COS20019 – Assignment 3

Serverless/Event-driven Architectural Design Report

Date of Submission: 28/07/2024

Bui Minh Thuan
Student ID: 104486358
Tutorial class: Wednesday 8:00AM

Vu Minh An
Student ID: 104993133
Tutorial class: Wednesday 8:00 AM

Nguyen Trung Thinh
Student ID: 104777544
Tutorial class: Wednesday 8:00 AM

Trinh Nhan Kiet
Student ID: 104988281
Tutorial class: Wednesday 8:00 AM

I. Introduction

The Photo Album application, developed for seamless media management and sharing, has achieved remarkable success and the business requires advanced architecture to sustain its growth. This report presents a comprehensive cloud architecture design using Amazon Web Services (AWS) to ensure high performance, security, and global accessibility. The solution adopts a serverless, event-driven approach to minimize in-house administration and optimize resources. Key AWS services are selected to enhance content delivery, authentication, media processing, and data storage. Selection is based on design rationale, offering justification and comparison for the chosen AWS services, addressing performance, reliability, security, and cost considerations. This cloud architecture addresses current needs while providing flexibility and scalability for future expansion and feature enhancements

II. Architecture Design

A. *Architectural Diagram*

The following architectural diagram illustrates the design and implementation details of our system, showcasing a serverless architecture, tiered structure and integration of various AWS services. We design this diagram based on given business requirements, which will be discussed in later sections.

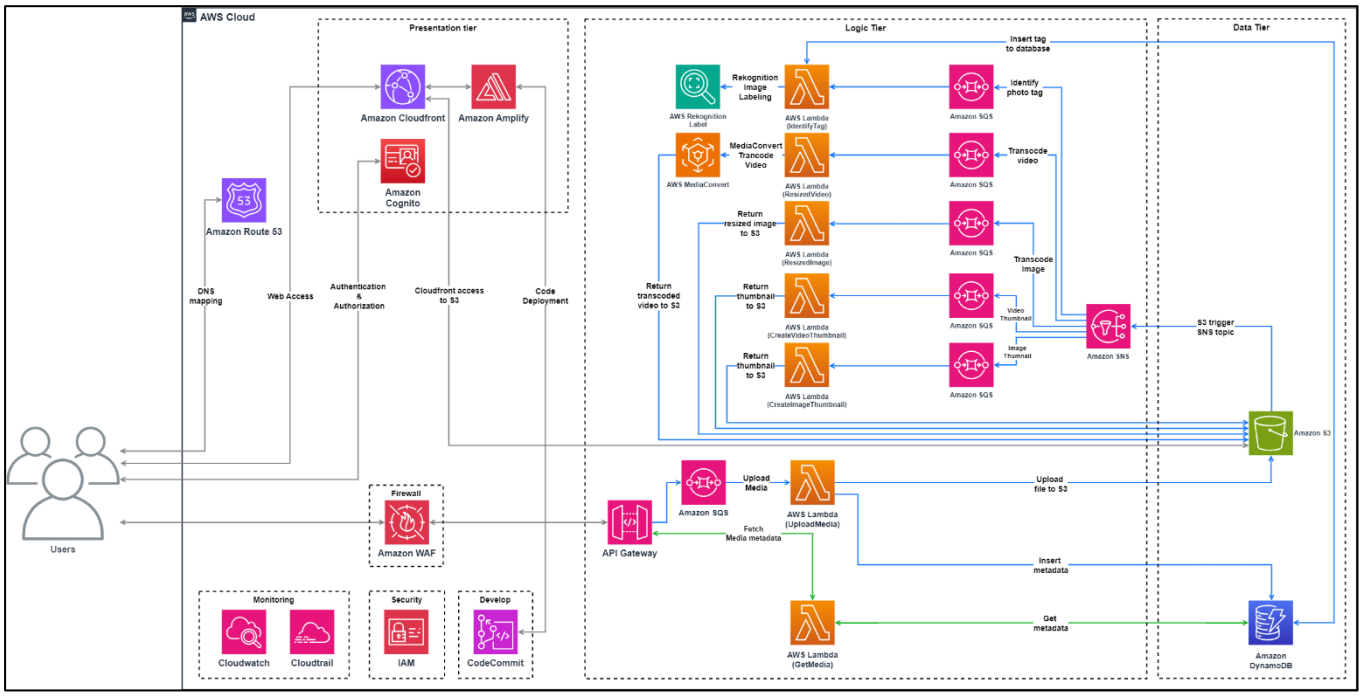

Figure 1: Infrastructure diagram

B. *Requirements Fullfilment*

Our architecture leverages a combination of AWS services to ensure that the photo album website meets all business and assignment requirements, providing a scalable, secure, and efficient solution for user image and video management.

*1) **Utilising AWS managed service***

Our architecture is built fully from AWS services, with no third-party platform or in-house systems administration required. This includes serverless components such as AWS Lambda for executing code, Amazon S3 for storage, and Amazon SQS for message queuing. By using these managed services, our architecture reduces the overhead needed to run our service. Additionally, AWS services offer built-in security features and compliance certifications, ensuring that our data is protected, and regulations are met. The managed nature of these services also means we benefit from regular updates and improvements without any effort on our part.

*2) **Cope with business growth***

Our serverless design, supported by AWS Lambda and DynamoDB, ensures that the system scales automatically to cope with growing business demands. AWS Lambda functions scale up or down based on the number of events for processing, while DynamoDB provides automatic horizontal scaling to handle increased loads and storage. This approach effectively supports business growth without requiring manual intervention for scaling. Furthermore, this dynamic scaling capability helps maintain optimal performance and cost efficiency during peak times and quieter periods. The architecture's resilience and ability to handle sudden traffic spikes ensure it can manage the website's growth, which is expected to double every year.

*3) **Adopting a serverless & cost-effective solution***

The company's system, running on t2.micro EC2 instances with over 80% capacity usage, is slow and costly. Therefore, we design a serverless architecture to help the company reduce costs, improve performance and deliver superior user experience by addressing latency issues and ensuring efficient scaling.

Serverless services offer a pay-as-you-go model [1], so the company only pays for resources when they are used, with no costs for idle run time, this helps optimize expenses. Serverless solutions can automatically scale to handle sudden traffic spikes without manual configuration. Services like AWS Lambda can scale from 0 to thousands of requests per second [2]. Serverless functions also greatly reduce the latency experienced by end users because serverless functions don't operate from an original server, so there's no single location to which an end user's traffic must be directed.

*4) **Cost-effective database solution***

We use DynamoDB's on-demand capacity mode for its cost-effectiveness, as it eliminates the need for manual capacity planning and charges based on actual read and write requests, aligning with a pay-as-you-go model. reduces cost by using non-relational DynamoDB instead of RDS for storage that rely on expensive physical instances DynamoDB is also incredibly effective with its fast performance, delivering single-digit millisecond response times for millions of requests per second, ensures rapid data access and high throughput. This ensures that our application remains responsive and can handle large volumes of transactions effortlessly. The integration with other AWS services like Lambda further optimizes data processing and reduces latency.

*5) **Global response***

The architecture incorporates Amazon CloudFront to deliver content globally from AWS Amplify. CloudFront caches content at edge locations around the world, significantly reducing latency and ensuring a fast, seamless user experience regardless of geographic location. Frequently accessed data such as images and videos are cached to reduce unnecessary strain on the system which reduces costs while maintaining content delivery. This global reach is essential for providing a consistent experience to users, no matter where they are accessing the application from.

