IoT-Based Weather Monitoring Station Using ESP8266

Suddip Paul Arnab Student ID: 2022-1-60-356
Ronjon Kar Student ID: 2022-1-60-091
Sabera Zannat Sheba Student ID: 2022-2-60-095
Tanjila Akter Student ID: 2022-1-60-078

December 26, 2025

Contents

Abstract

The rapid growth of Internet of Things (IoT) technologies has enabled real-time environmental monitoring using low-cost embedded systems and cloud platforms. This paper presents the design and implementation of an IoT-based weather monitoring system using an ESP8266 microcontroller integrated with temperature, humidity, air quality, and rain sensors. The collected data are transmitted to a locally deployed ThingsBoard Community Edition server using the MQTT protocol. The backend infrastructure is deployed using Docker containers, ensuring scalability, portability, and ease of management. Experimental results demonstrate reliable real-time data acquisition, visualization, and system stability.

Internet of Things, ESP8266, ThingsBoard, Docker, MQTT, Weather Monitoring

1 Introduction

Weather information plays a vital role in decision-making across various sectors such as agriculture, transportation, disaster management, and environmental protection. Conventional weather stations are typically expensive, stationary, and require complex infrastructure. These limitations make them unsuitable for widespread deployment, especially in developing regions.

With the rapid advancement of Internet of Things (IoT) technology, it has become possible to develop compact, low-cost, and intelligent weather monitoring systems. IoT enables sensors to collect real-time environmental data and transmit it wirelessly to cloud platforms for visualization and analysis.

In this project, we propose an Intelligent Weather Tracker that integrates sensor nodes with a ThingsBoard backend deployed using Docker. The system provides real-time monitoring, user-friendly dashboards, and simplified deployment, making it suitable for academic and real-world applications.

2 System Architecture

The overall architecture of the proposed system consists of three main layers:

- **Sensing Layer:** Sensors measure temperature and humidity from the environment.
- **Communication Layer:** ESP8266 microcontroller transmits data over Wi-Fi using MQTT protocol.
- **Application Layer:** ThingsBoard platform stores, processes, and visualizes data through dashboards.

