

# Hands-On Guide to LoRaWAN IoT Data Pipelines and Data Visualization

*IntERLab*

---

## Abstract

This hands-on session guides participants through the process of creating a data pipeline for LoRaWAN sensor networks. Building upon the foundation established in Part 1, participants will learn to capture data from The Things Network (TTN), store them efficiently in a SQLite database, and create real-time visualizations using Grafana. At the end of this session, participants will have implemented a functional end-to-end solution to collect, store, and visualize sensor data from LoRaWAN devices, enabling them to monitor environmental conditions in real time.

---

## Contents

|          |  |           |
|----------|--|-----------|
| <b>1</b> | <b>Introduction</b>                                  | <b>2</b>  |
| 1.1      | Objective . . . . .                                  | 2         |
| 1.2      | Prerequisites . . . . .                              | 2         |
| 1.3      | Hands-On Duration . . . . .                          | 2         |
| 1.4      | Required Materials . . . . .                         | 2         |
| 1.5      | Github Link . . . . .                                | 2         |
| <b>2</b> | <b>Data Storage with SQLite</b>                      | <b>3</b>  |
| 2.1      | SQLite Overview . . . . .                            | 3         |
| 2.2      | Creating the Database Schema . . . . .               | 3         |
| 2.3      | Enhanced MQTT Client with Database Storage . . . . . | 4         |
| <b>3</b> | <b>Visualization with Grafana</b>                    | <b>8</b>  |
| 3.1      | Installing Grafana . . . . .                         | 8         |
| 3.1.1    | For Windows . . . . .                                | 8         |
| 3.1.2    | For macOS . . . . .                                  | 8         |
| 3.2      | Configuring Grafana . . . . .                        | 8         |
| 3.3      | Creating a Data Source . . . . .                     | 9         |
| 3.4      | Building a Dashboard . . . . .                       | 9         |
| <b>4</b> | <b>Exercises</b>                                     | <b>12</b> |
| 4.1      | Sensor Data Configuration . . . . .                  | 12        |
| 4.2      | Handling Multiple Sensors . . . . .                  | 12        |
| 4.3      | Data Retention Policies . . . . .                    | 12        |

# 1 Introduction

## 1.1 Objective

At the end of this one-hour workshop, participants will:

- Set up a data pipeline for LoRaWAN sensor readings
- Store TTN MQTT messages in a local SQLite database
- Install and configure Grafana for data visualization
- Create a real-time dashboard for monitoring sensor data

## 1.2 Prerequisites

- Completed Part 1 (**Hands-On Guide to LoRa & LoRaWAN with Pycom LoPy4**) of the LoRa & LoRaWAN hands-on session
- Successfully sending data from LoPy device to TTN
- Functioning MQTT client receiving data from TTN
- Basic knowledge of Python and SQL

## 1.3 Hands-On Duration

| Task  | Duration |
|---|----------|
| Setting up SQLite database and enhanced MQTT client | 20 min   |
| Installing and configuring Grafana                  | 15 min   |
| Creating dashboards and visualizations              | 20 min   |
| Q&A   | 5 min    |

Table 1: Expected duration of each task in the hands-on session

The expected duration of the hands-on session is 1 hour.

## 1.4 Required Materials

- Computer with Python installed
- Working TTN application and device from Part 1
- Internet connection

## 1.5 Github Link

The source code for this hands-on is available at <https://github.com/rmukhia/lora-lab-session>.

## 2 Data Storage with SQLite

The project is an expansion of the *mqtt* implementation from Part 1 of the hands-on. The LoPy will run the *lorawan* project from Part 1 to continue transmitting temperature readings to TTN. Participants may want to send unlimited LoRaWAN messages to the server by replacing

```
while i < 100:
```

```
with
```

```
while True:
```

in the *lorawan* code running on the LoPy device.

### 2.1 SQLite Overview

SQLite is a C library that provides a lightweight, disk-based database that does not require a separate server process. It is built into Python as the `sqlite3` module.

