Hochschule für Technik
und Wirtschaft Berlin

University of Applied Sciences

< IoT-Based Temperature, Humidity & Dew Point Monitoring and Alert System >

Project Document

Name of the Study Programme

Master IT Business and Digitalization

Winter Semester 2025/26

Course Instructor: Prof. Dr. Alexander Huh

Submitted by: Ravi Kumar Thekare

Matriculation Number: 600663

Submission Date: 19.01.2026

Abstract

Environmental monitoring is critical in laboratories, storage facilities, and research environments where temperature and humidity affect equipment performance and product quality. Traditional cloud-based monitoring solutions raise privacy concerns, require continuous internet connectivity, and involve subscription costs. This project develops a local IoT environmental monitoring system using affordable open-source components: an ESP32 microcontroller reads a DHT22 sensor every 30 seconds, publishes data via MQTT to a Mosquitto broker on a Raspberry Pi, and Node-RED processes the data to calculate dew point (Magnus-Tetens formula), absolute humidity, and saturation depression. The system sends immediate Telegram alerts when temperature (>25°C or <10°C) or humidity (>65% or <30%) thresholds are breached, and delivers a comprehensive Environmental Analysis report once daily. A glassmorphism dashboard displays real-time temperature and humidity gauges, historical trend charts, and environmental metrics. Users can query system status via Telegram commands (/status, /help, /start). By processing data locally without cloud dependency, the system ensures complete data privacy, reduces latency, and maintains reliability independent of external infrastructure. This project demonstrates that practical environmental monitoring systems can be built with local edge computing using affordable components and open-source software, making it suitable for research facilities, educational institutions, and industrial applications requiring real-time monitoring with comprehensive alerts and historical analysis.

Contents

1. Introduction

Background

Environmental monitoring is important in many settings such as laboratories, data centers, storage facilities, and research environments. Temperature and humidity affect equipment performance, product quality, and human comfort. Traditional monitoring solutions often rely on cloud-based services where data is sent to external servers for processing. While these systems work, they create privacy concerns, depend on internet connectivity, and often require paid subscriptions.

Project Overview

This project develops a comprehensive IoT environmental monitoring system using affordable, open-source components. The system consists of an ESP32 microcontroller connected to a DHT22 temperature and humidity sensor, with a Raspberry Pi serving as the local processing hub. The ESP32 reads sensor data every 30 seconds and publishes it to a local MQTT broker running on the Raspberry Pi. Node-RED processes this data in real-time, calculates dew point using the Magnus-Tetens formula, monitors configurable temperature and humidity thresholds, and sends Telegram notifications when thresholds are exceeded. An interactive Node-RED dashboard displays live environmental data with gauges, trend charts, and comfort status indicators.

Objectives

The main objectives of this project are:

- To demonstrate that effective environmental monitoring can be built locally without relying on cloud services
- To implement a complete IoT system showing data acquisition, processing, storage, and user notification
- To ensure data privacy and system independence through local processing
- To provide real-time alerts and visualization of environmental conditions
- To create a practical, replicable solution suitable for research facilities and educational

2. Fundamentals

2.1 Internet of Things (IoT) Systems

The Internet of Things represents a network of interconnected devices that collect, process, and share data to enable intelligent decision-making and system control. Traditional IoT architectures rely on cloud computing where all data flows to distant servers for processing and storage. However, edge computing represents an alternative approach where processing occurs locally at or near the data source, reducing bandwidth requirements, minimizing latency, and enhancing privacy through local data retention. Edge computing systems are particularly valuable in environments where data privacy is critical, internet connectivity is unreliable, or real-time response is essential.

2.2 Microcontroller-Based Sensor Systems

Microcontrollers are compact processors designed for embedded applications, capable of reading sensors, performing calculations, and controlling devices with minimal power consumption. The ESP32 is a modern microcontroller combining WiFi and Bluetooth connectivity with sufficient processing power for real-time sensor data acquisition. The DHT22 is a digital temperature and humidity sensor providing calibrated measurements with ±2°C temperature accuracy and ±5% relative humidity accuracy, making it suitable for environmental monitoring applications. Digital sensors like the DHT22 communicate through timing protocols, providing calibrated output that eliminates the need for complex analog signal processing.

