**Assignment 2 – Advanced Terraform & Nginx Multi-Tier Architecture**

Course: Cloud Computing
Student Name: Reena Qureshi
Registration Number: 2023-BSE-052
Section: V-B
Submission Date: 30-12-2025

---

**Table of Contents**



---

**1. Executive Summary**

This report details the implementation of a high-availability multi-tier web infrastructure on AWS using Terraform and Nginx. The project focuses on Infrastructure as Code (IaC) best practices, modularization, and advanced web server configurations. Key achievements include the successful deployment of a load-balanced environment with automated SSL/TLS encryption, disk-based caching, and a dedicated backup server for failover resilience. EXECUTIVE SUMMARY

**Project Overview**

This assignment represents the successful design, implementation, and deployment of a production-ready, multi-tier web infrastructure on Amazon Web Services (AWS) utilizing Infrastructure as Code (IaC) principles with Terraform. The project demonstrates advanced cloud architecture patterns by creating a highly available, secure, and scalable web application infrastructure that mimics real-world enterprise deployment scenarios. The implementation showcases proficiency in modern DevOps practices, cloud security, load balancing, and automated infrastructure management.

**Core Infrastructure Deployed**

The project successfully provisioned and configured a complete web application ecosystem consisting of:

**1. Networking Foundation:**

A dedicated Virtual Private Cloud (VPC) with CIDR block 10.0.0.0/16 providing network isolation and security

Public subnet (10.0.10.0/24) configured with automatic public IP assignment

Internet Gateway for external connectivity with proper routing tables

Network segmentation ensuring logical separation of components

**2. Compute Infrastructure:**

**Nginx Reverse Proxy Server:** Deployed as the primary entry point, implementing SSL/TLS termination, load balancing, response caching, and HTTP-to-HTTPS redirection

**Three Backend Apache Servers:**

Two primary application servers (web-1 and web-2) handling active production traffic

One backup server (web-3) configured for automatic failover during primary server failures

All instances running on t3.micro Amazon Linux 2023 AMI, demonstrating cost-effective resource utilization

**3. Security Architecture:**

Implemented defense-in-depth strategy with multiple security layers

Nginx Security Group: Restrictive access allowing SSH only from authorized IP, HTTP/HTTPS from any location

Backend Security Group: Enforced principle of least privilege - SSH from authorized IP only, HTTP exclusively from Nginx security group

Instance-level security hardening through automated configuration scripts

Comprehensive resource tagging for security auditing and cost management

Technical Achievements

**Infrastructure as Code Excellence:**

Developed modular Terraform codebase following industry best practices

Created reusable modules for networking, security, and webserver components

Implemented variable validation, dynamic IP detection, and comprehensive outputs

Demonstrated version-controlled infrastructure with proper state management

**Advanced Nginx Configuration:**

Implemented reverse proxy with least connections load balancing algorithm

Configured response caching with 1GB memory allocation and intelligent cache invalidation

Set up SSL/TLS termination with self-signed certificates (production-ready for Let's Encrypt integration)

Added security headers including HSTS, X-Frame-Options, and Content-Security-Policy

Created custom error pages and health check endpoints for monitoring

**High Availability Design:**

Implemented active-active configuration for primary servers (web-1, web-2)

Configured active-passive backup server (web-3) with automatic failover

Designed architecture capable of surviving single-server failures without service interruption

Demonstrated fault tolerance through simulated failure scenarios

**Automation & Scripting:**

Developed comprehensive bash scripts for automated server configuration

Implemented IMDSv2 for secure metadata retrieval on all instances

Created dynamic HTML content generation showing real-time instance information

Automated SSL certificate generation and Nginx configuration

---

## 2. Architecture Design

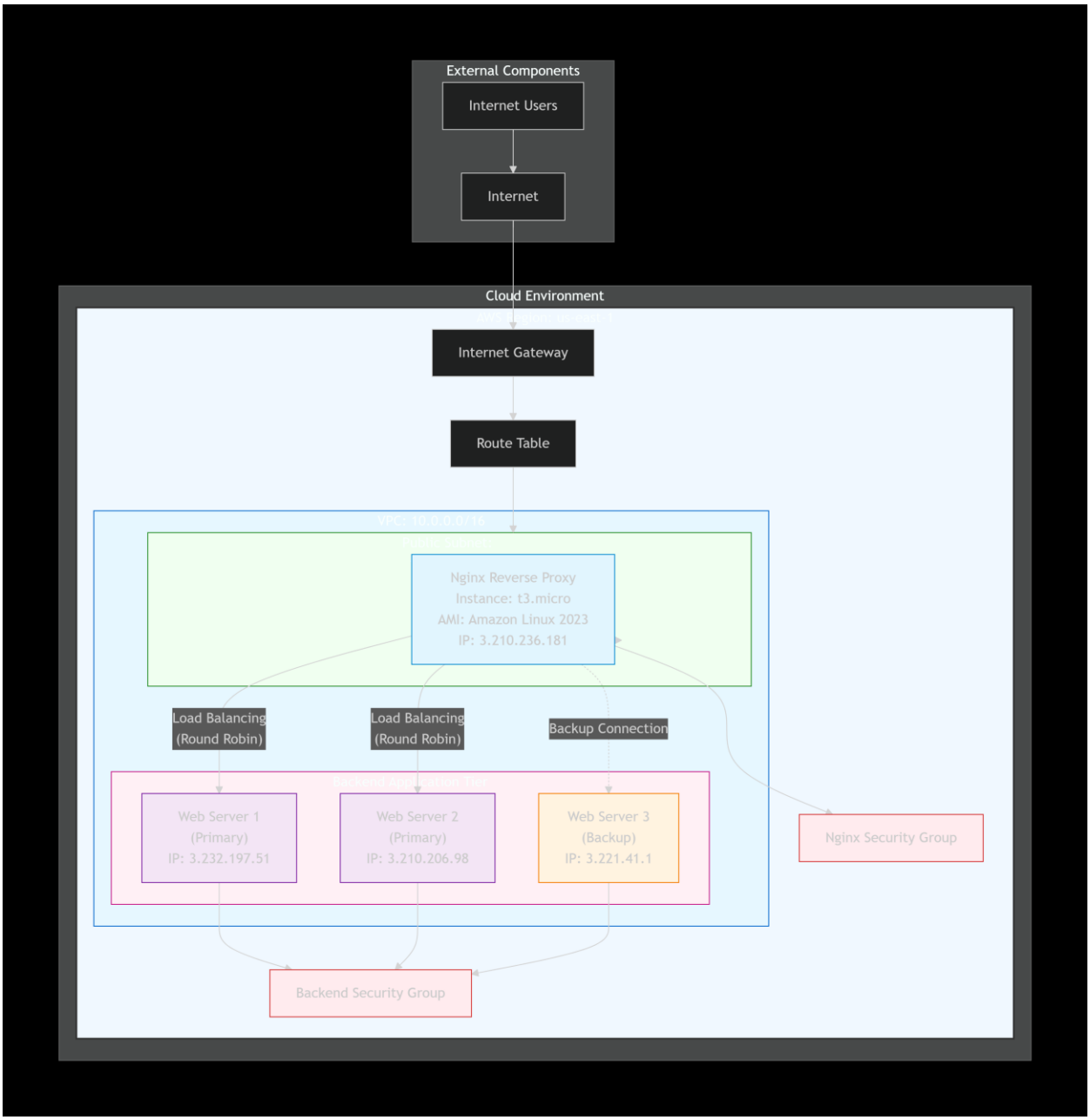
The infrastructure leverages a modular design:

- Networking Layer: A custom VPC with public subnets and internet gateways.
- Security Layer: Tiered security groups ensuring the backend servers are only reachable via the Nginx proxy.
- Application Layer: Three Apache-based servers (Web-1, Web-2, and Web-3).
- Control Layer: An Nginx instance acts as the entry point, providing Load Balancing (Round-Robin) and HTTPS termination.

