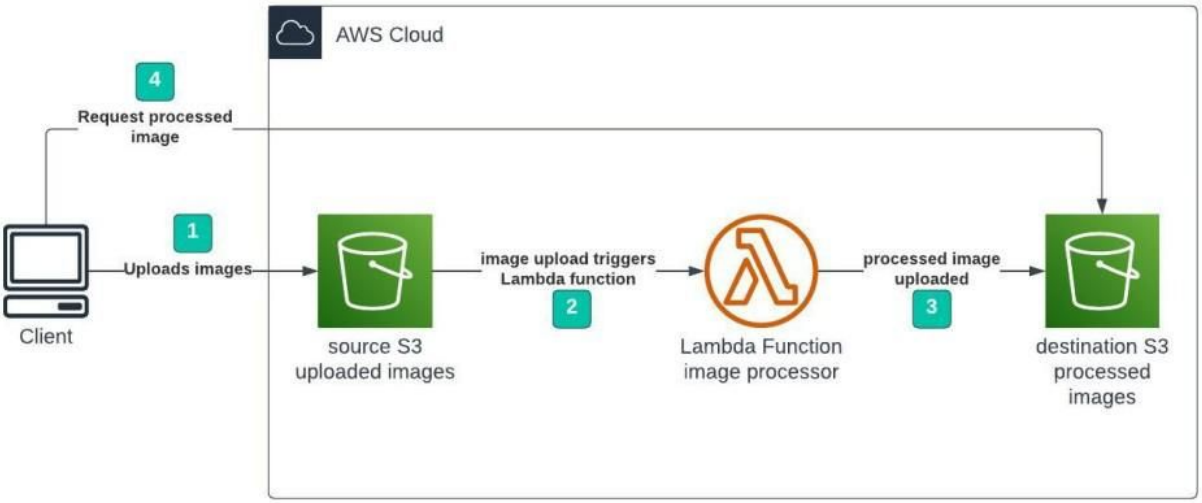
Project 1

Serverless Image Processing

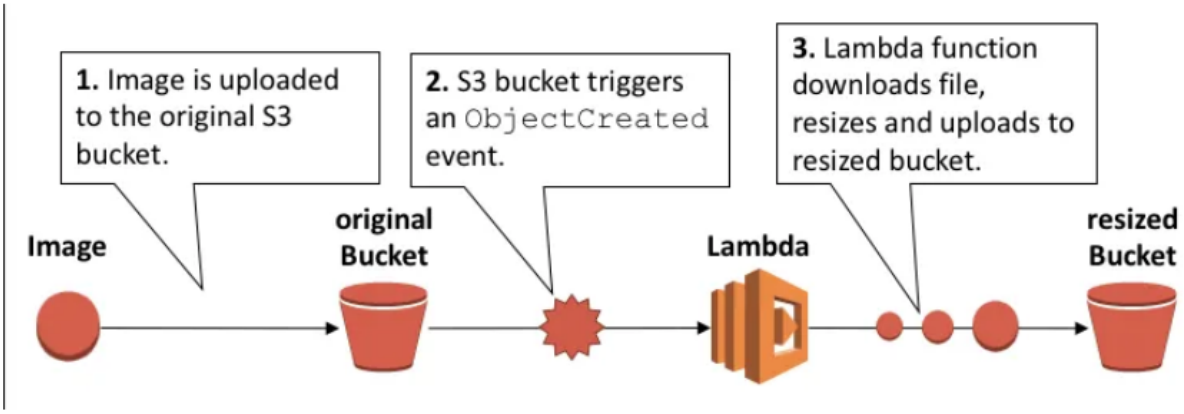


Serverless Image Processor

1 TableofContents

1 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.



Getting Started with AWS Lambda

In this tutorial, you use the console 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.

Topic

- Create a serverless image processing application that automatically resizes and optimizes images uploaded to an Amazon S3 bucket.

To complete this tutorial, you carry out the following steps:

1. Create source and destination Amazon S3 buckets and upload a sample image.
2. Create a Lambda function that resizes an image and outputs a thumbnail to an Amazon S3 bucket.
3. Configure a Lambda trigger that invokes your function when objects are uploaded to your source bucket.
4. Test your function, first with a dummy event, and then by uploading an image to your source bucket.

2 Process

Here's the process:

1. A user requests a resized asset from an S3 bucket through its static website hosting endpoint. The bucket has a routing rule configured to redirect to the resize API any request for an object that cannot be found.
2. Because the resized asset does not exist in the bucket, the request is temporarily redirected to the resize API method.
3. The user's browser follows the redirect and requests the resize operation via API Gateway.
4. The API Gateway method is configured to trigger a Lambda function to serve the request.
5. The Lambda function downloads the original image from the S3 bucket, resizes it, and uploads the resized image back into the bucket as the originally requested key.
6. When the Lambda function completes, API Gateway permanently redirects the user to the file stored in S3.
7. The user's browser requests the now-available resized image from the S3 bucket. Subsequent requests from this and other users will be served directly from S3 and bypass the resize operation. If the resized image is deleted in the future, the above process repeats and the resized image is re-created and replaced into the S3 bucket.

• Objective

Configure, control, and deploy secure applications:

- Environment variables modify application behavior without new code deployments.
- Versions safely test new features while maintaining stable production environments.
- Lambda layers optimize code reuse and maintenance by sharing common components across multiple functions.
- Code signing enforce security compliance by ensuring only approved code reaches production systems.

Scale and perform reliably:

- Concurrency and scaling controls precisely manage application responsiveness and resource utilization during traffic spikes.
- Lambda SnapStart significantly reduce cold start times. Lambda SnapStart can provide as low as sub-second startup performance, typically with no changes to your function code.
- Response streaming optimize function performance by delivering large payloads incrementally for real-time processing.
- Container images package functions with complex dependencies using container workflows.

••Features

Features of AWS Lambda Functions

The following are the some features which are provided by the AWS (Amazon Web Services):

No. Instead, AWS Lambda is an essential component of serverless technology for AWS customers. Here's how:

You need several components to build an application on serverless architecture. One typical architecture is the LAMP stack, which requires:

- Linux operating system
- Apache HTTP server
- MySQL relational database management
- PHP programming language

Another, more specific stack for a web application would look like this:

- Database service
- HTTP gateway service
- Computing service

For people already using AWS, Lambda is the computing service in that stack.

But that's not why serverless computing can be cost-efficient for running your operations in the cloud. There are two main benefits built into the serverless architecture:

- **Pay-per-request pricing model** – You only pay for requests Lambda has served and the time it took AWS to serve those requests. Before your compute request, you do not pay a cent like most other architectures that cost money by running continuously without actually doing anything for you.
- **Autoscaling** – This means the service scales up and down on demand. It increases compute resources such as CPU, RAM, and memory when demand increases. But it shuts some of those down when there is little activity inside the servers. It can automatically scale from zero to practically infinity and back to zero again when your system is idle.

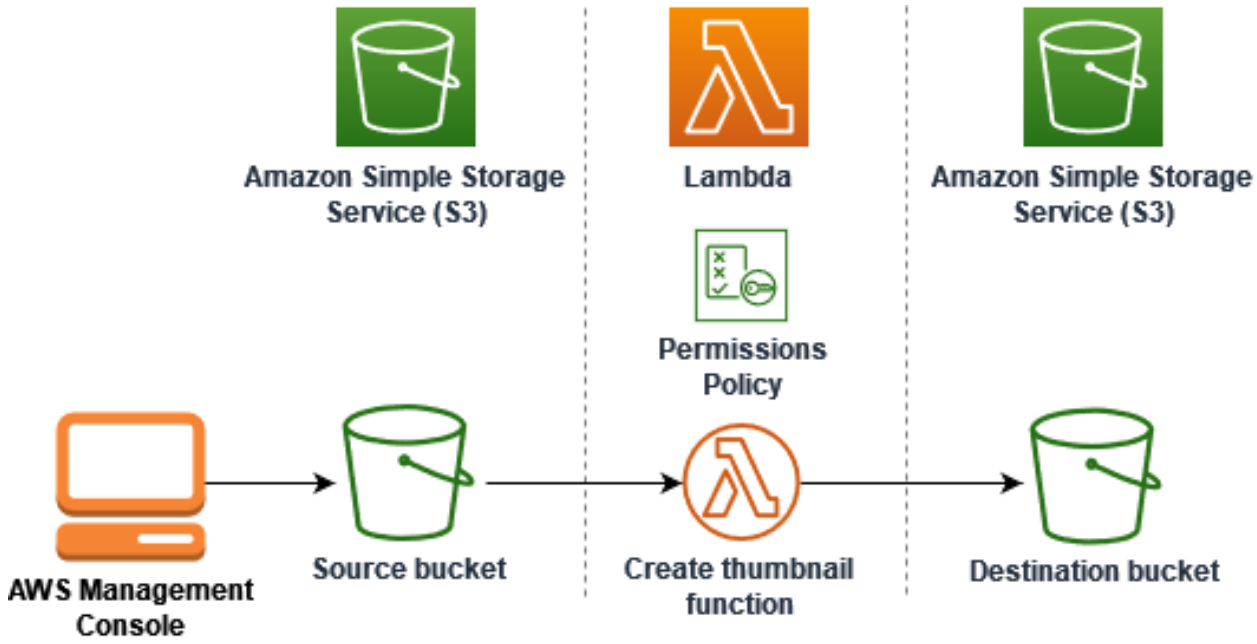
3 Implementation

Getting Started with AWS Lambda

In this tutorial, you use the console 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.

