****

**SUBMITTED BY :-**

**MANSI SINGH.**

**SUBMITTED TO:-**

**ABHINAV SIR**

**BCA SEMESTER- 2**

**SERVERLESS IMAGE PROCESSING**

**Introduction** –

Serverless image processing uses cloud services to process images without provisioning or managing servers. This approach allows developers to focus on writing code and processing images without worrying about infrastructure.

**How does AWS Serverless Image Processing work–**

1. Image Upload: Images are uploaded to a storage service like Amazon S3.

2. Trigger: An event is triggered, which invokes a serverless function (e.g., AWS Lambda).

3. Image Processing: The serverless function processes the image using libraries like ImageMagick or Pillow.

4. Output: The processed image is saved to a storage service or further processed.

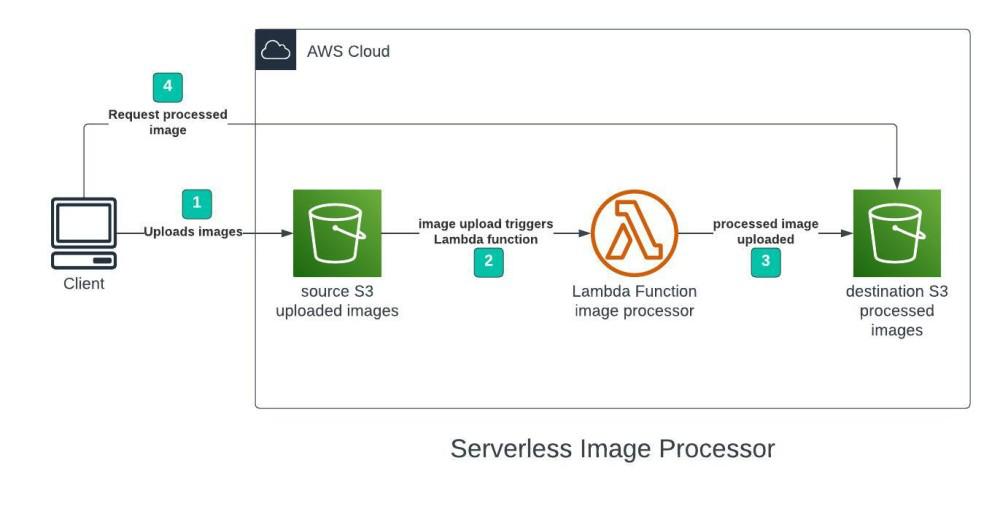
**Benefits**:

1. Scalability: Serverless image processing scales automatically to handle large volumes of images.

2. Cost-effective: You only pay for the compute time used to process images.

3. Reduced administrative burden: No need to manage servers or worry about infrastructure.

**ARCHITECTURE DIAGRAM –**

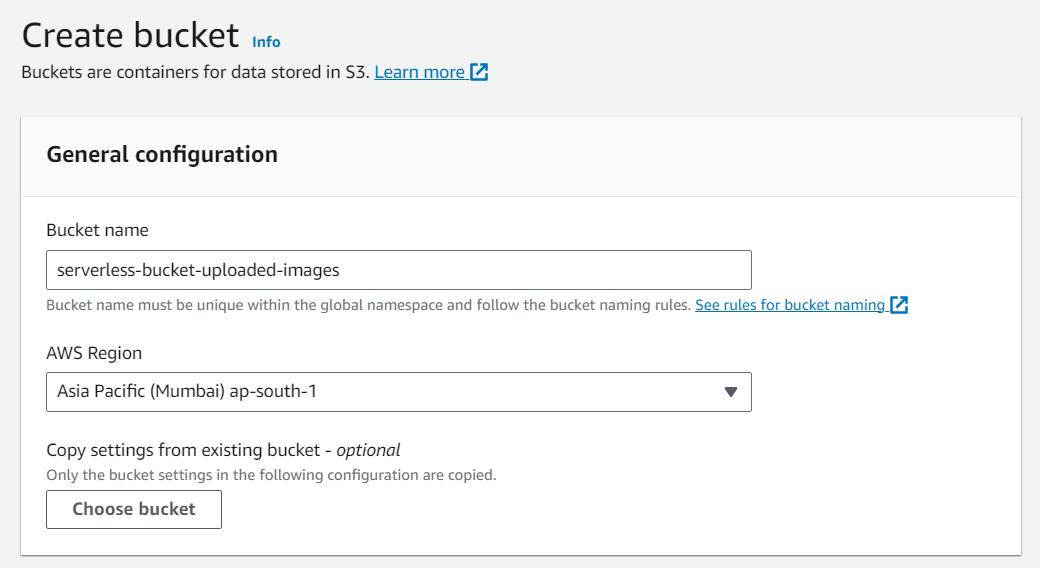


**STEP-BY-STEP SET UP**–

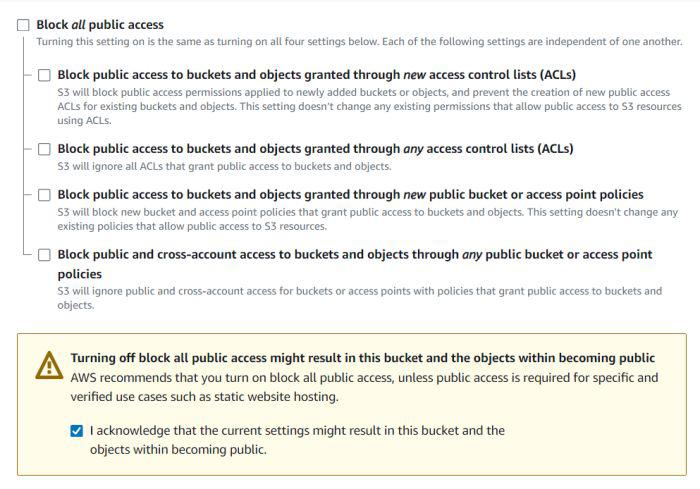
1.Creating S3 buckets

We will use two S3 buckets:

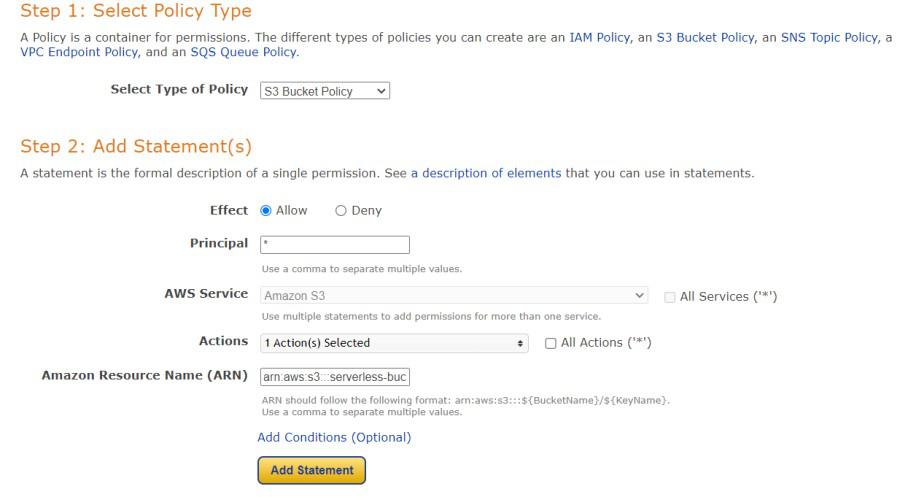
* Source Bucket: For storing uploaded images.
* Destination Bucket: For storing processed images.
* Go to S3 console and click Create bucket. Enter bucket name as ‘serverless-bucket-uploaded-images’.
* Choose any AWS region as ‘ap-south-1’.

