COS20019 - Cloud Computing Architecture

Assignment 3

Serverless/Event-Driven Architecture Design Report

Serverless/Event-Driven Architecture Design

by Truong Ngoc Gia Hieu | Pham Dinh Duc Anh | Duong Nguyen Dang

Department: Swinburne University of Technology

Course Name: COS20019 – Cloud Computing Architecture (HCMC)

Instructor Name: Pham Thai Ky Trung

Due Date: Monday, Nov 17th 2025

Submission Date: Monday, Nov 17th 2025

Word count: 4801 words

Table of Content

Part A: Introduction and Business Context

1.1 Assignment Overview

Due date: 09:00 AM (AEST) Monday of Week 12 (end of Week 11), to Canvas.

Late submissions will not be accepted for this assignment.

No Extensions will be available for this assignment

Presentation Interview (required): Taking place in Week 12. Appointments for presentation will be scheduled and announced during week 11. Go to **Canvas | People | Assignment 3 – Groups** and form a group and send your group details (screenshot) to your tutor by end of week 10.

Prerequisite requirements:

1. Submitted **Assignment 2**
2. **Complete Presentation/Demonstration for Assignment 3 in week 12**

All supporting materials mentioned in this document can be found in the corresponding assignment page on Canvas.

This assignment must be completed as a group of minimum three(3) and maximum four(4) students.
It will be beneficial to have multiple perspectives on a problem when discussing design alternatives.

Additional requirements will apply to group submissions, as indicated below in this document.
To form your group, go to *Canvas/People/Assignment 3 – Groups*.
Only one student per group should submit the assignment Report file, however, all group members must present in week 12.

Objectives

Examine alternative solutions to develop a design for a scalable highly available Web Site and justify your chosen solution with a design rationale.

NOTE: You may find the Architecture Review Questions in the Week 11 page in Canvas helpful in evaluating your architectural design. There are also many references available on good architectural practice in the cloud such as:

https://aws.amazon.com/architecture/

https://d1.awsstatic.com/whitepapers/aws-web-hosting-best-practices.pdf
https://d1.awsstatic.com/whitepapers/AWS_Cloud_Best_Practices.pdf
https://docs.aws.amazon.com/wellarchitected/latest/framework/w

ellarchitected-framework.pdf

http://en.clouddesignpattern.org/index.php/Main_Page

1.2 Abstraction

In the era of globalization, the significant growth of Cloud Computing in the era of globalization has fundamentally reshaped the global technological landscape, establishing flexible, pay-per-use operational models as the industry standard. Specifically, cloud services now underpin a vast array of global applications, from e-commerce and media streaming to complex data analytics which make architectures prioritizing scalability, elasticity, and high availability the definitive requirement for modern software development.

Due to these reasons, the Swinburne University of Technology published the third assignment, "Serverless/Event-Driven Architecture Design Report," for the Cloud Computing Technology (COS20019) course, requiring students to practically examine alternative solutions and develop a robust design for a scalable, highly available Web Site, thereby justifying their chosen solution with a comprehensive design rationale.Additionally, the report highlights the comprehensive design and analysis of a Serverless/Event-Driven Architecture as the optimal strategy to fulfill these objectives. Furthermore, the subsequent sections will detail the architecture's selection, structure, and operation, from its logical components and multi-tier separation to its physical security model and dynamic behavior (UML Sequence Diagrams). Ultimately, the report concludes with a rigorous Requirement Fulfillment Analysis and Cost Estimation to confirm the technical and economic feasibility of the proposed cloud solution.

1.3 Introduction

According to requirements of the assignment, which explicitly necessitate the development and justification of a scalable and highly available Web Site, my team researched and figured out what AWS services are adapted to an optimal Serverless/Event-Driven Architecture. As a result, my team decided to implement following AWS services to adress business scenarios, ensuring elasticity and cost-efficiency: API Gateway and Lambda will handle synchronous user access and authentication; Amazon S3 and Amazon SQS will establish the robust, decoupled event-driven mechanism for asynchronous media processing; and Amazon DynamoDB combined with WebSocket will provide the necessary functionality for real-time status updates. The subsequent sections of this report are organized to detail this entire chosen design, providing a comprehensive analysis of the structure, operation, and economic feasibility.

1.4 Business Scenario

During the research and analysis phase, my group realized that the successful "Photo Album" web application requires a significant architectural redesign to address escalating demand and current performance bottlenecks. Consequently, the innovative system must align with the core principles of a Serverless/Event-Driven architecture while focusing on scalability, robustness, high availability, and cost efficiency. Therefore, the following problems and requirements, identified by the company, form the critical constraints for the proposed design.

1.4.1 Operation anf Cost Optimization

The company involves the maximization of managed cloud services to minimize the need for in-house systems administration and all media will be stored in Amazon S3. In response to this challenge, team members selected Amazon DynamoDB, a Serverless NoSQL database, to replace the legacy system. Moreover, this decision ensures millisecond performance and automatic scalability for simple data structures, effectively addressing both the cost constraints and the operational goal of using managed services.

1.4.2 Scalability and Performance Requirements

The primary obstacle is accommodating the expected exponential growth—with demand doubling every six months—which necessitates absolute elasticity. Notably, the current EC2 performance issues are resolved by replacing the compute layer with AWS Lambda and Amazon API Gateway, enabling them to scale instantly and automatically to handle synchronous user traffic. In addition, Amazon CloudFront is integrated as a Content Delivery Network (CDN) to cache both static media (from S3) and API endpoints closer to the global user base, enhancing response times for users across the globe.

1.4.3 Architectural and Functional Requirements

The upgraded system must be built upon the fundamental principle of a Serverless/Event-Driven architecture to ensure full decoupling of components. Particularly, this changing is critical for the time-consuming process of media transformation, which must be handled asynchronously. Furthermore, the Amzaon S3 Bucket is designed to trigger a message in Amazon SQS, serving as a durable queue to prevent the application from being overloaded.

Besides, the processing capability is implemented via specialized worker nodes, leveraging AWS Lambda for high-velocity, ephemeral tasks, and potentially utilizing AWS Fargate to containerize workloads that require greater compute resources or longer execution times. Then, team leader agreed with members that the modular architectural design above secures intrinsic extensibility, facilitating the seamless integration of future services, such as Amazon Rekognition for sophisticated AI-based tagging, without introducing disruptive changes to the core application workflow.

