Setup Guide for Raspberry Pi IoT Project

1. Prerequisites

Hardware Requirements

1. Raspberry Pi Model:
  - Raspberry Pi 3, 4, or newer models are recommended. These models have a more powerful processor and sufficient memory, ideal for IoT projects like this one. This project was carried out using a Raspberry Pi 4 Model B - 4GB RAM.

2. DHT22 Sensor:
   - Sensor 1 Connections (GPIO 4):
     - VCC: Connect to the 3.3V or 5V pin on the Raspberry Pi.
     - GND: Connect to the GND (Ground) pin on the Raspberry Pi.
     - Data Pin: Connect to GPIO 4 (Pin 7) on the Raspberry Pi.
     - 10k Ohm Resistor: Connect between the Data pin and VCC as a pull-up resistor.

   - Sensor 2 Connections (GPIO 17):
     - VCC: Connect to the 3.3V or 5V pin on the Raspberry Pi.
     - GND: Connect to the GND pin on the Raspberry Pi.
     - Data Pin: Connect to GPIO 17 (Pin 11) on the Raspberry Pi.
     - 10k Ohm Resistor: Connect between the Data pin and VCC as a pull-up resistor.

3. Buzzer Connection:
   - Buzzer Pins:
     - Positive (Long leg): Connect to GPIO 22 (Pin 15) on the Raspberry Pi.
     - Negative (Short leg): Connect to the GND pin on the Raspberry Pi.
     - 1k Ohm Resistor: Connect between the Data pin and VCC as a pull-up resistor.

Software Requirements

- Raspberry Pi OS:
  - Ensure that Raspberry Pi OS (32-bit or 64-bit) is installed on the Raspberry Pi. The Raspberry Pi OS Lite version is sufficient, but the full version can be used if you require a graphical interface.

- Docker and Docker Compose Installation:
- To install Docker, run the following commands:

```
curl -sSL https://get.docker.com | sh
sudo usermod -aG docker pi
```

- To install Docker Compose:

```
sudo apt-get install -y libffi-dev libssl-dev
sudo apt-get install -y python3 python3-pip
sudo pip3 install docker-compose
```

- Once these steps are complete, restart your Raspberry Pi:

```
sudo reboot
```

2. Step-by-Step Guide

Project Directory Structure

Before we dive into the explanations of each file, let's take a closer look at the project directory structure:

```
Docker_Project
├── mqtt
│   └── config
│       └── mosquitto.conf
│   └── data
│   └── log
├── telegraf
│   └── telegraf.conf
├── docker-compose.yml
├── Dockerfile
├── iot_project.py
└── requirements.txt
```

This structure organizes the project into clear and logical sections, where each file serves a specific purpose in setting up the IoT project.

Dockerfile Explanation

The Dockerfile is the blueprint for creating the Docker image of your Python-based sensor application. It defines the environment and dependencies needed for the application to run smoothly inside a container. Here is a line-by-line breakdown:

FROM python:3.9
- This line sets the base image for your Docker container to Python 3.9, ensuring a clean environment with Python already installed.

WORKDIR /app
- This sets the working directory to /app inside the container. All subsequent commands will be executed in this directory, providing a consistent location for your application files.

COPY requirements.txt .
- This command copies your requirements.txt file from your local machine into the /app directory of the container.

RUN pip install --upgrade pip && apt-get update && apt-get install -y libgpiod2 libgpiod-dev && rm -rf /var/lib/apt/lists/*
- Upgrades pip and installs system libraries required for GPIO pin control, which is essential when interacting with hardware sensors on a Raspberry Pi.

RUN pip install --no-cache-dir -r requirements.txt
- Installs all the Python dependencies listed in your requirements.txt file, ensuring you always get the latest versions.

COPY iot_project.py /app/
- Copies the main Python script (iot_project.py) into the /app directory of the container.