*6) **Being able to handle video media and different format***

Our current architecture is designed to handle photo media efficiently and can be extended to manage video media in the future. The scalability of AWS Lambda, S3, and non-relational DynamoDB, along with the use of CloudFront for global content delivery, provides a robust foundation for accommodating additional media types. Unlike images, video media files are handled by AWS Media Converter, an AWS service that provides great support for handling media files with high quality. Our serverless design incorporates SQS queueing for decoupling to ensure that the system still performs well without being overloaded with intensive processing and memory stress. Since MediaConvert supports a wide variety of video input and output formats, such as broadcast and internet delivery formats, our architecture can adapt to handle a variety of media formats in the future. The video processing process happened exactly as expected in the business requirement, and details about it will be discussed in later sections.

C. Service Description

Below is a detailed list of the services used in our system, outlining their functionalities and contributions to enhancing performance, scalability, and cost-efficiency.

*1) **AWS Route53***

Amazon Route 53 is a scalable Domain Name System (DNS) web service that translates domain names into IP addresses and directs end-user requests to various AWS services or on-premises servers. It offers advanced DNS features such as health checks and traffic routing policies like latency-based routing, geo DNS, and weighted round-robin. By directing users to the nearest data center, Route 53 ensures applications maintain high availability and low latency while providing robust domain name management with enhanced security through DNSSEC. We use AWS Route 53 for ensuring efficient and reliable routing of user requests to our applications, optimizing performance and availability across global regions.

*2) **AWS Cognito***

Amazon Cognito manages user authentication and access control, offering sign-up and sign-in capabilities across various identity providers, including social (Facebook, Google), enterprise (SAML, LDAP), and custom directories. It provides user pools for authentication and identity pools for authorization, simplifying user management while supporting multi-factor authentication and data encryption for enhanced security. We use AWS Cognito for managing user identities and authentication processes in our applications, ensuring secure access and a streamlined user experience.

*3) **AWS CloudFront***

AWS CloudFront is a content delivery network (CDN) service that caches and delivers content from edge locations worldwide. It efficiently manages static and dynamic content with features like versioning and TTL (time to live) and supports video delivery with encoding capabilities. By caching content closer to users, CloudFront reduces latency and enhances global content delivery, while providing protection against DDoS attacks and integrating with AWS WAF for traffic management. We use AWS CloudFront for accelerating content delivery and improving the performance of our web and video applications by leveraging its global CDN infrastructure.

*4) **AWS WAF***

AWS Web Application Firewall (WAF) defends web applications from common threats and vulnerabilities, including SQL injection and cross-site scripting (XSS). It allows the creation of custom security rules to manage access based on IP addresses, HTTP headers, HTTP body, or URI strings. Integrating seamlessly with services like CloudFront, Application Load Balancer, and API Gateway, AWS WAF provides scalable and customizable protection for web applications. We use AWS WAF for safeguarding our web applications from malicious traffic and vulnerabilities, enhancing overall security and reliability.

*5) **AWS Amplify***

AWS Amplify offers a suite of tools and services designed to simplify the development of full-stack web and mobile applications. It provides a framework, hosting, and libraries for building and integrating modern apps with AWS services like REST APIs, storage, and authentication. Amplify accelerates development by reducing backend integration complexity, allowing developers to focus on delivering an excellent user experience. We use AWS Amplify for rapidly building, deploying, and managing scalable web and mobile applications, improving development efficiency and user experience.

*6) **AWS CodeCommit***

AWS CodeCommit is a managed source control service that supports secure Git-based repositories. It facilitates team collaboration on code, file management, and change tracking in a scalable and reliable environment. By integrating with other AWS services, CodeCommit streamlines the development workflow and enhances collaboration without requiring self-managed version control systems. We use AWS CodeCommit for managing and versioning our source code in a secure and scalable environment, facilitating smooth collaboration among development teams.

*7) **AWS API Gateway***

Amazon API Gateway enables developers to build, publish, maintain, and secure APIs at scale. It supports both RESTful and WebSocket APIs, simplifying the creation and management of API endpoints. API Gateway provides features for monitoring, throttling, and securing APIs, and its pay-as-you-go pricing model offers cost efficiency for API management. We use AWS API Gateway for creating and managing APIs that connect our applications to backend services, enabling scalable and secure communication.

*8) **AWS Lambda***

AWS Lambda offers serverless computing, allowing code execution in response to events without managing servers. It automatically scales applications based on triggers, supports multiple programming languages, and reduces operationaloverhead by handling infrastructure, scaling, and patching. This pay-as-you-go model provides flexibility and cost-effectiveness for running code functions. We use AWS Lambda for executing code in response to events without managing servers, enhancing scalability and reducing operational complexity.

*9) **AWS SNS***

Amazon Simple Notification Service (SNS) provides a managed messaging platform for decoupling microservices and serverless applications. It supports delivering messages to multiple endpoints and clients using various protocols like HTTP/S, email, and SMS. SNS facilitates asynchronous communication, improving system resilience and scalability. We use AWS SNS for enabling asynchronous messaging and notifications between application components, improving system resilience and communication efficiency.

*10) **AWS SQS***

Amazon Simple Queue Service (SQS) is a fully managed message queuing service that supports Standard and FIFO (First-In-First-Out) queues. It enables reliable communication between distributed components by decoupling and scaling microservices and serverless applications. SQS enhances fault tolerance and simplifies message processing without complex infrastructure management. We use AWS SQS for managing and processing messages between distributed components of our application, enhancing fault tolerance and ensuring our architecture is decoupled.

*11) **AWS MediaConvert***

AWS Elemental MediaConvert is a file-based video transcoding service that prepares media files for playback across a range of devices. It supports numerous video and audio codecs, adaptive bitrate streaming, and advanced features like captions and DRM. MediaConvert optimizes media files for diverse devices and network conditions, ensuring high-quality delivery and efficient media workflows. We use AWS MediaConvert for transcoding and optimizing video content for diverse devices and network conditions, ensuring high-quality media delivery.

*12) **AWS Rekognition***

AWS Rekognition provides advanced image and video analysis using deep learning, identifying and labeling objects, scenes, and activities. It automates the tagging process, which improves content organization, search accuracy, and reduces manual effort. Rekognition includes features like facial recognition and object detection, making it versatile for various applications. We use AWS Rekognition for automating the analysis and tagging of visual content, improving organization, searchability, and application functionality.

*13) **Amazon S3***

Amazon S3 offers scalable object storage for a wide range of data types with high durability and availability. It includes features like versioning, lifecycle policies, and cross-region replication to manage and protect data effectively. Integrating with other AWS services, S3 provides a cost-effective solution for storing and accessing large volumes of data. We use Amazon S3 for securely storing and managing data at scale, leveraging its durability and cost-effectiveness to support various application needs.

*14) **AWS DynamoDB***

Amazon DynamoDB is a fully managed NoSQL database service that ensures fast and consistent performance with seamless scalability. It supports key-value and document data structures, offering high availability, durability, and built-in security features like encryption at rest. Its serverless nature eliminates the need for manual database management and scales automatically to accommodate varying data and traffic volumes. We use AWS DynamoDB for handling data processing and storage needs, ensuring fast performance and scalability for our applications.

*15) **Others: AWS IAM, AWS CloudWatch, AWS CloudTrail***

AWS IAM provides a secure way to manage access to AWS services and resources. It enables users to create and control AWS users, groups, and roles, and to define permissions that dictate what actions they can perform. IAM supports fine-grained access control and integrates with other AWS services to enforce security policies and ensure that only authorized entities can access or modify resources. AWS IAM enhances security and compliance by offering precise control over user access and permissions. We use AWS IAM for managing user permissions and access controls in our photo album website, ensuring secure and authorized operations for users uploading and viewing images and videos.

