Serverless/Event-driven Architectural Design Report

Quang Tung Ta - 104222196
Student
Swinburne University of Technology
Hanoi, Vietnam
104222196@student.swin.edu.au

Thanh Trung Nguyen - 104060260
Student
Swinburne University of Technology
Hanoi, Vietnam
104060260@student.swin.edu.au

Quang Huy Nguyen - 104169507
Student
Swinburne University of Technology
Hanoi, Vietnam
104169507@student.swin.edu.au

Manh Dung Nguyen - 104181789
Student
Swinburne University of Technology
Hanoi, Vietnam
104181789@student.swin.edu.au

Tran Quang Minh Nguyen - 104179687
Student
Swinburne University of Technology
Hanoi, Vietnam
104179687@student.swin.edu.au

***Abstract*— This report paper presents a serverless cloud architecture designed to address the requirements of the Photo Album application. The architecture focuses on transitioning to managed cloud services, handling increasing demand, going serverless, and replacing the slow and costly relational database. Additionally, it aims to improve global response time and streamline video/photo uploads efficiently. The paper outlines its architecture and use cases and provides a detailed description of the AWS services utilized. Furthermore, it provides a rationale for the design, explaining how it meets the business requirements and design criteria.**

***Keywords*—aws, cloud architecture, serverless architecture, event-driven architecture, media upload site**

I. Introduction

This paper presents a serverless cloud architecture for the Photo Album application to fulfill a number of requirements such as transitioning towards managed cloud services, handling increasing demand, going serverless, replacing the slow and costly relational database, increasing global response time, and processing video/photo uploads.

The paper first presents the architecture of the solution as well as its use cases. The paper then describes the AWS services used in detail. Afterward, the paper provides a rationale for the design, including how it fulfills the business requirements and design criteria. Along the way, alternative solutions will be discussed. Finally, the paper summarizes the monthly costs incurred by each service.

The following assumptions are made about the functionality and access pattern of the Photo Album application:

- The application requires all users to be authenticated to access its services. Authenticated users can upload photos and videos to the site and see only the content they have uploaded. For each uploaded photo, two thumbnail versions (mobile and desktop) will be generated and photo tags are automatically identified. For each uploaded video, the application transcodes it into a suitable format for display (the exact format is unknown).
- The number of daily users is 300. The number of daily photo and video uploads per user is 5 and 0.5 respectively. The number of daily view requests per user is 10. The number of photos and videos requested per view is 20. The average size of a photo and a video is 2 MB and 500 MB respectively. The average uploaded video length is 5 minutes. No photo or video exceeds 5 GB.

II. Architectural diagram

The diagram below shows the architecture of the solution, including all AWS services used and their interactions. The architecture is divided into three tiers: presentation, logic, and data. The presentation tier is outlined in green, the data tier in blue, and the logic tier in red. The logic tier is split into four blocks for easier viewing.

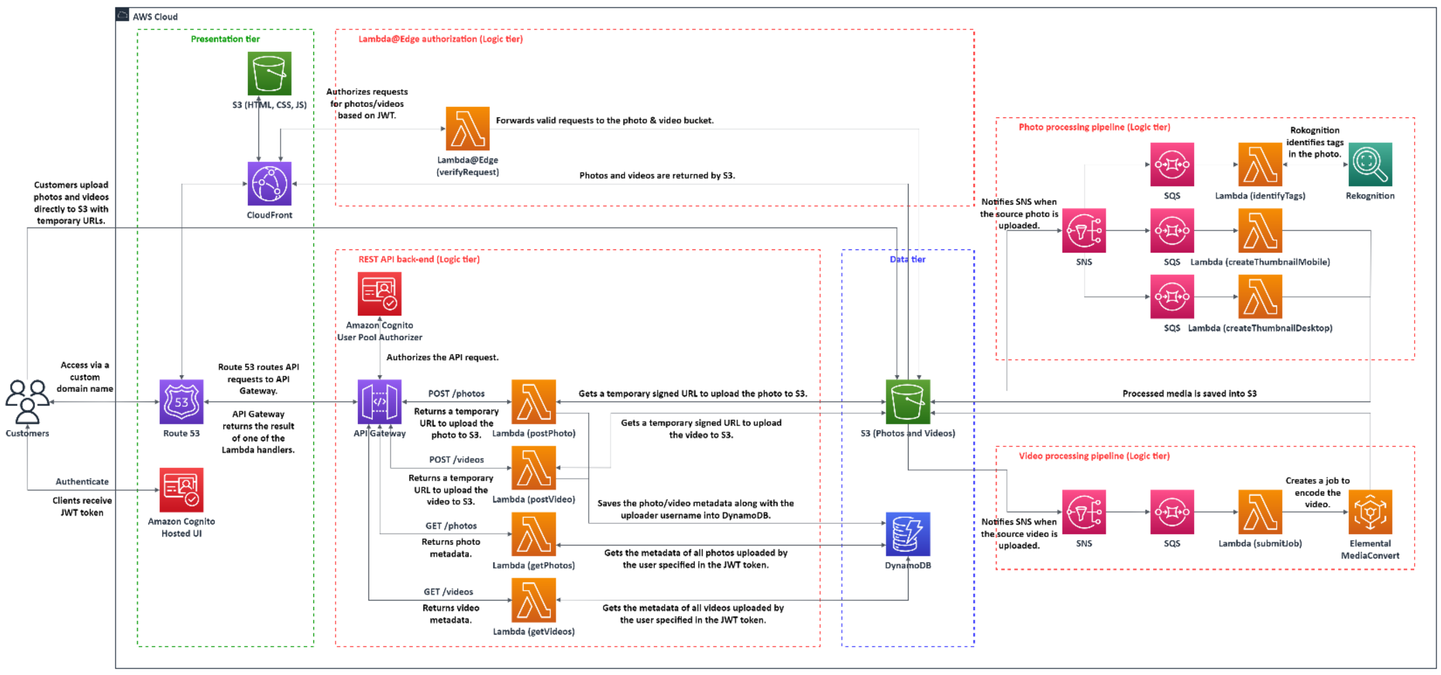
AWS Cloud
Presentation tier
S3 (HTML, CSS, JS)
CloudFront
Lambda@Edge authorization (Logic tier)
Authorizes requests for photos/videos based on JWT.
Lambda@Edge (verifyRequest)
Forwards valid requests to the photo & video bucket.
Photos and videos are returned by S3.
Customers upload photos and videos directly to S3 with temporary URLs.
REST API back-end (Logic tier)
Amazon Cognito User Pool Authorizer
Authorizes the API request.
Access via a custom domain name
Customers
Route 53
Route 53 routes API requests to API Gateway.
API Gateway returns the result of one of the Lambda handlers.
API Gateway
POST /photos
Returns a temporary URL to upload the photo to S3.
Lambda (postPhoto)
Gets a temporary signed URL to upload the photo to S3.
POST /videos
Returns a temporary URL to upload the video to S3.
Lambda (postVideo)
Gets a temporary signed URL to upload the video to S3.
Saves the photo/video metadata along with the uploader username into DynamoDB.
GET /photos
Returns photo metadata.
Lambda (getPhotos)
Gets the metadata of all photos uploaded by the user specified in the JWT token.
GET /videos
Returns video metadata.
Lambda (getVideos)
Gets the metadata of all videos uploaded by the user specified in the JWT token.
Authenticate
Clients receive JWT token
Amazon Cognito Hosted UI
Data tier
S3 (Photos and Videos)
DynamoDB
Photo processing pipeline (Logic tier)
Notifies SNS when the source photo is uploaded.
SNS
SQS
Lambda (identifyTags)
Rekognition identifies tags in the photo.
Rekognition
Lambda (createThumbnailMobile)
Lambda (createThumbnailDesktop)
Processed media is saved into S3
Video processing pipeline (Logic tier)
Notifies SNS when the source video is uploaded.
Lambda (submitJob)
Creates a job to encode the video.
Elemental MediaConvert

