Serverless Image Handler

Implementation Guide

Serverless Image Handler: Implementation Guide

Table of Contents

Serverless Image Handler

AWS Solutions Implementation Guide

AWS Solutions Builder Team

Publication date: June 2017 (last update (p. 33): January 2021)

This implementation guide discusses architectural considerations and configuration steps for deploying the Serverless Image Handler solution. It includes links to an AWS CloudFormation template that launches, configures, and runs the AWS compute, network, storage, and other services required to deploy this solution on AWS, using AWS best practices for security and availability.

The guide is intended for IT infrastructure architects, administrators, and DevOps professionals who have practical experience architecting web applications in the AWS Cloud.

Overview

To help you embed images on your websites and mobile applications to drive user engagement, Amazon Web Services (AWS) offers the Serverless Image Handler solution. This solution combines highly available AWS services and the open source image processing suite Sharp to activate fast and cost-effective image manipulation in the AWS Cloud. This solution can decrease your costs of image optimization, manipulation, and processing by automating version control and providing flexible storage and compute options for file reprocessing.

This solution automatically deploys and configures a serverless architecture optimized for dynamic image manipulation. It uses Amazon CloudFront for global content delivery and Amazon Simple Storage Service (Amazon S3) for reliable and durable cloud storage at a low cost.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of the date of publication, the estimated cost for running the Serverless Image Handler solution with an AWS Lambda processing time of two seconds per image, where each image is approximately 1 MB in size, and default settings in the US East (N. Virginia) Region is **$15.29/month** for 100,000 new images, **$149.30/month** for 1,000,000 new images, and **$744.91/month** for 5,000,000 new images (refer to the following table for the cost breakdown). This includes estimated charges for Amazon API Gateway, AWS Lambda, Amazon CloudFront, Amazon S3 storage, and AWS Secrets Manager.

	Cost to process # of new images/month		
Amazon API Gateway	$0.35	$3.50	$17.50
AWS Lambda (2 seconds processing time per image)	$3.35	$33.53	$167.67
Amazon CloudFront (process 1 MB images per request)	$8.40	$84.01	$420.04
Amazon S3 (store 1 MB images)	$2.29	$22.86	$114.30
AWS Secrets Manager*	$0.90	$5.40	$25.40
Total monthly cost:	**$15.29**	**$149.30**	**$744.91**

**The cost for AWS Secrets Manager is incurred only when the image URL signature feature is activated.*

If you choose to deploy the demo user interface (p. 18), the solution automatically deploys an additional Amazon CloudFront distribution and Amazon S3 bucket for storing the static website assets in your account. You are responsible for the incurred variable charges from these services.

This cost estimate does not account for Amazon S3 PUT and GET requests, that can vary because modified images are cached in CloudFront, and because certain use cases require special-use capabilities

such as smart cropping with Amazon Rekognition. Using Amazon Rekognition features may incur additional charges. There is no additional cost for using Sharp, which is an open source library. Prices are subject to change. For full details, refer to the pricing webpage for each AWS service you will be using in this solution.

Architecture overview

Deploying this solution with the default parameters builds the following environment in the AWS Cloud.

