CSCI 5409 Adv Topics in Cloud Computing

Term Assignment Report

Prepared by:
Vikram Venkatapathi - B00936916

Master of Applied Computer Science (Summer'23)
Faculty of Computer Science
Dalhousie University

GitLab Repo link : **https://git.cs.dal.ca/courses/2023-summer/csci4145-5409/vikramv**

Contents

Table of Contents 2

I. Introduction 1

1. Introduction 2
1.1. Purpose of the Application . . . 2
1.2. Technology Stack . . . 2
1.3. Target Users . . . 2
1.4. Performance Targets . . . 2

II. Menu Item Requirements 3

2. Menu Item Requirements 4
2.1. User Registration and Login . . . 4
2.2. Image Upload and Storage . . . 4
2.3. Image Analysis and Report Generation . . . 4
2.4. Notification and Event Handling . . . 5
2.5. Cost comparison table . . . 5

III. Deployment Model 8

3. Deployment Model 9

IV. Delivery Model 10

4. Delivery Model 11
4.1. Reason for Choosing the Delivery Model . . . 11
4.2. Justification for Hybrid Components . . . 12

V. Final Architecture 13

5. Final Architecture 14
5.1. Architecture Diagram . . . 14

5.2. Cloud Mechanisms and Services . . . 14
5.3. Programming Languages . . . 15
5.4. System Deployment . . . 15
5.5. Comparison of my architecture to the ones taught in lectures . . . 15
5.5.1. Dynamic Scalability Architecture . . . 16
5.5.2. Elastic Disk Provisioning Architecture . . . 16
5.5.3. Redundant Storage Architecture . . . 16
5.5.4. Service-Oriented Architecture (SOA) . . . 16
5.6. Conclusion . . . 17

VI. Data Security **18**

6. Data Security Vulnerabilities & Resolution in Application Architecture **19**
6.1. Public Buckets . . . 19
6.1.1. Vulnerability . . . 19
6.1.2. Resolution . . . 19
6.2. Exposed Backend . . . 19
6.2.1. Vulnerability . . . 19
6.2.2. Resolution . . . 19
6.3. Report Bucket Accessibility . . . 20
6.3.1. Vulnerability . . . 20
6.3.2. Resolution . . . 20
6.4. Conclusion . . . 20

VII. Private Cloud Reproduction **21**

7. Reproducing my architecture on-premise **22**
7.1. Cost estimation . . . 22
7.1.1. AWS Pricing Calculator . . . 22
7.1.2. Hardware Vendor Websites . . . 22
7.1.3. Software Licensing Costs . . . 22
7.1.4. Email Notification Service Providers . . . 22
7.1.5. Serverless Frameworks . . . 23
7.1.6. Database Server Costs . . . 23
7.1.7. Object Storage Devices . . . 23
7.2. My estimation report . . . 23

VIII. Cost Monitoring **25**

8. Monitoring AWS Lambda for Cost Control **26**
8.1. Justification . . . 26

IX. Future Development 27

9. Application Evolution and Future Features 28

X. Conclusion 30

10. Conclusion 31

XI. REFERENCES 32

Part I.

Introduction

1. Introduction

The cloud application developed for my term assignment is titled **"Serverless Image Processing System"**. This project aims to provide users with an intuitive and feature-rich platform to analyze images, detect labels, and analyze face details using Amazon Rekognition. The application leverages a combination of modern technologies and AWS cloud services to deliver a seamless and efficient user experience.

1.1. Purpose of the Application

The primary purpose of the application is to allow users to register themselves with basic information such as name, email, and password. Once registered, users can log in using their credentials and gain access to the image analysis functionalities. The application enables users to upload images, trigger an image analysis process, and receive a comprehensive report detailing the detected labels, identified celebrities, and analyzed facial details.

1.2. Technology Stack

The application is built upon a React-based frontend, providing an interactive and responsive user interface. The frontend communicates with a Flask-based backend, which handles user authentication, manages data interactions, and orchestrates AWS service integrations.

1.3. Target Users

The application is designed for users who require image analysis capabilities for various purposes, such as photographers, social media influencers, and researchers. The user interface is straightforward and user-friendly, catering to both tech-savvy individuals and those less familiar with advanced cloud technologies.

1.4. Performance Targets

The application is expected to provide a responsive user experience, with minimal latency during image analysis and report generation. The goal is to deliver accurate and insightful results to users within a reasonable amount of time. Additionally, the application aims to maintain high availability, ensuring that users can access the services without significant downtime or interruptions.

Part II.

Menu Item Requirements

2. Menu Item Requirements

2.1. User Registration and Login

1. Service Used: DynamoDB
2. Alternative Considered: Amazon RDS (Relational Database Service)
3. Reason for Selection:
    a) DynamoDB was chosen for user registration data storage due to its scalability, low-latency performance, and ease of integration with AWS services.
    b) As the application primarily deals with simple user data, a NoSQL database like DynamoDB is well-suited for this purpose.
    c) Amazon RDS, while a robust option for relational data, would have been an overkill for storing user registration data, and its pricing model might have been less cost-effective for this specific use case.