AWS CloudWatch is a monitoring and observability service that provides real-time insights into the performance and operational health of AWS resources and applications. It collects and tracks metrics, logs, and events, enabling users to set up alarms, visualize data through dashboards, and take automated actions based on defined thresholds. AWS CloudWatch ensures that systems are monitored effectively by providing detailed visibility into performance metrics and operational health. Its ability to create alarms and automate responses enhances proactive management, reduces downtime, and supports efficient troubleshooting. We use AWS CloudWatch for monitoring the performance and health of our photo album website, allowing us to set up alerts and take proactive actions to maintain system stability and performance.

AWS CloudTrail enables comprehensive governance, compliance, and operational auditing of AWS account activity. It records and logs API calls and events made by or on behalf of AWS accounts, providing detailed insights into user actions, resource changes, and security-related activities. AWS CloudTrail strengthens security and compliance by offering a thorough audit trail of AWS account activity. We use AWS CloudTrail for auditing and tracking user activities and changes on our photo album website, ensuring compliance and enhancing security by providing detailed logs of all interactions with the system.

D. UML Sequence Diagrams

The following UML sequence diagrams illustrate the interactions and flow of services within our AWS architecture, demonstrating how each component collaborates to achieve seamless operation and efficient resource management.

*1) **Authentication***

When a user logs into the application, they enter their username and password, which the application uses to perform Secure Remote Password (SRP) calculations. The application then sends an InitiateAuth request to AWS Cognito (Users pool) with USER_SRP_AUTH details [3]. AWS Cognito responds with a challenge, often a PASSWORD_VERIFIER challenge. The application prompts the user for any additional required information, such as an MFA code, and then sends a RespondToAuthChallenge request to AWS Cognito with the necessary SRP parameters and challenge response. If the response is correct, AWS Cognito issues access, ID, and refresh tokens, which the application stores securely. This allows the user to access the application content, ensuring secure authentication without transmitting the password over the network.

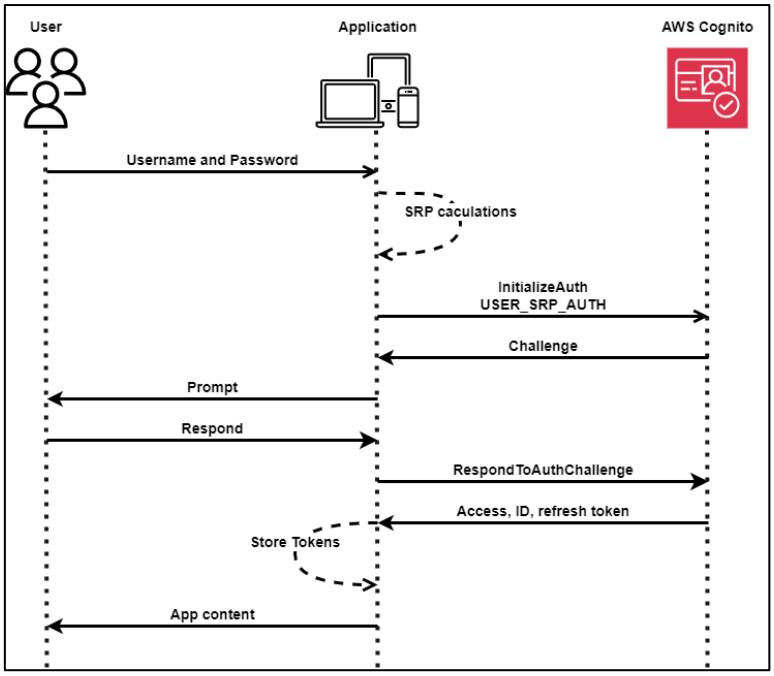

Figure 2: Authentication flow

*2) **Web accessing***

The sequence diagram outlines the process of delivering web content using AWS services [4]. When a user requests a website URL, AWS Route 53 resolves the request and returns the CloudFront IP address [5]. The user's HTTP request is then directed to CloudFront, which checks its cache for the content. If the content is already cached in an edge location, it is served directly to the user. If the content is not cached or the cache has expired, CloudFront fetches the latest content from AWS Amplify [7]. Amplify retrieves the source code from AWS CodeCommit, builds, and deploys the website [8]. The content is then returned to CloudFront, which stores it in the edge location before serving it to the user.

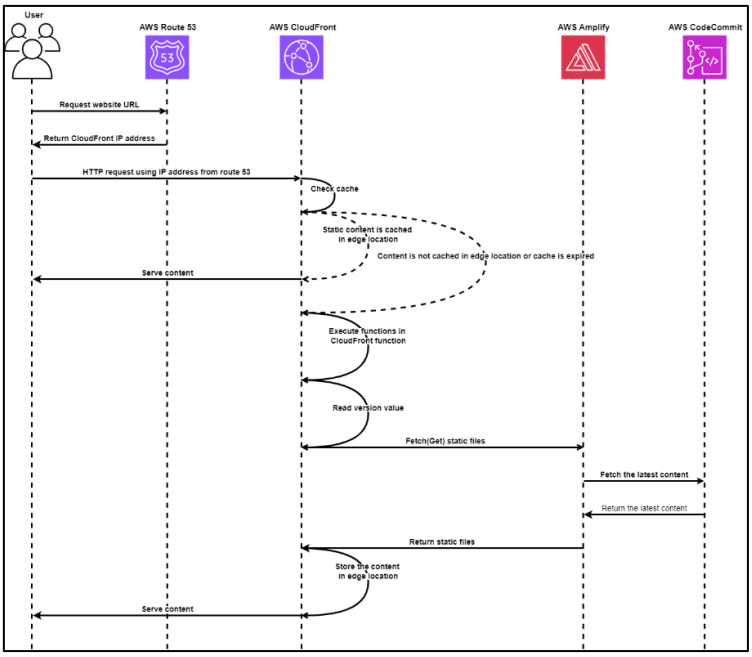

Figure 3: Web accessing flow

3) Display media

Whenever user request for displaying their album, an API will be called to API gateway, which then triggers a Lambda function called “GetMedia”. This function will query into database and return metadata as well as CloudFront link of files in this user’s album. While metadata will be displayed as normal when browser receive reply, the actual file content will be fetched again with the newly retrieved links. For each link, the website first checks whether each file is cached in nearest edge location or not. If yes, directly serve this file to browser, but if no, redirect to actual S3 bucket, get the file and send it to browser. Remember in the second case, the retrieved file will then be cached at an edge location so other users can benefit from it.