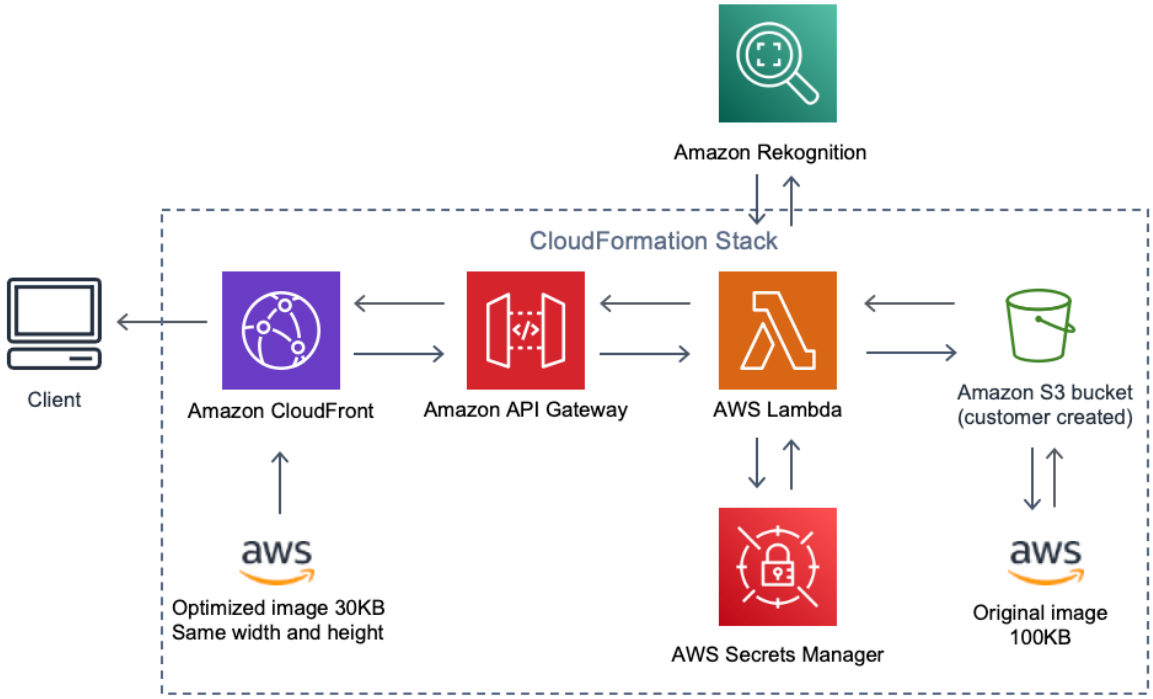

Figure 1: Serverless Image Handler architecture on AWS

> **Note**
> This solution is intended for customers with public applications who want to provide an option to dynamically change or manipulate their public images. Because of these public requirements, this template creates a publicly accessible, unauthenticated Amazon CloudFront distribution and Amazon API Gateway endpoint in your account, allowing anyone to access it. For more information on Amazon API Gateway authorization, refer to the Security (p. 6) section.

- The AWS CloudFormation template deploys a CloudFront distribution, Amazon API Gateway, 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 API Gateway provides endpoint resources and initiates the Lambda function.
- The Lambda function retrieves the image from a customer's 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.

If you activate the image URL signature feature, the Lambda function retrieves the secret value from your existing AWS Secrets Manager secret to validate the signature.

If you use the smart crop or content moderation features, the AWS Lambda function calls Amazon Rekognition to analyze your image and returns the results.

Note
AWS CloudFormation resources are created from AWS Cloud Development Kit (AWS CDK) components.

Solution components

In your front-end application, you can access both the original and modified images by creating an image request object, stringifying and encoding that object, and appending it to the path of the Amazon CloudFront URL as shown below.

```
https://distributionName.cloudfront.net/base64encodedrequest
```

Additional resources may be provisioned or used depending on whether the following optional features are activated:

- **Demo UI:** An optional demo user interface (UI) is deployed into your account to demonstrate the basic features of the solution. This UI allows you to interact directly with your new image handler API endpoint using image files that already exist in your account. If selected, this option deploys an additional Amazon S3 bucket and associated CloudFront distribution into your account.
- **Smart Cropping:** An image request option that allows you to crop images using the facial recognition capabilities of Amazon Rekognition. To generate a cropped image, the AWS Lambda function sends requests to Amazon Rekognition to identify faces in images and calculate crop areas.
- **Content Moderation:** An image request option that allows you to detect and blur inappropriate images. To detect an inappropriate image, the AWS Lambda function sends requests to Amazon Rekognition to identify inappropriate content.

> **Note**
> Amazon Rekognition supports only JPEG and PNG file formats for Smart Cropping and Content Moderation. As a result, these features are compatible with images containing the JPEG and PNG extensions only and do not support files without an extension.

Security

When you build systems on AWS infrastructure, security responsibilities are shared between you and AWS. This shared model can reduce your operational burden as AWS operates, manages, and controls the components from the host operating system and virtualization layer down to the physical security of the facilities in which the services operate. For more information about security on AWS, visit AWS Cloud Security.

Important
This solution creates Amazon CloudFront and Amazon API Gateway resources that are publicly accessible. Be aware that while this is likely appropriate for publicly facing websites, it may not be appropriate for all customer use cases for this solution. AWS offers several options for end-to-end security, such as AWS Identity and Access Management (IAM), Amazon Cognito User Pools, AWS Certificate Manager, and Amazon CloudFront signed URLs. For private image handling use cases, AWS recommends using signed URLs with Amazon CloudFront and implementing an Amazon API Gateway Lambda authorizer with Amazon CloudFront to secure your stack.

Demo user interface

This solution deploys a demo UI as a static website hosted in an Amazon Simple Storage Service (Amazon S3) bucket. To help reduce latency and improve security, this solution includes an Amazon CloudFront distribution with an origin access identity, which is a special CloudFront user that helps restrict access to the solution's website bucket contents. For more information, refer to Restricting Access to Amazon S3 Content by Using an Origin Access Identity in the *Amazon CloudFront Developer Guide*.

Implementation considerations

Cross-Origin Resource Sharing (CORS)

The solution's template contains two parameters: **CorsEnabled** and **CorsOrigin** that activate Cross-Origin Resource Sharing (CORS) for your image handler API. CORS defines how client web applications loaded in one domain can interact with resources in a different domain. You can use CORS support to make requests to your image handler API from outside the domain space of the API.

For example, if you have a public web application hosted on either a custom domain or a cloud domain outside of AWS, you can activate CORS to fetch original or modified images from the image handler API.

If you would like to change your CORS configuration after deployment, you can activate or deactivate CORS by editing the `CorsEnabled (Yes/No)` and `CorsOrigin` environment variables of the AWS Lambda image handler function.

AWS Lambda quotas

AWS Lambda has a 6 MB invocation payload request and response limit. For information about AWS Lambda quotas for the amount of compute and storage resources that you can use to run and store functions, refer to AWS Lambda quotas in the *AWS Lambda Developer Guide*.

Amazon API Gateway quotas

Amazon API Gateway sets the maximum integration timeout at 30 seconds for all integration types, including AWS Lambda. Processing large image files can result in a timeout error due to the maximum integration timeout being exceeded. For information about Amazon API Gateway quotas refer to Amazon API Gateway quotas in the *Amazon API Gateway Developer Guide*.

