Cloud Computing and Big Data Systems - Spring 2024

Assignment 3

Assignment:

Implement a photo album web application, that can be searched using natural language text. You will learn 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**
    a. Using AWS ElasticSearch service, create a new domain called **"photos"**.
2. **Upload & index photos**
    a. Create a S3 bucket (**B2**) to store the photos.
    b. Create a Lambda function (**LF1**) called **"index-photos"**.
    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.
        ii. 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. Given a S3 PUT event (**E1**) detect labels in the image, using Rekognition[6] ("detectLabels" method).
        ii. Use the S3 SDK's headObject method[7] to retrieve the S3 metadata created at the object's upload time. Retrieve the **x-amz-meta-customLabels** metadata field, if applicable, and create a JSON array (**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"**.
   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 searches ("show me trees", "show me photos with trees and birds in them").
         - You should be able to handle at least one or two keywords per query.
   c. Implement the Search Lambda function (**LF2**):
      i. Given a search query "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).

---

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

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**).
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. Upload new photos using the PUT /photos
            - In the upload form, allow the user to specify one or more custom labels, that will be appended to the list of labels detected automatically by Rekognition (see *2.d.iii* above). These custom labels should be converted to a comma-separated list and uploaded as part of the S3 object's metadata 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'*

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

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. **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 frontend code to its corresponding S3 bucket.
7. **Create a AWS CloudFormation[13] template for the stack**
    a. Create a CloudFormation template (T1) to represent all the infrastructure resources (ex. Lambdas, ElasticSearch, API Gateway, CodePipeline, etc.) and permissions (IAM policies, roles, etc.).
    b. NOTE: You do not need to have your assignment working using the cloudformation template. a basic cloudformation template which can just successfully spin up the following resources: both Lambdas, API Gateway, and both S3 buckets(front end and storage buckets). Elastic Search, Code Pipeline Not Required. For the lambdas created using this cloudformation template, code should get updated as you update it in the template. Hardcode the lambdas code in the template or provide a source like github, anything works. And for the frontend s3 bucket, it should be hosted as a public webpage and public url should be output.

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 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.

---

11 https://aws.amazon.com/codepipeline/
12 https://aws.amazon.com/cloudformation/

Acceptance criteria:

1. Using the CloudFormation template (T1) you should be able to create resources used in the assignment.
2. Once a new commit is pushed to GitHub (both for frontend and backend repos), CodePipeline should build and deploy your code to the corresponding AWS infrastructure.
3. The **x-amz-meta-customLabels** feature and all associated functionalities are mandatory.
4. For a given photo and a given search query, a correct search (as defined in the assignment) should be able to return every photo that matches the query. Specifically, if Rekognition returns 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.
5. All other functionality should be working as described above.

ANNEX

Architecture Diagram