**PROJECT NO-1**

**Serverless Image Processing with AWS Lambda and S3 :**

**It is an automated system where image resizing and optimization are handled without managing any servers manually.**

**Whenever a user uploads an image to an S3 bucket, a Lambda function is triggered that resizes the image using the Pillow library and stores the processed image in another S3 bucket.**

**Table of Content:**

**↪Project Overview**

**↪Objective**

**↪Technologies used**

**↪ Architecture**

**↪ Workflow**

**↪Lambda function and code overview**

**↪ Testing and Result**

**↪ Conclusion**

**1.Project overview:**

**This project aims to automate image resizing and optimization using a serverless architecture on AWS. It eliminates the need for manual intervention or traditional servers by using AWS Lambda and Amazon S3.**

* **Automatically triggered when an image is uploaded to the source bucket**
* **Resizes image to 50% using the Pillow library**
* **Stores the processed image in a separate destination bucket**
* **Logs the entire process using CloudWatch**

**+------------------------+**

**| Upload Image to S3 |**

**| (original-image-bucket)|**

**+----------+-------------+**

**|**

**v**

**+------------------------+**

**| S3 Triggers Lambda |**

**| Function Automatically |**

**+----------+-------------+**

**|**

**v**

**+------------------------+**

**| Lambda Downloads Image |**

**| Uses Pillow to Resize |**

**+----------+-------------+**

**|**

**v**

**+------------------------------+**

**| Upload Resized Image to S3 |**

**| (processed-image-bucket) |**

**+------------------------------+**

**2. Objective:**

**To build a serverless application that automatically resizes and optimizes images uploaded to an AWS S3 bucket using AWS Lambda and the Pillow library.**

* **Eliminate the need for manual image editing**
* **Automate the resizing process with event-driven triggers**
* **Leverage AWS cloud services for scalability and efficiency**
* **Ensure the system is lightweight, cost-effective, and easy to maintain**
* **Store processed images in a separate, organized S3 bucket**

**3.Technologies used:**

**This project uses various AWS services and tools to create a fully serverless image processing pipeline. Amazon S3 (Simple Storage Service) is used to store both the original uploaded images and the resized output images. It also acts as a trigger source whenever a new image is uploaded.**

**The core processing is done using AWS Lambda, which is a serverless compute service that runs code in response to events. In this project, Lambda is automatically triggered by S3, processes the uploaded image using the Pillow library, and uploads the resized image to another S3 bucket.**

**To ensure proper security and access control, AWS IAM (Identity and Access Management) is used. An IAM role is assigned to the Lambda function with the necessary permissions to read from and write to S3 buckets and to log into CloudWatch.**

**The image resizing itself is handled by Pillow, a Python Imaging Library that allows easy manipulation of images. It is used within the Lambda function to resize the uploaded image to 50% of its original dimensions**

**In short, technologies used**

* **AWS S3**
* **AWS Lambda**
* **AWS IAM**
* **Pillow (Python)**
* **Amazon CloudWatch.**

**4. Architecture:**

1. **Amazon S3 (Source Bucket)**

**Stores the original uploaded images  
 📂 Bucket name: original-image-bucket**

1. **AWS Lambda  
    Triggered automatically on image upload  
    📦 Contains code to resize images using Pillow**
2. **Amazon S3 (Target Bucket)  
    Stores the resized/optimized images  
    📂 Bucket name: processed-image-bucket**
3. **Amazon CloudWatch  
    Tracks logs, errors, and performance metrics**
4. **IAM Role  
    Grants Lambda permission to access S3 and CloudWatch**
5. **Workflow:**

**Step-by-Step Process:**

1. **Image Upload**
   * **A user uploads an image to the Original S3 Bucket (original-image-bucket)**
2. **Trigger Lambda Function**
   * **The image upload event automatically triggers the AWS Lambda function**
3. **Download Image**
   * **Lambda downloads the image from the source S3 bucket**
4. **Resize Image**
   * **Image is resized to 50% of its original size using the Pillow (Python Imaging Library)**
5. **Upload Processed Image**
   * **The resized image is uploaded to a second S3 bucket (processed-image-bucket)**
6. **Logging**
   * **Process details and logs are stored in Amazon CloudWatch**
7. **Lambda function and code overview**

* Automatically triggered when a new image is uploaded
* Uses **Pillow** to resize the image
* Saves the resized image to another S3 bucket
* Logs the process using **CloudWatch**

