COS20019 – Cloud Computing Architecture

Assignment 3

Serverless/Event-driven Architecture Design Report

Nhat Minh Nguyen Dinh - 10380249
Faculty of Science, Engineering and Technology
Swinburne University of Technology

Vi Luan Dang - 103802759
Faculty of Science, Engineering and Technology
Swinburne University of Technology

Quoc Anh Vu - 10379279
Faculty of Science, Engineering and Technology
Swinburne University of Technology

Abstract

Cloud computing has gained increasingly popularity for developing online architecture. Platform like Amazon Web Service allows developers to leverage managed services that handle the underlying infrastructure, allowing them to focus on application logic rather than server, networking equipment , and system configuration. This paper will propose a design and implementation for a serverless/event-driven architecture to improve the performance and efficiency of a business based on their scenario. The paper will leverage AWS Lambda, API Gateway, DynamoDB, SQS and other AWS services. Apart from proposing an architecture design, the paper will also discuss the design rationale regarding the design and justification for such design. Finally, the paper will look through some noticeable alternative regarding the proposed architecture.

Keywords: Cloud computing, AWS, Serverless/Event-driven, business scenario, .rationale design.

1. Introduction

Digital technologies play an increasingly important role in every operation of business nowadays. Cloud computing has developed to provide both business owner and the developer with a new tool to establish business foothold online. Cloud computing has revolutionized the way business approach online architecture, providing a cost-effective and efficient solution for managing infrastructure. It has shortened the gap between creating and managing a website on-premises and in the cloud, offering a more worriless web operating experience. Furthermore, the independence nature of AWS's services has created the opportunity for serverless/event-driven approach to increase efficiency and flexibility of web operation.

This paper proposes a serverless/event-driven architecture design and implementation using AWS services, aimed at improving the performance and efficiency of a business based on their specific scenario. Therefore, the business scenario of the company should be thoroughly examined before the design can be provided.

2. Business Scenario

The Photo Album application has experienced tremendous success and requires further development to meet the increasing demand. The following requirements and problem have been identified and measures have to be taken to leverage these matters so as to provide the customers with the best experience:

2.1 Infrastructure management:

- Use managed cloud services to minimize in-house system administration and storage.

2.2 Scalability:

- Design architecture have to cope with expected growth.
- Adopt a serverless/event-driven solution.

2.3 Database optimization:

- Explore cost-effective options for the relational database.

2.4 Cross-region performance:

- Improve global responses times.

2.5 Media processing evolution and optimization

- Handle video media in the future.
- Automatically produce various media versions upon media upload.
- Ensure extensible architecture for media processing.
- Ensure the architecture is effectively decoupled to avoid overloading.

The following design will address the above requirement and further discussion will justify the effectiveness as well as the performance of the design.

3. AWS Architecture Diagram

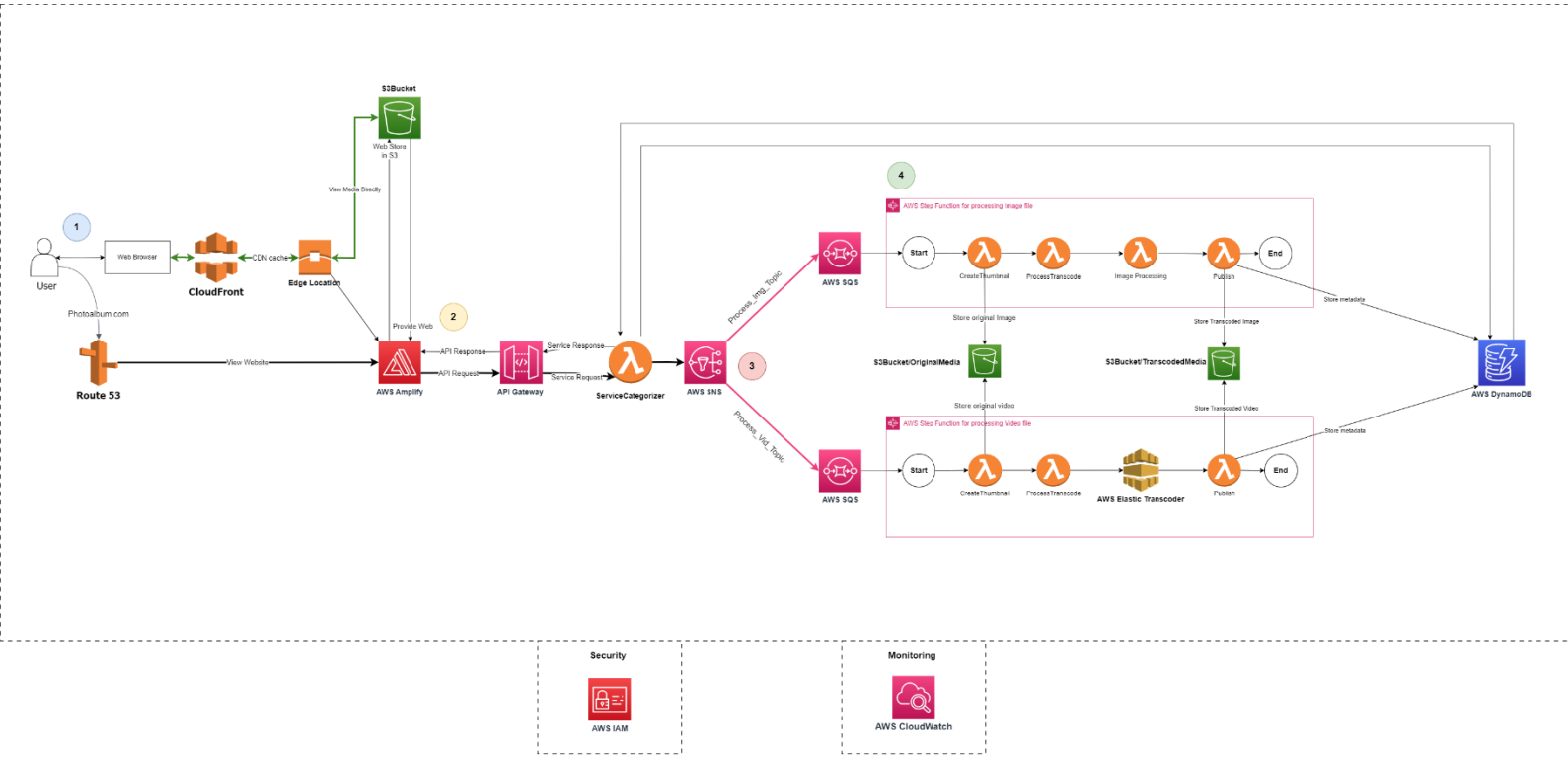

Figure 1: AWS Architecture Diagram

This diagram can be divided further into 4 smaller parts so as to provide a better explanation and operation of the architecture.