• Serverless Image Processing Flow

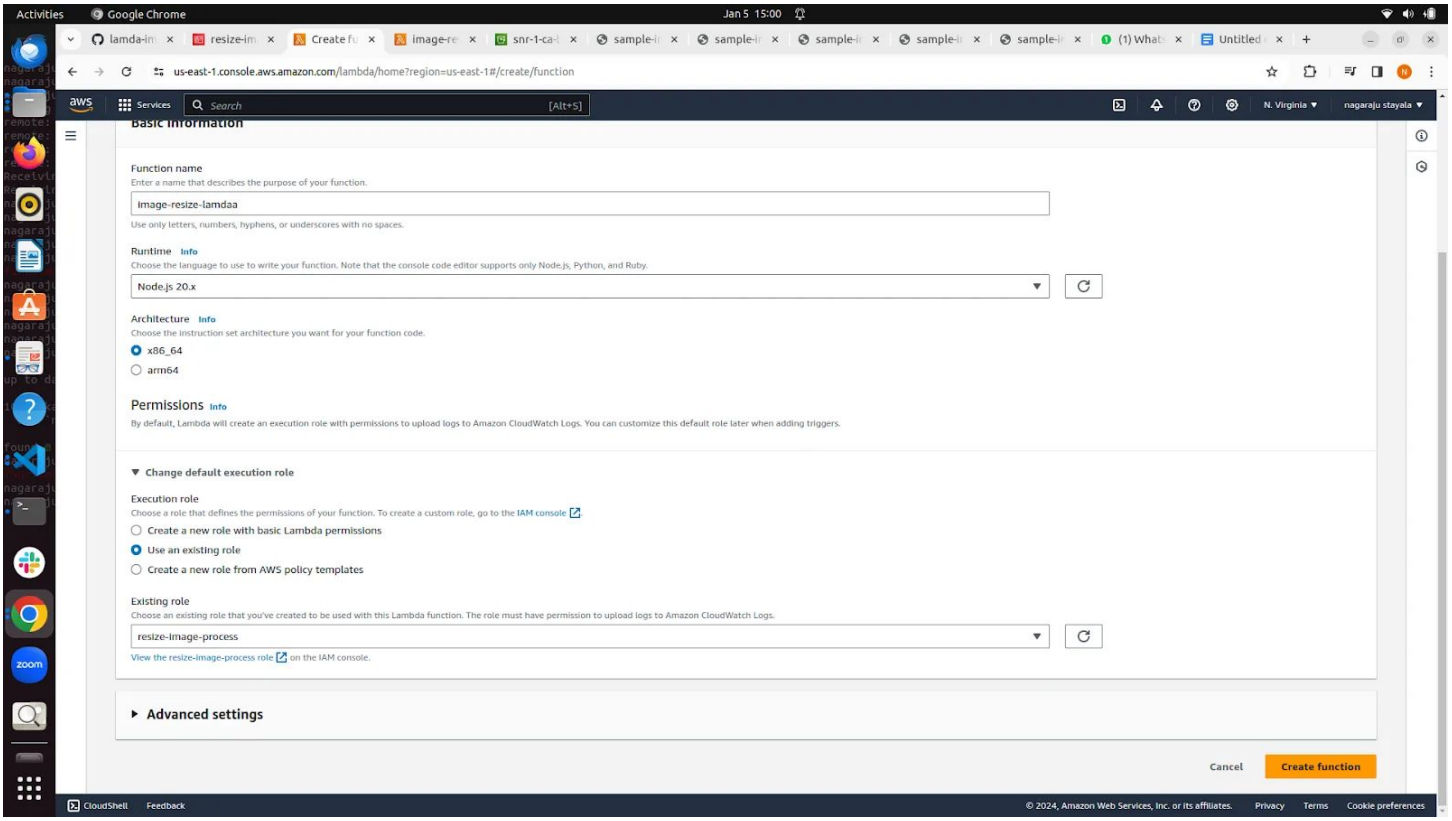
1. User uploads a file to the source S3 bucket (which is used for storing uploaded images)
2. When the image is uploaded to a source S3 bucket, it triggers an event which invokes the Lambda function. The lambda function processes the image.
3. Processed image is stored in the destination S3 bucket.
4. The processed image is requested by the user.



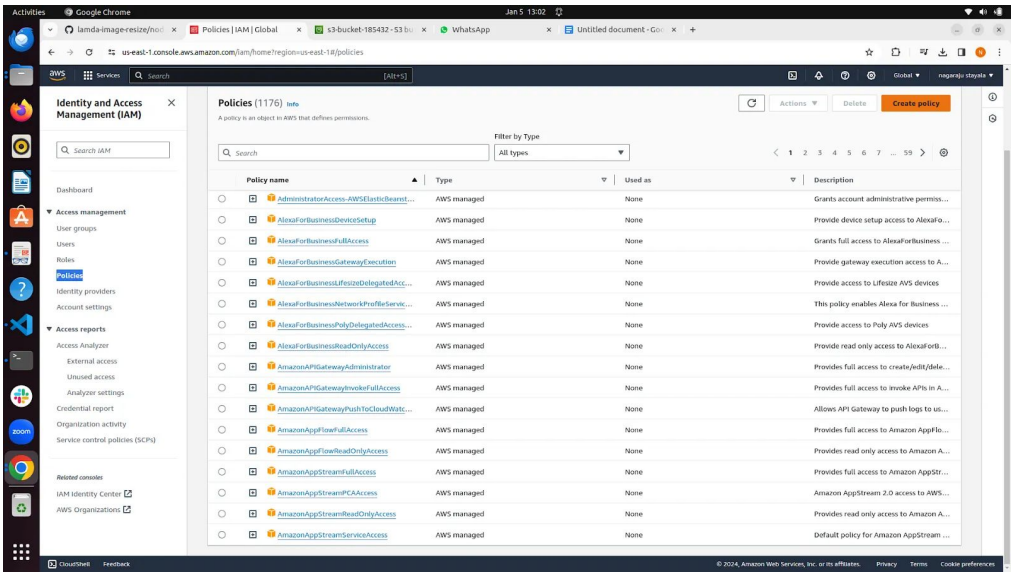
- create a lambda function

- run time select node.js20x

- select an existing role



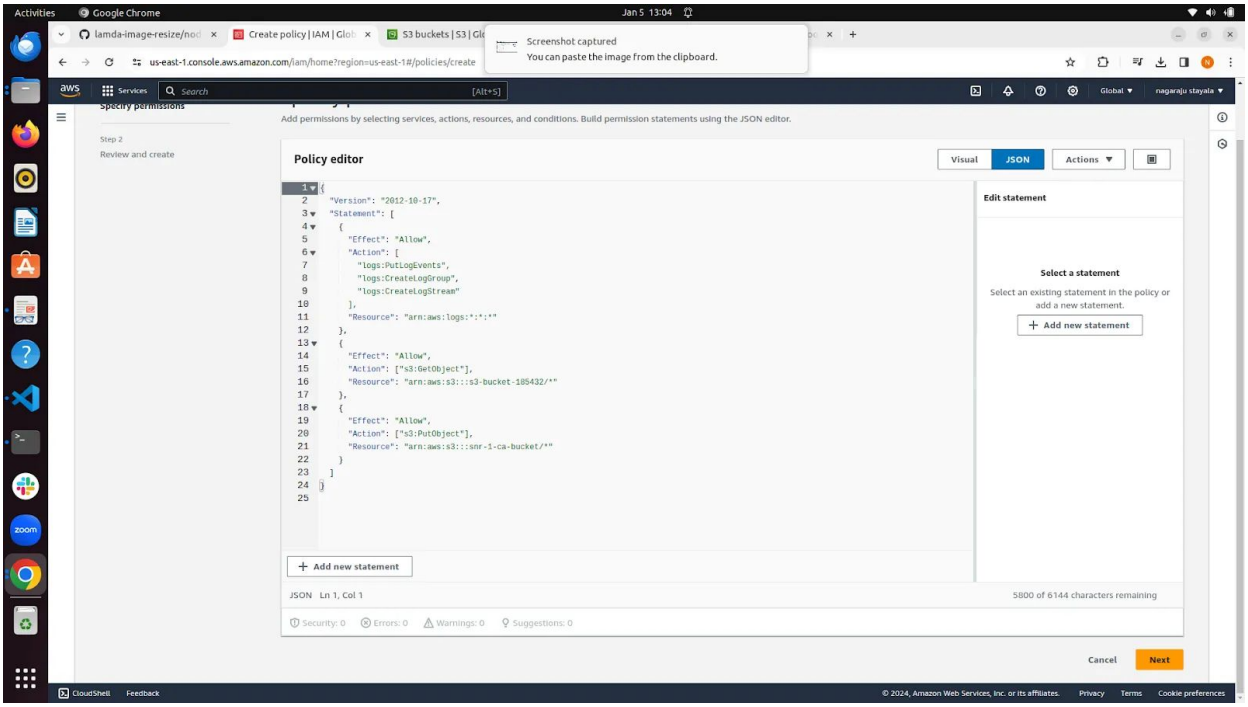
- before going to select a role create an IAM role and policy.



