**Final Project Report**

**Deploying a Highly Available WordPress Application on AWS**

**Abstract**

This document forms part of the assessment for SIT233 Cloud Computing. The project involves deploying a highly available WordPress application using AWS services, emphasizing scalability, fault tolerance, and automation through Infrastructure as Code (IaC). The solution utilizes a custom VPC, RDS with Multi-AZ and read-replica support, EC2 instances managed via Auto Scaling Groups, an Application Load Balancer, and S3 integration for media storage. The deployment process is demonstrated with detailed step-by-step screenshots and validated through functional testing.

# Introduction

The objective of this project is to design and implement a scalable and highly available WordPress application architecture on AWS. This initiative allows exploration of AWS foundational services such as EC2, RDS, VPC, S3, and Auto Scaling, while ensuring high availability and resilience through proper network design, load balancing, and failover mechanisms. The project also emphasizes automation using AWS CloudFormation for consistent and repeatable deployments.

# Design Diagram

The overall architecture for the WordPress deployment on AWS is structured to follow a three-tier web application model. The infrastructure was designed for high availability, performance, and fault tolerance using several AWS services.

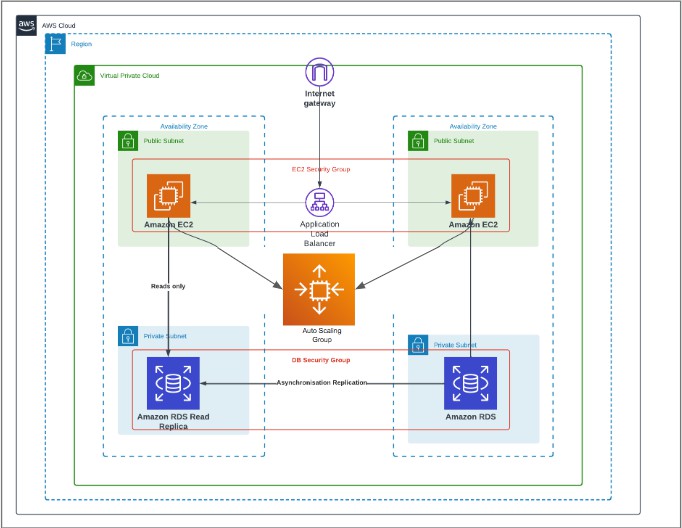
At the network layer, a custom Virtual Private Cloud (VPC) is used to logically isolate the deployment. This VPC includes both public and private subnets across multiple availability zones to provide fault tolerance and load distribution. Public subnets host the Application Load Balancer (ALB) and the NAT Gateway, which allows outbound internet access for instances in the private subnets. Private subnets are used to deploy the EC2 instances and RDS database for enhanced security.

At the compute layer, EC2 instances are launched in private subnets from a custom Amazon Machine Image (AMI) containing a pre-configured WordPress installation. These instances are part of an Auto Scaling Group (ASG), which adjusts capacity based on CPU utilization. The ASG ensures that the application remains available under varying load conditions.

The database layer consists of an Amazon RDS instance running MySQL, deployed with Multi-AZ support and a read replica for enhanced reliability and scalability. The database is deployed in private subnets and is not exposed to the public internet.

Other essential services include:

* Application Load Balancer (ALB): Distributes incoming traffic to the EC2 instances.
* Amazon S3: Used for storing WordPress media files via the “Offload Media” plugin.
* IAM Roles and Security Groups: Ensure least-privilege access and controlled communication between resources.
* CloudFormation Templates: Automate the provisioning of infrastructure using two separate templates for networking and application resources.



This architectural layout ensures a well-structured deployment where different components handle specific concerns, leading to better scalability, security, and maintainability.

# Implementation

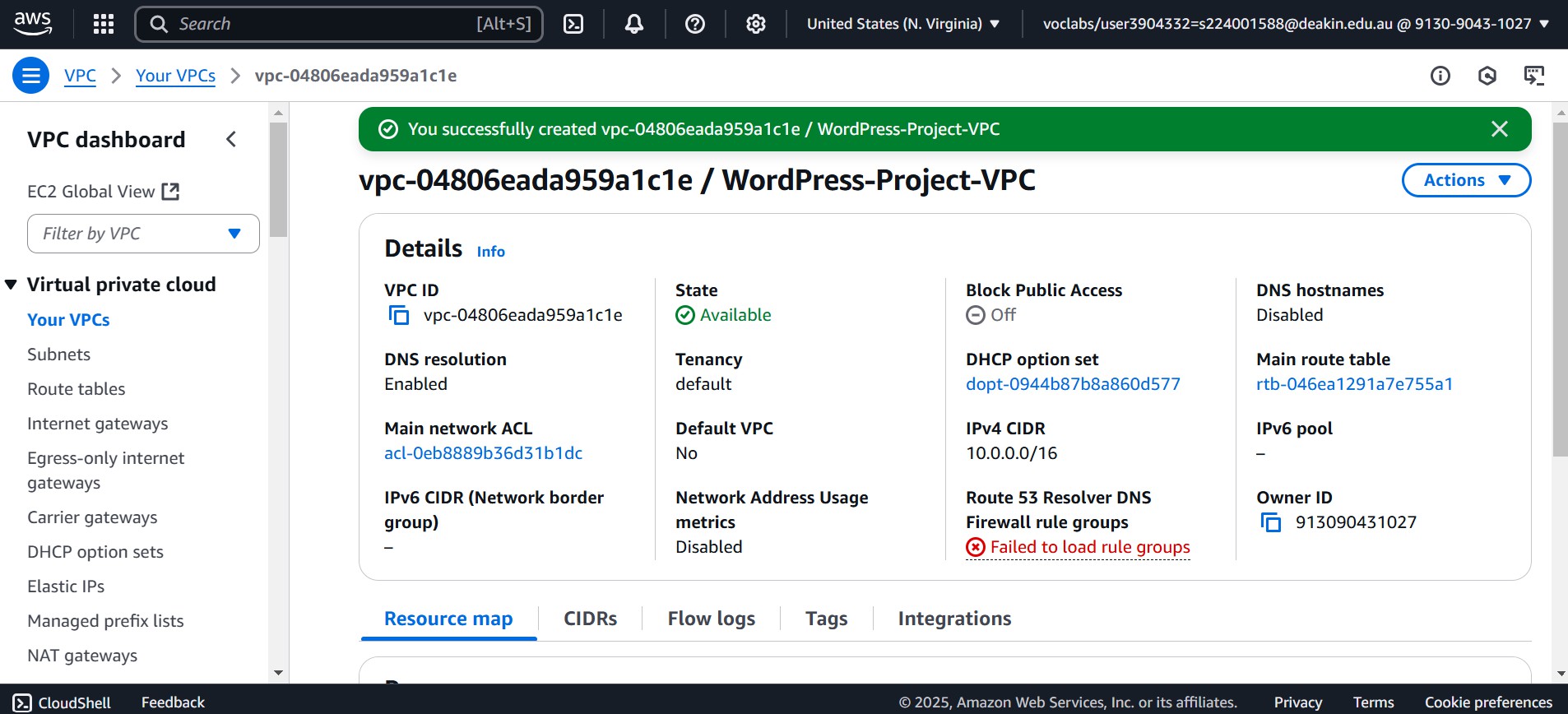
The hands-on implementation of the WordPress deployment was executed entirely using the AWS Management Console, leveraging a range of services and features designed to build a fault-tolerant and scalable architecture. This phase encompassed the setup of isolated networking with a VPC, provisioning of a managed relational database, configuration of web servers, application of load balancing, and automation through CloudFormation templates. Each step was approached methodically, ensuring both functional correctness and alignment with best practices. Screenshots of key stages were taken and embedded for verification, illustrating the successful realization of all technical requirements.

# Step 1: Create a Custom VPC with Subnets

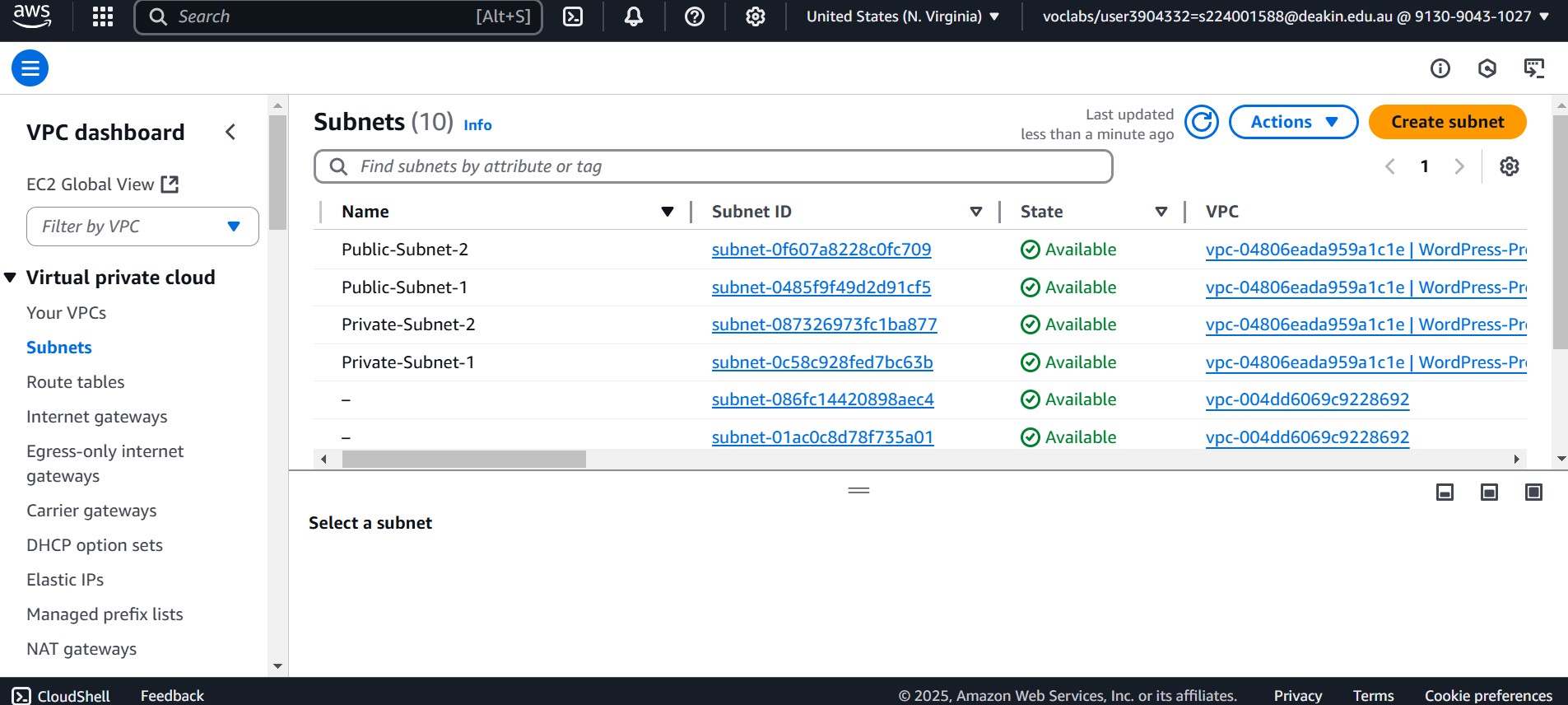
To begin, a Virtual Private Cloud (VPC) was created to logically isolate the cloud resources. Within this VPC, two public and two private subnets were configured. The public subnets host the Application Load Balancer and the NAT Gateway, while the private subnets are used for EC2 instances and RDS to enhance security.

