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.

Practical reasons you may need to process images for you 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

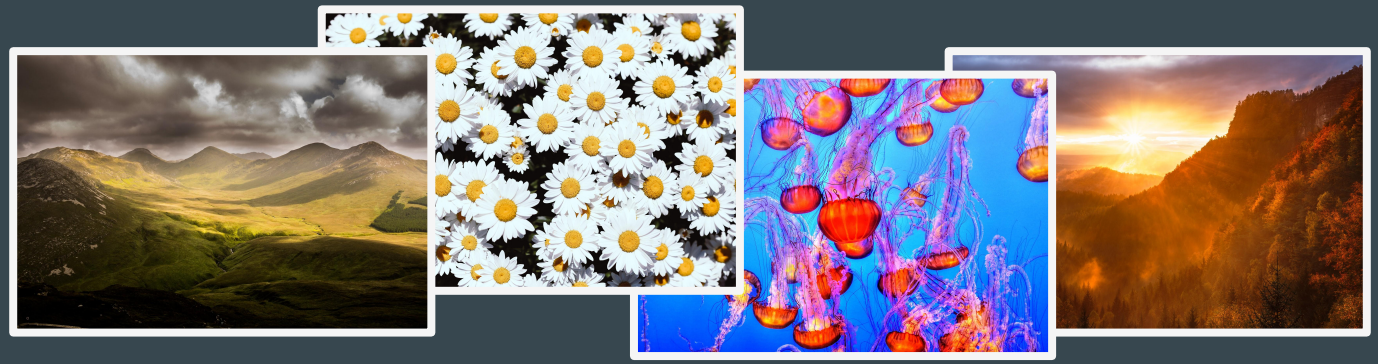
Why Serverless?

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.

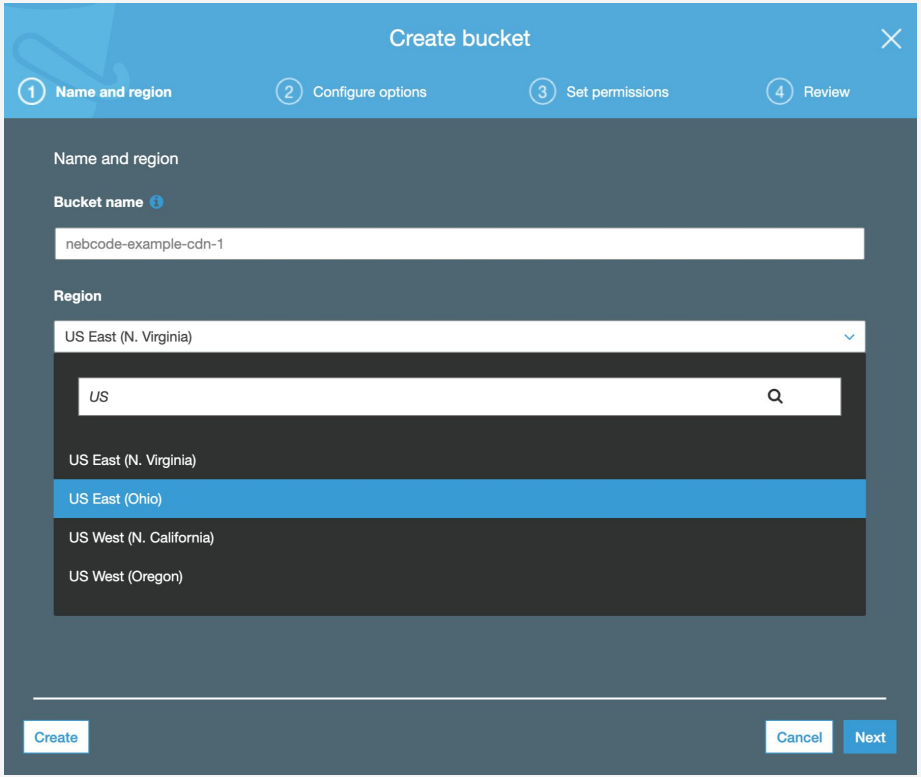
- No managing servers - focus on the code
- Auto-scaling for high availability
- Pay for what you use - it’s cheap.
- It’s fun to explore technologies outside of the LAMP stack