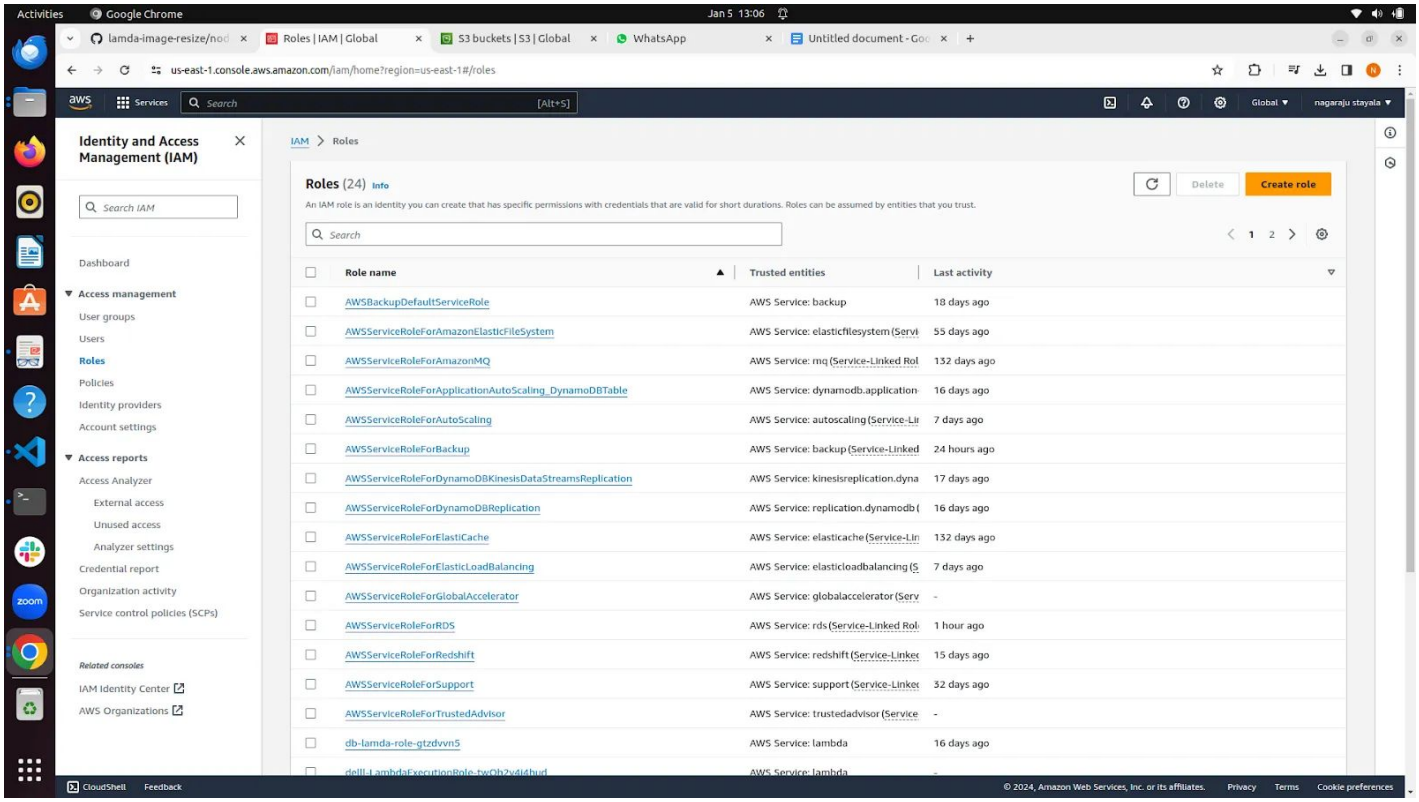
- go to json add policy in JSON format

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

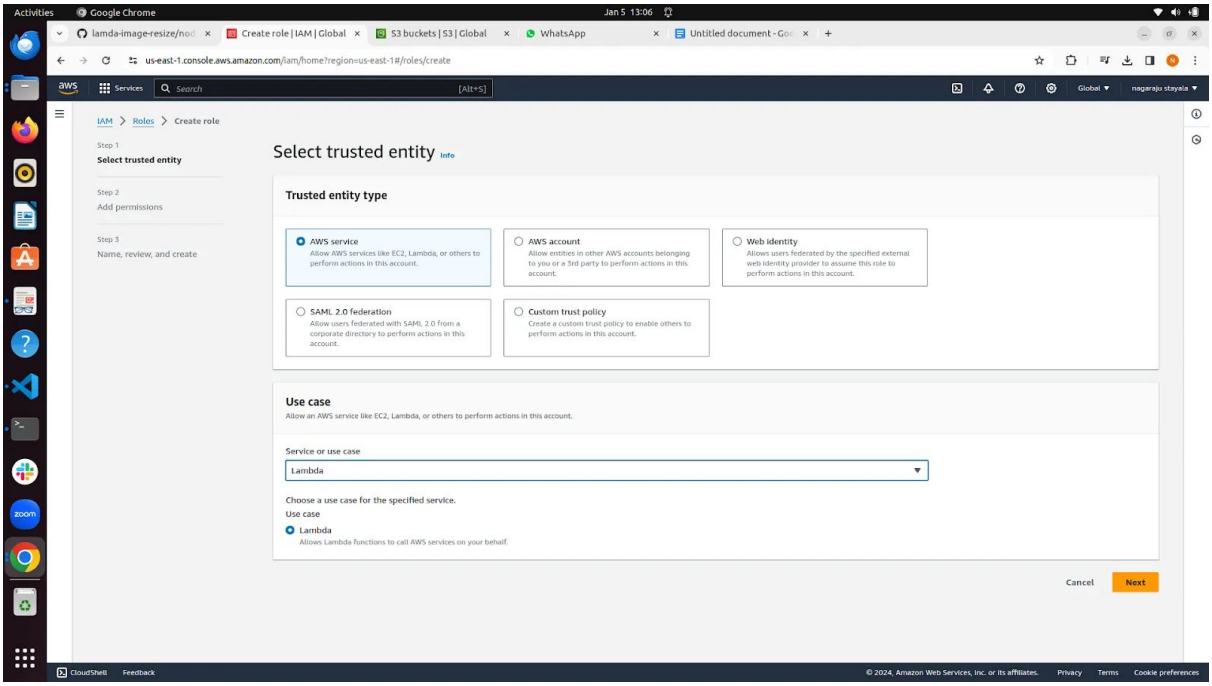
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:PutLogEvents",
        "logs:CreateLogGroup",
        "logs:CreateLogStream"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::BUCKET_NAME/*"
