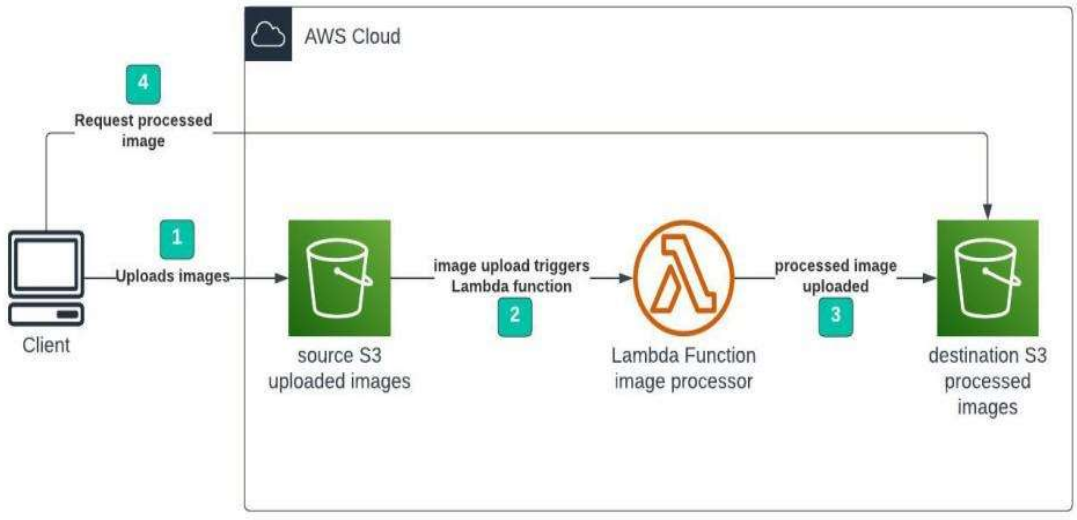
Serverless Image Processing

In this project we Create a serverless image processing application that automatically resizes and optimizes images uploaded to an Amazon S3 bucket with the help of lambda function.

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers. It automatically scales your application by running code in response to each trigger. This approach allows you to build various applications, from simple web applications to complex data processing systems. In this guide, we'll walk through the process of building an application with AWS Lambda, complete with an example.



Serverless Image Processor

Tools and Services:

AWS S3: To store input and output images.

AWS Lambda: To process the images.

AWS IAM: To manage permissions.

Steps for creating a serverless image processing site

step 1: Setting Up S3 Buckets.

Step 2: Setting Up IAM Role.

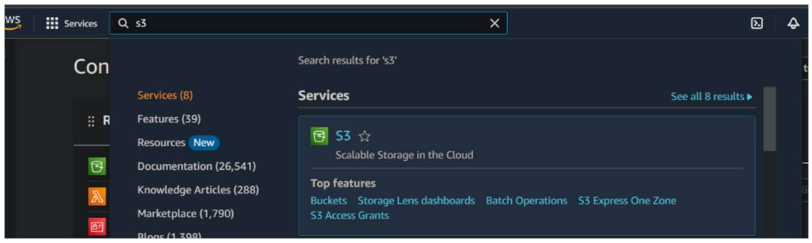
Step 3: Create role.

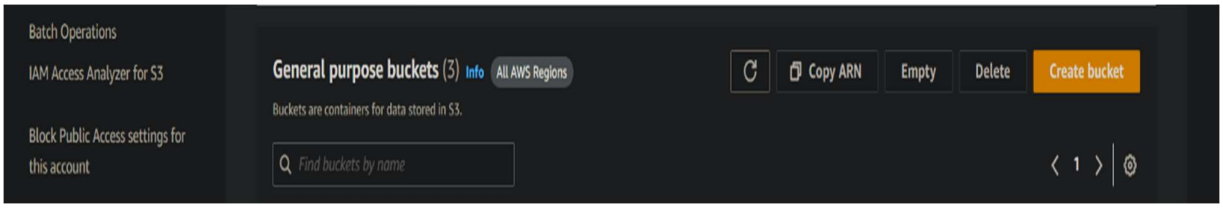
Step 4: Creating the Lambda Function.

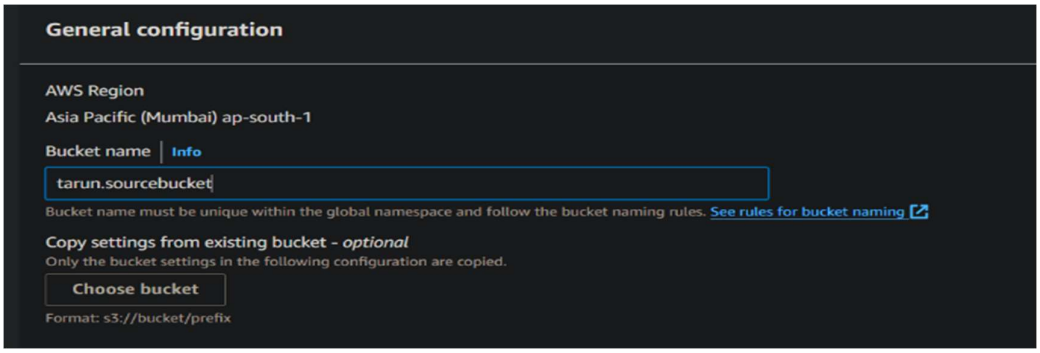
Step 5: Adding trigger in lambda function.

