InANutShell

A cloud based application to Store.Listen.Learn

Akhil Movva - 40106477
dept. of Electrical and Computer Engineering (ECE)
Concordia University
Montreal, Canada
akhilmovva6852@gmail.com

Rishabh Singh thakur - 40106439
dept. of Electrical and Computer Engineering (ECE)
Concordia University
Montreal, Canada
thakurrishabhsingh280497@gmail.com

Ram Gopal Srikar Katakam - 40106010
dept. of Electrical and Computer Engineering (ECE)
Concordia University
Montreal, Canada
katakamsrikar96@gmail.com

Sai yaswanth Reddy kudumala - 40114453
dept. of Electrical and Computer Engineering (ECE)
Concordia University
Montreal, Canada
saiyaswanth.kudumala@gmail.com

***Abstract*— "In a Nutshell" is an easy to use web service and a one stop access to store, listen and learn the information from personal documents. Different types of documents (such as pdf, txt) can be stored and users can learn the information through concise summaries provided for each document. The summaries are presented as audio clips so that users can consume information on the go. Additionally, Amazon Alexa skill has been implemented so that users can listen to the audio clips through voice commands. The web application with these functionalities is realized using variety of tools provided by the cloud platform AWS and REST API services. For developing front end of the applications, web technologies HTML, CSS and JavaScript are used. The back end was built using python-based Django web framework and NoSQL MongoDB. This document discusses about the detailed architectural design, technical implementation and economics computed for developing "In a Nutshell" application.**

Keywords—AWS, Cloud Computing, Alexa, API, NoSQL, Serverless, python, lambda

I. INTRODUCTION

A. Problem Statement

Due to surge in the use of smartphones and laptops, there has been a growth of digital information in the past decade. A lot of information in the form of weblinks and documents (such as pdf, docx and txt) are created and stored by users in their devices. Despite, having access to lot of information, there has been no proper channel to keep track of personal this documents. It is also time taking, to read lengthy documents to get the desired information.

B. Goal & Objectives

One of the ways to address these needs is by designing a platform (such as web application), to organise the personal documents, providing the users with an option to see the summary and audio clips of each document so that they can consume the information on the go.

C. Assumptions

With the arrival of cloud computing platforms (such as Aws, Azure) that provide easy access to computing power and data storage, there has been a huge cloud adoption rate in small, medium, and large enterprises. Cloud platforms like AWS and Azure come with variety of machine learning and IOT tools that can be leveraged to make the applications smarter.

D. Methodology

To solve the problems mentioned above we propose a cloud-based web application "In A Nutshell". The front-end of the application is developed on Django web framework and for solving most of the backend requirements, AWS tools like S3 bucket and lambda are used. To develop some important functions like summary and categorizations of the documents REST API's are used. Voice Assistance Alexa is used to access the audio version of the documents.

This paper discusses the detailed methodology and implementation of "In A Nutshell" application. The web application automatically organises the personal documents of the users as per article genres and provides summaries and audio clips for each document. Sections II to III discuss the architecture and functionalities developed for the application and Sections IV discusses the technology stack used for implementation.

II. PROJECT DESCRIPTION

A. Function provided by the service

Inanutshell web application offers services for the users who are actively involved in reading documents from multiple sources i.e., pdf, text files, docx, ppt etc. The interaction flow of the app and its functionalities are as follows,

Authentication: All the users are authenticated with Gmail at the login page to provide secure access to their previously uploaded documents.

File uploads and storage: Upon successful authentication, user can successfully upload files. The uploaded files can be viewed in documents section of the site.

Categorization, Summarization and Audio generation: Once the document is uploaded by the user, finding the Category, summary of document and generation of its corresponding audio clip takes place.

Voice Interaction: To provide hands free experience for users to listen audio clips generated for each document summary an Alexa skill has been implemented. The Alexa skill is capable of login for authentication, able to list audio files uploaded, able to play requested audio file, able to play radio version (a shuffled version of summarized audio files).

