CSCI 5409

Cloud Computing

Cloud Project Report

Banner ID: B00957587

Name: Shreya Kapoor

Table of Contents

Part 1: Introduction of Project

I have developed a web application designed to facilitate the sharing of personalized postcards among users. The primary objective of the application is to provide users with a platform where they can create customized postcards and share them with their loved ones.

Key Features:

1. **Customized Postcard**: Users can create personalized postcards by adding their own handwritten messages or in a document. Additionally, users have the option to upload photos to include in their postcards.

2. **Handwritten Recognition**: The application incorporates handwriting recognition technology to convert handwritten messages on uploaded documents into text. This feature enhances the user experience by allowing handwritten messages to be easily incorporated into the postcards.

3. **Sharing Functionality**: Once a postcard is created, users can share it with their intended recipients by generating a shareable link or by clicking on view postcard. This link can be sent via email or shared through other messaging platforms.

Target Audience:

The target audience for my application includes individuals who wish to send personalized postcards to their friends, family, or loved ones. This may include individuals celebrating special occasions such as birthdays and anniversaries, as well as those seeking to express their sentiments through handwritten messages.

Performance Targets:

1. **User Experience**: The application should provide a seamless and intuitive user experience, allowing users to easily create and share postcards without encountering significant technical hurdles.

2. **Handwritten Recognition Accuracy**: The handwriting recognition feature should accurately convert handwritten messages into text, ensuring that the intended message is conveyed correctly in the postcard.

3. **Response Time**: The application should have low latency, ensuring that users can quickly create and share postcards without experiencing significant delays.

Part 2: Menu Item Requirements

2.2: List of Services Selected

AWS SERVICE CATEGORY	AWS SERVICE NAME
COMPUTE	Elastic Container Service & Elastic Container Registry
	Lambda
STORAGE	S3
	Relational Database Service
NETWORK	Virtual Private Cloud
GENERAL	SNS
	Textract
	Elastic Load Balancing

2.3: Comparison of Alternative Services

2.3.1 Compute

Elastic Container Service & Elastic Container Registry
ECS provides a fully managed container orchestration service, while ECR offers secure and scalable storage for container images.

Reasons to Choose ECS and ECR over other alternatives:

1. **Container Orchestration**: ECS makes it easier to deploy and handle containers by offering functions such as auto-scaling, load balancing, and built-in monitoring. This simplifies the management of container lifecycles, cutting down on operational tasks and speeding up the development and deployment of applications.

2. **Cost Friendly**: ECS charges based on how much we use, which means we only pay for what we need. This is great for apps with changing workloads because we don't have to pay for unused resources.

3. **Scalability and Elasticity**: ECS offers built-in auto-scaling capabilities, allowing containers to scale dynamically based on demand. This elasticity ensures optimal resource utilization and performance, enabling applications to handle fluctuations in traffic and workload without manual intervention.

4. **Integration with AWS Ecosystem**: ECS integrates seamlessly with other AWS services, such as CloudWatch, IAM, and VPC, enabling easy integration and management of containerized applications within the AWS environment.

Lambda

Lambda provides a serverless computing platform that enables developers to run code without provisioning or managing servers. In my Postcard application, I've implemented AWS Lambda to handle text extraction and database updates when new postcard images are uploaded.

Reasons to Choose Lambda over other alternatives:

1. **Real time triggers**: Lambda enables real-time processing of postcard images as soon as they are uploaded to the S3 bucket. This ensures timely extraction and updating of text data in the database, enhancing user experience.

2. **Customizability**: Lambda provides flexibility in implementing custom logic for text extraction and database operations. I have complete control over the execution environment and can tailor the function to suit specific requirements.

3. **High Availability**: Lambda automatically replicates my function across multiple availability zones, ensuring high availability and fault tolerance. In the event of a failure in one zone, Lambda continues to execute the function in another zone, maintaining service reliability.

