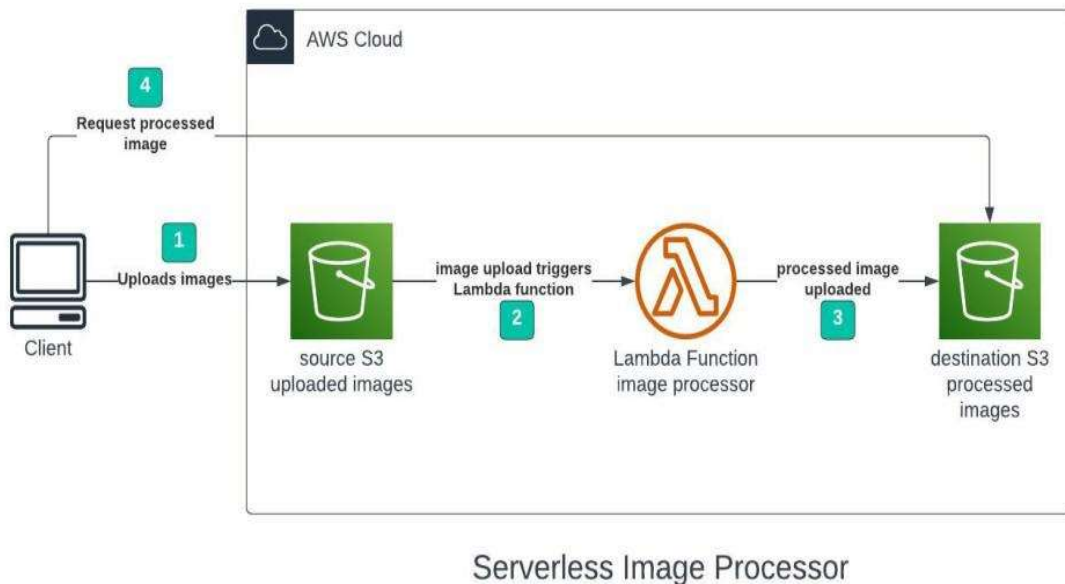


Serverless Image Processing

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.



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.

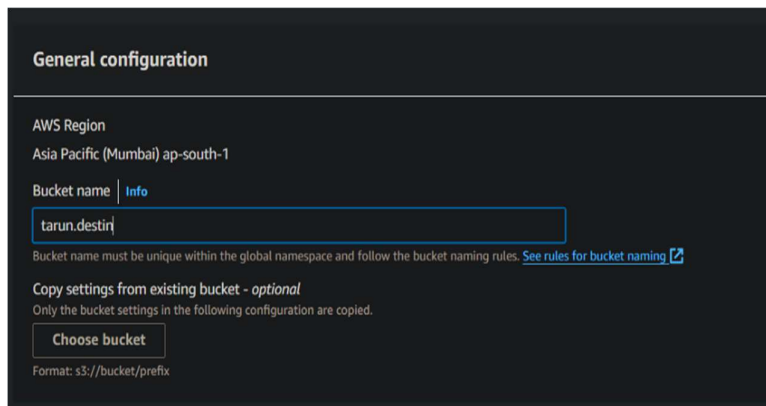
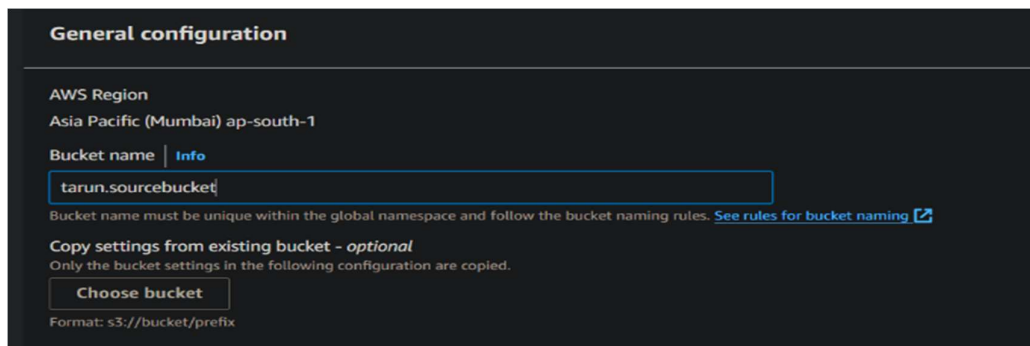
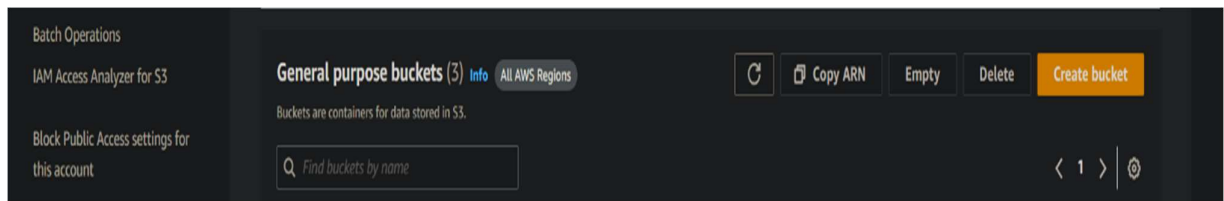
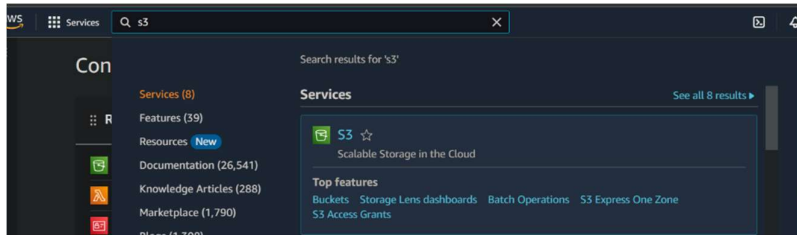
Step 3: Create role.

