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

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 2

Glossary

- **SSH** — Secure SHell
- **EC2** — Elastic Cloud Compute
- **SSH** — Secure SHell
- **AWS** — Amazon Web Services
- **S3** — Simple Storage Service
- **ELB** — Elastic Load Balancer
- **LAMP** — Linux, Apache, MySQL, PHP
- **HTTP** — HyperText Transfer Protocol
- **SQL** — Structured Query Language
- **VPC** — Virtual Private Cloud
- **AMI** — Amazon Machine Images
- **RAM** — Random Access Memory
- **CPU** — Central Processing Unit
- **SNS** — Simple Notification Service
- **RDS** — Relational Database Service
- **DB** — Database
- **AZ** — Availability Zone
- **IEEE** — Institute of Electrical and Electronics Engineers
- **OS** — Operating Systems
- **IP** — Internet Protocol
- **GB** — Gigabyte
- **IAM** — IANdsfoins

- **ACL** — Access Control List
- **ACM** — AWS Certificate Manager
- **CA** — Certificate Authority
- **CloudFront** — Certificate Authority
- **CloudWatch** — Certificate Authority
- **ML** — Machine Learning
- **ML** — Machine Learning

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 (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. As shown in Figure 3.1, the database which this web app uses consists of three tables: `users`, `stories`, and `migrations`.

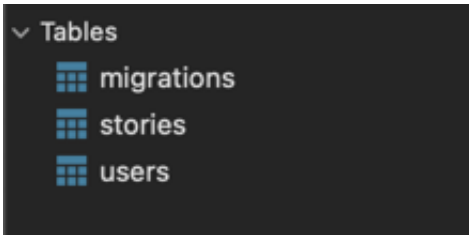

Figure 3.1: Database tables overview.

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 (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.

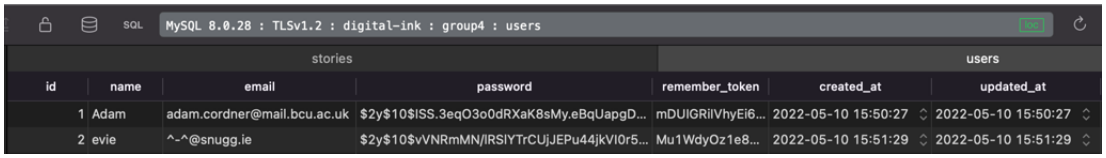

Figure 3.3: `users` table.

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/7537329101652200606.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.

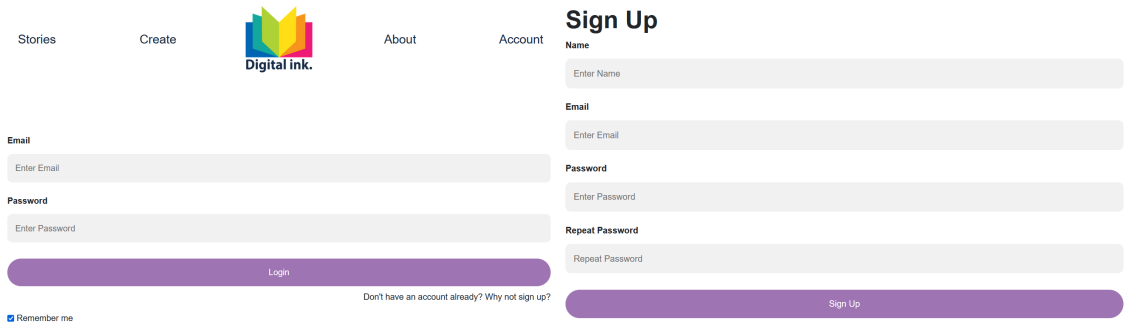

Figure 3.5: *Digital-Ink* home page log in and sign up forms.

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.

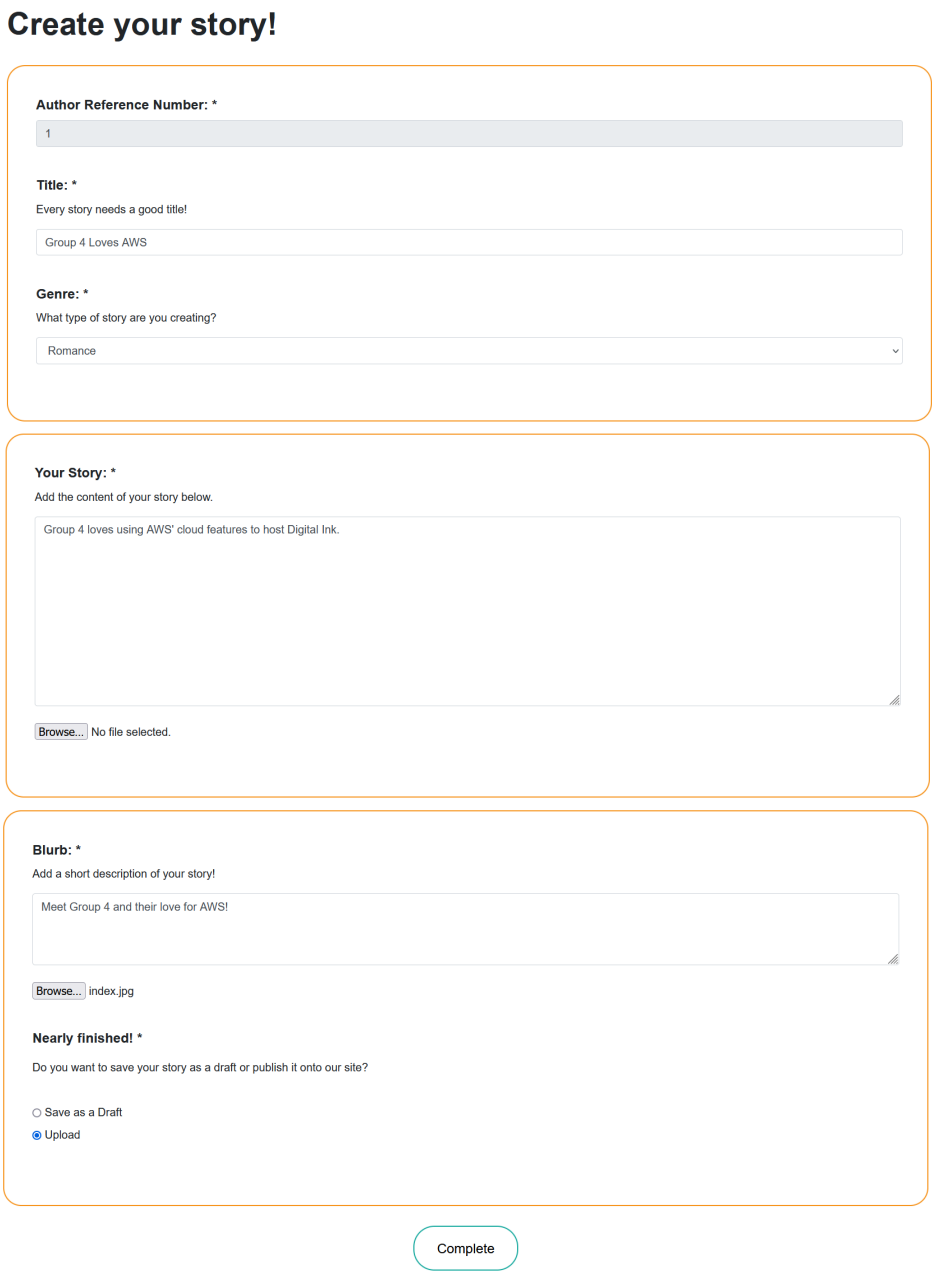

Figure 3.6: *Digital-Ink* story creation form.

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.

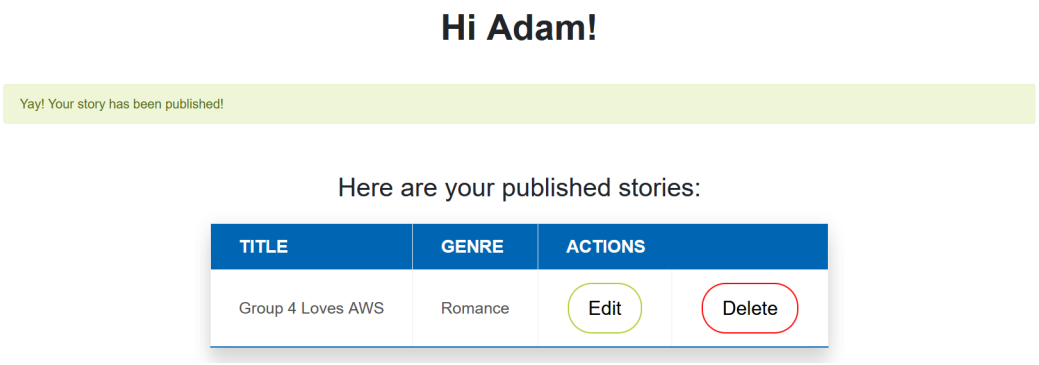

Figure 3.7: *Digital-Ink* account page.

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.

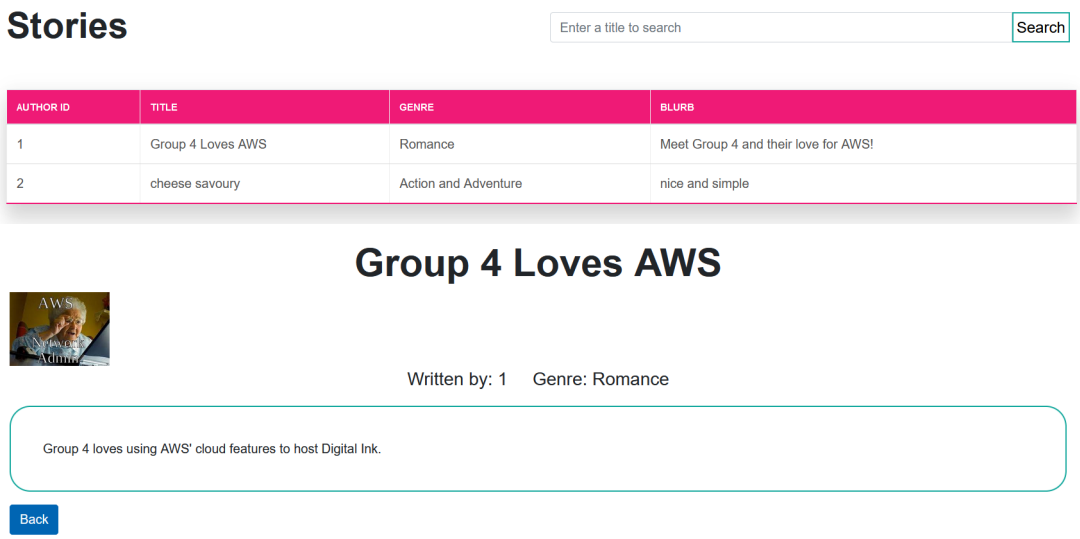

Figure 3.8: *Digital-Ink* stories page and story view.

Chapter 4

Virtual Private Cloud and Subnets

Amazon Virtual Private Cloud (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), 2022b). A VPC can have multiple assigned subnets, which are a range of IP addresses accessible in the VPC.

Chapter 5

Elastic Cloud Compute

5.1 AWS Setup

After the configuration of the VPC and subnets was completed, the initial deployment of the web app began through 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), 2022a). The web app will be stored on an EC2 instance of Amazon Linux, known as Amazon machine images (AMIs), which wil then be launched through a docker container stored on the app.

