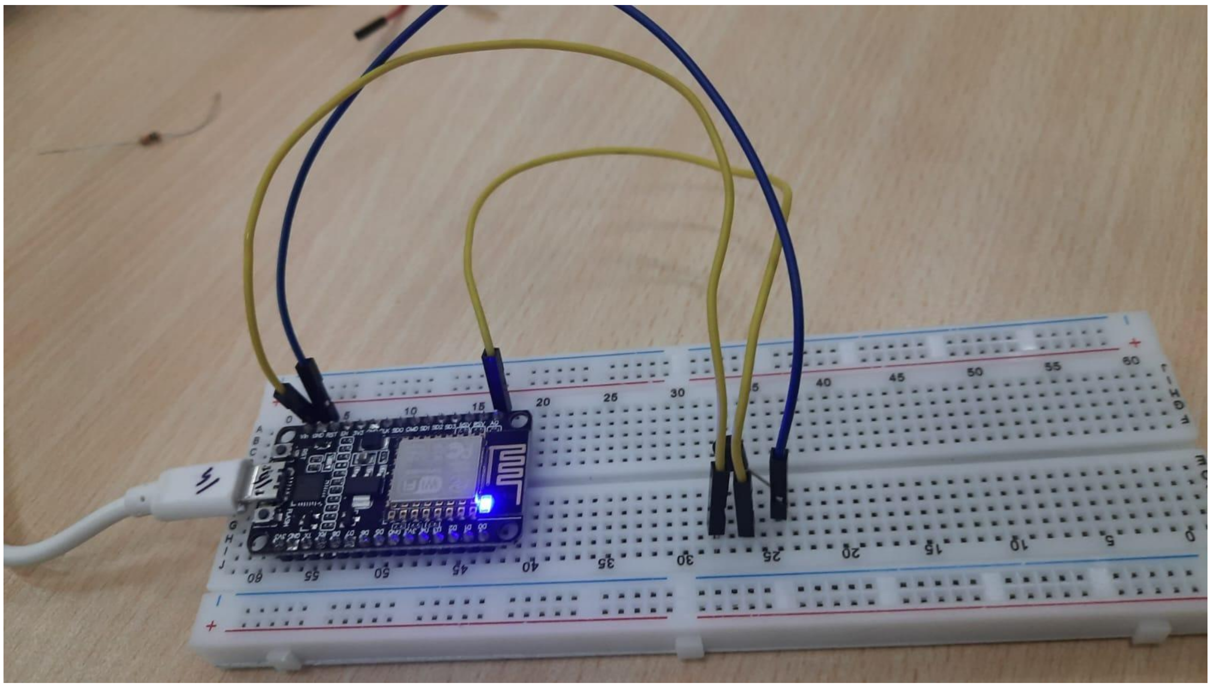
If a publisher doesn't receive the acknowledgement of the published packet, it will resend the packet with DUP flag set to true. A duplicate message contains the same Message ID as the original message.

