Cloud Computing and Big Data - Spring 2022

Homework Assignment 2

Due Date: 24th March, 2022

Assignment:

Implement a photo [illegible]ion that can be searched using natural language through both text a[illegible]arn how to use Lex, ElasticSearch, and Rekognition to create an intelligent search layer to query your photos for people, objects, actions, landmarks and more.

Outline:

This assignment has eight components:

1. **Launch an ElasticSearch instance**[1]
    a. Using AWS ElasticSearch service[2], create a new domain called **"photos"**.
    b. Make note of the Security Group (**SG1**) you attach to the domain.
    c. Deploy the service inside a VPC[3].
        i. This prevents unauthorized internet access to your service.

2. **Upload & index photos**
    a. Create a S3 bucket (**B2**) to store the photos.
    b. Create a Lambda function (**LF1**) called **"index-photos"**.
        i. Launch the Lambda function inside the same VPC as ElasticSearch. This ensures that the function can reach the ElasticSearch instance.
        ii. Make sure the Lambda has the same Security Group (**SG1**) as ElasticSearch.
    c. Set up a PUT event trigger[4] on the photos S3 bucket (**B2**), such that whenever a photo gets uploaded to the bucket, it triggers the Lambda function (**LF1**) to index it.
        i. To test this functionality, upload a file to the photos S3 bucket (**B2**) and check the logs of the indexing Lambda function (**LF1**) to see if it got invoked. If it did, your setup is complete.

---

[1] https://www.elastic.co/webinars/getting-started-elasticsearch?elektra=home&storm=sub1
[2] https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/es-createupdatedomains.html
[3] https://docs.aws.amazon.com/elasticsearch-service/latest/developerguide/es-vpc.html
[4] https://docs.aws.amazon.com/AmazonS3/latest/dev/NotificationHowTo.html

- If the Lambda (**LF1**) did not get invoked, check to see if you set up the correct permissions[5] for S3 to invoke your Lambda function.

d. Implement the indexing Lambda function (**LF1**):

i. vent (**E1**) detect labels in the image, using ectLabels" method).

ii. headObject method[7] to retrieve the S3 metadata ct's upload time. Retrieve the **omLabels** metadata field, if applicable, and ay (**A1**) with the labels.

iii. Store a JSON object in an ElasticSearch index ("photos") that references the S3 object from the PUT event (**E1**) and append string labels to the labels array (**A1**), one for each label detected by Rekognition.

Use the following schema for the JSON object:

```
{
      "objectKey": "my-photo.jpg",
      "bucket": "my-photo-bucket",
      "createdTimestamp": "2018-11-05T12:40:02",
      "labels": [
            "person",
            "dog",
            "ball",
            "park"
      ]
}
```


3. **Search**

a. Create a Lambda function (**LF2**) called **"search-photos"**.

i. Launch the Lambda function inside the same VPC as ElasticSearch. This ensures that the function can reach the ElasticSearch instance.

ii. Make sure the Lambda has the same Security Group (**SG1**) as ElasticSearch.

---

[5] https://docs.aws.amazon.com/lambda/latest/dg/with-s3-example.html (see Configure Amazon S3 to Publish Events)

[6] https://aws.amazon.com/rekognition/

[7] https://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/S3.html#headObject-property

b. Create an Amazon Lex bot to handle search queries
   i. Create one intent named 'SearchIntent'
   ii. Add training utterances to the intent, such that the bot can pick up both keyword searches ("trees", "birds"), as well as sentence ne trees", "show me photos with trees and birds
   be able to handle at least one or two keywords
c. Imple umbda function (**LF2**):
   i. ery "q", disambiguate the query using the Amazon Lex bot.
   ii. If the Lex disambiguation request yields any keywords (K_1, …, K_n), search the "photos" ElasticSearch index for results, and return them accordingly (as per the API spec).
      - You should look for ElasticSearch SDK libraries to perform the search.
   iii. Otherwise, return an empty array of results (as per the API spec).

4. **Build the API layer**
   a. Build an API using API Gateway.
      i. The Swagger API documentation for the API can be found here: https://github.com/001000001/ai-photo-search-columbia-f2018/blob/master/swagger.yaml
   b. The API should have two methods:
      i. PUT /photos

         Set up the method as an Amazon S3 Proxy[8]. This will allow API Gateway to forward your PUT request directly to S3.
         - Use a custom **x-amz-meta-customLabels** HTTP header to include any custom labels the user specifies at upload time.

