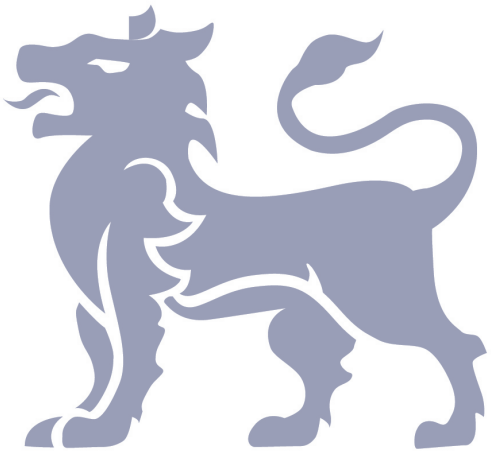


Cloud Computing with AWS

By Adam Cordner, Christopher Perry & Evie Snuggle (Group 4)

A report submitted as part of a required module
for the degree of BSc. (Hons.) in Computer Science
at the School of Computing and Digital Technology,
Birmingham City University, United Kingdom

May 2022,
CMP6210 Cloud Computing 2021–2022
Module Coordinator: Dr Khaled Mahbub

Student IDs: 18109958, 18103708, 18128599

Contents

List of Figures

Chapter 1

Glossary

- **ACL** — Access Control List
- **ACM** — AWS Certificate Manager
- **AMI** — Amazon Machine Images
- **App** — Application
- **AWS** — Amazon Web Services
- **AZ** — Availability Zone
- **CA** — Certificate Authority
- **CDS** — Content Delivery Network
- **CIDR** — Classless Inter-Domain Routing
- **CPU** — Central Processing Unit
- **DB** — Database
- **DIG** — Domain Information Groper
- **EB** — Elastic Beanstalk
- **EC2** — Elastic Cloud Compute
- **ELB** — Elastic Load Balancing
- **.env** — Environment File Extension
- **GB** — Gigabyte
- **HTTP** — HyperText Transfer Protocol
- **HTTPS** — HyperText Transfer Protocol Secure
- **IAM** — Identity and Access Management
- **IEEE** — Institute of Electrical and Electronics Engineers
- **IP** — Internet Protocol

- **IPv4** — Internet Protocol Version 4
- **IPv6** — Internet Protocol Version 6
- **LAMP** — Linux, Apache, MySQL, PHP
- **ML** — Machine Learning
- **NAT** — Network Address Translation
- **nslookup** — Name Server Lookup
- **OS** — Operating System
- **.pem** — Privacy Enhanced Mail File Extension
- **PHP** — PHP: Hypertext Preprocessor
- **RAM** — Random Access Memory
- **S3** — Simple Storage Service
- **RAM** — Random Access Memory
- **RDBMS** — Relational Database Management System
- **RDS** — Relational Database Service
- **SaaS** — Software as a Service
- **SNS** — Simple Notification Service
- **SQL** — Structured Query Language
- **SSH** — Secure Shell
- **SSL** — Secure Sockets Layer
- **TLS** — Transport Layer Security
- **URL** — Uniform Resource Locator
- **VPC** — Virtual Private Cloud
- **VPN** — Virtual Private Network
- **WAF** — Web Application Firewall

Chapter 2

Introduction

This report details the process of designing, developing, and deploying a cloud application onto Amazon Web Services (AWS). The application is called *Digital Ink* and allows users to create, edit, and delete their own short stories. Users can then view their own short stories and other users' short stories. It was first developed locally using a LAMP stack. This consisted of Linux - hosted through Docker - for the operating system, an Apache HTTP Server, MySQL for the relational database management, and PHP as the programming language.

After the application was built locally, it was gradually integrated onto AWS. This involved implementing several AWS cloud features to enhance the application, ensure application security, and increase availability. This was accomplished by using Simple Storage Service (S3), Elastic Compute Cloud (EC2), ELB (Elastic Load Balancing), and more. The process of implementing these cloud features will be discussed throughout the report.

After the application was integrated onto AWS, an evaluation of the process was conducted. This includes a discussion of the security practices used, estimated costs for different user scales, and thorough testing. Lastly, several enhancements which could be made to the application in the future will be discussed.

Chapter 3

Web App

This chapter of the report will detail the local design and development of the *Digital Ink* web application. This chapter will first discuss the software stack used to develop the app, then the design of the database used, and, lastly, the design of the user interface.

3.1 Software Stack

Digital Ink was first developed locally using a LAMP stack. LAMP refers to a generic software stack, where each letter in the acronym stands for one the following open source building blocks: Linux, Apache HTTP Server, MySQL, and PHP (Lee and Ware, 2003). The web app is hosted within a Docker container (Anderson, 2015) which runs a minified version of the Linux operating system. Apache is an open-source web server software which is used to host the app on the web (Fielding and Kaiser, 1997). MySQL is an open-source relational database management system (Widenius, Axmark, and Arno, 2002) which is used to store all the data used within the app, including user details and story details. PHP is a programming language aimed towards web development, chosen due to its stability and reliability (Lerdorf et al., 2002). Additionally, all developers involved have prior experience with PHP.

3.2 Database Design

As mentioned before, the web app uses the MySQL relational database management system to store its data. MySQL is a relational database management system (RDBMS) which stores data in the form of tables, where Structures Query Language (SQL) is used to access the database. `users`, `stories`, and `migrations`.

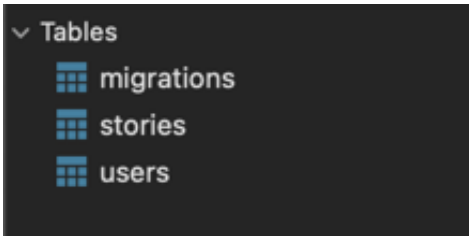


Figure 3.1: Database tables overview.

The `migrations` table (see Figure 3.2) contains records which correspond to the migrations within the Laravel web app. These migrations contain the scripts required to automatically generate the `users` and `stories` tables in SQL. It contains the following three columns:

- `id`: the unique ID for each migration.
- `migration`: points to the scripts used to create tables.
- `batch`: how many times the script has been ran.

id	migration	batch
1	2014_10_12_000000_create_users_table	1
4	2020_03_13_105916_create_stories_table	1

Figure 3.2: `migrations` table.

The `users` table (see Figure 3.3) contains all the information about user accounts, and it contains the following seven columns:

- `id`: the unique ID for each user account.
- `name`: the name associated with user account.
- `email`: the unique email used to log in.
- `password`: the password used to log in, encrypted with 184 bit hashing by Bcrypt (Laravel, 2022b).
- `remember_token`: keeps the user logged in if they select "Remember me".
- `created_at`: records what date and time the user account was first created at.
- `updated_at`: records what date and time the user account was last updated at.

SQL MySQL 8.0.28 : TLSv1.2 : digital-ink : group4 : users

	stories			users		
id	name	email	password	remember_token	created_at	updated_at
1	Adam	adam.cordner@mail.bcu.ac.uk	$2y$10$ISS.3eqO3o0dRXaK8sMy.eBqUapgD...	mDUlGRilVhyEi6...	2022-05-10 15:50:27	2022-05-10 15:50:27
2	evie	^-^@snugg.ie	$2y$10$vVNRmMN/IRSlYTrCUjJEPu44jkVI0r5...	Mu1WdyOz1e8...	2022-05-10 15:51:29	2022-05-10 15:51:29

Figure 3.3: `users` table.

The `stories` table (see Figure 3.4) contains all the information about user-created stories, and it contains the following 11 columns:

- `id`: the unique ID for each story.
- `author_id`: the unique ID associated with the user who created the story.
- `title`: the title associated with the story.
- `genre`: the genre associated with the story, which can be one of eight different genres.
- `blurb`: a brief description of the story.
- `content`: the full content of the story.
- `cover_image`: a thumbnail image for the story.
- `file_upload`: an optional PDF upload of the story.
- `published`: 1 if the story has been made public, or 0 if it is a draft.
- `created_at`: records what date and time the story was first created at.
- `updated_at`: records what date and time the story was last updated at.

id	author_id	title	genre	blurb	content
2	1 →	Group 4 Loves AWS	Romance	Meet Group 4 and their love for AWS!	Group 4 loves using AWS' cloud features to host Di...
3	2 →	cheese savoury	Action and Adventure	nice and simple	on malted granary

cover_image	file_upload	published	created_at	updated_at
../storage/cover_images/753732910165220060 6.j...	NULL	1	2022-05-10 16:36:46	2022-05-10 16:36:46
images/digital-ink-logo.png	NULL	1	2022-05-10 16:38:51	2022-05-10 16:38:51

Figure 3.4: `stories` table.

3.3 Interface Design

The design of the web app was created using Blade, a powerful templating engine (Laravel, 2022a). When the user initially accesses the web app, they are able to log in or sign up. This can be seen in Figure 3.5. When a user has created an account, a record is written to the `users` table in the database.

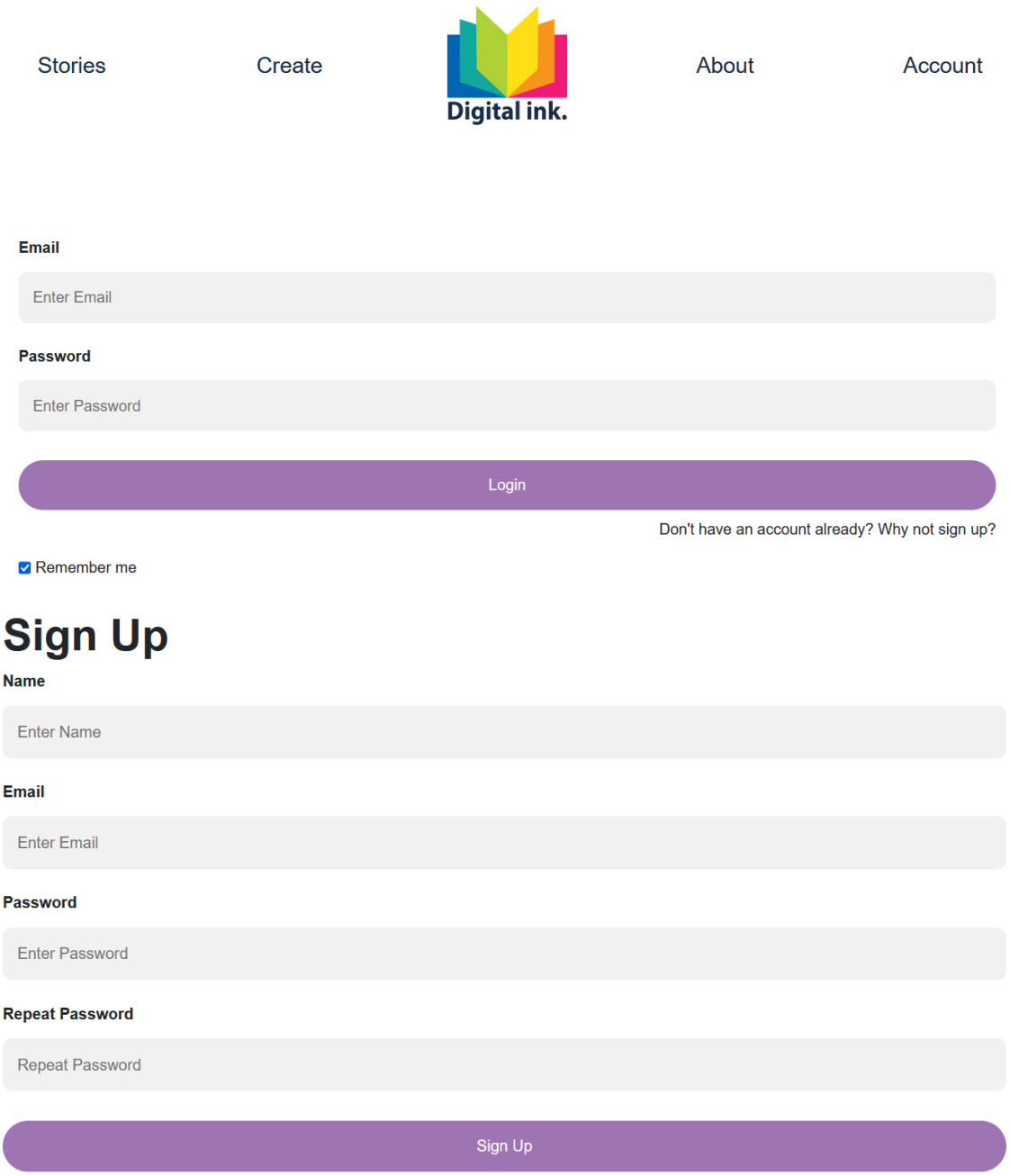


Figure 3.5: *Digital-Ink* home page log in and sign up forms.

Once a user is signed in, they can create a story. Creating a story requires the user to entire a title, a genre, the story itself, a blurb, and, optionally, a thumbnail image. This can be seen in Figure 3.6. Once a story has been created, it is written to the `stories` table.

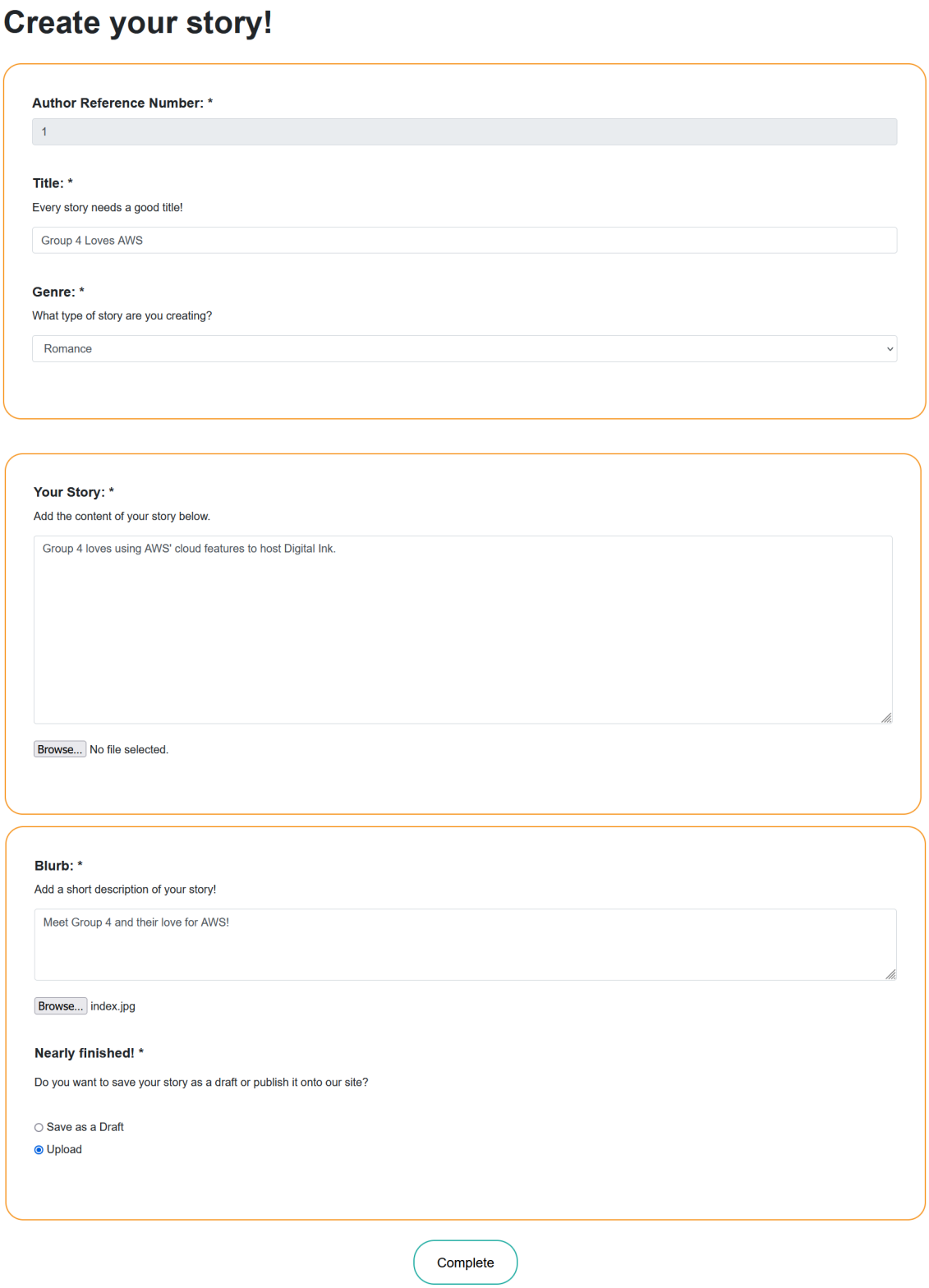


Figure 3.6: *Digital-Ink* story creation form.

After this, the user can see all of their uploaded stories on their account page. This can be seen in Figure 3.7. From here, a story can be edited or deleted, which either updates a record in the `stories`table or removes a record from it.

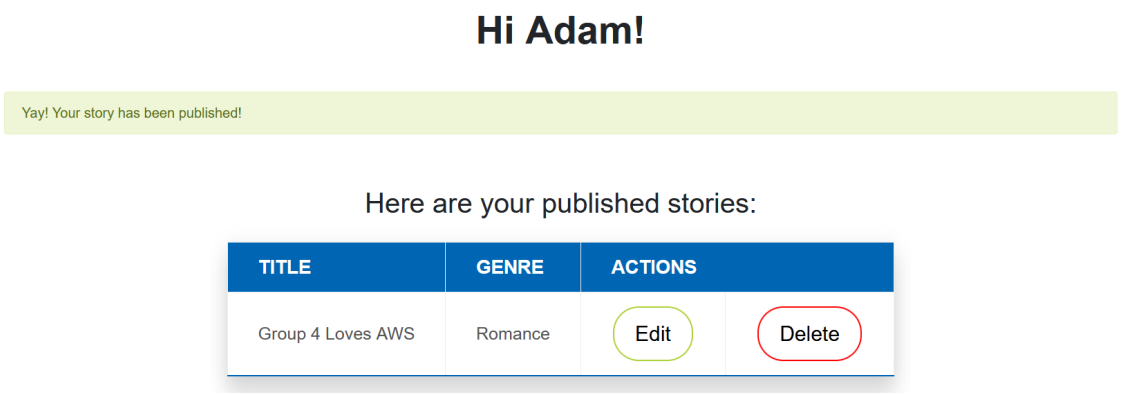


Figure 3.7: *Digital-Ink* account page.

Lastly, on the Stories page, a user can view and search through all uploaded stories across all users. Each story's title, genre, and blurb is shown in a list view. A user can click into one of these stories to see the thumbnail image and read the full story. These pages can be seen in Figure 3.8.

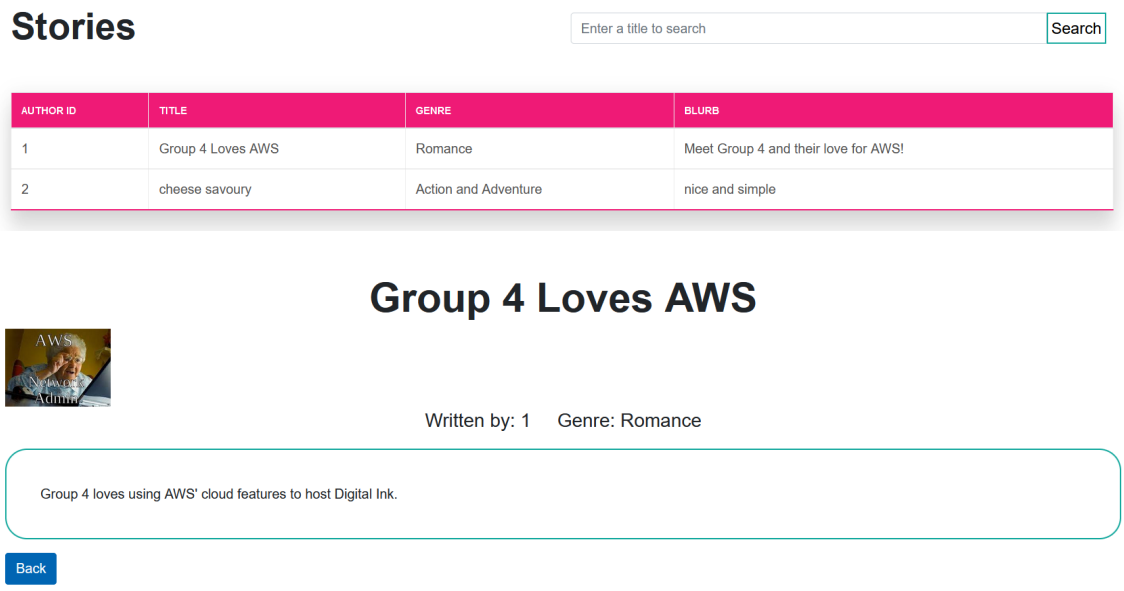


Figure 3.8: *Digital-Ink* stories page and story view.

Chapter 4

Virtual Private Cloud (VPC)

Amazon VPC allows for AWS resources to be launched in a virtual network that has been custom defined and configured. This virtual network is similar to a traditional network which operates within your own physical data center, with the added benefits of the scalable AWS infrastructure (Amazon Web Services (AWS), 2022i). A VPC can have multiple assigned subnets, which are a range of IP addresses accessible in the VPC. This chapter of the report will detail the creation of an Elastic IP address, a VPC, subnets, an internet gateway, and route tables.

4.1 Elastic IP Address

Before creating the VPC, an Elastic IP address must be created, which can then be allocated to the VPC. To do this, the **Allocate Elastic IP address** button on the Elastic IP Addresses page must be clicked.

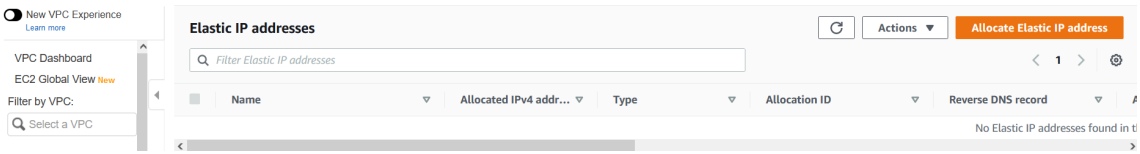


Figure 4.1: Elastic IP Addresses page.

To configure the Elastic IP address, a Network Border Group must be selected, ensuring that this is the same as the Availability Zone selected when configuring the VPC. Once this has been set, and the remaining options are set to their default values, clicking the **Allocate** button will create the Elastic IP address.

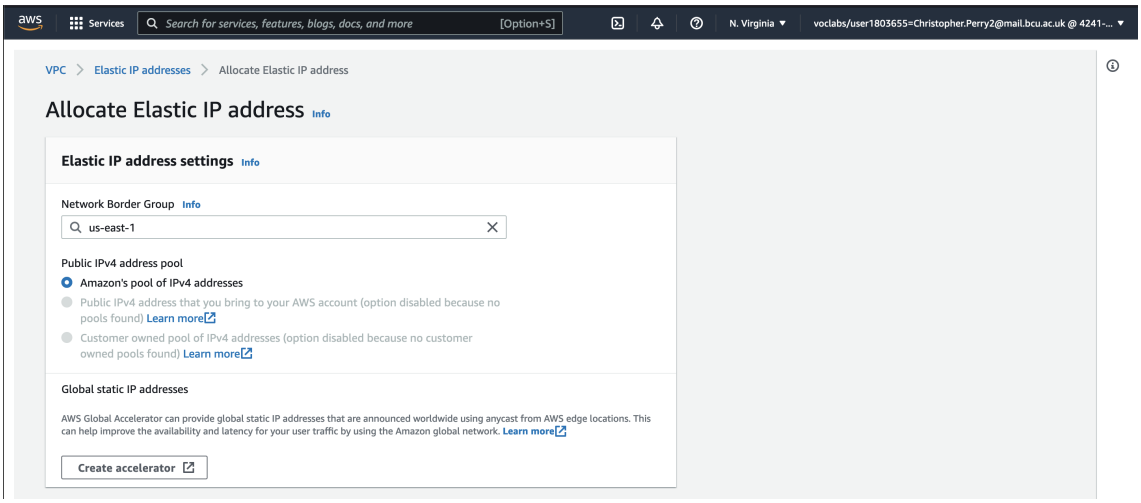


Figure 4.2: Configuring the Elastic IP address.

After this, the Elastic IP address created can be seen from the Elastic IP Addresses page.

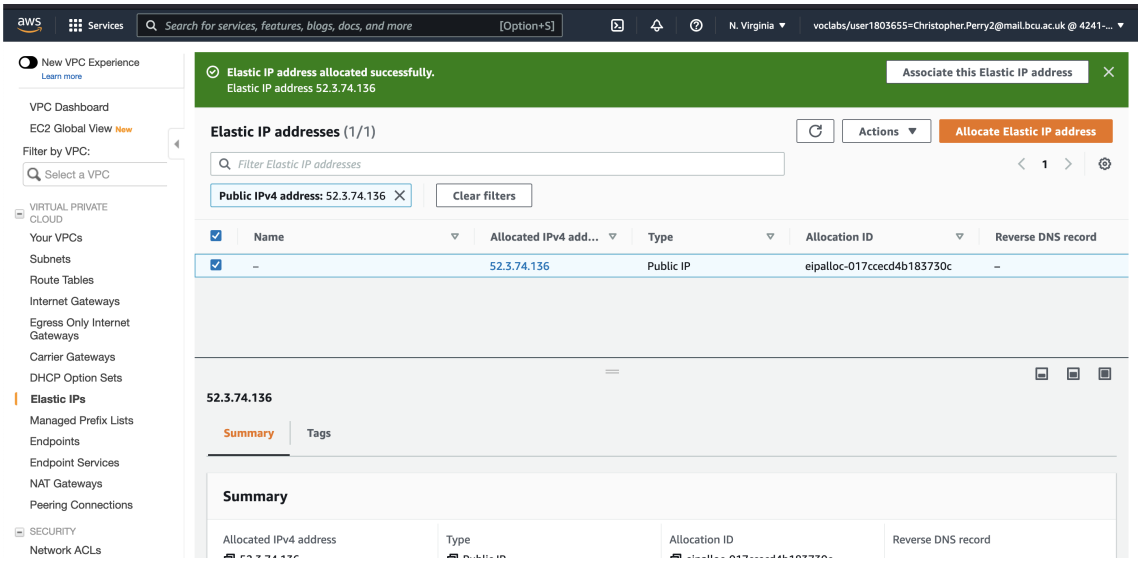


Figure 4.3: Created Elastic IP address.

4.2 VPC and Subnets

The first step of migrating the web app to the cloud was creating a VPC to deploy and manage AWS resources for the app. To do this, the VPC wizard must be used. This is accessed by navigating to the VPC Management Console and then click the **Launch VPC Wizard** button, which can be seen in Figure 4.4.

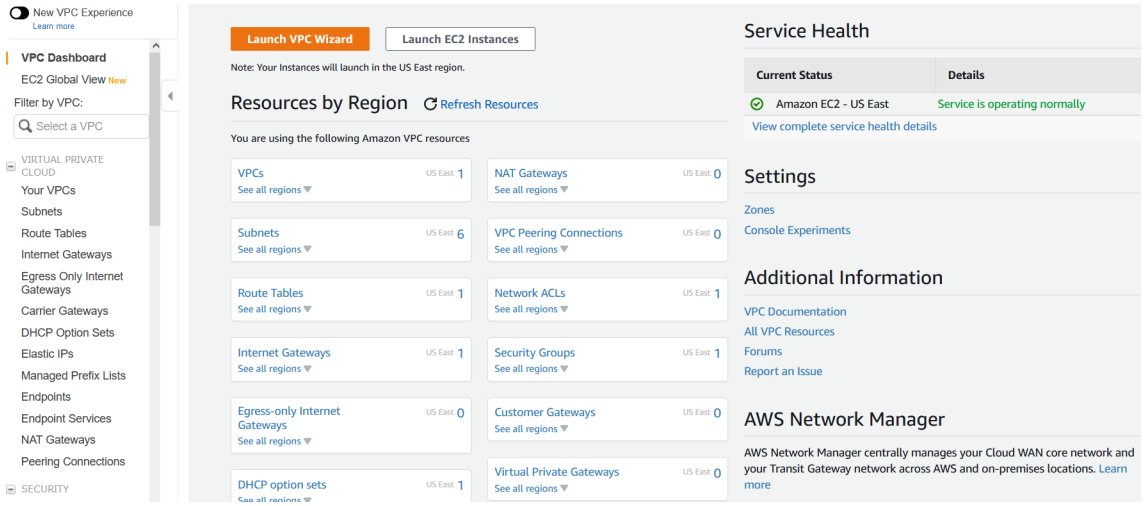


Figure 4.4: VPC Management Console.

The wizard presents us with step one of creating a VPC: selecting a VPC configuration. There are several options for this, where each configuration has the following features:

- VPC with a Single Public Subnet: Creates a /16 network with a /24 subnet where the subnet instances use Elastic IPs or Public IPs for internet access.
- VPC with Public and Private Subnets: Creates a /16 network and two /24 subnets where the public subnet instances use Elastic IPs for internet access and the private subnet instances use NAT for internet access.
- VPC with Public and Private Subnets and Hardware VPN Access: Creates a /16 network with two /24 subnets where one subnet is connected to the internet and another is connected to your personal network via a VPN.
- VPC with a Private Subnet Only and Hardware VPN Access: Creates a/16 network and a /24 subnet where the subnet is connected to your personal network via a VPN.

For the deployment of this web app, hardware VPN access is not necessary, so the VPC configuration with public and private subnets was selected, as seen in Figure 4.5.

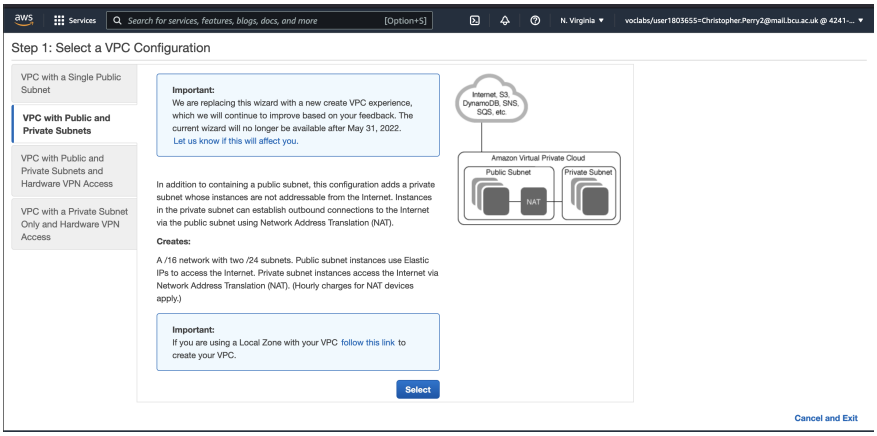


Figure 4.5: Selecting a VPC configuration.

Following this, the specific details, such as the IP address block, of the selected configuration must be entered. An IPv4 CIDR block of `10.0.0.0/16` was chosen as it provides a large amount of available IP addresses. Additionally, no IPv6 CIDR block was configured and the VPC was named `Group4_VPC`.

Next, the public and private subnets were configured. The public subnet was assigned an IPv4 CIDR of `10.0.0.0/24`, making 251 distinct IP addresses available for the public subnet. The availability zone of `us-east-1a` was chosen and the subnet was named `Public subnet 1`. The private subnet was assigned an IPv4 CIDR of `10.0.1.0/24`, making 251 distinct IP addresses available for the private subnet as well. The availability zone of `us-east-1a` was chosen and the subnet was named `Private subnet 1`.

Lastly, the previously created Elastic IP Allocation ID was selected, and the remaining settings were left at their default values. Clicking the **Create VPC** button now will generate the VPC with the specified configurations.

