

Name: Shukraditya Bose

Registration Number: 22MID0048

Interfacing an ESP8266 with a DHT11 sensor

CSI3001

Cloud Computing Methodologies

Table of Contents

1. [Introduction](#)
2. [Hardware Components](#)
3. [Software Requirements](#)
4. [System Design](#)
5. [Interfacing the DHT11 with ESP8266](#)
6. [Results and Analysis](#)
7. [Conclusion](#)

1. Introduction

The demand for environmental monitoring systems has increased recently, particularly in agricultural and industrial applications. This project focuses on interfacing a DHT11 temperature and humidity sensor with an ESP8266 microcontroller to collect real-time data and upload it to InfluxDB, a time-series database. The main objectives of the project are to:

- Measure temperature and humidity levels.
- Stream the collected data to InfluxDB for storage and analysis.
- Provide a foundation for further development of IoT-based monitoring systems.

2. Hardware Components

The hardware components used in this project are as follows:

- **ESP8266 NodeMCU:** A low-cost Wi-Fi microcontroller that allows for easy connection to the internet.
- **DHT11 Sensor:** A basic, low-cost digital temperature and humidity sensor with a typical accuracy of $\pm 2^{\circ}\text{C}$ and $\pm 5\%$ RH.
- **Breadboard:** For assembling the circuit without soldering.
- **Jumper Wires:** These are used to make connections between the components.
- **Power Supply:** Typically powered through USB.

Specifications:

- **ESP8266:**
 - Voltage: 3.3V
 - Current: $\sim 300\text{mA}$
- **DHT11:**
 - Voltage: 3.3V to 5.5V
 - Operating range: 0 to 50°C , 20% to 90% RH

Circuit Diagram:

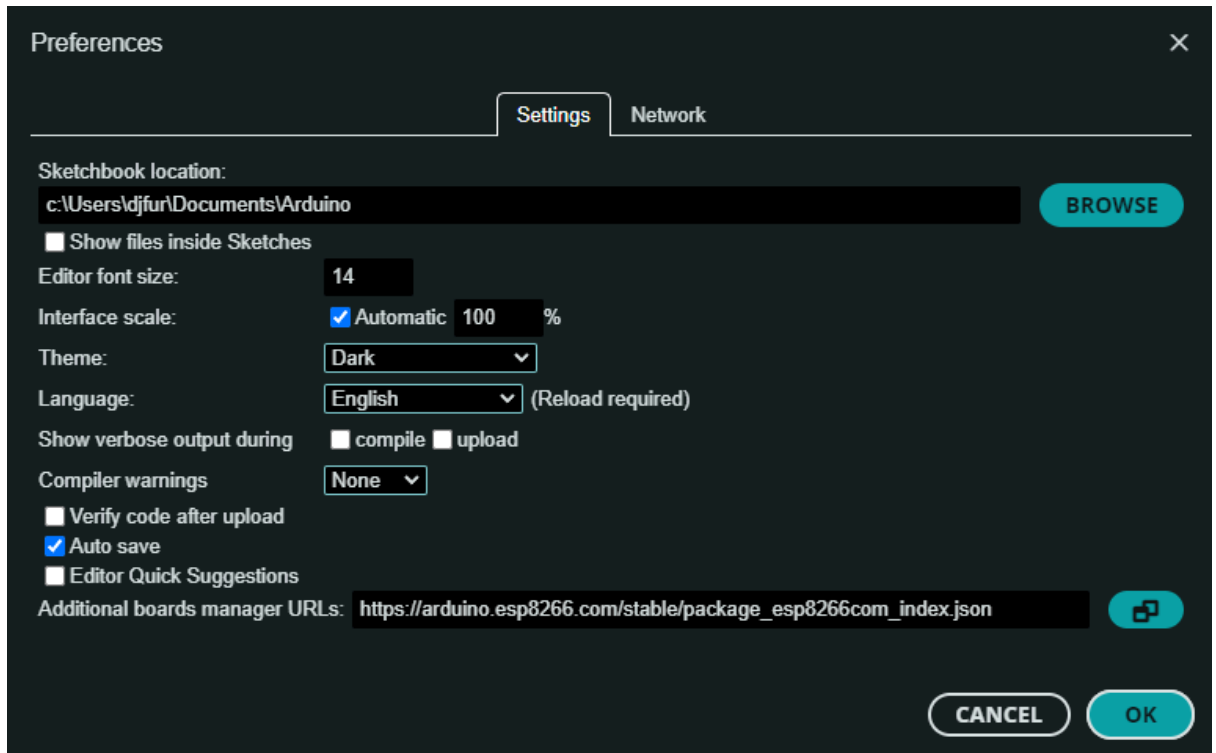
3. Software Requirements

- **Arduino IDE:** The primary development environment used for programming the ESP8266.
- **Libraries:**
 - `DHT.h`: Library for reading data from the DHT11 sensor.
 - `ESP8266WiFi.h`: Library for handling Wi-Fi connections.
 - `InfluxDbClient.h` and `InfluxDbCloud.h`: For integrating with InfluxDB.

