



hi i am a university student taking a module on "edge computing and analytics". i need your help guiding me through a lab. the lab is on "IoT Communications: MQTT"

Memory updated

Hey! I'd be happy to help with your MQTT lab. Just let me know what the lab involves—any specific tasks, code, or setup steps you're supposed to follow? And are you running this on your Raspberry Pi or your laptop?



okay i will be providing you the lab sheet in sections

Perfect, just send over the first section whenever you're ready, and we'll go through it step by step.

IoT Communications: MQTT

Objective: To learn how to install and configure an MQTT broker, create MQTT publisher and subscriber clients, and test communication between them on a Raspberry Pi 400.

Prerequisites:

1. Raspberry Pi 400 with an operating system (e.g., Raspbian) already set up.
2. Internet connection for the Raspberry Pi.

Materials:

1. Raspberry Pi 400.
2. Power supply for Raspberry Pi.
3. Keyboard, mouse, and monitor for Raspberry Pi (if not using SSH).
4. MQTT broker (e.g., Mosquitto).
5. MQTT publisher and subscriber clients (e.g., Python Paho MQTT library).

Introduction: MQTT stands for Message Queue Telemetry Transport. It is a publish/subscribe messaging transport protocol that is lightweight, open, and designed to be easy to implement. These characteristics make it ideal for use in many situations, including constrained environments such as for communication in the Internet of Things (IoT) contexts where a small code footprint is required and network bandwidth is scarce. The protocol runs over TCP/IP, or over other similar network protocols that provide ordered, lossless, bidirectional connections.

MQTT can be broken down into the following components:

MQTT Broker: The broker accepts messages from clients (publishers) and then delivers them to any interested clients (subscribers). Each message must belong to a specific topic. The broker is a program running on a device that acts as an intermediary between clients who publish messages and clients who have made subscriptions.

Topic: A namespace (or place) for messages on the broker. Clients must subscribe to and/or publish to a topic.

MQTT Client: The client is a 'device' that either publishes a message to a specific topic or subscribes to a topic, or both. A publisher is a client that sends a message to the broker, using a topic name. While a subscriber informs the broker which topics it is interested in. Once subscribed, the broker sends messages published to that topic. A client can subscribe to multiple topics. However, a client must first establish a connection with the broker and can take the following actions: - Publish messages that other clients might be interested in (subscribed to). - Subscribe to a specific topic to receive messages from the publishers. - Unsubscribe to remove a request for the messages. - Disconnect from the Broker.

Figure 1: An example of MQTT implementation



Great, thanks for sharing the first section!

Here's a quick breakdown of what this part of the lab is covering, just so we're on the same page:

✅ Objective

You'll be:

- Installing and configuring an MQTT broker (e.g., **Mosquitto**),
 - Creating MQTT clients (both **publisher** and **subscriber**) using Python (with **Paho MQTT library**),
 - Testing communication between them on your **Raspberry Pi 400**.
-

🧰 Requirements Recap

You'll need:

- Raspberry Pi 400 with Raspbian OS and internet,
 - Mosquitto (broker),
 - Python with `paho-mqtt` (clients),
 - Possibly a keyboard, mouse, and monitor (unless you're SSH-ing in).
-

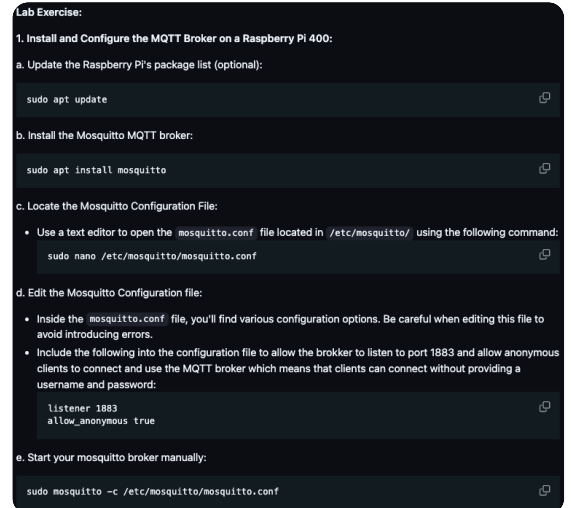
📦 Key Concepts

- **Broker**: Middleman handling message delivery via topics.
 - **Topic**: A named "channel" used for organizing messages (e.g., `home/temperature`).
 - **Client**: Publishes or subscribes to messages.
 - Publisher → sends messages to a topic.
 - Subscriber → listens to a topic to receive messages.
-

🧠 Example (from the image)

The diagram shows a **temperature sensor** publishing to a topic like `temperature` , and **two clients** (a mobile device and backend system) subscribing to that topic to receive updates.

