Serverless Image Handler

AWS Implementation Guide

Ian Hartz

Garvit Singh

June 2017

Last update: January 2019 (see 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 on 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 in order to ensure a great user experience across different devices, developers often provide multiple versions of each image to accommodate different bandwidth and layout constraints. This process can be difficult to manage and cause time delays, as it often requires version control, increased storage and compute costs for file reprocessing, and coordination with application teams and web developers to update image files.

To help customers provide a low-latency website response, and decrease the cost of image optimization, manipulation, and processing, AWS offers the Serverless Image Handler, a solution that combines highly available, trusted AWS services and the open source image processing suite Thumbor to enable fast and cost-effective image manipulation on the AWS Cloud. This reference implementation automatically deploys and configures a serverless architecture that is optimized for dynamic image manipulation, and that features 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 1 million images processed, 15 GB storage, and 50 GB data transfer, with default settings in the US East (N. Virginia) Region is as shown in the table below. This includes estimated charges for Amazon API Gateway, AWS Lambda, Amazon CloudFront, and Amazon S3 storage.

AWS Service	Total Cost per 1 Million Images
Amazon API Gateway	$3.50
AWS Lambda	$3.10
Amazon CloudFront	$6.00
Amazon S3	$0.23

This cost estimate does not account for Amazon S3 PUT and GET requests, which can vary per scenario because modified images are cached in CloudFront. There is no additional cost for using Thumbor, which is an open source tool. Prices are subject to change. For full details, see the pricing webpage for each AWS service you will be using in this solution.

Architecture Overview

Automatic deployment of this solution configures the following components and functionality.

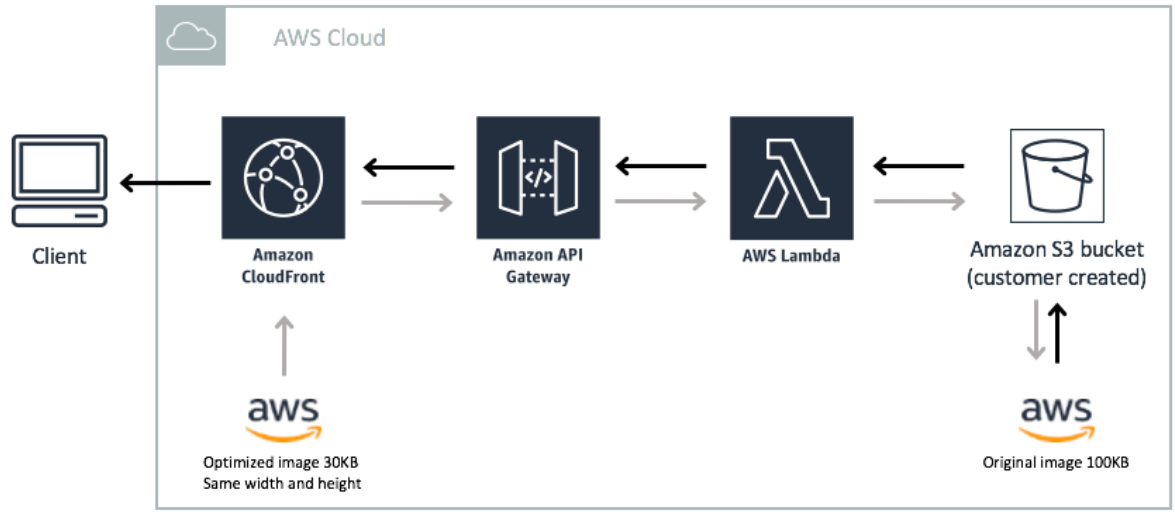


Figure 1: Serverless Image Handler architecture on AWS

> **Note:** This solution is intended for customers with public applications who want to provide an option for dynamically changing or manipulating their public images. As a result, 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, see 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 reduce 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 Simple Storage Service (Amazon S3) bucket and uses Thumbor to return a modified version of the image to the API Gateway.

