CSCI5409 - Cloud Computing
Term Assignment

Shivam Lakhanpal
B00932887

Gitlab:
https://git.cs.dal.ca/courses/2023-summer/csci4145-5409/slakhanpal/-/tree/main/Term-Assignment

Section 1 : What you built and what it is supposed to do?

Chrono Notes: A Collaborative Note-Taking Application

Chrono Notes is a user-friendly app designed to make note-taking and sharing easier and more collaborative. This cutting-edge tool acts as a central station for anyone who needs to keep track of ideas, tasks, or notes, and wants to share them in real-time with others.

The main goal of Chrono Notes is to make information sharing as simple and effective as possible. Its users are a diverse group, which includes project managers, researchers, students, and anyone else who needs a streamlined way to manage their notes. The most important measure of success for Chrono Notes is its ability to immediately update and share information, keeping users connected and informed.

Functionalities and Features

1. **Adding Notes:** Users can create notes with headings and descriptions. Upon submission, these notes are displayed as easily **navigable cards**. A click on any card reveals the option to Edit, Delete, or Close the note, allowing users to manage their notes effectively.

2. **Uploading Handwritten Notes**: Chrono Notes breaks the mold with a standout feature that enables users to upload images of handwritten notes. Utilizing Amazon Textract, the application can extract text from these images and generate digital notes, thereby bridging the gap between traditional and digital note-taking.

3. **Note Subscription:** Users can subscribe to specific notes by providing their email. Leveraging the capabilities of Amazon Simple Notification Service (AWS SNS), a confirmation email is dispatched to the user for subscribing to a note. There is no restriction on the number of subscribers a note can have.

4. **Real-time Note Updates**: Chrono Notes fosters a truly connected environment by sharing real-time note updates with all subscribers whenever a note is modified. This feature keeps everyone on the same page, alerting them about any changes, and thereby promoting effective and informed teamwork.

In essence, Chrono Notes is not just a traditional note-taking app. It's a dynamic, collaborative tool designed to streamline the sharing of information, catalyzing effective teamwork, and enhancing productivity. By offering features like **real-time updates** and the ability to **digitize handwritten notes**, Chrono Notes positions itself as a transformative solution in the realm of collaborative note-taking.

User Demographics

Chrono Notes provides a versatile platform that caters to a broad user base including individual professionals, collaborative teams, and those in the educational field.

For professionals, Chrono Notes offers a unified space to manage and organize their thoughts, tasks, and project milestones. It supports both typed and handwritten notes, appealing to diverse work styles and preferences. Collaborative teams, particularly in corporate settings, benefit from the subscription feature which facilitates team communication and coordination on shared projects. In the educational context, students and researchers can digitize their lecture notes or research findings, effectively merging traditional and modern note-taking practices.

Performance Objectives

The key performance targets for Chrono Notes are centered around its unique collaborative attributes and real-time update capabilities. The real-time updates need to be instantaneous and reliable to maintain the seamless collaborative experience. Furthermore, the application aims to ensure the efficient and precise transformation of handwritten notes into digital text. This blend of traditional note-taking with digital advancements is a significant performance metric. The objective is to minimize any potential error rate in this transformation process, thereby optimizing the overall user experience.