3.1 AWS Content Delivery System

The ever-growing popularity of the website require an equally efficient content delivery system so as to provide the best experience for the user as well as upkeeping a competitive edge over the competitors. This can be achieved using AWS DNS resolution such as AWS Route 53 and AWS Content Delivery Service such as AWS CloudFront:

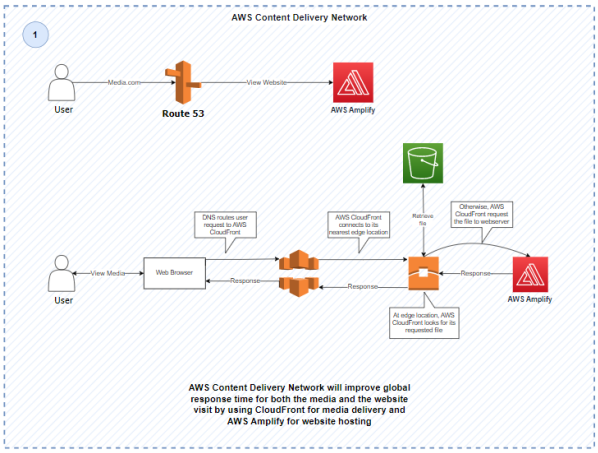

Figure 2: AWS Content Delivery System

There are two methods by which the user may user the website's services, the user can browse our website via a Domain Name Server after which, they will be routed to our website hosted on AWS Amplify with the help of Route 53. AWS Route is an amazing service for business owners to maintain. a well-regarded identity on the internet using a well-suited Domain Name Server.

In addition, our website can utilize a Cloud-based media streaming service similar to that of Google Drive for video and image content. This entails the use of AWS CloudFront to provide media to users via the nearest edge location. In instances where the media is not available in the CloudFront cache, the service will issue a request to our web server to retrieve the designated media. This approach ensures fast and efficient delivery of content to users, improving their overall experience on our website.

3.2 AWS Serverless Three-Tier Architecture

The most widely used implementation of a multi-tier architecture is the three-tier architecture, which is composed of three distinct layers: the presentation tier, logic tier, and data tier.

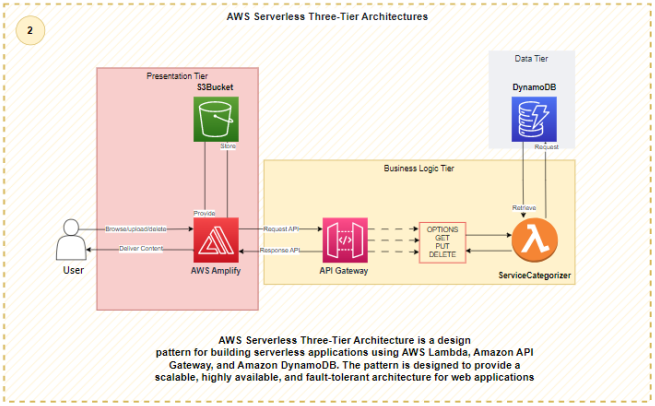

Figure 3: AWS Serverless Three-Tier Architecture

In the above architecture, static web content is stored on Amazon S3 and hosted on Amazon Amplify, which will make up of our presentation tier. The logic tier consists of API Gateway and Lambda functions to provide dynamic content for our website. Finally, the data tier will utilize AWS DynamoDB as our persistent storage. Various users' request can be supplemented with this architecture:

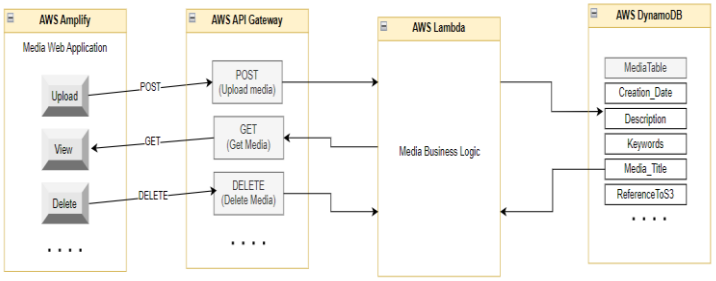

Figure 4: Web architecture UML Diagram

Using a serverless three-tier architecture on AWS offer several key advantages, including high availability, reliability, and flexibility for future scalability and evolution.

3.3 Reliable Fanout Architecture

Fanout architecture is a common design pattern in AWS that allows for efficient distribution of data to multiple downstream services or systems.

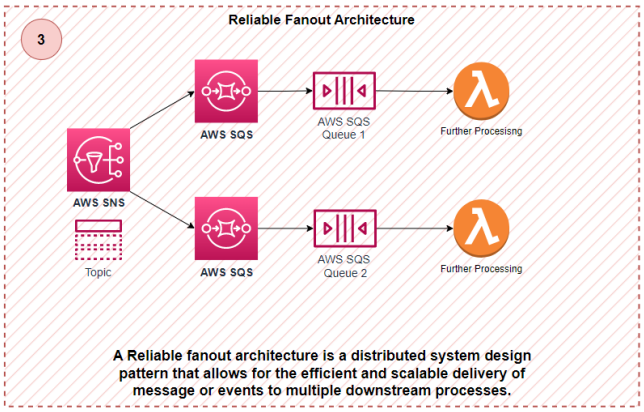

Figure 5: Fanout Architecture

This architecture involves using an event-driven approach where a single event is triggered and then distributed to multiple recipients. Particularly, in this system, this is achieved using Amazon SNS, AWS SQS, and AWS Lambda.

API calls from previous process will trigger AWS SNS to publish messages to a topic, AWS SQS is then used to queue messages, and AWS Lambda is used to process the messages. This architecture allows for decoupling of application components, scalability, and reliability.

3.4 Media Processing Steps

Finally, after being queued for processing, the media will be sent to a Media Processing Steps by AWS SQS and a Step Function will be provoked to handle this processing procedure. For more information, see Appendix A.

To provide an overview of the architecture's function and feature, see the UML diagram of the architecture at Appendix B.

4. Design Rationale

Having completed our discussion on the AWS architecture, we can proceed to examine how this design is capable of satisfying the pain points that have been identified in the business scenario described above. Further discussion above alternative solution and design criteria will also be included.

4.1 Alternative design

There are two versions of the AWS architecture that was created during the architecture building phase; the original architecture (Appendix C) used more simple AWS services; however, it is fundamentally more complex than the alternative, which is the currently used architecture.