4. **Cost Efficiency**: Lambda's serverless architecture allows me to execute code in response to events without the need to provision or manage servers. With Lambda, I only pay for the compute time consumed by my function, making it cost-effective for sporadic tasks like text extraction from postcard images.

2.3.2 Storage

S3

In my Postcard application, I utilize AWS S3 to store the images and documents with the text associated with each postcard uploaded by users. Storing these images and documents in an S3 bucket ensures efficient management and accessibility.

Reasons to Choose AWS S3 over Other Alternatives:

1. **Durability**: With an exceptional high durability which ensures the safety and availability of stored data. This level of durability is crucial for maintaining the visual representation of postcards, ensuring that users can reliably access images and documents associated with each postcard.

2. **Cost-Effectiveness**: S3 offers a cost-effective solution for storing a variety of unstructured data, including images, videos, and logs.

3. **Data Replication**: S3 automatically replicates data across different availability zones, enhancing data availability and disaster recovery capabilities. This ensures that postcard images and documents remain accessible even in the event of a system failure or disaster.

4. **Compliance and Security**: S3 offers robust security features to protect stored data, including encryption, access control, and compliance certifications. It complies with industry standards and regulations, ensuring data privacy and regulatory compliance.

Relational Database Service

In my application, Postcard, I rely on AWS RDS with Aurora (MySQL) as the primary database solution for storing postcard details particularly the image and document links and the extracted text.

Reasons to Choose AWS RDS Aurora (MySQL) over Other Alternatives:

1. **Performance and Scalability**: Aurora offers high-performance database capabilities with low latency and high throughput. Its architecture is designed for scalability, allowing seamless scaling of compute and storage resources to accommodate growing datasets and user loads.

2. **Compatibility with MySQL**: Aurora is MySQL-compatible, providing compatibility with MySQL databases and tools. In my application, I have inserted and updated the records into RDS using MySql queries.

3. **High Availability and Durability**: Aurora offers built-in high availability and durability features, including automated backups, continuous monitoring, and automatic failover. This ensures data availability and minimizes downtime, critical for maintaining a reliable and responsive application.

4. **Performance Optimization**: Aurora includes features for optimizing database performance, such as read replicas, which offload read traffic from the primary instance, enhancing overall system performance and scalability.

2.3.3 Network

Virtual Private Cloud

In my application architecture for Postcard, I've opted to utilize AWS Virtual Private Cloud (VPC) to manage the networking aspects of the system. I've created a custom VPC to isolate the application resources and provide a private network environment. I have placed Lambda, RDS (Relational Database Service), and ECS (Elastic Container Service) in a private subnet. This ensures that these resources are not directly accessible from the internet, enhancing security and compliance. The load balancer is deployed in a public subnet to

handle incoming traffic and distribute it to the backend services running in private subnets. Placing the load balancer in a public subnet allows it to have a public IP address and receive requests from the internet while still ensuring that the backend services remain protected.

Reasons for Choosing VPC and Subnet Configuration:

1. **Security**: By segregating resources into private subnets, sensitive components like Lambda functions and databases are shielded from unauthorized access from the internet, reducing the attack surface and strengthening overall security posture.
2. **Performance**: Placing resources like RDS and ECS in private subnets helps improve network performance by reducing latency and avoiding potential bottlenecks associated with internet-bound traffic.
3. **Scalability**: The VPC design facilitates horizontal scalability by allowing the seamless addition of new resources within the private subnets as the application grows, without compromising security or introducing network complexity.

2.3.4 General

SNS

I have used SNS in my application to send email notifications to the specific user with the customized postcard link. SNS was suitable in my case because I wanted to notify my user regarding the postcard.

Reasons for Choosing SNS

1. **Flexibility**: It provides support for multiple delivery protocols, including email, SMS, and push notifications, allowing for versatile communication channels based on user preferences.

2. **Reliability**: With built-in redundancy and fault tolerance, SNS ensures high availability and message delivery reliability, crucial for a real-time notification system.

3. **Scalability**: SNS allows for the distribution of messages to a large number of recipients simultaneously, ensuring that notifications are delivered promptly to all users.