The solution generates a CloudFront domain name that gives access to objects in a customer-specified bucket, as well as a partial URL. In your custom front-end application, you can append the CloudFront URL with the image path and name to create a direct URL for each image object. Your users can then manipulate the image dynamically by adding Thumbor filters to the URL. Additionally, the solution deploys an optional demo user interface where you can more easily generate filters and tasks, and create example URLs that you can use for your images. The demo UI is deployed in an Amazon S3 bucket to allow customers to immediately start manipulating images with a simple web interface. Amazon CloudFront is used to restrict access to the solution's website bucket contents. For a list of filters that the solution supports, see Appendix B.

For customers who don't want to add Thumbor filters into their URLs or who already have a naming convention in place, this solution includes a rewrite feature. This feature allows you to match existing URL patterns and apply Thumbor parameters automatically within your API. For more information, see Appendix C.

Implementation Considerations

Cross-Origin Resource Sharing (CORS)

The solutions template contains two parameters: **Enable CORS** and **CORS Origin** that allow you to enable CORS for your API resources. CORS defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. With CORS support, you may make API requests from a domain other than the API's domain. If you would like to change your CORS configuration after deployment, you can enable or disable CORS with the `ENABLE_CORS(YES/NO)` and AWS Lambda environment variable, or change the origin domain with the `CORS_ORIGIN` variable.

AWS CloudFormation Template

This solution uses AWS CloudFormation to automate the deployment of the Serverless Image Handler solution on 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, but you can also customize the template based on your specific network needs.

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 25 minutes

Prerequisites

Before you launch the solution's AWS CloudFormation template, you must specify an existing Amazon Simple Storage Service (Amazon S3) bucket. Use this bucket to store the images you want to manipulate. 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, see Appendix A.

What We'll Cover

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: **Origin S3 Bucket**, **Origin S3 Bucket Region**, **UI Prefix**
- Review the other template parameters, and adjust if necessary.

Step 2. Create an image URL

- Generate URLs for your images.

Step 1. Launch the Stack

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

> **Note**: You are responsible for the cost of the AWS services used while running this solution. See the Cost section for more details. For full details, see the pricing webpage for each AWS service you will be using in this solution.

1. Log in to the AWS Management Console and click 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.

   > **Note**: This solution uses the AWS Lambda and Amazon API Gateway services, which are currently available in specific AWS Regions only. Therefore, you must launch this solution in an AWS Region where Lambda and API Gateway are available. [1]

3. On the **Select Template** page, keep the default settings for **Stack** and **Template Source**.

4. Under **Parameters**, review the parameters for the template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
Origin S3 Bucket	*<Requires-Input>*	The S3 bucket that contains the images that you will manipulate
Origin S3 Bucket Region	`us-east-1`	The region that contains the **Origin S3 Bucket**
Enable Cors	`No`	Will this API require Cross-Origin Resource Sharing (CORS) support?
Cors Origin	`*`	This value will be returned by the API in the Access-Control-Allow-Origin header. A star (*) value will support any origin. We recommend specifying a specific origin (e.g. http://example.domain) to restrict cross-site access to your API.
Lambda Log Retention	`1`	Number of days to retain Lambda log data in CloudWatch logs

[1] For the most current service availability by region, see https://aws.amazon.com/about-aws/global-infrastructure/regional-product-services/

Parameter	Default	Description
Deploy UI	`Yes`	The demo UI that will be deployed to the Demo S3 bucket. For more information see Appendix A.
UI Prefix	`serverless-image-handler-ui/`	The prefix for the files in the Origin S3 bucket. Note that only one folder can be used.

5. Choose **Next.**
6. On the **Options** page, choose **Next**.
7. On the **Review** page, review and confirm the settings. Be sure to check the box acknowledging that the template will create AWS Identity and Access Management (IAM) resources.
8. Choose **Create** to deploy the stack.