Viewing and Uploading media.

Users who want to view media in the initial architecture will be routed via CloudFront to an EC2 instance that hosts the website; the website is in a private subnet, so it is only visible to the user via a NAT gateway.

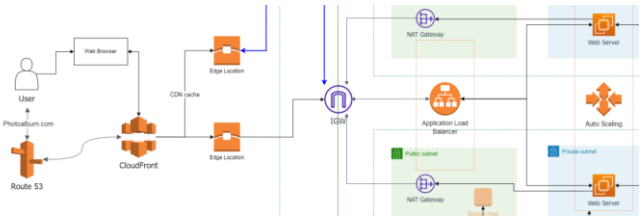

Figure 6: View Website on instance

In the second architecture, the user will access the website with its source code stored in the S3 Bucket via AWS Amplify. The process of the second architecture is more straightforward than that of the first one, as there is no need for instances or gateways; this also reinforces the high availability advantage of this architecture.

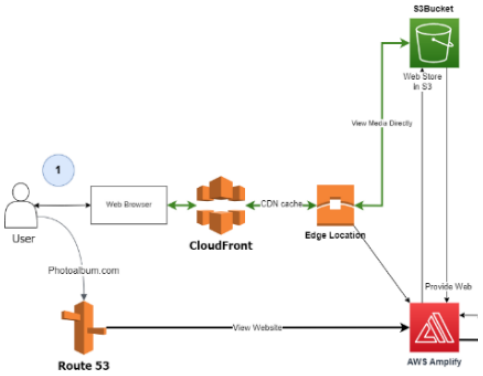

Figure 7: Website on AWS Amplify

As per the process of uploading media to the website, the initial design sends the original media to the S3 bucket, which triggers the step function to process the media and send the processed media back to the bucket. As the media is uploaded into the bucket, its metadata will then be sent to the database; the same is true for metadata that the AWS Transcoder processes.

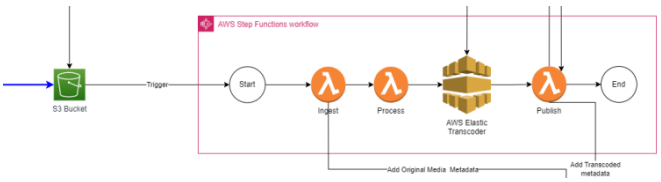

Figure 8: Process upload media

The process for uploading is somewhat different in the second alteration; as the media passes through AWS Amplify, it will trigger API to send a put request to the ServiceCategorizer, and the categorizer will send the media to the Reliable Fanout process that handles the media's type.

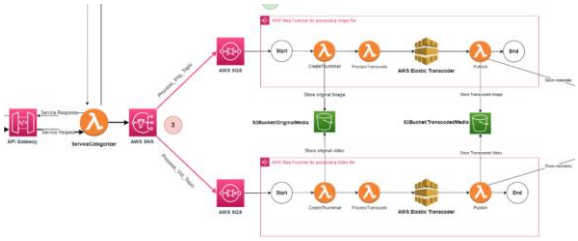

Figure 9: Decouple media.

VPC vs. Serverless three-tier architecture

The serverless three-tier architecture is chosen for its ease of management, as AWS will automatically do some of the management, like scaling and ensuring security. In a VPC like the original architecture, even though there is an auto-scaling group, the admin must watch for the instances and the bastion host, as they might need to be updated. With the serverless architecture, however, AWS Amplify will be the one to do all the scaling; this gives the admin much more time to improve and ensure performance, rather than spending much time monitoring scaling.

Security is also a hassle within the VPC environment; each item in the original VPC must be configured as a security group or NACL by the admin manually, which can be frustrating and challenging to figure out. For the architecture in use, however, AWS Amplify will provide security through authentication, authorization, and data encryption, giving it an edge over VPC-based applications. Additional security is granted via the IAM role.

It is important to note that the serverless three-tier architecture has limitations regarding infrastructure; however, it is not the primary concern so it can be negligible.

4.2 Design Criteria

After careful consideration of services provided by AWS, the following tables will discuss the design criteria for each pillar of well-architecture framework.

Performance

Criteria	Solution
• Fast media upload and download speeds. • Efficient storage and retrieval of photos. • The website should load quickly and efficiently with minimal latency, even with high traffic and large photo files, or the users are in countries outside Australia.	• Use CloudFront to cache static content of the website, as well as distributing content to edge locations for faster delivery to users. • Use Route 53 to route traffic to the nearest CloudFront edge location based on the user's location. • For long-distance transfer of large items, use S3 Transfer Acceleration, a feature of S3, to accelerate content transfers to and from S3 by up to 50–500%. • Use CloudWatch to monitor the performance of the website, including latency, error rates, and resource usage. • Use Elastic Transcoder for optimizing media (transcoding media into appropriate formats and resolutions) for web viewing.

Table 1: Performance criteria

Scalability

Criteria	Solution
• The website is able to handle increasing traffic and growing storage needs, distribute workload across multiple servers or regions, and automatically adjust resources based on traffic patterns. • The system procedure is highly reusable and flexible, which will be good for future modification (e.g. adding more features such as identifying tags in media using AI, etc.)	• Use CloudFront to scale the delivery of content to users globally and to handle spikes in traffic. • Use API Gateway to create a scalable RESTful API for the backend of the website, handle API requests and scale as if needed. • Use AWS Amplify to easily scale the frontend and backend of the application. • Use DynamoDB to scale the metadata storage and retrieval. • Use S3 to scale the storage and retrieval of content. • Use Amazon SNS and SQS for messaging and queueing to handle large numbers of requests. • The components in the system is designed to be decoupling enough to make the modifying process become flexible (e.g. easy to add a new Lambda function for a new feature in the future).

Table 2: Scalability criteria

Reliability

Criteria	Solution
• The website is highly available and able to recover quickly from failures (regular backups to prevent data loss).	• Use Amazon S3 for durable and highly available storage. • Use DynamoDB to replicate data across multiple regions for high availability. • Use Amazon DynamoDB for a highly available and scalable NoSQL database. • Use CloudFront to handle failovers and deliver content from the nearest accessible edge location.

Table 3: Reliability criteria

Security

Criteria	Solution
• User data (personal information, uploaded media, etc.) is protected. • The website is protected from unauthorized access, theft, or destruction of data.	• Use AWS IAM for access control and permissions management. • Use Amazon S3 for secure object storage with granular access control. • Use CloudWatch to monitor and detect security threats and anomalies. • Use API Gateway to implement authentication and authorization for the backend API.