2.3 MQTT Protocol Fundamentals

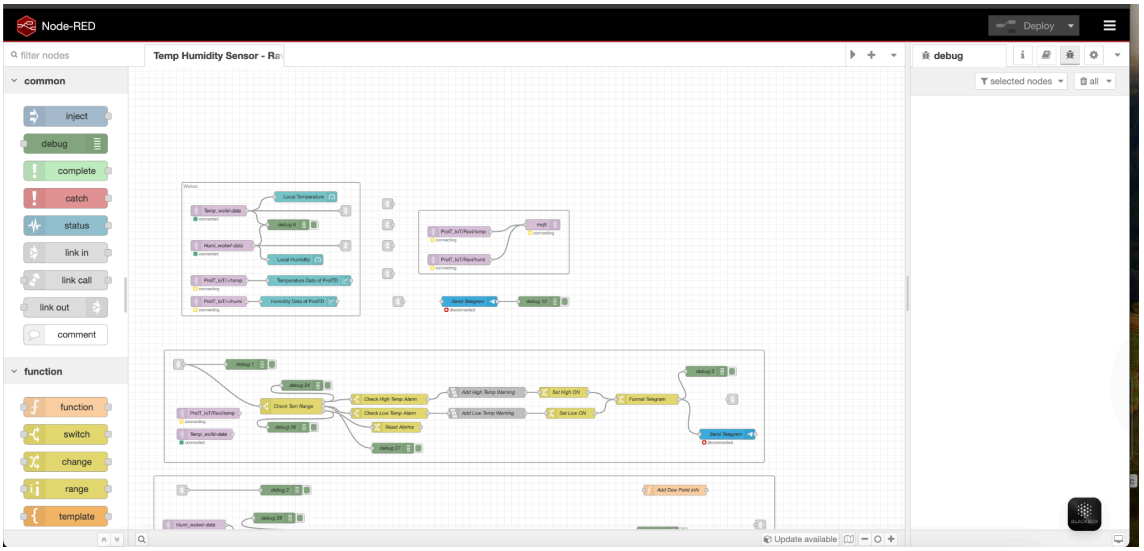
MQTT (Message Queuing Telemetry Transport) is a lightweight publish-subscribe protocol specifically designed for IoT applications operating under bandwidth and power constraints. In MQTT architecture, a central broker receives messages published by devices on specific topics and distributes these messages to all devices subscribed to those topics. This publish-subscribe model decouples message producers from consumers, enabling scalable system design where new devices can be added without modifying existing devices. MQTT supports multiple quality of service levels affecting delivery guarantees, making it suitable for both critical and non-critical IoT applications.

2.4 Node-RED and Visual Programming

Node-RED is a visual programming environment enabling creation of IoT applications through graphical workflow design without requiring traditional code writing. In Node-RED, discrete functional units called "nodes" are connected by wires defining data flow through the workflow. This visual approach makes complex data processing logic accessible to users without extensive programming experience while maintaining the flexibility and power of traditional programming. Node-RED includes pre-built nodes for common IoT operations including MQTT communication, mathematical calculations, message formatting, and external API integration.

2.5 Environmental Parameters Theory

Temperature is a fundamental environmental parameter representing the average kinetic energy of molecules, measured in degrees Celsius. Humidity, or relative humidity, represents the amount of water vapor in air relative to the maximum amount the air can hold at that temperature, expressed as a percentage. Dew point is the temperature at which air becomes saturated with water vapor, causing condensation to occur, and depends on both temperature and relative humidity. Absolute humidity represents the actual mass of water vapor in air expressed in grams per cubic meter, providing direct comparison of moisture content independent of temperature. Saturation depression, calculated as the difference between current temperature and dew point, indicates how far air is from saturation and the potential for condensation formation.



3. Analysis

3.1 Hardware Requirements Analysis

The system is built using a small set of reliable and cost-effective hardware components. The ESP32 microcontroller is used as the sensing node due to its integrated WiFi capability, low power consumption, and sufficient processing performance for sensor reading and MQTT communication.

Temperature and humidity measurements are provided by a DHT22 sensor, which offers adequate accuracy for indoor environmental monitoring. A Raspberry Pi 3 functions as the central processing unit, hosting both the Mosquitto MQTT broker and Node-RED for data processing and visualization.

All components operate using standard 5 V USB power supplies. The modest power requirements and low hardware cost make the system practical for continuous operation and easy deployment..

3.2 Software Requirements Analysis

The system relies entirely on lightweight, open-source software. MicroPython is used on the ESP32 to enable efficient sensor access and MQTT communication. The Mosquitto MQTT broker runs on the Raspberry Pi and handles message exchange with minimal resource usage.

Node-RED is used for data processing, calculation of environmental parameters, dashboard visualization, and alert logic. Telegram integration is implemented using the Telegram Bot API, allowing users to receive notifications through a widely used messaging platform.

3.3 Functional Requirements

The system must continuously measure temperature and humidity and transmit the data to the local MQTT broker at regular intervals. Node-RED must process incoming data in real time and calculate dew point, absolute humidity, and related environmental metrics.

Temperature and humidity values must be monitored against predefined upper and lower thresholds. When a threshold is exceeded, an alert must be sent immediately via Telegram. In addition, the system must generate a daily environmental summary report.

The dashboard must display current readings, historical trends, and qualitative comfort status. The Telegram bot must support basic commands such as system initialization, status requests, and help information.

3.4 Non-Functional Requirements

The system must provide timely alerts and near real-time dashboard updates. It must operate continuously without manual intervention and automatically recover from temporary network failures.

All data must remain within the local network to ensure privacy and independence from cloud services. The dashboard must be accessible through a standard web browser, and system configuration, including threshold values, must be simple and user-friendly.

4. Conception / System Design

4.1 System Architecture Overview

The system follows a classic IoT architecture where sensor data flows from the edge device through a message broker to a processing engine, then to user interfaces.