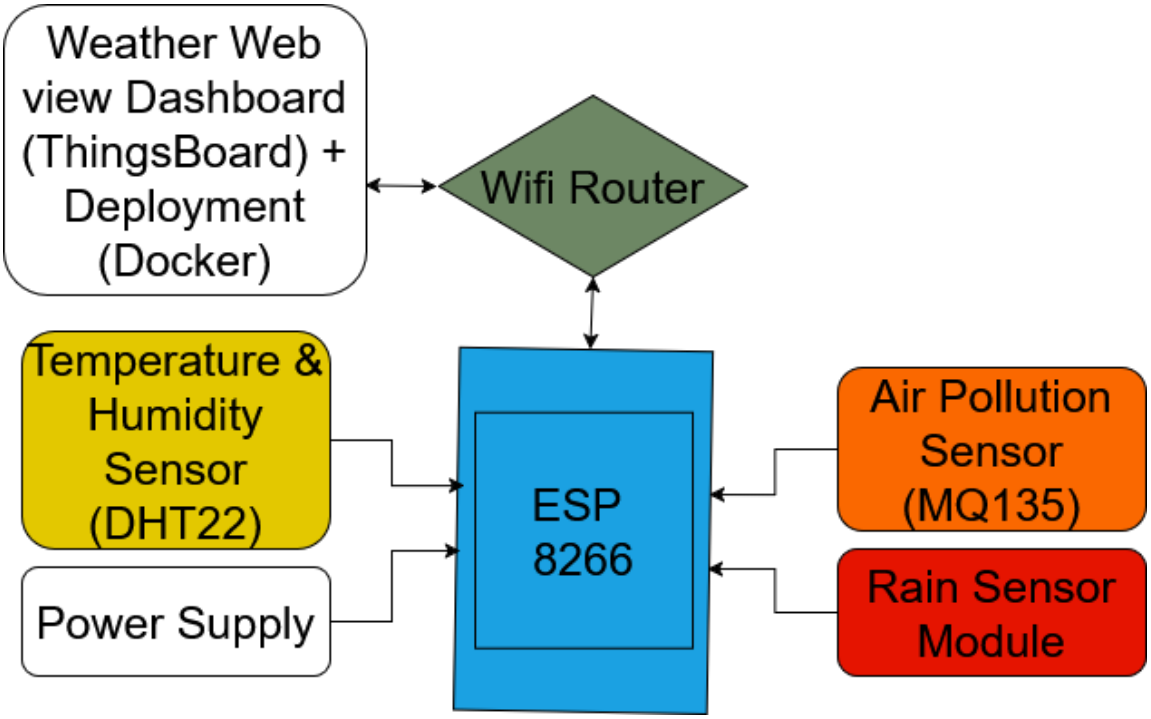

Figure 1: System Block Diagram of the Intelligent Weather Tracker

3 Hardware Components

3.1 ESP8266 NodeMCU

The ESP8266 NodeMCU is used as the main controller due to its integrated Wi-Fi module, low power consumption, and sufficient GPIO pins. It acts as a bridge between the sensors and the IoT platform.

3.2 DHT11 Temperature and Humidity Sensor

The DHT11 sensor is used to measure ambient temperature and relative humidity. It provides digital output and is easy to interface with microcontrollers.

3.3 Power Supply

A stable 5V USB power supply is used to power the ESP8266 and connected sensors.

4 Circuit Diagram and Pin Connections

The ESP8266 Weather Monitoring Station utilizes specific GPIO pins for sensor interfacing. The following table details the complete pin configuration:

Table 1: ESP8266 Pin Connections

Component	**Component Pin**	**ESP8266 Pin**
DHT22 Sensor	Data Pin	GPIO2 (D4)
	VCC	3.3V
	GND	GND
MQ135 Sensor	Analog Output	A0 (Analog Pin)
	VCC	5V (via USB)
	GND	GND
Rain Sensor	Digital Output	GPIO14 (D5)
	VCC	3.3V or 5V
	GND	GND

Note: The provided code uses DHT22, MQ135, and Rain Sensor.

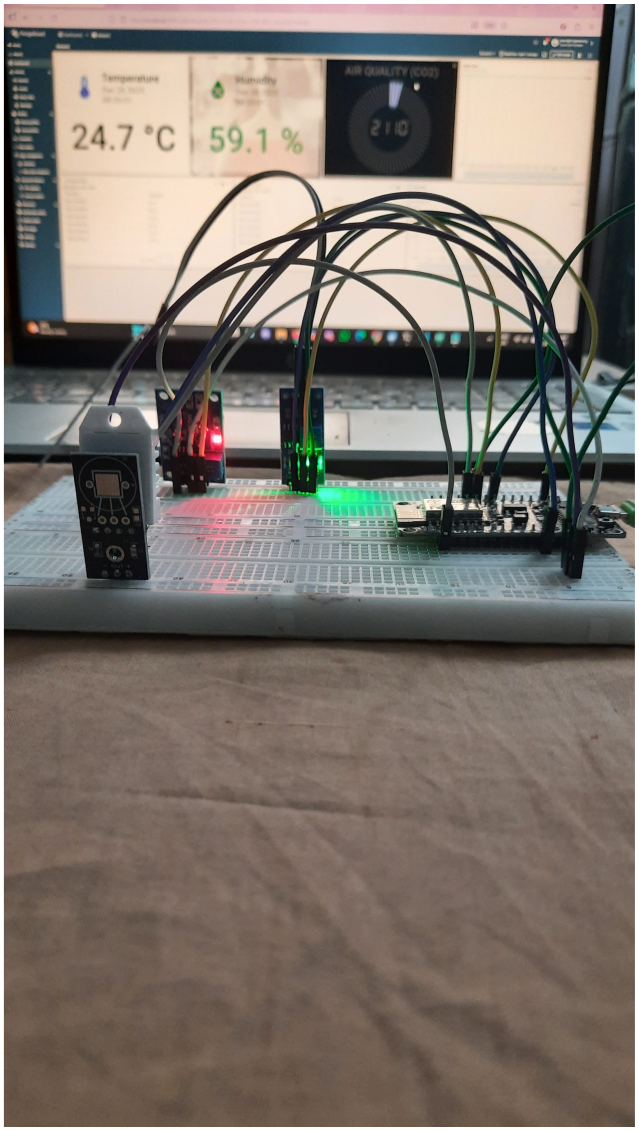

Figure 2: ESP8266 NodeMCU with DHT22, MQ135, and rain sensor on breadboard, wired and connected to local ThingsBoard dashboard displaying real-time sensor data