Table 4: Security

Cost

According to the provided information, the demand for the application has been doubling every 6 months, and it will continue to grow under that trend for the next 2 or 3 years. To predict how much it will cost to run the website in the future, cost summaries for total media upload sizes at 50GB, 100GB and 200GB are shown below:

The total cost of operation for the architecture for 50GB will be **$124.54182.** This cost has included the Content Delivery Network, Web architecture, and the media processing procedure. Detailed cost of each feature can be view via Appendix D.

Similarly, the forecast for 100GB and 200GB in the upcoming year would be **$308.88364** and **$744.38528** respectively. For more information, see Appendix E and Appendix F.

4.3 Fulfillment of the business scenario requirements and justification for designed architecture

a) Infrastructure management

The architecture utilizes AWS managed cloud services, such as AWS S3, AWS Amplify, AWS CloudFront, AWS DynamoDB, and AWS Lambda to minimize the need for in-house system administration and storage. Security and Monitoring services provided by AWS also play an important role in easing the burden of business owners allowing them to focus on other aspects of the business. Overall, the architecture's use of managed cloud services can bring significant improvements in scalability, reliability, and cost-effectiveness, solving the pain point for infrastructure management.

b) Scalability

With the constant growth of the website biannually, the proposed architecture above is designed to address the need to cope with expected growth by adopting a serverless/event-driven solution.

This is done via adopting various managed cloud services, such as AWS Lambda functions, which can automatically scale up or down based on the incoming traffic. Accordingly, the used of AWS S3 to store static web content and AWS DynamoDB as a persistent storage solution ensures that the web application can handle a high volume of request without any decrease in performance.

In addition, the event-driven approach of the architecture using AWS SNS, AWS SQS ,and AWS Lambda, ensures that the application components are decoupled, which in turns will increase the scalability and reliability of the architecture.

Overall, this approach allows the architecture to handle expected spike and decline in traffic without requiring on-premises system administration.

c) Database optimization

Instead of previous use of AWS RDS in the old architecture for the business, the use of DynamoDB, which is a fully managed NoSQL database service, will ensure automatic scaling of resources based on spike and decline of visit. Meaning that application can adapt to scalability and inflexibility. The serverless nature of DynamoDB will also be a cost-effective solution to our architecture, as DynamoDB will charge based on the amount of data stored and the throughput consumed according to the application. The replication of data across multiple availability zone of DynamoDB will also ensure the best user experience and ease of operating for the business owner. AWS RDS, on the other hand, requires manual scaling and maintenance of resources.

Therefore, the use of DynamoDB instead of RDS provides a cost-effective, scalable, and highly available database solution for our web application.

d) Cross-region performance

As the popularity and service reliability of the website continue to grow inland and offshore Australia, the need for catering the user experience of offshore customers is also required.

In the proposed architecture, the use of AWS CloudFront and AWS Amplify can significantly improve cross-region performance for our business requirements. The content delivery network that caches and delivers content from the closest location to the user by using CloudFront ensure minimized latency and faster content delivery for our users. AWS Amplify, furthermore, is integrated with AWS CloudFront, as the static assets and contents are automatically distributed by AWS CloudFront. Therefore, the use of these services will ensure that our web application can handle cross-region traffic without any issue, leading to greater user satisfaction, improved business outcomes and an "edge" advantage against our competitor.

e) Media Processing evolution and optimization

As the size of our website developed, so does its features to maintain a competitive advantage in the market as well as preserving and upholding the visitor retention rate. The proposed architecture aims to optimize the current features as well as revolutionize the procedure of which media are being processed.

Specifically, the fanout architecture is a distributed computing pattern that is built to efficiently process large volumes of data in parallel, allowing for media processing tasks to be distributed across multiple processing nodes. This will allow our architecture to scale up or down based on the current demand.

The proposed media processing procedure is a serverless/event-driven approach that utilized AWS Lambda functions to automatically process media files upon upload. This will ensure that media files are automatically processed without any manual intervention. This process can easily scale depend on the spike and decline of the time. Furthermore, the procedure is create decoupled to guarantee reusability and feature evolution. Meaning that, should new business requirements arise, new features and additional services can be added to this procedure without worrying about interfering with the existing functions.

The improvement is brought about using fanout architecture and new media processing procedure ensure significant improvement in scalability, reliability, and reusability to the web application.

5. Conclusion

In conclusion, a serverless/event-driven architecture on AWS can greatly benefit business that require scalable and cost-effective solution. AWS services such as AWS Lambda, Amazon SNS, and Amazon SQS can respond to real-time events without the need of manual management, allowing the business owner to pay more resources on bringing business value to the customers.

Reference

[1] "What is a Content Delivery Network (CDN)? – Amazon Web Services," Amazon Web Services, Inc., 2021. [Online]. Available: https://aws.amazon.com/what-is/cdn/ . [Accessed: April 9, 2023].

[2] A. Xu, "System Design Interview: An Insider's Guide," v. 1, System Design Interview: An Insider's Guide, Independently Published, 2020, ISBN: 9798664653403. [Online]. Available: https://books.google.com.vn/books?id=TZWmzQEACAAJ . [Accessed: April 9, 2023].

[3] "Introduction - Amazon CloudFront Developer Guide," Amazon Web Services, Inc. [Online]. Available: https://docs.aws.amazon.com/AmazonCloudFront/latest/DeveloperGuide/Introduction.html . [Accessed: April 9, 2023].

[4] "Serverless Logic Tier - Build a Serverless Web Application," Amazon Web Services, Inc. [Online]. Available: https://docs.aws.amazon.com/whitepapers/latest/serverless-multi-tier-architectures-api-gateway-lambda/serverless-logic-tier.html . [Accessed: April 9, 2023].

[5] "QSRsoft Launches Digital Huddle Board in 3 Months with AWS Serverless and Fire Devices," Amazon Web Services, Inc., October 22, 2020. [Online]. Available: https://aws.amazon.com/blogs/architecture/qsrsoft-launches-digital-huddle-board-in-3-months-with-aws-serverless-and-fire-devices/ . [Accessed: April 9, 2023].

[6] Y. Sharma, "Publish-Subscribe/Fan-out Pattern in Serverless Architectures using SNS, SQS and Lambda," Medium, June 22, 2018. [Online]. Available: https://medium.com/aws-lambda-serverless-developer-guide-with-hands/publish-subscribe-fan-out-pattern-in-serverless-architectures-using-sns-sqs-and-lambda-bccaa3abac9e . [Accessed: April 9, 2023].

[7] "Video on Demand - AWS Solutions," Amazon Web Services, Inc. [Online]. Available: https://www.amazonaws.cn/en/solutions/video-on-demand/ . [Accessed: April 9, 2023].

