**PROJECT CLOUD COMPUTING**

NAME: ARJUN SINGH

COURSE: BCA

SEMESTER: 4nd

TABLE OF CONTENTS

* Project 1:Serverless Image Processing
* Project Objective
* AWS Services Used
* Serverless Image Processing Flow
* Steps To Follow
* Why two different buckets?
* Real Time Applications
* Conclusion
* Project 2: Deploy a Static Website on AWS
* Project Objective
* Tools and Technologies Used
* Architecture Diagram
* Steps To Follow
* Advantages of Hosting On S3
* Real Life Applications
* Conclusion
* Project 3: Integrate Grafana with Linux Server
* Project Objective
* Technologies Used
* Architecture Diagram
* Steps to Follow
* Real Life Applications
* Advantages of Grafana + Prometheus
* Conclusion

Project 1

Serverless Image Processing

🔍PROJECT OBJECTIVE**:**

The objective of this project is to automate image resizing and optimization using a serverless architecture, reducing manual effort, storage costs, and improving application scalability.

AWS SERVICES USED:

1. 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:

Storage: It can be used for storing large amounts of data.

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.

Static website:S3 offers static website hosting through HTML files stored in S3.

Data lakes and big data analytics:Companies can use [AWS S3](https://www.geeksforgeeks.org/introduction-to-aws-simple-storage-service-aws-s3/)as a data lake and then run analytics on it for getting business insights and take critical decisions.

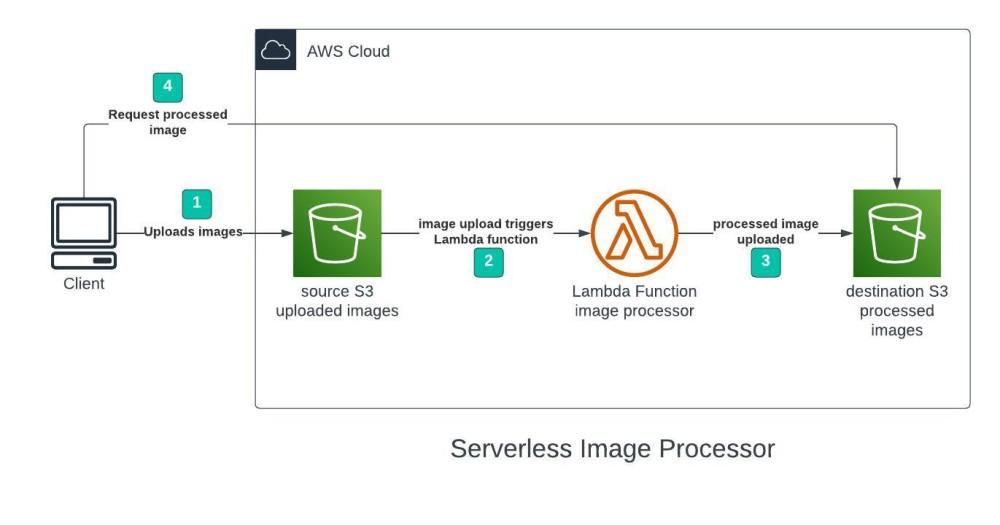
1. [AWS Lambda](https://www.geeksforgeeks.org/how-to-configure-aws-lambda/) is a serverless, event-driven compute service that lets you run code for virtually any type of application or backend service without provisioning or managing servers. Lambda functions run on demand i.e. they execute only when needed and you pay only for what you compute. Lambda is well integrated with may other AWS services. It supports a wide variety of programming languages.

Some common use cases for AWS Lambda are:

File processing:You can use Lambda for processing files as they are uploaded in an S3 bucket or whenever some event triggers the function.

Data and analytics:You can pass a data stream to your Lambda function and then create analysis from that.

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.

Serverless Image Processing Flow:

EXPLANATION:

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.

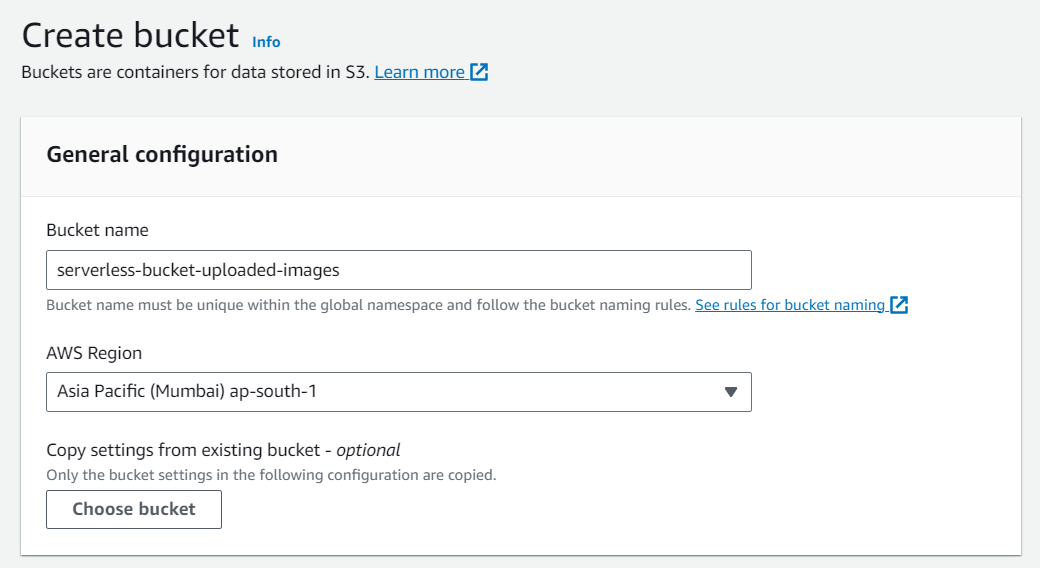
How to create a serverless image processing application that automatically resizes and optimizes images uploaded to an Amazon S3 bucket?