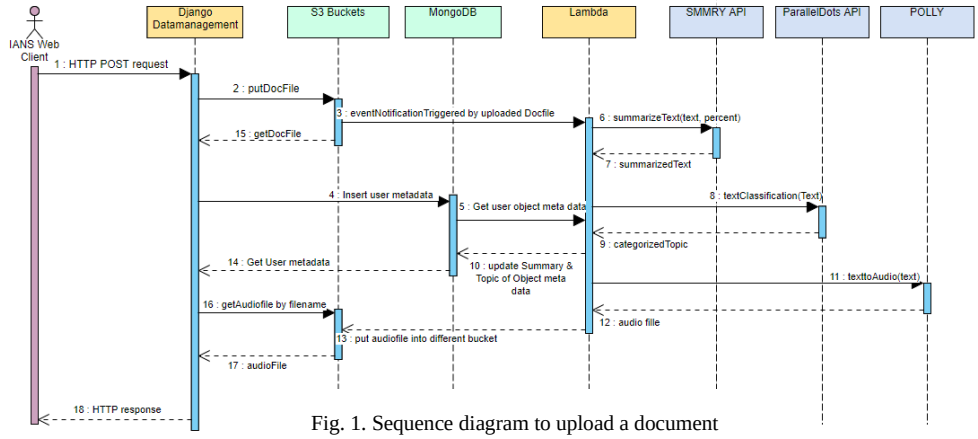


Fig. 1. Sequence diagram to upload a document

B. *Architectural design*

Fig.2 represents the architecture used for the "Inanutshell" application. The web application is served by Amazon Route 53 and users are authenticated using Google authenticator. The web application interacts with the backend using API gateway supported by AWS lambda function. Document processing is initiated by the API gateway, which triggers the lambda function to insert the document metadata in mongo DB. Documents are uploaded through web application and are stored in S3 bucket for bulk processing. After the upload of a document, S3 bucket triggers a second lambda function that supervises the processing. The second lambda function automatically categorizes, extracts the summary's and audios for all the documents in S3 bucket. "SMMRY" and "Parallel dots" API's are used for summarization and categorization, respectively. "AWS Polly" is used for converting summaries into audio files.

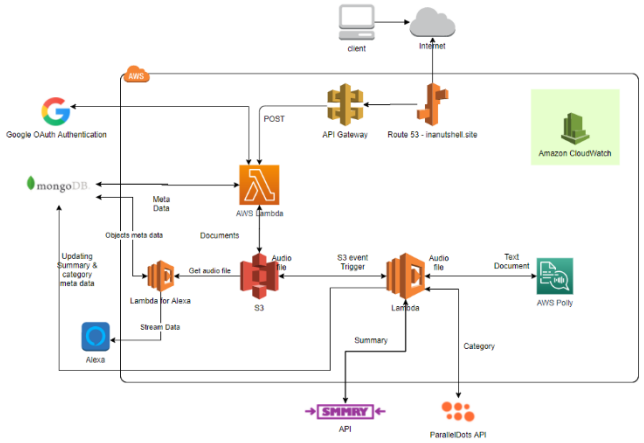


Fig. 2. Cloud Architecture Design

C. *Sequence Diagram:*

Fig.1 represents the sequence diagram developed for our application. The sequence diagram shows the sequence of interactions that happen for the use cases "Uploading the document" and "Getting audio files by name"

a. *Use-Case "Uploading the document":*

After uploading the document through web application, two functions "putdocfile" and "Insert meta data" are triggered to upload the document to S3 bucket and store the metadata in mongoDB. The "putdocfile" function triggers the other system lambda for getting summary and category name of the document. Lambda interacts with 'SMMRY' API through function summarizeText(text, percent) to get the 'summarizedText'. It also interacts with the 'Parallel Dots' API through function textClassification(text) to get the 'categorizedTopic'. After getting the 'summarizedText' and 'categorizedTopic', lambda interacts with the mongoDB to store the summary and object metadata.

b. *Use-Case "Getting Audio Files by name":*

For every document upload, lambda interacts with the polly system through the function texttoaudio(text) to fetch the 'audio file' of the document. After getting the 'audio file' lambda interacts with S3 system to store the audio files in different S3 buckets.

