

CLOUD COMPUTING WITH AWS SERVICES PROJECT

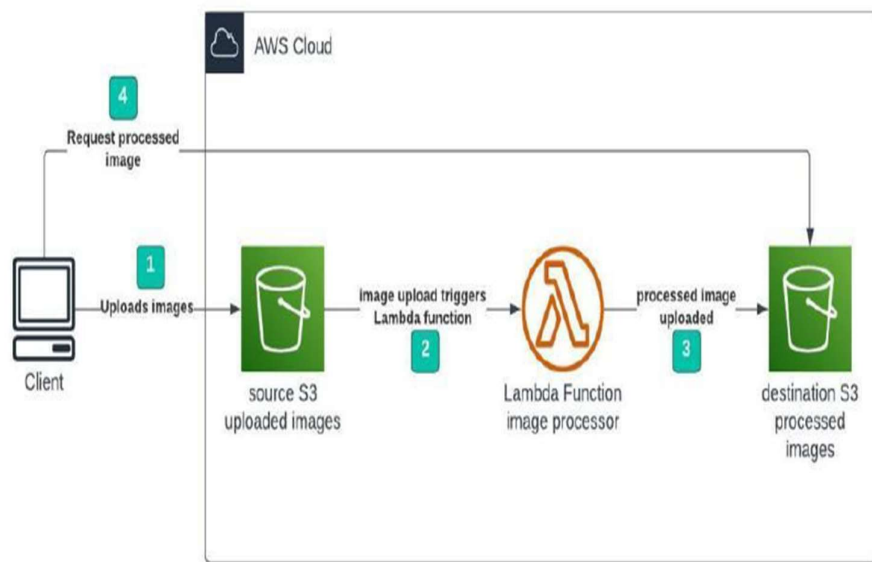
Serverless Image Processing



BY:-ANIKET MISHRA

Serverless Image Processing Flow

1. The user uploads a file to the source S3 bucket (which is used for storing uploaded images).
2. When the image is uploaded to a source S3 bucket, it triggers an event that invokes the Lambda function. The lambda function processes the image.
3. The processed image is stored in the destination S3 bucket.
4. The processed image is requested by the user.



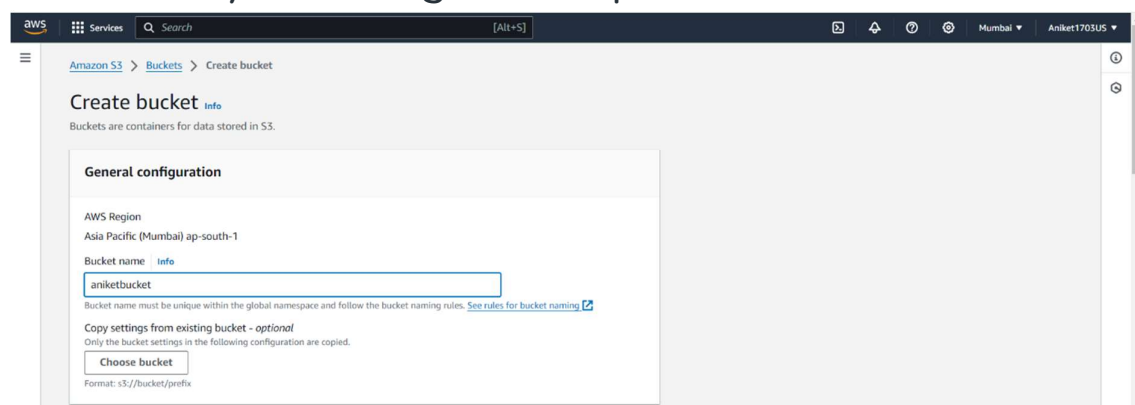
Serverless Image Processor

Step 1 – Creating S3 buckets

We will use two S3 buckets:

1. **source Bucket:** For storing uploaded images.
2. **destination Bucket:** For storing processed images.

