High Availability Web Server using Amazon Web Services

Savindu Wickramasinghe, Nidula Mallikarachchi, Thulith Hansana
Swinburne University of Technology, Australia
104834960@student.swin.edu.au, 104756611@student.swin.edu.au, 104852894@student.swin.edu.au

***Abstract*- The increasing demand for web services that are both scalable and resilient under varying loads necessitates an architecture that can adapt dynamically. This report introduces a comprehensive cloud-based solution utilizing Amazon Web Services (AWS) to address these needs. Our approach combines several AWS services to provide a scalable, secure, and fault-tolerant system architecture designed to support high-availability web servers, along with efficient handling of static and dynamic content, and real-time data processing and analysis.**

I. Introduction

In the era of cloud computing, the ability to provide reliable and scalable web services is significant. Enterprises seek architectures that not only support the elastic nature of web traffic but also incorporate robust security measures, data redundancy, and efficient processing capabilities. Amazon Web Services (AWS) offers a suite of scalable cloud computing solutions that can be used to construct such an environment. This report presents an architecture that leverages various AWS services to build a resilient web application platform.

The proposed architecture is designed within a Virtual Private Cloud (VPC) for network isolation and security. It utilizes AWS's Elastic Load Balancing (ELB) and Auto Scaling to distribute web traffic across multiple instances and zones, ensuring high availability and fault tolerance. Amazon Cognito is employed for user authentication, while AWS Identity and Access Management (IAM) provides fine-grained access control. AWS Shield and AWS Web Application Firewall (WAF) are integrated for protection against Distributed Denial of Service (DDoS) attacks and common web exploits.

For content delivery, Amazon CloudFront is used together with Amazon Route 53 for efficient DNS management and global content distribution. This setup ensures low-latency delivery of both static and dynamic web content. Amazon S3 serves as the repository for static resources, while the Elemental MediaConvert handles media processing tasks. All operational activities are monitored by AWS CloudWatch, and AWS CloudTrail audits are in place for compliance and governance.

In the context of data processing, AWS Lambda functions are triggered by Amazon Simple Queue Service (SQS) messages for asynchronous processing tasks. This setup facilitates the seamless transformation of uploaded content, which is then stored and managed in Amazon S3 buckets. For data persistence, Amazon DynamoDB provides a NoSQL database solution with built-in replication for durability, while AWS ElastiCache enhances database performance via caching.

The architecture also integrates Amazon Rekognition for image and video analysis, leveraging machine learning to identify objects and scenes. This capability is particularly beneficial for applications requiring content moderation, search and discovery, and user engagement.

A. Assumptions Made

To establish the foundation for our proposed architecture, we begin by outlining the core assumptions that have guided our design decisions. These assumptions are based on common industry practices, performance benchmarks, and prevailing security standards:

- Load Expectations: The architecture is designed assuming a web application with variable traffic patterns, including potential spikes during peak usage periods.
- Security Posture: We presume a threat model that includes protection against network and application-level attacks, in line with AWS best practices.
- Operational Scale: The system is presumed to be operating at a scale where cloud-based cost efficiencies become significant.We are presuming the number of users at this point in time is 15,000.

These assumptions delineate the boundaries within which our architectural framework has been conceptualized and should be considered when evaluating its applicability to specific use cases.

II. Architecture Design

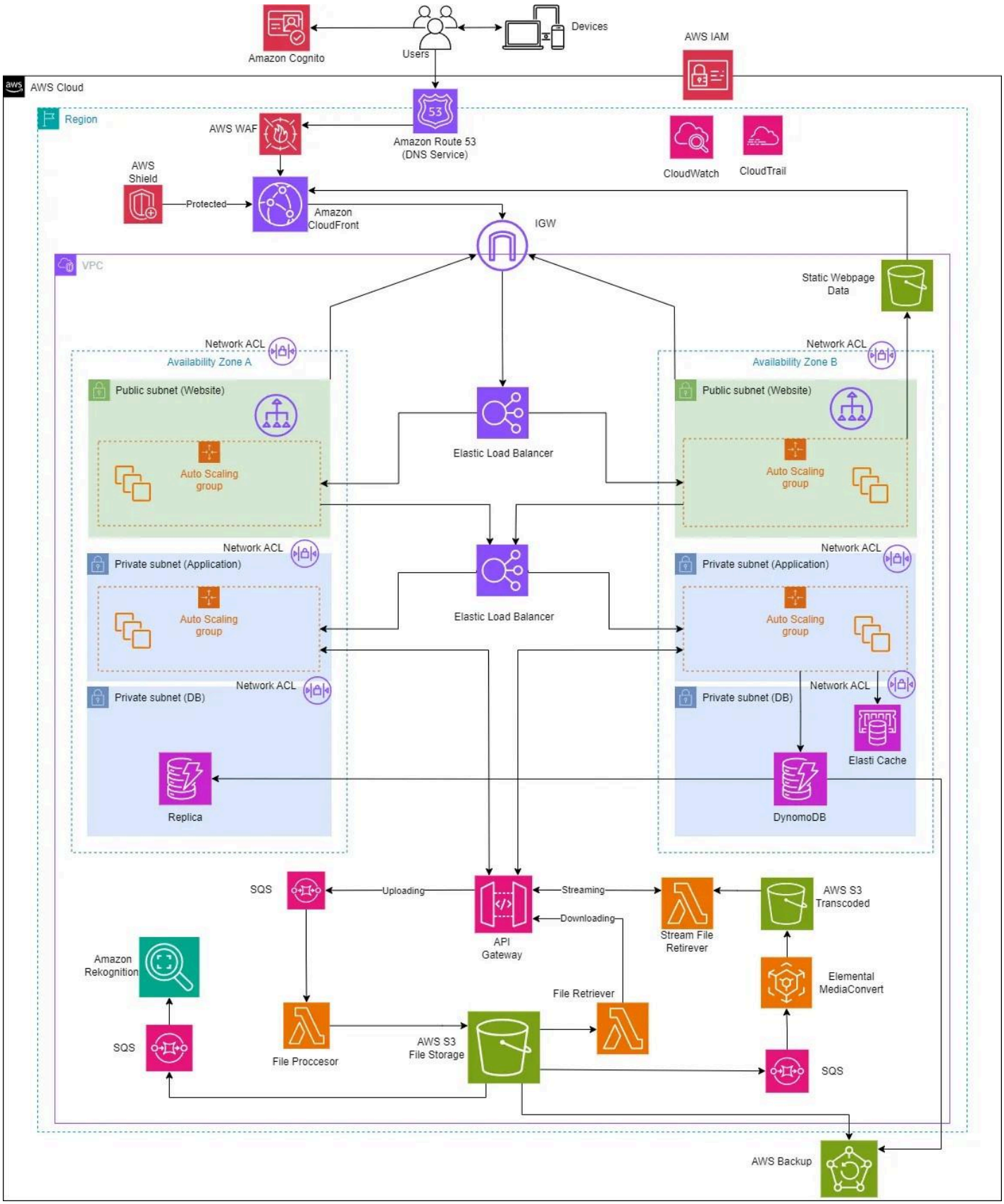

Figure 1.AWS Design Architecture

A. AWS Services Used

1) Amazon Virtual Private Cloud (VPC): Amazon VPC allows provisioning a logically isolated section of the AWS Cloud where it is possible to launch AWS resources in a virtual network that we define. We have complete control over the virtual networking environment, including selection of the IP address range, creation of subnets, and configuration of route tables and network gateways.