Routing tables were also created:

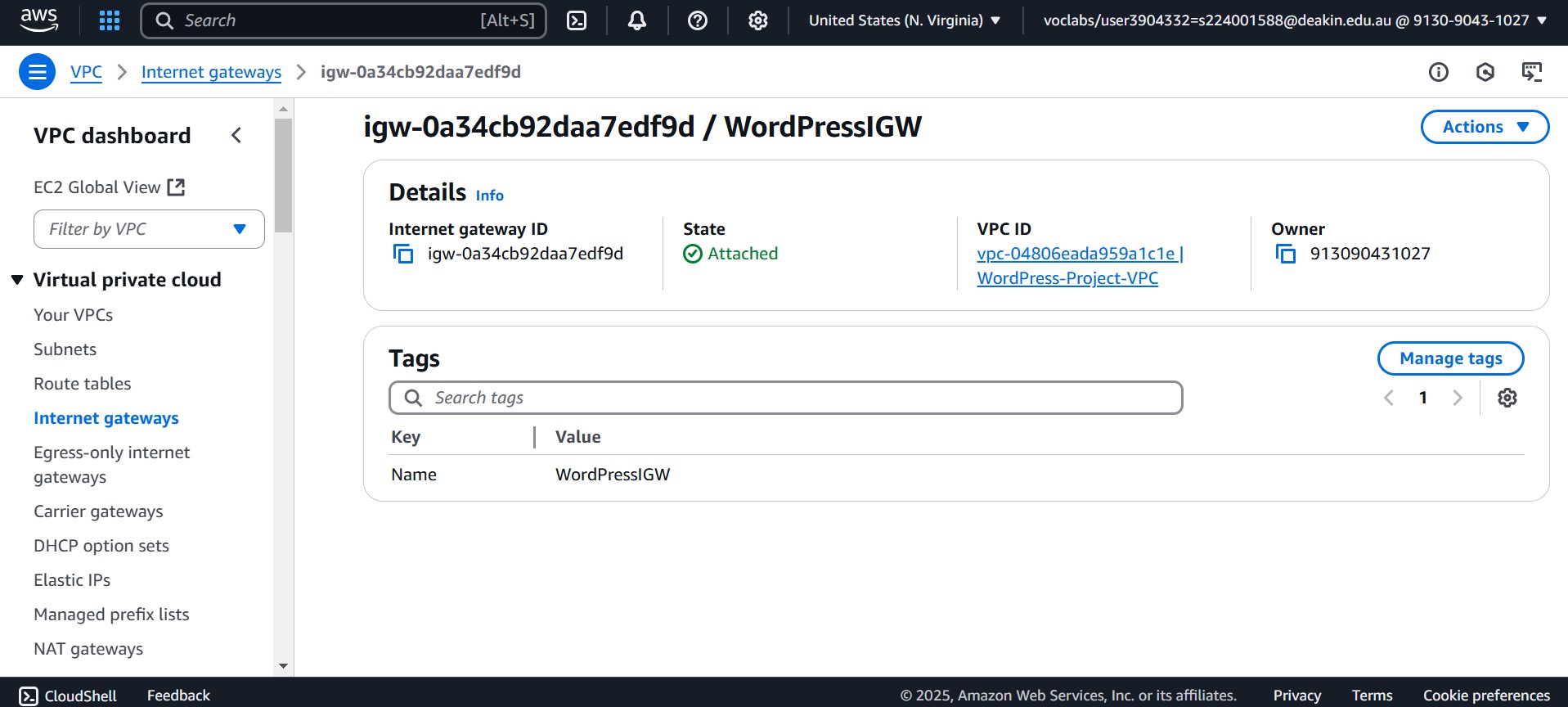
* A public route table was associated with the public subnets and had a route to the Internet Gateway.
* A private route table was associated with private subnets and included a route to the NAT Gateway for outbound internet access.



