Swinburne University of Technology

COS20019 - Cloud Computing Architecture

Assignment 3 Report

Student Name: Nguyen Duy Quang
Student ID: 104992981
Class: COS20019 (Friday Morning)
Swinburne VietNam - HCM Campus
104992981@student.swin.edu.au

Student Name: Tran Thien Thao Vy
Student ID: 104991221
Class: COS20019 (Friday Morning)
Swinburne VietNam - HCM Campus
104991221@student.swin.edu.au

I. INTRODUCTION

This report presents a detailed architectural design for the Photo Album application, addressing the challenges of rapid user growth and increasing traffic. To ensure scalability and high availability, the design adopts a serverless, event-driven architecture using AWS managed services. This approach minimizes the need for in-house systems administration while enhancing performance and reducing costs.

The report explores alternative architectural solutions and justifies the chosen design based on business requirements, focusing on resolving current performance issues, optimizing global response times, and enabling efficient media processing. The proposed system streamlines media uploads and transformations, ensuring responsiveness and compatibility with various media formats.

To illustrate the solution, the report includes an architectural diagram that maps interactions between cloud services and provides detailed descriptions of each component's role. UML collaboration diagrams further demonstrate how users and services interact during media uploads and processing.

By prioritizing performance, scalability, reliability, security, and cost-effectiveness, the proposed architecture aims to meet current demands while preparing the application for future growth and innovation.

II. BUSINESS SCENARIO

1. Where possible the company would like to use managed cloud services to minimize the need for in-house systems administration. Photo and other media will be stored in AWS S3.

Solution:
To minimize operational overhead and enhance reliability, the company should adopt AWS S3 for media storage. AWS S3 is a managed object storage service offering unmatched scalability, data availability, security, and performance. By using S3, the company eliminates the need to manage physical infrastructure for storing and retrieving media files, as AWS takes full responsibility for tasks such as provisioning, scaling, and maintaining the service. Additionally, S3's high durability ensures that user media is safeguarded against loss.

Implementation:

A. **Bucket Creation and Configuration:** Create an S3 bucket specifically for media storage, enabling versioning to maintain historical file copies and protect against accidental overwrites.
B. **Access Policies:** Apply bucket policies to ensure secure access, with roles assigned to applications or users based on the principle of least privilege.
C. **Lifecycle Policies:** Enable lifecycle rules to automatically transition objects to lower-cost storage classes such as S3 Intelligent-Tiering or S3 Glacier for archiving.
D. **Security Enhancements:** Activate encryption at rest using AWS-managed keys (SSE-S3) or customer-managed keys (SSE-KMS) and encryption in transit using SSL/TLS.

2. The company is not sure how demand for its application will grow in the future but over recently it has been doubling every 6 months. It expects this trend will continue for the next 2 or 3 years at least and it wants the architecture to be able to cope with this growth.

Solution:
To handle exponential growth in demand efficiently, the

system can adopt a Load Balancer-based architecture with Auto Scaling EC2 instances. This approach ensures both horizontal scalability and high availability. An Elastic Load Balancer (ELB) dynamically distributes incoming traffic across multiple EC2 instances, improving fault tolerance by redirecting requests to healthy instances only. Auto Scaling further enhances this solution by automatically adding or removing EC2 instances based on predefined metrics, such as CPU utilization, ensuring cost-efficiency and optimal performance.

Implementation:

A. **Elastic Load Balancer (ELB):** Deploy an ELB to act as the entry point for all user requests. The load balancer monitors the health of registered EC2 instances and only routes traffic to healthy instances.

B. **Auto Scaling Group:** Create an Auto Scaling group that monitors application performance and adjusts the number of EC2 instances based on real-time traffic patterns. For example:
   - **Use AmazonCloudWatch** to track CPU utilization and trigger scale-out events if utilization exceeds 60%.
   - **Trigger scale-in events** when utilization drops below 30% to reduce costs during low-demand periods.