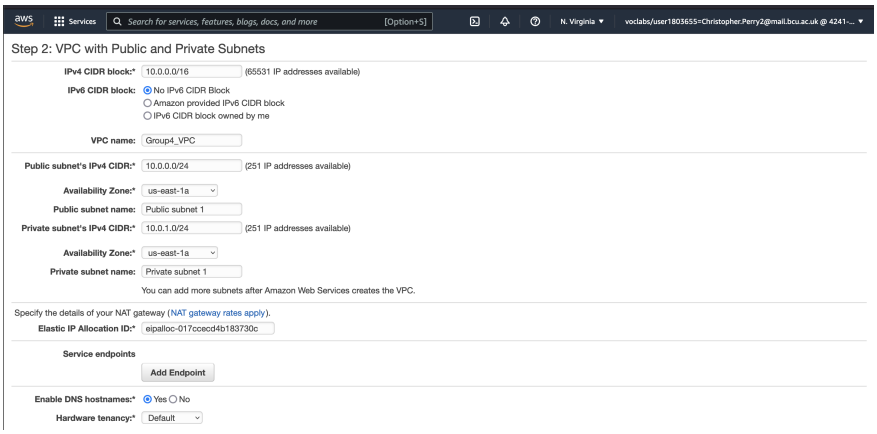


Figure 4.6: Configuring VPC public and private subnets.

After this, AWS takes a few minutes to generate the VPC.

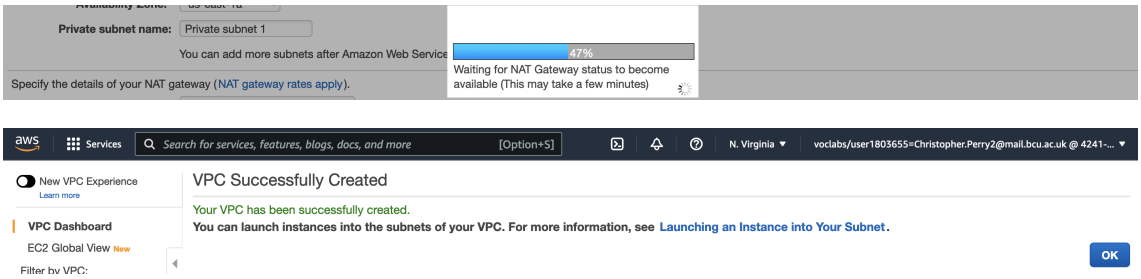


Figure 4.7: Generating the VPC.

Navigating from here to the Your VPCs page shows two available VPCs: the VPC created by the AWS sandbox environment, which will not be used, and the VPC which has just been created, named `Group4_VPC`.

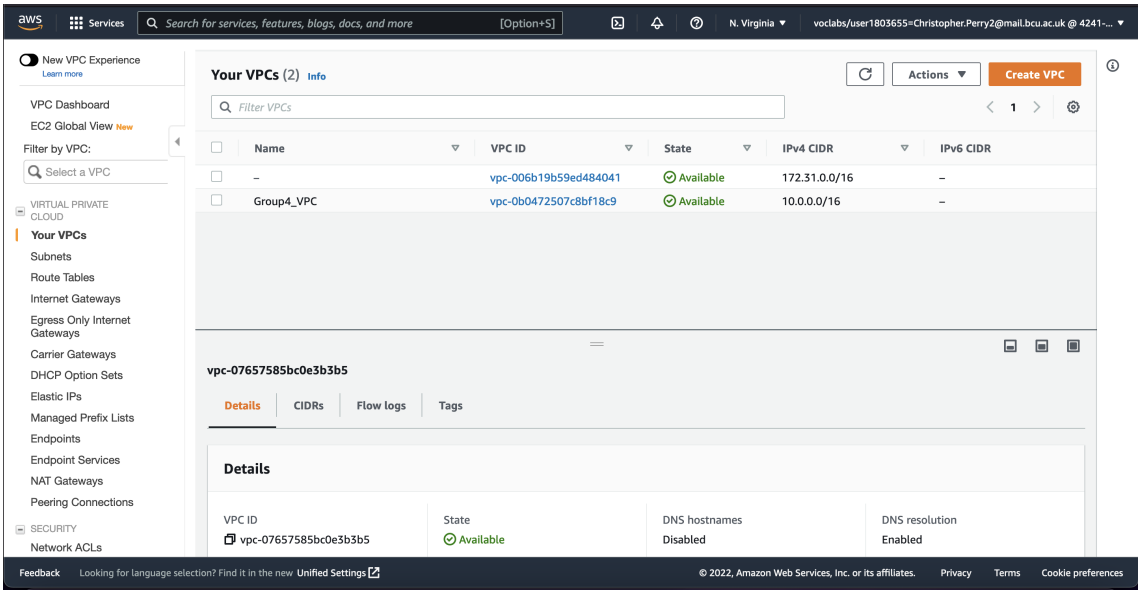


Figure 4.8: Your VPCs page.

4.3 Internet Gateway

An internet gateway is required for resources, such as EC2, to be able to communicate with the internet, if the resource has a public IP address (Amazon Web Services (AWS), 2022f). To do this, navigate to the Internet Gateways page and click the **Create internet gateway** button. Here, enter the internet gateway name, such as `Group4-Internet-Gateway`.

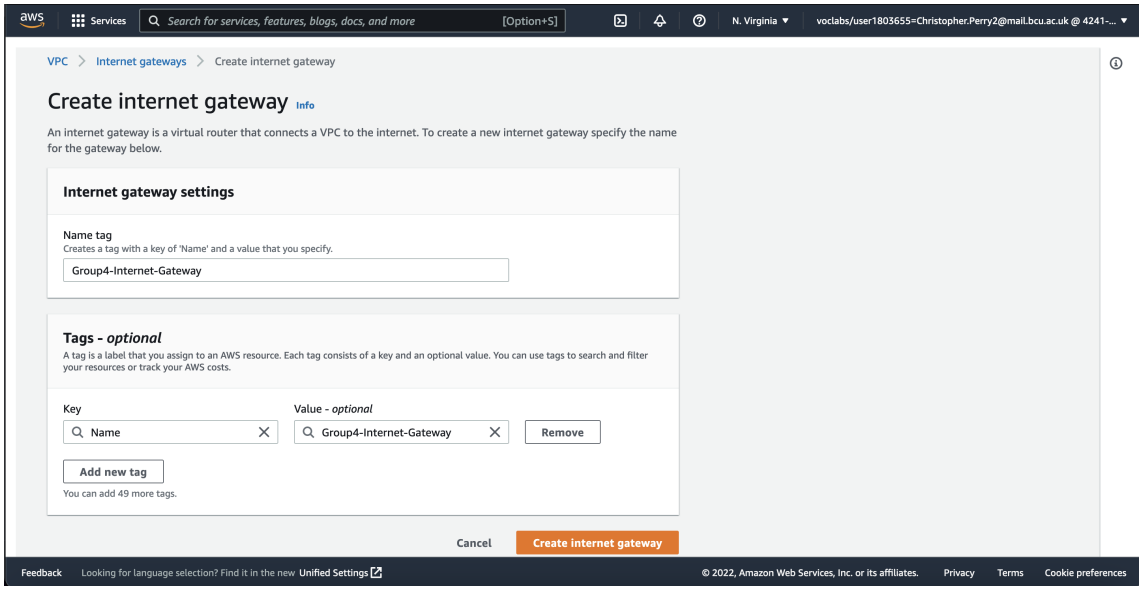


Figure 4.9: Creating an internet gateway.

After entering the internet gateway name and clicking the **Create internet gateway** button, the newly created internet gateway can be seen (as well as an automatically generated internet gateway for the default VPC, which will not be used).

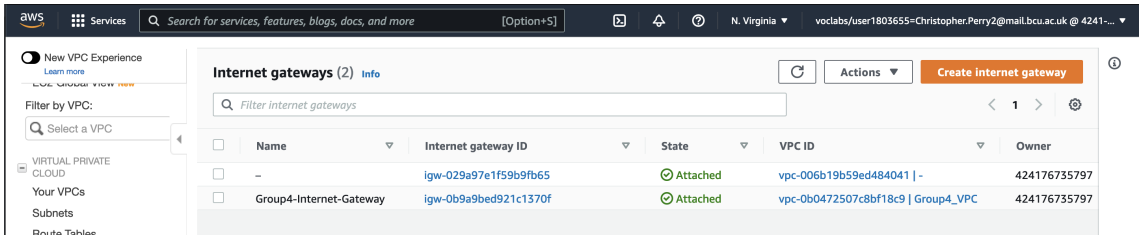


Figure 4.10: Viewing the newly created internet gateway.

4.4 Route Tables

The newly created VPC does not currently have a sufficient level of availability, as it is only available in one Availability Zone - `us-east-1a`. The availability of the VPC can be increased by creating a new pair of public and private subnets in a different Availability Zone. This is done by navigating to the Create Subnets page from the VPC Management Console.

First, to create `Public subnet 2`, it is required to specify a VPC ID, an Availability Zone, and an IPv4 CIDR block for the subnet. For this subnet, the ID of the newly created VPC is selected as the VPC ID. Additionally, `us-east-1b` is selected as the Availability Zone and `10.0.3.0/24` is selected as the IPv4 CIDR block.

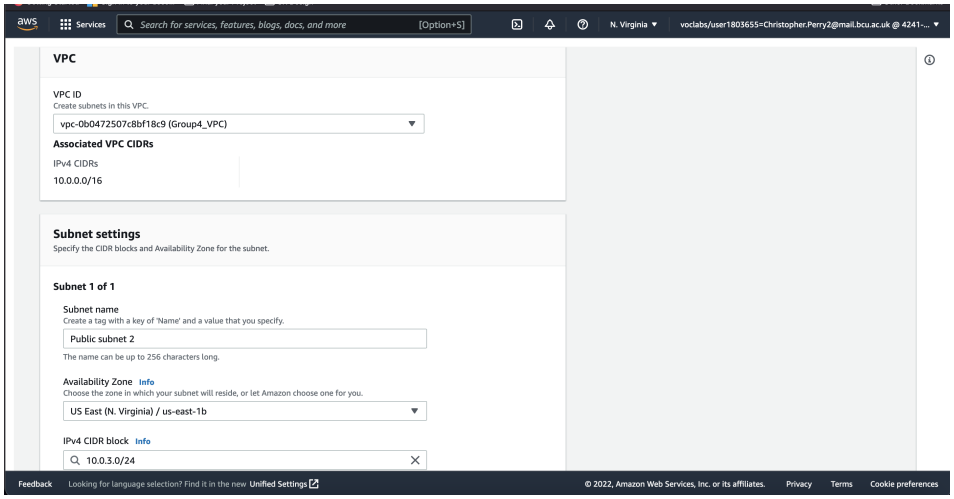


Figure 4.11: Creating a second public subnet.

Next, to create `Private subnet 2`, the ID of the newly created VPC is also selected as the VPC ID, `us-east-1b` is selected as the Availability Zone and `10.0.4.0/24` is selected as the IPv4 CIDR block.

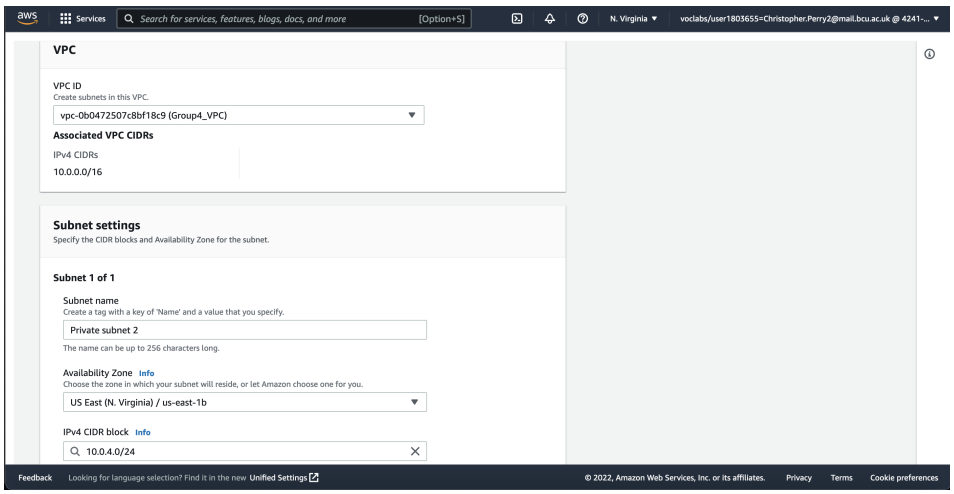


Figure 4.12: Creating a second private subnet.

Now the subnets can be filtered by Availability Zone and it can be seen that the two newly created subnets exist within `us-east-1b` (as well as an automatically generated default subnet, which will not be used).

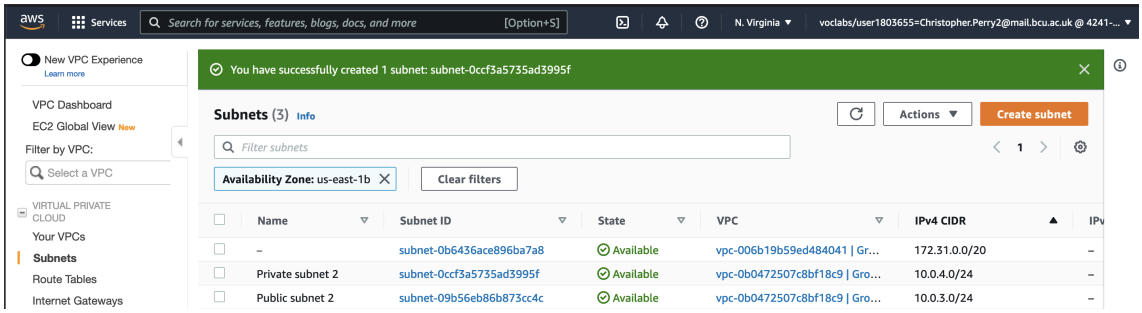


Figure 4.13: Viewing the newly created subnets.

After this, two routing tables must be created to link the two public subnets and the two private subnets. The Create Route Table page can be navigated to from the VPC Management Console. Two route tables are created, one at a time, called `Public route` and `Private route` with the previously created VPC.

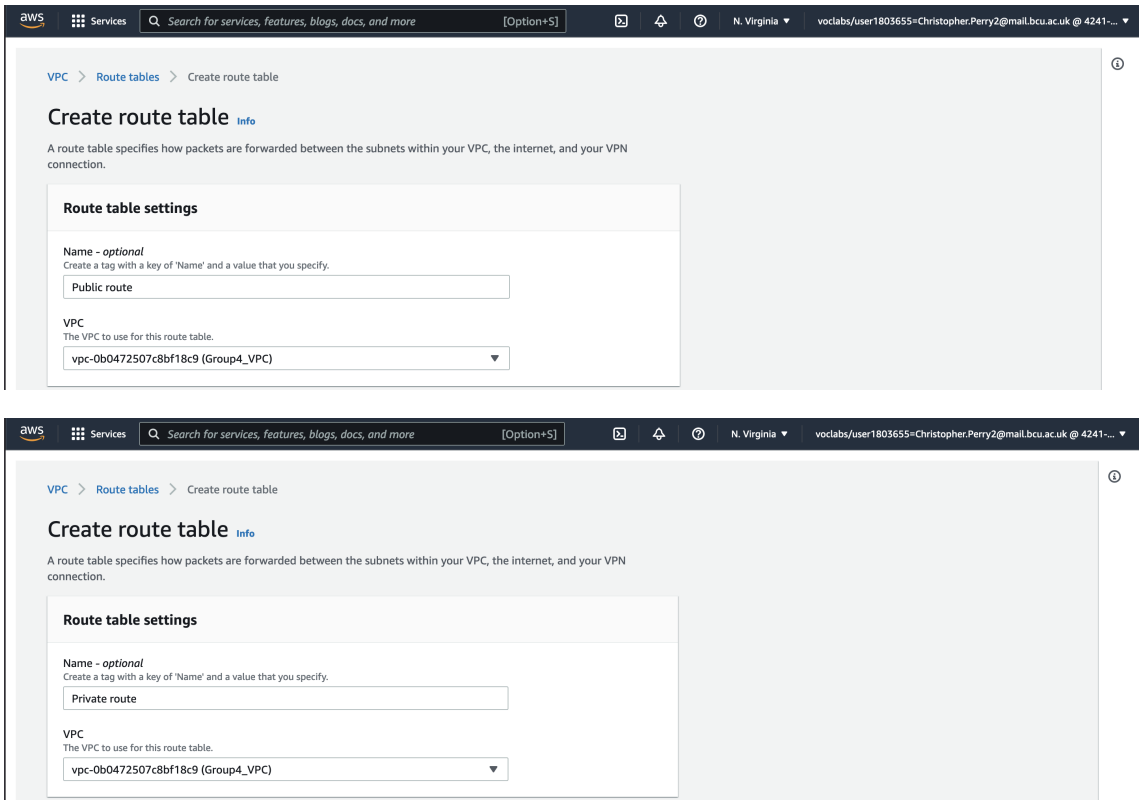


Figure 4.14: Creating a public route table and a private route table.

Once the two route tables have been created, the public and private subnets must be associated with the corresponding route table. From the Route Tables page, `Public route` is selected, and the **Edit subnet associations** button is clicked. On the Edit Subnet Associations page, the two public subnets are selected, and the **Save associations** button is clicked. This process is then repeated for `Private route`.

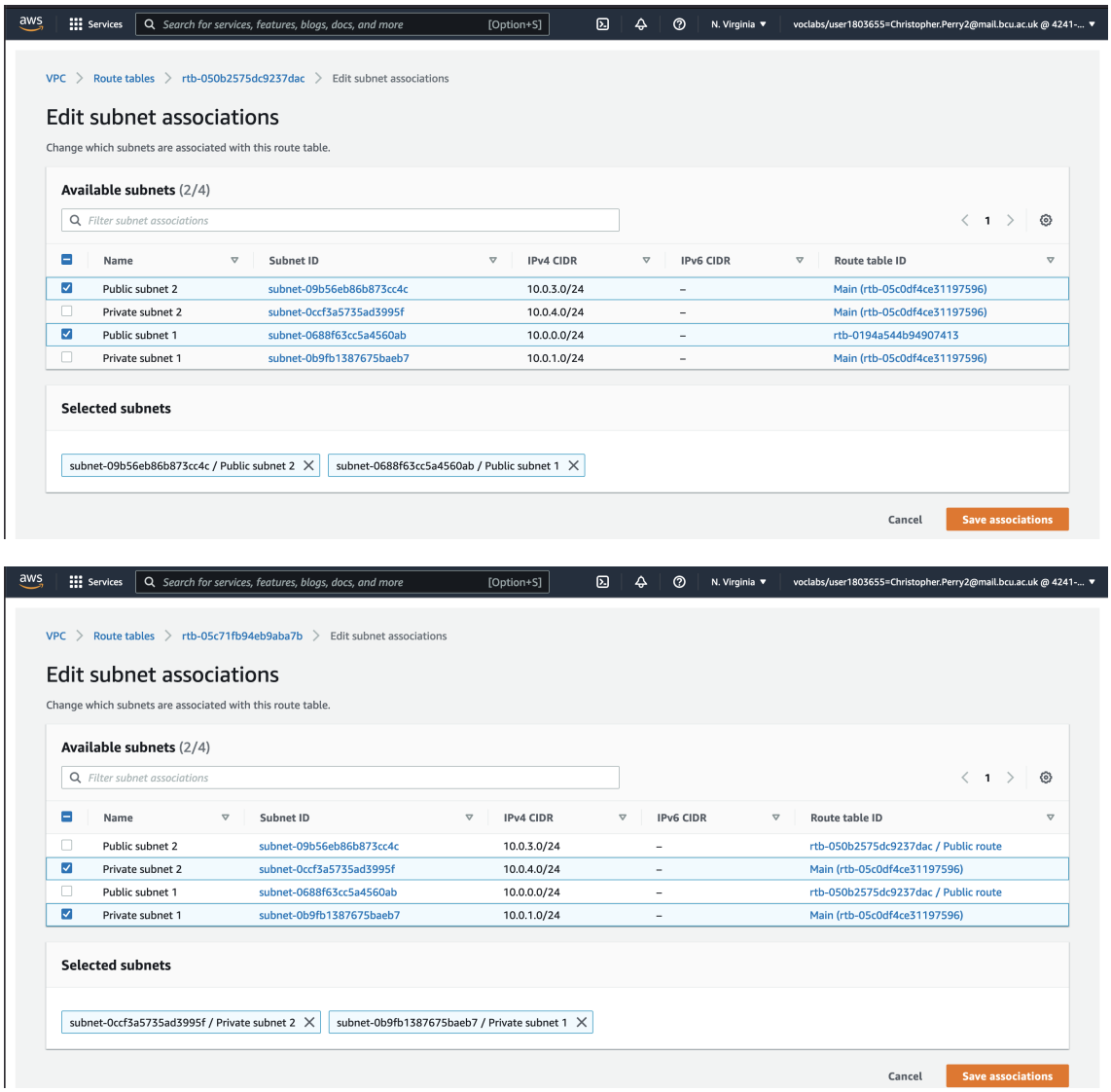


Figure 4.15: Associating the public and private subnets with the public and private route tables.

Lastly, the `Public route` must be updated to allow all public traffic that attempts to access the VPC to be routed via the internet gateway. This is done by selecting `Public route` and clicking the "Edit routes" Action. On this page, click the **Add route** button and enter a destination of `0.0.0.0/0` and, for the target, select the previously created internet gateway.

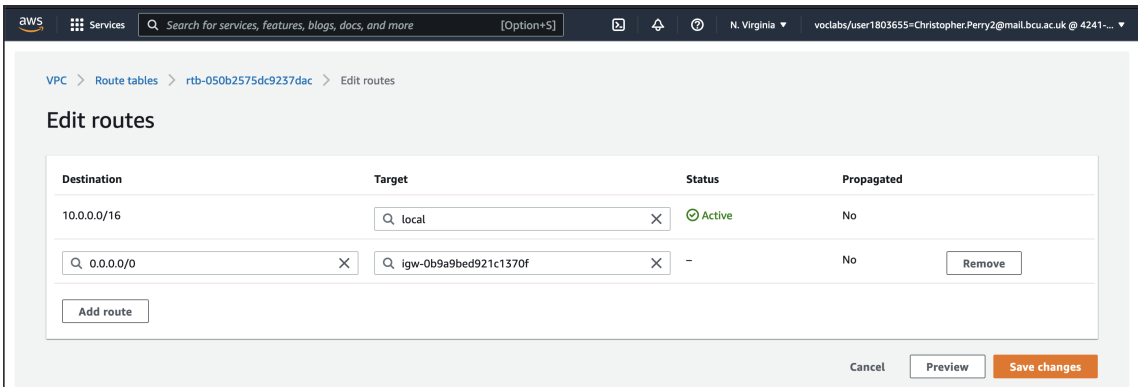


Figure 4.16: Editing the public route table.

Lastly, click the **Save changes** button. This will update the route for `Public route`.

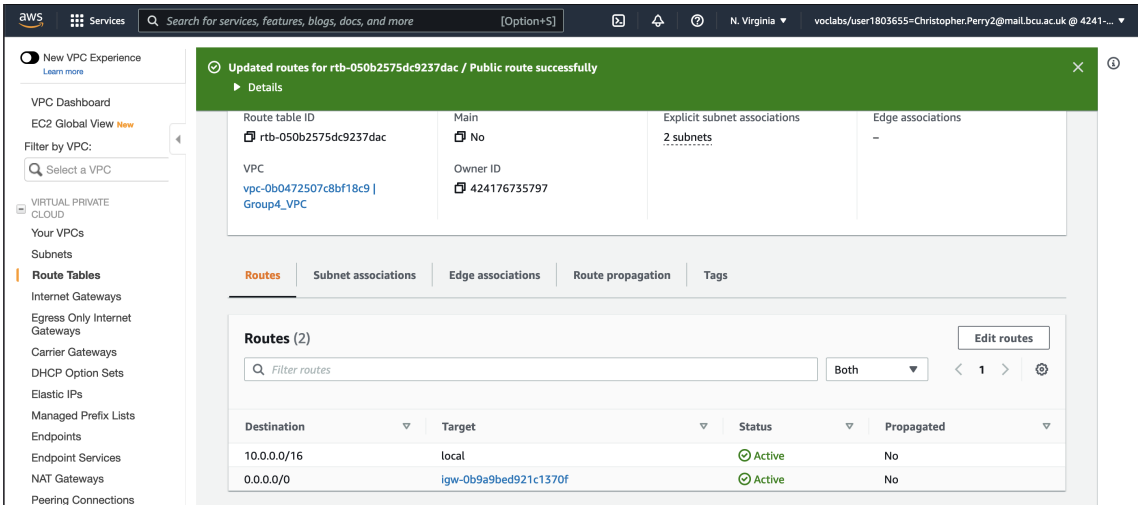


Figure 4.17: Updated public route table.

Chapter 5

Elastic Cloud Compute (EC2)

5.1 AWS Setup

After the VPC and subnets were configured, the initial deployment of the web app began with setting up EC2. This AWS service allows for scalable computing capacity through the use of a virtual computing environment hosted in the cloud (Amazon Web Services (AWS), 2022h). The web app will be stored on an EC2 instance of Amazon Linux, known as Amazon Machine Image (AMI), which will then be launched through a docker container stored on the app.

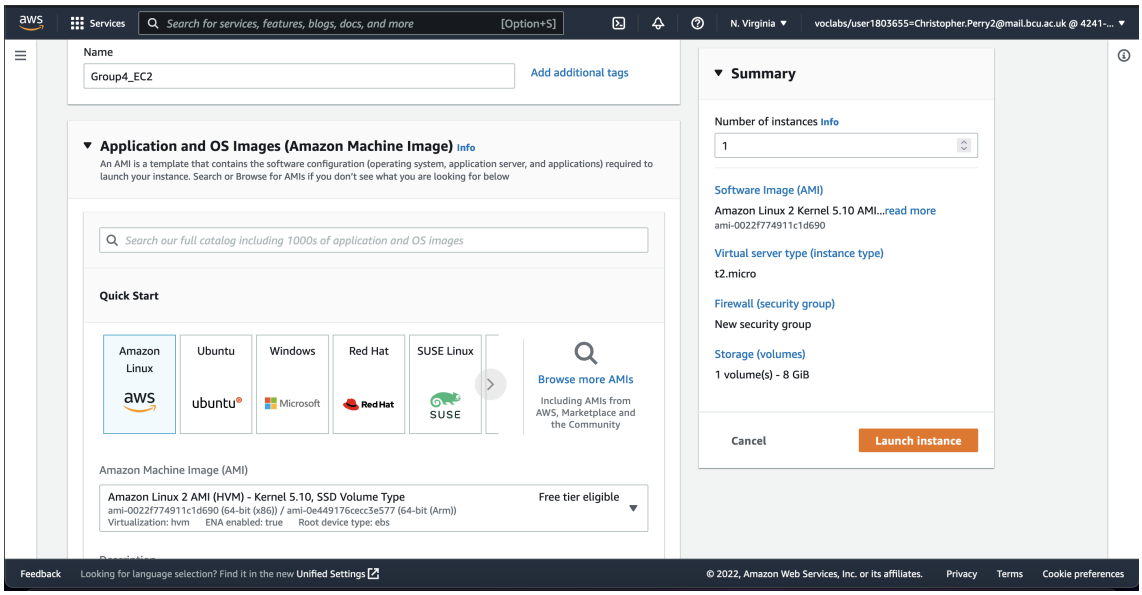


Figure 5.1: Selection of EC2 OS Image.

Figure 5.1 details the selection of the Operating System (OS) that will be used for the EC2 instance. The *Amazon Linux 2* AMI was selected, as it is already configured with Linux and does not need any more setup.

Now that an AMI has been chosen, the specific instance type that will be used within this AMI can be selected. It was decided that the instance type of *t2.micro* would be used, as it contains only 1GB of Random Access Memory (RAM), as seen in Figure 5.2.

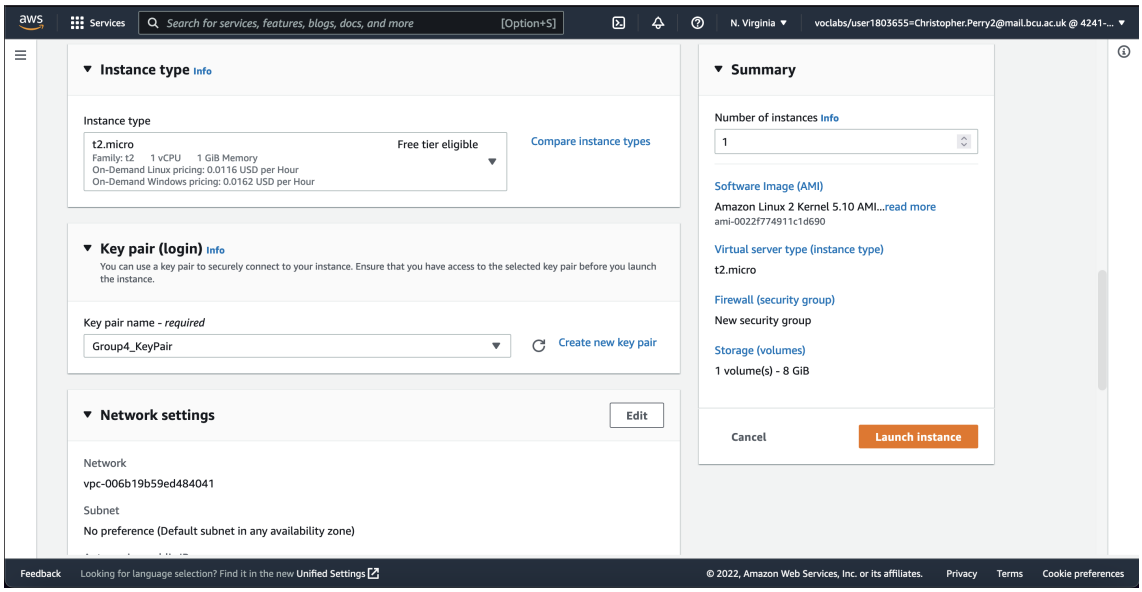


Figure 5.2: Selection of EC2 Instance.

A key pair will allow for the ability to sign in to the EC2 instance with a unique set of login credentials, heightening the security of the project.

The next stage of the setup process was to set up networking for the EC2 instance, in order for the web app to work with Docker to download relevant containers from Docker-Hub, which will allow a Laravel instance to be initialised, as discussed in Section 3. This process can be seen in Figure 5.3.

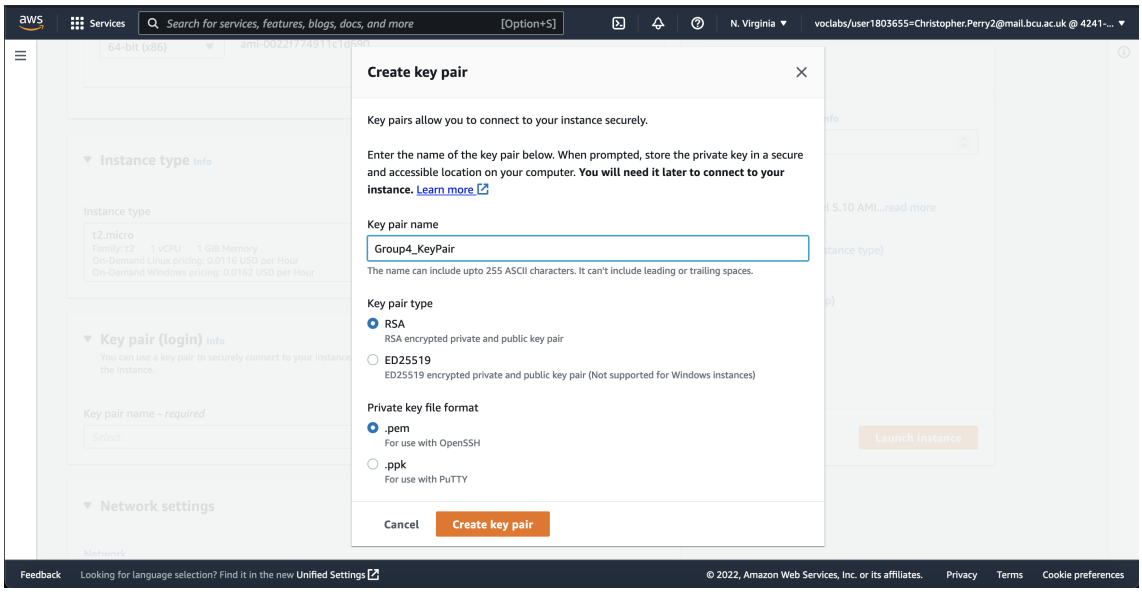


Figure 5.3: Selection of EC2 Keypair.

