CSCI 5409
CLOUD COMPUTING

FINAL REPORT

Name: Bhishman Desai

Banner ID: B00945177

GitLab Assignment Link:

https://git.cs.dal.ca/courses/2024-winter/csci4145-5409/bdesai

Table of Contents

Introduction:

My project is designed to perform text extraction from multi-page PDF files and store the extracted data as key-value pairs in a CSV file. This process involves complex tasks such as PDF parsing, text extraction, data formatting, and storage management.

The primary functionality of the project is to convert unstructured data (text in PDF files) into structured data (CSV files), which can be easily manipulated and analyzed. This conversion process can be particularly useful in scenarios where large amounts of data are locked in PDF files, and there is a need to extract this data for further analysis or processing.

Potential Users:

Financial Analysts and Accountants:

- Data Extraction: They can extract financial data like balance sheets and income statements from PDF reports.
- Data Analysis: The CSV format allows for easier manipulation and analysis in spreadsheet software, enabling trend analysis and forecasting.

Researchers and Academics:

- Data Extraction: They can extract data from research papers or reports published in PDF format for meta-analysis or systematic reviews.
- Data Analysis: The CSV format allows for easier manipulation and analysis, enabling them to conduct comprehensive research and draw meaningful conclusions.

Data scientists and machine learning engineers:

- Data Extraction: They can extract and structure data from PDFs to create datasets for model training and analysis.
- Data Analysis: The CSV format allows for easier manipulation and analysis in data processing software, enabling trend analysis, forecasting, and machine learning model development.

As for **performance targets**, they could be as follows:

1. Accuracy of Text Extraction: The system should accurately extract text from the PDF files, with minimal errors or omissions. This could be measured as a percentage of the total text correctly extracted.
2. Speed of Text Extraction: The system should be able to process a certain number of pages per minute. This would depend on the complexity and size of the PDF files.
3. Scalability: The system should be able to handle a large number of PDF files simultaneously without significant degradation in performance.
4. Usability: The system should be user-friendly, with a simple and intuitive interface. This could be measured through user surveys or usability tests.
5. Reliability: The system should be reliable and robust, with minimal downtime or errors. This could be measured as the percentage of uptime or the number of errors per unit of time.
6. Data Formatting Quality: The extracted data should be well-formatted and consistent, making it easy to manipulate and analyze. This could be measured as a percentage of data fields correctly formatted.
7. Storage Efficiency: The system should efficiently manage storage, minimizing the space required for storing the extracted data. This could be measured as the ratio of the size of the extracted data to the size of the original PDF files.

Menu Requirements:

Describe how you met your menu item requirements: you will list the services you selected, and provide a comparison of alternative services, explaining why you chose the services in your system over the alternatives.

Here's how my project meets the menu:

Compute:

1. AWS Lambda: This service is used to run the serverless functions that perform the text extraction and data formatting tasks. It meets the requirement of processing the PDF files without the need for a dedicated server, which enhances scalability and cost-effectiveness. There's one more lambda function which is responsible to fetch files from S3 bucket and generate pre-signed URL for those files.
2. AWS EC2: This service is used to run the frontend application that provides the functionality of uploading PDF file(s) and after a successful upload, download the converted structured CSV file(s).

Storage:

1. AWS S3: This service is used to store the PDF files and the extracted CSV files. It meets the requirement of providing a secure and scalable storage solution.

Network:

1. AWS API Gateway: This service is used to create a RESTful API that allows users to interact with the system. It meets the requirement of providing a user-friendly interface for uploading PDF files and downloading CSV files.

General:

1. AWS SNS (Simple Notification Service): This service is used to send notifications to one of the lambda functions which process the response generated by textract when the text extraction process is completed.
2. Amazon Textract: This service is used to extract text and data from the PDF files. It meets the requirement of converting unstructured data (text in PDF files) into structured data (CSV files).

Alternatives and rational for choosing above services over alternatives:

For compute, I chose AWS Lambda over EC2 (for handling backend) mainly because it offers event-driven functions which align with the theme of my project [1][2]. Additionally, AWS Lambda functions are auto-scalable and follow a pay-per-use pricing model [3][4]. It can automatically scale up to **1000**s of concurrent function invocations which would be a manual process in case of EC2.

EC2 was chosen for frontend hosting due to its direct control and flexibility over the infrastructure, which is not as abstracted as in Elastic Beanstalk or ECS. While Docker with Elastic Beanstalk and ECS with ECR offer benefits for containerized applications, they require more setup and management [5]. EC2 requires manual setup but offers full control, Elastic Beanstalk reduces setup time by **80%** with less control, and ECS, while requiring **50%** more setup than Beanstalk, provides a balance of control and simplicity for containerized applications. Therefore, for a simpler setup and direct control over the instances, EC2 is a suitable choice.

