



## Implementation Guide

# Dynamic Image Transformation for Amazon CloudFront



# Dynamic Image Transformation for Amazon CloudFront: Implementation Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

---

# Table of Contents

<b>Solution overview</b> .....	<b>1</b>
Features and benefits .....	2
Use cases .....	3
Architecture options .....	4
Concepts and definitions .....	4
<b>Architecture overview</b> .....	<b>6</b>
Lambda Architecture .....	6
ECS Architecture .....	8
Component descriptions .....	10
<b>Architecture details</b> .....	<b>12</b>
Lambda architecture details .....	12
Core components .....	12
Request processing workflow .....	12
Lambda architecture features .....	13
AWS services used for Lambda architecture .....	16
ECS architecture details .....	18
Key concepts .....	18
Container-based processing .....	20
Request processing workflow .....	20
Caching strategy .....	21
Origin Override Header .....	22
Logging and retention .....	23
Constraints, dependencies, and assumptions .....	23
AWS services used in ECS architecture .....	24
<b>Plan your deployment</b> .....	<b>27</b>
Choosing your deployment architecture .....	27
Lambda Architecture .....	27
ECS Architecture .....	27
Decision matrix and when to choose each .....	28
Supported AWS Regions .....	29
Opt-in Regions .....	30
Cost .....	30
Lambda architecture costs .....	31
ECS architecture costs .....	32

---

Cost considerations .....	33
Security .....	34
Demo UI .....	34
IAM roles .....	34
Amazon API Gateway .....	35
Amazon CloudFront .....	35
Amazon WAF .....	35
Quotas .....	35
AWS CloudFormation quotas .....	36
AWS Lambda quotas .....	36
Amazon API Gateway quotas .....	36
Optional Configurations .....	36
Operational Dashboard .....	37
Sharp Size Limit .....	37
Amazon CloudWatch Logs and Alarms .....	38
<b>Deploy the solution .....</b>	<b>39</b>
Deployment process overview .....	39
Deploy using AWS Launch Wizard .....	40
Deploy using AWS CloudFormation .....	40
AWS CloudFormation templates .....	40
Deploy Lambda architecture .....	42
Deploy ECS architecture .....	48
Attaching an existing CloudFront distribution .....	50
<b>Update the solution .....</b>	<b>53</b>
Update using AWS Launch Wizard .....	53
Update using AWS CloudFormation .....	53
Update Lambda architecture .....	54
Update ECS architecture .....	56
Backward compatibility .....	57
<b>Troubleshooting .....</b>	<b>60</b>
Contact AWS Support .....	60
Create case .....	60
How can we help? .....	60
Additional information .....	60
Help us resolve your case faster .....	61
Solve now or contact us .....	61

Transformation policy errors .....	61
Policy validation failures .....	61
Policy not being applied .....	61
Conditional transformation issues .....	62
Seeing cached images even after making configuration changes on Admin UI .....	62
Enable debug logs .....	63
Tracing API errors using request ID .....	64
<b>Uninstall the solution .....</b>	<b>65</b>
Using the AWS Management Console .....	65
AWS CloudFormation .....	65
AWS Launch Wizard .....	65
Using AWS Command Line Interface .....	65
Deleting the Amazon S3 buckets .....	66
<b>Use the solution .....</b>	<b>67</b>
Lambda architecture .....	67
Use the demo UI .....	67
Use the solution with a frontend application .....	68
Create and use image requests .....	69
Dynamically resize photos .....	70
Edit images .....	71
Use smart cropping .....	72
Use round cropping .....	73
Overlay an image .....	74
Overwrite animated status .....	76
Activate and use content moderation .....	77
Include custom response headers .....	78
Include request expiration .....	79
Use supported Query Parameter edits .....	79
Use supported Thumbor filters .....	81
Define the source bucket for the request .....	81
Resize an image .....	81
Use filters .....	82
Use multiple filters .....	83
Custom image requests .....	84
Updating template parameters .....	87
Use the rewrite feature .....	88

---

Replace filters- with filters: .....	89
Reverse path order .....	89
Parse request type .....	90
Rotate images manually .....	90
ECS architecture .....	90
Deploy the template .....	90
Access the Admin UI .....	90
Create origins .....	91
Create transformation policies .....	91
Transformation filter reference .....	94
Create mappings .....	96
Test image transformations .....	97
Monitor and manage .....	97
<b>Developer guide .....</b>	<b>98</b>
Source code .....	98
API reference .....	98
Authentication .....	98
Transformation Policies API .....	98
Origins API .....	100
Mappings API .....	101
Error Responses .....	103
<b>Reference .....</b>	<b>104</b>
Data collection .....	104
Related resources .....	104
Contributors .....	104
<b>Revisions .....</b>	<b>105</b>
<b>Notices .....</b>	<b>106</b>
Configuration Delay and Cached Images .....	106

# Solution Overview

Dynamic Image Transformation for Amazon CloudFront (formerly known as Serverless Image Handler) enables real-time image processing and optimization, eliminating the need for pre-processing images or maintaining multiple versions of the same image. Dynamic Image Transformation (DIT) empowers developers with the flexibility to transform images on-demand for specific use cases, enabling responsive design. DIT can automatically optimize images for each user's device, delivering the highest quality images at the smallest possible file size. This not only reduces operational costs through efficient storage and bandwidth usage but also enhances the end-user experience through faster load times and improved visual quality.

Users can specify image transformations either through URL query parameters or predefined transformation policies. DIT also includes advanced features such as request signing for enhanced security, smart cropping, and content moderation powered by Amazon Rekognition.

This implementation guide walks you through the Dynamic Image Transformation for Amazon CloudFront solution, including its architecture, components, deployment considerations, and configuration steps for AWS Cloud implementation.

The guide is designed for solution architects, business decision makers, DevOps engineers, data scientists, and cloud professionals who want to implement DIT in their environment.

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

If you want to . . .	Read . . .
Know the cost for running this solution.	<a href="#">Cost</a>
Understand the security considerations for this solution.	<a href="#">Security</a>
Know how to plan for quotas for this solution.	<a href="#">Quotas</a>
Know how to troubleshoot common issues with this solution.	<a href="#">Troubleshooting</a>
Know which AWS Regions support this solution.	<a href="#">Supported AWS Regions</a>

If you want to . . .	Read . . .
View or download the AWS CloudFormation template included in this solution to automatically deploy the infrastructure resources (the "stack") for this solution.	<a href="#">AWS CloudFormation template</a>
Access the source code and optionally use the AWS Cloud Development Kit (AWS CDK) to deploy the solution.	<a href="#">GitHub repository</a>

## Features and benefits

### Automatic image optimization

Automatically modify images based on users' devices and screen sizes using intelligent optimization capabilities that determine optimal image quality and format based on client viewport size, device pixel ratio, and user agent information.

### 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 and intelligent focal point detection.

### Low-cost image storage

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

**Version V8.0.0 introduces significant enhancements to meet enterprise requirements alongside the core solution capabilities:**

### Multi-origin support

Source images from Amazon S3 buckets, external domains, or any HTTP-accessible image source. Configure path-based and host-header mappings to route requests to appropriate origins without modifying application code.

## Transformation policies

Create reusable transformation configurations that can be applied consistently across your applications. Policies support conditional logic based on request headers and device characteristics, enabling consistent image processing across your applications.

## Dual architecture options

Choose between two deployment architectures based on your specific requirements:

- Lambda architecture for cost-optimized deployments with images up to 6MB
- ECS architecture for high-performance deployments supporting images greater than 6MB

## Management console

Manage origins, transformation policies, and mappings through a web-based administrative interface, simplifying configuration management.

# Use cases

## Accelerate page loads and optimize visual quality

Deliver faster-loading, high-quality images that automatically adjust for each user's device capabilities and screen size. Use automatic optimization to reduce image load times while maintaining visual quality, improving your end-user experience.

## Enable seamless CDN migration

Migrate from other CDN providers with minimal application changes. Use transformation policies and origin mappings to replicate existing image transformation workflows while leveraging AWS's global infrastructure.

## Reduce operational overhead

Eliminate the need to pre-generate and store multiple image variants. Transform images on-demand and cache results globally through CloudFront, reducing storage costs and simplifying content management.

## Support enterprise-scale deployments

Process large images up to 100 MB using the ECS architecture option, supporting enterprise content management systems and high-resolution image workflows.

### **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.

## **Architecture options**

This solution offers two distinct deployment architectures to meet different performance and cost requirements:

### **Lambda Architecture**

A cost-optimized serverless architecture suitable for most image transformation workloads. This architecture supports transformed images up to 6 MB in size with a pay-per-request pricing model that eliminates idle costs. It provides automatic scaling with built-in high availability, making it ideal for applications with moderate image processing requirements.

### **ECS Architecture**

A high-performance container-based architecture for demanding workloads. This architecture supports images up to 100 MB in size and includes all new v8.0.0 features such as transformation policies, non-S3 origins, and the Admin UI. It offers configurable resource allocation with t-shirt sizing (S, M, L, XL) and is optimized for high-throughput and large image processing.

Both architectures leverage Amazon CloudFront for global content delivery and caching, ensuring optimal performance for end users regardless of their geographic location.

## **Concepts and definitions**

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

### **origin**

The source location where original images are stored, which can be an Amazon S3 bucket or an external HTTP-accessible domain.

### **transformation policy**

A reusable configuration that defines a set of image transformations, output settings, and conditional logic that can be applied to image requests.

**mapping**

Configuration that routes image requests to specific origins based on request path patterns or host headers.

**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 offers two deployment architectures to meet different performance and cost requirements. The Lambda architecture provides cost-optimized serverless processing for images up to 6 MB. The ECS architecture delivers high-performance processing for images up to 100 MB and includes advanced features like transformation policies, non-S3 origin support, and an administrative interface.

This solution provides two architecture options, each optimized for different use cases and performance requirements.