    },
    {
      "Effect": "Allow",
      "Action": ["s3:PutObject"],
      "Resource": "arn:aws:s3:::DEST_BUCKET/*"
    }
  ]
}
```

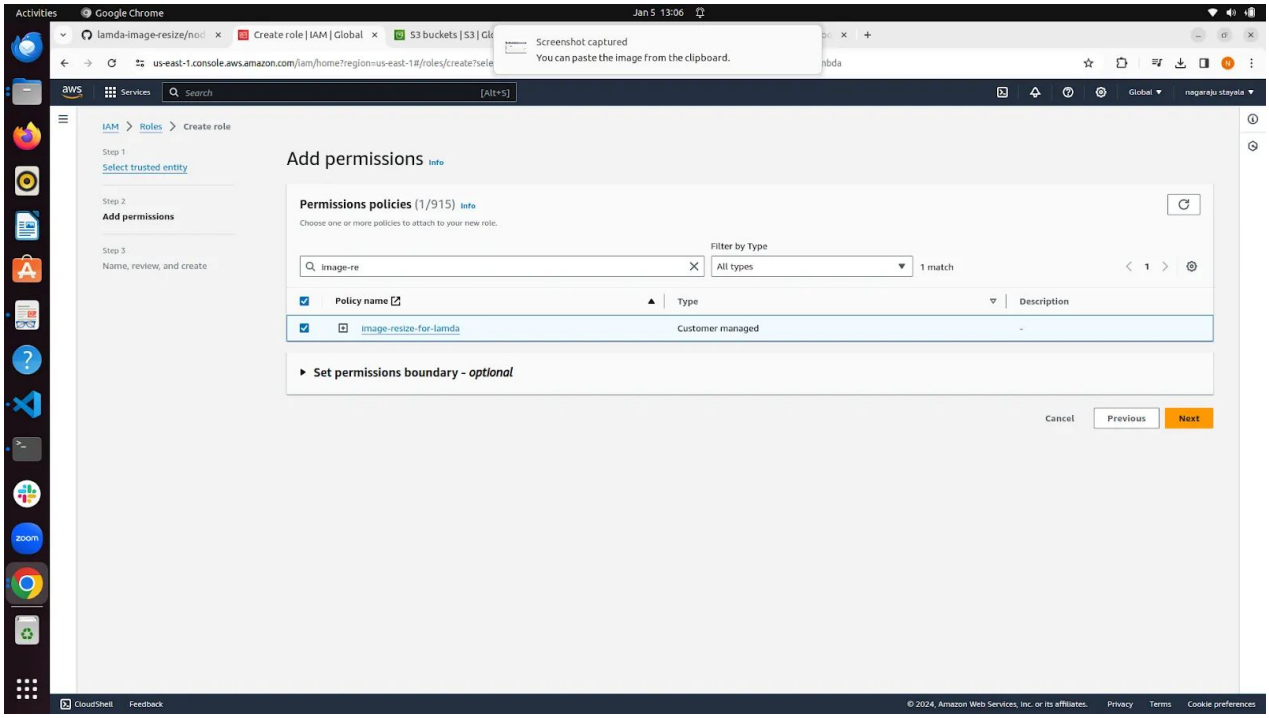


- create a policy with a name

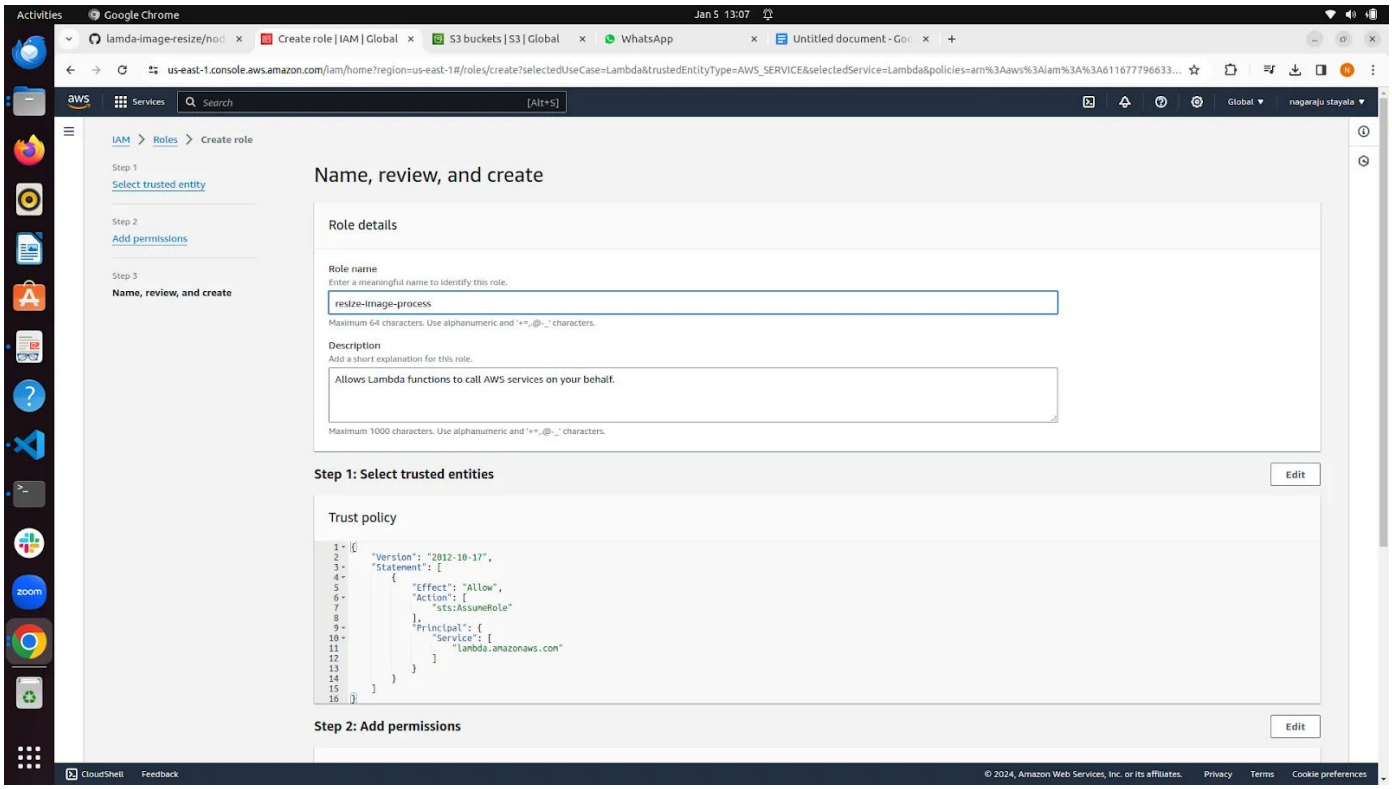


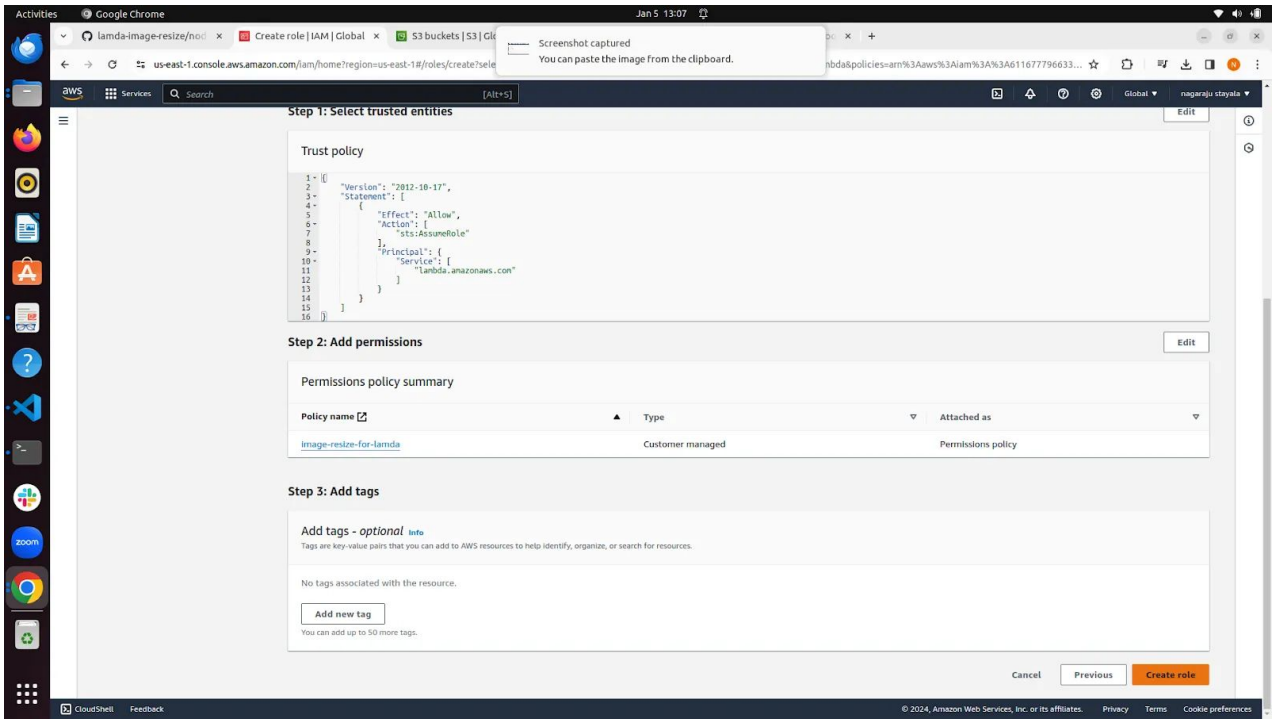
- Select AWS service
- Use case lambda
- Select next



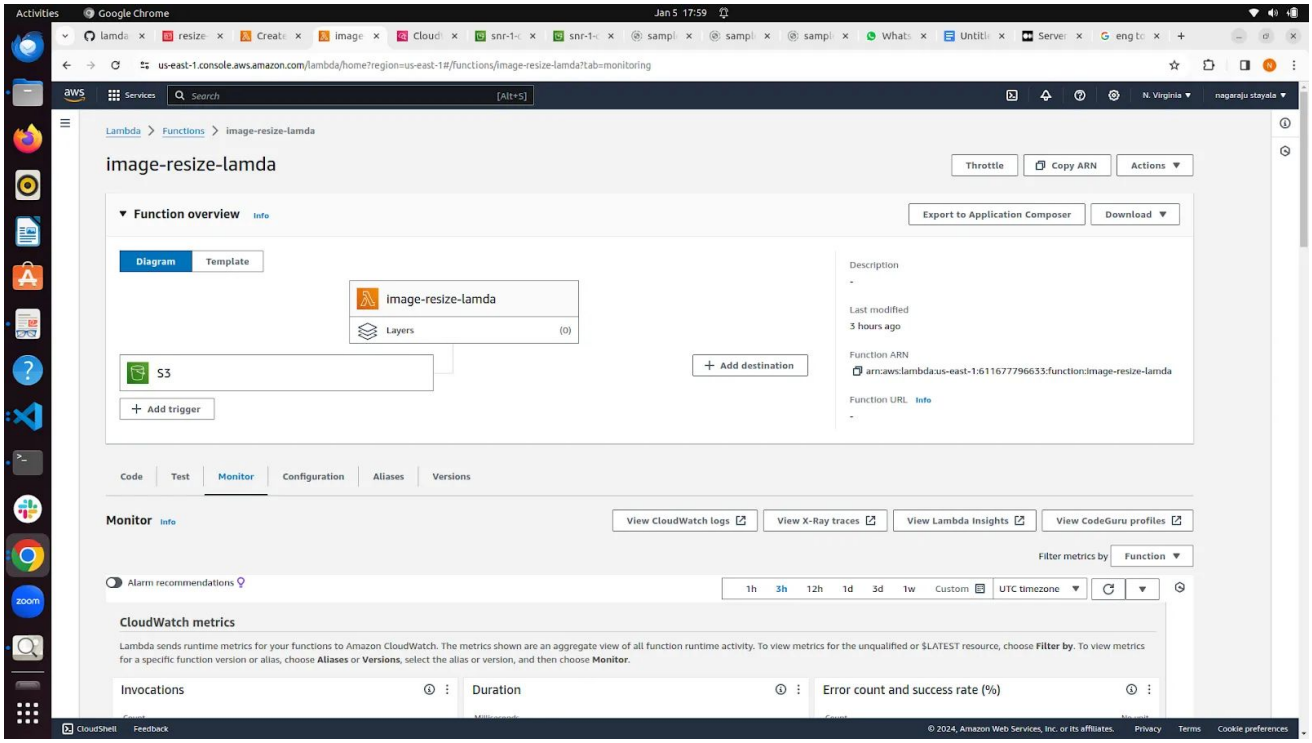


