DALHOUSIE
UNIVERSITY

CSCI 5409 – Cloud Computing

Final Report

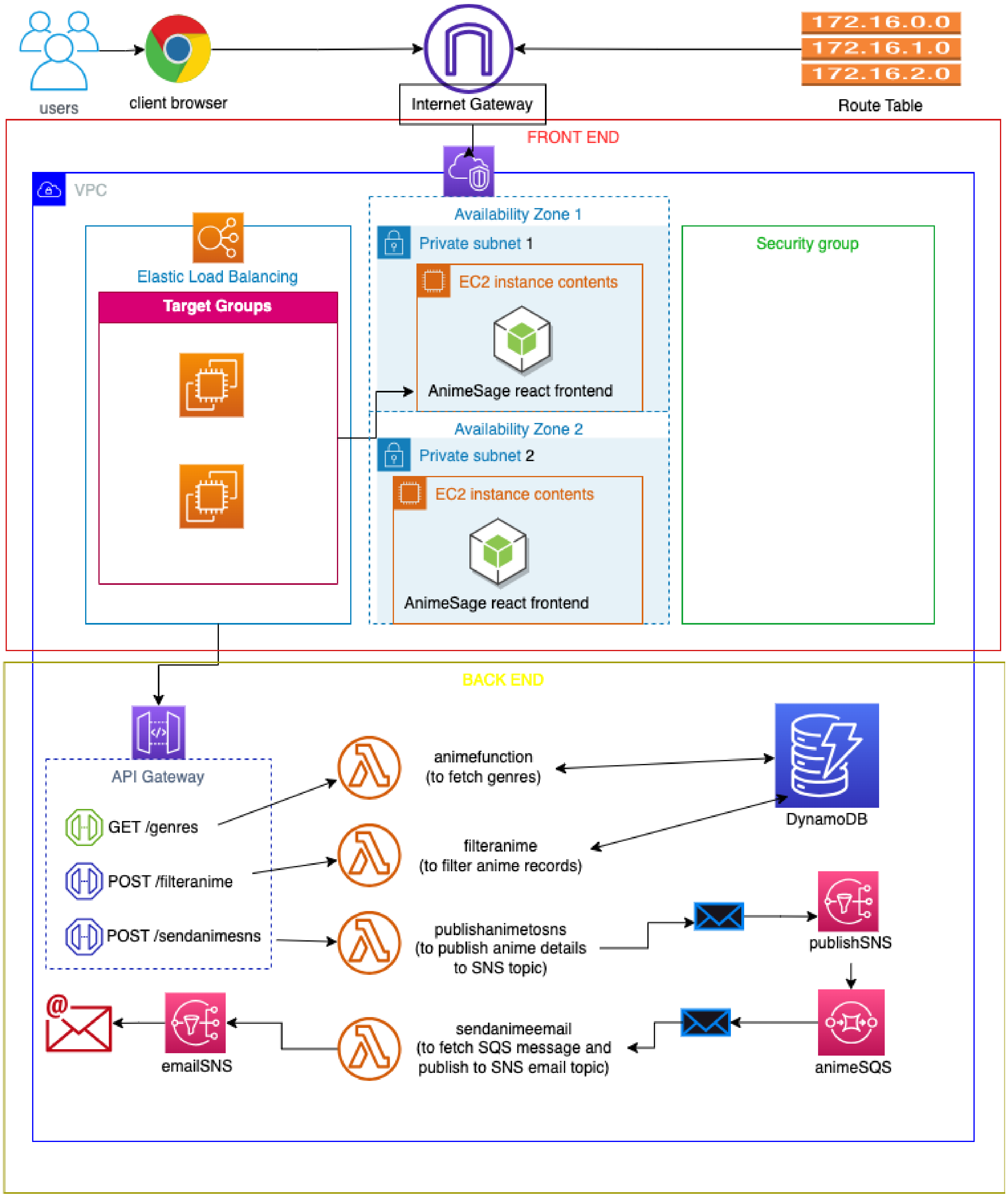
Tapan Patel
B00913706
tapan.patel@dal.ca

