

Serverless Image Processing with Amazon AWS

...

CJ O'Hara · cj.ohara@firespring.com

A little about myself - CJ O'Hara

- Senior Software Engineer at Firespring
 - Creative, Marketing, Printing, Websites for nonprofit and print industries
 - Websites team - CMS platform
 - LAMP Stack - AWS
- Lead developer for Givesource - Giving day platform for Nonprofits
 - Runs “Give to Lincoln Day”
 - 100% Serverless through AWS
 - Lambda (NodeJS), API Gateway, Cloudfront, DynamoDB, etc...
- Builder, Tinkerer, Retro Gamer, Father and Husband.



...and procrastinator (sorry)

Are you in the right room?



- Why you may need image processing in your application(s)
- Demo two solutions

I hope you become more familiar with:

- Serverless Architecture
- Image optimization functions
- AWS - CloudFormation, CloudFront, Lambda, S3

Slides and code examples will be provided to those who ~~survive~~ would like a copy.

Follow Along

- Code Examples
- Presentation PDF
- Links to Amazon Articles

<https://github.com/cjohara/aws-serverless-image-processing>

Practical reasons you may need to process images for your application and/or users

Our use-cases:

- Display images on client's websites - help them help themselves.
- Social messaging - Different social platforms have different requirements.
- Mobile optimization.

Other use-cases:

- Custom profile pictures
- Automated image thumbnail generation for galleries or list views
- Using the same image in multiple ways within your branding

Why Serverless?

Image processing, in most cases, is not core functionality, but rather an extension or feature of your application. Keeping resources separated, like your CDN, keeps your application instance sizes small and performance optimal.

- No managing servers - focus on the code
- Auto-scaling for high availability
- Pay for what you use - it's cheap.
- It's fun to explore technologies outside of the LAMP stack

Setup / Prerequisites - S3 Bucket

1. Create an S3 bucket
 - a. Select a region
 - b. Give it a name - these are unique across AWS
 - c. Give it public permissions
2. Images!
 - a. Fill our bucket with royalty free images: [Unsplash](#)



Create your AWS S3 Bucket

Create bucket

① Name and region ② Configure options ③ Set permissions ④ Review

Name and region

Bucket name i

Region

US East (N. Virginia)

US

US East (N. Virginia)

US East (Ohio) Selected

US West (N. California)

US West (Oregon)

Cancel Next

Create bucket

① Name and region ② Configure options ③ Set permissions ④ Review

Note: You can grant access to specific users after you create the bucket.

Block public access (bucket settings)

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, or both. In order to ensure that public access to all your S3 buckets and objects is blocked, turn on Block *all* public access. These settings apply only to this bucket. AWS recommends that you turn on Block *all* public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to your buckets or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#) i

