Serverless Image Processing Application

Submitted by-Nitya Tiwari

Table of Contents

1. Introduction

2. Setup Checklist for Mini Project

3. Instructions

4. Problem Statement

5. Objective

6. Abstract of the Project

7. Technology Used

8. Implementation

9. Short steps

10. Data Model (Architecture)

1. Introduction

This mini project focuses on developing a serverless image processing application using AWS. The application automatically resizes and compresses images uploaded to an input S3 bucket and stores the optimized versions in an output S3 bucket.

2. Setup Checklist for Mini Project

Hardware:

- Intel i3 processor or higher

- 4 GB RAM or more

- Stable internet connection

Software:

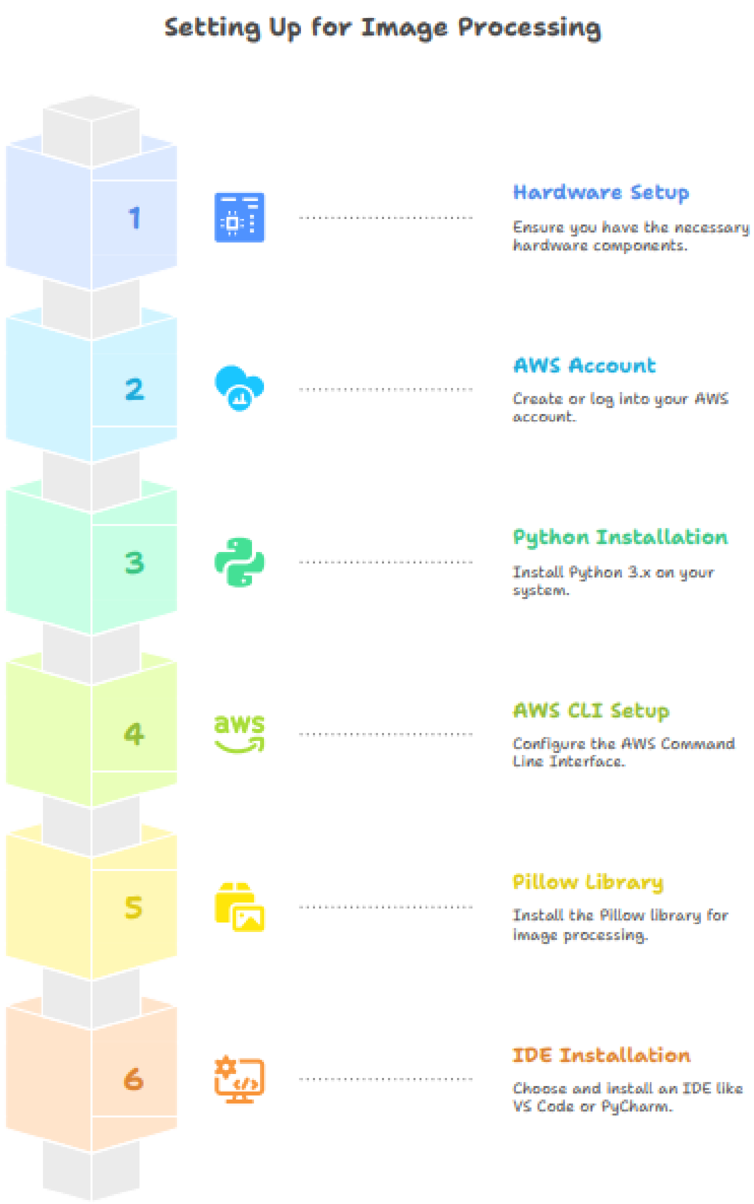
- AWS Account

- Python 3.x

- AWS CLI

- Pillow library

- VS Code or PyCharm



Setting Up for Image Processing

1. Hardware Setup — Ensure you have the necessary hardware components.

2. AWS Account — Create or log into your AWS account.

3. Python Installation — Install Python 3.x on your system.

4. AWS CLI Setup — Configure the AWS Command Line Interface.

5. Pillow Library — Install the Pillow library for image processing.

6. IDE Installation — Choose and install an IDE like VS Code or PyCharm.

3. Instructions

Follow AWS best practices. Keep your IAM permissions safe. Store all code and logs securely.

4. Problem Statement

Create a serverless application that processes images by resizing and compressing them when they are uploaded to an Amazon S3 bucket.

5. Objective

Develop a lightweight, cost-effective image optimizer using AWS Lambda and S3 to automate backend image processing.

6. Abstract of the Project

When a user uploads an image to S3, Lambda gets triggered. It resizes the image to 800x800, compresses it to about 80% quality JPEG, and saves it to a different output S3 bucket.

