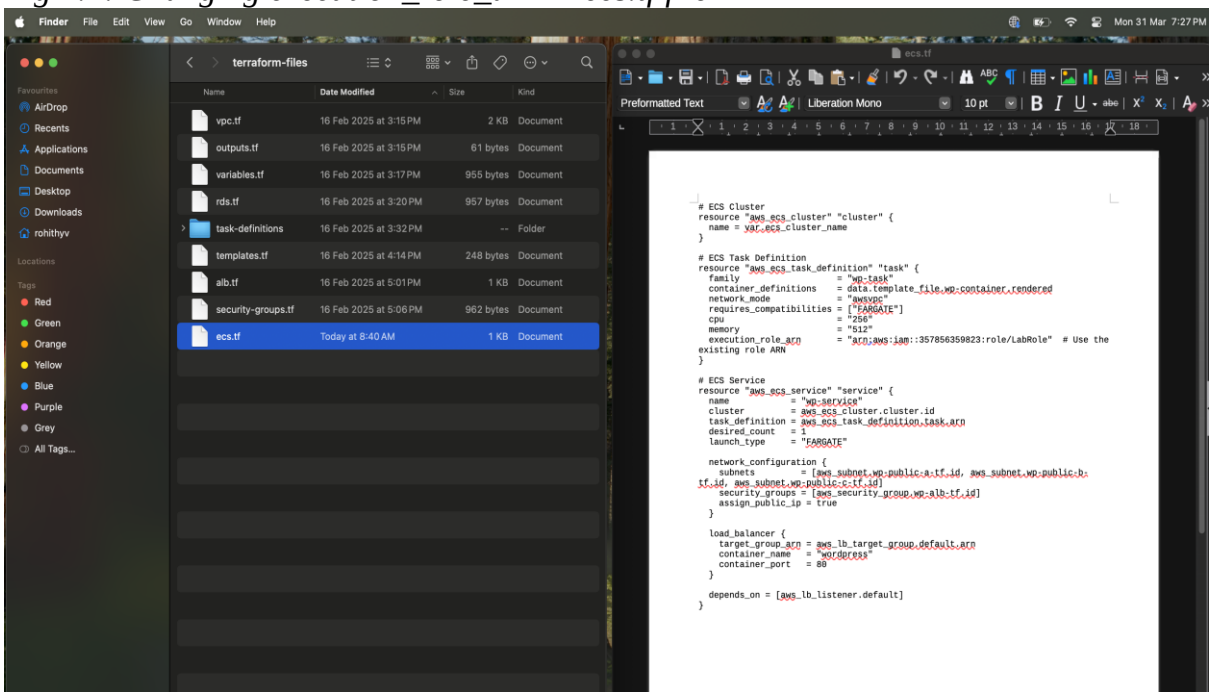
Part – 1: Initial Configuration

Fig 1.1: Changing execution_role_arn in ecs.tf file



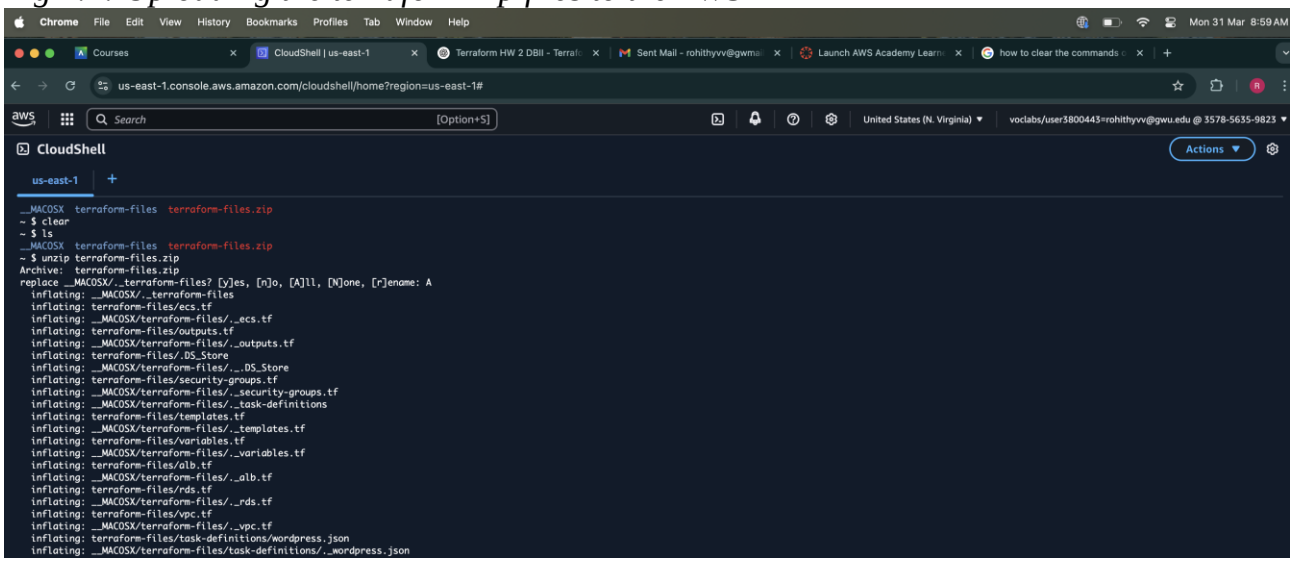
Note: From the Terraform setup files, the ecs.tf file by default has the execution_role_arn value set to a previous user. Update this with your own execution_role_arn, which you can find in the AWS Dashboard under your IAM role.

Key Note:
From the terraform setup files, update the execution_role_arn in ecs.tf file with our own IAM user ARN from AWS dashboard to allow ECS to access AWS services securely.

This step ensures that the ECS task gets the right execution permissions and doesn't fail due to using a previous user's IAM role.

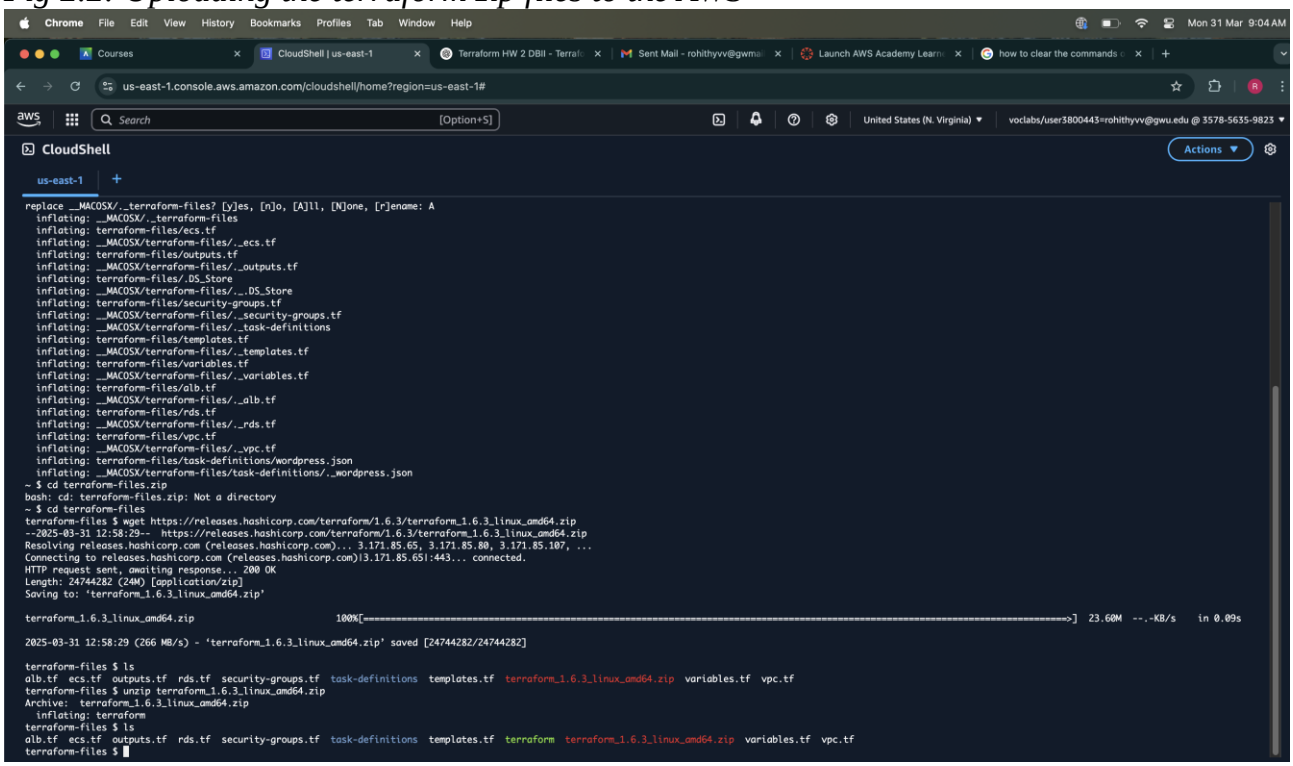
Part – 2: Infrastructure Deployment Using Terraform

