MQTT to InfluxDB and Grafana

Architecture Overview

The full telemetry pipeline runs: **Meshtastic node → MQTT broker → Telegraf (or Python subscriber) → InfluxDB 2.x → Grafana dashboard**. Each component is independently replaceable, so you can swap Telegraf for a custom Python script without touching InfluxDB or Grafana. This pipeline requires the gateway to publish decoded JSON (`mqtt.json_enabled true`); note that JSON-over-MQTT is not supported on nRF52 gateways (RAK4631, etc.) - use an ESP32 gateway.

InfluxDB 2.x Setup

Install InfluxDB 2.x on a server or Raspberry Pi:

```
# Debian / Ubuntu / Raspberry Pi OS
wget https://dl.influxdata.com/influxdb/releases/influxdb2-2.7.1-amd64.deb
sudo dpkg -i influxdb2-2.7.1-amd64.deb
sudo systemctl enable --now influxdb
```

After installation, visit `http://<host>:8086` to complete the setup wizard. Create an **organisation** (e.g. `meshamerica`), a **bucket** (e.g. `meshtastic`), and generate an **API token** with write access to that bucket. Store the token - you will need it for both Telegraf and the Python client.

Telegraf MQTT Consumer

Telegraf is a convenient path from MQTT to InfluxDB. Install it, then add a stanza to `/etc/telegraf/telegraf.conf`. Subscribe only to the JSON subtree - the bare `msh/#` wildcard also catches the binary protobuf `2/e/` topics, which the JSON parser cannot decode:

```
[[inputs.mqtt_consumer]]
 servers = ["tcp://localhost:1883"]
 topics = ["msh/+/2/json/#"]
 qos = 0
 data_format = "json_v2"

 [[inputs.mqtt_consumer.json_v2]]
 [[inputs.mqtt_consumer.json_v2.object]]
 path = "payload"
 tags = []
 # flattens the nested payload object (temperature, relative_humidity, ...)

[[outputs.influxdb_v2]]
 urls = ["http://localhost:8086"]
 token = "YOUR_INFLUXDB_TOKEN"
 org = "meshamerica"
 bucket = "meshtastic"
```

Meshtastic's JSON nests the numeric metrics under a `payload` object, so a plain `data_format = "json"` will not reach temperature/humidity by default - you must use the `json_v2` parser (or a path/query) to flatten `payload.*`, or use the Python subscriber below for full control. Ensure `mqtt.json_enabled true` is set on the gateway. Restart Telegraf after any config change: `sudo systemctl restart telegraf`.

Python Subscriber (Alternative)

If you need to apply custom logic, use a Python subscriber instead. For a JSON workflow (`mqtt.json_enabled true`) the gateway already publishes decoded JSON, so no protobuf module is needed - parse the JSON directly. (If you ever do need the protobuf `ServiceEnvelope`, the current import path is `from meshtastic.protobuf import mesh_pb2`.)

```
pip install paho-mqtt influxdb-client
```

```
import paho.mqtt.client as mqtt
from influxdb_client import InfluxDBClient, Point
from influxdb_client.client.write_api import SYNCHRONOUS
import json, os

INFLUX_URL = "http://localhost:8086"
```

```
INFLUX_TOKEN = os.environ["INFLUX_TOKEN"]
INFLUX_ORG = "meshamerica"
INFLUX_BUCKET = "meshtastic"

client_db = InfluxDBClient(url=INFLUX_URL, token=INFLUX_TOKEN, org=INFLUX_ORG)
write_api = client_db.write_api(write_options=SYNCHRONOUS)

# Optional: map node IDs to long names by caching NODEINFO_APP messages
# (payload.longname keyed by "from"); Meshtastic JSON has no top-level "fromName".
node_names = {}

def on_message(client, userdata, msg):
 try:
 envelope = json.loads(msg.payload)
 node_id = envelope.get("from", "unknown")
 # NODEINFO packets carry the long name in payload.longname
 if envelope.get("type") == "nodeinfo":
 ln = envelope.get("payload", {}).get("longname")
 if ln:
 node_names[node_id] = ln
 return
 if envelope.get("type") != "telemetry":
 return
 node_name = node_names.get(node_id, node_id)
 tel = envelope.get("payload", {}) # telemetry fields are flat, snake_case
 point = (
 Point("telemetry")
 .tag("node_id", node_id)
 .tag("node_name", node_name)
 .field("temperature", tel.get("temperature"))
 .field("humidity", tel.get("relative_humidity"))
 .field("battery_voltage", tel.get("voltage"))
 )
 write_api.write(bucket=INFLUX_BUCKET, record=point)
 except Exception as e:
 print(f"Parse error: {e}")

mqttc = mqtt.Client()
mqttc.on_message = on_message
mqttc.connect("localhost", 1883)
```

```
mqttc.subscribe("msh/+/2/json/#")
mqttc.loop_forever()
```

