
IoT Project: ESP32 Data Collection and Grafana Visualization

Zain Ramzan

Table of Contents

1. Introduction	2
2. Objective	2
3. Methodology	2
3.1 Step 1: Hardware Setup	3
3.2 Step 2: Software Setup	5
3.3 Step 3: Docker Container Setup	7
3.4 Step 4: Telegraf Configuration	9
3.5 Step 5: InfluxDB Setup	10
3.6 Step 6: Grafana Installation and Configuration	11
4. Conclusion	15

1. Introduction

In this IoT project, we will create a system to collect data using an ESP32 microcontroller equipped with DHT (Temperature and Humidity) and Ultrasonic sensors. The collected data will be sent to an InfluxDB database using Telegraf and displayed in real-time on Grafana. We will containerize the data collection and transmission process using Docker for easy deployment and management.

2. Objective

The primary objective of this project is to set up an end-to-end IoT monitoring system that allows us to collect environmental data, store it in a database, and visualize it using Grafana.

3. Methodology

Step 1: Hardware Setup

1. Connect the DHT sensor to the ESP32's GPIO pins and the Ultrasonic sensor as needed.
2. Ensure that ESP32 is connected with Wi-Fi network.

Step 2: Software Setup

Arduino IDE Setup

1. Install the Arduino IDE.
2. Install the ESP32 board support package in Arduino IDE.
3. Prepare an Arduino sketch to read data from DHT and Ultrasonic sensors and send it to an MQTT broker.

4. Upload the sketch to ESP32.

MQTT Broker Setup

1. Install an MQTT broker (Mosquitto) on a server.
2. Configure the MQTT broker with appropriate topic.

Step 3: Docker Container Setup

1. Install Docker on system.
2. Create a Dockerfile for project that includes Python and required libraries.
3. Create a subscriber.py script that subscribes to the MQTT topics and sends data to Telegraf.

Step 4: Telegraf Configuration

1. Install and configure Telegraf to collect data from Python script.
2. Set up Telegraf to push the data to InfluxDB.

Step 5: InfluxDB Setup

1. Install and configure InfluxDB to receive and store the data from Telegraf.
2. Create a database and retention policy for environmental data.

Step 6: Grafana Installation and Configuration

1. Install Grafana on server.
2. Access the Grafana web interface and log in.
3. Add InfluxDB as a data source in Grafana.
4. Create a dashboard in Grafana and design it to display the environmental data with appropriate graphs and panels.

3.1 Step 1: Hardware Setup

1. Connect the DHT sensor to the ESP32's GPIO pins and the Ultrasonic sensor as needed.

Connection Setup for ESP32, DHT Sensor, and Ultrasonic Sensor

Breadboard Setup:

On the breadboard, we established a common Vcc (5V) and Ground (GND) connection for both the DHT and Ultrasonic sensors. This connection provides power to these sensors.

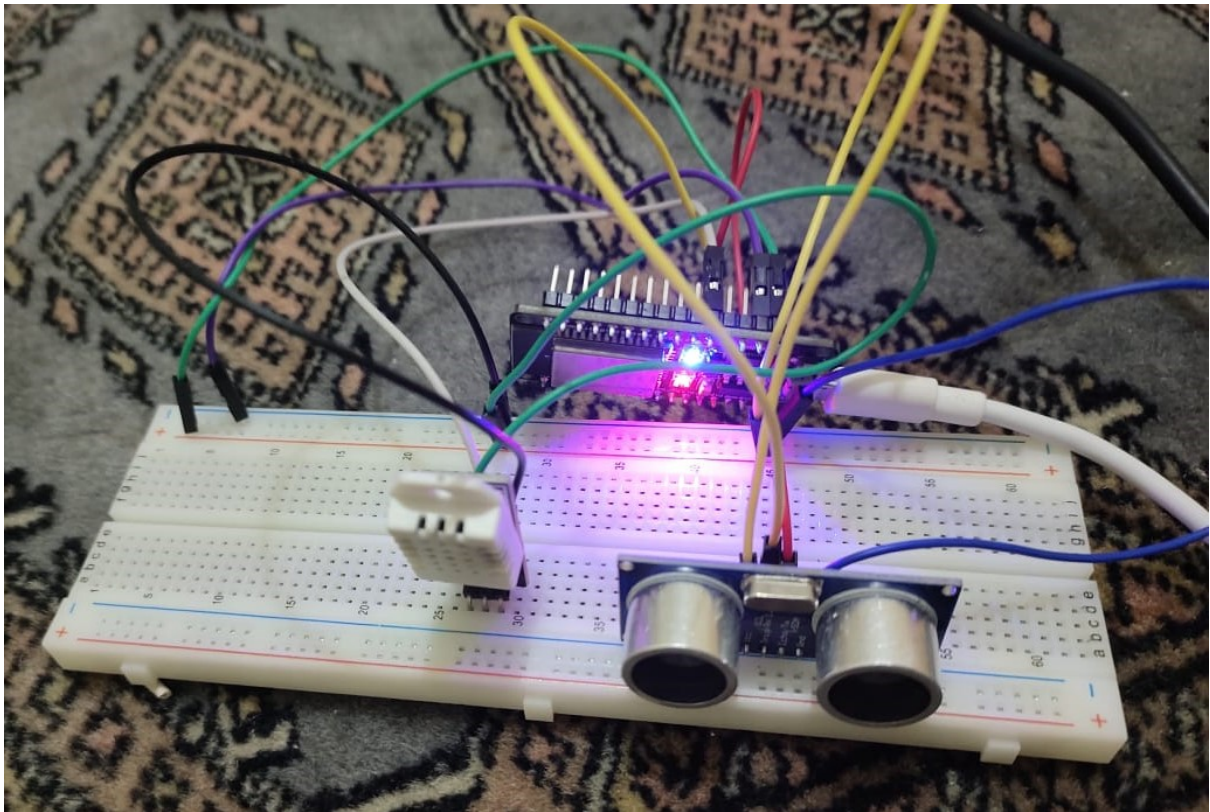
DHT Sensor (DHT22):

