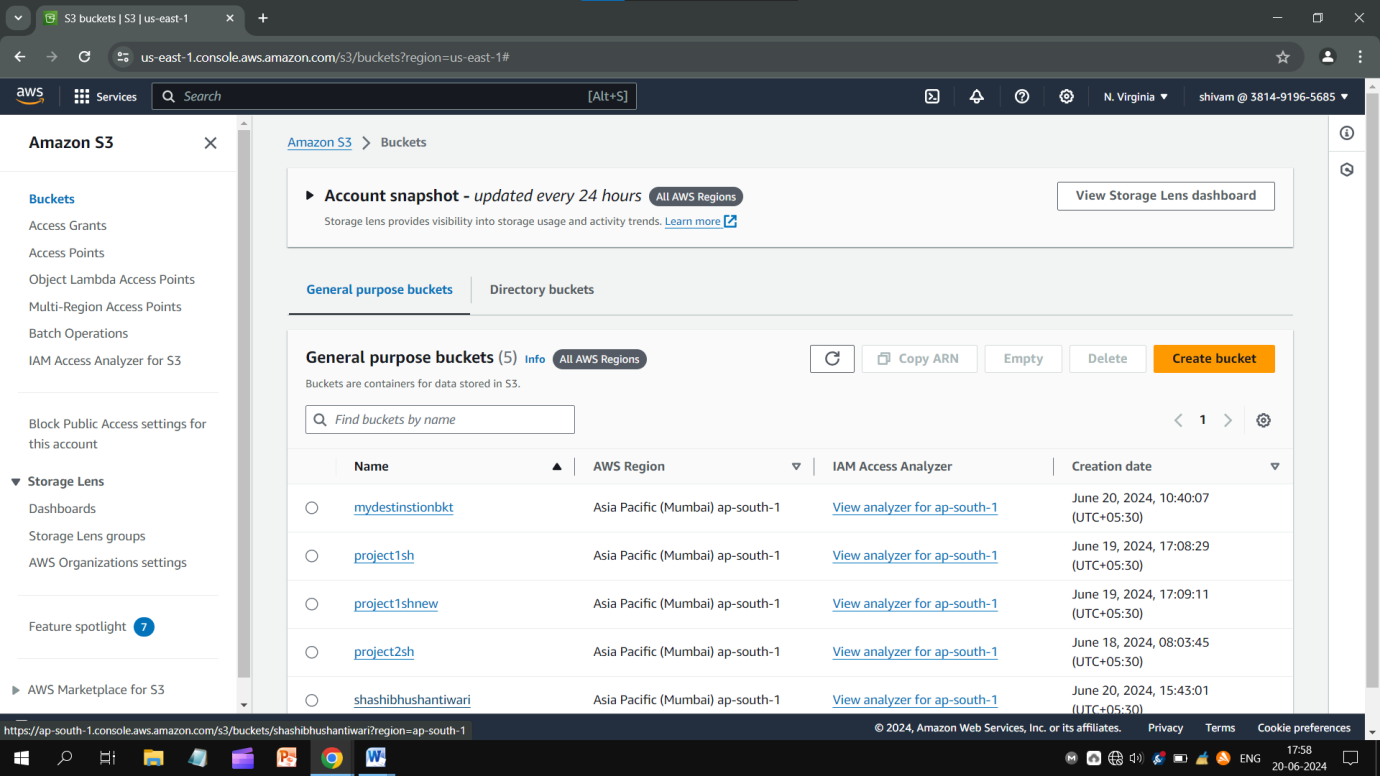
PROJECT – 1

**SERVERLESS IMAGE PROCESSING**

Object: Documentation of a serverless image processing application that automatically resizes and optimizes images uploaded to an Amazon S3 bucket.

* Log-in to AWS Management Console.
* Keep your location as set to “N.Virginia”.
* Go to the search bar and type “S3”.
* Go to the “Buckets” section and click on “Create bucket”.