**const sharp = require("sharp");**

**const path = require("path");**

**const AWS = require("aws-sdk");**

**// AWS region setup**

**AWS.config.update({ region: "ap-south-1" });**

**const s3 = new AWS.S3();**

**const processedImageBucket = "serverless-bucket-processed-images"; // Target bucket for resized images**

**exports.handler = async (event, context, callback) => {**

**console.log("📥 New object event:", JSON.stringify(event));**

**let records = event.Records;**

**for (let index = 0; index < records.length; index++) {**

**const record = records[index];**

**try {**

**const bucketName = record.s3.bucket.name;**

**const key = decodeURIComponent(record.s3.object.key.replace(/\+/g, " "));**

**const parsedPath = path.parse(key);**

**const fileName = parsedPath.name;**

**const filePath = parsedPath.dir ? parsedPath.dir + "/" : "";**

**const fileExt = parsedPath.ext;**

**const mimeType = "image/" + (fileExt.substring(1).toLowerCase() === "jpg" ? "jpeg" : fileExt.substring(1));**

**console.log(📂 File Info -> Path: ${filePath}, Name: ${fileName}, Ext: ${fileExt});**

**// Get the uploaded image**

**const originalImage = await s3.getObject({**

**Bucket: bucketName,**

**Key: key,**

**}).promise();**

**// Create thumbnail (300x300)**

**const thumbnail = await sharp(originalImage.Body)**

**.resize(300, 300, { fit: sharp.fit.cover })**

**.withMetadata()**

**.toBuffer();**

**// Create cover photo (800x800)**

**const coverphoto = await sharp(originalImage.Body)**

**.resize(800, 800, { fit: sharp.fit.cover })**

**.withMetadata()**

**.toBuffer();**

**// Upload thumbnail**

**await s3.putObject({**

**Bucket: processedImageBucket,**

**Key: ${filePath}${fileName}\_thumbnail${fileExt},**

**Body: thumbnail,**

**CacheControl: "max-age=3600",**

**ContentType: mimeType,**

**}).promise();**

**console.log(✅ Uploaded thumbnail: ${filePath}${fileName}\_thumbnail${fileExt});**

**// Upload cover photo**

**await s3.putObject({**

**Bucket: processedImageBucket,**

**Key: ${filePath}${fileName}\_coverphoto${fileExt},**

**Body: coverphoto,**

**CacheControl: "max-age=3600",**

**ContentType: mimeType,**

**}).promise();**

**console.log(✅ Uploaded cover photo: ${filePath}${fileName}\_coverphoto${fileExt});**

**} catch (err) {**

**console.error("❌ Error processing record:", err);**

**}**

**}**

**callback(null, "✅ Image processing completed.");**

**};**

**7.Testing and Result:**

**1. Uploaded Sample Image**

* **Uploaded a high-resolution image (test.jpg) to the original-image-bucket**

**2. Triggered Lambda Automatically**

* **Verified that the Lambda function was invoked through the S3 event**

**3. Verified Output**

* **Checked processed-image-bucket for resized image**
* **Image size reduced by 50% (maintaining aspect ratio)**

**4. Checked Logs**

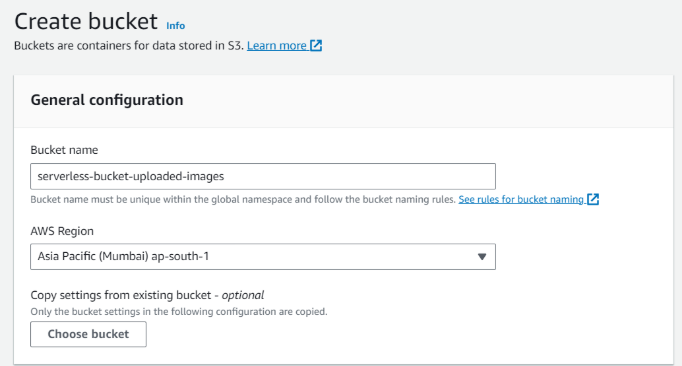
* **Used CloudWatch to confirm execution logs and any errors**
* **Log shows start, image size before and after, and successful upload**

**Step 1 - Creating S3 buckets**

**We will use two S3 buckets:**

1. **source Bucket: For storing uploaded images.**
2. **destination Bucket: For storing processed images.**

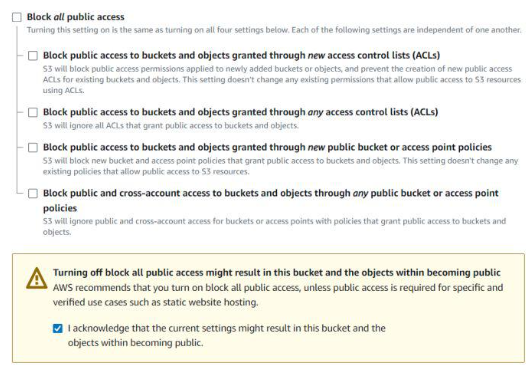
**Go to S3 console and click Create bucket. Enter bucket name as 'serverless-bucket-uploaded-images'. Choose any AWS region as 'ap-south-1'.**

****

**Step 2 - Configuring S3 bucket policy**

In 'Block Public Access settings for this bucket' section disable "block all public access". You will get a warning that the bucket and its objects might become public. Agree to the warning**.**

**(Note: we are making this bucket public only for this project, it is not recommended to make an S3 bucket public if not needed)**.



