**DR. VIRENDRA SWARUP INSTITUTE OF COMPUTER STUDIES**

**Project 1**

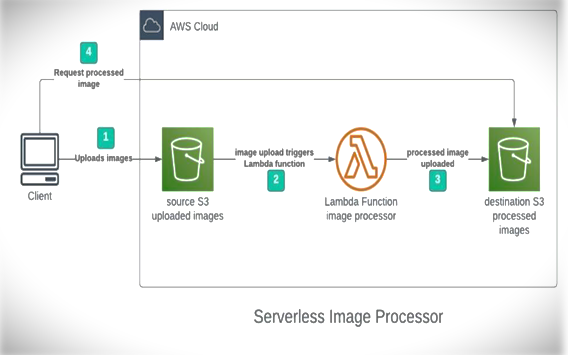
**Serverless Image Processing**

**SUBMITTED BY:- SUBMITTED TO:-**

**Infinity Storage Team ABHINAV SIR**

**Serverless Image Processing**

The Serverless Image Handler solution helps you embed images on your websites and mobile applications to drive user engagement. It uses the [sharp](https://sharp.pixelplumbing.com/en/stable/) Node.js library to provide high-speed image processing without sacrificing image quality. To minimize your costs of image optimization, manipulation, and processing, this solution automates version control and provides flexible storage and compute options for file reprocessing

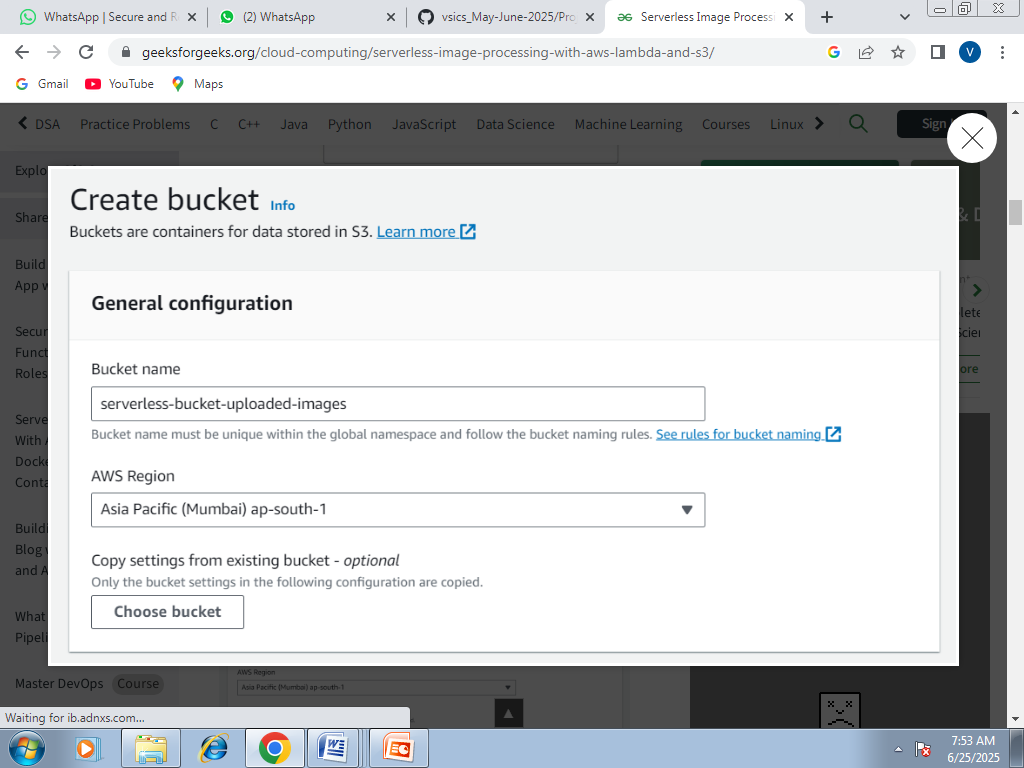
****

**Step 1 - Creating S3 buckets**

We will use two S3 buckets:

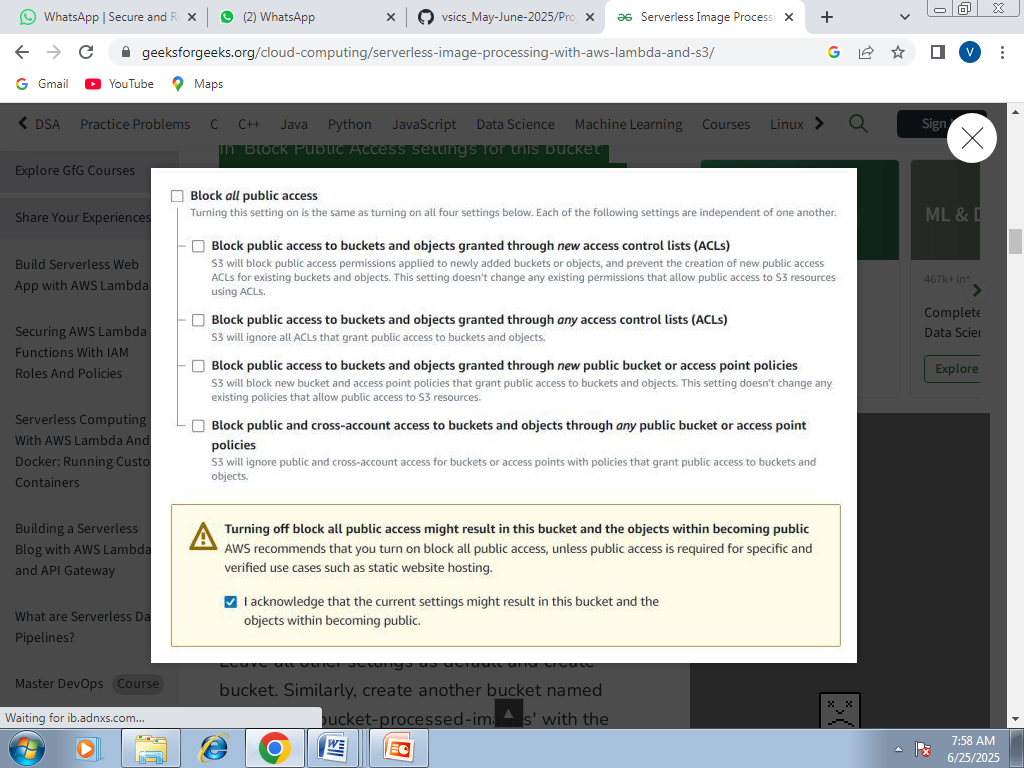
1. **source Bucket:** For storing uploaded images.
2. **destination Bucket:** For storing processed images.

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)**.