Application Workflow

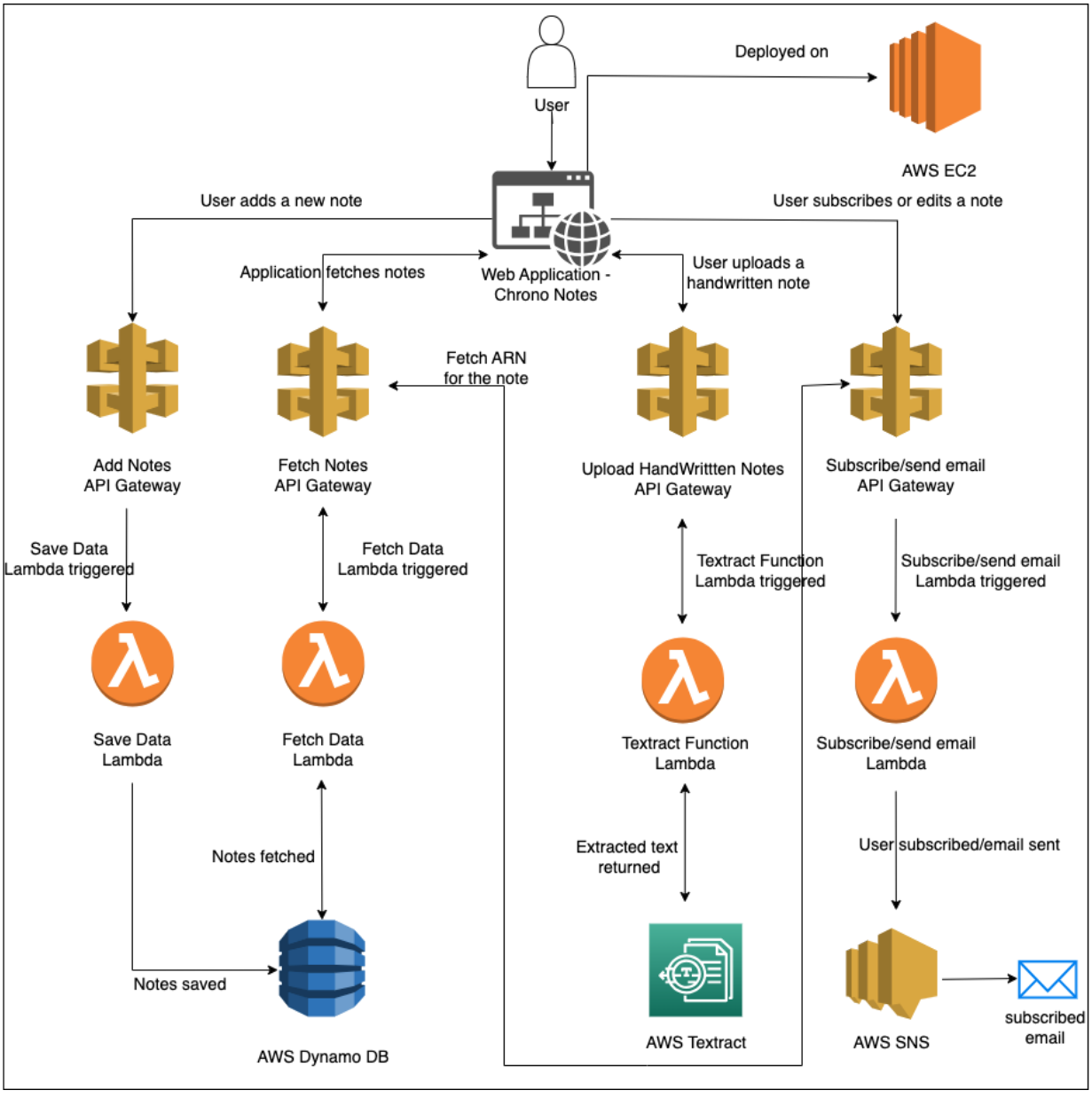

Fig 1: Application architecture - made with **draw.io [3]**

Section 2 : 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.

Table 1: AWS service I selected from each category

Category	Service
Compute	AWS EC2, AWS Lambda
Storage	AWS DynamoDB
Network	AWS API Gateway
General	AWS Textract , AWS SNS
Resource Allocator	AWS CloudFormation
Frontend (Client)	React(with Chakra UI), HTML, JavaScript

My explanation about the services selected

Compute

AWS Lambda: I selected this service because it enables running functions without servers. It's efficient, scalable, and cost-effective as you only pay for the compute time consumed.

AWS EC2: I chose EC2 to host the web application. It offers flexibility and a variety of instance types to fit different use cases. The ease of scaling resources up and down in response to changes in demand makes EC2 an ideal choice.

Storage

AWS DynamoDB: For storage, I opted for DynamoDB, a NoSQL database service offering fast and predictable performance with seamless scalability. This choice was influenced by the need for a highly responsive database to store and retrieve notes effectively.

Network

AWS API Gateway: I selected API Gateway to secure and route API requests to Lambdas. It helps manage the app's APIs and leverage AWS's robust infrastructure to handle traffic management, authorization, and access control, and monitoring.

General

Amazon Textract: I used Textract to automatically extract text, handwriting, and data from scanned documents. This enables the app's unique feature of transcribing handwritten notes.

AWS SNS (Simple Notification Service): SNS was chosen for its ability to send notifications across a wide range of subscribers. It enables the key feature of our app, which sends updates to subscribed users when a note is edited.