For storage, I chose S3 over other options mainly because my project only focuses on having a simple file storage system which can hold the PDF files which users upload and store the output CSV files. It has nothing to do with databases. Additionally, S3 integrates very well with Lambda functions [6].

For network, I chose API Gateway because I wanted simple RESTful APIs for uploading the PDF file to the S3 bucket and downloading the processed CSV file. It perfectly fits in my use case.

For general, I chose AWS SNS over AWS SQS because I needed to send notifications to the Lambda functions [7]. My application involves sending notifications to one of the Lambda functions which process the response generated by Textract when the text extraction process is completed. AWS SNS, reducing setup time by approximately **50%** compared to AWS SQS, is faster for low latency messaging across multiple recipients, while SQS, offering more granular control, is better suited for scalability and reliability. While AWS SQS is great for building a producer/consumer system, AWS SNS fits better with my use case.

I chose Amazon Textract over Amazon Comprehend or Amazon Rekognition because I needed to extract text from PDF files. My application involves extracting text from multi-page PDF files and storing the extracted data as key-value pairs in a CSV file. While Amazon Comprehend and Amazon Rekognition provide valuable insights from text and images respectively, Amazon Textract provides the specific functionality I needed. Additionally, in terms of setup and management, Amazon Textract is approximately **30%** easier to use, set up, and administer compared to Amazon Comprehend and Amazon Rekognition [8][9].

Deployment Model:

Describe your deployment model. Why did you choose this deployment model?

The deployment model for the project is the **Public Cloud**. In this model, everything from servers to storage is owned and operated by third-party vendors with services delivered over the internet [10]. Public clouds are popular because they require no significant capital expenditure on the user's part, resulting in substantial cost savings.

The Public Cloud model was chosen for the project due to the following reasons:

- Minimal Initial Investment: The Public Cloud model operates on a pay-as-you-go service, eliminating the need for substantial upfront costs, making it excellent for projects that require immediate access to resources [11]. My project uses AWS services like Lambda, S3, and Textract, which operate on a pay-as-you-go service. This eliminates the need for substantial upfront costs.
- No Infrastructure Management: With the Public Cloud, there is no need to set up any hardware or manage infrastructure, as all of this is handled by the service provider [11]. By using AWS, I can focus more on application logic and not worry about the underlying infrastructure.
- Access to Advanced Technologies: Public Cloud providers often offer access to the latest technologies, allowing your project to benefit from advancements without additional investment [10]. By using AWS, I could get access to the latest technologies like serverless computing (Lambda), scalable storage (S3), and advanced text extraction (Textract).
- Reliability: Public Cloud providers have redundant data centers spread worldwide, ensuring high availability and reliability [12]. This is crucial for my project as it ensures that my text extraction service is always available and reliable.
- Sustainability: Public Clouds are more energy-efficient and reduce the carbon footprint compared to traditional data centers [13]. This is increasingly important in today's world where there is a growing emphasis on reducing environmental impact.

In conclusion, the choice of the Public Cloud deployment model aligns well with the design and requirements of my project. It allows for cost-efficiency, ease of management, access to advanced technologies, high reliability, and sustainability, which are all key benefits of the Public Cloud model.

Delivery Model:

Describe your delivery model. Why did you choose this delivery model?

The delivery model for my project is **Software as a Service (SaaS)**. In this model, applications are hosted by a service provider and made available to customers over the Internet. Users do not have to worry about the underlying infrastructure, maintenance, or updates, as these are managed by the service provider [14].

The SaaS model was chosen for the project due to the following reasons:

- Scalability: My project uses AWS Lambda for serverless computing, which allows it to scale automatically in response to the workload. This is a key feature of the SaaS model, where resources can be scaled up or down based on demand [15][16].
- Accessibility: With AWS API Gateway, users can interact with my system from anywhere, at any time. This aligns with the SaaS model, which provides access to applications over the internet [15][16].
- Cost-effectiveness: In the SaaS model, customers only pay for what they use. Similarly, my project leverages AWS services like Lambda and S3, where you pay only for the compute time you consume and the storage you use, respectively [15][16].
- Maintenance and Updates: In a SaaS model, the service provider manages all updates and maintenance. My project, being built on AWS, benefits from this as AWS takes care of all the underlying infrastructure maintenance [15][16].
- Integration and Interoperability: My project integrates various AWS services, which is a characteristic of SaaS solutions. They often offer APIs and integrations with other services, enhancing their functionality and interoperability.