---

**Python comes with SQLite built in, so no additional installation is required. SQLite offers single-file database storage with zero configuration. For larger projects, other database systems are recommended**

---

Participants can verify their SQLite installation by opening a Python terminal and running:

```
>>> import sqlite3
>>> print(sqlite3.version)
```

For more information on SQLite in Python, refer to the official documentation: <https://docs.python.org/3/library/sqlite3.html>

### 2.2 Creating the Database Schema

The following Python script creates a SQLite database to store sensor data. Create a file named *create\_db.py* with this content:

```
import sqlite3
import os

# Database file path
DB_FILE = "sensor_data.db"

# Check if database already exists
db_exists = os.path.exists(DB_FILE)

# Connect to database (creates it if it doesn't exist)
conn = sqlite3.connect(DB_FILE)
cursor = conn.cursor()

# Create tables if they don't exist
cursor.execute("""
```

```

CREATE TABLE IF NOT EXISTS temperature_readings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    device_id TEXT,
    temperature REAL,
    rssi INTEGER,
    snr REAL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (device_id) REFERENCES devices (device_id)
)
"""

# Commit changes and close connection
conn.commit()
conn.close()

```

```
print(f"Database created successfully at {DB_FILE}")
```

**Step 1.** Run this script to create the database:

```
python create_db.py
```

**Step 2.** (Optional) Verify that the script creates the *temperature\_readings* table using tools like

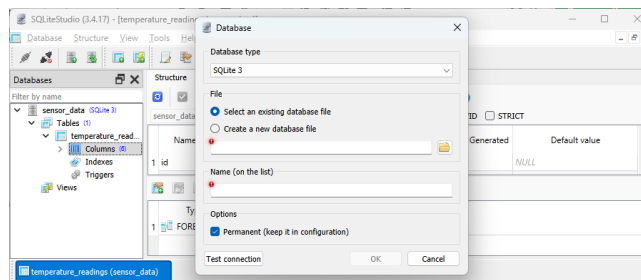


Figure 1: SQLiteStudio Interface

*SQLiteStudio* (<https://github.com/pawelsalawa/sqlitestudio/releases/>) or *DB Browser for SQLite* (<https://sqlitebrowser.org/dl/>). The portable version of these tools does not require installation. For SQLiteStudio (Figure 1), from the menu, select *Database* → *Add a Database* and select the *sensor\_data.db* file created by the above script.

## 2.3 Enhanced MQTT Client with Database Storage

Update the MQTT client from Part 1 to store the received data in the SQLite database. Create a file named *mqtt\_to\_sqlite.py*:

```

import paho.mqtt.client as mqtt
import json
import base64
import sqlite3
import time
import re

```

```

from datetime import datetime

# Database configuration
DB_FILE = "sensor_data.db"

# MQTT configuration
BROKER = "ttn.hazemon.in.th"
PORT = 1883
USERNAME = "your_app_name" # Replace with your TTN application name
PASSWORD = "your_api_key"   # Replace with your TTN API key
APP_NAME = USERNAME # Replace with your TTN application name
DEVICE_ID = "your_device_id" # Replace with your device ID

def extract_temperature(payload):
    """Extract temperature value from the payload string"""
    # Use regular expression to extract temperature value.
    match = re.search(r"Temperature: ([\d.]+)", payload)
    if match:
        return float(match.group(1))
    return None

def on_connect(client, userdata, flags, rc, properties=None):
    """Callback for when the client connects to the broker"""
    print(f"Connected with result code {rc}")
    # Subscribe to the topic for the device uplink messages
    client.subscribe(f"v3/{APP_NAME}/devices/{DEVICE_ID}/up")

def on_message(client, userdata, msg):
    """Callback for when a message is received from the broker"""
    try:
        # Decode the JSON payload from the message
        data = json.loads(msg.payload.decode())

        # Extract the base64 encoded payload
        payload_base64 = data["uplink_message"]["frm_payload"]

        # Decode the base64 payload to a UTF-8 string
        payload = base64.b64decode(payload_base64).decode()

        # Extract metadata
        rssi = data["uplink_message"]["rx_metadata"][0]["rssi"]
        snr = data["uplink_message"]["rx_metadata"][0]["snr"]

        # Extract temperature from payload
        temperature = extract_temperature(payload)

        if temperature is not None:

```

```

# Connect to the database
conn = sqlite3.connect(DB_FILE)
cursor = conn.cursor()

# Insert temperature reading
cursor.execute(
    "INSERT INTO temperature_readings (device_id, temperature, rssi, snr) VALUES
    (DEVICE_ID, temperature, rssi, snr)"
)

# Commit changes and close connection
conn.commit()
conn.close()

# Print the received message
print(f"Stored: Device={DEVICE_ID}, Temp={temperature}°C, RSSI={rssi}, SNR={snr}")
else:
    print(f"Could not extract temperature from payload: {payload}")

except Exception as e:
    print(f"Error processing message: {e}")

# Set up MQTT client
client = mqtt.Client(callback_api_version=mqtt.CallbackAPIVersion.VERSION2)
client.on_connect = on_connect
client.on_message = on_message

# Set authentication
client.username_pw_set(USERNAME, PASSWORD)

# Connect to the broker
client.connect(BROKER, PORT)

# Start the loop
print(f"Starting MQTT client, listening for data from device {DEVICE_ID}...")
print("Press Ctrl+C to stop")
client.loop_forever()

```

**Step 1.** Update the variables USERNAME, PASSWORD, APP\_NAME, and DEVICE\_ID variables with the TTN credentials and the TTN device information.