Here are their alternatives(if exists) and comparison of Alternative Services

Compute

Alternatives: AWS Elastic Beanstalk, Docker & AWS Elastic Beanstalk, AWS Elastic Container Service & Elastic Container Registry, Amazon Elastic Kubernetes Service (EKS), AWS Step Functions, AWS IoT 1-Click.

My reason for Selection:

AWS Lambda offers simplicity and cost-efficiency. Compared to services like Elastic Beanstalk or EKS, it reduces the need for system administration and auto-scales according to the workload. With EC2, the flexibility to choose among various instance types and the ability to maintain full control of the operating environment made it preferable over the alternatives.

Storage

Alternatives: AWS S3, AWS Aurora, AWS AppSync, AWS Athena, AWS IoT Analytics, AWS Neptune, AWS Relational Database Service.

My reason for Selection:

DynamoDB provides a managed NoSQL database service, which is ideal for this application's needs. Compared to RDS or Aurora, DynamoDB provides flexible data structuring and requires less maintenance and administrative tasks. It also provides automatic scalability, unlike S3, which made it a more suitable choice.

Network

Alternatives: AWS Virtual Private Cloud (VPC), Amazon CloudFront, Amazon EventBridge.

My reason for Selection: API Gateway provides built-in security, throttling, and system integration features. It also simplifies the process of building and deploying APIs, unlike VPC, where manual network setup is needed. Compared to CloudFront, API Gateway offers more functionalities necessary for the app, like request routing to Lambdas.

General

Alternatives: AWS Backup, AWS Batch, Amazon Comprehend, AWS Data Pipeline, AWS DeepComposer, AWS DeepLens, AWS DeepRacer, and many more.

My reason for Selection: Textract's ability to extract text from scanned images is unparalleled in the alternatives and serves a crucial feature in the app. AWS SNS was chosen over alternatives such as SQS due to its capacity to deliver notifications to a large number of recipients simultaneously, a functionality crucial for our note subscription feature.

Section 3 : Describe your deployment model. Why did you choose this deployment model?

Chosen Deployment Model: Public Cloud

Chrono Notes employs a Public Cloud deployment model. This choice was made considering various attributes like dynamic scalability, economical viability, management simplicity, and swift deployment.

Scalability:

One of the prominent reasons for choosing a public cloud deployment model for Chrono Notes is its superior scalability. This platform caters to a diverse range of users with varying loads of note-making, sharing, and collaboration. The public cloud enables us to automatically scale resources up or down based on the application demand, ensuring a smooth user experience even during peak usage times. For instance, if there's a significant increase in note uploads or subscriptions during exam periods for student users, the public cloud can effortlessly handle this surge.

Cost-Effectiveness:

A public cloud model follows a pay-as-you-go pricing approach, meaning you only pay for the resources you use. Given the variable usage patterns of Chrono Notes (for example, students may use it more intensively during the academic year, and less so during holidays), this flexibility allows for significant cost savings as we aren't tied to maintaining large, expensive data centers ourselves.

Ease of Management:

The public cloud providers take care of the infrastructure management, which includes tasks like server upkeep, resource allocation, and security updates. This allows the Chrono Notes team to focus more on developing features like real-time updates and handwritten note digitization, and less on routine system maintenance.

Quick Deployment:

Using the public cloud model, we can rapidly deploy updates and new features to Chrono Notes. For example, if we develop an enhancement for the handwritten note upload feature or add a new capability like voice note transcription, these can be quickly and easily pushed to the cloud without worrying about lengthy hardware or system update processes. This ensures our users always have access to the latest and

greatest features of Chrono Notes.

In a nutshell,
Chrono Notes benefits immensely from the public cloud's worldwide accessibility. The inherent scalability of public cloud allows it to seamlessly handle the application's potential data load, which is crucial given the data-intensive features like note creation, sharing, and digitizing handwritten notes.

Section 4 : Describe your delivery model. Why did you choose this delivery model?

The delivery model chosen for Chrono Notes is a combination of **Infrastructure as a Service (IaaS)** and **Function as a Service (FaaS)**. This hybrid approach was selected due to the distinct advantages offered by both models and the nature of different components within the Chrono Notes application.

