Cloud Computing Architecture

Assignment 3

Serverless/Event-driven Architectural Design Report

(This report is written to achieve the degree of the Cloud Computing unit in Swinburne University of Technology)

Xuan Tuan Minh Nguyen – 103819212

Tuesday 6:30pm - 8:30pm tutorial class

Trong Dat Hoang - 104194774

Tuesday 6:30pm - 8:30pm tutorial class

Phuong Doanh Ha – 104480318

Tuesday 6:30pm –8:30pm tutorial class

***Abstract* —** *As we all know, people are increasingly turning to Cloud Computing to address online architectural development challenges because of its easy features and applications. Cloud computing characteristics include easy innovation possibilities based on database storage, computation... and worldwide deployment capabilities. As a result, utilizing AWS, a secure cloud provider, and based on the scenario of a business, we handle "testing," "replacement," and "development" of the website for them, providing it future expansion and diversity. This report will give a justification to show our solution. At the same time, the report will clearly show our use of the services DynamoDB, Route 53, CloudFront, S3 bucket, Amplify, API Gateway, Simple Notification Service, Simple Query Service, Elastic Transcoder, Lambda and CloudWatch. Finally, mention the design that has been replaced in line with the intended architecture.*

***Keywords* —** *Cloud, architecture, AWS, serverless, event-driven, computing, business*

I. Introduction

Virtual technologies, such as cloud computing, are widely employed in the contemporary 4.0 age. This is an online service paradigm that lets customers access computer resources (such as servers, storage, applications, services, and so on) from a variety of locations and according to their needs. Since then, cloud computing has greatly aided businesses in enhancing competition through strong consumer relationships. Cost savings (no infrastructure investment expenditures), flexibility, high security (ensuring private corporate information), performance with integration capabilities, and the ability to serve customers better with the capacity to analyze data and deliver solutions based on each requirement are all positives.

Our team thoroughly studied and assessed the business's requirements before formulating and choosing the essential services. The decision was made to adopt AWS services, including DynamoDB, Route 53, CloudFront, S3 bucket, Amplify, API Gateway, Simple Notification Service, Simple Query Service, Elastic Transcoder, Lambda and CloudWatch. In order to improve dependability, availability, flexibility, and scalability. At the same time, remote login simplifies management while increasing performance and efficiency.

II. Business Scenario

Through assignment 1 and 2, the Photo Album application has been developed smoothly and achieved the desired success. However, to be more advanced, the company requires more than a few issues to widely meet other increased needs.

1. Infrastructure:

- To reduce the requirement for internal system administration, use the AWS S3 cloud service for management.

2. Fulfill growth criteria:

- Architectural design requirements must fulfill growth requirements (doubling every 6 months).
- Implemented solutions: DynamoDB, Route 53, CloudFront, S3 bucket, Amplify, API Gateway, Simple Notification Service, Simple Query Service, Elastic Transcoder, Lambda and CloudWatch

3. Optimize computer resources:

- Consider computational performance as an alternative to t2.micro
- Optimize source code, enhance performance to reduce computing capacity from 80% to 50-60%.

4. Serverless/ event-driven:

- Create and distribute source code for cloud computing services that replace direct server administration. (Lambda)
- Emphasize flexibility and the development of concepts so that actions can occur independently based on circumstances (SNS).

5. Cut expenses:

- Using basic table layout, making the relational database run more effectively.

6. Global Response Time Improvement:

- Aids in speedier response times for the outside of Australia.

7. Expansion of Video Prepare:

- Adding server resources and employing specialized ways for working with video data.

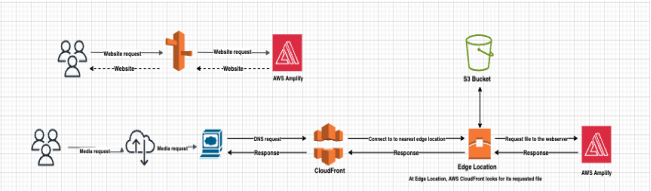
8. Build a multimedia architecture:

- That can automatically build versions and is extensible to avoid system overload during media processing.

III. Featured Architecture

A. Content Delivery Network System

In order to adapt with the growing popularity of the website, an efficient content delivery network (CDN) system is essential to provide low-latency data transmitting and receiving thus keeping the user experience in the highest standard regards of different user geographic locations. This can be done by using



AWS Route 53, which is a Domain Name System (DNS) host service that is provided by AWS and CloudFront (EdgeLocations extension) service to provide low-latency CDN.

Figure 1: Content Delivery Network System

The user can access the website via the registered domain name with Route 53, which will redirect the user to the website that is hosted on a server-based service that is used for hosting website applications called AWS Amplify. Overall, AWS Route 53 is a great service to register a domain, as it has numerous benefits with an affordable budget compared to the other competitors.