The instance is assigned the VPC created in Section 4, where it is assigned a subnet in the same availability zone of `us-east-1`.

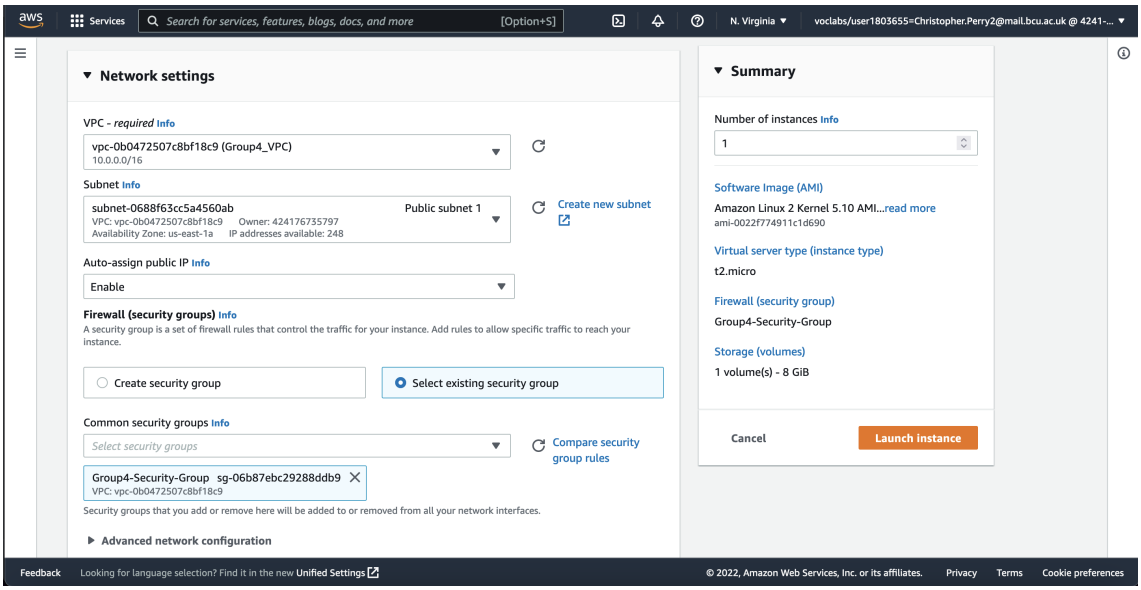


Figure 5.4: Selection of EC2 Networking options.

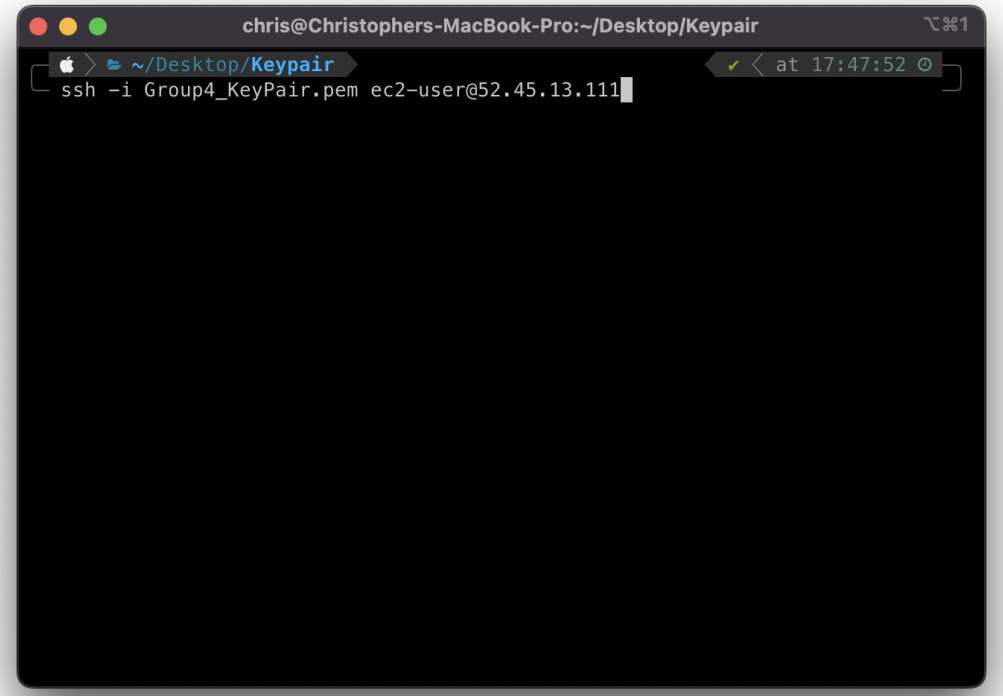


Figure 5.5: Generated EC2 Keypair in the `.pem` format.

This setup can be seen in Figure 5.4. An EC2 keypair is then generated in the `.pem` format.

This is enough to comfortably run the web app without any issues. Storage for the AMI was subsequently chosen. It was decided that 8GB of storage would be used, as this is enough to run the web app and still provide leftover storage for any system-critical tasks.

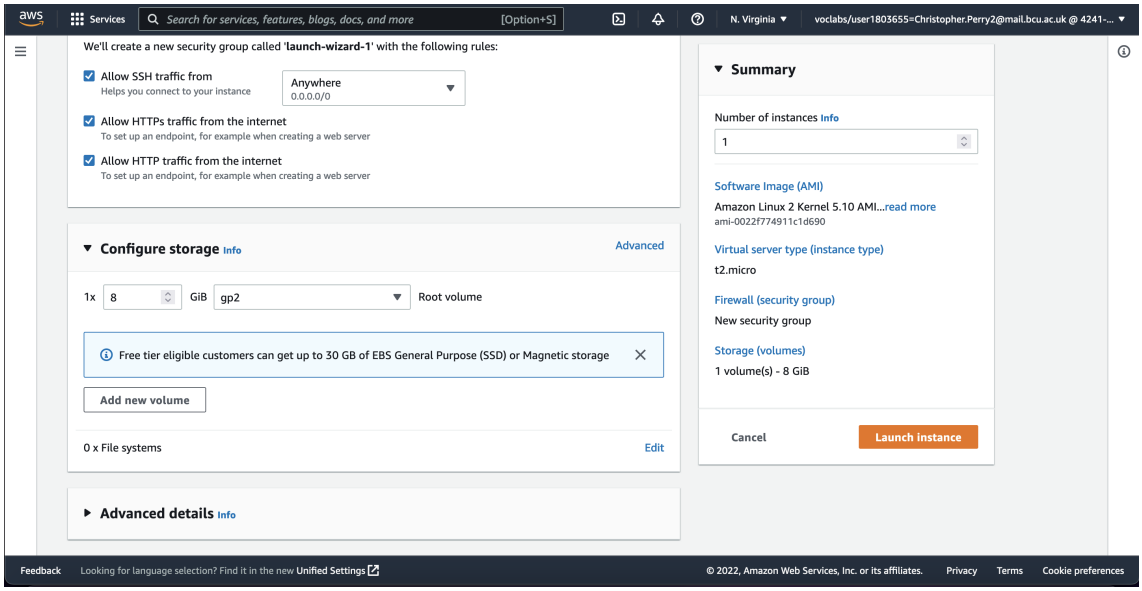


Figure 5.6: Selection of EC2 Storage Configuration.

The selection of these options can be found in Figure 5.6. In addition to this, the chosen options are eligible for "Free Tier", which means that it will use a limited amount of the $100 budget allocated for the project.

5.2 EC2 Login

The EC2 instance `group4-ec2` is now live, and the web app can be loaded onto it. The instance is first logged in through the use of the `ssh` command, followed by the `-i` argument to specify an identify file, which was generated earlier, and then the public IPv4 address of the EC2 instance. The command to log into the instance can seen being executed in Figure 5.7.

```
ssh -i ~/Desktop/Group4_KeyPair.pem ec2-user@52.45.13.111
```

Figure 5.7: SSH command to log into EC2 instance.

The EC2 instance that has been logged in to can be seen in Figure 5.8.

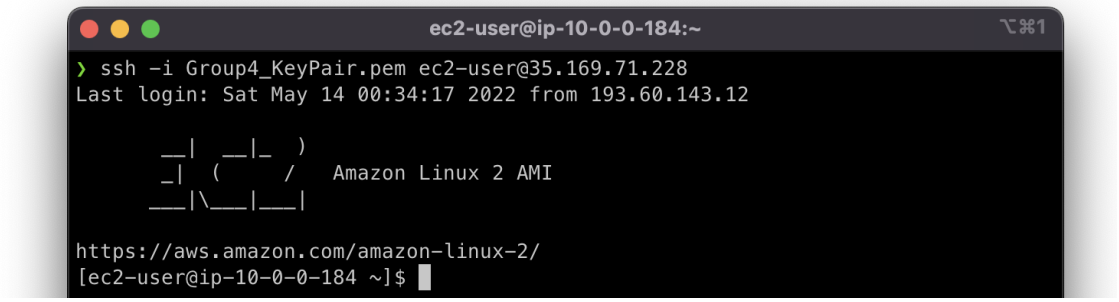


Figure 5.8: Logging into EC2 instance.

5.3 Package Setup

The web app is stored on GitHub, and the AMI does not come with GitHub by default. Git is subsequently installed via `yum install git`.

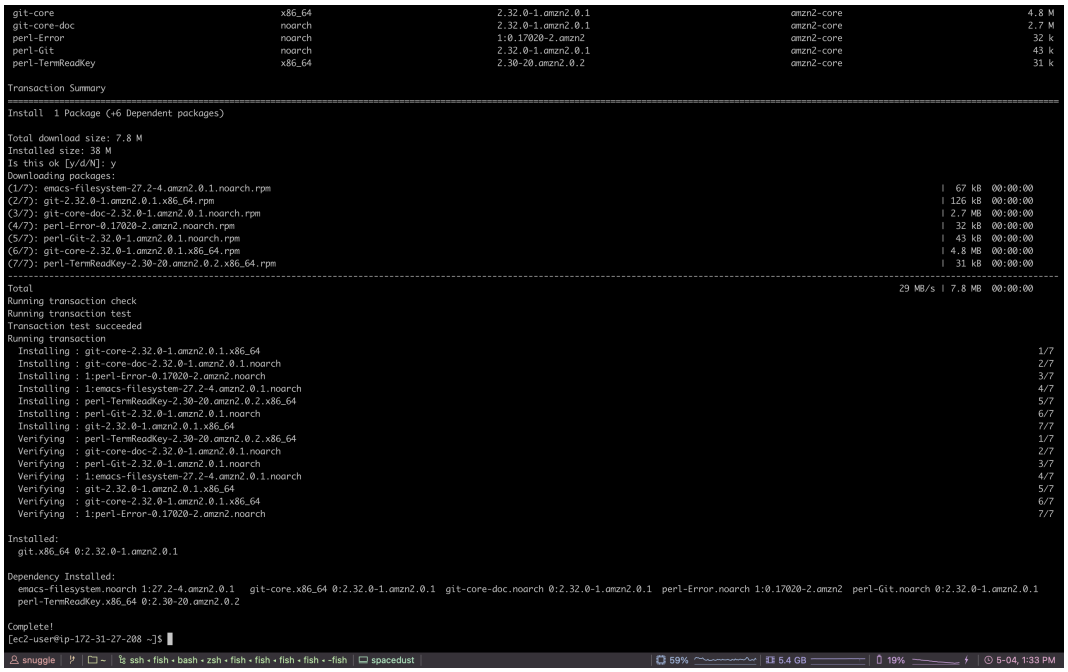


Figure 5.9: Installing Git.

The web app also requires docker, and `yum install docker` is executed to install Docker as a result.

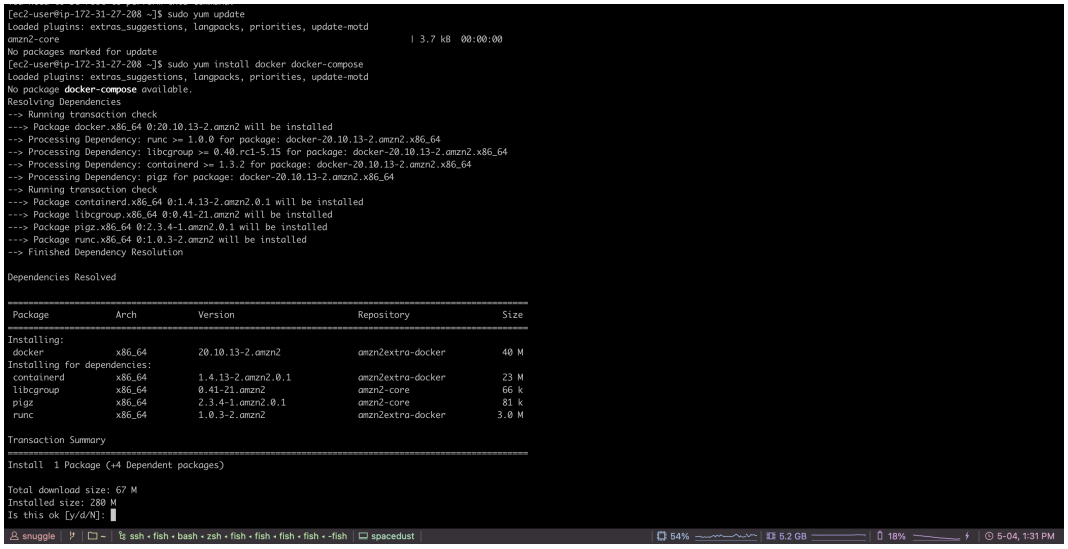


Figure 5.10: Installing Docker.

5.4 Web App Setup

The web app is firstly cloned from its repository via the `git clone` command, and a new `digital-ink` folder is made to store the contents.

```
[ec2-user@ip-172-31-27-208 ~]$ git clone https://github.com/ChrisP99/digital-ink
.git
-bash: git: command not found
[ec2-user@ip-172-31-27-208 ~]$ git clone https://github.com/ChrisP99/digital-ink
.git
Cloning into 'digital-ink'...
remote: Enumerating objects: 959, done.
remote: Counting objects: 100% (959/959), done.
remote: Compressing objects: 100% (324/324), done.
remote: Total 959 (delta 593), reused 959 (delta 593), pack-reused 0
Receiving objects: 100% (959/959), 3.32 MiB | 19.01 MiB/s, done.
Resolving deltas: 100% (593/593), done.
[ec2-user@ip-172-31-27-208 ~]$
```

Figure 5.11: Cloning the web app from Github.

The `cd` command is used to move into the `digital-ink` folder, and the web app is subsequently launched through the `docker-compose up -d` to launch the web app as a detached Docker container. Relevant containers that are required to be downloaded from the `dockerfile` are then pulled.

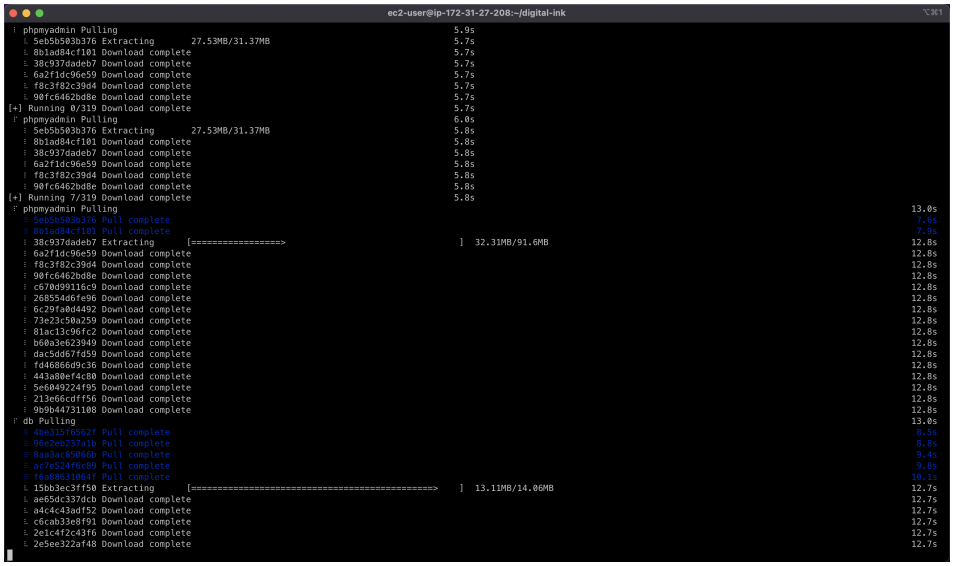


Figure 5.12: Containers required for the web app being pulled from Docker Hub.

The result of this command launches 3 containers:

1. `digital-ink`: An instance of the web app which uses a custom Laravel container.
2. `mysql`: An instance of a local database made in MySQL.
3. `phpmyadmin`: A way to locally manage the database through a UI.

At the minute, the web app is live through the `digital-ink` container, and is using a local version of MySQL as a database, stored within the `mysql`Docker container. The database has no tables, but can be populated through the use of Laravel. The container is firstly accessed through `docker exec app bash`, and the database is populated with tables with `php artisan migrate`. This then generates tables to store users and their stories.

Figure 5.13: Creation of tables through `php artisan migrate` command.

The subsequent output of this command can be found in Figure 5.13. When the website is accessed at the public IPv4 address at http://http://ec2-35-169-71-228.compute-1.amazonaws.com/, Digital Ink will now be shown.

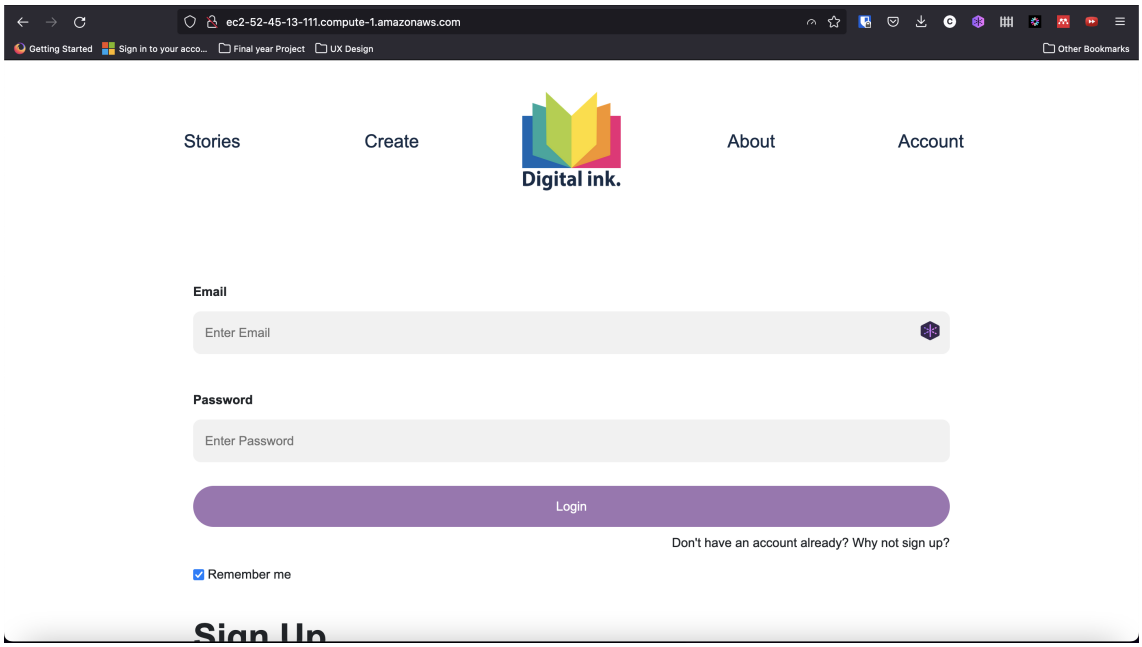


Figure 5.14: Digital Ink shown when accessed through the IPv4 address.

5.5 systemd Services

`systemd` is a service manager, which is a program that launches and monitors different services across the system. The digital-ink application is automatically started upon boot by the systemd process, which ensures that the application is started correctly and without errors. If any errors occur, systemd will log them in a tool known as `journalctl` and then follow a configuration file for instructions on how to handle those errors. This can range from simply restarting a program that has errored, to trying once again, or recording the failure and performing no action.

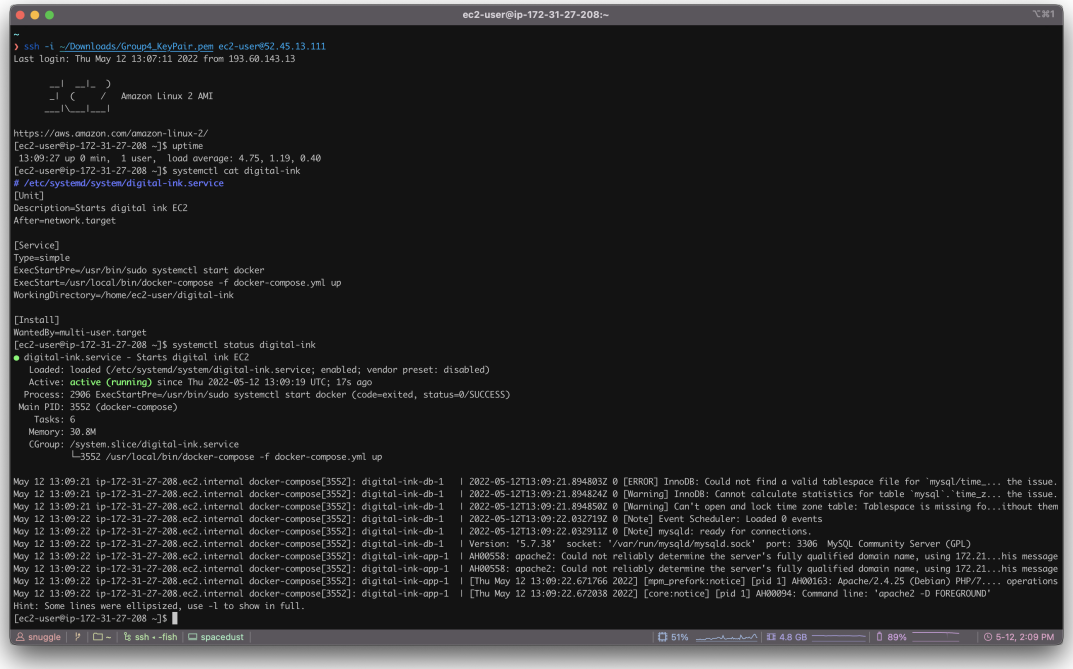


Figure 5.15: Creation of `systemd` Service.

Chapter 6

Simple Storage Service (S3)

AWS S3 is a form of cloud-based object storage. This is a style of network filesystem that treats each file as a separate object with a unique ID, which allows each object to be served individually over a network with a single URL, which can also be enhanced with a content delivery network (Amazon Web Services (AWS), 2022e).

Each S3 instance is seperated into logical containers known as buckets. Each S3 bucket can have its own credentials, its own endpoint and other permissions and configuration. Object storage is particularly useful when creating an application that scales as you are billed per unit of storage that you use, and in theory are able to use infinite storage as your application demands. The only limitation with object storage is how much you are able to pay for. Each object is automatically replicated across many nodes, providing data redundancy against multiple different availability zones.

S3 is perhaps the most popular AWS service and used by many SaaS applications across the internet, for instance Instagram, Facebook, Discord and Twitter are all known for using S3 or S3-style storage. The advent of object storage has created an almost de-facto standard, which has lead for the creation of many 'S3-compatible' or 'S3-like' competitor solutions, such as those run by Google Cloud and Microsoft Azure.

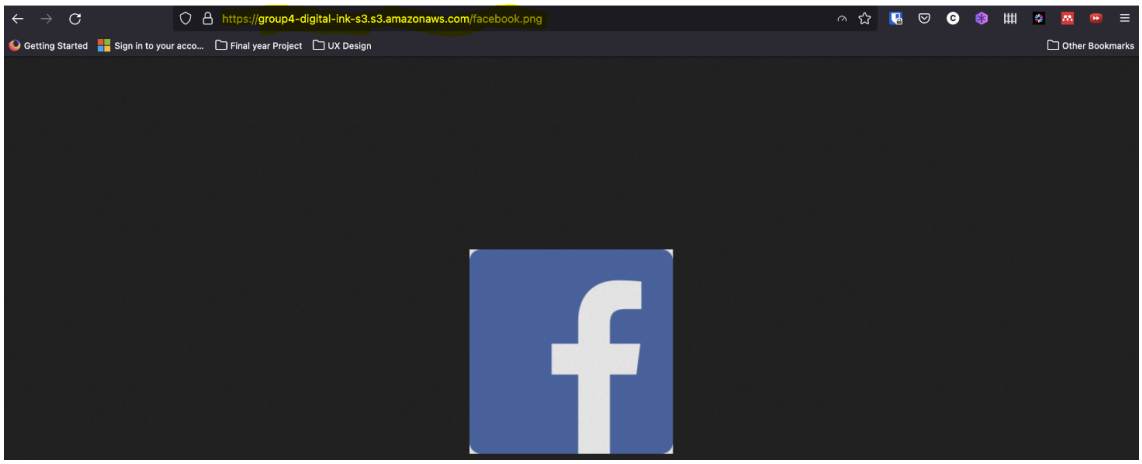


Figure 6.1: Image access through S3.

Chapter 7

CloudFront

CloudFront is a CDN which allows for the distribution of static and dynamic web images with more efficiency and availability. An international network of data centres, referred to as edge locations, are used to deliver content for CloudFront (Amazon Web Services (AWS), 2022g). An edge location which offers the lowest latency or delay in serving these files will be used. This will involve the creation of a CloudFront Distribution.

Firstly, an origin is chosen which is the S3 bucket created in Section 6. It is then given a name of `group4-digital-ink.s3.us-east-1.amazonaws.com`. The "Yes use OAI" option is selected, in order to restrict bucket access to only CloudFront. The "Bucket Policy" option is set to "Yes" to automatically update permissions on the bucket to allow read access for the OAI. These options can be seen configured in Figures 7.1 and 7.2.

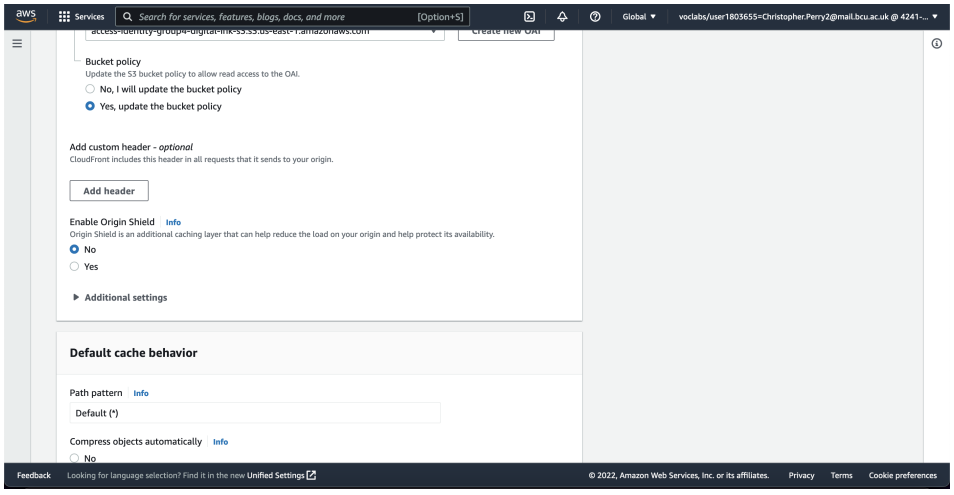


Figure 7.1: Applying CloudFront bucket policy.

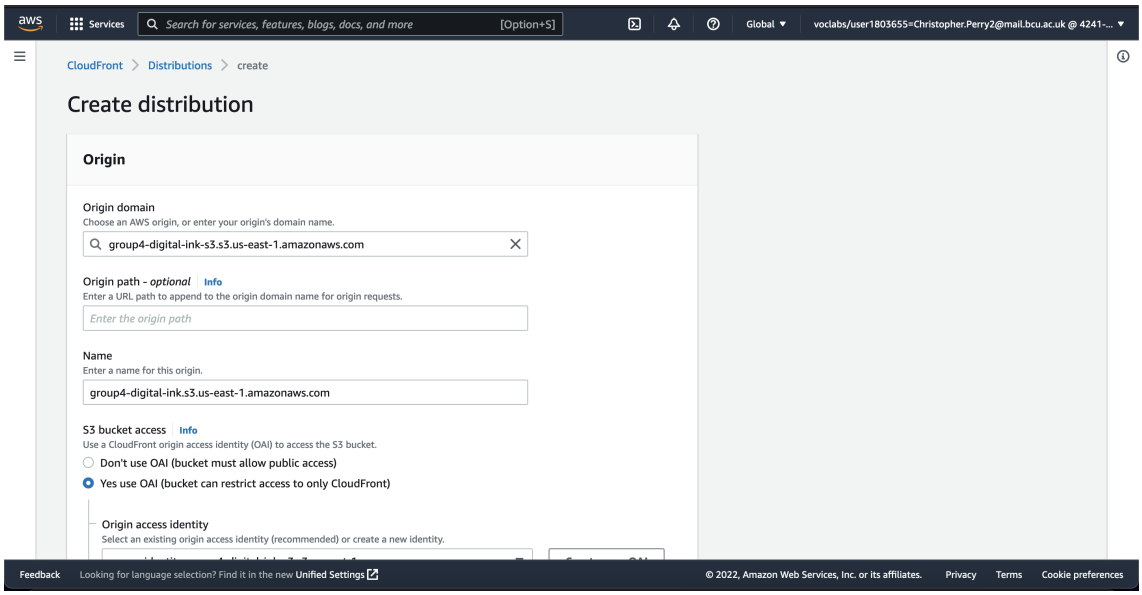


Figure 7.2: Applying CloudFront origins to S3 bucket.

Permissions are then applied to the CloudFront distribution itself. Objects are set to be compressed automatically to save space, and for viewing images, permissions are set to be in both HTTP and HTTPS, and to only allow `GET` and `HEAD`, so images can only be viewed.