C. **EC2 InstanceOptimization:**
Use compute-optimized instances (e.g., c5.large) for better handling of high workloads.

D. **Multi-AZ Deployment:** Distribute instances across multiple Availability Zones (AZs) to improve fault tolerance and ensure the application remains available even during outages in one AZ.

E. **Monitoring and Alerts:** Implement CloudWatch alarms to notify administrators of unusual traffic patterns or resource bottlenecks.

3. The current system EC2 instances are running on t2.micro. The compute capacity is regularly exceeding the 80% performance limit with 6 instances running. The desired load needs to decrease to between 50 and 60%. Ignore this requirement if your solution does not involve EC2.

Solution:

Optimizing compute performance can be achieved by either transitioning to serverless computing or upgrading the existing EC2 instances to a higher-performing instance type such as t3 or c5 instances, which provide better processing power for CPU-intensive workloads.

Implementation:

A. **Optimizing EC2 Instances:** select a more suitable instance type, such as compute-optimized instances (e.g., c5.large), and implement horizontal scaling using Elastic Load Balancing (ELB) to distribute traffic effectively.

B. **Performance Monitoring:** Use Amazon CloudWatch to track resource utilization and trigger alerts when predefined thresholds are breached, ensuring proactive management.

4. The company would like to adopt a serverless/event-driven solution.

Solution:

Adopting AWS Lambda functions, in conjunction with event sources such as S3, SNS, and SQS, offer a robust, decoupled solution for handling media uploads and processing tasks.

Implementation:

A. **Event Source Configuration:** Configure the S3 bucket to automatically trigger Lambda functions when new media files are uploaded. This ensures immediate processing without manual intervention.

B. **Decoupled Processing:** Use Amazon SQS to manage a queue of processing tasks, allowing Lambda functions to handle these tasks asynchronously. For example, tasks like generating thumbnails, resizing images, or transcoding videos can be queued and processed in parallel by specialized Lambda functions.

C. **Extensibility:** Design the architecture to accommodate future extensions, such as integrating AI-based photo tagging using AWS Rekognition.

5. The relational database is relatively slow and costly to run. Given the simple table structure, the company would like to explore more cost-effective options.

Solution:

Given the company's requirements for a simple table structure, a NoSQL database like Amazon DynamoDB is an optimal choice. DynamoDB offers high availability, low latency, and seamless scalability while eliminating the overhead associated with managing relational database systems.

Implementation:

A. **Migration Strategy:** Transfer existing relational data to DynamoDB, ensuring the data model leverages DynamoDB's schema-less nature for optimal performance. For instance, the database can be designed with partition and sort keys to efficiently retrieve and query data.

B. **Global Tables for Geographic Availability:** Enable DynamoDB Global Tables to replicate data across multiple AWS regions, reducing latency for global users.

C. **Capacity Management:** Use on-demand capacity mode to handle varying workloads without manual throughput provisioning.

6. The application has had wide uptake around the world but response time in countries other than Australia has been relatively slow. Global response times need to be improved.

Solution:

To improve global response times, a content delivery network (CDN) like AWS CloudFront can be used to cache media and serve it from edge locations closer to users. This reduces latency and ensures a smoother user experience regardless of geographic location.

Implementation:

A. **CloudFront Configuration:** Configure CloudFront to cache S3-hosted media files at edge locations. This minimizes the need for users to retrieve data from the origin server in Australia.

B. **DNS Management:** Use Route 53 for latency-based routing to direct users to the nearest

CloudFront edge location. This further enhances global response times.

C. **Custom Cache Behavior:** Set caching policies to prioritize frequently accessed media, optimizing both performance and costs.

7. It is expected the system will be extended to handle video media in the future.

Solution: We can extend the system with S3 for video uploads and add the AWS Elastic Transcoder for video transcoding into different platforms, and we can use Media Sorting for organization because there are now different contents which are delivered to storage via CloudFront.

