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



**BACHELOR OF COMPUTER APPLICATION**

**BATCH(2024-2027)**

# SUBMITTED BY:- SUBMITTED TO:-

# SANJANA GAUTAM ABHINAV SIR

# BCA SEMESTER-2

# Team Name :- DISTRIBUTED MINDS

**PROJECT-1**

**Serverless Image Processing**

# Create a serverless image processing application that automatically resizes and optimizes images

# uploaded to an Amazon S3 bucket.

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. In this blog post, we will guide you through the process of creating AWS Lambda functions that automatically process and resize images uploaded to an S3 bucket.**

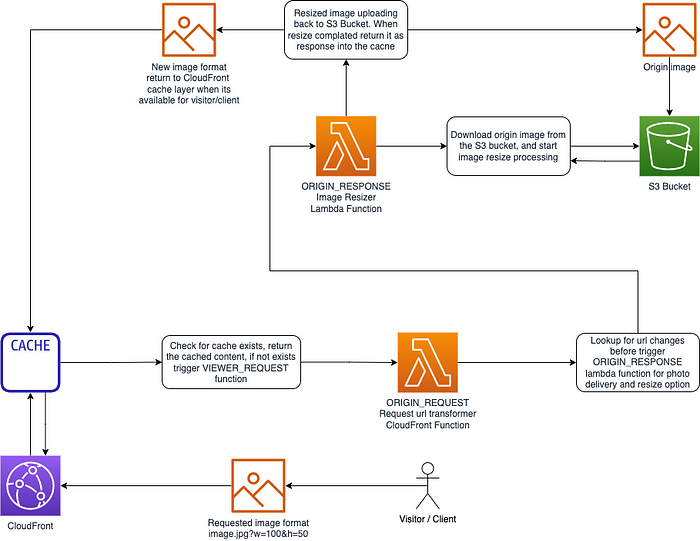
# https://miro.medium.com/v2/resize:fit:700/1*-r0WxwzDaLEn2n6Z8-gUFA.png

### Tech Stack

* **AWS S3** – For storing original and processed images
* **AWS Lambda** – Executes the image processing logic
* **Amazon CloudWatch** – For monitoring and logging
* **AWS IAM** – To manage secure permissions
* **Node.js + Sharp** – Image processing library

### How It Works

1. **User uploads an image** to a specific folder in an S3 bucket (e.g., /uploads/).
2. An **S3 event** triggers a **Lambda function**.
3. The Lambda function:
   * Reads the image from the S3 bucket.
   * Resizes it (e.g., 800x800).
   * Optimizes it (e.g., compressing or converting to webp).
   * Saves the output to another S3 folder (e.g., /processed/).

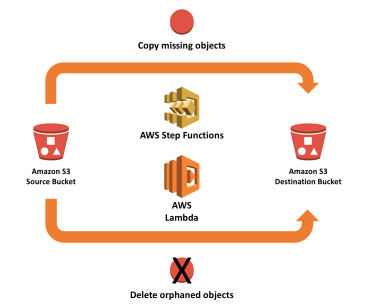


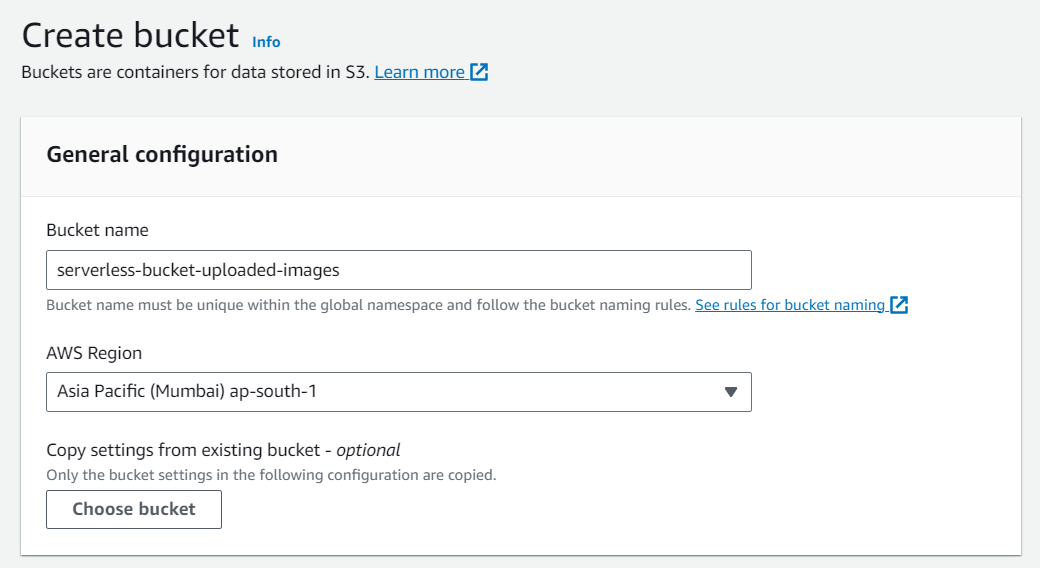
## Step 1 - Creating S3 buckets

We will use two S3 buckets:

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

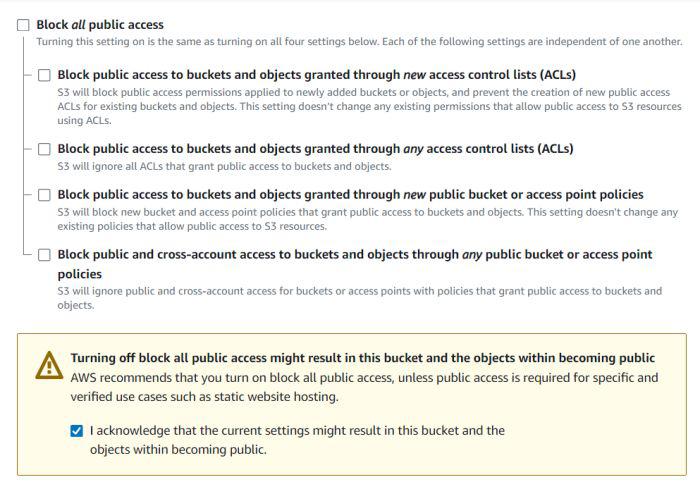
Go to S3 console and click Create bucket. Enter bucket name as 'serverless-bucket-uploaded-images'. Choose any AWS region as 'ap-south-1'.





## 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)**.



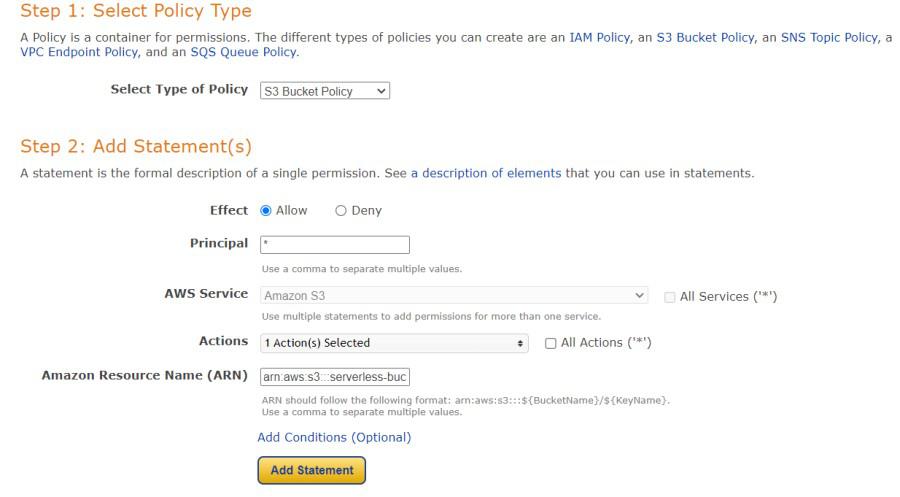
Leave all other settings as default and create bucket. Similarly, create another bucket named 'serverless-bucket-processed-images' with the same region. This bucket will be used to store the processed images. Although we enabled public access while creating the buckets, we still need to attach a bucket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

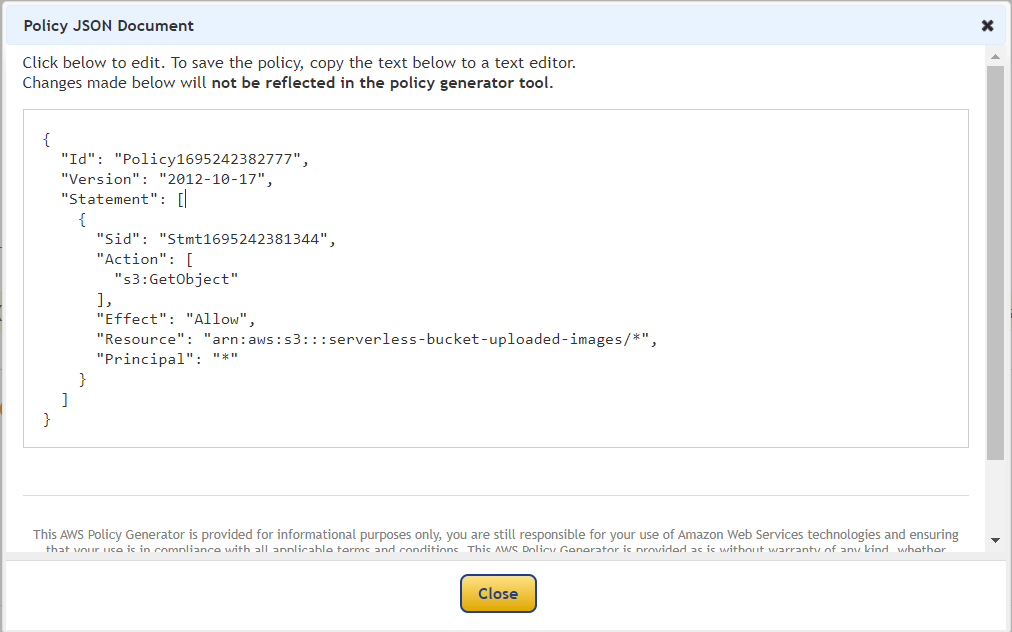
Enter the following settings:

* 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 image.