The DHT sensor has three pins: Vcc, Out, and GND. We have connected the Vcc pin of the DHT sensor to the 5V (Vcc) pin of the ESP32 on the breadboard. Then connected the GND pin of the DHT sensor to the Ground (GND) pin of the ESP32, which is also common on the breadboard. After that we have connected the Out pin of the DHT sensor to the D4 pin of the ESP32. This connection is used to read temperature and humidity data from the DHT sensor.

Ultrasonic Sensor (HC-SR04):

The Ultrasonic sensor has four pins: Vcc, Gnd, Trigger, and Echo. We have connected the Vcc pin of the Ultrasonic sensor to the 5V (Vcc) pin of the ESP32 on the breadboard. Then have connected the GND pin of the Ultrasonic sensor to the Ground (GND) pin of the ESP32, which is common on the breadboard. Then we have connected the Trigger pin of the Ultrasonic sensor to the D13 pin of the ESP32. This pin is responsible for triggering ultrasonic pulses. At the end, we have connected the Echo pin of the Ultrasonic sensor to the D12 pin of the ESP32. This pin receives the echo signal and is used to measure the time it takes for the signal to bounce back.

Below is the complete circuit image.



By establishing these connections, the ESP32 is configured to read data from the DHT sensor (temperature and humidity) and perform distance measurements using the Ultrasonic sensor. This setup allows for the collection of environmental data, which can then be transmitted, stored, and visualized as per the project's requirements.

2. Ensure that ESP32 is connected with Wi-Fi network.

Once the correct circuit connections were established, we proceeded to adapt an Arduino code to enable Wi-Fi connectivity for our ESP32. Additionally, we integrated the code for both the DHT and Ultrasonic sensors to facilitate the reading of sensor values. Below, you'll find a screenshot of the code for reference. To facilitate MQTT communication, we installed a relevant MQTT library.

```

#include <ArduinoMqttClient.h>
#include <WiFi.h>
#include <Arduino_JSON.h>

#include "my_secrets.h"

#include <Adafruit_Sensor.h>
#include <DHT.h>
#include <DHT_U.h>
#include <afstandssensor.h>
// code for reading sensor values
// AfstandsSensor(triggerPin, echoPin);
AfstandsSensor afstandssensor(13, 12); // Starter afstandssensoren på ben 13 og 12.
#define DHTPIN 4 // Digital pin connected to the DHT sensor
#define DHTTYPE DHT22 // DHT 22 (AM2302)

DHT_Unified dht(DHTPIN, DHTTYPE);

```

Compiling sketch

This code modification allows our ESP32 to connect to a Wi-Fi network and effectively gather data from the DHT and Ultrasonic sensors. These sensor readings are crucial for monitoring environmental conditions and will be instrumental in our IoT project's success.

3.2 Step 2: Software Setup

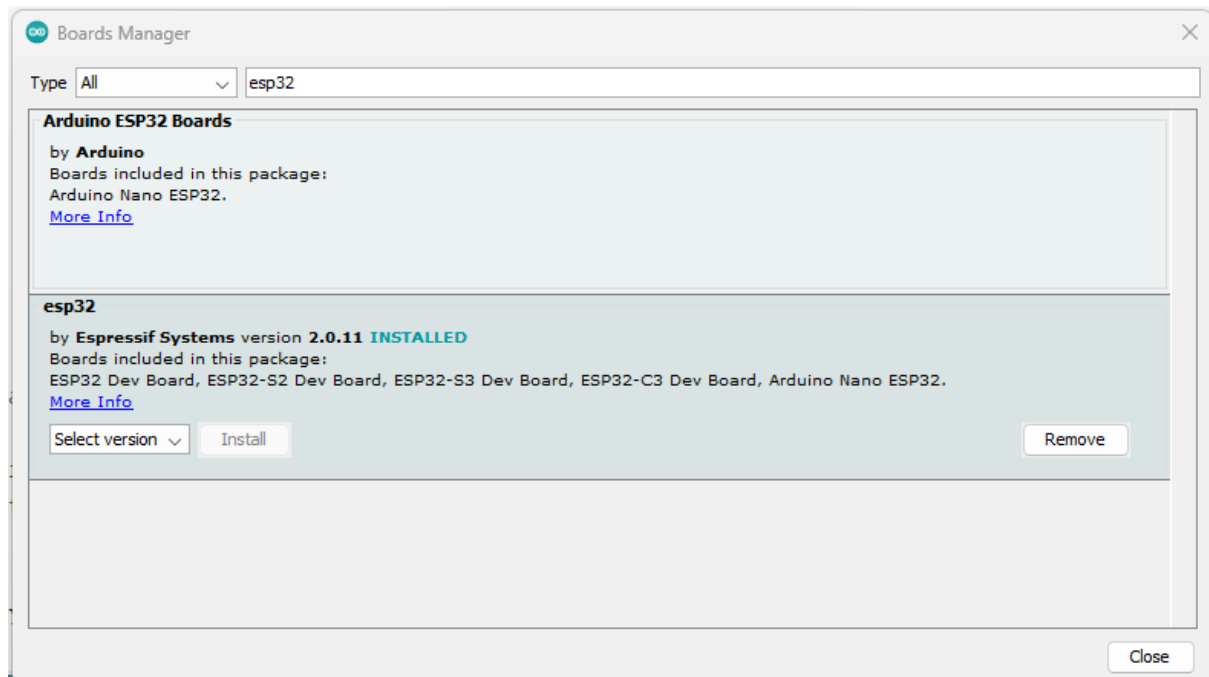
Arduino IDE Setup

1. Install the Arduino IDE.

Firstly we have installed Arduino IDE Version 1.8.19 in our system.