Project Description

Application Name: **AnimeSage**

AnimeSage is a user-friendly Anime recommendation app designed to help anime enthusiasts discover new anime titles that match their interests and preferences. You can access AnimeSage through a web browser or a mobile device. When you visit AnimeSage, you are presented with a simple and intuitive form. This form allows you to input your preferences for anime selection, such as Aired date, episodes, popularity, score, ranked, and genres. It offers a wide range of genres to choose from, making it easier for you to select your preferred genres and discover anime within those categories. The list of genres is dynamically populated from the database, ensuring that you have access to the most up-to-date genres. After submitting your preferences through the form, AnimeSage leverages its powerful backend algorithms to filter through the vast database of anime titles. It then presents you with a list of personalized anime recommendations that match your selected criteria. The recommended anime titles are displayed in a visually appealing manner, using cards and grids. Each anime card provides essential details such as the title, synopsis, score, and more. This presentation style allows you to quickly scan through the recommendations and find anime that pique your interest. If you come across an anime that you'd like to know more about or want to save for later, AnimeSage offers a convenient "send details via email" button on each anime card. When you click this button, you will receive an email containing all the relevant details of the selected anime. Behind the scenes, AnimeSage utilizes AWS services like Lambda functions, SNS, and SQS to process your preferences and deliver the anime recommendations efficiently. This ensures a seamless user experience without any delays or interruptions. Since AnimeSage uses DynamoDB in On-Demand mode, you don't have to worry about managing database capacity. The app automatically scales to handle varying levels of traffic, providing a smooth experience for all users. With its user-friendly interface, personalized recommendations, and the ability to receive anime details via email, AnimeSage makes the process of discovering and keeping track of anime a delightful experience for every user. Whether you are a seasoned anime enthusiast or a newcomer to the anime world, AnimeSage will assist you in finding your next favorite anime series.

Project Structure:



AWS Services used:

Based on the menu items provided, I selected the following services to meet the requirements for building the AnimeSage app:

Compute:

AWS Lambda: Lambda is ideal for handling API requests, data processing, and any other stateless functions, all without the need to manage servers. It fits well with the serverless architecture of AnimeSage and helps keep the application lightweight and cost-effective.

AWS EC2: While Lambda is great for serverless functions, for hosting the frontend React app, a virtual machine through EC2 provides more control and flexibility in managing the web application.

Compute Alternatives :

AWS Elastic Beanstalk: This service is designed for automatically running, load balancing, and scaling web apps via virtual machines. While it can be convenient, I prefer to have more control over my application's infrastructure, which is why I chose to use AWS EC2 to directly host my web app.

AWS Elastic Container Service & Elastic Container Registry (ECS): ECS is a great choice for running Docker containers, but for the simplicity and lightweight nature of AnimeSage, AWS Lambda serves as a more suitable option.

Storage:

AWS DynamoDB: As a NoSQL database, DynamoDB can store and query anime details efficiently, making it well-suited for AnimeSage's data requirements.

Storage Alternatives :

AWS S3: S3 is a simple file storage service, but for the AnimeSage app, we need a database to store and query anime details efficiently. Hence, AWS DynamoDB, a NoSQL database, is a better choice due to its ability to handle structured data with dynamic attributes like the anime details.

AWS Relational Database Service:

AWS RDS provides managed relational databases like MySQL, PostgreSQL, etc., which might be useful for certain applications but not for AnimeSage.

AnimeSage requires NoSQL storage, making DynamoDB a more suitable choice.

Network:

AWS VPC: Creating a private network with VPC ensures secure communication between the different services and enhances overall application security.

Network Alternatives:
Amazon CloudFront: While CloudFront is a content delivery network, it is not directly relevant to the core functionalities of AnimeSage, which primarily involves processing user preferences and displaying recommendations. Hence, I opted for AWS Virtual Private Cloud (VPC) to create a private network for secure communication between services.

General:

Amazon SNS and AWS SQS: These services provide reliable message delivery and help in implementing a producer/consumer pattern. They are useful for sending anime details via email, ensuring the app functions smoothly and efficiently. SNS helps in sending notifications, while SQS allows for decoupling and scaling.

AWS Elastic Load Balancing (ELB): ELB distributes incoming traffic across multiple EC2 instances, ensuring better fault tolerance and high availability. It helps in evenly distributing the load and scaling the application as traffic demands fluctuate.