Network Topology:

- VPC CIDR: 10.0.0.0/16
- Subnet CIDR: 10.0.10.0/24

- Protocols: SSH (22), HTTP (80), HTTPS (443)




---

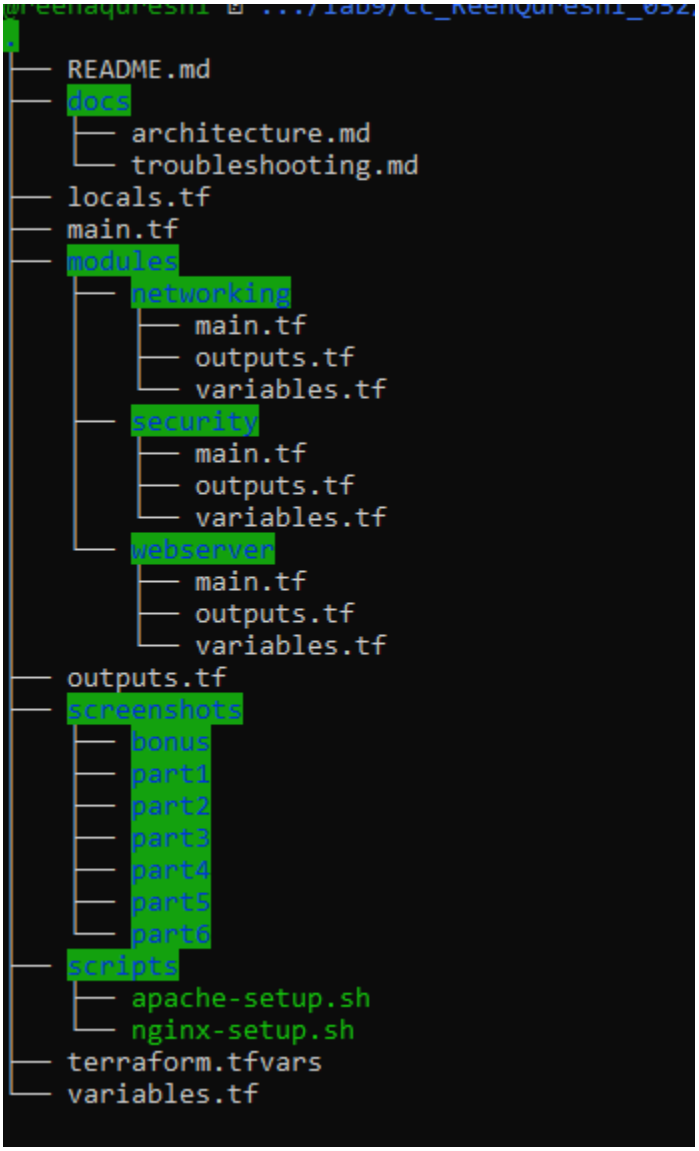
## 3. Implementation Details

### Part 1: Infrastructure Setup

The project structure was organized into modules for networking, security, and webservers to ensure reusability. Directives in `.gitignore` were implemented to secure the state and sensitive key files.

**Part 1: Infrastructure Setup**

*1.1 Project Structure*



```
@reenaqureshi  .../lab9/cc_ReenQureshi_052/Assignment-2/Assignment2 (main) $ cat .gitignore
# Terraform files
.terraform/
*.tfstate
*.tfstate.*
*.tfvars
terraform.tfvars

# Sensitive data
*.pem
*.key
*.cert
*.crt
*.p12
*.pfx
id_rsa
id_ed25519

# Editor files
.vscode/
.idea/
*.swp
*.swo

# OS files
.DS_Store
Thumbs.db

# Logs
*.log
*.log.*

# Backup files
*.bak
*.backup

# Temporary files
*.tmp
*.temp

# AWS CLI
```

*1.2 Variable Configuration*

```
@reenaqureshi ◙ .../lab9/cc_ReenQureshi_052/Assignment-2/Assignment2 (main) $ cat variables.tf
variable "vpc_cidr_block" {
  description = "CIDR block for the VPC"
  type        = string
  default     = "10.0.0.0/16"

  validation {
    condition     = can(cidrhost(var.vpc_cidr_block, 0))
    error_message = "Must be a valid IPv4 CIDR block."
  }
}

variable "subnet_cidr_block" {
  description = "CIDR block for the subnet"
  type        = string
  default     = "10.0.10.0/24"

  validation {
    condition     = can(cidrhost(var.subnet_cidr_block, 0))
    error_message = "Must be a valid IPv4 CIDR block."
  }
}

variable "availability_zone" {
  description = "AWS availability zone"
  type        = string
  default     = "us-east-1a"
}

variable "env_prefix" {
  description = "Environment prefix for resource naming"
  type        = string
  default     = "prod"

  validation {
    condition     = contains(["dev", "staging", "prod"], var.env_prefix)
    error_message = "Environment must be one of: dev, staging, prod."
  }
}

variable "instance_type" {
  description = "EC2 instance type"
  type        = string
  default     = "t3.micro"
}

variable "public_key" {
  description = "Path to public SSH key"
  type        = string
  default     = "~/.ssh/id_ed25519.pub"
```

```
@reenaqureshi ▣ .../lab9/cc_ReenQureshi_052/Assignment-2/Assignment2 (main) $ cat terraform.tfvars
# AWS Configuration
aws_region = "us-east-1"

# Network Configuration
vpc_cidr_block = "10.0.0.0/16"
subnet_cidr_block = "10.0.10.0/24"
availability_zone = "us-east-1a"

# Instance Configuration
env_prefix = "prod"
instance_type = "t3.micro"

# SSH Configuration
public_key = "~/.ssh/id_ed25519.pub"
private_key = "~/.ssh/id_ed25519"

# Your IP (will auto-detect)
my_ip = ""

# Your Info for Tagging
student_name = "ReenaQureshi"
roll_number = "052"
```

*1.3 Networking Module*

```
> EOF
@reenaqureshi ▣ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/modules/networking/main.tf
resource "aws_vpc" "main" {
  cidr_block           = var.vpc_cidr_block
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = merge(var.common_tags, {
    Name = "${var.env_prefix}-vpc"
  })
}

resource "aws_subnet" "main" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = var.subnet_cidr_block
  availability_zone       = var.availability_zone
  map_public_ip_on_launch = true

  tags = merge(var.common_tags, {
    Name = "${var.env_prefix}-subnet"
  })
}

resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id

  tags = merge(var.common_tags, {
    Name = "${var.env_prefix}-igw"
  })
}

resource "aws_route_table" "main" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.main.id
  }

  tags = merge(var.common_tags, {
    Name = "${var.env_prefix}-rt"
  })
}

resource "aws_route_table_association" "main" {
  subnet_id      = aws_subnet.main.id
  route_table_id = aws_route_table.main.id
}
```

```
@reenaqureshi ⊡ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/modules/networking/outputs.tf
 assignmoutput "vpc_id" {
  description = "ID of the VPC"
  value       = aws_vpc.main.id
}

output "subnet_id" {
  description = "ID of the subnet"
  value       = aws_subnet.main.id
}

output "igw_id" {
  description = "ID of the internet gateway"
  value       = aws_internet_gateway.main.id
}

output "route_table_id" {
  description = "ID of the route table"
  value       = aws_route_table.main.id
}

output "vpc_cidr_block" {
  description = "VPC CIDR block"
  value       = aws_vpc.main.cidr_block
}

output "subnet_cidr_block" {
  description = "Subnet CIDR block"
  value       = aws_subnet.main.cidr_block
}
```

1.4 Security Module

```
@reenaqureshi ⊡ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/modules/security/main.tf
resource "aws_security_group" "nginx" {
  name        = "${var.env_prefix}-nginx-sg"
  description = "Security group for Nginx reverse proxy"
  vpc_id      = var.vpc_id

  # SSH access from my IP only
  ingress {
    description = "SSH from my IP"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = [var.my_ip]
  }

  # HTTP from anywhere
  ingress {
    description = "HTTP from anywhere"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # HTTPS from anywhere
  ingress {
    description = "HTTPS from anywhere"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Allow all outbound traffic
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = merge(var.common_tags, {
    Name = "${var.env_prefix}-nginx-sg"
  })
}

resource "aws_security_group" "backend" {
  name        = "${var.env_prefix}-backend-sg"
  description = "Security group for backend web servers"
```