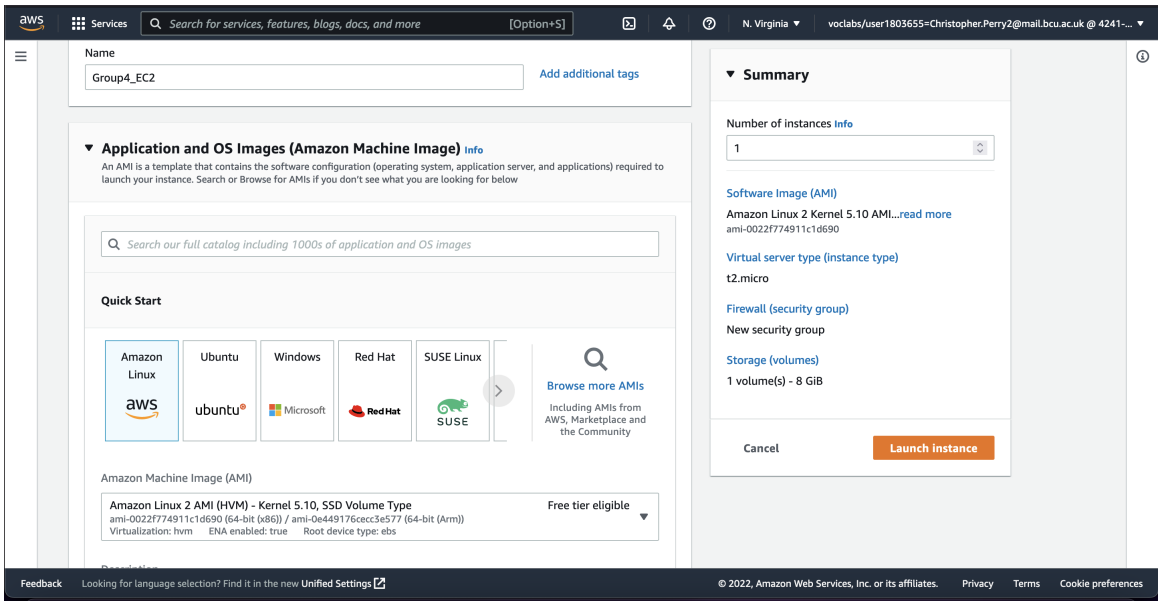

Figure 5.1: Selection of EC2 OS Image.

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). The selection of this can be found in Figure 5.2.

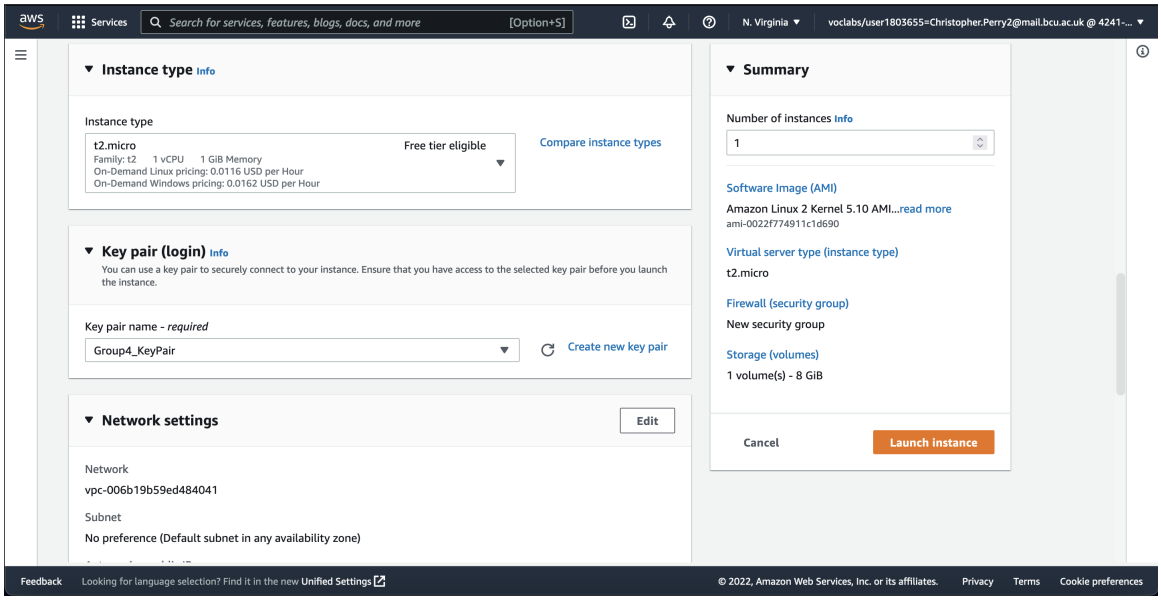

Figure 5.2: Selection of EC2 Instance.

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 DockerHub, which will allow a Laravel instance to be initialised, as discussed in Section 3.

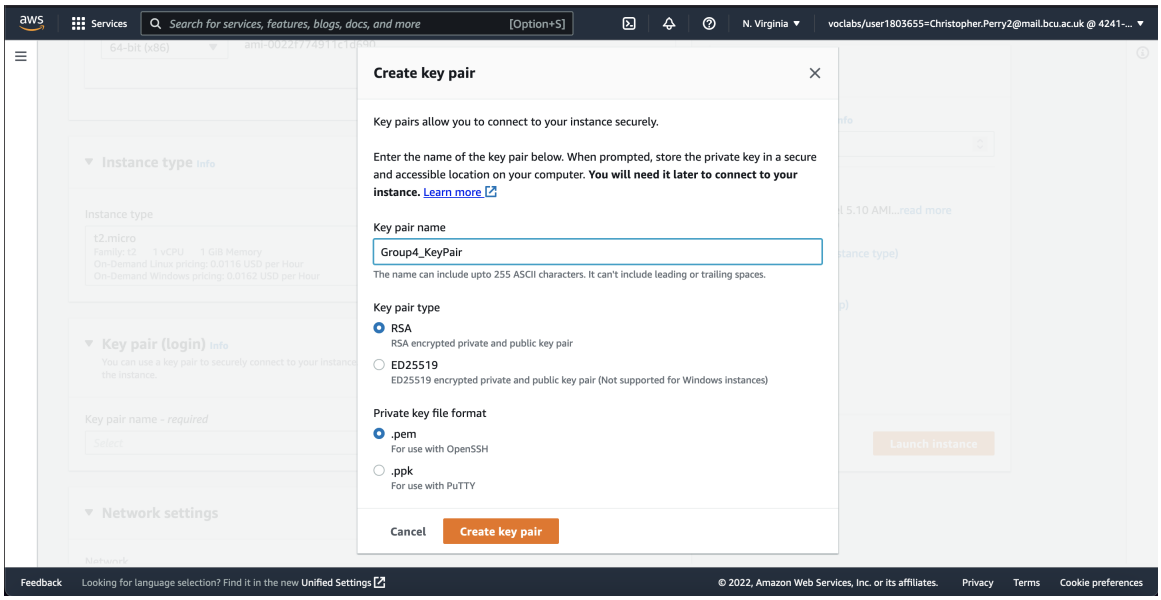

Figure 5.3: Selection of EC2 Keypair.

This process can be seen in Figure 5.3.

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

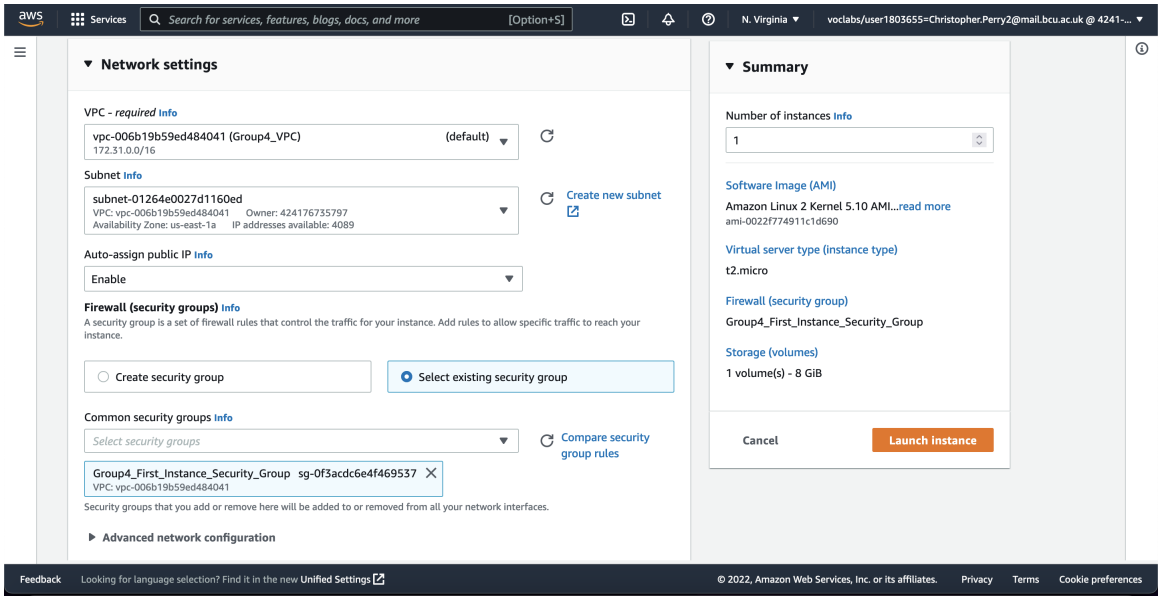

Figure 5.4: Selection of EC2 Networking options.

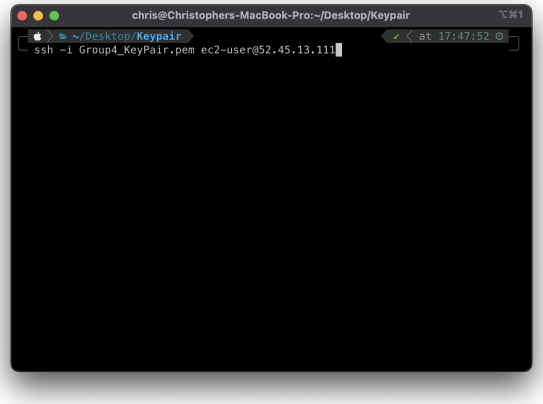

Figure 5.5: Generated EC2 Keypair in the `.pem` format.

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.

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.

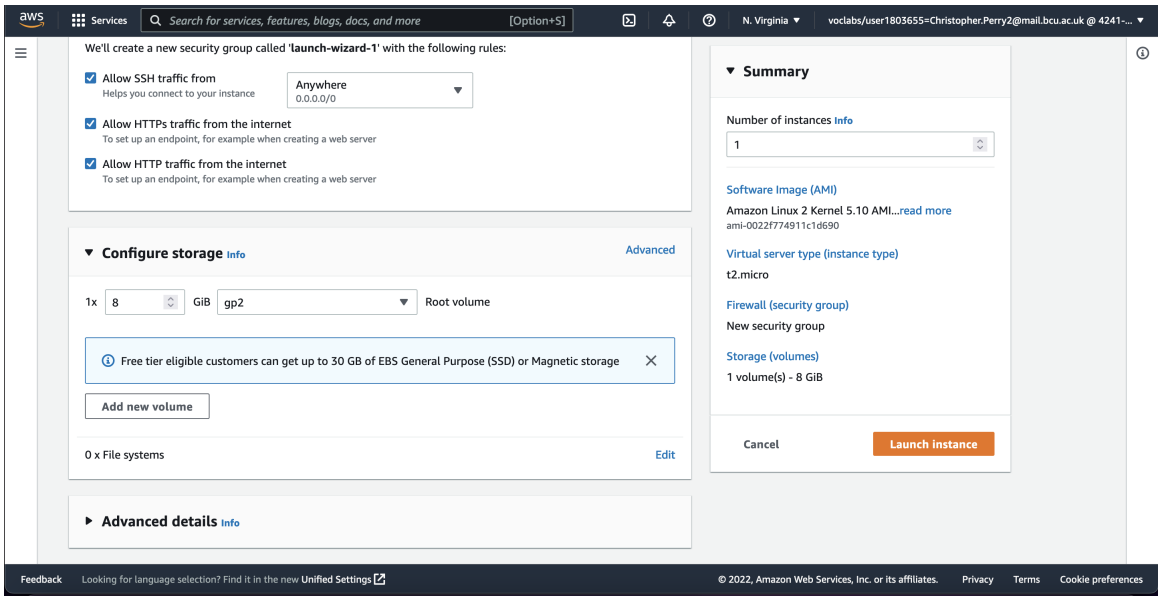

Figure 5.6: Selection of EC2 Storage Configuration.

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.

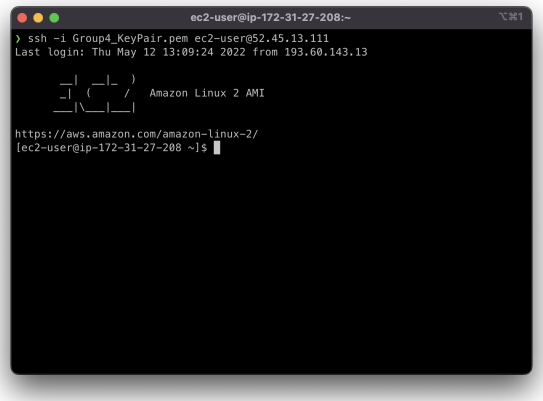

Figure 5.8: Selection of EC2 Storage Configuration.

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`.

