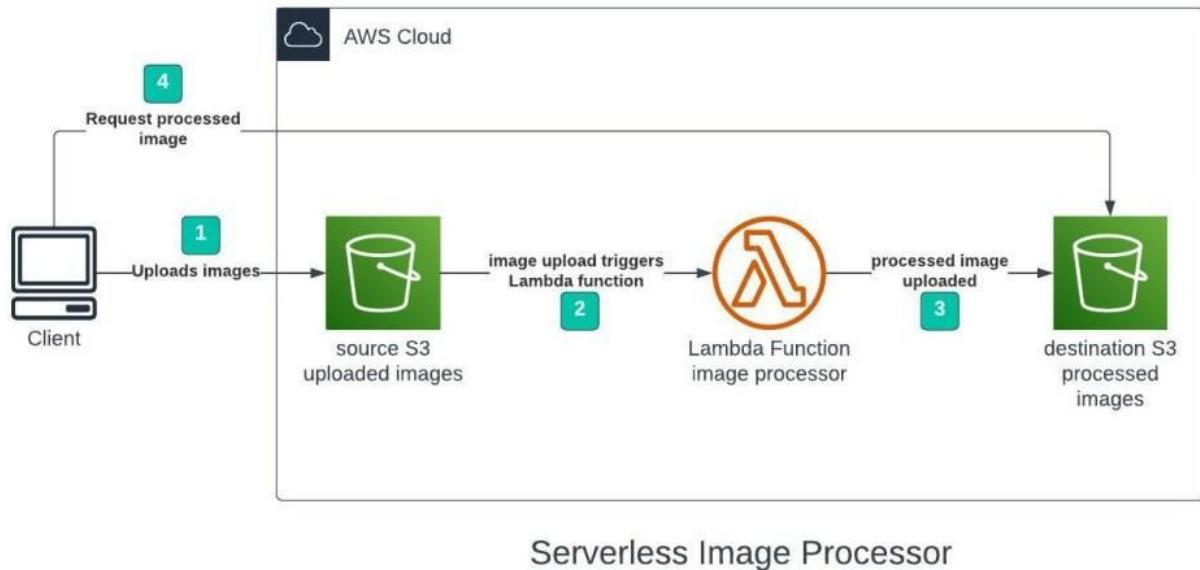


Project 1

Serverless Image Processing

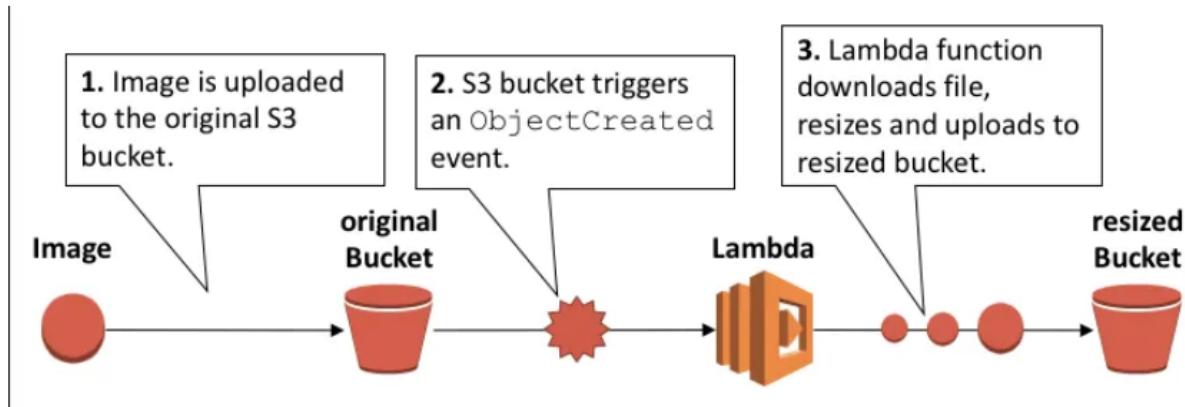


1 TableofContents

1	Introduction	2
1.1	Introduction	3
2	ProblemStatement	4
2.1	Process	5
2.2	Objective	4
2.3	Features	6
3	Implementation	7
3.1	Getting Started with AWS Lambda	7
3.2	Serverless Image Processing Flow	7

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

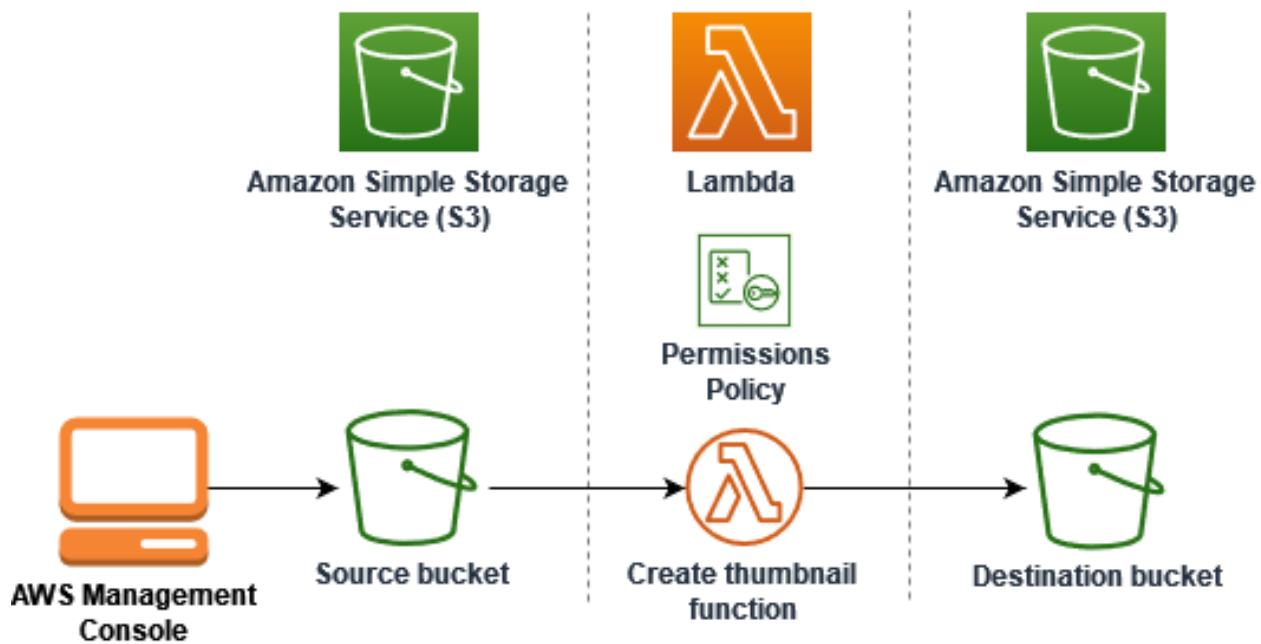
Getting Started with AWS Lambda

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

The screenshot shows the 'Create function' wizard in the AWS Lambda console. The 'Basic information' step is selected. The function name is 'Image-resize-lamda'. The runtime is set to 'Node.js 20.x'. The architecture is 'x86_64'. Under 'Permissions', the 'Execution role' is set to 'resize-image-process'. The 'Advanced settings' section is collapsed.

