Cloud Computing Assessment 3 Report

Pet Rating Site

Jacob Kumar – S3786688

Marcus Matic – S3703981

Contribution Agreement

Student Name: Marcus Matic	**Student Name:** Jacob Kumar
Student ID: S3703981	**Student ID:** S3786688
Contributions: 1. Back-end Development with Python Flask 2. Basic HTML Development for Integration Testing 3. Integration of Back-end with Elastic Beanstalk 4. Development of Required Lambda Functions 5. Integration of Lambda Functions with Rest API Gateway 6. Incorporating API with Back-end code to allow for POST and GET methods through lambda functions	**Contributions** 1. Front-end Development with ReactJS 2. EC2 Instance creation for Front-end 3. Elastic Container Service creation for Front-end 4. Elastic Container Repository creation for Front-end 5. API call backs within Flask for extra security between Front-end and Back-end
Contribution Percentage: 50%	**Contribution Percentage:** 50%
By signing below, I certify all information is true and correct to the best of my knowledge. **Signature:** **Date:** 13/06/2021	*By signing below, I certify all information is true and correct to the best of my knowledge.* **Signature:** **Date:** 13/06/2021

Links

Website: **http://ec2-13-239-83-16.ap-southeast-2.compute.amazonaws.com/**

GitHub Repo: **https://github.com/EnzoExemplary/CC-A3**

Summary

Our initial objective was to design a project that aimed to provide both a level of entertainment and functionality to a target userbase. We also needed to be sure that we selected a project idea that would be able to utilise the varying range of AWS services available.

Introduction

Our idea brainstorming focused primarily on our interests, creating a range of possible projects, such as an image sharing/hosting site (similar to sites like *Imgur*), a recipe/cooking site, or a pet hosting site. We both shared a great love for pets, so we decided on adapting the pet hosting site idea into a pet sharing and rating site, providing a good entertainment medium for fellow pet lovers and the likes.

We built our pet rating project with intent for users to create accounts and share their beloved pets (whether fictional or non-fictional) by uploading a photo of them and entering their name. By uploading the pet, it is registered with the account and is public visible to all site users (whether logged in or not). If another user is logged in, they can rate the pet from 1-5 stars, and/or leave a comment about them. We wanted users to be able to discover any of the uploaded pets by browsing the homepage which would display a pet at random for the user to admire, rate, and/or comment on. We also wanted a good range of pet discoverability so users would be able to look through users' public profiles and explore all their uploaded pets and the pets they have rated. Additionally, we wanted to add a search page so users could search for different pet names uploaded to the site.

Such a site is beneficial to the everyday person as it provides a calm and relaxing serotonin-boosting environment where users can sit back and cycle through different pets, admiring, rating, and discussing them in an easy-to-use public environment.

Given the amount of stress and work involved in the day-to-day life, such entertainment services are essential to boosting happiness and morale, especially from the recent demoralising pandemic and sporadic lockdown procedures.

Our goal was to provide the service for large target audiences so we wanted to be sure the site was simple and easy to use for of varying ages and technological skills, which we did by including the major functionality of the site on the home page, where uses can continue to look through and rate each pet as they come.

Related Works

From our initial research, we were unable to find existing websites that provide the same type of service that we intended to develop, which further increased our excitement to build the application.

However, there are quite a few existing sites that function as a social applications/sites but target animal and pet lovers. A couple examples we found were:
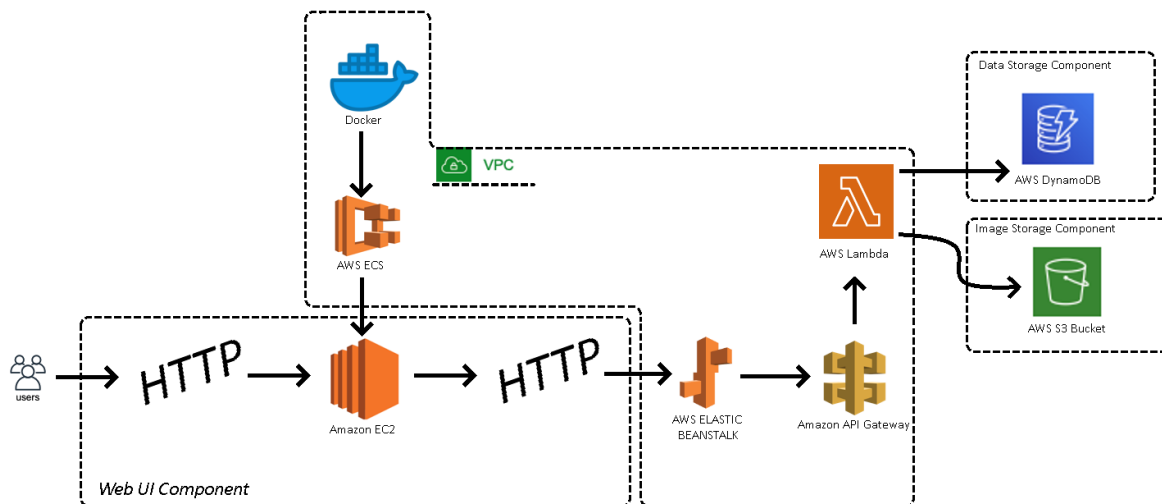
- https://petsforever.io/
- https://petzbe.com/