Textract

Textract is employed to extract text from uploaded documents, enabling the inclusion of handwritten or typed messages in the postcard.

Reasons for Choosing Textract

1. **Accuracy**: Textract utilizes advanced machine learning algorithms to accurately extract text from a variety of document formats, ensuring the fidelity of the extracted content.

2. **Automation**: By automating the text extraction process, Textract reduces manual effort and enhances operational efficiency, enabling seamless integration into the postcard creation workflow.

3. **Versatility**: Textract supports a wide range of document types, including images, PDFs, and scanned documents, making it suitable for diverse use cases within the application.

Load Balancer

1. **Distribution of Traffic**: A Load Balancer is employed to evenly distribute incoming traffic across multiple instances of the Postcard application, enhancing reliability and fault tolerance.

2. **Scalability**: Load Balancers facilitate auto-scaling by dynamically adjusting the distribution of traffic based on application demand, ensuring optimal performance during peak usage periods.

3. **High Availability**: By distributing traffic across multiple backend instances, Load Balancers enhance the availability of the application, mitigating the impact of potential instance failures.

Part 3: Deployment Model

My application is based on the Public Cloud, specifically leveraging services from AWS (Amazon Web Services).

Reasons why I chose this deployment model:

1. **Scalability**: Public cloud environments like AWS provide elasticity and scalability, allowing to seamlessly scale my application resources up or down based on demand. This is crucial for accommodating a potentially large user base and fluctuating traffic patterns efficiently.

2. **Global Reach**: Public cloud service providers have a global presence with data centers located across multiple regions. This allows me to reach users worldwide easily, ensuring low latency and improved user experience regardless of geographical location.

3. **Cost-Effectiveness**: Public cloud providers offer a pay-as-you-go pricing model, enabling me to optimize costs by paying only for the resources we use. This eliminates the need for upfront hardware investments and reduces operational expenses associated with managing on-premises infrastructure.

4. **Managed Services**: Public cloud providers handle the underlying infrastructure, including hardware provisioning, maintenance, and security updates. This offloads the burden of infrastructure management from the team, allowing me to focus on application development and innovation.

5. **High Availability**: Public cloud environments offer built-in redundancy and fault tolerance mechanisms, ensuring high availability of my application. In the event of hardware failures or outages, the cloud provider automatically redirects traffic to available resources, minimizing downtime.

6. **Faster Time-to-Market**: Leveraging the public cloud enables rapid deployment of new applications and services, reducing time-to-market. This agility allows me to respond quickly to changing market demands and deliver value to my users more efficiently.

7. **Reliability**: By relying on a reputable public cloud provider like AWS, we benefit from their robust infrastructure and adherence to industry-leading security and compliance standards. This instills confidence in the reliability and security of my application and data.

8. **Flexibility**: Public cloud environments offer a wide range of services and tools that cater to diverse application requirements. This flexibility allows me to choose the most suitable services for specific use case, ensuring optimal performance and efficiency

Part 4: Delivery Model

The delivery model I have chosen for my application is Software as a Service (SaaS). SaaS involves accessing software online via a subscription, rather than purchasing and installing it on individual devices. In this model, the software vendor hosts and maintains the servers, databases, and application code.

Advantages of SaaS for my Application:

1. **Accessibility**: My application, being SaaS-based, can be accessed from any device with an internet connection. This ensures users can manage their tasks from anywhere, enhancing flexibility and convenience.

2. **Maintenance and Upgrades**: In the SaaS model, we are responsible for maintaining the application and performing updates. Users benefit from always having access to the latest version without the hassle of maintenance tasks.

3. **Cost Effectiveness**: SaaS eliminates the need for users to invest in hardware infrastructure, reducing upfront costs. Additionally, the subscription-based pricing model spreads the cost of software, hardware, and maintenance across multiple users, making it cost-effective.

Why SaaS is Chosen Over Other Delivery Models:

- **Platform as a Service (PaaS):** While PaaS offers a platform for application development and management, my application is already developed and does not require further modification by users. Therefore, SaaS is a more suitable model.