* Perform the given task as given below:
* Step 1 - Creating S3 buckets

We will use two S3 buckets:

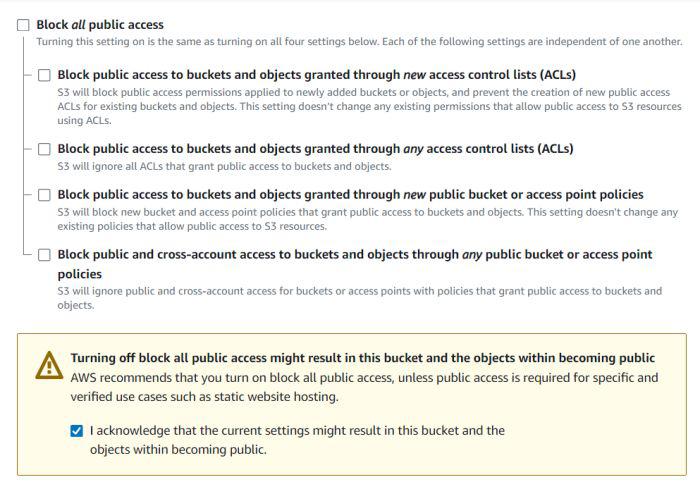
1. source Bucket: For storing uploaded images.
2. destination Bucket: For storing processed images.

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.



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.

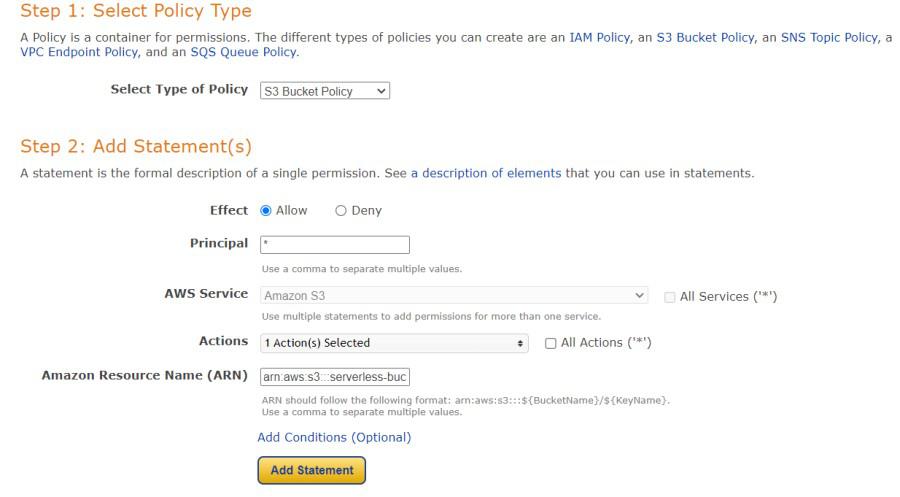
Go to your source bucket and then click on

Permissions tab. Scroll to Bucket Policy tab. Click Edit. You will be redirected to the policy editor. Click on policy generator.

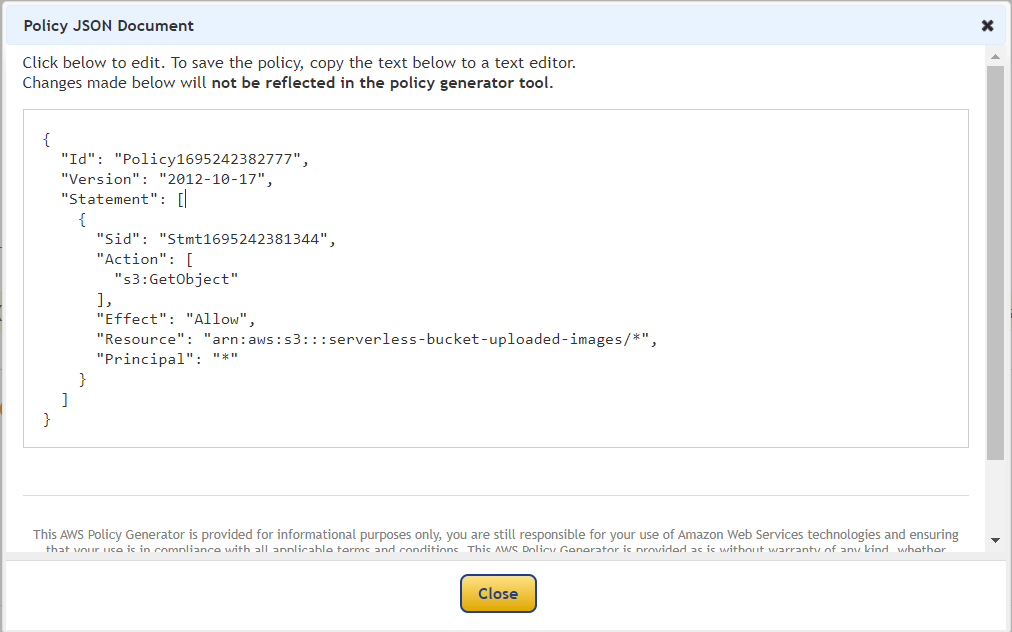
Enter the following settings:

* 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:

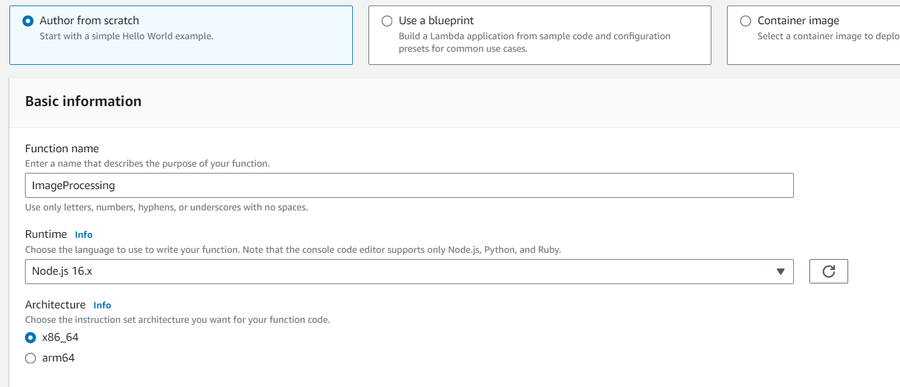
* Type of policy: S3 Bucket Policy
* Effect:Allow
* Principal: \*
* Actions: GetObject, PutObject, and PutObjectAcl
* Amazon Resource Name (ARN): arn:aws:s3:::DESTINATION\_BUCKET\_NAME/\*

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).**

CODE:

const sharp = require("sharp");

const path = require("path");

const AWS = require("aws-sdk");

AWS.config.update({

region: "ap-south-1",

});

const s3 = new AWS.S3();

const processedImageBucket = "serverless-bucket-processed-images";

exports.handler = async (event, context, callback) => { console.log("An object was added to S3 bucket", JSON.stringify(event));

let records = event.Records;

let size = records.length;

for (let index = 0; index < size; index++) {

let record = records[index];

console.log("Record: ", record);

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);

let imageObjectParam = {

Bucket: record.s3.bucket.name,

Key: record.s3.object.key,

};

let imageObject = await s3.getObject(imageObjectParam).promise();

let resized\_thumbnail = await sharp(imageObject.Body)

.resize({

width: 300,

height: 300,

fit: sharp.fit.cover,

})

.withMetadata()

.toBuffer();

console.log("thumbnail image created");

let resized\_coverphoto = await sharp(imageObject.Body)

.resize({

width: 800,

height: 800,

fit: sharp.fit.cover,

})

.withMetadata()

.toBuffer();

console.log("coverphoto image created");

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.