Step 4: Creating the Lambda Function.

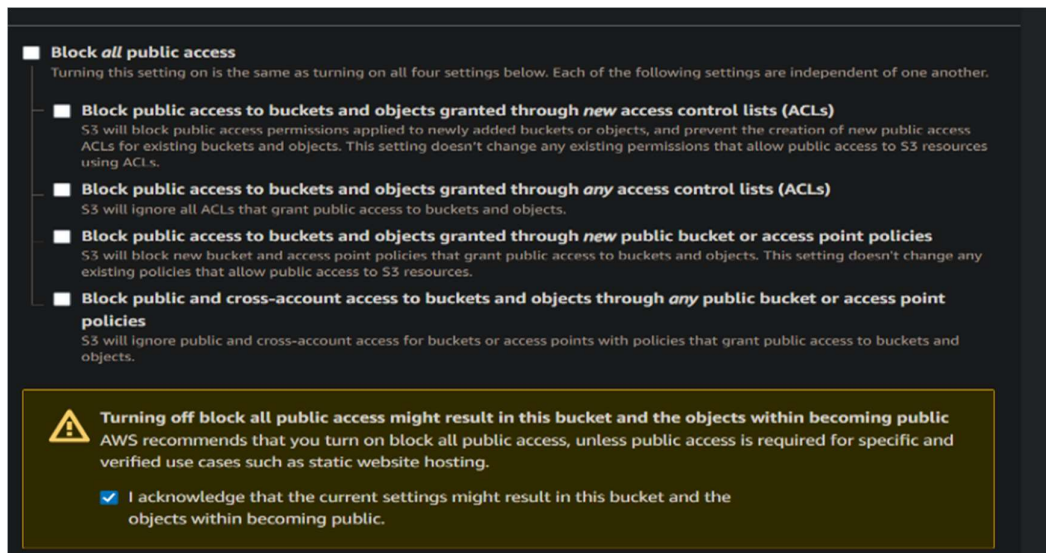
Step 5: Adding trigger in lambda function.

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.

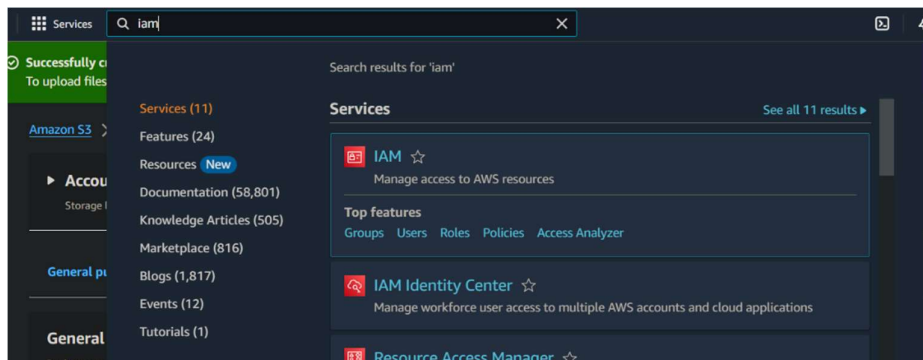


- Make sure to uncheck block all public access.

