CSCI 5409 - Advance topics in Cloud Computing

Submitted by: Giri Sharan Reddy Pusuluru(B00913674)

Project Name: Weather Alert Application - Ensuring Safety and Preparedness

Application Idea

In today's fast-paced and unpredictable world, staying informed about weather conditions has become increasingly vital for individuals to plan their daily activities, safeguard their well-being, and mitigate potential risks. Our innovative weather alert application provides an indispensable solution to this pressing need by offering users personalized and hyper-local weather updates based on their specified location. By subscribing to our application with their email accounts and sharing their location, users receive daily morning weather details tailored specifically to their area. This unique feature ensures that users are well-prepared for any adverse weather conditions that may arise throughout the day, such as heavy rainfall, snowstorms, extreme heat, or sudden temperature drops. With the knowledge of the weather conditions in their immediate location, users can proactively adapt their schedules, attire, and travel routes accordingly, significantly reducing the likelihood of weather-related inconveniences or hazards.

List of Services used

- Compute – AWS Lambda, Amazon EC2
- Storage – Amazon DynamoDB
- Network – Amazon API Gateway
- General – Amazon SQS, Amazon SNS, Amazon EventBridge

Importance of Each service that is used by me:

1. EC2 to host my ReactJS application: In the project, I chose to utilize an EC2 Linux server for deploying the frontend application over Amazon Elastic Container Service, as EC2 gives precise control over their infrastructure and top-notch performance because it is very flexible. Furthermore, EC2 outperforms other services in terms of computation power, and it offers a wide selection of instance types to choose from.
2. Lambda: To write/fetch data to/from DynamoDB, I used Lambda function over EC2. My application mainly runs only for a specific amount of time and that too daily once so by using FaaS, I can save huge application costs over long term. Lambda also scales the code based on the requests coming to the function so it is good in terms of scalability as well as over any other AWS services
3. Amazon DynamoDB: For this application I need to store user email id, location of his choice. For that I chose Dynamo DB over Amazon RDS. DynamoDB is known for high-

performance and has less latency and response time when reading and writing data. No matter the size of the tables we have, the latency of DynamoDB is minimum, and it maintains within milliseconds.

4. In order for React application to send data we need an entry point that is created using API Gateway which invokes lambda function that store user email id and location in DynamoDB. API Gateway is the only service on aws cloud with the help of which we can create an API and deploy it. I have created Rest API with the help of the API Gateway that is pointed to my lambda function logic.
5. Amazon Simple Queue Service (SQS) is a highly reliable and fault-tolerant messaging system. It stores messages until they are processed successfully or intentionally deleted, ensuring data integrity and minimizing the risk of data loss during temporary failures or system crashes. Additionally, SQS easily integrates with other AWS services, making it effortless to build distributed systems with various AWS tools. Where one of my lambda functions get invoked when a message comes into Queue. SQS allows to process messages asynchronously. Lambda retrieves and delete the message once it is used from the queue.
6. Every morning at 12:01am a lambda is triggered which retrieve details from DynamoDB and then get weather details of that particular location using google api and visual crossing api. Parse the received data and fetch required data like weather alerts, sunrise and sunset timings etc. For triggering the Lambda, Rule in Event bridge is used.
7. As my application need to send weather information of the user specified location every day at 12:01am, I used SNS to send weather notification to user given email id.

Deployment Model

In this project I decided to implement a public cloud deployment model and selected AWS as the cloud provider for its outstanding industry reputation, having an incredibly high uptime of 99.9999999% and being cost-effective. It offers the advantage of low upfront costs and a "pay-as-you-go" pricing model, which means we only pay for the resources we actually use. Since we don't handle personal information in the application, we have no data security concerns. The combination of a public cloud model and AWS's capabilities provides us with a powerful, scalable, and budget-friendly solution, giving my project a solid foundation to succeed and grow effectively in the competitive digital world.

Delivery Model

I have integrated various delivery models to add multiple features to the application, ensuring cost-effectiveness in my decision-making. For the application's backend, I have adopted a serverless approach, utilizing Lambda functions to interact with the database and Event Bridge to trigger a lambda function for weather information retrieval. Leveraging Functions as a Service (FAAS) proves highly advantageous as AWS manages these services entirely, requiring

only the code from me to execute. The automatic scaling and pay-as-you-go model result in significant cost savings compared to other resources. Additionally, the seamless integration with other AWS services enhances usability, enabling a wide range of operations to be performed efficiently.

