# COS20019 Assignment 3

Luu Tuan Hoang
104180391
*Tutor class: 8.00 am Saturday*
*Swinburne University of Technology*
Hanoi, Vietnam
104180391@student.swin.edu.au

Le Nho Bach
103487884
*Tutor class: 8.00 am Saturday*
*Swinburne University of Technology*
Hanoi, Vietnam
103487884@student.swin.edu.au

***Abstract*— This report presents a comprehensive design for a scalable and highly available web application tailored for the Photo Album application, showing key components of the architecture, including the integration of various AWS services like Amazon S3, DynamoDB, Lambda, Elasticsearch, and Personalize. It will outline the data flow of essential functions, such as media file uploads, custom board creation, or personalized content recommendations The application's immense success demands further development to meet the growing demand and deliver personalized user experiences.**

***Index terms*—Cloud computing, web service, serverless, system architecture**

## I. Introduction

In today's business landscape, digital technologies play an increasingly vital role in driving innovation and transforming various aspects of operations. Among these technologies, cloud computing stands out as a powerful tool, providing business owners and developers with a new way to establish a strong online presence. Our application leverages cloud computing, using Amazon Web Services (AWS) to have better cost-effectiveness and operational efficiency. The traditional barriers between creating and managing a website on-premises and in the cloud have been bridged, resulting in a seamless and worry-free web operating experience.

One of the key advantages of AWS's services lies in its independence, allowing us to adopt a serverless/event-driven approach. This approach enhances both efficiency and flexibility in our web operations. Serverless computing eliminates the need to manage servers, enabling us to focus on building and deploying applications without the complexity of infrastructure management. Additionally, event-driven architecture ensures that tasks are triggered in response to specific events, promoting a more responsive and scalable system. Through AWS's versatile and scalable services, we have built a web application focusing on art exploration, discovery, and expression, offering users an immersive environment to share, curate, and appreciate captivating images/videos. To achieve this, we propose an architecture that allows the system to operate these functions:

- Upload media files: The system must be able to handle different types of images (JPG, JPEG, PNG, etc) and video (MP4, MOV, GIF, etc). These media files should be resized and reprocessed to standard formats such as JPEG and MP4 is a common practice for simplifying storage, playback, and delivery.
- Create a custom board: The user should be able to create a custom board where they can include and manage their favourite images/videos.
- Dashboard: The dashboard can be considered as the main page that shows new images/videos. These images/videos are chosen based on the user's habits and preferences.
- Search engine: The user should be able to find desired images/videos based on some keywords. These keywords can be stored in the database by the owner's titles or by our recognition system.

## II. Architecture Overview

### A. Key Components

This Photo Album web application architecture (Figure 1) is a scalable and efficient solution that allows users to upload, share, and discover new photos or videos based on their preferences. To ensure seamless user interactions and personalized content delivery, the architecture leverages various AWS services, including:

- Amplify: Builds and deploys the web front.
- CloudFront: Hosts the web application.
- Cognito: Handles user authentication and authorization (with Lambda), ensuring secure access to the platform's features. It provides users temporary credentials to access other AWS services or APIs in API Gateway. Cognito will issue a temporary AWS IAM credential to the resources.
- S3: Provides secure and reliable storage for images and videos uploaded by users. It also stores additional data required for media processing.
- DynamoDB: A NoSQL database that stores metadata, user interactions, and board information. It serves as the primary data store for the media-sharing platform.
- API Gateway: Offers secure APIs for the front-end application to interact with backend services.
- Lambda: Executes serverless functions to handle backend operations, including file uploads, board creation, and fetching personalized content.
- Elasticsearch Service (Amazon ES): A managed search and indexing service used for efficient search functionality, allowing users to find relevant images/photos based on metadata.
- Simple Queue Service (Amazon SQS): A message queue used for handling time-consuming tasks like media reprocessing. It helps decouple components and avoid overloading the application.
- Personalize: An AWS ML service used to create personalized content recommendations for users based on their interactions and preferences.
- EventBridge: A serverless event bus that enables scheduling and triggering of data preprocessing and model training tasks.