Image URL signature

This solution's template contains three parameters that are required for the image URL signature functionality: `EnableSignature`, `SecretsManagerSecret`, and `SecretsManagerKey`. To activate this feature, set the `EnableSignature` parameter to `Yes`, and set the `SecretsManagerSecret` and `SecretsManagerKey` parameters to a valid secret and key that you originally created in AWS Secrets Manager.

> **Important**
> You are responsible for creating the AWS Secrets Manager secret and key. For more information about AWS Secrets Manager secret creation, refer to Create a secret in the *AWS Secrets Manager User Guide*.

When you activate this feature, the image handler AWS Lambda function checks for a valid signature in the image request. If the signature does not match, an error message is returned. When activating the image URL signature, you must provide the `signature` query string to your URL. For example, you can create the signature using the following Node.js code:

```
const secret = '<YOUR_SECRET_VALUE_IN_SECRETS_MANAGER>';
const path = '/<YOUR_PATH>'; // Add the first '/' to path.
const signature = crypto.createHmac('sha256', secret).update(path).digest('hex');
```

You can request your image using the image URL signature.

```
https://<distributionName>.cloudfront.net/<YOUR_PATH>?signature=<YOUR_SIGNATURE>
```

Note
If you update your existing solution deployment and activate the image URL signature, the updated stack will no longer be compatible with the existing URLs. You must update your application to provide the correct signature query string to your URLs. To update the solution stack, refer to Update the main stack (p. 8).

Default fallback image

This solution provides a default fallback image feature that returns the specified fallback image as a result if errors occur during processing, rather than a JSON object error message. Be default, this feature is deactivated. To activate this feature, set the `EnableDefaultFallbackImage` parameter to `Yes`, and set the `FallbackImageS3Bucket` and `FallbackImageS3Key` parameters to a valid Amazon Simple Storage Service (Amazon S3) bucket and object key.

This solution's template contains three parameters that are required for the default fallback image feature: `EnableDefaultFallbackImage`, `FallbackImageS3Bucket`, and `FallbackImageS3Key`. Before activating this feature, if you use an Amazon S3 bucket policy in the fallback image Amazon S3 bucket, you must edit the bucket policy to allow `CustomResourceFunction` and `ImageHandlerFunction` AWS Lambda functions to get the default fallback image object.

Opt-In Regions

An opt-in Region is an AWS Region that is deactivated by default. Opt-in Regions can be activated in the AWS console. For additional information about opt-in Regions and how to activate them, refer to Managing AWS Regions in the *AWS General Reference guide*.

This solution supports two opt-in Regions: Hong Kong (ap-east-1) and Bahrian (me-south-1). When launching in an opt-in Region, this solution creates an Amazon S3 logging bucket for Amazon CloudFront in the N. Virginia (us-east-1) Region. The S3 bucket is launched in N. Virginia (us-east-1) because CloudFront does not currently deliver access logs to buckets in Hong Kong (ap-east-1) or Bahrain (me-south-1). For more information about Amazon S3 buckets, refer to Choosing an Amazon S3 bucket in the *Amazon CloudFront Developer Guide*.

In order to deploy in an opt-in Region, the source S3 bucket that you provide must be in the same Region where you are launching the AWS CloudFormation template.

Solution updates

Update the main stack

Complete the following steps to update your AWS CloudFormation stack to the current version.

1. From the main account where the Serverless Image Handler template is deployed, sign in to the AWS CloudFormation console.
2. From the **Stacks** page, select this solution's stack and choose **Update**.
3. On the **Update stack** page, verify that **Replace current template** is selected.
    1. In the **Specify template** section, select **Amazon S3 URL**.
    2. Copy the link of the latest template.
    3. Paste the link in the **Amazon S3 URL** box.
    4. Verify that the correct template URL shows in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, under **Parameters**, review the parameters for the template and modify them as necessary. Refer to Step 1. Launch the Stack (p. 11) for details about the parameters.
5. Choose **Next**.
6. On the **Configure stack options** page, choose **Next**.
7. On the **Review** page, review and confirm the settings. Check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
8. Choose **View change set** and verify the changes.
9. Choose **Update stack** to deploy the stack.

You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive an **UPDATE_COMPLETE** status in approximately 15 minutes.

About Node.js versions

Serverless Image Handler v4.2 and later versions use the most up-to-date Node.js runtime. Version 4.0 and earlier versions use the Node.js 8.10 runtime, which reached end-of-life on December 31, 2019. As a result, AWS Lambda now blocks both the create operation, and the update operation. For more information, refer to Runtime Support Policy in the *AWS Lambda Developer Guide*.

To continue using this solution with the latest features and improvements, update the stack to the current version.

AWS CloudFormation template

This solution uses AWS CloudFormation to automate the deployment of the Serverless Image Handler solution in the AWS Cloud. It includes the following AWS CloudFormation template, which you can download before deployment:

serverless-image-handler.template: As of January 2021, the latest template is version 5.2.0. Use this template to launch the Serverless Image Handler and all associated components. The default configuration deploys Amazon CloudFront, Amazon API Gateway, AWS Lambda, and Amazon Simple Storage Service.

Automated deployment

Follow the step-by-step instructions in this section to configure and deploy the Serverless Image Handler into your account.

