AWS LAMBDA

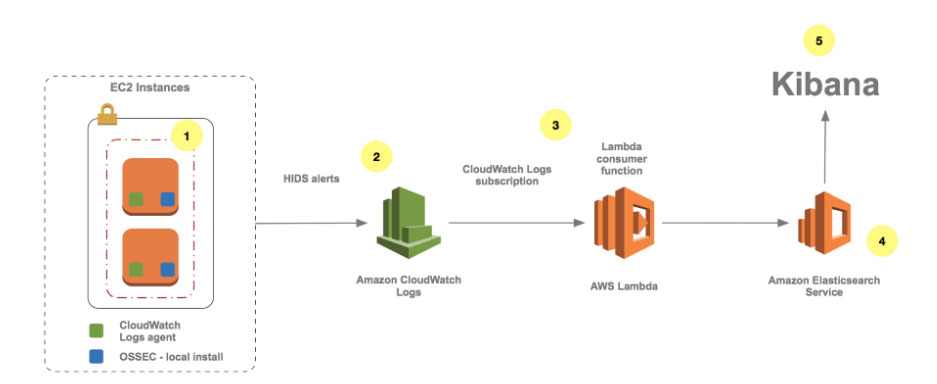
**AWS Lambda Step-by-Step Guide**

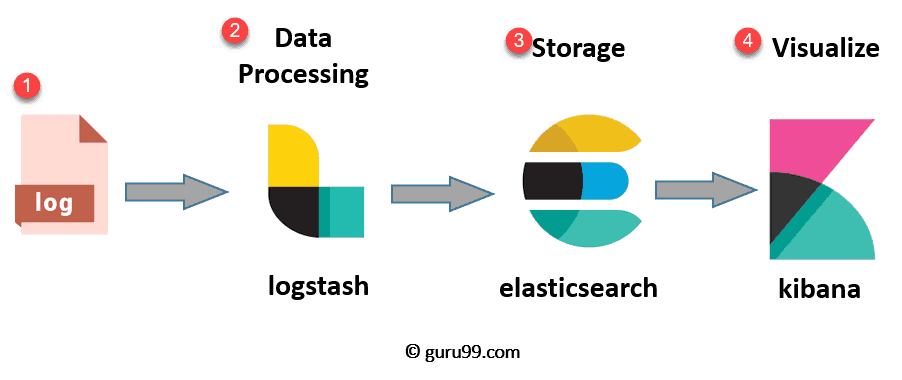
## **Introduction to AWS Lambda**

AWS Lambda is a serverless computing service that runs code in response to events and automatically manages the underlying infrastructure. It supports multiple programming languages, integrates with various AWS services, and charges only for execution time.

### **Key Features of AWS Lambda**

* Fully managed serverless service
* Event-driven execution
* Supports multiple programming languages (Python, Node.js, Java, etc.)
* Auto-scales based on demand
* Integrated with AWS services like S3, API Gateway, DynamoDB, and CloudWatch





## **Creating an AWS Lambda Function**

### **Step 1: Log in to AWS Console**

1. Go to the AWS Management Console.
2. Search for "Lambda" and open the AWS Lambda service.

### **Step 2: Create a New Lambda Function**

1. Click on "Create function."
2. Choose "Author from scratch."
3. Enter the function name (e.g., MyFirstLambda).
4. Choose the runtime (e.g., **Python 3.9**).
5. Select an execution role:
   * Choose "Create a new role with basic Lambda permissions."

{

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

"Statement": [

{

"Effect": "Allow",

"Action": [

"logs:\*"

],

"Resource": "arn:aws:logs:\*:\*:\*"

},

{

"Effect": "Allow",

"Action": "ec2:Describe\*",

"Resource": "\*"

},

{

"Effect": "Allow",

"Action": [

"ec2:CreateSnapshot",

"ec2:DeleteSnapshot",

"ec2:CreateTags",

"ec2:ModifySnapshotAttribute",

"ec2:ResetSnapshotAttribute"

],

"Resource": [

"\*"

]

}

]

}

1. Click "Create function."

### **Step 3: Write and Deploy the Code**

1. Scroll down to the **Code** section.
2. Replace the default code with the following Python script:

import json

# Backup all in-use volumes in all regions

import boto3

def lambda\_handler(event, context):

ec2 = boto3.client('ec2')

# Get list of regions

regions = ec2.describe\_regions().get('Regions',[] )

# Iterate over regions

for region in regions:

print("Checking region %s " % region['RegionName'])

reg=region['RegionName']

# Connect to region

ec2 = boto3.client('ec2', region\_name=reg)

# Get all in-use volumes in all regions

result = ec2.describe\_volumes( Filters=[{'Name': 'status', 'Values': ['in-use']}])

for volume in result['Volumes']:

print("Backing up %s in %s" % (volume['VolumeId'], volume['AvailabilityZone']))

# Create snapshot

result = ec2.create\_snapshot(VolumeId=volume['VolumeId'],Description='Created by Lambda backup function ebs-snapshots')

# Get snapshot resource

ec2resource = boto3.resource('ec2', region\_name=reg)

snapshot = ec2resource.Snapshot(result['SnapshotId'])

volumename = 'N/A'

# Find name tag for volume if it exists

if 'Tags' in volume:

for tags in volume['Tags']:

if tags["Key"] == 'Name':

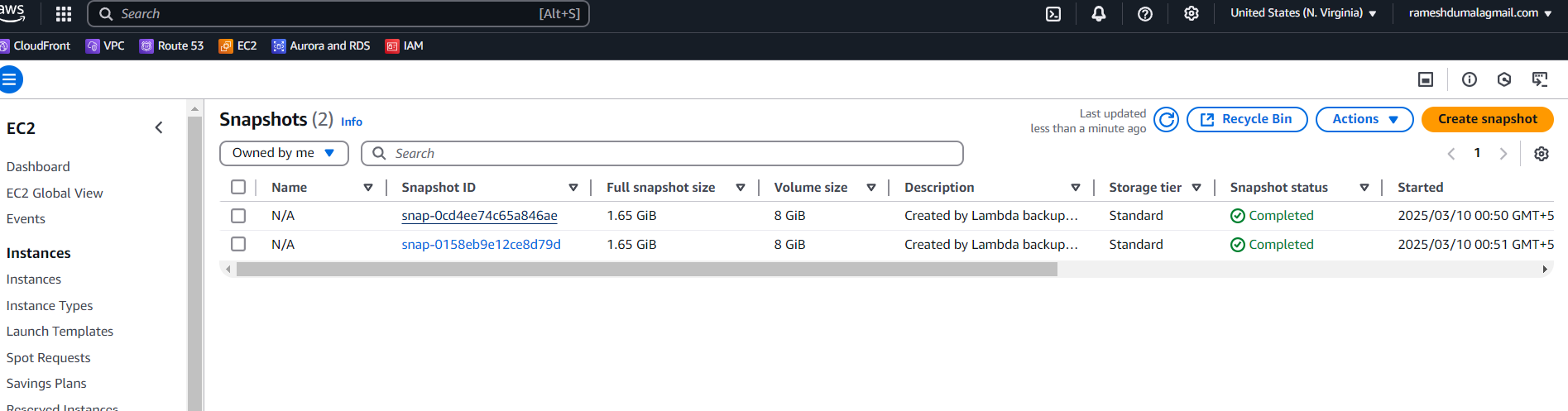
volumename = tags["Value"]

# Add volume name to snapshot for easier identification

snapshot.create\_tags(Tags=[{'Key': 'Name','Value': volumename}])

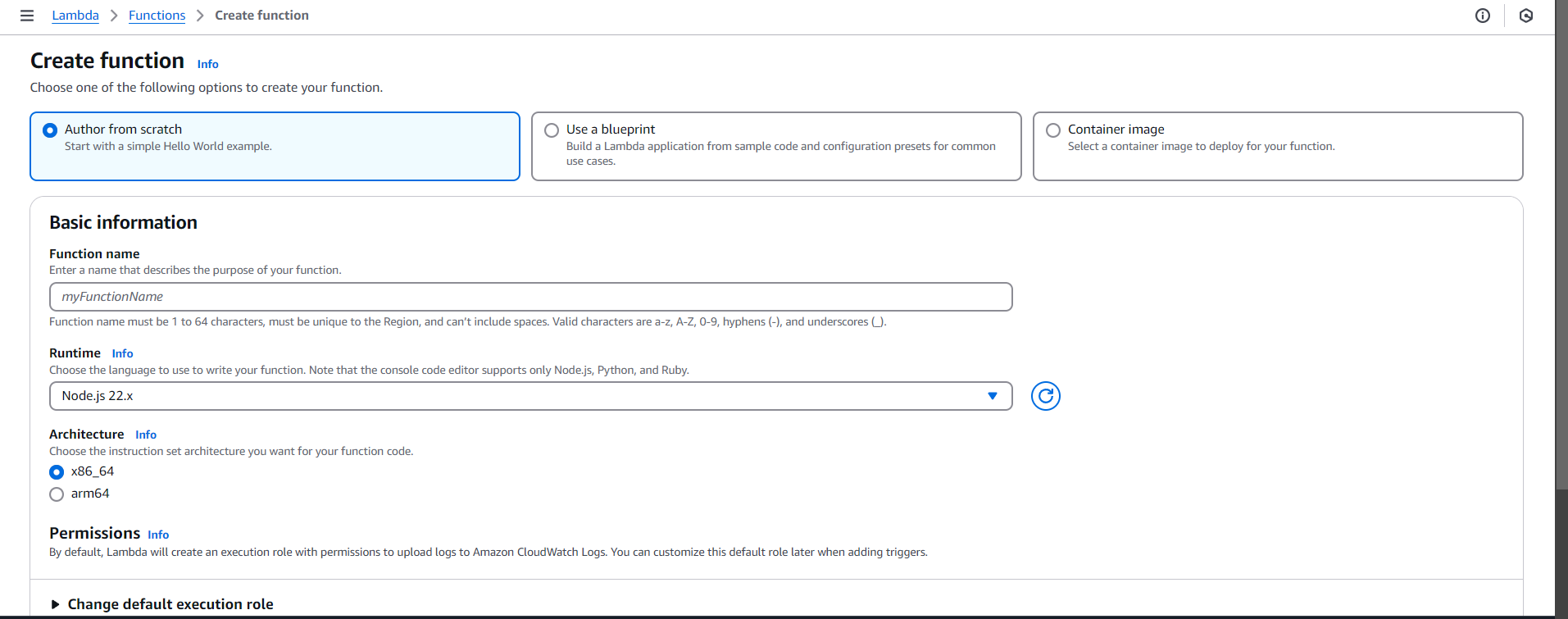
1. Click **Deploy**.

