Lomba Kompetensi Siswa Sekolah Menengah Kejuruan Tingkat Provinsi Jawa Barat Tahun 2023

Modul 2 - Lambda for Image Optimization

May 7, 2023

Bidang Lomba Cloud Computing

1 Overview

Optimizing images on a website is a critical aspect of enhancing user experience, reducing delivery costs, and boosting search engine rankings. Given that images are typically the heaviest components of a web page in terms of both bytes and number of HTTP requests, it is important to implement effective optimization strategies. Optimizing images can also enhance a website's search engine ranking, as demonstrated by Google's Largest Contentful Paint metric. By implementing this solution, websites can achieve faster load times, lower costs, and improved visibility in search engine results. You have been given a task to deploy an image optimizer solution on AWS. The API is built using AWS serverless components such as Amazon CloudFront, Amazon S3, and AWS Lambda. The solution requires to use a custom domain and an SSL.

2 General Rules

1. Failure to comply with the rules will result in immediate disqualification.
2. You have 3 hours to finish the tasks.
3. You may not open any website unless otherwise specified in section 7 and you may open the control panel of your domain provider to update the nameserver to Route 53.
4. You may use AWS Console and AWS CLI to deploy the solutions. You may not use SAM, CloudFormation or CDK.
5. Between and after the event, you may not access your account. Any activity on AWS during this period is not allowed.
6. Resources should be given the least privilege IAM permissions. For example if a lambda function needs a put object permission to a bucket, then only give put object permission, do not give get object permission.
7. During the event, multiple login is not permitted.
8. If you have any question, do not hesitate to ask.

3 Architecture

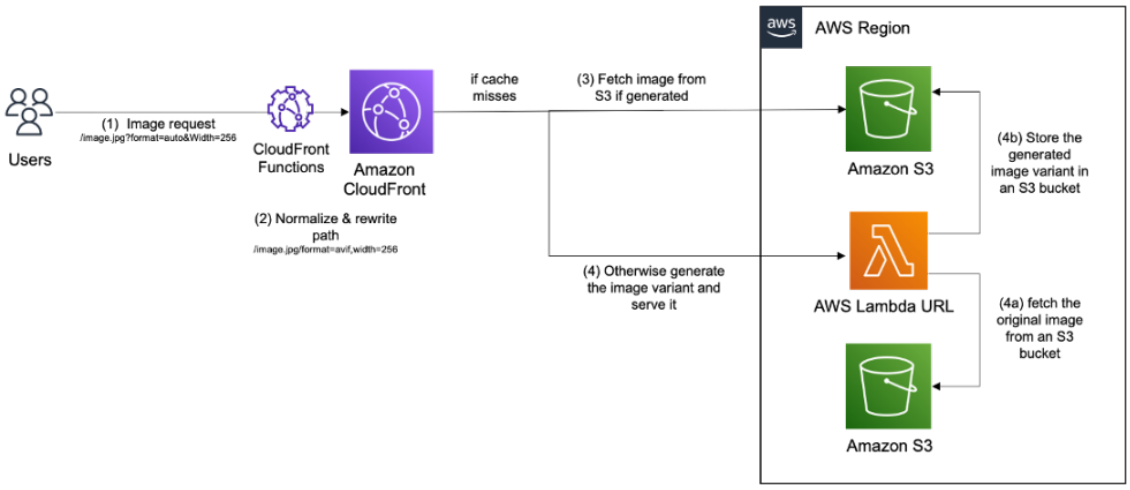

Figure 1: Architecture Diagram

1. A user requests an image with certain transformations, such as encoding and size, these transformations are encoded in the URL as query parameters. For instance, an example URL may look like this: https://example.com/images/01.jpg?format=webp&width=200.

2. The image request is initially handled by the nearest CloudFront edge location, ensuring optimal performance. A CloudFront Function is then executed on the viewer request event to rewrite the request URL before it is passed upstream. This function is designed for CDN customizations that require lightweight JavaScript functions capable of handling high-scale, latency-sensitive tasks. In our architecture, the URL is rewritten to validate the requested transformations, normalize the URL by ordering transformations and converting them to lower case, thus increasing the cache hit ratio. Additionally, when an automatic transformation is requested, the function determines the best one to apply. For example, if the user specifies the "format=auto" directive for the most optimized image format (JPEG, WebP, or AVIF), the CloudFront Function selects the optimal format based on the Accept header present in the request.