2.2. Image Upload and Storage

1. Service Used: S3 (Simple Storage Service)
2. Alternative Considered: EFS (Elastic File System)
3. Reason for Selection:
    a) S3 was chosen for image storage due to its durability, scalability, and cost-effectiveness.
    b) S3 is purpose-built for object storage and ideal for handling image files.
    c) While EFS provides shared file storage with high availability, it is better suited for scenarios where multiple EC2 instances need shared access to the same files.
    d) For this application, where images are uploaded by users and later analyzed asynchronously, S3's object storage model fits the requirements perfectly.

2.3. Image Analysis and Report Generation

1. Services Used: Amazon Rekognition, Step Functions
2. Alternative Considered: AWS Lambda, AWS Batch
3. Reason for Selection:

a) Amazon Rekognition was chosen for image analysis due to its powerful AI capabilities, particularly its label detection, celebrity recognition, and facial analysis features.

b) This service streamlines the process of image analysis without the need for manual coding.

c) Step Functions were selected to orchestrate the analysis workflow as they offer a serverless, scalable, and visually intuitive way to coordinate AWS services.

d) While AWS Lambda is suitable for event-driven functions, Step Functions provide better control and coordination for complex workflows.

e) AWS Batch is more suitable for large-scale batch processing, which is not the primary requirement for this real-time image analysis.

2.4. Notification and Event Handling

1. Services Used: SNS (Simple Notification Service), Amazon EventBridge

2. Alternative Considered: AWS SQS (Simple Queue Service), Amazon SES (Simple Email Service)

3. Reason for Selection:

   a) SNS was chosen for sending notifications to users due to its simplicity, ease of use, and support for multiple delivery protocols.

   b) It enables direct communication with users through email, SMS, and other endpoints.

   c) Amazon EventBridge was utilized to set up rules and triggers for the Step Function whenever an image is uploaded to S3.

   d) This serverless event bus simplifies event-driven architectures. While AWS SQS could handle message queuing, SNS's direct communication capabilities were more appropriate for sending notifications to users in real-time.

   e) While SES (Simple Email Service) could handle email notifications, it is not available in the academy account. Therefore, SNS provides a suitable alternative for sending notifications via email, SMS, and other protocols

2.5. Cost comparison table

Table 2.1.: Comparison of Services and Alternatives in Region US-EAST-1 (N. Virginia)

Service	**Pricing**	**Usage Cost per Month**	**Reason for Selection**
User Registration and Login			
DynamoDB	Pay per provisioned read/write capacity units and storage	For "Read Request Units (RRU)": $0.25 per million read request units For "Write Request Units (WRU)": $1.25 per million write request units For "Data Storage": First 25 GB stored per month is free using the DynamoDB Standard table class $0.25 per GB-month thereafter [29]	Scalable, low-latency, and cost-effective for user data storage
Amazon RDS	Pay per instance type and storage	Standard Instances - db.t4g.micro = $0.016 /hr General Purpose SSD storage (single AZ) (gp2) - Storage $0.115 per GB-month[30]	Overkill for simple user data, cost-effectiveness concerns
Image Upload and Storage			
S3	Pay per storage and data transfer	S3 Standard - General purpose storage for any type of data, typically used for frequently accessed data First 50 TB / Month $0.023 per GB Data Transfer IN = $0, Data Transfer OUT = $0,(since no data is transferred out) [31]	Ideal for handling image files, durable and cost-effective
EFS	Pay per storage and throughput	Effective storage price (\$0.043/GB-Mo) - One Zone* [32]	Better for shared access to files, not primary requirement

Table 2.2.: Comparison of Services and Alternatives in Region US-EAST-1 (N. Virginia) contd..

Service	**Pricing**	**Usage Cost per Month**	**Reason for Selection**
Image Analysis and Report Generation			
Amazon Rekognition	Pay per image	Group 2 - First 1 million images - $0.001/image[33]	Powerful AI capabilities, simplifies image analysis
AWS Lambda	Pay per request and duration	First 6 Billion GB-seconds / month $0.0000166667 for every GB-second $0.20 per 1M requests[34]	Suitable for event-driven functions, but Step Functions offer better control
AWS Batch	Pay per vCPU and memory usage	There is no additional charge for Amazon Batch.You only pay for what you use, as you use it[35]	More suited for large-scale batch processing, not real-time
Notification and Event Handling			
SNS	Pay per Email send	Email/Email-JSON - $2.00 per 100,000 notifications[36]	Simplicity, direct communication with users
AWS SQS	Pay per request	First 1 Million Requests/Month - FREE [37]	Suitable for message queuing, but not ideal for real-time notifications
Amazon SES	Pay per email sent and received	Outbound email from non-EC2 $0.10/1000 emails[38]	Not available in the academy account, SNS provides a suitable alternative

Part III.

Deployment Model

3. Deployment Model

For my application, I have chosen a **Hybrid Cloud deployment model**. I made this decision based on several key factors that align with my application's requirements and objectives. [22]

1. **Security Concerns**: Firstly, my application has specific security concerns that require me to maintain a high level of control and isolation over sensitive data and services. By adopting a hybrid cloud approach, I can keep the most critical components of my application within a private cloud environment. This allows me to implement stringent security measures, maintain data sovereignty, and ensure compliance with industry regulations.

2. **Scalability**: Secondly, while I haven't implemented scalability with a load balancer, I anticipate the possibility of increased resource demands as my user base grows. With the hybrid cloud model, I have the flexibility to scale certain components of my application using the resources available in the public cloud. This enables me to meet varying workloads and spikes in traffic without compromising on security or incurring unnecessary costs.