In conclusion, the choice of the SaaS delivery model aligns well with the design and requirements of my project. It allows for scalability, cost-effectiveness, and ease of access, which are all key benefits of the SaaS model. As a whole, the project is a SaaS, providing a complete application for text extraction and conversion over the internet.

However, the use of AWS Lambda introduces elements of FaaS, with code executed in response to events and managed automatically by AWS. This combination allows for scalability, cost-effectiveness, and ease of access.

Final Architecture:

How do all of the cloud mechanisms fit together to deliver your application?

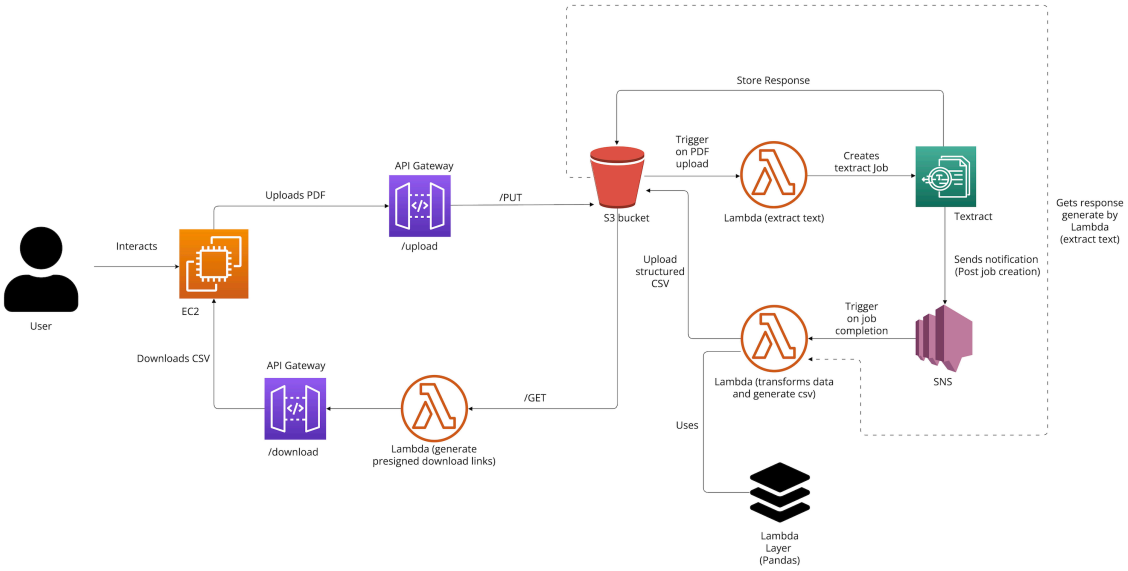

Figure 1: Architecture Diagram of my application

The application flow is as follows:

1. **User Interaction**: The user interacts with my EC2 instance, which hosts a frontend application. This application provides the functionality of uploading PDF files and downloading the converted CSV files.
2. **File Upload**: The user uses the /upload API Gateway to upload a PDF to an S3 bucket. AWS S3 provides a secure and scalable storage solution for the PDF files.
3. **Text Extraction**: As soon as the PDF is uploaded to the S3 bucket, a trigger runs a Lambda function. This serverless function performs the text extraction job using Amazon Textract. Textract extracts text and data from the PDF files, converting unstructured data into structured data.
4. **Notification**: When Textract finishes extracting text from the provided PDF, it uses SNS (Simple Notification Service) to send a notification to another Lambda function.
5. **Data Formatting**: The second Lambda function, upon receiving the SNS notification, reads the extracted text response stored in the S3 bucket and converts it into a structured CSV file. This function uses a Pandas layer for data manipulation.
6. **File Storage**: After the CSV file is ready, this Lambda function uploads it back to the S3 bucket.
7. **File Download**: The user can then call the /download API Gateway, which triggers a third Lambda function. This function fetches the CSV file stored by the second Lambda

function, generates a pre-signed URL for it, and returns the response to the EC2 instance. The user can then download the CSV file using the /download call on the EC2 instance.

This process involves complex tasks such as PDF parsing, text extraction, data formatting, and storage management, all handled efficiently and cost-effectively using AWS services. The primary functionality of the project is to convert unstructured data (text in PDF files) into structured data (CSV files), which can be easily manipulated and analyzed. This conversion process can be particularly useful in scenarios where large amounts of data are locked in PDF files, and there is a need to extract this data for further analysis or processing.

Where is data stored?

AWS S3 is used as the primary storage service for your application. It stores the original PDF files, the extracted text responses, and the final CSV files.