Implementation:

1. **S3 Setup for Video Uploads:** Create an S3 bucket for video storage and set up event notifications to trigger processing when videos are uploaded.
2. **Video Transcoding with Elastic Transcoder:** Set up an Elastic Transcoder pipeline to handle transcoding and use Lambda to trigger transcoding jobs when videos are uploaded to S3.
3. **Media Sorting:** Use AWS Media Sorting to automatically organize transcoded media in S3, categorizing it by type or resolution.
4. **Content Delivery via CloudFront:** Create a CloudFront distribution to deliver processed media globally, pulling from the S3 bucket with optimized video streaming settings.

8. The media can be uploaded by users in all sorts of formats. The company would like various versions of media to be automatically produced (e.g. thumbnails, low resolution versions suitable for mobile phones, or video transcoding). The process for reformatting/transcoding/reprocessing media should meet the following criteria:

a. When a media item is uploaded to the S3 bucket, the creation of these alternative versions should be triggered automatically. Transformed media will also be stored in S3.

Solution: We can add **S3 event notifications** to trigger an AWS **Lambda function** upon media upload. Lambda analyzes the file type and initiates processing tasks (e.g., resizing or transcoding). The transformed versions are then saved in appropriate folders in **S3**.

b. The architecture for processing media should be extensible. For example, in the future it may be desirable to add the ability to automatically identify tags in photos using AI.

Solution: This can be configured by **Rekognition**. Rekognition can analyze photos or videos to detect objects, scenes, or faces, generating metadata (e.g., tags) that can be **stored in S3 or indexed in DynamoDB** for search.

c. Different processing services should be able to be run on the most suitable platform – e.g. EC2 instance, Lambda, or other AWS managed services. Given cost and performance constraints it is assumed that all services will be provided in the AWS ecosystem.

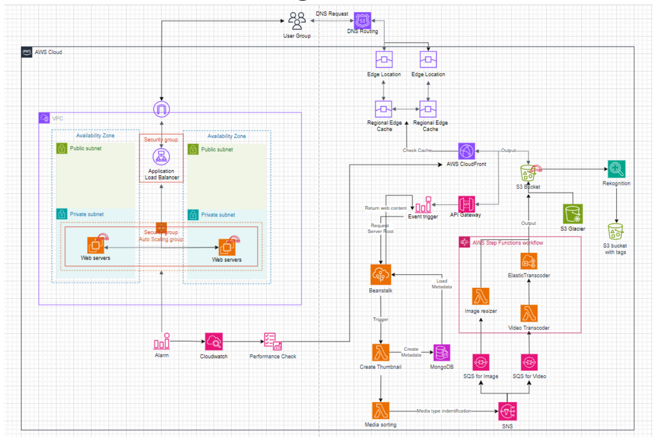
Solution: Run lightweight tasks like **image resizing or thumbnail** creation on **AWS Lambda** for cost efficiency. Use AWS Elastic Transcoder for video processing, and Media Sorting for organizing outputs. For more **complex workflows**, integrate **AWS Step Functions** to manage different AWS-managed services based on task complexity.

d. The reprocessing/reformatting of media is often a time-consuming task. The architecture should be designed so that the application does not become overloaded and is effectively decoupled. For example, multiple 'worker' nodes can process transformation jobs that have been placed on a queue. The worker nodes may specialize in particular tasks. For example, one node may specialize in video transcoding which is much more processor and memory intensive than reformatting a photograph.

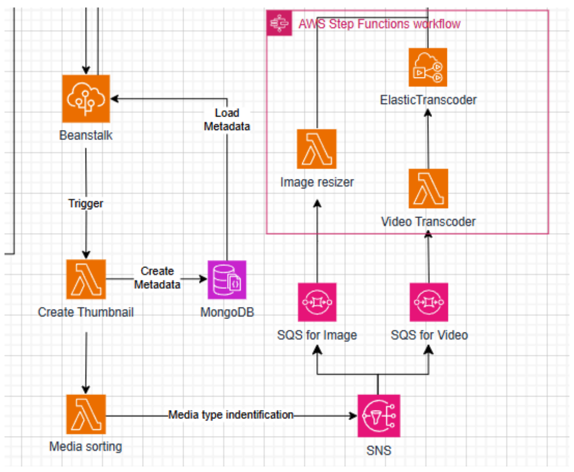
Solution: Decouple media processing by using Amazon **SQS** to queue tasks. Implement specialized **Lambda** functions to process tasks independently. Scale worker nodes based on demand AWS **Step Functions** to ensure the system remains responsive and efficient without overloading.