Part B: Architecture Design and Structure

2.1 Logical Component Model

As noted by Lucidchart (2023), “Logical diagrams serve as blueprints that guide cloud deployment, monitoring, and troubleshooting,” reinforcing their role as strategic assets in cloud-native environments. Following this principle, the Logical Architecture Diagram provides a high-level abstraction essential for translating analyzed requirements into a system solution. Subsequently, the primary function is to illustrate how the selected Serverless AWS services are organized into decoupled logical tiers, focusing on the data and event flow. This significant visualization serves as a critical "roadmap" for stakeholders, validating that the design meets criteria for scalability, security enforcement, and modularity before analyzing the operational flow.

2.1.1 Diagram Overview

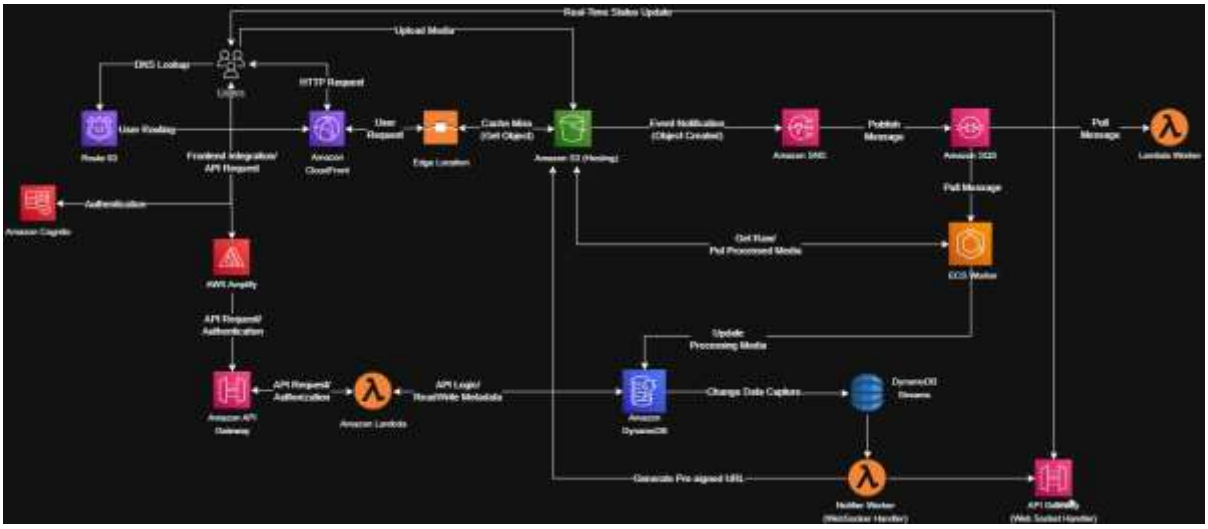

Figure 1: Service Interaction Diagram

The figure above demonstrates a highly decoupled, event-driven architecture centered on fulfilling the scalability and asynchronous media processing requirements. The design separates the system into three main flows:

- **Frontend & Authentication**: User access is managed via Route 53 and authenticated using Amazon Cognito. Thus, AWS Amplify and Amazon CloudFront services handle frontend integration and caching, significantly improving global performance.
- **Synchronous API Flow**: API requests are routed through Amazon API Gateway to AWS Lambda (API Logic), handling metadata reads/writes to Amazon DynamoDB. As my team expected, this fully serverless pathway ensures instant, automatic scalability for all user interactions.
- **Asynchronous Media Processing Flow**: The core event-driven mechanism is triggered when media is uploaded to Amazon S3. Next, S3 service generates an Event Notification to Amazon SNS that publishes the job to Amazon SQS. Then, T=this queue decouples the application from time-consuming tasks and the queue is consumed by specialized workers, including Lambda Workers (for quick tasks) and ECS Workers (for processor-intensive tasks like video transcoding)

Finally, real-time status updates are managed by DynamoDB Streams activating a Notifier Worker (Lambda), pushing status updates back to the client via an API Gateway (WebSocket Handler). In short, the comprehensive architecture successfully migrates the entire system to a fully managed and serverless model that adapts to business demands and assignment's requirements.