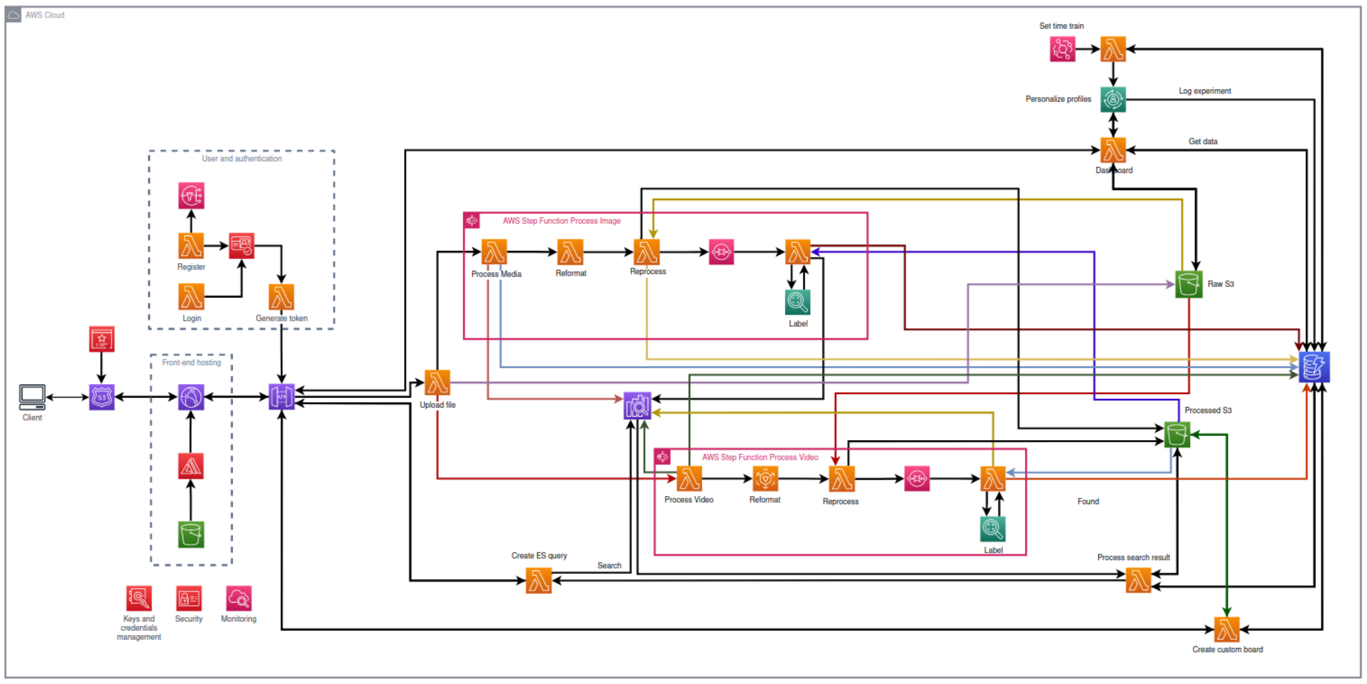


Figure 1: Architecture Overview

## *B. Data Flow*

Because the request is processed similarly from the user to the API gateway between different features, we demonstrate here not to make the duplication in the following features' descriptions. We utilized AWS Cognito, a managed identity service that handles registration and authentication to manages user sign-ups and authentication. During user authentication, Cognito provides temporary IAM credentials to access other AWS resources via API Gateway.

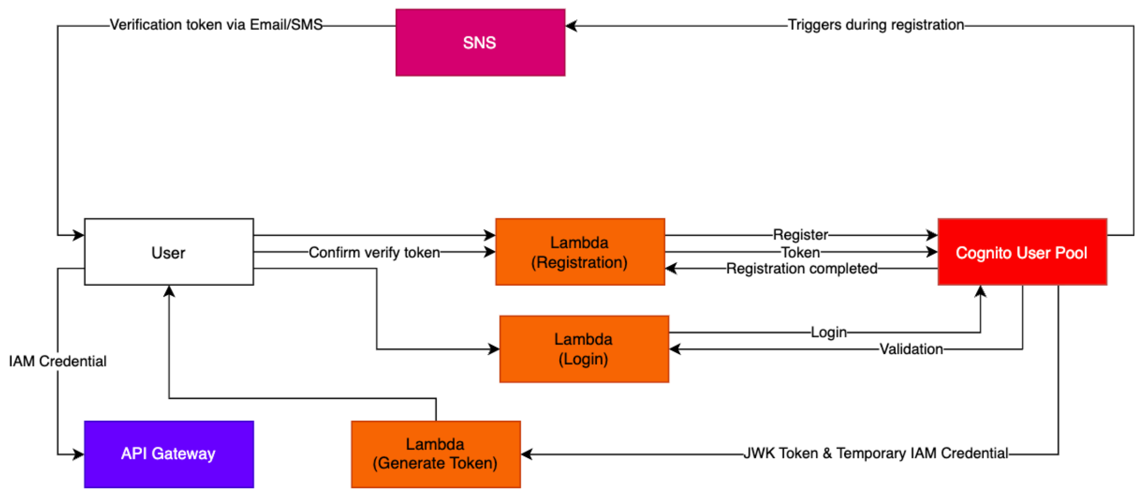


Figure 2: Securing API Gateway with AWS Cognito

The mentioned architecture allows the web application to smoothly operate four features: Upload media files, create a custom board, view the dashboard, and search for arts. Each feature has a different data flow with dissimilar relevant AWS components, which is described below:

- Uploading images/videos (Figure 3): The Amazon API Gateway forwards the upload request to an AWS Lambda function responsible for handling file uploads. The Lambda function `Upload File` receives the image or video file sent from the front-end application, and then recognizes it as an image or a video to trigger the corresponding step function group. The file then be saved by a Lambda Function `Process` in an S3 which is used to store raw data, while its metadata like upload data, user,... are stored in a DynamoDB. If the file is an image, it will be reformated by another Lambda function, else if it is a video, it will be transcoded by AWS MediaConvert. The next Lambda function reprocesses it before storing a new image in another S3. New information is also added to the DynamoDB for further use. After that, images/videos are queued to be labelled by a Rekognition module, which helps us to gain more data about the file. This data can also acknowledge NSFW content which leads to suitable restrictions. The output of the model is stored in the DynamoDB. In the whole process, all information that is stored in the DynamoDB is also sent to ElasticSearch, which offers the search engine of the web application.
- Dashboard (Figure 4): To create a Dashboard, the API Gateway will request the Lambda function `Dashboard`. This Lambda function will use an AWS Personalize module to find what kinds of art are the user caring about based on the user's history saved in the DynamoDB. It then based on the result finds the information on those files in the DynamoDB and retrieves it from the S3. On the other hand, an EventBridge is used to train the Machine Learning model in Personalize at some times in the day, which automates the training process. The trained data is handled by another Lambda function which retrieves data from the DB and preprocesses it.
- Search Engine (Figure 5): As we mentioned above, this engine is an application of AWS ElasticSearch. From the API Gateway, the Lambda function `Create Query` is called to create a query to push to ElasticSearch based on the given keywords from the user. Being given data during the upload process, the results from ElasticSearch can be used to trace the suitable data from DynamoDB. The images/videos then will be retrieved based on that data to return to the front and show images/videos the user wants to find.
- Custom Board (Figure 6): This is quite a simple function. When the user sends a request to create a custom board, the Lambda function `Custom Board` is triggered to retrieve metadata and media files to show to the user. Then the data of the chosen images/videos will be added to a table in the DynamoDB to save the information of that custom board. Whenever the users want to see their created board, the Lambda function will read data from the corresponding table in DynamoDB and retrieve media files to show them. In the future, this function structure can also be implemented to allow users to share or save media files.

