Name- Ishan Hemwani **Div- D15C** **Roll No- 16**

REPORT OF AWS LAMBDA S3 IMAGE OF RESIZING USING PYTHON

Introduction

In the digital age, images play a critical role in web and mobile applications. However, serving large images can lead to performance issues. This case study presents a solution that utilizes AWS services to automatically resize images uploaded to an S3 bucket, enhancing user experience and optimizing storage costs.

Objectives

- Automate image resizing upon upload to S3.
- Utilize serverless architecture for cost-effectiveness and scalability.
- Minimize latency and enhance application performance.

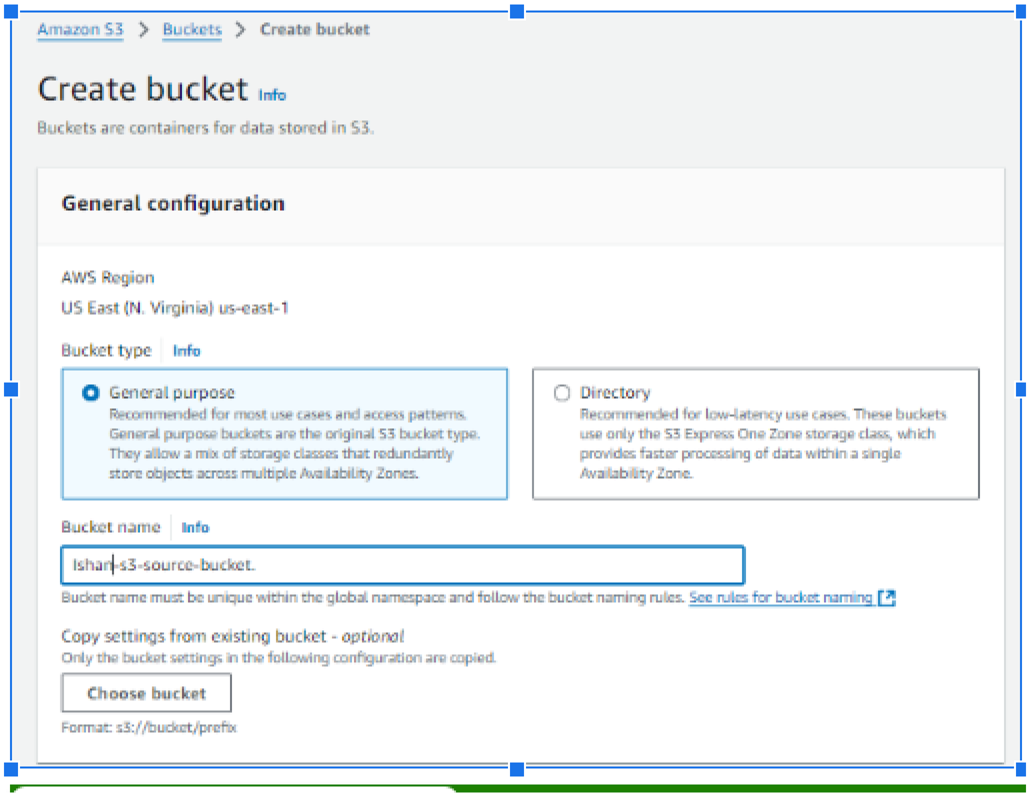
AWS Services Used

1. **Amazon S3 (Simple Storage Service)**
    - Storage for original and resized images.
    - Event notifications to trigger Lambda functions.
2. **AWS Lambda**
    - Serverless compute service to run Python code for image processing.
    - Automatically scales based on the number of incoming requests.
3. **Amazon IAM (Identity and Access Management)**
    - Manage permissions for Lambda functions to access S3.
4. **AWS SDK for Python (Boto3)**
    - Python library for interacting with AWS services

Implementation Steps

Step 1: Set Up S3 Buckets

- Create an S3 bucket to store original images.
- Create another S3 bucket for resized images.


Amazon S3 > Buckets > Create bucket

Create bucket Info

Buckets are containers for data stored in S3.

General configuration

AWS Region
US East (N. Virginia) us-east-1

Bucket type Info

General purpose
Recommended for most use cases and access patterns. General purpose buckets are the original S3 bucket type. They allow a mix of storage classes that redundantly store objects across multiple Availability Zones.

