CSCI 5409: Advanced topics in Cloud Computing

Term Assignment: OceanSpy

Priya Mandyal (B00974718)

pr335751@dal.ca

Repo: https://github.com/priya-mandyal/Ocean-Pollution-Detector

Table of Contents

Table of Figures

Project Overview

Introduction

We all have heard of "User paper straws, Save the turtles". Well, straws are not the only things that affect the marine species. Over 460 million tons of plastic are produced every year and at least 20 million tons of plastic end up in the ocean every year [11]. Marine species ingest or get entangled in the plastic leading to injuries and death. The Ellen McArthur Foundation has predicted that **by 2050, oceans will have more plastic than fish** [12]**.** It is high time to take some action towards it. This project is an effort of trying to contribution to marine species protection.

OceanSpy

OceanSpy is a web application that aims to identify plastic pollution and notifies the user about it. Detection is a step that comes before cleaning. OceanSpy helps exactly with that. It helps in detecting pollution so the area can get cleaned.

Users:

This web application has many users.

- Volunteers/Ngo can subscribe to the web application to get notified about pollution and help with cleanups.
- Government agencies can view pollution data to make informed decisions about policies and resources.
- Environmental scientists can study pollution patterns and their effects on marine life.
- General public can stay informed about local pollution through OceanSpy and participate in cleanups.

Features:

Image Upload: Users can upload images of the ocean through the web application.

Image Storage: Uploaded images are stored in AWS S3.

Pollution Detection: Images are processed to detect pollution using AWS Rekognition's custom trained model.

Notification: Users receive notifications via AWS SNS if pollution is detected.

CICD: I have implemented complete CICD workflow i.e. every push to GitHub repo (https://github.com/priya-mandyal/Ocean-Pollution-Detector) will start a GitHub workflow i.e. builds docker image, pushes docker image to ECR, start a new deployment in ECS.

How It Works

Image Upload:

Users go to the application and upload an image using interface to submit them. The image upload request is routed from React through AWS API Gateway to an AWS Lambda function which puts the images in S3 bucket. The Lambda function decodes the base64 the image and puts it in the AWS S3 bucket.

Image Processing:

The put operation in S3 bucket triggers another Lambda function that processes the image through AWS Rekognition. The confidence threshold is set as 80 so if any new image is detected as pollution with confidence equal to or greater than 80, it is labelled as Pollution.

Image Analysis:

AWS Rekognition does the image analysis and identifies any plastic-related pollution. The model in AWS Rekognition is trained using Kaggle dataset [10].

Notification:

If pollution is detected i.e. confidence >= 80, the system triggers an AWS SNS notification to inform the subscribed user. For subscription, user can subscribe from the frontend. On submission of email, React Application calls AWS lambda function via AWS API Gateway to subscribe to AWS SNS.

How to Test It

Step 1. Go to the application. In the home page you will see “Upload Image” and “Subscribe” buttons (There is “Live Detection”, but it is not implemented, I will continue my work on this as a personal project, not a part of term project).

Step 2. Click on subscribe button, put your email ID and save.

Step 3. You will get an email from AWS to subscribe. Go to your account and confirm the subscription.

Step 4. Click on image upload and submit.

If the uploaded image has pollution, you will get an email, otherwise no image.

The CI/CD pipeline is facilitated by a GitHub workflow YAML file that creates Docker images and pushes them to ECR. The image upload then triggers a task in ECS, ensuring seamless and automated deployment.

Meeting Menu Item Requirements

Services Used:

Category	Service
Compute	AWS ECR and AWS ECS, AWS Lambda
Storage	AWS S3
Network	AWS API Gateway
General	AWS SNS, Amazon Rekognition

Compute Services

Selected Service: AWS Elastic Container Service (ECS) & Elastic Container Registry (ECR)

AWS ECS is the AWS Docker container service that handles the orchestration and provisioning of Docker containers and AWS ECR is a managed Docker container registry that simplifies storing, managing, and deploying Docker container images [1] [3].

Comparison with Alternatives:

- **AWS EC2:** AWS EC2 requires manual setup where there is need to manually launch application, manage the instance, do load balancing. Whereas ECS automates these tasks. User only needs to provide the details like machine type, memory, network details and the instances are managed as containerized applications.
- **AWS Elastic :** AWS Beanstalk simplifies deployment and helps scaling application. ECS also helps in easy deployment and scaling management. I wanted to use CICD for my application which is why I went with ECS as ECS with ECR makes CICD straightforward and seamless [2].