2. Install the ESP32 board support package in Arduino IDE.

After successful installation of Arduino IDE, we have downloaded the support package of Esp32 board (esp32 by Espressif Systems).



3. Prepare an Arduino sketch to read data from DHT and Ultrasonic sensors and send it to an MQTT broker.

We have prepared a code for both sensors, and also installed the library for MQTT.

```
Test1_mattino my_secrets.h
void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while(!Serial){delay(100);}

  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println("*****");
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
}
```

4. Upload the sketch to ESP32.

At the end we have uploaded the prepared sketch to our ESP32 board.



```
Test1_mqttino | Arduino 1.8.19
File Edit Sketch Tools Help

Test1_mqttino my_secrets.h

void setup() {
  //Initialize serial and wait for port to open:
  Serial.begin(9600);
  while(!Serial){delay(100);}

  // We start by connecting to a WiFi network

  Serial.println();
  Serial.println("*****");
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, pass);

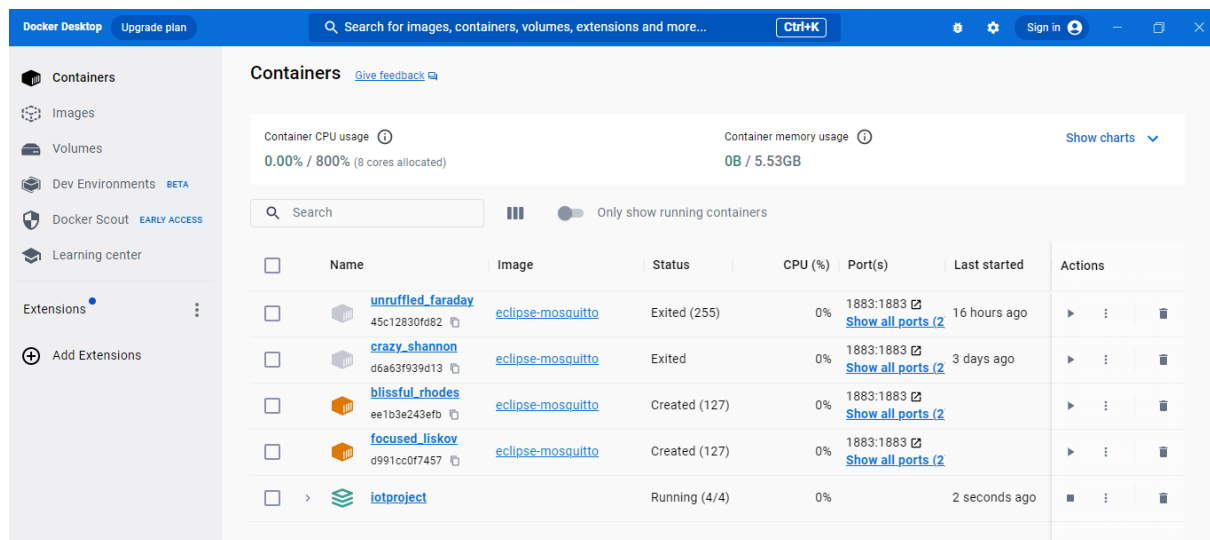
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
}
```

Compiling sketch...

3.3 Step 3: Docker Container Setup

1. Install Docker on system.

We have installed the docker desktop app in our system.



