Deploying a WordPress Site on AWS Using Terraform – HW2

Yashika LNU
The George Washington University

Database Systems - II
R. Fernandez
March 31, 2025

Author Note

Correspondence concerning this article should be addressed to Yashika LNU,
Department of
Computer Science and Engineering, The George Washington University.

Contact: yashikal@gwu.edu

Problem Statement Document

Background:
Infrastructure as Code (IaC) enables developers and system administrators to automate the provisioning and management of cloud resources using configuration files. Terraform, an open-source IaC tool by HashiCorp, allows users to define and deploy cloud infrastructure across various providers such as AWS. This assignment leverages Terraform to automate the deployment of a WordPress website hosted on AWS.

Objective:
The objective of this assignment is to gain hands-on experience with Terraform and AWS by automating the setup of a WordPress site. Students will learn to configure infrastructure components like VPCs, EC2 instances, security groups, and Application Load Balancers using Terraform scripts.

Tasks:

1. **Download and Prepare Terraform Files:** Retrieve and unzip the pre-configured Terraform files provided.
2. **Configure with AWS Account:** Modify the configuration files to include your AWS Account ID.
3. **Upload to AWS CloudShell:** Upload the updated files to AWS CloudShell for deployment.
4. **Set Up Environment:** Install Terraform in CloudShell and prepare the working directory.
5. **Initialize and Plan Deployment:** Run Terraform commands to initialize and preview infrastructure changes.
6. **Apply Terraform Plan:** Deploy the infrastructure using Terraform and verify successful resource creation.
7. **Access WordPress Site:** Use the Application Load Balancer DNS to access the hosted WordPress site.
8. **Clean-Up:** Destroy the deployed resources using Terraform to prevent unnecessary AWS charges.

Below are the exact steps followed to achieve the same.

Step 1: Download and Set Up Your Terraform Files

Get the ZIP File: Log in to Blackboard and download the terraform_files.zip file available under HWK_2.

Extract the Contents: Unzip the downloaded file to access the terraform_files directory.

Step 2: Customize Terraform Files with Your AWS Account ID

Modify the Configuration: Navigate to the terraform_files folder and open the ecs.tf file using your preferred text editor.

Insert Your Account ID: Locate the placeholder for execution_role_arn and replace it with your actual AWS Account ID. (Ensure there are no hyphens "-" in the ID.)

Figure – 1
Inserting Your Account ID in ecs.tf

```
# ECS Cluster
resource "aws_ecs_cluster" "cluster" {
  name = var.ecs_cluster_name
}

# ECS Task Definition
resource "aws_ecs_task_definition" "task" {
  family                   = "wp-task"
  container_definitions    = data.template_file.wp-container.rendered
  network_mode             = "awsvpc"
  requires_compatibilities = ["FARGATE"]
  cpu                      = "256"
  memory                   = "512"
  execution_role_arn       = "arn:aws:iam::300400931083:role/LabRole"  # Use the existing role ARN
}

# ECS Service
resource "aws_ecs_service" "service" {
  name            = "wp-service"
  cluster         = aws_ecs_cluster.cluster.id
  task_definition = aws_ecs_task_definition.task.arn
  desired_count   = 1
  launch_type     = "FARGATE"

  network_configuration {
    subnets         = [aws_subnet.wp-public-a-tf.id, aws_subnet.wp-public-b-tf.id, aws_subnet.wp-public-c-tf.id]
    security_groups = [aws_security_group.wp-alb-tf.id]
    assign_public_ip = true
  }

  load_balancer {
    target_group_arn = aws_lb_target_group.default.arn
    container_name   = "wordpress"
    container_port   = 80
  }

  depends_on = [aws_lb_listener.default]
}
```

Note: Replace the placeholder under execution_role_arn with your actual AWS Account ID. Make sure to remove any hyphens (-) from the ID before inserting it. This step ensures that the ECS task execution role is correctly linked to your AWS account.

Recompress the Folder: After saving your changes, re-zip the updated terraform_files folder into a new ZIP file.

Step 3: Upload Files to AWS CloudShell

Open CloudShell: On the AWS homepage, click the CloudShell icon located at the bottom left to launch the terminal.

Upload the ZIP File: In CloudShell, click on Actions in the top-right corner and select Upload file. Choose the ZIP file you created with the updated Terraform configurations.

Confirm the Upload: Run the command ls to ensure the ZIP file is now visible in your CloudShell working directory.

