

## Lab: S3 Lambda

- 1) Create two buckets with default configuration
  - a. Amit23comp
  - b. Amit23comp-replica
- 2) Create a new policy in iam , Change source and destination bucket name using arn

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "putObject",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::awsbucketamit-replica/*"
      ]
    },
    {
      "Sid": "getObject",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::myamit23bucket/*"
      ]
    }
  ]
}
```

Create a role

Select trusted entity [Info](#)

Trusted entity type

☒ **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.

Select your policy

- Step 1
- Select trusted entity
- Step 2
- Add permissions**
- Step 3
- Name, review, and create

Add permissions [Info](#)

**Permissions policies (1/1035)** [Info](#)

Choose one or more policies to attach to your new role.

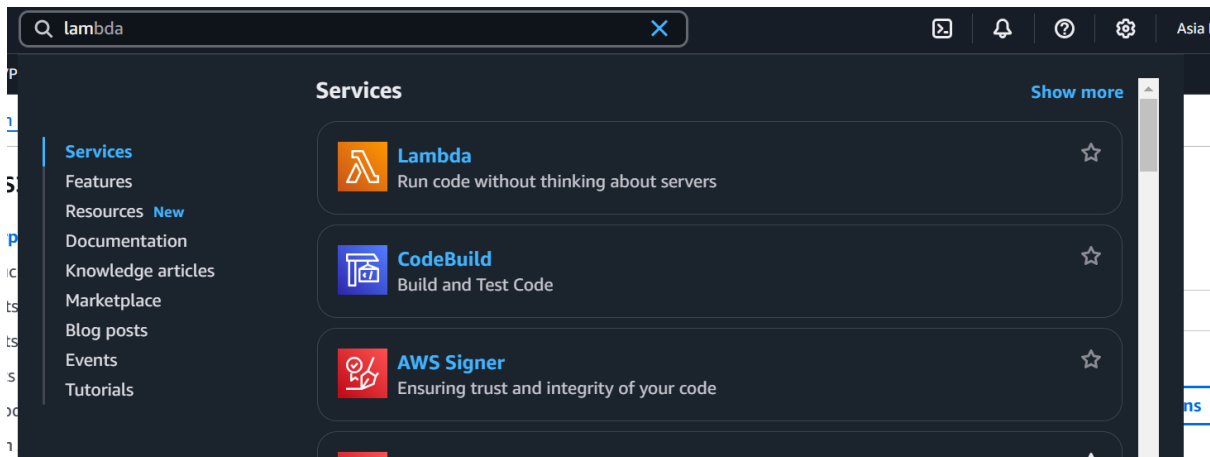
Filter by Type

All types

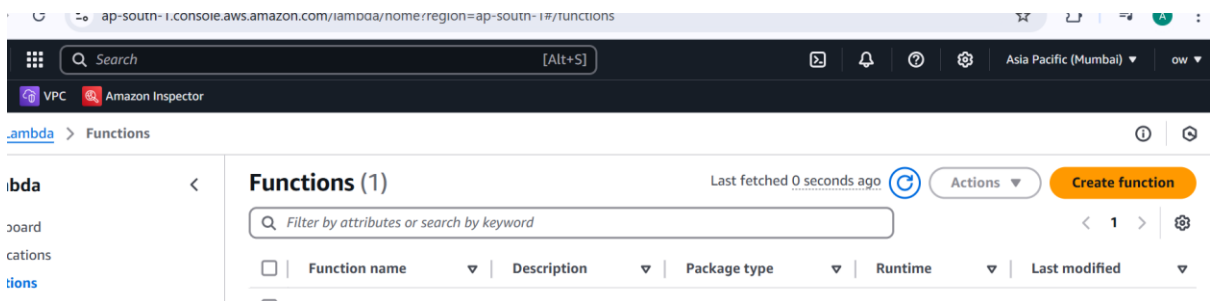
 7 matches < 1 >

| <input type="checkbox"/>            | Policy name <a href="#">↗</a> | Type             | Description |
|-------------------------------------|-------------------------------|------------------|-------------|
| <input checked="" type="checkbox"/> | <a href="#">mylambda</a>      | Customer managed | -           |
| <input type="checkbox"/>            | <a href="#">mypolicy</a>      | Customer managed | -           |

Check for lambda service



## Create a function



## Provide function name and python3.13

[Lambda](#) > [Functions](#) > Create function

### Create function Info

Choose one of the following options to create your function.

☒ **Author from scratch**  
Start with a simple Hello World example.

☐ **Use a blueprint**  
Build a Lambda application from sample code and configuration presets for common use cases.

☐ **Container image**  
Select a container image to deploy for your function.

---

#### Basic information

**Function name**  
Enter a name that describes the purpose of your function.

Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (\_).

**Runtime** Info  
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

## Provide the role

### ▼ Change default execution role

#### Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

- ☐ Create a new role with basic Lambda permissions
- ☒ Use an existing role
- ☐ Create a new role from AWS policy templates

#### Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

mylamda

[View the mylamda role](#) on the IAM console.

Copy the code

Code

Test

Monitor

Configuration

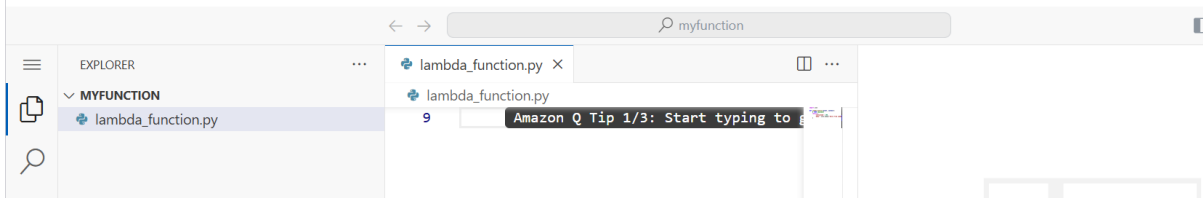
Aliases

Versions

### Code source

Info

Upload from