2.1.2 User Access and Metadata Mechanism

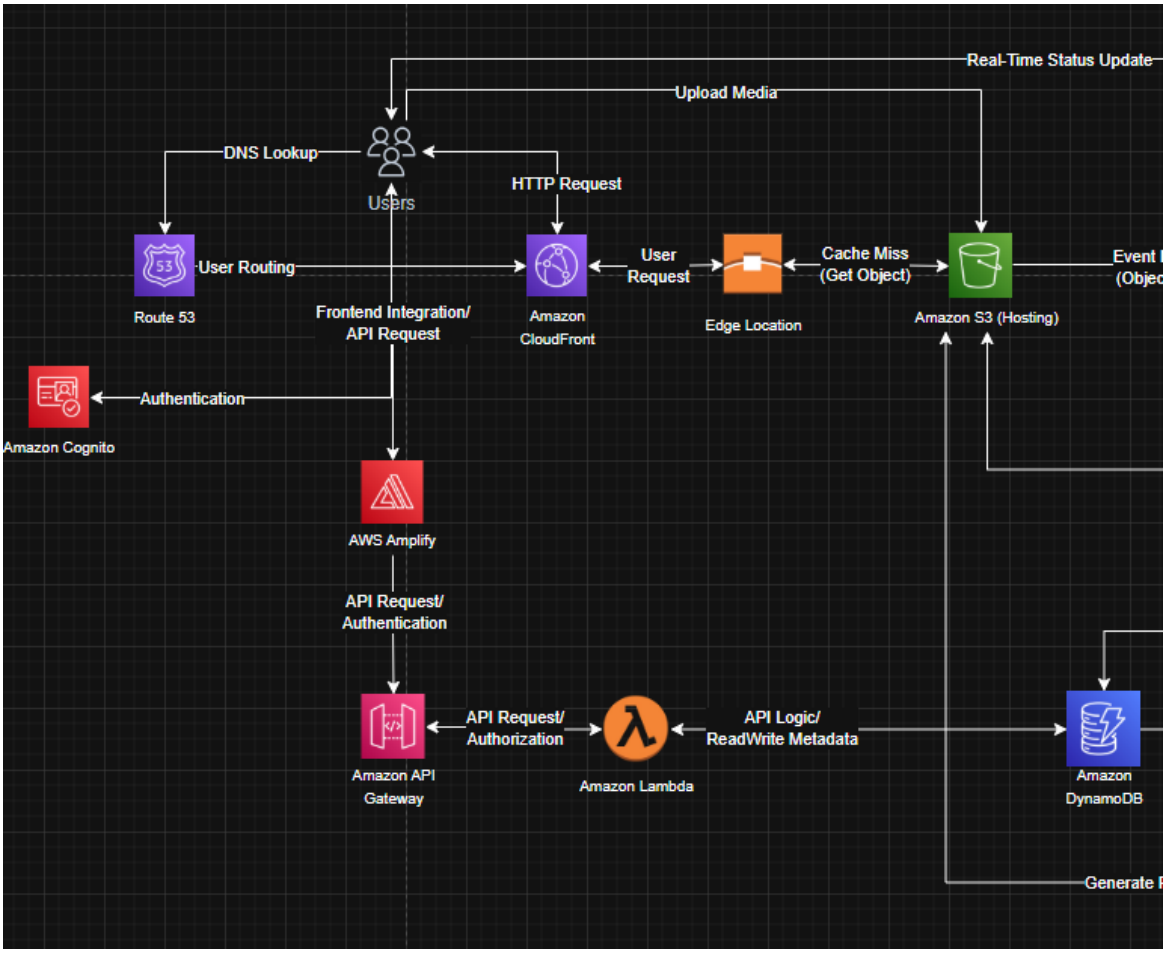

Figure 2: User Access and Metadata Mechanism

The diagram above illustrates a high-level structure detailing the Synchronous Flow and Metadata Management Mechanism. Firstly, when users access to website named (for example, photoalbum.com), the HTTP request triggers a DNS Lookup handled by Amazon Route53 service which manages User Routing to the nearest point. Next, the request reaches Amazon CloudFront, acting as the Content Delivery Network (CDN) to serve he cached frontend from its Edge Location, directly addressing the requirement for improved global response times. From a security perspective, Amazon Cognito handles the necessary Authentication to verify the user's identity before any application logic is accessed. Therefore, these available services establish a secure and highly available Presentation Tier, responsible solely for the delivery of static application assets and resources such as files and images

Following completion of successful authentication, the intergrated frontend, supported by AWS Amplify, directs synchronous HTTP API requests for core functions including creating albums and fetching metadata) to the Amazon API Gateway. Moreover, the Gateway performs Authorization before invoking a dedicated AWS Lambda function (API Logic), executing the crucial Read/Write Metadata operations directly onto the non-relational Amazon DynamoDB. As a result, this sequential chain of fully Serverless services forms the Application and Data Tier, ensuring that core transactions are processed rapidly and providing the absolute elasticity necessary to handle the mandated exponential growth. Specifically, this tier manages the dynamic application state and metadata, entirely separate from the static content delivery.

2.1.3 Asynchronous Task Processing Flow

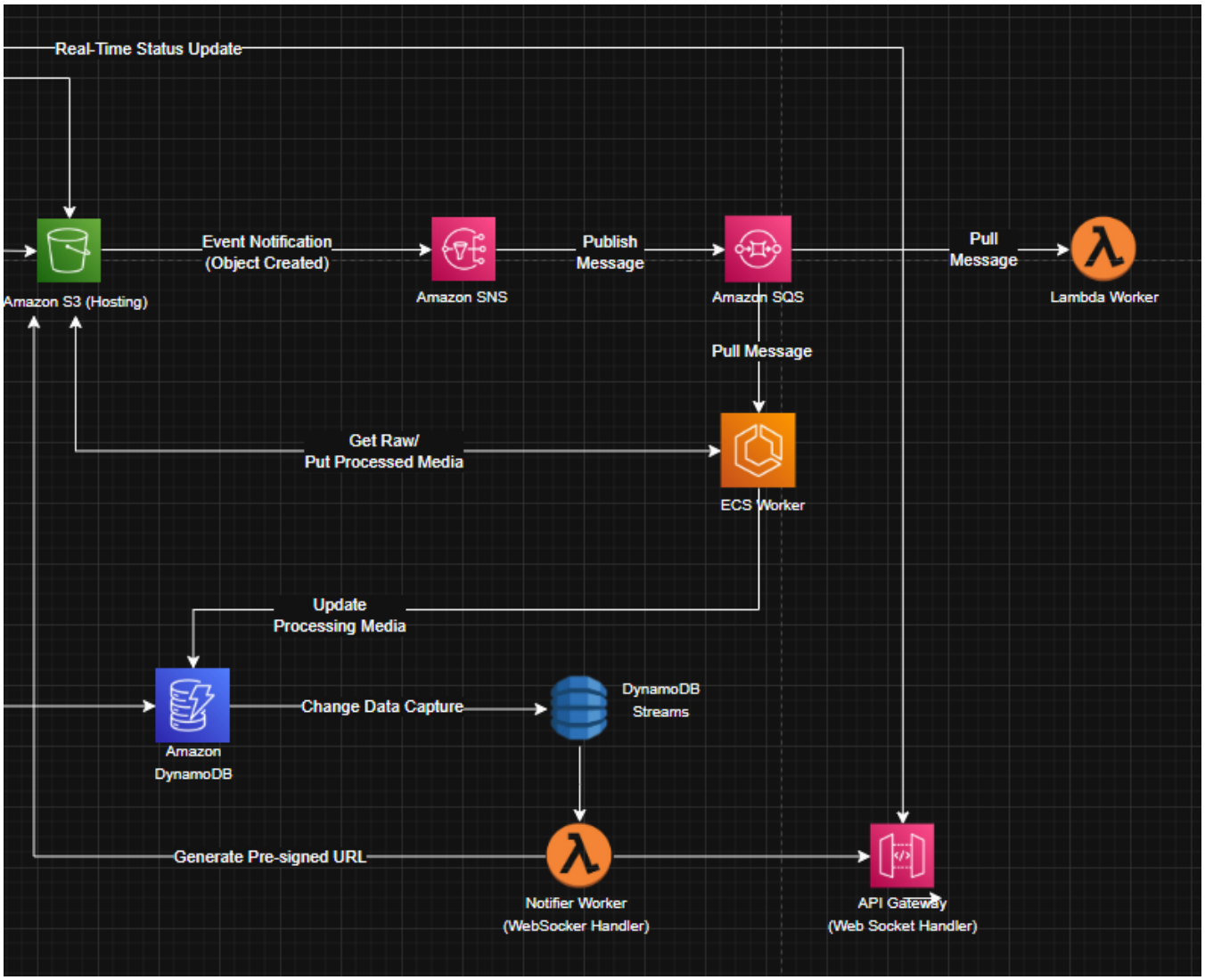

Figure 3: Decoupled Media Processing Pipeline

Based on the diagram above, this section introduces the Event Orchestration and Asynchronous Processing Mechanism that is particularly designed to handle time-consuming and resource-intensive media processing tasks, contrasting with the synchronous flow that handles metadata. First, the process starts when a user completes the media upload directly to Amazon S3 (Media Storage) (such as a large 4K video file or a high-resolution image). The upload event triggers an Event Notification (Object Created), which is immediately sent to Amazon SNS (Simple Notification Service). Subsequently, SNS acts as a fanout service, pushing this message into a single Amazon SQS (Simple Queue Service). In this cae, my team chose the Amazon SQS service for its critical role in completely decoupling the frontend application from the heavy processing tasks, ensuring message durability, fault tolerance, and allowing the synchronous system to respond instantly to the user.