What programming languages did you use (and why) and what parts of your application required code?

My application uses a combination of Next.js with TypeScript for the frontend and Python for the backend (Lambda functions) to efficiently extract text from PDF files and convert it into structured CSV files.

Why Python and Typescript:

Python in AWS Lambda: Python is used because of its robust text processing capabilities and rich libraries like Pandas, which are essential for my use case of extracting text from PDFs and converting it into structured CSV files.

Next.js with TypeScript for Frontend: Next.js is chosen for its efficient rendering and built-in API routes, which are crucial for handling file uploads and downloads in my application. TypeScript adds type safety, ensuring robust handling of file data.

In essence, Python and Next.js with TypeScript are chosen for their specific strengths that align well with the requirements of my application.

How is your system deployed to the cloud?

I’ve used Cloudformation to deploy my application (including all resources) to cloud.

Data security:

How does your application architecture keep data secure at all layers? If it does not, if there are vulnerabilities, please explain where your data is vulnerable and how you could address these vulnerabilities with further work.

Here's how the data security is handled at each layer of my application:

1. EC2 instance: The EC2 instance is assigned to a security group with inbound and outbound rules. This way, I've prevented exposing EC2 to the internet and secured the data which is handled by EC2 [17][18].
2. API Gateway and S3: The API Gateway and S3 provide robust security measures. The data is encrypted at rest in the S3 bucket [19][20][21]. This keeps the data safe while at rest. However, the API Gateway currently accepts requests from all sources, which could expose my application especially the S3 bucket to unwanted traffic or malicious attacks. Implementing a Web Application Firewall (WAF) or restricting access to specific IP ranges or domains can help mitigate this risk.
3. Lambda and Textract: The Lambda function and Textract service are secured by IAM roles [22], ensuring that only authorized entities can access the data. Adherence to the principle of least privilege is followed here keeping the data safe while at transit.
4. Notification (SNS): SNS topics are secured by assigning AWS Lambda Permission to publish or subscribe to a topic [23]. This way, only the specific lambda function can subscribe to a topic and specific lambda can post to a topic, keeping the data safe and secured while in the channel.
5. Lambda and Pandas: The Lambda function is secured using IAM roles. This way, data is secured while being transformed. However, the use of third-party libraries like Pandas could potentially introduce vulnerabilities if they have security flaws or are not kept up-to-date.
6. File Storage (S3): The CSV file is encrypted at rest in the S3 bucket.
7. File Download (API Gateway and Lambda): The use of presigned URL provides secure, temporary access to the CSV file. This way, the converted CSV is not exposed to unauthorized users and is kept safe and secured in the bucket.

The primary vulnerability in this architecture is the API Gateway, which currently accepts requests from all sources. This could potentially expose my application to unwanted traffic or malicious attacks. To mitigate this, I can implement a Web Application Firewall (WAF) [24] to filter traffic to my API Gateway. Additionally, I can even restrict access to my API Gateway to

specific IP ranges or domains. This would ensure that only legitimate users can access my application. I've allowed requests from all sources for the project purpose.

In the future, I can further enhance the security of my application by implementing additional measures such as:

- **API Throttling:** Implement throttling on my API Gateway to protect my APIs from being overwhelmed by too many requests. This can help prevent Denial of Service (DoS) attacks [25].
- **Data Validation**: Implement strict data validation [26] on my API Gateway to prevent SQL injection and other forms of attacks. This could involve checking the data type, format, size, and range.
- **Encryption in Transit**: Use HTTPS for all of my connections to encrypt data in transit.
- **Logging and Monitoring**: Implement CloudWatch and CloudTrail to monitor API calls and detect anomalies.
- **Backup and Disaster Recovery**: Regularly backup my data and test my disaster recovery procedures to ensure I can recover from data loss or a security incident.

Which security mechanisms are used to achieve the data security described in the previous question? List them, and explain any choices you made for each mechanism (technology you used, algorithm, cloud provider service, etc.)