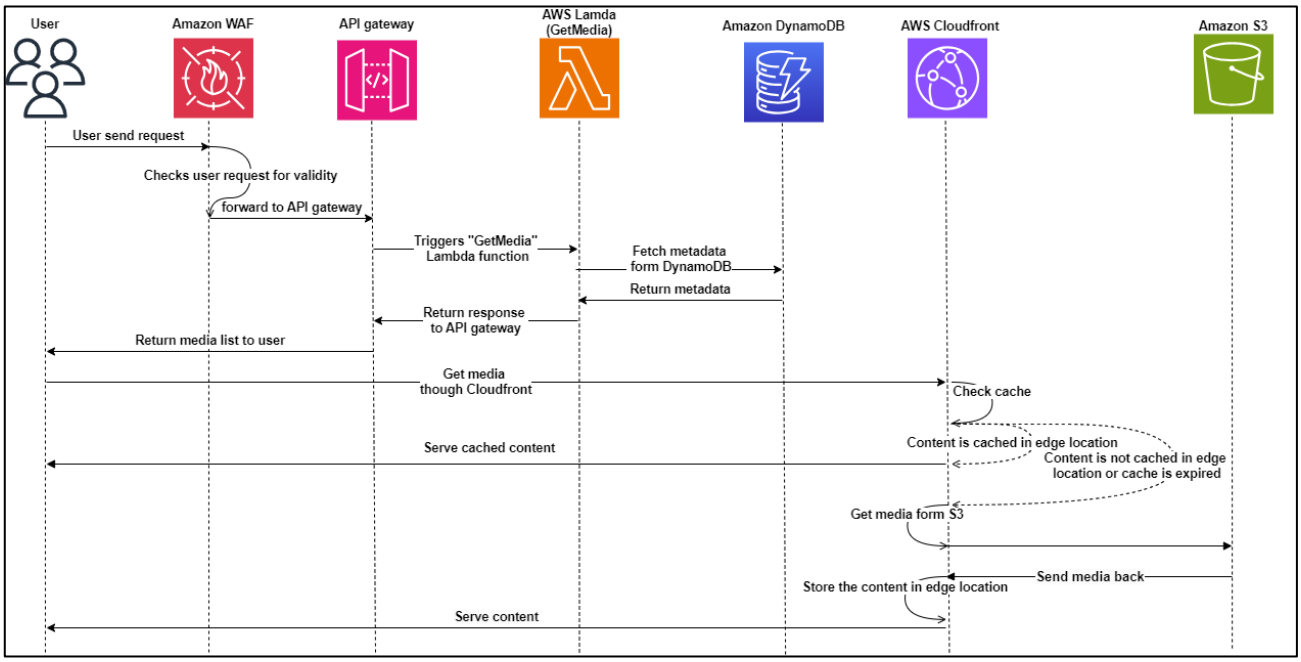

Figure 4: Displaying media album flow

4) Uploading media

For uploading, there are two types of uploading processes: image uploading and video uploading. Let’s start will the image uploading. Similarly to other backend functions, first the upload request is filtered and managed by AWS WAF and API Gateway. This gateway triggers a Lambda function named “UploadMedia”, which uploads this file directly to S3 bucket. The metadata of this image is also inserted into the database, and when S3 receives a new file, it triggers the topic of an AWS SNS, which will forward new requests into different SQS, before it calls different Lambda for image processing. In our architecture, a thumbnail and resized version of the uploaded image will be stored in S3, while tags detected by ReKognition is stored in the record of this image in the database.

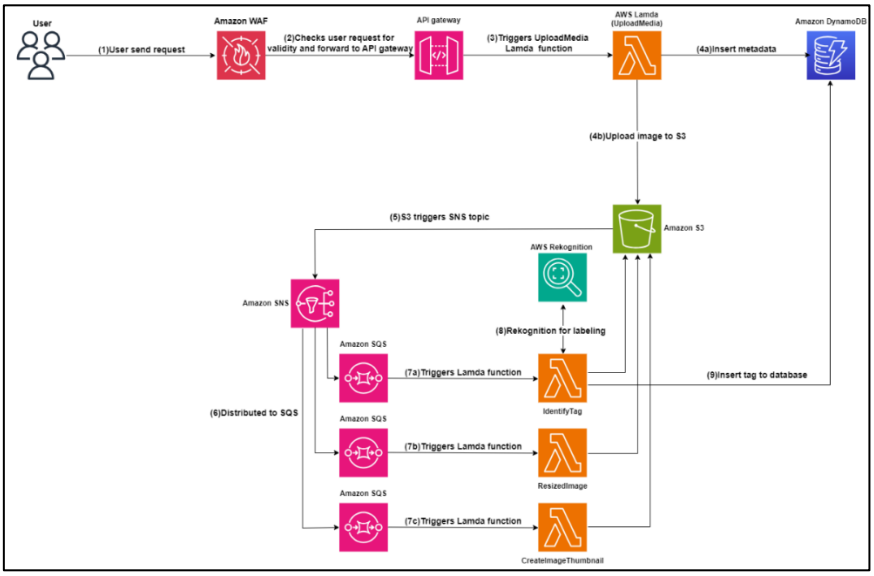

Figure 5: Uploading image

For uploading video, the concept is almost similar. The only difference is that, in the media processing phase, our architecture will create a transcoded version of this video using AWS MediaConverter. This newly created video will also be stored in S3.

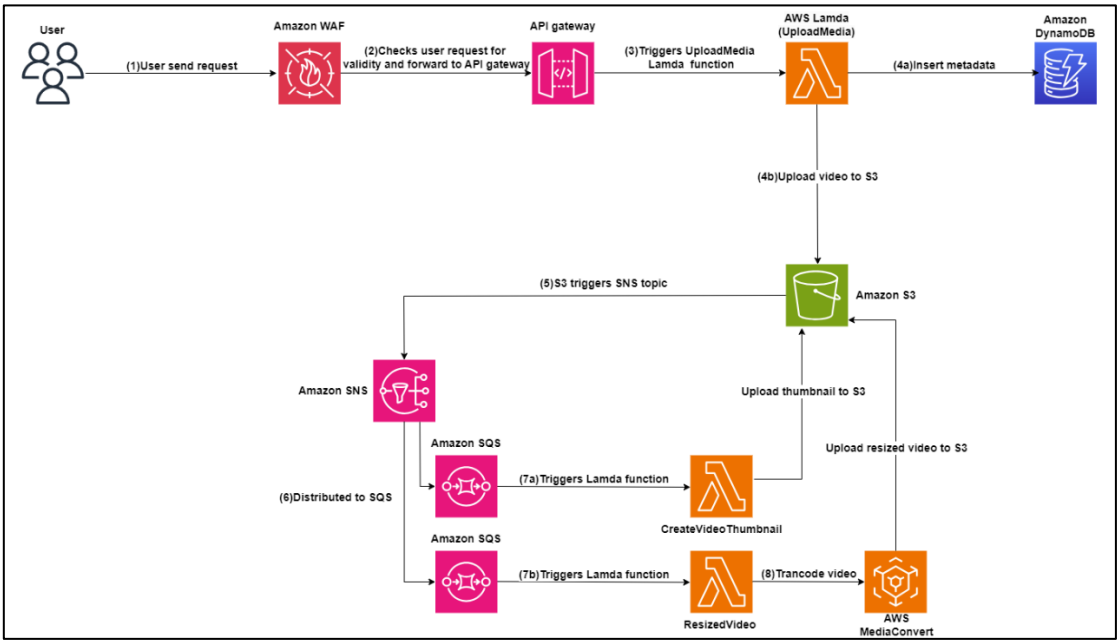

Figure 6: Uploading video

III. Design Rationale

After setting up the AWS architecture, the second part of the assignment focuses on deploying the website in this architecture. In this assignment, the website is a photo album, in which images are stored on AWS, and will be displayed whenever user access the web page. Most of the code for the website is already provided in the instruction folder, so my main tasks will be modifying the code to match the architecture, deploying it to the web server, and testing it functionality.

A. Design Justification

The following section details the rationale behind our design choices, framed within the five pillars of the AWS Well-Architected Framework: operational excellence, security, reliability, performance efficiency, and cost optimization.

*1) **Operational Excellence***

We define standards in our solution to successfully meet the daily operational needs of the PhotoAlbum website according to business requirements, ensuring a secure, scalable, high-performance, and cost-effective cloud architecture. Our serverless architecture is designed with a clear workflow for each microservice, divided into three tiers—presentation, application, and data—making management and website function implementation convenient.