These applications focus more on the social media approach to their implementations whereas our intention is to focus more on the pets themselves and browsing through all the different submitted pets to admire and appreciate them.

While not a specific website itself, a social media account known as 'WeRateDogs' is widely popular on both Twitter and Facebook. This account rates dogs that can be submitted here and posts them on these social media pages with a rating that generally exceeds their rating cap of 10 (although it is often justified).

The WeRateDogs pages shows that there's a definite interest in such a platform where sharing and rating pets is the major focus.

System Architecture

Implemented Services

Service	Description	Purpose
EC2	Scalable virtual computers that run any software	EC2 is used to host the frontend of the website, separated from the backend. Its purpose is to be both scalable and easily accessible with limited security needs.
ECS	Container management and deployment using automation	Connected to the EC2 instances, manages the allocation of the ECR repositories for rapid deployment of changes to the frontend
ECR	A central host for storing generated and ready to go software packages	ECR stores the frontend in docker images
Docker	Containerizes software packages for use in scalable environments	Through GitHub actions, pushes frontend changes to ECR and notifies the ECS to update the EC2 instance
Elastic Beanstalk	Web application management service used for deployment and web application scaling with varying compatible scripts.	We chose EB as our web application manager for the backend as it is very streamline, easy to use, supports Python Flask, handles load balancing, and provides great health monitoring.

API Gateway (REST)		A REST API Gateway was the best option for our 'front door' service as it is built for serverless workless and HTTP backends in an easy-to-use single service. The API Gateway is a must for improving our implementation's security as it essentially adds a 'middle-man' between the application and the cloud data so that the application never directly accesses the storage.
	Fully managed servicing for secure APIs and acts as a '*front door*' real-time two-way communication service for accessing back-end data and providing it to requesting applications.	
Lambda	Serverless compute service that allows for code to be run without any form of provisioning or managing servers. This allows utilisation with the API Gateway to develop our GET and POST methods under each endpoint. The lambda code directly accesses the cloud data and returns specific data based on the function called.	Lambda functions are essential for our project as they work in congruence with the API Gateway to improve the overall security of the application.
DynamoDB	Serverless NoSQL key-value document database that includes built in security, restore/backups, and in-memory caching.	Our application did not require a complex data structure, so DynamoDB was the optimal choice over other available database services such as RDS as it provides excellent read and write speeds for simple data structures and queries such as the likes our data structures and queries.
S3	Simple Storage Service for files that provides scalability, security, high performance, and simple management features.	For our current implementation of our project, we only needed a simple storage for each pet image, so an S3 bucket was the most appropriate option as it includes its own security management and performs best with simple storage systems.

DynamoDB Structure

Table Name	Primary Key	Other Keys
comment	id (String)	*pet_id* (String), *text* (String), *username* (String)
pet	id (String)	*img_url* (String), *name* (String), *owner* (String)
rating	id (String)	*pet_id* (String), *rating* (int), *username* (String)
user	username (String)	*email* (String), *num_pets*(int), *password*(String), *profile_bio*(String)

APIs

- */add-pet*
  - *POST*: Used to post a given pet into the database (data stored in DynamoDB, image stored in S3 bucket)
- */login*
  - *POST*: Passes provided login information to check against existing users and whether a matching username and password is found
- */pet*
  - *GET*: Gets a pet from the database based on provided pet_id
    - */average-rating*
      - *GET*: Get average rating of a specific pet (from pet_id)
    - */comment*
      - *GET*: Get the comments posted on a specific pet page (from pet_id)
      - *POST*: Post a comment for a specific pet
    - */random*
      - *GET*: Get a random pet from the database (for the homepage)
- */rating*
  - *GET*: Get a particular rating based on its id
  - *POST*: Add a new rating for a pet into the database
- */register*
  - *POST*: Add a new account into the database (if possible)
- */search*
  - *GET*: Get search results based on given search query
- */user*
  - *GET*: Get a user by their username
    - */bio*
      - POST: Add/update an existing user's bio
    - */pets*
      - GET: Get all pets that the user has uploaded
    - */rated*
      - GET: Get all pets that the user has rated (including the rating they gave to that pet)

