Assignment 2 – Advanced Terraform & Nginx Multi-Tier Architecture

Name: Zunaira Noor

Roll Number: 2023-BSE-075

Course Name: Cloud Computing

Submission Date: 30-12-2025

Contents

1. Executive Summary

This assignment demonstrates the deployment of a **high-availability, multi-tier web infrastructure on AWS** using **Terraform modules** and **Nginx** as a reverse proxy and load balancer. The infrastructure includes **one Nginx server** configured for HTTPS, caching, and load balancing, along with **three backend Apache servers** (web-1, web-2, web-3), where web-3 serves as a backup to ensure high availability. This setup ensures continuous availability of the web application under various failure scenarios.

1.1 Key achievements include:

- **Terraform Modularization:** Separate modules for networking, security, and web servers for reusable and organized code.
- **Secure Networking:** VPC, subnet, Internet Gateway, and security groups configured for controlled access.
- **Load Balancing & High Availability:** Traffic is distributed between web-1 and web-2, with web-3 activating automatically if any primary server fails.
- **Caching & Performance:** Nginx caching verified with MISS and HIT responses to improve performance.
- **SSL/TLS & Security Headers:** HTTPS enabled with self-signed certificates and headers applied for security.
- **Automation:** Scripts automate server setup and metadata reporting, ensuring consistency and reducing manual effort.

This assignment demonstrates the **end-to-end deployment of a secure, scalable, and reliable web infrastructure** on AWS, highlighting best practices in **IaC, DevOps, and cloud architecture**.

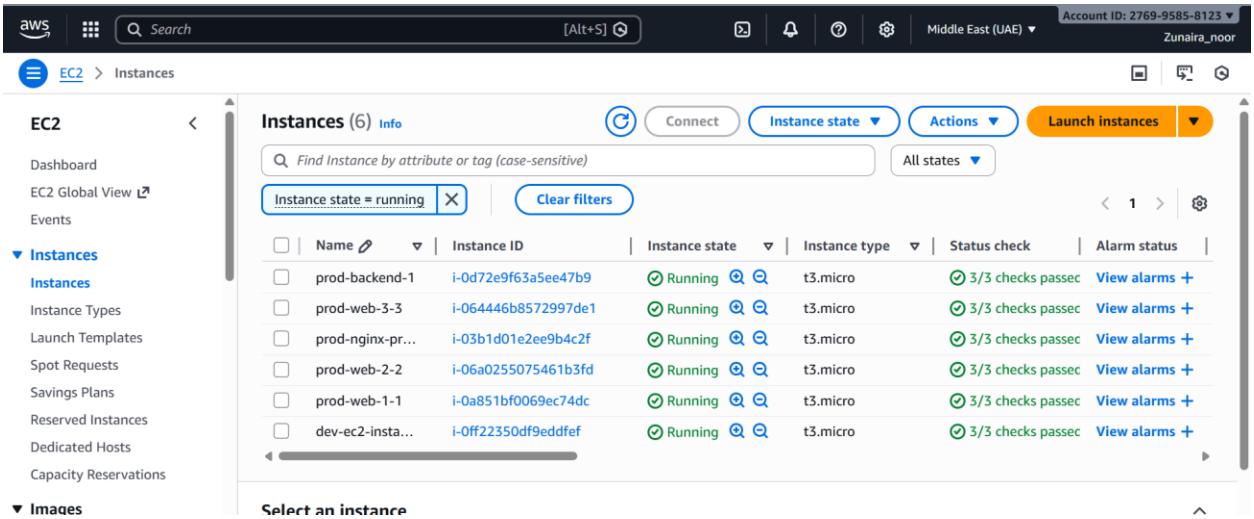
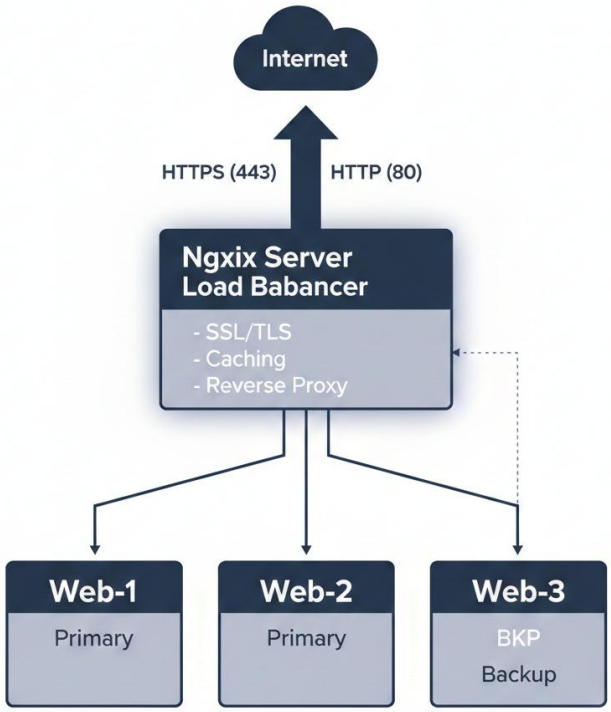