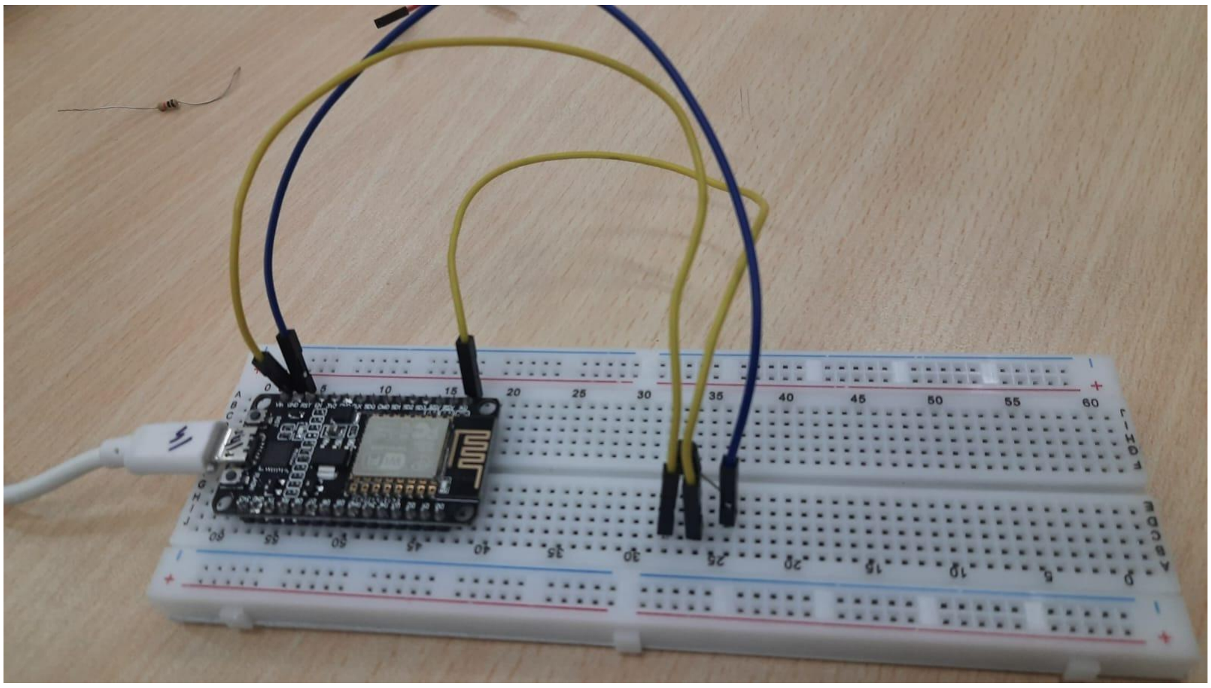
7. Session:

In general, when a client connects with a broker for the first time, the client needs to create subscriptions for all topics for which they are willing to receive data/messages from the broker. Suppose a session is not maintained, or there is no persistent session, or the client lost a connection with the broker, then users must resubscribe to all the topics after reconnecting to the broker. For the clients with limited resources, it would be very tedious to subscribe to all topics again. So, brokers use a persistent session mechanism, in which it saves all information relevant to the client. 'clientId' provided by client is used as 'session identifier' when the client establishes a connection with the broker.

Uploading LM35 sensor data to MQTT server as publisher and control the led connected to NodeMCU using MQTT client.