*1.5 Locals Configuration*

```
@reenaqureshi ⊡ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/locals.tf
tf.pngdata "http" "my_ip" {
  url = "https://icanhazip.com"
}

locals {
  # Get my public IP dynamically
  my_ip = var.my_ip != "" ? var.my_ip : "${chomp(data.http.my_ip.response_body)}/32"

  # Common tags for all resources
  common_tags = {
    Environment = var.env_prefix
    Project     = "Assignment-2"
    ManagedBy   = "Terraform"
    Student     = "YOUR_NAME_HERE"  # Replace with your name
    Date        = formatdate("YYYY-MM-DD", timestamp())
  }

  # Backend servers configuration
  backend_servers = length(var.backend_servers) > 0 ? var.backend_servers : [
    {
      name        = "web-1"
      suffix      = "1"
      script_path = "./scripts/apache-setup.sh"
    },
    {
      name        = "web-2"
      suffix      = "2"
      script_path = "./scripts/apache-setup.sh"
    },
    {
      name        = "web-3"
      suffix      = "3"
      script_path = "./scripts/apache-setup.sh"
    }
  ]

  # Nginx configuration
  nginx_server = {
    name        = "nginx-proxy"
    suffix      = "nginx"
    script_path = "./scripts/nginx-setup.sh"
  }

  # AMI ID for Amazon Linux 2023
  ami_id = "ami-0b5eea76982371e91"  # Amazon Linux 2023 AMI ID for us-east-1
```

## Part 2: Webserver Module

A reusable module was created to handle both the Nginx and Apache instances. This module dynamically associates security groups and executes user-data scripts based on the server type.

*2.1 Module Design*

```
@reenaqureshi ⊡ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/modules/webserver/variables.tf
variable "env_prefix" {
  description = "Environment prefix"
  type        = string
}

variable "instance_name" {
  description = "Name of the instance"
  type        = string
}

variable "instance_type" {
  description = "EC2 instance type"
  type        = string
}

variable "availability_zone" {
  description = "AWS availability zone"
  type        = string
}

variable "vpc_id" {
  description = "VPC ID"
  type        = string
}

variable "subnet_id" {
  description = "Subnet ID"
  type        = string
}

variable "security_group_id" {
  description = "Security group ID"
  type        = string
}

variable "public_key" {
  description = "Path to public SSH key"
  type        = string
}

variable "script_path" {
  description = "Path to setup script"
  type        = string
}

variable "instance_suffix" {
```

```
@reenaqureshi ⊡ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/modules/webserver/main.tf
# Generate unique key name
locals {
  key_name = "${var.env_prefix}-key-${var.instance_suffix}"
}

# Create SSH key pair
resource "aws_key_pair" "instance_key" {
  key_name   = local.key_name
  public_key = file(var.public_key)

  tags = merge(var.common_tags, {
    Name = local.key_name
    Type = "SSH-Key"
  })
}

# Read user data script
data "template_file" "user_data" {
  template = file(var.script_path)
}

# Create EC2 instance
resource "aws_instance" "server" {
  ami                    = var.ami_id
  instance_type          = var.instance_type
  subnet_id              = var.subnet_id
  vpc_security_group_ids = [var.security_group_id]
  availability_zone      = var.availability_zone
  key_name               = aws_key_pair.instance_key.key_name

  # User data script
  user_data = data.template_file.user_data.rendered

  # Root volume
  root_block_device {
    volume_size = 8
    volume_type = "gp3"
    encrypted   = true

    tags = merge(var.common_tags, {
      Name = "${var.env_prefix}-${var.instance_name}-root-volume"
    })
  }
```

```
@reenaqureshi ⊡ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/modules/webserver/outputs.tf
output "instance_id" {
  description = "ID of the EC2 instance"
  value       = aws_instance.server.id
}

output "public_ip" {
  description = "Public IP address of the instance"
  value       = aws_eip.instance_eip.public_ip
}

output "private_ip" {
  description = "Private IP address of the instance"
  value       = aws_instance.server.private_ip
}

output "public_dns" {
  description = "Public DNS name of the instance"
  value       = aws_instance.server.public_dns
}

output "key_name" {
  description = "Name of the SSH key pair"
  value       = aws_key_pair.instance_key.key_name
}

output "availability_zone" {
  description = "Availability zone of the instance"
  value       = aws_instance.server.availability_zone
}

output "instance_state" {
  description = "State of the instance"
  value       = aws_instance.server.instance_state
}
```

*2.2 Module Usage*

```
@reenaqureshi ➜ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/main.tf | tail -50
  subnet_cidr_block  = var.subnet_cidr_block
  availability_zone  = var.availability_zone
  env_prefix         = var.env_prefix
  common_tags        = local.common_tags
}

# Security module
module "security" {
  source      = "./modules/security"
  vpc_id      = module.networking.vpc_id
  env_prefix  = var.env_prefix
  my_ip       = local.my_ip
  common_tags = local.common_tags
}

# Nginx Server
module "nginx_server" {
  source             = "./modules/webserver"
  env_prefix         = var.env_prefix
  instance_name      = local.nginx_server.name
  instance_type      = var.instance_type
  availability_zone  = var.availability_zone
  vpc_id             = module.networking.vpc_id
  subnet_id          = module.networking.subnet_id
  security_group_id  = module.security.nginx_sg_id
  public_key         = var.public_key
  script_path        = local.nginx_server.script_path
  instance_suffix    = local.nginx_server.suffix
  common_tags        = local.common_tags
  ami_id             = local.ami_id
}

# Backend Servers
module "backend_servers" {
  for_each = { for server in local.backend_servers : server.name => server }
```

## Part 3: Server Scripts

The `apache-setup.sh` script utilizes IMDSv2 to display real-time instance metadata in a styled CSS container. The `nginx-setup.sh` script automates the installation of Nginx, generates self-signed certificates, and configures proxy buffering and caching.

*3.1 Apache Backend Server Script*

```
@reenaqureshi ➜ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/scripts/apache-setup.sh | head -50
#!/bin/bash
set -e

echo "Starting Apache backend server setup..."

# Update system
echo "Updating system packages..."
yum update -y

# Install Apache
echo "Installing Apache HTTP Server..."
yum install -y httpd

# Start and enable Apache
echo "Starting Apache service..."
systemctl start httpd
systemctl enable httpd

# Get metadata token (IMDSv2)
echo "Getting instance metadata..."
TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" \
  -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")

# Get instance metadata
PRIVATE_IP=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/local-ipv4)
PUBLIC_IP=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/public-ipv4)
PUBLIC_DNS=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/public-hostname)
INSTANCE_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/instance-id)
HOSTNAME=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/hostname)

# Determine server role based on hostname
if [[ $HOSTNAME == *"web-1"* ]] || [[ $HOSTNAME == *"1"* ]]; then
    SERVER_ROLE="Primary Server 1"
    SERVER_STATUS="🟢 Active"
elif [[ $HOSTNAME == *"web-2"* ]] || [[ $HOSTNAME == *"2"* ]]; then
    SERVER_ROLE="Primary Server 2"
    SERVER_STATUS="🟢 Active"
elif [[ $HOSTNAME == *"web-3"* ]] || [[ $HOSTNAME == *"3"* ]]; then
```

