

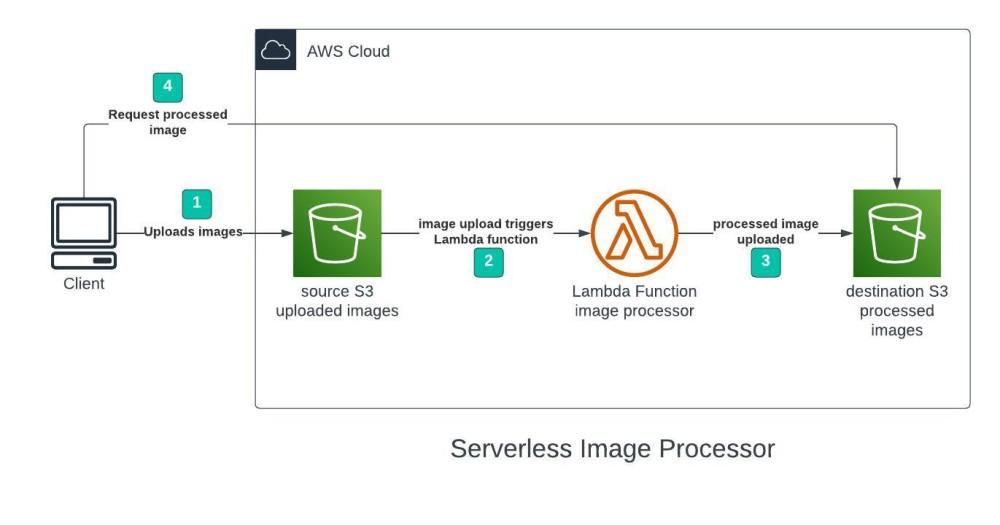
Name: RIDDHI PANDEY

Semester: 2nd

**Project -1**

**Topic:**

**Create a serverless image processing application that automatically resizes and optimizes images uploaded to an Amazon S3 bucket.**



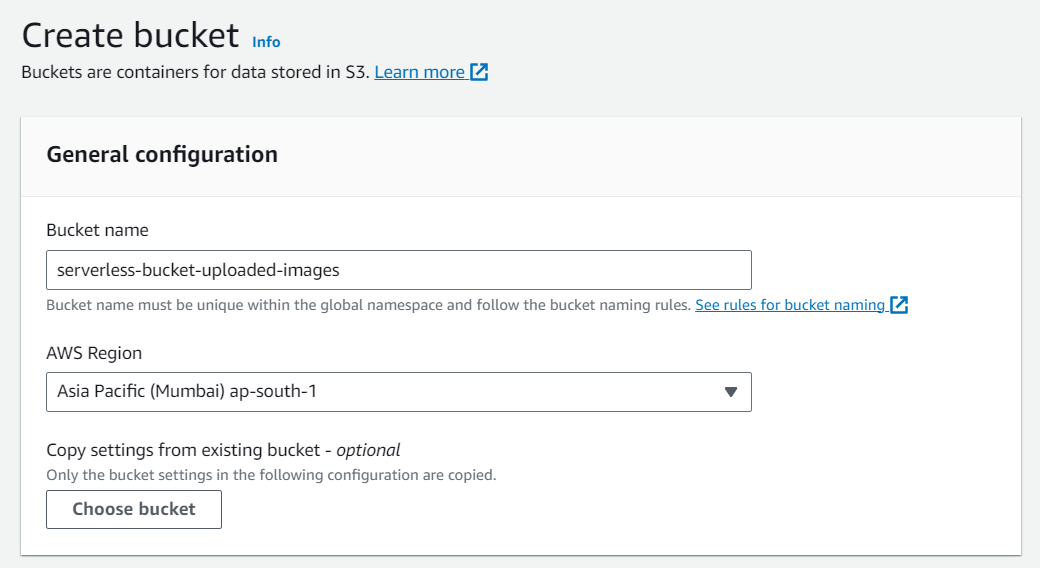
**Table of contents**

1. Creating S3 Bucket
2. Configuring S3 Bucket Policy
3. Creating Lambda Function
4. Creating Lambda layer and attaching it to Lambda function
5. Creating S3 trigger
6. Testing the application
7. **Creating S3 Bucket**