Figure 1: *Deployed EC2 Instances on AWS*

2. Architecture Design

The architecture for this assignment is a **high-availability, multi-tier web infrastructure** deployed on AWS, designed for scalability, security, and performance. It consists of **one Nginx server** acting as a reverse proxy and load balancer, and **three backend Apache web servers** (web-1, web-2, web-3). Web-3 serves as a backup server, activated only if any primary server fails, ensuring continuous availability.

2.1 Architecture Diagram



2.2 Components Description

Nginx Server (Load Balancer/Reverse Proxy):

- Handles HTTPS traffic, forwards requests to backend servers.
- Implements caching for performance.
- Redirects HTTP traffic to HTTPS.
- Monitors backend server health.

Backend Apache Servers (Web-1, Web-2, Web-3):

- Serve application content and metadata pages.
- Web-3 remains in backup mode, activated when primary servers fail.

Networking Components:

- **VPC & Subnets:** Private network for web servers; public subnet for Nginx.
- **Internet Gateway & Route Tables:** Enable internet access for Nginx.

Security Groups:

- **Nginx SG:** Allows SSH from admin IP and HTTP/HTTPS from anywhere.
- **Backend SG:** Allows SSH from admin IP and HTTP only from Nginx SG.

2.3 Key Design Features

- **High Availability:** Backup server ensures continuous service.
- **Scalability:** Terraform modules allow easy addition of more backend servers.
- **Security:** Restricted access via security groups, SSL/TLS, and headers.
- **Performance:** Caching and load balancing reduce latency and improve response times.