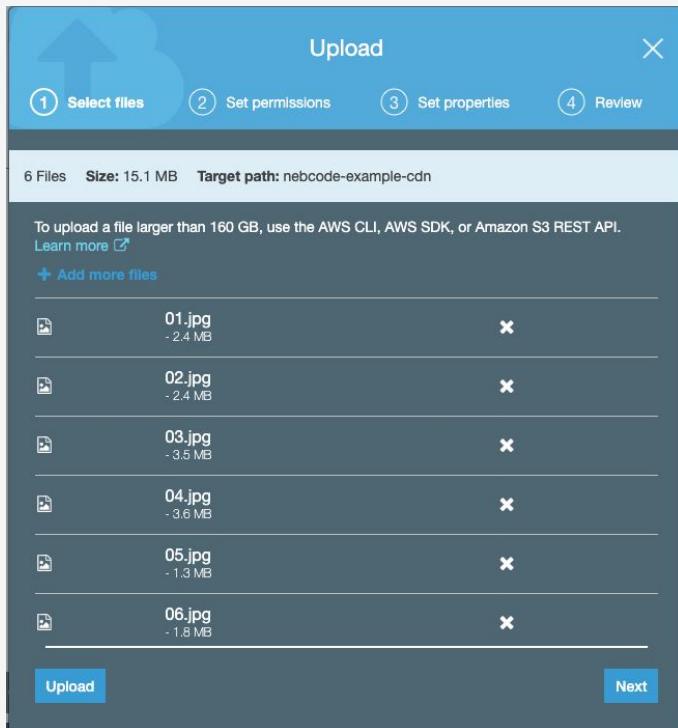
Block *all* public access

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

- Block public access to buckets and objects granted through *new* access control lists (ACLs)**
S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.
- Block public access to buckets and objects granted through *any* access control lists (ACLs)**
S3 will ignore all ACLs that grant public access to buckets and objects.
- Block public access to buckets and objects granted through *new* public bucket policies**
S3 will block new bucket policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.
- Block public and cross-account access to buckets and objects through *any* public bucket policies**
S3 will ignore public and cross-account access for buckets with policies that grant public access to buckets and objects.

Previous Next

Create your AWS S3 Bucket

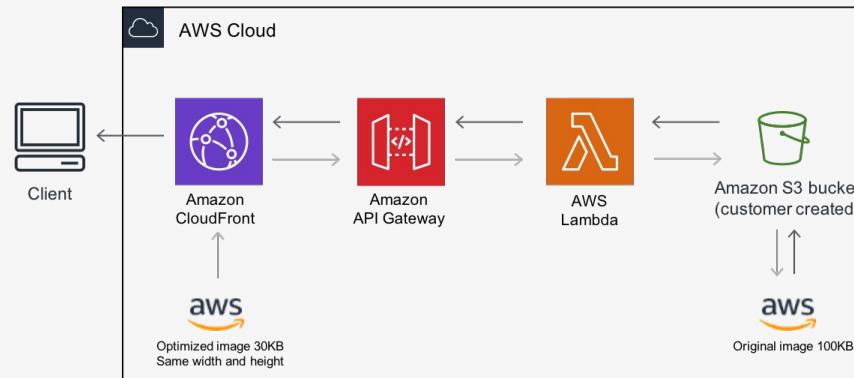


The screenshot shows the AWS S3 bucket list interface. At the top, there is a search bar with placeholder text 'Type a prefix and press Enter to search. Press ESC to clear.', and buttons for 'Upload', 'Create folder', 'Download', and 'Actions'. The main area lists the uploaded files:

Name
01.jpg
02.jpg
03.jpg
04.jpg
05.jpg
06.jpg

Serverless Image Handler - AWS Solution

- Pre-made solution by AWS
- Extensive image manipulation capabilities
- Powered by [Sharp](#) - image manipulation library
- Bring your own front-end (if needed)
- [Full Documentation](#) including a CloudFormation template
- Best for User uploaded image processing and editing.



Serverless Image Handler - Deploying the Solution

Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

Template is ready Use a sample template Create template in Designer

Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

Amazon S3 URL Upload a template file

Upload a template file
Choose file serverless-image-handler.template
JSON or YAML formatted file

S3 URL: <https://s3-us-west-1.amazonaws.com/cf-templates-onqnun6ec1tm-us-west-1/2019224nQ9-serverless-image-handler.template>

View in Designer

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

CORS Options

CorsEnabled
Would you like to enable Cross-Origin Resource Sharing (CORS) for the image handler API? Select 'Yes' if so.

No

CorsOrigin

If you selected 'Yes' above, please specify an origin value here. A wildcard (*) value will support any origin. We recommend specifying an origin (i.e. https://example.domain) to restrict cross-site access to your API.

*

Image Sources

SourceBuckets
(Required) List the buckets (comma-separated) within your account that contain original image files. If you plan to use Thumber or Custom image requests with this solution, the source bucket for those requests will be the first bucket listed in this field.

nebcode-example-cdn

Demo UI

DeployDemoUI
Would you like to deploy a demo UI to explore the features and capabilities of this solution? This will create an additional Amazon S3 bucket and Amazon CloudFront distribution in your account.

Yes

Event Logging

LogRetentionPeriod
This solution automatically logs events to Amazon CloudWatch. Select the amount of time for CloudWatch logs from this solution to be retained (in days).