   You can view the status of the stack in the AWS CloudFormation Console in the **Status** column. You should see a status of CREATE_COMPLETE in approximately 25 minutes.

Step 2. Create an Image URL

The solution generates a CloudFront domain name that gives you access to the images in your S3 bucket, as well as a partial URL, you can use to manipulate images. Note that you must set the **Deploy UI** template parameter to `No` to generate the partial URL (SampleRequest URL).

1. In the AWS CloudFormation stack **Outputs** tab, copy the **Sample Request** URL.
2. Past the URL into your browser and append it with a Thumbor filter, the path (if applicable), and filename for an image in your S3 bucket.

> **Note:** If the image name and extension in the URL doesn't match the image name in the S3 bucket, you will receive an error.

In your custom front-end application, you can append the CloudFront URL with the bucket folder path and image name to create a URL for each image object (example: http://cloudfront.../path/image01.jpg).

> **Note:** If you choose to deploy the optional demo user interface, you can experiment with all of the supported image manipulation features, preview the results, and create example URLs that you can use for your other images. For more information, see Appendix A.

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 custom 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, see 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 S3

Appendix A: Using the Demo UI

The solution provides an optional demo user interface you can use to more easily generate filters and tasks. This feature allows customers to interact directly with their new API endpoint using six example images.

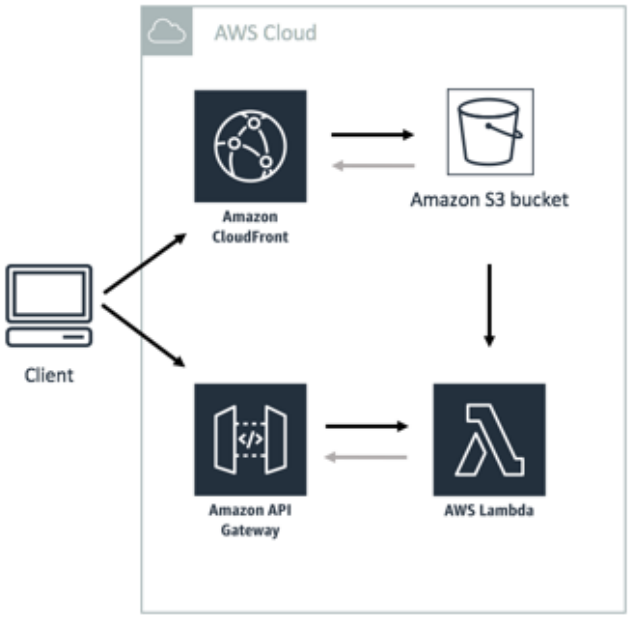


Figure 2: Serverless Image Handler Demo UI

Use the following procedure to experiment with all of the supported image manipulation features, preview the results, and create example URLs that you can use for all your other images:

1. In the AWS CloudFormation stack **Outputs** tab, select the **DemoUI URL**.
2. Adjust the available filters to understand the solutions behavior.

> **Note**: To use your existing images after testing the functionality with the demo UI, you must update the AWS CloudFormation Stack and change the **Deploy UI** template parameter to `No`. This automatically updates the solution to work with the images in the Amazon S3 bucket you specified in the **OriginS3Bucket** template parameter.

Appendix B: List of Supported Filters

This solution currently supports the filters listed in the table below. You must append your CloudFront URL with the syntax below and image name to use the filters. 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):rotate(90)/<somepath>/<example>.jpg`

Filters process the image in the order they are specified. For detailed instructions and more options, see the Thumbor documentation.

