

ABSTRACT

"We present a serverless image generation pipeline that combines the machine learning capabilities of Amazon Sage Maker with the scalability and cost-effectiveness of AWS Lambda. Our approach enables real-time image generation without the need for infrastructure management or upfront costs. We train image generation models using Sage Maker's advanced machine learning tools and deploy them as Lambda functions, allowing us to generate images on demand. This pipeline is ideal for applications requiring dynamic image generation, such as personalized content, product visualization, and AI-generated art. Our solution showcases the power of serverless computing in enabling innovative applications while reducing costs and improving scalability."

Serverless Image Handler (SIH) is an AWS Solution Implementation that enables fast and cost-effective image management. It allows you to store a single version of every image, while dynamically delivering different versions at runtime based on the end user's device. SIH uses a combination of highly available, trusted AWS services and the open-source image processing software Sharp for cost-effective image manipulation in the AWS Cloud. SIH leverages Amazon Cloud Front for fast content delivery, Amazon Simple Storage Service (Amazon S3) for object storage, and Amazon Recognition for face detection in images. This solution provides a great way to manipulate and serve images on the fly with speed and security.

Vision and Mission:

Vision of the institute	To be a world-class leader in transforming lives through an innovative, rigorous and compassionate approach by imparting high-quality academic excellence in technical education to uplift the living standards of the rural youth by promoting cutting edge technologies, employability, higher education, and research with socio-technical, eco-friendly, and entrepreneurial values.
Mission of the institute	To develop high quality technical personal with a sound footing on basic engineering principles, innovative capabilities and exemplary professional conduct to lead and to use technology for the progress of mankind, training and adopting them to changing technological environment by providing the high quality instruction, infra, faculty, effective training and learning methods imbibing socioethical, eco-friendly and entrepreneurial values as the inner strength for the rural development.
Vision of the Department	To become a center for academic excellence by producing high quality engineers in Computer science & Engineering with global competence empowered by technical expertise, innovation, research and the knowledge of cutting edge technologies, employability, higher education, entrepreneurial skills along with socio-ethical and eco-friendly values for the rural development.
Mission of the Department	To develop the high quality computer science & Engineers with the fundamental engineering principles, intellectually adept, professionally deft, innovative research capabilities, exemplary skills, technological progress of mankind, tracking technological environment by providing the high quality instruction, infra, faculty, modern teaching and learning methods, training with socio-ethical and entrepreneurial values as the inner strength for the rural development.

TABLE OF CONTENTS

1. Introduction
2. Prerequisites
3. Implementation
4. Security
5. Conclusion

1.INTRODUCTION

Serverless Image Generation is a cloud-based solution that enables fast, scalable, and cost-effective image processing and delivery. This solution leverages the power of AWS services, including AWS Lambda, Amazon S3, API Gateway, and Amazon CloudFront, to generate and deliver images on demand.

Scalability

Handle large volumes of image requests without provisioning or managing servers
Automatically scale to meet changing demand, ensuring high availability and performance

High Performance

Leverage the power of AWS services to deliver fast and responsive image processing and delivery

Use Amazon CloudFront's content delivery network (CDN) to reduce latency and improve image delivery

Flexibility

Easily integrate with existing applications and services

Support a wide range of image formats and processing operations

Use AWS services to generate images in various sizes, resolutions, and formats

Security

Leverage AWS's robust security features, including IAM roles and permissions, to control access to image processing and storage

Use SSL/TLS encryption to secure image delivery and protect user data

Managed Services

Focus on image generation and processing, while AWS manages the underlying infrastructure and services

Integration with Other AWS Services

Seamlessly integrate with other AWS services, such as Amazon Rekognition, Amazon SageMaker, and Amazon EC2

Real-time Processing

Process images in real-time, enabling fast and responsive image delivery

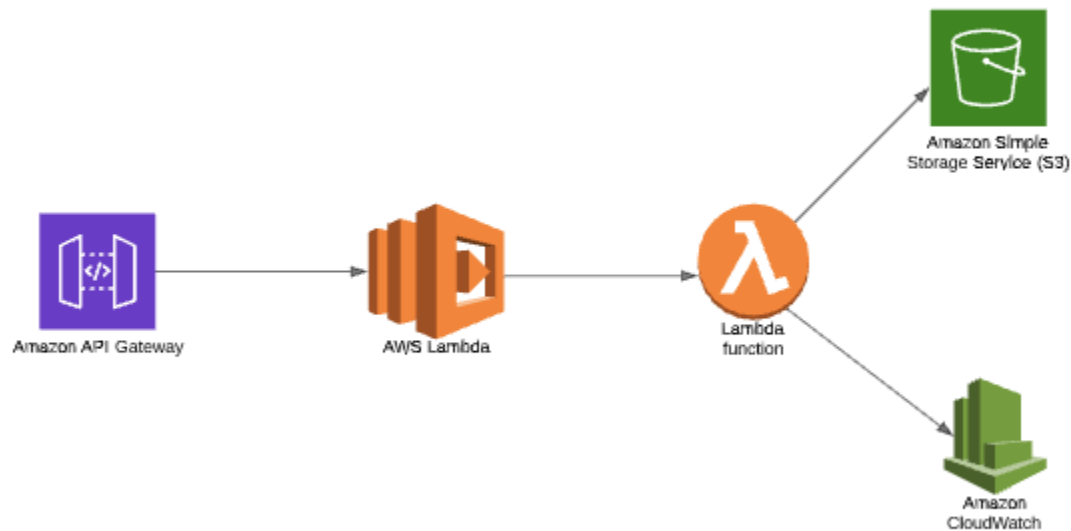
Use AWS Lambda's event-driven architecture to process images as soon as they are uploaded or requested