- **Infrastructure as a Service (IaaS):** Although IaaS provides flexibility, it requires users to manage a significant portion of the system, from the application down to the virtualization layer. This level of management is unnecessary for my application, making SaaS the preferred choice.

Part 5: Final Architecture

5.1: How do all of the cloud mechanisms fit together to deliver the application

In the context of the postcard application, the cloud mechanisms work together to ensure seamless operation and delivery of features to users:
The frontend of the application is hosted on ECS, providing a user interface based on ReactJS. Elastic Load Balancing (ELB) forwards incoming requests from users to the backend services, ensuring efficient distribution of traffic and high availability.
The backend of the application comprises Springboot applications also deployed on ECS. It is configured to listen to requests on a specific port and handle incoming requests from the frontend.
Amazon S3 is utilized for storing images and documents uploaded by users, providing durable and scalable storage. Relational Database Service (RDS), specifically Aurora (MySQL), hosts the database for storing structured data related to postcards like the image and document urls and the extracted text.
AWS Lambda functions are triggered when the images and the documents are uploaded in the S3 bucket and then my lambda function would call the textract and return the extracted text which would be stored in the Aurora database. Once we have the extracted text the we can send the email to the user using SNS.

5.2: Where is data stored?

In my application, data is stored in two primary locations:

1. **S3 Bucket**: Images and documents uploaded by users are stored in the S3 bucket named cloud-media-upload bucket in the images and documents folder respectively. S3 provides durable and scalable object storage, ensuring that uploaded files are securely stored and easily accessible.

2. **Aurora Database**: The Aurora database stores additional metadata related to the uploaded images and documents. Specifically, it maintains the URLs of the stored images and documents, along with any extracted text. This relational database ensures structured data management and facilitates efficient querying and retrieval of information associated with the uploaded content.

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

For my postcard application, I utilized Spring Boot for the backend development and React for the frontend. Here's how these choices were made and where code was required in the application:

Backend with Spring Boot:
Spring Boot was chosen for the backend due to its robustness, ease of development, and strong community support. It provides a comprehensive framework for building Java-based web applications.
The backend code was necessary to handle various functionalities such as handling the logic to upload the image and document submitted by the user and also fetching them after they are processed. They are also used to interact with the S3 bucket to store the data.
Spring Boot allowed me to implement RESTful APIs efficiently, enabling communication between the frontend and backend components of the application.

Frontend with React:
React was selected for the frontend development because of its component-based architecture, which promotes code reusability and maintainability. Additionally, React's virtual DOM ensures optimal performance by efficiently updating the user interface.
Frontend code was required to create the user interface, handle user interactions, and communicate with the backend to fetch and display data. In my application, I provided the user interface so that the user can upload the image and document and once after processing their postcard is ready it can be displayed to the user through my website.

5.4: How is your system deployed to the cloud?

My system is deployed to the AWS cloud using AWS CloudFormation. CloudFormation allows me to model my entire infrastructure in a text file, providing an automated way to quickly and reliably provision the resources needed for my application.
I have created a CloudFormation template that defines the architecture and resources required for my application deployment. This template includes configurations for AWS services such as ECS, Lambda functions, RDS database instances, S3 buckets, SNS and other necessary components.
By using CloudFormation, I can specify the desired state of my infrastructure in a declarative manner, defining the relationships between resources and their configurations. This approach enables me to easily replicate the deployment environment across different AWS regions or for different environments (e.g., development, staging, production) with consistency and reliability.
Figure [x] represents the design of my CloudFormation stack used to deploy the infrastructure and services to the cloud. This visual representation helps in understanding the architecture and dependencies of the deployed resources, facilitating easier management and troubleshooting.
Overall, CloudFormation simplifies the deployment process by automating the provisioning of resources according to the defined template, reducing manual effort and ensuring consistency in the deployed environment.

5.5: If your system does not match an architecture taught in the course, you will describe why that is, and whether your choices are wise or potentially flawed.