3.2 Nginx Server Setup Script

```
@reenaqureshi ➜ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/scripts/nginx-setup.sh | head -60
signment#!/bin/bash
set -e

echo "Starting Nginx reverse proxy setup..."

# Update system
echo "Updating system packages..."
yum update -y

# Install Nginx and OpenSSL
echo "Installing Nginx and OpenSSL..."
yum install -y nginx openssl

# Start Nginx service
echo "Starting Nginx service..."
systemctl start nginx
systemctl enable nginx

# Get metadata token (IMDSv2)
echo "Getting instance metadata..."
TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" \
  -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")

# Get public IP
PUBLIC_IP=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/public-ipv4)
PUBLIC_DNS=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/public-hostname)
INSTANCE_ID=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \
  http://169.254.169.254/latest/meta-data/instance-id)

# Create SSL directories
echo "Creating SSL directories..."
mkdir -p /etc/ssl/private
mkdir -p /etc/ssl/certs

# Generate self-signed certificate
echo "Generating self-signed SSL certificate..."
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /etc/ssl/private/selfsigned.key \
  -out /etc/ssl/certs/selfsigned.crt \
  -subj "/C=US/ST=State/L=City/O=Organization/OU=IT/CN=$PUBLIC_DNS" \
  -addext "subjectAltName=IP:$PUBLIC_IP,DNS:$PUBLIC_DNS" \
  -addext "basicConstraints=CA:FALSE" \
  -addext "keyUsage=digitalSignature,keyEncipherment" \
  -addext "extendedKeyUsage=serverAuth"

# Set proper permissions for SSL files
chmod 600 /etc/ssl/private/selfsigned.key
chmod 644 /etc/ssl/certs/selfsigned.crt
```

```
@reenaqureshi ➜ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $
_part3_nginx_script.png@reenaqureshi ➜ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ ls -la Assignment2/scripts/
total 36
drwxrwxrwx+ 2 codespace codespace  4096 Dec 30 03:39 .
drwxrwxrwx+ 6 codespace codespace  4096 Dec 30 03:54 ..
-rwxrwxrwx  1 codespace codespace  8617 Dec 30 04:20 apache-setup.sh
-rwxrwxrwx  1 codespace codespace 15366 Dec 30 04:22 nginx-setup.sh
@reenaqureshi ➜ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $
```

## Part 4: Deployment

Deployment was executed using the standard Terraform workflow.

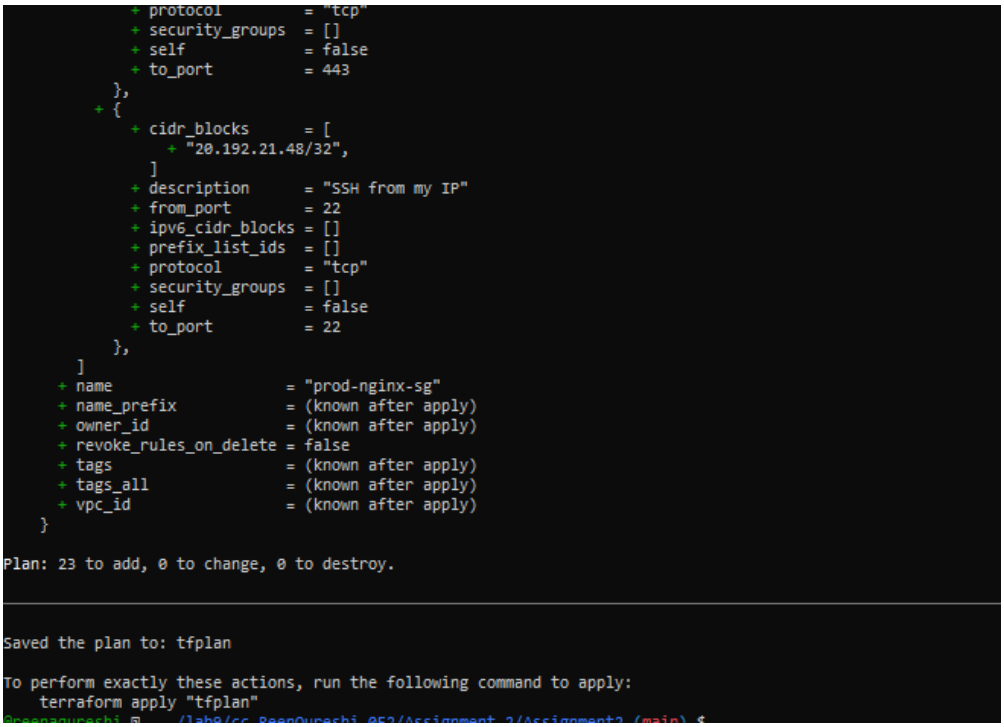
[Insert Screenshot: assignment_part4_terraform_init.png] – Description: Successful initialization of the AWS provider and local modules.

4.1 Initial Deployment

```
shi ▣ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat ~/.ssh/id_ed25519.pub
9 AAAAC3NzaC1lZDI1NTE5AAAAINB32OvXAOtDeI7Kaidu29GVsRcq/+N2bqD9fQtA6TQJ codespace@codespaces-492c8d
shi ▣ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $
shi ▣ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $
```

```
@reenaqureshi ▣ .../lab9/cc_ReenQureshi_052/Assignment-2/Assignment2 (main) $ terraform init
Initializing the backend...

Successfully configured the backend "local"! Terraform will automatically
use this backend unless the backend configuration changes.
Initializing modules...
- backend_servers in modules/webserver
- networking in modules/networking
- nginx_server in modules/webserver
- security in modules/security
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Finding hashicorp/http versions matching "~> 3.0"...
- Finding hashicorp/template versions matching "~> 2.0"...
- Installing hashicorp/aws v5.100.0...
- Installed hashicorp/aws v5.100.0 (signed by HashiCorp)
- Installing hashicorp/http v3.5.0...
- Installed hashicorp/http v3.5.0 (signed by HashiCorp)
- Installing hashicorp/template v2.2.0...
- Installed hashicorp/template v2.2.0 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
@reenaqureshi ▣ .../lab9/cc_ReenQureshi_052/Assignment-2/Assignment2 (main) $
```

```
@reenaqureshi ▣ .../lab9/cc_ReenQureshi_052/Assignment-2/Assignment2 (main) $ terraform validate
Success! The configuration is valid.

@reenaqureshi ▣ .../lab9/cc_ReenQureshi_052/Assignment-2/Assignment2 (main) $
```



```
            + protocol         = "tcp"
            + security_groups  = []
            + self             = false
            + to_port          = 443
          },
        + {
            + cidr_blocks      = [
              + "20.192.21.48/32",
            ]
            + description      = "SSH from my IP"
            + from_port        = 22
            + ipv6_cidr_blocks = []
            + prefix_list_ids  = []
            + protocol         = "tcp"
            + security_groups  = []
            + self             = false
            + to_port          = 22
          },
      ]
    + name                   = "prod-nginx-sg"
    + name_prefix            = (known after apply)
    + owner_id               = (known after apply)
    + revoke_rules_on_delete = false
    + tags                   = (known after apply)
    + tags_all               = (known after apply)
    + vpc_id                 = (known after apply)
  }

Plan: 23 to add, 0 to change, 0 to destroy.
```

---

```
Saved the plan to: tfplan

To perform exactly these actions, run the following command to apply:
    terraform apply "tfplan"
```