Moreover, to maximize the user experience when it comes to stream images and videos, we have used AWS CloudFront to provide the images and videos to the user from the nearest physical locations by taking the advantage of EdgeLocations feature. This feature works by fetching the media in the object-store services, such as S3, etc. in the user's nearest location. If there are no medias on the storage then the EdgeLocations will proceed to retrieve the medias from the web server. This solution ensures that the content is delivered in a fast and efficient way to the users, thus increasing the overall experience on the website.

B. Three-tier Serverless

Multi-tier architecture is a kind of architecture that is widely used when it comes to design an AWS program, there are multiple kinds of multi-tier architectures, but the one that we have chosen to stick with is the three-tier architecture. The three-tier architecture consists of three distinct layers: the presentation tier layer, logic tier and data tier.

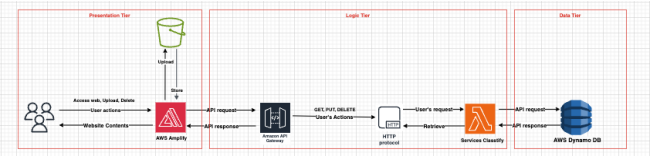


Figure 2: Three-tier Architecture for Serverless

Based on the given architecture, the presentation layer, which consists of front-end web contents, are stored on the AWS S3 and hosted using Amplify, which is a service that is optimized for hosting web applications. For the logic layer, it contains an API Gateway and Lambda function to determine the best way to provide dynamic contents for the website. Finally, the data layer will utilize the use of a NoSQL persistent storage solution that is introduced by AWS called DynamoDB.

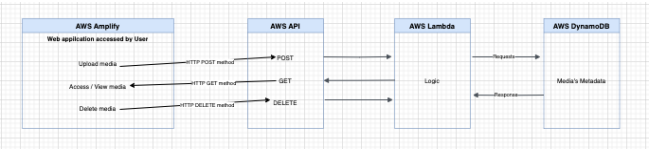
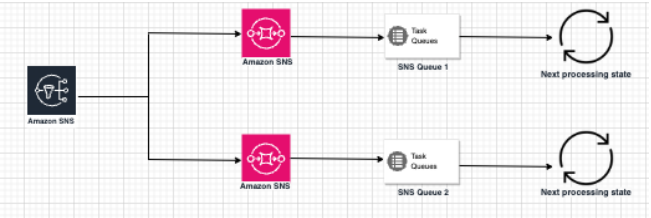


Figure 3: Diagram for architecture

Since three-tier architecture is well-known for its reliability, flexibility, high scalability and evolution. It offers numerous benefits in terms of creating a serverless and high-availability application.

C. Fanout design

Handling requests is a controversial issue when it comes to designing a system, especially when the amount of users that send requests is unpredictable. One available solution for this problem is to use AWS Auto Scaling Group (ASG) to automatically scale up the amount of instances when



needed. However, this solution could be pricey for big companies who could encounter a large amount of traffic daily. Another solution is to use the fanout architecture, which is a more economical yet effective solution.

Figure 4: Fanout architecture

This approach involves using AWS Simple Notification System (SNS), AWS Simple Query System (SQS), AWS Lambda and AWS API Gateway to distribute multiple requests onto the right processing flow. Particularly, the SNS will be triggered to publish messages into the right topic when receiving the API request from the user, it then directs those messages to the queue for further proceedings.

D. Media Processing

After messages are put on the queues, those messages will be directed to the Media Processing Steps, which consists of a range of Lambda functions to transcode and

save those informations into S3 buckets and DynamoDB in order to query the information easier. Refer to the *Appendix A* for more details about implementations and *Appendix B* for the UML explanation of how this step works

IV. Design Principle

In this stage, we will discuss further on the given architecture, we will debate on how this architecture is suitable to solve the pimples that have been identified in the business scenario. Furthermore, we are going to evaluate the solution based on the criteria of performance, scalability, reliability and security.

A. Posting and getting media

Rather than hosting a server using EC2 in a private subnet, which will only be visible to the user via a NAT gateway, this results in harder to maintain and low access. By hosting servers on Amplify and storing the source code in a S3 bucket, the need for instances and gateways are non-compulsory thus making the design more straightforward yet maintain the availability advantage. The process of posting and getting media are decoupled into two separate procedures as part of fanout architecture instead of doing it all-in-one. As the media requests, either POST or GET, passing through Amplify, it will be routed to the categorizing Lambda function via API gateway, and the Lambda function will decide which procedure is suitable for that media's type and send it there for further processing.

