Vivekanand Education Society's Institute of Technology, Chembur, Mumbai,
Department of Technology,
Year: 2024-2025(ODD SEM)

Advance DevOps Practical Examination

Name:Tarunkumar Sharma **Div: D15B**
Roll No: 56 **Date of exam: 24/10/2024**

Case Study:16

Event-Driven Image Processing

Problem Statement: "Create an AWS Lambda function that automatically resizes an image when it is uploaded to an S3 bucket. The resized image should be stored in a different S3 bucket."

Theory:

Amazon S3 (Simple Storage Service)

Amazon S3 is a highly durable and scalable object storage service designed to store and retrieve any amount of data from anywhere on the web. It organizes data into **buckets** (similar to folders) and **objects** (individual data items, typically files). Each object is identified by a unique key within the bucket.

Key features:

- **Durability and Availability**: S3 provides 99.999999999% (11 9's) durability by redundantly storing objects across multiple devices in multiple Availability Zones.
- **Scalability**: Automatically scales to handle growing data volumes without upfront provisioning or capacity management.
- **Security**: Supports encryption (both in transit and at rest), along with comprehensive access control policies via IAM.

- **Storage Classes**: Offers a variety of storage classes (Standard, Intelligent-Tiering, Glacier, etc.) for different use cases, balancing cost and retrieval time.
- **Lifecycle Management**: Allows automatic transition of objects between different storage classes or deletion based on defined rules.

Common use cases for S3 include:

- Data backups and disaster recovery.
- Static website hosting.
- Big data analytics.
- Storing media files and serving them to users.

AWS Lambda

AWS Lambda is a **serverless** compute service that runs code in response to defined triggers, such as an HTTP request or an upload to an S3 bucket. Lambda takes care of provisioning and managing the underlying infrastructure, allowing developers to focus on writing and deploying code.

Key concepts:

- **Event-driven architecture**: Lambda responds to triggers from various AWS services like S3, DynamoDB, or CloudWatch Events. It can also be invoked through APIs using Amazon API Gateway.
- **No infrastructure management**: Lambda automatically scales to accommodate the number of incoming requests, handling thousands of concurrent executions.
- **Pay-per-use**: You only pay for the compute time (measured in milliseconds) when the function runs, along with any memory allocated to the function.
- **Runtime flexibility**: Supports multiple languages (Node.js, Python, Ruby, Java, Go, etc.) and custom runtimes.

Use cases:

- Data processing (e.g., resizing images, log analysis).
- Real-time file processing (e.g., uploading data to S3 and triggering Lambda to process it).
- Backends for mobile, web, and IoT applications.

IAM (Identity and Access Management)

IAM enables secure management of access to AWS resources by creating policies that define **who** can access certain resources and **what** actions they can perform.

Key components:

- **Users**: Individual identities with specific credentials.
- **Groups**: A collection of users, allowing common policies to be applied to all members.
- **Roles**: Temporary credentials used to delegate access to AWS services (e.g., EC2 accessing S3).
- **Policies**: Documents that define the permissions (allow or deny) granted to users, groups, or roles. Policies are written in JSON format.

IAM provides **fine-grained access control**, enforcing the principle of least privilege, which ensures that users and applications have the minimum necessary permissions to perform their tasks.

IAM also integrates with:

- **MFA (Multi-Factor Authentication)** for stronger security.
- **Identity Federation**, allowing external identities (like Google or corporate directories) to access AWS.

Additional Concepts:

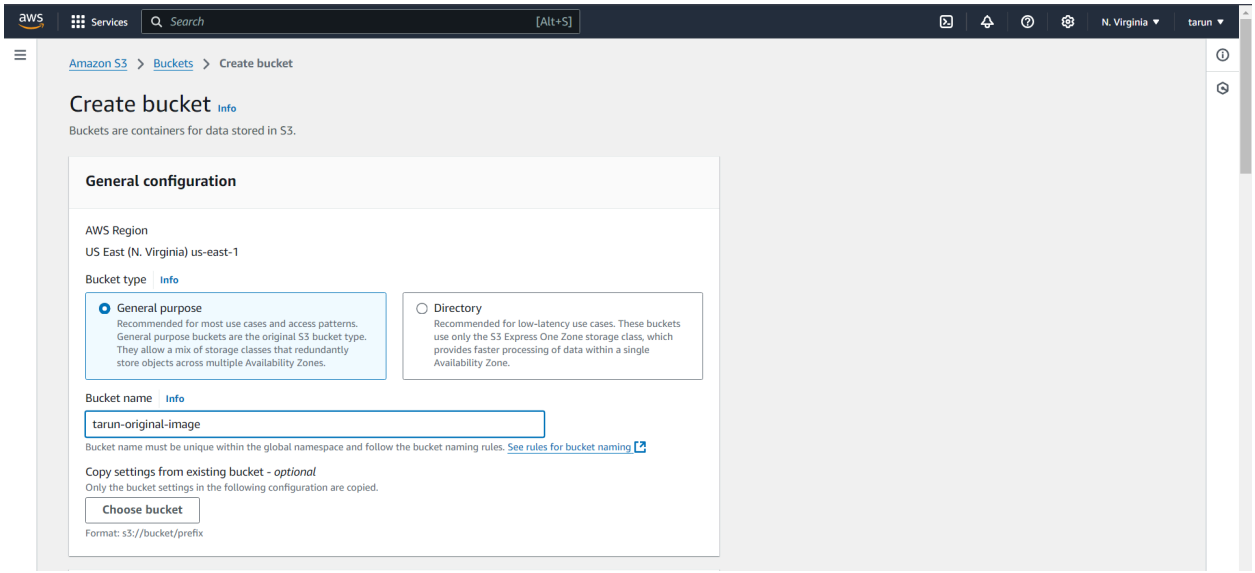
- **CloudWatch**: Monitors AWS resources and applications, providing logging, alarms, and metrics. It can trigger Lambda functions based on these alarms or thresholds.
- **SNS (Simple Notification Service)**: Allows sending notifications based on events from AWS services, such as Lambda function invocations or S3 bucket activity.

Step-by-Step Implementation:

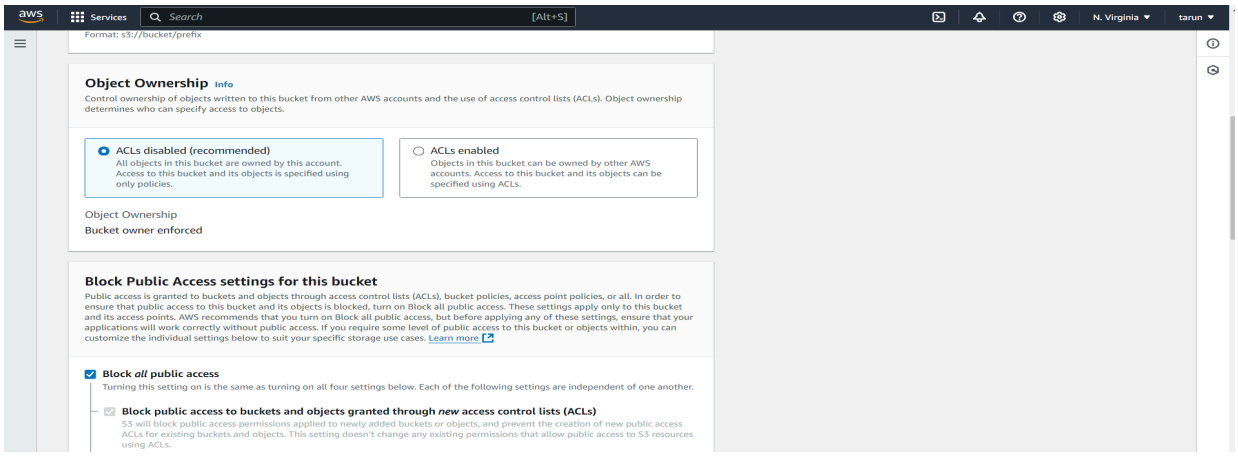
1.) Creating S3 Buckets:-

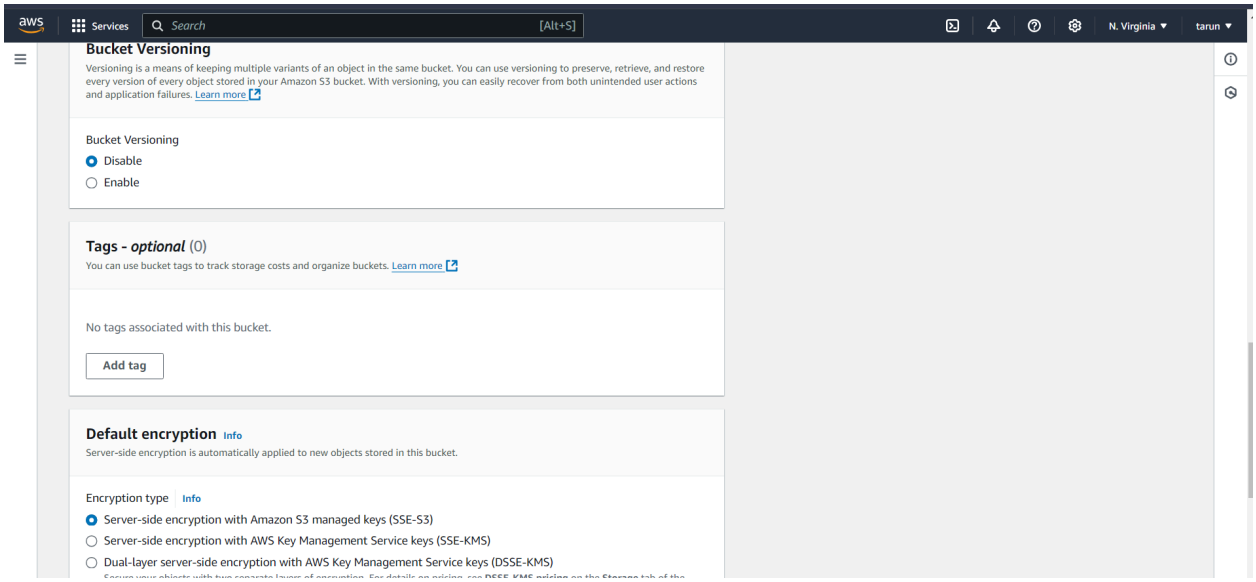
1.) Sign into AWS Management Console
2.) Search S3 Service or access it from the Console itself
3.) Click on Create Bucket
4.) Give a name to the bucket. This is the bucket which will contain the original image

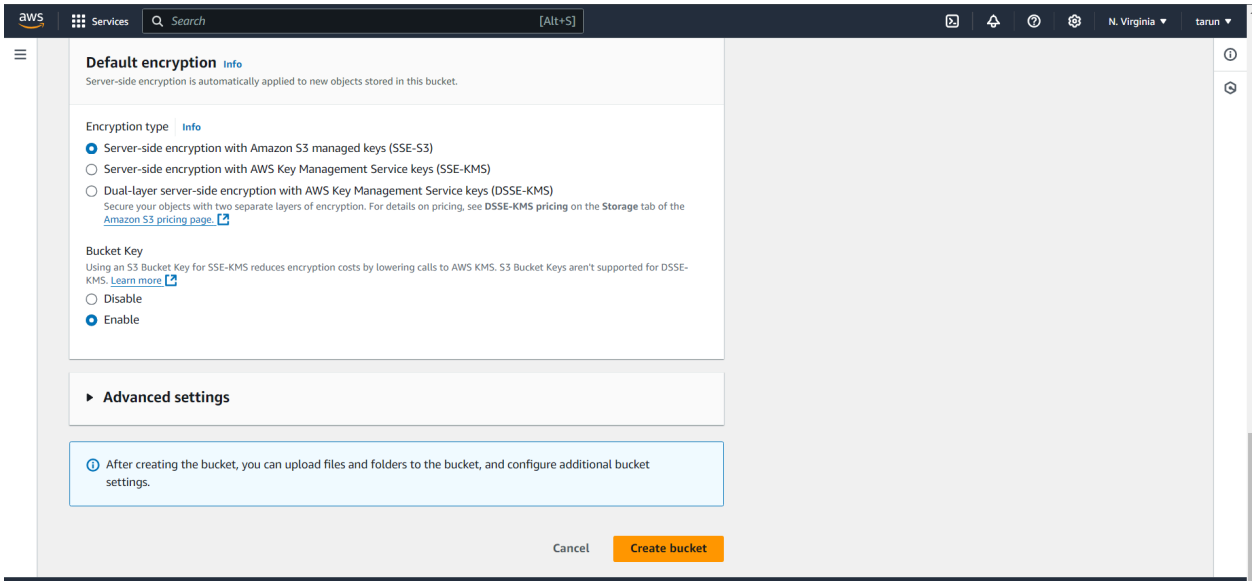
which has not been resized by our Lambda function.Here I have given the name **tarun-original-image**.