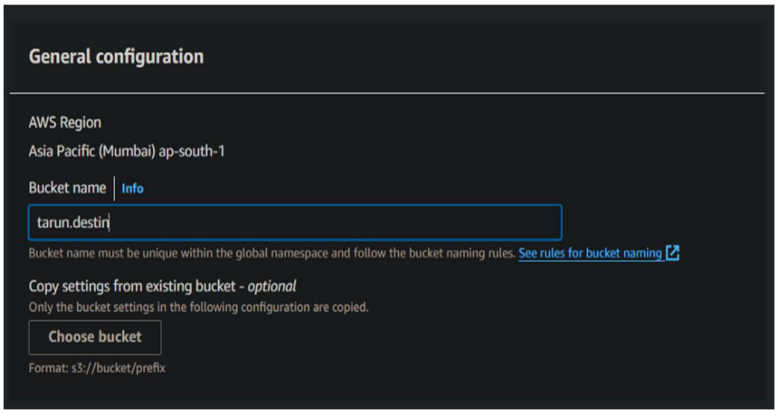
Step 1: Setting Up S3 Buckets

- Create Two S3 Buckets: One for uploading the original images (`input-bucket`) and another for storing the resized images (`output-bucket`) in my case input bucket - tarun.sourcebucket and output bucket - tarun.destin.
- Navigate to S3 and create 2 buckets.

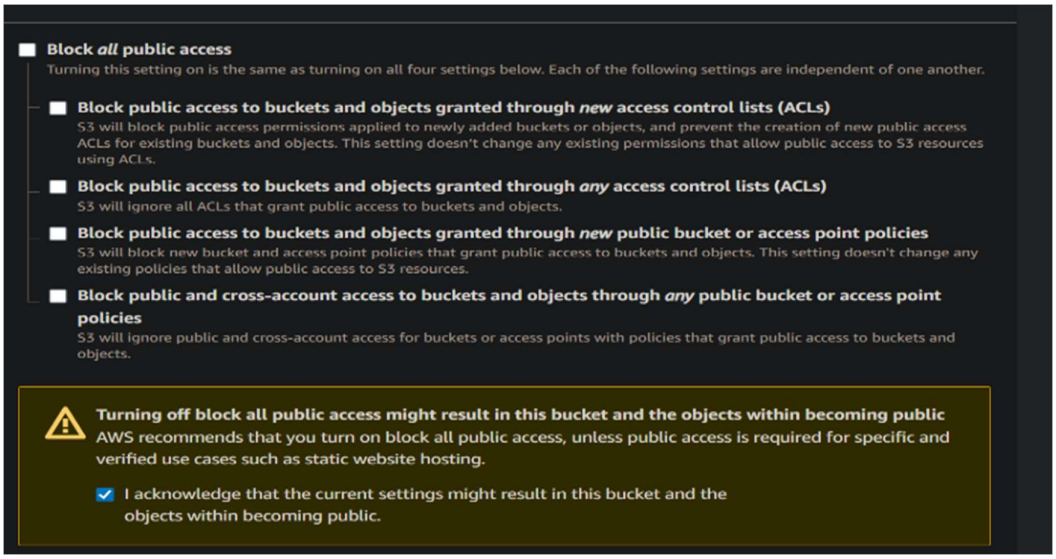






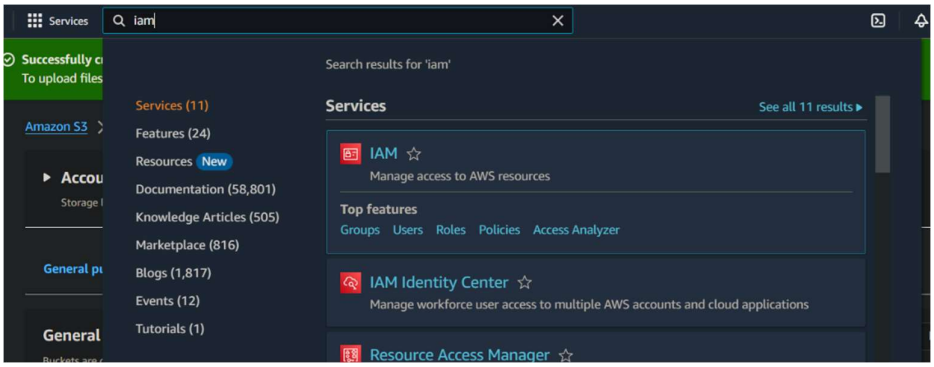


- Make sure to uncheck block all public access.