2. Create a Dockerfile for a project that includes Python and required libraries.

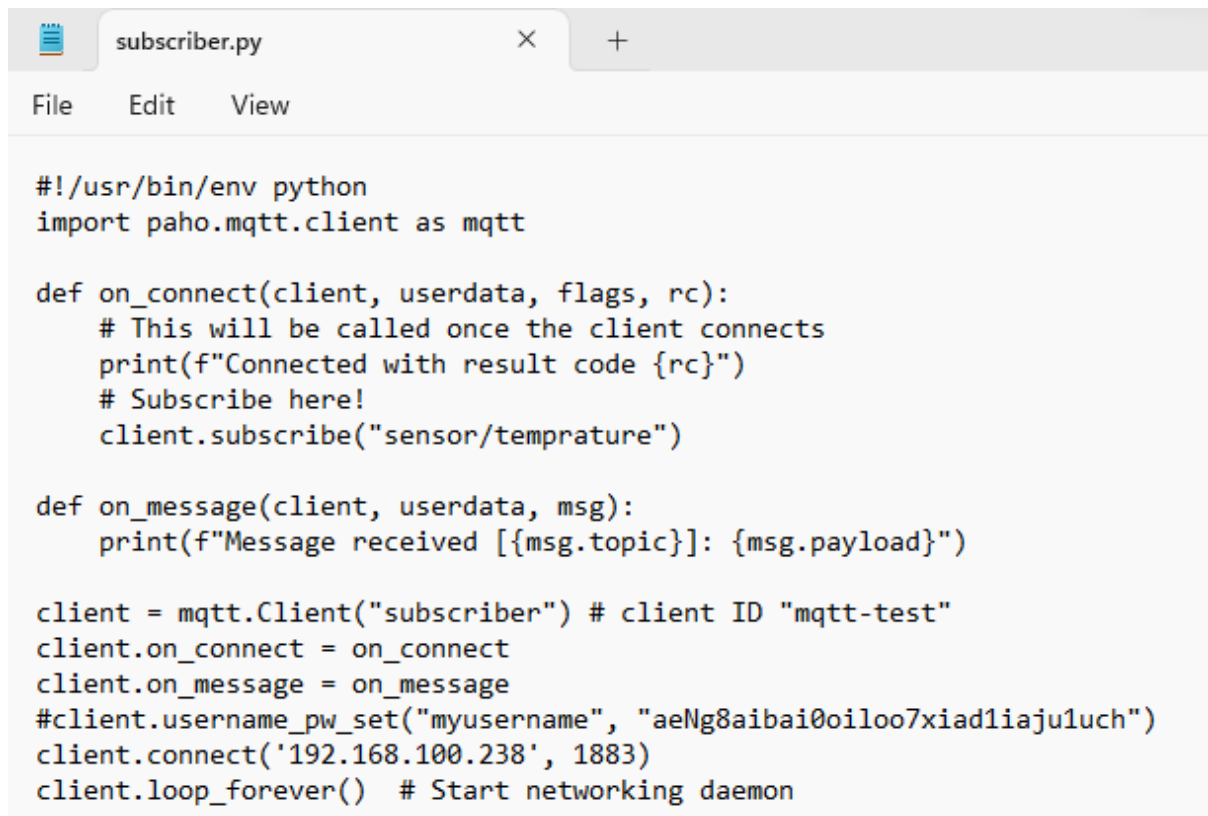
After installing docker, we have set up all the required libraries.

3. Create a subscriber.py script that subscribes to the MQTT topics and sends data to Telegraf.

After setting up the docker containers and pulling all the images such as grafana/grafana, eclipse-mosquitto, telegraf and influxdb. We have then prepared a subscriber.py code as shown below in the code screenshot.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
grafana/grafana	latest	854e0474bbc1	3 days ago	391MB
eclipse-mosquitto	latest	e94324731e38	12 days ago	13.3MB
telegraf	latest	1c4cc0044eb2	13 days ago	422MB
influxdb	latest	55857d99abe6	2 weeks ago	266MB

This Python script acts as an MQTT subscriber, using the paho.mqtt.client library. It connects to an MQTT broker at '192.168.100.238' on port 1883, subscribing to the "sensor/temprature" topic. Two callback functions, on_connect and on_message, handle events when the client connects and when messages arrive. The former prints the connection result, and the latter prints received messages' topic and payload. The script creates an MQTT client with the ID "subscriber" and starts a networking daemon with client.loop_forever() to maintain the connection and process incoming messages continuously. It provides a simple way to monitor and react to MQTT messages on the specified topic.



```
#!/usr/bin/env python
import paho.mqtt.client as mqtt

def on_connect(client, userdata, flags, rc):
    # This will be called once the client connects
    print(f"Connected with result code {rc}")
    # Subscribe here!
    client.subscribe("sensor/temperature")

def on_message(client, userdata, msg):
    print(f"Message received [{msg.topic}]: {msg.payload}")

client = mqtt.Client("subscriber") # client ID "mqtt-test"
client.on_connect = on_connect
client.on_message = on_message
#client.username_pw_set("myusername", "aeNg8aibai0oiloo7xiad1iaju1uch")
client.connect('192.168.100.238', 1883)
client.loop_forever() # Start networking daemon
```

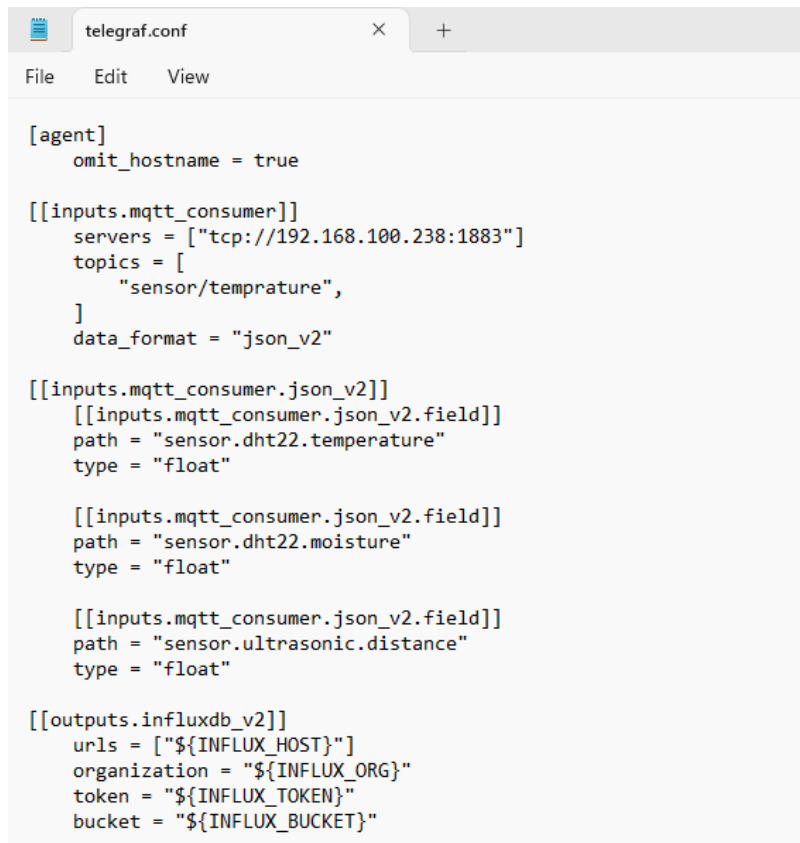
3.4 Step 4: Telegraf Configuration

1. Install and configure Telegraf to collect data from Python script and Set up Telegraf to push the data to InfluxDB.

First we have to pull the telegraf image through docker and then write the code as seen in the below screenshot.