Public and Private Subnets



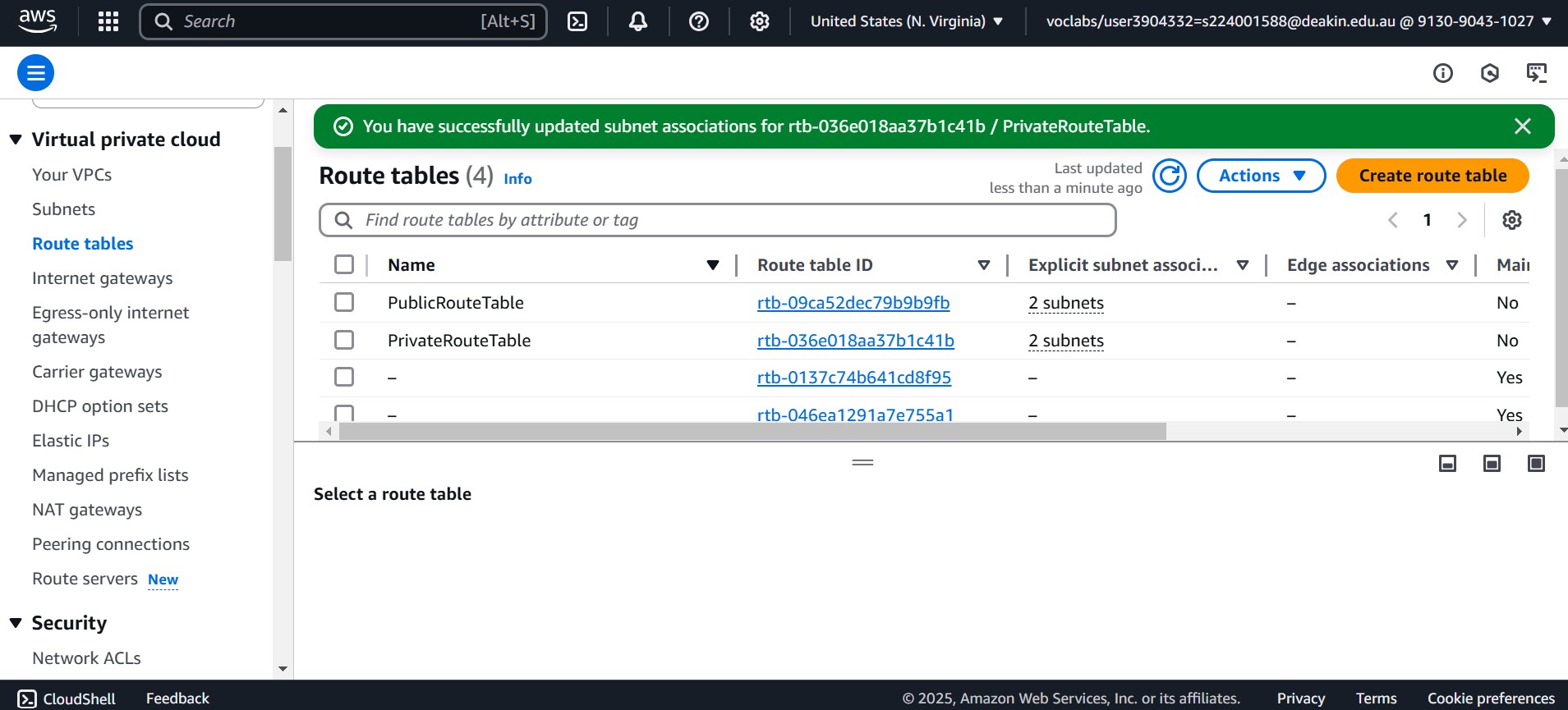
An Internet Gateway



NAT Gateway



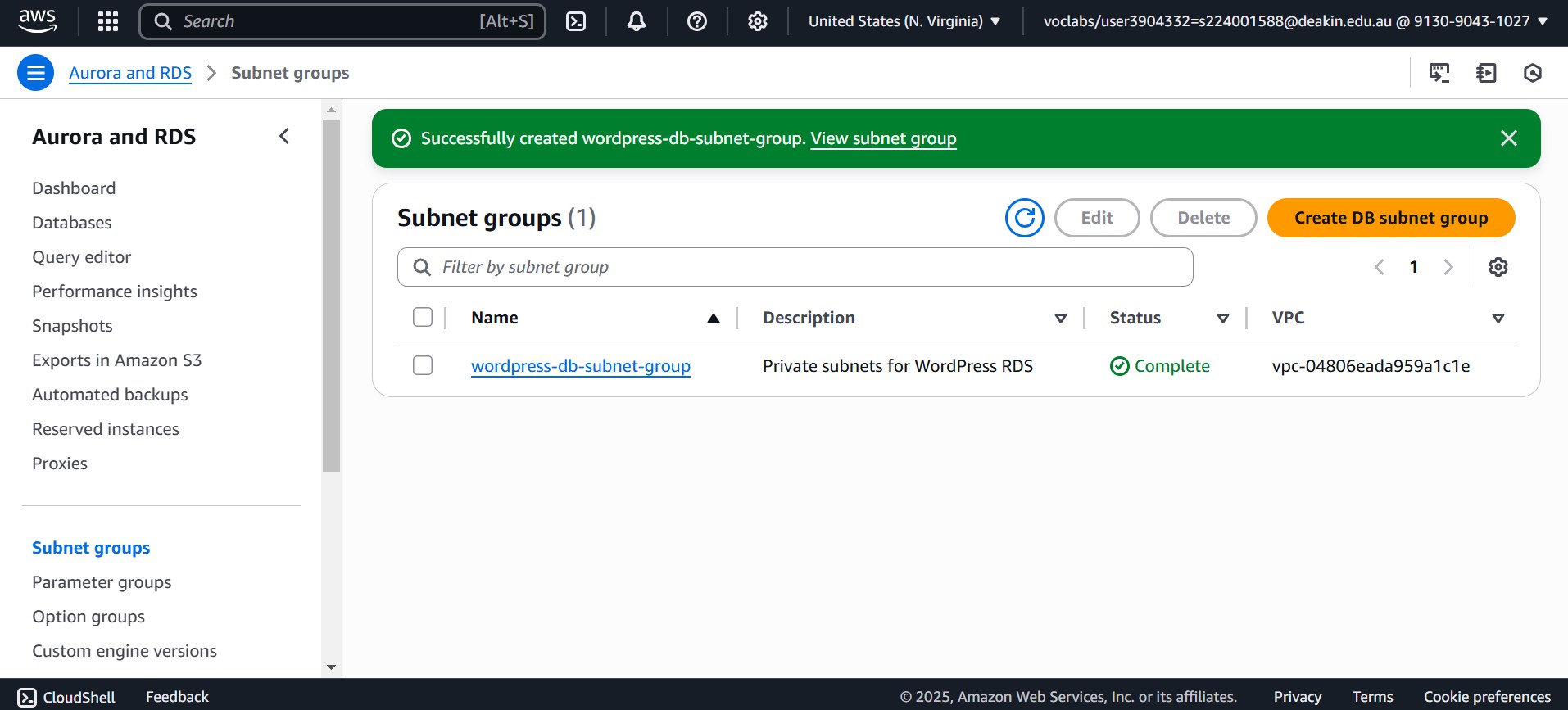
Public & Private Route Tables



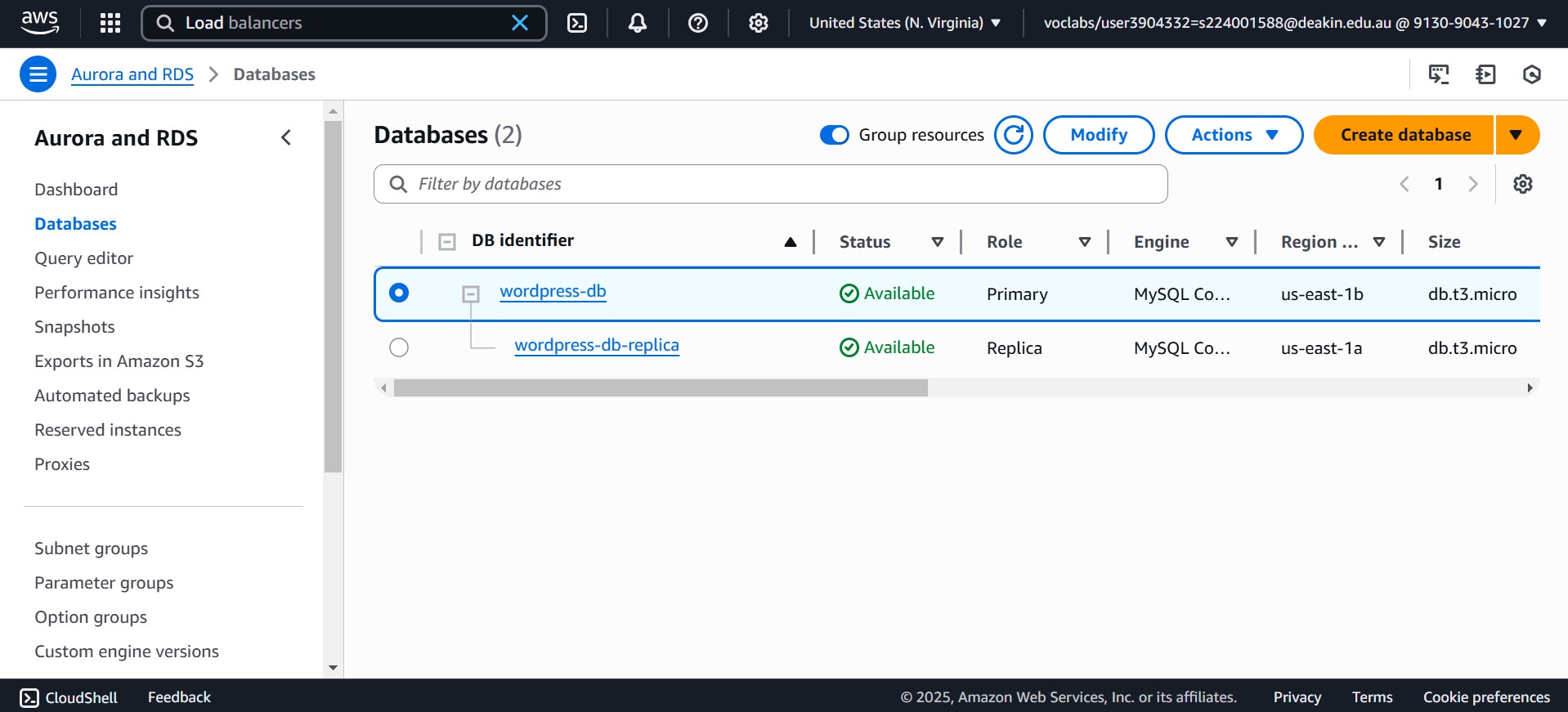
# Step 2: Launch RDS (MySQL) in Private Subnets

Amazon RDS (MySQL 8.0.35) was deployed in private subnets. It was launched using the free tier template with the following configuration:

* Public access is disabled to enhance security.
* A database subnet group was created to include only the private subnets.
* Multi-AZ deployment was enabled to ensure fault tolerance.
* A read replica was created in a different availability zone to enhance read scalability and redundancy.
* A dedicated security group was configured to allow access only from EC2 instances within the VPC.



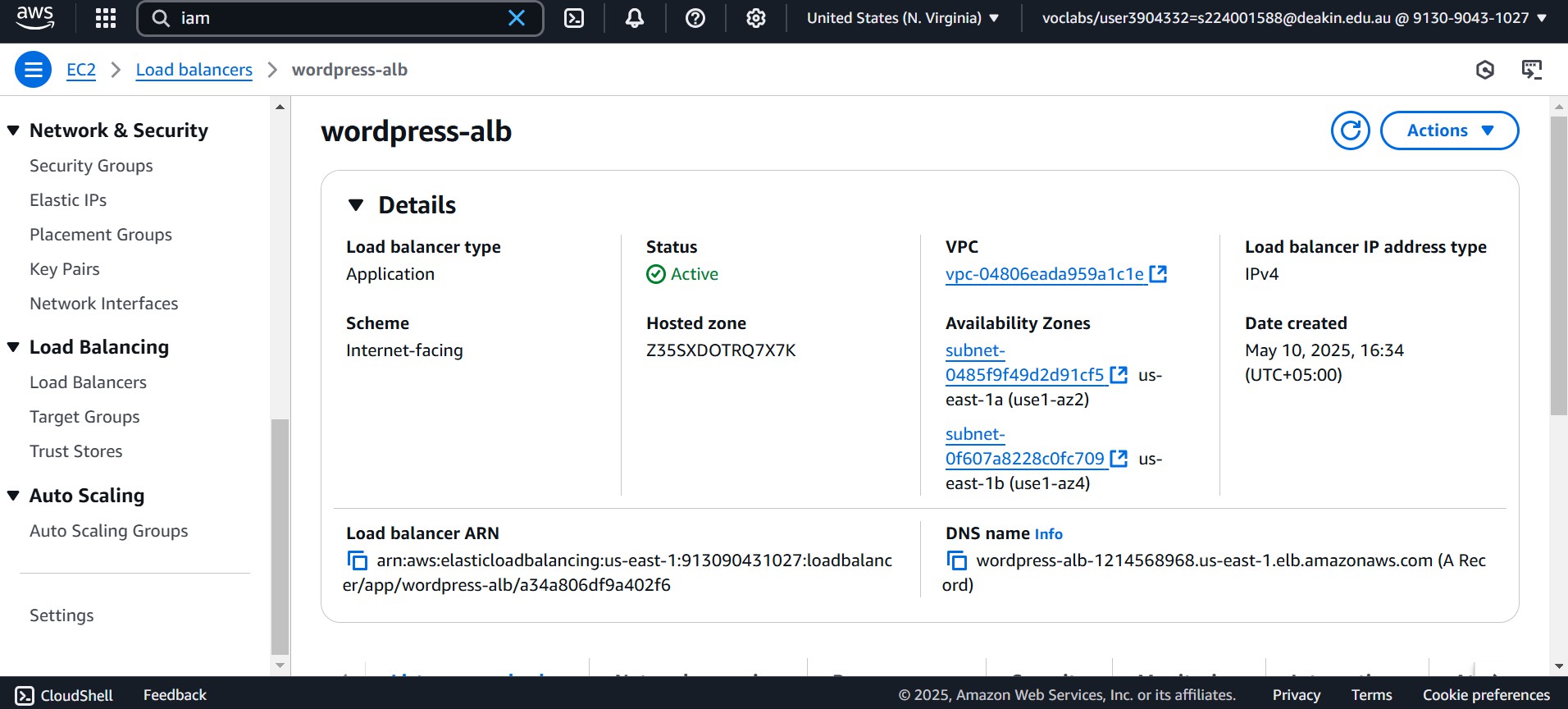
RDS with MySQL Database



# Step 3: Create and Configure the Application Load Balancer (ALB)

An Application Load Balancer was created and deployed in both public subnets to distribute incoming HTTP traffic across multiple EC2 instances. The ALB configuration included:

* A target group with EC2 instances
* A security group that allows HTTP (port 80) traffic from the internet
* A listener rule to forward all traffic to the target group

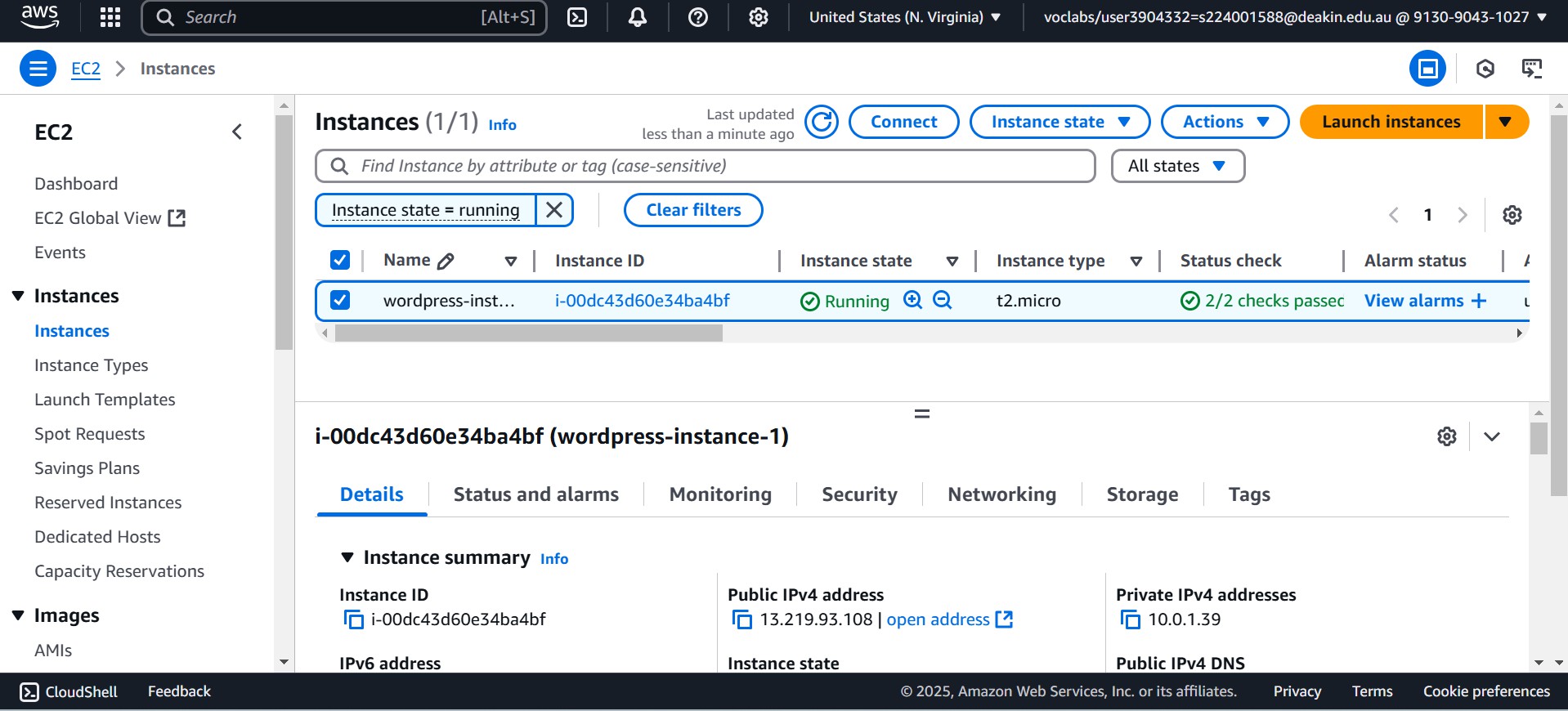


# Step 4: Launch EC2 Instance and Install WordPress

An EC2 instance was launched in a public subnet using the Amazon Linux 2 AMI (t2.micro type). A new key pair was created for SSH access. A custom User Data script was provided to automate the following:

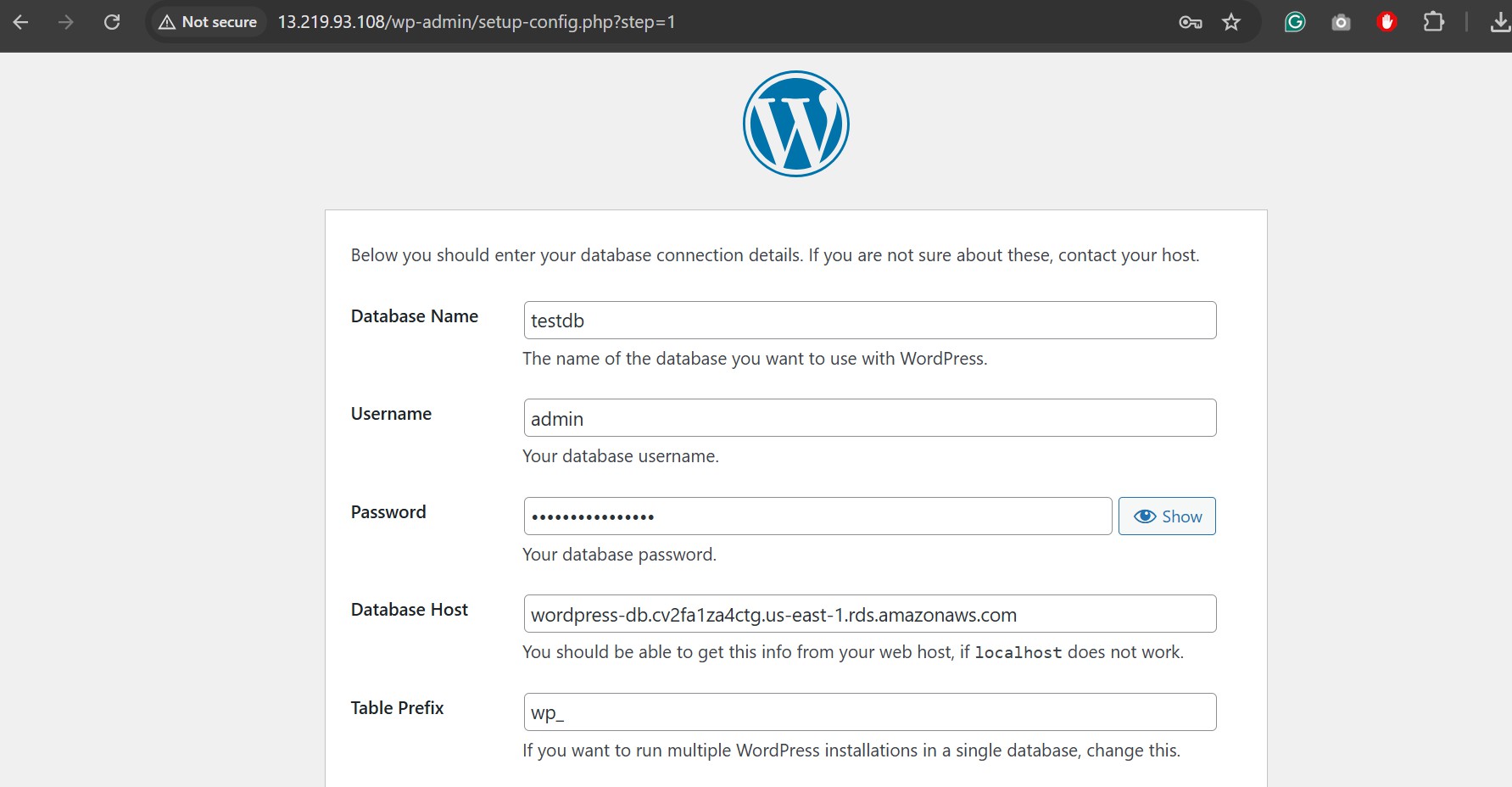
* Install updates
* Install Apache, PHP, and MySQL client
* Configure and extract the WordPress package
* Start the web server

Once launched, the WordPress setup screen was accessed using the EC2 Public IP. The RDS credentials were entered to complete the setup.

