# 

****

# 

**Cloud-Based Web Hosting with EC2 and EBS**

**using Terraform**

**Capstone Project – Part 1**

Design a Cloud-Based Application (e.g., Web App with DB)

Plan Compute, Storage, and Networking

Start Implementation with IaC

Hands-On: Deploy Core Infrastructure

**Submission Details**

Submitted by: Shivansh Mishra

Submitted to: RamyaSri

Date: July 16, 2025

# 

# 

**Table of Contents**

[**Project Objective 3**](#_ixhyc62iod9g)

[**Prerequisites 3**](#_cq3d4oxdkzfl)

[**Architecture Overview 3**](#_e9qz9yl5b8mt)

[**Technical Specifications 4**](#_d2mjj7a8yxv0)

[**Implementation Overview 4**](#_ij9dl36rsnah)

[**Detailed Implementation 5**](#_h5t52qjbi2eg)

[**Conclusion 1**](#_444ofh8dtcp6)**3**

# Project Objective

The primary objective of Part 1 of capstone is to design and deploy core cloud infrastructure components for hosting a static portfolio web page using AWS EC2 and EBS, fully automated via Terraform. This project reflects the practical application of **cloud computing and Infrastructure as Code (IaC)** concepts learned during the first 22 days of the program, with a focus on:

1. **Infrastructure Planning**: Design a cloud-native architecture by creating a Virtual Private Cloud (VPC) with subnetting, routing, and internet access using Terraform.
2. **Compute Services**: Provision an **EC2 instance (Ubuntu)** using Terraform to act as a web server for hosting the portfolio application.
3. **Storage Solutions**: Attach a 20 GiB **Elastic Block Store (EBS)** volume to the EC2 instance, format and mount it for future data or logs.
4. **Web Server Deployment**: Automatically install and configure **Apache** web server using a shell script (user\_data) to serve a static HTML portfolio page.
5. **Security Implementation**: Use AWS **Security Groups** to allow controlled access to the EC2 instance via **SSH (port 22)** and serve the portfolio through **HTTP (port 80)**.
6. **Version Control**: Manage and version the infrastructure using **Terraform files** stored and maintained in a **GitHub repository**, enabling reproducibility and collaboration.

# Prerequisites

* **AWS Account** with appropriate IAM permissions to create VPC, EC2, EBS, Security Groups, and related resources
* **GitHub Account** for version control of Terraform and shell script files
* **Terraform Installed** locally to write and deploy infrastructure as code
* **EC2 Key Pair** (PEM file) to SSH into your instance
* Basic Knowledge of AWS EC2, EBS, VPC, and Security Groups
* **Understanding of Apache Web Server Basics** for hosting HTML pages
* Basic understanding of HTML basics
* **Command-Line Interface Usage** (Git Bash, terminal) for interacting with Terraform and SSH

# Architecture Overview

**High-Level Design**

The phase 1 implementation creates a foundational **cloud infrastructure** architecture consisting of:

* **Frontend Layer**: Static HTML portfolio page hosted via Apache on an Amazon EC2 instance
* **Compute Layer**: Amazon EC2 instance (Ubuntu) provisioned using Terraform
* **Storage Layer**: Amazon EBS (gp3) volume attached to the EC2 instance for persistent storage
* **Automation Layer**: Terraform Infrastructure as Code automates creation, provisioning, and teardown of all components
* **Security Layer**: AWS Security Group with rules to allow SSH (22) and HTTP (80) access; IAM user credentials configured securely for Terraform

**Data Flow Design**

1. User accesses the portfolio website using the public IP of the EC2 instance over HTTP
2. Apache server running on the EC2 instance serves a static HTML page stored in /var/www/html
3. All HTML content is rendered directly from the EC2 instance's web root
4. No dynamic forms or backend processing are involved (static portfolio only)
5. Any changes to the web page require manual updates to the HTML file or automation via future CI/CD
6. Persistent storage can be leveraged using the attached EBS volume (e.g., for logs, uploads)

# Technical Specifications

**Infrastructure Components**

* VPC: Custom virtual network with CIDR 10.0.0.0/16
* Public Subnet: 10.0.1.0/24 for hosting EC2 instance
* Internet Gateway: Provides internet access to public subnet
* EC2 Instance: Ubuntu 22.04, t2.micro, runs Apache for static page
* EBS Volume: 20 GiB (gp3), mounted at /data
* Security Group: Allows SSH (22) from personal IP and HTTP (80) from anywhere
* Terraform Files: Define and manage all resources in code