III. ARCHITECTURE DESIGN

1. Architectural Diagram



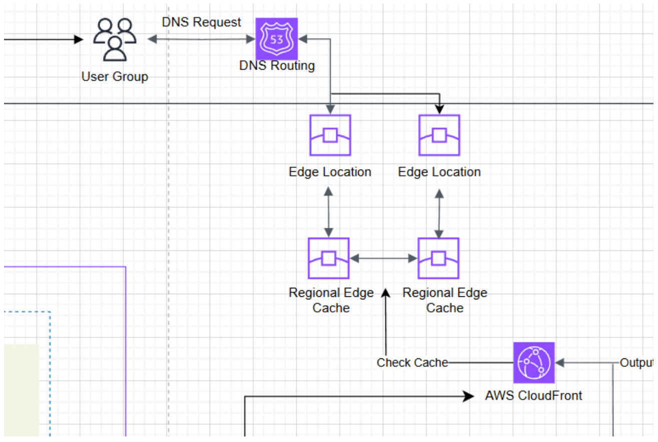
2. Description of used services



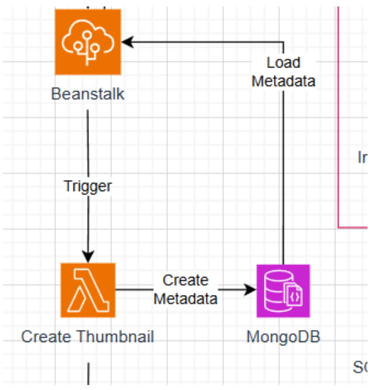
AWS Lambda: Employing the benefits of the serverless compute service that executes code in response to events, scaling automatically with demand. Used for processing media tasks such as image resizing, thumbnail creation, and initiating video transcoding.

Amazon SQS: A managed message queuing service that

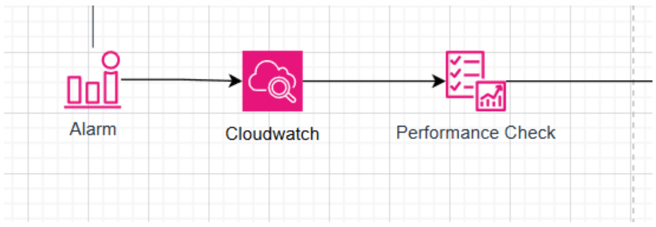
decouples system components. Queues media processing tasks, ensuring asynchronous and scalable workflows while avoiding application overload.
AWS Elastic Transcoder: A media transcoding service that converts video files into formats suitable for various devices. Used to transcode uploaded videos into multiple resolutions and formats, meeting platform requirements.
AWS Media Sorting: Automatically categorizes and organizes media files in storage. Helps to manage processed media efficiently in S3, organizing by type or resolution.
AWS Step Functions: A serverless orchestration service for coordinating multiple AWS services into workflows. Manages complex processes like video transcoding and AI-based tagging.
Amazon API Gateway: Provides a RESTful API interface for interacting with the system. Works with Lambda to handle dynamic requests and implement an event-driven architecture.



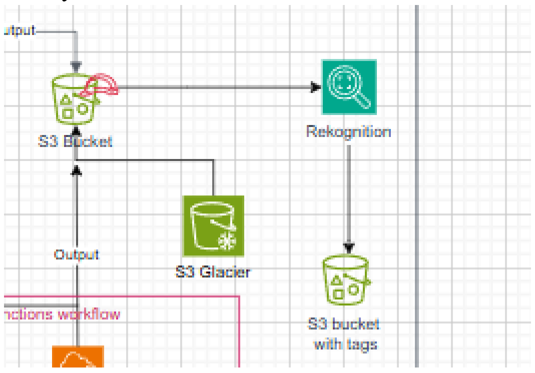
Amazon CloudFront: Employing a content delivery network (CDN) that caches content at edge locations worldwide to reduce latency. It helps deliver media files globally and improves performance for users outside Australia.
Amazon Route 53: A DNS service providing latency-based routing to direct users to the nearest CloudFront edge location. Improves response times for global users accessing the system.



