PROJECT-1

Serverless Image Processing with AWS Lambda and S3

AWS S3 (Simple Storage Service) is a cloud data storage service. It is one of the most popular services of AWS. It has high scalability, availability, security and is cost effective. S3 has different storage tiers depending on the use case. Some common use cases of AWS S3 are:

1. **Storage:** It can be used for storing large amounts of data.
2. **Backup and Archive:** S3 has different storage tiers based on how frequent the data is accessed which can be used to backup critical data at low costs.
3. **Static website:** S3 offers static website hosting through HTML files stored in S3.
4. **Data lakes and big data analytics:** Companies can use AWS S3 as a data lake and then run analytics on it for getting business insights and take critical decisions.

AWS Lambda

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. Lambda functions run on demand i.e. they execute only when needed and you pay only for what you compute. Lambda

Scanned with OKEN Scanner

PROJECT-1

Serverless Image Processing with AWS Lambda and S3

AWS S3 (Simple Storage Service) is a cloud data storage service. It is one of the most popular services of AWS. It has high scalability, availability, security and is cost effective. S3 has different storage tiers depending on the use case. Some common use cases of AWS S3 are:

Aman Shukla

1. **Storage:** It can be used for storing large amounts of data.
2. **Backup and Archive:** S3 has different storage tiers based on how frequent the data is accessed which can be used to backup critical data at low costs.
3. **Static website:** S3 offers static website hosting through HTML files stored in S3.
4. **Data lakes and big data analytics:** Companies can use AWS S3 as a data lake and then run analytics on it for getting business insights and take critical decisions.

AWS Lambda

AWS Lambda is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. Lambda functions run on demand i.e. they execute only when needed and you pay only for what you compute. Lambda

Scanned with OKEN Scanner

is well integrated with may other AWS services. It supports a wide variety of programming languages.

Some common use cases for AWS Lambda are:

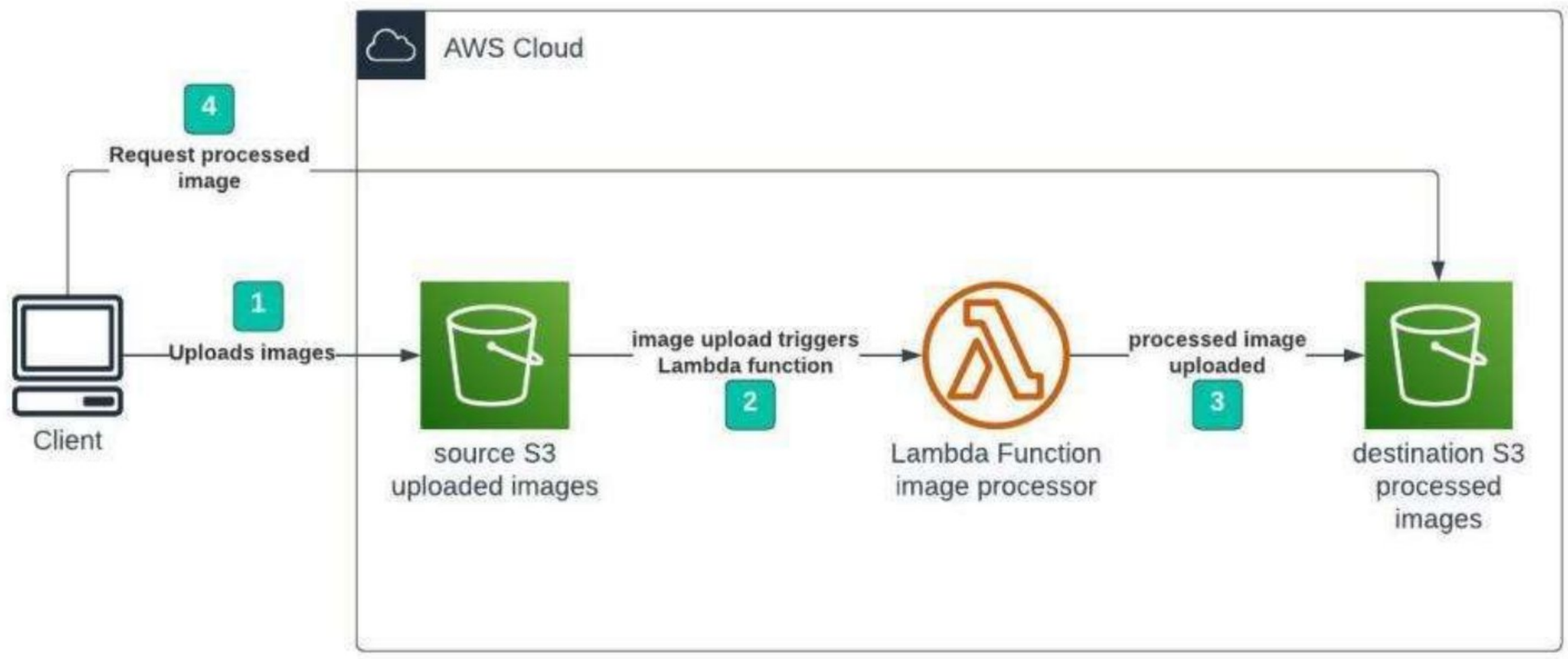
1. **File processing:** You can use Lambda for processing files as they are uploaded in an S3 bucket or whenever some event triggers the function.
2. **Data and analytics:** You can pass a data stream to your Lambda function and then create analysis from that.
3. **Website:** Lambda can also be used for creating websites. This is cost effective because you are charged only for the time when the servers are running.