```
import boto3
import botocore
import os
import logging

# Set up logging
logger = logging.getLogger()
logger.setLevel(logging.INFO)

# Initialize S3 resource
s3 = boto3.resource('s3')

def lambda_handler(event, context):

    logger.info("New files uploaded to the source bucket.")

    # Retrieve the key and bucket name from the event
    try:
        key = event['Records'][0]['s3']['object']['key']
        source_bucket = event['Records'][0]['s3']['bucket']['name']
    except KeyError as e:
        logger.error("Missing key information in event: %s", e)
        return

    # Get the destination bucket from environment variables
    destination_bucket = os.getenv('destination_bucket')

    if not destination_bucket:
        logger.error("Destination bucket environment variable is not set.")
        return

    # Define the source object
    source = {'Bucket': source_bucket, 'Key': key}

    try:
        # Copy the file
        s3.meta.client.copy(source, destination_bucket, key)

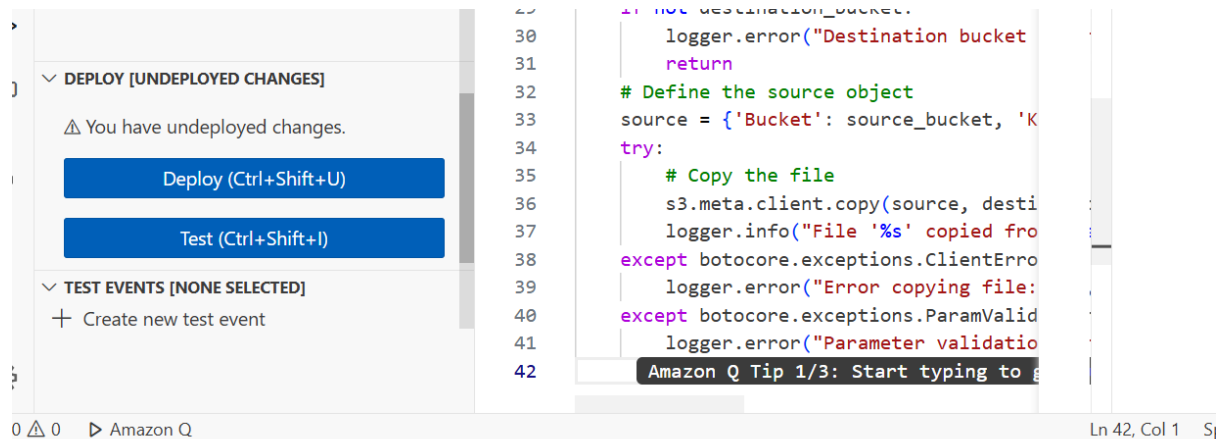
        logger.info("File '%s' copied from '%s' to '%s' successfully!", key, source_bucket,
            destination_bucket)
    except botocore.exceptions.ClientError as error:
```

```
logger.error("Error copying file: %s", error)