Fig. 1: The architectural diagram for the serverless solution.

III. UML Collaboration Diagrams

This section presents three UML collaboration diagrams to demonstrate how the proposed AWS services interact to achieve the desired use cases. Four use cases are considered: uploading and processing photos, uploading and processing videos, viewing all photos, and viewing all videos. The latter two use cases are combined into one UML diagram.

A. *Upload and process photos*

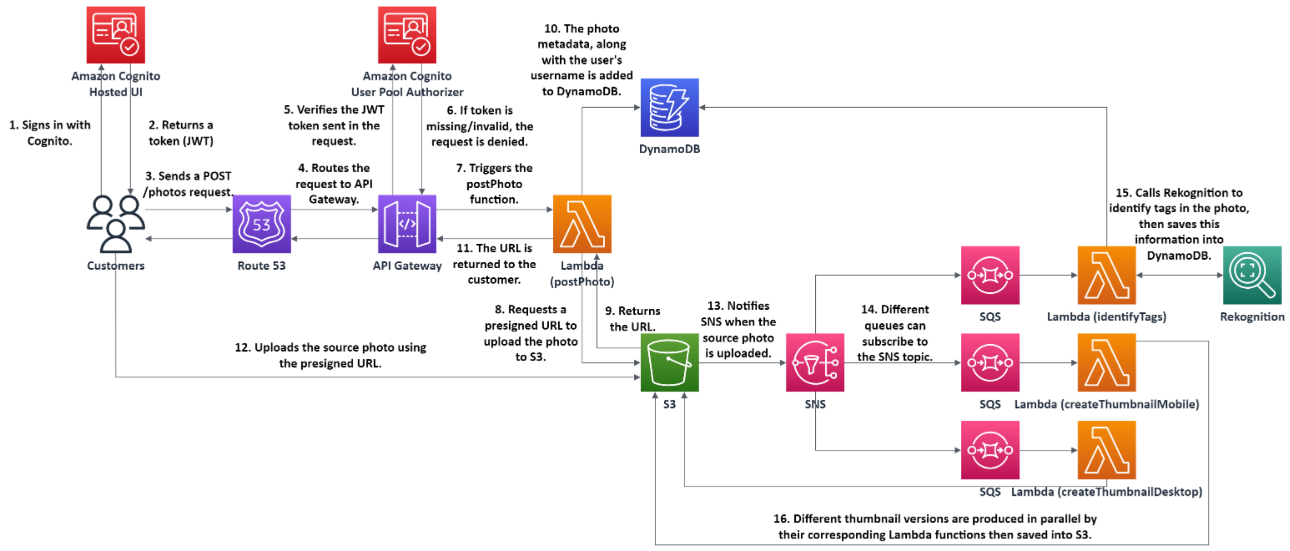
Amazon Cognito Hosted UI
1. Signs in with Cognito.
2. Returns a token (JWT)
3. Sends a POST /photos request.
Customers
Route 53
Amazon Cognito User Pool Authorizer
5. Verifies the JWT token sent in the request.
4. Routes the request to API Gateway.
API Gateway
6. If token is missing/invalid, the request is denied.
7. Triggers the postPhoto function.
11. The URL is returned to the customer.
10. The photo metadata, along with the user's username is added to DynamoDB.
DynamoDB
Lambda (postPhoto)
8. Requests a presigned URL to upload the photo to S3.
9. Returns the URL.
12. Uploads the source photo using the presigned URL.
S3
13. Notifies SNS when the source photo is uploaded.
SNS
14. Different queues can subscribe to the SNS topic.
SQS
Lambda (identifyTags)
15. Calls Rekognition to identify tags in the photo, then saves this information into DynamoDB.
Rekognition
Lambda (createThumbnailMobile)
Lambda (createThumbnailDesktop)
16. Different thumbnail versions are produced in parallel by their corresponding Lambda functions then saved into S3.

Fig. 2: The UML diagram showing the process for uploading and processing photos.

B. *Upload and process videos*

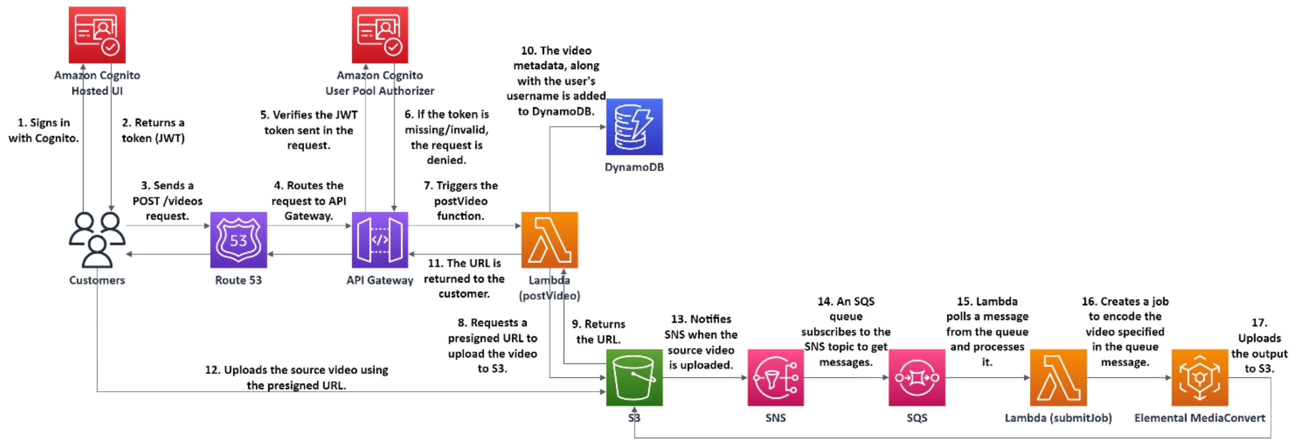
Amazon Cognito Hosted UI
Amazon Cognito User Pool Authorizer
10. The video metadata, along with the user's username is added to DynamoDB.
1. Signs in with Cognito.
2. Returns a token (JWT)
5. Verifies the JWT token sent in the request.
6. If the token is missing/invalid, the request is denied.
DynamoDB
3. Sends a POST /videos request.
4. Routes the request to API Gateway.
7. Triggers the postVideo function.
Customers
Route 53
API Gateway
11. The URL is returned to the customer.
Lambda (postVideo)
8. Requests a presigned URL to upload the video to S3.
9. Returns the URL.
13. Notifies SNS when the source video is uploaded.
14. An SQS queue subscribes to the SNS topic to get messages.
15. Lambda polls a message from the queue and processes it.
16. Creates a job to encode the video specified in the queue message.
17. Uploads the output to S3.
12. Uploads the source video using the presigned URL.
S3
SNS
SQS
Lambda (submitJob)
Elemental MediaConvert

Fig. 3: The UML diagram showing the process for uploading and processing videos.

