Serverless Image Processing using AWS Lambda and Amazon S3

1. Introduction
2. What is Serverless Computing?
3. About AWS Services Used
4. Problem Statement
5. Objective of the Project
6. System Architecture
7. Tools and Technologies Used
8. Step-by-Step Implementation
9. Code Explanation
10. Testing and Output
11. Advantages
12. Limitations
13. Conclusion

Introduction

- In today's digital world, high-quality images are essential for websites, e-commerce, and social media platforms. However, large images can slow down page loading times and negatively affect user experience. Therefore, image optimization and resizing is a common and critical task in modern web development.
- To solve this efficiently, we use a serverless solution on AWS that automatically processes and resizes images using AWS Lambda when they are uploaded to Amazon S3.

What is Serverless Computing?

- Serverless computing is a cloud model where you write code and run it without managing servers. In AWS, this is mainly done using Lambda functions, which are automatically triggered by events (like uploading a file to S3).

? About AWS Services Used

- 🟡 1. Amazon S3 (Simple Storage Service):

- A highly scalable object storage service. Used to:
- Store original uploaded images
- Store resized images after processing

- 🟢 2. AWS Lambda:

- A serverless function service that:
- Automatically runs code when triggered
- Resizes and optimizes images using libraries like Pillow (Python) or Sharp (Node.js)

3. IAM (Identity & Access Management):

- Used to define roles and permissions
- Allows Lambda to access S3 securely
-

4. CloudWatch:

- For monitoring logs and Lambda performance
- Helps debug errors in the function

Problem Statement

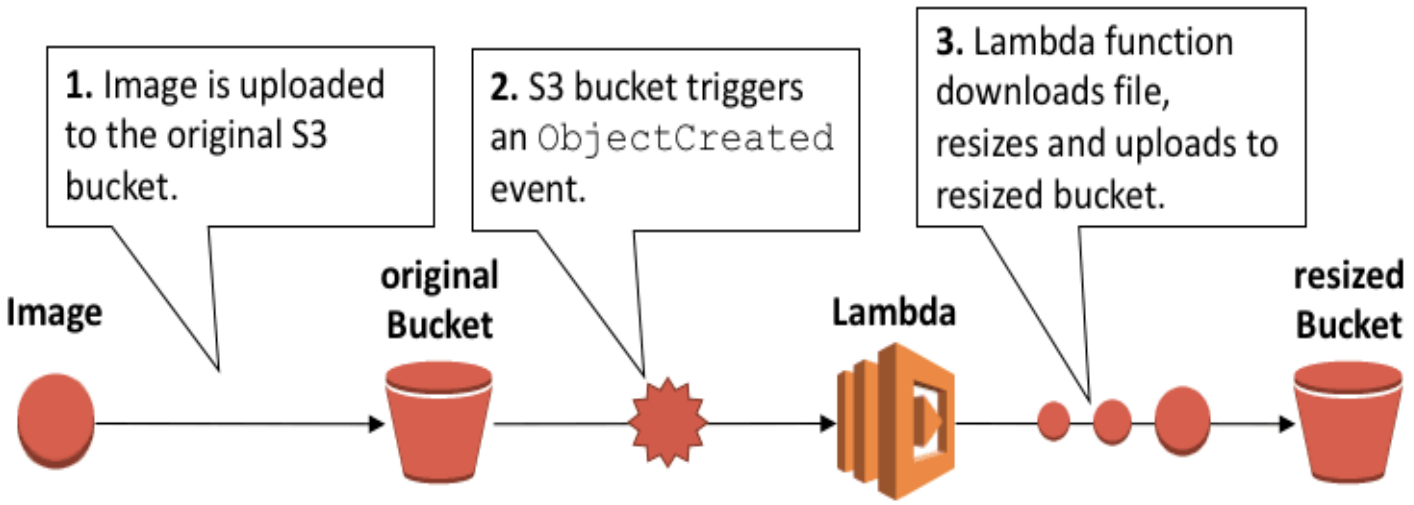
- Uploading large images directly to a website or app can:
- Slow down page load times
- Waste bandwidth
- Reduce user satisfaction
- Manual image resizing is not scalable.

Objective of the Project

- To automate the image processing workflow by:
- Using AWS Lambda to automatically resize images
- Using Amazon S3 to store images
- Using a completely serverless, auto-scaling, and cost-effective solution

System Architecture

- [User]
- ↓
- Upload Image to S3 (original-images)
- ↓
- Trigger AWS Lambda
- ↓
- Resize Image (e.g. 300x300)
- ↓
- Save to S3 (resized-images)
- ↓
- Monitor with CloudWatch



Tools and Technologies Used

Tool	Purpose
AWS Lambda	Serverless image processing
Amazon S3	File storage
IAM	Security and access roles
CloudWatch	Logging and monitoring
Python + Pillow / Node.js + Sharp	Image resizing library

Step-by-Step Implementation

- 🔸 Step 1: Create S3 Buckets

- Go to AWS Console → S3 → Create two buckets:
- original-images
- resized-images

- 🔸 Step 2: Create IAM Role for Lambda

- Go to IAM → Create Role
- Choose Lambda as trusted entity
- Attach policies:
- AmazonS3FullAccess
- CloudWatchLogsFullAccess

Step 3: Write Lambda Code (Python Example)

.

- lambda_function.py
- import boto3
- from PIL import Image
- import io
- s3 = boto3.client('s3')
- def lambda_handler(event, context):
- source_bucket = 'original-images'
- target_bucket = 'resized-images'
- for record in event['Records']:
- key = record['s3']['object']['key']
- image_obj = s3.get_object(Bucket=source_bucket, Key=key)
- image_data = image_obj['Body'].read()
- img = Image.open(io.BytesIO(image_data))
- img = img.resize((300, 300))
- out_buffer = io.BytesIO()
- img.save(out_buffer, format='JPEG')
- out_buffer.seek(0)
- s3.put_object(Bucket=target_bucket, Key=key, Body=out_buffer)

Step 4: Zip the Code

- Zip lambda_function.py and dependencies
- Upload to Lambda console

Step 5: Create Lambda Function

- Go to Lambda → Create Function
- Runtime: Python 3.9 / Node.js
- Upload ZIP file
- Attach IAM role

Step 6: Add Trigger

- Add S3 trigger
- Choose original-images bucket
- Event type: PUT (Object Created)

Code Explanation (Line by Line)

- s3 = boto3.client('s3')
- Creates an S3 client object.
- for record in event['Records']:
- key = record['s3']['object']['key']
- Fetches the name of the uploaded file.
- image_obj = s3.get_object(...)
- image_data = image_obj['Body'].read()
- Downloads the original image from S3.
- img = Image.open(io.BytesIO(image_data))
- img = img.resize((300, 300))
- Opens and resizes the image.
- s3.put_object(...)
- Uploads resized image to the resized-images bucket.

Testing and Output

- Upload image to original-images
- Wait a few seconds
- Check resized-images bucket for resized image
- Go to CloudWatch for log verification
-

Advantages

- Fully automated process
- No server required
- Scalable on demand
- Cost-effective
- Can handle large number of image

Limitations

- Max image size is limited by Lambda memory (~50MB)
- Advanced editing (cropping, filtering) not included
- Only works on S3 uploads (not URL

Conclusion

- This project shows how to use AWS serverless services to automate image processing workflows in a scalable and efficient way. With the power of Lambda and S3, developers can avoid manual work and build smarter applications.