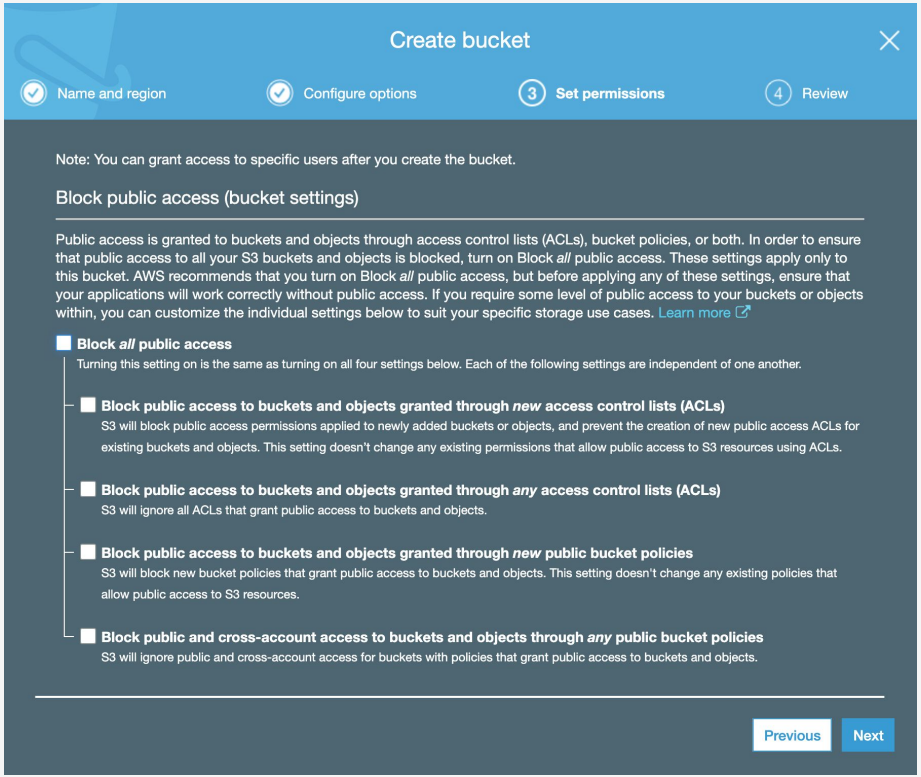
Setup / Prerequisites - S3 Bucket

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



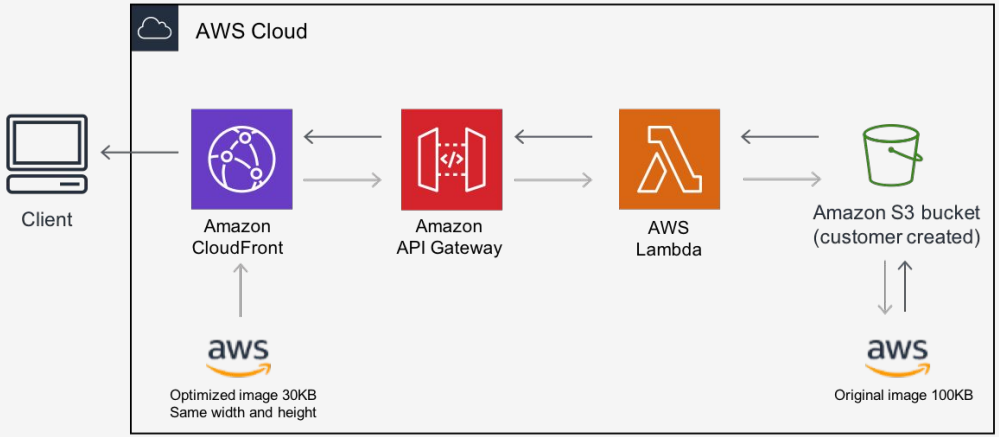
Create your AWS S3 Bucket



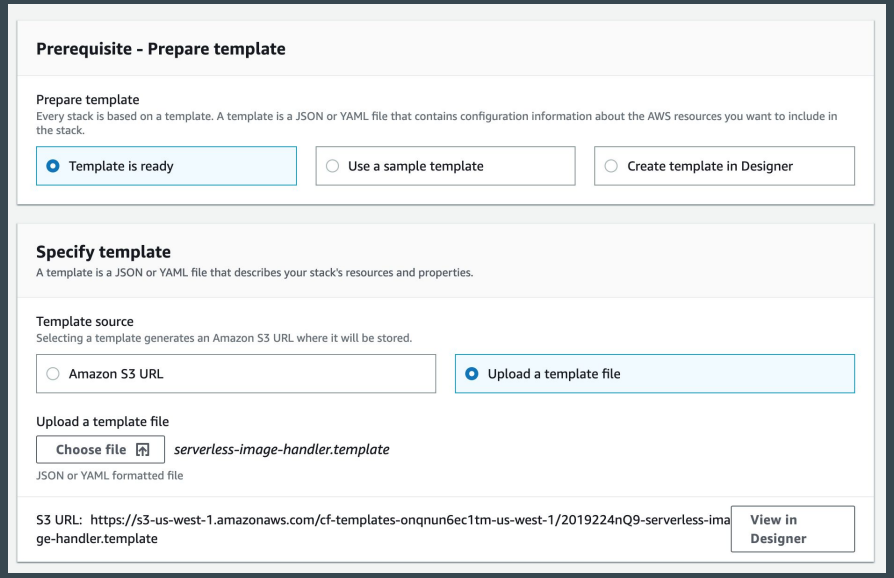


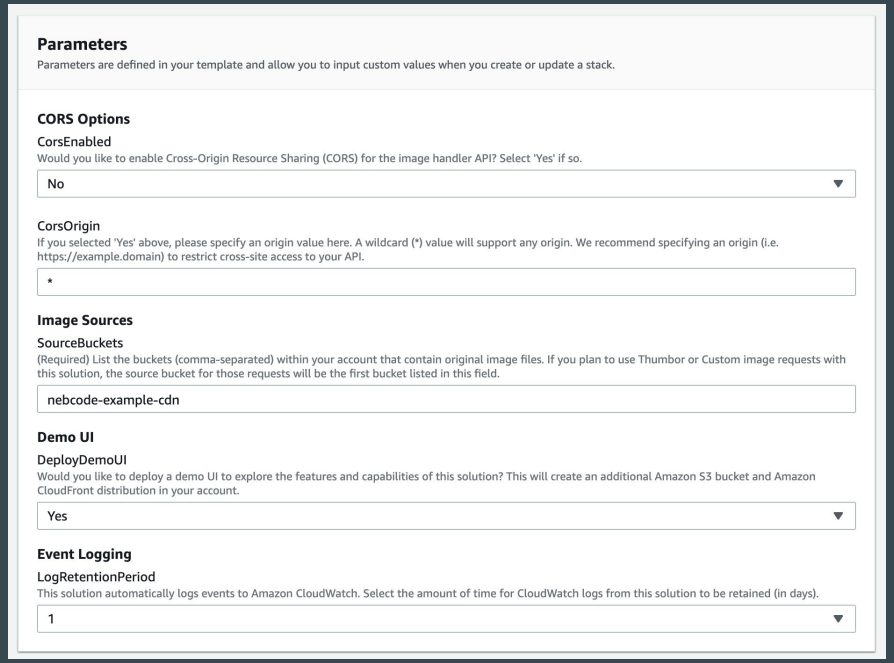
Serverless Image Handler - AWS Solution

- 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



Serverless Image Handler - Deploying the Solution





DEMO

Serverless Image Handler *Demo*

Image Source

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

nebcode-example-cdn

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

01.jpg

Import

Original Image