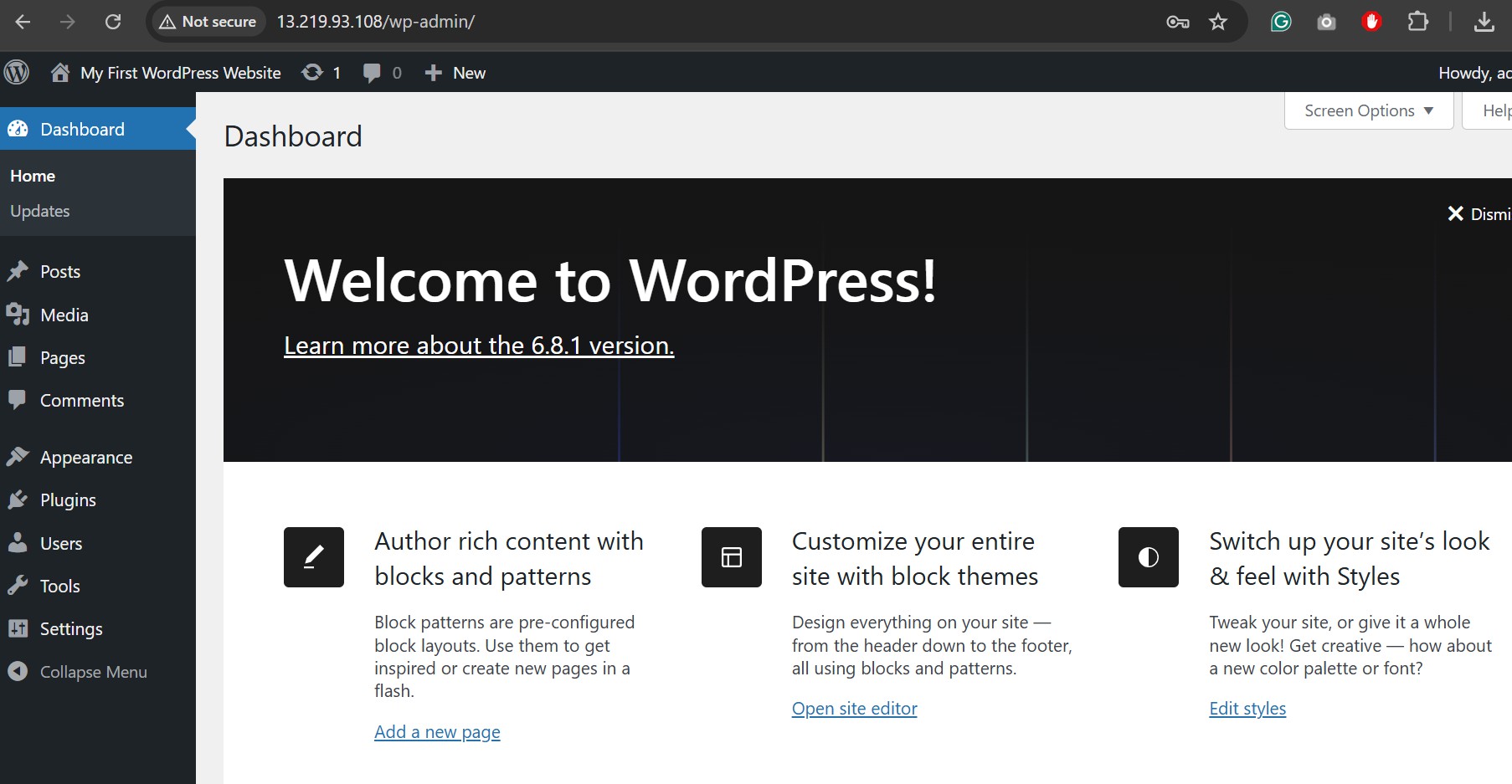


<http://44.203.163.232/>

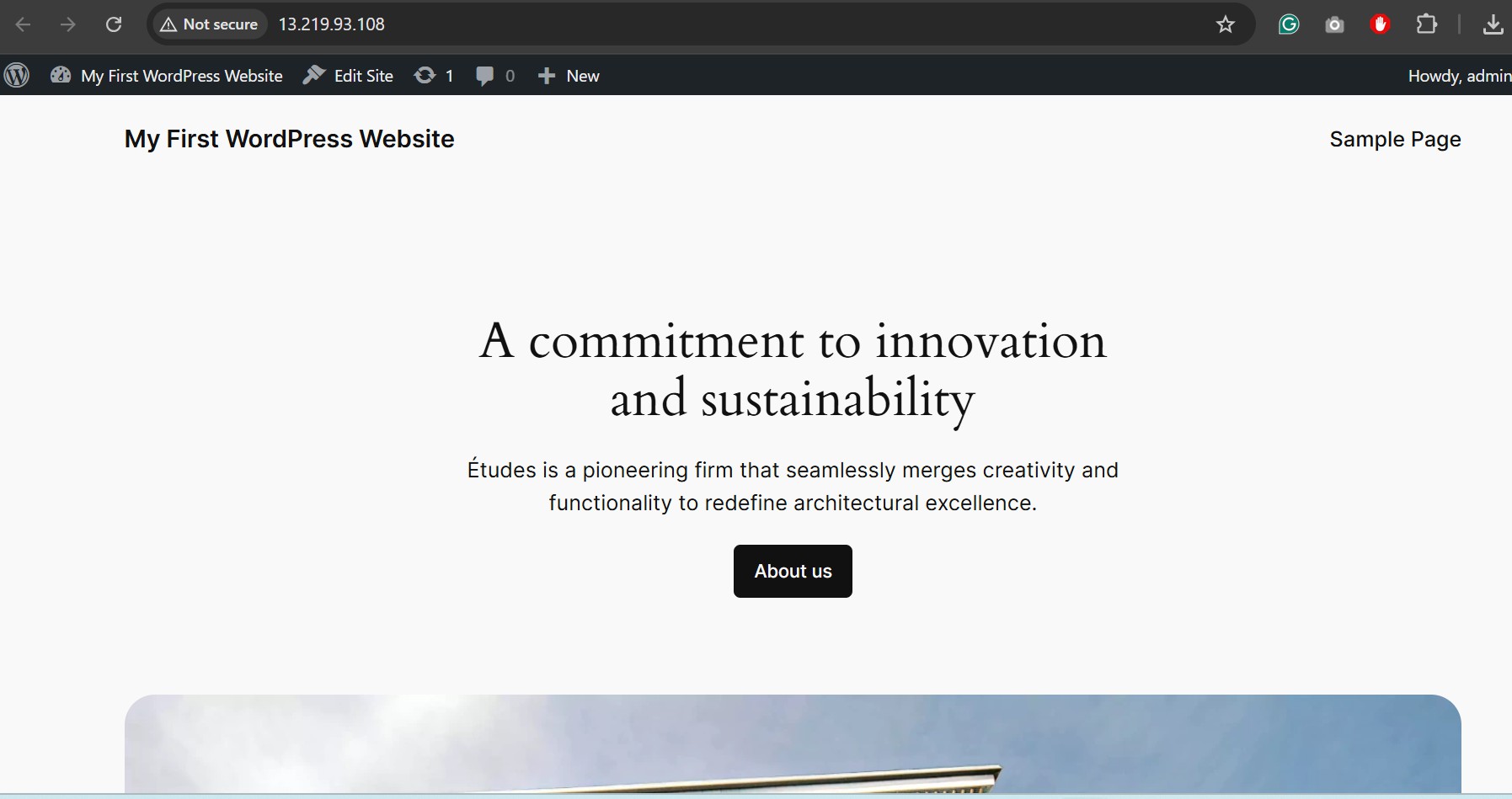
WordPress Setup Configuration



WordPress Dashboard Looks Like



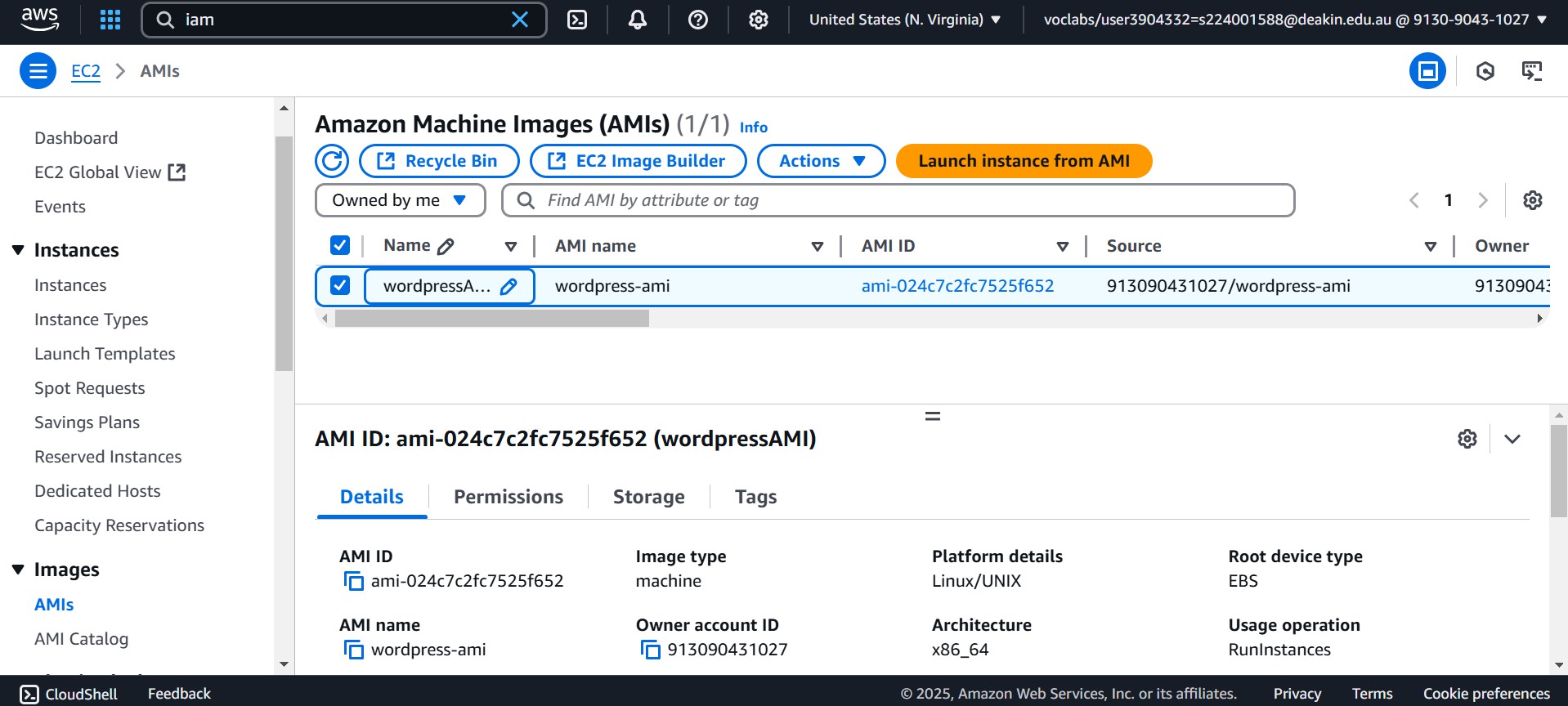
Website Looks Like this



# Step 5: Create an Amazon Machine Image (AMI) & Launch Template

After confirming the WordPress installation was successful, an Amazon Machine Image (AMI) was created from the configured EC2 instance. This AMI was used to create a launch template. The template included:

* The custom AMI
* t2.micro instance type
* Key pair
* A security group allowing traffic from ALB only



Launch Template for AutoScaling Group (ASG)

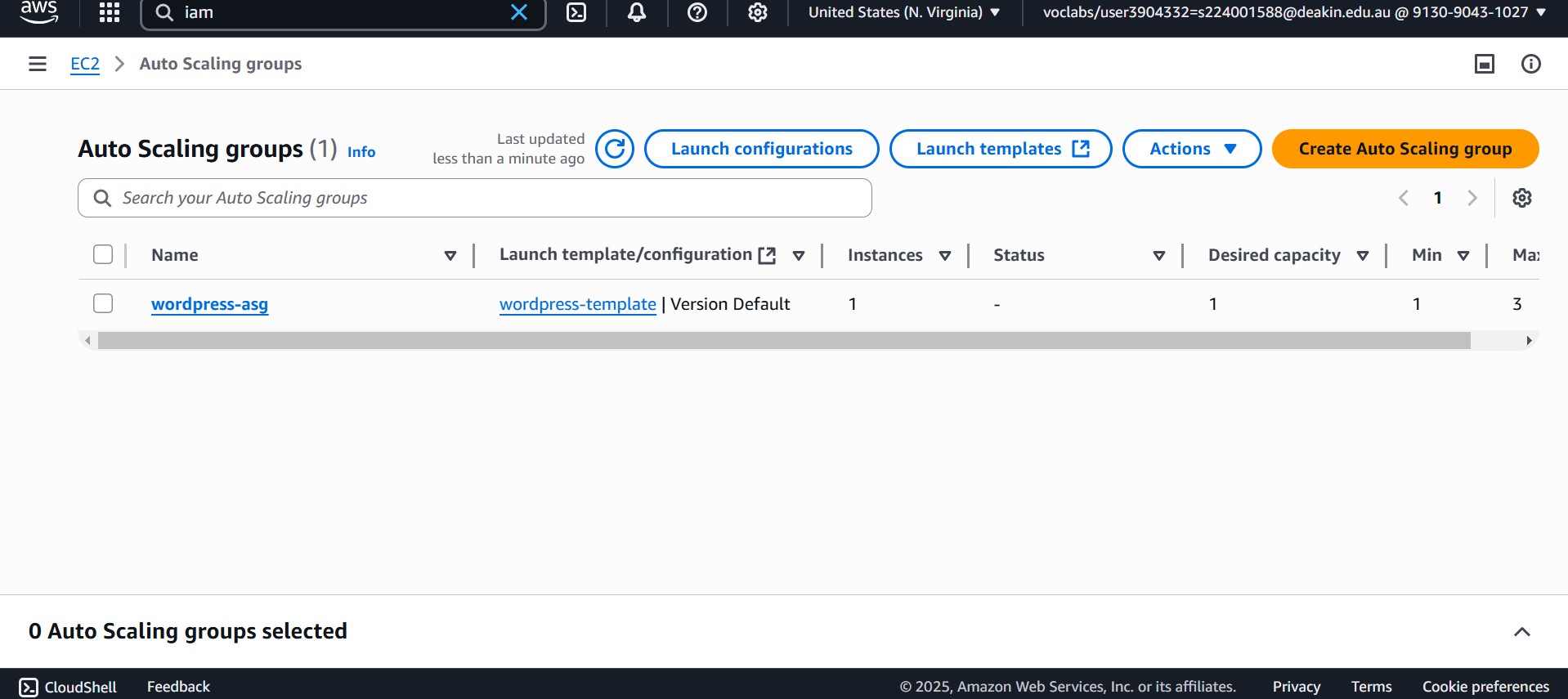


# Step 6: Create the Auto Scaling Group (ASG)

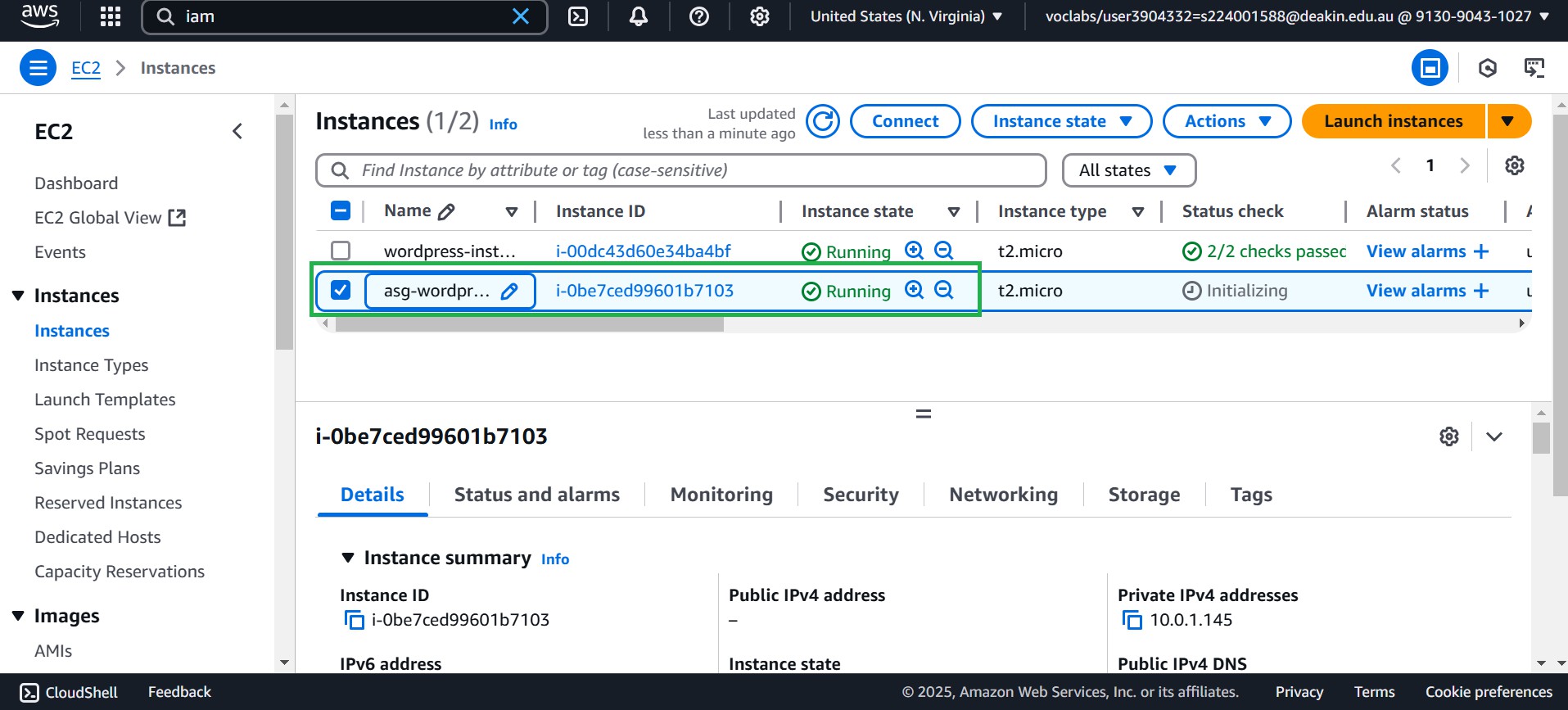
An Auto Scaling Group (ASG) was set up using the launch template with the following policies:

* Minimum instances: 1
* Desired capacity: 1
* Maximum instances: 3
* Scaling policy:
  + Scale out when CPU > 70%
  + Scale in when CPU < 25%