3. **Cost Feasibility**: Cost-wise, I find that a hybrid cloud deployment is still feasible for my application. By carefully managing resource allocation and leveraging the pay-as-you-go pricing model of the public cloud, I can optimize costs while benefiting from the advantages of cloud computing, such as easy provisioning and dynamic resource allocation.

4. **Geographical Distribution**: Lastly, my application is not geographically distributed, so there is no immediate need to have data and services spread across multiple regions. The hybrid cloud allows me to concentrate my resources in a controlled environment, providing better performance and consistent service delivery to my users.

In summary, I chose the hybrid cloud deployment model as it strikes the right balance between security, scalability, cost-effectiveness, and performance. It allows me to address my specific security concerns, efficiently handle potential growth, and make the most of cloud resources while maintaining a level of control that aligns with my application's unique needs.

Part IV.

Delivery Model

4. Delivery Model

The chosen delivery model for the cloud application is a **Software-as-a-Service (SaaS) model**. In a SaaS delivery model, the application is centrally hosted and managed by the service provider (in this case, the development team/me - the developer), and users access the application over the Internet. Users do not need to install or maintain any software on their local devices, as the entire application is delivered as a service through a web browser. [23]

4.1. Reason for Choosing the Delivery Model

1. **Accessibility and Convenience**: The SaaS delivery model offers a high level of accessibility and convenience for users. They can access the application from any device with an internet connection and a web browser. This eliminates the need for users to install any client-side software, making it easy to use and access the application from anywhere.

2. **Rapid Deployment**: As a SaaS application, the development team can deploy updates and new features quickly and efficiently. Since the application is hosted centrally, there is no need to distribute updates to individual users. This rapid deployment process allows for continuous improvements and enhances the user experience.

3. **Scalability**: SaaS applications are inherently scalable, as the infrastructure is managed by the service provider. As the user base grows, the application can easily accommodate additional users without any disruptions or manual intervention.

4. **Cost-Effective**: The SaaS model is cost-effective for both the service provider and the users. For the service provider, resources can be optimized and shared among users, resulting in cost efficiencies. For users, there are no upfront costs or hardware requirements, as they only pay for the subscription or usage, making it an affordable option.

5. **Maintenance and Support**: With the SaaS model, the development team is responsible for maintaining and supporting the application. Users do not need to worry about updates, patches, or technical issues, as the service provider handles all these aspects. This frees up users' time and resources, allowing them to focus on using the application for their specific tasks.

6. **Security and Data Protection**: The SaaS model enables the development team to implement robust security measures centrally. This ensures that data is protected, and users can trust the application with their sensitive information. Additionally, the team can monitor and respond to security threats proactively.

7. **Global Accessibility**: As a web-based application, the SaaS model enables users from around the world to access and use the application without any geographical limitations. This global accessibility enhances the application's reach and potential user base.

Considering the nature of the cloud application, the SaaS delivery model aligns well with the goals of providing image analysis capabilities to a broad range of users. The model ensures easy accessibility, continuous updates, and a cost-effective solution for users while allowing the development team to maintain and improve the application efficiently.

However, it is essential to acknowledge that the application also incorporates some hybrid components, which involve the use of EC2-Infrastructure-as-a-Service (IaaS) & S3-Storage-as-a-Service(STaaS).

4.2. Justification for Hybrid Components

The inclusion of EC2 and S3 complements my architecture by fulfilling the following requirements:

1. **Scalability and Flexibility**: EC2 instances provide greater control and flexibility over the application's frontend and backend components, allowing seamless scaling based on traffic demands.

2. **Specialized Storage**: S3 is ideal for storing images and generated reports, offering durable, cost-effective, and scalable data storage.

While the application primarily operates on a SaaS delivery model, the hybrid components (EC2 and S3) serve specific purposes to enhance overall functionality, scalability, and data storage capabilities. The combination of SaaS, IaaS, and STaaS elements ensures an efficient, cost-effective, and responsive cloud application for image processing and analysis.

Part V.