Having trouble?

Editor

Width: 400

Height: 100

Resize Mode: cover

Fill Color [?]: #FF0000

Background Color [?]: #FF0000

- [ ] Grayscale
- [ ] Flip
- [ ] Flop
- [ ] Negative
- [ ] Flatten
- [ ] Normalize

RGB [?]: 255, 0, 255

- [ ] 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/eyJidWNrZXQiOiJuZWJjb2

Having trouble?

Send a Request

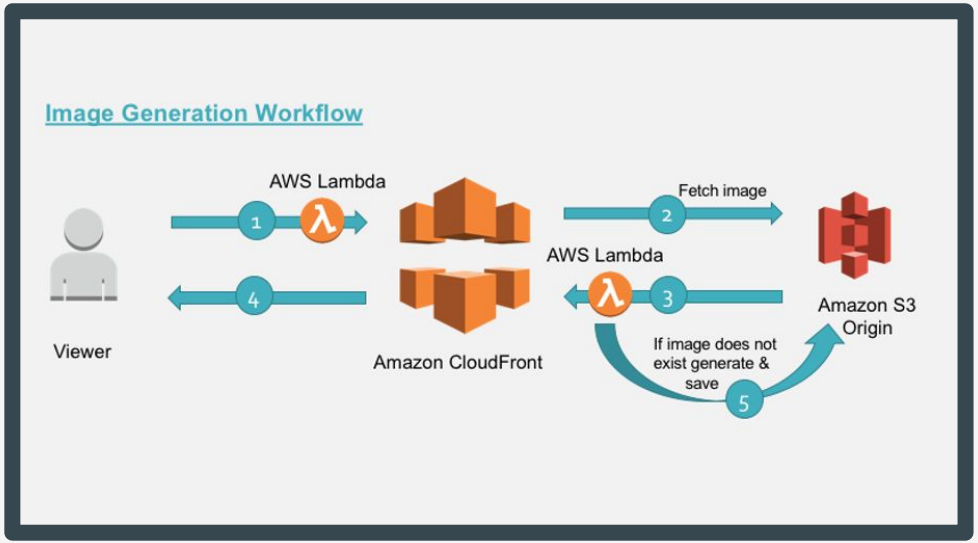
1. Find the Api Endpoint URL from your stack's output variables.
2. Create a JSON formatted request that includes your S3 Bucket name (bucket), the source image file name (key) and the desired Edits (edits).
3. Base64 encode the JSON request and append it to the Api Endpoint URL.

