Home Assistant Integration via MQTT

Overview

Integrating Meshtastic into Home Assistant unlocks powerful home automation possibilities: track family members on a mesh map, get alerts when a node goes offline, and trigger smart-home actions based on mesh events. The integration uses MQTT as the transport layer, with Mosquitto as the local broker.

Step 1: Install and Configure Mosquitto

In Home Assistant, navigate to **Settings → Add-ons → Add-on Store** and install the *Mosquitto broker* add-on. After installation, open its configuration tab and add a user:

```
logins: - username: meshuser
 password: yourpassword
```

Start the add-on and enable *Start on boot*. Note your Home Assistant host IP - your Meshtastic node will connect to this address.

Step 2: Configure Your Meshtastic Node for MQTT

Open the Meshtastic app, go to **Radio Configuration → MQTT** and set:

- **MQTT Server Address:** your HA host IP (e.g. `192.168.1.100`)

- **Username / Password:** the credentials you set above
- **Root Topic:** `msh/US` (or your region)
- **JSON output:** enable *JSON enabled* so the node publishes the human-readable JSON used by the sensors below.
- **Uplink Enabled:** on. **Leave Downlink DISABLED** for Home Assistant monitoring - you only need uplink to read data. Downlink lets MQTT inject messages back into your RF mesh, which is an injection risk if the broker is ever reachable by anyone else; only enable it if you specifically intend to send messages from HA into the mesh.

Also ensure your node has WiFi configured under **Radio Configuration → Network**.

Step 3: MQTT Sensor YAML in Home Assistant

Add the following to your `configuration.yaml` (adjust the region, channel name, and node IDs as needed). The Meshtastic JSON topic format is `msh/REGION/2/json/CHANNELNAME/!gatewayNodeId` - the segment after `json` is the **channel name** (e.g. `LongFast`), not the packet type. The packet type (`telemetry`, `position`, etc.) is a field *inside* the JSON payload (`value_json.type`), so we subscribe with a `+` wildcard for the channel and filter on the type in each template:

```
mqtt:
 sensor: - name: "Node !a1b2c3d4 Battery"
 # Wildcard channel segment; replace !a1b2c3d4 with the publishing gateway node's id
 state_topic: "msh/US/2/json/+/!a1b2c3d4"
 value_template: >-
 {% if value_json.type == 'telemetry' %}
 {{ value_json.payload.battery_level }}
 {% else %}{{ states('sensor.node_a1b2c3d4_battery') }}{% endif %}
 unit_of_measurement: "%"
 device_class: battery - name: "Node !a1b2c3d4 Latitude"
 state_topic: "msh/US/2/json/+/!a1b2c3d4"
 value_template: >-
 {% if value_json.type == 'position' %}
 {{ value_json.payload.latitude_i | float / 1e7 }}
 {% else %}{{ states('sensor.node_a1b2c3d4_latitude') }}{% endif %} - name: "Node !a1b2c3d4
Longitude"
 state_topic: "msh/US/2/json/+/!a1b2c3d4"
 value_template: >-
 {% if value_json.type == 'position' %}
```

```
  {{ value_json.payload.longitude_i | float / 1e7 }}
  {% else %}{{ states('sensor.node_a1b2c3d4_longitude') }}{% endif %}
```

Note: the last topic segment is the *gateway* node that published to MQTT, which is not necessarily the originating node. Verify the exact JSON payload field names (e.g. `battery_level`, `latitude_i`) against the JSON your firmware actually emits, and see this wiki's *MQTT Topic Structure and Packet Format* page for the canonical topic layout.

Reload your YAML configuration after saving.

Step 4: Automations

Alert when a node hasn't been heard in 30 minutes:

```
automation: - alias: "Mesh node offline alert"
 trigger:
 platform: state
 entity_id: sensor.node_a1b2c3d4_battery
 to: unavailable
 for: "00:30:00"
 action:
 service: notify.mobile_app_your_phone
 data:
 message: "Node !a1b2c3d4 has not reported in 30 minutes"
```

Notify on critical battery:

```
 - alias: "Mesh node low battery"
 trigger:
 platform: numeric_state
 entity_id: sensor.node_a1b2c3d4_battery
 below: 15
 action:
 service: notify.mobile_app_your_phone
 data:
 message: "Node !a1b2c3d4 battery critical: {{ states('sensor.node_a1b2c3d4_battery') }}%"
```

Trigger lights when a family member arrives home via mesh position:

```
- alias: "Welcome home via mesh"
trigger:
platform: zone
entity_id: device_tracker.mesh_family_member
zone: zone.home
event: enter
action:
service: light.turn_on
target:
entity_id: light.porch
```

To use zone-based presence, create a `device_tracker` that updates from the latitude/longitude sensors using a template or the MQTT device tracker integration.

Step 5: Lovelace Dashboard

For a mesh map view, install the **Map card** in Lovelace. Add a card of type `map` and reference your device tracker entities. You can also use the *auto-entities* HACS card to dynamically list all mesh node sensors. For richer visualization, some community members use Grafana with the InfluxDB integration to feed in MQTT telemetry.

Revision #5
Created 2026-05-03 06:43:25 UTC by Mesh America Admin
Updated 2026-06-09 12:09:39 UTC by Mesh America Admin