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:

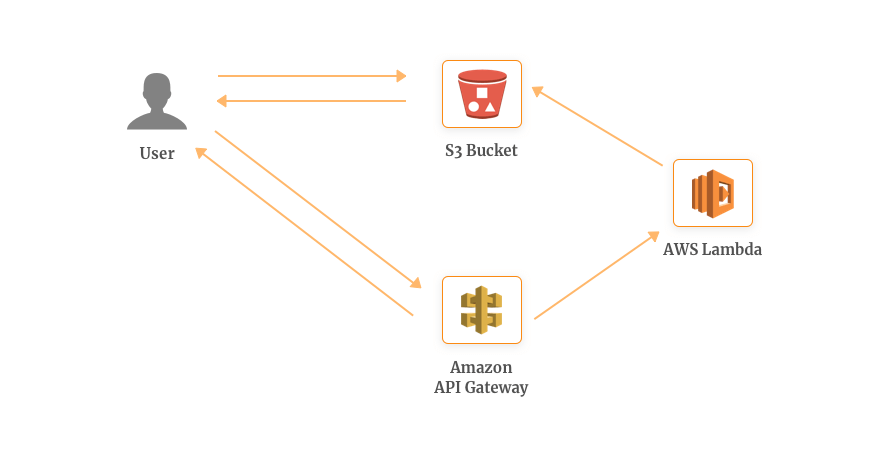
* Type of policy: S3 Bucket Policy
* Effect:Allow
* Principal: \*
* Actions: GetObject, PutObject, and PutObjectAcl
* Amazon Resource Name (ARN): arn:aws:s3:::DESTINATION\_BUCKET\_NAME/\*

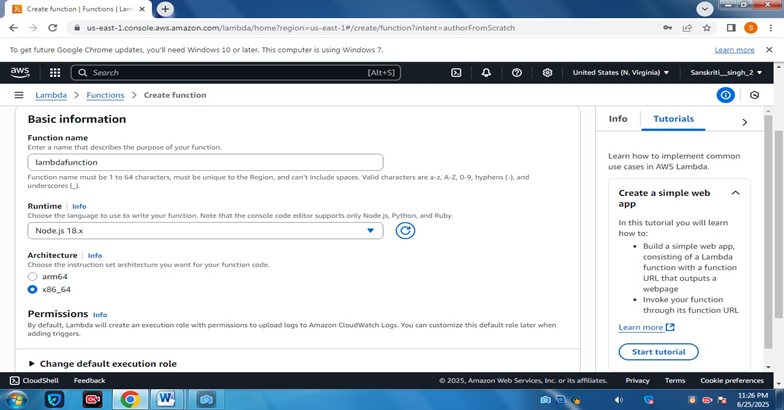
DESTINATION\_BUCKET\_NAME is the name of the bucket used for storing processed images.



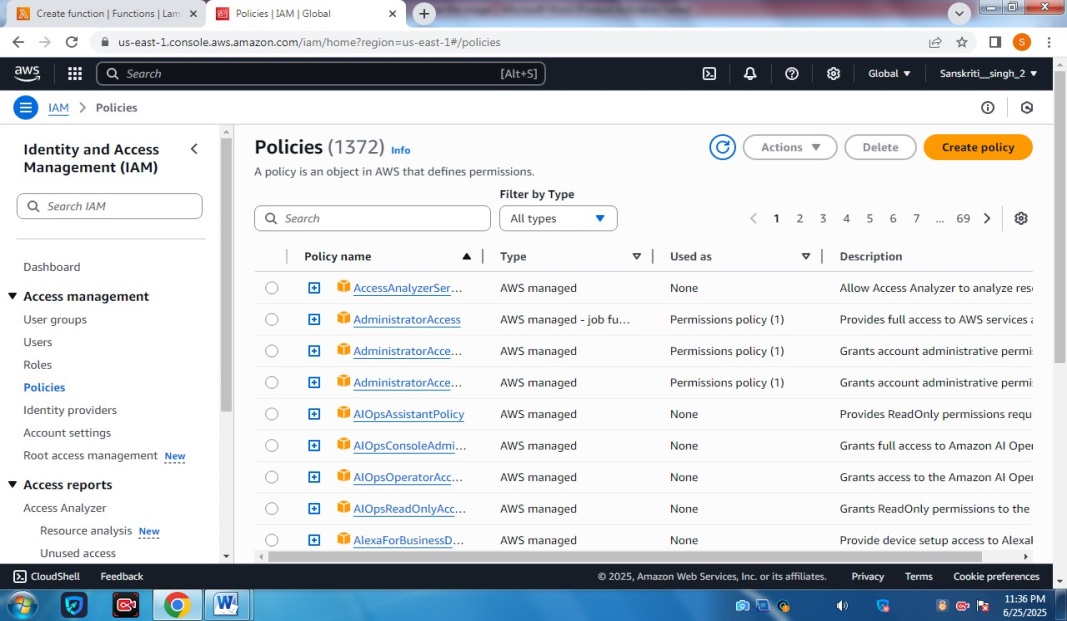
## Step 3 - Creating Lambda function

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.



****

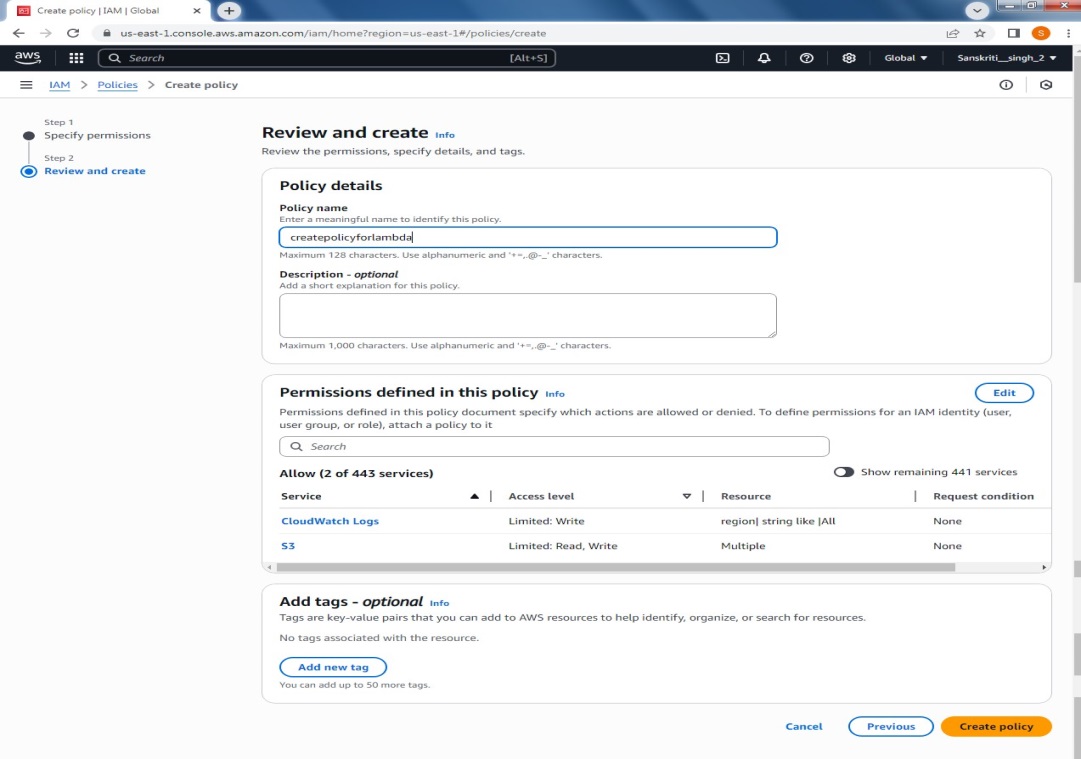
* before going to select a role create an IAM role and policy.

