Name: Anshi Tiwari Class: D15C Roll Number: 56

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:

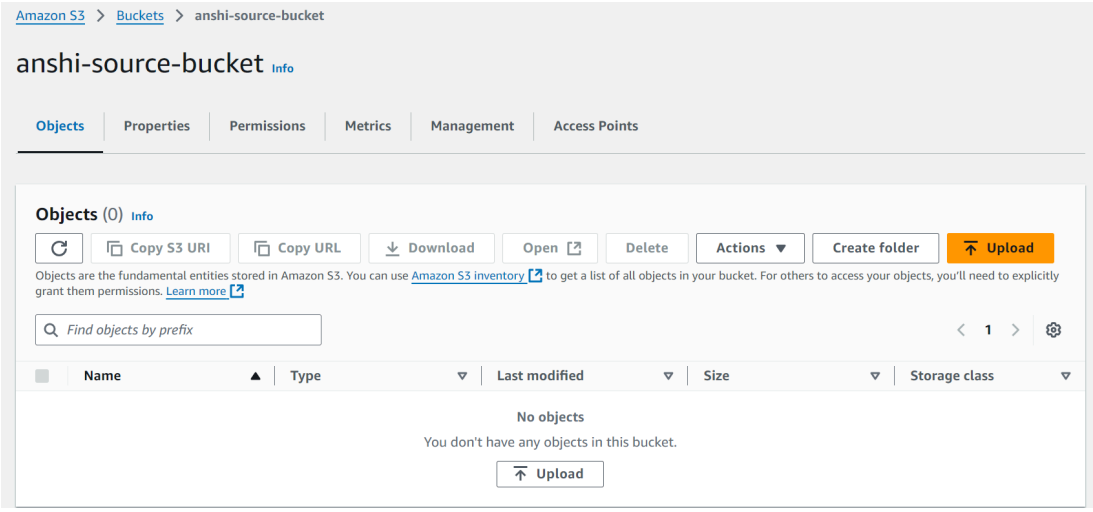
The unique feature of this solution is its automatic resizing capability. Upon image upload to the source bucket, a Lambda function is triggered to resize the image and save the resized version to a destination bucket. This is particularly useful in scenarios where images need to be optimized for web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

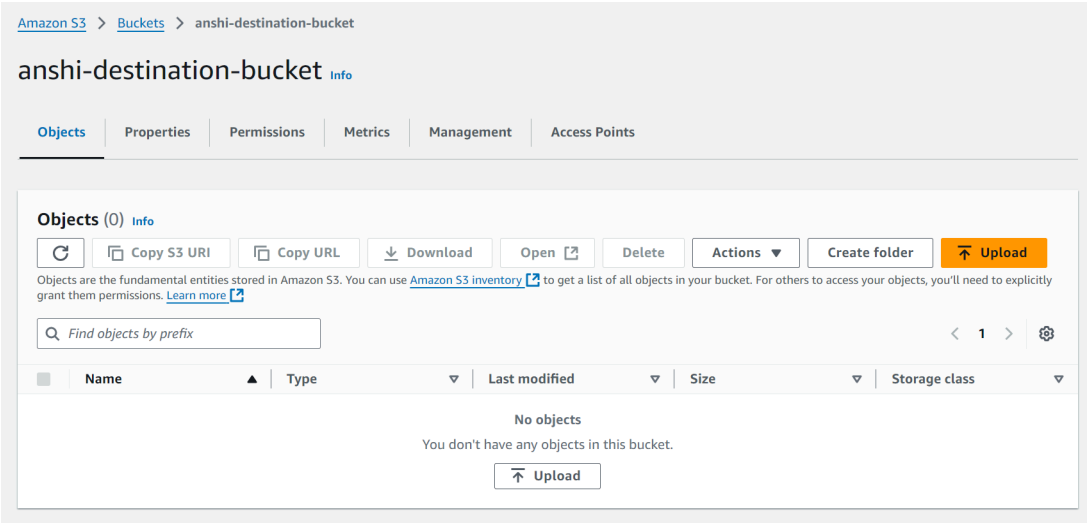
Step-by-Step Explanation

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.