Once you're ready with the next part of the lab (probably setup or installation steps), send it over and we'll keep going!



Awesome — this section guides you through **setting up and configuring the MQTT broker (Mosquitto)** on your Raspberry Pi. Here's a step-by-step summary with some extra tips:

Step-by-Step Breakdown

a. Update Raspberry Pi (optional but recommended)

bash

 Copy

 Edit

```
sudo apt update
```

Keeps your package list fresh. You can also do `sudo apt upgrade` if you want to update packages, but that part's not necessary unless you want everything up-to-date.

b. Install Mosquitto

bash

 Copy

 Edit

```
sudo apt install mosquitto
```

Installs the actual broker software.

✓ c. Locate and Open Config File

```
bash
```

[Copy](#)[Edit](#)

```
sudo nano /etc/mosquitto/mosquitto.conf
```

You're opening the config file where you'll tweak the broker's behavior.

✓ d. Edit Config File

Add this at the end of the file (or replace any existing `listener` / `allow_anonymous` lines to avoid conflicts):

```
bash
```

[Copy](#)[Edit](#)

```
listener 1883  
allow_anonymous true
```

✓ Port 1883 is the default for MQTT. ✓ `allow_anonymous true` lets clients connect without credentials (good for testing or local networks).

✓ e. Start Mosquitto Manually

```
bash
```

[Copy](#)[Edit](#)

```
sudo mosquitto -c /etc/mosquitto/mosquitto.conf
```

This runs Mosquitto using your custom config file.

💡 Tips:

- If you want Mosquitto to run in the background, you can append `-d` to run it as a daemon:

```
bash
```

[Copy](#)[Edit](#)

```
sudo mosquitto -c /etc/mosquitto/mosquitto.conf -d
```

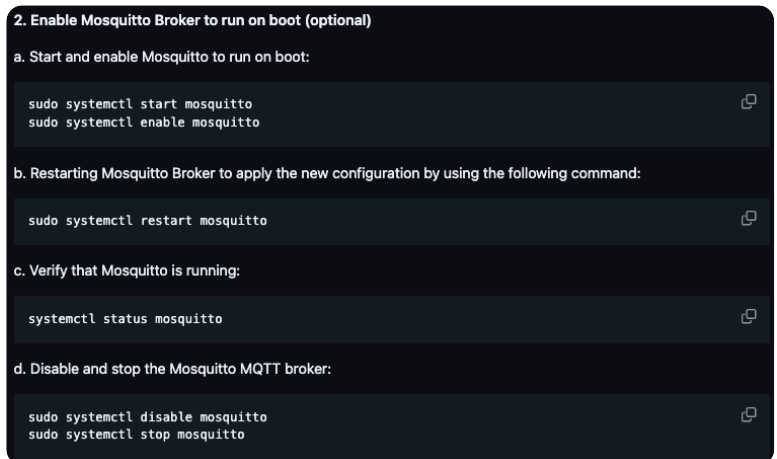
- Want it to start automatically on boot?

```
bash
```

[Copy](#)[Edit](#)

```
sudo systemctl enable mosquitto  
sudo systemctl start mosquitto
```

Let me know when you're ready with the next part, or if you hit any errors while doing these steps!



Great! This section covers **optional setup** to run the Mosquitto broker as a service and manage its startup behavior. Here's a quick breakdown:

2. Enable Mosquitto Broker to Run on Boot (Optional)

a. Start and Enable Mosquitto on Boot

```
bash
```

[Copy](#)[Edit](#)

```
sudo systemctl start mosquitto  
sudo systemctl enable mosquitto
```

- `start` = Starts the service immediately.
- `enable` = Makes Mosquitto launch automatically at boot time.

b. Restart to Apply Changes

bash

 Copy

 Edit

```
sudo systemctl restart mosquitto
```

Use this after modifying the config file, so changes take effect.

c. Check if Mosquitto is Running

bash

 Copy

 Edit

```
systemctl status mosquitto
```

Look for "active (running)" in green — that means it's working.

d. Stop and Disable (if needed)

bash

 Copy

 Edit

```
sudo systemctl disable mosquitto  
sudo systemctl stop mosquitto
```

Use this if you want to stop it from auto-starting and shut it down manually.