Appendix A: Media Processing Step Procedure

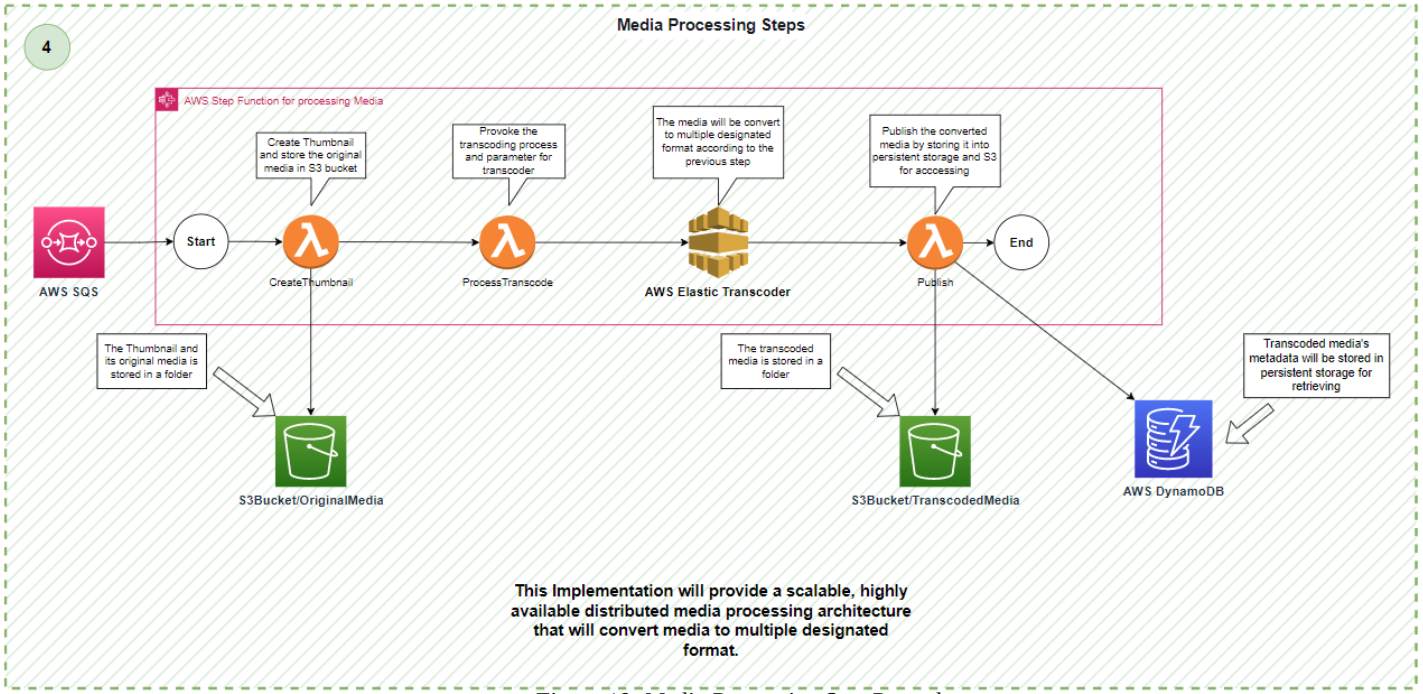

Figure 10: Media Processing Step Procedure

In this procedure, SQS queue will invoke a step function that will start by using AWS Lambda function to create thumbnail for the uploaded media and store it alongside with its original media into the original media folder in S3. The second step is performed by another Lambda function that prepares the media file for the transcoding process. This could involve checking the available formats that the website can transcode and generating metadata for the transcoded file. After that, Elastic Transcoder will transcode the media into designated format and finally send it to the publishing lambda function. Our process will end by storing the transcoded media into TranscodedMedia folder in S3 and store the metadata of the transcoded media into DynamoDB.

This processing procedure is highly reusable and flexible, allowing for the addition of new processing steps or the modification of existing ones to meet changing requirements. Suppose we want to implement AI detection into our processing Step, simply add another Lambda function into the existing step wherever needed.

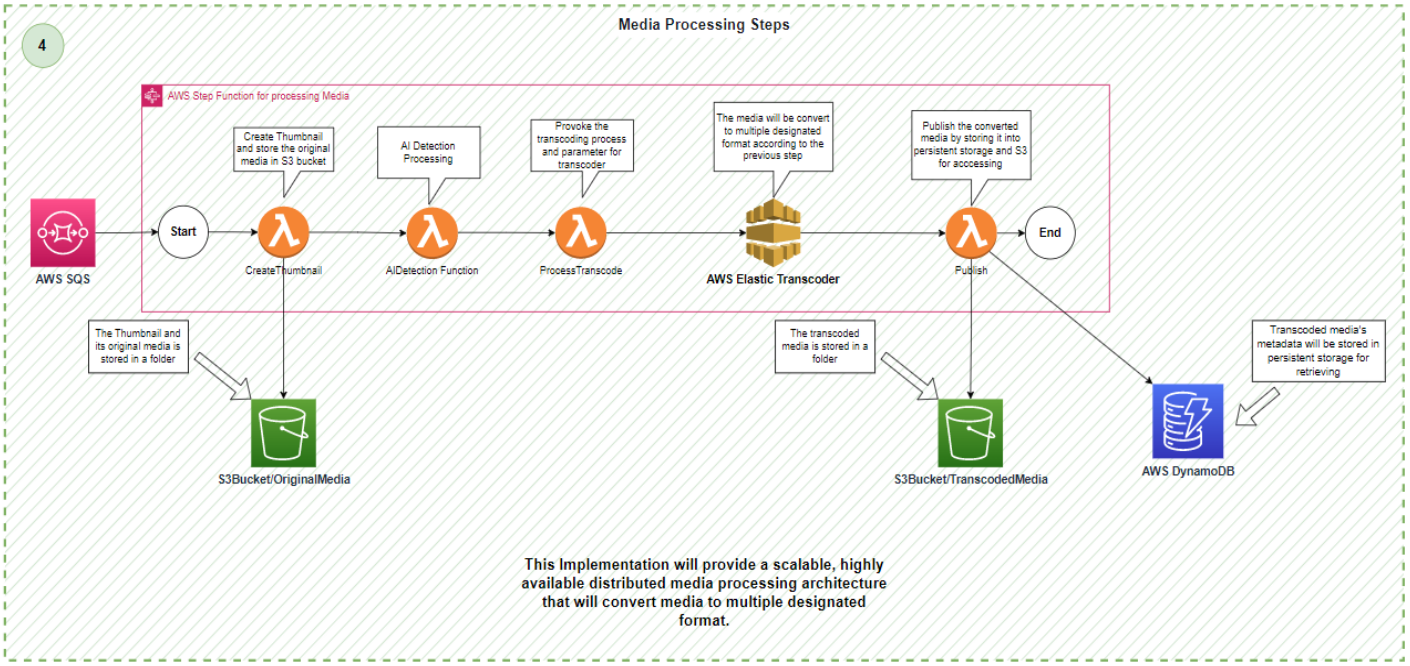

Figure 11: Alternative Media Processing Step Procedure

Additional services and steps may be needed for additional features; however, it can be added into this procedure with ease. Allowing our website reusability and flexibility with little overhead burden on existing featur

