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

A Project Based Learning Report Submitted in partial fulfilment of the requirements for the award of the degree

of

Bachelor of Technology

in the Department of Computer Science & Engineering

Cloud Based AI/ML Speciality (22SDCS07A)

Submitted by

2210030452: D Revanth kumar

Under the guidance of

Ms. P. Sree Lakshmi

Department of Computer Science and Engineering

Koneru Lakshmaiah Education Foundation, Aziz Nagar

Aziz Nagar – 500075

March - 2025.

Introduction

Overview of the Project: The Real-Time Weather Monitoring System is a cloud-based solution designed to collect, process, and store weather data using Amazon Web Services (AWS). By leveraging weather APIs, the system fetches real-time weather information, such as temperature, humidity, and wind speed, from various locations. The data is processed and stored for analysis, enabling users to monitor weather conditions efficiently. This project aims to demonstrate the integration of AWS services for building a scalable and serverless weather monitoring application, focusing on real-time data ingestion, processing, and storage without the use of hardware components.

AWS Services Being Used and Justification: The project utilizes AWS IoT Core, AWS Lambda, Amazon API Gateway, and Amazon DynamoDB. AWS IoT Core facilitates secure data ingestion from weather APIs via MQTT protocol, ensuring reliable communication. AWS Lambda processes the incoming data in a serverless manner, reducing operational overhead. Amazon API Gateway enables the system to interact with external weather APIs securely. Amazon DynamoDB, a NoSQL database, stores the processed weather data, offering scalability and low-latency access. These services were chosen for their ability to handle real-time data workflows, provide scalability, and minimize infrastructure management, making them ideal for a weather monitoring system.

Project Purpose and Expected Outcome: The purpose of this project is to build a scalable system for real-time weather monitoring using AWS. The expected outcome is a fully functional application that ingests weather data, processes it, and stores it in DynamoDB, allowing users to access up-to-date weather information through a simple interface.

Methodology

Architecture and Workflow: The system architecture begins with Amazon API Gateway, which acts as the entry point to fetch data from a weather API (e.g., OpenWeatherMap). API Gateway triggers an AWS Lambda function to process the HTTP request and retrieve weather data. The Lambda function then publishes the data to an AWS IoT Core topic using the MQTT protocol. AWS IoT Core routes the data to another Lambda function for processing, where it is parsed and formatted. The processed data is then stored in Amazon DynamoDB for persistence. A rule in AWS IoT Core ensures seamless data flow between services. Users can query the stored data via a simple interface (e.g., a web app) by invoking another Lambda function through API Gateway, which retrieves data from DynamoDB. This serverless architecture ensures scalability and cost-efficiency.

Explanation of AWS Services Interaction: Amazon API Gateway initiates the workflow by securely calling the weather API and passing the response to the first Lambda function. This function formats the data and publishes it to an AWS IoT Core topic. AWS IoT Core, acting as a message broker, uses its rules engine to forward the data to a second Lambda function. This function processes the data (e.g., extracting temperature and humidity) and writes it to DynamoDB. DynamoDB stores the data in a table with attributes like location, timestamp,

and weather metrics. For data retrieval, API Gateway exposes an endpoint that triggers a third Lambda function to query DynamoDB and return the results to the user. The interaction between these services ensures a seamless, real-time data pipeline with minimal latency and high reliability.

Justification for AWS Service Selection: AWS IoT Core was chosen for its MQTT support and scalability in handling real-time data. Lambda ensures serverless processing, reducing costs. API Gateway provides secure API management, and DynamoDB offers fast, scalable storage. These services collectively enable a cost-effective, low-maintenance solution for real-time weather monitoring.

Implementation Steps

AWS Infrastructure Setup: First, set up an Amazon DynamoDB table named "WeatherData" with a partition key (e.g., "location") and a sort key (e.g., "timestamp") to store weather metrics like temperature and humidity. Next, configure AWS IoT Core by creating a thing and a topic (e.g., "weather/data"). Define an IoT rule to route messages from this topic to a Lambda function. Create an Amazon API Gateway with a REST API endpoint to fetch data from the weather API (e.g., OpenWeatherMap). Link this endpoint to the first Lambda function, which will handle API requests. Set up three Lambda functions: one to fetch and publish data to IoT Core, another to process data and write to DynamoDB, and a third to retrieve data from DynamoDB for user queries. Ensure all services are in the same AWS region for low latency.

Security Policies, IAM Roles, and Access Controls: Create IAM roles for each Lambda function with least-privilege access. The first Lambda role should have permissions to invoke API Gateway and publish to AWS IoT Core (e.g., iot:Publish). The second Lambda role needs permissions to write to DynamoDB (e.g., dynamodb:PutItem) and subscribe to IoT topics (e.g., iot:Subscribe). The third Lambda role requires read access to DynamoDB (e.g., dynamodb:Query). Attach an IoT policy to allow AWS IoT Core to invoke the second Lambda function. For API Gateway, enable AWS IAM authentication to secure the endpoint and restrict access to authorized users. Use AWS Secrets Manager to store the weather API key securely, ensuring the Lambda function retrieves it at runtime. Encrypt DynamoDB data at rest using AWS-managed keys to protect sensitive weather data, ensuring compliance with security best practices.

Automation and CI/CD Pipeline: To automate deployment, use AWS CloudFormation to define the infrastructure as code, including DynamoDB, Lambda functions, API Gateway, and IoT Core resources. Store the CloudFormation template in a Git repository. Set up a CI/CD pipeline using AWS CodePipeline and CodeBuild. CodePipeline will monitor the Git repository for changes, trigger CodeBuild to validate the template, and deploy the stack to AWS. This ensures consistent and error-free deployments. Lambda functions can be updated automatically via the pipeline by packaging the code and deploying it to the respective functions. This automation reduces manual effort and ensures the system remains up-to-date.