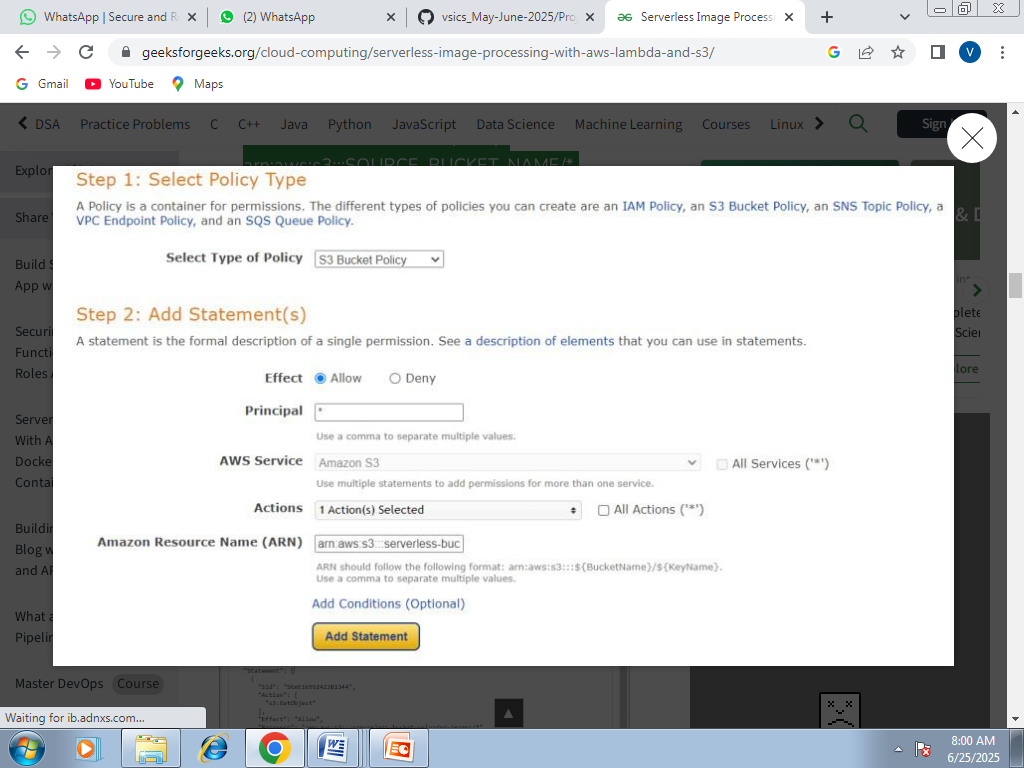
Leave all other settings as default and create bucket. Similarly, create another bucket named 'serverless-bucket-processed-images' with the same region. This bucket will be used to store the processed images. Although we enabled public access while creating the buckets, we still need to attach a bucket 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.

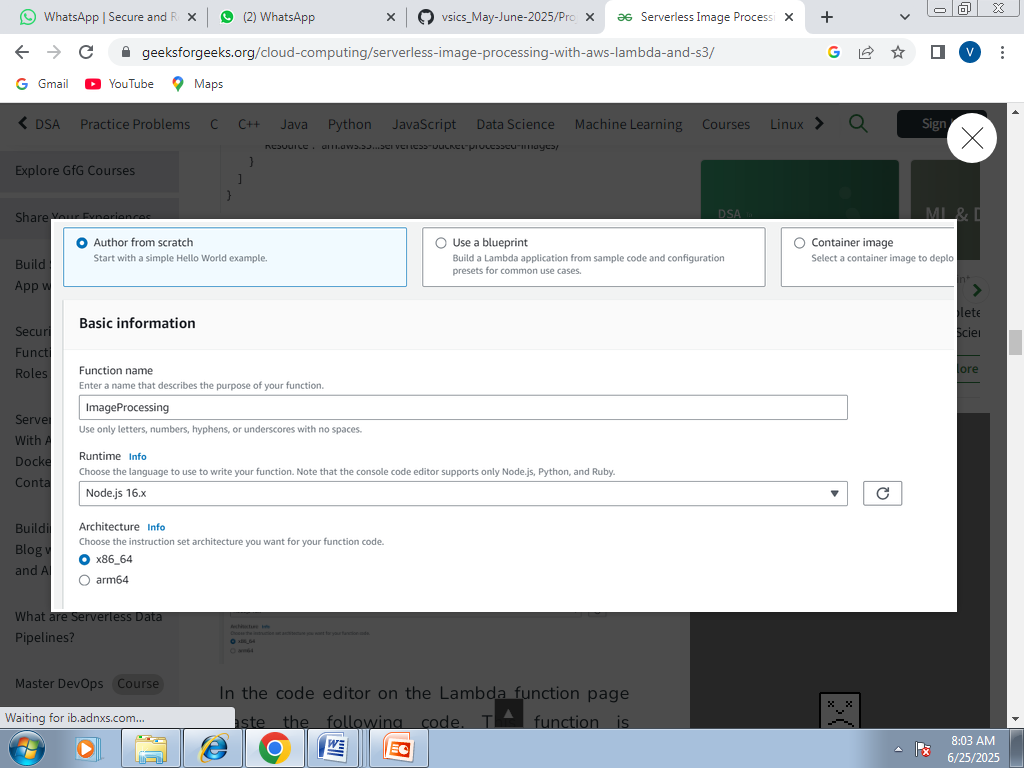
Follow same steps to attach a policy to the processed images S3 bucket. The policy settings for destination bucket are:

* 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 = requir

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();

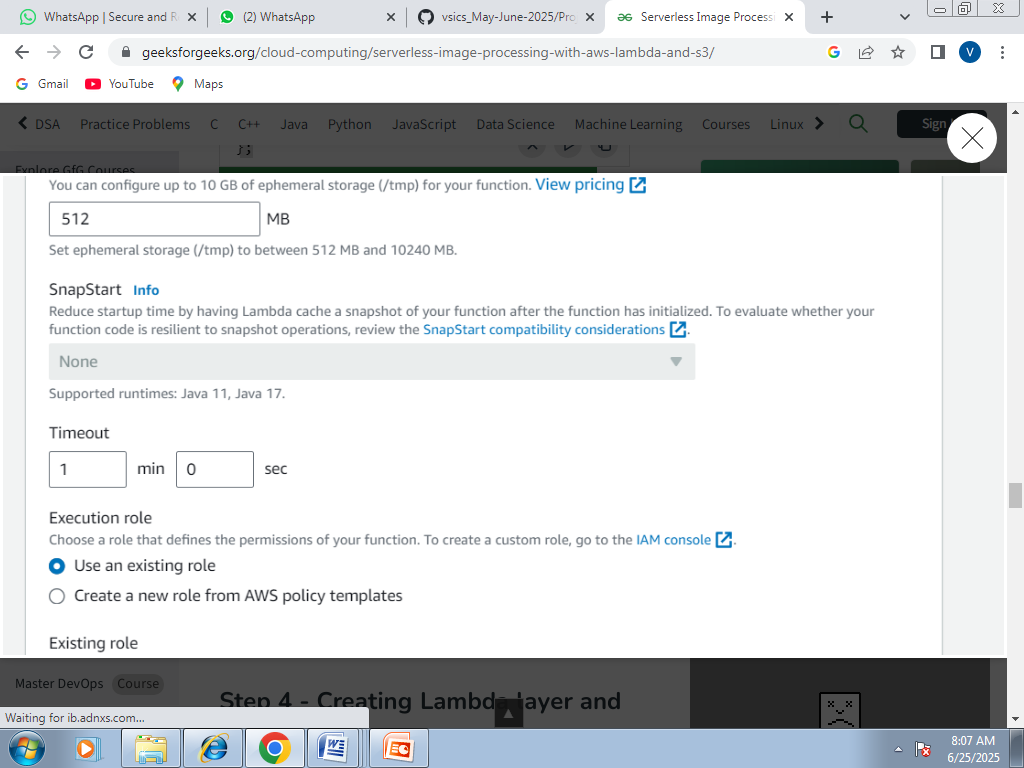
console.log("coverphoto image uploaded:" + JSON.stringify(result2));

}

};

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.