Fig 2.1: Uploading the terraform zip files to the AWS



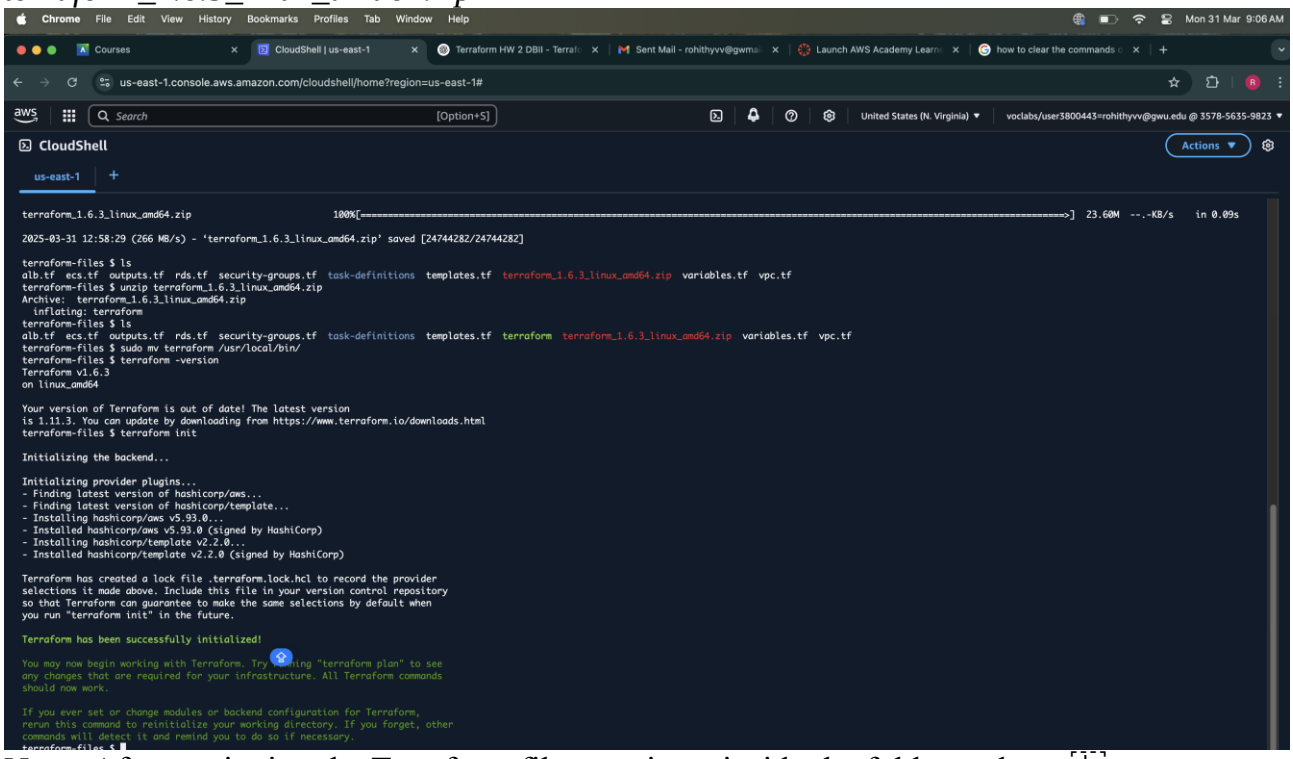
Note: The Terraform zip file is uploaded into AWS CloudShell and is ready to be unzipped for the next steps.

Fig 2.2: Uploading the terraform zip files to the AWS



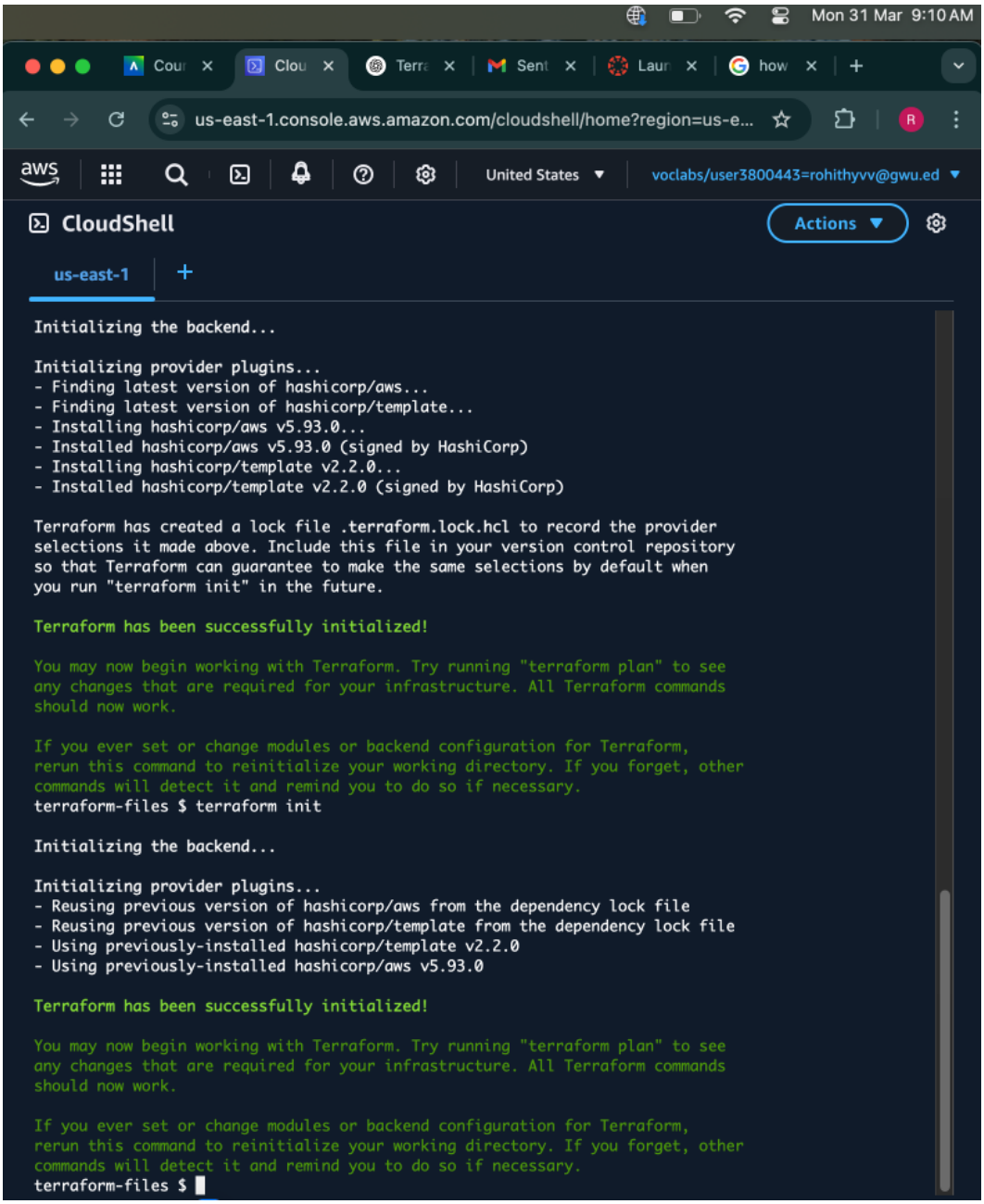
Note: Same as above, showing the terraform-files.zip uploaded into CloudShell and prepared for unzip.

Fig 2.3: Navigate inside the unzipped folder to download the terraform_1.6.3_linux_amd64.zip



Note: After unzipping the Terraform files, navigate inside the folder and run: wget https://releases.hashicorp.com/terraform/1.6.3/terraform_1.6.3_linux_amd64.zip to download the required binary, then unzip it.

