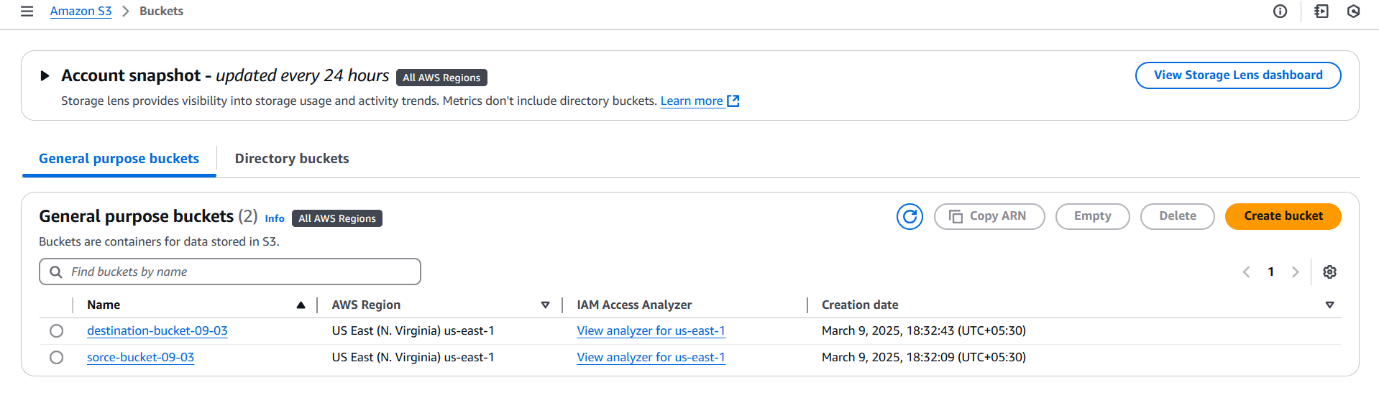
First, we create two S3 buckets named **"source"** and **"destination."**



After that, we create a policy to grant access to the bucket and its objects.

**Policy:**

{

"Version": "2012-10-17",

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:ListBucket",

"s3:DeleteObject"

],

"Resource": [

"arn:aws:s3:::sorce-bucket-09-03",

"arn:aws:s3:::sorce-bucket-09-03/\*"

]

},

{

"Effect": "Allow",

"Action": [

"s3:PutObject"

],

"Resource": [

"arn:aws:s3:::destination-bucket-09-03",

"arn:aws:s3:::destination-bucket-09-03/\*"

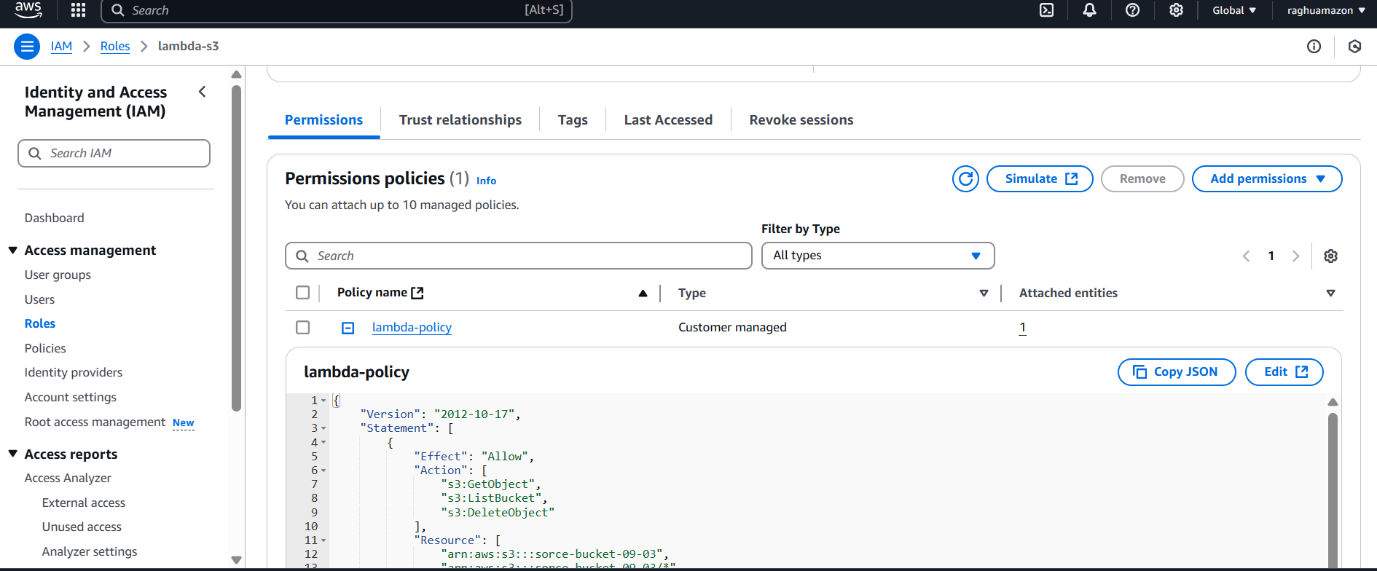
]