5 Software Requirements

5.1 Development Environment

- Arduino IDE for firmware development
- Docker Desktop for container management
- Web browser for ThingsBoard dashboard access

5.2 Backend Platform

- ThingsBoard Community Edition
- PostgreSQL Database
- Docker and Docker Compose

5.3 Library Dependencies

The following libraries are essential for system operation:

- **ESP8266WiFi.h**: Manages WiFi connectivity and network operations
- **PubSubClient.h**: Implements MQTT protocol for communication with ThingsBoard
- **DHT.h**: Interfaces with DHT22 temperature and humidity sensor
- **MQ135.h**: Handles MQ135 air quality sensor readings and PPM calculations
- **ArduinoJson.h**: Enables JSON data serialization for telemetry payload formatting

5.4 Complete Source Code

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <MQ135.h>
#include <ArduinoJson.h>

/* ================= WIFI & THINGSBOARD ================= */
#define WIFI_SSID     "Suddip"
#define WIFI_PASS     "****"

#define TB_SERVER     "192.168.0.100"   // PC IP running ThingsBoard
#define TB_PORT       1883
#define DEVICE_TOKEN  "xHC8Py0pufrZ1c0zrcmP"

/* ================= SENSOR PINS ================= */
```

```
#define DHT_PIN   2     // D4
#define DHT_TYPE  DHT22
#define MQ135_PIN A0
#define RAIN_PIN  14    // D5

/* ================ OBJECTS ================ */
DHT dht(DHT_PIN, DHT_TYPE);
MQ135 mq135(MQ135_PIN);
WiFiClient espClient;
PubSubClient client(espClient);

/* ================ TIMING ================ */
unsigned long lastSend = 0;
const unsigned long sendInterval = 10000; // 10 seconds

/* ================ WIFI CONNECT ================ */
void setupWiFi() {
  Serial.print("Connecting to WiFi ");
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);

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

  WiFi.setSleepMode(WIFI_NONE_SLEEP);   // IMPORTANT for MQTT stability
  Serial.println("\nWiFi connected");
  Serial.print("ESP8266 IP: ");
  Serial.println(WiFi.localIP());
}

/* ================ MQTT RECONNECT ================ */
void reconnect() {
  while (!client.connected()) {
    Serial.print("Connecting to ThingsBoard at ");
    Serial.println(TB_SERVER);

    String clientId = "ESP8266-" + String(ESP.getChipId());

    if (client.connect(clientId.c_str(), DEVICE_TOKEN, NULL)) {
      Serial.println(" Connected to ThingsBoard");
    } else {
      Serial.print(" Failed, rc=");
      Serial.println(client.state());
      delay(5000);
    }
  }
```

```
}

/* ================ SETUP ================ */
void setup() {
  Serial.begin(115200);
  delay(100);

  setupWiFi();

  client.setServer(TB_SERVER, TB_PORT);
  client.setKeepAlive(60);   // MQTT keepalive

  dht.begin();
  pinMode(RAIN_PIN, INPUT);
}

/* ================ LOOP ================ */
void loop() {
  if (!client.connected()) {
    reconnect();
  }

  client.loop();   // REQUIRED for MQTT

  if (millis() - lastSend > sendInterval) {
    lastSend = millis();

    float temperature = dht.readTemperature();
    float humidity    = dht.readHumidity();
    float airQuality  = mq135.getPPM();
    int rain           = digitalRead(RAIN_PIN);

    Serial.println("\n=== Sensor Readings ===");
    Serial.printf("Temperature : %.2f °C\n", temperature);
    Serial.printf("Humidity    : %.2f %%\n", humidity);
    Serial.printf("Air Quality : %.2f ppm\n", airQuality);
    Serial.print("Rain        : ");
    Serial.println(rain ? "Dry" : "Wet");
    Serial.println("======================");