Go to the S3 console and click Create bucket. Enter bucket name as 'serverless-bucket-uploaded-images'. Choose any AWS region as 'ap-south-1'.



Step 2 – Configuring the S3 bucket policy

In the 'Block Public Access settings for this bucket' section disable "block all public access". You will get a warning that the bucket and its objects might become public. Agree to the warning. **(Note: we are making this bucket public only for this project, it is not recommended to make an S3 bucket public if not needed).**

☐ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☐ **Block public access to buckets and objects granted through *new* access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ **Block public access to buckets and objects granted through *any* access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☐ **Block public access to buckets and objects granted through *new* public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☐ **Block public and cross-account access to buckets and objects through *any* public bucket or access point policies**
S3 will ignore public access for buckets or access points with policies that grant public access to buckets and objects.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

another bucket named 'serverless-bucket-processed-images' with the same region.

STEP 3: Create an IAM Policy

1. As a pre-requisite for creating the Lambda function, we need to create a user role with a custom policy.
2. Go to **Services** and Select **IAM** under **Security, Identity, and Compliance**.
3. Click on **Policies** in the left navigation bar and click on the **Create Policy** button.

The screenshot shows the AWS IAM console interface. On the left, the 'Policies' link is highlighted in the navigation menu. The main content area shows a list of policies. The 'Create policy' button is highlighted with a red box.

Policy name	Type	Used as	Description
Whiz_Policy_5081117421306	Customer managed	Permissions policy (1)	IAM Policy created for IAM User Whiz...
AdministratorAccess	AWS managed - Job function	Permissions policy (1)	Provides full access to AWS services a...
PowerUserAccess	AWS managed - Job function	None	Provides full access to AWS services a...
ReadOnlyAccess	AWS managed - Job function	None	Provides read-only access to AWS serv...
AWSCloudFormationReadOnlyAccess	AWS managed	None	Provides access to AWS CloudFormall...