In summary, the chosen services in the system were selected based on their relevance to AnimeSage's core functionalities, simplicity, and ability to meet the application's requirements effectively. These services offer a scalable, secure, and cost-efficient infrastructure to support the app's anime recommendation process and user interactions.

Deployment model used:

For the AnimeSage system, I chose the public cloud deployment model. This means that the entire application infrastructure, including servers, databases, and services, is hosted and managed by a third-party cloud provider, Amazon Web Services (AWS).

Scalability and Elasticity: Public cloud providers like AWS offer virtually unlimited resources and automatic scaling capabilities. As AnimeSage may experience varying levels of traffic and data storage needs, the public cloud allows us to scale up or down as required without upfront hardware investments or capacity planning.
Cost-Effectiveness: The pay-as-you-go model of public cloud services allows us to only pay for the resources we consume. This eliminates the need for large upfront costs, making it a cost-effective solution for a startup project like AnimeSage.
Global Accessibility: The public cloud provides a distributed infrastructure with data centers located in different regions worldwide. This ensures that AnimeSage can be accessed quickly and efficiently by users from various geographic locations.
Security and Compliance: Reputable public cloud providers invest heavily in security and compliance measures. They adhere to industry standards and regulations, helping to ensure data protection and maintaining a high level of security for the application and its users.
Managed Services: Public cloud providers offer a wide range of managed services, such as AWS Lambda, DynamoDB, SNS, and SQS, which simplify the development and deployment process. Utilizing these services allows us to focus on building the application's core features rather than managing infrastructure.

Delivery model used:

For my AnimeSage system, I have chosen a combination of Infrastructure as a Service (IaaS) and Function as a Service (FaaS) delivery models.
1. IaaS (Infrastructure as a Service):I chose IaaS for hosting the frontend of the AnimeSage web application. Hosting the frontend on an EC2 instance allowed me to have more control over the environment and customize the virtual machine to meet specific requirements. With IaaS, I can install custom software and frameworks on the virtual machine as needed, giving me greater flexibility in the frontend setup. I can also install my company's security software for better theft protection.
2. FaaS (Function as a Service):
In the AnimeSage project, I used Function as a Service (FaaS) with AWS Lambda at its core. AWS Lambda handles various tasks, such as filtering anime records and processing user preferences. The application is further enhanced with AWS API Gateway, which provides a RESTful API for interacting with the backend Lambda functions securely. Asynchronous messaging is achieved using Amazon SNS and Amazon SQS, allowing decoupling of components and enabling efficient processing of tasks like sending email notifications. The data storage is managed by AWS DynamoDB, a fully managed NoSQL database that ensures seamless scaling and fast querying of anime details. This serverless setup using FAAS optimizes performance, cost-efficiency, and scalability while

freeing developers from server management concerns, facilitating the delivery of personalized anime recommendations to users.

By combining IaaS and FaaS, I get the best of both worlds. IaaS provides me with control and customization options for the frontend, while FaaS offers cost-effective and scalable backend processing. This hybrid approach allowed me to optimize my system based on the specific requirements of each component.

Architecture used:

In the final AnimeSage system architecture, I have implemented a workload distribution architecture. This architecture aims to distribute the workload efficiently across various components of the application to optimize resource utilization and improve performance. Here's how it aligns with AnimeSage:
By leveraging the cloud's on-demand usage, AnimeSage can scale resources up or down based on demand. This ensures that we only pay for what we use and avoid wasting resources, making the application more cost-effective.
The workload distribution architecture fits well with AnimeSage's needs as a recommendation system handling varying levels of user requests. By utilizing AWS Elastic Load Balancer (ELB) in front of the EC2 instance hosting the frontend, we achieve balanced web servers, ensuring fair distribution of user traffic and reducing the risk of server overloading.
Furthermore, AWS Lambda, as a serverless component, efficiently handles backend processing, allowing automatic scaling based on demand. Combined with DynamoDB's seamless scaling, we ensure the backend can handle fluctuating workloads effectively.

As a result, the workload distribution architecture optimizes resource usage, provides high availability, and delivers efficient performance. This architecture choice is wise for AnimeSage, as it aligns with the goal of providing personalized anime recommendations while minimizing infrastructure management complexities.