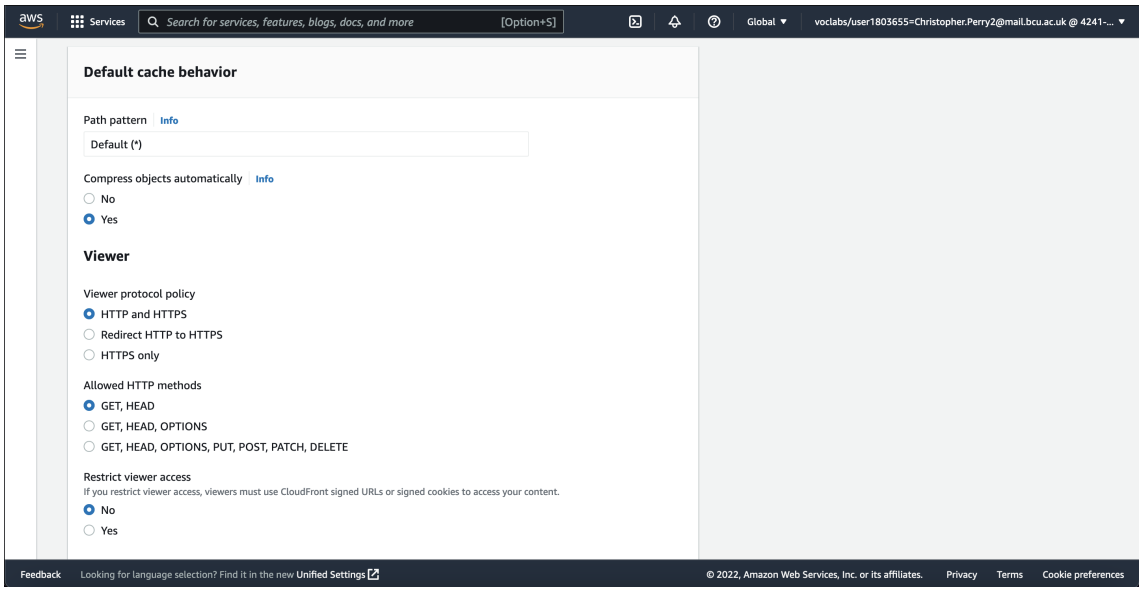


Figure 7.3: Applying CloudFront Distribution Permissions.

A cache policy and origin request policy is subsequently applied to the aforementioned permissions. The "CachingOptimized" policy is applied for compression. Origin Request policy is set to "CORS-S3Origin" in order for CORS to be automatically set up for the S3 bucket, Any GET requests are then compatible in the response headers through "SimpleCORS".

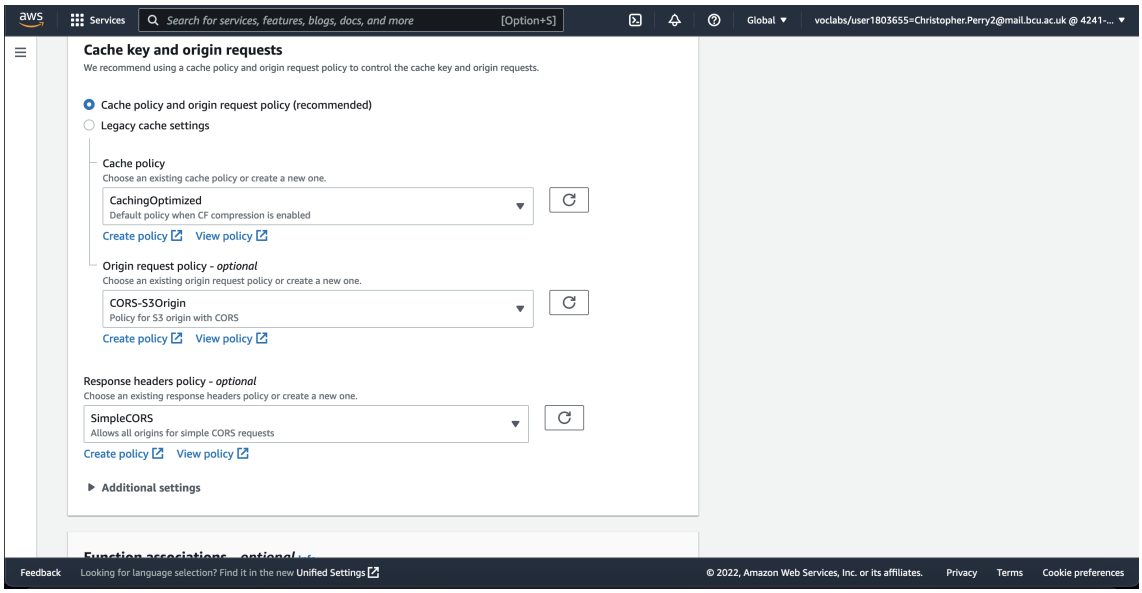


Figure 7.4: Applying CloudFront Cache Keys and Origin Requests.

No function associations are set due to the lack of CloudFront functions and Lambdas.

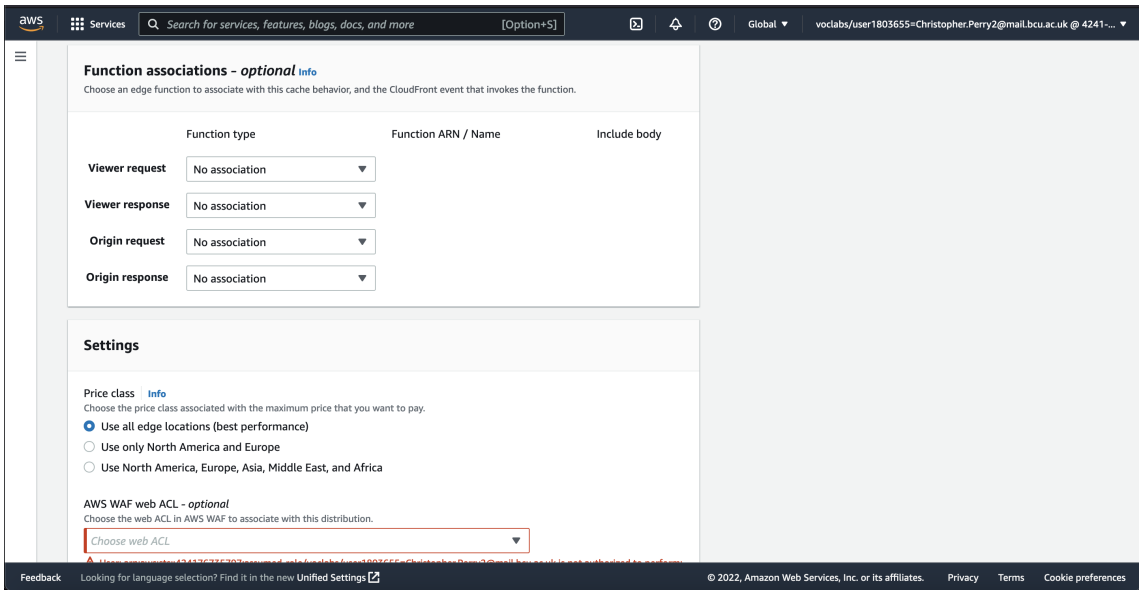


Figure 7.5: Function Association.

The settings for the CloudFront Distribution can now be set. In order to ensure high availability, the option "Use all edge locations" is selected, so that images can be distributed to the user from the closest edge location in relation to their IP address. No ACL can be added to the WAF due to permissions issues, and a SSL Certificate cannot be set as the web app is not being served from the internet. For logging purposes "Standard Logging" is set to "On", and logs are saved to the same S3 bucket. Cookie logging is enabled, and IPv6 is set to "On", to allow for more IP addresses to access the CloudFront Distributed content. These settings can be found in Figure 7.6.

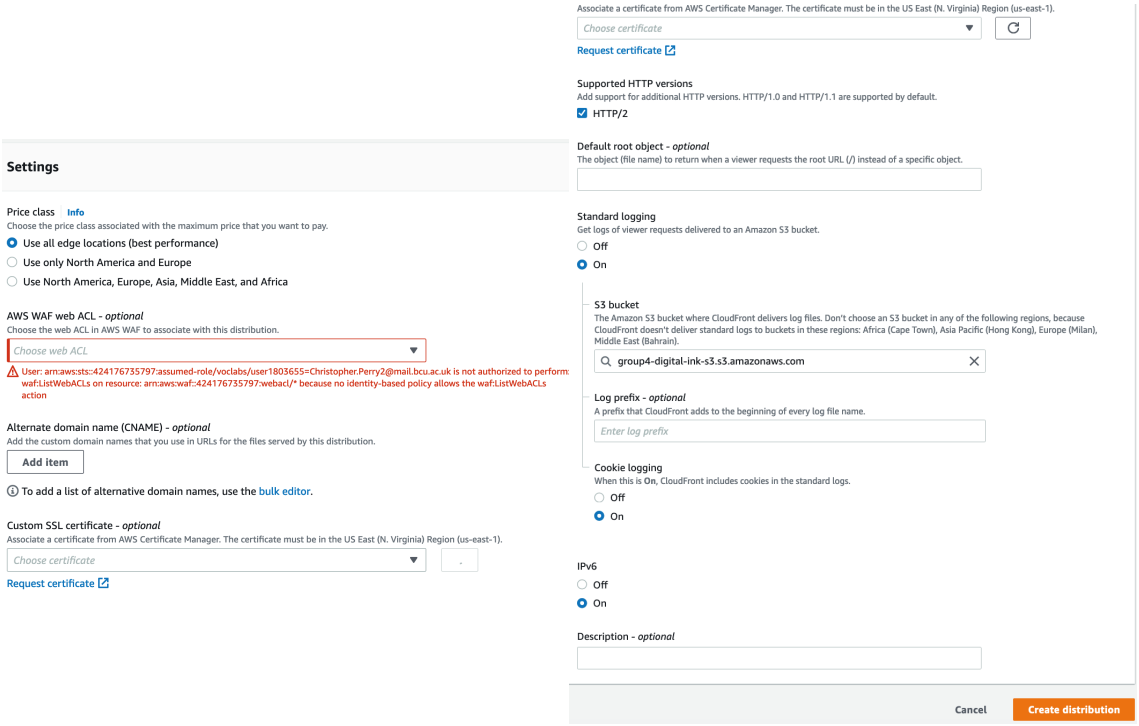


Figure 7.6: Applying CloudFront settings.

A CloudFront Distribution has now been created, and can be found in Figure 7.7.

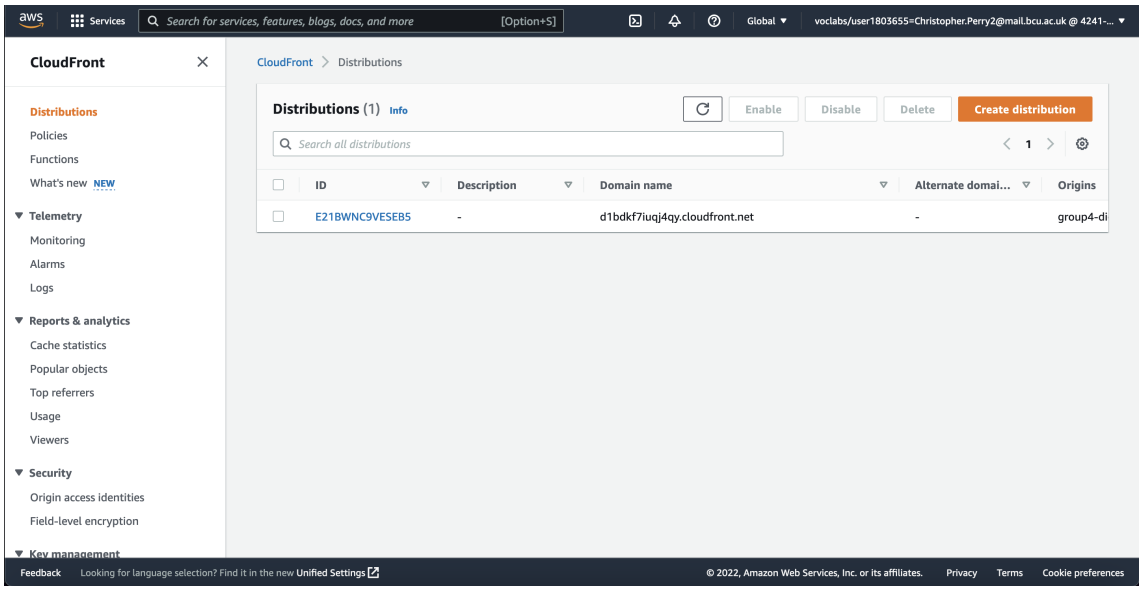


Figure 7.7: Created CloudFront Distribution.

Finally, the images contained within the webserver were changed from their local location to their relevant CloudFront locations. This change can be seen in Figures 7.8 and 7.9.

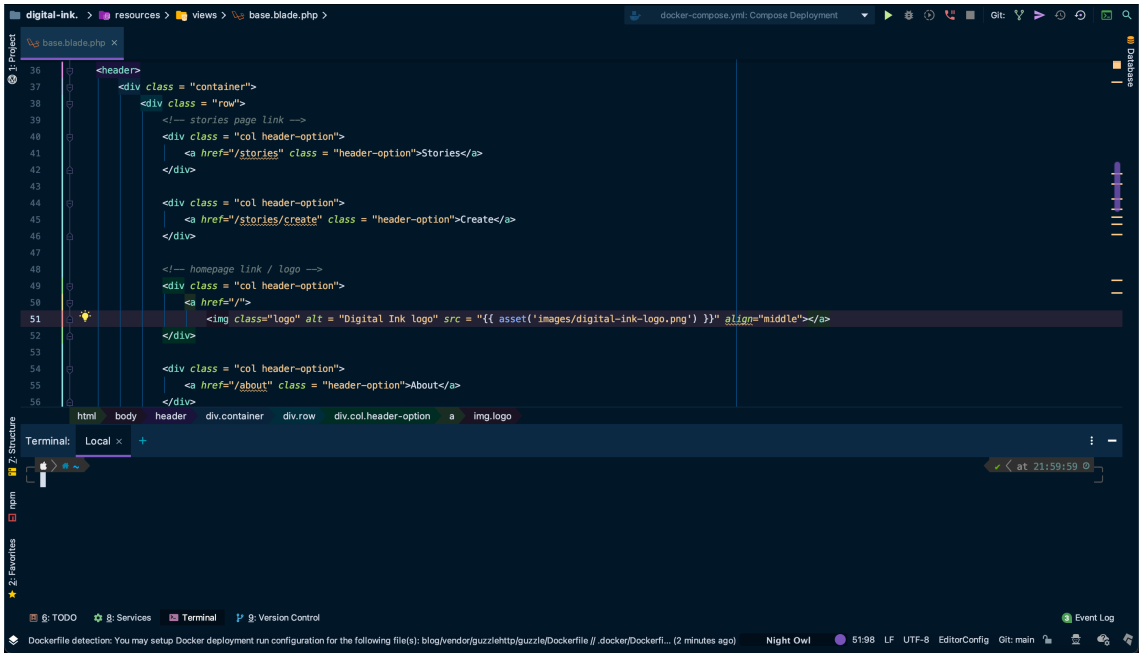


Figure 7.8: Image location before CloudFront.

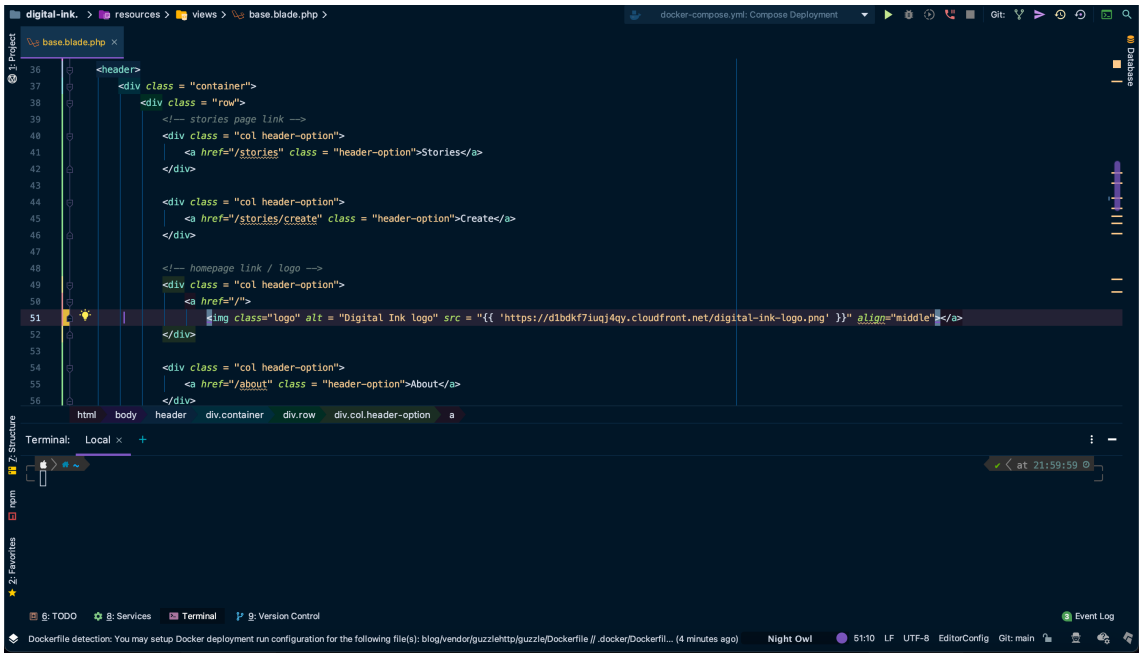


Figure 7.9: Image location after CloudFront.

As seen in the URL shown within Figure 7.10, the Digital Ink logo image is now hosted on CloudFront.

Figure 7.10: Image location after CloudFront.

Chapter 8

CloudWatch

Amazon CloudWatch is a monitoring and observation tool which can provide developers with insights to monitor applications, react to performance changes, and optimise resource consumption. CloudWatch collects monitoring and operational data about the application as logs and metrics to provide a complete overview of operation health, resource consumption, and the services currently running on AWS. This is useful for any cloud-based application as it allows developers to set alarms, visualise metric logs, troubleshoot errors and, most importantly, identify and correct anomalous behaviour (Amazon Web Services (AWS), 2022a).

An example of CloudWatch usage would be an alarm set to alert developers when a specified resource consumption level is over a certain threshold. For the purposes of the project, multiple CloudWatch alarms will be set up which will allow monitoring for budgeting and performance across the various AWS services used.

The first alarm which will be set up is a network alarm. Through selecting the `NetworkPacketsOut` metric on the `Group4_EC2` instance, the packets which are being outputted can be monitored.

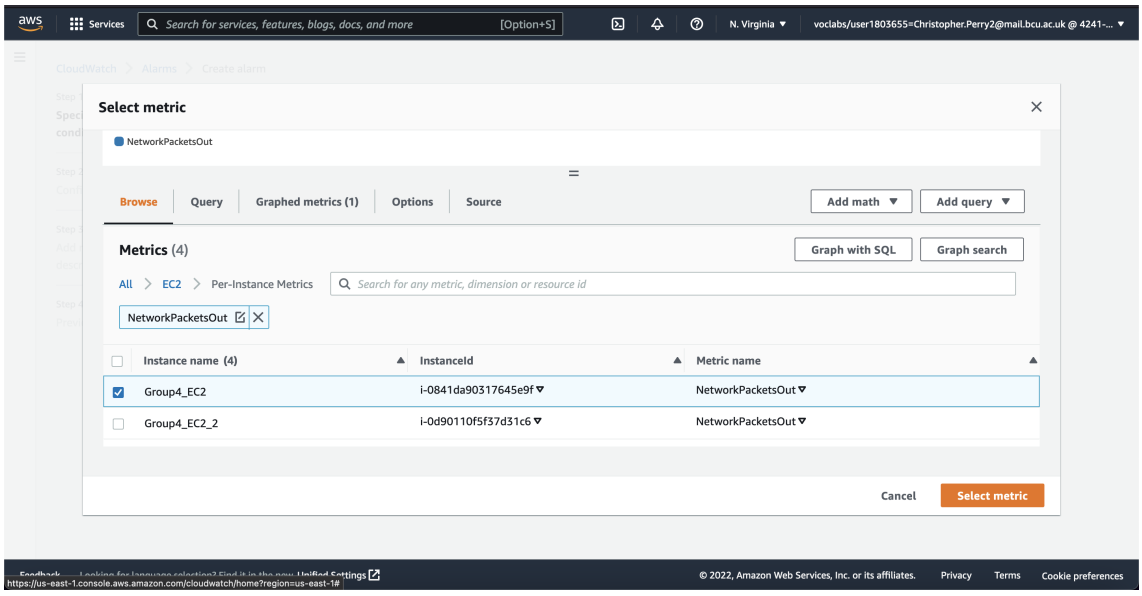


Figure 8.1: Selection of CloudWatch metric for EC2 instance.

The metric will be configured to alarm in the event that there is less than 5 packets of data sent per day from the instance. As the instance currently outputs nearly, 30,000 packets per day, this will be useful to check if the web app has failed. Figure 8.2 details the alarm being configured to activate an alarm when the sum of packets sent is less than 5 per day.

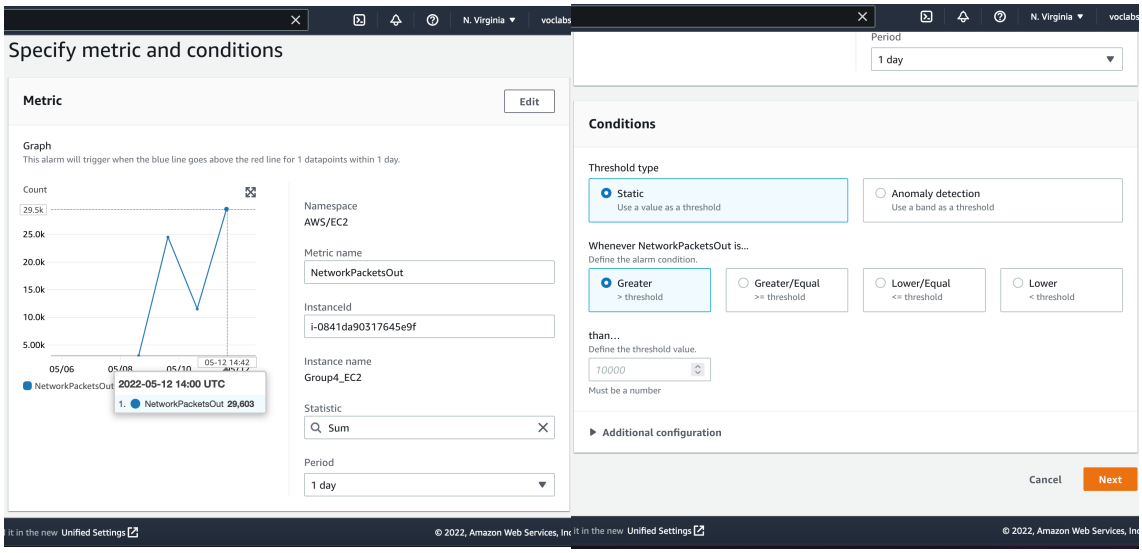


Figure 8.2: Configuration of NetworkPacketsOut Metric.

In addition to the initial configuration of these CloudWatch alarms, SNS topics will also be set up so that every member of the group will be emailed in the event that the alarms activate. Figure 8.3 details this process.

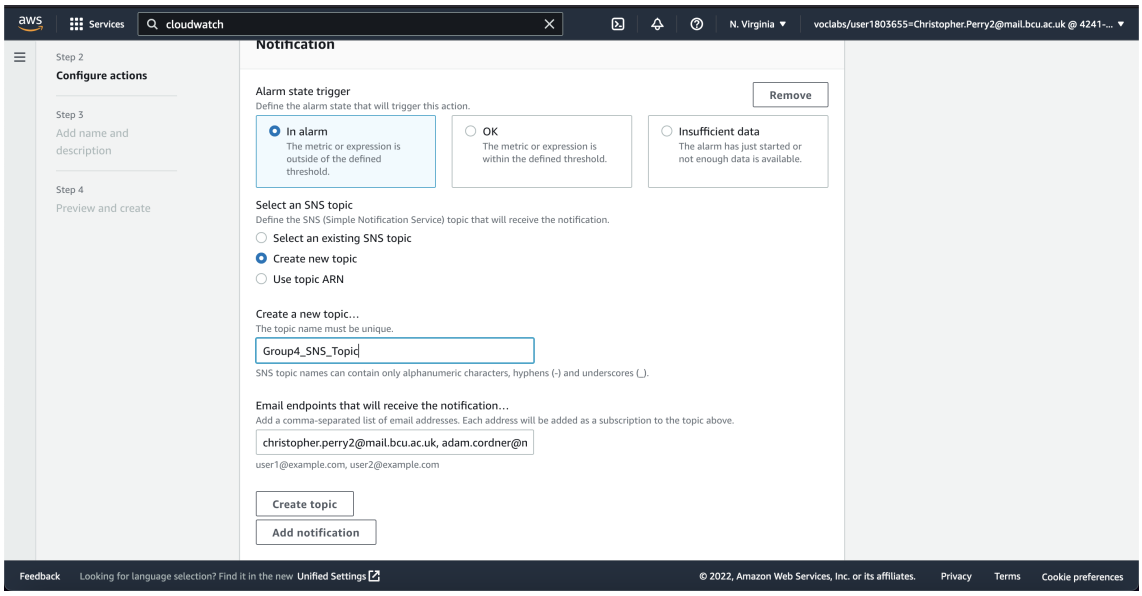


Figure 8.3: Configuration of SNS Topic for email alerts on alarm activation.

An email was then sent to all group members upon completion of this form, and the SNS topic was subsequently subscribed to, as shown in Figures 8.4 and 8.5.

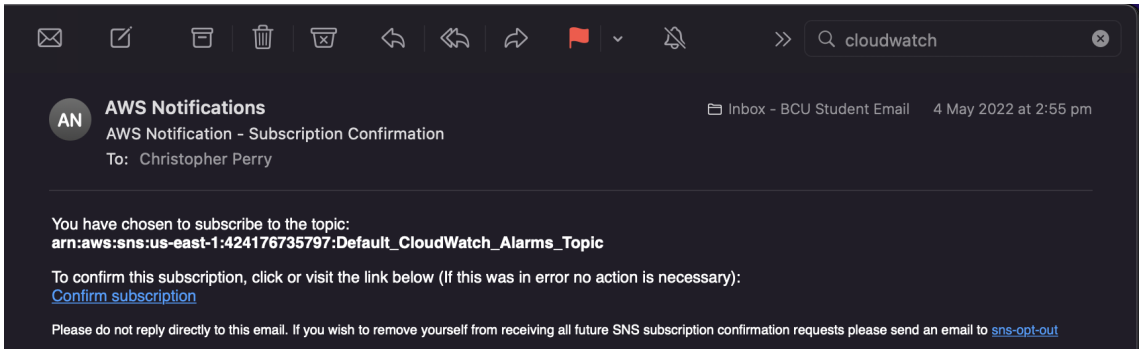


Figure 8.4: CloudWatch SNS Topic Email.

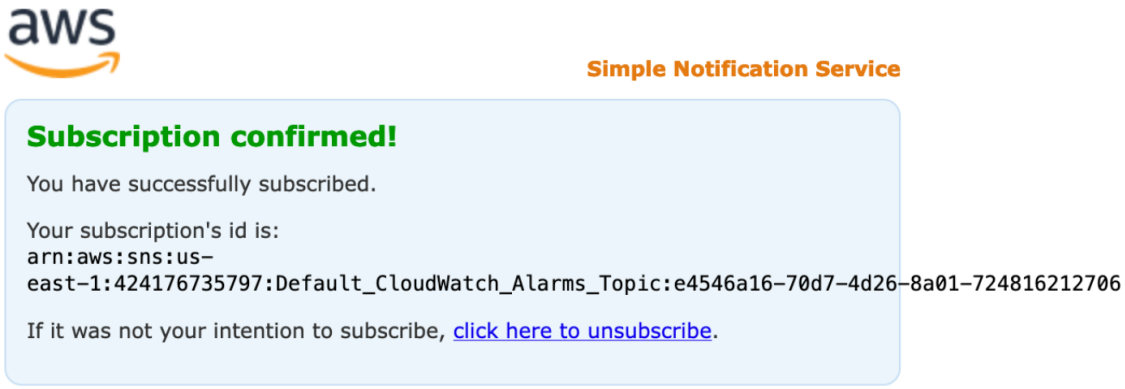


Figure 8.5: Successful subscription to SNS topic.

The EC2 instance has also been configured to restart when this alarm activates. This is done by automatically re-running the scripts used to start the web app on the instance starting up again. This process can be seen in Figure 8.6.

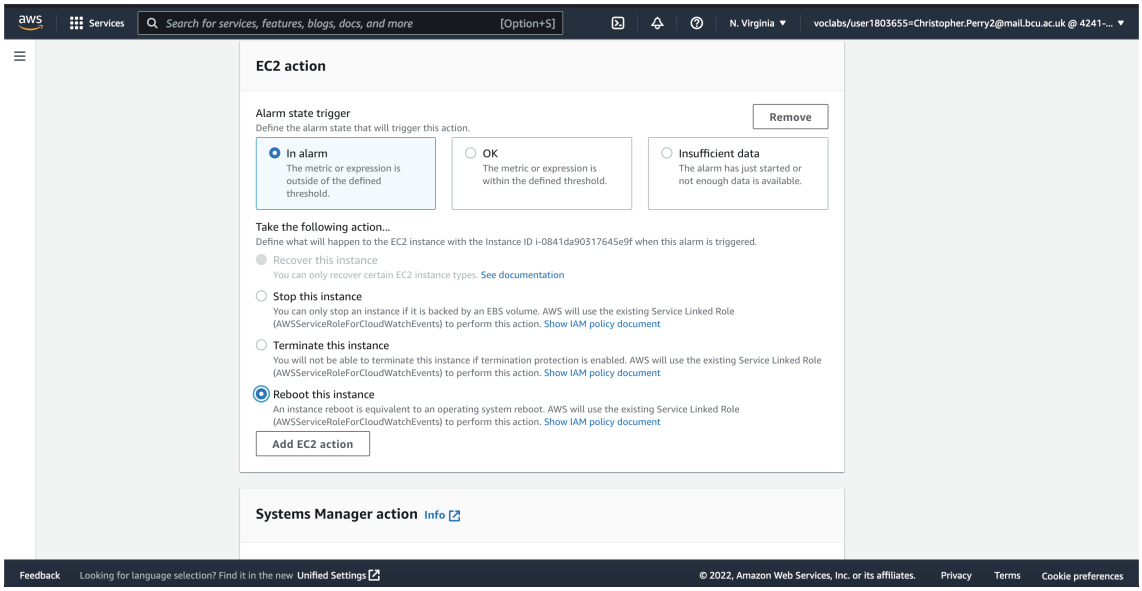


Figure 8.6: Configuration for EC2 instance to reboot on alarm activation.

A brief description of the CloudWatch alarm is then added. This can be seen in Figure 8.7.

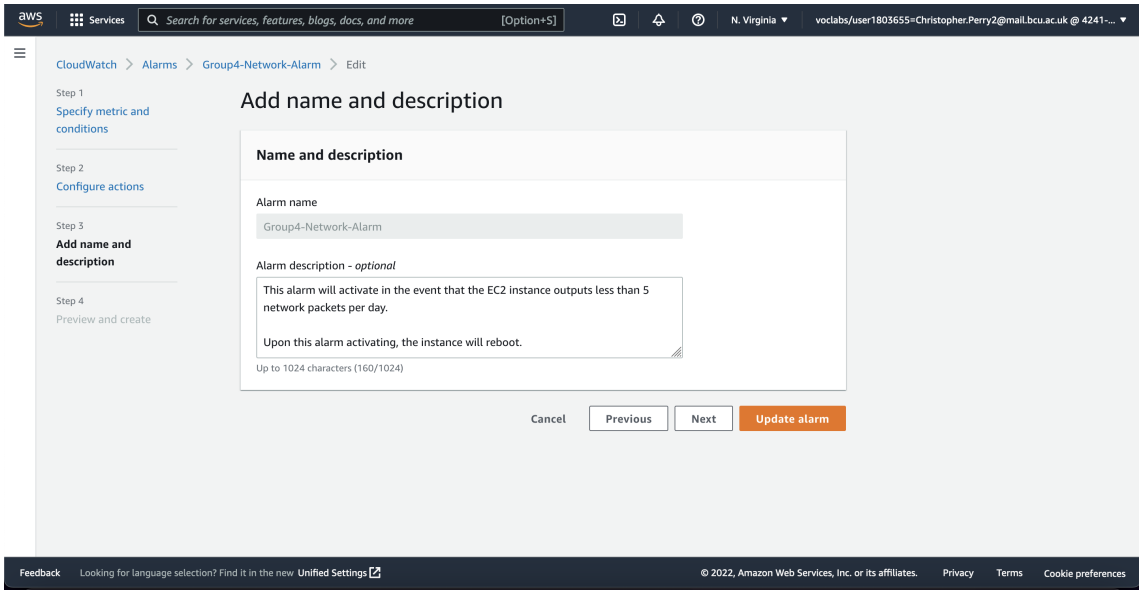


Figure 8.7: CloudWatch alarm description.