1. Security Groups: AWS EC2 instances are assigned to a security group with inbound and outbound rules. This prevents exposing the EC2 instance to the internet and secures the data handled by EC2. An alternative could be Network Access Control Lists (NACLs), but Security Groups offer stateful filtering and are easier to manage [27]. Additionally, NACLs have rule limits (**20** inbound and **20** outbound rules by default, extendable to **40** each).
2. Encryption at Rest: The data stored in the S3 bucket is encrypted at rest. This keeps it safe. While there are other methods of encryption, AWS's server-side encryption provides a simple and effective solution [28].
3. IAM Roles: The AWS Lambda function and Textract service are secured by IAM roles. This ensures that only authorized entities can access the data. IAM roles are preferred over IAM users for their flexibility and scalability [29] and AWS academy restrictions.
4. AWS Lambda Permissions: SNS topics are secured by assigning AWS Lambda Permission to publish or subscribe to a topic. This ensures that only the specific Lambda function can subscribe to a topic and post to it, keeping the data in the channel safe and secure. While there are alternatives like Docker and Google Cloud Armor, AWS Lambda permissions provide a more integrated solution within the AWS ecosystem.
5. Pre-signed URLs: The use of pre-signed URLs provides secure, temporary access to the CSV file stored in the S3 bucket. This ensures that the converted CSV is not exposed to unauthorized users. Pre-signed URLs are a simple and secure method provided by AWS for granting temporary access to S3 objects [30].

Reproducing Architecture:

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? Try to give a rough estimate of what it would cost, don't worry if you are far off. These systems are complicated and you don't know all the exact equipment and software you would need to purchase. Just explore and try your best to figure out the combination of software and hardware you would need to buy to reproduce your app on-premise.

To reproduce my architecture in a private cloud, my organization would need to invest in hardware, software and operational cost. Here's a rough estimate of the components the organization would need and their associated costs:

Hardware and Infrastructure:

Servers: My organization would need servers to host the applications and data. The cost can vary widely depending on the specifications [31].

Storage: Considering the amount of data my application would store, my organization would need a mid-range storage system [31].

Networking Equipment: This includes switches, routers, and firewalls. The cost can range from a few hundred to several thousand dollars depending on the complexity of your network.

Data Center Space: Since my hardware would be on-premises, I would also need to consider the cost of the physical space, power, and cooling systems.

Software:

Server Software and Licensing: This includes the operating system for my servers and any other software that my applications might need. The cost can vary widely depending on the software, but we can expect to spend a few hundred to a few thousand dollars per server [32].

Virtualization Software: To create a cloud-like environment, I would need virtualization software. There are free open-source options like OpenStack, but commercial options can cost a few thousand dollars [33].

Management Software: This software gives administrators centralized control over the infrastructure and applications running on it [33].

Operational Costs:

System Administration and Hardware Monitoring: These are the costs of the IT staff needed to manage the private cloud [34].

IT Staff Training and Turnover: There might be additional costs for training your IT staff to manage the private cloud, as well as costs associated with staff turnover [32].

Now let's calculate how much each resource would cost if we want to reproduce this architecture on a private cloud:

Private Cloud Setup:

Servers: The cost of servers for a private cloud can range from $100 to $200 per month for a small business dedicated server [35]. For larger businesses, the cost can go up to $300 per month [36]. So, for two servers (one for lambda functions and other for EC2), the cost could range from $200 to $600 per month or $2,400 to $7,200 per year [35][36].

Storage Systems: The cost of cloud storage can start at $0.02 per GB per month [37]. This cost would depend on the amount of storage needed, but let's assume $2,500 per year for a mid-range storage.

Networking Equipment: The cost of networking equipment for a private cloud can vary widely and is often based on bandwidth usage and data transferred in or out of the cloud infrastructure [38]. Let's assume $1,000 per year for meeting our architecture needs.

Data Center Space: The cost to build a small data center can range from $200,000 to $500,000 [39]. This includes costs for constructing the physical facility, purchasing and installing the necessary hardware, and setting up the cooling and power systems. This would be a one-time cost though.

Server Software and Licensing: The cost of server software and licensing for a private cloud can vary. For example, Canonical's Charmed Openstack offers a license at $0, with design and delivery costs ranging from $75,000 to $150,000 per engagement [40].

Virtualization Software: The cost of virtualization software for a private cloud can also vary. Some providers offer free services along with their paid offerings [41]. Let's assume $1,000 per year for meeting our architecture needs.

Management Software: This cost can vary, but let's assume a few thousand dollars.

System Administration and Hardware Monitoring: This cost can vary, but let's assume a few thousand dollars for this as well.

IT Staff Training and Turnover: On average, companies spent $1,111 per employee on training employees in 2020 [32]. $1,111 per employee per year. Assuming 5 employees, that's $5,555 per year.

So, the total rough cost for setting up a private cloud could be in the range of $300,000 to $650,000 for the first year, including the one-time costs. The recurring costs from the second year onwards could be in the range of $14,500 to $19,000 per year.

AWS Cloud Setup:

AWS Lambda: $0.20 per 1 million requests [42].

AWS EC2: The cost varies widely depending on the instance type, region, and pricing model [43].

AWS S3: S3 Standard Storage costs $0.023 per GB for the first 50 TB [44].