CIRCUIT OUTPUT



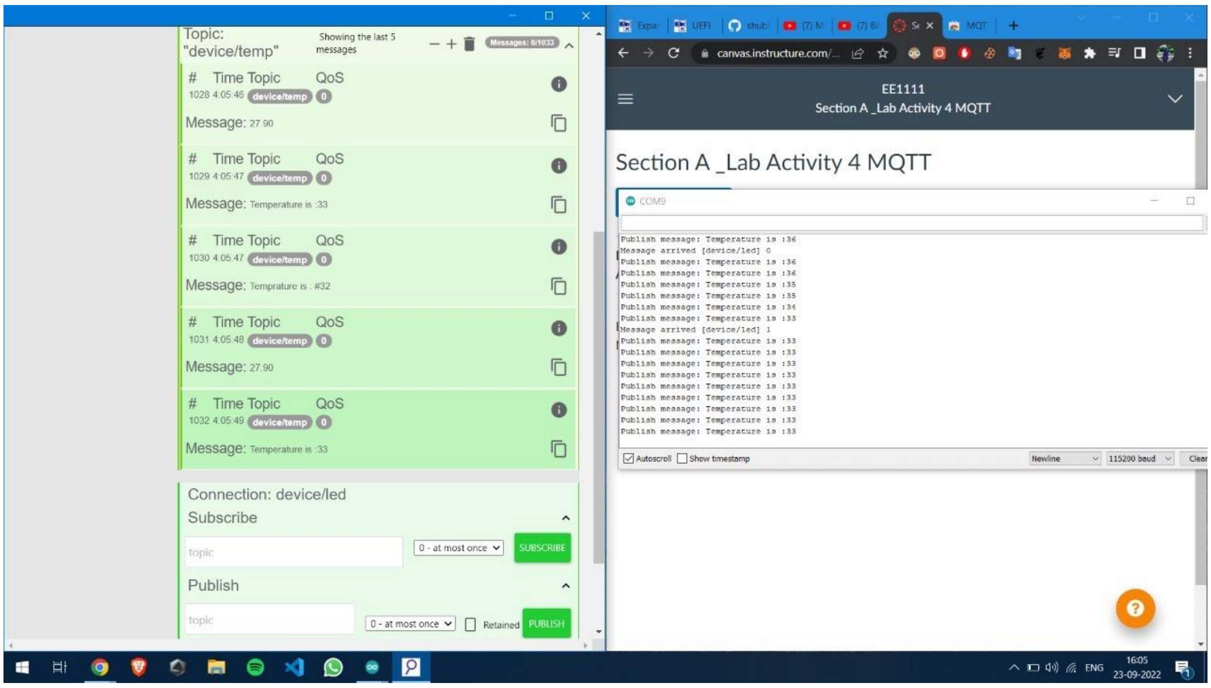


CODE

```
#include <ESP8266WiFi.h>

#include <PubSubClient.h>

// Update these with values suitable for your network.

const char* ssid = "C.K.";

const char* password = "besseuuhm";

const char* mqtt_server = "91.121.93.94";

WiFiClient espClient;

PubSubClient client(espClient);

unsigned long lastMsg = 0;

#define MSG_BUFFER_SIZE        (50)

char msg[MSG_BUFFER_SIZE];

int value = 0;

void setup_wifi() {

  delay(10);

  // We start by connecting to a WiFi network

  Serial.println();

  Serial.print("Connecting to ");

  Serial.println(ssid);

  WiFi.mode(WIFI_STA);

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL_CONNECTED) {

    delay(500);

    Serial.print(".");

  }

  randomSeed(micros());

  Serial.println("");

  Serial.println("WiFi connected");

  Serial.println("IP address: ");

  Serial.println(WiFi.localIP());

}
```

```
void callback(char* topic, byte* payload, unsigned int length) {
  Serial.print("Message arrived [");
  Serial.print(topic);
  Serial.print("] ");
  for (int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if an 1 was received as first character
  if ((char)payload[0] == '1') {
    digitalWrite(BUILTIN_LED, LOW);   // Turn the LED on (Note that LOW is the voltage level
    // but actually the LED is on; this is because
    // it is active low on the ESP-01)
  } else {
    digitalWrite(BUILTIN_LED, HIGH);  // Turn the LED off by making the voltage HIGH
  }

}
void reconnect() {
  // Loop until we're reconnected
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    // Create a random client ID
    String clientId = "ESP8266Client-";
    clientId += String(random(0xffff), HEX);
    // Attempt to connect
    if (client.connect(clientId.c_str())) {
      Serial.println("connected");
      // Once connected, publish an announcement...
      client.publish("device/temp", "MQTT Server is Connected");
      // ... and resubscribe
      client.subscribe("device/led");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      // Wait 5 seconds before retrying
      delay(5000);
    }
  }
}
void setup() {
  pinMode(BUILTIN_LED, OUTPUT);     // Initialize the BUILTIN_LED pin as an output
  Serial.begin(115200);
  setup_wifi();
  client.setServer(mqtt_server, 1883);
  client.setCallback(callback);
}
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
  unsigned long now = millis();
  if (now - lastMsg > 2000) {
    lastMsg = now;
    value = analogRead(A0)*0.32;
    snprintf (msg, MSG_BUFFER_SIZE, "Temperature is #%ld", value);
    Serial.print("Publish message: ");
    Serial.println(msg);
    client.publish("device/temp", msg);
  }
}
```