To store user email IDs and weather preferred location, I opted for DynamoDB, a versatile solution that can be considered both Platform as a Service (PAAS) and Database as a Service (DAAS). For handling incoming requests, I utilized API Gateway, another PAAS option. Employing PAAS for these purposes offers several advantages, including auto-scaling capabilities and the absence of infrastructure setup burdens. Amazon EC2 for running my front-end react application, where it is Infrastructure as a Service (IaaS). I have utilized CloudFormation script to set up the entire infrastructure, adhering to the principles of Infrastructure as Code (IaC).

Architecture

The architecture follows the Microservice System approach, where each component has a specific role and doesn't rely heavily on other services. All the units work independently and are loosely connected. They only take input when needed, process it, and pass it on to other units as required, without knowing too much about each other's functions. Their main task is to handle their job efficiently and, if necessary, call other units for additional processing.

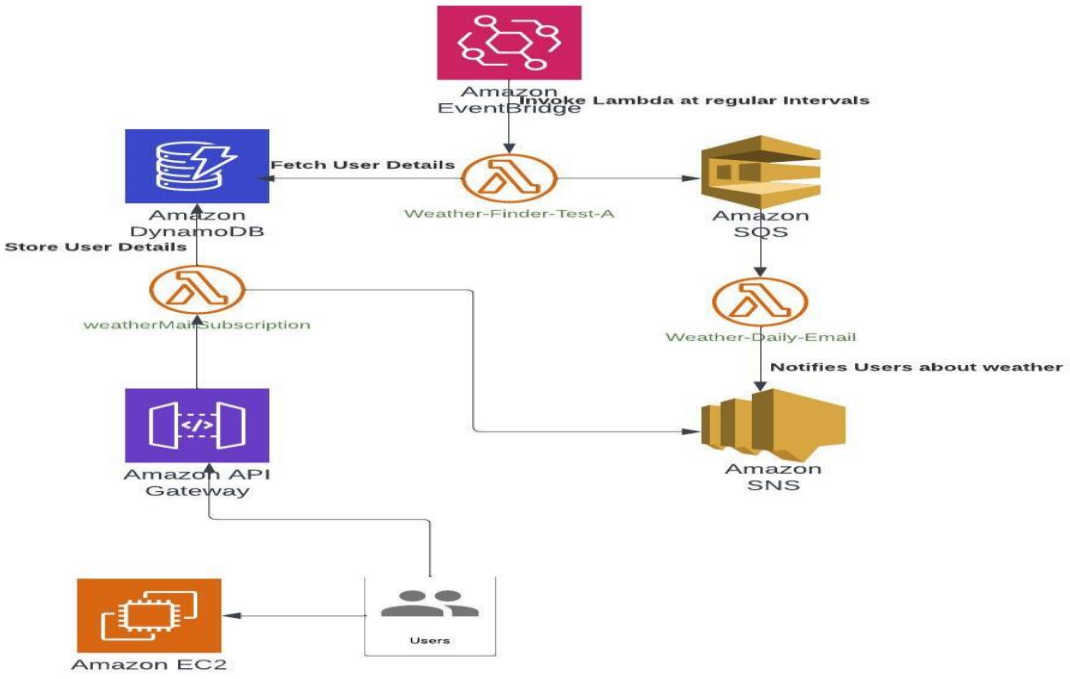

Figure 1: Application Architecture

Data is stored in DynamoDB.

I have total 3 lambda functions where 2 of them are written in Python Programming Language and another lambda function in Java Programming Language as it required to call external apis to get the weather information. In Java we can add required dependencies to call the apis easily.

First, I used CloudFormation script to deploy my backend code, once the backend is deployed, the API Gateway endpoint URL is generated. I then use these URLs in my frontend logic to interact with the backend services. Finally, I deploy the frontend using CloudFormation script, making it accessible to users and enabling seamless communication with the backend through the API Gateway endpoints. This ensures a smooth end-to-end deployment of the entire application, from the backend to the frontend.

Organization expenditure for on-premise infrastructure

A strong and scalable hardware foundation is crucial before deploying any components and services. This infrastructure supports hundreds to thousands of users simultaneously, providing the necessary computing power, storage, and networking capabilities. It forms the backbone of the system, ensuring smooth operations even during high user activity and accommodating the growing user base effectively. The costs can be divided mainly into two types

a. Up Front Costs – Buying huge cost servers, Network equipment, Skilled persons to install and run servers, new software cost.
b. Ongoing Costs – Like support and maintenance of servers, Network equipment, renewing software licenses, salaries for developers, support staff, changing or upgrading equipment on regular basis.

Security

As per fig. 1 application flow is as below

- First user requests for web-page which will be responded by returning single page application
- To securely send data, the webpage makes a request to the API Gateway using HTTPS, which ensures protection against eavesdropping on public networks. However, there is a potential flaw in this flow: users can make continuous calls to the API Gateway, which could overload the server and exhaust its resources. This makes the system vulnerable to Denial of Service (DoS) and Distributed Denial of Service (DDoS) attacks. To address this vulnerability and ensure the application's resilience against such attacks, security providers like CloudShield or third-party services like Cloudflare can be implemented.