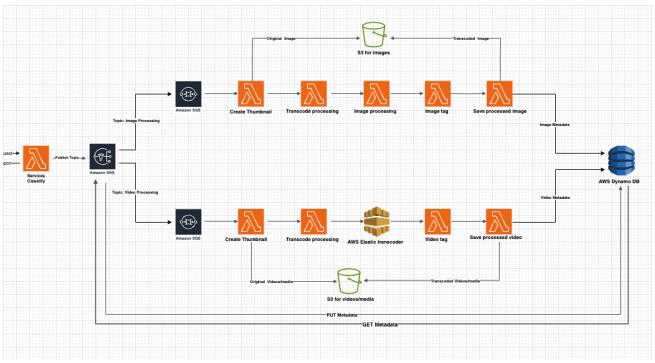


Figure 5: Procedure Decoupling

B. Traditional vs Serverless Three-tier design

In the traditional three-tier architecture, which will mostly use VPC for ensuring security, the administrator must guard for the "health" status of both scaled and bastion instance for any further updates or maintenance. This might be useful for small programs with less user access, however that would not be the case when it comes to the website where the expected user's access is enormous as it could lead to higher monthly cost and be harder to guard. With the serverless architecture in action, the Amplify instance will do the scaling job by itself without the need of checking the instance's status, thus reducing the workload for the administrator. In addition, when it comes to security, Amplify provides a range of automated VPC-based security methods to ensure the process of authentication, authorization and data encryption are automated and has same functionality as the traditional VPC design, which requires configuring Security Group and Network Access Control List (NACL) for each instances and subnets. Moreover, AWS IAM is also integrated into Amplify to fine-grained user's permission to the particular instances.

C. Design solution on given criteria

Criteria	Solution
• Swift uploading and downloading of media. • Providing quick and smooth website performance, with little delays even when dealing with significant traffic and huge images files, particularly for users outside of Australia (globally).	• **Leverage CloudFront:** serving as a dynamic caching mechanism for the website's static portions, along with the distribution of content across edge locations to expedite its delivery to end-users. • **Employ S3 Transfer Acceleration**: An S3 feature that may greatly speed up the transfer of material to and from the storage for the effective transmission of massive data over long distances. • **Employ Route 53:** ensuring the best responsiveness by intelligently routing incoming user traffic to the closest CloudFront edge point depending on geographic location. • **Put Elastic Transcoder:** for work by cleaning up media assets and transforming them into the ideal formats and resolutions. to access the internet. • **Employ CloudWatch:** keep an eye on and measure the performance of your website by recording important metrics (response time, resource consumption, etc.)

Table 1: Functioning criteria

Criteria	Solution
• It should be simple for the website to accommodate more users and data, split up work across several servers or geographical locations, and modify resources in response to traffic patterns. • To accommodate future additions like AI-based media labelling, the system should be flexible and reusable.	• **Employ DynamoDB:** enable scalable retrieval and storage of metadata. • **Employ CloudFront**: increase your ability to serve content globally and to be prepared for sudden increases in traffic. • **Add more Lambda function:** to have the ability to decouple, creating flexible conditions for future changes. • **Employ API Gateway**: on the backend, providing a scalable RESTful API, that can handle API calls and grow as needed. • **Leverage AWS Amplify**: seamlessly enhance both of the application's front-end and back-end. • **Use S3:** improve storage and content retrieval capabilities. • **Employ Amazon SNS and SQS:** to manage queues and send messages effectively, especially while handling large numbers of requests.

Table 2: Scalability criteria

Criteria	Solution
• Make sure the website is incredibly dependable and has frequent backups to avoid data loss so it can rapidly recover from any problems or outages.	• **Leverage CloudFront:** To oversee failover scenarios and distribute content from the closest reachable boundary position. • **Employ DynamoDB:** to duplicate data across several locations and guarantee excellent availability. • **Implement Amazon S3:** for dependable and easily accessible storage. • **Employ Amazon DynamoDB:** make a NoSQL database that provides remarkable availability along with scalability.

Table 3: Dependability criteria

Criteria	Solution
• Safeguard user data, such as uploaded files and private information. • Assure the protection of the website from data loss, theft, or unapproved entry.	• **Employ AWS IAM**: to handle permission and entry management. • **Implement API Gateway:** to establish permission for the backend API. • **Employ CloudWatch:** Keep an eye out for anomalies and security concerns. • **Employ Amazon 53:** to store data in object storage while maintaining precise access control.

Table 4: Security criteria

D. Operation cost

The provided cost below is created to fit with the given demand, which the application will periodically double its size every 6 months and will continue to expand for the next 2 to 3 years. The operating cost will be divided into the upload sizes of 25GB, 75GB and 125GB.

The summary cost for the architecture to operate with 25 gigabytes upload size in the first 6 months is **$75.5364456**, which has included all of the components in the presentation tier, logic tier and data tier. Likewise, the operating cost for 75 gigabytes and 125 gigabytes is **$223.0843368** and **$596.756728** , appropriately. For specific details, refer to *Appendix C*, *Appendix D* and *Appendix E.*

E. Rationale on the architecture

1. Expandability

By using various AWS cloud services, such as Lambda functions that are able to expand itself based on the incoming traffic, S3 buckets for

strong static web contents and DynamoDB for large and persistent database storage. This architecture ensures a well handling when it comes to high requesting volume from users.