SERVER OUTPUT

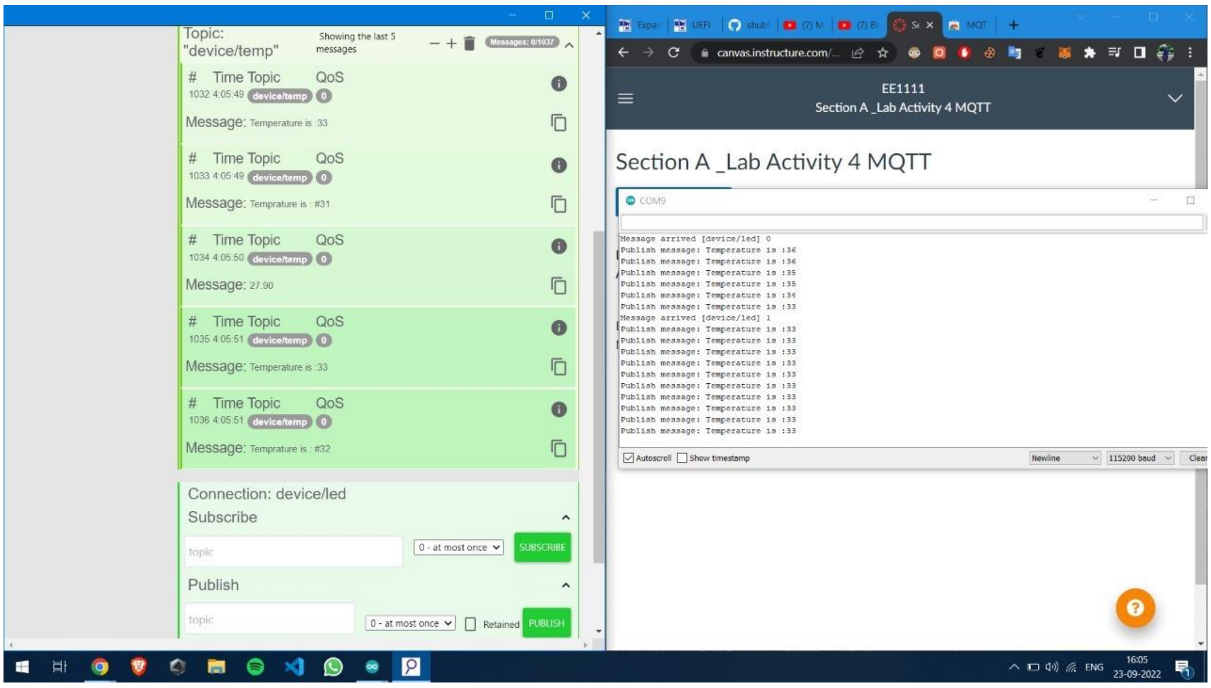


Topic: "device/temp" — Showing the last 5 messages — Messages: 0/1033

#	Time	Topic	QoS	Message
1028	4:05:46	device/temp	0	27.90
1029	4:05:47	device/temp	0	Temperature is :33
1030	4:05:47	device/temp	0	Temprature is : #32
1031	4:05:48	device/temp	0	27.90
1032	4:05:49	device/temp	0	Temperature is :33

Connection: device/led

```
Publish message: Temperature is :36
Message arrived [device/led] 0
Publish message: Temperature is :36
Publish message: Temperature is :36
Publish message: Temperature is :35
Publish message: Temperature is :35
Publish message: Temperature is :34
Publish message: Temperature is :33
Message arrived [device/led] 1
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
```

