**WeatherSnap**

**Abstract**

WeatherSnap offers a streamlined and user-friendly approach for distributing meteorological updates, harnessing the power of Amazon Web Services' Simple Notification Service (SNS). This service stands as a testament to the potential of Pub-Sub notification mechanisms, functioning as a practical demonstration of their utility. Designed to seamlessly integrate into users' daily routines, WeatherSnap is programmed to dispatch succinct weather alerts directly to subscribers' emails or phones at two pivotal times of the day: 7 AM and 4 PM, aligned with Milwaukee's local time. The primary objective of WeatherSnap is to ensure that individuals are well-informed about the day's weather conditions, thereby enhancing their ability to make timely and informed decisions related to weather events.

**Architecture**

**A screenshot of a computer

Description automatically generated**

1. **Amazon SNS to Subscribers:**

The architecture begins with Amazon Simple Notification Service (SNS), a managed service that provides message delivery or sending of notifications to subscribing endpoints or clients. In this flow, Amazon SNS is responsible for directly sending notifications to the system's subscribers. This could be set up as a topic in SNS to which users subscribe to receive updates.

1. **Subscribers to Amazon EC2 Instance:**

The information flow from subscribers to an Amazon EC2 instance suggests that the subscribers' actions or responses might be monitored or logged by the system. This could be like registering the subscribers.

1. **Amazon EC2 Instance to Amazon Lightsail:**

The Amazon EC2 instance interacts with Amazon Lightsail, which in this context is likely used for its database capabilities. Currently we are using the lightsail database to register the subscribers.

1. **Amazon EC2 Instance to Amazon SNS:**

This flow indicates that the EC2 instance has the capability to communicate with Amazon SNS, to publish messages to the SNS topics based on the processed data or to trigger notifications as a result of certain conditions being met within the application logic.

1. **OpenWeather API to Lambda Function to Amazon SNS:**

The OpenWeather API is used to fetch real-time weather data. A Lambda function is triggered, possibly on a scheduled basis or in response to certain events. This serverless function processes the weather data and then publishes it to Amazon SNS.

The use of AWS Lambda suggests that the operation of fetching and processing data from OpenWeather is managed without provisioning or managing servers, allowing for a scalable and cost-effective solution.

In summary, the architecture describes a serverless, event-driven system where weather updates are provided by the OpenWeather API, processed by AWS Lambda, and then sent to Amazon SNS. The EC2 instance seems to serve additional roles such as further data processing, interaction with a database, and possibly managing subscription and notification logic. The subscribers receive notifications directly through SNS, and their interactions may feed back into the EC2 instance for logging or analytical purposes. Amazon Lightsail's database is used to store necessary data which supports the operations carried out by the EC2 instance.

**Features**

* **Real-Time Weather Alerts:**

**Feature:** Subscribers receive real-time weather alerts based on data from OpenWeather.

**Use:** Users can set up notifications for specific times or weather events, staying informed about weather conditions that could affect their daily activities.

* **Scheduled Notifications:**

**Feature:** The system is capable of sending scheduled weather alerts, as it is connected to Amazon SNS which can be configured to deliver messages at specific times.

**Use:** Subscribers can receive weather updates at predetermined times, such as early morning and late afternoon, to plan their day or commute accordingly.

* **Serverless Data Processing:**

**Feature:** AWS Lambda is utilized for processing the weather data fetched from the OpenWeather API.

**Use:** This serverless approach ensures that the system is scalable and can handle varying loads without manual intervention for server management.

* **Persistent Data Management:**

**Feature:** Integration with Amazon Lightsail for database services.

**Use:** Stores subscription information, user preferences, and historical weather data, enabling a personalized and reliable service.

* **Automated Message Publishing:**

**Feature:** The Amazon EC2 instance can automatically publish messages to the SNS topic.

**Use:** This can be used to trigger notifications based on specific data points or thresholds identified by the application logic.

* **Scalability and Reliability:**

**Feature:** Utilization of AWS services like EC2 and SNS ensures that the system is both scalable and reliable.

**Use:** The system can scale up to handle a large number of subscribers and is reliable enough to ensure consistent delivery of notifications.

* **User Subscription Management:**

**Feature:** The system likely includes a mechanism for users to subscribe and manage their notification preferences.

**Use:** Users can easily sign up for the service and customize their alert settings according to their personal needs.

**How it Works**

* **Weather Data Retrieval:**

The system starts by fetching the latest weather data from the OpenWeather API, a third-party service that provides real-time weather information.