Developer Manual

Local Setup:

- Navigate to the directory the frontend source code is in
- Install Docker
- Install AWS-CLI on your PC, run 'aws configure' in the command line to set it up
- Run the command:

```
aws ssm get-parameters --names /aws/service/ecs/optimized-ami/amazon-linux-
2/recommended
```

Make note of the 'image_id' in the output JSON. It will look like 'ami-0ba6df717ed766b7c'. Remember this for setting up EC2

Setting up the ECS Cluster:
- Go to Services, Containers and then select 'Elastic Container Service'
- Click the 'Create Cluster' button
- Select 'Networking Only' and then click 'Next Step'
- Enter in a cluster name, remember this for setting up EC2. Leave the rest as default.
- Press 'Create'

Setting up an ECR Image for Docker:
- Go to Services, Containers and then select 'Elastic Container Registry'
- In the private tab, click 'Create repository'
- Give it a name then click 'Create repository' at the bottom of the page
- Copy and store the newly created repository URI from the table. This will be needed for the task definition.

Setting up EC2 for ECS:
- Go to Services, Compute and then select EC2
- Find the 'Launch Instances' button and press it
- Select Community AMIs
- Search for the image id you retrieved earlier in 'Local Setup for AWS'
- Select the Amazon Machine Image, it should have 'ecs' somewhere in the name.
- Select the default free tier (1vCPU, 1GB)
- Select the default VPC
- Set 'auto-assign public ip' to 'Enable'
- In IAM Role, if 'ecsInstanceRole' is there, select it. If not, press 'Create new IAM role' and do the following:
    - Press 'Create Role'
    - Select 'AWS Service' for 'Type of Trusted Entity'
    - For 'Use Case' select 'Elastic Container Service', then it will show more options, select 'EC2 Role for Elastic Container Service'
    - Go to the 4th page, under Role name, enter 'ecsInstanceRole'
    - Press 'Create Role'
    - Go back to the previous tab and press the refresh buttons for the roles. Select 'ecsInstanceRole' now.
- Under 'User data' in the Advanced Details tab, enter the following

    Replace [cluster name] with the name of the cluster you created earlier.

    ```
    #!/bin/bash
    echo ECS_CLUSTER=[cluster name] >> /etc/ecs/ecs.config
    ```
- In "Configure Security Group", add HTTP and HTTPS rules
- Press 'Review and Launch', confirm the details are correct then press 'Launch'

Creating a new ECS Task Definition:
- Navigate to the 'Task Definitions' page under 'Amazon ECS'
- Click 'Create new Task Definition'
- For launch type compatibility, select EC2
- Name the task definition
- Scroll down to 'Container Definitions', click 'Add container'
- Name the container
- Set the 'image' to the URI you saved earlier.
- Setup the port mappings. The host port is 80, the container port is 8080
- Press 'Create'

Creating the ECS Service:
- In the ECS cluster created earlier, navigate to the services tab and then press 'Create'
- For 'Launch type' select EC2
- For service name, name it the same as the task definition,
- Set the 'number of tasks' to 1
- Set the 'service type' to 'replica'
- Press 'next step' until you reach the final page, review the configuration and then press 'Create Service'

Deploying and Updating the service:
- Inside the working directory for the frontend, open a command line terminal.
- Run 'aws ecr get-login-password --region <Region> | docker login --username AWS --password-stdin <AWSAccountID>.dkr.ecr.<Region>.amazonaws.com', replacing <Region> with the region the previous steps were setup in and <AWSAccountID> with the account id associated. Both of those can be derived from the URI of the repository
- Run 'docker build -t <Repository>', replacing <Repository> with the name of the repository created
- Run the command 'docker tag <Repository>:latest <URI>:latest', replacing <Repository> with the name of the repository created and <URI> with the uri of that repository
- Run the command 'docker push <URI>:latest', replacing <URI> with the uri of that repository
- After some time, the EC2 instance will finish deploying the Image. To view it:
  - o Go to the page for the EC2 instance created earlier
  - o Copy the address listed under 'Public IPv4 DNS', change it to 'http' as SSL was not set up.
  - o Now you can visit the webpage