C. *View all photos or videos*

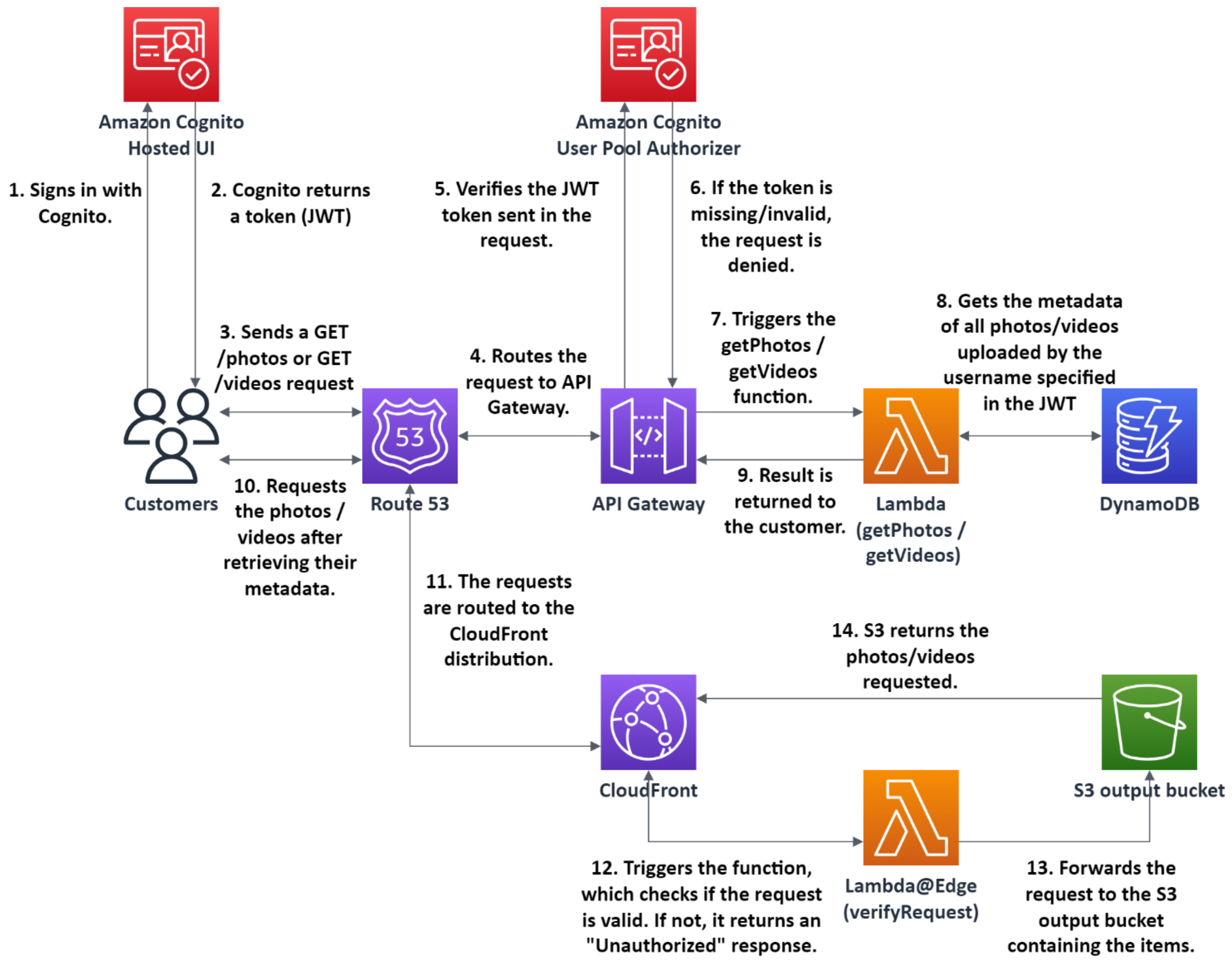
Amazon Cognito Hosted UI
Amazon Cognito User Pool Authorizer
1. Signs in with Cognito.
2. Cognito returns a token (JWT)
5. Verifies the JWT token sent in the request.
6. If the token is missing/invalid, the request is denied.
8. Gets the metadata of all photos/videos uploaded by the username specified in the JWT
3. Sends a GET /photos or GET /videos request
4. Routes the request to API Gateway.
7. Triggers the getPhotos / getVideos function.
Customers
Route 53
API Gateway
Lambda (getPhotos / getVideos)
DynamoDB
10. Requests the photos / videos after retrieving their metadata.
9. Result is returned to the customer.
11. The requests are routed to the CloudFront distribution.
14. S3 returns the photos/videos requested.
CloudFront
S3 output bucket
12. Triggers the function, which checks if the request is valid. If not, it returns an "Unauthorized" response.
Lambda@Edge (verifyRequest)
13. Forwards the request to the S3 output bucket containing the items.

Fig. 4: The UML diagram showing the process for viewing photos or videos.

IV. AWS SERVICES USED

This section describes in detail the AWS services used as part of this architecture. A total of 11 different services are used.

A. *Route 53*

Service description: Amazon Route 53 is a web service for the Domain Name System (DNS) that is highly available and scalable. Domain registration, DNS routing, and health checking are the three main tasks that Route 53 can be used for. User queries are routed through the AWS Route 53 DNS service to AWS-based infrastructure such as Amazon EC2 instances, Amazon S3 buckets, and ELB load balancers.

Requirement fulfillment: Route 53 is used in this application to let customers access the service with a custom domain name. AWS' globally distributed network of DNS servers enables the consistent routing of customers to the application by overcoming network issues.

B. CloudFront

Service description: Amazon CloudFront is a content delivery network that speeds up the distribution of static and dynamic content, such as web pages, photos, videos, scripts, and other assets, to end users all over the world. It works by caching content in edge locations around the world, shortening the distance between customers and content and making delivery quicker and more dependable.