Analytics and Insights

Use AWS services, such as Amazon CloudWatch and Amazon X-Ray, to monitor and analyze image processing and delivery

Gain insights into image usage and performance, enabling data-driven decision making

Architecture Overview :-



The AWS CloudFormation template deploys an Amazon CloudFront distribution, Amazon API Gateway REST API, and an AWS Lambda function. Amazon CloudFront provides a caching layer to reduce the cost of image processing and the latency of subsequent image delivery. The Amazon API Gateway provides endpoint resources and triggers the AWS Lambda function. The AWS Lambda function retrieves the image from the customer's Amazon Simple Storage Service (Amazon S3) bucket and uses Sharp to return a modified version of the image to the API Gateway. Additionally, the solution generates a CloudFront domain name that provides cached access to the image handler API.

2. Prerequisites

AWS Account and Credentials :

- You need an AWS account with valid credentials (access key ID and secret access key) to use AWS services.
- Make sure you have the necessary permissions to create and manage AWS resources.

Node.js and sharp Library Installation

- Install Node.js (version 14 or later) on your local machine.
- Install the sharp library using npm by running the command `npm install sharp` in your terminal.

Amazon S3 Bucket Creation

- Create an Amazon S3 bucket to store the generated images.
- Make sure the bucket is in the same region as the AWS Lambda function.
- Note down the bucket name and region, as you'll need them later.

AWS CLI Installation (Optional)

- If you want to use the AWS CLI to deploy and manage your AWS resources, install it on your local machine.
- Make sure you have the necessary permissions to use the AWS CLI.

AWS IAM Role Creation (Optional)

- If you want to use an IAM role to manage permissions for your AWS Lambda function, create a new IAM role.
- Attach the necessary policies to the role, such as `AWSLambdaExecute` and `AmazonS3ReadOnlyAccess`.
- Note down the IAM role ARN, as you'll need it later.

Amazon CloudFront Distribution Creation (Optional)

- If you want to use Amazon CloudFront to distribute your generated images, create a new CloudFront distribution.
- Configure the distribution to use your Amazon S3 bucket as the origin.
- Note down the CloudFront distribution ID, as you'll need it later.

Familiarity with AWS Services

- Familiarize yourself with AWS services, including AWS Lambda, Amazon API Gateway, Amazon S3, and Amazon CloudFront.

- Understand the concepts of serverless computing, API gateways, and content delivery networks.