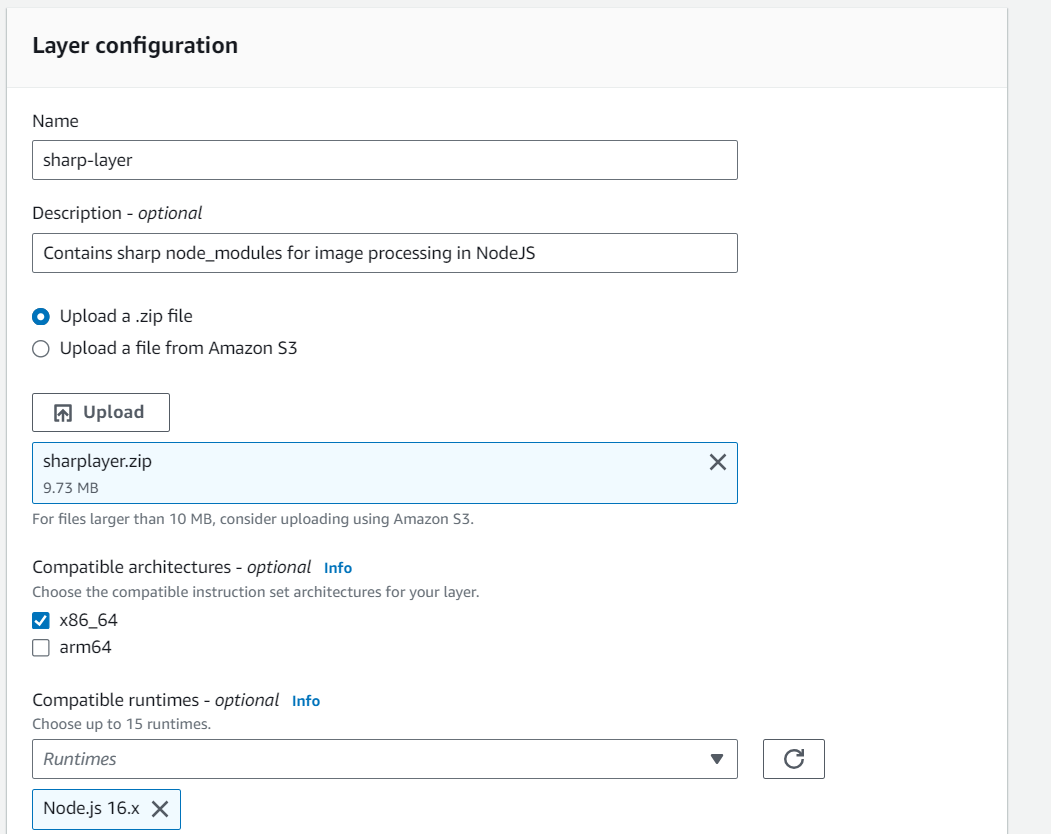
First we need to create a zip file with all the dependencies (node modules in our case) required by our Lambda function.

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).

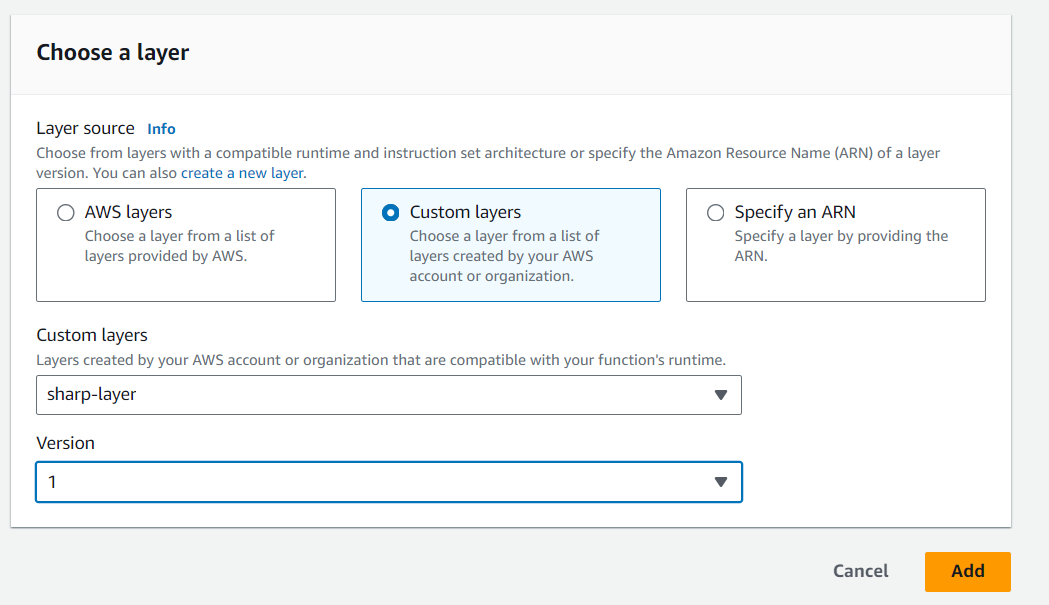
*npm install --arch=x64 --platform=linux sharp*