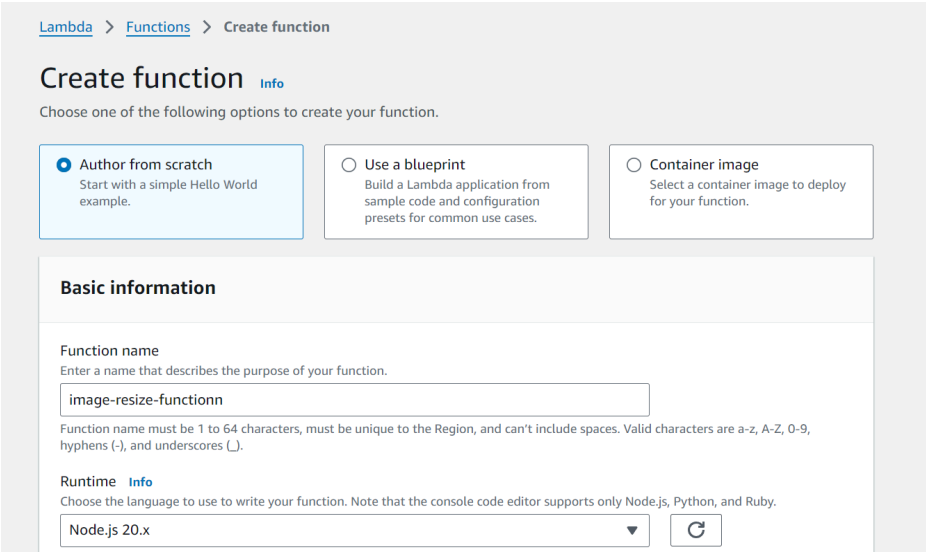
Name: Anshi Tiwari Class: D15C Roll Number: 56

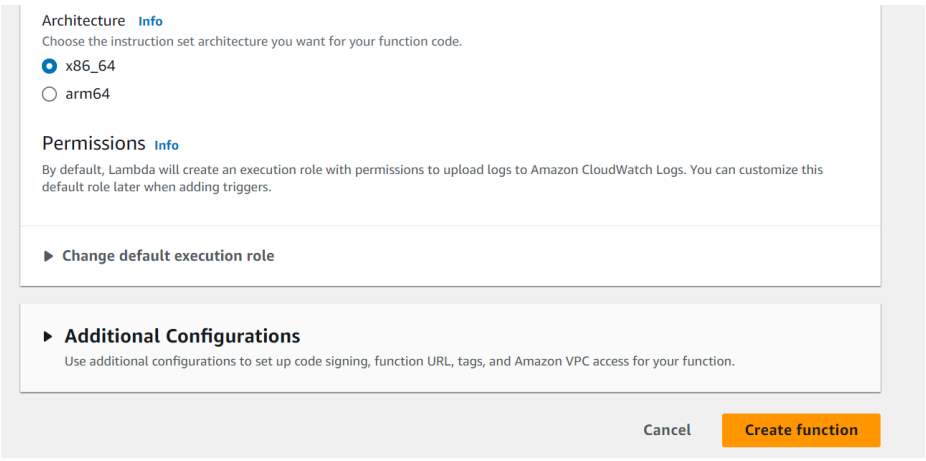


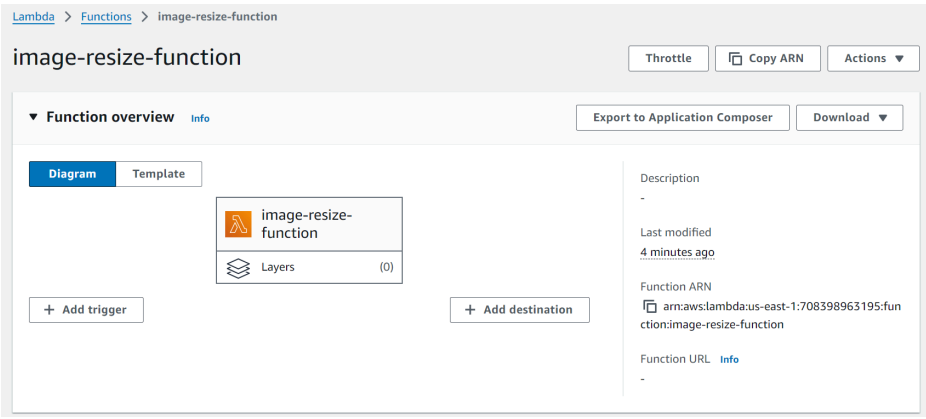


2. **Create Lambda Function**:
    - Navigate to the AWS Lambda service in the AWS Management Console.
    - Click on Create Function and select Author from Scratch.
    - Name the function SimpleImageResizeFunction, select Python 3.11 as the runtime, and assign a role with S3 and CloudWatch permissions.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

