Assignment 3

Vihanga Bandara
COS20019
Thursday_8:30
Melbourne VIC
vihanga7b@gmail.com

Randew Kumarasinghe
COS20019
Thursday_8:30
Melbourne VIC
randewkumarasinghe@gmail.com

Introduction

It is important to evaluate and revise design alternatives for any solution so that we can gauge whether costs can be optimized or whether efficiency can be improved upon. Serverless and event-driven architectures for applications is a much more modern idea that removes the aspect of server maintenance and increases the focus on business/product logic. Adapting a serverless design that automates scalability while reducing costs is essential to a better product towards a client.

I. Architecture

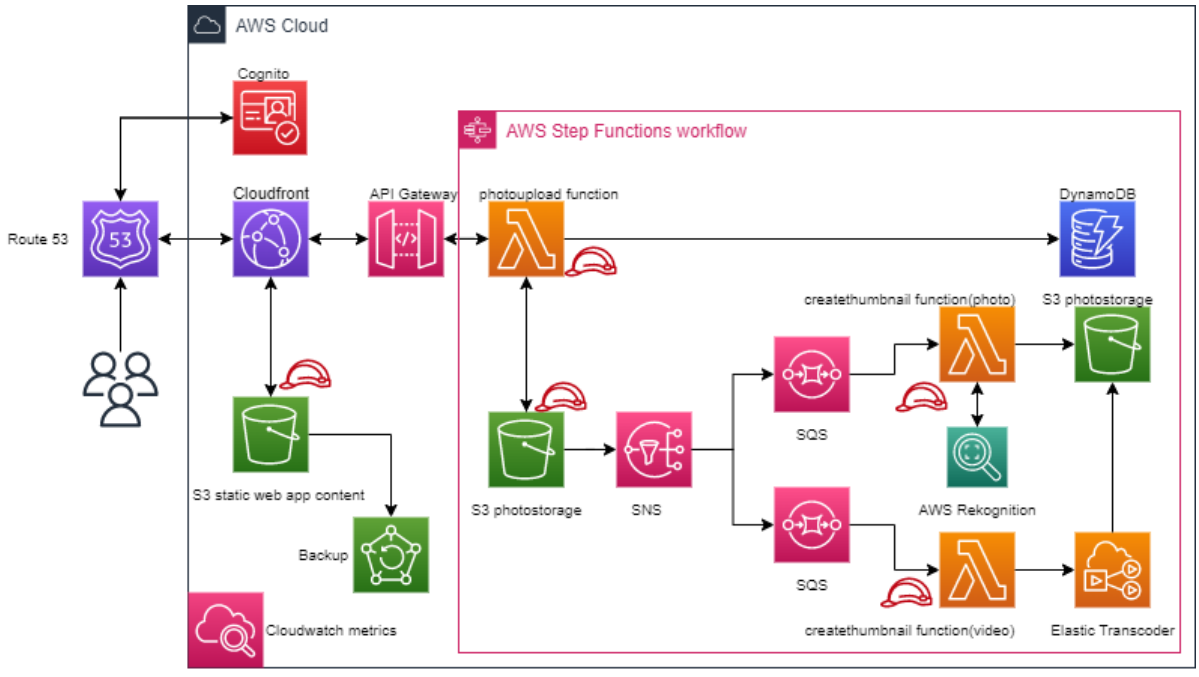

Figure 1: Architecture diagram

The user accesses the web application through the Route 53 service which is a DNS web service that connects user requests to applications that run on AWS. Amazon Cognito provides security with identity verification and features to protect the users as well as the developers. Users who access the web application can sign into the site to access the correlating resources. The correct policies can also be enforced this way.

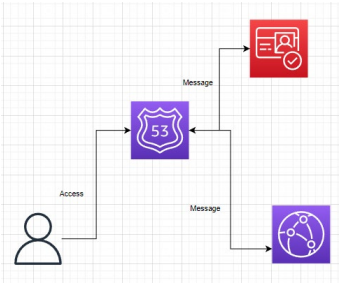

Figure 2: UML Collab Route 53_Cognito_Cloudfront

The static application content that was previously held in the EC2 instances have been moved to an S3 bucket. Cloudfront acts as a CDN front to deliver the application content that is stored in S3 to the users. CloudFront allows global use cases and scales accordingly depending on the resources required, which is extremely efficient especially when considering that CloudFront also caches the content, meaning that response times for viewers of the website will be fast regardless of where they are.