- Select role name

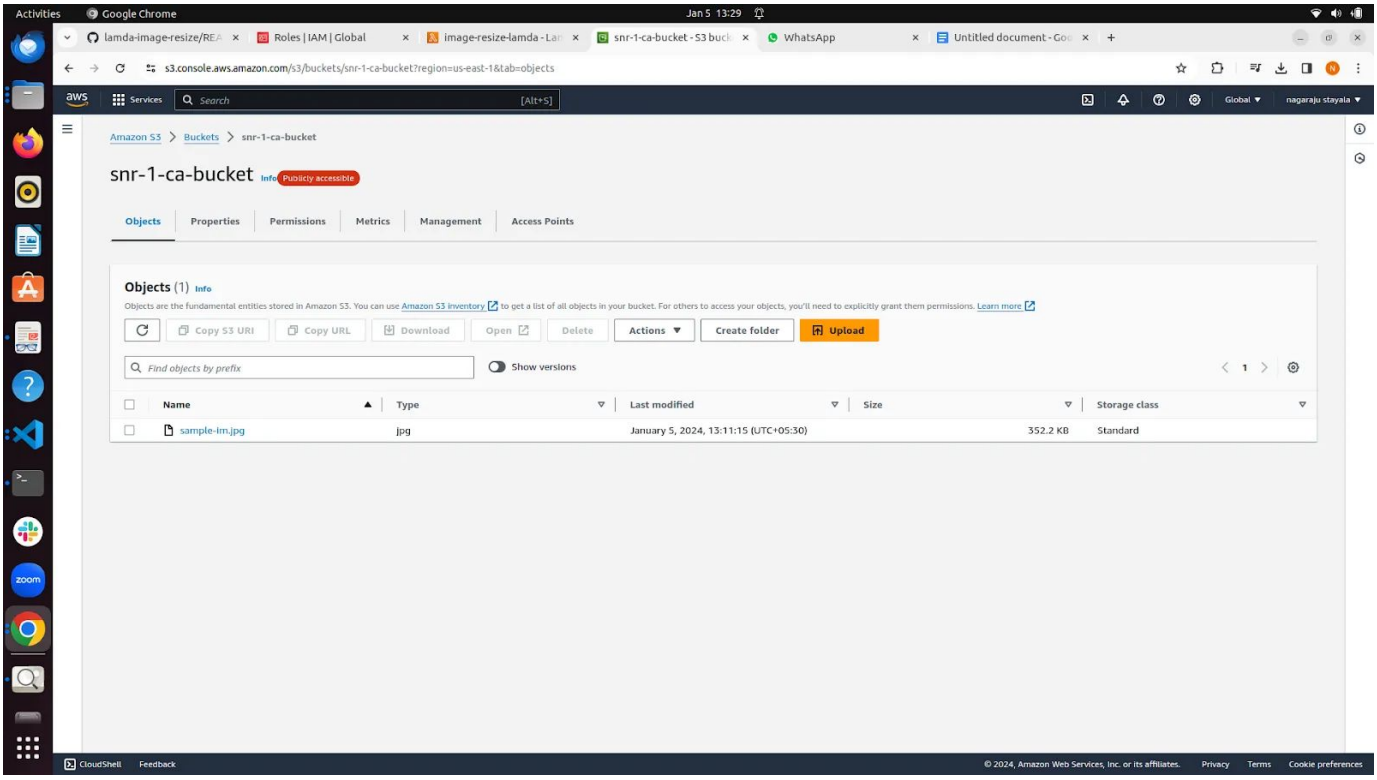




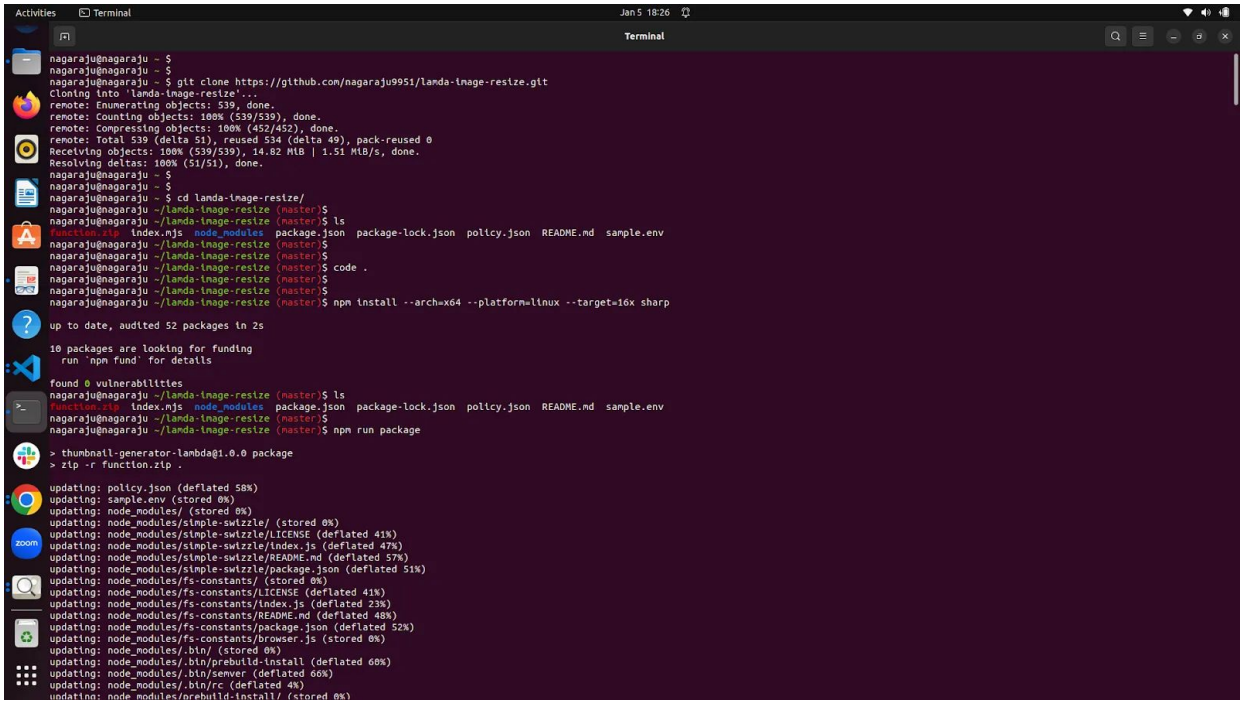
- after selecting Create a lambda function
- Now you can see the function
- Now add triggers
- Select s3 select the bucket the image is uploaded



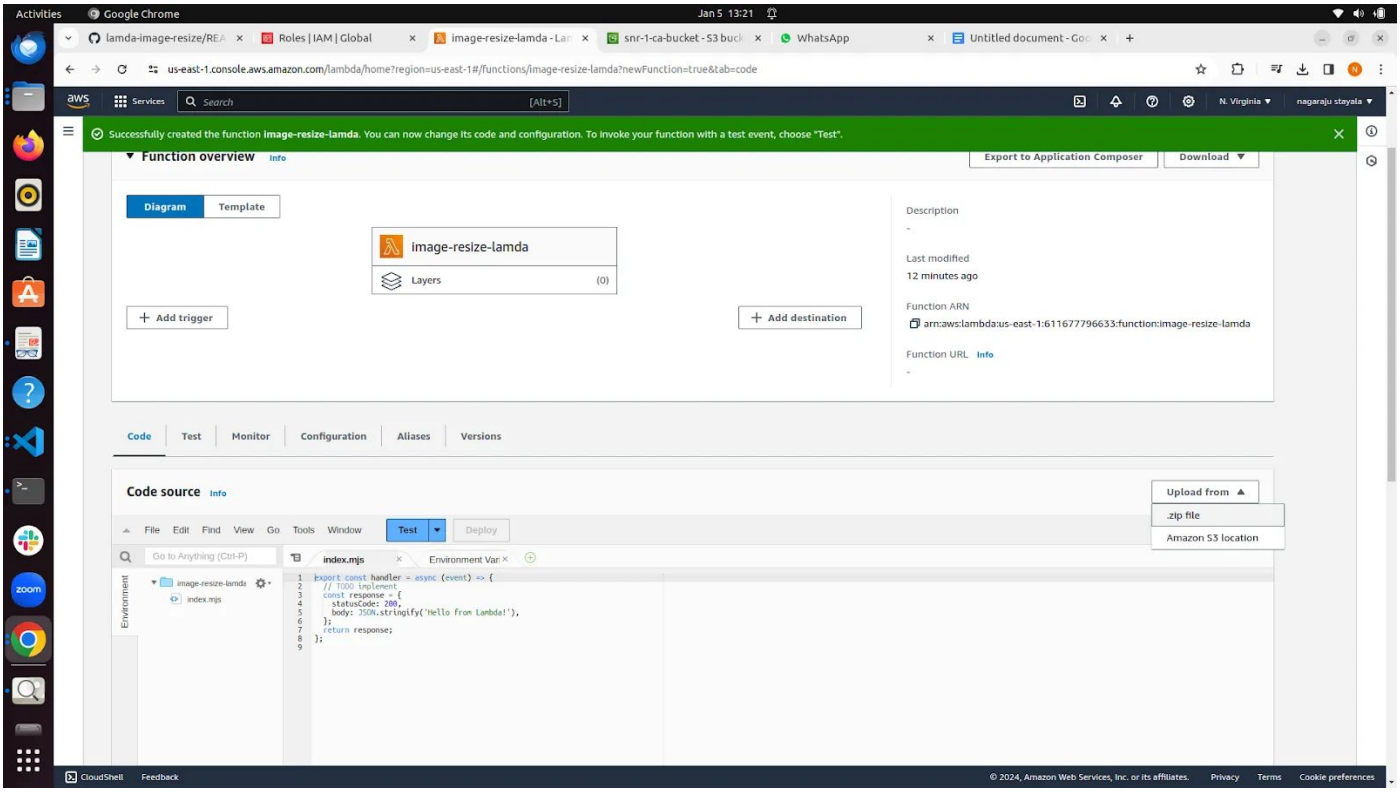
- **Now create two s3 buckets**
- Now upload the jpg image to process lambda to s3



