Notes Distribution System

Notes Distribution System is a comprehensive course management and note sharing platform. A user friendly web application for students to register for courses and receive uploaded notes from the instructor. The instructor gets a dashboard for each course where they can upload images of their handwritten notes. The application extracts text from the image and creates a text file. Registered students receive a link to this file which can be downloaded.

This application makes notes sharing easier for the instructor as the instructor does not need to draft comprehensive text notes for in class briefs. The notes written on the blackboard can be easily shared among students. The process is also made easier for the students as they receive a link of the notes through their email without fail.

Key Features and Target Performance:

1. Student Course Registration: Students can register to courses through their mail ID. A confirmation mail will be delivered and upon validation they will be registered successfully.
2. Course Dashboard: The dashboard allows instructors to upload handwritten notes in image format.
3. Automatic Text Extraction: Handwritten notes are converted to text using AWS Textract [1].
4. Notes Distribution: The link of the created text file is shared among all students registered to the course
5. Notes Distribution: The link of the created text file is shared among all students registered to the course through email.

Benefits:

1. Enhanced Note Sharing: Instructors can quickly share notes written on the blackboard as images. These images are automatically converted to text using AWS Textract [1], making it easier for students to access and study the content.
2. Time-Saving for Instructors: Instructors no longer need to spend time drafting comprehensive text notes for in-class briefs. They can simply upload images of their handwritten notes, which are then processed and shared with students in text format.
3. Seamless Email Notifications: The system automatically sends email notifications to students with links to the text notes. This ensures that students receive the notes without any manual intervention.
4. Scalable Infrastructure: The use of AWS Textract [1] for text extraction ensures scalability, allowing the system to handle a large number of courses and students efficiently.

5. Accessibility and Flexibility: As the notes are available in text format, students can access them from various devices and platforms. This enhances the accessibility and flexibility of the learning materials.

AWS Services

Services used in Notes Distribution System:

1. Compute:
    a. AWS Elastic Beanstalk [2]: Web Application Hosting.
    b. AWS Lambda [3]: Functions deployed to handle events when a student registers for course, instructor login, notes sharing, text extraction, etc.
2. Storage:
    a. AWS S3 [4]: Buckets that store the files uploaded by instructors and the extracted text file.
    b. AWS DynamoDB [5]: Stores course data and provides course login authentication.
3. Network:
    a. API Gateway [6]: Routes corresponding API to Lambda functions.
4. General:
    a. AWS SNS [7]: Share email containing link of extracted text to all students registered to course topic.
    b. AWS Textract [1]: Extract text data from file uploaded.
    c. AWS CloudFormation [8]: Infrastructure provisioning.

Service Comparison:

1. AWS Elastic Beanstalk:
    a. Elastic Beanstalk provides a more managed and straightforward approach to hosting web apps, while AWS Elastic Container Service and Elastic Kubernetes Service offer more control for containerized applications.
    b. Easier to deploy and manage applications compared to EC2.
    c. Provides support for multiple platforms and provides automatic scaling.
2. AWS Lambda:
    a. Compared to running virtual machines (EC2) or containers (ECS/EKS), Lambda is cost-effective as you only pay for the actual execution time of your code in milliseconds.
    b. Lambda functions are easy and quick to deploy. You upload your code, define the triggers, and Lambda takes care of the rest.
    c. Lambda integrates seamlessly with other AWS services, such as API Gateway, S3, DynamoDB, and more.
3. AWS S3:
    a. Amazon S3 is designed to scale effortlessly and can handle any amount of data
    b. S3 is designed to provide high availability, and data stored in S3 is accessible from anywhere at any time.

4. AWS DynamoDB:
    a. DynamoDB is a fully managed NoSQL database service.
    b. Designed for seamless horizontal scaling.
    c. Store and retrieve data without defining a table schema beforehand, providing flexibility in data modeling.
5. AWS API Gateway:
    a. API Gateway integrates seamlessly with various AWS services, such as Lambda, DynamoDB, S3, and more.
    b. API Gateway enables CORS support.
    c. API Gateway is designed to scale automatically based on demand, ensuring that your APIs can handle any level of traffic.
6. AWS SNS:
    a. SNS provides real-time message delivery with high reliability and low latency, ensuring that messages are delivered to subscribers immediately.
    b. Supports multiple message formats, including SMS, email, push notifications, and HTTP/HTTPS.
7. AWS Textract:
    a. Textract uses machine learning algorithms to achieve high accuracy in text recognition.
    b. It processes a wide range of document formats.

Deployment Model

Notes Distribution System uses the Public Cloud Deployment model. It uses services provided by Amazon Web Services to handle functions and perform necessary tasks.