## 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.

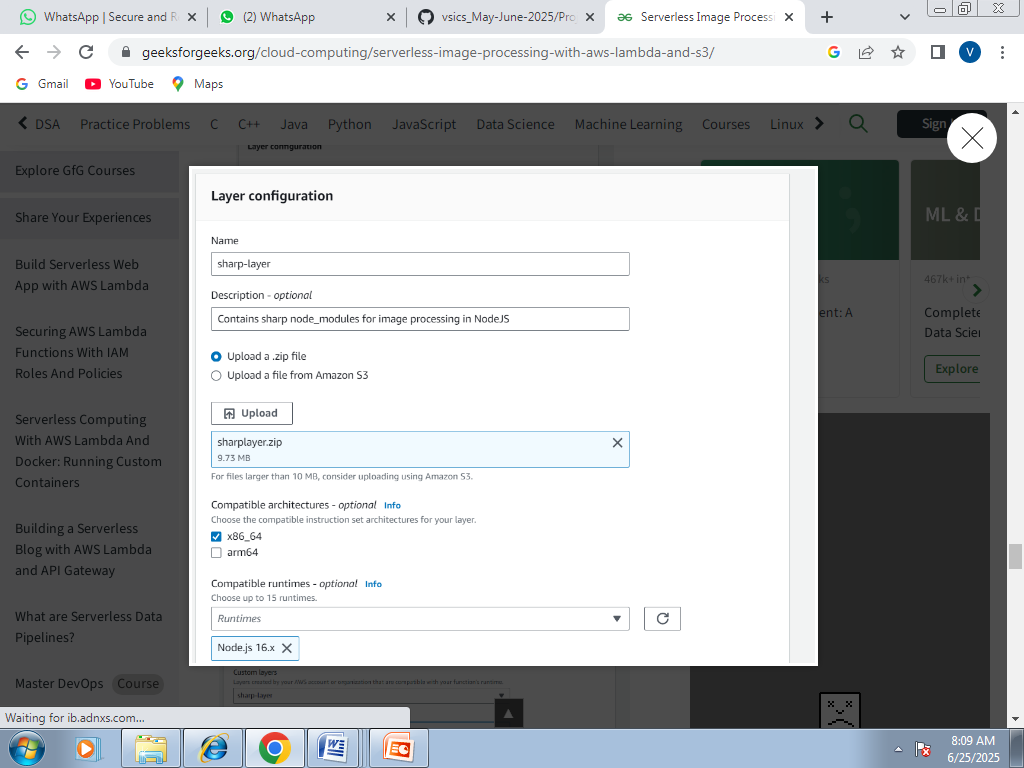
First we need to create a zip file with all the dependencies (node modules in our case) required by our Lambda function.

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).

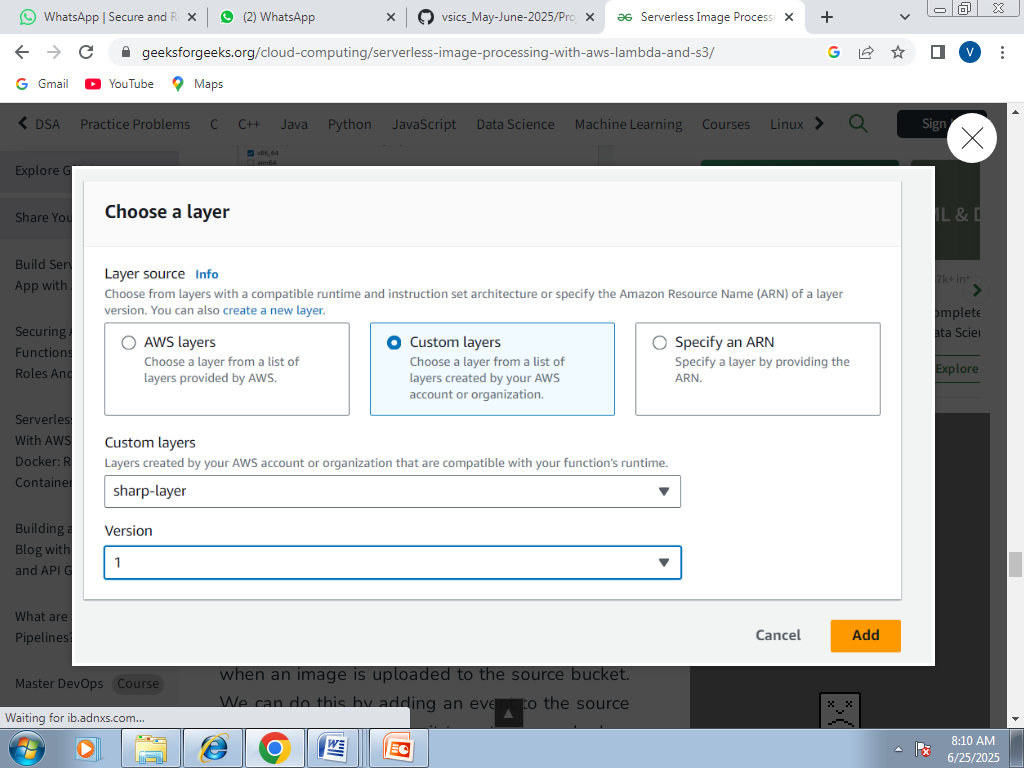
npm install --arch=x64 --platform=linux sharp