Security Analysis:

In developing the AnimeSage project, security has been a paramount consideration at every stage of the system, with a focus on securing data both in transit and at rest.
Securing Data in Transit:
To ensure data security during transmission, AnimeSage implements secure communication protocols:
HTTPS (SSL/TLS): All data exchanged between the frontend and backend, including user preferences and requests, is transmitted over HTTPS using SSL/TLS encryption. This

encryption protects sensitive data from unauthorized access and eavesdropping during transit.
AWS API Gateway Security: API Gateway, acting as the frontend for the Lambda functions, ensures secure communication between clients and the serverless backend.
Securing Data at Rest:
AWS DynamoDB Encryption at Rest: Data stored in DynamoDB is automatically encrypted at rest using AWS managed encryption keys (SSE-S3). This encryption provides an additional layer of security, protecting the data from unauthorized access in the event of physical theft or unauthorized access to the storage media.
Server-Side Data Encryption: Sensitive data, such as user preferences and email content, is handled with appropriate encryption techniques before storage in DynamoDB or when sent through messaging services like SNS and SQS. This ensures that data remains confidential even if the storage infrastructure is compromised.
AWS Lambda Environment Security: The Lambda functions themselves are executed within a secure environment provided by AWS. This environment includes network isolation and protection mechanisms to prevent unauthorized access to Lambda's resources and data.
Secure IAM Policies: The Identity and Access Management (IAM) policies for AWS resources, including Lambda functions, API Gateway, DynamoDB, SNS, and SQS, are carefully configured to grant the minimum required privileges. This practice follows the principle of least privilege, reducing the attack surface and protecting against potential misuse of permissions.

Overall, AnimeSage's comprehensive approach to security ensures that user data is protected at all stages of the system. Encryption at rest and in transit, combined with secure IAM policies, help guard against potential security breaches and unauthorized access. By following industry best practices for data security in the cloud environment, AnimeSage provides users with a safe and trustworthy platform to explore and enjoy personalized anime recommendations.

Cost Analysis:

In the real-world deployment of the AnimeSage system, several cost metrics need to be considered, including upfront costs and ongoing operational costs.
1. Upfront Costs:
The upfront costs for AnimeSage will include expenses related to setting up the initial infrastructure, including AWS resources such as EC2 instances, API Gateway, DynamoDB, SNS, and SQS. Additionally, there might be costs for domain registration and SSL certificate purchase for securing HTTPS communication.

The frontend hosting on an EC2 instance may require a one-time setup cost for customizing and configuring the virtual machine, including installing necessary software and frameworks.

2. Ongoing Operational Costs:
The majority of AnimeSage's operational costs will be related to the usage of AWS services, primarily Lambda, DynamoDB, SNS, and SQS. These services follow a pay-as-you-go model, charging based on actual usage and request volumes.
DynamoDB costs depend on the provisioned read and write capacity units, as well as the amount of data stored. Utilizing DynamoDB on-demand mode leads to a cost-effective approach, as it eliminates the need to provision capacity in advance.
Lambda charges are based on the number of requests and the compute time consumed. Optimal function design and efficient code execution can help minimize Lambda costs.

3. Additional Costs:
Additional costs may arise due to data transfer charges, particularly if AnimeSage involves heavy data transfers between services or external endpoints. S3 storage costs might be incurred if AnimeSage utilizes S3 to store static assets for the frontend, although these costs are generally low.

Alternative Approaches for Cost Savings:
One approach to potentially save costs is by using AWS Elastic Beanstalk for my frontend. It can potentially save costs compared to hosting it on an EC2 instance.
Leveraging AWS Cost Explorer and Trusted Advisor can provide insights into cost optimization opportunities, identifying unused or underutilized resources that can be scaled down or terminated.
Optimizing Lambda function code for faster execution and minimizing unnecessary invocations can help reduce Lambda costs further.

While certain alternatives might save costs, the hybrid architecture with an EC2 instance and serverless backend components is justified by specific needs, such as server-side rendering and dynamic content processing in the frontend. Additionally, using serverless components like Lambda, DynamoDB, SNS, and SQS enhances scalability, reduces operational overhead, and enables efficient event-driven processing, all contributing to better user experiences and faster development times.

Gitlab repo : https://git.cs.dal.ca/courses/2023-summer/csci4145-5409/tapan/-/tree/main/TermAssignment

References

[1] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html. [Accessed: 31-Jul-2023].