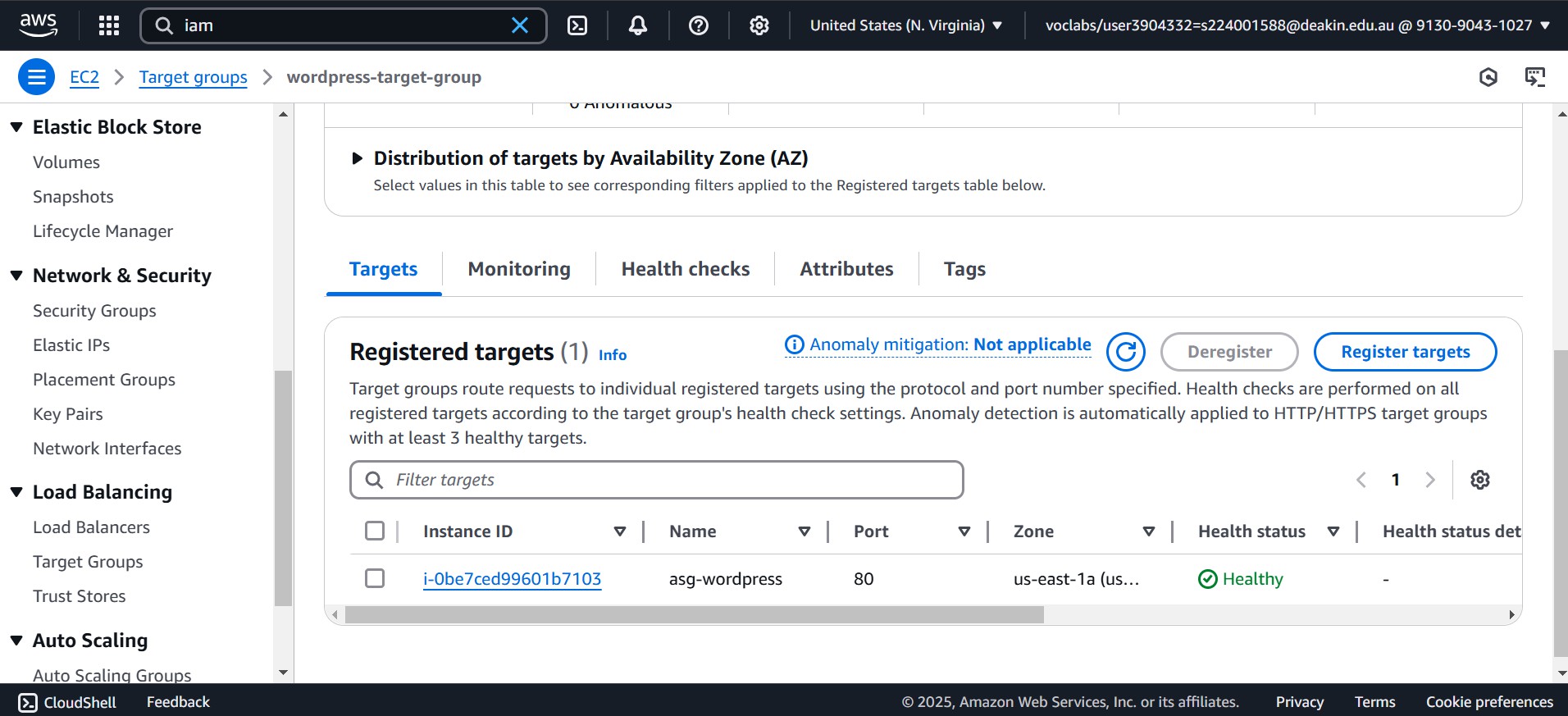
Instances were launched in the private subnets. The ALB target group was attached to the ASG to automatically distribute traffic.



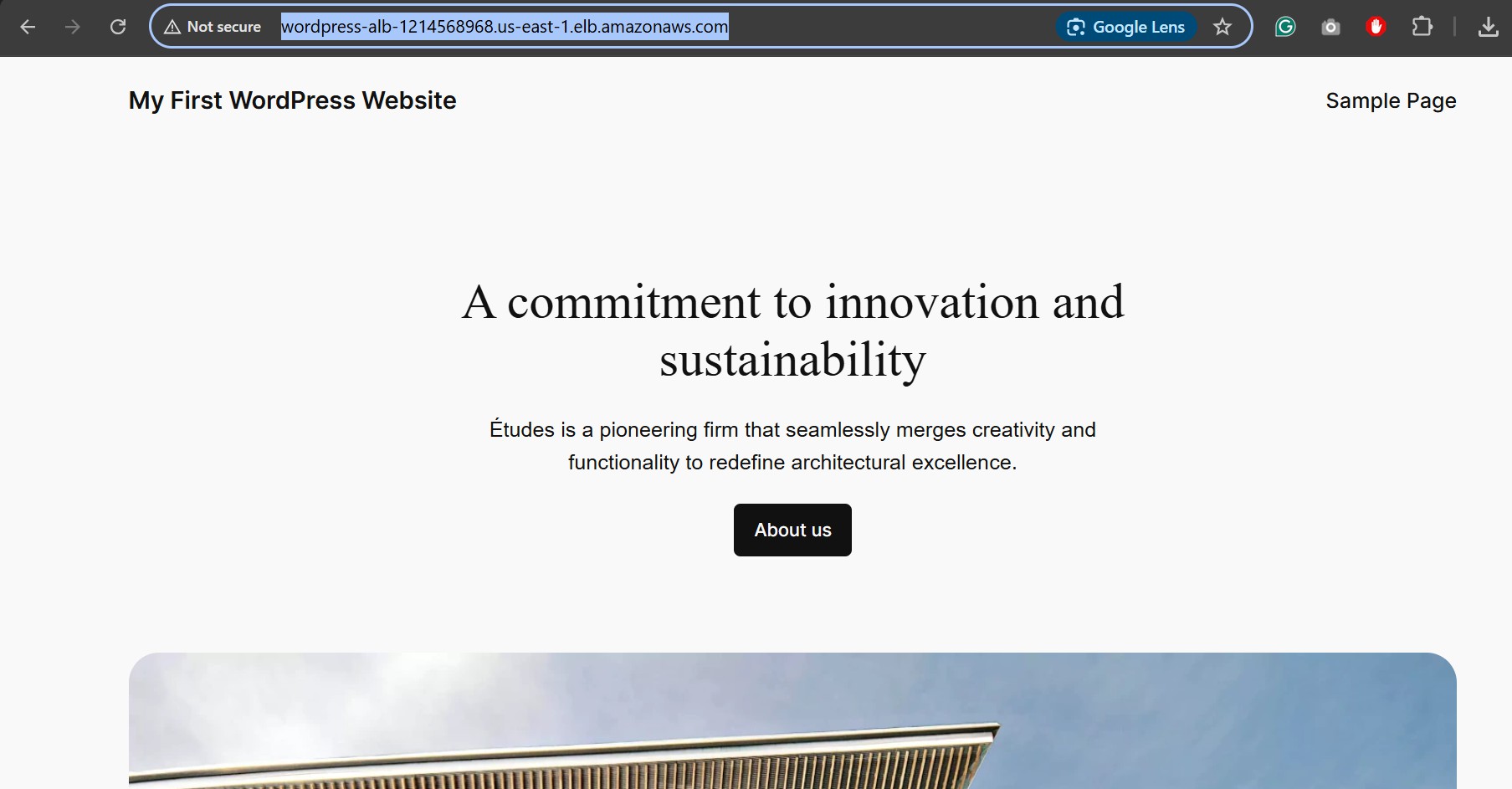
Instance Launched by ASG



EC2 Instanced launched by ASG - Targets for Load Balancer



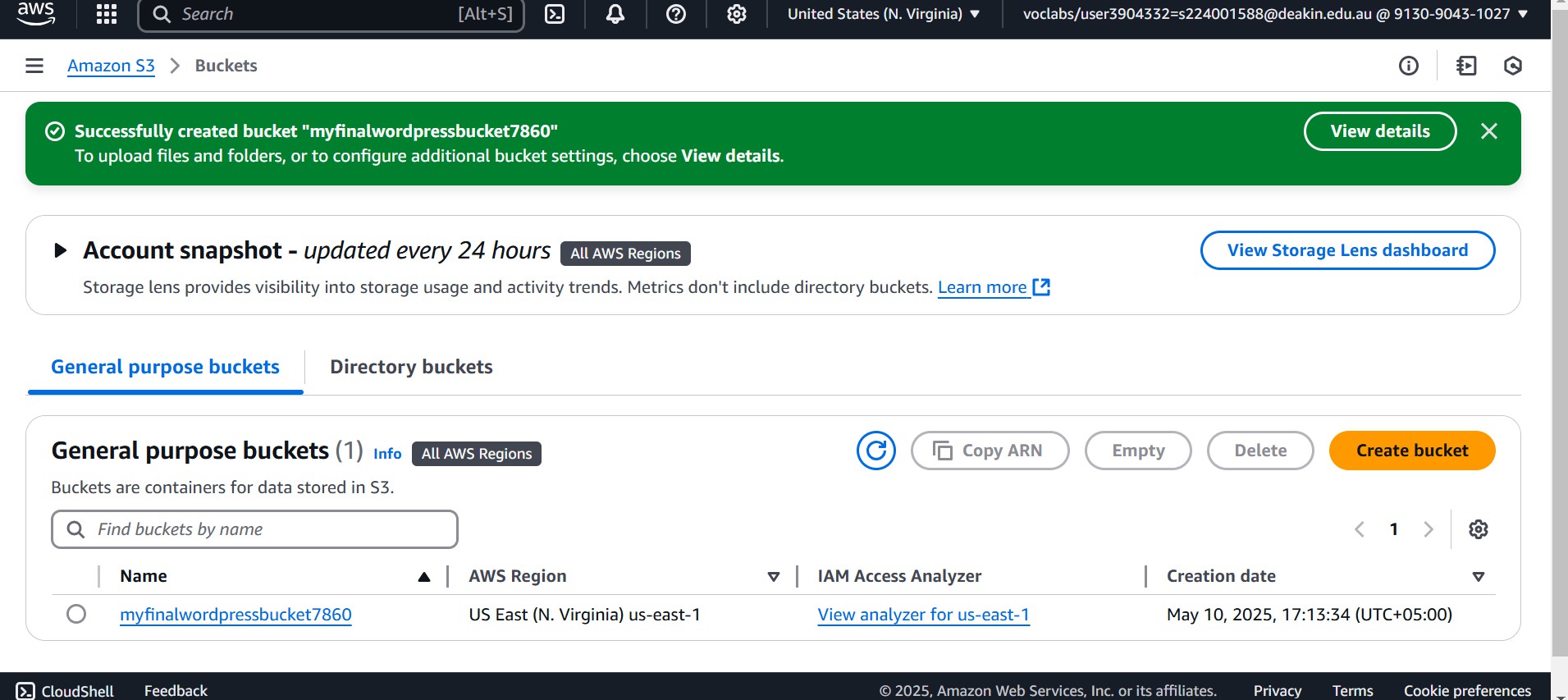
Website Access through the Load Balancer