Filter Name	Filter Syntax
Background color	/filters:background_color(color)/
Blur	/filters:blur(7)/
Brightness	/filters:brightness(40)/
Color fill	/filters:fill(color)/
Contrast	/filters:contrast(40)/
Convolution	/filters:convolution(1;2;1;2;4;2;1;2;1,3,false)/
Equalize	/filters:equalize()/
Format	/filters:format(jpeg)/
Grayscale	/filters:grayscale()/
Image type (jpeg, png,webp,gif)	/filters:format(jpeg)/
Max bytes	/filters:max_bytes(40000)/
Noise	/filters:noise(40)/
Quality	/filters:quality(40)/
Resize	/fit-in/800x1000/
RGB	/filters:rgb(20,-20,40)/
Rotate	/filters:rotate(90)/

Round Corner	/filters:round_corner(20,255,255,255)/
Strip ICC	/filters:strip_icc(10)
Watermark	/filters:watermark(https://<imageurl,x,y,z>)
SmartCrop	/smart

Appendix C: 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. To use this feature, modify the solution's rewrite feature settings.

> **Note:** If you have already deployed the Serverless Image Handler solution and want to use the rewrite feature, you must redeploy the AWS CloudFormation stack.

In the AWS Lambda console, select the solution's primary Lambda function and set the **REWRITE_ENABLED** environment variable to `YES`. Then, in the **REWRITE_PATTERNS** environment variable, add your rewrite rules. The list of rules must be comma separated and wrapped in square brackets. Each rule must be wrapped in parentheses and contain regular expressions composed of two strings: one for the match condition and one for substitution. Note that the regex must be compatible with Python. For example:

```
[(r'^/string_to_match$', r'rewritten_string')]
```

The solution checks to see if the rewrite feature is enabled before it sends the request to Thumbor. If it is enabled, the rewrite feature parses any customer defined rules in order until it finds a match, and uses the provided patterns to transform the image while preserving the originally requested URL. If no match is found, the original unmodified image request is sent to Thumbor.

Below are `REWRITE_PATTERNS` examples:

- This example matches a string ('-zoom') in a requested image name in the /s/ folder, and returns a 630x630 resized image of the original file located in the /s/ folder (notice that the original file name is appended with the string '-zoom'). This example demonstrates the ability to create identifiers in the requested URL that map to different image filters.

Original Path	`/s/0002fe146de274f90ae005346834e40c-zoom.jpg`
Match pattern	`r'^/([s])/([0-9a-f]+)-(zoom)\.([a-zA-Z0-9]+)$'`
Replace pattern	`r'/fit-in/630x630/s/\2.\4'`
Rewritten request	`/fit-in/630x630/s/0002fe146de274f90ae005346834e40c.jpg`

This example would be implemented by modifying the following Lambda function environment variables:

REWRITE_ENABLED	`Yes`
REWRITE_PATTERNS	`[(r'^/([s])/([0-9a-f]+)-(zoom)\.([a-zA-Z0-9]+)$', r'/fit-in/630x630/s/\2.\4')]`

- This example builds on the previous example; however, it also matches requests from the /p/ folder and rewrites the request to retrieve the original image from the S3 origin's /product/ folder.

Original Path	`/p/0002fe146de274f90ae005346834e40c-zoom.jpg`
Match pattern	`r'^/([p])/([0-9a-f]+)-(zoom)\.([a-zA-Z0-9]+)$'`
Replace pattern	`r'/fit-in/630x630/product/\2.\4'`
Rewritten request	`/fit-in/630x630/product/0002fe146de274f90ae005346834e40c.jpg`

This example would be implemented by modifying the following Lambda function environment variables:

REWRITE_ENABLED	`Yes`
REWRITE_PATTERNS	`[(r'^/([p])/([0-9a-f]+)-(zoom)\.([a-zA-Z0-9]+)$', r'/fit-in/630x630/product/\2.\4')]`