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

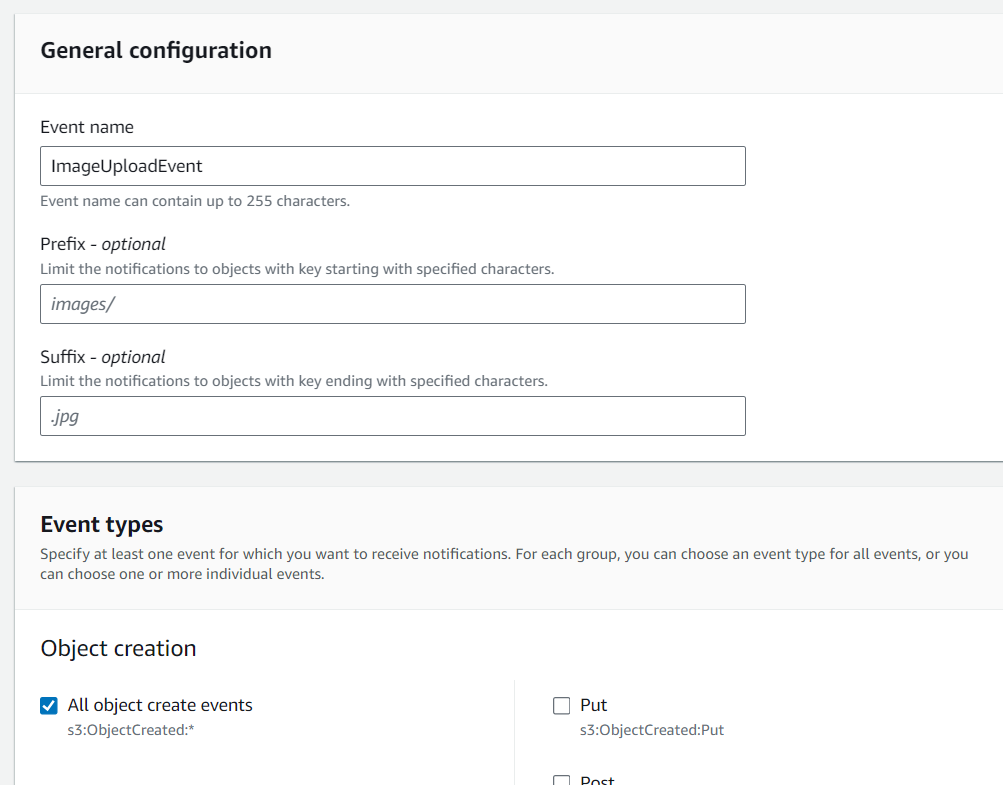


* Step 5 - Creating S3 trigger

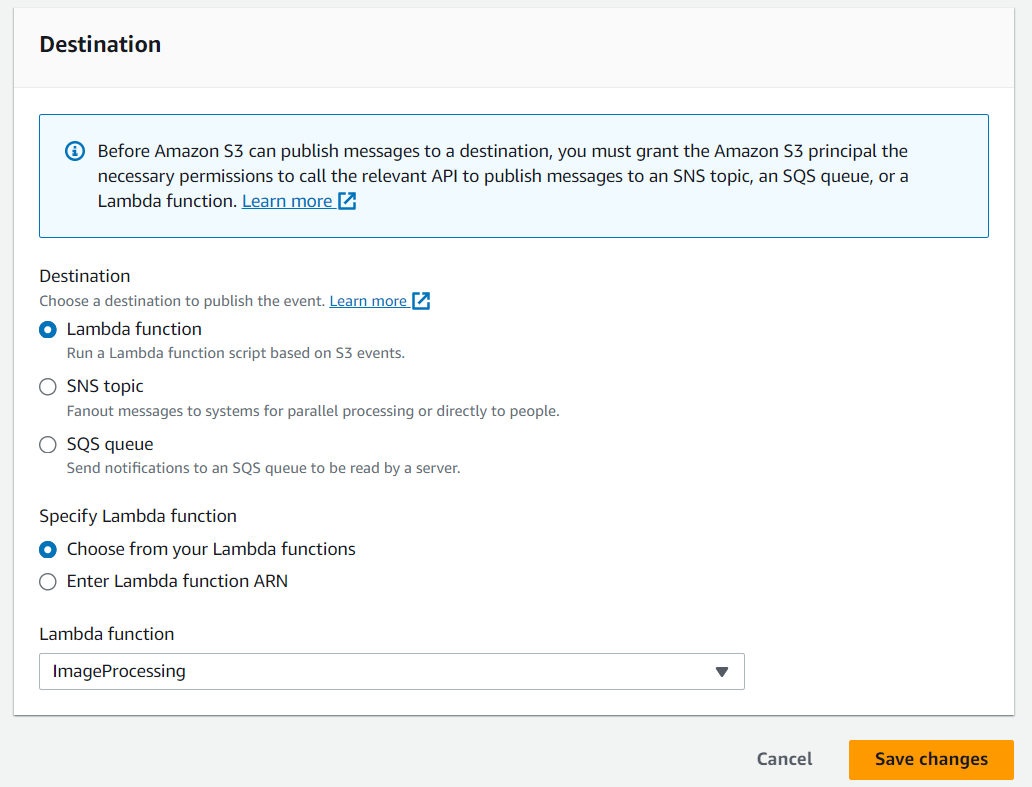
Now we need our Lambda function to know when an image is uploaded to the source bucket. We can do this by adding an event to the source S3 bucket and configure it to get triggered when an image is uploaded to the bucket which in turn invokes the Lambda function.

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 coverphoto).

Congratulations, you just built a serverless Image processing application.

Why Two Different Buckets?

We created two different buckets for this application because whenever the lambda function uploads the processed images to the source bucket it will create 2 triggers and the function will process the processed images once again which in creates 4 triggers. This creates an infinite loop and generated many images, so we created two buckets to prevent an infinite loop.

✅ Real-Life Application

* Profile picture optimization
* E-commerce product image resizing
* CDN pre-processing (for faster load)

✅ Conclusion

* + Importance of serverless
  + Working with AWS hands-on
  + Deploying and testing Lambda with S3 events

Project 2

Deploy a Static Website on AWS

🔍PROJECT OBJECTIVE**:**

To host a static website using AWS services, primarily Amazon S3, without the need for backend processing.

📁 TOOLS & TECHNOLOGIES USED:

* Amazon S3 (Simple Storage Service)
* HTML, CSS, JavaScript (for static content)
* AWS Management Console

🏗️Architecture Diagram:

A diagram of a cloud computing process

AI-generated content may be incorrect.

🧩 Architecture Explanation

1. User Browser
   * This is where users access your website by entering the URL.
2. AWS S3 Bucket with Static Files
   * Your static website files (HTML, CSS, JS) are stored in an Amazon S3 bucket.
   * S3 acts as the web server and file storage for your site.
3. Website Endpoint URL (Hosted Site)
   * AWS S3 provides a public endpoint URL that serves your static content.
   * When a user accesses this URL, the website loads directly from S3.

How to Deploy a Static Website on AWS?