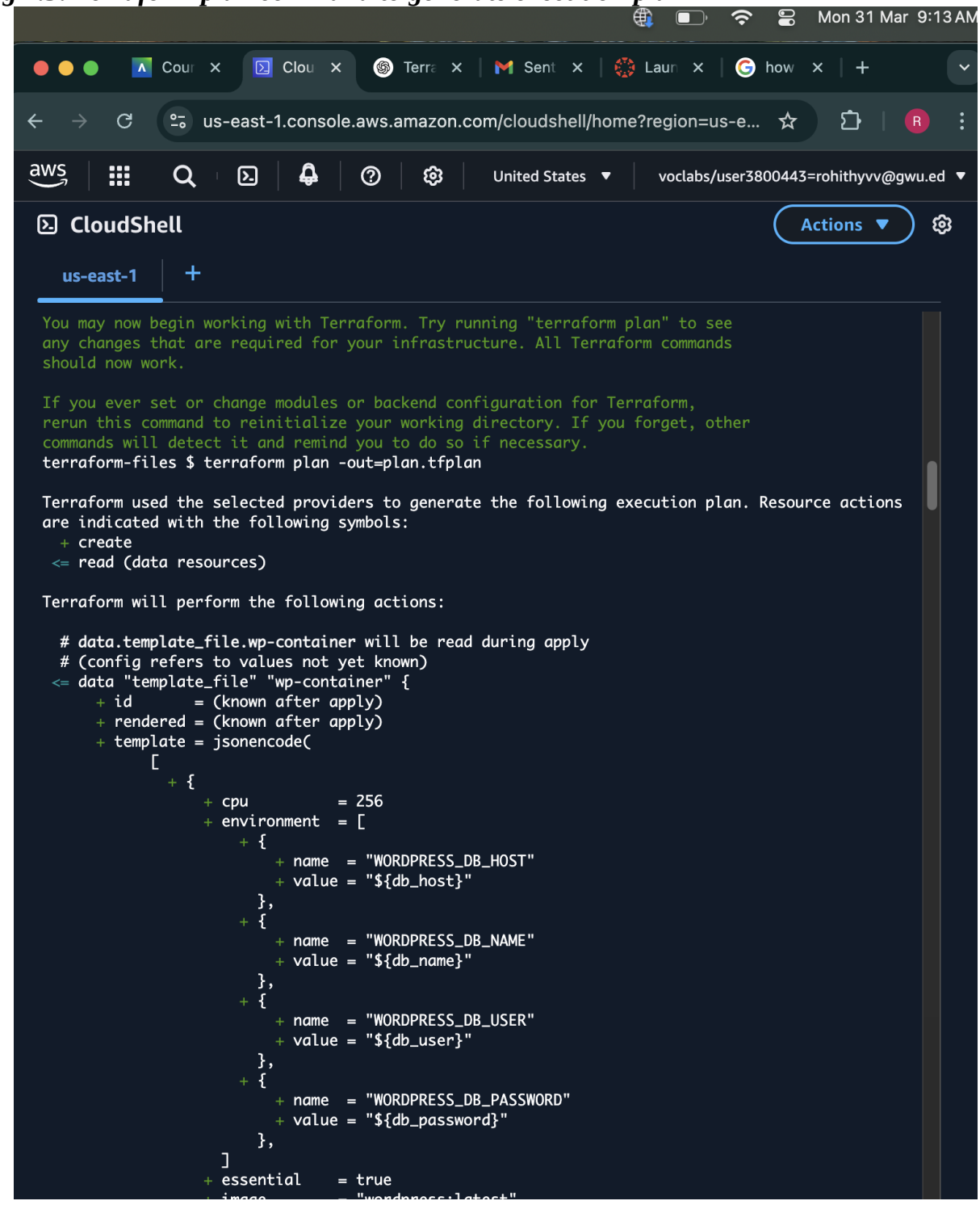
Title: *Fig 2.4: Move the terraform file to root and initialize terraform*



Note: Move the Terraform binary to /usr/local/bin to execute it globally. Then initialize with terraform init and prepare execution using terraform plan -out=plan.tfplan.

plan.tfplan generation

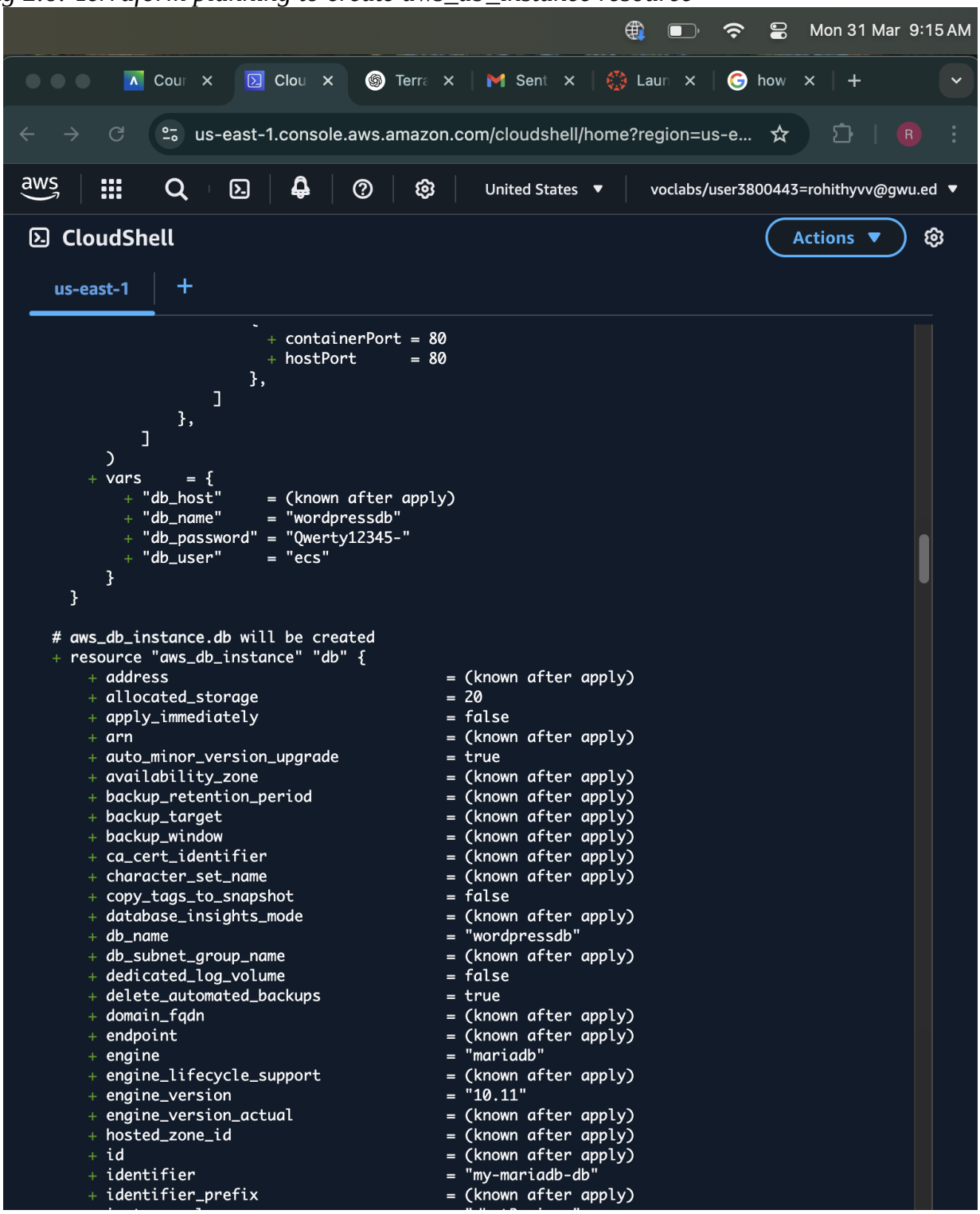
Fig 2.5: Terraform plan command to generate execution plan



Note: This command terraform plan -out=plan.tfplan analyzes the Terraform configuration and prepares an execution plan to create resources. It shows that a template_file data source is being rendered for the WordPress container with environment variables like WORDPRESS_DB_HOST, DB_NAME, DB_USER, and DB_PASSWORD which will be injected into the ECS task during deployment.