2.1.4 Real-Time Status Update

Based on the business demands in the business scenario description, my team researched, analyzed, gathered toghether, selected carefully, and published the Final Architectural Solutio which includes the

specialized Real-Time Status and User Feedback Mechanism. Additionally, this specific flow was implemented to meet the critical business requirement of providing immediate and non-blocking feedback to users during long-running media processing tasks. In details, when the workers (Lambda or ECS) complete processing and write the new status (processed/failed) into Amazon DynamoDB, this event triggers DynamoDB Streams (Change Data Capture). Next, the Streams immediately transfer this change data to a dedicated Notifier Worker and then this Worker uses the persistent connection provided by the API Gateway (WebSocket Handler) to push the status notification directly to the user's browser. My team believes that this WebSocket mechanism is the optimal choice, replacing traditional polling methods (repeated HTTP requests), significantly reducing the load on the synchronous system and ensuring users receive accurate information in real-time without interrupting the application experience.

2.2 Serverless Multi-Tier Architecture

2.2.1 Layered Architectural Concept

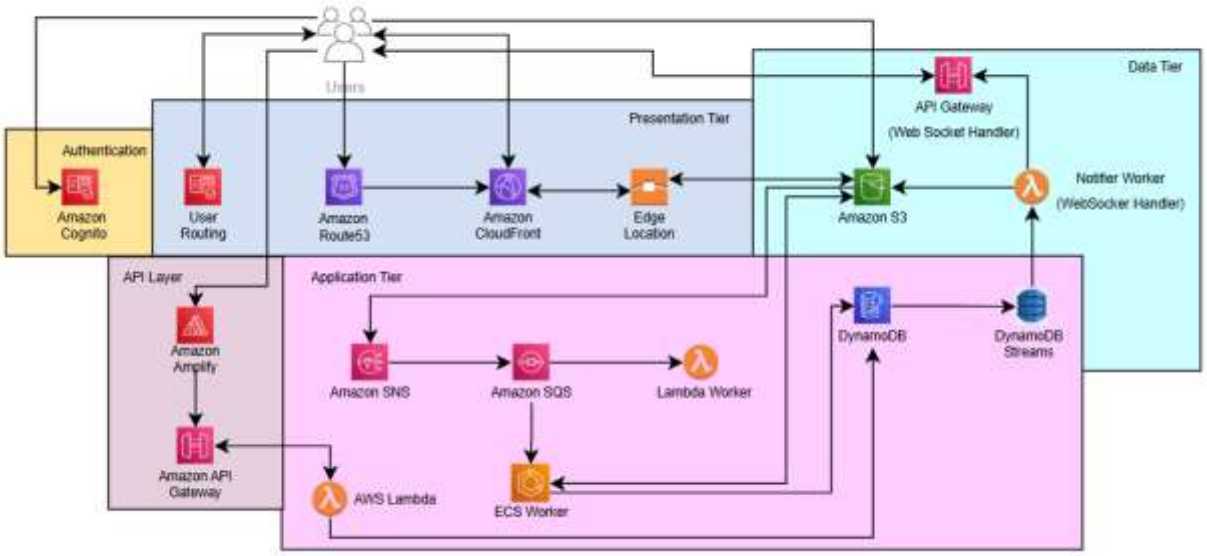

Figure 4: Three-Tier Design Overview

The proposed architecture adopts a clean, layered architectural concept to ensure separation of concerns and maximize the benefits of a Serverless approach. According to the IBM, the chief benefit of three-tier architecture is that because each tier runs on its own infrastructure, each tier can be developed simultaneously by a separate development team, and can be updated or scaled as needed without impacting the other tiers. In addition, ***Figure 4*** showcases the clear logical separation of the three specialized tiers: the Presentation Tier, the Application Tier, and the Data Tier, forming the foundation of this resilient and scalable system.

2.2.2 Presentation Tier

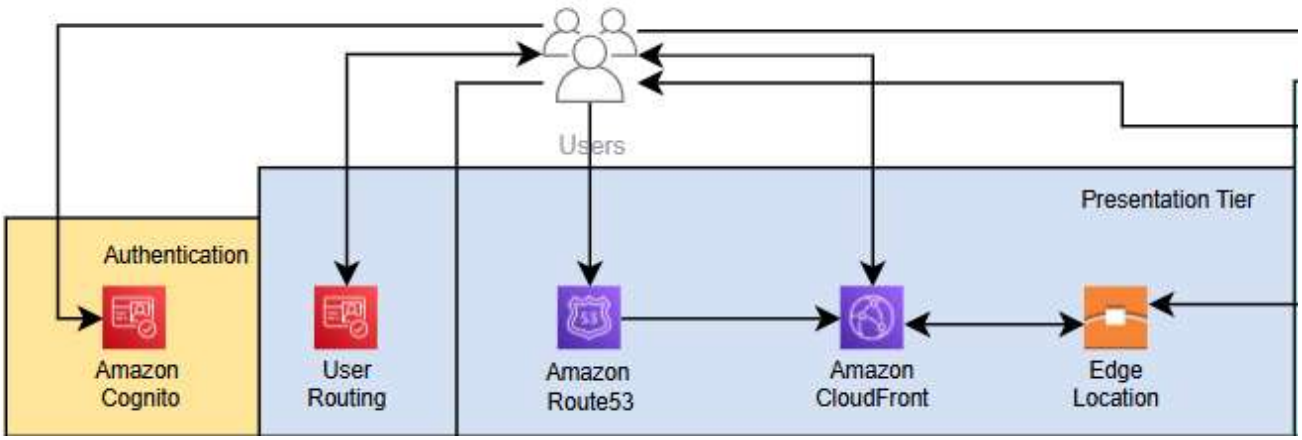

Figure 5: Three-Tier Design Overview

The Presentation Tier is the system's edge layer, responsible for global content delivery, user routing, and front-end authentication. Its primary goal is to ensure the fastest possible response time and secure initial access. Specificially, its primary goal is to ensure the fastest possible response time and secure initial access.