The alarm has now been set up, and can be seen in the CloudWatch Management Dashboard in Figure 8.8.

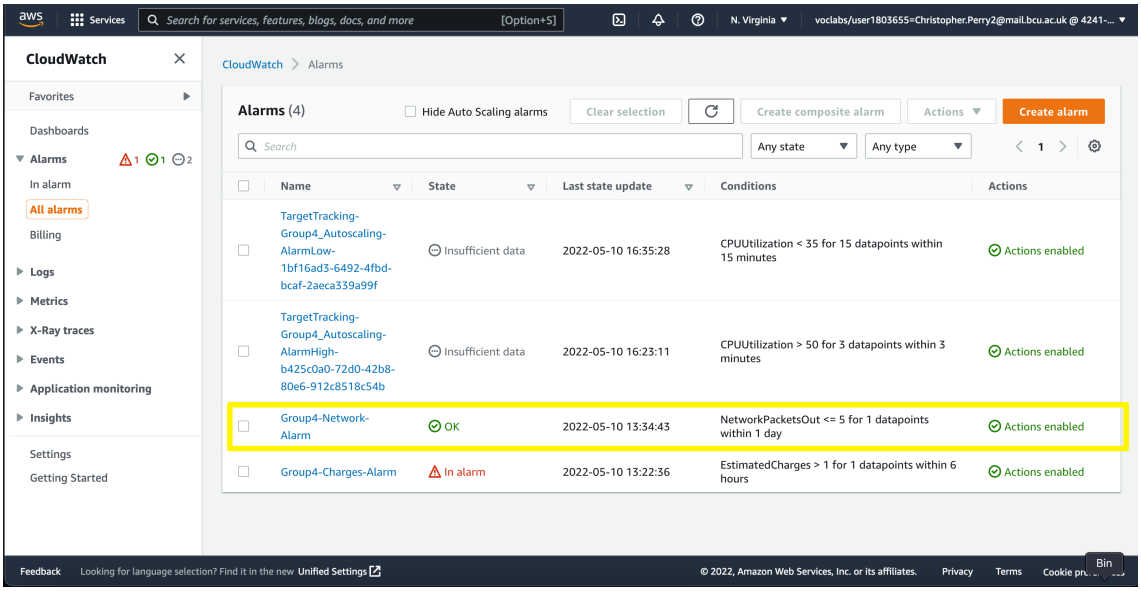


Figure 8.8: CloudWatch network alarm in dashboard.

In addition to the created network alarm, a charges alarm will also be set up. A similar process is followed whereby the metric of `EstimatedCharges` is applied to all `AmazonEC2` instances, as detailed in Figure 8.9.

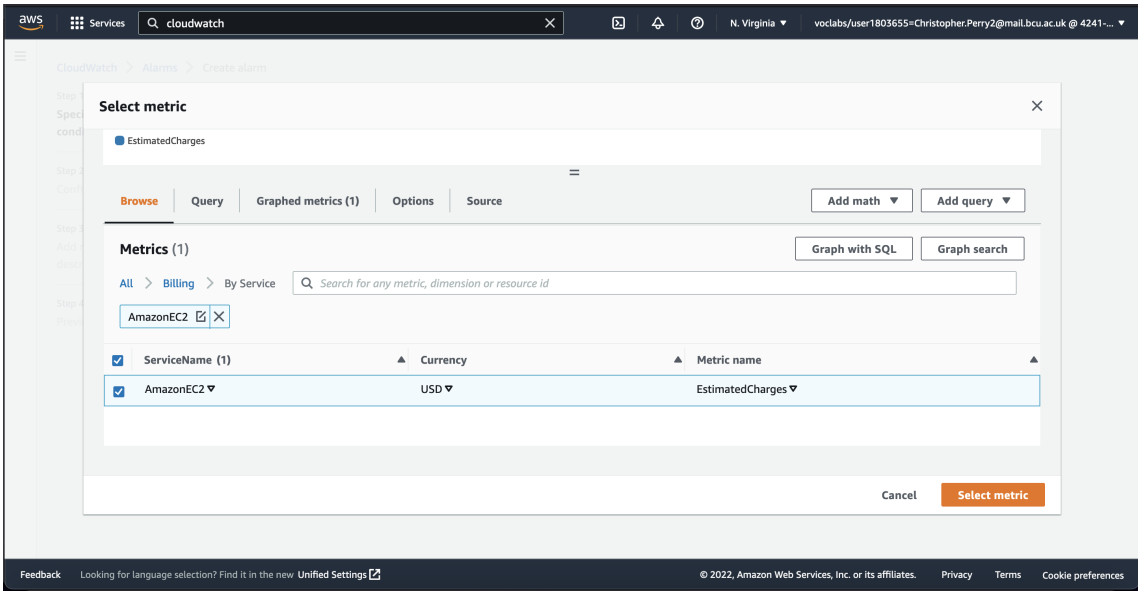


Figure 8.9: Selection of `EstimatedCharges` CloudWatch metric.

This metric will be configured to alarm in the event that the EC2 instance uses more than $15 every 6 hours. This process can be seen in Figure 8.10.

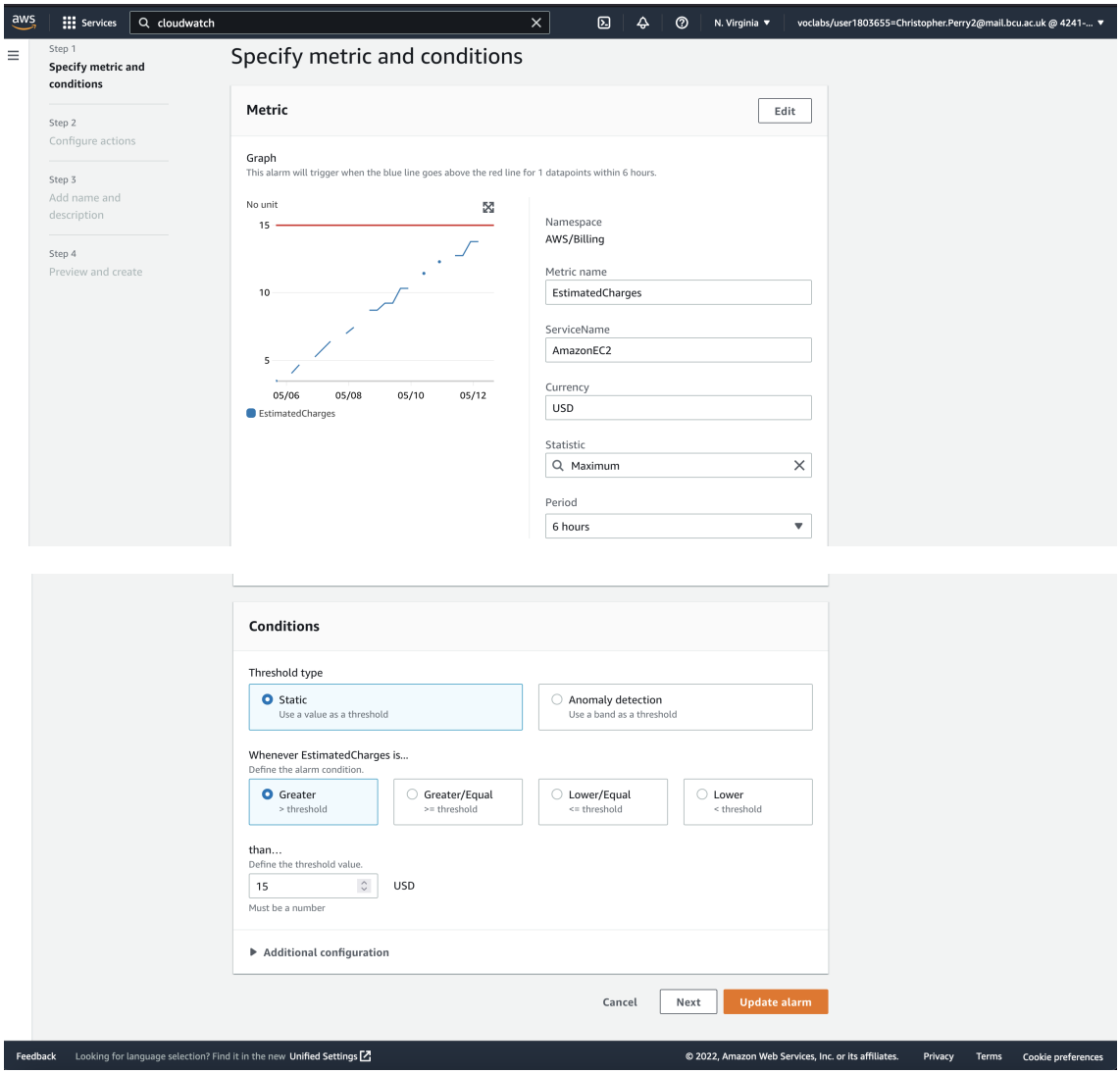


Figure 8.10: Configuration of CloudWatch alarm.

In addition to this, alerts will be sent to the same SNS topic configured earlier. This can be seen in Figure 8.11.

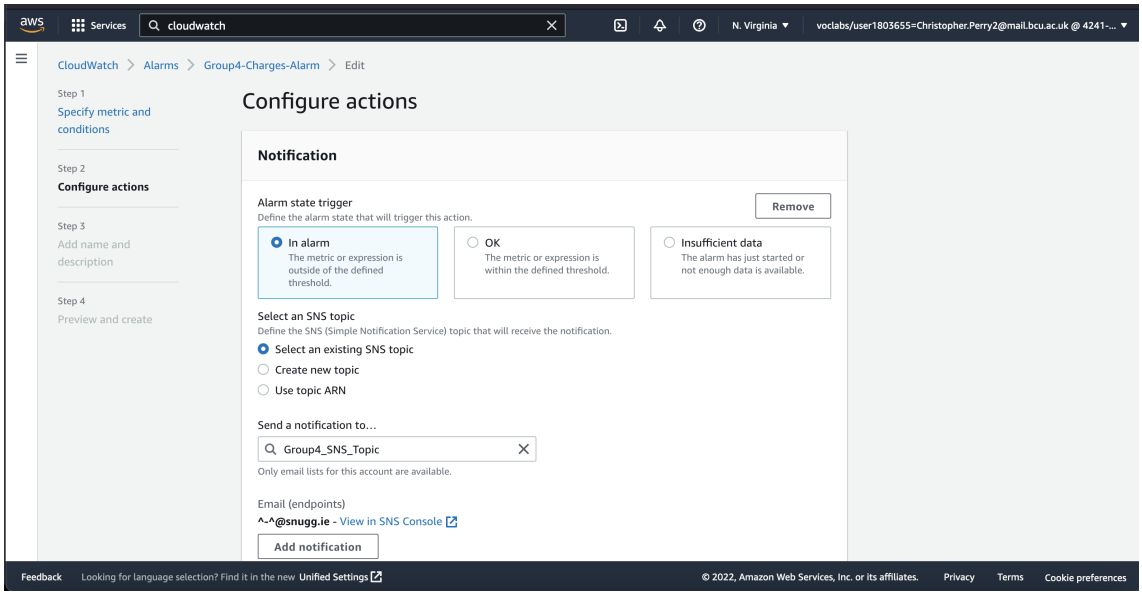


Figure 8.11: Setting CloudWatch charges SNS topic

A brief description was added to the alarm, as seen in Figure 8.12.

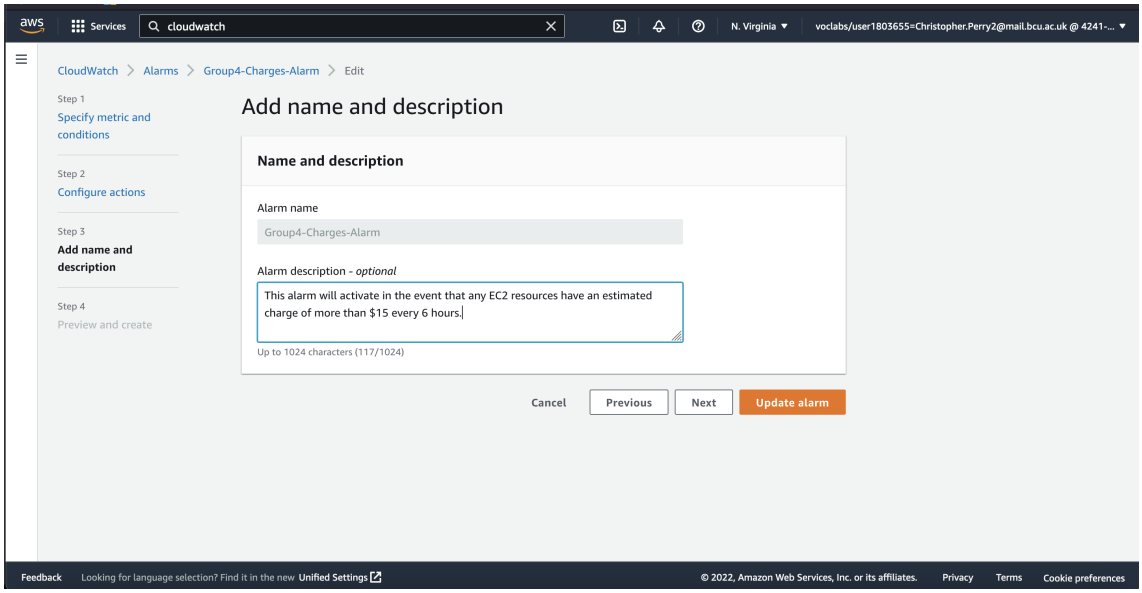


Figure 8.12: Description of CloudWatch alarm.

The alarm is now live and can be seen in Figure 8.13.

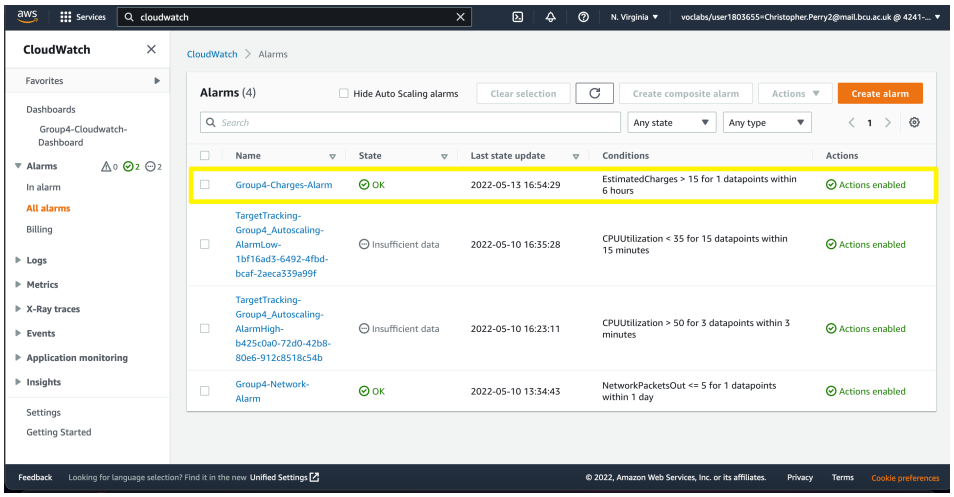


Figure 8.13: CloudWatch charges alarm live in CloudWatch dashboard.

An example of an email generated by the CloudWatch charges alarm can be seen in Figure 8.14.

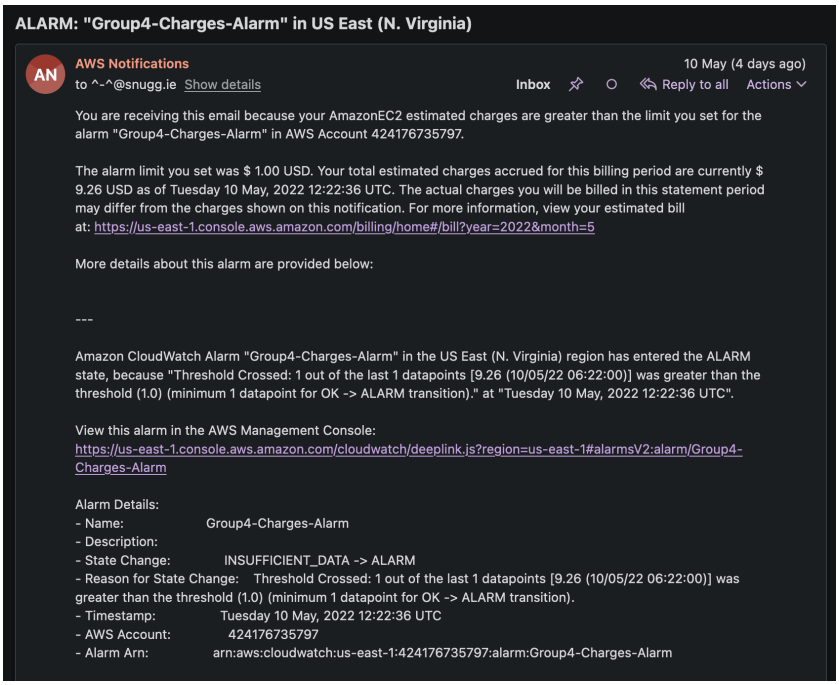


Figure 8.14: CloudWatch alarm example.

Chapter 9

CloudTrail

The use of CloudTrail will allow for logging across the project. This will give the group control over storage, analysis and any remediation actions. Visibility of AWS account activity is a recommended AWS security best practice (Amazon Web Services (AWS), 2022j).

CloudTrail will be enabled for any changes which occur within the project's S3 bucket and RDS instances. The CloudTrail is first given the name of `Group4-CloudTrail`. A location for the logs that CloudTrail outputs is required, so the S3 bucket created in Chapter 6 is selected as the storage location. The "Log file SSE-KMS encryption" option is Enabled, which encrypts the log files that are generated, heightening security. A KMS is required to encrypt this data, and a new one is created and named `Group4-kms`. These options can be seen in Figure 9.1.

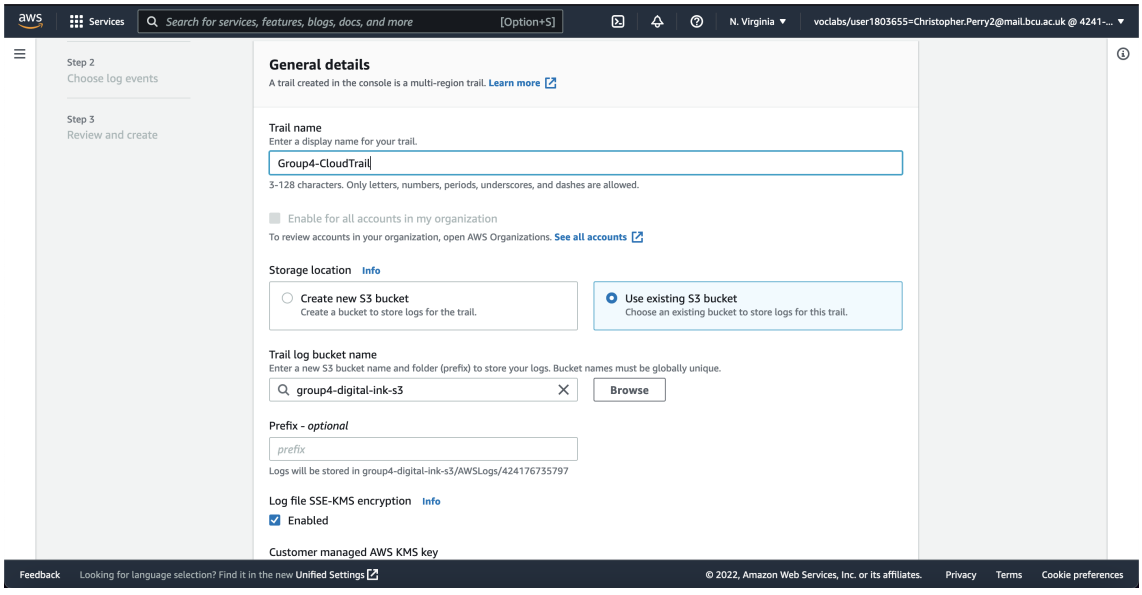


Figure 9.1: Initial CloudTrail set up.

The "Log file validation" option is selected, which ensures that any changes which have been to made to the log file that are not from CloudTrail directly are notified to the group. These aforementioned notifications have been set up through the creation of new SNS topic called `Group4-cloudtrail-sns`. These options can be seen in Figure 9.2.

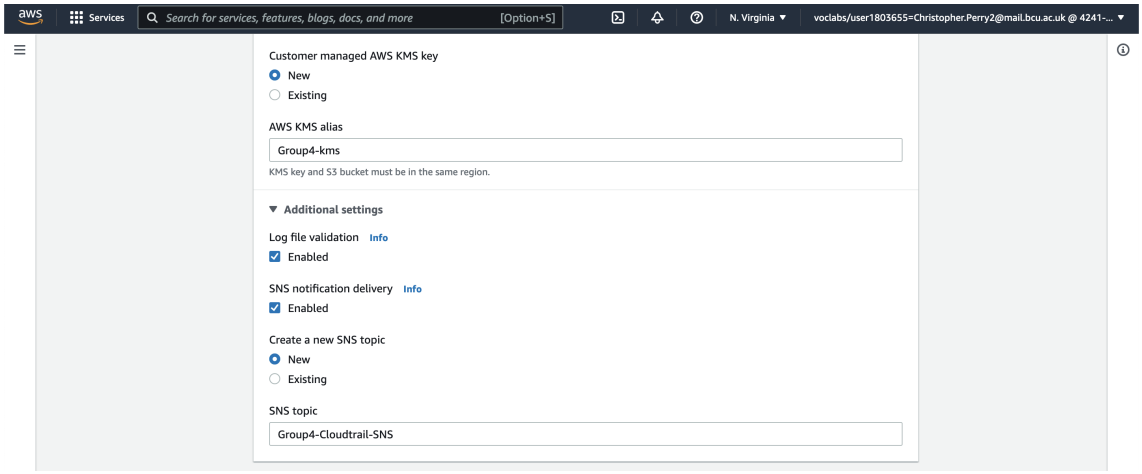


Figure 9.2: Initial CloudTrail set up.

Now that the CloudTrail initial setup is complete, the selection of the specific events which will be monitored can be configured. "Management Events" is firstly selected - this will allow for any read or write events which have been applied on the S3 and RDS services to be logged. In addition to this, "Insights Events" is also enabled, which will allow for any unusual activity on these services to also be logged through the use of the API Call Rate and Error Rate - If they are high, this will be logged via CloudTrail. which is of a security benefit to the project. These settings can be seen configured in Figure 9.3.

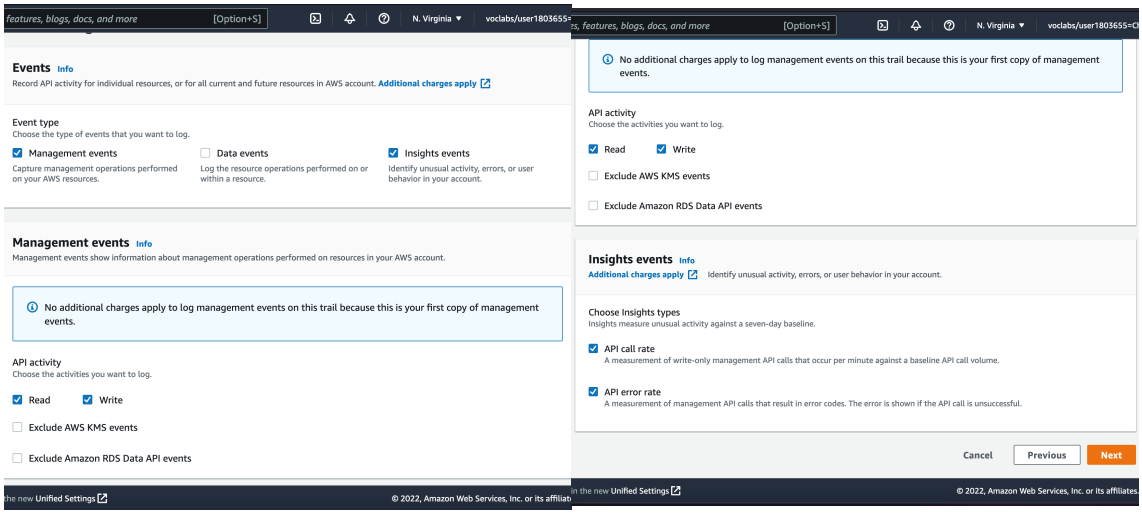


Figure 9.3: Configuring CloudTrail events.

The CloudTrail has now been added and is now monitoring for any activity. This can be seen in Figure 9.4.

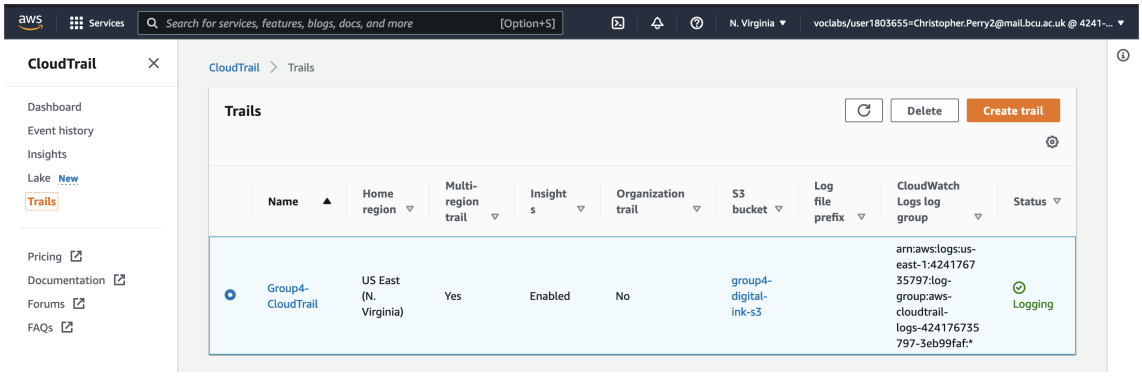


Figure 9.4: Added CloudTrail instance.

Like with CloudWatch, all users in the SNS topic receive a confirmation email once it has been set up. This can be seen in Figure 9.5.

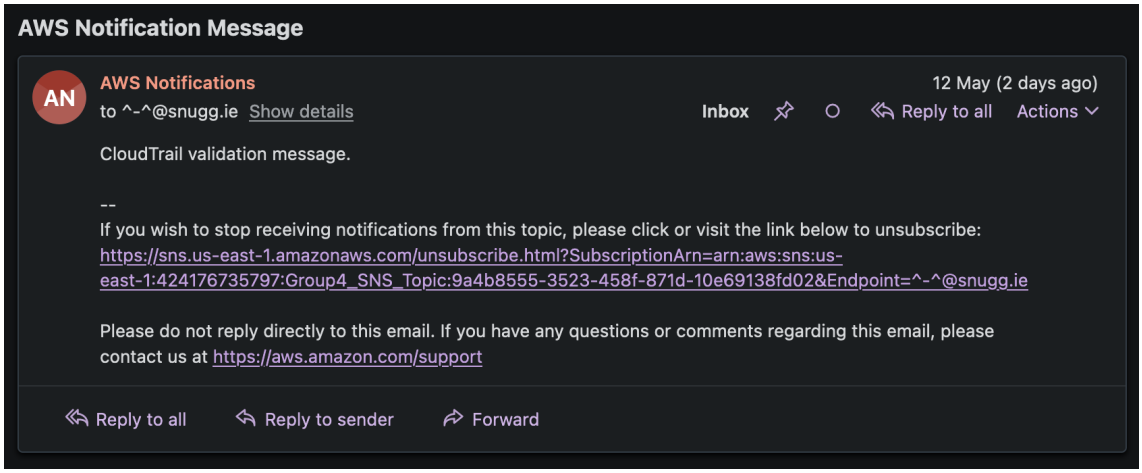


Figure 9.5: CloudTrail validation message.

Chapter 10

Relational Database Service (RDS)

Amazon RDS service allows a user to create a fully-featured and highly-available SQL database that is automatically replicated to another availability zone. (TODO: Oops, no multi-az) This means that if the primary database becomes unavailable, there is automatic failover providing redundancy for all the data stored within.

To create an Amazon RDS instance, a suitable name/identifier for the database is required before created as well as a selection for the resource limits for the virtual server. The database requires a username and passphrase, although for additional security there is the option to automatically generate a passphrase.

Afterwards, the type of SQL database required (such as MySQL, PostgreSQL, MariaDB or others) will be selected and then the database should begin provisioning.

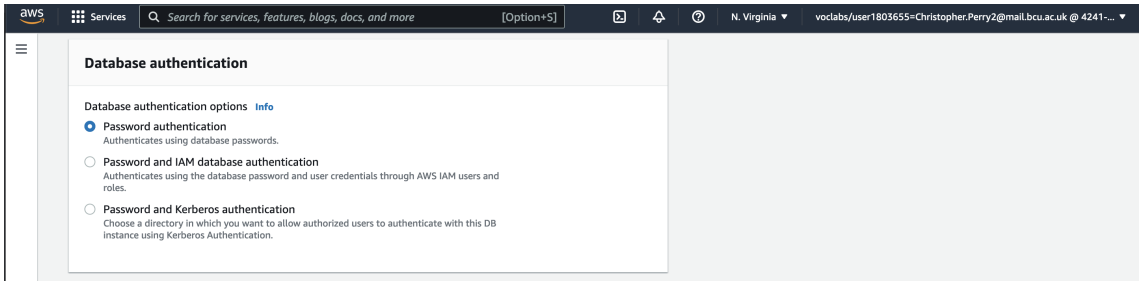


Figure 10.1: Selection of RDS Authentication Option.

Password authentication was used for the RDS instance as we did not have permission from the root user to create IAM roles within this lab environment. In the future, it would be recommended to select one of the other options to improve security, however this was not possible for the project.

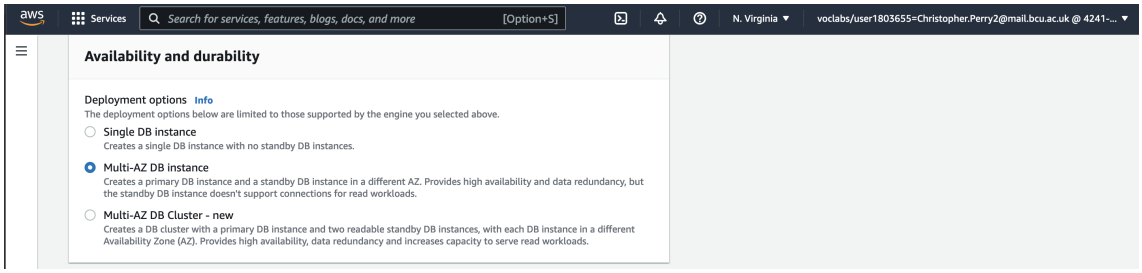


Figure 10.2: Selection RDS Multiple Availability Zone.

Multiple availability zones were selected to ensure that the database was highly available.