D. *Data Model*

We have represented the data of the user using json format. All the users meta data are inserted as documents in mongoDB cluster. The entities stored in metadata are username (string), user email (string), display filename (string), docs (fileField), category of the document (string), summary (string) and uploaded time (dataTimeField).


```
_id: ObjectId("5fc88a6ba1b4a8aaf88e1be5")
id: 112
username: "akhilmovva286"
user_email: "akhilmovva286@gmail.com"
filename: "test103"
docs: "test103.pdf"
tag: "BUSINESS"
summary: "Bitcoin has exploded back into the limelight this year, solidifying it..."
uploaded_at: 2020-12-03T06:49:15.497+00:00
```


Fig. 3. Sample Data Model of a user file object in the Mongodb cluster

E. *Technical implementation details*

a. *Cloud Provider:*

AWS (Amazon Web Services) [1] has been chosen as the cloud platform for hosting our application. A Serverless execution model has been chosen. In this approach the servers are fully managed by Amazon and the application is served based on invocations from clients. Each client request is served by a lambda function which only runs until the request is complete or for a maximum of 15 minutes.

The Serverless Architecture consists of an API Gateway, Lambda function and S3. The following AWS serverless technologies are used for the application:

1) API Gateway: The API Gateway is a HTTPS endpoint to which the clients communicate using HTTP methods such as GET, POST etc. The requests are forwarded to a lambda function which hosts the web application.

2) Lambda: The Lambda function is a serverless compute service provided by amazon. It is fully managed by amazon and can scale horizontally automatically based on the number of incoming requests. It only runs for the duration of the request or the set timeout or for a maximum of 15 minutes. In our architecture lambda processes the request forwarded by API Gateway and interacts with amazon S3 to fetch the static files required by the website such as the HTML, CSS and

JavaScript files which are then returned to the client via the API Gateway to display the web pages.

3) Amazon S3: It is a serverless storage service which can scale infinitely in terms of the storage capacity. It is being used to store flat files of our application such as HTML, CSS and JavaScript, user documents, audio files etc. Buckets are created to organize files. Each bucket has policies to control access to the files that it holds. All the buckets are private except for two buckets which store the audio files and summarized text documents.

4) Amazon Polly: It's an NLP (Natural Language Processing) tool-based service provided by amazon to convert text data to audio files. In our application the user text documents uploaded to s3 bucket trigger a lambda function which extracts the text data from the documents and send it to amazon Polly. The audio versions of the data is sent back to lambda by Polly which are then uploaded to another S3 bucket to be available for viewing on the web pages.

b. Front-End:

The front end of the web application was developed using the following technologies:

1) HTML: The structure of web pages were developed using HTML 5.

2) Bootstrap: The styling was done using a CSS framework called Bootstrap which provides beautiful templates for various web page elements which can be easily assembled to build a modern website.

3) JavaScript: To add behaviour to various elements vanilla JavaScript was used.

c. Back-End:

The primary backend programming language is Python 3.6.0. The back end of the website manages all the data and the following technologies have been used to build it:

1) Django: This is a python based back end web framework. This is the heart of the application which provides integration of the front end with the database, storage and rest of the external services used. Django[2] provides a framework for tasks such as handle databases in the form of models, map URLs to different resources, views to render html pages, admin to manage the website users.

2) Django-S3-Storage: This is a plugin to replace the default Django storage with amazon S3. Using this all the files stored by Django are by default stored in S3. In our application the files uploaded by users are stored by Django in S3 in a bucket dedicated for the user documents.

3) Djongo: This is a database engine which provides the capability of changing the default database of Django to mongo db. Our application uses MongoDB Atlas which is a remote MongoDB database hosted on server. With the help of pymongo and djongo[3] all the data that's stored in default SQL database of Django is converted to NoSQL format and written to MongoDB Atlas.

4) Boto3: This is Amazon SDK for python which is used to interact with various AWS services. Using boto3[4] library API calls are made to S3 for uploading or fetching data and Amazon Polly for converting text data to audio.

5) Django-all-auth: [5]It enables authenticating users using their existing social accounts. Using this library the users of our application are authenticated using their google accounts and the entire authentication process is handled by google and no user information is stored. The google provider is OAuth2 based. This makes our authentication mechanism highly secure.

d. Deployment:

The application is developed on a local computer first. A Django application is created on the local machine which encapsulates both front-end and back-end. This Django app is deployed to the AWS cloud in the form of a serverless Django application so that it integrates with other AWS services and entirely resides on the AWS cloud. To Achieve this the following python library was used:

1) Zappa: [6]It's a python library which allows deploying local Django applications to AWS cloud. It only requires specifying an S3 bucket for static files, AWS region for deployment and location of settings.py of the Django app. It takes care of the rest. Once initiated, it spins up AWS CloudFormation to deploy the required AWS services.

