

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

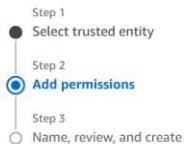
Choose a use case for the specified service.

Use case

Lambda

Allows Lambda functions to call AWS services on your behalf.

Select your policy



Add permissions [Info](#)

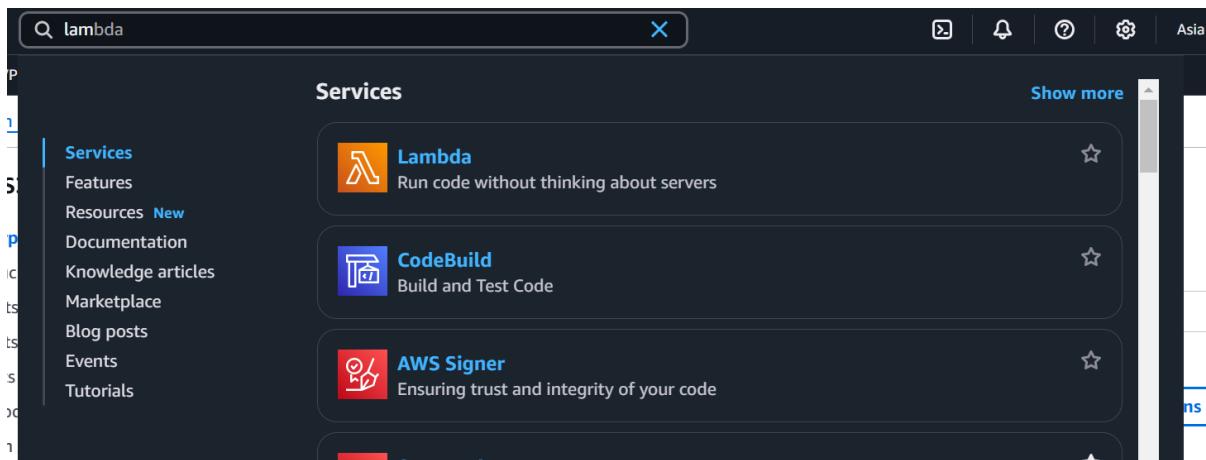
Permissions policies (1/1035) [Info](#)

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

Filter by Type

<input type="text"/> my	<input type="button"/>	All types	7 matches
<input type="checkbox"/> Policy name <input type="text"/>	<input type="button"/>	Type	<input type="button"/>
<input checked="" type="checkbox"/> <input type="checkbox"/> mylambda	<input type="button"/>	Customer managed	-

Check for lambda service



Create a function

A screenshot of the AWS Lambda Functions list page. The URL is "ap-south-1.console.aws.amazon.com/lambda/home?region=ap-south-1#/functions". The top navigation bar includes "Search [Alt+S]", "Actions", "Create function", and "Asia Pacific (Mumbai)". The main area shows a table titled "Functions (1)" with one item. The table columns are "Function name", "Description", "Package type", "Runtime", and "Last modified". A "Filter by attributes or search by keyword" input field is also present.

Provide function name and python3.13

A screenshot of the "Create function" wizard. The top navigation bar shows "Lambda > Functions > Create function". The main section is titled "Create function" with an "Info" link. It asks to choose an option: "Author from scratch" (selected), "Use a blueprint", or "Container image". Below this is a "Basic information" section with "Function name" set to "myfunction". The "Runtime" section shows "Python 3.13" selected. A note says "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](#)

EXPLORER ...

MYFUNCTION

lambda_function.py

lambda_function.py

Amazon Q Tip 1/3: Start typing to g...

```
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:
```

```

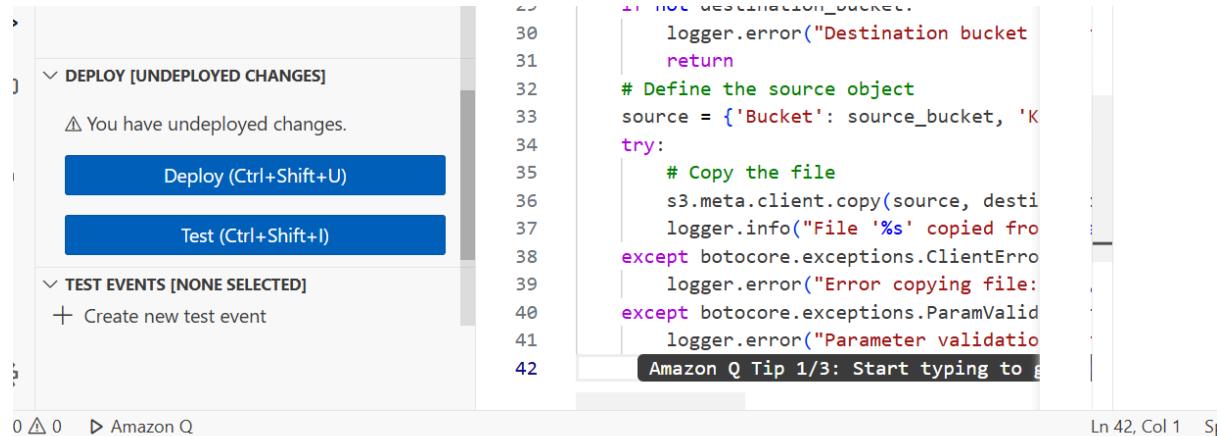
logger.error("Error copying file: %s", error)

except botocore.exceptions.ParamValidationError as error:

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