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

https://dczn4uvjbk7m.cloudfront.net/ewogICAgImJ1Y2tldCI6ICJuZWJjb2RlLWV4YW1wbGUtY2RuIiwKICAgICJrZXkiOiAiMDEuanBnIiwKICAgICJlZGl0cyI6IHsKICAgICAgICAicmVzaXplIjogewogICAgICAgICAgICAid2lkdGgiOiA0MDAsCiAgICAgICAgICAgICJoZWlnaHQiOiAxMDAsCiAgICAgICAgICAgICJmaXQiOiAiY292ZXIiCiAgICAgICAgfQogICAgfQp9Cg==

Image Processing via Lambda@Edge

- Custom solution - more finite control
- Powered by Sharp
- Uses Lambda@Edge - Lambda functions invoked directly from the CloudFront distribution



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
  if (!(request.uri.m
    callback(null, re
    return;
  }

  const query = qu
  if (!query.width &&
    callback(null, re
    return;
  }

  const height = ge
  const width = get
  const style = get

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

```
  const uri = [];

  /**
   * /400w/
```

```
ension;
```

```
  uri.push(image);

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

We've turned this:

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

Into this:

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

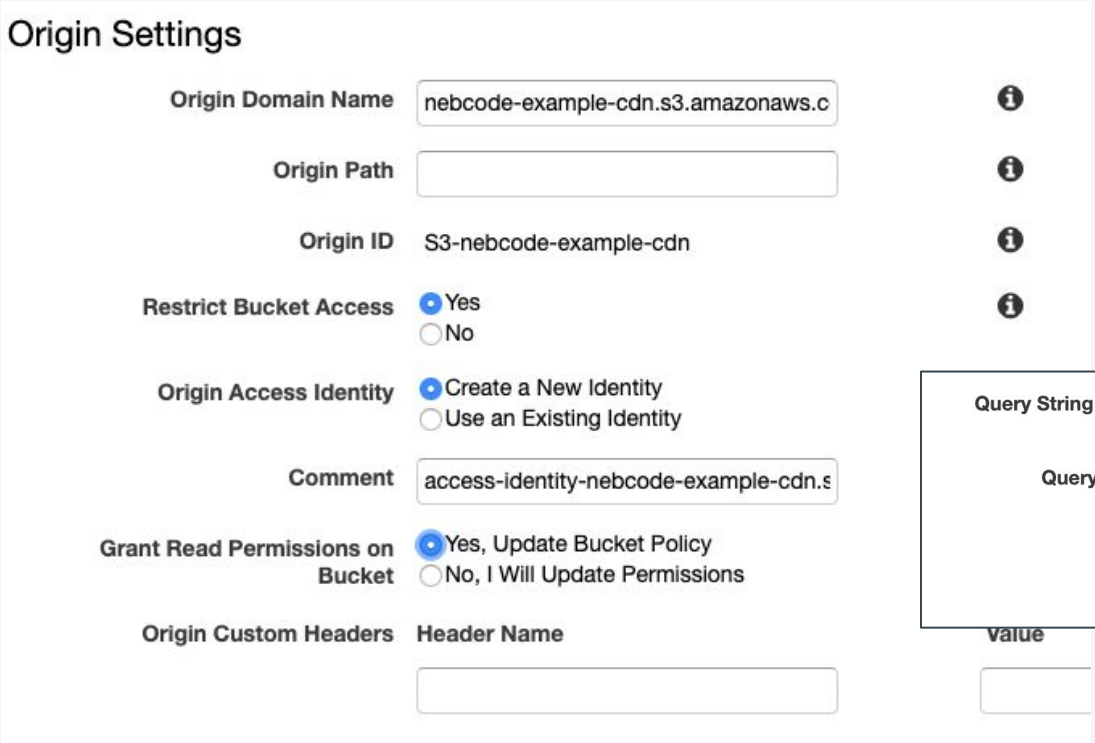
Lambda@Edge Functions - Origin Response

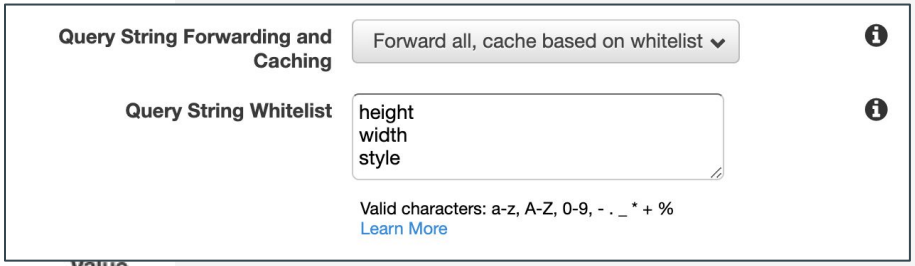
```
const aws = require('aws-sdk');
const mime = require('mime');
const sharp = require('sharp');