The provided configuration is for Telegraf, a data collection and processing agent. It's set up to collect MQTT data from a broker at 192.168.100.238:1883 on the topic "sensor/temperature" in JSON format. It specifies how to parse and format this data, extracting temperature, moisture, and ultrasonic distance values. The collected data is then pushed to an InfluxDB instance hosted at 192.168.100.238:8086. The InfluxDB details (organization, bucket, and token) are passed as environment variables for security and flexibility. This configuration allows Telegraf to continuously gather

sensor data and store it in the specified InfluxDB database, enabling real-time monitoring and analysis of IoT-related information.

A screenshot of a web-based text editor window titled 'telegraf.conf'. The window has a menu bar with 'File', 'Edit', and 'View'. The main area contains the following configuration text:

```
[agent]
  omit_hostname = true

[[inputs.mqtt_consumer]]
  servers = ["tcp://192.168.100.238:1883"]
  topics = [
    "sensor/temprature",
  ]
  data_format = "json_v2"

[[inputs.mqtt_consumer.json_v2]]
  [[inputs.mqtt_consumer.json_v2.field]]
    path = "sensor.dht22.temperature"
    type = "float"

  [[inputs.mqtt_consumer.json_v2.field]]
    path = "sensor.dht22.moisture"
    type = "float"

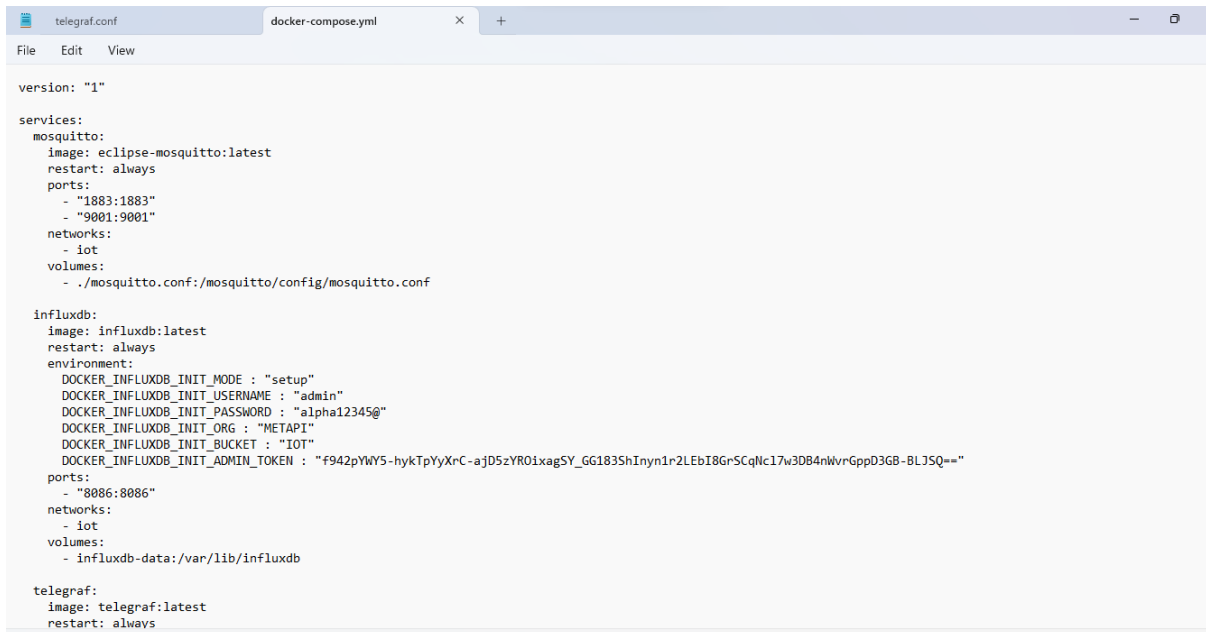
  [[inputs.mqtt_consumer.json_v2.field]]
    path = "sensor.ultrasonic.distance"
    type = "float"

[[outputs.influxdb_v2]]
  urls = ["${INFLUX_HOST}"]
  organization = "${INFLUX_ORG}"
  token = "${INFLUX_TOKEN}"
  bucket = "${INFLUX_BUCKET}"
```

3.5 Step 5: InfluxDB Setup

1. Install and configure InfluxDB to receive and store the data from Telegraf.

We have configured InfluxDB in docker-compose.yml as shown below. After that we start docker compose using “docker-compose up”.



```
version: "1"

services:
  mosquitto:
    image: eclipse-mosquitto:latest
    restart: always
    ports:
      - "1883:1883"
      - "9001:9001"
    networks:
      - iot
    volumes:
      - ./mosquitto.conf:/mosquitto/config/mosquitto.conf

  influxdb:
    image: influxdb:latest
    restart: always
    environment:
      DOCKER_INFLUXDB_INIT_MODE : "setup"
      DOCKER_INFLUXDB_INIT_USERNAME : "admin"
      DOCKER_INFLUXDB_INIT_PASSWORD : "alpha12345@"
      DOCKER_INFLUXDB_INIT_ORG : "METAPI"
      DOCKER_INFLUXDB_INIT_BUCKET : "IOT"
      DOCKER_INFLUXDB_INIT_ADMIN_TOKEN : "f942pYWY5-hykTpYyXrC-ajD5zYR0ixagSY_GG183ShInyn1r2LEbI8GrSCqNc17w3DB4nWvrGppD3GB-BLJSQ=="
    ports:
      - "8086:8086"
    networks:
      - iot
    volumes:
      - influxdb-data:/var/lib/influxdb

  telegraf:
    image: telegraf:latest
    restart: always
```