Moreover, to strictly follow the approach of event-driven design, Simple Notification System, Simple Query System and Lambda function to automatically direct requests into the right processing section are used in order to guarantee a decoupled design thus increasing the expandability and reliability of the architecture. Overall, this approach is able to deal with high pressure of traffic flow without any constant guarding, which fulfills the criteria of biannual expansion of the website.

2. *Database optimization*
   DynamoDB is a persistent NoSQL database storage that offers various advantages to the other Amazon database storage services, such as RDS. DynamoDB will make sure that there will always be available storage for incoming databases by elastically scaling the resources in multiple availability zones based on the volume of traffic.

   Moreover, DynamoDB is a profitable storage solution, as it will charge based on the data quantity and the consumed throughput of the application. Therefore, DynamoDB is an outstanding candidate compared to the other solutions for a limited budget, as it will ensure the scalability, flexibility of the storage based on the request capacity while still maintaining the top-notch user experience in every corner of the world.
3. *Media processing*
   The design aims for revolutionizing the way that media files are being performed while the current features are still getting up-to-date. In order to serve the purpose of increasing the visitor retention rate while still maintaining an edge advantage in the market.

   This advantage's implementation is finished by using the fan out architecture, which is a pattern that is designed to distribute traffic into multiple processing nodes. Specifically, the system will trigger Lambda function with the integration of AWS SNS, SQS to divide media files into the right procedure, which will ensure that the files are correctly placed on the correct procedure without any interruption from manual intervention. Furthermore, this procedure is designed in a decouple way, in order to guarantee for reusability and future improvements. For instance, new features and procedures could be added later when needed without any fear of interfering with existing procedures.
4. *Multi-region accessing performance*
   Since the website is growing not only inside but outside of Australia, it is essential to deliver the same quality of user experience for the international customers. It is easily achievable by using CloudFront and Amplify, which are the two main components to handle global traffic. CloudFront will be responsible for caching and delivering contents into the user's nearest location using EdgeLocations feature, making a minimal delay and faster delivery of content to the user. Furthermore, CloudFront is also integrated in Amplify, which Amplify will take all of the static contents and assets that are fetched by CloudFront. Therefore, this solution is capable of handling excellent cross-region performance, thus leading to a top user experience and raising the business outcomes.

V. Conclusion

In general, implementing a serverless/event-driven architecture on AWS platform could benefit any business that is in need of an expandable and cost-effective system. This system can greatly handle real-time events without the need of manual guarding thanks to services such as Lambda, SNS and SQS, allowing the owner to concentrate on bringing the greatest experience to the users.

References

[1] "Elastic Transcoder – Amazon Web Services," Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/elastictranscoder [Accessed: Oct. 22, 2023].

[2] "AWS Amplify | Build full-stack web and mobile apps in hours. Easy to start, easy to scale,", Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/amplify/?nc=sn&loc=0 [Accessed: Oct. 22, 2023].

[3] "Host a Static Website | Services used and costs", Amazon Web Services, Inc. [Online]. Available:https://aws.amazon.com/getting-started/projects/host-static-website/services-costs/ [Accessed: Oct. 22, 2023].

[4] Jeremy Daly "How To: Use SNS and SQS to Distribute and Throttle Events", Jeremy Daly [Online]. Available: https://www.jeremydaly.com/how-to-use-sns-and-sqs-to-distribute-and-throttle-events [Accessed: Oct. 22, 2023].

[5] "Amazon CloudFront - Securely deliver content with low latency and high transfer speeds", Amazon Web Services, Inc. [Online]. Available: https://aws.amazon.com/cloudfront [Accessed: Oct. 22, 2023].

[6] "AWS Pricing Calculator - Estimate the cost for your architecture solution.", Amazon Web Services, Inc. [Online]. Available: https://calculator.aws/ [Accessed: Oct. 22, 2023].