RDS Instance Creation Plan

Fig 2.6: Terraform planning to create aws_db_instance resource

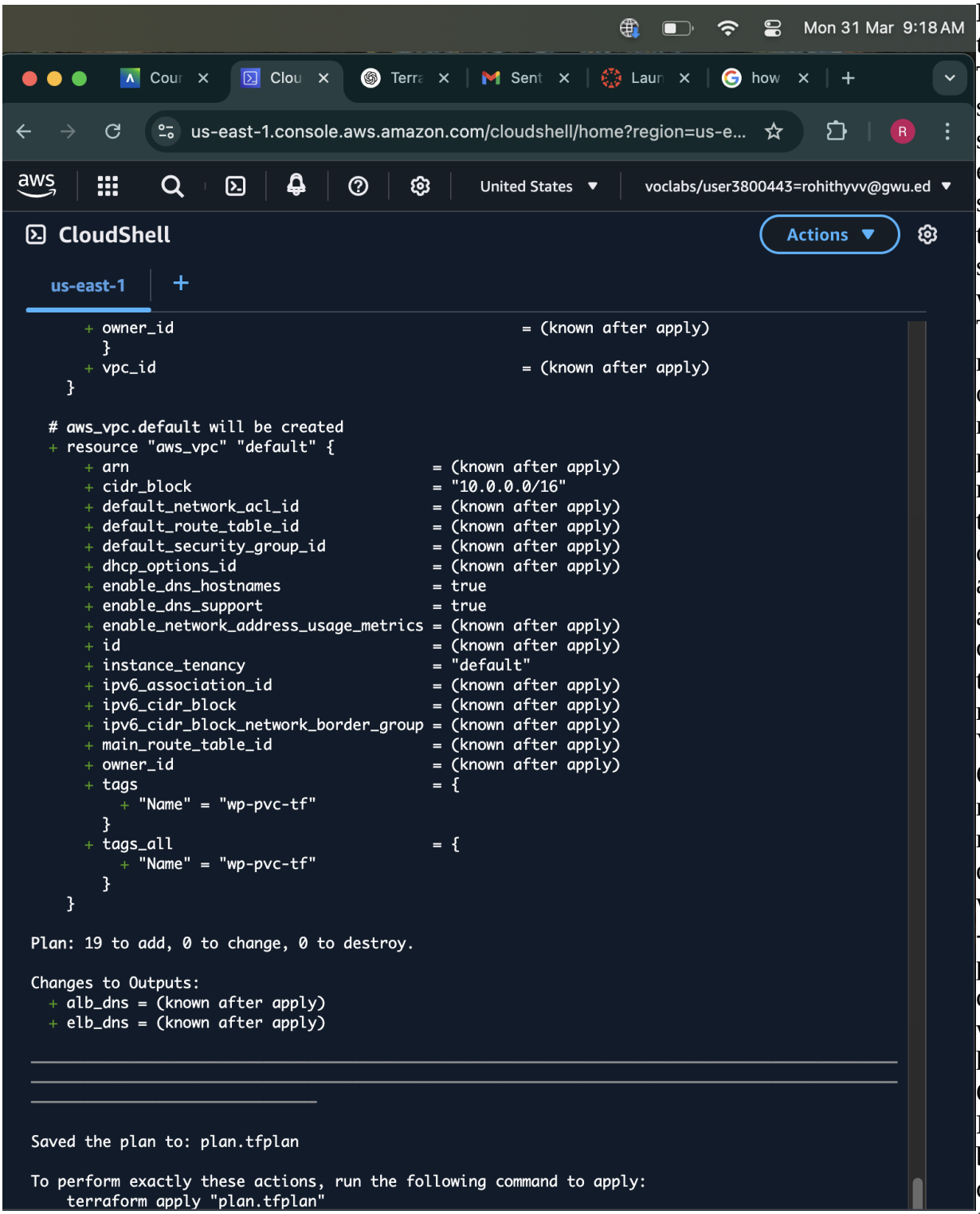


Note: This screenshot shows part of the execution plan where Terraform prepares to create an aws_db_instance resource for the WordPress database. It defines the engine as mariadb,

with a database name wordpressdb, username ecs, and password, and specifies instance class db.t3.micro. This RDS instance will be provisioned as part of the infrastructure.

VPC Resource Creation Plan

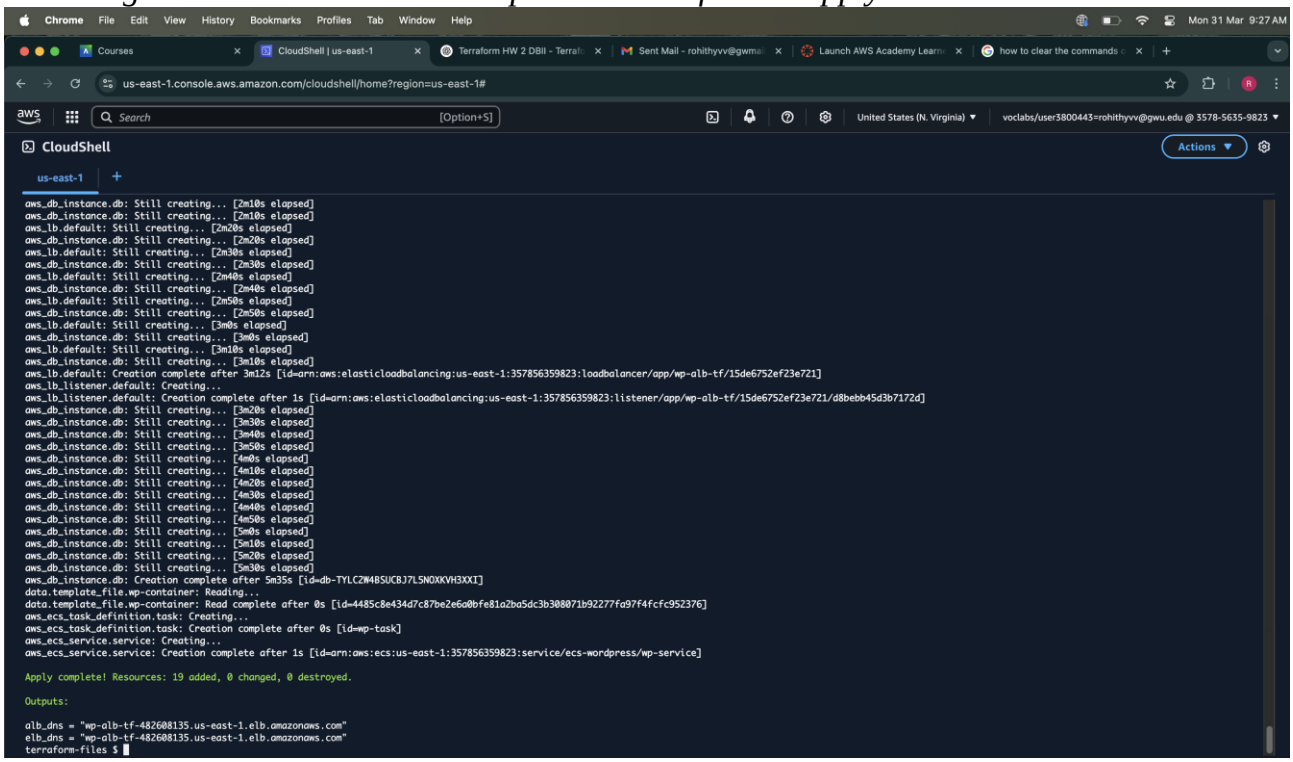
Fig 2.7: Terraform planning to create aws_vpc resource



Note: This screenshot shows Terraform's plan to create a custom VPC named wp-pvc-tf with CIDR block 10.

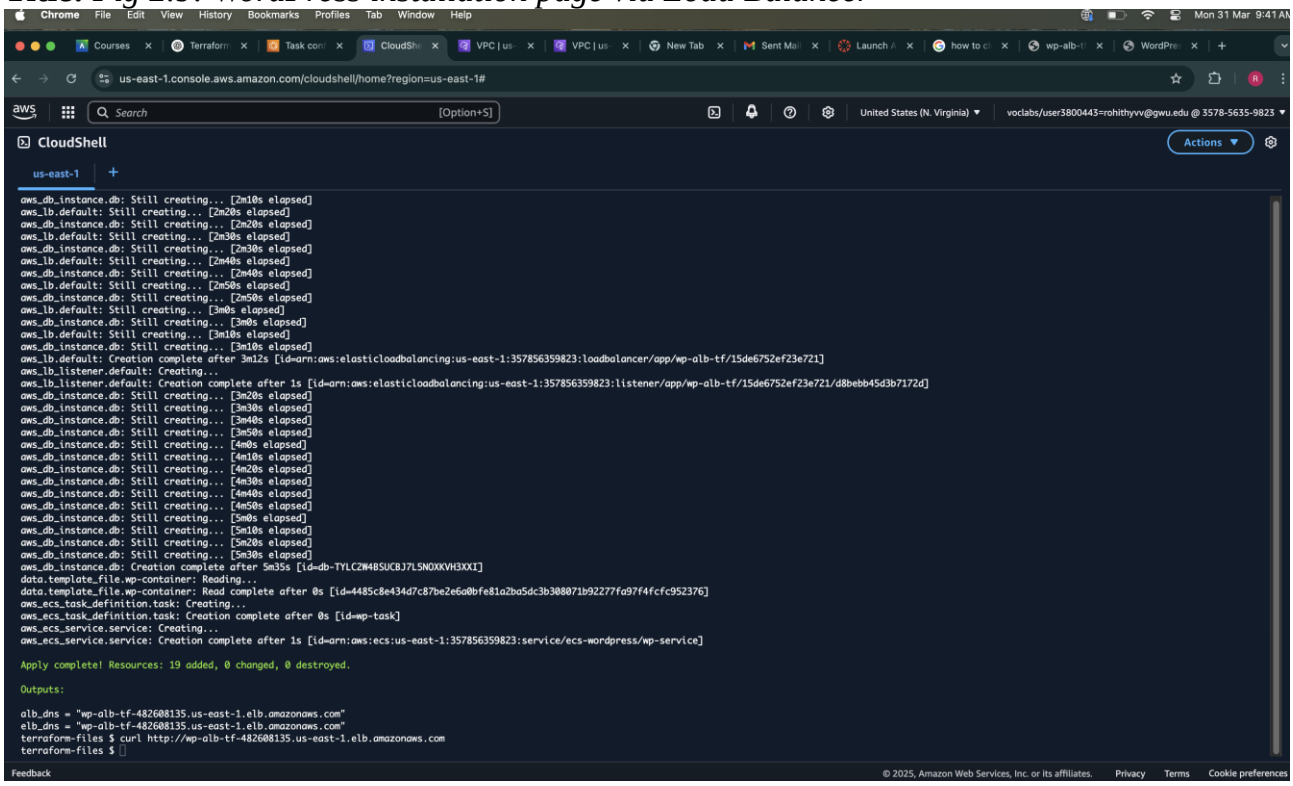
0.0.0/16, DNS support, and hostname enabled. The plan also confirms that 19 resources will be added with no changes or deletions, and that outputs such as alb_dns and elb_dns will be generated after the apply step.