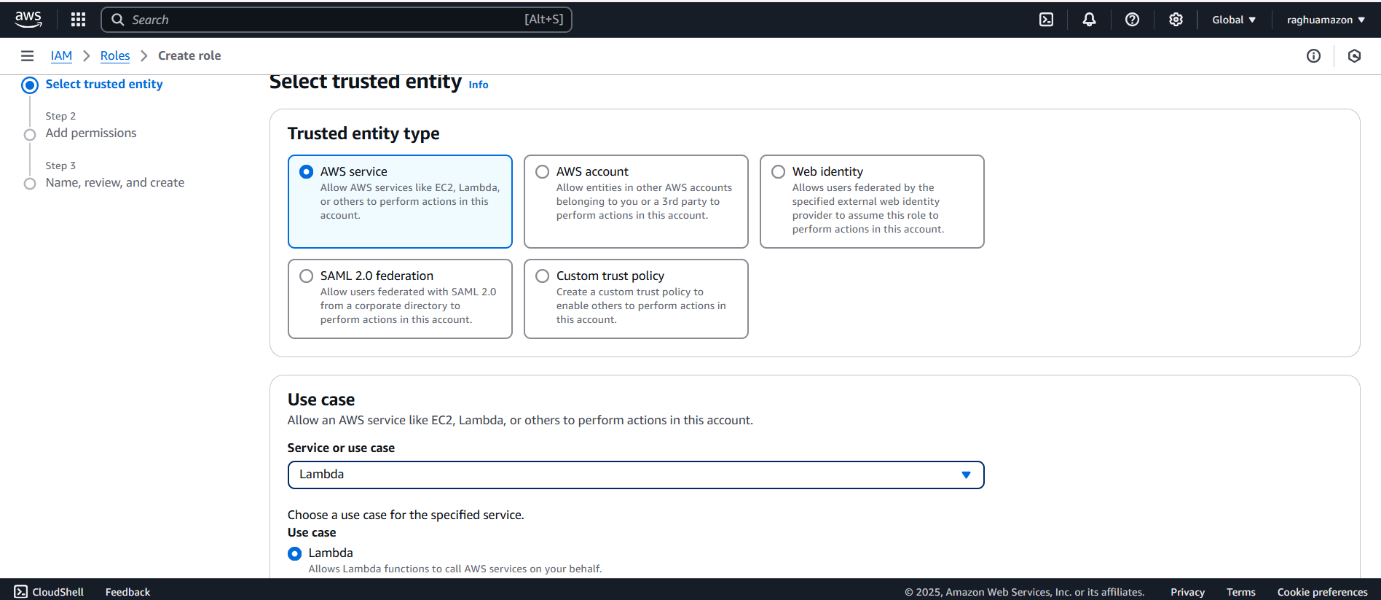
}

]

}

After creating the policy, we move on to the role and attach the created policy to it.