We will use 2 S3 Bucket:

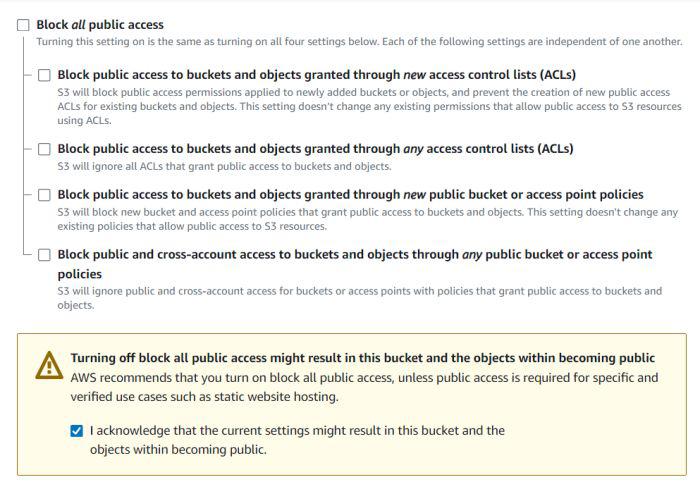
1. Source bucket: For storing uploaded images.
2. Destination bucket: For storing processes images.

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



1. **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.)

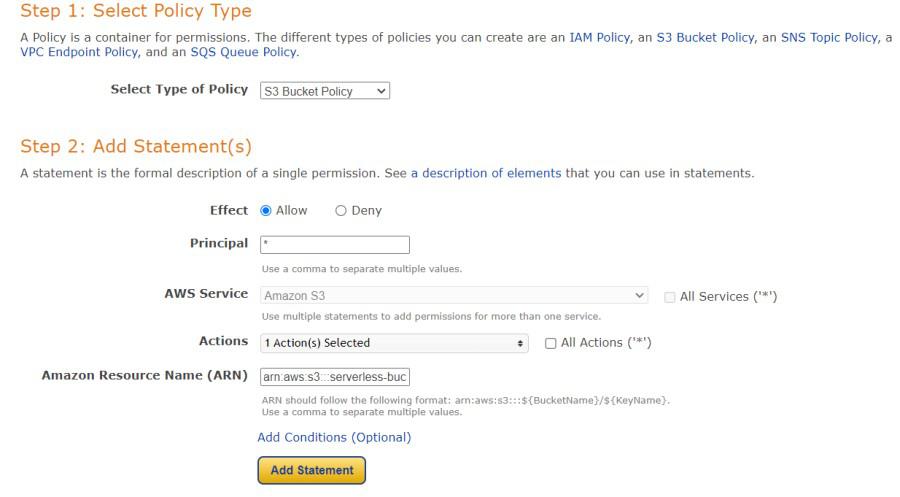
Go to your source bucket and then click on Permissions tab. Scroll to Bucket Policy tab. Click Edit. You will be redirected to the policy editor. Click on policy generator.

Enter the following settings:

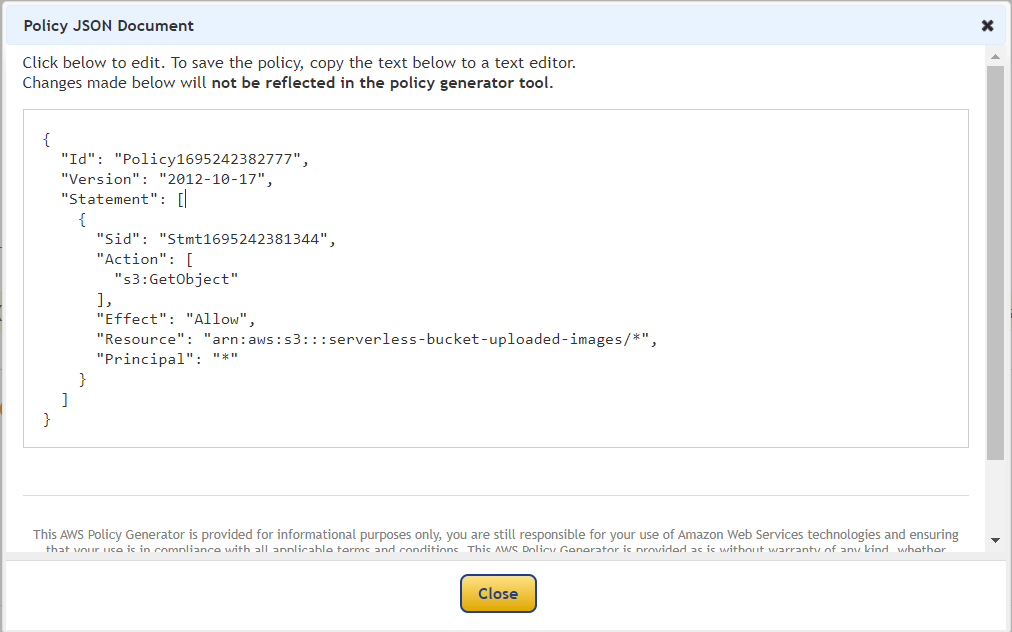
Type of policy:

* S3 Bucket Policy
* 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.

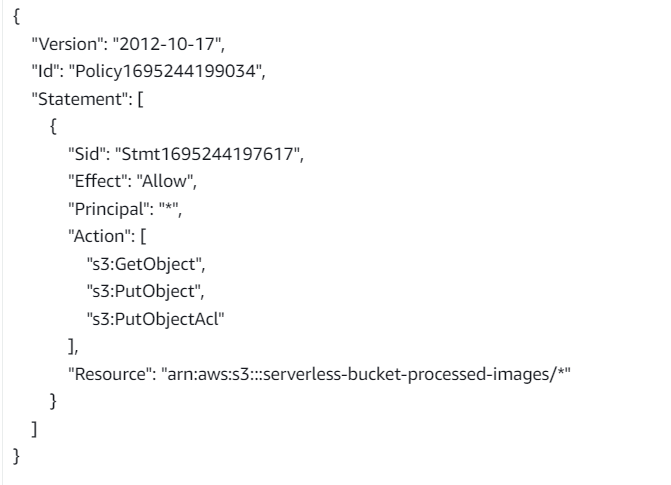


Paste it in the policy editor and then save changes.

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

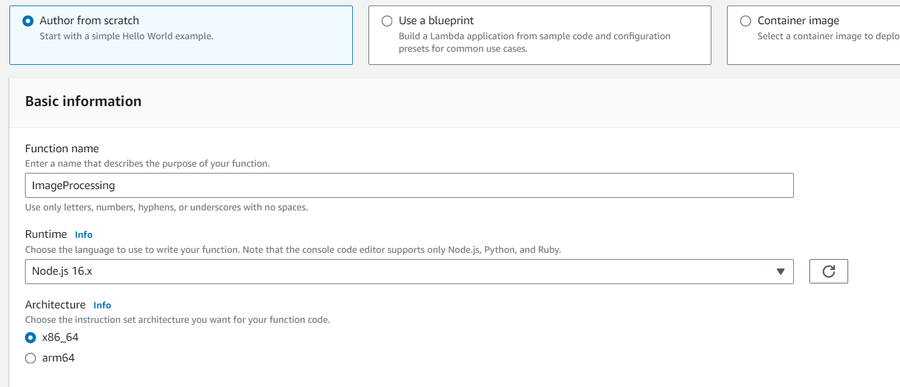
* Type of policy: S3 Bucket Policy
* Effect:Allow
* Principal: \*
* Actions: GetObject, PutObject, and PutObjectAcl
* Amazon Resource Name (ARN): arn:aws:s3:::DESTINATION\_BUCKET\_NAME/\*

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



1. **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 (300×300) and coverphoto(800×800)) 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).