Following is a comprehensive list of reasons of choosing public cloud deployment model:

1. Scalability: Public cloud providers offer virtually unlimited resources, allowing businesses to easily scale up or down based on demand. This elasticity ensures that applications can handle varying workloads without the need for large upfront investments.
2. Reliability and Redundancy: Public cloud providers maintain multiple data centers across regions, ensuring redundancy and high availability.
3. Speed of Deployment: With public cloud services I can rapidly deploy applications and services without the need for time-consuming hardware setup and configuration.
4. Automatic Backups and Disaster Recovery: Public cloud providers offer built-in backup and disaster recovery solutions, ensuring data integrity in case of unexpected events.
5. Flexibility: Public cloud services offer a wide range of tools, platforms, and services that cater to diverse use cases and workloads.

Delivery Model

Notes Distribution System is a software as a service. Students and instructors can use the platform without worrying about setting up servers, managing infrastructure, or performing updates. The application's core features, such as course registration, course dashboard, automatic text extraction, and notes distribution, are provided as a service over the internet, making it easy for users to access and utilize the platform.

The primary reason for choosing SaaS is due to high availability. The application can be used in universities and schools to list their own courses and maximize notes sharing. A subscription policy can provision the application to organzions. This subscription plan is a future scope of notes distribution system which I plan to work on if possible. New features and updates can be rolled out automatically, ensuring all users have access to the latest version of the application.

Architecture

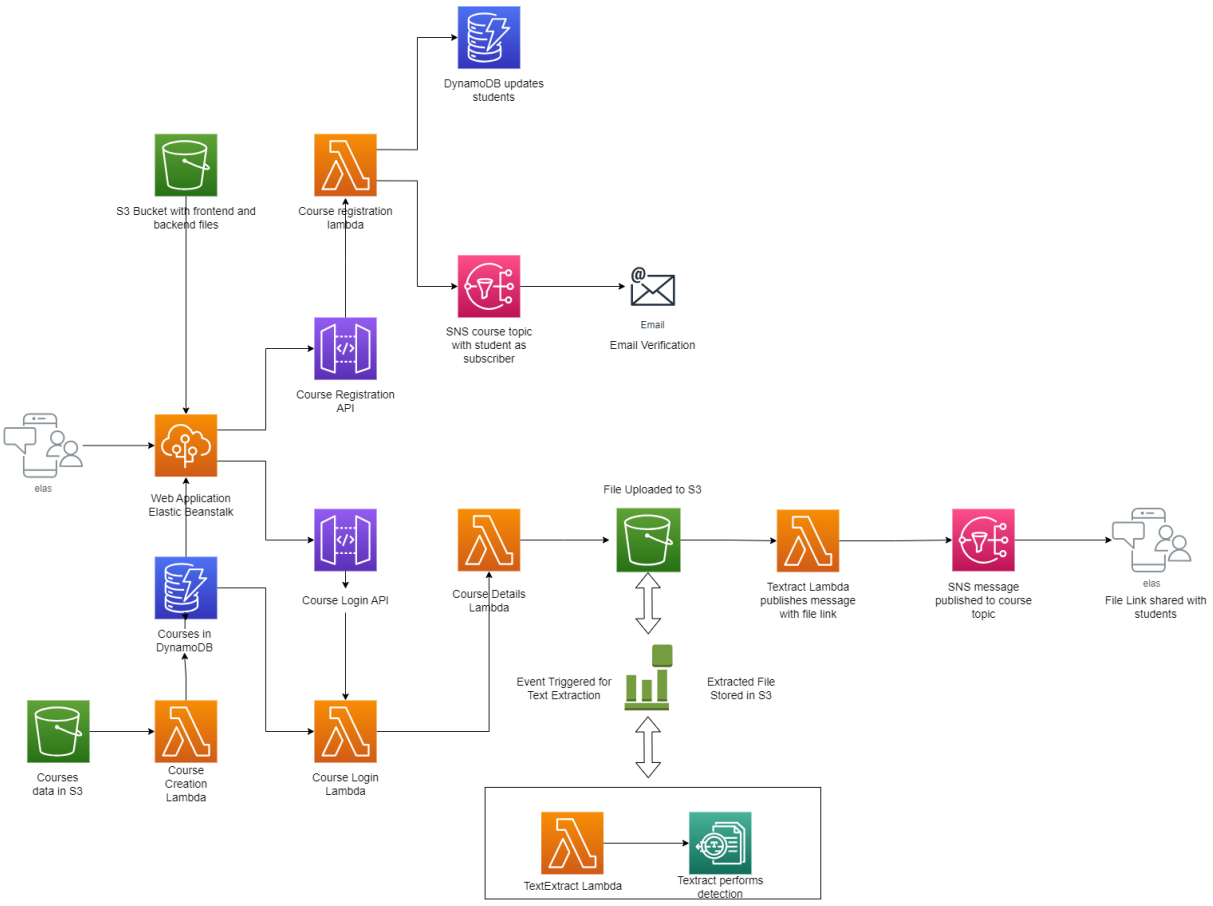

Figure 1: Notes Distribution System Architecture.