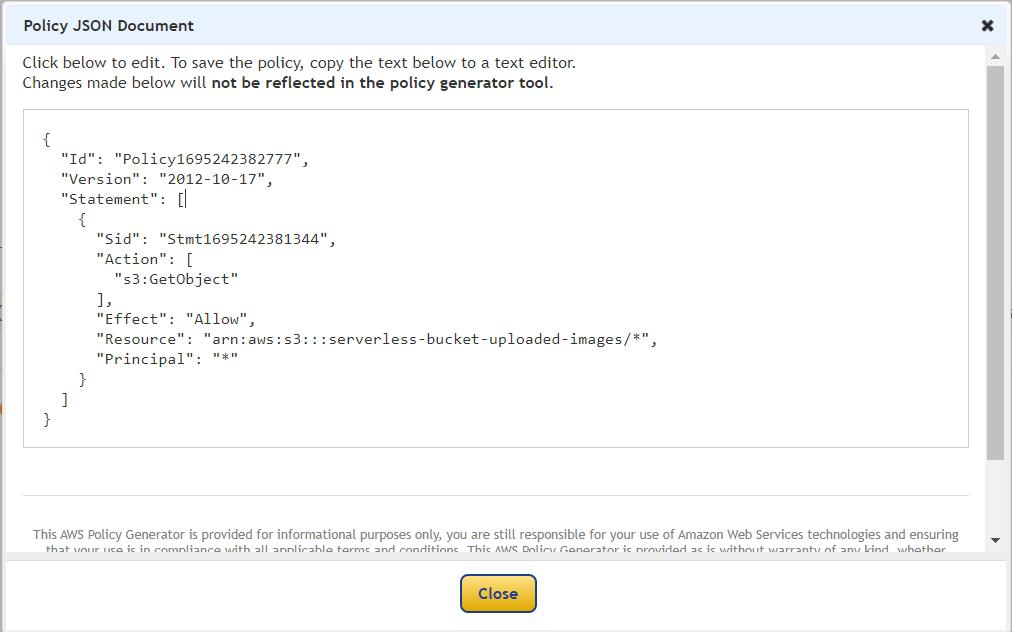


2. Configuring S3 bucket policy–

* Disable “block all public access”.
* Agree to the warning.
* Leave all other settings default and create bucket.
* Similarly, create another bucket. This bucket will be used to store the processed images.
* Go to your source bucket and then click on Permissions tab. Scroll to Bucket Policy tab. Click Edit.
* Click on policy generator.

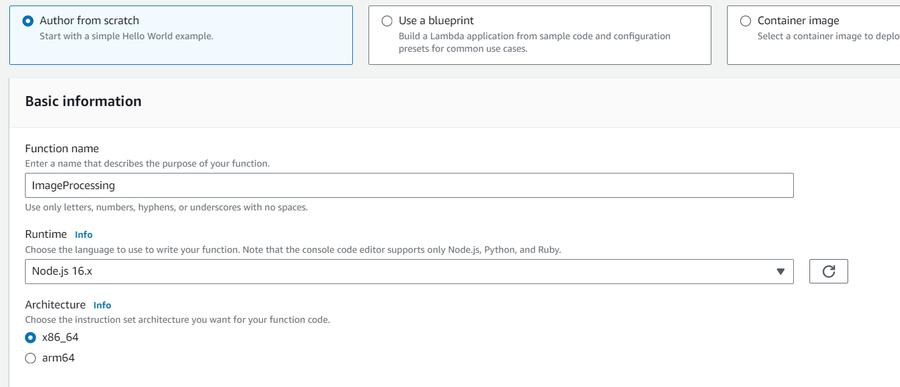
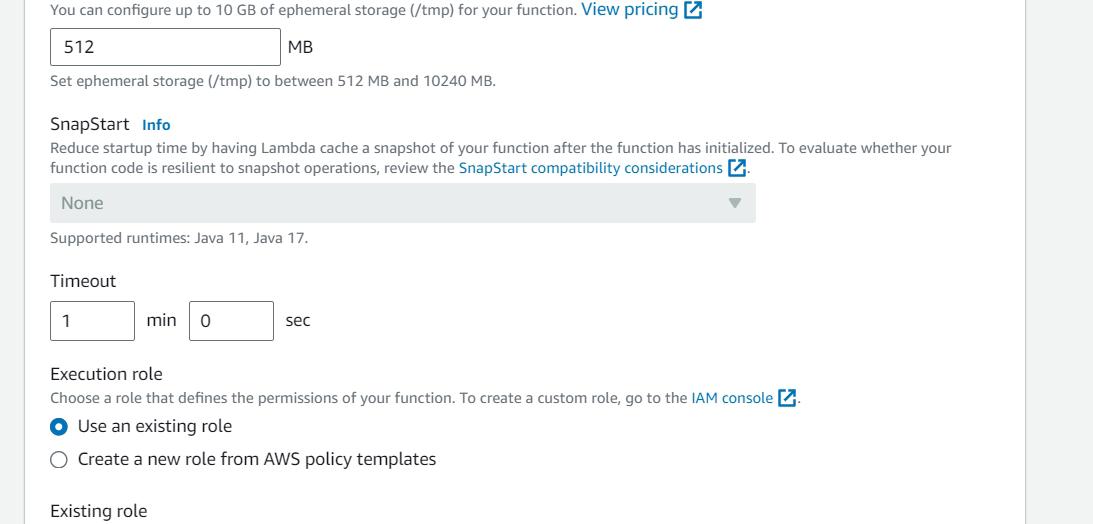
Enter the following settings:

* Type of policy: S3 Bucket Policy
* Effect:Allow
* Principal: \*
* Actions: GetObject
* Amazon Resource Name (ARN): arn:aws:s3:::SOURCE\_BUCKET\_NAME
* Click Add Statement and then generate policy. Copy the JSON object.



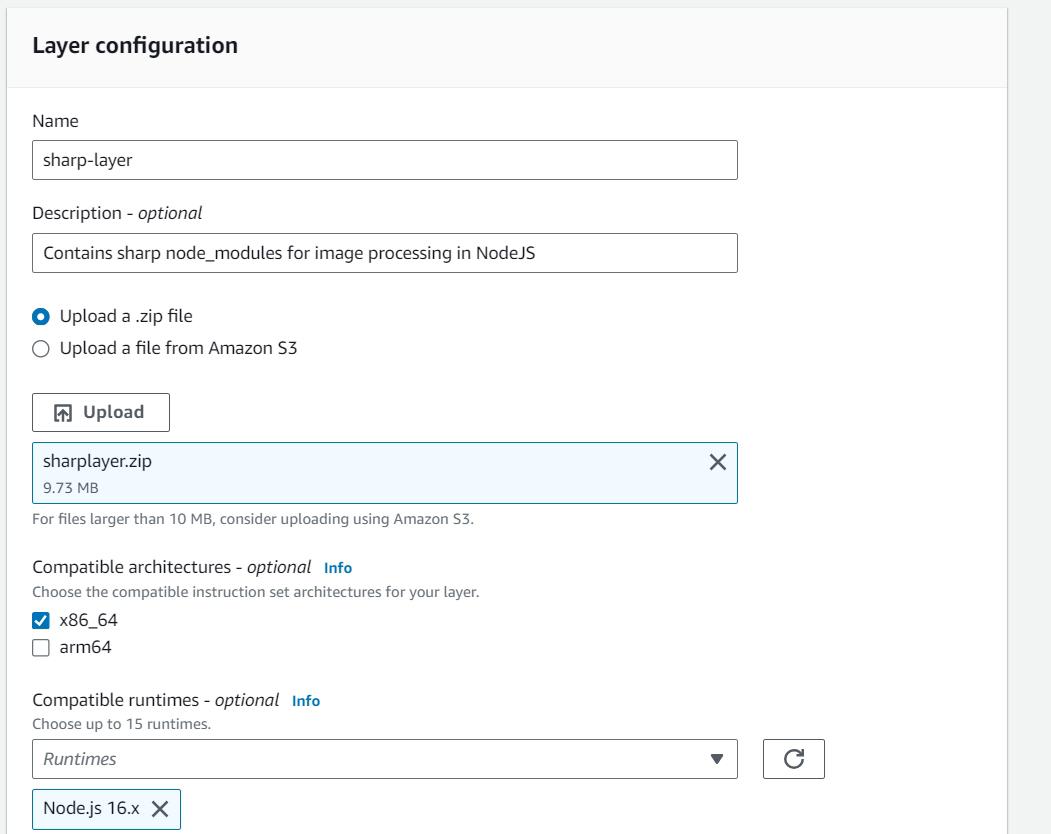
* Paste it in the policy editor and then save changes. Follow same steps to attach a policy to the processed images S3 bucket.

