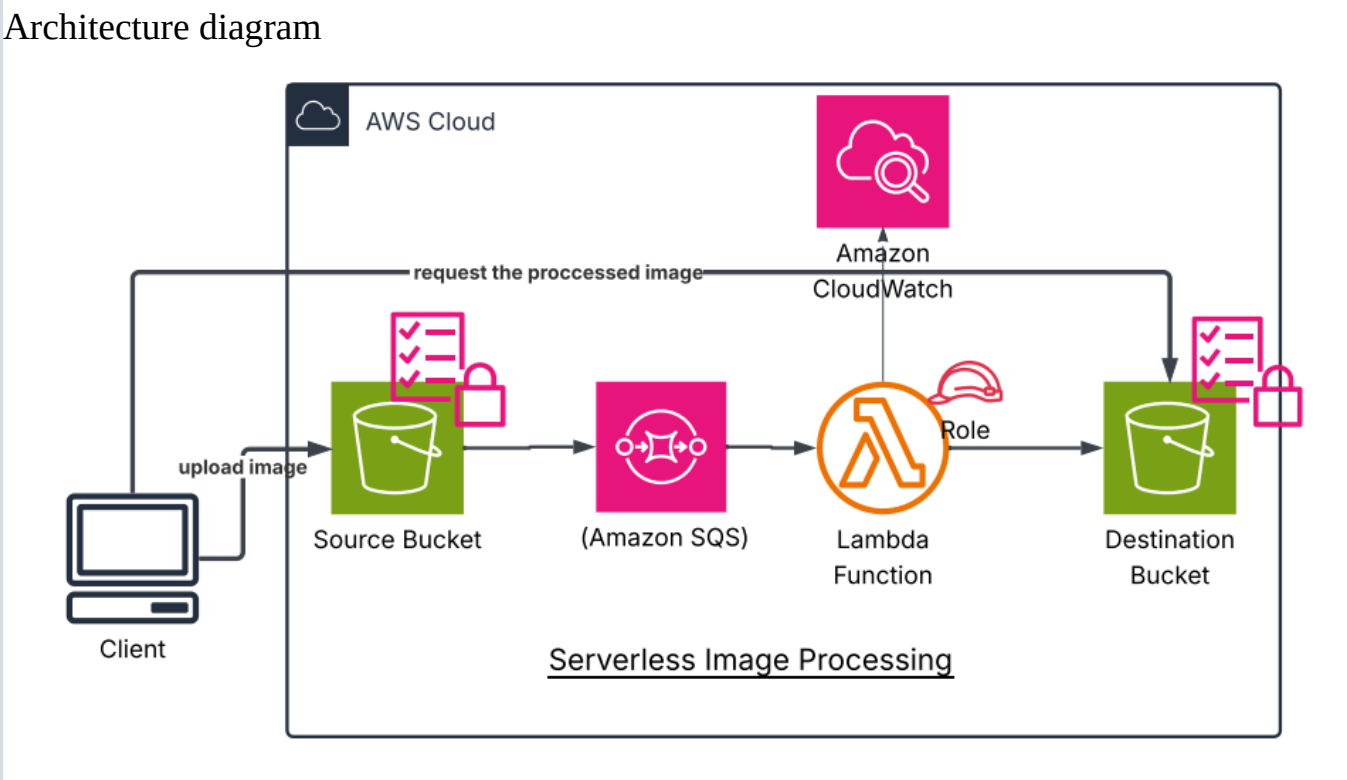
Serverless Image Processing

Architecture diagram



Introduction

This project implements a serverless image processing pipeline using AWS services. The architecture enables users to upload images for processing (e.g., resizing, filtering, format conversion) without provisioning or managing servers. It leverages AWS's scalability, event-driven capabilities, and pay-as-you-go model.

Architecture Overview

The system is composed of the following AWS components:

- **Amazon S3 (Source Bucket)**: Stores the uploaded raw images.
- **Amazon SQS**: Acts as a buffer/queue to decouple image upload from processing, ensuring reliability and scalability.
- **AWS Lambda Function**: The core processing unit that performs image manipulation using libraries like Pillow or Sharp.

- **Amazon S3 (Destination Bucket)**: Stores the processed images ready for client retrieval.
- **Amazon CloudWatch**: Monitors and logs Lambda function executions and system health.