Figure 2: AWS Architecture Overview

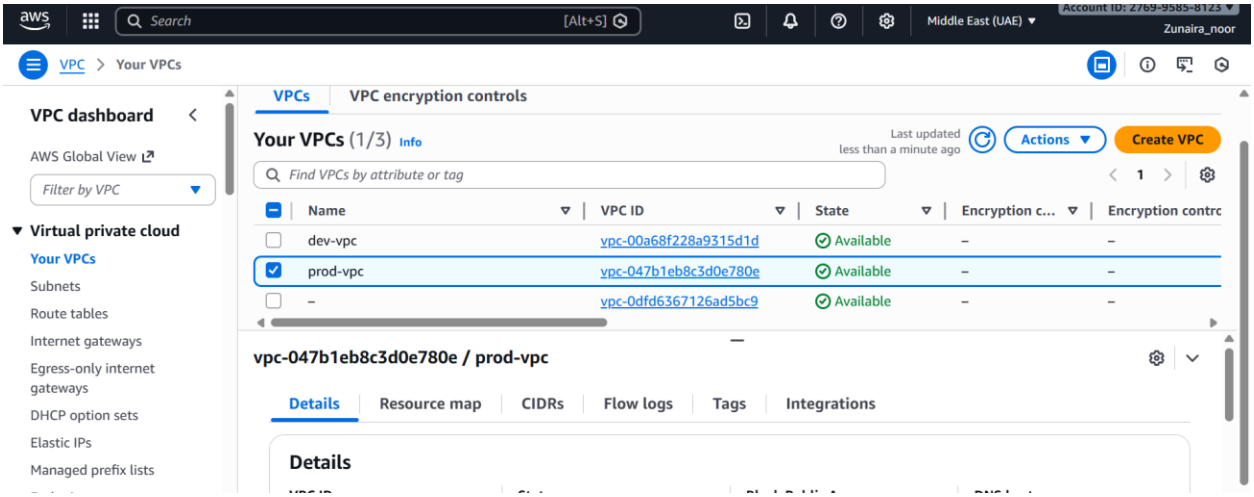


Figure 2a: Deployed VPC

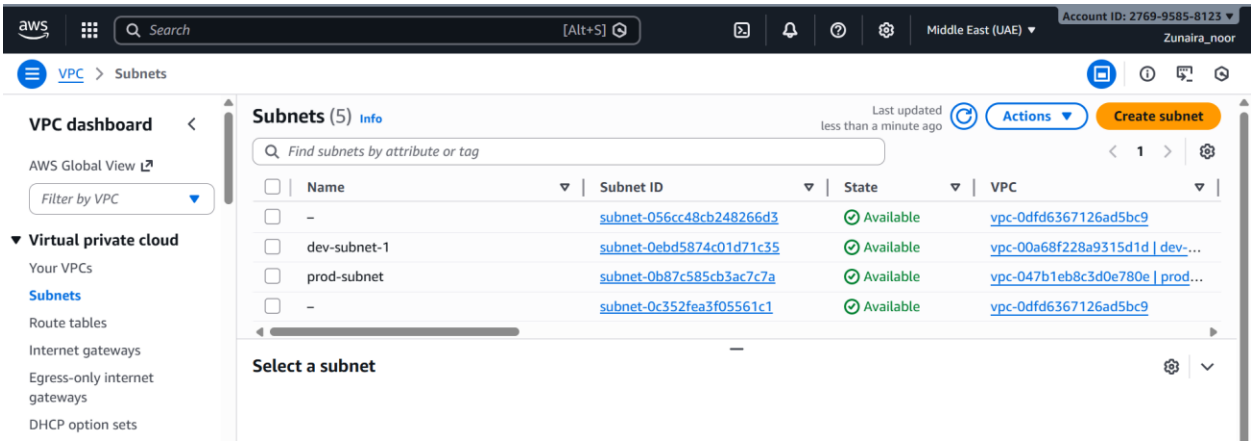


Figure 2b: Deployed Subnets

