FIT 5225 Ass2 Team Report

———— A Modern Image Storage on the Cloud



**Group Id:** group 40

**Student Name:**

Hanying Li 30529360

Shizhen Wang 29339995

XinYuan Ren 30994853

Christopher Chang 31521541

# System Introduction

## Background

The system is an online cloud-based application whose primary function is to allow for users to store images and retrieve them based on automatically generated tags. End users can upload their images to a public cloud storage space. The uploaded images can then be inspected by the system and tagged with tags corresponding to the type of object, e.g. person, car, etc. Users are allowed to search for images of the same source based on different types of tags.

## System design and architecture

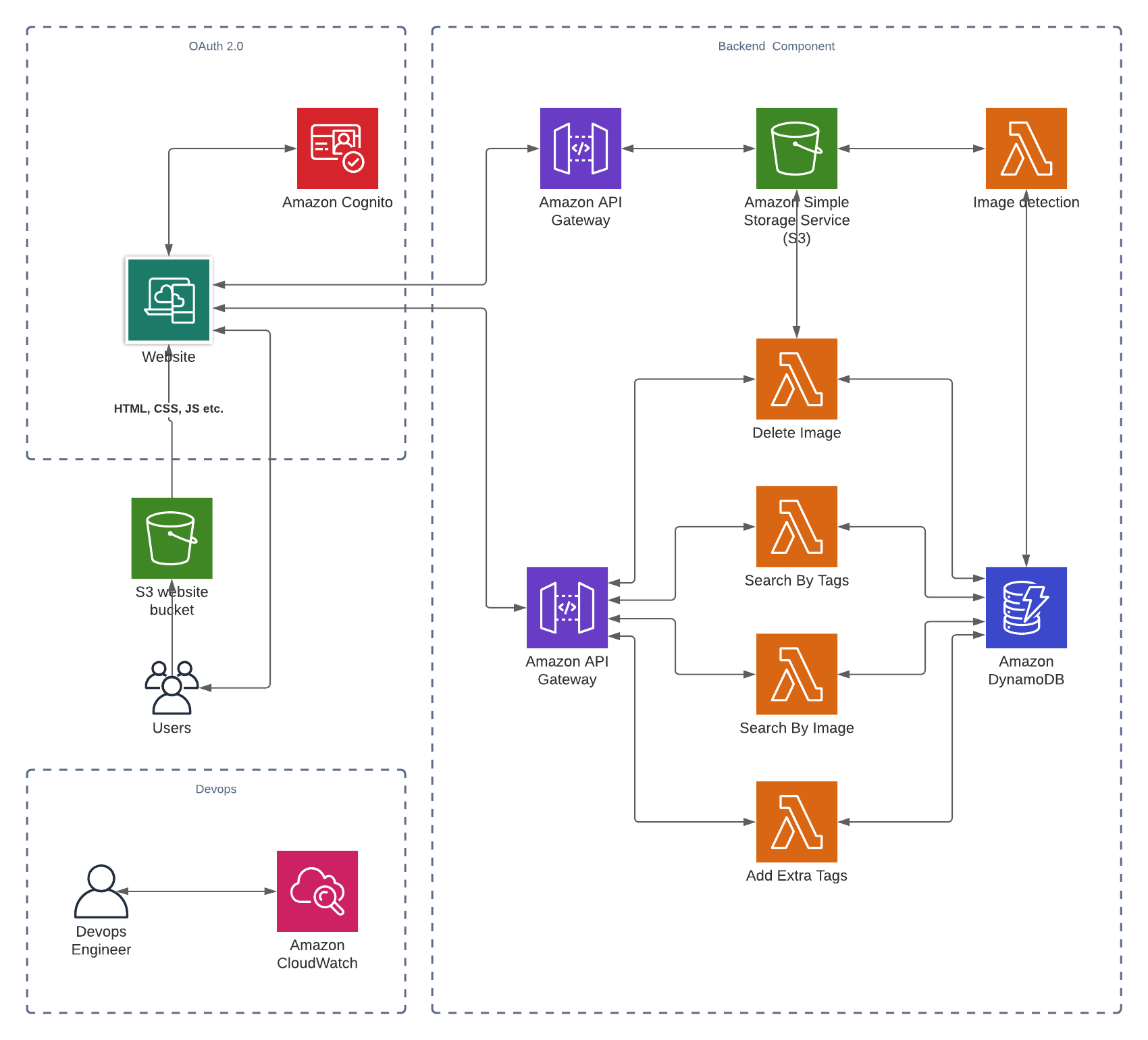


Diagram 1. System architecture diagram

The system consists of four main functionalities: authentication, image upload and storage and image query. The details are as follows:

We use the React framework to build the front-end and deploy it on an S3 bucket. When a user opens the website page, he/she needs to login and new users have to sign up. User creation and authentication are done through AWS Cognito service and extra protected authentications have been implemented to prevent unauthorised access from non-logged in users.