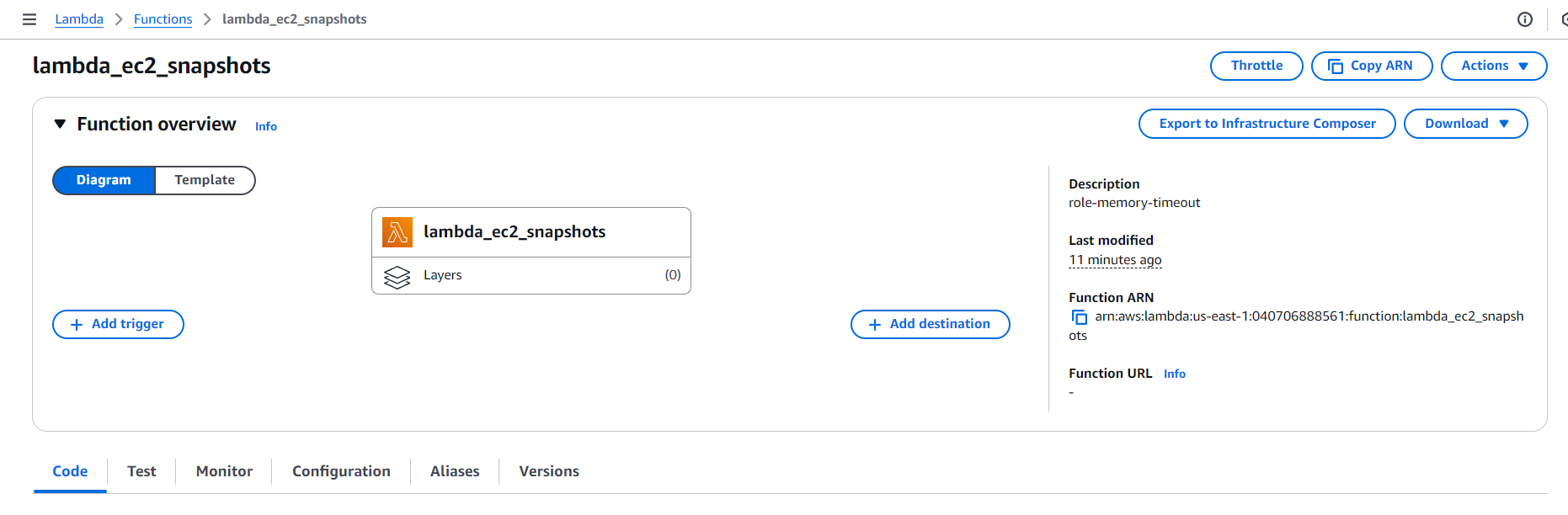
### **Step 4: Test the Function**

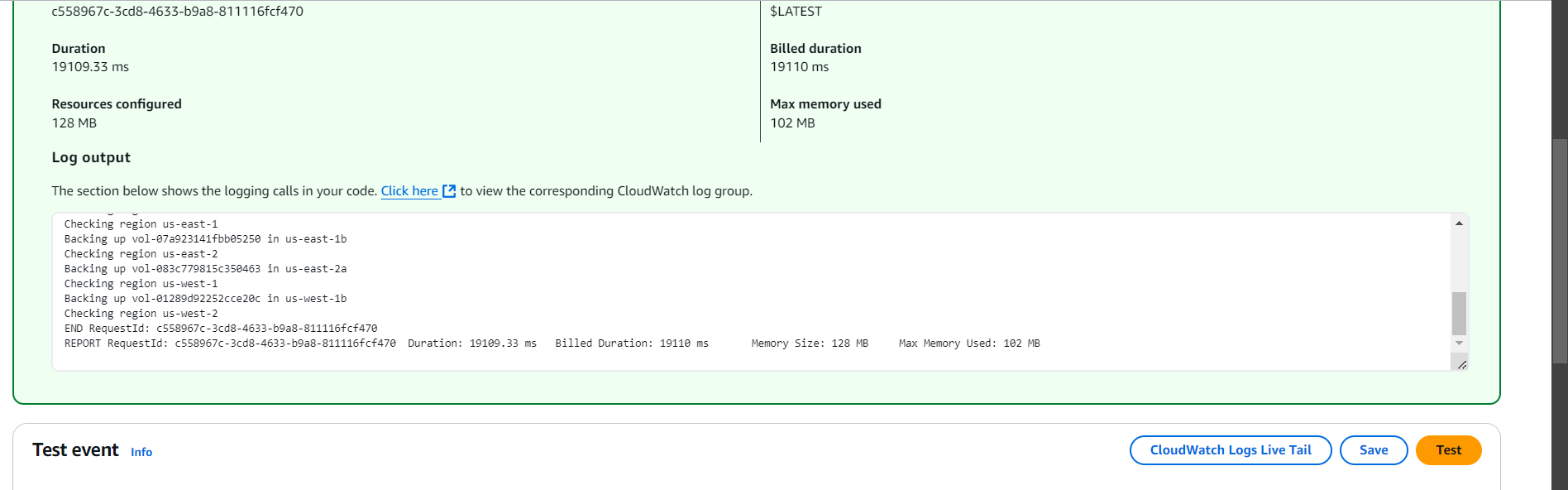


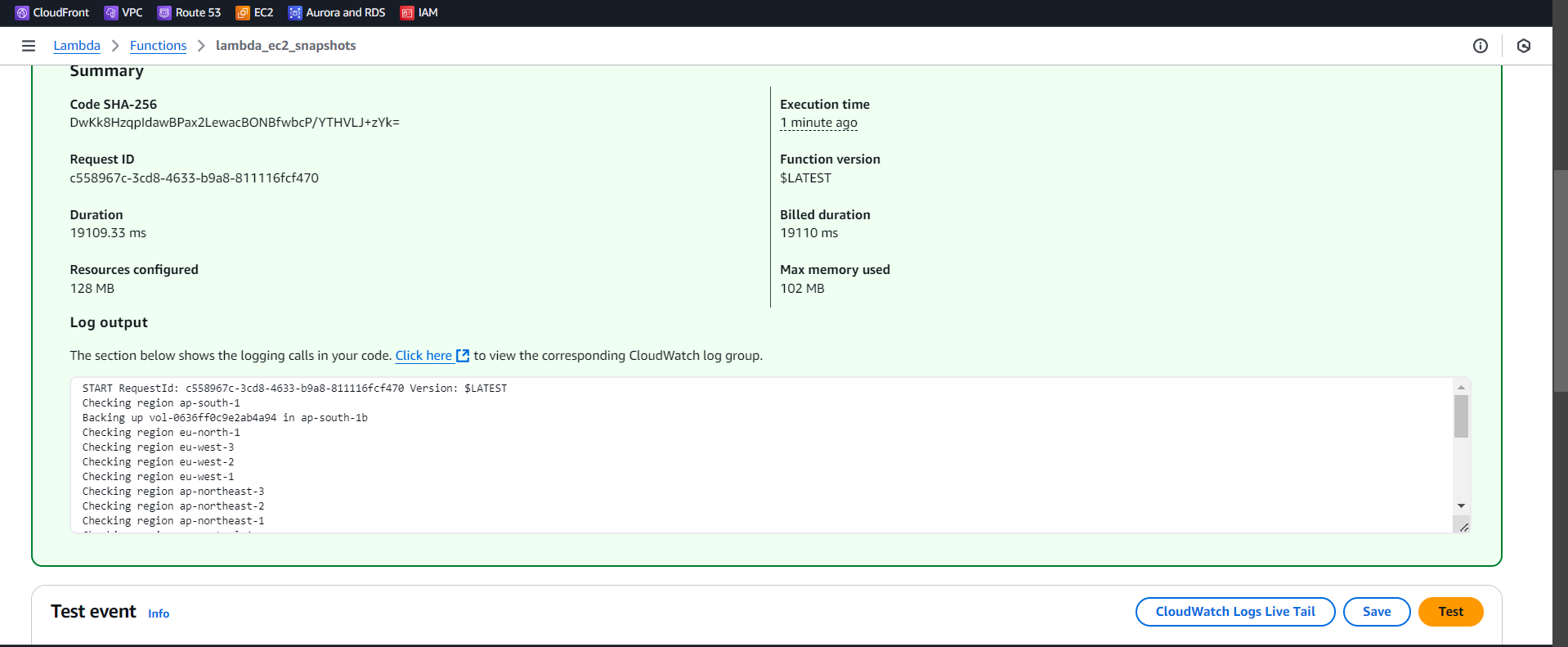
### **Step 5: Monitor Execution with CloudWatch**