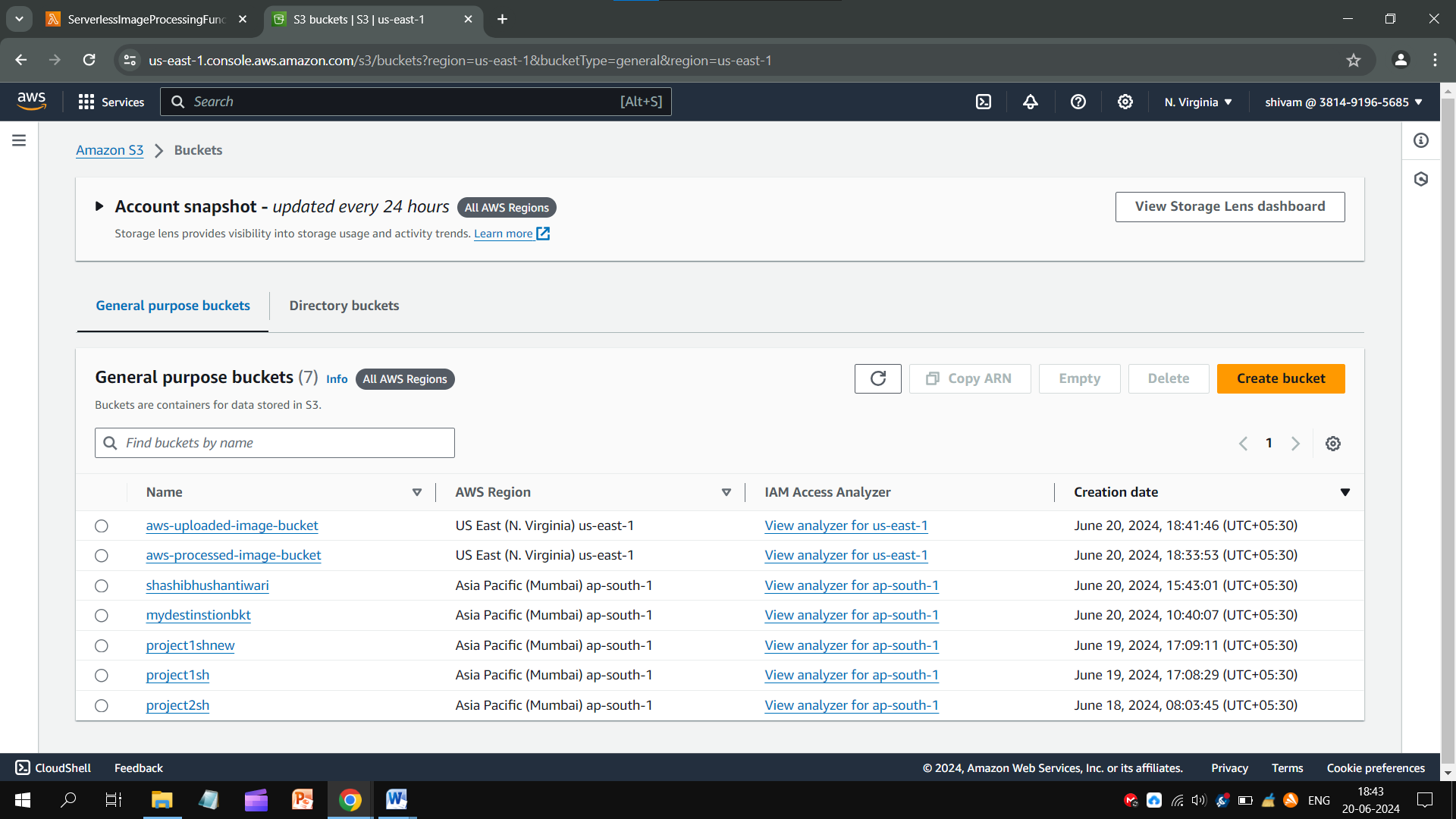


* Put a bucket a bucket name that should be unique, unblock “all public access” and let other settings be as they are.
* Just click on “Create bucket” and go ahead.....

