VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

AMAZON WEB SERVICES

Project

Name: Abhinit Mishra

Registration Number:22BIT0658

Course: AWS Solutions Architect

Course Code: BCSE355L

Faculty: Dr. ARUNKUMAR A

SLOT: G2+TG2

Topic: Mass Emailing Using AWS Lambda

Introduction

The objective of this project is to implement a **serverless mass emailing solution** using **AWS Lambda**. AWS Lambda, along with services such as **Amazon SES** (Simple Email Service) and **Amazon S3**, offers a cost-effective, scalable, and manageable solution for sending bulk emails. This project explores how to create, deploy, and manage a Lambda function that sends emails to a list of recipients using **Amazon SES**.

Mass emailing is commonly used for **marketing campaigns**, **newsletters**, and **notifications** to large audiences. Building a serverless architecture on AWS ensures scalability, cost-efficiency, and flexibility, as it charges you only for what you use.

Key AWS Services Used

1. **AWS Lambda**: To run the serverless code for sending emails.

2. **Amazon SES (Simple Email Service)**: To send emails.
3. **Amazon S3**: To store email content and recipient lists (optional).
4. **Amazon CloudWatch**: For logging and monitoring.
5. **AWS IAM (Identity and Access Management)**: To control access to resources.

Project Requirements

- **AWS Account**: To access the necessary services such as Lambda, SES, and S3.
- **Node.js** or **Python**: As the primary runtime for Lambda (Python is used in this project).
- **Amazon SES Domain Setup**: Configure Amazon SES to use your domain or email for sending emails.
- **IAM Role**: Lambda needs access to SES for sending emails and, optionally, to S3 to read recipient lists.

Design and Architecture

The system is designed to be fully serverless, ensuring scalability and low operational costs. The architecture involves:

- **Lambda Function**: Executes the code for sending mass emails.
- **Amazon SES**: Handles email delivery.
- **S3 (optional)**: Stores recipient lists and email templates.
- **CloudWatch**: Logs all function invocations and any errors or warnings.

Step-by-Step Implementation

1. Setting Up Amazon SES

Before configuring the Lambda function, you need to set up **Amazon SES**:

- **Verify Email/Domain**:
  - Go to the **SES Management Console**.
  - Verify your email address or domain that will be used to send emails.

- Once verified, you can start sending emails from the domain or email.

- **Request Production Access (Optional)**:
  - By default, SES operates in **Sandbox Mode**, which limits the number of emails you can send.
  - If you're using SES for mass emailing, request production access to remove these limitations.

2. Creating the Lambda Function

Go to **AWS Lambda** and create a new function:

1. **Open Lambda Console** → **Create Function**.
2. Choose **Author from Scratch**.
3. **Function Name**: massEmailSender.
4. **Runtime**: Choose Python 3.x or Node.js (in this example, we'll use Python).
5. **Permissions**: Create a new role with basic Lambda permissions, including SES access.

3. Lambda Code for Mass Emailing

Here is a Python Lambda function that reads a list of recipients and sends an email to each using Amazon SES.

PYTHON CODE:

```
import json
import boto3
import os

# Initialize AWS SES client
ses_client = boto3.client('ses',
region_name='us-east-1')

# Lambda handler function
def lambda_handler(event, context):
    # List of recipients (can be retrieved from S3
or event)
    recipients = ['recipient1@example.com',
'recipient2@example.com']

    # Email content
    subject = 'Mass Email Notification'
```

```
    body_text = 'Hello, this is a test email using
AWS Lambda and SES.'

    # Email sender
    sender = 'your-verified-email@example.com'

    # Iterate over the recipient list
    for recipient in recipients:
        # Send email via SES
        try:
            response = ses_client.send_email(
                Source=sender,
                Destination={
                    'ToAddresses': [recipient]
                },
                Message={
                    'Subject': {
                        'Data': subject
                    },
                    'Body': {
                        'Text': {
                            'Data': body_text
```

```
                }
            }
        }
    )
        print(f"Email sent to {recipient}")
    except Exception as e:
        print(f"Error sending email to {recipient}:
{str(e)}")

    return {
        'statusCode': 200,
        'body': json.dumps('Emails sent
successfully!')
    }
```

4. Upload the Lambda Code

1. **Go to Lambda Console → Upload the Lambda Function Code**.
2. Save the function.
3. Set the **timeout** (e.g., 30 seconds) and **memory size** (e.g., 128 MB) based on your requirements.