- **Amazon Route 53:** Manages DNS lookup and user routing.
- **Amazon CloudFront:** Functions as the Content Delivery Network (CDN) to serve cached static assets from its Edge Locations.
- **Amazon S3 (Hosting):** Serves as the origin for static frontend files, providing reliable web hosting.
- **Amazon Cognito:** Manages user authentication and identity verification before access to the Application Tier is granted.

2.2.3 API Layer and Application Tier

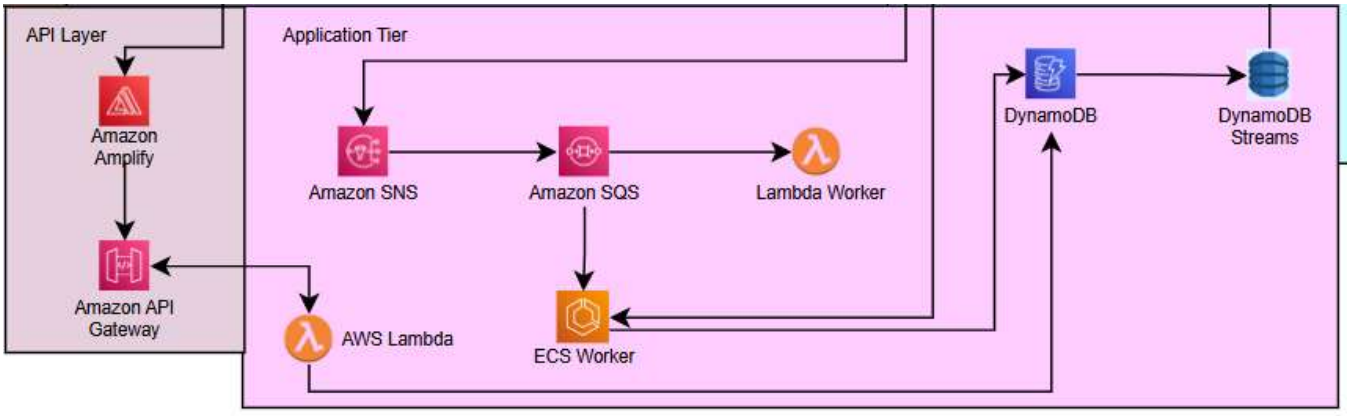

Figure 6: API Layer and Application Tier

The Application Tier is the central hub for the system's business logic, handling synchronous API requests and orchestrating asynchronous media processing. This tier ensures absolute elasticity by employing Serverless compute and messaging services.

- **AWS Amplify & Amazon API Gateway**: Manage the frontend integration, request routing, and authorization for synchronous API calls.
- **AWS Lambda (API Logic)**: Executes the core synchronous business logic, such as metadata read/write operations.
- **Amazon SNS & Amazon SQS**: Implement the event orchestration and message queuing for asynchronous processing, ensuring decoupling and message durability.
- **Lambda Worker & ECS Worker (Fargate):** Specialized compute resources responsible for consuming SQS messages and performing heavy, long-running media processing tasks

2.2.4 Data Tier

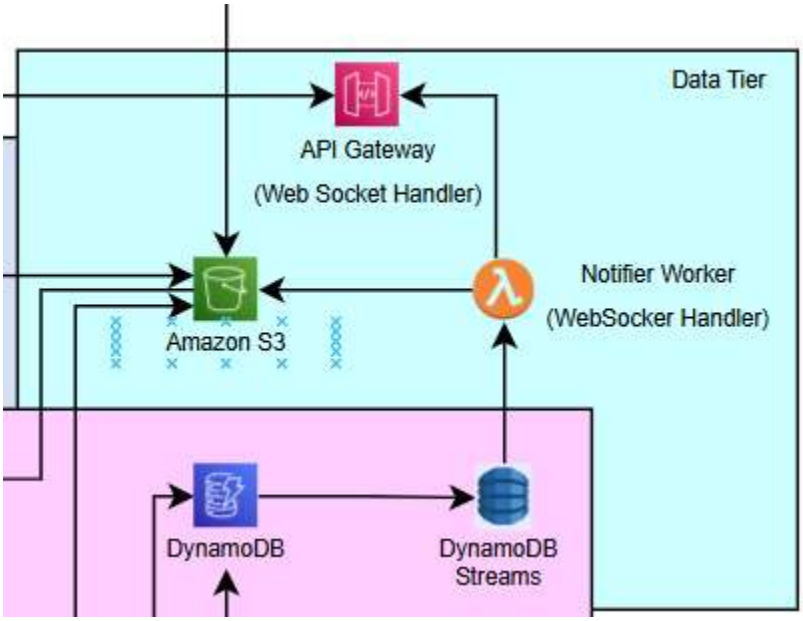

Figure 7: Data Tier

The Data Tier is responsible for the persistent storage of all application metadata, media files, and real-time status updates. This tier exclusively utilizes highly scalable managed database and storage services.

- **Amazon DynamoDB**: A fully managed NoSQL database used for storing core application metadata, user profiles, and processing statuses, providing high-speed synchronous access.
- **Amazon S3 (Media Storage):** Provides scalable, durable object storage for raw and processed media files (photos, videos).
- **DynamoDB Streams**: Provides Change Data Capture (CDC) functionality, triggering real-time notification workers upon data change.
- **Notifier Worker & API Gateway (WebSocket):** While technically cross-cutting, these components are intrinsically linked to the Data Tier, facilitating the final step of the real-time status update flow by pushing data from the backend to the user.

2.3 AWS Infrastructure Configuration

According to Rajesh Dhiman (2025), “One of the most prominent examples of scalable physical architecture is Amazon Web Services (AWS). AWS's global infrastructure is designed with redundancy, fault tolerance, and scalability in mind.” Based on this fundamental understanding, my team decided to leverage the following sections to detail the physical setup of the solution, which establishes a secure, highly available foundation necessary to meet all mandated business requirements.