Appendix B: UML Diagrams for the function and features of the architecture

a. Viewing UML

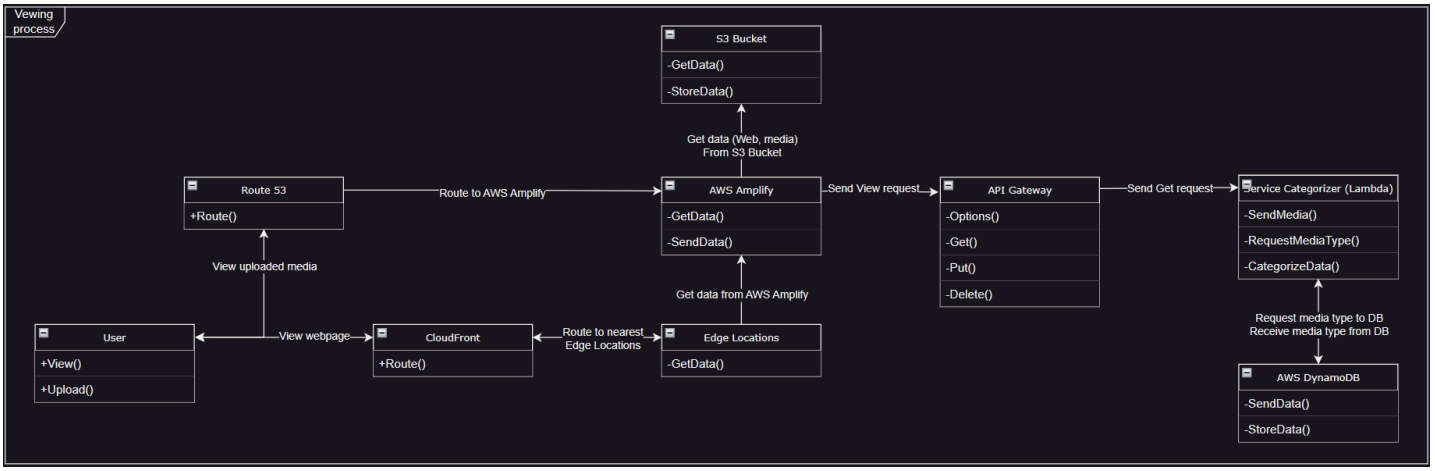

Figure 12: User view website

Users who browse the website will be routed to the nearest edge location via CloudFront, and AWS Amplify will display the website and some media stored in the bucket. The process differs a bit for viewing uploaded media as the user will be routed to CloudFront through Route 53 first. AWS Amplify will send a request to get the metadata for a specific media type to the API Gateway; the API then send a request to get the metadata requested from the database via the help of the Categorizer function.

b. Uploading UML

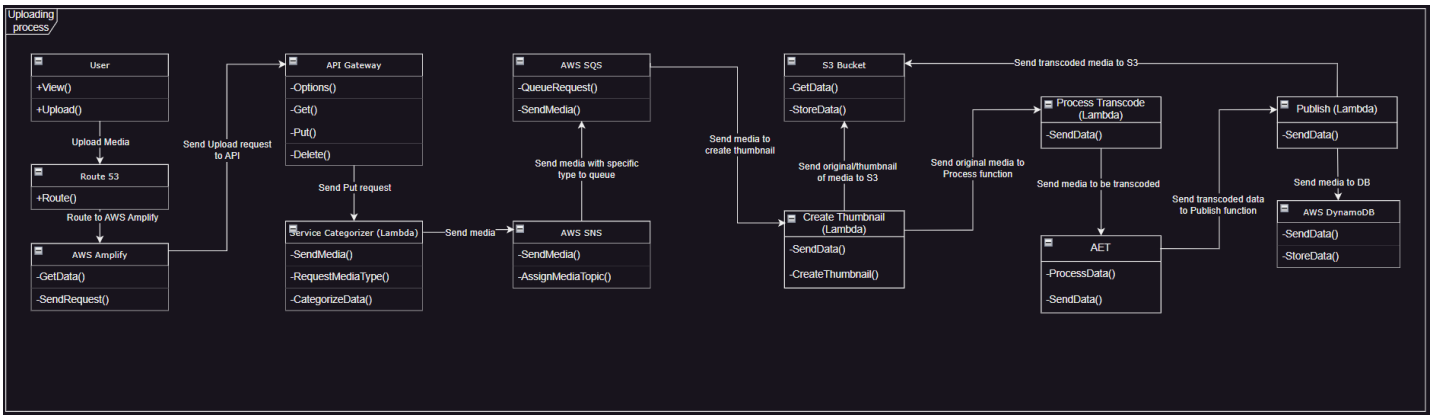

Figure 13: User Uploading file

When the user wants to upload a piece of media, the media will be sent through Route 53 and CloudFront to AWS Amplify, and then AWS Amplify issue an upload request to API, which will trigger the PUT method. The media coming out of the API will then be sent to the Service Categorizer function, where it will be assigned a topic based on its media type (i.e. image_topic, video_topic, etc.). The topic will let AWS SNS knows where the media should go, and then AWS SNS sends the media towards AWS SQS to queue for processing. From that point, the original media, as well as its thumbnail, will be uploaded on S3 because of the create thumbnail function, then the media is transcoded with AET, and the metadata will be produced; the transcoded media will then be sent to S3 again while its metadata will be stored in the DynamoDB database.