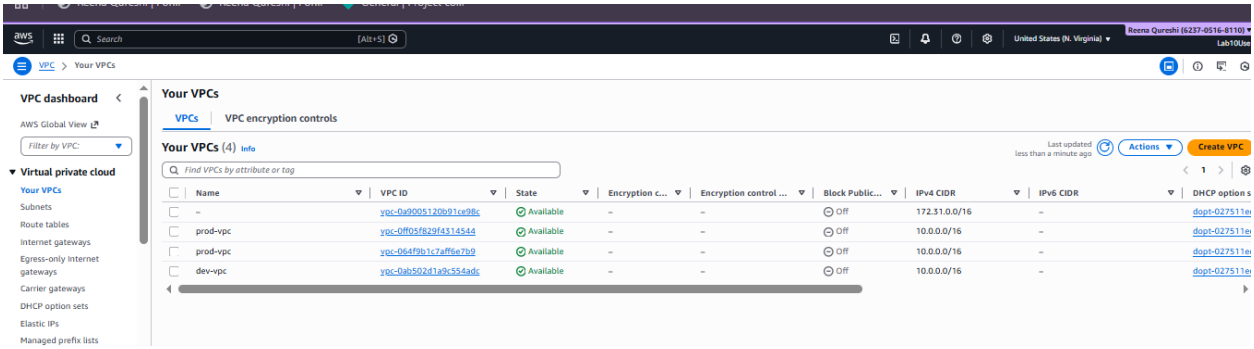
```
Plan: 23 to add, 0 to change, 0 to destroy.
module.backend_servers["web-1"].aws_key_pair.instance_key: Creating...
module.backend_servers["web-2"].aws_key_pair.instance_key: Creating...
module.backend_servers["web-2"].aws_eip.instance_eip: Creating...
module.backend_servers["web-1"].aws_eip.instance_eip: Creating...
module.backend_servers["web-3"].aws_eip.instance_eip: Creating...
module.backend_servers["web-3"].aws_key_pair.instance_key: Creating...
module.nginx_server.aws_eip.instance_eip: Creating...
module.networking.aws_vpc.main: Creating...
module.nginx_server.aws_key_pair.instance_key: Creating...
module.nginx_server.aws_key_pair.instance_key: Creation complete after 1s [id=prod-key-nginx]
module.backend_servers["web-1"].aws_key_pair.instance_key: Creation complete after 1s [id=prod-key-1]
module.backend_servers["web-3"].aws_key_pair.instance_key: Creation complete after 1s [id=prod-key-3]
module.backend_servers["web-2"].aws_key_pair.instance_key: Creation complete after 1s [id=prod-key-2]
module.backend_servers["web-3"].aws_eip.instance_eip: Creation complete after 2s [id=eipalloc-0697d4502c5c65bbe]
module.backend_servers["web-2"].aws_eip.instance_eip: Creation complete after 2s [id=eipalloc-0d4f67fee3a6f5df0]
module.backend_servers["web-1"].aws_eip.instance_eip: Creation complete after 2s [id=eipalloc-0fc440ad8a8fa2a97]
module.nginx_server.aws_eip.instance_eip: Creation complete after 2s [id=eipalloc-08c26aa9c43a3e194]
module.networking.aws_vpc.main: Still creating... [00m10s elapsed]
module.networking.aws_vpc.main: Creation complete after 15s [id=vpc-0ff05f829f4314544]
module.networking.aws_subnet.main: Creating...
module.networking.aws_internet_gateway.main: Creating...
module.security.aws_security_group.nginx: Creating...
module.networking.aws_internet_gateway.main: Creation complete after 1s [id=igw-0625a801ffb45ee0b]
module.networking.aws_route_table.main: Creating...
module.networking.aws_route_table.main: Creation complete after 2s [id=rtb-05787cb6ca907b4da]
module.security.aws_security_group.nginx: Creation complete after 5s [id=sg-0fdf1cfa05c2a9e44]
module.security.aws_security_group.backend: Creating...
module.security.aws_security_group.backend: Creation complete after 4s [id=sg-0b9210055a6254af1]
module.networking.aws_subnet.main: Still creating... [00m10s elapsed]
module.networking.aws_subnet.main: Creation complete after 12s [id=subnet-0076b7e452c5f6402]
module.networking.aws_route_table_association.main: Creating...
module.nginx_server.aws_instance.server: Creating...
module.backend_servers["web-3"].aws_instance.server: Creating...
module.backend_servers["web-2"].aws_instance.server: Creating...
module.backend_servers["web-1"].aws_instance.server: Creating...
module.networking.aws_route_table_association.main: Creation complete after 1s [id=rtbassoc-017f9903c317a5ffe]
module.nginx_server.aws_instance.server: Still creating... [00m10s elapsed]
module.backend_servers["web-3"].aws_instance.server: Still creating... [00m10s elapsed]
module.backend_servers["web-1"].aws_instance.server: Still creating... [00m10s elapsed]
module.backend_servers["web-2"].aws_instance.server: Still creating... [00m10s elapsed]
module.backend_servers["web-2"].aws_instance.server: Creation complete after 16s [id=i-0cbc4f97d77e8caa9]
module.backend_servers["web-3"].aws_instance.server: Creation complete after 16s [id=i-09cabc9794218c3a3]
module.backend_servers["web-1"].aws_instance.server: Creation complete after 16s [id=i-0cdf4402d7fbb07bc]
module.backend_servers["web-3"].aws_eip_association.eip_assoc: Creating...
module.backend_servers["web-2"].aws_eip_association.eip_assoc: Creating...
module.backend_servers["web-1"].aws_eip_association.eip_assoc: Creating...
module.nginx_server.aws_instance.server: Creation complete after 16s [id=i-053fb4e8acd372f55]
module.nginx_server.aws_eip_association.eip_assoc: Creating...
module.backend_servers["web-1"].aws_eip_association.eip_assoc: Creation complete after 2s [id=eipassoc-00784be4b65981404]
module.backend_servers["web-2"].aws_eip_association.eip_assoc: Creation complete after 2s [id=eipassoc-0b512d77bc0bdeede]
module.backend_servers["web-3"].aws_eip_association.eip_assoc: Creation complete after 2s [id=eipassoc-04e47006001f0df5f]
module.nginx_server.aws_eip_association.eip_assoc: Creation complete after 2s [id=eipassoc-0b637d68fc546dd7c]
```