In this design, the VPC is the foundation of the cloud architecture. It provides a secure and isolated environment where resources such as EC2 instances can run. It also enables the creation of public and private subnets, which is essential for defining a network that segregates the publicly accessible components of our application from the backend systems. This separation enhances security and allows for more granular control over traffic access and flow.

2) Amazon CloudFront: CloudFront is a fast content delivery network (CDN) service that securely delivers data, videos, applications, and APIs to customers globally with low latency and high transfer speeds. CloudFront works by caching content in edge locations closer to our users, reducing the time taken to deliver content.

For a file storage and sharing service, speed and user experience are crucial. CloudFront is used here to cache and deliver the static content of the website, such as CSS and PHP files which are stored in a separate S3 bucket, quickly to users worldwide. This decreases load times, which is critical for user satisfaction. Additionally, CloudFront integrates well with other AWS services, like WAF and Shield, adding layers of security against common web threats.

3) Amazon Route 53: Amazon Route 53 is a highly available and scalable cloud Domain Name System (DNS) web service. It is designed to give developers and businesses an extremely reliable and cost-effective way to route end users to Internet applications by translating domain names (www.example.com) into the numeric IP addresses (192.0.2.1) that computers use to connect to each other.

Route 53 is used in this architecture to manage DNS entries and to route user requests to the infrastructure running in AWS. It can direct traffic to CloudFront for content delivery, ensuring that DNS queries are resolved with low latency by routing each request to the nearest location. This service is also robust in managing traffic for failover configurations, which is essential for maintaining high availability and performance of the website.

4) Elastic Load Balancing (ELB): Elastic Load Balancing automatically distributes incoming application traffic across multiple targets, such as Amazon EC2 instances and Lambda functions. It can handle the varying load of our application traffic across multiple Availability Zones.

ELB is a critical component of the architecture, used to distribute incoming traffic across multiple EC2 instances to improve the fault tolerance of the application. By using ELB, the system can handle peaks in workload or traffic more efficiently by balancing the load, which helps in maintaining smooth operation and optimizing the use of resources. It also monitors the health of the instances and only routes traffic to the healthy ones, further enhancing the reliability of the service.

5) AWS Auto Scaling: AWS Auto Scaling monitors the applications and automatically adjusts capacity to maintain steady, predictable performance at the lowest possible cost. It can be used to automatically increase the number of Amazon EC2 instances during demand spikes to maintain performance and decrease capacity during lulls to reduce costs.

In this design, Auto Scaling ensures that the computing resources match the demand of the file storage and sharing service without manual intervention. This is crucial for handling the unpredictable workload patterns, as user upload and download activities can vary significantly. By using Auto Scaling, the service can maintain performance levels regardless of usage spikes while optimizing costs by scaling down during quieter periods. We have set the minimum number of EC2 instances as 8 and the ability to scale up to 32 instances during peak usage. Hence horizontal scaling can be done with ease.

6) Amazon CloudWatch: Amazon CloudWatch is a monitoring and observability service that provides data and actionable insights to monitor our applications, respond to system-wide performance changes, optimize resource utilization, and get a unified view of operational health.

CloudWatch is integrated into this architecture to monitor resources and applications. It tracks metrics, collects log files, and sets alarms. For a high availability web server, this means having insight into the performance of the system, which is vital for maintaining operational health and ensuring a good user experience. It helps in identifying and responding to issues quickly, minimizing downtime.

7) AWS Lambda: AWS Lambda allows us to run code without provisioning or managing servers.Payment is only for the computer time we consume - there is no charge when our code is not running. With Lambda, its possible to run code for virtually any type of application or backend service with zero administration.

Lambda functions are used in this architecture for their serverless execution of event-driven processes. They can be triggered by AWS services like S3 upon file uploads to process or transform data (like generating thumbnails or extracting metadata). This eliminates the need for dedicated servers for these tasks, reducing costs and management overhead. The scalability of Lambda also ensures that as the number of files or the processing needs grow, the infrastructure adapts automatically.

8) Amazon Simple Storage Service (S3): Amazon S3 is an object storage service that offers industry-leading scalability, data availability, security, and performance. It allows the storage and protection of any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, and enterprise applications. S3 provides easy-to-use management features so we can organize the data and configure finely-tuned access controls to meet specific business, organizational, and compliance requirements.

S3 is the storage backbone of this design architecture. It's used to store the static web objects that make up the website, such as CSS and PHP files, as well as the user files that are uploaded to the service. Its high durability and availability make it ideal for this purpose. S3 also integrates well with other AWS services, such as CloudFront for content delivery and Lambda for executing code in response to events like file uploads, making it a versatile component of the architecture.

9) AWS Identity and Access Management (IAM): IAM allows security personnel to manage access to AWS services and resources securely. Using IAM, it's possible to create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.

IAM is critical for managing the security of the services in the cloud architecture. It's used to control who can access what resources within the AWS environment. This is particularly important for this design where it is needed to ensure that only authorized users and services can access or modify the user data or the application configuration. By using IAM, we can follow the principle of least privilege, ensuring that each component and user has only the permissions necessary to perform its tasks, thereby enhancing the overall security posture of the service. Amazon Cognito can be used together for a seamless experience for the developers.

10) Amazon Cognito: Amazon Cognito provides authentication, authorization, and user management for web and mobile applications. Users can sign in directly with a username and password, or through third parties such as Google, Facebook, or Apple.

In this design, Cognito is essential for managing user identities and granting the correct access to user data within the platform. It allows the service to scale securely, handling potentially hundreds of thousands of users, and their authentication data. Cognito also integrates with other AWS services, enabling developers to create a seamless and secure user experience across the application.

11) Amazon DynamoDB: Amazon DynamoDB is a managed NoSQL database service acclaimed for its low-latency performance and scalability. The service excels in providing a durable database without the traditional burdens of manual setup, ensuring operational efficiency and developer ease.

In our design, DynamoDB serves as the backbone for storing and retrieving various types of structured data. Its use is justified by the need for high availability and quick data access, which are crucial to delivering a seamless user experience and maintaining the responsive performance of our services. Data stored in DynamoDB includes,

a) *User information:* This is mainly User IDs, authentication tokens from Amazon Cognito, and subscription types. DynamoDB's ability to handle high-velocity data makes it ideal for managing user identities and credentials.
b) *File metadata:* Attributes like file ID, name, type, size, and timestamps related to creation and last modification, as well as ownership, sharing, and encryption status. DynamoDB's fast read and write capabilities facilitate real-time access and updates to this metadata.
c) *Access permissions:* Records of access permissions, combining user and file or folder IDs with access levels. The use of DynamoDB ensures that permission checks are swift and do not become a bottleneck.
d) *Activity logs:* Storing logs with details of user activities, timestamps, IP addresses, and device IDs. The scalability of DynamoDB is vital here to handle potentially large volumes of log data.
e) *Sharing information:* Data about shared files or folders, including the originating and recipient user details, permissions, and expiration dates. DynamoDB's capacity to quickly serve these records is essential for sharing features.

12) Amazon ElastiCache: Amazon ElastiCache is a fully managed in-memory caching service, providing fast, scalable caching for our cloud applications. It supports two open-source in-memory caching engines: Redis and Memcached. ElastiCache facilitates the easy setup, management, and scaling of these in-memory cache environments in the AWS Cloud.