AWS API Gateway: The cost depends on the number of API calls you receive and the amount of data transferred [45].

AWS SNS: The cost depends on the number of messages you publish and deliver [46].

Amazon Textract: The cost is based on the number of pages processed [47].

The total cost of the AWS setup would depend on the usage, but it's likely to be significantly less than the cost of setting up a private cloud, especially when you consider the operational costs and the cost of scaling the infrastructure as the needs grow.

Monitoring:

Which cloud mechanism would be most important for you to add monitoring to in order to make sure costs do not escalate out of budget unexpectedly? In other words, which of your cloud mechanisms has the most potential to cost the most money?

Based on the services I've used in my project and the information I found, here are some insights:

AWS Lambda: AWS Lambda charges are based on the number of requests for your functions and the duration it takes for your code to execute [42]. The price depends on the amount of memory you allocate to your function [48]. If your functions are triggered frequently or run for a long duration, the costs can add up.

AWS EC2: EC2 instances are billed per second, with a minimum of 60 seconds [43]. Depending on the instance type, region, and whether you're using On-Demand Instances, Reserved Instances, or Spot Instances, the costs can vary [49].

AWS S3: With S3, you pay for storing objects in your S3 buckets. The rate you're charged depends on your objects' size, how long you stored the objects during the month, and the storage class [48].

AWS API Gateway: With Amazon API Gateway, you only pay when your APIs are in use. For HTTP APIs and REST APIs, you pay only for the API calls you receive and the amount of data transferred out [45].

AWS SNS (Simple Notification Service): With SNS, you pay based on the number of requests made, the number of notifications that you deliver, and any additional API calls for managing topics and subscriptions [46].

Amazon Textract: Textract charges are based on the number of pages processed whether you extract text, text with tables, form data, queries or process invoices and identity documents [47].

Among these services, **AWS Lambda, AWS EC2, and Amazon Textract** could potentially cost the most depending on the usage. Therefore, it would be important to add monitoring to these services to ensure costs do not escalate out of budget unexpectedly.

Further Development:

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?

If the development of the ExtractGen application were to continue, there are several potential enhancements and features that could be added to increase its functionality and efficiency:

1. **Machine Learning for Data Classification**: Implementing a machine learning model using AWS SageMaker could help in classifying and sorting the extracted data. This would make the data more meaningful and easier to analyze.
2. **Optical Character Recognition (OCR) for Handwritten Text**: AWS Textract does a great job with printed text, but struggles with handwritten text. Integrating an OCR solution for handwritten text could be a valuable addition.
3. **Multi-Language Support**: Currently, the application might be limited to processing English language PDFs. Adding multi-language support using Amazon Translate could make the application more versatile and useful to a wider audience.
4. **Real-Time Processing**: Instead of waiting for the entire PDF to be processed, implementing real-time processing would allow users to start receiving data as soon as it's extracted.
5. **User Interface Enhancements**: Improvements could be made to the user interface to make it more intuitive and user-friendly. This could be achieved using AWS Amplify which provides a set of tools and services for building secure, scalable mobile and web applications.
6. **Data Visualization**: Once the data is structured and stored, AWS QuickSight could be used to create visualizations of the data, making it easier for users to understand and interpret the data.
7. **Automated Data Pipelines**: AWS Glue could be used to create automated data pipelines for ETL (Extract, Transform, Load) jobs. This would automate the process of data preparation for analysis.
8. **Integration with AWS Comprehend**: AWS Comprehend is a natural language processing (NLP) service that uses machine learning to find insights and relationships in text. This could be used to analyze the extracted text and provide additional insights.
9. **Data Encryption**: To enhance the security of the stored data, AWS Key Management Service (KMS) could be used to encrypt the data at rest in S3 and the data in transit.
10. **Cost Optimization**: Implementing cost optimization strategies can help to manage the costs associated with running the application. AWS Cost Explorer could be used to visualize, understand, and manage AWS costs and usage over time.

References:

[1] “AWS Lambda vs EC2: Compared on performance, cost, security, and more,” *Lumigo* [Online]. Available: https://lumigo.io/blog/aws-lambda-vs-ec2 [Accessed: Mar. 9, 2024].

[2] “AWS Lambda vs EC2: Comparison of AWS compute resouces,” *Simform* [Online]. Available: https://www.simform.com/blog/aws-lambda-vs-ec2/ [Accessed: Mar. 9, 2024].

[3] Ali Chahine, “Choosing Between AWS Lambda and EC2: A Practical Guide,” *Medium* [Online]. Available: https://medium.com/@skewnest21/choosing-between-aws-lambda-and-ec2-a-practical-guide-828ffb0c05be [Accessed: Mar. 10, 2024].

