**PROJECT-**

**Serverless Image Processing With Aws Lambda**

A logo on a blue background

AI-generated content may be incorrect.

**Table of Contents**

1. Introduction
2. Objective
3. Why I Built This Project
4. Technology Used
5. Project Folder Structure
6. Setup Checklist
7. Processing Flow
8. Step-by-Step Implementation
9. Conclusion

# Introduction

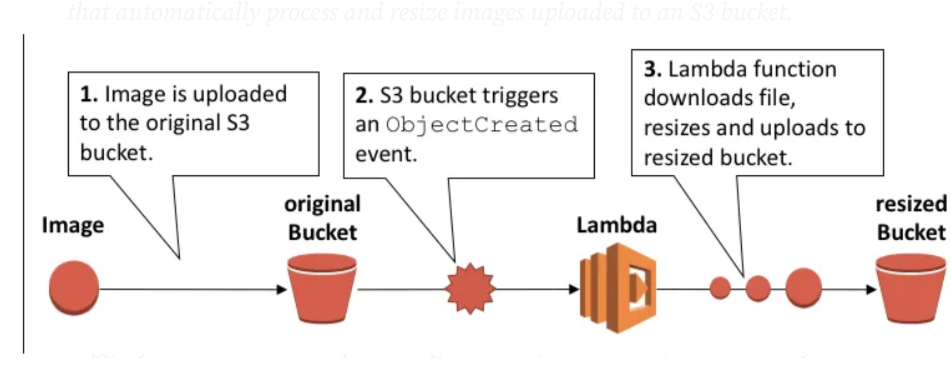
*In today’s fast-paced digital world, automating image processing tasks can significantly streamline workflows and enhance efficiency.* ***AWS Lambda, a serverless compute service, paired with Amazon S3, a scalable storage solution, offers a powerful combination for automating image******processing tasks.***

This document describes a **project** that focuses on building a **Serverless Image Processing Application** using **AWS Cloud Services**. The system will monitor an

**S3 bucket** for **image uploads**, trigger a **Lambda function** to **resize and optimize those images**, and **store the optimized version in a separate S3 bucket.**

# 2. Objective

***To create a Lambda function and configure a trigger for an Amazon Simple Storage Service (Amazon S3) bucket. Every time that you add an object to your Amazon S3 bucket, your function runs and outputs the object type to Amazon CloudWatch Logs.****You can use a Lambda function with an Amazon S3 trigger to perform many types of file-processing tasks. For example, you can use a Lambda function to create a thumbnail whenever an image file is uploaded to your Amazon S3 bucket or to convert uploaded documents into different formats*



# 3. Why I Built This Project

* To gain hands-on experience with AWS Lambda and event-driven architecture.
* To automate repetitive image handling tasks.
* Learn and implement event-driven architecture via S3 triggers

# 4. Technology Used

| * + **Technology** | * + **Description** |
| --- | --- |
| * + AWS S3 | * + Storage buckets for image uploads |
| * + AWS Lambda | * + Serverless compute to process images |
| * + AWS CloudWatch | * + Monitoring and logging |
| * + IAM | * + Access permissions |
| * + Sharp (Node.js) | * + Image processing library |
| * + Boto3 (Python) / AWS SDK | * + AWS service interaction |

# 5. Folder Structure

Project-1/  
├── image-resizer/  
│ ├── lambda\_function.py  
│ ├── requirements.txt  
│ └── image-resizer.zip  
│  
├── image-uploader-frontend/  
│ ├── index.html  
│ ├── script.js  
│ └── style.css

