Serverless Image Handler

AWS Implementation Guide

Ryan Hayes

June 2017

Last update: August 2020 (refer to revisions)

Contents

About this guide

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

Many Amazon Web Services (AWS) customers use images on their websites and mobile applications to drive user engagement. Websites with large image files can experience high load times, so developers often provide multiple versions of each image to accommodate different bandwidth and layout constraints. This process can be difficult to manage and can slow down development because it requires version control, increased storage and compute costs for file reprocessing, and coordination with application and development teams to update image files.

The Serverless Image Handler solution helps customers provide a low-latency website response, and decrease the cost of image optimization, manipulation, and processing. The solution combines highly available, trusted AWS services and the open source image processing suite Sharp to enable fast and cost-effective image manipulation in the AWS Cloud. 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 for one million newly processed images, 15 GB storage, and 50 GB data transfer, with default settings in the US East (N. Virginia) Region is shown in the following table. This includes estimated charges for Amazon API Gateway, AWS Lambda, Amazon CloudFront, and Amazon S3 storage.

AWS service	Total cost per 1 million new images/month
Amazon API Gateway	$3.50
AWS Lambda	$8.53
Amazon CloudFront	$6.25
Amazon S3	$0.345

If you choose to deploy the demo user interface, 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, which can vary because modified images are cached in CloudFront, and because certain scenarios require special-use capabilities such as smart cropping with Amazon Rekognition. 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.

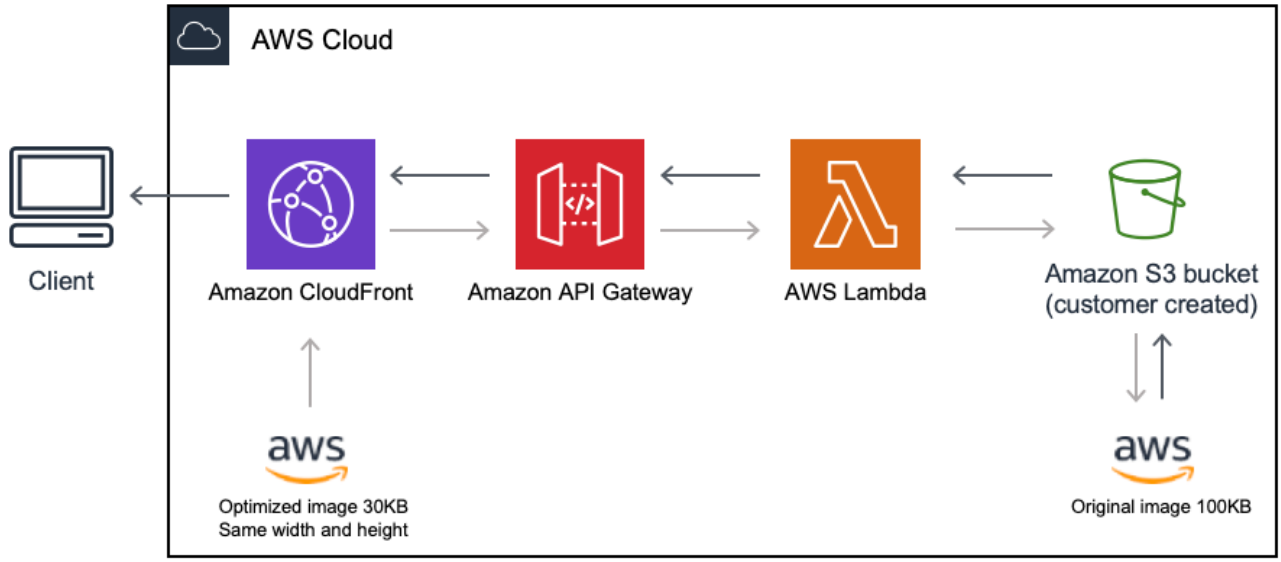

Figure 1: Serverless Image Handler architecture on AWS

> **Note:** This solution is intended for customers with public applications who want to provide an option for dynamically changing or manipulating 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 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 triggers 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.

> **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 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 enabled:

- **Demo UI:** An optional demo user interface (UI) that 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.

Implementation considerations

Cross-Origin Resource Sharing (CORS)

This solution's template contains two parameters: **CorsEnabled** and **CorsOrigin** that allow you to enable 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 enable 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 enable or disable 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*.

Solution updates

Serverless Image Handler v4.2 and later uses the most up-to-date Node.js runtime. Version 4.0 uses the Node.js 8.10 runtime, which reached end-of-life on December 31, 2019. As of January 2020, AWS Lambda blocks the create operation, and, as of February 2020, AWS Lambda blocks 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:

View template

serverless-image-handler.template: Use this template to launch the Serverless Image Handler and all associated components. The default configuration deploys Amazon CloudFront, Amazon API Gateway, and AWS Lambda.

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 bucket to store the images you want to manipulate. Note that if you have multiple image source 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.

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 Appendix B.

Deployment overview

The procedure for deploying this architecture on AWS consists of the following steps. For detailed instructions, follow the links for each step.

Step 1. Launch the stack

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

Step 2. Create and use image requests

- 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 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 use the button to the right to launch the `serverless-image-handler` AWS CloudFormation template.
   You can also download the template as a starting point for your own implementation.