[4] Stephen J, “An overview of Amazon EC2 vs. AWS Lambda,” *Tech Target* [Online]. Available: https://www.techtarget.com/searchcloudcomputing/tip/An-overview-of-Amazon-EC2-vs-AWS-Lambda [Accessed: Mar. 10, 2024].

[5] “Deploying Elastic Beanstalk applications from Docker containers,” *AWS* [Online]. Available: https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create_deploy_docker.html [Accessed: Mar. 10, 2024].

[6] David McCracken, “AWS S3 vs Azure Blob Storage: Choosing the Right Cloud Storage Solution,” *Medium* [Online]. Available: https://towardsdev.com/aws-s3-vs-azure-blob-storage-choosing-the-right-cloud-storage-solution-e7049461d651 [Accessed: Mar. 10, 2024].

[7] “AWS SNS vs SQS,” *Ably* [Online]. Available: https://ably.com/topic/aws-sns-vs-sqs [Accessed: Mar. 15, 2024].

[8] “AWS Rekognition vs Textract,” *Myrestraining* [Online]. Available: https://myrestraining.com/blog/aws/aws-rekognition-vs-textract/ [Accessed: Mar. 15, 2024].

[9] Abish Pius, “AWS Texttract vs AWS Comprehend,” *Medium* [Online]. Available: https://medium.com/chat-gpt-now-writes-all-my-articles/aws-texttract-vs-aws-comprehend-668504162680 [Accessed: Mar. 15, 2024].

[10] “Cloud Deployment Models: Advantages and Disadvantages,” *QuickStart* [Online]. Available: https://www.quickstart.com/cloud-computing/cloud-deployment-models:-advantages-and-disadvantages/ [Accessed: Mar. 17, 2024].

[11] “Cloud Deployment Models,” *GeeksforGeeks* [Online]. Available: https://www.geeksforgeeks.org/cloud-deployment-models/ [Accessed: Mar. 17, 2024].

[12] “Public vs Private Cloud: Cloud Deployment Models,” *Enterprise Networking Planet* [Online]. Available: https://www.enterprisenetworkingplanet.com/management/public-vs-private-cloud-cloud-deployment-models/ [Accessed: Mar. 17, 2024].

[13] “Cloud Deployment Models,” *Simform* [Online]. Available: https://www.simform.com/blog/cloud-deployment-models/ [Accessed: Mar. 17, 2024].

[14] “SaaS,” *IBM* [Online]. Available: https://www.ibm.com/topics/saas [Accessed: Mar. 17, 2024].

[15] “SaaS Delivery,” *Apptension* [Online]. Available: https://www.apptension.com/blog-posts/saas-delivery [Accessed: Mar. 17, 2024].

[16] Eline Hagene, “7 Good Reasons Why Choose SaaS,” *Frontcore* [Online]. Available: https://frontcore.com/blog/7-good-reasons-why-choose-saas/ [Accessed: Mar. 17, 2024].

[17] “EC2 Instance,” *Hands-On Cloud* [Online]. Available: https://hands-on.cloud/aws-services/ec2/instance/ [Accessed: Mar. 26, 2024].

[18] “AWS EC2 Security,” *Geekflare* [Online]. Available: https://geekflare.com/aws-ec2-security/ [Accessed: Mar. 26, 2024].

[19] “Security Best Practices,” *Amazon S3* [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/security-best-practices.html [Accessed: Mar. 26, 2024].

[20] “Secure S3 Resources,” *Repost AWS* [Online]. Available: https://repost.aws/knowledge-center/secure-s3-resources [Accessed: Mar. 26, 2024].

[21] “Amazon S3 Security Best Practices,” *Wiz.io* [Online]. Available: https://www.wiz.io/academy/amazon-s3-security-best-practices [Accessed: Mar. 26, 2024].

[22] “Security,” *AWS Textract* [Online]. Available: https://docs.aws.amazon.com/textract/latest/dg/security.html [Accessed: Mar. 27, 2024].

[23] “SNS Security,” *AWS* [Online]. Available: https://docs.aws.amazon.com/sns/latest/dg/sns-security.html [Accessed: Mar. 27, 2024].

[24] “Web Application Firewall (WAF),” *Cloudflare* [Online]. Available: https://www.cloudflare.com/learning/ddos/glossary/web-application-firewall-waf/ [Accessed: Mar. 27, 2024].

[25] “What is API Throttling,” *Tibco* [Online]. Available: https://www.tibco.com/glossary/what-is-api-throttling [Accessed: Mar. 27, 2024].

