Design Document

Serverless Image Processor

Supervisor: Dr. Sumit Kalra

Software and Data Engineering

Team Members:

Ayush Jain (B21BB006)

Lokesh Kiran Chaudhari (B21CS041)

Shrashti Saraswat (B21CS081)

Abstract

This document outlines the design of a serverless image processor web application using AWS services. The application enables users to upload images via a web interface, processes the images using AWS Lambda functions, and stores the processed images back in an S3 bucket.

Objective -

- Develop a highly scalable, serverless application to handle image uploads and processing tasks.
- Leverage AWS services to minimize infrastructure management and costs.
- Provide fast and reliable image processing with auto-scaling capabilities.

Architecture Overview

1.1 AWS services involved:

- **Amazon S3 (Simple Storage Service):** Used to store both original and processed images.
- **AWS Lambda:** Responsible for triggering image processing logic when new images are uploaded to the S3 bucket.
- **Amazon API Gateway:** Provides a web interface for users to interact with the application.
- **Amazon CloudWatch:** Used for logging and monitoring application activity.
- **IAM (Identity and Access Management):** Configured to manage roles and permissions for S3 and Lambda access.

1.2 Application Flow:

- The user uploads an image via the web interface.
- The image is stored in an S3 bucket (Input Bucket).
- AWS Lambda, triggered by an S3 event, processes the uploaded image (e.g., resizing, watermarking).
- The processed image is saved to another S3 bucket (Output Bucket).
- The user can download the processed image from the Output Bucket.