New feature development and code deployment are efficiently managed using AWS CodeCommit. CodeCommit is our fully managed source control service that hosts our Git repositories. By using CodeCommit, we enable PhotoAlbum developers to collaborate, commit code changes, and manage branches. This facilitates continuous integration and delivery, ensuring the website can rapidly evolve according to business requirements.

To maintain operational health and performance, we utilize Amazon CloudWatch and AWS CloudTrail. CloudWatch provides real-time monitoring of AWS resources and applications, collecting and tracking metrics, logging files, and setting alarms. With CloudTrail, we can track system changes and analyze their effects on system health and performance. Amazon Route 53 further contributes to our operational excellence by performing health checks of web endpoints. Our approach enables us to respond quickly to events and adapt to the evolving business needs of the PhotoAlbum website

*2) **Security***

We have implemented a robust identity and access management system to ensure the security of user data. To secure user authentication sessions, we use Amazon Cognito, which provides access tokens for authenticated users who match the designated user pool. These users are then granted temporary credentials associated with IAM roles created by Cognito, which restricts their access to only the necessary PhotoAlbum website services to manage permissions. Sensitive data such as tokens and generated temporary symmetric keys are encrypted with AWS KMS in transit and at rest, ensuring comprehensive data protection. Asymmetric keys are used to encrypt the symmetric keys generated from Cognito and to verify digital signatures and data integrity.

Our multi-layered security infrastructure includes Amazon WAF (Web Application Firewall), which has managed rules by AWS experts to automatically detect and mitigate SQL injection and XSS attacks. We also use WAF to filter web traffic such as HTTP headers and mitigate DDOS attacks by setting thresholds for requests from individual IP addresses. We monitor our architecture security using CloudWatch and CloudTrail. AWS CloudTrail captures and logs all API activity within our environment, providing detailed visibility into user actions and service interactions for auditing and threat detection. Meanwhile, AWS CloudWatch monitors system performance and generates alarms based on specific log patterns and metrics, enabling us to detect and respond to security events in real time.

3) ***Reliability***

We focus on ensuring the reliability, continuous operation, and robustness of the PhotoAlbum website, even under adverse conditions. To achieve this, we have implemented several AWS services designed to handle failures automatically and recover reliably: AWS Lambda, S3, and SQS. These serverless services are inherently resilient, managed by AWS to recover from failures without requiring manual intervention. For example, Lambda automatically retries failed executions, S3 ensures data durability with multiple copies across different locations, and SQS manages message durability and availability. DynamoDB plays a critical role in our reliability strategy by replicating data across multiple regions working alongside our serverless architecture's automatic scaling.

Additionally, DynamoDB's horizontal scaling capability partitions data to distribute the load, which increases system availability by adding more resources as needed, rather than upgrading existing ones. These resources are cached using CloudFront at edge locations globally, reducing the load on the origin servers, and ensuring content availability even during high-traffic periods. This cross-region replication enhances data availability and fault tolerance, ensuring that data remains accessible even if a region experiences an outage. To handle web errors, we utilize Route 53 DNS failover routing's automatic redirection of traffic to alternate locations in case of errors, ensuring high availability and reliability.
We avoid the challenges of capacity planning by using CloudWatch for comprehensive monitoring and insights into our system's performance to help us make informed decisions and manually adjust resources as needed. Our automated deployment and scaling strategy further reduces human error and enhances reliability. This automation ensures that the website remains functional and responsive, even in the event of crashes or other disruptions.

4) ***Performance Efficiency***

For performance efficiency, we have implemented serverless technologies, leveraging AWS Amplify to host our website with asynchronous Lambda processing. This approach eliminates the overhead of server management and ensures that our resources are dynamically allocated based on real-time demand. This aligns with the principle of democratizing advanced technologies, making sophisticated, scalable solutions accessible and easy to implement using AWS-managed services. We selected DynamoDB, known for its exceptional performance and scalability, efficiently handles large volumes of data with low latency while seamlessly scaling horizontally to manage increased traffic. By utilizing DynamoDB's on-demand capacity mode, we align with the principles of mechanical sympathy, avoiding the complexities of manual capacity planning. This approach ensures our database operations are inherently efficient and adapt dynamically to workload demands, enhancing overall performance,

Our website's main performance comes from executing Lambda function code in response to various triggers, such as changes in data or system states, and user actions. In our architecture, we incorporate a pub/sub mechanism with Amazon SNS to publish messages and notify AWS SQS, which then triggers Lambda functions to process tasks asynchronously. This setup allows us to handle numerous user requests and uploads concurrently, maintaining high performance and ensuring the system remains responsive under heavy load. This design also supports frequent experimentation by allowing quick implementation and function testing.

Our global reach is enhanced through CloudFront, which allows our website to go global in minutes alongside our serverless automatic scaling and global content delivery. By caching content close to users, CloudFront offers high performance, and significantly reduces latency, providing a fast and seamless user experience regardless of their geographic location. We used CloudWatch to monitor real-time metrics and insights regarding system performance, enabling us to maintain perform efficiency of the application architecture

5) ***Cost Optimization***

We reduce costs through our serverless architecture by leveraging the pay-as-you-go approach [10], ensuring we only pay for the computing time used rather than maintaining expensive, idle physical instances. By choosing the on-demand capacity mode for DynamoDB, we eliminate the need for capacity guessing, reducing costs by charging only for actual read and write requests. Additionally, our managed services such as AWS SNS, SQS, and DynamoDB offload the responsibility of maintaining and updating infrastructure to AWS, lowering the need for IT experts and increasing our ROI. To manage cost-effectiveness, we use AWS Cost Explorer, AWS Pricing Calculator and CloudWatch to monitor monthly costs and application statistics, ensuring optimal resource utilization and continuous cost monitoring [11].

B. Alternative Solutions

In this section, we will compare our chosen services with alternative solutions, highlighting their respective advantages and disadvantages to provide a comprehensive analysis of potential options.

1) ***Caching Options: Cloudfront vs ElastiCache***

Providing users with low-latency content delivery is crucial for the Photo Album application due to its requirement for quick image and video access. The two popular caching options for AWS are Amazon CloudFront and Amazon ElastiCache. Understanding the use cases and differences between these services is essential for optimizing the application's performance and user experience.

Amazon ElastiCache is a managed service that provides in-memory data to cache database queries and session data [16], reducing the load on the primary database by storing frequently accessed data in memory. ElastiCache is suited for real-time fetching of dynamic data from databases rather than static files. However, for our use case, the primary caching needed is for static content delivery (i.e., images, videos) rather than the queries.

On the other hand, AWS CloudFront caches image and video files at edge locations worldwide [17], which is essential for our application's requirement for global accessibility and low latency. It also integrates seamlessly with other AWS services, enhancing scalability, reliability, and security. Therefore, CloudFront is the optimal choice for our use case of caching static media files

2) ***Database: SQL vs NoSQL***

Efficient database management is crucial for the Photo Album application due to its requirement for quick media access. Two popular database options for AWS are Amazon RDS for SQL databases and Amazon DynamoDB for NoSQL databases. Understanding the use cases and differences between these services is essential for optimizing the application's performance and user experience.

The company currently uses a relational database with Amazon RDS which is optimized for structured data and complex queries rather than unstructured data [18]. However, the current setup has proven to be costly and slow for our use case of storing and retrieving large media files (images, videos). Although SQL databases maintain data integrity and complex relationships, they lack NoSQL's ability to store and handle the disparate media data required by the website [19].