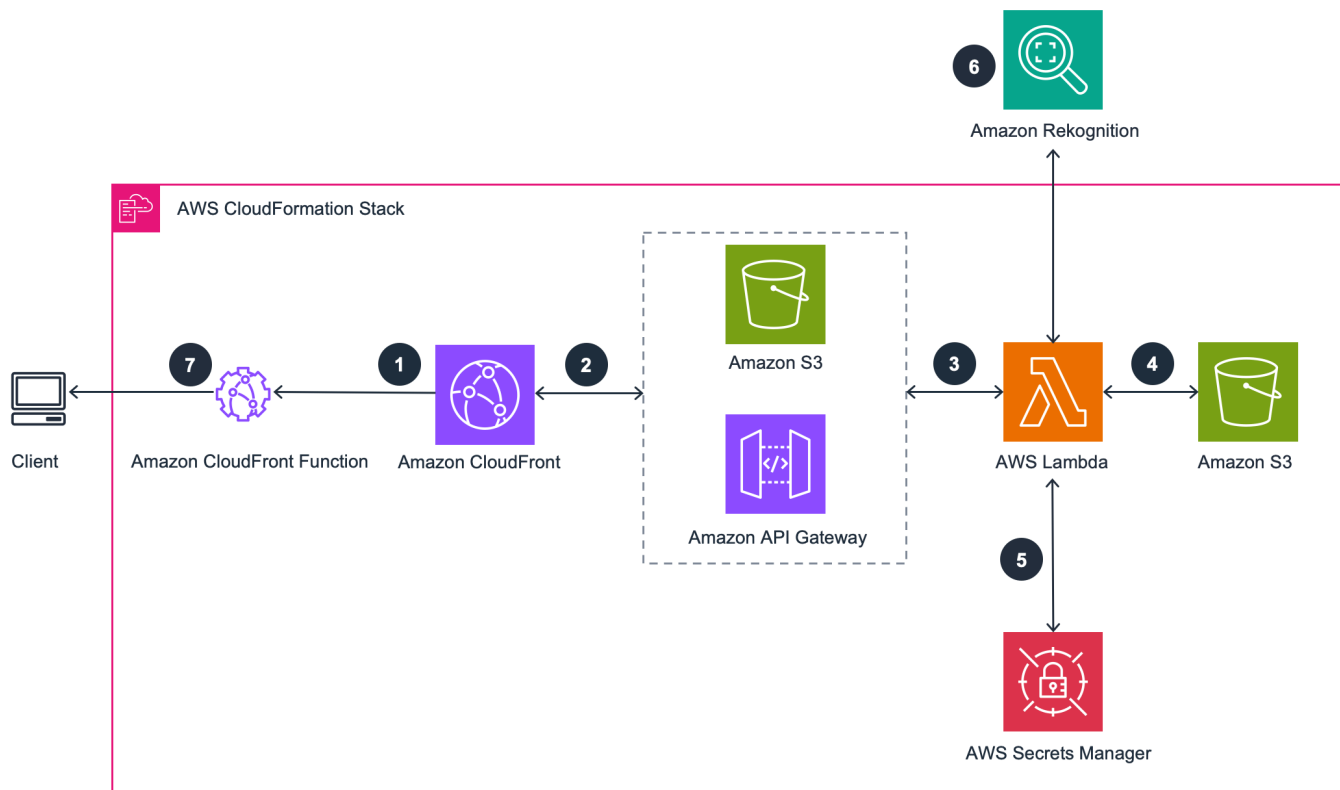
## Lambda Architecture

This cost-optimized serverless architecture is suitable for most image transformation workloads with images up to 6 MB.

### 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](#).

### Lambda architecture for cost-optimized image processing



### Note

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

The high-level process flow for the Lambda architecture is as follows:

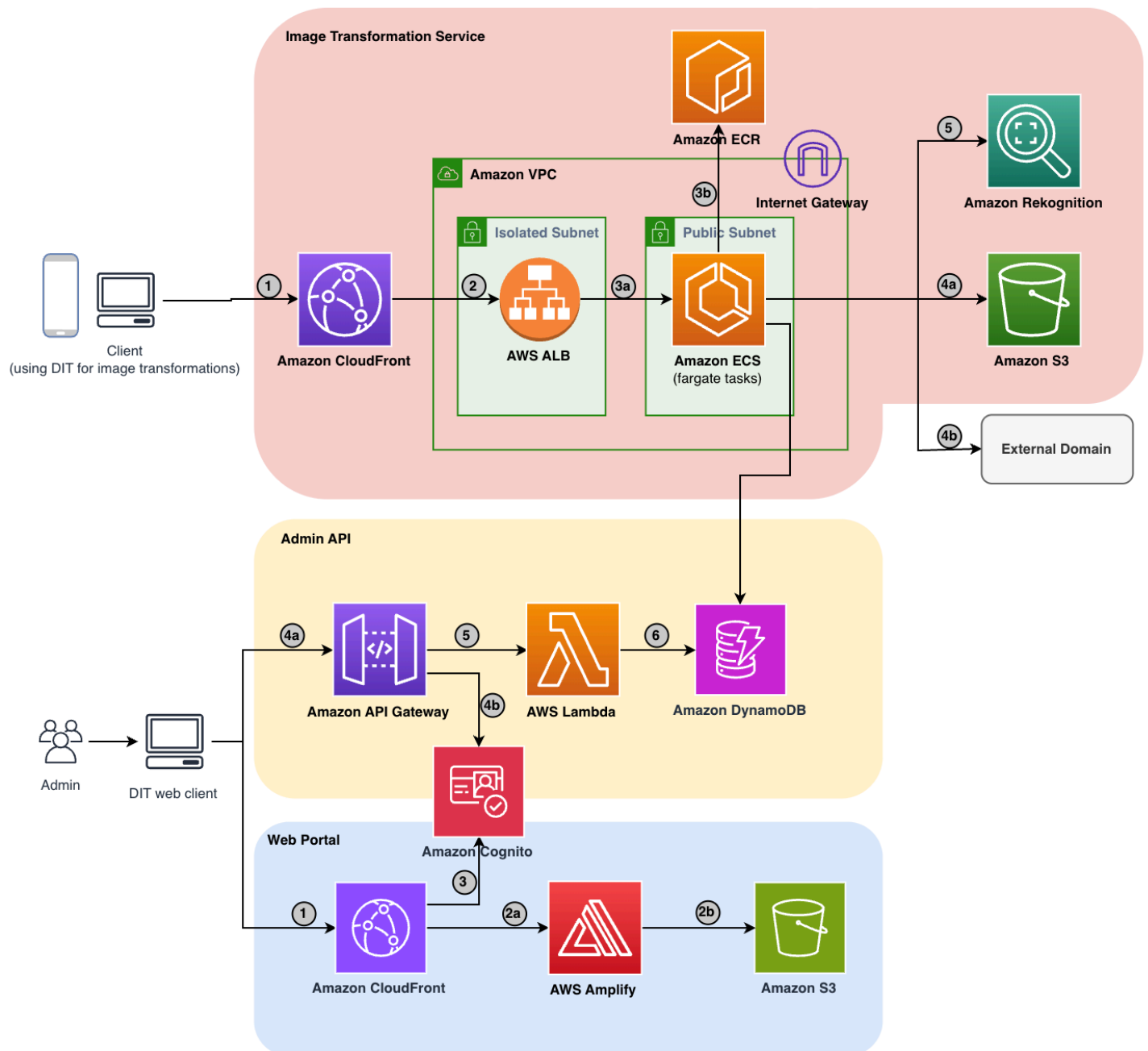
1. An [Amazon CloudFront](#) distribution provides a caching layer to reduce the cost of image processing and the latency of subsequent image delivery.
2. [Amazon API Gateway](#) provides endpoint resources and initiates the [AWS Lambda](#) function.
3. A Lambda function retrieves the image from a customer's existing [Amazon S3](#) bucket and uses sharp to return a modified version of the image to the API Gateway.
4. A solution-created S3 bucket provides log storage, separate from your customer-created S3 bucket for storing images.
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.

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 to normalize headers and query parameters for improved cache hit rates.

## ECS Architecture

This high-performance container-based architecture supports images up to 100 MB and includes all advanced v8.0.0 features including transformation policies, non-S3 origin support, and administrative interface.

### ECS architecture for high-performance image processing



The high-level process flow for the ECS architecture is as follows:

1. An Amazon CloudFront distribution provides global caching and content delivery.
2. An [Application Load Balancer](#) (ALB) distributes incoming requests across multiple ECS tasks for high availability and scalability.
3. [Amazon Elastic Container Service](#) (ECS) tasks running on [AWS Fargate](#) process image transformation requests using containerized applications.

4. ECS tasks maintain in-memory caches of transformation policies and origin mappings for fast request resolution and reduced latency.
5. Images are retrieved from multiple origin types: Amazon S3 buckets or external HTTP-accessible domains based on configured origin mappings.
6. An administrative interface built with [AWS Amplify](#) provides policy and origin management capabilities through a secure web interface.
7. [Amazon DynamoDB](#) stores transformation policies, origin configurations, and mapping rules with high availability and performance.
8. [Amazon Cognito](#) provides authentication and authorization for the administrative interface.
9. (Optional) Amazon Rekognition integration for smart cropping and content moderation features.

## Component descriptions

### Application Load Balancer (ALB)

Distributes incoming image transformation requests across multiple ECS tasks, providing high availability and automatic scaling capabilities. The ALB performs health checks on ECS tasks and routes traffic only to healthy instances.

### Amazon ECS with AWS Fargate

Runs containerized image processing applications without managing servers. ECS tasks automatically scale based on demand and maintain in-memory caches for optimal performance. Fargate provides serverless compute for containers, eliminating the need to provision and manage EC2 instances.

### Administrative Interface

A web-based management console built with AWS Amplify that allows administrators to configure and manage image origins (both S3 and external sources), create and edit transformation policies, set up origin mappings using path-based and host-header routing, monitor system performance and usage metrics, and test configurations before deployment to ensure proper functionality.