# 6. Setup Checklist

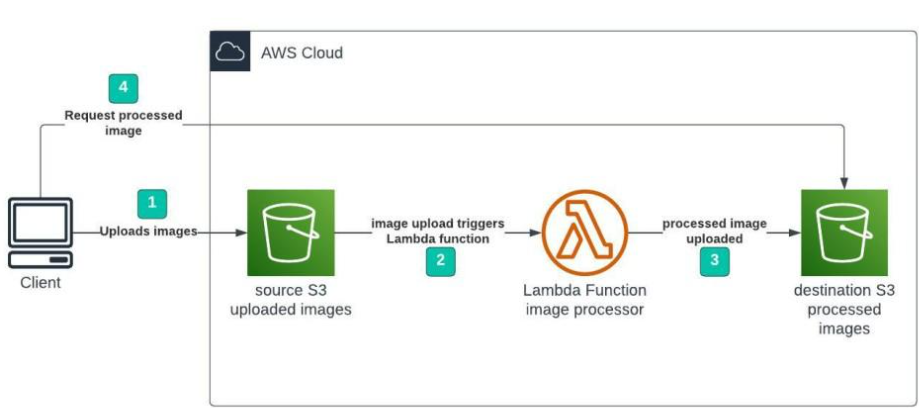
## A. Hardware

* Window 10 or above
* 4 GB RAM or higher

## B. Software

* + Python Installed
  + AWS CLI Configured
  + Notepad / VS Code

# Steps7. Processing Flow



# 8. Implementation Steps

Let’s move forward with the **step-by-step implementation** of our serverless image optimization system using AWS services.

## 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.

uA screenshot of a bucket

AI-generated content may be incorrect.



