CLOUD COMPUTING AND BIG DATA (COMS E6998_008_3)

NOTE SHARE

Goutham Reddy Kotapalle(gk2547)
Reshma Asharaf Beena (ra3027)
Shiv Venkatagiri (skv2109)
Shashank Jaiprakash (sj3003)

ABSTRACT:

The goal of this project is to create an online web application that serves as a collaboration platform for groups of students to share and engage in intellectual discussion over classroom notes. The application aims to support students in the task of organizing their classroom notes and provide any functionality that makes this experience worthwhile.

The project is motivated towards solving collective problems faced by the members of the group over the course of many years of trying to organize and share classroom materials. We allow the bulk upload of classroom material to select audiences. This solves the problem of figuring out online storage and sharing notes across different student circles. Using OCR, we automatically infer meaning from these documents during upload time. This provides the scope to power search and find content, a major problem when dealing with physical notes. By creating study groups, we also create the scope for discussion and conflict resolution right at the point of note creation enabling easy referencing.

From a technology standpoint, the project aims at applying the design patterns and architectural patterns that we have come to learn over the course of the semester. The end product of this is a highly reliable, scalable and secure application build atop of AWS technologies. This combined with a minimalistic UI experience serves as a fully functional prototype of the overall idea.

This report goes through the various aspects of the project. First, we go over the architecture and the user workflows to understand the overall working of the application. We also try to go over some of the architectural choices and the specific problems they serve to address. Finally, we discuss the results and conclusions. These include learnings over the course of this semester-long project as well as further improvements.

TECHNOLOGIES:

We use AWS as the foundation to build our application. AWS is known to provide services that address key issues associated with applications of the modern-day. Leveraging AWS as a service helps provide industry quality standards to said problems without having to dive too much into the inner details. We make use of ***API Gateway and Lambda*** to provide a serverless architecture that is highly scalable. We were inspired by the checkout feature of Amazon which asynchronously processes post checkout operations. Since our application is dependent on third-party services that we do not provide a guarantee for, we decided to use ***S3, SQS, and CloudWatch*** to create a similar checkout like experience. This enables the user to seamlessly go about other tasks without having to bother about a spinning icon when their huge upload is going through. We use ***Cognito*** to provide the promise of security. Using ***ElasticSearch*** provides efficient means to query text. Since our application depends a lot on search, this is highly optimized to fit our necessity. Finally, we use ***SES*** to promote user experience by using push notifications when necessary.

Google Cloud Vision API is used to support the core functionality of the application namely Optical Character Recognition (OCR). OCR is used to infer knowledge from scanned handwriting/text which can be used in multiple ways to enhance the user experience. We go into more detail in the following sections.

USER WORKFLOWS:

We briefly discuss user workflows in this section. For cohesiveness, we discuss further workflow improvements in this section itself. Technological improvements and learnings are discussed in later sections.

Sign up/Login

Sign-on systems usually prove to be a hassle. By allowing Sign in using Google OAuth, we create a one-click sign-up/sign-in while automatically retrieving all user information we require in the most secure way.

Groups:

We use groups as the single starting point for all of our operations. We follow the WhatsApp's approach of being able to create a group, add users to the group and further follow up by sharing information onto that group. We employ this approach to inherit its simplicity and clarity of workflows.

Typically a new user would follow the following steps to get started with our application

- Create a group. The group would traditionally be aimed at a particular subject or a subset of a class that you would want to collaborate over.
- User Management:
    - User addition: Add users to a group by simply inputting an email address. Currently, existing users on the platform are valid inputs to this operation
      User Deletion: The creator of the group is allowed to delete users from the group.
      Improvements:
    - Ability to leave a group
    - Notification and permissions - Notifying a user they have been added to a group and successful addition to the group on acceptance

Documents:

Documents are representative of a collection of relevant pages - maybe the notes corresponding to a particular class/topic. Post creation of a group or the addition to a group, a user is allowed to share documents to those groups.

- Document addition: As discussed earlier, the starting point for most operations is a particular group. Once a group is selected we can upload a document (set of pages) to that group. Only the users of that group will be able to view and perform operations related to that document.
- Document deletion: The creator of the group is allowed to delete a document from a group if they no longer wish to share that material.

- Commenting:
    - Add comments onto a particular page of a document allowing for threads of discussion

<u>Search:</u>

The application is powered by search to make the process of information retrieval easier

- Document retrieval - Documents which match your search in terms of group title or description
- Page retrieval - Pages which have content or relevant comments that match that context of your search
- Suggestions - As the project is built from an educational standpoint, we use Stack Exchange API to get relevant content from Stack overflow.