Title: *Fig 2.8: Final command to implement terraform is apply*



Note: Apply the execution plan using terraform apply plan.tfplan. This command deploys all resources including VPC, ECS, ALB, and RDS.

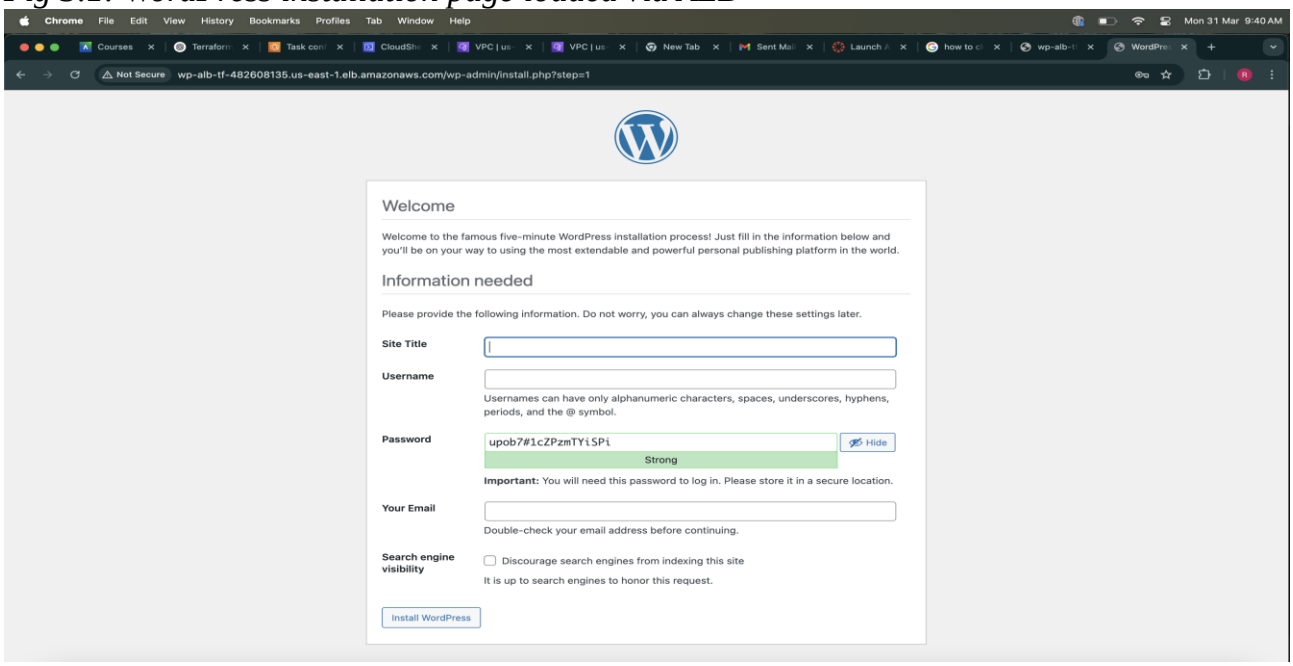
Title: *Fig 2.9: WordPress installation page via Load Balancer*



Note: Access the public ALB URL to open the WordPress installation page. This confirms the ECS container is running the WordPress service.

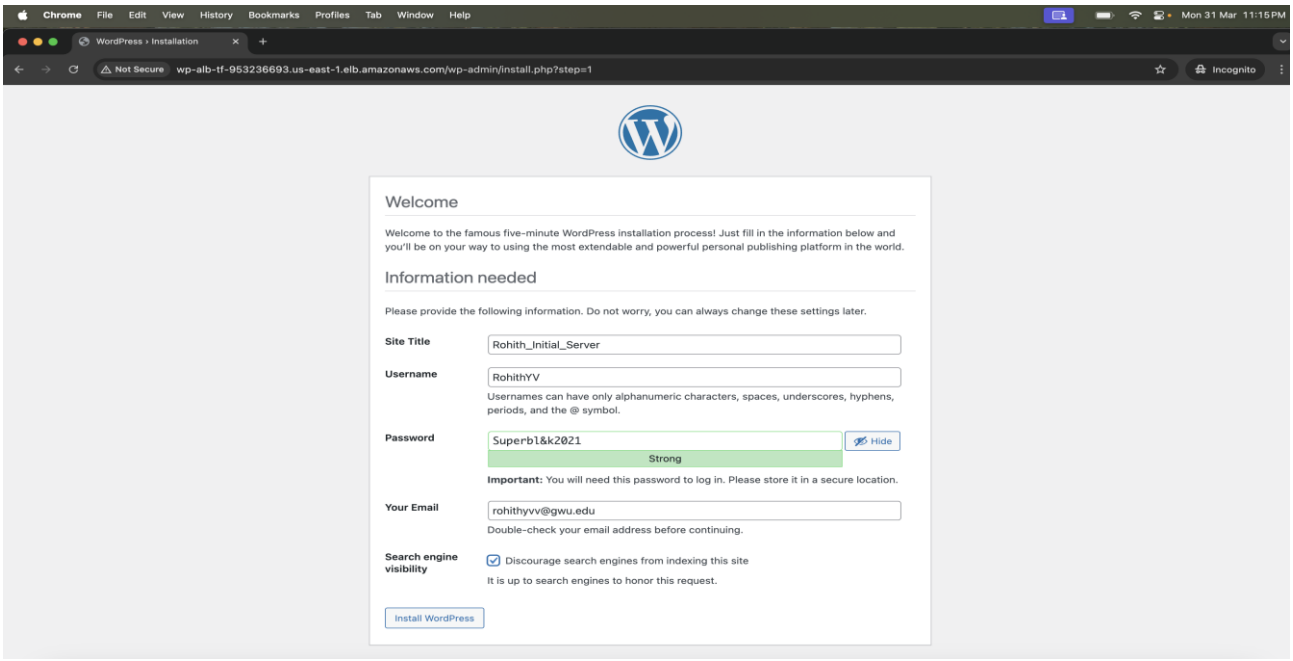
Part 3: WordPress Setup Page

Fig 3.1: WordPress installation page loaded via ALB



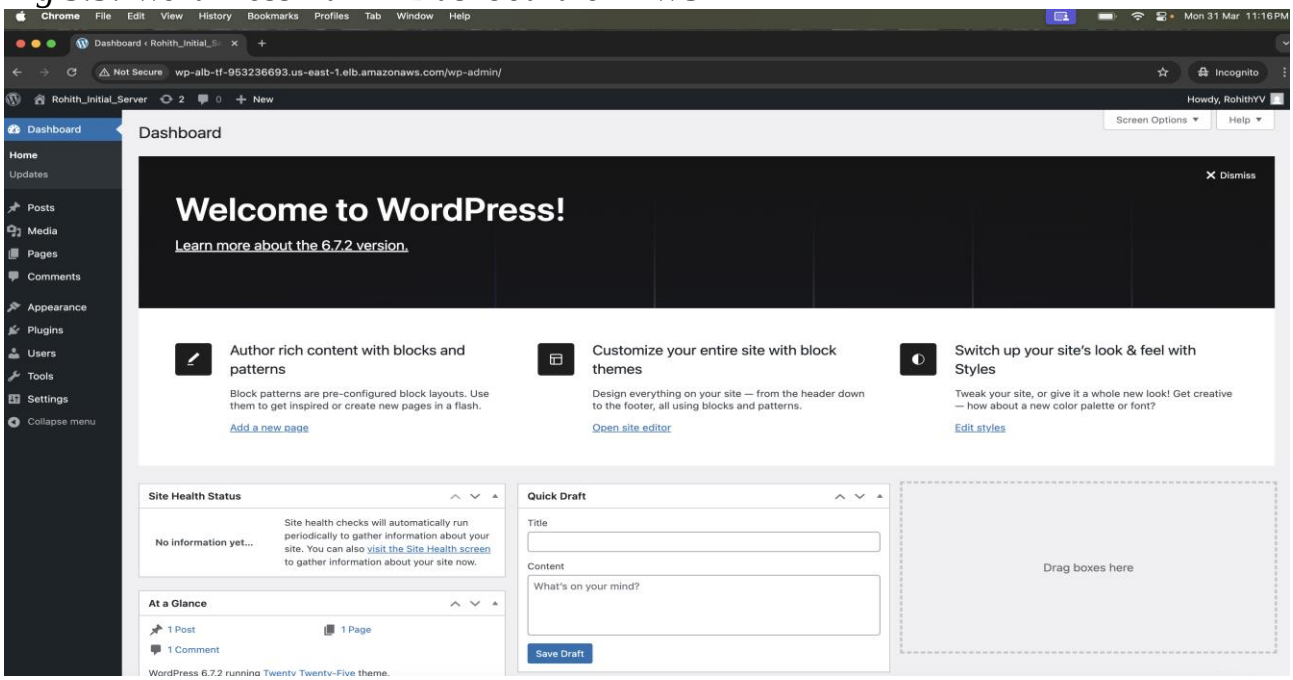
Note: This screenshot shows the default WordPress setup wizard accessed through the Application Load Balancer (ALB) URL. It confirms that the ECS Fargate task running the WordPress container is active and accessible publicly. The form prompts for site title, username, password, and email to complete the WordPress installation process.