Java 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 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();

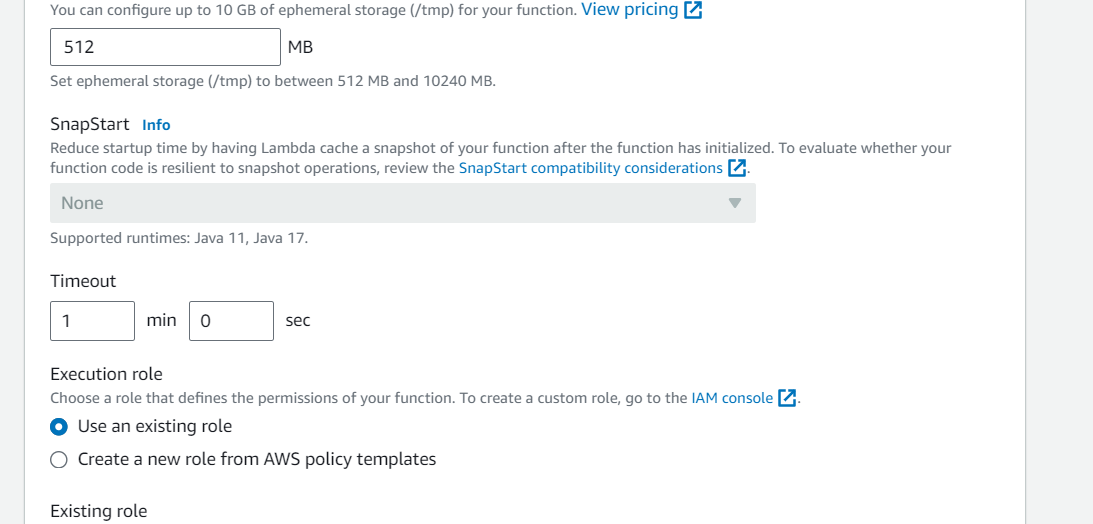
console.log("coverphoto image uploaded:" + JSON.stringify(result2));

}

};

Save the code and click Deploy to deploy the changes.

Go to Configuration tab and Edit the general configuration. 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.



1. **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.

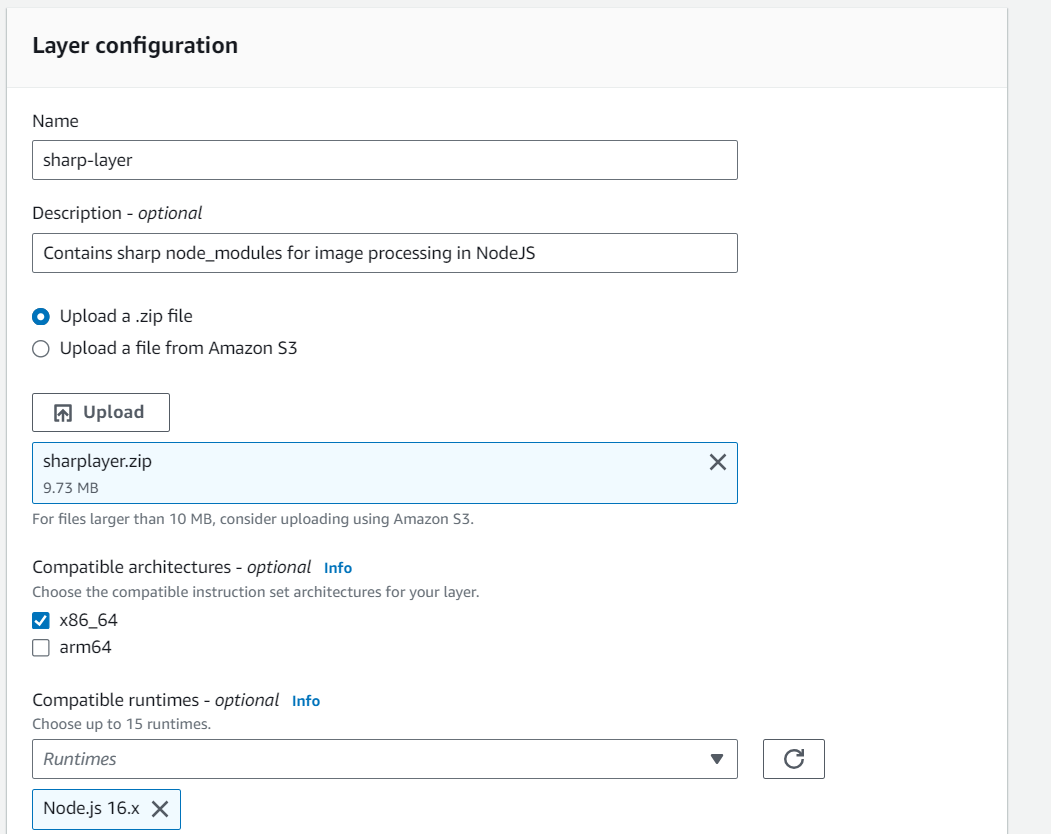
First we need to create a zip file with all the dependencies (node modules in our case) required by our Lambda function.