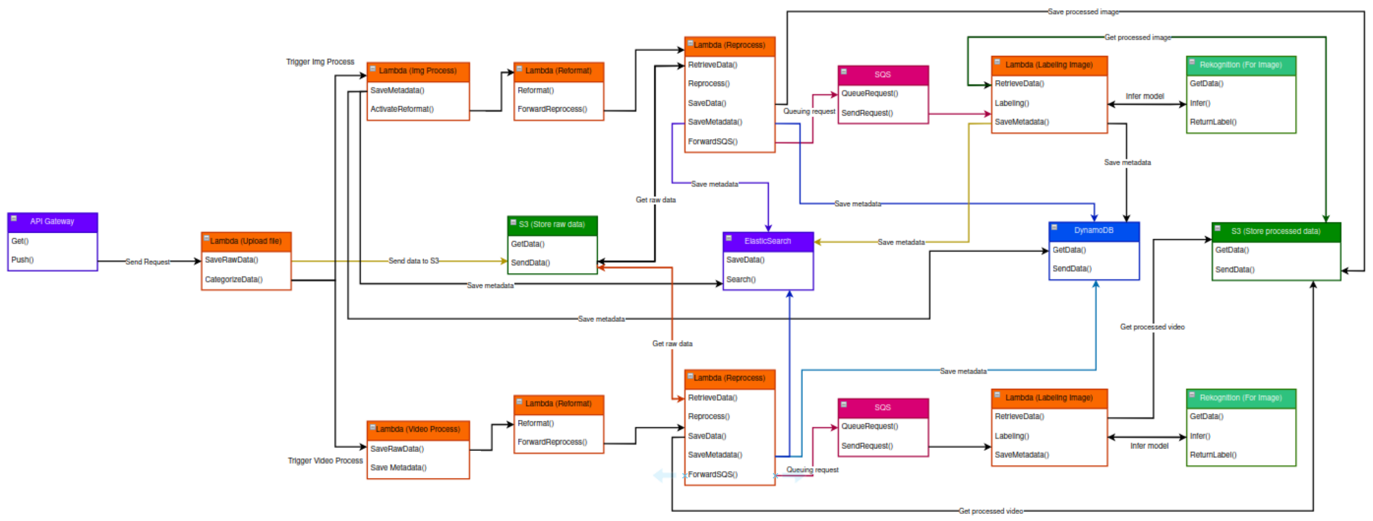


Figure 3: Upload Media File Data Flow

## III. Service Selection and Justification

### A. *Managed Cloud Services*

The preference of the company for managed cloud services aligns well with the objective of reducing its in-house administration requirements. The business can make use of an entirely managed cloud service offered by Amazon Web Services by leveraging AWS Lambda as the main computing resource, AWS S3 (Simple Storage Service) as the storage option for images and media, AWS DynamoDB as a database for storing metadata and many other services to accommodate the requirements of the website from the company.

### B. *Scalability to Handle Growth*

The architecture leverages various AWS services, such as Amazon DynamoDB, Amazon S3, AWS Lambda, Amazon Elasticsearch, Amazon Personalize, and Amazon SQS. These services are designed to be highly scalable and can automatically handle increased workloads. As demand grows, these services can dynamically scale to adapt to the increased traffic, ensuring that the system remains responsive and performant [1], [2], [3], [4], [5], [6]. Furthermore, services like Amazon DynamoDB and Amazon Elasticsearch can automatically adjust their provisioned capacity to handle increased traffic [1], [4]. For example, Amazon DynamoDB can adjust read and write capacity units based on the actual request rate, while Amazon Elasticsearch can scale its nodes to handle increased search and indexing.

### C. *Adopting Serverless/Event-Driven Solution*

The Photo Album application migrate from a server approach to a serverless/event-driven approach. The application benefits from automatic scaling based on demand by using AWS Lambda as the architecture's core component, ensuring that resources are used effectively. Lambda functions are triggered in response to media uploads to Amazon S3, which promotes seamless integration and lowers administrative burden. With no need to worry about the behind infrastructure administration, this serverless solution enables quick feature launch and extensibility in the future [7].

The event-driven model offers fault tolerance and resilience, as AWS Lambda operates in a highly available and fault-tolerant manner [7]. In case of hardware failures, AWS automatically handles recovery, ensuring a reliable and high-availability application. The on-demand billing model of Lambda aligns with the company's cost optimization strategy, charging only for the actual computing resource used, thereby providing cost control. The integrated ecosystem of AWS services allows smooth interactions between services, enhancing the overall responsiveness of the application.

### D. *Cost-Effective Database Option*

Changing from a traditional relational database to Amazon DynamoDB, a fully managed NoSQL database service offered by AWS, would be a suitable option to address the cost-effectiveness of the database for the Photo Album application, as DynamoDB is designed to be extremely cost-efficient for workloads of any scale [8].

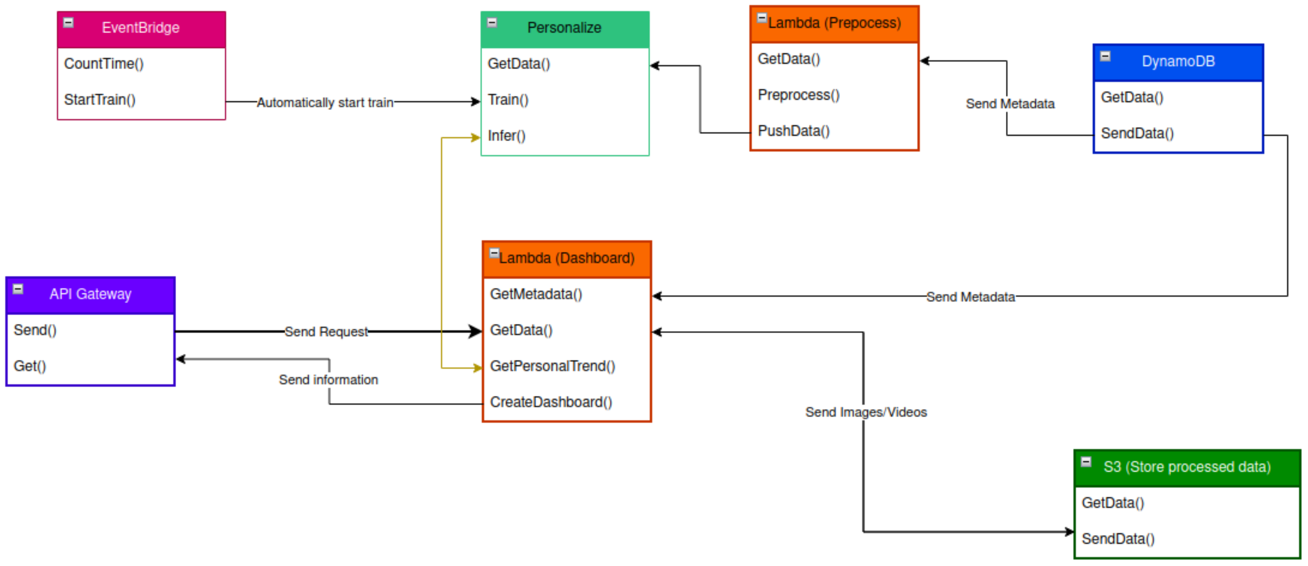


Figure 4: Dashboard Data Flow

There are two pricing options available for Amazon DynamoDB: on-demand capacity mode and provisioned capacity mode. Given the requirement that the company is unsure about the growth of demand in the future for the Photo Album application, and that the recent growth has been doubling every 6 months, the most suitable pricing option for Amazon DynamoDB would be the On-Demand Capacity Mode.

