# Cloud Computing with AWS

By Adam Cordner, Chris Perry & Evie Snuggle

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
- **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

- **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
- **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. We 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 (**lee2003open**). The web app is hosted within a Docker container (**anderson2015docker**) 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 (**fielding1997apache**). MySQL is an open-source relational database management system (**widenius2002mysql**) 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 (**lerdorf2002programming**). 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. As shown in Figure 3.1, the database which this web app uses consists of three tables: `users`, `stories`, and `migrations`.

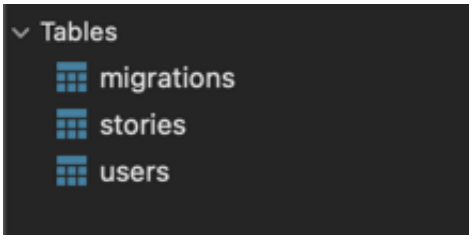


Figure 3.1: Database tables overview.

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 (**laravel2022hashing**).
- `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.

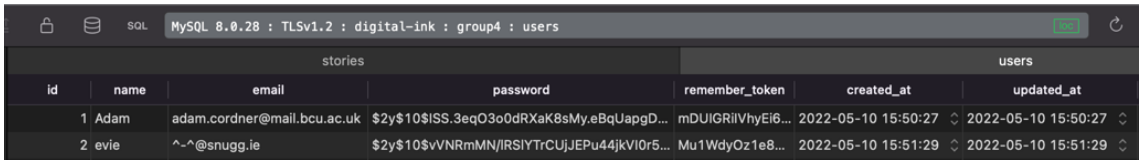


Figure 3.3: `users` table.

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.

[width=150mm]resources/database/stories-records-1
[width=150mm]resources/database/stories-records-2

Figure 3.4: `stories` table.

## 3.3 Interface Design

The design of the web app was created using Blade, a powerful templating engine (**laravel2022blade**). 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.

[width=150mm]resources/webapp/digital-ink-home
[width=150mm]resources/webapp/digital-ink-sign-up

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.

[width=150mm]resources/webapp/digital-ink-create-story-1
[width=150mm]resources/webapp/digital-ink-create-story-2
[width=150mm]resources/webapp/digital-ink-create-story-3

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.

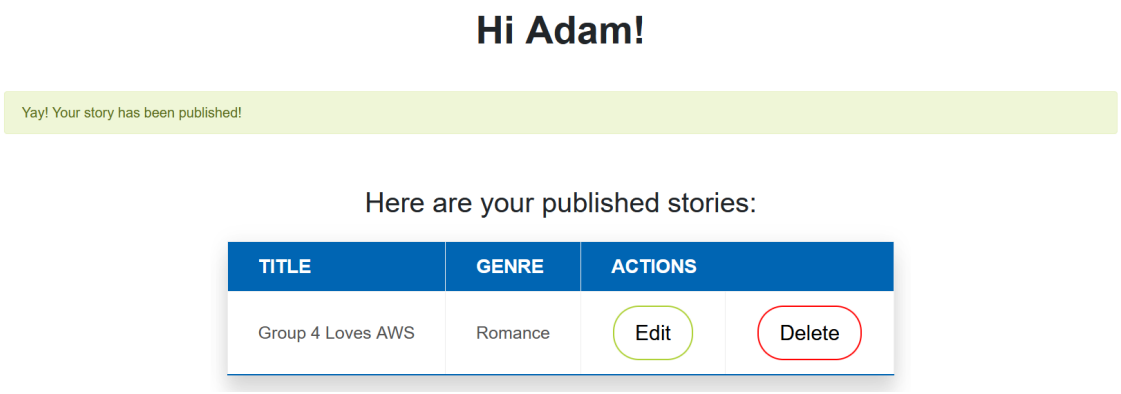


Figure 3.7: *Digital-Ink* account page.

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.

[width=150mm]resources/webapp/digital-ink-stories
[width=150mm]resources/webapp/digital-ink-story

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 (**amazon2022what**). A VPC can have multiple assigned subnets, which are a range of IP addresses accessible in the VPC.

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 4.1.

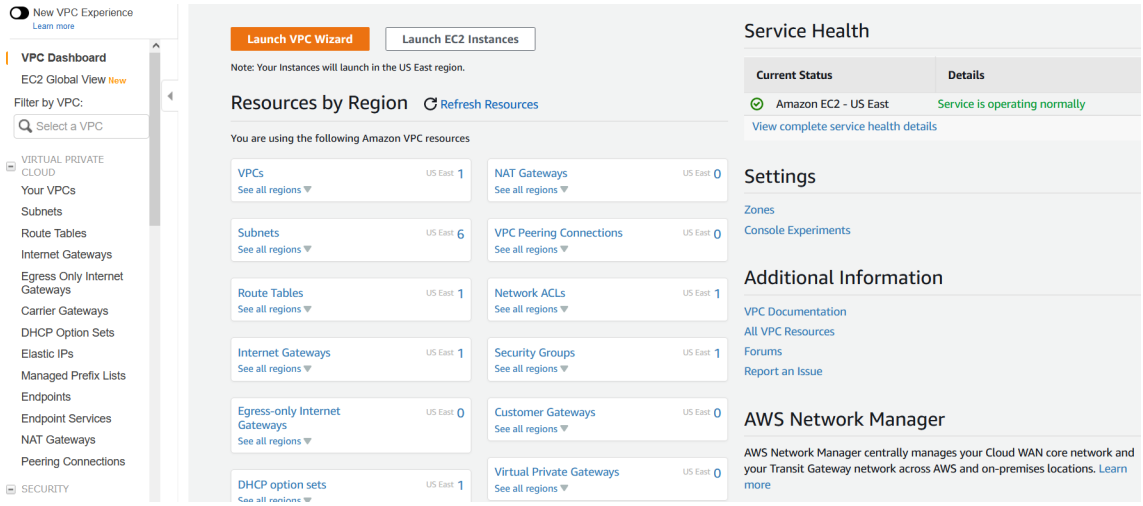


Figure 4.1: VPC Management Console.

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 4.2.

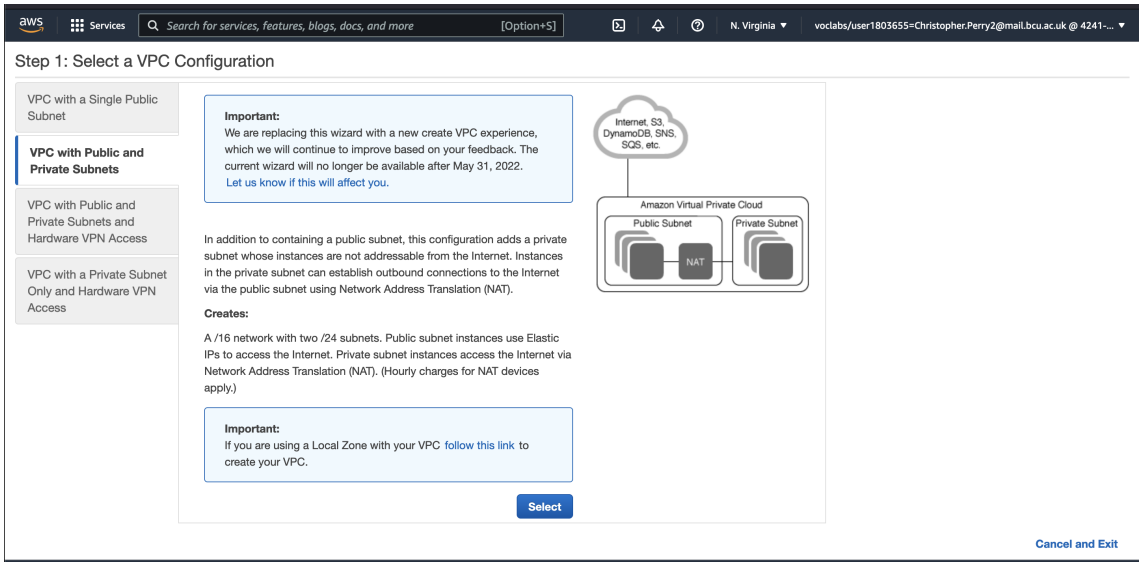


Figure 4.2: Selecting a VPC configuration.

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.

# 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 (**aws2022ec2**). 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.

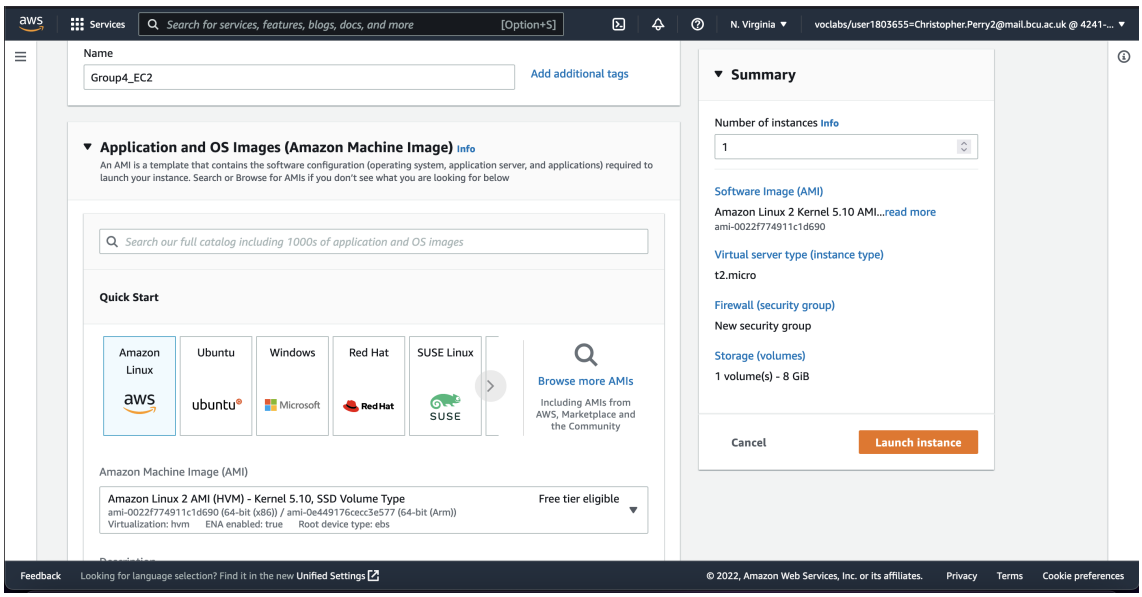


Figure 5.1: Selection of EC2 OS Image.

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.

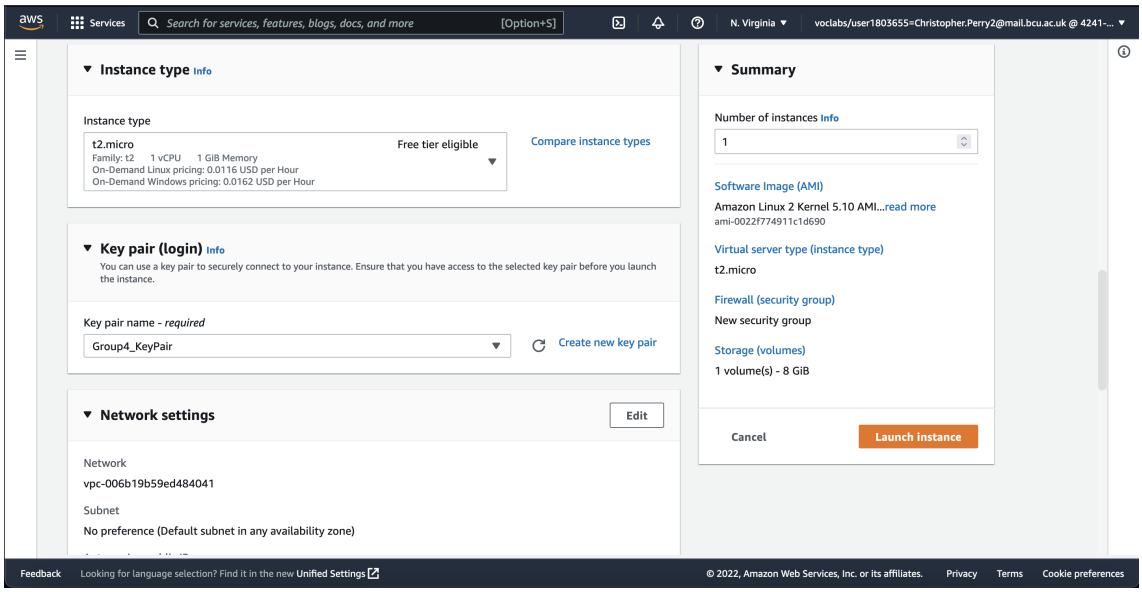


Figure 5.2: Selection of EC2 Instance.