7. Technology Used

- AWS S3

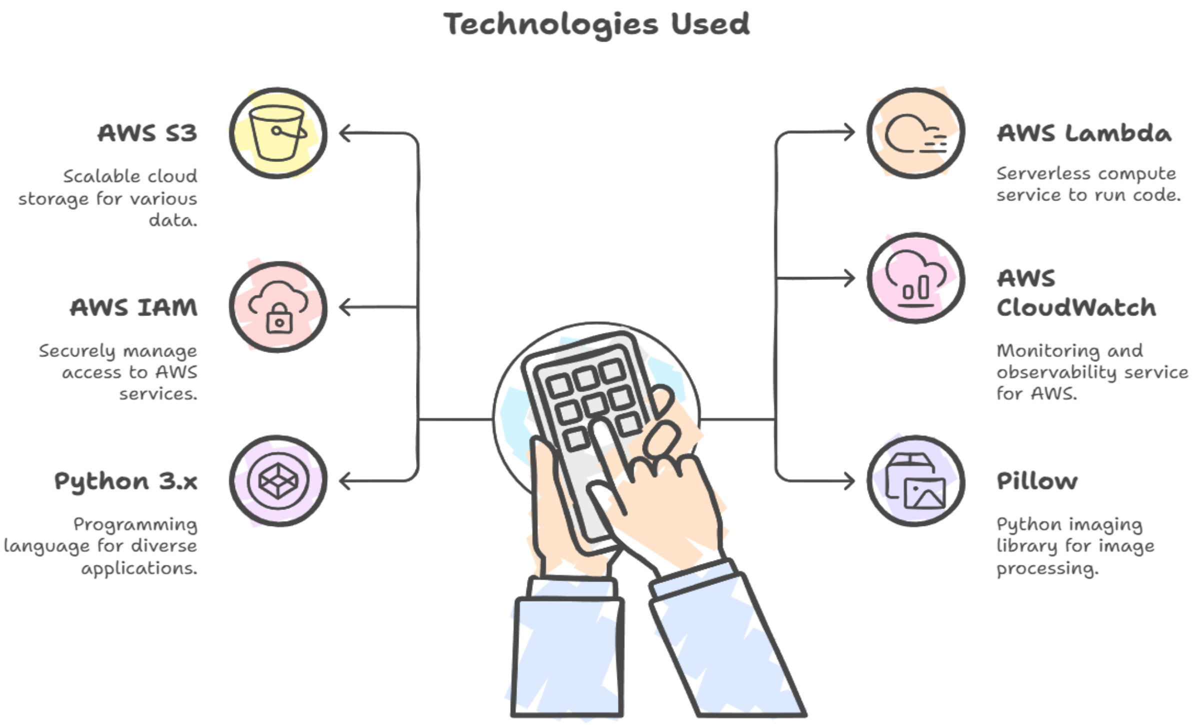
- AWS Lambda

- AWS IAM

- AWS CloudWatch

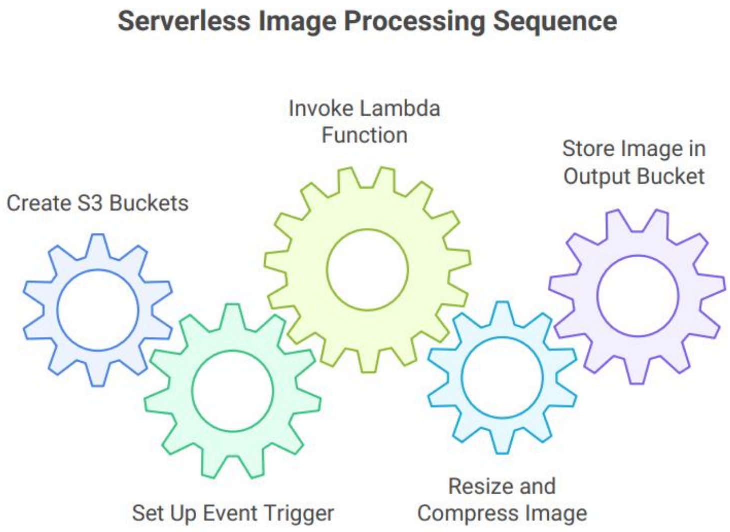
- Python 3.x

- Pillow


Technologies Used
AWS S3
Scalable cloud storage for various data.
AWS IAM
Securely manage access to AWS services.
Python 3.x
Programming language for diverse applications.
AWS Lambda
Serverless compute service to run code.
AWS CloudWatch
Monitoring and observability service for AWS.
Pillow
Python imaging library for image processing.

8. Implementation

Create S3 buckets named 'inputbucket' and 'outputbucket.' Set up an event trigger on inputbucket to invoke a Lambda function. Lambda will resize and compress the image, then store it in 'outputbucket.'


