Managing Image Content for a Static Website

Business Case Scenario

A small-scale fashion retailer runs a static website using AWS cloud platform. They maintain their content on S3, which consists of image files of their apparels and merchandise. When an image of each of their product is prepared, they need to:

a) Copy it to the S3 bucket being used as the production website and

b) Generate a smaller version of it to use as a thumbnail picture on the web pages.

Problem Statement

As the number of images for their website is increasing, they are looking for an automated solution for the above two tasks, as and when an image file is uploaded to S3.

You need to decide on an automated solution and implement it on AWS platform.

A set of their web page content which includes image files is given in the zip as a sample input for you to use for testing the solution.

You need to determine which services of AWS can be used for solving this kind of business problems, deploy them and demonstrate them with the sample data.

List of Resources and References

1. LMS Assignments on:
    - AWS S3
    - AWS Lambda services
    - CloudWatch
2. AWS user guides and developer guides for the above services
3. Webinar sessions on the above topics

Implementation Guide

Steps Involved:

Below are the steps to be followed. Details of each task listed in these steps are also given.

To begin with, download all the relevant files from the MiniProject2 folder of the shared drive.

URL:
https://drive.google.com/drive/folders/1Km1xqW4wlpuIkCHu7XrPZXR68KGx24AK?usp=sharing

Unzip the zip files – *Webpages.zip* and *AdditionalImages.zip*

The steps involved are:

1. Create the Amazon S3 Buckets and configure one of them as static website
2. Create a role in IAM services for the Lambda functions to access S3 buckets for read and write files besides add logs in CloudWatch
3. Create and deploy a Lambda function which will be triggered whenever an image is uploaded to the first bucket. It creates thumbnail image in the appropriate folder on static website bucket and also copies the original larger image to its folder on the static website bucket
4. Test the Lambda function by uploading an image to S3 bucket
5. Monitor and check the logs
6. Upload the rest of the images which will run the Lambda functions and add the remaining images to the website
7. Troubleshooting and Debugging

Under S3 Service

1. Create the Amazon S3 Buckets

- You need to create two Amazon S3 buckets -- one as a staging area for initial upload of the images and the other for the static website (live production website)
- For the first one – Click on *Create bucket* and configure.
    - Bucket name: Give a unique name such as *<your name>-miniproj2*
    - Make a note of the bucket name in any text editor
    - Click on *Create bucket*
- For the second one – Click on *Create bucket* and configure as below:
    - Bucket name: Copy & paste the name of the first bucket and append *-prod*
    - Click on *Create bucket*
- Now in your *Bucket Name* list you should have two buckets. For example:
    - *raj-miniproj2*
    - *raj-miniproj2-prod*
- Configure the second (*-prod*) bucket for website static hosting
    - Select the second (*-prod*) bucket from the *Bucket Name* list and choose *Properties* tab
    - Select *Static website hosting* option
    - Select *Use this bucket to host a website*
    - Enter the name of the index document as *open_close.html*
    - Enter the name of the error document as *error.html*
    - Make a note of the *Endpoint* under *Static website hosting*
    - Click on *Save*
- Upload index html and website content
    - In the *Buckets* list select the second (*-prod*) bucket
    - On *Overview* tab in the bucket click on *Upload*
    - Drag and drop the files and folders from Webpages directory that was created on your local system/laptop when you unzipped *Webpages.zip* file
    - Click on *Upload* to complete the uploading
- Edit public access settings
    - By default public access is blocked for an S3 bucket

- Select *Permissions* tab and choose *Edit*
- Uncheck the checkbox *Block* all *public access* and then *Save*
- In the confirmation box enter *confirm* and click on *Confirm*

- Attach a bucket policy
  - You need to add a bucket policy to provide public access to all the content of the bucket
  - From *Buckets* – select the second (-*prod*) bucket by clicking on the bucket name
  - Select *Permissions* → *Bucket Policy* options
  - Copy the bucket policy below paste it in the *Bucket policy editor*
  - Enter your second i.e. -*prod* bucket name in Resource field and click on *Save*