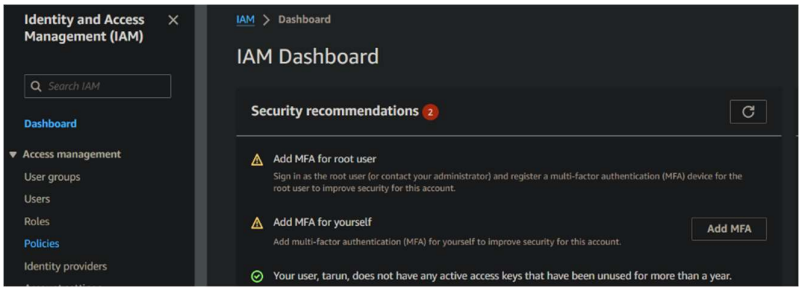


Step 2: Setting Up IAM Role

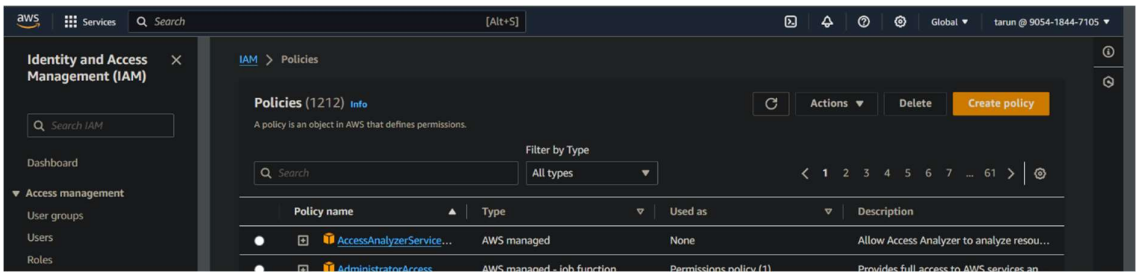
- Create a Custom IAM Policy.
- Go to the IAM dashboard



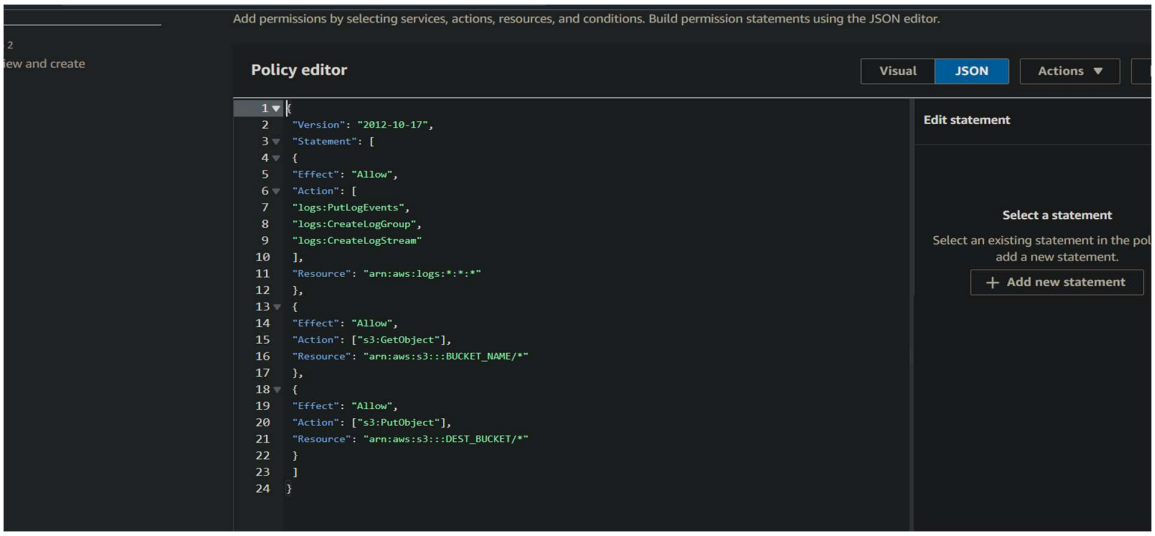
- Click to policies on the left corner.



- Click on create new policy

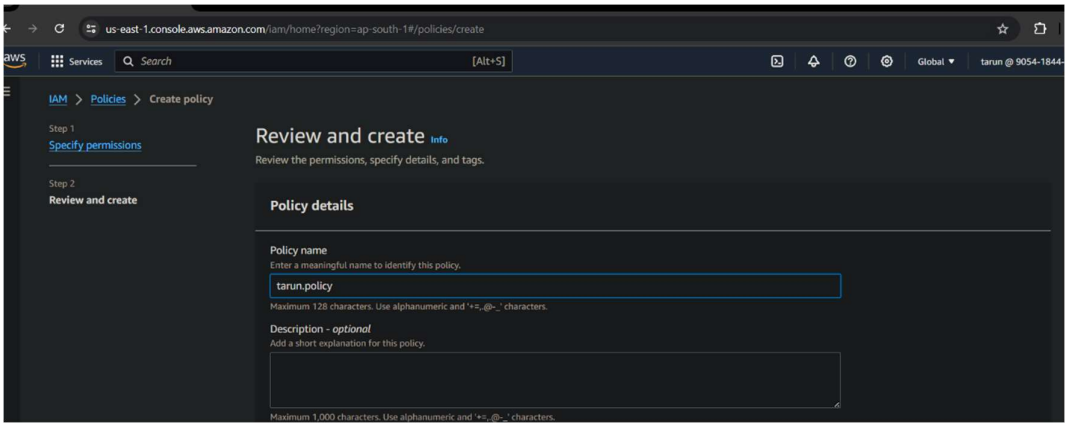


- Replace the policy with the given code and change the source and destination bucket ARN.