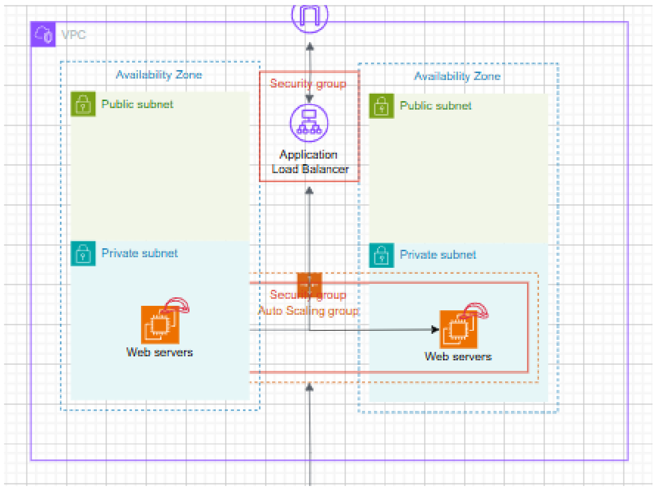
Amazon DynamoDB: NoSQL database designed for high availability, low latency, and scalability. Replaces the traditional relational database to store metadata for media and manage simple data structures.



Amazon CloudWatch: Monitors resources and tracks metrics, logs, and application performance. Used to trigger alerts for resource overutilization and adjust scaling policies dynamically.



AWS Rekognition: A machine learning service for image and video analysis. Used to extend the system for AI-based photo tagging, enabling enhanced categorization and search features.

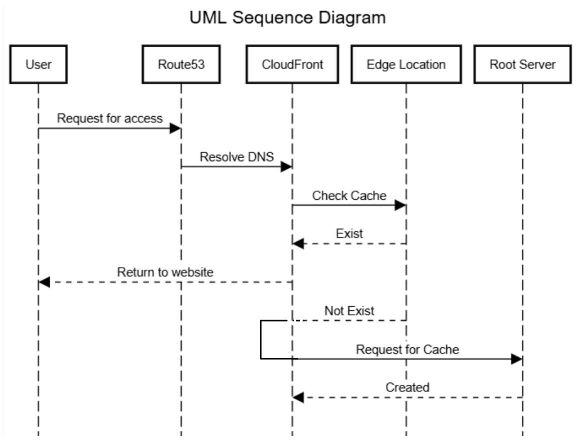


Auto Scaling and EC2:

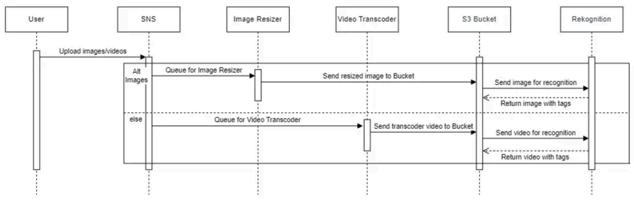
- EC2 provides scalable compute capacity.
- Scaling adjusts the number of instances dynamically based on demand.
- Used for stateful or CPU-intensive tasks that exceed serverless capabilities.

3. **UML sequence diagram**

A. UML Sequence diagram of the website access process



B. UML Sequence Diagram of the user interaction.



IV. DESIGN RATIONALE

1. Alternative solutions

For scenario 2, Kubernetes can be effectively utilized through **AWS Elastic Kubernetes Service (EKS)**. This powerful tool offers detailed control over how applications scale and distribute workloads, presenting a viable alternative to **traditional Auto Scaling Groups**. By leveraging Kubernetes on EKS, organizations can automate scaling based on real-time traffic, ensuring that resources are allocated efficiently to meet user needs without manual intervention. This capability is particularly beneficial for businesses experiencing fluctuating demand, as it allows for rapid adjustments to maintain performance and reliability.