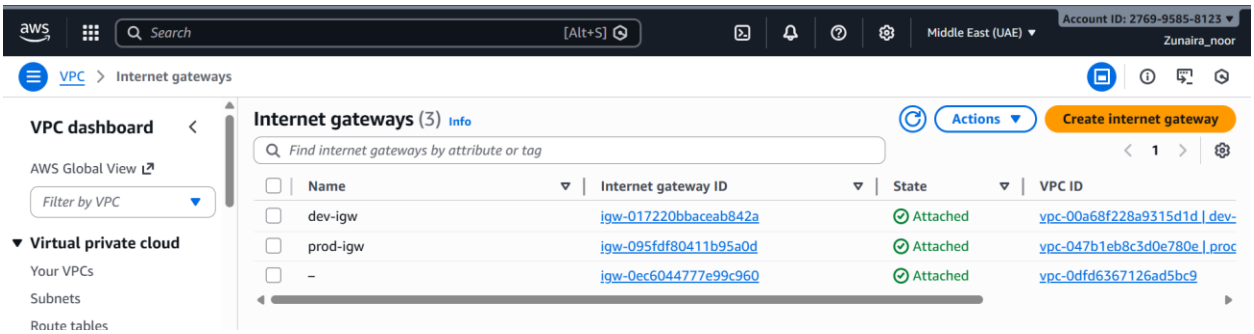


Figure 2c: Internet Gateway

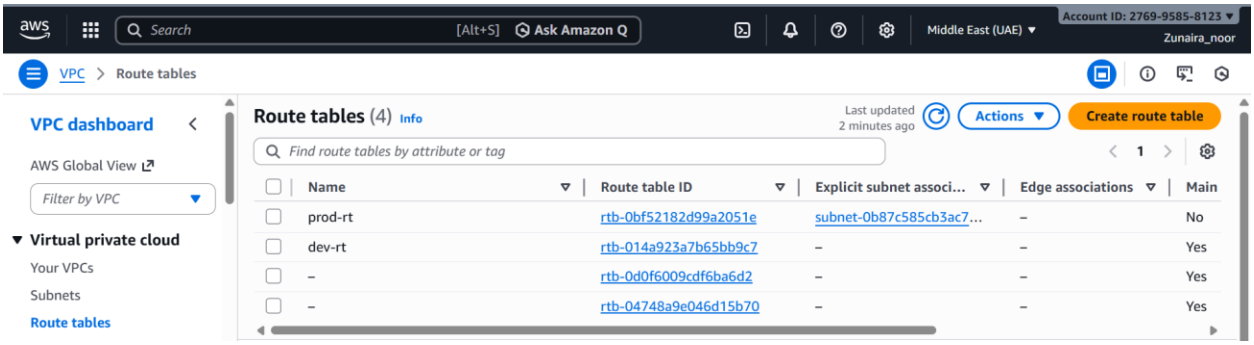


Figure 2d: Route Table

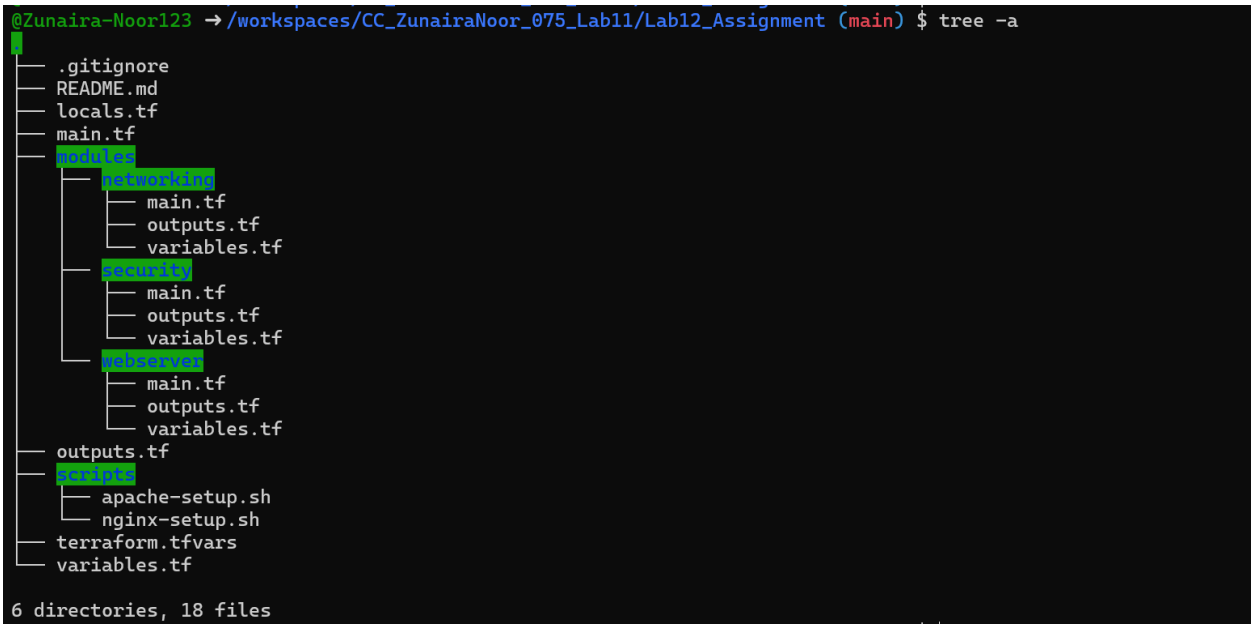