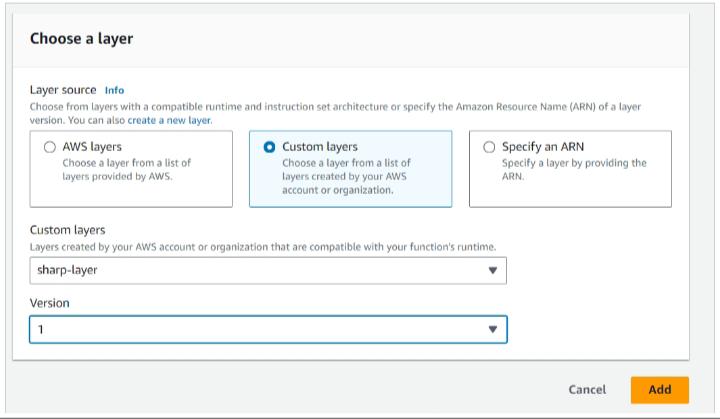
3.Creating Lambda function–

* 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.
* 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.
* Click on save changes.

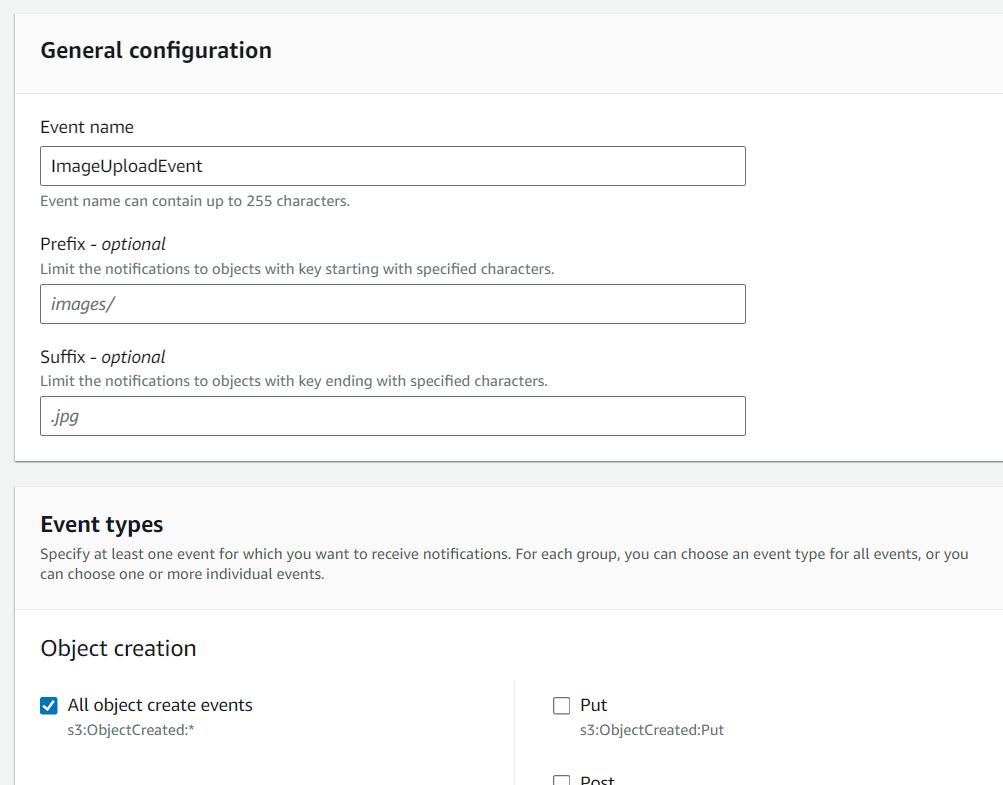
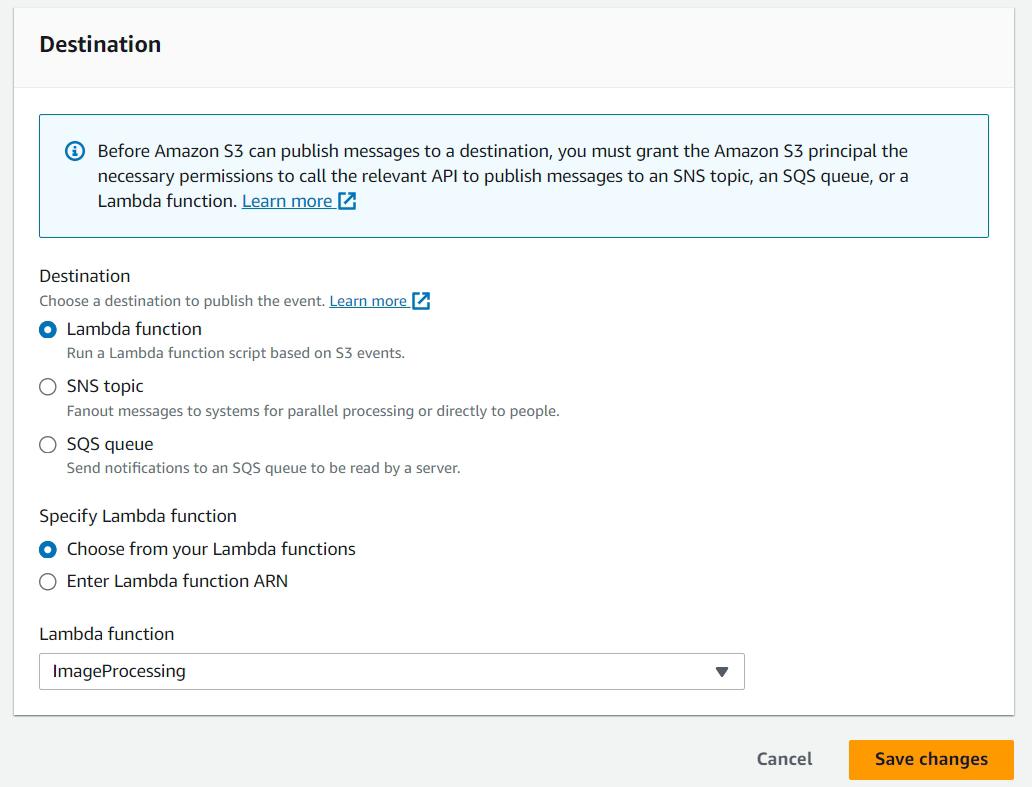
4. Creating Lambda layer and attaching it to Lambda function

* Create a zip file of the nodejs directory and name it “sharplayer.zip”.
* 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.

5.Creating S3 Trigger –

* Go to S3 console. Select the source bucket.
* Go to the Properties tab. Navigate to “EventNotifications”. Click “Create Event Notifications”.
* Give an appropriate name to the event. Check the “Allobject create events”.
* Navigate to the “Destination” and select your lambda function. Save changes.

6. Testing the application–

* Upload an image file to source S3 bucket .
* Wait for few seconds and check the destination bucket. There you will see two images.

**AWS Services Used –**

* Amazon S3.
* AWS Lambda.
* IAM

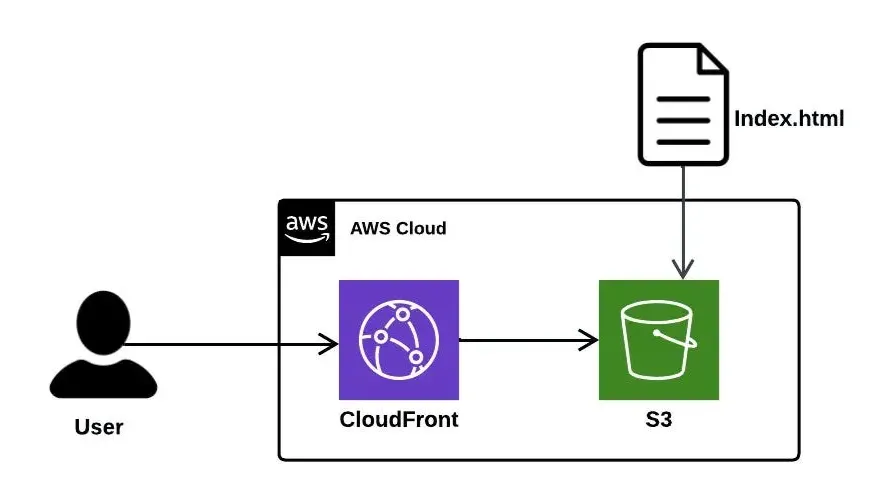
**Conclusion** –

In the world of modern applications, speed and scalability are no longer luxuries—they’re essentials. With AWS Serverless Image Processing, we’ve unlocked a powerful, efficient, and cost-effective pipeline where images are transformed in the blink of an eye—without a single server to manage.