Note:- In this way, you should have to create two buckets -

1. Source bucket to store the uploaded images.

2. Destination bucket to store the processed images.

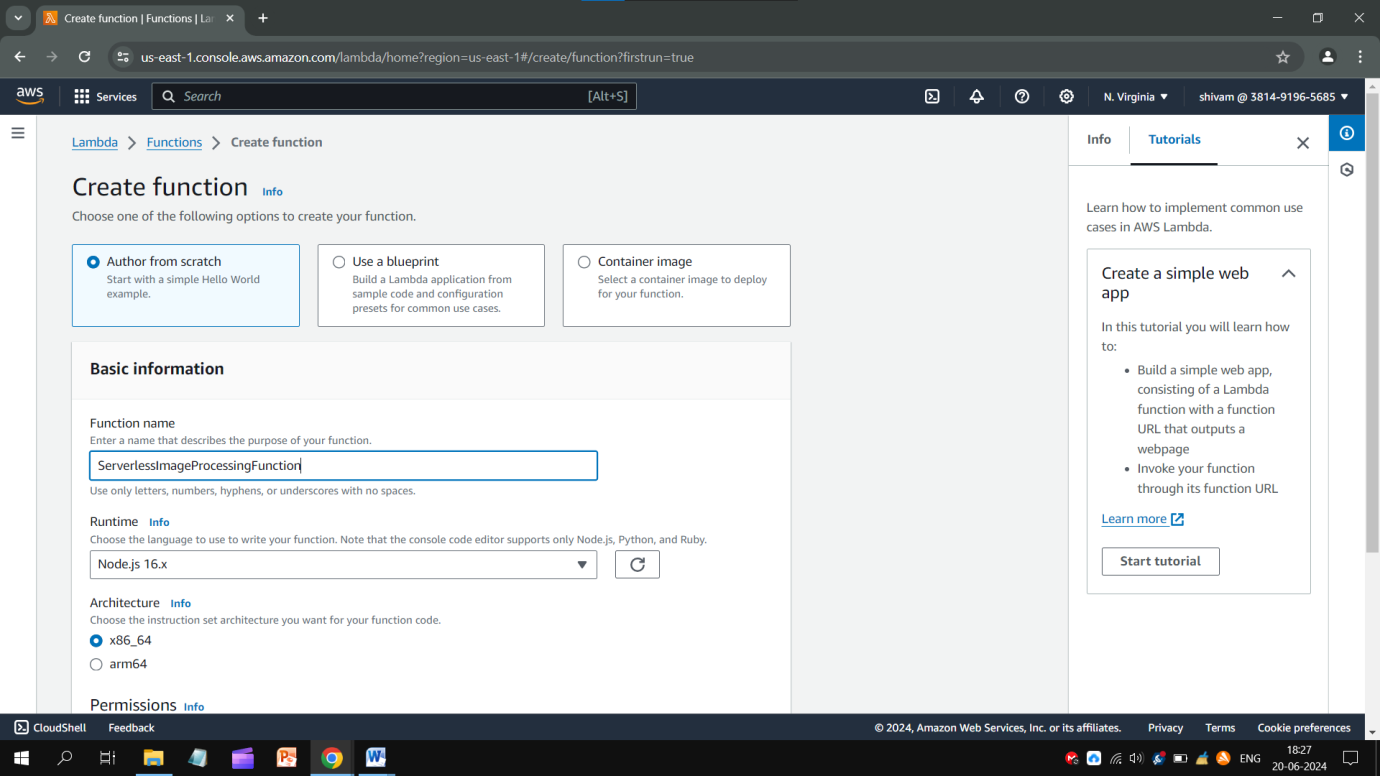


*The above image shows that “aws-uploaded-image-bucket” and “aws-processed-image-bucket” have been create.*

Creating a Lambda function

A serverless service, Lambda function, allows us to run our code virtually for any kind of application.

* Go to the search bar and type “Lambda”.
* Click on “Create function” and put a unique name.
* Change “Runtime” and leave other settings as the below image shows.
* Go for the “Create function” bottom.