**Resource Naming Convention**

* VPC: main-vpc
* Subnet: public-subnet
* EC2 Instance: web-instance
* EBS Volume: extra-ebs-volume
* Security Group: web-sg
* Terraform Key: my-terraform-key (PEM file for SSH)

# Implementation Overview

**Phase 1: Environment Setup and Configuration**

* Installed and configured Terraform
* Set up AWS credentials for programmatic access
* Wrote and structured Terraform files (main.tf, web-deploy.sh)
* Verified setup using terraform init and terraform validate

**Phase 2: Web Server Setup and Frontend Deployment**

* Created a simple static HTML portfolio page
* Used Apache server installed via EC2 user\_data script
* Placed HTML file in /var/www/html/index.html to serve content

**Phase 3: Backend Infrastructure Creation**

* Provisioned custom VPC, subnet, internet gateway, route table, and security groups
* Deployed EC2 instance and attached a 20 GiB EBS volume
* Mounted EBS at /data for persistent storage needs

**Phase 4: Infrastructure Automation**

* Used Terraform for end-to-end infrastructure provisioning
* Managed changes using terraform plan, deployed using terraform apply
* Applied best practices for idempotency and modularity

**Phase 5: Testing and Validation**

* Verified Apache service status and website availability via EC2 public IP
* Tested SSH connectivity using the .pem key
* Confirmed EBS volume mounted correctly and data is persistent across reboots

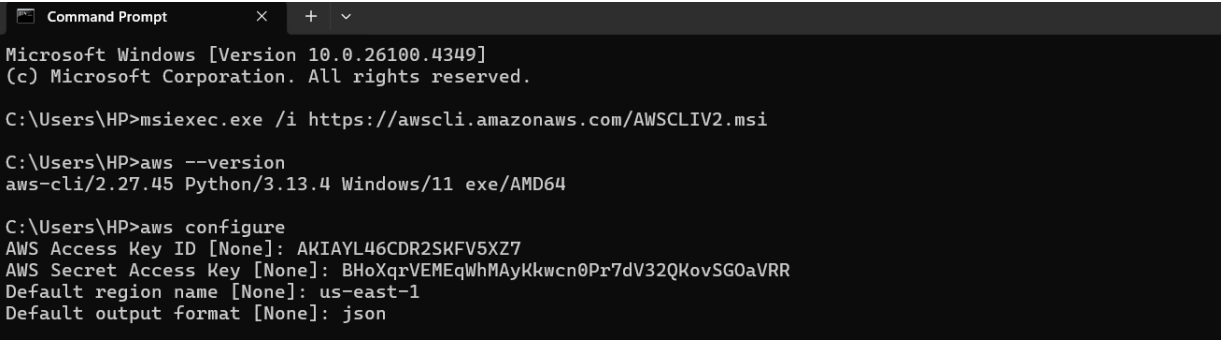
# Detailed Implementation

**1. Install Required Tools**

* **Terraform:** Installed the latest version (≥ 1.0) on the local machine.
* **AWS CLI:** Installed and configured to authenticate using IAM credentials.
* **Git:** Used for version control and syncing with GitHub.