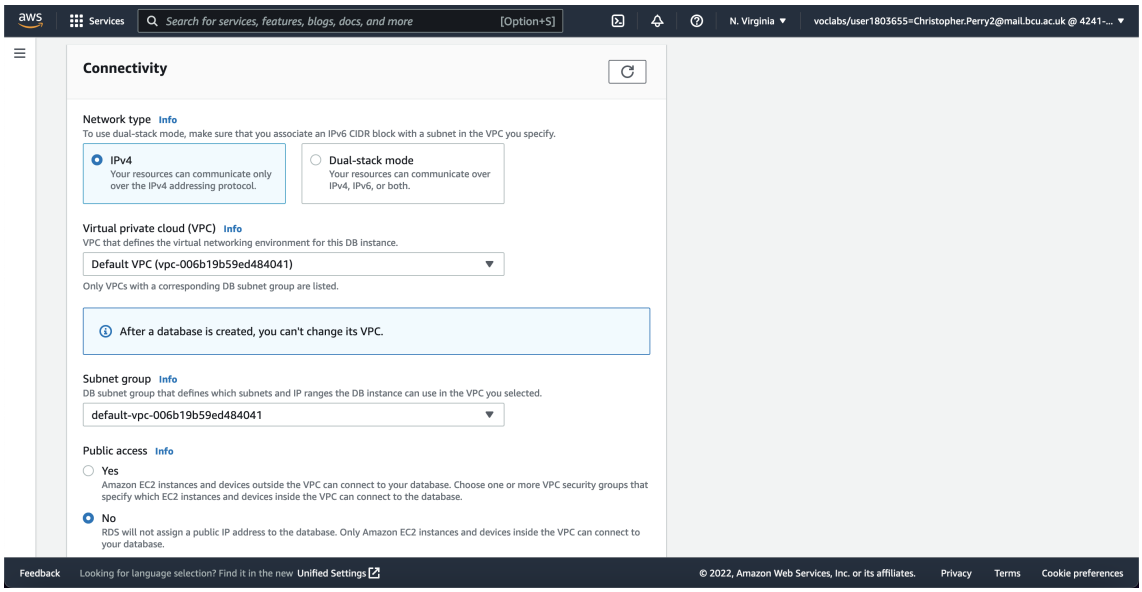


Figure 10.3: Selection of VPC and Subnet Groups.

Public access was turned off to further increase our application's security.

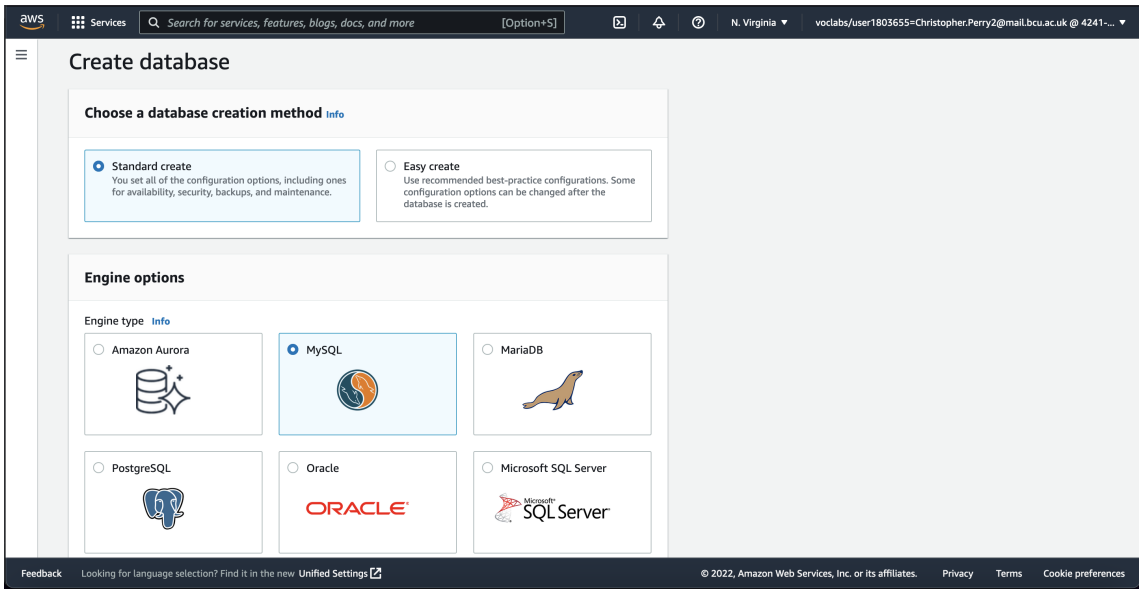


Figure 10.4: Selection of Database Engine.

A micro-sized instance was selected as we must save on AWS billing credit.

This is the estimated cost of the RDS instance for a month.

We selected an automatically generated password to ensure that it was created with current best security practices.

A selection of how much storage the RDS instance is able to access should be made. We decided that, due to billing constraints, we should go as small as possible but with

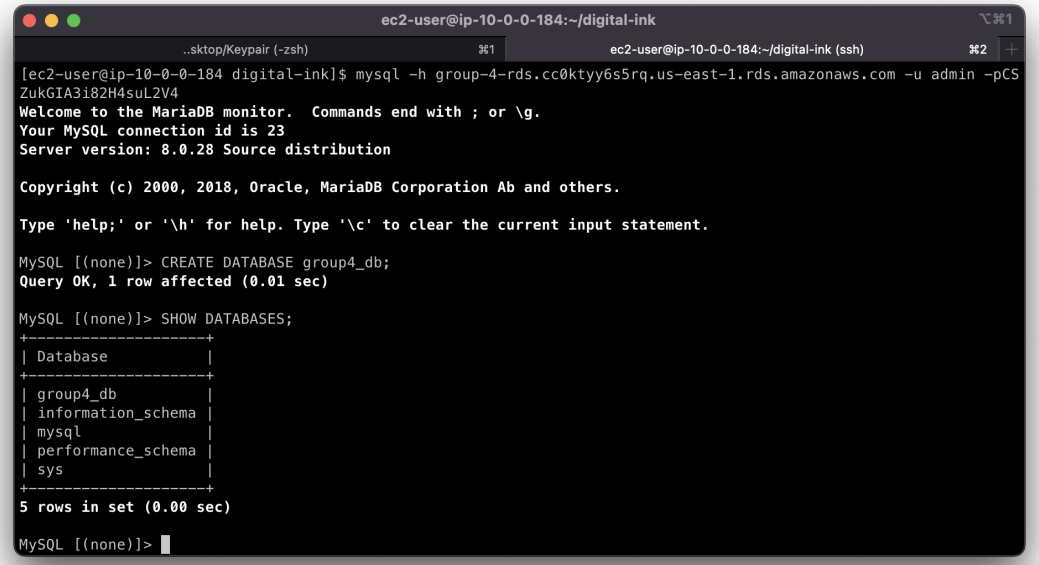


Figure 10.5: Selection of CloudWatch metric for EC2 instance.

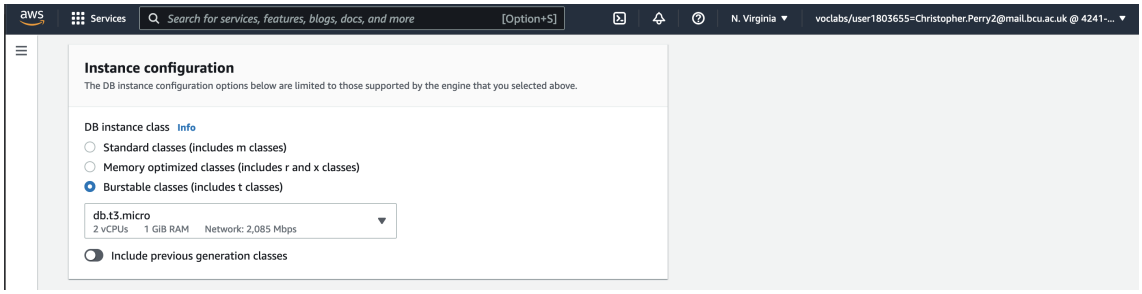


Figure 10.6: Selection of RDS Instance Size.

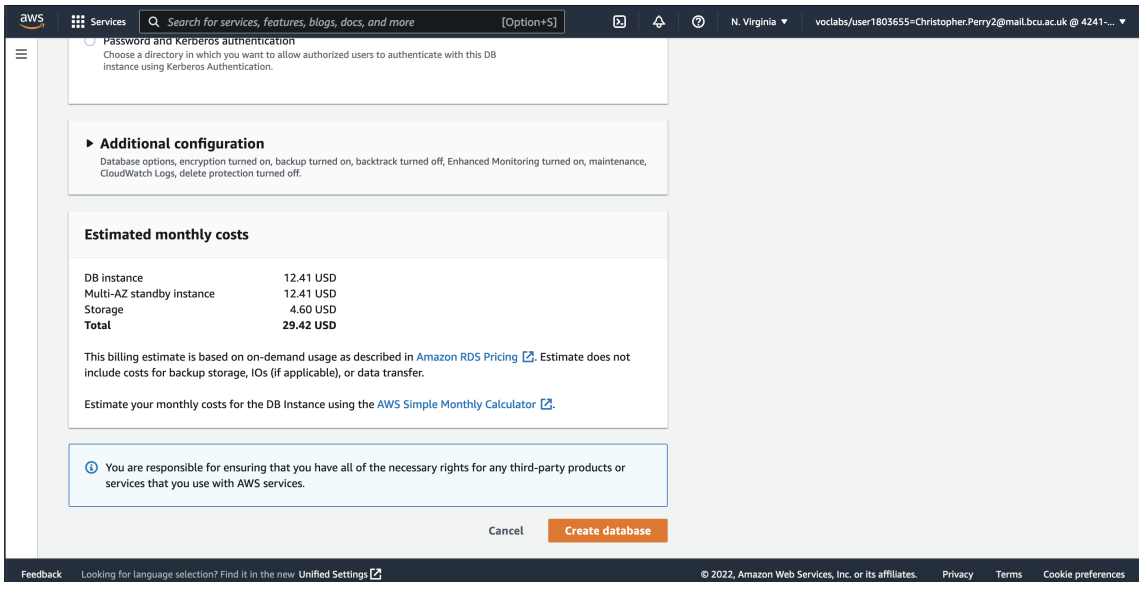


Figure 10.7: Estimated Monthly RDS Cost.

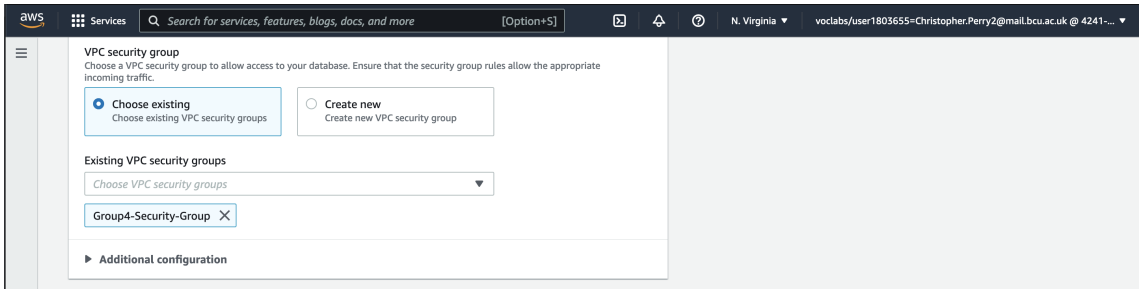


Figure 10.8: Selection RDS Security Group.

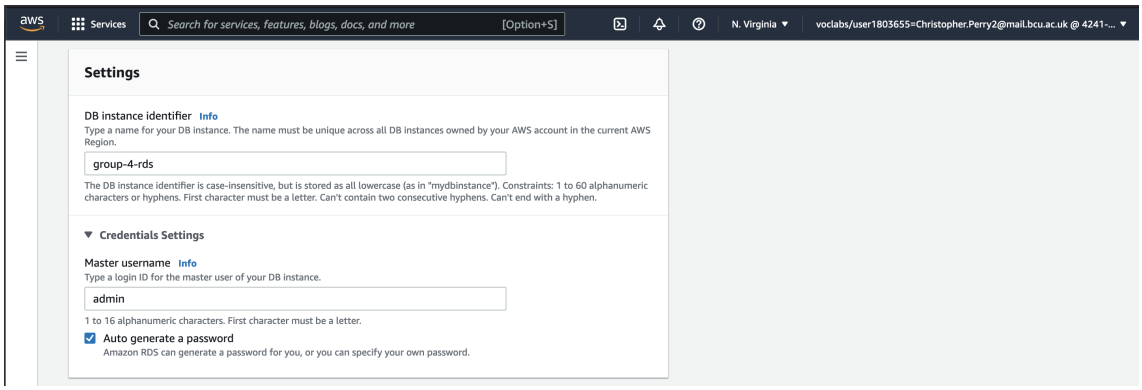


Figure 10.9: Credentials for RDS Authentication.

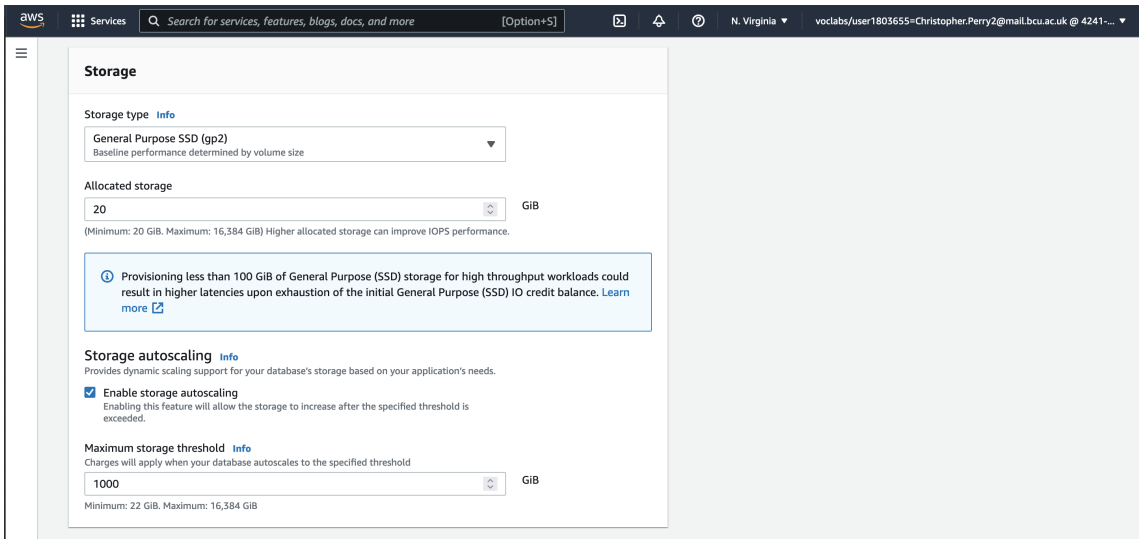


Figure 10.10: Selection of Storage Parameters.

size to automatically grow if the application was presented with more users.

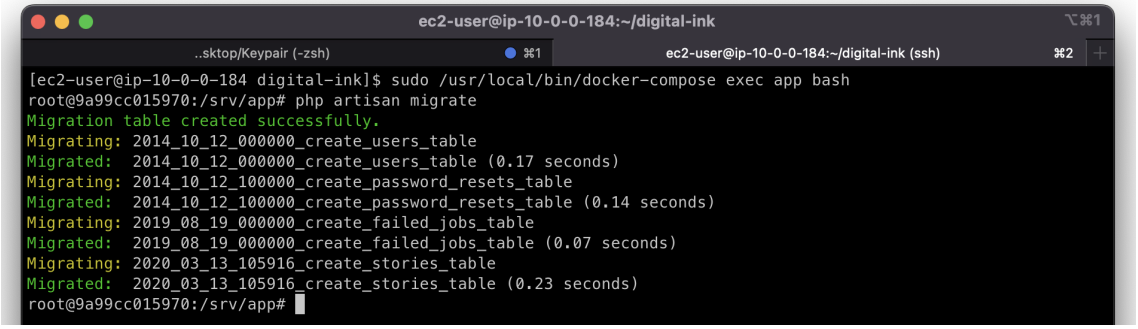


Figure 10.11: Creation of RDS Tables.

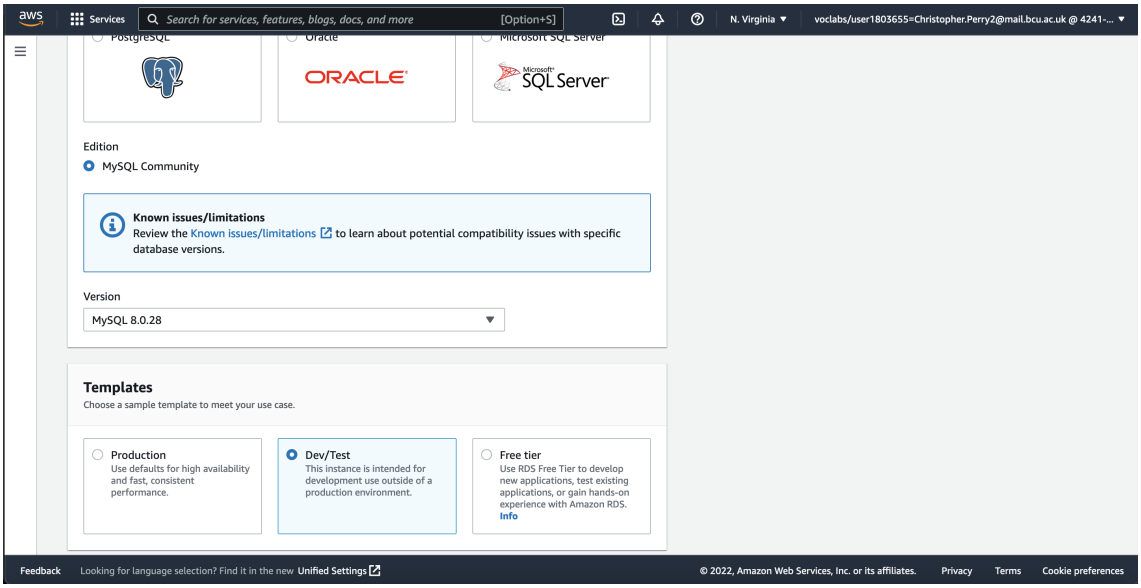


Figure 10.12: Selection of CloudWatch metric for EC2 instance.

MySQL was selected as it was the same database engine that the web application was originally using within Docker.

Then, MySQL was installed on the EC2 instance in order to sign in to the new database and create tables.

After the database was set up, tested and working, we were able to remove the old MySQL docker container from the docker-compose file.

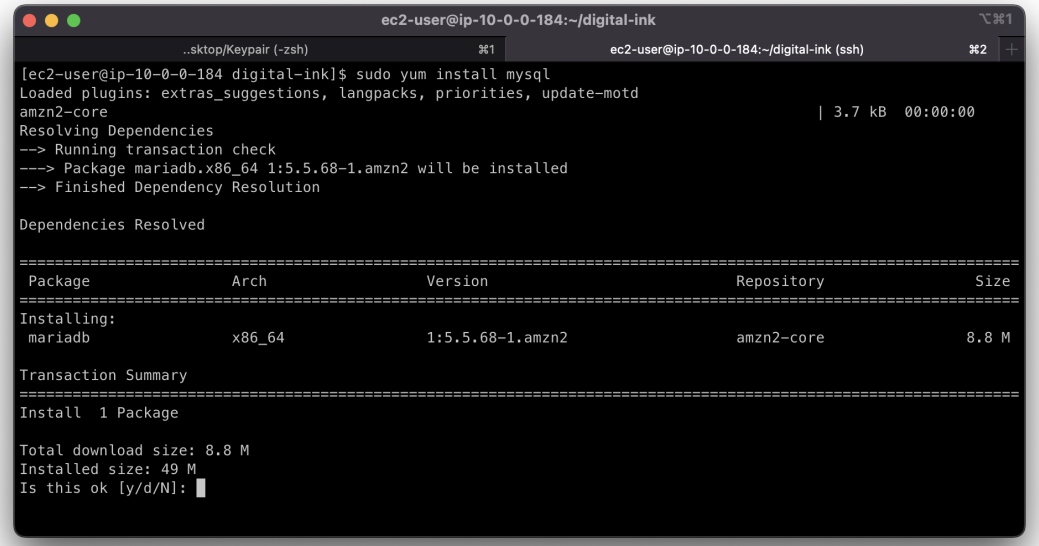


Figure 10.13: Installation of MySQL on EC2 Instance.

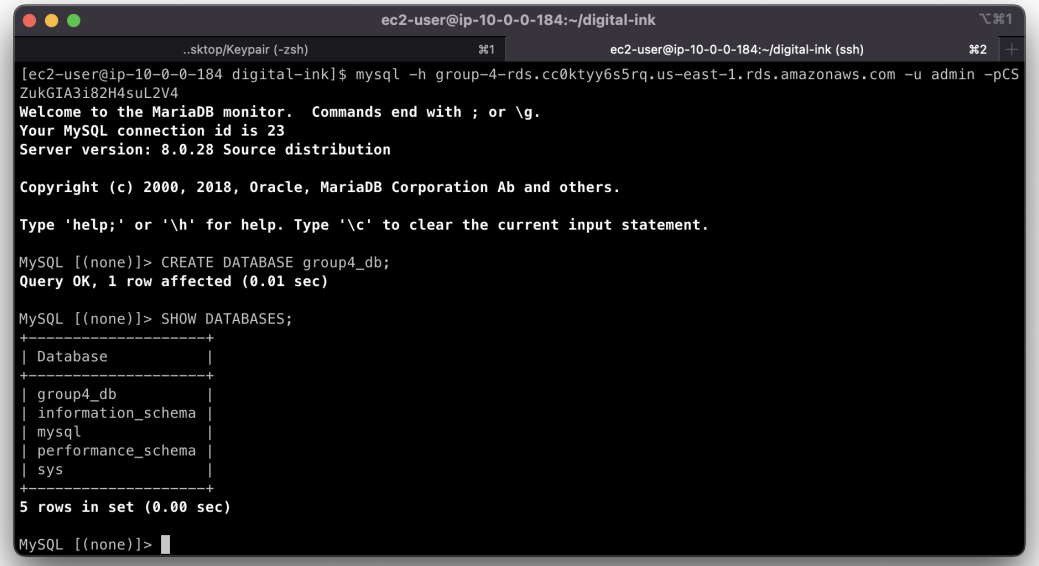


Figure 10.14: Creation of database table.

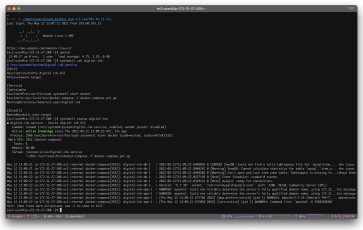


Figure 10.15: Changing Database Endpoint from Docker to AWS RDS.

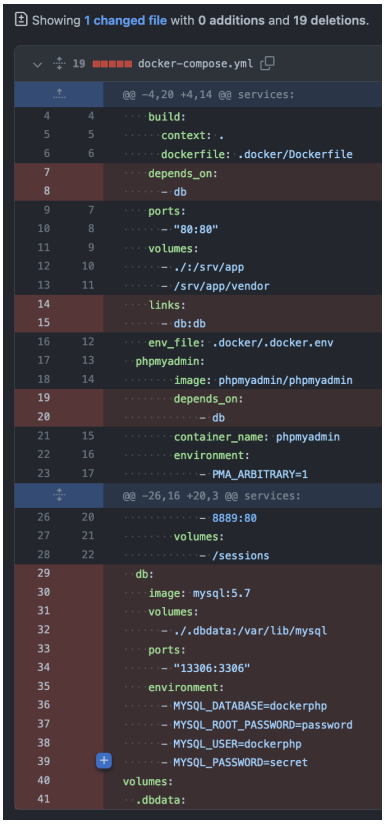


Figure 10.16: Removal of Database from Docker File.

Chapter 11

Elastic Load Balancing (ELB)

Elastic Load Balancing is used to automatically scale your EC2 instances to meet any changes in demand from your users. ELB can be used to both scale up or down the instance's resources to ensure that your application is kept performant and available regardless of how many users visit. It is also possible to scale down the resources again, after the spike in usage has passed, to save money when it comes to billing and ensure you aren't paying for any additional resources that are not being used.

1. Creation of new AMI

- Selection of AMI previously created and then "Create Instance from AMI" option (elb/elb-instance-from-ami)
- AMI selected to use same instance as "Group4-EC2" so everything is the same (elb/elb-instance-2-name)
- Same keypair used, so it is easier to switch between both instances (elb/elb-instance-2-type-and-keypair) Network Settings
- VPC set to be Group4 VPC (created in Section 4) (elb/elb-instance-2-network-settings)
- Subnet set to be "Public Subnet 2" (elb/elb-instance-2-network-settings)
- Public IP auto-assigned (elb/elb-instance-2-network-settings)
- Existing security group of "Group4-Security-Group" selected to allow HTTP, HTTPS, SSH and MySQL traffic on the instance (elb/elb-instance-2-network-settings)
- Storage remains the same as previous instance (elb/elb-instance-2-storage-config)
- Second instance now created (elb/elb-instance-2-created)

2. Creation of target group

Now that there is 2 instances, they both need to be stored within a target group.

- The Target Group type of "Instances" is selected - this will store both running instances of Digital-Ink in a group to be handled by the Load Balancer (elb/elb-target-group-basic-config)

- Target Group name of "Group4-Target-Group" given (elb/elb-target-group-basic-config)
- Protocol of HTTP and port 80 given (elb/elb-target-group-basic-config)
- Group4-VPC selected (elb/elb-vpc)
- The next screen asks for instances that will be registered to the Target Group to be selected
- The 2 running instances of Digital Ink are shown and added as targets through the "Include as pending below" button (elb/elb-register-targets)
- Target group now created, and contains both EC2 instances within them (elb/elb-target-group-created)

3. Creation of Load Balancer

- The Target groups are not associated with a Load Balancer, so one will be created as an Application Load Balancer (elb/elb-load-balancer-created)
- Given the name of "Group4-Load-Balancer' (elb/elb-basic-config)
- Set to be internet facing and will handle ipv4 addresses (elb/elb-network-mapping)
- The security group of Group4-Security-Group is selected, which allows HTTP, HTTPS, SSH and MySQL traffic within the load balancer (elb/elb-security-groups-and-listeners)
- In the event that either of the web apps goes down, it will be forwarded to the Group4-Target-Group, which will then display the available instance (elb/elb-security-groups-and-listeners)
- Could not create Global Accelerator due to permissions issue (Future improvement) (elb/elb-accelerator)
- Summary (elb/elb-summary)
- Load balancer created (elb/elb-created)
- When the load balancer is visited at http://http://group4-load-balancer-1914525647.us-east-1.elb.amazonaws.com/, the website is shown (elb/elb-working)

Chapter 12

Security Practices

AWS provides a variety of services and tools which can be configured to ensure a high level of security within the cloud architecture used for the deployment. Cyber security is a vital aspect of any online service as it is important to protect the resources hosted on AWS from cyberattacks and to prevent the web app itself from being taken off the web (**cavelty2010cyber**). It is critical that the best security practices are followed to ensure a high level of security. This chapter will detail the security measures taken during the deployment of the AWS architecture.

When configuring EC2, a KeyPair was set up for logging into the instance. This KeyPair is stored on a secure `.pem` file which should only be accessible to those who created the EC2 instance. The KeyPair is required to log into the instance and, therefore, only the authorised users who are part of the development team have access to editing the EC2 instance.

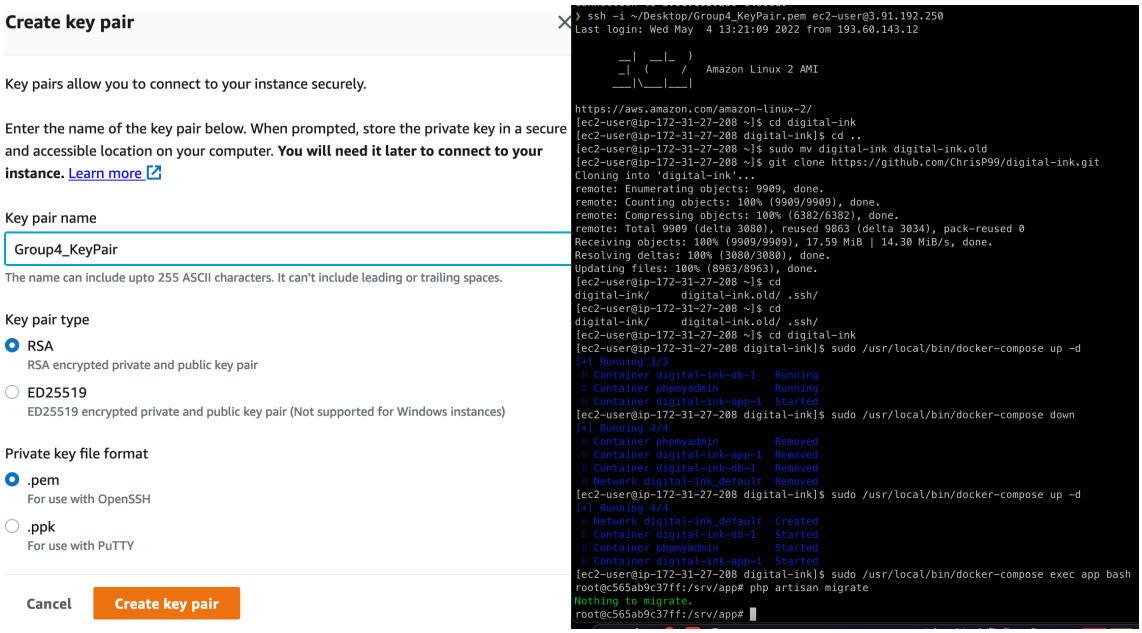


Figure 12.1: Creating and using the KeyPair.

EC2 security groups are virtual firewalls used to control incoming and outgoing traffic for the AWS deployment (**amazon2022amazon2**). Security groups were implemented which limit connections to HTTP, HTTPS, SSH, and MySQL. The HTTP, HTTPS, and MySQL connections were made accessible by public IP addresses as all users who access the web app would require these permissions (HTTP and HTTPS to view the web app, and MySQL to query the database through the web app). The SSH connections are limited to one IP address (eduroam) as the public does not need access. In a future deployment, the IP address with permission to access the SSH would be set to a private location, such as an office, rather than eduroam. During this deployment, the IP address of eduroam was allowed access due to the location in which the deployment was configured.