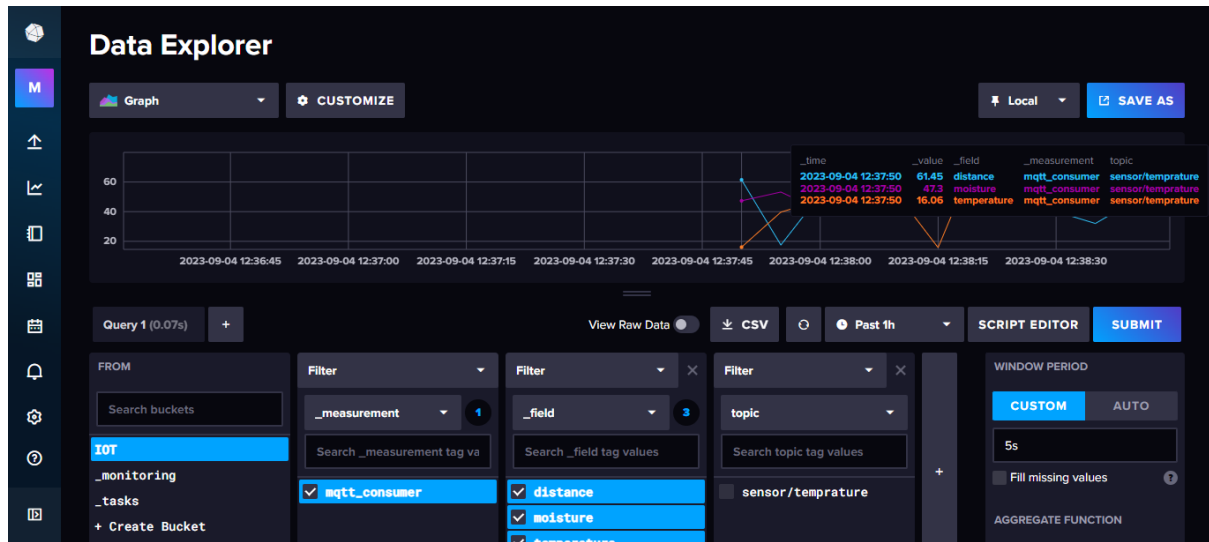
- In the web browser, we navigate to the InfluxDB user interface by going to <http://localhost:8086>.
- Then logged in using the credentials as defined in our docker-compose.yml file:

Username: admin

Password: alpha12345@

2. Create a database and retention policy for environmental data.

- Click on the "Data" menu on the left sidebar.
- Click the "Buckets" tab.
- Click the "Create Bucket" button.
- Now have filled the following details:
- Name: "IOT" (matching the one in our docker-compose.yml file).
- Retention: Define a retention policy, e.g., "30d" for 30 days.
- Click the "Create Bucket" button to create your database.



3.6 Step 6: Grafana Installation and Configuration

1. Install Grafana on the server.

We will pull grafana/grafana images in docker. And using below given code we have installed grafana on the server.

```
grafana:
  image: grafana/grafana:latest
  restart: always
  environment:
    GF_FEATURE_TOGGLES_ENABLE : publicDashboards
  ports:
    - "3000:3000"
  networks:
    - iot
  depends_on:
    - influxdb
  volumes:
    - grafana-data:/var/lib/grafana

networks:
  iot:

volumes:
  influxdb-data:
  grafana-data:
```

2. Access the Grafana web interface and log in.

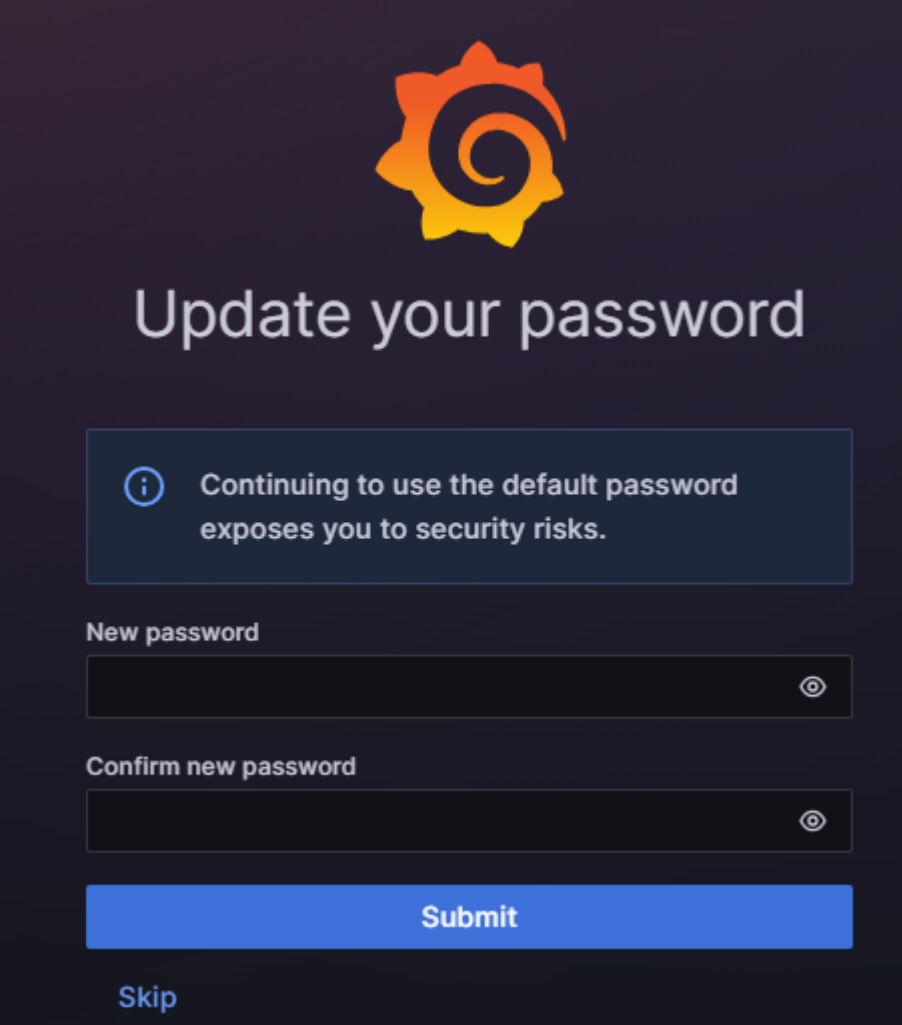
Open a web browser and navigate to `http://localhost:3000` (or the server's IP address and port if Grafana is not on the local machine).