# Step 2 – Configuring S3 bucket policy

**In ‘Block Public Access settings for this bucket’ section disable “block all public access”.**

**A screenshot of a bucket

AI-generated content may be incorrect.**

# Step 3 – Creating police in Iam

**Go to AWS I am console.**

**Navigate to policies section.**

**Click Create policies in (JSON) and name it “imgbucketpoilicy”. Leave all other settings as default.**

**Create the policy.**

**Policy:**

**"Version": "2012-10-17",**

**"Statement": [**

**{**

**"Effect": "Allow",**

**"Action": [**

**"logs:PutLogEvents",**

**"logs:CreateLogGroup",**

**"logs:CreateLogStream"**

**],**

**"Resource": "arn:aws:logs:\*:\*:\*"**

**},**

**{**

**"Effect": "Allow",**

**"Action": ["s3:GetObject"],**

**"Resource": "arn:aws:s3:::BUCKET\_NAME/\*"**

**},**

**{**

**"Effect": "Allow",**

**"Action": ["s3:PutObject"],**

**"Resource": "arn:aws:s3:::DEST\_BUCKET/\*"**

**}**

**]**

**}**

**\*BUCKET\_NAME = SOURSE BUCKET NAME**

**\*DEST\_BUCKET = DESTINATION BUCKET NAME**

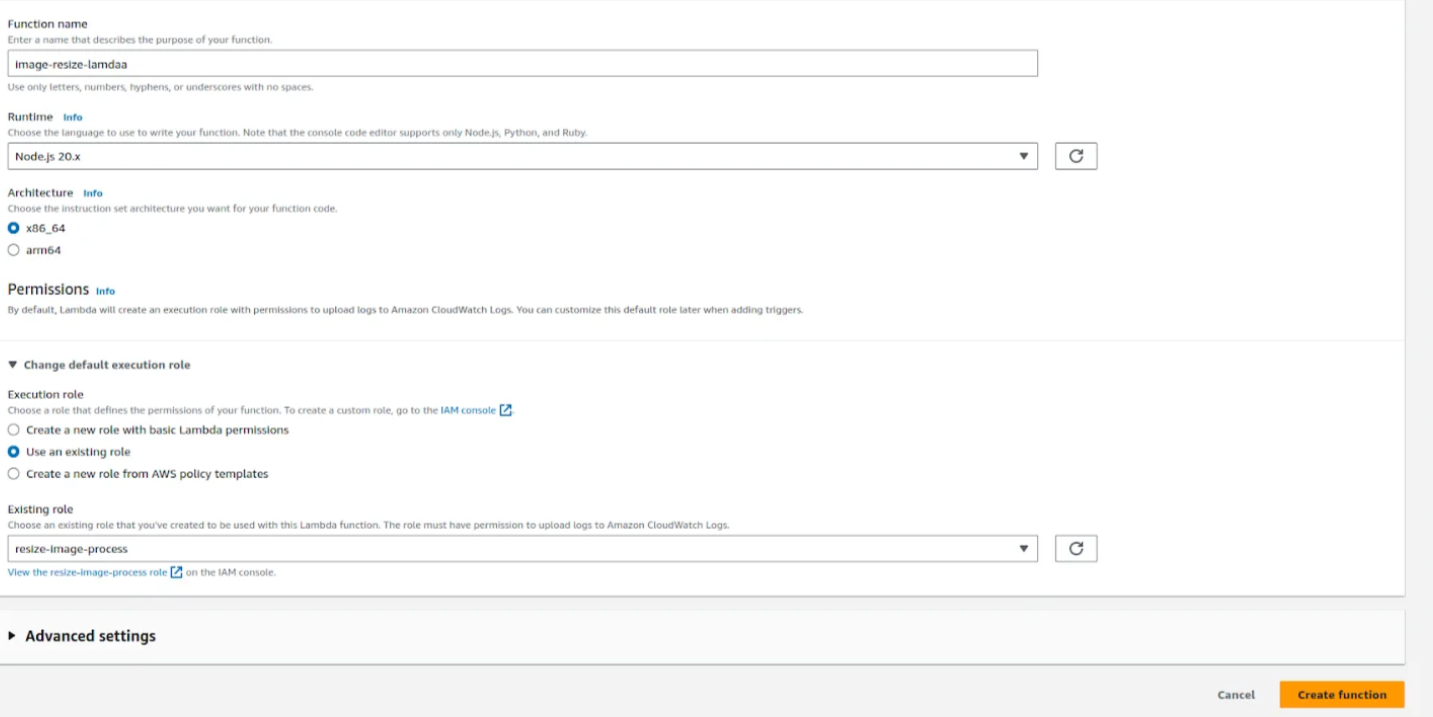
# Step 4 –Creating Role In Iam

**Following Steps are Follows:**

* **Go to aws I am console**
* **Create role**
* **name imageresizerlambdarole**
* **Use case – Lambda**
* **Select-ImageBucketPolicy**
* **Then create role**

# Step 5 – CREATING LAMBDA FUNCTION

* Go to AWS Lambda console. Navigate to Functions section. Click Create Function and name it “image-resize-lambdaa”. Select runtime as “NodeJS 16.x” and architecture as “x86\_64”.
* Change default execution role
* Leave all other settings as default.
* Create the function.

****

# ****Step 6 -Upload zip file in Lambda function****

**Create a ZIP file containing:**

* **Your index.js (Lambda handler)**
* **package.json**
* **node\_modules/ including sharp  
  So it can be directly uploaded to Lambda and run without errors.**

**[Install Node.js (LTS version recommended)  
✔ Install zip utility (default on macOS/Linux, PowerShell or 7-Zip for Windows]**

**[For Linux (Ubuntu/Debian-based)**

**✅ Option 1: NodeSource (stable version)**

**curl -fsSL https://deb.nodesource.com/setup\_18.x | sudo -E bash -**

**sudo apt-get install -y nodejs**

**✅ Option 2: Snapsudo snap install node –classic]**

**To Install zip & unzip:**

**sudo apt update**

**sudo apt install zip unzip**

**To zip a folder:**

**zip -r myarchive.zip myfolder/]**

**Step-by-Step Instructions for zip file:**

**1. Create project folder:**

*mkdir image-resizer*

*cd image-resizer*

2. **Create Lambda handler file:**

touch index.js

**Paste this code into index.js:**

js

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

const S3 = new AWS.S3();

const Sharp = require('sharp');

exports.handler = async (event) => {

const record = event.Records[0];

const bucket = record.s3.bucket.name;

const key = decodeURIComponent(record.s3.object.key.replace(/\+/g, ' '));

const outputBucket = process.env.OUTPUT\_BUCKET;

try {

const inputImage = await S3.getObject({ Bucket: bucket, Key: key }).promise();

const resizedImage = await Sharp(inputImage.Body).resize({ width: 800 }).toBuffer();

await S3.putObject({

Bucket: outputBucket,

Key: `resized-${key}`,

Body: resizedImage,

ContentType: 'image/jpeg'