ESP32 (Sensor) → WiFi/MQTT → Mosquitto Broker → Node-RED Processing → Dashboard & Telegram

The ESP32 continuously reads temperature and humidity from the DHT22 sensor every 30 seconds and publishes the data to an MQTT broker running on the Raspberry Pi. Node-RED subscribes to these MQTT topics, processes the incoming data, evaluates alert thresholds, calculates advanced environmental metrics (dew point, absolute humidity), and routes the information to both a real-time web dashboard and Telegram bot notifications.

4.2 Hardware Specifications

ESP32 Microcontroller:

- Processor
- WiFi
- GPIO
- Power: USB

DHT22 Sensor:

- Measurement Range: Temperature -40 to +80°C, Humidity 0-100%
- Accuracy: ±2% humidity, ±0.5°C temperature
- Power: 3.3-5.5V

Raspberry Pi (Processing Unit):

- Processor
- Network: WiFi/Ethernet
- Power: 5V micro USB

Wiring Connection:

DHT22 Pin 1 (VCC) → ESP32 3V3 (3.3V)

DHT22 Pin 2 (DATA) → ESP32 GPIO 2

DHT22 Pin 3 (GND) → ESP32 GND

DHT22 Pin 4 → Not connected

4.3 Software Architecture

ESP32 Firmware Layer: MicroPython-based firmware handles WiFi connectivity, MQTT client management, DHT22 sensor reading at 30-second intervals, and data publishing to the broker.

MQTT Message Broker (Mosquitto): Lightweight message broker installed on Raspberry Pi that accepts sensor data from ESP32 and distributes it to Node-RED subscribers on the local network at port 1883.

Node-RED Processing Engine: Flow-based visual programming environment that ingests MQTT data, applies business logic, calculates metrics, monitors thresholds, and routes alerts. Runs on Raspberry Pi with built-in dashboard capability.

User Interfaces:

- Node-RED Dashboard UI: Real-time visualization accessible via browser at `http://raspi-ip:1880/ui/`
- Telegram Bot: Messaging interface for alerts and commands via Telegram API

4.4 Data Flow & Topic Structure

MQTT Topics Used:

ProIT_IoT/Ravi/temp → Temperature readings (e.g., "23.5")

ProIT_IoT/Ravi/humi → Humidity readings (e.g., "55")

Data Flow Steps:

1. ESP32 reads DHT22 sensor every 30 seconds
2. Temperature and humidity values published to respective MQTT topics
3. Node-RED subscribes and receives the messages
4. Data validated and converted to numeric format
5. Values routed simultaneously to:
    a. Dashboard UI nodes for real-time display
    b. Alert logic for threshold checking
    c. Data accumulator for daily reports
6. If thresholds exceeded → Telegram alert sent immediately
7. Daily report compiled and sent at scheduled time

5. Algorithms and Calculations

4.5 Dew Point Calculation

What is Dew Point? Dew point is the temperature at which air becomes saturated with moisture. It tells us how much moisture is actually in the air. If the dew point is high, the air feels humid. If it's low, the air feels dry. Unlike relative humidity which changes with temperature, dew point is more stable and better for understanding real moisture conditions.

Magnus-Tetens Formula The system calculates dew point using the Magnus-Tetens approximation formula. This formula takes temperature and relative humidity as inputs and outputs the dew point temperature. The formula is:

Dew Point = (b * f(T,RH)) / (a - f(T,RH))

where:

f(T,RH) = (a * T / (b + T)) + ln(RH/100)

a = 17.27

b = 237.7

T = temperature in °C

RH = relative humidity in %

The formula gives accurate results for the temperature range we're working with (-40°C to +80°C). Node-RED implements this formula in a JavaScript function node that calculates the result whenever new sensor data arrives.

Why This Formula? The Magnus-Tetens formula is chosen because it's accurate, computationally lightweight (doesn't require much processing power), and works well for typical environmental conditions. More complex formulas exist but aren't necessary for this application.

4.6 Absolute Humidity

What is Absolute Humidity? Absolute humidity is the actual mass of water vapor in the air, measured in grams per cubic meter (g/m^3). Unlike relative humidity which is a percentage, absolute humidity tells you the real amount of moisture. Two rooms can have the same relative humidity but different absolute humidity if they're at different temperatures.

Calculation Method Absolute humidity is calculated from temperature and relative humidity using the formula:

AH = (6.112 * e^((17.27 * T) / (237.7 + T)) * RH) / ((273.15 + T) * 100)

where:

T = temperature in °C

RH = relative humidity (0-100%)

e = mathematical constant (2.71828)

AH = absolute humidity in g/m³

This calculation also runs in a Node-RED function node and updates whenever sensor data changes.

4.7 Threshold Monitoring

Temperature Thresholds The system monitors two temperature limits:

- **High threshold:** 25°C (default) - Alert if temperature exceeds this
- **Low threshold:** 10°C (default) - Alert if temperature falls below this

The alert logic compares the current temperature against both thresholds. If either threshold is crossed, an alert is generated and sent to Telegram immediately.