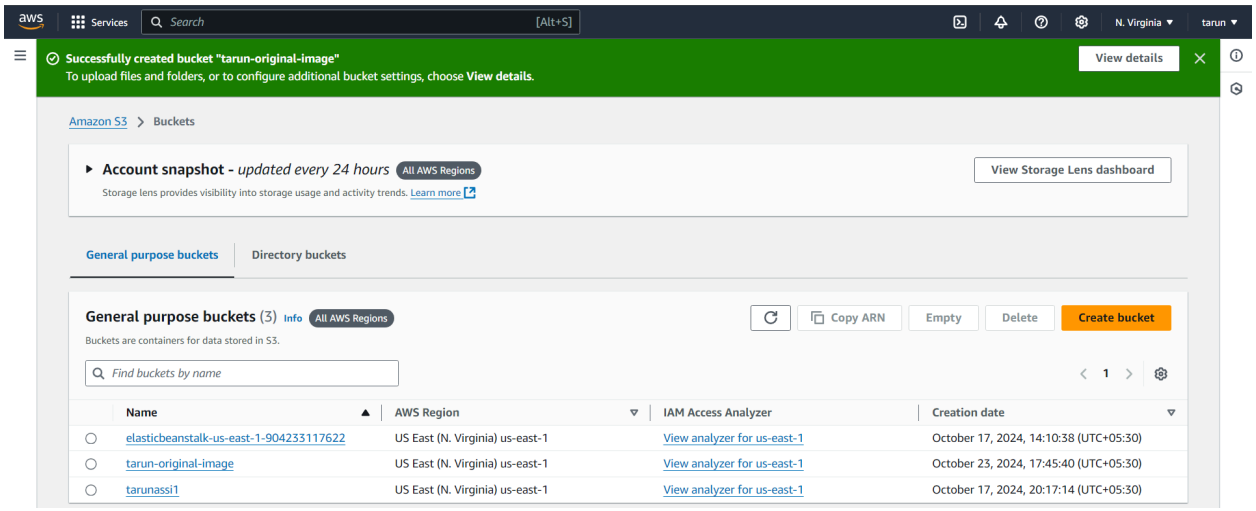
5.) Keep the default Configurations

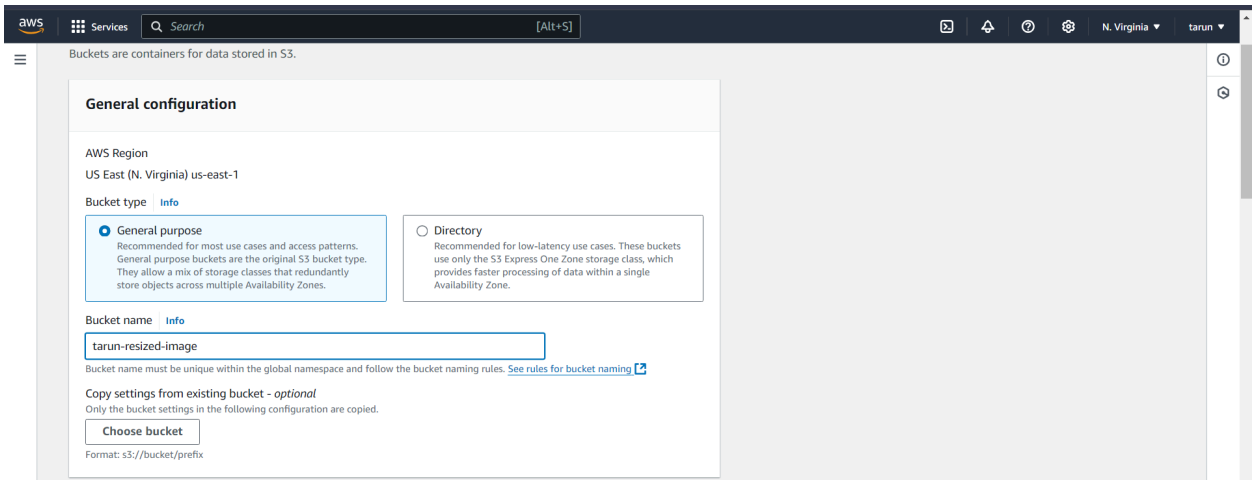


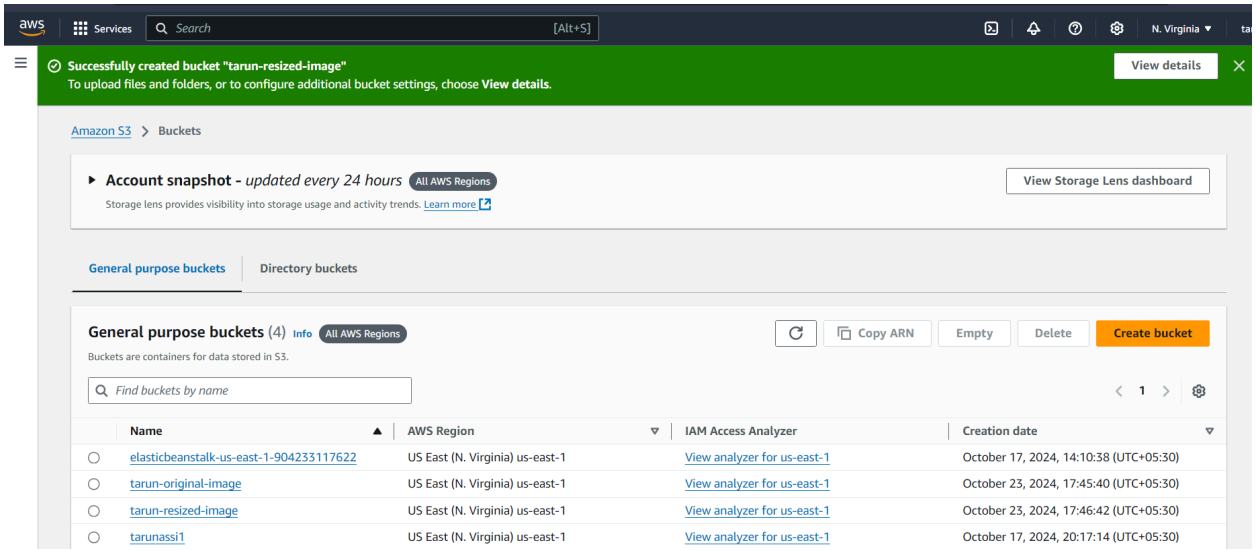




6.) Click on ‘Create Bucket’ to finalize and create the original image bucket.
7.) Repeat the same procedure to create a bucket to store the resized image.Here I have given the bucket name **tarun-resized-image**.





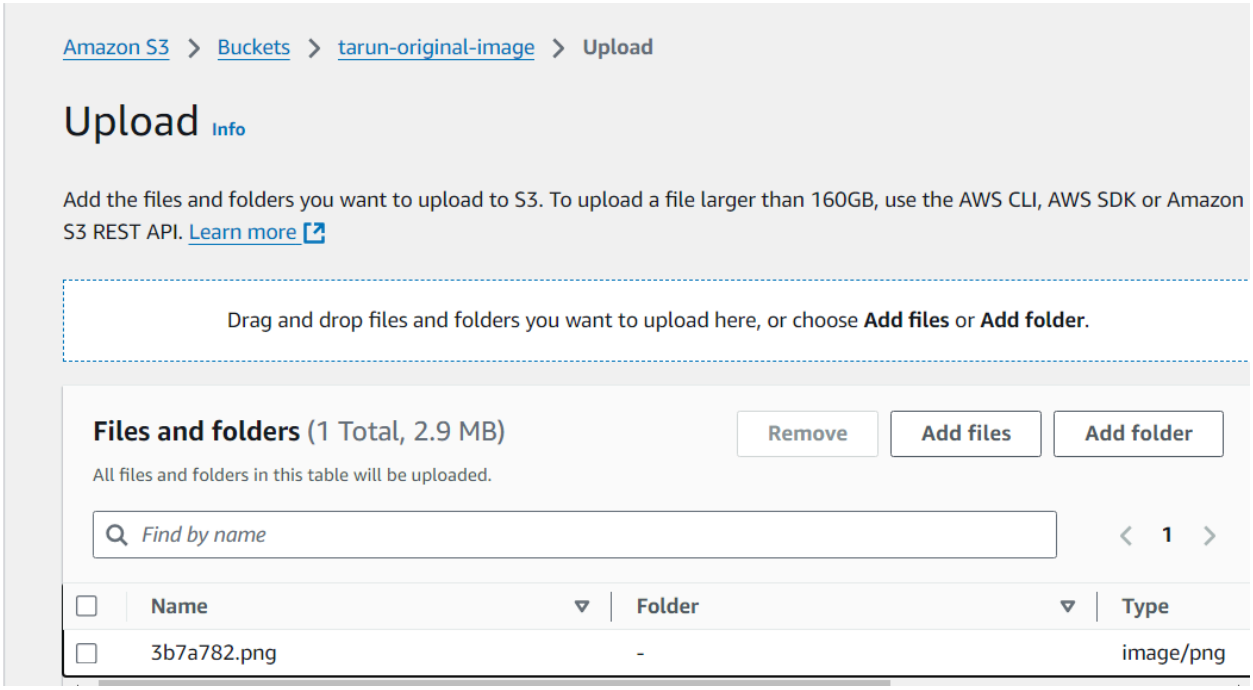


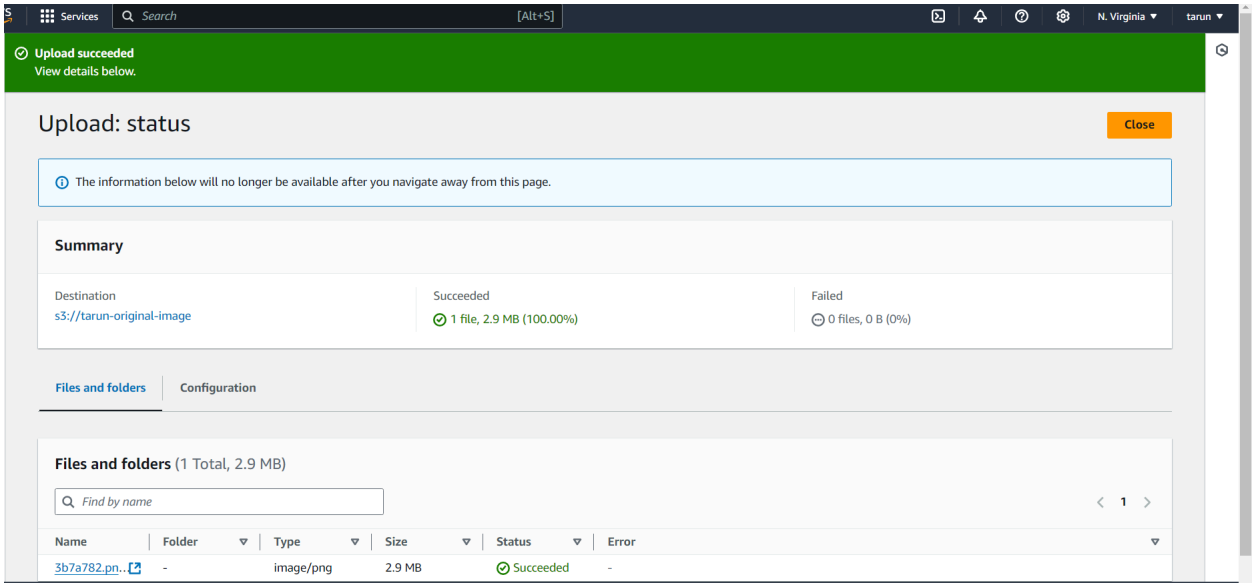
8.) Now both the buckets have been created.

2.)Upload Image to original image bucket:-

1.) Select your original image bucket.

2.) Select the Upload option and Drag and Drop the image which you want to use for the test case of our code.Here I have uploaded an image named axolotl.jpg.

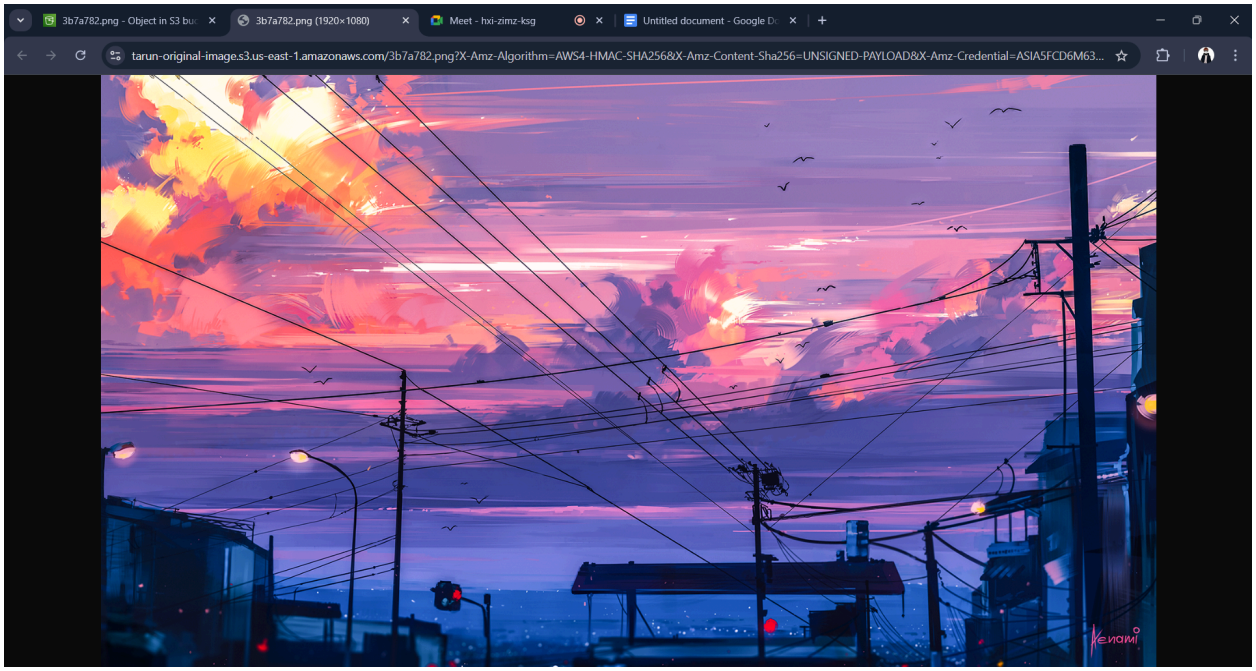




3.) The image has been uploaded to the original image bucket.

