

Case Study Topic Number 16:

- **Concepts Used:** AWS Lambda, S3.
- **Problem Statement:** "Create an AWS Lambda function that automatically resizes an image when it is uploaded to an S3 bucket. The resized image should be stored in a different S3 bucket."
- **Tasks:**
 - Write a Lambda function in Python that triggers when an image is uploaded to a specific S3 bucket.
 - Use the Python **PIL** library to resize the image.
 - Store the resized image in a different S3 bucket.
 - Test the functionality by uploading an image and verifying the output.

Case Study Overview:

The chosen case study focuses on developing an event-driven image processing system using AWS Lambda and Amazon S3. The main goal is to automate the resizing of images when they are uploaded to a specific S3 bucket, with the resized images saved in a different S3 bucket.

This approach leverages serverless computing, reducing the need for manual image processing and making the solution scalable and cost-efficient.

Key Feature and Application:

The unique feature of this solution is its automatic resizing capability. Upon image upload to the source bucket, a Lambda function is triggered to resize the image and save the resized version to a destination bucket. This is particularly useful in scenarios where images need to be optimized for web use, reducing storage costs and improving website performance through smaller image sizes.

Step-by-Step Explanation

Step 1: Initial Setup

1. **Create S3 Buckets:**
 - Create a source bucket named my-source-bucket-simple for uploading original images.
 - Create a destination bucket named my-destination-bucket-simple where resized images will be stored.
 - Ensure that both buckets are created in the same region as the Lambda function.

The image shows two screenshots of the Amazon S3 console. The top screenshot is for the 'anshi-source-bucket' and the bottom is for the 'anshi-destination-bucket'. Both buckets are empty and show the same interface with tabs for Objects, Properties, Permissions, Metrics, Management, and Access Points. The 'Objects' tab is selected, showing a list of objects (0) and a message: 'No objects. You don't have any objects in this bucket.' The interface includes a search bar, a table with columns for Name, Type, Last modified, Size, and Storage class, and a list of actions like Copy S3 URI, Copy URL, Download, Open, Delete, and Upload.

Amazon S3 > Buckets > anshi-source-bucket

anshi-source-bucket [Info](#)

[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

Objects (0) [Info](#)

[Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

< 1 > [Settings](#)

Name	Type	Last modified	Size	Storage class
No objects				
You don't have any objects in this bucket.				

[Upload](#)

Amazon S3 > Buckets > anshi-destination-bucket

anshi-destination-bucket [Info](#)

[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

Objects (0) [Info](#)

[Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

< 1 > [Settings](#)

Name	Type	Last modified	Size	Storage class
No objects				
You don't have any objects in this bucket.				

[Upload](#)

2. Create Lambda Function:

- Navigate to the AWS Lambda service in the AWS Management Console.
- Click on Create Function and select Author from Scratch.
- Name the function SimpleImageResizeFunction, select Python 3.11 as the runtime, and assign a role with S3 and CloudWatch permissions.
- Click Create Function.

[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.

Node.js 20.x

↕

↻

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

☒ x86_64

☐ arm64

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

► [Change default execution role](#)

► **Additional Configurations**
Use additional configurations to set up code signing, function URL, tags, and Amazon VPC access for your function.

Cancel

Create function

[Lambda](#) > [Functions](#) > image-resize-function

image-resize-function

Throttle Copy ARN Actions ▼


▼ **Function overview** [Info](#)


Export to Application Composer

Download ▼

Diagram

Template

 image-resize-function


 Layers (0)

+ Add trigger

+ Add destination

Description
-

Last modified
4 minutes ago

Function ARN
 arn:aws:lambda:us-east-1:708398963195:function:image-resize-function

Function URL [Info](#)
-


3. Add S3 Trigger to Lambda:

- Scroll down to the Function Overview section and click Add trigger.
- Choose S3 as the trigger type.
- Select the source bucket (my-source-bucket-simple) and choose All object create events.
- Click Add to set the S3 bucket as a trigger for the Lambda function.