On-Demand Capacity Mode is intended for workloads with variable demand and unpredictable traffic patterns [9]. Provisioning read and write capacity units in advance is not required when using this mode. Instead, you only pay for the read and write requests you make, and DynamoDB automatically scales the provisioned capacity based on the actual usage.

### E. *Global Response Times Improvement*

The Photo Album application's global response times can be effectively improved by integrating Amazon CloudFront and Amazon Route 53 into the architecture. CloudFront is a content delivery network (CDN) that distributes and caches material to edge locations that are carefully chosen all across the globe. Users from different regions can access content from neighbouring servers, speeding up response times and minimizing data transfer times, by caching commonly accessed assets, such as images and videos, at these edge locations [10]. In order to further optimize data delivery and reduce latency for users in various regions, CloudFront also enables content compression.

Amazon Route 53, a global DNS service that works in conjunction with CloudFront, automatically directs users to the closest CloudFront edge location based on their location using geolocation-based routing, which lowers the number of network hops and improves responsiveness [11]. Route 53 also provides latency-based routing, which routes users to the location with the lowest latency, enhancing overall performance for users outside Australia [12]. Users gain from faster content delivery and shorter response times because of the integration of CloudFront and Route 53, regardless of where they are in the world.

### F. *Handling Imagge, Video Media and Media Processing*

To meet the requirements for automatically creating various versions of uploaded media and ensuring scalability, flexibility, and efficiency in processing, the Photo Album application can implement a serverless/event-driven architecture by leveraging AWS services such as AWS Lambda, Amazon S3, and Amazon SQS.

1) *Event-Driven Media Processing:* When a user uploads a media file to the S3 bucket, an event trigger can be set up to automatically initiate the media processing workflow. AWS Lambda functions can be used to handle the processing tasks, such as generating low-resolution versions, or video transcoding. Once the transformations are complete, the resulting media will be stored in another S3 bucket specified for processed media files, while metadata is stored in a DynamoDB which allows retrieval later.

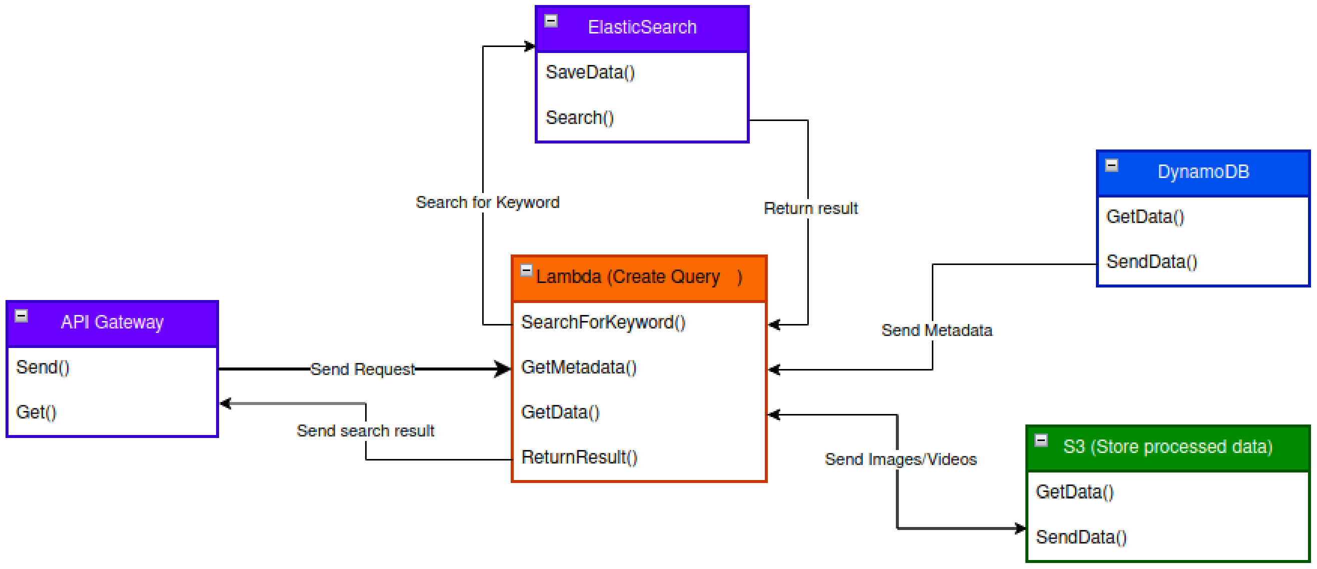


Figure 5: Search Engine Data Flow

2) *Extensibility with AI Integration:* To prepare for future requirements, the architecture can be designed to easily integrate AI-based image recognition services. By adding Amazon Rekognition to Lambda functions, the application can automatically identify tags and metadata in photos as well as remove explicit media, enhancing the user experience and providing valuable insights. We also utilized AWS Step Functions for media automated processing so extending the AI pipelines will be much easier in the future.

3) *Platform Flexibility:* The architecture can be designed to run processing tasks on the most suitable platform based on the nature of the task. In the architecture, Lambda can handle lightweight tasks, while more compute-intensive tasks like video transcoding can be offloaded to AWS Elemental MediaConvert. This approach optimizes cost and performance within the AWS ecosystem.

4) *Scalability and Decoupling:* To prevent overloading the application during reprocessing tasks, a decoupled design can be implemented using Amazon SQS. Lambda functions can place media transformation jobs on an SQS queue, and multiple "worker" nodes, specialized for specific tasks, can process these jobs asynchronously. This ensures efficient resource utilization and effective load balancing, preventing bottlenecks and providing scalability during peak demands.

