Raspberry Pi MQTT Gateway Setup

What This Achieves

The Meshtastic node acts as the MQTT gateway, bridging LoRa packets to an MQTT broker (it uplinks/downlinks LoRa<->MQTT using its own network connection or the connected client's). Mosquitto running on the Raspberry Pi is just the broker: it provides a persistent, always-on broker and host so you do not need a phone or laptop running the Meshtastic app. The node must have `mqtt.enabled` set and point at the broker for any bridging to happen - running Mosquitto alone does not bridge the mesh. Once configured it forwards mesh packets to MQTT (locally and/or to a cloud broker), enables remote monitoring of all nodes on your mesh, and allows the node to relay messages between the mesh and internet-connected services.

Hardware Requirements

Component	Notes
Raspberry Pi 3B+, 4, or Zero 2W	3B+ or 4 for comfort; Zero 2W for power-constrained installs. All run Pi OS Lite adequately.
MicroSD card (16 GB+)	Class 10 / A1 rated. Use a quality brand - SD card failures are the #1 Pi reliability issue.
Meshtastic USB node	T-Beam, Heltec V3, RAK WisBlock, or any supported device that presents a USB serial port (native CDC-ACM, or via a CP210x/CH340 USB-UART bridge that appears as a ttyUSB device). See meshtastic.org/docs/hardware for supported devices.
USB cable	Data-capable USB-A to USB-C (or micro, depending on node).
Case and power supply	Official Pi PSU or PoE HAT for rooftop deployments.

Software Setup

Step 1 - Flash the OS

Use **Raspberry Pi Imager** to flash **Raspberry Pi OS Lite (64-bit)** to the SD card. In the imager's Advanced Options, pre-configure:

- Hostname (e.g. `mesh-gw-01`)
- Username and password (since Pi OS Bookworm there is no default `pi` user - the first user is created here; note the name you choose, you will need it for the systemd unit below)
- SSH enabled with your public key
- WiFi credentials (or leave blank if using Ethernet)

Step 2 - Install dependencies

```
sudo apt update && sudo apt upgrade -y
sudo apt install -y mosquitto mosquitto-clients python3-pip
pip3 install meshtastic
```

Step 3 - Configure Mosquitto

Edit `/etc/mosquitto/mosquitto.conf` (or create a file in `/etc/mosquitto/conf.d/`). **Mosquitto 2.0+ (the version shipped by apt) defaults `allow_anonymous` to false and, with no listener defined, binds to localhost only** - a bare install rejects LAN clients until you add a listener and either allow anonymous access or a password file:

```
# Allow anonymous local connections (safe for LAN-only installs)
listener 1883
allow_anonymous true

# For remote access, use authentication instead:
# listener 1883 0.0.0.0
# allow_anonymous false
# password_file /etc/mosquitto/passwd
```

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

Step 4 - Configure the Meshtastic node

Connect to the node via the Meshtastic app or CLI and set:

- **MQTT server / address**: `localhost` (if running on the same Pi) or the Pi's LAN IP from another device.
- **MQTT port**: 1883
- **Uplink enabled**: Yes (per-channel; uplink defaults to off, so enable it on each channel you want to bridge)
- **Downlink enabled**: Yes (to receive messages from MQTT back to the mesh)
- **Encryption**: `mqtt.encryption_enabled` is a separate toggle for whether protobuf payloads are sent encrypted. Set `mqtt.encryption_enabled = false` to send unencrypted protobuf. For plaintext JSON, set `mqtt.json_enabled = true` - JSON packets are always unencrypted, regardless of `encryption_enabled`. (JSON is not supported on nRF52 boards such as RAK WisBlock.)

Via CLI:

```
meshtastic --set mqtt.address localhost --set mqtt.enabled true --set mqtt.json_enabled true
# uplink is per-channel and defaults to off - enable it on the channel you want bridged:
meshtastic --ch-index 0 --ch-set uplink_enabled true
```

Step 5 - Verify packets are flowing

```
mosquitto_sub -h localhost -t 'msh/#' -v
```

With JSON enabled (`mqtt.json_enabled true`), text, position, and telemetry packets appear as readable JSON under `msh/REGION/2/json/...`. Without JSON, the payload under `msh/REGION/2/e/...` is raw protobuf (encrypted or unencrypted) and `mosquitto_sub` displays it as binary, not readable separate messages. If `mosquitto_sub` returns immediately or hangs with nothing at all, confirm `allow_anonymous true` is actually applied - Mosquitto 2.0 refuses anonymous connections by default; add `-u`/`-P` if you set a password.

Remote Access Options

- **LAN only**: Listen on `127.0.0.1` or LAN IP. Accessible only within your local network - simplest and most secure.
- **Internet-exposed with auth**: Set `listener 1883 0.0.0.0` with a password file. Open port 1883 in your router/firewall only if you need external access. Consider using TLS (Mosquitto supports it natively).
- **Cloud MQTT broker**: Point your nodes at EMQX Cloud, HiveMQ Cloud, or a self-hosted Mosquitto VPS. Multiple Pi gateways in different locations all publish to the same broker - gives you a unified view of all gateways from anywhere.

systemd Service for the Meshtastic Connection

If you run a Python script to bridge or monitor the mesh, create `/etc/systemd/system/mesh-bridge.service`. **Replace `youruser` below with the actual username you created when imaging** - since Pi OS Bookworm there is no default `pi` account, and the unit will fail to start if the user does not exist:

```
[Unit]
Description=Meshtastic mesh bridge
After=network.target mosquitto.service

[Service]
User=youruser
ExecStart=/usr/bin/python3 /home/youruser/mesh_bridge.py
Restart=on-failure
RestartSec=10

[Install]
WantedBy=multi-user.target
```

```
sudo systemctl enable mesh-bridge
sudo systemctl start mesh-bridge
```

Node-RED on the Same Pi

Install Node-RED for visual flow-based packet processing with zero additional cloud dependency:

```
bash <(curl -sL https://github.com/node-red/linux-installers/releases/latest/download/update-noc
```

Use the MQTT-in node subscribed to `msh/#` to receive all mesh packets, then add function nodes to parse JSON, filter by type, and push to dashboards, databases, or notification services. The Node-RED UI dashboard module provides a web-accessible map and message log without any external services.

Power Budget

Board	Idle / light-load power	Notes

Raspberry Pi 4 (2 GB)	~3.4 W	Idle/light-load figure, not peak. PoE HAT adds ~1 W; suitable for rooftop enclosure with PoE switch
Raspberry Pi 3B+	~2.9 W	Idle/light-load figure. Good balance of capability and power
Raspberry Pi Zero 2W	~0.9 W	Idle/light-load figure. Best for solar/battery; limited to single USB device, requires USB OTG adapter

Add ~0.5 - 1 W (average) for the connected Meshtastic node; TX bursts at +22 dBm draw more momentarily. The figures above are idle/light-load estimates, not peak. A 12 V/7 Ah SLA battery is ~84 Wh nominal, but SLA chemistry should only be discharged to ~50% depth to avoid damage (~42 Wh usable), and a 12 V->5 V buck converter loses ~10-15%. Against a Zero 2W + node load of roughly 1.5-1.9 W, expect about **20-25 hours** of runtime, not 40+. Reaching 40+ hours would require deep discharge that shortens SLA lifespan.

Revision #5
Created 2026-05-03 04:10:59 UTC by Mesh America Admin
Updated 2026-06-10 05:32:41 UTC by Mesh America Admin