Amazon DynamoDB, on the other hand, provides near-real-time response time and predictable performance with serverless scalability [20]. It supports key-value and document data models, making it ideal for applications requiring quick read/write access to large volumes of data. Furthermore, DynamoDB is a NoSQL database, which means it's more flexible and cheaper to scale horizontally using distributed clusters compared to RDS SQL vertical scaling of expensive servers [21]. For our use case, DynamoDB's ability to handle rapid access to stored media and scale to meet increasing demands makes it the optimal choice.

3) ***Web hosting: Amplify vs Amazon S3***

Both Amazon S3 and AWS Amplify are used for web hosting. While S3 provides a simpler solution for hosting static websites [22], Amplify offers a more comprehensive and suitable platform for complex and scalable web applications.

Amazon S3 (Simple Storage Service) does have the capability to host static websites, making it a reliable option for serving HTML, CSS, and JavaScript files [23]. However, hosting websites is not its primary function, making it harder to extend and integrate with other AWS resources compared to Amplify.

On the other hand, AWS Amplify provides a comprehensive set of tools and services for building, deploying, and scaling full-stack web and mobile applications. Amplify's managed services minimize the need for in-house systems administration, making it perfect for businesses seeking to streamline operations and focus on development. Unlike S3, Amplify offers features like seamless integration with other AWS services, serverless computing with AWS Lambda, and robust authentication and analytics. This makes Amplify particularly suitable for your rapidly growing Photo Album application, which requires a scalable, serverless, and event-driven architecture to support current and future needs, making it a superior choice over S3 for hosting static websites

4) ***Media transcoding: Elastic Transcoder vs AWS MediaConverter***

AWS Elemental MediaConvert and Amazon Elastic Transcoder are both AWS services designed for transcoding media files stored in S3 into formats suitable for various playback devices. Comparing these two is essential for determining which service best meets specific business requirements, particularly for applications with increasing demand and diverse media needs.

Amazon Elastic Transcoder is a straightforward, cloud-based media transcoding service [24]. It offers a simple setup with presets for common formats, allowing easy conversion of media files into multiple formats within a single job. While it is practical for basic transcoding needs, it lacks advanced control over encoding settings and supports fewer input and output formats, which can be limiting for more complex requirements.

AWS Elemental MediaConvert, however, is more advanced and flexible service [25]. It supports a broader range of formats, provides finer control over video quality and encoding settings, and includes features like adaptive bitrate streaming and built-in content encryption [26]. For the Photo Album application, which anticipates exponential growth and diverse media formats, MediaConvert is more suitable. Its scalability, advanced features, and automatic media processing upon S3 upload make it ideal for handling increased demand, ensuring high-quality delivery, and accommodating future needs like video transcoding and additional processing capabilities.

*5) **Decoupling architecture: AmazonSNS vs AWS EventBridge***

When designing serverless applications, it's essential to consider why implementing either the pub-sub or queue pattern is more beneficial than calling Lambda functions directly. Direct invocation can lead to tight coupling between services, introducing several drawbacks such as increased dependencies, reduced modularity, and harder maintenance.

Additionally, relying on a single endpoint can create bottlenecks, leading to performance issues and potential failures under high load. Messaging patterns like queue and pub-sub effectively decouple services and distribute the load more efficiently, enhancing the system's reliability and scalability [27]. In our architecture, the queue pattern allows messages to be sent to a queue and processed asynchronously by consumer services, decoupling the producer (e.g., API Gateway) from the consumer (e.g., Lambda function), and buffering messages to smooth out traffic spikes. This prevents consumers from being overwhelmed and is particularly useful between API Gateway and Lambda functions. Conversely, the pub-sub pattern broadcasts messages to multiple subscribers simultaneously, enabling real-time updates and decoupled communication between producers and consumers. This is ideal for scenarios where multiple services need to react to events independently, such as in the transcoding section, where multiple services can process the same event concurrently and flexibly.

While both Amazon SNS (Simple Notification Service) and Amazon EventBridge can trigger Lambda functions, SNS is often the more practical choice for S3 and pub-sub architectures due to its simplicity and performance. EventBridge offers advanced routing and extensive AWS integrations [28], but for straightforward, many-to-many communication, SNS excels. Its ability to deliver messages to multiple subscribers [29], including Lambda functions, ensures prompt reaction to new S3 objects, enhancing system responsiveness and decoupling. SNS supports various communication protocols, making it versatile and efficient in distributing messages to multiple endpoints simultaneously [30]. This makes SNS an ideal solution for ensuring immediate and reliable message delivery in pub-sub architectures, facilitating effective communication and timely processing tasks.

C. Quantitive compare

Below is the cost estimation of based on our assumption of website usage:

Service	Usage	Pricing	Monthly cost
AWS Route 53	- Number of hosted zones: 1 - IP (CIDR) blocks: 3	- Tiered price for: 1 - 1 x 0.50 USD = 0.50 USD - Total tier cost = 0.50 USD (Hosted Zone cost) - Total Hosted Zones & RRset records cost: 0.50 USD - Tiered price for: 3 IP (CIDR) blocks - 3 IP (CIDR) blocks x 0.00 USD = 0.00 USD - Total tier cost = 0.00 USD (IP (CIDR) blocks cost) **=> Route53 Hosted Zone cost (monthly): 0.50 USD**	$0.5
AWS CloudFront	- 300,000 requests per month - 90 GB Data Transfer	- Tiered price for: 90 GB - 90 GB x 0.114 USD = 10.26 USD - Total tier cost = 10.26 USD (Data transfer out to internet from Australia) **- Data transfer out to internet cost: 10.26 USD** **- Data transfer out to origin cost: 0 USD** - 300,000 requests x 0.00000125 USD = 0.38 USD (HTTPS requests from Australia) **- Requests cost: 0.38 USD** - 10.26 USD + 0.38 USD = 10.64 USD (Total cost Australia) **=> CloudFront price Australia (monthly): 10.64 USD**	$10.64

AWS Cognito	1000 Monthly Active Users (MAUs)	- 1,000 MAUs x 0.10 SAML or OIDC federation requests = 100.00 SAML or OIDC federation MAU requests - 100.00 SAML or OIDC federation MAUs - 50 free SAML or OIDC federation MAU requests per month = 50.00 billable SAML or OIDC federation MAU requests - Max (50.00 billable SAML or OIDC federation MAU requests, 0 minimum billable SAML or OIDC federation MAU requests) = 50 total billable SAML or OIDC federation MAU requests - 50 MAUs x 0.015 USD = 0.75 USD (SAML or OIDC federation MAU requests) **=> SAML or OIDC federation cost (monthly): 0.75 USD** - 1,000 MAUs - 50000 free MAU requests per month = -49,000.00 billable MAU requests - Max (-49000.000000 billable MAU requests, 0 Constant Unit) = 0.00 total billable MAU requests - Tiered price for: 0.00 MAUs - Total tier cost = 0.00 USD (User Pool MAUs) **=> User Pool MAU cost (monthly): 0.00 USD** - 1,000 MAUs x 1 Advanced security features option enabled = 1,000.00 Advanced security feature MAUs - Tiered price for: 1,000.00 MAUs - 1,000 MAUs x 0.05 USD = 50.00 USD - Total tier cost = 50.00 USD (ASF MAUs) **=> Advanced security feature cost (monthly): 50 USD** - 50 USD + 0.75 USD = 50.75 USD **=> Cognito MAU cost (monthly): 50.75 USD**	$50.75
AWS Amplify	- 10 build minutes per month - 15 GB data transfer per month - 30,000 requests per month	- 0.19 data stored x 0.023 USD = 0.00 USD (Data stored cost) - 15 data served x 0.15 USD = 2.25 USD (Data served cost) - 30,000 SSR requests per month x 0.0000003 USD per request = 0.01 USD (total monthly SSR request cost) - 30,000 SSR requests per month x 500ms x 0.001ms to sec conversion factor = 15,000.00 seconds (total monthly SSR compute duration) - 15,000.00 seconds x 0.0000555556 USD per second = 0.83 USD (total monthly SSR compute duration cost) - 0.10 USD (Build minutes cost) + 2.25 USD (Data served cost) + 0.01 USD (SSR request cost) + 0.83 USD (SSR compute duration cost) = 3.19 USD (Total cost) **=> AWS Amplify static web hosting cost (monthly): 3.19 USD**	$3.19
AWS Lambda	- 352,000 requests per month - Average duration: 5000ms - Allocated memory: 128 MB	- 352,000 requests * 500ms * 0.0001ms = 176,000.00 (Total compute seconds) - 0.125GB * 176,000.00 = 22,000.00 total compute (Total compute GB) - 22,000.00 * 0.0000166667 USD = 0.38 USD (monthly compute charges) - 22,000.00 * 0.0000166667 USD = 0.38 USD (monthly compute charges) - 352,000 * 0.0000002 USD = 0.07 USD (monthly request charges) - 0.5GB - 0.5GB (no additional charge) = 0.0GB (billable ephemeral storage per function) **=> Fees = 0.37 + 0.07 = 0.44 USD**	$0.44
AWS CloudTrail	- 200,000 events per month	$0	$0