Through AWS Amplify, the application communicates with the Cognito service, which enables user creation and authentication. Upon successful authentication, the web page receives a user pool token from Cognito. API gateway could validate this token and allow the user further access to resources.

The first feature of this site is image upload and storage. The images will be uploaded to AWS S3 for storage via the API gateway. When the images are successfully uploaded, a Lambda function will be triggered automatically. This Lambda will recognize the uploaded images with yolo3 tiny's model. After that, the recognized image information will be stored in DynamoDB.

Another feature of this website is the search and modification of images. Specifically, the user can perform four operations: search by tags, search by image, add extra tags and delete image. In this section, we build four Lambda functions to implement these four queries. Through the REST APIs, each of these four lambda functions can be called and their functions implemented.

When deployed and operating, we use AWS Cloudwatch to monitor logs and ensure smooth functionality.

## System architecture justification

In this section, we will explain the reasons and benefits of this system architecture. We will describe each of the four sections: serverless architecture, security, data storage and access, and functional implementation.

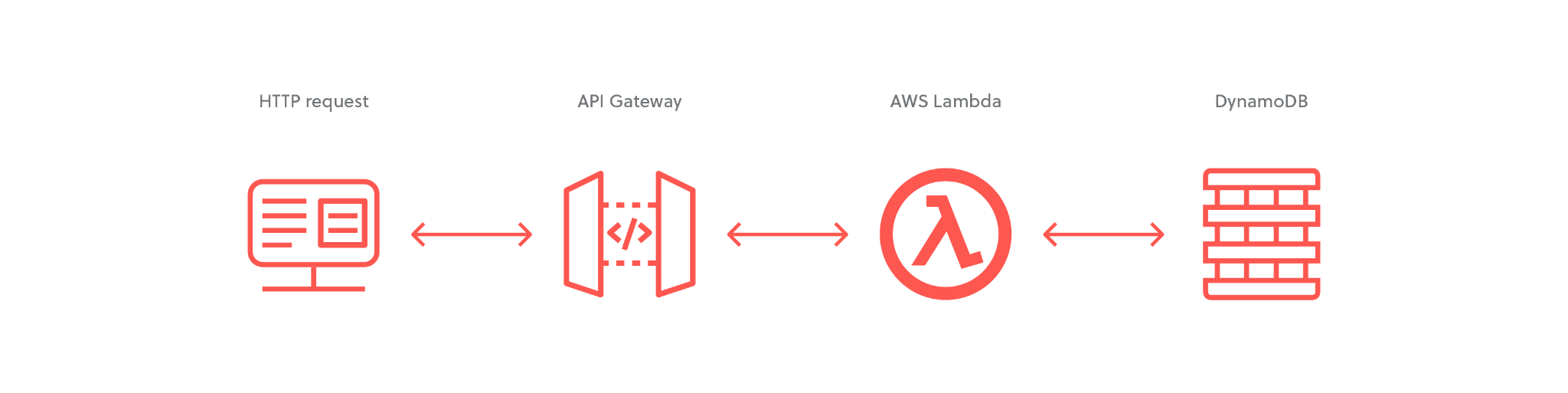


Diagram 2. Server architecture

This site uses a serverless architecture. Serverless is a model of cloud computing. In this framework, there is no need for users to deploy, configure, and manage servers. They only need to use the services provided by the cloud service platform. There are three main reasons why serverless architecture is adopted:

1. Faster from idea to market. Serverless helps users focus on features and code, resulting in faster and smoother feature releases and iterations.
2. To reduce costs. Using a serverless architecture can avoid paying for over-premises while meeting elastic usage requirements for small applications like this site.
3. Serverless is easier to build and configure because of AWS' service integration.

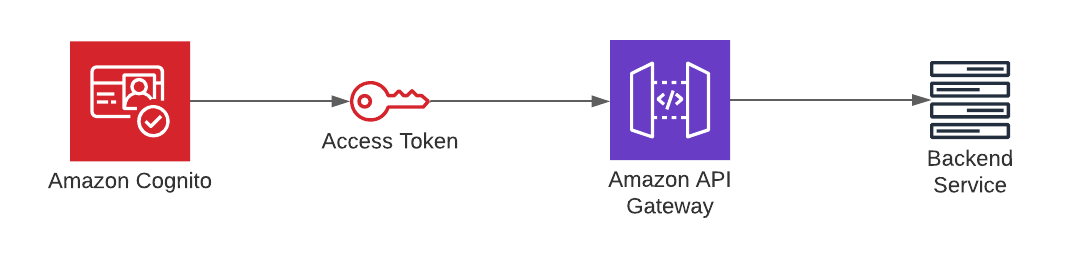


Diagram 3. Server architecture

Security is one of the most critical aspects of application development. To protect the endpoints and resources of this website, we use AWS Cognito to block unauthorised access. We chose Cognito service because it can easily help us to implement user registration and authentication. It also supports user login using other external social identities, such as Facebook, Twitter, etc. Also, the integration between Cognito and the application is simple. In this case, we used AWS Amplify to integrate the website with Cognito.