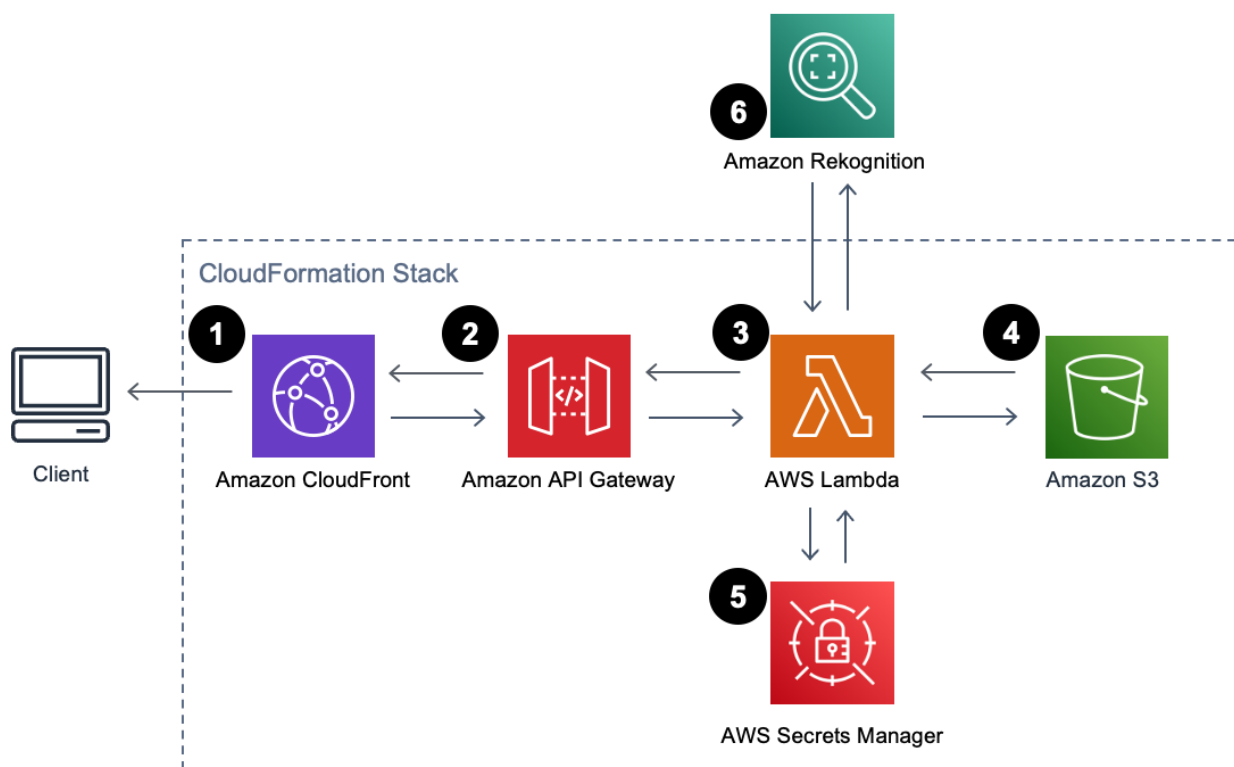
Code Editor or IDE

- Choose a code editor or IDE (such as Visual Studio Code, IntelliJ IDEA, or

Sublime Text) to write and deploy your AWS Lambda function code.

Internet Connection

- Make sure you have a stable internet connection to deploy and manage your AWS resources.



3.IMPLEMENTATION

Serverless Image Processing Flow :

- 1.User uploads a file to the source S3 bucket (which is used for storing uploaded images).
- 2.When the image is uploaded to a source S3 bucket, it triggers an event which invokes the Lambda function. The lambda function processes the image.
- 3.Processed image is stored in the destination S3 bucket.
- 4.The processed image is requested by the user.



Step 1 – Creating S3 buckets

We will use two S3 buckets:

1. Source Bucket: For storing uploaded images.
2. Destination Bucket: For storing processed images.

Go to S3 console and click Create bucket. Enter bucket name as 'serverless-bucket-uploaded-images'. Choose any AWS region as 'ap-south-1'.

Create bucket [Info](#)

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name

Bucket name must be unique within the global namespace and follow the bucket naming rules. [See rules for bucket naming](#)

AWS Region

Copy settings from existing bucket - *optional*


Only the bucket settings in the following configuration are copied.

Step 2 – Configuring S3 bucket policy

In 'Block Public Access settings for this bucket' section disable "block all public access". You will get a warning that the bucket and its objects might become public. Agree to the warning. (Note: we are making this bucket public only for this project, it is not recommended to make an S3 bucket public if not needed).

☐ **Block all public access**
Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- ☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- ☐ **Block public access to buckets and objects granted through any access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
- ☐ **Block public access to buckets and objects granted through new public bucket or access point policies**
S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- ☐ **Block public and cross-account access to buckets and objects through any public bucket or access point policies**
S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

 **Turning off block all public access might result in this bucket and the objects within becoming public**
AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.
☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Leave all other settings as default and create bucket. Similarly, create another bucket named 'serverless-bucket-processed-images' with the same region. This bucket will be used to store the processed images. Although we enabled public

access while creating the buckets, we still need to attach a bucket policy to access the objects stored in it. (Policies in AWS are JSON documents which defines the permissions for performing actions on a certain resource.)

Go to your source bucket and then click on Permissions tab. Scroll to Bucket Policy tab. Click Edit. You will be redirected to the policy editor. Click on policy generator.

Enter the following settings:

Type of policy:

- S3 Bucket Policy
- Effect:Allow
- Principal: *
- Actions: GetObject
- Amazon Resource Name (ARN):
- arn:aws:s3:::SOURCE_BUCKET_NAME/*

SOURCE_BUCKET_NAME is the name of the bucket used for uploading the images.

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an IAM Policy, an S3 Bucket Policy, an SNS Topic Policy, VPC Endpoint Policy, and an SQS Queue Policy.

Select Type of Policy S3 Bucket Policy

Step 2: Add Statement(s)

A statement is the formal description of a single permission. See a [description of elements](#) that you can use in statements.

Effect ☒ Allow ☐ Deny

Principal

Use a comma to separate multiple values.

AWS Service Amazon S3 ☐ All Services (*)

Use multiple statements to add permissions for more than one service.

Actions 1 Action(s) Selected ☐ All Actions (*)

Amazon Resource Name (ARN)

ARN should follow the following format: arn:aws:s3:::{BucketName}/{Keyname}.
Use a comma to separate multiple values.

[Add Conditions \(Optional\)](#)

Click Add Statement and then generate policy. Copy the JSON object.

Policy JSON Document

Click below to edit. To save the policy, copy the text below to a text editor.
Changes made below will **not be reflected in the policy generator tool**.

```

{
  "Id": "Policy1695242382777",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1695242381344",
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::serverless-bucket-uploaded-images/*",
      "Principal": "*"
    }
  ]
}

```

This AWS Policy Generator is provided for informational purposes only, you are still responsible for your use of Amazon Web Services technologies and ensuring that your use is in compliance with all applicable terms and conditions. This AWS Policy Generator is provided as is without warranty of any kind, whether

Close

Type of policy: S3 Bucket Policy

→Effect:Allow

→Principal: *

→Actions: GetObject, PutObject, and PutObjectAcl

→Amazon Resource Name (ARN):

arn:aws:s3:::DESTINATION_BUCKET_NAME/*

DESTINATION_BUCKET_NAME is the name of the bucket used for storing processed images.

```

{
  "Version": "2012-10-17",
  "Id": "Policy1695244199034",
  "Statement": [
    {
      "Sid": "Stmt1695244197617",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::serverless-bucket-processed-images/*"
    }
  ]
}

```

Step 3 – Creating Lambda function

Go to AWS Lambda console. Navigate to Functions section. Click Create Function and name it “ImageProcessing”. Select runtime as “NodeJS 16.x” and architecture as “x86_64”. Leave all other settings as default. Create the function.