Humidity Thresholds Similarly, humidity is monitored against:

- **High threshold:** 65% (default) - Alert if humidity exceeds this
- **Low threshold:** 30% (default) - Alert if humidity falls below this

Alert Generation When a threshold is crossed, the system generates an alert message. The message includes the sensor reading, which threshold was breached, and a timestamp. This alert is then sent to the user via Telegram within 5 seconds.

4.8 Comfort Status

Comfort Categories Based on dew point value, the system classifies air conditions into comfort categories:

- **Very Dry** (dew point < 0°C) - Very uncomfortable, skin feels dry
- **Dry** (0°C to 10°C) - Comfortable for most people
- **Comfortable** (10°C to 15°C) - Ideal indoor conditions
- **Humid** (15°C to 20°C) - Starting to feel muggy
- **Very Humid** (dew point > 20°C) - Uncomfortable, feels sticky

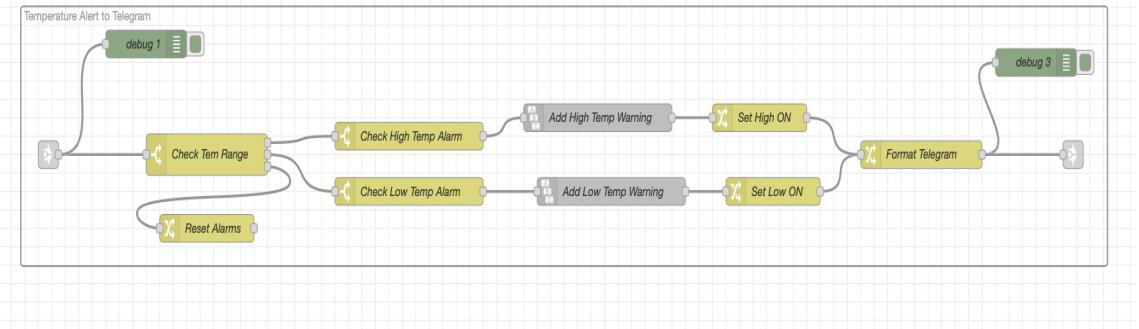
4.9 Alert Logic

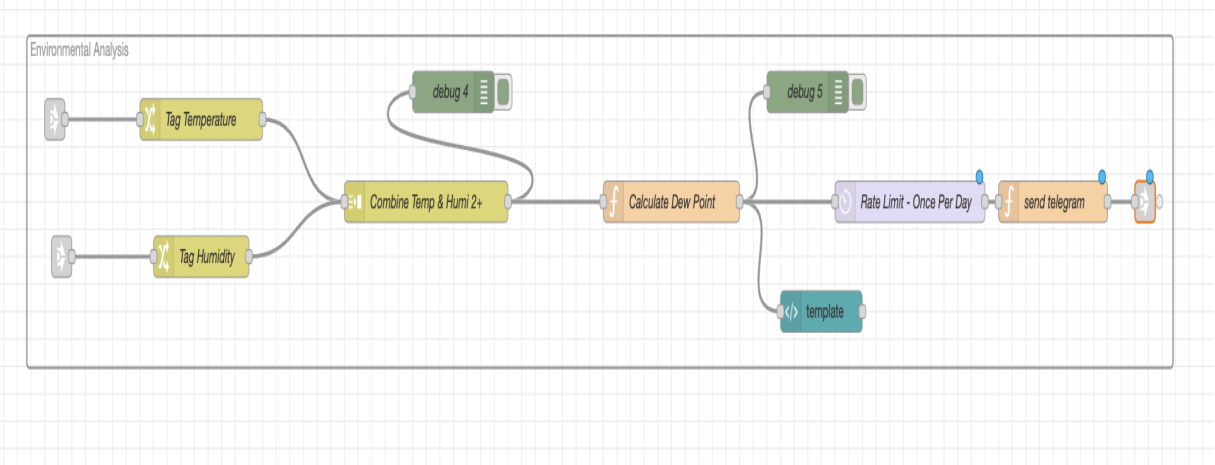
Different Alert Types The system has two different types of alerts with different delivery methods:

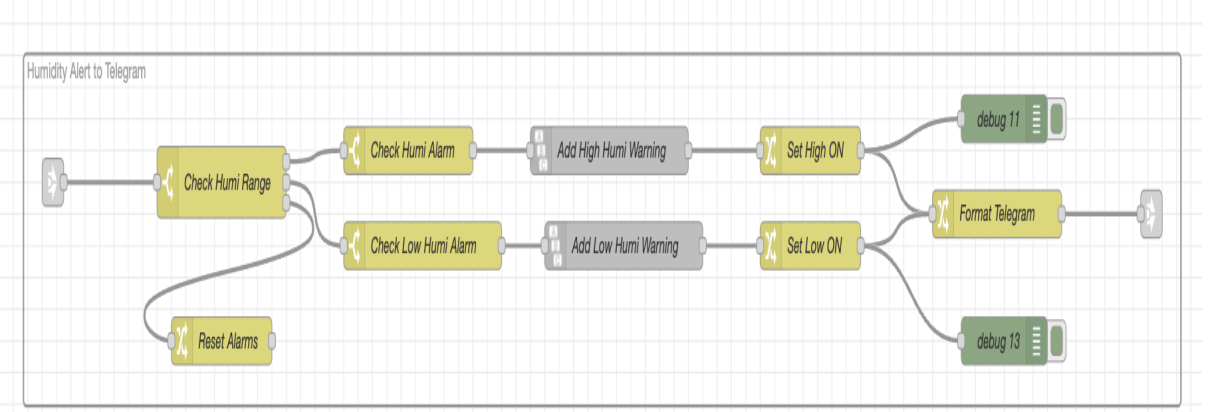
Temperature and Humidity Alerts When temperature or humidity crosses a threshold, an alert is sent immediately to Telegram. The alert message includes the current value, which threshold was exceeded, and a timestamp. Each threshold breach triggers a separate alert notification to inform the user promptly of environmental changes.

Environmental Analysis Alert The Environmental Analysis flow sends a comprehensive daily report once per 24-hour period. This includes all calculated metrics like dew point, absolute humidity, saturation depression, and comfort status. Unlike the threshold alerts which trigger on specific breaches, this alert provides a complete daily summary regardless of whether thresholds were crossed.

Why Different Delivery Methods? Temperature and humidity alerts need to notify quickly when conditions change, so they are sent immediately upon threshold breach. The environmental analysis report is informational and is sent once daily to avoid overwhelming the user with frequent updates. This balances immediate alerts for important environmental changes with useful daily summaries for trend analysis.




Environmental Analysis
Tag Temperature
Tag Humidity
debug 4
Combine Temp & Humi 2+
Calculate Dew Point
debug 5
Rate Limit - Once Per Day
send telegram
template


Humidity Alert to Telegram
Check Humi Range
Check Humi Alarm
Add High Humi Warning
Set High ON
debug 11
Format Telegram
Check Low Humi Alarm
Add Low Humi Warning
Set Low ON
Reset Alarms
debug 13

5. Implementation

5.1 Hardware Setup

ESP32 Firmware Installation:

- Flash MicroPython firmware using Thonny IDE
- Download firmware from MicroPython official site
- Connect ESP32 via USB and upload firmware

DHT22 Connection:

- DHT22 Pin 1 (VCC) → ESP32 3V3
- DHT22 Pin 2 (DATA) → ESP32 GPIO 2
- DHT22 Pin 3 (GND) → ESP32 GND
- Verify connections with multimeter (no shorts)

Sensor Testing:

- Test DHT22 reading with simple MicroPython script
- Compare readings with reference thermometer
- Verify temperature and humidity values are within expected ranges

---

5.2 Network Configuration

ESP32 WiFi Setup:

- Connect to "Rechnernetze" network
- WiFi password: rnFIW625
- Verify IP address assignment from DHCP

Raspberry Pi Setup:

- Access assigned Raspberry Pi
- Connect to same WiFi network

5.3 MQTT Broker Installation

Mosquitto Installation:

bash

```
sudo apt install mosquitto mosquitto-clients -y

sudo systemctl start mosquitto

sudo systemctl enable mosquitto
```

Verification:

- Check broker status: `sudo systemctl status mosquitto`
- Confirm listening on port 1883
- Test with `mosquitto_sub` and `mosquitto_pub`

5.4 ESP32 Firmware Development

Required Libraries:

- MicroPython dht library (built-in)
- umqttsimple.py (download and upload to ESP32)

Main Code (boot.py):

python

```
import dht

from machine import Pin

from umqttsimple import MQTTClient

import network

import time


# Configuration

WIFI_SSID = 'Rechnernetze'

WIFI_PASSWORD = 'rnFIW625'
```

```
MQTT_BROKER = 'raspi3e26.f4.htw-berlin.de'
MQTT_PORT = 1883
MQTT_CLIENT_ID = 'Ravi'
MQTT_TOPIC_TEMP = 'ProIT_IoT/Ravi/temp'
MQTT_TOPIC_HUMI = 'ProIT_IoT/Ravi/humi'

sensor = dht.DHT22(Pin(2))

def connect_wifi():
    sta_if = network.WLAN(network.STA_IF)
    sta_if.active(True)
    sta_if.connect(WIFI_SSID, WIFI_PASSWORD)
    timeout = 0
    while not sta_if.isconnected() and timeout < 20:
        time.sleep(1)
        timeout += 1
    return sta_if.isconnected()

def connect_mqtt():
    client = MQTTClient(MQTT_CLIENT_ID, MQTT_BROKER, port=MQTT_PORT)
    try:
        client.connect()
        return client
    except:
        return None

# Main Loop
if connect_wifi():
```

```
client = connect_mqtt()
if client:
    while True:
        try:
            sensor.measure()
            temperature = sensor.temperature()
            humidity = sensor.humidity()
            client.publish(MQTT_TOPIC_TEMP, str(temperature))
            client.publish(MQTT_TOPIC_HUMI, str(humidity))
            print(f"Temp={temperature}°C, Humi={humidity}%")
            time.sleep(30)
        except Exception as e:
            print("Error:", e)
            time.sleep(5)
```