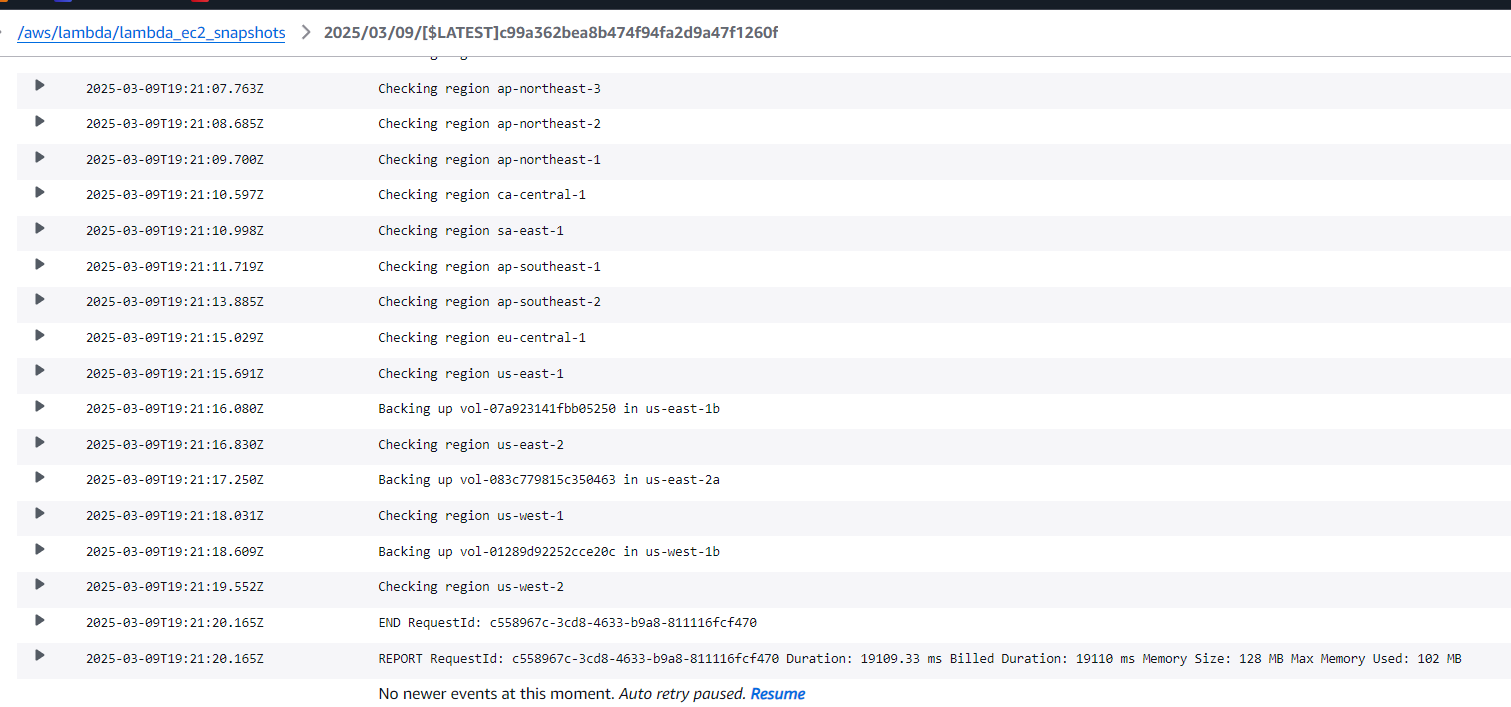
1. Click on the **Monitor** tab.
2. View logs in **Amazon CloudWatch Logs** to analyze execution details.