- This example shows how you can create a comma-separated list of patterns to specify multiple rewrite rules. Notice how the `REWRITE_PATTERNS` environment variable value begins and ends with brackets, and each rule is wrapped in parentheses. Additionally, we recommended placing your most frequently used patterns at the beginning of the list to increase rule processing performance.

```
[(r'^/([s])/([0-9a-f]+)-(zoom)\.([a-zA-Z0-9]+)$', r'/fit-
in/630x630/sample/\2.\4'),(r'^/([s])/([0-9a-f]+)-(product)\.([a-zA-
Z0-9]+)$', r'/fit-in/220x220/sample/\2.\4'),(r'^/([s])/([0-9a-f]+)-
(cart)\.([a-zA-Z0-9]+)$', r'/fit-
in/53x53/sample/\2.\4'),(r'^/([s])/([0-9a-f]+)-(catalog)\.([a-zA-Z0-
9]+)$', r'/fit-in/120x120/sample/\2.\4'),(r'^/([s])/([0-9a-f]+)-
(gallery)\.([a-zA-Z0-9]+)$', r'/fit-
in/33x33/sample/\2.\4'),(r'^/([s])/([0-9a-f]+)-(list)\.([a-zA-Z0-
9]+)$', r'/fit-in/92x92/sample/\2.\4')]
```

Appendix D: Watermarking Feature

The solution allows customers to watermark their images. The demo UI comes with a sample image that includes an AWS logo to show you the watermarking feature. You can apply similar watermarks to your images by leveraging filters:watermark(<image-url>,x,y,z) where x,y,z represents x, y coordinates and transparency of the watermark. For more information about watermarking, see Thumbor Watermarking.

Appendix E: Safe URL

This solution allows customers to deploy secured URLs using a custom security key. The security key needs to be provided to the Thumbor configuration using AWS Lambda environment variables. For more information about using Lambda variables, see Appendix J.

When end-users access the page and load the image, Thumbor generates an authentication code for the image URLs and filters, using the **SECURITY_KEY** provided in the Thumbor config file. If the hash in the request URL and the authentication codes match, Thumbor processes the image. For more information, see Thumbor's safe URL.

Use the following procedure to implement and verify Safe-URL implementation:

1. Log in to the AWS Lambda console, select the ***<stack-name>*-ImageHandlerFunction-*<id>*** Lambda function.
2. Remove the Lambda environment variable: **Key**=**ALLOW_UNSAFE_URL**

> **Note:** You can only use one environment variable at a time. Use **ALLOW_UNSAFE_URL=true** for allowing unsafe URLs, or use **SECURITY_KEY='mysecuritykey'** when safe URLs are needed.

3. Add the Lambda environment variable: **Key**=**SECURITY_KEY**, **Value**=**mysecuritykey**

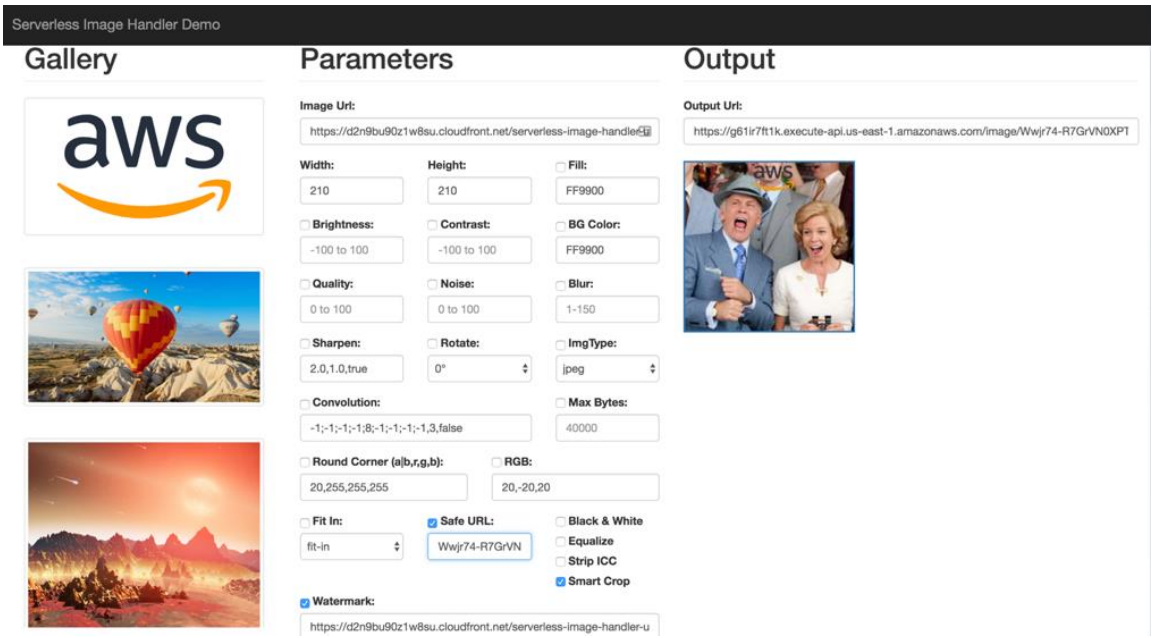
4. Select **Save Changes**.
5. In the **Outputs** section, select the **Solution UI URL**.
6. Change the height/width to make sure you are not getting cached version of image.
7. Select **Safe URL** and set the value to calculated hash for **mysecuritykey**.

```
```
http_key='mysecuritykey' # security key provided to lambda env
variable
http_path='200x200/smart/sub-folder/myimage.jpg' # sample options
for myimage
hashed = hmac.new(str(http_key),str(http_path),sha1)
encoded = base64.b64encode(hashed.digest())
signed_path = encoded.replace('/','_').replace('+','-')
```
```