➔

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PublicReadGetObject",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::<Second ie -prod
bucketname>/*"
            ]
        }
    ]
}
```

🡐

  - A warning appears indicating that the bucket has public access. In *Bucket Policy*, a *Public* label appears

- Test your static website endpoint
  - From *Buckets* – select the second (-*prod*) bucket by clicking on the bucket name
  - Select *Properties* → *Static website hosting*
  - Click on website endpoint link. In a new browser tab the web page *open_close.html* should be displayed
  - This indicates the static website is correctly configured

Under IAM Service

2. Create a role with access to S3 buckets for read and write files besides add logs in CloudWatch

- Select *Create role* and give the following details to set up the role
    - Name: *lambda-s3-role*
    - Select *Use case* as *Lambda*
    - Click on *Permissions* and in *filter policies* textbox, filter for *AWSLambdaBasic* and select it
    - Again in *filter policies* textbox, enter *S3full* and select S3 full access role as well
    - Ensure that the name *lambda-s3-role* is entered
    - Click on *Create Role*

Under Lambda Service

3. Create the Lambda function for thumbnail pictures first

- Click on *Create function* and select *Author from scratch*
- Give *Function name* as *Create-Thumbnail*
- Make sure to choose *Runtime* as *Python 3.7*
    - The code given is written for Python 3.7 and does not work with Python 3.8
- Expand *Choose or create an execution role* by clicking on the icon ▸
    - For *Execution role* select *Use an existing role*
    - For *Existing role* select *lambda-s3-role* which is created in an earlier task.
    - This role gives permissions to the Lambda function to access Amazon S3 buckets to read and write images and any files
- Now click on *Create function*
- We need to add a trigger now for this Lambda function
    - Lambda functions can be triggered automatically by different activities. For this project we will trigger the Lambda function whenever a new object is created in our first S3 bucket
- Click on *+Add trigger* and give the following configurations
    - For *Select a trigger* choose *S3*
    - For *Bucket* choose the first bucket you created in the previous step; for example *raj-miniproj2*

  - For *Event type* choose *All object create events*
  - Scroll down and make sure that *Enable* button is checked
  - Select check box for recursive warning and click *Add* button
- Now we need to add the code and complete the other configuration settings of this Lambda function
- Click on *Create-Thumbnail* at the top of the page
- Scroll down to the *Function Code* section and configure the following:
  - Click on *Actions* menu (seen on right-hand side) and select *Upload a .zip file*
  - Navigate to the folder where you downloaded the zip file *CreateThumbnail.zip* and select it for uploading
  - Click on *Save*
  - This zip file contains the Python code and all the required libraries for the function to run and generate the thumbnail images from the original image
  - It is a good idea to take a look at the Python code in the file *CreateThumbnail.py*
  - It receives an *Event* which contains the name of the incoming object (bucket name for example *raj-miniproj2*, and file name referred to as key)
  - It then downloads the image to local storage
  - Resizes it using a library called *Pillow library*
  - Uploads the resized image to the second bucket i.e. the bucket with *-prod* suffix to the name, for example *raj-miniproj2-prod* in the folder *thumbnails*
  - It then copies the original larger image to its corresponding folder on the second i.e. the *-prod* bucket; for example *raj-miniproject2-prod* in the folder *images*
- Now scroll down further to *Basic settings* section and click on *Edit*
  - In *Description* enter *Create a thumbnail image*
  - For *Handler* enter *CreateThumbnail.handler* since *handler* is the name of the unction in the Python file *CreateThumbnail.py*
  - Click on *Save*
- Rest of the settings can have the default values
- That completes configuration of this Lambda function

Under S3 Service

4. Test the Lambda functions by uploading an image to the S3 bucket

- From the list of *Bucket names* select the first bucket for example *raj-miniproj2*
- On *Overview* tab in the bucket click on *Upload*
- Drag and drop one of the image files from AdditionalImages directory that was created on your local system/laptop when you unzipped *AdditionalImages.zip* file
- Click on *Upload* to complete the uploading
- The Lambda function should get triggered now to execute their respective actions

Under CloudWatch Service

5. Test the Lambda functions by uploading an image to the S3 bucket

- Select *Logs* → *Log groups*
- We should see our Lambda functions listed
- Click on the function name to see the log group details
- Click on the *Log stream* shown to see the logs
- Expand the messages by clicking on the icon ▸ to see the details

Monitoring can be done from Lambda Service also by clicking on Monitoring tab.

Under S3 Service

6. Upload the rest of the images which will run the Lambda functions and add the remaining images to the website

- From the list of *Bucket names* select the first bucket for example *raj-miniproj2*
- On *Overview* tab in the bucket click on *Upload*
- Drag and drop the rest of the image files from *AdditionalImages* directory that was created on your local system/laptop when you unzipped *AdditionalImages.zip* file
- Click on *Upload* to complete the uploading
- The Lambda function should get triggered now to execute their respective actions

We can check out by giving the URL endpoint of the static website in a browser tab or window. We should now see the web page with all images populated.

Troubleshooting and Debugging

7. It is a best practice to debug and verify each step

- At each of the steps above, we should check if the step is successfully completed or not
- We can do this by testing and verifying whether we see the expected outcome
- If we do not see the expected results at any step
- Check if all the steps are completed in the sequence given or not
- Then all the inputs are entered correctly or not
- This way we can ensure the proper deployment of the solution end-to-end.