## IV. Alternative Considerations

### A. *Serverless FaaS and CaaS*

AWS provides two technology options for building modern serverless applications: AWS Lambda and AWS Fargate. AWS Lambda allows running code without server management, offering automatic scaling, and event-driven execution. AWS Fargate allows deploying containers without managing infrastructure, providing more control over resource allocation and scaling compared to Lambda. We decided to use Lambda as the core of the website serverless architecture because of the following reasons.

1) *Trigger Type Preference:* Lambda natively supports various triggers to initiate work in AWS, such as SQS subscriptions, S3 events, and DynamoDB events. However, for scenarios where the primary requirement is to solely respond to API requests, AWS Fargate is a more suitable option [13]. It is important to note that the company prefers an event-driven approach, making Lambda a viable choice for handling the various triggers effectively.

2) *Scalability and Cost-Effectiveness:* As the company expects the application's demand to double every 6 months, AWS Lambda's automatic scaling aligns perfectly with unpredictable growth. On-demand billing ensures cost optimization by charging only for actual compute time, ideal for handling variable media processing workloads without paying extra for idle time.

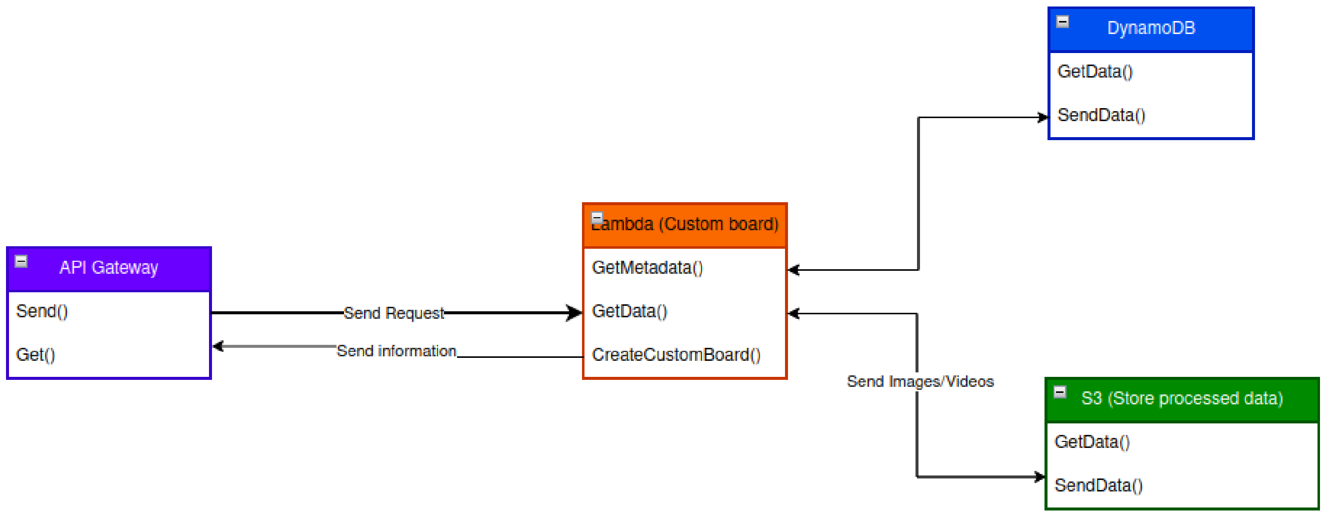


Figure 6: Custom Board Data Flow

### B. *Relational database and Non-relational database*

Original architecture makes use of RDS, a relational database service from AWS. As the company wants to make the shift to serverless architecture and is exploring cost-effective database options, there are two suitable AWS services: Aurora Serverless and DynamoDB.

1) *Scalability and Performance:* DynamoDB is a fully managed NoSQL database service that automatically scales based on demand, allowing it to handle rapidly increasing workloads and traffic. With the application's expected growth of doubling every 6 months, DynamoDB's auto-scaling capability ensures seamless performance without the need for manual intervention. In contrast, Aurora Serverless requires manual configuration and scaling, which may not be as efficient for handling unpredictable growth.

2) *Cost-Effectiveness:* DynamoDB offers a pay-as-you-go pricing model, charging only for the provisioned read and write capacity units consumed. This allows for better cost control and budget management, especially for a fast-growing application with fluctuating demands. Aurora Serverless, on the other hand, requires provisioning a minimum capacity, which may lead to unnecessary costs during low-traffic periods.

In conclusion, choosing DynamoDB over Aurora Serverless is justified by its scalability, cost-effectiveness, and low administration overhead. These advantages align well with the application's requirements for handling rapid growth, minimizing operational efforts, and maintaining data consistency and reliability.

### C. *Migration from Elastic Transcoder to Elemental MediaConvert*

AWS Elemental debuted five new media processing and delivery services at re: Invent 2017. One of these services, AWS Elemental MediaConvert, shares the same functionalities with Amazon Elastic Transcoder to provide file-based video-converting functionality.

1) *Greater Flexibility:* The migration to AWS MediaConvert is justified as it offers more flexibility in configuring video encoding jobs compared to Elastic Transcoder [14]. With Elastic Transcoder, users are limited to predefined presets, which restricts customization options for the output. On the other hand, AWS MediaConvert enables fine-tuning of encoding settings, allowing adjustments to parameters like bit rate, resolution, frame rate, and color space. This level of customization ensures the Photo Album application can deliver media content with the best output quality, tailored to specific requirements and user preferences.

2) *Faster Encoding:* AWS MediaConvert provides faster video encoding compared to Elastic Transcoder [15]. Its distributed architecture allows for parallel processing of video encoding jobs, enabling simultaneous encoding of multiple videos. This significantly reduces the overall time required to encode content, improving the application's efficiency and responsiveness. As the demand for media processing increases with the application's growth, the faster encoding capability of MediaConvert ensures the timely delivery of media content to users, even during peak periods.

3) *Cost-Effectiveness:* In terms of cost, AWS Elemental MediaConvert follows an hourly billing model based on the duration of the output, whereas AWS Elastic Transcoder charges based on the input duration, which might result in higher costs for longer videos or higher-resolution content [14].

Therefore, we decide to migrate from Elastic Transcoder to AWS Elemental MediaConvert for the benefits that Elemental MediaConvert brings to the Photo Album application.

## V. Architecture Evaluation

### A. Performance Pillar

1) *AWS Lambda:* Lambda enables serverless computing, providing a highly scalable and event-driven execution environment. Lambda allows the application to respond to events, such as media uploads with low latency and high throughput. By auto-scaling based on demand, Lambda ensures that resources are efficiently utilized to handle varying workloads, optimizing performance.