**PROJECT:2**

DEPLOY A STATIC WEBSITE IN AWS

Deploying a static website on AWS involves hosting it on Amazon S3, a cost-effective and scalable storage service, and optionally utilizing Amazon **CloudFront** for enhanced performance and security. This approach eliminates the need for web servers and simplifies the deployment process.



:

Cost-Effectiveness:

Pay-as-you-go pricing:

AWS S3 and CloudFront charge based on usage, making it a cost-effective solution, especially for websites with fluctuating traffic.

No server maintenance:

Static websites don't require traditional servers, eliminating the need for server management and associated costs.

Free Tier:

AWS offers a Free Tier that can be utilized for hosting small static websites, further reducing costs.

Reliability and Scalability:

High Availability:

AWS services like S3 and CloudFront are designed for high availability, ensuring your website remains accessible to users.

Scalability:

AWS can easily scale to handle increased traffic without requiring manual intervention or additional costs.

Global CDN:

CloudFront distributes content across multiple edge locations, providing low latency and high availability for users worldwide.

Security:

Secure Content Delivery:

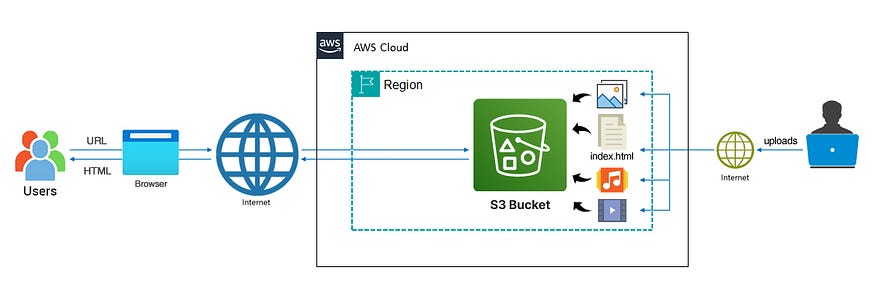
CloudFront can be configured to serve content over HTTPS, ensuring secure connections between users and your website.

Fine-grained access control:

S3 offers features like Access Control Lists (ACLs) and bucket policies to control access to your website's content.

Data Encryption:

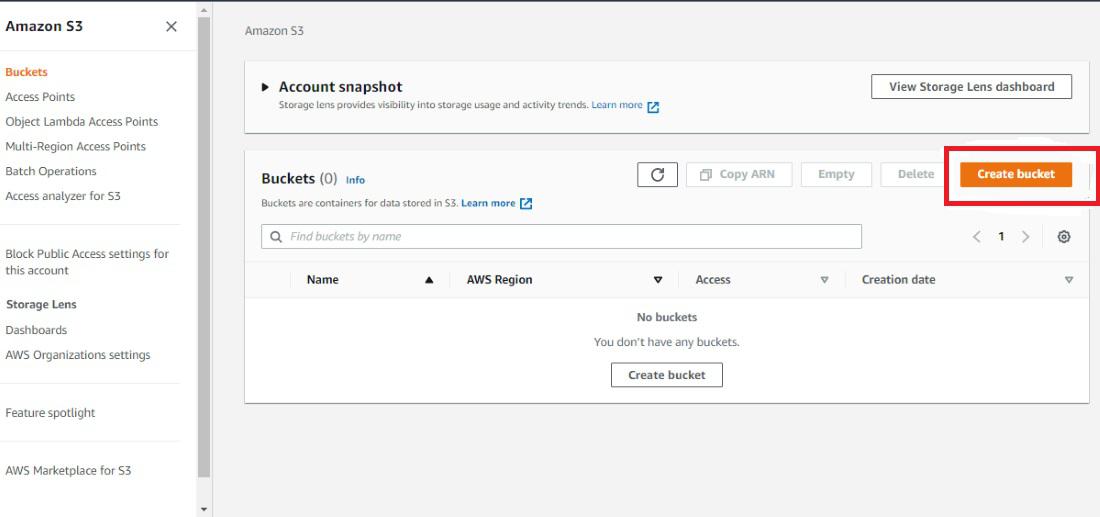
AWS provides encryption options for data at rest (in S3) and in transit (through CloudFront), enhancing security.



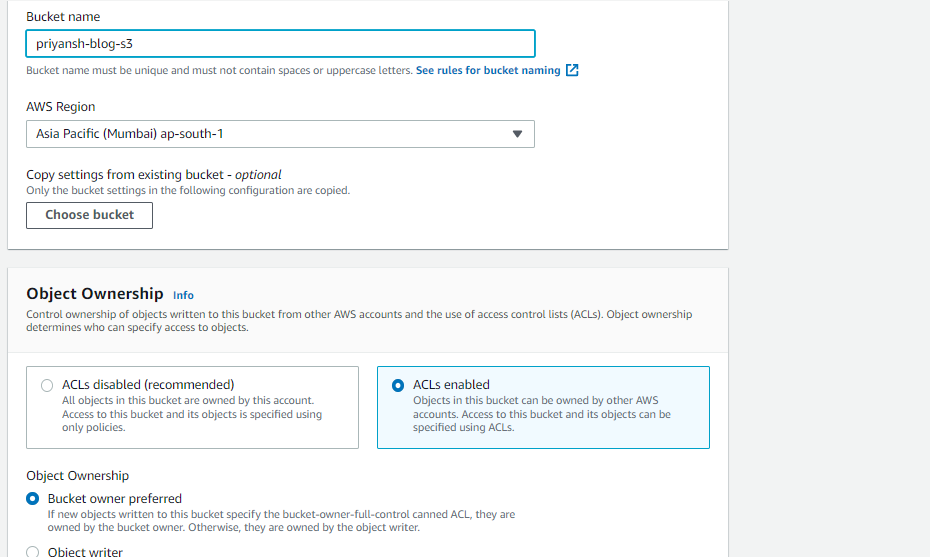
**Step 1: Creating a Bucket**

First, we have to launch our S3 instance. Follow these steps for creating a Bucket

* Open the Amazon S3 console by logging into the AWS Management Console at https://console.aws.amazon.com/s3/.
* Click on **Create Bucket**.