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

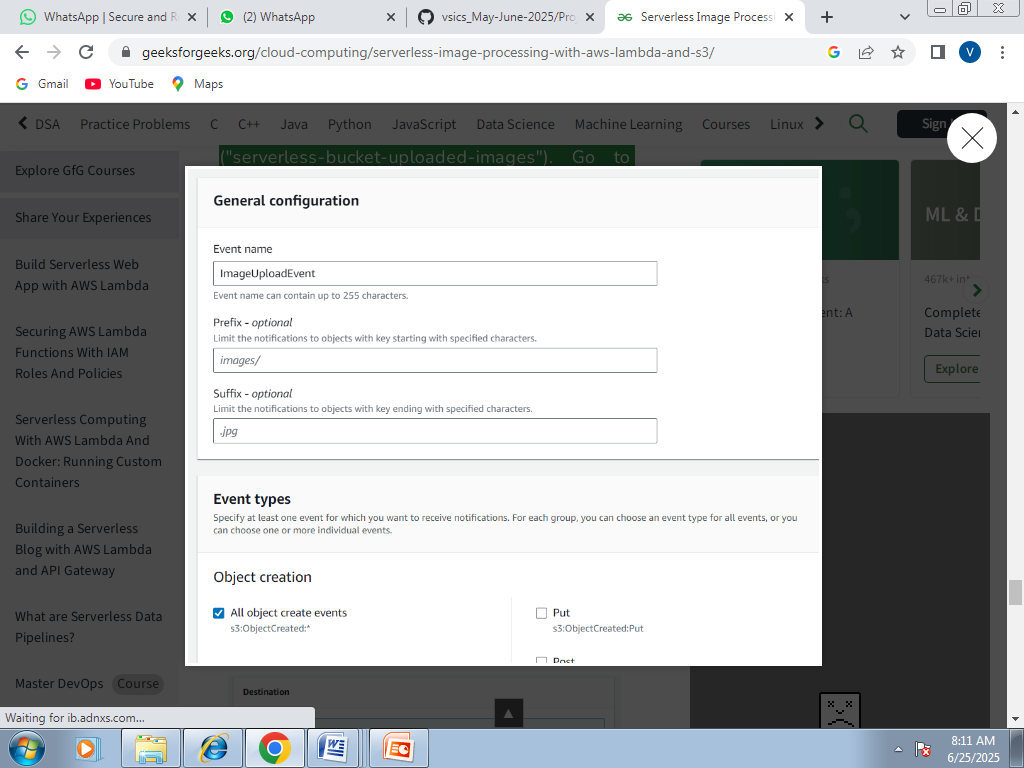


## Step 5 - Creating S3 trigger

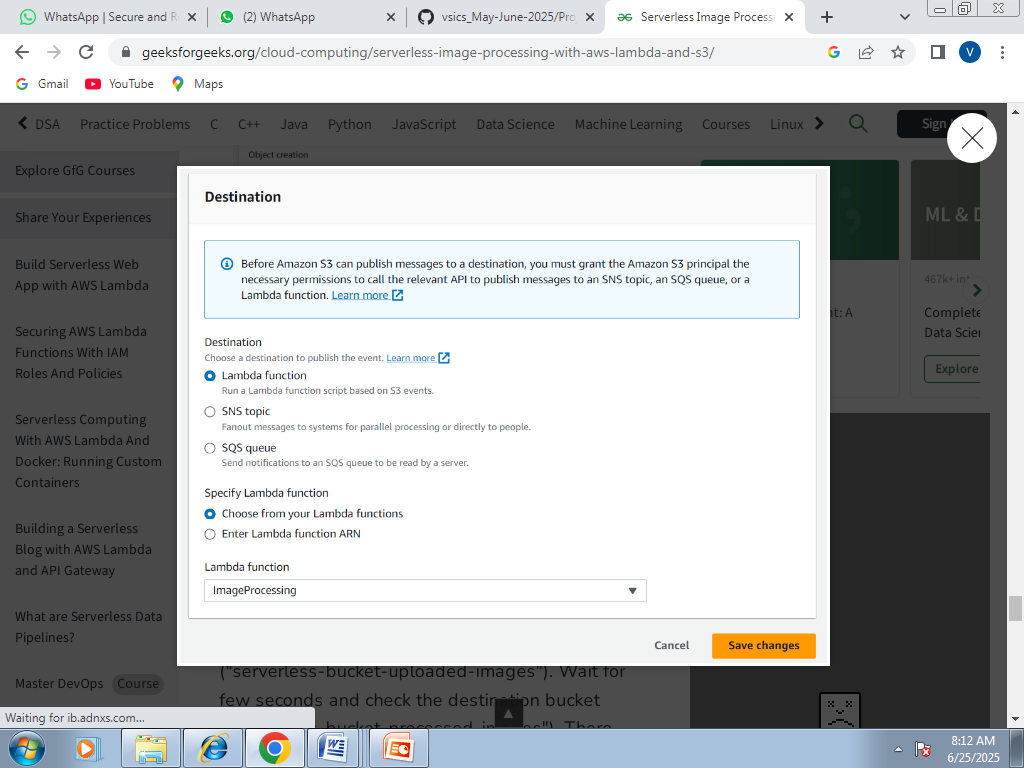
Now we need our Lambda function to know when an image is uploaded to the source bucket. We can do this by adding an event to the source S3 bucket and configure it to get triggered when an image is uploaded to the bucket which in turn invokes the Lambda function.

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 cover photo).

.

**PROJECT-2**

**TOPIC-DEPLOY A STATIC WEBSITE**

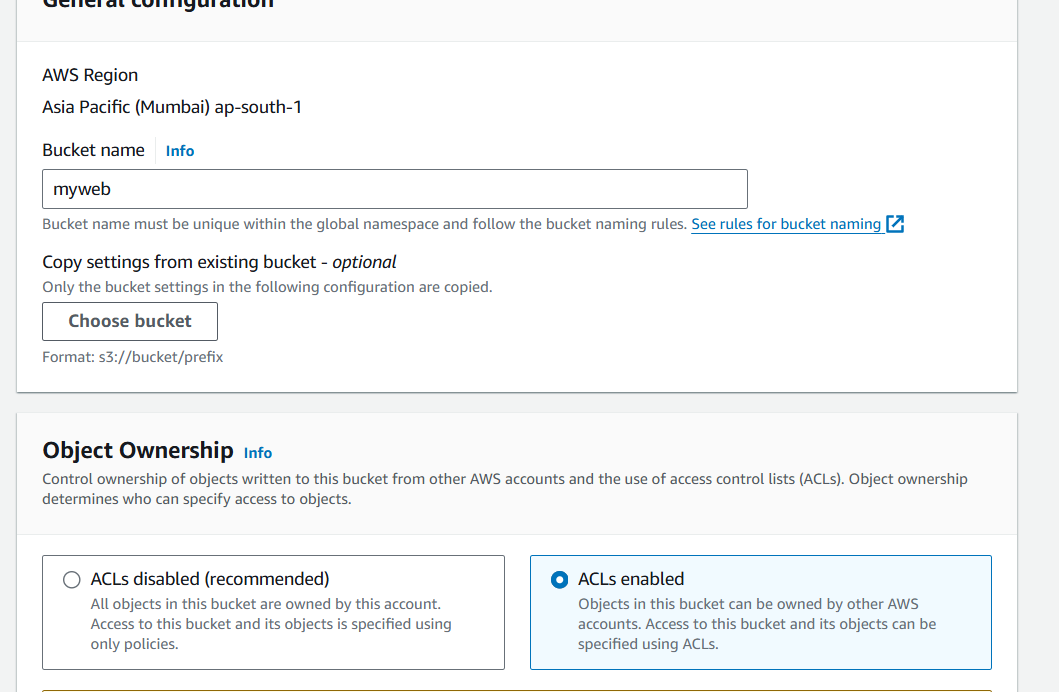
**Topics**

* [Step 1: Create a bucket](https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html#step1-create-bucket-config-as-website)
* [Step 2: Enable static website hosting](https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html#step2-create-bucket-config-as-website)
* [Step 3: Edit Block Public Access settings](https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html#step3-edit-block-public-access)
* [Step 4: Add a bucket policy that makes your bucket content publicly available](https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html#step4-add-bucket-policy-make-content-public)
* [Step 5: Configure an index document](https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html#step5-upload-index-doc)
* [Step 6: Configure an error document](https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html#step6-upload-error-doc)
* [Step 7: Test your website endpoint](https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html#step7-test-web-site)
* [Step 8: Clean up](https://docs.aws.amazon.com/AmazonS3/latest/userguide/HostingWebsiteOnS3Setup.html#getting-started-cleanup-s3-website-overview)

**Step 1: Create a bucket**

The following instructions provide an overview of how to create your buckets for website hosting. For detailed, step-by-step instructions on creating a bucket.

**To create a bucket**

****

1. Sign in to the AWS Management Console and open the Amazon S3 console .
2. Choose **Create bucket**.
3. Enter the **Bucket name** (for example, **webbucket**).
4. Choose the Region where you want to create the bucket.

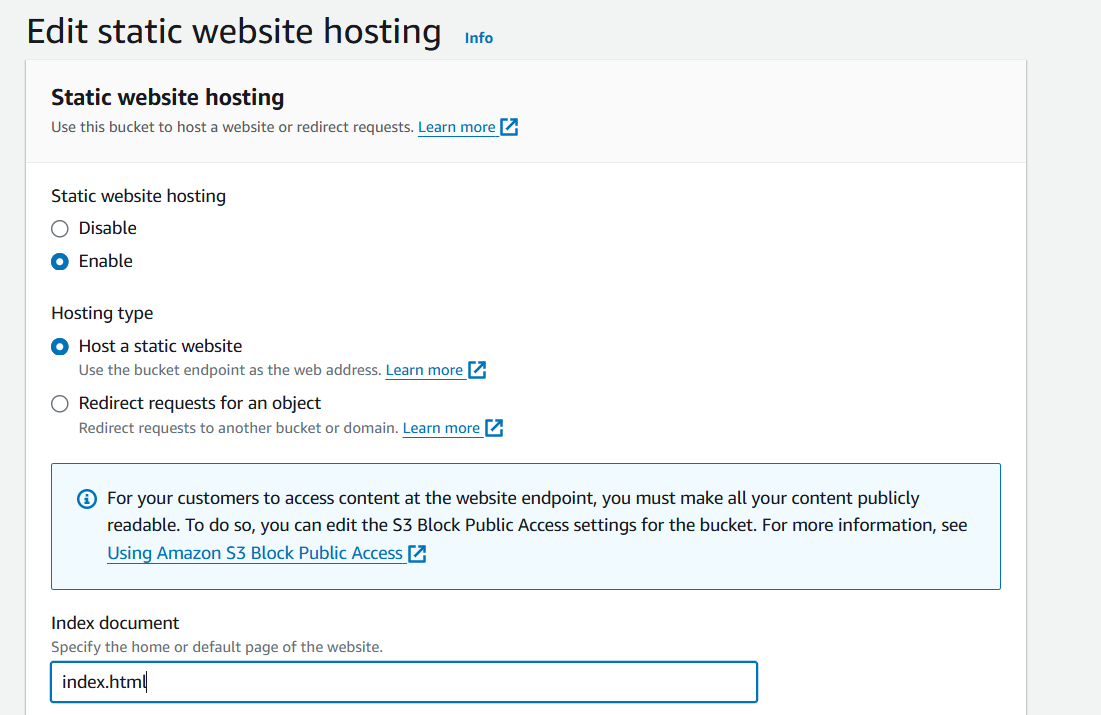
Choose a Region that is geographically close to you to minimize latency and costs, or to address regulatory requirements. The Region that you choose determines your Amazon S3 website endpoint.