Log In:

The default login credentials are:

- ❑ Username: admin
- ❑ Password: admin

We will be prompted to change the password on the first login.



The screenshot shows the Grafana 'Update your password' interface. At the top is the Grafana logo, a stylized orange and yellow gear with a spiral. Below the logo, the text 'Update your password' is displayed in a large, white, sans-serif font. A dark blue warning box with a white information icon contains the text: 'Continuing to use the default password exposes you to security risks.' Below this, there are two input fields: 'New password' and 'Confirm new password', both with white text and a white eye icon for toggling visibility. At the bottom, there is a large blue 'Submit' button and a smaller blue 'Skip' link.

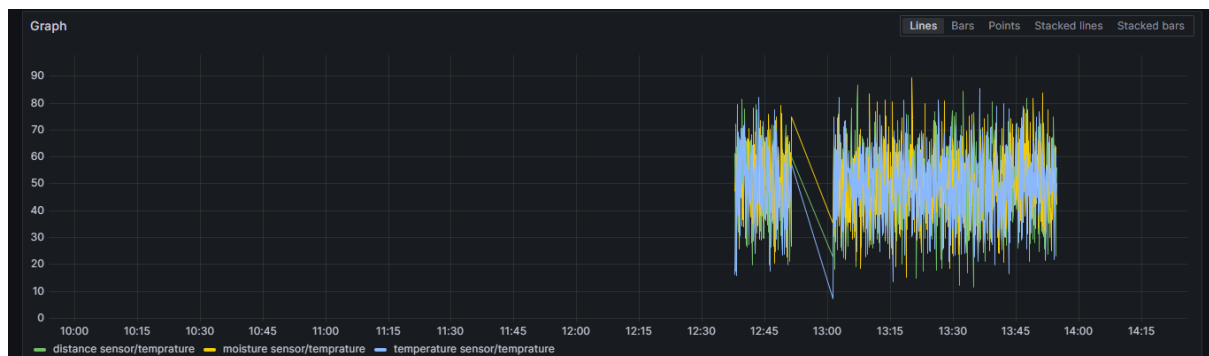
3. Add InfluxDB as a data source in Grafana.

After logging in we will Data Sources and add InfluxDB and then will add Query language as Flux, then will URL and other details as shown in the below screenshot.

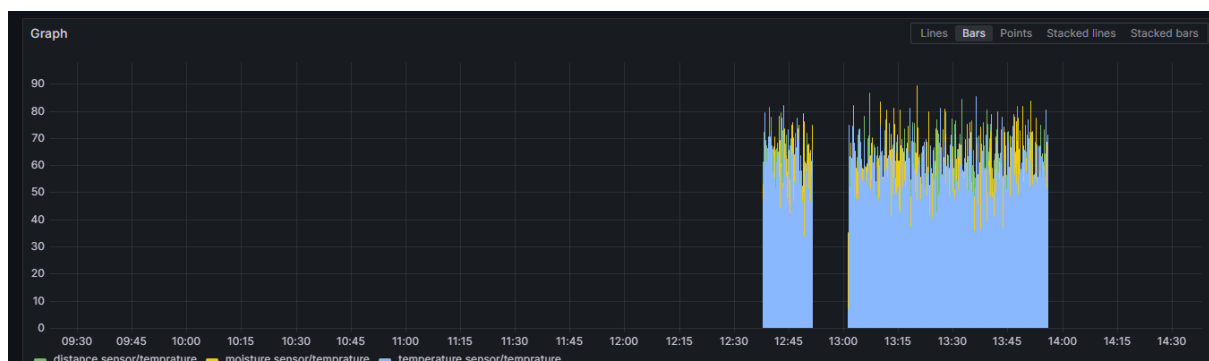
4. Create a dashboard in Grafana and design it to display the environmental data with appropriate graphs and panels.

At the end we will visualize our data in different kinds of plots in Grafana as shown in below screenshots and use it to implement different models later on.

