Assignment 2

FIT5225 2023 SM1

***CloudSnap*: A Serverless Image Storage System with Tagging**

1 Synopsis

This assignment aims to build a cloud-based online system that allows users to store and retrieve images based on auto-generated tags. The focus of this project is to design a serverless application that enables clients to upload their images to public cloud storage. Upon image upload, the application automatically tags the image with the objects detected in it, such as person, car, etc. Later on, clients can query images based on the objects present in them. To achieve this, the application provides users with a list of image URLs (or tumbnails) that include the specific queried objects.

2 Group Assignment

The majority of software developers will work in a team at some point in their career. Therefore, to prepare students to experience software development as a team, this assignment is designed as a group project. Students will be placed into software teams of up to four members to work on implementing the system described above. In this assignment, students need to organize their team and their collective involvement. There is no designated team leader, but teams may decide to establish processes for agreeing on work distribution. Understanding the dependencies between individual efforts and their successful integration is crucial to the success of the work and for software engineering projects in general. If teams encounter any "issues", they should inform the teaching team as soon as possible, and help will be provided to resolve them. The due date for this assignment is firm, and there is no **'special consideration'** for granting an extension. If a team member fails to complete their work, other team members should handle their tasks and report this to the teaching team as soon as possible. We will evaluate individual involvement in the project, and if a team member fails to satisfy their tasks within the team, they may be penalized heavily, regardless of the project's success.

3 Assignment Description

Teams should develop an AWS cloud-based solution that leverages services such as S3, Lambda, API Gateway, and database services (e.g., DynamoDB) to build a system for automated object detection tagging and query handling. The teams should produce a solution that enables end-users to upload their images into an S3 bucket. Upon uploading an image to a designated S3 bucket, a lambda function is automatically triggered, which uses the Yolo object detection feature to identify the objects in the image and stores the list of detected objects along with the image's S3 URL in a database. Furthermore, the end-user should be able to submit queries to an API endpoint using API Gateway to search for tagged images (more details to come). Table 1 provides a glossary of terms used in the assignment description.

Table 1: Glossary of terms

Term	Meaning
team	A group of 4 students who are doing the project
queries	You will implement multiple APIs to query different information including a request message by the end user to find list of images with specific tags, find images with similar tags of an image, update tags of an image, and delete record of an image.
tags	list of objects detected in an image, e.g., person, car, and cat.
federated authentication	Federated authentication is a Single sign-on (SSO) mechanism that allows you to use authentication credentials of external identity providers such as Google or Facebook to access your AWS services such as Lambda, S3 or DynamoDB.
Peer assessment	Each team member is expected to complete a **confidential** report and evaluation on his/her role in the project and the experiences in working with other team members on the demonstration day.

3.1 Authentication and Authorisation

Security is one of the most crucial aspects of developing cloud-first applications. When your application is publicly exposed, you must ensure that your endpoints and resources are safeguarded against unauthorized access and malicious requests. AWS, through its Cognito service, provides a straightforward, secure, and centralized approach to protect your web application and its various resources from unauthorized access.

To leverage the AWS Cognito service, first, you need to create a user pool that stores user credentials. Then, you need to create and configure a client app that provides access to your application and/or other AWS services to query and use the user pool. Finally, you have two options to communicate with the AWS Cognito service and perform authentication and/or authorization: 1) Use the AWS Amplify JavaScript Library to initialize the authentication module of your application or 2) Use the AWS JavaScript SDK to access the user pool and identity provider(s) that you have defined earlier.

Your application should support the following workflow and features:

- Detect whether a user is authenticated or not. If the user has not signed in, access to all pages/endpoints except the sign-up service needs to be blocked, and the user should be redirected to the "sign-up.html" page to register a new account. For each new account, you need to record the user's *email address*, *first name*, *last name*, and *password*. Cognito will take care of sending an email to new users, asking them to verify their email address and change their temporary password.

- Your application should include a login page that allows users to sign in. After successful authentication, users should be able to upload images, submit queries, view query results, and sign out of the application. All of these services must be protected against unauthorized access. You can implement login and sign-up web pages using either the Hosted UI feature of Cognito or your own version that calls Cognito APIs.

- Uploading files to an S3 bucket, invoking Lambda functions to execute the business logic of your application, and accessing the database for data storage and retrieval all require fine-grained access control permissions that you need to set up via IAM roles and appropriate policies. It is important to

note that IAM roles in AWS Academy have several limitations. Therefore, you should carefully consider how to perform authentication and authorization in your system while taking these limitations into account.

As an **optional** feature, you can add federated authentication using AWS Cognito Identity Pools to your application and earn bonus marks (up to 5 points shall be awarded if you add federated authentication to your project). For this purpose, you need to create a Facebook or Google app that serves as an external identity provider and authenticates users on behalf of your application, then forwards authentication tokens to your application. Note that having external authentication providers in your project is **not** mandatory. Since federated authentication might be challenging and maybe impossible with your AWS academy account, I **strongly** recommend that you finish the requirements of the assignment first, and then, if you have extra time, work on this feature.

3.2 Image Upload

Your solution should provide a mechanism to upload an image to an S3 bucket. Uploading an image to an S3 bucket can be done either through an API Gateway endpoint (using POST REST APIs) or it can be done directly using AWS SDKs (for instance, boto3 if you are using Python). Whenever an image is uploaded to the S3 bucket, your system must trigger an event and invoke a Lambda function.