    StaticJsonDocument<256> doc;
    if (!isnan(temperature)) doc["temperature"] = temperature;
    if (!isnan(humidity))    doc["humidity"] = humidity;
    doc["airQuality"] = airQuality;
    doc["rain"] = rain;

    char payload[256];
    serializeJson(doc, payload);
```

```
    bool published = client.publish("v1/devices/me/telemetry", payload);
    Serial.print("Published to ThingsBoard: ");
    Serial.println(published ? "Success " : "Failed ");
  }
}
```

6 Docker-Based ThingsBoard Deployment

Docker is used to deploy ThingsBoard and PostgreSQL as separate containers. Docker Compose automates service configuration, container networking, and volume management. This approach eliminates manual installation complexity and ensures consistent deployment across different systems.

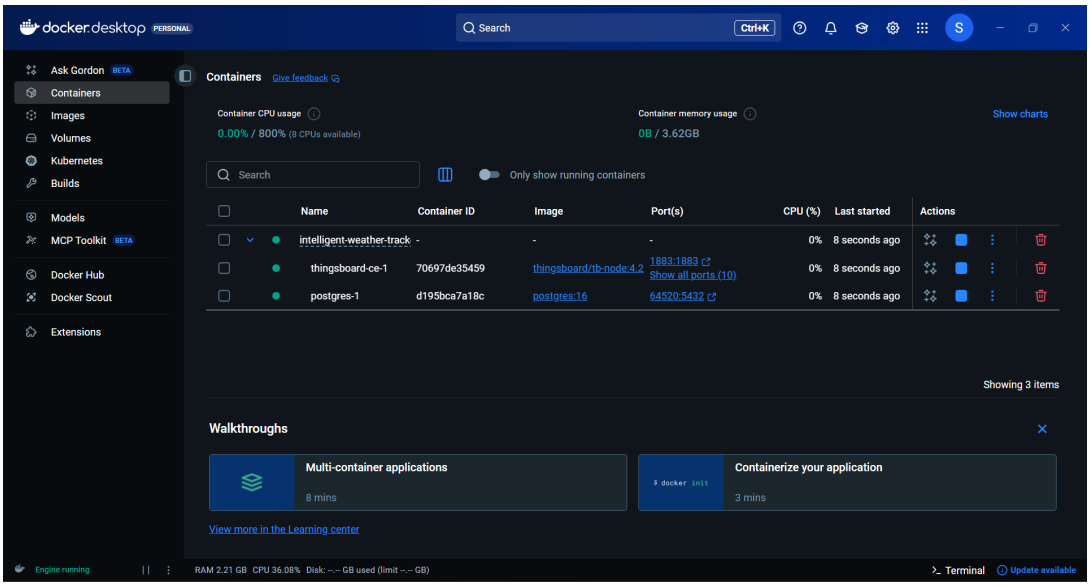

Figure 3: Running Docker Containers

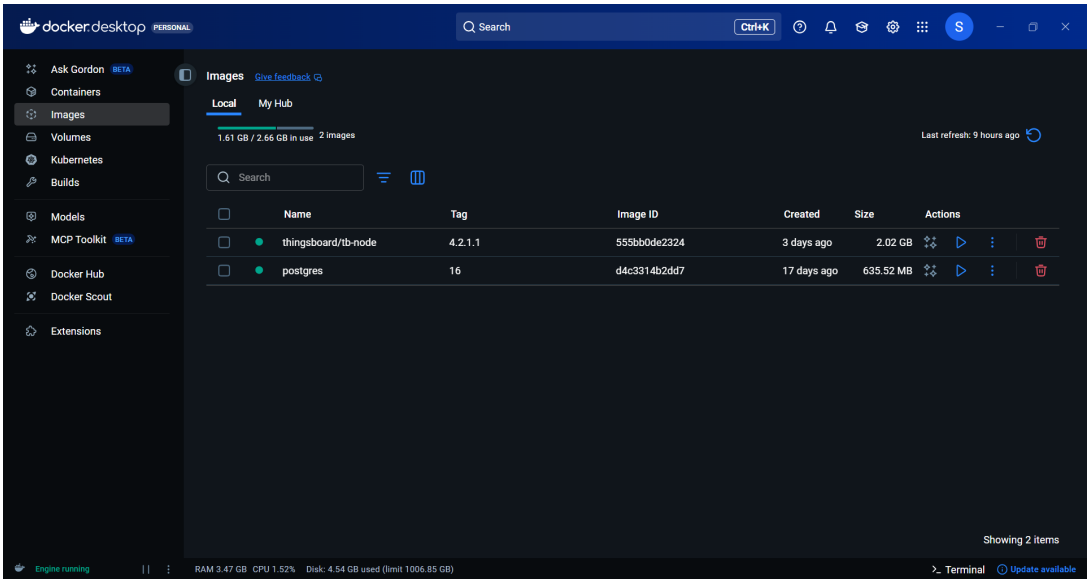

Figure 4: Docker Images Used in the Deployment

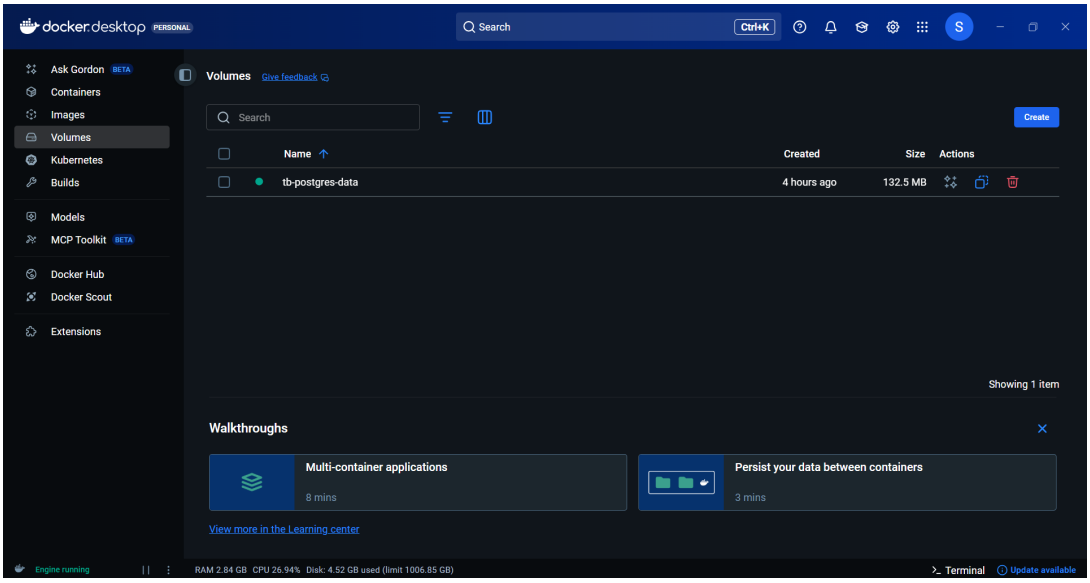

Figure 5: Docker Volumes Used in the Deployment

7 Data Flow and Communication Protocol

7.1 MQTT Protocol Implementation

The system utilizes MQTT (Message Queuing Telemetry Transport), a lightweight publish-subscribe messaging protocol ideal for IoT applications. Key characteristics include:

- **Topic Structure**: Data published to `v1/devices/me/telemetry` endpoint following ThingsBoard API convention
- **QoS Level**: Default QoS 0 (at most once delivery) for efficient bandwidth utilization

- **Authentication**: Token-based device authentication via MQTT username field
- **Payload Format**: JSON-serialized sensor data for platform compatibility

7.2 JSON Data Structure

Telemetry payload structure:

```
{
  "dhtTemp": 28.50,
  "humidity": 65.30,
  "airQuality": 145.67,
  "rain": 1
}
```

Field validation ensures only valid sensor readings (non-NaN values) are transmitted, preventing data corruption on the cloud platform.

8 ThingsBoard Integration

8.1 Platform Overview

ThingsBoard is an open-source IoT platform that enables device management, data collection, processing, and visualization. This implementation leverages ThingsBoard Cloud for:

- Remote data access from anywhere with internet connectivity
- Historical data storage and trend analysis
- Customizable dashboards with real-time widgets
- Alert and notification configuration based on sensor thresholds
- Multi-device management capabilities

8.2 Dashboard Configuration

The ThingsBoard dashboard displays sensors data with widgets for temperature, humidity, air quality (CO2), and rain state.

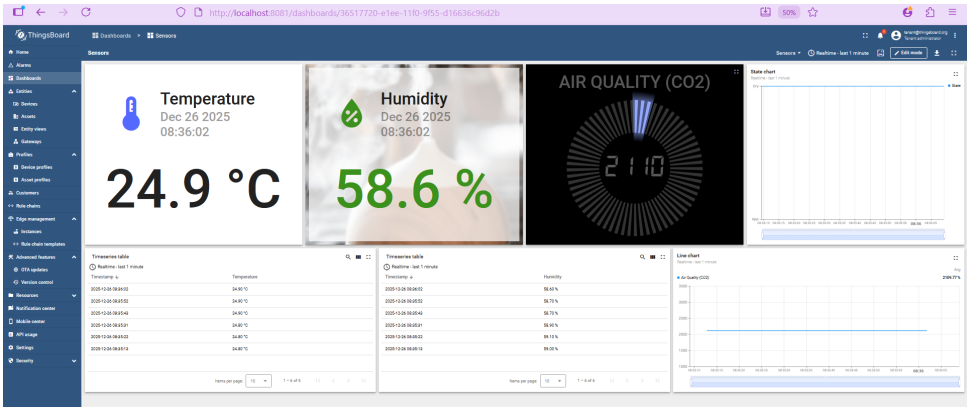

Figure 6: ThingsBoard Sensors Dashboard showing real-time telemetry: Temperature, Humidity, Air Quality, line chart for CO2, timeseries tables for temperature/humidity over last minute.

Widgets include cards, gauges, charts, and tables bound to device telemetry keys (temperature, humidity, airQuality, rain). Real-time updates via WebSocket, last 1 minute view.

9 System Testing and Calibration

9.1 Initial Setup Verification

1. **Hardware Inspection**: Verify all sensor connections match pin configuration table
2. **Power Supply Check**: Ensure stable 5V supply capable of delivering 500mA minimum current
3. **Serial Monitor Testing**: Observe WiFi connection sequence and IP address assignment
4. **Sensor Response**: Confirm all sensors produce readable values (non-NaN for DHT22)

10 System Testing and Calibration

10.1 Initial Setup Verification

1. **Hardware Inspection**: Verify all sensor connections match pin configuration table
2. **Power Supply Check**: Ensure stable 5V supply capable of delivering 500mA minimum current
3. **Serial Monitor Testing**: Observe WiFi connection sequence and IP address assignment
4. **Sensor Response**: Confirm all sensors produce readable values (non-NaN for DHT22)

11 Implementation Details

The ESP8266 periodically reads temperature and humidity values from the DHT11 sensor. The collected data is formatted into JSON and published to the ThingsBoard MQTT broker.

