Infrastructure Deployment and Application Setup using Terraform & GitHub Actions

1. Overview

This project demonstrates the provisioning of a secure AWS infrastructure using Terraform and automating deployments using GitHub Actions. The setup includes:

- VPC with 2 public and 2 private subnets
- EC2 instances:
    - Public subnet: SSH host
    - Private subnet: Application server
- NAT Gateway for private subnet internet access
- S3 bucket access from private EC2
- Application Load Balancer (ALB) to serve traffic
- Automated deployment using GitHub Actions workflows
- Nginx web server installation and deployment of a sample index.html page

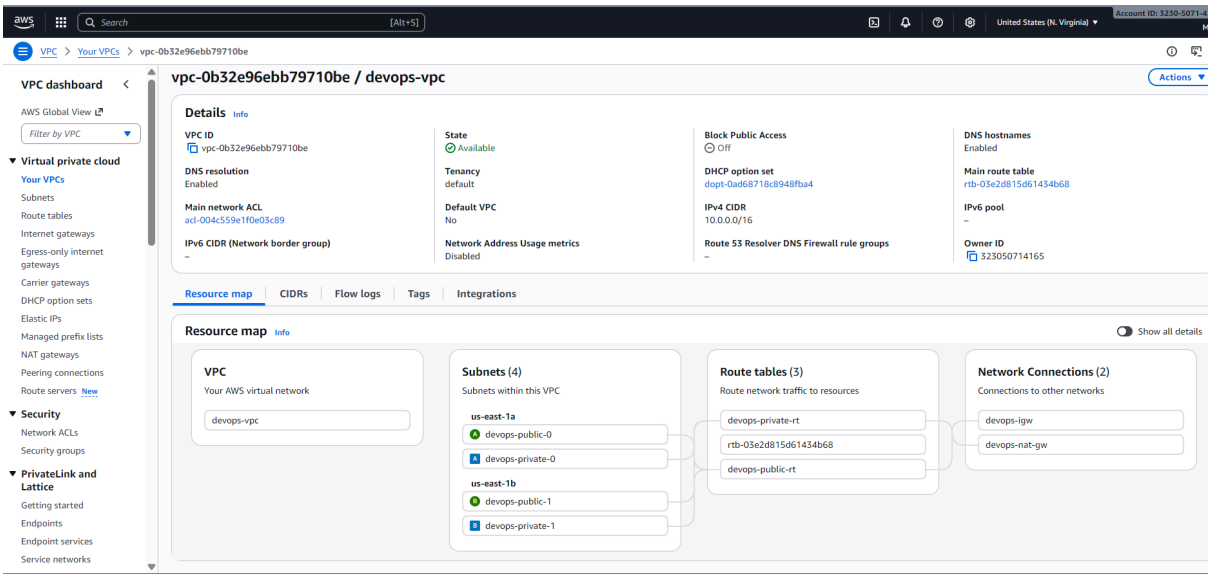
2. Terraform Script Overview

The Terraform code creates all necessary resources in a modular structure.

2.1 VPC and Networking

- VPC: CIDR 10.0.0.0/16
- Public subnets: 10.0.1.0/24, 10.0.2.0/24
- Private subnets: 10.0.3.0/24, 10.0.4.0/24

- Internet Gateway (IGW) attached to VPC
- Route Tables:
    - Public route table routes 0.0.0.0/0 via IGW
    - Private route table routes 0.0.0.0/0 via NAT Gateway