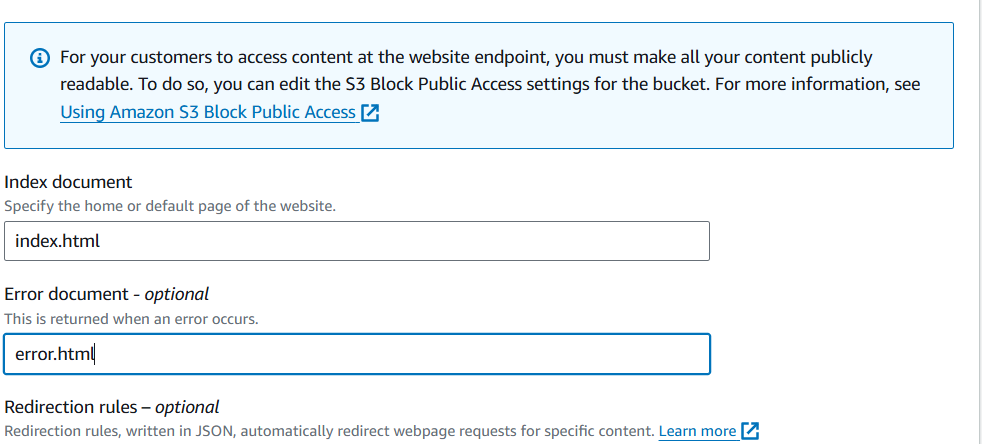
1. To accept the default settings and create the bucket, choose **Create**.

**Step 2: Enable static website hosting**

After you create a bucket, you can enable static website hosting for your bucket.

**To enable static website hosting**

****

****

1. Sign in to the AWS Management Console and open the Amazon S3 console .
2. In the **Buckets** list, choose the name of the bucket that you want to enable static website hosting for.
3. Choose **Properties**.
4. Under **Static website hosting**, choose **Edit**.
5. Choose **Use this bucket to host a website**.
6. Under **Static website hosting**, choose **Enable**.
7. In **Index document**, enter the file name of the index document, typically index.html.

The index document name is case sensitive and must exactly match the file name of the HTML index document that you plan to upload to your S3 bucket. When you configure a bucket for website hosting, you must specify an index document. Amazon S3 returns this index document when requests are made to the root domain or any of the subfolders.

1. (Optional) If you want to specify advanced redirection rules, in **Redirection rules**, enter JSON to describe the rules.

For example, you can conditionally route requests according to specific object key names or prefixes in the request.

1. Choose **Save changes**.

Amazon S3 enables static website hosting for your bucket. At the bottom of the page, under **Static website hosting**, you see the website endpoint for your bucket.

1. Under **Static website hosting**, note the **Endpoint**.

The **Endpoint** is the Amazon S3 website endpoint for your bucket. After you finish configuring your bucket as a static website, you can use this endpoint to test your website.

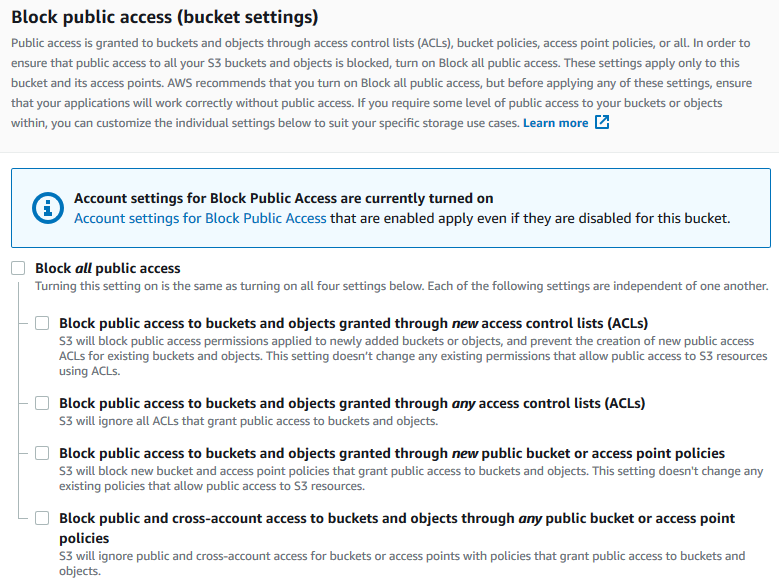
**Step 3: Edit Block Public Access settings**

By default, Amazon S3 blocks public access to your account and buckets. If you want to use a bucket to host a static website, you can use these steps to edit your block public access settings.

1. Open the Amazon S3 console at .
2. Choose the name of the bucket that you have configured as a static website.
3. Choose **Permissions**.
4. Under **Block public access (bucket settings)**, choose **Edit**.
5. Clear **Block *all* public access**, and choose **Save changes**.

**Warning**

Before you complete this step, review [Blocking public access to your Amazon S3 storage](https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-block-public-access.html) to ensure you understand and accept the risks involved with allowing public access. When you turn off block public access settings to make your bucket public, anyone on the internet can access your bucket. We recommend that you block all public access to your buckets.



Amazon S3 turns off Block Public Access settings for your bucket. To create a public, static website, you might also have to [edit the Block Public Access settings](https://docs.aws.amazon.com/AmazonS3/latest/user-guide/block-public-access-account.html) for your account before adding a bucket policy.

**Step 4: Add a bucket policy that makes your bucket content publicly available**

After you edit S3 Block Public Access settings, you can add a bucket policy to grant public read access to your bucket. When you grant public read access, anyone on the internet can access your bucket.

**Important**

The following policy is an example only and allows full access to the contents of your bucket.

1. Under **Buckets**, choose the name of your bucket.
2. Choose **Permissions**.
3. Under **Bucket Policy**, choose **Edit**.
4. To grant public read access for your website, copy the following bucket policy, and paste it in the **Bucket policy editor**.
5. {
6. "Version": "2012-10-17",
7. "Statement": [
8. {
9. "Sid": "PublicReadGetObject",
10. "Effect": "Allow",
11. "Principal": "\*",
12. "Action": [
13. "s3:GetObject"
14. ],
15. "Resource": [
16. "arn:aws:s3:::*Bucket-Name*/\*"
17. ]
18. }
19. ]

}

1. Update the Resource to your bucket name.

In the preceding example bucket policy, *Bucket-Name* is a placeholder for the bucket name. To use this bucket policy with your own bucket, you must update this name to match your bucket name.

1. Choose **Save changes**.

A message appears indicating that the bucket policy has been successfully added.

If you see an error that says Policy has invalid resource, confirm that the bucket name in the bucket policy matches your bucket name.

If you get an error message and cannot save the bucket policy, check your account and bucket Block Public Access settings to confirm that you allow public access to the bucket.

**Step 5: Configure an index document**

When you enable static website hosting for your bucket, you enter the name of the index document (for example, **index.html**). After you enable static website hosting for the bucket, you upload an HTML file with this index document name to your bucket.