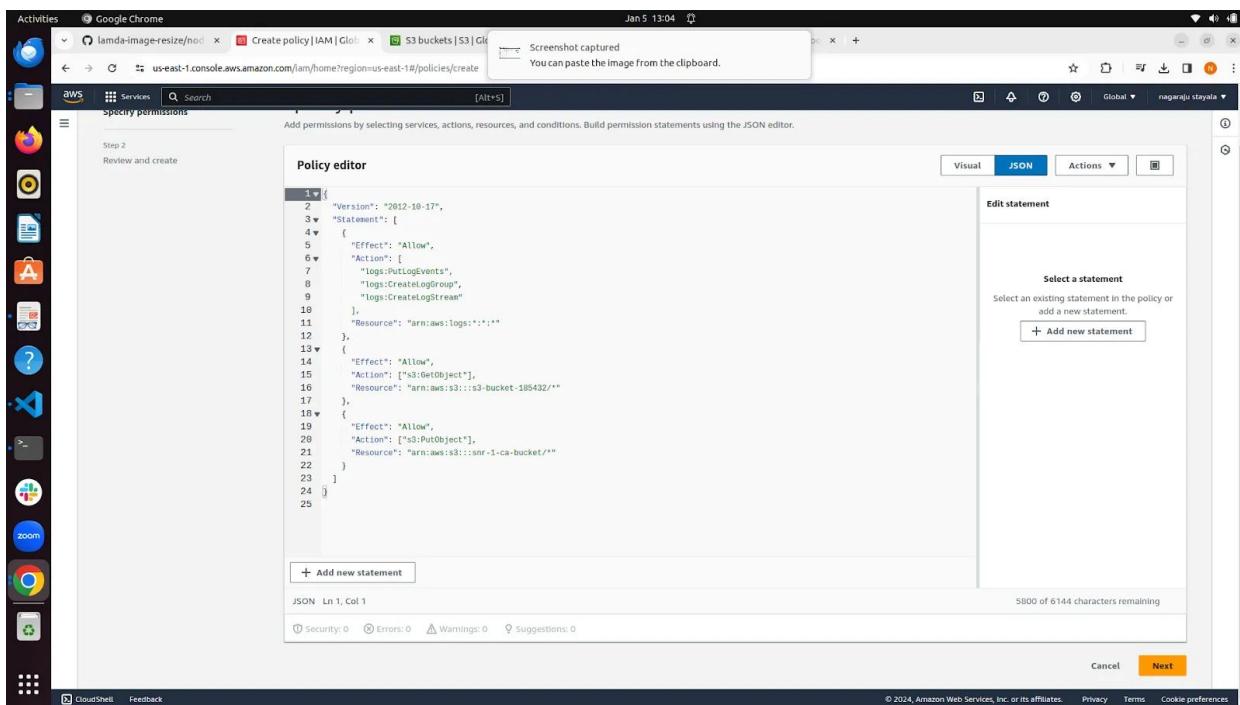
- before going to select a role create an IAM role and policy.

The screenshot shows the 'Policies' page in the AWS IAM console. It lists 1176 managed policies. Some visible policy names include 'AdministratorAccess', 'AlexaForBusinessDeviceSetup', 'AlexaForBusinessFullAccess', 'AlexaForBusinessGatewayExecution', 'AlexaForBusinessIfsDelegatedAccess...', 'AmazonAPIGatewayNetworkProfileService...', 'AlexaForBusinessPolyDelegatedAccess...', 'AlexaForBusinessReadonlyAccess', 'AmazonAPIGatewayAdministrator', 'AmazonAPIGatewayInvokeFullAccess', 'AmazonAPIGatewayPushToCloudWatchLogs...', 'AmazonAppFlowFullAccess', 'AmazonAppFlowReadOnlyAccess', 'AmazonAppStreamFullAccess', 'AmazonAppStreamCAccess', 'AmazonAppStreamReadonlyAccess', and 'AmazonAppStreamServiceAccess'. The 'Create policy' button is visible at the top right.

- 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

The screenshot shows the AWS IAM Roles page. On the left, there's a sidebar with various AWS services like CloudWatch, Lambda, and S3. The main area has a heading 'Roles (24) Info' with a note about IAM roles being identities with specific permissions. A search bar is at the top. Below it is a table with columns for Role name, Trusted entities, and Last activity. The table lists numerous roles, many of which are AWS service-linked roles. At the bottom right of the table, there are buttons for 'Create role' and 'Delete'.

- Select AWS service
- Use case lambda
- Select next

The screenshot shows the 'Create role' wizard, Step 1: Select trusted entity. It has three tabs: Step 1 (Select trusted entity), Step 2 (Add permissions), and Step 3 (Name, review, and create). The first tab is active. It shows a 'Trusted entity type' section with four options:

- AWS service: Allows entities like EC2, Lambda, or others to perform actions in this account.
- AWS account: Allows entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.
- Web identity: Allows users identified by the specified external web identity provider to assume this role to perform actions in this account.
- SAML 2.0 federation: Allows users federated with SAML 2.0 from a corporate directory to perform actions in this account.
- Custom trust policy: Creates a custom trust policy to enable others to perform actions in this account.