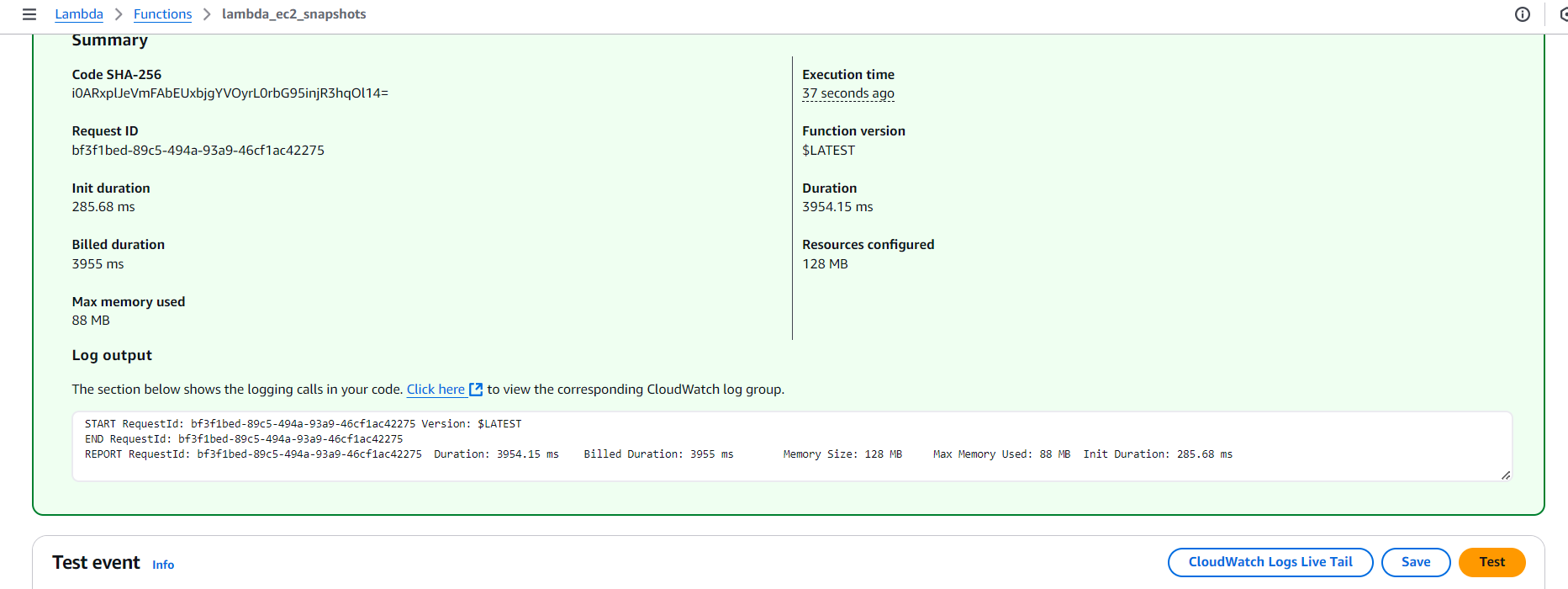


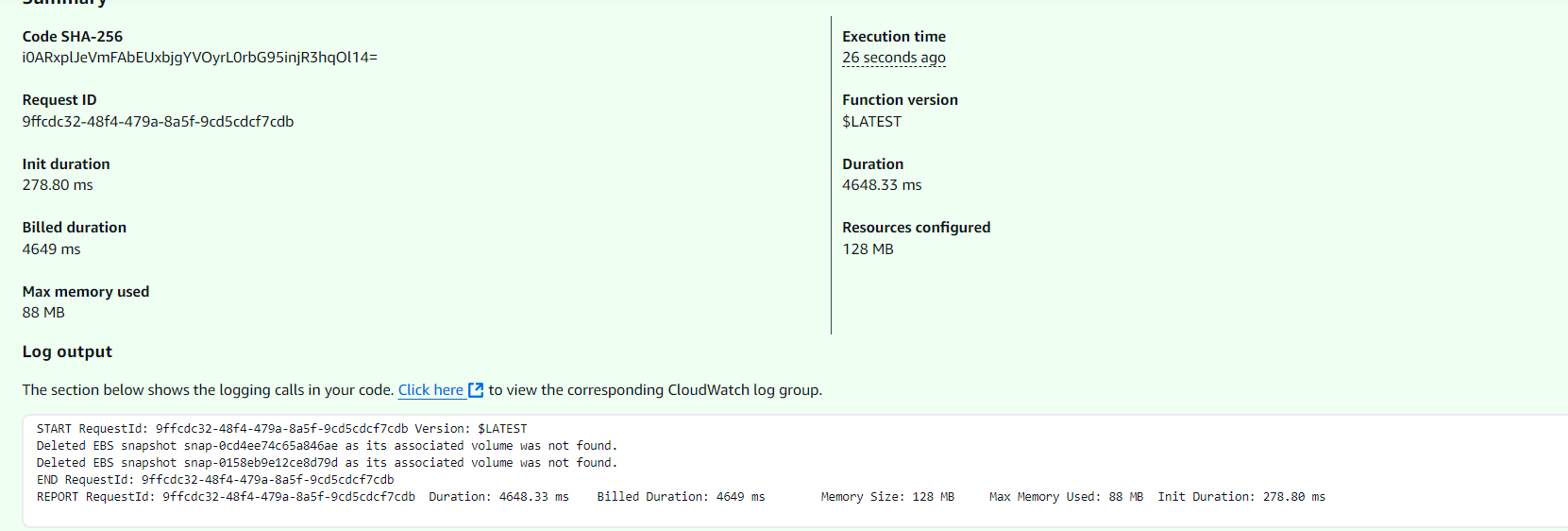


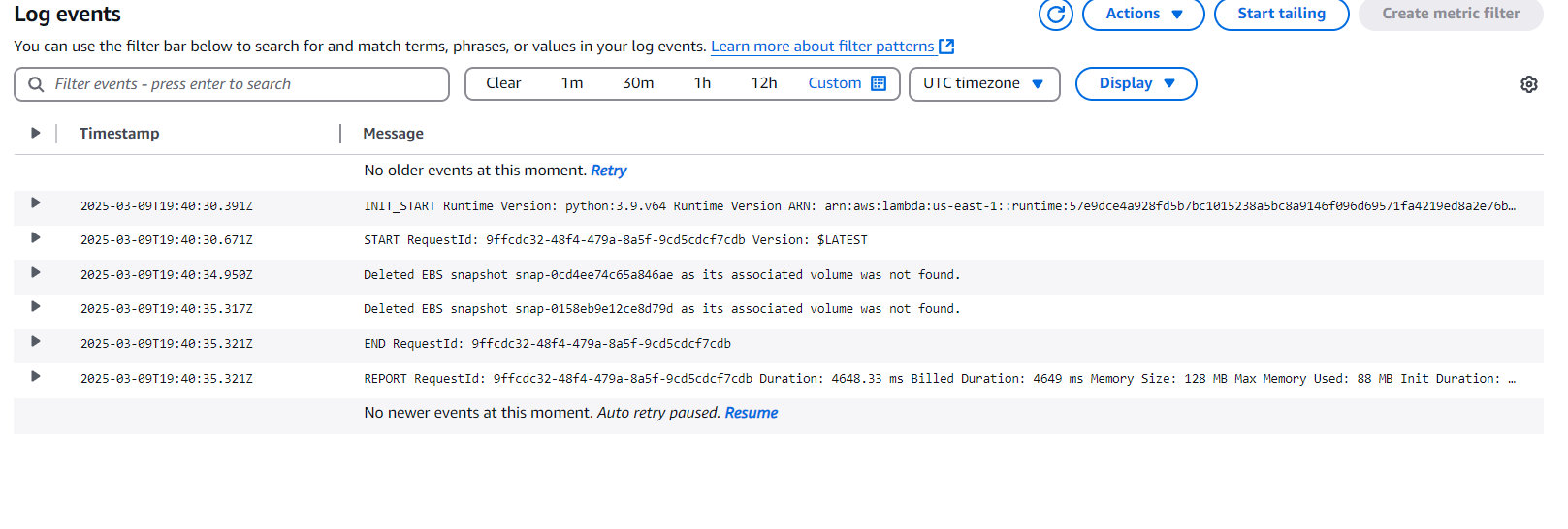












import boto3

def lambda\_handler(event, context):

ec2 = boto3.client('ec2')

# Get all EBS snapshots

response = ec2.describe\_snapshots(OwnerIds=['self'])

# Get all active EC2 instance IDs

instances\_response = ec2.describe\_instances(Filters=[{'Name': 'instance-state-name', 'Values': ['running']}])

active\_instance\_ids = set()

for reservation in instances\_response['Reservations']:

for instance in reservation['Instances']:

active\_instance\_ids.add(instance['InstanceId'])

# Iterate through each snapshot and delete if it's not attached to any volume or the volume is not attached to a running instance

for snapshot in response['Snapshots']:

snapshot\_id = snapshot['SnapshotId']

volume\_id = snapshot.get('VolumeId')

if not volume\_id:

# Delete the snapshot if it's not attached to any volume

ec2.delete\_snapshot(SnapshotId=snapshot\_id)

print(f"Deleted EBS snapshot {snapshot\_id} as it was not attached to any volume.")

else:

# Check if the volume still exists

try:

volume\_response = ec2.describe\_volumes(VolumeIds=[volume\_id])

if not volume\_response['Volumes'][0]['Attachments']:

ec2.delete\_snapshot(SnapshotId=snapshot\_id)

print(f"Deleted EBS snapshot {snapshot\_id} as it was taken from a volume not attached to any running instance.")

except ec2.exceptions.ClientError as e:

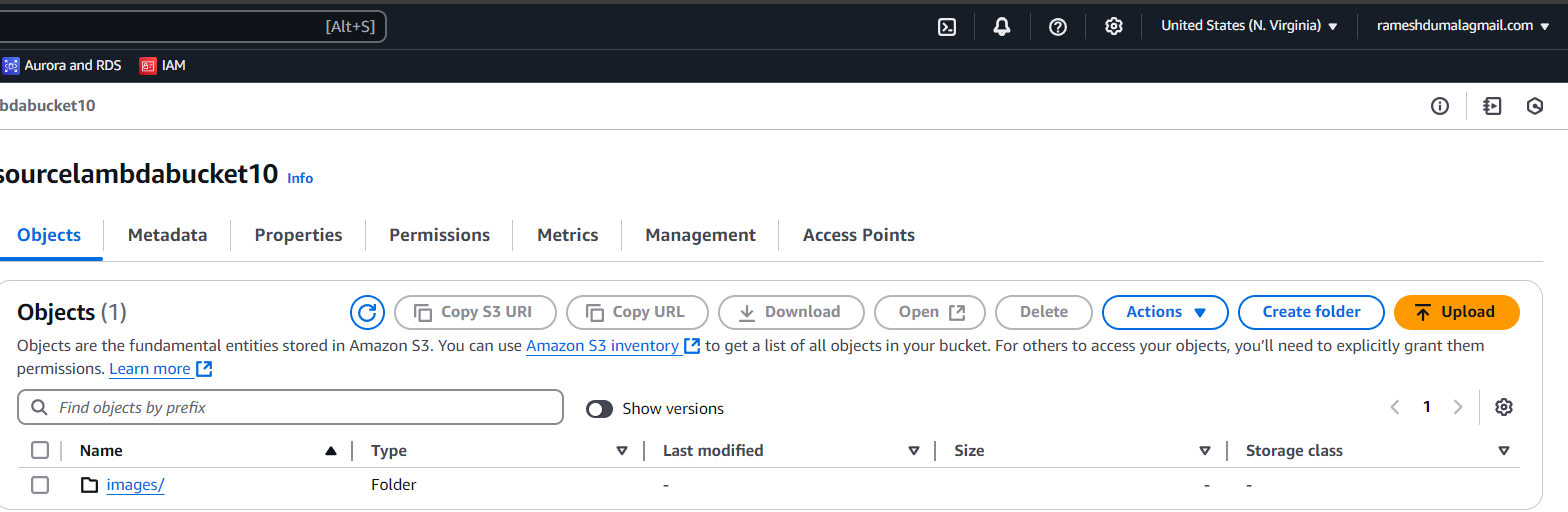
if e.response['Error']['Code'] == 'InvalidVolume.NotFound':

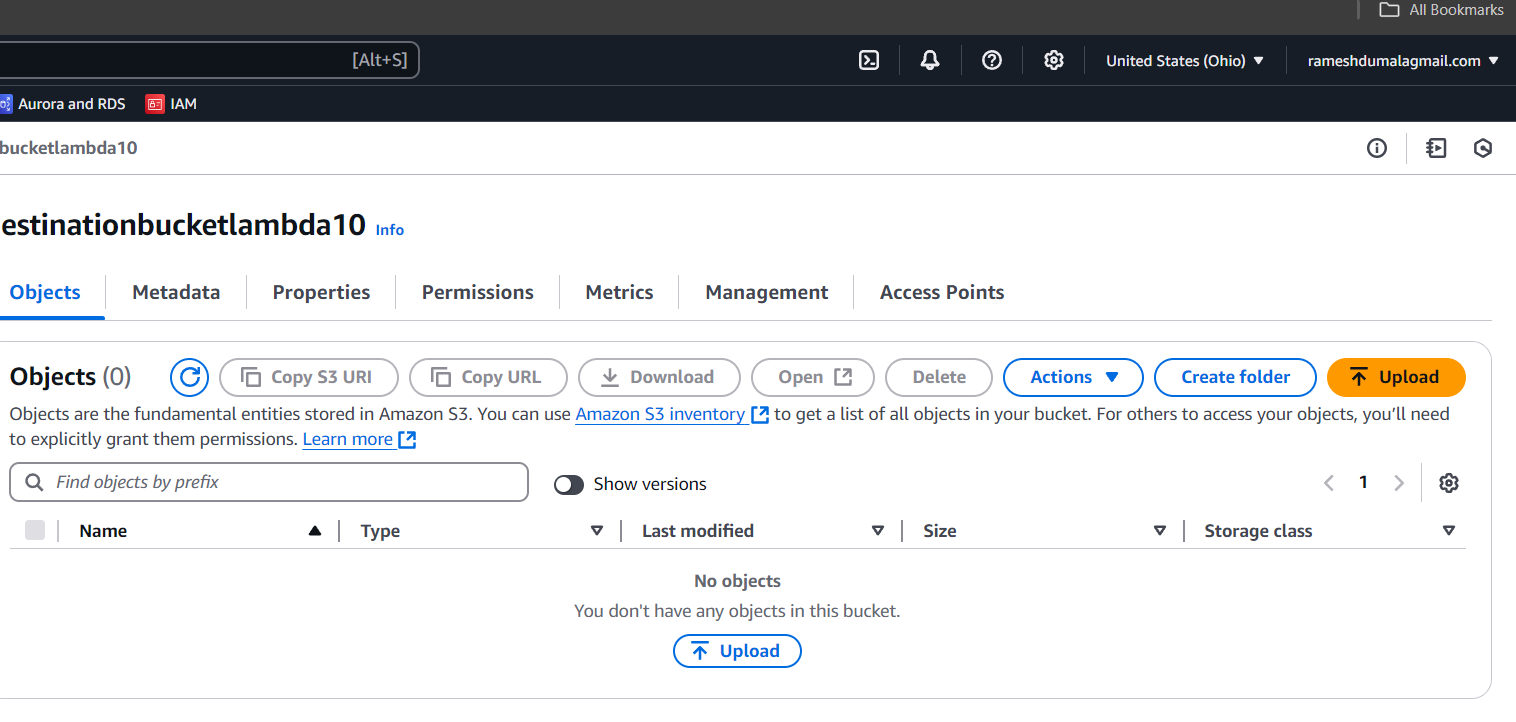
# The volume associated with the snapshot is not found (it might have been deleted)

ec2.delete\_snapshot(SnapshotId=snapshot\_id)