1

Serverless Image Handler - Deploying the Solution

NebCode-ServerlessImageHandler																											
Stack info		Events	Resources																								
		Outputs	Parameters																								
		Template	Change sets																								
<h3>Outputs (5)</h3> <div><input type="text"/> Search outputs</div> <div><table><thead><tr><th>Key</th><th>Value</th><th>Description</th><th>Export name</th></tr></thead><tbody><tr><td>ApiEndpoint</td><td>https://dczn4uvjbk7m.cloudfront.net</td><td>Link to API endpoint for sending image requests to.</td><td>-</td></tr><tr><td>CorsEnabled</td><td>No</td><td>Indicates whether Cross-Origin Resource Sharing (CORS) has been enabled for the image handler API.</td><td>-</td></tr><tr><td>DemoUrl</td><td>https://d5y8353yx7sf5.cloudfront.net/index.html</td><td>Link to the demo user interface for the solution.</td><td>-</td></tr><tr><td>LogRetentionPeriod</td><td>1</td><td>Number of days for event logs from Lambda to be retained in CloudWatch.</td><td>-</td></tr><tr><td>SourceBuckets</td><td>nebcode-example-cdn</td><td>Amazon S3 bucket location containing original image files.</td><td>-</td></tr></tbody></table></div>				Key	Value	Description	Export name	ApiEndpoint	https://dczn4uvjbk7m.cloudfront.net	Link to API endpoint for sending image requests to.	-	CorsEnabled	No	Indicates whether Cross-Origin Resource Sharing (CORS) has been enabled for the image handler API.	-	DemoUrl	https://d5y8353yx7sf5.cloudfront.net/index.html	Link to the demo user interface for the solution.	-	LogRetentionPeriod	1	Number of days for event logs from Lambda to be retained in CloudWatch.	-	SourceBuckets	nebcode-example-cdn	Amazon S3 bucket location containing original image files.	-
Key	Value	Description	Export name																								
ApiEndpoint	https://dczn4uvjbk7m.cloudfront.net	Link to API endpoint for sending image requests to.	-																								
CorsEnabled	No	Indicates whether Cross-Origin Resource Sharing (CORS) has been enabled for the image handler API.	-																								
DemoUrl	https://d5y8353yx7sf5.cloudfront.net/index.html	Link to the demo user interface for the solution.	-																								
LogRetentionPeriod	1	Number of days for event logs from Lambda to be retained in CloudWatch.	-																								
SourceBuckets	nebcode-example-cdn	Amazon S3 bucket location containing original image files.	-																								

DEMO

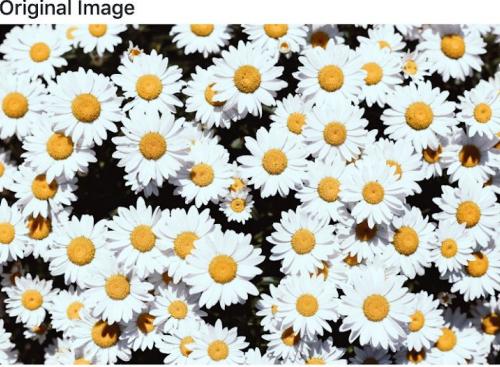
Serverless Image Handler Demo

Image Source

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

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

Import



[Having trouble?](#)

Editor

Width: 400 Height: 100 Resize Mode: cover

Fill Color: #FF0000 Background Color: #FF0000

Grayscale Negative RGB [?] 255, 0, 255
 Flip Flatten Normalize

Smart Cropping [?] Focus Index: 0 Crop Padding: 0

Reset **Preview**

Preview



Request Body: [?]

```
{ "bucket": "nebcode-example-cdn", "key": "01.jpg", "edits": { "resize": { "width": 400, "height": 100, "fit": "cover" } } }
```

Encoded URL: [?]