Step 4: Set Up Your CloudShell Environment

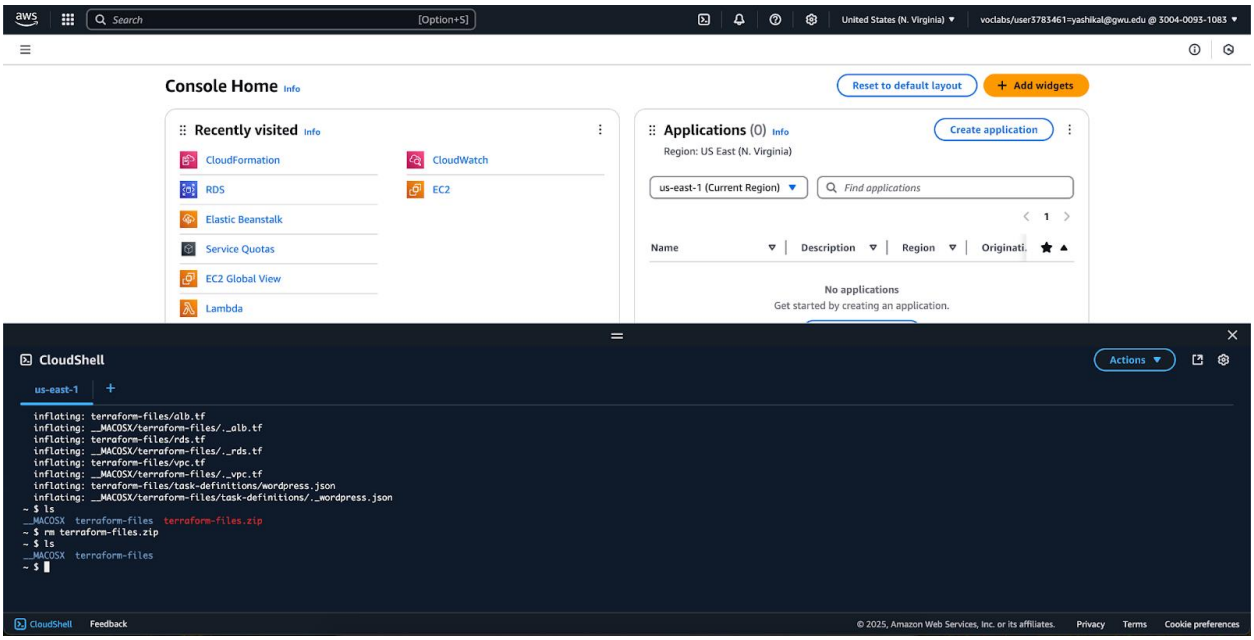
Extract the Contents: Run the following command to unzip the files:

```
unzip terraform_files.zip
```

Delete the ZIP File: Clean up by removing the ZIP file:

```
rm terraform_files.zip
```

Figure – 2
Deleting the ZIP File



Note: After unzipping the terraform_files.zip, it's good practice to delete the ZIP file using the rm command to keep your CloudShell workspace clean and organized.

Navigate to the Folder: Move into the unzipped directory:

```
cd terraform_files
```

Step 5: Install Terraform in CloudShell

Download the Terraform Binary: Use the following command to download Terraform version 1.6.3:

```
wget
https://releases.hashicorp.com/terraform/1.6.3/terraform_1.6.3_linux_amd64.zip
```

Extract the Binary: Unzip the downloaded file:

```
unzip terraform_1.6.3_linux_amd64.zip
```

Remove the ZIP File: Clean up by deleting the ZIP archive:s

```
rm terraform_1.6.3_linux_amd64.zip
```

Add Terraform to Your PATH: Move the Terraform executable to a system-wide location so it can be accessed globally:

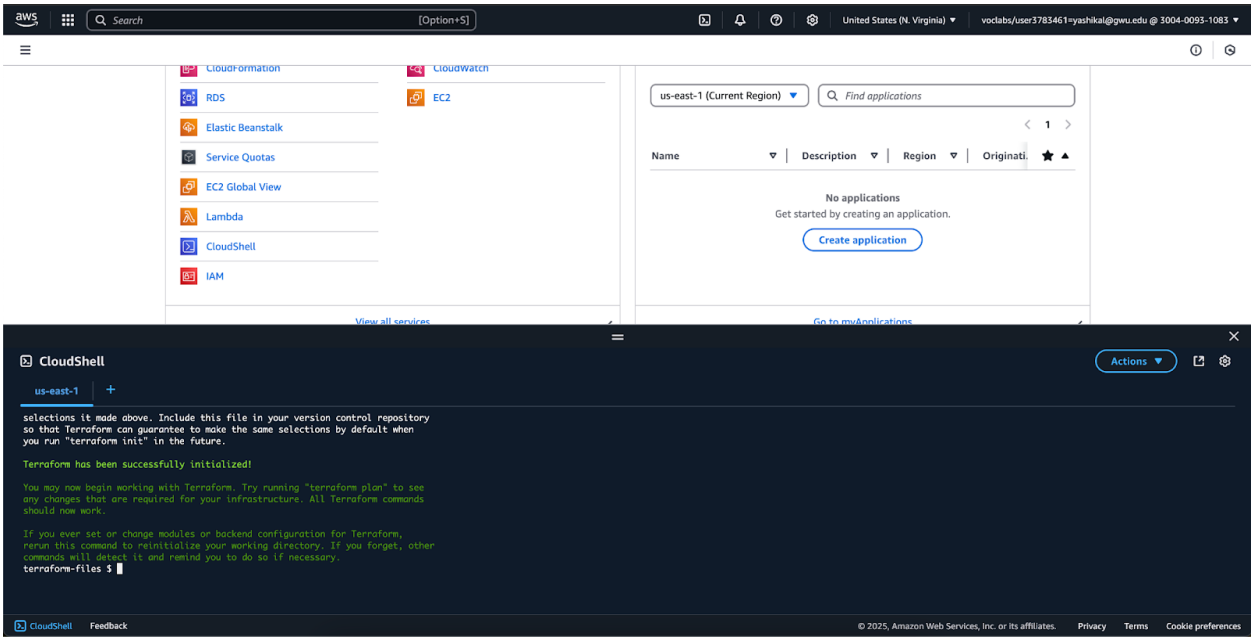
```
sudo mv terraform /usr/local/bin
```

Step 6: Initialize and Preview Your Terraform Deployment

Initialize Terraform: While inside the terraform_files directory, run:

```
terraform init
```

Figure – 3
Terraform successfully initialized



Note: A successful initialization message confirms that Terraform is ready to deploy your infrastructure.

Generate a Plan: Create an execution plan with the following command:

```
terraform plan -out=plan.tfplan
```

Confirm the Plan File: List the directory contents to ensure plan.tfplan was created:

```
ls
```