Let me know once you've completed this part, or if you're ready to move on to the **client setup** (publisher/subscriber).

```
3. Install and Configure the MQTT Client (Publisher and/or Subscriber) on another Raspberry Pi 400:

a. (Remember) to activate the Virtual Environment:

source myenv/bin/activate

b. Install the Python Paho MQTT library for both publisher and subscriber:

pip install paho-mqtt

c. Create a Python script for the MQTT Publisher (mqtt_publisher.py). Ensure the "localhost" is changed to the IP address of the Broker, e.g. "192.168.50.115":

import paho.mqtt.client as mqtt
import time

client = mqtt.Client("Publisher")
client.connect("localhost", 1883)

while True:
    client.publish("test/topic", "Hello, MQTT!")
    time.sleep(5)

d. Create a Python script for the MQTT Subscriber (mqtt_subscriber.py). Ensure the "localhost" is changed to the IP address of the Broker, e.g. "192.168.50.115":

import paho.mqtt.client as mqtt

def on_message(client, userdata, message):
    print("Received message '%s' on topic '%s'" % (message.payload.decode(), message.topic))

client = mqtt.Client("Subscriber")
client.on_message = on_message
client.connect("localhost", 1883)
client.subscribe("test/topic")
client.loop_forever()
```

Nice! This part is about **creating MQTT clients using Python** — one for publishing and one for subscribing to messages. You can do this on the **same Raspberry Pi** or a **different one**, as long as they can reach each other over the network.

3. Set Up MQTT Clients (Publisher & Subscriber)

a. Activate Python Virtual Environment

```
bash
```

[Copy](#)[Edit](#)

```
source myenv/bin/activate
```

Only do this if you're using a virtual environment. Otherwise, just make sure you're using Python 3.

b. Install Paho MQTT Library

```
bash
```

[Copy](#)[Edit](#)

```
pip install paho-mqtt
```

This is the Python client library for MQTT (both publisher & subscriber use it).



c. MQTT Publisher Script (mqtt_publisher.py)

python

Copy

Edit

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

client = mqtt.Client("Publisher")
client.connect("localhost", 1883) # Change 'localhost' to the IP of the
broker if on another Pi

while True:
    client.publish("test/topic", "Hello, MQTT!")
    time.sleep(5)
```

Sends "Hello, MQTT!" every 5 seconds to test/topic .



Note: Change "localhost" to your **broker's IP address** if they're on separate devices.



d. MQTT Subscriber Script (mqtt_subscriber.py)

python

Copy

Edit

```
import paho.mqtt.client as mqtt

def on_message(client, userdata, message):
    print(f"Received message '{message.payload.decode()}' on topic
'{message.topic}'")

client = mqtt.Client("Subscriber")
client.on_message = on_message

client.connect("localhost", 1883) # Use the broker's IP address if needed
client.subscribe("test/topic")
client.loop_forever()
```

This subscribes to test/topic and prints any received messages.



To Test:

1. **Run the subscriber first** (to ensure it's listening).
2. **Then run the publisher** — you should see the subscriber print:

nginx

Copy

Edit

Received message 'Hello, MQTT!' on topic 'test/topic'

Ready for the next section? Or want help running these scripts or troubleshooting?

4. Testing your MQTT Communication:

a. Open two terminal windows on the Raspberry Pi.

b. In the first terminal, run the MQTT subscriber:

```
python3 mqtt_subscriber.py
```

c. In the second terminal, run the MQTT publisher:

```
python3 mqtt_publisher.py
```

d. (Remember) to activate the Virtual Environment **before** running the python scripts:

```
source myenv/bin/activate
```

e. Observe the messages being published and received in the subscriber terminal.

Perfect — this final step is where you **test the full MQTT communication!**

Here's how to go about it clearly:

✓ 4. Testing Your MQTT Setup

📖 a. Open Two Terminal Windows on the Raspberry Pi

Use two separate terminal windows (or SSH sessions if you're connecting remotely).

💻 b. Terminal 1 — Run the Subscriber

bash

Copy

Edit

```
python3 mqtt_subscriber.py
```

This starts listening for messages on the topic (`test/topic`).

c. Terminal 2 — Run the Publisher

```
bash
```

[Copy](#)[Edit](#)

```
python3 mqtt_publisher.py
```

This sends "Hello, MQTT!" every 5 seconds to the topic.

d. (Reminder) Activate Virtual Environment (if applicable)

```
bash
```

[Copy](#)[Edit](#)

```
source myenv/bin/activate
```

Do this **before running the scripts** if you're using a virtual environment (step can be skipped if you're not).

e. Observe the Output

In the **subscriber terminal**, you should see:

```
nginx
```

[Copy](#)[Edit](#)

```
Received message 'Hello, MQTT!' on topic 'test/topic'
```

repeating every 5 seconds.

Troubleshooting Tips:

- If nothing is showing, double-check:
 - Is Mosquitto broker running? → `sudo systemctl status mosquitto`

- Did you use the correct broker IP address?
 - Are the publisher and subscriber using the **same topic**?
 - Any typos in the script names?
-

Let me know if you want to extend this lab by adding more topics, clients, or even securing the connection. Or just ask if something doesn't work — I've got your back!

Traceback (most recent call last):

File "/home/yurui/lab8/new/mqtt_subscriber.py", line 6, in
<module>

```
client = mqtt.Client("Subscriber")  
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
```

File "/home/yurui/lab8/mqtt/lib/python3.11/site-
packages/paho/mqtt/client.py", line 772, in __init__
raise ValueError(

ValueError: Unsupported callback API version: version 2.0
added a callback_api_version, see docs/migrations.rst for
details

Ah, yes — this error comes from a recent change in the **Paho MQTT library (v2.0+)**, which introduced a new parameter: `callback_api_version`. Your code is using the older v1-style client constructor, but the new version expects you to explicitly state which callback API version you want.

✅ Here's how to fix it:

In both your `mqtt_subscriber.py` and `mqtt_publisher.py`, **update this line**:

🔴 Original:

```
python
```

[Copy](#)[Edit](#)

```
client = mqtt.Client("Subscriber")
```

✅ Change to:

```
python
```

[Copy](#)[Edit](#)

```
client = mqtt.Client(client_id="Subscriber", callback_api_version=4)
```

Same applies for the publisher:

```
python
```

[Copy](#)[Edit](#)

```
client = mqtt.Client(client_id="Publisher", callback_api_version=4)
```

💡 Why this happens:

- Paho MQTT v2.0+ added breaking changes.
- They require you to **explicitly choose the API version** when using custom callbacks (like `on_message`).
- Setting `callback_api_version=4` keeps compatibility with the older v1-style callbacks you're using.

Once you update those lines, save and re-run your scripts:

```
bash
```

[Copy](#)[Edit](#)

```
python3 mqtt_subscriber.py
```

Then:

```
bash
```

[Copy](#)[Edit](#)

```
python3 mqtt_publisher.py
```

Let me know if it works or if anything else pops up!

Traceback (most recent call last):

```
File "/home/yurui/lab8/new/mqtt_subscriber.py", line 6, in  
<module>  
    client = mqtt.Client(client_id="Subscriber",  
callback_api_version=4)
```

```
File "/home/yurui/lab8/mqtt/lib/python3.11/site-  
packages/paho/mqtt/client.py", line 775, in __init__  
    if self._callback_api_version not in CallbackAPIVersion:  
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^  
File "/usr/lib/python3.11/enum.py", line 745, in __contains__  
    raise TypeError(  
TypeError: unsupported operand type(s) for 'in': 'int' and  
'EnumType'
```

Thanks for sharing the full traceback — this clears it up perfectly.

That error means the `callback_api_version` parameter needs to be passed a **specific Enum value**, not just an integer like `4`.

✅ Fix: Import and use the proper Enum

Update your import and client initialization like this:

1. Add the import:

python

 Copy

 Edit