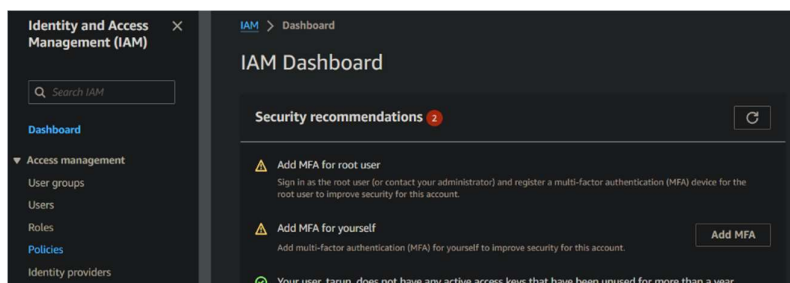


Step 2: Setting Up IAM Role

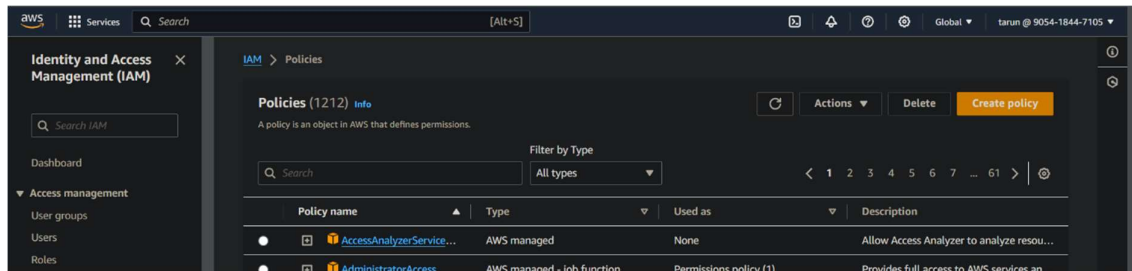
- Create a Custom IAM Policy.
- Go to the IAM dashboard



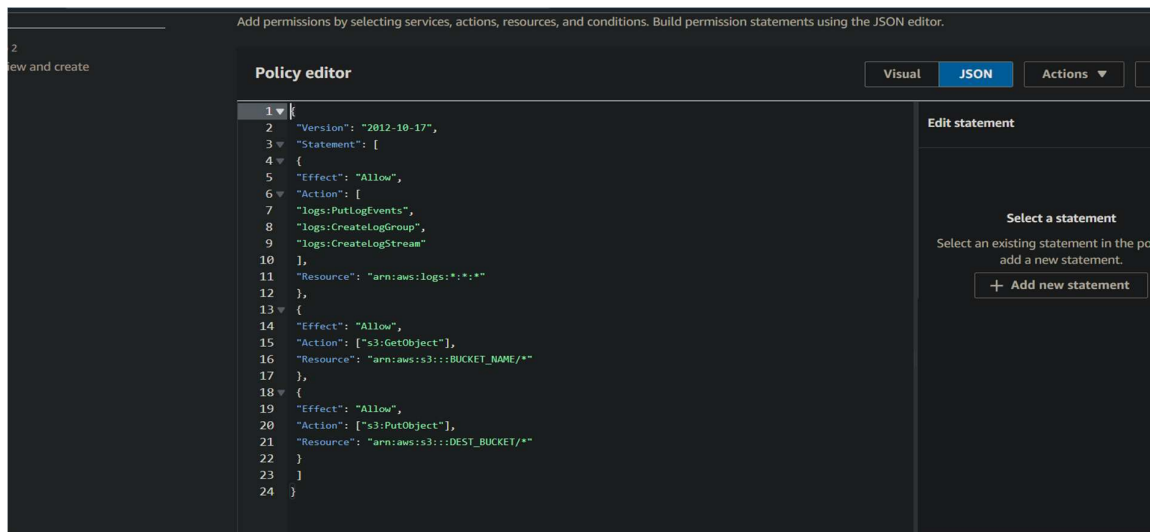
- Click to policies on the left corner.