AWS CodeCommit	- 1 active repository - 1000 git request per month	$0	$0
AWS API Gateway	- 200,000 requests per month	- 0.20 requests x 1,000,000 unit multiplier = 200,000 total REST API requests - Tiered price for: 200,000 requests - 200,000 requests x 0.0000035 USD = 0.70 USD - Total tier cost = 0.70 USD (REST API requests) **=> Tiered price total for REST API requests: 0.70 USD** - 0 USD per hour x 730 hours in a month = 0.00 USD for cache memory **=> Dedicated cache memory total price: 0.00 USD** - REST API cost (monthly): 0.70USD	$0.70
AWS SQS	- 160,000 requests per month	- 0.16 requests per month x 1000000 multiplier for million = 160,000.00 total standard queue requests - Tiered price for: 160,000 requests - 160,000 requests x 0.0 USD = 0.0 USD - Total tier cost = 0.0 USD (Standard queue requests cost) **=> Total SQS cost = 0.0 USD**	$0
AWS SNS	- 80,000 notifications per month	- Max (0 requests, 0 requests) = 0.00 requests - 80,000 notifications * 0.00 USD = 0.00 USD **=> Total SNS cost = 0.0 USD**	$0
S3	- 1000GB storage - 300,000 requests per month - Data Transferred: 150 GB per month	- Tiered price for: 1000GB - 1000GB * 0.025 USD = 25.00 USD **=> Total tier cost = 25.00 USD** - 80,000 PUT request = 0.44 USD - 220,000 GET requests = 0.0968 USD - 1000GB * 0.025 USD = 25.00 USD - Total tier cost = 25.00 USD - 80,000 PUT request = 0.44 USD **=> Total S3 cost = 25.0 + 0.0968 + 0.44 = 25.54 USD**	$25.54
AWS IAM	- 3 roles	$0	$0
AWS CloudWatch	- 30 custom metrics - 3 custom dashboards - 10 alarms - 5GBs logs ingestion	- Tiered price for: 30 metrics - 30 metrics x 0.30 USD = 9.00 USD - Total tier cost = 9.00 USD (Metrics cost (includes custom metrics)) **=> CloudWatch Metrics cost (monthly): 9.00 USD** - Tiered price for: 3 Dashboards - 3 Dashboards x 0.00 USD = 0.00 USD - Total tier cost = 0.00 USD (Dashboards cost) - 10 alarm metrics x 0.10 USD = 1.00 USD (Standard Resolution Alarms cost) **=> CloudWatch Dashboards and Alarms cost (monthly): 1.00 USD** - 5 GB x 0.67 USD = 3.35 USD **=> Standard Logs Data Ingested cost: 3.35 USD** **=> CloudWatch Logs Data Ingested tiered pricing cost: 0 USD** - 5.00 GB per month x 0.15 Storage compression factor x 1 Logs retention factor x 0.033 USD = 0.02475 USD **- Standard/Vended Logs data storage cost: 0.02475 USD** **- Logs delivered to S3 Data Ingested cost: 0 USD** **- Logs converted to Apache Parquet format cost: 0 USD** - 3.35 USD + 0.02475 USD = 3.37475 USD **=> CloudWatch Logs Ingested and Storage cost (monthly): 3.37 USD**	$13.37
AWS WAF	- 1 web ACL - 10 rules per ACL	- 1 Web ACLs per month x 5.00 USD = 5.00 USD (WAF Web ACLs cost)	$15.12

	- 200,000 requests per month	- 1 Web ACLs per month x 10.00 Billable Rules per web ACL per month x 1.00 USD = 10.00 USD (WAF Rules cost) - 0.20 requests per month x 1000000 multiplier for million x 0.0000006 USD = 0.12 USD (WAF Requests cost) - 5.00 USD + 10.00 USD + 0.12 USD = 15.12 USD **=> WAF cost (monthly): 15.12 USD**	
AWS Rekognition		**- Image analysis cost group 1 (monthly): 0 USD** - Tiered price for: 72,000 API calls - 72,000 API calls x 0.0012 USD = 86.40 USD - Total tier cost = 86.40 USD for images group 2 **=> Image analysis cost group 2 (monthly): 86.40 USD** **- Image analysis cost for Image Properties (monthly): 0 USD** **- Image analysis cost (monthly): 86.40 USD** **- Face metadata storage cost (monthly): 0 USD** **=> Image pricing (monthly): 86.40 USD**	
AWS DynamoDB	- 80,000 write requests per month - 120,000 read request per month - 50GB storage	-50 GB x 0.285 USD = 14.25 USD (Data storage cost) **- DynamoDB data storage cost (monthly): 14.25 USD** - 1 KB average item size / 1 KB = 1.00 unrounded write request units needed per item - RoundUp (1.000000000) = 1 write request units needed per item - 80,000 number of writes x 1 standard portion x 1 write request units for standard writes x 1 write request units needed per item = 80,000.00 write request units for standard writes - 80,000 number of writes x 0 transactional portion x 2 write request units for transactional writes x 1 write request units needed per item = 0.00 write request units for transactional writes - 80,000.00 write request units for standard writes + 0.00 write request units for transactional writes = 80,000.00 total write request units - 80,000.00 total write request units x 0.0000014231 USD = 0.11 USD write request cost **=> Monthly write cost (monthly): 0.11 USD** - 1 KB average item size / 4 KB = 0.25 unrounded read request units needed per item - Round Up (0.250000000) = 1 read request units needed per item - 120,000 number of reads x 1 eventually consistent portion x 0.5 read request units for eventually consistent reads x 1 read request units needed per item = 60,000.00 read request units for eventually consistent reads - 120,000 number of reads x 0 strongly consistent portion x 1 read request units for strongly consistent reads x 1 read request units needed per item = 0.00 read request units for strongly consistent reads - 120,000 number of reads x 0 transactional portion x 2 read request units for transactional reads x 1 read request units needed per item = 0.00 read request units for transactional reads - 60,000.00 read request units for eventually consistent reads + 0.00 read request units for strongly consistent reads + 0.00 read request units for transactional reads = 60,000.00 total read request units	

		- 60,000.00 total read request units x 0.0000002846 USD = 0.02 USD read request cost **=> Monthly read cost (monthly): 0.02 USD**	
AWS Media Convert	- 8,000 jobs per month - Average 5 minutes per job at 1080P - 1 output per job	-40,000 minutes utilized x 0.017 USD per minute = 680.00 USD for output(s) **=> Total cost for Output configuration 1 (monthly): 680.00 USD**	$680
Total:			$901.03

IV. Conclusion

This project outlines a robust cloud architecture for the Photo Album application, leveraging AWS services to ensure performance, scalability, reliability, security, and cost-efficiency. The design utilizes CloudFront for content caching, Cognito for authentication, and Route 53 for DNS mapping, among other services. By adopting a serverless and event-driven approach, the solution minimizes administration and optimizes resources, supporting the application's continued growth and success. This architecture meets current needs while providing flexibility for future expansions and enhancements, ensuring the application remains efficient, secure, and highly available.