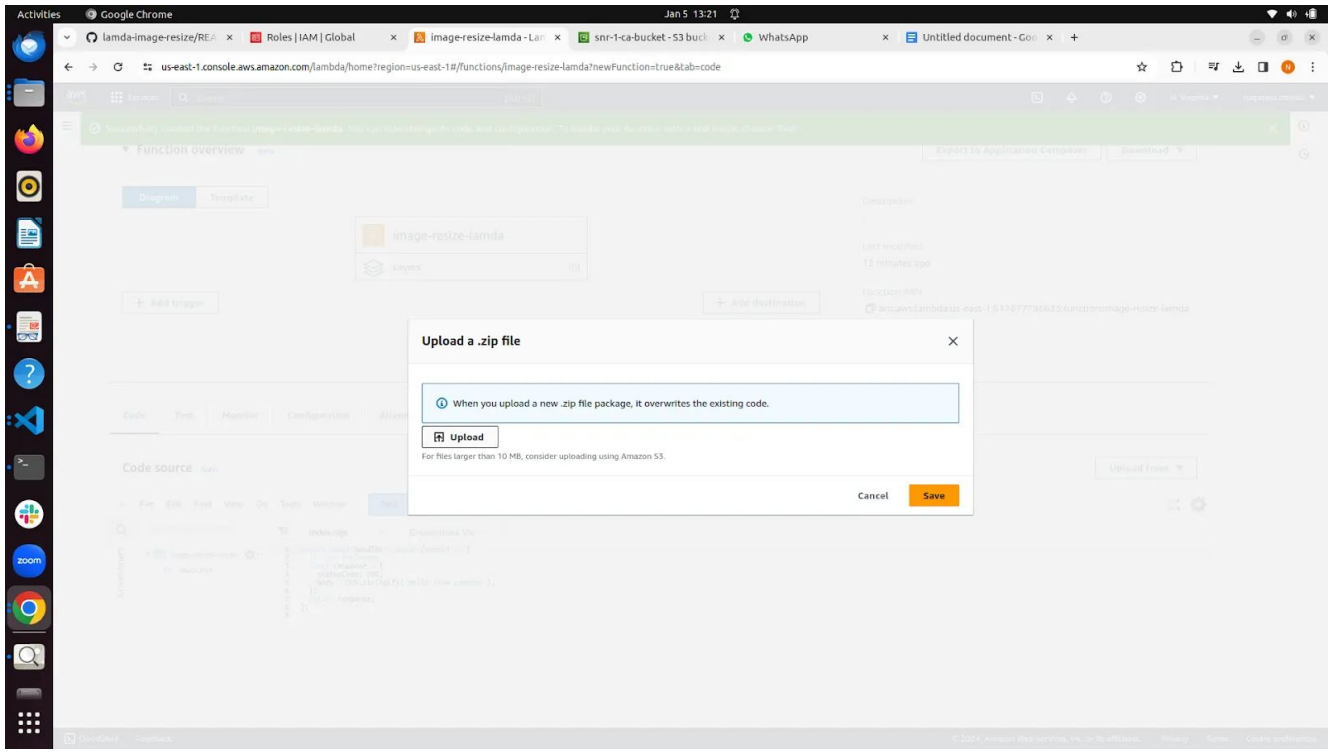
- Using this git command download files locally
- git clone https://github.com/nagaraju9951/lamda-image-resize.git
- Npm run packages
- Now all the packages have been compiled into the function.zip file

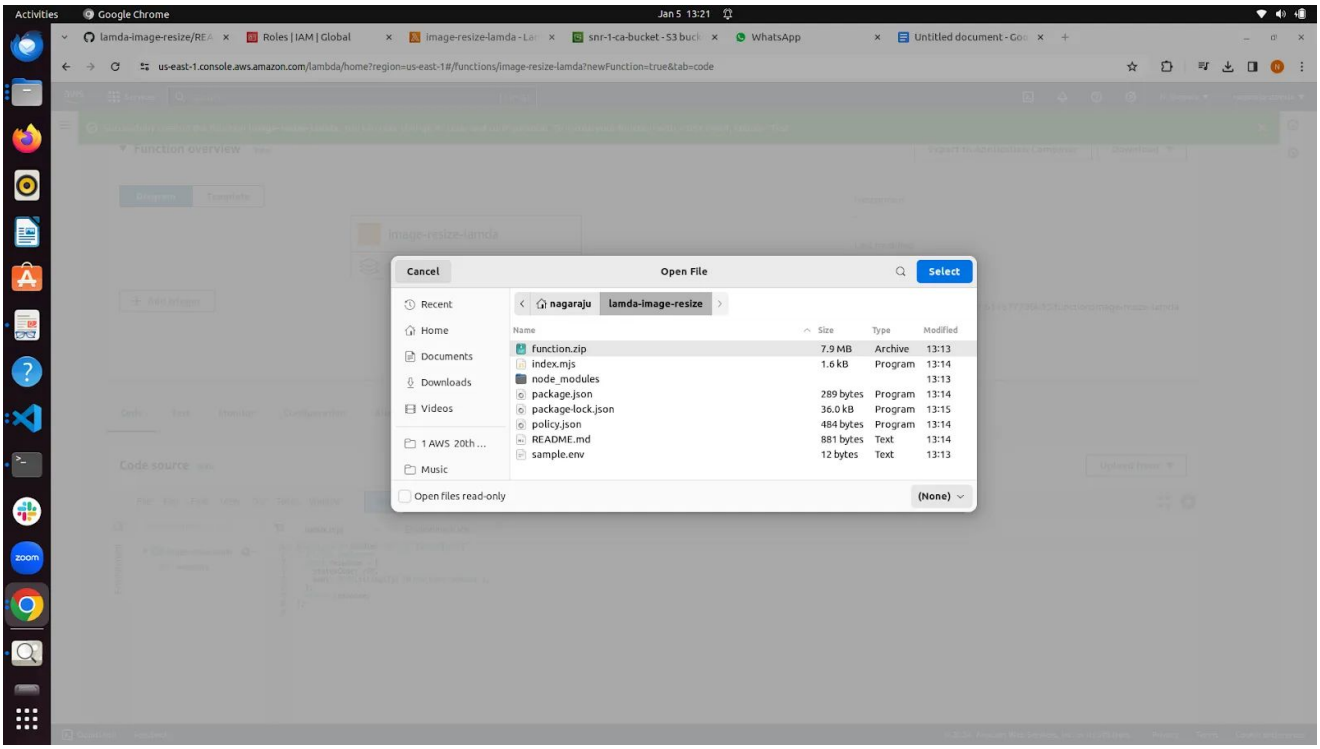


- Now add zip to the lambda function

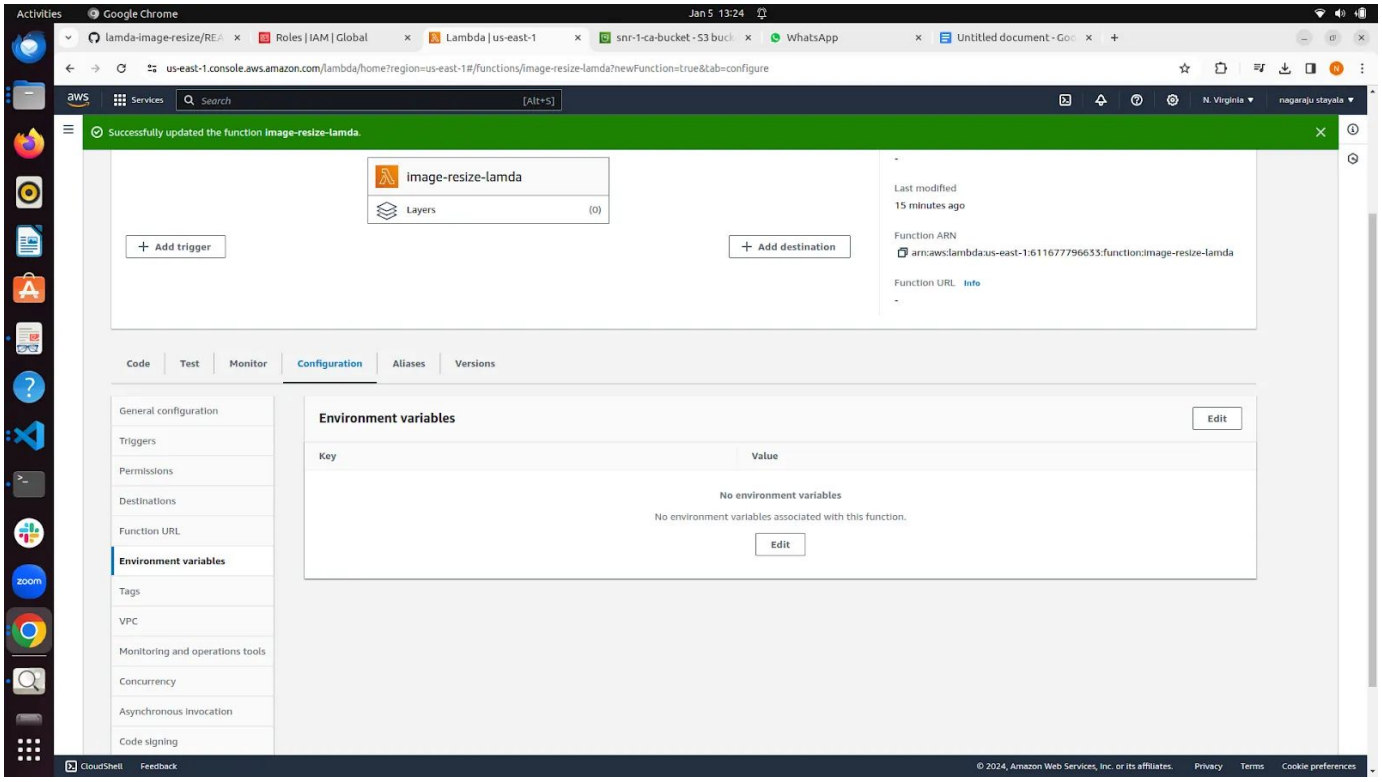