The IaaS model was chosen because of its flexibility, control, and scalability. Amazon EC2, an IaaS component, is utilized to host the frontend of the Chrono Notes application. The use of EC2 offers several benefits, such as the ability to manage the underlying infrastructure, to select from a wide variety of available hardware configurations based on the needs of the application, and to rapidly scale resources in response to demand. It also provides Chrono Notes with a high degree of control over the hosting environment.

On the other hand, the FaaS model was chosen for the backend logic of Chrono Notes, which is handled using AWS Lambda. The choice of FaaS was driven by its ability to abstract away the server layer completely, thereby allowing the team to focus purely on functionality. The FaaS model is cost-effective because it only charges for the actual execution time of the functions, making it a perfect fit for sporadic or unpredictable workloads. It also scales automatically and effortlessly in response to incoming event triggers or traffic.

By combining IaaS and FaaS, Chrono Notes can leverage the strengths of both delivery models, providing an efficient and scalable solution while allowing the team to concentrate on delivering core functionality and value to the users.

Section 5 : Final Architecture and System Description of Chrono Notes

Our final architecture for Chrono Notes is a **Rapid Provisioning Architecture**. This means we've used a combination of ready-made services available on the cloud to rapidly set up, test, and deploy our application, thus reducing the time taken from development to production.

Cloud Mechanisms and Data Storage

Chrono Notes employs various cloud mechanisms that work in harmony to deliver a seamless note-taking and sharing experience. The system is built on Amazon Web Services (AWS), one of the most reliable and versatile cloud platforms.

In the Chrono Notes system architecture, several AWS services work in synergy to provide a robust and scalable note-taking application. We leverage Amazon DynamoDB, a NoSQL database service, to securely store all note data. The backend code for the application, which powers various functionalities, is executed through AWS Lambda, a serverless computing service.

To facilitate communication between the frontend and backend, we employ Amazon API Gateway. It provides secure and efficient API management. The frontend of Chrono Notes, constructed with ReactJS, is hosted on an Amazon EC2 instance offering flexible and scalable computing capacity. Furthermore, we use AWS CloudFormation for efficient resource allocation, Amazon SNS for sending notification emails to users, and Amazon Textract to process handwritten notes into digital text.

Programming Languages and Code Usage

The backend code for Chrono Notes is written in **Node.js**, a JavaScript runtime environment known for its performance, **scalability, and compatibility with AWS Lambda**. This makes it a suitable choice for the backend operations of a web-based application.

ReactJS, a popular JavaScript library, is **used for the frontend** due to its efficiency, flexibility, and robustness, which are essential for creating interactive UIs.

Various parts of the application required code, including the frontend UI, API creation and management, Lambda functions to interact with DynamoDB, text extraction from handwritten notes, and email notification service.

Deployment to the Cloud

The front-end segment of Chrono Notes is hosted on a cloud server using an Amazon EC2 instance. This provides our application with the necessary computing resources for smooth operation. Additionally, the application's APIs have been developed as Lambda functions, which provide serverless compute services, further enhancing the scalability and efficiency of Chrono Notes.

Architecture Analysis

The architecture of Chrono Notes aligns with the serverless architecture taught in the course, utilizing services like AWS Lambda, DynamoDB, and API Gateway. This architecture was chosen due to its ability to auto-scale based on demand, reduce operational overhead, and offer a pay-per-use model, making it cost-effective.

However, it's worth mentioning that serverless architectures can potentially introduce complexities in monitoring and debugging due to the distributed nature of the services. Despite this, for an application like Chrono Notes, which requires high scalability and availability, a serverless architecture is a wise choice.