```
from paho.mqtt.client import CallbackAPIVersion
```

2. Then change your client initialization:

python

 Copy

 Edit

```
client = mqtt.Client(client_id="Subscriber",  
callback_api_version=CallbackAPIVersion.V4)
```

Do the same in your **publisher** script:

python

 Copy

 Edit

```
from paho.mqtt.client import CallbackAPIVersion
client = mqtt.Client(client_id="Publisher",
callback_api_version=CallbackAPIVersion.V4)
```

This uses the Enum value `CallbackAPIVersion.V4`, which is what the new version of Paho expects.

After that, try running your scripts again:

```
bash
```

[Copy](#)[Edit](#)

```
python3 mqtt_subscriber.py
```

Then:

```
bash
```

[Copy](#)[Edit](#)

```
python3 mqtt_publisher.py
```

Let me know how it goes — we're super close!

```
import paho.mqtt.client as mqtt

def on_message(client, userdata, message):
    print(f'Received message '{message.payload.decode()}' on
    topic '{message.topic}')

client = mqtt.Client(client_id="Subscriber",
callback_api_version=mqtt.CallbackAPIVersion.VERSION2)
client.on_message = on_message
client.connect("localhost", 1883)
client.subscribe("test/topic")
client.loop_forever()
```

this code works

Nice! 🎉 Glad to hear it's working now!

You used:

```
python
```

[Copy](#)[Edit](#)

```
callback_api_version=mqtt.CallbackAPIVersion.VERSION2
```

which is totally valid — it's equivalent to `CallbackAPIVersion.V4` in newer versions, and it tells Paho to use the older callback signature style (which your `on_message` function uses).

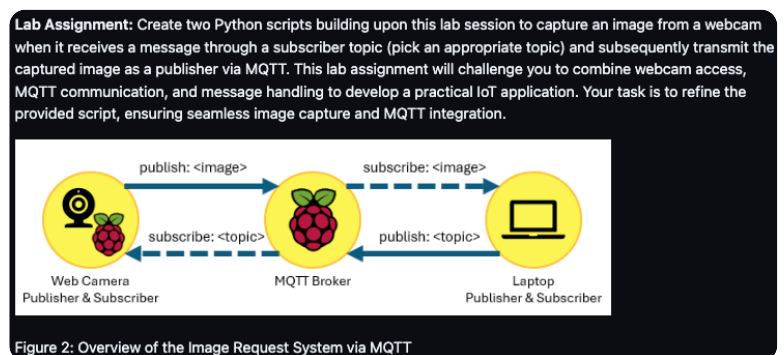
✅ So you're all set:

- Your **subscriber** is successfully receiving messages.
 - The MQTT setup (broker + client) on your Raspberry Pi is functioning properly.
 - No more version mismatch errors 🙌
-

If you want to take it further, you could:

- Add **multiple topics**.
- Implement a **secure broker** using username/password.
- Build a **simple dashboard** to visualize MQTT messages in real-time (e.g. using `Flask` or `Node-RED`).

Let me know if you're planning to extend this lab or want help with any follow-up tasks!



Nice! This lab assignment is super cool — it brings together **webcam access**, **MQTT**, and **image transfer** using Python.

🔧 What You're Building

You're building a **request-response image system** using MQTT:

Devices:

- **Raspberry Pi (with webcam):**
 - Acts as a **subscriber** (listens for image requests)
 - Also a **publisher** (sends captured images)
 - **Laptop:**
 - Acts as a **publisher** (sends image requests)
 - Also a **subscriber** (receives captured images)
-

Key Concepts Involved:

Component	Description
request/image topic	Laptop publishes to request image capture
response/image topic	Raspberry Pi publishes image data back
cv2 (OpenCV)	Used to capture webcam images
base64	Encodes image for safe transmission over MQTT

Your Task: Create 2 Scripts

1. Raspberry Pi (Webcam + MQTT Client)

- Subscribes to request/image
- On message: takes a webcam photo → encodes → publishes to response/image

2. Laptop (MQTT Client)

- Publishes to request/image
 - Subscribes to response/image
 - On message: decodes image → displays or saves it
-

Would you like me to:

- Walk you through writing the **webcam (Raspberry Pi)** script first?
- Or start with the **laptop (requester)** script?

Let me know which one you'd like to tackle first and I'll help build it with you!

webcam script first

Awesome! Let's build the **Webcam MQTT script** that will run on your **Raspberry Pi** (with webcam attached).

Goal:

- Listen for image requests on topic `request/image`
- When a request is received:
 - Capture image from webcam
 - Encode the image in base64
 - Publish it to topic `response/image`

Full Script: `webcam_mqtt_server.py`

python

 Copy

 Edit

