Day 3 - IoT Systems - Open Source Cloud Solutions

Details on the used Cloud Infrastructure

Cloud Data Processing with NIG

This workshop walks you through using the **NIG stack** — Node-RED, InfluxDB and Grafana — to collect, store and visualise data. By the end you will have published numbers to your **own** MQTT broker, subscribed to them, cleaned them up, stored them in a time series database, and plotted them on a dashboard.

You don't need any hardware. We'll generate test values inside Node-RED itself. Once the pattern is clear, swap the test source for a real sensor and the rest of the flow stays the same.

Your instructor has already started an instance for you and will hand you the URLs and login. This page is only about *using* the three tools together.

The three tools, briefly

- **Node-RED** is a flow-based programming tool. You wire *nodes* (coloured blocks) into *flows*. Messages travel from one node to the next, carrying a payload that each node can read, modify or react to. We use it as the glue that pulls data in (MQTT, HTTP, serial, …), transforms it, and pushes it out (to a database, a dashboard, another device).
- **InfluxDB** is a time series database — a database optimised for storing measurements with timestamps. We write our cleaned-up values into it.
- **Grafana** is a visualisation and dashboarding tool. It reads from InfluxDB (and many other sources) and turns the data into graphs, gauges, alerts and dashboards.

This combination — sometimes called the **NIG stack** — is one of the standard ways to do IoT data processing.

What you'll build

A complete data pipeline:

```
[inject node]──►[mqtt out]──┐
                            │ test/value
[aedes broker]◄─────────────┘
```

```
                                  │ test/value
[mqtt in]──►[clean payload]──┴──►[influxdb out]──►[ Grafana panel ]
```

- **inject** — a button you click to fire a test number into the flow.
- **mqtt out** — publishes that number to a topic on your broker.
- **aedes broker** — your own MQTT broker, running inside your Node-RED.
- **mqtt in** — subscribes to the same topic.
- **clean payload** — ensures the data is stored as a number, not a string.
- **influxdb out** — writes it to InfluxDB.
- **Grafana** — reads it back and plots it.

This round-trip (publish → subscribe → store) is artificial for the workshop, but it's exactly the shape of a real IoT pipeline: a sensor publishes to MQTT, Node-RED subscribes, transforms and stores, Grafana visualises.

Your URLs and credentials

Your instructor will give you:

- A **username** and **password** (the same for Node-RED and Grafana).
- Your personal **Node-RED URL** and **Grafana URL**.
- A **host:port** to reach your MQTT broker from outside Node-RED (useful later if you want to connect a real device).

1. Node-RED basics

Open the Node-RED URL and log in. You'll see:

- **Editor canvas (middle)** — where you drag nodes and wire them up.
- **Palette (left)** — searchable list of every node type available to you.
- **Sidebar (right)** — tabs for *debug* messages, *help* for the selected node, *context* data, and more. The bug icon at the top opens the debug log; you'll live in this tab.
- **Deploy button (top right)** — *nothing happens until you press Deploy.* Changes you make are pending until then. This is the single most common “why isn't it working?” trap.

The message object is called `msg`. The actual data usually lives on `msg.payload`. Other useful fields appear as you go (`msg.topic` for MQTT, `msg.error` for errors, …), but `msg.payload` is the one you care about 90 % of the time.

Try it before continuing: drag an **inject** node and a **debug** node onto the canvas, wire them together, hit Deploy, and click the inject node's button. You should see a timestamp appear in the debug sidebar. That's a complete Node-RED flow.

2. Install the extra nodes

The base Node-RED installation doesn't include MQTT broker or InfluxDB support, so we install them from the palette manager.

1. Open the burger menu (top right) → **Manage palette**.
2. Switch to the **Install** tab.
3. Search for `node-red-contrib-aedes` → click **Install**. This adds an embedded MQTT broker.
4. Search for `node-red-contrib-influxdb` → click **Install**. This adds the InfluxDB client nodes.
5. Close the palette manager.