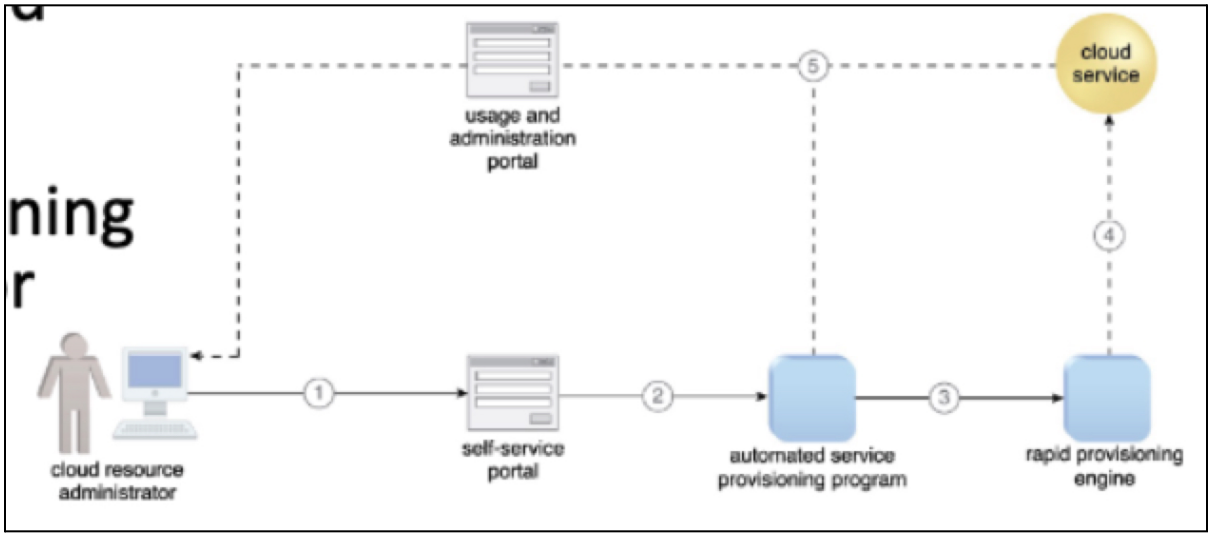

Fig 2. Rapid Provisioning Architecture [4]