In our application it used CloudFormation to deploy an API Gateway, lambda to host the Django app, S3 bucket to host the website static files and automatically configured them into a fully functional serverless Django app. It returns an API Gateway endpoint link which can be used to access the application.

2) Route53: This is an AWS service which provides DNS routing to applications hosted on the AWS cloud. The domain name of our application was purchased from GoDaddy.com [7]. The name server records for our domain in GoDaddy.com were changed to those of Amazon.

Zappa provides the ability to automatically map a domain name to the API Gateway endpoint. By specifying the domain name and a certificate ARN (Amazon Resource Name) (obtained using AWS Certificate Manager) Zappa maps the domain name to the API Gateway endpoint using an Alias Record in Route53. This enables accessing the web application using a custom domain name.

e. Externel APIs and Services:

1) SMMRY API: This API uses natural language processing for summarizing the text documents uploaded by the users. A lambda function is triggered when a text document is uploaded to S3 bucket. This document is the given to the SMMRY API [8] in a HTTP POST request using python requests library along with an API key and the number of summarized output lines.

The API is used with a free account allowing 100 requests/day with a gap of 10 seconds between each request. The full account provides unlimited requests for a fee of 0.001$ for 1 credit and 500 characters consume 1 credit.

2) Parallel Dots API: This API [9] uses machine learning based text classification to generate a category label for a piece of text so that the uploaded text documents can be categorized into different sections such as sports, news etc.

It provides a 30-day free trial with 1000 hits/day. The starter plan provides 6000 hits/day, 60 hits/min, 15000 rows of text per month for a fee of 69$/ month.

3) MongoDB Atlas: This is a NoSQL database service provided on a remote server by MongoDB [10]. It allows the clients to store data on clusters. Our app uses this service to store document meta data such as filename, date uploaded etc and it is integrated with Django using the djongo engine.

The free plan provides database servers with 512mb of storage and shared RAM.

F. Alexa Technical Details

Alexa: For the purpose of development of the required Alexa skill [11], we had to design a voice interaction model for the Alexa to interact with the user. The voice interaction model gives the detail view of how Alexa behaves when a certain input is triggered and is the first step in designing the Alexa skill.

Invocation Name: This is a word letter combination that is required by Alexa to trigger your skill that you have designed. In our case, the invocation name would be "hey cloud".

Slots: Slot types define how phrases in utterances are recognized and handled as well as the type of data passed between components. For the purpose of the project we, have created a new SLOT naming AUDIOS which can accept user input on file names which is utilized in Intents.

Intents: Once the Alexa skill is triggered, we need to configure intents which are basically the activity you want to trigger by saying a phrase to Alexa. In our project, we have used a combination of custom and built-in intents to execute the required tasks. The difference between built-in and custom intents is that there is no training required by the model to find the required intent compared to using a custom intent. Below are the built-in intents used while developing the Alexa skill.

- AMAZON.CancelIntent: Triggered when Cancel command is passed to Alexa.
- AMAZON.HelpIntent: Triggered when Help command is passed
- AMAZON.StopIntent: Triggered when Stop command is passed
- AMAZON.PauseIntent: Triggered when pause command is passed
- AMAZON.ResumeIntent: Triggered when resume command is passed
- AMAZON.NextIntent: Triggered when next command is passed.

Lexa for custom Intent and the corresponding phrases used to train the model are:

- GetSmmryAudioIntent:

Phrases:

- summary of {audiofile} is
- summary of {audiofile}
- play summarized version of {audiofile}
- play summary of {audiofile}
- summarized version of {audiofile}

where {audiofile} is the name of slot value created of type AUDIOS mentioned above in SLOT.

- GetAudioIntent:

Phrased:

- play {audioQuery} file
- search for {audioQuery}
- find {audioQuery}
- play {audioQuery}
- start playing {audioQuery}

Where {audioQuery} is the name of the slot value created of type AUDIO mentioned above in SLOT.