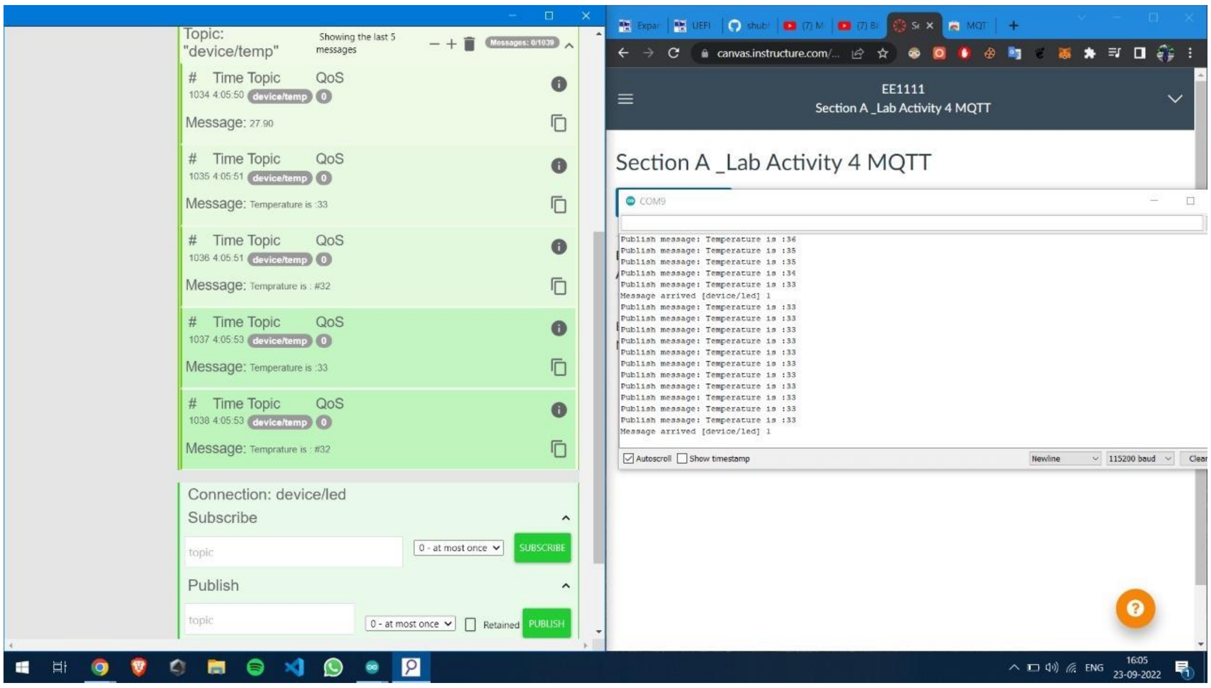


Topic: "device/temp" — Showing the last 5 messages — Messages: 0/1037

#	Time	Topic	QoS	Message
1032	4:05:49	device/temp	0	Temperature is :33
1033	4:05:49	device/temp	0	Temprature is : #31
1034	4:05:50	device/temp	0	27.90
1035	4:05:51	device/temp	0	Temperature is :33
1036	4:05:51	device/temp	0	Temprature is : #32

Connection: device/led

```
Message arrived [device/led] 0
Publish message: Temperature is :36
Publish message: Temperature is :36
Publish message: Temperature is :35
Publish message: Temperature is :35
Publish message: Temperature is :34
Publish message: Temperature is :33
Message arrived [device/led] 1
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
```



Topic: "device/temp" Showing the last 5 messages Messages: 0/1039

#	Time	Topic	QoS	Message
1034	4:05:50	device/temp	0	27.90
1035	4:05:51	device/temp	0	Temperature is :33
1036	4:05:51	device/temp	0	Temprature is : #32
1037	4:05:53	device/temp	0	Temperature is :33
1038	4:05:53	device/temp	0	Temprature is : #32