Final Architecture

5. Final Architecture

5.1. Architecture Diagram

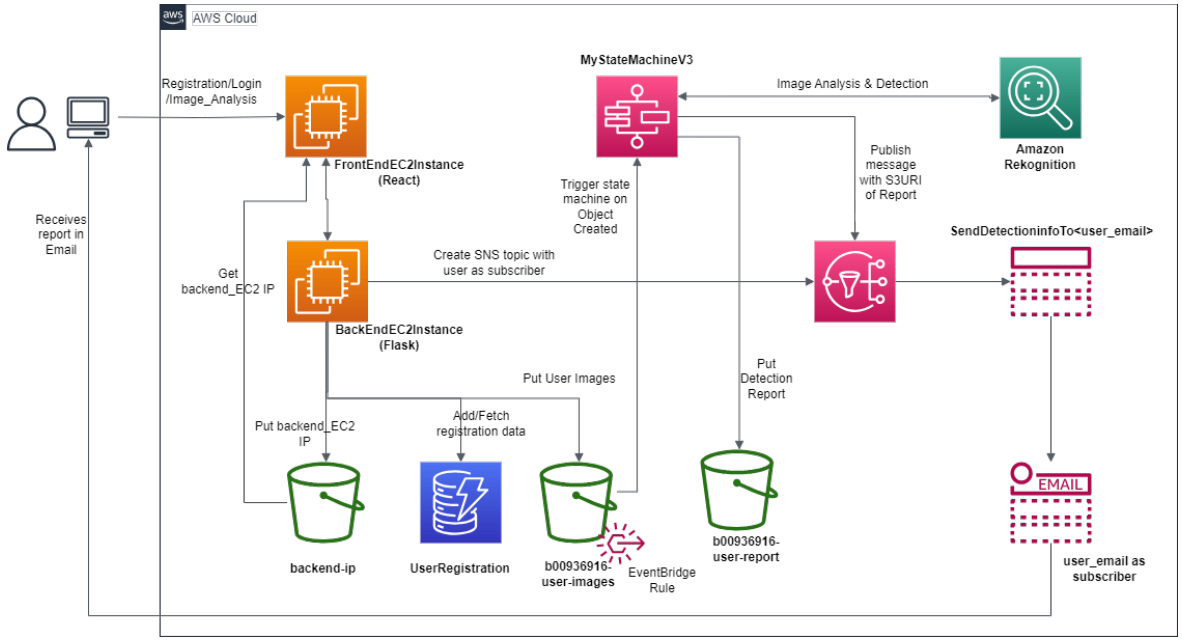

Figure 5.1.: ***AWS Architecture diagram of my project (Tool used: draw.io[11])***

The final architecture of the cloud application combines various cloud mechanisms and services to deliver a scalable, efficient, and user-friendly platform for image processing and analysis. The architecture leverages a serverless model for image processing tasks while utilizing other AWS services for data storage and application hosting.

5.2. Cloud Mechanisms and Services

1. **Serverless Architecture**: The core of the application's image processing capabilities is based on a serverless architecture. AWS Lambda is used to execute code in response to image upload events. Lambda functions are responsible for image analysis tasks using Amazon Rekognition. This serverless model ensures cost-efficiency, automatic scaling, and real-time responsiveness.

2. **Hosting**: The frontend and backend of the application is hosted on Amazon EC2 instances. These instances serve as web servers, handling user interactions and communication with the

serverless components. The choice of EC2 for hosting provides greater control and flexibility over the application's user interface.

3. **Data Storage**: The application utilizes Amazon S3 for object storage and Amazon DynamoDB for NoSQL data storage. Amazon S3 is used to store uploaded images, and DynamoDB is used to store user registration data, including user details such as name, email, and password.

4. **State Machine**: AWS Step Functions are utilized to orchestrate the image analysis workflow. The state machine coordinates the various steps involved in analyzing the uploaded image using Amazon Rekognition and generating the report.txt file.

5.3. Programming Languages

1. **Backend**: The backend of the application is implemented using **Flask, a Python-based web framework**. Python is chosen for its simplicity, ease of use, and extensive libraries, making it suitable for developing the application's server-side logic and integrating with AWS services.

2. **Frontend**: The frontend is developed using **React.js, a popular JavaScript library** known for its flexibility, component-based structure, and efficient rendering. React.js enables the creation of dynamic and interactive user interfaces, enhancing the user experience.

5.4. System Deployment

The system is deployed to the cloud by utilizing AWS services, provisioned via cloud formation template. The frontend, implemented using React.js, is hosted on Amazon EC2 instances, making it accessible to users through their web browsers. The backend, built with Flask, is deployed on separate EC2 instances to handle user authentication and communication with Lambda functions for image processing.

Lambda functions for image analysis are triggered automatically by image uploads to Amazon S3, which serves as the event source. AWS Step Functions coordinate the image analysis workflow, ensuring a seamless flow of tasks and generating the report.txt file.

Additionally, Docker images are utilized for both the frontend and backend components of the application. These Docker images facilitate the containerization of the application, ensuring consistency and portability across different environments. The Docker images are hosted on public repository in my personal docker account, allowing for easy management and distribution of the application containers.

5.5. Comparison of my architecture to the ones taught in lectures

Upon reviewing the architectures taught in class, my cloud application architecture aligns closely with the "Dynamic Scalability Architecture" and partially incorporates elements from the "Elastic Disk Provisioning Architecture" and the "Redundant Storage Architecture." Let's compare my architecture with these taught architectures:

5.5.1. Dynamic Scalability Architecture

1. My application's serverless deployment model with AWS Lambda and Step Functions reflects the dynamic scalability architecture. AWS Lambda allows automatic scaling based on the incoming image processing requests, ensuring efficient resource utilization.[25]

2. **Justification**: The serverless architecture is well-suited for your application, which involves sporadic image processing tasks. It automatically scales resources up or down based on demand, optimizing costs and ensuring real-time responsiveness.

5.5.2. Elastic Disk Provisioning Architecture

1. While my application uses Amazon S3 for object storage, it does not fully align with the elastic disk provisioning architecture, as this architecture typically involves provisioning storage volumes with automatic scaling capabilities.[26]

2. **Justification**: The usage of Amazon S3 as object storage is appropriate for my application. S3 is inherently elastic and allows me to store and retrieve any amount of data without worrying about provisioning or managing disk volumes. Its pay-as-you-go pricing model makes it cost-effective for variable storage requirements.