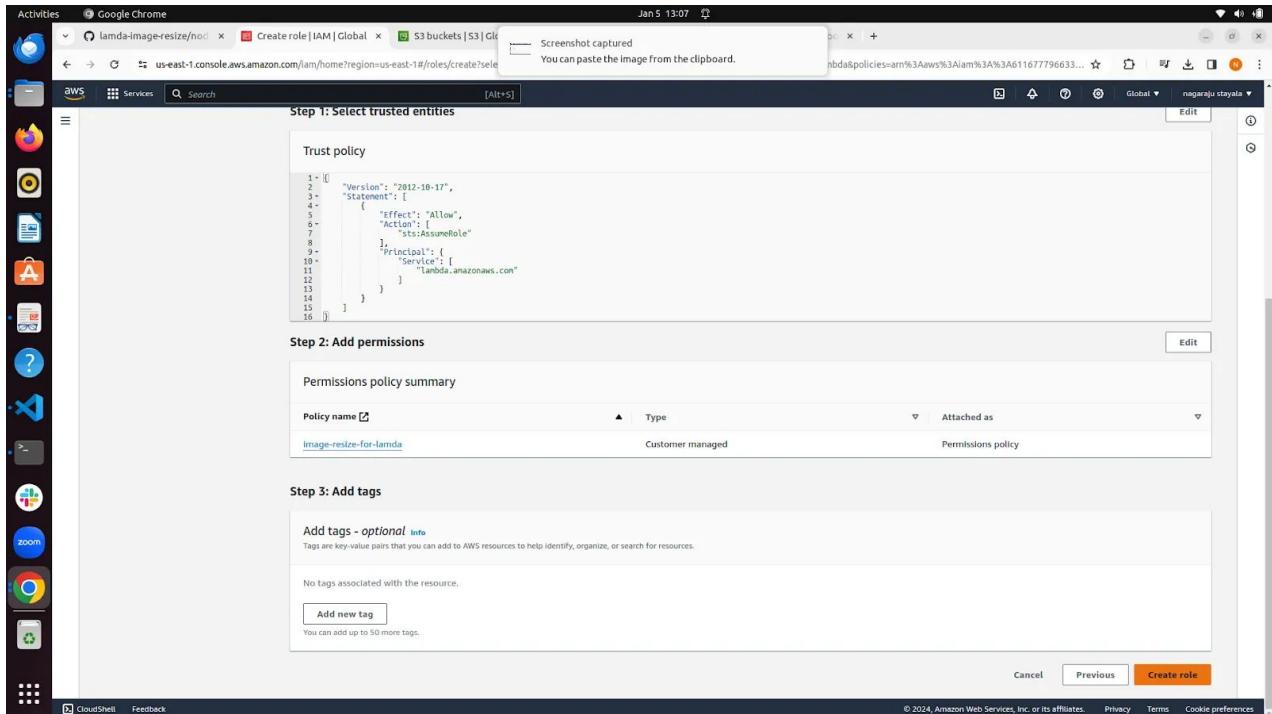
A 'Use case' section below allows selecting a service or use case, with 'Lambda' selected. At the bottom right are 'Cancel' and 'Next' buttons.

The screenshot shows the 'Add permissions' step of the IAM role creation wizard. It lists a single policy named 'image-resize-for-lambda' under 'Customer managed'. A search bar at the top right filters results by type ('All types') and contains the text 'image-re'. Below the search bar is a table with columns for 'Policy name', 'Type', and 'Description'. The table shows one row for the selected policy. At the bottom right of the wizard are 'Cancel', 'Previous', and 'Next' buttons.

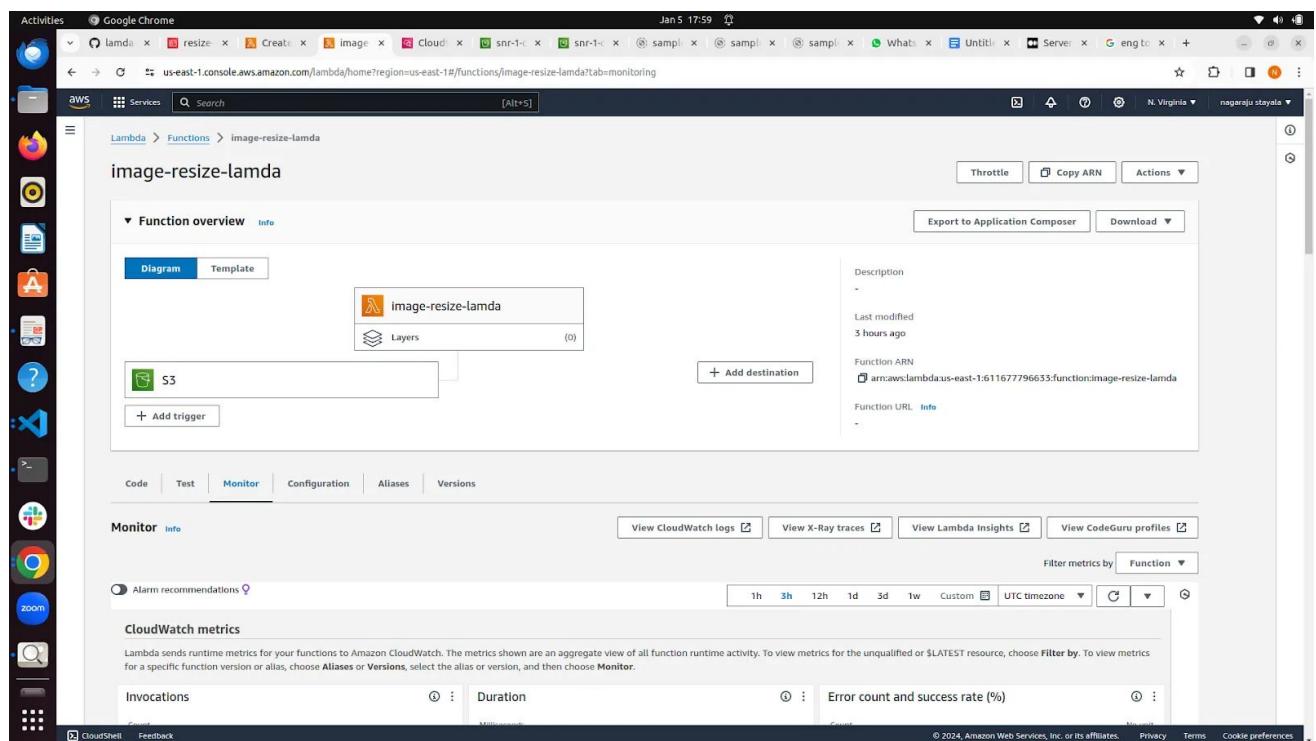
- Select role name

The screenshot shows the 'Name, review, and create' step of the IAM role creation wizard. It includes sections for 'Role details' (role name: 'resize-image-process', description: 'Allows Lambda functions to call AWS services on your behalf.'), 'Step 1: Select trusted entities' (trust policy JSON shown), and 'Step 2: Add permissions' (permissions table). The trust policy JSON is as follows:

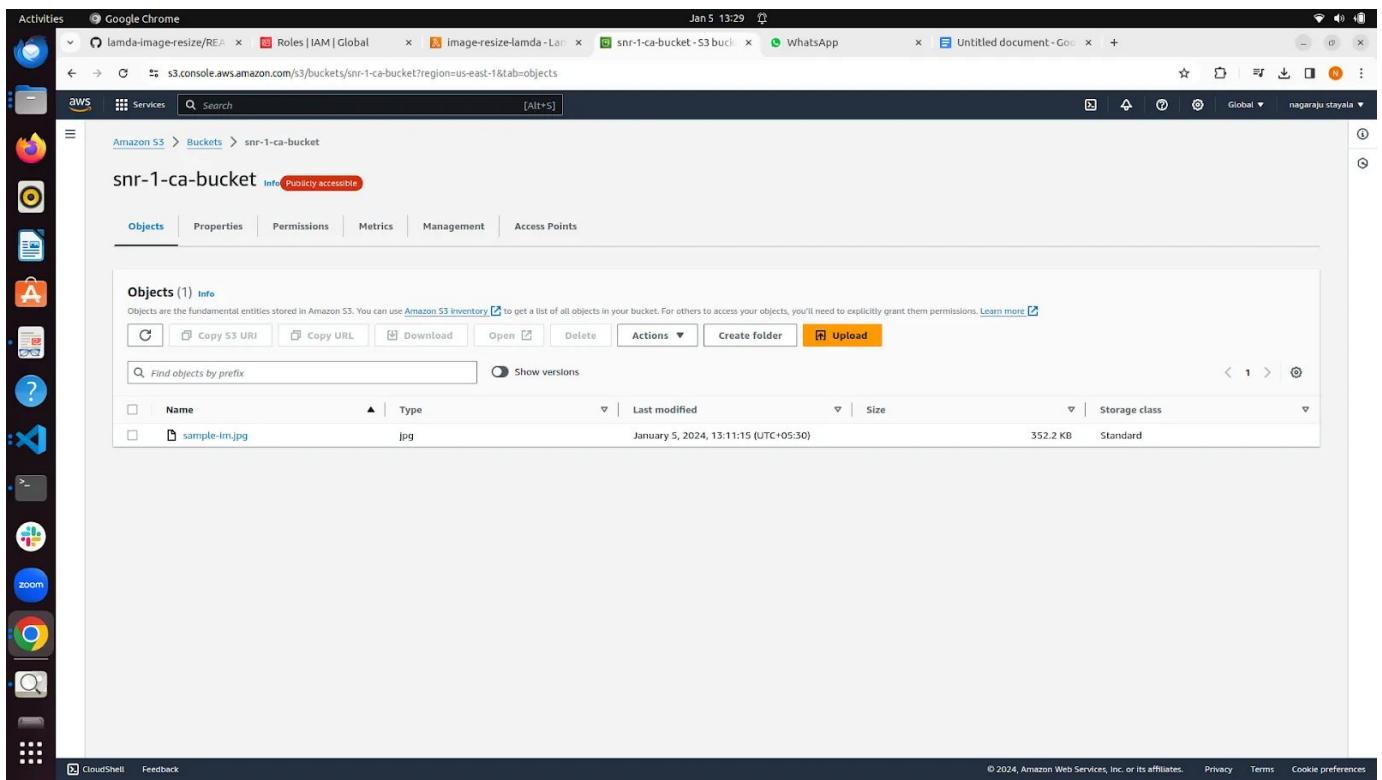
```
1 - [{
2 -   "Version": "2012-10-17",
3 -   "Statement": [
4 -     {
5 -       "Effect": "Allow",
6 -       "Action": [
7 -         "sts:AssumeRole"
8 -       ],
9 -       "Principal": [
10 -         "Service": [
11 -           "Lambda.amazonaws.com"
12 -         ]
13 -       ]
14 -     }
15 -   ]
16 - }]
```



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



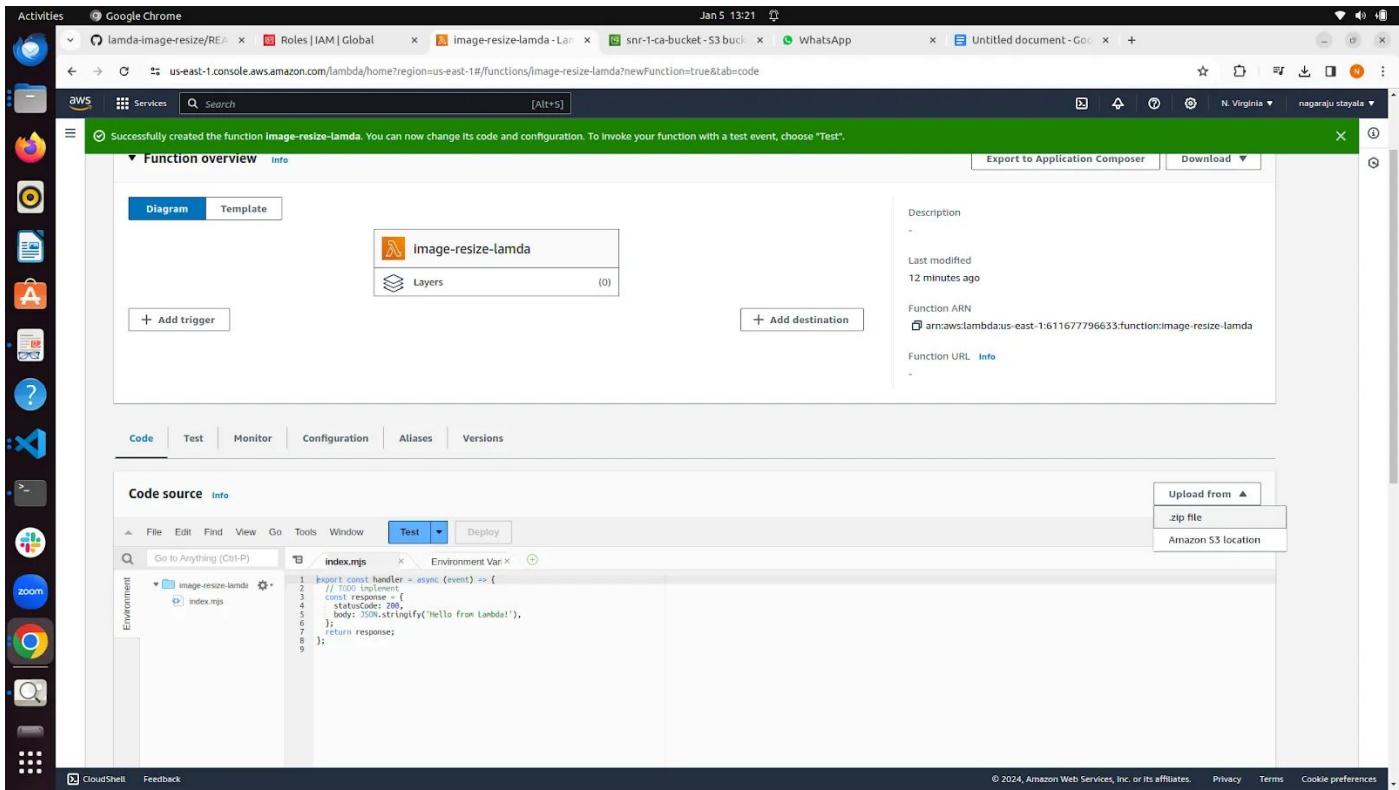
- Now create two s3 buckets
 - Now upload the jpg image to process lambda to s3



