Implementation Guide

Dynamic Image Transformation for Amazon CloudFront (Formerly known as Serverless Image Handler)

Dynamic Image Transformation for Amazon CloudFront (Formerly known as Serverless Image Handler): Implementation Guide

Table of Contents

Serverless architecture for cost-effective image processing

Publication date: *June 2017*. Check the CHANGELOG.md file in the GitHub repository to see all notable changes and updates to the software. The changelog provides a clear record of improvements and fixes for each version.

The Dynamic Image Transformation for Amazon CloudFront solution helps you embed images on your websites and mobile applications to drive user engagement. It uses the sharp Node.js library to provide high-speed image processing without sacrificing image quality. To minimize your costs of image optimization, manipulation, and processing, this solution automates version control and provides flexible storage and compute options for file reprocessing.

This solution automatically deploys and configures a serverless architecture optimized for dynamic image manipulation. Images can be rendered and returned spontaneously. For example, you can automate resizing of an image based on different screen sizes by adding code on your website that leverages this solution. This helps you adapt your website's presentation to meet your users' different modes of viewing. This solution uses Amazon CloudFront for global content delivery and Amazon Simple Storage Service (Amazon S3) for reliable and durable cloud storage.

This implementation guide provides an overview of the Dynamic Image Transformation for Amazon CloudFront solution, its reference architecture and components, considerations for planning the deployment, configuration steps for deploying the solution to the Amazon Web Services (AWS) Cloud.

The intended audience for implementing this solution in their environment includes solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals.

Use this navigation table to quickly find answers to these questions:

If you want to . . .	Read . . .
Know the cost for running this solution. The estimated cost for running this solution in the US East (N. Virginia) Region is **approxima tely USD $5.30 per month** for 100,000 new images.	Cost

If you want to . . .	Read . . .
Understand the security considerations for this solution.	Security
Know how to plan for quotas for this solution.	Quotas
Know which AWS Regions support this solution.	Supported AWS Regions
View or download the AWS CloudForm ation template included in this solution to automatically deploy the infrastructure resources (the "stack") for this solution.	AWS CloudFormation template
Access the source code and optionally use the AWS Cloud Development Kit (AWS CDK) to deploy the solution.	GitHub repository

Features and benefits

This solution provides the following features:

Dynamic content delivery

Automatically modify images based on users' devices and screen sizes.

Content moderation

Use Amazon Rekognition to automatically detect and blur inappropriate user-uploaded images.

Smart cropping

Use Amazon Rekognition to crop images using facial recognition.

Low-cost image storage

Save on image storage costs by generating modified images at runtime, and caching generated images in CloudFront.

Integration with Service Catalog AppRegistry and Application Manager, a capability of AWS Systems Manager

This solution includes a Service Catalog AppRegistry resource to register the solution's CloudFormation template and its underlying resources as an application in both Service Catalog AppRegistry and Application Manager. With this integration, you can centrally manage the solution's resources and enable application search, reporting, and management actions.

Use cases

Drive user engagement

Improve engagement with your mobile application or website by maintaining high-quality images that adjust for device screen size.

Improve user and brand safety

Automatically detect and blur inappropriate user-uploaded images with machine learning trained to recognize pre-defined and user-defined categories .

Concepts and definitions

This section describes key concepts and defines terminology specific to this solution:

cross-origin resource sharing (CORS)

Defines a way for client web applications that are loaded in one domain to interact with resources in a different domain.

fallback image

Image that you set to show when the intended image doesn't load.

Note

For a general reference of AWS terms, see the AWS Glossary.

Architecture overview

This section provides a reference implementation architecture diagram for the components deployed with this solution. Dynamic Image Transformation for Amazon CloudFront maintains two options for architecture. The Default Deployment uses API Gateway as a CloudFront origin and is limited to 6 MB responses. The S3 Object Lambda deployment uses a S3 Object Lambda Access Point as the CloudFront origin and can support as large an image as can be processed before the 30s response timeout. For more information about differences between the options, refer to choosing an architecture.

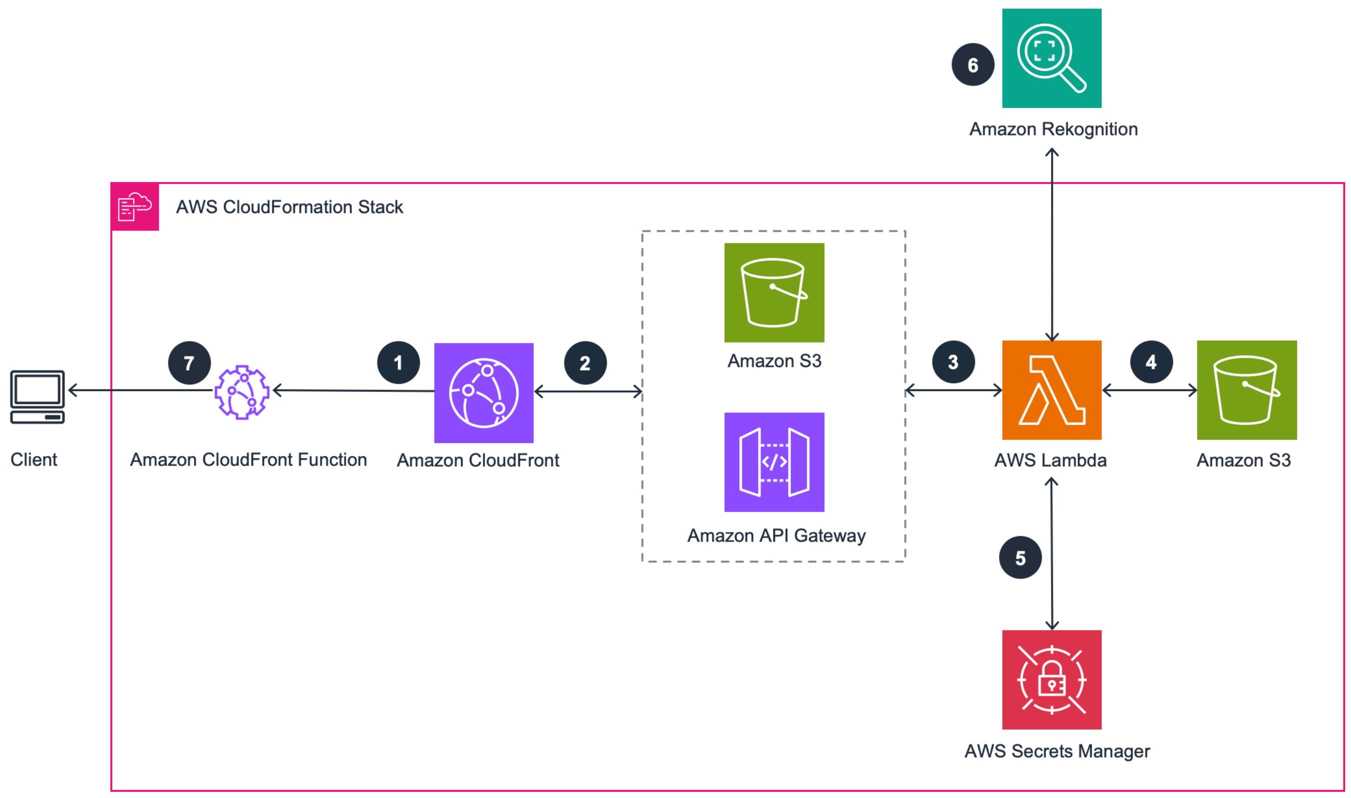
Architecture diagram

Deploying this solution with the default parameters deploys the following components in your AWS account.

Important

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 CloudFront distribution and Amazon API Gateway endpoint in your account, allowing anyone to access it. For more information on API Gateway authorization, refer to the Security section. This solution supports signing requests, which can serve to restrict unauthorized requests, for more information, refer to the Image URL Signature section.

CloudFormation template deploys AWS resources for serverless image processing.



Note

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

The high-level process flow for the solution components deployed with the AWS CloudFormation template is as follows:

1. An Amazon CloudFront distribution provides a caching layer to reduce the cost of image processing and the latency of subsequent image delivery. The CloudFront domain name provides cached access to the image handler application programming interface (API).
2. Amazon API Gateway / Amazon S3 Object Lambda provides endpoint resources and initiate the AWS Lambda function /
3. A Lambda function retrieves the image from a customer's existing Amazon S3 bucket and uses `sharp` to return a modified version of the image to the API Gateway/S3 Object Lambda Access Point.

4. A solution-created S3 bucket provides log storage, separate from your customer-created S3 bucket for storing images. If you enter `Yes` (default entry) for **the Deploy Demo UI** template parameter, the solution deploys another S3 bucket for storing the optional demo user interface (UI).
5. (Optional) If you enter `Yes` for the **Enable Signature** template parameter, the Lambda function retrieves the secret value from your existing AWS Secrets Manager secret to validate the signature. For more information, see Launch the stack.
6. (Optional) If you use the smart crop or content moderation features, the Lambda function calls Amazon Rekognition to analyze your image and returns the results.
7. The viewer request is proxied through an Amazon CloudFront function. This function is responsible for normalizing the accept header and query params to increase the cache hit rate. As well, if S3 Object Lambda is enabled in the CloudFormation template parameters, the viewer response will be proxied through a CloudFront function to allow for the rebuilding of certain response elements that are not natively supported by S3 Object Lambda.

S3 Object Lambda Architecture Info

Note

The S3 Object Lambda architecture allows for returning images which are larger than 6 MB. This infrastructure replaces the API Gateway component in the default architecture. This architecture will be used if the Enable S3 Object Lambda template parameter is set to Yes.

Important

This optional architecture makes several changes to the resources in the CloudFormation stack. Though care has been taken to ensure that responses are as similar as possible to those returned by the API Gateway Architecture, certain response headers may be slightly different. If you are updating an existing stack, please validate your application's functionality post update. As API Gateway is no longer used as part of this optional architecture, any existing REST API will be deleted upon updating. When updating a stack to the S3 Object Lambda architecture, the existing CloudFront distribution will be replaced, resulting in a new endpoint URL and an empty cache.

Choosing an Architecture

Cost consideration

With 50KB response sizes and 350ms of image processing per image, the Object Lambda architecture is ~14% less expensive per image. As image sizes increase, the costs for the Object Lambda architecture will grow faster than for the API Gateway architecture, breaking even when response sizes are an average of 700 KB. For more information, refer to Transform & Query on the AWS S3 cost page or Cost.

Image Size

The default architecture limits response sizes to a maximum of 6 MB. The Object Lambda architecture is only limited by the 30 second CloudFront S3 Origin response timeout. Refer to Object Lambda architecture response latency for information surrounding expected processing time for basic requests of various sizes.

Updating an existing deployment

Updating an existing default deployment to a deployment using the S3 Object Lambda architecture will result in a new CloudFront distribution, with a new API endpoint and an empty cache. You will need to update references to this endpoint in your application to ensure functionality. For information on a workaround to use an alternate architecture type while maintaining the current endpoint URL and cache, refer to the instructions on maintaining the existing endpoint and cache when modifying architecture type.