[2] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-network-iface-embedded.html. [Accessed: 31-Jul-2023].

[3] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance-launchtemplatespecification.html. [Accessed: 31-Jul-2023].

[4] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/quickref-ec2.html. [Accessed: 31-Jul-2023].

[5] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-apigateway-method.html. [Accessed: 31-Jul-2023].

[6] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-function.html. [Accessed: 31-Jul-2023].

[7] P. Mescalchin, *CloudFormation API Gateway endpoint calling a Lambda function using proxy integration example*. .

[8] "Nickolas Kraus," *Nickolaskraus.io*. [Online]. Available: https://nickolaskraus.io/articles/creating-an-amazon-api-gateway-with-a-lambda-integration-using-cloudformation/. [Accessed: 31-Jul-2023].

[9] L. Hannigan, "Using CloudFormation to import your DynamoDB table from S3," *Medium*, 30-Aug-2022. [Online]. Available: https://medium.com/@leeroy.hannigan/using-cloudformation-to-import-your-dynamodb-table-68f8aefeabbf. [Accessed: 31-Jul-2023].

[10] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-dynamodb-table.html. [Accessed: 31-Jul-2023].

[11] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/S3DataImport.HowItWorks.html. [Accessed: 31-Jul-2023].

[12] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-dynamodb-table-keyschema.html. [Accessed: 31-Jul-2023].

[13] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-dynamodb-table-importsourcespecification.html. [Accessed: 31-Jul-2023].

[14] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-sns-topic.html. [Accessed: 31-Jul-2023].

[15] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-sns-subscription.html. [Accessed: 31-Jul-2023].

[16] Preeti, "How to use CloudFormation to Create SNS topic and Subscription," *CloudKatha*, 16-Aug-2021. [Online]. Available: https://cloudkatha.com/how-to-use-cloudformation-to-create-sns-topic-and-subscription/. [Accessed: 31-Jul-2023].

[17] A. W. S. Official, “I want to create a subscription between my Amazon Simple Queue Service (Amazon SQS) queue and Amazon Simple Notification Service (Amazon SNS) topic in AWS CloudFormation,” *Amazon Web Services, Inc,* 24-Aug-2016. [Online]. Available: https://repost.aws/knowledge-center/sqs-sns-subscribe-cloudformation. [Accessed: 31-Jul-2023].

[18] L. Musebrink, “SQS Queue as Lambda Trigger in AWS CloudFormation,” *Itonaut.com,* 11-Jul-2018. [Online]. Available: https://www.itonaut.com/2018/07/11/sqs-queue-as-lambda-trigger-in-aws-cloudformation/. [Accessed: 31-Jul-2023].

[19] A. R. Verma, “CloudFormation template for SQS with lambda triggers,” *Towards Data Engineering,* 11-Mar-2021. [Online]. Available: https://medium.com/towards-data-engineering/sqs-lambda-triggers-cloudformation-template-900999a01de5. [Accessed: 31-Jul-2023].

[20] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-eventsourcemapping.html. [Accessed: 31-Jul-2023].

[21] “Create an application load balancer using CloudFormation—ArcGIS Enterprise in the cloud,” *Arcgis.com*. [Online]. Available: https://enterprise.arcgis.com/en/server/latest/cloud/amazon/cf-create-application-elb.htm. [Accessed: 31-Jul-2023].

[22] *Githubusercontent.com*. [Online]. Available: https://raw.githubusercontent.com/awslabs/cloudformation-ldaps-nlb-template/master/NLB_SimpleAD.template. [Accessed: 31-Jul-2023].

[23] A. Mahapatra, “AWS Classic Load Balancer through Cloudformation. Everything you need,” *Medium,* 21-Feb-2018. [Online]. Available: https://anupam-ncsu.medium.com/aws-classic-load-balancer-through-cloudformation-everything-you-need-e9ef6f2cd19a. [Accessed: 31-Jul-2023].

[24] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-lambda-permission.html. [Accessed: 31-Jul-2023].

[25] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-lambda-function-code.html. [Accessed: 31-Jul-2023].

[26] *Amazon.com*. [Online]. Available: https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/successfully-import-an-s3-bucket-as-an-aws-cloudformation-stack.html. [Accessed: 31-Jul-2023].