5.5.3. Redundant Storage Architecture

1. My application's use of Amazon S3 for data storage partly aligns with the redundant storage architecture, as S3 automatically replicates data across multiple availability zones to ensure durability and availability.[27]

2. **Justification**: Storing data in Amazon S3 provides a level of redundancy as S3 automatically replicates data across multiple geographic locations, making it highly durable and available. This redundancy ensures that your application's data is protected from potential hardware failures.

5.5.4. Service-Oriented Architecture (SOA)

1. My cloud application utilizes a Service-Oriented Architecture (SOA) to achieve its functionalities efficiently. The architecture consists of loosely coupled and independent services, each handling specific tasks. AWS Lambda serves as the image processing service, triggered by image uploads to Amazon S3. User registration data is stored in Amazon DynamoDB, representing a separate service. The frontend, developed using React.js, acts as a service for user interface and interactions. The backend, implemented with Flask, handles user authentication and communication with the image processing service. Amazon SNS serves as the notification service, communicating messages to users. [28]

2. **Justification**: SOA enables a modular and scalable design, enhancing flexibility, reusability, and maintainability. It promotes seamless integration with other services and simplifies updates, ensuring agility and efficient communication among components. Adopting SOA ensures a robust and future-proof architecture for the cloud application.

5.6. Conclusion

My cloud application architecture is well-designed, combining the advantages of serverless computing with EC2 instances for frontend and backend hosting. It effectively leverages the dynamic scalability architecture with AWS Lambda and Step Functions for image processing tasks. The usage of Amazon S3 for object storage ensures redundancy and data durability, even though it does not strictly align with the elastic disk provisioning architecture. Overall, my choices are justified, as the architecture optimizes performance, cost, and user experience for the image analysis application.

Part VI.

Data Security

6. Data Security Vulnerabilities & Resolution in Application Architecture

6.1. Public Buckets

One notable security concern is the presence of public buckets in my application. Making buckets public can expose sensitive data to unauthorized access, making it a potential target for malicious actors.

6.1.1. Vulnerability

The public buckets can be accessed by anyone with the bucket's URL, leading to potential data leaks and unauthorized data modification.

6.1.2. Resolution

To address this vulnerability, it is crucial to avoid making buckets public whenever possible. Instead, implement fine-grained access control using AWS IAM (Identity and Access Management) and bucket policies. I can grant specific permissions to authenticated users (e.g., registered users) or specific AWS EC2 instances while denying public access. [19]

6.2. Exposed Backend

Although my frontend is exposed to the internet while the backend is only accessible to the frontend EC2 instance, there may still be potential risks.

6.2.1. Vulnerability

If a malicious actor gains access to the frontend EC2 instance, they could potentially leverage it to attempt unauthorized access to the backend.

6.2.2. Resolution

Implement additional security layers, such as secure communication protocols (e.g., HTTPS) and API authentication mechanisms (e.g., API keys or tokens) between the frontend and backend EC2 instances. Additionally, I can consider using a Virtual Private Cloud (VPC) to isolate the backend EC2 instance from the internet, allowing access only through private IP addresses or VPN connections.

6.3. Report Bucket Accessibility

While making the bucket public is required for sending the image processing report to users, it introduces a security trade-off.

6.3.1. Vulnerability

Public access to the report bucket might allow unauthorized users to view or manipulate other users' reports.

6.3.2. Resolution

To mitigate this risk, I can explore alternatives such as generating pre-signed URLs for the report files, which grant temporary access to specific users only. Additionally, I can consider implementing user authentication and authorization mechanisms to control access to the report bucket based on registered user identities.

6.4. Conclusion

Addressing these vulnerabilities involves implementing stricter access controls, securing communication channels, and ensuring that sensitive data is accessible only to authorized users and services. By adopting these security measures, my application can significantly improve data security and protect against potential threats and unauthorized access.

Part VII.

Private Cloud Reproduction

7. Reproducing my architecture on-premise

To replicate my cloud-based application on-premise, the organization needs servers(not high-end) for frontend and backend, Docker licenses for efficient image management, a moderately powerful database server, object storage, image processing software licenses, email notification service, and on-premise serverless features for seamless execution. These investments will enable the organization to achieve a secure, scalable, and high-performing on-premise environment for the application.

7.1. Cost estimation

To estimate the cost of reproducing my architecture on-premise, the organization can use various cloud cost calculators and hardware pricing tools available online. Here are some resources they can use to explore and estimate the expenses:

7.1.1. AWS Pricing Calculator

The AWS Pricing Calculator allows you to estimate the cost of AWS services based on your usage. While you may not be deploying in the cloud, it can give you an idea of the costs associated with different services, which can help in comparing with on-premise solutions. You can find it on the official AWS website.

7.1.2. Hardware Vendor Websites

Check the websites of hardware vendors (e.g., Dell, HP, Lenovo) to explore the pricing of servers and other hardware components you may need for your on-premise deployment. They often have online configurators that allow you to customize and estimate the cost of your desired hardware setup.

7.1.3. Software Licensing Costs

Reach out to the respective software vendors (e.g., Docker, image processing software providers) to inquire about the licensing costs for their products. They usually provide pricing information on their websites or through their sales representatives.

7.1.4. Email Notification Service Providers

For email notification services, explore the pricing plans of email service providers like SendGrid, Mailgun, or Amazon SES. They offer different plans based on the volume of emails sent.