   Launch Solution

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 enable Cross-Origin Resource Sharing (CORS).
CorsOrigin	`*`	This value is returned by the API in the Access-Control-Allow-Origin header. A star (*) 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>*	The S3 bucket (or buckets) in your account that contains the images that you manipulate. If providing multiple buckets, separate them by commas.
DeployDemoUI	`Yes`	The demo UI that deploys to the `Demo` S3 bucket. For more information refer to Appendix B.
LogRetentionPeriod	`1`	Number of days to retain Lambda log data in CloudWatch logs.
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 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>
```

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 the AWS Security Center.

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 different 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 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.

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

Appendix A: Backward compatibility

The updated 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 note 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 (sourcing images from a subfolder within an Amazon S3 bucket results in an error) 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>
```

Appendix B: Using the demo UI

The solution provides an optional demo user interface that can be deployed into your AWS account to show 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.

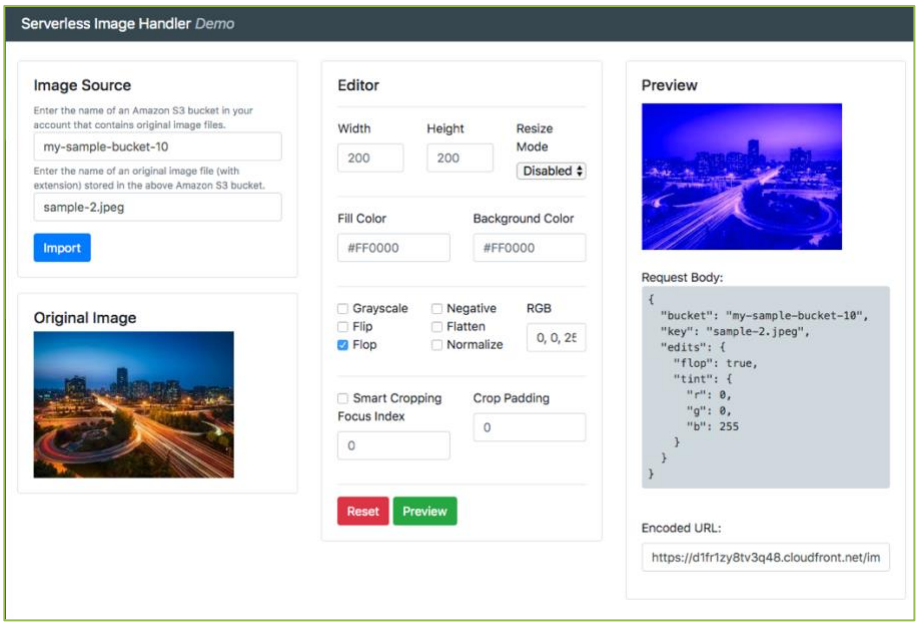

Figure 2: Serverless Image Handler demo UI

Use the following procedure to experiment with all of 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. We recommended using your own front-end application for image modification.

Appendix C: 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 enable smart cropping on an image, add the **smartCrop** property to the **edits** property in the image request.

- **smartCrop**: (optional, Boolean || Object) enables 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
    }
})
```

- **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, as shown in the example below.

```
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
        }
    }
})
```

Appendix D: 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)/

Filter Name	Filter Syntax
Sharpen	/filters:sharpen(0.0-10.0, 0.0-2.0, true/false)/
Stretch	/filters:stretch()/
Strip Exif	/filters:strip_exif()/
Strip ICC	/filters:strip_icc()/
Upscale	/filters:upscale()/
Watermark	/filters:watermark(bucket,key,x,y,alpha[,w_ratio[,h_ratio]])

Appendix E: 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:

Parameter	Value Type	Description
CorsEnabled	`Yes/No`	Indicates whether to return an Access-Control-Allow-Origin header with the image handler API response.
CorsOrigin	`String`	This value is returned by the API in the Access-Control-Allow-Origin header. A star (*) 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 **CorsEnabled** is set to `No`.

Parameter	Value Type	Description
RewriteMatchPattern	`Regex`	By default, this parameter is empty. Contains a JavaScript-compatible regular expression for matching custom image requests using the rewrite function.
RewriteReplacePattern	`String`	By default, this parameter is empty. Contains a substitution string for custom image requests using the rewrite function.
SourceBuckets	`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.
AutoWebP	`Yes/No`	Choose whether to automatically accept webp image formats.

Appendix F: Rewrite feature

This feature allows customers to migrate their current image request model to the Serverless Image Handler solution, without changing their 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	Value Type	Description
RewriteMatchPattern	`Regex`	By default, this parameter is empty. Contains a JavaScript-compatible regular expression for matching custom image requests using the rewrite function.
RewriteReplacePattern	`String`	By default, this parameter is empty. Contains a substitution string for custom image requests using the rewrite function.

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.

Appendix G: Collection of operational metrics

This solution includes an option to send anonymous operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When enabled, 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

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.

Modify the AWS CloudFormation template mapping section as follows:

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

to

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

Source code

You can visit our GitHub repository to download the templates and scripts for this solution, and to share your customizations with others.

Document 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

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 current AWS 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. The responsibilities and liabilities of AWS 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.