After that, I create a Lambda function and select **"Author from scratch."**

There are three different options available:

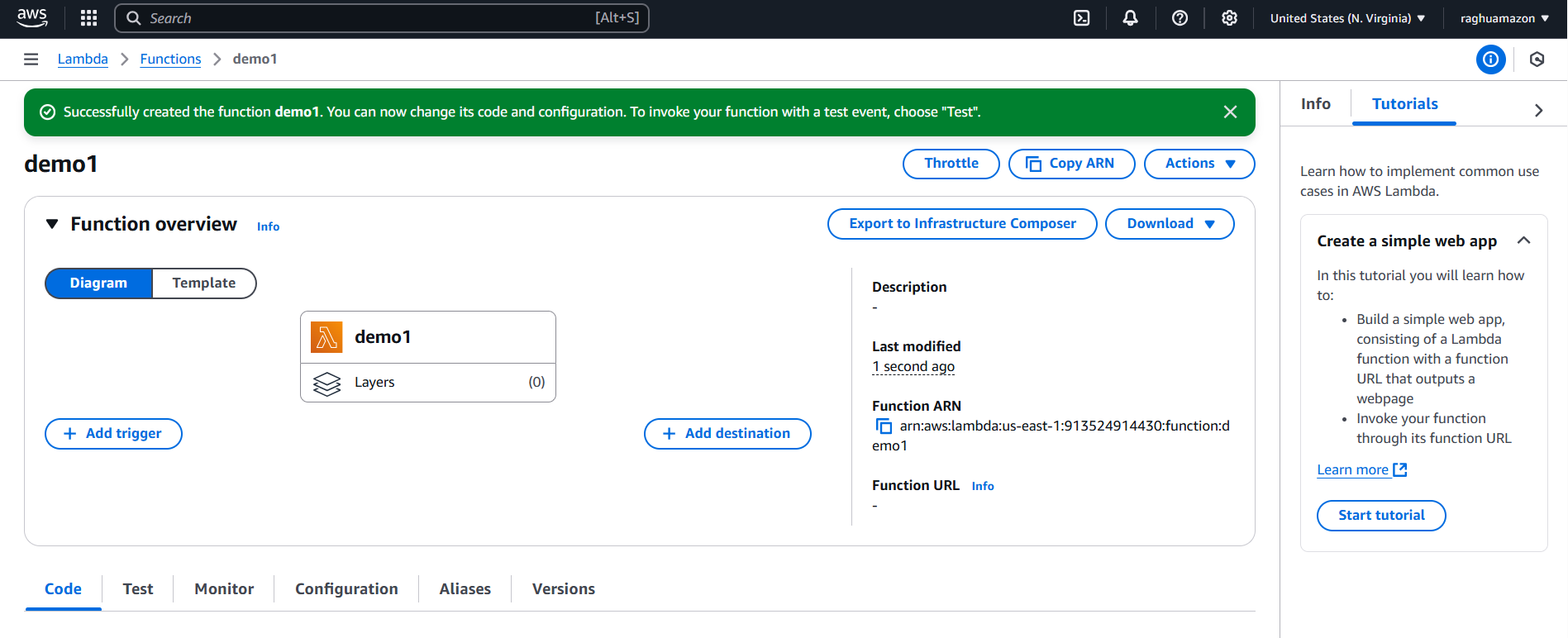
1. **Author from scratch**
2. **Use a blueprint**
3. **Container image**

I choose **"Author from scratch."**

In the runtime settings, I select **Python** as the programming language.