[Lambda](#) > Add triggers

Add trigger

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.

✕ ↻

Bucket region: us-east-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 configurations with overlapping prefixes or suffixes that could match the same object key.

PUT ✕

Function overview [Info](#)

[Export to Application Composer](#) [Download](#) ▼

Diagram **Template**

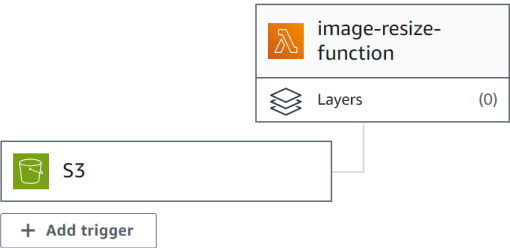



image-resize-function
Layers (0)

S3
+ Add trigger

+ Add destination

Description
-

Last modified
[10 minutes ago](#)

Function ARN
 arn:aws:lambda:us-east-1:708398963195:fun
ction:image-resize-function

Function URL [Info](#)
-

Step 2: Lambda Function Code

1. Write the Lambda Function Code:

```
import json
import boto3
from io import BytesIO

s3 = boto3.client('s3')

def lambda_handler(event, context):
    # Get the S3 bucket and object key from the event
    source_bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']
    destination_bucket = 'my-destination-bucket-simple'

    try:
```

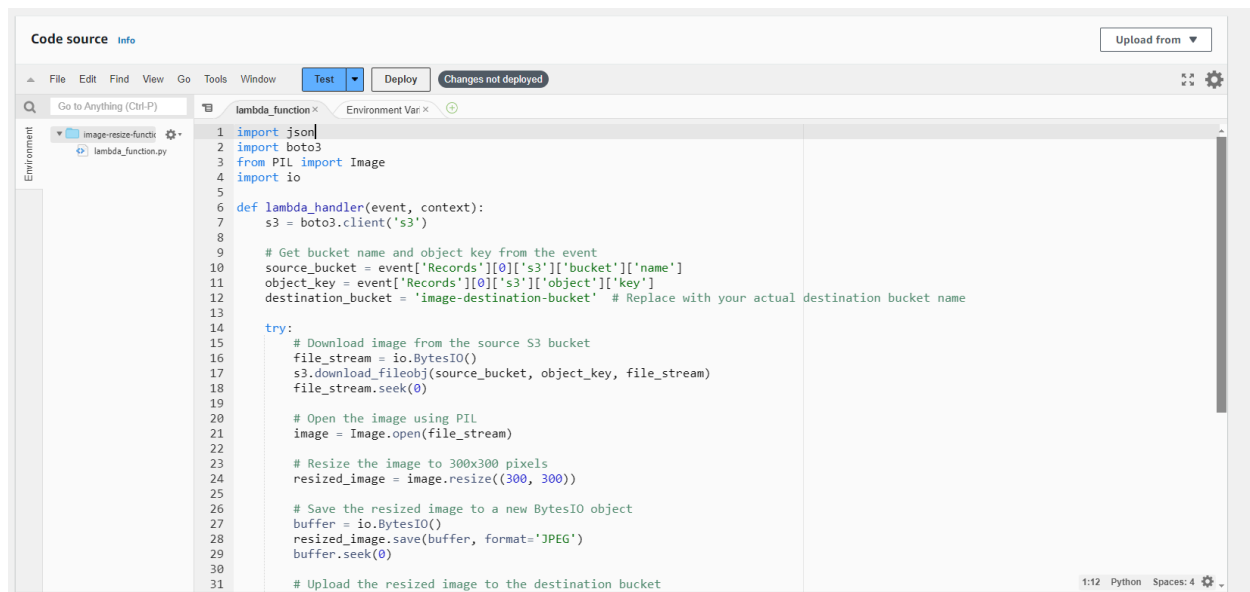
```

# Get the image file from the source S3 bucket
response = s3.get_object(Bucket=source_bucket, Key=key)
file_content = response['Body'].read()

# Simulate image resizing by reducing byte size by half
buffer = BytesIO(file_content)
buffer.seek(0)
reduced_image = buffer.read(int(len(file_content) * 0.5))

# Upload the resized image to the destination bucket
s3.put_object(
    Bucket=destination_bucket,
    Key=f'resized-{key}',
    Body=reduced_image,
    ContentType=response['ContentType']
)
return {
    'statusCode': 200,
    'body': json.dumps(f'Image {key} resized and uploaded successfully to {destination_bucket}')
}
except Exception as e:
    print(f"Error processing image: {str(e)}")
    return {
        'statusCode': 500,
        'body': json.dumps(f'Error processing file {key} from bucket {source_bucket}: {str(e)}')
    }