You are expected to configure the triggers and grant the required Amazon resource permissions (execution roles) for the Lambda function. The Lambda function should read the uploaded image, detect objects in the image, and save the list of detected objects (called tags) along with the S3 URL for that image in an AWS database for future queries. Please note that you should update your Yolo script that you were given in your **first assignment** to suit the AWS Lambda function. This includes removing any Flask-related code, incorporating the Lambda function and required libraries to read the image from S3 buckets, and storing the S3 URL and tags in the database. You may create a separate S3 bucket to store the Yolo and other config files, which can be referenced in your Lambda function.

You may also ignore detected objects with an accuracy of detection below a specific threshold (e.g., 0.6). Please note that **Amazon Rekognition** is a specialised AWS service for identifying objects in images and can be used instead of Yolo. However, in this assignment, you are **not** allowed to use the Amazon Rekognition service.

3.3 Queries

Your solution should provide APIs which allow following queries.

1. **Find images based on the tags**: The user can send a text-based query to request URLs of images that contain specific tags with a minimum repetition number for each tag (e.g., "person, 1", "person, 2, car, 1"). You are expected to create an API Gateway with a RESTful API that allows users to submit their requests, such as GET or POST requests.

   Your application might send a list of tags via specific GET parameters in the requested URL, for example:

   ```
   https://jyufwbyv8.execute-api.us-east-1.amazonaws.com/dev
   /search?tag1=cat&tag1count=1&tag2=car&tag2count=1
   ```

   or it can be a POST request with a JSON object including a list of tags and their counts. A sample JSON object for a query request is given below:

```
{
"tags": [
    {
        "tag": "person",
        "count": 1
    },
    {
        "tag": "desk",
        "count": 2
    },
    ...
    ]
}
```

A response should include the list of URLs to all images that contain **all** the requested tags with **at least** the number in the count value in the query. This can be a JSON message similar to the following:

```
{
    "links": [
        "https://cloudsnap.s3.amazonaws.com/image1.png",
        "https://cloudsnap.s3.amazonaws.com/image59.png",
        "https://cloudsnap.s3.amazonaws.com/image180.png"
        ]
}
```


Your query may require to trigger one more Lambda function that receives a list of tags and finds s3-url of images containing **all tags in the query with least number of repetition** from the database, i.e., logical **AND** operation, not **OR** operation between tags.

By default, a count of 1 should be considered for each tag if it is not specified in the UI or the query.

In your UI instead of showing the s3-urls you can preview the images found as the results.

2. **Find images based on the tags of an image**: The user can send an image as part of an API call. The list of all objects (tags) and their counts in the sent image is discovered and then all the images in *CloudSnap* storage containing those set of tags and with at least count numbers are found. Finally, as a response, the list of URLs to matching images (similar to to the previous section) are returned to the user. You should make sure that the image uploaded for the query purpose is not added to the database or stored in s3.

3. **Manual addition or removal of tags**: Your solution should also provide an API that allow end-user to add or remove tags of an image. You are expected to create a POST API which allow users to submit their requests.

   A sample JSON message sent to add/remove tags is:

```
{
    "url":"https://cloudsnap.s3.amazonaws.com/image1.png",
    "type": 1, /* 1 for add and 0 for remove */
    "tags": [
        {
```

```
        "tag": "person",
        "count": 2
        },
        {
        "tag": "alex",
        "count": 1
        }
    ]
}
```

"type" can be set to 1 or 0 for adding or removing a tag, respectively. The above request adds two `"person"` tags to the image and one `"alex"` tag to the tag list of the image in the URL: If "type" is set to 0, the tags are removed from the tag list of the image up to the maximum of either the available tags or the count value. For example, if the count for the tag person is 2 and only one tag of person is in the tag list, we remove the only existing tag. If a tag is not included in the list of tags requested for deletion, you can simply ignore it in the request. Please note that if "count" is not specified in the request, the default value of 1 should be considered.

`https://cloudsnap.s3.amazonaws.com/image1.png`.

This can be done through a Lambda function to update the entry in the database.

4. **Delete an image:** The user can communicate the URL of an image to an API and the system should remove the image from s3 and relevant entries from the database.

3.4 User Interface

You can design a simple user interface (UI) (We suggest web-based GUI). But UI can be of any form that includes the following: upload images, submit queries, and view query results. A UI makes your system easier and more enjoyable to use. However, you have the option not to create a UI for application or have UI for some parts and not the others. If you opt to ignore UI or full-fledged UI, you can write scripts (e.g., Python) to handle communications with your application APIs. Please be aware that you might need to manually copy credentials provided by the identity pool to your scripts each time if you choose to work with the later option.

4 Final Reports

You should write an **individual** and **team** report on the developed application. You should use a highly-legible font type such as: *Arial*, *Helvetica*, and *Times New Roman* with font size 12 points.

4.1 Team Report - 750 words

Please prepare a team report based on the following guidelines:

- Include an architecture diagram in the team report. For the architecture diagram, you must use AWS Architecture Icons (more info can be found here: `https://aws.amazon.com/architecture/icons/`).
- Include a table to describe the role of each team member in your team report, You can provide a three-column table in your report that shows student name and id, percentage of contribution and elements of the project the member contributed (maximum of 100 words per member).