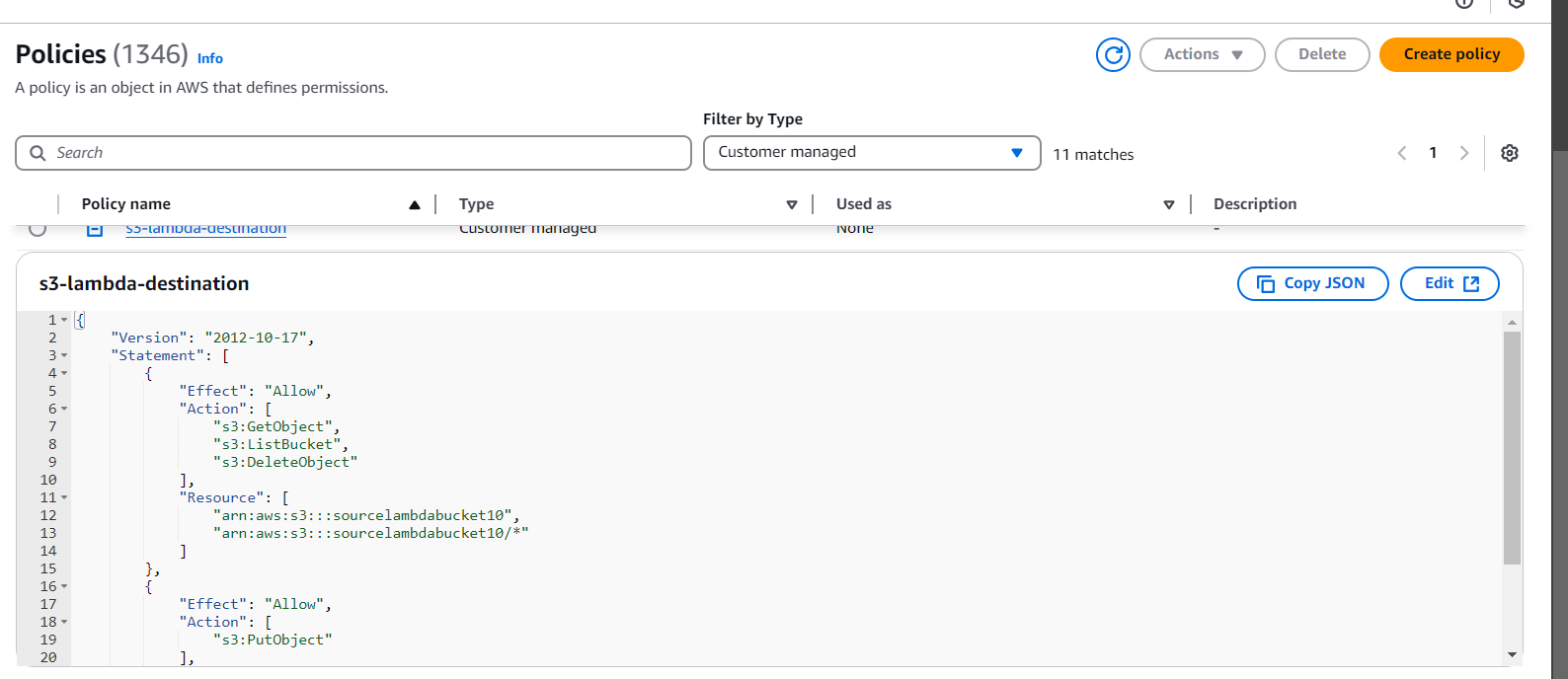
print(f"Deleted EBS snapshot {snapshot\_id} as its associated volume was not found.")

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.



{

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

"Statement": [

{

"Effect": "Allow",

"Action": [

"s3:GetObject",

"s3:ListBucket",

"s3:DeleteObject"

],

"Resource": [

"arn:aws:s3:::sourcelambdabucket10",

"arn:aws:s3:::sourcelambdabucket10/\*"

]

},

{

"Effect": "Allow",

"Action": [

"s3:PutObject"

],

"Resource": [

"arn:aws:s3:::destinationbucketlambda10",

"arn:aws:s3:::destinationbucketlambda10/\*"

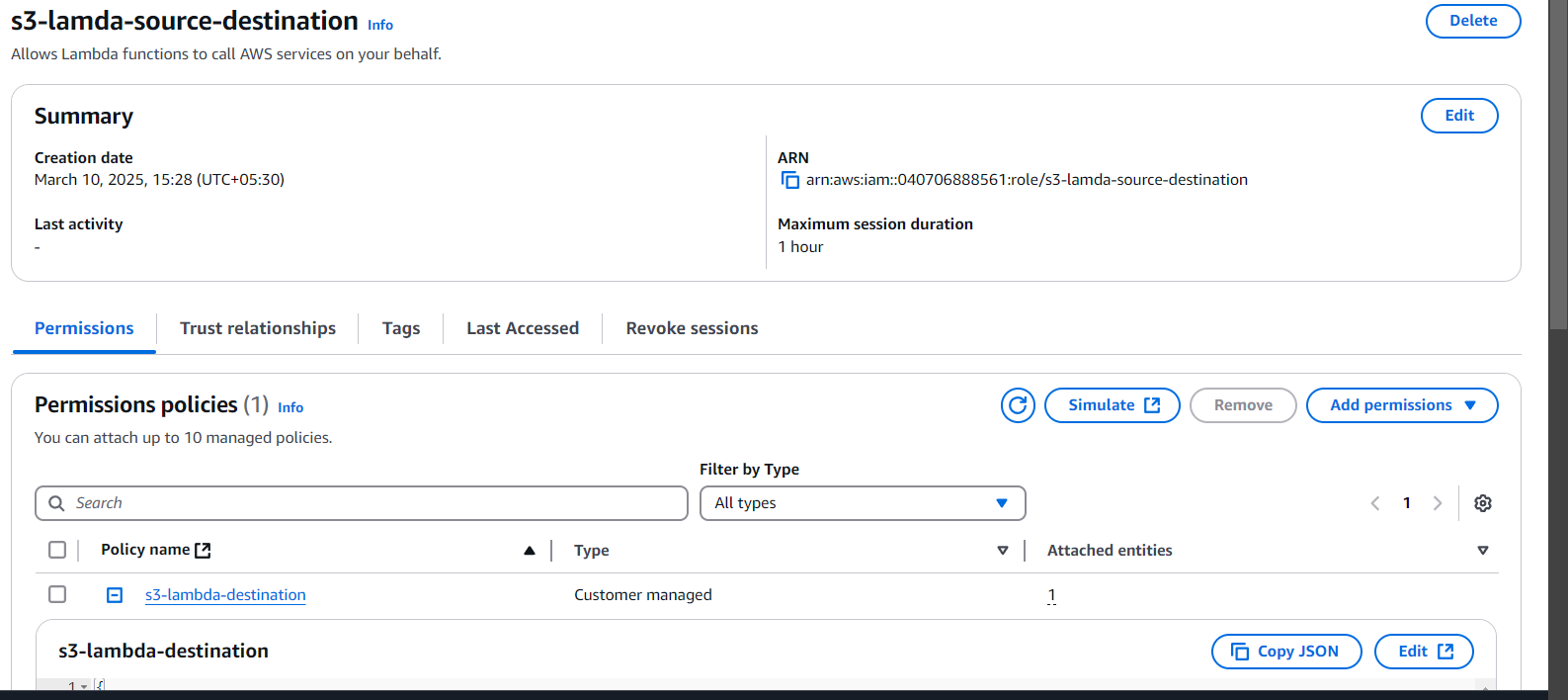
]

}

]