The above procedure implements the following image URL and filters with options:

210x210/smart/filters:watermark(https://d2n9bu90z1w8su.cloudfront.net/serverless-image-handler-ui/img/aws-logo-watermark.png,75,0,0)/serverless-image-handler-ui/img/multiface.jpg

hash: Wwjr74-R7GrVN0XPT-Aq_DGFOh8=



Following this procedure successfully implements a secured URL. You may rotate security keys as per your requirements and as often as needed by updating the Lambda environment variable values. To verify the implementation, change the **Security Key** value in the Demo UI console, navigate to the Lambda logs and check for the following error:

Appendix F: URL Encoding for Watermarking

This solution supports URL encoding as per the industry standard RFC 3986 when watermarking images. Use the following code logic to encode the URL:

```
def encoding_string(string):
    """
    Encoding URL per RFC 3986.
    """
    return(quote(string,safe=''))
```

Example:
Watermarking image URL:
http://secure.gravatar.com/avatar/a432db53989289d4f251c52c72ff2bd3?d=identicon&r=g&s=14000

As per RFC 3986, the value to be provided to watermark filter should be:

http%3A%2F%2Fsecure.gravatar.com%2Favatar%2Fa432db53989289d4f251c52c72ff2bd3%3Fd%3Didenticon%26r%3Dg%26s%3D14000

Appendix G: Viewing the Metadata

This solution leverages thumbor_rekognition to incorporate Amazon Rekognition capabilities. To understand and identify Thumbor operations and validate the solution before using it in a production environment, you must prepend all of the solution parameters **/debug**. (for example: http://<cloudfront-id>.net/debug/300x300/smart/<s3-prefix-path>/picture.png)

To view the metadata on all Thumbor operations, add **/meta/** to all of the URLs (for example: http://<cloudfront-id>.net/meta/300x300/left/top/smart/<s3-prefix-path>/picture.png). Once the processing is finished, Thumbor will return a .json object with the metadata on the image and the operations that would have been performed. For more information, see Metadata Endpoint.

```
{
    thumbor: {
        source: {
            url: "path/to/my/nice/image.jpg",
            width: 800,
```

```
            height: 600
        },
        operations: [
            {
                type: "crop",
                left: 10,
                top: 10,
                right: 300,
                bottom: 200
            },
            {
                type: "resize",
                width: 300,
                height: 200
            },
            { type: "flip_horizontally" },
            { type: "flip_vertically" }
        ]
    }
}
```

Appendix H: OpenCV vs Amazon Rekognition

The solution comes with OpenCV and Amazon Rekognition as the supported face detectors. By default, the solution uses Amazon Rekognition and recommends customers use it as the preferred face detector. For more information, see Thumbor detectors.