Requirement fulfillment: CloudFront is used in this application to improve response times for global customers by caching content in edge locations scattered across the globe. CloudFront adds an extra layer of security to the application by delivering content over HTTPS. Additionally, a Lambda@Edge is used with CloudFront for controlling access to photos and videos in the S3 bucket.

C. Lambda

Service description: AWS Lambda is a serverless computing service that enables developers to run code without provisioning or managing servers. Lambda functions are triggered by events, such as HTTP requests, changes to Amazon Simple Storage Service (S3) buckets, or Amazon CloudWatch events. Lambda functions can be written in a variety of programming languages, including Python, Java, Node.js, and Go.

Requirement fulfillment: Lambda aligns with the business' aim to go serverless. It enables the application's backend logic to be implemented without provisioning servers. Lambda meets the application's scaling requirements by automatically scaling to meet demand, allowing the application to handle traffic spikes. Lambda invokes functions in a secure execution environment, which adds extra security to the application. It is also extremely cheap, with the first one million requests each month free of charge.

D. API Gateway

Service description: AWS API Gateway is a fully managed service that makes it easy to create, publish, maintain, monitor, and secure APIs at any scale. API Gateway provides a single point of entry for APIs, making it easy for developers to access services. API Gateway also supports a variety of features, such as API throttling, CORS, and authentication, which can help secure and protect APIs.

Requirement fulfillment: Since the backend logic for the application is implemented in Lambda functions, API Gateway will be used to connect the frontend with the backend. The business scenario for the Photo Album application requires it to handle high volumes of traffic without impacting performance. The application also needs to be scalable, reliable, and secure. AWS API Gateway can help to fulfill these requirements by providing a scalable and reliable API endpoint for the application. API Gateway can also help to secure the application by providing features such as API throttling and authentication.

E. Simple Storage Service (S3)

Service description: Amazon Simple Storage Service (Amazon S3) is a cloud storage solution that is fully managed and scalable, ensuring 99.999999999% uptime. It offers a wide range of management features, allowing businesses to efficiently organize and optimize storage according to their specific requirements. Utilizing an object storage architecture, Amazon S3 prioritizes scalability, high availability, low latency, and durability. Data is organized into buckets, and each object is identified by a unique user-assigned key, with the capacity to be as large as five terabytes.

Requirement fulfillment: Amazon S3 is used to store customers' uploaded photos and videos due to its vast storage capacity and ability to store objects in any format, which makes it suitable for the application's aim to handle many media formats in the future. It also integrates seamlessly with other services like Lambda and Elemental MediaConvert, enabling the automatic production of reformatted or transcoded versions of uploaded files. This versatility makes it a powerful and flexible cloud storage solution for a wide range of use cases.

F. DynamoDB

Service description: Amazon DynamoDB is a fully managed and serverless NoSQL database optimized for high-performance applications of any size. It provides essential features like built-in security, continuous backups, automated multi-Region replication, in-memory caching, and data import/export tools. With these capabilities, DynamoDB offers a reliable and scalable solution for handling various workloads and data requirements.

Requirement fulfillment: DynamoDB will be used to store the metadata of photos and videos uploaded by customers. Since DynamoDB uses a simple key-value data model, it is suitable for the business' use case of storing metadata. DynamoDB not only offers cost efficiency but also scalability for the company. DynamoDB is cheaper than traditional relational databases and provides faster speeds. Its horizontal scaling feature allows cost-efficient scaling as demand grows.

G. Cognito

Service description: Amazon Cognito is a highly scalable service that helps developers add authentication features to mobile and web applications. Developers can integrate external identity providers (such as Facebook, Twitter, or Amazon) with their own identity providers. Cognito allows user management and helps customers access their resources securely.

Requirement fulfillment: The application is assumed to require authentication before users can start using its services. Cognito meets this requirement by adding an authentication feature to the application. Users will be asked to authenticate to receive a JSON Web Token containing their identity which is then used for the verification of every request on the backend. Cognito is a fully-managed service that can scale up to millions of users, meeting the business' requirements of using managed services and catering to increasing demand.

H. Elemental MediaConvert

Service description: Elemental MediaConvert is a file-based video transcoding tool that offers broadcast-quality functionality. It also supports multiple input and output formats. Based on the length of the processed movies, MediaConvert processes videos with an on-demand pricing model.

Requirements fulfillment: MediaConvert is used to transcode videos when users upload them to S3. MediaConvert fully manages resource provisioning, orchestration, and scaling, which makes it suitable for the business' goal to use fully-managed services. It also supports the production of a wide range of video outputs, making it highly extensible for future requirements. MediaConvert charges in minutes of transcoded video, and there are no up-front fees. This prevents resource wastage when the service is not being used.

I. Rekognition

Service description: Rekognition is a computer vision solution for identifying features of images and videos. It provides pre-trained AI models that are ready for cloud architecture integration.

Requirement fulfillment: Rekognition helps the application identify tags from uploaded photos using machine learning. It is simple to use and eliminates the costs of creating, training, and deploying custom models. Rekognition is also a fully-managed service with features available out-of-the-box.

J. Simple Queue Service (SQS)

Service description: Amazon SQS is a fully-managed message queue service that enables communication between software components such as microservices and serverless applications. SQS stores messages between components in queues, and consumers of these messages can pull them for processing at their own pace. SQS enables the decoupling of the application, as the failure of one component does not affect other components.

Requirement fulfillment: SQS works with SNS to decouple the photo and video processing pipelines of the application. Photo and video processing is a time-consuming task and the time it takes to process one file varies greatly depending on the file size. SQS stores jobs in queues and lets nodes (which can be Lambda functions, Elemental MediaConvert, etc.) pull jobs for processing at their own pace. This prevents nodes from being overloaded and jobs from being lost, thus increasing the reliability of the system.

K. Simple Notification Service (SNS)

Service description: Amazon SNS is a fully-managed messaging service that enables communication between applications and applications (A2A) and applications and people (A2P). Following a publish-subscribe paradigm, SNS is a scalable, flexible, and cost-effective solution for publishing messages from an application to subscribers. SNS can be used to decouple architectures.

Requirement fulfillment: SNS works with SNS to decouple the photo and video processing pipelines of the application. The business requires that multiple versions of a source media file be produced when it is uploaded to S3. Ideally, these processes should run in parallel to improve performance. An SNS topic can be set up to receive notifications from S3 when a file is uploaded and push them to subscribing SQS queues for processing, creating a fan-out architecture. This makes the system highly extensible as more SQS queues corresponding to different processing jobs can be attached to SNS to produce other output formats.

V. Design Rationale

A. Requirement fulfillment and alternatives

1) Minimize the need for in-house systems administration with managed cloud services

This architecture leverages a number of fully-managed cloud services to align with the business' goal of minimizing in-house systems administration. These services include DynamoDB, S3, API Gateway, and Cognito.