* Perform the given 🔨steps as given below:
* STEP 1:Create a Simple Static Website

Prepare these files:

* index.html (home page)
* style.css (styling)
* script.js (optional JavaScript)

👉 Example index.html:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0"/>

<title>My AWS Static Site</title>

<link rel="stylesheet" href="style.css" />

</head>

<body>

<h1>Welcome to My Static Website on AWS!</h1>

<p>This site is hosted using Amazon S3.</p>

</body>

</html>

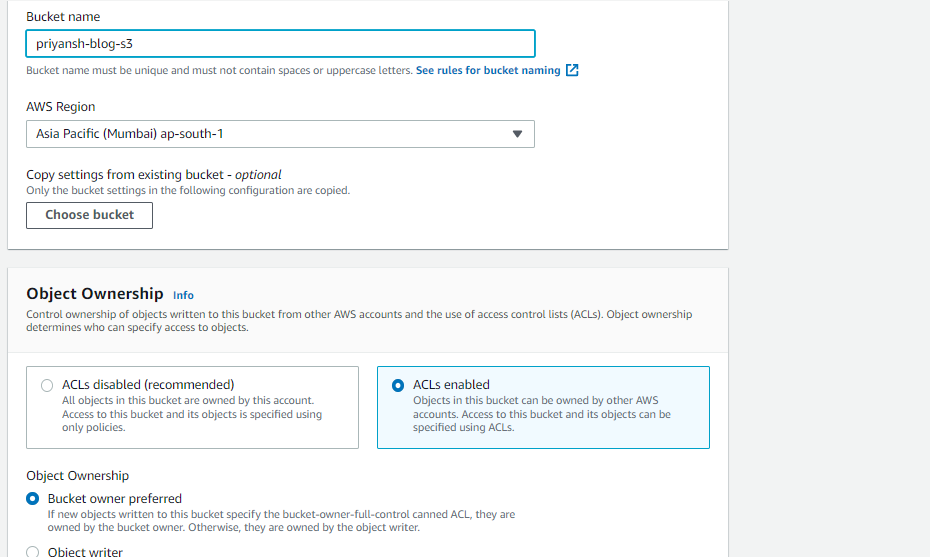
* STEP 2: Create an S3 Bucket

1. Go to the AWS Management Console > S3.
2. Click "Create Bucket".

A screenshot of a computer

AI-generated content may be incorrect.

1. Bucket name: e.g., my-static-website-123



1. Region: Choose your preferred region.
2. Uncheck "Block all public access".

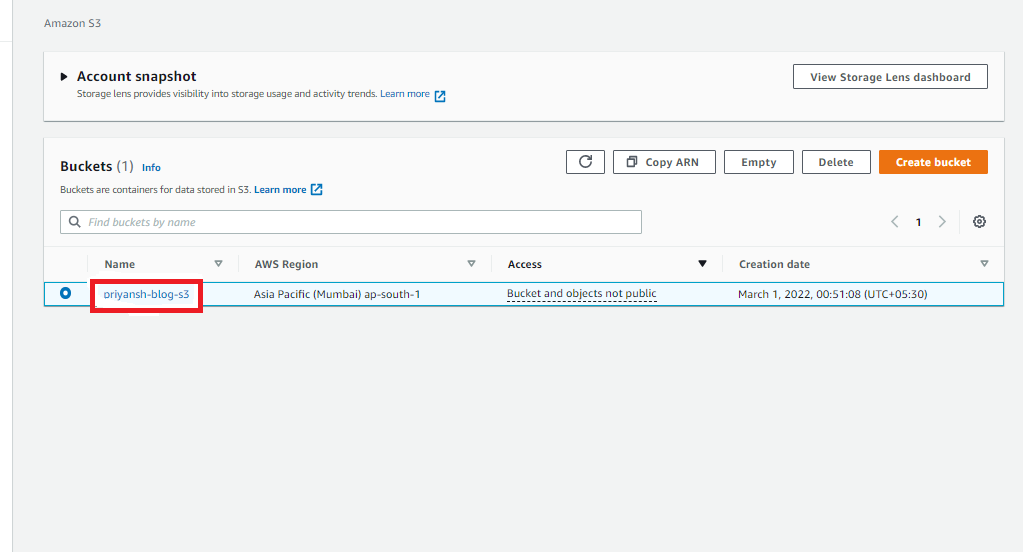
A screenshot of a computer

AI-generated content may be incorrect.

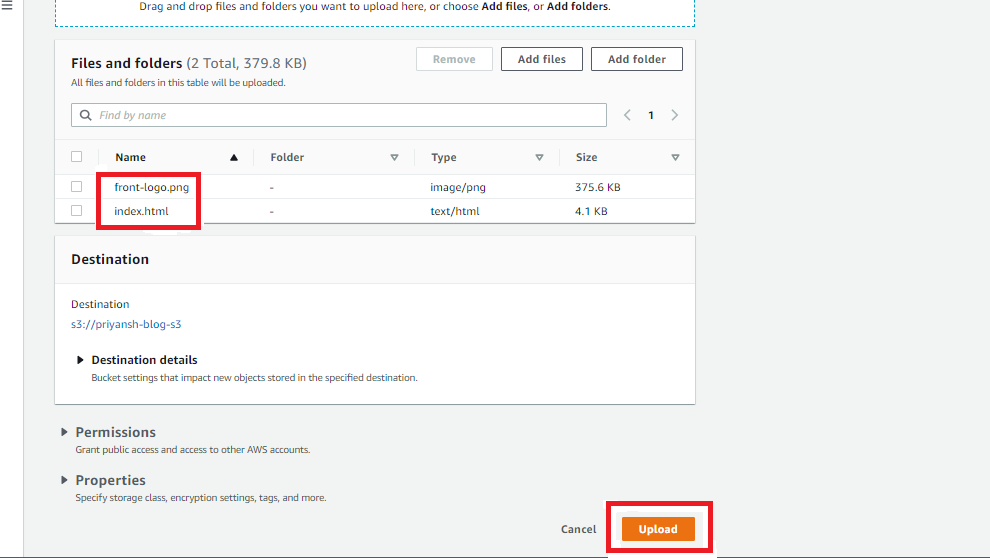
1. Acknowledge the warning checkbox.
2. Click Create Bucket.

