**Cloud Computing Assignment 3**

Timothy Loh 103595473: Mainly Diagrams and Architecture Design, Some Justifications

Ayesha Shaikh 103165863: Researching and documenting the design rationale, comparisons and justification

Bhuvan Virmani 103847174 : Architecture Designs, Mainly Research and Development

Due Date: 24/10/2022 9:00am

Tutor: Ramya Kalivarapu

Lab: Thursday 6:30pm

Page Limit: Maximum 15 Pages

**Architecture Design**

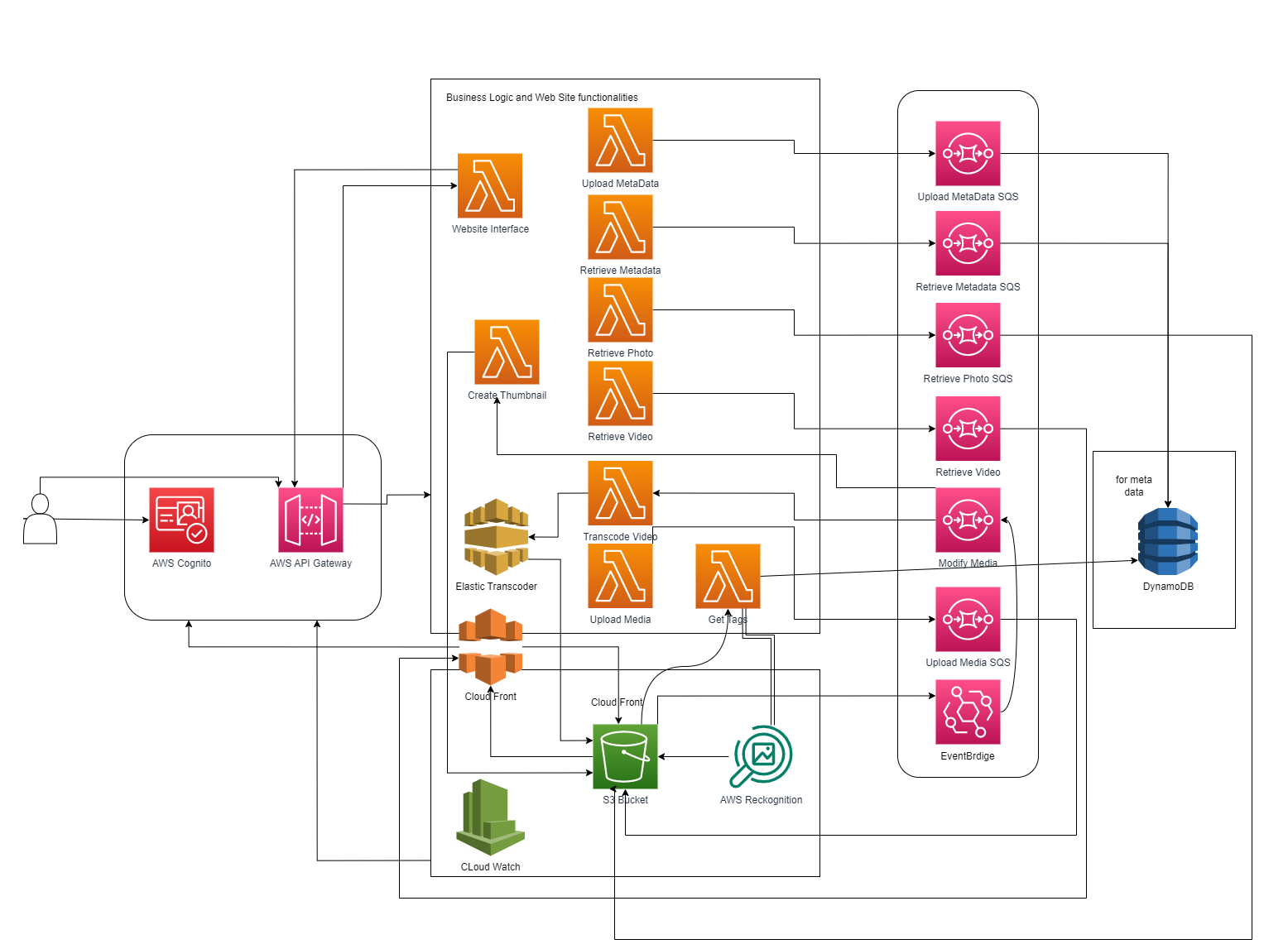


Fig 1. Architecture Diagram

**Descriptions of services used**

**AWS Cognito:**

AWS Cognito is a user identification and authentication web service. It can be used to define roles and map authenticated users with specific roles.

It is used to authenticate users and allowing them to do certain things only after they are authenticated via email-verification or multifactor authentication to ensure higher security and provide trust to its users.[1]

**AWS API Gateway:**

Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST, HTTP, and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services, as well as data stored in the AWS Cloud

It is used to support for stateful (WebSocket) and stateless (HTTP and REST) APIs. [2]

**AWS Lambda:**

A computational service called Lambda enables you to run code without setting up or maintaining servers. Lambda executes your code on a highly available computing infrastructure and manages all of the computing resources.