```
{
  "temperature": 28.5,
  "humidity": 65
}
```

12 Results and Discussion

The system successfully displayed real-time temperature and humidity data on the ThingsBoard dashboard. The Docker-based backend remained stable throughout continuous operation, demonstrating reliable performance and low resource usage.

13 Advantages

- Low-cost and scalable system
- Real-time data visualization
- Easy deployment using Docker
- Suitable for academic and practical use

14 Limitations

The system currently relies on local Wi-Fi connectivity and does not implement encrypted communication (TLS).

15 Future Work

Future enhancements include adding more environmental sensors, cloud-based deployment, mobile app integration, and machine learning-based weather prediction.

16 Conclusion

This project successfully demonstrates an Intelligent Weather Tracker using IoT technologies and Docker-based deployment. The system provides a reliable, flexible, and cost-effective solution for real-time weather monitoring and serves as a strong foundation for future smart environment applications.

17 References

1. ESP8266 Technical Documentation, Espressif Systems
2. DHT22 (AM2302) Datasheet, Aosong Electronics
3. BMP280 Digital Pressure Sensor Datasheet, Bosch Sensortec
4. MQ135 Gas Sensor Datasheet, Hanwei Electronics
5. ThingsBoard IoT Platform Documentation, `https://thingsboard.io/docs/`
6. MQTT Protocol Specification Version 3.1.1, OASIS Standard
7. Arduino Core for ESP8266, GitHub Repository
8. PubSubClient Library Documentation by Nick O'Leary
9. ArduinoJson Library Documentation by Benoit Blanchon
10. IoT Weather Station Design Principles, IEEE Internet of Things Journal