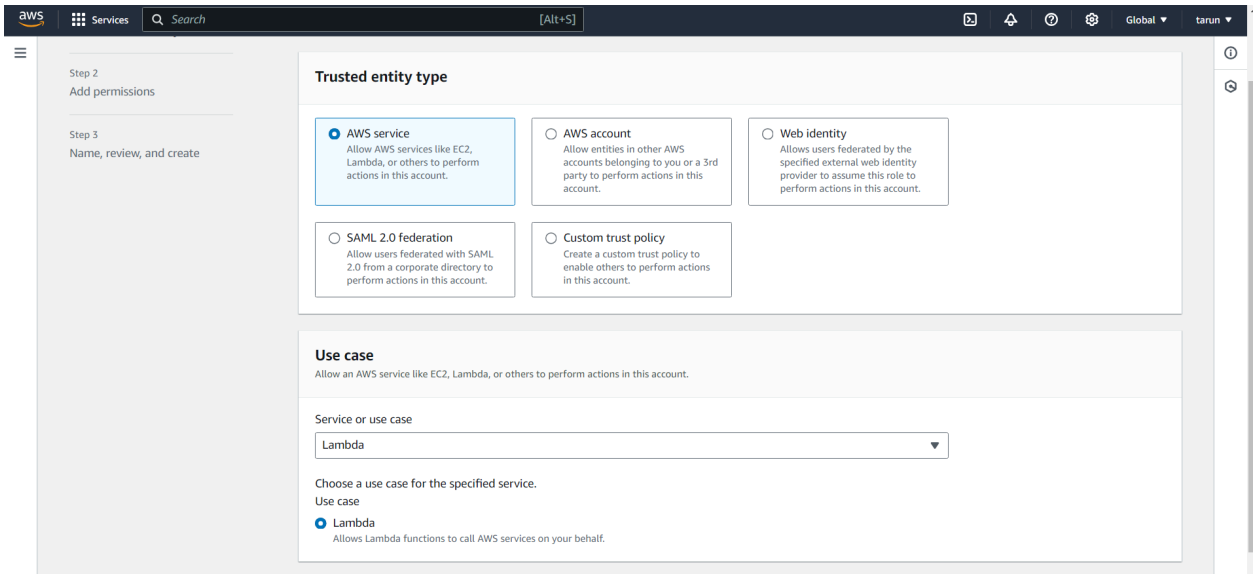
4.) Select the image and click Open option to view the Image in Browser.

5.) The dimensions of the image in the original image bucket(1920 X 1080 Here).

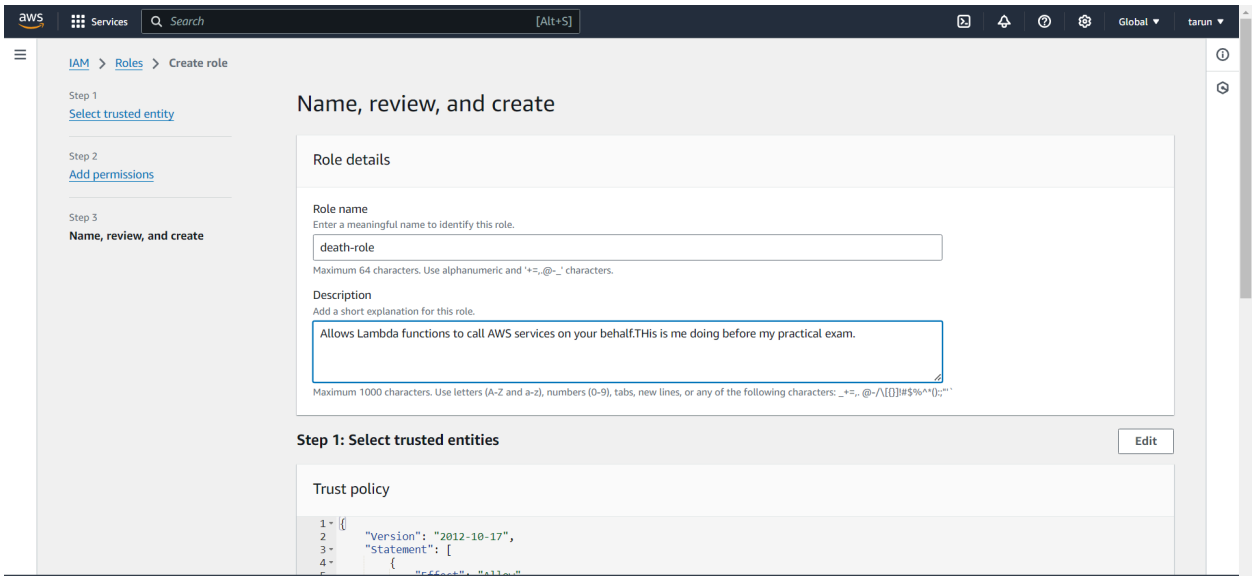


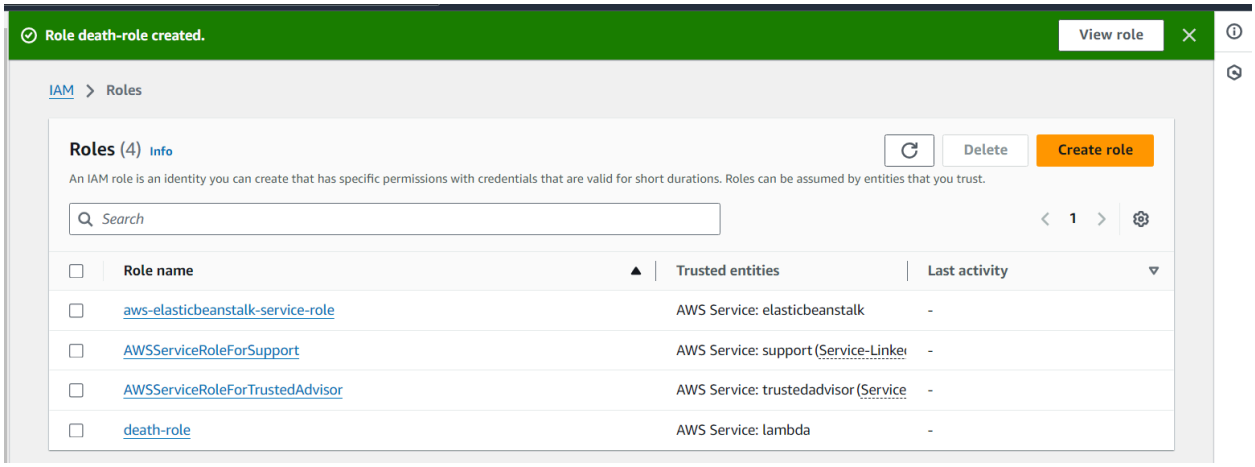
3.)Creating IAM Role for Lambda function:-

1.) Go to the IAM Console from the AWS Management Console.
2.) Select ‘Create a new Role’ and select **Use Case as Lambda**.



3.) Give a name to Your role and create a role without attaching any policies for now.