It uses Process data at scale, Execute code at the capacity you need, as you need it. Scale to match your data volume automatically and enable custom event triggers. Can also be used to create event-driven applications, Build event-driven functions for easy communication between decoupled services. Reduce costs by running applications during times of peak demand without crashing or over-provisioning resources. [3]

**Amazon SQS:**

Amazon Simple Queue Service (SQS) lets you send, store, and receive messages between software components at any volume, without losing messages or requiring other services to be available.

It is used to Decouple microservices and process event-driven applications. Separate frontend from backend systems, such as in a banking application. Customers immediately get a response. It Increases application reliability and scale Amazon SQS provides a simple and reliable way for customers to decouple and connect components (microservices) together using queues.[4]

**Amazon DynamoDB:**

A fully managed NoSQL database service, Amazon DynamoDB offers quick and reliable performance along with seamless scaling. You can build database tables using DynamoDB that can handle any volume of request traffic and store and retrieve any quantity of data. Your tables' throughput capacity can be increased or decreased without causing downtime or performance deterioration.

It is used because it is a highly scalable, serverless and easily administered. It also generates automatic backups and gives high security of data. [5]

**Amazon S3:**

Amazon Simple Storage Service (Amazon S3) is an object storage service offering industry-leading scalability, data availability, security, and performance.

It has strong access controls, adaptable replication tools, and enterprise-wide visibility that make managing data at any scale simple.[6]

**AWS Rekognition:**

Amazon Rekognition is an AI driven tool to analyze media and extract information and insights from them.

It can be used to detect inappropriate content across images and videos. Can be used to identify the information contained in the media.[7]

**Amazon Elastic Transcoder:**

Media transcoding is done in the cloud with Amazon Elastic Transcoder. It is intended to be a highly scalable, simple-to-use, and economical method for businesses and developers to transcode media files from their original formats into forms that can be played back on gadgets like smartphones, tablets, and PCs.

It is used to change the resolution of the media making it flexible to run on different devices. [8]

**AWS Cloudfront:**

AWS CloudFront is a broadly distributed network provided by Amazon Web Services that securely and quickly sends content to clients, including software, SDKs, videos, and other types of content.

One of the purposes of using CloudFront is to reduce the number of requests that your origin server must respond to directly. With CloudFront caching, more objects are served from CloudFront edge locations, which are closer to your users. This reduces the load on your origin server and reduces latency.[9]

**Amazon CloudWatch**

Your Amazon Web Services (AWS) resources and the AWS applications you use are continuously monitored by Amazon CloudWatch.

Every AWS service you use has metrics automatically displayed on the CloudWatch home page. Additionally, you can design dashboards that show metrics related to your own applications as well as arbitrary collections of metrics.[10]