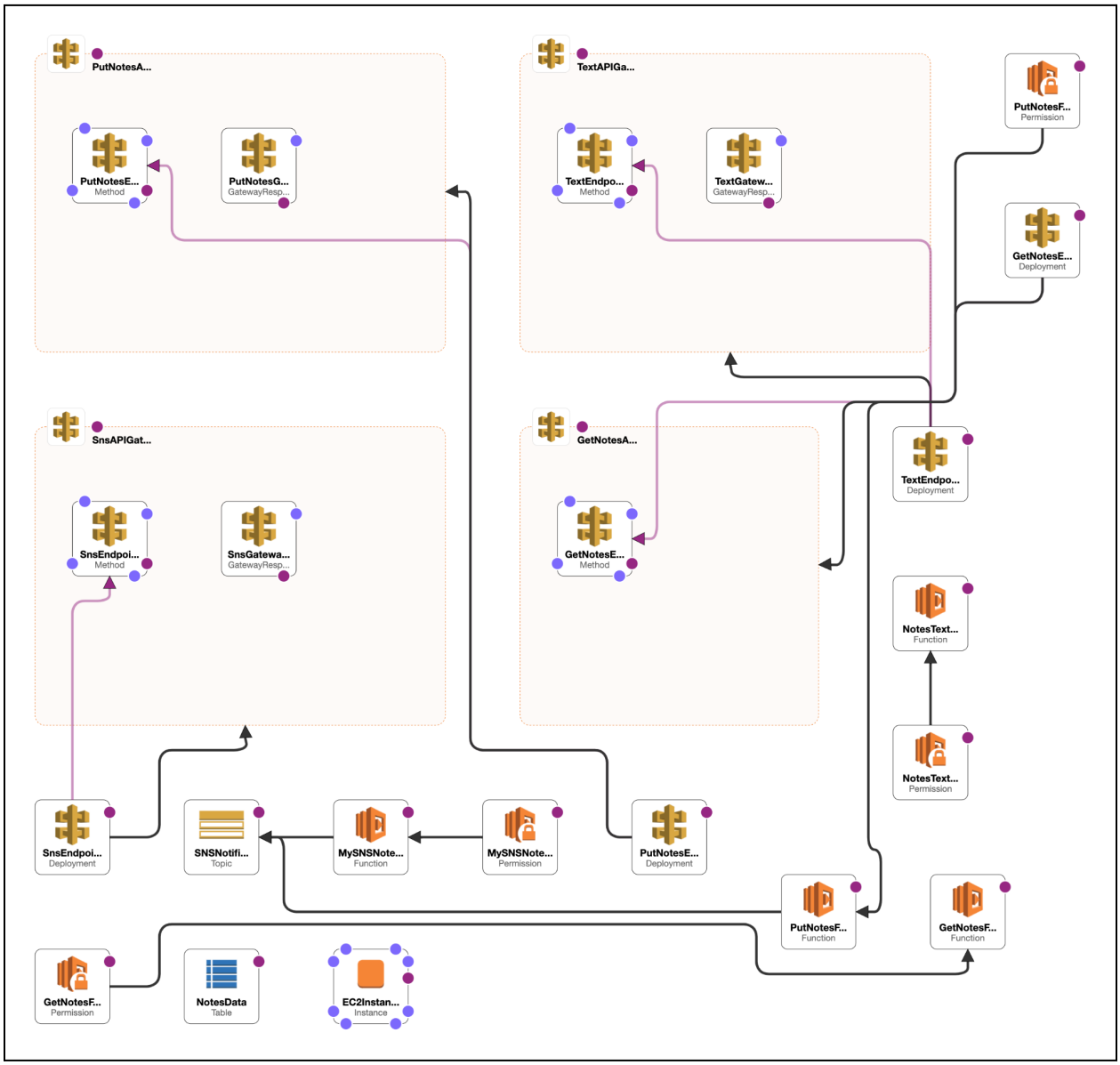

Fig 3. CloudFormation Resource Architecture in Designer [8]

Section 6 : An analysis of your project's approach to security, particularly its approach to securing data through all stages of the system (in transit, at rest).

1. **Encrypting Data in Transit:** Chrono Notes prioritizes data safety during its transfer by utilizing HTTPS for all the API interactions. This protocol guarantees data is encrypted as it moves, providing a secure shield against potential interceptions.

2. **Adopting IAM for Access Control:** We utilize AWS's Identity and Access Management (IAM) for governing access to our resources. We assign IAM roles to our AWS Lambda functions and fine-tune permissions based on the principle of least privilege. This approach restricts each function's access to only the necessary resources, thus reducing any potential security risks.

3. **Safeguarding Critical Information:** In Chrono Notes, we ensure key data like AWS Access Keys and Secrets are never stored in the code. Instead, these pieces of information are securely passed as environment variables to our AWS Lambda functions, significantly reducing the possibility of unintentional data exposure.

Data Security Vulnerabilities and Mitigation in Chrono Notes

While the architecture of Chrono Notes is designed with the best security practices in mind, potential vulnerabilities may exist, particularly concerning data transmission and storage. For instance, an unencrypted data transmission could expose sensitive user information. Similarly, lack of proper access controls to the DynamoDB database could potentially allow unauthorized access to data.

To mitigate these vulnerabilities, we incorporate several security mechanisms in our architecture. We use Amazon's Secure Sockets Layer (SSL) for data-in-transit encryption, ensuring that the data exchanged between the server and client is not susceptible to interceptions. Additionally, AWS Identity and Access Management (IAM) is used to control access to AWS services and resources, including DynamoDB. IAM allows us to create and manage AWS users and groups, and grant or deny necessary permissions to our resources, ensuring data is accessed only by authenticated and authorized users.

Furthermore, we can enhance data security by integrating services like AWS Shield for DDoS mitigation and AWS WAF (Web Application Firewall) for protection against common web exploits. While our current security measures provide robust protection, the addition of these services could further strengthen our defense against cyber threats.

Section 7 : 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.

Table 2: AWS service I selected from each category

AWS Service	Private Cloud Equivalent	Explanation	Estimated Initial Cost
EC2	VMware vSphere	VMware vSphere provides a virtualization platform for creating and managing virtual machines on-premise.[1]	$5,000 (license cost for 3 hosts) [1]
API Gateway	Kong or Apigee	Kong and Apigee are popular solutions for managing APIs that can be hosted on your own infrastructure. They would require more manual setup and maintenance compared to AWS API Gateway. [5]	Free (Open Source, but hardware and setup costs apply for Kong), $1,500/month (for Apigee Standard tier)

DynamoDB	Managed NoSQL Database	Apache Cassandra or MongoDB [8]	Free (Open Source, but hardware and setup costs apply) [8]
Lambda	OpenFaaS (Functions as a Service)	OpenFaaS allows running serverless functions on your own hardware, but it requires more manual setup and management compared to Lambda.[2]	Free (Open Source) [2]
Textract	Kofax Omnipage	Kofax Omnipage is an enterprise-grade OCR solution offering similar services to Textract. It provides powerful text and data extraction capabilities but would require manual setup and maintenance.[7]	Licensing starts at around $2,500 for a basic setup. [7]
SNS	RabbitMQ or ActiveMQ	RabbitMQ or ActiveMQ are popular open-source messaging systems that can be self-hosted for a publish-subscribe model. However, they require more manual setup and maintenance compared to AWS SNS.[6]	Free for RabbitMQ and ActiveMQ (Open Source, but hardware and setup costs apply) [6]

Note:

Please note that these figures are rough estimates and the actual costs could be considerably higher, especially when factoring in the total cost of ownership over several years. The complexity and cost of managing an on-premise private cloud is why many businesses opt for public cloud solutions, despite the potential for higher long-term costs.