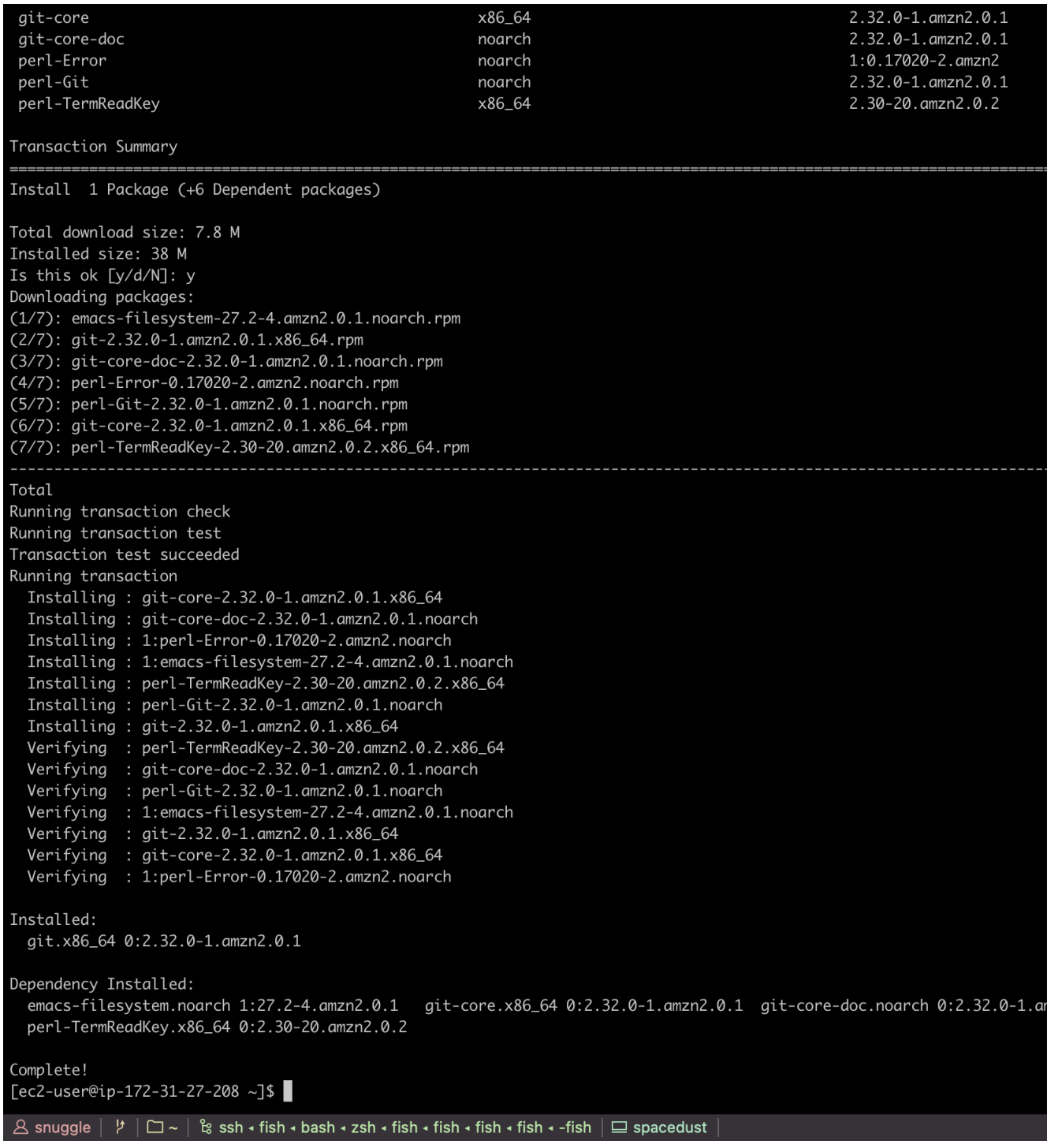

Figure 5.9: Installing Git.

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

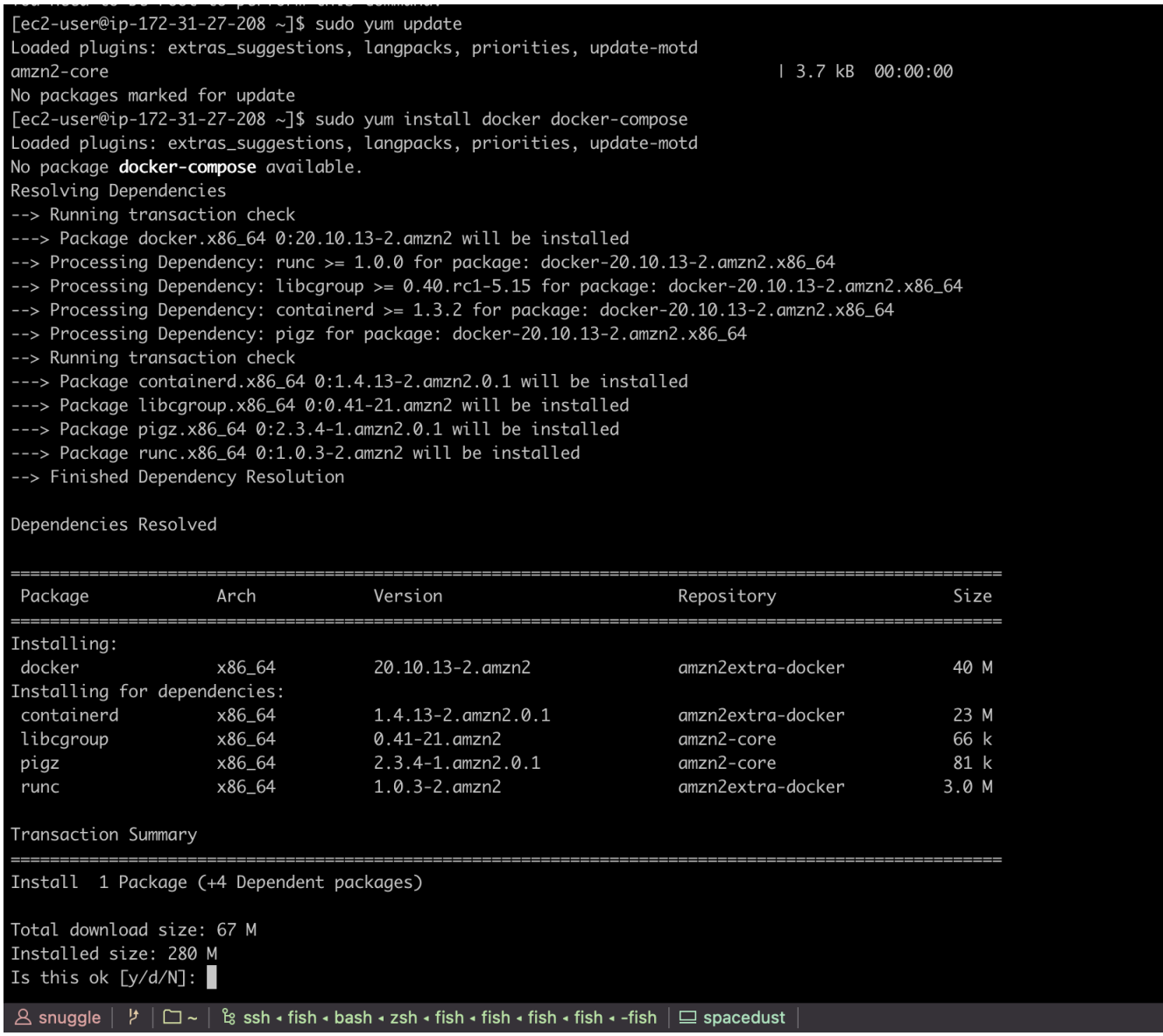

Figure 5.10: Installing Docker.

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.

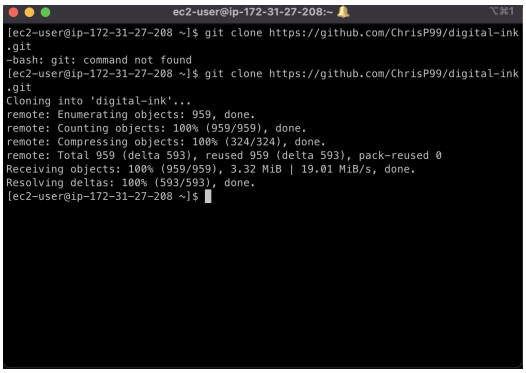

Figure 5.11: Cloning the web app from Github.

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.

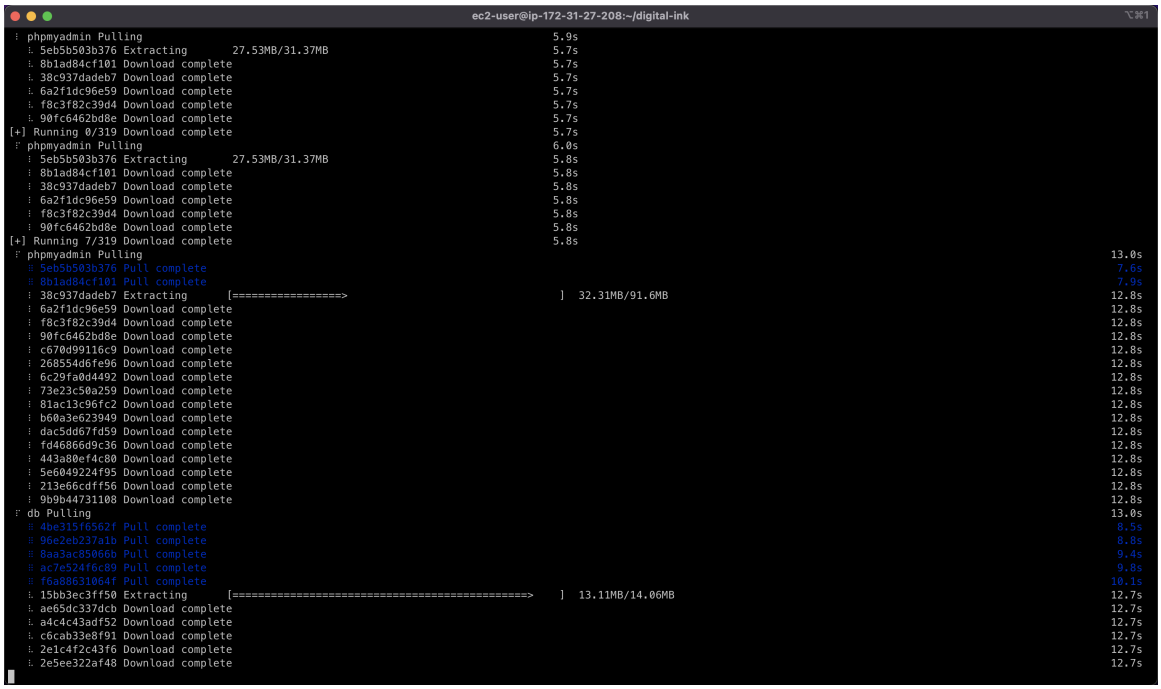

Figure 5.12: Containers required for the web app being pulled from Docker Hub.

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 . 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

5.5 systemd Services

systemd services was used to ensure the automatic starting of the digital-ink web application

EVIE FILL THIS OUT YOU SILLY LLAMA

Chapter 6

Simple Storage Service

Chapter 7

CloudFront

Chapter 8

CloudWatch

Chapter 9

CloudTrail

Chapter 10

Relational Database Service

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

Chapter 13

Security Practices

Chapter 14

Cost Breakdown

14.1 Estimated Costs

14.2 Scaling Up to 10,000 Users

14.3 Scaling Up to 1 Million Users

14.4 Scaling Up to 10 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 (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 15.1.

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

Figure 15.1: Gherkin example.