DynamoDB is used to store the metadata of photos and videos uploaded to the application. DynamoDB is schema-less and stores data in a simple key-value data model, making it ideal for storing metadata. DynamoDB offers great performance and scalability [1]. It is excellent for high-throughput applications since it offers read and write operations with single-digit millisecond latency. Because of its automatic scaling features, it can effectively manage a range of workloads. Thanks to its multi-AZ replication and automated failover features, DynamoDB is incredibly dependable. By duplicating data over numerous servers located inside AWS data centers, it provides data durability. DynamoDB offers both at-rest and in-transit data encryption. Through AWS Identity and Access Management (IAM) policies and granular access control, access to tables and data may be restricted. DynamoDB uses a pay-per-request pricing approach, where you pay for both the storage you utilize and the read and write queries you perform. It is cost-effective for small to medium workloads, making it ideal for the photo album application.

API Gateway is a fully-managed service that simplifies the creation, publication, maintenance, monitoring, and security of APIs at any size for developers. In this application, it acts as a secure bridge between the frontend application and the backend logic, which is implemented in Lambda functions. API Gateway is built for excellent performance and has the capacity to deal with high demand for incoming requests. It adjusts automatically to accommodate traffic needs. As API Gateway is a fully managed service, it comes with automated load balancing and fault tolerance, making it very reliable [2]. You may utilize IAM roles, unique authentication methods, or AWS Cognito for user authentication and authorization to restrict access to APIs using API Gateway. Pricing for API Gateway is determined by the volume of API requests, the amount of data sent, and the amount of caching used [3]. It is cost-effective for APIs with moderate to high usage.

Amazon S3 is an object storage service that allows developers to store and retrieve any amount of data over the Internet. It provides highly scalable, durable, and secure storage for various types of data, such as images, videos, documents, and backups. In this application, it is used to host the static website and store photos and videos uploaded by customers. S3 offers excellent performance and scalability, and it can handle massive volumes of data and concurrent queries. S3 has a high level of durability, and data replication across many availability zones ensures that data is available even in the event of hardware failure. S3 provides strong data security features, including server-side encryption and access control guidelines [4]. The cost of S3 is determined by the quantity of storage used and the volume of outbound data transfer made. In general, it is economical, especially when storing a lot of data.

Amazon Cognito is a fully managed service that provides identity verification and user management capabilities to secure and control access to applications. Cognito is a highly scalable service that can manage user sign-ups, sign-ins, and user data storage on a large scale [5]. Because Cognito is a fully managed service, high availability, and reliability are guaranteed by AWS. For safe user authentication, Cognito supports a number of identity suppliers and offers multi-factor authentication (MFA). For the first 50,000 active monthly customers, Cognito provides a free tier. Beyond that, fees are determined by the number of users that are active each month [6]. This makes Cognito ideal for the current number of users.

2) Cope with demand doubling every 6 months

In order to meet increasing demand while also aligning with the business' preference for fully-managed services and serverless architecture, Lambda, DynamoDB, and API Gateway will be included as part of the solution.

Lambda offers automatic scaling, ensuring that the application can handle varying workloads and accommodate the Photo Album's anticipated growth. By executing code only when triggered, Lambda efficiently utilizes resources and scales seamlessly to meet increasing demand. Moreover, Lambda can maintain its concurrency by reusing the pre-initialized execution environment [7], which makes it catch up with a tremendous number of incoming requests.

DynamoDB's auto-scaling feature utilizes the AWS Application Auto Scaling service to automatically adjust provisioned throughput capacity based on real-time traffic patterns. This ensures that tables or global secondary indexes can scale up their read and write capacity to accommodate sudden spikes in traffic without experiencing throttling. When the workload decreases, Application Auto Scaling reduces the throughput, preventing charges for unused provisioned capacity [8]. In essence, DynamoDB auto scaling dynamically manages capacity to optimize performance and cost efficiency based on actual demand. In this business scenario, DynamoDB is one of the best solutions for dealing with the increasing demand.

API Gateway functions as a bridge between frontend and backend services, and it dynamically adjusts its capacity to accommodate the incoming traffic [9]. There are no arbitrary limits or throttling of invocations on our backend operations by API Gateway. All requests not affected by throttling or caching settings in the API Gateway console are directed to designated backend operations. This ensures the smooth and scalable handling of API traffic, making API Gateway a dependable solution for managing our API workloads. For that reason, API Gateway is a great choice of proxy to both protect the backend and adapt to the rising demand.

3) Adopt a serverless/event-driven solution

In order to go serverless, the business must consider options to host its website and backend logic. This solution hosts the website statically in an S3 bucket and implements the backend logic with Lambda functions. The static webpages interact with the backend by making API calls to API Gateway through the AWS SDK for JavaScript.

S3 is an ideal option for web hosting in this business scenario as the web pages are entirely static. S3 also closely integrates with Amazon CloudFront, a content delivery network (CDN) service, allowing static web content such as HTML, CSS, and JavaScript to be cached for faster retrieval. S3 boasts high durability and availability of stored data, replicating data across multiple availability zones for reliable storage. Security features such as access control lists (ACLs), bucket policies, and server-side encryption can protect the hosted content from unauthorized access. S3 charges for the amount of data stored and the number of requests made, making it a very cheap hosting solution as web pages are typically very small in size.

Lambda is a serverless computing service used to implement business logic such as adding and retrieving photos and videos. Its serverless architecture enables automatic scaling to match varying workloads, ensuring that the application efficiently handles peak demand while minimizing costs during idle periods. Lambda's ability to execute code in response to events allows it to instantly scale up to process incoming requests, making it a perfect fit for the application's event-driven nature. Its distributed nature across availability zones enhances the reliability of the application as functions are automatically replicated across multiple data centers. Additionally, Lambda provides built-in error handling and retries, guaranteeing that media processing tasks are resilient to transient failures. Lambda supports data encryption in transit and at rest, providing end-to-end data security and privacy for media uploads and processed content [10]. Users are charged by request and by processing duration, making Lambda ideal for the unpredictable workload of the business [11].

A serverless alternative to Lambda is ***AWS Fargate***, which is a service used to run containers without having to provision servers [12]. Fargate is more complex to implement as it requires the development and maintenance of containers. While Fargate also offers automatic scaling, high availability, and security, it requires additional configuration to have these features in place. Fargate's cost-efficiency depends on the container runtime and resource utilization, which may involve more upfront provisioning. This makes Lambda the better option for serverless backend logic.

API Gateway acts as a front door for the backend application, efficiently managing and routing incoming API requests to backend services. It automatically scales to handle varying levels of traffic, seamlessly accommodating the increasing demand as the user base grows. API Gateway enhances the application's reliability through its fault-tolerant architecture. Its automatic scaling and load-balancing capabilities ensure that the system can gracefully handle surges in traffic without affecting overall reliability. API Gateway also supports built-in features like request throttling and rate limiting, preventing overload situations and improving overall system stability. It adds an extra layer of security to backend services by handling authentication, authorization, and access control for incoming API requests. Moreover, it supports SSL/TLS encryption for secure data transmission, safeguarding sensitive information from interception. API Gateway cuts costs by eliminating the need for maintaining and managing individual API servers. With API Gateway, developers only pay for the number of API calls and data transferred, making it a cost-effective choice, especially for applications with varying levels of traffic.