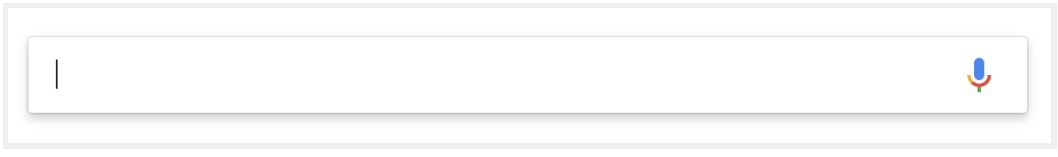
      ii. GET /search?q={query text}

         Connect this method to the search Lambda function (**LF2**).
   c. Setup an API key for your two API methods.
   d. Deploy the API.
   e. Generate a SDK for the API (**SDK1**).

[8] https://docs.aws.amazon.com/apigateway/latest/developerguide/integrating-api-with-aws-services-s3.html

5. **Frontend**
   a. Build a simple frontend application that allows users to:
      i. Make search requests to the GET /search endpoint
      ii. Display the results (photos) resulting from the query
      iii. [redacted]s using the PUT /photos
      [redacted]d form, allow the user to specify one or more
      [redacted]els, that will be appended to the list of labels
      [redacted]utomatically by Rekognition (see *2.d.iii* above).
      [redacted]om labels should be converted to a
      [redacted]parated list and uploaded as part of the S3
      object's metadata[9] using a **x-amz-meta-customLabels**
      metadata HTTP header.

      For instance, if you specify two custom labels at upload time, "Sam" and "Sally", the metadata HTTP header should look like: *x-amz-meta-customLabels: "Sam,Sally"*
   b. Create a S3 bucket for your frontend (**B1**).
   c. Set up the bucket for static website hosting (same as HW1).
   d. Upload the frontend files to the bucket (**B2**).
   e. Integrate the API Gateway-generated SDK (**SDK1**) into the frontend, to connect your API.
6. **Implement voice accessibility in the frontend**
   a. Give the frontend user the choice to use voice rather than text to perform the search.
   b. Use Amazon Transcribe[10] on the frontend to transcribe speech to text (STT) in real time[11], then use the transcribed text to perform the search, using the same API like in the previous steps.
   c. Note: You can use a Google-like UI (see below) for implementing the search: 1. input field for text searches and 2. microphone icon for voice interactions.



[9] https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingMetadata.html
[10] https://aws.amazon.com/transcribe/
[11] https://docs.aws.amazon.com/transcribe/latest/dg/streaming.html

7. **Deploy your code using AWS CodePipeline**[12]
    a. Define a pipeline (P1) in AWS CodePipeline that builds and deploys the code for/to all your Lambda functions
    b. Define a pipeline (P2) in AWS CodePipeline that builds and deploys your fronte[redacted]sponding S3 bucket
8. **Create a AW[redacted]n**[13] **template for the stack**
    a. Creat[redacted] template (T1) to represent all the infrastructure resou[redacted] ElasticSearch, API Gateway, CodePipeline, etc.) and p[redacted]licies, roles, etc.).

At this point you should be able to:

1. Visit your photo album application using the S3 hosted URL.
2. Search photos using natural language via voice and text.
3. See relevant results (ex. If you searched for a cat, you should be able to see photos with cats in them) based on what you searched.
4. Upload new photos (with or without custom labels) and see them appear in the search results.

[12] https://aws.amazon.com/codepipeline/
[13] https://aws.amazon.com/cloudformation/

Acceptance criteria:

1. Using the CloudFormation template (T1) you should be able to stand up the entire functional stack for this assignment.
2. Once a new [illegible] to GitHub (both for frontend and backend repos), CodePipelin[illegible] deploy your code to the corresponding AWS infrastructure
3. For a given [illegible] search query, a correct search (as defined in the assignment) [illegible] return every photo that matches the query. Specifically, [illegible]ns 12 labels for a given photo, your search should return the photo for any one of those 12 labels, if searched independently (“show me **dogs**”) or in groups (“show me **cats** and **dogs**”).
    a. Also, if you search for any custom label, the search should return all the results with that custom label.
4. All other functionality should be working as described above.

ANNEX

Architecture Diagram