Real-Time Weather Monitoring System with AWS IoT Core and DynamoDB

A Course Project Report Submitted in partial fulfillment of the course requirements for the award of grades in the subject of

CLOUD BASED AIML SPECIALITY
(22SDCS07A)

by

DESHABOINA REVANTH KUMAR

(2210030452)

Under the esteemed guidance of

Ms. P. Sree Lakshmi
Assistant Professor,
Department of Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

K L Deemed to be UNIVERSITY

Aziznagar, Moinabad, Hyderabad,
Telangana, Pincode: 500075

April 2025

K L Deemed to be UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is Certified that the project entitled **“**Real-Time Weather Monitoring System with AWS IoT Core and DynamoDB**”** which is a experimental &/ theoretical &/ Simulation&/ hardware work carried out by DESHABOINA REVANTH KUMAR (2210030452), in partial fulfillment of the course requirements for the award of grades in the subject of **CLOUD BASED AIML SPECIALITY**, during the year **2024-2025**. The project has been approved as it satisfies the academic requirements.

Ms.P.Sree Lakshmi	**Dr. Arpita Gupta**
Course Coordinator	**Head of the Department**

Ms. P. Sree Lakshmi

Course Instructor

CONTENTS

1. INTRODUCTION

Real-time weather monitoring plays a crucial role in enabling smarter decision-making across sectors such as agriculture, urban planning, transportation, and disaster management. Traditionally, these systems rely on physical sensors and embedded devices to collect environmental data. However, with the increasing availability of powerful cloud services and public APIs, it is now possible to build intelligent, real-time weather monitoring solutions without depending on specialized hardware.

This project explores a cloud-native approach to real-time weather tracking using AWS services and the OpenWeatherMap API [6]. Instead of deploying physical sensors, weather data such as temperature, humidity, and general conditions are fetched programmatically for various cities. This data is then processed through a serverless backend using AWS Lambda [2], stored in DynamoDB [3], and broadcasted to listeners via AWS IoT Core [4]. If extreme weather conditions are detected (e.g., temperatures exceeding 35°C), the system sends out alerts using Amazon SNS (Simple Notification Service) [5].

The solution demonstrates how modern cloud infrastructure can replace physical components while maintaining the core functionality of an IoT system. It also highlights the advantages of using a serverless architecture, such as automatic scaling, cost efficiency, and ease of deployment. Furthermore, by simulating an IoT ecosystem with real-time alerting and data processing, the project showcases the potential for rapid prototyping in smart environment applications without significant capital investment.

Overall, this system offers a scalable, cost-effective, and hardware-free way to simulate and analyze weather data in real time, making it highly applicable in both academic and practical use cases.

2. AWS Services Used as part of the project

1. Amazon API Gateway – This service serves as the main interface for external users or applications to interact with the backend system. It securely exposes the AWS Lambda function as a RESTful API endpoint. In this project, API Gateway is used to accept HTTP POST requests containing JSON-formatted weather data, allowing seamless integration between the external data source (OpenWeatherMap API) and the AWS cloud infrastructure [1].

2. AWS Lambda – Lambda is the core compute service where the business logic of the project resides. Upon receiving a request from API Gateway, the Lambda function parses the incoming weather data, stores it into DynamoDB, and performs analysis to determine if it is the highest recorded temperature. If a critical temperature threshold is reached (e.g., above 35°C), it triggers further actions such as publishing a message to IoT Core and sending notifications via SNS. This event-driven and serverless approach enables cost-effective and scalable computation without managing servers [2].

3. Amazon DynamoDB – DynamoDB acts as the persistent storage layer for all weather records. It is a fully managed NoSQL database that offers fast and predictable performance. Each weather record includes fields like id, city, temperature, humidity, and condition. The system performs scans on this table to evaluate the highest temperature recorded so far, which makes it suitable for real-time analytics on incoming weather data [3].

4. AWS IoT Core – IoT Core is used to simulate real-world IoT functionality by enabling the publication of weather alerts to specific MQTT topics. In this project, it broadcasts weather updates and notifications to the weather/alert topic. Subscribers (like dashboards, mobile apps, or other backend services) can receive these updates in real-time, making it ideal for building responsive systems that act on changing environmental conditions [4].

5. Amazon Simple Notification Service (SNS) – SNS is used for sending email alerts when the recorded temperature crosses a defined threshold (35°C in this case). This helps in implementing emergency alert systems by immediately notifying users via email, ensuring timely awareness and response. SNS is highly scalable and supports various notification formats, but email was used here due to its simplicity and effectiveness [5].

3. Steps involved in solving project problem statement

Step 1: Fetch Weather Data

To simulate real-world IoT sensors, this project integrates the OpenWeatherMap API [6], which provides real-time weather data for any specified city. When the user enters a city name, the API returns a structured JSON response that includes temperature, humidity, and weather conditions. These values are parsed from the API response and packaged into a JSON object to be used in the next step. This approach eliminates the need for physical sensors while maintaining realistic input behavior.

Step 2: Trigger Lambda via API Gateway

The collected data is sent as a POST request to a REST endpoint created using Amazon API Gateway [1]. This API Gateway serves as the primary interface for external systems and securely connects incoming HTTP requests to the backend. Once triggered, it invokes an AWS Lambda function [2], which contains the core business logic of the system. The Lambda function acts as a lightweight, serverless compute layer that parses the input, validates it, and performs subsequent operations such as storage, alert generation, and data broadcasting.

Step 3: Store Data in DynamoDB

Once the Lambda function processes the input data, it stores the information in Amazon DynamoDB [3], a fast and scalable NoSQL database. Each weather record is stored as a separate item, with fields such as id, city, temperature, humidity, and condition. This persistent storage not only allows for historical tracking of weather entries but also supports querying and analytics such as identifying extreme weather patterns.