In this article, we will be using AWS Lambda for processing images.

Serverless Image Processing Flow

1. User uploads a file to the source S3 bucket (which is used for storing uploaded images).
2. When the image is uploaded to a source S3 bucket, it triggers an event which invokes the Lambda function. The lambda function processes the image.
3. Processed image is stored in the destination S3 bucket.
4. The processed image is requested by the user.

Scanned with OKEN Scanner



Serverless Image Processor

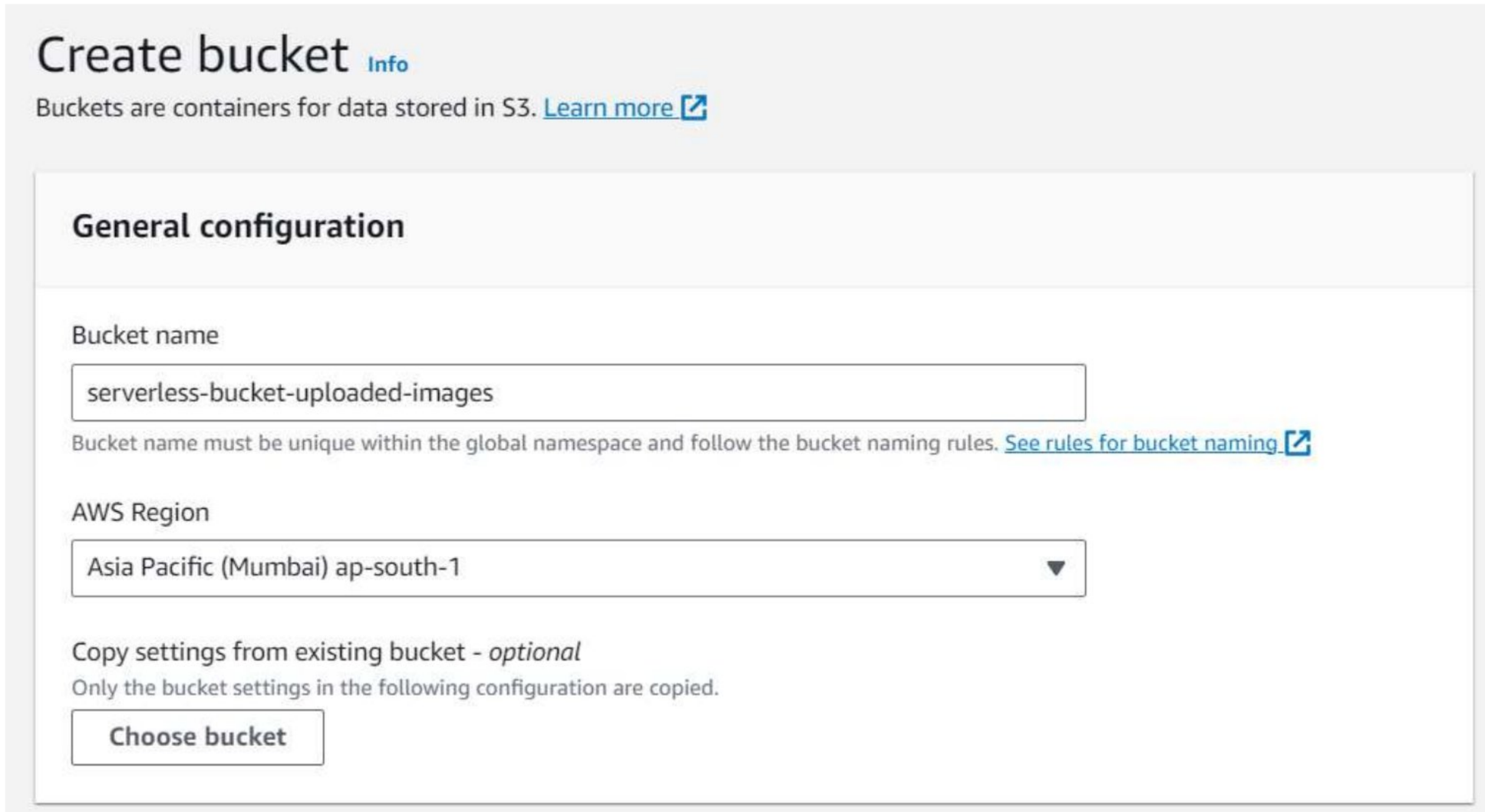
Step 1 - Creating S3 buckets

We will use two S3 buckets:

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

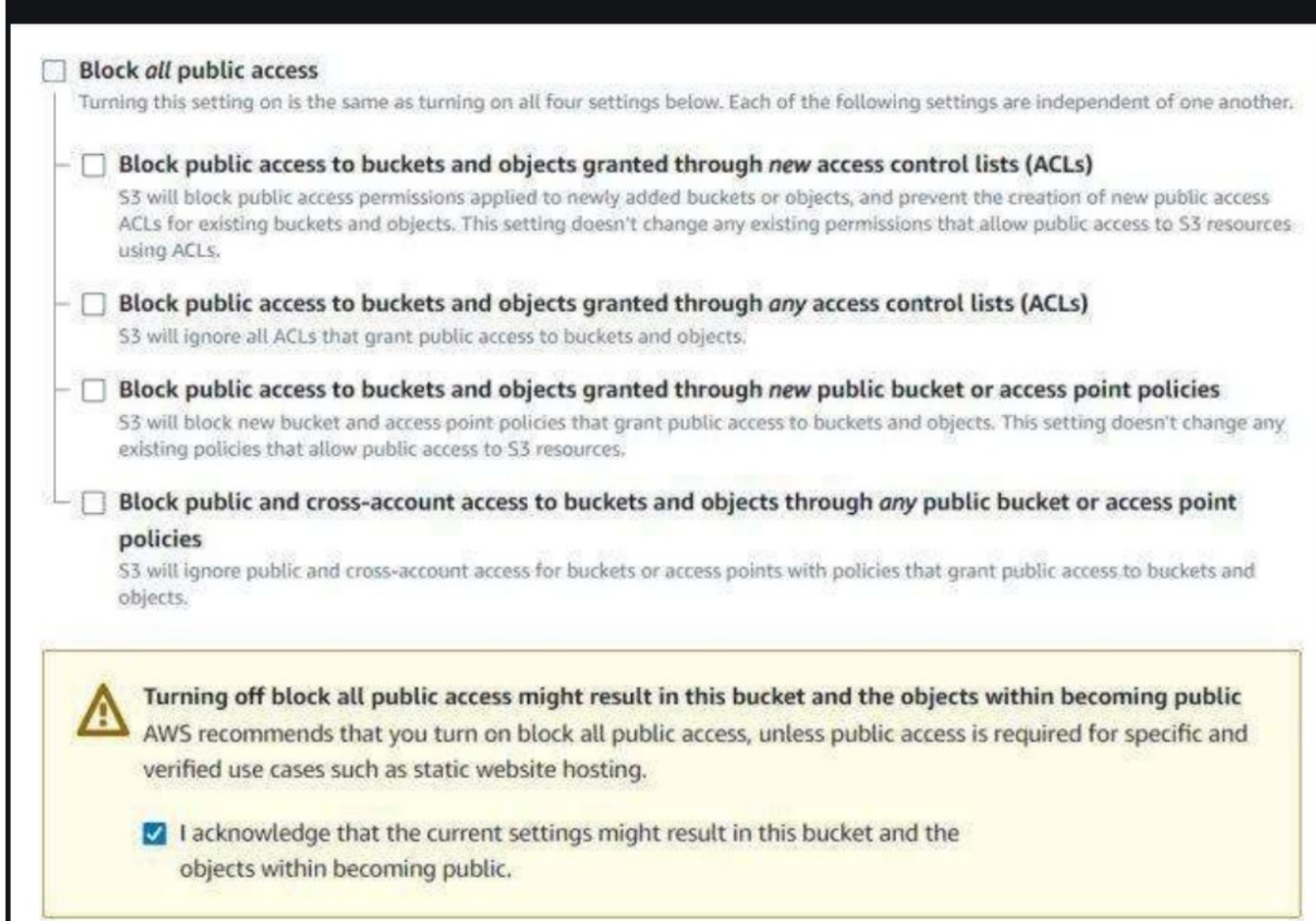
Go to S3 console and click Create bucket. Enter bucket name as 'serverless-bucket-uploaded-images'. Choose any AWS region as 'ap-south-1'.

Scanned with OKEN Scanner



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)**.



Scanned with OKEN Scanner

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.

Scanned with OKEN Scanner

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an IAM Policy, an S3 Bucket Policy, an SNS Topic Policy, a VPC Endpoint Policy, and an SQS Queue Policy.

Select Type of Policy S3 Bucket Policy

Step 2: Add Statement(s)

A statement is the formal description of a single permission. See a description of elements that you can use in statements.

Effect ◉ Allow ○ Deny

Principal *

Use a comma to separate multiple values.

AWS Service Amazon S3 ☐ All Services ('*')

Use multiple statements to add permissions for more than one service.

Actions 1 Action(s) Selected ☐ All Actions ('*')

Amazon Resource Name (ARN) arn:aws:s3:::serverless-buc

ARN should follow the following format: arn:aws:s3:::${BucketName}/${KeyName}.
Use a comma to separate multiple values.