For **ECS,** I have used AWS FARGATE as launch instance type instead of EC2 because of its simplicity as there is no need to manage infrastructure and instances.

Reason for Choosing:
ECS and ECR were the best option for deployment of containerized application. ECR offers image versioning, image scanning for vulnerabilities which provides additional layer of security. Also, ECR has seamless integration with ECS which is a bonus. ECS with ECR provides integrated environment for deploying containers with manages scaling, infrastructure. This setup allows for efficient scaling and management of containers. Also, ECS and ECR were chosen because I wanted to enable Continuous Integration and Continuous Deployment (CI/CD).

Selected Service: AWS Lambda

AWS Lambda is a serverless event-driven services which runs code without provisioning or managing servers [7]. It automatically scales and manages the execution environment.

Comparison with Alternatives:

- **AWS Step Functions**: Step Functions are used for complex workflows but are more suited for multiple state or executions. Considering the use cases of OceanSpy, Lambda is sufficient for the simple processes like Image Upload, SNS Subscription.

Reason for Choosing: Lambda was selected for its ability to run code in response to events (e.g., image uploads) without managing servers. This fits well with the event-driven nature of the Ocean Pollution Detector's workflow.

Storage

Selected Service: AWS S3

AWS S3 is a scalable object storage service used for storing media files like image, videos (specific to OceanSpy's use case) and retrieving any amount of data [6]. It is highly durable and available.

Comparison with Alternatives:

- **AWS DynamoDB**: DynamoDB is a NoSQL database suitable for structured data, but S3 is better unstructured data such as images, videos (specific to OceanSpy's use case).
- **AWS AppSync**: AppSync is used for GraphQL data synchronization, which is not required for OceanSpy's image storage.

Reason for Choosing: I selected S3 for its cost-effectiveness, scalability. Also, as per OceanSpy's need for storing large amounts of unstructured data such as images, videos, S3 is the best option.

Network

Selected Service: API Gateway

AWS API Gateway is a fully managed service designed for creating, publishing, maintaining, and securing APIs. It provides features such as API routing, request transformation, security, and monitoring.

Comparison with Alternatives:

- **AWS VPC (Virtual Private Cloud):** VPC provides network isolation and control over the networking environment but not handle API requests directly.
- **Amazon EventBridge:** Event bridge routes events but no routing of API requests.

Reason for Choosing AWS API Gateway:

OceanSpy uses AWS Lambda and to expose the lambda so that they can be triggered API Gateway is one of the best way. AWS API Gateway exposes Lambda to an API that anyone

can call. Apart from this API Gateway because of its security features like AWS IAM, API keys, and integration with AWS WAF for protecting against common web exploits [4]. As per OceanSpy's need to call Lambda from React application API gateway is the best fit.

General Services

Selected Service: AWS SNS

AWS SNS is a fully managed messaging service for sending notifications to users via SMS, email, or other protocols [8].

Comparison with Alternatives:

- **AWS SQS**: SQS is a message queue service requires poll i.e. messages sits in queue unless they get picked up. As per OceanSpy's use case, subscribed users should get notifications (push-based) thus SNS is better.
- **AWS SQS**: SES is use for email communication, sending transactional emails. As per OceanSpy's use case, service should support push thus SNS is better.

Reason for Choosing: SNS was selected for its capability to send notifications to users when pollution is detected in the uploaded images, fulfilling the OceanSpy's notification requirements.

Selected Service: Amazon Rekognition

Amazon Rekognition provides image and video analysis capabilities, including object detection, image moderation, and facial analysis. Although, Rekognition has its own models like PPE detection etc. I have trained a custom model using Marine Plastic pollution dataset.

Comparison with Alternatives:

- **AWS DeepLens**: DeepLens is a deep learning enabled camera that performs image/video analysis on edge devices. It is more complex than Rekognition and also needs hardware setup.

Reason for Choosing: I selected Rekognition because of its ability to analyze images/videos. Rekognition also allows custom training of models which I used to train a new model for ocean pollution detection. Because of its flexibility in training, easy use and access, Rekognition is a good fit for OceanSpy's image processing needs.

Deployment Model: Public Cloud

OceanSpy uses a public cloud deployment model, I chose it because of its scalability and cost efficiency. In this deployment model, all are hosted on AWS's cloud infrastructure, which is shared among multiple customers and are managed by the service provider.