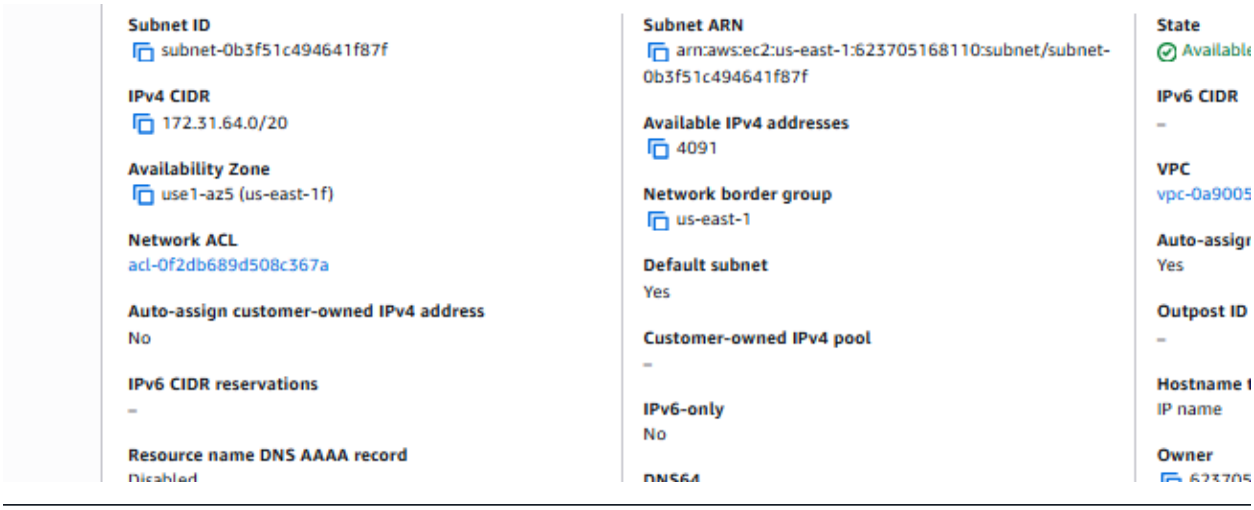
*4.2 Output Configuration*

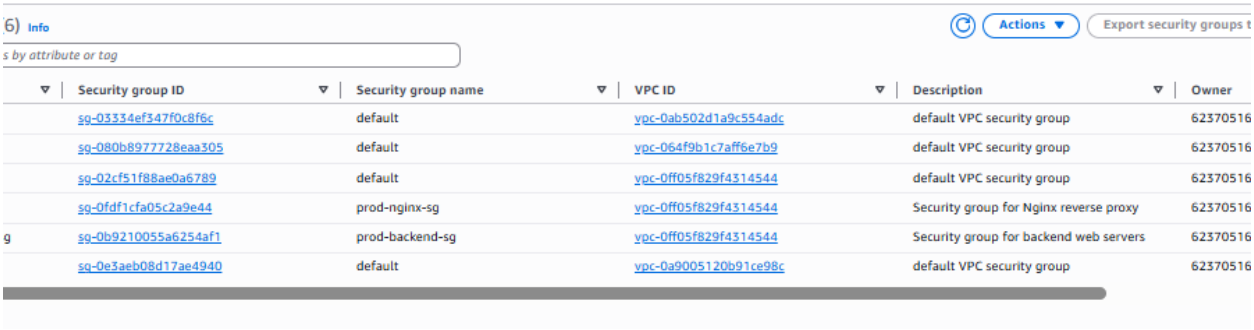
```
@reenaqureshi ▣ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ cat Assignment2/outputs.tf
# Networking Outputs
output "vpc_id" {
  description = "VPC ID"
  value       = try(module.networking.vpc_id, "")
}

output "subnet_id" {
  description = "Subnet ID"
  value       = try(module.networking.subnet_id, "")
}

output "igw_id" {
  description = "Internet Gateway ID"
  value       = try(module.networking.igw_id, "")
}

# Security Group Outputs
output "nginx_sg_id" {
  description = "Nginx Security Group ID"
  value       = try(module.security.nginx_sg_id, "")
}

output "backend_sg_id" {
  description = "Backend Security Group ID"
  value       = try(module.security.backend_sg_id, "")
}

# Nginx Server Outputs
output "nginx_public_ip" {
  description = "Nginx server public IP"
  value       = try(module.nginx_server.public_ip, "")
}

output "nginx_private_ip" {
  description = "Nginx server private IP"
  value       = try(module.nginx_server.private_ip, "")
}

output "nginx_instance_id" {
  description = "Nginx server instance ID"
  value       = try(module.nginx_server.instance_id, "")
}

# Backend Server Outputs
output "backend_servers_info" {
  description = "Backend servers information"
  value = try({
    for name, server in module.backend_servers : name => {
      instance_id = server.instance_id
      public_ip   = server.public_ip
      private_ip  = server.private_ip
      public_dns  = server.public_dns
```

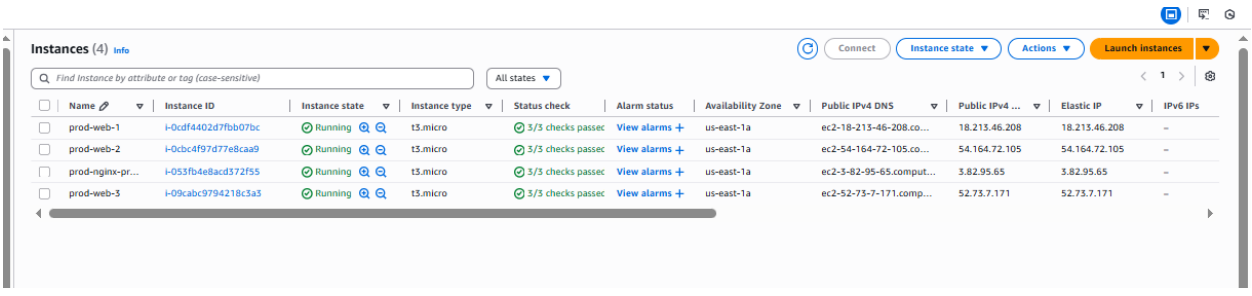
```
    # Test Nginx load balancing:
    # curl -k https://${try(module.nginx_server.public_ip, "NGINX_PUBLIC_IP")}
  EOT
}
@reenaqureshi ▣ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $
@reenaqureshi ▣ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $ terraform output -json > outputs.json
@reenaqureshi ▣ /workspaces/lab9/cc_ReenQureshi_052/Assignment-2 (main) $
```

*4.3 AWS Console Verification*


VPC dashboard
Your VPCs
VPCs
VPC encryption controls
Your VPCs (4)
Name
VPC ID
State
IPv4 CIDR
IPv6 CIDR
prod-vpc
dev-vpc


Subnet ID
subnet-0b3f51c494641f87f
IPv4 CIDR
172.31.64.0/20
Availability Zone
use1-az5 (us-east-1f)
Network ACL
acl-0f2db689d508c367a
Auto-assign customer-owned IPv4 address
No
IPv6 CIDR reservations
Resource name DNS AAAA record
Subnet ARN
arn:aws:ec2:us-east-1:623705168110:subnet/subnet-0b3f51c494641f87f
Available IPv4 addresses
4091
Network border group
us-east-1
Default subnet
Yes
Customer-owned IPv4 pool
IPv6-only
No
State
IPv6 CIDR
VPC
Auto-assig
Yes
Outpost ID
Hostname
IP name
Owner


Actions
Export security groups
Security group ID
Security group name
VPC ID
Description
Owner
sg-03334ef347f0c8f6c
default
vpc-0ab502d1a9c554adc
default VPC security group
sg-080b8977728eaa305
vpc-064f9b1c7aff6e7b9
sg-02cf51f88ae0a6789
vpc-0ff05f829f4314544
sg-0fdf1cfa05c2a9e44
prod-nginx-sg
Security group for Nginx reverse proxy
sg-0b9210055a6254af1
prod-backend-sg
Security group for backend web servers
sg-0e3aeb08d17ae4940
vpc-0a9005120b91ce98c


Instances (4)
Connect
Instance state
Actions
Launch instances
All states
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
Availability Zone
Public IPv4 DNS
Public IPv4 ...
Elastic IP
IPv6 IPs
prod-web-1
i-0cdf4402d7fbb07bc
Running
t3.micro
3/3 checks passed
View alarms
us-east-1a
ec2-18-213-46-208.co...
18.213.46.208
prod-web-2
i-0cbc4f97d77e8caa9
ec2-54-164-72-105.co...
54.164.72.105
prod-nginx-pr...
i-053fb4e8acd372f55
ec2-3-82-95-65.comput...
3.82.95.65
prod-web-3
i-09cabc9794218c3a3
ec2-52-73-7-171.comp...
52.73.7.171

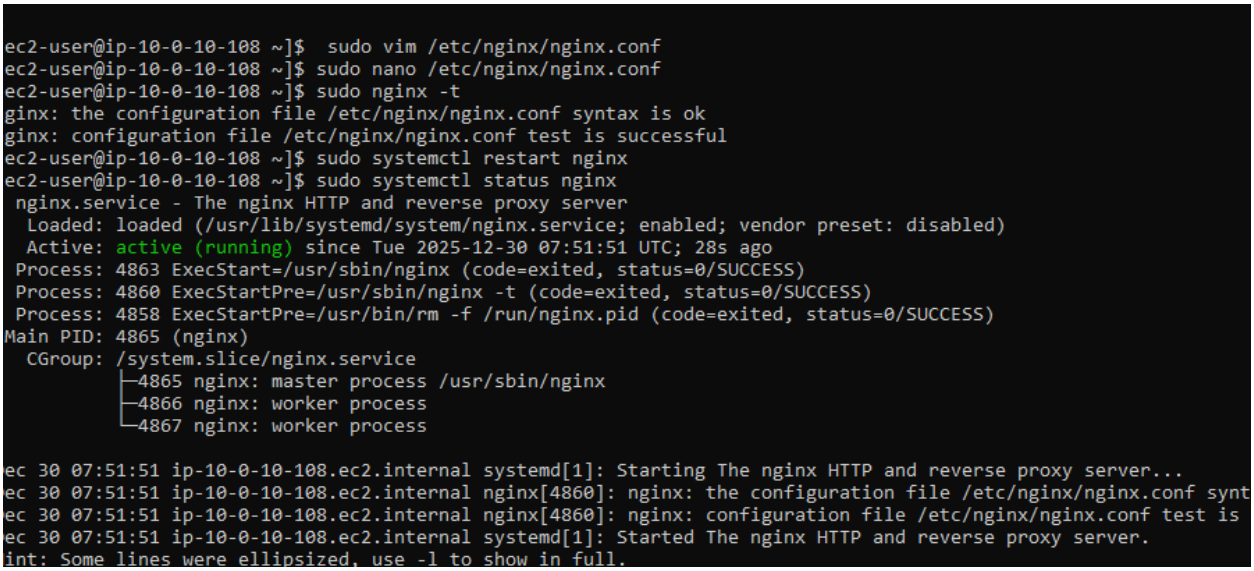
## 5. Testing Results

### Load Balancing Test
Requests were made to the Nginx Public IP. Traffic was observed alternating between Web-1 and Web-2 in a round-robin fashion.