The primary purpose of ElastiCache within our architecture is to enhance the performance of our applications by allowing quick retrieval of information. By caching frequently accessed data, ElastiCache reduces the need to perform expensive read operations on the database. This is particularly beneficial when dealing with high-demand web applications where speed and latency are critical factors.The service is utilized for a variety of functions in the design,

a) *User Session Data:* ElastiCache stores user session information, which enables faster access and improved responsiveness, ensuring a seamless user experience.
b) *Recently Accessed Files:* By caching the metadata or content of recently accessed files, ElastiCache reduces the load on the file system and databases, speeding up file retrieval processes.
c) *File Thumbnail Links:* Thumbnails, often used in file browsing, are cached to speedup the display of directories containing images or videos each time they are accessed.
d) *Directory Listings:* Common directory listings are cached to provide rapid display of folder contents, enhancing the user's navigation experience.
e) *Access Patterns:* Understanding and storing the patterns in which users access data can improve the system's predictive loading of content, leading to a more efficient user experience.
f) *User Preferences:* Caching individual user preferences allows for quick application personalization, contributing to a more tailored and responsive user interface.
g) *Collaboration Data:* For applications that involve user collaboration, caching relevant data can reduce the access time and increase the speed of collaborative actions.
h) *Search Indexes:* ElastiCache can store parts of search indexes to facilitate quicker search operations, which is vital for applications where search functionality is heavily utilized.
i) *Rate Limiting Counters:* To enforce rate limiting within the application, ElastiCache can be used to store and update counters without imposing a high load on the database.

13) AWS Elemental MediaConvert: Elemental MediaConvert is a media transcoding service in the cloud. It converts media files into different formats suitable for various devices, like smartphones, tablets, and desktops, ensuring compatibility and optimal playback quality.

For a high availability webs server that deals with media, the Elemental MediaConvert is critical. It ensures that uploaded videos can be viewed on any device, regardless of their original format, by transcoding them into universally playable versions. This service is key for providing a seamless user experience, where users can access their media anytime, on any device, without worrying about format compatibility.

14) AWS Rekognition: AWS Rekognition is an image and video analysis service that uses machine learning to identify objects, people, text, scenes, and activities, as well as detect any inappropriate content that would violate our terms of service.

Rekognition is integrated to automate the scanning of images and videos for content that violates the terms of service as well as tagging uploaded images. This proactive approach to content moderation is essential for maintaining platform integrity and user trust, especially when user-generated content can scale rapidly. This will also result in lower administration costs due to lower staffing required for content moderation.

15) Amazon Simple Queue Service (SQS): Amazon SQS is a managed message queuing service that enables us to decouple and scale microservices, distributed systems, and serverless applications. SQS eliminates the complexity and overhead associated with managing and operating message-oriented middleware, and it allows developers to focus on differentiating work. It provides a secure and durable host for messages that may be processed asynchronously.

By utilizing SQS, the system places a message in the queue representing the task to be performed, rather than executing it right away. This message can then be processed independently by worker processes, which are running on Lambda functions. This decoupling allows the upload process to be swift and responsive for the user, as they're not waiting for all the processing to complete.

Moreover, using SQS aids in smoothing out the workload. If there is a surge in uploads, messages in the queue can be processed at a steady rate without overloading the compute resources. This ensures that the system remains responsive even under high load, enhancing the overall reliability of the service.

16) AWS EC2: Amazon EC2 (Elastic Compute Cloud) is the backbone of computing power within the AWS ecosystem, providing scalable virtual servers. In this design, EC2 serves as the driving force behind application hosting, offering customizable and on-demand computing resources.

The justification for its utilization in this architecture hinges on several core benefits , Scalability wise EC2 instances can be scaled up or down automatically with Auto Scaling groups to match the required load, which is essential for handling traffic surges and maintaining a seamless user experience.Also with a broad selection of instance types, EC2 allows us to tailor the hardware to the specific needs of our application, whether that's CPU, memory, storage, or networking capacity use of EC2 is flexible as well.We have prompted to use the M5.large in the start since, since that family of EC2 lineup have balanced CPU and memory it provide a good balance of compute, memory, and networking resource.

Deploying instances across multiple Availability Zones ensures high availability and fault tolerance, protecting the application against outages and disruptions ensures reliability. EC2's integration into our architecture underpins its capability to deliver content and services reliably, maintaining the robust performance that users expect from modern web applications.

17) API Gateway: In this architecture, Amazon API Gateway stands as a critical component. It acts as the front door to our web services, it centralizes the entry points for all inbound API traffic. The API Gateway manages and authorizes every call, ensuring that only legitimate requests reach our backend services, thereby enhancing security.

The use of API Gateway in this design also simplifies the management of traffic by allowing us to define throttling rules that prevent our backend services from being overwhelmed during peak times. Moreover, API versioning capabilities mean we can introduce changes and new features seamlessly without disrupting existing services.

Choosing API Gateway over direct service exposure to the internet is justified as it provides a controlled, secure, and efficient way to handle API traffic. This approach aligns with best practices for API management by leveraging AWS's robust infrastructure to safeguard against unauthorized access and maintain service integrity.

B. Aligning AWS Service Selection with Business Goals

1) Managed Cloud Services: The core storage solution for this architecture is based on Amazon S3, which is a managed service that eliminates the need for extensive in-house systems administration. This helps to minimize administrative overhead as S3 is capable of integrating with other AWS services. Additionally, services like AWS Lambda for serverless computing and Amazon Rekognition for AI-based media analysis also aid in reducing the burden on internal resources. Amazon Rekognition can be used to identify users who might violate the terms of service of this service.

2) Scalability for Growth: Utilizing AWS services like S3, Lambda, and EC2 with Auto Scaling, the architecture is skillfully designed to accommodate the expected growth. This setup ensures seamless scalability in response to user demand, which is expected to double every six months. Auto Scaling efficiently adjusts the number of EC2 instances or Lambda function invocations, aligning with traffic fluctuations. This elastic nature of AWS services guarantees that the architecture is well-prepared to handle increasing loads, ensuring consistent performance and reliability for the coming years.

3) Performance Optimization: To optimize compute capacity and address EC2 performance issues, the architecture uses AWS Auto Scaling, which dynamically scales compute resources to match demand, aiming to maintain the workload within a desired 50-60% capacity range. This approach may involve resizing EC2 instances to a more suitable type, guaranteeing a more effective balance of the compute load and keeping performance consistently below the 80% threshold. Additionally, considering a serverless solution, such as Lambda functions, offers an alternative optimization method. This serverless approach inherently resolves scalability and management concerns, as Lambda automatically adjusts to demand, and thereby further streamlining performance optimization.

4) Serverless/Event-Driven Architecture: Our new architecture utilizes AWS Lambda to create a serverless, event-driven solution, which aligns perfectly with the company's goal of adopting an event-driven model. With Lambda functions, the system can automatically respond to specific events, such as file uploads to S3 or media uploads. This reduces the need for running servers continuously, which not only cuts down on costs and administrative overhead but also ensures that the system is inherently scalable. This setup is well-suited for efficiently handling variable workloads, making it ideal for achieving a cost-effective, serverless architecture.

5) Database Efficiency: To enhance the efficiency of the database and tackle issues related to speed and cost, we can move the architecture from a traditional relational database to Amazon DynamoDB, a managed NoSQL database service as well as a fast ElastiCache. DynamoDB provides fast and scalable performance, which makes it especially suitable for simple table structures. This transition not only offers a more cost-effective solution but also provides the added benefit of DynamoDB's seamless scaling capabilities, ensuring that the database can effectively manage increasing data requirements and varying workloads without compromising performance with the help of ElastiCache for lower response time and speed.

6) Global Response Times: Amazon CloudFront is a content delivery network (CDN) that is used to boost global response times. This incredible service speeds up content delivery by caching data at edge locations all over the world, which significantly reduces latency. By delivering media and other content from the closest geographical location to each user, CloudFront ensures faster and more efficient access to data worldwide, resulting in a noticeable improvement in overall user experience.