CMD ["python", "iot_project.py"]
- This sets the command that will run when the container starts, initiating the sensor data collection and MQTT communication.

docker-compose.yml Explanation

The docker-compose.yml file defines and manages all the services that your IoT project requires, including MQTT broker, InfluxDB, Telegraf, Grafana, Node-RED, and your custom sensor application.

version: '3.8'
- Specifies the version of Docker Compose to be used.

Services Section

1. MQTT Broker (Mosquitto)
- Uses the latest version of the Mosquitto MQTT broker to handle communication between devices.

2. InfluxDB
- Specifies the InfluxDB version, initializes it with default settings, and maps the data storage location for persistence.

3. Telegraf
- Configures Telegraf to collect data from the MQTT broker and send it to InfluxDB.

4. Sensor Application (sensor-app)
- Builds the sensor application image, provides GPIO access, and ensures that the container has the necessary permissions for hardware control.

5. Grafana
- Uses Grafana for data visualization, mapping its dashboard port to the host for easy access.

6. Node-RED
- Deploys Node-RED for flow-based programming and IoT integration, with permissions to interact with the Raspberry Pi's GPIO pins.

Mosquitto Configuration File: mosquitto.conf

The mosquitto.conf file configures the Mosquitto MQTT broker for handling messages between devices and services.

persistence true
- Enables data persistence, which stores MQTT messages and retains them even after a restart.

listener 1883
- Configures the broker to listen on port 1883, the standard MQTT port.

allow_anonymous true
- Allows clients to connect to the broker without authentication, simplifying initial setup.

Telegraf Configuration File: telegraf.conf

The telegraf.conf file is used to configure the data collection from MQTT and send it to InfluxDB.

[[inputs.mqtt_consumer]]
- Specifies that Telegraf will consume data from the MQTT broker on specified topics.

[[outputs.influxdb_v2]]
- Configures Telegraf to send data to InfluxDB, setting authentication details and data storage location.

interval and flush_interval
- Sets how frequently data is collected from sensors and sent to InfluxDB, ensuring efficient data processing.

Detailed Explanation of iot_project.py for Raspberry Pi IoT Project

1. Importing Required Libraries

```
import time
import board
import adafruit_dht
import RPi.GPIO as GPIO
import paho.mqtt.client as mqtt
import json
import datetime
```

- time: Used for introducing delays in the code execution.
- board: Provides access to the Raspberry Pi's GPIO pins.
- adafruit_dht: Library to interact with DHT22 temperature and humidity sensors.
- RPi.GPIO: Enables control of the Raspberry Pi's GPIO pins.
- paho.mqtt.client: Python client library to interact with MQTT brokers.
- json: Used for encoding data into JSON format to send over MQTT.
- datetime: Helps to format timestamps for logging.

2. GPIO Pin Configurations

```
DHT_SENSOR1 = adafruit_dht.DHT22(board.D4)  # GPIO 4 #pin_no=7
DHT_SENSOR2 = adafruit_dht.DHT22(board.D17) # GPIO 17 #pin_no=11
BUZZER_PIN = 22  # GPIO 22
```

- DHT_SENSOR1 and DHT_SENSOR2: These lines set up the sensors on GPIO pins 4 and 17 respectively.
- BUZZER_PIN: The buzzer is connected to GPIO pin 22, and it will be used to provide audio alerts.

3. MQTT Configuration

```
MQTT_BROKER = "mqtt"  # Docker service name for MQTT
MQTT_PORT = 1883  # MQTT port
MQTT_TOPIC_SENSOR1 = "sensor1/data"
MQTT_TOPIC_SENSOR2 = "sensor2/data"
MQTT_TOPIC_ERROR = "sensor/error"
MQTT_TOPIC_CONTROL_GLOBAL = "sensor/control"  # Global control topic
MQTT_TOPIC_CONTROL_SENSOR1 = "sensor1/control"  # Sensor 1 control topic
MQTT_TOPIC_CONTROL_SENSOR2 = "sensor2/control"  # Sensor 2 control topic
```