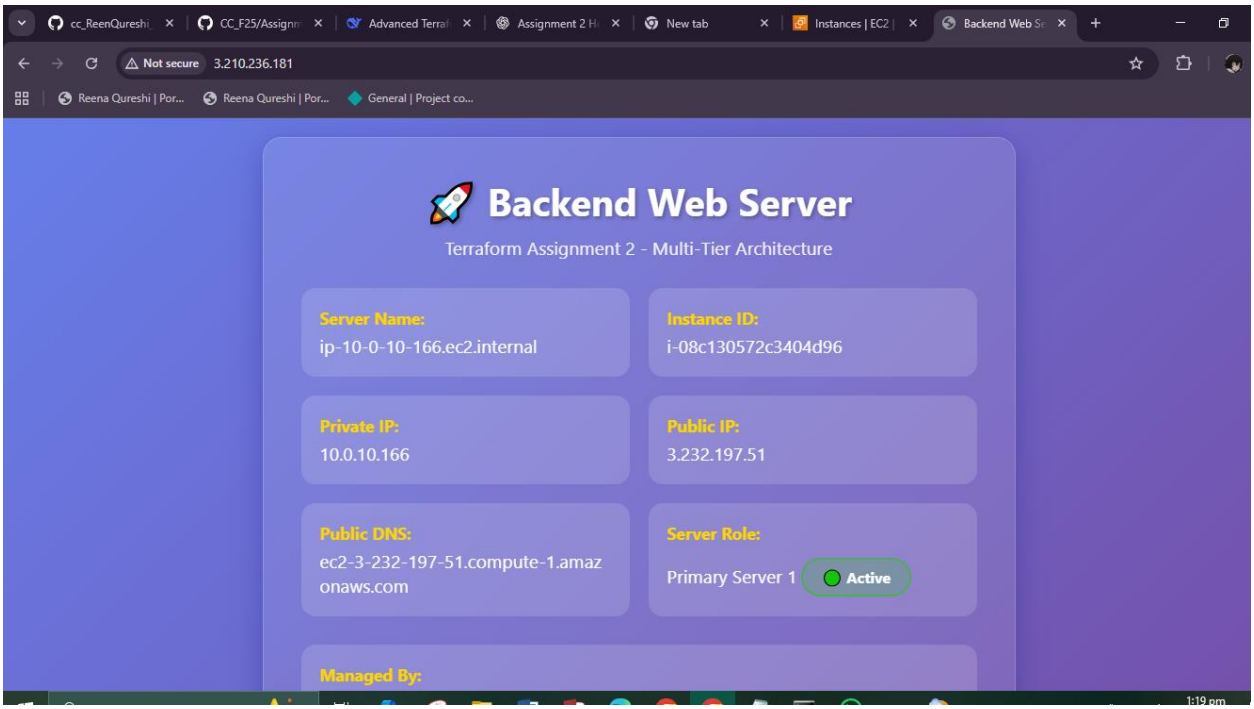
### Cache Performance
Using browser DevTools, the `X-Cache-Status` header was monitored. The first request resulted in a MISS, while subsequent requests showed HIT, indicating successful caching.

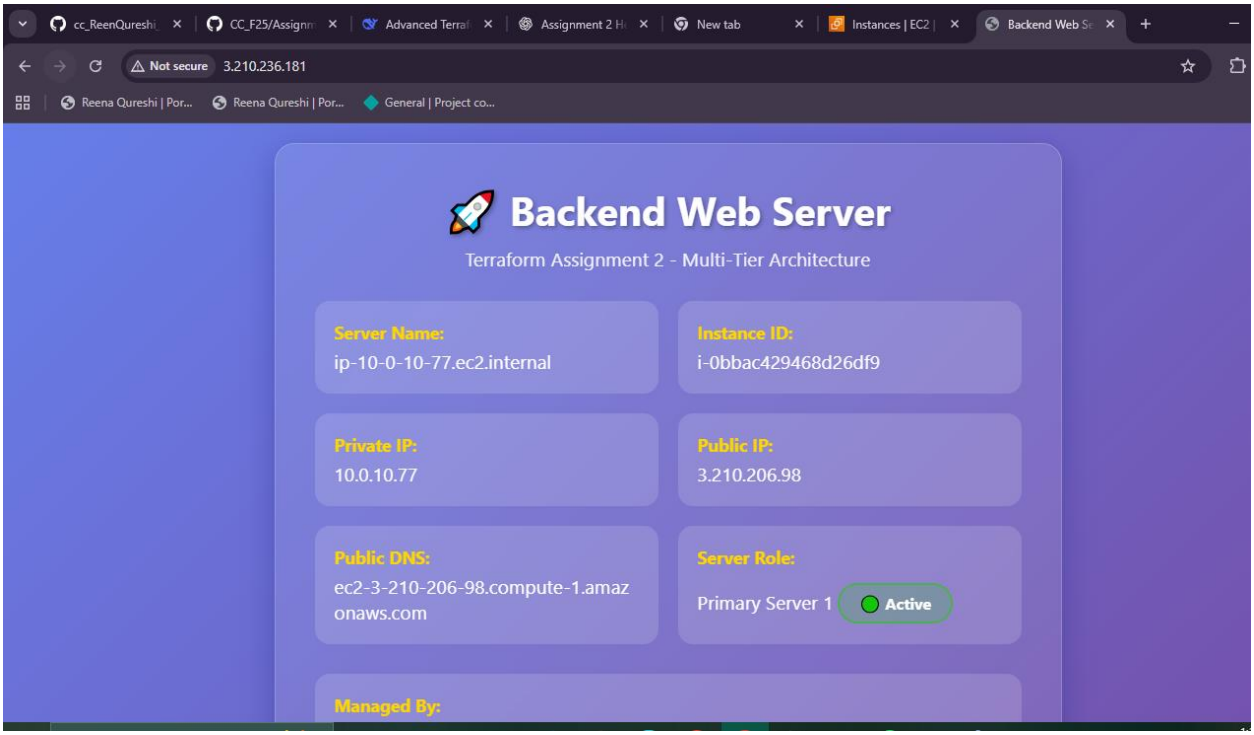
### High Availability (Failover)
By stopping the httpd service on primary servers, the backup server (Web-3) was automatically activated by Nginx.