****

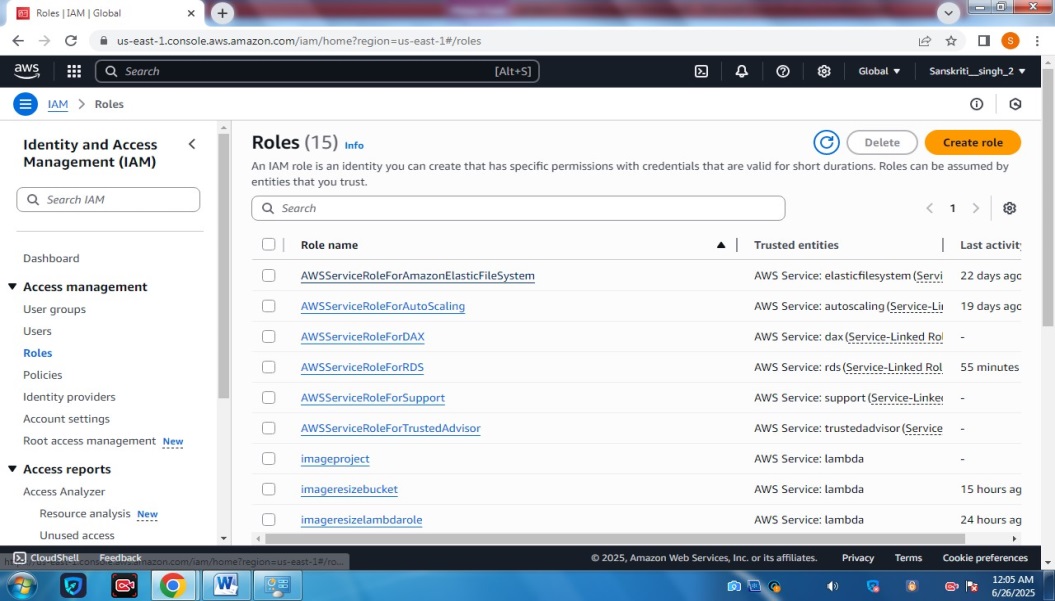
 go to json add policy in JSON format

* select next
* This code in resources adds you are s3 bucket name

- create a policy with a name

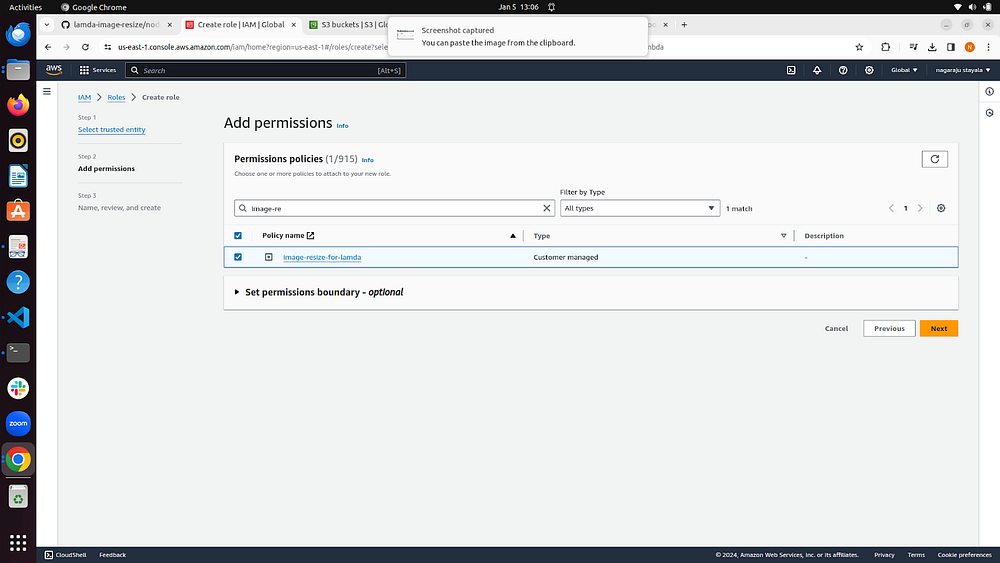


now create a role for this policy

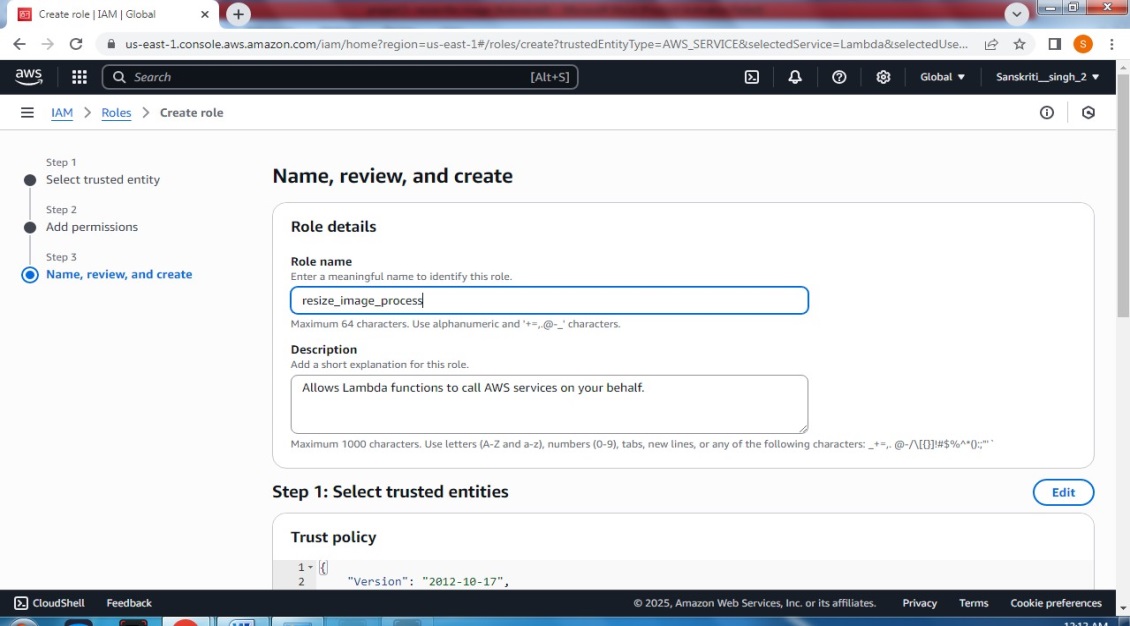


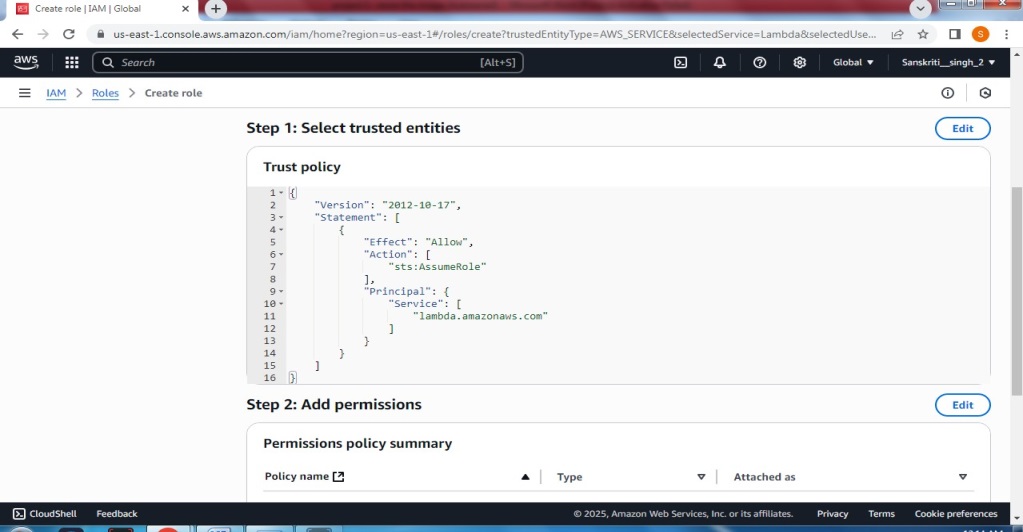
* Select AWS service
* Use case lambda
* Select next