4) Replace the relational database with faster and cheaper options

The application will require a database to store the metadata of uploaded photos and videos. Given the relatively simple structure of the data, a relational data model is not an ideal option. Relational models require a fixed schema and are typically used for complex data that require joins. For simple media metadata like the business' use case, a NoSQL model is more appropriate. NoSQL comprises many data models such as key-value, documents, graphs, and more. In particular, the key-value model is used as it is suitable for the high throughput, low-latency reads and writes, and scaling requirements of a media uploading application [13]. This architecture uses ***DynamoDB*** for this purpose. DynamoDB is a fully-managed NoSQL database service that stores data in a key-value data model. DynamoDB offers single-digit millisecond performance at nearly unlimited throughput [14]. It scales horizontally and offers auto-scaling features to meet database access demands. DynamoDB also encrypts data at rest and provides access control through IAM for security.

5) Improve global response times for users outside Australia

In order to make the application faster for global users, this architecture uses ***Amazon CloudFront*** to cache content. The types of content cached include static web files such as HTML, CSS, and JavaScript, and media files uploaded by customers. CloudFront brings content closer to customers by distributing it over a worldwide network of edge locations [15]. While configuring a distribution, settings like protocol policies or cache behavior can be adjusted. CloudFront integrates closely with S3, which this architecture uses extensively to store web and user-uploaded content. In order to limit customers' access to only the photo and video files they have uploaded, a ***Lambda@Edge*** function will be implemented to verify the request. CloudFront invokes this function every time it receives a request for a photo or video file, adding an extra layer of security to the application.

6) Handle video media

To cater to video uploads, the solution will build on top of the photo upload architecture. Uploaded videos are stored in the same S3 bucket as photos and their metadata is stored in the same DynamoDB database. An additional service called ***Elemental MediaConvert*** is added to transcode videos. It supports a broad variety of input and output formats, codecs, and resolutions, including well-known video codecs like H.264, H.265, VP9, and AV1, and audio codecs like AAC, MP3, and others [16]. Customization of audio channels, bitrates, and other elements is also possible. It offers options including resizing, watermarking, and video cropping. MediaConvert offers data encryption, secure access control through IAM, and compliance with industry standards to ensure data protection. It is a fully-managed, highly available, and reliable service that can handle large transcoding workloads. Elemental MediaConvert charges in minutes of video output, so there is no wastage when the service is not being used [17].

An alternative to Elemental MediaConvert is ***Elastic Transcoder***. It is a video transcoding service offered by AWS that supports a wide range of input and output formats, codecs, and resolutions. This service offers data encryption and access control with IAM to secure input and output files [18]. It is also scalable and uses the same pricing model as Elemental MediaConvert. Compared to MediaConvert, however, it offers fewer features and customization options. AWS now recommends switching to MediaConvert as Elastic Transcoder is an older service that does not have the same advanced feature set. In the future, it may be desirable for the business to produce a wide range of transcoding outputs. This makes Elemental MediaConvert the better option for video transcoding.

7) Automatically process uploaded media with an extensible and decoupled architecture

In the architecture, when photos and videos are uploaded by customers into the S3 bucket, they are automatically sent to their corresponding processing pipelines. This is enabled by ***S3 Event Notifications*** that are sent to SNS when a media file is uploaded.

The photo processing pipeline consists of an SNS topic that receives notifications from S3 when new photos are uploaded, multiple SQS queues that subscribe to the SNS topic, and multiple Lambda functions that poll messages from corresponding SQS queues for processing. The combination of SNS and SQS creates a "fan-out" architecture that enables one image to be processed in parallel by different Lambda operations [19]. This effectively decouples the photo processing architecture by making different processes run independently from each other. For instance, one Lambda function can call Rekognition to identify tags in the photo while another creates a mobile thumbnail for it. This architecture is also highly extensible, as additional SQS queues can subscribe to the SNS topic to allow for new processing outputs.

The video processing pipeline follows a similar structure. Currently, it consists of only one SQS queue as this is enough for present transcoding purposes. However, in the future, more queues can be added to cater to more advanced processing features.

The key services of this architecture are ***SNS (Simple Notification Service)*** and ***SQS (Simple Queue Service)***. SNS delivers messages from publishers of a topic to its subscribers (which in this architecture are S3 and SQS, respectively) with nearly unlimited throughput. SQS decouples application components that may not process the same amount of work at the same time. Together, they increase the application performance by allowing multiple processes to run in parallel and allowing each process to work at its pace without being overwhelmed by the number of requests. They also increase the application's reliability because if a process fails, its work is still saved in the queue to be processed later when it recovers.

Another related service to decouple architectures is Amazon MQ, which provides message brokers that let different application components communicate using different languages, operating systems, and protocols. However, this service is excessively complex for this application, and AWS recommends using SNS and SQS instead [20].

8) Authenticate users to control access to the application

The Photo Album application should only be available to authenticated customers. After authenticating, customers can interact with API Gateway to access the backend logic. This will be achieved with ***Amazon Cognito***, which adds authentication and authorization capabilities to applications. It comes with a user pool, which is a directory of users, and a pre-built authentication server called Hosted UI where users can sign up and sign in to the application [21]. Authenticated customers' requests to API Gateway will automatically include a JSON Web Token that contains their identity. API Gateway can use Cognito User Pool Authorizer to verify that the token is valid before invoking the backend logic [22]. This solution is simple to implement, available out-of-the-box, and can support up to 40 million users per user pool [23]. Cognito supports compromised credential protection and complies with many security requirements, which makes it a secure solution.

An alternative to using Cognito is to implement a custom authentication solution, then authorize requests to API Gateway with a custom ***Lambda authorizer function*** [24]. This approach offers greater flexibility to developers but requires more work as custom authentication and authorization code must be written. It poses security risks if the custom solution does not meet security requirements. Additionally, it necessitates an extra Lambda call for every API request, which can incur extra costs. Given the business' preference for simplicity, using Cognito is the better solution.

B. Design criteria

1) Performance and scalability

This section details how the proposed services meet the performance and scalability criteria of the application.

CloudFront caches content in edge locations worldwide, significantly improving response times for global users.

Lambda offers automatic scaling and is event-driven. Its ability to execute code in response to events allows it to seamlessly handle peak demand, efficiently processing media uploads and generating alternative versions like thumbnails and transcoded videos.

API Gateway efficiently manages incoming API requests and automatically scales to handle varying levels of traffic. Its serverless architecture and built-in caching capabilities improve overall performance. At the same time, the ability to distribute requests across multiple backend services ensures seamless scalability.

Simple Storage Service (S3) provides efficient and reliable storage for photo and media assets. S3's ability to handle massive amounts of data and concurrent requests, coupled with its global distribution through Amazon CloudFront, ensures optimized performance and reduced latency.

