Project 1

Serverless image processing



Table of content

1. Introduction
2. Requirements
3. Benefits of Serverless Image Processing in AWS
4. Folder Structure
5. Install Sharp
6. Architecture overview

## Step-by-Step Instructions

1. Testing

*1.1 Introduction*

Serverless image processing is a method of processing images in the cloud where the underlying infrastructure (servers) is managed by a cloud provider, rather than the user. This allows developers to focus on writing code for image manipulation without managing servers, making it scalable, cost-effective, and easy to maintain. This can include tasks such as resizing, cropping, filtering, and format conversion.

* **Requirements**
* AWS account
* Amazon S3 bucket
* AWS Lambda function
* IAM Role with proper permissions
* AWS SDK (Node.js or Python)
* **Benefits of Serverless Image Processing in AWS**
* **No server management** (automatic scaling, high availability).
* **Cost-effective**: Pay only for execution time and storage.
* **Easily extensible**: Add features like image optimization, watermarking, face detection, etc.
* **Secure**: IAM policies control access to S3 and Lambda.

**📦 Folder Structure**

text

Copy code

/image-processing-lambda

├── index.js

├── package.json

└── node\_modules/

* Install Sharp

Run in your local project directory:

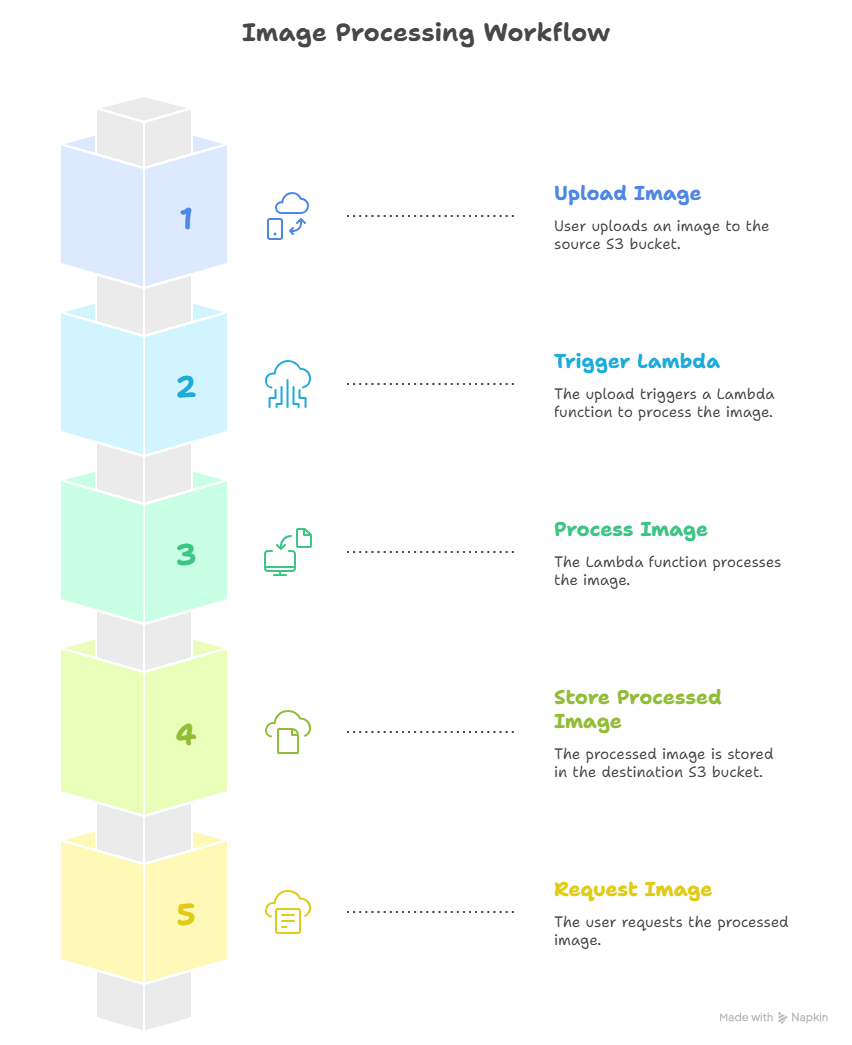
bash

Copy code

npm init –

npm install sharp

* Architecture overview



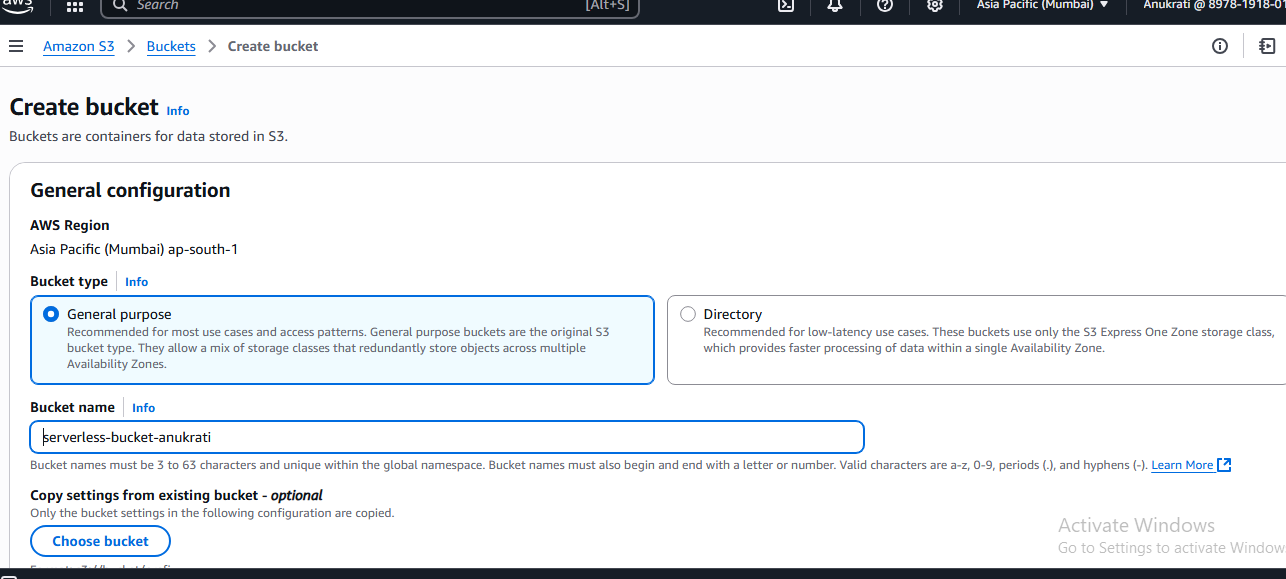
## ✅ Step-by-Step Instructions

## Step 1 - Creating S3 buckets

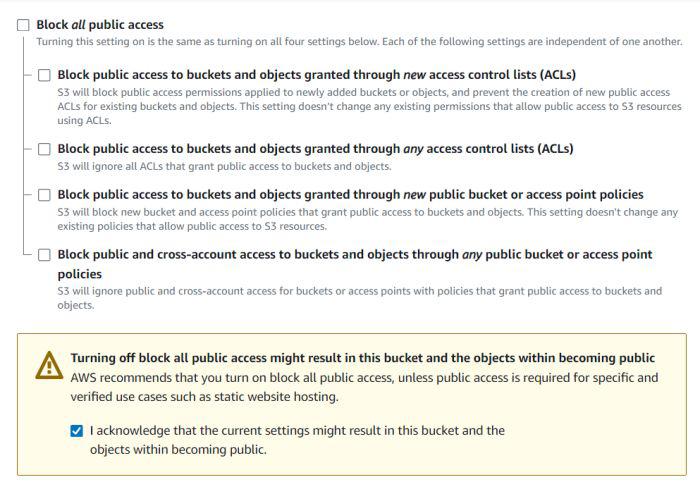
 Go to the **AWS S3 Console**.

 Click **Create bucket**.

 Name it, e.g., serverless-bucket-anukrati



 Enable public access **off** (for security).



Copy the JSON object

{

"Version": "2012-10-17",

"Id": "Policy1",

"Statement": [

{

"Sid": "Stmt1",

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": " arn:aws:s3:::serverless-bucket-processed /\*"

}

]

}

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. 

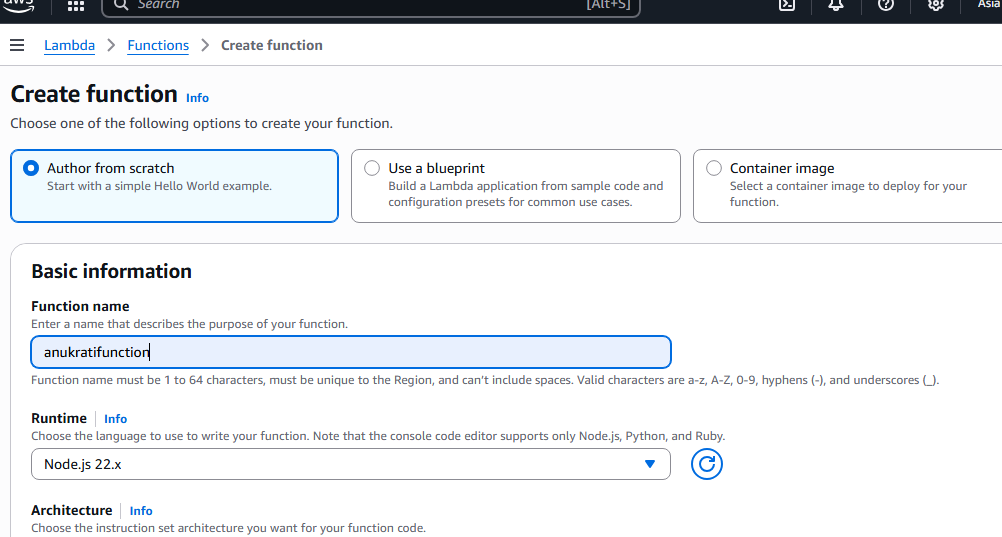
**Step 2: Enable S3 Event Notifications**