**UML Collaboration Diagram for each Use Case**

**Uploading a Video**

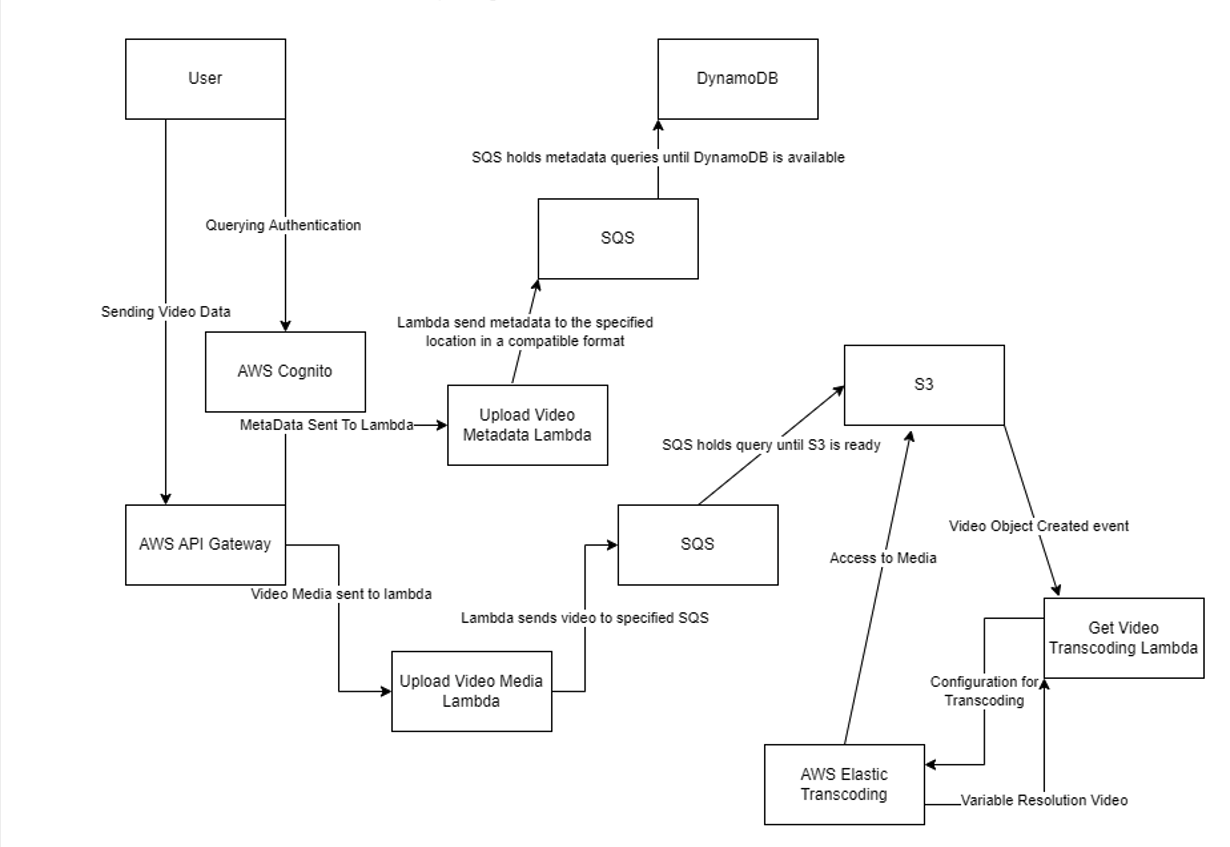
****

Figure 2. Uploading a video to the server. The metadata, any data like Title, Date, Description etc. That metadata will be passed to the DynamoDB to be stored in a NoSQL database. The video wil

**Uploading an Image**

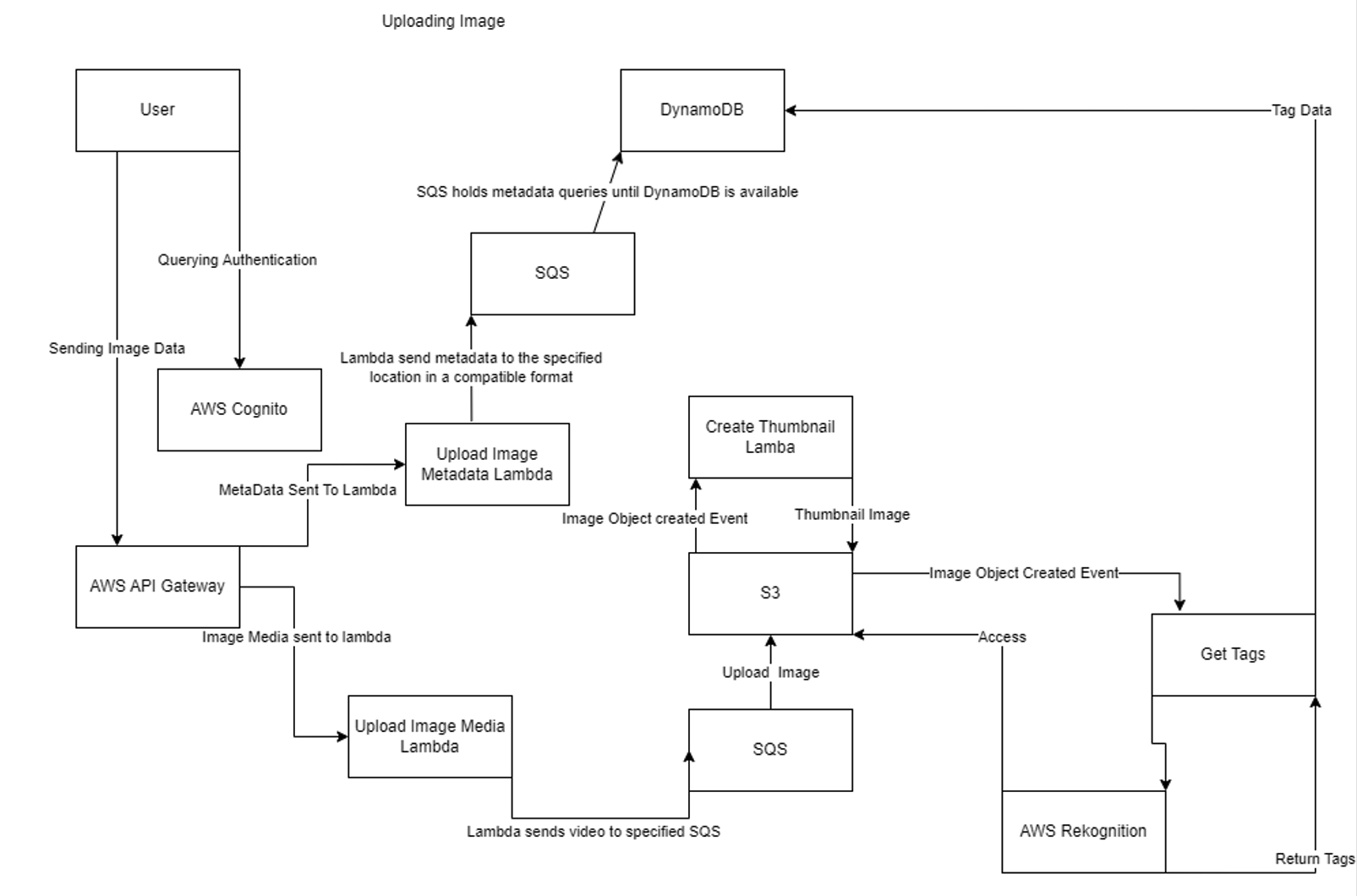
****

Figure 3. Upload an image. The metadata will go to the DynamoDB. The image media will go to the S3 Bucket. This will trigger an event for Rekognition and Thumbnail creation.