Section 8 : 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?

A critical component of managing costs for Chrono Notes is monitoring our **Amazon EC2 instances**. These are the backbone of our application and can quickly lead to rising expenses if not managed properly. Amazon CloudWatch is the key tool here, allowing us to keep an eye on resource utilization and alert us when costs approach our budget limit. For example, if our monthly budget for EC2 is $100, we can set an alert for when we reach $80, allowing us to take action before going over budget.

Additionally, Amazon EC2 Auto Scaling can dynamically adjust the number of active EC2 instances based on demand, reducing costs during off-peak hours by decreasing the number of instances. This feature is particularly useful for Chrono Notes during off-hours when user activity might be low.

Section 9 : 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 I were to continue the development of the Chrono Notes application, several enhancements and new features could significantly improve the user experience and the functionality of the application.

1. **Timestamps for Notes**: Adding a timestamp for each note created, edited, or deleted will allow users to track the history of changes to their notes. This feature can be easily implemented using DynamoDB's time-to-live (TTL) attribute, which provides an automatic timestamp.

2. **Password Protection for Notes:** To enhance security, users might want to password-protect some of their notes. For this feature, AWS Cognito can be used to manage user authentication and provide an extra layer of security. It would enable users to set a password for certain notes, which would be required to access those notes.

3. **User Profiles:** A user profile section would be a great addition, allowing users to manage their account settings and personal information. It would also enable users to manage their notification preferences for each note they have subscribed to. For this, we can use Amazon Cognito User Pools to manage and authenticate users, and AWS AppSync to manage real-time updates to user profile data.

4. **Collaborative Editing of Notes:** A feature that allows multiple users to edit a note simultaneously could improve the application's collaboration capabilities. Amazon AppFlow could be used for this, as it supports real-time data syncing across different AWS services.

5. **Integration with Calendars:** Users might find it useful to integrate their notes with a calendar feature. This feature could remind users of important notes or tasks. AWS Step Functions could help orchestrate the business logic of integrating with popular calendar services.

References

[1] “VMware vSphere,” VMware, 12-Jul-2023. [Online]. Available: https://www.vmware.com/ca/products/vsphere.html. [Accessed: 01-Aug-2023].

[2] OpenFaaS Ltd, “Home,” OpenFaaS - Serverless Functions Made Simple. [Online]. Available: https://www.openfaas.com/. [Accessed: 01-Aug-2023].

[3] “draw.io,” Drawio.com. [Online]. Available: https://www.drawio.com/. [Accessed: 01-Aug-2023].

[4] “Cloud Computing Architectures,” Brightspace.com. [Online]. Available: https://dal.brightspace.com/d2l/le/content/274266/viewContent/3647785/View. [Accessed: 01-Aug-2023].

[5] V. Kasipuri, “Apigee vs Kong,” Mindmajix, 03-Apr-2023. [Online]. Available: https://mindmajix.com/apigee-vs-kong. [Accessed: 01-Aug-2023].

[6] “RabbitMQ: easy to use, flexible messaging and streaming — RabbitMQ,” Rabbitmq.com. [Online]. Available: https://www.rabbitmq.com/. [Accessed: 01-Aug-2023].

[7] Kofax.com. [Online]. Available: http://kofax.com/products/omnipage. [Accessed: 01-Aug-2023].

[8] “NoSQL database,” DataStax. [Online]. Available: https://www.datastax.com/lp/nosql-database?utm_medium=search_pd&utm_source=google&utm_campaign=ggl_s_nam_dev_nonbrand_nosql&utm_content=&_bt=586427080205&_bk=nosql&_bm=b&_bn=g&_bg=132583165605&gclid=Cj0KCQjw2qKmBhCfARIsAFy8buLwRl2NPyDatu6Zzx6c8rGIcdzQH2TqfU4FcvVwjiHKXbH4Z7K5txUaAoasEALw_wcB. [Accessed: 01-Aug-2023].