You should now see new nodes in the palette: an **aedes broker** node (under “network”) and **influxdb in** / **influxdb out** / **influxdb batch** nodes (under “storage”).

3. Host your own MQTT broker with Aedes

Aedes is a full MQTT broker that runs *inside* Node-RED itself. No separate Mosquitto installation, no external service — your Node-RED *is* the broker.

1. Drag an **aedes broker** node onto the canvas.
2. Double-click it. Set:
   - **MQTT Port:** 1881
   - Leave the rest at defaults (no auth — fine for the workshop, **not** for production).
3. Click **Done**.
4. Hit **Deploy**. The node should show a green “running” status underneath it.

Your broker is now reachable in two ways:

- **From inside your Node-RED** (the `mqtt in` and `mqtt out` nodes we'll add next) — at `localhost:1881`.
- **From outside** (your laptop, an ESP32, a phone app like *MQTT Explorer*) — at the **host:port** your instructor gave you. The number on the outside may differ from 1881 on the inside; just use what you were given.

Since the broker has no authentication, treat it as untrusted if you ever wire up a real device.

4. Publish a test value with an inject node

We'll generate fake “sensor” readings using an inject node and publish them to a topic on your broker.

1. Drag an **inject** node onto the canvas.
2. Double-click it. Set:
   - **msg.payload:** *number* → `23.5` (any number you like).
   - **Repeat:** *interval* → every `5 seconds` (so it auto-publishes; uncheck to publish only when you click the button).
3. Click **Done**.

4. Drag an **mqtt out** node onto the canvas.
5. Double-click it. Set:
    - **Server:** click the pencil icon to add a new broker config.
        - **Server:** `localhost`
        - **Port:** 1881
        - Leave the rest at defaults. Click **Add**.
    - **Topic:** `test/value`
    - **QoS:** `0`
    - **Retain:** *false*
6. Click **Done**.
7. Wire the inject node's output to the mqtt out node's input.
8. Hit **Deploy**.

Click the inject node's button (or wait for the 5-second interval). The mqtt out node should show "connected" underneath it. Your broker is now receiving messages on `test/value`.

5. Subscribe with mqtt in and debug

Publishing without subscribing is shouting into the void. Let's listen.

1. Drag an **mqtt in** node onto the canvas.
2. Double-click it. Set:
    - **Server:** select the broker config you just created (`localhost:1881`).
    - **Topic:** `test/value`
    - **QoS:** `0`
    - **Output:** *auto-detect (parsed JSON object, string or buffer)*
3. Click **Done**.
4. Drag a **debug** node onto the canvas. Leave it at defaults (it'll show `msg.payload` in the sidebar).
5. Wire `mqtt in` → `debug`.
6. Hit **Deploy**.
7. Open the debug sidebar (the bug icon on the right). Within five seconds you should see `23.5` appear, over and over.

Congratulations — you've just sent a message through your own broker and received it back. The MQTT layer works.

If nothing shows up, check (in this order): (1) is the debug node enabled? (the small green dot next to it should be lit), (2) is the mqtt in node "connected"? (status underneath), (3) is the topic in mqtt in *exactly* the same as in mqtt out — including upper/lowercase? (4) did you Deploy?

6. Clean the payload so it stores as a number

This step looks fussy but it's important. InfluxDB cares about *types*: a value stored as the string

“23.5” cannot be averaged, charted on a numeric axis, or compared to thresholds. It needs to be a *number*.

When the mqtt in node is set to *auto-detect*, a payload that looks like a number is already parsed as one — but only if it *looks* numeric. Sensors that send “23.5°C” or “value: 23.5” would land as strings. To be safe and explicit, convert it yourself.

1. Drag a **change** node onto the canvas (it's in the “function” group, blue).
2. Double-click it. Add a rule:
   - **Set** `msg.payload` **to the value** `$number(payload)`
   - Set the type to **expression** (the *J:* dropdown), not *string*.
3. Click **Done**.
4. Wire `mqtt in` → `change` → `debug` (re-route through the change node).
5. Deploy and check the debug sidebar. The value should still appear, but now look at the small grey label next to it — it should say `number`, not `string`.