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)\/(bw|default|greyscale|grayscale)\/(.*).(gif|jpe?g|png|svg|tiff)/i);
  if (!match) {
    callback(null, response);
    return;
  }
```

Lambda@Edge Functions - Origin Response

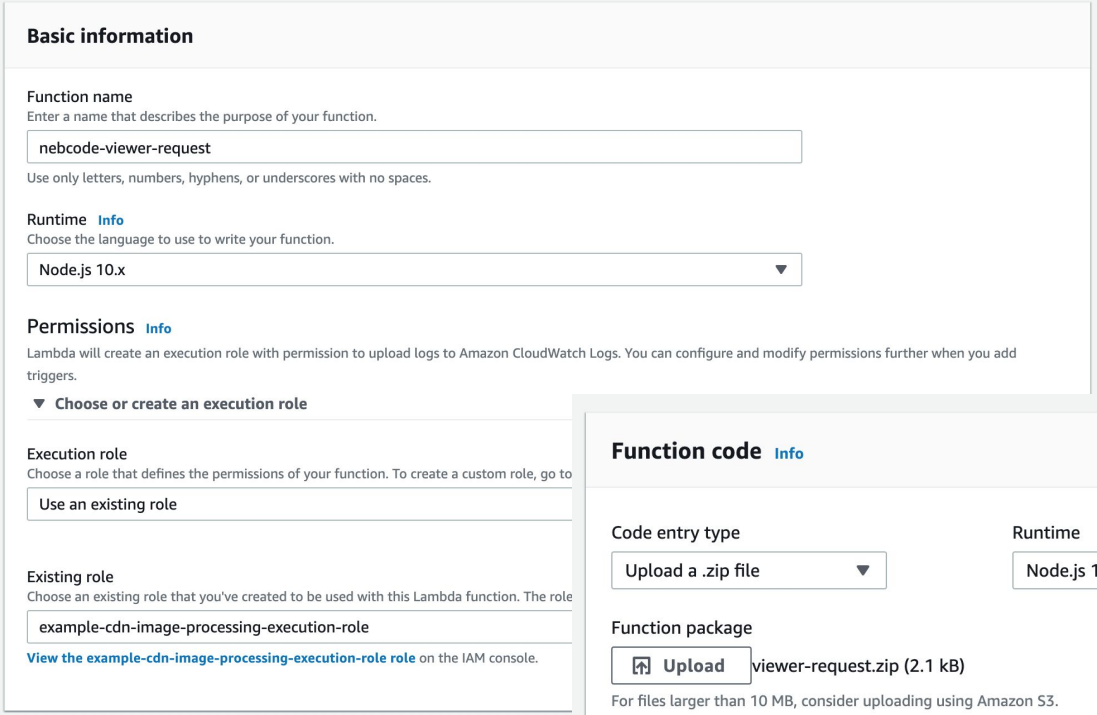
```
 // Perform the image manipulations
const dimensions = match[1].match(/(\d+h)?(\d+w)?(default)?/i);
const height = (dimensions[1] && !dimensions[3]) ? parseInt(dimensions[1]) : null;
const width = (dimensions[2] && !dimensions[3]) ? parseInt(dimensions[2]) : null;
const style = match[2];
const image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