4. Click on the **JSON** tab, Remove the existing code, and copy-paste the below policy statement into the editor:

- Policy JSON:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::mysourcebucket12345/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::mydestinationbucket12345/*"
      ]
    }
  ]
}
```

the **Source** and **destination ARN name** of the bucket (which you have saved before) in the option **Resource**. Make sure to add **/*** at the end of the **ARN name**.

- Leave everything as default and click on the **Next** button.

On the Review Policy Page: Policy Name: Enter

mypolicy

Click on the **Create policy** button.

Name* mypolicy
Use alphanumeric and '+', '@', '-', '_' characters. Maximum 128 characters.

Description
Maximum 1000 characters. Use alphanumeric and '+', '@', '-', '_' characters.

Summary

Service	Access level	Resource	Request condition
Allow (1 of 374 services) Show remaining 373			
S3	Limited: Read, Write	Multiple	None

Tags

Key	Value
No tags associated with the resource.	

[Cancel](#) [Previous](#) [Create policy](#)

5. An IAM Policy with the name ***mypolicy*** is created.

Filter policies	Policy name	Type	Used as
Q myp	mypolicy	Customer managed	None

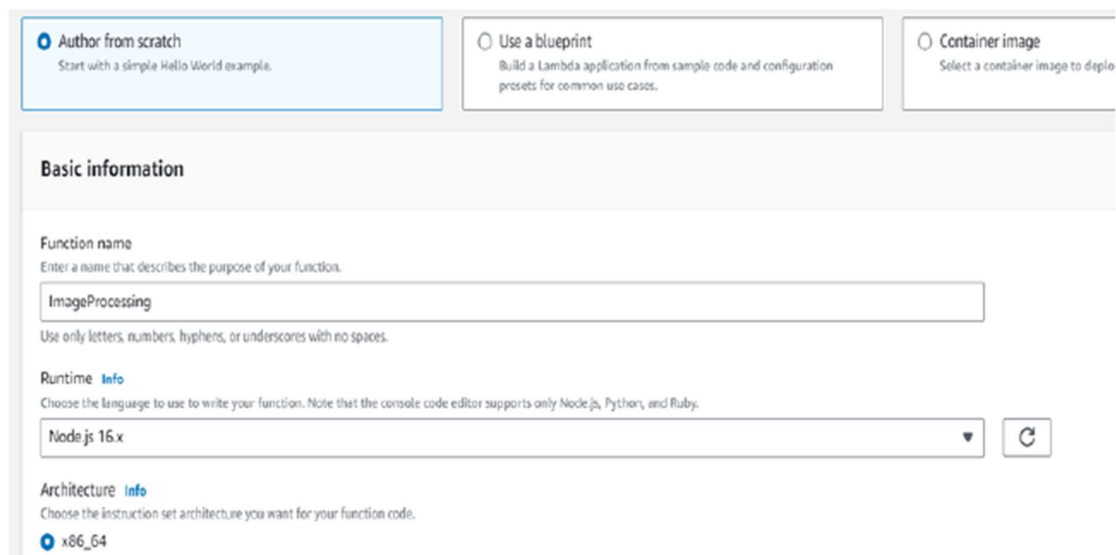
STEP 4: Create an IAM Role

1. In the left menu, click on **Roles**. Click on the **Create role** button.

2. Select **Lambda** from AWS Services list.
- From **Trusted Entity Type**: Select **AWS Service**
- From **Use case**: Select **Lambda**
- Click on the **Next** button.

Step 5 – Creating Lambda function

Go to AWS Lambda console. Navigate to the Functions section. Click Create Function and name it “ImageProcessing”. Select runtime as “NodeJS 16.x” and architecture as “x86_64”. Leave all other settings as default. Create the function.



The screenshot shows the 'Create Function' wizard in the AWS Lambda console. At the top, there are three tabs: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below the tabs is the 'Basic information' section. It contains three fields: 'Function name' with the value 'ImageProcessing', 'Runtime' with the value 'Node.js 16.x', and 'Architecture' with the value 'x86_64'. Each field has a description and a link to 'Info'.

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.

Basic information

Function name
Enter a name that describes the purpose of your function.
ImageProcessing
Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Node.js 16.x

Architecture [Info](#)
Choose the instruction set architecture you want for your function code.
x86_64

In the code editor on the Lambda function page upload the zip file Then click on the configuration and go to the environment variables.

- Use key: **DUST_BUCKET** and fill in the name of your destination bucket in the value block.
- Now click on the test tab and select the S3-put.

Event name

MyEventName

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

s3-put

Event JSON

```

1 {
2   "Records": [
3     {
4       "eventVersion": "1.0",
5       "eventSource": "aws:s3",
6       "awsRegion": "us-east-1",
7       "eventTime": "2019-01-01T00:00:00.000Z",
8       "eventName": "ObjectCreated:Put",
9       "userIdentity": {
10        "principalId": "EXAMPLE"
11      },
12      "requestParameters": {
13        "sourceAddress": "127.0.0.1"
14      },
15      "responseElements": {
16        "etag": "\"1234567890\"",
17        "x-aws-logs-2": "EXAMPLE1234567890123456789012345678901234567890"
18      },
19      "s3": {
20        "bucket": "example-bucket",
21        "key": "test%2Fkey",
22        "size": 1024000,
23        "storageClass": "STANDARD",
24        "metadata": {
25          "Content-Type": "image/jpeg"
26        },
27        "serverSideEncryption": true
28      }
29    ]
30  }

```

– Change the example bucket to the source bucket and also change "test%2Fkey", to your uploaded image name. know it's ready to see so click on the test button

STEP 6: Testing the application

Upload an image file to the source S3 bucket ("serverless-bucket-uploaded-images"). Wait for a few seconds and check the destination bucket ("serverless-bucket-processed-images"). There you will see two images (thumbnail and cover photo).

Congratulations, you just built a serverless Image-processing application.