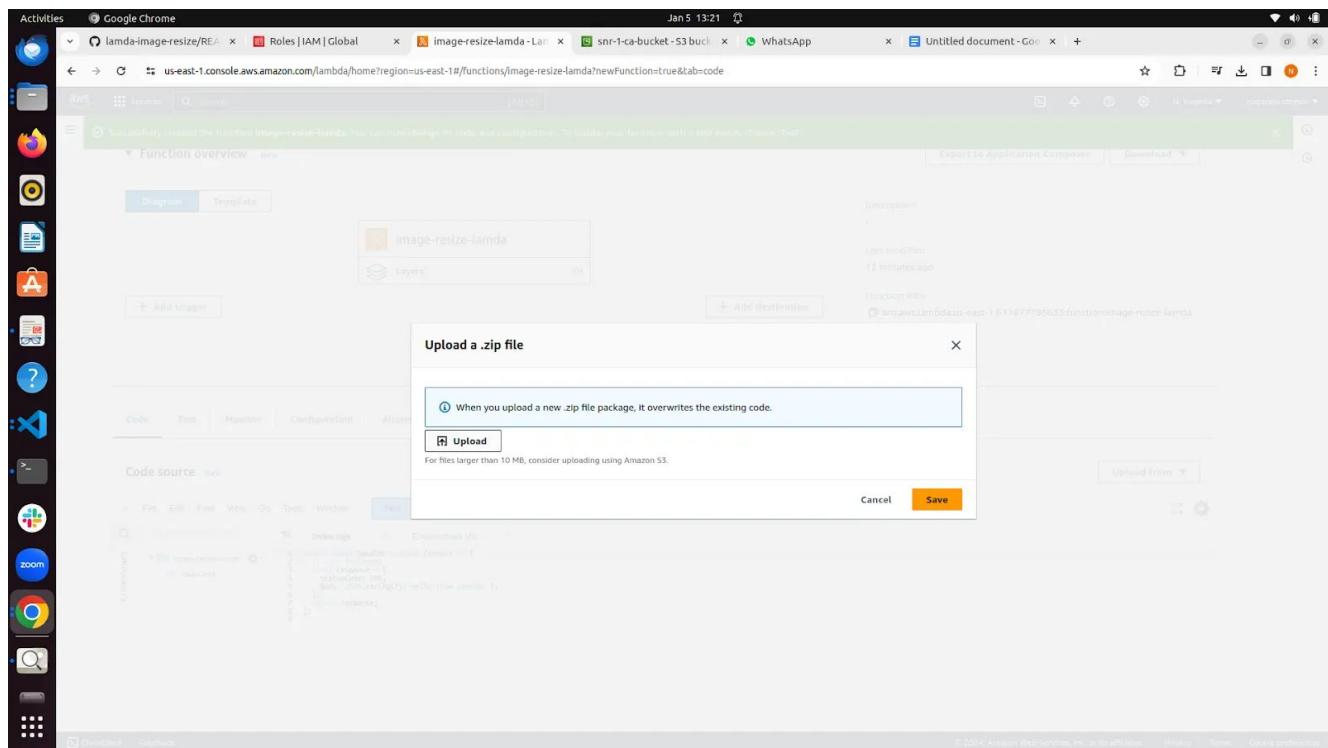
- Using this git command download files locally
 - git clone <https://github.com/nagaraju9951/lamda-image-resize.git>
 - Npm run packages
 - Now all the packages have been compiled into the function.zip file