Note: Meshtastic's JSON envelope has **no** top-level `rxSnr`/`rxRssi` (or `snr`/`rssi`) fields, and telemetry metrics are flat snake_case under `payload` (e.g. `relative_humidity`, not `relativeHumidity`). Add signal metrics only if a downstream processor explicitly supplies them.

InfluxDB Line Protocol

Whether you use Telegraf or Python, each telemetry reading lands in InfluxDB as a **measurement** with tags and fields:

```
telemetry,node_id=!abc12345,node_name=ridge-repeater
temperature=22.4,humidity=61.0,battery_voltage=3.91 1746230400000000000
```

- **Measurement:** `telemetry`
- **Tags:** `node_id`, `node_name` (indexed, low-cardinality)
- **Fields:** `temperature`, `humidity`, `battery_voltage` (add `snr`/`rssi` only if a downstream processor supplies them; they are not in the gateway's JSON)

Grafana Setup

Install Grafana on the same host or a separate machine, then add InfluxDB as a data source using the **Flux** query language:

1. Configuration → Data Sources → Add data source → InfluxDB
2. Set Query Language to **Flux**
3. URL: `http://localhost:8086`
4. Paste your InfluxDB org, bucket, and token

Recommended dashboard panels:

- Temperature over time (line graph per node)
- Battery voltage trends (multi-node time series)
- Node count (stat panel - unique node_id values in last hour)
- Signal quality map (table of last RSSI/SNR per node)

Example Flux query - temperature trend for last 24 hours:

```
from(bucket: "meshtastic")
 |> range(start: -24h)
 |> filter(fn: (r) => r._measurement == "telemetry" and r._field == "temperature")
 |> aggregateWindow(every: 5m, fn: mean, createEmpty: false)
```

Retention Policies and Downsampling

In InfluxDB 2.x, retention is set per bucket. For raw telemetry data set the bucket retention to **90 days**. Create a second bucket (e.g. `meshtastic_hourly`) with indefinite retention and use a Flux task to downsample:

```
// Task: run every 1h - downsample telemetry to hourly averages
option task = { name: "downsample_telemetry", every: 1h }

from(bucket: "meshtastic")
 |> range(start: -2h)
 |> filter(fn: (r) => r._measurement == "telemetry")
 |> aggregateWindow(every: 1h, fn: mean, createEmpty: false)
 |> to(bucket: "meshtastic_hourly", org: "meshamerica")
```

Alerting

Grafana alerting fires when conditions are met. Useful rules for mesh deployments:

- **Low battery:** `battery_voltage < 3.5` for any node - triggers SMS or email alert.
- **Temperature exceeded:** `temperature > 50` - useful for enclosure health monitoring in summer.
- **Node offline:** no telemetry from a given `node_id` for more than N hours - check the "Last seen" derived field using a Flux `last()` query against the current time.

Revision #3
Created 2026-05-03 05:29:10 UTC by Mesh America Admin
Updated 2026-06-10 03:21:31 UTC by Mesh America Admin