<https://dczn4uvjbk7m.cloudfront.net/eyJidWNrZXQiOiJuZWJbZ>

[Having trouble?](#)

Send a Request

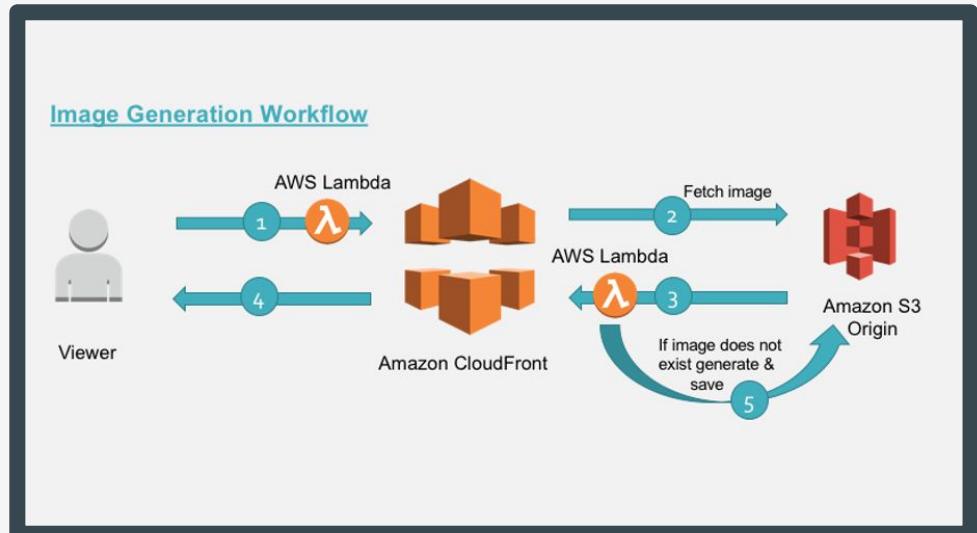
1. Find the Api Endpoint URL from your stack's output variables.
2. Create a JSON formatted request that includes your S3 Bucket name (bucket), the source image file name (key) and the desired Edits (edits).
3. Base64 encode the JSON request and append it to the Api Endpoint URL.

```
{  
  "bucket": "nebcode-example-cdn",  
  "key": "01.jpg",  
  "edits": {  
    "resize": {  
      "width": 400,  
      "height": 100,  
      "fit": "cover"  
    }  
  }  
}
```

<https://dczn4uvjbk7m.cloudfront.net/ewogICAgImJ1Y2tlCI6ICJuZWJjb2RlWV4YW1wbGUtY2RuliwKICAgICJrZXkiOiAiMDEuanBnliwKICAqICJIZGI0cyI6IHsKICAqICAqICAicmVzaXpljogewoqICAqICAqICAqICAid2IkdggiOiA0MDAsCiAgICAqICAqICAqICJoZWlnaHQiOjAxMDAsCiAgICAqICAqICAqICJmaXQiOiAiY292ZXliCiAgICAqICAqQogICAqfQp9Cg==>

Image Processing via Lambda@Edge

- Custom solution - more finite control
- Powered by [Sharp](#)
- Uses Lambda@Edge - Lambda functions invoked directly from the CloudFront distribution
- Ideal for a CDN - resize via URI
- Automated mobile optimizations



AWS CloudFront Triggers

- **CloudFront Viewer Request*** – The function executes when CloudFront receives a request from a viewer and before it checks to see whether the requested object is in the edge cache
- **CloudFront Origin Request** – The function executes only when CloudFront forwards a request to your origin. When the requested object is in the edge cache, the function doesn't execute.
- **CloudFront Origin Response** – The function executes after CloudFront receives a response from the origin and before it caches the object in the response.
- **CloudFront Viewer Response ***– The function executes before returning the requested object to the viewer. The function executes regardless of whether the object is already in the edge cache.