- Click on create new policy

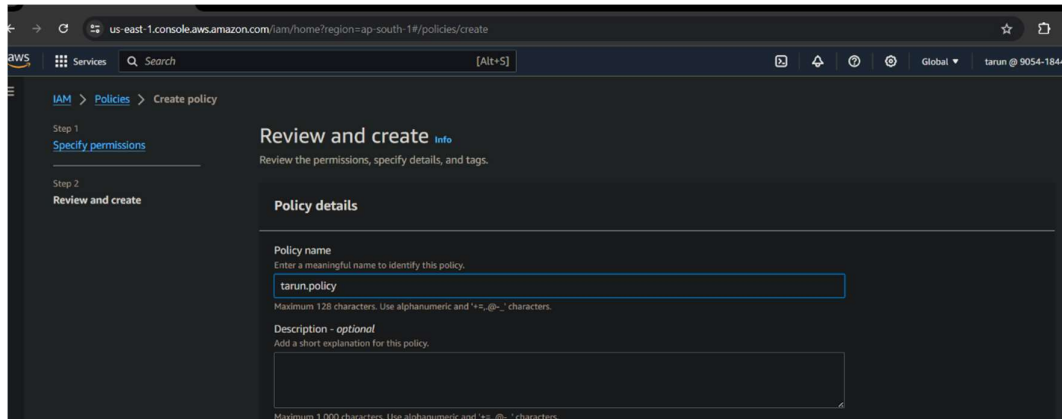


- Replace the policy with the given code and change the source and destination bucket ARN.



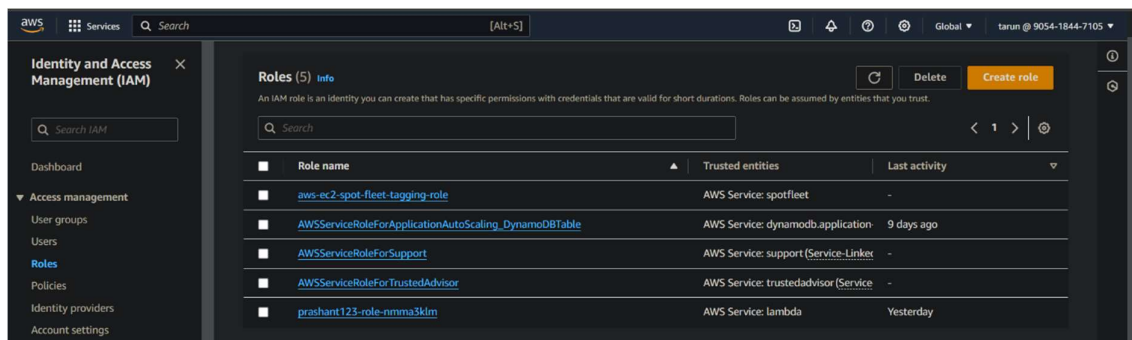
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": "arn:aws:s3:::input-bucket-199/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
  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

- In the review and 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.

AWS service
Allow AWS services like EC2, Lambda, or others to perform actions in this account.

AWS account
Allow entities in other AWS accounts belonging to you or a 3rd party to perform actions in this account.

Web identity
Allows users federated by the specified external web identity provider to assume this role to perform actions in this account.

SAML 2.0 federation
Allow users federated with SAML 2.0 from a corporate directory to perform actions in this account.

Custom trust policy
Create a custom trust policy to enable others to perform actions in this account.

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
Lambda

Choose a use case for the specified service.
Use case
☒ **Lambda**
 Allows Lambda functions to call AWS services on your behalf.

Cancel Next

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

IAM > Roles > Create role

Step 1
Select trusted entity

Step 2
Add permissions

Step 3
Name, review, and create

Add permissions Info

Permissions policies (1/939) Info
Choose one or more policies to attach to your new role.

Filter by Type
 All types 1 match

<input checked="" type="checkbox"/>	Policy name	Type	Description
<input checked="" type="checkbox"/>	tarun.policy	Customer managed	-

► Set permissions boundary - optional

Cancel Previous 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 2
Add permissions

Step 3
Name, review, and create

Role details

Role name
Enter a meaningful name to identify this role.

