COMP 4651 Project Report – Serverless Smart Album

Team Information

Introduction

We have re-implemented the Image Recognition example [1] provided by AWS in open source alternatives.

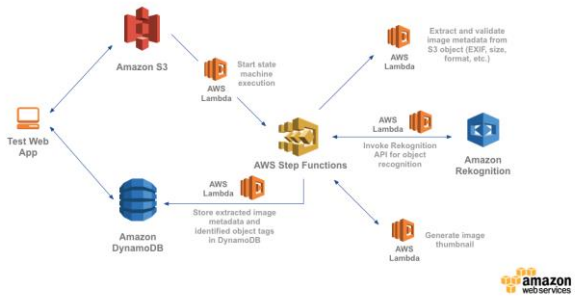


Figure 1 System Architecture of AWS Image Recognition Pipeline

In particular, we have replaced:

- Amazon S3 with Redis
- Amazon DynamoDB with MongoDB
- AWS Lambda with OpenFaaS functions
- Amazon Rekognition with PyTorch

We did not mirror the abilities of AWS Step Function. Instead, the flow is directly implemented inside each OpenFaaS functions.

Most of the AWS Lambdas are translated into OpenFaaS functions written in Node.js and Python. The image thumbnail generation was done on Node.js while the object recognition was done using Python and PyTorch with the Yolov3-tiny model. We did not implement meta-data extraction because we find this function is less useful.

Implementation

Web Client

This function renders a static single page web application using React, interfacing with the user, allowing users to manage albums and photos. Requests are sent to other functions for processing via AJAX.

Storage

This mirrors the AWS S3 service to allow users to upload and retrieve data in a key-value pair manner. This is implemented using a Redis instance. This function exposes a REST API endpoint:

- Show: Retrieve a value using a key from Redis
- Store: Update a key-value pair to Redis

Albums

Allows users to manage albums. This function exposes a REST API endpoint:

- Index: List all albums
- Show: Display details of one album by the album ID
- Store: Create a new album

Photos

Allows users to manage photos. This function exposes a REST API endpoint:

- Show: Display details of one photo by its ID
- Store: Create a new photo

Thumbnail

This is a thumbnail generator function implemented using Node.js smartcropjs library. To obtain a thumbnail, POST request the function with application/text header with a Redis key to the image or image Data URI as body, then the function returns a Redis key to the thumbnail or thumbnail Data URI respectively.

Recognition

This module is replaced with a Yolov3-tiny model written in PyTorch. It takes a POST request as input, which specifies a list of Redis keys of the images. After the model inference, the function returns an JSON object specifying the locations and confidence levels of each detected object in each image.