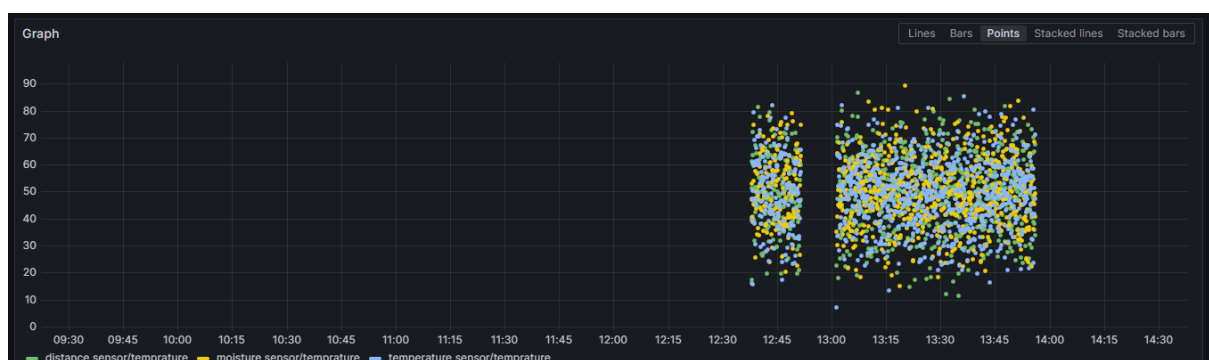
Lines



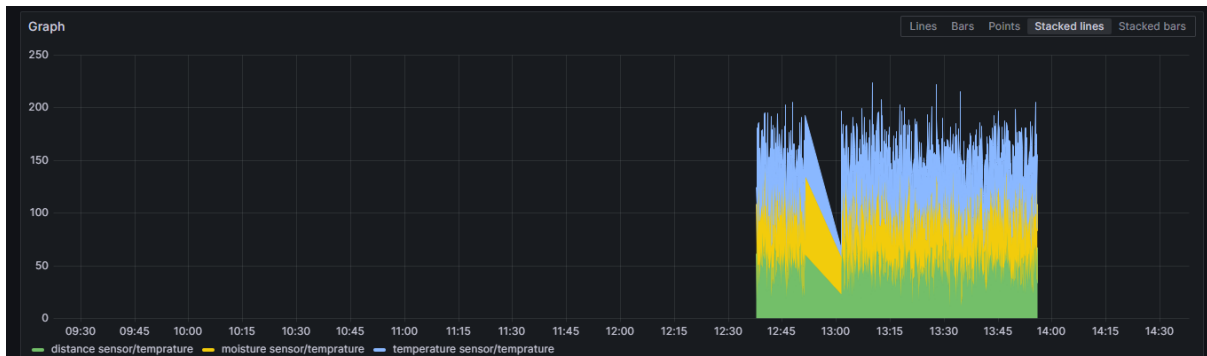
Bars



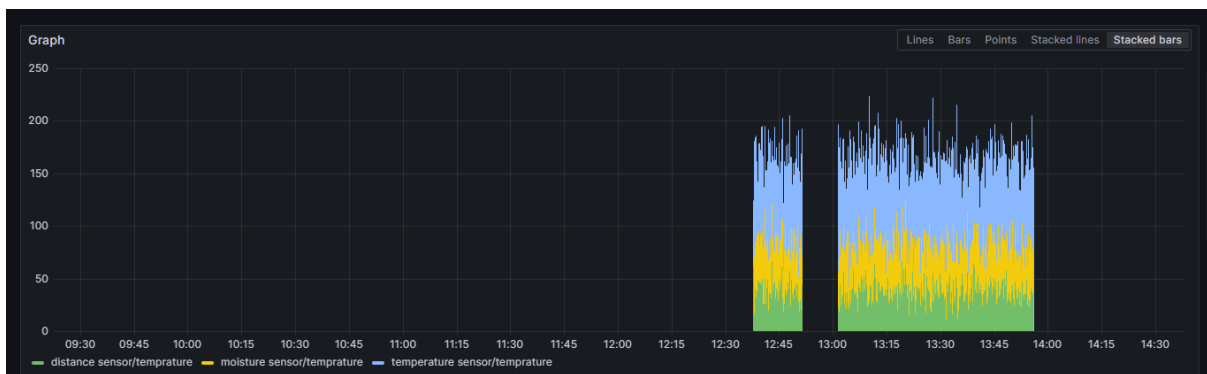
Points



Stacked Lines



Stacked Bars



4. Conclusion

In conclusion, this IoT project has successfully demonstrated the creation of a comprehensive end-to-end monitoring system, showcasing the capabilities of ESP32 microcontrollers for data collection, Docker for containerization, InfluxDB for data storage, and Grafana for data visualization. The project's primary objective was to establish a robust infrastructure capable of collecting environmental data, processing it, and presenting it in real-time, and it has been achieved through a well-defined methodology.

The hardware setup ensured the accurate connection of DHT and Ultrasonic sensors to the ESP32, enabling the collection of critical temperature, humidity, and distance data. Software configuration, including the Arduino sketch and MQTT broker setup, facilitated seamless data transmission to the MQTT broker.

Docker containerization streamlined deployment and management, enhancing scalability and portability. Telegraf and InfluxDB were effectively configured to store and process data securely, while Grafana served as the visual centerpiece, offering a customizable dashboard with various data visualization options.

Security, an integral component of any IoT project, was emphasized, with the use of authentication and access controls in all components of the system.

Looking ahead, this project can be extended by integrating additional sensors, implementing more sophisticated data analytics, and enhancing security measures further. Additionally, considerations for scalability and redundancy should be explored to support growing data volumes and ensure system resilience. Nevertheless, the successful realization of this IoT project serves as a solid foundation for future endeavors in the field of data monitoring and visualization.