Public Cloud

In a public cloud deployment, resources such as compute, storage, and networking are provided and managed by a third-party cloud service provider (in my case, AWS). Also, are shared among multiple users.

AWS Services used for Public Deployment Model:

- **AWS Lambda**: For serverless computation, handling image processing and pollution detection tasks without the need for dedicated servers.
- **AWS ECS (Elastic Container Service)** and **ECR (Elastic Container Registry)**: For managing and orchestrating containerized applications.
- **AWS S3**: For scalable and durable object storage of user-uploaded images.
- **AWS API Gateway**: For managing and routing API requests to backend services.
- **AWS SNS**: For sending notifications to users when pollution is detected.
- **Amazon Rekognition**: For automated image analysis to detect pollution in ocean images.

Benefits of Public Cloud Deployment Model

- **Scalability**: The public cloud offers unlimited scalability(virtually). AWS services like Lambda and ECS can automatically scale based on demand, ensuring that the system can handle varying loads of image uploads and processing without manual intervention.
- **Cost Efficiency**: With a public cloud, we only pay for what you use and there is also no infrastructure setup thus making it pocket friendly.
- **High Availability and Reliability**: AWS's public cloud infrastructure is designed for high availability (99.999%) and reliability.
- **Security**: Although public cloud environments are shared, AWS provides robust security features to protect data and applications using features like IAM, encryption etc.
- **Accessibility**: Globally accessible.
- **Managed Services**: Public cloud providers like AWS offer managed services like Lambda and API Gateway reducing the complexity of system management.

In conclusion, the public cloud deployment model is well-suited for the OceanSpy project due to its scalability, cost efficiency, flexibility, reliability and global reach. This model supports OceanSpy's requirements of handling large volumes of data, performing complex image processing, and providing global access and notifications.

Delivery Model

OceanSpy uses: PaaS and FaaS

OceanSpy is: SaaS

I chose both PaaS and FaaS to streamline deployment and management. Also, FaaS is very cost effective.

OceanSpy uses both Platform as a Service and Function as a service to deliver its functions. I used PaaS by leveraging ECS. This approach facilitates the deployment and management of the application, handling container orchestration and scaling automatically. This approach makes sure that OceanSpy's application remains highly available and high performing without the need to manage the underlying infrastructure or deal with scaling issues.

OceanSpy also uses AWS Lambda for its backend functionality leveraging the FaaS model to handle event-driven tasks efficiently. With Lambda, there's no need for server management. The code executes backend functions such as image processing and pollution detection in response to triggers such as API requests or S3 data uploads, and the service scales automatically based on demand. This model helps control costs by charging only for the compute time used making it pocket friendly.

By integrating both PaaS and FaaS, OceanSpy manages both the static and dynamic components of the application, resulting in a smooth and responsive user experience.