## **Amazon DynamoDB**

Stores configuration data including transformation policies, origin definitions, and mapping rules. DynamoDB provides fast, predictable performance with seamless scalability for configuration lookups.

## **Amazon Cognito**

Provides secure authentication and authorization for the administrative interface, supporting user management, multi-factor authentication, and role-based access control.

# 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.

## Lambda architecture details

The Lambda architecture provides cost-optimized serverless image processing for images up to 6 MB with pay-per-request pricing and automatic scaling.

### Core components

**Amazon API Gateway** - RESTful API endpoint for image transformation requests - Request validation and parameter parsing - Integration with AWS Lambda for processing - Built-in throttling and caching capabilities

**AWS Lambda** - Serverless compute for image processing using Sharp library - Automatic scaling based on request volume - 6 MB payload limit for requests and responses - Memory allocation from 128 MB to 10,240 MB

**Amazon CloudFront** - Global content delivery network for caching processed images - Edge locations reduce latency for end users - Custom cache behaviors for different request types - Origin Shield for additional caching layer

**Amazon S3** - Source image storage with versioning and encryption support - Processed image caching (optional) - Static website hosting for Demo UI - Cross-region replication capabilities

## Request processing workflow

### Lambda processing steps:

1. API Gateway receives image transformation request
2. Request parameters are validated and parsed
3. Lambda function is invoked with transformation parameters
4. Original image is fetched from S3 bucket
5. Sharp library applies transformations (resize, crop, format conversion, etc.)
6. Processed image is returned through API Gateway

## 7. CloudFront caches the processed image for future requests

**Supported transformations:** - Resize with various fit options (cover, contain, fill, inside, outside) - Crop with precise coordinates or smart cropping using Amazon Rekognition - Format conversion ("jpg", "jpeg", "heic", "png", "raw", "tiff", "webp", "gif", "avif") - Quality optimization and compression - Rotation and flip operations - Watermarking and overlay effects - Content moderation using Amazon Rekognition

## Lambda architecture features

### 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:

**ℹ 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>
```

**ℹ 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:

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

**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](#).

## AWS services used for Lambda architecture

AWS service	Description
<a href="#">Amazon CloudFront</a>	<b>Core.</b> Provides a caching layer to reduce latency and the cost of image processing for subsequent identical requests. Allows pre/post processing of requests and responses.

AWS service	Description
<a href="#">AWS Lambda</a>	<b>Core.</b> Runs functions to retrieve, modify, and invoke other services to analyze images. Also runs a function to support URL signature validation.
<a href="#">Amazon S3</a>	<b>Core.</b> Stores images, logs, and a demo UI.
<a href="#">Amazon API Gateway</a>	<b>Supporting.</b> Provides API endpoints to invoke Lambda functions. Only used if EnableS3ObjectLambda is set to "No".
<a href="#">Amazon S3 Object Lambda</a>	<p>This option has been deprecated. Amazon S3 Object Lambda will no longer be open to new customers starting on November 7, 2025. If you were not an existing user of S3 Object Lambda before November 7, 2025, select 'No'. For more information, please visit <a href="https://docs.aws.amazon.com/AmazonS3/latest/userguide/amazons3-ol-change.html">https://docs.aws.amazon.com/AmazonS3/latest/userguide/amazons3-ol-change.html</a>.</p> <p><b>Supporting.</b> Provide S3 Origin to invoke Lambda functions. Only used if EnableS3ObjectLambda is set to "Yes".</p>
<a href="#">AWS CDK</a>	<b>Supporting.</b> Provides infrastructure as code constructs to generate the solution's underlying CloudFormation templates.
<a href="#">AWS CloudFormation</a>	<b>Supporting.</b> Deploys the solution's underlying AWS resources.
<a href="#">AWS Identity and Access Management (IAM)</a>	<b>Supporting.</b> Allows for fine-grained access permissions.
<a href="#">Amazon Rekognition</a>	<b>Optional.</b> Uses machine learning (ML) to analyze images.

AWS service	Description
<a href="#">AWS Secrets Manager</a>	<b>Optional.</b> Manages secrets to support URL signatures.

## ECS architecture details

The ECS architecture provides high-performance image processing capabilities for demanding workloads and large images up to 100 MB.

### Key concepts

#### Origins

An origin defines the source location where original images are stored. The solution supports two types of origins:

**Amazon S3 origins** - S3 buckets containing your original images - Configured with valid S3 endpoint URL and optional path prefix - Supports all S3 features including versioning and encryption

**External origins** - HTTP-accessible domains hosting images - Content management systems, CDNs, or web servers - Must use HTTPS with valid SSL certificates from trusted certificate authorities - Self-signed certificates are not supported

Origins are configured with the following properties:

- **Origin name:** A descriptive identifier for management purposes
- **Origin domain:** A valid S3 endpoint URL or external domain (e.g., `images.example.com`)
- **Origin path** (optional): A path prefix appended to all requests (e.g., `/assets/images`)
- **Origin headers** (optional): Custom headers sent with each origin request for authentication or routing

### Transformation policies

Transformation policies are reusable configurations that define sets of image transformations, output settings, and conditional logic. They enable consistent image processing across applications without requiring transformation parameters in each request.

## Policy structure:

- **Transformations:** Array of image operations (resize, crop, format conversion, etc.) with optional conditional logic based on request headers
- **Output settings:** Quality, format, and optimization configurations
- **Default policy:** Only one policy can be designated as default (applied when no specific policy is requested)

## Example policy capabilities:

- Automatic format selection (WebP for supported browsers, JPEG for others)
- Quality optimization based on device pixel ratio

## Policy application precedence:

1. Explicit transformations in request query parameters (highest)
2. Named policy specified in request
3. Policy configured in the mapping
4. Default policy (lowest)

## Mappings

Mappings define how incoming image requests are routed to specific origins. The solution supports two types of mappings with different precedence rules:

### Host-header mappings (highest precedence)

- Route requests based on the Host header in the incoming request
- Support exact matches (`images.mysite.com`) and wildcard patterns (`*.mysite.com`)
- Useful for multi-tenant applications or domain-based routing
- Example: `images.customer1.com` → Customer 1's S3 bucket

### Path-based mappings (lower precedence)

- Route requests based on URL path patterns
- Support exact paths (`/thumbnails`) and wildcard patterns (`/customer-123/*`)

- Enable content-based routing within a single domain
- Example: `/product-images/*` → Product catalog origin

### Mapping precedence rules:

1. Host-header mappings are evaluated first
2. If no host-header mapping matches, path-based mappings are evaluated
3. Most specific pattern takes precedence over wildcards
4. If no mapping matches, the request returns a 404 error

## Container-based processing

### ECS tasks with AWS Fargate

Containerized image processing applications running on AWS Fargate with automatic scaling based on CPU utilization and request volume. The architecture offers T-shirt sizing options (S, M, L, XL) for different performance requirements while requiring no server management as Fargate handles infrastructure provisioning.

### Application Load Balancer

Distributes requests across multiple ECS tasks for high availability with health checks ensuring traffic routes only to healthy container instances. The load balancer provides support for connection pooling and keep-alive connections, along with SSL termination and HTTP/2 support for optimal performance.

### Scaling configuration

The minimum task count ensures baseline capacity while the maximum task count prevents runaway scaling costs. Target CPU utilization triggers scale-out events, and scale-in cooldown periods prevent thrashing to maintain stable performance.

## Request processing workflow

### Request resolution

1. Incoming requests are analyzed to determine origin mapping
2. Host-header mappings take precedence over path-based mappings

3. Request resolver identifies the appropriate origin and transformation policy
4. Headers and authentication parameters are prepared for origin requests

### **Transformation resolution**

1. Policy resolver retrieves transformation policies from in-memory cache
2. Conditional logic evaluates client hints and request headers
3. Explicit query parameters override policy-defined transformations
4. Final transformation parameters are validated and normalized

### **Image processing**

1. Original image is fetched from the resolved origin (S3 or external)
2. Sharp library applies transformations in the specified order
3. Output format and quality are optimized based on client capabilities
4. Processed image is returned through the ALB to CloudFront

## **Caching strategy**

The ECS architecture implements a multi-layer caching strategy to optimize performance and reduce latency.

### **In-memory cache**

- Each ECS task maintains local caches for transformation policies and origin mappings
- Cache is populated during task startup from DynamoDB
- Fast lookup times for frequently accessed configurations

### **Cache initialization**

1. New ECS tasks query DynamoDB during startup
2. All transformation policies are loaded into memory
3. Origin mappings are stored in optimized data structures

### **Cache invalidation**

- Rolling deployment strategy updates all ECS tasks with new configurations
- Eventual consistency model balances performance with data freshness
- Cache refresh typically takes up to 5 minutes

### Cache consistency

- New ECS tasks always load the latest configuration data
- Existing tasks maintain cached data until manual refresh or natural replacement
- Auto-scaling events automatically provision tasks with current data
- Stale cache scenarios are resolved through the rolling update process

## Origin Override Header

The capability is meant for advanced users. When you implement the changes described here, mapping lookup will be bypassed in favor of the origin override header when routing requests.

Image transformations can continue to be supplied either through individual query strings in the request or through `?policyId=123456` or by leveraging default policy.

### Warning

These changes can have undesired security implication like sending requests to origin that are not onboarded on DIT. \*\*Consider using the host header mapping feature unless it is not feasible for you to enumerate every host mapping in the DIT config.

To use this capability following steps need to be taken:

1. Update **Origin Override Header** parameter in CloudFormation stack - this identifies the custom header that will be added by CF function and used by the image processing layer to route requests to the targeted origin. All solution specific custom headers are prepended with `dit-` we recommend the same for this header for eg. `dit-origin`.

This change updates the implemented cache policy and ECS environment variable to use the identified custom header.

2. Update CF function in CloudFront console

- a. Use same header that was supplied in the CloudFormation stack parameters:

```
const ditOriginHeader = ""; // eg. dit-origin
```

- b. Update the business logic to populate origin override header value - this could be as simple as using a request header that your application is already sending or it could involve a more "on-the-fly" origin calculation:

```
// Customer-specific origin header mapping (placeholder for extension)
// Customers can extend this logic to set dit-origin based on their routing needs
if (ditOriginHeader) {
  // Example: Set based on host or custom logic
  // request.headers[ditOriginHeader] = { "value": "custom-origin-value" };
}
```

## Logging and retention

### CloudWatch log retention

The ECS architecture maintains separate log groups with different retention policies:

- **Admin API logs:** 10 years retention for audit and compliance purposes
- **Image processing logs:** 10 years retention for operational monitoring and troubleshooting

Log retention policies are automatically configured during deployment and help balance operational visibility with storage costs.

## Constraints, dependencies, and assumptions

### Constraints

- Lambda architecture: Supports transformation for images up to 6MB
- ECS architecture: Supports transformation for images up to 100MB
- Maximum 100 transformations per policy
- ECS architecture: Auto-scaling based on CPU utilization

### Dependencies

- Sharp Node.js library for image processing
- Thumbor compatibility layer for legacy support

## Assumptions

- Source images in supported formats (JPEG, PNG, WebP, AVIF, TIFF, GIF)
- ECS architecture: Configuration changes may take up to 5 minutes to propagate
- Cached images remain until expiration or manual invalidation
- Lambda architecture: Standard AWS Lambda concurrency limits (1000 default)
- CloudFront caching improves performance for repeated requests

## AWS services used in ECS architecture

AWS service	Description
<b>Compute &amp; Serverless</b>	
<a href="#">AWS Lambda</a>	Image processing functions, API handlers
<a href="#">Amazon Rekognition</a>	Image transformation features (smart cropping, content moderation)
<b>API &amp; Networking</b>	
<a href="#">Amazon API Gateway</a>	REST API for image transformation requests
<a href="#">Amazon CloudFront</a>	CDN for content delivery and caching
<a href="#">Elastic Load Balancing v2</a>	Application load balancing (for ECS architecture)
<b>Storage</b>	
<a href="#">Amazon S3</a>	Source image storage, transformed image cache
<b>Database</b>	

<b>AWS service</b>	<b>Description</b>
<a href="#">Amazon DynamoDB</a>	Storing transformation policies, mappings, and origins
<b>Authentication &amp; Authorization</b>	
<a href="#">Amazon Cognito</a>	User authentication for admin UI
<a href="#">AWS IAM</a>	Roles, policies, and permissions
<b>Monitoring &amp; Logging</b>	
<a href="#">Amazon CloudWatch</a>	Supporting service for metrics, dashboards, alarms, and log management
<b>Container Services (ECS Architecture)</b>	
<a href="#">Amazon ECS</a>	Container orchestration for image processor
<a href="#">Amazon ECR</a>	Container image registry
<a href="#">Amazon EC2</a>	VPC, subnets, security groups for ECS
<b>Automation &amp; Events</b>	
<a href="#">Amazon EventBridge</a>	Event-driven automation
<a href="#">AWS Application Auto Scaling</a>	ECS task auto-scaling
<b>Security &amp; Encryption</b>	
<a href="#">AWS KMS</a>	Encryption key management
<a href="#">AWS Secrets Manager</a>	Manages secrets to support URL signatures
<b>Deployment &amp; Management</b>	
<a href="#">AWS CloudFormation</a>	Infrastructure as code
<a href="#">AWS CDK</a>	Infrastructure definition framework

**⚠ Important****Configuration Delay and Cached Images**

Changes to origins, transformation policies, or mappings may take up to 5 minutes to propagate across all image processing tasks. During this period, some requests may still use the previous configuration. Configuration updates do not automatically refresh cached images. Cached images processed with previous settings will remain available until they expire or are manually invalidated.

# Plan your deployment

This section describes the [cost](#), [security](#), [quotas](#), and other considerations before deploying the solution.

## Choosing your deployment architecture

The Dynamic Image Transformation solution offers two deployment architectures, each optimized for different use cases, performance requirements, and cost considerations.

### Lambda Architecture

This cost-optimized serverless architecture is ideal for most image transformation workloads with moderate performance requirements.

#### Key characteristics:

- **Image size limit:** Up to 6 MB per image
- **Pricing model:** Pay-per-request with no idle costs
- **Scaling:** Automatic scaling with built-in high availability
- **Maintenance:** Fully managed serverless infrastructure
- **Feature set:** Core image transformation capabilities

#### Best suited for:

- Small to medium-sized images (under 6 MB)
- Cost-sensitive deployments
- Variable or unpredictable traffic patterns
- Simple transformation requirements
- S3 only origins

### ECS Architecture

This high-performance container-based architecture supports demanding workloads and provides access to all v8.0.0 features.

**Key characteristics:**

- **Image size limit:** Up to 100 MB per image
- **Pricing model:** Fixed infrastructure costs with usage-based scaling
- **Scaling:** Configurable auto-scaling with t-shirt sizing options
- **Feature set:** Complete v8.0.0 feature set including policies and non-S3 origins
- **Management:** Administrative web interface included

**Best suited for:**

- Large images (6 MB to 100 MB)
- Enterprise deployments requiring advanced features
- Non-S3 origin requirements
- Consistent high-volume traffic
- Complex transformation workflows
- Migration from other CDN providers

**Advanced capabilities:**

- Transformation policies with conditional logic
- Multi-origin support (S3 and external)
- Administrative interface for configuration management
- In-memory caching for optimal performance

**Decision matrix and when to choose each**

Requirement	Lambda Architecture	ECS Architecture
Image size	≤ 6 MB	≤ 100 MB
Cost optimization	Yes - Lowest cost	Higher cost
Transformation policies	No	Yes

Requirement	Lambda Architecture	ECS Architecture
Non-S3 origins	No	Yes
Administrative UI	No	Yes
Auto-scaling	Yes - Built-in	Yes - Configurable

## 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

The cost for running this solution varies between the two deployment architectures. Choose the architecture that best balances your performance requirements with cost considerations.

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.

Dynamic Image Transformation for Amazon CloudFront uses CloudFront's pay-as-you-go pricing model by default. Depending on your expected traffic, you may be able to optimize costs by switching to CloudFront's fixed-pricing model. Before deployment, evaluate your expected workload including monthly data transfer volume and image request count against CloudFront's pricing models to determine which option provides the best value for your specific use case.

To switch to these tiers after deployment, navigate to the CloudFront console, select your CloudFront distribution, under the Billing section click "Switch to a plan" and select one of the available fixed pricing plans.

The pricing estimates below reflect costs when using the fixed-pricing tiers.

For detailed CloudFront pricing information, including pay-as-you-go rates and fixed-pricing tier benefits, visit the [CloudFront Pricing page](https://aws.amazon.com/cloudfront/pricing/).

## Lambda architecture costs

You are responsible for the cost of the AWS services used while running the Lambda architecture. The below table gives the breakdown of costs for different workload sizes.

### Assumptions

- 90% cache hit rate
- Average image size: 45 KB
- 350 ms processing time per image

AWS service	10M images served	125M images served	500M images served
Amazon API Gateway	\$1.00	\$13.00	\$50.00
AWS Lambda	\$0.93	\$12.08	\$46.46
Amazon CloudFront	\$200.00	\$200.00	\$1000.00
Amazon S3	\$0.40	\$5.00	\$20.00
Amazon CloudWatch	\$1.15	\$19.17	\$75.67
<b>Total</b>	<b>\$203.48</b>	<b>\$249.25</b>	<b>\$1192.13</b>

## ECS architecture costs

The ECS architecture has higher baseline costs due to always-running infrastructure but provides better performance and advanced features. Costs scale with the selected t-shirt size and auto-scaling configuration.

### ECS architecture cost summary

Dynamic Image Transformation for Amazon CloudFront uses CloudFront's pay-as-you-go pricing model by default. However, if you expect your monthly usage on your distribution to be below 50TB of data transfer and 500M image requests, you can optimize costs by switching to CloudFront's Business or Premium fixed-pricing tiers after deployment.

To switch to these tiers after deployment, navigate to the CloudFront console, select your CloudFront distribution, under the Billing section click "Switch to a plan" and select one of the available fixed pricing plans.

The pricing estimates below reflect costs when using the fixed-pricing tiers.

For detailed CloudFront pricing information, including pay-as-you-go rates and fixed-pricing tier benefits, visit the [CloudFront Pricing page](https://aws.amazon.com/cloudfront/pricing/).

### ECS architecture cost breakdown

The following table provides sample cost breakdowns for different ECS deployment sizes in the US East (N. Virginia) Region for one month.

#### Assumptions

- 90% cache hit rate
- Average image size: 45 KB
- ALB configuration is Small, Medium and Large for the corresponding workloads
- Actual cost might vary based on burst traffic patterns

AWS Service	10M images served(Small deployment size)	125M images served(Medium deployment size)	500M images served(Large deployment size)
Amazon CloudFront	\$200	\$200	\$1000

AWS Service	10M images served(Small deployment size)	125M images served(Medium deployment size)	500M images served(Large deployment size)
AWS ECS	\$72.08	\$216.24	\$576.64
Application Load Balancer	\$16.79	\$22.27	\$34.30
CloudWatch Logs	\$1.15	\$19.17	\$75.67
<b>Total</b>	<b>\$290.02</b>	<b>\$457.68</b>	<b>\$1686.61</b>

## Cost considerations

**Cost Considerations:** Lambda architecture incurs AWS Secrets Manager costs only when image URL signature feature is activated, and operational dashboard usage may fall under CloudWatch free tier. ECS architecture costs reflect continuous operation with minimum task counts, with auto-scaling increasing costs during high-traffic periods, and DynamoDB costs based on typical usage patterns. Additional costs may include Amazon S3 PUT/GET requests depending on caching effectiveness, Amazon Rekognition charges for smart cropping or content moderation features, external origin data transfer costs, and negligible Cognito costs for Admin UI authentication (typically under \$1/month for single-user operation).

*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 setup 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.

## Amazon WAF

This solution's default configuration doesn't deploy a web application firewall (WAF) in front of the API endpoints. To enhance your API security by setting up a WAF, you must do so manually. AWS provides an in-depth guide on how you can control access to your API Gateway with AWS WAF. For instructions on how to implement AWS WAF in front of your API and increase distributed denial of service (DDoS) protection for your web applications, see [Using AWS WAF to protect your APIs](#).

[Get started with AWS WAF](#)

## 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 *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 Configurations

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.

## Amazon CloudWatch Logs and Alarms

This solution captures application and service logs by creating CloudWatch logs groups in your account. By default, logs are kept indefinitely and never expire. You can adjust the `LogRetentionPeriod` parameter for each log group, keeping the indefinite retention, or choosing a retention period between on day and 10 years based on your requirements.

CloudWatch alarms help you monitor the solution's functional and security assumptions are being followed. Following CloudWatch metrics can be leveraged with API Gateway to monitor client/server errors.

[What is Amazon CloudWatch Logs?](#)

[Amazon API Gateway dimensions and metrics](#)

[Creating CloudWatch alarms to monitor API Gateway](#)

# Deploy the solution

AWS Launch Wizard is the recommended deployment method for this solution in commercial AWS Regions. It provides:

- A guided configuration experience with detailed help panels at each step
- A centralized page to monitor the health of all your deployments
- Indication when there is a more recent version of the solution available for deployment or upgrade

For opt-in Regions, AWS Launch Wizard is not supported. Use the [AWS CloudFormation template](#) to deploy in these Regions. For information about opt-in Regions, refer to [Managing AWS Regions](#) in the *AWS General Reference guide*.

## Deployment process overview

Before you deploy the solution, review the [cost](#), [architecture](#), [security](#), and other considerations discussed earlier in this guide. Additionally, review the [deployment architecture options](#) to determine which template best meets your requirements.

**Time to deploy:** Approximately 20 minutes.

### Note

This solution includes data collection 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 through this survey. Data collection is subject to the [AWS Privacy Notice](#).