**Viewing a Video**

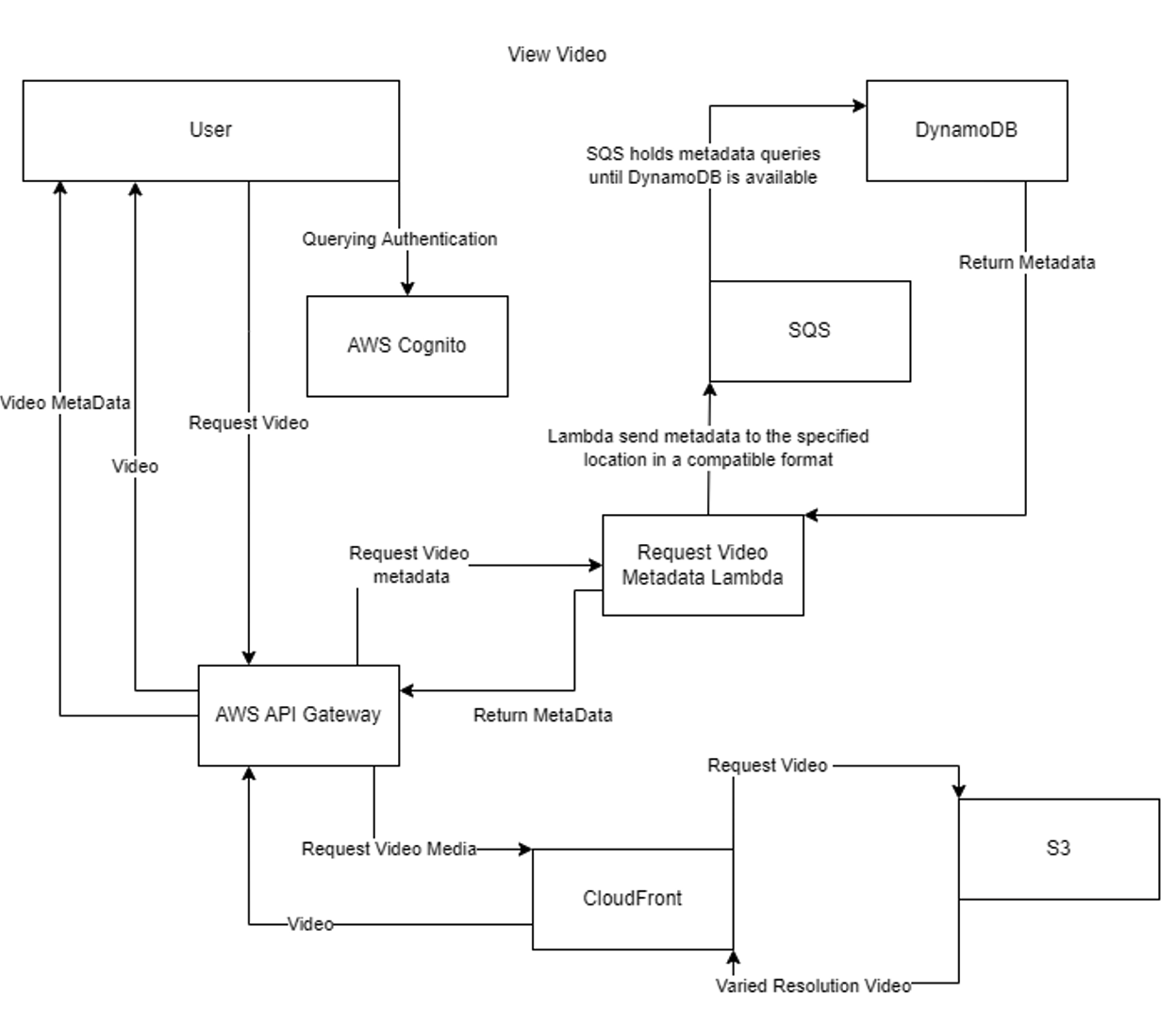
****

Figure 4. Viewing a video. The video will be viewed through cloudfront to deliver dynamic resolution for the user.