The screenshot shows the AWS Lambda console interface for creating a new function. At the top, there are three tabs: 'Author from scratch' (selected), 'Use a blueprint', and 'Container image'. Below the tabs is the 'Basic information' section. It contains the following fields:

- Function name:** A text input field containing 'ImageProcessing'. Below it, a note says 'Enter a name that describes the purpose of your function. Use only letters, numbers, hyphens, or underscores with no spaces.'
- Runtime:** A dropdown menu set to 'Node.js 16.x'. Below it, a note says 'Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.'
- Architecture:** A dropdown menu set to 'x86_64'. Below it, a note says 'Choose the instruction set architecture you want for your function code.'

In the code editor on the Lambda function page paste the following code. This function is executed whenever an image is uploaded to our source S3 bucket and creates two images (thumbnail (300×300) and coverphoto(800×800)) and stores it in the destination S3 bucket. **(Note: The value of processedImageBucket in the code should be set to the name of the destination bucket).**

Java script :

```
const sharp = require("sharp");

const path = require("path");

const AWS = require("aws-sdk");

AWS.config.update({

    region: "ap-south-1",

});

const s3 = new AWS.S3();
```

```
const processedImageBucket = "serverless-bucket-processed-images";

exports.handler = async (event, context, callback) => {

  console.log("An object was added to S3 bucket", JSON.stringify(event));

  let records = event.Records;

  let size = records.length;

  for (let index = 0; index < size; index++) {

    let record = records[index];

    console.log("Record: ", record);

    let fileName = path.parse(record.s3.object.key).name;

    let filePath = path.parse(record.s3.object.key).dir;

    let fileExt = path.parse(record.s3.object.key).ext;

    console.log("filePath:" + filePath + ", fileName:" + fileName + ", fileExt:" +
fileExt);

    let imageObjectParam = {

      Bucket: record.s3.bucket.name,

      Key: record.s3.object.key,

    };

    let imageObject = await s3.getObject(imageObjectParam).promise();

    let resized_thumbnail = await sharp(imageObject.Body)

      .resize({

        width: 300,

        height: 300,

        fit: sharp.fit.cover,
```

```

    })

    .withMetadata()

    .toBuffer();

console.log("thumbnail image created");

let resized_coverphoto = await sharp(imageObject.Body)

    .resize({

        width: 800,

        height: 800,

        fit: sharp.fit.cover,

    })

    .withMetadata()

    .toBuffer();

console.log("coverphoto image created");

let thumbnaillImageParam = {

    Body: resized_thumbnail,

    Bucket: processedImageBucket,

    Key: fileName + "_thumbnail" + fileExt,

    CacheControl: "max-age=3600",

    ContentType: "image/" + fileExt.substring(1),

};

let result1 = await s3.putObject(thumbnaillImageParam).promise();

console.log("thumbnail image uploaded:" + JSON.stringify(result1));

let coverphotoImageParam = {

    Body: resized_coverphoto,

```

```

    Bucket: processedImageBucket,

    Key: fileName + "_coverphoto" + fileExt,

    CacheControl: "max-age=3600",

    ContentType: "image/" + fileExt.substring(1),

  };

  let result2 = await s3.putObject(coverphotoImageParam).promise();

  console.log("coverphoto image uploaded:" + JSON.stringify(result2));

}

};

```

Save the code and click Deploy to deploy the changes.

Go to Configuration tab and Edit the general configuration. There set the timeout to 1 min (timeout is the maximum time for which a Lambda function will run after which it stops running). We need to increase the timeout because the image can take time to process. Click on Save changes

You can configure up to 10 GB of ephemeral storage (/tmp) for your function. [View pricing](#)

MB

Set ephemeral storage (/tmp) to between 512 MB and 10240 MB.

SnapStart [Info](#)

Reduce startup time by having Lambda cache a snapshot of your function after the function has initialized. To evaluate whether your function code is resilient to snapshot operations, review the [SnapStart compatibility considerations](#).

Supported runtimes: Java 11, Java 17.

Timeout

min sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

☒ Use an existing role

☐ Create a new role from AWS policy templates

Existing role

Step 4 – Creating Lambda layer and attaching it to Lambda function

Layers in Lambda is used to add dependencies to a Lambda Function. Lambda Layers reduces the code size of Lambda functions as we do not need to upload the dependencies with the function. It also useful for code reusability as we can reuse the layer with multiple functions if they require the same dependencies.

First we need to create a zip file with all the dependencies (node modules in our case) required by our Lambda function.

