Assignment 2

FIT5225 2021 SM1

TagTag: A Modern Image Storage on the Cloud

1 Synopsis

This assignment aims at building a cloudified online system that allows users to store the images and retrieve the images based on the auto-generated tags. The focus of this assignment is to design a serverless application that allows the client to upload their images to public cloud storage. Upon the image upload, the application automatically tags the image with the type of objects detected in it, for example, person, car, etc. Later on, clients can query images based on the type of objects in the images. To this end, the application provides users with a list of URLs for images that include specific queried objects.

2 Group Assignment

Majority of software developers will work in a team at a certain point in their career. Hence, to prepare the students to experience software development as a team, this assignment is designed as a group project. Students will be put into software teams to work on the implementation of the system described above. Each team will have up to 4 students. In this assignment, students need to organise their team and their collective involvement. There is no team leader as such, but teams may decide to set up processes for agreeing on the work and who does what. Understanding the dependencies between individual efforts and their successful integration is key to the success of the work and for software engineering projects more generally. If teams have "issues", then please let the lecturer know asap and help will be provided to resolve them. We will evaluate your involvement in the project individually and if you fail to satisfy your tasks within the team you might be penalised heavily irrespective of your project success.

3 Assignment Description

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

3.1 Authentication and Authorisation

Security is one of the most important aspects of developing cloud-first applications and when your application is publicly exposed, you need to ensure that your endpoints and resources are safeguarded against unauthorised access and malicious requests. AWS, via its Cognito service, provides an easy, secure, and centralised approach to protect your web application and its various resources from unauthorised access.

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.

To leverage the AWS Cognito service, first you need to create a **user pool** that persists user credentials. Then you need to create and configure a client app that grants 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 authorisation: 1) Use AWS Amplify JavaScript Library to bootstrap the authentication module of your application 2) Use 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 "sign-up.html" page to register a new account. For each new account, you need to record user's *email address*, *first name*, *last name*, and *password*. Cognito will take care of sending an email to new users and asking them to verify their email address and change their temporary password.

- Your application should have a login page that allows users to sign-in. Upon successful authentication, the user should be able to upload images, submit queries, view query results, and sign out of the application. All these services need to be protected against unauthorised access. You can either use Hosted UI feature of Cognito to implement login and sign-up web pages or create your own version and call 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.

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, I **strongly** recommend you to finish the requirement of the assignment first; 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 (POST REST APIs) call 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 thus invoke a Lambda function.

You are expected to configure the triggers, and grant required Amazon resource permissions (execution roles) for the Lambda function. The Lambda function should read the uploaded image and detect objects in the image and save the list of detected objects (we call them tags) along with the S3-url for that image in an AWS database for future queries. Please be advised that you should update your Yolo script that you have be given in your first assignment to suit with the AWS Lambda function. This includes removing any FLASK related code, incorporating Lambda function and required libraries to read the image from s3 buckets, storing 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 might also ignore detected objects with an accuracy of detection below a specific threshold (e.g., 0.5). Please note that **Amazon Rekognition** is specialised AWS service for identification of objects in images and can be used instead of Yolo. However, in this assignment, you are **not** allowed to use 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 (e.g., person, person and car, person and desk and cat). You are expected to create an API Gateway with a RESTful API that allow users to submit their requests, e.g., GET or POST requests. Your application might send a list of tags via certain GET parameters through the requested URL, for example:

   ```
   https://jyufwbyv84.execute-api.us-east-1.amazonaws.com/dev/search?tag1=cat&tag2=car
   ```

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

   ```
   {
       "tags": [
           "person",
           "desk",
           ...
           "cat"
       ]
   }
   ```

   A response should include the list of URLs to all images that contain all the requested tags in the query. This can be a JSON message similar to the following:

```
{
    "links": [
        "https://tagtag.s3.amazonaws.com/image1.png",
        "https://tagtag.s3.amazonaws.com/image59.png",
        "https://tagtag.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** from the database.

Please note that duplicate tags should not affect the results. This means a query with

`{"tags":["person","person"]}` results in the same response as `{"tags": ["person"]}`.

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) in the sent image is discovered and then all the images in TagTag storage containing those set of tags are found. Finally, as a response, the list of URLs to matching images (similar to the above query) 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. Similar to previous query, duplicate tags would not affect the results.

3. **Add extra tags to an image**: Your solution should also provide an API that the end-user can add extra tags to an image. You are expected to create an API Gateway with a POST API that allow users to submit their requests to add extra tags to an image.

   A sample JSON message sent to the API is:

```
{
    "url":"https://tagtag.s3.amazonaws.com/image1.png",
    "tags": [
        "person",
        "alex"
    ]
}
```

   This request adds `"person"` and `"alex"` to the tag list of the image in the URL:

   `https://tagtag.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. But it can be any form of UI) that includes the following pages: login, sign-up, 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 second option.

4 Final Report

You should collectively write a team report on the developed application and include an architecture diagram. For the architecture, you must use AWS Architecture Icons (more info can be found here: `https://aws.amazon.com/architecture/icons/`).

You should describe the role of each team member in your report and explain where the team worked well and where issues arose and how they were addressed. You can provide a three-column table in your report that shows student name, percentage of contribution and elements each member contributed.

We will ask members of each team to assess their peers' contributions and involvement in the project on the demonstration day (possibly through a Google form). Each team member is expected to complete a **confidential** report on his/her role in the project and the experiences in working with their individual team members on that day. This will be handled along with your final report. (This is not to be used to blame people, but to ensure that all team members are able to provide feedback and to ensure that no team has any member that does nothing!). Please be a good citizen of your team and collaborate efficiently towards the goal of the project. If you recognise someone is struggling with his/her tasks other team members might help him/her. If someone does not deliver allocated tasks or does not work on the assignment as expected, you should report it to me immediately do not postpone this to the demo date!

Please prepare your report based on the following guidelines:

- The length of your report is **Seven A4 pages maximum** (including tables, figures, references, etc.) with at least 1.5 cm margin on each side and at top and bottom. You should use a highly-legible font type such as: *Arial*, *Helvetica*, and *Times New Roman* with font size 12 points.
- Report should comprise a description of the system design and architecture and how/why such design choices were made (roughly 3 to 4 pages). It is highly recommended that the architecture of the system is represented with AWS Architecture Icons.
- Report should describe the role of team members and how each member contributed in the delivery of the system (maximum one page).
- Report includes a simple user guide for testing (you should keep your application up and running for two weeks after the submission deadline).
- Your report should include a link to the source code (GitHub or Bitbucket). It is recommended that all students commit their code to a private code repository rather than delegate this to a single team member. This can provide an evidence base if teams run into "issues". You should share your private repository with the whole teaching team. Please do not use a public repository to avoid any **plagiarism**.
- In your report you should answer the following questions (a paragraph with max 300 characters):
  1. What are the updates you will perform to your application if you have users from all over the world?
  2. What sort of design changes you will make to reduce chance of failures in your application?
  3. What are the design changes you will make to increase the performance of your applications in terms of response time, query handling, and loading images?
- [**Optional**]: we are interested to hear a real-world application for the solution you provided in this assignment. Bonus points will be awarded to the final mark of this unit (the amount is a surprise!!!) to a team that provides the best use case scenario for this application. Think of a start-up company that extend your solution. You might consider any modification to your solution, but the core logic of object detection with Yolo should be kept, i.e., you cannot replace object detection with

face detection. You can think of polling objects from Twitter or Instagram (Not more than 300 characters).

5 Submission

You need to submit the following via Moodle:

- Your report in **PDF format**. Please do not forget to include your team members' name and student ids.

Submissions are open now and **due at 11:55pm on 28th of May**. This is a **firm** deadline and we do not provide any extensions!

6 Demonstration Schedule and Venue

Teams are required to give a virtual presentation (with a few slides if you wish) and a demonstration of the working application via Zoom. All team members are required to be present at the demonstration time slot to answer questions. The presentation should include the architecture of your application as well as the design and implementation choices made. Each team has up to 10 minutes to present their work and 10 minutes for Q&A. This will take place on **8th and 9th of June**!! We ask you to select a time slot during the working hours to join a Zoom meeting. You have almost a week between the time you submit your assignment and the demonstration day. During this week, you can prepare your presentation but you are not allowed to update your code or work on the implementation (No commit to the repository and no updates on your AWS account).