Appendix C: Initial version of the architecture

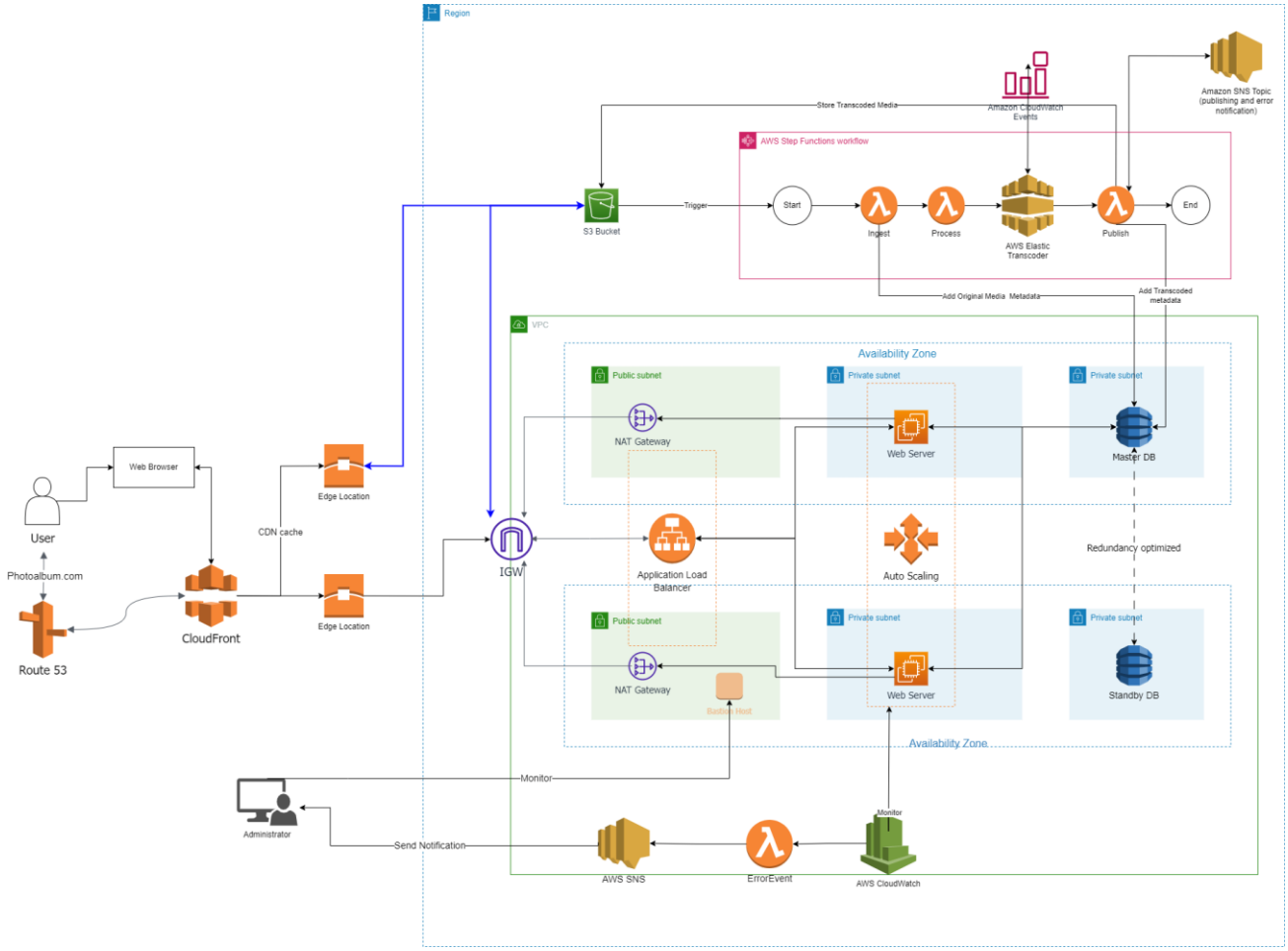

Figure 14: Traditional Web Architecture

Appendix D: Detailed cost of operation for 50GB of media upload

Services	Cost of each feature	Total
DynamoDB	• Write Request Unit (WRU): Each WRU = 1 write operation for 1 item with size up to 1kB. 50 x 1024^2 / 1000000 x $1.25 = **$65.536** • Read Request Unit (RRU): Each RRU = 1 read operation for 1 item with size up to 4kB. 50 x 1024^2 / 1000000 / 4 x $0.25 = **$3.2768** • Data Storage (per month): 25 x 0 + 25 x $0.25 = **$6.25** • Data Transfer IN: **$0** • Data Transfer OUT (per month): **$0**	$65.536 + $3.2768 + $6.25 = **$75.0628**
Route 53	Assume all the media uploaded are images and each image's size is 150kB. • Hosted zone (per month): **$0.50** • Standard Queries (per month): 50 x 1024^2 / 150 / 1000000 x $0.40 = **$0.14**	$0.50 + $0.14 = **$0.64**
CloudFront	Assume the price in Australia is the average price. • Data Transfer Out (per month): $0.114 x 50 = **$5.7** • Assume that the average size of images uploaded to the website is 150kB, and all the media uploaded are images. Assume that each image	Up to $5.7 + $1.31072= **$7.01072**

	will be associated with 3 HTTPS requests. The price for HTTPS requests will be up to: 50 x 1024^2 / 150 / 10000 x 3 x $0.0125 = **$1.31072**	
S3	• S3 Standard - Infrequent Access: $0.0125 x 50 = **$0.625** • Data Transfer IN: **$0** • Data Transfer OUT (per month): $0.09 x 50 = **$4.5**	$0.625 + $4.5 = **$5.125**
Amplify	It's free for the first 12 months, so the cost here is **$0**.	**$0**
API Gateway	Assume all the media uploaded are images and each image's size is 150kB. • API Calls (per month): Up to: 50 x 1024^2 / 150 / 1000000 x $3.50 = **$1.2233**	**$1.2233**
SNS	The endpoint is Simple Queue Service (SQS), so it is free for using SNS.	**$0**
SQS	Assume all the media uploaded are images and each image's size is 150kB. • FIFO Queues (per month): **$0** • Data Transfer IN: **$0** • Data Transfer OUT (per month): $0.09 x 50 = **$4.5**	**$4.5**
Elastic Transcoder	• Assume all of 50GB is used for uploading video, and all of them are 720p videos, their length will be about 200 minutes. The cost for Elastic Transcoder will be up to $0.03 x 200 = **$6**	Up to **$6**
Lambda	Assume all the media uploaded are images and each image's size is 150kB; assume that each image is associated with 4 requests. • Requests: 50 x 1024^2 / 150 / 1000000 x 4 x $0.20 = **$0.28** • Memory: **$5.5**	$0.28 + $5.5 = **$5.78**
CloudWatch	There are 8 Lambda function. Assume each of them is associated with 8 metrics. • Metrics (per month): 8 x 8 x 0.3$ = **$19.2**	**$19.2**
Total		**$124.54182**