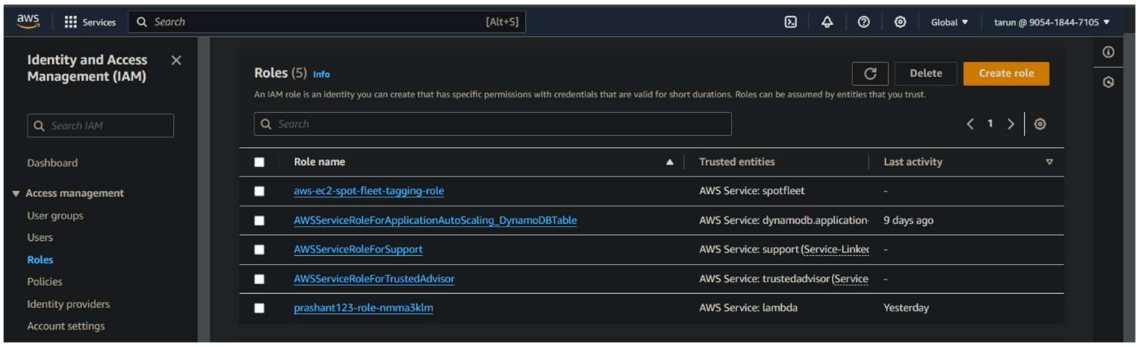

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::input-bucket-199/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::output-bucket-199/*"
    }
  ]
}
```

- In the review an create section give the name of your policy and then click on create policy.

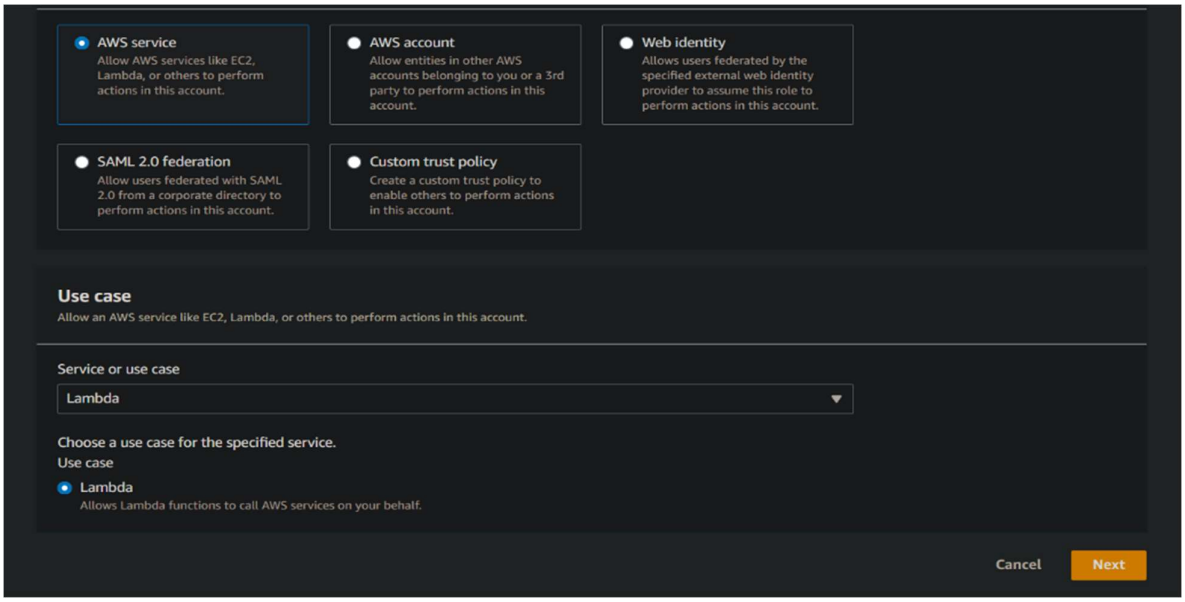


Step 3: Create role

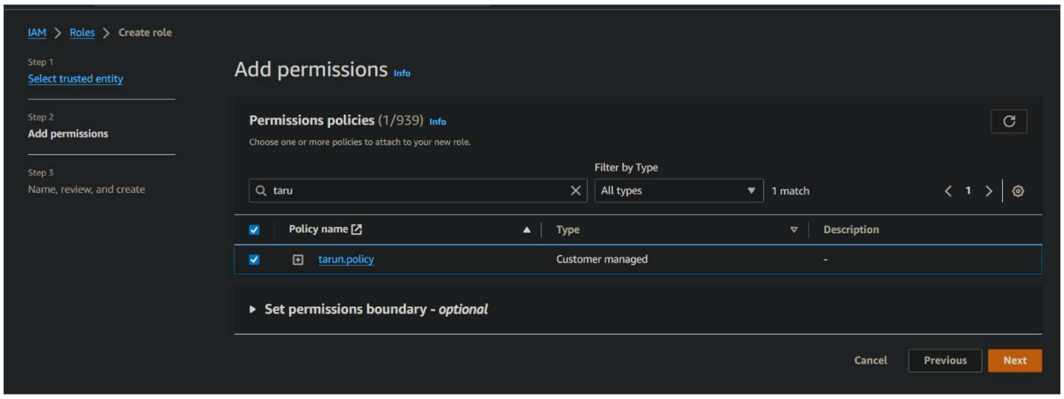
- Now go to the IAM dashboard and click on the create role.



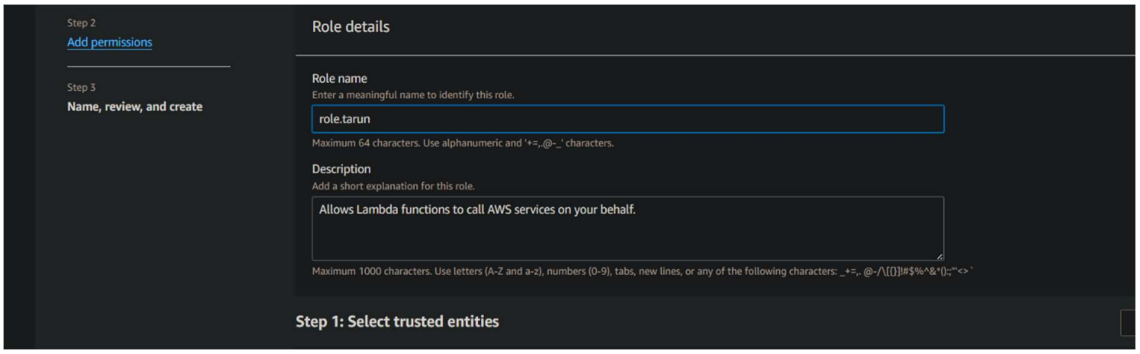
- Choose the AWS service in the upper box and
- In service as use case choose lambda as a service.
- Now click on the next button in the right-bottom corner.



- Find the policy you just created using the policy name, select it, and click next.

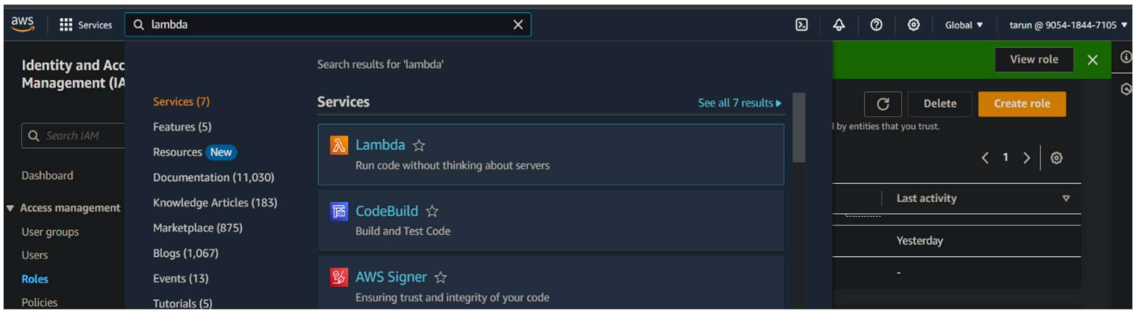