Deployment:

- Save as boot.py on ESP32
- Runs automatically on power-on
- Verify: Messages appear every 30 seconds in `mosquitto_sub`

5.5 Node-RED Setup

Installation:

bash

```
sudo npm install -g --unsafe-perm node-red
sudo systemctl start node-red
sudo systemctl enable node-red
```

Package Installation:

bash

```
npm install node-red-dashboard
```

```
npm install node-red-contrib-telegram
```

Access: `http://raspberry-pi-ip:1880`

5.6 MQTT Data Ingestion

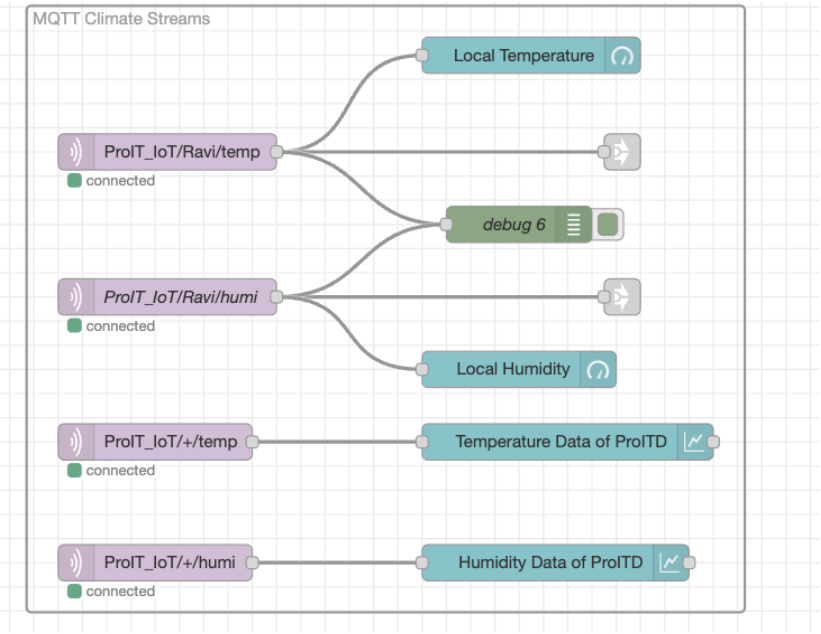
Add MQTT In Nodes:

- Node 1: Topic `ProIT_IoT/Ravi/temp`, QoS 1, Server localhost:1883
- Node 2: Topic `ProIT_IoT/Ravi/humi`, QoS 1, Server localhost:1883

Add Link In Nodes:

- TempIn1: Routes temperature data
- HumiIn1: Routes humidity data

Wire: MQTT nodes → Link in nodes



Testing: Debug nodes should show messages every 30 seconds

5.7 Dashboard Widgets

Temperature Gauge:

- Type: ui_gauge
- Range: 0-50°C
- Color zones: Green (15-25), Yellow (warning), Red (alert)
- Wire: TempIn1 → Gauge

Humidity Gauge:

- Type: ui_gauge
- Range: 0-100%
- Wire: HumiIn1 → Gauge

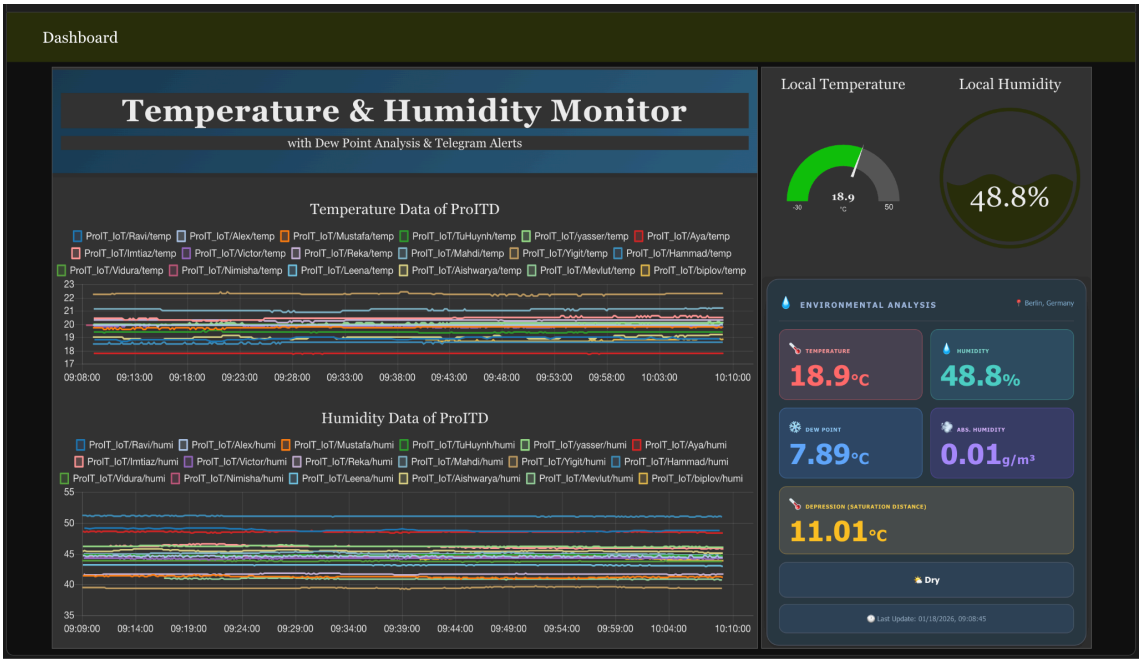
Temperature Chart:

- Type: ui_chart (Line)
- Label: "Temperature Trend (24h)"
- Wire: TempIn1 → Chart

Humidity Chart:

- Type: ui_chart (Line)
- Label: "Humidity Trend (24h)"
- Wire: HumiIn1 → Chart

View:http://rpi-ip:1880/ui/



5.8 Environmental Calculations

Dew Point Function:

javascript

```
var T = parseFloat(msg.payload);
var RH = flow.get('humidity') || 50;
var a = 17.27, b = 237.7;
var alpha = ((a * T) / (b + T)) + Math.log(RH / 100.0);
msg.dew_point = parseFloat(((b * alpha) / (a - alpha)).toFixed(2));
return msg;
```

Absolute Humidity Function:

javascript

```
var T = parseFloat(msg.payload);
var RH = flow.get('humidity') || 50;
var exp_term = (17.27 * T) / (237.7 + T);
```

```
msg.abs_humidity = parseFloat(((6.112 * Math.exp(exp_term) * RH) / ((273.15 + T) * 100)).toFixed(2));

return msg;
```

Display Metrics:

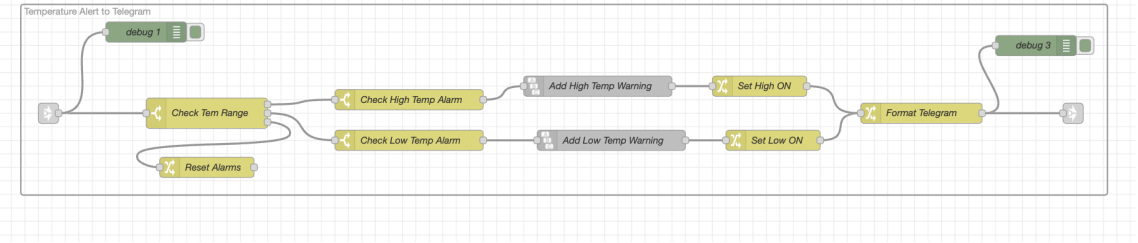
- Type: ui_template
- Show Dew Point, Absolute Humidity, Saturation Depression

5.9 Alert System

Your system uses Node-RED switch and change nodes (not JavaScript functions) to detect threshold breaches and format alert messages.

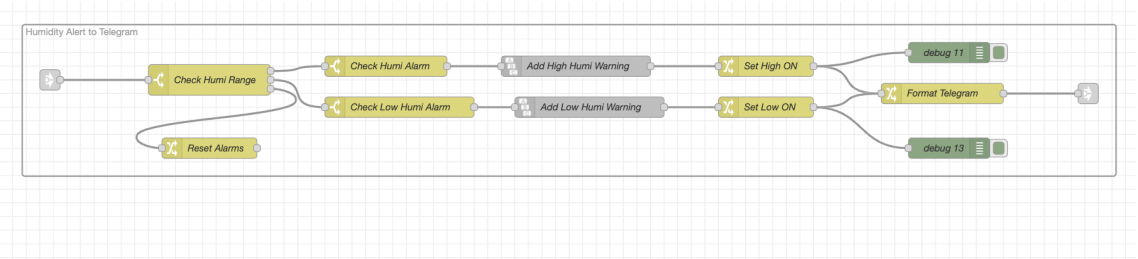
Temperature Alert Logic:

- Switch node checks if temperature > 25°C → output " High temperature alert!"
- Switch node checks if temperature < 10°C → output " Low temperature alert!"



Humidity Alert Logic:

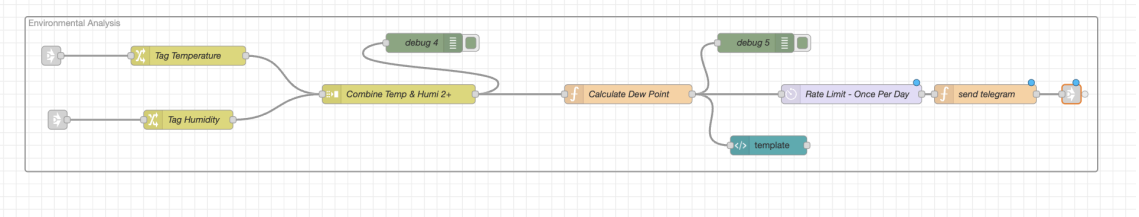
- Switch node checks if humidity > 65% → output " High humidity alert!"
- Switch node checks if humidity < 30% → output " Low humidity alert!"