Create a folder “aws-serverless-image-processor”. Inside this directory create another directory “nodejs” (it is compulsory to name this as “nodejs”). Open a terminal and go to nodejs directory. Install sharp module with the following command (platform is linux because the Lambda function runs on a Linux machine so we require the node_modules for Linux).

```
npm install --arch=x64 --platform=linux sharp
```

Now create a zip file of the nodejs directory and name it “sharpplayer.zip”.

Go to Layers in Lambda console. Click Create layer. Name it “sharp-layer”. Upload your nodejs “sharpplayer.zip” file here. Select x86_64 architecture. Select NodeJS 16.x in compatible runtimes. Click on Create Layer.

Layer configuration

Name

sharp-layer

Description - optional

Contains sharp node_modules for image processing in NodeJS

☒ Upload a .zip file

☐ Upload a file from Amazon S3

Upload

sharpplayer.zip
9.73 MB

For files larger than 10 MB, consider uploading using Amazon S3.

Compatible architectures - optional [Info](#)

Choose the compatible instruction set architectures for your layer.

☒ x86_64

☐ arm64

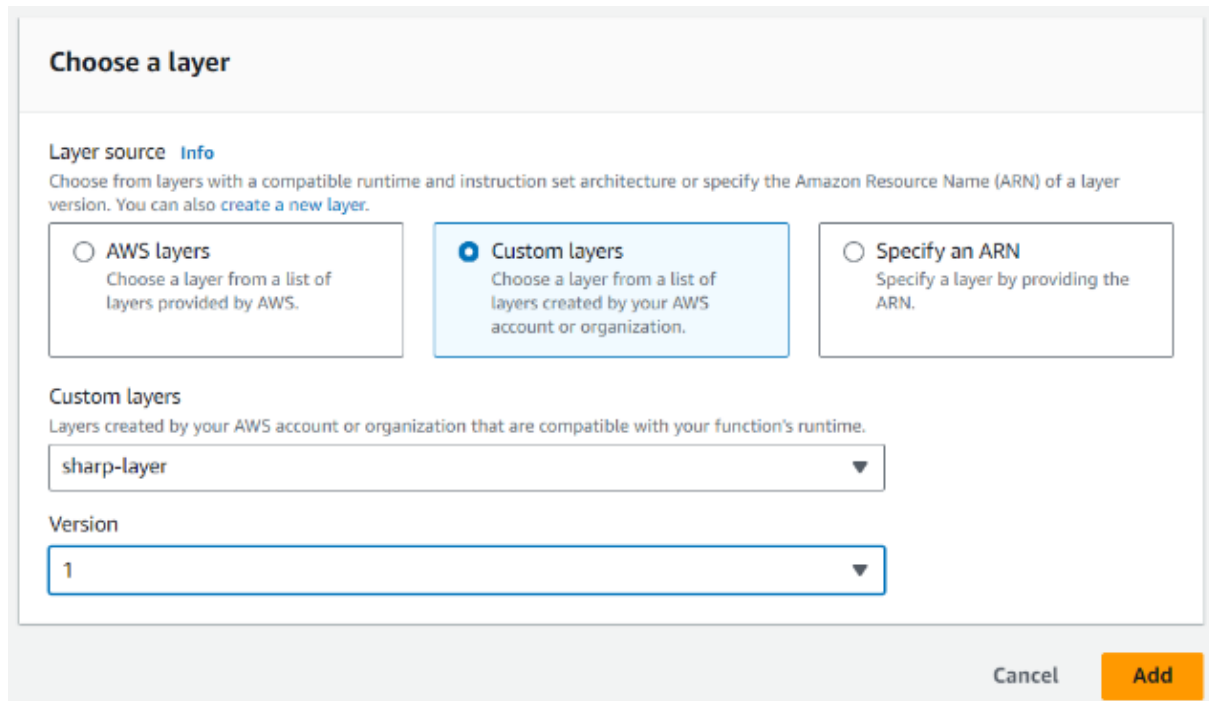
Compatible runtimes - optional [Info](#)

Choose up to 15 runtimes.

Runtimes

Nodejs 16.x

Now go to your lambda function page. In Layers section click on Add layer button. Select Custom Layer. Choose “sharp-layer”. Select version 1



The screenshot shows the 'Choose a layer' dialog box in the AWS Lambda console. At the top, it says 'Choose a layer' with a blue 'Info' link. Below this, it says 'Layer source' and 'Choose from layers with a compatible runtime and instruction set architecture or specify the Amazon Resource Name (ARN) of a layer version. You can also [create a new layer](#).' There are three radio button options: 'AWS layers' (Choose a layer from a list of layers provided by AWS.), 'Custom layers' (Choose a layer from a list of layers created by your AWS account or organization.), and 'Specify an ARN' (Specify a layer by providing the ARN.). The 'Custom layers' option is selected. Below the options, there is a section for 'Custom layers' with the text 'Layers created by your AWS account or organization that are compatible with your function's runtime.' There is a dropdown menu showing 'sharp-layer' and another dropdown menu showing 'Version 1'. At the bottom right, there are 'Cancel' and 'Add' buttons.