**To configure the index document**

1. Create an index.html file.

If you don't have an index.html file, you can use the following HTML to create one:

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8">
5. <meta name="viewport" content="width=device-width, initial-scale=1.0">
6. <title>VSICS Kanpur - Home</title>
7. <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
8. <link rel="stylesheet" href="styles/main.css">
9. <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/animate.css/4.1.1/animate.min.css">
10. <style>
11. .hero-section {
12. background: linear-gradient(135deg, #0f2027, #203a43, #2c5364);
13. color: white;
14. padding: 80px 20px;
15. text-align: center;
16. perspective: 1000px;
17. }
18. .hero-section h1 {
19. font-size: 3rem;
20. transform: rotateY(10deg);
21. transition: transform 0.3s ease-in-out;
22. }
23. .hero-section h1:hover {
24. transform: rotateY(0deg) scale(1.05);
25. }
26. .card-3d {
27. transform-style: preserve-3d;
28. transition: transform 0.5s;
29. }
30. .card-3d:hover {
31. transform: rotateY(10deg) scale(1.02);
32. }
33. .feature-icon {
34. font-size: 3rem;
35. margin-bottom: 10px;
36. }
37. </style>
38. </head>
39. <body>
40. <header class="bg-light py-3 border-bottom shadow">
41. <div class="container d-flex justify-content-between align-items-center">
42. <img src="img/logo.png" alt="VSICS Logo" height="80">
43. <div class="text-end">
44. <h1 class="h4 text-primary">Dr. Virendra Swarup Institute of Computer Studies</h1>
45. <small class="text-muted">Affiliated to CSJM University, Kanpur</small>
46. </div>
47. </div>
48. </header>
49. <nav class="navbar navbar-expand-lg navbar-dark bg-dark sticky-top shadow">
50. <div class="container">
51. <a class="navbar-brand" href="index.html">VSICS</a>
52. <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">
53. <span class="navbar-toggler-icon"></span>
54. </button>
55. <div class="collapse navbar-collapse" id="navbarNav">
56. <ul class="navbar-nav ms-auto">
57. <li class="nav-item"><a class="nav-link active" href="index.html">Home</a></li>
58. <li class="nav-item"><a class="nav-link" href="about.html">About</a></li>
59. <li class="nav-item"><a class="nav-link" href="courses.html">Courses</a></li>
60. <li class="nav-item"><a class="nav-link" href="departments.html">Departments</a></li>
61. <li class="nav-item"><a class="nav-link" href="placement.html">Placements</a></li>
62. <li class="nav-item"><a class="nav-link" href="gallery.html">Gallery</a></li>
63. <li class="nav-item"><a class="nav-link" href="news.html">News</a></li>
64. <li class="nav-item"><a class="nav-link" href="contact.html">Contact</a></li>
65. </ul>
66. </div>
67. </div>
68. </nav>
69. <section class="hero-section">
70. <div class="container">
71. <h1 class="animate\_\_animated animate\_\_fadeInDown">Welcome to VSICS Kanpur</h1>
72. <p class="lead animate\_\_animated animate\_\_fadeInUp">Empowering students with futuristic education in Technology, Management & Commerce</p>
73. <a href="about.html" class="btn btn-outline-light mt-3">Explore More</a>
74. </div>
75. </section>
76. <main class="container py-5">
77. <h2 class="text-center mb-5">Our Highlights</h2>
78. <div class="row text-center">
79. <div class="col-md-4">
80. <div class="card card-3d p-4 mb-4 shadow">
81. <div class="feature-icon text-primary">
82. 💡
83. </div>
84. <h4>Innovative Learning</h4>
85. <p>Modern labs, industry projects, and live sessions with professionals to make learning future-ready.</p>
86. </div>
87. </div>
88. <div class="col-md-4">
89. <div class="card card-3d p-4 mb-4 shadow">
90. <div class="feature-icon text-success">
91. 🚀
92. </div>
93. <h4>Career Focus</h4>
94. <p>Placement-oriented training and tie-ups with leading firms ensure a strong professional launchpad.</p>
95. </div>
96. </div>
97. <div class="col-md-4">
98. <div class="card card-3d p-4 mb-4 shadow">
99. <div class="feature-icon text-danger">
100. 🌐
101. </div>
102. <h4>Global Outlook</h4>
103. <p>Exposure to global trends, digital platforms, and seminars to keep students industry-aligned.</p>
104. </div>
105. </div>
106. </div>
107. </main>
108. <footer class="bg-dark text-white text-center py-3">
109. <p class="mb-0">&copy; 2025 Dr. Virendra Swarup Institute of Computer Studies, Kanpur. All Rights Reserved.</p>
110. </footer>
111. <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
112. <script src="js/script.js"></script>
113. </body>
114. </html>
115. Save the index file locally.

The index document file name must exactly match the index document name that you enter in the **Static website hosting** dialog box. The index document name is case sensitive. For example, if you enter index.html for the **Index document** name in the **Static website hosting** dialog box, your index document file name must also be index.html and not Index.html.

1. Sign in to the AWS Management Console and open the Amazon S3 console .
2. In the **Buckets** list, choose the name of the bucket that you want to use to host a static website.
3. Enable static website hosting for your bucket, and enter the exact name of your index document (for example, index.html).

After enabling static website hosting, proceed to step 6.

1. To upload the index document to your bucket, do one of the following:
   * Drag and drop the index file into the console bucket listing.
   * Choose **Upload**, and follow the prompts to choose and upload the index file.
2. (Optional) Upload other website content to your bucket.

**Step 6: Configure an error document**

When you enable static website hosting for your bucket, you enter the name of the error document (for example, **404.html**). After you enable static website hosting for the bucket, you upload an HTML file with this error document name to your bucket.

**To configure an error document**

1. Create an error document, for example 404.html.
2. Save the error document file locally.

The error document name is case sensitive and must exactly match the name that you enter when you enable static website hosting. For example, if you enter 404.html for the **Error document** name in the **Static website hosting** dialog box, your error document file name must also be 404.html.

1. Sign in to the AWS Management Console and open the Amazon S3 console .
2. In the **Buckets** list, choose the name of the bucket that you want to use to host a static website.
3. Enable static website hosting for your bucket, and enter the exact name of your error document (for example, 404.html).

After enabling static website hosting, proceed to step 6.

1. To upload the error document to your bucket, do one of the following:
   * Drag and drop the error document file into the console bucket listing.
   * Choose **Upload**, and follow the prompts to choose and upload the index file.

**Step 7: Test your website endpoint**

After you configure static website hosting for your bucket, you can test your website endpoint.

1. Under **Buckets**, choose the name of your bucket.
2. Choose **Properties**.
3. At the bottom of the page, under **Static website hosting**, choose your **Bucket website endpoint**.

Your index document opens in a separate browser window.

Now you have hosted a website on Amazon S3. This website is available at the Amazon S3 website endpoint. However, you might have a domain, such as example.com, that you want to use to serve the content from the website you created.

**Step 8: Clean up**

If you created your static website only as a learning exercise, delete the AWS resources that you allocated so that you no longer accrue charges. After you delete your AWS resources, your website is no longer available.

**PROJECT-3**

Integrate Grafana with Linux Server for high cpu utilization and create a graph in Grafana.

A screenshot of a computer

AI-generated content may be incorrect.

INDEX



1. INTRODUCTION
2. OBJECTIVE
3. WHY I BUILT THIS PROJECT?
4. TECHNOLOGY USED

1. SETUP CHECKLIST
2. STEP-BY-STEP IMPLEMENTATION VII. OUTPUT & TESTING

VII. CONCLUSION

I. INTRODUCTION



This project outlines how to integrate Grafana with a Linux server to monitor high CPU utilization and visualize the data with a Grafana dashboard. The integration involves setting up Prometheus to collect CPU metrics from the Linux server and then configuring Grafana to query and display these metrics in a user-friendly graph.