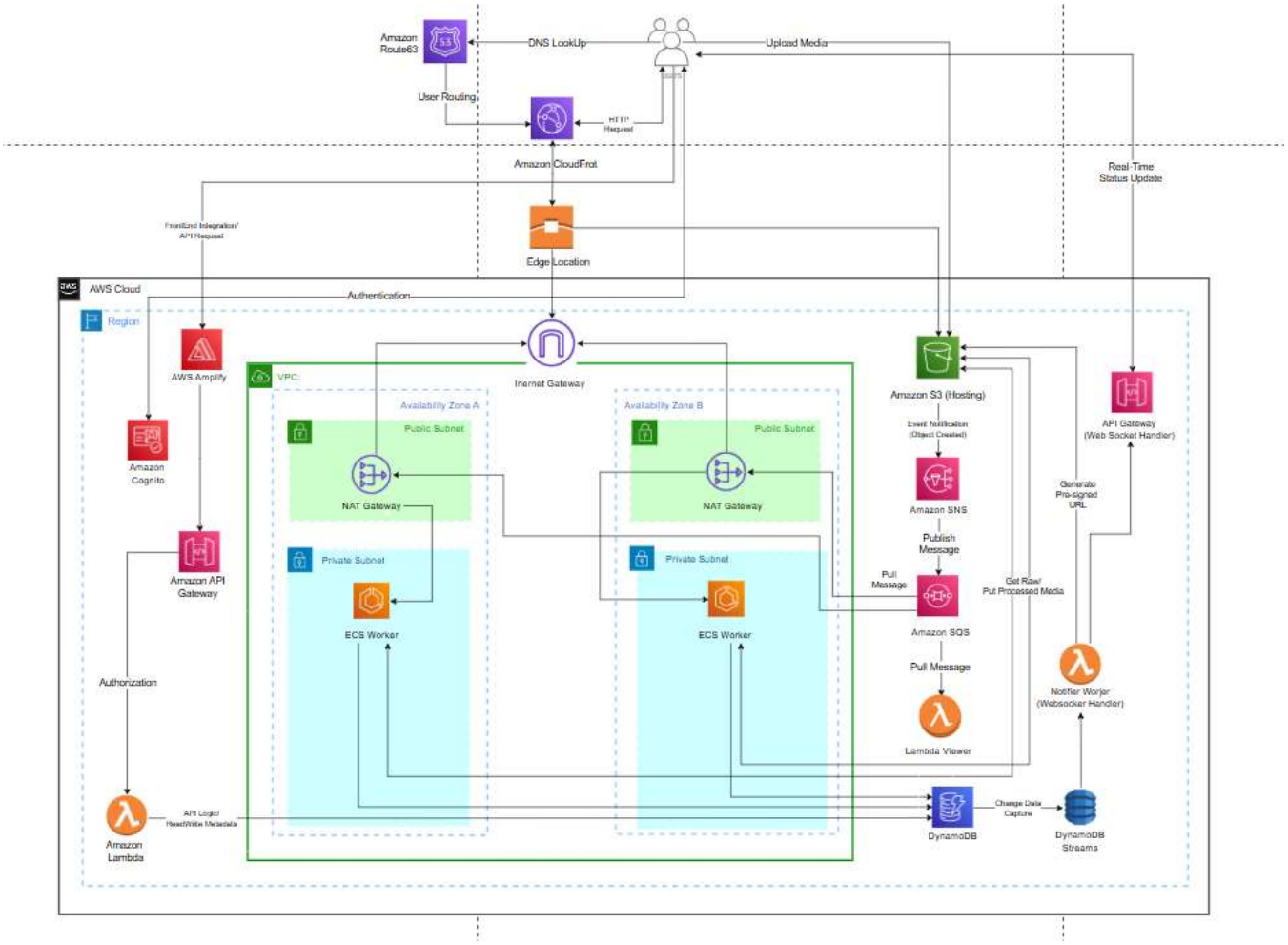

Figure 8: AWS Physical Architecture

2.3.1 High Availability and Zoning

The architecture prioritizes High Availability (HA) and resilience by ensuring that core services are distributed across multiple failure domains.

- **Multi-AZ Deployment**: The foundational Amazon Virtual Private Cloud (VPC) is configured to span at least two Availability Zones (AZs) (e.g., AZ-A and AZ-B). This geographical separation ensures that service continuity is maintained even in the event of an outage in a single AZ.
- **Service Redundancy**: Stateless Serverless services (such as Amazon DynamoDB, Amazon S3, AWS Lambda, Amazon SNS, and Amazon SQS) inherently provide HA and automatic failover by replicating data and compute across multiple AZs managed by AWS.

- **ECS Worker Redundancy**: Critical processing resources, specifically the ECS Workers running on Fargate, are deployed across the Private Subnets in both AZs. This ensures that asynchronous media processing continues uninterrupted even if one AZ experiences downtime.

The designed architecture in the thrid assignment achieves High Availability (HA) and resilience by deploying the VPC across multiple Availability Zones (AZs) and leveraging the inherent multi-AZ redundancy of Serverless services (DynamoDB, SQS, and Lambda). As a result, this architecture fundamentally differs from traditional IaaS (Assignment 2), as it eliminates the need for manual configuration and management of Auto Scaling Groups (ASGs) and Load Balancers (ALBs), shifting the responsibility for fault tolerance entirely to the AWS managed control plane.

2.3.2 Network Security Strategy

Based on my group AWS Physical diagram, the network security strategy is based on the principle of least privilege and defense in depth through layered segregation between public and private resources.

- **Public/Private Subnets**: The deployment strictly segments the network into Public Subnets (hosting the Internet Gateway and NAT Gateway) and Private Subnets (hosting the highly sensitive ECS Workers). This limits the attack surface dramatically.
- **Security Groups**: All resources within the VPC (e.g., ECS Workers) are governed by Security Groups, acting as stateful virtual firewalls that control both inbound and outbound traffic at the instance level. They are configured to only allow necessary communication.
- **WAF Integration**: Amazon CloudFront is strategically placed at the edge of the Presentation Tier, enabling simple integration with AWS WAF (Web Application Firewall) to mitigate common web exploits and DDoS attacks before they reach the core infrastructure.

Additionally, security is enforced through strict network segregation using Public and Private Subnets, ensuring that only non-sensitive components are publicly accessible. Furthermore, the Serverless architecture utilizes the AWS Service Boundary for protection, significantly reducing the attack surface. In stark contrast to Assignment 2, where every application component required continuous, meticulous management of Network ACLs and Security Groups for every application port, this solution's security posture is simpler, stronger, and inherently managed.