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.

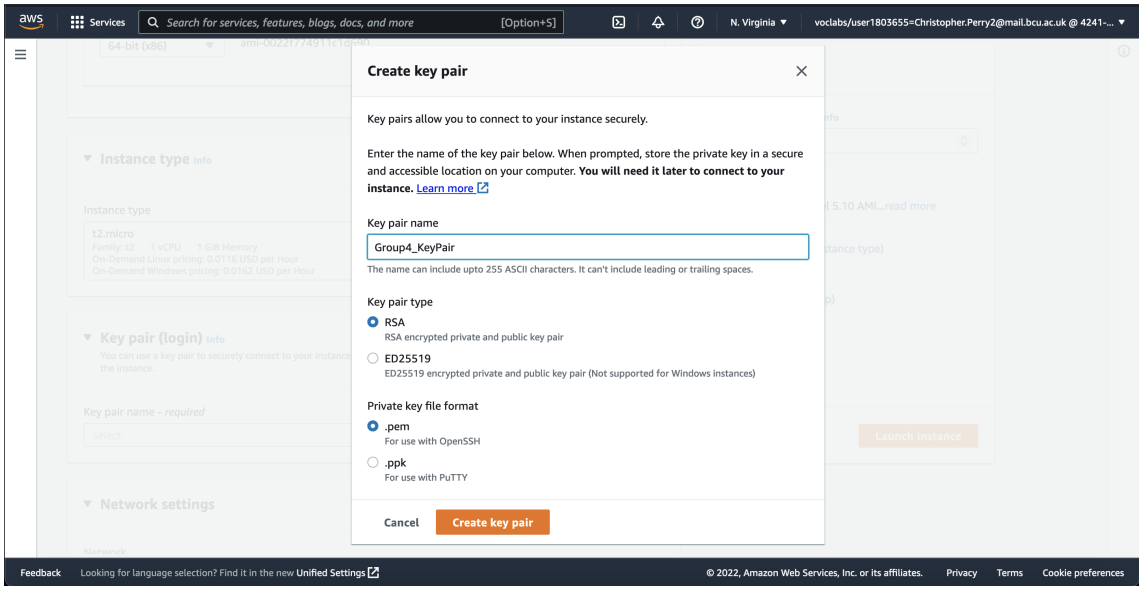


Figure 5.3: Selection of EC2 Keypair.

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

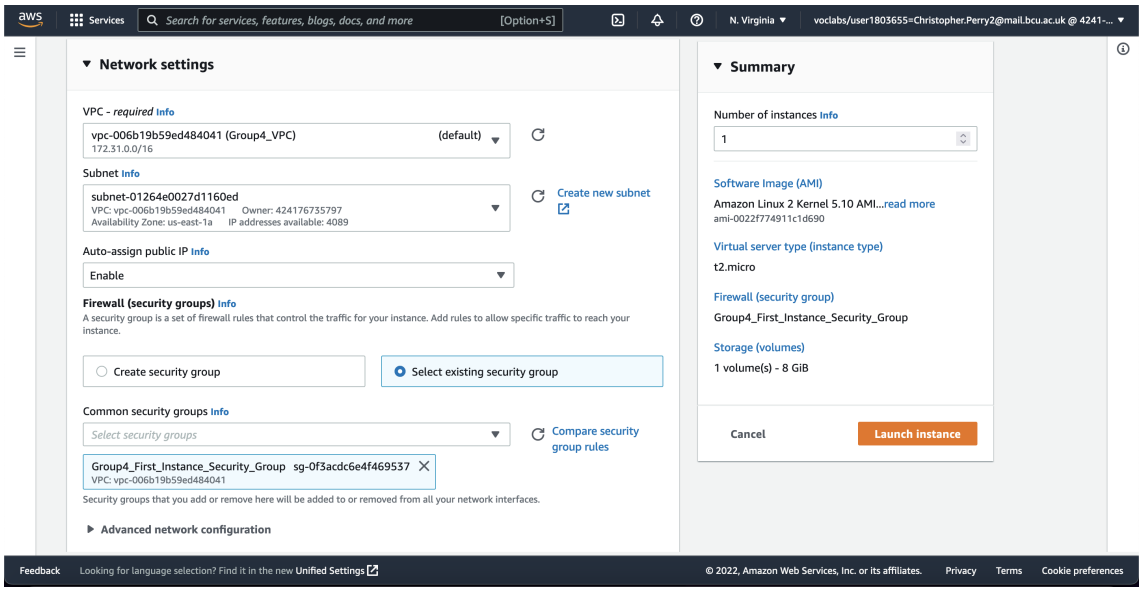


Figure 5.4: Selection of EC2 Networking options.

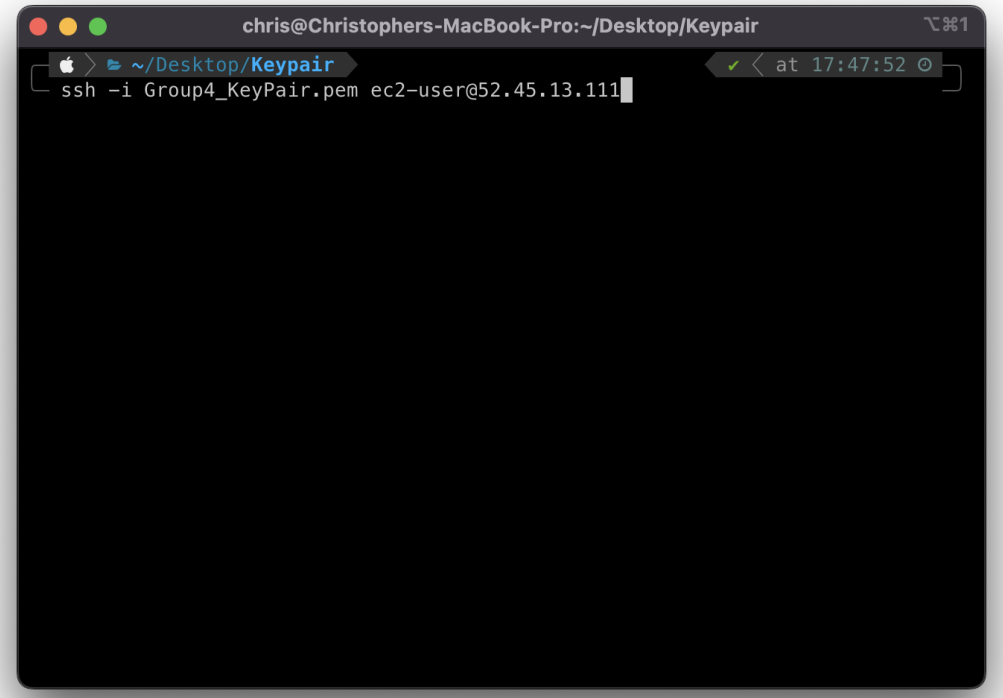


Figure 5.5: Generated EC2 Keypair in the `.pem` format.

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.

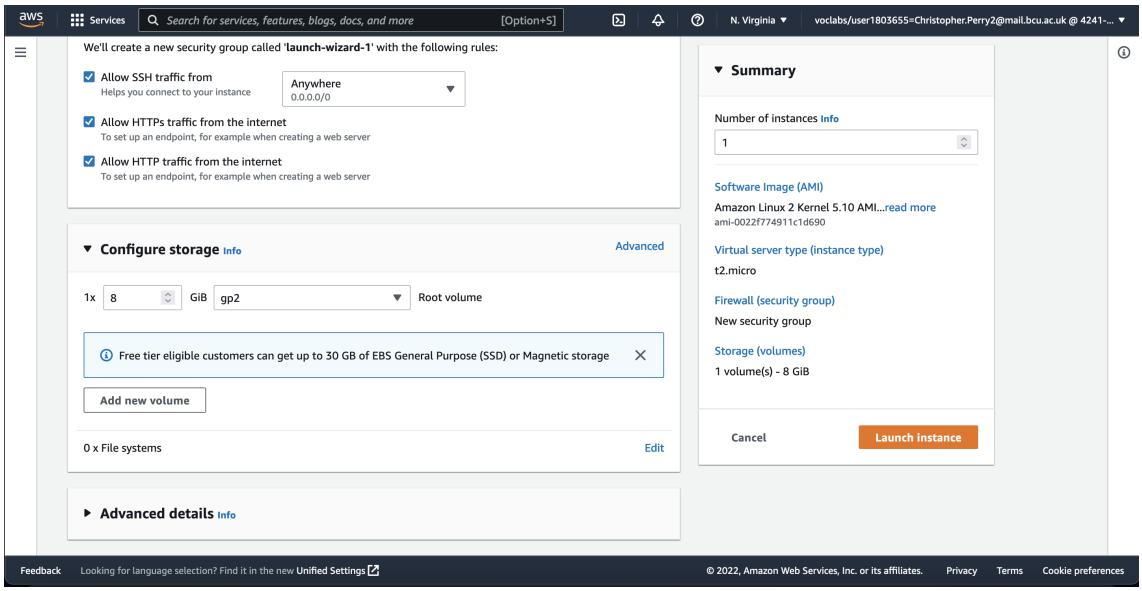


Figure 5.6: Selection of EC2 Storage Configuration.

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 webapp can be loaded onto it. The instance is first logged in to 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 instance.

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

Figure 5.7: SSH command to log into EC2 instance.

This command can seen being executed in Figure 5.7.

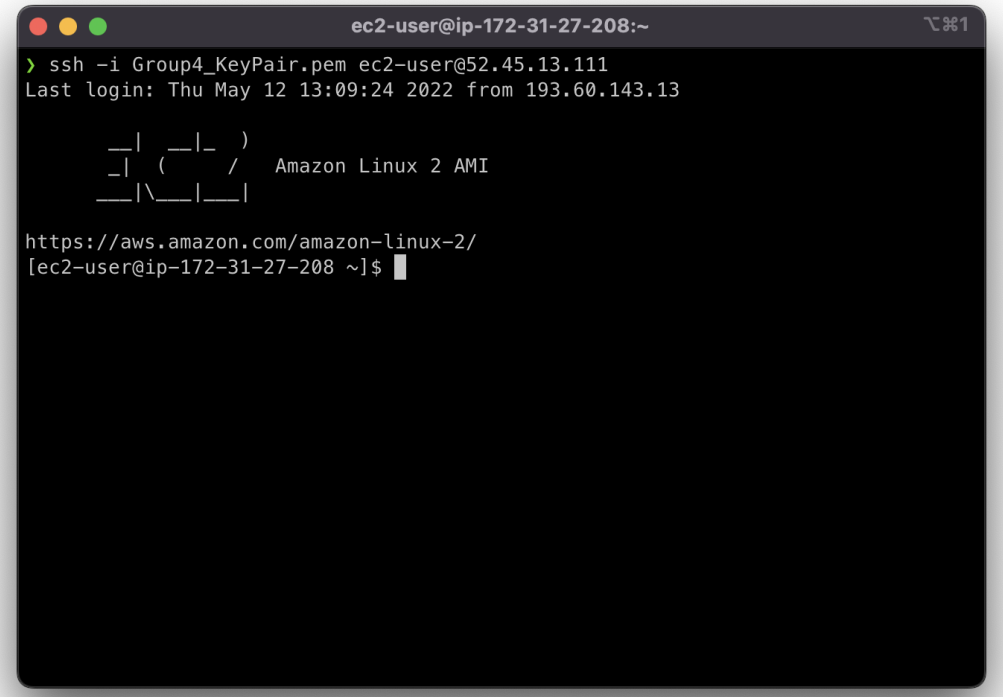


Figure 5.8: Logging into EC2 instance.