II. OBJECTIVE



Implement a robust monitoring solution for Linux server CPU utilization using Grafana, including data collection, visualization, and alert notifications.

The objective of this project, "Linux Server CPU Utilization Monitoring with Grafana," is to design, implement, and demonstrate a robust and userfriendly monitoring solution for Linux server CPU utilization.

III. WHY I BUILT THIS PROJECT?



These reasons often stem from real-world challenges faced in managing IT infrastructure. Here are the primary motivations:

* Proactive Problem Detection and Prevention
* Performance Optimization and Resource Planning
* Troubleshooting and Root Cause Analysis
* Operational Efficiency and Automation
* Cost-Effectiveness (especially with open-source tools)

IV. TECHNOLOGY USED



* Linux Operating System
* Node Exporter
* Node Exporter
* PromQL (Prometheus Query Language) ▪ Windows Operating System

V. SETUP CHECKLIST



1. HARDWARE:
   * Window 10 or more
   * 8GB RAM
2. SOFTWARE:
   * Linux Server
   * Node Exporter
   * Prometheus
   * Grafana Setup

VI. STEP-BY-STEP IMPLEMENTATION



Step-1: Linux Server - Node Exporter Installation

This part needs to be done on the Linux server you want to monitor.

1. SSH into your Linux Server:

Open your SSH client on Windows (e.g., PowerShell, Command Prompt, or PuTTY) and connect to your Linux server.

Bash

ssh your\_username@your\_linux\_server\_ip

(Replace your\_username and your\_linux\_server\_ip with your actual credentials and IP).

1. Download Node Exporter:

Go to the Prometheus Node Exporter GitHub releases page in your web browser to find the latest stable version. Look for the node\_exporter-X.Y.Z.linux-amd64.tar.gz file.

On your Linux server, use wget to download it.

Bash

# Replace X.Y.Z with the latest version number you found (e.g., 1.8.1) wget https://github.com/prometheus/node\_exporter/releases/download/vX.

Y.Z/node\_exporter-X.Y.Z.linux-amd64.tar.gz Example: wget https://github.com/prometheus/node\_exporter/releases/download/v1.

8.1/node\_exporter-1.8.1.linux-amd64.tar.gz

Extract the Archive:

Bash

tar -xzf node\_exporter-X.Y.Z.linux-amd64.tar.gz

This will create a directory like node\_exporter-X.Y.Z.linux-amd64.

1. Move the Binary to a System Path:

Bash

sudo mv node\_exporter-X.Y.Z.linux-amd64/node\_exporter

/usr/local/bin/

1. Create a Dedicated System User for Node Exporter:

This is a security best practice.

Bash

sudo useradd -rs /bin/false node\_exporter

1. Create a Systemd Service File:

This allows Node Exporter to run as a service, start automatically on boot, and be managed easily.

1. Open a new file for editing using nano or vim:

Bash

sudo nano /etc/systemd/system/node\_exporter.service

Paste the following content into the file:

Ini, TOML

[Unit]

Description=Node Exporter

After=network.target

[Service]

User=node\_exporter

Group=node\_exporter

Type=simple

ExecStart=/usr/local/bin/node\_exporter

[Install]

WantedBy=multi-user.target

Save the file: If using nano, press Ctrl+X, then Y to confirm save, then Enter.

1. Reload Systemd, Enable, and Start Node Exporter Service:

Bash

sudo systemctl daemon-reload sudo systemctl enable node\_exporter sudo systemctl start node\_exporter

1. Verify Node Exporter Status:

Bash

sudo systemctl status node\_exporter

You should see "Active: active (running)". Press Q to exit the status view.

1. Verify Node Exporter Metrics Locally (Optional, on Linux server):

Bash

curl http://localhost:9100/metrics

This command should output a long list of metrics.

1. Open Firewall Port (if applicable on Linux server):

If your Linux server has a firewall enabled (like ufw or firewalld), you need to allow incoming connections on port 9100 from your Prometheus server (your Windows machine).

For UFW (Ubuntu/Debian):

Bash

sudo ufw allow 9100/tcp sudo ufw reload

For Firewalld (CentOS/RHEL):

Bash

sudo firewall-cmd --add-port=9100/tcp --permanent sudo firewall-cmd –reload

k.Verify Node Exporter Access from Windows Browser: On your Windows machine, open your web browser and go to: http://<your\_linux\_server\_ip>:9100/metrics

You should see the same metrics page as you did with curl. If not, double-check the Linux server's IP, firewall, and that Node Exporter is running.

Step-2: Windows Machine - Prometheus Installation

This part needs to be done on your Windows machine.

1. Download Prometheus:

Go to the Prometheus Downloads page in your web browser. Download the latest stable release for Windows (amd64) (it will be a

.zip file).

1. Extract Prometheus:

Locate the downloaded .zip file (e.g., prometheus-X.Y.Z.windowsamd64.zip).

Right-click the file and select "Extract All...".

Choose a destination folder, for example, C:\ to extract it into C:\prometheus-X.Y.Z.windows-amd64.

For simplicity, rename the extracted folder to something like C:\Prometheus.

1. Configure prometheus.yml:

This is Prometheus's main configuration file, where you tell it what to scrape.

Navigate to your Prometheus installation directory (e.g., C:\Prometheus).

Open the prometheus.yml file using a text editor (e.g., Notepad++, VS Code).

Locate the scrape\_configs: section. You'll likely see a default job\_name: 'prometheus' for self-monitoring.

Add a new job\_name entry below the existing prometheus one to scrape your Linux server's Node Exporter.

YAML

# my global config global: scrape\_interval: 15s # Set the scrape interval to every 15 seconds.

Default is every 1 minute. evaluation\_interval: 15s # Evaluate rules every 15 seconds. Default is every 1 minute.

# scrape\_timeout is set to the global default (10s).

# A scrape configuration scraping as themselves.

scrape\_configs:

|  |
| --- |
| * job\_name: "prometheus" static\_configs: * targets: ["localhost:9090"]      * job\_name: 'linux\_server\_cpu' # <--- ADD THIS BLOCK static\_configs: * targets: ['your\_linux\_server\_ip:9100'] # <--- IMPORTANT:   REPLACE WITH YOUR LINUX SERVER'S ACTUAL IP |

Save the prometheus.yml file.

1. Run Prometheus:

Open a Command Prompt or PowerShell window as Administrator.

1. Navigate to your Prometheus directory:

DOS

cd C:\Prometheus

Start Prometheus:

DOS

.\prometheus.exe --config.file=prometheus.yml

You will see a lot of log output in the command prompt. Look for messages indicating it's starting and scraping targets.

f.Verify Prometheus UI and Targets:

Open your web browser on Windows.

Go to http://localhost:9090

In the Prometheus UI, navigate to Status > Targets.

You should see two targets: prometheus and linux\_server\_cpu.

Ensure both have a State of UP. If linux\_server\_cpu is DOWN, recheck the IP address in prometheus.yml and the firewall on your Linux server.

Step-3: Windows Machine - Grafana Installation

This part needs to be done on your Windows machine.

1. Download Grafana:

Go to the Grafana Downloads page for Windows in your web browser.

Download the recommended installer (.msi file).

1. Install Grafana:

Locate the downloaded .msi file.

Double-click the installer.

Follow the on-screen prompts. Accept the license agreement, choose the default installation location, and click "Install."

Once the installation is complete, you can click "Finish." Grafana should typically start as a Windows service automatically.

1. Verify Grafana UI:

Open your web browser on Windows.

Go to http://localhost:3000

You will be presented with a login screen.

Default Username: admin

Default Password: admin

Upon first login, Grafana will prompt you to change the default password. Choose a strong password and keep it safe.

