MQTT Integration for Sensor Data

MQTT Integration for Sensor Data

Meshtastic includes a built-in MQTT bridge that can publish all mesh traffic - including telemetry sensor packets - to an MQTT broker. This enables real-time integration with Home Assistant, InfluxDB, Grafana, and other data platforms without any custom firmware modifications.

Enabling the MQTT Bridge on a Gateway Node

The MQTT bridge runs on any Meshtastic node with a direct internet connection. **JSON output over MQTT is only supported on ESP32 gateways** (such as the T-Beam or Heltec V3); it is **not** supported on the nRF52 platform (RAK4631, etc.), so an nRF52 node cannot itself be the JSON-MQTT gateway - use an ESP32 gateway or a host-side bridge. Configure the bridge via the Meshtastic CLI or the Python API:

```
meshtastic --set mqtt.address mqtt.example.com
meshtastic --set mqtt.username meshuser
meshtastic --set mqtt.password s3cr3t
meshtastic --set mqtt.root msh/region/us-east
meshtastic --set mqtt.enabled true
meshtastic --set mqtt.json_enabled true # publish decoded JSON, not raw protobuf
meshtastic --set mqtt.tls_enabled true # strongly recommended for production
```

Uplink and downlink are configured per-channel, e.g. `meshtastic --ch-index 0 --ch-set uplink_enabled true`.

The gateway node will now publish received packets to the topic `msh/<REGION>/2/json/<CHANNEL_NAME>/<USER_ID>` (the protobuf equivalent is `msh/<REGION>/2/e/<CHANNEL_NAME>/<USER_ID>`). Note that the 4th segment is the **channel name** and the final segment is the gateway's **user/node ID** - the packet type (e.g. telemetry) is **not** a topic

segment; it is carried in the `type` field inside the JSON body.

JSON Packet Format

A typical decoded telemetry JSON payload looks like this. Telemetry fields are flat, snake_case, under `payload`:

```
{
 "from": 1234567890,
 "to": 4294967295,
 "type": "telemetry",
 "payload": {
 "temperature": 22.4,
 "relative_humidity": 58.2,
 "barometric_pressure": 1013.7,
 "air_util_tx": 0.4
 },
 "timestamp": 1714500000,
 "channel": 0,
 "sender": "!499602d2"
}
```

The documented top-level fields are `id`, `channel`, `from`, `payload`, `sender`, `timestamp`, `to`, and `type`. There are **no** top-level `rssi`/`snr` fields in Meshtastic's JSON output - if you need signal metrics, add them with a downstream processor.

Node-RED Integration

Node-RED provides a low-code flow editor ideal for parsing and routing Meshtastic telemetry. A typical flow consists of:

1. **MQTT In node** - subscribe to `msh/+/2/json/#` (do not subscribe to bare `msh/#`, which also catches the binary `2/e/` topics that the JSON parser chokes on), then filter on the JSON `type` field == `"telemetry"` in a function node, since the portnum is not a topic level
2. **JSON node** - parse the payload string to a JavaScript object
3. **Function node** - extract fields, add tags (node name, location)
4. **InfluxDB Out node** or **Home Assistant node** - write the data to your chosen store

Home Assistant MQTT Sensor Configuration

Add the following to your Home Assistant `configuration.yaml` to create sensors for a Meshtastic node named `SensorNode-01`. Set the `state_topic` to match your actual `msh/<REGION>/2/json/<CHANNEL_NAME>/<USER_ID>` topic, or use a wildcard such as `msh/+/2/json/+/+` and filter in the `value_template`:

```
mqtt:
 sensor:
 - name: "SensorNode-01 Temperature"
 state_topic: "msh/+/2/json/+/+"
 value_template: "{{ value_json.payload.temperature }}"
 unit_of_measurement: "°C"
 device_class: temperature
 - name: "SensorNode-01 Humidity"
 state_topic: "msh/+/2/json/+/+"
 value_template: "{{ value_json.payload.relative_humidity }}"
 unit_of_measurement: "%"
 device_class: humidity
 - name: "SensorNode-01 Pressure"
 state_topic: "msh/+/2/json/+/+"
 value_template: "{{ value_json.payload.barometric_pressure }}"
 unit_of_measurement: "hPa"
 device_class: pressure
```

Restart Home Assistant after editing the configuration. The sensors will populate with live data as new telemetry packets arrive via the MQTT bridge.

Grafana Dashboard for Long-Term Trending

For historical analysis, write telemetry data to InfluxDB (v2 recommended) and visualise with Grafana:

1. Create an InfluxDB bucket named `mesh_telemetry`.
2. Use the Node-RED InfluxDB Out node (or the Python `influxdb-client` library) to write measurements with tags `node_id` and `node_name`.

3. In Grafana, add InfluxDB as a data source and create a dashboard with time-series panels for temperature, humidity, and pressure. Use a 7-day or 30-day range to identify trends, calibration drift, or equipment failure.
4. Configure Grafana alerting to notify via email or Slack when temperature exceeds a threshold or a sensor node stops reporting (absence of data alert).

Revision #3
Created 2026-05-03 06:11:40 UTC by Mesh America Admin
Updated 2026-06-10 03:20:29 UTC by Mesh America Admin