```

Click on Deploy



The screenshot shows the AWS Lambda function editor interface. On the left, there's a sidebar with sections for 'DEPLOY [UNDEPLOYED CHANGES]' (containing a warning about undeployed changes) and 'TEST EVENTS [NONE SELECTED]'. Below these are buttons for 'Deploy (Ctrl+Shift+U)' and 'Test (Ctrl+Shift+I)'. The main area is a code editor with the following Python code:

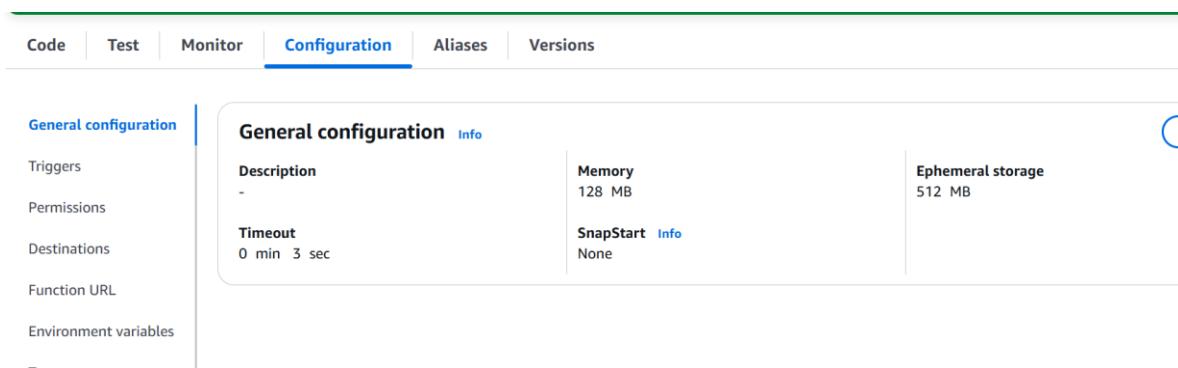
```

1  if not destination_bucket:
2      logger.error("Destination bucket not found")
3      return
4
5  # Define the source object
6  source = {'Bucket': source_bucket, 'Key': key}
7
8  try:
9      # Copy the file
10     s3.meta.client.copy(source, destination_bucket, key)
11     logger.info("File '%s' copied from %s to %s", key, source['Bucket'], destination_bucket)
12
13 except botocore.exceptions.ClientError as error:
14     logger.error("Error copying file: %s", error)
15 except botocore.exceptions.ParamValidationError as error:
16     logger.error("Parameter validation error: %s", error)

```

At the bottom right of the code editor, there's a tip: 'Amazon Q Tip 1/3: Start typing to get started'.

Click on Configuration



The screenshot shows the 'Configuration' tab of the AWS Lambda function configuration page. The left sidebar lists configuration categories: General configuration, Triggers, Permissions, Destinations, Function URL, and Environment variables. The 'General configuration' section is expanded, showing the following details:

General configuration	
Description	-
Timeout	0 min 3 sec
Memory	128 MB
SnapStart	None
Ephemeral storage	512 MB

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

destination_bucket

Value

awsbucketamit-replica

[Remove](#)

[Add environment variable](#)

▶ Encryption configuration

[Cancel](#)

[Save](#)

Click on Add triggers

myfunction

▼ Function overview

[Info](#)

[Ex](#)

[Diagram](#)

[Template](#)



[+ 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.

s3/myamit23bucket [X](#) [C](#)

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 characters [F2](#) must be URL encoded.

Upload a jpg file in source bucket.

Create a test.

The screenshot shows the AWS Lambda console interface for creating a test event. The top navigation bar includes tabs for Code, Test (which is selected), Monitor, Configuration, Aliases, and Versions. Below the tabs, a section titled "Test event" has a "Info" link. A note says "To invoke your function without saving an event, configure the JSON event, then choose Test." There are two options for "Test event action": "Create new event" (selected) and "Edit saved event". An "Event name" field contains "mytest". A note below it states: "Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores." On the right side of the interface are three buttons: "CloudWatch Logs Live Tail", "Save", and "Test".

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