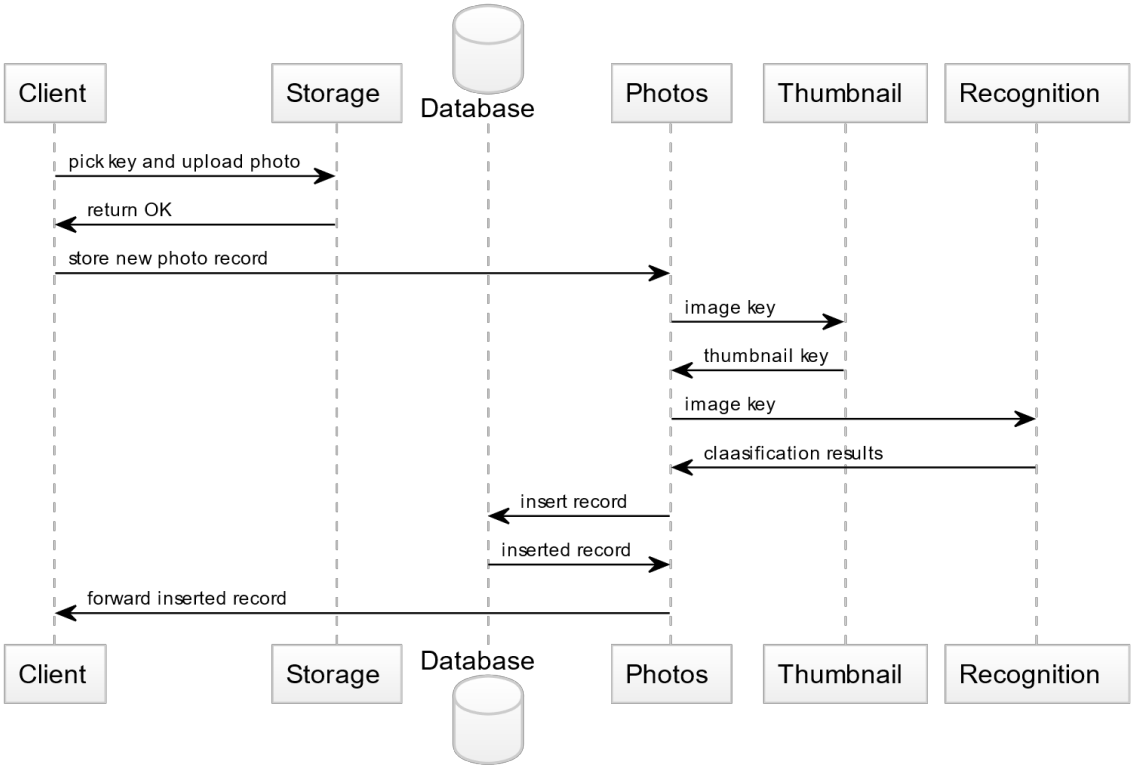


Figure 2 Uploading a new photo

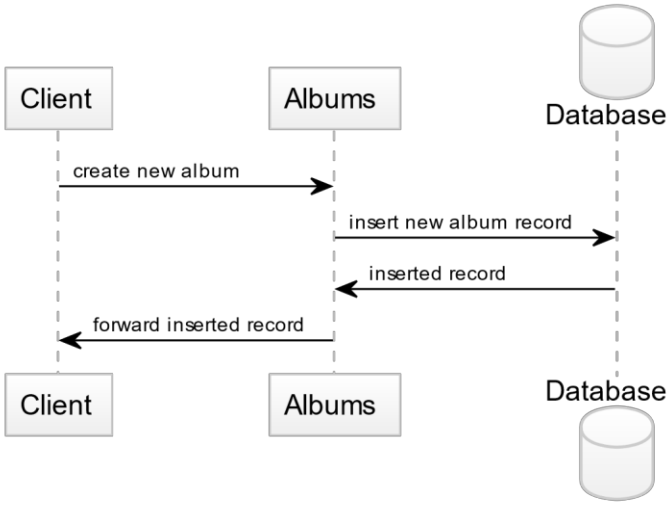


Figure 3 Creating a new album

Setup/Removal

This project is easily deployable to any Kubernetes clusters. Commands are executed in the project root folder.

1. Install OpenFaaS in Kubernetes, if not present.
2. Setup a local docker repository, if not present. Change the repository host in `./stack.yml` accordingly. Default is `localhost:32000`.
3. Setup the necessary helm services by `sh ./scripts/helm-up.sh`
4. Deploy the serverless functions by `faas-cli up`.

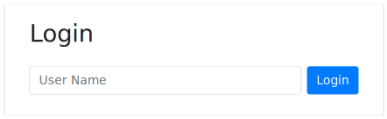
This project can then be removed with ease.

1. Remove the serverless functions by `faas-cli rm`.
2. Remove the helm services by `sh ./script/helm-down.sh`
3. (Optional) Remove the local docker repository
4. (Optional) Remove OpenFaaS.

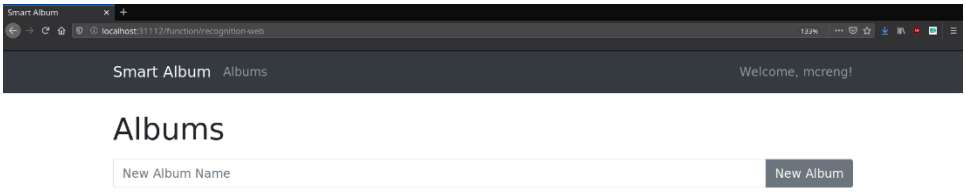
Demonstration

Here we give a simple demonstration of our project.

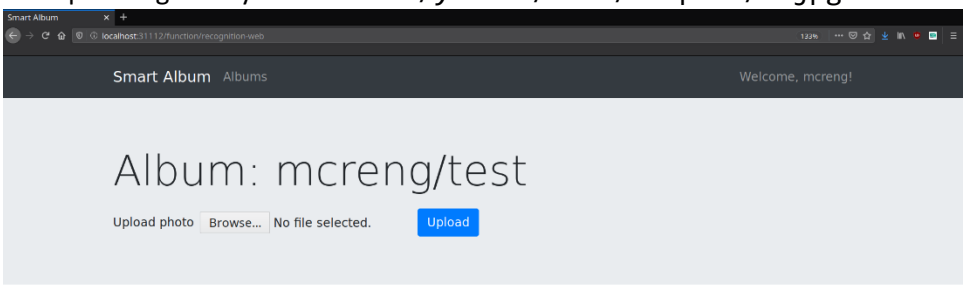
1. After setting everything up, we can login to our web interface by visiting `<openfaas-host>/function/recognition-web` We can login by entering a username.