Add Conditions (Optional)

Add Statement

Policy generator page

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

Policy JSON Document

Click below to edit. To save the policy, copy the text below to a text editor.
Changes made below will **not be reflected in the policy generator tool.**


```
{
  "Id": "Policy1695242382777",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1695242381344",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::serverless-bucket-uploaded-images/*",
      "Principal": "*"
    }
  ]
}
```


This AWS Policy Generator is provided for informational purposes only, you are still responsible for your use of Amazon Web Services technologies and ensuring that your use is in compliance with all applicable terms and conditions. This AWS Policy Generator is provided as is without warranty of any kind, whether

Close

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

Scanned with OKEN Scanner

- Principal: *
- Actions: GetObject, PutObject, and PutObjectAcl
- Amazon Resource Name (ARN): arn:aws:s3:::DESTINATION_BUCKET_NAME/*

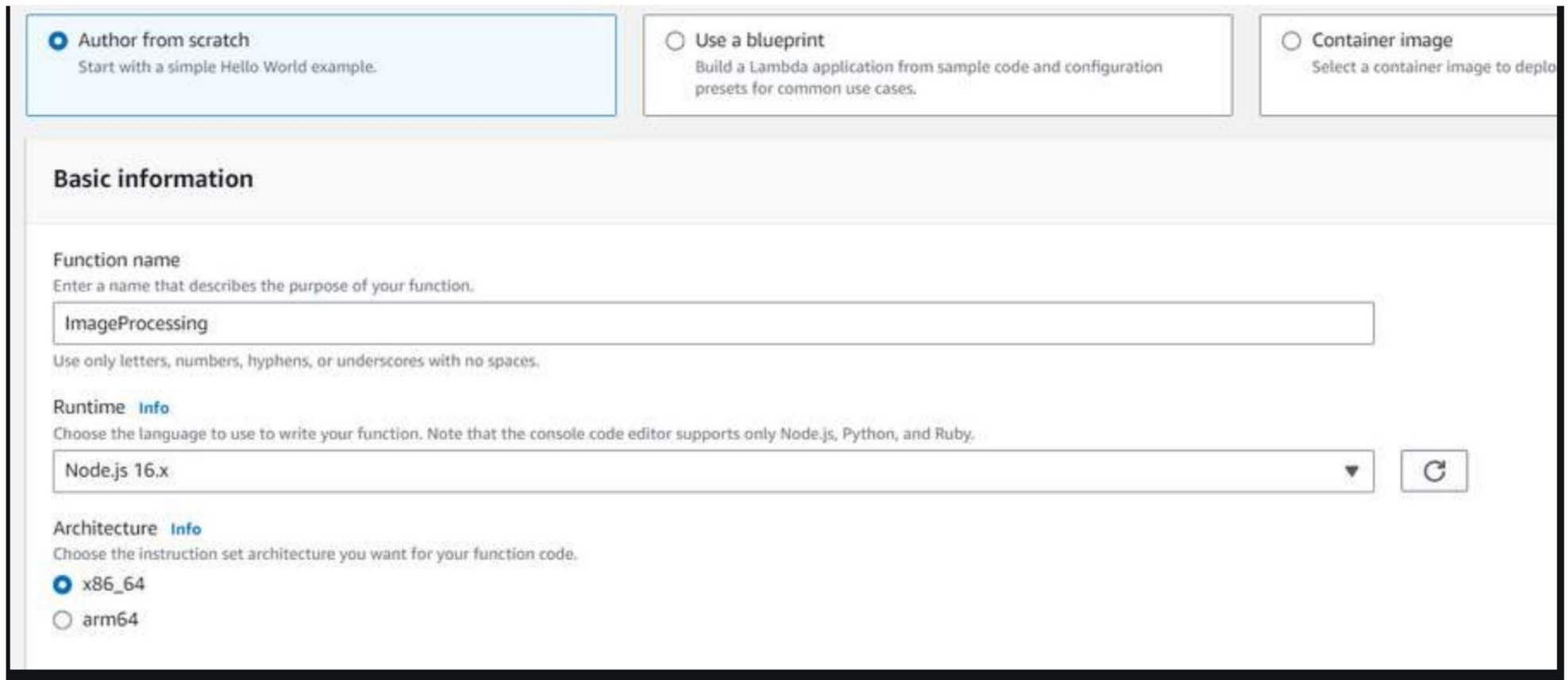
DESTINATION_BUCKET_NAME is the name of the bucket used for storing processed images.

```
{
  "Version": "2012-10-17",
  "Id": "Policy1695244199034",
  "Statement": [
    {
      "Sid": "Stmt1695244197617",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::serverless-bucket-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.

Scanned with OKEN Scanner



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"); // Set the
REGIONAWS.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: ",
```