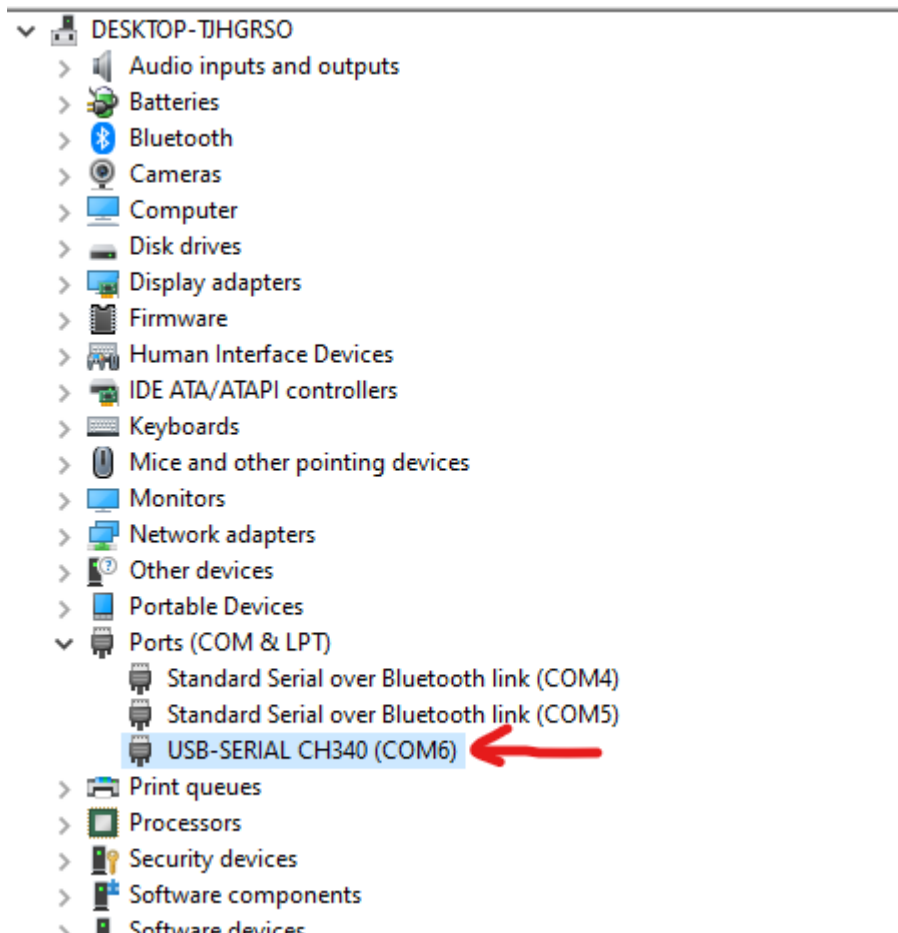
Configuration:

- Install the necessary libraries through the Arduino Library Manager.

- https://arduino.esp8266.com/stable/package_esp8266com_index.json Paste this URL in the Additional Board Manager URL section.