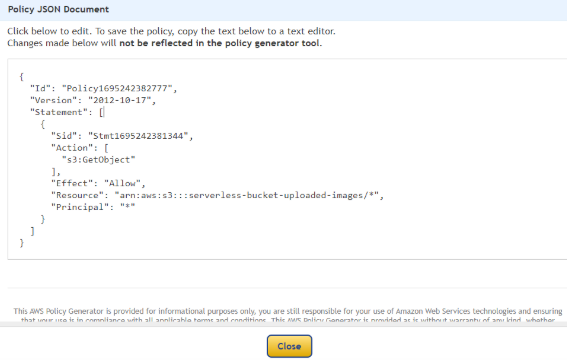
Leave all other settings as default and create bucket. Similarly, create another bucket named 'serverless-bucket-processed-images' with the same region. This bucket will be used to store the processed images. Although we enabled public access while creating the buckets, we still need to attach a bucket policy to access the objects stored in it. (Policies in AWS are JSON documents which defines the permissions for performing actions on a certain resource.)

Go to your source bucket and then click on Permissions tab. Scroll to Bucket Policy tab. Click Edit. You will be redirected to the policy editor. Click on policy generator.

Enter the following settings:

* Type of policy: [S3 Bucket Policy](https://www.geeksforgeeks.org/create-bucket-policy-in-aws-s3-bucket-with-python/)
* Effect:Allow
* Principal: \*
* Actions: GetObject
* Amazon Resource Name (ARN): arn:aws:s3:::SOURCE\_BUCKET\_NAME/\*

SOURCE\_BUCKET\_NAME is the name of the bucket used for uploading the images

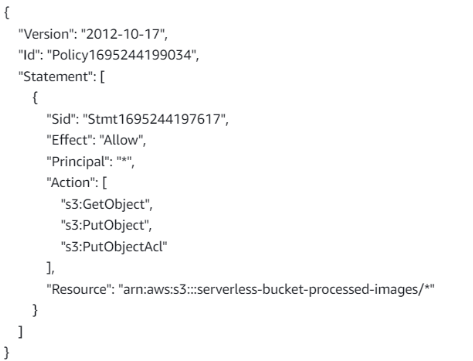
Click Add Statement and then generate policy. Copy the JSON object.

Paste it in the policy editor and then save changes.

Follow same steps to attach a policy to the processed images S3 bucket. The policy settings for destination bucket are:

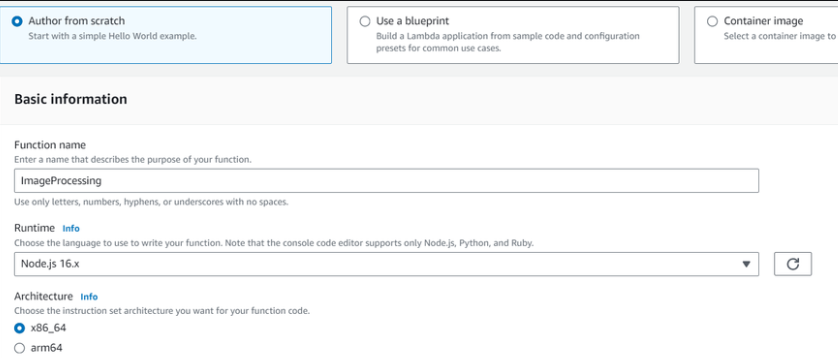
* Type of policy: S3 Bucket Policy
* Effect:Allow
* Principal: \*
* Actions: GetObject, PutObject, and PutObjectAcl
* Amazon Resource Name (ARN): arn:aws:s3:::DESTINATION\_BUCKET\_NAME/\*

DESTINATION\_BUCKET\_NAME is the name of the bucket used for storing processed images.



**Step 3 - Creating Lambda function**

Go to AWS Lambda console. Navigate to Functions section. Click Create Function and name it "ImageProcessing". Select runtime as "NodeJS 16.x" and architecture as "x86\_64". Leave all other settings as default. Create the function.