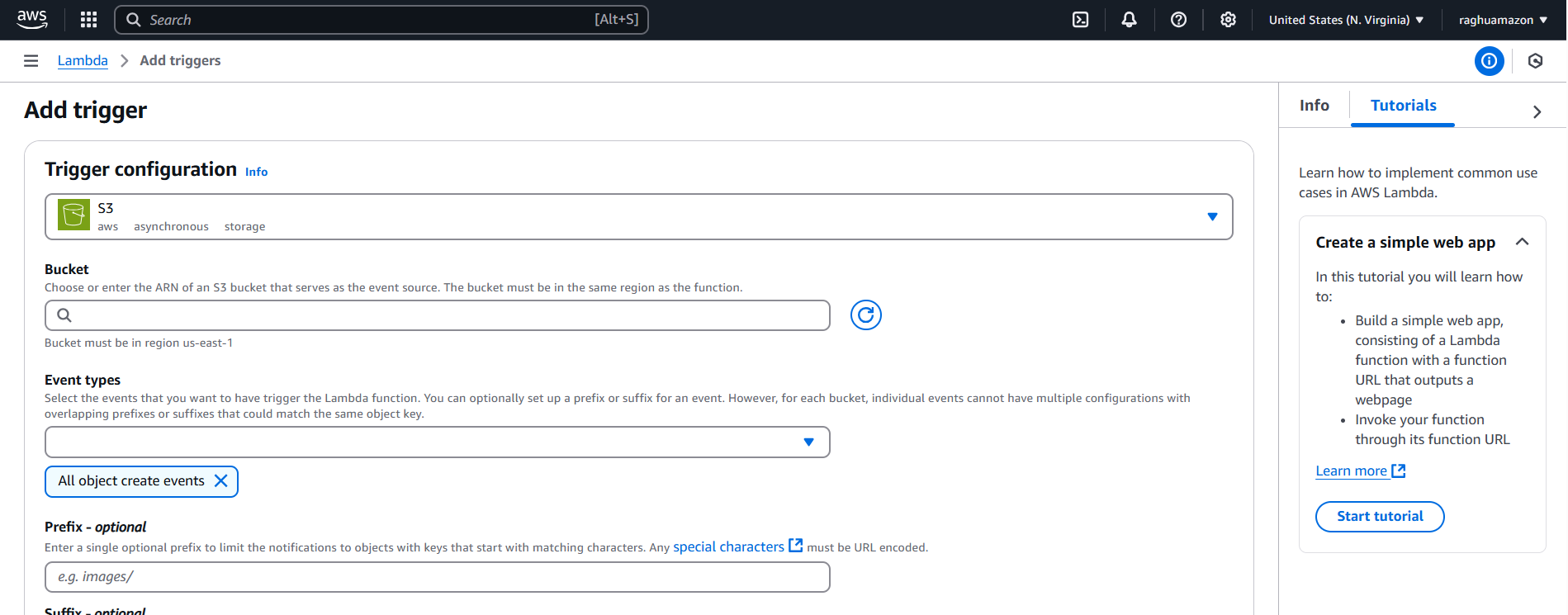
After that, I change the default execution role and attach the previously created role to the Lambda function.

**And then, I create the Lambda function.**

**So, the Lambda function is created, and the development environment is displayed below.**

We create a trigger so that Lambda knows when a new object is added to the source bucket. Without a trigger, Lambda won't detect changes in the bucket.