}).promise();

return { statusCode: 200, body: 'Success' };

} catch (err) {

console.error(err);

return { statusCode: 500, body: 'Error' };

}

};

**3. Create package.json and install sharp:**

npm init -y

npm install sharp

**4. Create ZIP including everything:**

**zip -r image-resizer.zip index.js node\_modules package.json**

* In your new function page, scroll down to the **Code** section
* Click the dropdown next to **"Upload from"** and select **.zip file**
* Click **Upload**, then **Browse** and select your image-resizer.zip file
* Click **Save**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# Step 7 - Edit environment variables

* select edit
* Now select the destination bucket
* Save it

# Step 8 – Test Lambda Function

* Go to AWS Lambda console. Navigate to Functions section.
* open function then will be created
* open test console
* template=s3-put

A screenshot of a computer

AI-generated content may be incorrect.

**EVENT JSON**

In event json we can change only 3 value

\*name, arn, key



"name":"demo-user-images-bucket",->

"arn": "arn:aws:s3::: demo-user-images-bucket "->

"key": "golf-course.jpg",->

A screenshot of a computer

AI-generated content may be incorrect.

**Now we can test**

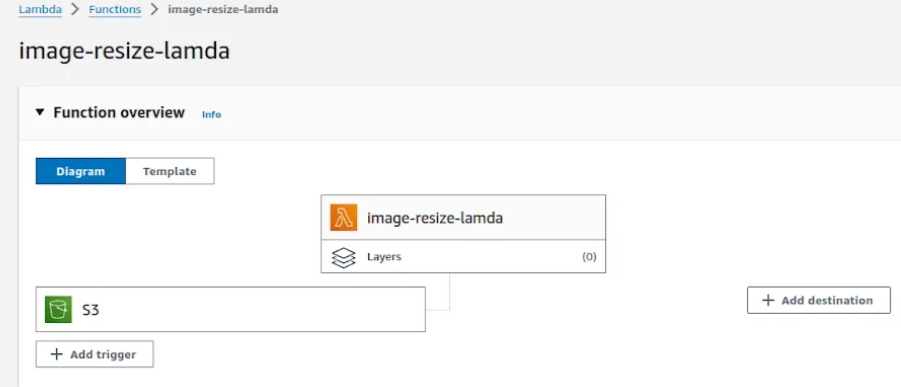
A screenshot of a computer

AI-generated content may be incorrect.