3. Implementation Details

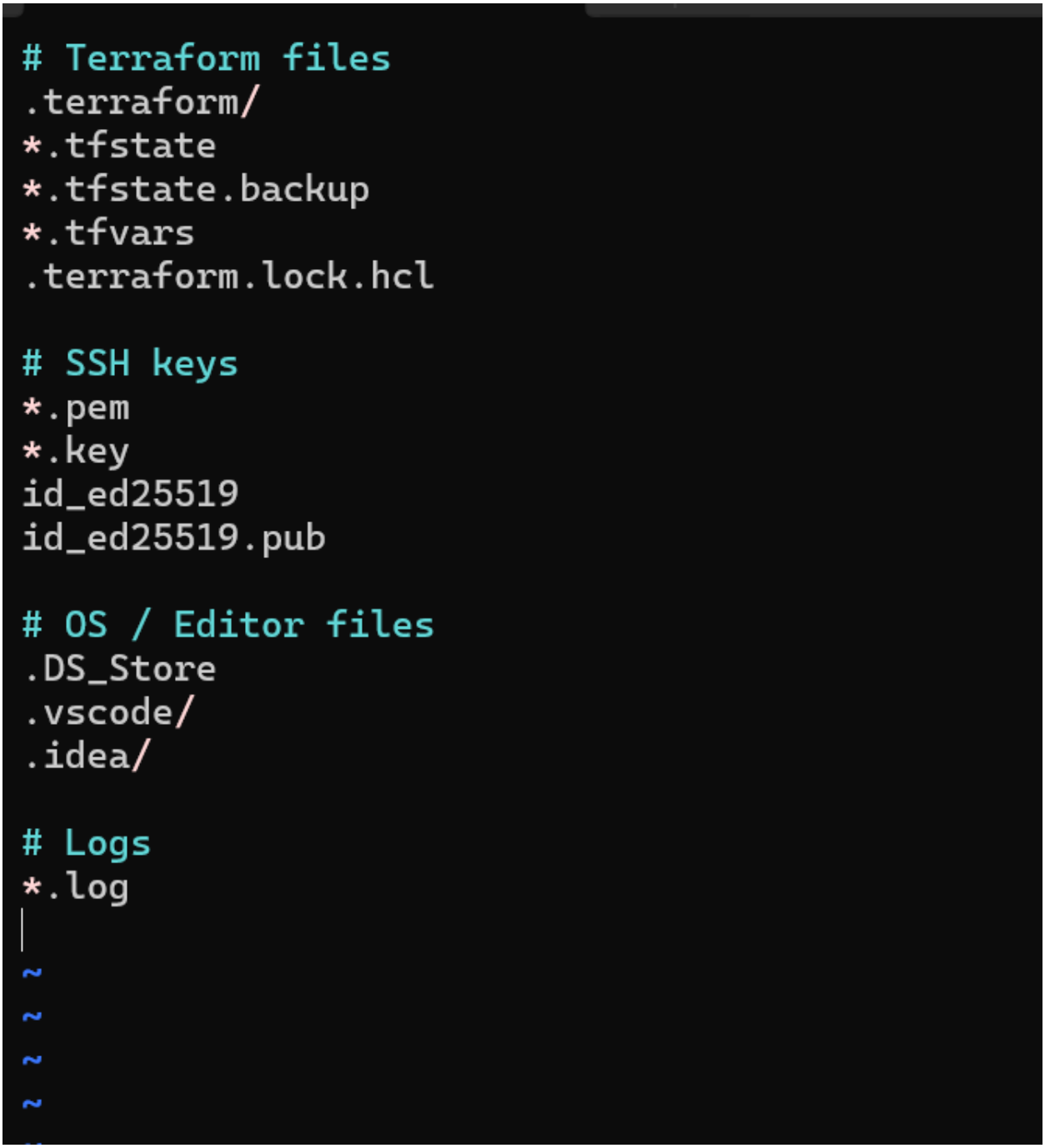
This section details the step-by-step deployment of the multi-tier web infrastructure on AWS using Terraform and server scripts.