2) *AWS Elemental MediaConvert:* MediaConvert offers faster video encoding with its distributed architecture, enabling parallel processing of video encoding jobs. This accelerates the media processing time, ensuring quick delivery of media content to users and enhancing the overall application performance.

3) *Amazon SQS:* SQS is a fully managed message queuing service that decouples components in the architecture, enabling asynchronous processing. By buffering messages and decoupling the architecture, SQS ensures that media processing tasks, such as image processing or video transcoding, can be performed without overloading the application, enhancing performance.

4) *Amazon DynamoDB:* DynamoDB is a high-performance, fully managed NoSQL database that can handle large-scale and highly concurrent workloads. It provides single-digit millisecond response times, allowing the application to retrieve and store media metadata quickly, optimizing the performance of data access operations. DynamoDB auto scaling is enabled by default to ensure scalability of the website.

5) *Amazon CloudWatch:* CloudWatch provides real-time monitoring and visibility into the application's performance metrics and resource utilization. By closely monitoring Lambda function invocations, MediaConvert job completion times, SQS queue metrics, and DynamoDB performance, the architecture can proactively identify performance bottlenecks and optimize resources.

### B. Security Pillar

1) *Identity and Access Management (IAM):* By implementing IAM roles and policies, the architecture ensures that only authorized entities can access specific resources, reducing the risk of unauthorized access and potential security breaches.

2) *Amazon CloudFront:* CloudFront acts as a Content Delivery Network (CDN) and provides a layer of security by mitigating Distributed Denial of Service (DDoS) attacks and offering SSL/TLS encryption for data in transit. It helps in reducing the risk of direct attacks on the application's origin server.

3) *Amazon CloudHSM:* CloudHSM offers hardware-based key storage and management, providing a secure environment for cryptographic operations. This service protects sensitive data and cryptographic keys, which is critical for ensuring the confidentiality and integrity of the application's data.

4) *Encryption:* The architecture employs encryption in transit and at rest. SSL/TLS encryption is enabled through CloudFront for data transmitted between the application and users, while S3 bucket policies and IAM roles control access to encrypted data in S3.

5) *Logging and Monitoring:* Amazon CloudWatch enables comprehensive logging and monitoring of the application's activities and events, providing valuable insights into potential security threats or anomalous behaviour, and allowing for quick identification and response.

### C. Reliability Pillar

1) *AWS Lambda:* Lambda functions are highly reliable as AWS automatically manages the underlying infrastructure, ensuring that the functions are automatically scaled and distributed across multiple Availability Zones (AZs). This inherent fault tolerance of Lambda ensures that the application can continue to function even if there are hardware failures or disruptions in a specific AZ.

2) *AWS Elemental MediaConvert:* MediaConvert is designed to be highly available and fault-tolerant. It uses a distributed architecture to process video encoding jobs in parallel, and AWS automatically handles recovery from any hardware failures or issues, ensuring continuous operation.

3) *Amazon SQS:* SQS provides reliable message queuing by replicating messages across multiple AZs. This ensures that even if a message is lost or a queue is unavailable in one AZ, the message can still be processed from another AZ, increasing the overall reliability of media processing.

4) *Amazon DynamoDB:* DynamoDB is a fully managed service that replicates data across multiple AZs to ensure data durability and availability. It provides automatic backups and point-in-time recovery, enabling the architecture to recover from data loss or corruption incidents.

5) *Amazon CloudWatch:* CloudWatch provides proactive monitoring and alarming for the entire architecture. By setting up alarms and metrics to track the health and performance of the components, CloudWatch helps detect and respond to potential issues, ensuring the architecture's reliability

### *D. Cost Optimization Pillar*

1) *AWS Lambda:* Lambda follows a on-demand pricing model, where you are billed based on the number of requests and the time your code executes. This serverless approach eliminates the need for provisioning and managing servers, reducing operational costs. Additionally, Lambda automatically scales based on demand, ensuring resources are used efficiently and cost-effectively.

2) *AWS Elemental MediaConvert:* MediaConvert offers on-demand pricing based on the duration of video output. With its parallel processing capabilities, MediaConvert completes video encoding jobs faster, reducing the overall cost of media processing.

3) *Amazon SQS:* SQS provides a cost-effective way to decouple the architecture. You pay for the number of requests and storage used, making it a cost-efficient message queuing service for media processing tasks.

4) *Amazon DynamoDB::* DynamoDB offers a On-Demand Capacity Mode pricing model based on the number of reads and writes to the database. Its fully managed nature eliminates the need for hardware provisioning and maintenance, reducing operational costs.

5) *Amazon CloudWatch:* CloudWatch provides cost and usage reports that help track resource utilization and identify opportunities for cost optimization. By monitoring and analyzing usage patterns, the architecture can identify underutilized resources and implement cost-saving measures.

## VI. Cost

We assume our web application will have about 10,000 daily users. Therefore, the following is the estimated capacity and cost for 10000 DAU.

### *A. Capacity Planning*

1) *Storage Capacity:*

- Average image size: 4MB
- Average processed image size: 1MB
- Average video size: 200MB
- Average processed video size: 100MB
- Each user uploads 5 images and 1 video a day
- Estimated number of media files: 3000000/month
- Maximum S3 capacity = 150TB/month
- Maximum S3 request return = 100TB/month