7) Extension to Video Media and Live Streaming: The architecture has been designed with the ability to support video media and live streaming, keeping in mind future expansion needs. This has been made possible by integrating AWS Elemental MediaConvert. AWS Elemental MediaConvert efficiently handles video transcoding, ensuring that video content is optimally formatted for various devices. Simultaneously, AWS Media Services enables high-quality live streaming. Together, these services ensure that the architecture is not only capable of storing and processing video content alongside photos, but is also well-equipped for efficient video streaming and transcoding, meeting the demands of future scalability.

8) Media Processing and Transcoding: The architecture is designed to process media efficiently and automatically, ensuring responsiveness and scalability. Whenever media is uploaded to S3, AWS Lambda functions are triggered automatically, which coordinates with services such as AWS Elemental MediaConvert to create various media versions. This includes creating lower-resolution files for mobile devices. This automation streamlines the media handling process.

The design of the architecture is flexible, allowing for future integrations with advanced services like Amazon Rekognition for AI-powered tagging. Such additions can be made without major overhauls, thanks to the use of Lambda functions and the extensible nature of the system.

Different processing tasks are allocated to the most suitable platforms within AWS's ecosystem. While Lambda handles lighter tasks efficiently, heavy-duty operations like intensive video transcoding are assigned to more powerful specialized services like AWS Elemental MediaConvert. This approach ensures that each operation is executed in the most cost-effective and performance-efficient manner. By utilizing SQS, the system maintains application responsiveness, even under heavy processing loads.

C. Decoupling and Load Management

1) AWS Lambda: Lambda functions are inherently decoupled, as they are stateless and can be invoked in response to a variety of events. By using Lambda, we can create small, single-purpose functions that respond to specific triggers (e.g., a new file uploaded to S3), which means that a change or failure in one function doesn't directly impact others.This also allows to easily integrate with other AWS services and create workflows that automatically pass data between services without creating tight dependencies.

2) Amazon Simple Queue Service (SQS): SQS acts as a message buffer between the producers and consumers of messages. When an event occurs (like a file upload), a message is placed in an SQS queue. Worker nodes or Lambda functions then process messages from the queue when they are available. This decouples the process that generates the event from the process that handles the event.

3) Amazon S3 Event Notifications: With S3 event notifications, we can decouple the storage of the files from the processing of them. When a file is uploaded to S3, an event is emitted that can trigger a Lambda function, place a message in SQS, or send a notification through SNS. The processing services can then handle the data as needed without being tightly integrated with the storage service.A system would be implemented to ensure that we have appropriate filtering in place so that only relevant events trigger the workflows, which optimizes the processing and prevents unnecessary invocations of Lambda functions or messages being sent to SQS.

4) AWS Elemental MediaConvert: For media processing, these services are used to decouple the CPU-intensive task of video transcoding from our main application flow. We trigger these services through events, and they run independently, scaling as needed to process media files without impacting the performance of our core application.

5) AWS CloudFront: CloudFront decouples the delivery of content from its origin. By caching content at edge locations closer to the end-users, CloudFront handles the distribution independently of where the content is stored or generated.

D.UML Diagram

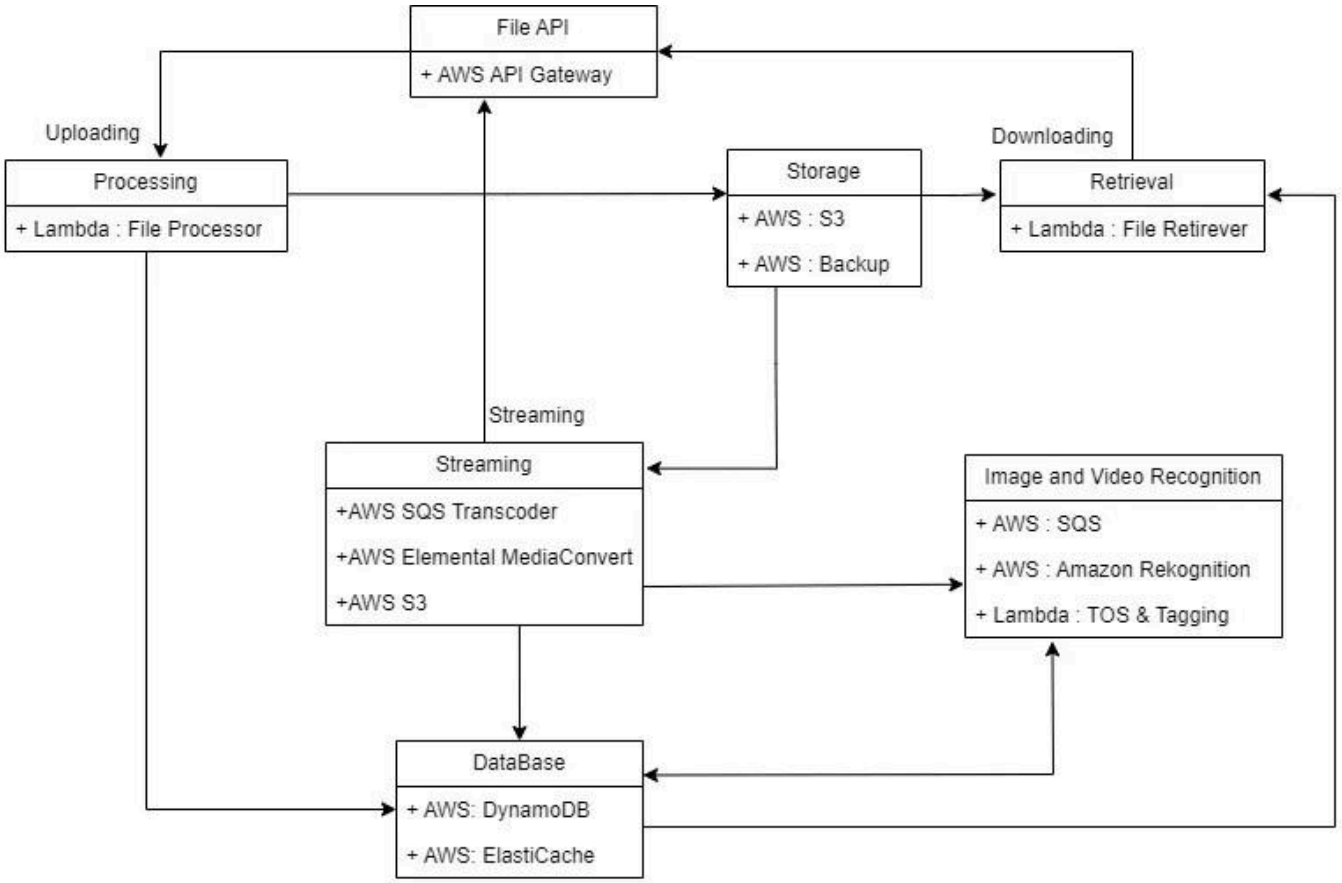

Figure 2. High level overview of Request Pattern

Fig 2. Depicts a high level overview of Users connected to the web server which sends requests to the File API, powered by AWS API Gateway it controls file traffic, sending uploads to Lambda for processing, then to S3 for storage. The content's sent via SQS Transcoder and MediaConvert, while DynamoDB and ElastiCache manage database interactions. AWS Rekognition, assisted by Lambda, handles image and video recognition, feeding data back to the database. For downloads, Lambda retrieves files from S3, completing the cycle.