1. How do all of the cloud mechanisms fit together to deliver your application?
    a. An S3 file containing the frontend web application and backend lambda functions is used to provide all resources used by cloud formation to setup my application.
    b. The elastic beanstalk web application loads up and displays all the courses created using a lambda function with data stored in S3 bucket.
    c. A user registers for any of the courses listed providing their mail ID.
    d. A POST request is made to my registration API which triggers the courseRegister lambda. This lambda stores the data and creates an SNS topic if it doesn't exist and adds the email ID as a subscriber.
    e. The user receives an email to verify their subscription to the course.
    f. Similarly, instructors can login to the application through the course login feature provided on the dashboard.
    g. They are required to enter the course ID and password which uses the login API to trigger the login lambda function. This lambda function performs email ID and password verification.
    h. The instructor is then redirected to the course dashboard where course details are listed. They can then upload images of their handwritten notes which get stored in the respective course folder in S3 bucket.
    i. This file upload on S3 triggers an event to my textExtract lambda function. The function uses Textract to extract text from the uploaded image and creates a text file with the data. The file is stored in the course's output folder in the same S3 bucket.
    j. The lambda creates a link with this file and publishes a message to the course SNS topic with the file link.
    k. The subscribed users get a link to their extracted notes which they can then download.

2. Where is data stored?
    a. The course related information such as course ID, password, instructor and course name is stored in 'Courses' DynamoDB table.
    b. The files uploaded by instructors are stored in an S3 bucket. The same bucket is used to store the extracted files.

3. What programming languages did you use (and why) and what parts of your application required code?
    a. For the frontend web application I have used node.js [11] with styling provided by bootstrap [12].
    b. I used react because it's easier to integrate with elastic beanstalk. React applications on elastic beanstalk can also easily leverage services like S3 for storage, API Gateway for creating APIs, and more.
    c. The backend of the application is done with individual lambda functions using Python 3.10 [13].

d. The reason for using python was the extensive libraries supported by python for Textract and other functionalities. If using node.js I might have had to create deployment packages with dependencies to perform the same functions which were much easier to import using AWS lambda.

Data Security

Notes Distribution System is not a highly secure application because I have provided public access to the S3 bucket containing all the files. The reason for public access is as follows:

1. IAM policy restriction: I was facing issues with assigning policy to my lab role for specific actions. I tried creating another role but figured it was causing problems triggering the texExtract function.
2. CORS policy: Even after managing CORS policy to allow my headers the file creation function was still giving me a blocked error caused by CORS. So I had to give Get and Put permissions to my header in the S3 bucket properties.
3. File Access: My application distributes the link of the file created to all students registered to the course. Which required that objects have public access allowing students to receive accessible file links.

How could the vulnerabilities be addressed in the future?

1. To lift the IAM policy restriction I would like to create a separate role which provides S3 access to create a separate role which interacts with the textExtract lambda. This role will have only the policies needed and be provided to registered users only.

Apart from the S3 bucket security constraint Notes Distribution System has solid security mechanisms in place. They are as follows:

1. Authentication and Verification: The use of AWS DynamoDB for course data and login authentication is implemented securely.
2. Secure APIs: API Gateway are configured to use HTTPS, and API endpoints are properly authenticated and authorized to prevent unauthorized access to Lambda functions.
3. Monitoring and Logging: AWS CloudWatch is used to monitor Lambda function execution, API Gateway logs, and other critical system components.

Cost of implementing the same application with a private cloud:

What would your organization have to purchase to reproduce your architecture in a private cloud while providing relatively the same level of availability as your cloud implementation?

Private cloud deployments typically involve purchasing and maintaining physical hardware, networking equipment, and software licenses, which can lead to higher upfront costs compared to using a public cloud service like AWS. The cost of implementing the Notes Distribution System on a private cloud can vary significantly depending on various factors. The factors are as follows:

1. Hardware Costs

2. Software Licenses
3. Infrastructure Management
4. Security and Compliance
5. Scaling and Capacity Planning
6. Backup and Disaster Recovery
7. Network Connectivity
8. Personnel

To create an on premise server for Notes Distribution System the cost would be around $10,000 - $15,000 [14].