2) *Network Bandwith:*

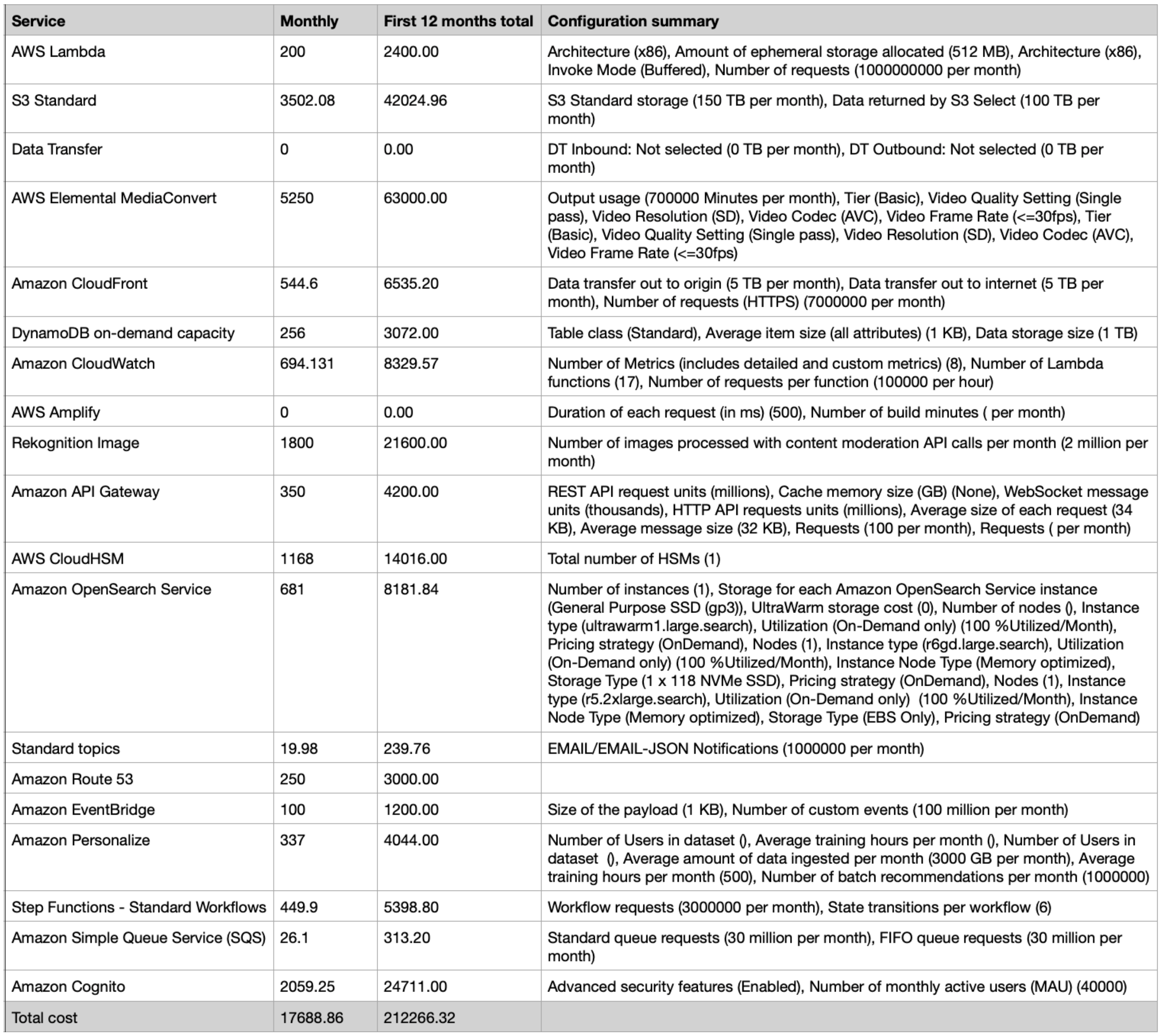
- Data transfer out to origin and Internet: 10TB/m
- Assume each media file needs 3 HTTPS request
- Maximum number of requests (HTTPS): 7000000
- 100000000 total REST API requests/month
- 300000000 standard queries/month, 100000000 geo DNS queries/month, 100000000 latency-based queries/month

3) *Computing Resource:*

- Each user requests Lambda execution 2000 times/day
- Average duration of each Lambda request: 5000ms
- Average duration of each video: 2 mins
- Maximum 700000 minutes of Elemental MediaConvert
- Number of images processed with content moderation API calls: 2 million/month
- r5.2xlarge.search for OpenSearch
- Rekognition: Average amount of data ingested: 3000 GB/month), Average training: 500 hours/month, Number of batch recommendations 1000000/per month
- Estimated 3000000 executions of media files.

4) *Database capacity:*

- Metadata size per media file: 25 bytes
- Maximum database size: 1TB



| Service | Monthly | First 12 months total | Configuration summary |
|---|---|---|---|
| AWS Lambda | 200 | 2400.00 | Architecture (x86), Amount of ephemeral storage allocated (512 MB), Architecture (x86), Invoke Mode (Buffered), Number of requests (1000000000 per month) |
| S3 Standard | 3502.08 | 42024.96 | S3 Standard storage (150 TB per month), Data returned by S3 Select (100 TB per month) |
| Data Transfer | 0 | 0.00 | DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month) |
| AWS Elemental MediaConvert | 5250 | 63000.00 | Output usage (700000 Minutes per month), Tier (Basic), Video Quality Setting (Single pass), Video Resolution (SD), Video Codec (AVC), Video Frame Rate (<=30fps), Tier (Basic), Video Quality Setting (Single pass), Video Resolution (SD), Video Codec (AVC), Video Frame Rate (<=30fps) |
| Amazon CloudFront | 544.6 | 6535.20 | Data transfer out to origin (5 TB per month), Data transfer out to internet (5 TB per month), Number of requests (HTTPS) (7000000 per month) |
| DynamoDB on-demand capacity | 256 | 3072.00 | Table class (Standard), Average item size (all attributes) (1 KB), Data storage size (1 TB) |
| Amazon CloudWatch | 694.131 | 8329.57 | Number of Metrics (includes detailed and custom metrics) (8), Number of Lambda functions (17), Number of requests per function (100000 per hour) |
| AWS Amplify | 0 | 0.00 | Duration of each request (in ms) (500), Number of build minutes ( per month) |
| Rekognition Image | 1800 | 21600.00 | Number of images processed with content moderation API calls per month (2 million per month) |
| Amazon API Gateway | 350 | 4200.00 | REST API request units (millions), Cache memory size (GB) (None), WebSocket message units (thousands), HTTP API requests units (millions), Average size of each request (34 KB), Average message size (32 KB), Requests (100 per month), Requests ( per month) |
| AWS CloudHSM | 1168 | 14016.00 | Total number of HSMs (1) |
| Amazon OpenSearch Service | 681 | 8181.84 | Number of instances (1), Storage for each Amazon OpenSearch Service instance (General Purpose SSD (gp3)), UltraWarm storage cost (0), Number of nodes (), Instance type (ultrawarm1.large.search), Utilization (On-Demand only) (100 %Utilized/Month), Pricing strategy (OnDemand), Nodes (1), Instance type (r6gd.large.search), Utilization (On-Demand only) (100 %Utilized/Month), Instance Node Type (Memory optimized), Storage Type (1 x 118 NVMe SSD), Pricing strategy (OnDemand), Nodes (1), Instance type (r5.2xlarge.search), Utilization (On-Demand only) (100 %Utilized/Month), Instance Node Type (Memory optimized), Storage Type (EBS Only), Pricing strategy (OnDemand) |
| Standard topics | 19.98 | 239.76 | EMAIL/EMAIL-JSON Notifications (1000000 per month) |
| Amazon Route 53 | 250 | 3000.00 | |
| Amazon EventBridge | 100 | 1200.00 | Size of the payload (1 KB), Number of custom events (100 million per month) |
| Amazon Personalize | 337 | 4044.00 | Number of Users in dataset (), Average training hours per month (), Number of Users in dataset (), Average amount of data ingested per month (3000 GB per month), Average training hours per month (500), Number of batch recommendations per month (1000000) |
| Step Functions - Standard Workflows | 449.9 | 5398.80 | Workflow requests (3000000 per month), State transitions per workflow (6) |
| Amazon Simple Queue Service (SQS) | 26.1 | 313.20 | Standard queue requests (30 million per month), FIFO queue requests (30 million per month) |
| Amazon Cognito | 2059.25 | 24711.00 | Advanced security features (Enabled), Number of monthly active users (MAU) (40000) |
| Total cost | 17688.86 | 212266.32 | |