 Maximum 64 characters. Use alphanumeric and '+', '@', '-', '.' characters.

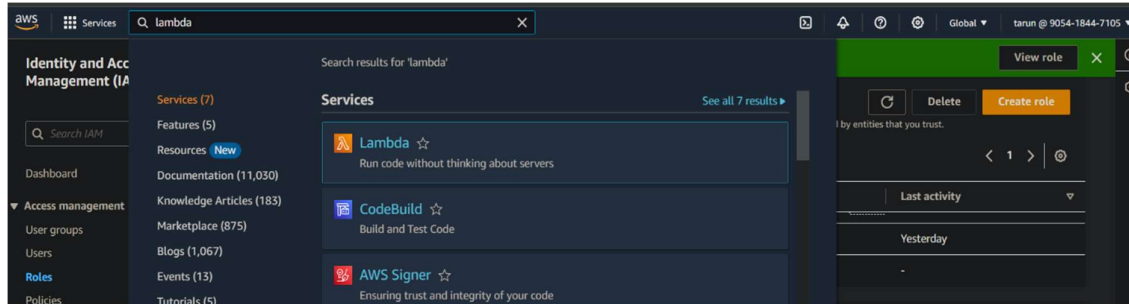
Description
Add a short explanation for this role.

 Maximum 1000 characters. Use letters (A-Z and a-z), numbers (0-9), tabs, new lines, or any of the following characters: _ + = , @ - / [] ! # \$ % ^ & * ' () ; : < > .

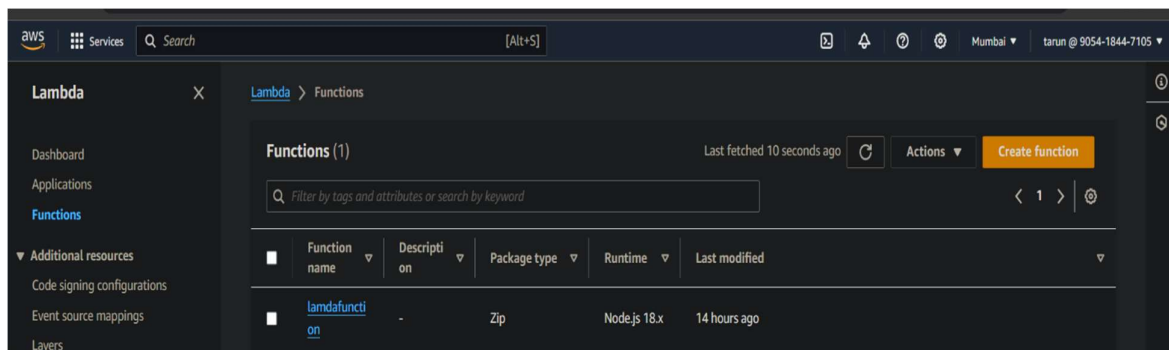
Step 1: Select trusted entities

Step 4: Creating the Lambda Function

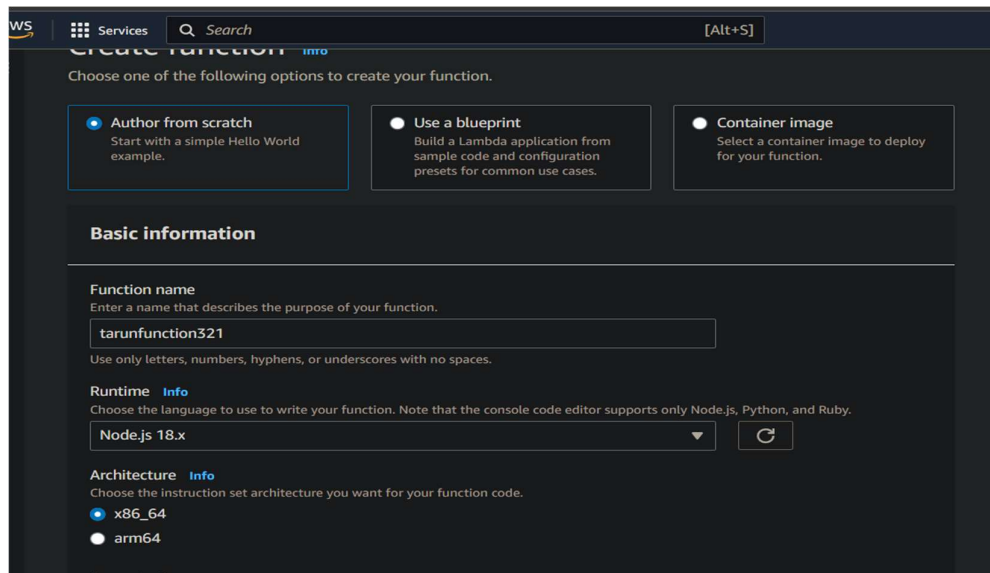
- Head over to the AWS Lambda dashboard by searching lambda on the services.