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 \mintinline{sql}|.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

```
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 ...
```

15.4 Testing RDS

```
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

Chapter 17

Conclusion

References

Amazon Web Services (AWS) (2022a). *What is Amazon EC2?* AVAILABLE AT: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

– (2022b). *What is Amazon VPC?* AVAILABLE AT: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.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.

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/isbn/9780201770612.

Lerdorf, R. et al. (2002). *Programming Php*. " O’Reilly Media, Inc.". AVAILABLE AT: https://isbnsearch.org/isbn/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.

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

Appendix A: Screenshots

I am an appendix, please be kind.

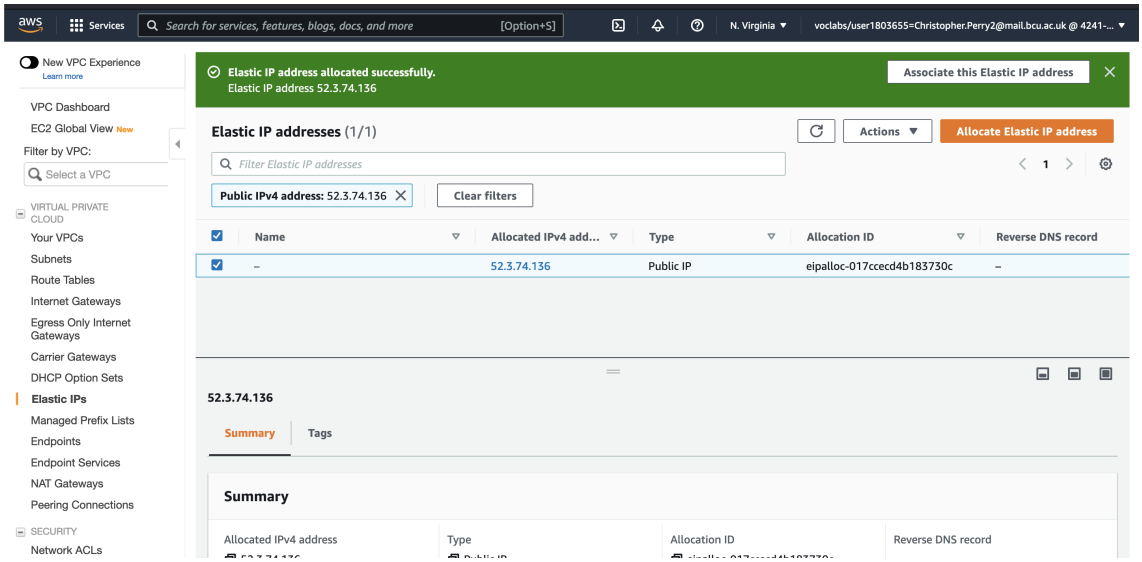

Figure A.1: After Allocating Elastic IP Address

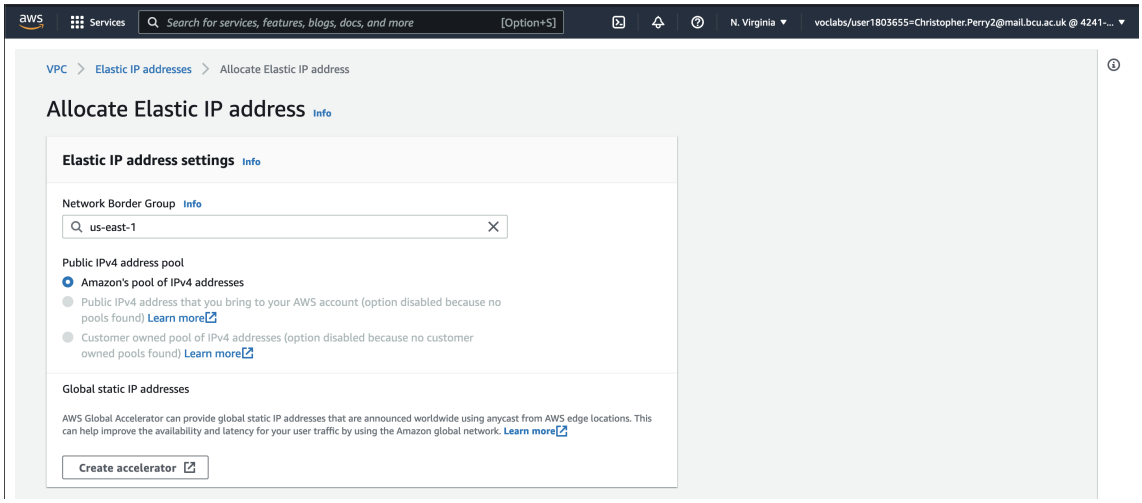

Figure A.2: Allocating Elastic IP Address

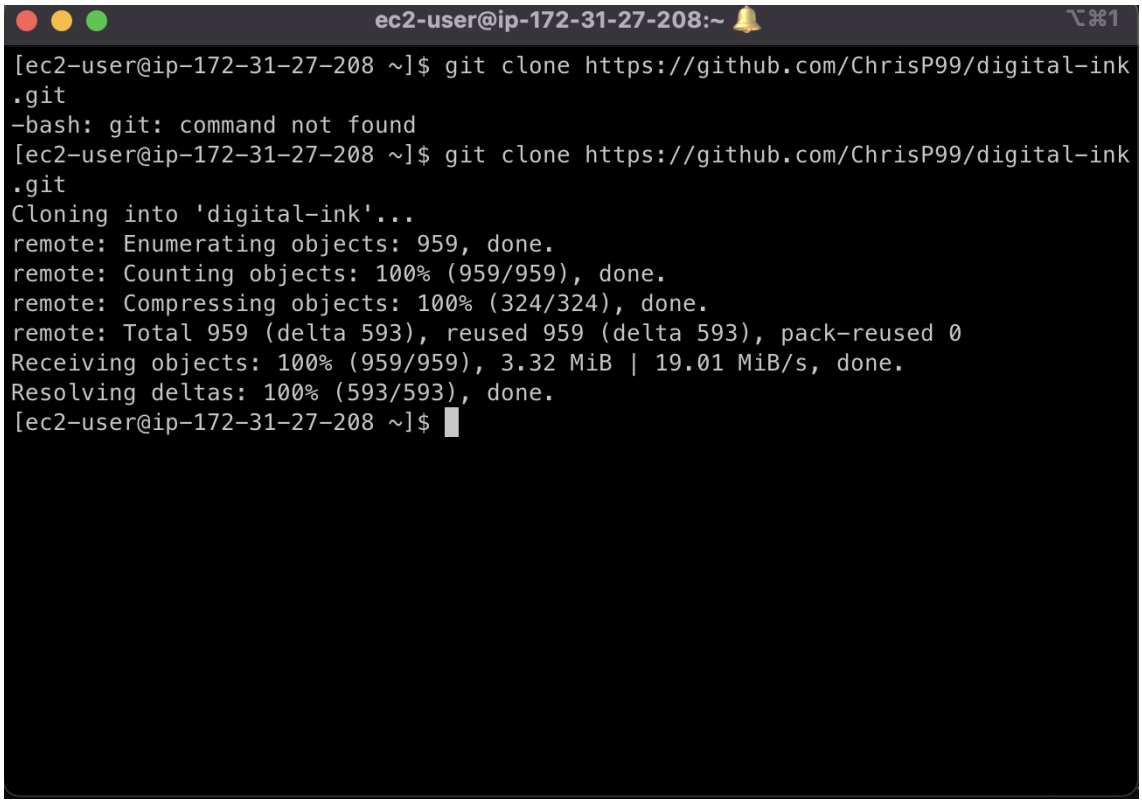

Figure A.3: Cloning the App

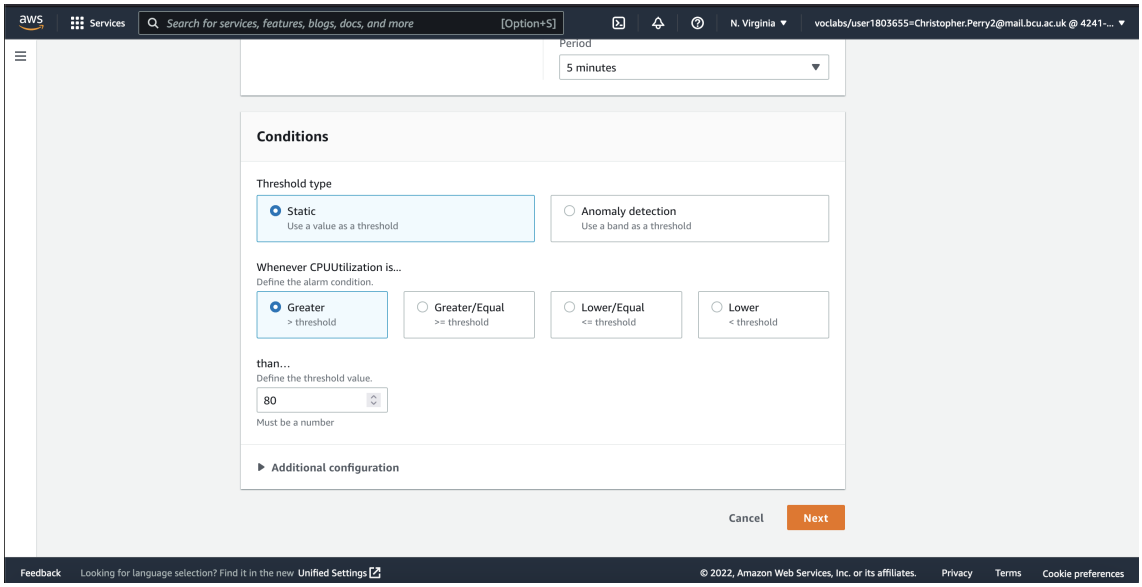

Figure A.4: CloudWatch Conditions