The logged in EC2 instance can be seen in Figure 5.8.

## 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`.

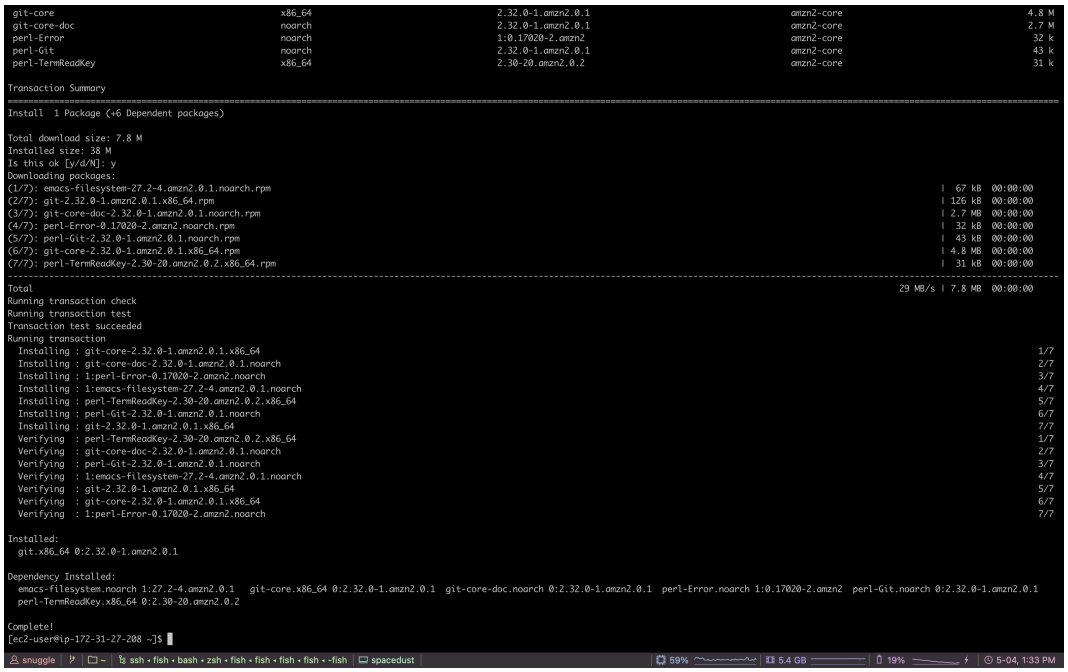


Figure 5.9: Installing Git.

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

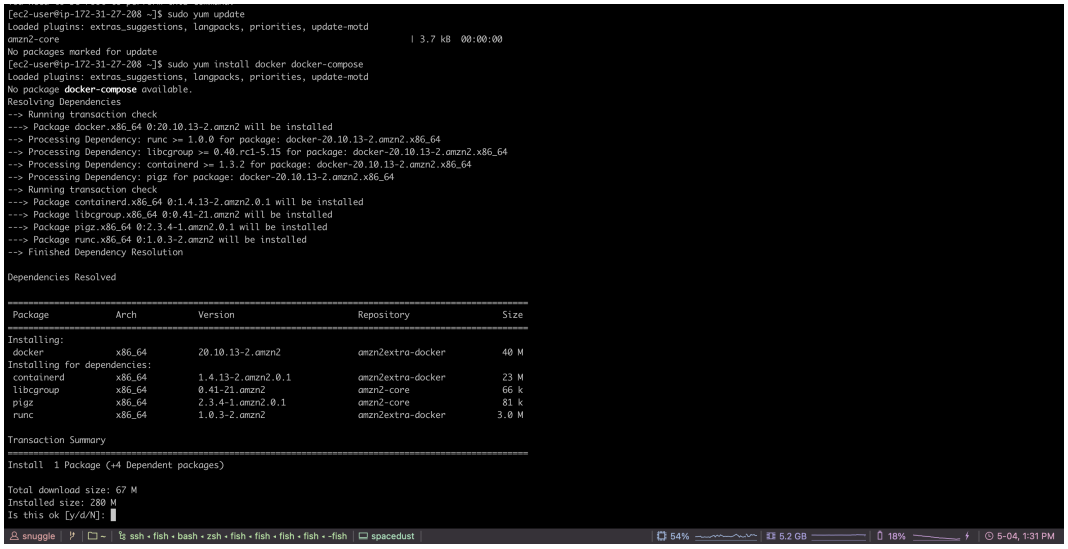


Figure 5.10: Installing Docker.

## 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.

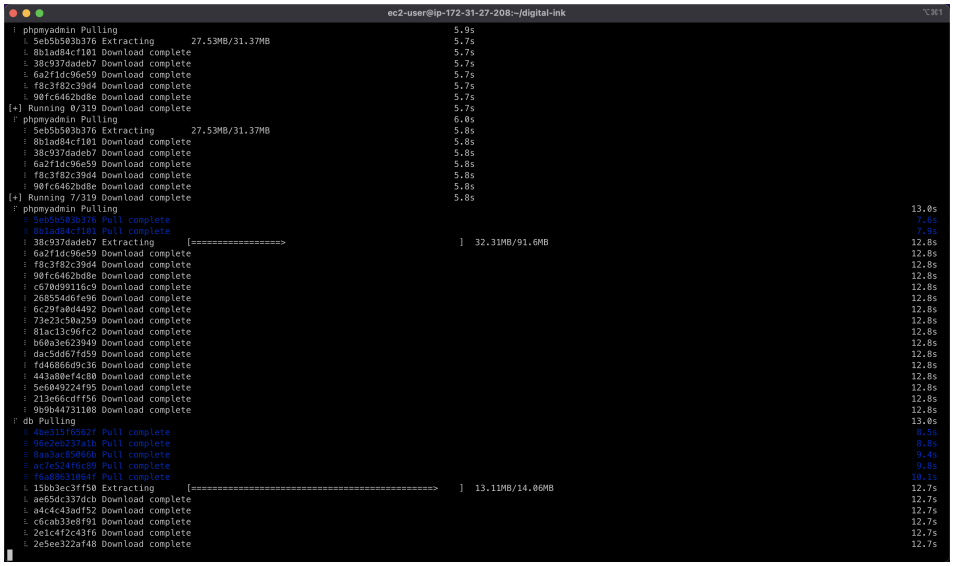


Figure 5.12: Containers required for the web app being pulled from Docker Hub.

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 , Digital Ink will now be shown.

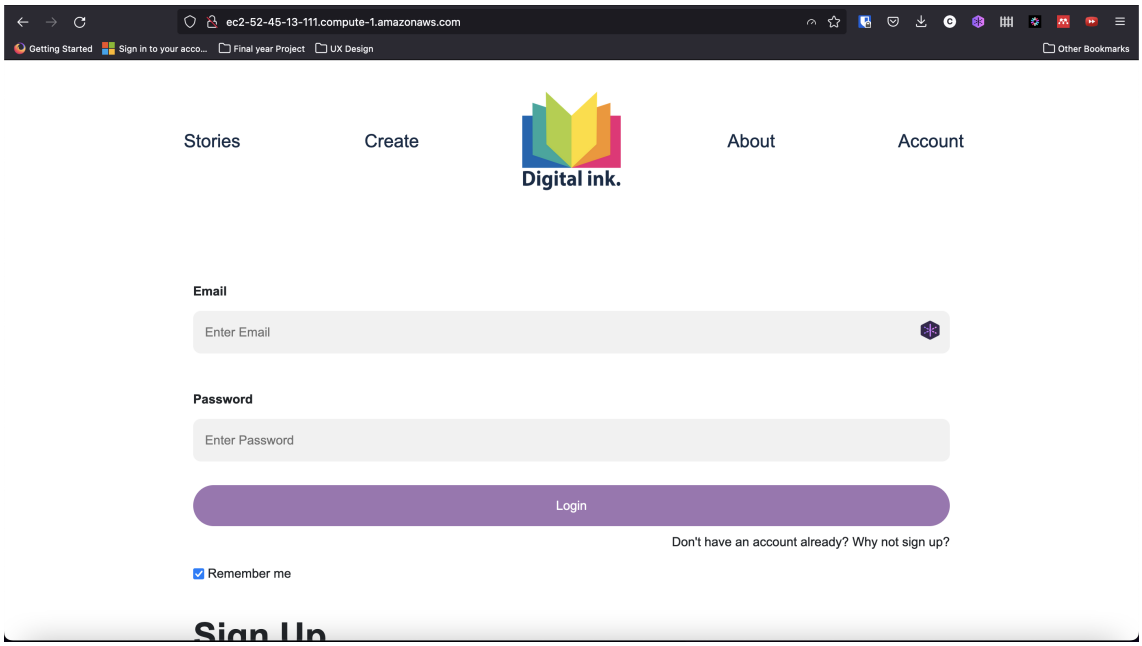


Figure 5.14: Digital Ink shown when accessed through the IPv4 address.

## 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.

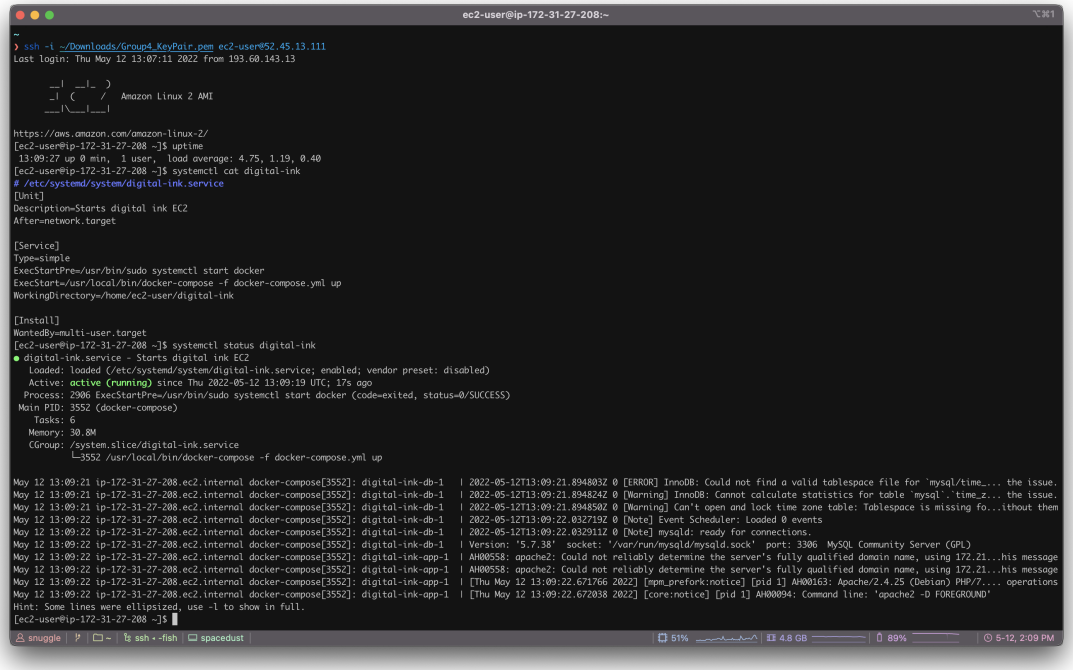


Figure 5.15: Creation of `systemd` Service.

# 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 each file as a seperate 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.

Each S3 instance is seperated into logical containers known as buckets. Each S3 bucket can have its own credentails, 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 different nodes, providing data redundancy against multiple different availability zones.

S3 is perhaps the most popular AWS service and used by many different 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.

# Chapter 7

# CloudFront

CloudFront will allow for the distribution of static and dynamic web images more efficiently. An international network of data centres, referred to as edge locations, are used to deliver content for CloudFront. 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.