Configure S3 event trigger on the uploads/ prefix.

Trigger a Lambda function whenever a new image is uploaded.

**Step 3: Write a Lambda Function**

* Go to **AWS Lambda Console**.
* Click **Create function** and then Author from scratch.
* Choose runtime (e.g., **Python** or **Node.js**).



In the code editor on the Lambda function page paste the following code.

const sharp = require("sharp");

const path = require("path");

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

// Set the REGION

AWS.config.update({

region: "ap-south-1",

});

const s3 = new AWS.S3();

const processedImageBucket = "serverless-bucket-processed-images";

// This Lambda function is attached to an S3 bucket. When any object is added in the S3

// bucket this handler will be called. When an image file is added in the S3 bucket, this function

// creates a square thumbnail of 300px x 300px size and it also creates a cover photo of

// 800px x 800px size. It then stores the thumbnail and coverphotos back to another S3 bucket

// at the same location as the original image file.

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

console.log("An object was added to S3 bucket", JSON.stringify(event));

let records = event.Records;

// Each record represents one object in S3. There can be multiple

// objects added to our bucket at a time. So multiple records can be there

// How many records do we have? Each record represent one object in S3

let size = records.length;

for (let index = 0; index < size; index++) {

let record = records[index];

console.log("Record: ", record);

// Extract the file name, path and extension

let fileName = path.parse(record.s3.object.key).name;

let filePath = path.parse(record.s3.object.key).dir;

let fileExt = path.parse(record.s3.object.key).ext;

console.log("filePath:" + filePath + ", fileName:" + fileName + ", fileExt:" + fileExt);

// Read the image object that was added to the S3 bucket

let imageObjectParam = {

Bucket: record.s3.bucket.name,

Key: record.s3.object.key,

};

let imageObject = await s3.getObject(imageObjectParam).promise();

// Use sharp to create a 300px x 300px thumbnail

// withMetadata() keeps the header info so rendering engine can read

// orientation properly.

let resized\_thumbnail = await sharp(imageObject.Body)

.resize({

width: 300,

height: 300,

fit: sharp.fit.cover,

})

.withMetadata()

.toBuffer();

console.log("thumbnail image created");

// Use sharp to create a 800px x 800px coverphoto

let resized\_coverphoto = await sharp(imageObject.Body)

.resize({

width: 800,

height: 800,

fit: sharp.fit.cover,

})

.withMetadata()

.toBuffer();

console.log("coverphoto image created");

// The processed images are written to serverless-image-processing-bucket.

let thumbnailImageParam = {

Body: resized\_thumbnail,

Bucket: processedImageBucket,

Key: fileName + "\_thumbnail" + fileExt,

CacheControl: "max-age=3600",

ContentType: "image/" + fileExt.substring(1),

};

let result1 = await s3.putObject(thumbnailImageParam).promise();

console.log("thumbnail” + fileExt,

CacheControl: "max-age=3600",

ContentType: "image/" + fileExt.substring(1),

};

let result1 = await s3.putObject(thumbnailImageParam).promise();

console.log("thumbnail image uploaded:" + JSON.stringify(result1));

let coverphotoImageParam = {

Body: resized\_coverphoto,

Bucket: processedImageBucket,

Key: fileName + "\_coverphoto" + fileExt,

CacheControl: "max-age=3600",

ContentType: "image/" + fileExt.substring(1),

};

let result2 = await s3.putObject(coverphotoImageParam).promise();