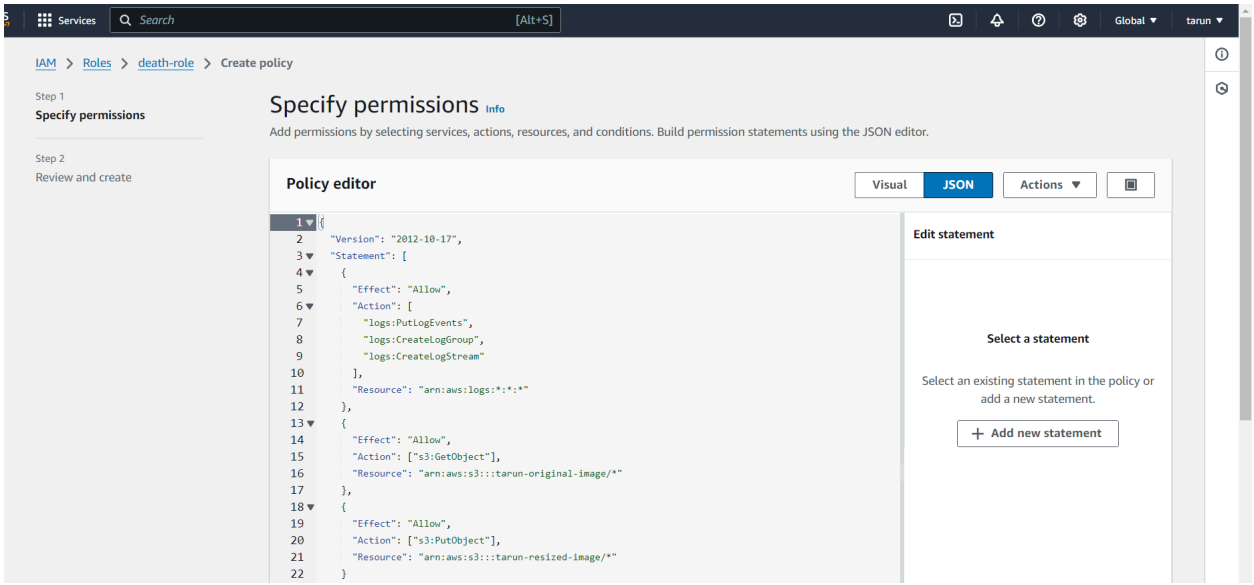
4.) Select the role created . Here I had created the role **'Lambda Role'.**
5.) Scroll Down to view Permissions.

6.) Click on Add permissions and Select **'Create a Inline Policy'** Option. On the create policy window select **JSON** Option and replace the default JSON with this policy:-

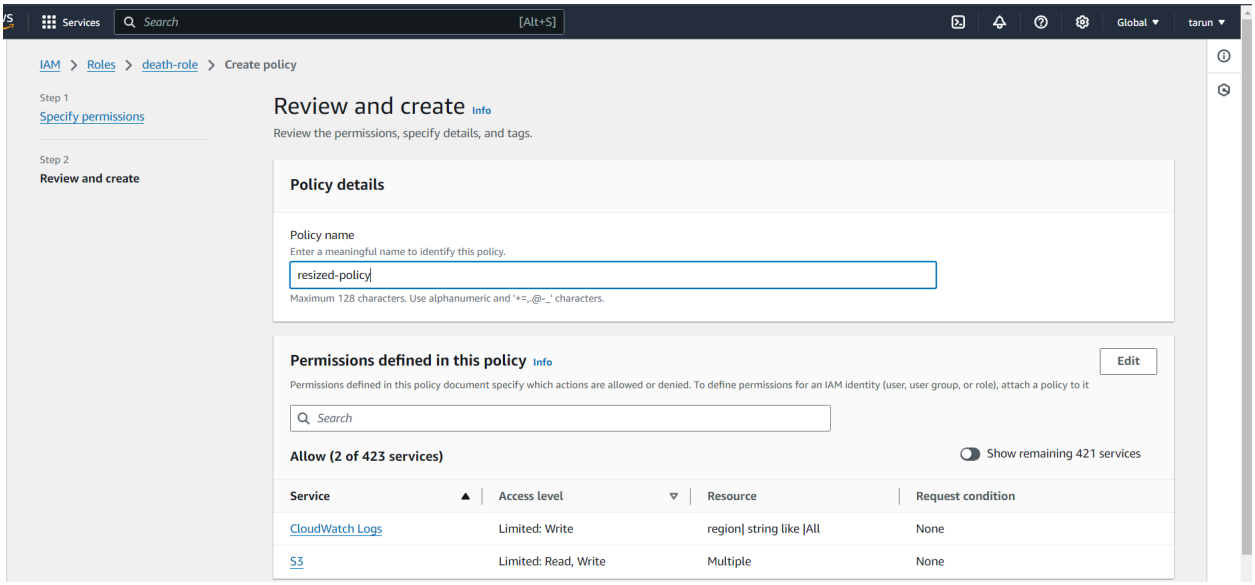
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
```