Serverless Image Processing Sequence
Invoke Lambda Function
Store Image in Output Bucket
Create S3 Buckets
Set Up Event Trigger
Resize and Compress Image

9. Short Steps to Deploy the Project on AWS

○

1. **Set up two S3 buckets:**
   - inputbucket (for your original images)
   - outputbucket (for the processed images)

2. Create a Lambda function using Python and the Pillow library to handle image resizing and compression.

3. In the AWS Console, **create a new Lambda function with the Python runtime.**

4. **Upload your code** as a deployment package (lambda.zip).

5. **Add a trigger to your Lambda function:**

- Choose S3 - Bucket: inputbucket - Event type: PUT (for .jpg or .png files)

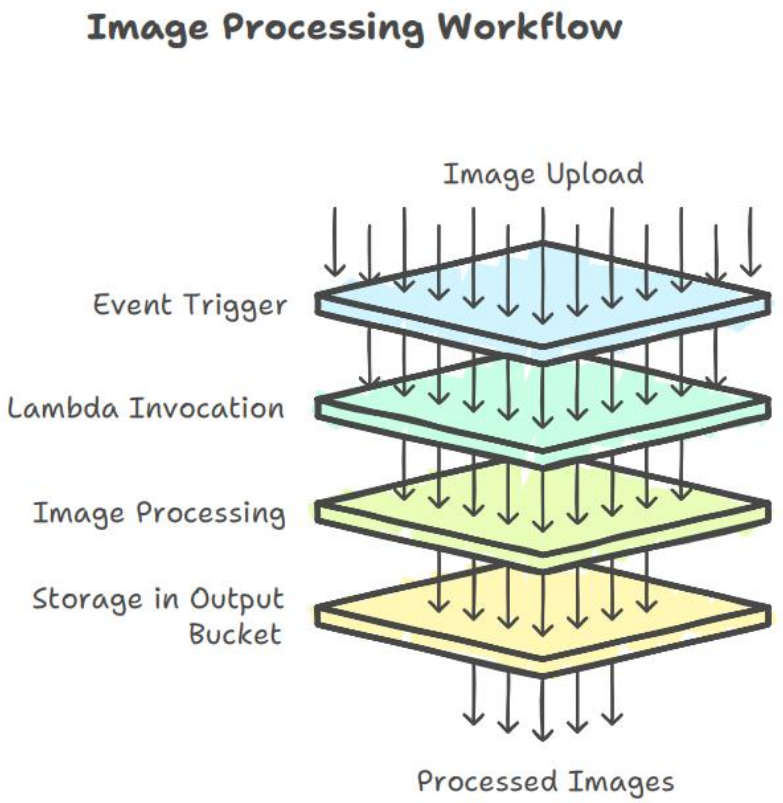
6. **Set up IAM permissions:** - Attach policies for S3 and CloudWatch access to the Lambda role.

7. **Test your application:** - Upload a large image (like a 5MB file) to inputbucket - You should see a resized, optimized image (1–2MB) in outputbucket.

8. **Use AWS CloudWatch to monitor logs** and ensure the function is running smoothly.

10. Data Model (Architecture)

Input Bucket (S3) -> Trigger -> AWS Lambda (Resize, Compress) -> Output Bucket (S3)



Lambda Function Code (lambda_function.py)

```
import boto3
import os
from PIL import Image
import io
s3 = boto3.client('s3')
def lambda_handler(event, context):
    input_bucket = 'inputbucket'
    output_bucket = 'outputbucket'
    # Get uploaded file key from event
    input_key = event['Records'][0]['s3']['object']['key']
    try:
        # Fetch image from input bucket
        response = s3.get_object(Bucket=input_bucket, Key=input_key)
        image_data = response['Body'].read()

        # Open with Pillow
        img = Image.open(io.BytesIO(image_data))

        # Convert to RGB if needed (JPEG safe)
        if img.mode not in ("RGB", "L"):
            img = img.convert("RGB")

        # Resize image, keeping aspect ratio
        img.thumbnail((800, 800))
        # Save to buffer
        buffer = io.BytesIO()
        img.save(buffer, format='JPEG', optimize=True, quality=80)

        buffer.seek(0)

        # Output filename (.jpg version)

        output_key = os.path.splitext(input_key)[0] + '.jpg'

        # Upload to output bucket

        s3.put_object(

            Bucket=output_bucket,

            Key=output_key,

            Body=buffer,

            ContentType='image/jpeg'

        )

        return {

            'statusCode': 200,

            'body': f'Success: Image saved to {output_bucket}/{output_key}'

        }

    except Exception as e:

        return {

            'statusCode': 500,

            'body': f'Error: {str(e)}'

}
```

THANK YOU!