BirdTag: An AWS-powered Serverless Media Storage System with Advanced Tagging Capabilities

FIT5225 2025 S1
Due date: Beginning of Week 14 (11:55PM, Monday, 09/June/2025)

1 Background

Monash Birdy Buddies (MBB) is a group of researchers, students, and staff who love everything about birds at Monash University. MBB is all about exploring the amazing world of birds – from the tiniest wrens hopping in the bush to the majestic raptors soaring overhead. During the past few years, MBB has expanded its operations and membership to multiple Monash campuses in many countries.

Think about all the fantastic stuff MBB captures: those stunning shots of birds in action, the unique calls and songs recorded, and those hilarious or insightful video clips of behaviour. This is the gold mine for bird research! It's how MBB identify species, tracks individuals, studies behaviour, monitors populations, and ultimately, figures out how to help protect our feathered friends. But right now, imagine trying to find that specific photo of the rare bird from last year if it's buried on someone's random hard drive, or losing a whole batch of important audio recordings because a memory card got corrupted. Data scattered everywhere is data that's hard to use, easy to lose, and impossible to share properly.

A good storage system means everything is safe, backed up, and organised in one central location. It means any MBB member who needs data can find it efficiently. Collaborating on projects is a breeze because we all look at the same organised data. It ensures the valuable information we collect isn't lost, allowing us to build on our work over time and contribute robust findings to bird conservation and science. It's the essential nest that keeps all our precious bird observations secure and accessible, allowing us to spend less time searching and stressing and more time doing the awesome work of understanding and helping our avian pals!

This assignment aims to build an AWS-based online storage system that allows users to store, organise and retrieve media files based on auto-generated and manual tags. The focus of this project is to design a serverless application that enables clients to upload their media files (images, audios and videos) to public cloud storage. Upon upload, the application automatically tags the file with the bird species detected in it, automatically sorts/moves the file to the corresponding directory and adds a record in a database for searching. Later, files can also be queried based on the bird species in them.

2 Group Assignment

Most software developers will work in a team at some point in their career. Therefore, this assignment is designed as a group project to prepare students to experience software development as a team. 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 organise their teams and 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 for any reason, 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. If a team member fails to satisfy their tasks, they may be penalised heavily, regardless of the project's success.

2.1 Assignment Description

Teams should develop an AWS-based solution that leverages services such as S3, Lambda, API Gateway, and database services (e.g., DynamoDB) to build a system for automated bird tagging (pretrained models and sample code are provided) and query handling. The teams should produce a solution that enables end-users to upload their files into an S3 bucket. Upon uploading a file to a designated S3 bucket, a lambda function is automatically triggered, which uses the pretrained model to identify the bird in the file and stores the list of detected birds along with the file'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 files (more details to come).

3 Authentication and Authorisation (10%)

Security is one of the most crucial aspects of developing cloud applications. When your application is publicly exposed, you must ensure that your endpoints and resources are safeguarded against unauthorised access and malicious requests. AWS, through its Cognito service, provides a straightforward, secure, and centralised approach to protect your web application and its various resources from unauthorised access. To leverage the AWS Cognito service, you first 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 options to communicate with the AWS Cognito service and perform authentication and/or authorisation. You can use the AWS JavaScript/Python 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 or endpoints except the sign-up service needs to be blocked, and the user should be redirected to the sign-up 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 files, submit queries, view query results, and sign out of the application. All of these services must be protected against unauthorised 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 authorisation in your system while considering these limitations.

4 Core Functionalities (50%)

4.1 Model Handling

Machine learning and deep learning models are often updated to improve their quality. If MBB provide a new version of the model to you, your solution design should be flexible enough to use the new model without changing your source code/lambda function.

4.2 File Handling (20%)

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

- **Building Images Thumbnails**: Upon uploading an image to S3, a Lambda function should make a thumbnail for that image. Creating a thumbnail for an image involves resizing the original image to a smaller dimension while maintaining its aspect ratio, followed by compression to reduce file size. You can use OpenCV library for this purpose. You are expected to configure the triggers and grant the required Amazon resource permissions (execution roles) for the Lambda function.

- **Tagging Files**: Another Lambda function should be triggered to read the uploaded file, detect birds in the images/audios/videos, and save the file type and a list of detected bird species (tags) along with the S3 URLs for that file and its thumbnail (in case of image) in an AWS database for future queries.

4.3 Queries (25%)

Your solution should provide APIs that allow the following queries:

- Find images and videos based on the tags: The user can send a JSON-based query to request URLs of files that contain specific tags with a species of bird and minimum counts (e.g., {"crow": 3}, {"pigeon": 2, "crow": 1}, which shows all files with ≥ 3 crows in the first case and ≥ 2 pigeon AND ≥ 1 crow in the second example). You are expected to create an API Gateway with a RESTful API allowing users to submit their requests, such as GET or POST. Your application might send a list of tags via specific GET parameters in the requested URL, for example:

  https://xxx.us-east-1.amazonaws.com/dev/search?tag1=crow&count1=1&tag2=pigeon&count2=2

  or it can be a POST request with a JSON object including a list of tags and their counts. A response should include the list of URLs to all thumbnails of images and full URLs for videos that contain all the requested tags in the query. This can be a JSON message similar to the following:

```
{
"links": [
"https://xxx.s3.amazonaws.com/bird1-thumb.png",
"https://xxx.s3.amazonaws.com/bird59-thumb.png",
"https://xxx.s3.amazonaws.com/bird180-thumb.png"
]
}
```

  Your query may require triggering one or more Lambda function that receives a list of tags from the database, i.e., logical AND operation, not OR operation between tags. In your UI, instead of

showing the s3-urls of thumbnails you can preview the thumbnails found as the results. The user can request the full-size image by clicking on the thumbnails or sending another query to get the full-size image. In case of video files, show a link to allow the user to download the files.