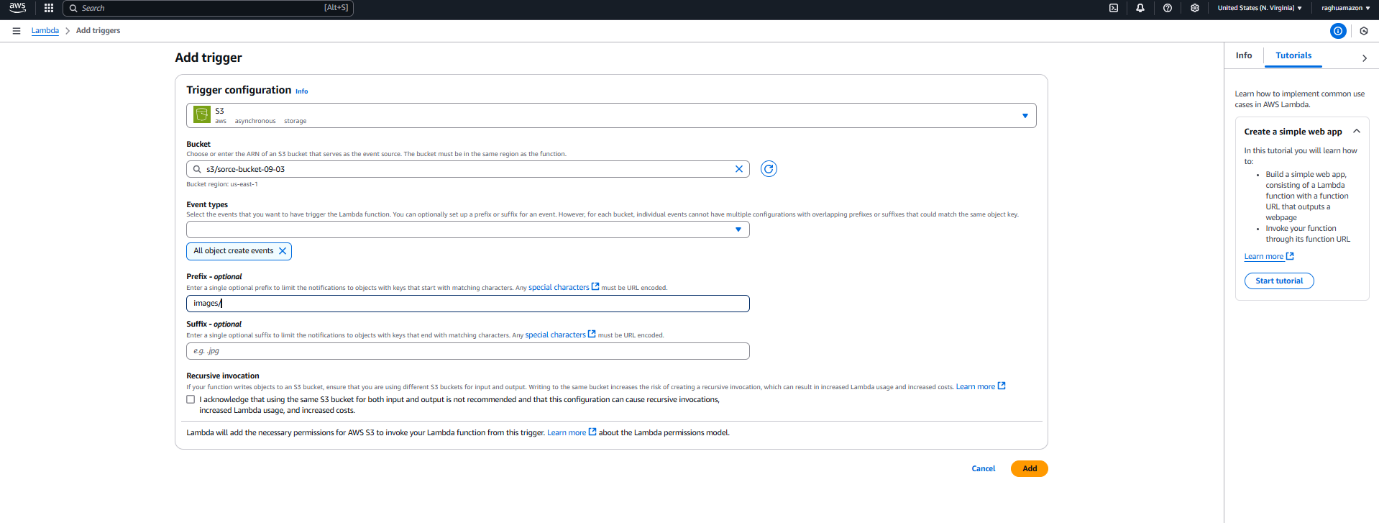
Create a trigger

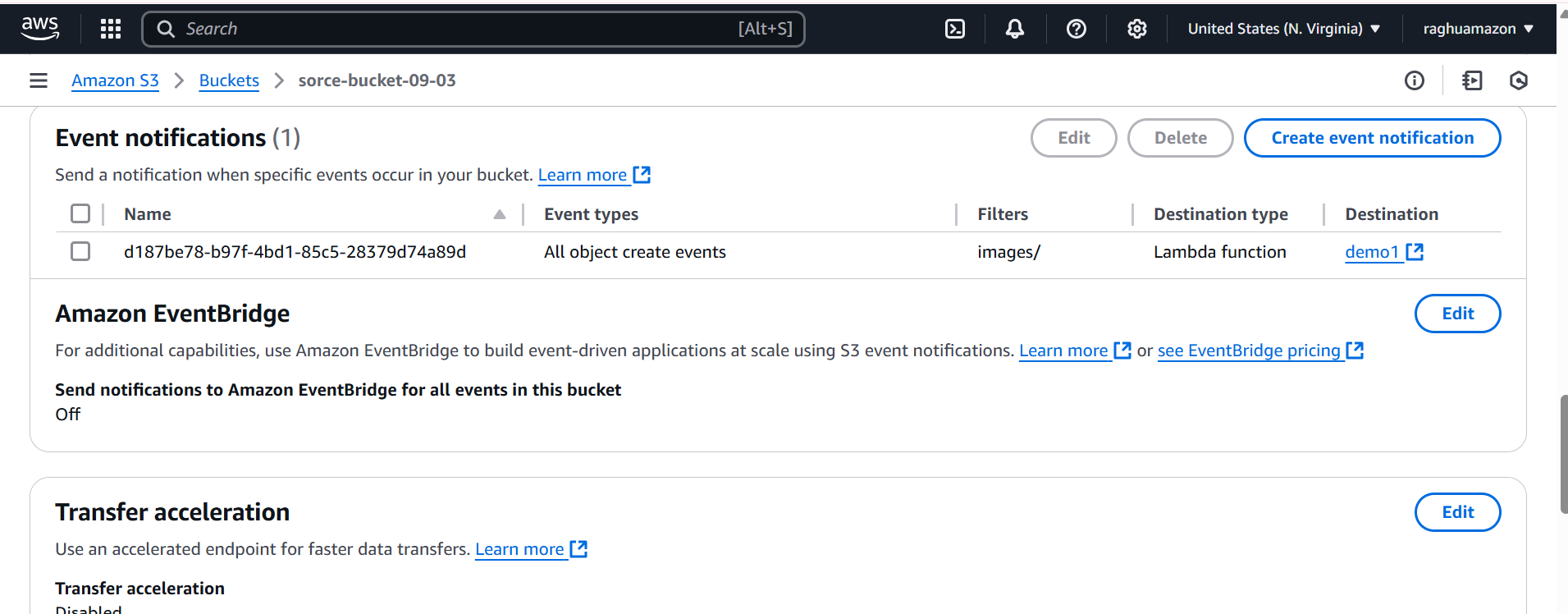


1. I select the event type based on the requirement, like **"All object create events."**

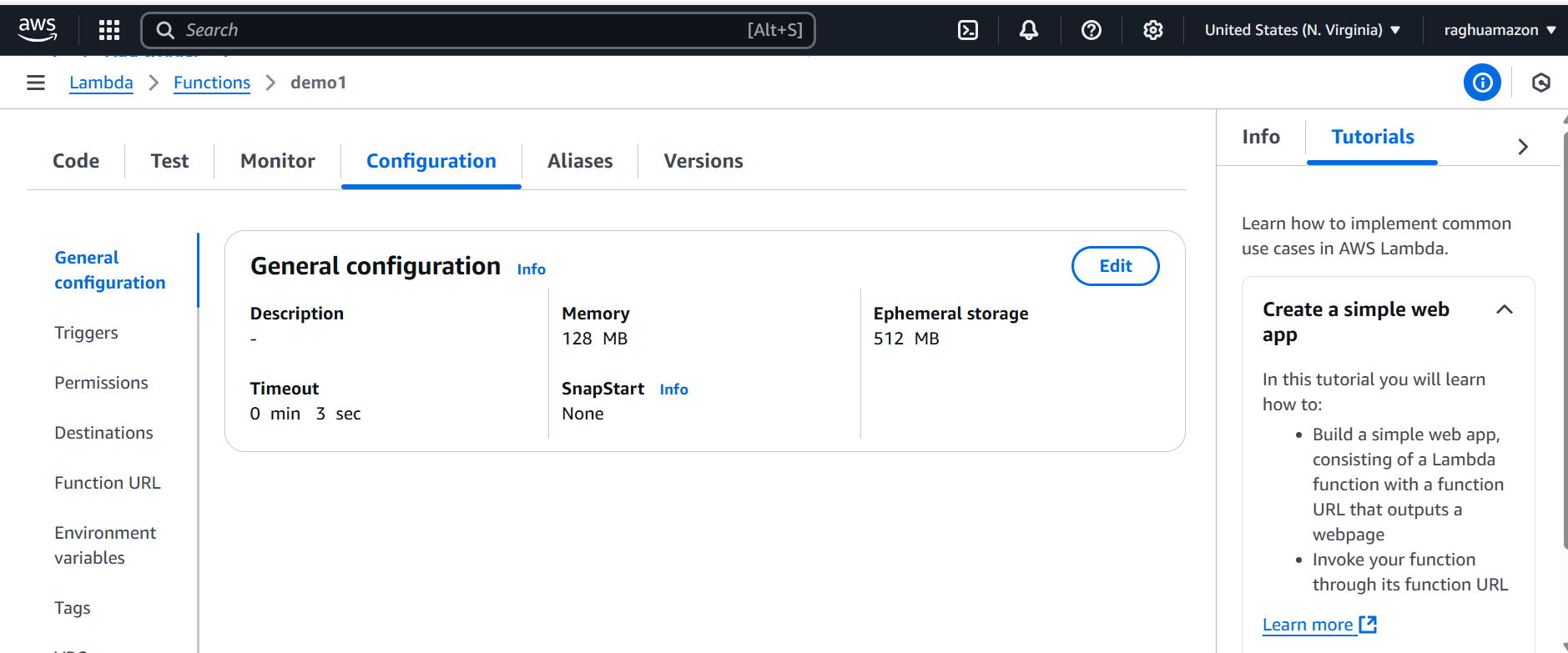
2. If the **prefix** is empty, it includes all objects. But I set it to **"images/"** so only files in the **images folder** will be sent to the destination.

3. The **suffix** is optional. If I enter **".jpg" or ".png,"** only those image types will be sent.



After that, I go to the S3 bucket and add an **event notification**. This is needed because Lambda has permission to access S3, but it needs an event to trigger it.