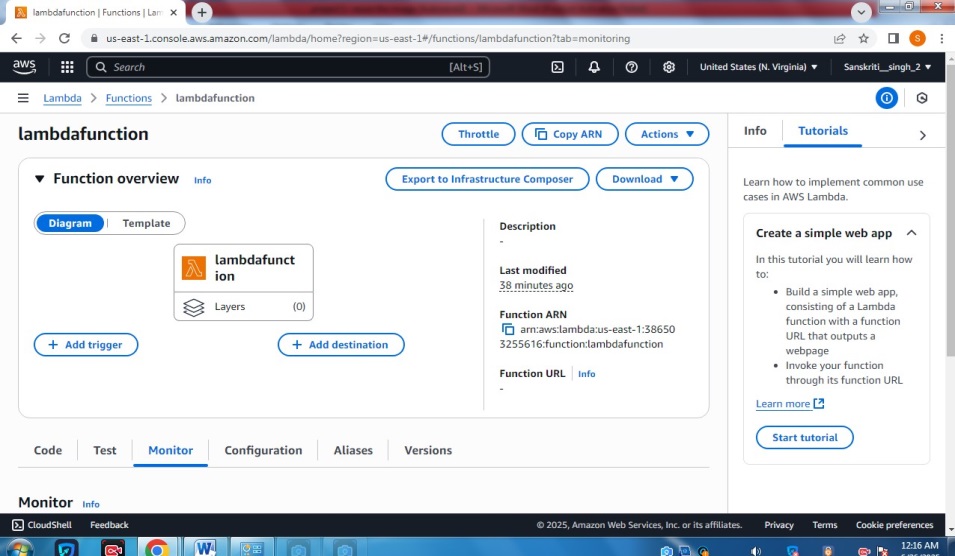


* Select role name

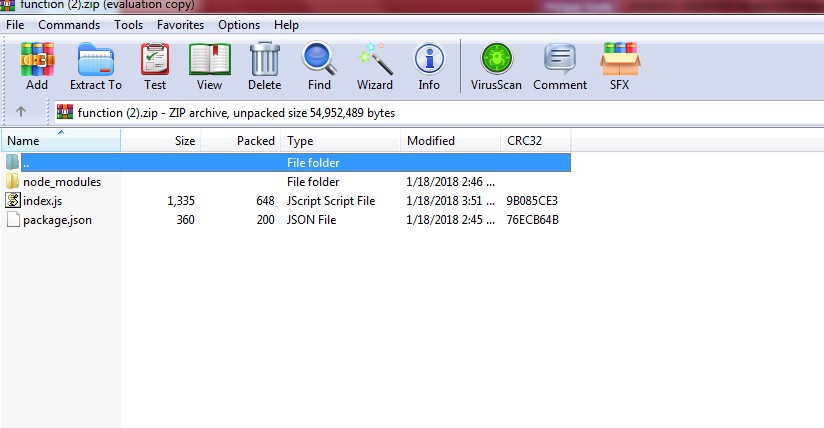




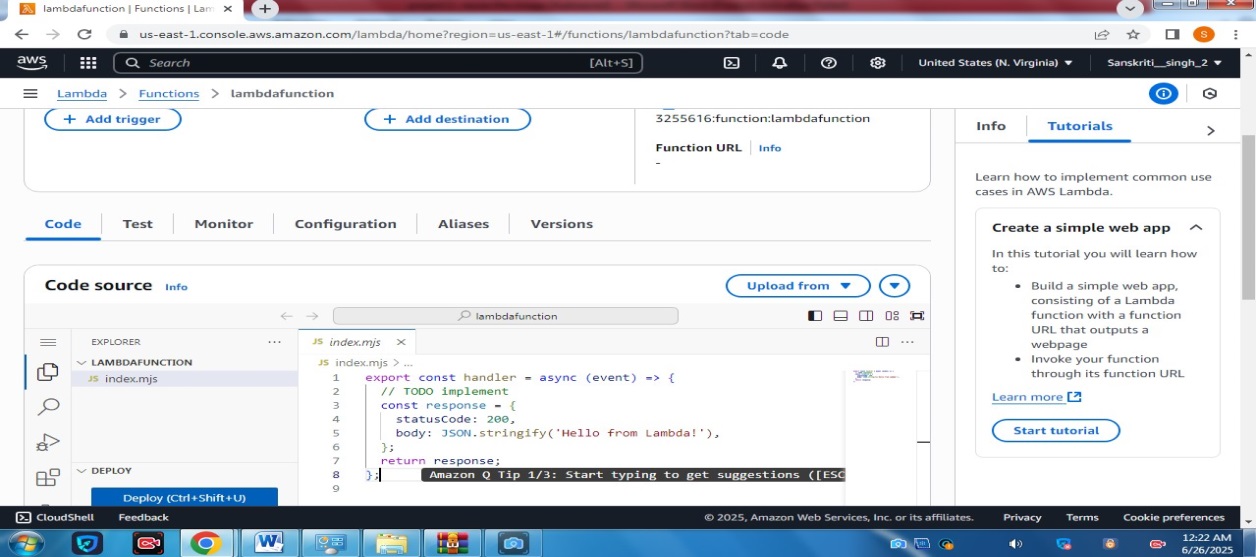
* after selecting Create a lambda function
* Now you can see the function
* Now add triggers
* Select s3 select the bucket the image is uploaded



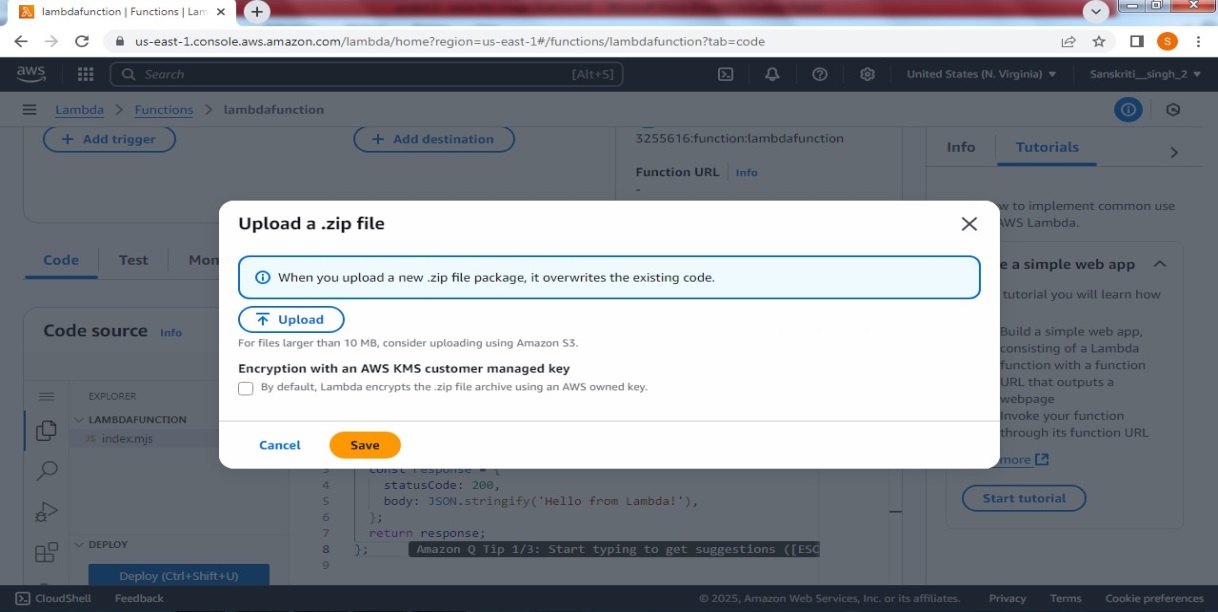
* Using this git command download files locally
* git clone <https://github.com/nagaraju9951/lamda-image-resize.git>
* Npm run packages
* Now all the packages have been compiled into the function.zip file