The `$number()` expression is a JSONata function that coerces whatever it's given into a number, throwing an error if it can't. This catches malformed data at the right place: the flow stops with a visible error instead of silently writing garbage to the database.

> **For real sensor data with multiple fields** — e.g. a payload like `{“temperature”: 23.5, “humidity”: 60}` — keep `msg.payload` as an object instead of a single number. Each key becomes its own InfluxDB field automatically (see the next step). The cleaning step then becomes “make sure each value is a number”, which you can do with a small **function** node:

```
msg.payload = {
    temperature: Number(msg.payload.temperature),
    humidity: Number(msg.payload.humidity)
};
return msg;
```

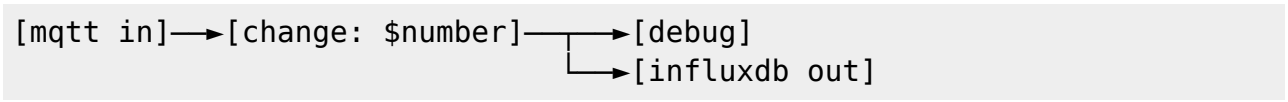
7. Save it to InfluxDB

The `node-red-contrib-influxdb` package added an **influxdb out** node that takes `msg.payload` and writes it to the database.

You're connecting to **InfluxDB 1.x**, which is much simpler to configure than 2.x or 3.x — no tokens, no organisations, just a database name and a username/password.

1. Drag an **influxdb out** node onto the canvas.
2. Double-click it. Set:
   - **Server:** click the pencil icon to add a new server config.
     - **Version:** `1.x`
     - **Host:** `influxdb`
     - **Port:** `8086`
     - **Database:** `db`
     - **Username:** *your workshop username*
     - **Password:** *your workshop password*

        - Click **Add**.
    - **Measurement:** `workshop` (a *measurement* is roughly what a table is in SQL — pick any name; you'll use it again in Grafana).
3. Click **Done**.
4. Wire `change` → `influxdb out`. Your flow should now read:



5. Deploy.

The influxdb out node should show “connected” underneath. After a few seconds (and a few inject firings) your data is in the database.

Why is the host `influxdb` and not `localhost`?
InfluxDB doesn't run inside Node-RED — it's a separate service that Node-RED reaches over the network by its name, `influxdb`. From *inside* your Node-RED, `localhost` means “Node-RED itself”, which is why MQTT in/out earlier used `localhost` (the Aedes broker *is* inside Node-RED). For the database, use `influxdb`. The same hostname works in Grafana, in the next section.

A quick word on InfluxDB

You won't interact with InfluxDB's own UI in this workshop — InfluxDB 1.x doesn't ship a friendly one, and you'd query through Grafana anyway. Just know that:

- Data is organised into **measurements** (≈ tables). Each measurement has **fields** (the actual numbers, like `value` or `temperature`) and optional **tags** (string labels for filtering, e.g. `location=lab1`). Every point has a **timestamp**, which InfluxDB assigns automatically if you don't.
- In your case, the influxdb out node sent a single number — so InfluxDB stored it as a field called `value` in the `workshop` measurement. If you'd sent an object like `{temperature: 23.5, humidity: 60}`, each key would be its own field.

InfluxDB 1.x is end-of-life. We use it here because it's the simplest version to configure for a teaching environment. For anything you build after this workshop, **use TimescaleDB or InfluxDB 3.x instead.**

TimescaleDB is PostgreSQL with a time-series extension — if you already know SQL, you already know it. InfluxDB 3.x is a complete rewrite with a different query model again (SQL via Flight SQL); it's faster and actively maintained, but migrating 1.x dashboards/queries to it is non-trivial.

8. Visualise it in Grafana

Open your Grafana URL and log in with the same credentials as Node-RED.