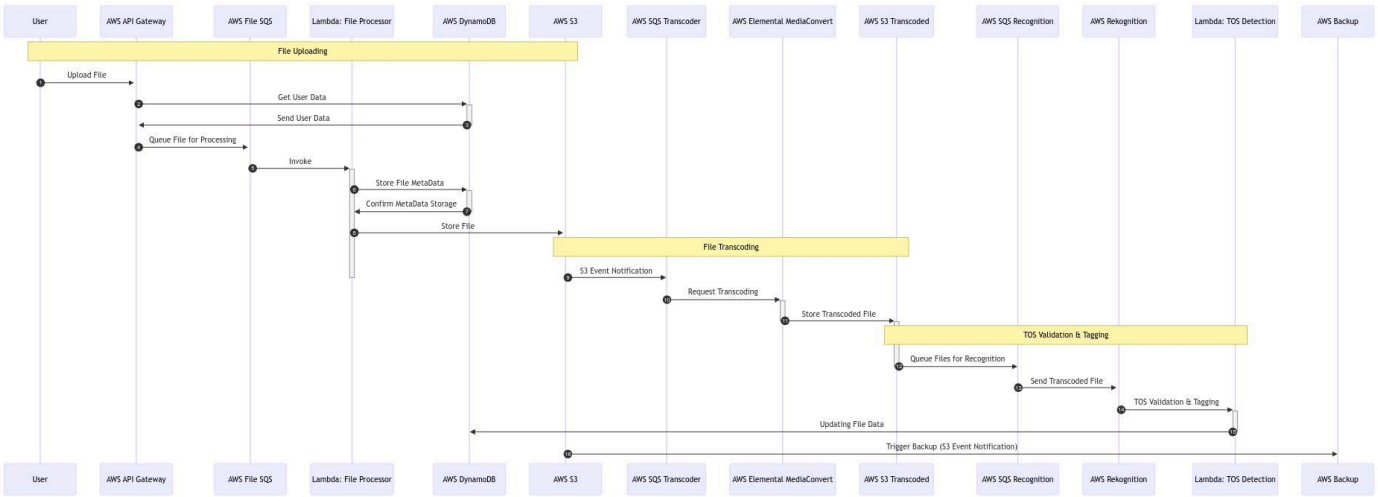

Figure 3. UML Sequence Uploading

Fig 3. The user uploads a file through AWS API Gateway, which queues it to Lambda to process. Lambda Function gets metadata in DynamoDB and it stores the file in S3 as well, triggering an S3 Event. This prompts AWS Elemental MediaConvert to transcode the file and make the thumbnail, then stored back in S3. SQS queues the transcoded file for AWS Rekognition for TOS checks and image Tagging, while AWS Backup safeguards the data.

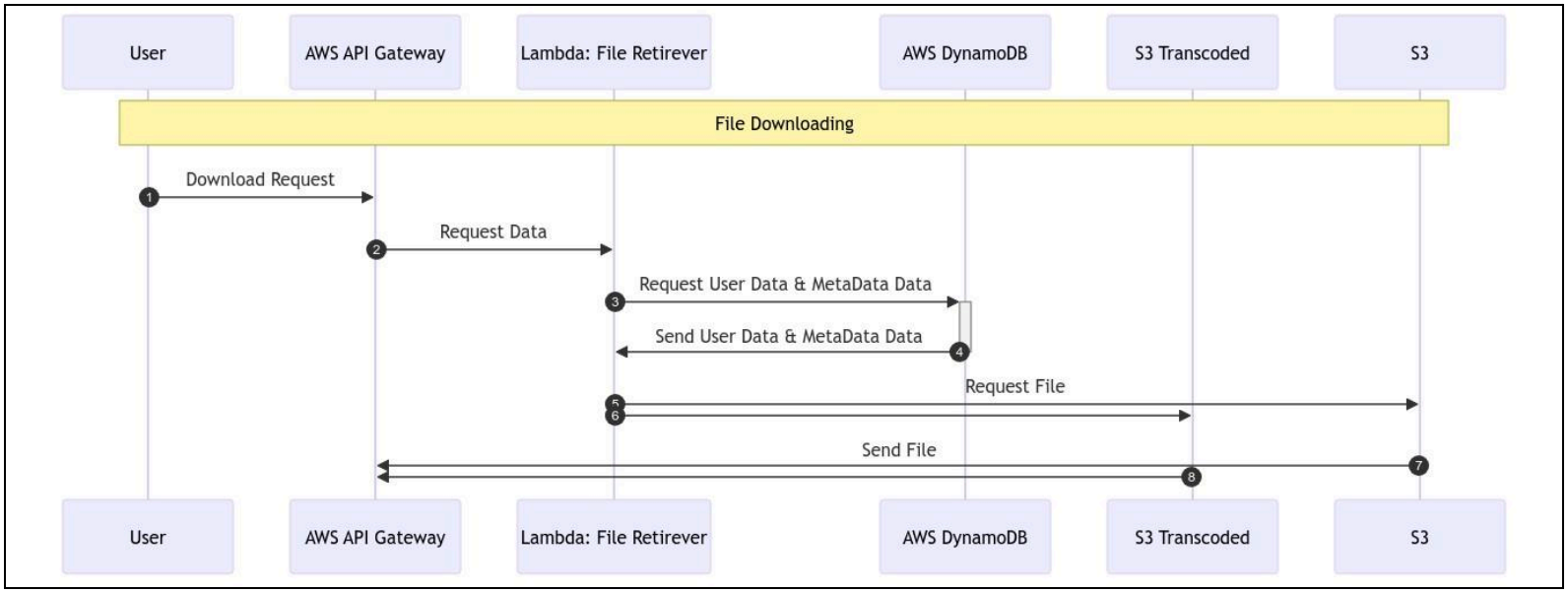

Figure 4. UML Sequence Downloading

Fig 4. The user initiates a download request via AWS API Gateway, which triggers a Lambda Function to retrieve the necessary user and metadata from DynamoDB. Lambda then fetches the requested file from S3, whether it's the original or a transcoded version, and delivers it back to the user through the API Gateway.