* STEP 3: Upload Website Files

1. Open your bucket.

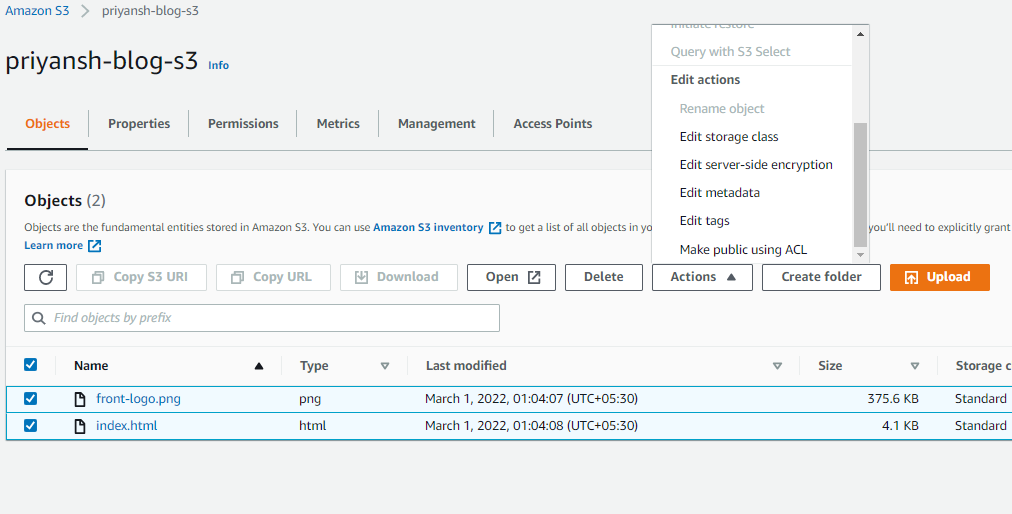


1. Click Upload → Add index.html, style.css, script.js → Click Upload.



* STEP 4:Make Files Public

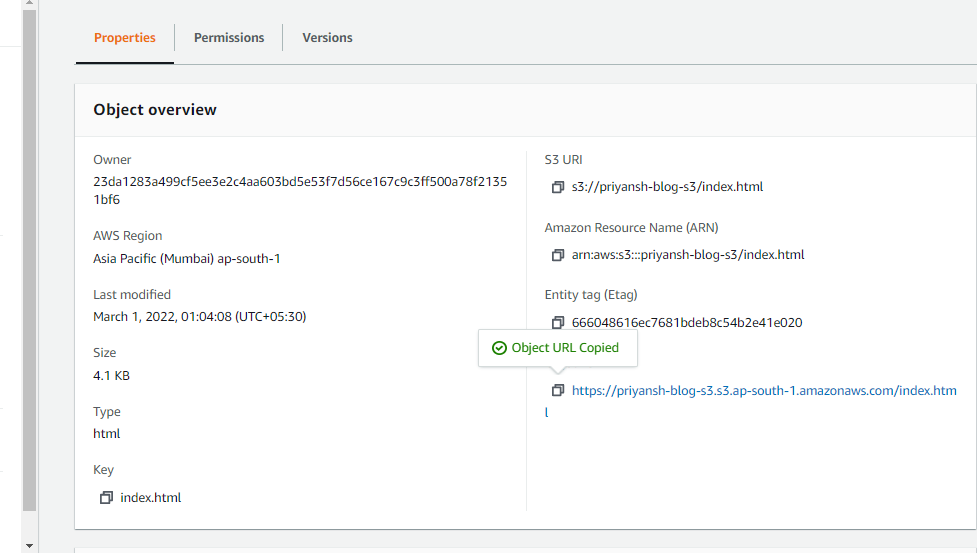
1. Select index.html → Click Actions → Make public.



1. Repeat for other files OR apply a bucket policy.

* STEP 5: Enable Static Website Hosting

1. Go to Properties tab of the bucket.
2. Scroll to Static website hosting → Click Edit.
3. Enable it.
4. Enter index.html as Index document.
5. Save changes.
6. You will now see a website endpoint URL like:  
   <http://my-static-website-123.s3-website.ap-south-1.amazonaws.com>



* STEP 6: 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.to

📊 Advantages of Hosting on S3

* Low-cost solution
* Highly available and scalable
* No server maintenance
* Easy integration with CloudFront and Route 53 for CDN and custom domains.

✅ Real-Life Application

* Personal Portfolio
* Event Landing Page
* Product Promotion Page
* Resume Website
* Educational Static Notes Website

✅ Conclusion

This project provided hands-on experience with AWS S3 and

static website hosting. It demonstrated how cloud platforms

can be used for scalable and cost-effective website deployment.”