**Viewing an Image**

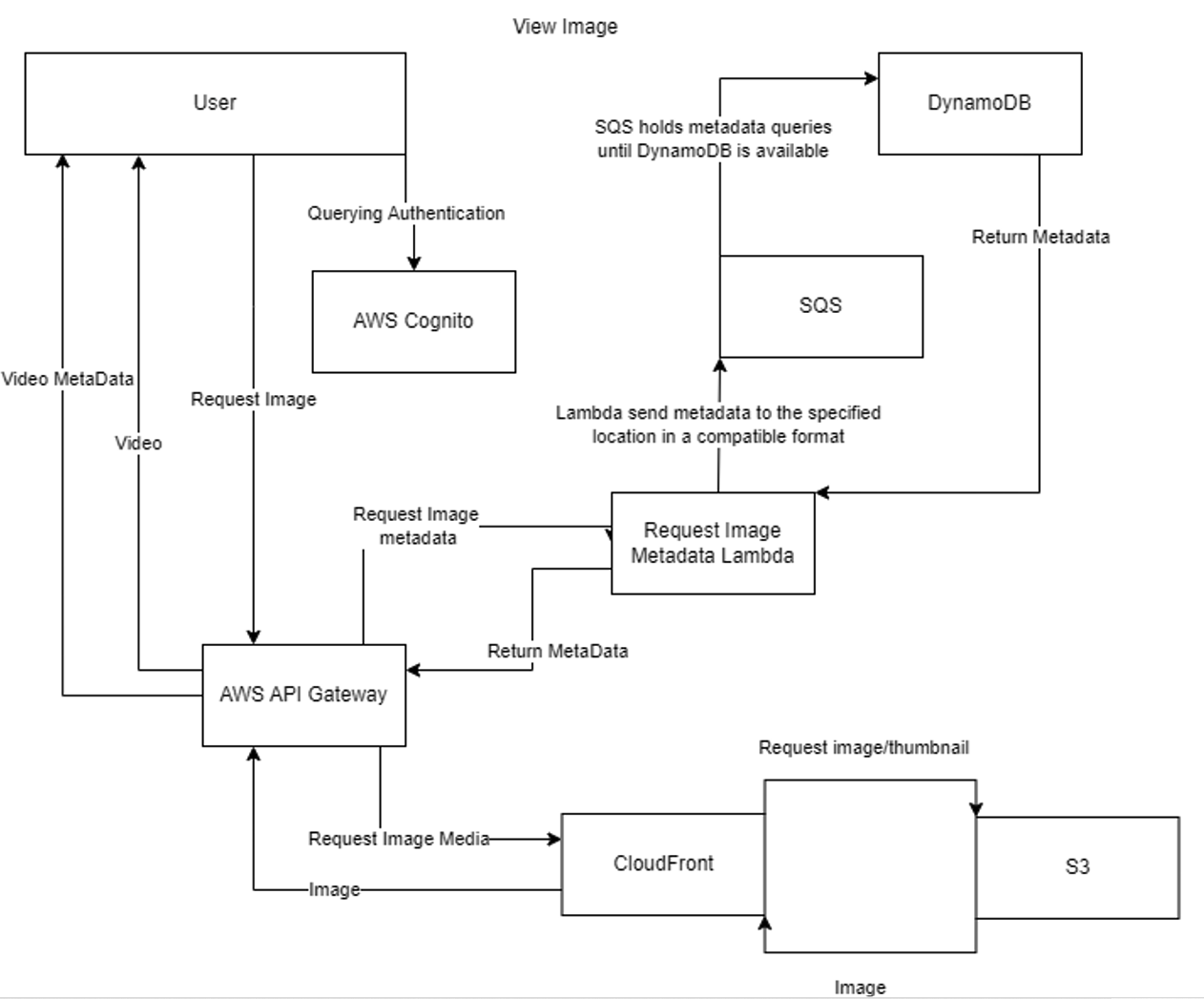
****

Figure 5. View an Image. Image requests go to dynamodb for metadata. Depending on the request, there are thumbnail versions of images for faster processing.

**Design Rationale**

Solution Introduction

To address the requirement specifications and the problem domain as described in the brief our as arrived at the solution of implementing a serverless architecture, following microservices serverless framework using lambda.

The web application would be decomposed into its composite functionality which would be encapsulated in dedicated lambda functions. These would handle each of the web application’s core tasks, called upon according to the requests/traffic received by the web application via the API Gateway logging(which distributes work flows). The application comprises the compute platform that has code which executes upon those network requests, receiving incoming network requests, managed data store, object storage, messaging services for queues, sub/pub streams and so on. It is cost-effective as it optimizes the cloud principle of pay per usage billing and minimizing to zero the management overhead.

Migrating to this architecture requires restructuring the entire website application to follow the Serverless Application Model (SAM) template. (Cloud Formation resources may be used to assist in this initially). This would define the application functionalities in a lambda function with its endpoint. This endpoint is highly scalable and highly available. Additionally, using the default [Service Quotas](https://docs.aws.amazon.com/servicequotas/latest/userguide/intro.html), this Endpoint can handle millions of requests a day. Lambda manages the scaling automatically, and also scales down as needed without any intervention from the developer.

There is no need to define VPCs, subnets or security groups, or install and manage a web server stack. A considerable part of the underlying infrastructure is managed for you, letting you focus primarily on the business logic of the application.