* triggers on every request

Example NodeJS Lambda Function

```
// index.js
const lib = require('npm-package');

exports.handler = (event, context, callback) => {
  let request = event.Records[0].cf.request;
  let response = event.Records[0].cf.response;

  // Business logic here

  // optional params: error, response
  callback(null, response);
};
```

```
// package.json
{
  "name": "origin-response",
  "version": "1.0.0",
  "main": "index.js",
  "dependencies": {
    "npm-package": "^1.0.0"
  }
}
```

Lambda@Edge Functions - Viewer Request

```
const querystring = require('querystring');

exports.handler = (event, context, callback) => {
    const request = event.Records[0].cf.request;

    // Skip non-image requests
    if (!request.uri.match(/.(gif|jpe?g|png|svg|tiff|i)/)) {
        callback(null, request);
        return;
    }

    const query = querystring.parse(request.querystring);
    if (!query.width && !query.height && !query.style) {
        callback(null, request);
        return;
    }

    const height = getDimension(query.height);
    const width = getDimension(query.width);
    const style = getStyle(query.style);

    // Skip if all options are defaults
    if (height === 0 && width === 0 && style === 'default') {
        callback(null, request);
        return;
    }
```

```
const uri = [];

/** 
 * /400w/
 * /400h400w/
 * /default/
 */
if (height || width) {
    let dimension = height ? height + 'h' : '';
    dimension = width ? dimension + width + 'w' : dimension;
    uri.push(dimension);
} else {
    uri.push('default');
}

/** 
 * /800w/bw/
 */
uri.push(style);

const image = request.uri.split('/').pop();
uri.push(image);

// Final modified url is of format /400h800w/bw/image.jpg
request.uri = '/' + uri.join('/');
callback(null, request);
};
```

Lambda@Edge Functions - Viewer Request

```
const querystring = require('querystring');

exports.handler = (event, context, callback) => {
  const request = event.Records[0].cf.request;

  // Skip non-image requests
  if (!request.uri.match(/\.(jpe?g|png|gif|webp|svg)$/)) {
    callback(null, request);
    return;
  }

  const query = querystring.parse(request.queryString);
  if (!query.width && !query.height && !query.style) {
    callback(null, request);
    return;
  }

  const height = get(query.height);
  const width = get(query.width);
  const style = get(query.style);

  // Skip if all options are defaults
  if (height === 0 && width === 0 && style === 'default') {
    callback(null, request);
    return;
  }
```

We've turned this:

http://domain.com/01.jpg?height=400&width=200&style=bw

Into this:

http://domain.com/400h200w/bw/01.jpg

```
const uri = [];

/**
 * /400w/
 */

// Final modified url is of format /400h800w/bw/image.jpg
request.uri = '/' + uri.join('/');
callback(null, request);
```

Lambda@Edge Functions - Origin Response

```
const aws = require('aws-sdk');
const mime = require('mime');
const sharp = require('sharp');

exports.handler = (event, context, callback) => {
  let response = event.Records[0].cf.response;

  // Image exists in S3 bucket already, we're already done.
  if (response.status !== '404') {
    callback(null, response);
    return;
  }

  const request = event.Records[0].cf.request;
  if (!request.uri.match(/^(gif|jpe?g|png|svg|tiff)/i)) {
    callback(null, response);
    return;
  }

  // Check if request matches expected input
  const match = request.uri.match(/^\d+h\d+w\d+h\d+w|default\b|bw\b|default\b|greyscale\b|grayscale\b\b|(.*).(gif|jpe?g|png|svg|tiff)/i);
  if (!match) {
    callback(null, response);
    return;
  }
```

Lambda@Edge Functions - Origin Response

```
// Perform the image manipulations
const dimensions = match[1].match(/(\d+h)?(\d+w)?(default)?/i);
const height = (dimensions[1] && !dimensions[3]) ? parseInt(dimensions[1]) : null;
const width = (dimensions[2] && !dimensions[3]) ? parseInt(dimensions[2]) : null;
const style = match[2];
const image = match[3] + ' ' + match[4];

let buffer = null;
const s3 = new aws.S3();
s3.getObject({
  Bucket: process.env.INPUT_BUCKET,
  Key: image

}).promise().then(data => {
  // Manipulate the image
  let newImage = sharp(data.Body);
  if (style === 'greyscale') {
    newImage = newImage.greyscale();
  }
  if (height || width) {
    newImage = newImage.resize({
      height: height,
      width: width,
      fit: 'inside'
    });
  }
  return newImage.toBuffer();
}).then(data => {
  buffer = data;
  // Save the new image
  return s3.putObject({
    Body: buffer,
    Bucket: process.env.INPUT_BUCKET,
    ContentType: mime.getType(image),
    CacheControl: 'max-age=31536000',
    Key: request.uri.substring(1),
  }).promise().catch(err => {
    console.log(`There was an error saving the manipulated image: ${err}`);
  });

}).then(() => {
  if (buffer) {
    response.status = 200;
    response.body = buffer.toString('base64');
    response.bodyEncoding = 'base64';
    response.headers['content-type'] = [{}];
    response.headers['Content-Type'] = [{}];
    response.headers['Content-Type'].value = mime.getType(image);
  }
  callback(null, response);
}).catch(err => {
  console.log(err);
  callback(null, response);
});
```