To choose your preferred detector, make the following changes in `thumbor.conf`:

To use OpenCV

```
DETECTORS = [
#'thumbor.detectors.face_detector'
 'thumbor_rekognition'
 #'thumbor.detectors.profile_detector'
   #'thumbor.detectors.glasses_detector',
  'thumbor.detectors.feature_detector',
]
```

To use Amazon Rekognition

```
DETECTORS = [
 'thumbor.detectors.face_detector',
#'thumbor_rekognition',
 #'thumbor.detectors.profile_detector',
```

```
    #'thumbor.detectors.glasses_detector',
        'thumbor.detectors.feature_detector',
    ]
```

Un-comment the needed detector and upload the `thumbor.conf` file. For more information, see Appendix I.

Appendix I: Using Lambda Environment Variables

You can use AWS Lambda environment variables to set a variety of Thumbor config parameters and solution configurations.

Certain common values in the Thumbor configuration can be controlled using Lambda environment variables such as:

- **RESPECT_ORIENTATION**
- **ALLOW_UNSAFE_URL**
- **SECURITY_KEY**

The following solution configurations are controlled by environment variables:

- **SEND_ANONYMOUS_DATA**
- **LOG_LEVEL**
- **ENABLE_CORS**

Use the following procedure to adjust these Lambda environment variables :

1. Navigate to the AWS Lambda console.
2. Select ***<stack-name>*-ImageHandlerFunction-*<id>***, and navigate to the **configuration** tab.
3. Navigate to the **Environment variables** section.
4. Add the new environment variable by providing the **key** and **value**.

Appendix J: Customizing Thumbor Lambda package

Use the following procedure to make customizations to the Thumbor AWS Lambda package:

Note that you must create two publicly accessible Amazon S3 buckets: *my-bucket-name*, and *my-bucket-name-<aws-region>*.

1. Configure the bucket name to your Amazon S3 distribution bucket.

```
export TEMPLATE_OUTPUT_BUCKET=my-bucket-name # bucket where cfn
template will reside
export DIST_OUTPUT_BUCKET=my-bucket-name # bucket where customized
code will reside
export VERSION=my-version # version number for the customized code
```

2. Setup your OS/Python Environment:

```
$ yum install yum-utils epel-release -y
$ sudo yum-config-manager --enable epel
$ sudo yum update -y
$ sudo yum install zip wget git libpng-devel libcurl-devel gcc
python-devel libjpeg-devel -y
$ alias sudo='sudo env PATH=$PATH'
$ sudo pip install setuptools==39.0.1
$ sudo pip install virtualenv==15.2.0
```

3. Clone the github repo:

```
$ git clone https://github.com/awslabs/serverless-image-
handler.git
```

4. Navigate to the deployment folder:

```
$ cd serverless-image-handler/deployment
```

5. Run ./build-s3-dist.sh *<bucket-name> <version>*:

```
$ ./build-s3-dist.sh mybucket v1.0
```

6. Deploy the distribution to an Amazon S3 bucket in your account. Note that you must have AWS Command Line Interface installed.

```
bash
aws s3 cp ./dist/ s3://$DIST_OUTPUT_BUCKET-[region_name]/serverless-image-handler/$VERSION/ --recursive --exclude "*" --include "*.zip"
aws s3 cp ./dist/serverless-image-handler.template s3://$TEMPLATE_OUTPUT_BUCKET/serverless-image-handler/$VERSION/
```

Note that the solution will verify the source code should is located in the *my-bucket-name-[region-name]* bucket with the prefix: `serverless-image-handler/my-version/serverless-image-handler.zip`

7. Copy the **serverless-image-handler.template link** uploaded to your Amazon S3 bucket.
8. Using the copied link above, deploy the Serverless Image Handler into your account.

```
```bash
https://s3.amazonaws.com/my-bucket-name/serverless-image-handler/my-version/serverless-image-handler.template
```
```

Appendix K: Troubleshooting

To troubleshoot known issues with the Serverless Image Handler solution, AWS recommends customers set the AWS Lambda environment variable **`LOG_LEVEL`** to `DEBUG`. This will enable you to see detailed error logs in Amazon CloudWatch. For more information about modifying environment variables, see Appendix I.