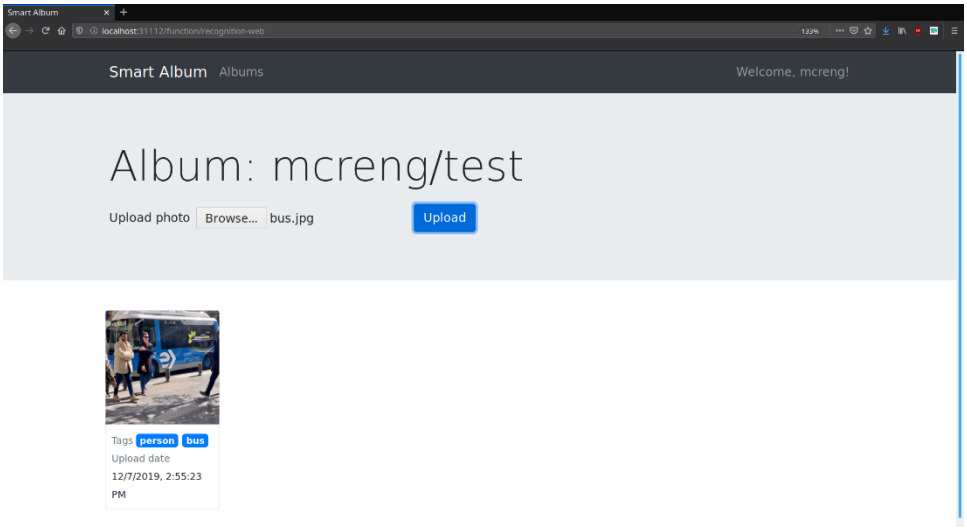
2. After logging in, we can see all the albums uploaded under this user.



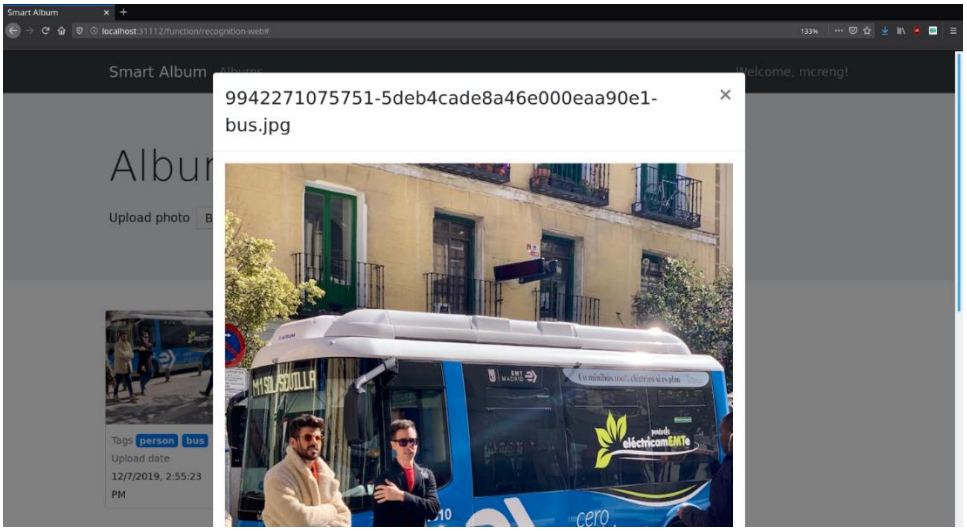
3. We can create a new album by entering an album name and clicking ‘New Album’. Some example images may be found in `./yolov3/data/samples/*.jpg`



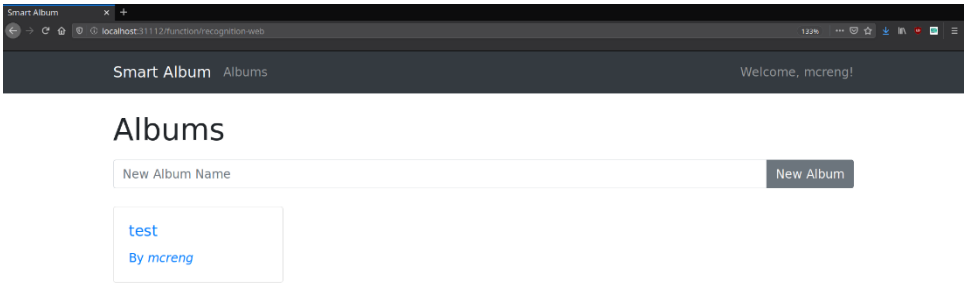
4. We can then upload images to this album. After an image is uploaded, its thumbnail (generated from our thumbnail function), its tags (generated from our yolov3 function) and its upload date are shown.



5. By clicking onto the thumbnail, we can see the original image.



6. We can go back to the main screen and view the created album by clicking ‘Albums’ in the top navigation bar.



References

[1]: https://github.com/aws-samples/lambda-refarch-imagerecognition