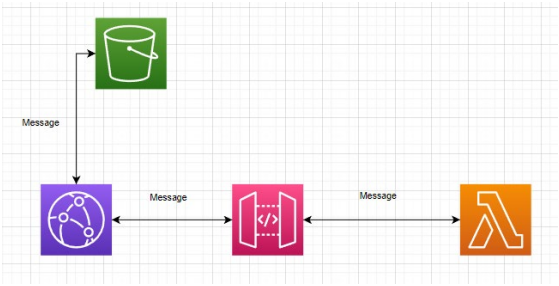

Figure 3:UML Cloudfront_API_Lambda_S3

Amazon API Gateway is setup to allow for the web application to interact with the backend services. CloudFront and API Gateway together allow for higher performance and a decreased likelihood that the backend functions are not used unnecessarily. It also allows for monitoring important metrics of the services used through CloudWatch. As the architecture is serverless, it operates through Lambda function compute services. When a user uploads a new photo, it will invoke a lambda function through the API Gateway that stores the photo in S3 and creates the meta data records in DynamoDB.

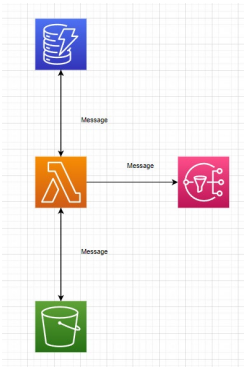

Figure 4:UML_Lambda_Dynamo_S3_SNS

This process is within an AWS Step Function workflow. Step function makes it so that the code need not be maintained overtime and workflows will automate and scale on demand. Since we use multiple Lambda functions, they will be combined into a responsive serverless application through the step function. When the photo is uploaded to the S3 bucket, an SNS topic is invoked to send a message to two SQS components. SQS is used so that requests to upload a file will be queued and managed accordingly without any data being lost. The use of SNS and SQS also mean that the architecture is more decoupled, and services are connected through the queues. Messages through SQS are also encrypted, which protects user data.

SQS sends request messages to two Lambda functions that convert the uploaded media into viewable format. One of two functions will be used depending on whether it is an image or a video upload. In the case of a video upload, the “createthumbnail video” lambda function is invoked, and Amazon Elastic Transcoder is used to transcode the video media to playable format before uploading the transcoded version to the S3 bucket. In the case of an image upload, the

"createthumbnail photo" function is invoked, and AWS Rekognition uses AI based management to identify and analyse tags in the uploaded image before returning it to the function, which then uploads the resized thumbnail version to the S3 bucket. Rekognition allows the developer to extract information about images and use that information to adjust the site or even detect unwanted content.

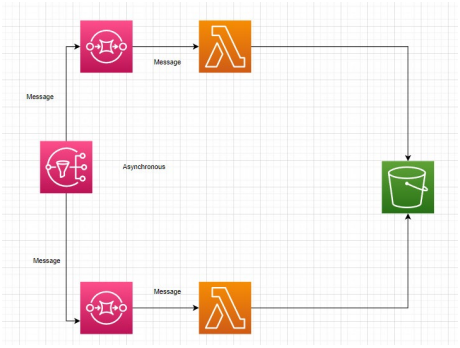

Figure 5: UML_SQS_SNS_S3

A non-relational database, DynamoDB, is used to store the meta data records of the uploaded photos and videos. All the content in S3 is backed up for disaster protection using AWS Backup, which allows for central management and automated data protection against accidental or malicious incidents of data loss. CloudWatch collects and monitors metrics and event data in real time, which allows for a deeper understanding of any performance issues in the application and more efficient perspectives to resolve those issues. IAM roles are implemented to allow for lambda functions and S3, as well as the database to interact and access each other's resources to fulfill the needs of the web application without providing unnecessary permissions to other services.

Decoupling is achieved in this architecture in a multitude of ways. Being serverless in and out of itself adds an element of low coupling to the architecture. If we need to upgrade or a scale one part of the architecture, we do not need to upscale the entire server that holds the application, rather we can simply isolate the area that needs work and add on to it without harming the other processes. Having services trigger and communicate between each other through events means that they need not be tightly coupled to function and have a proper workflow. By using services such as SQS, that are asynchronous services, we decouple the architecture even further. Rather than waiting for the functions invoked by a message to process the vent and return a response, SQS queues the event and returns a response immediately. The status of services will not affect the response from the destination. Having two nodes instead of one for the processing of images and video also mean that in the case where one goes down, needs maintenance, or simply takes too long, the other is not affected and are effectively "loosely coupled" in the architecture.

Requirements of the business scenario are met as per the client's requests. Cloud services are used extensively to remove in house server maintenance. A serverless and event driven architecture allows for automatic scaling and load balancing compared to the use of ECS. Better global response times adapted along with various versions of media upload along with viewable versions of them available on the site for the users to access. Decoupled architecture design is implemented, and all services are provided in the AWS environment to reduce cost and performance constraints.