Here is given a 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),

};

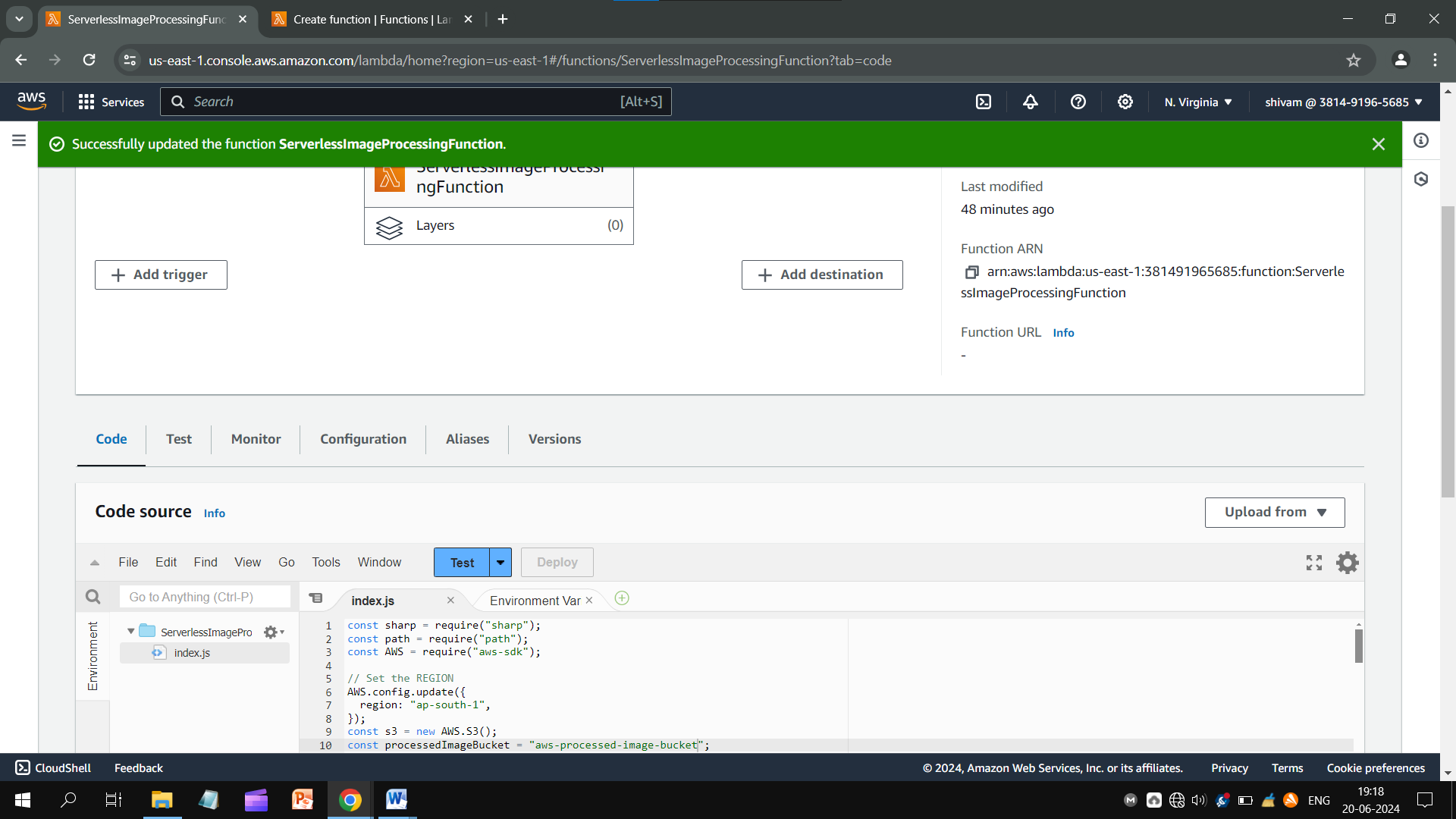
let result2 = await s3.putObject(coverphotoImageParam).promise();