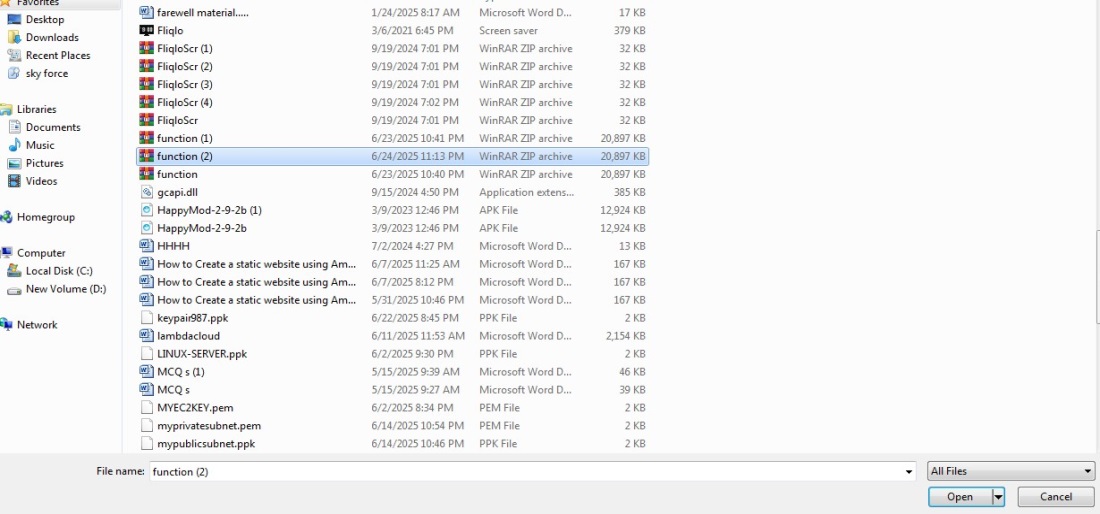


* Now add zip to the lambda function

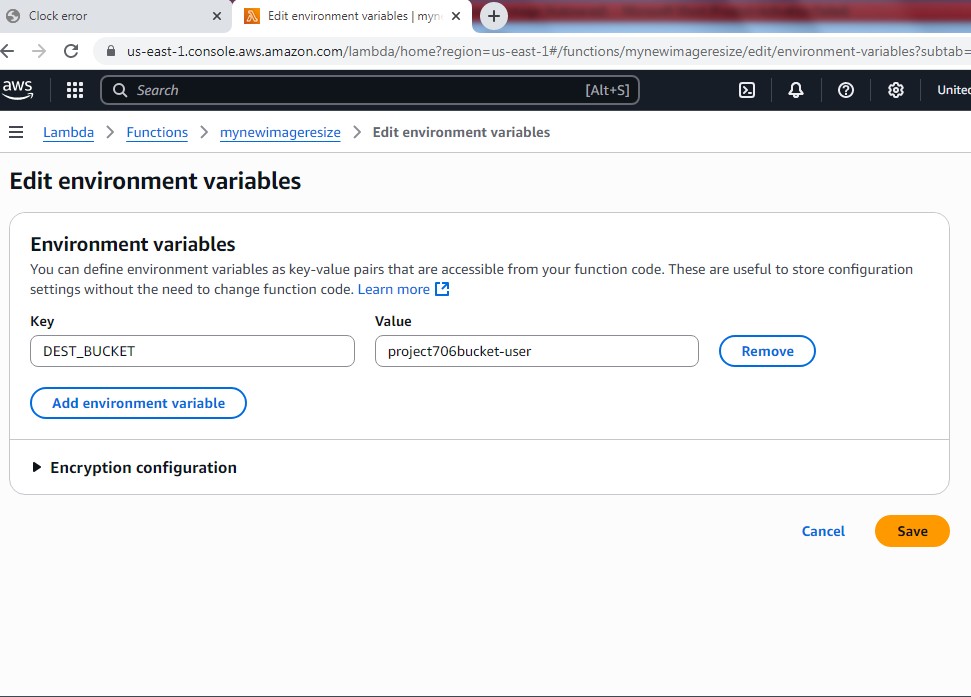


* upload the zip file
* Upload file and save

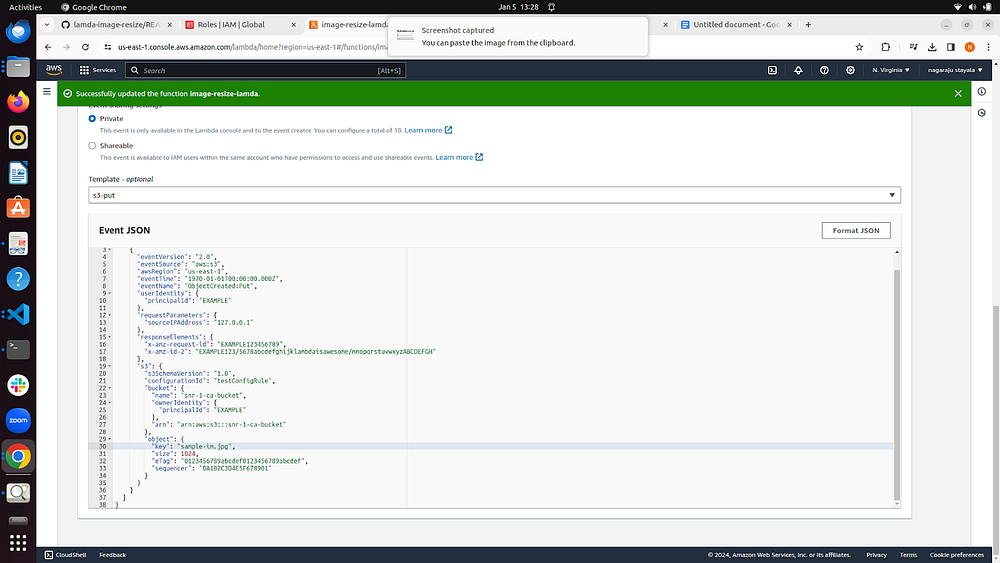




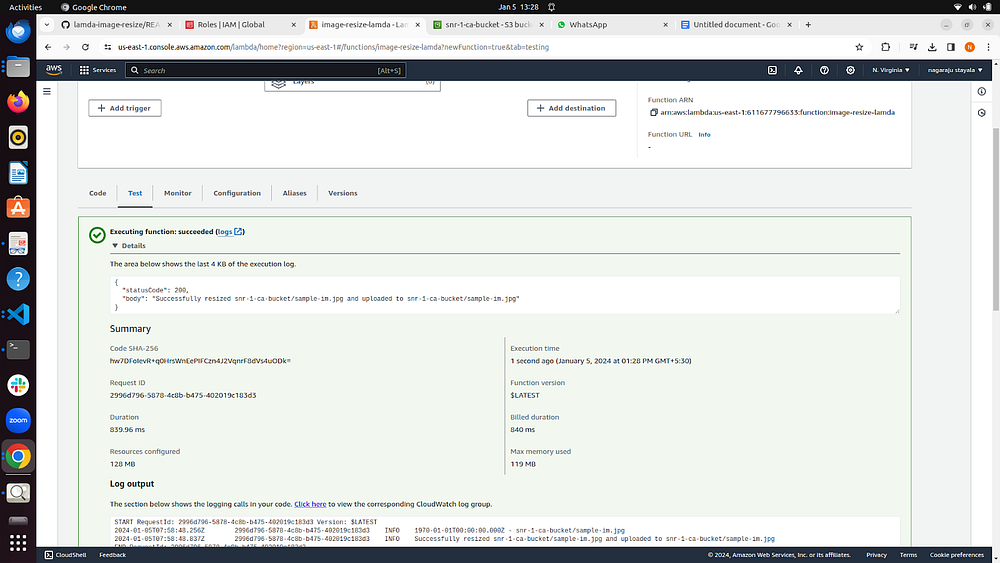
* Now go to Configuration
* Select edit
* now go to environment variables and
* select edit
* Now select the destination bucket

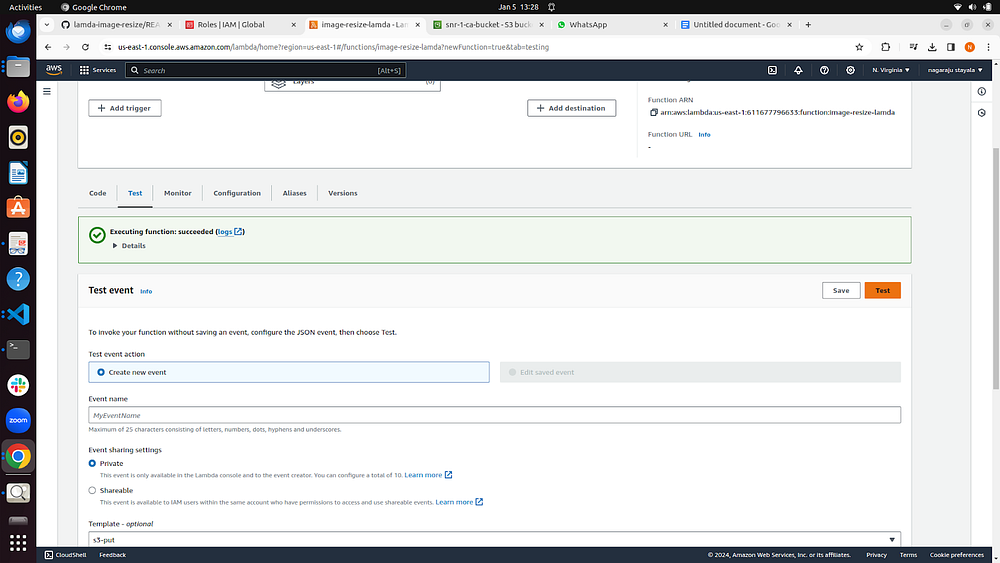


* Save it
* Now go to the test
* Select template
* Select s3 on the search bar
* This code is modified with your existing s3 bucket and the key name is your image name in the s3 bucket