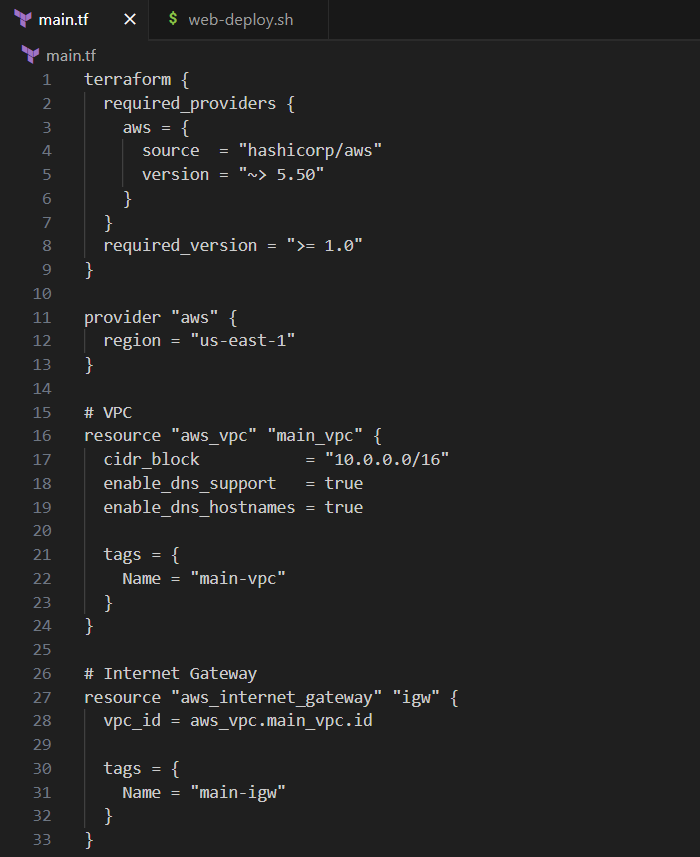
**2. AWS Access Configuration**

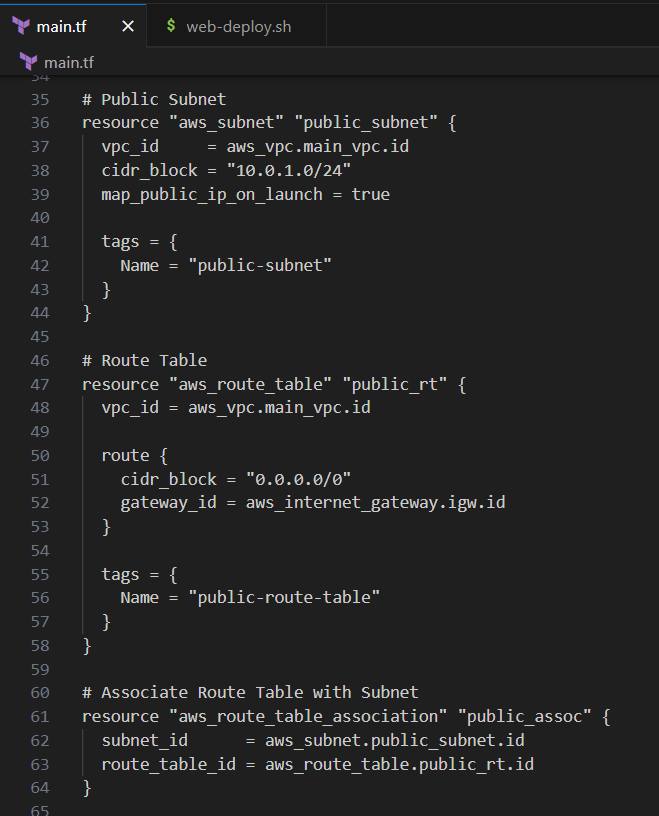
* Created or used an existing **IAM user** with programmatic access and permissions for EC2, VPC, EBS, and Security Groups.