3. When a user requests an image that is already cached in CloudFront, the image is immediately returned from the CloudFront cache, resulting in a cache hit. However, if the image is not present in the CloudFront cache, the request is forwarded to an S3 bucket created specifically for storing transformed images. If the requested image has already been transformed and stored in S3, it is served and cached in CloudFront, with no need for further transformation.

4. If the requested image does not exist in the bucket for the transformed images, S3 will respond with a 403 error code. CloudFront's Origin Failover feature detects this error and retries the same URL, but this time using the secondary origin based on the Lambda function URL. The Lambda function is then invoked and downloads the original image from a separate S3 bucket where the original images are stored. Using the Sharp library, the Lambda function transforms the image, stores it in S3, and serves it through CloudFront, where it is cached for future requests.

4 Information

1. The repository URL for the required source code to deploy this solution is https://github.com/kensasongko/lksccjabar2023modul2_aplikasi

2. This solution must be deployed in **us-east-1 (N. Virginia) region**. Deploying in another region will result in a major point reduction.

5 Task

Your task is to create the solution from the section 3.

1. Create 2 S3 buckets, one to store the original images and another one to store the transformed images.

   - Original image bucket:
     - Add a tag to the bucket: Key=LKS-ID, Value=MODUL2-ORIGINAL
   - Transformed image bucket:
     - Create a lifecycle policy to remove images after 30 days.
     - Add a tag to the bucket: Key=LKS-ID, Value=MODUL2-TRANSFORMED
   - Both buckets should have the following configuration:
     - Enable block all public access.
     - Access: Bucket and objects not public

2. Deploy lambda:

   - Install Sharp

- Create a development package from the image-processing directory.
- Create and deploy the lambda function with the following configuration:
  - Lambda runtime: nodejs16.x
  - Memory size: 256 MB
  - Give the function policies as stated in the README.md.
  - Add environment variables as stated in the README.md.
  - Create function URL with Auth Type = None.
  - Add a tag to the function: Key=LKS-ID, Value=MODUL2-IMAGE-HANDLER
- Check README.md for more instruction.

3. Create a CloudFront distribution with the following configuration:
   - Price class: Use all edge locations (best performance)
   - Supported HTTP versions: HTTP/2
   - Viewer protocol policy: Redirect HTTP to HTTPS
   - Security policy: TLSv1.2_2021
   - Standard logging: Off
   - Cookie logging: Off
   - Alternate domain name (CNAME): https://modul2.[YOUR_DOMAIN]
   - Cache Policy:
     - TTL Settings - Minimum TTL (seconds): 10
     - TTL Settings - Maximum TTL (seconds): 31536000
     - TTL Settings - Default TTL (seconds): 86400
     - Cache Keys - Headers: None
     - Cache Keys - Cookies: None
     - Cache Keys - Query strings: All
     - Compression support - Gzip: Disabled
     - Compression support - Brotli: Disabled
   - Response headers policy:
     - Cross-origin resource sharing (CORS) - Access-Control-Allow-Credentials: False
     - Cross-origin resource sharing (CORS) - Access-Control-Allow-Headers: *
     - Cross-origin resource sharing (CORS) - Access-Control-Allow-Methods: GET
     - Cross-origin resource sharing (CORS) - Access-Control-Allow-Origin: *
     - Cross-origin resource sharing (CORS) - Access-Control-Expose-Headers: -
     - Cross-origin resource sharing (CORS) - Access-Control-Max-Age (seconds): 600
   - Create 2 origin with origin failover (failover criteria: 403), as explained in section 3. The first origin is the transformed bucket, the second origin is the lambda function URL. Add a custom header to the second origin, the name of the custom header is x-origin-secret-headerx and, the value is the secrey key you have previously added to the lambda function.
   - Add tag: Key=LKS-ID, Value=MODUL2
4. Create a CloudFront function from the url-rewrite directory and add it to Viewer request function of the CloudFront distribution.

6 How to test your deployment

1. Upload an image to the original image bucket (this example uses test1.png as the test image)
2. Using your browser open http://modul2.[YOUR_DOMAIN]/test1.png.
3. Check whether /test1.png/original exists in your transformed bucket image.
4. Using your browser open http://modul2.[YOUR_DOMAIN]/test1.png?width=200.
5. Check whether /test1.png/width=200 exists in your transformed bucket image.

7 References

- Lambda documentation
- API Gateway documentation
- Route 53 documentation
- Certificate Manager documentation
- Sharp documentation

Good luck!