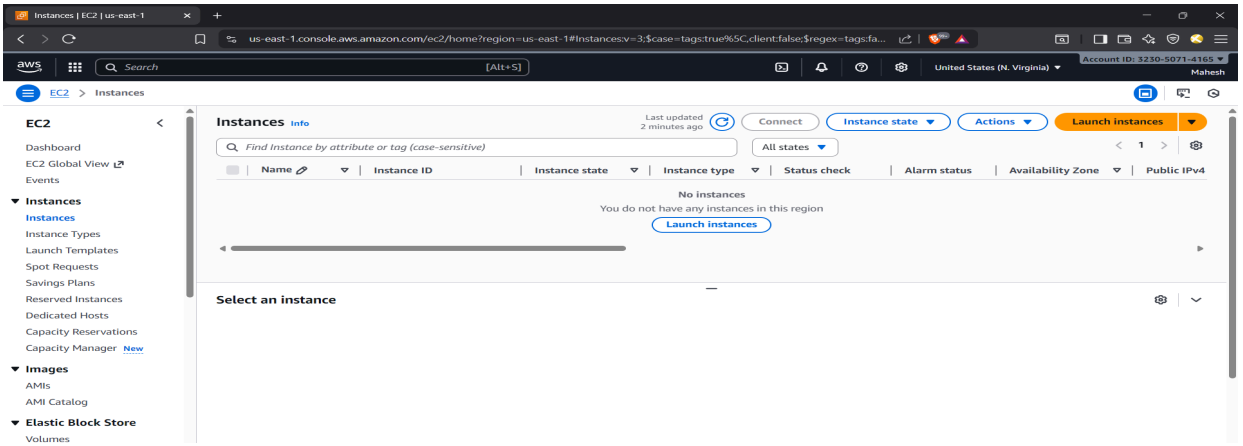
2.2 NAT Gateway

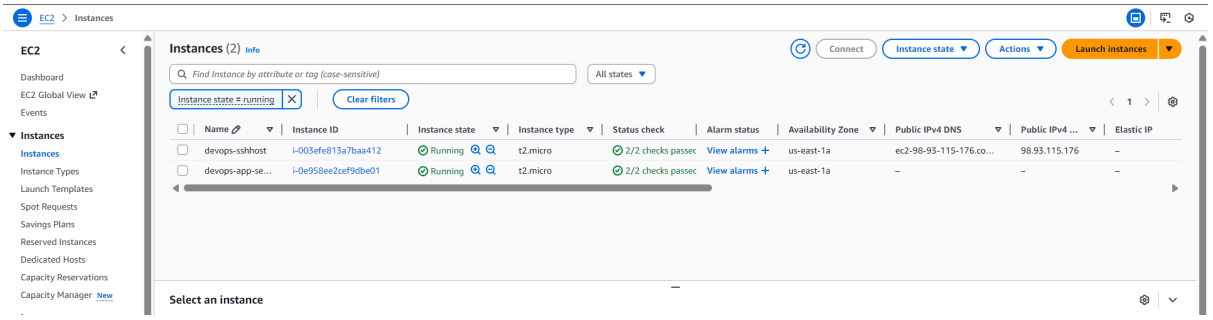
- Allocated Elastic IP for NAT
- NAT Gateway deployed in public subnet to allow private EC2 internet access

2.3 EC2 Instances

1. SSH Host (Public EC2)
    - Subnet: Public
    - Security Group: allows inbound SSH (port 22)
    - Used as a jump host to reach private EC2
2. App Server (Private EC2)

- Subnet: Private
- Security Group: allows inbound traffic from SSH host and port 80 from ALB
- No public IP (accessed via NAT or SSH host)
- IAM role attached to allow S3 bucket access





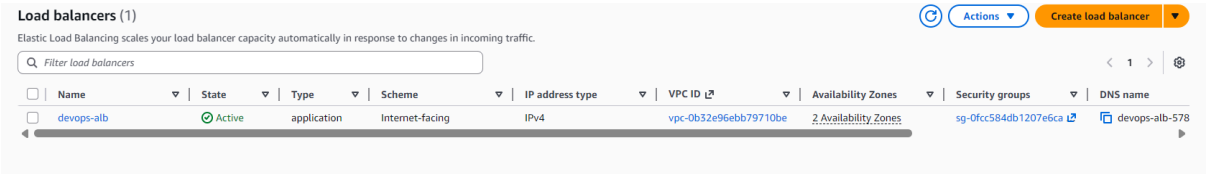
2.4 S3 Bucket

- Private bucket created
- Accessed from private EC2 via IAM role
- VPC endpoint can be added for fully private S3 access

2.5 Application Load Balancer (ALB)

- ALB created in public subnets

- Security Group allows HTTP traffic from internet
- Target Group attaches private EC2 instance
- Listener configured on port 80



2.6 Terraform Outputs

- vpc_id: VPC ID
- sshhost_public_ip: Public IP of SSH Host
- app_private_ip: Private IP of App EC2
- alb_dns_name: DNS of ALB
- s3_bucket_name: Name of S3 bucket

3. GitHub Actions Workflows

3.1 Shared Workflow (shared-workflow.yaml)

- Handles Terraform deployment
- Steps:
    1. Checkout repository
    2. Configure AWS credentials with Secrets
    3. Setup Terraform
    4. Terraform init, plan, and apply
    5. Capture outputs:

        - SSH host public IP
        - Private EC2 IP
        - ALB DNS
        - PEM key file path
- Outputs declared to be consumed by main workflow