Reasons behind this solution path

Having no servers removes operational complexity of running and monitoring underlying infrastructure. This solution is completely built of cloud managed services. It is a simplified and more automated approach to achieve scalability and high availability. This microservices approach would communicate with each other, the client and the backend using API calls. It keeps the application lightweight and decoupled. It replaces manually configured infrastructure and code libraries with a model that builds, deploys and runs applications automatically and scales to respond to the level of traffic.

Security

AWS Cognito is used to manage users and provide authentication to secure the backend API**.** Alternatively, authorization and authentication can be implemented in serverless development by using open standard [JSON Web Tokens](https://jwt.io/) (JWTs). API Gateway then authenticates the user at the service level using [Amazon Cognito](https://aws.amazon.com/cognito/), a [Lambda authorizer](https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-use-lambda-authorizer.html), or with a [JWT authorizer with HTTP APIs](https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-jwt-authorizer.html). You use an identity provider such as [Amazon Cognito](https://aws.amazon.com/cognito/) or [Auth0](https://auth0.com/) to generate the user token. You pass the token in the API request in the [Authorization header](https://docs.aws.amazon.com/apigateway/api-reference/signing-requests/). The API Gateway service then validates the token before the request is sent downstream to your application. This separates out the security functionality following our serverless microservices approach. Therefore, in the future extensions or added support for social logins, [multi-factor authentication](https://en.wikipedia.org/wiki/Multi-factor_authentication) (MFA) and [OAuth](https://en.wikipedia.org/wiki/OAuth) can be incorporated without changing your code.

For security across regions, as the application grows to more functions or across Regions, you are not relying on a single authentication point in your architecture. Each [microservice](https://en.wikipedia.org/wiki/Microservices) validates a JWT independently and can verify the authorization claims that can be securely embedded in the token’s payload.

Feature functionalities

Media Handling

The web application would be handling large binary files such as user uploads. Processing these on web servers can be compute and network-intensive. Therefore, file uploads are done on our serverless model by using Amazon S3 directly. In this process, you request a [pre-signed URL](https://docs.aws.amazon.com/AmazonS3/latest/dev/PresignedUrlUploadObject.html) from the S3 service and upload the binary data directly to this endpoint. This reduces load on your infrastructure and increases scalability.

For processing binaries, you can use the S3 [PutObject](https://docs.aws.amazon.com/AmazonS3/latest/API/API_PutObject.html) event to trigger serverless workflows. For [processing images](https://aws.amazon.com/solutions/implementations/serverless-image-handler/), [translating documents](https://aws.amazon.com/blogs/compute/translating-documents-at-enterprise-scale-with-serverless/), or [transcribing audio](https://aws.amazon.com/blogs/compute/converting-call-center-recordings-into-useful-data-for-analytics/). For [complex business workflows](https://aws.amazon.com/blogs/compute/automating-scalable-business-workflows-using-minimal-code/), the event can trigger [AWS Step Functions](https://aws.amazon.com/step-functions/) workflows. This is a highly scalable way to bring automation and custom processing to binary uploads in the web application.

The amazon API gateway endpoint is called, which invokes the *getSignedURL* Lambda function. This gets a [signed URL](https://docs.aws.amazon.com/AmazonS3/latest/dev/ShareObjectPreSignedURL.html) from the S3 bucket in order to directly upload the file from the application to the S3 bucket.

The browser downloads the application’s files from [Amazon S3](https://aws.amazon.com/s3/) via [Amazon CloudFront](https://aws.amazon.com/cloudfront/) when the user visits the web application’s url. To get data, the application calls API Gateway endpoints to fetch and store dynamic data. The HTML page building and rendering is managed entirely by the client browser, improving responsiveness and reducing network traffic with the application backend.

Triggers are set using the Eventbridge event handlers, which triggers bucket actions via lambda function calls.(upload media file, get media file, resize media file, do processing, store metadata in dynamoDB)

[S3 Object Lambda](https://aws.amazon.com/s3/features/object-lambda) is not used as an acting proxy layer which allows data processing to happen on the retrieved datased, as the web application only deals with objects rather than relational database entries. This considerably reduces the complexity and cost of the solution.

Flexibility and scalability benefits

Based on cluster demand the application hosting infrastructure can scale up and down. When dealing with virtual machines it is difficult to indicate how many requests or traffic the cluster can actually handle, serverless services have a very specific limit attached to the service. This metric allows us to have a definite limit to eliminate the possibility of the application collapsing under heavy traffic situations. The application under the model would return a clear distinct response under these conditions as part of the amazon SDK used to interact with them. This allows us to cleanly handle the service limits for the clients. It also means that the services are scaled down to zero when no load or traffic is incoming, you are not paying for a machine sitting idle, like you would when servers are created.

When our web application would be subject to high traffic Lambda functions hosting the web application would work as monoliths being responsible for and performing a single task.

Flexibility is there for the choice of stack and development suite, and modifications to the code base are then well-handled and easy to deploy into production.

The Lambda service scales up if S3 sends multiple events simultaneously. if any active instances of the function are already processing events, the Lambda service scales up. It’s this rapid scaling and parallelization in both S3 and Lambda that make this pattern such a powerful core architecture for many applications.