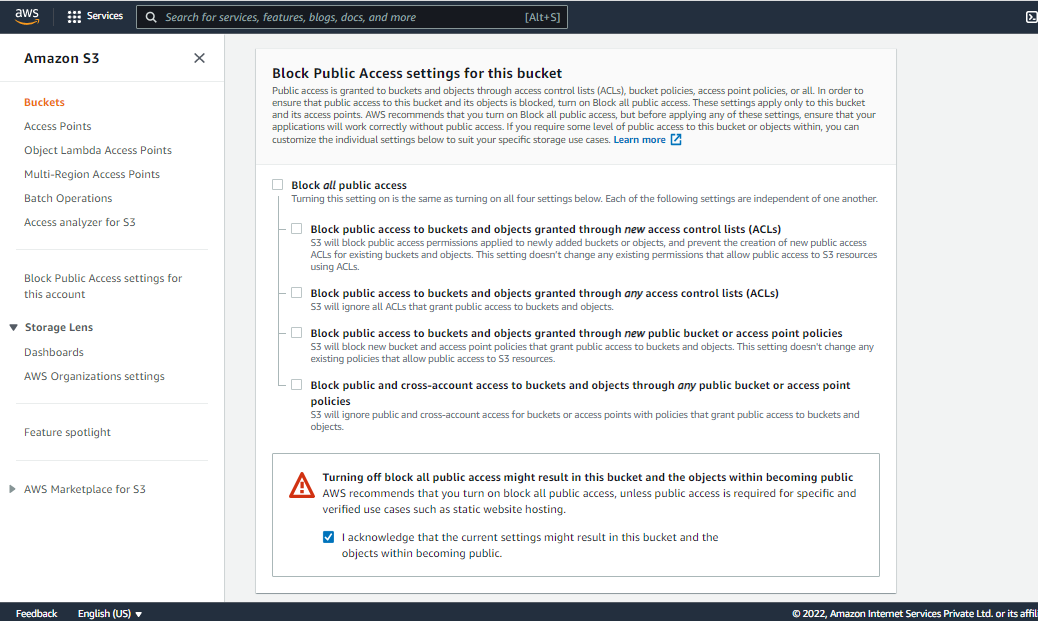


* Choose **Bucket Name -**Bucket Name Should be Unique
* **AWS Region -**  Choose a region close to you or the region where you want to create the bucket (Example — Mumbai)
* **Object Ownership -**Enable for making Public, Otherwise disable

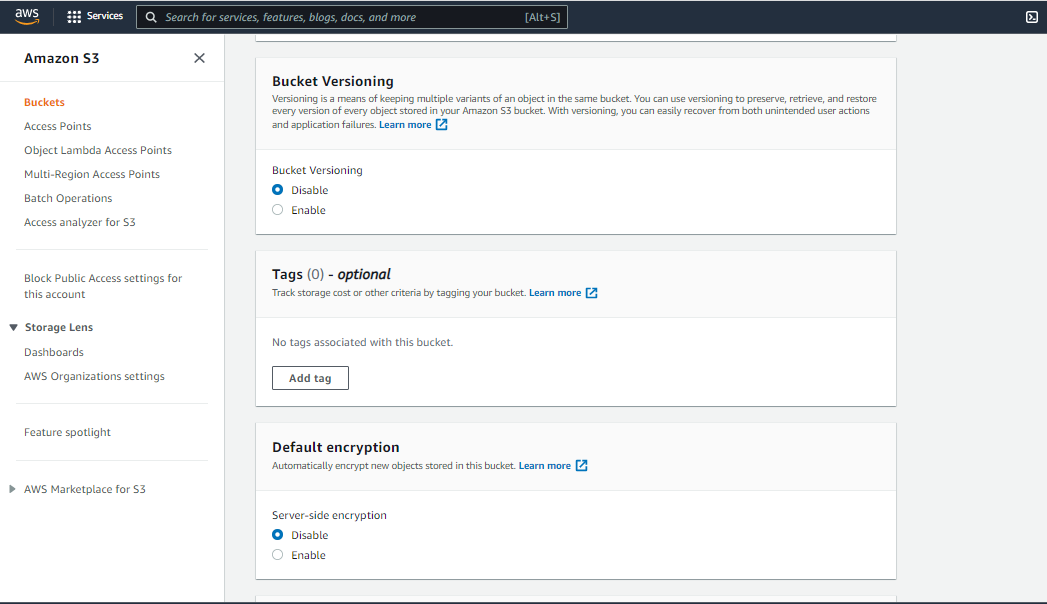


**Step 2: Block Public Access settings for the bucket**

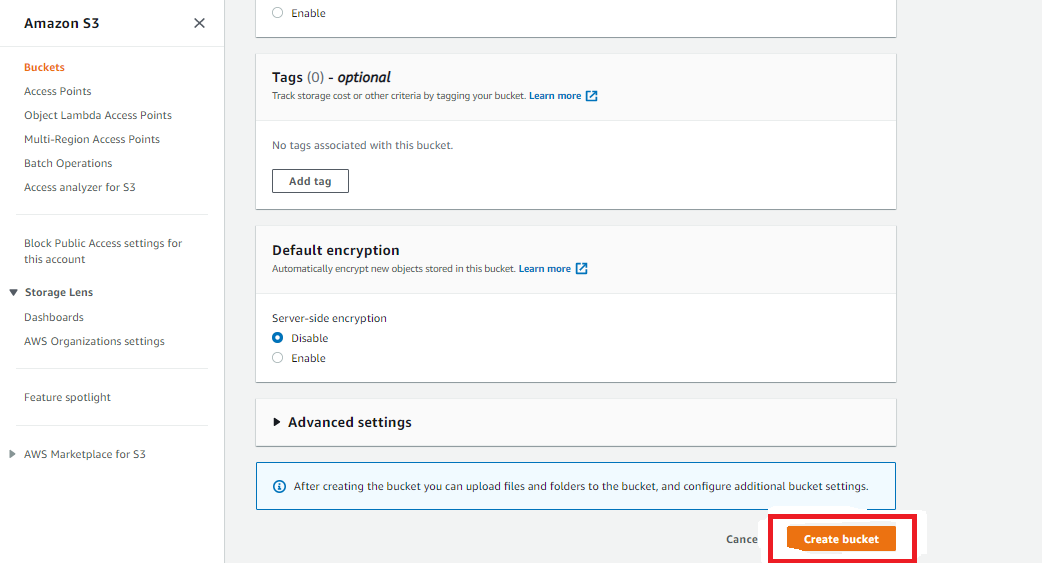
* Uncheck (Block all public access) for the public, otherwise set default. If you uncheck (Block all public keys).



* **Bucket Versioning**:- You have to do Nothing (Disable)
* **Tags(0)** : Optional
* **Default encryption**: Disable