Fig 3.2: WordPress Initial Setup Screen on AWS ECS Fargate



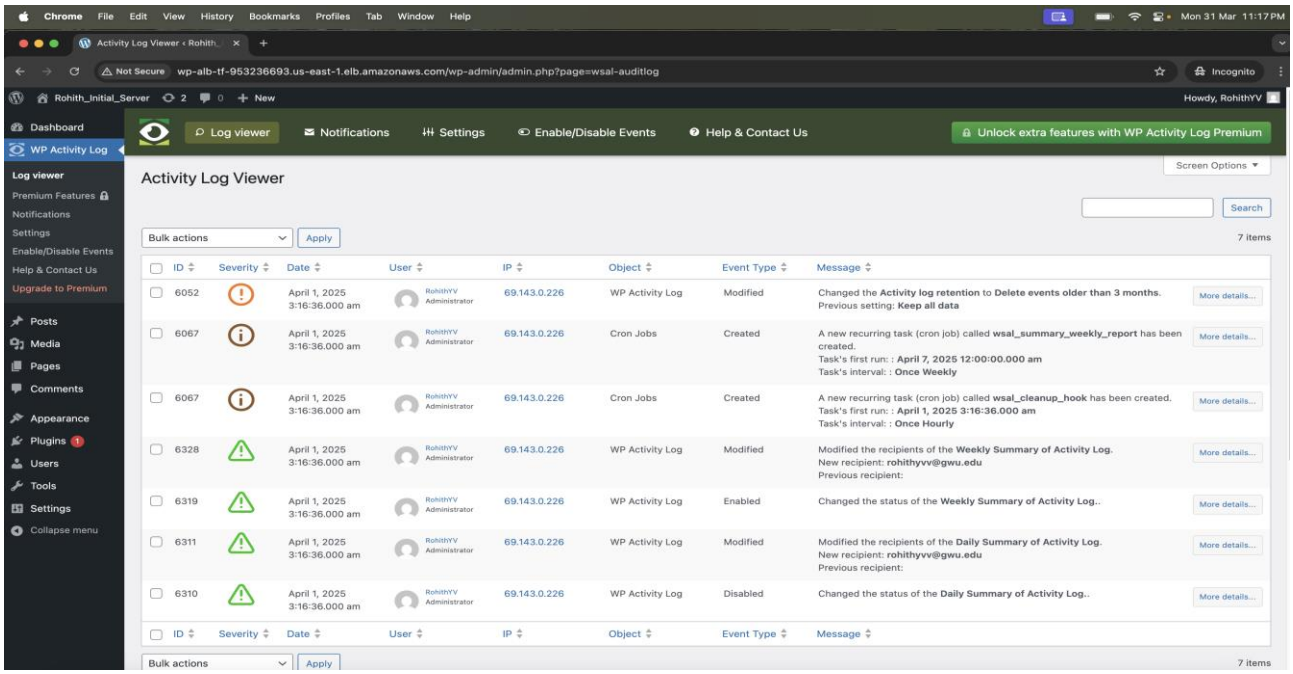
Note: This screenshot shows the successful deployment of WordPress on AWS using ECS Fargate, RDS, and Application Load Balancer. The instance is accessible through the ALB DNS, and the WordPress installation page is now live. The user is prompted to set up the site title, admin username, password, and email as part of the famous 5-minute WordPress setup process.

Fig 3.3: WordPress Admin Dashboard on AWS



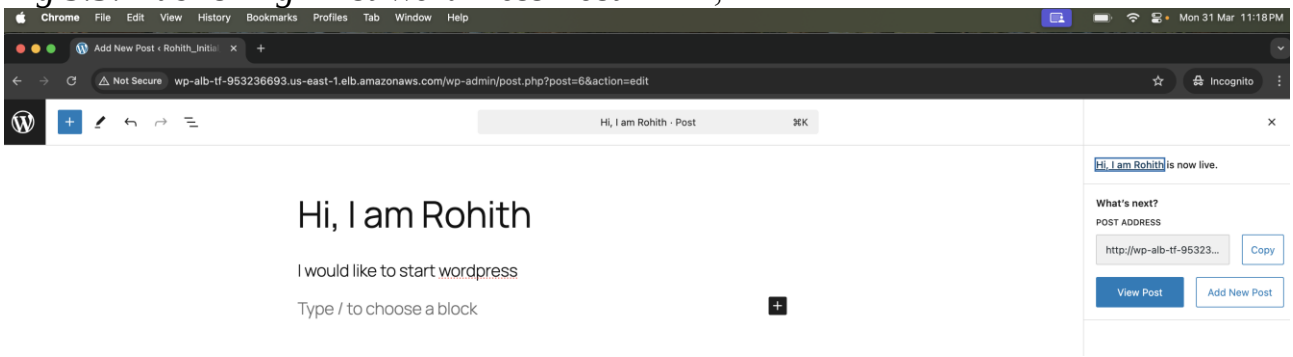
Note: *This screenshot confirms successful deployment and configuration of WordPress on AWS using ECS Fargate, RDS, and an Application Load Balancer. The admin dashboard is accessible via the ALB endpoint, and the WordPress version 6.7.2 is fully operational. The user can now manage content, pages, plugins, and themes directly from the dashboard interface.*

Fig 3.4: WordPress Activity Log Viewer on AWS Deployment



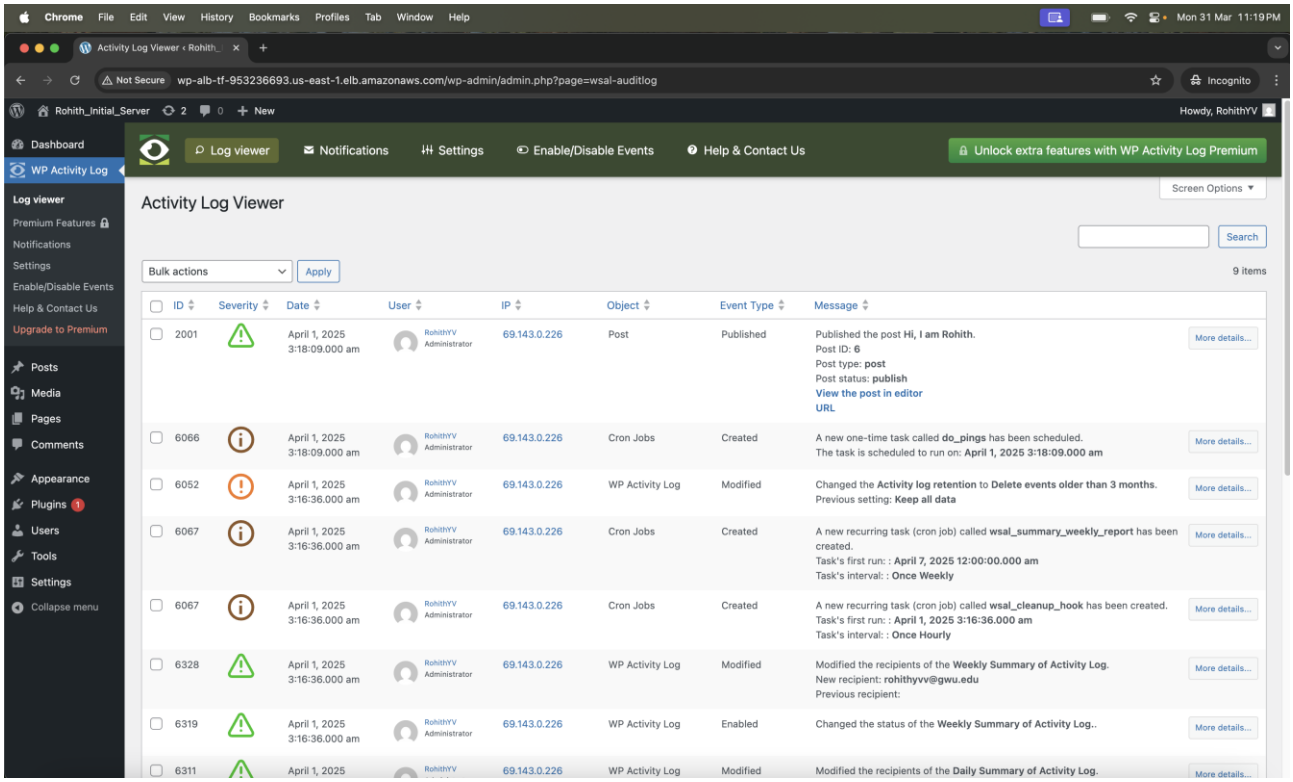
Note: *This displays the* **WP Activity Log** plugin in action within the WordPress dashboard hosted on AWS ECS. It logs key administrative activities such as cron job creation, log retention changes, and summary email updates. This visibility into backend events helps ensure operational transparency and security for WordPress deployments, especially in production environments.