Step 4: Analyze Data and Send Alerts

After the data is stored, the Lambda function retrieves all weather entries from the database using **a** Scan operation. It then identifies the highest recorded temperature among all entries. If the temperature of the current entry exceeds a predefined threshold (e.g., 35°C), an alert email is triggered using Amazon SNS (Simple Notification Service) [5]. This ensures emergency conditions are communicated immediately, simulating real-world disaster alert systems.

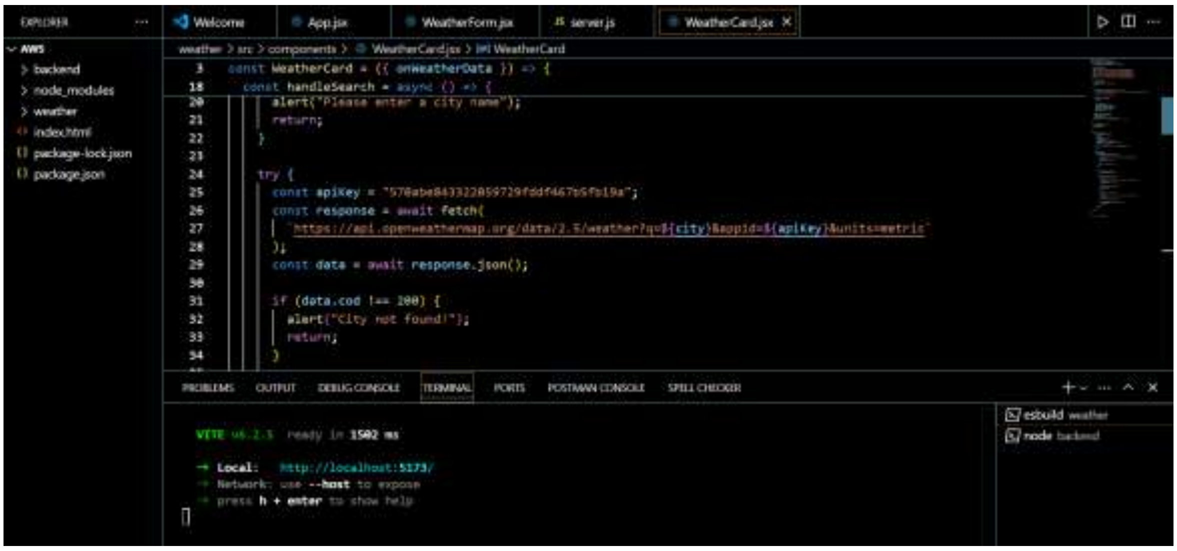
Step 5: Publish to AWS IoT Core

Simultaneously, the Lambda function publishes the latest weather update to the AWS IoT Core topic weather/alert [4]. This mimics a real-time IoT environment where devices publish sensor data to the cloud. Any application or dashboard subscribed to this topic receives live updates, enabling real-time visualizations or decisions based on the latest weather data. This completes the IoT simulation loop, even without physical hardware.

4. Stepwise Screenshots with brief description

Step 1: Fetching Weather Data from OpenWeatherMap API

The system collects real-time weather data using the OpenWeatherMap API, which includes temperature, humidity, and condition for a specific city. This replaces the need for physical IoT sensors. [6]



```
const WeatherCard = ({ onWeatherData }) => {
  const handleSearch = async () => {
      alert("Please enter a city name");
      return;
    }

    try {
      const apiKey = "570abe843322859729fddf467b5fb19a";
      const response = await fetch(
        `https://api.openweathermap.org/data/2.5/weather?q=${city}&appid=${apiKey}&units=metric`
      );
      const data = await response.json();

      if (data.cod !== 200) {
        alert("City not found!");
        return;
      }
```

```
VITE v6.2.5  ready in 1502 ms

➜  Local:   http://localhost:5173/
➜  Network: use --host to expose
➜  press h + enter to show help
```