My application is following service load balancing architecture. Elastic Load Balancing (ELB) is used to distribute incoming requests across multiple instances or containers in my backend services, such as ECS (Elastic Container Service) or Lambda functions. It ensures that each request is routed to a healthy target, optimizing resource utilization and providing fault tolerance by automatically detecting and rerouting traffic away from unhealthy targets. This architecture efficiently distributes incoming traffic across multiple backend targets, ensuring high availability, fault tolerance, and scalability of my application in the AWS cloud environment.

Part 6: Data security in Application

6.1: Security mechanism to keep data secure at all layers

1. **Network Level Security**
   Technology Used: Virtual Private Cloud (VPC), Security Groups, Network ACLs.

   My application operates within a Virtual Private Cloud (VPC), which provides isolated network segments for enhanced security. VPC allows to define network access control policies, including inbound and outbound traffic rules, to restrict access to resources based on IP addresses, ports, and protocols.
   By configuring security groups and network access control lists (ACLs), I have controlled the traffic flow between different components of my application, enforcing the principle of least privilege.

2. **Compute Level Security**
   Technology Used: ECS tasks, Lambda functions, IAM roles and policies.

   ECS tasks and Lambda functions run within private subnets of my VPC, ensuring that compute resources are not directly accessible from the internet. Access to ECS containers and Lambda functions is restricted using IAM (Identity and Access Management) roles and policies, allowing only authorized entities to execute code and access resources.

3. **Storage Layer Security**
   Technology Used: Amazon S3 (Server-Side Encryption), Relational Database Service (RDS).

   Data stored in Amazon S3 buckets and RDS databases is encrypted at rest by default using server-side encryption (SSE). The access to S3 buckets and RDS instances is controlled through IAM policies of LabRole, which allowed me to grant permissions to the users. Additionally, my RDS is present in the private subnet making it secure.

4. **Application Layer Security**

Technology Used: AWS services like SNS, Textract, Elastic Load Balancing (ELB).

AWS services like SNS and Textract authenticate requests using AWS credentials and enforce encryption in transit to protect sensitive information.
Elastic Load Balancing provides SSL termination, allowing HTTPS traffic to be decrypted at the load balancer and re-encrypted before forwarding it to backend services, maintaining end-to-end encryption.

6.2: Please explain where your data is vulnerable and how you could address these vulnerabilities with further work.

To address potential vulnerabilities and enhance the security of my postcard application, several measures can be implemented:

1. **Implement HTTPS**:
   Currently, my application uses HTTP for communication between the client and the server. By transitioning to HTTPS and configuring SSL/TLS certificates for my frontend hosted on ECS, I can establish a secure communication channel, reducing the risk of eavesdropping and data tampering during transit.

2. **Enhance IAM Role Permissions**:
   Currently, limitations exist in modifying existing configurations in the Lab role. However, gaining permissions to modify the role would enable me to apply security patches. This proactive approach can mitigate the potential risks associated with security vulnerabilities and data breaches.

3. **User Authorisation and Authentication**:
   Currently, the user which is visiting my application is not authenticated due to which there are chances that unauthorised user visits my website. To mitigate this, I can use AWS Cognito in future which supports multi-factor authentication and encryption of data-at-rest and in-transit, providing a secure environment for user data.

Part 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.

To reproduce the architecture of my postcard application in a private cloud environment while maintaining the same level of availability and redundancy as in original public cloud implementation, I would need to invest in several key components:

1. **Hardware Infrastructure:**
   Servers: I would need physical servers to host my application components, such as the frontend, backend, and database. These servers should be powerful enough to handle the application's workload effectively.

   Networking Equipment: I would require routers, switches, and firewalls to connect the servers and manage network traffic securely. These devices ensure that data flows smoothly between my servers and protect against unauthorized access.

   Storage Arrays: To store the application data, including images and documents files, I will need storage arrays with sufficient capacity and performance.