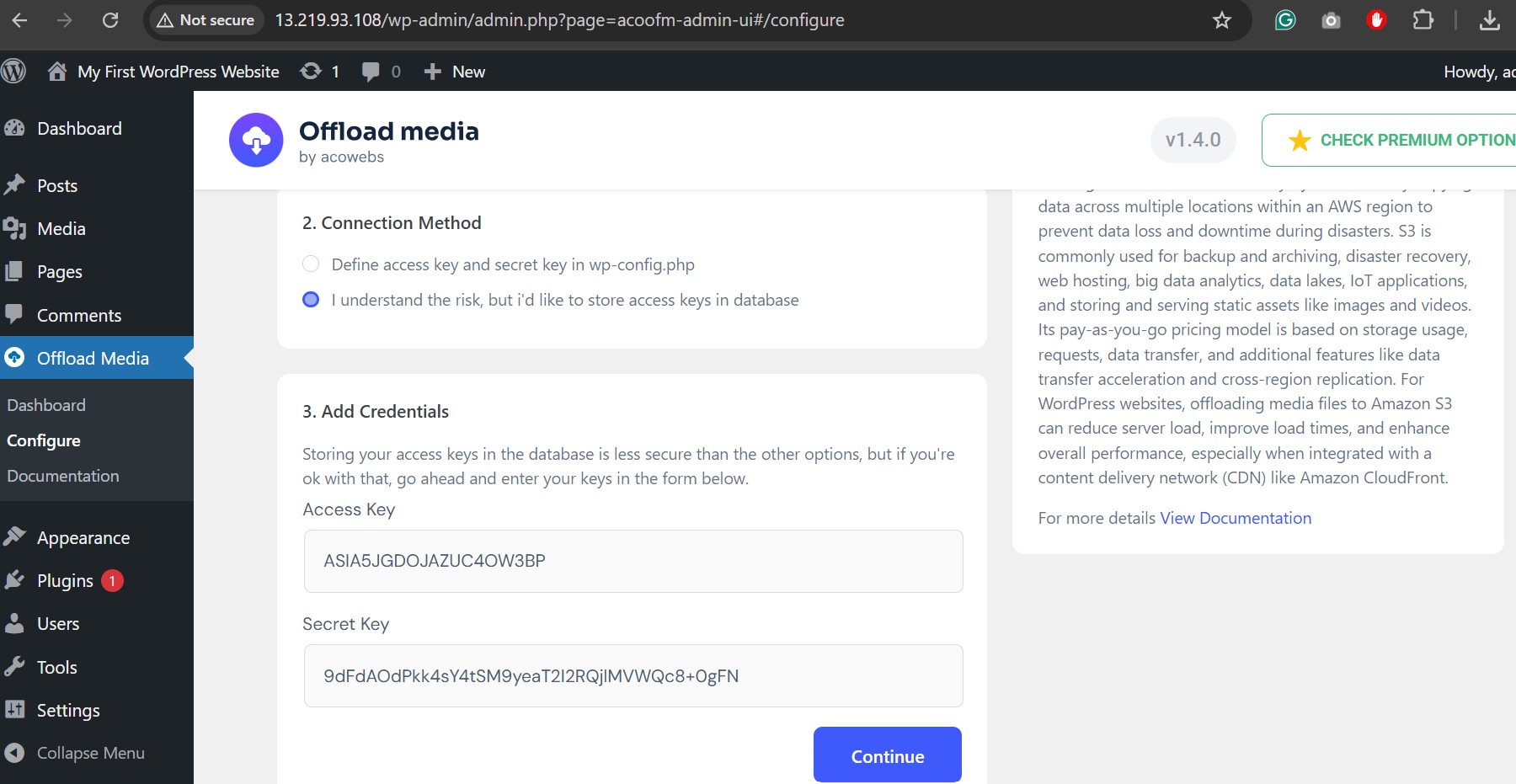
<http://wordpress-alb-1214568968.us-east-1.elb.amazonaws.com/>

# Step 7: Create an S3 Bucket and Configure WordPress Plugin

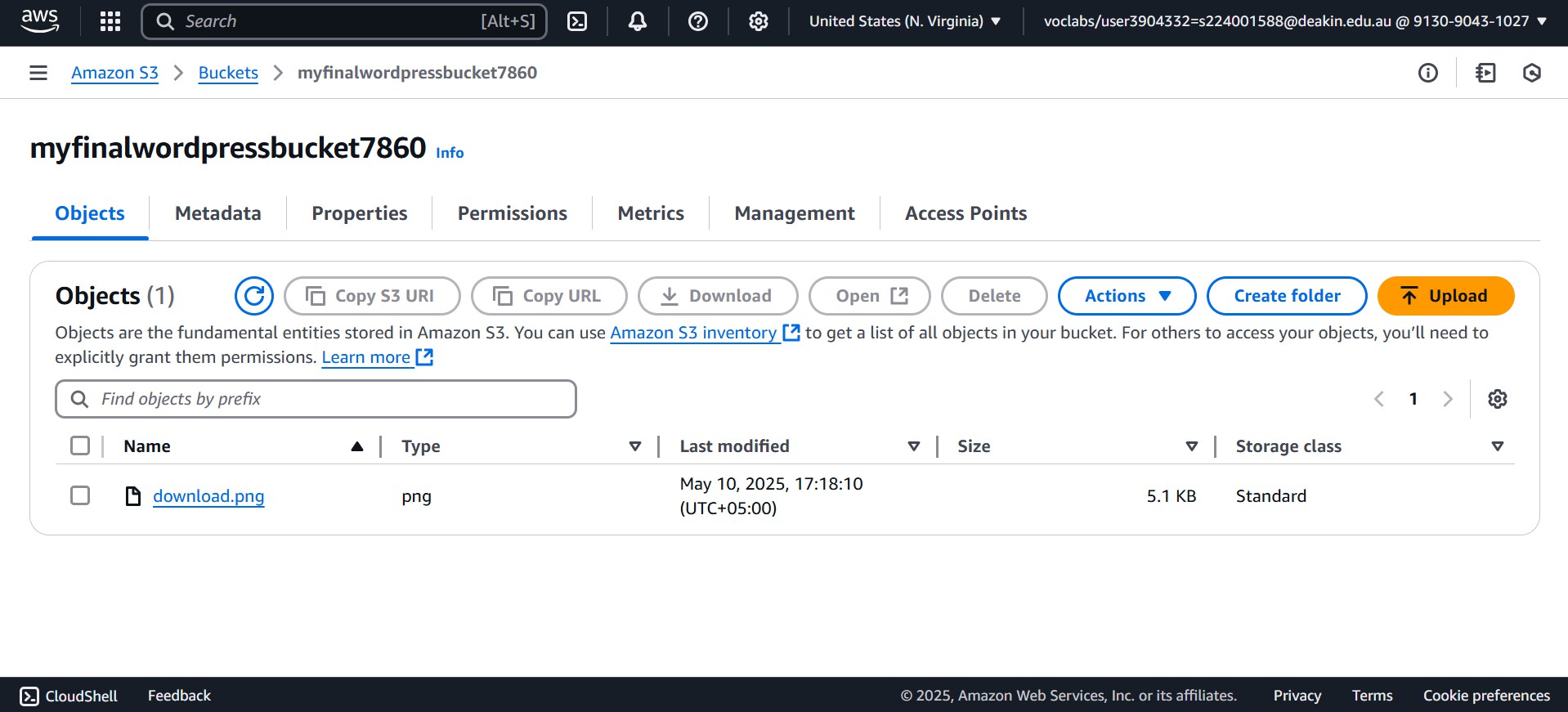
An S3 bucket was created with public access disabled. Within WordPress, the “Offload Media” plugin was installed and configured using AWS credentials. The plugin was tested by uploading media files and verifying their storage in S3.



Installed and integrated the Plugin in WordPress



Uploaded an Object to S3

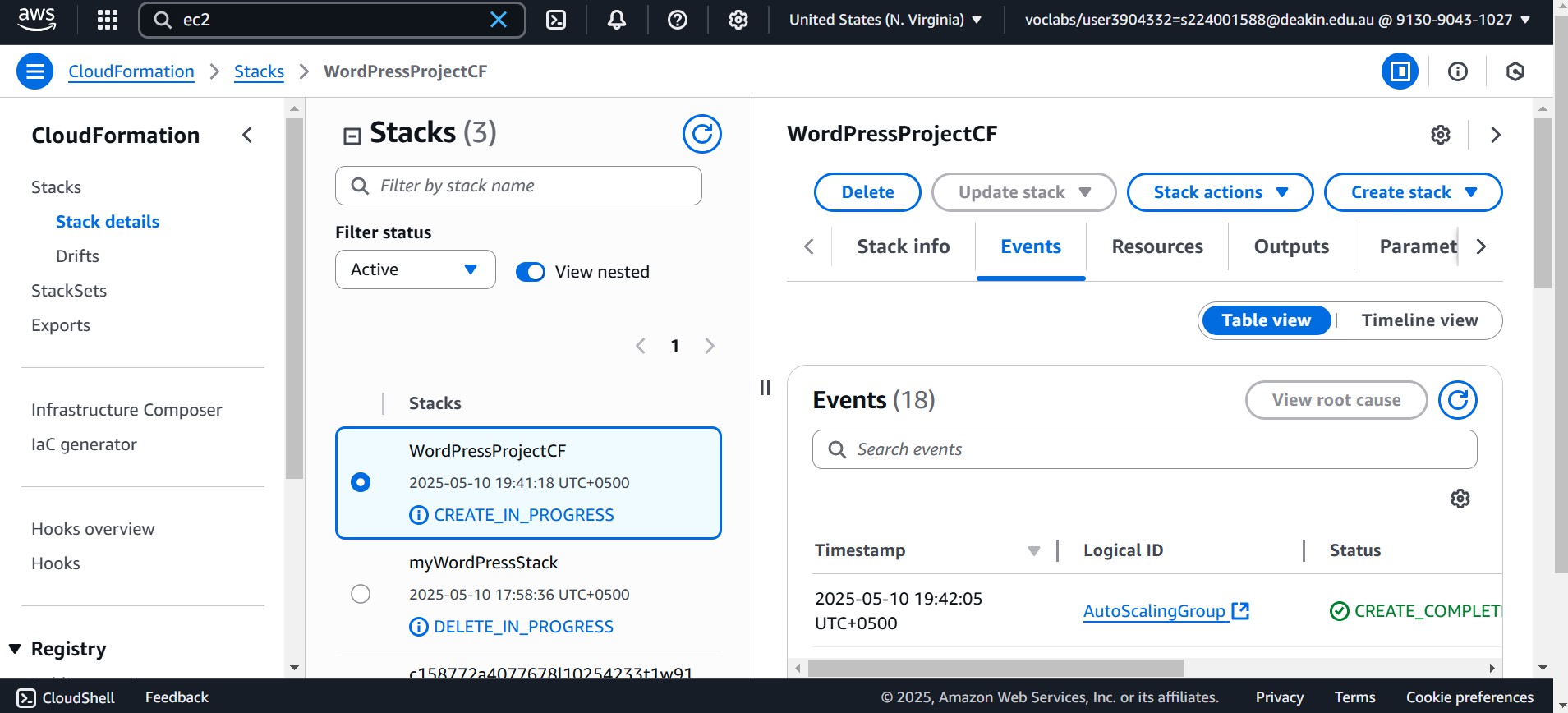


# Step 8: CloudFormation Stack – Infrastructure as Code

Infrastructure as Code was implemented using two CloudFormation templates:

* Networking Template: Created the VPC, Subnets, IGW, NAT, and route tables.
* Application Template: Provisioned EC2, ALB, ASG, RDS, and necessary roles/security groups.

Stacks were launched and tested in an isolated environment to ensure repeatable and modular deployments.