- In the Add Permissions section give the name our your role in my case it is “role.tarun”. and then click on create role button.



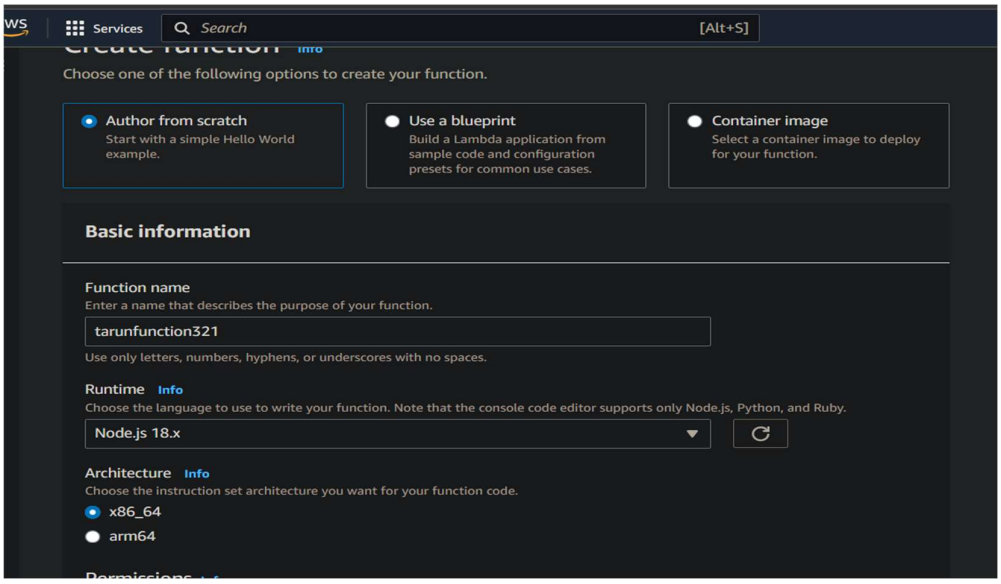
Step 4: Creating the Lambda Function

- Head over to the AWS Lambda dashboard by searching lambda on the services.

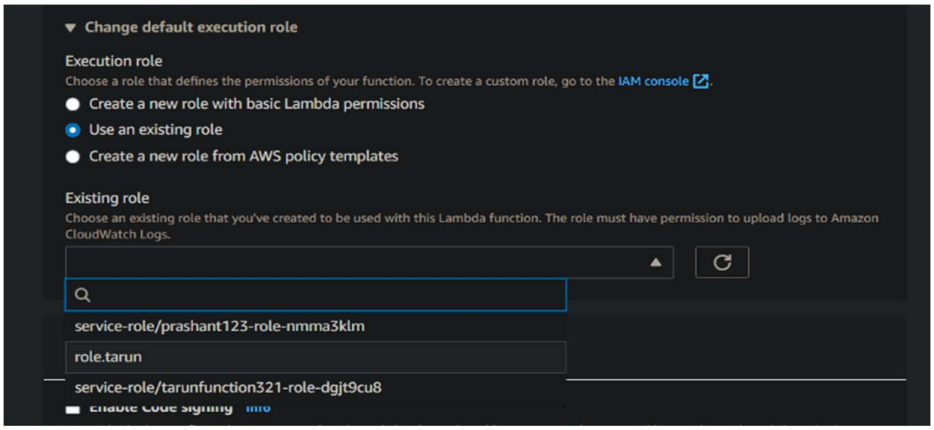


- Now click on the create function.

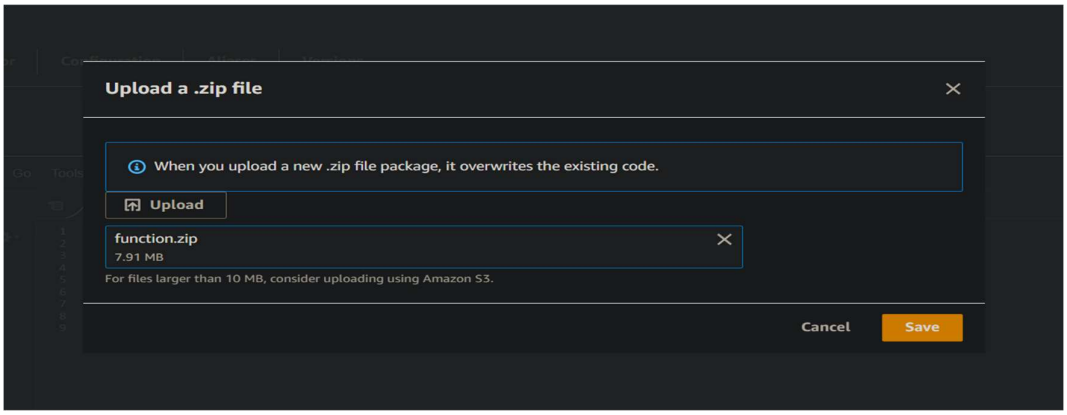
- Now choose author from scratch in the above section and the in the function name give your function a name in my case it is tarunfunction321.and in runtime select node.js 18x.



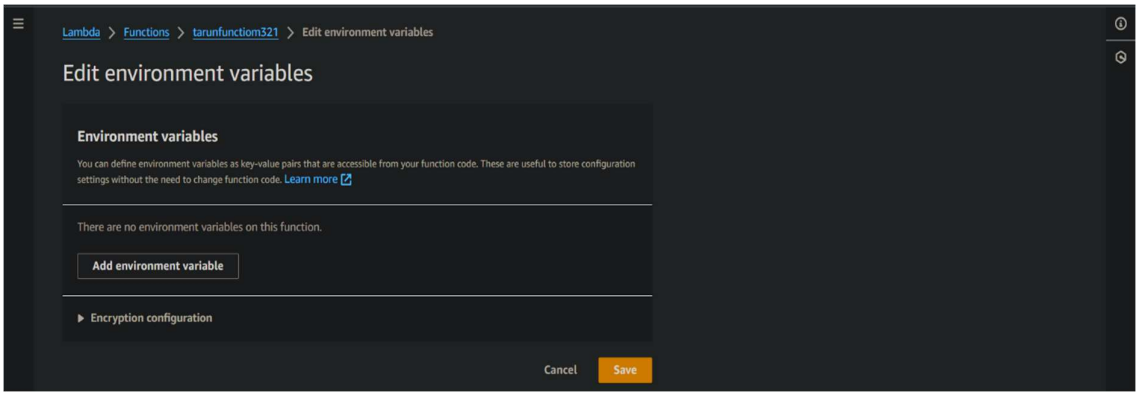
- Now click use an existing role in the execution role and search or select your role in the existing role section.



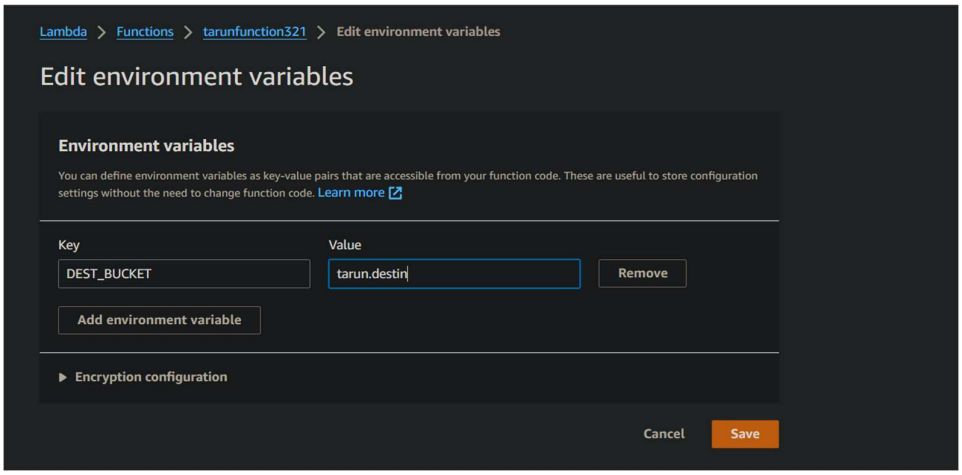
- Now upload your code in the upload section.