2.3.3 Private Access Mechanism

A key aspect of maintaining security is ensuring that resources in the Private Subnets can access necessary public AWS services without exposure to the internet.

- **NAT Gateway Usage**: A NAT Gateway is deployed in each Public Subnet. The primary function of the NAT Gateway is to allow the ECS Workers in the Private Subnets to establish secure outbound connections to services like Amazon S3 (to pull container images or put processed media) and Amazon DynamoDB (to update metadata status).
- **VPC Endpoints (Alternative/Enhancement)**: For superior security and cost control, services like DynamoDB and S3 can be accessed directly from the Private Subnets using VPC Endpoints (Interface or Gateway Endpoints). This eliminates the need for traffic to traverse the NAT

Gateway for these specific AWS services, keeping the data entirely within the AWS network perimeter.

The Private Access Mechanism is established via NAT Gateways deployed in the public subnets to enable secure, controlled outbound connections required by the ECS Workers in the private subnets (e.g., for pulling container images or updating S3/DynamoDB). This represents a highly focused use case compared to traditional IaaS (Assignment 2), where the NAT Gateway was often required by *every* application server for general internet access, leading to increased routing complexity and potentially higher operational costs.

Overall, the innovative designed AWS physical architecture directly addresses the mandates of Assignment 3. In addition, the deployment across multiple Availability Zones ensures the non-negotiable requirement for high fault tolerance and resilience. Furthermore, the specialized use of Public and Private Subnets and NAT Gateways guarantees a robust Network Security Strategy. By integrating the Serverless operational model within this resilient network fabric, the solution achieves the required absolute elasticity and operational cost efficiency that traditional VM-based architectures (like those typical in Assignment 2) cannot match.

Part C: Strengths, Weaknesses, and Challenges

3.1 Key Advantages

Based on requirements in the third assignment, the Serverless Multi-tier Architecture provides significant strategic and operational benefits, directly addressing the limitations of traditional architectures in the second assignment and and fulfilling the core mandates of the project.

- **Infinite Scalability and Elasticity**: All core services (AWS Lambda, Amazon DynamoDB, Amazon SQS, and ECS Fargate) scale automatically and instantaneously from zero to peak demand. This capability provides absolute elasticity, ensuring the system can handle unexpected traffic spikes without manual provisioning, which was a recurring manual challenge in traditional IaaS (EC2) environments.
- **Operational Cost Efficiency (Pay-per-Use):** The architecture leverages the "pay-per-execution" model, meaning the organization only pays for the actual compute time and resources consumed. By eliminating idle server costs, the solution guarantees optimal operational cost efficiency.
- **High Resilience and Fault Tolerance**: The design achieves inherent resilience through the multi-AZ configuration of the VPC and the managed nature of services (e.g., DynamoDB and SQS). Furthermore, the SNS/SQS decoupling ensures that asynchronous failures in the ECS Worker do not halt the synchronous API flow, maximizing uptime.
- **Zero Operational Overhead**: The architecture is entirely managed, eliminating the need for the operations team to handle server maintenance, patching, operating system management, and infrastructure security hardening, allowing the team to focus purely on business logic development.
- **Faster Time-to-Market**: Developers can deploy new features quickly using Infrastructure as Code (e.g., Serverless Framework or CloudFormation), accelerating the deployment lifecycle compared to provisioning and configuring traditional server clusters.

3.2 Limitations and Implementation Difficulties

Despite its many advantages, the Serverless architecture introduces specific technical limitations and complexity challenges that must be acknowledged and managed.

- **Vendor Lock-in**: The deep reliance on proprietary AWS services such as DynamoDB, SQS, SNS, and API Gateway creates a significant degree of vendor lock-in. Migrating the core logic or data persistence to another cloud provider would require substantial re-engineering of the entire event orchestration mechanism.
- **Cold Start Latency (Lambda)**: While generally low, the initial invocation of an idle AWS Lambda function (a "cold start") can introduce a brief, unpredictable latency spike (up to a few seconds) for the first synchronous API request. This must be closely monitored to maintain the required fast user experience.
- **Complexity of Monitoring and Debugging**: Troubleshooting issues in a highly decoupled, event-driven system is inherently complex. Debugging requires tracing an event across multiple boundaries (API Gateway, Lambda, SQS, ECS, and DynamoDB Streams), requiring specialized tools like AWS X-Ray rather than simple log file analysis.
- **Cost Predictability Challenges**: Although the pay-per-use model is cost-efficient, it can be difficult to accurately forecast costs under highly variable and exponential traffic loads, making budget planning challenging compared to the fixed monthly costs of traditional EC2 instances.
- **Resource Limits**: Serverless components, particularly AWS Lambda, are subject to specific quotas (e.g., execution time limits, memory limits) that constrain the complexity and duration of synchronous tasks. This necessitates careful architectural decisions, such as using ECS Fargate for long-running media processing, rather than relying solely on Lambda.

Part D: Evaluation, Cost Analysis, Conclusion, and References

4.1 Requirement Fullfillment Analysis

This part formally evaluates the proposed Serverless Multi-Tier Architecture against the core business and technical mandates, demonstrating that the design successfully addresses every critical requirement.

4.1.1 Scalability and Reliability

The requirement for infinite scalability and non-disruptive reliability is met by leveraging the inherent elasticity of the Serverless model.

- **Scalability**: Services like Amazon Lambda, DynamoDB, SQS, and Fargate automatically scale to zero during idle periods and instantly scale to handle massive loads without manual provisioning.

- **Reliability:** The deployment across multiple Availability Zones (AZs) ensures resilience, while the SNS/SQS decoupling isolates the heavy media processing tasks from the synchronous API path, preventing high latency or failure in one area from impacting the other.

4.1.2 Secure and Authorized Upload

The architecture guarantees a secure and authenticated process for all media uploads, maintaining data integrity and access control.

- **Authentication:** Amazon Cognito handles all user authentication, providing secure identity management and issuing temporary credentials (tokens).
- **Authorization:** The API Gateway integrates with the authentication process to enforce authorization before routing requests to Lambda.
- **Direct Upload:** Users are granted a pre-signed URL to upload media directly to Amazon S3 (Media Storage). This bypasses the application server entirely, minimizing load and eliminating the security risk associated with handling large files through the API Gateway/Lambda layer.

4.1.3 Efficient Asynchronous Processing

The system must handle large, long-running media tasks efficiently without locking up user resources.