Appendix A: Media Processing Procedure

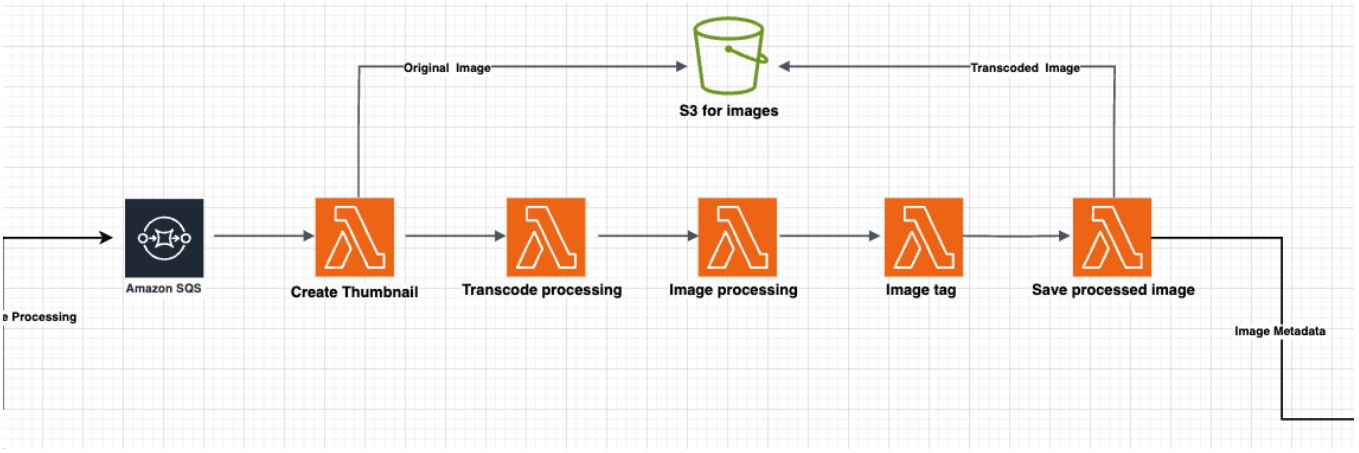


Figure 6: Media Processing for Image

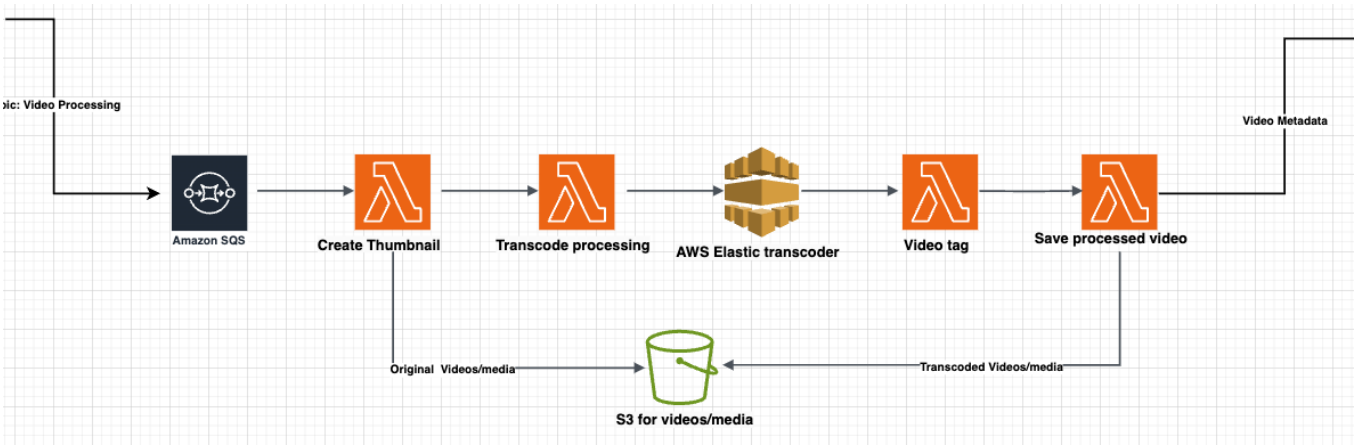


Figure 7: Media Processing for Video

In this implementation procedure, AWS Simple Query Service will trigger a range of Lambda functions to process the media file. Firstly, it will create the thumbnail for the uploaded media and store it into the S3 bucket with the original media file, note that the thumbnail and the original media file will be stored separately in two distinct folders. Secondly, it will move the media files into the process of transcoding and processing the media files, which could involve detecting the formats that the website could transcode and transcoding the new media files based on the qualified formats. Finally, the media files will be transferred to the last Lambda function to check if the media is fully transcoded before those files could be saved onto the S3 folder. Metadata of the media will also be routed to DynamoDB and getting stored. This procedure is a highly reusable solution, allowing for extending or modifying the existing step to satisfy the requirement. For instance, if the owner wants to implement something new into this processing step, the owner could simply add another Lambda function, or a range of Lambda functions below those existing ones without any interference to the existing functions.