DynamoDB efficiently stores the metadata of photos and videos. With its simple key-value data model, DynamoDB is well-suited for this use case, providing cost efficiency and faster speeds compared to traditional relational databases. The horizontal scaling feature allows for seamless and cost-efficient scaling as demand for storage grows.

Cognito provides a fully managed authentication service that can scale to support millions of users. Its managed nature aligns with the business' preference for managed services, ensuring a smooth and scalable user authentication experience without the need for in-house systems administration.

Elemental MediaConvert provides seamless video transcoding with fully managed resource provisioning, orchestration, and scaling.

Rekognition provides a fully managed and cost-effective solution for identifying tags from uploaded photos using machine learning. With its out-of-the-box features, Rekognition simplifies the process, eliminating the need for creating, training, and deploying custom models, making it an efficient and scalable choice for photo tagging needs.

Simple Queue Service (SQS) decouples the photo and video processing pipelines through job queuing. SQS stores jobs in queues, allowing nodes (such as Lambda functions or Elemental MediaConvert) to pull and process jobs at their own pace, preventing overload and job loss, thus increasing system reliability.

Simple Notification Service (SNS) decouples the photo and video processing pipelines, enabling parallel processing to improve performance. By setting up an SNS topic to receive notifications from S3 when a file is uploaded and pushing them to subscribing SQS queues for processing, SNS creates a fan-out architecture, enabling extensibility.

2) Reliability

This section details how the proposed services meet the reliability criteria of the application.

Route 53 provides a globally distributed network of DNS servers, ensuring consistent and reliable routing of customers to the service using a custom domain name.

API Gateway handles high volumes of traffic without compromising performance. API Gateway's features, such as API throttling and authentication, enhance the application's security, ensuring a secure and reliable user experience.

Simple Storage Service (S3) provides vast storage capacity and support for various media formats, ensuring secure and reliable storage for customers' uploaded photos and videos.

DynamoDB provides a fast and reliable storage solution for the metadata of photos and videos thanks to its simple key-value data model and horizontal scaling feature.

Simple Queue Service (SQS) decouples the photo and video processing pipelines, ensuring a robust and fault-tolerant system. SQS stores jobs in queues and allows processing nodes like Lambda functions and Elemental MediaConvert to pull and process jobs at their own pace, preventing overload and job loss, and enhancing the overall reliability of the application's processing tasks.

Simple Notification Service (SNS) decouples the photo and video processing pipelines, enabling parallel processing of multiple versions of source media files to improve performance. This lets different operations work independently.

3) Security

This section details how the proposed services meet the security criteria of the application.

Cognito enables robust security with token-based authentication, claims-based tokens, and encryption for sensitive data. Token revocation and session management enhance security, while temporary AWS credentials limit exposure to potential breaches. Its flexibility in handling token size and support for Multi-Factor Authentication (MFA) further bolster its security features.

Lambda offers robust security with features like IAM roles for precise permissions, fine-grained access control to restrict resource access, and encryption for data privacy. Isolated execution environments and regular security updates further enhance Lambda's security.

S3 incorporates strong security measures for data and media asset protection. Access control lists (ACLs) and bucket policies enable restricting access to authorized users only. Moreover, S3 provides server-side encryption, ensuring data remains encrypted at rest, and safeguarding sensitive information from unauthorized access [19].

DynamoDB provides data encryption both at rest and during transit. Using AWS Identity and Access Management (IAM) policies and fine-grained access control, access to tables and data can be limited.

API Gateway uses IAM roles, unique authentication methods, or AWS Cognito for request authentication and authorization. These mechanisms enable you to restrict access to your APIs based on user identities and permissions effectively.

4) Cost

This section details how the proposed services meet the cost criteria of the application.

Pay-per-use pricing: The majority of these services employ a pay-per-use pricing structure (e.g. Lambda, API Gateway, Rekognition, and Elemental MediaConvert), which means that the business only pays for the resources and requests that it actually uses. Since unused resources are not charged, these services are cost-effective for small to large-scale applications.

Scalability: The application can react to changing workloads without over-provisioning resources thanks to these services automatically scaling up and down in response to demand (e.g.: Lambda, DynamoDB). This flexibility guarantees effective resource management and cost reduction.

Serverless architecture: With serverless computing services like Lambda and API Gateway, the business only pays for the computing time needed when executing functions and making API calls. As a result, there is no longer a need to maintain and pay for dedicated servers.

No up-front fees: There are no up-front costs or long-term obligations with these services. As a result, the financial risk is mitigated, and the business may start small and grow as necessary.

Managed services: By managing the infrastructure that supports these services for developers, Amazon lowers operating overhead. This enables businesses to concentrate on application development rather than infrastructure management. Most services used by this architecture is fully managed, namely API Gateway, Cognito, Rekognition, Elemental MediaConvert, S3, and DynamoDB.

Effective data storage: Amazon S3 and DynamoDB offer effective storage options that let the application store data more cheaply while enjoying great availability and durability.

VI. Budget Estimatation

The table below lists the services used, their configurations, and monthly costs. The calculations are based on the assumptions mentioned in the introduction. The costs are obtained by plugging the configuration settings into the AWS Pricing Calculator. All services are assumed to be in the Asia Pacific (Sydney) region.

Service	Configurations	Monthly costs
Route 53	Number of hosted zones: 1 IP (CIDR) blocks: 1,000	51.29 USD
CloudFront	Price for countries outside of Australia, such as Canada, Asia Pacific, Europe,... For example: USA Tiered price for: 256 GB 256 GB x 0.0850000000 USD = 21.76 USD Total tier cost = 21.76 USD (Data transfer out to internet from United States) Data transfer out to internet cost: 21.76 USD 256 GB x 0.02 USD = 5.12 USD (Data transfer out to origin from United States) Data transfer out to origin cost: 5.12 USD 2,500 requests x 0.000001 USD = 0.00 USD (HTTPS requests from United States) Requests cost: 0.00 USD 21.76 USD + 5.12 USD = 26.88 USD (Total cost United States)	393.00 USD

	CloudFront price United States (monthly): 26.88 USD	
Lambda	Standard Lambda Architecture: x86 Number of requests: 141,407 per month (30.41 days * 300 daily users * 15.5 view/upload requests) Duration of each request: 500ms Allocated memory: 128 MB Ephemeral storage: 512 MB Lambda@Edge Number of requests: 2,736,900 per month (30.41 days x 300 monthly users x 10 daily view requests x 20 photos/videos requested per view) Duration of each request: 500ms Allocated memory: 128MB	10.19 USD
API Gateway	API type: REST API Number of requests: 141,407 per month (30.41 days * 300 daily users * 15.5 view/upload requests) Caching: None	0.49 USD
S3	S3 tier: Standard Standard storage amount: 2280.75 per month (300 daily users * 5 daily photo uploads * 30.41 days * 2 MB per photo + 300 daily users * 0.5 daily video uploads * 30.41 days * 500MB per video) Data will be moved into S3 through PUT, COPY, POST requests. Average object size = 47.2MB ((2MB * 5 + 500MB * 0.5) / 5.5) PUT, COPY, POST, LIST requests into S3 = 50177 (300 users * 5.5 requests per user * 30.41 days) GET requests from S3 standard: 1824600 (300 users * 10 requests * 20 photos/videos retrieved per request * 30.41 days) Data returned or scanned by S3: 0 Inbound Data Transfer from Internet (free) Outbound data transfer to CloudFront (free)	58.10 USD
DynamoDB	DynamoDB on-demand capacity (standard table class) Data storage size(100GB) Number of write 50176.5 = 300 users per day * 30.41 days * 5.5 writes per days each user	29.25 USD