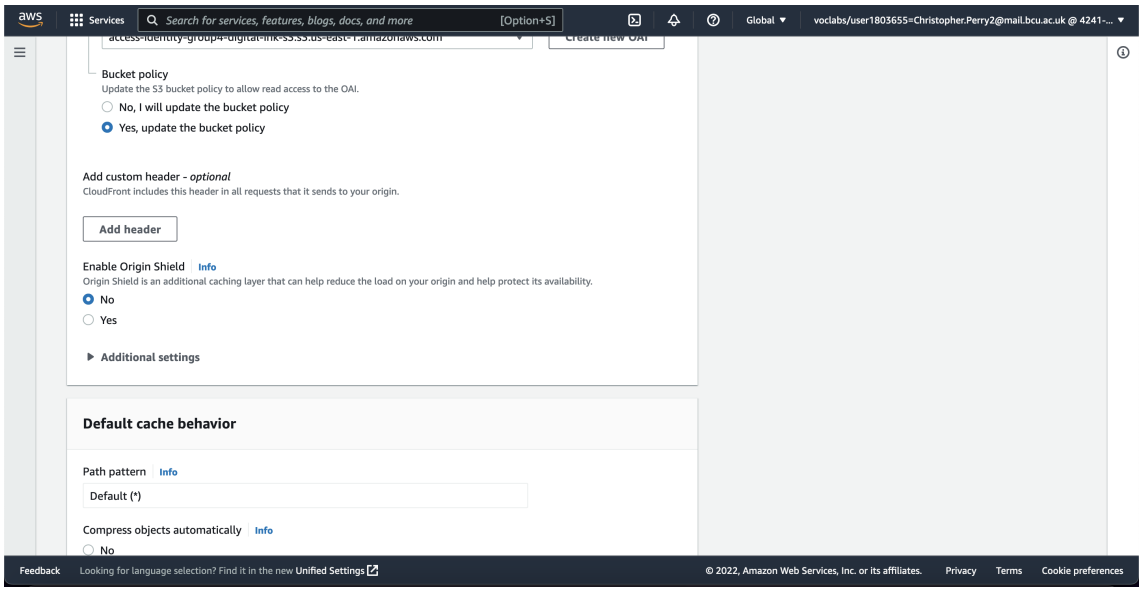


Figure 7.1: Applying CloudFront bucket policy.

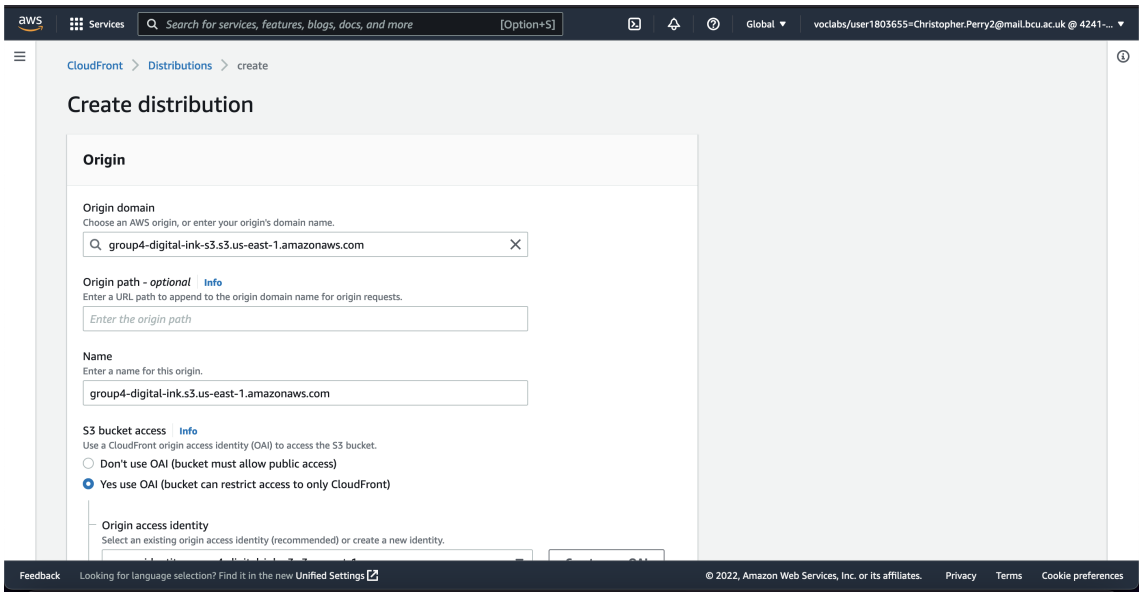


Figure 7.2: Applying CloudFront origins to S3 bucket.

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.

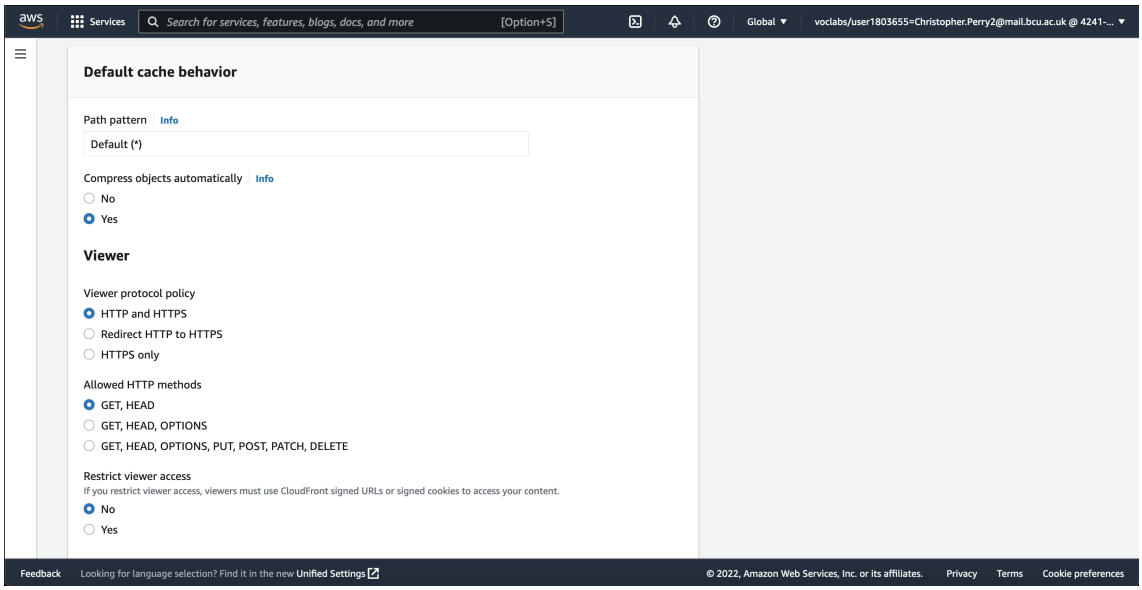


Figure 7.3: Applying CloudFront Distribution Permissions.

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".

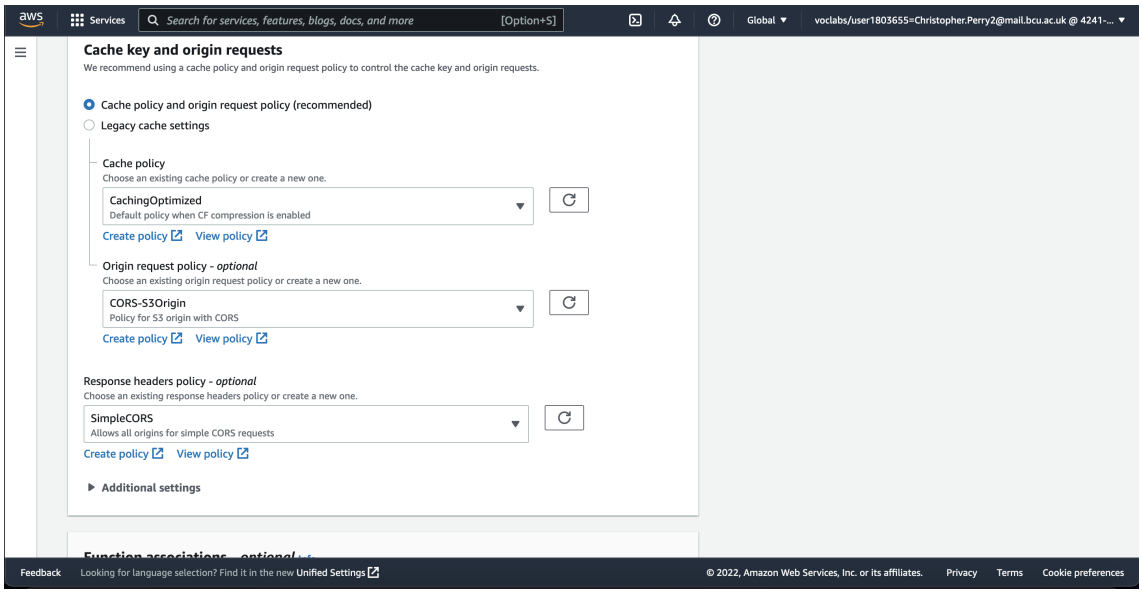


Figure 7.4: Applying CloudFront Cache Keys and Origin Requests.

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

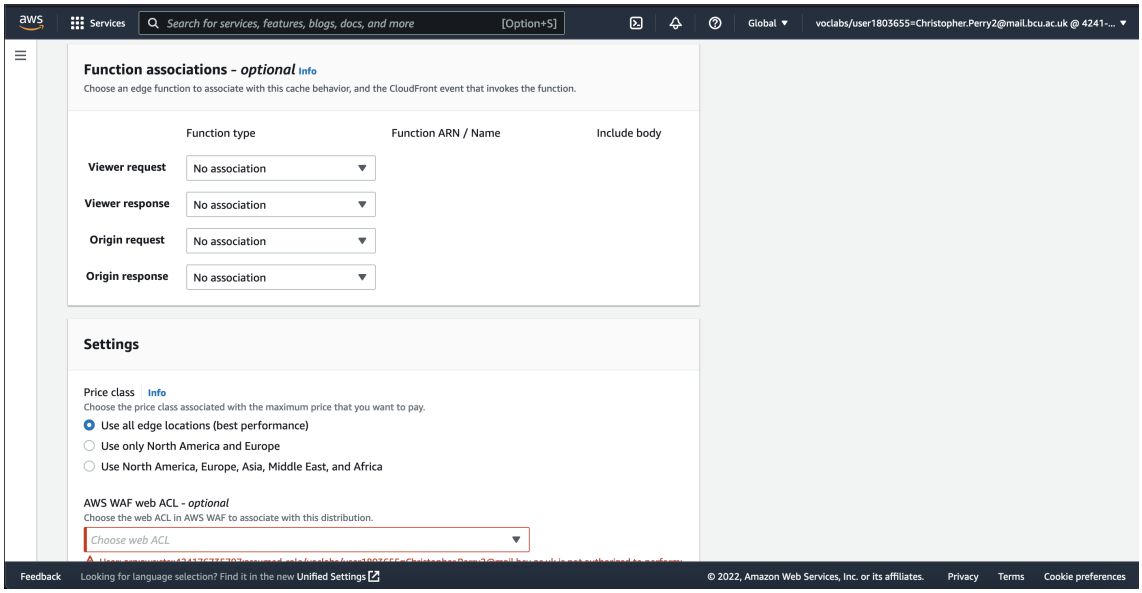


Figure 7.5: Function Association.

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 Figures **??** and 7.7.