}

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**

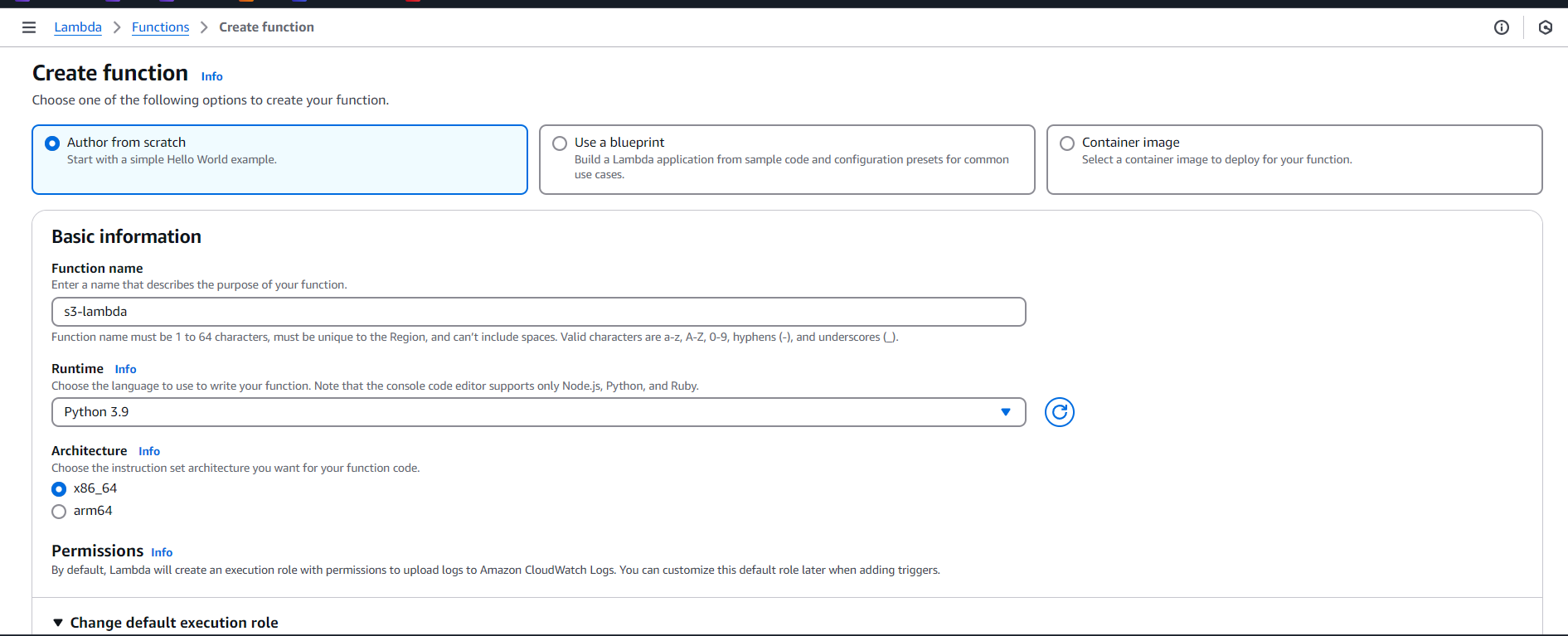
I chose **"Author from scratch."**

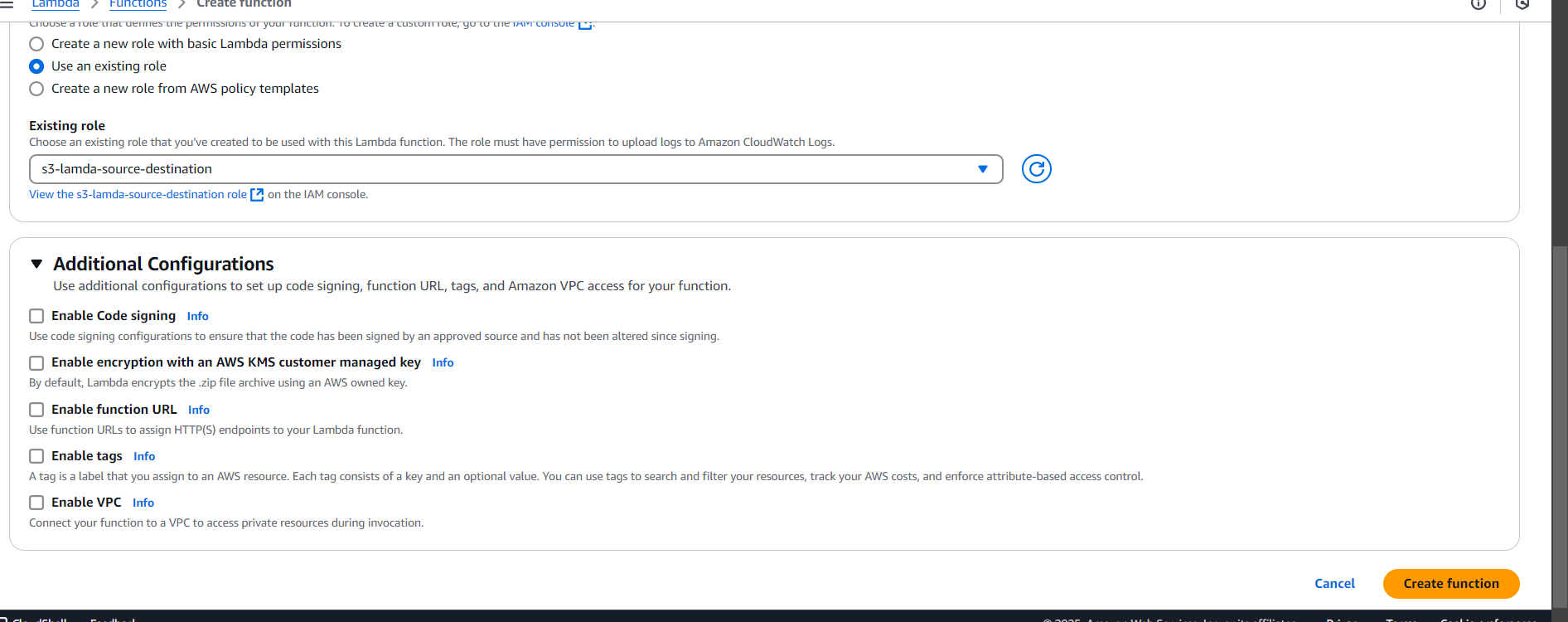
In the runtime settings, I select **Python** as the programming language.

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

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

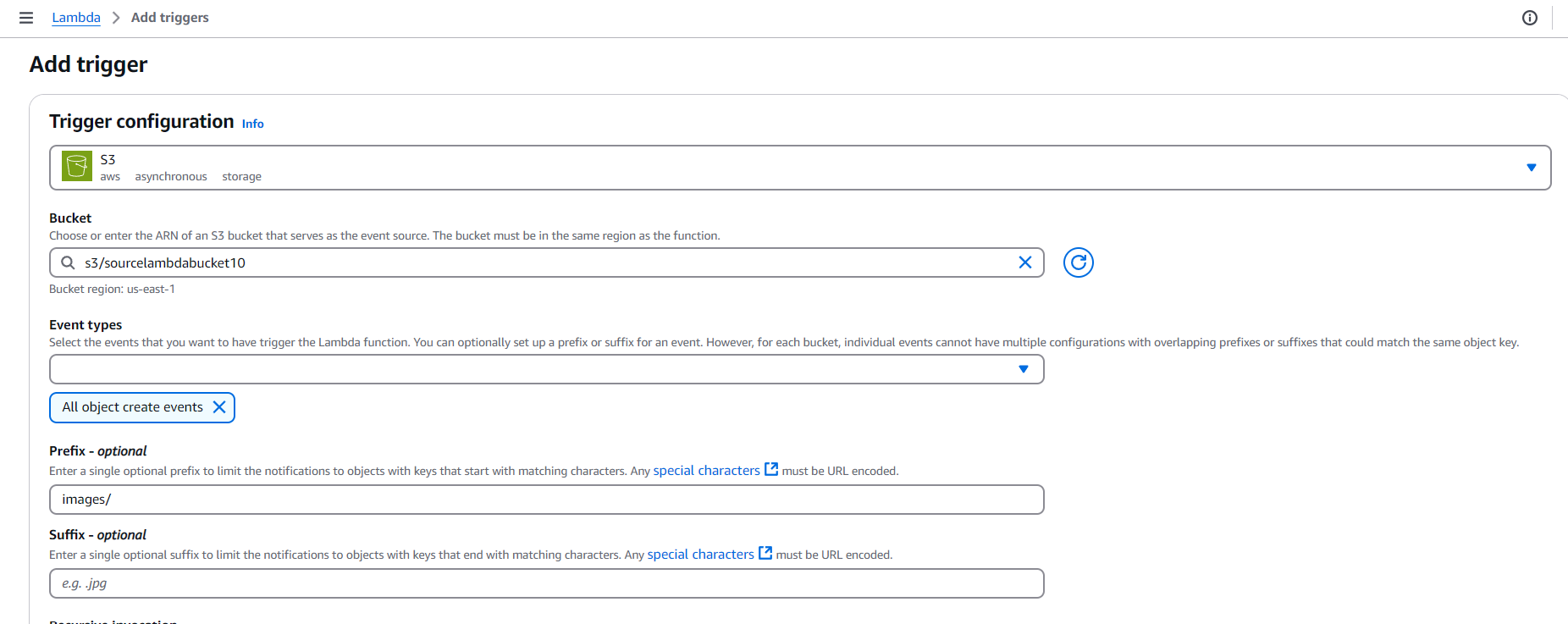
**So, the Lambda function is created, and the development environment is displayed below.**