Fig 3.5: Publishing First WordPress Post – "Hi, I am Rohith"



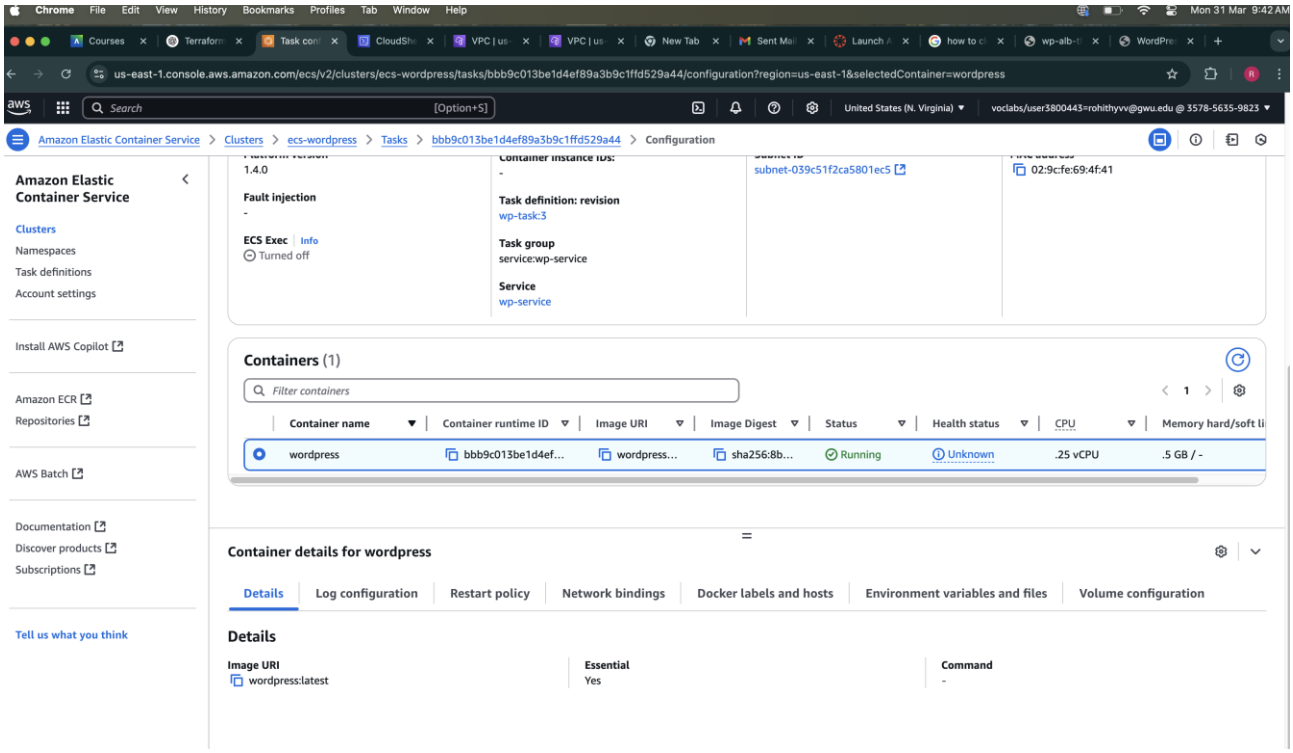
Note: This shows a newly created and published blog post using the WordPress block editor on a site deployed through AWS ECS Fargate. The post, titled *"Hi, I am Rohith"*, confirms successful content creation and publishing functionality. The post is now publicly accessible via the ALB-provided DNS, verifying that the entire WordPress pipeline — from setup to live content — is working as expected.

Fig 3.6: WordPress Activity Log Showing Published Post and System Events



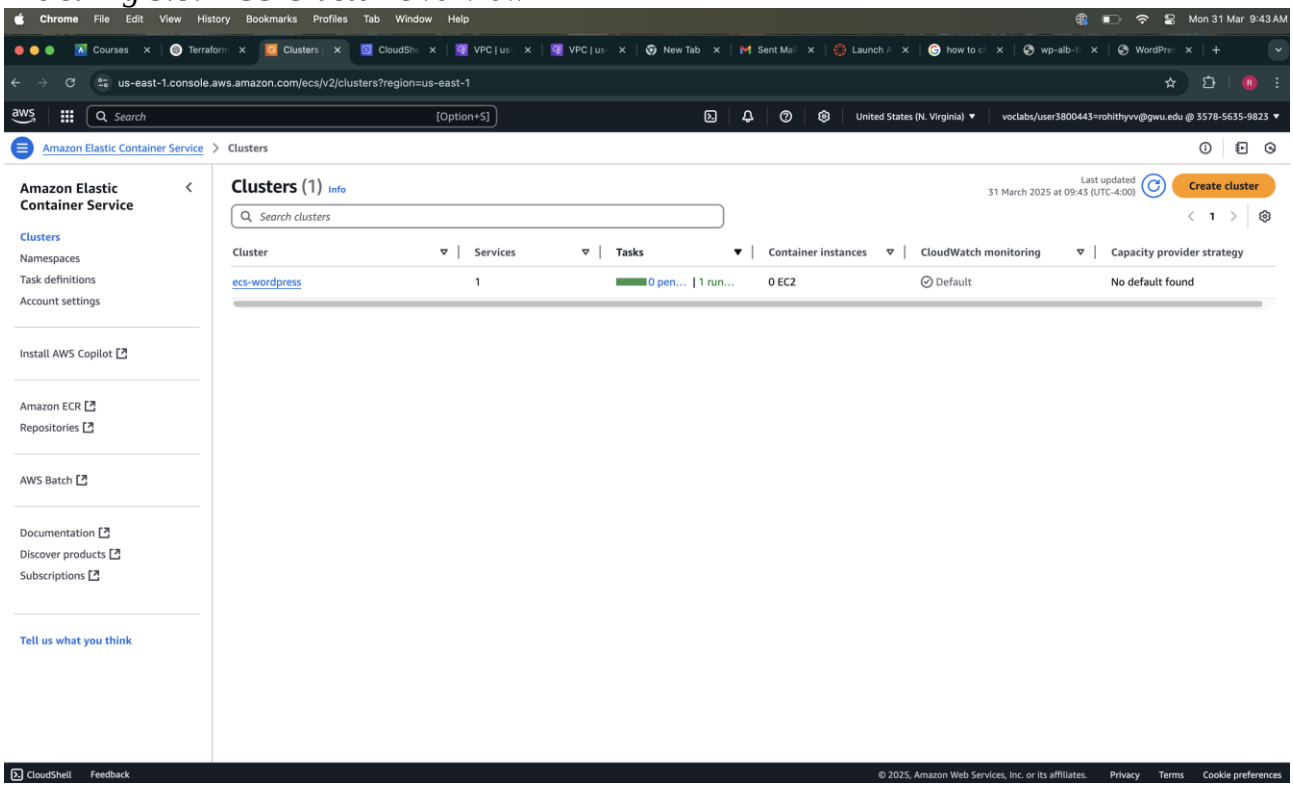
Note: This activity log captures the successful publishing of the post *"Hi, I am Rohith"* along with related system events like scheduled cron jobs and configuration updates. The WP Activity Log plugin provides a detailed audit trail of administrator actions, ensuring transparency and traceability within the WordPress site hosted on AWS ECS Fargate.