Architecture diagram

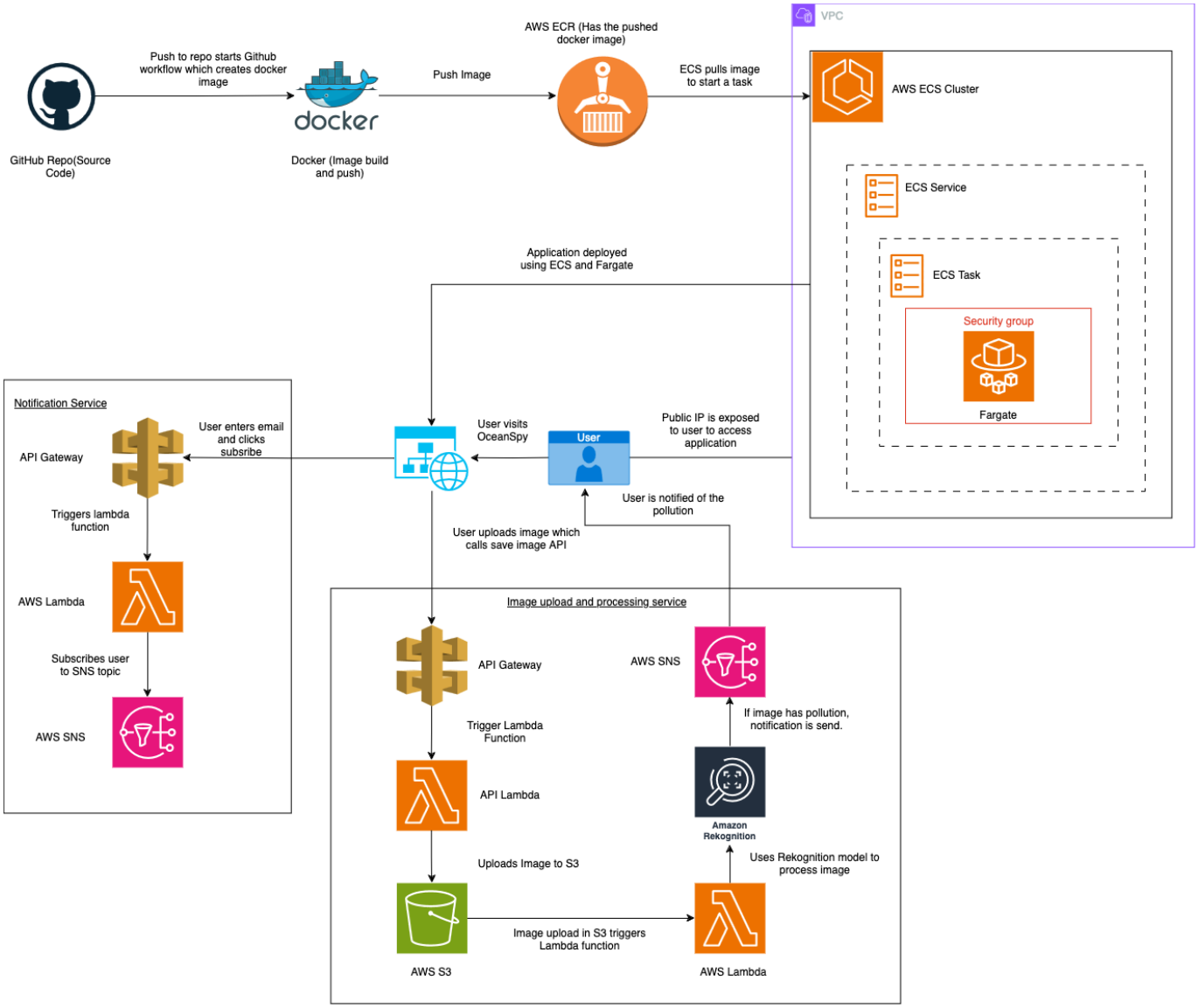

Figure 1 OceanSpy architecture diagram [9]

The architecture diagram shows multiple parts like CICD Pipeline, SNS Subscription pipeline, Image Upload and Processing pipeline. For the application, I have created a React based application (HTML, CSS, JavaScript, React) and for backend I have used AWS services as backend. I have chosen React as it offers a reactive and interactive user experience.

User interface created using React. AWS Lambda as backend as a service.

CICD pipeline with ECS and ECR

For the CICD, I have used AWS services i.e. ECS and ECR. Apart from that I have used GitHub workflow, Docker to automate the deployment.

- Every code commit or push to the repo initiates a GitHub workflow. The GitHub workflow builds the image and pushes the image to ECR. It also starts a new deployment in ECS.
- Image is pushed to ECR.
- Deployment in ECS is started.

Elastic Container Service

ECS has service and a task. The task has a task definition that has link to the image in ECR, task definition also have network configurations which include security group that exposes application to custom port and internet. ECS deploys the application to instance using Fargate.

Elastic Container Registry

ECR stores the docker images securely and has seamless integration with ECS for easy deployment.

Notification Subscription pipeline

It involves subscribing to SNS topic to receive notification.

- User goes to website and clicks on subscribe button and enters email ID to subscribe.
- The client application sends API request using the API exposed using API gateway which then triggers AWS lambda to subscribe the email to the AWS SNS topic.

AWS Simple Notification Service

SNS has a topic that user can subscribe to.

AWS Lambda

Lambda function that takes email from user and subscribes it to SNS topic.

AWS Api Gateway

It exposes the lambda function using API.

Image upload and processing pipeline

This pipeline takes care of image upload to S3 bucket and processing using Rekognition and sending notification to user.

- User uploads an image through web interface. The client side sends API request which triggers lambda function.
- Lambda function stores the image to S3 bucket.
- Upload to S3 buckets triggers image processing lambda.
- Lambda function using AWS Rekognition services to process image.

- If image has pollution, it sends notification to users through SNS.

AWS S3

Stores uploaded image.

AWS Rekognition

Processed image and sends label and confidence.