Fig1.1 openweathermapAPI to fetch Weather Data

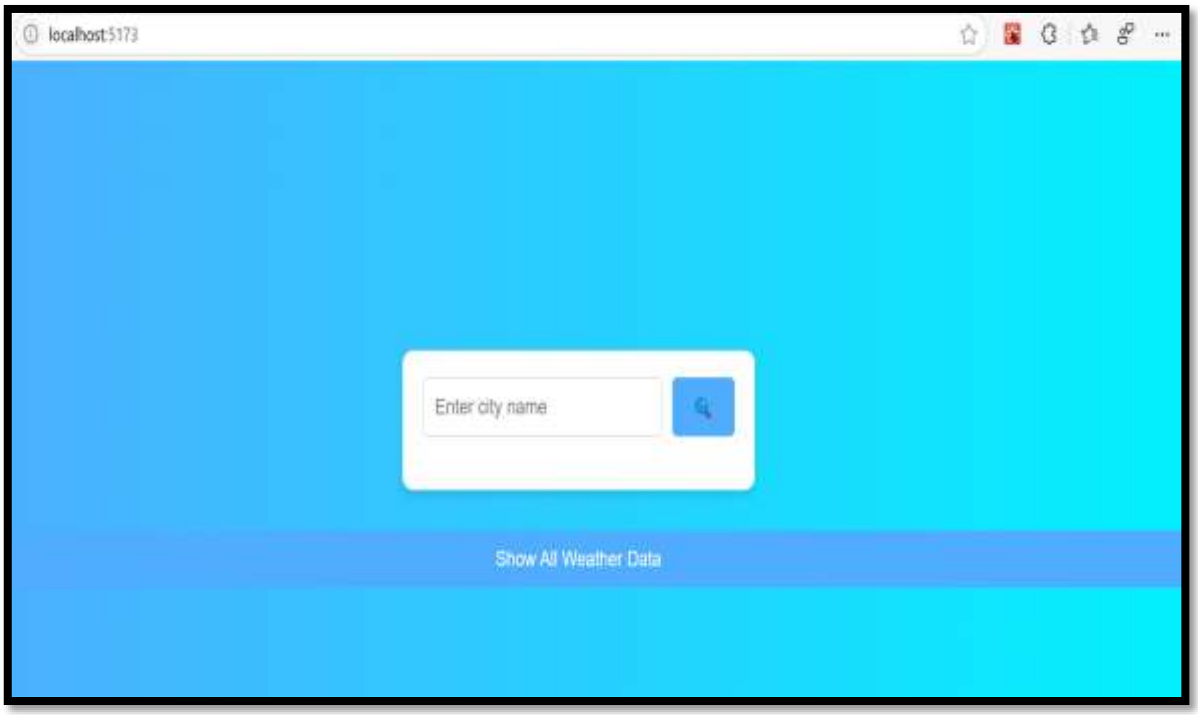


Fig1.2 Enter City to get Weather data

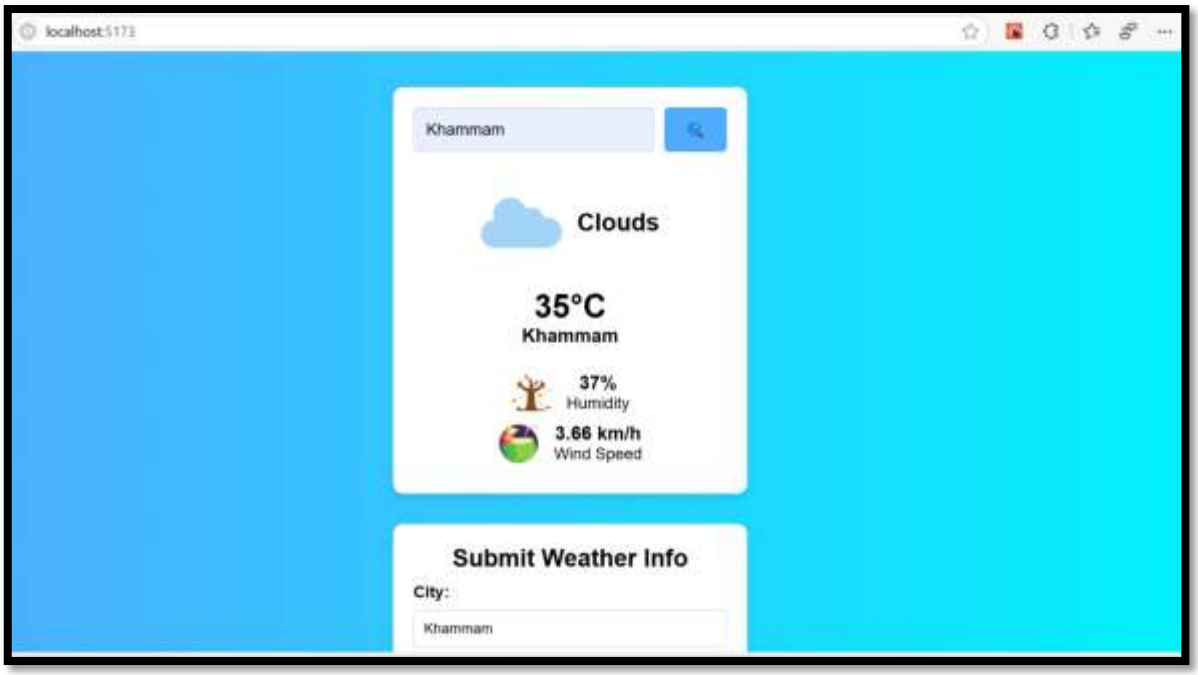
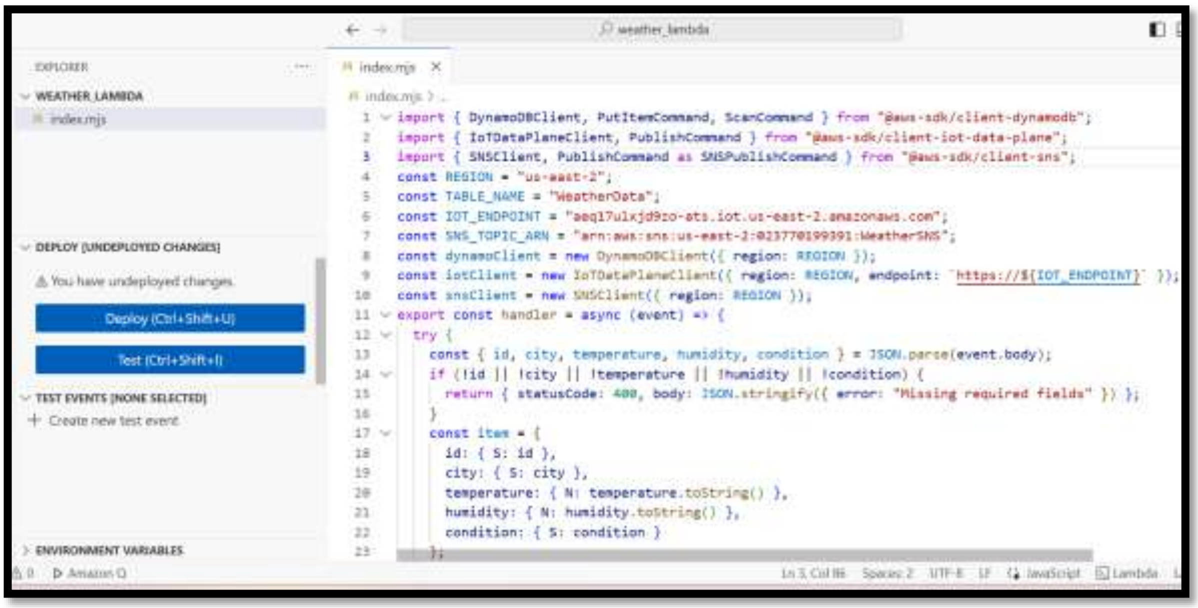


Fig1.3 Weather Data fetched from API

Step 2: Lambda Function to Process Incoming Data

An AWS Lambda function is triggered with incoming JSON data. It parses the weather data and prepares it for storage and broadcasting. This serverless function acts as the processing unit of the IoT system. [2]