[26] “API Gateway Method Request Validation,” *AWS* [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-method-request-validation.html [Accessed: Mar. 30, 2024].

[27] Bhavesh Atara, “Decoding AWS ACLs vs Security Groups: A Comprehensive Guide,” *Medium* [Online]. Available: https://medium.com/@bhavesh.atara/decoding-aws-acls-vs-security-groups-a-comprehensive-guide-%EF%B8%8F-3200305e7d57 [Accessed: Mar. 30, 2024].

[28] “Encryption Best Practices,” *AWS* [Online]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/encryption-best-practices/welcome.html [Accessed: Mar. 30, 2024].

[29] “12 AWS IAM Security Best Practices,” *Spacelift* [Online]. Available: https://spacelift.io/blog/aws-iam-best-practices [Accessed: Mar. 30, 2024].

[30] “Securing Amazon S3 Presigned URLs,” *Insecurity* [Online]. Available: https://insecurity.blog/2021/03/06/securing-amazon-s3-presigned-urls/ [Accessed: Mar. 30, 2024].

[31] “Private Cloud,” *TechRepublic* [Online]. Available: https://www.techrepublic.com/article/private-cloud/ [Accessed: Mar. 30, 2024].

[32] “Cloud Cost Models & Management Strategies,” *Spot.io* [Online]. Available: https://spot.io/resources/cloud-cost/cloud-cost-models-management-strategies/ [Accessed: Mar. 30, 2024].

[33] “Private Cloud,” *IBM* [Online]. Available: https://www.ibm.com/topics/private-cloud [Accessed: Mar. 30, 2024].

[34] “Public vs Private Cloud Cost Comparison,” *CloudPap* [Online]. Available: https://cloudpap.com/blog/public-vs-private-cloud-cost-comparison/ [Accessed: Mar. 30, 2024].

[35] “How Much Does a Server Cost for a Small Business,” *ServerMania* [Online]. Available: https://www.servermania.com/kb/articles/how-much-does-a-server-cost-for-a-small-business [Accessed: Mar. 30, 2024].

[36] “How Much Does It Cost to Rent a Server,” *ServerMania* [Online]. Available: https://www.servermania.com/kb/articles/how-much-does-it-cost-to-rent-a-server [Accessed: Mar. 30, 2024].

[37] “Cloud Storage Pricing,” *Enterprise Storage Forum* [Online]. Available: https://www.enterprisestorageforum.com/cloud/cloud-storage-pricing/ [Accessed: Mar. 31, 2024].

[38] “Network Pricing,” *Google Cloud* [Online]. Available: https://cloud.google.com/vpc/network-pricing [Accessed: Mar. 31, 2024].

[39] “Understanding Data Center Costs and How They Compare to the Cloud,” *Granulate* [Online]. Available: https://granulate.io/blog/understanding-data-center-costs-and-how-they-compare-to-the-cloud/ [Accessed: Mar. 31, 2024].

[40] “Private Cloud Pricing,” *Clinked* [Online]. Available: https://blog.clinked.com/private-cloud-pricing [Accessed: Mar. 31, 2024].

[41] “Virtual Private Cloud,” *TrustRadius* [Online]. Available: https://www.trustradius.com/virtual-private-cloud [Accessed: Mar. 31, 2024].

[42] “Understanding AWS Lambda Pricing,” *Guille Ojeda* [Online]. Available: https://blog.guilleojeda.com/understanding-aws-lambda-pricing [Accessed: Mar. 31, 2024].

[43] “On-Demand Pricing,” *AWS EC2* [Online]. Available: https://aws.amazon.com/ec2/pricing/on-demand/ [Accessed: Apr. 1, 2024].

[44] “Pricing,” *Amazon S3* [Online]. Available: https://aws.amazon.com/s3/pricing/ [Accessed: Apr. 1, 2024].

[45] “Pricing,” *AWS API Gateway* [Online]. Available: https://aws.amazon.com/api-gateway/pricing/ [Accessed: Apr. 1, 2024].

[46] “Pricing,” *AWS SNS* [Online]. Available: https://aws.amazon.com/sns/pricing/ [Accessed: Apr. 1, 2024].

[47] “Pricing,” *AWS Textract* [Online]. Available: https://aws.amazon.com/textract/pricing/ [Accessed: Apr. 1, 2024].

[48] “Pricing,” *AWS Lambda* [Online]. Available: https://aws.amazon.com/lambda/pricing/ [Accessed: Apr. 1, 2024].

[49] “Guide to AWS EC2 Costs,” *Apptio* [Online]. Available: https://www.apptio.com/blog/guide-to-aws-ec2-costs/ [Accessed: Apr. 1, 2024].