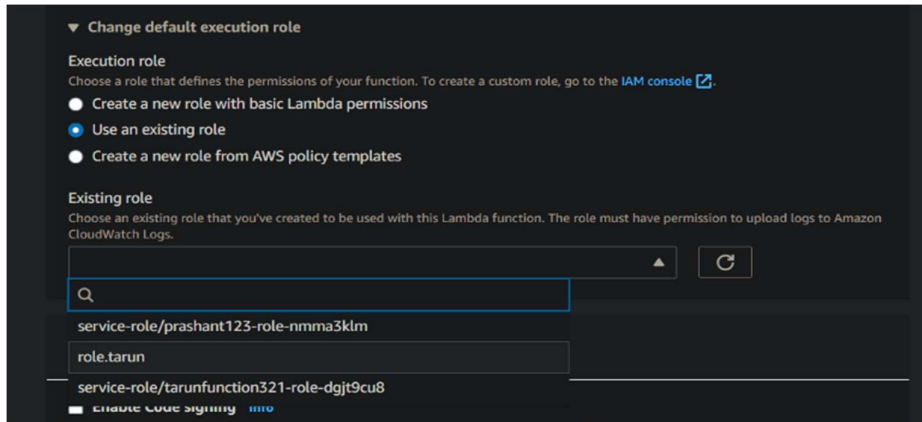
- Now click on the create function.



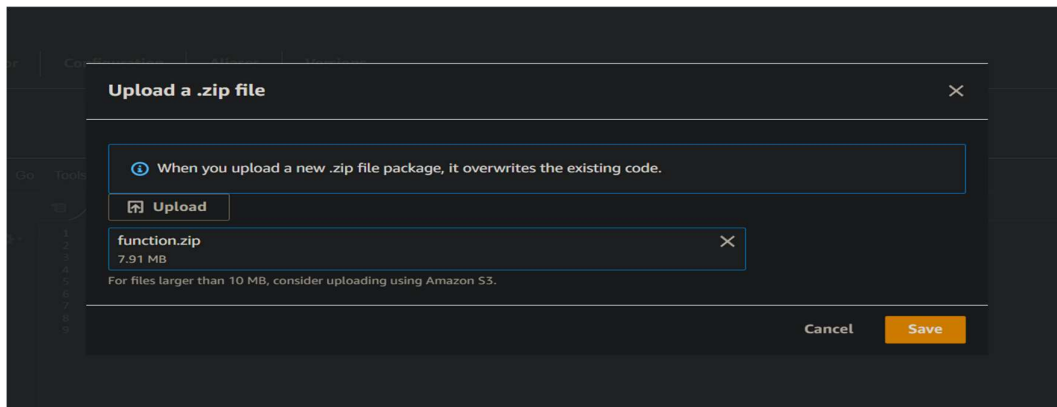
- Now choose author from scratch in the above section and the in the function name give your function a name in my case it is tarunfunction321.and in runtime select node.js 18x.



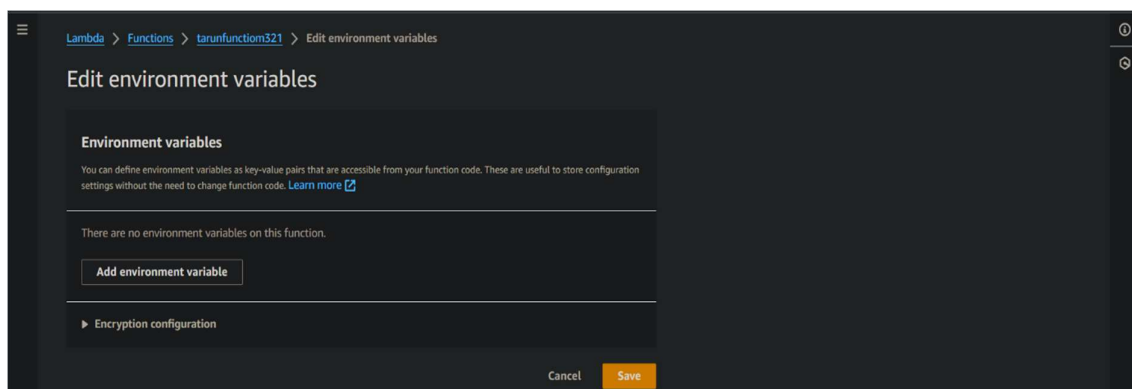
- Now click use an existing role in the execution role and search or select your role in the existing role section.