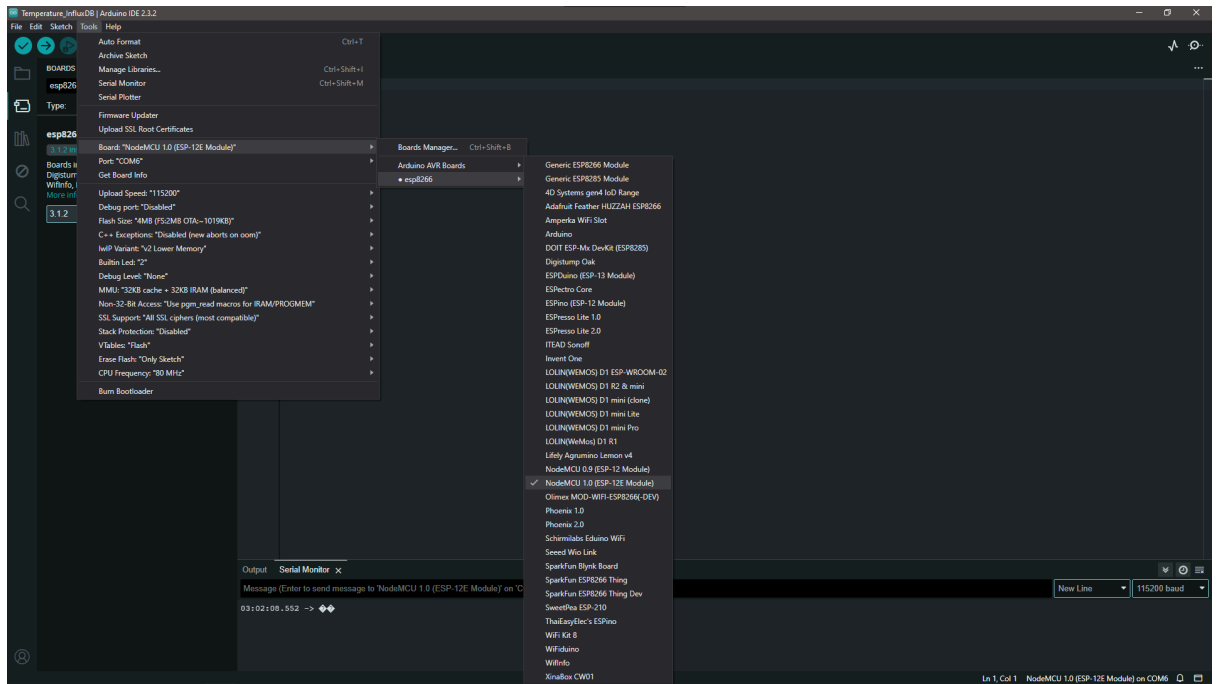


- Now, in the Manager Section on the side tab, install the **esp8266** by **ESP8266 Community**.
- Find the COM Port number from Device Manager on Windows after connecting ESP8266 via USB cable.



Here, it's COM6.

- Now, in Arduino IDE, in the Tools section, under the Boards section, go to esp8266 and select NodeMCU 1.0 ESP-12E Module. Select the Port as found in the previous step.



Now time to set up InfluxDB.

- After signing in, we need to create a workspace and name it.

Welcome to InfluxData

Just a few things before we get started. No credit card required.

Workspace Details

Account

Your company name

shukraditya

Organization

Your first organization or project in your account

dev team 610

Storage Provider

This is where we will store your time series data.

Amazon Web Services



US East (N. Virginia)

Don't see the region you need? [Let us know.](#)




I have viewed and agree to the [InfluxDB Cloud 2.0 Services Subscription Agreement](#) and [InfluxData Global Data Processing Agreement](#).

- Choose the free plan and continue to choose the use case.


How do you plan to use InfluxDB?

We'll use this to customize your experience



What are you looking to do on InfluxDB?*


- ☐ Build an Application
- ☒ Monitor & Analyze Data
- ☐ Learn about InfluxDB
- ☐ Other



What kind of data do you plan to bring into InfluxDB?*

Check all that are applicable to you

- ☒ Data from IOT sensors, devices etc.
- ☐ Metrics data from server, network, infrastructure, applications etc.
- ☐ Tracing, events, or log data from applications
- ☐ I don't know
- ☐ Other

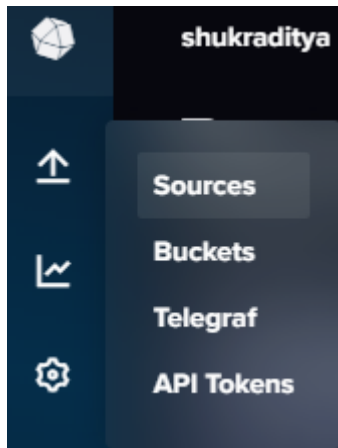


Are you migrating an existing workload?*

- ☒ I'm starting a new workload.
- ☐ I'm migrating from InfluxDB OSS.
- ☐ I'm migrating from a previous InfluxDB Cloud.
- ☐ I'm migrating from elsewhere (specify).

CONTINUE

- Now you have signed in. from the sidebar, click on Sources.



Select Arduino from the list.

- Create a bucket in the Initialize Client Section.

shukraditya > dev team 610

Get \$250 free credit

UPGRADE NOW

SB

↑

📈

⚙️

?

📄

Setting Up Arduino

5 minutes

✓ Overview

✓ Prepare Arduino IDE

✓ Install Dependencies

⚙️ Initialize Client

✎ Write Data

★ Finish

Setting Up

Initialize Client

Select or Create a bucket

A **bucket** is used to store time-series data. Here is a list of your existing buckets. You can select one to use for the rest of the tutorial, or create one below.

BUCKET

+ CREATE BUCKET

You don't have any Buckets

Configure an InfluxDB profile

Next we'll need to configure the client and its initial connection to InfluxDB. InfluxDB Cloud uses Tokens to authenticate API access. We've created an all-access token for you for this set up process.

Paste the following snippet into a blank Arduino sketch file.

```
#include <InfluxDbCloud.h>

// WiFi AP SSID
#define WIFI_SSID "YOUR_WIFI_SSID"
// WiFi password
#define WIFI_PASSWORD "YOUR_WIFI_PASSWORD"

#define INFLUXDB_URL "https://us-east-1-1.aws.cloud2.influxdata.com"
#define INFLUXDB_TOKEN "YBRlcUhykmRr1kdnDALnyc1PoKx5sqVcjK"
#define INFLUXDB_ORG "fcb8a28b455638ac"
#define INFLUXDB_BUCKET "YOUR_BUCKET"
```

10

Create Bucket

Name*

temperature

Data Retention Preferences

NEVER DELETE

DELETE OLDER THAN

14 days

CANCEL

CREATE

- A code snippet is generated based on the requirements chosen and use it to upload test data.