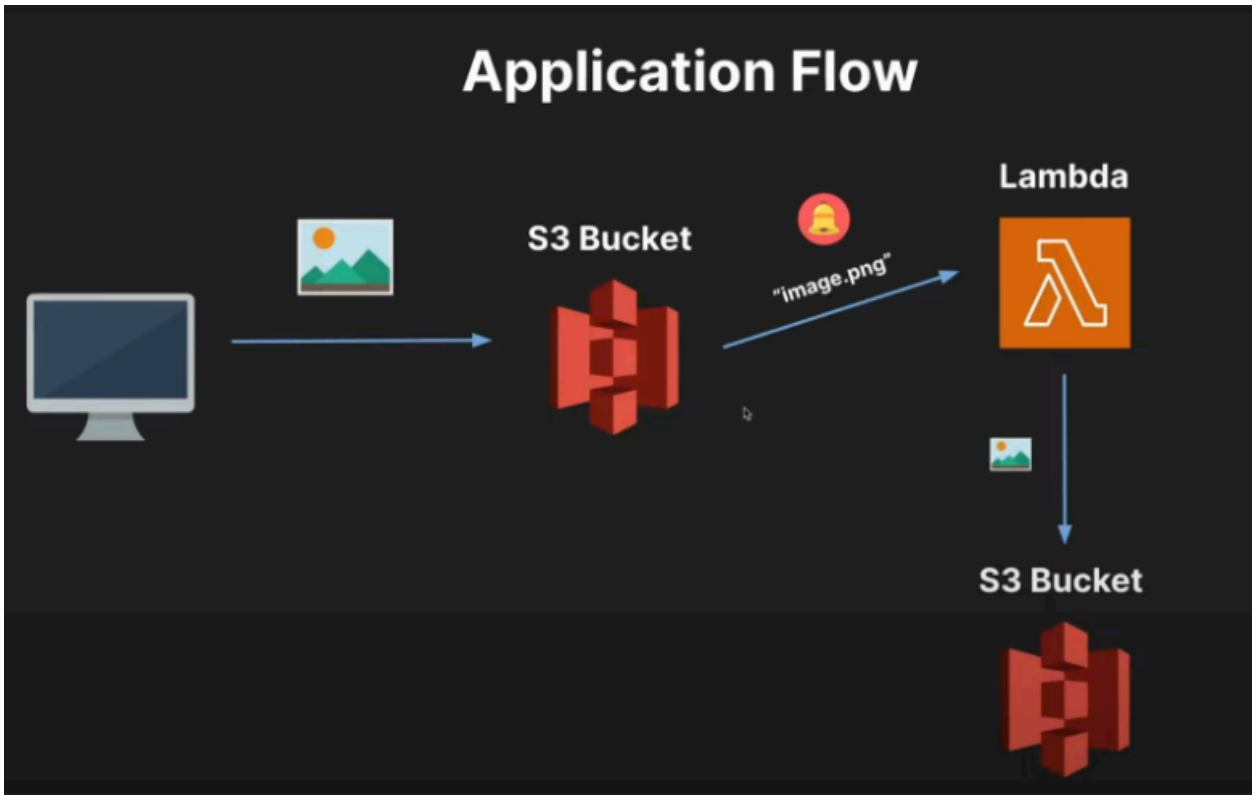

Fig: Application Flow

Detailed Design

2.1 S3 Buckets

- **Input Bucket:** Holds the original images uploaded by users.
  - Name: `image-processor-input-bucket`
  - Permissions: Allow Lambda to read images.
- **Output Bucket:** Holds processed images.
  - Name: `image-processor-output-bucket`
  - Permissions: Allow Lambda to write processed images.

2.2 Lambda Functions

- **Image Processing Lambda Function:**
  - **Trigger:** S3 event when an image is uploaded to the Input Bucket.
  - **Functionality:**
    - Resizes the image to predefined dimensions.
    - Optionally adds a watermark.
    - Stores the processed image in the Output Bucket.
  - **Timeout:** Set to 5 minutes to allow for larger image processing.
  - **Memory Allocation:** 512 MB for better performance.

2.3 API Gateway

- **Functionality:** Provides the HTTP endpoints for users to upload and view images.
- **Method:** `POST` to upload images, `GET` to retrieve processed images.
- **Integration:** API Gateway triggers Lambda, which interacts with the S3 bucket to retrieve or upload images.

2.4 Security

- **IAM Roles:**
  - Lambda functions have a role that grants read permissions for the Input Bucket and write permissions for the Output Bucket.
  - API Gateway has a role that grants the necessary permissions to invoke Lambda functions.
- **S3 Bucket Policies:**
  - The Input Bucket has restricted public access.
  - The Output Bucket allows authenticated users to download processed images.

Implementation

3.1 S3 Bucket Setup

1. Create two S3 buckets:
    - Input Bucket: `image-processor-input-bucket`
    - Output Bucket: `image-processor-output-bucket`
2. Set up appropriate permissions and event triggers for the Input Bucket.

3.2 Lambda Function Setup

1. Create a Lambda function:
    - Runtime: Python 3.x (or Node.js depending on your preference).
    - Handler: `lambda_function.lambda_handler`
    - Environment Variables: Define variables like image dimensions and watermark text.
2. Link the Lambda function to the Input Bucket to trigger the processing function when a new image is uploaded.

3.3 API Gateway Setup

1. Create an API in API Gateway:
    - Define HTTP methods (`POST` for uploading, `GET` for retrieving images).
    - Integrate API Gateway with Lambda.
2. Secure the API using API keys or IAM authentication.

Monitoring and Logging

4.1 CloudWatch Monitoring

- Enable CloudWatch logs for both API Gateway and Lambda to monitor errors and performance.
- Set up alarms for critical failures (e.g., Lambda timeout or memory errors).

4.2 S3 Metrics

- Enable S3 bucket metrics for storage usage and event-based triggers.

Performance and Scaling

5.1 Performance Considerations

- Lambda functions auto-scale based on demand, allowing for efficient image processing.
- Consider using AWS S3 Transfer Acceleration for faster uploads.

5.2 Cost Optimization

- AWS Lambda provides cost-effective scaling. Use CloudWatch alarms to monitor overuse.
- S3 pricing is based on storage, so ensure processed images are deleted after a certain period or are moved to Glacier for long-term storage.