Scanned with OKEN Scanner

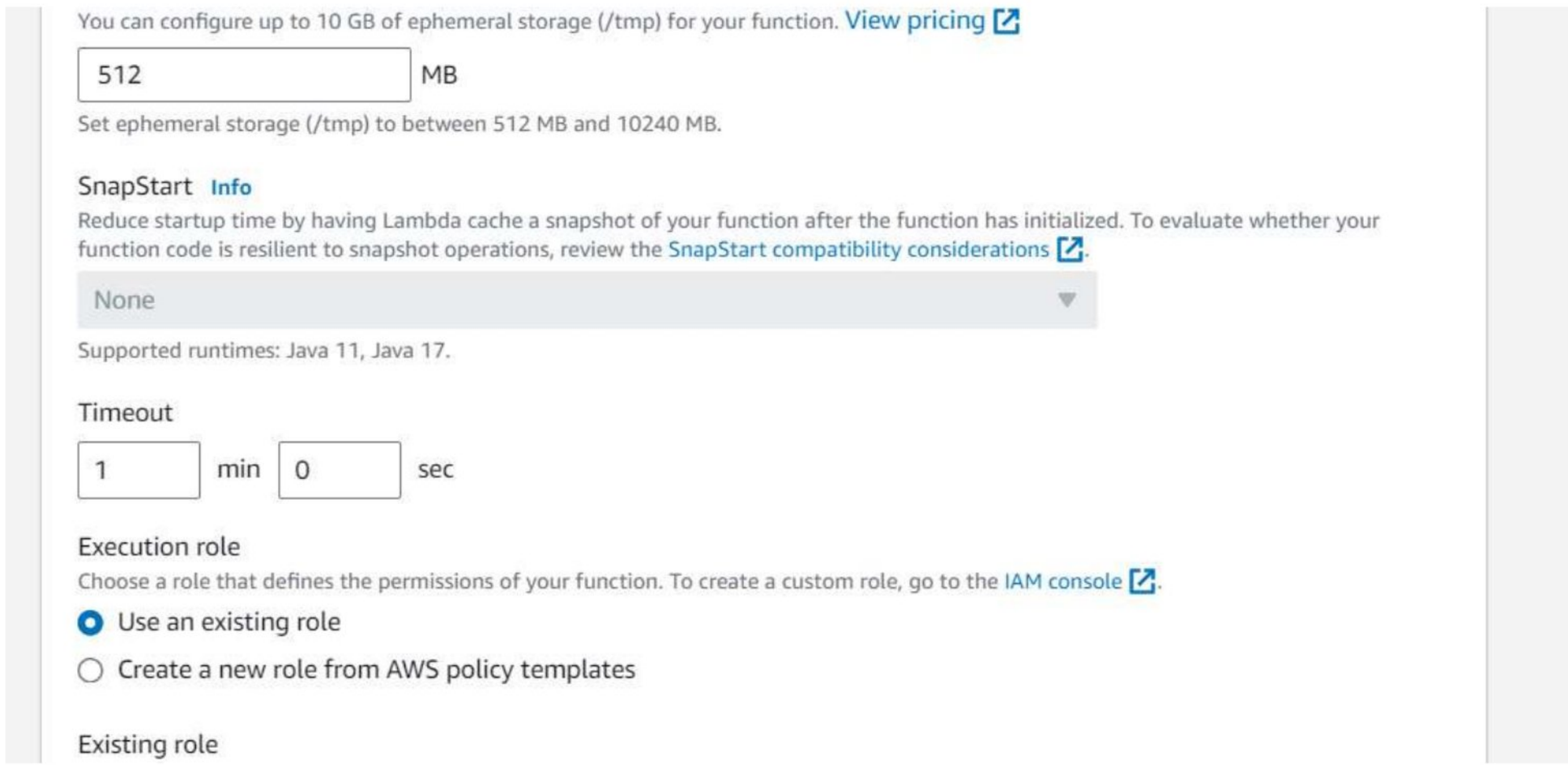
```
record);        // Extract the file name, path and extension        let
fileName = path.parse(record.s3.object.key).name;        let filePath =
path.parse(record.s3.object.key).dir;        let fileExt =
path.parse(record.s3.object.key).ext;         console.log("filePath:" +
filePath + ", fileName:" + 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()            .toBu