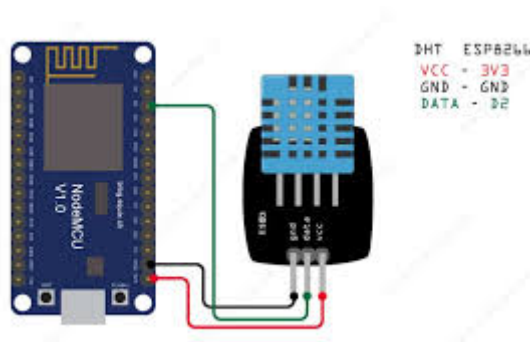
4. System Design

The system architecture involves the following components:

1. DHT11 sensor connected to the ESP8266.
2. ESP8266 reads data from the sensor and connects to a Wi-Fi network.
3. Data is formatted and sent to the InfluxDB server using HTTP POST requests.

5. Interfacing the DHT11 with ESP8266

Wiring Diagram:



- Connect the DHT11 sensor to the ESP8266 as follows:
 - VCC to 3.3V
 - GND to GND
 - Data Pin to D4 (GPIO2)

Code Explanation: The provided code handles the setup and loop processes efficiently. It establishes a Wi-Fi connection, initializes the DHT11 sensor, reads temperature and humidity data, and uploads it to InfluxDB.

```
#if defined(ESP32)
  #include <WiFiMulti.h>
  WiFiMulti wifiMulti;
  #define DEVICE "ESP32"
#elif defined(ESP8266)
  #include <ESP8266WiFiMulti.h>
  ESP8266WiFiMulti wifiMulti;
  #define DEVICE "ESP8266"
#endif

#include "DHT.h"
#include <InfluxDbClient.h>
#include <InfluxDbCloud.h>

// WiFi AP SSID
#define WIFI_SSID "WIFI_SSID"
#define WIFI_PASSWORD "WIFI_PASSWORD"

// Place InfluxDB specific code here

// Time zone info
#define TZ_INFO "IST-5:30"

//defining DHT variables
#define DHTPIN 4
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
```

```

// Declare InfluxDB client instance with preconfigured InfluxCloud
certificate
InfluxDBClient client(INFLUXDB_URL, INFLUXDB_ORG, INFLUXDB_BUCKET,
INFLUXDB_TOKEN, InfluxDbCloud2CACert);

// Declare Data point
Point sensor("temperature data");

void setup() {
  Serial.begin(115200);

  // Setup wifi
  WiFi.mode(WIFI_STA);
  wifiMulti.addAP(WIFI_SSID, WIFI_PASSWORD);

  Serial.print("Connecting to wifi");
  while (wifiMulti.run() != WL_CONNECTED) {
    Serial.print(".");
    delay(100);
  }
  Serial.println();
  dht.begin();

  sensor.addTag("device", "ESP8266");
  sensor.addTag("location", "room1");

  // Accurate time is necessary for certificate validation and
writing in batches
  // We use the NTP servers in your area as provided by:
https://www.pool.ntp.org/zone/
  // Syncing progress and the time will be printed to Serial.
  timeSync(TZ_INFO, "pool.ntp.org", "time.nis.gov");

  // Check server connection

```

```

    if (client.validateConnection()) {
        Serial.print("Connected to InfluxDB: ");
        Serial.println(client.getServerUrl());
    } else {
        Serial.print("InfluxDB connection failed: ");
        Serial.println(client.getLastErrorMessage());
    }
}

void loop() {
    // Clear fields for reusing the point. Tags will remain the same
    // as set above.
    sensor.clearFields();

    // Store measured value into point
    // Report RSSI of currently connected network
    float humidity = dht.readHumidity();
    float temperatureC = dht.readTemperature();

    if (isnan(humidity) || isnan(temperatureC)) {
        Serial.println(F("Failed to read from DHT sensor!"));
        return;
    }

    // Store temperature and humidity values into point
    sensor.addField("humidity", humidity);
    sensor.addField("temperature", temperatureC);

    // Print what we're writing to InfluxDB
    Serial.print("Writing: ");
    Serial.println(sensor.toLineProtocol());

    // Check WiFi connection and reconnect if needed
    if (wifiMulti.run() != WL_CONNECTED) {
        Serial.println("Wifi connection lost");
    }
}

```

```

// Write point
if (!client.writePoint(sensor)) {
    Serial.print("InfluxDB write failed: ");
    Serial.println(client.getLastErrorMessage());
}

Serial.println("Waiting 60 seconds");
delay(60000);
}