Create the CloudFront Distribution

Origin Settings

Origin Domain Name



Origin Path



Origin ID S3-nebcode-example-cdn



Restrict Bucket Access Yes
 No



Origin Access Identity Create a New Identity
 Use an Existing Identity

Comment



Grant Read Permissions on Bucket Yes, Update Bucket Policy
 No, I Will Update Permissions



Origin Custom Headers

Value

Query String Forwarding and Caching



Query String Whitelist

Valid characters: a-z, A-Z, 0-9, - . _ * + %

[Learn More](#)

Deploying the Lambda@Edge Functions - IAM Role

Select type of trusted entity

AWS service EC2, Lambda and others

Another AWS account Belonging to you or 3rd party

Web identity Cognito or any OpenID provider

SAML 2.0 federation Your corporate directory

Allows AWS services to perform actions on your behalf. [Learn more](#)

Choose the service that will use this role

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

API Gateway	Comprehend	ElastiCache	Lex	SMS
AWS Backup	Config	Elastic Beanstalk	License Manager	SNS
AWS Support	Connect	Elastic Container Service	Machine Learning	SWF

▼ Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy

Filter policies ▾ Showing 1 result

	Policy name	Used as	Description
<input checked="" type="checkbox"/>	AWSLambdaExecute	None	Provides Put, Get access to S3 and full a...

▼ Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy

Filter policies ▾ Showing 1 result

	Policy name	Used as	Description
<input checked="" type="checkbox"/>	AmazonS3FullAccess	Permissions policy (1)	Provides full access to all buckets via th...

Deploying the Lambda@Edge Functions - IAM Role

Review

Provide the required information below and review this role before you create it.

Select type of trusted entity

AWS service EC2, Lambda and others

Allows AWS services to perform actions on your behalf.

Choose the service that will be able to assume this role

EC2 Allows EC2 instances to call AWS services

Lambda Allows Lambda functions to call AWS services

API Gateway Comprehensive API management

AWS Backup Config

AWS Support Connect

Role name* nebcode-lambda-s3-role

Use alphanumeric and '+,-,_' characters. Maximum 64 characters.

Role description Nebcode-demo

Maximum 1000 characters. Use alphanumeric and '+,-,_' characters.

Trusted entities AWS service: lambda.amazonaws.com

Policies

- AmazonS3FullAccess
- AWSLambdaExecute

Permissions boundary Permissions boundary is not set

Deploying the Lambda@Edge Functions - Viewer Request

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
▼ Choose or create an execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to [Use an existing role](#)

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role
[View the example-cdn-image-processing-execution-role role](#) on the IAM console.

Function code [Info](#)

Code entry type Runtime Handler [Info](#)

Function package
 For files larger than 10 MB, consider uploading using Amazon S3.

Deploying the Lambda@Edge Functions - Origin Response

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.

Permissions [Info](#)
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.
▼ Choose or create an execution role

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
 [C](#)
[View the example-cdn-image-processing-execution-role role](#) on the IAM console.