Common Errors

Malformed URL

This error can result if the security key provided in user request and the one provided to Thumbor config do not match.

Resolution

Verify that both security keys match. For more information, see Appendix E.

Body Size is Too Long

If the image is larger than 6 MB, in your Amazon CloudWatch logs you will receive the following error: **body size is too long**. For more information see, AWS Lambda limits.

```
[INFO] 2018-07-08T13:30:39.22Z 16121296-82b3-11e8-872e-a1cab863f08b 200 GET /unsafe/fit-in/800x0/nbv4mbq3/31508336956304.gif (0.0.0.0)
7394.32ms

body size is too long

END RequestId: 16121296-82b3-11e8-872e-a1cab863f08b
```

Resolution

Customize your Lambda package to save the resulting image in the Amazon S3 bucket. Currently, the solution requires you to manually update your result storage settings. For more information, see tc_aws result storage.

Data Type is Not Supported

If the data type isn't supported in the Lambda environment variable for Thumbor config, you will receive this error. For more information, see non-string environment variables.

```
[ERROR] 2018-04-24T13:45:14.10Z b5e9ae1d-47c5-11e8-b9b1-e37552703238 [BaseHander.finish_request] Invalid quality setting
Traceback (most recent call last):
File "/var/task/thumbor/handlers/__init__.py", line 406, in inner
future_result = future.result()
File "/var/task/concurrent/futures/_base.py", line 455, in result
return self.__get_result()
File "/var/task/concurrent/futures/_base.py", line 414, in __get_result
raise exception_type, self._exception, self._traceback
ValueError: Invalid quality setting
```

Resolution

Configure your `thumbor.conf` file with the specific Thumbor parameter and customize your Lambda package. See allow_environment_variables() for more information.

Appendix L: Collection of Anonymous Data

This solution includes an option to send anonymous usage data 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 Thumbor version
- **Region:** The AWS Region the solution is being deployed in

Note that AWS will own the data gathered via this survey. Data collection will be subject to the AWS Privacy Policy. To opt out of this feature, complete one of the following tasks:

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

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

to

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

OR

b) After the solution has been launched, find the `serverless-image-handler` function in the Lambda console and set the **SEND_ANONYMOUS_DATA** environment variable to `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	In sections
June 2017	Initial Release	--
August 2017	Solution updated to add the rewrite feature and the optional deployment of a demo UI	Cost; Architecture Overview; Appendix A; Appendix B
October 2017	Solution updated to provide CORS support	Implementation Considerations; Parameters
September 2018	Added information on watermarking, URL encoding, debugging, and troubleshooting	Appendix C; Appendix D; Appendix E; Appendix F; Appendix G; Appendix H; Appendix I; Appendix J
December 2018	Added information about the Amazon CloudFront distribution for the static website hosted in the Amazon S3 bucket	Security
January 2019	Added information about using the demo UI, safe URLs, and customizing the Thumbor Lambda package	Appendix A; Appendix E; Appendix J

Notices

This document is provided for informational purposes only. It represents AWS's current product offerings and practices as of the date of issue of this document, which are subject to change without notice. Customers are responsible for making their own independent assessment of the information in this document and any use of AWS's products or services, each of which is provided "as is" without warranty of any kind, whether express or implied. This document does not create any warranties, representations, contractual commitments, conditions or assurances from AWS, its affiliates, suppliers or licensors. 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.

The Serverless Image Handler solution is licensed under the terms of the Amazon Software License available at https://aws.amazon.com/asl/.