8.1 The data source

A data source tells Grafana *where* to read data from. Yours may already be set up — if so, walk through this section anyway, because adding a data source is a skill you'll use on every Grafana instance you ever touch.

1. Burger menu (top left) → **Connections** → **Data sources**.
2. If “InfluxDB” is already in the list, click it to inspect the settings below. Otherwise click **Add new data source** → **InfluxDB** and create one yourself.
3. Settings:
   - **Query language:** `InfluxQL` (because we run InfluxDB 1.x; the Flux option is for 2.x).
   - **HTTP – URL:** `http://influxdb:8086`
   - **InfluxDB Details – Database:** db
   - **User:** *your workshop username*
   - **Password:** *your workshop password*
   - **HTTP Method:** GET
4. Scroll down and click **Save & test**. You should get a green “datasource is working” message.

8.2 Create a dashboard

1. Burger menu → **Dashboards** → **New** → **New dashboard**.
2. Click **+ Add visualization**.
3. Pick your **InfluxDB** data source.

You're now in the panel editor. The graph fills the top; the bottom half has tabs for the **query**, with panel **options** on the right.

8.3 Build the query

For InfluxQL the query builder is point-and-click:

1. **FROM:** `workshop` (the measurement you wrote to in Node-RED).
2. **SELECT:** `field(value)` — keep the default `mean()` aggregator, or change it to `last()` if you'd rather see the raw last reading per time bucket.
3. **GROUP BY:** `time($__interval) fill(null)` (the default).

The graph should immediately show your injected values, one point every few seconds.

If you used multiple fields (the `{temperature, humidity}` object from the tip earlier), click **+** next to SELECT to add a second field, or add a second query (the **+ Query** button below) so each line is independent.

8.4 Tidy up the panel

On the right side panel, useful starting points:

- **Title** — give it a name like “Workshop test value”.
- **Visualization** — *Time series* is the default; *Stat* is nice for a single big-number readout, *Gauge* for a dial.
- **Standard options → Unit** — pick the right unit (°C, %, kWh, …). Grafana will format axes and tooltips automatically.
- **Standard options → Min/Max** — fix the Y-axis range if you have known sensor bounds.
- **Thresholds** — add coloured bands (e.g. red above 30) that show on the graph and in stat/gauge visualisations.

Click **Apply** (top right) to drop the panel onto the dashboard, then **Save dashboard** to persist it. Give it a name.

8.5 Time range

Top right of the dashboard, you can change the visible range (*Last 5 minutes*, *Last 24 hours*, …) and the auto-refresh interval. For a live demo of inject → MQTT → InfluxDB, “Last 5 minutes” with a 5-second refresh is satisfying.

Where to go from here

You now have a working end-to-end pipeline. The natural next steps:

- **Replace the inject node with a real sensor.** Anything that can publish MQTT works: an ESP32 with Tasmota firmware, a Raspberry Pi running a Python script, a phone app like *IoT MQTT Panel*. Point it at the broker host:port your instructor gave you and publish to your topic.
- **Add more topics.** One broker can carry hundreds of topics, organised hierarchically (`lab1/room2/temperature`). MQTT wildcards (`lab1/#`) let one `mqtt in` subscribe to a whole subtree.
- **Use the function node** for transformations more complex than `$number()` — converting raw ADC counts to engineering units, smoothing, alerting on out-of-range values.
- **Alerting in Grafana.** Add an *alert rule* to a panel — Grafana can email, post to Slack, or hit a webhook when a threshold is crossed.
- **Export your flows.** Burger menu → *Export* in Node-RED gives you a JSON file. Keep it somewhere safe; it's your own offline copy if you ever want to recreate the flow elsewhere.

Further reading

- Node-RED documentation (start here if you've never used Node-RED before)
- node-red-contrib-aedes
- node-red-contrib-influxdb
- InfluxDB 1.x documentation (current workshop)
- InfluxDB 3.x documentation (recommended for new projects)

- TimescaleDB documentation (recommended for new projects)
- Grafana documentation
- MQTT protocol overview