Name: Anshi Tiwari Class: D15C Roll Number: 56

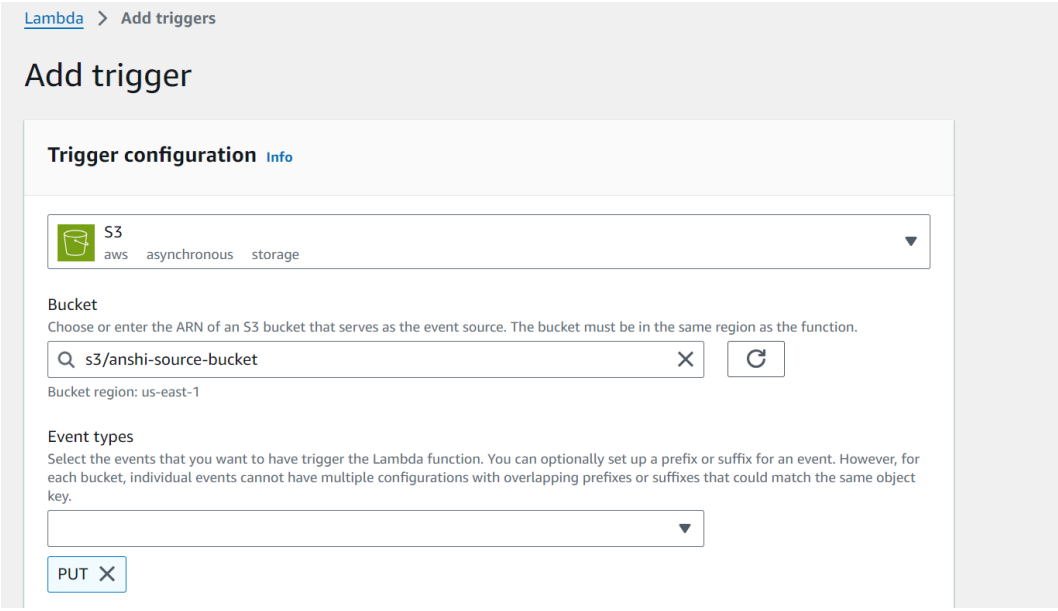


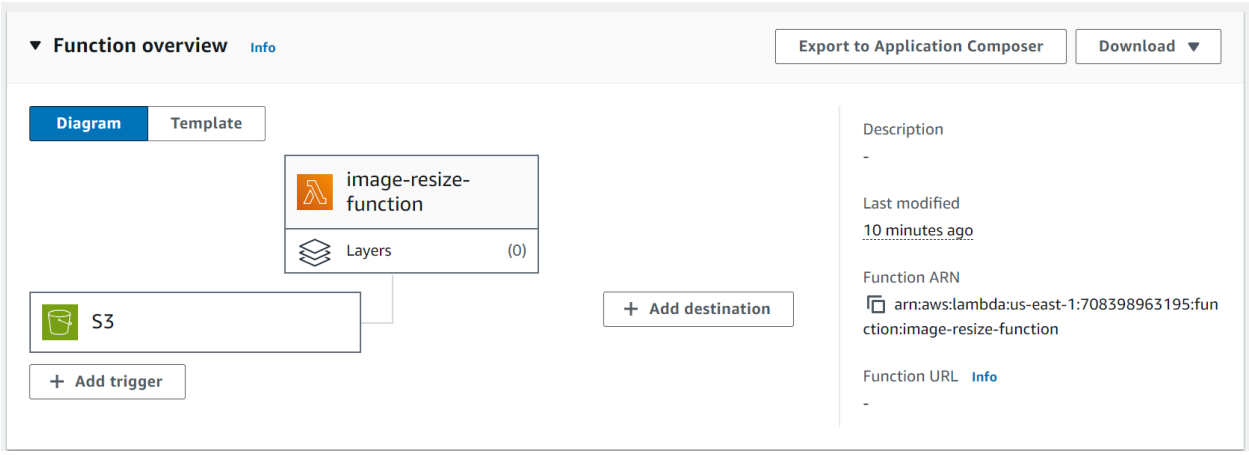




3. **Add S3 Trigger to Lambda**:
    - Scroll down to the Function Overview section and click Add trigger.
    - Choose S3 as the trigger type.
    - Select the source bucket (my-source-bucket-simple) and choose All object create events.
    - Click Add to set the S3 bucket as a trigger for the Lambda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