Monitoring Mechanism

Monitoring AWS costs is essential to ensure that expenses remain under control and within the allocated budget. Here are some strategies and tools I can use to monitor AWS costs effectively:

1. AWS Cost Explorer: AWS provides a built-in tool called Cost Explorer that allows you to visualize, analyze, and forecast your AWS costs. It provides access to cost and usage reports, which I can use to track expenses across various AWS services.
2. AWS Budgets: I can use AWS Budgets to set custom cost and usage budgets for specific AWS accounts, services, or tags. AWS Budgets can send me email or SMS notifications when costs exceed predefined thresholds.
3. AWS Billing Alerts: I can set up billing alerts using AWS Simple Notification Service (SNS) to receive notifications when costs reach specific thresholds.

The cloud mechanism that can shoot up costs if not monitored regularly would be AWS Lambda. AWS Lambda is used to handle various events and functions in the system, such as when a student registers for a course, instructors upload notes, share notes with students, and perform text extraction using AWS Textract. If Lambda functions are not optimized or monitored, the costs can increase rapidly due to the following reasons:

1. Invocation Frequency: If Lambda functions are invoked frequently due to high traffic or frequent events, it can lead to increased costs based on the number of requests and duration.
2. Resource Utilization: Lambda functions consume resources such as memory, CPU, and execution time. If the functions are not optimized to use resources efficiently, it can lead to higher costs.
3. Large Payloads: If the functions process large payloads, it can consume more resources and increase costs.
4. Cold Starts: Lambda functions experience cold starts when they are invoked for the first time or after a period of inactivity. Cold starts can add latency and increase costs due to initialization time.

Future Scope

How would your application evolve if you were to continue development? What features might you add next and which cloud mechanisms would you use to implement those features?

1. Subscription Model: I wish to set up a subscription model for the Notes Distribution System. In this schools and universities can set up their own courses on the application. This platform can be made available to their students who can register and receive extracted notes.
2. Multimedia Support: I want to extend support for multimedia content, allowing instructors to upload and share videos, audio recordings, or interactive media along with their notes.
3. Analytics and Insights: Implement analytics and reporting capabilities to provide instructors with insights into how students interact with their notes and the platform. This can help instructors improve their teaching methods and materials.
4. Amazon Rekognition: This can help automatically tag notes with relevant keywords, making it easier for students to search and find specific content within the notes.
5. Amazon Polly: This can enable students to listen to the notes as an audio format, aiding those with visual impairments or those who prefer audio-based learning.
6. Amazon Translate: This can allow instructors to upload notes in different languages, making the platform accessible to a broader audience.

Gitlab URL:
https://git.cs.dal.ca/courses/2023-summer/csci4145-5409/mudra/-/tree/main/Project

References

1. "AWS Textract", *aws*. [Online], Available: https://aws.amazon.com/textract/ [Accessed: 23th August, 2023].

2. "AWS Elastic Beanstalk", *aws*. [Online], Available: https://aws.amazon.com/elasticbeanstalk/ [Accessed: 23th August, 2023].

3. "AWS Lambda", *aws*. [Online], Available: https://aws.amazon.com/lambda/ [Accessed: 23th August, 2023].

4. "Amazon S3", *aws*. [Online], Available: https://aws.amazon.com/s3/ [Accessed: 23th August, 2023].

5. "AWS DynamoDB", *aws*. [Online], Available: https://aws.amazon.com/dynamodb/ [Accessed: 23th August, 2023].

6. "AWS API Gateway", *aws*. [Online], Available: https://aws.amazon.com/api-gateway/ [Accessed: 23th August, 2023].

7. “Amazon SNS”, *aws*. [Online], Available: https://aws.amazon.com/sns/ [Accessed: 23th August, 2023].

8. “AWS Cloud Fomation”, *aws*. [Online], Available: https://aws.amazon.com/cloudformation/ [Accessed: 23th August, 2023].

9. “AWS Elastic Container Service”, *aws*. [Online], Available: https://aws.amazon.com/ecs/ [Accessed: 23th August, 2023].

10. “AWS Elastic Kubernetes Service”, *aws*. [Online], Available: https://aws.amazon.com/eks/ [Accessed: 23th August, 2023].

11. “React”, *react*. [Online], Available: https://react.dev/ [Accessed: 23th August, 2023].

12. “Bootstrap”, *Bootstrap*. [Online], Available: https://getbootstrap.com/ [Accessed: 23th August, 2023].

13. “Python”, *Python*. [Online], Available: https://getbootstrap.com/ [Accessed: 23th August, 2023].

14. “How Much Does It Cost To Build Cloud Computing Service?”, *CustomerThink*. [Online], Available: https://customerthink.com/how-much-does-it-cost-to-build-cloud-computing-service/ [Accessed: 23th August, 2023].