- MQTT_BROKER: This is set to the MQTT service running in the Docker container.
- MQTT_TOPIC_SENSOR1 and MQTT_TOPIC_SENSOR2: Topics where sensor data will be published.
- MQTT_TOPIC_CONTROL_GLOBAL: This is a global control topic that overrides individual sensor controls.
- MQTT_TOPIC_CONTROL_SENSOR1 and MQTT_TOPIC_CONTROL_SENSOR2: These topics control the individual state of data collection for each sensor.

4. GPIO Setup

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER_PIN, GPIO.OUT)
```

- GPIO.setmode(GPIO.BCM): Sets the GPIO pin naming convention to BCM (Broadcom), which is recommended for Raspberry Pi projects.
- GPIO.setup(BUZZER_PIN, GPIO.OUT): Configures the buzzer pin as an output pin.

5. MQTT Client Setup

```
client = mqtt.Client()
```

- This creates an instance of the MQTT client that will be used to communicate with the broker.

6. Global Variables

```
collect_data_global = False  # Global data collection control
collect_data_sensor1 = False  # Sensor 1 data collection control
collect_data_sensor2 = False  # Sensor 2 data collection control
last_global_command = None   # Last global command
```

- These variables keep track of the state of data collection, both globally and for each individual sensor, and store the last global command received.

7. MQTT Connection and Message Handling

7.1 on_connect Callback Function

```
def on_connect(client, userdata, flags, rc):
   print(f"[{datetime.datetime.now()}] Connected with result code {rc}")
   client.subscribe([(MQTT_TOPIC_CONTROL_GLOBAL, 0),
                (MQTT_TOPIC_CONTROL_SENSOR1, 0),
                (MQTT_TOPIC_CONTROL_SENSOR2, 0)])  # Subscribe to control topics
```

- on_connect: This function is called when the client successfully connects to the MQTT broker. It subscribes to the control topics to listen for commands that will dictate the data flow.

7.2 on_message Callback Function

```
def on_message(client, userdata, message):
   global collect_data_global, collect_data_sensor1, collect_data_sensor2,
last_global_command
   msg = message.payload.decode('utf-8')
   topic = message.topic
   print(f"[{datetime.datetime.now()}] Received message: {msg} on topic: {topic}")
```

- on_message: This function handles incoming MQTT messages. It controls the logic of starting or stopping data collection based on the received commands.
- Global Control Handling: The global command takes precedence over individual sensor commands. If the global command is "stop," all data collection stops, regardless of individual settings.
- Individual Sensor Handling: Each sensor can be controlled independently as long as the global control is set to "start."

8. Publishing Data to MQTT

```
def publish_data(topic, message):
    result = client.publish(topic, message)
    if result.rc == mqtt.MQTT_ERR_SUCCESS:
        print(f"Successfully published to {topic}: {message}")
    else:
        print(f"Failed to publish to {topic}: {message}, Result: {result.rc}")
```

- publish_data: This function is responsible for sending data to the specified MQTT topic. It checks if the publishing operation was successful and logs the result.

9. Buzzer Control Functions

```
def buzz_on():
    GPIO.output(BUZZER_PIN, GPIO.HIGH)