- **Decoupling**: The use of Amazon SNS and Amazon SQS creates a robust message queue, ensuring that the initial synchronous API call can return an instant "Accepted" status to the user while the heavy processing task (like video transcoding) is queued securely.
- **Worker Specialization**: The architecture utilizes the right tool for the job: Lambda Workers handle quick, light tasks, while ECS Workers (Fargate) are reserved for containerized, resource-intensive, and long-running media processing tasks, maximizing efficiency.

4.1.4 Real-Time Status Feedback

The requirement for immediate feedback on the long-running process status is met using an event-driven notification mechanism.

- **Trigger Mechanism:** After a worker completes its task, it updates the status in Amazon DynamoDB. This update instantly triggers DynamoDB Streams (Change Data Capture).
- **Real-Time Push**: The change data invokes a Notifier Worker, which uses the persistent connection of the API Gateway (WebSocket Handler) to push the status notification directly to the client's browser. This prevents resource-intensive polling, ensuring timely, efficient, and non-blocking user feedback.

4.2 Cost Estimation

The Cost Estimation section is essential for validating the financial feasibility of the proposed Serverless solution. In the Cost Estimation part, my team decided to utilize various calculating methods to accurately

model the consumption-based cost structure and quantify the superior cost efficiency of the architecture, focusing specifically on eliminating the fixed costs typical of IaaS models. To achieve this, my team divided this section into three distinct parts including Traffic Assumptions, Monthly Cost Breakdown, and Cost Efficiency Analysis which are mock data.

4.2.1 Traffic Assumptions

To construct a realistic cost estimate, the team established clear traffic assumptions for a normal operational month. These figures are crucial for calculating consumption across the compute and data transfer components of the Serverless architecture.

Metric (Monthly)	Assumed Value	Calculation Unit	Note
Total API Requests	100 Million (M)	API Calls	Covers Synchronous Read/Write Metadata via API Gateway and Lambda.
Media Uploads	5 Million (M)	S3 Object PUTs	Triggers the asynchronous flow (SNS/SQS).
Heavy Processing Ratio	20%	Percentage	20% of events require heavy processing by ECS Worker (e.g., video transcoding).
Average Processing Time	15 Seconds	Seconds/ECS Worker	Average runtime for resource-intensive ECS Fargate tasks.
DynamoDB R/W Units	500 Million	Read/Write Capacity Units (RCU/WCU)	Utilizes On-Demand mode for DynamoDB scalability.
Data Out (CloudFront/S3)	10 Terabyte (TB)	GB/TB	Cost of content distribution via CDN and S3 access.

Figure 9: Traffic Assumptions

4.2.2 Monthly Cost Breakdown

Based on the ***Figure 9***, my team generated mock data in the the following table presents an estimated monthly cost breakdown, focusing on the major consumption services. All costs are estimated using standard AWS On-Demand pricing (excluding Free Tier).

A	B	C
AWS Service	Cost Calculation Factor	Estimated Cost (USD)
API Gateway	100M API Calls ×$3.50/M	350.00
AWS Lambda	(API Logic + Notifier + Lambda Worker) (≈200ms/request)	200.00
Amazon Dynam	500M RCU/WCU (On-Demand) + Storage	1,500.00
Amazon S3	50 TB Storage + 5M PUTs + Data Transfer	1,200.00
Amazon SQS/SI	5M Notifications ×2 (Standard/Fanout)	20.00
ECS Fargate	1M ECS Worker ×15 seconds CPU (Resource Intensive)	2,500.00
Amazon CloudF	10 TB Data Transfer Out (CDN)	500.00
NAT Gateway	Fixed Cost + Data Processing (≈5 TB)	450.00
Estimated Total Monthly Cost		6,720.00

Figure 10: Monthly Cost Table Mock Data

4.2.3 Cost Efficiency Analysis

The calculated cost demonstrates the architectural shift toward consumption-based pricing, validating the mandate for cost efficiency. As a result, my team predicted that the monthly expenditure of $6,720 USD for a high-traffic month is highly efficient because:

- **Elimination of Idle Costs**: The Serverless architecture completely eliminates fixed costs associated with idle servers (for instance dedicated EC2 instances or load balancers typical of Assignment 2). When application traffic is low, the compute cost approaches zero.
- **Optimal Resource Matching**: The most expensive resource, ECS Fargate, is only used for the \$20\%\$ of media uploads that require heavy processing. This specialization ensures that resources are not wasted on simple tasks.
- **Superior Price-Performance Ratio**: The payment is directly linked to the value created (API calls served, media processed). This provides a far more favorable Price-Performance Ratio compared to traditional IaaS, where costs remained fixed regardless of utilization**.**

4.3 Conclusions

The successful completion of this complex Serverless architecture assignment provided valuable insights for both the project leadership and the development team regarding cloud-native delivery:

- Learnings for the Leader: The leader gained proficiency in decoupled orchestration and risk management in event-driven systems. A crucial lesson was the necessity of early and detailed cost modeling in pay-per-use environments to maintain financial control, especially regarding potentially expensive services like ECS Fargate and NAT Gateway.
- Learnings for Teammates: The development team transitioned from monolithic thinking to microservice and function-as-a-service deployment. The primary technical growth areas included mastering DynamoDB access patterns for high-scale NoSQL and efficiently using AWS X-Ray for distributed tracing and debugging across decoupled components (Lambda, SQS, ECS).

The ultimate outcome of this assignment is a fully justified architectural proposal that achieves the highest standard of modern cloud design:

- Architecture Validation: The team successfully mapped the proposed design to every business mandate, proving that the Serverless architecture offers superior resilience, elasticity, and cost efficiency compared to traditional IaaS solutions.
- Future-Proof Foundation: The delivered design provides a future-proof, highly secure, and optimized platform ready to support exponential user growth and rapid feature iteration. By establishing clear separation between synchronous and asynchronous flows, the team laid the groundwork for robust, scalable product development.

Part E

References

(1) Lucidchart, “Cloud architecture diagrams guide,” 2023. [Online]. Available:

https://www.lucidchart.com/pages/cloud-architecture-diagram

[2] IBM, “What is three-tier architecture?,” IBM. [Online]. Available:
https://www.ibm.com/think/topics/three-tier-architecture. . [Accessed: Nov. 17, 2025].

[3] Amazon Web Services, “Global infrastructure,” AWS. [Online]. Available:
https://aws.amazon.com/about-aws/global-infrastructure/. . [Accessed: Nov. 17, 2025].