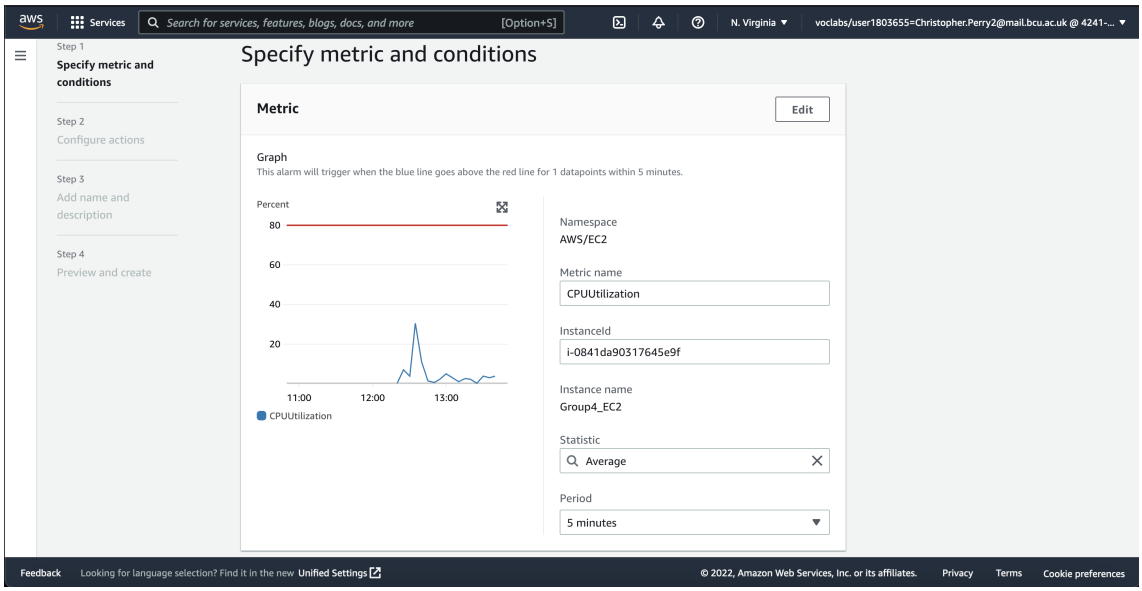

Figure A.5: CloudWatch Specify Metric

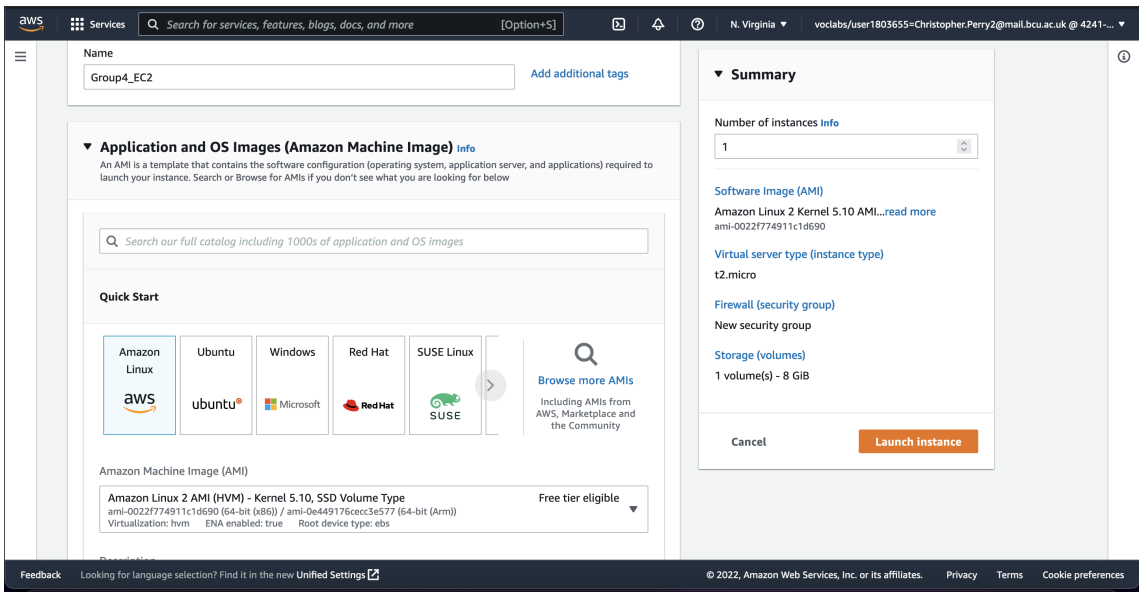

Figure A.6: Create Instance - Application and OS Images

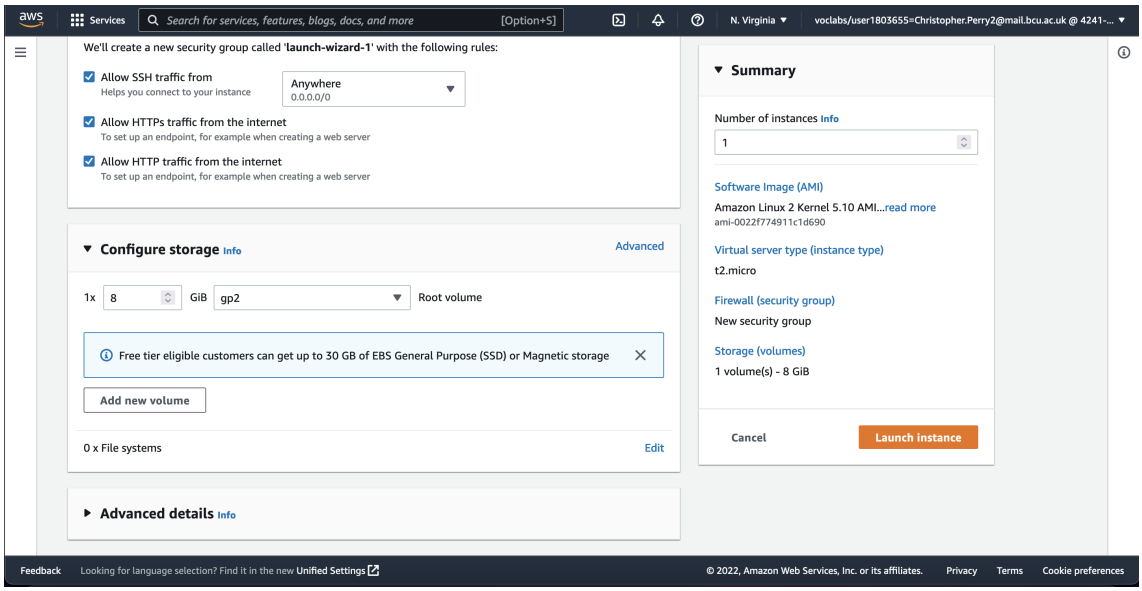

Figure A.7: Create Instance - Configure Storage

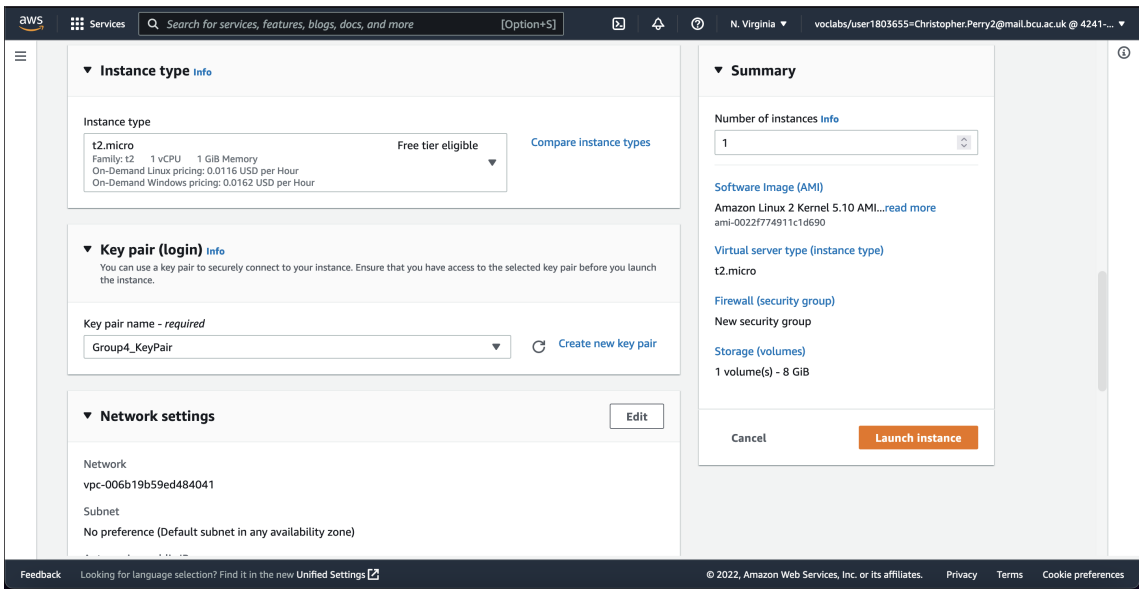

Figure A.8: Create Instance - Instance Type

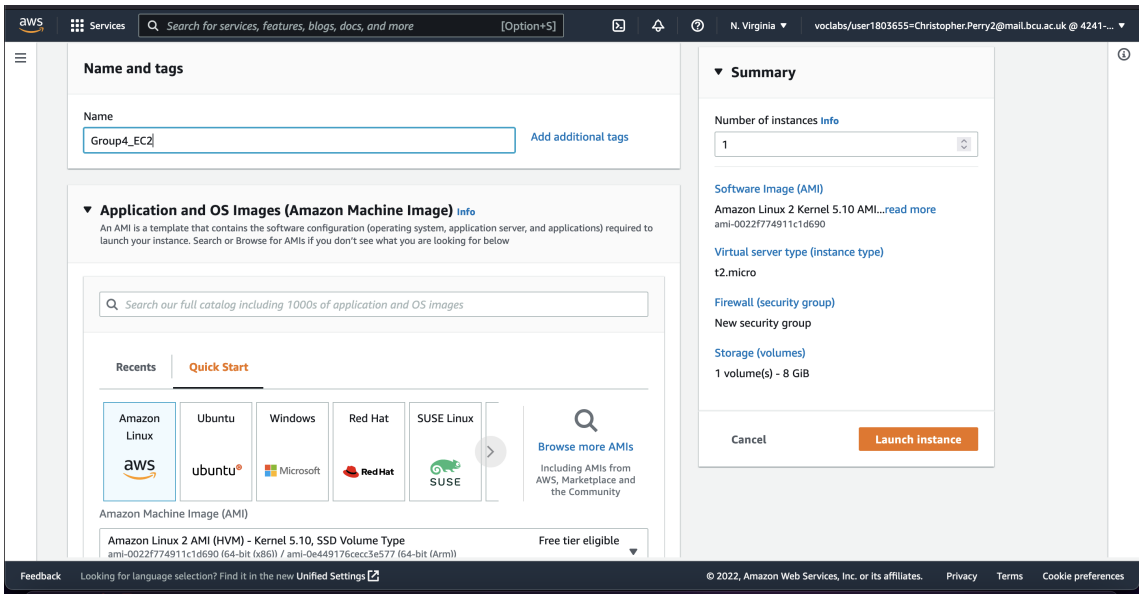

Figure A.9: Create Instance - Name & Tags

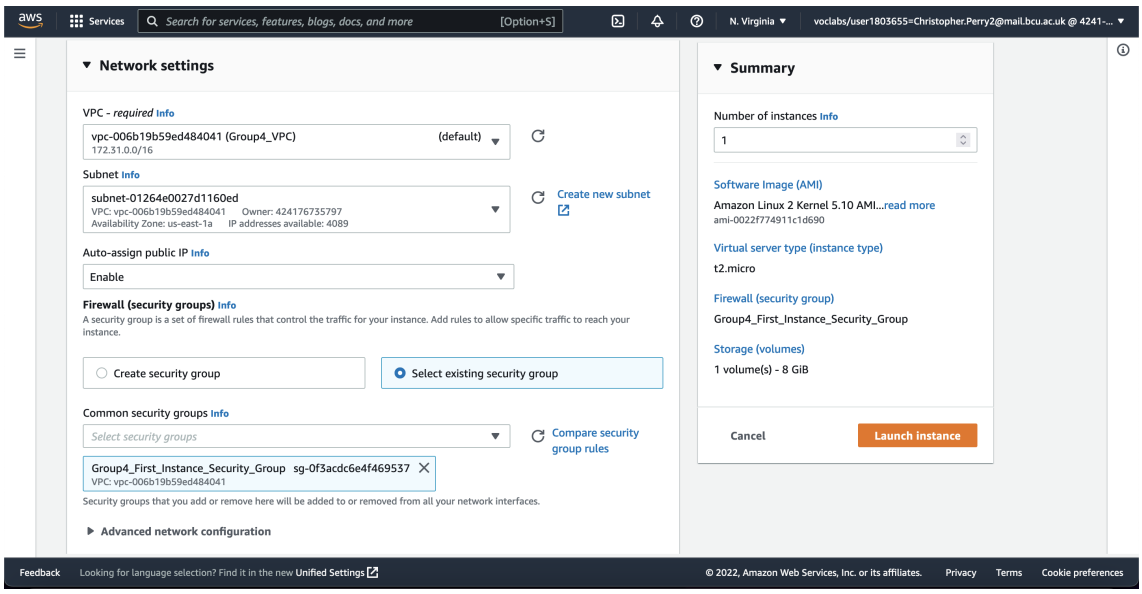

Figure A.10: Create Instance - Network Settings

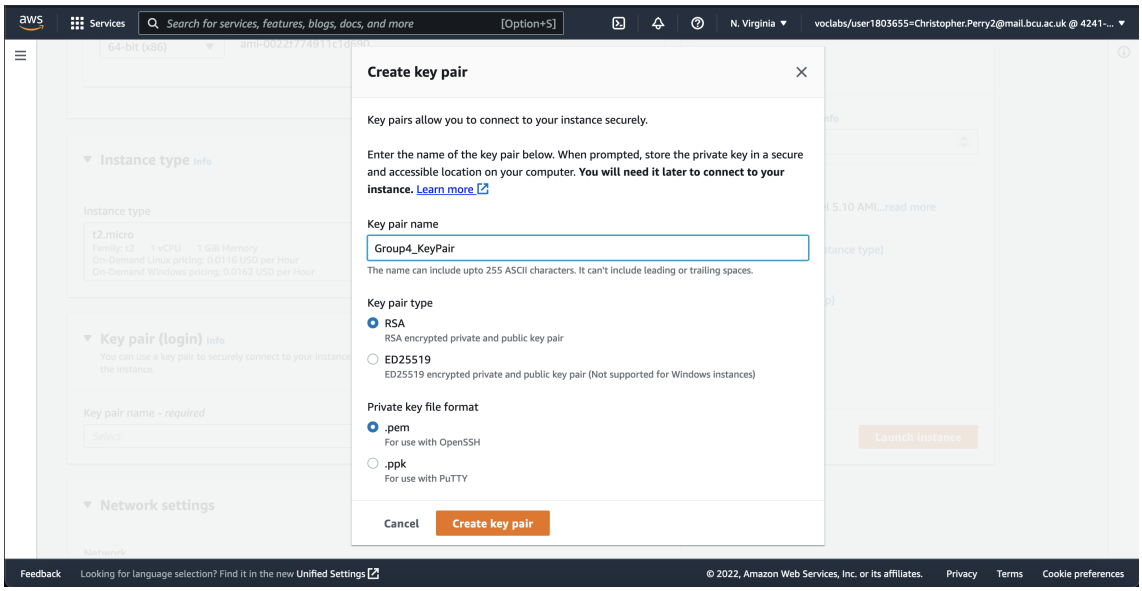

Figure A.11: Creating Key Pair

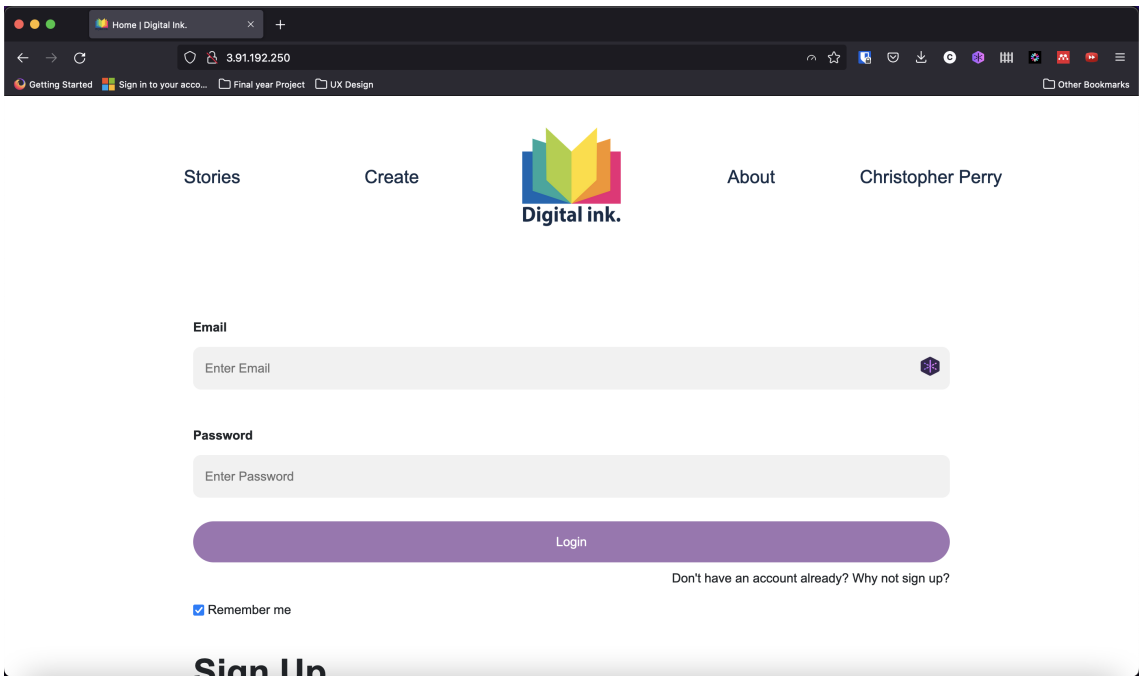