3.2 Main Workflow (main.yaml)

- Triggers:
    1. workflow_dispatch (manual)
    2. push to main branch
- Jobs:
    1. Terraform job
        - Calls the shared workflow
    2. Configure EC2 & Verify ALB
        - Runs on ubuntu-latest
        - Steps:
            1. Checkout repository
            2. Get Terraform outputs
            3. Install SSH client
            4. Write PEM file from GitHub secrets
            5. Upload index.html and PEM to SSH host
            6. SSH into SSH host and copy file to private EC2

7. Install Nginx, move index.html to /var/www/html/, enable & restart Nginx

8. Verify application via ALB using curl

4. Application Deployment

- Nginx installed on private EC2
- index.html uploaded via SSH host
- ALB forwards HTTP traffic to private EC2
- Verify using ALB DNS:

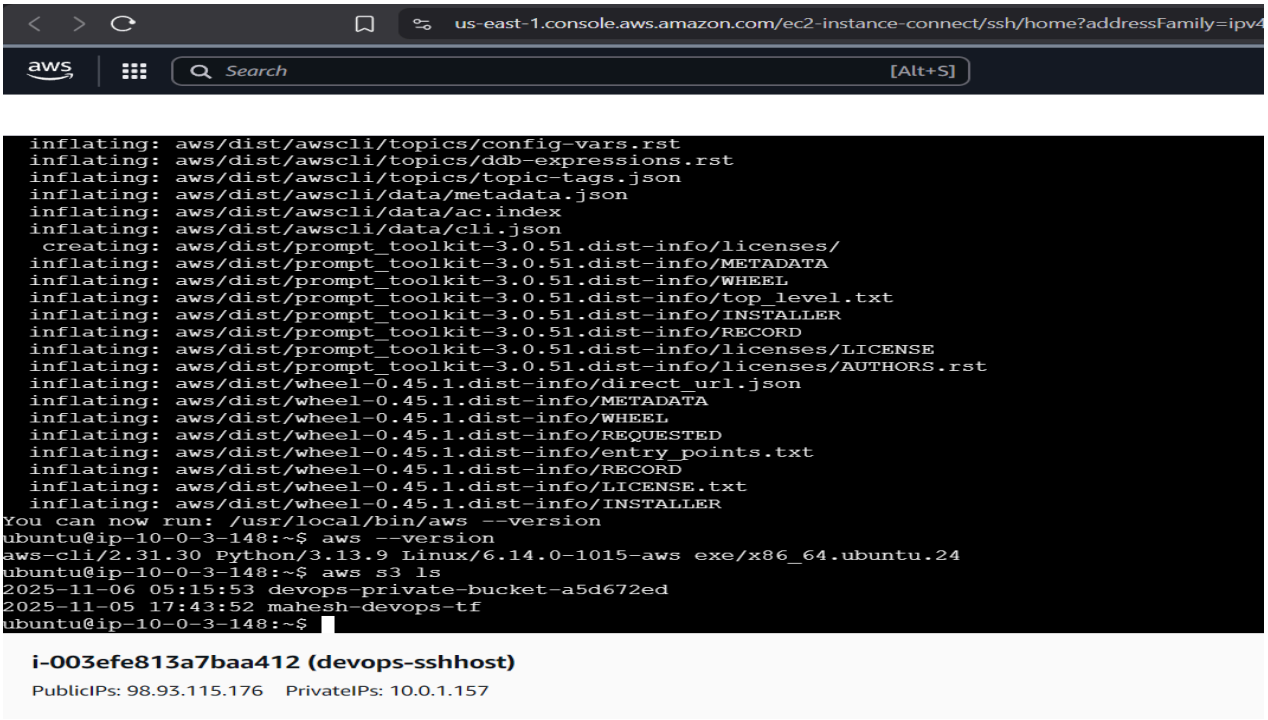
  # curl -I <alb dns>

  Expected response: HTTP 200 OK serving your Hello World page

5. Private S3 Access Verification

- EC2 instance has IAM role allowing S3 access
- VPC Endpoint (optional) ensures traffic does not traverse the public internet
- Test access from private EC2:

  # aws s3 ls



6. Summary

This setup demonstrates:

1. Infrastructure as Code with Terraform
2. Secure network using public/private subnets and NAT
3. Deployment automation using GitHub Actions
4. Private EC2 access to S3
5. Load balanced application using ALB
6. Application deployment via SSH host