In the code editor on the Lambda function page paste the following code. This function is executed whenenver an image is uploaded to our source S3 bucket and creates two images (thumbnail (300x300) and coverphoto(800x800)) and stores it in the destination S3 bucket. **(Note: The value of processedImageBucket in the code should be set to the name of the destination bucket).**

**const sharp = require("sharp");**

**const path = require("path");**

**const AWS = require("aws-sdk");**

**// AWS region setup**

**AWS.config.update({ region: "ap-south-1" });**

**const s3 = new AWS.S3();**

**const processedImageBucket = "serverless-bucket-processed-images"; // Target bucket for resized images**

**exports.handler = async (event, context, callback) => {**

**console.log("📥 New object event:", JSON.stringify(event));**

**let records = event.Records;**

**for (let index = 0; index < records.length; index++) {**

**const record = records[index];**

**try {**

**const bucketName = record.s3.bucket.name;**

**const key = decodeURIComponent(record.s3.object.key.replace(/\+/g, " "));**

**const parsedPath = path.parse(key);**

**const fileName = parsedPath.name;**

**const filePath = parsedPath.dir ? parsedPath.dir + "/" : "";**

**const fileExt = parsedPath.ext;**

**const mimeType = "image/" + (fileExt.substring(1).toLowerCase() === "jpg" ? "jpeg" : fileExt.substring(1));**

**console.log(📂 File Info -> Path: ${filePath}, Name: ${fileName}, Ext: ${fileExt});**

**// Get the uploaded image**

**const originalImage = await s3.getObject({**

**Bucket: bucketName,**

**Key: key,**

**}).promise();**

**// Create thumbnail (300x300)**

**const thumbnail = await sharp(originalImage.Body)**

**.resize(300, 300, { fit: sharp.fit.cover })**

**.withMetadata()**

**.toBuffer();**

**// Create cover photo (800x800)**

**const coverphoto = await sharp(originalImage.Body)**

**.resize(800, 800, { fit: sharp.fit.cover })**

**.withMetadata()**

**.toBuffer();**

**// Upload thumbnail**

**await s3.putObject({**

**Bucket: processedImageBucket,**

**Key: ${filePath}${fileName}\_thumbnail${fileExt},**

**Body: thumbnail,**

**CacheControl: "max-age=3600",**

**ContentType: mimeType,**

**}).promise();**

**console.log(✅ Uploaded thumbnail: ${filePath}${fileName}\_thumbnail${fileExt});**

**// Upload cover photo**

**await s3.putObject({**

**Bucket: processedImageBucket,**

**Key: ${filePath}${fileName}\_coverphoto${fileExt},**

**Body: coverphoto,**

**CacheControl: "max-age=3600",**

**ContentType: mimeType,**

**}).promise();**

**console.log(✅ Uploaded cover photo: ${filePath}${fileName}\_coverphoto${fileExt});**

**} catch (err) {**

**console.error("❌ Error processing record:", err);**

**}**

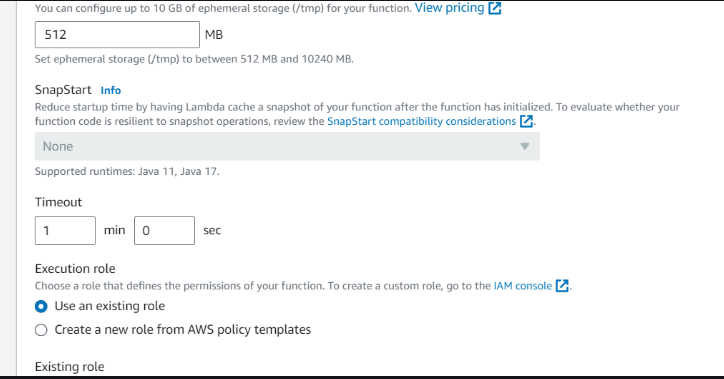
**}**

**callback(null, "✅ Image processing completed.");**

**};**

**Save the code and click Deploy to deploy the changes.**

**Go to Configuration tab and Edit the general configuration. There set the timeout to 1 min (timeout is the maximum time for which a Lambda function will run after which it stops running). We need to increase the timeout because the image can take time to process. Click on Save changes.**

****

**Step 4 - Creating Lambda layer and attaching it to Lambda function**

**Layers in Lambda is used to add dependencies to a Lambda Function. Lambda Layers reduces the code size of Lambda functions as we do not need to upload the dependencies with the function. It also useful for code reusability as we can reuse the layer with multiple functions if they require the same dependencies.**