def buzz_off():
    GPIO.output(BUZZER_PIN, GPIO.LOW)
```

- buzz_on: Activates the buzzer to signal an alert.
- buzz_off: Turns off the buzzer when no alerts are present.

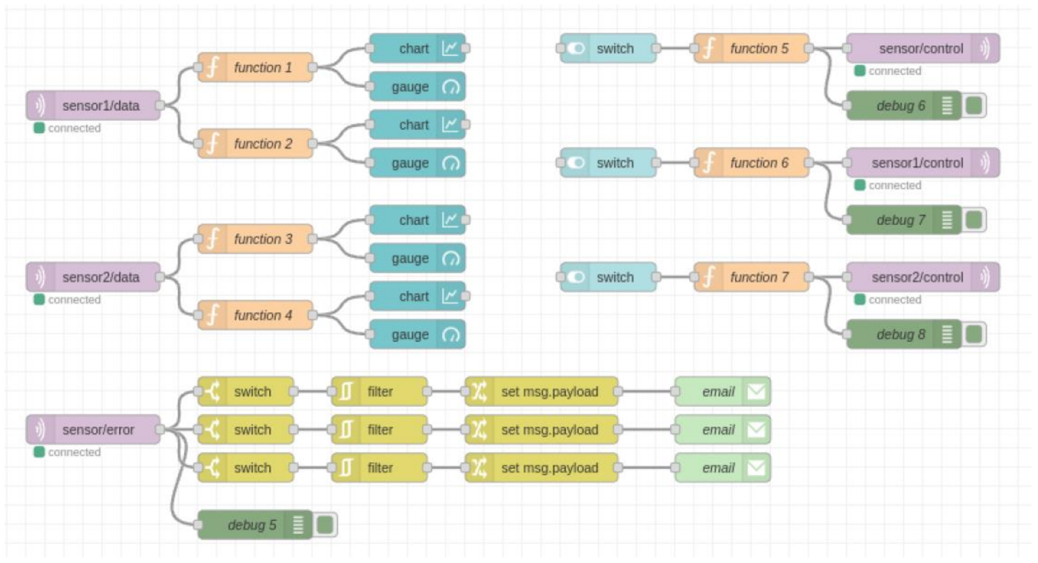
10. Main Loop for Data Collection