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

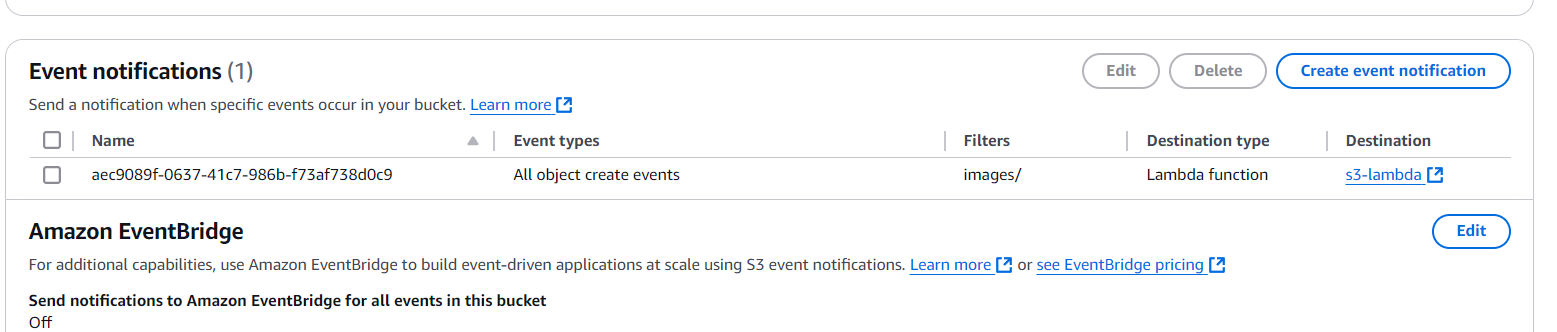


1. I select the event type based on the requirement, like **"All object create events."**

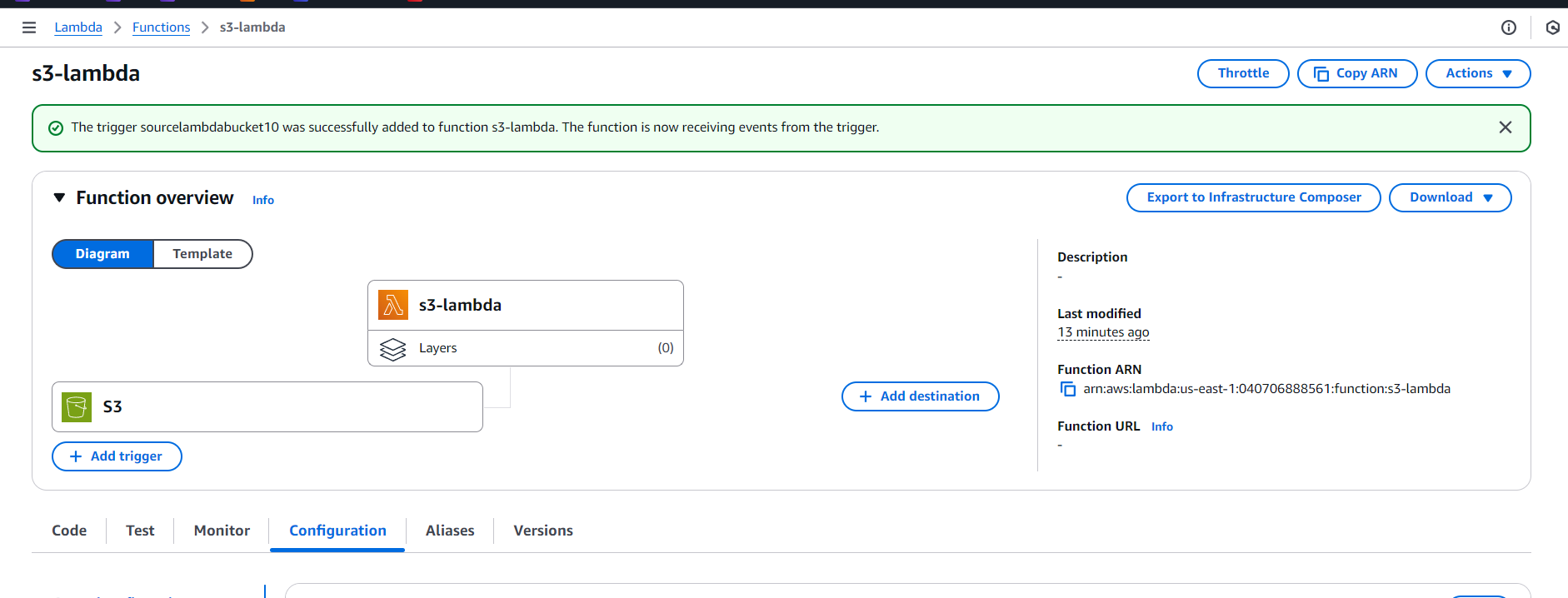
2. If the **prefix** is empty, it includes all objects. But I set it to **"images/"** so only files in the **images folder** will be sent to the destination.

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.



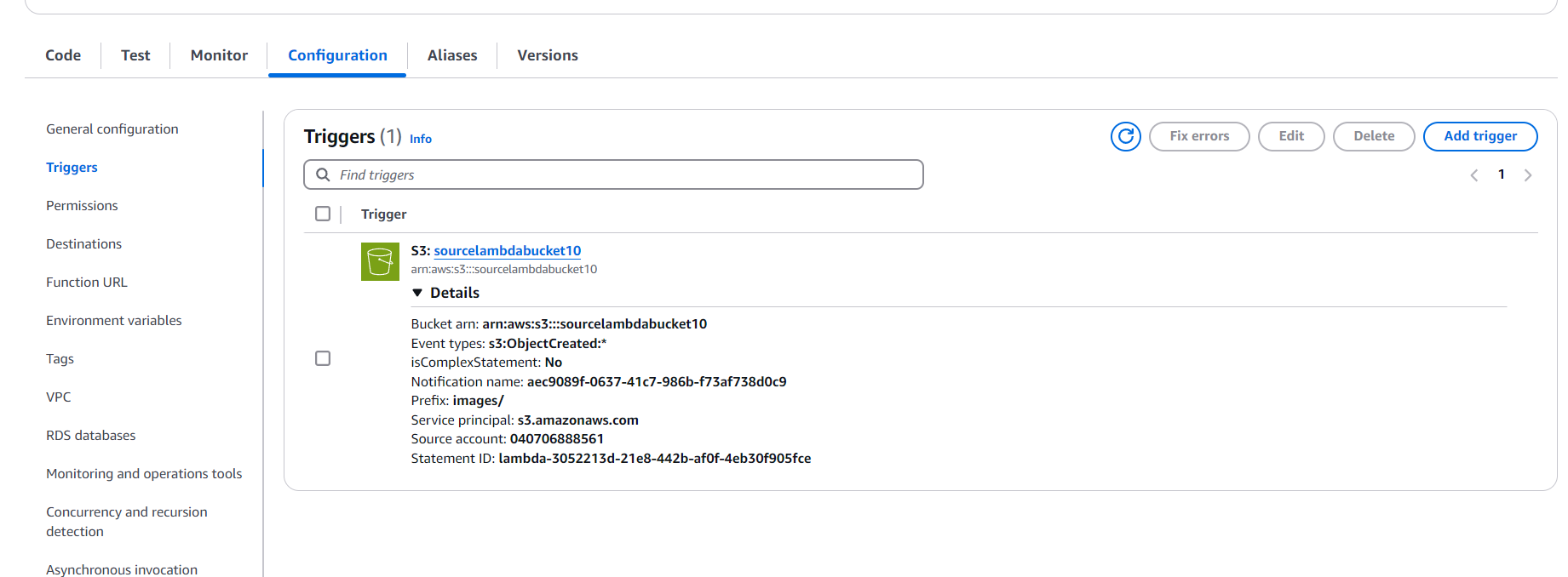
In the above process, when we upload an object to S3, the event is sent to Lambda, and Lambda receives it. However, **Lambda doesn’t automatically know what action to perform**—we need to write code to define the action.



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.



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 = "arn:aws:s3:::sourcelambdabucket10"

DESTINATION\_BUCKET = "arn:aws:s3:::destinationbucketlambda10"