7.1.5. Serverless Frameworks

If you plan to replicate the serverless features of AWS Lambda on-premise, you can explore open-source serverless frameworks like OpenFaaS, Knative, or Apache OpenWhisk. [39]

7.1.6. Database Server Costs

Check the prices of database server software like MySQL, PostgreSQL, or Microsoft SQL Server[40], and consider the hardware requirements for running the database efficiently.

7.1.7. Object Storage Devices

For on-premise object storage, you can explore storage solutions like Network-Attached Storage (NAS) or Storage Area Network (SAN) devices. Check with vendors for pricing and capacity options.

7.2. My estimation report

To estimate the upfront and ongoing costs for running my application on-premise with 10-100 requests a day, we will consider the following components:

1. Upfront Costs for servers: Servers for Frontend and Backend: Cost will vary based on the specifications and quantity of servers we purchase. Let's assume that we will buy the server: E3-1240 v5 (3.50 GHz), 32GB RAM, 500GB SSD, with an estimated upfront cost of $1,748.24 for the required servers. [20]
2. Ongoing Costs (Monthly):
    a) Power and Cooling: Let's assume an estimated monthly cost of $500 for power and cooling for the servers.
    b) Internet Connectivity: Let's assume an estimated monthly cost of $200 for internet connectivity to support the application's traffic.
    c) Amazon Rekognition License: First 1 million images : $0.0010 per image. Based on the estimated 10-100 requests a day, the monthly cost for Amazon Rekognition would be $3.3 (assuming 100 requests/day for 30 days) [21]
    d) MongoDB database server -$57/month for a Dedicated server.[22]
    e) EMail: SNS- First 1 million Amazon SNS requests per month are free, $0.50 per 1 million requests thereafter. Hence, no cost for those 300,000 requests/month. [23]
    f) Serverless cost - entirely depends on no. of requests, time of execution and memory allocated. So, cannot give an estimate.
3. **Total cost = 2,451.3 (approx. for the 1st month, including upfront and monthly costs)**

NOTE: It's important to note that estimating on-premise costs can be complex, and the actual expenses may vary based on factors such as hardware specifications, software licensing terms, maintenance, power, cooling, and other operational costs. Since cloud providers offer pay-as-you-go models, on-premise costs may involve higher upfront capital expenditures. We must be sure to consider long-term TCO (Total Cost of Ownership) to get a more comprehensive comparison.

Part VIII.

Cost Monitoring

8. Monitoring AWS Lambda for Cost Control

The cloud mechanism that would be most important to add monitoring to control costs and prevent unexpected budget escalations is the **AWS Lambda service**. [18]

8.1. Justification

AWS Lambda operates on a pay-as-you-go pricing model, where you are billed based on the number of requests for the functions and the time your code executes. While the serverless nature of Lambda is cost-effective for sporadic or event-driven workloads, it can potentially lead to unexpected costs if not monitored carefully.

The following reasons highlight the importance of monitoring AWS Lambda to manage costs effectively:

1. **Invocation Rate**: Monitoring the invocation rate of Lambda functions is crucial as each invocation incurs a cost. A sudden spike in requests, such as due to a surge in user activity, could significantly impact costs if not anticipated.

2. **Execution Time**: AWS Lambda charges based on the time the code executes. Monitoring the execution time of Lambda functions helps identify functions that consume excessive resources and lead to higher costs.

3. **Memory Allocation**: The amount of memory allocated to Lambda functions affects their cost. Monitoring memory usage ensures efficient memory allocation to avoid unnecessary expenses.

4. **Cold Starts**: Cold starts occur when a function is invoked after being idle, leading to increased latency and potential costs. Monitoring cold starts helps optimize function performance and reduce the associated costs.

5. **Error Rates**: Frequent errors in Lambda functions can result in repeated invocations, leading to higher costs. Monitoring and addressing errors promptly can prevent unnecessary expenses.

6. **Exploring Long-Term Alternatives**: Lambdas are suitable for short-term workloads since their execution time is quick. But for long-term aspects, we should look into alternatives. Such as evaluating the feasibility of transitioning certain workloads to other AWS services that may offer more cost-efficient pricing models

By closely monitoring AWS Lambda and setting up appropriate alarms, cost control measures can be implemented proactively. You can identify cost patterns, optimize resource allocation, and take corrective actions to prevent budget escalations unexpectedly. Proper monitoring also helps in identifying inefficient functions or code, enabling optimization for cost savings.

Part IX.

Future Development

9. Application Evolution and Future Features

If I were to continue the development of my application, I would make the following changes:

1. **User Authentication and Authorization**: Implement a robust user authentication and authorization system to secure user data and control access to various functionalities. Use AWS Cognito for user management, enabling secure sign-up, login, and user attribute management.

2. **User Profile Management**: Allow users to update their profiles, including profile pictures and other relevant information. Use AWS Lambda to handle profile updates and store user data in DynamoDB.

3. **Serverless File Uploads**: Enhance the file upload feature to support multiple file types and larger files. Utilize AWS API Gateway and Lambda to process and store user-uploaded files securely in Amazon S3.

4. **Real-time Notifications**: Integrate WebSocket functionality using AWS API Gateway and AWS Lambda to provide real-time notifications to users, such as when image processing is completed or when new reports are available.

5. **Application Monitoring and Analytics**: Implement monitoring and analytics using AWS CloudWatch and AWS X-Ray to gain insights into application performance, identify bottlenecks, and optimize resource utilization.