Step 5 – Creating S3 trigger

Now we need our Lambda function to know when an image is uploaded to the source bucket. We can do this by adding an event to the source S3 bucket and configure it to get triggered when an image is uploaded to the bucket which in turn invokes the Lambda function.

Go to S3 console. Select the source bucket (“serverless-bucket-uploaded-images”). Go to the Properties tab. Navigate to “Event Notifications”. Click “Create Event Notifications”.

Give an appropriate name to the event. Check the “All object create events”.

General configuration

Event name

Event name can contain up to 255 characters.

Prefix - optional
Limit the notifications to objects with key starting with specified characters.

Suffix - optional
Limit the notifications to objects with key ending with specified characters.

Event types
Specify at least one event for which you want to receive notifications. For each group, you can choose an event type for all events, or you can choose one or more individual events.

Object creation

☒ All object create events
s3:ObjectCreated:*

☐ Put
s3:ObjectCreated:Put

☐ --

Navigate to the “Destination” and select your lambda function. Save changes.

Destination

Before Amazon S3 can publish messages to a destination, you must grant the Amazon S3 principal the necessary permissions to call the relevant API to publish messages to an SNS topic, an SQS queue, or a Lambda function. [Learn more](#)

Destination
Choose a destination to publish the event. [Learn more](#)

☒ **Lambda function**
Run a Lambda function script based on S3 events.

☐ SNS topic
Fanout messages to systems for parallel processing or directly to people.

☐ SQS queue
Send notifications to an SQS queue to be read by a server.

Specify Lambda function

☒ Choose from your Lambda functions

☐ Enter Lambda function ARN

Lambda function

Cancel Save changes

Step 6 – Testing the application

Upload an image file to source S3 bucket (“serverless-bucket-uploaded-images”). Wait for few seconds and check the destination bucket (“serverless-bucket-processed-images”). There you will see two images (thumbnail and coverphoto). Congratulations, you just built a serverless Image processing application.

4. Security

Amazon S3 security best practices

The following best practices for Amazon S3 can help prevent security incidents.

Disable access control lists (ACLs)

S3 Object Ownership is an Amazon S3 bucket-level setting that you can use to control ownership of objects uploaded to your bucket and to disable or enable ACLs. By default, Object Ownership is set to the Bucket owner enforced setting and all ACLs are disabled. When ACLs are disabled, the bucket owner owns all the objects in the bucket and manages access to data exclusively using access management policies.

A majority of modern use cases in Amazon S3 no longer require the use of [access control lists \(ACLs\)](#). We recommend that you disable ACLs, except in unusual circumstances where you must control access for each object individually. To disable ACLs and take ownership of every object in your bucket, apply the bucket owner enforced setting for S3 Object Ownership. When you disable ACLs, you can easily maintain a bucket with objects uploaded by different AWS accounts.

When ACLs are disabled access control for your data is based on policies, such as the following:

- AWS Identity and Access Management (IAM) user policies
- S3 bucket policies
- Virtual private cloud (VPC) endpoint policies
- AWS Organizations service control policies (SCPs)

Disabling ACLs simplifies permissions management and auditing. ACLs are disabled for new buckets by default. You can also disable ACLs for existing buckets. If you have an existing bucket that already has objects in it, after you disable ACLs, the object and bucket ACLs are no longer part of the access-evaluation process. Instead, access is granted or denied on the basis of policies.

Before you disable ACLs, make sure that you do the following:

- Review your bucket policy to ensure that it covers all the ways that you intend to grant access to your bucket outside of your account.
- Reset your bucket ACL to the default (full control to the bucket owner).

After you disable ACLs, the following behaviors occur:

- Your bucket accepts only PUT requests that do not specify an ACL or PUT requests with bucket owner full control ACLs. These ACLs include the bucket-owner-full-control canned ACL or equivalent forms of this ACL that are expressed in XML.
- Existing applications that support bucket owner full control ACLs see no impact.
- PUT requests that contain other ACLs (for example, custom grants to certain AWS accounts) fail and return an HTTP status code 400 (Bad Request) with the error code `AccessControlListNotSupported`.

Ensure that your Amazon S3 buckets use the correct policies and are not publicly accessible

Unless you explicitly require anyone on the internet to be able to read or write to your S3 bucket, make sure that your S3 bucket is not public. The following are some of the steps that you can take to block public access:

- Use S3 Block Public Access. With S3 Block Public Access, you can easily set up centralized controls to limit public access to your Amazon S3 resources. These centralized controls are enforced regardless of how the resources are created. For more information, see [Blocking public access to your Amazon S3 storage](#).
- Identify Amazon S3 bucket policies that allow a wildcard identity such as "Principal": "*" (which effectively means "anyone"). Also look for policies that allow a wildcard action "*" (which effectively allows the user to perform any action in the Amazon S3 bucket).
- Similarly, look for Amazon S3 bucket access control lists (ACLs) that provide read, write, or full-access to "Everyone" or "Any authenticated AWS user."
- Use the ListBuckets API operation to scan all of your Amazon S3 buckets. Then use GetBucketAcl, GetBucketWebsite, and GetBucketPolicy to determine whether each bucket has compliant access controls and a compliant configuration.
- Use [AWS Trusted Advisor](#) to inspect your Amazon S3 implementation.
- Consider implementing ongoing detective controls by using the [s3-bucket-public-read-prohibited](#) and [s3-bucket-public-write-prohibited](#) managed AWS Config Rules.