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

 }).catch(err => {
  console.log(err);
  callback(null, response);
 });
};
```

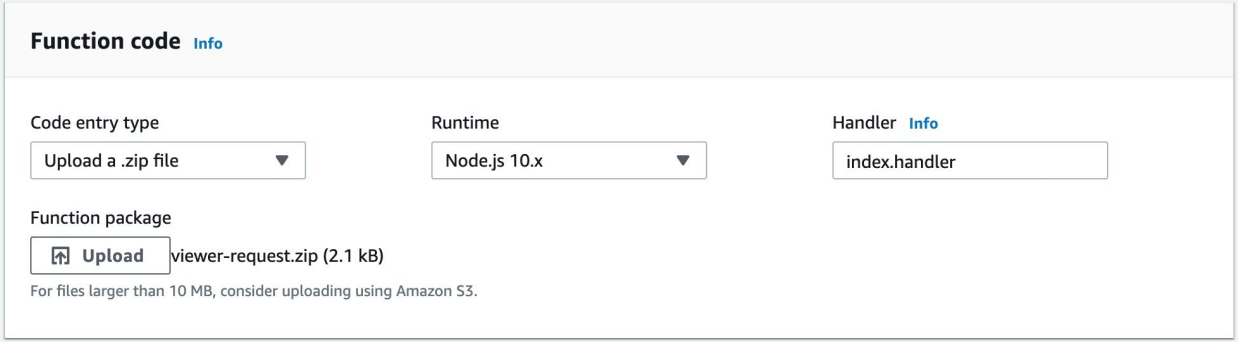
Create the CloudFront Distribution



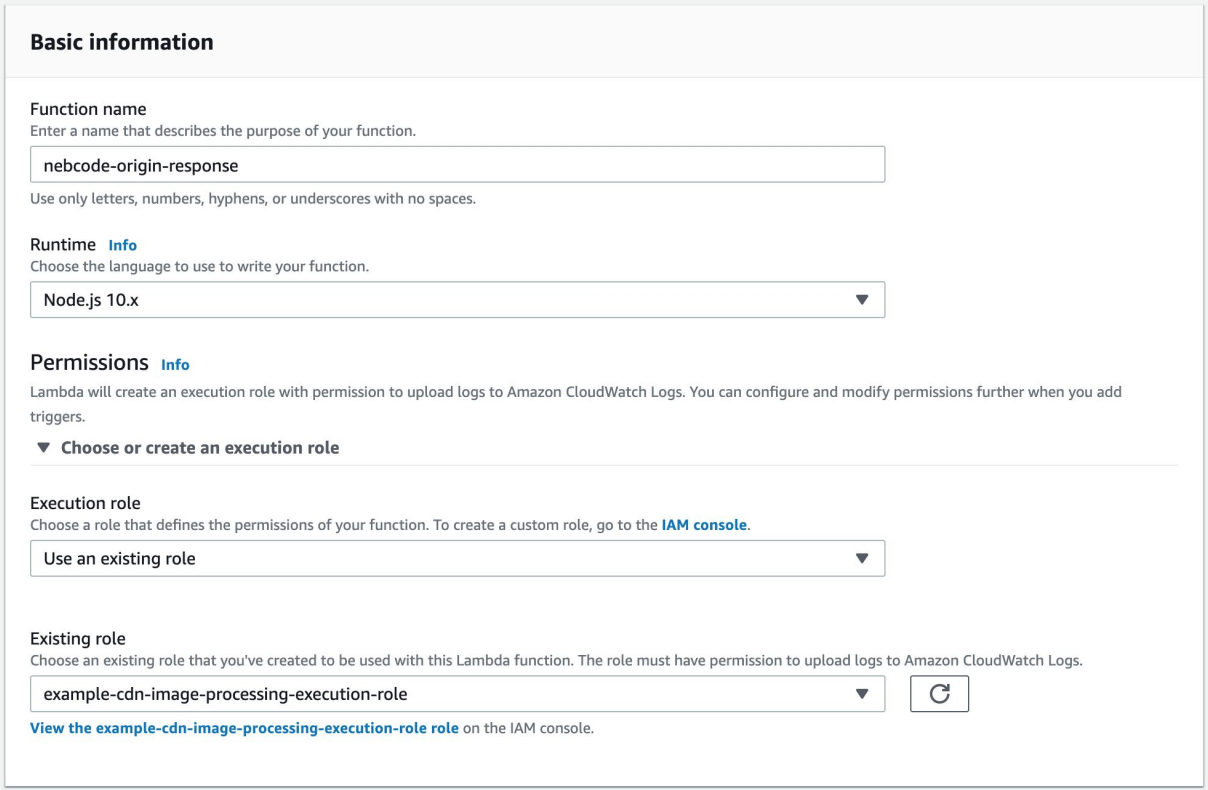


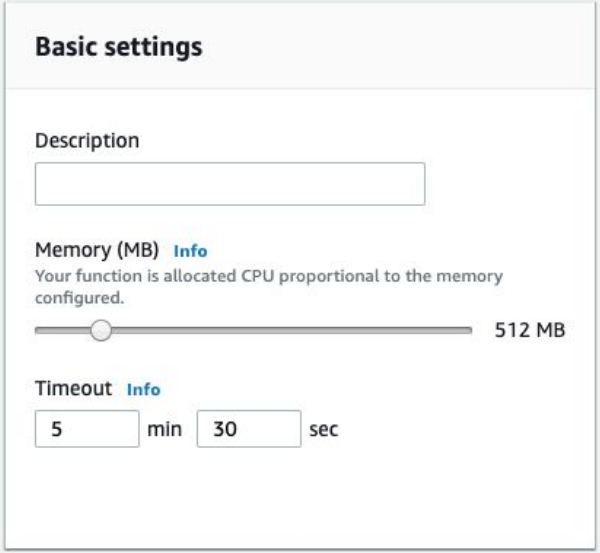
Deploying the Lambda@Edge Functions - Viewer Request



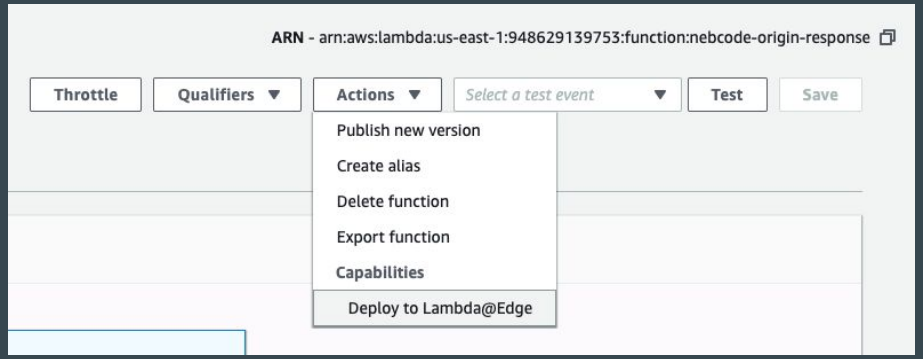


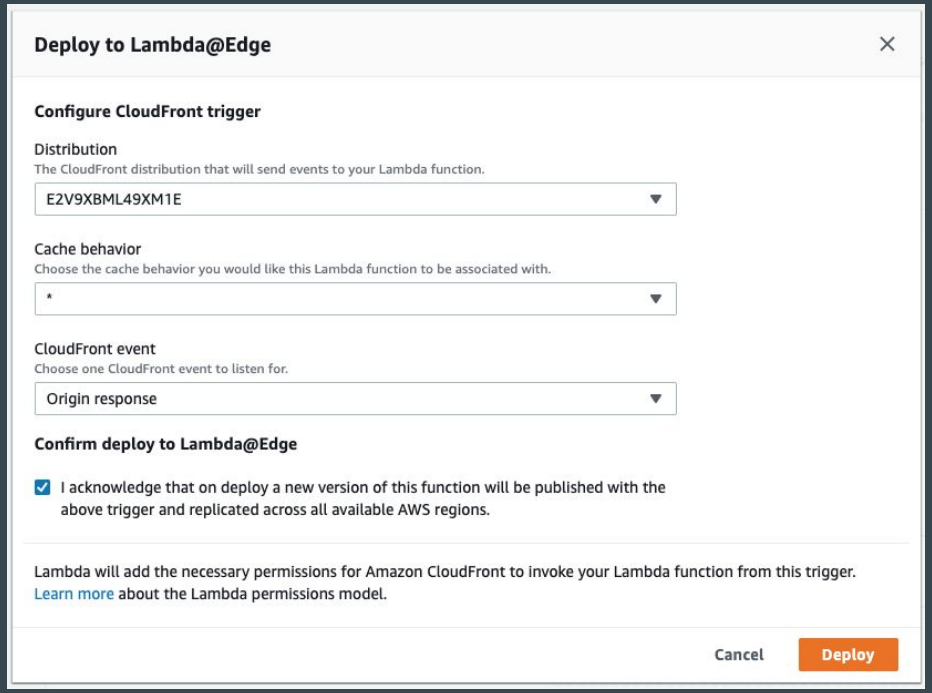