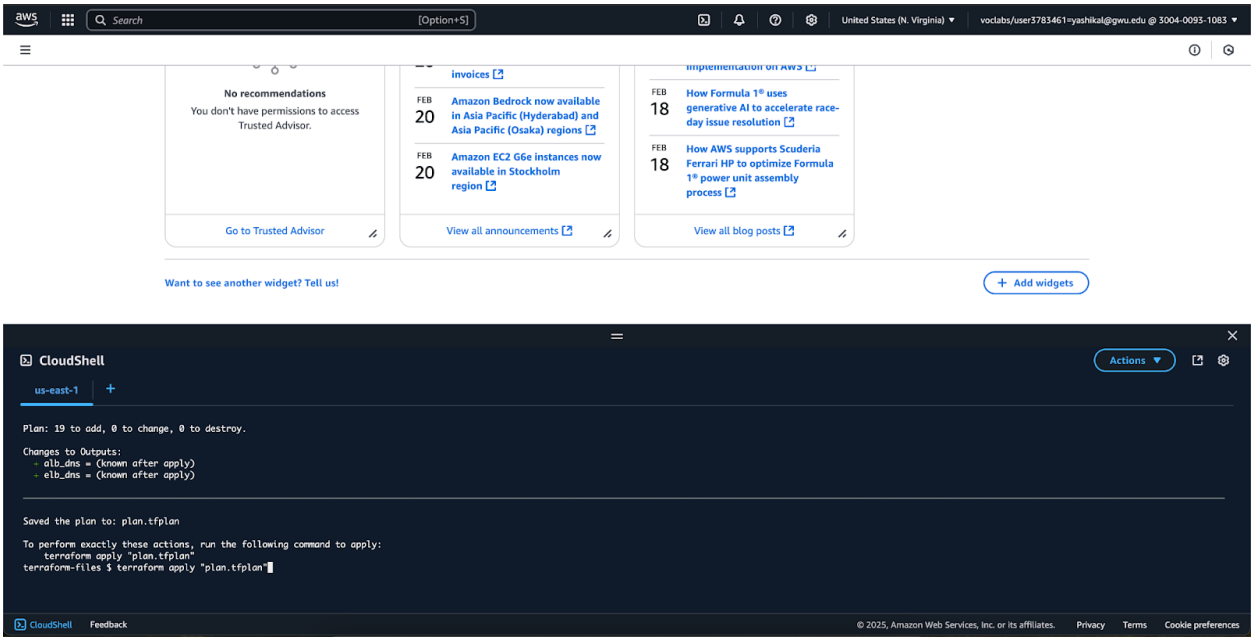
Step 7: Deploy Your AWS Infrastructure

Run the Terraform Plan: Execute the command below to launch your custom VPC and associated resources:

```
terraform apply "plan.tfplan"
```

Figure - 4

Command used to apply the Terraform plan.

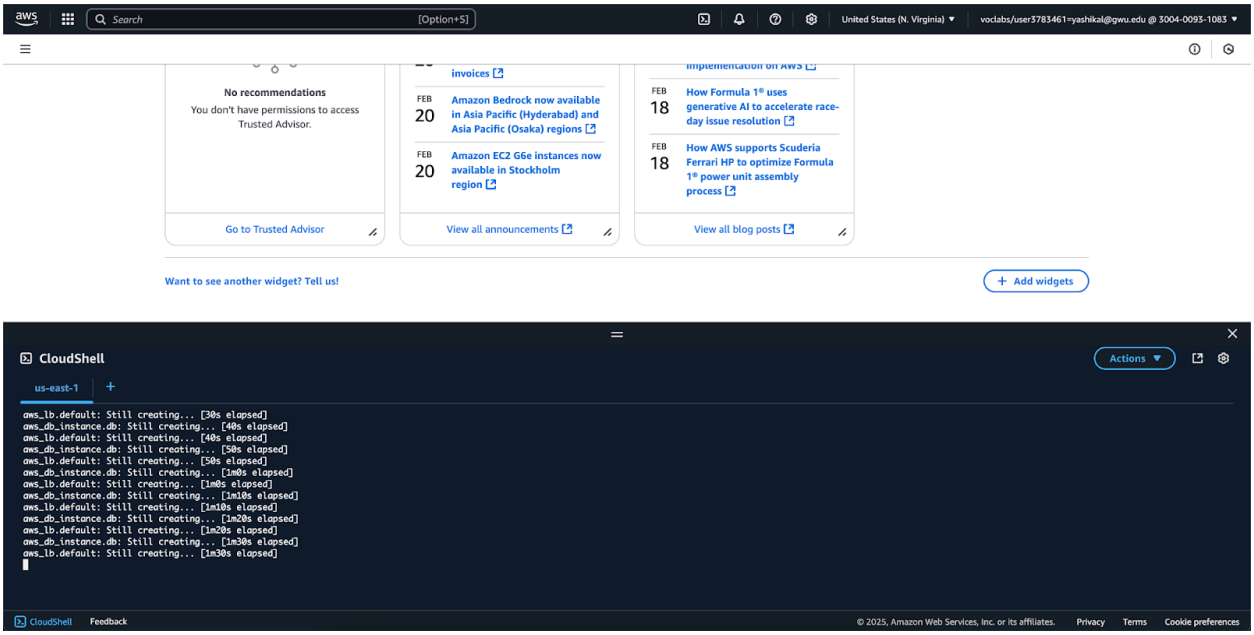


Note: This figure shows the command terraform apply "plan.tfplan" being executed. This step uses the previously generated plan to provision the AWS resources defined in your Terraform configuration.

Monitor the Deployment: The process may take a few minutes. Once finished, Terraform will display the output values confirming successful deployment.

Figure – 5

Terraform apply process running – provisioning AWS infrastructure.