Connection: device/led

Subscribe: topic, 0 - at most once, SUBSCRIBE

Publish: topic, 0 - at most once, Retained, PUBLISH

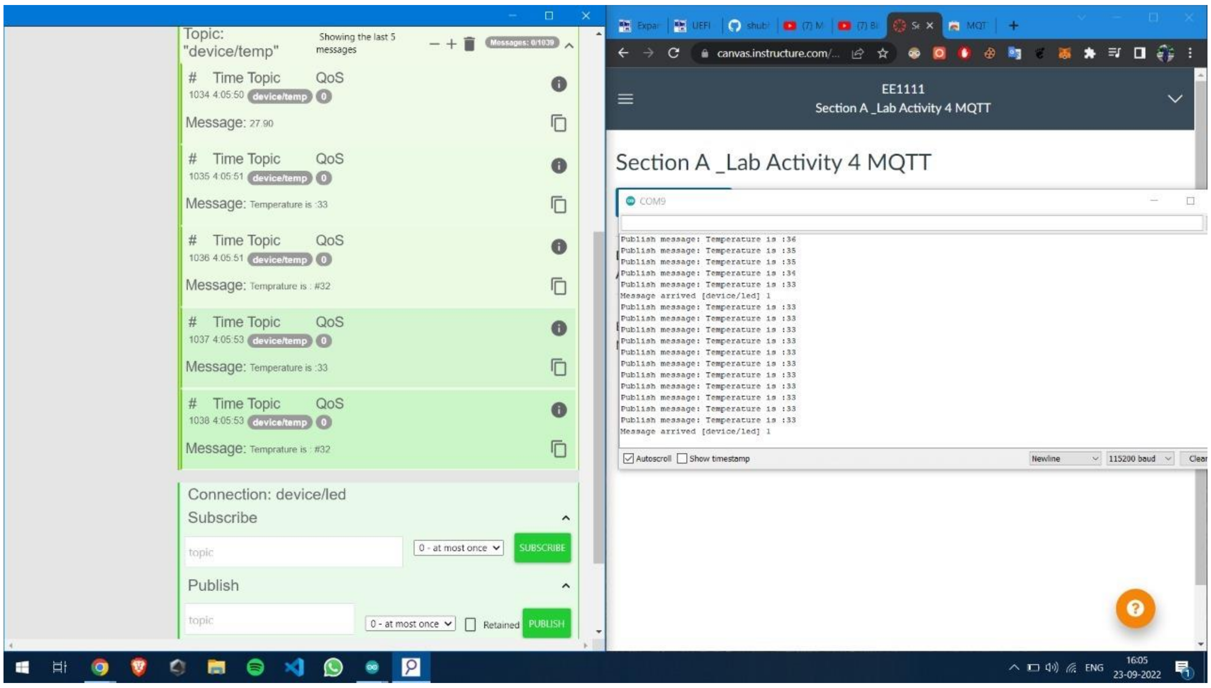
EE1111
Section A _Lab Activity 4 MQTT

Section A _Lab Activity 4 MQTT

COM9

```
Publish message: Temperature is :36
Publish message: Temperature is :35
Publish message: Temperature is :35
Publish message: Temperature is :34
Publish message: Temperature is :33
Message arrived [device/led] 1
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Message arrived [device/led] 1
```

Autoscroll ✓, Show timestamp, Newline, 115200 baud, Clear



Topic: "device/temp" Showing the last 5 messages Messages: 0/1039

#	Time	Topic	QoS	Message
1034	4:05:50	device/temp	0	27.90
1035	4:05:51	device/temp	0	Temperature is :33
1036	4:05:51	device/temp	0	Temprature is : #32
1037	4:05:53	device/temp	0	Temperature is :33
1038	4:05:53	device/temp	0	Temprature is : #32