Environmental Metrics Display:

- Function node collects all calculated metrics:
    - Temperature (°C)

- Humidity (%)
- Dew Point (°C)
- Absolute Humidity (g/m³)
- Saturation Depression (°C)



5.10 Telegram Integration

Create Bot:

- Chat @BotFather on Telegram
- Command: `/newbot`
- Copy Bot Token (123456:ABC-DEF...)

Get Chat ID:

- Send message to bot
- Visit: `https://api.telegram.org/bot<TOKEN>/getUpdates`
- Copy Chat ID from response

Add Telegram Out Node:

- Bot Token: [paste token]
- Chat ID: [paste ID]
- Wire alert nodes → Telegram out

5.11 Testing & Validation

MQTT Verification:

bash

```
mosquitto_sub -h localhost -t "ProIT_IoT/Ravi/#" -v
```

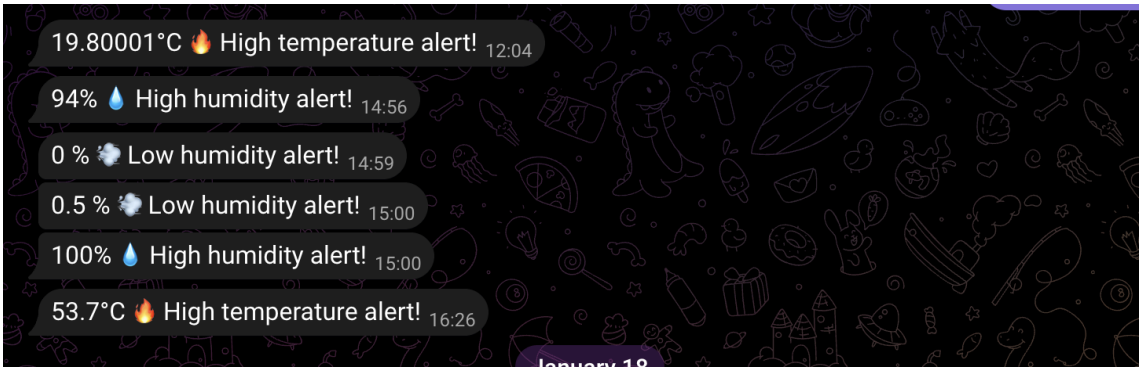
Expected: Messages every 30 seconds

Dashboard Test:

- Open `http://rpi-ip:1880/ui/`
- Verify gauges update every 30 seconds
- Check charts accumulate data

Alert Test:

- Heat sensor >25°C → Check Telegram alert
- Cool sensor <10°C → Check Telegram alert
- Increase humidity >65% → Check Telegram alert
- Decrease humidity <30% → Check Telegram alert



8. Conclusion

8.1 Summary

This project is an IoT-Based Temperature, Humidity & Dew Point Monitoring and Alert System. It measures temperature and humidity using a DHT22 sensor connected to an ESP32 microcontroller. The ESP32 publishes this data to an MQTT broker on a Raspberry Pi. Node-RED processes the data, calculates dew point using the Magnus-Tetens formula, and displays everything on an interactive dashboard. When temperature or humidity crosses set thresholds, the system sends alerts via Telegram.

8.2 Critical Review

The system operates in a simple cycle: the ESP32 reads sensor data and publishes it to MQTT every 30 seconds. Node-RED receives this data in real-time and performs calculations to derive dew point, absolute humidity, and saturation deficit. The dashboard updates automatically to show current values and historical trends. Temperature and humidity thresholds are monitored continuously - when values cross these limits, Telegram alerts are sent immediately to notify the user.

8.3 Achievements

The system successfully demonstrates a complete IoT solution. It reads environmental data accurately, calculates metrics correctly, displays information in real-time, and delivers alerts reliably. All components work together as intended. The dashboard shows current conditions and trends, and the Telegram bot provides instant notifications when environmental conditions change. The system operates continuously without requiring manual intervention.

9. References

1. Node-RED – “Node-RED Documentation.” *https://nodered.org/docs/*
2. Espressif Systems – “ESP32 Technical Reference Manual.” https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf
3. Aosong – “DHT22 (AM2302) Temperature and Humidity Sensor Datasheet.” https://cdn-shop.adafruit.com/datasheets/DHT22.pdf
4. Raspberry Pi Foundation – “Raspberry Pi Computers Documentation.” https://www.raspberrypi.com/documentation/computers/raspberry-pi.html
5. Eclipse Mosquitto – “Mosquitto MQTT Broker Documentation.” https://mosquitto.org/documentation/
6. MicroPython – “MicroPython Documentation.” https://docs.micropython.org/en/latest/
7. Telegram – “Telegram Bot API.” https://core.telegram.org/bots/api
8. Thonny – “Thonny Python IDE for Beginners.” https://thonny.org/