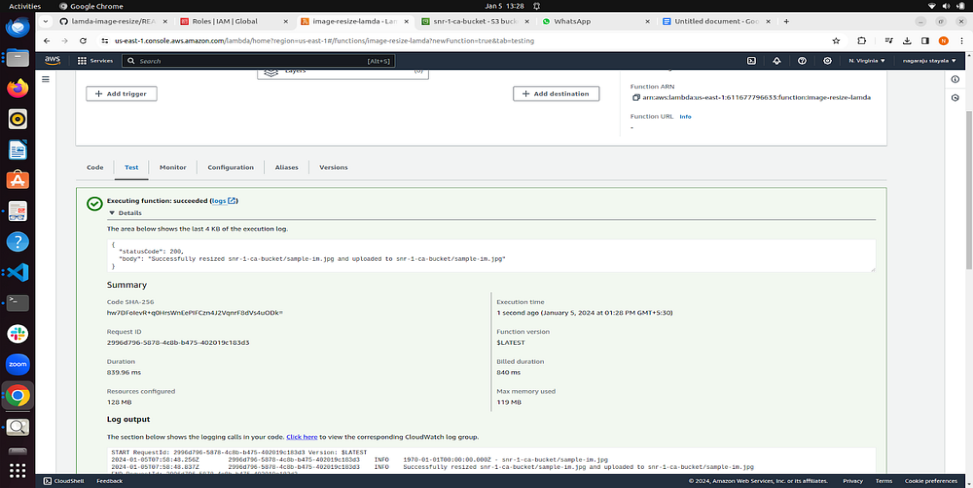


* Now go to test this code

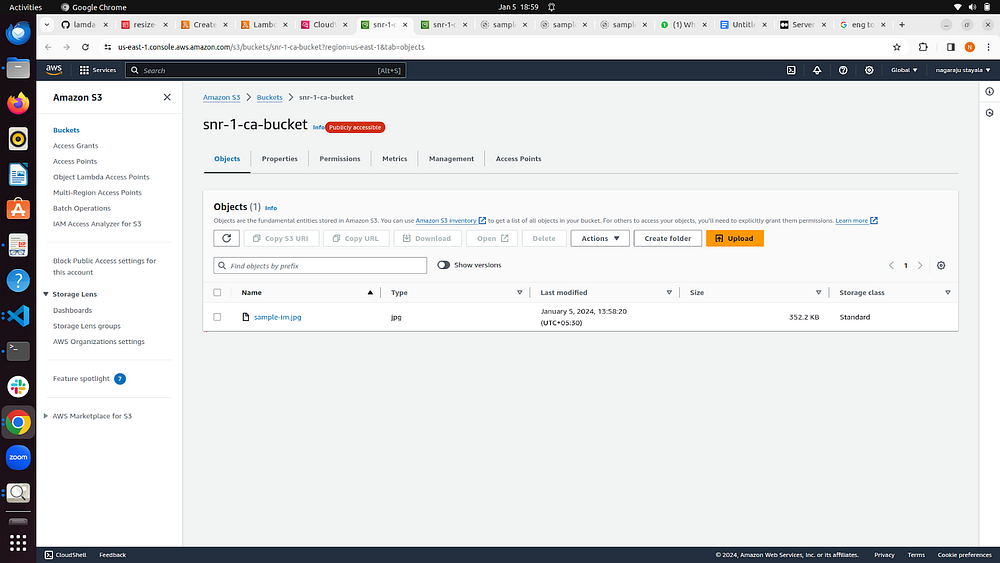


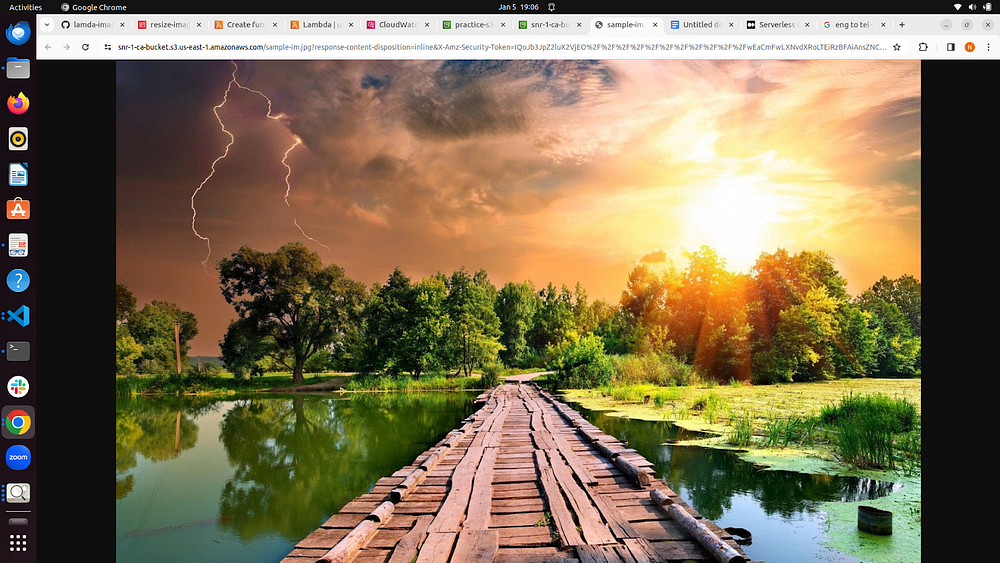


* The code test is successfully run

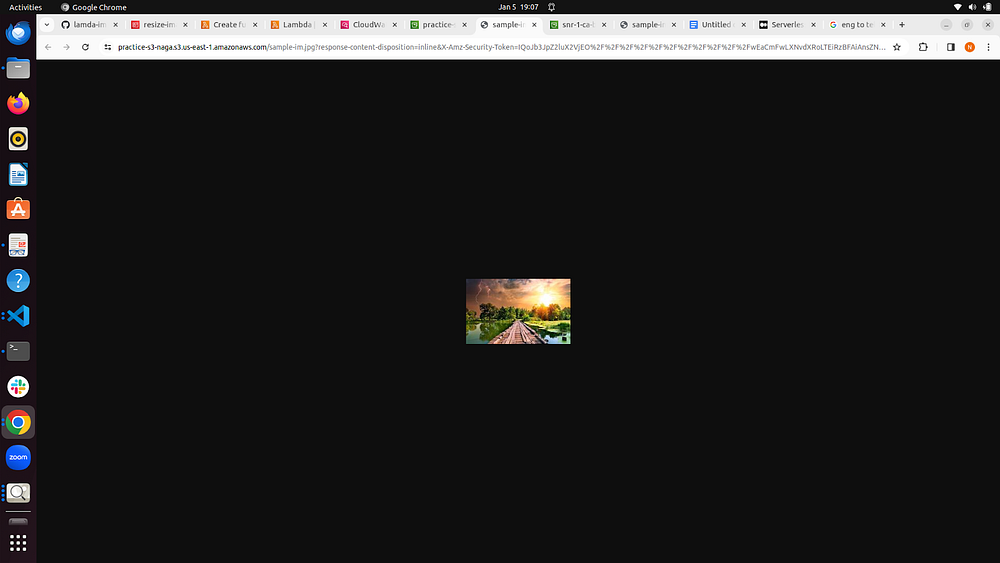


* Now go cloud watch monitoring
* Now we can see s3 buckets with the actual size of the image.





* Now you can see the re-sized image now



## 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.

**Project-2**

**DEPLOY A STATIC WEBSITE ON AMAZON AWS S3**

SIMPLE AND SCALABLE HOSTING SOLUTION

**What is Amazon S3?**

Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance. Millions of customers of all sizes and industries store, manage, analyse, and protect any amount of data for virtually any use case, such as data lakes, cloud-native applications, and mobile apps. With cost-effective storage classes and easy-to-use management features, you can optimize costs, organize and analyse data, and configure fine-tuned access controls to meet specific business and compliance requirements.

### <https://youtu.be/ecv-19sYL3w>

**Amazon S3 (Simple Storage Service)** is a cloud-based **object storage service** offered by Amazon Web Services (AWS). It allows users to store and retrieve any amount of data at any time, from anywhere on the web.

**Key Features of Amazon S3:**

* **Object Storage:** Stores data as objects (not files or blocks), each with a unique key (name).
* **Scalable:** Automatically scales to handle growing data, from a few files to billions.
* **Durable:** Offers **99.999999999% (11 nines)** durability by replicating data across multiple devices and facilities.
* **Secure:** Supports encryption, access control, and integrates with AWS Identity and Access Management (IAM).
* **Access Anywhere:** Data can be accessed via the internet using a unique URL.
* **Static Website Hosting:** Allows hosting of static websites (HTML, CSS, JS) directly from S3 buckets.

**Basic Components:**

* **Bucket:** A container for storing objects (like folders).
* **Objects:** The actual files/data stored in the bucket.
* **Keys:** The unique identifier for each object within a bucket.

**Common Use Cases:**

* Storing backups and archives
* Hosting static websites
* Serving images, videos, and other media
* Big data analytics storage
* Disaster recovery

### Prerequisites

* An AWS account
* A basic static website (e.g., index.html, style.css, etc.)

**Step-by-Step: Deploy a Static Website on Amazon S3**

**1. Sign in to AWS Console**

Go to <https://console.aws.amazon.com> and log in.

**2. Create an S3 Bucket**

1. Navigate to **S3** service.
2. Click **"Create bucket"**.
3. Enter a **unique name** (e.g., my-website-bucket).
4. Choose an **AWS Region**.
5. Scroll to **"Block Public Access settings for this bucket"**:
   * **Uncheck**: "Block all public access".
   * Confirm by checking the warning box.
6. Click **"Create bucket"**.

**3. Configure Bucket for Static Website Hosting**

1. Go to your S3 bucket, then click the “Properties” tab.

2. Scroll to the “Static website hosting” section.

3. Click “Edit”.

4. Select “Enable”.

5. For Index document, enter: index.html.

6. Optionally, enter error.html for Error document.

7. Save the changes.

8**. Note the Endpoint URL — this is your public site URL (e.g., http://my-static-site-2025.s3-website-us-east-1.amazonaws.com).**

**4. Upload Website Files to the S3 Bucket**

1. Go to the “Objects” tab of your bucket.

2. Click “Upload” → “Add files” or “Add folder”.

3. Select your index.html, style.css, images, etc

4. Click “Upload”. 8. Set Permissions (Bucket Policy)

**5.To make your files publicly accessible:**

1. Go to the Permissions tab of your bucket.

2. Scroll to “Bucket policy” and click Edit.

3. Paste the following JSON (replace your-bucket-name):

{

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

"Statement": [

{

"Sid": "PublicReadGetObject",

"Effect": "Allow",

"Principal": "\*",

"Action": "s3:GetObject",

"Resource": "arn:aws:s3:::your-bucket-name/\*"

}

]

}

4. Click Save changes.

**Now, all objects in your bucket are readable by the public.**

**6. Set Index and Error Document**

**(If not done in step 6 or if you need to edit later.)**

1. Navigate to your S3 bucket → Properties → Static website hosting.

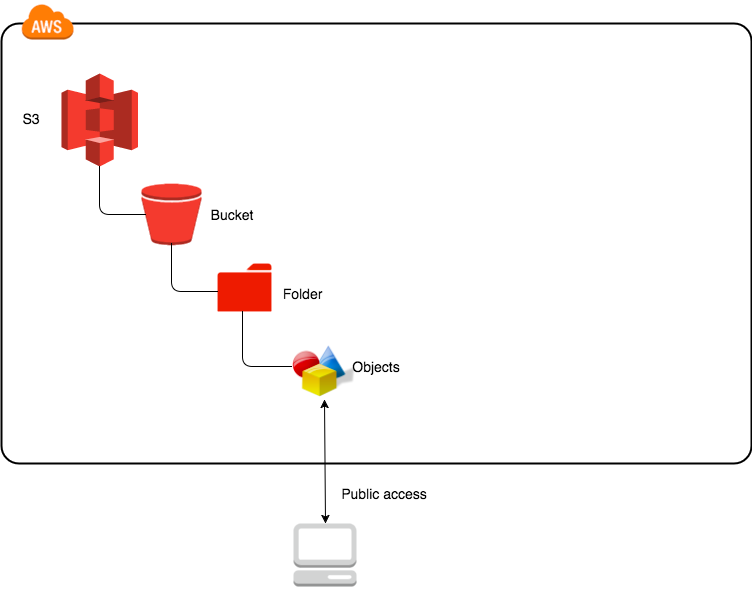
2. Click Edit.

3. Enter: Index document: index .html ,Error document: error.html (optional)

4. Click Save changes.

7. Access Your Website

1. Go back to the **Properties** tab.
2. In the **Static website hosting** section, copy the **Endpoint URL**.
3. Open it in a browser — your website is now live! 🎉



**For testing the website: Accessing the S3 static website to verify deployment.**

**PROJECT-3**

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

**Note:**

* **Grafana is an Open Source tool.**
* **You have to create pictorial representation of your project.**
* **You have to create presentation for your project as guided.**
* **Please make sure that you have created sprint and working accordingly.**

**Table of Contents**

1. Project Overview  
   1.1 Introduction  
   1.2 Objective  
   1.3 Scope
2. Architecture Diagram  
   2.1 AWS Components Used  
   2.2 Monitoring Flowchart
3. Prerequisites  
   3.1 AWS Account  
   3.2 Linux EC2 Instance Setup  
   3.3 Security Group Rules  
   3.4 Key Pair & SSH Access
4. Step-by-Step Implementation  
   4.1 Launch a Linux EC2 Instance  
   4.2 Install Node Exporter on Linux  
   4.3 Configure Prometheus for Data Scraping  
   4.4 Install and Configure Grafana  
   4.5 Add Prometheus as a Data Source in Grafana  
   4.6 Create a CPU Utilization Dashboard in Grafana
5. Testing & Verification  
   5.1 Generating High CPU Load on Linux  
   5.2 Real-Time Monitoring in Grafana
6. Troubleshooting  
   6.1 Common Issues  
   6.2 Logs & Debugging
7. Security Considerations  
   7.1 IAM Roles and Access  
   7.2 Instance and Port Security
8. Conclusion  
   8.1 Summary  
   8.2 Future Improvements
9. Appendix  
   9.1 Commands Used  
   9.2 Useful Links  
   9.3 References
10. Screenshots (Step-wise pictorial guide)

**1. Project Overview**

**1.1 Introduction**

In cloud computing setups, especially on systems such as Amazon Web Services (AWS), pre-emptive monitoring of server performance is important for uptime, scalability, and efficiency of the system. CPU utilization, which reflects directly on application response times and server health, is one of the most important metrics to monitor within production environments.

This project illustrates how Grafana, an open-source visualization platform, can be integrated with a Linux-based EC2 instance on AWS. We can scrape real-time CPU metrics from the EC2 instance by installing Node Exporter, which are then scraped and stored by Prometheus, an open-source monitoring tool. These metrics are finally visualized through a custom Grafana dashboard that provides real-time CPU load and historical trends.

This end-to-end monitoring configuration enables AWS customers to monitor and act on high CPU load situations with industry-standard tools in a scalable and modular fashion.