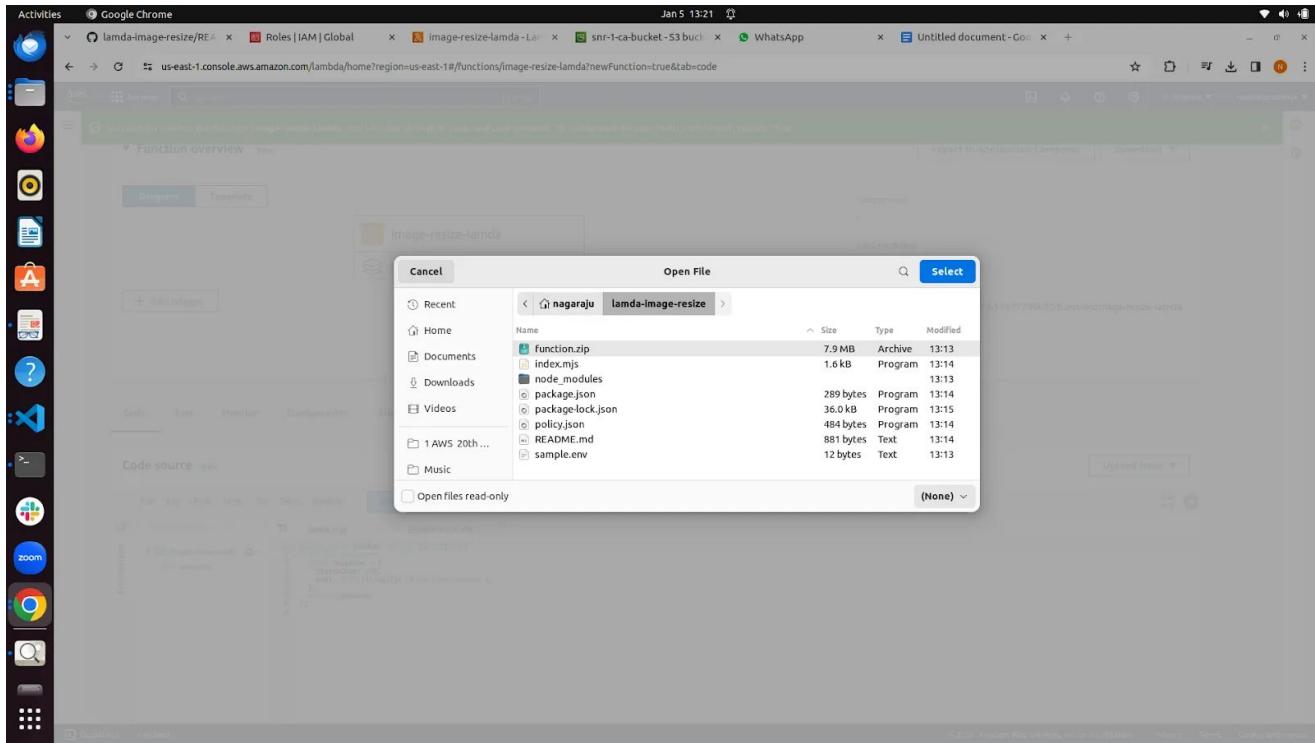
```
Activities Terminal Jan 5 18:26
nagaraju@nagaraju ~ $ nagaraju@nagaraju ~ $ nagaraju@nagaraju ~ $ git clone https://github.com/nagaraju9951/lambda-image-resize.git
Cloning into 'lambda-image-resize'...
remote: Enumerating objects: 539, done.
remote: Counting objects: 100% (539/539), done.
remote: Compressing objects: 100% (452/452), done.
remote: Total 539 (delta 51), reused 534 (delta 49), pack-reused 0
Receiving objects: 100% (539/539), 14.82 MB | 1.51 MB/s, done.
Resolving deltas: 100% (51/51), done.
nagaraju@nagaraju ~ $ nagaraju@nagaraju ~ $ nagaraju@nagaraju ~ $ cd lambda-image-resize/
nagaraju@nagaraju ~/lambda-image-resize [master] $ nagaraju@nagaraju ~/lambda-image-resize [master] $ ls
index.js node_modules package.json package-lock.json policy.json README.md sample.env
nagaraju@nagaraju ~/lambda-image-resize [master] $ nagaraju@nagaraju ~/lambda-image-resize [master] $ nagaraju@nagaraju ~/lambda-image-resize [master] $ code .
nagaraju@nagaraju ~/lambda-image-resize [master] $ nagaraju@nagaraju ~/lambda-image-resize [master] $ nagaraju@nagaraju ~/lambda-image-resize [master] $ nagaraju@nagaraju ~/lambda-image-resize [master] $ npm install --arch=x64 --platform=linux --target=16 sharp
nagaraju@nagaraju ~/lambda-image-resize [master] $ npm install --arch=x64 --platform=linux --target=16 sharp
[?] up to date, audited 52 packages in 2s
10 packages are looking for funding
  run `npm fund` for details
[?] found 0 vulnerabilities
nagaraju@nagaraju ~/lambda-image-resize [master] $ ls
functions index.js node_modules package.json package-lock.json policy.json README.md sample.env
nagaraju@nagaraju ~/lambda-image-resize [master] $ nagaraju@nagaraju ~/lambda-image-resize [master] $ npm run package
[?] > thumbnail-generator-lambda@1.0.0 package
[?] > zlp -r function.zip .
[?] updating: policy.json (deflated 58%)
[?] updating: sample.env (stored 0%)
[?] updating: node_modules/ (stored 0%)
[?] updating: node_modules/simple-swizzle/ (stored 0%)
[?] updating: node_modules/simple-swizzle/LICENSE (deflated 0%)
[?] updating: node_modules/simple-swizzle/LICENSE (deflated 47%)
[?] updating: node_modules/simple-swizzle/README.md (deflated 57%)
[?] updating: node_modules/simple-swizzle/package.json (deflated 51%)
[?] updating: node_modules/fs-constants/ (stored 0%)
[?] updating: node_modules/fs-constants/LICENSE (deflated 41%)
[?] updating: node_modules/fs-constants/package.json (deflated 23%)
[?] updating: node_modules/fs-constants/README.md (deflated 48%)
[?] updating: node_modules/fs-constants/package.json (deflated 52%)
[?] updating: node_modules/fs-constants/browser.js (stored 0%)
[?] updating: node_modules/.bin/ (stored 0%)
[?] updating: node_modules/.bin/prebuild-install (deflated 60%)
[?] updating: node_modules/.bin/rollup (deflated 60%)
[?] updating: node_modules/.bin/rcc (deflated 4%)
[?] updating: node_modules/archbuild/install/ (stored 0%)
```