AWS Lambda

Lambda function that takes image, decodes it and uploads to S3 bucket. Another lambda function calls Rekognition service to process image then calls SNS to send notification if pollution is detected.

AWS Api Gateway

It exposes the lambda functions using API.

AWS Simple Notification Service

SNS sends notification saying "Pollution detected" when Rekognition finds pollution in image.

Security

Security measures for this project has been taken care for every service and through all staged.

Security: Data at rest

Data at rest secured through various measures. Uploaded images are stored in Amazon S3 which has server-side encryption options such as SSE-S3 or SSE-KMS, ensuring that data is encrypted while stored [6]. Also, access to these images is carefully controlled using S3 bucket policies and IAM roles. For roles, I have used the LabRole given to us.

Also, my ECR repository is private thus it is not accessible to anyone out of my AWS account and is only accessible to ECS.

Security: Data in transit

With the use HTTPS for secure communication, the data in transit is also protected. When images are uploaded from the UI to AWS API Gateway and then to AWS Lambda, the data is encrypted using SSL/TLS protocols, preventing any security issues during transmission.

Security of AWS Services:

S3 bucket: AWS S3 has server side encryption with AWS KMS and Amazon S3-managed keys which protects data. In my cloud formation, I have provided policies to services related to bucket for limited and necessary access. Also there is versioning for protection of data from accidental delete.

SNS topic: SNS topics are secured using IAM policies. The SNS topic sends subscription email to user to verify that the email exists and user has the email ID which helps in providing an extra layer of security.

Lambda Functions: Lambda functions handles errors and exceptions and validates inputs. Also, Lambda functions are secured by providing permissions as per least privilege.

API Gateway: API Gateways are secured by IAM policies. Also have CORS configured to avoid exposing the API to unauthorized access.

AWS Rekognition: My Rekognition model uses IAM role i.e. LabRole. Only necessary permissions are given to Rekognition.

Apart from these, I have CloudWatch to monitor the issues.

What could I have improved?

I have saved the ARN between services that needs to be used inside Lambda function code as environment variables. For the next time, I would like to use AWS secrets for enhanced security and versioning.

Upfront Costs

Upfront Costs to build this application in real world

In real world, this application would be based on private cloud for the maximum security.

Below is the upfront cost estimate:

Infrastructure and Setup:

Infrastructure setup would take up most of the upfront cost.

Server: To support models in Rekognition, high performing GPUs would be required which increases the upfront cost from usual.

Estimate cost: $10,000 - $ 30,000

Storage: Using NAS (Network Attached Storage) or SAN (Storage Area Network) to support storage will incur cost.

Estimate cost: $3,000 - $ 12,000

Network Setup: Setup will require routers, wires, switches.

Estimate cost: $5,000 - $ 10,000

So, total upfront cost would be around $18000 – $52000

On-going Costs to build this application in real world

Maintenance: Maintenance like upgrades, software maintenance to keep application running and prevent from vulnerabilities will also incur cost. Also, power and cooling infrastructure to cool the servers will also cause cost.

Estimate cost: $1000 - $ 2,000 (monthly)

Employees: IT admins, engineers, architects would be required to manage the private cloud.

Estimate cost: $12000 - $ 36,000 (monthly)

So, total on-going cost would be around $22000 – $38000 (monthly)

Additional Costs to build this application in real world

Custom Model training of Amazon Rekognition: The training cost varies on the hours required to train the model. This would also depend upon the dataset. For my Kaggle dataset it took approx. 30 min for training. I am keeping the hour range from 1-3 for estimation

3 training hours x 60 minutes x 0.01666667 USD per minute = 3.00 USD

Estimate cost: $3.00 (one time cost, but for new dataset it has to be retrained again)

Custom domain name: In real world, application will have a domain name not just IP.

Estimate cost: $50-80

In real world, backup and disaster recovery would also incur additional cost.

Cost estimate of AWS services

For cost estimation of AWS services, I have used AWS calculator [5].

Amazon Rekognition

This is the service that can cost the most money i.e. it is dependent on the dataset and training hours and the number of requests thus it needs to be watched and monitored.

Let's say 1-2 million images are processed every month. The rate of image processing per minute 0.000111 USD.

Custom labels usage pricing (monthly): 113.5 USD – 224.50 USD

Amazon S3

For S3 standard with storage 50GB and ongoing monthly number of PUT, GET, COPY, POST or LIST requests as 1-2 million.