Figure A.12: Digital Ink

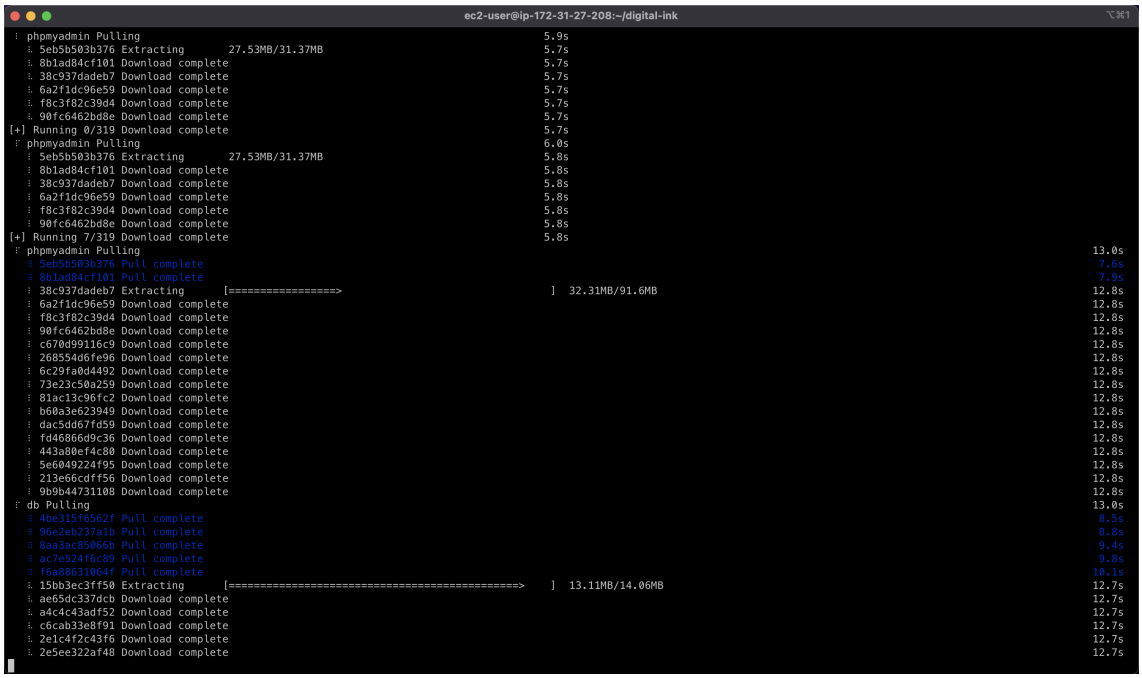

Figure A.13: Docker Compose

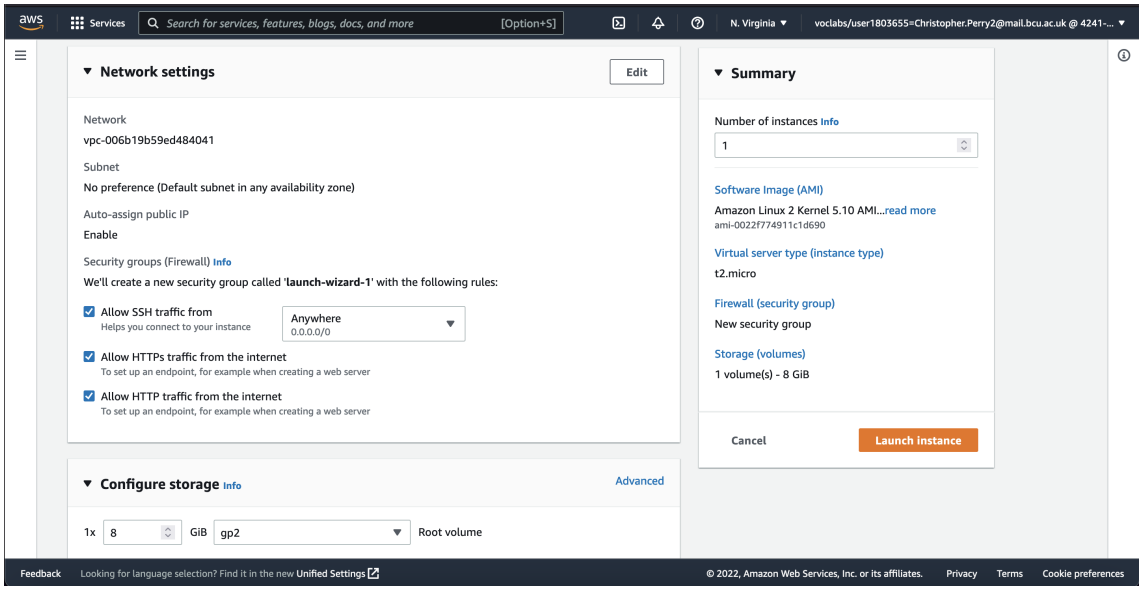

Figure A.14: Edit Instance - Network Settings

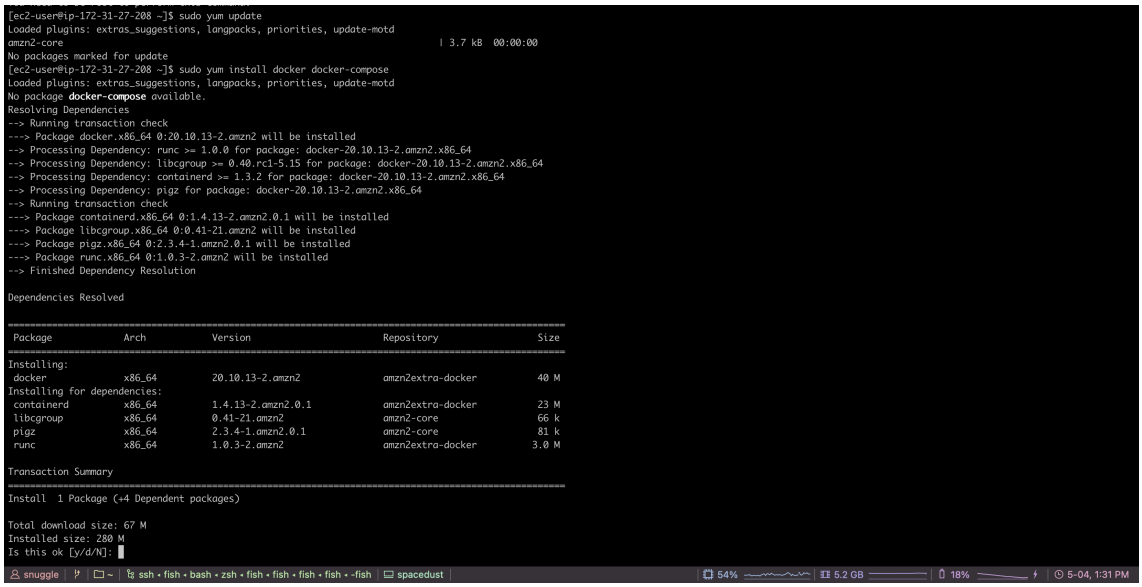

Figure A.15: Installing Docker using Package Manager

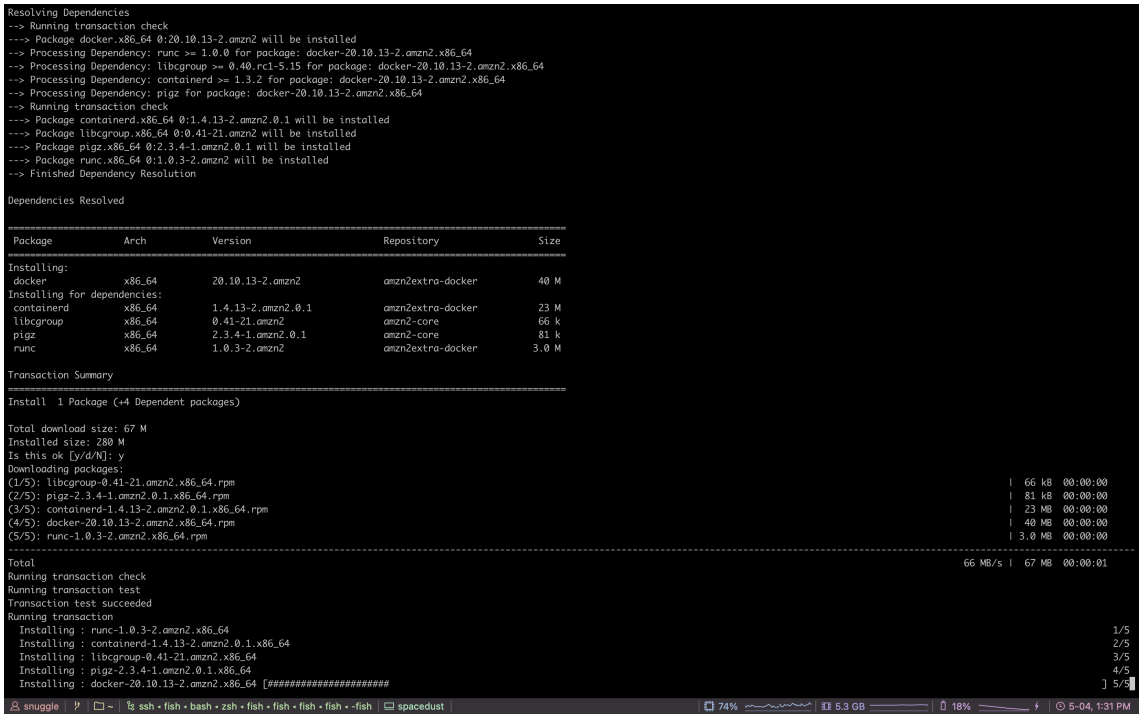

Figure A.16: Installing Docker using Package Manager (In Progress)

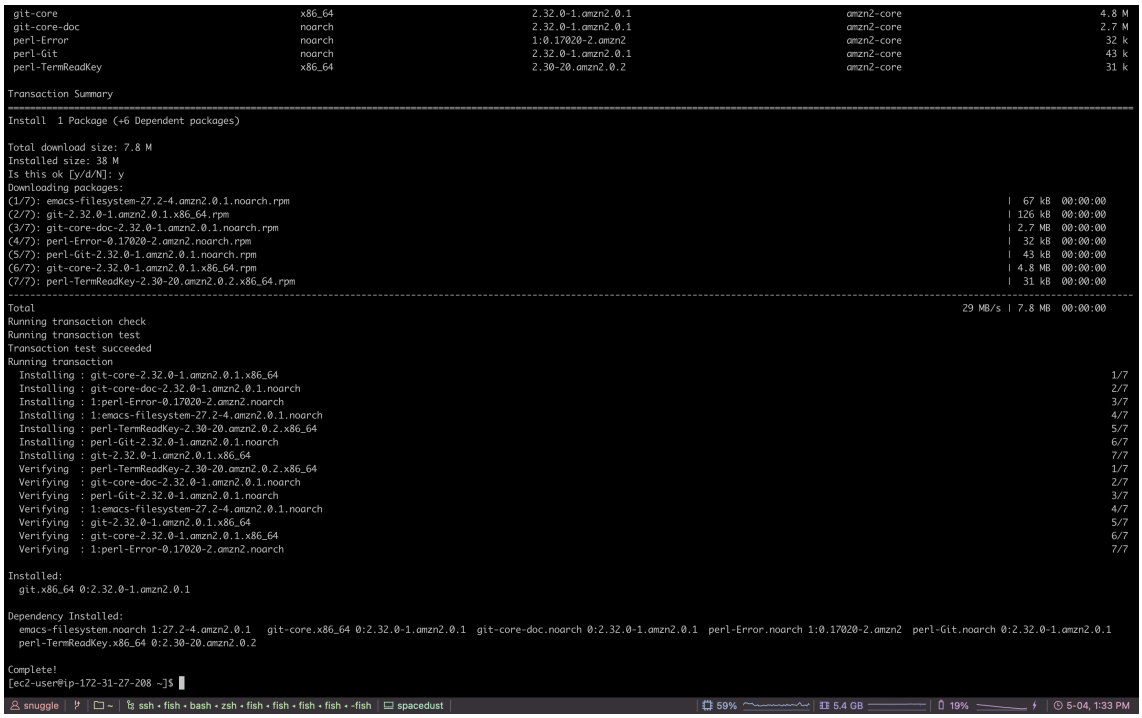

Figure A.17: Installing Git

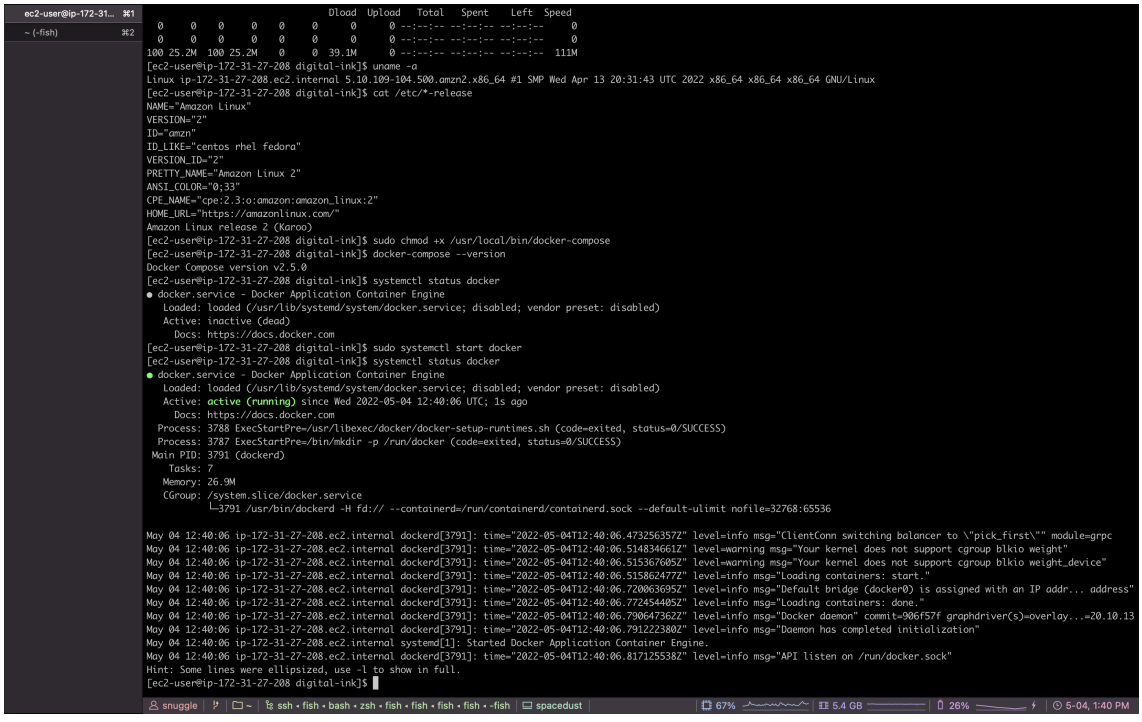

Figure A.18: Starting Docker systemd Service

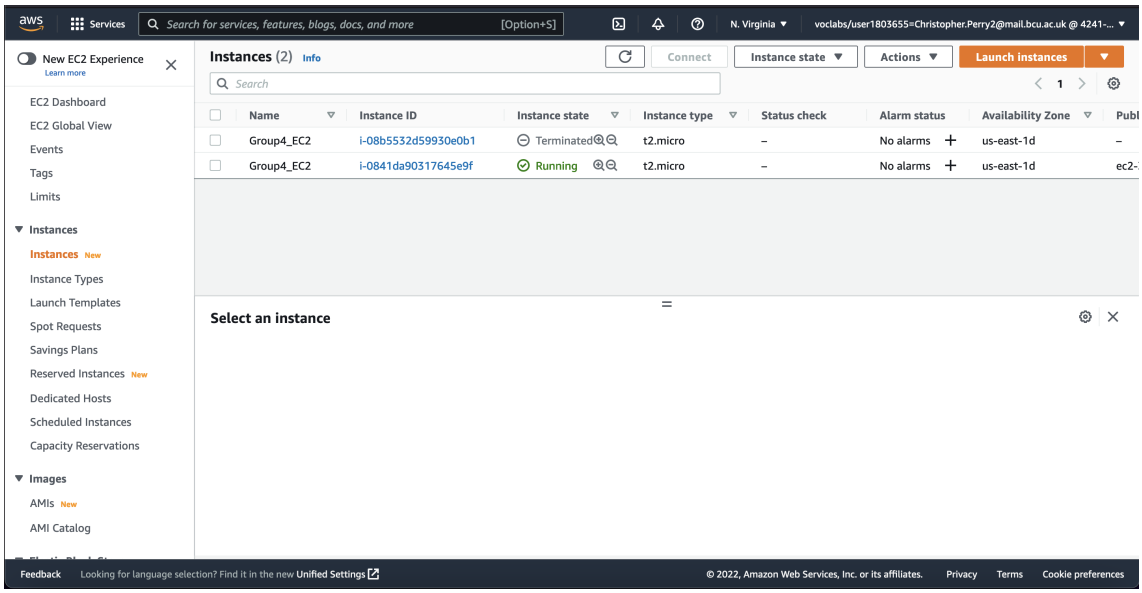

Figure A.19: Instances

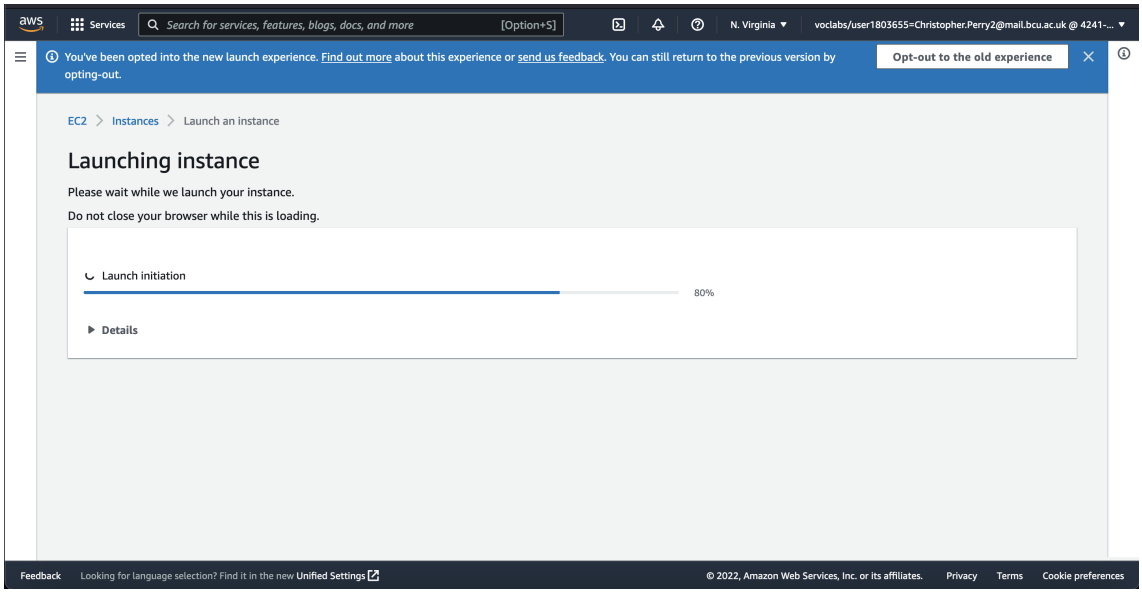