Appendix B: UML Diagrams for illustrating the features of the architecture

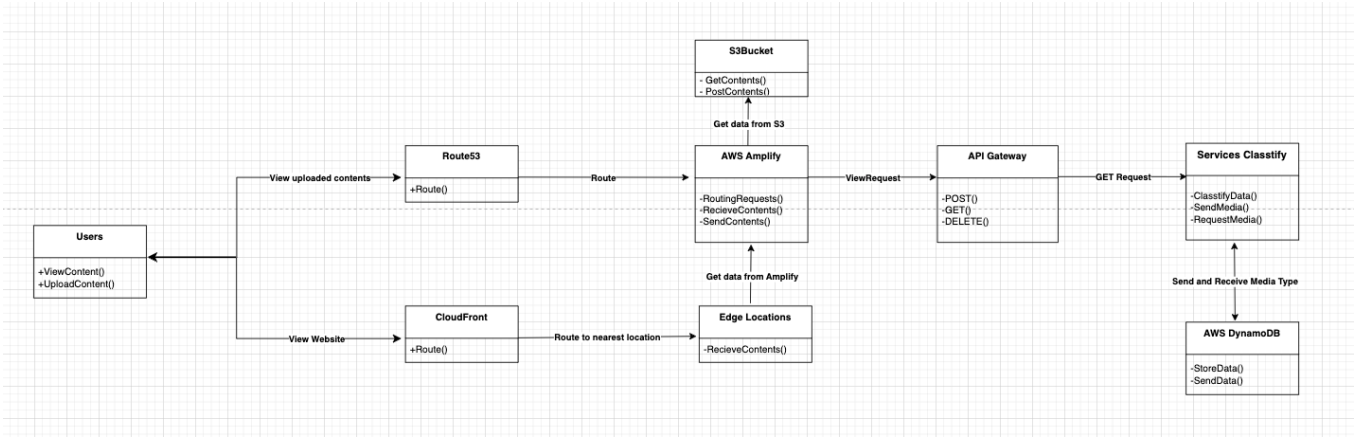


Figure 8: View website

When accessing the website, users will be redirected to the nearest location by CloudFront, Amplify will be responsible for taking the static web contents and media stored in S3 bucket and displaying it. When it comes to display the uploaded media files, instead of routing directly to CloudFront, users will be directed through Route 53 and Amplify will send an API request to the API gateway. The API gateway then sends the request to the ServicesClassify Lambda function to get the metadata and return it to Amplify, which will display all taken metadata from the DynamoDB.

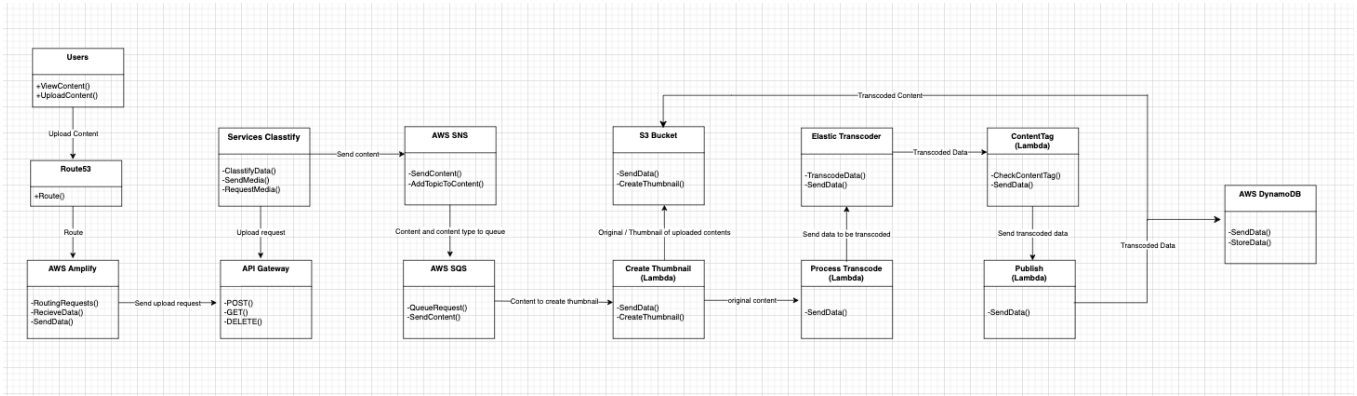


Figure 9: Upload contents

In advance of uploading media files to the website, the first few steps are basically the same as the viewing uploaded media procedure, in which users will go through Route 53 and CloudFront to Amplify. However, instead of sending a GET request, the API gateway will send the POST request, which will send the information to the ServicesClassify Lambda function. After receiving the POST method, the ServicesClassify function will attach a topic based on the media's type (i.e. media/processing/image, media/processing/video, e.t.c.) that aids the SNS to know where to send the data. The SNS then sends media to the designated queue of the SQS for processing. At the processing step, the original media and its data type will be sent to a specific folder in the S3 bucket via the CreateThumbnail Lambda function. Then the media is transcoded with TranscodeProcessing and ImageProcessing functions (or Elastic Transcoder for videos) and will be on the final check to see if the files are in the right transcoded format before saving those objects into the S3 bucket and DynamoDB.