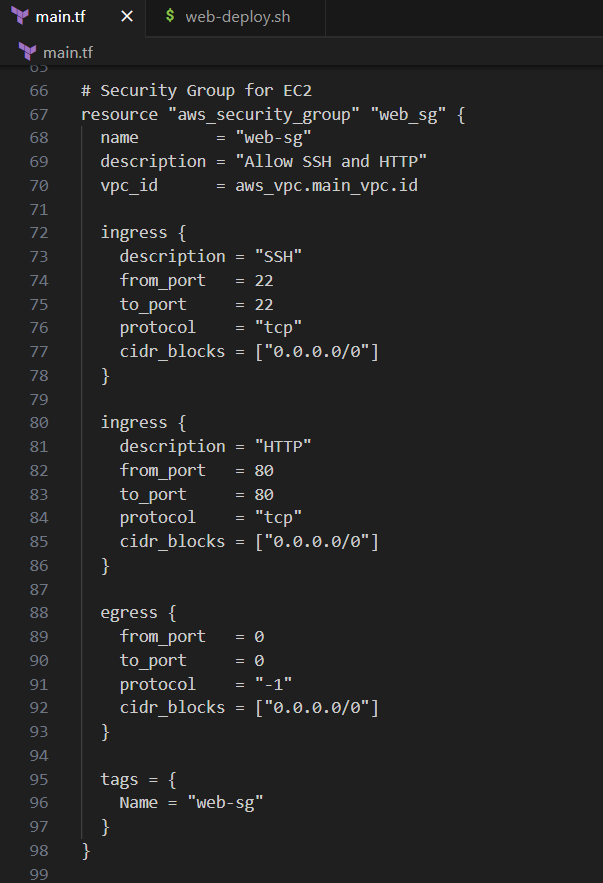


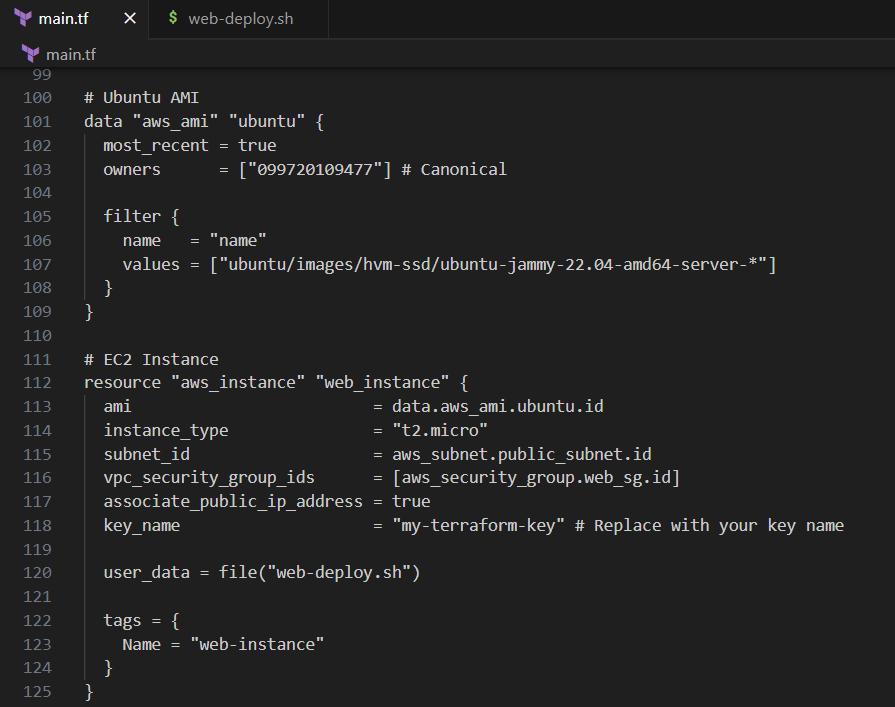
**3. Project Directory Structure**

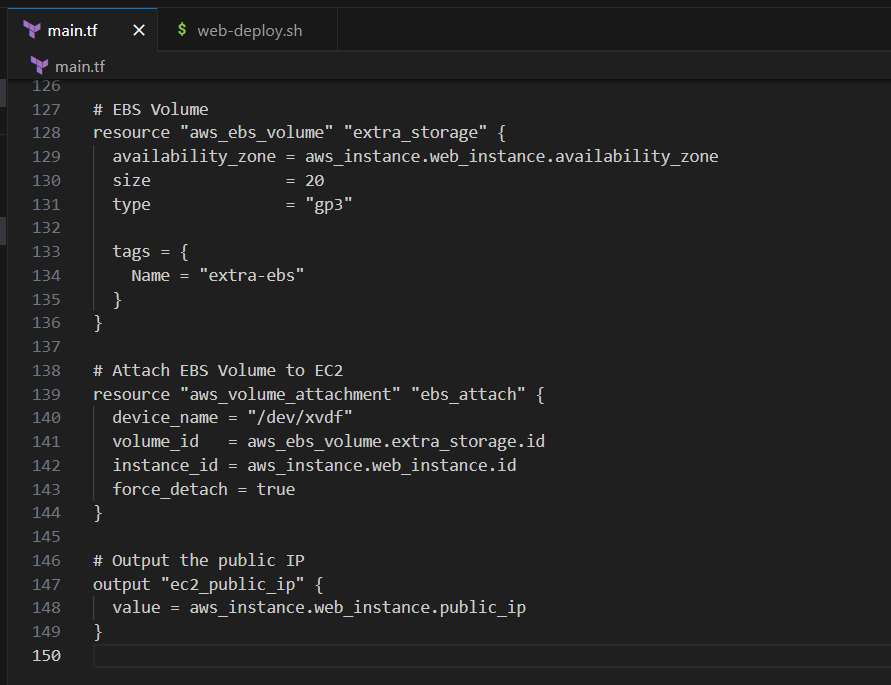
* **Main.tf**

****

****

****

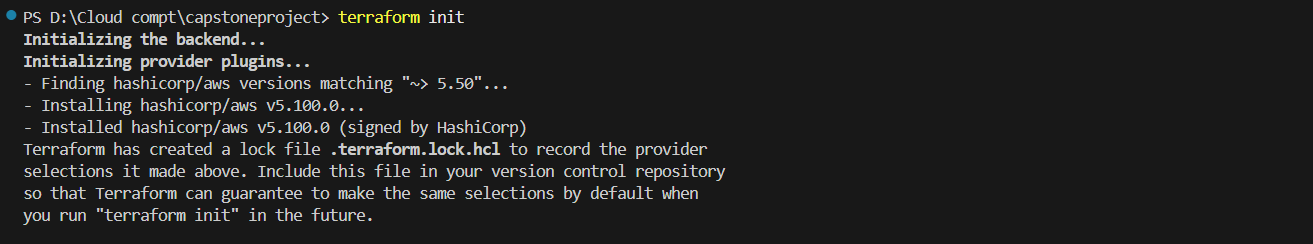
****

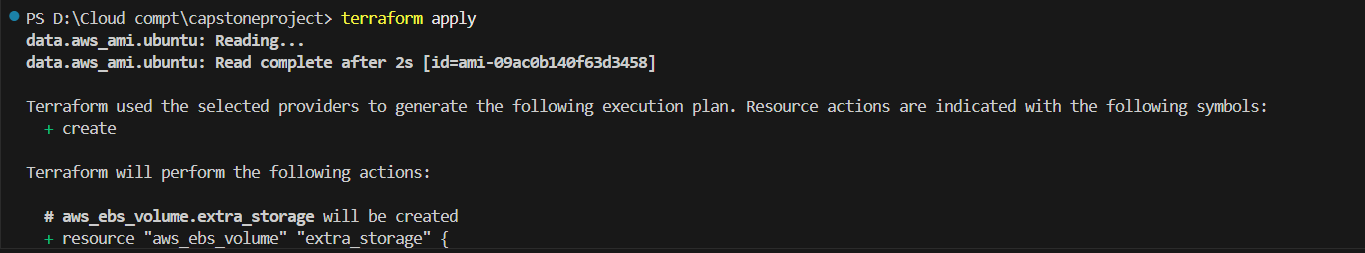