* The code test is successfully run

# Step 8 – Creating S3 trigger

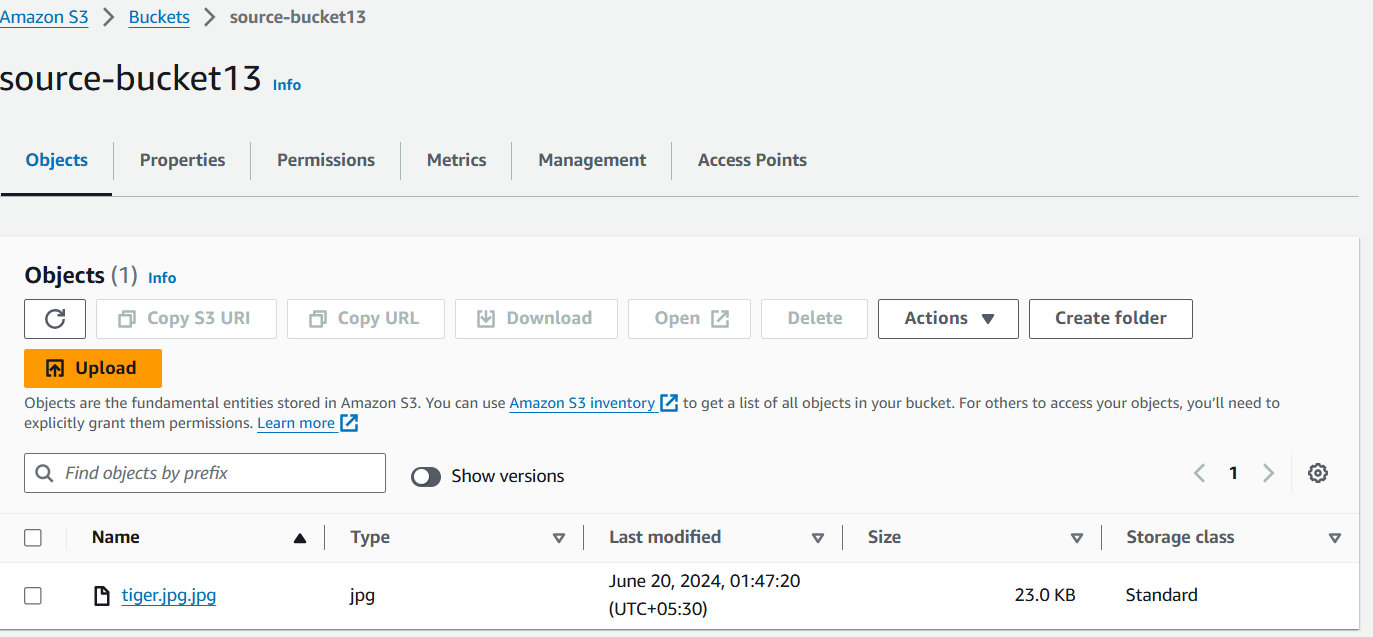
* Add trigger
* Select s3
* choose source Bucket name
* Now Add



A screenshot of a computer

AI-generated content may be incorrect.

**Upload image in source Bucket**



****

**Original image**

**Destination Bucket**

A screenshot of a computer

AI-generated content may be incorrect.

A tiger lying on a rock

AI-generated content may be incorrect..

THE RESIZED IMAGE IS HERE.

# 10. Conclusion

Automating image processing with AWS Lambda and S3 provides a scalable and efficient solution for handling image-related tasks. Whether you need to process images as they are uploaded or resize them for specific use cases, this step-by-step guide should help you get started with creating Lambda functions tailored to your needs. Experiment with different configurations and functionalities to optimize the automation of your image processing workflow on AWS.

THANKYOU

SUBMITTED TO

**ABHINAV SIR**

**Submitted by:**

ISHMEET KAUR

BCA 2ND SEM /SECTION-D

PROJECT

ON

Static

Website

**Topic-**

**Deploy a Static Website on AWS**

In this project, we will learn how to create a static website and deploy it using AWS services. A static website is a site that consists of HTML,CSS, and JavaScript files, and it doesn't require server-side processing or a database.

**Table of contents**

**1 Introduction.**

1.1 Setup Checklist for Static website Project

1.2 Instructions

**2 Problem Statement.**

2.1 Objective

2.2 Abstract of the project

2.3 Technology used

**3 Implementation**

3.1 Summary of static website

3.2 Process of creating a website

**4 Appendix (**Source files**)**

**1 Introduction**

On Deploying a static website, it involves hosting pre-built web pages (HTML, CSS, JavaScript) on a server for delivery to users, without requiring server-side processing for content generation.

A static website is a site that consists of HTML, CSS, and JavaScript files, and it doesn't require server-side processing or a database.