Name: Anshi Tiwari Class: D15C Roll Number: 56



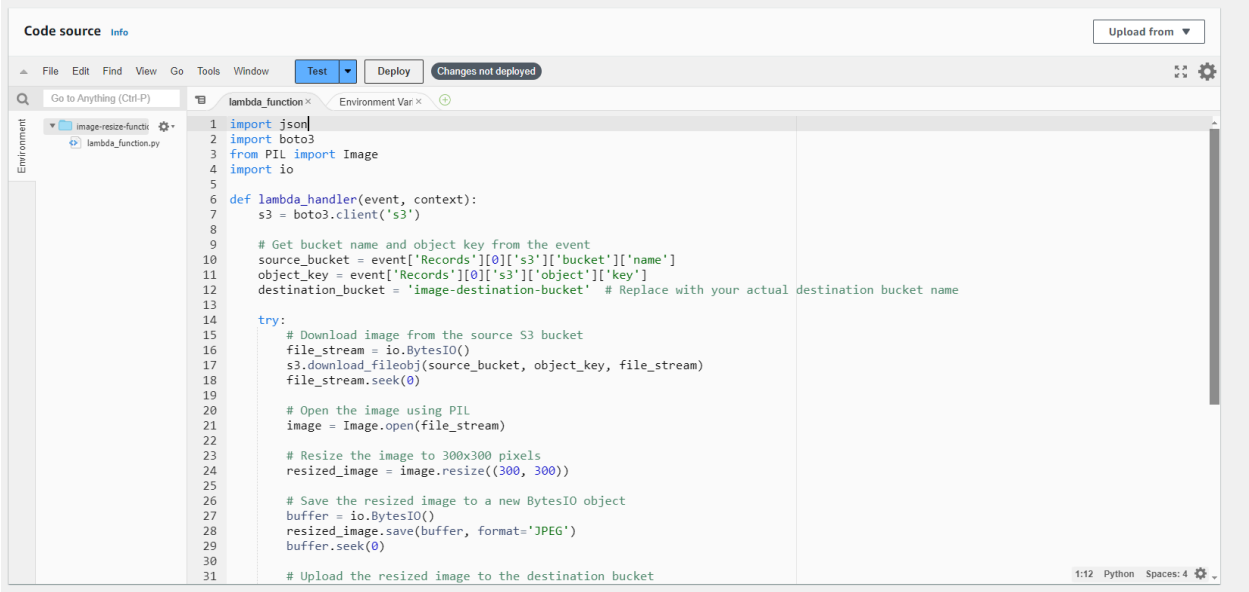


Step 2: Lambda Function Code

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)}')
    }
```

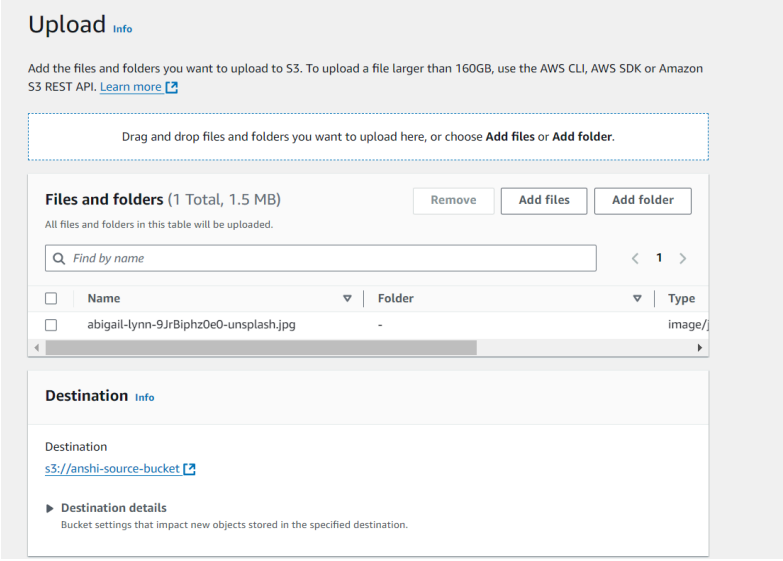


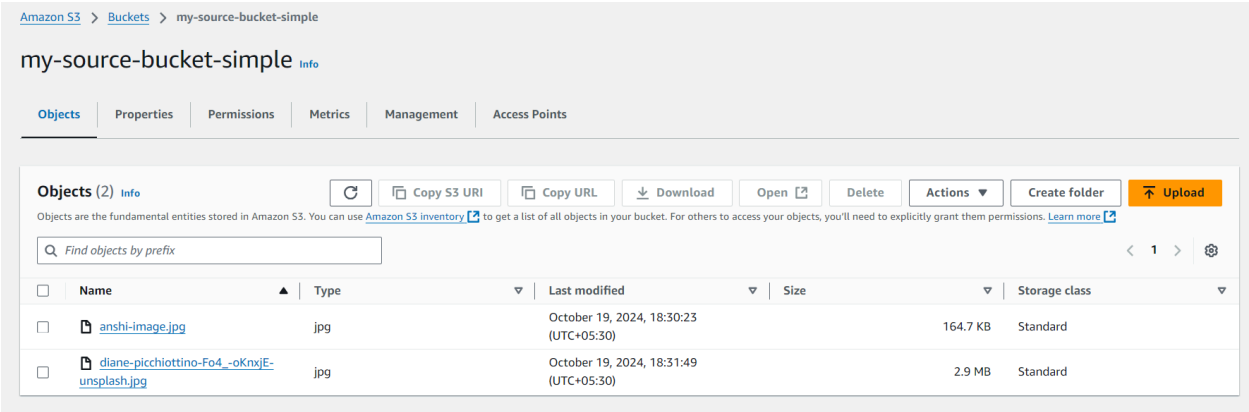