Workflow

1. **Image Upload**:
   The client uploads an image to the **Source S3 Bucket**.
2. **Trigger via Event Notification**:
   The upload triggers an **S3 event**, which places a message into **Amazon SQS**.
3. **Lambda Execution**:
   A **Lambda function** subscribed to the SQS queue is triggered. It:
   - Fetches the image from the Source Bucket.
   - Processes the image (e.g., resizing, watermarking).
   - Stores the processed image into the **Destination Bucket**.
4. **Monitoring**:
   CloudWatch logs all function activities and errors for observability and debugging.
5. **Image Retrieval**:
   The client retrieves the processed image from the **Destination S3 Bucket**, either directly or via a pre-signed URL.

Lambda Function Overview

✅ Function Purpose

The Lambda function is triggered when a new image is uploaded to the **source S3 bucket**. It:

1. Downloads the original image from the source bucket
2. Processes the image (e.g., resizes, converts, watermarks, etc.)
3. Uploads the processed image to the **destination S3 bucket**
4. Optionally deletes the original image or sends a notification

⚙ Lambda Configuration

Setting	Value
Runtime	Python 3.10 (or Node.js, etc.)
Trigger	S3 → `PutObject` event on the source bucket
Timeout	30–60 seconds (depending on image size and processing time)
Memory	256–1024 MB
Role (IAM)	Custom execution role with scoped access to S3 and SQS (if used)

Sample Python Lambda Code (Pillow for image processing)

```
python
CopyEdit
import boto3
from PIL import Image
import io

s3 = boto3.client('s3')

def lambda_handler(event, context):
    source_bucket = event['Records'][0]['s3']['bucket']['name']
    source_key = event['Records'][0]['s3']['object']['key']
    destination_bucket = 'destination-bucket-name'

    # Download the image from source S3
    response = s3.get_object(Bucket=source_bucket, Key=source_key)
    image_content = response['Body'].read()
    image = Image.open(io.BytesIO(image_content))

    # Process image (e.g., resize)
    image = image.resize((300, 300))  # Example: resize to 300x300

    # Save to memory
    buffer = io.BytesIO()
    image.save(buffer, format="JPEG")
    buffer.seek(0)

    # Upload to destination S3
    destination_key = f"processed/{source_key}"
    s3.put_object(Bucket=destination_bucket, Key=destination_key, Body=buffer,
ContentType='image/jpeg')

    return {
        'statusCode': 200,
        'body': f'Image processed and uploaded to {destination_key}'
    }
```

Polices And IAM Role.

1.AM Role for Lambda Function

The Lambda function requires an **execution role** that allows it to access specific AWS services (S3 and SQS) securely and with least privilege.

- Allows reading objects from the source S3 bucket.
- Allows writing objects to the destination S3 bucket.
- Grants permissions to receive, delete, and get attributes from a specific SQS queue.

2. S3 Bucket Policies

Bucket policies are applied directly to **S3 buckets** to control who can access or modify objects.

Source Bucket Policy:

Allows public clients to upload images, but restricts read access.

```
{
  "Effect": "Allow",
  "Principal": "*",
  "Action": "s3:PutObject",
  "Resource": "arn:aws:s3:::source-bucket-name/*"
}
```

Destination Bucket Policy:

Allows public read access to the processed images, enabling any user (e.g., a web or mobile client) to retrieve them. To maintain security:

- Public users can access files via **HTTP GET** requests
- Only **HTTPS (secure transport)** is permitted
- Uploads or modifications are **still restricted**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyInsecureTransport",
```

```
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::destination-bucket-name/*",
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    },
    {
      "Sid": "AllowPublicReadAccess",
      "Effect": "Allow",
      "Principal": "*",
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::destination-bucket-name/*"
    }
  ]
}
```

AWS Services and features Used

Service	Purpose
Amazon S3	Object storage for both original and processed images
Amazon SQS	Decouples upload and processing for reliability
AWS Lambda	Executes image processing logic without managing servers
Amazon CloudWatch	Provides logging and performance monitoring
Resource based policy	To controle access to the s3 buckets
IAM role	For lambda to access s3 buckets

Benefits

- **Scalability**: Automatically scales with load
- **Cost Efficiency**: Pay only for usage (no idle compute cost)
- **Decoupled Design**: SQS adds reliability and improves fault tolerance
- **Maintenance-Free**: No server management required

Possible Enhancements

- Add **Amazon Rekognition** for automated tagging or facial recognition
- Use **Step Functions** for complex workflows
- Enable **pre-signed URLs** for secure direct uploads/downloads
- Integrate with **API Gateway** for a more complete RESTful interface

---

Conclusion

This serverless architecture provides a robust, cost-effective, and scalable solution for image processing using AWS services. It aligns with best practices for modern cloud-native applications, ensuring efficient resource usage and high availability.