- 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.

Lambda > Functions > tarunfunction321 > Edit environment variables

Edit environment variables

Environment variables

You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. [Learn more](#)

Key	Value	
DEST_BUCKET	tarun.destin	Remove

[Add environment variable](#)

► Encryption configuration

Cancel 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.

To invoke your function without saving an event, configure the JSON event, then choose Test.

Test event action

☒ Create new event ☐ Edit saved event

Event name

tarun

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

☒ Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

☐ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

hello-world

Event JSON

[Format JSON](#)

1 + {
2 "key1": "value1",
3 "key2": "value2",
4 "key3": "value3",
5 }

Learn how to implement common cases in AWS Lambda.

Create a simple web app

In this tutorial you will learn how to:

- Build a simple web app consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

[Learn more](#)

[Start tutorial](#)

- In the template section select S3 put request .

Event sharing settings

☒ Private

This event is only available in the Lambda console and to the event creator. You can configure a total of 10. [Learn more](#)

☐ Shareable

This event is available to IAM users within the same account who have permissions to access and use shareable events. [Learn more](#)

Template - optional

hello-world

Q s3

AWS

Recognition S3 Request

S3 Batch Operations Invocation

S3 Delete

S3 Put

- 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.

```

3  {
4    "eventVersion": "2.0",
5    "eventSource": "aws:s3",
6    "awsRegion": "us-east-1",
7    "eventTime": "1970-01-01T00:00:00.000Z",
8    "eventName": "ObjectCreated:Put",
9    "userIdentity": {
10     "principalId": "EXAMPLE"
11   },
12   "requestParameters": {
13     "sourceIPAddress": "127.0.0.1"
14   },
15   "responseElements": {
16     "x-amz-request-id": "EXAMPLE123456789",
17     "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"
18   },
19   "s3": {
20     "s3SchemaVersion": "1.0",
21     "configurationId": "testConfigRule",
22     "bucket": {
23       "name": "tarun.sourcebucket",
24       "ownerIdentity": {
25         "principalId": "EXAMPLE"
26       },
27       "arn": "arn:aws:s3:::tarun.sourcebucket"
28     },
29     "object": {
30       "key": "lakeview.jpg",
31       "size": 1024,
32       "eTag": "0123456789abcdef0123456789abcdef",
33       "sequencer": "0A1B2C3D4E5F678901"
34     }
35   }
36 }
37
38 }

```

Step 5: Adding trigger in lambda function

- Now we have to add trigger in our function.

The screenshot shows the AWS Lambda console interface for a function named 'tarunfunction321'. The 'Function overview' tab is active, displaying a diagram of the function with one layer. The 'Add trigger' button is visible. On the right, the 'Info' tab shows the function's ARN and URL. A tutorial sidebar on the far right offers guidance on creating a simple web app.

Function overview

Diagram | Template

tarunfunction321

Layers (0)

+ Add trigger

+ Add destination

Export to Application Composer | Download

Throttle | Copy ARN | Actions

Description

Last modified: 2 minutes ago

Function ARN: arn:aws:lambda:ap-south-1:905418447105:function:tarunfunction321

Function URL: [Info](#)

Info | Tutorials

Learn how to implement common use cases in AWS Lambda.