```

Now, start the process by uploading the code to the board.
Once uploaded, we can see this.

```

. Instruction RAM (IRAM_ATTR, ICACHE_RAM_ATTR), used 60735 / 65536 bytes (92%)
| SEGMENT BYTES DESCRIPTION
| ICACHE 32768 reserved space for flash instruction cache
| IRAM 27967 code in IRAM
. Code in flash (default, ICACHE_FLASH_ATTR), used 388728 / 1048576 bytes (37%)
| SEGMENT BYTES DESCRIPTION
| IROM 388728 code in flash
esptool.py v3.0
Serial port COM6
Connecting...
Chip is ESP8266EX
Features: WiFi
Crystal is 26MHz
MAC: c4:d8:d5:03:59:c4
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Compressed 424608 bytes to 313429...
Writing at 0x00000000... (5 %)
Writing at 0x00004000... (10 %)
Writing at 0x00008000... (15 %)
Writing at 0x0000c000... (20 %)
Writing at 0x00010000... (25 %)
Writing at 0x00014000... (30 %)
Writing at 0x00018000... (35 %)
Writing at 0x0001c000... (40 %)
Writing at 0x00020000... (45 %)
Writing at 0x00024000... (50 %)
Writing at 0x00028000... (55 %)
Writing at 0x0002c000... (60 %)
Writing at 0x00030000... (65 %)
Writing at 0x00034000... (70 %)
Writing at 0x00038000... (75 %)
Writing at 0x0003c000... (80 %)
Writing at 0x00040000... (85 %)
Writing at 0x00044000... (90 %)
Writing at 0x00048000... (95 %)
Writing at 0x0004c000... (100 %)
Wrote 424608 bytes (313429 compressed) at 0x00000000 in 27.6 seconds (effective 123.3 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting via RTS pin...

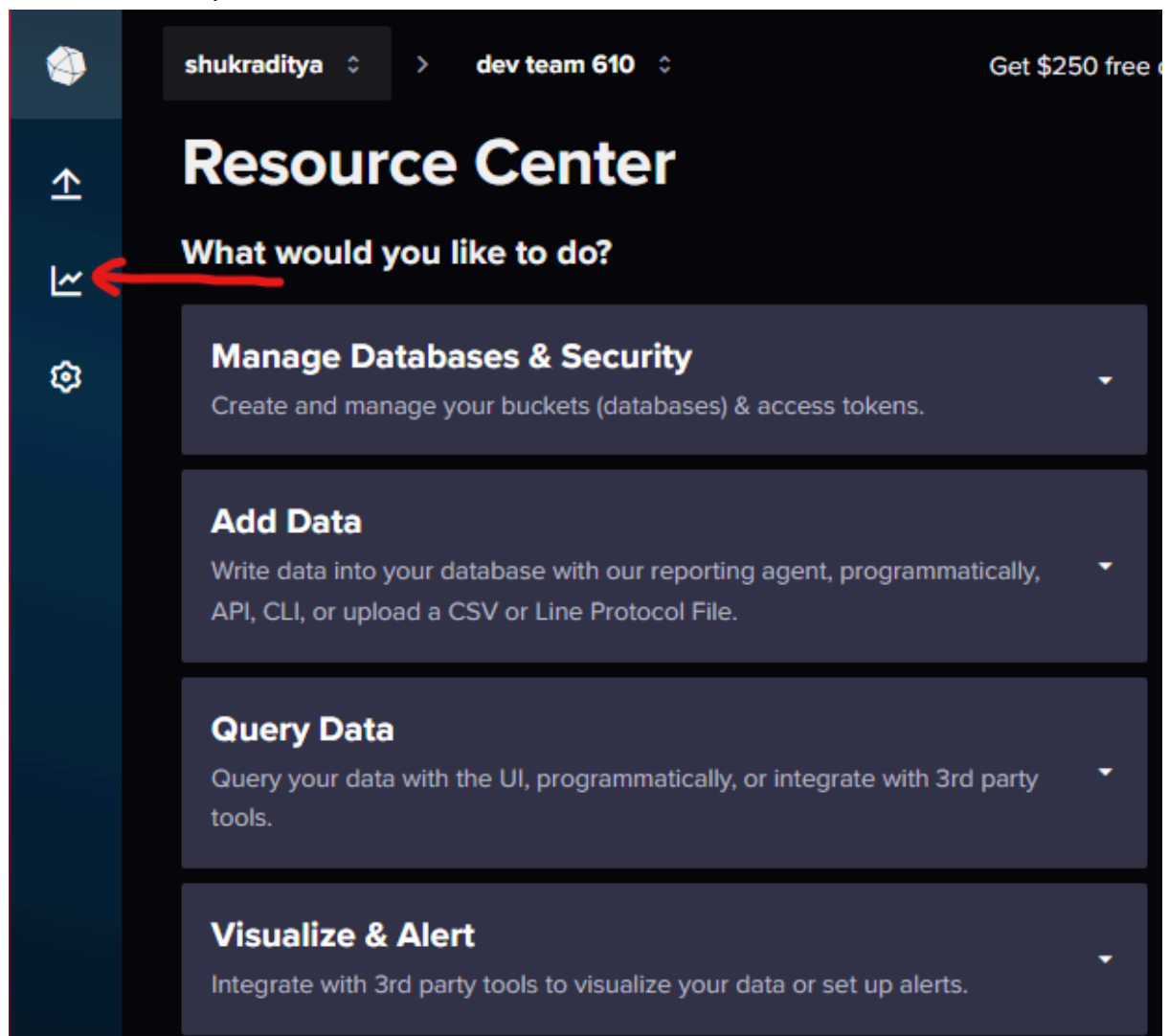
```

Then, we open the serial monitor, ensure the baud rate is same (115200 here) and we can see the output.