II. Design Rationale

Serverless architecture means that scaling and load balancing is automatic. Regardless of how much the demand for application grows, the serverless architecture and its functions will grow to meet the demand without us having to manually upgrade the resources. Rather than using in-house management, VMs or containers, all of which must be maintained and manually worked on, being serverless allows for an increase in business logic focus and a decrease in stack implementation. It allows for more areas of the business scenario to be fulfilled in a cost efficient and reliable manner.

Since it is an event driven solution, there is no cost when the application is not being used. All costs are applied when events occur and for the duration they occur for. Which is in the 100ms range, this is far more cost effective compared to a server-based solution, which will rack up costs for simply maintaining the architecture idly.

Non-relational database adapted to reduce costs and increase speed. DynamoDB is also native to the AWS environment, removing costs for third party database hosting. In the business scenario, the photouploader application does not require a large, structured data store. Rather than using a relational database that will create unnecessary costs as well as performance issues, it is far better from an analytical standpoint to adapt a non-relational database. This also allows for a far more flexible scenario as we could even implement functions in future iterations that use the DynamoDB data.

CloudFront makes sure that the response time in all regions is equally fast and it will not matter where you are accessing the site from. All services provided in the AWS ecosystem. Architecture decoupled so that video transcoding does not slow down the thumbnail reformatting of image uploads. Asynchronous messaging is adopted, considering that the scaling of the application is unpredictable, and we require it to be highly available. SQS is used in the design to integrate independency into the workflow and introduce flexibility. Synchronous messaging on the other hand is far less performance effective, and that type of solution scales very poorly when met with high demand. SNS allows for services to be subscribed to it and receive messages immediately. This design also protects information. In the case of the application going down or a crash occurring, data that was put in the SQS queue will remain there until it is acknowledged.

IAM roles are implemented in the design for security purposes along with Cognito. Lambda functions are assigned IAM roles that are independent to other parts of the architecture. Only the necessary permissions are provided so that they can achieve a very specific set of results, which allows them to be more independent and lets them evolve overtime. Amazon Cognito provides authorization and user management for the web application. User pools and identity pools are used to let users sign into the application and identity pools allow for access to be granted to various Amazon services. In this way, it is possible to control which users get access to what. In terms of data security, the information that SQS puts in a queue are encrypted upon interception, which protects the data of the user.

CloudFront was the ideal solution in terms of caching compared to other services in the AWS environment such as ElasticCache. ElastiCache is more expensive to maintain and does not comply with the requirements of the client as much as CloudFront. Having data be retrieved from caches close to the traffic source allows for better global response times compared to reading data from managed data-stores.

Cost comes up to 15190.51 USD per month, comparatively, a serverless architecture excluding the labour costs for server maintenance, simply to keep it up and running would cost 15000USD on average apart from the other services used. A few assumptions were made during the calculation. Assumed that all regions are provided, and 500000 ACTIVE users accessing the application per month. Cost also assumed as the most expensive amount that it could be. Variable costs will be lower.

Cost breakdown

All services except for DynamoDB have no upfront cost. Lambda costs 69.23 USD per month with 5 million requests and a 100ms duration for each request. 9GB of memory allocated. StepFunctions costs 124.90 USD for half a million requests and 10 workflow transitions. API Gateway cots 6.75 USD with 5 million API requests with an average size of 512 KB per request. CloudFront is one of the more expensive services, costing 5051.20 USD per month while transferring 50000 GB of data out to the internet and out to origin. Amazon Cognito costs 2864.25 USD with the assumption that half a million new users are active every month. DynamoDB costs 1276.14 USD for a 5000GB date storage size. Elastic Transcoder costs 750 USD with 10000 video assets to transcode every month and the average length of the videos are assumed to be five minutes, which is quite lengthy for a video clip however in this cost analysis we will assume the most expensive scenario to get a solid perspective on the maximum costs. Rekognition costs 1400 USD with half a million images processed every month. Route 53 costs 41.50 USD with all regions provided and 50 million DNS queries a month. SNS costs 1000.24 USD in the scenario where half a million notifications are sent every month. S3 costs 1602.70 USD when 50000 GB is allocated to storage. SQS costs 1003.60 where there are 5 million standard queue requests per month and 50000 GB of outbound data transfer.

Summary

Design solution implemented as per the client’s business requests and cost/performance efficiency. Architecture decoupled and made serverless as well as event driven to provide more effort towards business logic compared to maintenance and stack implementation, the type of infrastructure that comes with server architectures.

Task Allocation

All tasks were distributed among group members equally. Weighting towards architecture design and implementation taken on by Vihanga. Cost related data and components of design rationale taken on by Randew.