References

[1] Amazon Web Services, "Serverless Computing," Amazon.com, [Online]. Available: https://aws.amazon.com/serverless/. [Accessed: 27-Jul-2024].

[2] A. S. Writer, "Serverless Computing: Pros and Cons," TechRepublic, [Online]. Available: https://www.techrepublic.com/article/serverless-computing-pros-and-cons-5-benefits-and-3-drawbacks/. [Accessed: 27-Jul-2024].

[3] Amazon Web Services, "Amazon Cognito User Pools Authentication Flow," 2024. [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/amazon-cognito-user-pools-authentication-flow.html. [Accessed: 27-Jul-2024].

[4] Amazon Web Services, "Introducing CloudFront Hosting Toolkit," 2024. [Online]. Available: https://aws.amazon.com/vi/blogs/networking-and-content-delivery/introducing-cloudfront-hosting-toolkit/. [Accessed: 27-Jul-2024].

[5] Amazon Web Services, "Routing to a CloudFront Distribution," 2024. [Online]. Available: https://docs.aws.amazon.com/Route53/latest/DeveloperGuide/routing-to-cloudfront-distribution.html. [Accessed: 27-Jul-2024].

[6] Amazon Web Services, "Integrations with AWS CodeCommit," 2024. [Online]. Available: https://docs.aws.amazon.com/codecommit/latest/userguide/integrations.html. [Accessed: 27-Jul-2024].

[7] Amazon Web Services, "How CloudFront Works," 2024. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/HowCloudFrontWorks.html. [Accessed: 27-Jul-2024].

[8] Amazon Web Services, "Evolution of Full-Stack Development with AWS Amplify," 2024. [Online]. Available: https://aws.amazon.com/blogs/mobile/evolution-of-full-stack-development-with-aws-amplify/. [Accessed: 27-Jul-2024].

[9] Amazon Web Services, "Configuring Caching with Amazon CloudFront," 2024. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/ConfiguringCaching.html. [Accessed: 27-Jul-2024].

[10] Amazon Web Services, "AWS Pricing," Amazon.com, [Online]. Available: https://aws.amazon.com/pricing/. [Accessed: 27-Jul-2024].

[11] Amazon Web Services, "AWS Pricing Calculator," Amazon.com, [Online]. Available: https://calculator.aws/#/. [Accessed: 27-Jul-2024].

[12] Byju's, "Difference Between Two-Tier and Three-Tier Database Architecture," Byju's, [Online]. Available:

https://byjus.com/gate/difference-between-two-tier-and-three-tier-database-architecture/. [Accessed: 27-Jul-2024].

[13] P. N. Demo, "2 and 3 Tier Architecture," Medium, [Online]. Available: https://medium.com/@paulndemo/2-and-3-tier-architecture-4a473e5ced3d. [Accessed: 27-Jul-2024].

[14] Amazon Web Services, "Serverless Computing," Amazon.com, [Online]. Available: https://aws.amazon.com/serverless/. [Accessed: 27-Jul-2024].

[15] A. S. Writer, "Serverless Computing: Pros and Cons," TechRepublic, [Online]. Available: https://www.techrepublic.com/article/serverless-computing-pros-and-cons-5-benefits-and-3-drawbacks/. [Accessed: 27-Jul-2024].

[16] Amazon Web Services, "Amazon ElastiCache Features," Amazon.com, [Online]. Available: https://aws.amazon.com/elasticache/features/. [Accessed: 27-Jul-2024].

[17] Amazon Web Services, "Amazon CloudFront Features," Amazon.com, [Online]. Available: https://aws.amazon.com/cloudfront/features/. [Accessed: 27-Jul-2024].

[18] Amazon Web Services, "SQL Databases," Amazon.com, [Online]. Available: https://aws.amazon.com/sql/. [Accessed: 27-Jul-2024].

[19] Amazon Web Services, "NoSQL Databases," Amazon.com, [Online]. Available: https://aws.amazon.com/nosql/. [Accessed: 27-Jul-2024].

[20] Amazon Web Services, "Amazon RDS," Amazon.com, [Online]. Available: https://aws.amazon.com/rds/. [Accessed: 27-Jul-2024].

[21] Amazon Web Services, "Amazon DynamoDB," Amazon.com, [Online]. Available: https://aws.amazon.com/dynamodb/. [Accessed: 27-Jul-2024].

[22] Back4App, "AWS Amplify vs S3," Back4App, [Online]. Available: https://blog.back4app.com/aws-amplify-vs-s3/. [Accessed: 27-Jul-2024].

[23] Amazon Web Services, "Amazon S3 Website Hosting," Amazon.com, [Online]. Available: https://docs.aws.amazon.com/AmazonS3/latest/userguide/WebsiteHosting.html. [Accessed: 27-Jul-2024].

[24] StackShare, "Amazon Elastic Transcoder vs AWS Elemental MediaConvert," StackShare, [Online]. Available: https://stackshare.io/stackups/amazon-elastic-transcoder-vs-aws-elemental-mediaconvert. [Accessed: 27-Jul-2024].

[25] Amazon Web Services, "Amazon Elastic Transcoder Developer Guide," Amazon.com, [Online]. Available: https://docs.aws.amazon.com/elastictranscoder/latest/developerguide/introduction.html. [Accessed: 27-Jul-2024].

[26] Amazon Web Services, "AWS Elemental MediaConvert User Guide," Amazon.com, [Online]. Available: https://docs.aws.amazon.com/mediaconvert/latest/ug/what-is.html. [Accessed: 27-Jul-2024].

[27] J. Effendi, "Queue-Based vs Publish-Subscribe Messaging Systems," Junaid Effendi, [Online]. Available: https://www.junaideffendi.com/p/messaging-systems-queue-based-vs. [Accessed: 27-Jul-2024].

[28] M. H. Cloud, "AWS EventBridge vs AWS SNS: Comparison," Medium, [Online]. Available: https://medium.com/awesome-cloud/aws-difference-between-amazon-eventbridge-and-amazon-sns-comparison-aws-eventbridge-vs-aws-sns-46708bf5313. [Accessed: 27-Jul-2024].

[29] GeeksforGeeks, "What is Pub-Sub?" GeeksforGeeks, [Online]. Available: https://www.geeksforgeeks.org/what-is-pub-sub/. [Accessed: 27-Jul-2024].

[30] Ably, "Amazon EventBridge vs Amazon SNS," Ably, [Online]. Available: https://ably.com/compare/amazon-eventbridge-vs-amazon-sns. [Accessed: 27-Jul-2024].