**1.2 Objective**

The primary goal of this AWS-powered project is to build an end-to-end and working monitoring system that:

* Provisions and sets up a Linux EC2 instance on AWS for monitoring.
* Installs and sets up Node Exporter to gather system-level measurements (namely CPU usage).
* Deploys Prometheus on the same or a different EC2 instance to gather and store time-series data.
* Installs and configures Grafana to visualize real-time CPU metrics with Prometheus as the data source.
* Develops a CPU Utilization Dashboard in Grafana, including graphs and alerts (optional).
* Simulates excessive CPU load on the targeted EC2 instance to confirm that the monitoring stack picks up and displays these changes correctly.

**1.3 Scope**

The work scope of this project entails all the required steps in order to set up an AWS real-time monitoring environment using open-source tools. It covers:

In Scope:

* Launching a Linux EC2 instance in AWS with IAM roles and security group rules in place.
* Installing and setting up Node Exporter on the monitored instance.
* Installing Prometheus on a monitoring instance (can be the same or different EC2 instance).
* Setting up Prometheus to collect metrics from Node Exporter.
* Deployment of Grafana and configuration of it as a data source against Prometheus.
* Creation of a custom Grafana dashboard to display CPU usage.
* Simulation of CPU spikes through Linux tools (such as stress or yes command) for dashboard testing.

Out of Scope:

* High-availability or clustered architecture of Prometheus/Grafana.
* Central logging or alerting systems such as Alertmanager or CloudWatch.
* Monitoring of containers (such as ECS, EKS, or Docker monitoring).
* Advanced IAM security policies or SSO integration.

**2. Architecture Diagram**

2.**1 AWS Components Used**

* Amazon EC2 (Elastic Compute Cloud):
  + Virtual servers (Linux) for:
    - Prometheus (collects metrics)
    - Node Exporter (provides system metrics)
    - Grafana (visual dashboards)
* Amazon VPC (Virtual Private Cloud):
  + Isolated network for secure instance communication.
* Security Groups:
  + Virtual firewalls for EC2 instances.
  + Control access (e.g., allow Prometheus on port 9090, Grafana on port 3000).
* IAM (Identity and Access Management):
  + Manages permissions for AWS services.
  + Assigns roles to EC2 instances (if needed).
* Elastic IP (Optional):
  + Static public IP address.
  + For reliable access to Grafana/Prometheus from your browser.

**2.2** **Monitoring Flowchart**

1. Launch EC2 Linux Instance:
   * Start an EC2 instance to be monitored (e.g., Ubuntu or Amazon Linux).
2. Install Node Exporter:
   * Put Node Exporter on the monitored EC2 instance.
   * It collects system metrics (CPU, memory, disk I/O).
3. Set Up Prometheus on EC2:
   * Install Prometheus on a separate EC2 instance (or the same).
   * Configure it to get metrics from Node Exporter.
4. Install Grafana on EC2:
   * Deploy Grafana on another EC2 instance (or the same).
   * Connect Grafana to Prometheus as a data source.
5. Create Grafana Dashboard:
   * Build a dashboard in Grafana.
   * Display real-time CPU usage and trends using data from Prometheus.

**3. Prerequisites**

**3.1 AWS Account**

Before you begin, make sure you have an active AWS (Amazon Web Services) account. If you haven’t created one yet, you can easily sign up at https://aws.amazon.com.

Accessing the EC2 Dashboard

Once your account is set up:

1. Log in to your AWS Console.
2. In the navigation bar at the top, click on Services.
3. From the drop-down list, find and select EC2 (Elastic Compute Cloud).
4. On the EC2 page, click on Instances.

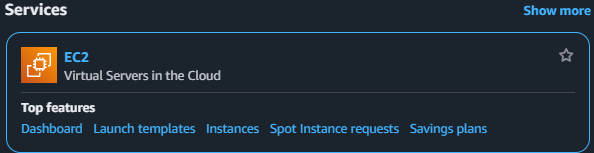
* NOTE: You are operating in a selected region like us-east-1 or ap-south-1.

**3.2 Linux EC2 Instance setup**

To run tools like *Prometheus*, *Node Exporter*, and *Grafana*, you'll need one or more Linux-based EC2 instances. Here's a step-by-step guide to help you get started:

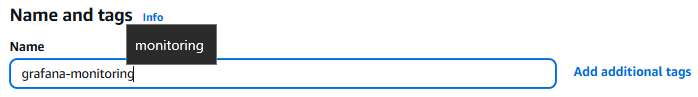
**Step-by-Step Setup Instructions**

1. Navigate to the EC2 Dashboard Open the AWS Console and go to EC2 → Launch Instance.



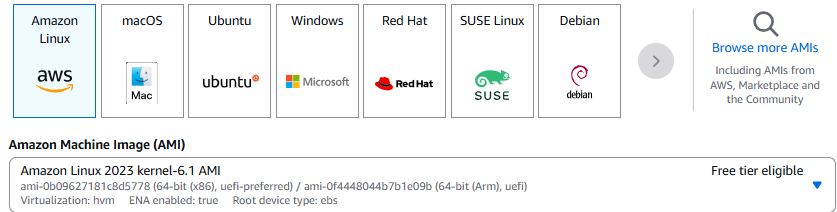


1. Name Your Instance Set your instance name to something like: grafana-monitoring

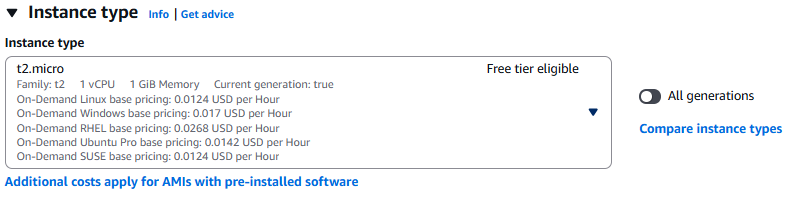


1. Choose an Amazon Machine Image (AMI) Select one of the following:
   * Ubuntu Server 22.04 LTS
   * Amazon Linux 2

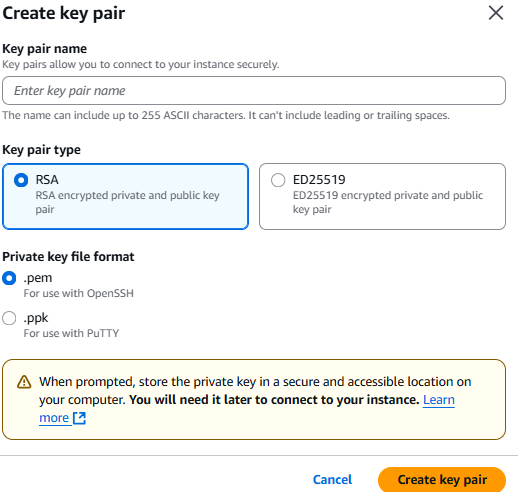
(Free tier eligible)



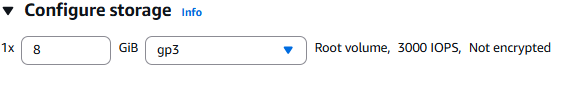
1. Select Instance Type Choose t2.micro – it’s perfect for testing and is *Free Tier eligible*.



1. Create or Choose a Key Pair Select an existing key pair or create a RSA new one.



1. Configure Networking
   * VPC: Use the default option.
   * Subnet: Choose one with Auto-Assign Public IP enabled.
2. Security group
   * Type: ssh
   * Source type: Anywhere
3. Configure Storage
   * Default: 8 GB
   * Optional: Increase to 20 GB if your setup requires more space.



1. Launch the Instance After reviewing your settings, click Launch Instance to spin it up.

Important Tip: Once your instance is running, make sure to note its public IP address — you'll need it to connect via SSH.

**3.3 Configuring Security Group Rules for EC2**

When launching your EC2 instance, it's important to configure a Security Group that allows traffic through the ports required by your monitoring tools. Here's a helpful example of what your inbound rules might look like:

| Type | Protocol | Port | Source | Purpose |
| --- | --- | --- | --- | --- |
| SSH | TCP | 22 | Your IP or 0.0.0.0/0 | Connect to your EC2 via SSH |
| Custom TCP | TCP | 3000 | 0.0.0.0/0 | Access Grafana web dashboard |
| Custom TCP | TCP | 9090 | 0.0.0.0/0 | Access Prometheus interface |
| Custom TCP | TCP | 9100 | 0.0.0.0/0 | View Node Exporter metrics |

\*You can always update your rules anytime: AWS Console → EC2 → Security Groups → Edit Inbound Rules

**3.4 Key Pair SSH & Access**

To securely connect to your Amazon EC2 Linux instance from your local machine, AWS uses a Key Pair authentication method. A key pair consists of:

* Private Key (.pem file) – downloaded and stored securely on your system.
* Public Key – automatically installed in the EC2 instance during launch.

Create a Key Pair in AWS

1. Go to the EC2 Dashboard → Key Pairs  
   AWS Console: Services > EC2 > Network & Security > Key Pairs
2. Click on Create key pair
3. Fill in:
   * Name: grafana-key (or any custom name)
   * Key pair type: RSA
   * Private key file format: .pem
4. Click Create key pair  
   The .pem file (e.g., grafana-key.pem) will be automatically downloaded.

Important: Keep this file secure. You will not be able to download it again.

Connect to EC2 Instance Using SSH:

Once your EC2 instance is running, note its Public IPv4 address from the EC2 Console.