Deploying the Lambda@Edge Functions - Origin Response



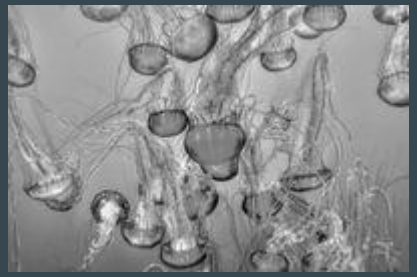


Associate the Lambda Functions with CloudFront

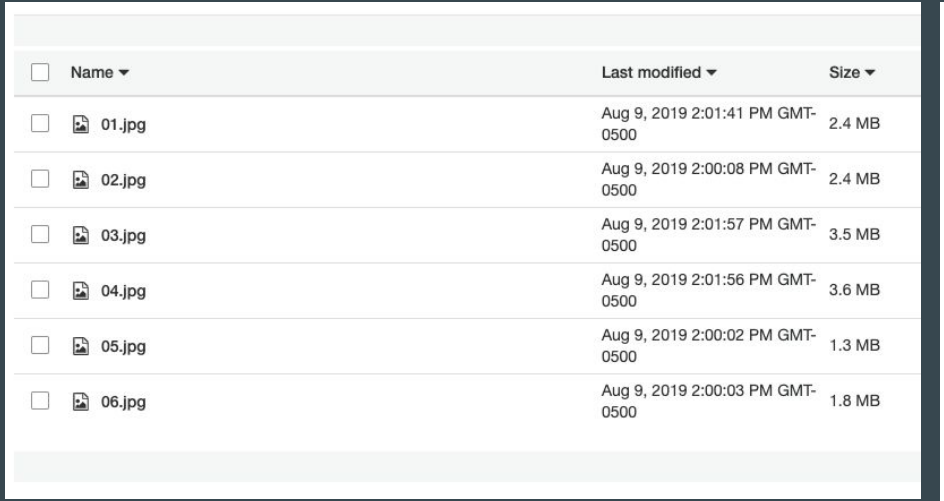




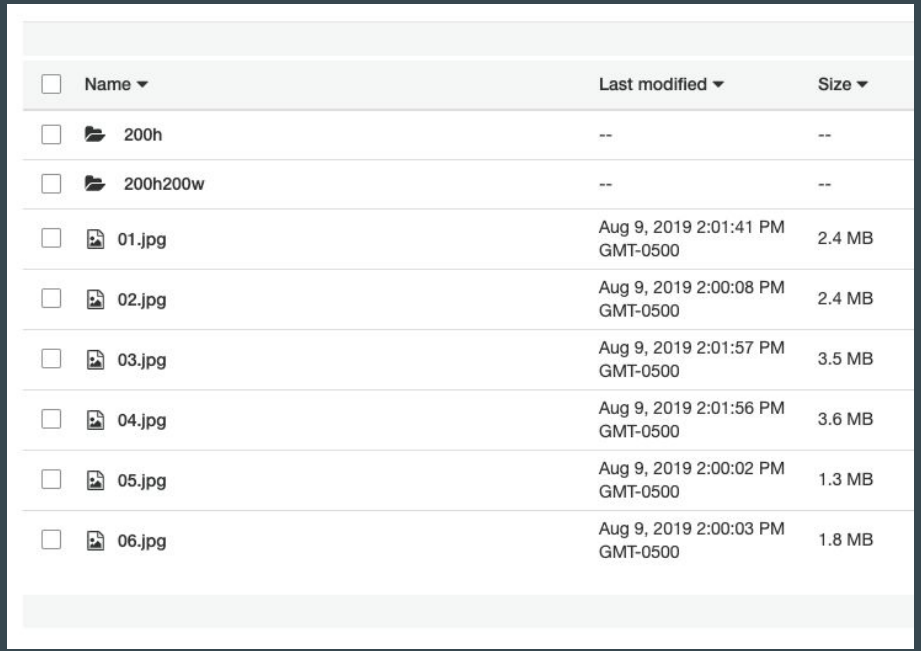
DEMO



Before and After - S3



Name	Last modified	Size
01.jpg	Aug 9, 2019 2:01:41 PM GMT-0500	2.4 MB
02.jpg	Aug 9, 2019 2:00:08 PM GMT-0500	2.4 MB
03.jpg	Aug 9, 2019 2:01:57 PM GMT-0500	3.5 MB
04.jpg	Aug 9, 2019 2:01:56 PM GMT-0500	3.6 MB
05.jpg	Aug 9, 2019 2:00:02 PM GMT-0500	1.3 MB
06.jpg	Aug 9, 2019 2:00:03 PM GMT-0500	1.8 MB



Name	Last modified	Size
200h	--	--
200h200w	--	--
01.jpg	Aug 9, 2019 2:01:41 PM GMT-0500	2.4 MB
02.jpg	Aug 9, 2019 2:00:08 PM GMT-0500	2.4 MB
03.jpg	Aug 9, 2019 2:01:57 PM GMT-0500	3.5 MB
04.jpg	Aug 9, 2019 2:01:56 PM GMT-0500	3.6 MB
05.jpg	Aug 9, 2019 2:00:02 PM GMT-0500	1.3 MB
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