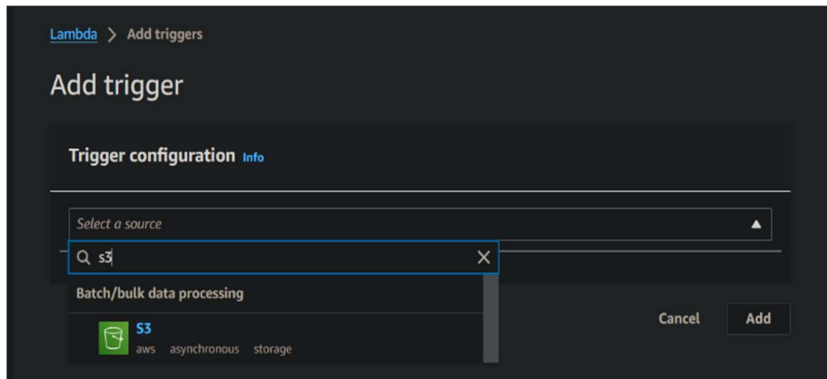
Create a simple web app

In this tutorial you will learn how to:

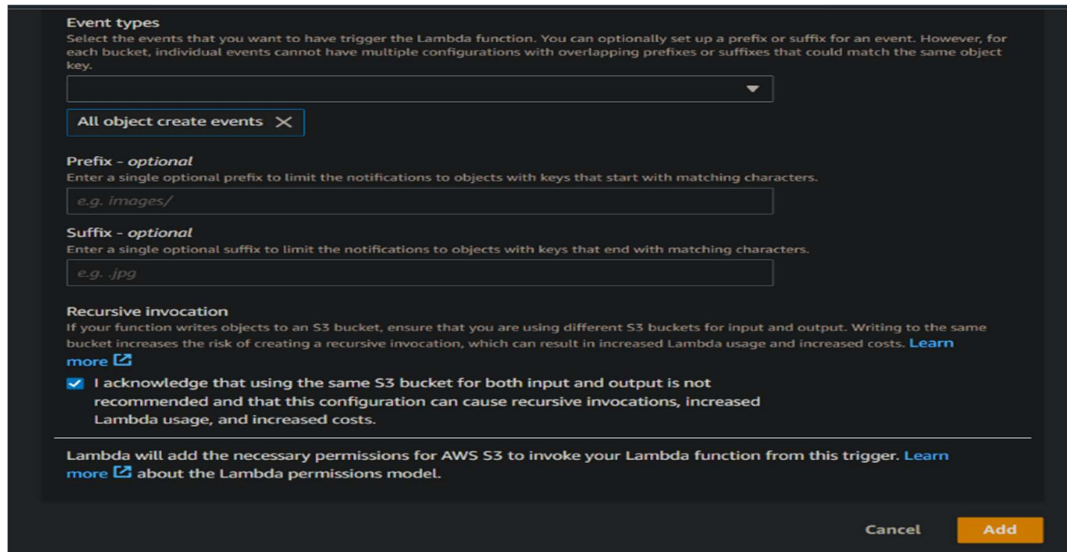
- Build a simple web app, consisting of a Lambda function with a function URL that outputs a webpage
- Invoke your function through its function URL

[Learn more](#)

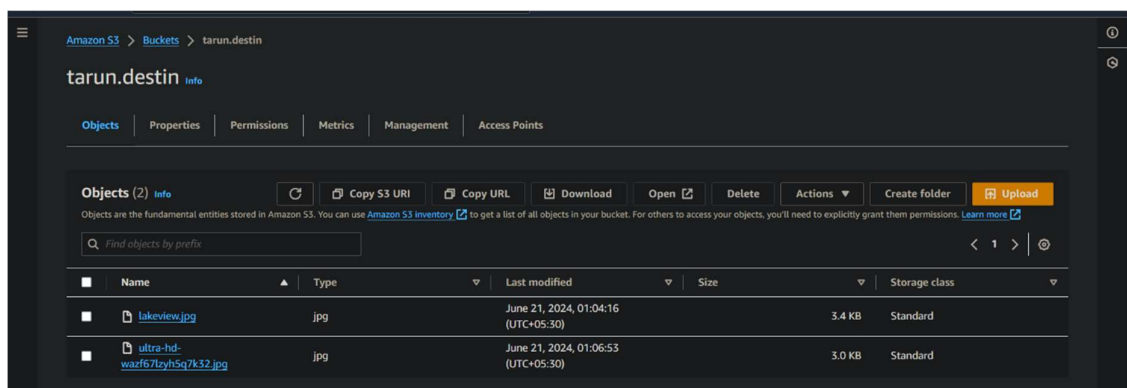
- In add trigger select S3 as source.



- 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/*"]  
  }]  
}
```