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).

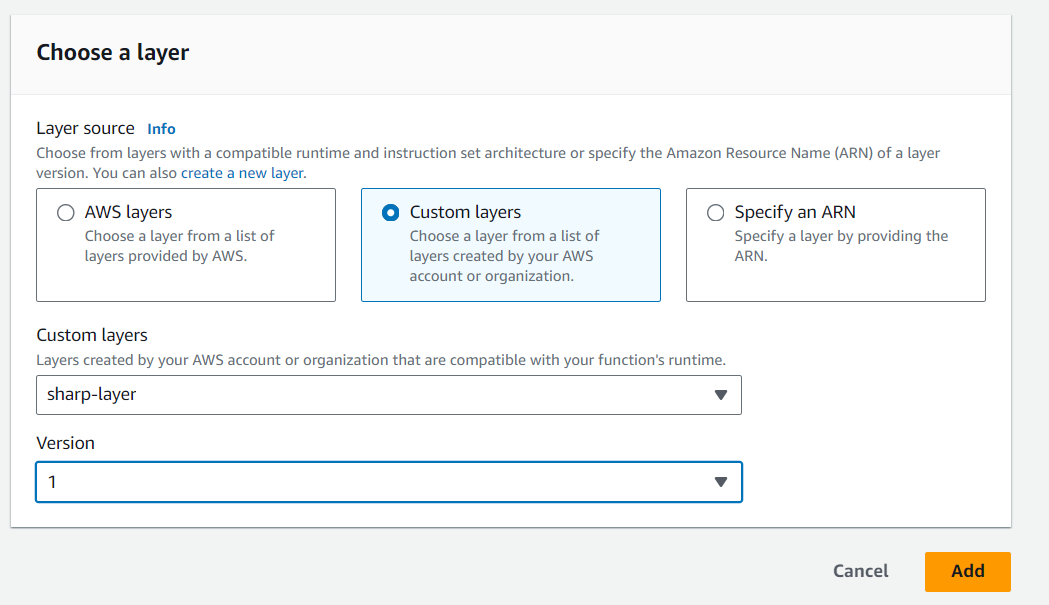
**npm install --arch=x64 --platform=linux sharp**