Creating Permission Role for Lambda Functions:
- Navigate to the IAM portal on AWS
- Select roles on the left hand side
- Click 'Create Role'
- Select Lambda as the use case and click 'Next: Permissions'.
- Search for and select the following permissions:

- o AmazonS3FullAccess
- o AmazonDynamoDBFullAccess
- o AWSLambdaBasicExecutionRole
- o AWSLambda_FullAccess
- Click 'Next: Tags'
- Click 'Next: Review'
- Give the role an appropriate name, such as 'lambda-permissions'
- Select 'Create Role'

Deploying Lambda Functions:

- For each .zip file in the 'lambda functions' folder:
  - o From the Lambda Management Console on AWS, select 'Create Function'
  - o Set function name to match the .zip's title (excluding the unique ID after it)
  - o Set runtime as Python 3.8
  - o Select the drop-down menu 'Change default execution role' and select the 'lambda-rest-api' permissions created previously
  - o Click 'Create Function'
  - o On the upper right corner of the 'Code source' window, select 'Upload from' -> '.zip file' and select the appropriate .zip file for that function

Deploying API Gateway and Linking with Lambda Functions:

1. Go to the API Gateway management paeg and select 'Create API'
2. Find the public 'REST API' and click 'Build'
3. Set the API name to something appropriate (e.g. cc-a3-rest)
4. Click 'Create API'
5. On the 'Actions' drop-down menu, select 'create-resource'.
6. Give the resource an appropriate name and path for the function (see function mapping below) and click Create Resource
7. For that resource, select 'Create Method' from the action drop-down menu
8. Select GET/POST from the newly presented drop-down menu depending on the requirement for the function, then click the tick
9. In the 'Lambda Function' input field, type in and select the appropriate function for the API method, then click save
10. For each API mapped to their respective lambda functions in the function mapping table, repeat steps 5-9 until all APIs have been added
11. On each method, select 'Enable CORS' from the action menu and continue to enable for all methods
12. Finally, in the action menu select deploy API

API – Lambda Function Mapping Table

API	Method	Lambda Function
/add-pet	POST	upload-image-to-s3
/login	POST	check-password-match
/pet	GET	get-pet-by-pet-id
/pet/average-rating	GET	get-average-rating-by-pet-id
/pet/comment	GET	get-comments-by-pet-id
/pet/comment	POST	add-comment
/pet/random	GET	get-random-pet
/rating	GET	get-rating-by-rating-id
/rating	POST	update-rating
/register	POST	register-account
/search	GET	get-search-results
/user	GET	get-user-by-username
/user/bio	POST	update-user-bio
/user/pets	GET	get-pets-by-owner
/user/rated	GET	get-pets-rated-by-username

Brief Manual

1. To properly use the website, you will first need to create an account, which can be done by selecting 'Register' in the top right corner.

2. Fill out the appropriate fields and select register to create your account.

3. Navigate back to the homepage, and you will now be able to rate each random pet that comes up (whenever you hit the 'Show another random pet!' button

4. Additionally, comments can be left by registered accounts on pets by clicking the comment drop down tab and submitting a comment.

- If there is a particular pet that you would like to search for, you can navigate to the search page and enter letters or pet names to get all the pets whose name equals or contains your search query.

- Registered users can also submit their own pet to their account by selecting the 'Add Pet' button on the navigation bar, then uploading an image and entering a name.
  - o This will input your pet to the website so other users can see them and rate them.

- User profiles are public so you may want to edit your bio! This can be done from the 'user' panel on the navbar, where you will be able to edit your profile bio.

- On your user profile you and other users will be able to see all your submitted pets and all the pets you have previously rated (as well as the rating you gave)

References

[1] M. Klems, *AWS Lambda Quick Start Guide: Learn How to Build and Deploy Severless Applications on AWS*. Birmingham: Packt Publishing, Limited, 2018.

[2] M. Wittig, A. Wittig, and B. Whaley, *Amazon Web Services in Action*, 2nd ed. Shelter Island, NY: Manning, 2019.

[3] J. Bradshaw, *The Animals Among Us: How Pets Make Us Human*. NY: Basic Books, 2017.

Appendix

Initial Project Sketches

Rating PET PAGE

NAV BAR

★★★★☆ (42)

Scroll wheel on overflow

Comments

[Text] [Date] [User]

*Container filled w/ all comments
* Each comment shows date created and the user's name (username links to that profile)

PET IMAGE

Leave a comment...

Redirect to comment

LEAVING A COMMENT PAGE

NAV BAR

Comment...

Text box
* Max 250 chars.

Comment

Button to post