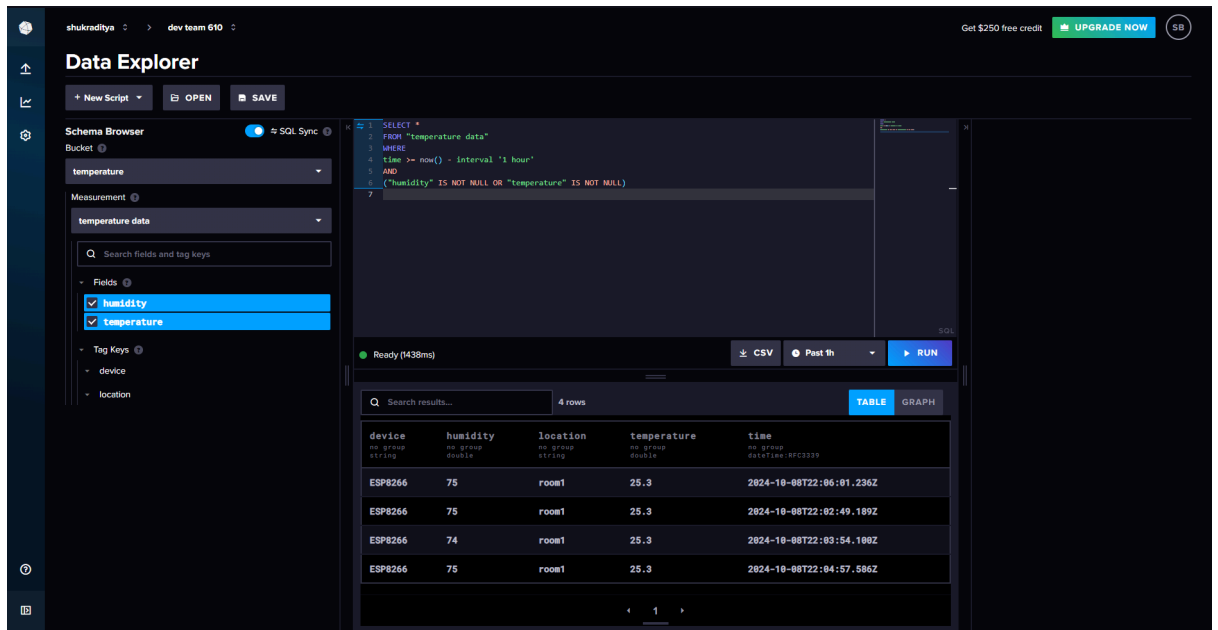
```
03:32:37.172 -> .  
03:32:40.835 -> Syncing time.  
03:32:41.300 -> Synchronized time: Wed Oct 9 03:32:41 2024  
03:32:41.346 ->  
03:32:45.730 -> Connected to InfluxDB: https://us-east-1-1.aws.cloud2.influxdata.com  
03:32:45.777 -> Writing: temperature\ data,device=ESP8266,location=room1 humidity=75.00,temperature=25.30  
03:32:49.453 -> Waiting 60 seconds  
03:33:49.434 -> Writing: temperature\ data,device=ESP8266,location=room1 humidity=74.00,temperature=25.30  
03:33:54.205 -> Waiting 60 seconds
```

Now, we verify the same output in InfluxDB.

- Click on Data Explorer on the side bar.