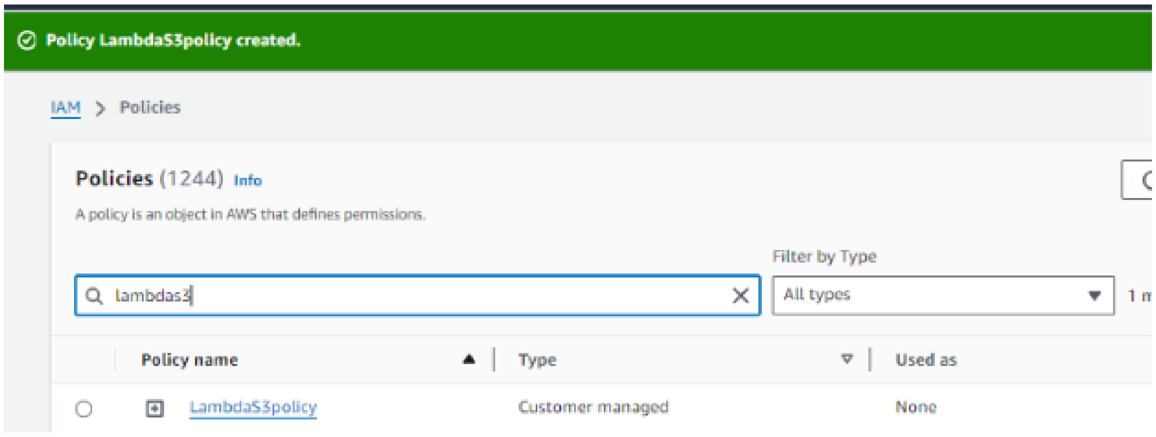
Directory
Recommended for low-latency use cases. These buckets use only the S3 Express One Zone storage class, which provides faster processing of data within a single Availability Zone.

Bucket name Info
Ishan-s3-source-bucket.
Bucket name must be unique within the global namespace and follow the bucket naming rules. See rules for bucket naming

Copy settings from existing bucket - *optional*
Only the bucket settings in the following configuration are copied.
Choose bucket
Format: s3://bucket/prefix

Step 3: Develop the Lambda Function

Create lambda S3 policy


Policy LambdaS3policy created.

IAM > Policies

Policies (1244) Info
A policy is an object in AWS that defines permissions.

Filter by Type
lambdas3
All types

Policy name	Type	Used as
LambdaS3policy	Customer managed	None

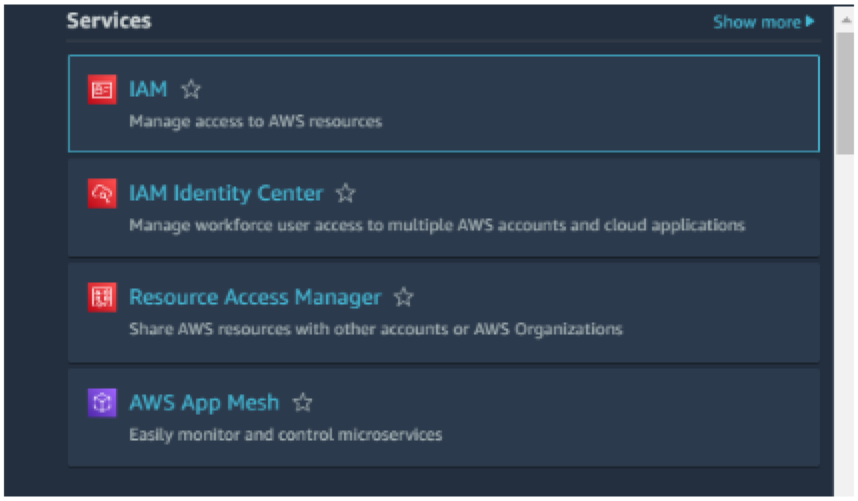
- Create a Python script using the following key components:

{

"Version": "2012-10-17",

```
"Statement": [
    {
        "Effect": "Allow",
        "Action": [
            "logs:PutLogEvents",
            "logs:CreateLogGroup",
            "logs:CreateLogStream"
        ],
        "Resource": "arn:aws:logs:*:*:*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:GetObject"
        ],
        "Resource": "arn:aws:s3:::*/*"
    },
    {
        "Effect": "Allow",
        "Action": [
            "s3:PutObject"
        ],
        "Resource": "arn:aws:s3:::*/*"  } ]}
```