Certain response headers in deployments using the S3 Object Lambda architecture are different from those in a default deployment. Of note, the X-Amz-Apigw-Id and X-Amzn-Trace-Id headers are no longer present, and the X-Cache header will no longer return "Error from CloudFront" when the solution returns an error, rather returning whether the Error itself was a cache hit/miss. If you depend on specific response elements, the default deployment will maintain that functionality.

Architecture details

This section describes the components and AWS services that make up this solution and the architecture details on how these components work together.

Demo UI

This solution optionally deploys a demo UI into your account to demonstrate the basic features of the solution. You can use the UI to interact directly with your new image handler API endpoint, using image files that already exist in your account.

This solution’s template contains a **Deploy Demo UI** parameter that’s activated (set to `Yes`) by default. If activated, this option deploys an additional Amazon S3 bucket and associated CloudFront distribution into your account.

Smart cropping

You can use this image request option to crop images using the facial recognition capabilities of Amazon Rekognition. To generate a cropped image, a Lambda function sends requests to Amazon Rekognition to identify faces in images and calculate crop areas.

Note

Amazon Rekognition supports only JPEG and PNG file formats for smart cropping. When using the Amazon Rekognition features with an image that isn’t JPEG or PNG, the solution automatically converts the image to PNG for use with Amazon Rekognition, then converts it back to the original format.

Content moderation

You can use this image request option to detect and blur inappropriate images. To detect an inappropriate image, a Lambda function sends requests to Amazon Rekognition to identify inappropriate content.

Note

Amazon Rekognition supports only JPEG and PNG file formats for content moderation. When using the Amazon Rekognition features with an image that isn't JPEG or PNG, the solution automatically converts the image to PNG for use with Amazon Rekognition, then converts it back to the original format.

Cross-origin resource sharing

This solution's template contains two parameters that activate Cross-origin resource sharing (CORS) for your image handler API: **CorsEnabledParameter** and **CorsOriginParameter**. CORS defines how client web applications loaded in one domain can interact with resources in a different domain. You can activate CORS for your image handler API 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.

Note

If you want to change your CORS configuration after deployment, you can activate or deactivate CORS by editing the `CORS_ENABLED (Yes/No)` and `CORS_ORIGIN` environment variables of the Lambda image handler function. See Using AWS Lambda environment variables in the *AWS Lambda Developer Guide* for more information.

Image URL signature

This solution's template contains three parameters that are required for the image URL signature functionality: **EnableSignatureParameter**, **SecretsManagerSecretParameter**, and **SecretsManagerKeyParameter**. To activate this feature:

- Set the **EnableSignatureParameter** parameter to `Yes`
- Set the **SecretsManagerSecretParameter** and **SecretsManagerKeyParameter** parameters to a valid secret and key that you originally created in Secrets Manager

Important

You are responsible for creating the Secrets Manager secret and key. For more information about Secrets Manager secret creation, refer to Create and manage secrets with AWS Secrets Manager 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 doesn't match, the solution returns an error message. 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:

Note

If you are using query parameter based edits, the query parameters must be sorted prior to signature generation.

```
const secret = '<YOUR_SECRET_VALUE_IN_SECRETS_MANAGER>';
const path = '/<YOUR_PATH>'; // Add the first '/' to path.
const query_params = '?<YOUR_QUERY_PARAMS>'
const sorted_query_params = query_params.slice(1).split("&").sort().join("&")
const signature = crypto.createHmac('sha256', secret).update(path
+(sorted_query_params ? `?${sorted_query_params}` : '')).digest('hex');
```

You can request your image using the image URL signature:

```
https://<distributionName>.cloudfront.net/<YOUR_PATH>?
query_param2=val2&query_param_1=val1&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 solution.

Note

If you plan to use the Expires query parameter alongside signed requests, ensure you include the expiration when creating your signature. For more information, refer to Include request expiration.

Default fallback image

This solution provides a default fallback image feature that returns the specified fallback image as a result of errors occur during processing, rather than a JSON object error message. This solution's template contains three parameters that are required for the default fallback image feature: **EnableDefaultFallbackImageParameter**, **FallbackImageS3BucketParameter**, and **FallbackImageS3KeyParameters**.

By default, this feature is deactivated. To activate this feature:

Note

Before activating this feature, if you use an S3 bucket policy in the fallback image S3 bucket, you must edit the bucket policy to allow the `CustomResourceFunction` and `ImageHandlerFunction` AWS Lambda functions to get the default fallback image object. For more information, see Adding a bucket policy by using the Amazon S3 console.

- Set the **EnableDefaultFallbackImageParameter** parameter to `Yes`
- Set the **FallbackImageS3BucketParameter** and **FallbackImageS3KeyParameter** parameters to a valid S3 bucket and object key

AWS services in this solution

AWS service	Description
Amazon CloudFront	**Core.** Provides a caching layer to reduce latency and the cost of image processing for

AWS service	Description
	subsequent identical requests. Allows pre/post processing of requests and responses.
AWS Lambda	**Core.** Runs functions to retrieve, modify, and invoke other services to analyze images. Also runs a function to support URL signature validation.
Amazon S3	**Core.** Stores images, logs, and a demo UI.
Amazon API Gateway	**Supporting.** Provides API endpoints to invoke Lambda functions. Only used if EnableS3O bjectLambda is set to "No".
Amazon S3 Object Lambda	**Supporting.** Provide S3 Origin to invoke Lambda functions. Only used if EnableS3O bjectLambda is set to "Yes".
AWS CDK	**Supporting.** Provides infrastructure as code constructs to generate the solution's underlyin g CloudFormation templates.
AWS CloudFormation	**Supporting.** Deploys the solution's underlying AWS resources.
AWS Identity and Access Management (IAM)	**Supporting.** Allows for fine-grained access permissions.
Amazon Rekognition	**Optional.** Uses machine learning (ML) to analyze images.
AWS Secrets Manager	**Optional.** Manages secrets to support URL signatures.

Plan your deployment

This section describes the cost, security, quotas, and other considerations before deploying the solution.

Supported AWS Regions

This solution uses AWS services that aren’t available in all AWS Regions. You must launch this solution in an AWS Region where these services are available. For the current availability of AWS services by Region, see the AWS Regional Services List.

This solution is available in the following AWS Regions:

Region name	
US East (Ohio)	Canada (Central)
US East (N. Virginia)	China (Beijing)
US West (Northern California)	China (Ningxia)
US West (Oregon)	Europe (Frankfurt)
Africa (Cape Town)	Europe (Ireland)
Asia Pacific (Hong Kong)	Europe (London)
Asia Pacific (Mumbai)	Europe (Milan)
Asia Pacific (Seoul)	Europe (Paris)
Asia Pacific (Singapore)	Europe (Stockholm)
Asia Pacific (Sydney)	Middle East (Bahrain)
Asia Pacific (Tokyo)	South America (São Paulo)

Opt-in Regions

An opt-in Region is an AWS Region that's deactivated by default. You can activate 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 four opt-in Regions:

- Asia Pacific (Hong Kong)
- Middle East (Bahrain)
- Africa (Cape Town)
- Europe (Milan)

When launched in an opt-in Region, this solution creates an S3 logging bucket for CloudFront in the US East (N. Virginia) Region. This is because CloudFront doesn't deliver access logs to buckets in the supported opt-in Regions. For more information about S3 buckets, refer to Choosing an Amazon S3 bucket for your standard logs in the *Amazon CloudFront Developer Guide*.

To deploy in an opt-in Region, the S3 bucket(s) that you provide for the **Source Buckets** parameter must be in the same Region where you launch the CloudFormation template.

Cost

You are responsible for the cost of the AWS services used while running this solution. As of this revision, the cost for running the solution with the default settings in the US East (N. Virginia) Region is approximately **$5.30 per month** for 100,000 new images, **$22.51 per month** for 1,000,000 new images, and **$7.65 per month** for 1,000,000 cached images (refer to the Sample cost table for the cost breakdown). These estimates use information about image sizing and lambda processing time from common use cases.

We recommend creating a budget through AWS Cost Explorer to help manage costs. Prices are subject to change. For full details, see the pricing webpage for each AWS service used in this solution.

Sample cost table

The following table provides a sample cost breakdown for deploying this solution with the default parameters in the US East (N. Virginia) Region for one month.

AWS service	Dimensions	Cost [USD] (100,000 new images)	Cost [USD] (1,000,000 new images)	Cost [USD] (1,000,000 cached images)
Amazon API Gateway *		$0.35	$3.50	$0
Amazon S3 Object Lambda	upload 50 KB image per request	$0.03	$0.25	$0
AWS Lambda	350 milliseconds processing time per image	$0.60	$6.03	$0
Amazon CloudFront	transfer 50 KB image per request	$0.42	$4.25	$4.25
AWS Secrets Manager **		$0.90	$5.40	$0.40
Amazon CloudWatch Dashboard ^		$3.00	$3.00	$3.00
Amazon CloudWatch Logs		$0.03	$0.33	$0.00
Total		**$5.30/$4.98**	**$22.51/$19.26**	**$7.65**

** The cost for Amazon API Gateway is incurred only if Enable S3 Object Lambda is set to No. The cost for Amazon S3 Object Lambda is incurred only if Enable S3 Object Lambda is set to Yes.*

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

^ The operational dashboard included with the solution may fall under the free tier, refer to CloudWatch pricing for the most up to date pricing information. For information on how to disable the deployment of the operational dashboard, refer to Optional Mappings.

Demo UI

If you choose to deploy the demo UI, the solution automatically deploys an additional CloudFront distribution and S3 bucket for storing the static website assets in your account. You are responsible for the incurred variable charges from these services. For more information, see Amazon S3 pricing.

Image modification and analysis

This cost estimate doesn't account for Amazon S3 PUT and GET requests, which can vary because modified images are cached in CloudFront, and because certain use cases require special-use capabilities such as smart cropping and content moderation with Amazon Rekognition. Using Amazon Rekognition features may incur additional charges. For more information, see Amazon Rekognition pricing.

There is no additional cost for using `sharp`, which is an open source library.

Security

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

Important

This solution creates CloudFront and API Gateway resources that are publicly accessible. Be aware that while this is likely appropriate for publicly facing websites, it might 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 CloudFront signed URLs. For private image handling use cases, AWS recommends using CloudFront signed URLs and implementing an API Gateway Lambda authorizer with CloudFront to secure your stack.

Demo UI

This solution optionally deploys a demo UI as a static website hosted in an S3 bucket. To help reduce latency and improve security, this solution includes a CloudFront distribution with an origin access identity, which is a CloudFront user that helps restrict access to the solution's website S3 bucket contents. For more information, refer to Restricting access to an Amazon S3 origin in the *Amazon CloudFront Developer Guide*.

IAM roles

IAM roles allow customers to assign granular access policies and permissions to services and users on the AWS Cloud. This solution creates IAM roles that grant the solution's Lambda functions access to create Regional resources.

Amazon API Gateway

This solution deploys an Amazon API Gateway REST API and uses the default API endpoint and SSL certificate. The default API endpoint supports TLSv1 security policy. It is recommended to use the TLS_1_2 security policy to enforce TLSv1.2+ with your own custom domain name and custom SSL certificate. For more information, refer to choosing a minimum TLS version for a custom domain in API Gateway in the Amazon API Gateway Developer Guide.

API Gateway custom domains TLS

How to custom domains

Amazon CloudFront

This solution deploys a web console hosted in an Amazon S3 bucket. To help reduce latency and improve security, this solution includes a CloudFront distribution with an origin access identity, which is a CloudFront user that provides public access to the solution's website bucket contents. For more information, see Restricting access to an Amazon S3 origin in the Amazon CloudFront Developer Guide.

Amazon CloudFront is deployed using the default CloudFront domain name and TLS certificate. To use a later TLS version, use your own custom domain name and custom SSL certificate. For more information, refer to using alternate domain names and HTTPS in the Amazon CloudFront Developer Guide.

Quotas

Service quotas, also referred to as limits, are the maximum number of service resources or operations for your AWS account. Make sure you have sufficient quota for each of the services implemented in this solution. For more information, see AWS service quotas.

Use the following links to go to the page for that service. To view the service quotas for all AWS services in the documentation without switching pages, view the information in the Service endpoints and quotas page in the PDF instead.

AWS CloudFormation quotas

Your AWS account has CloudFormation quotas that you should consider when launching the stack for this solution. By understanding these quotas, you can avoid limitation errors that can prevent you from deploying this solution successfully. For more information, see AWS CloudFormation quotas in the in the *AWS CloudFormation User Guide*.

AWS Lambda quotas

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

The default architecture for this solution does not support image responses larger than 6 MB, to allow for this functionality, use the S3 Object Lambda architecture by setting the Enable S3 Object Lambda template parameter to Yes. For more information, refer to Choosing an Architecture.

Amazon API Gateway quotas

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

Optional Mappings

The following sections provide information surrounding optional features and how to disable them. For each feature, use the following instructions.

1. Download the `dynamic-image-transformation-for-amazon-cloudfront.template` AWS CloudFormation template to your local hard drive.
2. Open the CloudFormation template with a text editor.
3. Locate the AWS CloudFormation template mapping section. It will be under the following location:

```
Mappings:
    Solution:
        Config:
```

4. Follow the instructions in the section for the feature you would like to disable.
5. Save the template and launch or update your CloudFormation stack using this modified template.

Operational Dashboard

The solution will deploy a Cloudwatch Dashboard for Solution Observability by default. This dashboard allows you to see the following information about your Dynamic Image Transformation for Amazon CloudFront deployment:

1. Lambda Errors
2. Lambda Duration
3. Lambda Invocations
4. CloudFront Requests
5. CloudFront Bytes Downloaded
6. Cache Hit Rate (% of requests to CloudFront which were returned from the cache)
7. Average Image Size
8. Estimated Cost (Based on us-east-1 pricing with a default deployment, doesn't include cost of observability)

Unless the dashboard is included in your AWS Free Tier, it will add a cost of $3 per month to your Dynamic Image Transformation for Amazon CloudFront deployment. To prevent the inclusion of the dashboard in your deployment, in combination with the instructions in Mappings, change the value under `DeployCloudWatchDashboard` from "Yes", to "No".

Sharp Size Limit

Sharp restricts the pixel size of input images to `268402689` by default (`16383 ^ 2`). To modify the value the solution passes to sharp, in combination with the instructions in Mappings, modify the value under `SharpSizeLimit` from "", to the pixel limit you choose, based on the following rules:

1. An empty string or a non-number string will use the existing default.
2. A value of "0" will remove the limit.
3. A positive integer value will set the limit to that value, for example, a mapping value of `"2500000"` will cause the value passed to Sharp to be `2500000`.
4. Negative and decimal values are not permitted, and may cause errors with image processing.

Deploy the solution

This solution uses CloudFormation templates and stacks to automate its deployment. The CloudFormation template specifies the AWS resources included in this solution and their properties. The CloudFormation stack provisions the resources that are described in the templates.

Before you launch the solution, review the cost, architecture, network security, and other considerations discussed earlier in this guide.

Important

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. AWS owns the data gathered though this survey. Data collection is subject to the AWS Privacy Notice.
To opt out of this feature, download the template, modify the AWS CloudFormation mapping section, and then use the AWS CloudFormation console to upload your updated template and deploy the solution. For more information, see the Anonymized data collection section of this guide.

AWS CloudFormation template

You can download the CloudFormation template for this solution before deploying it.

dynamic-image-transformation-for-amazon-cloudfront.template - Use this template to launch the solution and all associated components. The default configuration deploys CloudFront, API Gateway, Lambda, CloudWatch and EventBridge. You can customize the template to meet your specific needs.

Note

CloudFormation resources are created from AWS CDK constructs.

Before you launch the solution's AWS CloudFormation template, you must specify an S3 bucket in the **Source Buckets** template parameter. Use this S3 bucket to store the images that you want to manipulate. If you have multiple image source S3 buckets, you can specify them as commaseparated values. For lower latency, use an S3 bucket in the same AWS Region where you launch your CloudFormation template. Additional cross-region data transfer costs may apply if the solution is not deployed in the same AWS Region as the S3 bucket(s) provided in the Source Buckets template parameter.

Note

If you are launching from a supported opt-in Region, the source S3 bucket you created and provided as the **Source Buckets** template parameter must be in the same Region where you're launching the CloudFormation template.

We recommend deploying the optional demo UI when you first deploy the solution to test the solution's functionality. For more information, refer to Use the demo UI.

Note

If you have previously deployed this solution, see Update the solution for update instructions.
Dynamic Image Transformation for Amazon CloudFront version 6.0 and newer include significant changes, and you can't update the solution from versions before 6.0 to version 6.0 or later. To use version 6.0 or later, launch a new stack using version 6.x of the CloudFormation template and uninstall your previous version of this solution.

Launch the stack

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

Time to deploy: Approximately 15 minutes

1. Sign in to the AWS Management Console and select the button to launch the `dynamic-image-transformation-for-amazon-cloudfront` AWS CloudFormation template.
2. Sign into AWS Management Console and select the button to launch `dynamic-image-transformation-for-amazon-cloudfront` CloudFormation template.

   Launch solution

3. The template launches in the US East (N. Virginia) Region by default. To launch the solution in a different AWS Region, use the Region selector in the console navigation bar. For a list of which AWS Regions support this solution, see Supported AWS Regions.
4. On the **Create stack** page, verify that the correct template URL is in the **Amazon S3 URL** text box and choose **Next**.
5. On the **Specify stack details** page, assign a name to your solution stack. For information about naming character limitations, see IAM and AWS STS quotas in the *AWS Identity and Access Management User Guide*.
6. Under **Parameters**, review the parameters for this solution template and modify them as necessary. This solution uses the following default values.

Parameter	Default	Description
CORS Enabled	No	Choose whether to activate CORS. For information about this parameter, refer to Cross-origin resource sharing (CORS).
CORS Origin	*	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 (Ex: http://example.domain) to restrict cross-site access to your API. **Note:** This value is ignored if **CORS_ENABLED** is set to No.
Source Buckets	<Requires input>	Specifies the S3 bucket (or buckets) in your account that contain(s) the images that you manipulate. To specify

Parameter	Default	Description
		multiple buckets, separate them by commas.
Enable S3 Object Lambda	No	Determines which component to use to use as the CloudFront distribution origin. No uses API gateway, Yes uses an S3 Object Lambda Access Point, which supports images larger than the existing 6 MB size limit. Only the origin in use will be created by the template.
Deploy Demo UI	Yes	The demo UI that deploys to the `Demo` S3 bucket. For more information refer to Use the demo UI.
Log Retention Period	180	Specifies the number of days to retain Lambda log data in CloudWatch logs.
Enable Signature	No	Choose whether to activate the image URL signature feature. For information about this feature, refer to Image URL signature.

Parameter	Default	Description
SecretsManager Secret	*<Optional input>*	Define the Secrets Manager secret name that contains the secret key for the image URL signature. **Note:** This value is ignored if the **Enable Signature** parameter is set to No.
SecretsManager Key	*<Optional input>*	Define the Secrets Manager secret key that contains the secret value to create the image URL signature. **Note:** This value is ignored if the **Enable Signature** parameter is set to No.
Enable Default Fallback Image	No	Choose whether to activate the default fallback image feature. For information about this feature, refer to Default fallback image.
Fallback Image S3 Bucket	*<Optional input>*	Specify the S3 bucket which contains the default fallback image. **Note:** This value is ignored if the **Enable Default Fallback Image** parameter is set to No.

Parameter	Default	Description
Fallback Image S3 Key	*<Optional input>*	Specify the default fallback image S3 object key, including prefix. See Creating object key names for more information. **Note:** This value is ignored if the **Enable Default Fallback Image** parameter is set to No.
AutoWebP	No	Choose whether to automatically convert responses to the WebP image formats if the Accept request header allows it.
Origin Shield Region	`Disabled`	The Region to set up the Origin Shield caching layer for the CloudFront distribut ion. May result in a better cache hit ratio, as well as lower latency on repeat requests in new regions. For more information on choosing an Origin Shield region, see the Amazon CloudFront Developer Guide.
CloudFront PriceClass	PriceClass_All	The CloudFront price class to use. For more informati on, refer to Choosing the price class for a CloudFront distribution in the *Amazon CloudFront Developer Guide*.

Parameter	Default	Description
Use Existing CloudFront Distribution	No	Choose whether to deploy the solution in a way that it can be attached to an existing CloudFront distribution. If No is selected, a CloudFront distribution will be created for you. If you have selected `Yes`, manual action will need to be performed to finish the attachment, refer to Attaching an Existing CloudFront distribution for more information.
Existing CloudFront Distribution ID	`<Optional Input>`	The Distribution ID for the existing CloudFront distribution being attached to. This field is required if Use Existing CloudFront Distribution is set to Yes, and will be used to set up IAM permissions, metrics, and CloudFormation template outputs. **Note:** This value is ignored if **Use Existing CloudFront Distribution** is set to No.

7. Choose **Next**.
8. On the **Configure stack options** page, choose **Next**.
9. On the **Review and create** page, review and confirm the settings. Select the box acknowledging that the template creates IAM resources.
10. Choose **Submit** 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.

Attaching an existing CloudFront distribution

If you've deployed your stack and have set the Use Existing CloudFront Distribution template parameter to Yes, use the following instructions to complete your setup.

Note

In the following instructions, UUID is used to reference the deployment UUID of your Dynamic Image Transformation for Amazon CloudFront stack. You can find this value by inspecting the Physical ID of a `AWS::CloudFront::Function` deployed in your stack, and extracting the value found after the word `modifier-`.
Whether you are using the API Gateway architecture or S3 Object Lambda architecture is determined by the value of the Enable S3 Object Lambda template parameter in your CloudFormation stack (Yes = S3 Object Lambda architecture, No = API Gateway architecture). You can ignore any instructions that are for the opposite architecture type.

Setting the Origin

1. In the CloudFront console, navigate to the distribution you indicated in the Existing CloudFront Distribution ID template parameter.
2. Select the Origins tab and click **Create origin**.
3. For the API Gateway architecture, set the Origin domain as the API Gateway execution link. This value can be found by placing the Physical ID of the stack's AWS::ApiGateway::RestApi in the search field and selecting LambdaRestApi under API Gateway.
4. For the S3 Object Lambda architecture, set the Origin domain as the Object Lambda Access Point alias, this value will begin with `sih-olap-{UUID_11}`, where UUID_11 is the first 11 characters of your UUID.
5. Leaving other values as their default, set the Origin path to `/image`.
6. For the S3 Object Lambda architecture, set the Origin access control to the deployed value for your stack. This value will be equal to `SIH-origin-access-control-${UUID}`
7. Select **Create origin**.

Setting the behavior

1. In the CloudFront console, navigate to the distribution you indicated in the Existing CloudFront Distribution ID template parameter.
2. Select the Behaviors tab and choose **Create behavior**
3. Set the Path pattern you'd like to point to your solution instance, in a Solution created distribution, this is Default (*)
4. Set the Origin to the Origin created in the previous section.
5. Set the Viewer Protocol policy to `Redirect HTTP to HTTPS`
6. Set the Cache Policy to the one named `ServerlessImageHandler-${UUID}`.
7. Set the Origin request policy to the one named `ServerlessImageHandler-${UUID}`.
8. For the API Gateway Architecture, set the Viewer request Function type to CloudFront Functions, and the Function ARN to the one named `sih-apig-request-modifier-${UUID}`.
9. For the Object Lambda Architecture, set the Viewer request Function type to CloudFront Functions, and the Function ARN to the one named `sih-ol-request-modifier-${UUID}`. As well, set the Viewer response Function type to CloudFront Functions, and the Function ARN to the one named `sih-ol-response-modifier-${UUID}`.

10Select **Create behavior**.

Monitor the solution with AppRegistry

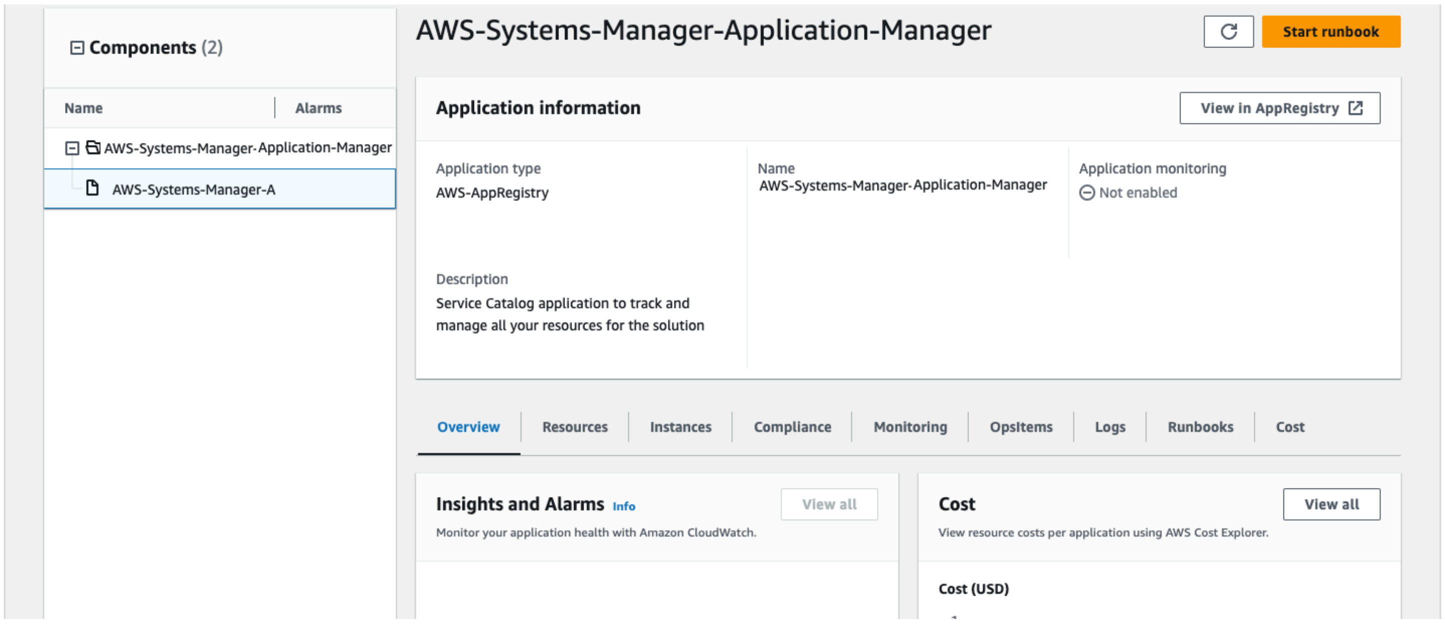
The solution includes a Service Catalog AppRegistry resource to register the CloudFormation template and underlying resources as an application in both Service Catalog AppRegistry and AWS Systems Manager Application Manager.

AWS Systems Manager Application Manager gives you an application-level view into this solution and its resources so that you can:

- Monitor its resources, costs for the deployed resources across stacks and AWS accounts, and logs associated with this solution from a central location.
- View operations data for the resources of this solution in the context of an application. For example, deployment status, CloudWatch alarms, resource configurations, and operational issues.

The following figure depicts an example of the application view for the solution stack in Application Manager.

Depicts solution stack in Application Manager



Application information

Application type: AWS-AppRegistry

Name: AWS-Systems-Manager-Application-Manager

Application monitoring: Not enabled