ffer();        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()            .toBu
ffer();        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

Scanned with OKEN Scanner

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.

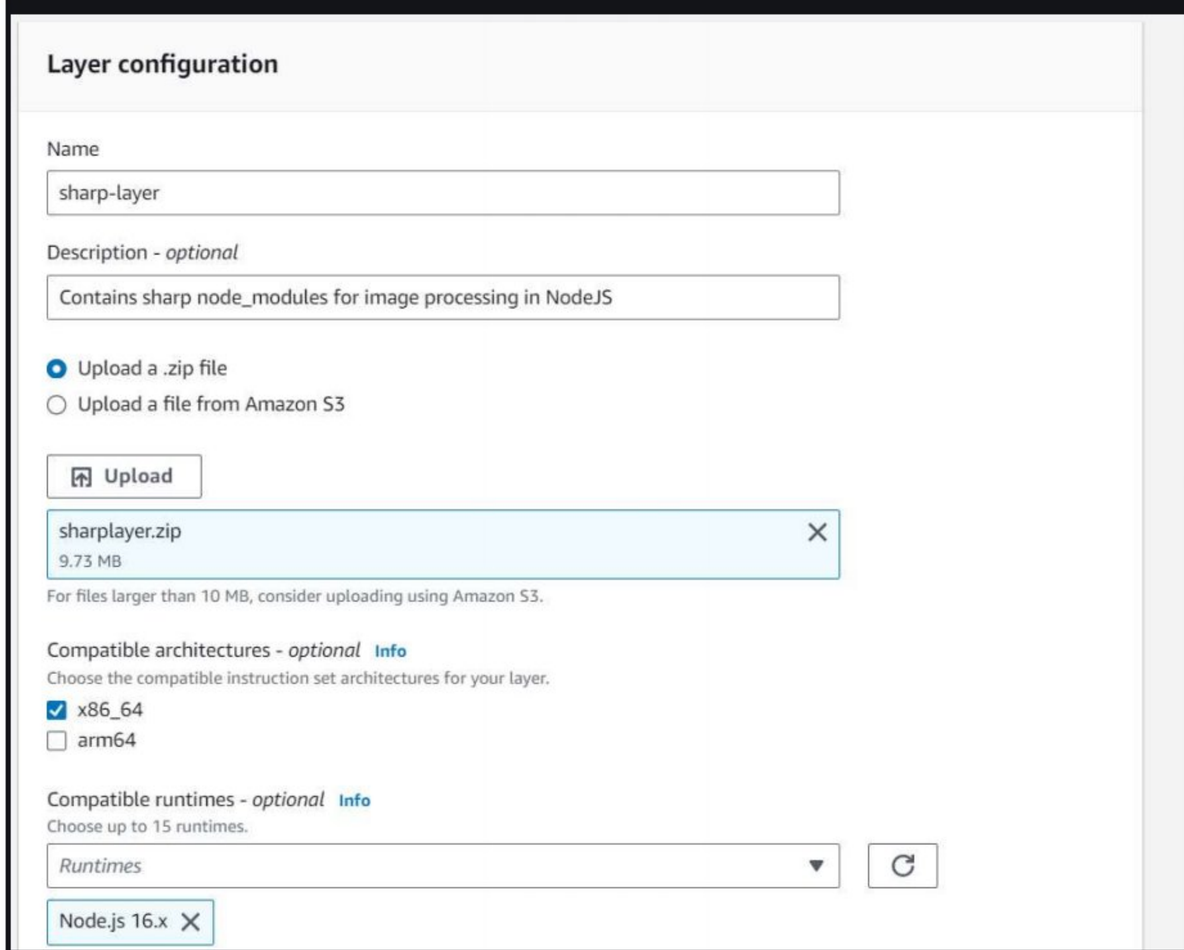
First we need to create a zip file with all the dependencies (node modules in our case) required by our Lambda function.