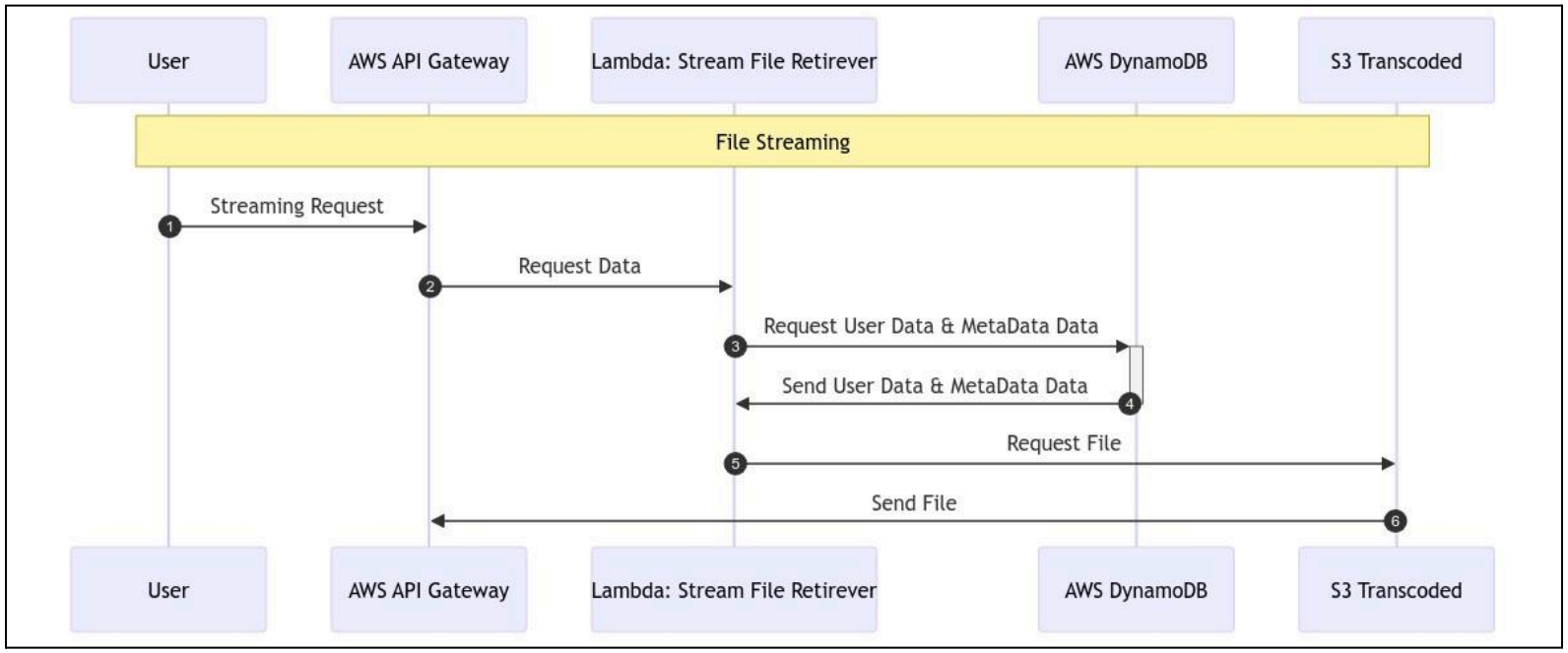

Figure 5. UML Sequence Streaming

Fig 5. Upon a streaming request from the user, AWS API Gateway engages a Lambda Function to fetch the file. Lambda retrieves user data and metadata from DynamoDB, then requests the actual file from S3 Transcoded. Once retrieved, the file is streamed back to the user through Lambda and API Gateway, completing the process.

III. Design Rationale

A. Performance

In our AWS architecture proposal, we've designed a system focused on high-availability and exceptional performance for web services. At its core, this architecture integrates well-chosen AWS services, creating a resilient and scalable system. Amazon CloudFront, our chosen global CDN, significantly enhances performance by caching content near users, reducing latency, and improving load times for an accelerated user experience. This efficiency reduces the load on origin servers, bolstering system availability.

To distribute incoming traffic effectively, we employ Elastic Load Balancers across multiple availability zones. This setup ensures balanced load distribution across EC2 instances, avoiding bottlenecks and facilitating uninterrupted service even during instance failures. Additionally, Auto Scaling plays a critical role in maintaining consistent performance, automatically adjusting EC2 instances in response to varying traffic loads. This feature is pivotal in managing traffic spikes without manual intervention, ensuring consistent performance.

This design's data layer is strengthened with Amazon DynamoDB and ElastiCache, utilizing Multi-AZ deployments for DynamoDB to ensure fast, predictable performance with high resilience. Amazon S3's robust storage capabilities further enhance the system's durability. The AWS Region and Availability Zone model fortifies our architecture, minimizing the impact of potential failures. Amazon Route 53's DNS service and traffic management add an extra layer of robustness, efficiently directing user requests. Overall, this architectural design demonstrates our commitment to delivering a high-availability web server that is not just resilient but also adaptive and responsive to user demands, ensuring a seamless and efficient user experience.

B. Reliability

The architecture’s reliability is established through a multi-tiered strategy. Firstly, the use of Elastic Load Balancing (ELB) across multiple Availability Zones ensures that incoming traffic is distributed evenly and that the service remains available even if one zone experiences issues. Secondly, the Auto Scaling groups for EC2 instances provide the ability to dynamically adjust resources, which is critical for handling sudden traffic surges and maintaining service continuity.

Additionally, Amazon RDS with Multi-AZ deployments for databases and Amazon S3's highly durable storage, with 99.999999999% durability and 99.99% availability, ensure that data is replicated and can be quickly recovered in case of a failure. Amazon Route 53 effectively manages DNS, routing users to the most available and low-latency endpoints, further cementing the system’s reliability.[10]

Reliability in the given architecture is achieved through redundancy and failover strategies intrinsic to AWS services. The multi-AZ deployment of resources ensures that even in the event of a zone disruption, the system remains operational, with traffic seamlessly rerouted to healthy zones. The highly durable nature of Amazon S3, coupled with Amazon RDS's automated backups and failovers, guarantees that data integrity and availability are not compromised. Moreover, Amazon Route 53’s intelligent DNS routing further bolsters the reliability, directing users to the best available endpoint.

C. Security

Security is approached in a layered fashion. At the perimeter, AWS WAF and AWS Shield provide defenses against common web exploits and DDoS attacks, ensuring that the infrastructure is shielded from malicious traffic. Amazon CloudFront further contributes by serving content through secure HTTPS connections, mitigating the risk of data interception.

Amazon Cognito offers secure user identity management, adding an authentication layer that safeguards against unauthorized access. Within the network, VPC Network ACLs act as virtual firewalls for EC2 instances and other resources, offering fine-grained ingress and egress controls.

Furthermore, AWS IAM plays a critical role by ensuring that principle of least privilege is adhered to, granting only the necessary permissions required for each service to operate. AWS CloudTrail provides governance, compliance, and risk auditing by logging and monitoring account activity.

By integrating these security measures, the architecture not only protects against a wide spectrum of security threats but also ensures that the system can reliably withstand and recover from any disruptions. This comprehensive approach to reliability and security demonstrates a strong commitment to maintaining a robust web service platform that users can trust.

The security framework within the architecture is multi-layered, ensuring robust protection. AWS WAF and Shield form the first line of defense, safeguarding the infrastructure from web attacks and DDoS threats. Amazon CloudFront's encrypted content delivery further secures data in transit. At the core, Amazon Cognito provides user authentication, helpted by AWS IAM's meticulous access controls. Network ACLs and Security Groups create a secure network environment, while AWS CloudTrail's monitoring capability adds a layer of operational security, ensuring actions are traceable and auditable. This comprehensive security strategy not only defends against a spectrum of cyber threats but also gives confidence in the system's ability to withstand and recover from disruptions.

D. Cost Analysis

1) Fixed Costs:
   a) Amazon Shield: $3,000.00 USD per month.

2) Variable Costs & Assumptions:
   a) Amazon Cognito: Configured for 15,000 MAUs, with advanced security features disabled. Cost: $44.25 USD per month. Amazon CloudFront: Setup for 10 TB data transfer out to the internet and 5 TB to the origin per month, with 10 million HTTPS requests. Cost: $973.80 USD per month.
   b) Amazon S3: Usage of 300 TB of standard storage and 60 TB data retrieval with S3 Select per month. Cost: $6,942.61 USD per month.
   c) AWS WAF: 4 Web ACLs, 10 rule groups, and 10 managed rule groups each per Web ACL. Cost: $546.00 USD per month.
   d) AWS Shield: For CloudFront and Elastic Load Balancing, 10 TB per month. Cost: $3,256.05 USD per month (Fixed $3,000.00 + $256.05).
   e) Amazon DynamoDB: Tables using Standard-Infrequent Access class, 10 TB storage, and average item size of 10 KB. Cost: $1,024.00 USD per month.
   f) Amazon Elasticache: Average cache data size of 1 GB and average data transfer per request of 20 KB. Cost: $616.27 USD per month.
   g) Amazon Rekognition: Image processing for 200,000 images and label detection for 10 million images per month. Cost: $1,160.00 USD per month.
   h) Amazon VPC: Includes 2 NAT Gateways and other networking configurations. Cost: $987.30 USD per month.
   i) AWS Lambda: 1 million requests per month with 1 GB allocated ephemeral storage. Cost: $100.67 USD per month.
   j) Amazon EC2: m5.large instance with Linux OS, considering daily workload. Cost: $461.36 USD per month.
   k) AWS Elemental MediaConvert: 1,500 hours of output usage per month, HD and SD video resolution. Cost: $2,538.00 USD per month.
   (Total Monthly Cost: $19,281.63 USD.
   Annual Cost: $231,379.56 USD.)