### Note

You are responsible for the cost of the AWS services used while running this solution. For more details, visit the [Cost](#) section in this guide and refer to the pricing webpage for each AWS service used in this solution.

## Deploy using AWS Launch Wizard

This solution features a guided deployment process using AWS Launch Wizard. Follow these steps to deploy Dynamic Image Transformation for Amazon CloudFront into your account.

1. Sign in to the AWS Management Console and select the button below to start the deployment process.

A blue rounded rectangular button with the text "Launch solution" in white.

2. If there are more than one deployment patterns available for the solution, select the one that's most applicable to your use case.
3. Select a version to deploy. The latest version is recommended.
4. Click on the **Launch deployment wizard** button.

You will then follow a series of steps to collect the information needed to deploy the solution. It will take approximately 20 minutes to provision the required resources. Select your deployment from the [Deployment list](#) to view its status.

### Note

AWS Launch Wizard is not available in opt-in Regions. To deploy in an opt-in Region, refer to [Deploy using AWS CloudFormation](#).

## Deploy using AWS CloudFormation

This solution uses [CloudFormation templates and stacks](#) to automate its deployment. The solution provides two CloudFormation templates, each optimized for different deployment architectures. The CloudFormation stack provisions the resources that are described in the templates.

### AWS CloudFormation templates

This solution provides two CloudFormation templates, each designed for a specific deployment architecture:

**Lambda Architecture Template** [dynamic-image-transformation-for-amazon-cloudfront-lambda.template](#) - Use this template for cost-optimized deployments with images up to 6 MB. The configuration deploys CloudFront, API Gateway, Lambda, CloudWatch, and EventBridge.

**ECS Architecture Template** [dynamic-image-transformation-for-amazon-cloudfront-ecs.template](#) - Use this template for high-performance deployments with images up to 100 MB. The configuration deploys CloudFront, Application Load Balancer, ECS with Fargate, DynamoDB, Amplify Admin UI, Cognito, CloudWatch, and EventBridge.

**Note**

CloudFormation resources are created from AWS CDK constructs.

**Template selection guidance:** - Choose the **Lambda template** for cost-optimized deployments with basic transformation needs - Choose the **ECS template** for advanced features including transformation policies, non-S3 origins, and administrative interface - Refer to the [deployment architecture guide](#) for detailed comparison

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 comma-separated 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.

## Deploy Lambda architecture

Follow the step-by-step instructions in this section to configure and deploy the cost-optimized Lambda architecture 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-lambda AWS CloudFormation template.
2. Sign into [AWS Management Console](#) and select the button to launch dynamic-image-transformation-for-amazon-cloudfront-lambda 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.

## Lambda architecture template parameters

+

Parameter	Default	Description
<b>CORS Enabled</b>	No	Choose whether to activate CORS. For information about this parameter, refer to <a href="#">Cross-origin resource sharing (CORS)</a> .
<b>CORS Origin</b>	*	This value is returned by the API in the <b>Access-Control-Allow-Origin</b> 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.  <b>Note:</b> This value is ignored if <b>CORS_ENABLED</b> is set to No.
<b>Source Buckets</b>	<Requires input>	Specifies the S3 bucket (or buckets) in your account that contain(s) the images that you manipulate. To specify multiple buckets, separate them by commas.
<b>Enable S3 Object Lambda (Deprecated)</b>	No	This option has been deprecated. Amazon S3 Object Lambda will no longer be open to new customers starting on November 7, 2025. If you were not an existing user of S3 Object

Parameter	Default	Description
		<p>Lambda before November 7, 2025, select 'No'. For more information, please visit <a href="https://docs.aws.amazon.com/AmazonS3/latest/userguide/amazons3-ol-change.html">https://docs.aws.amazon.com/AmazonS3/latest/userguide/amazons3-ol-change.html</a>.</p> <p>Determines which component 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.</p>
<b>Deploy Demo UI</b>	Yes	The demo UI that deploys to the Demo S3 bucket. For more information refer to <a href="#">Use the demo UI</a> .
<b>Log Retention Period</b>	180	Specifies the number of days to retain Lambda log data in CloudWatch logs.
<b>Enable Signature</b>	No	Choose whether to activate the image URL signature feature. For information about this feature, refer to <a href="#">Image URL signature</a> .

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

Parameter	Default	Description
<b>Fallback Image S3 Key</b>	<Optional input>	<p>Specify the default fallback image S3 object key, including prefix. See <a href="#">Creating object key names</a> for more information.</p> <p><b>Note:</b> This value is ignored if the <b>Enable Default Fallback Image</b> parameter is set to No.</p>
<b>AutoWebP</b>	No	<p>Choose whether to automatically convert responses to the <a href="#">WebP</a> image formats if the Accept request header allows it.</p>
<b>Origin Shield Region</b>	Disabled	<p>The Region to set up the Origin Shield caching layer for the CloudFront distribution. 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 <a href="#">Amazon CloudFront Developer Guide</a>.</p>
<b>CloudFront PriceClass</b>	PriceClass_All	<p>The CloudFront price class to use. For more information, refer to <a href="#">Choosing the price class for a CloudFront distribution</a> in the <i>Amazon CloudFront Developer Guide</i>.</p>

Parameter	Default	Description
<b>Use Existing CloudFront Distribution</b>	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 <a href="#">Attaching an Existing CloudFront distribution</a> for more information.
<b>Existing CloudFront Distribution ID</b>	<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.  <b>Note:</b> This value is ignored if <b>Use Existing CloudFront Distribution</b> is set to No.

1. Choose **Next**.
2. On the **Configure stack options** page, choose **Next**.
3. On the **Review and create** page, review and confirm the settings. Select the box acknowledging that the template creates IAM resources.
4. 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.

## Deploy ECS architecture

Follow the step-by-step instructions in this section to configure and deploy the high-performance ECS architecture into your account.

**Time to deploy:** Approximately 20 minutes

1. Sign in to the [AWS Management Console](#) and select the button to launch the dynamic-image-transformation-for-amazon-cloudfront-ecs AWS CloudFormation template.
2. Sign into [AWS Management Console](#) and select the button to launch dynamic-image-transformation-for-amazon-cloudfront-ecs 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.

## ECS architecture template parameters

+

Parameter	Default	Description
<b>Admin Email</b>	<Requires input>	The email address of the admin user for the Admin UI.

Parameter	Default	Description
		Must be a valid email address (7-100 characters). This email will receive the initial login credentials for the administrative interface.
<b>Deployment Size</b>	small	T-shirt sizing for ECS Fargate deployment configuration. Options: small, medium, large, xlarge. This determines the number of ECS tasks, CPU, and memory allocation. Refer to <a href="#">sizing guidance</a> for details.
<b>Origin Override Header</b>	<Optional input>	HTTP header name used to override the origin destination for image requests. Must be a valid HTTP header name or empty. Useful for routing requests to different origins based on custom headers.
<b>Cors Origin Parameter</b>	<Optional input>	If you would like to specify an origin to use for CORS, please specify an origin value here. We recommend specifying an origin (i.e. https://example.domain) to restrict cross-site access to your API. Leave empty to default to wildcard (*).

1. Choose **Next**.
2. On the **Configure stack options** page, choose **Next**.

3. On the **Review and create** page, review and confirm the settings. Select the box acknowledging that the template creates IAM resources.
4. 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 20 minutes.

## Post-deployment configuration (ECS architecture)

After deploying the ECS template, additional configuration steps are required to fully utilize the advanced features:

**Admin UI Access:** The Admin UI link is available in the CloudFormation stack outputs section. Access the Admin UI from there and sign in using the provided Cognito credentials.

### Initial Configuration:

- **Configure Origins:** Use the Admin UI to add your S3 buckets and external origins
- **Create Mappings:** Set up path-based or host-header mappings to route requests to origins
- **Define Policies:** Create transformation policies for consistent image processing

## Attaching an existing CloudFront distribution

This section provides instructions for integrating the solution with your existing CloudFront distribution for both architectures.

### 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-`.

## Lambda architecture

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

## 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. 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. Set the Origin path to `/image`.
5. 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. Set the Viewer request Function type to `CloudFront Functions`, and the Function ARN to the one named `sih-apig-request-modifier-${UUID}`.
9. Select **Create behavior**.

## ECS architecture

For the ECS architecture, deploy the solution as-is to provision all necessary resources. Then manually configure your existing CloudFront distribution to use the solution's resources.

## Deploy the solution

1. Deploy the ECS architecture CloudFormation template without specifying an existing CloudFront distribution.

2. The solution will create its own CloudFront distribution along with all required resources including the Application Load Balancer and CloudFront function.

### Configure your existing distribution

After the solution deployment is complete:

1. In the CloudFront console, navigate to your existing CloudFront distribution.
2. Select the Origins tab and click **Create origin**.
3. Set the Origin domain as the Application Load Balancer DNS name. This value can be found in the CloudFormation stack outputs under the key `LoadBalancerDNS`.
4. Leave the Origin path empty (default).
5. Select **Create origin**.

### Create behavior for image processing

1. In your existing CloudFront distribution, select the Behaviors tab and choose **Create behavior**.
2. Set the Path pattern for image requests (e.g., `/images/*` or your preferred pattern).
3. Set the Origin to the ALB origin created in the previous step.
4. Set the Viewer Protocol policy to `Redirect HTTP to HTTPS`.
5. Set the Cache Policy to the one named `dit-cache-policy` (created by the solution).
6. Set the Response headers policy to the one named `SecurityHeadersPolicy` (created by the solution).
7. Set the Viewer request Function type to `CloudFront Functions`, and the Function ARN to the one named `dit-header-normalization` (created by the solution).
8. Select **Create behavior**.

This approach allows you to leverage the solution's provisioned resources while maintaining control over your existing CloudFront distribution configuration.

# Update the solution

Updating the solution applies the latest features, security patches, and bug fixes to your deployment. To update to the latest version, refer to the appropriate section based on your original deployment method: [AWS Launch Wizard](#) or [AWS CloudFormation](#).

## Update using AWS Launch Wizard

The console automatically displays the newest available version of the solution in the **Deployment version** dropdown. If you have previously deployed the solution, follow this procedure to update your deployment to the latest version.

1. Go to [Launch Wizard Deployments](#).
2. Select the deployment you want to update.
3. Choose **Actions**, then **Update deployment version**.
4. Select the latest version from the available **Deployment versions**.
5. Review configuration.
6. Make changes needed on each step.
7. Confirm the update.

## Update using AWS CloudFormation

### 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.

S3 Object Lambda has been deprecated. Amazon S3 Object Lambda will no longer be open to new customers starting on November 7, 2025. If you were not an existing user of S3 Object Lambda before November 7, 2025, select 'No'. For more information, please visit <https://docs.aws.amazon.com/AmazonS3/latest/userguide/amazons3-ol-change.html>.

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](#).

### Important

#### Version 8.0.0 Breaking Changes:

Dynamic Image Transformation for Amazon CloudFront version 8.0.0 introduces significant architectural changes and is **not compatible** with previous versions. This release includes:

- **New ECS Architecture:** The new high-performance container-based architecture cannot be deployed as an update to existing stacks
- **Breaking Changes:** Version 8.0.0 includes fundamental changes to the solution architecture, configuration, and feature set
- **Clean Deployment Required:** To use the ECS architecture or any v8.0.0 features, you must deploy a **new stack** using the v8.0.0 CloudFormation template

**Migration Path:** - **Lambda Architecture:** Existing deployments can be updated to v8.0.0 Lambda template for maintenance and security updates - **ECS Architecture:** Requires a completely new deployment - cannot be updated from any previous version - **Feature Access:** Advanced features (transformation policies, non-S3 origins, Admin UI) are only available in the new ECS architecture


**Recommendation:** Deploy the new ECS architecture as a separate stack, test thoroughly, then migrate traffic and [uninstall](#) the previous version.

## Update Lambda architecture

Follow these steps to update an existing Lambda architecture deployment to the latest version:

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-lambda.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-lambda.template` would become `https://solutions-reference.s3.amazonaws.com/dynamic-image-transformation-for-amazon-cloudfront/v8.0.0/dynamic-image-transformation-for-amazon-cloudfront-lambda.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 [Lambda architecture template parameters](#).
  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.

## Update ECS architecture

Follow these steps to update an existing ECS architecture deployment to the latest version:

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-ecs.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.
  - e. Verify that the correct template URL shows in the *\*Amazon S3 URL\** text box, and choose **Next**. Choose **Next** again.
4. Under **Parameters**, review the parameters for the template and modify them as necessary. For details about the parameters, see [ECS architecture template parameters](#).
5. Choose **Next**.
6. On the **Configure stack options** page, choose **Next**.
7. On the **Review** page, review and confirm the settings. Select the box acknowledging that the template creates IAM resources.
8. Choose **View change set** and verify the changes.
9. Choose **Update stack** to deploy the stack.

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

## Backward compatibility

### Note

**Lambda Architecture Only:** The backward compatibility features described in this section only apply to the Lambda architecture template. The ECS architecture uses a different request format and does not support legacy Thumbor or custom request formats.

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 Business Support+](#), [AWS Enterprise Support](#), or [Unified Operations](#), you can use the AWS 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 **Dynamic Image Transformation for Amazon CloudFront**.
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, including the name of this solution and the version you are using, such as this example: **Dynamic Image Transformation for Amazon CloudFront vX.Y.Z**.
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**.

## Transformation policy errors

### Policy validation failures

**Symptoms:** - 400 Bad Request errors when creating policies - Policy creation fails in Admin UI - "Invalid policy JSON" error messages

#### Solutions:

##### Check JSON syntax:

- Validate JSON structure using online JSON validators
- Ensure all required fields are present
- Verify transformation parameter types match schema requirements

**Review transformation limits:** - Policy size limit: 400KB - Ensure transformation values are within valid ranges

### Policy not being applied

**Symptoms:** - Images not transformed according to policy - Default transformations applied instead of policy - Policy appears in Admin UI but doesn't affect images

#### Solutions:

##### Verify policy mapping:

- Check if policy is correctly associated with origin mapping
- Ensure policy ID is specified correctly in requests
- Verify default policy is set if no policy ID provided

### **Refresh configuration cache:**

- Use Admin UI "Refresh Cache" button after policy changes
- Wait 5 minutes for rolling update to complete
- Check ECS task logs for cache refresh confirmation

## **Conditional transformation issues**

**Symptoms:** - Conditional transformations not triggering - Wrong transformation applied based on conditions - Client hints not being evaluated correctly

### **Solutions:**

#### **Review condition syntax:**

- Verify field names match available headers
- Check condition values and operators
- Ensure client is sending required headers

#### **Test condition logic:**

- Use Admin UI testing tools to validate conditions
- Check request headers in browser developer tools
- Verify condition evaluation in ECS task logs

## **Seeing cached images even after making configuration changes on Admin UI**

**Problem:** Images continue to show previous transformations even after updating origins, policies, or mappings through the Admin UI.

**Cause:** CloudFront caches processed images based on the request URL and headers. Configuration changes don't automatically invalidate existing cached images.

**Solution:** Create a CloudFront invalidation to clear cached images:

1. Navigate to the CloudFront console
2. Select the distribution created by the solution
3. Go to the **Invalidations** tab
4. Click **Create invalidation**
5. Enter the path pattern for images to invalidate:
  - For all images: /\*
  - For specific paths: /images/ or /mobile/
6. Click **Create invalidation**

The invalidation will clear cached images, forcing CloudFront to request fresh images with the new configuration applied.

## Enable debug logs

**Problem:** Need detailed logging information for troubleshooting image processing issues.

**Solution:** Debug logs are not enabled by default. To enable debug logging:

1. Navigate to the AWS Lambda console
2. Find the image processing Lambda function created by the solution
3. Go to the **Configuration** tab
4. Select **Environment variables**
5. Add or modify the debug logging environment variables (exact variable names can be found in the CDK construct)
6. Save the changes

Debug logs will provide detailed information about image processing operations, transformation steps, and error details to help with troubleshooting.

## Tracing API errors using request ID

**Problem:** Need to identify the root cause of API errors or failed image processing requests.

**Solution:** Each API response includes an `x-amz-request-id` header that can be used to trace the request in CloudWatch logs:

1. Capture the `x-amz-request-id` from the API response headers
2. Navigate to the CloudWatch console
3. Go to **Log groups**
4. Find the log group for the image processing service:
  - Lambda architecture: Look for the Lambda function log group
  - ECS architecture: Look for the ECS task log group
5. Search for the request ID in the logs:
  - Use the CloudWatch Logs Insights query: `fields @timestamp, @message | filter @message like /REQUEST_ID/`
  - Replace `REQUEST_ID` with the actual request ID value
6. Review the log entries to identify error messages, stack traces, or processing details

The request ID allows you to trace the complete request lifecycle and identify exactly where errors occurred during image processing.

# Uninstall the solution

## Important

**Data Loss Warning:** Deleting the solution will permanently delete all origins, transformation policies, and mappings configured in the Admin UI. This data cannot be recovered after deletion. If you need to preserve your configuration, take manual backups of your origins, policies, and mappings before proceeding with the uninstall process.

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

### AWS CloudFormation

1. Sign in to the [AWS CloudFormation console](#).
2. On the **Stacks** page, select this solution's installation stack.
3. Choose **Delete**.

### AWS Launch Wizard

1. Sign in to the AWS Launch Wizard console.
2. On the [Launch Wizard Deployments](#) page, select this solution's deployment.
3. Choose **Actions**, then **Delete**.
4. Confirm the deletion.

## 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 based on your chosen architecture.

## Lambda architecture

This section covers how to use the solution when deployed with the Lambda architecture.

## 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.**

Serverless Image Handler Demo

**Image Source**

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

Import

**Original Image**

**Editor**

Width: 200 Height: 200 Resize Mode: Disabled

Fill Color: #FF0000 Background Color: #FF0000

Grayscale  Negative  RGB  
 Flip  Flatten  RGB  
 Flop  Normalize 0, 0, 255

Smart Cropping Crop Padding: 0  
Focus Index: 0

Reset Preview

**Preview**

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: `YYYYMMDDTHHmssZ`. 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
<a href="#"><u>format</u></a>	Sets the output to the provided format	jpg, jpeg, heic, png, raw, tiff, webp, gif, avif	None
<a href="#"><u>fit</u></a>	The method that should be used when resizing	cover, contain, fill, inside, outside	cover
<a href="#"><u>width</u></a>	The width in pixels, the image should be resized to.	Positive integer or 0	None
<a href="#"><u>height</u></a>	The height in pixels, the image should be resized to.	Positive integer or 0	None
<a href="#"><u>rotate</u></a>	The number of degrees the image should be rotated	0-359 or blank (for null)	None
<a href="#"><u>flip</u></a>	Mirror the image vertically	True/False	False

Query parameter name	Description	Options	Default
<a href="#">flop</a>	Mirror the image horizontally	True/False	False
<a href="#">greyscale</a>	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
<b>Animated</b>	<code>/filters:animated(true/false)/</code>
<b>Autojpg</b>	<code>/filters:autojpg()/</code>
<b>Background color</b>	<code>/filters:background_color(c olor)/</code>
<b>Blur</b>	<code>/filters:blur(7)/</code>
<b>Color fill</b>	<code>/filters:fill(color)/</code>
<b>Convolution</b>	<code>/filters:convolution(1;2;1; 2;4;2;1;2;1,3,false)/</code>
<b>Crop</b>	<code>/10x10:100x100/</code>
<b>Equalize</b>	<code>/filters:equalize()/</code>
<b>Grayscale</b>	<code>/filters:grayscale()/</code>

Filter name	Filter syntax
<b>Image format</b> (.gif, .jpeg, .png, .avif, .webp, .tiff)	/filters:format(image_format)
<b>No upscale</b>	/filters:no_upscale()/
<b>Proportion</b>	/filters:proportion(0.0-1.0)/ ,
<b>Quality</b>	/filters:quality(0-100)/
<b>Resize</b>	/fit-in/800x1000/
<b>RGB</b>	/filters:rgb(20,-20,40)/
<b>Rotate</b>	/filters:rotate(90)/
<b>Sharpen</b>	/filters:sharpen(0.0-10.0, 0.0-2.0, true/false)/
<b>Smart Crop</b>	/filters:smart_crop(faceIndex, facePadding)/
<b>Stretch</b>	/filters:stretch()/
<b>Strip Exif</b>	/filters:strip_exif()/
<b>Strip ICC</b>	/filters:strip_icc()/
<b>Upscale</b>	/filters:upscale()/
<b>Watermark</b>	/filters:watermark(bucket,k ey,x,y,alpha[,w_ratio[,h_ra tio]])

## 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
<b>AUTO_WEBP</b>	Yes/No	Choose whether to automatically accept webp image formats.
<b>CORS_ENABLED</b>	Yes/No	Indicates whether to return an <b>Access-Control-Allow-Origin</b> header with the image handler API response.
<b>CORS_ORIGIN</b>	String	<p>This value is returned by the API in the <b>Access-Control-Allow-Origin</b> header. An asterisk value supports any origin. We recommend specifying a specific origin (Ex: <code>https://example.domain</code> ) to restrict cross-site access to your API.</p> <p><b>Note:</b> This value is ignored if <b>CORS_ENABLED</b> is set to No.</p>
<b>ENABLE_DEFAULT_FALLBACK_IMAGE</b>	Yes/No	Choose whether to return the default fallback image when errors occur.
<b>DEFAULT_FALLBACK_IMAGE_BUCKET</b>	String	<p>Specifies the S3 bucket which contains the default fallback image.</p> <p><b>Note:</b> This value is ignored if the <b>ENABLE_DEFAULT_FALLBACK_IMAGE</b> parameter is set to No.</p>

Variable Key	Value Type	Description
<b>DEFAULT_FALLBACK_IMAGE_KEY</b>	String	Defines the default fallback image S3 object key, including the prefix.  <b>Note:</b> This value is ignored if the <b>ENABLE_DEFAULT_FALLBACK_IMAGE</b> parameter is set to No.
<b>ENABLE_SIGNATURE</b>	Yes/No	Choose whether to use the image URL signature.
<b>REWRITE_MATCH_PATTERN</b>	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, <code>/(filters-)/gm</code> .
<b>REWRITE_SUBSTITUTION</b>	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, <code>filters:.</code>

Variable Key	Value Type	Description
<b>SECRETS_MANAGER</b>	String	Defines the Secrets Manager secret that contains the secret key for the image URL signature.  <b>Note:</b> This value is ignored if <code>[.replaceable]ENABLE_SIGNATURE</code> is set to No.
<b>SECRET_KEY</b>	String	Defines the Secrets Manager secret key that contains the secret value to create the image URL signature.  <b>Note:</b> This value is ignored if <b>ENABLE_SIGNATURE</b> is set to No.
<b>SOURCE_BUCKETS</b>	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, <code>(filters-)/gm</code> .

Variable Key	Value Type	Description
<b>REWRITE_SUBSTITUTION</b>	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, <code>filters:</code> .

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
```