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

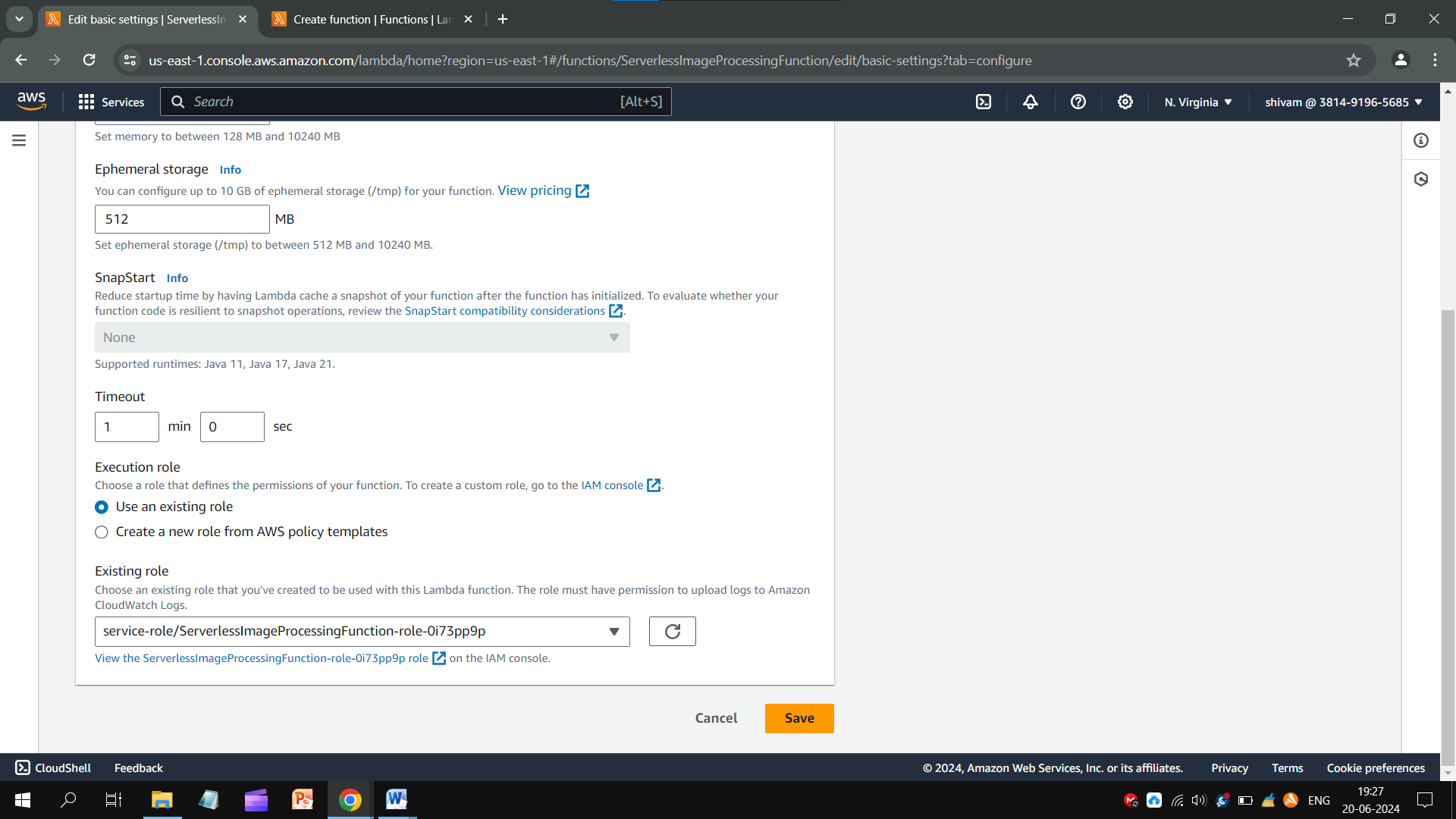
}

};