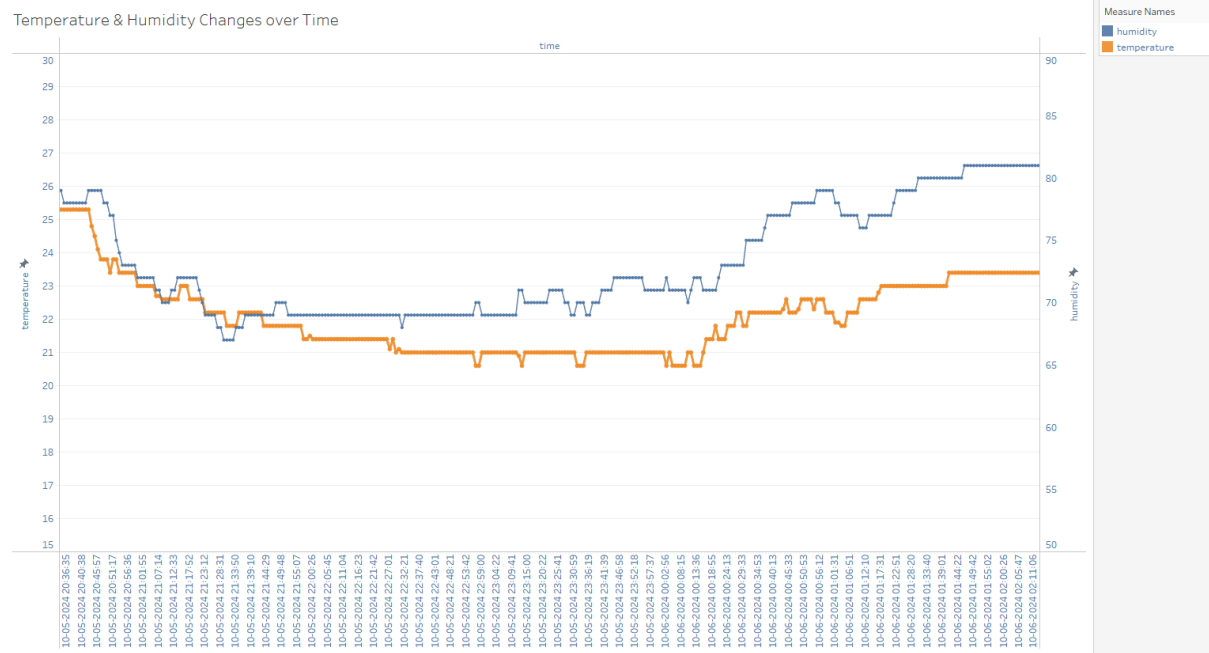


- Now, select the options you want to view, which generates a SQL Query, Run it to see the results.



Output has hence been verified.

To visualise the fluctuation in temperature over 6 hours, data was collected every 60 seconds and plotted.

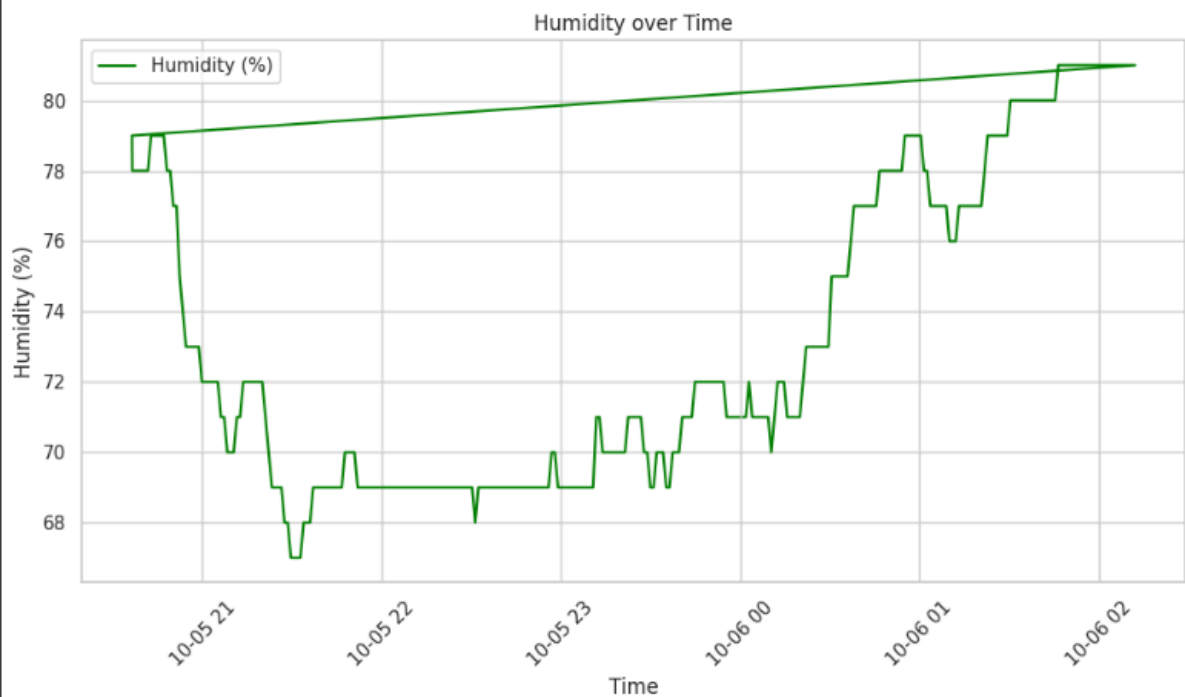
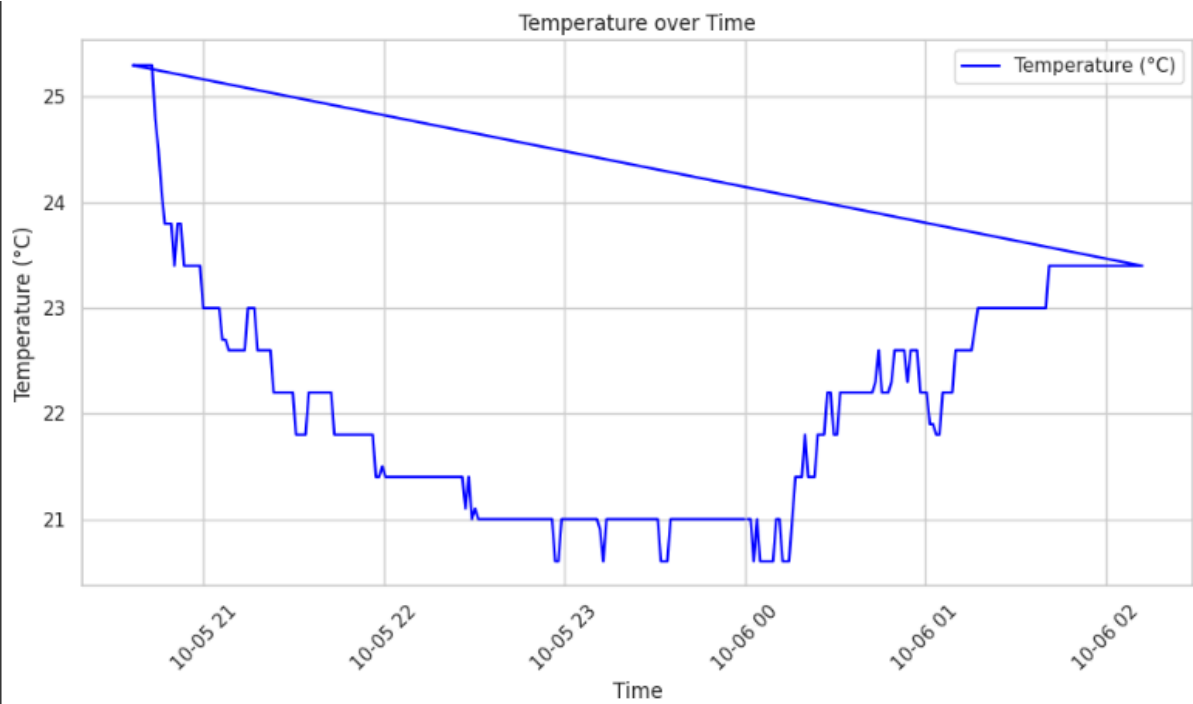