- upload the zip file
- Upload file and save

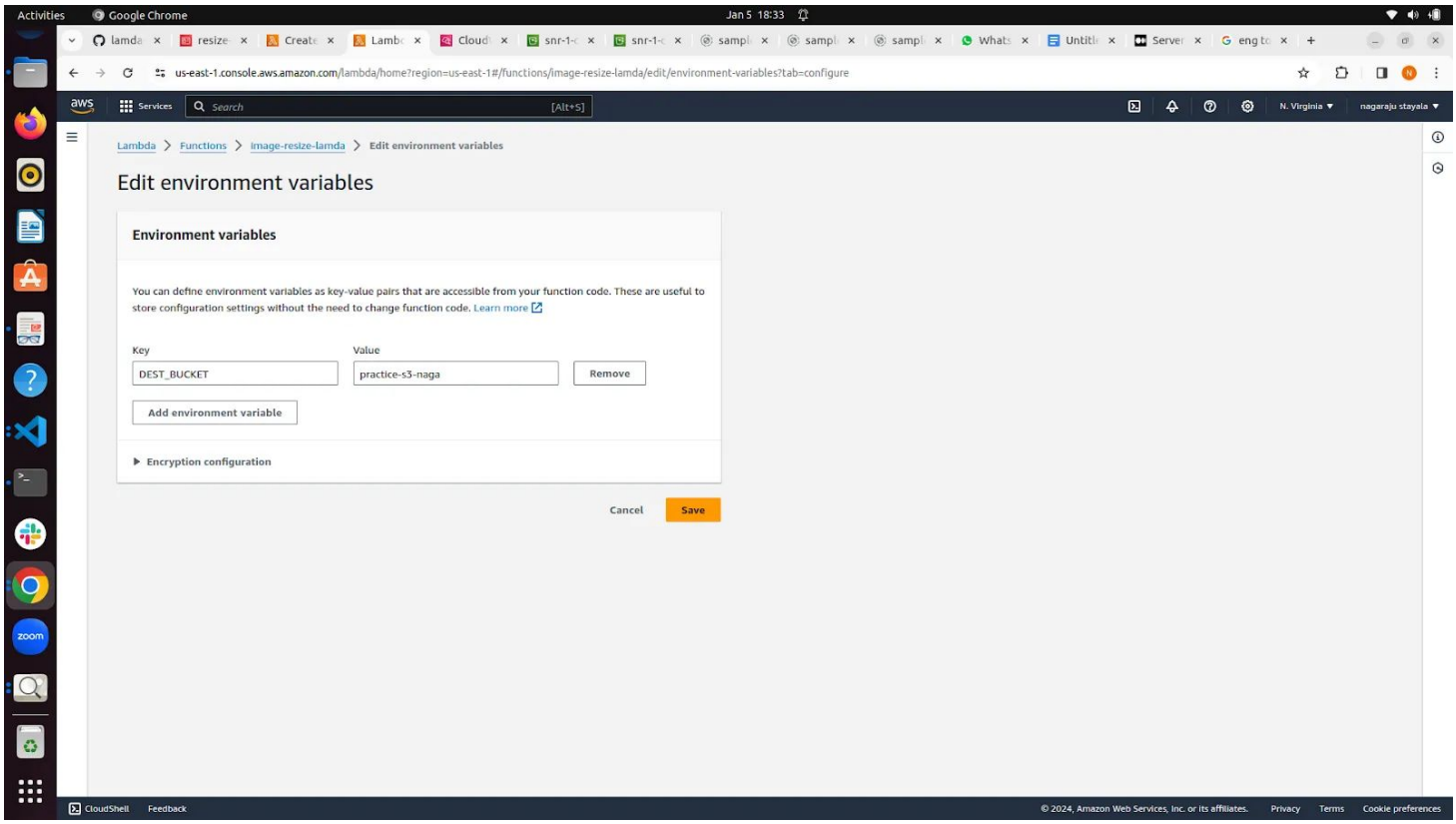




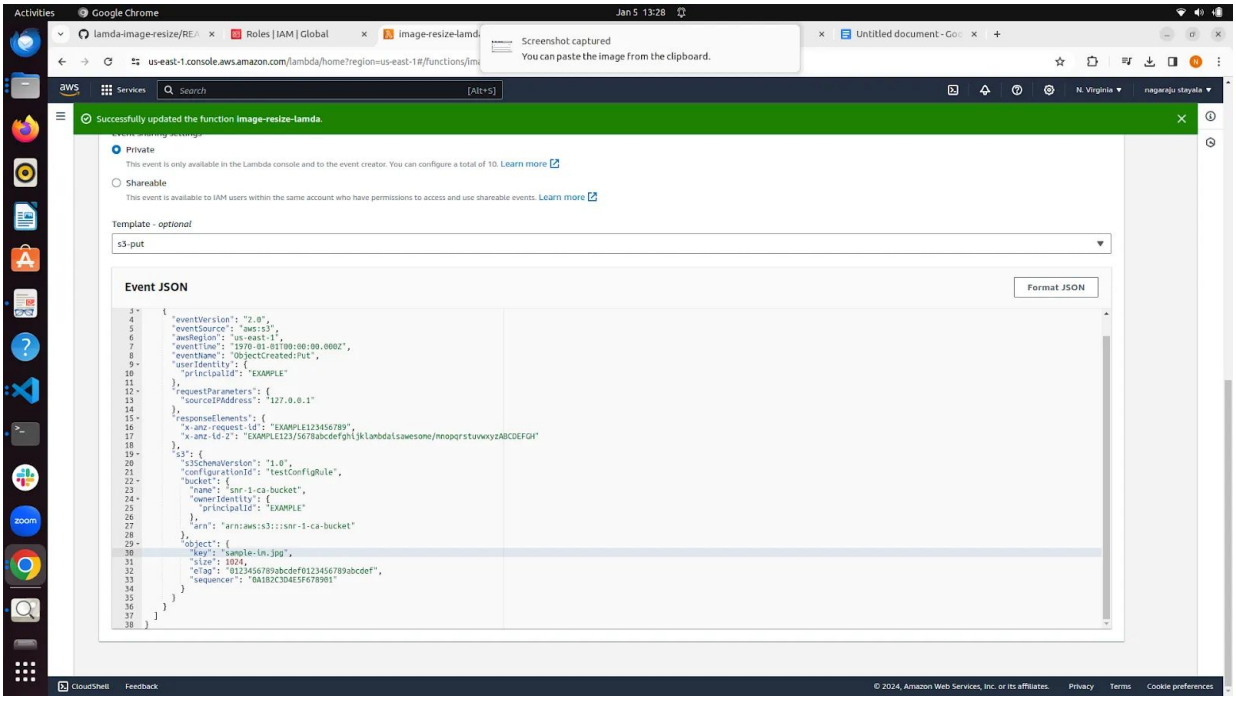
- Now go to Configuration
- Select edit
- now go to environment variables and
- select edit



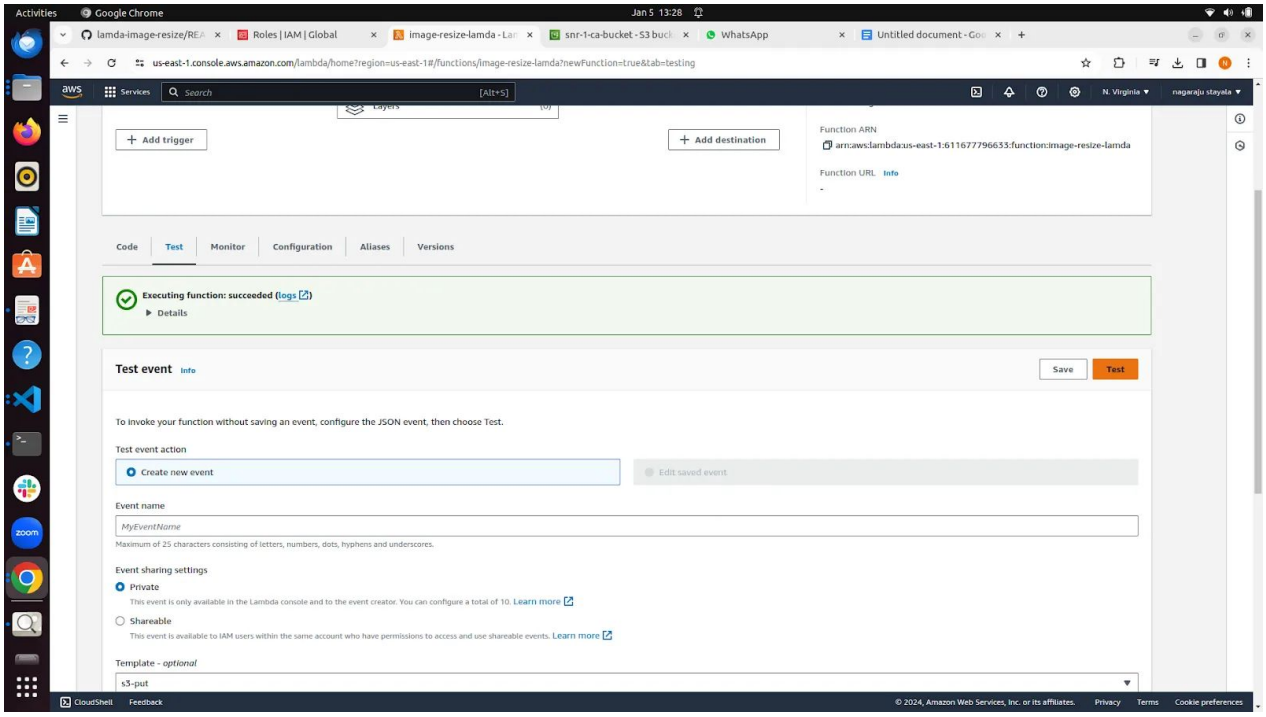
- Now select the destination bucket



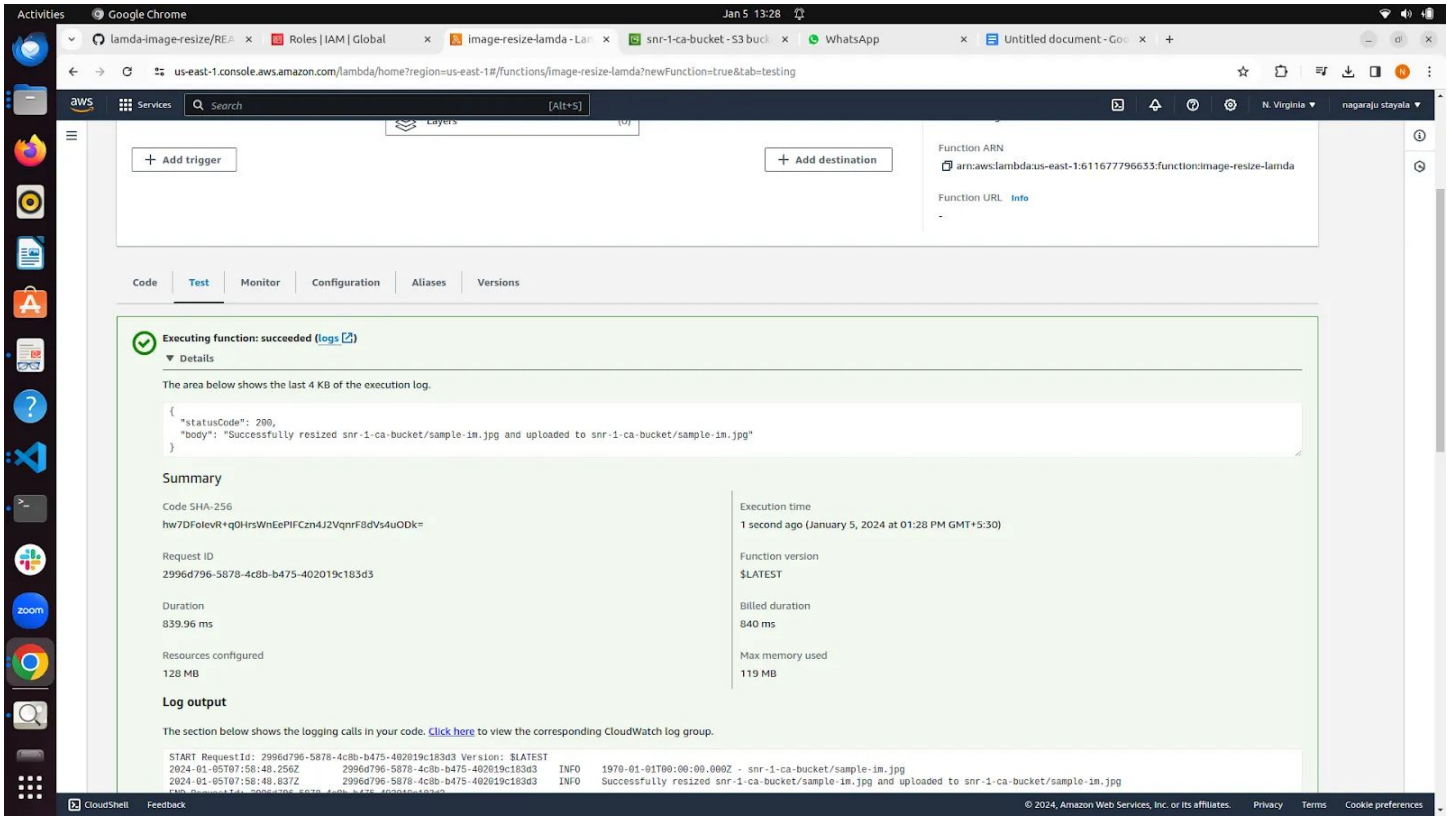
- Save it