Figure 7: Cost Estimation

### B. *Estimated Cost*

From the previous capacity and use estimation, we propose a cost prediction in Figure 7 that shows monthly cost, first-12-month cost, and configuration regarding services. The used currency is USD.

## VII. Conclusion

In conclusion, our AWS serverless/event-driven architecture provides a scalable and cost-effective solution for our platform. With AWS Lambda, EventBridge, and Amazon SQS handling real-time events fluently, we can focus on delivering superior value to our users, fostering an engaging and inspiring community. All requirements are met efficiently with high-descriptive justification and evaluation.

## References

[1] D. Yoder, and S. Shriver, "Amazon dynamodb auto scaling: performance and cost optimization at any scale," Amazon Web Services, 2019. [Online]. Available: https://aws.amazon.com/blogs/database/amazon-dynamodb-auto-scaling-performance-and-cost-optimization-at-any-scale/

[2] AWS, "Horizontal scaling and request parallelization for high throughput - best practices design patterns: optimizing amazon s3 performance," docs.aws.amazon.com. Accessed: Aug. 5, 2023. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/s3-optimizing-performance-best-practices/horizontal-scaling-and-request-parallelization-for-high-throughput.html

[3] AWS, "Lambda function scaling - aws lambda," docs.aws.amazon.com. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/lambda-concurrency.html

[4] D. Parmaksız, "How insider learned to scale a production grade elasticsearch cluster on aws | aws partner network (apn) blog," aws.amazon.com, 2020. Accessed: Aug. 5, 2023. [Online]. Available: https://aws.amazon.com/blogs/apn/how-insider-learned-to-scale-a-production-grade-elasticsearch-cluster-on-aws/

[5] AWS, "Amazon personalize increases limits for all customers, simplifying large scale deployments," Amazon Web Services, Inc., 2022. Accessed: Aug. 5, 2023. [Online]. Available: https://aws.amazon.com/about-aws/whats-new/2022/09/amazon-personalize-increases-limits-customers-large-scale-deployments/

[6] J. Barr, "Scaling with amazon sqs | aws news blog," aws.amazon.com, 2012. Accessed: Aug. 5, 2023. [Online]. Available: https://aws.amazon.com/blogs/aws/scaling-with-amazon-sqs/

[7] A. Dobosz, "Serverless vs. server computing: detailed comparison," www.rumblefish.dev, 2023. Accessed: Aug. 4, 2023. [Online]. Available: http://www.rumblefish.dev/blog/post/serverless-vs-server-computing-detailed-comparison/

[8] S. Sivasubramanian, "Amazon dynamodb: a seamlessly scalable non-relational database service," in *Proc. 2012 ACM SIGMOD Int. Conf. Manage. Data*, 2012, pp. 729–730.

[9] A. DeBrie, "Dynamodb on-demand: when, why and how to use it in your serverless applications," serverless.com, 2016. Accessed: Aug. 4, 2023. [Online]. Available: https://www.serverless.com/blog/dynamodb-on-demand-serverless

[10] O. Moses, "Hosting a static website on aws using s3, cloudfront, and route53, with just 7 steps," DEV Community, 2023. Accessed: Aug. 4, 2023. [Online]. Available: https://dev.to/aws-builders/guide-to-hosting-a-static-website-on-aws-using-s3-cloudfront-and-route53-with-just-7-steps-220b

[11] E. Shipton, "What is geoproximity vs geolocation? (2022) | abstractapi," www.abstractapi.com. Accessed: Aug. 4, 2023. [Online]. Available: https://www.abstractapi.com/guides/geoproximity-vs-geolocation

[12] A. Lovan, "Using latency-based routing with amazon cloudfront for a multi-region active-active architecture | networking & content delivery," aws.amazon.com, 2022. Accessed: Aug. 4, 2023. [Online]. Available: https://aws.amazon.com/blogs/networking-and-content-delivery/latency-based-routing-leveraging-amazon-cloudfront-for-a-multi-region-active-active-architecture/

[13] N. Pantic, "Serverless showdown: fargate vs. lambda | serverless guru," www.serverlessguru.com, 2021. Accessed: Aug. 4, 2023. [Online]. Available: http://serverlessguru.com/blog/serverless-showdown-fargate-vs-lambda

[14] J. Johnson, "How to migrate workflows from amazon elastic transcoder to aws elemental mediaconvert | aws for m&e blog," aws.amazon.com, 2018. Accessed: Aug. 4, 2023. [Online]. Available: https://aws.amazon.com/vi/blogs/media/how-to-migrate-workflows-from-amazon-elastic-transcoder-to-aws-elemental-mediaconvert/

[15] A. Joshi, "Aws mediaconvert: the powerful alternative to elastic transcoder," CloudThat Resources, 2023. Accessed: Aug. 4, 2023. [Online]. Available: https://www.cloudthat.com/resources/blog/aws-mediaconvert-the-powerful-alternative-to-elastic-transcoder