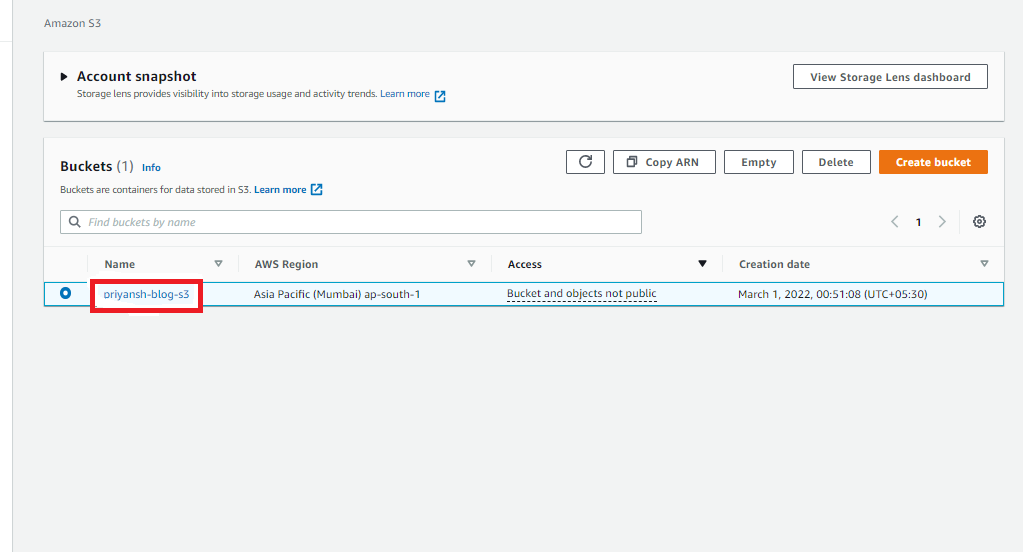


* Now, click on**Create Bucket**

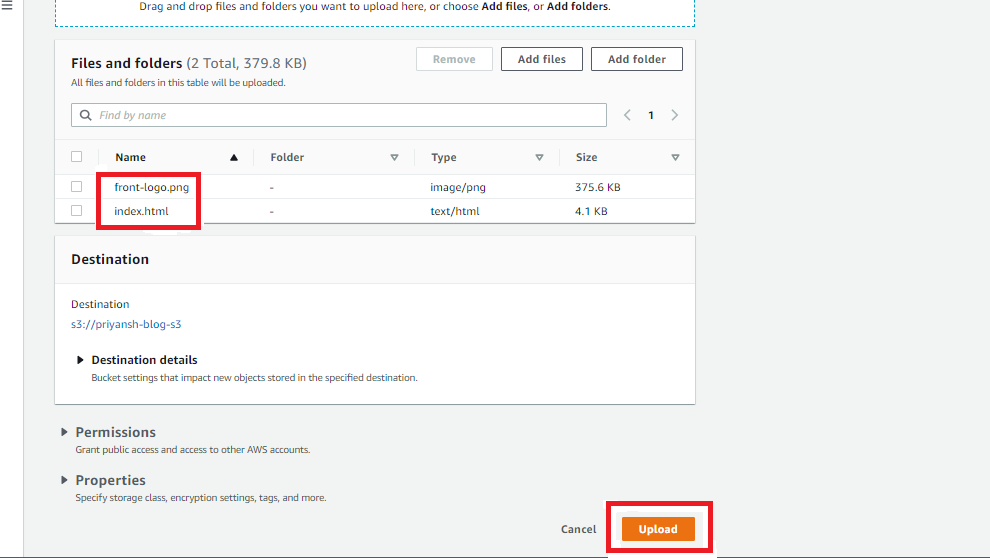


**Step 3: Now upload code files**

* **Select Bucket** and Click your**Bucket Name**.



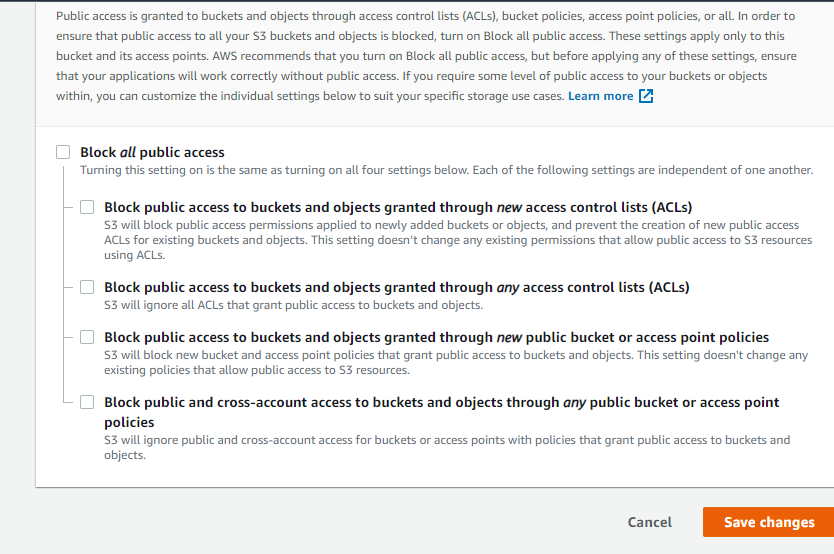
* Now, click on upload (then click add File/folder) and select your **HTML code**file from your PC/Laptop.



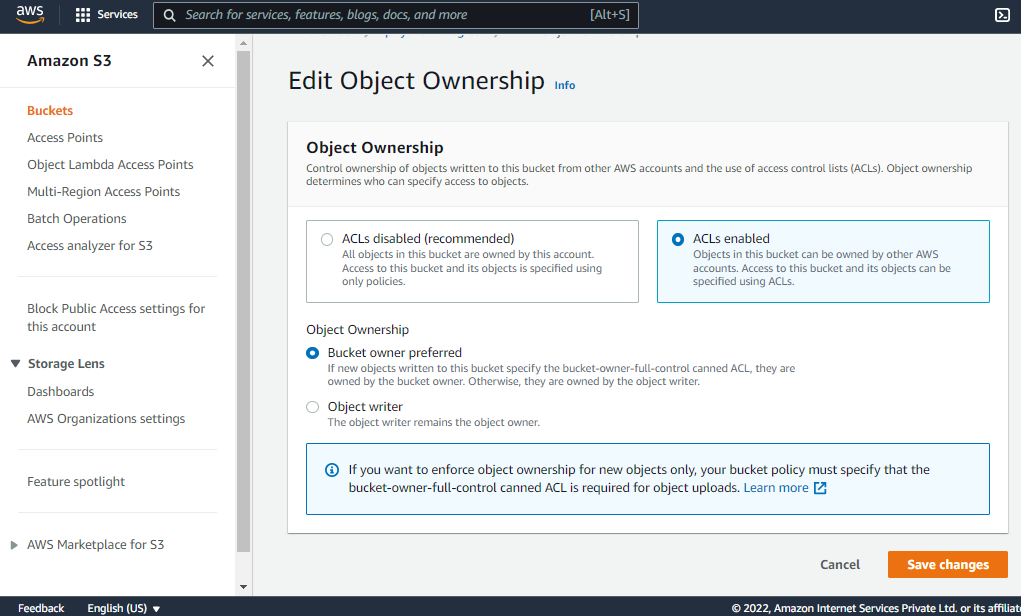
* After uploading, click on**Close**.

**Step 4:**Once the Files are uploaded successfully, click on **Permissions**and now follow this Process -

* **Block public access:**
  + Click on **Edit,**under**Bucket Policy**.
  + **Uncheck Block** all public access.
  + **Save changes** then type “**confirm”**.

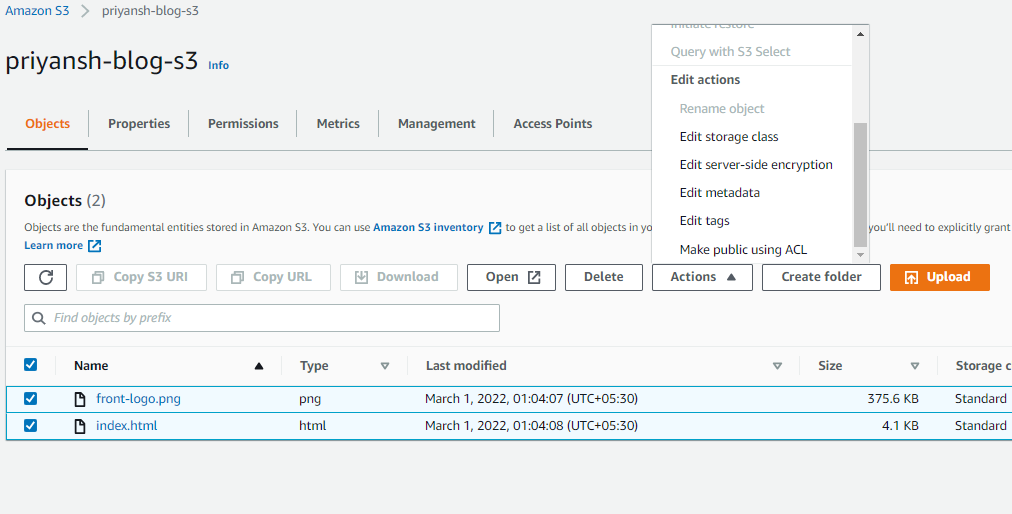


* **Object Ownership:**
  + Click on **Edit**.
  + Click on **ACLs Enabled.**
  + **Check**I acknowledge ….. restored.
  + Choose**Save Changes.**



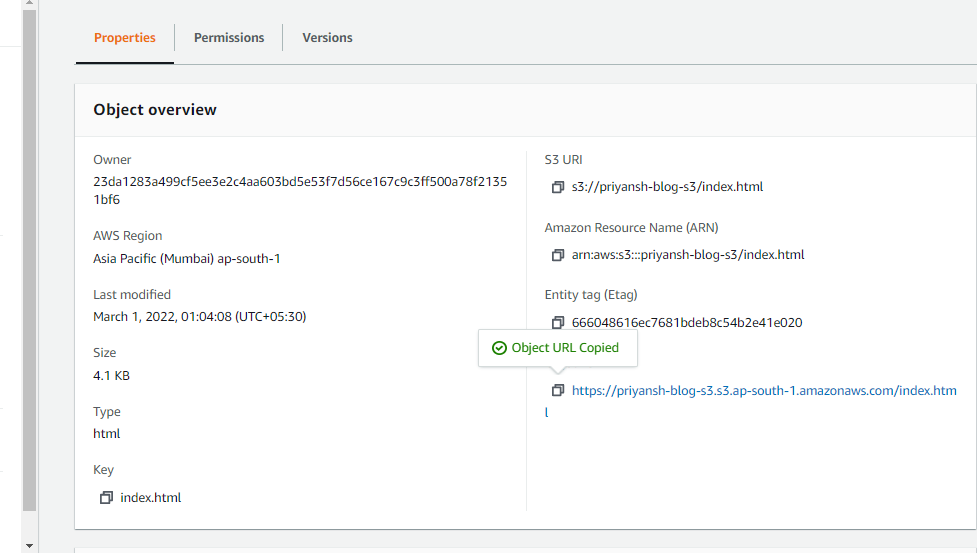
**Step 5:- Make public Object**

* Now, Click on **Objects**.
* Select your **All Objects**.
* Now, Click on **Actions**.
* Select**Make Public Using ACL**.
* Now, Click on **Make Public** and **Close**.