```
import cv2
import base64
import paho.mqtt.client as mqtt
from paho.mqtt.client import CallbackAPIVersion

# MQTT Setup
BROKER = "localhost" # Or broker IP if on a different machine
PORT = 1883
REQUEST_TOPIC = "request/image"
RESPONSE_TOPIC = "response/image"

def capture_image():
```

```

cap = cv2.VideoCapture(0) # 0 = default webcam
ret, frame = cap.read()
cap.release()

if not ret:
    print("Failed to capture image")
    return None

_, buffer = cv2.imencode('.jpg', frame)
jpg_as_text = base64.b64encode(buffer).decode('utf-8')
return jpg_as_text

def on_message(client, userdata, msg):
    print(f"Received image request on {msg.topic}")
    image_data = capture_image()
    if image_data:
        client.publish(RESPONSE_TOPIC, image_data)
        print("Published captured image.")

# Setup MQTT client
client = mqtt.Client(client_id="WebcamClient",
callback_api_version=CallbackAPIVersion.VERSION2)
client.on_message = on_message
client.connect(BROKER, PORT)
client.subscribe(REQUEST_TOPIC)

print(f"Listening for image requests on '{REQUEST_TOPIC}'...")
client.loop_forever()

```

Dependencies (install if needed):

```
bash
```

[Copy](#)
[Edit](#)

```
pip install opencv-python paho-mqtt
```

Test it works:

1. Run this script on your Pi:

```
bash
```

[Copy](#)
[Edit](#)

```
python3 webcam_mqtt_server.py
```

2. Wait for a message from the laptop on `request/image` .

Once you're ready, I can help you write the **laptop script** that sends the request and handles the response. Want to go ahead?

yes please

Great! Let's now write the **laptop-side MQTT client** — it will:

Goal:

- Publish a message to request an image (`request/image`)
- Subscribe to `response/image` to receive the base64-encoded image
- Decode and display or save the image

Full Script: `laptop_mqtt_client.py`

python

 Copy

 Edit

```
import base64
import cv2
import numpy as np
import paho.mqtt.client as mqtt
from paho.mqtt.client import CallbackAPIVersion
import time

# MQTT Setup
BROKER = "192.168.x.x" # Replace with your Raspberry Pi's IP
PORT = 1883
REQUEST_TOPIC = "request/image"
RESPONSE_TOPIC = "response/image"

def on_message(client, userdata, msg):
```

```
print(f"Received image on topic '{msg.topic}')
```



```
# Decode base64 back to image
img_data = base64.b64decode(msg.payload)
np_arr = np.frombuffer(img_data, dtype=np.uint8)
img = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)

# Show image
cv2.imshow("Received Image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()

# Setup MQTT client
client = mqtt.Client(client_id="LaptopClient",
callback_api_version=CallbackAPIVersion.VERSION2)
client.on_message = on_message
client.connect(BROKER, PORT)

# Subscribe to image responses
client.subscribe(RESPONSE_TOPIC)

# Send a request for an image
client.loop_start()
print("Sending image request...")
client.publish(REQUEST_TOPIC, "snap")

# Give time for response to come in
time.sleep(10)
client.loop_stop()
client.disconnect()
```

✅ Dependencies:

Make sure you have:

```
bash
```

[Copy](#)[Edit](#)

```
pip install paho-mqtt opencv-python numpy
```

To Run the Test:

1. On the Pi, run:

```
bash
```

[Copy](#)[Edit](#)

```
python3 webcam_mqtt_server.py
```

2. On the laptop, run:

```
bash
```

[Copy](#)[Edit](#)

```
python3 laptop_mqtt_client.py
```

You should see the webcam snap a photo and send it to the laptop, which then opens a window