Appendix C: Operating cost for 25GB of media upload

Service	Cost of each feature	Total
DynamoDB	• Write Request Unit (WRU): 1WRU = 1KB/ 1 item 25 x 1024^2 / 1000000 x $1.25 = **$32.768** • Read Request Unit (RRU): 1 RRU = 4KB/ 1 item 25 x 1024^2 / 1000000 / 4 x $0.25 = **$1.6384** • Data retention (each month): 25 x 0 (First 25GB is free per month) = 0 • Data ingress: **$0** • Data egress (each month): **$0**	$32.768 + $1.6384 = $34,4064
Route 53	In the case where all the uploaded files are images, and has the size of each image is 100 kilobytes: • DNS zone (each month): **$0.50** • Typical queries (each month): 25 x 1024^2 / 100 / 1000000 x $0.40 = **$0.1048576**	$0.50 + $0.1048576 = **$0.6048576**
CloudFront	Considering the price in Australia as the price that's in the middle or average: • Data egress (each month): $0.114 x 25 = **$2.85** • The typical image size uploaded to the website is 100kB, and everything uploaded is in picture format. Now, imagine each of these pictures is associated with three HTTPS queries. The cost for these HTTPS requests could climb as high as: • 25 x 1024^2 / 100 / 10000 x 3 x $0.0125 = **$0.98304**	$2.85 + $0.98304 =**$3.83304**
S3	S3 Standard - Infrequent Access: $0.0125 x 25 = **$0.3125** • Data ingress: **$0** • Data egress (each month): $0.09 x 25 = **$2.25**	$0.3125 + $2.25 = **$2.5625**

Amplify	During the initial 12 months, there is no charge => $0.	**$0 First 12 months**
API Gateway	Every image is 100kB in size. • API Calls (each month): Up to: 25 x 1024^2 / 100 / 1000000 x $3.50 = **$0.917504**	**$0.917504**
SNS	Since the endpoint is Simple Queue Service (SQS), there are no charges for using SNS.	**$0**
SQS	Every image is 100kB in size. • First-In-First-Out Queues (each month): **$0** • Data ingress **$0** • Data egress (each month): $0.09 x 25 = **$2.25**	**$2.25**
Elastic Transcoder	Picture this scenario: We've utilized a total of 25GB for uploading 720p videos, and all these videos add up to roughly 200 minutes in length. The expense for Elastic Transcoder could reach a maximum of: $0.03 x 200 = **$6**	Up to **$6**
Lambda	Scenario: Every piece of uploaded content is an image, and each image has 100kB for size. Now, consider that every one of these images is linked to five requests. • Requests: 25 x 1024^2 / 100 / 1000000 x 5 x $0.20 = **$0.262144** • Memory: **$5.5**	$0.262144 + $5.5 = **$5.762144**
CloudWatch	Imagine there are eight Lambda functions, and each of these functions has eight associated metrics. • Metrics (each month): 8 x 8 x 0.3$ = **$19.2**	**$19.2**
Total		**$75.5364456**

Table 5: Cost for 25GB

Appendix D: Operating cost for 75GB of media upload

Service	Cost of each feature	Total
DynamoDB	• Write Request Unit (WRU): 1WRU = 1KB/ 1 item 75 x 1024^2 / 1000000 x $1.25 = **$98.304** • Read Request Unit (RRU): 1 RRU = 4KB/ 1 item 75 x 1024^2 / 1000000 / 4 x $0.25 = **$4.9152** • Data retention (each month): Up to now, the total of data is about: 25 x 6 + 75 = 225 GB. 25 x $0 + 225 x $0.25 = **$56.25** • Data ingress: **$0** • Data egress (each month): **$0**	$98.304 + $4.9152 + $56.25 = **$159,4692**
Route 53	In the case where all the uploaded files are images, and has the size of each image is 100 kilobytes: • DNS zone (each month): **$0.50** • Typical queries (each month): 75 x 1024^2 / 100 / 1000000 x $0.40 = **$0.3145728**	$0.50 + $0.3145728 = **$0.8145728**
CloudFront	Considering the price in Australia as the price that's in the middle or average: • Data egress (each month): $0.114 x 75 = **$8.55** • The typical image size uploaded to the website is 100kB, and everything uploaded is in picture format. Now, imagine each of these pictures is associated with three HTTPS queries. The cost for these HTTPS requests could climb as high as: • 75 x 1024^2 / 100 / 10000 x 3 x $0.0125 = **$2.94912**	$8.55 + $2.94912 = **$11.49912**