Time to deploy: Approximately 15 minutes

Prerequisites

Before you launch the solution's AWS CloudFormation template, you must specify an Amazon Simple Storage Service (Amazon S3) bucket in the **SourceBuckets** template parameter. Use this S3 bucket to store the images you want to manipulate. If you have multiple image source S3 buckets, you can specify them as comma-separated values. For lower latency, use an S3 bucket in the same AWS Region where you launch your AWS CloudFormation template.

> **Note**
> If you are launching from either the Hong Kong (ap-east-1) or Bahrain (me-south-1) Region, the source S3 bucket you created and provided as the **SourceBuckets** template parameter must be in the same Region where you are launching the CloudFormation template.

We recommend deploying the optional demo user interface when you first deploy the solution to test the solution's functionality. For more information, refer to Using the demo UI (p. 18).

Deployment overview

Deploying this architecture on AWS includes the following steps. For detailed instructions, follow the links for each step.

Step 1. Launch the stack (p. 11)

- Launch the AWS CloudFormation template into your AWS account.
- Enter values for required parameters: **CorsEnabled**, **CorsOrigins**, **SourceBuckets**, **DeployDemoUI**, **LogRetentionPeriod**, **EnableSignature**, **EnableDefaultFallbackImage**, and **AutoWebP**.
- Review the other template parameters, and adjust if necessary.

Step 2. Create and use image requests (p. 14)

- Set up an image request on the front-end.
- Send an image request to your API.

Step 1. Launch the stack

This automated AWS CloudFormation template deploys the Serverless Image Handler solution in the AWS Cloud.

Note
You are responsible for the cost of the AWS services used while running this solution. Refer to the Cost (p. 2) section for more details. For full details, refer to the pricing webpage for each AWS service you will be using in this solution.

1. Log in to the AWS Management Console and select the button to launch the `serverless-image-handler` AWS CloudFormation template.

   You can also download the template as a starting point for your own implementation.
2. The template is launched in the US East (N. Virginia) Region by default. To launch the Serverless Image Handler in a different AWS Region, use the Region selector in the console navigation bar.
3. On the **Create stack** page, verify that the correct template URL shows in the **Amazon S3 URL** text box and choose **Next**.
4. On the **Specify stack details** page, assign a name to your solution stack.
5. Under **Parameters**, review the parameters for the template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
CorsEnabled	`No`	Choose whether to activate Cross-Origin Resource Sharing (CORS). For information about this parameter, refer to Cross-Origin Resource Sharing (p. 7).
CorsOrigin	`*`	This value is returned by the API in the Access-Control-Allow-Origin header. An asterisk (*) value supports any origin. We recommend specifying a specific origin (e.g. http://example.domain) to restrict cross-site access to your API. **Note** This value is ignored if the **CorsEnabled** parameter is set to `No`.
SourceBuckets	*`<Requires input>`*	Specifies the S3 bucket (or buckets) in your account that contains the images that you manipulate. To specify multiple buckets, separate them by commas.
DeployDemoUI	`Yes`	The demo UI that deploys to the `Demo` S3 bucket. For more information refer to Using the demo UI (p. 18).

Parameter	Default	Description
LogRetentionPeriod	`1`	Specifies the number of days to retain Lambda log data in CloudWatch logs.
EnableSignature	`No`	Choose whether to activate the image URL signature feature. For information about this feature, refer to Image URL signature (p. 7).
SecretsManagerSecret	<Optional input>	Define the AWS Secrets Manager secret name that contains the secret key for the image URL signature. **Note** This value is ignored if the **EnableSignature** parameter is set to `No`.
SecretsManagerKey	<Optional input>	Define the AWS Secrets Manager secret key that contains the secret value to create the image URL signature. **Note** This value is ignored if the **EnableSignature** parameter is set to `No`.
EnableDefaultFallbackImage	`No`	Choose whether to activate the default fallback image feature. For information about this feature, refer to Default fallback image (p. 8).
FallbackImageS3Bucket	<Optional input>	Specify the Amazon S3 bucket which contains the default fallback image. **Note** This value is ignored if the **EnableDefaultFallbackImage** parameter is set to `No`.
FallbackImageS3Key	<Optional input>	Specify the default fallback image Amazon S3 object key including prefix. **Note** This value is ignored if the **EnableDefaultFallbackImage** parameter is set to `No`.

Parameter	Default	Description
AutoWebP	`No`	Choose whether to automatically accept webp image formats.

6. Choose **Next**.
7. On the **Configure stack options** page, choose **Next**.
8. On the **Review** page, review and confirm the settings. Check the box acknowledging that the template creates AWS Identity and Access Management (IAM) resources.
9. Choose **Create stack** to deploy the stack.

   You can view the status of the stack in the AWS CloudFormation console in the **Status** column. You should receive a **CREATE_COMPLETE** status in approximately 15 minutes.

Step 2. Create and use image requests

This solution generates a CloudFront domain name that gives you access to both original and modified images via the image handler API. The domain name is found in the **Outputs** section of the CloudFormation template as an **ApiEndpoint**. Parameters such as the image's location and edits to be made are specified in a JSON object on the front-end.

For example, the following code block specifies the image location as **myImageBucket** and specifies edits of **grayscale: true** to change the image to grayscale.