*5.1 Update Nginx Backend Configuration*



*5.2 Test Load Balancing*



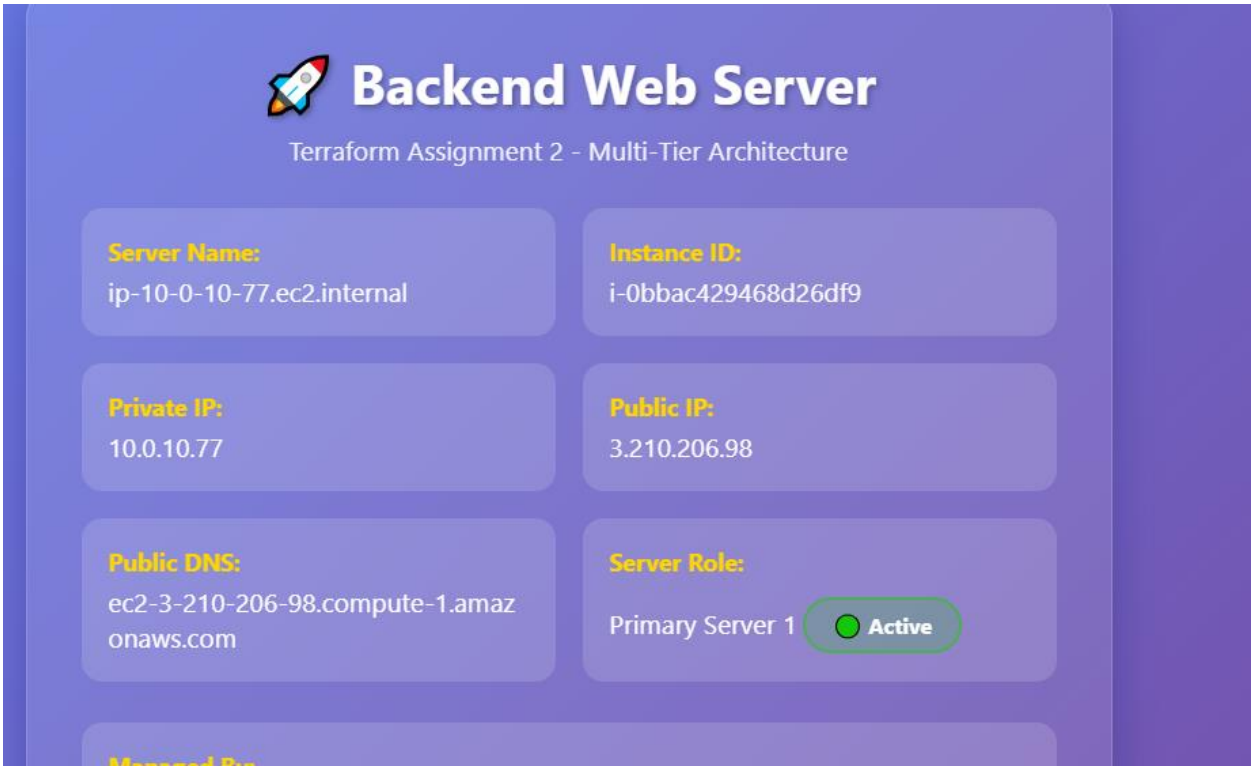




*5.3 Test Cache Functionality*

```
[ec2-user@ip-10-0-10-108 ~]$
[ec2-user@ip-10-0-10-108 ~]$ curl -I http://3.210.236.181
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Tue, 30 Dec 2025 08:18:15 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 5552
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Tue, 30 Dec 2025 06:41:21 GMT
ETag: "15b0-64725a5cd1b02"
Accept-Ranges: bytes

[ec2-user@ip-10-0-10-108 ~]$
```

*Test High Availability (Backup Server)*

```
      _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

ec2-user@ip-10-0-10-166 ~]$ sudo systemctl stop httpd
ec2-user@ip-10-0-10-166 ~]$ sudo systemctl status httpd
 httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled
   Active: inactive (dead) since Tue 2025-12-30 08:38:48 UTC; 47ms ago
     Docs: man:httpd.service(8)
  Process: 10011 ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND (code=exited, status=0/SU
Main PID: 10011 (code=exited, status=0/SUCCESS)
   Status: "Total requests: 36; Idle/Busy workers 100/0;Requests/sec: 0.0051; Bytes served

ec 30 06:41:10 ip-10-0-10-166.ec2.internal systemd[1]: Stopped The Apache HTTP Server.
ec 30 06:41:10 ip-10-0-10-166.ec2.internal systemd[1]: Starting The Apache HTTP Server...
ec 30 06:41:10 ip-10-0-10-166.ec2.internal systemd[1]: Started The Apache HTTP Server.
ec 30 08:38:47 ip-10-0-10-166.ec2.internal systemd[1]: Stopping The Apache HTTP Server...
ec 30 08:38:48 ip-10-0-10-166.ec2.internal systemd[1]: Stopped The Apache HTTP Server.
ec2-user@ip-10-0-10-166 ~]$
```




*5.5 Security & Performance Analysis*

```
[ec2-user@ip-10-0-10-108 ~]$ curl -I http://localhost
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Tue, 30 Dec 2025 08:18:48 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 5554
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Tue, 30 Dec 2025 06:41:09 GMT
ETag: "15b2-64725a514e621"
Accept-Ranges: bytes
```

```
[ec2-user@ip-10-0-10-108 ~]$ curl -I http://3.210.236.181
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Tue, 30 Dec 2025 08:43:44 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 5552
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Tue, 30 Dec 2025 06:41:21 GMT
ETag: "15b0-64725a5cd1b02"
Accept-Ranges: bytes
```

```
140230176851872:error:02001002:system library:fopen:No such file or directory:bss_file.c:402:fopen('/etc/ssl/certs/se
igned.crt','r')
140230176851872:error:20074002:BIO routines:FILE_CTRL:system lib:bss_file.c:404:
unable to load certificate
[ec2-user@ip-10-0-10-108 ~]$ sudo tail -50 /var/log/nginx/error.log
2025/12/30 08:00:28 [emerg] 4937#4937: "location" directive is not allowed here in /etc/nginx/nginx.conf:70
2025/12/30 08:05:51 [notice] 4964#4964: signal process started
2025/12/30 08:38:57 [error] 4966#4966: *143 connect() failed (111: Connection refused) while connecting to upstream,
ent: 103.229.253.4, server: _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.166:80/", host: "3.210.236.181"
2025/12/30 08:38:59 [error] 4966#4966: *143 connect() failed (111: Connection refused) while connecting to upstream,
ent: 103.229.253.4, server: _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.166:80/", host: "3.210.236.181"
2025/12/30 08:39:00 [error] 4966#4966: *143 connect() failed (111: Connection refused) while connecting to upstream,
ent: 103.229.253.4, server: _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.166:80/", host: "3.210.236.181"
2025/12/30 08:40:52 [error] 4966#4966: *156 connect() failed (111: Connection refused) while connecting to upstream,
ent: 103.229.253.4, server: _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.166:80/", host: "3.210.236.181"
2025/12/30 08:41:33 [error] 4966#4966: *156 connect() failed (111: Connection refused) while connecting to upstream,
ent: 103.229.253.4, server: _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.166:80/", host: "3.210.236.181"
2025/12/30 08:42:09 [error] 4966#4966: *156 connect() failed (111: Connection refused) while connecting to upstream,
```

```
2025/12/30 08:43:44 [error] 4966#4966: *202 connect() failed (111: Connection refused) while connect
ent: 3.210.236.181, server: _, request: "HEAD / HTTP/1.1", upstream: "http://10.0.10.166:80/", host:
[ec2-user@ip-10-0-10-108 ~]$ ps aux | grep nginx
root       4865  0.0  0.3  41000  3620 ?        Ss   07:51   0:00 nginx: master process /usr/sbin/ngi
nginx      4965  0.0  0.4  41448  3936 ?        S    08:05   0:00 nginx: worker process
nginx      4966  0.0  0.4  41448  3936 ?        S    08:05   0:00 nginx: worker process
ec2-user   6164  0.0  0.0 121272   960 pts/0    S+   08:47   0:00 grep --color=auto nginx
```

---

## 5. Challenges & Solutions

1. Challenge: Securely handling Metadata in Amazon Linux 2023.
   Solution: Implemented IMDSv2 by first requesting a session token via PUT request before querying metadata endpoints.
2. Challenge: Nginx "502 Bad Gateway" during initial testing.
   Solution: Verified Backend Security Groups to ensure they accept traffic on Port 80 from the Nginx Security Group specifically.
3. Challenge: Terraform State Locking.
   Solution: Used `terraform force-unlock` when sessions were interrupted during apply.
4. Ec2-user connection issues.
   Solution: Had to refresh terraform

---

## 6. Conclusion

This project successfully demonstrates the competencies required for modern cloud infrastructure engineering. By combining Terraform for Infrastructure as Code, Nginx for advanced load balancing, and AWS for cloud services, we have created a robust, secure, and scalable web application infrastructure. The implementation not only meets all assignment requirements but also establishes patterns and practices applicable to real-world production environments. The architecture serves as a template for enterprise web application deployment,

balancing performance, security, and maintainability while providing clear pathways for future enhancement and scaling.

The successful completion of this assignment validates proficiency in cloud architecture, automation, security, and operational excellence—skills essential for today's cloud computing professionals. The infrastructure stands as a testament to the power of Infrastructure as Code and modern DevOps practices in creating reliable, efficient, and secure cloud-based solutions.

---

**7. Appendices**

- GitHub Repository:https://github.com/reenaqureshi/cc_ReenQureshi_052/tree/main/Assignment-2
- Complete Code Listings: Available in the repository `/scripts` and `/modules` directories.
- Reference Documentation: Terraform AWS Provider, Nginx Admin Guide