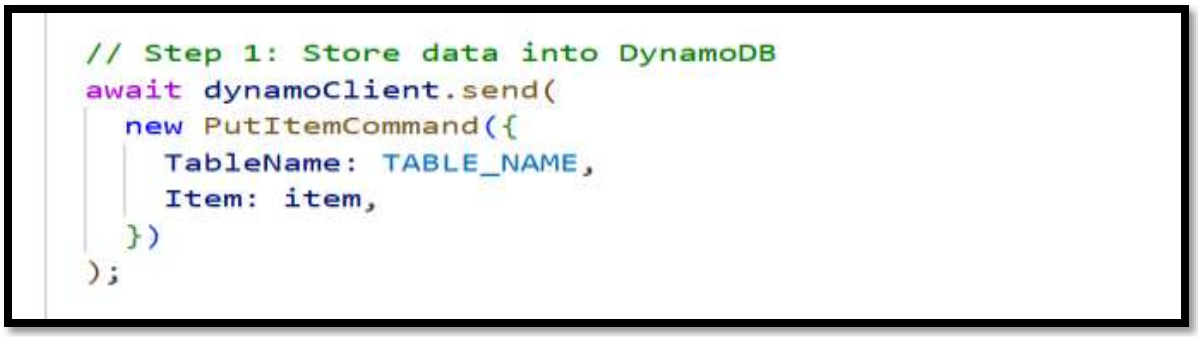


```javascript
import { DynamoDBClient, PutItemCommand, ScanCommand } from "@aws-sdk/client-dynamodb";
import { IoTDataPlaneClient, PublishCommand } from "@aws-sdk/client-iot-data-plane";
import { SNSClient, PublishCommand as SNSPublishCommand } from "@aws-sdk/client-sns";
const REGION = "us-east-2";
const TABLE_NAME = "WeatherData";
const IOT_ENDPOINT = "aeq17ulxjd9zo-ats.iot.us-east-2.amazonaws.com";
const SNS_TOPIC_ARN = "arn:aws:sns:us-east-2:023770199391:WeatherSNS";
const dynamoClient = new DynamoDBClient({ region: REGION });
const iotClient = new IoTDataPlaneClient({ region: REGION, endpoint: `https://${IOT_ENDPOINT}` });
const snsClient = new SNSClient({ region: REGION });
export const handler = async (event) => {
  try {
    const { id, city, temperature, humidity, condition } = JSON.parse(event.body);
    if (!id || !city || !temperature || !humidity || !condition) {
      return { statusCode: 400, body: JSON.stringify({ error: "Missing required fields" }) };
    }
    const item = {
      id: { S: id },
      city: { S: city },
      temperature: { N: temperature.toString() },
      humidity: { N: humidity.toString() },
      condition: { S: condition }
    };
```

Fig2.1 Lambda code

Step 3: Storing Data into DynamoDB

The structured weather data is inserted into a DynamoDB table called WeatherData. DynamoDB allows for fast, scalable storage of semi-structured data. [3]



```
// Step 1: Store data into DynamoDB
await dynamoClient.send(
  new PutItemCommand({
    TableName: TABLE_NAME,
    Item: item,
  })
);
```

Fig3.1 Code to store data in DynamoDB

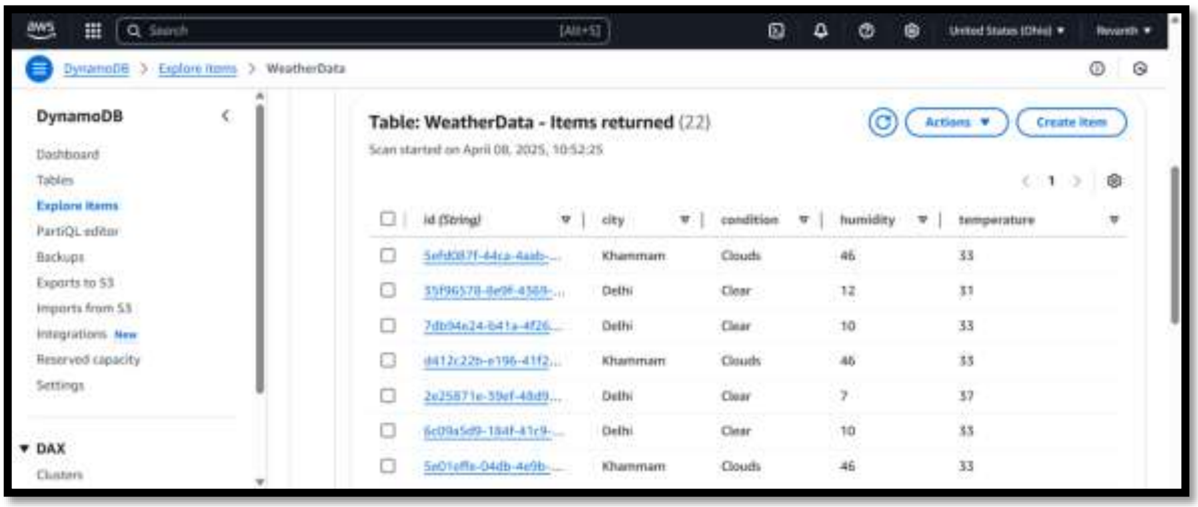


Table: WeatherData - Items returned (22)