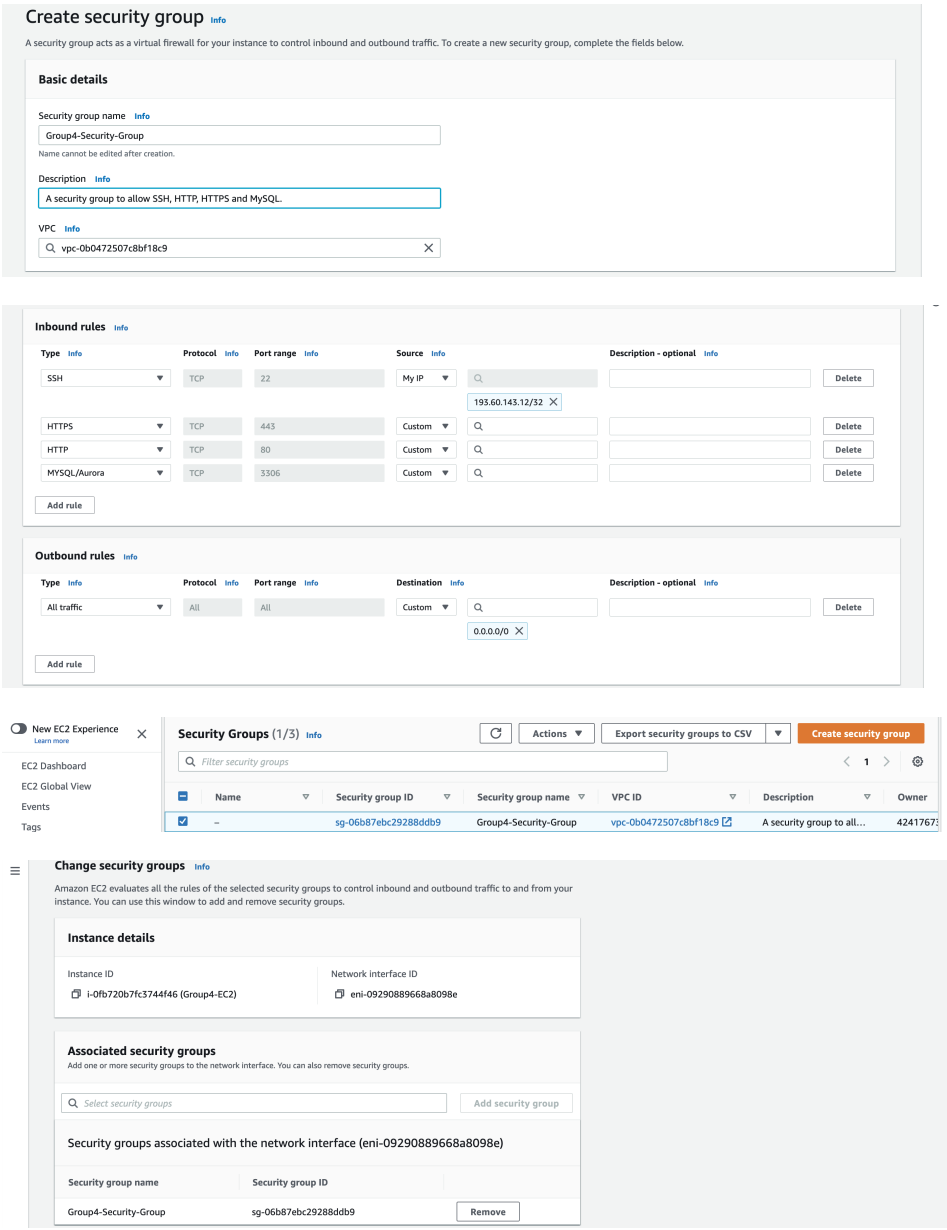


Figure 12.2: Configuring security group.

IAM is an AWS service which provides fine-tuned access control across the entire AWS deployment by creating a policy which specifies which users have permission to access certain features and resources (Amazon Web Services (AWS), 2022c). Ideally, this would be implemented for security purposes. Unfortunately, this feature could not be used due to restricted permissions.

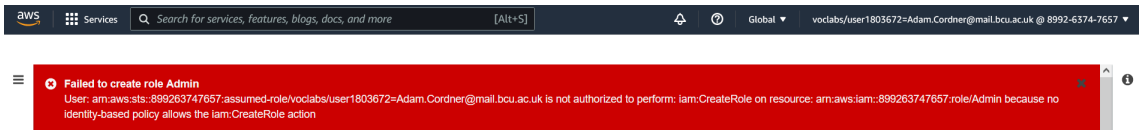


Figure 12.3: Restricted permissions preventing the creation of IAM roles.

The S3 bucket was configured to use server-side encryption to safely store the data in the bucket. This was done by specifying to use an Amazon S3-managed key (SSE-S3), i.e. an encryption key which Amazon S3 automatically creates, manages, and uses to encrypt the data.

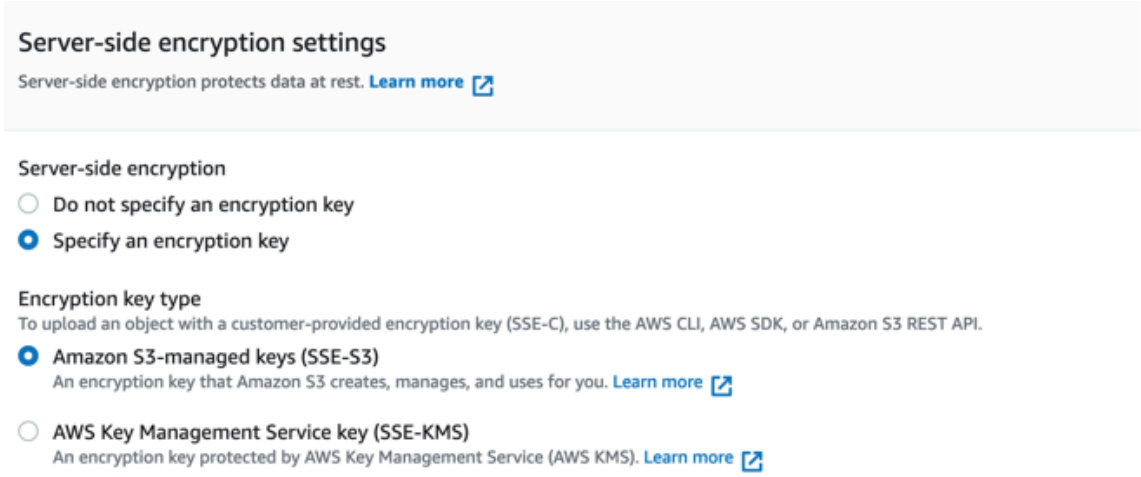


Figure 12.4: S3 bucket encryption settings.

When setting up the RDS for the web app, database authentication was configured to require password authentication. This means that direct access to the SQL database requires the use of a master username and a master password generated by AWS.

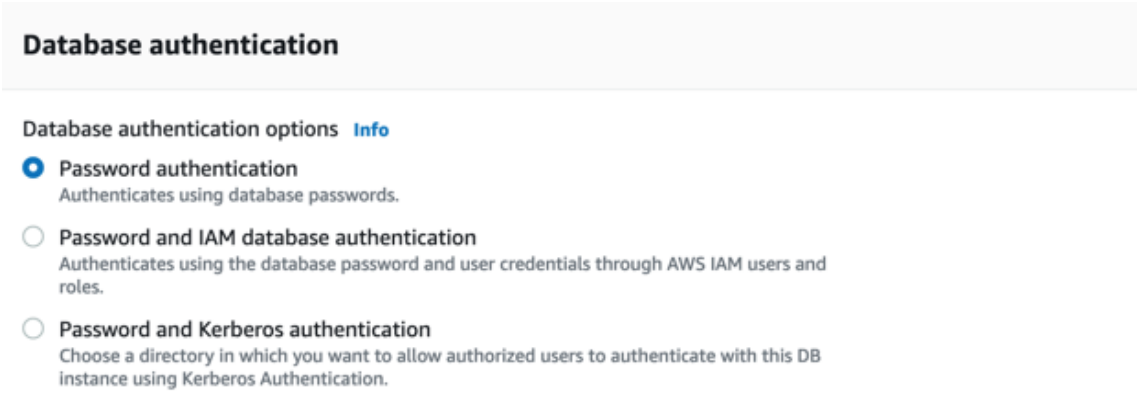


Figure 12.5: Setting up database authentication.

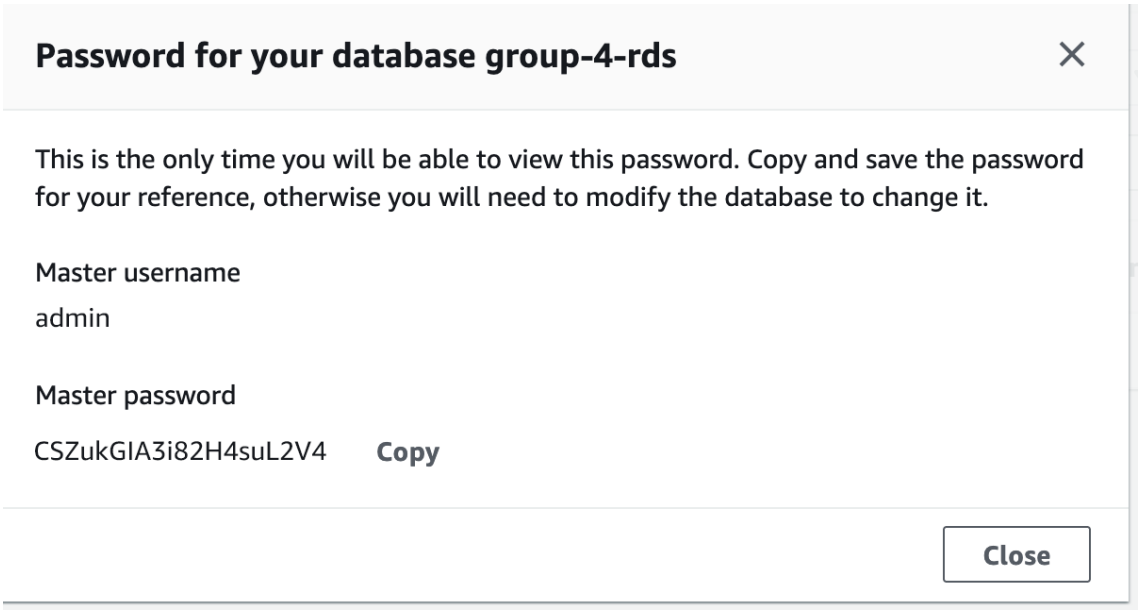


Figure 12.6: Viewing log in details for database authentication.

Chapter 13

Cost Breakdown

If this web app was to be deployed to the public, or be expanded to a larger market, it would be important to gather estimated costs for hosting the app in the cloud with all the AWS services being used. The AWS Pricing Calculator can be used to calculate current costs and predict future costs. To calculate these costs, it is required to specify every implemented feature and several projected inputs and outputs, such as amount of data transferred on the app per month (Amazon Web Services (AWS), 2022d). The monthly cost and a yearly cost of deploying the app in its current state was calculated first. Following this, the calculator was used to predict monthly and yearly costs for scaling the deployment up to larger user-bases. These figures were calculated for scaling the app up to 10,000 users, one million users, and ten million users, which would require upgrading some selected AWS features.

13.1 Estimated Costs

13.2 Scaling Up to 10,000 Users

13.3 Scaling Up to One Million Users

13.4 Scaling Up to Ten Million Users

Chapter 14

Testing

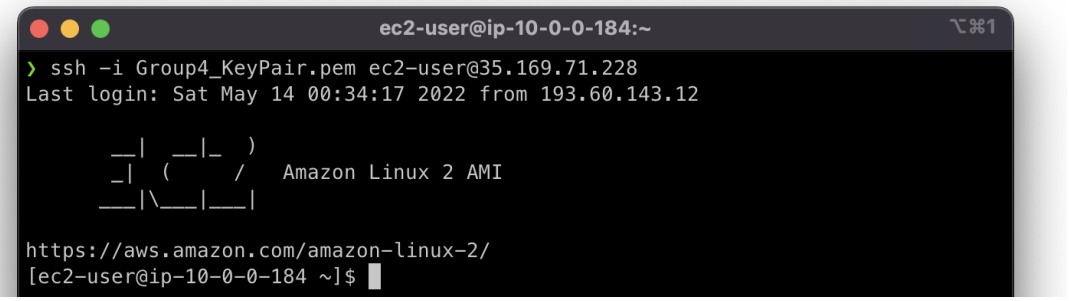
This chapter of the report will detail the testing conducted on the configured AWS services. This was done to determine the accuracy and efficiency of the configurations made during the deployment process. The testing was conducted by using Gherkin, a language used to define behaviour and test cases (Santos and Vilain, 2018). It is non-technical and is intended to be easily human-readable. Gherkin uses set keywords for structure and meaning: Given, When, and Then. An example of this structure can be seen in figure below.

```
Scenario: Does the web application work?
    Given ...
    When ...
    Then ...
```

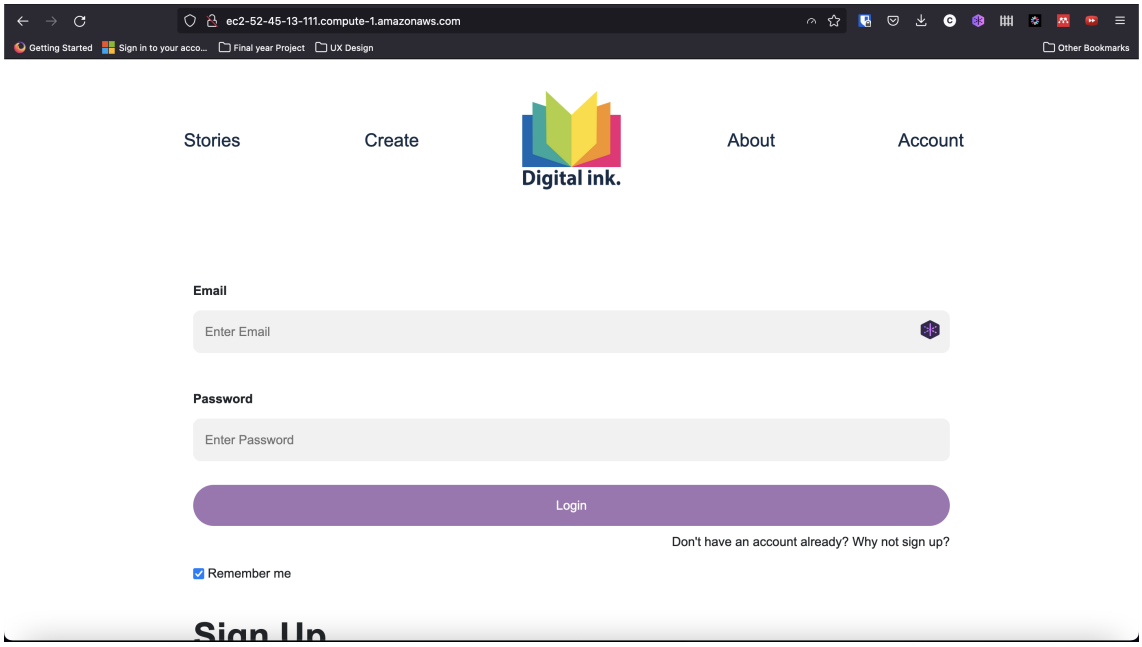
EC2, S3, CloudFront, RDS, CloudWatch, and CloudTrail were all tested using this approach. Screenshots are included to illustrate the results of these tests.

14.1 Testing EC2

```
Scenario: Accessing instance through SSH with .pem file private key.
    Given that the EC2 instance is running on AWS and the user has EC2 keypair
    When the user enters the command
    "ssh -i Group4_KeyPair.pem ec2-user@35.169.71.228" in the terminal
    Then the user will be logged into the EC2 instance
```



```
Scenario: Accessing web app through EC2 domain name.
    Given that the web app is running on the EC2 instance
    When the user accesses
    "http://ec2-35-169-71-228.compute-1.amazonaws.com/" in their browser
    Then the web app will be loaded
```



14.2 Testing S3

```
Scenario: Accessing web app image through S3 domain name.
    Given that an image on the web app is in an S3 bucket
    When the user accesses the URL
    "https://group4-digital-ink-s3.s3.amazonaws.com/" followed by the name of the image
    Then the user will see the image displayed from the S3 bucket
```

14.3 Testing CloudFront

```
Scenario: Accessing web app image through CloudFront domain name.
    Given that an image on the web app is in a CloudFront distribution
    When the user accesses the URL
    "https://d1bdkf7iuqj4qy.cloudfront.net/" followed by the name of the image
    Then the user will see the image displayed from the CloudFront distribution
```

```
Scenario: Accessing web app image through CloudFront domain name in another region.
    Given that the user is connected to the internet
    When the user accesses a resource on the web app
    Then CloudFront will distribute that content in the region closest to them
```

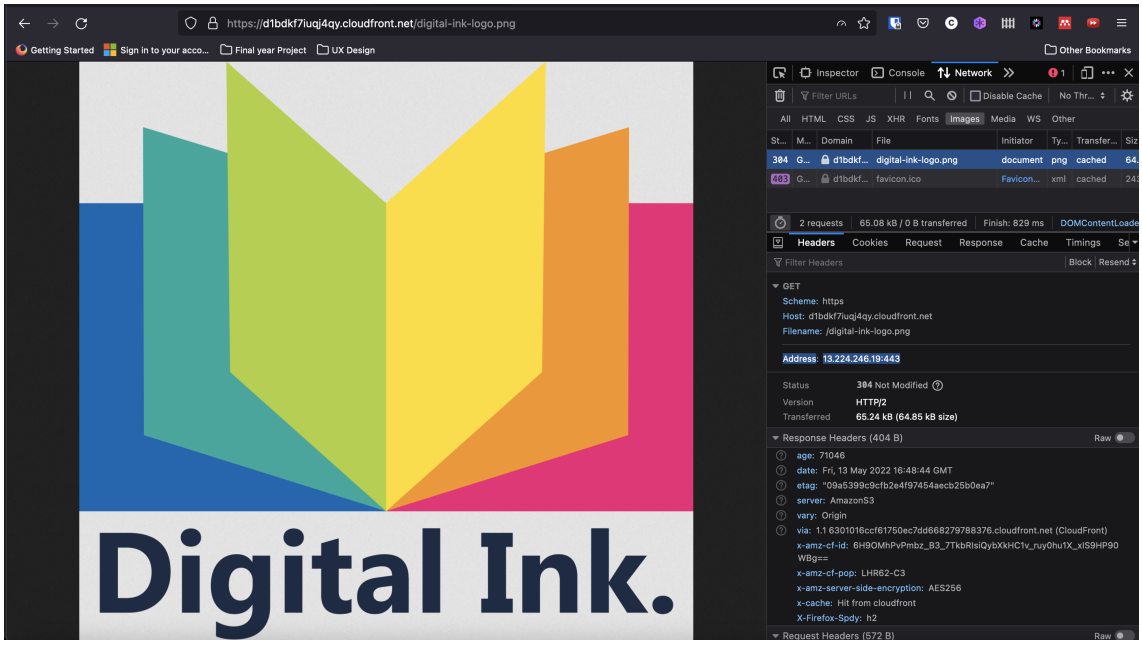


Figure 14.1: Digital Ink image whilst connected to UK IP.

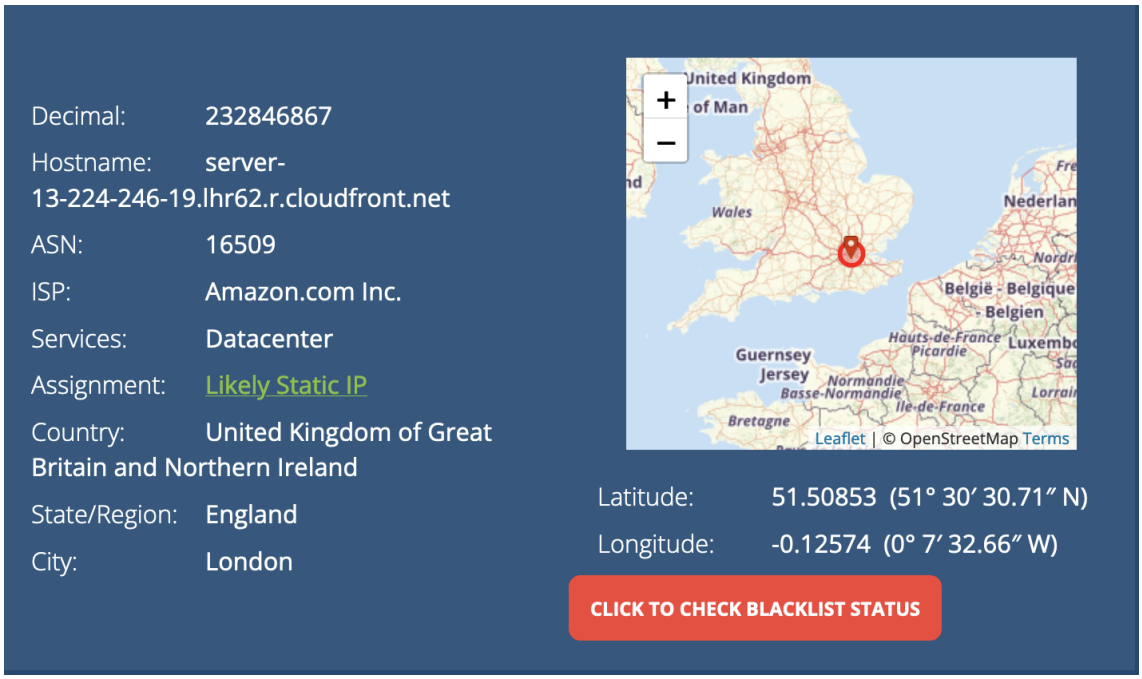


Figure 14.2: UK IP Location.

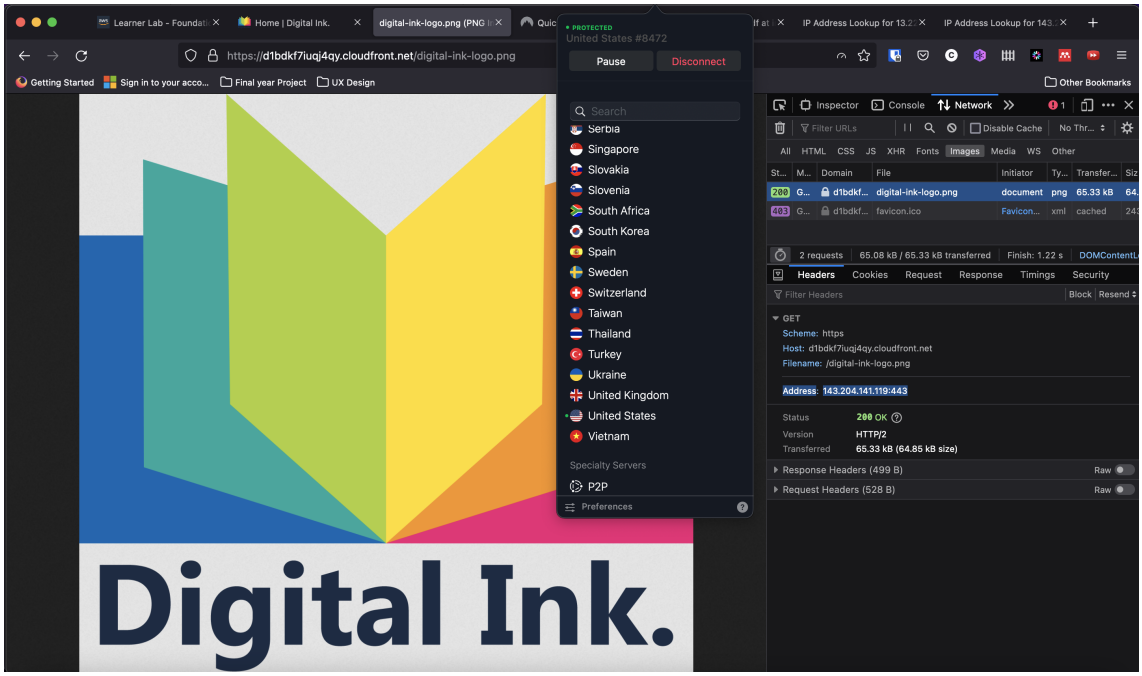


Figure 14.3: Digital Ink image whilst connected to US IP.

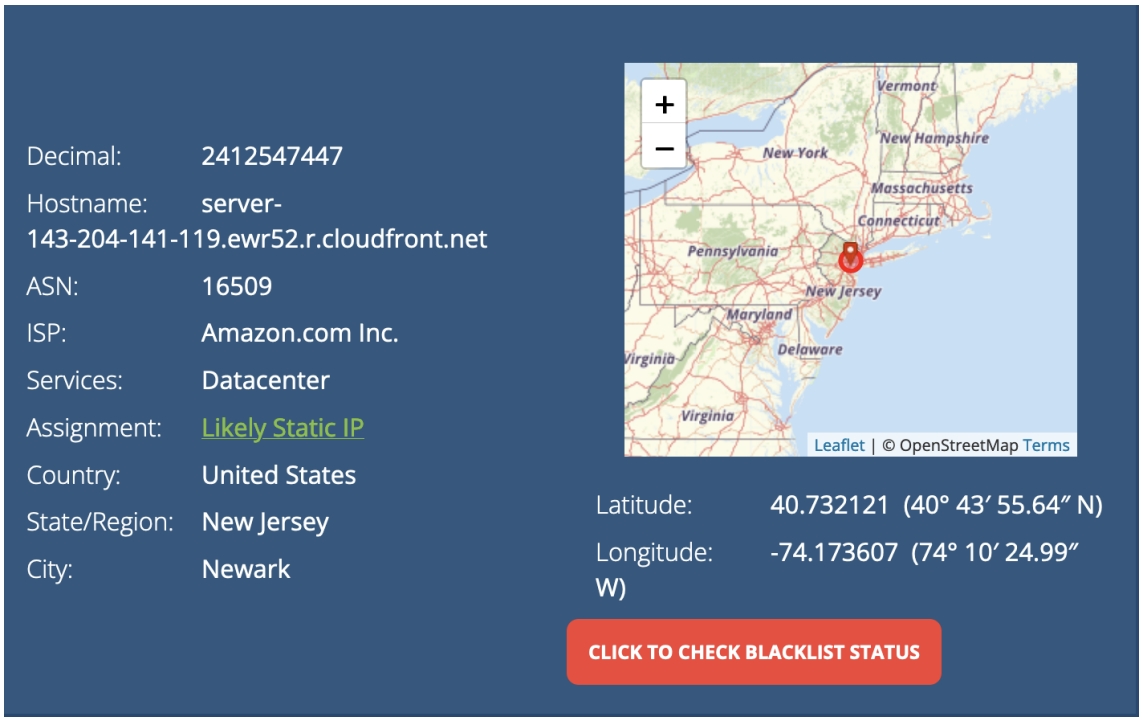


Figure 14.4: US IP Location.

14.4 Testing RDS

```
Scenario: Creating user information through the web app.
    Given that the user does not have an account on the web app
    When the user creates an account through the web app
    Then their account will be added to the users table on the RDS instance
```

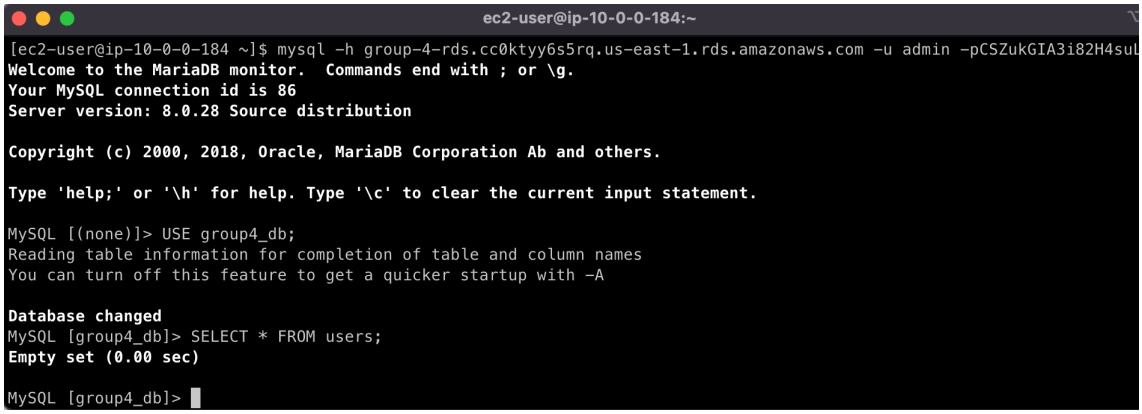


Figure 14.5: Empty users table before test.

Figure 14.6: Users table now populated after test.

```
Scenario: Creating story information through the web app.
    Given the user has a digital ink account
    When they create a story on the website
    Then their story will be added to the stories table on the RDS instance
```

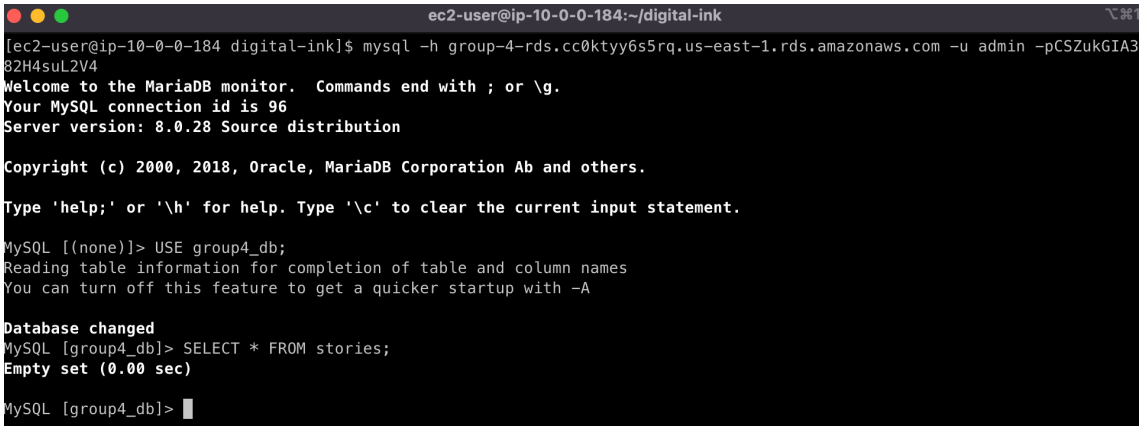


Figure 14.7: Empty stories table before test.

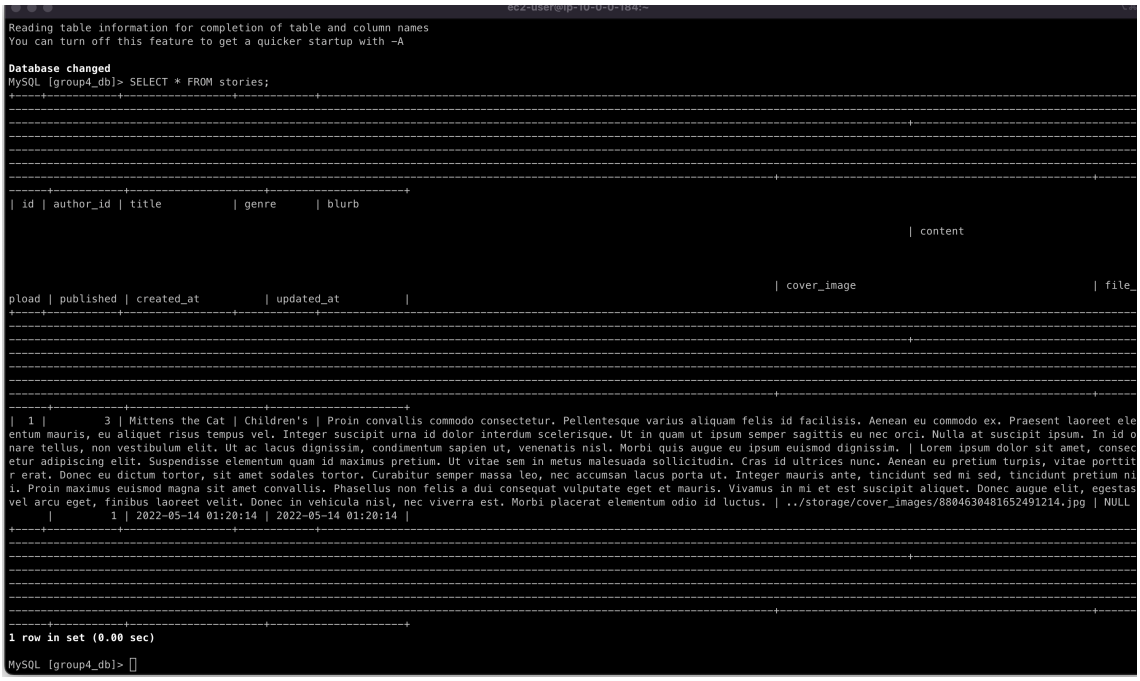


Figure 14.8: Stories table now populated after test.

```
Scenario: Updating user information in the database through the web app.
    Given the user has an account in the web app
    When they change their name and email
    Then the information will be updated in the users table in the RDS instance
```

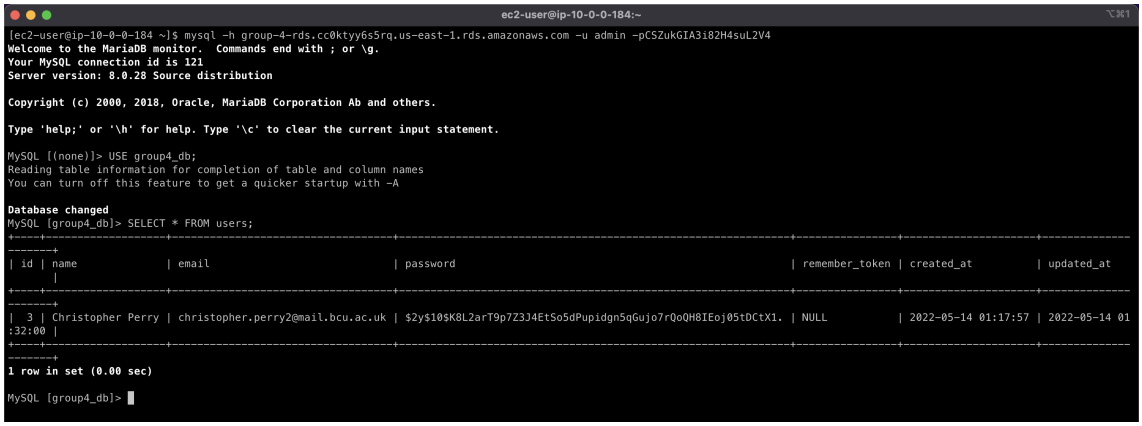


Figure 14.9: User before update.