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 {

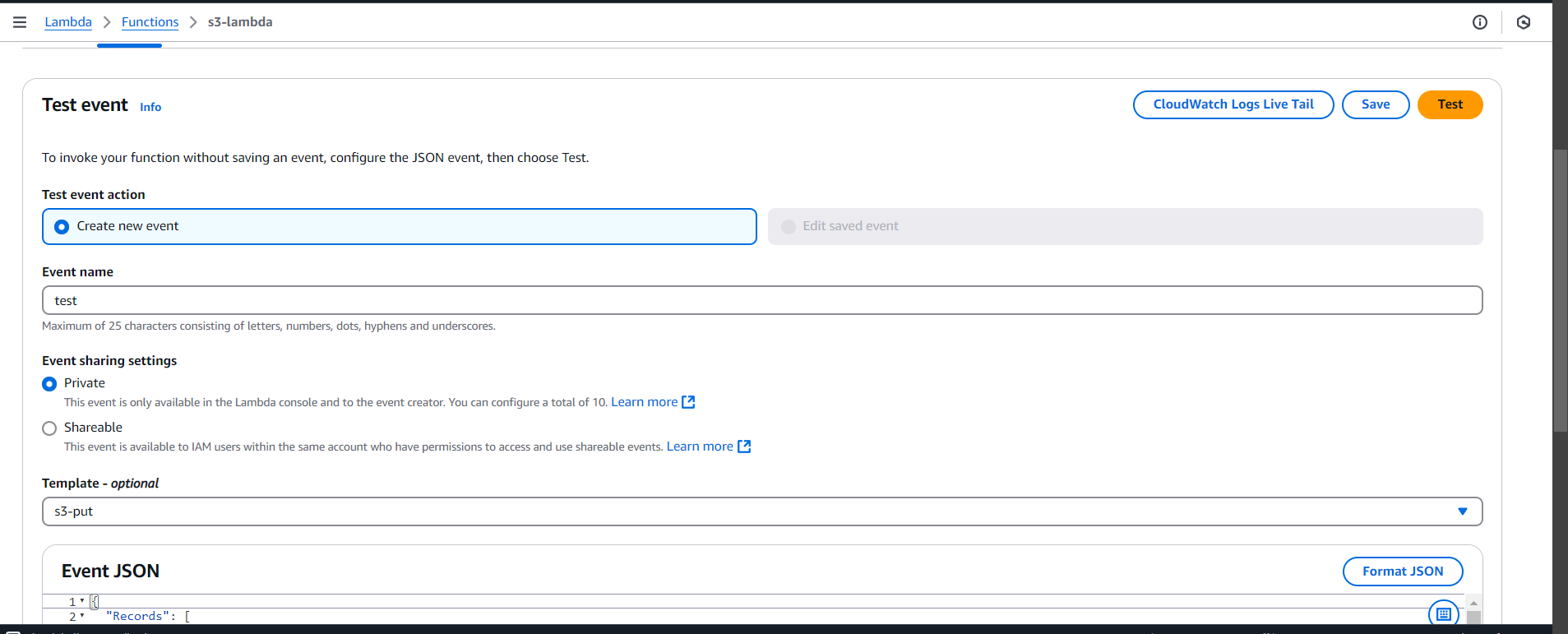
"statusCode": 500,

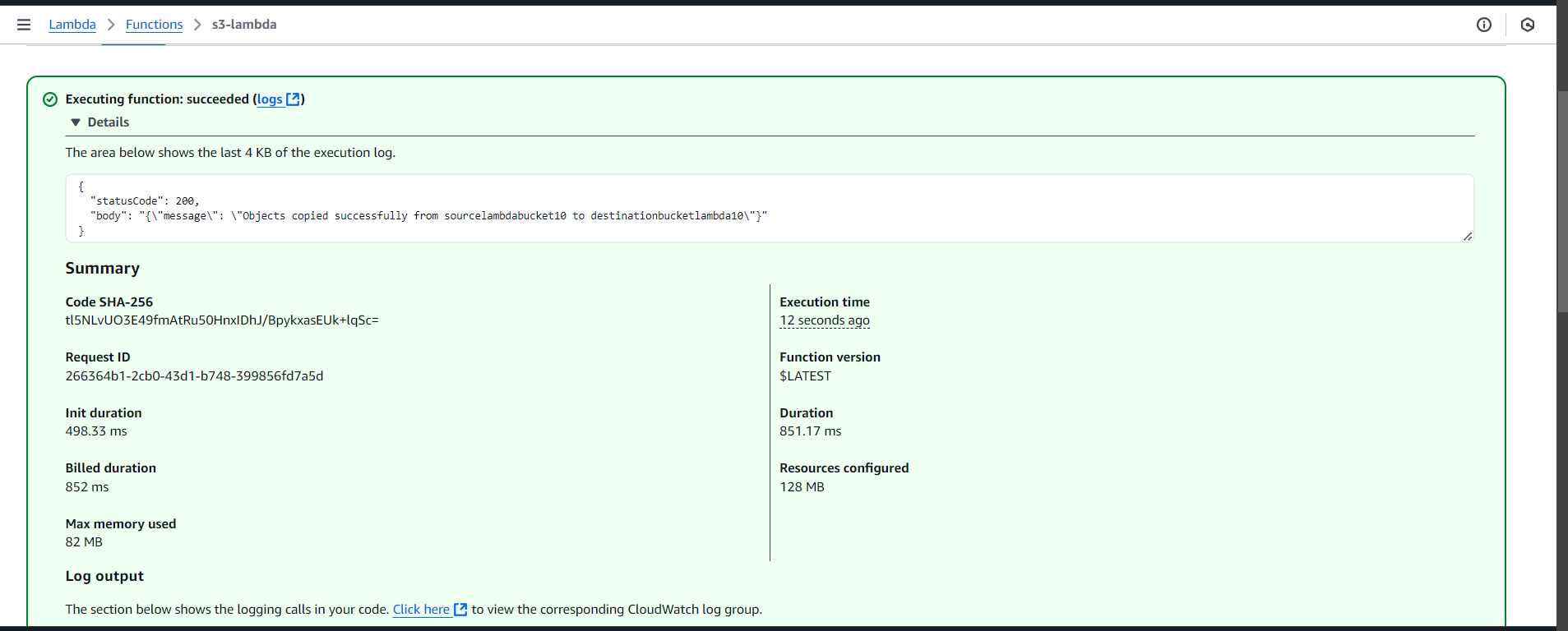
"body": json.dumps({

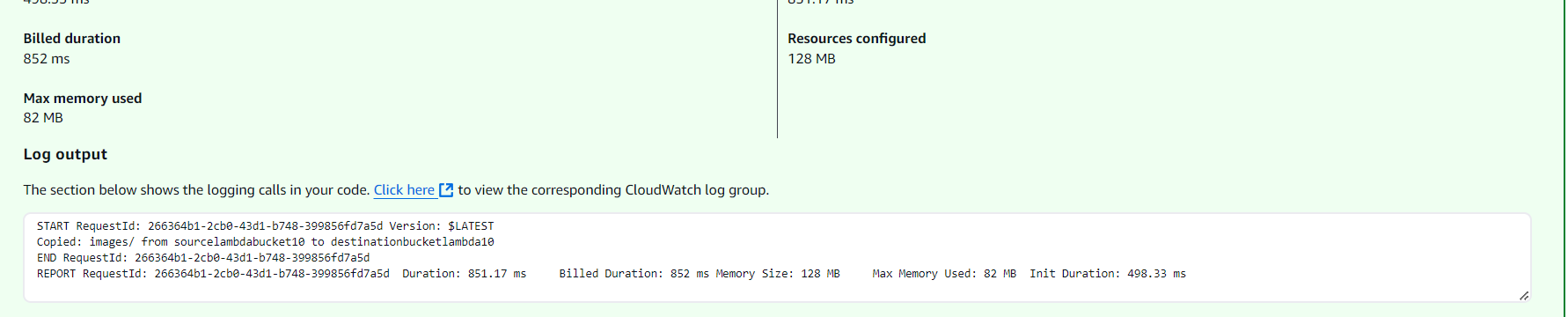
"error": f"Error copying objects: {str(e)}"

})

}



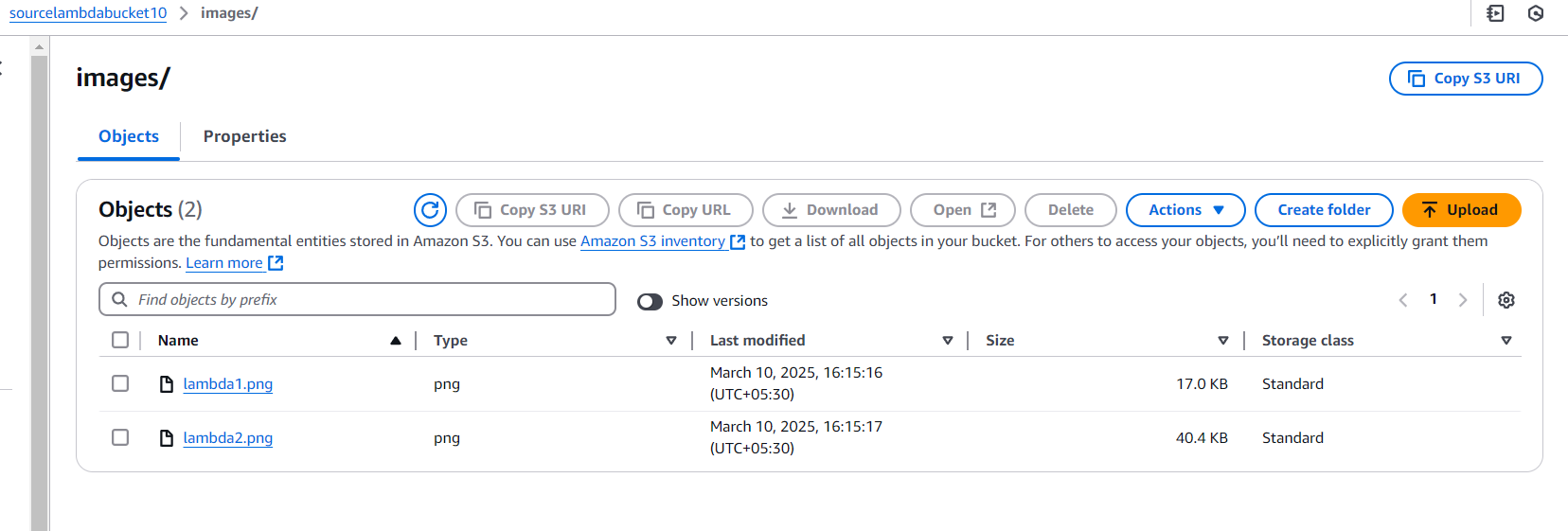


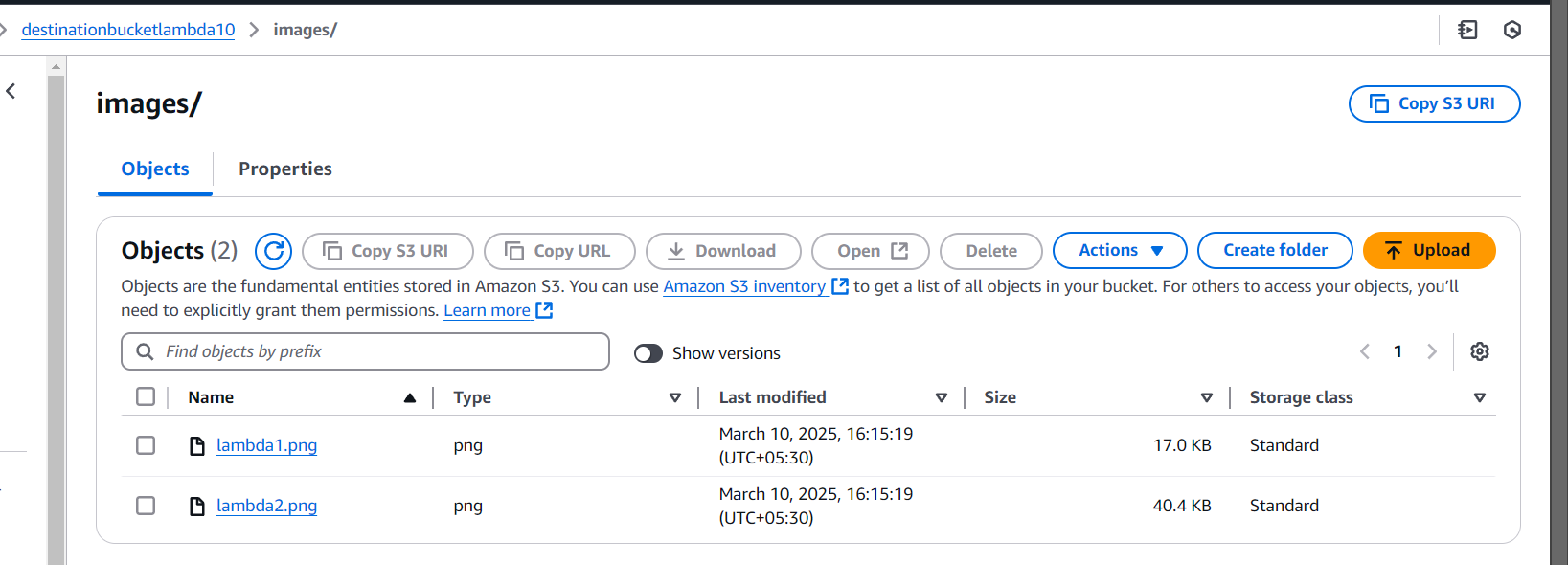


After deploying (saving) the code, if you want to test it, **create a test event**.

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

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





## **Conclusion**

* AWS Lambda provides a powerful serverless environment.
* Optimize execution time to reduce costs.
* Integrate with various AWS services for automation.
* Monitor performance and improve efficiency.