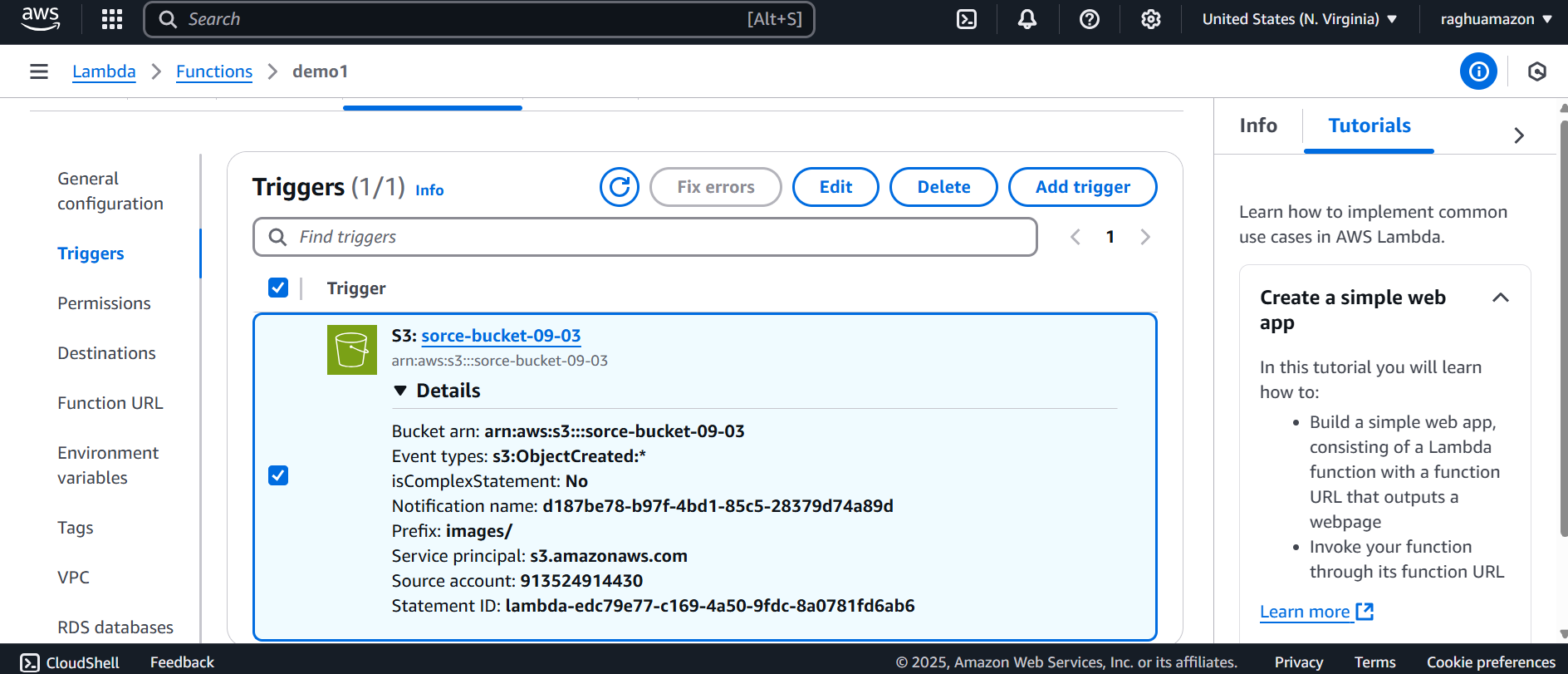
In the above process, when we upload an object to S3, the event is sent to Lambda, and Lambda receives it. However, **Lambda doesn’t automatically know what action to perform**—we need to write code to define the action.



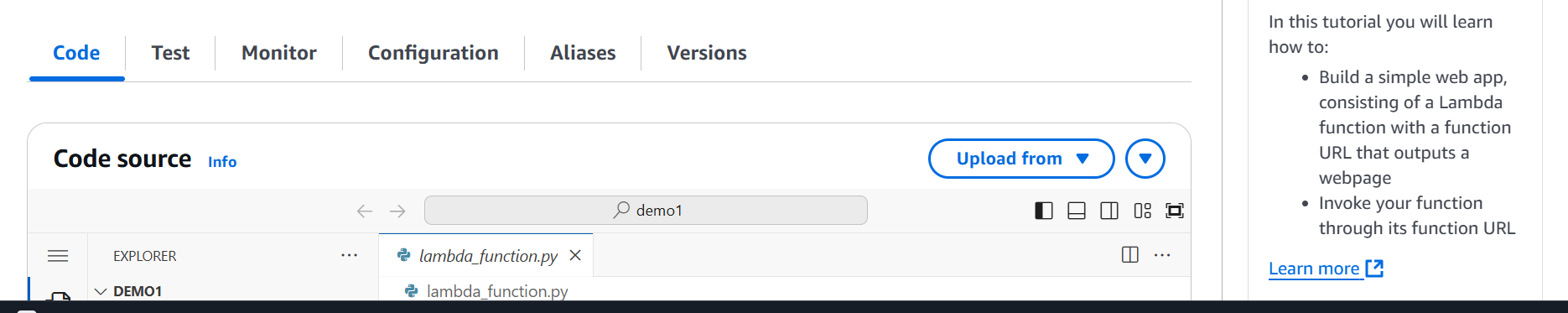
Go to **Configuration** → Then go to **Triggers**.

If anything needs to be changed, go to **Triggers** and update it.

We can also **add multiple triggers** as needed.



We have some different options there



Since we are working with AWS, we need to **install Boto3**.