- Now add zip to the lambda function



- upload the zip file
- Upload file and save

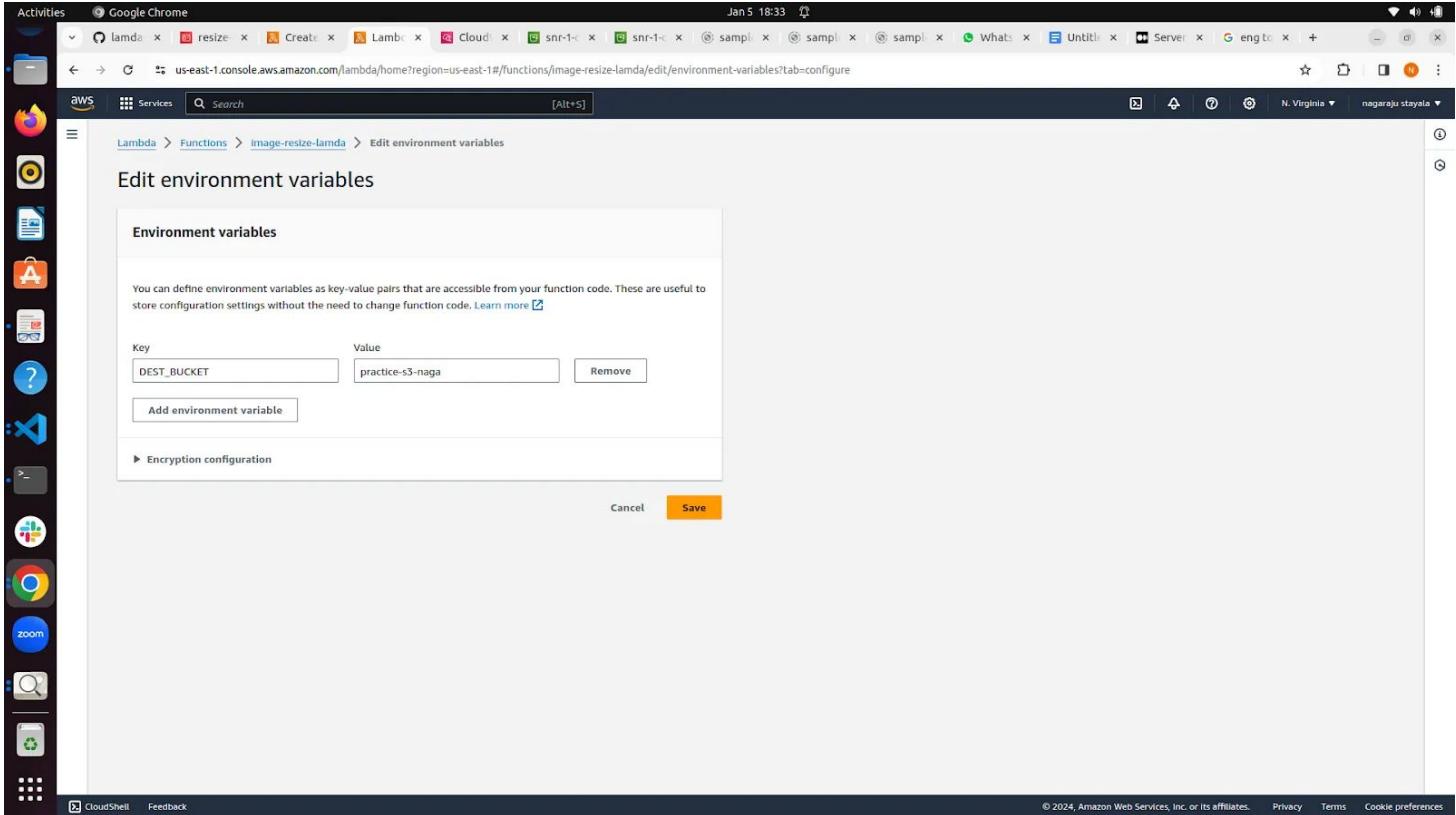




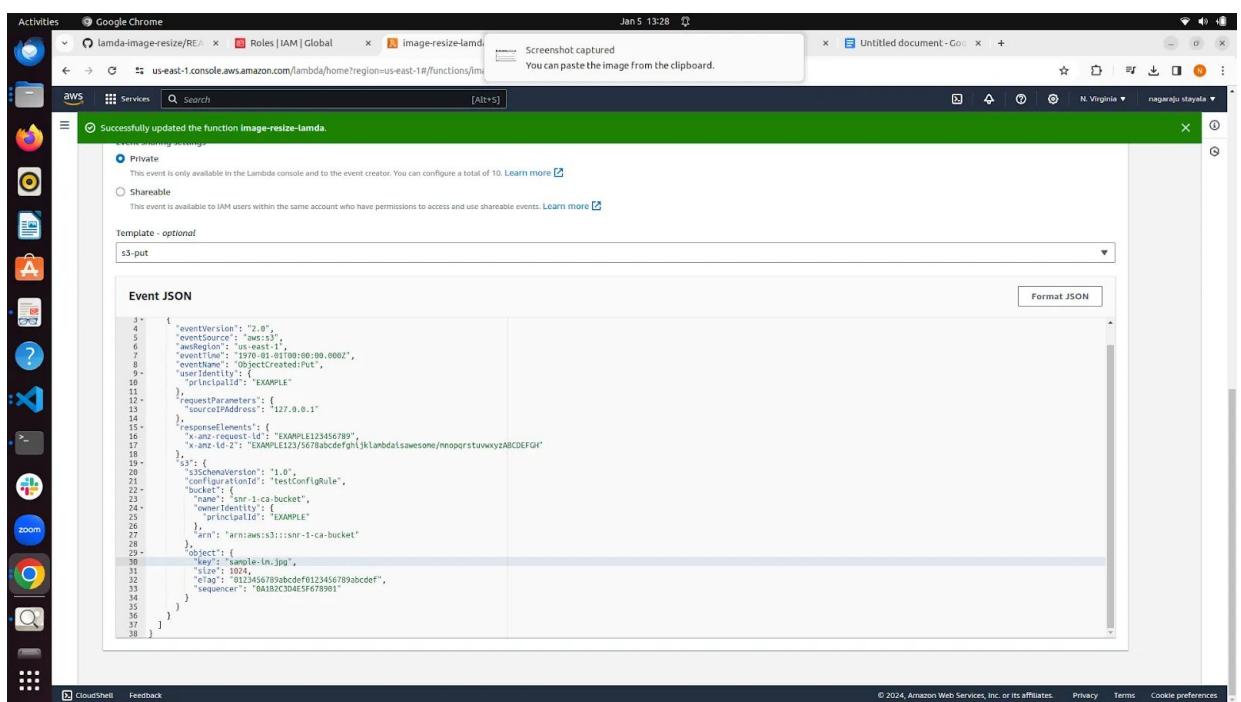
- Now go to Configuration
- Select edit
- now go to environment variables and
- select edit

A screenshot of the AWS Lambda function configuration page. The URL is 'us-east-1.console.aws.amazon.com/lambda/home?region=us-east-1#/functions/image-resize-lambda?newFunction=true&tab=configuration'. The 'Configuration' tab is selected. On the left, there is a sidebar with various options: General configuration, Triggers, Permissions, Destinations, Function URL, Environment variables, Tags, VPC, Monitoring and operations tools, Concurrency, Asynchronous Invocation, and Code signing. The 'Environment variables' section is currently active. It shows a table with one row: 'Key' (empty) and 'Value' (empty). Below the table, it says 'No environment variables' and has an 'Edit' button. At the top, there is a green success message: 'Successfully updated the function image-resize-lambda.' The Lambda function name 'image-resize-lambda' is also visible in the top bar.