- Two Lambda functions reach the Database level for performing read and write operations in DynamoDB
- To keep the system safe from attacks, we don't let the front-end directly access the database. Instead, we use special functions (Lambda) to handle data access. These functions act as guards and ensure only authorized requests reach the database. The steps in between also make sure only the right data gets stored. This approach boosts security and protects against unauthorized access or data changes, giving users a safe and reliable experience.
- The only point of contact of whole backend system to non-developers is the api gateway making it a bit more secure as anything beyond api gateway cannot be accessed

Most expensive service in current application

- AWS Lambda costs $0.20 per million requests
- DynamoDb costs $1.25 per 1m write request & $0.25 per million read request
- API Gateway costs $3.50 per 333 million

So, in my case performing operations on DynamoDB costs more. In order to monitor this, we need to make use of Amazon Cloudwatch service to make sure my budget do not exceed. Moreover, we can consider archiving or deleting old data that is no longer needed.

Future Improvements

Currently, the application provides weather details for a single location and a specific day. But in future, it will allow users to check the weather for multiple locations they are interested in. Also, for travelers, if they provide their travel locations and date within the next 15 days, the application will send them weather alerts and details for their journey route. These updates aim to make the application more useful and personalized, helping users better plan for their activities and trips based on the weather forecast. These improvements will enhance the overall user experience, making the application even more reliable and valuable for weather information and planning.

Mail I will be receiving with information about weather and any alerts (if available) is like

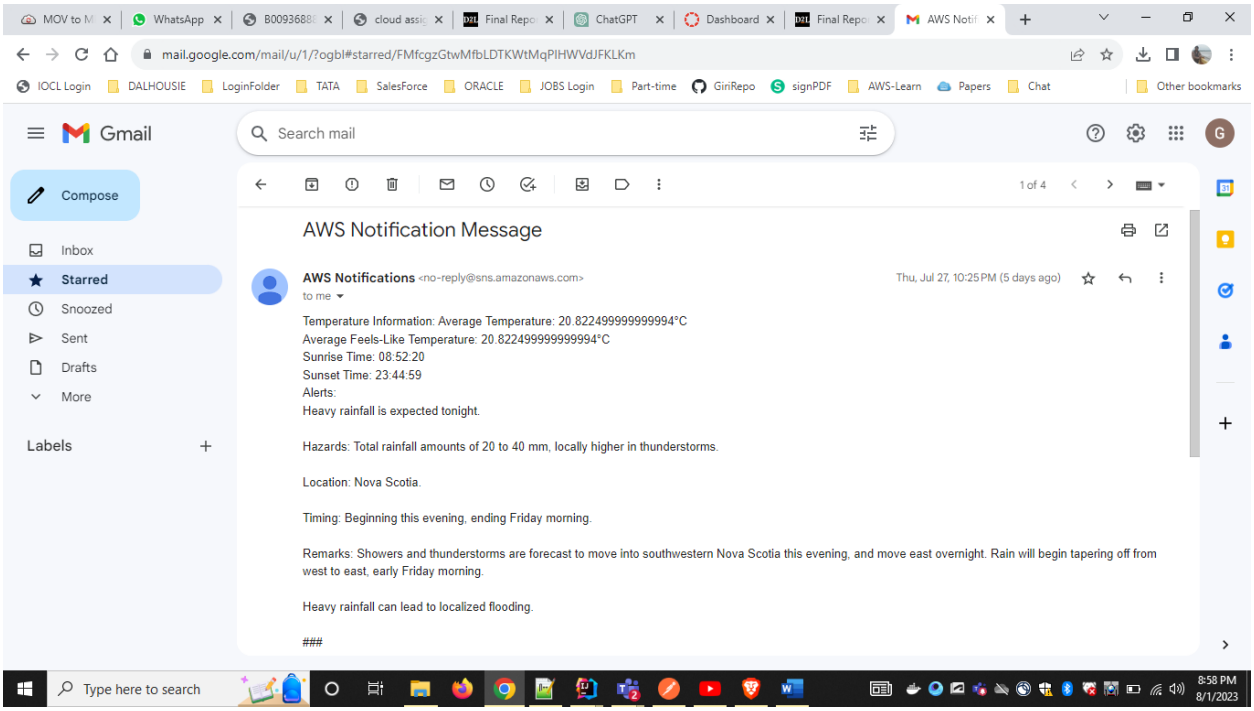

Figure 2: Mail with weather alert