```
        "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/*"
    }
  ]
}
```

Here BUCKET_NAME = souce bucket(tarun-original-image)
DEST_BUCKET = destination bucket(tarun-resized-image)



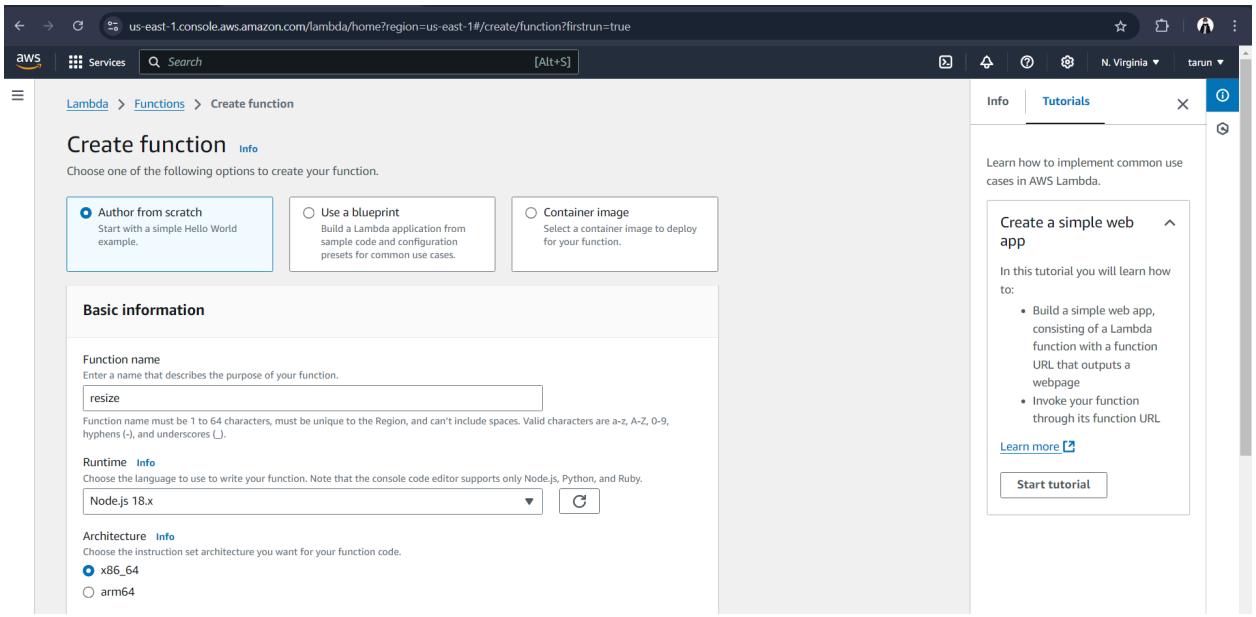
7.) Give the policy a name and click on create policy.After doing this the policy will automatically be attached to your IAM Role.Here I had given the policy name casestudy and it had automatically attached to LambdaRole on creation.



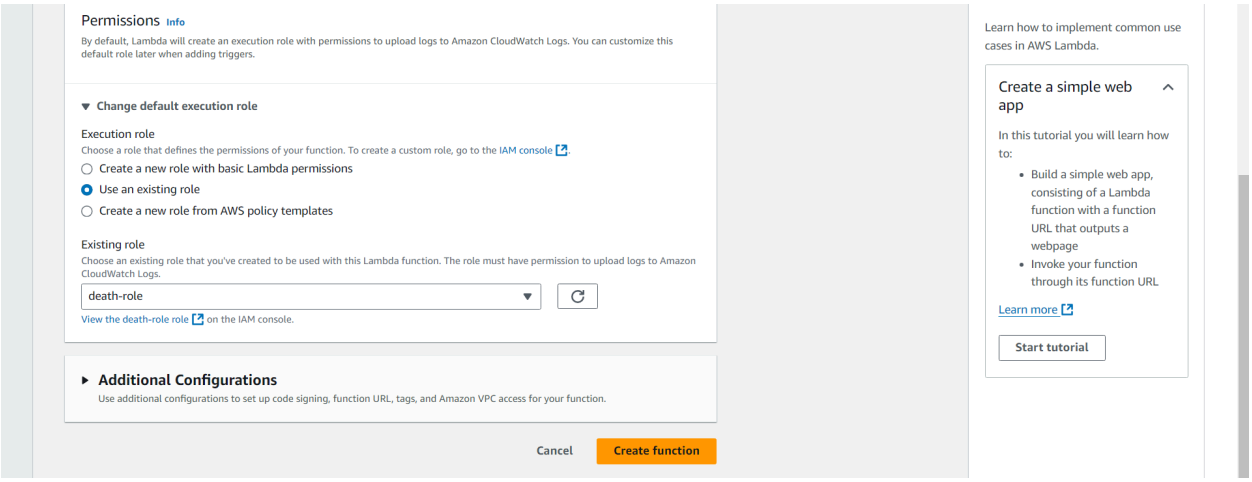
4.)Creating a Lambda Function for resizing:-

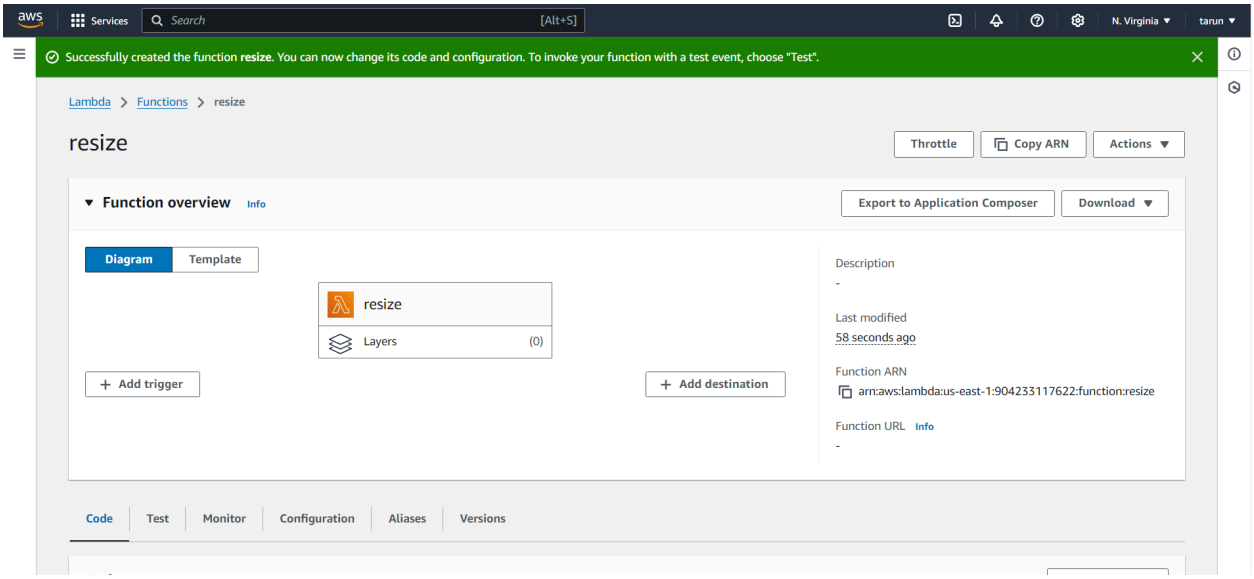
1.) Select **'Create Function'** Option.

2.) Give a name to your Function and select a Runtime Environment. Here I have given the function name 'image-resizer' and selected the Runtime Environment as Node.js 18.x.

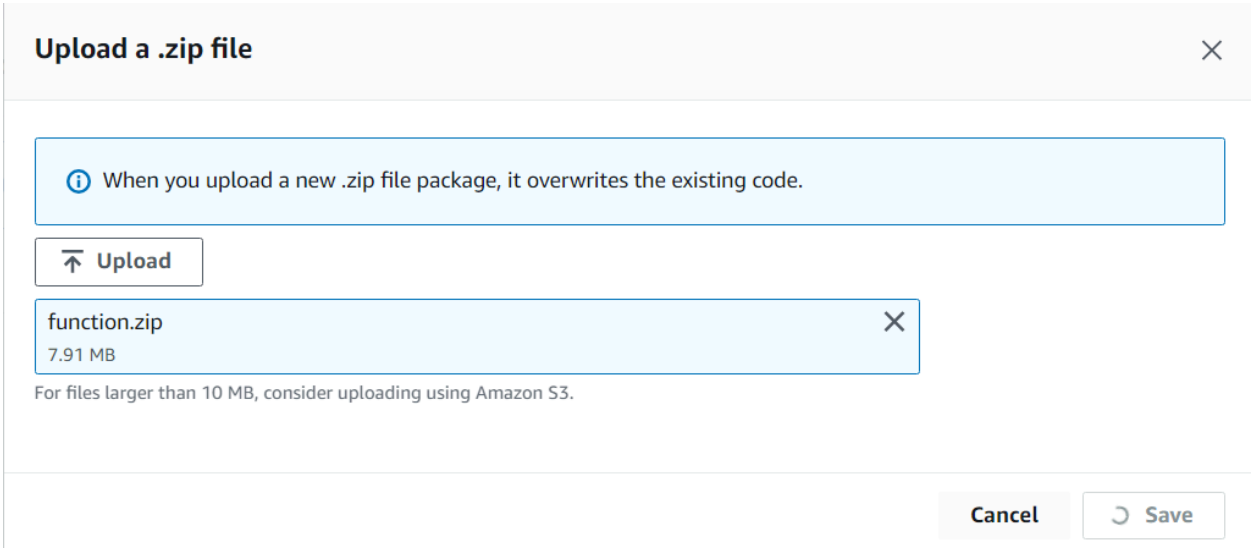


3.) Now Click on **'Change Default Execution Role'** in Basic Information below Permissions and Select **'Use an existing Role'** Option and select the role created in Step 3.

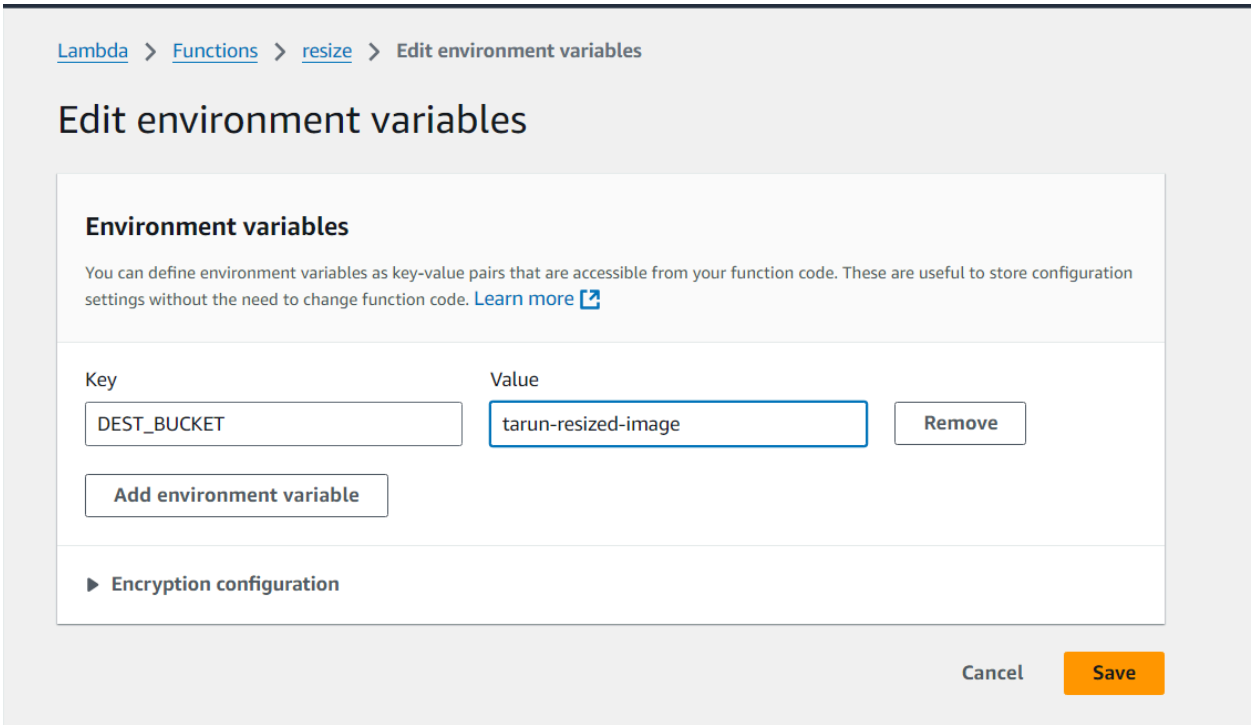




4.) Now after the function has been created select **upload from** Option in the Code Section of the created Lambda Function. Here image-resizer. Select **.zip file** option.