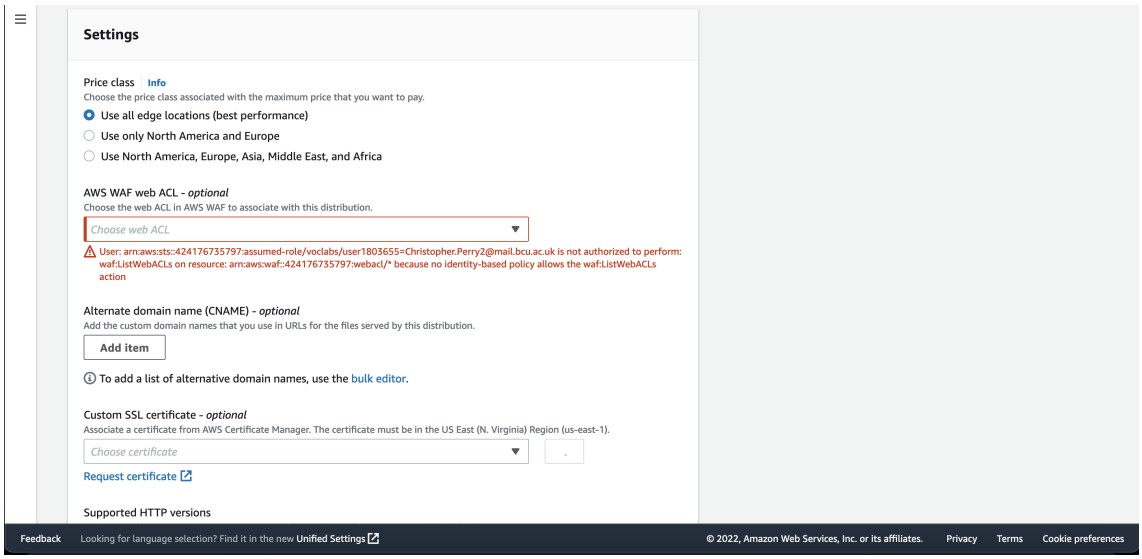


Figure 7.6: Applying CloudFront Settings.

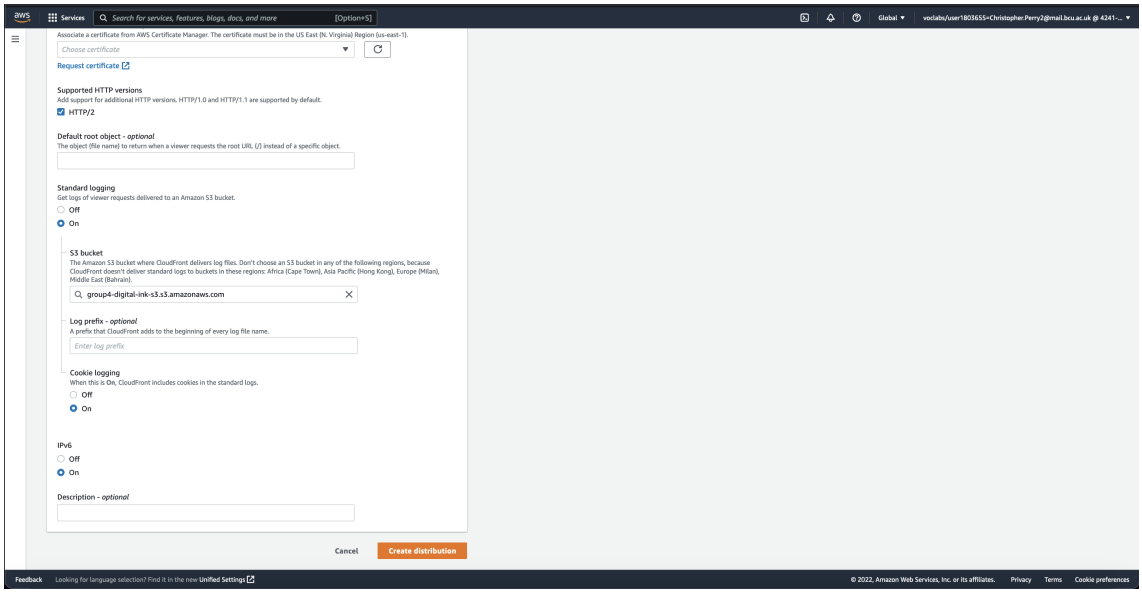


Figure 7.7: Applying CloudFront Settings.

A CloudFront Distribution has now been created, and can be found in Figure 7.8.

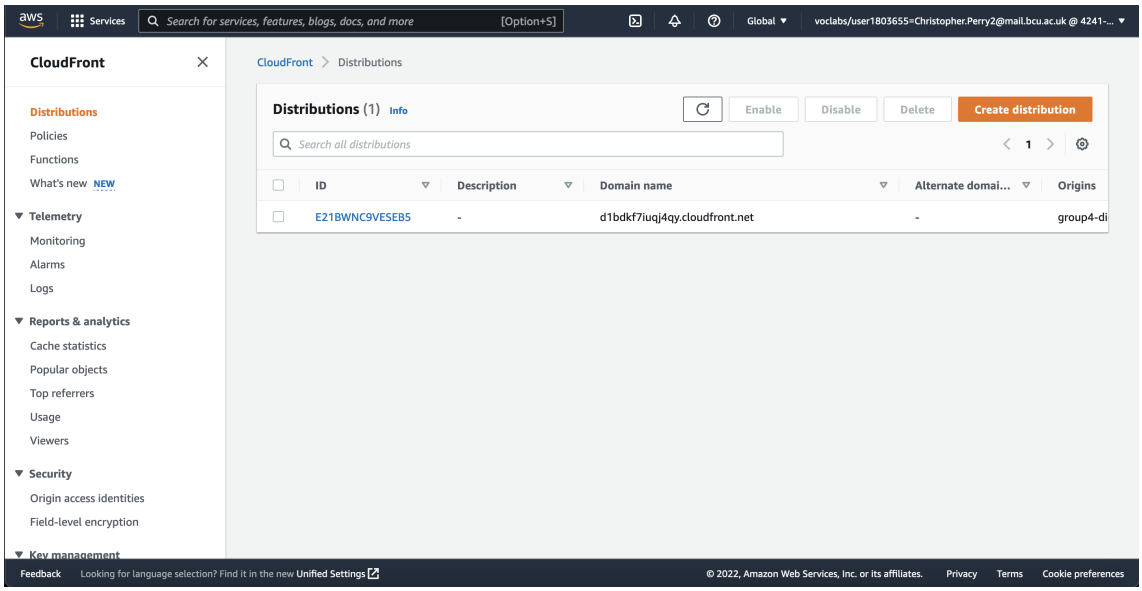


Figure 7.8: Created CloudFront Distribution.

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 7.9 and 7.10.

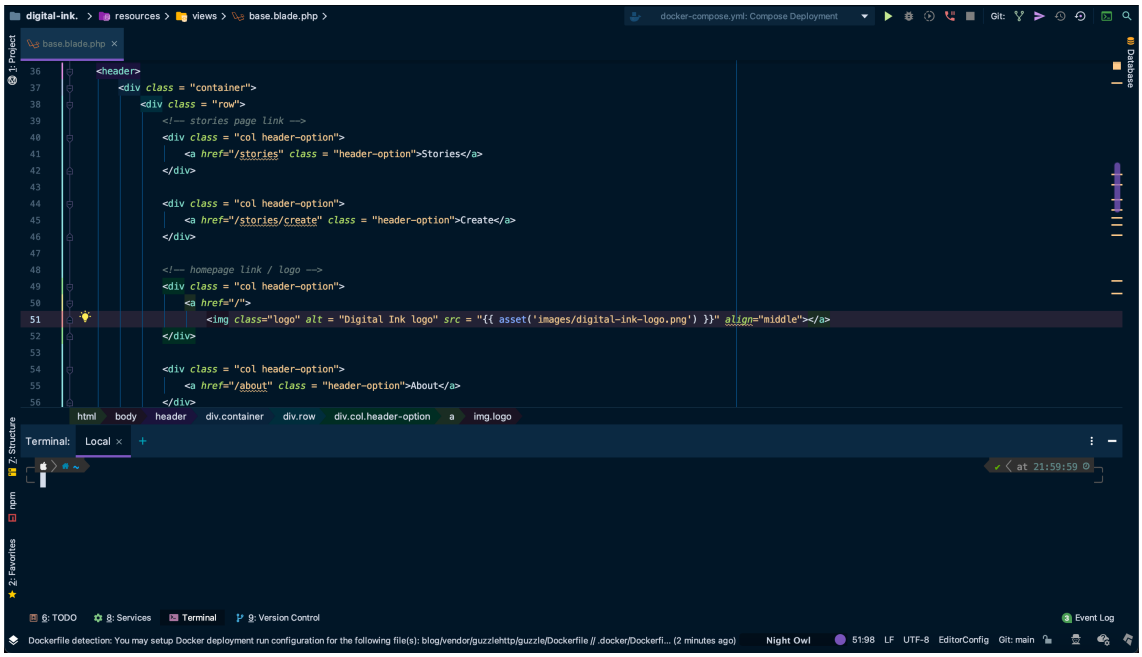


Figure 7.9: Image location before CloudFront.

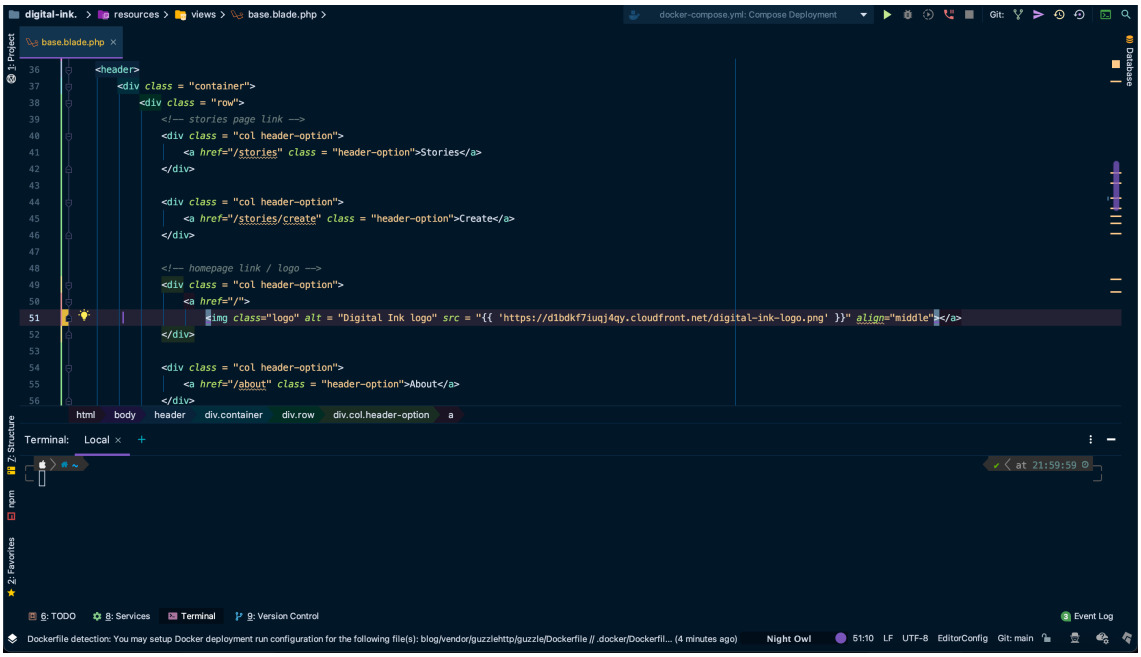


Figure 7.10: Image location after CloudFront.


Figure 7.11: Image location after CloudFront.

As seen in Figure 7.11, the Digital Ink logo is now hosted on 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 (**amazon2022amazon**). 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.