3) Justification for Cost-Effectiveness: The current AWS cost structure is cost-effective because it avoids the need for large initial investments in physical infrastructure, which is especially helpful for startups and growing businesses. The pay-as-you-go model means we only pay for the resources we use, avoiding wasted spending on underused hardware. Additionally, AWS's managed services reduce operational and labor costs by eliminating the need for a large IT staff to maintain and update servers. This can lead to significant savings, considering the complexity and manpower required to manage such infrastructure. Furthermore, the AWS pricing model offers lower unit costs as usage increases, taking advantage of economies of scale, which is particularly beneficial as the service grows in size and scope.

4) Cost with Scaling in Mind for an Expansion to 960,000 Users: With the goal of expanding the user base to 960,000 users over three years by doubling users every 6 months, AWS's infrastructure is built for scalability. This allows for a smooth transition from serving 15,000 to 960,000 users. AWS offers various services, such as Reserved Instances or Savings Plans, which can lead to long-term cost savings through discounts for committed usage. Using serverless architectures like AWS Lambda can further reduce costs by automatically scaling with demand, ensuring payment for only necessary resources. Additionally, AWS's bulk pricing options and potential for custom pricing negotiations could significantly lower costs at high volume levels. AWS also provides advanced tools for real-time cost monitoring and optimization.

IV. Alternative solutions

A. Virtual Machines vs. Containers vs. Serverless Computing

1) Virtual Machines (VMs): Offers complete isolation with the operating system-level separation. As an alternative to AWS EC2 Auto Scaling groups, VMs can be managed using services like Amazon Lightsail or directly on EC2 without Auto Scaling. The advantage of using standalone VMs is the granular control over the environment, which can be crucial for legacy applications or specific compliance requirements. However, they typically result in higher costs due to underutilization and require more management overhead for scaling and resilience [1] .

2) Containers: Provides a lightweight alternative, allowing for more efficient resource utilization than VMs. Running containers in Amazon ECS without Auto Scaling might offer more straightforward orchestration and potentially lower costs due to higher density deployment. But this approach might not handle sudden spikes in traffic as gracefully as an Auto Scaling setup and may require manual scaling.

3) Serverless Computing: Like AWS Lambda, abstracts the server layer entirely, offering a pay-as-you-go model that can scale automatically to the exact demand. Serverless is an excellent alternative for certain workloads within this architecture, especially for the File Processor and Rekognition components, due to its event-driven nature and scalability. However, serverless can introduce challenges in debugging and might have limitations with long-running processes compared to containerized or VM-based solutions [2].

B. SQL vs. NoSQL Database

1) SQL Databases: Such as Amazon RDS (Relational Database Service), offer structured schema and powerful query languages, which are excellent for complex queries and transactions. They are a traditional choice for systems that require strict data integrity and complex joins. The trade-off, however, often comes in terms of scalability and performance under high-load web applications, as they might require additional effort to scale horizontally.

2) NoSQL Databases: Such as Amazon DynamoDB, on the other hand, provide flexibility in terms of data models. They are designed to scale out by distributing the data across multiple nodes, which can handle high throughput and provide low latency access to large volumes of data. This makes DynamoDB an ideal candidate for applications with less complex query requirements but a need for rapid scaling and performance [3].

C. Caching Options

1) In-memory Caches (Redis or Memcached): These are powerful for scenarios requiring rapid access to data, such as session storage or real-time analytics. Redis, with its rich data structures, is suitable for complex tasks like leaderboards or counting. Mem-cached, being simpler and multi-threaded, is often used for straightforward caching scenarios. Both offer lightning-fast data access, but Redis provides more advanced features like persistence and replication.

2) Content Delivery Networks (CDNs): A CDN, like Amazon CloudFront, caches static and dynamic web content geographically closer to users, speeding up the delivery. It’s an efficient way to reduce latency and offload traffic from origin servers. CDNs are excellent for global reach but are less suited for personalized content that can't be cached effectively.

3) Database Caching: This involves temporarily storing frequently queried data directly within the database layer. It's beneficial for reducing repetitive query load, but it's usually less flexible and slower than in-memory caching [4].

D. Push vs. Pull Message Handling Options to Promote Decoupling

1) Push-Based Messaging: In this model, messages are actively pushed to the receiving service or component as they arrive. An AWS service that facilitates this is SNS (Simple Notification Service), which can directly invoke serverless functions or notify end-users. The push model ensures real-time data flow, which is crucial for time-sensitive operations. However, it can lead to overloading the consumer if the message rate exceeds the processing rate.

2) Pull-Based Messaging: This is the current model implemented using SQS, where the consumer polls and retrieves messages as needed. This method inherently regulates the processing rate, as consumers fetch messages at their own pace. While this can introduce slight delays versus a push model, it provides a more controlled processing environment and can help to manage and scale workloads effectively without overwhelming the consumers.

E. Number of Tiers in the Architecture, e.g., 3-tier vs 2-tier

1) 3-tier Architecture: This is a widely embraced model with separate layers for presentation, business logic, and data storage, each running on different servers. Our design utilizes this structure, with Amazon Cognito and Route 53 at the front end, EC2 instances and Lambda functions as the middle tier for business logic, and DynamoDB and S3 for data persistence. The benefits include enhanced security, as each layer acts as a gatekeeper to the next, and improved scalability, since layers can be scaled independently. A potential downside could be the increased complexity in setup and management, but AWS services are designed to streamline these processes.

2) 2-tier Architecture: This model combines the presentation and business logic layers or the business logic and data storage layers. It can simplify the architecture and reduce latency by minimizing the number of hops between layers. However, it may not scale as effectively when dealing with complex applications or large amounts of traffic, as the coupled layers can become a bottleneck.

F. Object Storage vs. Block Storage vs. File Storage

1) Object Storage (Amazon S3): Is perfect for storing and retrieving any amount of data from anywhere. It shines with its simplicity, scalability, and data availability. It's inherently designed for high durability, archiving, backup, and serving web content. Given that our system involves web applications that likely handle large amounts of unstructured data, S3 is the most fitting solution. It offers a balance of performance, cost-effectiveness, and ease of use, making it the optimal choice for the architecture.

2) Block Storage: Services like Amazon Elastic Block Store (EBS) provide block-level storage volumes for use with EC2 instances. It's akin to a traditional hard drive, allowing us to store data in raw block form. This is ideal for databases or applications that require a file system and perform frequent read and write operations. However, it's not as easily scalable or accessible as object storage and typically comes at a higher cost.

3) File Storage: Amazon Elastic File System (EFS) is a managed file storage service for use with AWS Cloud services and on-premises resources. It's excellent for use cases where we need a traditional file system, shared access, or to store and manage files in a hierarchy. But it may not match the performance of object storage for web-scale applications or provide the same level of durability and availability as S3.

G. API Gateway vs. Direct Service Exposure

1) Direct Service Exposure: This approach involves services being accessed directly over the internet without an intermediary. The upside here is that it can reduce complexity and latency, as there are fewer layers between the client and the service. However, this simplicity comes with significant drawbacks, such as the increased burden on individual services to handle security, traffic management, and scaling, which can lead to potential vulnerabilities and additional maintenance overhead.