Table 5: Cost of operation for 50GB

Appendix E: Detailed cost of operation for 100GB of media upload

Services	Cost of each feature	Total
DynamoDB	• Write Request Unit (WRU): Each WRU = 1 write operation for 1 item with size up to 1kB. 100 x 1024^2 / 1000000 x $1.25 = **$131.072** • Read Request Unit (RRU): Each RRU = 1 read operation for 1 item with size up to 4kB. 100 x 1024^2 / 1000000 / 4 x $0.25 = **6.5536** • Data Storage (per month): Up to now, the total of data is about: 50 x 6 + 100 = 400 GB. 25 x $0 + 375 x $0.25 = **$93.75** • Data Transfer IN: **$0** • Data Transfer OUT (per month): **$0**	$131.072 + $6.5536 + $93.75 = **$231.3756**
Route 53	Assume all the media uploaded are images and each image's size is 150kB. • Hosted zone (per month): **$0.50**	$0.50 + $0.28 = **$0.78**

	• Standard Queries (per month): 100 x 1024^2 / 150 / 1000000 x $0.40 = **$0.28**	
CloudFront	Assume the price in Australia is the average price. • Data Transfer Out (per month): $0.114 x 100 = **$11.4** • Assume that the average size of images uploaded to the website is 150kB, and all the media uploaded are images. Assume that each image will be associated with 3 HTTPS requests. The price for HTTPS requests will be up to: 100 x 1024^2 / 150 / 10000 x 3 x $0.0125 = **$2.62144**	Up to $11.4 + $2.62144 = **$14.02144**
S3	• S3 Standard - Infrequent Access: $0.0125 x (50 x 6 + 100) = **$5** • Data Transfer IN: **$0** • Data Transfer OUT (per month): $0.09 x 100 = **$9**	$5 + $9 = **$14**
Amplify	It's free for the first 12 months, so the cost here is **$0**.	**$0**
API Gateway	Assume all the media uploaded are images and each image's size is 150kB. • API Calls (per month): Up to: 100 x 1024^2 / 150 / 1000000 x $3.50 = **$2.4466**	**$2.4466**
SNS	The endpoint is Simple Queue Service (SQS), so it is free for using SNS.	**$0**
SQS	Assume all the media uploaded are images and each image's size is 150kB. • FIFO Queues (per month): **$0** • Data Transfer IN: **$0** • Data Transfer OUT (per month): $0.09 x 100 = **$9**	**$9**
Elastic Transcoder	• Assume all of 100GB is used for uploading video, and all of them are 720p videos, their length will be about 400 minutes. The cost for Elastic Transcoder will be up to $0.03 x 400 = **$12**	Up to **$12**
Lambda	Assume all the media uploaded are images and each image's size is 150kB; assume that each image is associated with 4 requests. • Requests: 100 x 1024^2 / 150 / 1000000 x 4 x $0.20 = **$0.56** • Memory: **$5.5**	$0.56 + $5.5 = **$6.06**
CloudWatch	There are 8 Lambda function. Assume each of them is associated with 8 metrics. • Metrics (per month): 8 x 8 x 0.3$ = **$19.2**	**$19.2**
Total		**$308.88364**

Table 6: Cost of operation for 100GB

Appendix F: Detailed cost of operation for 200GB of media upload

Services	Cost of each feature	Total
DynamoDB	• Write Request Unit (WRU): Each WRU = 1 write operation for 1 item with size up to 1kB. 200 x 1024^2 / 1000000 x $1.25 = **$262.144** • Read Request Unit (RRU): Each RRU = 1 read operation for 1 item with size up to 4kB. 200 x 1024^2 / 1000000 / 4 x $0.25 = **13.1072** • Data Storage (per month): Up to now, the total of data is about: 50 x 6 + 100 x 6 + 200 = 1100 GB. 25 x $0 + 1075 x $0.25 = **$268.75** • Data Transfer IN: **$0** • Data Transfer OUT (per month): **$0**	$262.144+ $13.1072 + $268.75 = **$544.0012**
Route 53	Assume all the media uploaded are images and each image's size is 150kB. • Hosted zone (per month): **$0.50** • Standard Queries (per month): 200 x 1024^2 / 150 / 1000000 x $0.40 = **$0.56**	$0.50 + $0.56 = **$1.06**
CloudFront	Assume the price in Australia is the average price. • Data Transfer Out (per month): $0.114 x 200 = **$22.8** • Assume that the average size of images uploaded to the website is 150kB, and all the media uploaded are images. Assume that each image will be associated with 3 HTTPS requests. The price for HTTPS requests will be up to: 200 x 1024^2 / 150 / 10000 x 3 x $0.0125 = **$5.24288**	Up to $22.8 + $5.24288 = **$28.04288**
S3	• S3 Standard - Infrequent Access: $0.0125 x (50 x 6 + 100 x 6 + 200) = **$13.75** • Data Transfer IN: **$0** • Data Transfer OUT (per month): $0.09 x 200 = **$18**	$13.75 + $18 = **$31.75**
Amplify	Assume web app size = 100MB, average size of page requested = 1.5 MB Assume that average build time = 3 minutes each day. Assume that daily active users = 10000. • Build & Deploy: 3 x 30 x 0.01 = **$0.9** • Data storage: 100 / 1024 x $0.023 = $0.00225 • Data Transfer Out: 10000 x (1.5 / 1024) x 30 x $0.15 = **$65.918**	$0.9 + $65.918 = **$66.818**
API Gateway	Assume all the media uploaded are images and each image's size is 150kB. • API Calls (per month): Up to: 200 x 1024^2 / 150 / 1000000 x $3.50 = **$4.8932**	**$4.8932**
SNS	The endpoint is Simple Queue Service (SQS), so it is free for using SNS.	**$0**
SQS	Assume all the media uploaded are images and each image's size is 150kB. • FIFO Queues (per month): **$0** • Data Transfer IN: **$0** • Data Transfer OUT (per month): $0.09 x 200 = **$18**	**$18**
Elastic Transcoder	• Assume all of 200GB is used for uploading video, and all of them are 720p videos, their length will be about 800 minutes. The cost for	Up to **$24**

	Elastic Transcoder will be up to $0.03 x 800 = **$24**	
Lambda	Assume all the media uploaded are images and each image's size is 150kB; assume that each image is associated with 4 requests. • Requests: 200 x 1024^2 / 150 / 1000000 x 4 x $0.20 = **$1.12** • Memory: **$5.5**	$1.12 + $5.5 = **$6.62**
CloudWatch	There are 8 Lambda function. Assume each of them is associated with 8 metrics. • Metrics (per month): 8 x 8 x 0.3$ = **$19.2**	**$19.2**
Total		**$744.38528**

Table 7: Cost of operation for 200GB