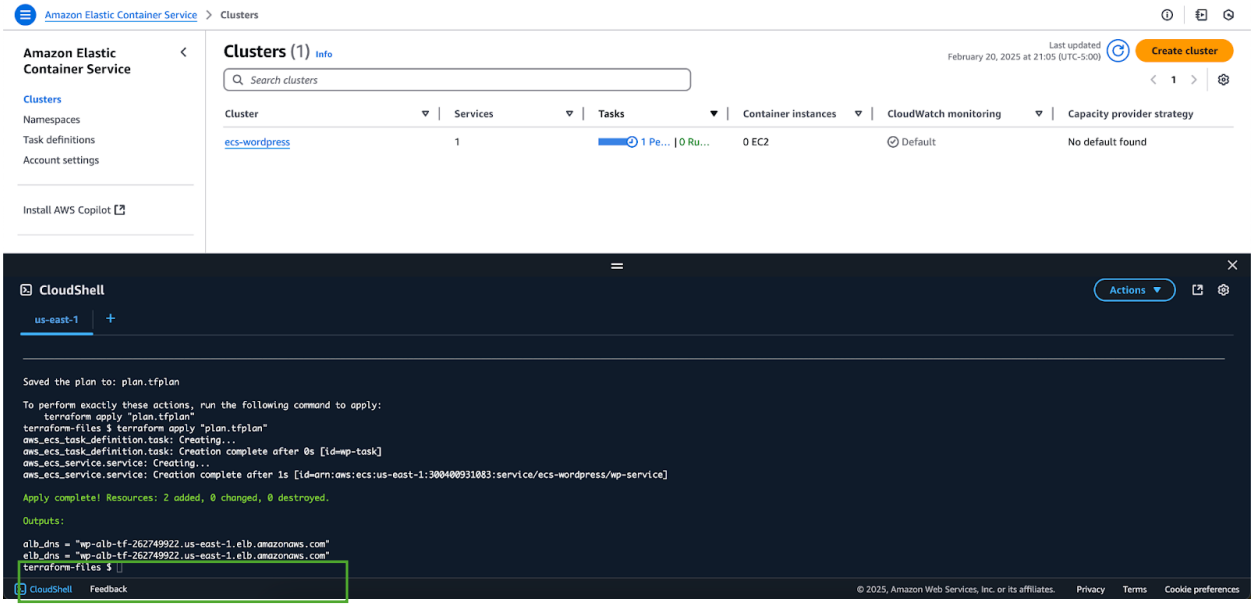
Note: This figure displays the Terraform apply process in progress. Terraform is now creating the infrastructure based on your configuration, including networking, compute resources, and load balancers.

Step 8: Access Your Live WordPress Site

Find the ALB DNS: Once the Terraform apply process is complete, look for the alb_dns value in the output. This is the public DNS of your Application Load Balancer.

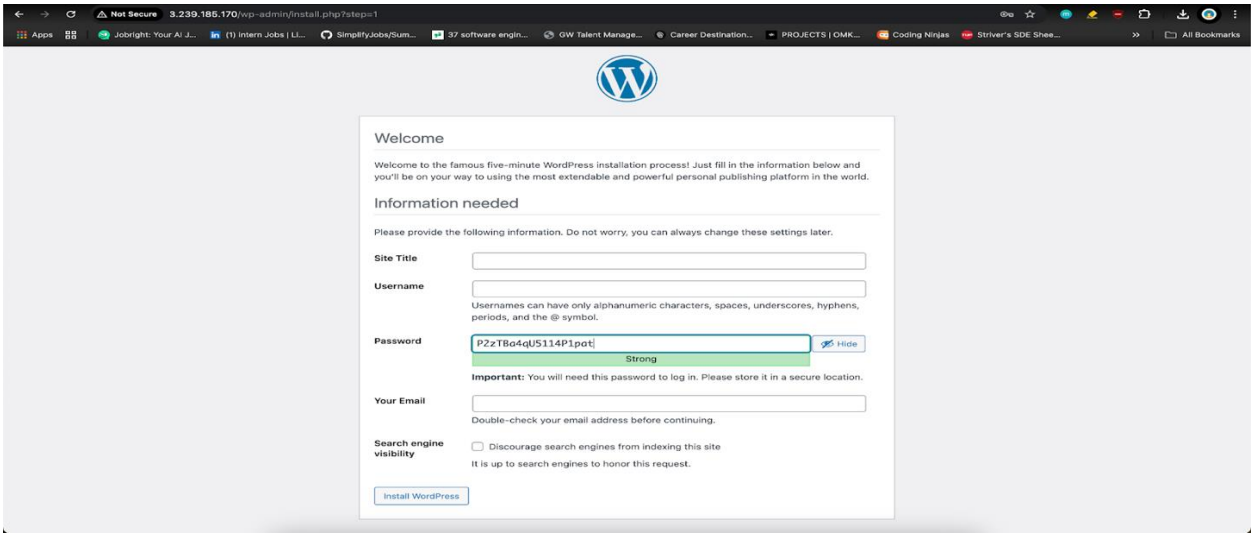
Visit the Website: Copy the alb_dns link and paste it into a new browser tab to open your WordPress site.Note: It might take a few minutes for the site to load completely and become fully functional.