1.1 Setup Checklist for Static website Project

**# Hardware**

**~ Processor-** AMD A4-9125 RADEON R3, 4 COMPUTE CORES 2C+2G 2.30 GHz.

**~Edition-** Window 10 pro

**~Memory-** 4 G**#**

**Software~**Visual Studio Code

* 1. **Instructions**

>The code modules in the Static website project should follow all the coding standards.

**>** In this website I have created 1 folder, 3 files of .html and 1 file of .css .

**>** This website contains all rights.

**Problem Statement**

**2.1 Objective**

It is a project given in **AWS** training by trainer.

**2.2 Abstract of the project**

On Deploying a static website involves hosting pre-built HTML, CSS, and JavaScript files on a server or a cloud service, allowing for fast loading times and efficient content delivery. The process typically includes preparing the website files, choosing a deployment platform (like AWS S3, Netlify, or DigitalOcean), configuring the platform, and then pushing the website's code to the chosen destination. Optimizing the website's assets (images, code) is also crucial for performance.

**2.3 Technology used**

**#** HTML

**#** CSS

**#** JavaScript

**3.Implementation**

**3.1 Summary of static website**

This website is related to travelling blog, in this website we learn different types of things related to different cultures. In this website there are some Famous cities and countries with their famous spots is also present.

**3.2 Process of creating a website**

**A diagram of a software project

AI-generated content may be incorrect.**

**1. Naming Your S3 Bucket**

Before you begin hosting your awesome static website out of S3, you need a bucket first. It is critical that your bucket has the same name as your domain name.

If your website domain is www.my-awesome-site.com, then your bucket name must be www.my-awesome-site.com.

The reasoning for this has to do with how requests are routed to S3. The request comes into the bucket, and then S3 uses the Host header in the request to route to the appropriate bucket.

Host: www.my-awesome-site.com

**2. Configuring Your S3 Bucket for Static Website Hosting**

Alright, you have your bucket. It has the same name as your domain name, yes? Time to configure the bucket for static website hosting.

Guess what? Turning on static website hosting for your bucket is as simple as a few clicks in the AWS Console.

* Navigate to S3 in the AWS Console.
* Click into your bucket.
* Click the “Properties” section.
* Click the “Static website hosting” option.
* Select “Use this bucket to host a website”.
* Enter “index.html” as the Index document.
* Or if you are all about command lines and would rather not have a graphical user interface (GUI) in your way, this AWS CLI command turns website hosting on for your bucket.

aws s3 website s3://www.my-awesome-site.com/ --index-document index.html --error-document error.html

Your bucket is configured for static website hosting, and you now have an S3 website url like this http://www.my-awesome-site.com.s3-website-us-east-1.amazonaws.com/.

Your bucket serves your static website, so it must be accessible to anyone in the world. This is referred to as anonymous access to the bucket.

By default, any new buckets created in an AWS account deny you the ability to add a public access bucket policy. This is in response to the recent leaky buckets where private information has been exposed to bad actors.

**However, for our use case, we need a public access bucket policy. To allow this you must complete the following steps before adding your bucket policy.**

* Click into your bucket.
* Select the “Permissions” tab at the top.
* Under “Public Access Settings” we want to click “Edit”.
* Change “Block new public bucket policies”, “Block public and cross-account access if bucket has public policies”, and “Block new public ACLs and uploading public objects” to be false and Save.
* You must complete this step before adding the bucket policy to your static website bucket.
* Now you must update the Bucket Policy of your bucket to have public read access to anyone in the world. The steps to update the policy of your bucket in the AWS Console are as follows:
* Navigate to S3 in the AWS Console.
* Click into your bucket.
* Click the “Permissions” section.
* Select “Bucket Policy”.

**Add the following Bucket Policy and then Save**

**{**

"Version": "2008-10-17",

"Id": "PolicyForPublicWebsiteContent",

"Statement": [

{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": {

"AWS": "\*"

},

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::www.my-awesome-site.com/\*"

}

]

}

Or for the command line fans out there, if policy.json is the above bucket policy, then use:

aws s3api put-bucket-policy --bucket www.my-awesome-site.com --policy file://policy.json

It is important to note the "Principal": { "AWS": "\*" }section of the bucket policy. This part of the policy opens up your bucket to anyone in the world.