Regarding data storage and reading, there are two primary components involved in this system: S3 and DynamoDB. S3 serves to store images uploaded by users. S3 has the advantage of extremely high data persistence. With automatically created and stored copies of objects, S3 can guarantee up to 99% data persistence. DynamoDB is the database for storing image information in this system. DynamoDB is a non-relational database that provides extremely high write and read performance. Using DynamoDB for storage can help support users to query images quickly and bring a better user experience.

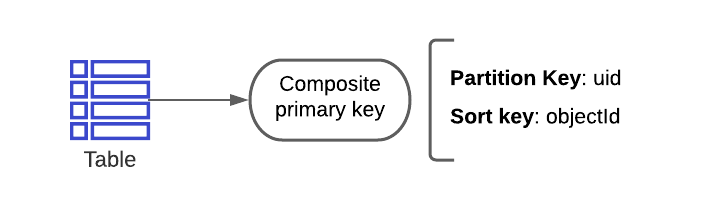


Diagram 4. Composite primary key

In DynamoDB, we design a composite primary key to ensure efficient and easy data queries. We use user ID as a partition key to ensure that all user images are stored in a single partition, ensuring query efficiency. We also design the object ID as the sorting key to find the exact image that the user is querying.

The advantage of Lambda is that it helps the user enable the correct number of instances to implement the functionality based on the number of events, without the need for the user to configure and manage the instances. This Lambda is triggered by the S3 event, does the computation, and stores the information in DynamoDB. Here we break up uploading images and recording information into two steps. This is because it takes a long time to use yolo. Therefore this part of the processing should be done in the background, thus improving the user experience.

# Role of Team Members

| Student Name | percentage of contribution | Contributed Section |
| --- | --- | --- |
| Hanying Li | 25% | 1. Finished four queries in 3.3. Including find image based on tags, find image based on image, add extra tags to an image, and delete image. 2. Drew system architecture diagram 3. Finished chapter 1 and 4 of the team report. |
| Christopher Chang | 25% | 1. Developed UI for Dashboard with Router Links, Signup Page, Login Page, Query Page and Upload Image Page. 2. Cognito Service |
| Shizhen Wang | 25% | 1. Completed the task 3.2.   Including achieving the trigger, load the image and detect the image.   1. Create the connection part between the client and S3. |
| Xinyuan Ren | 25% | 1. Configure userpool and S3 bucket for react app 2. Using amplify library to refine the react app code so it can connect to the userpool |

# 

# User Guide

1. Navigate to “http://5225-team40-app.s3-website-us-east-1.amazonaws.com/” to view the log in page.
2. Register a new account by providing email address, first name and last name, password.
3. Login using email address and password.
4. Upon login, Select an option:
   1. Upload an Image
   2. Find Image based on Tags (User entered)
   3. Find Image based on Tags of Image
   4. Add Tags to Image
   5. Delete Image
   6. Log Out
5. Follow the prompts based on your selection.

# Improve Direction

Q1. To support global user access, we need to deploy our app in every AWS region and ensure the app across multiple Availability Zones in each region. Also we could use Lambda@Edge and Amazon CloudFront content delivery network (CDN) to run code closer to users with low latency.

Q2. To reduce the chance of failures, we can use Rolling deployments for Lambda functions. Through it we can monitor the status of the Lambda function and initiate a rollback if any problems are detected. Also, we need to do a good information verification on the front-end page to prevent users from uploading invalid images or information.

Q3. To reduce response time, we should launch Provisioned Concurrency, which could keep functions initialized and ready to respond in a very short time. Also we can use a plugin to invoke Lambda function in a selected time interval to keep Lambda "warm". To make query efficient, we can add suitable indexes to the table. To help load images, we can use a cache like Amazon CloudFront, Amazon ElastiCache, or AWS Elemental MediaStore. They can help to improve performance.

# Github link

1. <https://github.com/hsuchristopher/FIT5225_A2>
2. https://github.com/hlii0132/FIT5225\_a2

# Reference

# Github: [mubbashir10](https://github.com/mubbashir10)/[aws-cognito-react-auth](https://github.com/mubbashir10/aws-cognito-react-auth)

1. *awsdocs/aws-lambda-developer-guide*. (n.d.). GitHub. Retrieved May 30, 2021, from https://github.com/awsdocs/aws-lambda-developer-guide