To download the Zip file follow this link:-
https://github.com/nikitadev-yt/image-resizer-lambda
And download function.zip from the above repository.It has all the dependencies included to function properly.



5.) Now go to the Configurations Section of the function and select the **'Environment variables'** option and Edit . Add the variable as shown



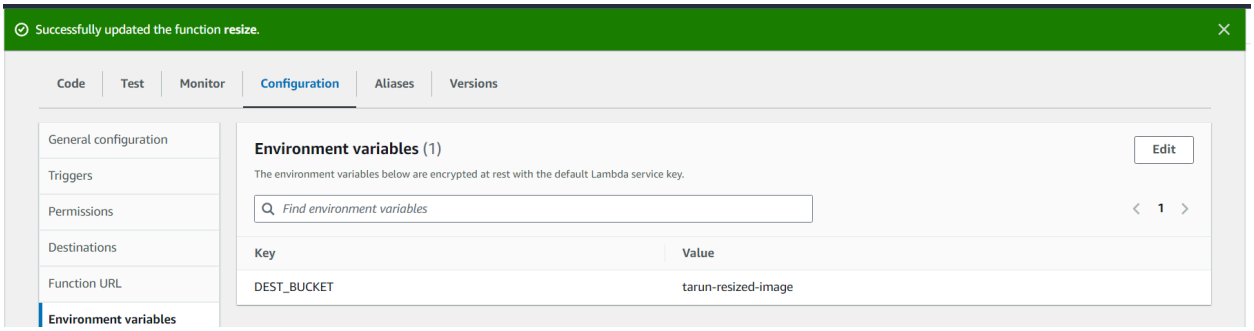
Instead of tarun-resized-image put your bucket name where you want to store the resized image.

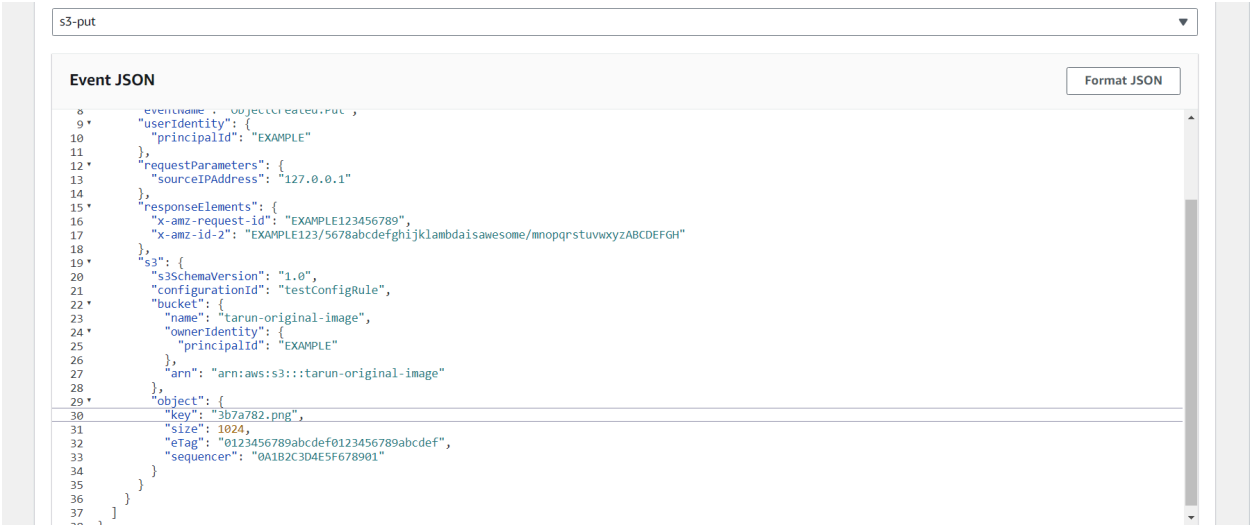
5.)Testing the Lambda function:-

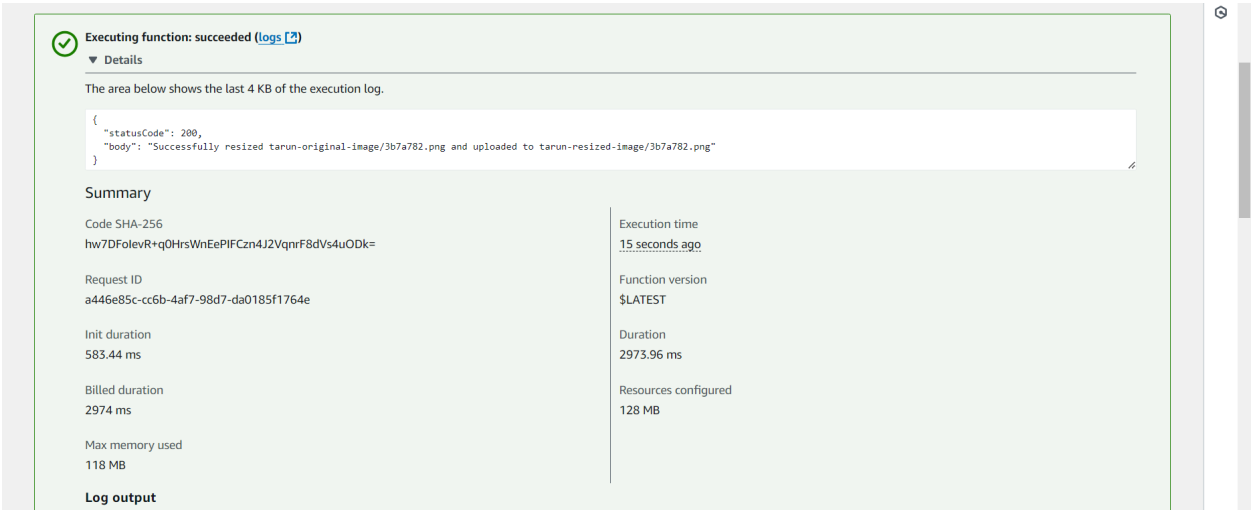
1.) Go to the Test section of the Lambda function.

2.) Keep the rest of the configurations default and just change the template to **S3-Put .**

3.) In the Event JSON of the S3-Put Template change example-bucket (**near name and arn in S3 Section of the JSON file**)to your original image bucket and key(**in object section**) to the image which you had uploaded in Step 2 for testing. Here I have changed the example - bucket to karan-original-bucket and key to axolotl.jpg.

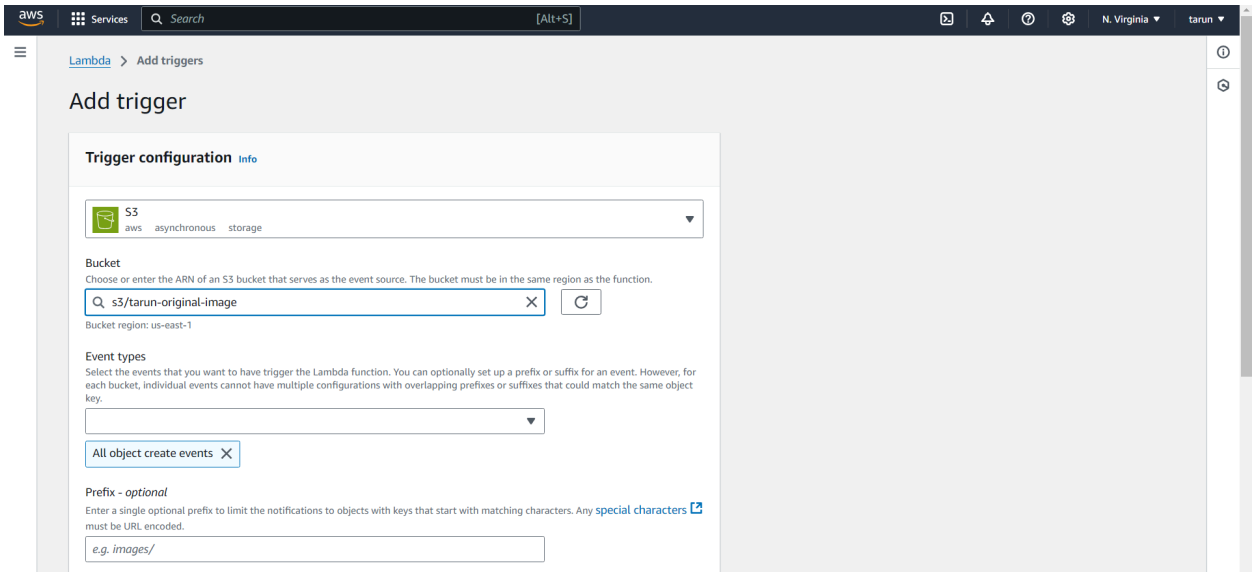


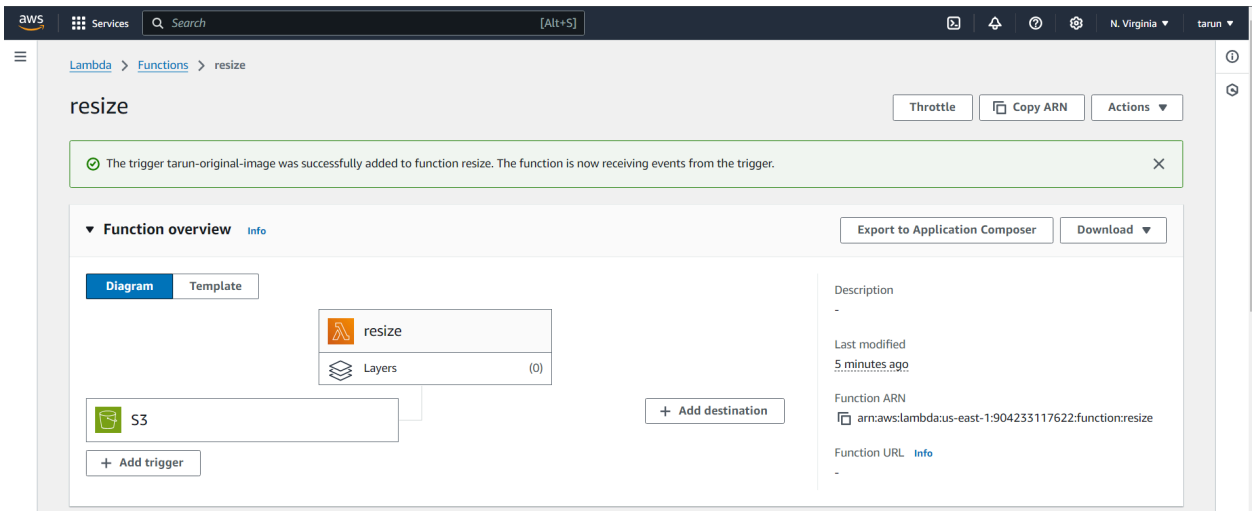