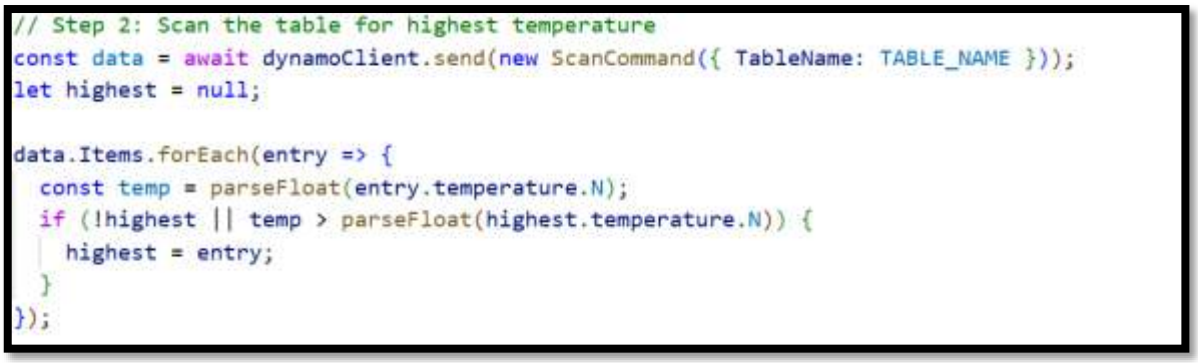
Scan started on April 08, 2025, 10:52:25

id (String)	city	condition	humidity	temperature
5efd087f-44ca-4aab-...	Khammam	Clouds	46	33
35f96578-8e9f-4369-...	Delhi	Clear	12	31
7db94e24-b41a-4f26...	Delhi	Clear	10	33
d412c22b-e196-41f2...	Khammam	Clouds	46	33
2e25871e-39ef-48d9...	Delhi	Clear	7	37
6c09a5d9-184f-41c9-...	Delhi	Clear	10	33
5e01effe-04db-4e9b-...	Khammam	Clouds	46	33

Fig3.2 DynamoDB Table

Step 4: Finding the Highest Temperature Record

The Lambda function scans the DynamoDB table to determine the highest recorded temperature. This logic helps identify alert-worthy data points.



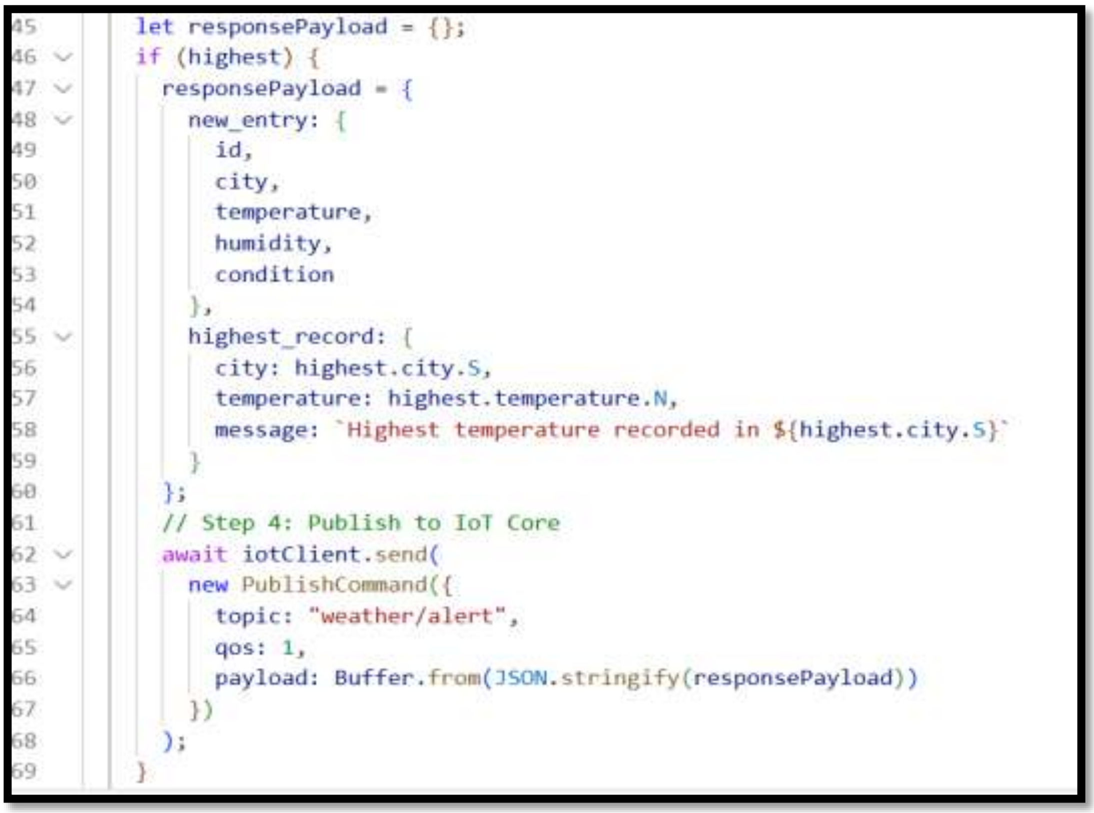
```
// Step 2: Scan the table for highest temperature
const data = await dynamoClient.send(new ScanCommand({ TableName: TABLE_NAME }));
let highest = null;

data.Items.forEach(entry => {
  const temp = parseFloat(entry.temperature.N);
  if (!highest || temp > parseFloat(highest.temperature.N)) {
    highest = entry;
  }
});
```