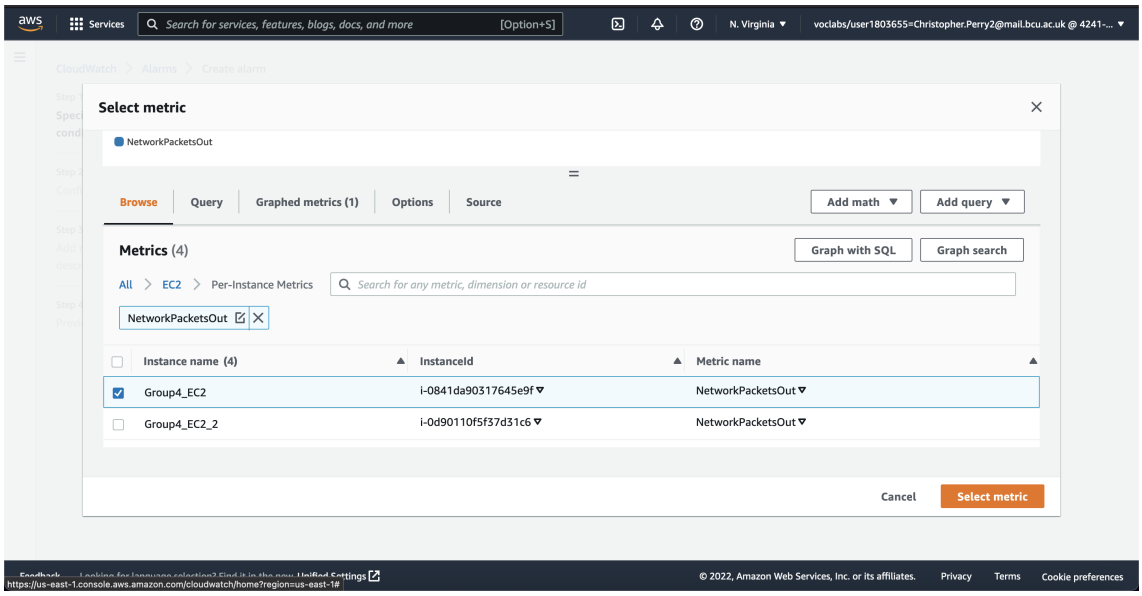


Figure 8.1: Selection of CloudWatch metric for EC2 instance.

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.

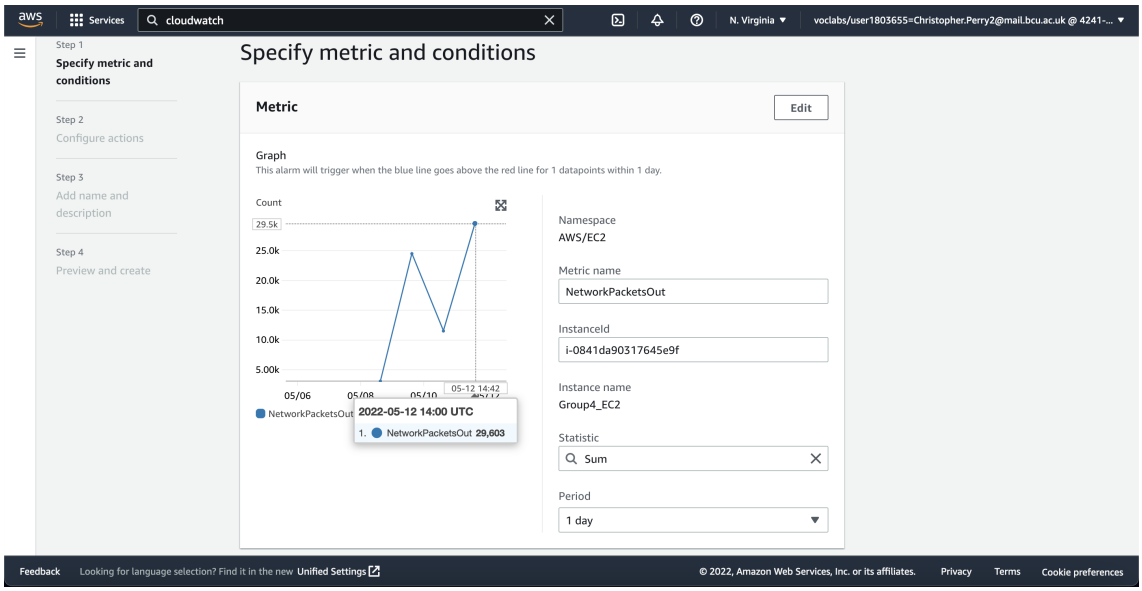


Figure 8.2: Configuration of NetworkPacketsOut Metric.

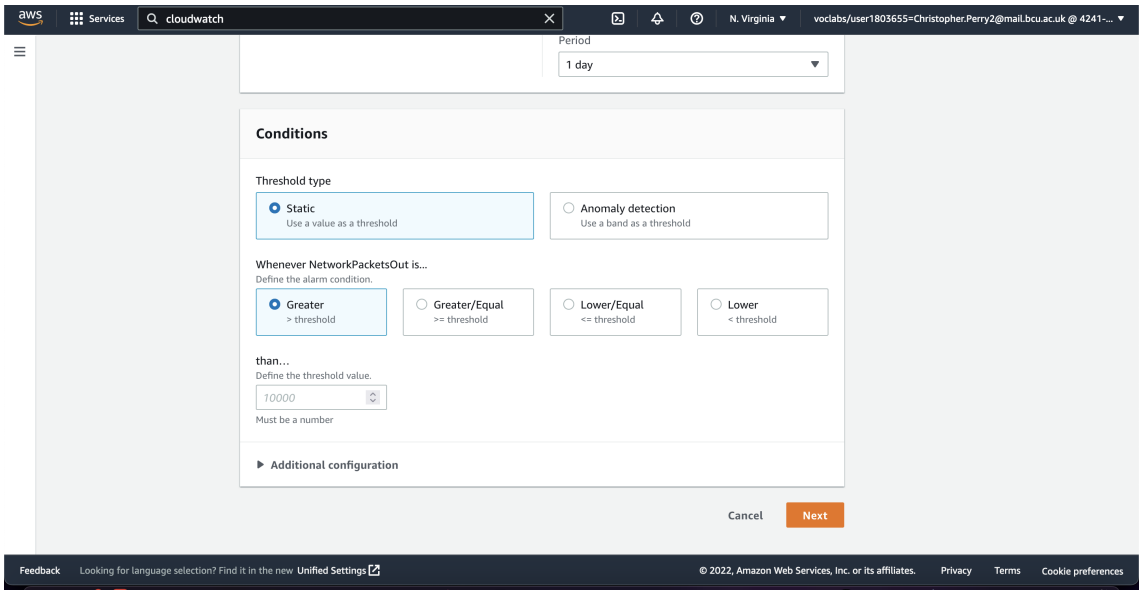


Figure 8.3: Configuration of NetworkPacketsOut Metric.

Figures 8.2 and 8.3 detail the alarm being set so that when the sum of packets sent is less than 5 per day, the alarm will activate.

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 8.4 details this process.

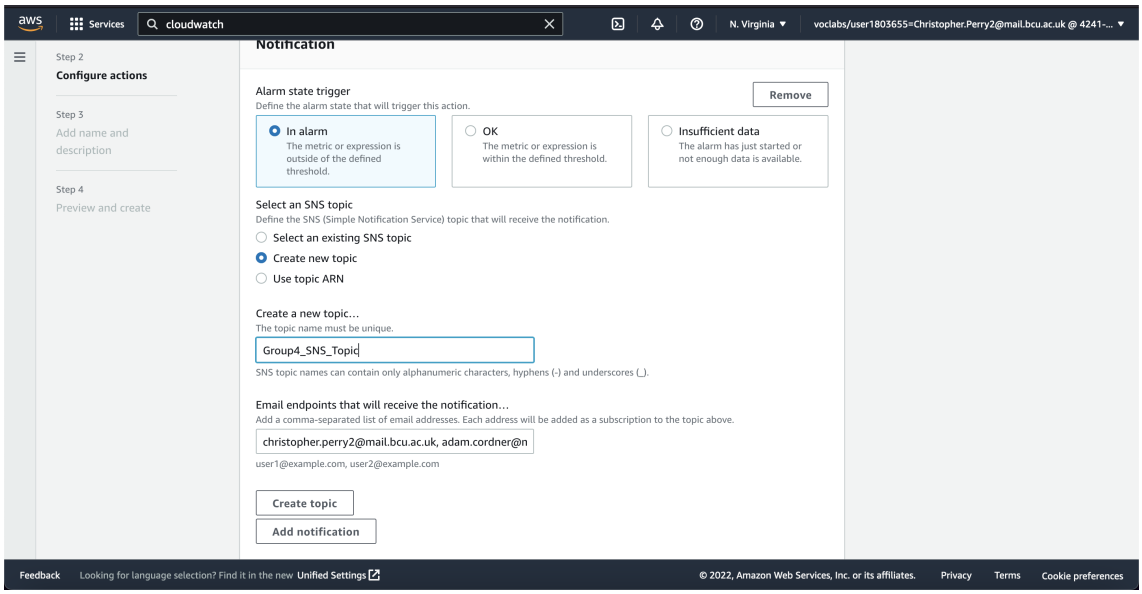


Figure 8.4: Configuration of SNS Topic for email alerts on alarm activation.

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 8.5 and 8.6.

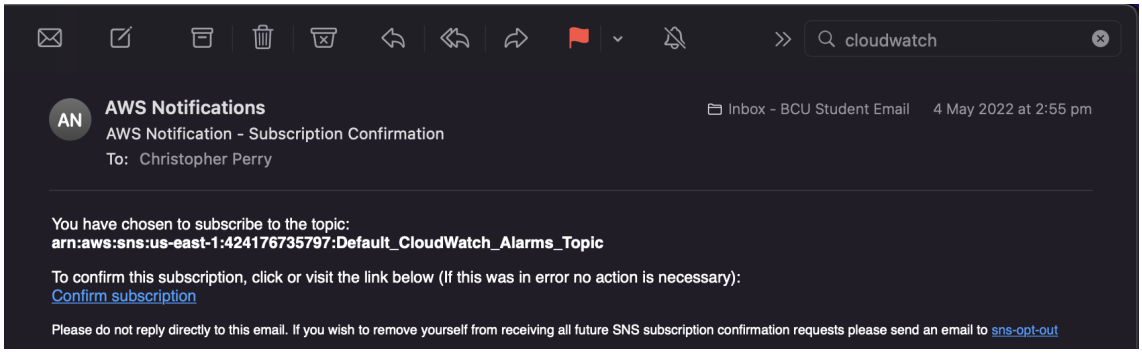


Figure 8.5: CloudWatch SNS Topic Email.

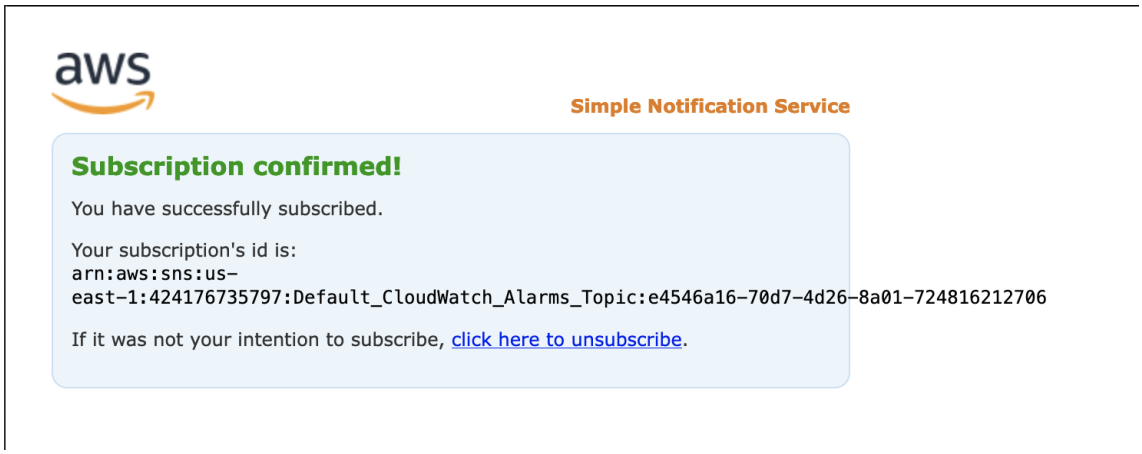


Figure 8.6: Successful subscription to SNS topic.

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 8.7.