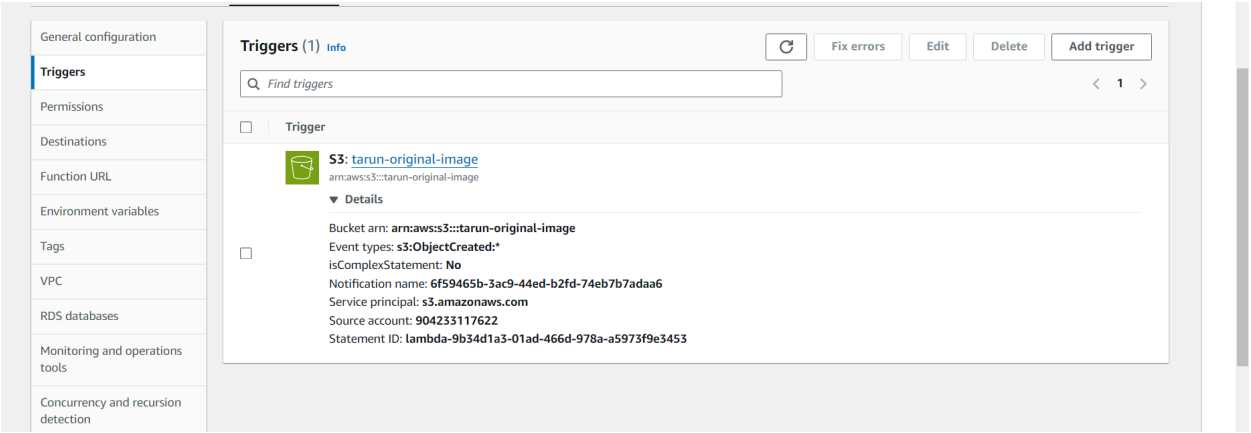


6.)Add Lambda Trigger:-

1.)Select Add Trigger on the Lambda Function Page.
2.)Select S3 in the Configuration and select your original image bucket(source bucket) so that whenever an Image is uploaded on that bucket the function is triggered and the Image is resized and stored on the resized image bucket.
3.)Keep Rest of the configurations Default.

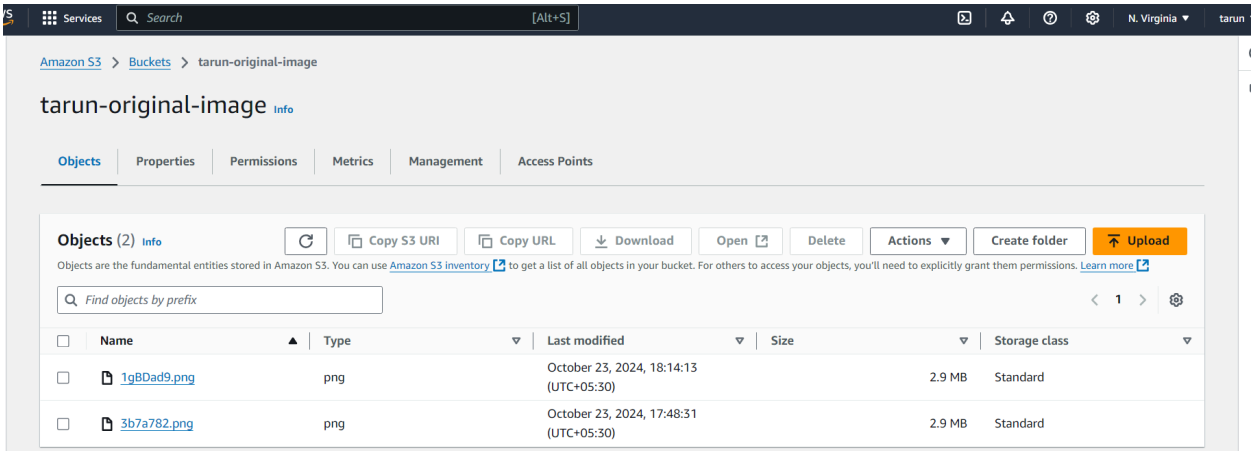




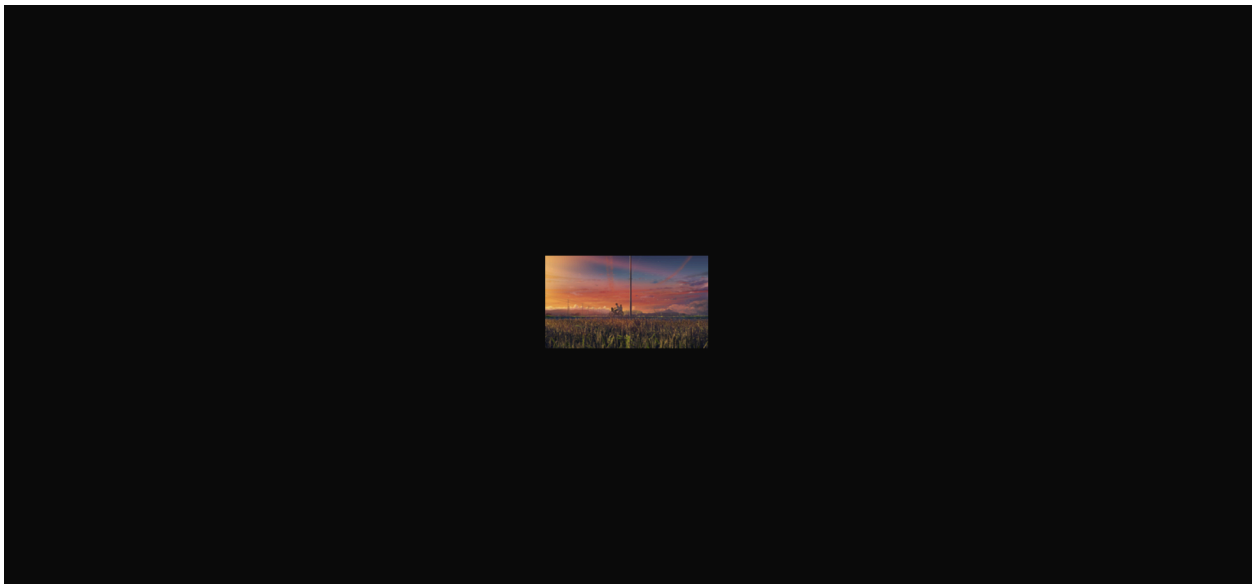


7.)Final Test:-

1.)Upload an image again in the original image bucket and open it in the browser and see its dimensions. Here I have uploaded an 1gBDad9.png for this .



2.)Go to your resized image bucket and open the image in the browser to see changed Dimensions



3.)Experiment successfully Done.

Conclusion:

By leveraging AWS Lambda and S3, an efficient and scalable solution for automatic image resizing is implemented. The Lambda function, developed in Node.js, ensures that each image uploaded to the source S3 bucket is resized and stored in the destination bucket without manual intervention. This approach enhances the image processing workflow, providing a cost-effective and automated system that can handle dynamic workloads.