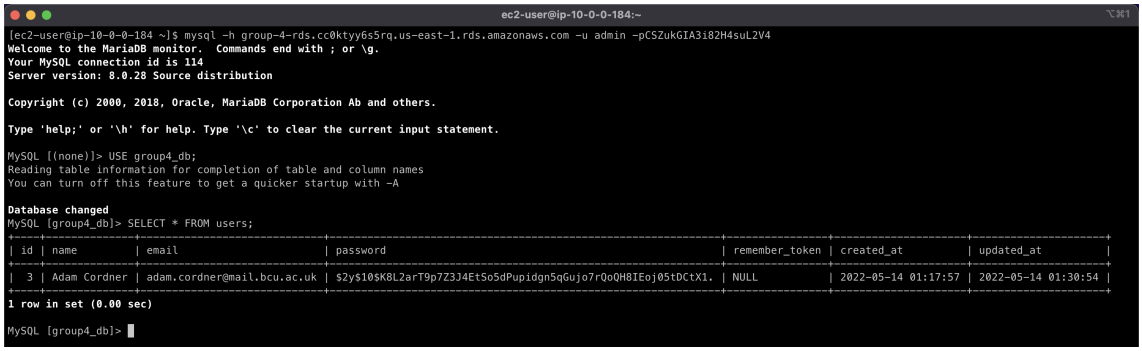


Figure 14.10: User after update. Notice that the hashed password remains the same.

```
Scenario: Deleting story information in the database through the web app.
    Given the user has an account and a story on the web app
    When they delete a story associated with their account
    Then the story will be removed from the stories table in the RDS instance
```

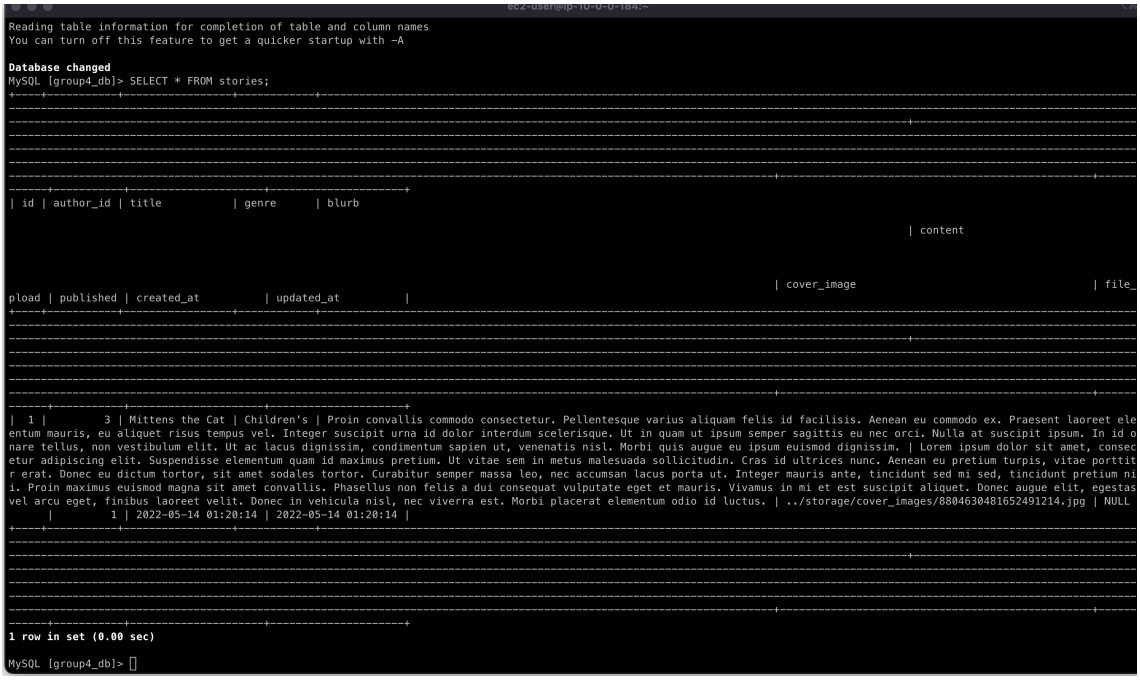


Figure 14.11: Stories table before deletion.

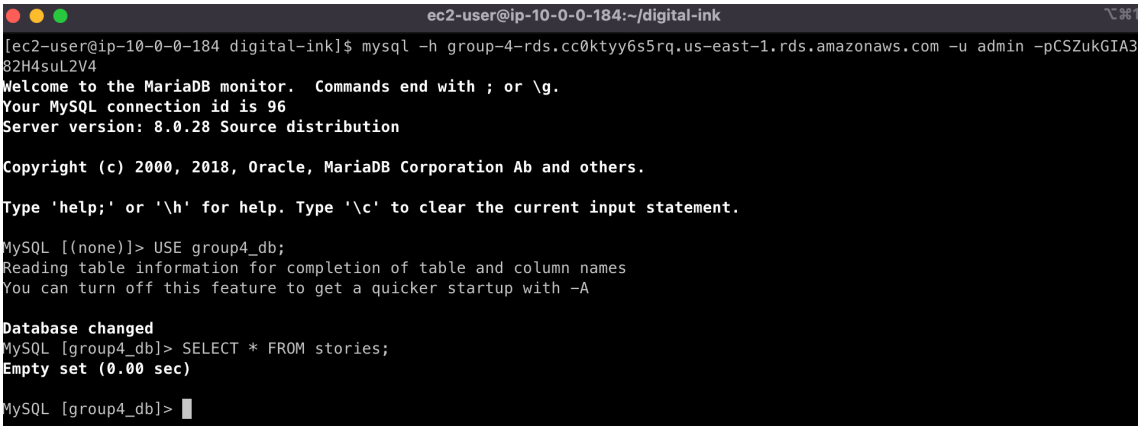


Figure 14.12: Stories table after deletion.

14.5 Testing CloudWatch

For the purposes of these tests, alarms were purposefully activated.

```
Scenario: Testing that NetworkAlarm activates
    Given that the NetworkAlarm is active
    When the EC2 instance outputs more than 500 packets of data
    in 1 minute
    Then the alarm should activate.
```

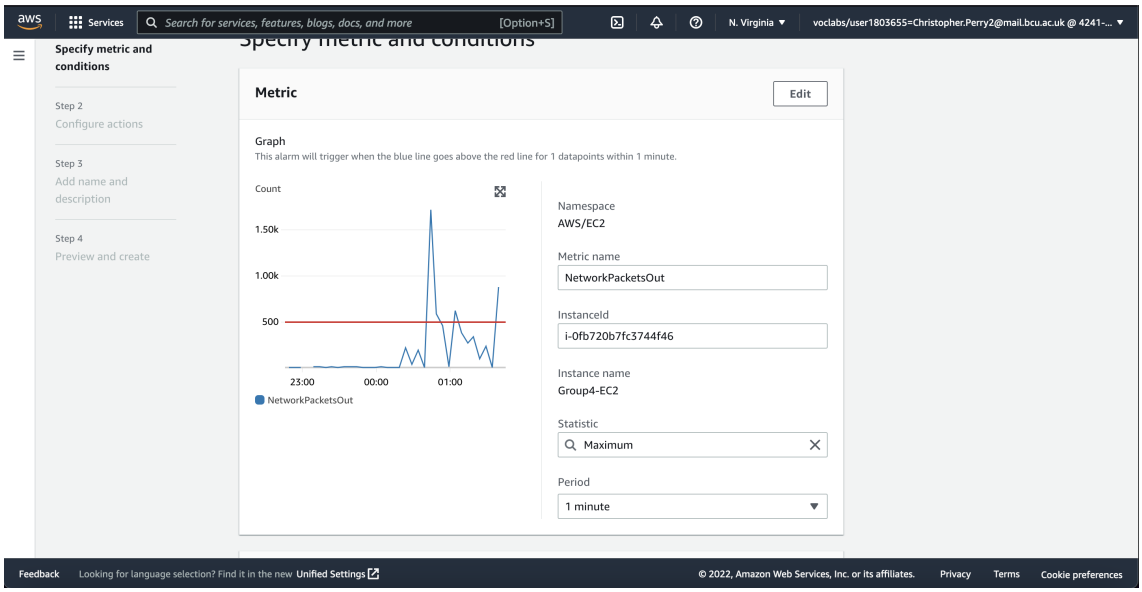


Figure 14.13: CloudWatch NetworkAlarm setup.

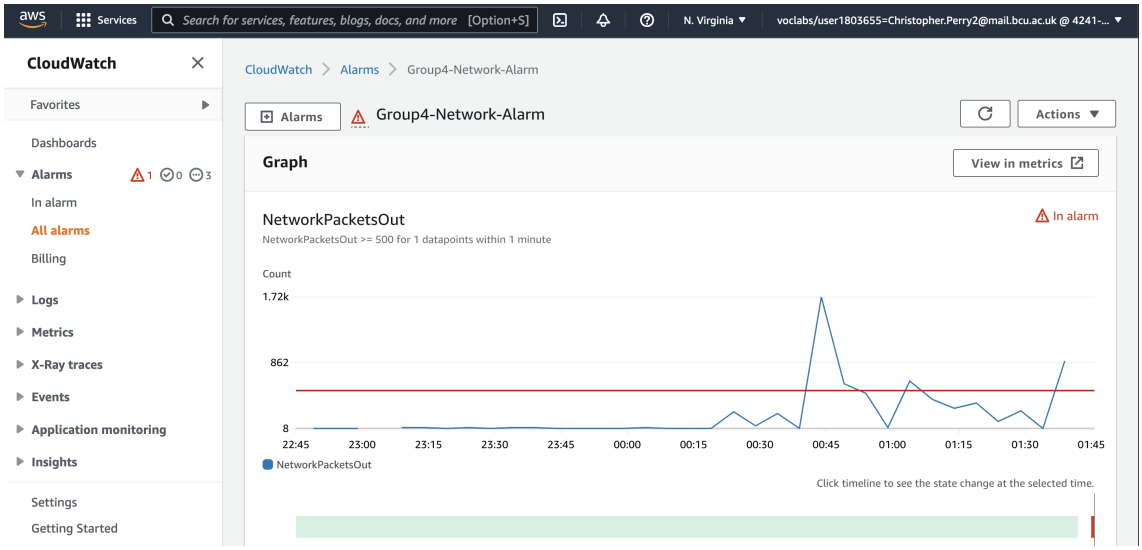


Figure 14.14: CloudWatch NetworkAlarm activated.

14.6 Testing CloudTrail

```
Scenario: Testing CloudTrail logs RDS activity.
    Given that the Group4-CloudTrail service is active
    When data is added to the RDS database
    Then this information is logged through CloudTrail
```

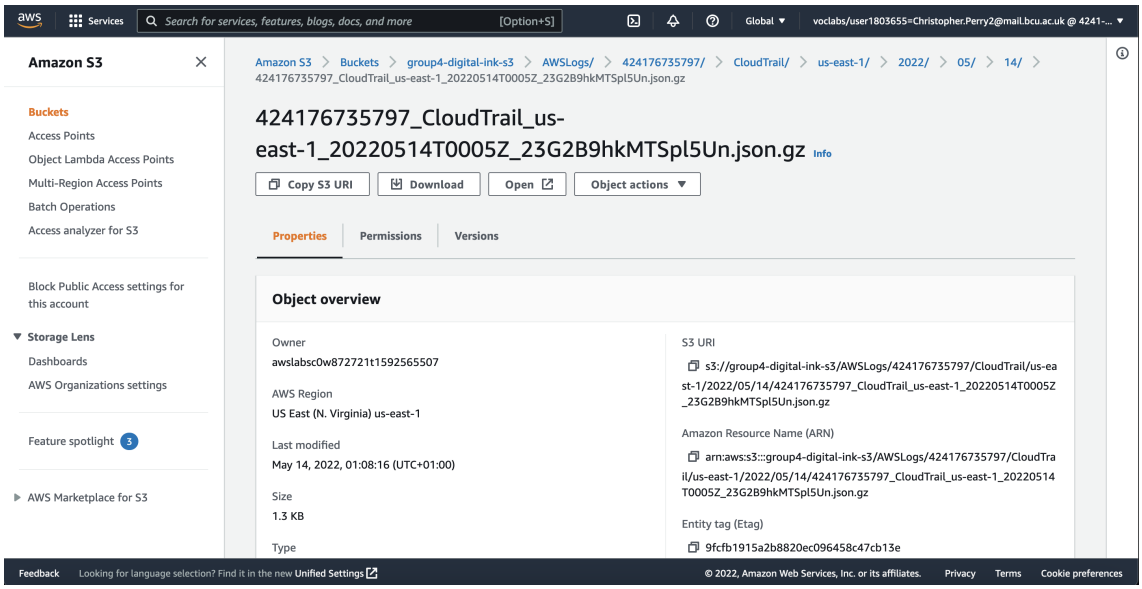


Figure 14.15: CloudTrail test.

(Add a screenshot here?)

14.7 Testing ELB

```
Scenario: Load Balanancer redirect from Instance 1 to Instance 2
    Given that the Group4-Load-Balancer service is active
    When Group4-EC2 is shut down
    Then traffic should be moved to Group4-EC2-Instance-2
```

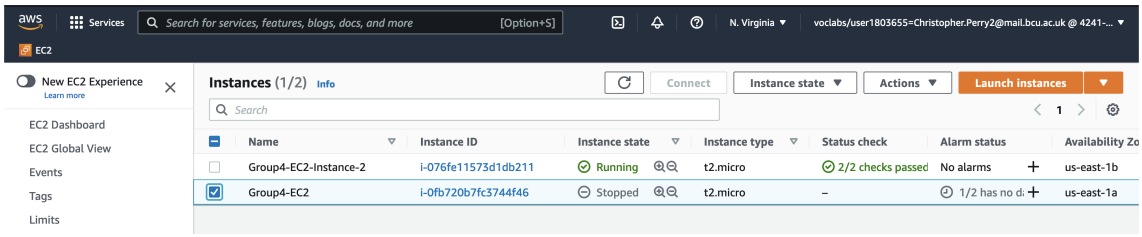


Figure 14.16: Stopped Group4-EC2 instance.

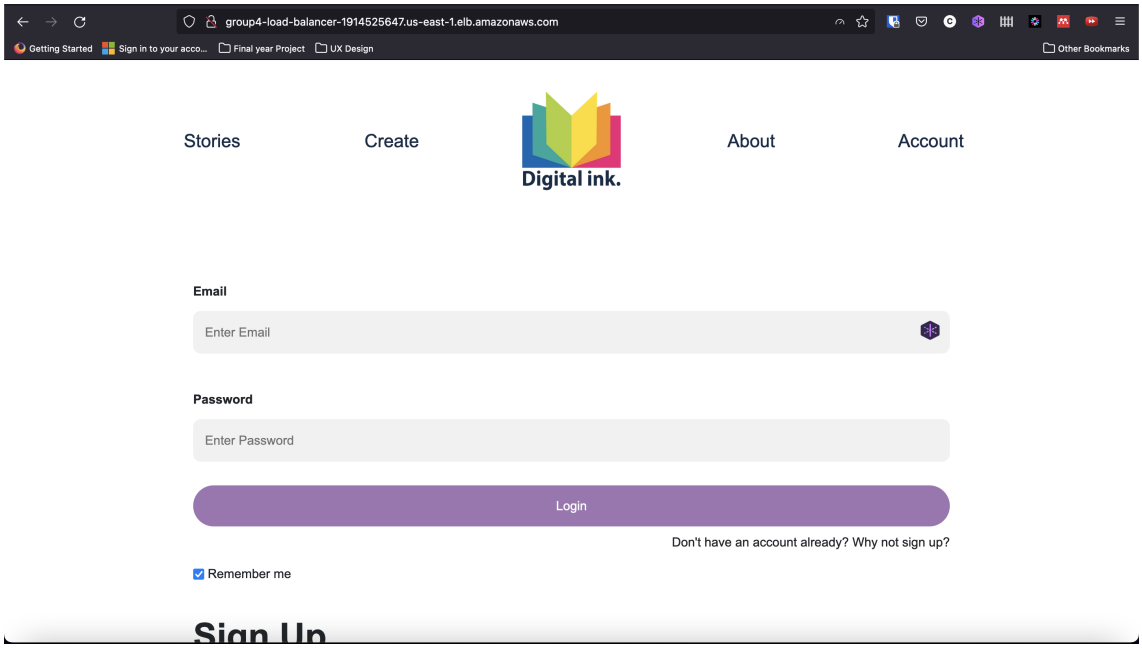


Figure 14.17: Web app page still loading through other EC2 instance.

```
Scenario: Load Balanancer redirect from Instance 2 to Instance 1
    Given that the Group4-Load-Balancer service is active
    When Group4-EC2-Instance-2 is shut down
    Then traffic should be moved to Group4-EC2
```

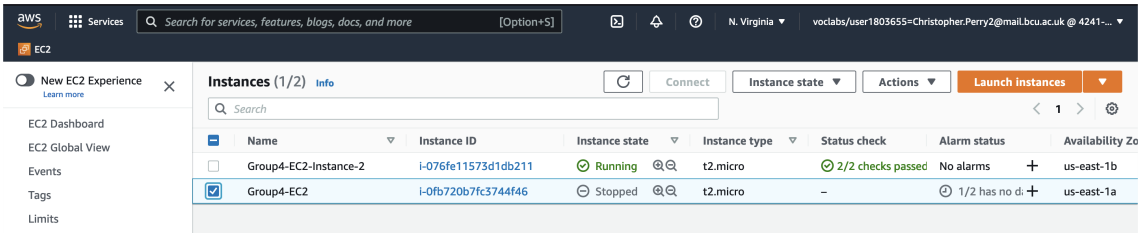


Figure 14.18: Stopped Group4-EC2-Instance-2 instance.

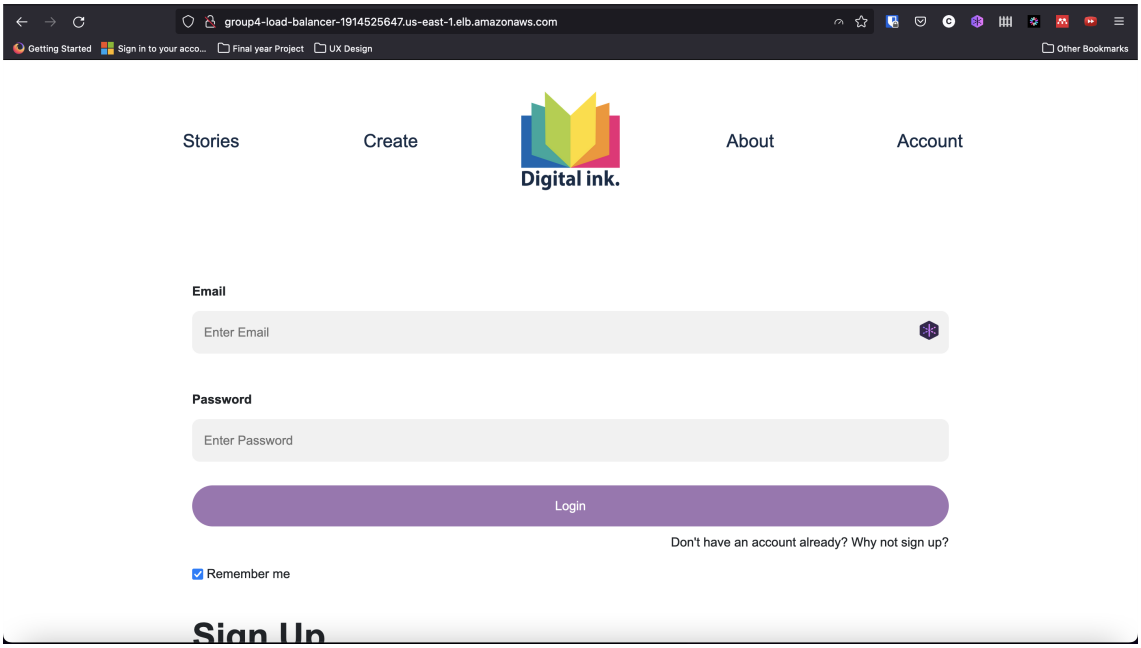


Figure 14.19: Web app page still loading through other EC2 instance.

Chapter 15

Future Enhancements

SSL and TLS are protocols for establishing safe and secure connections between networked computers and, today, support for these protocols is built into most browsers and web servers (Thomas, 2000). Adding an SSL or TLS certificate to the web app would greatly increase its security, however, it was not possible to implement such a feature during the deployment of the app. This was due to the restricted permissions granted to the AWS account used to create this deployment, which restricted the ability for certain features to be implemented, including the use of certificates. In future deployments of the web app, it would be greatly beneficial to the security of the app, for both users and the app itself, to add an SSL or TLS certificate to encrypt the connections to the EC2 instance. This would be especially important in a future deployment if the user-base would be expected to grow in size.

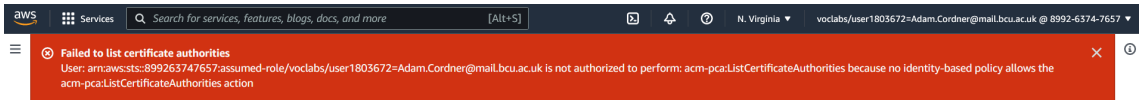


Figure 15.1: Restricted permissions preventing the use of certificates.

Another feature which was limited due to restricted permissions was AWS IAM roles. IAM is an AWS service which provides fine-tuned access control across the entire AWS deployment by creating a policy which specifies which users have permission to access certain features and resources (Amazon Web Services (AWS), 2022c). IAM would be very useful if the development team behind *Digital-Ink* was scaled up with future deployments, and it is especially appealing due to the fact that it is an entirely free service with no additional costs to use.

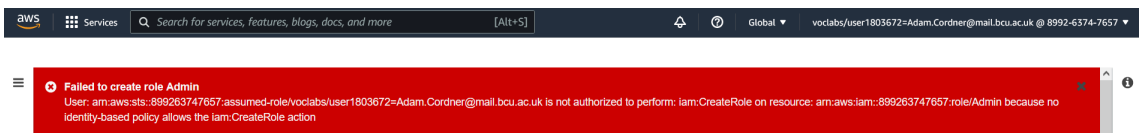


Figure 15.2: Restricted permissions preventing the creation of IAM roles.

Another AWS feature which would be useful in a larger deploy of the web app is AWS Global Accelerator. Global Accelerator is a service used to optimise user traffic. Amazon claims that it can improve the performance of user traffic by up to 60% (**amazon2022aws4**). This is useful as it can reduce packet loss, jitter, and latency. Global Accelerator works by providing two global static IPs, which improves availability, and by adding or removing endpoints of AWS services, such as ELB, EC2, and Elastic IP, in the backend without making user-facing changes. Global Accelerator automatically redirects all traffic to the most optimal endpoint. During the creation of the Load Balancer for this deployment, Global Accelerator was unavailable due to restricted permissions on the account. In larger deployments, this service would ideally be used to improve the performance of the web app.

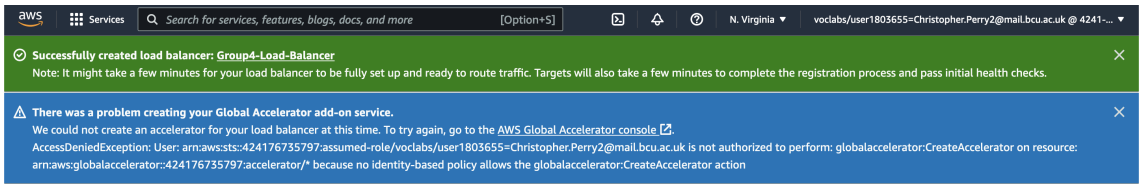


Figure 15.3: Restricted permissions preventing the creation of Global Accelerator.

If we were to redeploy the app again from scratch, we would consider using AWS Elastic Beanstalk. EB is a service AWS provides for deploying and scaling web apps automatically. EB is currently compatible with Java, .NET, PHP, Node.js, Python, Ruby, Go, and Docker (Amazon Web Services (AWS), 2022b). Considering that *Digital-Ink* uses both PHP and Docker, using EB seems appropriate for future deployments of the app. Using EB is simple, simply upload the web app code and EB automatically deploys it, including appropriate capacity provisioning, load balancing, auto-scaling, and more. EB can also be used to automatically scale-up and scale-down the web app, which would be useful as the app grows over time. In many ways, this automatic handling of the deployment is preferable as it removes the risk of human error causing the deployment to fail. This is especially helpful as human error is one of the most significant factors in failed deployments of online services, and the risk only increases as those services become bigger and harder to manage (Kraemer and Carayon, 2007). Additionally, like IAM, EB has no additional fee to use, as payment is only necessary for the AWS resources configured by EB, not the use of EB itself.

Chapter 16

Conclusion

Everything went very well and sometimes AWS might be a good idea for your project, just be aware of how much money you're spending though!

Also be careful when typing, we spent two hours on this bug/issue:

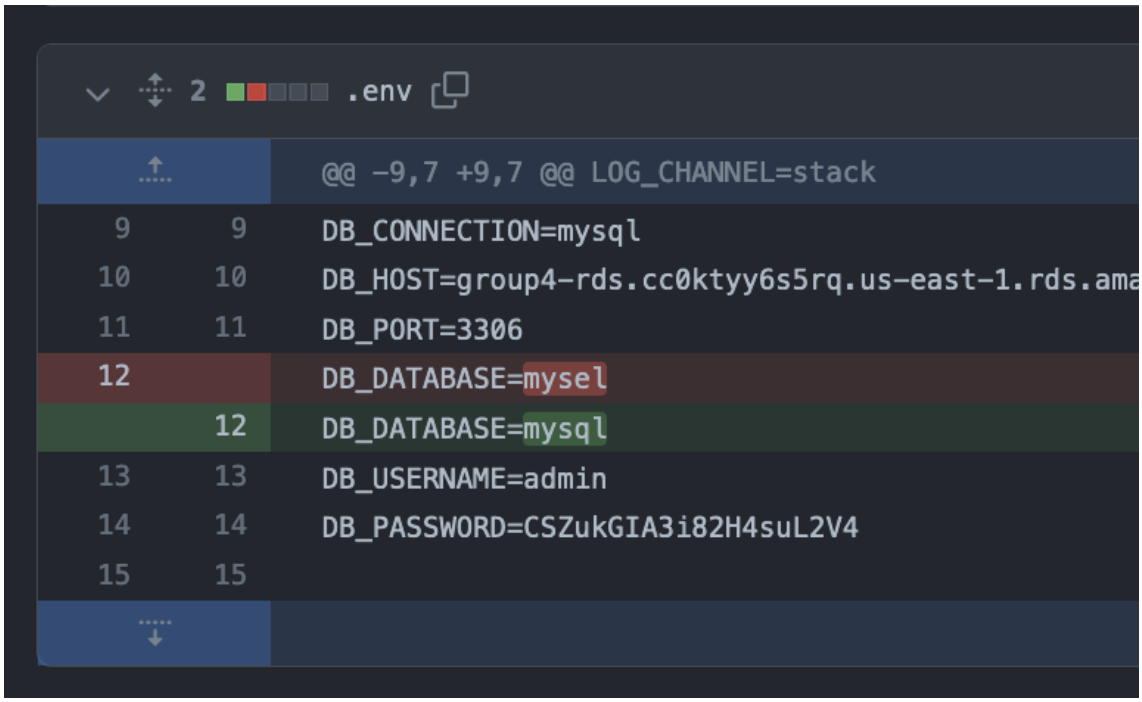


Figure 16.1: Major Bug Fixes Can Be Small

References

Amazon Web Services (AWS) (2022a). *Amazon CloudWatch - Application and Infrastructure Monitoring*. Available at: https://aws.amazon.com/cloudwatch.

– (2022b). *AWS Elastic Beanstalk - Deploy Web Applications*. Available at: https://aws.amazon.com/elasticbeanstalk/.

– (2022c). *AWS Identity and Access Management - Amazon Web Services*. Available at: https://aws.amazon.com/iam/.

– (2022d). *AWS Pricing Calculator*. Available at: https://calculator.aws.

– (2022e). *Cloud Object Storage - Amazon S3 - Amazon Web Services*. Available at: https://aws.amazon.com/s3/.

– (2022f). *Connect to the internet using an internet gateway*. Available at: https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html.

– (2022g). *Low-Latency Content Delivery Network (CDN) - Amazon AWS*. Available at: https://aws.amazon.com/cloudfront/.

– (2022h). *What is Amazon EC2?* Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

– (2022i). *What is Amazon VPC?* Available at: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html.

– (2022j). *What is AWS CloudTrail?* Available at: https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-user-guide.html.

Anderson, C. (2015). “Docker [software engineering]”. In: *Ieee Software* 32.3, pp. 102–c3. Available at: https://doi.org/10.1109/MS.2015.62.

Fielding, R. T. and G. Kaiser (1997). “The Apache HTTP server project”. In: *IEEE Internet Computing* 1.4, pp. 88–90. Available at: https://doi.org/10.1109/4236.612229.

Kraemer, S. and P. Carayon (2007). “Human errors and violations in computer and information security: The viewpoint of network administrators and security specialists”. In: *Applied ergonomics* 38.2, pp. 143–154. Available at: https://doi.org/10.1016/j.apergo.2006.03.010.

Laravel (2022a). *Blade Templates*. Available at: https://laravel.com/docs/9.x/blade.

– (2022b). *Hashing*. Available at: https://laravel.com/docs/9.x/hashing.

Lee, J. and B. Ware (2003). *Open Source Web Development with LAMP: Using Linux, Apache, MySQL, Perl, and PHP*. Addison-Wesley Professional. Available at: https://isbnsearch.org/9780201770612.

Lerdorf, R. et al. (2002). *Programming Php*. " O’Reilly Media, Inc." Available at: https://isbnsearch.org/9781565926103.

Santos, E. C. dos and P. Vilain (2018). “Automated acceptance tests as software requirements: An experiment to compare the applicability of fit tables and gherkin language”.

In: *International Conference on Agile Software Development*. Springer, pp. 104–119. Available at: https://doi.org/10.1007/978-3-319-91602-6_7.

Thomas, S. (2000). *SSL and TLS essentials*. Available at: https://isbnsearch.org/9780471383543.

Widenius, M., D. Axmark, and K. Arno (2002). *MySQL reference manual: documentation from the source*. " O'Reilly Media, Inc." Available at: https://isbnsearch.org/9780596002657.