2) API Gateway: Using an API Gateway, as chosen in this architecture, offers several substantial benefits. It acts as a robust gatekeeper, providing an additional layer of security by enabling us to implement authorization checks, request validation, and rate limiting to protect the backend services. It simplifies the developer experience by offering version control, API key management, and a single entry point for multiple services. The Gateway also readily scales to accommodate varying levels of traffic, ensuring that the services remain responsive and available.

H. Load Balancer Types (Application vs. Network Load Balancer)

1) Application Load Balancer (ALB): Is adept at managing advanced traffic routing for HTTP and HTTPS traffic, making it ideal for modern applications. ALB can route traffic based on the content of the request, such as URL paths or hostnames, which is particularly beneficial for microservices and container-based applications. It's also capable of handling sticky sessions, SSL termination, and WebSocket, all of which can simplify the architecture and improve performance.

2) Network Load Balancer (NLB): Operating at the transport layer, is optimized for low latency and high throughput, capable of handling millions of requests per second. NLB preserves the client's IP, allowing back-end servers to see the source of the traffic, which can be crucial for certain applications that require direct client IP visibility [5].

I. Automated Scaling vs. Manual Scaling

1) Automated scaling: As provided by AWS Auto Scaling, is the method of choice within the architecture. This intelligent service monitors our applications and automatically adjusts capacity to maintain steady, predictable performance at the lowest possible cost. It's a set-it-and-forget-it approach that ensures that the application has the right amount of resources at all times, without manual intervention.

The pros of automated scaling include its responsiveness to actual demand, contributing to a cost-efficient operation by scaling down when demand wanes, and maintaining user experience by scaling up during demand spikes. However, it requires initial configuration and tuning to define the appropriate scaling policies that align with traffic patterns and application needs.

2) Manual Scaling: An alternative to automated scaling is manual scaling, which involves the manual adjustment of resources based on anticipated demand. The advantage of this approach lies in its simplicity and direct control. It's possible to allocate resources based on expected traffic, which could be predictable for some businesses or applications.

The downside is that manual scaling is not reactive to unexpected changes in demand. If traffic spikes suddenly, the manual method may lead to performance degradation or downtime until resources are adjusted, which can impact user experience. Moreover, manual scaling often leads to over-provisioning to handle peak loads, resulting in higher costs [6].

J. Multi-Region Deployment vs. Single Region

1) Multi-Region Deployment: Opting for a multi-region deployment, as seen in the design architecture, involves hosting the application across multiple geographic regions. This strategy significantly enhances availability and disaster recovery capabilities. In the event of a regional outage, traffic can be rerouted to another region, ensuring uninterrupted service. Furthermore, it improves performance for global users, as the content is served from a region closer to them, reducing latency.

However, multi-region deployment brings added complexity in terms of data replication, synchronization, and managing cross-region networking. There's also an increase in cost due to running duplicate resources across multiple regions. Despite these challenges, our design benefits from increased reliability and global user experience, showcasing a commitment to high-quality service delivery.

2) Single Region Deployment: An alternative to this is single region deployment, where resources are hosted in just one geographic region. This approach is simpler to manage and less costly than a multi-region deployment. It is often suitable for applications with a geographically concentrated user base or where the highest level of availability is not critical.

The primary disadvantage is the potential for service disruption if the chosen region experiences an outage. Additionally, users far from the region might experience higher latency [7].

V. TASK ALLOCATION

A. Savindu - Architectural Design and Documentation

- Drafting the initial architectural diagrams.
- Selecting appropriate AWS services to meet the design requirements.
- Ensuring the architecture is scalable, secure, and cost-effective.
- Writing the design rationale, detailing why each AWS service was chosen and how it fits into the overall architecture.
- Keeping the team aligned with AWS best practices and the latest updates from AWS.

B. Thulith - Justification and Cost Analysis

- Justifying the architecture's alignment with business goals.
- Conducting a cost analysis to estimate the budget required for implementing the architecture.
- Researching
- Potential alternatives for each AWS service used and providing a rationale for the selections based on performance and cost-efficiency.
- Report on how the architecture will handle anticipated growth and demand, including a scaling strategy and potential challenges.

C. Nidula - Research and Innovation

- Researching new AWS features and services that could benefit the architecture.
- Proposing enhancements to the architecture, such as introducing machine learning capabilities with Amazon Rekognition.
- Developing a prototype for a critical component of the architecture to demonstrate its viability.
- Creating a repository of resources, research papers, and case studies that support the architecture's approach and can serve as a knowledge base for the team.

VI. CONCLUSION

In conclusion, our AWS-based architecture presents a highly efficient solution, showcasing the optimal balance between cost, performance, scalability, and security. The design we have chosen demonstrates not just theoretical viability but practical robustness in handling real-world demands. By intelligently leveraging AWS services, we've achieved a cost-effective structure without compromising on performance. Scalability is at the heart of our design, ensuring that the system can adapt dynamically to varying workloads with minimal manual intervention. Security, a paramount concern in today's digital landscape, is meticulously integrated into every layer of our architecture, providing a fortified environment against evolving threats. It stands as a testament to the power of AWS in crafting solutions that are not just functional but forward-thinking. Overall, this design is a robust, scalable, secure, and cost-effective choice for any high-availability web server requirements.

Acknowledgment

We would especially want to express our sincere gratitude to Mr. Tharindu Suraj for his professional assistance and the academic personnel at Swinburne University of Technology for their contributions to this project. The commitment of our group members, the technical know-how from the AWS support community, and the resources provided by the institution were priceless. We also value the ongoing support from our friends and family, as well as the helpful criticism from our colleagues.

References

[1] “AWS Pricing Calculator,” Amazon Web Services, Inc. [Online]. Available: https://calculator.aws/#/addService
[2] “The State of Serverless Computing,” 451 Research. [Online]. Available: https://451research.com/serverless-computing
[3]“NoSQL Vs SQL Databases,” MongoDB. [Online]. Available: https://www.mongodb.com/nosql-explained/nosql-vs-sql
[4] "Caching Strategies and How to Choose the Right One," CodeAhoy. [Online]. Available: https://codeahoy.com/2020/03/08/caching-strategies-and-how-to-choose-the-right-one/
[5]G. Author, “Benchmarking 5 Popular Load Balancers: Nginx, HAProxy, Envoy, Traefik, and ALB | Loggly,” *Log Analysis | Log Monitoring by Loggly*, Dec. 10, 2018. [Online]. Available: https://www.loggly.com/blog/benchmarking-5-popular-load-balancers-nginx-haproxy-envoy-traefik-and-alb/
[6]martinekuan, “Autoscaling guidance - Best practices for cloud applications,” *Autoscaling guidance - Best practices for cloud applications | Microsoft Learn*, Dec. 16, 2022. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/best-practices/auto-scaling
[7]“AWS Multi-Region Fundamentals - AWS Multi-Region Fundamentals,” AWS Multi-Region Fundamentals - AWS Multi-Region Fundamentals. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/aws-multi-region-fundamentals/aws-multi-region-fundamentals.html
[8]umer, “Caching Strategies and How to Choose the Right One,” *CodeAhoy*, Aug. 11, 2017. [Online]. Available: https://codeahoy.com/2017/08/11/caching-strategies-and-how-to-choose-the-right-one/
[9]“Load Balancer Types: How to Choose the Right One,” *Deploy Mastery*, Apr. 18, 2023. [Online]. Available: https://www.deploymastery.com/2023/04/18/load-balancer-types-how-to-choose-the-right-one/
[10] "Amazon Simple Storage Service FAQs," Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/s3/faqs/