The new regional API endpoint feature in [Amazon API Gateway](https://aws.amazon.com/api-gateway) to make latency-based routing and health checks a seamless process for the API client making the requests. The default API endpoint type in API Gateway is the edge-optimized API endpoint, which enables clients to access an API through an Amazon CloudFront distribution. This typically improves connection time for geographically diverse clients. By default, a custom domain name is globally unique and the edge-optimized API endpoint would invoke a Lambda function in a single region in the case of Lambda integration. The new regional API endpoint in API Gateway moves the API endpoint into the region and the custom domain name is unique per region. This makes it possible to run a full copy of an API in each region.

Storage resources

S3 is highly durable and highly available. For functions that need a durable store of user data that can be rehydrated between invocations, [Amazon DynamoDB](https://aws.amazon.com/dynamodb/) tables provide a low-latency, cost-effective solution. It also stores metadata from the objects stored in S3. S3 object permissions and policies are described in the S3 bucket according to organization and web application constraints.

DynamoDB is designed to provide consistent performance at any scale. Object attributes (metadata and tags) stored in the S3 bucket are cataloged in the DynamoDB NoSQL database as item collections.

Tagging media files using AI

It would use an intelligent prediction tool or a pattern identification tool to perform image processing and identification to assign appropriate tags. This approach also reduces the steps to the processing pipeline for these heavy machine learning functions.

To add powerful visual analysis on the large media store our application will be managing, Amazon Rekognition is used. It is capable of searching, using neural network models for image labeling, verifying, and organizing millions of images using its own SDK. It offers its own API for all its functionalities. It takes in objects either directly from S3 in png, or jpeg format or even as a byte array, it can also work on videos stored in S3 in the H.264 encoding.

Backend Queueing

S3 raises events when objects are put, copied, or deleted in a bucket. It also raises a broad number of other notifications, such as when lifecycle events occur. You can configure S3 to invoke Lambda from these events. S3 passes [details of the event](https://docs.aws.amazon.com/AmazonS3/latest/dev/notification-content-structure.html), not the object itself, to the Lambda function in a JSON object. This object contains an array of records, so it’s possible to receive more than one S3 event per invocation.

The lambda handler code can take advantage of [concurrent asynchronous executions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/all) available in different language constructs. This means that multiple objects are processed in parallel to minimize the overall function execution time. Instead of handling and logging any errors within the function’s code, it’s also possible to use destinations for asynchronous invocations. Amazon’s Simple Queueing Service and CloudWatch resources may be utilized here. This would provide greater control for managing events that fail processing.

According to the reserved concurrency, the events from S3 are queued internally until a Lambda instance is available for processing.

When it is required to invoke multiple Lambda functions per S3 event. S3 can publish notifications to SNS, where events are delivered to a range of targets, like Lambda functions, web hooks, SQS queues, HTTP endpoints, email, text messages and push notifications. SNS has the capability to distribute and spread out the tasks, enabling one event to be delivered to multiple destinations.

In busy applications, the volume of S3 events may be too large for a downstream system, such as a non-serverless service. This is where an SQS queue is used as a notification target or a producer. After events are published to a queue, they can be consumed by Lambda functions and other services. The queue acts as a buffer and can help streamline traffic for systems consuming these events.

In [decoupling larger applications with Amazon EventBridge](https://aws.amazon.com/blogs/compute/decoupling-larger-applications-with-amazon-eventbridge/), services can be decoupled within an application using an event bus. This pattern helps separate the producers and consumers of events in your workload. This can make each service become more independent and more resilient to changes with the overall application. [Amazon EventBridge](https://aws.amazon.com/eventbridge/) acts as the event router as it defines the EventBridge rule to filter for events, and publishes data back to the event bus after processing is complete.

Costs

This low cost environment saves 70% in server costs by automatically scaling based on demand and compute needs.

Serverless services only bill for when they are actually used.

*Infrastructure costs* - Since we are using Lambdas as the serverless compute services instead of provisioning and managing virtual machines or containers. It has minimal administration and a pay for what you use model. Charges are based on the number of requests for your functions and the duration it takes for your code to execute. Lambda counts a request each time it starts executing in response to an event notification trigger, such as from Amazon Simple Notification Service (SNS) or Amazon EventBridge, or an invoke call, such as from Amazon API Gateway, or via the AWS SDK, including test invokes from the AWS Console. Duration is calculated from the time your code begins executing until it returns or otherwise terminates, rounded up to the nearest 1 ms\*. The price depends on the amount of memory you allocate to your function. In the AWS Lambda resource model, you choose the amount of memory you want for your function, and are allocated proportional CPU power and other resources.

The AWS Lambda free tier includes one million free requests per month and 400,000 GB-seconds of compute time per month, usable for functions powered by both x86, and Graviton2 processors, in aggregate. Lambda also offers tiered pricing options for on-demand duration above certain monthly usage thresholds. AWS Lambda participates in Compute Savings Plans, a flexible pricing model that offers low prices on Lambda usage, in exchange for a commitment to a consistent amount of usage (measured in $/hour) for a one- or three-year term. With Compute Savings Plans, you can save up to 17% on AWS Lambda. Savings apply to duration and Provisioned Concurrency.