ARCHITECTURAL DECISIONS:

1. Sign up/Login
    a. Use AWS Cognito to manage secure sign-up and login providing support for third-party login providers, making the onboarding process hassle-free
2. S3
    a. The application is aimed at handling massive amounts of images
    b. This could be potentially huge given most photos are clicked nowadays on high-resolution cameras
    c. S3 provides highly reliable/available storage which can be scaled as and when required
3. Google Cloud Vision API
    a. Handwriting recognition is a hard problem due to various factors. The Cloud Vision API provides good results on well-separated text and good image quality. The performance is extremely good on scanned text.
    b. As the objective of handwriting recognition is to infer text which powers the search, a moderate accuracy suffices for our application
    c. Since this is an external API, the workflow is well abstracted providing the ability to replace the Cloud Vision API with anything better in the future
4. DynamoDB/ElasticSearch
    a. DynamoDB - Manage all user related information and content.
    Documents are uploaded to groups users are part of and pages are smaller units within documents. We have created collections to easily query and update this information. Collections created:
        i. Persons : Stores user login information - mail id, name, groups
        ii. Documents : Stores all data relevant to a document - name and description of the document, pageIds, document owner, groups the document is part of
        iii. Pages : Stores information at the page level. As comments are at the page level, it contains the list of comment Ids, the document Id and the S3 URL of where the page is stored
        iv. Groups : Stores the information regarding documents and people part of the group including the group admin.
        v. Comments : Comments are added to pages. The information related to comments like the user who added the comments, the timestamp and the pageId to which the comment is added are stored here.
    b. ElasticSearch - Provides an efficient way to query over unstructured data including text representation of the documents/comments.
    We first query dynamoDB to get the pages and documents a user has access to and then filter the elastic search text query based on those Ids. Separate indices were created for documents and pages. This enables us to show the documents and pages separately in the search query results.
    Index info:

        i. documents : Stores the document name, description and documentIds. https://search-noteshare-khiicienqkujyqqz7vm4mmbwzm.us-east-1.es.amazonaws.com/documents/_search?
        ii. pages : Stores pageId, extracted text and comments. https://search-noteshare-khiicienqkujyqqz7vm4mmbwzm.us-east-1.es.amazonaws.com/pages/_search?

5. SQS
    a. The upload workflow is asynchronous and might be slow owing to huge file sizes
    b. Using SQS frees up the pipeline and user to carry out other operations
    c. This was used as a trigger for the lambda handling text extraction.
6. Notifications
    a. Handled using SES, enabling users to get important notifications and updates in case the document upload fails.
7. Scalability and reliability
    a. Use SQS to make upload workflow asynchronous
    b. Handle OCR failures : Retry the OCR call for 5 times in case of failures from Google OCR API. This was done by inserting the message back into the SQS queue in case of failures.
    c. Status flag in dynamoDB : Persist the meta-data details of documents and pages in dynamoDB and elastic before getting the extracted document details. The status of documents and pages are marked as 'ACTIVE' only after we get the extracted text information. This allows users to view the document meta-data and images uploaded before even getting the extracted text from OCR. Searching on the extracted contents is possible only after the status flag is 'ACTIVE'.'
8. CloudWatch
    a. CloudWatch metrics were leveraged to monitor application logs.

EXTERNAL LINKS:

Clickable Prototype:

https://projects.invisionapp.com/share/GTULQHT2CYA#/screens/390991909_Login

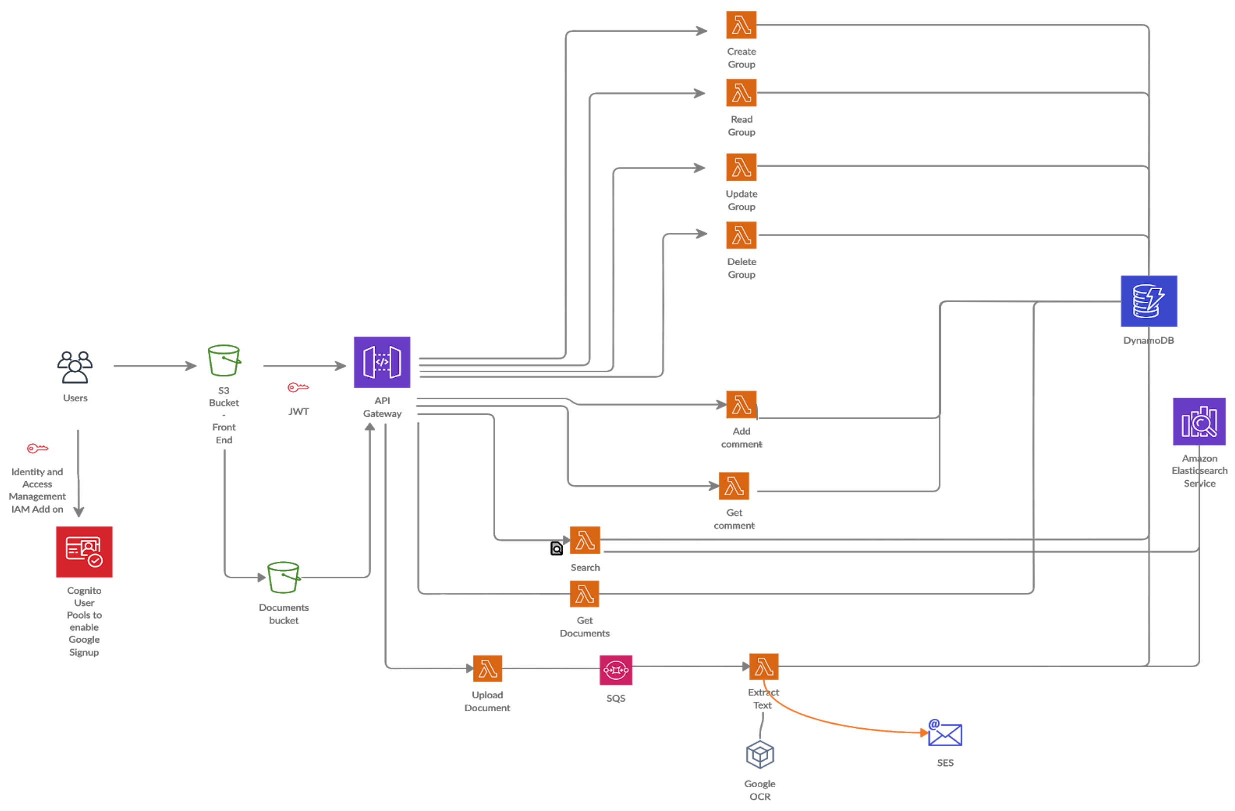
Swagger File:

https://drive.google.com/file/d/1QLzVmWhQt8EodajlyBDYU-M6HRvxbYsa/view?usp=sharing

Demo:

https://s3.amazonaws.com/frontend.noteshare/index.html

High level architecture diagram



RESULTS:

We were able to demonstrate the power of the application and future scope of work via a demo to the instructors. Future work in terms of workflows has been listed out in the previous section.

By following a methodical approach to building this web application we saved a lot of time and effort. By charting out an architecture and schema diagram before writing even a single line of code enabled us to have clarity of what we were trying to accomplish before diving into the finer details. Using Swagger as a tool for development allowed us to list out the API specs as per our architecture diagram. Also, seamless integration for API creation on API gateway helped us to conform to this spec. Further, usage of the API Gateway Javascript SDK ensured that these specs were followed right from the generation of an API request from the frontend to how a Lambda processes these requests on the backend.

A point of difficulty was extensibility. Once we had the initial APIs deployed on Gateway, we realized that we wanted to implement Authorizers for these APIs to only allow authorized requests from valid IAM users. In the given time frame, we weren't able to reconfigure Swagger to these specifications and re-deploy all our APIs. We ended up prototyping select APIs to implement Authorizers and could not extend this to the entire architecture.

All our ElasticSearch Clusters are currently public. Post Homework 3, we intend to extend our learnings about VPCs to this project. Since a lot of classroom material may be confidential to individuals, hosting our ElasticSearch cluster inside a VPC would make logical sense.

Lastly, a bottleneck would be the performance of Google CloudVision API. OCR is, in general, a hard problem to solve and we are limited as a platform by the accuracy of the OCR results that we get. We tested Amazon Textract as an alternative but did not get great results. Since the focus of the project was deploying a cloud architecture, we decided to go ahead with Google CloudVision API. This is implemented and used as a third-party service, which can easily be swapped with any new better service in the future

CONCLUSION:

The section focuses on the future improvements that can be incorporated:

1. Since we handle massive amount of text, we can employ NLP and ML techniques like text summarization and content based recommendation to add more features.
2. Scope for more collaborative features : Our idea of groups can be further enhanced by adding more features:
   a. Collaborative code editing,
   b. Video streaming
   c. Scheduling calendar events
3. This application has an immense potential from an educational perspective. Marketing it to reach a larger audience will help us get feedback and improve further.
4. Apart from the benefits it offers from a user viewpoint, this application is challenging from a technology perspective.