For scenario 3, **AWS Fargate** emerges as a compelling option. Fargate simplifies the deployment of containerized applications by removing the need for server management entirely. This serverless model is particularly advantageous for tasks that are resource-intensive, such as transcoding video or resizing images, as it automatically scales resources in response to workload fluctuations. Moreover, Fargate operates on a **pay-as-you-go** basis, which means that organizations only incur costs for the resources consumed during processing tasks. This not only minimizes overhead but also allows for more predictable budgeting, making it ideal for projects with variable workloads.

2. Design criteria

Issue	**Criteria**	**Designed solution**
Performance	- High-speed media delivery. - Efficient processing pipelines. - Reduce latency globally.	- AWS CloudFront for scaling and caching. - AWS Route53 for sustainable routing and transferring data. - SQS ensures efficient queueing and asynchronous processing.
Reliability	- Fault-tolerant and highly available systems. - Ensure durability of stored data. - Robust failover mechanisms.	- S3 provides 11 nines of durability. - DynamoDB Global Tables enable multi-region availability. - CloudWatch monitors system health and triggers alerts for proactive response. - AWS-managed services reduce single points of failure.
Security	- Secure storage and access. - Data protection during transit and at rest. - Least privilege access policies.	- S3 Encryption (SSE-S3/SSE-KMS) secures data at rest. - SSL/TLS ensures secure transmission. - IAM Roles with least privilege policies manage access. - AWS CloudFront enhances security with edge-based delivery and DDoS protection.
Cost	- Efficient storage and processing costs while handling 1TB of data monthly.	- S3 standard storage. - CloudFront for delivery, Lambda for processing, and optimized EC2 usage for CPU-intensive tasks (~120$/month, review the table for detailed calculation).

Estimated cost calculation:

Service	Component	Estimated Usage	Estimated Cost
Amazon S3	- Storage (Standard) - Data Transfer to CloudFront	- 1TB for storage - 1TB for data transfer	- $23 (at $0.023/GB/month) - $90 (at $0.09/GB for transfer out to CDN)
Amazon CloudFront	- Data transfer from CloudFront to Users - Request	- 1TB - 1 million request (at $0.0075 per 10,000 GET requests)	- $85 (at $0.085/GB for the first 10 TB) - $0.01 (at $0.0075 per 10,000 GET requests)
Amazon Lambda	- Compute time - Request	- 1 million requests, 1 GB RAM - 1 million requests	- $2 (based on 100 ms average execution time) - $0.40 (at $0.40 per million requests)
Amazon SQS	- Message request	- 1 million messages	$0.20 (at $0.20 per million requests)
Amazon Elastic Transcoder	- Video Transcoding	10 hours	- $20 (at $2.00 per hour of HD video)
Amazon Rekognition	- Image Analysis	1,000 images	- $1.00 (at $1.00 per 1,000 images analyzed)
Amazon Route53	- DNS Queries	1 million queries	$0.50 (at $0.50 per million requests)
Amazon MongoDB	- On-Demand Read/Write Operations - Data Storage	- 1 million read/write ops - 1BG	- $1.25 (at $0.25 per million writes and $0.25 per million reads) - $0.25 (at $0.25/GB)

Estimated cost is around $225 for running 1 TB of data in a month.

3. **Justification for best solution based on the design criteria**
   - **Global Reach with Low Latency:** The use of CloudFront and Route 53 latency-based routing ensures that users worldwide experience consistent and fast response times for global access.
   - **Cost-Effective:** The architecture avoids over-provisioning by using on-demand capacity modes for DynamoDB and pay-as-you-go services like Lambda and S3. Lifecycle policies and intelligent tiering further optimize costs as the company requires.
   - **Reliability and High Availability**: AWS-managed services such as **S3** (11 nines durability) and DynamoDB Global Tables ensure data is always available and fault-tolerant. CloudWatch enables real-time monitoring and proactive incident response to maintain high availability.