```



- Click Deploy to save and activate the Lambda function.

Step 3: Testing and Verification

1. Upload an Image to my-source-bucket-simple using the S3 console.
2. Check the Destination Bucket: Verify that a resized image with the prefix resized- is stored in my-destination-bucket-simple.
3. Troubleshoot with CloudWatch Logs if any errors occur. Access logs through the CloudWatch console to find the root cause of any issues.

Upload [Info](#)

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. [Learn more](#)

Drag and drop files and folders you want to upload here, or choose **Add files** or **Add folder**.

Files and folders (1 Total, 1.5 MB)
All files and folders in this table will be uploaded.

< 1 >

<input type="checkbox"/>	Name	Folder	Type
<input type="checkbox"/>	abigail-lynn-9JrBiphz0e0-unsplash.jpg	-	image/

Destination [Info](#)
Destination
<s3://anshi-source-bucket>
► **Destination details**
Bucket settings that impact new objects stored in the specified destination.

[Amazon S3](#) > [Buckets](#) > my-source-bucket-simple

my-source-bucket-simple [Info](#)

[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

Objects (2) [Info](#)

[Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

< 1 > [Settings](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	anshi-image.jpg	jpg	October 19, 2024, 18:30:23 (UTC+05:30)	164.7 KB	Standard
<input type="checkbox"/>	diane-picchiottino-Fo4_-oKnxjE-unsplash.jpg	jpg	October 19, 2024, 18:31:49 (UTC+05:30)	2.9 MB	Standard

my-destination-bucket-simple [Info](#)

[Objects](#) | [Properties](#) | [Permissions](#) | [Metrics](#) | [Management](#) | [Access Points](#)

Objects (2) [Info](#)

[Refresh](#) [Copy S3 URI](#) [Copy URL](#) [Download](#) [Open](#) [Delete](#) [Actions](#) [Create folder](#) [Upload](#)

Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)

< 1 > [Settings](#)

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	resized-anshi-image.jpg	jpg	October 19, 2024, 18:30:25 (UTC+05:30)	82.4 KB	Standard
<input type="checkbox"/>	resized-diane-picchiottino-Fo4_-oKnxjE-unsplash.jpg	jpg	October 19, 2024, 18:31:51 (UTC+05:30)	1.5 MB	Standard

Conclusion:

The event-driven image processing system using AWS Lambda and Amazon S3 demonstrates the power of serverless architecture for automating tasks that would otherwise require manual intervention. By leveraging AWS services, we created a solution that automatically resizes images upon upload, streamlining the process and minimizing resource usage. This solution is scalable, cost-effective, and aligns with best practices in cloud computing, making it ideal for real-world applications where dynamic image processing is required.

Throughout the experiment, key concepts like Lambda triggers, S3 bucket integration, and IAM role configuration were applied, offering practical experience in deploying serverless applications. The simplified approach avoids dependency management complexities, focusing instead on leveraging AWS's native capabilities. This makes it a great starting point for more advanced image processing solutions, which could include more precise resizing with libraries like Pillow or integrating additional AWS services for enhanced functionality.