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>"
    key: "<myImage>.jpg",
    edits: {
        grayscale: true
    }
})
```

Use the following procedure to create image requests:

1. In the AWS CloudFormation Management Console, choose the **Outputs** tab and make a note of the URL that appears next to **ApiEndpoint**. This URL is the endpoint URL for your newly provisioned image handler API.
2. In a code sandbox, or in your front-end application, create a new JSON object. This object contains the key-value pairs needed to successfully retrieve and perform edits on your images.
3. Using the code sample above and the Sharp documentation, adjust the following properties to meet your image editing requirements.
   - **Bucket** – Specify the Amazon S3 bucket containing your original image file. This is the name that is specified in the **SourceBuckets** template parameter. You can update the image location by adding it into the `SOURCE_BUCKETS` environment variable of your image handler AWS Lambda function.
   - **Key** – Specify the filename of your original image. This name should include the file extension as well as any subfolders between its location and the root of the bucket. For example, `folder1/folder2/image.jpg`.
   - **Edits** – Specify any image edits as key-value pairs. If you do not specify image edits, the original image returns with no changes made.
4. Stringify and encode your image request. You can use JavaScript's `JSON.stringify()` property, followed by encoding the result using the `btoa()` property.
5. Append the encoded result to your ApiEndpoint URL and use this as the value for the HTML **img src** property or in a GET request. Refer to the following example.

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>"
    key: "<myImage>.jpg",
    edits: {
        grayscale: true
    }
});
const url = `${CloudFrontUrl}/${btoa(imageRequest)}`;

// Alternatively, you can call the url directly in an <img> element, similar to:
<img src=`${url}` />
```

The following is an example of the preceding code results in an encoded image request:

`https://<distributionName>.cloudfront.net/<base64encodedRequest>`

For information regarding how to use additional features in an image request, refer to Smart cropping (p. 19), Round cropping (p. 20), and Content moderation (p. 21). For additional features supported by Sharp, refer to the Sharp documentation.

Additional resources

AWS services

- AWS CloudFormation
- AWS Lambda
- Amazon CloudFront
- Amazon API Gateway
- Amazon Rekognition
- AWS Identity and Access Management
- Amazon S3
- AWS Cloud Development Kit

Image handler

- Sharp

Backward compatibility

The Serverless Image Handler solution is compatible with legacy image request formats, including the Thumbor and Custom (with rewrite function) formats from previous versions of this solution. If you are using a previous version of this solution (version 3.x and earlier) and have image requests formatted for use with that version, review the following notes to ensure minimal breaking changes or parities.

> **Note**
> Legacy requests (Thumbor and Custom) are currently limited to: sourcing images from the root level of Amazon S3 buckets and sourcing original images from the first bucket only in the **SOURCE_BUCKETS** environment variable. You can adjust this in the environment variables section of your image handler AWS Lambda function. For example: `SOURCE_BUCKETS: “my-bucket-001, my-bucket-002, my-bucket-003”`.

Thumbor

Thumbor image requests can be specified as normal, with filters and other relevant properties added on as suffixes to the default CloudFront ApiEndpoint. For example:

```
https://<distName>.cloudfront.net/filters:grayscale()/image.png
```

Custom

Custom image requests that used the previous solution versions rewrite feature can also be specified as normal. Note that the **REWRITE_MATCH_PATTERN** and **REWRITE_SUBSTITUTION** environment variables for your image handler function must be updated with the appropriate (JavaScript/ECMAScript-compatible) regular expressions and strings.

```
https://<distName>.cloudfront.net/<customRequestHere>
```

Using the demo UI

The solution provides an optional demo user interface that you can deploy into your AWS account to display basic capability and functionality. This user interface (UI) allows you to interact directly with the new handler using images from the specified Amazon Simple Storage Service (Amazon S3) buckets in your account.

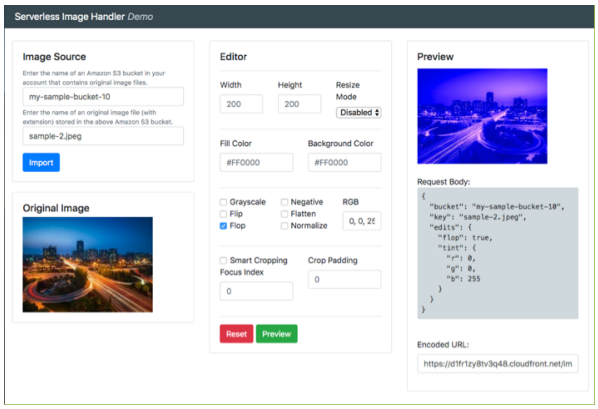

Figure 2: Serverless Image Handler demo UI

Follow this procedure to experiment with the supported image editing features, preview the results, and create example URLs that you can use in your applications:

1. In the AWS CloudFormation stack **Outputs** tab, select the **DemoUI URL**. The Serverless Image Handler demo UI opens in a new tab.
2. On the **Image Source** card, specify a bucket name and image key to use for the demo. You must include the file extension in the key, and the bucket you specify must be listed in the **SOURCE_BUCKETS** environment variable of the AWS Lambda function.
3. Select **Import**. The original image appears in the Original Image card.
4. In the **Editor** section, adjust the image settings and select **Preview** to generate the modified image. You can select **Reset** to revert the settings back to their original values.