	Number of read: 91230 = 300 users per day * 30.41 days per month * 10 reads per day each user Monthly read cost (Monthly): 0.04 USD = 136,845.00 total read request units x 0.0000002846 USD DynamoDB data storage cost (Monthly): 28.50 USD Monthly write cost (Monthly): 0.71 USD = 501,765.00 total write request units x 0.0000014231 USD Total Monthly cost: 29.25 USD	
SNS	Number of requests/notifications: 5 photos per user * 0,5 video per user * 300 users monthly * 30.41 days per month = 50,177 requests/notifications 50,177 requests - 1000000 free tier requests = -949,823.00 billable SNS requests per month Max (-949823.000000 requests, 0 requests) = 0.00 requests 50,177 notifications x 0.00 USD = 0.00 USD (SQS Notifications cost) SNS Requests and Notifications cost (monthly): 0.00 USD	0.00 USD
SQS	Number of photo queue requests: 5 photos per user * 300 users monthly * 30 days monthly * 3 SQS requests per SNS requests = 135,000 requests Number of video queue requests: 0.5 video per user * 300 users monthly * 30 days monthly = 4,500 requests Number of queue requests: 135,000 + 4,500 = 139,500 requests 0.1395 requests per month x 1000000 multiplier for million = 139,500.00 total standard queue requests Tiered price for: 139500.00 requests 139500 requests x 0.0000000000 USD = 0.00 USD Total tier cost = 0.0000 USD (Standard queue requests cost) Total SQS cost (monthly): 0.00 USD	0.00 USD
Rekognition	Number of images processed with labels API calls per month: 45615 45615 = 300 users per day x 5 images per user per day x 30,41 days	54.47 USD

Elemental MediaConvert	Output usage: 22807 22807 = 300 users per day x 0.5 video per day x 5 minutes per video x 30,41 days	581.58 USD
Cognito	Number of Monthly Active Users(MAUs): 300 users monthly * 30.41 days monthly = 9,123 MAUs 9,123 MAUs x 0.75 SAML or OIDC federation requests = 6,842.25 SAML or OIDC federation MAU requests 6,842.25 SAML or OIDC federation MAUs - 50 free SAML or OIDC federation MAU requests per month = 6,792.25 billable SAML or OIDC federation MAU requests Max (6792.25 billable SAML or OIDC federation MAU requests, 0 minimum billable SAML or OIDC federation MAU requests) = 6,792.25 total billable SAML or OIDC federation MAU requests 6,792.25 MAUs x 0.015 USD = 101.88 USD (SAML or OIDC federation MAU requests) SAML or OIDC federation cost (monthly): 101.88 USD 9,123 MAUs - 50000 free MAU requests per month = -40,877.00 billable MAU requests Max (-40877.000000 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 Advanced security feature cost (monthly): 0 USD Cognito MAU cost (monthly): 101.88 USD	101.88 USD
Total		1324.24 USD

Fig. 5: A table showing the cost for this solution.

References

[1] AWS Web Services. *Amazon DynamoDB features* [Online]. Available: https://aws.amazon.com/dynamodb/features/

[2] AWS Web Services. *Amazon API Gateway Features* [Online]. Available: https://aws.amazon.com/api-gateway/features/

[3] AWS Web Services. *Amazon API Gateway pricing* [Online]. Available: https://aws.amazon.com/api-gateway/pricing/

[4] AWS Web Services. *Amazon S3 Security and Access Management* [Online]. Available: https://aws.amazon.com/s3/security/

[5] AWS Web Services. *Amazon Cognito Features*[Online]. Available: https://aws.amazon.com/cognito/details/

[6] AWS Web Services. *Amazon Cognito Pricing* [Online]. Available: https://aws.amazon.com/cognito/pricing/

[7] AWS Web Services. *Lambda execution environments* [Online]. Available: https://docs.aws.amazon.com/lambda/latest/operatorguide/execution-environments.html

[8] AWS Web Services. *Managing throughput capacity automatically with DynamoDB auto scaling* [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AutoScaling.html

[9] AWS Web Services. *Amazon API Gateway FAQs* [Online]. Available: https://aws.amazon.com/api-gateway/faqs/

[10] AWS Web Services. *Data protection in AWS Lambda* [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/security-dataprotection.html

[11] AWS Web Services. *AWS Lambda Pricing* [Online]. Available: https://aws.amazon.com/lambda/pricing/

[12] AWS Web Services. *What is AWS Fargate?* [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html

[13] AWS Web Services. *What Is a Key-Value Database?*[Online]. Available: https://aws.amazon.com/nosql/key-value/

[14] AWS Web Services. *Amazon DynamoDB*[Online]. Available: https://aws.amazon.com/dynamodb/

[15] AWS Web Services. *Amazon CloudFront Key Features* [Online]. Available: https://aws.amazon.com/cloudfront/features/

[16] AWS Web Services. *AWS Elemental MediaConvert FAQs* [Online]. Available: https://aws.amazon.com/mediaconvert/faqs/

[17] AWS Web Services. *AWSElemental MediaConvert Pricing* [Online]. Available: https://aws.amazon.com/mediaconvert/pricing/

[18] AWS Web Services. *Amazon Elastic Transcoder FAQs* [Online]. Available: https://aws.amazon.com/elastictranscoder/faqs/

[19] AWS Architecture Blog. *Get Started with Amazon S3 Event Driven Design Patterns* [Online]. Available: https://aws.amazon.com/blogs/architecture/get-started-with-amazon-s3-event-driven-design-patterns/

[20] AWS Web Services. *Differences between Amazon SQS, Amazon MQ, and Amazon SNS* [Online]. Available: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/sqs-difference-from-amazon-mq-sns.html

[21] AWS Web Services. *Setting up and using the Amazon Cognito hosted UI and federation endpoints* [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-pools-app-integration.html

[22] AWS Web Services. *Control access to a REST API using Amazon Cognito user pools as authorizer* [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-integrate-with-cognito.html

[23] AWS Web Services. *Quotas in Amazon Cognito* [Online]. Available: https://docs.aws.amazon.com/cognito/latest/developerguide/limits.html

[24] AWS Web Services. *Use API Gateway Lambda authorizers* [Online]. Available: https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html