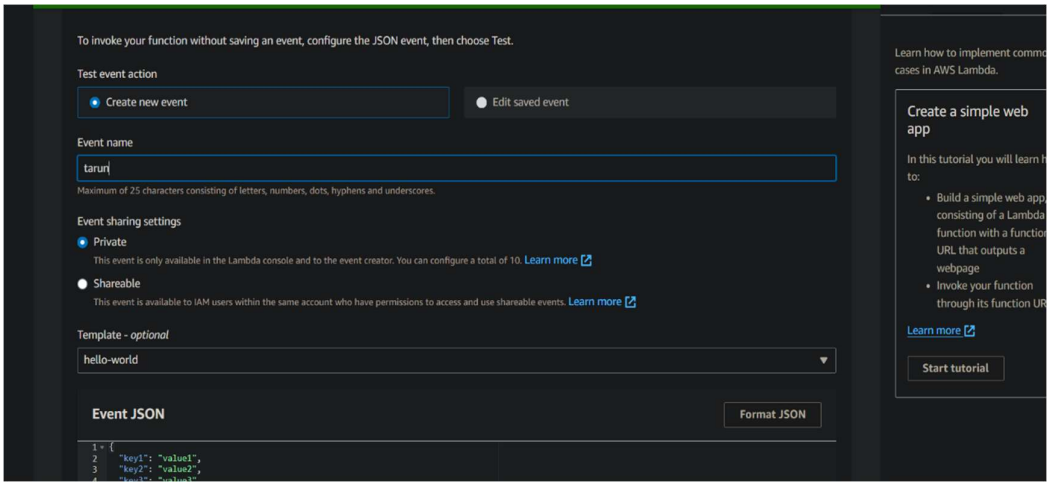
- Folder link is https://github.com/OneLightWebDev/image-resizer-lambda
- Navigate to the Edit environmental variables in the configuration section.



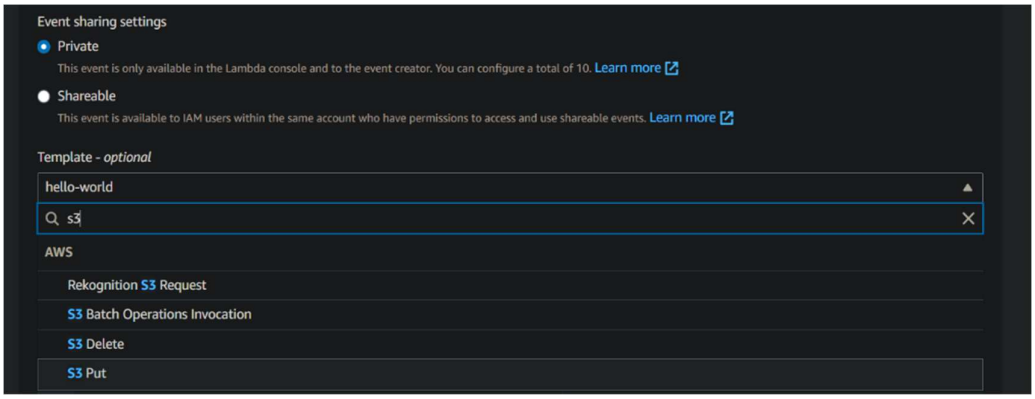
- **Now click on the add environmental variable.in key write DEST_BUCKET and in the values section write the name of your destination bucket and then click on save.**



- **Now we have to test our event , to test the event go to the test section click on create new event enter the name of your event.**



- **In the template section select S3 put request .**

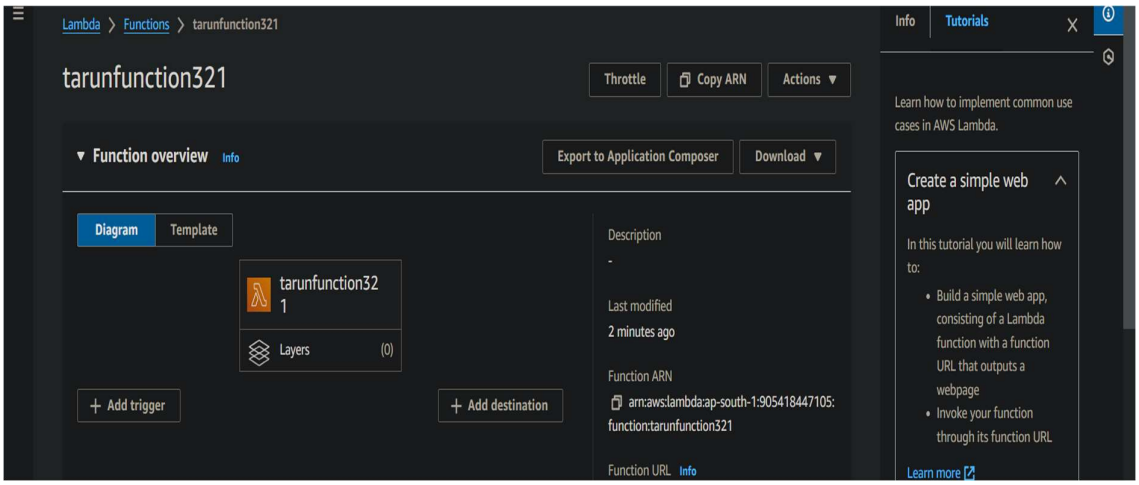


- Now replace the name and ARN by your source bucket and by your any upload image file name. note: we have to upload the file first in the source bucket that we enter in the key.