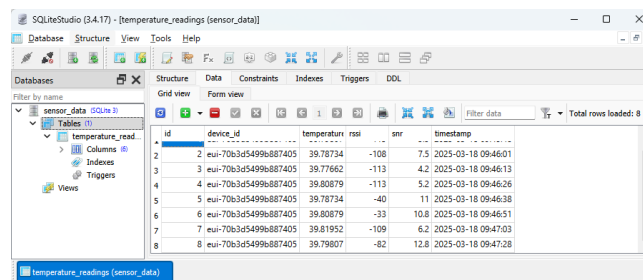
**Step 2.** Run the script to start capturing data:

```
python mqtt_to_sqlite.py
```

---

**This MQTT client connects to TTN, subscribes to device messages, extracts temperature data, and stores them in the SQLite database. Keep it running to continuously collect data while sending messages from the LoPy device.**

**Step 3.** (Optional) Verify the data stored in the SQLite database (Figure 2).



The screenshot shows the SQLiteStudio interface with the 'temperature\_readings' table selected. The table structure is as follows:

| id | device_id            | temperature | rssi | snr  | timestamp           |
|----|----------------------|-------------|------|------|---------------------|
| 2  | eui-70b3d5499b887405 | 39.78734    | -108 | 7.5  | 2025-03-18 09:46:01 |
| 3  | eui-70b3d5499b887405 | 39.77662    | -113 | 4.2  | 2025-03-18 09:46:13 |
| 4  | eui-70b3d5499b887405 | 39.80879    | -113 | 5.2  | 2025-03-18 09:46:26 |
| 5  | eui-70b3d5499b887405 | 39.78734    | -40  | 11   | 2025-03-18 09:46:38 |
| 6  | eui-70b3d5499b887405 | 39.80879    | -33  | 10.8 | 2025-03-18 09:46:51 |
| 7  | eui-70b3d5499b887405 | 39.81952    | -109 | 6.2  | 2025-03-18 09:47:03 |
| 8  | eui-70b3d5499b887405 | 39.79807    | -82  | 12.8 | 2025-03-18 09:47:28 |

Figure 2: Data Stored in SQLite Database

## 3 Visualization with Grafana

Grafana is an open source platform for monitoring and observability that allows users to query, visualize, and alert to metrics.

### 3.1 Installing Grafana

#### 3.1.1 For Windows

**Step 1.** Download and install Grafana from <https://grafana.com/grafana/download?edition=enterprise&platform=windows>.

If participants do not want to install Grafana, they can download the zip version, extract the contents to a directory of their choice, navigate to the extracted directory, and then run *grafana-server.exe*. This will open a *cmd* prompt and run Grafana.

#### 3.1.2 For macOS

**Step 1.** Download and install Grafana from <https://grafana.com/grafana/download?edition=enterprise&platform=mac>. To start the Grafana service, navigate to the extracted directory, and run the command:

```
./bin/grafana server
```

For more detailed installation instructions, refer to the official documentation: <https://grafana.com/docs/grafana/latest/installation/>

### 3.2 Configuring Grafana

Once Grafana is running, participants can access it through their web browser:

<http://localhost:3000>

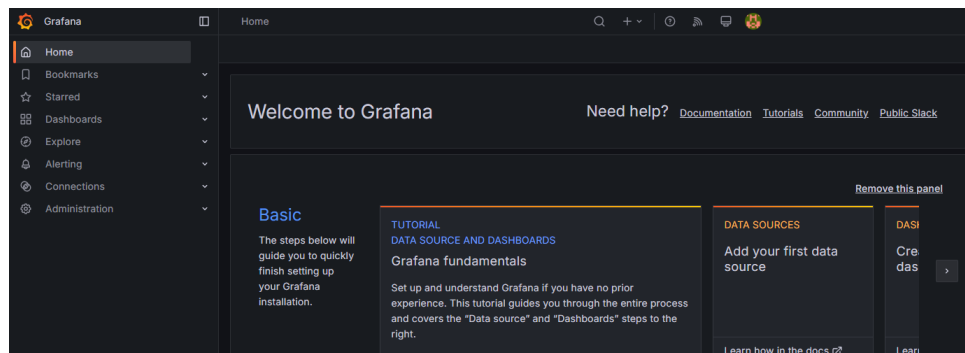


Figure 3: Grafana Landing Page

**Step 1.** Log in with the default credentials:

- Username: admin
- Password: admin



**Step 2.** When prompted, change the default password to a secure one. Users will be greeted with the landing page as shown in Figure 3.