Step 4: Configure IAM Roles open on AWS Management Console



Services

Show more ▸

IAM ☆
Manage access to AWS resources

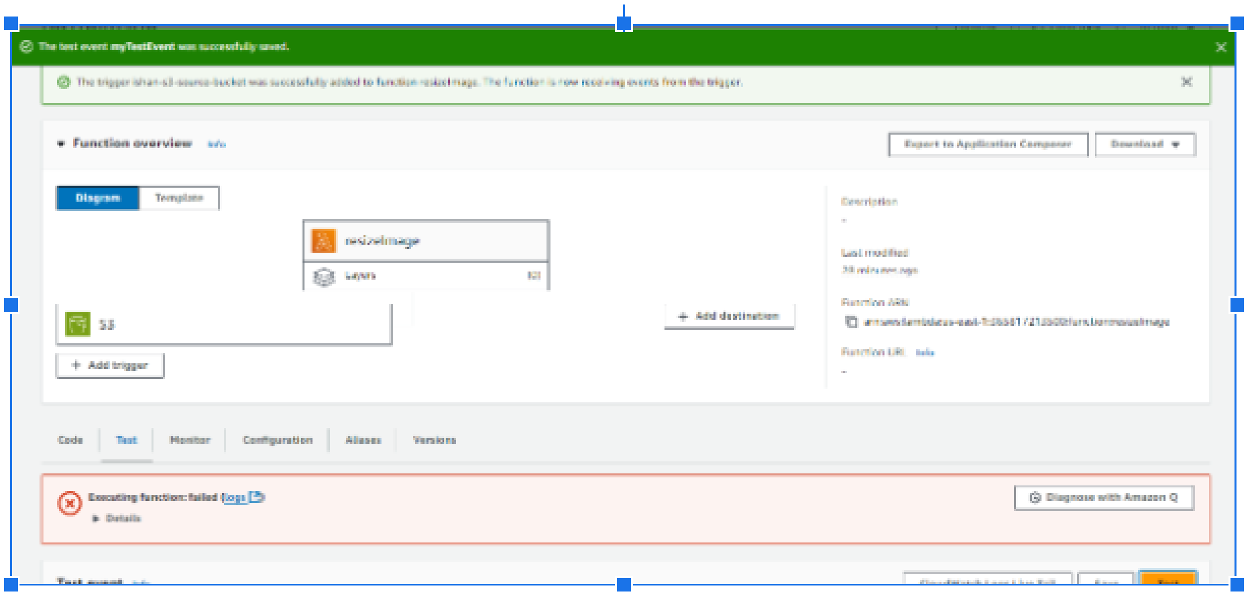
IAM Identity Center ☆
Manage workforce user access to multiple AWS accounts and cloud applications

Resource Access Manager ☆
Share AWS resources with other accounts or AWS Organizations

AWS App Mesh ☆
Easily monitor and control microservices

Step 5: Testing and Validation

- Upload test images to the original S3 bucket and verify that resized images appear in the designated output bucket.



The test event **myTestEvent** was successfully saved.

The trigger [illegible]-s3-source-bucket was successfully added to function resizeImage. The function is now receiving events from the trigger.

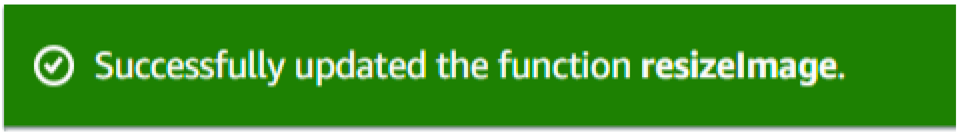
Description: -

Last modified: 20 minutes ago

Function ARN: [illegible]

Function URL: -

Executing function: failed (logs)



Successfully updated the function **resizeImage.**

Results

- The solution successfully automated the image resizing process, allowing for real-time image processing.
- Reduced storage costs by resizing images to appropriate dimensions.
- Improved website performance by serving optimized images.

Challenges and Considerations

- **Lambda Execution Time:** Ensure that the Lambda function's execution time does not exceed the limits .
- **Image Formats:** Handle various image formats (e.g., PNG, JPEG) as needed.
- **Concurrency Limits:** Monitor Lambda function concurrency to avoid throttling during peak uploads.

Conclusion

1. **Automation**: By triggering image resizing automatically upon upload to S3, the process eliminates manual intervention, enhancing operational efficiency.
2. **Improved Performance**: Resizing images optimizes load times, crucial for user satisfaction in a mobile-first environment.
3. **Scalability**: AWS Lambda’s ability to scale automatically ensures that the solution can handle fluctuating traffic without requiring extensive infrastructure management.
4. **Cost-Effectiveness**: The serverless model reduces costs by only charging for actual usage, making it an attractive option for businesses of all sizes.
5. **Security**: Utilizing IAM roles ensures that only authorized components can access sensitive data, maintaining compliance with security standards.

.

Future Work

Here are some additional future work suggestions for enhancing the AWS Lambda-S3 image resizing solution:

1. **Dynamic Resizing Options**:

- Implement a user interface that allows users to specify their preferred image sizes during upload. This could include predefined options or custom dimensions, making the solution more flexible.

2. **Batch Processing:**

   - Develop functionality to handle batch uploads of images, enabling the simultaneous processing of multiple files. This would enhance efficiency, particularly for applications where users frequently upload multiple images.

3. **Image Quality Enhancement**:

   - Explore integrating image enhancement algorithms (e.g., sharpening, color correction) to improve the quality of resized images, ensuring that visual fidelity is maintained even after resizing.

4. **Integration with Other AWS Services:**

   - Consider incorporating Amazon Rekognition for image analysis capabilities, allowing applications to automatically tag and categorize images based on their content.

5. **Cost Optimization Features**

   - Develop features that analyze usage patterns and suggest optimizations for storage and processing costs, helping businesses manage their AWS expenses more effectively.

6. **User Authentication and Authorization**

   - Introduce user authentication to control access to image uploads and processing, ensuring that only authorized users can submit or modify images.

7.. **AI-Powered Features**

   - Investigate the use of AI models to provide additional features, such as automatic tagging, facial recognition, or even generating thumbnails based on image content.

By pursuing these future enhancements, the solution can evolve into a more robust and versatile platform that meets the growing demands of users and leverages the full potential of AWS services.

References

- AWS Lambda Documentation
- Amazon S3 Documentation
- Pillow Documentation (Python Imaging Library)