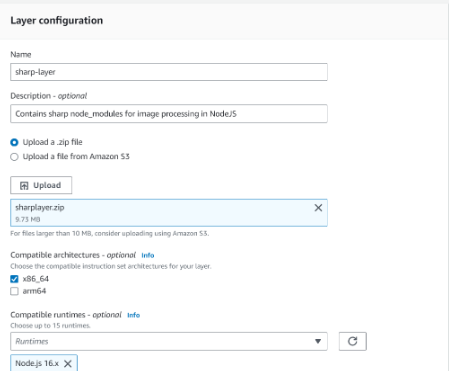
**First we need to create a zip file with all the dependencies (node modules in our case) required by our Lambda function.**

**Create a folder "aws-serverless-image-processor". Inside this directory create another directory "nodejs" (it is compulsory to name this as "nodejs"). Open a terminal an go to nodejs directory. Install *sharp* module with the following command (platform is linux because the Lambda function runs on a Linux machine so we require the node\_modules for Linux).**

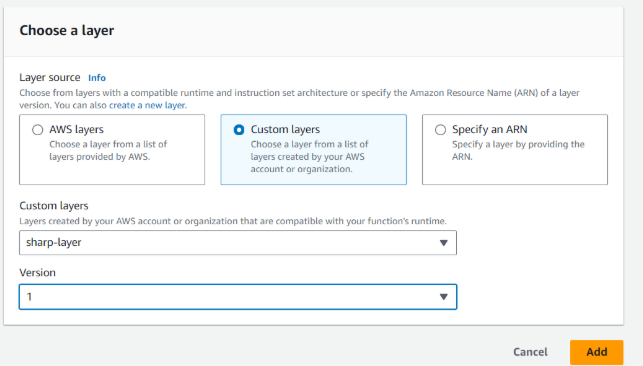
**npm install --arch=x64 --platform=linux sharp**

**Now create a zip file of the nodejs directory and name it "sharplayer.zip".**

**Go to Layers in Lambda console. Click Create layer. Name it "sharp-layer". Upload your nodejs "sharplayer.zip" file here. Select x86\_64 architecture. Select NodeJS 16.x in compatible runtimes. Click on Create Layer.**

****

**Now go to your lambda function page. In Layers section click on Add layer button. Select Custom Layer. Choose "sharp-layer". Select version 1**

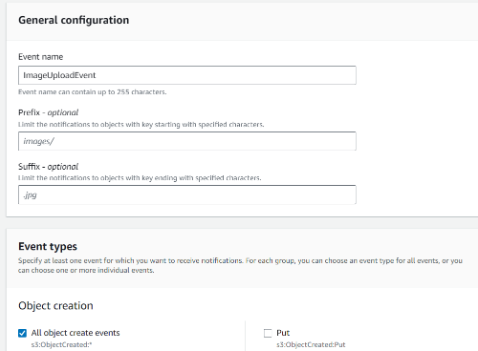
****

**Step 5 - Creating S3 trigger**

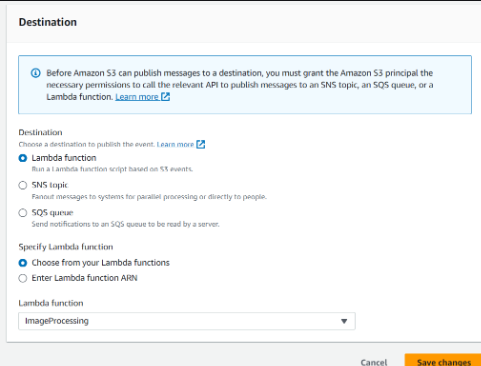
**Now we need our Lambda function to know when an image is uploaded to the source bucket. We can do this by adding an event to the source S3 bucket and configure it to get triggered when an image is uploaded to the bucket which in turn invokes the Lambda function.**

**Go to S3 console. Select the source bucket ("serverless-bucket-uploaded-images"). Go to the Properties tab. Navigate to "Event Notifications". Click "Create Event Notifications".**

**Give an appropriate name to the event. Check the "All object create events".**

****

**Navigate to the "Destination" and select your lambda function. Save changes.**

****

**Step 6 - Testing the application**

**Upload an image file to source S3 bucket ("serverless-bucket-uploaded-images"). Wait for few seconds and check the destination bucket ("serverless-bucket-processed-images"). There you will see two images (thumbnail and coverphoto).**

**8.Conclusion:**

* **Successfully built a serverless image processing pipeline using AWS services**
* **Automated the process of resizing and optimizing images with no manual effort**
* **Used event-driven architecture for efficiency and scalability**
* **Reduced infrastructure management using AWS Lambda**