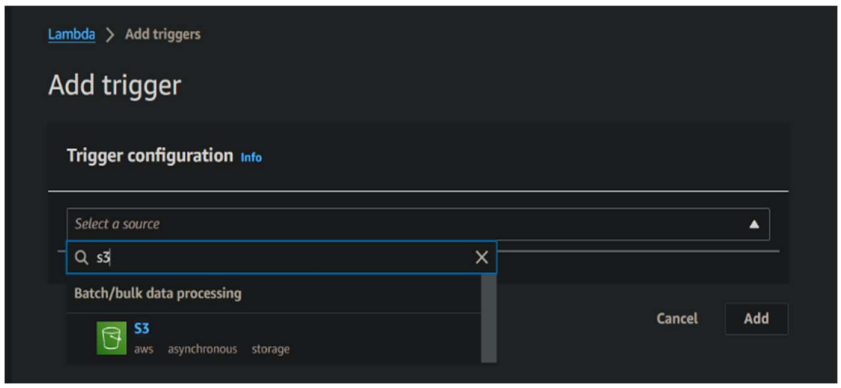
```
{
  "eventVersion": "2.0",
  "eventSource": "aws:s3",
  "awsRegion": "us-east-1",
  "eventTime": "1970-01-01T00:00:00.000Z",
  "eventName": "ObjectCreated:Put",
  "userIdentity": {
    "principalId": "EXAMPLE"
  },
  "requestParameters": {
    "sourceIPAddress": "127.0.0.1"
  },
  "responseElements": {
    "x-amz-request-id": "EXAMPLE123456789",
    "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
  },
  "s3": {
    "s3SchemaVersion": "1.0",
    "configurationId": "testConfigRule",
    "bucket": {
      "name": "tarun.sourcebucket",
      "ownerIdentity": {
        "principalId": "EXAMPLE"
      },
      "arn": "arn:aws:s3:::tarun.sourcebucket"
    },
    "object": {
      "key": "lakeview.jpg",
      "size": 1024,