*SSH command (for Ubuntu-based instances):*



*For Amazon Linux 2 AMI:*

**

**4. Step-by-step Implementation**

**This section walks you through the entire setup for monitoring CPU utilization on a Linux-based AWS EC2 instance.**

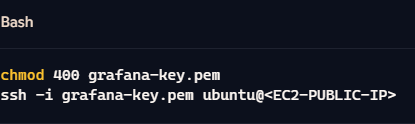
**4.1 Launch a EC2 Instance**

Follow these steps to get your monitoring environment up and running:

* Go to the EC2 Dashboard and click Launch Instance
* Name your instance: grafana-monitoring-node
* Choose an AMI:
  + Ubuntu Server 22.04 LTS (recommended)
  + or Amazon Linux 2
* Select instance type: t2.micro (*Free Tier eligible*)
* Choose or create a Key Pair (.pem file), e.g. grafana-key.pem
* Configure network settings:
  + VPC: Default
  + Auto-assign Public IP: Enabled
* Set up Security Group:
  + Allow ports: 22 (SSH), 9100 (Node Exporter), 9090 (Prometheus), 3000 (Grafana)
* Choose storage: At least 10 GB

· Click Launch Instance

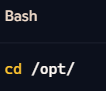
SSH Access Example:

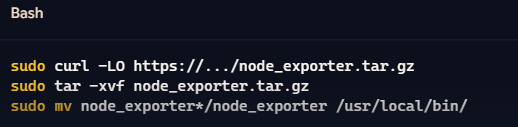


**4.2 Install Node Exporter**

Node Exporter gathers basic system metrics (CPU, memory, etc.).

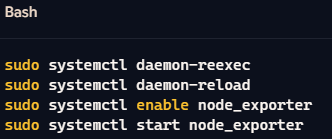
1. Navigate to working directory:



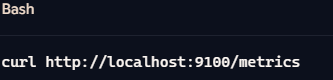
1. Download & install: 
2. Create Node Exporter service:



1. Enable and start the service:



1. Test:

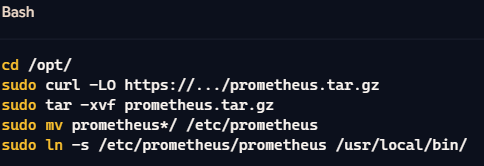


Visit in browser: http://<EC2-PUBLIC-IP>:9100/metrics

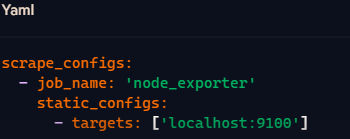
**4.3 Install and Configure Prometheus**

Prometheus will scrape metrics from Node Exporter.

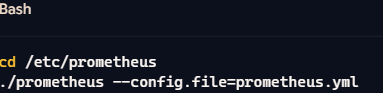
* Download and install:



* Edit Prometheus config:



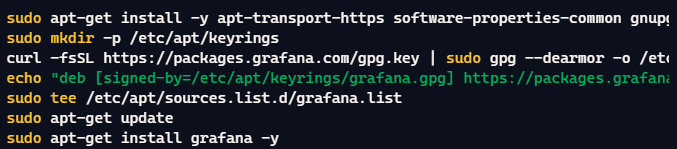
* Start Prometheus:



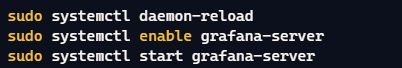
**4.4 Install Grafana**

Grafana helps visualize the data pulled by Prometheus.

* Add Grafana repo and install:



* Start Grafana:



Access Grafana UI: http://<EC2-PUBLIC-IP>:3000 Login: admin / admin *(change password after login)*

4.5 Add Prometheus in Grafana

In Grafana:

1. Go to Settings → Data Sources → Add data source
2. Choose Prometheus
3. Set URL:
   * http://localhost:9090 (or public IP if remote)
4. Click Save & Test
5. You’ll see: *Data source is working*

4.6 Create a CPU Utilization Dashboard

1. Go to Create → Dashboard → Add new panel
2. Enter the following PromQL query:



1. Set panel title: CPU Utilization
2. Choose visualization: Line graph
3. Click Apply

You now have real-time CPU monitoring on your EC2 instance!

Simulate CPU Load

To test your dashboard, simulate high CPU usage:



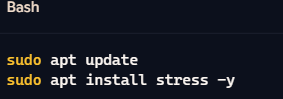
**5. Testing & Verifying Your Monitoring Setup**

**Now that you’ve set up Prometheus, Node Exporter, and Grafana, it’s time to test everything by generating some CPU load and watching it show up in** real-time on your **dashboard.**

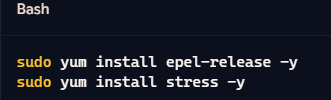
**5.1 Simulate High CPU Load**

To check if your metrics are being collected correctly, let’s simulate a CPU spike using a tool called stress.

For Ubuntu:



For Amazon Linux 2:



Run the stress test:



5.**2 Watch the Metrics in Grafana**

Once stress is running:

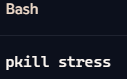
1. Open Grafana in your browser: http://<EC2-IP>:3000
2. Navigate to your CPU Utilization Dashboard
3. If you're using this query



1. Hover over the graph to see real-time CPU percentage value

Stopping the Stress Test

* Let it complete naturally after 60 seconds, or
* Stop it manually:



**6. Troubleshooting**

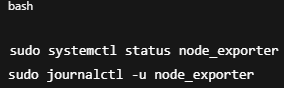
This section covers common issues that may occur during setup or testing, along with steps to view logs and debug services such as Node Exporter, Prometheus, and Grafana.

**6.1 Common issue**

| Issue | Cause | Solution |
| --- | --- | --- |
| Permission denied (publickey) when connecting via SSH | Wrong username, missing key, or incorrect permissions | Ensure you use ubuntu (for Ubuntu AMI) or ec2-user (for Amazon Linux), and run chmod 400 key.pem |
| Grafana not accessible on port 3000 | Port 3000 not allowed in security group | Add inbound rule in EC2 security group to allow TCP 3000 from your IP |
| Prometheus shows “Target down” | Prometheus can’t connect to Node Exporter | Check if Node Exporter is running and verify its port (9100) is open |
| Grafana data source test fails | Wrong Prometheus URL or server not running | Use http://localhost:9090 if Grafana  and Prometheus are on same instance |
| Graph not updating in Grafana | Incorrect PromQL query or Node Exporter misconfigured | Recheck query and confirm data is being scraped by Prometheus |

**6.2 Logs & Debugging**

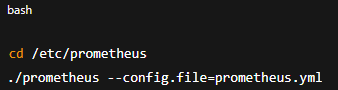
Use logs to identify and fix errors in services:



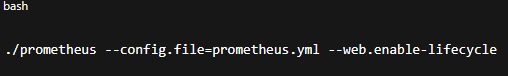
* Look for “Active: running”
* Check for permission or port binding errors

*Check Prometheus Logs*

If started manually:

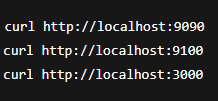


* Errors in YAML format or targets will show here
* Also validate your config file using:



Additional Debugging Tips

* Use curl to test local ports:



**7. Security Considerations**

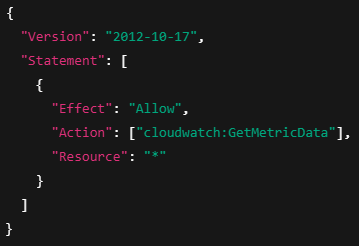
Security is essential when deploying monitoring tools like Prometheus and Grafana in a cloud environment. This section covers IAM permissions, instance-level security, and network port access.

**7.1 IAM Roles and Access**

AWS Identity and Access Management (IAM) ensures that only authorized users or services can access your resources.

*Best Practices:*

* Create IAM roles for EC2 with least privilege:
  + If your instance accesses other AWS services (e.g., S3, CloudWatch), attach an IAM Role with required permissions only.
  + Example policy for read-only access to CloudWatch (optional):



·Use IAM users/groups for accessing AWS Console instead of root account.

·Avoid hardcoding AWS credentials in scripts or EC2 environments.

*Implementation Tip:*

Attach an IAM Role when launching EC2:

EC2 Console → Actions → Security → Modify IAM Role

**7.2 Instance and Port Security**

Proper configuration of network and instance security settings is critical to avoid unauthorized access.

| Port | Service | Access Scope | Notes |
| --- | --- | --- | --- |
| 22 | SSH | Your IP only (recommended) | Use My IP to avoid global SSH access |
| 9100 | Node Exporter | Your IP or internal subnet | Avoid exposing to public internet |
| 9090 | Prometheus | Your IP only | Optional: restrict to monitoring host |
| 3000 | Grafana | Your IP or specific subnet | Restrict access to authorized users |
|  |  |  |  |

**8. Conclusion**

This section provides a summary of the work completed and outlines possible improvements and enhancements for the future.

**8.1 Summary**

In this project, we successfully integrated Grafana with a Linux EC2 server hosted on Amazon Web Services (AWS) to monitor high CPU utilization in real time. The following key components were implemented:

* EC2 instance setup with Ubuntu Linux as the monitoring node.
* Node Exporter installation to collect system metrics (CPU, memory, disk).
* Prometheus configuration to scrape metrics from Node Exporter.
* Grafana setup for visualizing live data via dynamic dashboards.
* A custom CPU utilization graph was created to display performance spikes during load testing.
* Security best practices were applied, including proper IAM usage and firewall rules through EC2 security groups.

The system proved to be robust, scalable, and responsive to real-time monitoring needs for high CPU usage.

**8.2 Future Improvements**

While the current setup fulfills core monitoring needs, several enhancements can be made:

*Feature Improvements:*

* Email/SMS alerts in Grafana for CPU thresholds using Alertmanager or SNS integration.
* Dashboard templating for multi-instance support (monitor many EC2s).
* Persistent storage for Prometheus metrics using EBS volumes or EFS.

*Security Enhancements:*

* Reverse proxy setup using NGINX with SSL for Grafana.
* SSO (Single Sign-On) integration for secure dashboard access.
* Audit logs and log shipping to centralized services like Amazon CloudWatch or ELK stack.

*Scalability:*

* Use Prometheus Federation or Thanos for large-scale metrics collection.
* Deploy using Docker or Kubernetes for containerized environments.

This concludes the project. If implemented in a production environment, these enhancements will make the system more secure, resilient, and enterprise-ready.