Identify potential threats to your Amazon S3 buckets by using Amazon GuardDuty

[Amazon GuardDuty](#) is a threat detection service that identifies potential threats to your accounts, containers, workloads, and the data within your AWS environment. By using machine learning (ML) models, and anomaly and threat detection capabilities, Amazon GuardDuty continuously monitors different data sources to identify and prioritize

potential security risks and malicious activities in your environment. When you enable GuardDuty, it offers threat detection for foundational data sources that includes [AWS CloudTrail management events](#), VPC flow logs, and DNS logs. To extend threat detection to data plane events in S3 buckets, you can enable the [GuardDuty S3 Protection](#) feature. This feature detects threats such as data exfiltration and suspicious access to S3 buckets via Tor nodes. GuardDuty also establishes a normal baseline pattern in your environment and when it identifies a potentially anomalous behavior, it provides contextual information to help you remediate the potentially compromised S3 bucket or AWS credentials

Implement least privilege access

When granting permissions, you decide who is getting what permissions to which Amazon S3 resources. You enable specific actions that you want to allow on those resources. Therefore, we recommend that you grant only the permissions that are required to perform a task. Implementing least privilege access is fundamental in reducing security risk and the impact that could result from errors or malicious intent.

The following tools are available to implement least privilege access:

- [Policy actions for Amazon S3](#) and [Permissions Boundaries for IAM Entities](#)
- [How Amazon S3 works with IAM](#)
- [Access control list \(ACL\) overview](#)
- [Service Control Policies](#)

Use IAM roles for applications and AWS services that require Amazon S3 access

In order for applications running on Amazon EC2 or other AWS services to access Amazon S3 resources, they must include valid AWS credentials in their AWS API requests. We recommend not storing AWS credentials directly in the application or Amazon EC2 instance. These are long-term credentials that are not automatically rotated and could have a significant business impact if they are compromised.

Instead, use an IAM role to manage temporary credentials for applications or services that need to access Amazon S3. When you use a role, you don't have to distribute long-term credentials (such as a username and password or access keys) to an Amazon EC2 instance or AWS service, such as AWS Lambda. The role supplies temporary permissions that applications can use when they make calls to other AWS resources.

Consider encryption of data at rest

You have the following options for protecting data at rest in Amazon S3:

- **Server-side encryption** – All Amazon S3 buckets have encryption configured by default, and all new objects that are uploaded to an S3 bucket are automatically encrypted at rest. Server-side encryption with Amazon S3 managed keys (SSE-S3) is the default encryption configuration for every bucket in Amazon S3. To use a different type of encryption, you can either specify the type of server-side encryption to use in your S3 PUT requests, or you can set the default encryption configuration in the destination bucket.

Amazon S3 also provides these server-side encryption options:

- Server-side encryption with AWS Key Management Service (AWS KMS) keys (SSE-KMS)
 - Dual-layer server-side encryption with AWS Key Management Service (AWS KMS) keys (DSSE-KMS)
 - Server-side encryption with customer-provided keys (SSE-C)
- **Client-side encryption** – Encrypt data client-side and upload the encrypted data to Amazon S3. In this case, you manage the encryption process, the encryption keys, and related tools. As with server-side encryption, client-side encryption can help reduce risk by encrypting the data with a key that is stored in a different mechanism than the mechanism that stores the data itself.

Enforce encryption of data in transit

You can use HTTPS (TLS) to help prevent potential attackers from eavesdropping on or manipulating network traffic by using person-in-the-middle or similar attacks.

Important

We recommend that your application not pin Amazon S3 TLS certificates as AWS doesn't support pinning of publicly-trusted certificates. S3 automatically renews certificates and renewal can happen any time before certificate expiry. Renewing a certificate generates a new public-private key pair. If you've pinned an S3 certificate which has been recently renewed with a new public key, you won't be able to connect to S3 until your application uses the new certificate.

Consider using S3 Object Lock

With S3 Object Lock, you can store objects by using a "Write Once Read Many" (WORM) model. S3 Object Lock can help prevent accidental or inappropriate deletion of data.

Enable S3 Versioning

S3 Versioning is a means of keeping multiple variants of an object in the same bucket. You can use versioning to preserve, retrieve, and restore every version of every object stored in your bucket. With versioning, you can easily recover from both unintended user actions and application failures.

Consider using S3 Cross-Region Replication

Although Amazon S3 stores your data across multiple geographically diverse Availability Zones by default, compliance requirements might dictate that you store data at even greater distances. With S3 Cross-Region Replication (CRR), you can replicate data between distant AWS Regions to help satisfy these requirements. CRR enables automatic, asynchronous copying of objects across buckets in different AWS Regions.