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

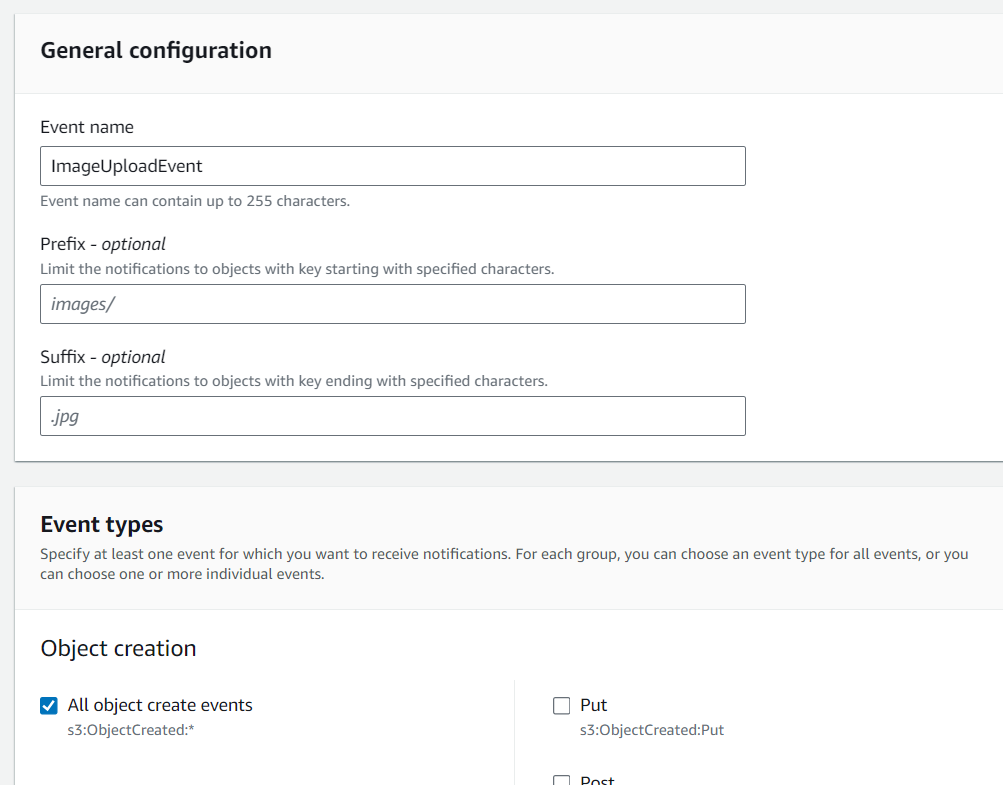


1. **Creating S3 trigger**

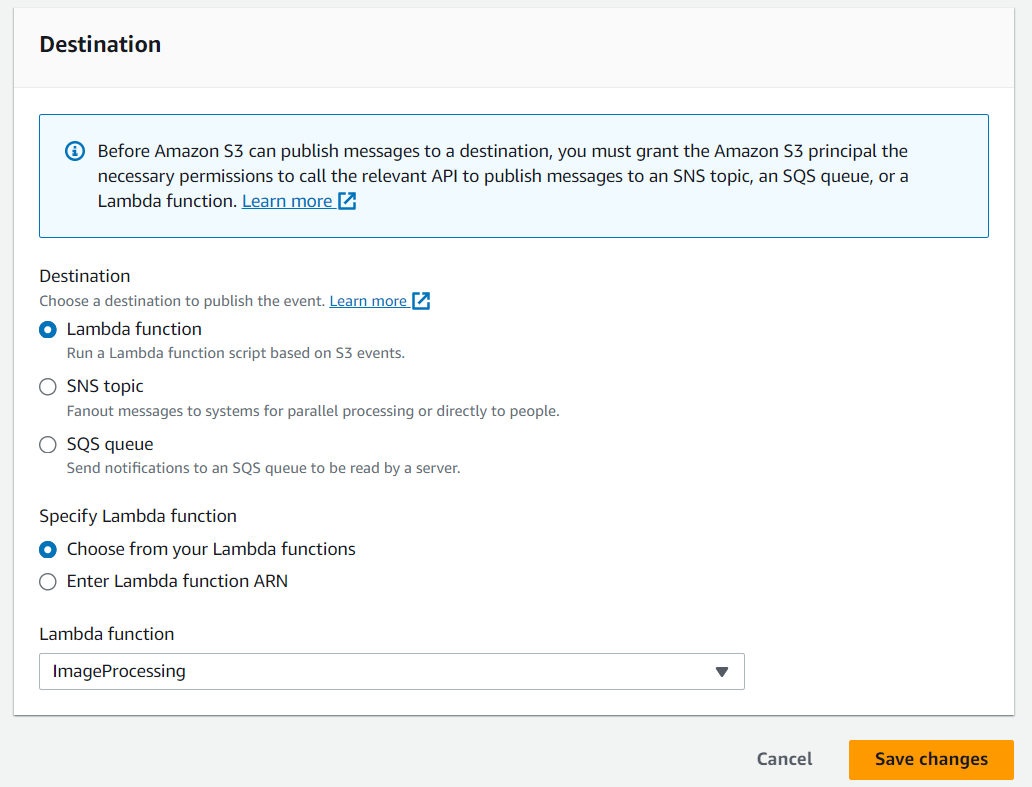
Now we need our Lambda function to know when an image is uploaded to the source bucket. We can do this by adding an event to the source S3 bucket and configure it to get triggered when an image is uploaded to the bucket which in turn invokes the Lambda function.

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.



**VI-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).

**Congratulations, you just built a serverless Image processing application.**