Figure A.20: Launching Instance

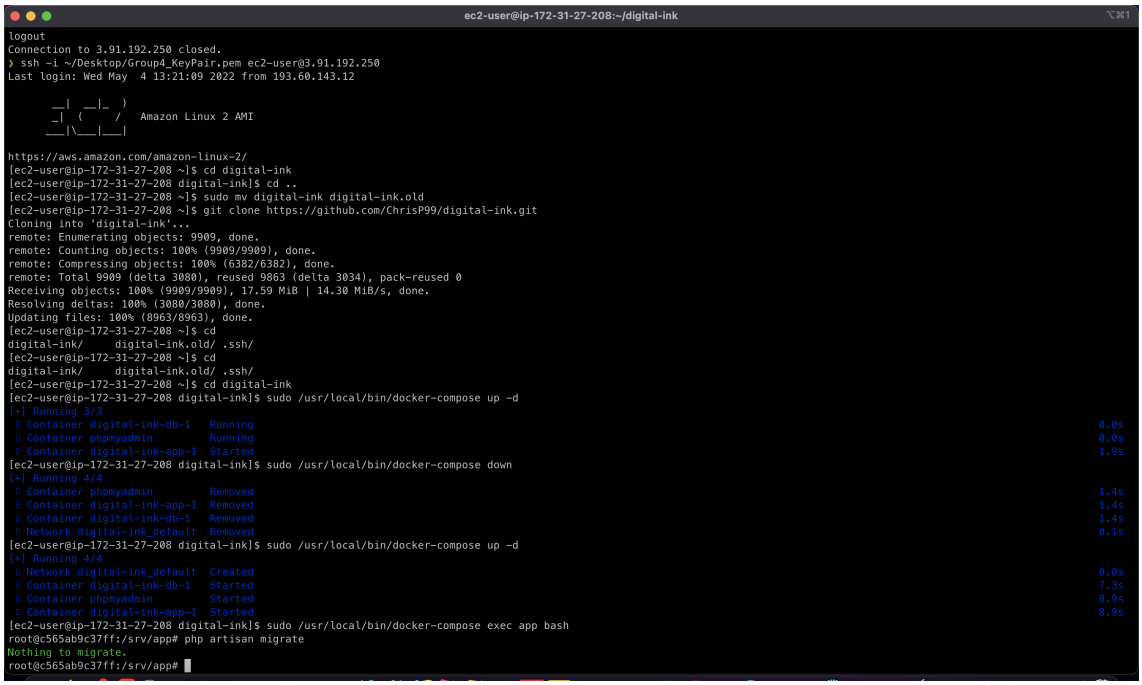

Figure A.21: Log In with Key Pair

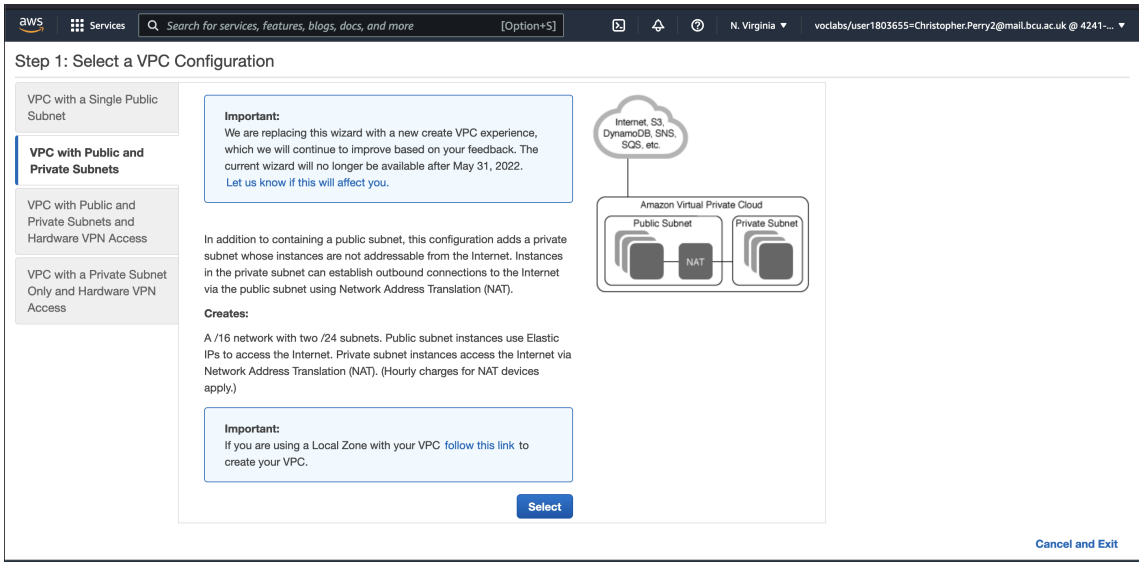

Figure A.22: Selecting a VPC Configuration

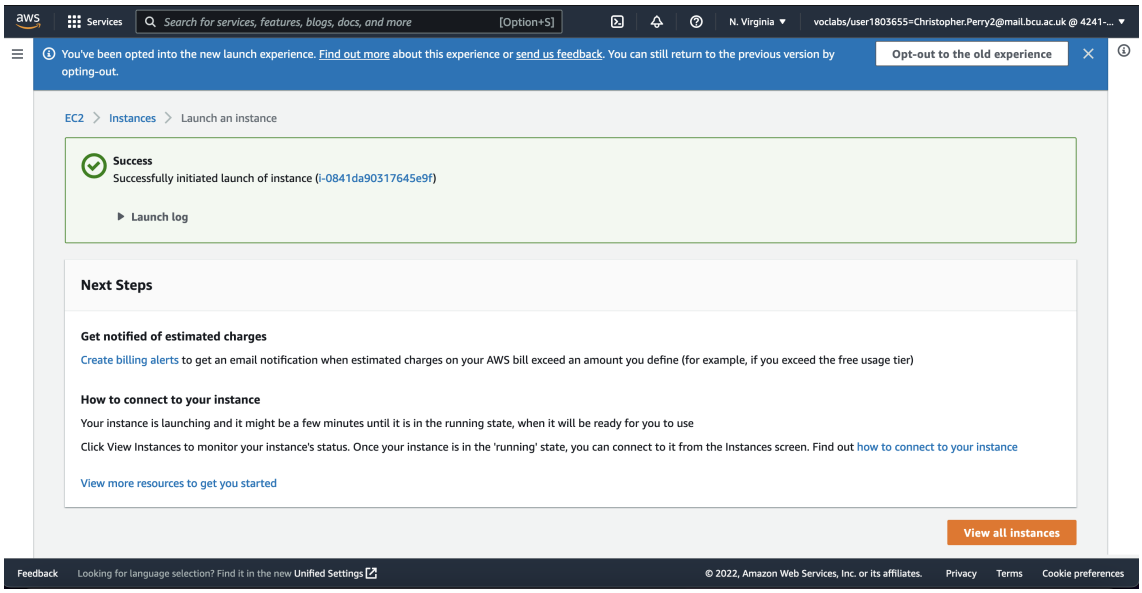

Figure A.23: Successfully Initiated Instance

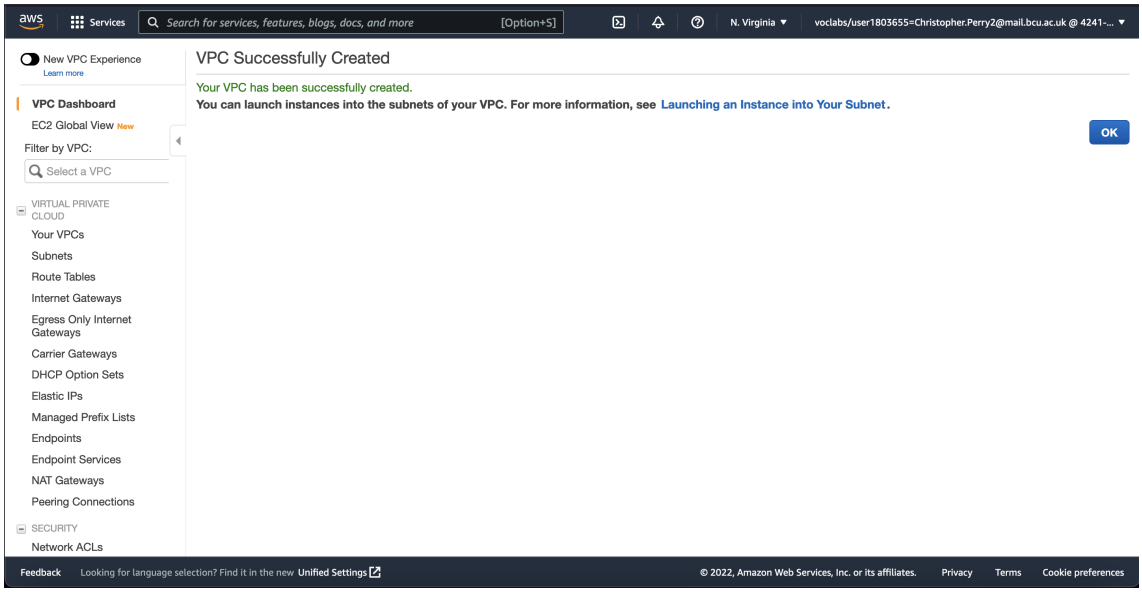

Figure A.24: VPC Successfully Created

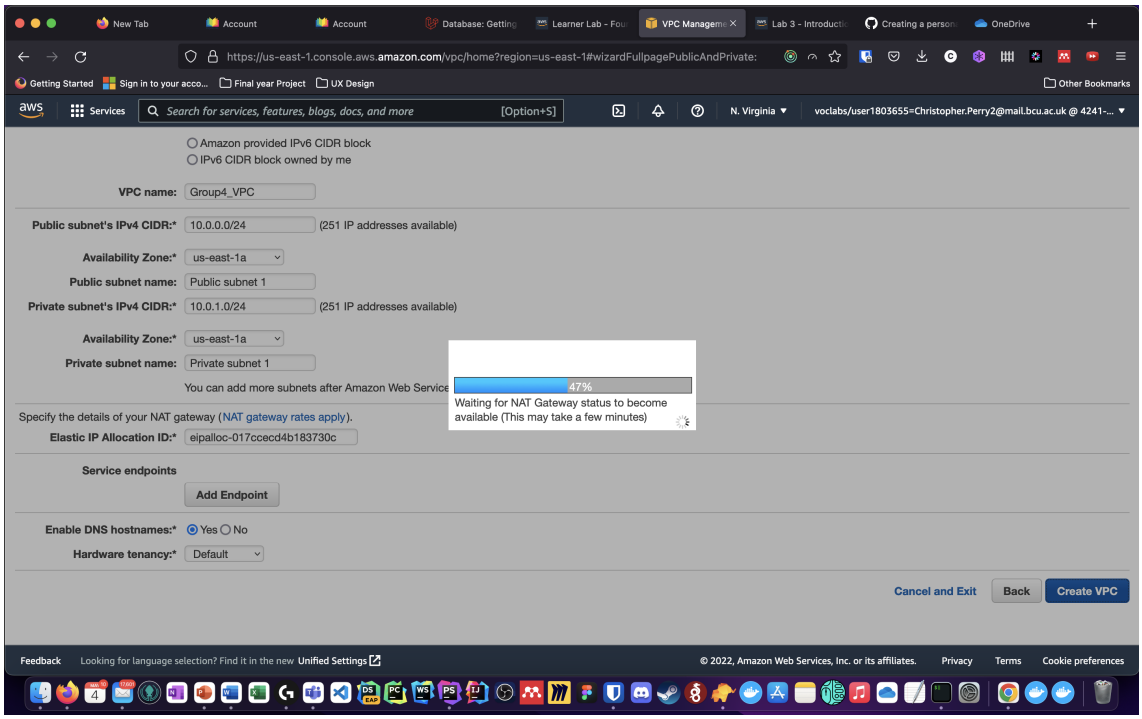

Figure A.25: VPC with Public and Private Subnets, Loading

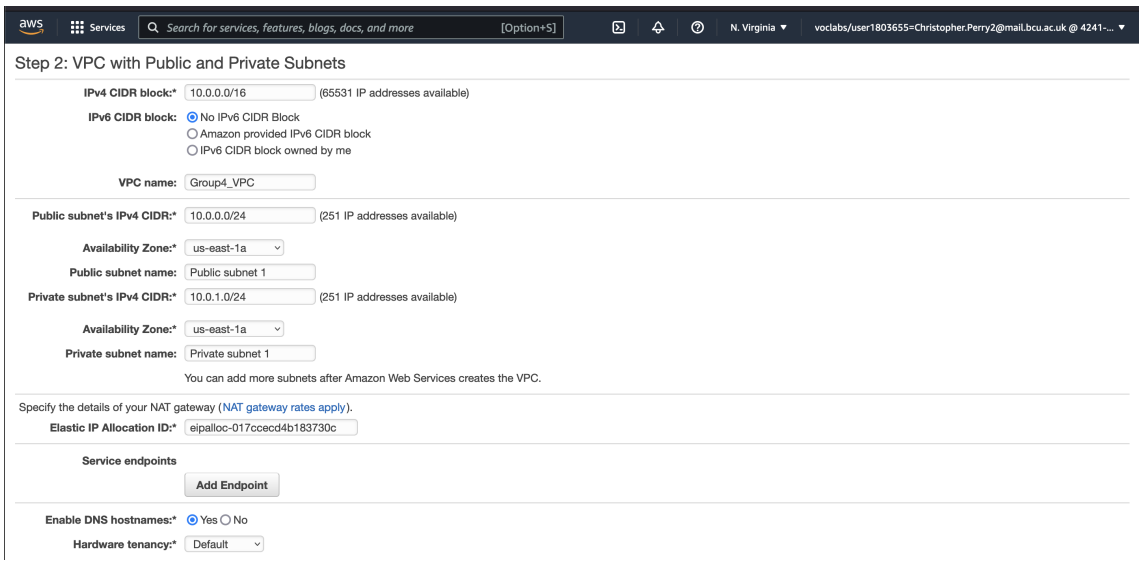

Figure A.26: VPC with Public and Private Subnets

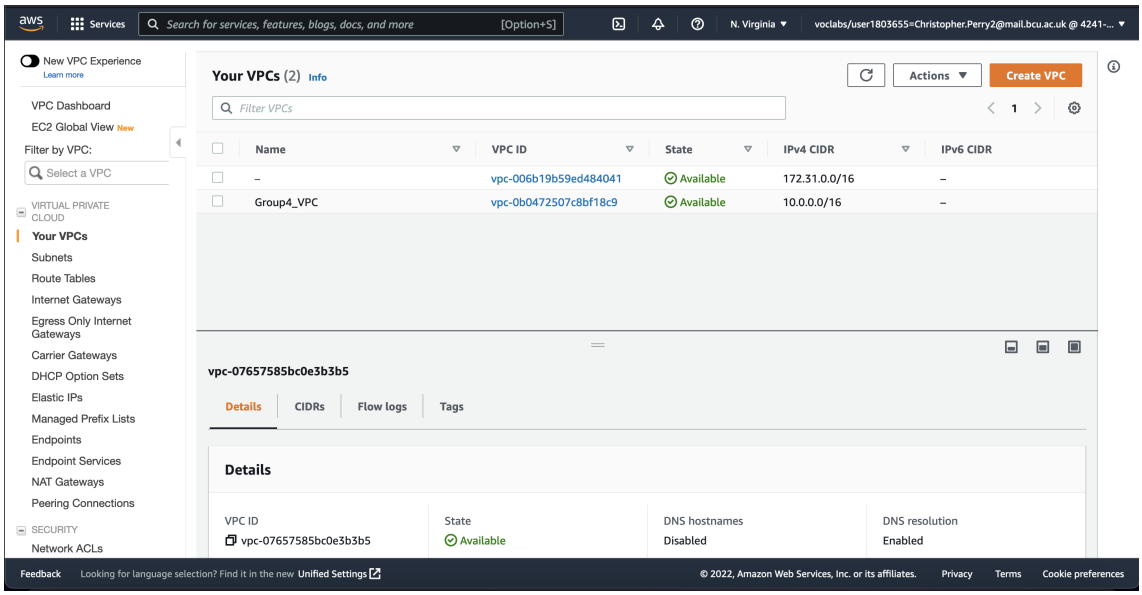

Figure A.27: Your VPCs