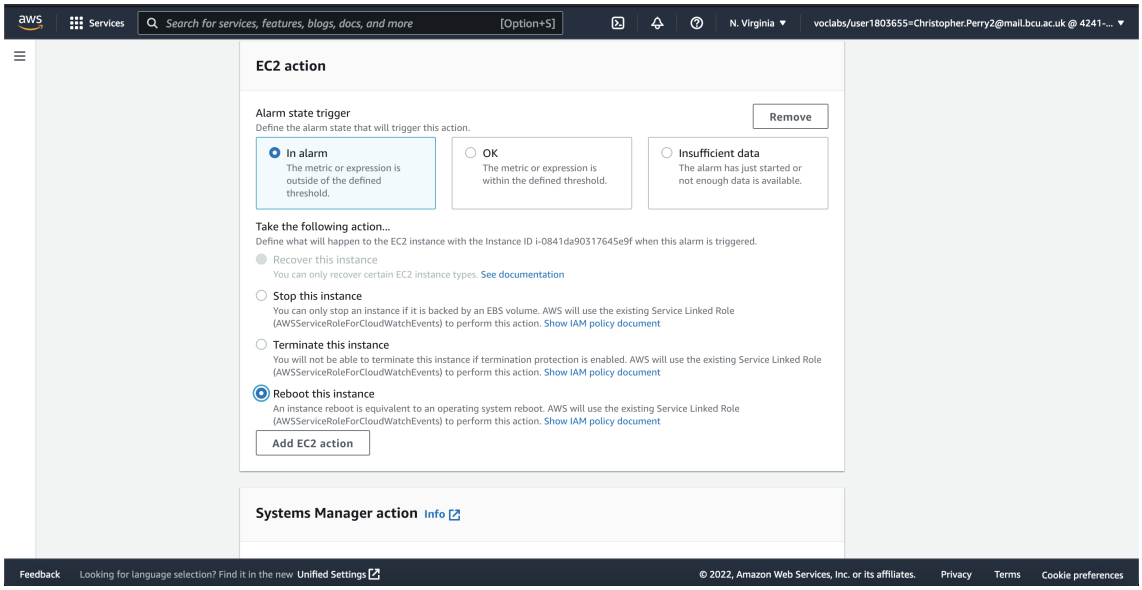


Figure 8.7: Configuration for EC2 instance to reboot on alarm activation.

A brief description of the CloudWatch alarm is then added. This can be seen in Figure 8.8.

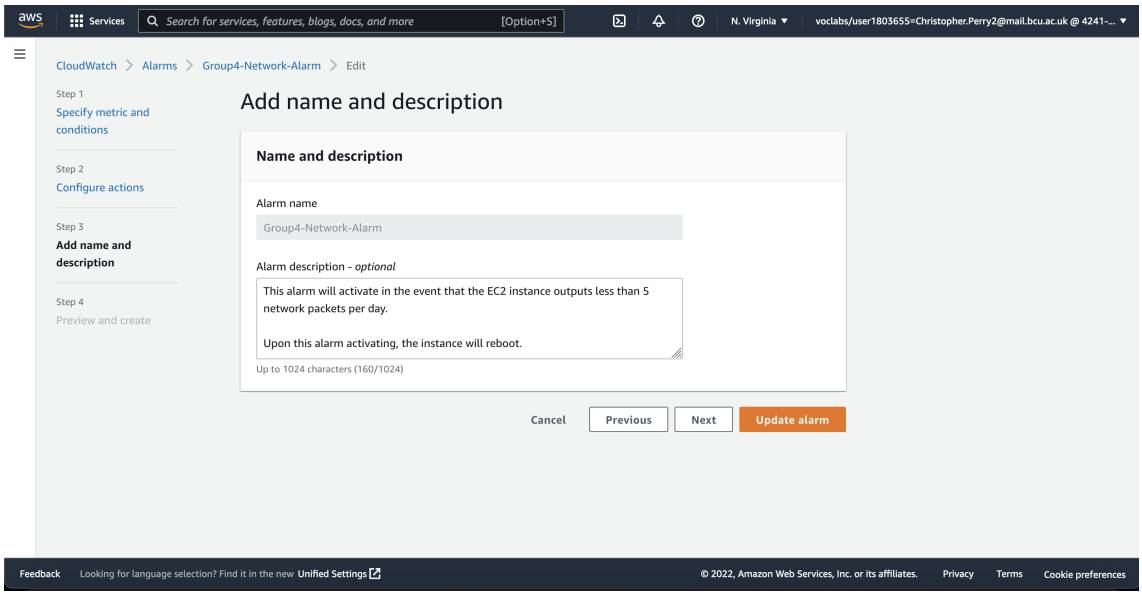


Figure 8.8: CloudWatch alarm description.

The alarm has now been set up, and can be seen in the CloudWatch Management Dashboard in Figure 8.9.

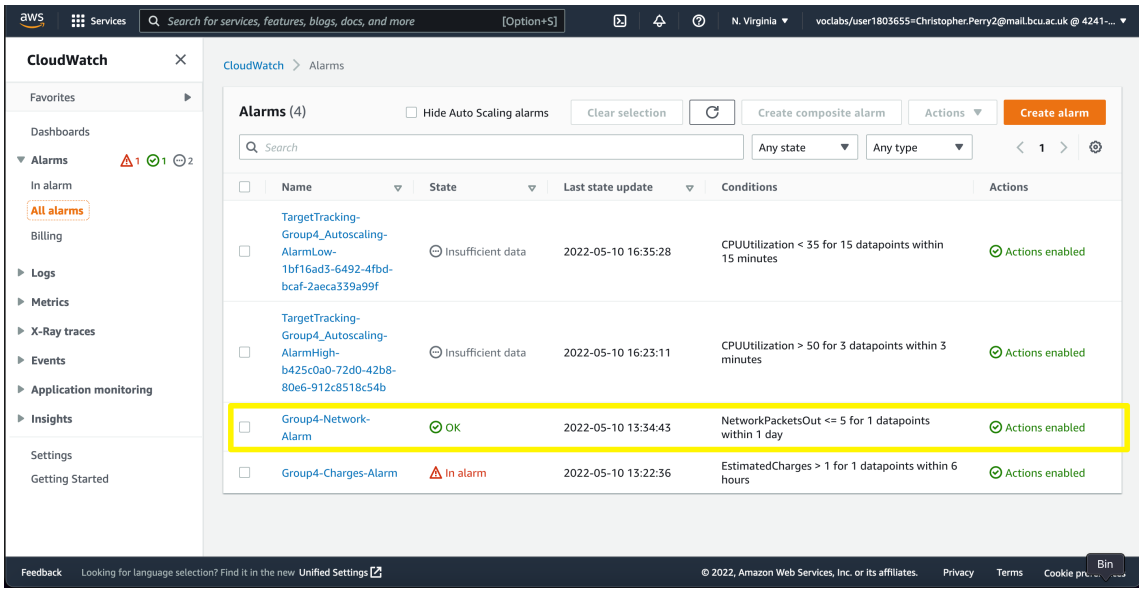


Figure 8.9: CloudWatch network alarm in dashboard.

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 8.10.

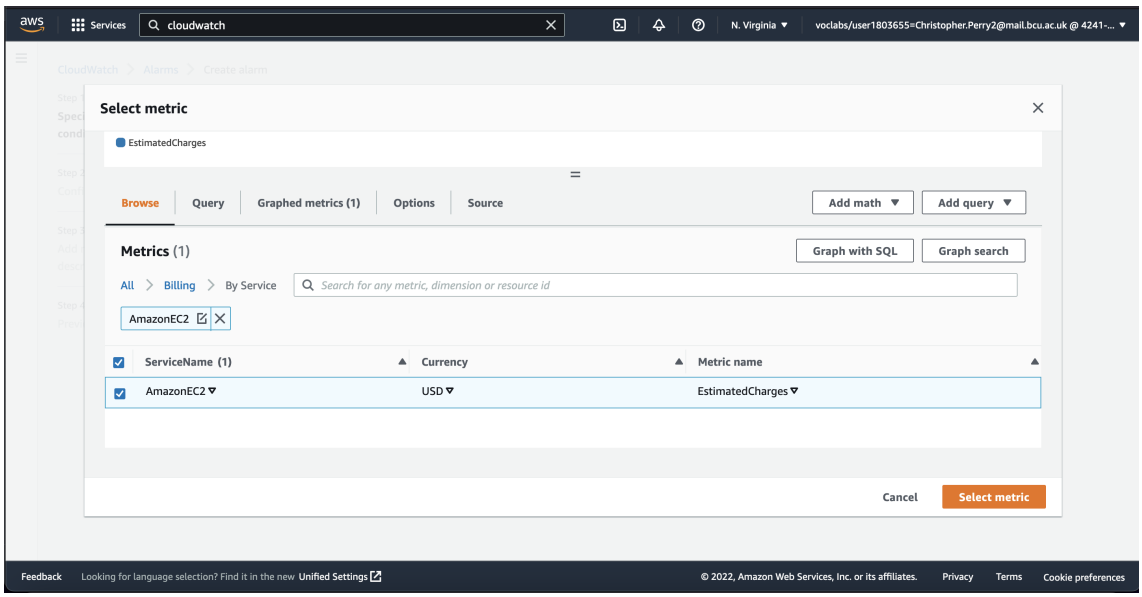


Figure 8.10: Selection of `EstimatedCharges` CloudWatch metric.

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 Figures 8.11 and 8.12.

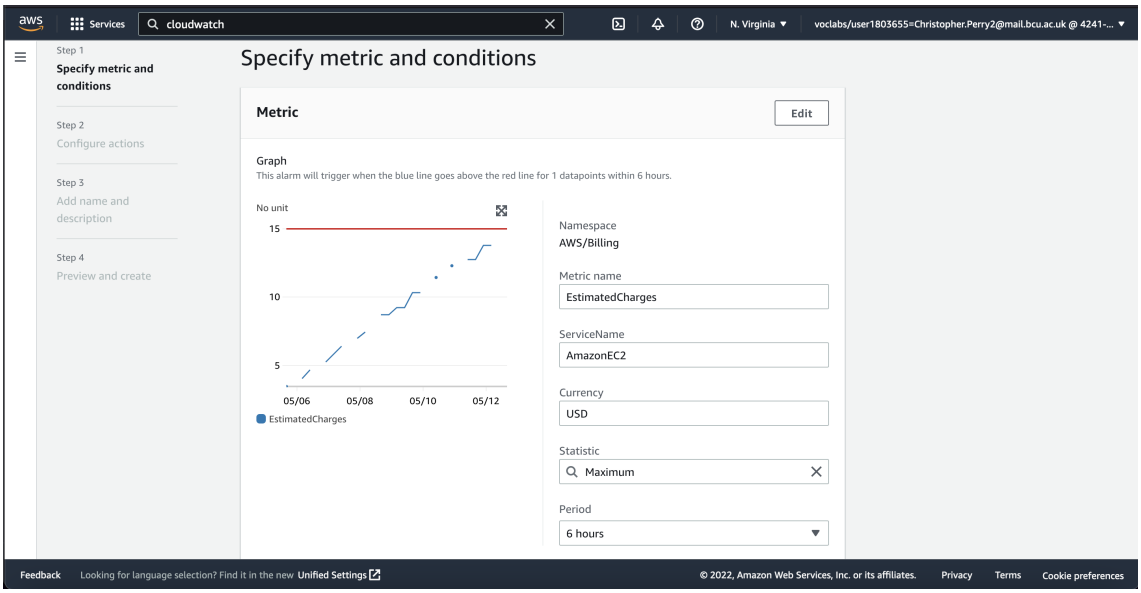


Figure 8.11: Configuration of CloudWatch alarm.