- GetRadioIntent:

Phrases:

- play radio
- radio

- ListFileIntent:

Phrases:

- my list files
- my files
- show my list files
- show my list
- list files

Audio interfaces: Since the project involves playing audio files from s3 bucket, it is required that the audio interface is enabled.

Account linking: Use the allauth information available upon creating all-auth.

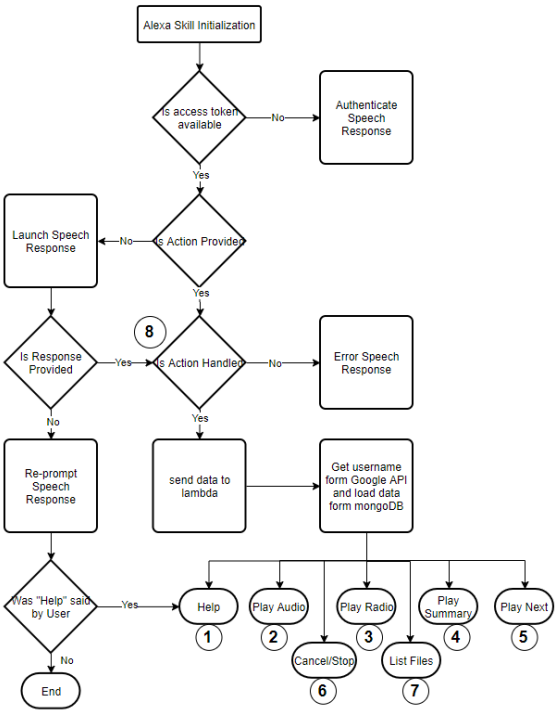
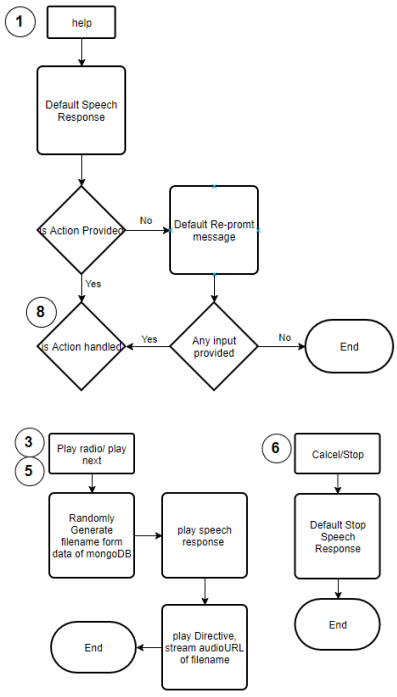
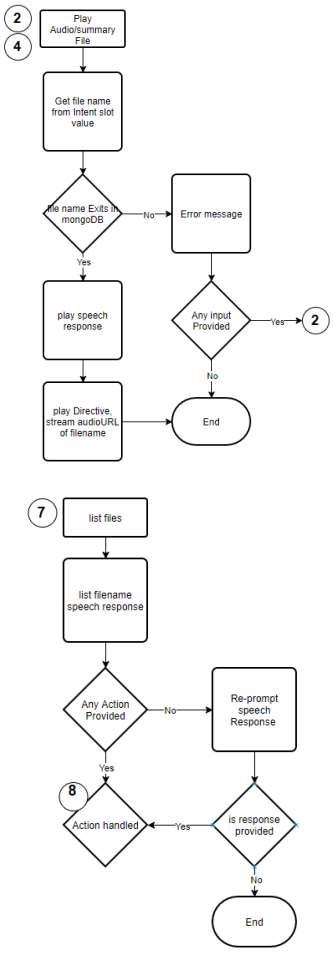


Fig. 4. Voice Interaction Model for Alexa skell (below is the model continuation of the above diagram with appropiate numbers)





For running the code backend on lambda function, we need to configure the lambda to get triggered upon Alexa activity. The lambda function is based on python 3.8 and Alexa ask SDK core and ask SDK is used to handle the intents generated from Alexa skill. Based on voice interaction model, we program the lambda function to behave according to it.

III. DISCUSSION

A. *Experience and Lessons learnt*

There were many challenges and lessons that have been learned during the development of the website from creating a website from scratch to deploying the whole website on cloud using serverless architecture and would like to present our journey below specifying how our idea started and has evolved overtime while implementing.