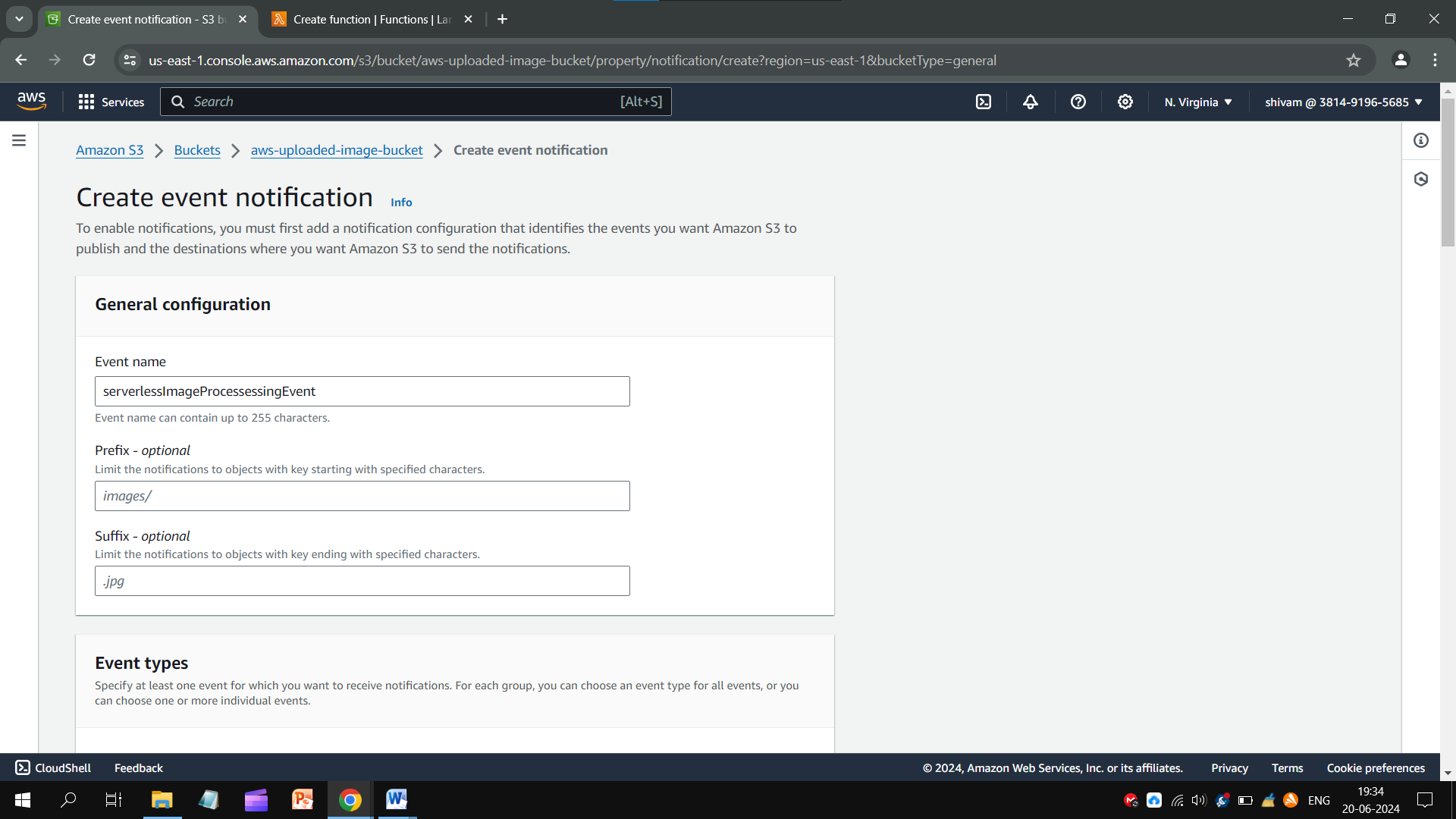
* Copy it and paste to the code editor on function page.
* And change the option “serverless-bucket-processed-images” to the name of the destination bucket name, now click on “deploy” to get it done.