Consider using VPC endpoints for Amazon S3 access

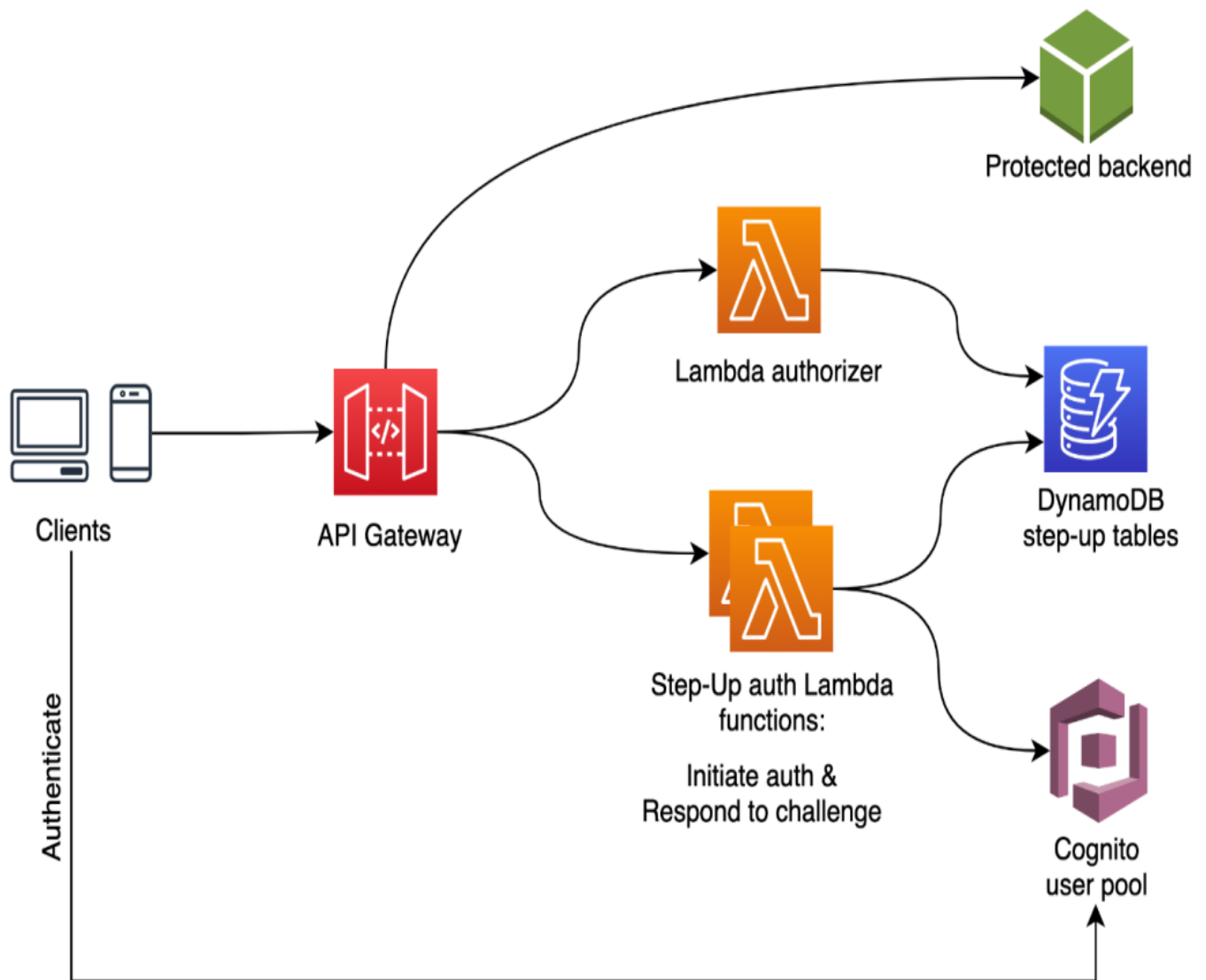
A virtual private cloud (VPC) endpoint for Amazon S3 is a logical entity within a VPC that allows connectivity only to Amazon S3. VPC endpoints can help prevent traffic from traversing the open internet.

VPC endpoints for Amazon S3 provide multiple ways to control access to your Amazon S3 data:

- You can control the requests, users, or groups that are allowed through a specific VPC endpoint by using S3 bucket policies.
- You can control which VPCs or VPC endpoints have access to your S3 buckets by using S3 bucket policies.
- You can help prevent data exfiltration by using a VPC that does not have an internet gateway.

Use managed AWS security services to monitor data security

Several managed AWS security services can help you identify, assess, and monitor security and compliance risks for your Amazon S3 data. These services can also help you protect your data from those risks. These services include automated detection, monitoring, and protection capabilities that are designed to scale from Amazon S3 resources for a single AWS account to resources for organizations spanning thousands of accounts.



6.CONCLUSION

Serverless image generation using AWS Services is a powerful and flexible solution for generating and delivering images in a serverless environment. By leveraging AWS services such as AWS Lambda, Amazon API Gateway, Amazon S3, and Amazon Cloud Front, you can create a highly scalable and cost-effective image generation pipeline that can handle large volumes of requests with minimal overhead.

Throughout this documentation, we've covered the key concepts and best practices for building a serverless image generation solution using AWS services. From creating an API Gateway to handling image transformations and caching, you should now have a solid understanding of how to build and deploy a serverless image generation pipeline.