Connection: device/led

Subscribe: topic, 0 - at most once, SUBSCRIBE

Publish: topic, 0 - at most once, Retained, PUBLISH

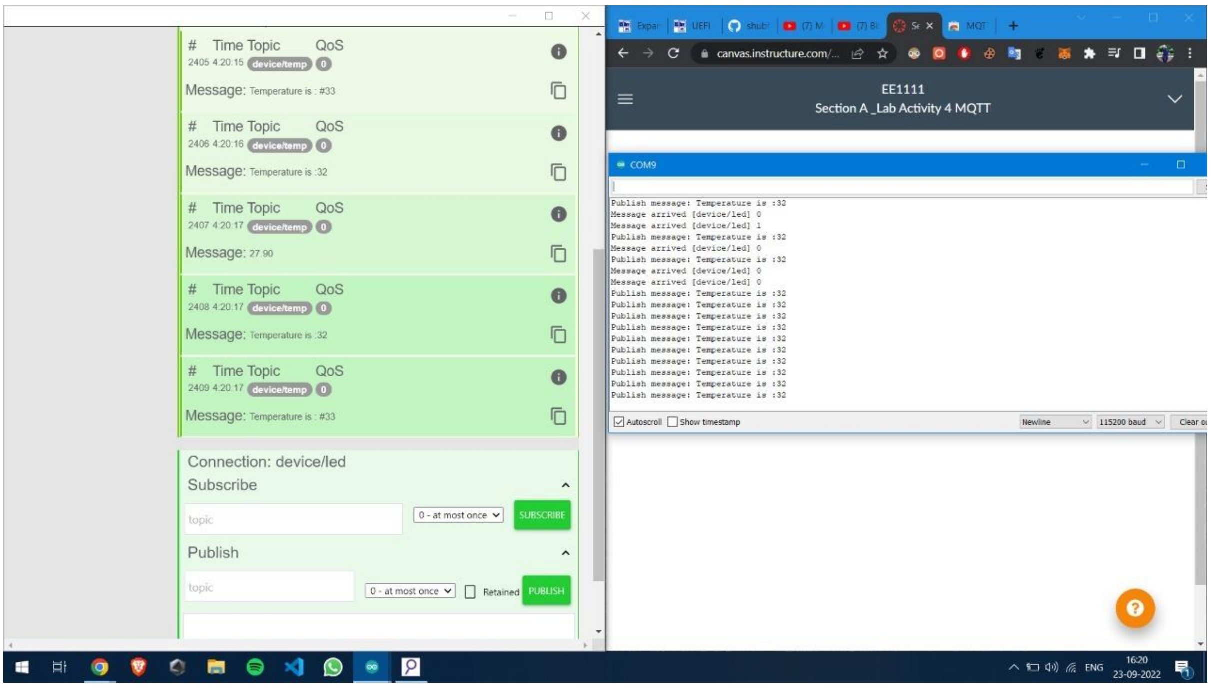
EE1111
Section A _Lab Activity 4 MQTT

Section A _Lab Activity 4 MQTT

COM9

```
Publish message: Temperature is :36
Publish message: Temperature is :35
Publish message: Temperature is :35
Publish message: Temperature is :34
Publish message: Temperature is :33
Message arrived [device/led] 1
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Publish message: Temperature is :33
Message arrived [device/led] 1
```

Autoscroll ✓, Show timestamp, Newline, 115200 baud, Clear



Time Topic QoS
2405 4:20:15 device/temp 0
Message: Temperature is : #33

Time Topic QoS
2406 4:20:16 device/temp 0
Message: Temperature is :32

Time Topic QoS
2407 4:20:17 device/temp 0
Message: 27.90

Time Topic QoS
2408 4:20:17 device/temp 0
Message: Temperature is :32

Time Topic QoS
2409 4:20:17 device/temp 0
Message: Temperature is : #33

Connection: device/led

Subscribe

topic | 0 - at most once | SUBSCRIBE

Publish

topic | 0 - at most once | Retained | PUBLISH

EE1111
Section A _Lab Activity 4 MQTT

COM9

```
Publish message: Temperature is :32
Message arrived [device/led] 0
Message arrived [device/led] 1
Publish message: Temperature is :32
Message arrived [device/led] 0
Publish message: Temperature is :32
Message arrived [device/led] 0
Message arrived [device/led] 0
Publish message: Temperature is :32
Publish message: Temperature is :32
Publish message: Temperature is :32
Publish message: Temperature is :32
Publish message: Temperature is :32
Publish message: Temperature is :32
Publish message: Temperature is :32
Publish message: Temperature is :32
Publish message: Temperature is :32
Publish message: Temperature is :32
```

☑ Autoscroll ☐ Show timestamp | Newline | 115200 baud