For AWS lambda - $0.0000002 per request

For AWS Cognito - $0.0055 per monthly active user

For AWS SQS - $0.0000004 per request

For AWS DynamoDB - $0.25 per GB per month

For AWS S3 - $0.0221 per GB per month

For AWS API gateway - $0.90 per million requests

For AWS Elastic Transcoder - $0.03 per minute

For AWS Cloudfront - $0.085 per GB

For AWS CloudWatch logs - $0.50 per GB

Estimated Cost Criteria

* 5,000 active users per month
* 4 videos + 4 images making 2gb of data per user per month
* 10,000 gb per month for S3
* 10 MBs of Metadata in DynamoDB
* 50 MBs of logs in CloudWatch
* Guessing we get 1,000 minutes of video per month (which will take 500 GB)
* Getting 150,000 API requests per month

For AWS lambda - $0.001 \* 9 = $0.009

For AWS Cognito - $27.5

For AWS SQS - $0.06 \* 6 = $0.36

For DynamoDB - $0.0025

For S3 - $221

For API Gateways - $0.135

For Elastic Transcoder - $30

For Cloudfront - $42.5

For CloudWatch - $0.025

Estimated Budget Would be:- $321.5315

Operationalization

This solution increases team efficiency by reducing development tasks and interventions, and having more focused autonomous teams which perform monitored and management tasks.

For this large-scale serverless application, it is practical to split applications into multiple services and repositories for separate teams. Often, individual services must integrate with existing S3 buckets, not create these in the application templates. You may also have to integrate a single service with multiple S3 buckets.

CloudWatch is used for its [custom metric](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/publishingMetrics.html) capability and new [embedded log format](https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/CloudWatch_Embedded_Metric_Format_Generation.html) feature.Cloud watch rules can trigger certain code blocks in lambda functions every minute and performs multiple Remote Function Calls through the Lambda layer. It allows us to live monitor the serverless compute capacity and performance metrics.

Tiered Architecture

The solution follows the 3-tier architecture. The first tier is client-facing and includes the security layer, the API management layer and the content distribution layer features. It has the amazon resources of cognito, API gateway and cloud front. The middle tier comprises the Business logic layer, which has the AWS step functions, aws lambda functions, lambda indexer, cloudwatch and aws rekognition. The final tier in the backend consists of the data layer which has the S3 and DynamoDB; and the integration layer having the Amazon SNS, SQS and EventBridge.

This three tiered architecture creates a workload-aware cluster scaling logic whilst reliably maintaining event integrations, and managing runtimes computing with ease. It follows the popular serverless backend designs of aws architecture. It groups together the common responsibilities and resources by logically splitting them in different layers and then tiers. This infrastructure decision optimizes costs and efficiency of the overall cloud solution.

**REFERENCES:**

[1] Cloud Infrastructure Services and Enterprise grade Network & IT Solutions Provider. (2020). AWS Cognito - Features, Architecture and Use Cases by SNDK Corp. [online] Available at: https://www.sndkcorp.com/aws-cognito-features-architecture-use-cases/#:~:text=AWS%20Cognito%20removes%20the%20load%20of%20creating%20and [Accessed 23 Oct. 2022].

[2] docs.aws.amazon.com. (n.d.). *What is Amazon API Gateway? - Amazon API Gateway*. [online] Available at: https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html#api-gateway-overview-features [Accessed 23 Oct. 2022].

‌

[3] AWS (2019). *AWS Lambda – Serverless Compute - Amazon Web Services*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/lambda/>.

[4]Amazon Web Services, Inc. (2018). *Amazon Simple Queue Service (SQS) | Message Queuing for Messaging Applications | AWS*. [online] Available at: <https://aws.amazon.com/sqs/>.

[5]docs.aws.amazon.com. (n.d.). *What Is Amazon DynamoDB? - Amazon DynamoDB*. [online] Available at: <https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html>.

[6] Amazon Web Services, Inc. (n.d.). *Amazon S3 Security Features - Amazon Web Services*. [online] Available at: https://aws.amazon.com/s3/security/?pg=ln&sec=be#Access\_management\_and\_security.

‌[7] Amazon (2017). *Amazon Rekognition – Video and Image - AWS*. [online] Amazon Web Services, Inc. Available at: https://aws.amazon.com/rekognition/.

‌

[8] Amazon Web Services, Inc. (2019). *AWS | Amazon Elastic Transcoder - Media & Video Transcoding in the Cloud*. [online] Available at: <https://aws.amazon.com/elastictranscoder/>.

‌[9] Simplilearn.com. (n.d.). *AWS CloudFront: Everything You Need to Know*. [online] Available at: <https://www.simplilearn.com/tutorials/aws-tutorial/aws-cloudfront>.

[10] Amazon.com. (2019). *What is Amazon CloudWatch? - Amazon CloudWatch*. [online] Available at: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/WhatIsCloudWatch.html.

‌

‌

‌