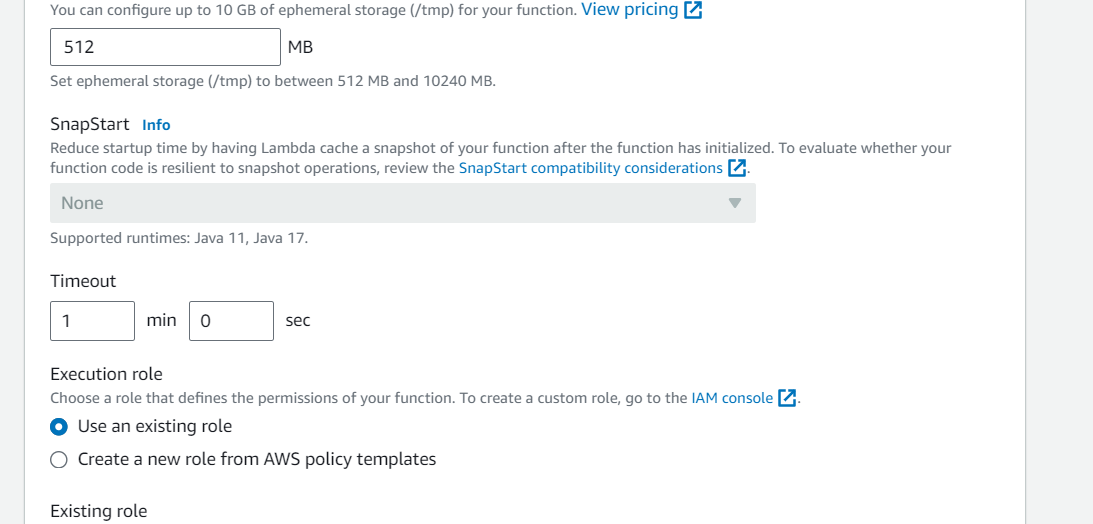
console.log("coverphoto image uploaded:" + JSON.stringify(result2));

}

};

Save the code and click Deploy to deploy the changes.

Go to Configuration tab and Edit the general configuration.



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

### **Install Sharp**

Run this locally and zip it with your Lambda:

bash

CopyEdit

mkdir image-resizer &&cd image-resizer

npminit -y

npm install sharp

Zip contents for upload:

bash

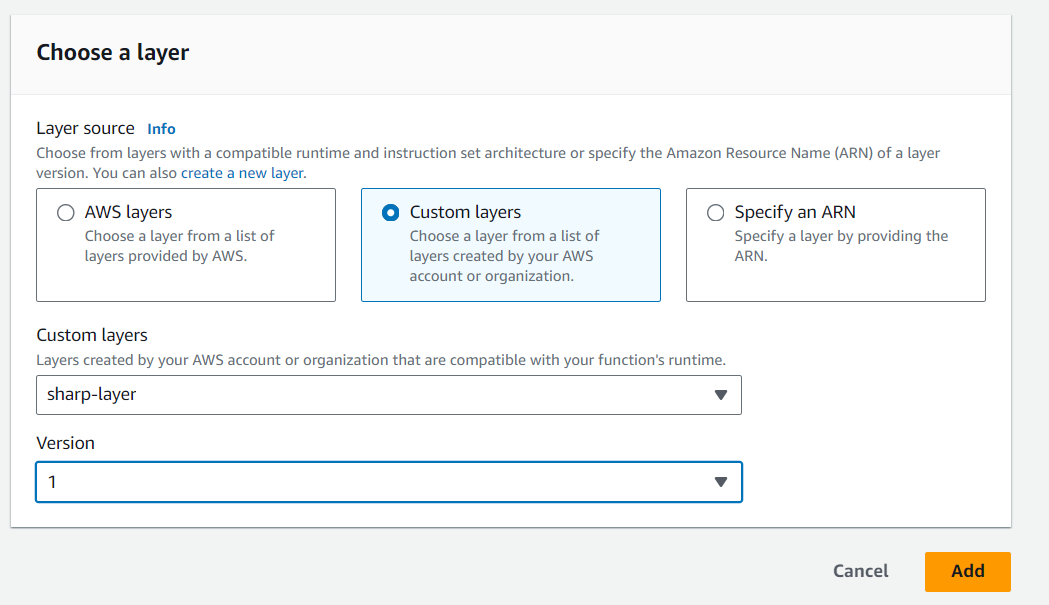
CopyEdit

zip -r image-resizer.zip .

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

Go to Layers in Lambda console. Click Create layer.

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



## Step 5 - Creating S3 trigger

**Step 6: Set Up IAM Roles and Permissions**

Ensure the Lambda function has permissions to:

Read from and write to S3

Log to CloudWatch

json

Copy code

{

"Effect": "Allow",

"Action": ["s3:GetObject", "s3:PutObject"],

"Resource": "arn:aws:s3:::my-image-bucket/\*"

}

## 🌐 *Testing Tip*

You can manually upload a sample image to the image-uploads bucket via the AWS Console and verify that:

1. Lambda is triggered
2. A processed image appears in image-processed
3. Logs show success (CloudWatch Logs)