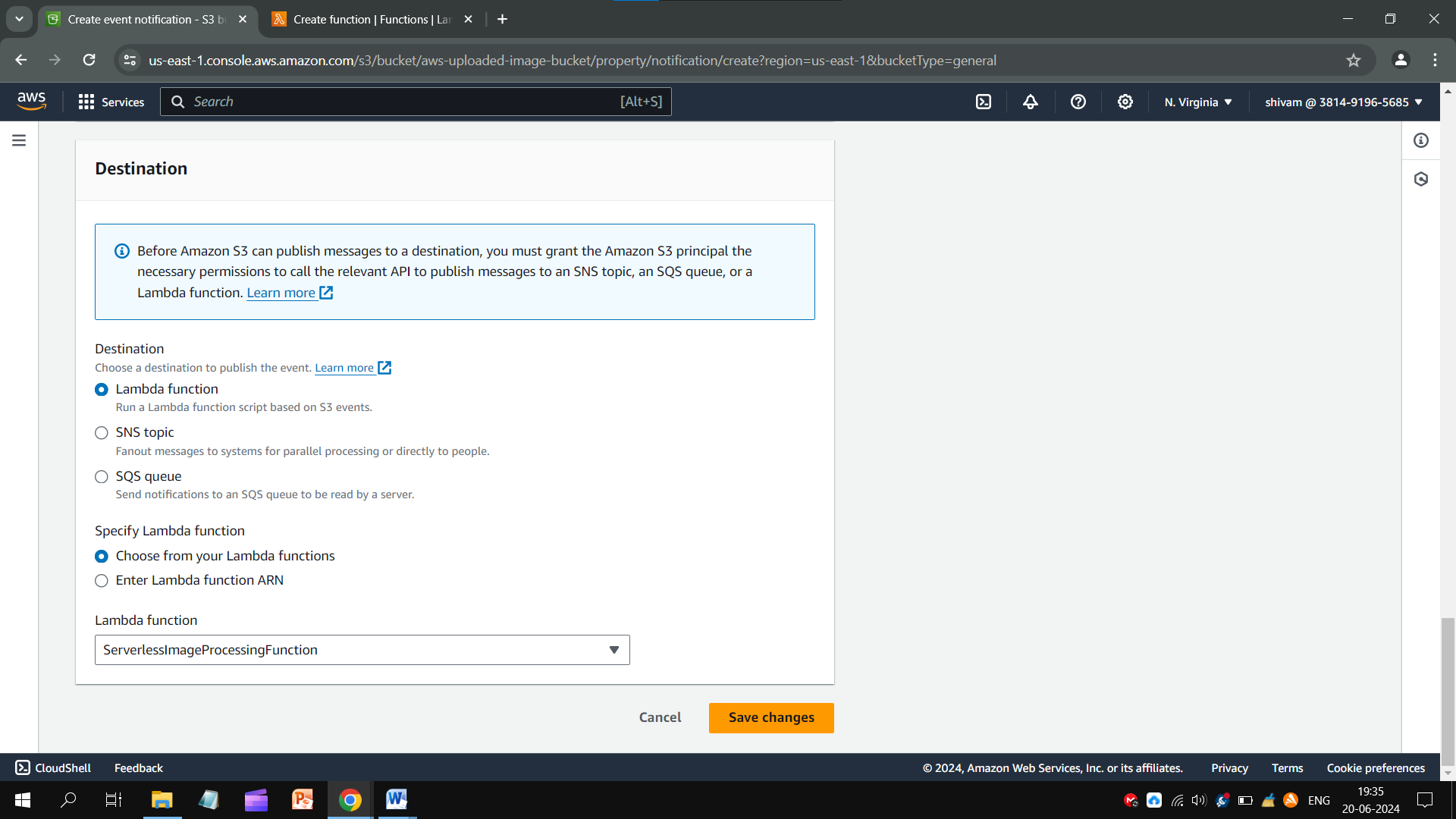


* Open the “Configuration” section.
* Change the “Timeout” to 1 min 0 sec or more than this. B’cause user’s image might take time in uploading.
* Select the “Existing role”and “Save” it.