- 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.

## ECS architecture

This section covers how to use the solution when deployed with the ECS architecture.

## Deploy the template

Deploy the ECS CloudFormation template following the instructions in [Deploy ECS architecture](#). The deployment will provision all necessary resources including the Admin UI, Application Load Balancer, ECS service, and CloudFront distribution.

## Access the Admin UI

After deployment completes:

- Navigate to the CloudFormation stack outputs in the AWS console.

2. Find the `AdminUIUrl` output value and open the link.
3. Sign in using the Cognito credentials sent to the admin email address specified during deployment.

## Create origins

Origins define the source locations for your images.

1. In the Admin UI, navigate to the Origins section.
2. Click **Create Origin** and provide:
  - **Origin Name:** Descriptive name for the origin
  - **Origin Domain:** Domain name (e.g., `my-bucket.s3.amazonaws.com`)
  - **Origin Path:** Optional path prefix (e.g., `/images`)
  - **Origin Headers:** Optional custom headers
3. Click **Save** to create the origin.

## Create transformation policies

Transformation policies define how images are processed based on conditions. You can create policies using either the Admin UI interface or by providing JSON configuration directly.

### Using the Admin UI

1. In the Admin UI, navigate to the **Policies** section.
2. Click **Create Policy** and provide:
  - **Policy Name:** Descriptive name for the policy
  - **Description:** Optional description of what the policy does
3. Configure transformations using the UI:
  - Click **Add Transformation** to add image processing operations
  - Select transformation type (resize, format, quality, etc.)
  - Configure transformation parameters using the form fields
4. Configure outputs using the UI:
  - Click **Add Output** to define output specifications
  - Select output type (quality, format, autosize)

- Configure output parameters using the form fields
5. Click **Save** to create the policy.

## Using Management API

Alternatively, you can create policies by providing JSON configuration directly using the Management API:

1. In the Admin UI, navigate to the **Policies** section.
2. Click **Create Policy** and provide:
  - **Policy Name:** Descriptive name for the policy
  - **Description:** Optional description
  - **Policy JSON:** JSON configuration defining transformations and outputs
3. Example policy JSON:

```
{
  "outputs": [
    {
      "type": "quality",
      "value": [
        77,
        [0,1,50],
        [1,2,75],
        [2,500,90]
      ]
    },
    {
      "type": "format",
      "value": "auto"
    },
    {
      "type": "autosize",
      "value": [320,480,640,960,1440,1920]
    }
  ],
  "transformations": [
    {
      "transformation": "blur",
      "value": 7
    },
  ],
}
```

```
{
  "transformation": "convolve",
  "value": {
    "width": 3,
    "height": 3,
    "kernel": [1,0,-1,0,0,0,-1,0,1]
  }
},
{
  "transformation": "extract",
  "value": [10,10,100,100]
},
{
  "transformation": "flip",
  "value": true,
  "condition" : {
    "field": "header.XYZ",
    "value": "ABC"
  }
},
{
  "transformation": "flop",
  "value": false
},
{
  "transformation": "grayscale",
  "value": true
},
{
  "transformation": "normalize",
  "value": true
},
{
  "transformation": "resize",
  "value": {
    "width": 400,
    "height": 600,
    "fit": "contain"
  }
},
{
  "transformation": "rotate",
  "value": 60,
```

```
        "condition" : {
            "field": "header.XYZ",
            "value": "ABC"
        }
    },
    {
        "transformation": "sharpen",
        "value": {
            "sigma": 5
        }
    },
    {
        "transformation": "smartCrop",
        "value": true
    },
    {
        "transformation": "stripExif",
        "value": true
    },
    {
        "transformation": "stripIcc",
        "value": true
    },
    {
        "transformation": "tint",
        "value": "blue"
    },
    {
        "transformation": "watermark",
        "value": [ "watermarkURL", [15, 15, 0.1, 0.4, 0.4]]
    }
]
}
```

4. Click **Save** to create the policy.

## Transformation filter reference

The ECS architecture supports a comprehensive set of image transformation filters using a simplified syntax. The following table provides the complete filter reference:

Filter Name	Filter Syntax	Notes
<b>Blur</b>	<code>blur=30</code>	Integer from 0.3 to 1000
<b>Convolve</b>	<code>convolve.width=3 convolve.height=3 convolve.kernel=[1,0,-1,0,0,0,-1,0,1]</code>	Requires 3 parameters: width, height, kernel
<b>Extract</b>	<code>extract=[10,10,200,200]</code>	Array of 4 non-negative integers
<b>Flatten</b>	<code>flatten=aliceblue flatten=[0,0,255,1]</code>	Accepts color names or RGBA tuples
<b>Flip</b>	<code>flip=true</code>	Boolean
<b>Flop</b>	<code>flop=true</code>	Boolean
<b>Format</b>	<code>format=webp</code>	Accepts: jpg, jpeg, png, tiff, webp, gif, avif
<b>Greyscale</b>	<code>greyscale=true</code>	Boolean
<b>Normalize</b>	<code>normalize=true</code>	Boolean
<b>Quality</b>	<code>quality=0.5</code>	Integer from 0 to 1
<b>Resize</b>	<code>resize.width=200 resize.ratio=0.5 resize.fit=contain resize.withoutEnlargement=true resize.background=blue</code>	Must specify: height, width, or ratio
<b>Rotate</b>	<code>rotate=90</code>	Integer
<b>Sharpen</b>	<code>sharpen=true or sharpen=sigma=5 sharpen.m1=2 sharpen.m2=1 sharpen.x1=2</code>	Accepts either boolean to perform a fast mild sharpen, or additional parameters for a slower but more accurate sharpen. See <a href="#">Sharp docs</a> for more information.