An AWS Lambda function is scheduled to invoke at regular intervals or in response to specific triggers. When this function runs, it calls the OpenWeather API to retrieve current weather data.

* **Data Processing:**

Once the AWS Lambda function receives the weather data, it processes this information. This processing might involve converting raw data into a more user-friendly format, checking for weather conditions that meet certain alert criteria, or customizing the data based on user preferences.

* **Notification Preparation:**

After processing the data, the Lambda function interacts with Amazon SNS to prepare the weather alerts.

The function publishes the processed weather information to a topic in Amazon SNS. A topic is a communication channel to which users can subscribe to receive notifications.

* **User Subscription and Alert Delivery:**

Subscribers have previously signed up to receive notifications and are subscribed to the relevant SNS topic.

Amazon SNS then takes the published message and sends it out to all subscribed users through the configured channel i.e. mail.

* **User Interaction and Feedback:**

When users receive the notification, any interaction they have with the message (such as opening an email or clicking a link) can be tracked and sent back to an Amazon EC2 instance.

The EC2 instance can log this interaction data for analytics and use it to improve the service.

* **Data Storage and Management:**

An Amazon Lightsail database is connected to the EC2 instance, providing persistent storage for the system.

This database stores subscription details, user preferences, historical weather data, and possibly logs of user interactions.

* **Ongoing Operation and Maintenance:**

The Amazon EC2 instance also serves as a central control unit for the system, overseeing operations, managing the database, and ensuring the smooth execution of scheduled tasks.

The system is designed to run automatically, with the EC2 instance and Lambda functions operating based on the schedule or triggers without manual input.

Through this sequence of steps, the weather notification system automates the process of delivering weather alerts currently we have hardcoded the location to be “Milwaukee”, ensuring that users receive relevant and timely information which we have set to receive the notification in the morning at 7:00 PM and in the evening at 4:00 PM with minimal latency. The use of AWS services ensures that the system is scalable to accommodate a growing number of users and resilient enough to provide reliable notifications around the clock.

Try it Yourself! COMPSCI 790, Cloud Computing, WeatherSnap Application.

You’ll be creating a Lambda Function that calls an OpenWeatherMaps API to gather weather data. Youll then be sending that data to subscribers of an SNS Queue. Youll also be creating a database and an application that will store this data. For the database, we will be using Lightsail SQL Database and will be creating a CLI Application that simply takes the register command and asks for the users email id. This registration application will be deployed on an EC2 Instance.

Go to OpenWeatherMaps and register for an API Key

Make an SNS Queue in AWS that will have the user emails

Navigate to SNS in the AWS Console and create a standard topic, give it a name and keep the default settings. Note the topic ARN number, we will use this to connect the lambda function and the ec2 to the SNS topic.

Create a lightsail database

Head over to Lightsail in the AWS Console and go to databases and create a sql 8.0 database. Keep the default settings but create a user and password. Connect to the database using mysql workbench via the database hostname. Run a simple query to create a table that stores emails of users.

Run the following Query:

CREATE DATABASE Email;

USE Email;

CREATE TABLE IF NOT EXISTS EmailSubs(

id INT AUTO\_INCREMENT PRIMARY KEY,

Email VARCHAR(255) NOT NULL );

Create a Lambda Function.

Head over to Lambda in the AWS console and create a function, Use python 3.10 as the runtime environment. Keep everything else the same.

We need to deploy the following code as both dependencies and as code which requires some special work in order to work with a specific requests library.

For this we will be provisioning an ec2 server to create a lambda deployment package.

To arrive at a Lambda layer that works  we built the dependency layer on Linux. This is an outline of the process used.

1. Create a small EC2 instance running Ubuntu 20.04 LTS or locate an instance you already have   
running. This instance will require access to the public internet in order to install software.   
2. Run the following command sequence. We did this starting from our home folder (~):   
sudo apt update   
sudo apt install pip3-python zip   
mkdir layer   
cd layer   
mkdir python   
cd python   
pip install mysql-connector-python –target .   
pip install Pillow –target .   
pip install requests –target .   
cd ..   
zip -r DepsLayer.zip python/   
3. Download the DepsLayer.zip file to your client computer and then deploy it to your Lambda layer.