Description: Service Catalog application to track and manage all your resources for the solution

Insights and Alarms Info

Monitor your application health with Amazon CloudWatch.

Cost

View resource costs per application using AWS Cost Explorer.

Cost (USD)

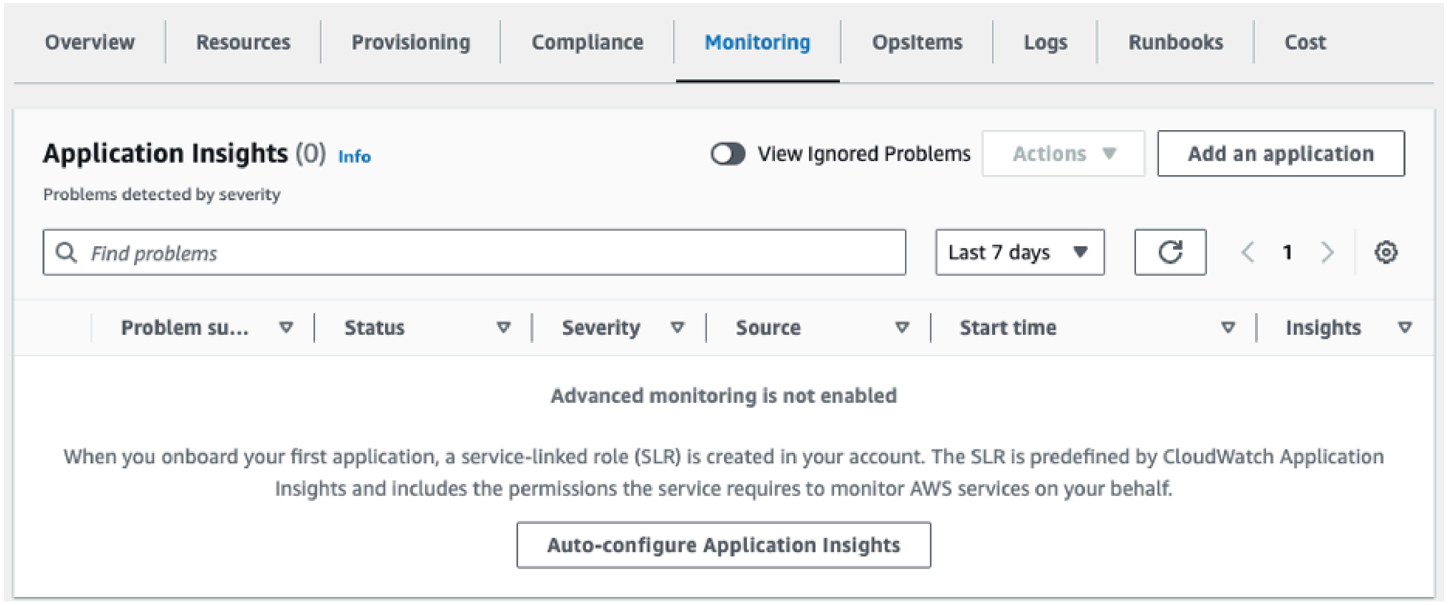
Activate CloudWatch Application Insights

1. Sign in to the Systems Manager console.

2. In the navigation pane, choose **Application Manager**.
3. In **Applications**, search for the application name for this solution and select it.

   The application name will have **App Registry** in the **Application Source** column, and will have a combination of the solution name, Region, account ID, or stack name.
4. In the **Components** tree, choose the application stack you want to activate.
5. In the **Monitoring** tab, in **Application Insights**, select **Auto-configure Application Insights**.

Application Insights dashboard showing no detected problems and option to auto-configure.

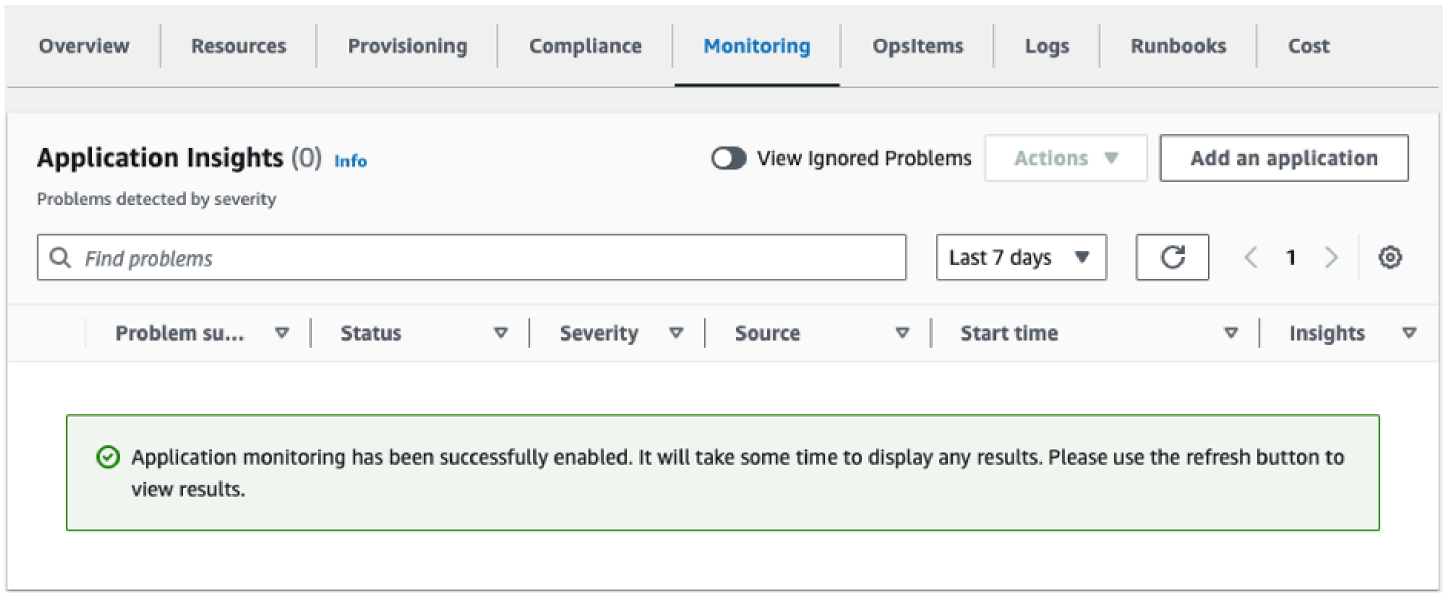


Advanced monitoring is not enabled

When you onboard your first application, a service-linked role (SLR) is created in your account. The SLR is predefined by CloudWatch Application Insights and includes the permissions the service requires to monitor AWS services on your behalf.

Monitoring for your applications is now activated and the following status box appears:

Application Insights dashboard showing successful monitoring activation message.



Application monitoring has been successfully enabled. It will take some time to display any results. Please use the refresh button to view results.

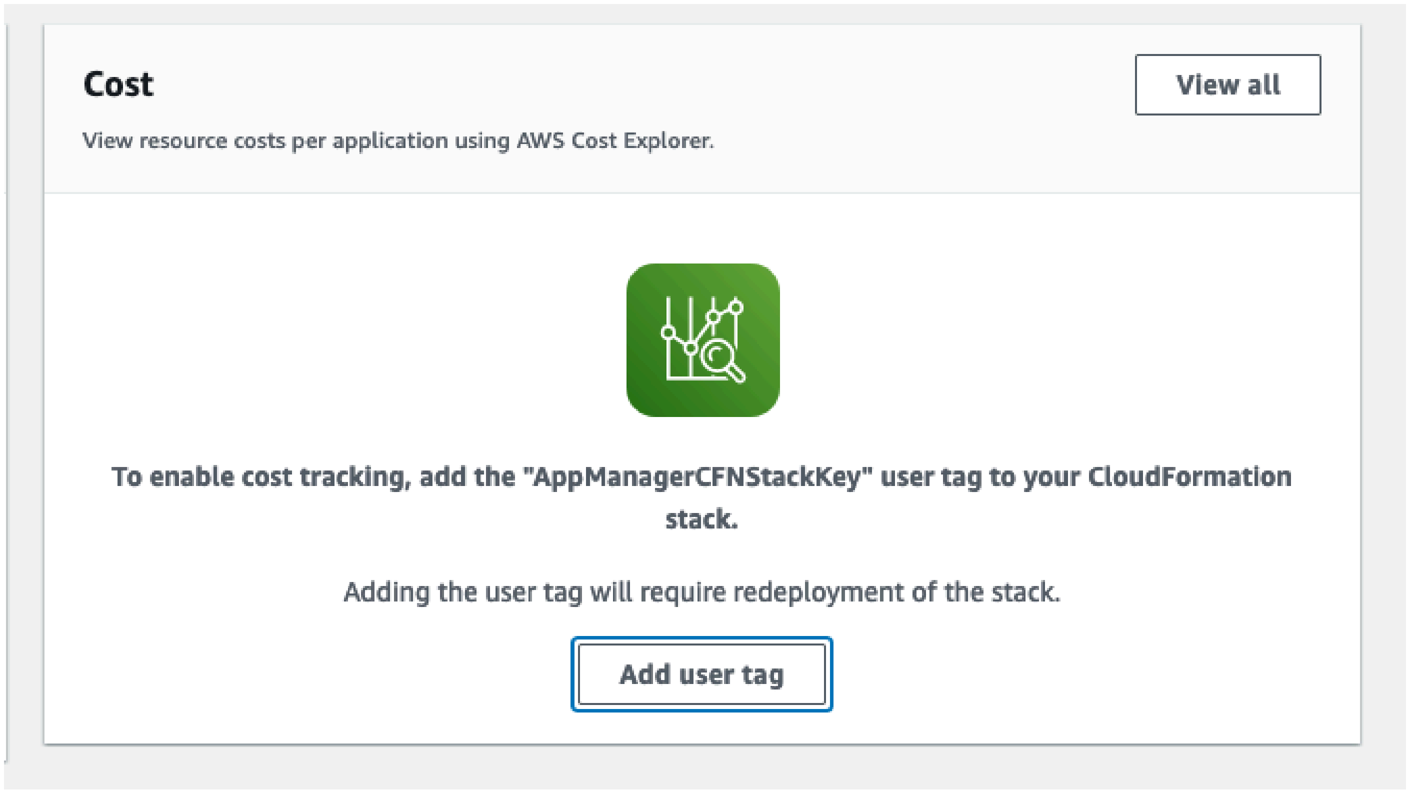
Confirm cost tags associated with the solution

After you activate cost allocation tags associated with the solution, you must confirm the cost allocation tags to see the costs for this solution. To confirm cost allocation tags:

1. Sign in to the Systems Manager console.
2. In the navigation pane, choose **Application Manager**.
3. In **Applications**, choose the application name for this solution and select it.

   The application name will have **App Registry** in the **Application Source** column, and will have a combination of the solution name, Region, account ID, or stack name.
4. In the **Overview** tab, in **Cost**, select **Add user tag**.

   Screenshot depicting the Application Cost add user tag screen


Cost

View resource costs per application using AWS Cost Explorer.

To enable cost tracking, add the "AppManagerCFNStackKey" user tag to your CloudFormation stack.

Adding the user tag will require redeployment of the stack.

5. On the **Add user tag** page, enter `confirm`, then select **Add user tag**.

The activation process can take up to 24 hours to complete and the tag data to appear.

Activate cost allocation tags associated with the solution

After you activate Cost Explorer, you must activate the cost allocation tags associated with this solution to see the costs for this solution. The cost allocation tags can only be activated from the management account for the organization. To activate cost allocation tags:

1. Sign in to the AWS Billing and Cost Management and Cost Management console.
2. In the navigation pane, select **Cost Allocation Tags**.
3. On the **Cost allocation tags** page, filter for the AppManagerCFNStackKey tag, then select the tag from the results shown.
4. Choose **Activate**.

AWS Cost Explorer

You can see the overview of the costs associated with the application and application components within the Application Manager console through integration with AWS Cost Explorer, which must be first activated. Cost Explorer helps you manage costs by providing a view of your AWS resource costs and usage over time. To activate Cost Explorer for the solution:

1. Sign in to the AWS Cost Management console.
2. In the navigation pane, select **Cost Explorer** to view the solution's costs and usage over time.

Update the solution

If you have previously deployed the solution, follow this procedure to update the CloudFormation stack to get the latest version of the solution's framework.

Important

Dynamic Image Transformation for Amazon CloudFront version 6.0 and newer include significant changes, and you can't update the solution from versions prior to 6.0 to version 6.0 or later. To use version 6.0 or later, launch a new stack using version 6.x of the CloudFormation template and uninstall your previous version of this solution. Modifying the architecture of an existing deployment by changing the value of the `Enable S3 Object Lambda` template parameter will cause a deletion and recreation of the CloudFront distribution associated with the deployment. This recreation will result in a new API endpoint URL and an empty cache. For information on a workaround to use an alternate architecture type while maintaining the current endpoint URL and cache, refer to the instructions on maintaining the existing endpoint and cache when modifying architecture type.

1. Sign in to the AWS CloudFormation console, select your existing Dynamic Image Transformation for Amazon CloudFront CloudFormation stack, and select **Update**.
2. Select **Replace current template**.
3. Under **Specify template**:
   a. Select **Amazon S3 URL**.
   b. Copy the link of the `dynamic-image-transformation-for-amazon-cloudfront.template` AWS CloudFormation template.
   c. Paste the link in the **Amazon S3 URL** box.
   d. This link will point to the latest template by default, to modify which version you update to, replace the word `latest` with the desired version.
      i. For example: `https://solutions-reference.s3.amazonaws.com/dynamic-image-transformation-for-amazon-cloudfront/latest/dynamic-image-transformation-for-amazon-cloudfront.template` would become `https://solutions-reference.s3.amazonaws.com/dynamic-image-transformation-`

`for-amazon-cloudfront/v7.0.0/dynamic-image-transformation-for-amazon-cloudfront.template`

Note

Alongside the rename in v7.0.0 from Serverless Image Handler to Dynamic Image Transformation for Amazon CloudFront, the location of the cloudformation template has changed, if you'd like to follow the above instructions for a version before v7.0.0, use the following template URL as a baseline: https://solutions-reference.s3.amazonaws.com/serverless-image-handler/latest/serverless-image-handler.template

a. Verify that the correct template URL shows in the **Amazon S3 URL** text box, and choose **Next**. Choose **Next** again.

   1. Under **Parameters**, review the parameters for the template and modify them as necessary. For details about the parameters, see Deployment process overview.
   2. Choose **Next**.
   3. On the **Configure stack options** page, choose **Next**.
   4. On the **Review** page, review and confirm the settings. Select the box acknowledging that the template creates IAM resources.
   5. Choose **View change set** and verify the changes.
   6. 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.

Backward compatibility

This 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) will source images from the first bucket in the `SOURCE_BUCKETS` environment variable by default. To use a different bucket, you can use the `s3:BucketName` tag in your request or you can adjust which bucket is first in the environment variables section of your image handler Lambda function. See Using AWS Lambda environment variables in the *AWS Lambda Developer Guide* for more information.

Thumbor compatibility

You can specify Thumbor image requests as you normally would, with filters and other relevant properties added on as suffixes to the default CloudFront **ApiEndpoint**. For more information about using Thumbor, see List of supported Thumbor filters.

Note

Dynamic Image Transformation for Amazon CloudFront includes a Thumbor-style interface in the API; however, those requests are mapped to comparable Sharp library calls, and might not include all available Thumbor filters. For more information about available Thumbor-style filters, see List of supported Thumbor filters.

Custom compatibility

You can specify custom image requests that used the version 3.x and earlier solution versions' rewrite feature as you normally would. First, you must update the `REWRITE_MATCH_PATTERN` and `REWRITE_SUBSTITUTION` environment variables for your image handler function with the appropriate (JavaScript/ECMAScript-compatible) regular expressions and strings. For example:

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

For more information about using custom image requests, see Custom image requests.

Maintain existing endpoint and cache when modifying architecture type

With the release of the Object Lambda architecture, customers have the ability to enable an architecture which supports larger images. Modifying an existing distribution to use this architecture will cause a deletion and recreation of the CloudFront distribution associated with the deployment. This will result in a change to the domain name, as well as the cache being cleared. The following workaround can be used to modify the architecture type while avoiding this deletion.

Note

This workaround is not officially supported, and may run into instability. This workaround requires that both stacks are maintained in order to maintain functionality. Any updates to the original stack may undo some of the changes performed here. You may experience downtime

1. Follow the process for deploying a new Dynamic Image Transformation for Amazon CloudFront stack. Refer to deploy the solution for additional guidance on this step.
2. Modify the Enable S3 Object Lambda template parameter to select your desired architecture.
3. Set Use Existing CloudFront Distribution to `Yes`
4. Set Existing CloudFront Distribution Id to the ID of the Image Handler distribution for your existing stack.
5. Set the remaining template parameters to the same values used in the original deployment.
6. Deploy the stack.
7. Upon completion of the deployment, follow the instructions in Attaching an Existing CloudFront distribution to attach the CloudFront distribution referenced to the newly deployed resources. This will require that you overwrite the existing values on the distribution.

Troubleshooting

If you need help with this solution, contact AWS Support to open a support case for this solution.

Contact AWS Support

If you have AWS Developer Support, AWS Business Support, or AWS Enterprise Support, you can use the Support Center to get expert assistance with this solution. The following sections provide instructions.

Create case

1. Sign in to Support Center.
2. Choose **Create case**.

How can we help?

1. Choose **Technical**.
2. For **Service**, select **Solutions**.
3. For **Category**, select **Other Solutions**.
4. For **Severity**, select the option that best matches your use case.
5. When you enter the **Service**, **Category**, and **Severity**, the interface populates links to common troubleshooting questions. If you can't resolve your question with these links, choose **Next step: Additional information**.

Additional information

1. For **Subject**, enter text summarizing your question or issue.
2. For **Description**, describe the issue in detail.
3. Choose **Attach files**.
4. Attach the information that AWS Support needs to process the request.

Help us resolve your case faster

1. Enter the requested information.
2. Choose **Next step: Solve now or contact us**.

Solve now or contact us

1. Review the **Solve now** solutions.
2. If you can’t resolve your issue with these solutions, choose **Contact us**, enter the requested information, and choose **Submit**.

Uninstall the solution

You can uninstall the solution from the AWS Management Console or by using the AWS Command Line Interface (AWS CLI). You must manually delete the S3 buckets created by this solution. AWS solutions don't automatically delete these resources 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 this solution's installation stack.
3. Choose **Delete**.

Using AWS Command Line Interface

Determine whether the AWS CLI is available in your environment. For installation instructions, see What Is the AWS Command Line Interface? in the *AWS CLI User Guide*. After confirming that the AWS CLI is available, run the following command.

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

Deleting the Amazon S3 buckets

This solution is configured to retain the solution-created S3 buckets if you decide to delete the AWS CloudFormation stack to prevent accidental data loss. After uninstalling the solution, you can manually delete this S3 bucket if you don't need to retain the data. Follow these steps to delete the Amazon S3 buckets.

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 CloudFormation template to delete the Amazon S3 bucket automatically. Before deleting the stack, change the deletion behavior in the CloudFormation DeletionPolicy attribute.

Note

Neither of these methods deletes the source bucket you created and provided as a parameter to the CloudFormation template.

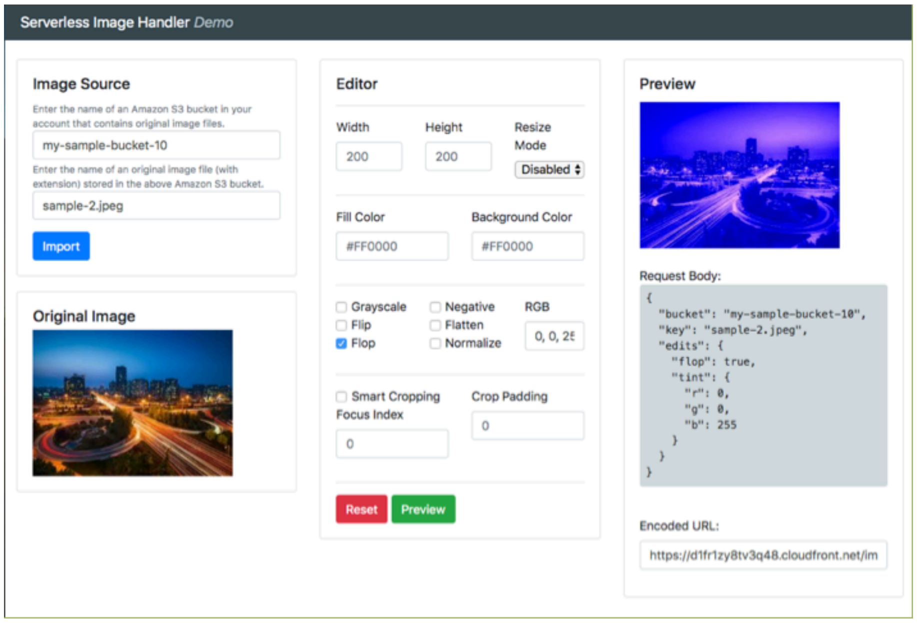
Use the solution

This section provides a user guide for utilizing the AWS solution.

Use the demo UI

The solution provides an optional demo UI that you can deploy into your AWS account to display basic capability and functionality. With this UI, you can interact directly with the new image handler using images from the specified Amazon S3 buckets in your account.

Screenshot of demo UI showing image source, original image, editing options, preview, code, and encoded URL.



Enter the name of an Amazon S3 bucket in your account that contains original image files.
my-sample-bucket-10

Enter the name of an original image file (with extension) stored in the above Amazon S3 bucket.
sample-2.jpeg

Width: 200
Height: 200
Resize Mode: Disabled

Fill Color: #FF0000
Background Color: #FF0000

☐ Grayscale ☐ Negative RGB: 0, 0, 25
☐ Flip ☐ Flatten
☑ Flop ☐ Normalize

☐ Smart Cropping Focus Index: 0
Crop Padding: 0

Request Body:

```
{
  "bucket": "my-sample-bucket-10",
  "key": "sample-2.jpeg",
  "edits": {
    "flop": true,
    "tint": {
      "r": 0,
      "g": 0,
      "b": 255
    }
  }
}
```

Encoded URL:
https://d1fr1zy8tv3q48.cloudfront.net/im

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. Sign in to the AWS CloudFormation console.
2. Select the solution's installation stack.
3. Choose the **Outputs** tab, and then select value for the **DemoUrl** key. The Dynamic Image Transformation for Amazon CloudFront Demo UI opens in your browser.
4. In the **Image Source** card, perform the following actions:
    a. Specify a bucket name to use for the demo. The bucket you specify must be listed in the `SOURCE_BUCKETS` environment variable of the AWS Lambda function.

   b. Specify an image key to use for the demo. You must include the file extension in the key.

5. Select **Import**. The original image appears in the **Original Image** card.
6. In the **Editor** card, 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 Dynamic Image Transformation for Amazon CloudFront demo UI offers a limited set of image edits and doesn't include the full scope of capabilities offered by the Image Handler API and the image URL signature. We recommended using your own frontend application for image modification.

Use the solution with a frontend application

In your frontend 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 API call. Follow these steps to retrieve your API endpoint for the solution:

1. Sign in to the AWS CloudFormation console.
2. On the **Stacks** page, select this solution's installation stack.
3. Choose the **Outputs** tab. The domain name appears as the value for the **ApiEndpoint** key. This URL is the endpoint URL for your newly provisioned image handler API.

To use the solution with your frontend application, use the following example syntax for the API call:

```
https://<ApiEndpoint>/<encodedRequest>
```

Create and use image requests

This solution generates a CloudFront domain name that gives you access to both original and modified images through the image handler API. You can specify parameters such as the image's location and edits to be made in a JSON object on the frontend.

Follow these step-by-step instructions to create image requests:

> **Note**
>
> The following image formats are supported for modifications: JPG/JPEG, PNG, TIFF/TIF, WEBP, GIF and 8-bit AVIF. For retrieval, the following formats are supported: All those listed previously, as well as AVIF (all bit depths) and SVG. Edited SVG files will be converted to .png by default.

1. Retrieve your API endpoint for the solution. Refer to Use the solution with a frontend application for instructions.
2. In a code sandbox, or in your frontend application, create a new `imageRequest` JSON object. This object contains the key-value pairs needed to successfully retrieve and perform edits on your images. Using the following code sample and the sharp documentation, adjust the following properties to meet your image editing requirements.
   - **Bucket** - Specify the S3 bucket containing your original image file. This is the name that's 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 Lambda function. See Using AWS Lambda environment variables in the *AWS Lambda Developer Guide* for more information.
   - **Key** - Specify the filename of your original image. This name should include the file extension and subfolders between its location and the root of the bucket. For example, `folder1/folder2/image.jpeg`.
   - **Edits** - Specify image edits as key-value pairs. If you don't specify image edits, the original image returns with no changes made.

   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.jpeg>",
       edits: {
           grayscale: true
       }
   ```

```
})
```