Fig4 Code to scan and find highest temperature

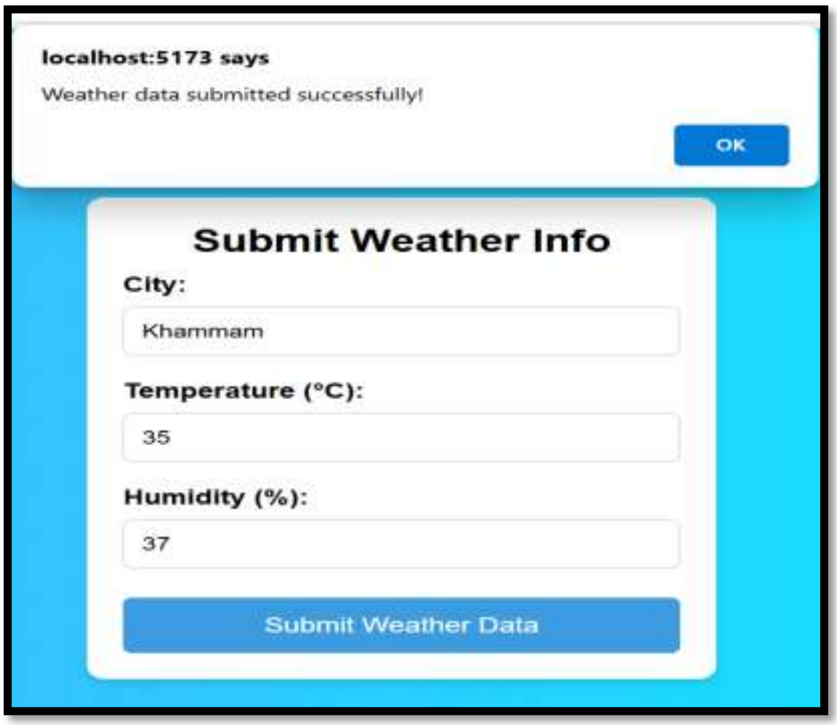
Step 5: Publishing Alerts to AWS IoT Core

If a new temperature reading is valid, it is published to the topic weather/alert via AWS IoT Core. This mimics real-time data delivery in a typical IoT pipeline. [4]



```
let responsePayload = {};
if (highest) {
  responsePayload = {
    new_entry: {
      id,
      city,
      temperature,
      humidity,
      condition
    },
    highest_record: {
      city: highest.city.S,
      temperature: highest.temperature.N,
      message: `Highest temperature recorded in ${highest.city.S}`
    }
  };
  // Step 4: Publish to IoT Core
  await iotClient.send(
    new PublishCommand({
      topic: "weather/alert",
      qos: 1,
      payload: Buffer.from(JSON.stringify(responsePayload))
    })
  );
}
```

Fig5.1 Code to send alert to MQTT



localhost:5173 says

Weather data submitted successfully!