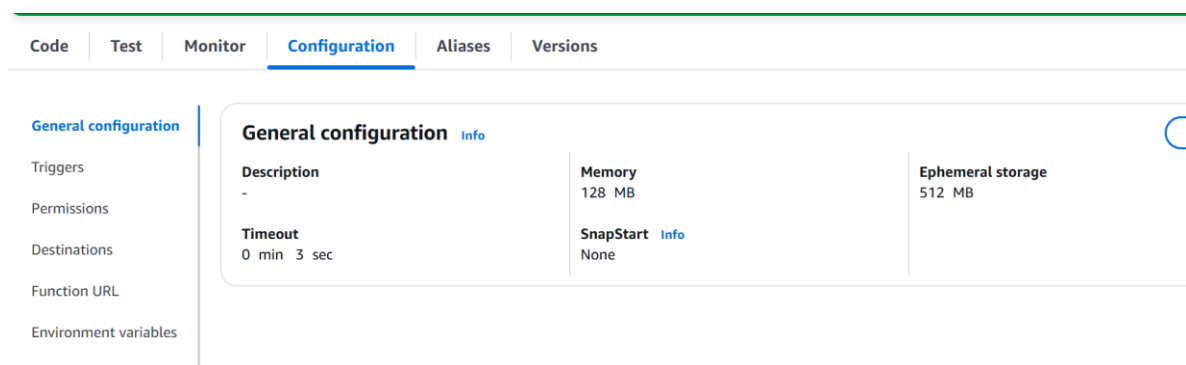
except botocore.exceptions.ParamValidationError as error:

    logger.error("Parameter validation error: %s", error)
```

Click on Deploy



Click on Configuration



Click on Environment variables

Key :destination\_bucket

Value: awsbucketamit-replica

## 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                 |                         |
|--------------------|-----------------------|-------------------------|
| destination_bucket | awsbucketamit-replica | <button>Remove</button> |

Add environment variable

► Encryption configuration

CancelSave


Click on Add triggers


## myfunction

**Function overview** [Info](#)

Diagram

Template


 **myfunction**


 Layers (0)

+ Add trigger

+ Add destination

**Trigger configuration** [Info](#)

 **S3**  
aws asynchronous storage

**Bucket**  
Choose or enter the ARN of an S3 bucket that serves as the event source. The bucket must be in the same region as the function.  
 X   
Bucket region: ap-south-1

**Event types**  
Select the events that you want to have trigger the Lambda function. You can optionally set up a prefix or suffix for an event. However, for each bucket, individual events cannot have multiple overlapping prefixes or suffixes that could match the same object key.  

All object create events X

PUT X

POST X

COPY X

Multipart upload completed X

**Prefix - optional**  
Enter a single optional prefix to limit the notifications to objects with keys that start with matching characters. Any [special character](#) must be URL encoded.

Upload a jpg file in source bucket.

Create a test.

[Code](#) [Test](#) [Monitor](#) [Configuration](#) [Aliases](#) [Versions](#)

Test event [Info](#)

CloudWatch Logs Live Tail Save Test

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

mytest

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

Json

```
{
  "Records": [
    {
      "s3": {
        "bucket": {
          "name": "provide the source bucket name"
        },
        "object": {
          "key": "object name which you uploaded"
        }
      }
    }
  ]
}
```

Save and Test



# Updated and Rectified Lab Steps for S3 Lambda

## 1. Create Two Buckets:

Create two S3 buckets with default configuration:

- Source: myamit23bucket
- Destination: awsbucketamit-replica

## 2. Create an IAM Policy:

Go to IAM → Policies → Create Policy and paste the JSON (update bucket ARNs accordingly).

## 3. Create an IAM Role:

Go to IAM → Roles → Create Role. Choose Lambda as trusted entity, attach the policy created above.

## 4. Create a Lambda Function:

Name it s3CopyFunction, select Python 3.13, and attach the IAM role.

## 5. Paste the Lambda Code:

Use the provided Python code that uses boto3 to copy from source to destination bucket.

## 6. Deploy the Function:

Click Deploy to apply the code changes.

## 7. Add Environment Variable:

Go to Configuration → Environment variables:

- Key: destination\_bucket
- Value: awsbucketamit-replica

## 8. Add S3 Trigger:

Attach a trigger for PUT events from source bucket.

## 9. Upload Test File:

Upload a file to the source bucket to trigger the function.

## 10. Create Test Event:

Manually test with a JSON simulating an S3 PUT event using the file key.

## 11. Rectification Steps:

- Verify logs in CloudWatch
- Check destination bucket for copied file
- Validate error handling for missing env or bad bucket name