S3 Standard cost (monthly): 6.56- 11.96 USD

AWS Lambda: Negligible cost.

API Gateway

For 1-2 million requests per month.

REST API cost (monthly): 3.50 – 7.00 USD

AWS SNS

For 1-2 million requests and 1-2 million Email notifications.

SNS Requests and Notifications cost (monthly): 19.98 USD - 40.48 USD

AWS ECR

For data storage of 5-10GB per month to be stored in repository.

Elastic Container Registry pricing (monthly): 0.50 USD – 10.00 USD

AWS ECS

For a 24*7 run application and machine configuration as 2GB memory, 1 CPU, 1 task/pod running.

Fargate cost (monthly): 865.15 USD

Summary

In total the cost varies on which type of configuration you have for the services/infrastructure. As per the configuration provided with me below is the summary.

AWS Service cost (Low end): $1,008.19

AWS Service cost (High end): $1,159.09

Total upfront cost would be around $18000 – $52000

Total on-going cost would be around $22000 – $38000 (monthly)

Additional cost - $53-83$

Future work

I want to implement real-time video analysis which can analyse video in real-time and send notification if pollution is detected. For this implementation I am thinking of utilizing services

such as Amazon Kinesis Video Streams for capturing and streaming live video, and Amazon Rekognition Video for analysing the content in real time.

I also want to incorporate IoT devices for real-time video analysis, I can use devices equipped with cameras to capture live video streams and send them to the cloud for processing. I want it to connect to AWS IoT Core to stream video data to Amazon Kinesis Video Streams. AWS Lambda will be triggered and send that to analysis to Amazon Rekognition.

References

[1] T. Nguyen, "Gentle Introduction to How AWS ECS Works with Example Tutorial," *Medium*, 9-Sep-2017. [Online]. Available: https://medium.com/boltops/gentle-introduction-to-how-aws-ecs-works-with-example-tutorial-cea3d27ce63d. [Accessed: 07-Aug-2024].

[2] K. Wattamwar, "AWS Elastic Beanstalk vs ECS: Which is Better?", *LinkedIn*, 5-Aug-2023. [Online]. Available: https://www.linkedin.com/pulse/aws-elastic-beanstalk-vs-ecs-which-better-krishna-wattamwar/. [Accessed: 07-Aug-2024].

[3] C. Osuji, "AWS Containerization with Docker, ECR, ECS & Fargate," *Medium*, 29-Sep-2023. [Online]. Available: https://chineloosuji.medium.com/aws-containerization-with-docker-ecr-ecs-fargate-4bd458e5de2e/. [Accessed: 07-Aug-2024].

[4] B. Gaur, "Amazon API Gateway: Concepts and Use Cases," *K21 Academy*, 21-Oct-2023. [Online]. Available: https://k21academy.com/amazon-web-services/aws-certified-security-specialty-amazon-web-services/amazon-api-gateway/. [Accessed: 07-Aug-2024].

[5] "AWS Pricing Calculator," *Amazon Web Services*, [Online]. Available: https://calculator.aws/#/. [Accessed: 07-Aug-2024].

[6] "Cloud Object Storage - Amazon S3," *Amazon Web Services*, [Online]. Available: https://aws.amazon.com/pm/serv-s3. [Accessed: 07-Aug-2024].

[7] "Compute Service - AWS Lambda," *Amazon Web Services*, [Online]. Available: https://aws.amazon.com/pm/lambda. [Accessed: 07-Aug-2024].

[8] "Amazon Simple Notification Service," *Amazon Web Services*, [Online]. Available: https://aws.amazon.com/sns/. [Accessed: 07-Aug-2024].

[9] Draw.io, "Flowchart Maker & Online Diagram Software," *app.diagrams.net*. [Online] https://app.diagrams.net/ [Accessed: 07-Aug-2024].

[10] S. Mondal, "Marine Plastic Pollution," Kaggle, 2023. [Online]. Available: https://www.kaggle.com/datasets/surajit651/souvikdataset/data. [Accessed: 07-Aug-2024].

[11] International Union for Conservation of Nature (IUCN), "Plastic Pollution," [Online]. Available: https://iucn.org/resources/issues-brief/plastic-pollution. [Accessed: 07-Aug-2024].

[12] "More Plastic than Fish," *Plastic Soup Foundation.* [Online]. Available: https://www.plasticsoupfoundation.org/en/plastic-problem/plastic-soup/more-plastic-than-fish/. [Accessed: 07-Aug-2024].