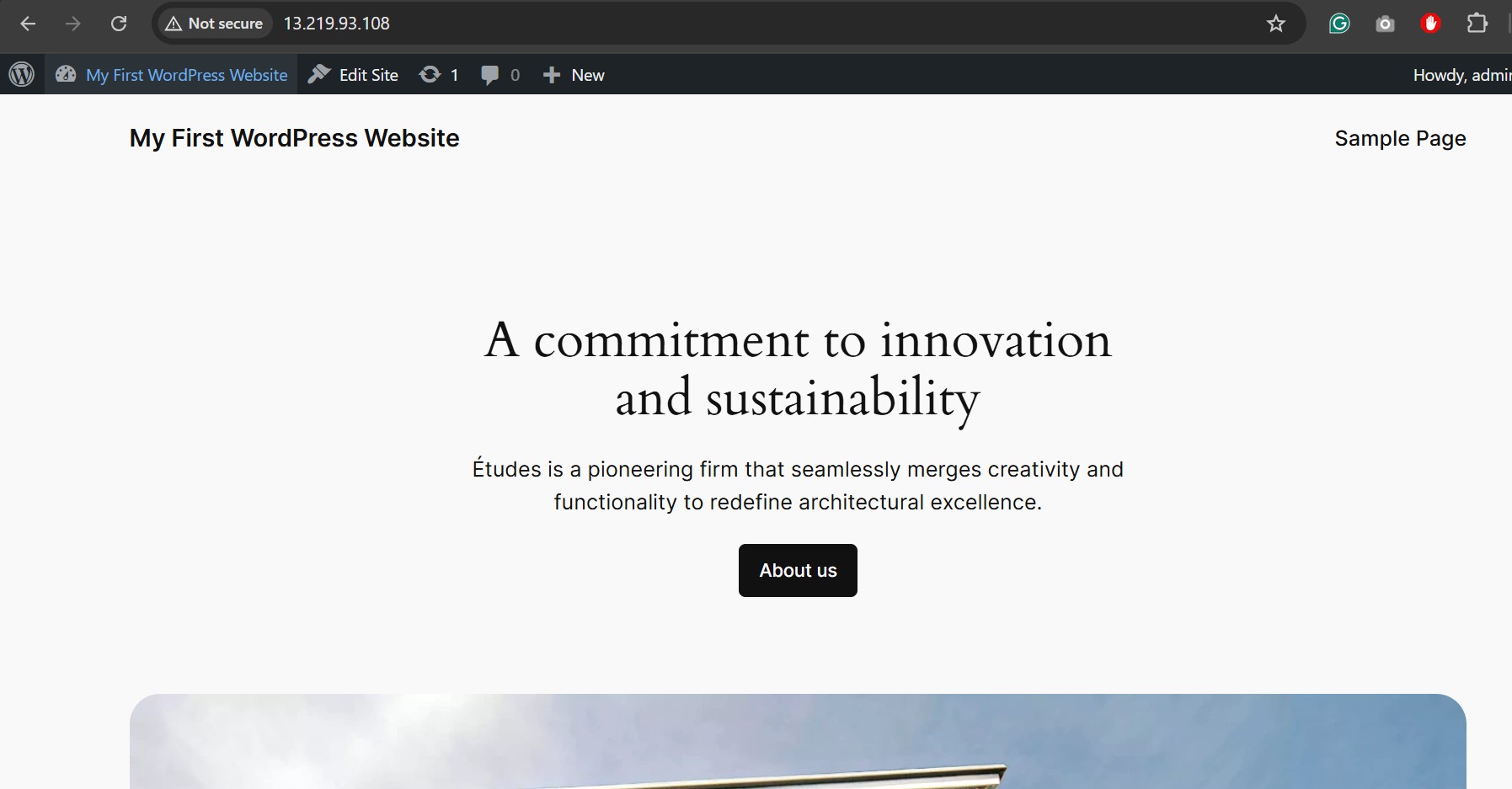
# Step 9: Testing and Validation

Several tests were conducted:

* WordPress was accessible via the ALB DNS and EC2 IP
* Terminating an EC2 instance triggered ASG to launch a new one
* Uploading media files and storing them in S3 via the plugin
* RDS failover was simulated and verified through zone switching
* EC2 was accessible through Systems Manager Session Manager

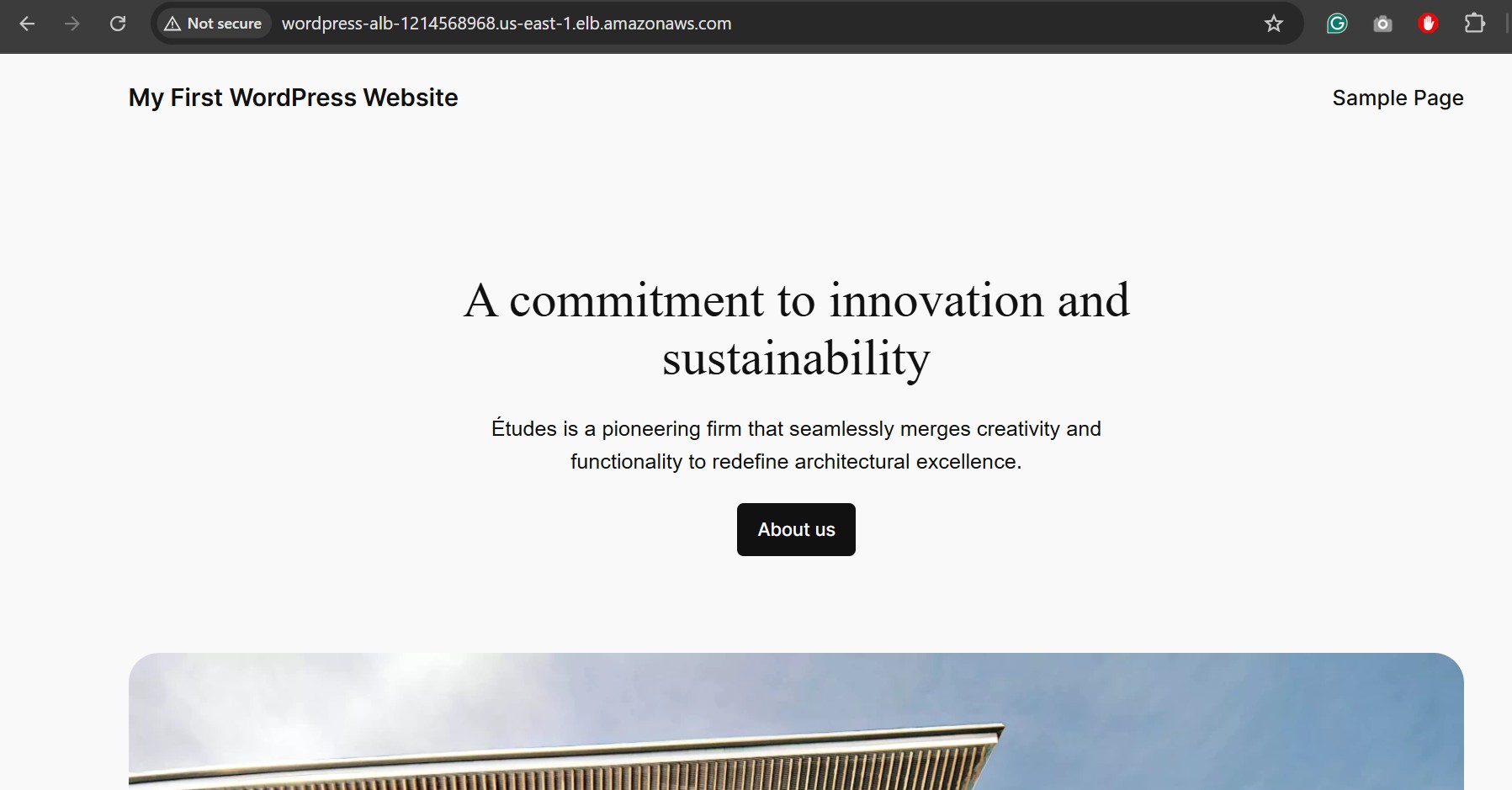
Overall, the implementation phase ensured that all requirements outlined in the project specification were addressed thoroughly. The architecture successfully integrated essential AWS components into a cohesive, secure, and scalable application deployment. Each component was tested for reliability and functionality, proving the design's effectiveness in meeting both academic and real-world standards.

# EC2 Instance Testing

****

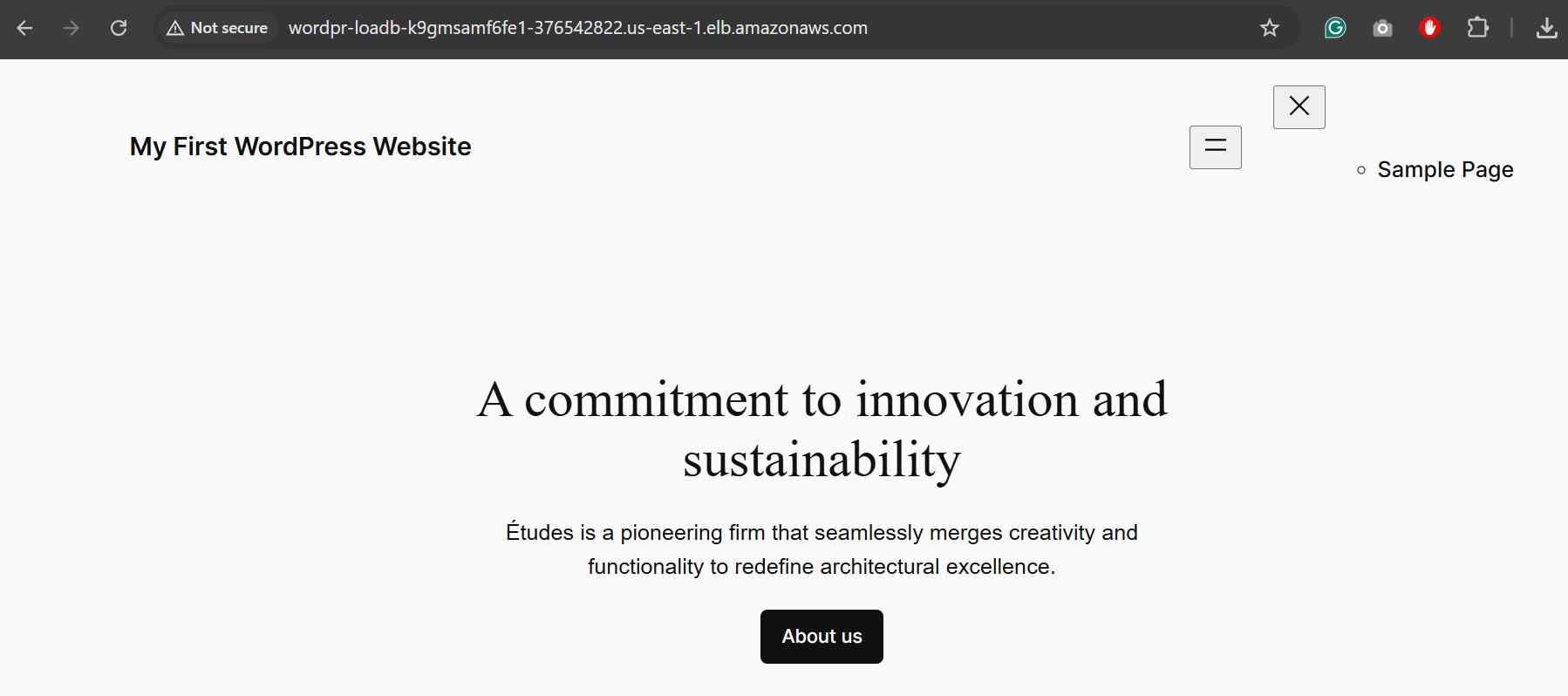
<http://13.219.93.108/>

# Testing Through Load Balancer

****

<http://wordpress-alb-1214568968.us-east-1.elb.amazonaws.com/>

# Testing Through CloudFormation Created Load Balancer

****

<http://wordpr-loadb-k9gmsamf6fe1-376542822.us-east-1.elb.amazonaws.com/>

# Discussion and Reflections

The deployment validated key cloud computing principles such as high availability, scalability, and automation. All infrastructure components worked as expected. Auto Scaling replaced failed instances effectively, and S3 integration ensured durable media storage. One

challenge was IAM key management for plugin integration, which required proper access configuration. CloudFormation made re-deployment efficient and consistent.

# Conclusion

This project successfully demonstrates the deployment of a robust WordPress architecture using AWS. The use of EC2, RDS, S3, and Auto Scaling ensures performance, availability, and cost-efficiency. Automation via CloudFormation underscores the importance of IaC in modern DevOps practices.

# References

1. Amazon Web Services (AWS) Documentation. Available: <https://docs.aws.amazon.com/>
2. WordPress.org, “Installing WordPress,” Available: <https://wordpress.org/support/article/how-to-install-wordpress/>
3. AWS CloudFormation User Guide. Available:

<https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html>

1. Amazon EC2 Auto Scaling. Available: [https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.ht](https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html) [ml](https://docs.aws.amazon.com/autoscaling/ec2/userguide/what-is-amazon-ec2-auto-scaling.html)

Video link - <https://deakin.au.panopto.com/Panopto/Pages/Viewer.aspx?id=581f7a4b-25e8-48c3-bf32-b2dd00da8f75>