****

* **Web-deploy.sh**

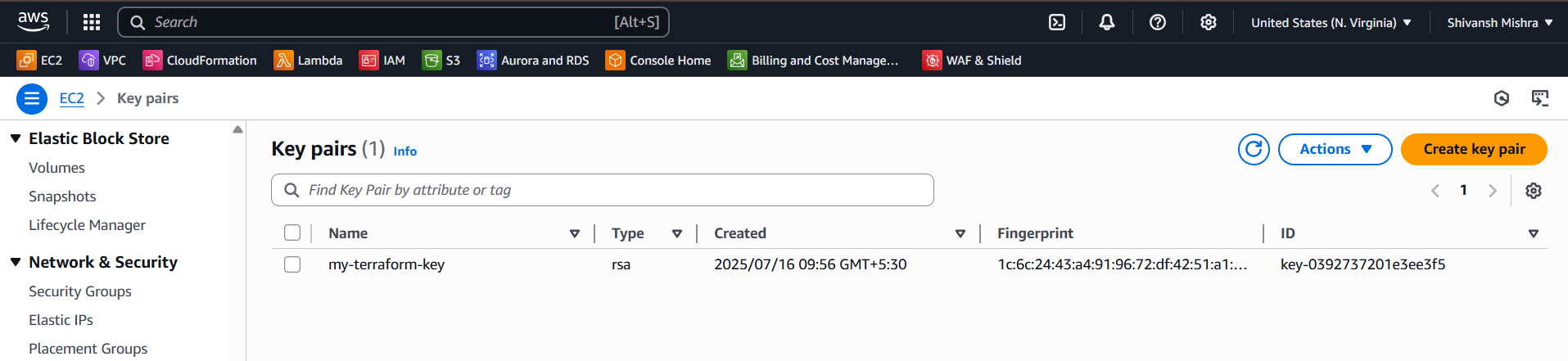
****

**4. Initialize Terraform Project**

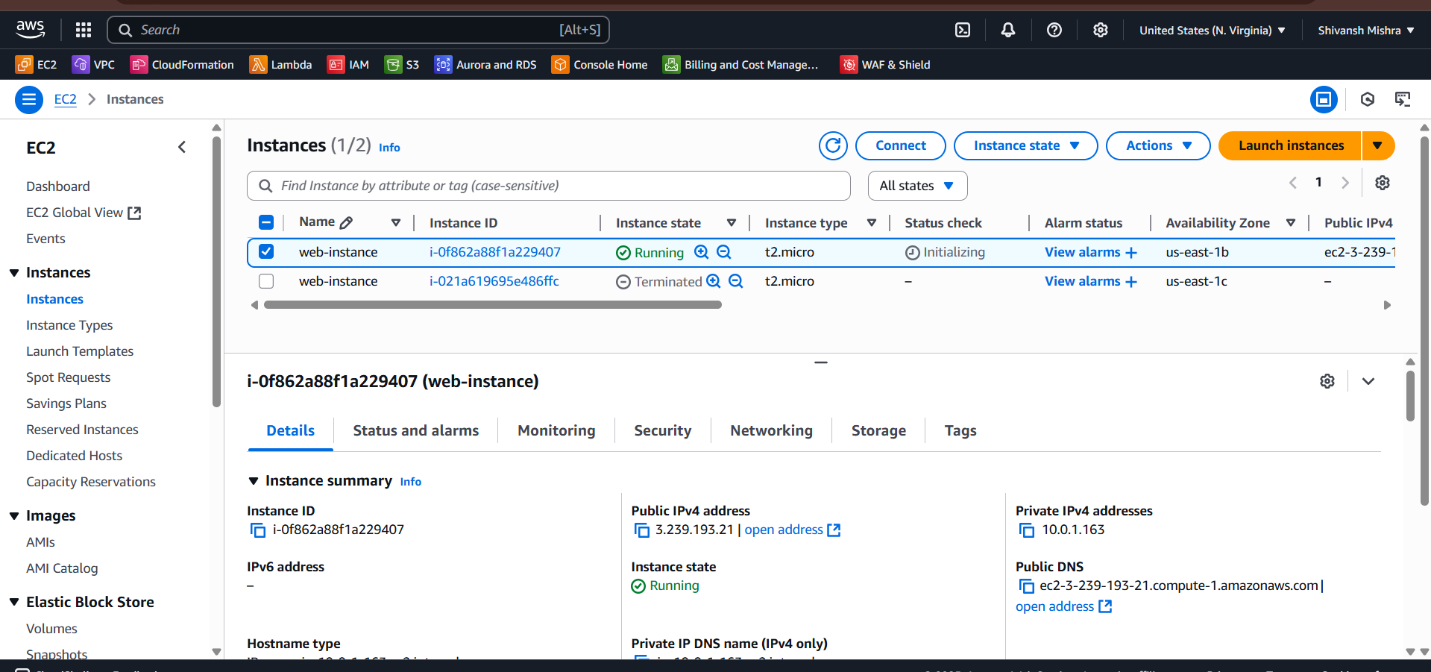
****

****

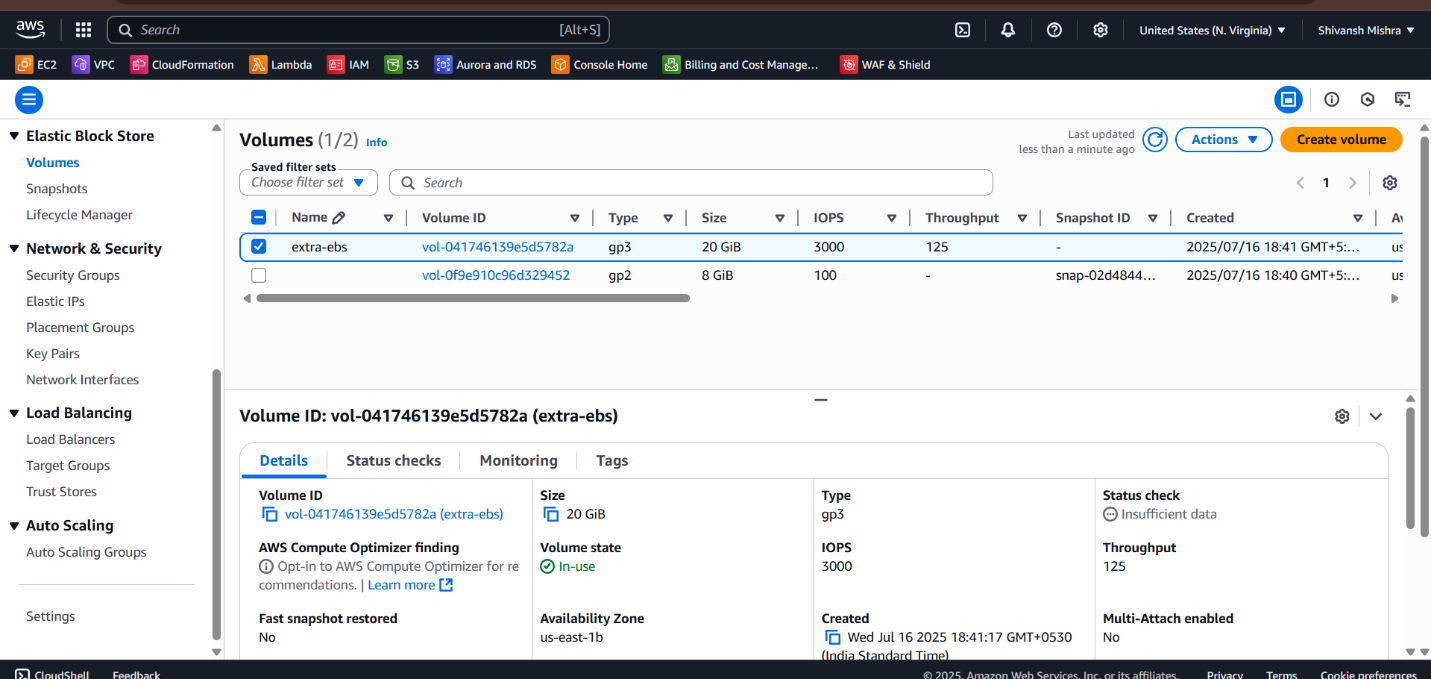
**5. Key Pair Setup**



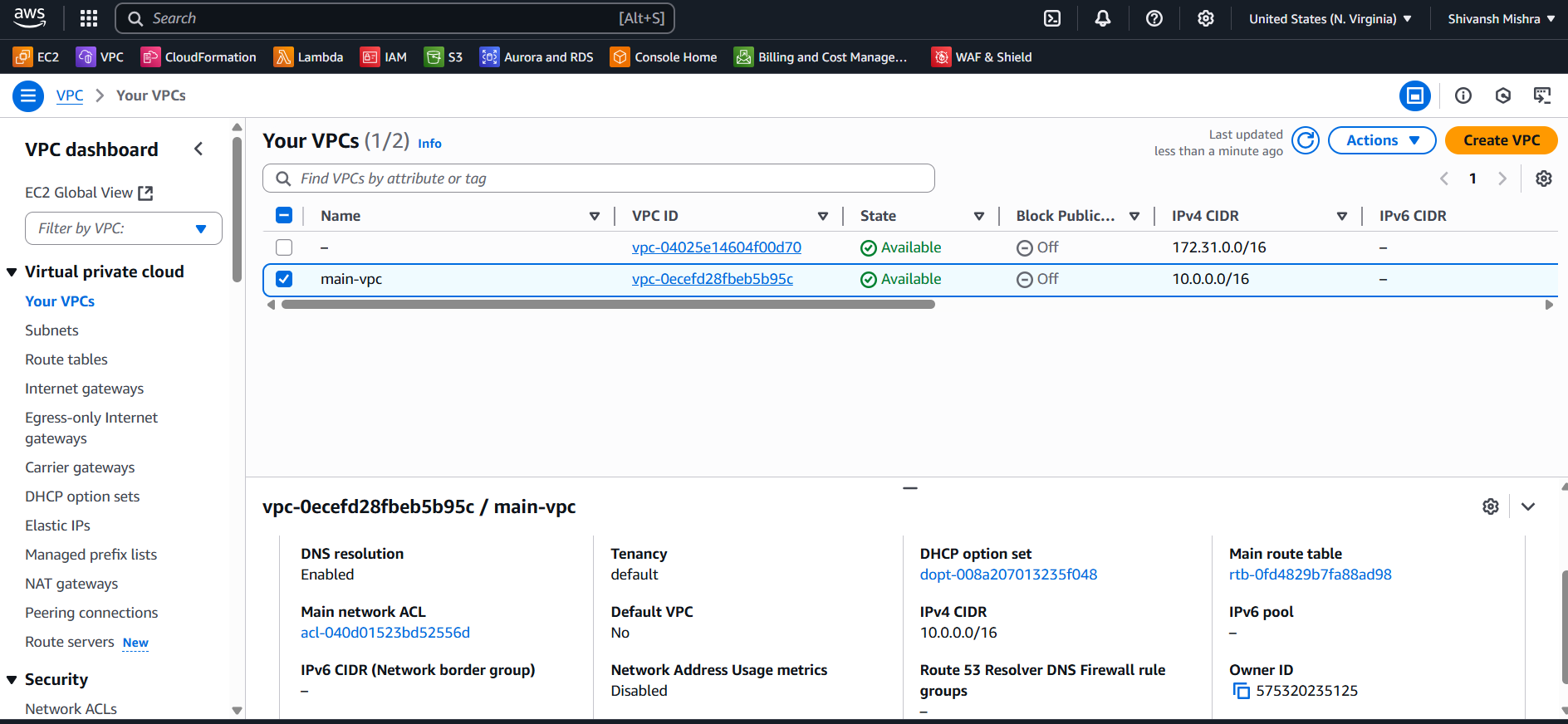