Title: *Fig 3.7: ECS Task Details*



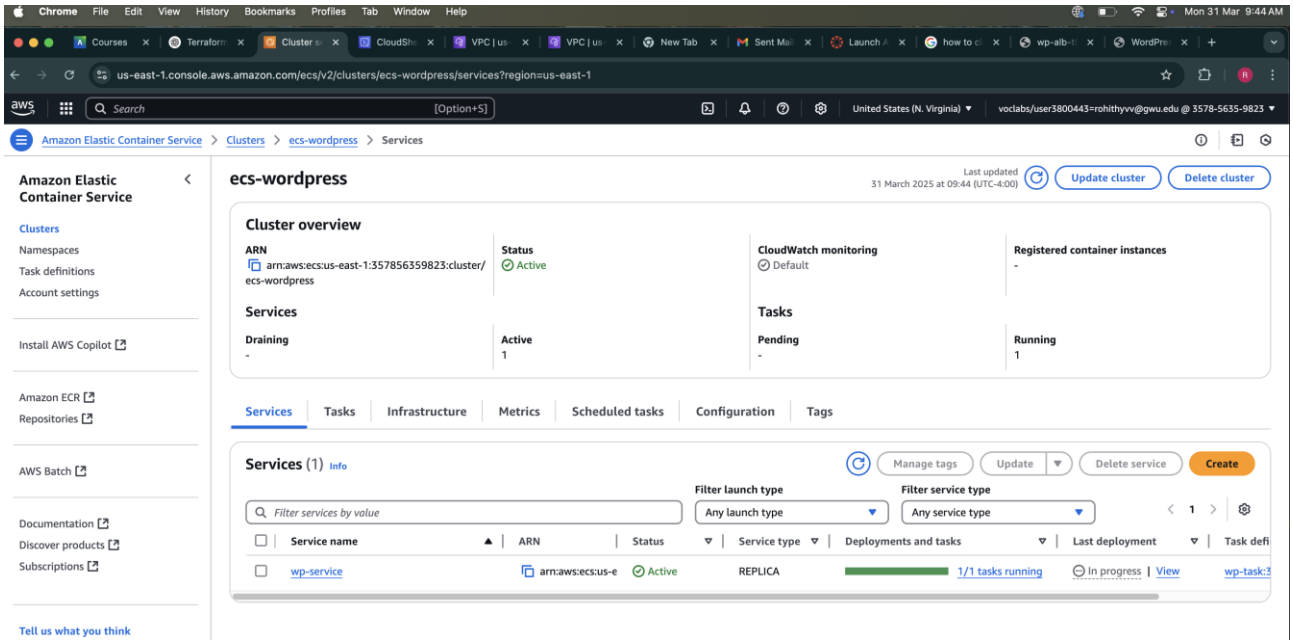
Note: The container is shown running inside ECS. The task is active, and the image wordpress:latest is deployed with essential variables.

Title: *Fig 3.8: ECS Cluster Overview*



Note: Shows the ECS cluster ecs-wordpress is active with 1 running task associated with wp-service.

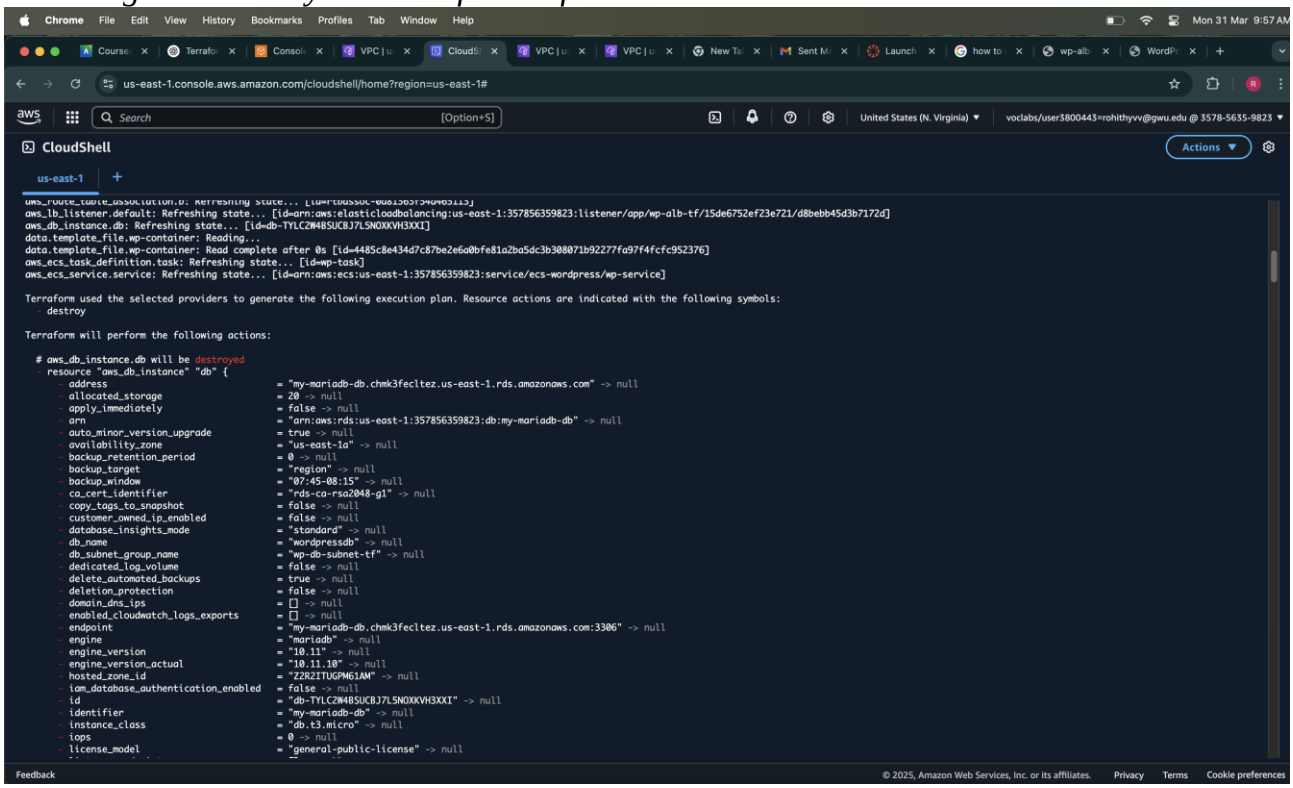
Title: *Fig 3.9: VPC and Subnet Structure*



Note: Depicts the custom VPC wp-pvc-tf, with 3 public subnets and proper route tables, matching the Terraform network setup.

Part – 4: Teardown Using Terraform Destroy

Title: *Fig 4.1: Destroy the Terraform Infrastructure*



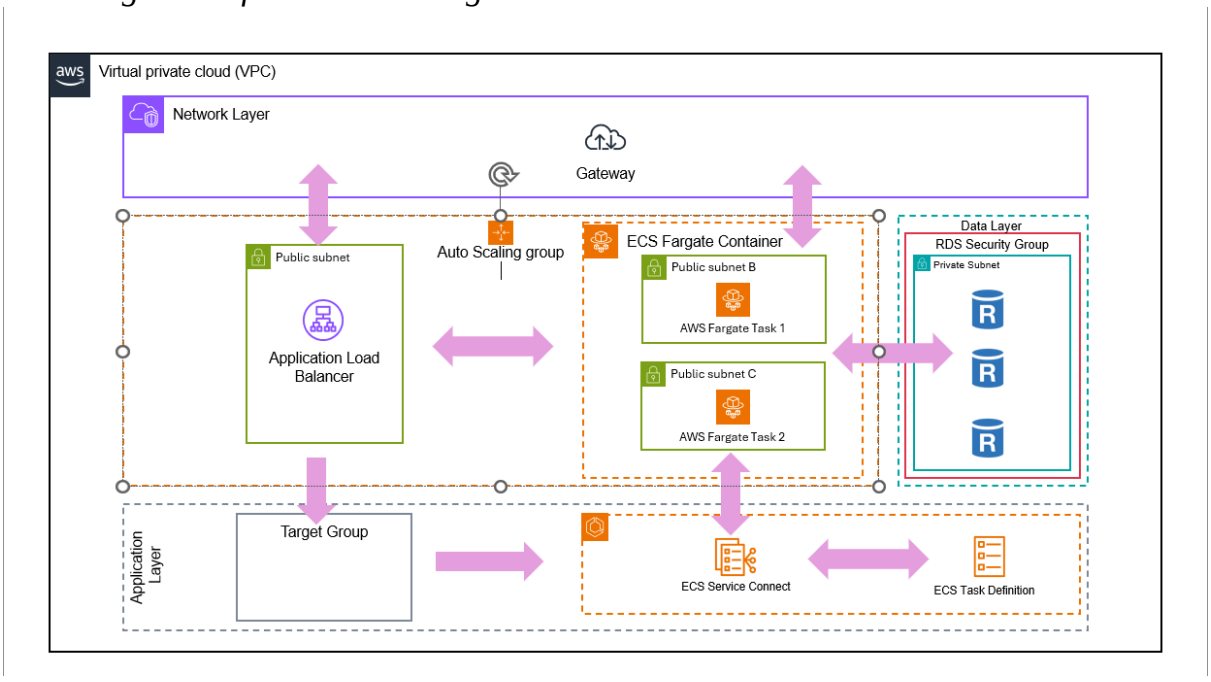
Note: After testing, terraform destroy is executed to tear down all resources to avoid charges. The log shows successful destruction of RDS, subnets, VPC, ALB, etc.

Key Notes:
Once testing is completed, use the terraform destroy command to remove all AWS infrastructure components created during the deployment.

This avoids unnecessary AWS billing and ensures clean removal of all deployed services like ECS tasks, subnets, VPCs, and databases.

Part – 5: Infrastructure Diagram

Title: *Fig 5.1: Infrastructure Diagram*



Key Notes:

This architecture diagram represents a cloud-based application deployed on **AWS** using **ECS Fargate** for containerized workloads. It features an **Application Load Balancer (ALB)** distributing traffic across ECS tasks running in different public subnets, ensuring scalability and fault tolerance.

The **data layer** consists of an **RDS database** in a private subnet, protected by a security group, ensuring secure database access.