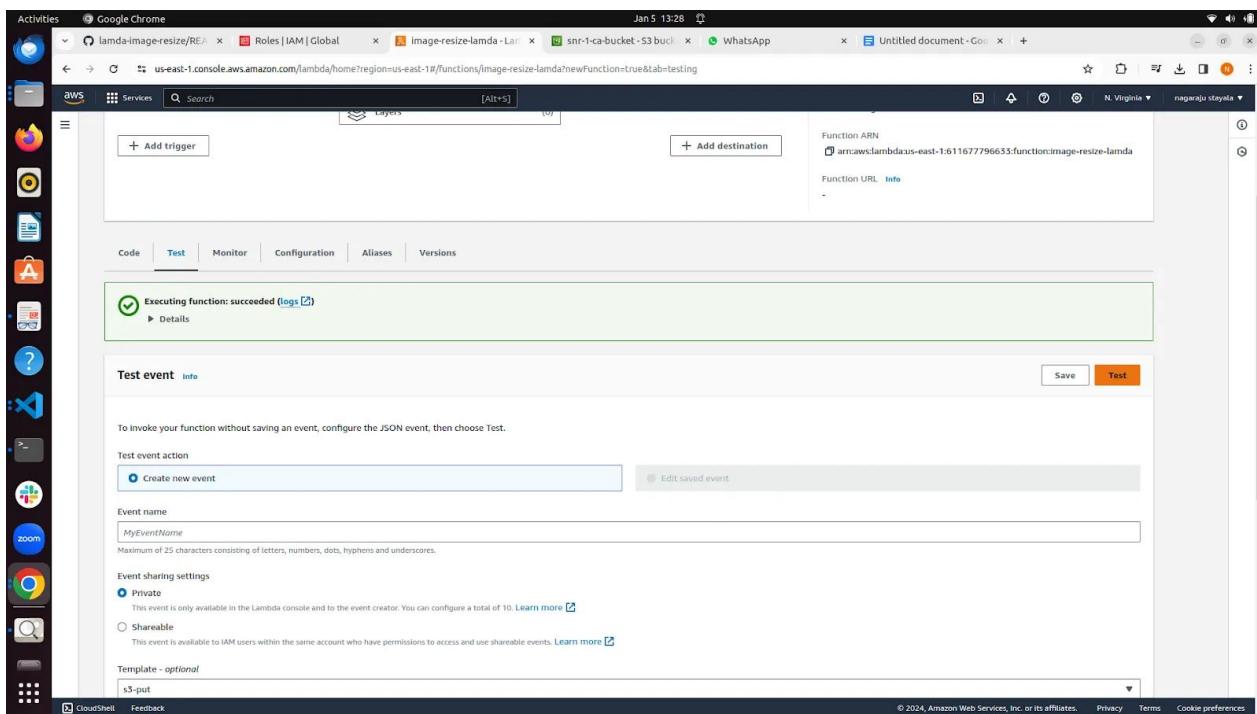
- Now select the destination bucket



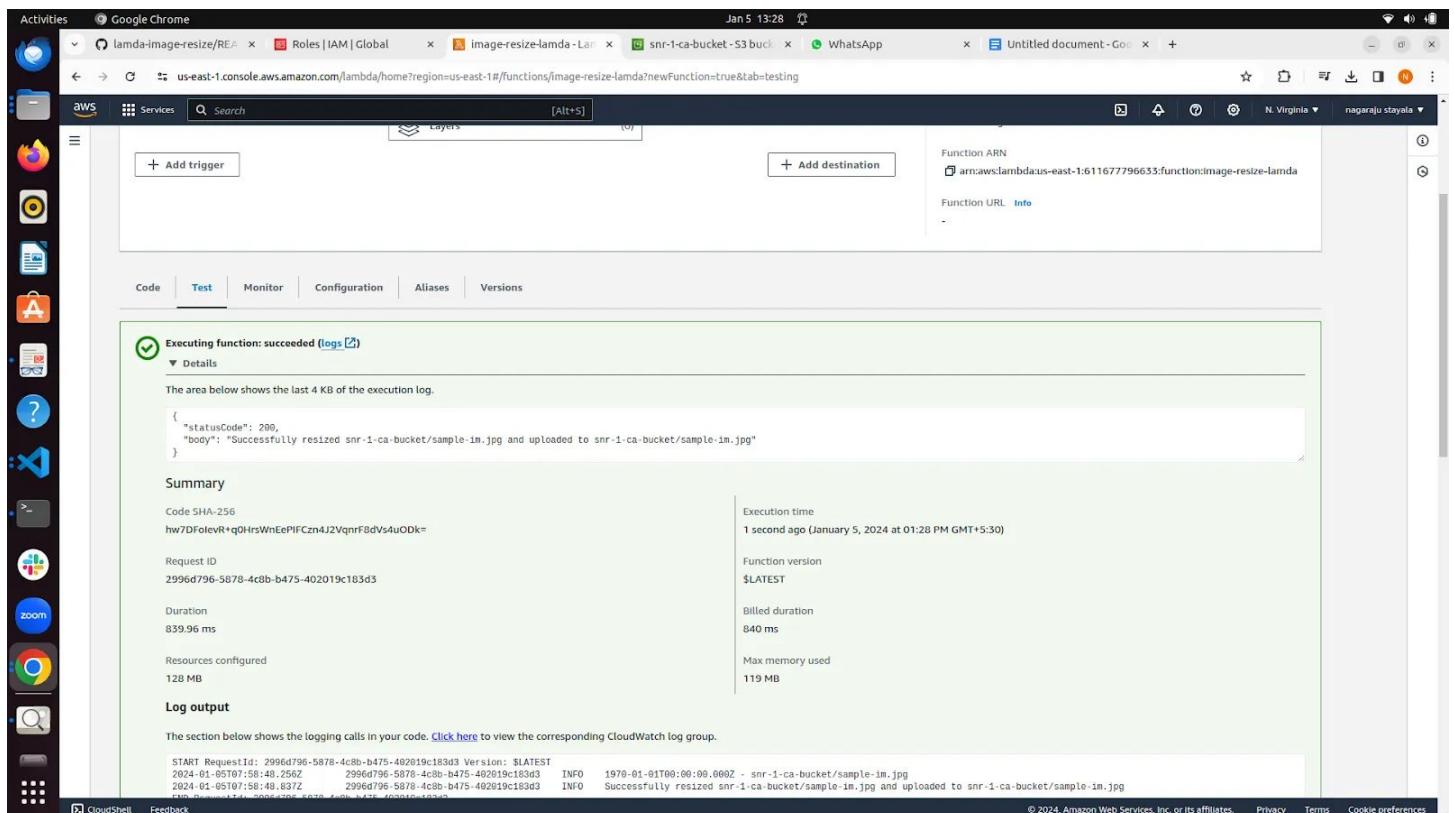
- Save it
 - Now go to the test
 - Select template
 - Select s3 on the search bar
 - This code is modified with your existing s3 bucket and the key name is your image name in the s3 bucket



- Now go to test this code



- The code test is successfully run



- Now go cloud watch monitoring

The screenshot shows the AWS CloudWatch Log Events console. The left sidebar has a 'Logs' section with 'Log groups' selected, showing 'snr-1-ca' as the current group. The main area displays log events for a specific entry ID. The log entries show the function starting, reading from an S3 bucket, processing the image, and successfully uploading it back to the same bucket. The log entries are timestamped and include detailed information about the request ID, AWS Lambda runtime environment, and the image file being processed.

- Now we can see s3 buckets with the actual size of the image.

The screenshot shows the AWS S3 Buckets console. The left sidebar has a 'Buckets' section with 'snr-1-ca-bucket' selected. The main area shows the bucket's properties, including its name and public access status. Below this, the 'Objects' tab is selected, displaying a list of objects. There is one object named 'sample-im.jpg' which is a jpg file uploaded on January 5, 2024, at 13:58:20 UTC+05:30, with a size of 352.2 KB and a standard storage class.

- Now you can see the re-sized image now