- Now go to the test
- Select template
- Select s3 on the search bar
- This code is modified with your existing s3 bucket and the key name is your image name in the s3 bucket



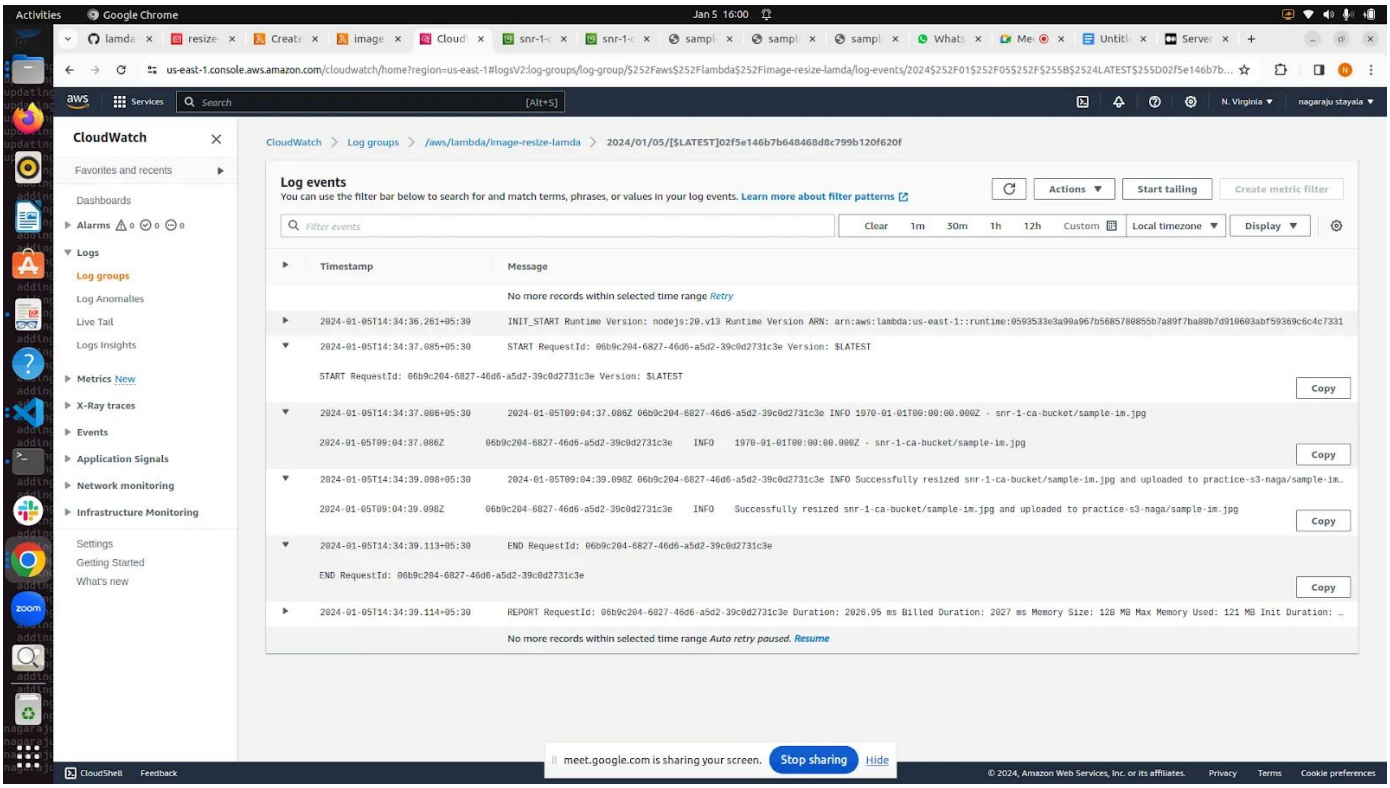
- Now go to test this code



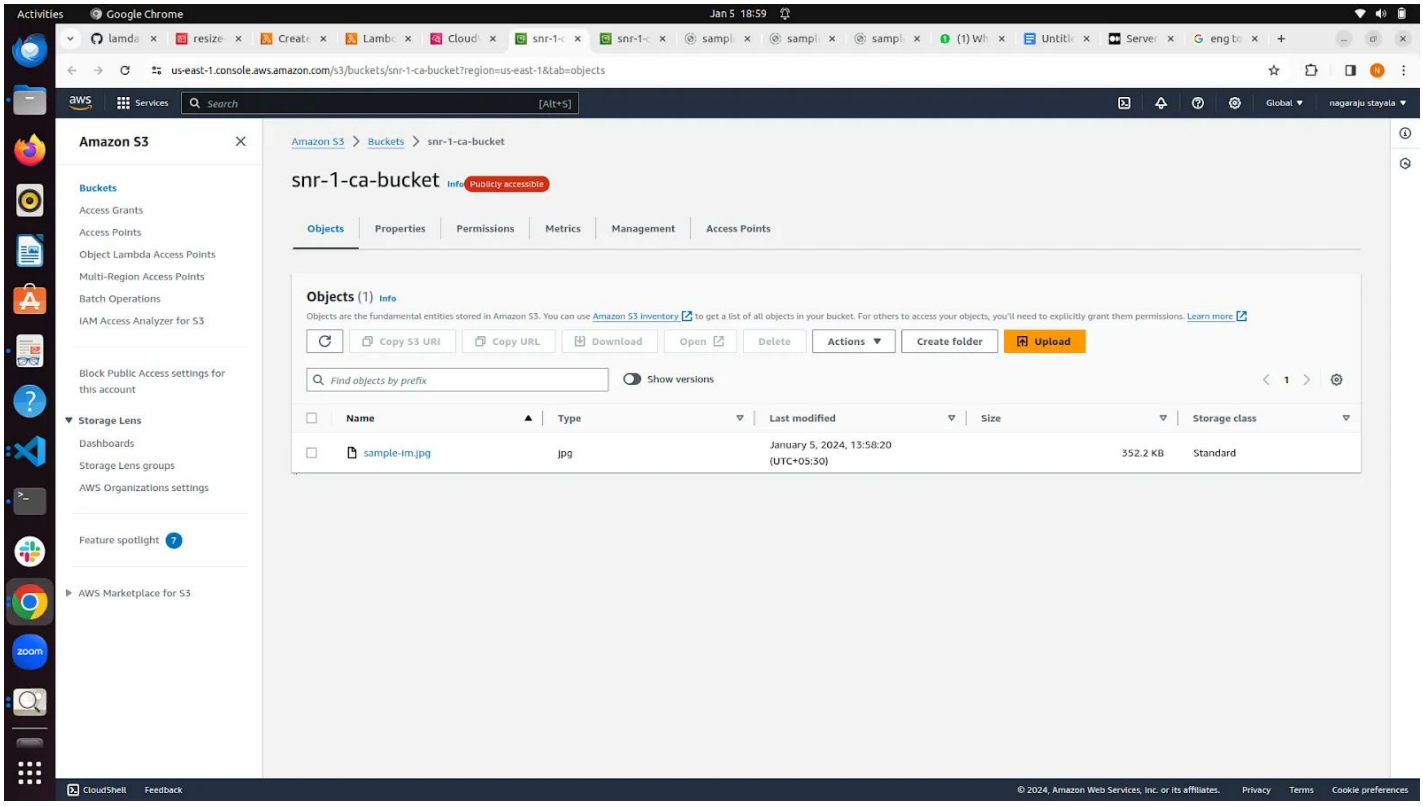
- The code test is successfully run



- 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