> **Note**
> The Serverless Image Handler demo UI offers a limited set of image edits and does not include the full scope of capabilities offered by the Image Handler API and the image URL signature. We recommended using your own front-end application for image modification.

Smart cropping with Amazon Rekognition

The Serverless Image Handler solution leverages Amazon Rekognition for face detection in images submitted for smart cropping.

Image request use

To activate smart cropping on an image, add the **smartCrop** property to the **edits** property in the image request (p. 14).

- **smartCrop:** (optional, boolean || object) activates the smart cropping feature for an original image. If the value is `true`, the feature returns the first face detected from the original image with no additional options.

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>"
    key: "<myImage>.jpg",
    edits: {
        smartCrop: true
    }
})
```

- The following smartCrop variables are shown in the code sample:
  - **smartCrop.faceIndex:** (optional, number) specifies which face to focus on if multiple are present within an original image. Detected faces are indexed in a zero-based array from the largest detected face to the smallest. If this value is not specified, Amazon Rekognition returns the largest face detected from the original image.
  - **smartCrop.padding:** (optional, number) specifies an amount of padding in pixels to add around the cropped image. The padding value is applied to all sides of the cropped image. Additionally, the extend properties of the image handler can be used to apply more specific padding adjustments to the cropped image.

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>"
    key: "<myImage>.jpg",
    edits: {
        smartCrop: {
            faceIndex: 1    // zero-based index of detected faces
            padding: 40     // padding expressed in pixels, applied to all sides
        }
    }
})
```

Round cropping

The Serverless Image Handler solution can crop images in a circular pattern.

Image request use

To activate round cropping on an image, add the **roundCrop** property to the **edits** property in the image request (p. 14).

- **roundCrop:** (optional, boolean || object) activates the round cropping feature for an original image. If the value is `true`, the feature returns a circular cropped image that is centered from the original image and has a diameter of the smallest edge of the original image.

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>"
    key: "<myImage>.jpg",
    edits: {
        roundCrop: true
    }
})
```

- The following **roundCrop** variables are shown in the code sample:
  - **roundCrop.rx:** (optional, number) specifies the radius along the x-axis of the ellipse. If a value is not provided, defaults to a value that is half the length of the smallest edge.
  - **roundCrop.ry:** (optional, number) specifies the radius along the y-axis of the ellipse. If a value is not provided, defaults to a value that is half the length of the smallest edge.
  - **roundCrop.top:** (optional, number) specifies the offset from the top of the original image to place the center of the ellipse. If a value is not provided, defaults to a value that is half of the height.
  - **roundCrop.left:** (optional, number) specifies the offset from the left-most edge of the original image to place the center of the ellipse. If a value is not provided, defaults to a value that is half of the width.

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>"
    key: "<myImage>.jpg",
    edits: {
        roundCrop: {
            rx: 30   // x-axis radius
            ry: 20   // y-axis radius
            top: 300    // offset from top edge of orginal image
            left: 500    // offset from left edge of original image
        }
    }
})
```

Content moderation with Amazon Rekognition

The Serverless Image Handler solution can detect inappropriate content using Amazon Rekognition.

Image request use

To activate content moderation, add the **contentModeration** property to the **edits** property in the image request (p. 14).

- **contentModeration:** (optional, boolean || object) activates the content moderation feature for an original image. If the value is `true`, the feature detects inappropriate content using Amazon Rekognition with a minimum confidence that is set above 75%. If inappropriate content is found, the image is blurred.

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>"
    key: "<myImage>.jpg",
    edits: {
        contentModeration: true
    }
})
```

- The following **contentModeration** variables are shown in the code sample:
  - **contentModeration.minConfidence:** (optional, number) specifies the minimum confidence level for Amazon Rekognition to use. Amazon Rekognition only returns detected content that is above the minimum confidence. If a value is not provided, the default value is set to 75%.
  - **contentModeration.blur:** (optional, number) specifies the intensity level that an image is blurred if inappropriate content is found. The number represents the sigma of the Gaussian mask, where sigma = 1 + radius /2. For more information, refer to the Sharp documentation. If a value is not provided, the default value is set to 50.
  - **contentModeration.moderationLabels:** (optional, number) identifies the specific content to search for. The image is blurred only if Amazon Rekognition locates the content specified in the **smartCrop.moderationLabels** provided. You can use either a top-level category or a second-level category. Top-level categories include its associated second-level categories. For more information about moderation label options, refer to Content moderation in the *Amazon Rekognition Developer Guide*.

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>"
    key: "<myImage>.jpg",
    edits: {
        contentModeration: {
            minConfidence: 90   // minimum confidence level for inappropriate content
            blur: 80            // amount to blur image
            moderationLabels: [ // labels to search for
                “Hate Symbols”,
                “Smoking”
                ]
        }
    }
```

```
})
```

List of supported Thumbor filters

This solution currently supports the filters listed in the table below. To use the filters, append your CloudFront URL using the following syntax, including the image name (*<example>*). To use multiple filters on an image, list them in the same section of the URL. Example:

```
https://<yourcloudfronturl>/fit-in/300x400/filters:fill(00ff00)/
filters:rotate(90)/<example>.jpg
```

Filters process the image in the order they are specified.