Figure – 6
Terraform apply completed successfully with alb_dns displayed in the output.



Note: It may take a couple of minutes for the WordPress site to become fully available. If the page doesn't load immediately, wait a moment and try refreshing.

Figure – 7
Successfully loaded WordPress site in the browser using the alb_dns link.



Note. This screenshot confirms that the WordPress deployment on AWS was successful and the site is accessible via the Application Load Balancer's DNS.

Step 9: Tear Down the Environment (Clean-Up)

Remove Deployed Resources: To delete all the infrastructure you've created, run:

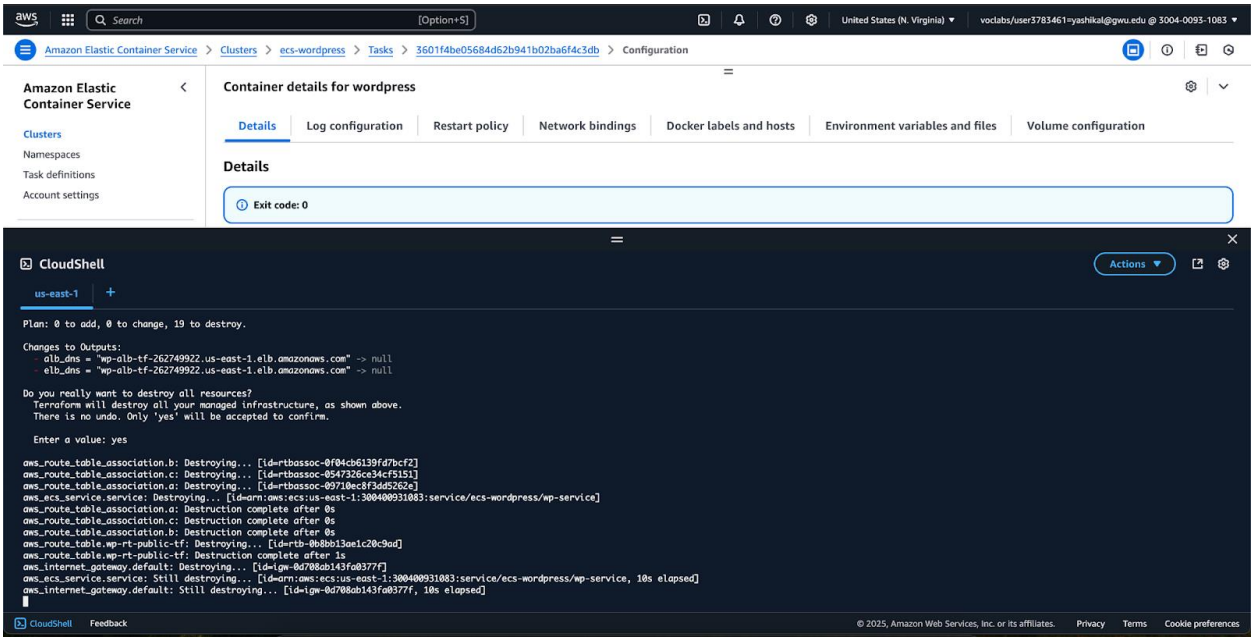
```
terraform destroy
```

Confirm the Action: When prompted, type yes to proceed with the destruction.

Let the Process Finish: Terraform will take a few moments to remove all associated resources and clean up the environment.

Figure – 8

Terraform destroying deployed infrastructure and cleaning up resources.



Note: This step ensures that no unnecessary resources remain active, helping to avoid unexpected AWS charges. Always verify that the environment has been fully destroyed before closing CloudShell.

References

AWS CloudShell Documentation
https://docs.aws.amazon.com/cloudshell/
Guide to using AWS CloudShell for deploying and managing resources from the browser.

Deploying WordPress on AWS with Terraform
https://learn.hashicorp.com/tutorials/terraform/aws-build
Step-by-step tutorials on deploying web applications using Terraform on AWS.

AWS Fargate Overview
https://aws.amazon.com/fargate/
Overview and benefits of AWS Fargate for serverless container deployments.

AWS VPC Documentation
https://docs.aws.amazon.com/vpc/
Learn about VPC components like subnets, route tables, internet gateways, and security groups.