End-to-End MQTT Cloud Setup (Google Cloud VM + Mosquitto + TLS)

This document is a **complete, repeatable, step-by-step guide** for setting up a secure MQTT broker in the cloud using **Google Cloud Compute Engine (Always Free tier)** and **Mosquitto**, exactly following what we did today.

The goal is: - A **cloud-hosted MQTT broker** - Secure remote connections from anywhere in the world - TLS encryption (MQTTS on port 8883) - Username/password authentication - A setup suitable for IoT (Raspberry Pi, ESP32, backend services)

No Python client code is included yet, by design.

---

0. Conceptual Overview (What You Are Building)

Architecture:

```
[Raspberry Pi / IoT Device]
        |
        |  MQTTS (TLS, TCP 8883)
        v
[ Google Cloud VM ]
   ├─ Mosquitto (MQTT Broker)
   └─ (Later) FastAPI Service
```

- Devices publish data securely over the internet
- Mosquitto acts as the message hub
- Backend services (FastAPI later) consume MQTT locally

This is **real cloud computing** and **industry-standard IoT architecture**.

---

1. Create Google Cloud Account

1. Go to: https://cloud.google.com
2. Create a Google Cloud account
3. Enable **Billing** (credit card required, but free tier is $0)
4. Create a new **Project**

   Billing must be enabled even for free resources.

---

2. Create the Always-Free VM Instance

Required Free-Tier Conditions (VERY IMPORTANT)

Setting	Value
Machine type	**e2-micro**
vCPU / RAM	1 shared core, 1 GB RAM
Region	us-central1 OR us-east1 OR us-west1
Disk	Standard persistent disk ≤ 30 GB
VM count	1

If **any** of these are wrong, you will be charged.

Steps

1. Go to **Compute Engine → VM Instances → Create Instance**
2. Ignore presets
3. Set:
4. **Series**: E2
5. **Machine type**: e2-micro
6. Region: us-central1 (recommended)
7. Boot disk:
8. Ubuntu or Debian
9. Standard persistent disk
10. Size ≤ 30 GB
11. Networking: keep default
12. Click **Create**

    The UI may show ~$7/month. This is list price. Free tier credits cancel it to $0.

---

3. Connect to the VM

1. In VM Instances list, click **SSH**
2. A browser terminal opens
3. You are now logged into your cloud Linux server

---

4. Update the System

```
sudo apt update && sudo apt upgrade -y
```

- `apt` = Linux package manager
- `-y` = auto-confirm "yes"

---

5. Install Mosquitto (MQTT Broker)

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

Enable and start the service:

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

Check status:

```
sudo systemctl status mosquitto
```

You should see: - **Active: active (running)** - No errors

---

6. Create MQTT Users (Authentication)

Each device gets its own MQTT user.

Create the first user (creates the password file)

```
sudo mosquitto_passwd -c /etc/mosquitto/passwd greenhouse-broker
```

- You will be prompted for a password
- Password is hashed (not stored in plaintext)

Add more users later (NO -c)

```
sudo mosquitto_passwd /etc/mosquitto/passwd raspi-1
```

`-c` is used **only once**.

---

7. Create TLS Certificates (Encryption)

Create certificates directory

```
sudo mkdir -p /etc/mosquitto/certs
cd /etc/mosquitto/certs
```

Create Certificate Authority (CA)

```
sudo openssl genrsa -out ca.key 2048
sudo openssl req -x509 -new -nodes -key ca.key -sha256 -days 3650 -out ca.crt
```

Create Server Certificate

```
sudo openssl genrsa -out server.key 2048
sudo openssl req -new -key server.key -out server.csr
sudo openssl x509 -req -in server.csr
  -CA ca.crt -CAkey ca.key -CAcreateserial
  -out server.crt -days 3650
```

When asked for **Common Name (CN)**, use: - Your VM **external IP address**

---

8. Secure Certificate Permissions

```
sudo chown mosquitto:mosquitto /etc/mosquitto/certs/server.key /etc/mosquitto/
certs/server.crt
sudo chmod 600 /etc/mosquitto/certs/server.key
sudo chmod 644 /etc/mosquitto/certs/server.crt /etc/mosquitto/certs/ca.crt
```

---

9. Configure Mosquitto Listeners (LOCAL + SECURE REMOTE)

At this stage we configure **two listeners**:

- **1883** → Plain MQTT, **LOCAL ONLY** (FastAPI ↔ Mosquitto inside the VM)
- **8883** → MQTT over TLS, **REMOTE ACCESS** (Raspberry Pi / external clients)

This separation is a **best practice** in real systems.