> **Note**
> Some Thumbor filters are not supported in the current version of Serverless Image Handler. This may affect legacy users with advanced image request configurations. For examples of filter usage, refer to the Thumbor documentation.

Filter Name	Filter Syntax
Autojpg	/filters:autojpg()/
Background color	/filters:background_color(color)/
Blur	/filters:blur(7)/
Color fill	/filters:fill(color)/
Convolution	/filters:convolution(1;2;1;2;4;2;1;2;1,3,false)/
Equalize	/filters:equalize()/
Grayscale	/filters:grayscale()/
Image format (heic, heif, jpeg, png, raw, tiff, webp)	/filters:format(image_format)
Image type (jpeg, png, gif)	/filters:format(jpeg)/
No upscale	/filters:no_upscale()/
Proportion	/filters:proportion(0.0-1.0)/
Quality	/filters:quality(0-100)/
Resize	/fit-in/800x1000/
RGB	/filters:rgb(20,-20,40)/
Rotate	/filters:rotate(90)/
Sharpen	/filters:sharpen(0.0-10.0, 0.0-2.0, true/false)/
Stretch	/filters:stretch()/
Strip Exif	/filters:strip_exif()/

Filter Name	Filter Syntax
Strip ICC	/filters:strip_icc()/
Upscale	/filters:upscale()
Watermark	/filters:watermark(bucket,key,x,y,alpha[,w_ratio[,h_ratio]])

Image handler function environmental variables

Most settings and customizations to the Serverless Image Handler solution can be made by editing and updating the environment variables associated with the image handler AWS Lambda function.

The image handler function can be found in the AWS Management Console using one of the following methods:

- In the AWS Lambda console, the image handler function is listed with the following naming convention: `<StackName>-ImageHandlerFunction-<UniqueId>`.
- In the AWS CloudFormation console, the image handler function is listed under the Resources tab of your deployed stack with a Logical ID of `ImageHandlerFunction`.

After opening the Lambda function, scroll down to the **Environment variables** section. Use the following key-value pairs to customize the solutions settings:

Variable Key	Value Type	Description
AUTO_WEPB	`Yes/No`	Choose whether to automatically accept webp image formats.
CORS_ENABLED	`Yes/No`	Indicates whether to return an Access-Control-Allow-Origin header with the image handler API response.
CORS_ORIGIN	`String`	This value is returned by the API in the Access-Control-Allow-Origin header. An asterisk (*) value supports any origin. We recommend specifying a specific origin (e.g. http://example.domain) to restrict cross-site access to your API. **Note** This value is ignored if **CORS_ENABLED** is set to `No`.
ENABLE_DEFAULT_FALLBACK_IMAGE	`Yes/No`	Choose whether to return the default fallback image when errors occur.
DEFAULT_FALLBACK_IMAGE_BUCKET	`String`	Specifies the Amazon S3 bucket which contains the default fallback image. **Note** This value is ignored if the

Variable Key	Value Type	Description
		ENABLE_DEFAULT_FALLBACK_IMAGE parameter is set to `No`.
DEFAULT_FALLBACK_IMAGE_KEY	`String`	Defines the default fallback image Amazon S3 object key including prefix. **Note** This value is ignored if the **ENABLE_DEFAULT_FALLBACK_IMAGE** parameter is set to `No`.
ENABLE_SIGNATURE	`Yes/No`	Choose whether to use the image URL signature.
REWRITE_MATCH_PATTERN	`Regex`	By default, this parameter is empty. Contains a JavaScript-compatible regular expression for matching custom image requests using the rewrite function. It should match the JavaScript compatible regular expression. For example, `/(filters-)/gm`
REWRITE_SUBSTITUTION	`String`	By default, this parameter is empty. Contains a substitution string for custom image requests using the rewrite function. For example, `filters:`
SECRETS_MANAGER	`String`	Defines the AWS Secrets Manager secret that contains the secret key for the image URL signature. **Note** This value is ignored if **ENABLE_SIGNATURE** is set to `No`.
SECRET_KEY	`String`	Defines the AWS Secrets Manager secret key that contains the secret value to create the image URL signature. **Note** This value is ignored if **ENABLE_SIGNATURE** is set to `No`.

Variable Key	Value Type	Description
SOURCE_BUCKETS	`String/Regex`	The S3 bucket (or buckets) in your account that contains the original images. If providing multiple buckets, separate them by commas. Regular expression can be used as bucket prefix for multiple buckets.

Rewrite feature

This feature allows you to migrate your current image request model to the Serverless Image Handler solution, without changing the applications to accommodate new image URLs. This feature requires that you populate the following environment variables in the image handler function. These environment variables are added to the function by default, but are left empty for user input if the rewrite feature is needed.

Variable Key	Value Type	Description
REWRITE_MATCH_PATTERN	`Regex`	By default, this parameter is empty. Contains a JavaScript-compatible regular expression for matching custom image requests using the rewrite function. It should match the JavaScript compatible regular expression. For example, `/(filters-)/gm`
REWRITE_SUBSTITUTION	`String`	By default, this parameter is empty. Contains a substitution string for custom image requests using the rewrite function. For example, `filters:`

The rewrite feature translates custom URL image requests into Thumbor-consumable formats, based on JavaScript-compatible regular expression match patterns and substitution strings. After the image request is converted into Thumbor-consumable form, it is then processed as a Thumbor image request and edits are mapped to the new Sharp image library.