OK

Submit Weather Info

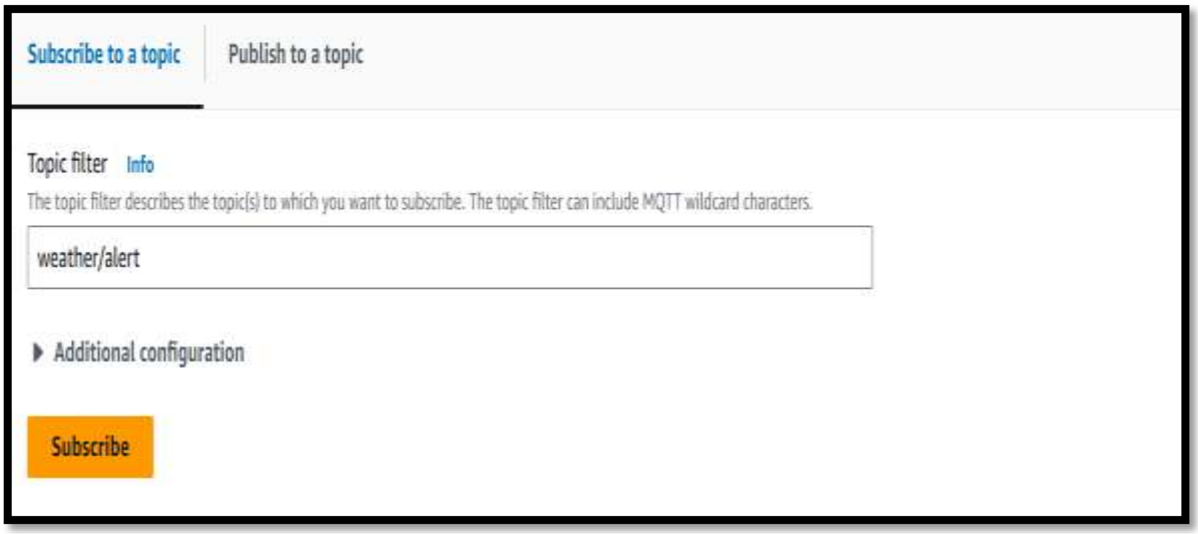
City: Khammam

Temperature (°C): 35

Humidity (%): 37

Submit Weather Data

Fig5.2 Weather data form to store in DynamoDB


Subscribe to a topic | Publish to a topic

Topic filter Info

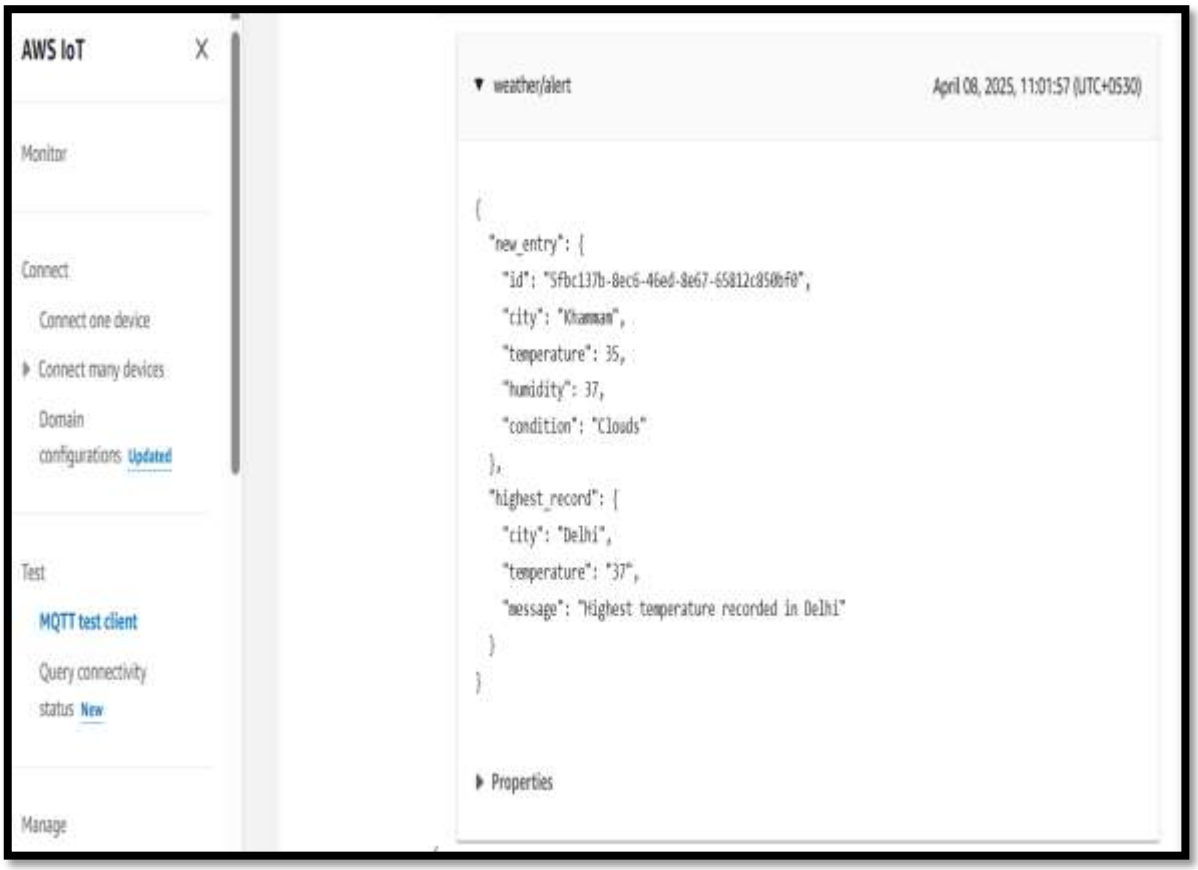
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

Topic filter: weather/alert

▶ Additional configuration

Subscribe

Fig5.3 Subscribing to weather/alert topic in MQTT



▼ weather/alert   April 08, 2025, 11:01:57 (UTC+0530)

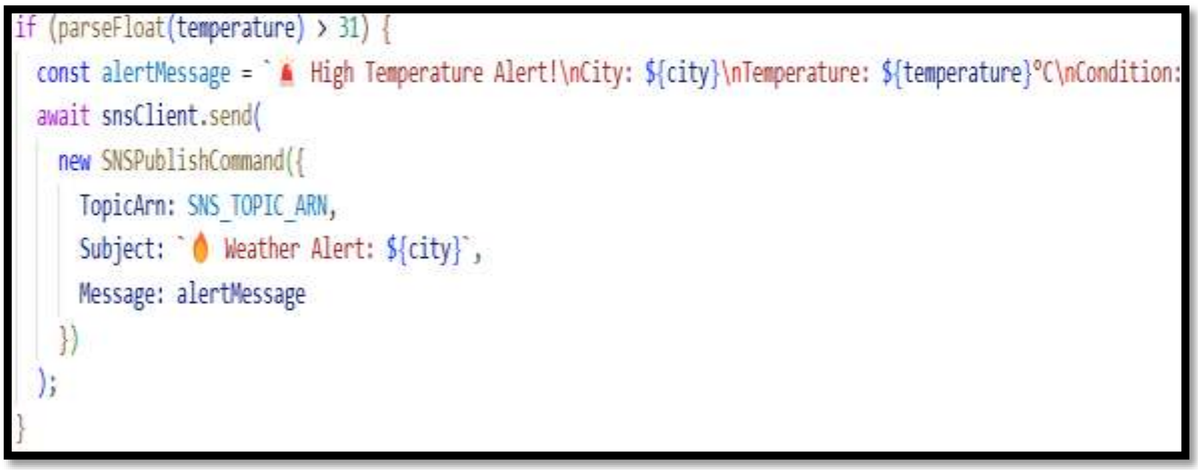
```
{
  "new_entry": {
    "id": "5fbc137b-8ec6-46ed-8e67-65812c850bf0",
    "city": "Khammam",
    "temperature": 35,
    "humidity": 37,
    "condition": "Clouds"
  },
  "highest_record": {
    "city": "Delhi",
    "temperature": "37",
    "message": "Highest temperature recorded in Delhi"
  }
}
```

▶ Properties

Fig5.4 Alert Message from Subscribed topic whenever DynamoDB receives new record

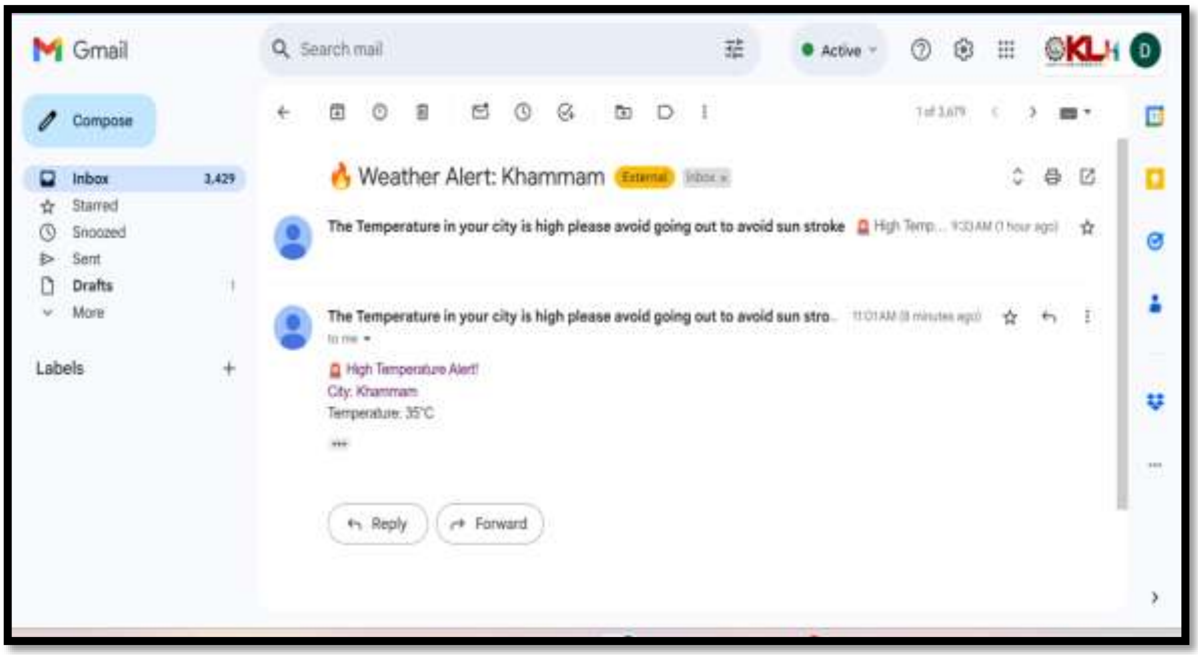
Step 6: Sending SMS/Email Alert via Amazon SNS