**6. Instance Creation**

****

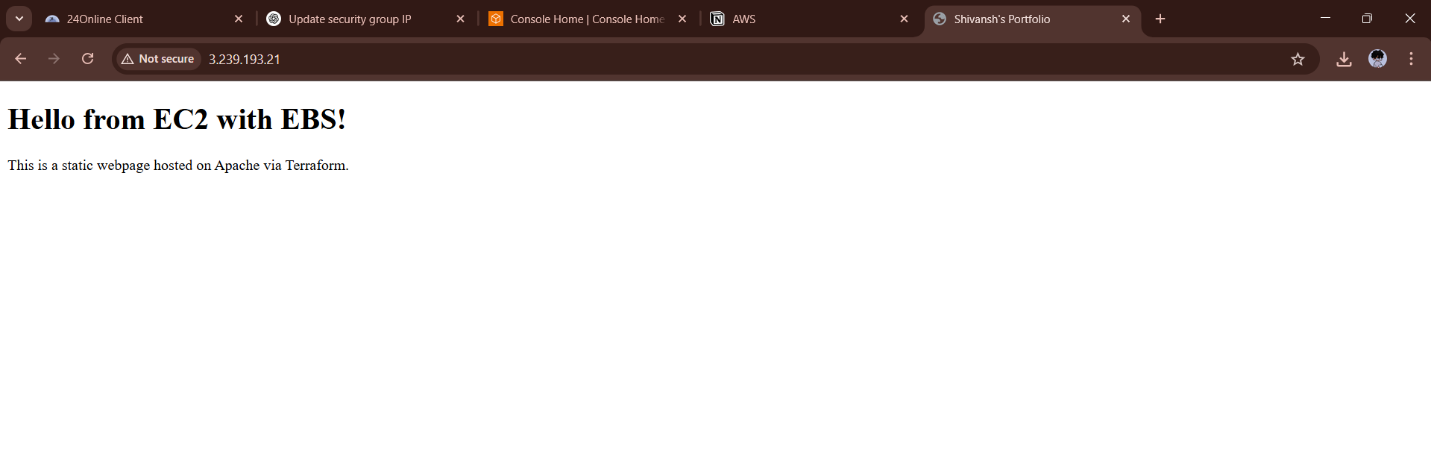
**7. EBS Creation**

****

**8. VPC creation**

****

**9.Web page**

****

**Conclusion:**

This project successfully demonstrates the deployment of a cloud-based web hosting environment using AWS EC2 and EBS, fully automated through Terraform. By leveraging Infrastructure as Code (IaC), we were able to design, configure, and manage scalable infrastructure efficiently. The project highlights foundational cloud skills such as provisioning compute instances, configuring networking components, attaching storage volumes, and automating server configuration. It reflects practical knowledge of deploying real-world applications in the cloud and sets the stage for more complex deployments in future phases.