1. Stringify the JSON request object. For example:

```
const stringifiedObject = JSON.stringify(<myObject>);
```

2. Base64 encode the JSON string. For example:

```
const encodedObject = btoa(<stringifiedObject>);
```

3. Append the encoded string onto the CloudFront URL. For example:

```
const url = '${<ApiEndpoint>}/${<encodedObject>}';
```

4. Use that URL either in the JavaScript as part of a `GET` request, or in the frontend as part of an HTML `img` tag's `src` property.

For information regarding how to use additional features in an image request, refer to Dynamically resize photos, Use smart cropping, Use round cropping, and Activate and use content moderation. For additional features supported by `sharp`, refer to the sharp documentation.

Note

The following filters are not supported for multi-page GIF images due to limitations in the underlying libraries: **rotate**, **smartCrop**, **roundCrop**, and **contentModeration**.

Dynamically resize photos

This solution offers the following **fit** options to dynamically resize an image: `cover`, `contain`, `fill`, `inside`, and `outside`. Refer to the sharp documentation for a description of each fit. For example:

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>",
    key: "<myImage.jpeg>",
    edits: {
        resize: {
            width: 200,
            height: 250,
            fit: "cover"
        }
    }
})
```

If you use `contain` as the resize **fit** mode, you can specify the color of the fill by providing the hex code of the color you want to use. For example:

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>",
    key: "<myImage.jpeg>",
    edits: {
        resize: {
            width: 200,
            height: 250,
            fit: "contain",
            background: {
                r: 255,
                g: 0,
                b: 0,
                alpha: 1
            }
        }
    }
})
```

Edit images

You can use this solution to edit your images, such as rotating them or changing the coloring to negative. Refer to the sharp documentation for a description of each operation. For example, to produce a negative of an image, enter the following:

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

Restricted operations

Certain Sharp operations are restricted by the solution to help enhance security. This includes (but may not be limited to):

- clone
- metadata
- stats
- composite (Though this is permitted through the use of overlayWith)
- certain output options (Including toFile, toBuffer, tile and raw)

For an exact list of allow-listed Sharp operations, you can visit constants.ts on the Solution GitHub repository.

Use smart cropping

This solution uses Amazon Rekognition for face detection in images submitted for smart cropping. To activate smart cropping on an image, add the **smartCrop** property to the **edits** property in the image request.

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

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

  The following **smartCrop** variables are shown in the following code sample:

  smartCrop.faceIndex(optional, number) - Specifies which face to focus on if multiple are present within an original image. The solution indexes detected faces in a zero-based array from the largest detected face to the smallest. If this value isn't 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 solution applies the padding value to all sides of the cropped image.

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

Note

smartCrop is not supported for animated (such as, GIF) images.

Use round cropping

This solution can crop images in a circular pattern. To activate round cropping on an image, add the **roundCrop** property to the **edits** property in the image request.

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

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

The following **roundCrop** variables are shown in the following code sample:

roundCrop.rx (optional, number) - Specifies the radius along the x-axis of the ellipse. If a value isn’t provided, the image handler defaults to a value that’s half the length of the smallest edge. **roundCrop.ry (optional, number)** - Specifies the radius along the y-axis of the ellipse. If a value isn’t provided, the image handler defaults to a value that’s 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 isn’t provided, the image handler defaults to a value that’s 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 isn’t provided, the image handler defaults to a value that’s half of the width.

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

Note

roundCrop is not supported for animated (such as, GIF) images.

Overlay an image

This solution can overlay images on top of others, for cases like watermarking copyrighted image. To overlay an image, add the **overlayWith** property to the **edits** property in the image request.

overlayWith(optional, object) - Overlays an image on top of the original. For example:

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>",
    key: "<myImage.jpeg>",
    edits: {
    overlayWith: {
        bucket: "<myImageBucket>",
        key: "<myOverlayImage.jpeg>",
        alpha: 0-100, // Opaque (0) to Transparent (100)
        wRatio: 0-100, // Ratio of the underlying image that the overlay width should
 be
        hRatio: 0-100, // Ratio of the underlying image that the overlay height should
 be
        options: {
                top: "-10p",
                left: 150
            }
        }
    }
})
```

The following **overlayWith** variables are shown in the previous code sample:

- **overlayWith.bucket (required, string)** - Specifies the bucket that the overlay image should be retrieved from. This bucket must be present in the `SOURCE_BUCKETS` parameter.
- **overlayWith.key (required, string)** - Specifies the object key that is used for the overlay image.
- **overlayWith.alpha (optional, number)** - Specifies the opacity that should be used for the overlay image. This can be set from 0 (fully opaque) and 100 (fully transparent).
- **overlayWith.wRatio (required, number)** - Specifies the percentage of the width of underlying image that the overlay image should be sized to. This can be set from 0 and 100, where 100 indicates that the overlay image has the same width as the underlying image.
- **overlayWith.hRatio (required, number)** - Specifies the percentage of the height of underlying image that the overlay image should be sized to. This can be set from 0 and 100, where 100 indicates that the overlay image has the same height as the underlying image.
- **overlayWith.options.top (optional, number | string)** - Specifies the distance in pixels from the top edge of the underlying photo that the overlay should be placed. A number formatted as a string with a p at the end is treated as a percentage.
- **overlayWith.options.left (optional, number | string)** - Specifies the distance in pixels from the left edge of the underlying photo that the overlay should be placed. A number formatted as a string with a p at the end is treated as a percentage.

Note

overlayWith is not fully supported for animated (such as, GIF) images. Instead, only the first frame will receive an overlay.

Overwrite animated status

This solution assumes that GIF files with multiple pages should be animated. If you'd like to indicate that a GIF should not be animated, or that another file type should be animated, include the animated property in the edits property in the image request.

- **animated (optional, boolean)** - Overwrites the initial animated status of the image. If the value is `true` , the solution will attempt to process the image as animated. For example:

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

  If it is `false`, the solution will process the image as a still image. For example:

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>",
    key: "<myImage.gif>",
    edits: {
        animated: false
    }
})
```

Note

If an image does not have multiple pages, it will always be processed as still, regardless of the **edits.animated** property. The following filters are not supported for images that are animated: **rotate**, **smartCrop**, **roundCrop**, and **contentModeration**.

Activate and use content moderation

This solution can detect inappropriate content using Amazon Rekognition. To activate content moderation, add the **contentModeration** property to the **edits** property in the image request.

- **contentModeration (optional, boolean || object)** - Activates the content moderation feature for an original image. If the value is true, then the feature detects inappropriate content using Amazon Rekognition with a minimum confidence that's set higher than 75%. If Amazon Rekognition finds inappropriate content, the solution blurs the image. For example:

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

  The following **contentModeration** variables are shown in the following code sample:

- **contentModeration.minConfidence (optional, number)** - Specifies the minimum confidence level for Amazon Rekognition to use. Amazon Rekognition only returns detected content that's higher than the minimum confidence. If a value isn't 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 isn't provided, the default value is set to 50.
- **contentModeration.moderationLabels (optional, array)** - 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.jpeg>",
    edits: {
        contentModeration: {
            minConfidence: 90,  // minimum confidence level for inappropriate content
            blur: 80,           // amount to blur image
            moderationLabels: [ // labels to search for
               "Hate Symbols",
               "Smoking"
             ]
        }
    }
})
```

Note

contentModeration is not supported for animated (such as, GIF) images.

Include custom response headers

This solution allows you to include headers you'd like returned alongside the response, as part of your request.

- **headers (optional, object)** - Includes the provided headers in the response. Header should be written in Pascal-Case and cannot overwrite headers that would otherwise be present in the response (Except for Cache-Control).

```
const imageRequest = JSON.stringify({
    bucket: "<myImageBucket>",
    key: "<myImage.jpeg>",
    headers: {
        "Cache-Control":"max-age=86400,public"
        "Custom-Header":"some-custom-value"
    }
})
```

Note

A deny-list is maintained which restricts which headers can be included with this feature. Headers which may serve a purpose for the browser or are used to support authentication/

authorization are included in this deny-list. For an exact list of the regular expressions which are restricted, visit constants.ts on the Solution GitHub repository.
The presence of the `expires` query parameter will cause the Cache-Control header to be overridden, regardless of any value provided in the headers field.
If your deployment is using the S3 Object Lambda Architecture , the headers at this link cannot be included as custom headers.

Include request expiration

This solution supports the expires query parameter, which is used to decide whether the Lambda should process a request. If an Expiry date is in the future, the request will be processed. If the Expiry date has already passed, the Lambda will return a 400 Bad Request error with the code: ImageRequestExpired.

Values in the expires query parameter are expected in the format: YYYYMMDDTHHmmssZ. For example: April 9th, 2024, at 10:30:15 UTC -4 would become 20240409T143015Z.

Request expiry is compatible with signatures, and should be included as part of the path when signing a request. For example:

```
const secret = '<YOUR_SECRET_VALUE_IN_SECRETS_MANAGER>';
const path = '/<YOUR_PATH>'; // Add the first '/' to path.
const expires = 'expires=<YOUR_EXPIRY>';
const to_sign = `${path}?${expires}`
const signature = crypto.createHmac('sha256', secret).update(to_sign).digest('hex');
```

Note

If the expires query parameter is being used in conjunction with any query parameter based edits. When generating a signature, please ensure that the query parameters are sorted. For more information, see Image URL signature.

Use supported Query Parameter edits

The solution supports the definition of certain image edits through the use of query parameters. These query parameters can be used by themselves, or in conjunction with base64 or Thumbor-style edits. For example

```
https://<ApiEndpoint>/<image.jpeg>?format=<FormatType>
```

```
https://<ApiEndpoint>/<base64EncodedRequest>?format=<FormatType>
```

```
https://<ApiEndpoint>/<modification>/<image.jpeg>?format=<FormatType>
```

If a query parameter includes an edit already included in the request, it will overwrite the included value.

The following query parameter edits are currently available:

Query parameter name	Description	Options	Default
format	Sets the output to the provided format	jpg, jpeg, heic, png, raw, tiff, webp, gif, avif	None
fit	The method that should be used when resizing	cover, contain, fill, inside, outside	cover
width	The width in pixels, the image should be resized to.	Positive integer or 0	None
height	The height in pixels, the image should be resized to.	Positive integer or 0	None
rotate	The number of degrees the image should be rotated	0-359 or blank (for null)	None
flip	Mirror the image vertically	True/False	False

Query parameter name	Description	Options	Default
flop	Mirror the image horizontally	True/False	False
greyscale	Convert to 8-bit greyscale	True/False	False

Use supported Thumbor filters

This solution supports the Thumbor filters listed in this section, using API calls. To retrieve your API endpoint for the solution, refer Use the solution with a frontend application for instructions.

To use the filters, use the following example syntax for the API call:

```
https://<ApiEndpoint>/<modification>/<image.jpeg>
```

Define the source bucket for the request

To define the bucket used when getting the image for a request, include **s3:BucketName** as a modification in your request. For example, if your source buckets were "`test-bucket-1, the-other-test-bucket`", to indicate that `the-other-test-bucket` should be used when processing an image, enter the following:

```
https://<ApiEndpoint>/s3:the-other-test-bucket/<image.jpeg>
```

Note

Using the **s3:BucketName** tag requires that the bucket chosen is part of the **SourceBuckets** provided upon deployment. For information on how to change the **SourceBuckets** after deployment, see Backward compatibility.

Resize an image

To resize an image, specify `fit-in` and the desired image size. For example, to resize a JPEG image to 300 pixels wide and 400 pixels tall, enter the following:

```
https://<ApiEndpoint>/fit-in/<300x400>/<image.jpeg>
```

Use filters

To use filters, specify a filter from the following table. For example, to blur a JPEG image, enter the following:

```
https://<ApiEndpoint>/filters:blur(7)/<image.jpeg>
```

Note

Some Thumbor filters aren't supported in the current version of this solution. This might affect legacy users with advanced image request configurations. For notes about Thumbor compatibility and source image storage limitations, see Backward compatibility. For examples of filter usage, refer to the Thumbor documentation.

Filter name	Filter syntax
Animated	`/filters:animated(true/false)/`
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)/`

Filter name	Filter syntax
Crop	`/10x10:100x100/`
Equalize	`/filters:equalize()/`
Grayscale	`/filters:grayscale()/`
Image format `(.gif, .jpeg, .png, .avif, .webp, .ti`	`/filters:format(image_format)`
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)/`
Smart Crop	`/filters:smart_crop(faceIndex, facePadding)/`
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]])`

Use multiple filters

To use multiple filters on an image, list them in the same section of the URL. Filters process the image in the order that you specify them. For example:

```
https://<api-endpoint>/fit-in/<300x400>/filters:<fill>(<00ff00>)/
filters:<rotate>(<90>)/<image.jpeg>
```

Custom image requests

Note

As of recent releases of Dynamic Image Transformation for Amazon CloudFront, editing the environment variables directly is not supported for the AutoWebP (v7.0.0), SourceBuckets (v6.2.6), OriginShieldRegion(v7.0.0) and EnableS3ObjectLambda(v7.0.0) template parameters, instead, follow the instructions in Updating template parameters.

You can customize most settings for this solution by editing and updating the environment variables associated with the image handler Lambda function. You can find the image handler function in the AWS Management Console using one of the following methods:

Using the AWS Lambda console:

1. Sign in to the AWS Lambda console.
2. Select **Functions**. The image handler function is listed with the following naming convention: `[.replaceable]<StackName>`-ImageHandlerFunction-`[.replaceable]<UniqueID>`.

Using the AWS CloudFormation console:

1. Sign in to the AWS CloudFormation console.
2. On the **Stacks** page, select this solution's installation stack.
3. Choose the **Resources** tab. The image handler function is listed with a **Logical ID** of `ImageHandlerFunction`.

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

Note: The solution uses the template parameter inputs to determine these initial key values, except for **REWRITE_MATCH_PATTERN** and **REWRITE_SUBSTITUTION**.

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 (Ex: `https://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 S3 bucket which contains the default fallback image. **Note:** This value is ignored if the **ENABLE_DEFAULT_FAL**

Variable Key	Value Type	Description
		LBACK_IMAGE parameter is set to No.
DEFAULT_FALLBACK_IMAGE_KEY	`String`	Defines the default fallback image S3 object key, including the 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. If you overwrite this default value, use a JavaScript-compatible regular expression for matching custom image requests using the rewrite function. This value should match the JavaScript compatible regular expression. For example, `/(filters-)/gm` .
REWRITE_SUBSTITUTION	`String`	By default, this parameter is empty. If you overwrite this default value, use a substitution string for custom image requests using the rewrite function. For example, `filters:`.

Variable Key	Value Type	Description
SECRETS_MANAGER	`String`	Defines the Secrets Manager secret that contains the secret key for the image URL signature. **Note:** This value is ignored if `[.replaceable]ENABLE_SIGNATURE` is set to No.
SECRET_KEY	`String`	Defines the 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.
SOURCE_BUCKETS	`String`	The S3 bucket (or buckets) in your account that contain(s) the original images. If you're providing multiple buckets, separate them by commas.

Updating template parameters

Use the following instructions to update template parameters in such a way that there is no drift between your resources and your CloudFormation stack, and to ensure that all necessary changes are made to your resources:

1. Sign in to the AWS CloudFormation console, select your existing Dynamic Image Transformation for Amazon CloudFront CloudFormation stack, and select Update.
2. Leaving Use Current Template selected, click Next
3. Modify the template parameters as needed

4. Continue through the rest of the workflow as you would when creating the stack.

Note

Modifications made to SIH which are not reflected in the CloudFormation template may be removed when updating template parameters in this fashion

Use the rewrite feature

You can use this solution's rewrite feature to migrate your current image request model to the Dynamic Image Transformation for Amazon CloudFront solution, without changing the applications to accommodate new image URLs.

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's then processed as a Thumbor image request and edits are mapped to the new sharp image library.

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. If you overwrite this default value, use a JavaScript-compatible regular expression for matching custom image requests using the rewrite function. This value should match the JavaScript compatible regular expression. For example, `/(filters-)/gm` .

Variable Key	Value Type	Description
REWRITE_SUBSTITUTION	`String`	By default, this parameter is empty. If you overwrite this default value, use a substitution string for custom image requests using the rewrite function. For example, `filters:`.

You can use any of the Thumbor-supported filters listed in this section with the rewrite feature. The following sections provide examples.

Replace filters- with filters:

If you put `/(filters-)/gm in REWRITE_MATCH_PATTERN` and `filters:` in **REWRITE_SUBSTITUTION**, you can call

```
https://<your-CloudFront-distribution>/filters:rotate(90)/<your-image>
```

instead of

```
https://<your-CloudFront-distribution>/filters-rotate(90)/<your-image>
```

to rotate your image. In this example, the solution replaces `filters-` (filters hyphen syntax) with `filters:` (filters colon syntax).

Reverse path order

You can place filters at the end of the path rather than before the image key.

1. Use the `REWRITE_MATCH_PATTERN` with a regular expression that parses the path into two groups. The solution then uses `REWRITE_SUBSTITUTION` to switch the order of the groups.
2. Use a regular expression specified by `REWRITE_MATCH_PATTERN` to parse the path into groups for a request like `https://abcd.cloudfront.net/imagekey.png/fit-in/200x200` , where the image key appears before the filters. For example:

```
REWRITE_MATCH_PATTERN = /^\/(.*?\..*?)\/(.+)$/gm
```

3. Reverse the order of the fields with `REWRITE_SUBSTITUTION` to convert the request into a Thumbor style request like `https://abcd.cloudfront.net/fit-in/200x200/imagekey.png` , where the image key is moved to the end of the request. For example:

```
REWRITE_SUBSTITUTION = /$2/$1
```

Parse request type

Refer to image-request.spec.js.file, in the Dynamic Image Transformation for Amazon CloudFront GitHub repository.

Rotate images manually

Not all browsers support rotational EXIF data for all image formats and you may notice visual issues when viewing your images through the browser. This tends to be more common with the WebP image format. Sharp allows the passing of a null value in the rotate field to indicate that the orientation associated with the EXIF orientation tag should be manually applied (and the tag removed). You can implement this for base64 image requests through the inclusion of a `rotate: null` edit, or for Thumbor-style requests by including `filters:strip_exif()` or `filters:strip_icc()` in your request path.

Developer guide

This section provides the source code and an API reference for the solution.

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 AWS CDK generates the Dynamic Image Transformation for Amazon CloudFront templates. See the README.md file for additional information.

API reference

This uses the sharp Node.js library to provide high-speed image processing. Open the library, then select **API** from the navigation menu to view the API guides.

Reference

This section includes information about an optional feature for collecting unique metrics for this solution, data surrounding response times of a deployment using the S3 Object Lambda architecture, pointers to related resources, and a list of builders who contributed to this solution.

Anonymized data collection

This solution includes an option to send anonymized operational metrics to AWS. We use this data to better understand how customers use this solution and related services and products. When invoked, the following information is collected and sent to AWS:

- **Solution ID** - The AWS solution identifier
- **Version** - The Dynamic Image Transformation for Amazon CloudFront solution version
- **Unique ID (UUID)** - Randomly generated, unique identifier
- **Timestamp** - The timestamp when the solution's Lambda function runs
- **Region** - The AWS Region the solution is being deployed in
- **CorsEnable** - Whether CORS is activated
- **NumberOfSourceBuckets** - Number of source buckets
- **DeployDemoUi** - Whether the Demo UI deployment is activated
- **LogRetentionPeriod** - The log retention period
- **AutoWebP** - Whether AutoWebP is activated
- **EnableSignature** - Whether the image URL signature is activated
- **EnableDefaultFallbackImage** - Whether the default fallback image is activated
- **UseExistingCloudFrontDistribution** - Whether the deployment uses a natively created, or an existing CloudFront distribution
- **EnableS3ObjectLambda** - Whether the S3 Object Lambda architecture is being used
- **OriginShieldRegion** - What region Origin Shield is enabled in (or if disabled)
- **AWS/Lambda/Invocations** - Quantity of image handler Lambda function invocations
- **AWS/CloudFront/Requests** - Quantity of requests hitting the image handler distribution
- **AWS/CloudFront/BytesDownloaded** - Quantity of bytes downloaded from the image handler distribution

- **AWSLambdaBilledDuration** - Sum of billed duration for image handler Lambda function
- **AWSLambdaMemorySize** - Memory size of image handler Lambda function
- **DefaultRequestsCount** - The count of requests which use the default base64 request encoding
- **ThumborRequestsCount** - The count of requests which use Thumbor-style request encoding
- **CustomRequestsCount** - The count of requests which use a custom request encoding
- **QueryParamRequestsCount** - The count of requests which use query parameter based image edits
- **ExpiresRequestsCount** - The count of requests which use the expires query parameter

AWS owns the data gathered through this survey. Data collection is subject to the AWS Privacy Notice. To opt out of this feature, complete the following steps before launching the CloudFormation template.

1. Download the `dynamic-image-transformation-for-amazon-cloudfront.template` AWS CloudFormation template to your local hard drive.
2. Open the CloudFormation template with a text editor.
3. Modify the AWS CloudFormation template mapping section from:

```
Solution:
    Config:
        AnonymousUsage: "Yes",
        DeployCloudWatchDashboard: …,
        SharpSizeLimit: …,
        SolutionId: …,
        Version: …
```

   to:

```
Solution:
    Config:
        AnonymousUsage: No,
        DeployCloudWatchDashboard: …,
        SharpSizeLimit: …,
        SolutionId: …,
        Version: …
```

4. Sign in to the AWS CloudFormation console.

5. Select **Create stack**.
6. On the **Create stack** page, **Specify template** section, select **Upload a template file**.
7. Under **Upload a template file**, choose **Choose file** and select the edited template from your local drive.
8. Choose **Next** and follow the steps in Deployment process overview to launch the solution.

S3 Object Lambda architecture response latency

Average response latency for uncached images on a deployment using the S3 Object Lambda architecture. As the processing is identical between both architectures, processing durations should be similar to those when using the API Gateway architecture, this dataset can serve as a reference for determining the maximum size of an image that can be returned by the solution, especially for sizes which are not supported by the API Gateway architecture. In cases where Timeouts are encountered, increasing the memory allocation of the BackendImageHandler Lambda function may lead to performance enhancements.

Image Size/Filters	No filters	Grayscale	Equalize	Format (PNG to JPG)
1 MB	0.35s	0.65s	0.95s	0.41s
10 MB	0.78s	2.95s	4.64s	1.30s
50 MB	2.28s	10.96s	22.55s	4.65s
100 MB	6.99s	25.74s	Timeout (>30s)	11.56s

Related resources

Refer to the sharp Node.js library for more information about `sharp`.

Contributors

- Simon Krol
- Kamyar Ziabari

- Ryan Hayes
- Beomseok Lee
- George Lenz
- Dmitry Fisenko
- Doug Toppin
- Garvit Singh

Revisions

Publication date: *June 2017*.

Check the CHANGELOG.md file in the GitHub repository to see all notable changes and updates to the software. The changelog provides a clear record of improvements and fixes for each version.

Notices