Uninstall the solution

You can uninstall the Serverless Image Handler solution from the AWS Management Console, or by using the AWS Command Line Interface (AWS CLI). You must manually delete the Amazon Simple Storage Service (Amazon S3) buckets created by this solution. AWS Solutions Implementations do not automatically delete this resource in case you have stored data to retain.

Using the AWS Management Console

1. Sign in to the AWS CloudFormation console.
2. On the **Stacks** page, select the solution stack.
3. Choose **Delete**.

Using AWS Command Line Interface

Verify that the AWS Command Line Interface (AWS CLI) is available in your environment. For installation instructions, refer to What Is the AWS Command Line Interface in the *AWS CLI User Guide*. After confirming the AWS CLI is available, run the following command.

```
$ aws cloudformation delete-stack --stack-name <your-stack-name>
```

Deleting the Amazon S3 buckets

This solution is configured to retain the solution-created Amazon S3 bucket (for deploying in an opt-in Region) if you decide to delete the AWS CloudFormation stack to prevent against accidental data loss. After uninstalling the solution, you can manually delete this S3 bucket if you do not need to retain the data. Follow these steps to delete the Amazon S3 bucket.

1. Sign in to the Amazon S3 console.
2. Choose **Buckets** from the left navigation pane.
3. Locate the *<stack-name>* S3 buckets.
4. Select the S3 bucket and choose **Delete**.

To delete the S3 bucket using AWS CLI, run the following command:

```
$ aws s3 rb s3://<bucket-name> --force
```

Alternatively, you can configure the AWS CloudFormation template to delete the Amazon S3 bucket automatically. Before deleting the stack, change the deletion behavior in the AWS CloudFormation DeletionPolicy attribute. This will not delete the source bucket you created and provided as a parameter to the CloudFormation template.

Collection of operational metrics

This solution includes an option to send anonymous operational metics to AWS. We use this data to better understand how customers use this solution and related services and products. When activated, the following information is collected and sent to AWS each time the AWS Lambda function runs:

- **Solution ID:** The AWS solution identifier
- **Unique ID (UUID):** Randomly generated, unique identifier
- **Timestamp:** The timestamp when the solution's Lambda function runs
- **Version:** The Serverless Image Handler solution version
- **Region:** The AWS Region the solution is being deployed in
- **EnableSignature:** Whether the image URL signature is activated
- **EnableDefaultFallbackImage:** Whether the default fallback image is activated

Note that AWS owns the data gathered via this survey. Data collection is subject to the AWS Privacy Policy. To opt out of this feature, complete the following task.

a) Modify the AWS CloudFormation template mapping section as follows:

```
"Send" : {
    "AnonymousUsage" : { "Data" : "Yes" }
},
```

to

```
"Send" : {
    "AnonymousUsage" : { "Data" : "No" }
},
```

Source code

Visit our GitHub repository to download the source files for this solution and to share your customizations with others. Additionally, if you require an earlier version of the CloudFormation template, you can request from the GitHub issues page. The Serverless Image Handler templates are generated using the AWS Cloud Development Kit (AWS CDK). Refer to the README.md file for additional information.

Contributors

The following individuals contributed to this document:

- Ryan Hayes
- Beomseok Lee
- George Lenz

Revisions

Date	Change
June 2017	Initial release
August 2017	Solution updated to add the rewrite feature and the optional deployment of a demo UI
October 2017	Solution updated to provide CORS support
September 2018	Added information on watermarking, URL encoding, debugging, and troubleshooting
December 2018	Added information about the Amazon CloudFront distribution for the static website hosted in the Amazon S3 bucket
January 2019	Added information about using the demo UI, safe URLs, and customizing the Thumbor Lambda package
June 2019	Added support for Sharp, multiple image sources, basic image editing, smartcropping with Amazon Rekognition, backward compatibility, and refresh of demo UI
August 2019	Updated the list of supported Thumbor filters
December 2019	Added information on support for Node.js update
February 2020	Added watermark support for Thumbor filter; added AutoWebP parameter for viewing webp image formats automatically
August 2020	Updated the AWS CloudFormation template; for more information, refer to the CHANGELOG.md file in the GitHub repository
November 2020	Added the image URL signature and the default fallback image features; for more information, refer to the CHANGELOG.md file in the GitHub repository
January 2021	Release v5.2.0: Added content moderation, round crop, and support for opt-in Regions; for more information on new features, refer to the CHANGELOG.md file in the GitHub repository

Notices

Customers are responsible for making their own independent assessment of the information in this document. This document: (a) is for informational purposes only, (b) represents AWS current product offerings and practices, which are subject to change without notice, and (c) does not create any commitments or assurances from AWS and its affiliates, suppliers or licensors. AWS products or services are provided “as is” without warranties, representations, or conditions of any kind, whether express or implied. AWS responsibilities and liabilities to its customers are controlled by AWS agreements, and this document is not part of, nor does it modify, any agreement between AWS and its customers.

Serverless Image Handler is licensed under the terms of the of the Apache License Version 2.0 available at The Apache Software Foundation.