**Step 6: Copy your Object URL**

* Now, click on your **HTML File** Object Name.
* Copy the **Object URL**.



**Step 7: Check out your Website!**

* Directly Paste this URL into the Other Tab or your other System.
* **Congratulation**, Now Your Website is available in the Public.
* You **Successfully Host** Your**Website by AWS S3.**

AWS SERVICES USED:

1. Amazon S3 (Simple Storage Service):

S3 is used to store the static website files (HTML, CSS, JavaScript, images, etc.).

It provides a simple and scalable way to host websites by enabling static website hosting on a bucket.

2. Amazon CloudFront:

CloudFront acts as a Content Delivery Network (CDN).

CloudFront also enables HTTPS for secure connections to the website.

3. AWS Amplify:

Amplify Hosting is a fully managed service that simplifies the deployment of static websites.

It integrates with S3 and CloudFront, handling much of the setup and configuration automatically.

**Here's a breakdown of potential problems and their solutions:**

1. S3 Bucket Configuration:

* Incorrect Permissions:

Ensure your S3 bucket and objects are publicly readable.

* Static Website Hosting Not Enabled:

Verify that static website hosting is correctly enabled in the S3 bucket's properties and that you have specified the index document (e.g., index.html) and error document.

* Incorrect Content-Type:

The Content-Type metadata for your index.html file should be set to text/html.

* Root Folder:

Make sure your website files are in the root folder of your S3 bucket.

2. CloudFront Integration:

* Incorrect Origin Settings:

When configuring CloudFront to point to your S3 bucket, ensure you are using the regional endpoint name for your bucket, not the global endpoint name.

* Default Root Object:

The CloudFront distribution's "Default Root Object" setting should be set to your index document (e.g., index.html).

* Propagation Delay:

CloudFront distributions can take some time to propagate changes. Be patient if your website is not immediately accessible after configuration.

3. Accessing the Website:

* Incorrect Endpoint:

Use the S3 website endpoint URL (which includes http and potentially a region code), not the regular S3 bucket URL.

* "Access Denied" Error:

If you see this error, it usually indicates a problem with permissions or incorrect configuration of the S3 static website endpoint.

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

**Introduction –**

High CPU utilization can lead to performance issues, slow response times, and even crashes. Monitoring CPU utilization helps you:

1. Identify bottlenecks: Detect performance issues before they become critical.

2. Optimize resources: Adjust resources (e.g., instance types, auto-scaling) to match workload demands.

3. Improve reliability: Ensure high availability and reduce downtime.

**Tools Used**:

1. Grafana: A popular monitoring and visualization platform.

2. Prometheus: A metrics collection and alerting system.

3. AWS EC2: A Linux-based virtual server in the cloud.

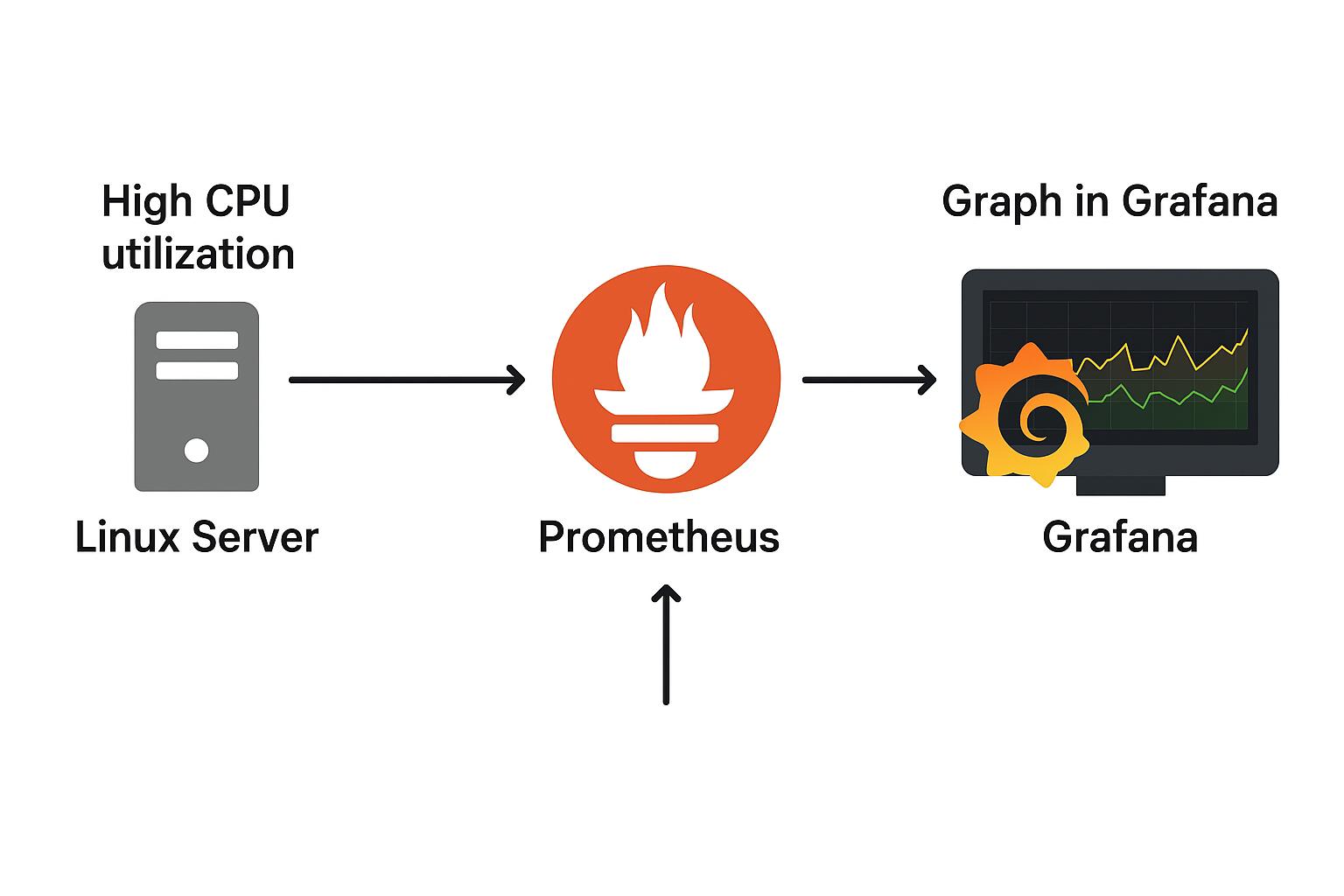
**Benefits**:

1. Real-time monitoring: Get instant insights into CPU utilization.

2. Customizable dashboards: Create tailored dashboards for specific use cases.

3. Scalability: Easily scale monitoring to accommodate growing infrastructure.

.