   Estimated cost per server is around $3,000 to $5,000, depending on the hardware specifications. So for my infrastructure, I would need roughly 3 to 5 servers which would cost around $25,000.

2. **Virtualization and Orchestration Software:**
   Virtualization Platform: Virtualization software will enable me to create virtual machines (VMs) on the physical servers, maximizing server utilization.

   Orchestration Software: Orchestration tools will help automate the deployment, management, and scaling of my virtualized infrastructure, streamlining processes like provisioning new VMs and monitoring resource usage.

   Software such as VMware vSphere or Microsoft Hyper-V are efficient for resource management. Estimate licensing fees ranging from several hundred to a few thousand dollars per server, totaling around $4,000 to $8,000 for 5 servers.

3. **Networking and Security Software:**
   Firewall and Security Software: I will need robust security measures, including firewalls and security software, to protect my private cloud environment from unauthorized access and malicious activity.

   Load Balancing Software: Load balancing software will help distribute incoming traffic across multiple servers, ensuring optimal performance and high availability for my application.

   Networking hardware including routers, switches, and firewalls to establish connectivity and enforce security policies within the private cloud environment. Estimate for networking equipment is around $5,000 to $10,000.

4. **Database Software:**

I will require a reliable database management system to store and manage my application's data. Similar to my current setup with RDS Aurora, the database software will play a crucial role in ensuring data integrity and accessibility.

5. **Monitoring Tools:**
   Monitoring tools will be essential for keeping track of the health and performance of my infrastructure and applications. These tools will provide insights into resource usage, identify any potential issues, and alert to any problems before they impact my users.

   Overall, by investing in these components, I can replicate the architecture of my postcard application in a private cloud environment while maintaining the same level of availability and redundancy as my current setup in the public cloud.

Part 8: Which cloud mechanism would be most important for you to add monitoring

For my postcard application, ensuring that the cloud costs remain within budget is crucial to maintaining the sustainability of my service. The cloud mechanisms that require the most monitoring in this regard are the Compute components, specifically the AWS ECS instances and AWS Lambda functions.

1. **AWS ECS Instances:**
   In my postcard application, ECS instances serve as the backbone of my backend infrastructure, application's backend services and handling incoming traffic.
   These ECS instances are responsible for dynamically scaling to meet the demands of incoming traffic, ensuring optimal performance and responsiveness for the users.
   Monitoring ECS instances allows me to closely track resource utilization, including CPU and memory usage, disk I/O, and network traffic. By analyzing these metrics, we can identify any instances that are underutilized or overprovisioned, optimizing instance types and configurations to minimize costs.
   Additionally, ECS instances incur costs based on factors such as instance type, running hours, and data transfer. By monitoring these metrics, we can accurately forecast the monthly expenses and identify opportunities for cost savings.

2. **AWS Lambda Functions:**
   Lambda functions play a critical role in my postcard application, handling tasks such as using the textract service and updating the database
   These serverless functions are invoked in response to the trigger added to the S3 bucket.
   Monitoring Lambda functions allows to track the frequency and duration of function invocations, as well as resource utilization metrics such as memory allocation and execution time.
   By analyzing these metrics, I can identify any functions that are being invoked excessively or running for extended periods, optimizing code and configurations to reduce execution costs.

Additionally, Lambda functions are billed based on the number of invocations and execution duration. Monitoring these metrics enables me to accurately predict my usage-based charges and identify opportunities for cost optimization.

In summary, implementing comprehensive monitoring for AWS ECS instances and Lambda functions is essential for effectively managing cloud costs in our postcard application. By closely tracking resource utilization, analyzing usage patterns, and optimizing configurations, we can ensure that our cloud spending remains within budget while delivering a seamless and cost-effective user experience.

Part 9: How would your application evolve if you were to continue development?

If I will continue the development of my postcard application, there are several features I would consider adding to enhance its functionality and user experience.
Here's how the application might evolve, along with the cloud mechanisms which I could use to implement these features:

1. **Real-Time Collaboration:**
   Allow users to collaborate in real-time on creating and designing postcards. Users could invite others to join a collaborative session where they can work together on designing a postcard, adding images, text, and other elements.