1) Website Development: For the purpose of building a website, we have chosen Django framework. Having little experience with Django led to difficulty in understanding how the backend and front-end components of the website interact, where we had explore the concept of model Template view for storing the data generated form the website in the backend database and viewing the data back to the website, it took quite a struggle in learning our understanding towards developing a basic website.

2) Storage Handling: During the testing phase of the website development, we were relying on local database called sql-lite, but for our deployment on cloud we needed a cloud-based storage for storing meta-data of the users files and the uploaded files, upon exploration we have decided to use object-based storage S3 as a file storage for our application and mongo dB for handling metadata. Initial problem we encountered while integrating these storages were, Django was designed to use sql database and mongo dB was a NoSQL database hence we needed to find a package which acts a s connecter between Django and mongo dB, which djongo, and it took a while for a proper integration. S3 on the other hand for file storage was fortunately easier to handle, where by just replacing default file storage variable in setting.py we could accomplish that.

3) Lambda and its interactions with API: As per our application design we needed a compute service which would only be active when there is a necessary file upload in s3 bucket, We came across lambda function (serverless compute service) in aws, were s3 event notification could be used to trigger the lambda function to perform the intended operations of interacting with necessary api's such as parallel dot, smmry, polly. Finding the right api source for our intented application took a while, since it involved exploring other alternatives to find the best fit for our application.

4) Architecture Design Shift: Our initial deployment was decided to be on aws EC2 and have been developing our website and architecture in accordance to the decided compute service, however to minimize the cost of operation and management of EC2, we. Tried to look for serverless compute service lambda, where with proper implementation could replace EC2 for website deployment. To deploy Django on lambda, we found a service called zappa which could be used to deploy the website serverless. There were many problems around version compatibility to each individual package from python version to mongo dB package (djongo) to fit properly under zappa service to be able to deploy on lambda, for example djongo package version used during website development was not compatible with lambda, the website couldn't deploy properly, with proper testing and version monitoring we were able to deploy the website successfully on lambda with api gateway and s3 bucket to store css static files .

5) Change in Voice interaction: During our project proposal, we came up with idea to use lex for voice interaction to access files, but later we found alexa to be more user friendly and easy to acess rather than lex due to large number of alexa devices avaible, hence there was a shift in developing in website voice interaction to alexa skill development with account linking for user authentication and personalization.

6) Authentication and user profiles: From the available choices for user authentication service , we choose google all auth for user authentication and create a profile when user logins through google. Access to google account information was done using google api to get necessary information from google.

B. *Total Expenses*

To evaluate the costs that will incur for the project a user base of 1000 per month has been assumed and all the charges for using the tools and API services are estimated accordingly monthly.

1) AWS Service costs:

For communicating the clients with the backend business logic, API gateway was used as an interface which will cost 0.0035 USD per 1000 requests. To monitor the health of the

application, Amazon CloudWatch is used at a monthly cost of 0.50 USD per log storage of 1GB. Memory of 0.125 GB is allocated for lambda function for a cost of 0.64 USD per month. To convert the summary of each document to audio amazon Polly services are used at a cost of 4 USD for million characters. All the documents of the users are securely stored in S3 buckets at a standard monthly cost of 0.11 USD for 1 GB. Besides this the service route 53 charges 0.5 USD for 1000 standard queries per month.

2) Non-AWS service Costs (API):

In addition to AWS services, "In A Nutshell" application uses 2 other API services "SMMRI" and "Parallel Dots" for summarizing and categorization of documents, respectively. "SMMRI" API charges 0.5 USD for summarizing a single document and "Parallel Dots" API is charged on monthly basis for 400 USD and can be used for unlimited requests. All the documents meta data is stored in a dedicated cluster of MongoDB Atlas at a charge of 57 USD per month.

3) Total Costs involved:

Tables I and II show the monthly costs incurred for all the tools used in "In a Nutshell" Application. Assuming a user base of 1000 per month, the application can be maintained at a cost of around 713 USD.

TABLE I. COST OF AWS SERVICES IN USD/MONTH

Aws Service	Costs Involved (USD)
API Gateway	0.035
CloudWatch	0.5
AWS Lambda	0.64
Polly	4

TABLE II. COST OF NON-AWS SERVICES IN USD/MONTH

Non-Aws Service	Costs Involved (USD)
SMMRI API	250
ParallelDots API	450
MongoDB Atlas (Cluster)	57

C. Quality Attributes

Our web service is fully serverless. Due to deployment of web application on AWS lambda all the quality attributes are based on features of lambda tool.