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).

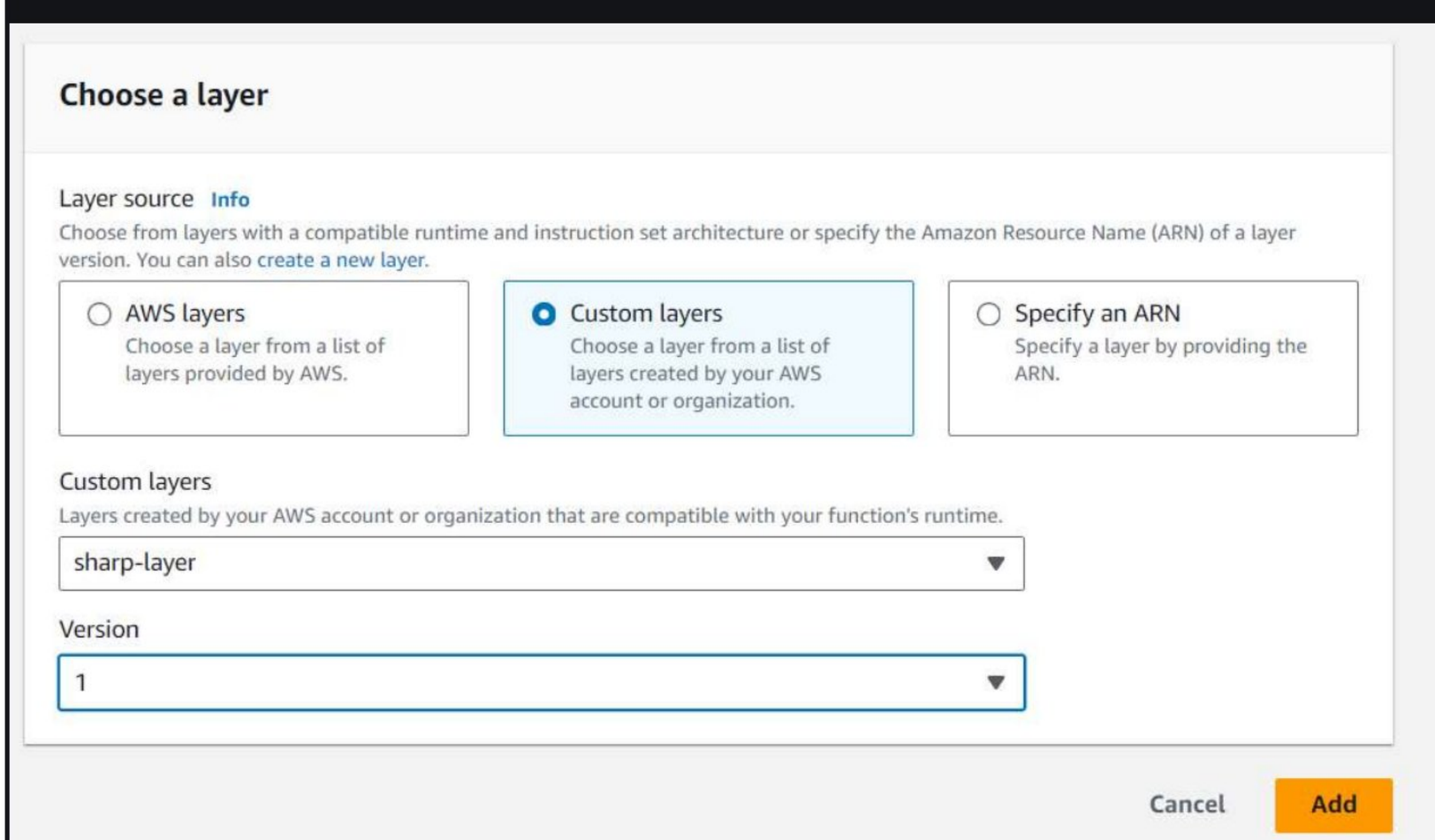
Scanned with OKEN Scanner

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



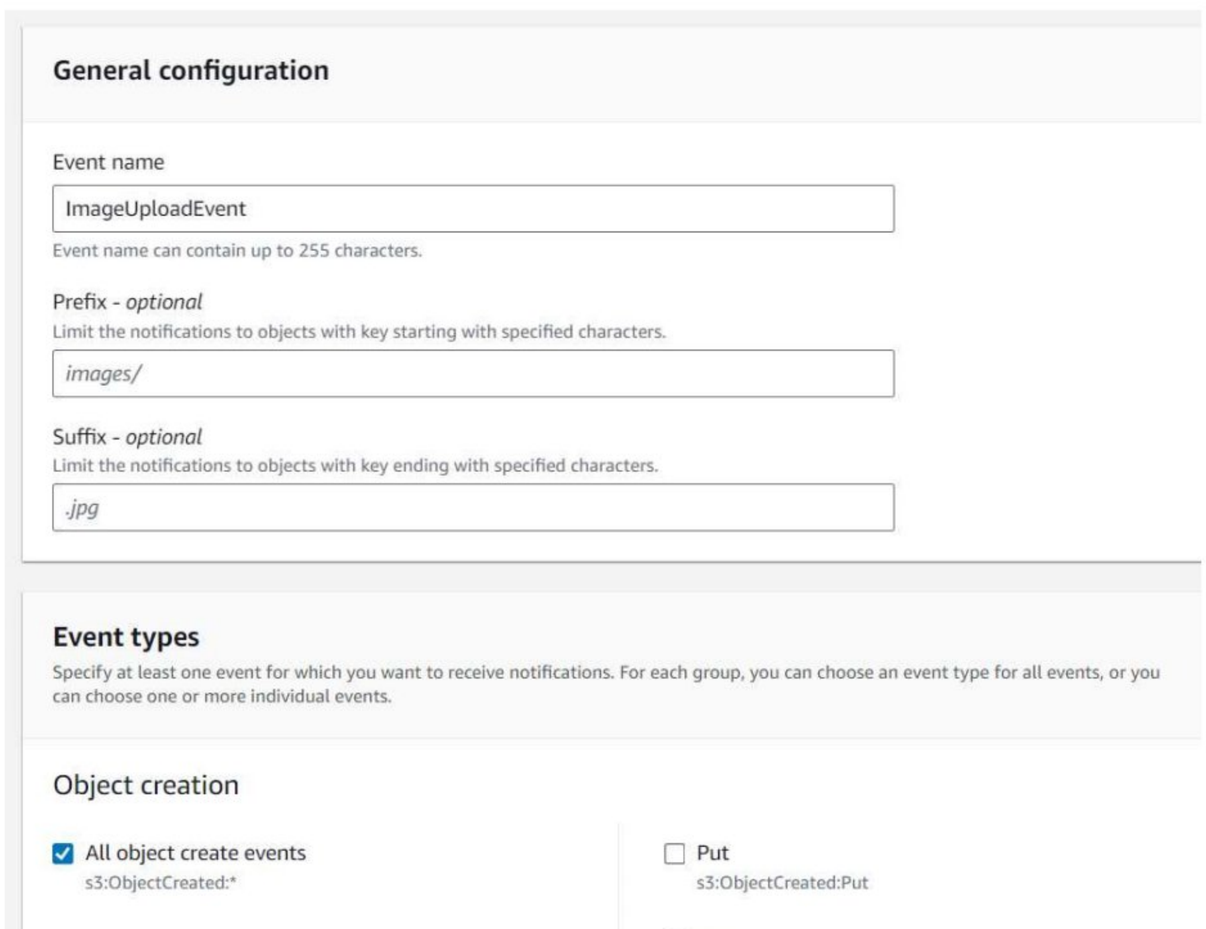
Scanned with OKEN Scanner

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.

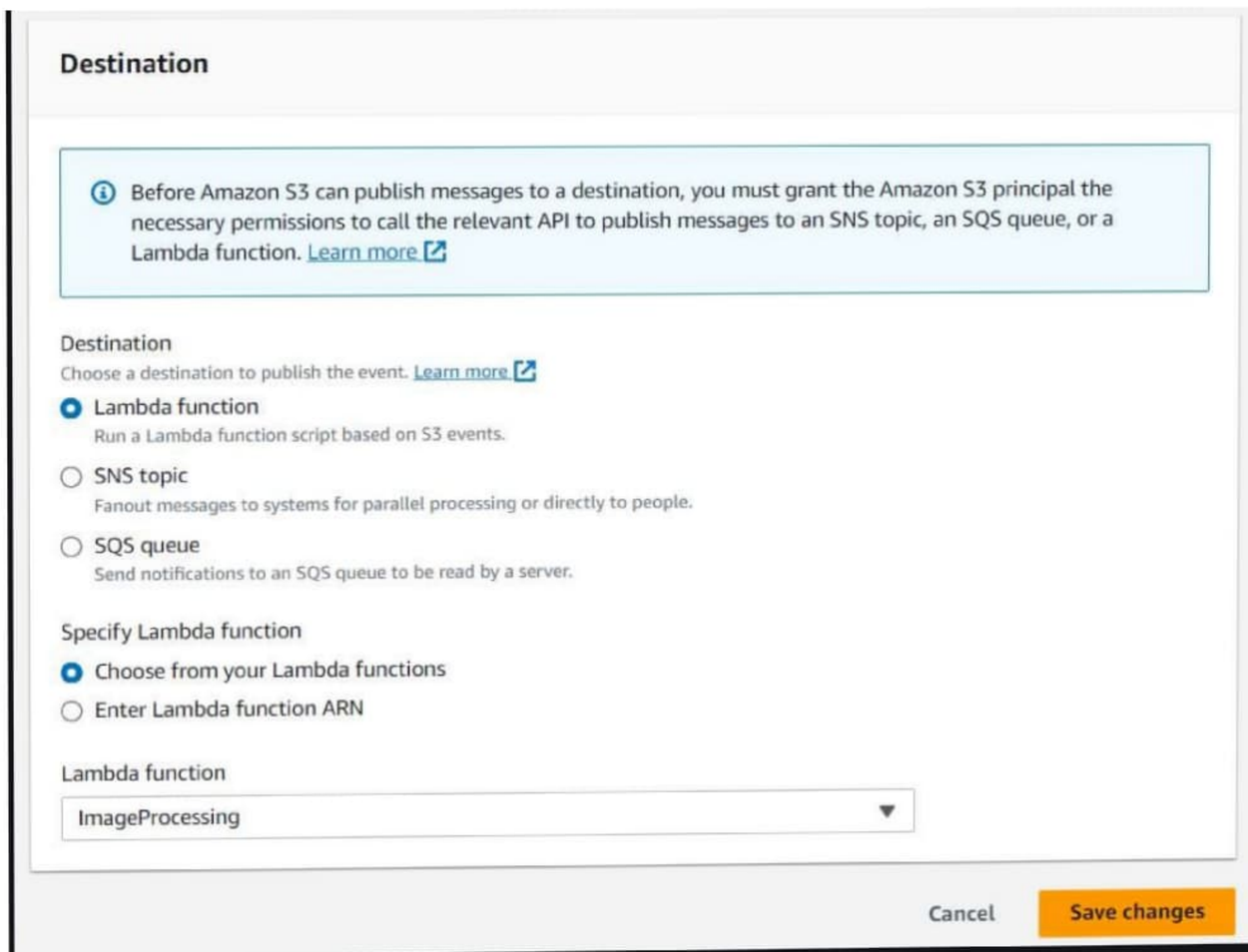
Go to S3 console. Select the source bucket ("serverless-bucket-uploaded-images"). Go to the Properties tab. Navigate to "Event Notifications". Click "Create Event Notifications".

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



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

Scanned with OKEN Scanner



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).

Congratulations, you just built a serverless Image processing application

Scanned with OKEN Scanner