2. **Deploy the Lambda Function**:
    - Click Deploy to save and activate the Lambda function.

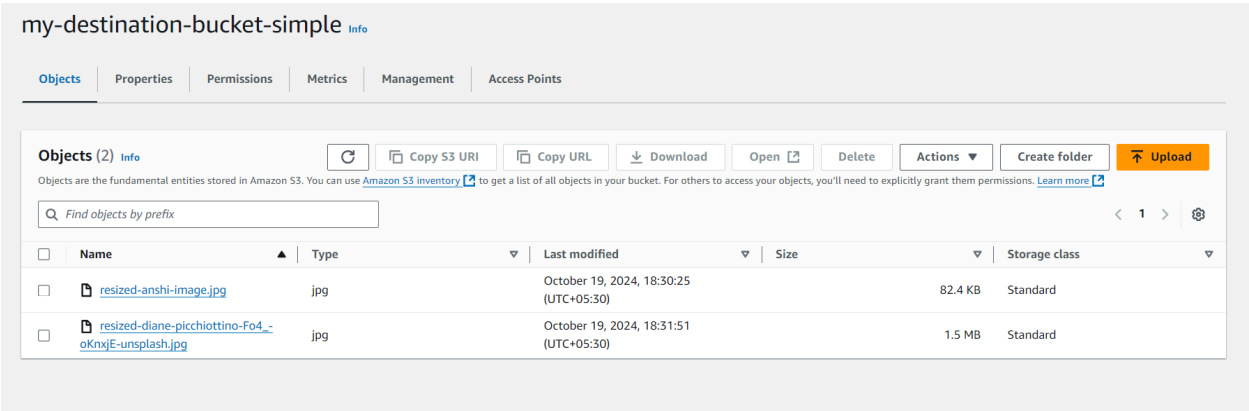
Name: Anshi Tiwari Class: D15C Roll Number: 56

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.







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