* Go to Amazon S3 > **Buckets** and open Source Bucket.
* Get into Properties tab and scroll down.
* Find the **Event notifications** and Create an event.
* Select a unique name, fill the followings as the below photos show.





Now test this application by uploading an image. Remember that two files will appear – one is thumbnail and next is cover photo, according to the code.

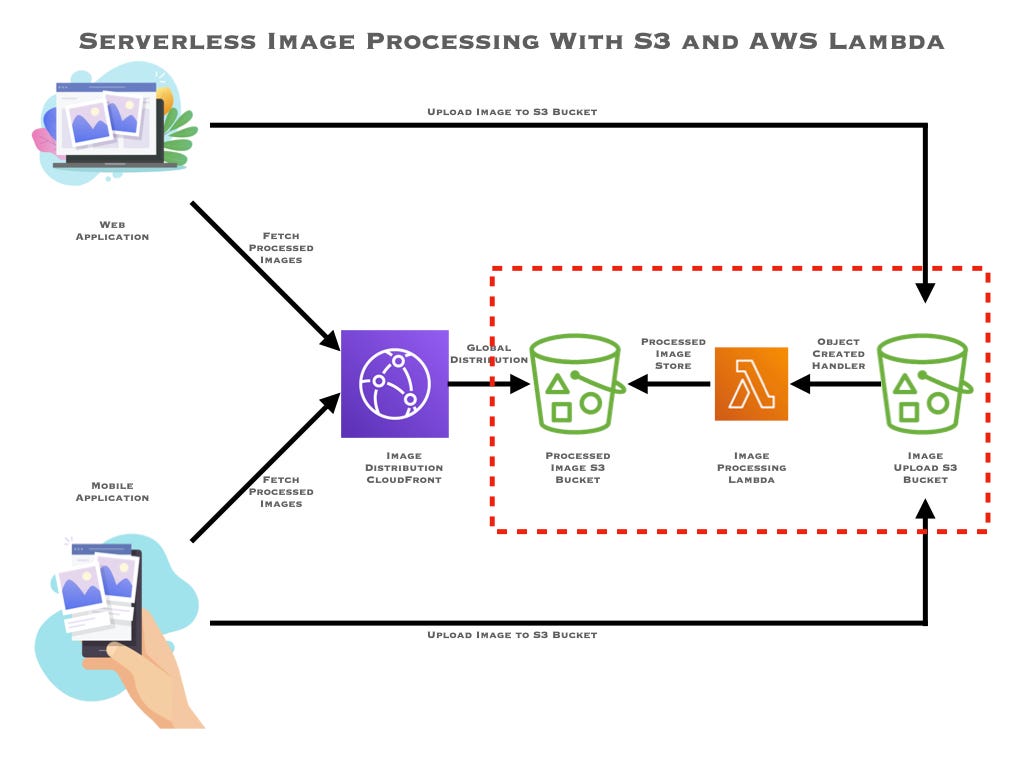
Thank You!

**Conclusion:** In this project we learn to create an adaptable and modular serveless image processing application that automatically resizes and optimizes the image is uploaded. Also it provides a better way to handle image uploading and processing.

Source of Info : <https://www.geeksforgeeks.org/serverless-image-processing-with-aws-lambda-and-s3/>

Source of Image: <https://sanjay-dandekar.medium.com/aws-knowledge-series-serverless-image-processing-with-s3-and-aws-lambda-ad6ad08f829b>

How it works?

 When a user shares or uploads an image file to the AWS S3. Through the Lambda function, it’s automatically resized and optimized with the given resolution in code.

Very first, the images gets uplaoaded to the Uploading Image Bucket (Source Bucket). The Lambda function starts processing the image. When processed, it goes to the Processing Image Bucket (Destination Bucket). After this process, that image gets saved in cloud storage of AWS. By Image Distribution CloudFront or using public URL, we can get/download the image.