Step-4: Grafana Configuration & Dashboard Creation Now we connect Grafana to Prometheus and build your CPU utilization graph.

* 1. Add Prometheus as a Data Source in Grafana: In Grafana (http://localhost:3000), click on Connections (the plug icon on the left sidebar).

Under "Connections," click on Data sources.

Click the "Add new data source" button.

Type "Prometheus" in the search bar or scroll down to find it, then click on the Prometheus option.

* 1. Settings:

Name: Prometheus-Local (or any descriptive name you prefer). HTTP URL: http://localhost:9090 (This is the address where your Prometheus is running on Windows).

Click the "Save & Test" button at the bottom. You should see a green pop-up notification "Data source is working."

* 1. Create a New Grafana Dashboard:

Click the + icon on the left sidebar.

Select New Dashboard.

Click on "Add new panel".

* 1. Create CPU Utilization Graph Panel:

In the new panel editor, under the Query tab:

Ensure your Prometheus-Local data source is selected (it usually defaults to the first one).

In the "Code" editor area, paste the following PromQL query for overall CPU utilization (excluding idle time):

Code snippet

100 - (avg by (instance)

(rate(node\_cpu\_seconds\_total{mode="idle",job="linux\_server\_cpu"}[5

m])) \* 100)

e.Explanation of the query:

node\_cpu\_seconds\_total: Metric from Node Exporter. mode="idle": Filters for the time the CPU spent idle.

job="linux\_server\_cpu": Filters for metrics from your specific Linux server.

[5m]: Calculates the rate over the last 5 minutes. rate(...): Computes the per-second average rate of increase.

avg by (instance): Averages the rate across all CPU cores for each server instance.

100 - (...): Converts the idle percentage into CPU utilization percentage.

Observe the graph updating on the right side.

In the Visualization tab (or Panel options on the right side): Panel Title: Change it to "Linux Server CPU Utilization (%)" Under Standard options > Unit: Select percent (0-100).

(Optional) Under Legend: You can use {{instance}} as the Value. This will display the IP address of your Linux server in the legend.

Click "Apply" in the top right corner.

f.Save Your Dashboard:

Click the "Save dashboard" icon (looks like a floppy disk) in the top right corner of the dashboard.

Enter a descriptive name, e.g., "Linux Server Monitoring Dashboard".

Click "Save".

Step-5: Grafana Alerting Configuration

Now, let's set up an alert for high CPU utilization.

a.Configure a Contact Point:

This is where Grafana will send notifications. For testing, a simple email or webhook is good.

On the left sidebar in Grafana, click the bell icon (Alerting).

Go to Contact points.

Click "Add contact point".

Name: MyEmailNotifications (or MyWebhookNotifications) Type:

For Email: Select "Email". Enter your email address in "Addresses". For Webhook (easy for testing): Select "Webhook". Go to https://webhook.site/ in a new browser tab, copy the unique URL it provides, and paste it into Grafana's "URL" field.

Click "Save & Test". If using email, check your inbox for a test email. If using webhook, check the webhook.site page for the incoming test request.

b.Create an Alert Rule:

In Grafana's Alerting section (bell icon), go to Alert rules.

Click "New alert rule".

Rule name: High CPU Utilization on Linux Server.

Folder: You can create a new folder (e.g., "Server Alerts") or use

"General."

Data source: Select your Prometheus-Local data source.

Query: Use the same PromQL query as your graph panel:

Code snippet

100 - (avg by (instance)

(rate(node\_cpu\_seconds\_total{mode="idle",job="linux\_server\_cpu"}[5

m])) \* 100)

Condition:

WHEN: avg() of Query A

IS ABOVE: 80 (This means alert if CPU goes above 80%. You can adjust this threshold.)

FOR: 5m (This means the condition must be true for 5 continuous minutes before the alert fires, preventing flapping alerts from transient spikes.)

Evaluation Behavior:

Evaluate every: 1m (How often Grafana checks the rule).

Alert state if no data or all values are null: Select No Data or Alerting based on your preference (e.g., Alerting if you want to know if data stops flowing).

Alert state if execution error or timeout: Select Alerting.

Notifications:

Under "Send to," select your configured contact point (e.g., MyEmailNotifications).

Summary: High CPU on {{ $labels.instance }}: Current utilization is {{

$value | humanize }}%

Annotations (Optional, but useful): Add more details.

Key: description

Value: CPU utilization on server {{ $labels.instance }} has been above {{ $threshold }}% for 5 minutes.

Click "Save rule".

c.Test the Alert (Important!):

Go back to your Linux server (via SSH).

Install a stress testing tool (if you don't have one). stress-ng is powerful.

For Debian/Ubuntu:

Bash

sudo apt update sudo apt install stress-ng

For CentOS/RHEL:

Bash

sudo yum install epel-release sudo yum install stress-ng

Generate High CPU Load: Run stress-ng to utilize your CPU. Adjust -cpu to the number of cores on your server.

Bash

stress-ng --cpu $(nproc) --timeout 300s --metrics-brief

# nproc gets the number of CPU cores. --timeout 300s runs for 5 minutes.

Alternative simple CPU hog (for single core, run multiple times for multiple cores):

Bash

yes > /dev/null &

yes > /dev/null & # Repeat for each core

Monitor: Watch your Grafana dashboard. You should see the CPU utilization spike.

Wait: Wait for the FOR duration (e.g., 5 minutes) of your alert rule. Verify Alert: Check your email or webhook.site to see if the alert notification was received.

Stop Stress Test:

For stress-ng, it will stop after the timeout.

For yes > /dev/null &, you'll need to kill the processes:

Bash

Kill all yes

Observe the alert state in Grafana returning to "Normal" and potentially a "Resolved" notification being sent.

This completes the step-by-step implementation. You now have a working monitoring solution with visualization and alerting! Remember to capture screenshots at various stages as planned for your pictorial representation and presentation.

VII. OUTPUT & TESTING



Linux Server : Node Exporter running as a service, exposing CPU metrics on port 9100, accessible via http://<Linux\_Server\_IP>:9100/metrics.

Windows (Prometheus ): Prometheus running, configured to scrape Node Exporter, and its web UI (http://localhost:9090/targets) shows the Linux server as an "UP" target, indicating successful data collection.

Windows (Grafana): Grafana running, successfully connected to Prometheus as a data source. A dashboard displays a dynamic CPU utilization graph, and an alert rule is configured for high CPU. Notifications : Alert notifications (e.g., email, webhook) are received when the CPUthreshold is breached.

Testing :

Node Exporter : Verify service status (active (running)), local curl output, and remote browser access from Windows.

Prometheus : Check its Targets page in the UI to confirm Node Exporter is being scraped successfully (status UP).

Grafana Visualization : Observe the CPU graph on the dashboard updating in real-time. Perform a CPU load test on the Linux server and visually confirm the graph reflects the spike.

Grafana Alerting: Send a test notification from Grafana to verify the contact point. Then, induce high CPU load on the Linuxserver and confirm an alert notification is received within the configured timeframe.

Resolution : After stopping the load, observe the alert resolving and potentially a "resolved" notification.

VIII. CONCLUSION



This project successfully demonstrates the design, implementation, and operationalization of a robust and cost-effective monitoring solution for Linux server CPU utilization using a powerful open-source stack comprising Node Exporter, Prometheus, and Grafana.

Through a structured, sprint-based approach, we have achieved the following key objectives:

* Seamless Data Collection
* Centralized Data Storage and Retrieval
* Intuitive Visualization
* Proactive Alerting

**TEAM MEMBER NAME-**

# Infinity Storage

**Anurag Singh Yadav**

**Navdeep Thakur**

**Nikhil kanaujiya**

**Abhishek Singh Yadav**

**Anurag Tiwari**

+