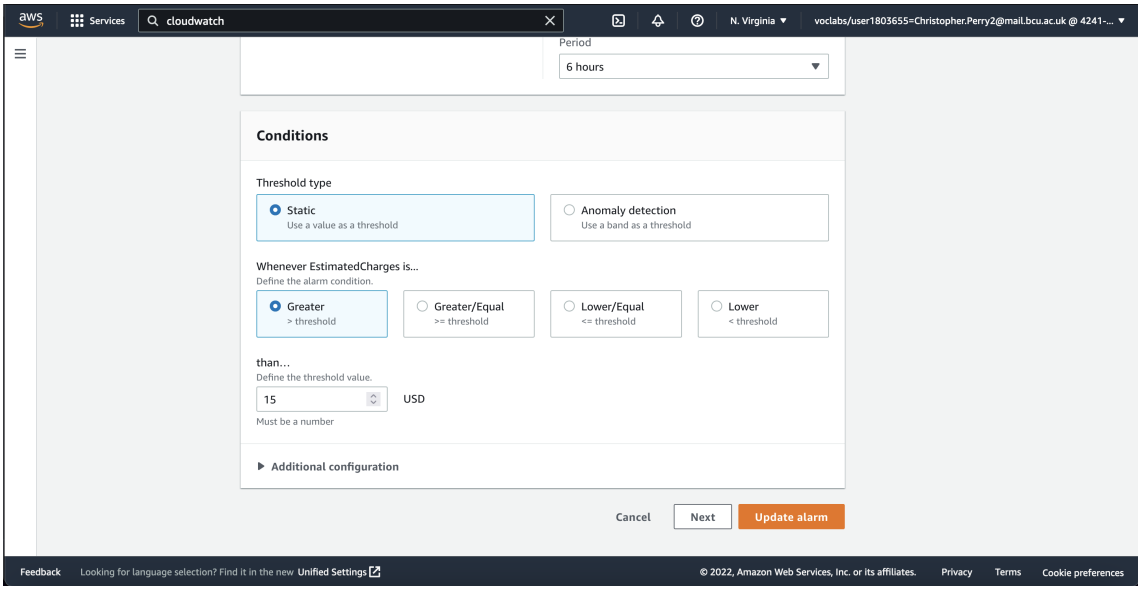


Figure 8.12: Configuration of CloudWatch alarm.

In addition to this, alerts will be sent to the same SNS topic configured earlier. This can be seen in Figure 8.13.

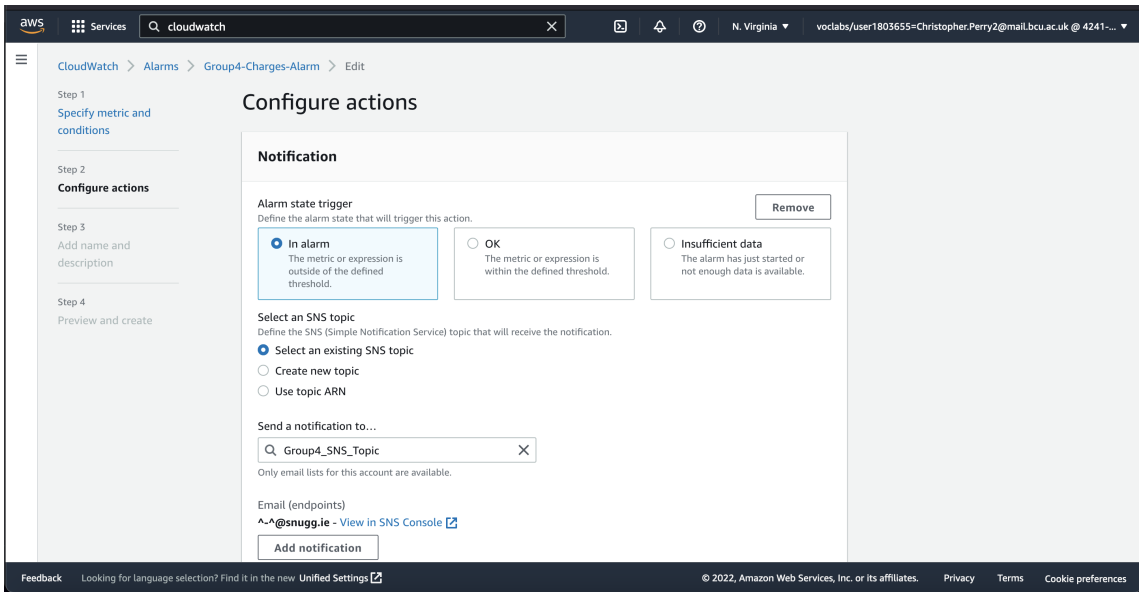


Figure 8.13: Setting CloudWatch charges SNS topic

A brief description was added to the alarm, as seen in Figure 8.14.

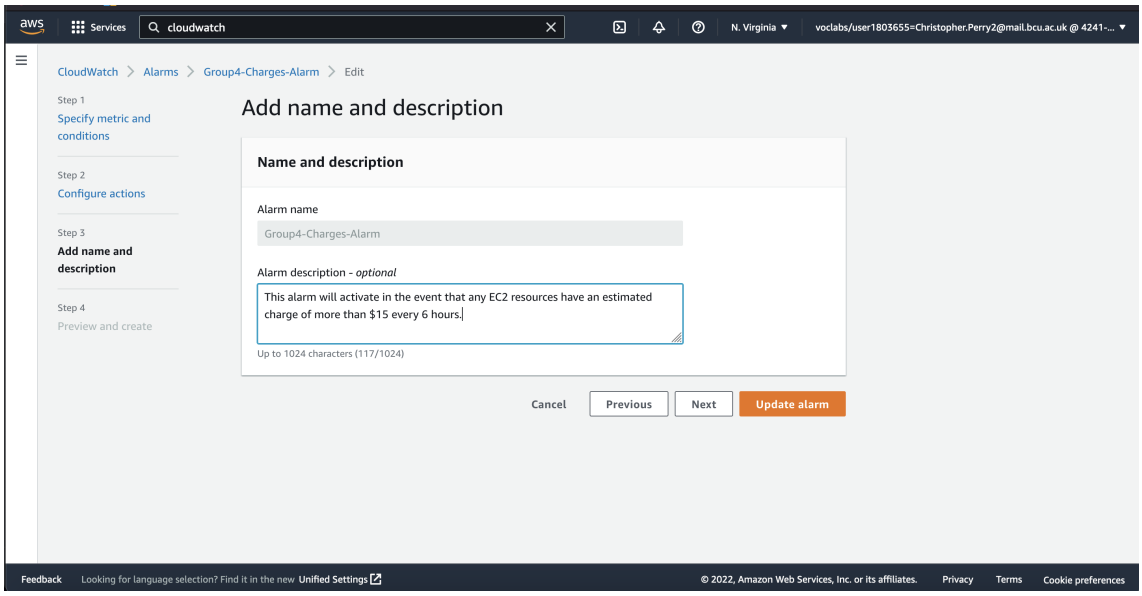


Figure 8.14: Description of CloudWatch alarm.

The alarm is now live and can be seen in Figure 8.15.

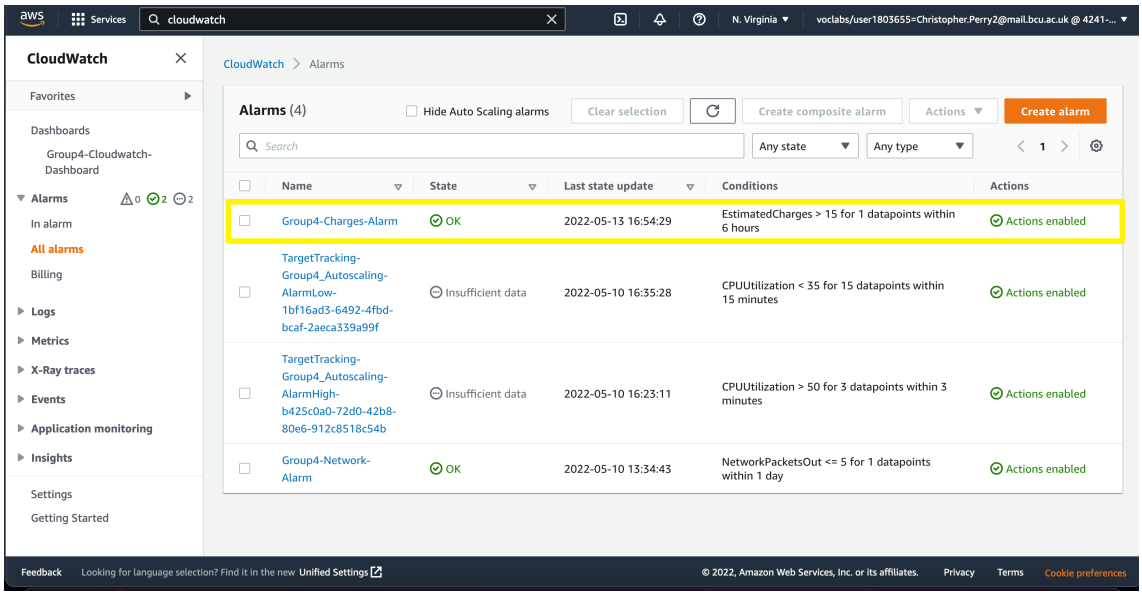


Figure 8.15: CloudWatch charges alarm live in CloudWatch dashboard.

# Chapter 9

# CloudTrail

# 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.

# Chapter 11

# Availability Zones

# Chapter 12

# 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.

# Chapter 13

# Security Practices

# Chapter 14

# 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 (**amazon2022aws**). 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.

## 14.1 Estimated Costs

## 14.2 Scaling Up to 10,000 Users

## 14.3 Scaling Up to One Million Users

## 14.4 Scaling Up to Ten Million Users

# Chapter 15

# 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 (**dos2018automated**). 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 **??**.

```
Scenario: ...
    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.

## 15.1 Testing EC2

```
Scenario: Accessing instance through SSH with .pem file private key.
    Given ...
    When ...
    Then ...
Scenario: Accessing web app through EC2 domain name.
    Given ...
    When ...
    Then ...
```

## 15.2 Testing S3

```
Scenario: Accessing web app image through S3 domain name.
    Given ...
    When ...
    Then ...
```

## 15.3 Testing CloudFront

## 15.4 Testing RDS

```
Scenario: Accessing web app image through CloudFront domain name.
    Given ...
    When ...
    Then ...

Scenario: Accessing web app image through CloudFront domain name in another region.
    Given ...
    When ...
    Then ...

Scenario: Accessing web app image through CloudFront domain name in another IP address.
    Given ... (NSLookup test)
    When ...
    Then ...

Scenario: Creating user information through the web app.
    Given ...
    When ...
    Then ...

Scenario: Creating story information through the web app.
    Given ...
    When ...
    Then ...

Scenario: Reading user information from the database into the web app.
    Given ...
    When ...
    Then ...

Scenario: Reading story information from the database into the web app.
    Given ...
    When ...
    Then ...

Scenario: Updating user information in the database through the web app.
    Given ...
    When ...
    Then ...

Scenario: Updating story information in the database through the web app.
    Given ...
    When ...
    Then ...

Scenario: Deleting user information in the database through the web app.
    Given ...
    When ...
    Then ...

Scenario: Deleting story information in the database through the web app.
    Given ...
    When ...
    Then ...
```

## 15.5 Testing CloudWatch

(One test for each of the metrics we set up.)

```
Scenario:
    Given ...
    When ...
    Then ...
```

## 15.6 Testing CloudTrail

(One test for each of the metrics we set up.)

```
Scenario:
    Given ...
    When ...
    Then ...
```

## 15.7 Testing ELB

(Test for turning off instance 1. Test for turning off instance 2.)

# Chapter 16

# Future Enhancements

An SSL/TLS certificate could be added, as it was not possible to do this from within the lab tutorial due to a lack of permissions from the root AWS account. By adding a certificate we would be able to encrypt the connections to our EC2 instance and increase the security of our application for all users.

We would like to add more roles and users to our IAM, but were limited by the amount that we were able to access.

If we were to do the project again, we would use AWS Elastic Beanstalk as it allows automatic orchestration of all the resources needed for a LAMP stack style application such as ours, it also has support for Docker which is a containerization technology that we had used in our application project.

# Chapter 17

# Conclusion

# Appendix A: Additional Screenshots

TODO: I am an appendix, please be kind.