If the temperature exceeds 35°C, an alert is sent through Amazon SNS to a subscribed email address or phone number. This ensures timely notification of extreme weather. [5]



```
if (parseFloat(temperature) > 31) {
  const alertMessage = `🚨 High Temperature Alert!\nCity: ${city}\nTemperature: ${temperature}°C\nCondition:
  await snsClient.send(
    new SNSPublishCommand({
      TopicArn: SNS_TOPIC_ARN,
      Subject: `🔥 Weather Alert: ${city}`,
      Message: alertMessage
    })
  );
}
```

Fig6.1 SNS code to send notification when it receives temperature>31



🔥 Weather Alert: Khammam External Inbox

The Temperature in your city is high please avoid going out to avoid sun stroke 🚨 High Temp... 9:33 AM (1 hour ago)

The Temperature in your city is high please avoid going out to avoid sun stro... 11:01 AM (8 minutes ago)
to me

🚨 High Temperature Alert!
City: Khammam
Temperature: 35°C
...

Fig6.2 Email receives notification when Temperature>31 it alerts

5. Learning Outcomes

By completing this project, the following skills and insights were gained:

Gained practical experience in integrating various AWS services such as Lambda, DynamoDB, IoT Core, and SNS, enabling the development of a full-stack serverless application. The project demonstrated how these services can work together to process, store, and communicate real-time data effectively. [2][3][4][5]

1. Working with Serverless Architecture

Learned to build a serverless architecture using AWS Lambda, eliminating the need to manage underlying infrastructure. This contributed to a more scalable, cost-effective, and easily deployable solution. [2]

2. Handling Real-Time Data Flows

Understood the fundamentals of real-time data collection and processing by utilizing the OpenWeatherMap API instead of physical sensors. This highlighted an effective alternative approach for IoT simulation and real-world testing. [6]

3. Data Management Using DynamoDB

Gained hands-on experience in designing and managing NoSQL data using DynamoDB, including inserting, scanning, and querying data for actionable insights. [3]

4. Implementing Alert Systems Using SNS

Learned to create and configure notification services using Amazon SNS, including topic creation, subscription management, and message publishing based on custom logic (e.g., temperature > 35°C). [5]

5. Publishing Data to IoT Core Topics

Successfully implemented the mechanism of publishing structured messages to AWS IoT Core topics, simulating how smart sensors might communicate with cloud applications in real-time. [4]

6. Conclusion

The "Real-Time Weather Monitoring System with AWS IoT Core and DynamoDB" project has successfully illustrated how cloud-based solutions can replicate real-world IoT behavior without relying on physical hardware. By leveraging the **OpenWeatherMap API** [6], we were able to continuously fetch live weather data, making the system act as if it were connected to real sensors. This simulated data pipeline is especially useful in development environments, educational settings, or early-stage prototyping where physical devices might not be available.

Throughout the project, we made use of AWS Lambda for serverless compute, Amazon DynamoDB for efficient and scalable data storage**,** AWS IoT Core [4] for publishing weather updates to subscribed clients in real-time, and Amazon SNS [5] for triggering emergency alerts via email when the temperature exceeded a critical threshold (e.g., above 35°C). This multi-service integration allowed us to build a fully event-driven, reactive system that continuously monitors and responds to changing weather conditions without manual intervention.

The architecture we implemented is not only scalable and cost-efficient, but also highly adaptable. It can be extended to support additional weather metrics, user-specific alerts, or even real IoT sensor integration in the future. It can also serve as a backend for web/mobile applications that visualize weather data in real time or offer predictive insights. This proves the feasibility of using cloud platforms like AWS to solve real-world problems in areas such as climate monitoring, agriculture, public safety, and smart city infrastructure.

Moreover, the project deepened our understanding of serverless architectures, cloud automation, and real-time messaging, providing hands-on experience with real-world tools and design patterns used in the industry today. It has reinforced the potential of cloud technologies to democratize innovation, especially in fields like environmental data monitoring where responsiveness and scalability are key.

In summary, this project demonstrates the power of combining cloud computing with open data sources to create intelligent, scalable, and responsive systems that align with modern IoT use cases and environmental monitoring goals.

7. References

[1] AWS Lambda Documentation – https://docs.aws.amazon.com/lambda/

[2] Amazon DynamoDB Documentation – https://docs.aws.amazon.com/dynamodb/

[3]AWS JavaScript SDK (v3)

https://docs.aws.amazon.com/AWSJavaScriptSDK/v3/latest/index.html

[4] AWS IoT Core – https://aws.amazon.com/iot-core/

[5] AWS SNS (Simple Notification Service) – https://aws.amazon.com/sns/

[6] OpenWeatherMap API – https://openweathermap.org/api/

[7]AWS IoT Publishing Guide –

https://docs.aws.amazon.com/iot/latest/developerguide/publishing.html

[8]AWS API Gateway Integration Settings –

https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-integration-settings-integration-type.html

[9]React Documentation – https://reactjs.org/docs/getting-started.html