Part 1: Infrastructure Setup

1.1 Project Structure

The Terraform project is organized into modules for networking, security, and web servers. Proper .gitignore prevents sensitive files from being committed.


@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11/Lab12_Assignment (main) $ tree -a
.
├── .gitignore
├── README.md
├── locals.tf
├── main.tf
├── modules
│   ├── networking
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── security
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   └── webserver
│   ├── main.tf
│   ├── outputs.tf
│   └── variables.tf
├── outputs.tf
├── scripts
│   ├── apache-setup.sh
│   └── nginx-setup.sh
├── terraform.tfvars
└── variables.tf
6 directories, 18 files



```
# Terraform files
.terraform/
*.tfstate
*.tfstate.backup
*.tfvars
.terraform.lock.hcl

# SSH keys
*.pem
*.key
id_ed25519
id_ed25519.pub

# OS / Editor files
.DS_Store
.vscode/
.idea/

# Logs
*.log
```

1.2 Variable Configuration

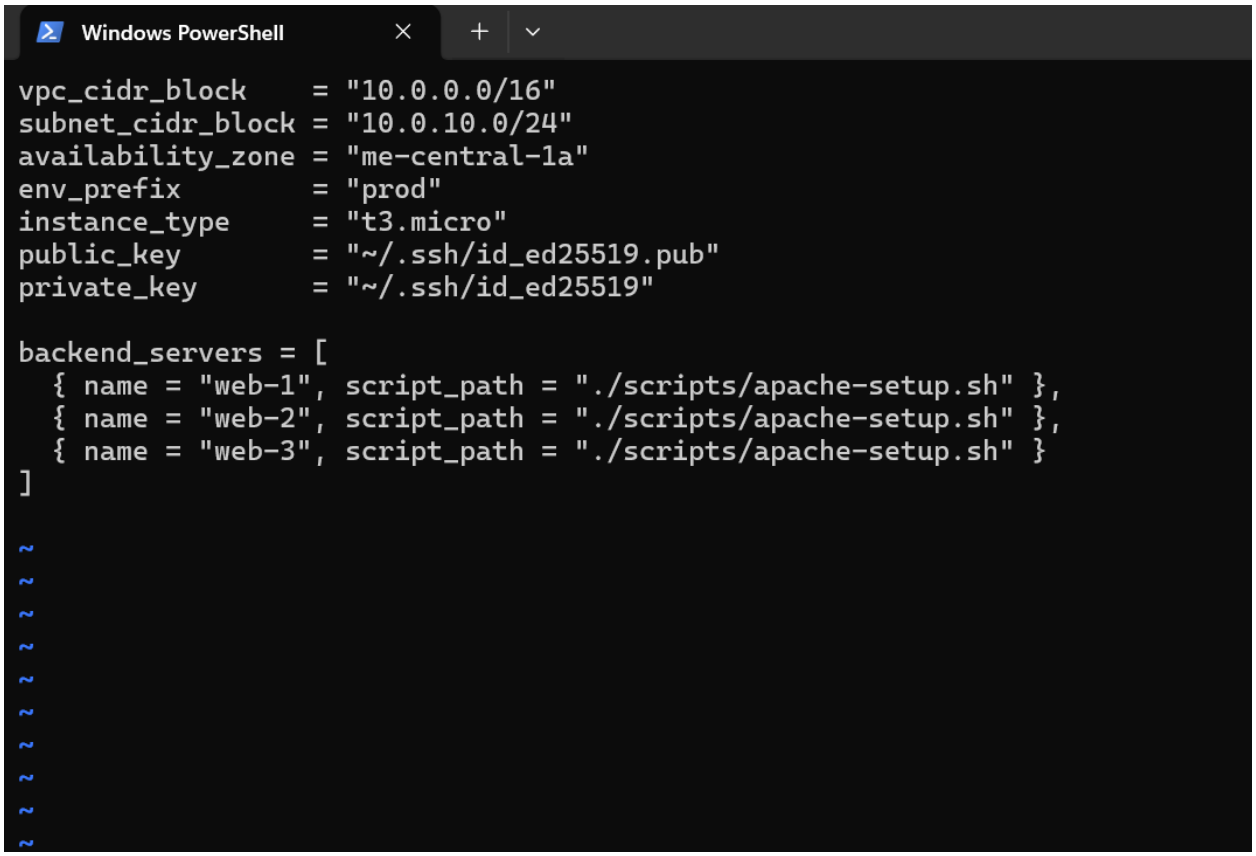
All required variables are defined in variables.tf with validation, descriptions, and defaults. Values are populated in terraform.tfvars.