Filter Name	Filter Syntax	Notes
	<code>sharpen.y2=20 sharpen.y3=20</code>	
<b>Smart Crop</b>	<code>smartCrop=true</code> or <code>smartCrop.index=1 smartCrop.padding=200</code>	Accepts either boolean or index + padding parameters. Boolean will automatically perform cropping.
<b>Strip ICC</b>	<code>stripIcc=true</code>	Strip ICC and enforces sRGB color space
<b>Strip EXIF</b>	<code>stripExif=true</code>	Removes image metadata
<b>Tint</b>	<code>tint=aliceblue</code> <code>tint=[0,0,255,1]</code>	Accepts color names or RGBA tuples
<b>Watermark</b>	<code>watermark=https://example.com/overlayImage.png,[15,15,0.1,0.4,0.4]</code>	Expects a format of: [watermarkURL, [x, y, alpha, widthRatio, heightRatio]] Note: For security reasons, the origin the watermark image is hosted at must be configured as an origin within DIT.

## Create mappings

Mappings connect URL patterns to specific origins and transformation policies.

1. In the Admin UI, navigate to the Mappings section.
2. Click **Create Mapping** and provide:
  - **Mapping Name:** Descriptive name for the mapping
  - **Mapping Type:** Choose `PATH_MAPPING` or `HOST_HEADER_MAPPING`
  - **Pattern:** URL pattern (e.g., `/mobile/*` or `example.com`)
  - **Origin:** Select the origin created earlier
  - **Policy:** Select the transformation policy (optional)
  - **Priority:** Numeric priority for mapping evaluation

3. Click **Save** to create the mapping.

## Test image transformations

Test your configuration using the CloudFront distribution created by the solution.

1. Get the CloudFront distribution URL from the CloudFormation stack outputs (CloudFrontDistributionDomainName).
2. Access images using the pattern defined in your mappings:

```
https://<cloudfront-domain>/<mapping-pattern>/<image-path>
```

3. Example:

```
https://d1234567890.cloudfront.net/mobile/sample-image.jpg
```

4. The image will be processed according to the transformation policy associated with the matching mapping.

## Monitor and manage

Use the Admin UI to:

- View and edit existing origins, policies, and mappings
- Update configurations as needed

The ECS architecture provides a scalable, containerized solution for advanced image processing with full administrative control through the web-based Admin UI.

## Developer guide

This section includes API specifications for Admin APIs, transformation policy schemas, parameter references, information about anonymized data collection, performance benchmarks, and related resources for the Dynamic Image Transformation for Amazon CloudFront 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

The ECS architecture includes Admin APIs for managing origins, transformation policies, and mappings. These APIs are secured with AWS Cognito authentication and provide programmatic access to configuration management.

### Authentication

All Admin API requests require authentication through AWS Cognito:

```
Authorization: Bearer <cognito-jwt-token>  
Content-Type: application/json
```

### Transformation Policies API

Manage image transformation policies that define how images are processed.

#### List Policies

```
GET /policies?nextToken={token}
```

Response:

```
{
  "items": [
    {
      "policyId": "550e8400-e29b-41d4-a716-446655440000",
      "policyName": "mobile-optimized",
      "description": "Mobile device optimization policy",
      "policyJSON": "{\"transformations\":[...]}",
      "isDefault": false,
      "createdAt": "2024-01-15T10:30:00Z",
      "updatedAt": "2024-01-15T10:30:00Z"
    }
  ],
  "nextToken": "optional-token-for-next-page"
}
```

## Create Policy

```
POST /policies
{
  "policyName": "mobile-optimized",
  "description": "Mobile device optimization policy",
  "policyJSON": "{\"transformations\":[{\n\"condition\":\n\"width > 800\",\n\"operations\":\n[{\n\"resize\":{\n\"width\":800,\n\"height\":600,\n\"fit\":\n\"cover\"}],{\n\"format\":\n\"webp\"},\n{\n\"quality\":85}]]]}",
  "isDefault": false
}
```

## Get Policy

```
GET /policies/{policyId}
```

## Update Policy

```
PUT /policies/{policyId}
{
  "policyName": "updated-mobile-optimized",
  "description": "Updated mobile device optimization policy",
  "policyJSON": "{\"transformations\":[...]}"
}
```

## Delete Policy

```
DELETE /policies/{policyId}
```

## Origins API

Manage origin configurations that define source locations for images.

### List Origins

```
GET /origins?nextToken={token}
```

Response:

```
{
  "items": [
    {
      "originId": "550e8400-e29b-41d4-a716-446655440001",
      "originName": "my-s3-origin",
      "originDomain": "my-images-bucket.s3.amazonaws.com",
      "originPath": "/images",
      "originHeaders": {
        "x-custom-header": "value"
      },
      "createdAt": "2024-01-15T10:30:00Z",
      "updatedAt": "2024-01-15T10:30:00Z"
    }
  ],
  "nextToken": "optional-token-for-next-page"
}
```

### Create Origin

```
POST /origins
{
  "originName": "my-s3-origin",
  "originDomain": "my-images-bucket.s3.amazonaws.com",
  "originPath": "/images",
  "originHeaders": {
    "x-custom-header": "value"
  }
}
```

## Get Origin

```
GET /origins/{originId}
```

## Update Origin

```
PUT /origins/{originId}
{
  "originName": "updated-origin-name",
  "originDomain": "updated-bucket.s3.amazonaws.com"
}
```

## Delete Origin

```
DELETE /origins/{originId}
```

# Mappings API

Manage path-based or host-header based routing to map requests to specific origins and policies.

## List Mappings

```
GET /mappings?nextToken={token}
```

### Response:

```
{
  "items": [
    {
      "mappingId": "550e8400-e29b-41d4-a716-446655440002",
      "mappingName": "mobile-path-mapping",
      "description": "Mobile path routing",
      "mappingType": "PATH_MAPPING",
      "pattern": "/mobile/*",
      "originId": "550e8400-e29b-41d4-a716-446655440001",
      "policyId": "550e8400-e29b-41d4-a716-446655440000",
      "priority": 100,
      "createdAt": "2024-01-15T10:30:00Z",
      "updatedAt": "2024-01-15T10:30:00Z"
    }
  ]
}
```

```
  ],  
  "nextToken": "optional-token-for-next-page"  
}
```

## Create Path Mapping

```
POST /mappings  
{  
  "mappingName": "mobile-path-mapping",  
  "description": "Mobile path routing",  
  "mappingType": "PATH_MAPPING",  
  "pattern": "/mobile/*",  
  "originId": "550e8400-e29b-41d4-a716-446655440001",  
  "policyId": "550e8400-e29b-41d4-a716-446655440000",  
  "priority": 100  
}
```

## Create Host Header Mapping

```
POST /mappings  
{  
  "mappingName": "subdomain-mapping",  
  "description": "Subdomain routing",  
  "mappingType": "HOST_HEADER_MAPPING",  
  "pattern": "*.example.com",  
  "originId": "550e8400-e29b-41d4-a716-446655440001",  
  "policyId": "550e8400-e29b-41d4-a716-446655440000",  
  "priority": 200  
}
```

## Get Mapping

```
GET /mappings/{mappingId}
```

## Update Mapping

```
PUT /mappings/{mappingId}  
{  
  "mappingName": "updated-mapping-name",  
  "pattern": "/updated-path/*",  
  "priority": 150  
}
```

```
}
```

## Delete Mapping

```
DELETE /mappings/{mappingId}
```

## Error Responses

All APIs return structured error responses:

```
{
  "errorCode": "POLICY_NOT_FOUND",
  "message": "Policy with ID 550e8400-e29b-41d4-a716-446655440000 not found"
}
```

Common error codes: - **BAD\_REQUEST**: Invalid request format or parameters - **INVALID\_JSON**: Malformed JSON in request body - **MISSING\_REQUIRED\_FIELD**: Required field missing from request - **INVALID\_FIELD\_VALUE**: Field value doesn't meet validation requirements - **NOT\_FOUND**: Resource not found - **POLICY\_NOT\_FOUND**: Transformation policy not found - **ORIGIN\_NOT\_FOUND**: Origin configuration not found - **INTERNAL\_SERVER\_ERROR**: Server-side error

# Reference

This section includes information about data collection, 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.

## Data collection

This solution sends operational metrics to AWS (the "Data") about the use of this solution. We use this Data to better understand how customers use this solution and related services and products. AWS's collection of this Data is subject to the [AWS Privacy Notice](#).

## 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
- Bryce Lee
- Chaitanya Deolankar

# Revisions

Publication date: *June 2017 (last update: November 2025)*

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

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

Dynamic Image Transformation for Amazon CloudFront is licensed under the terms of the [Apache License Version 2.0](#).

## Configuration Delay and Cached Images

### Warning

Changes to origins, transformation policies, or mappings may take up to 5 minutes to propagate across all image processing tasks. During this period, some requests may still use the previous configuration. Configuration updates do not automatically refresh cached images. Cached images processed with previous settings will remain available until they expire or are manually invalidated.