- **Find files based on bird species**: User can send a request similar to previous query **without count**, e.g. {"crow"}, your application will return all images, audios and videos contain at least one crow.
- **Find files based on the thumbnail's URL**: Users can input the s3-url of a thumbnail into the system and receive the corresponding full-size image s3-url.
- **Find files based on the tags of a file**: The user can send a file as part of an API call. The list of all birds (tags) in the sent file is discovered and then all the files in the bucket containing those set of tags are found. Finally, as a response, the list of s3-URLs of the matching images thumbnails or audio/video files (similar to to the previous section) are returned to the user. You should ensure that the file uploaded for the query purpose is not added to the database or stored in s3.
- **Manual addition or removal of tags with bulk tagging**: Your solution should also provide an API that allows end-users to add or remove tags from files. You are expected to create a POST API that enables users to submit their requests. A sample JSON message sent to add/remove tags is:

```
{
"url": [
"https://xxx.s3.amazonaws.com/image1-thumb.png",
"https://xxx.s3.amazonaws.com/image60-thumb.png",
"https://xxx.s3.amazonaws.com/image23-thumb.png",
]
"operation": 1, /* 1 for add and 0 for remove */
"tags": [
"crow,1",
"pigeon,2"
]
}
```

  "operation" can be set to 1 or 0 for adding or removing a tag, respectively. The above request adds 1 "crow" and 2 "pigeon" tags to the tag list of the files in the list of URLs. If "operation" is set to 0, the tags are removed from the tag list of the file. If a tag is not included in the list of tags requested for deletion, you can simply ignore it in the request.

- **Delete files**: The user can send a list of URLs to an API, and the system should remove the files and their thumbnails (in case of images) from S3, and all relevant entries from the database.

4.4 Tag-based Notifications (5%)

Users can receive email notifications or updates related to files based on specific tags. For example, users can receive notifications when new images with specific birds are added or updated to the system. You can use AWS SNS for this purpose.

5 User Interface (15%)

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 files, 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 latter option.

6 Demo and Final Reports (25%)

You should write an individual and team report on the developed application. Use a highly legible font such as Arial, Helvetica, or Times New Roman with a font size of 12 points.

6.1 Demo Performance

6.2 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).
- Report includes a simple user guide for testing your application.
- Your report should include a link to the source code (GitHub, Gitlab or Bitbucket). All students must commit their code to a private code repository rather than delegate this to a single team member. This can also provide an evidence base if teams run into "issues" and how you contributed to the project. You should share your private repository with the whole teaching team. Please do not use a public repository to avoid any plagiarism. Note that the tables, architecture diagram, screen shots of UI, and references are excluded from the word count limit.

6.3 Individual Report - 750 words

Please prepare an individual report based on the following guidelines:

- In your individual report, you should describe your role in the project, and how you contributed to the delivery of the system (150 words).
- Include a personal reflection on the team work your experience in working with other individual team members in your group. Comment on your teammates' work, challenges the entire team faced, and how well the team worked (150 words).
- You should answer the following questions (a paragraph with 100 word each):
  - What sort of design changes you will make to reduce chance of failures in your application?

  - 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?
  - What are the updates you will perform to your application if you have users from all over the world?

Note that the individual report should be done with little or no consultation with other team members. Please be careful about plagiarism and collusion.

7 Submission

You need to submit the following via Moodle:

- Your team report in PDF format. Only one of the team members submit this report. Please do not forget to include your team members' names and student ids. One submission per team!
- Your individual report in PDF format. Every team member should submit this report. Please do not forget to include your name and student ids.

8 Demonstration Schedule, Venue and Peer Assessment

- Teams are required to demonstrate the working application via Zoom.
- All team members must be present at the demonstration time slot to answer questions.
- The presentation should start with the architecture of your application as per your report, along with the design and implementation choices made.
- Each team has up to 3 minutes to present their architecture.
- This is followed by the demo of application as per examiners' guideline for up to 15 minutes and Q&A. We will announce the demo date and time closer to the due date.
- We will ask you to select a time slot during the allocated hours to join a Zoom meeting.
- As mentioned earlier, this is a group assignment. If one of the team members is experiencing difficulties or has special circumstances, the rest of the team should proceed with the project without their fellow member. This aligns with industry practices where the entire project cannot be delayed due to a team member's departure or absence.
- At the end of the session, teams will participate in peer assessments regarding their fellow members' contributions and involvement in the project. **Those who do not actively contribute to the group project may be penalised for up to 50% off their group mark.**
- This feedback may be gathered through a Google form. Each team member is required to submit a confidential report detailing their role in the project and their experiences collaborating with team members.
- These reports will be treated with utmost confidentiality, alongside your final report. The purpose of this evaluation is not to assign blame, but rather to ensure that all team members have an opportunity to provide comments and to guarantee active participation from everyone.

- We encourage all team members to collaborate effectively towards achieving the project's objectives to optimise your team's performance. Should you observe a team member encountering difficulties with their tasks, please offer assistance. If a team member fails to fulfil their assigned tasks or deviates from the project's expectations, we urge you to promptly inform the teaching team. Alternatively, you can utilise this opportunity to bring the matter to our attention.