```
try:
    while True:
        alert_triggered = False  # Track if an alert is triggered
        # Only collect data if global control is True
        if collect_data_global:
            # Sensor 1 data collection
            if collect_data_sensor1:
                # Logic for collecting data from Sensor 1
            # Sensor 2 data collection
            if collect_data_sensor2:
                # Logic for collecting data from Sensor 2
```

- Main Data Collection Loop: This loop runs continuously, checking if the global control allows data collection. If so, it reads data from the sensors and publishes it to the MQTT broker.
- Alert Mechanism: If a sensor reading fails or the data is out of expected ranges, an alert is triggered, and the buzzer is activated.
- Graceful Exit: The loop will stop and clean up GPIO resources if interrupted manually.

Detailed Explanation of Node-RED Flow, InfluxDB, and Grafana Setup for Raspberry Pi IoT Project

Node-RED Flow Explanation



1. Sensor Data Input and Visualization

- MQTT Nodes for Sensor Data:
  - The MQTT input nodes labeled sensor1/data and sensor2/data receive data from the respective sensors.
  - The data payload is processed by different function nodes that extract either the temperature or humidity data from the JSON message.

- Function Nodes (function 1 to 4):
  - These nodes are used to format the incoming data for visualization. For example:
    - function 1 extracts the temperature from the sensor 1 data.
    - function 2 extracts the humidity from sensor 1 data.
    - function 3 and function 4 perform similar operations for sensor 2.

- Visualization:
  - Charts and Gauges: The processed data is then fed into chart and gauge nodes to visually represent the sensor readings in real-time. The charts display temperature trends over time, while the gauges show the current values.

2. Control Mechanism for Data Collection

- Switch Nodes and Control Topics:
  - The flow includes switch nodes that are connected to function nodes (function 5, 6, and 7) to control the data collection process for each sensor.
  - These switches toggle the state of data collection and publish a command (start or stop) to MQTT topics: sensor/control, sensor1/control, and sensor2/control.

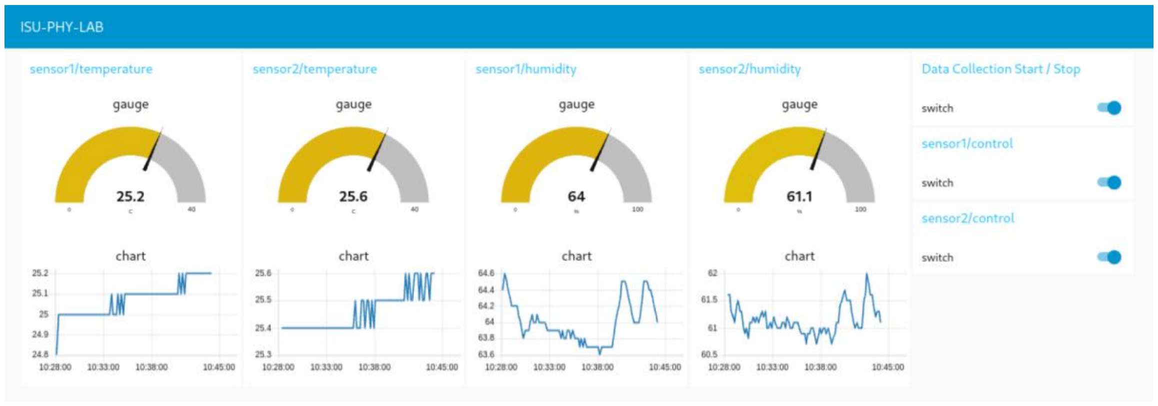
- Function Nodes for Control (function 5, 6, 7):
  - These nodes determine the payload to be sent based on the state of the switches. For example, if a switch is turned on, the function node will set the payload to start. If turned off, it sets the payload to stop.

3. Error Handling Flow

- MQTT Input Node for Errors (sensor/error):
  - This node monitors for any errors reported by the sensors.
- Switch Nodes for Error Filtering:
  - Three switch nodes filter the error messages based on the payload (Sensor 1 Error, Sensor 2 Error, Threshold Exceeded).
  - The errors are then processed by RBE (Report By Exception) nodes to avoid sending repetitive alerts for the same error.
- Change Nodes and Email Alerts:
  - Change nodes modify the payload to include a descriptive error message.
  - The processed error messages are then sent to the email node, which sends alerts to a specified email address whenever an error is detected.

4. Debugging Nodes

- Debug nodes (debug 5, debug 6, debug 7, and debug 8) are used throughout the flow to display messages in the debug sidebar of Node-RED, aiding in troubleshooting and monitoring the data flow.

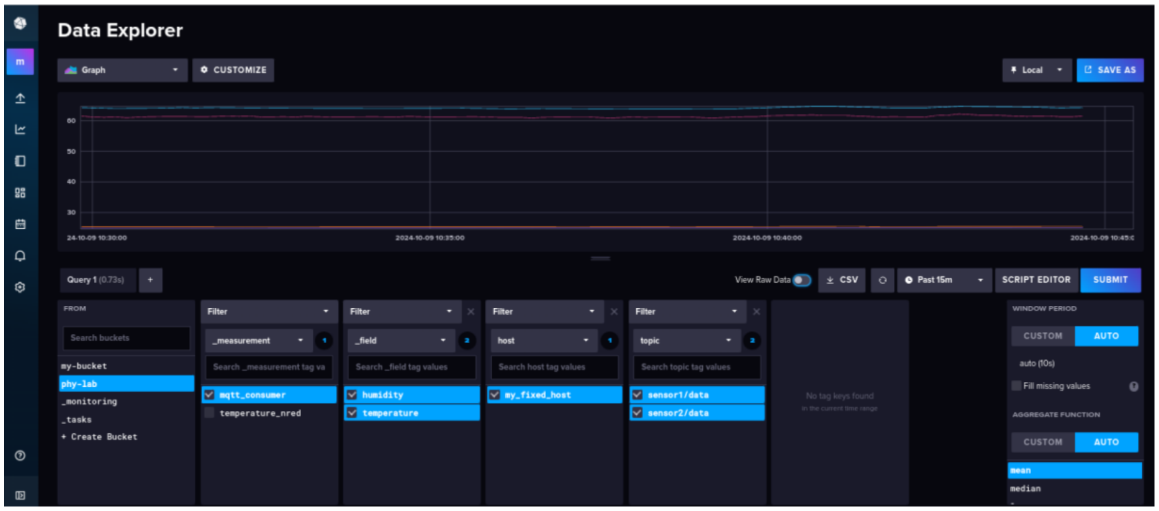


InfluxDB and Grafana Configuration

1. InfluxDB Setup

- Configuration:
  - The docker-compose.yml file specifies the setup for InfluxDB with environment variables like DOCKER_INFLUXDB_INIT_USERNAME, DOCKER_INFLUXDB_INIT_PASSWORD, and bucket (phy-lab).

- Telegraf Configuration:
  - Telegraf listens to MQTT topics and sends the data to InfluxDB using the settings defined in telegraf.conf.

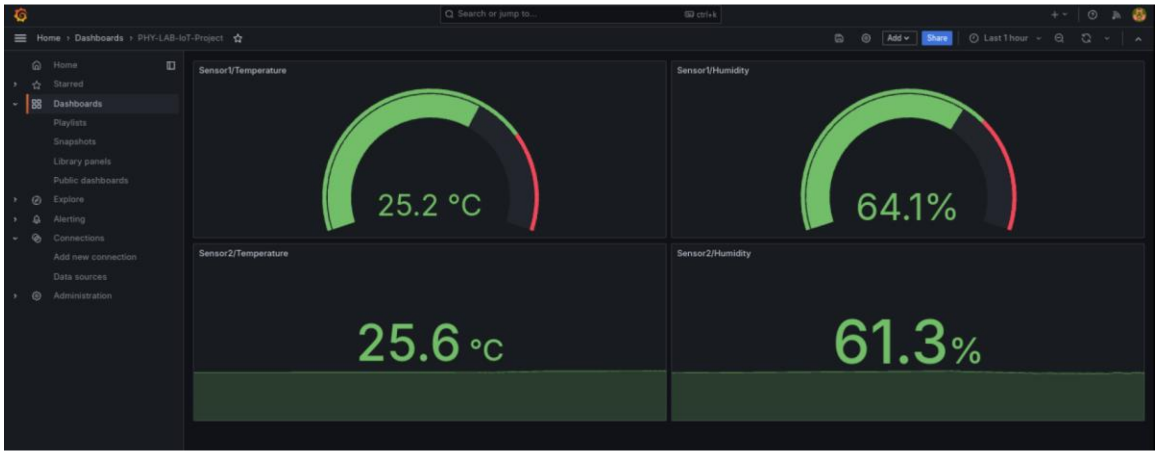


2. Grafana Setup

- Connecting Grafana to InfluxDB:
  - Access Grafana via the web UI and add InfluxDB as a data source by entering the URL, token, and bucket details.

- Creating Dashboards:
  - Create dashboards in Grafana by adding panels to display data from InfluxDB, using charts, gauges, and other visualization types.

- Alerting in Grafana:
  - Configure alerts in Grafana to notify when specific thresholds are exceeded using notification channels like email or messaging apps.



Conclusion, Tips, and Project Expansion for Raspberry Pi IoT Project

5. Conclusion and Tips

Possible Issues and Troubleshooting

1. Sensor Data Collection Issues
   - Problem: The sensors may fail to provide readings or show inaccurate values.
   - Possible Causes:
     - Faulty sensor wiring or poor connections.
     - High humidity or temperature beyond the sensor's operational range.
     - Software exceptions or GPIO pin conflicts.

- Troubleshooting Steps:
  - Check Connections: Verify that the sensors are correctly connected to the appropriate GPIO pins and that the pull-up resistors are properly in place.
  - Environmental Conditions: Ensure the environment is within the operational range of the DHT22 sensors (Temperature: -40°C to 80°C, Humidity: 0% to 100%).
  - Error Handling: Implement more robust error handling in the code to manage exceptions and automatically retry data collection if a failure occurs.

2. Debugging Tips
 - Use Node-RED Debug Nodes: Make use of debug nodes to display the incoming messages and error logs in the Node-RED debug panel. This makes it easier to identify where the problem lies.
 - Log MQTT Traffic: Use tools like mosquitto_sub to monitor the MQTT messages being published to and from the broker.

To view messages coming from the MQTT broker in the terminal, you can use the `mosquitto_sub` command. This command allows you to subscribe to the MQTT broker and see messages for a specific topic. Here's a step-by-step guide on how to do it:

Steps

1. Ensure the Docker Container is Running

If you have started the MQTT broker using your Docker Compose file, check whether the container is running:

```
docker ps
```

Make sure that a container named `mqtt` is listed as running in the output.

2. Run the Mosquitto Subscribe Command

To view messages from the MQTT broker, you can run the following command in your terminal:

```
docker exec -it mqtt mosquitto_sub -t <topic_name> -h localhost -p 1883
```

- `mqtt`: This is the container name (defined as `container_name` in your Docker Compose file).

- `-t <topic_name>`: The topic to which you want to subscribe. Change this to the topic you wish to monitor for MQTT messages. For example, to see all messages, you can use `#` as the topic:

```
docker exec -it mqtt mosquitto_sub -t '#' -h localhost -p 1883
```

Example

If you want to subscribe to a topic called `sensor_data`:

```
docker exec -it mqtt mosquitto_sub -t 'sensor_data' -h localhost -p 1883
```

This command will display all messages sent to the `sensor_data` topic in your terminal.

Using this command, you will be able to see all MQTT messages received for the subscribed topic in the terminal.

Project Expansion

1. Adding More Sensors
  - Steps to Integrate New Sensors:
    - Wiring: Connect the additional sensors to the Raspberry Pi GPIO pins, ensuring they do not conflict with the existing pins used by other devices.
    - Code Changes: Update the iot_project.py file to include the new sensors. For example, if adding a third DHT22 sensor:
      ```python
      DHT_SENSOR3 = adafruit_dht.DHT22(board.D27)  # GPIO 27
      MQTT_TOPIC_SENSOR3 = "sensor3/data"
      ```
    - Node-RED Configuration: Add new MQTT input nodes to listen for data from the new sensors and create additional function, chart, and gauge nodes to visualize this data.
  - MQTT Topic Naming: Use a consistent naming convention for new MQTT topics (e.g., sensor3/data, sensor3/control) to keep the data organized.

2. Enhancing Data Analysis in Grafana
  - Advanced Dashboards: Improve your Grafana dashboards by adding more visualization panels, such as:
    - Multi-Sensor Comparison: Create panels that display side-by-side comparisons of temperature and humidity readings from all sensors.
    - Correlation Graphs: Analyze how environmental changes in one sensor's readings might correlate with another's, using advanced graph features in Grafana.
  - Data Queries: Utilize InfluxDB's query language (Flux) to write more complex queries that provide insights like average, minimum, maximum, or even predictive trends of the data.
  - Create alerts based on these queries to notify you if specific thresholds or trends are detected in the sensor data.

3. Integrating Additional IoT Services
  - Webhook Integration: Use Node-RED to trigger webhooks based on specific sensor conditions (e.g., when temperature exceeds a certain limit, send data to a third-party API or service).
  - Edge Computing: Implement basic edge computing by processing and filtering the data directly on the Raspberry Pi before sending it to the cloud. This can reduce the amount of data transmitted and improve system responsiveness.
  - Cloud Integration: Consider integrating the project with cloud platforms like AWS IoT, Google Cloud IoT, or Azure IoT Hub for scalable data storage, machine learning, and more advanced analytics.