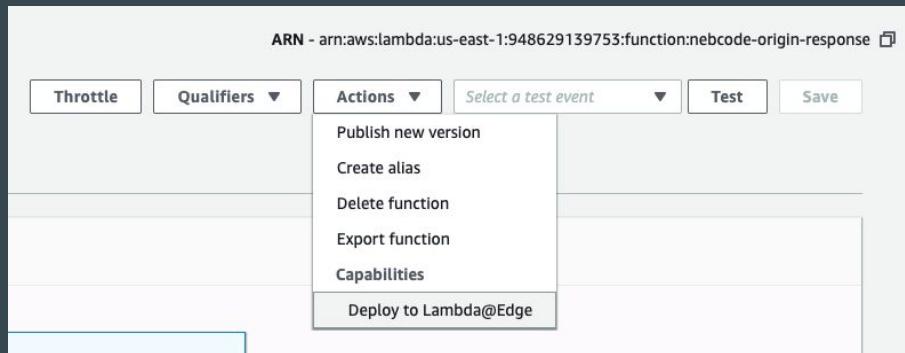
Basic settings

Description

Memory (MB) [Info](#)
Your function is allocated CPU proportional to the memory configured.
 512 MB

Timeout [Info](#)
 min sec

Associate the Lambda Functions with CloudFront



Deploy to Lambda@Edge

Configure CloudFront trigger

Distribution
The CloudFront distribution that will send events to your Lambda function.
E2V9XBL49XM1E

Cache behavior
Choose the cache behavior you would like this Lambda function to be associated with.
*

CloudFront event
Choose one CloudFront event to listen for.
Origin response

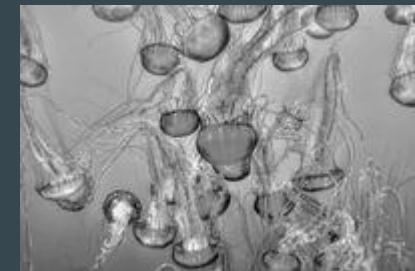
Confirm deploy to Lambda@Edge

I acknowledge that on deploy a new version of this function will be published with the above trigger and replicated across all available AWS regions.

Lambda will add the necessary permissions for Amazon CloudFront to invoke your Lambda function from this trigger. [Learn more](#) about the Lambda permissions model.

[Cancel](#) [Deploy](#)

DEMO



Before and After - S3

<input type="checkbox"/> Name ▾	Last modified ▾	Size ▾
<input type="checkbox"/>  01.jpg	Aug 9, 2019 2:01:41 PM GMT-0500	2.4 MB
<input type="checkbox"/>  02.jpg	Aug 9, 2019 2:00:08 PM GMT-0500	2.4 MB
<input type="checkbox"/>  03.jpg	Aug 9, 2019 2:01:57 PM GMT-0500	3.5 MB
<input type="checkbox"/>  04.jpg	Aug 9, 2019 2:01:56 PM GMT-0500	3.6 MB
<input type="checkbox"/>  05.jpg	Aug 9, 2019 2:00:02 PM GMT-0500	1.3 MB
<input type="checkbox"/>  06.jpg	Aug 9, 2019 2:00:03 PM GMT-0500	1.8 MB

<input type="checkbox"/> Name ▾	Last modified ▾	Size ▾
<input type="checkbox"/>  200h	--	--
<input type="checkbox"/>  200h200w	--	--
<input type="checkbox"/>  01.jpg	Aug 9, 2019 2:01:41 PM GMT-0500	2.4 MB
<input type="checkbox"/>  02.jpg	Aug 9, 2019 2:00:08 PM GMT-0500	2.4 MB
<input type="checkbox"/>  03.jpg	Aug 9, 2019 2:01:57 PM GMT-0500	3.5 MB
<input type="checkbox"/>  04.jpg	Aug 9, 2019 2:01:56 PM GMT-0500	3.6 MB
<input type="checkbox"/>  05.jpg	Aug 9, 2019 2:00:02 PM GMT-0500	1.3 MB
<input type="checkbox"/>  06.jpg	Aug 9, 2019 2:00:03 PM GMT-0500	1.8 MB

Improvements

1. Create specific parameters
 - a. dimensions=twitter-card
 - b. device-type=mobile
2. Add base64 encoded JSON object query string parameter
 - a. Easier parameter handling
 - b. More configurable
 - c. Harder to cache
 - d. Harder to use non-programmatically
3. Generate UUIDs for image keys
4. Change URI Cache mechanism to UUIDs

Questions?

cj.ohara@firespring.com

<https://github.com/cjohara/aws-serverless-image-processing>