      "eTag": "0123456789abcdef0123456789abcdef",
      "sequencer": "0A1B2C3D4E5F678901"
    }
  }
 }
]
}
```

Step 5: Adding trigger in lambda function

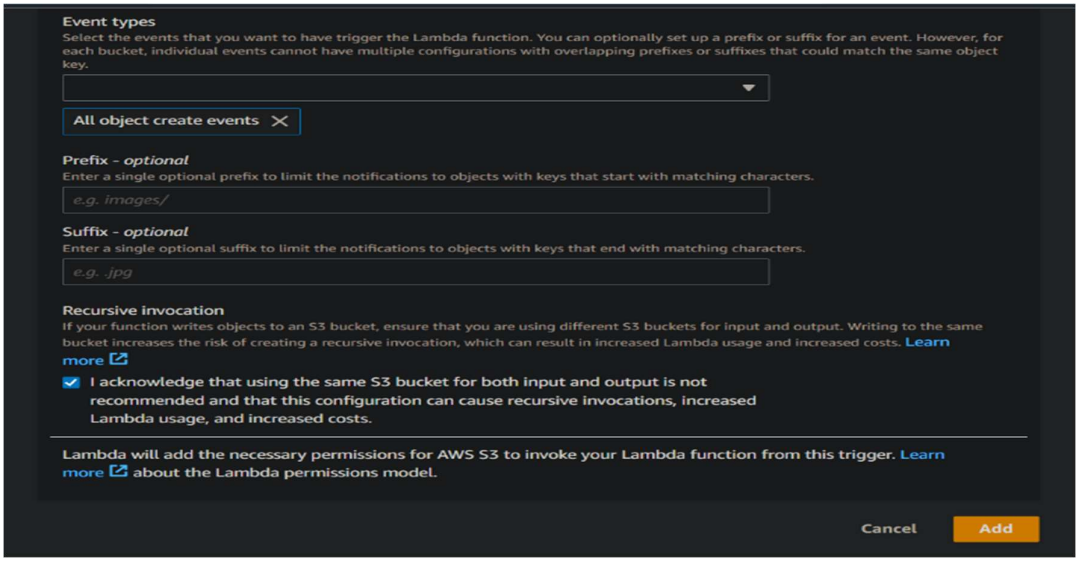
- Now we have to add trigger in our function.



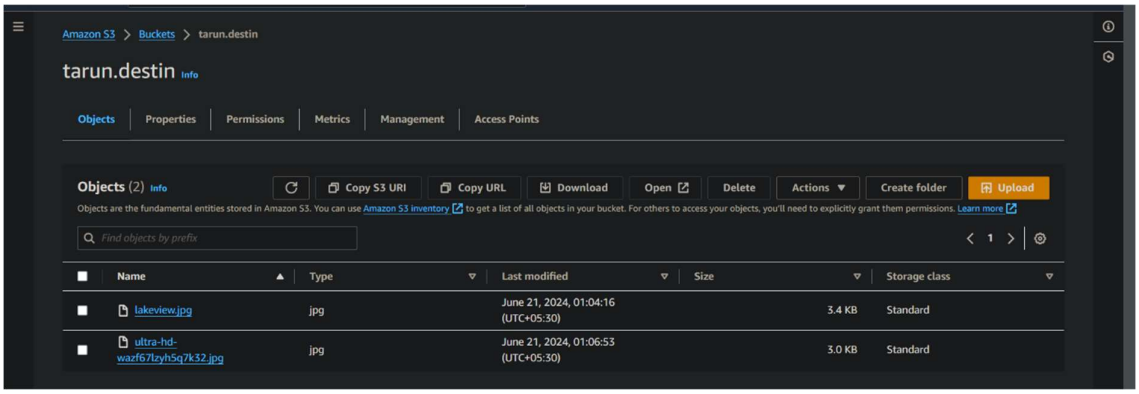
- In add trigger select S3 as source.



- Select your source bucket and all object create event and then click on the add button.



- Now we can see the source bucket file in destination bucket and the images size are also changing.



Results:

make sure to edit the policy of the bucket to make it public and see objects.

```
{
 "Version":"2012-10-17",
 "Statement": [{
   "Sid":"PublicReadGetObject",
   "Effect":"Allow",
   "Principal": "*",
   "Action": ["s3:GetObject"],
   "Resource": ["arn:aws:s3:::example-bucket/*"]
 }]
}
```