Project 3

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

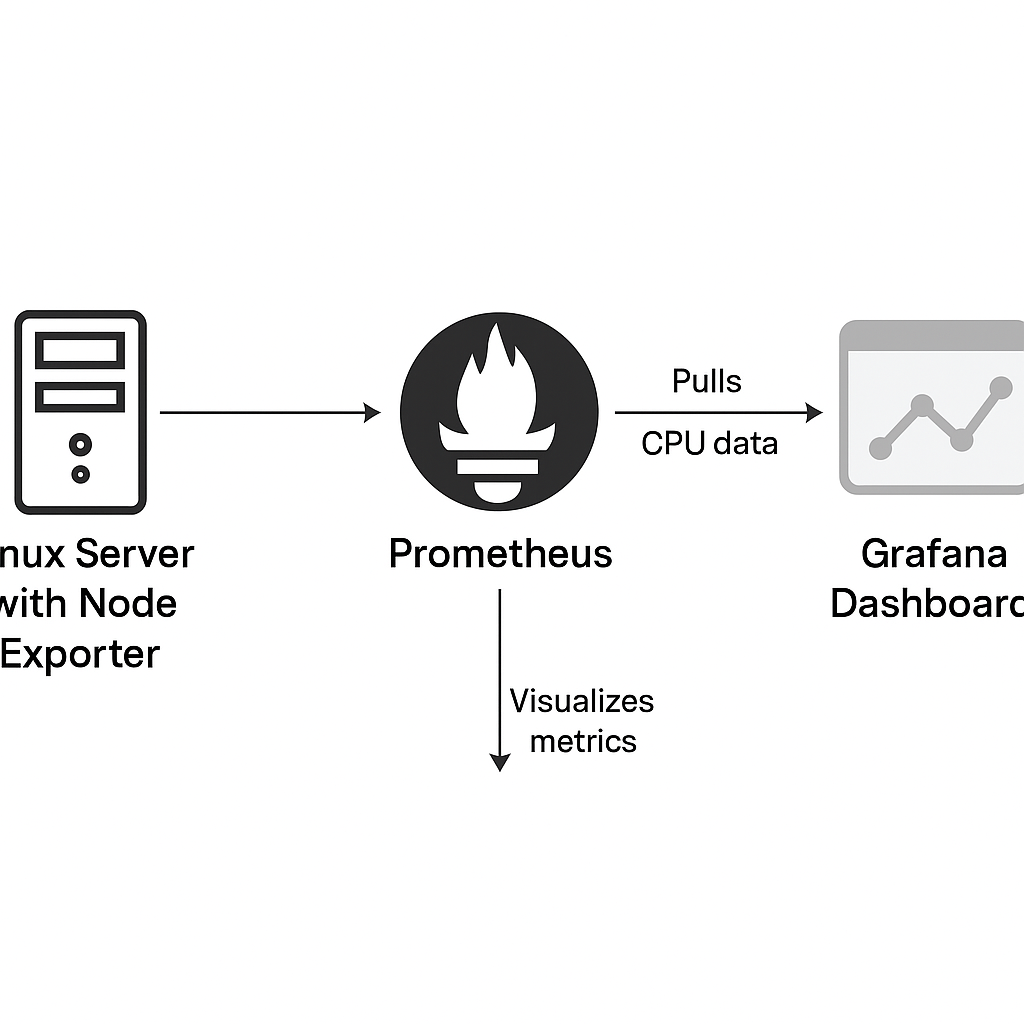
🎯PROJECT OBJECTIVE**:**

To integrate Grafana with a Linux server and monitor CPU utilization in real-time by visualizing metrics using an interactive Grafana graph.

🧰 TECHNOLOGIES USED:

* Linux (Ubuntu)
* Grafana
* Prometheus (time-series database)
* Node Exporter (for collecting Linux system metrics)
* Web Browser (for accessing Grafana dashboard)

🏗️Architecture Diagram:



🧩 Architecture Explanation

1. Linux Server with Node Exporter
   * Collects system metrics like CPU, memory, disk, etc.
   * Node Exporter exposes these metrics on port 9100.
2. Prometheus
   * Acts as a time-series database and monitoring tool.
   * Periodically pulls data from Node Exporter on the Linux server.
   * Stores and organizes metrics for querying.
3. Grafana Dashboard
   * Connects to Prometheus as a data source.
   * Visualizes metrics (like CPU usage) in real-time through customizable graphs and panels.

🔁 Data Flow:  
Linux Server ➝ Node Exporter ➝ Prometheus ➝ Grafana (Visualization)

🧾 Steps to Perform the Lab:

* Step 1: Update Linux System

*sudo apt update && sudo apt upgrade -y # For Ubuntu*

* Step 2: Install Node Exporter (on Linux Server)

*Wget https://github.com/prometheus/node\_exporter/releases/download/v1.7.0/node\_exporter-1.7.0.linux-amd64.tar.gz*

*tar xvf node\_exporter-1.7.0.linux-amd64.tar.gz*

*cd node\_exporter-1.7.0.linux-amd64/*

*./node\_exporter &*

* Node Exporter runs on <http://localhost:9100/metrics>
* Step 3: Install Prometheus

*wget https://github.com/prometheus/prometheus/releases/download/v2.52.0/prometheus-2.52.0.linux-amd64.tar.gz*

*tar xvf prometheus-2.52.0.linux-amd64.tar.gz*

*cd prometheus-2.52.0.linux-amd64/*

**Edit the prometheus.yml file:**

Add Node Exporter job:

*scrape\_configs:*

*- job\_name: 'node\_exporter'*

*static\_configs:*

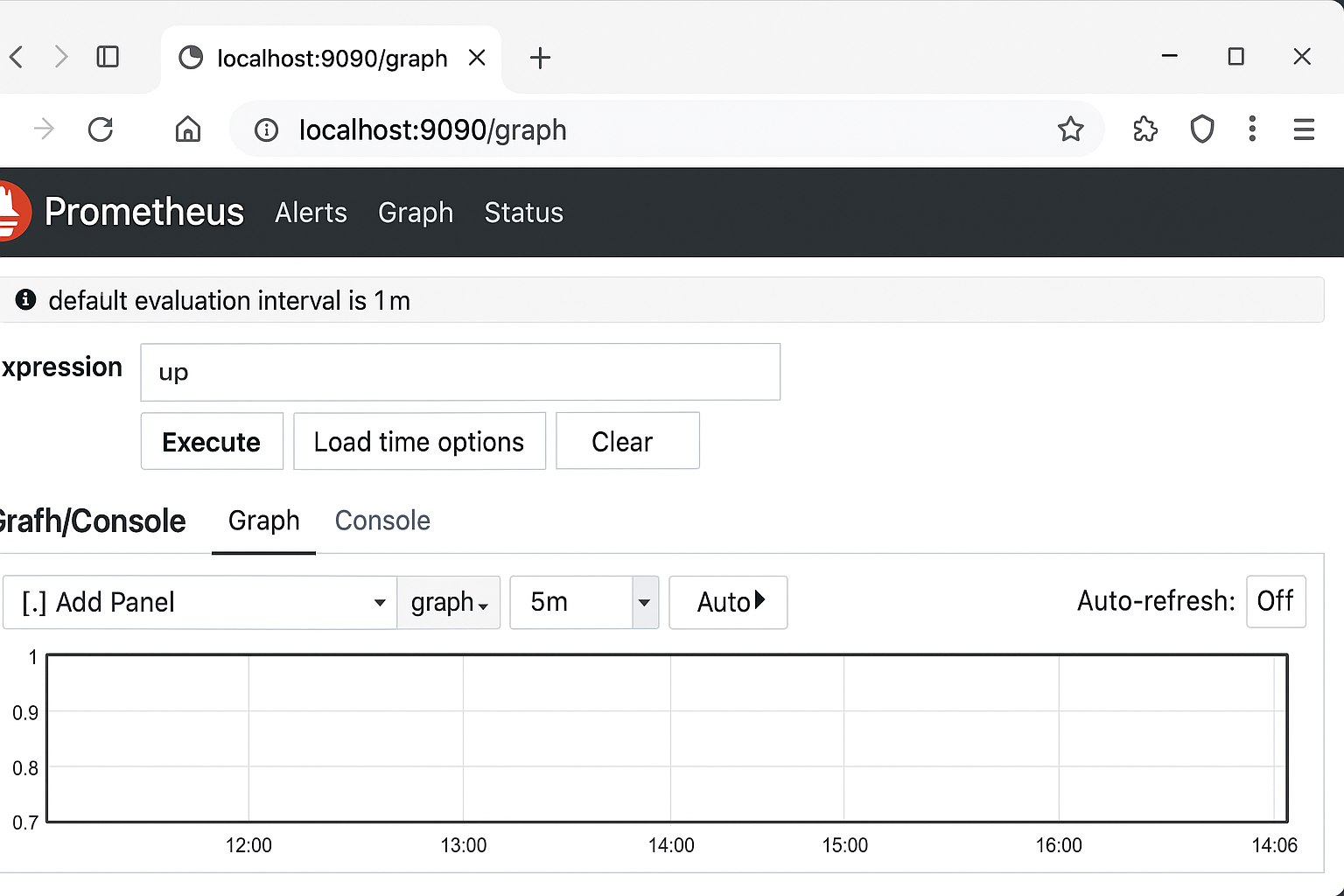
*- targets: ['localhost:9100']*

**Start Prometheus:**

*./prometheus --config.file=prometheus.yml &*

* Prometheus is accessible at http://localhost:9090

“After successfully running Prometheus, navigate to http://localhost:9090. You’ll see the Prometheus interface where you can query and test metrics. Here's how it looks:”



* Step 4: Install Grafana

*# For Ubuntu*

*sudo apt install -y software-properties-common*

*sudo add-apt-repository "deb https://packages.grafana.com/oss/deb stable main"*

*wget -q -O - https://packages.grafana.com/gpg.key | sudo apt-key add -*

*sudo apt update*

*sudo apt install grafana -y*

*sudo systemctl start grafana-server*

*sudo systemctl enable grafana-server*

* Access Grafana at: http://localhost:3000
* Default login: admin / admin
* Step 5: Configure Prometheus Data Source in Grafana

1. Login to Grafana (http://localhost:3000)
2. Go to "Configuration" > "Data Sources"
3. Click "Add Data Source"
4. Choose Prometheus
5. Set URL to http://localhost:9090
6. Click "Save & Test"

* Step 6: Create CPU Utilization Dashboard

1. Go to "Create" > "Dashboard"
2. Click "Add new panel"
3. In the Query editor, select:

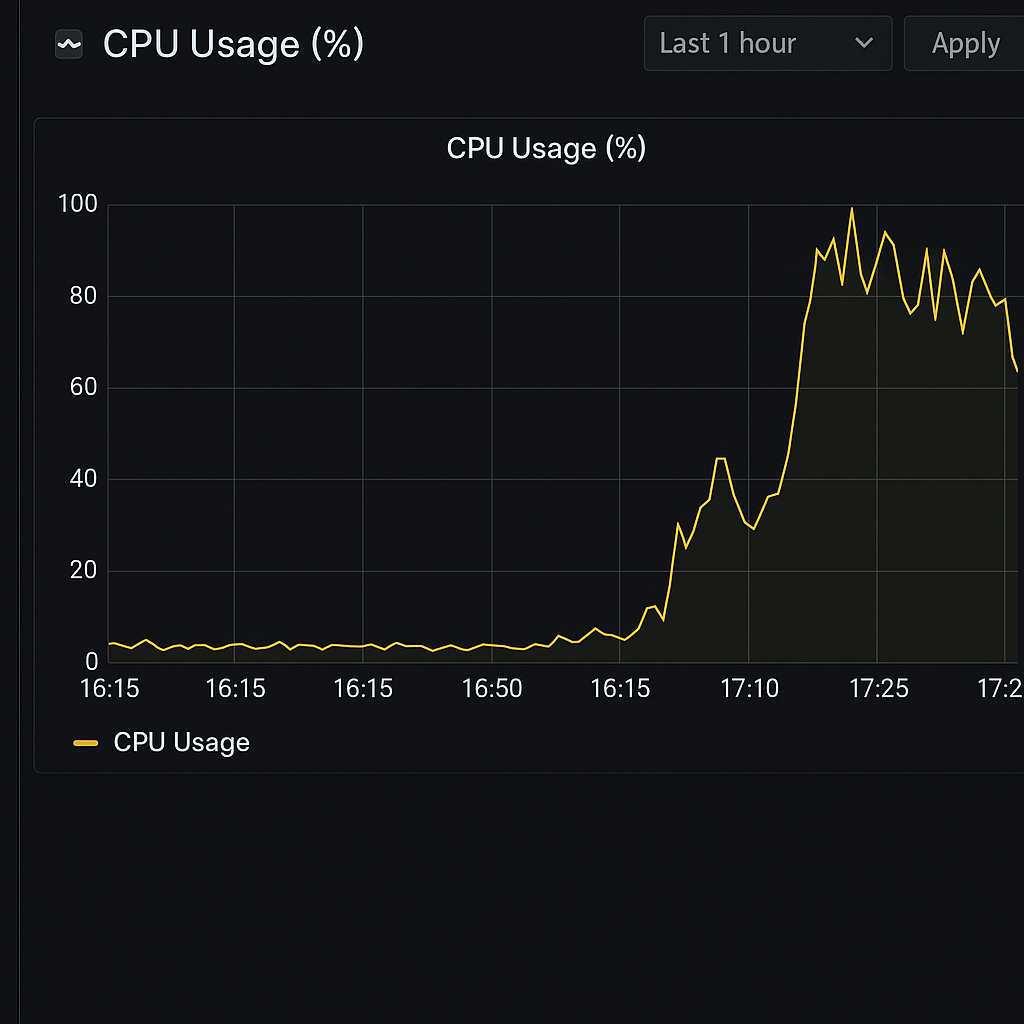
*rate(node\_cpu\_seconds\_total{mode="user"}[1m]) \* 100*

4.Set Visualization: Line Graph

5.Title it: CPU Usage (%)

6.Click Apply

“Once the query is added and panel settings are configured, Grafana will generate a CPU Usage graph in real time as shown below:”



**📚** Real-Life Application

* Real-time server monitoring
* Early detection of performance issues
* Infrastructure health checks
* Resource usage analysis for scaling decisions

💡Advantages of Grafana + Prometheus

* Open-source and free
* Real-time monitoring
* Custom dashboards and alerts
* Easy integration with cloud and on-premise systems

📌Conclusion

This project demonstrates the power of combining Prometheus and Grafana to monitor server performance. By visualizing CPU metrics in real-time, system admins can detect anomalies early and ensure optimal server health.