9.1 Create Mosquitto Configuration File

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

Paste **exactly**:

```
# ================================
# LOCAL MQTT (NO TLS)
# ================================
listener 1883 127.0.0.1
protocol mqtt

allow_anonymous false
password_file /etc/mosquitto/passwd

# ================================
# SECURE REMOTE MQTT (TLS)
# ================================
listener 8883
protocol mqtt

allow_anonymous false
password_file /etc/mosquitto/passwd

cafile /etc/mosquitto/certs/ca.crt
certfile /etc/mosquitto/certs/server.crt
keyfile /etc/mosquitto/certs/server.key

tls_version tlsv1.2
```

Important details: - `127.0.0.1` means port **1883 is NOT reachable from the internet** - Only **8883** is exposed externally - Both listeners use authentication

Save and exit.

9.2 Restart Mosquitto

```
sudo systemctl restart mosquitto
sudo systemctl status mosquitto --no-pager
```

You should see: - `Active: active (running)` - No TLS or permission errors

---

10. TLS CERTIFICATE FIX (SAN – Subject Alternative Name)

Why this step exists

When connecting via **IP address**, modern TLS requires that the server certificate contains the IP inside a **Subject Alternative Name (SAN)** field.

Without this, clients (Python, browsers, mobile apps) will fail with:

```
certificate verify failed: IP address mismatch
```

This step was required after the first TLS test failed.

10.1 Create SAN configuration file

```
sudo nano /etc/mosquitto/certs/san.cnf
```

Paste (replace IP if yours changes):

```
[req]
prompt = no
distinguished_name = dn
req_extensions = req_ext

[dn]
CN = 34.55.246.208

[req_ext]
subjectAltName = @alt_names

[alt_names]
IP.1 = 34.55.246.208
DNS.1 = localhost
```

10.2 Regenerate server certificate with SAN

```
cd /etc/mosquitto/certs

sudo openssl genrsa -out server.key 2048
sudo openssl req -new -key server.key -out server.csr -config san.cnf

sudo openssl x509 -req
  -in server.csr
  -CA ca.crt -CAkey ca.key -CAcreateserial
  -out server.crt -days 3650
  -extfile san.cnf -extensions req_ext
```

10.3 Fix permissions again

```
sudo chown mosquitto:mosquitto server.key server.crt
sudo chmod 600 server.key
sudo chmod 644 server.crt ca.crt
```

Restart Mosquitto:

```
sudo systemctl restart mosquitto
```

```
---

## 10. Open Cloud Firewall (CRITICAL)

In Google Cloud Console:

1. Go to **VPC Network → Firewall Rules**
2. Create rule:
   - Name: allow-mqtts
   - Direction: Ingress
   - Source IP ranges: 0.0.0.0/0
   - Protocols: tcp:8883
3. Save

> Do NOT open port 1883 in production.

---

## 11. Test TLS Locally on the VM
```

```bash
openssl s_client -connect <VM_EXTERNAL_IP>:8883 -CAfile /etc/mosquitto/certs/
ca.crt
```

You should see:

```
Verify return code: 0 (ok)
```

---

12. Test MQTT Subscription on the VM

```
mosquitto_sub
  -h <VM_EXTERNAL_IP>
  -p 8883
  -t '#'
  -v
  -u greenhouse-broker
  -P <PASSWORD>
  --cafile /etc/mosquitto/certs/ca.crt
```

This confirms: - Broker is reachable globally - TLS works - Authentication works

---

13. What You Have Achieved

✔️ Always-free cloud VM ✔️ Secure MQTT broker ✔️ TLS encryption ✔️ Authentication ✔️ Firewall protection ✔️ Global access

This setup can now safely accept MQTT data from **anywhere in the world**.

---

14. What Comes Next (Later)

- FastAPI subscribing to MQTT locally
- PostgreSQL / time-series storage
- Device certificates
- Topic-level ACLs
- Monitoring & logging

---

15. Key Takeaway

This is **real cloud IoT infrastructure**, not a toy setup.

What you built today mirrors how production IoT systems work.

You can now repeat this entire process confidently.

End of documentation