6. User **Access Control for Reports**: Enhance security by allowing users to access only their own reports. Implement AWS IAM-based access control to restrict access to specific S3 objects based on user identity.

7. **Data Backup and Archiving**: Set up automated data backup and archiving processes using AWS Data Lifecycle Manager and Glacier for long-term storage of historical data and reports.

8. **Performance Optimization**: Continuously optimize Lambda functions, frontend code, and database queries to improve application responsiveness and reduce latency.

9. **Cost Optimization**: Regularly review cost metrics and optimize resource allocation. Utilize AWS Cost Explorer to analyze costs and identify opportunities for cost-saving measures.

10. **Geolocation Services**: Integrate geolocation services like Amazon Location Service to add location-based functionality, such as tagging images with location data or displaying image locations on a map.

11. **Enhanced Image Analysis**: Explore additional image analysis capabilities using AWS services like Amazon Rekognition Custom Labels to create custom image classification models tailored to specific use cases.

12. **Data Privacy and Compliance**: Implement data privacy and compliance measures to adhere to relevant regulations. Utilize AWS Key Management Service (KMS) for data encryption at rest and in transit.

By incorporating these features and leveraging various AWS cloud mechanisms, the application can evolve into a comprehensive and powerful image analysis platform. Continuous development and improvement will ensure that the application meets user needs, maintains high performance, and remains cost-efficient in the dynamic cloud environment.

Part X.

Conclusion

10. Conclusion

In conclusion, the cloud-based image analysis application has been successfully designed and implemented with a combination of serverless and traditional cloud mechanisms. The application utilizes AWS Lambda and Step Functions for event-driven image processing, enabling automatic scaling and real-time responsiveness. The frontend and backend are hosted on Amazon EC2 instances, providing flexibility and control over application resources.

Though certain buckets have been made public temporarily for report delivery, it is essential to address this security vulnerability by implementing more granular access control mechanisms in the future. AWS CloudWatch facilitates continuous monitoring of application performance and costs, allowing proactive measures to control expenses and optimize resource utilization.

Moving forward, long-term cost optimization and security enhancements will remain a priority. Continuous development may involve adding user authentication using AWS Cognito and implementing features like user profile management and real-time geolocation services.

Overall, the application showcases the potential of cloud computing in delivering scalable, secure, and cost-effective solutions. With a focus on ongoing improvement and adherence to best security practices, the image analysis application is poised to meet evolving user needs and ensure a seamless and secure experience in the cloud environment.

PERSONAL NOTE

As a cloud enthusiast, I thoroughly enjoyed all the assignments offered in the course. It provided me with opportunities to explore various cloud services, delve into real-world scenarios like using CloudFormation, and follow industry best practices. The term assignment was truly top-notch and a fantastic learning experience.

Part XI.

REFERENCES

References

[1] Amazon Web Services, "AWS::S3::Bucket Properties," *AWS CloudFormation User Guide*, [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-bucket.html, [Accessed: August 01, 2023].

[2] Amazon Web Services, "AWS::EC2::Instance Properties," *AWS CloudFormation User Guide*, [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html, [Accessed: August 01, 2023].

[3] Amazon Web Services, "AWS::DynamoDB::Table Resource," *AWS CloudFormation User Guide*, [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-dynamodb-table.html, [Accessed: August 01, 2023].

[4] Amazon Web Services, "AWS::Lambda::Function Resource," *AWS CloudFormation User Guide,* [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html, [Accessed: August 01, 2023].

[5] Amazon Web Services, "AWS::StepFunctions::StateMachine Resource," *AWS CloudFormation User Guide*, [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-stepfunctions-statemachine.html, [Accessed: August 01, 2023].

[6] Amazon Web Services, "Amazon Rekognition Console," *AWS* [Online]. Available: https://us-east-2.console.aws.amazon.com/rekognition/home?region=us-east-2#/, [Accessed: August 01, 2023].

[7] Amazon Web Services, "Boto3 SNS Documentation," *AWS*, [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/sns.html, [Accessed: August 01, 2023].

[8] Amazon Web Services, "Boto3 Rekognition Documentation" , *AWS*, [Online]. Available: https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/rekognition.html, [Accessed: August 01, 2023].

[9] cURL Command in Linux with Examples: GeeksforGeeks, "cURL Command in Linux with Examples," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/curl-command-in-linux-with-examples/, [Accessed: August 01, 2023].

[10] Amazon S3 Share Object Pre-Signed URL: Amazon Web Services, "Share an Object with Others Using a Pre-Signed URL," *Amazon S3 User Guide*, [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/ShareObjectPreSignedURL.html, [Accessed: August 01, 2023].

[11] draw.io, "draw.io", *draw.io* [Online]. Available: https://draw.io/, [Accessed: August 01, 2023].

[12] Pallets Projects, "Flask Quickstart," *Flask Documentation*, Version 2.3.x, [Online]. Available: https://flask.palletsprojects.com/en/2.3.x/quickstart/, [Accessed: August 01, 2023].

[13] React, "Learn React," *React* [Online]. Available: https://react.dev/learn, [Accessed: August 01, 2023].

[14] Stack Overflow, "How to Enable CORS in Flask?,", *Stack Overflow* [Online]. Available: https://stackoverflow.com/questions/25594893/how-to-enable-cors-in-flask, [Accessed: August 01, 2023].