5. Configuring IAM Role for SES

- Make sure the Lambda function has the required permissions to access Amazon SES.
- In the IAM console, attach a **policy** to your Lambda execution role:

4. Upload the Lambda Code

1. Go to Lambda Console → Upload the Lambda Function Code.
2. Save the function.
3. Set the **timeout** (e.g., 30 seconds) and **memory size** (e.g., 128 MB) based on your requirements.

5. Configuring IAM Role for SES

- Make sure the Lambda function has the required permissions to access Amazon SES.
- In the IAM console, attach a **policy** to your Lambda execution role:

JSON CODE:

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Action": "ses:SendEmail",
         "Resource": "*"
      }
   ]
}
```

This policy allows your Lambda function to send emails using Amazon SES.

6. Testing the Lambda Function

- **Test Locally or in the Lambda Console**: You can manually invoke the function from the AWS Lambda console or using an event to simulate email sending.

JSON CODE:

```
{
   "recipients": ["recipient1@example.com",
"recipient2@example.com"],
   "subject": "Important Notice",
   "body_text": "Please read this email
carefully."
}
```

7. Monitoring and Logging

- Use **Amazon CloudWatch Logs** to monitor your Lambda function's performance, track successful email deliveries, and debug errors.
- View invocation metrics such as **duration**, **error count**, and **throttling** in **CloudWatch Metrics**.

8. Optional: Store Email Templates and Recipients in S3

- You can store email templates and recipient lists in Amazon S3 and retrieve them in your Lambda function. This provides flexibility in managing email content and recipient data.

PYTHON CODE:

```
s3 = boto3.client('s3')

# Get recipient list from S3
def get_recipient_list(bucket_name, key):
    response =
s3.get_object(Bucket=bucket_name, Key=key)
    recipient_list =
response['Body'].read().decode('utf-
8').splitlines()
    return recipient_list
```

Deploying the Solution

1. **Create a Trigger (Optional)**:

- You can trigger the Lambda function using **Amazon API Gateway** (if you need an API), or **S3** (for file upload triggers), or **EventBridge** for scheduled mass emails.

2. **Publish Lambda to Production**:

- Once tested, you can deploy the Lambda function into production and monitor its performance using CloudWatch.

Challenges Faced

1. SES Limits: SES has limits in sandbox mode; to send more emails or reach a wider audience, you need to request **production access**.

2.Error Handling: In the case of invalid email addresses or other failures, it's important to implement robust error handling and logging in the Lambda function.

3.Cold Start Issue: Serverless functions like Lambda may experience a cold start delay, especially if not frequently invoked.

Future Work

- **Personalization**: The system can be extended to personalize emails for each recipient by including dynamic content in the email body.
- **Bounced Email Handling**: Implement logic to handle bounced emails using **SNS (Simple Notification Service)**.

- **Scheduled Emails**: Utilize **Amazon EventBridge** to schedule mass email campaigns.
- **Reporting and Analytics**: Implement email open-rate tracking and click-through analysis.

Conclusion

This project demonstrated how to set up a **mass emailing system using AWS Lambda and SES**. It was a serverless, scalable, and cost-effective solution that can handle the high volume of emails without managing any physical infrastructure. By integrating Amazon SES with AWS Lambda, the system leveraged AWS services to efficiently send bulk emails.

By utilizing AWS Lambda and SES, this project exemplifies the benefits of **serverless computing** in modern email solutions.