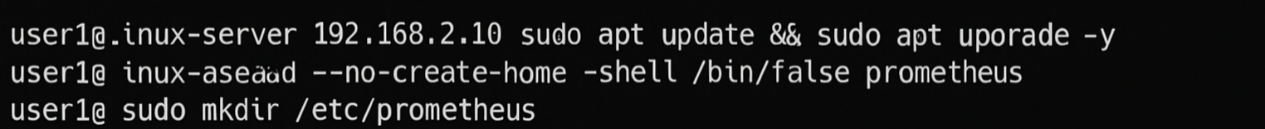
****

**Step-by-step set up–**

Step 1: Install Prometheus on the Linux server

1. Install Prometheus: Run `sudo apt-get install prometheus` (for Ubuntu-based systems).

2. Configure Prometheus: Edit the Prometheus configuration file to scrape CPU metrics.

3. Start Prometheus: Run `sudo systemctl start prometheus` and `sudo systemctl enable prometheus`.

Step 2: Install Grafana on the Linux server

1. Install Grafana: Run `sudo apt-get install grafana-server` (for Ubuntu-based systems).

2. Start Grafana: Run `sudo systemctl start grafana-server` and `sudo systemctl enable grafana-server`.

Step 3: Configure Grafana to use Prometheus as a data source

1. Access Grafana: Open the Grafana web interface .

2. Add data source: Go to “Configuration” > “Data Sources” and add a new data source.

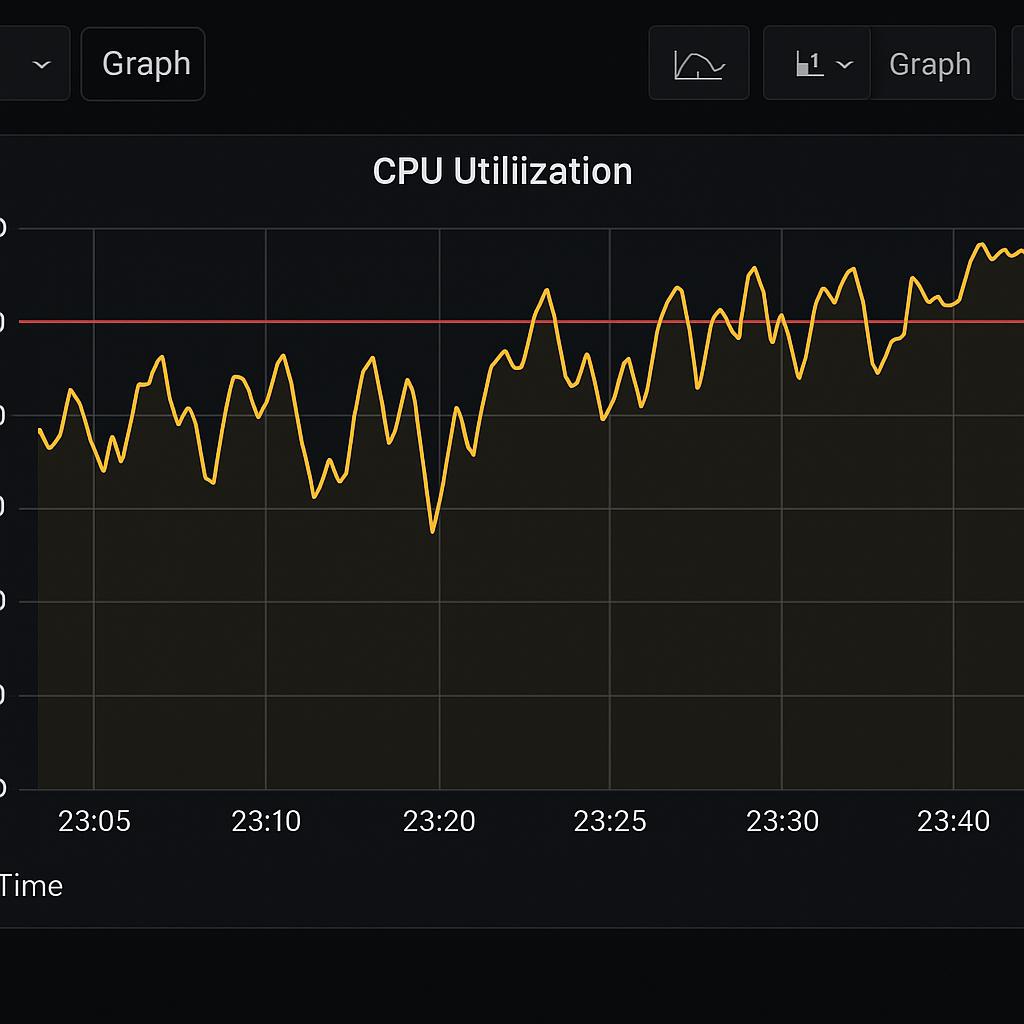
3. Configure Prometheus: Set the data source type to “Prometheus” and enter the Prometheus URL .

Step 4: Create a graph in Grafana for high CPU utilization

1. Create a new dashboard: Go to “Dashboards” > “New Dashboard” and add a new panel.

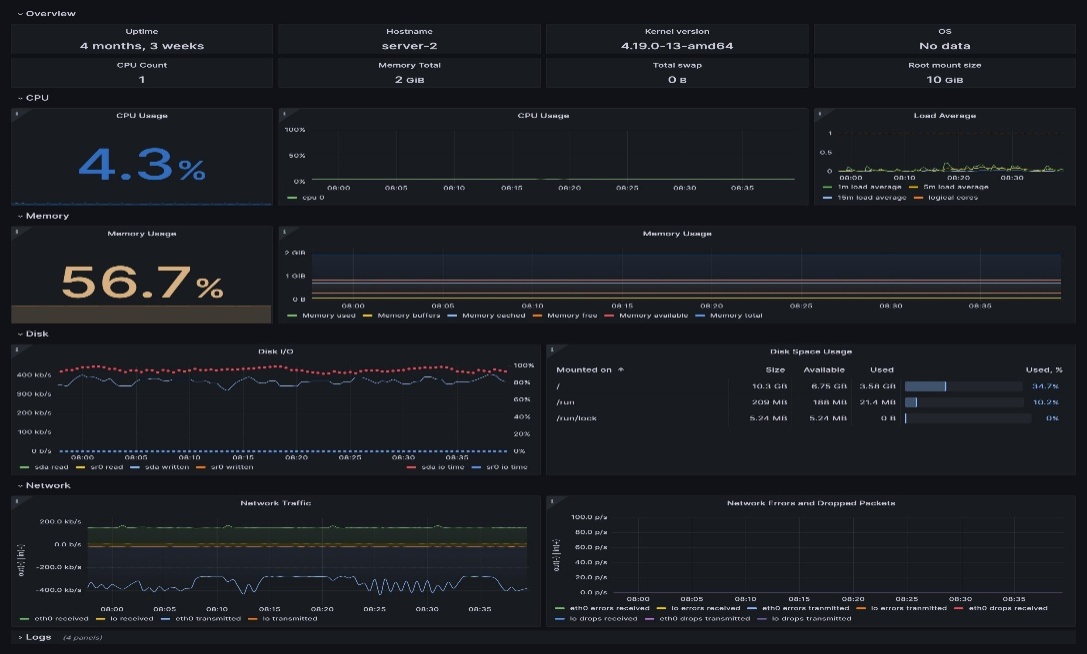
2. Choose visualization: Select a graph visualization (e.g., “Time series”).

3. Query Prometheus: Use the Prometheus query language (PromQL) to query CPU utilization metrics.

4. Customize the graph: Adjust the graph settings (e.g., title, labels, colors).

Step 5: Visualize high CPU utilization

1. Save the dashboard: Save the dashboard and panel.

2. View the graph: View the graph to visualize high CPU utilization.

**Tips and Variations:**

1. Use Amazon CloudWatch: Instead of Prometheus, use Amazon CloudWatch to collect metrics.

2. Use Amazon Managed Grafana: Use a managed Grafana service like Amazon Managed Grafana.

3. Add alerts: Set up alerts in Grafana to notify when CPU utilization exceeds a threshold.

**Conclusion –**

By integrating Grafana with a Linux server, you’ve transformed raw performance data into actionable intelligence. The high CPU utilization graph isn’t just a chart—it’s a real-time pulse of your system, helping you catch performance spikes before they become problems.