S3	S3 Standard - Infrequent Access: $0.0125 x (25 x 6 + 75) = **$2.8125** • Data ingress: **$0** • Data egress (each month): $0.09 x 75 = **$6.75**	$2,8125 + $6.75 = **$9,5625.**
Amplify	During the initial 12 months, there is no charge => $0.	**$0**
API Gateway	Every image is 100kB in size. • API Calls (each month): Up to: 75 x 1024^2 / 100 / 1000000 x $3.50 = **$2.752512**	**$2.752512**
SNS	Since the endpoint is Simple Queue Service (SQS), there are no charges for using SNS.	**$0**
SQS	Every image is 100kB in size. • First-In-First-Out Queues (each month): **$0** • Data ingress **$0** • Data egress (each month): $0.09 x 75 = **$6.75**	**$6.75**
Elastic Transcoder	Scenario: We've utilized a total of 75GB for uploading 720p videos, and all these videos add up to roughly 200 minutes in length. The expense for Elastic Transcoder could reach a maximum of: $0.03 x (25 x 6 + 75) = **$6.75**	Up to **$6.75**
Lambda	Scenario: Every piece of uploaded content is an image, and each image has 100kB for size. Now, consider that every one of these images is linked to five requests. • Requests: 75 x 1024^2 / 100 / 1000000 x 5 x $0.20 = **$0.786432** • Memory: **$5.5**	$0.786432 + $5.5 = **$6.286432**
CloudWatch	Imagine there are eight Lambda functions, and each of these functions has eight associated metrics. • Metrics (each month): 8 x 8 x 0.3$ = **$19.2**	**$19.2**

Table 6: Cost for 75GB

Appendix E: Operating cost for 75GB of media upload

Service	Cost of each feature	Total
DynamoDB	• Write Request Unit (WRU): 1WRU = 1KB/ 1 item 125 x 1024^2 / 1000000 x $1.25 = **$163.84** • Read Request Unit (RRU): 1 RRU = 4KB/ 1 item 125 x 1024^2 / 1000000 / 4 x $0.25 = **$8.192** • Data retention (each month): Up to now, the total of data is about: 25 x 6 + 75 x 6 + 125 = 725 GB. 25 x $0 + 700 x $0.25 = **$175** • Data ingress: **$0** • Data egress (each month): **$0**	$163.84 + $8.192 + $175 = **$347.032**
Route 53	In the case where all the uploaded files are images, and has the size of each image is 100 kilobytes: • DNS zone (each month): **$0.50** • Typical queries (each month): 125 x 1024^2 / 100 / 1000000 x $0.40 = **$0.524288**	$0.50 + $0.524288 = **$1.024288**

CloudFront	Considering the price in Australia as the price that's in the middle or average: • Data egress (each month): $0.114 x 125 = **$14.25** • The typical image size uploaded to the website is 100kB, and everything uploaded is in picture format. Now, imagine each of these pictures is associated with three HTTPS queries. The cost for these HTTPS requests could climb as high as: • 125 x 1024^2 / 100 / 10000 x 3 x $0.0125 = **$4.9152**	$14.25 + $4.9152 = **$19.1652**
S3	S3 Standard - Infrequent Access: $0.0125 x (25 x 6 +75 x 6 + 125) = **$9.0625** • Data ingress: **$0** • Data egress (each month): $0.09 x 125 = **$11.25**	$9.0625 + $11.25 = **$20.3125**
Amplify	Assume web app size = 200MB, average size of page requested = 2.2 MB Assume that average build time = 2 minutes each day. Assume that daily active users = 15000. • Build & Deploy: 2 x 30 x 0.01 = **$0.6** • Data retention: 200 / 1024 x $0.023 = **$0.0045** • Data egress: 15000 x (2.2 / 1024) x 30 x $0.15 = **$145.02**	**$145.6245**
API Gateway	Every image is 100kB in size. • API Calls (each month): Up to: 125 x 1024^2 / 100 / 1000000 x $3.50 = **$4.58752**	**$4.58752**
SNS	Since the endpoint is Simple Queue Service (SQS), there are no charges for using SNS.	**$0**

SQS	Every image is 100kB in size. • First-In-First-Out Queues (each month): **$0** • Data ingress **$0** • Data egress (each month): $0.09 x 125 = **$11.25**	**$11.25**
Elastic Transcoder	Picture this scenario: We've utilized a total of 125GB for uploading 720p videos, and all these videos add up to roughly 200 minutes in length. The expense for Elastic Transcoder could reach a maximum of: $0.03 x 725 = **$21.75**	Up to **$21.75**
Lambda	Let's create a scenario where every piece of uploaded content is an image, and each image has 100kB for size. Now, consider that every one of these images is linked to five requests. • Requests: 125 x 1024^2 / 100 / 1000000 x 5 x $0.20 = **$1.31072** • Memory: **$5.5**	$1.31072 + $5.5 = **$6.81072**
CloudWatch	Imagine there are eight Lambda functions, and each of these functions has eight associated metrics. • Metrics (each month): 8 x 8 x 0.3$ = **$19.2**	**$19.2**
Total		**$596.756728**

Table 6: Cost for 125GB

Appendix F: Detailed architecture diagram