Any object in this bucket is available to the public via the S3 website url. Don’t put anything in this bucket that you’re not willing to share with the world.

**3. Adding A CNAME Record For Your Bucket Url**

In order for a user to load your S3 website you’ll need to provide mapping from your domain name www.my-awesome-site.com, to your S3 website url www.my-awesome-site.com.s3-website-us-east-1.amazonaws.com.

This mapping is often referred to as a CNAME record inside of your Domain Name Servers (DNS) records.

www.my-awesome-site.com CNAME www.my-awesome-site.com.s3-website-us-east-1.amazonaws.com

The process to complete this step varies depending on who your DNS provider is. In general this is what you are looking for within your DNS provider:

**Create a record for a host like www**

The record type must be CNAME (Canonical name)

The value must be your S3 website url www.my-awesome-site.com.s3-website-us-east-1.amazonaws.com

**4. Uploading Your Static Website**

Your bucket is configured for static website hosting. You have a CNAME record in your DNS records that resolves to the S3 website url? Awesome, it’s showtime then.

Remember S3 is a flat object store, which means each object in the bucket represents a key without any hierarchy. While the AWS S3 Console makes you believe there is a directory structure, there isn’t. Everything stored in S3 is keys with prefixes.

**This is important to note because if you have a website structure like this:**

about/

index.html

contact/

index.html

css/

styles.min.css

...

...

Index

THANKYOU

BY-ISHMEET KAUR

PROJECT-3

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

INDEX



1. INTRODUCTION
2. OBJECTIVE
3. WHY I BUILT THIS PROJECT?
4. TECHNOLOGY USED /SETUP CHECKLIST
5. STEP-BY-STEP IMPLEMENTATION
6. OUTPUT & TESTING
7. CONCLUSION

# INTRODUCTION

**Grafana is an open-source data visualization and monitoring platform that allows you to create dashboards to visualize your data and metrics. It is a popular choice for visualizing time series data, and it integrates with a wide range of data sources, including Prometheus, Elasticsearch, and InfluxDB.**

**Grafana has a user-friendly interface that allows you to create and customize dashboards with panels that display your data in a variety of formats, including graphs, gauges, and tables. You can also use Grafana to set up alerts that trigger notifications when certain conditions are met.**

**In addition to its core features, Grafana has a rich ecosystem of plugins and integrations that extend its functionality. For example, you can use Grafana to integrate with other tools and services, such as Slack or PagerDuty, to receive alerts and notifications. Grafana is a powerful tool for visualizing and monitoring your data and metrics, and it is widely used in a variety of industries and contexts.**

**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.**

# 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 user-friendly 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.

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) wgethttps://github.com/prometheus/node\_exporter/releases/download /vX.

Y.Z/node\_exporter-X.Y.Z.linux-amd64.tar.gz Example:

wgethttps://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

sudouseradd -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

sudosystemctl daemon-reload sudosystemctl enable node\_exportersudosystemctl start node\_exporter

1. **Verify Node Exporter Status:**

Bash

sudosystemctl 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

sudoufw allow 9100/tcpsudoufw 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., prometheusX.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"  }[5m])) \* 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 |
| killall 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**

THANKYOU

**SUBMITTED TO**

**ABHINAV SIR**

**SUBMITTED BY-**

**ISHMEET KAUR**

**BCA2ND SEM**

**SEC D**

PROJECT-3

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

INDEX



1. INTRODUCTION
2. OBJECTIVE
3. WHY I BUILT THIS PROJECT?
4. TECHNOLOGY USED /SETUP CHECKLIST
5. STEP-BY-STEP IMPLEMENTATION
6. OUTPUT & TESTING
7. CONCLUSION