4. deploy the following code in the AWS Lambda function from the AWS Console:

**NOTE: change all the keys, api keys, city details, arn topic, aws-region etc. to your own keys and preferences. I would recommend using environment variables to add some security rather than hard coding these values into the code.**

import requests

import boto3

import json

import os

from datetime import datetime

import uuid

def lambda\_handler(event, context):

    # AWS credentials and SNS topic details

    aws\_access\_key = 'Your access key'

    aws\_secret\_key =  'Your Secret Key'

    language = 'en'

    sns\_topic\_arn = 'Your SNS Topic ARN’

    # OpenWeatherMap API details

    api\_key = 'Your API Key to OpenweatherMap’

    city = 'The city you want'

    sns\_client = boto3.client(

        'sns',

        aws\_access\_key\_id=aws\_access\_key,

        aws\_secret\_access\_key=aws\_secret\_key,

        region\_name='The region you are using'

    )

    def get\_weather():

        base\_url = f'http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}&units=metric&lang={language}'

        response = requests.get(base\_url)

        if response.status\_code == 200:

            data = response.json()

            weather = {

                'city': data['name'],

                'temperature': data['main']['temp'],

                'description': data['weather'][0]['description']

            }

            return weather

        else:

            return None

    def send\_weather\_update():

        weather\_info = get\_weather()

        if weather\_info:

            message = f"Weather in {weather\_info['city']} - Temperature: {weather\_info['temperature']}°C, Description: {weather\_info['description']}"

            sns\_client.publish(

                TopicArn=sns\_topic\_arn,

                Message=message,

                Subject='Weather Update'

            )

            print("Weather update sent successfully to subscribers!")

        else:

            #print(f"Failed to fetch weather information. Status code: {response.status\_code}")

            #print(f"Response content: {response.text}")

            #return {"error": "Failed to fetch weather information"}

            print("Failed to fetch weather information")

    send\_weather\_update()

5. Create trigger that you deem fit for triggering this sns queue, we used 2 time triggers that run at 7am and at 4 pm. These are crontab triggers that run on eventbridge cloudwatch. Click on add trigger and add two triggers that use the following or any of your choosing:

Add Trigger > EventBridge(CloudWatch Events) > Create New Rule > name > **cron(0 13 \* \* ? \*)**

Add Trigger > EventBridge(CloudWatch Events) > Create New Rule > name > **cron(0 22 \* \* ? \*)**

Now create an EC2 instance to register users.

Launch an EC2 instance that uses python 3.10 and runs on ubuntu 20.04

Connect to the instance and run the following commands

Sudo apt update

Sudo apt upgrade

sudo apt update   
sudo apt install pip3-python zip   
pip install mysql-connector-python –target .   
pip install requests –target .

Mkdir sns

Cd sns

Nano app.py

Paste the following code into app.py

import mysql.connector

import boto3

# AWS credentials and region configuration

aws\_access\_key\_id = 'YOUR\_AWS\_ACCESS\_KEY\_ID'

aws\_secret\_access\_key = 'YOUR\_AWS\_SECRET\_ACCESS\_KEY'

aws\_region = 'YOUR\_AWS\_REGION'

# AWS SNS setup

sns\_client = boto3.client('sns', aws\_access\_key\_id=aws\_access\_key\_id,

                          aws\_secret\_access\_key=aws\_secret\_access\_key,

                          region\_name=aws\_region)

sns\_topic\_arn = 'YOUR\_SNS\_TOPIC\_ARN'

# Lightsail MySQL Database configuration

db\_config = {

    'user': 'YOUR\_DB\_USER',

    'password': 'YOUR\_DB\_PASSWORD',

    'host': 'YOUR\_DB\_HOST',

    'database': 'YOUR\_DB\_NAME',

}

def register\_email(email):

    if email:

        try:

            # Save to Lightsail MySQL database

            db\_connection = mysql.connector.connect(\*\*db\_config)

            cursor = db\_connection.cursor()

            # Insert email into the database

            insert\_query = "INSERT INTO email (email) VALUES (%s)"

            cursor.execute(insert\_query, (email,))

            db\_connection.commit()

            # Subscribe email to AWS SNS topic

            sns\_client.subscribe(

                TopicArn=sns\_topic\_arn,

                Protocol='email',

                Endpoint=email

            )

            cursor.close()

            db\_connection.close()

            return f"Email '{email}' registered successfully and subscribed to the SNS topic!"

        except Exception as e:

            return f"Error: {e}"

    else:

        return "Email not provided."

if \_\_name\_\_ == "\_\_main\_\_":

    email\_to\_register = input("Enter email to register: ")

    result = register\_email(email\_to\_register)

    print(result)

Now to register a user

Get into the sns folder on the ec2 instance

Enter the command

Python3 app.py

Enter an email

If all steps were completed properly, it should give you a successfully registered email message and you can confirm this by checking if the email was added to both the database and sns queue via mysql workbench and the aws sns console respectively.