[15] Amazon Web Services, "Serverless Architectures with AWS Lambda," *AWS*, [Online]. Available: https://aws.amazon.com/lambda/serverless-architectures-learn-more/, [Accessed: August 01, 2023].

[16] Amazon Web Services, "What Is Infrastructure as a Service (IaaS)?," *AWS*, [Online]. Available: https://aws.amazon.com/what-is/iaas/, [Accessed: August 01, 2023].

[17] Intel Corporation, "Storage as a Service (SaaS)", *Intel Corporation* [Online]. Available: https://www.intel.com/content/www/us/en/cloud-computing/storage-as-a-service.html, [Accessed: August 01, 2023].

[18] Ernesto Marquez, "Use These Tools to Keep your AWS Lambda Cost Under Control," *Concurrency Labs*, [Online]. Available: https://www.concurrencylabs.com/blog/aws-lambda-cost-optimization-tools/, [Accessed: August 01, 2023].

[19] Stuart Burns, "Establish S3 bucket security with proper access control", [Online]. Available: https://www.techtarget.com/searchcloudcomputing/tip/Establish-S3-bucket-security-with-proper-access-control, [Accessed: August 01, 2023].

[20] Amazon Web Services, "Amazon Rekognition Pricing," *AWS* , Online: https://aws.amazon.com/rekognition/pricing/, [Accessed: August 01, 2023].

[21] MongoDB, "MongoDB Pricing," *MongoDB* , Online: https://www.mongodb.com/pricing, [Accessed: August 01, 2023].

[22] Amazon Web Services, "Amazon SNS Pricing," *AWS*, Online: https://aws.amazon.com/sns/pricing/, [Accessed: August 01, 2023].

[23] Cloudflare, "What is Hybrid Cloud?," *Cloudflare* , Online: https://www.cloudflare.com/en-in/learning/cloud/what-is-hybrid-cloud/, [Accessed: August 01, 2023].

[24] Microsoft Azure, "What is SaaS?," *Azure* , Online: https://azure.microsoft.com/en-ca/resources/cloud-computing-dictionary/what-is-saas/, [Accessed: August 01, 2023].

[25] Nirmal Ganesh Yarramaneni, "Dynamic Scalability Architecture," *LinkedIn*, Online: https://www.linkedin.com/pulse/dynamic-scalability-architecture-nirmal-ganesh-yarramaneni/, Accessed: August 01, 2023.

[26] Elastic Disk Provisioning, "Cloud Computing Design Patterns," *Arcitura*, Online: https://patterns.arcitura.com/cloud-computing-patterns/design_patterns/elastic_disk_provisioning, Accessed: [Accessed: August 01, 2023]

[27] Redundant Storage, "Cloud Computing Design Patterns," *Arcitura*, Online: https://patterns.arcitura.com/cloud-computing-patterns/design_patterns/redundant_storage, [Accessed: August 01, 2023]

[28] Amazon Web Services, "What is Service-Oriented Architecture?," *AWS* , Online: https://aws.amazon.com/what-is/service-oriented-architecture/, [Accessed: August 01, 2023]

[29] Amazon DynamoDB On-Demand Pricing: Amazon Web Services. "Amazon DynamoDB On-Demand Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/dynamodb/pricing/on-demand/, [Accessed: August 01, 2023].

[30] Amazon RDS MySQL Pricing: Amazon Web Services. "Amazon RDS MySQL Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/rds/mysql/pricing/?pg=pr&loc=2, [Accessed: August 01, 2023].

[31] Amazon S3 Pricing: Amazon Web Services. "Amazon S3 Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/s3/pricing/?nc=sn&loc=4, [Accessed: August 01, 2023].

[32] Amazon EFS Pricing: Amazon Web Services. "Amazon EFS Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/efs/pricing/, [Accessed: August 01, 2023].

[33] Amazon Rekognition Pricing. "Amazon Rekognition Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/rekognition/pricing/, [Accessed: August 01, 2023].

[34] AWS Lambda Pricing: Amazon Web Services. "AWS Lambda Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/lambda/pricing/, [Accessed: August 01, 2023].

[35] AWS Batch Pricing: Amazon Web Services. "AWS Batch Pricing," *AWS Pricing*, [Online]. Available: https://www.amazonaws.cn/en/batch/pricing/, [Accessed: August 01, 2023].

[36] Amazon SNS Pricing: Amazon Web Services. "Amazon SNS Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/sns/pricing/, [Accessed: August 01, 2023].

[37] Amazon SQS Pricing: Amazon Web Services. "Amazon SQS Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/sqs/pricing/, [Accessed: August 01, 2023].

[38] Amazon Web Services. "Amazon Simple Email Service (SES) Pricing," *AWS Pricing*, [Online]. Available: https://aws.amazon.com/ses/pricing/, [Accessed: August 01, 2023].

[39] StackShare. "Apache OpenWhisk vs. OpenFaaS - Comparison," *StackShare*, [Online]. Available: https://stackshare.io/stackups/apache-openwhisk-vs-openfaas, [Accessed: August 01, 2023].

[40] Jelvix. "MySQL vs PostgreSQL vs SQL Server: Comparison," Jelvix Blog, [Online]. Available: https://jelvix.com/blog/mysql-postgresql-sql-server, [Accessed: August 01, 2023].