Performance features: Process lambda events within seconds so that the web application can be triggered with low latency to the users. The bottlenecks are the external service APIs whose performance depend upon the type of subscription.

Scalability and Availability: AWS Lambda uses replication and redundancy to provide high availability and horizontal scalability for the Lambda functions ensuring no downtime for the application and the ability to scale to any number of users through concurrent lambda executions.

Security: All data in transit is HTTPS providing secure connection. The data at rest is at MongoDB and S3. MongoDB provides its own encryption and data in s3 can be protected through different S3 encryption schemes. AWS Lambda stores code in Amazon S3 with encryption. It also performs additional integrity checks while code is in use.

D. Cloud Strengths and Limitations

1) Strengths:

- Pay only for what you use model - Serverless
- Horizontal Scalability
- High Availability
- Infrastructure as a service (IaaS)
- Rapid provisioning of resources (Low Latency)

2) Limitations:

- Provisioned Concurrency for Lambda limits its performance.
- API Gateway has throttling limits given a burst limit of 5,000 and an account-level rate limit of 10,000 requests per second.
- AWS Polly has 3000-character limit for text to audio conversion.
- SMMRY API free account limits 100 requests/day
- Paralleldots API free monthly trial limits 1000 requests/day

IV. TEAM MEMBER'S CONTRIBUTION

A. Akhil Movva – Full Stack Development

Akhil was responsible for all the front-end development. He implemented the Django and its integrations with front-end, AWS service and MongoDB for inserting and fetching data. He also configured route53 for DNS configuration and worked on ParallelDots API for classification in Lambda.

B. Rishabh Thakur – Cloud Deployment

Rishabh was responsible for the complete project deployment on AWS service Lambda using Zappa and mapped the API Gateway endpoint. He implemented Polly for audio conversion and SMMRY API's to summarize text on the lambda and maintained the AWS administration.

C. Srikar Katakam – Service Development

Srikar was responsible for the s3 storage handling using boto3 and voice interaction. He completely implemented the voice model on Alexa. He developed the Alexa skill with account linking for user authentication and personalization. He also helped in setting up Django.

D. Yaswanth kudumula – Data Management

Yaswanth is responsible for the google OAuth2 authentication using Django-All-Auth and setting up the MongoDB NoSQL database service using Djongo. He also helped in data management with s3 and service maintenance on the AWS cloud like Lambda.

V. FUTURE WORKS

We can give support to other document files like pptx, docx and web links through containerization using AWS Fargate. For there is a limit of 3000 characters for audio conversion using Amazon Polly, so we need use different method to convert more characters. And finally, we need to design our own custom signup page along with account management.

VI. SOURCE CODE

GitHub link for the Source Code:
https://github.com/AkhilMovva/Inanutshell

Recorded demos of our Web Service and Alexa skill:

https://www.dropbox.com/s/082ealj9ppema3k/inanutshell_demo.mp4?dl=0

https://www.dropbox.com/s/mhyl3uqrnf2qovx/alexa_demo.MP4?dl=0

VII. Web service Access

URL for our web service:
https://inanutshell.site/

References

[1] https://docs.aws.amazon.com/
[2] https://docs.djangoproject.com/en/3.1/
[3] https://pypi.org/project/djongo/
[4] https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
[5] https://developers.google.com/identity/protocols/oauth2
[6] https://www.agiliq.com/blog/2019/01/complete-serverless-django/
[7] https://ca.godaddy.com/
[8] https://smmry.com/api
[9] https://www.paralleldots.com/text-analysis-apis
[10] https://docs.mongodb.com/
[11] https://developer.amazon.com/en-US/docs/alexa/ask-overviews/build-skills-with-the-alexa-skills-kit.html
[12] https://medium.com/datadriveninvestor/how-to-django-with-mongodb-the-power-of-djongo-df92317f8714
[13] https://aws.amazon.com/blogs/machine-learning/build-your-own-text-to-speech-applications-with-amazon-polly/
[14] https://templatemo.com/