### 3.3 Creating a Data Source

Connect Grafana to the SQLite database:

**Step 1.** Install the SQLite plugin for Grafana:

- Click on the gear icon (*Administration*) in the side menu
- Select *Plugins and data* → *Plugins*
- Search for *SQLite* and find the *Grafana SQLite Datasource* plugin
- Click *Install*

**Step 2.** Add a new data source:

- In the side menu, select *Connections* → *Data Sources*
- Click *Add data source*
- Search for and select "SQLite"
- Fill in the following details:
  - Name: LoRaWAN Sensor Data
  - Path: Enter the full path to the *sensor\_data.db* file
- Click *Save & Test* to verify the connection

For more information on Grafana data sources, refer to <https://grafana.com/docs/grafana/latest/datasources/>

### 3.4 Building a Dashboard

**Step 1.** Select *Dashboards* from the side menu, then choose *Create Dashboard*.

**Step 2.** Click "Add visualization"

**Step 3.** Select the *LoRaWAN Sensor Data* as the SQLite data source.

**Step 4.** Create a temperature time series panel:

- In the query editor, enter:

```
SELECT
  strftime('%s', timestamp) AS time,  -- Convert to Epoch timestamp
  temperature AS value,
  device_id AS metric
FROM temperature_readings;
```



Figure 4: LoRaWAN Sensor Dashboard. The range to display and update frequency can be set in the dashboard.

- In the right panel, make sure to select *Time series* for visualization, set the *Panel options* → *Title* to "Temperature Readings" and change the *Axis* → *Label* to "Celsius".
- Click on *Back to the dashboard* to leave the editing view of the visualization.

**Step 5.** Add a gauge panel to show the latest temperature:

- Click *Add* → *Add visualization*
- In the query editor, enter:

```
SELECT
  temperature as value
FROM temperature_readings
ORDER BY timestamp DESC
LIMIT 1
```

- Under visualization, select *Gauge*, set the panel title to "Current Temperature", set the *Standard options* → min value to 0 and max value to 50. Add *Thresholds* (e.g., 35 for warning, 45 for critical)
- To exit the visualization's editing view, select *Back to the dashboard*.

**Step 6.** Save the dashboard:

- Click the *Save dashboard* button in the top right, enter a name for the dashboard (e.g., "LoRaWAN Sensor Dashboard"), and click *Save*
- In the dashboard view, exit the edit mode by clicking *Exit edit*.

**Exercise:** Add visualization for Signal Strength to the dashboard. Try to make the dashboard look like Figure 4.

---

**The dashboard will now display real-time temperature data and signal strength metrics from the LoRaWAN devices. Grafana will automatically refresh the visualizations at regular intervals.**

---

For more information on creating Grafana dashboards, refer to: <https://grafana.com/docs/grafana/latest/dashboards/>

## 4 Exercises

Participants can attempt the following exercises if time is available.

### 4.1 Sensor Data Configuration

Transmit additional sensor data such as humidity, pressure, and luminosity along with temperature measurements and integrate all of these metrics into the dashboard. One option is to format the LoRaWAN payload as JSON; however, note that at SF12 only 11 bytes (AS923 with dwell time restriction of 400ms) are available for the payload, and JSON uses string-based formatting that increases the overall size. How would participants mitigate this limitation?

---

**Note:** LoRaWAN payloads are typically formatted using byte-level encoding to keep payload size to a minimum.

---

### 4.2 Handling Multiple Sensors

In this exercise, only one sensor is registered in the TTN application. How would participants extend this scenario to handle data from multiple sensors?

**Hint:** One strategy is to use MQTT wildcard topics to manage data streams from multiple sensors.

### 4.3 Data Retention Policies

To prevent the database from growing indefinitely, implement a data retention policy. Create a script named *cleanup\_data.py* that clears all data more than 15 minutes old. This script can be scheduled to run periodically using system task schedulers such as Windows Task Scheduler or a cron job on macOS or Linux to maintain optimal database performance.

---

**Note:** In a real-world scenario, data retention policies often involve transferring old data to a lower priority backup database while indexing the active data for efficient querying.

---