After that, we **write a Python script** to handle the S3 event and perform the required action.

import boto3

import json

s3 = boto3.client('s3')

SOURCE\_BUCKET = "source-bucket-name"

DESTINATION\_BUCKET = "destination-bucket-name"

def lambda\_handler(event, context):

try:

# List all objects in the source bucket

objects = s3.list\_objects\_v2(Bucket=SOURCE\_BUCKET)

# Check if the bucket is not empty

if "Contents" in objects:

for obj in objects["Contents"]:

source\_key = obj["Key"] # Get object key (file name)

# Copy object to destination bucket

copy\_source = {'Bucket': SOURCE\_BUCKET, 'Key': source\_key}

s3.copy\_object(

Bucket=DESTINATION\_BUCKET,

Key=source\_key,

CopySource=copy\_source

)

print(f"Copied: {source\_key} from {SOURCE\_BUCKET} to {DESTINATION\_BUCKET}")

else:

print(f"No objects found in {SOURCE\_BUCKET}")

return {

"statusCode": 200,

"body": json.dumps({

"message": f"Objects copied successfully from {SOURCE\_BUCKET} to {DESTINATION\_BUCKET}"

})

}

except Exception as e:

print(f"Error: {str(e)}")

return {

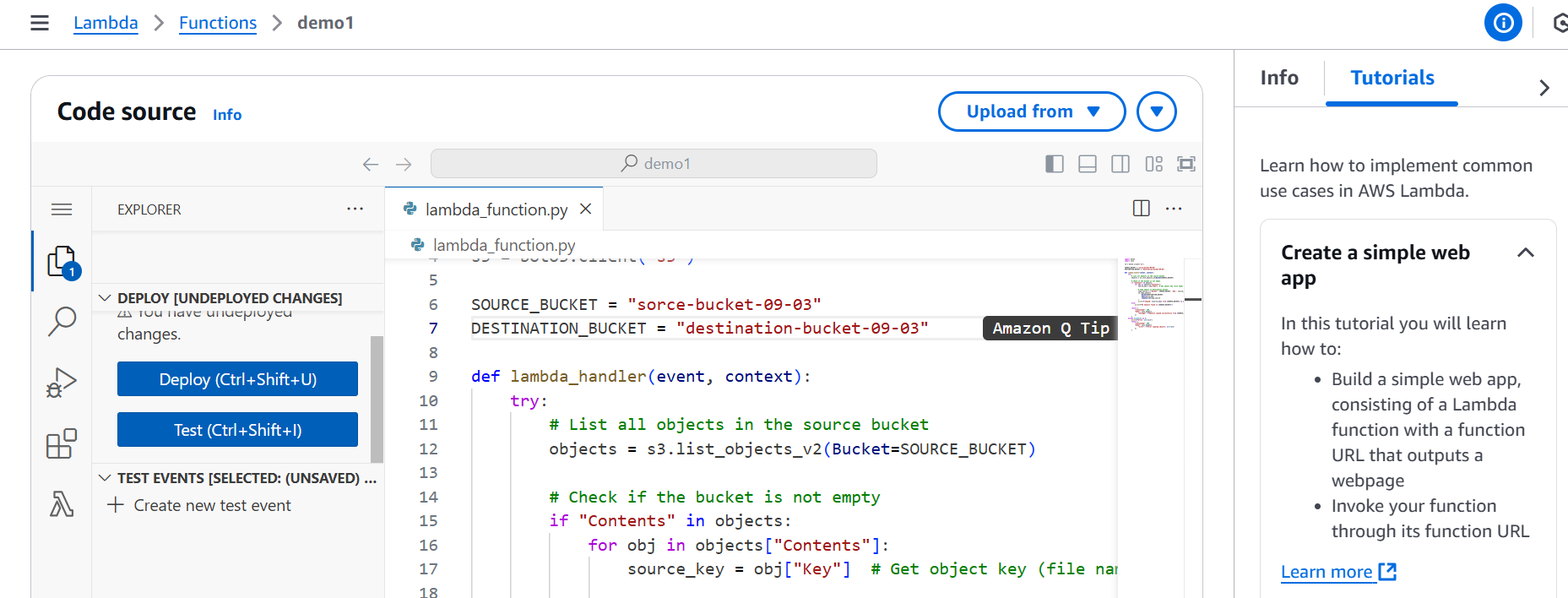
"statusCode": 500,

"body": json.dumps({

"error": f"Error copying objects: {str(e)}"

})

}



After deploying (saving) the code, if you want to test it, **create a test event**.

At first, both buckets are **empty**.

When I upload an image to the **source bucket**, it is **automatically copied** to the **destination bucket**.