   Cloud Mechanisms:
   I could implement real-time collaboration using WebSockets for bi-directional communication between clients and the server. Services like AWS IoT Core or Amazon API Gateway with WebSocket support could facilitate real-time messaging between users.

2. **Template Marketplace:**
   Create a marketplace where users can browse and purchase pre-designed postcard templates created by professional designers. Users could customize these templates with their own images and text.

   Cloud Mechanisms:
   We could utilize AWS Lambda and Amazon S3 to manage the storage and retrieval of template images and metadata. AWS DynamoDB could be used to store information about available templates, such as pricing, categories, and ratings.

3. **Advanced Image Editing:**
   Enhance the image editing capabilities of the application by adding features such as filters, overlays, and cropping tools. Users could apply various effects to their images to personalize their postcards.

Cloud Mechanisms:
We could leverage AWS Lambda for serverless image processing tasks, such as applying filters and transformations to images. Amazon S3 would continue to be used for storing and serving user-uploaded images.

4. **Integration with Social Media Platforms:**
   Allow users to share their postcards directly to social media platforms like Facebook, Instagram, and Twitter. Integration with social media APIs would enable seamless sharing of postcards with friends and followers.

   Cloud Mechanisms:
   We could use AWS API Gateway to create RESTful APIs for interacting with social media platforms. AWS Lambda functions could handle the authentication and posting of content to social media APIs.

5. **Analytics and Personalization:**
   Implement analytics to track user interactions with the application and personalize the user experience based on their preferences and behavior. Analyzing user data could help improve engagement and retention.

   Cloud Mechanisms:
   AWS services like Amazon Redshift could be used for data warehousing and analytics. Amazon Personalize could be employed for building recommendation models to suggest postcard designs based on user preferences.

By integrating these features into my postcard application and leveraging various cloud mechanisms provided by AWS, I can continue to enhance the functionality, usability, and scalability of the application to meet the evolving needs of my users.

References:

[1] A. Martin, “What is ethical hacking?,” *MyComputerCareer*. [Online], December 30, 2021. Available: https://www.mycomputercareer.edu/news/what-is-ethicalhacking/#:~:text=These%20include%3A,boundaries%20set%20by%20the%20organizatio n. [Accessed: January 27, 2024].

[2] K. Lange, “The ethical hacking guide: Hacking for security,” *Splunk*. [Online]. Available: https://www.splunk.com/en_us/blog/learn/ethical-hacking.html. [Accessed: January 27, 2024].

[3] R. Shapland, “Top 5 key ethical hacker skills,” *Security*, December 1, 2022. [Online]. Available: https://www.techtarget.com/searchsecurity/tip/Top-key-ethical-hacker-skills. [[Accessed: January 27, 2024].

[4] “Wireshark Q&A,” *Wireshark.org*. [Online]. Available: https://osqaask.wireshark.org/questions/60763/how-to-detect-ddos-attacks-on-my-network/. [Accessed: January 27, 2024].

[5] “How to perform TCP SYN Flood DoS attack & detect it with Wireshark - Kali Linux hping3,” *Firewall.cx*. [Online]. Available: https://www.firewall.cx/tools-tipsreviews/network-protocol-analyzers/performing-tcp-syn-flood-attack-and-detecting-itwith-wireshark.html. [Accessed: January 27, 2024].

[6] “Snort - intrusion detection system & prevention system,” *Techofide.com*. [Online]. Available: https://techofide.com/blogs/snort-intrusion-detection-system-preventionsystem-installation-use-in-linux/. [Accessed: January 27, 2024].

[7] B. Matthew, “Detecting DDOS attacks and Port Scanning with snort,” *Medium*, August 8, 2023. [Online]. Available: https://medium.com/@bmatth21/detecting-ddos-attacks-andport-scanning-techniques-with-snort-11e249a5eba9. [Accessed: January 27, 2024].