6. Results and Analysis

We perform a statistical analysis and achieve the following values based on the values stored in the database.

Statistical Summary:

| | humidity | temperature |
|-------|------------|-------------|
| count | 319.000000 | 319.000000 |
| mean | 73.169279 | 22.074922 |
| std | 4.390119 | 1.103852 |
| min | 67.000000 | 20.600000 |
| 25% | 69.000000 | 21.000000 |
| 50% | 71.000000 | 21.800000 |
| 75% | 77.500000 | 23.000000 |
| max | 81.000000 | 25.300000 |



Correlation between Humidity and Temperature:

| | humidity | temperature |
|-------------|----------|-------------|
| humidity | 1.000000 | 0.737524 |
| temperature | 0.737524 | 1.000000 |

This shows a moderate-to-high degree of correlation between humidity and temperature.

7. Conclusion

The analysis of the temperature and humidity data collected from the ESP8266 sensor system in “room1” provided valuable insights into the environmental conditions. Key findings include:

1. Statistical Summary:

- The average temperature and humidity in the monitored environment were approximately 22.074922°C and 73.169279%, respectively.
- Temperature and humidity exhibited reasonable variations, with the highest temperature recorded at 25.3°C and the highest humidity reaching 81%.

2. Time Series Trends:

- Temperature and humidity displayed observable fluctuations over time, likely influenced by external environmental factors or activities within the room.
- Temperature remained relatively stable during the initial period but showed spikes around certain intervals, indicating possible heat sources or changes in room conditions.
- Humidity followed a similar pattern, slightly increasing over time, possibly correlated with temperature changes or external factors like ventilation or moisture sources.

3. Correlation Analysis:

- A moderately high correlation was identified between temperature and humidity, suggesting that changes in one metric tend to have a measurable impact on the other.
- This correlation can be further explored to understand better the dynamics between temperature and moisture in controlled environments.

4. Visualisation Insights:

- The dual-axis plot effectively showcased the relationship between temperature and humidity over time. While there were some simultaneous shifts, there were periods where the metrics diverged, highlighting potential shifts in room activity or external influences.

This data serves as a preliminary study of the room1's environmental conditions. Further analysis with additional metrics, such as airflow, light exposure, or human activity, could provide a more comprehensive understanding of the interactions between temperature, humidity, and other factors. Future work could also involve comparing the environmental conditions across multiple rooms or devices to identify broader trends or anomalies.

This analysis helps understand the importance of monitoring real-time environmental data, particularly in applications like climate control, IoT-based home automation, or environmental research, where temperature and humidity play crucial roles.

For a quick video demo: <https://youtu.be/3-G1RQXqzJY>