1.2.1 variables.tf

```
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11/Lab12_Assignment (main) $ cat  variables.tf
# VPC CIDR block
variable "vpc_cidr_block" {
  description = "CIDR block for the VPC"
  type        = string
  default     = "10.0.0.0/16"
  validation {
    condition     = can(regex("^([0-9]{1,3}\\.){3}[0-9]{1,3}/[0-9]{1,2}$", var.vpc_cidr_block))
    error_message = "vpc_cidr_block must be a valid CIDR, e.g., 10.0.0.0/16"
  }
}

# Subnet CIDR block
variable "subnet_cidr_block" {
  description = "CIDR block for the subnet"
  type        = string
  default     = "10.0.10.0/24"
  validation {
    condition     = can(regex("^([0-9]{1,3}\\.){3}[0-9]{1,3}/[0-9]{1,2}$", var.subnet_cidr_block))
    error_message = "subnet_cidr_block must be a valid CIDR, e.g., 10.0.10.0/24"
  }
}

# Availability Zone
variable "availability_zone" {
  description = "AWS Availability Zone"
  type        = string
  default     = "me-central-1a"
}

# Environment prefix
variable "env_prefix" {
```

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

# SSH Public Key
variable "public_key" {
  description = "Path to the public SSH key"
  type        = string
  default     = "~/.ssh/id_ed25519.pub"
}

# SSH Private Key
variable "private_key" {
  description = "Path to the private SSH key"
  type        = string
  default     = "~/.ssh/id_ed25519"
}

# Backend servers
variable "backend_servers" {
  description = "List of backend servers with name and setup script path"
  type = list(object({
    name        = string
    script_path = string
  }))
  default = [
    { name = "web-1", script_path = "./scripts/apache-setup.sh" },
    { name = "web-2", script_path = "./scripts/apache-setup.sh" },
    { name = "web-3", script_path = "./scripts/apache-setup.sh" }
  ]
}
```

1.2.2 terraform.tfvars



1.3 Networking Module

The networking module provisions: VPC, subnet, Internet Gateway, and route table. Resources are tagged with env_prefix for organization

1.3.1 modules/networking/main.tf

```
  tags = {
    Name = "${var.env_prefix}-subnet"
  }
}

# Create Internet Gateway
resource "aws_internet_gateway" "this" {
  vpc_id = aws_vpc.this.id
  tags = {
    Name = "${var.env_prefix}-igw"
  }
}

# Create Route Table
resource "aws_route_table" "this" {
  vpc_id = aws_vpc.this.id
  tags = {
    Name = "${var.env_prefix}-rt"
  }

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

# Associate Route Table with Subnet
resource "aws_route_table_association" "this" {
  subnet_id      = aws_subnet.this.id
  route_table_id = aws_route_table.this.id
}

:wq
```

1.3.2 modules/networking/outputs.tf

```
output "vpc_id" {
  description = "ID of the VPC"
  value       = aws_vpc.this.id
}

output "subnet_id" {
  description = "ID of the Subnet"
  value       = aws_subnet.this.id
}

output "igw_id" {
  description = "ID of the Internet Gateway"
  value       = aws_internet_gateway.this.id
}

output "route_table_id" {
  description = "ID of the Route Table"
  value       = aws_route_table.this.id
}
~
~
~
~
~
~
~
~
```

1.4 Security Module

Two security groups are created:

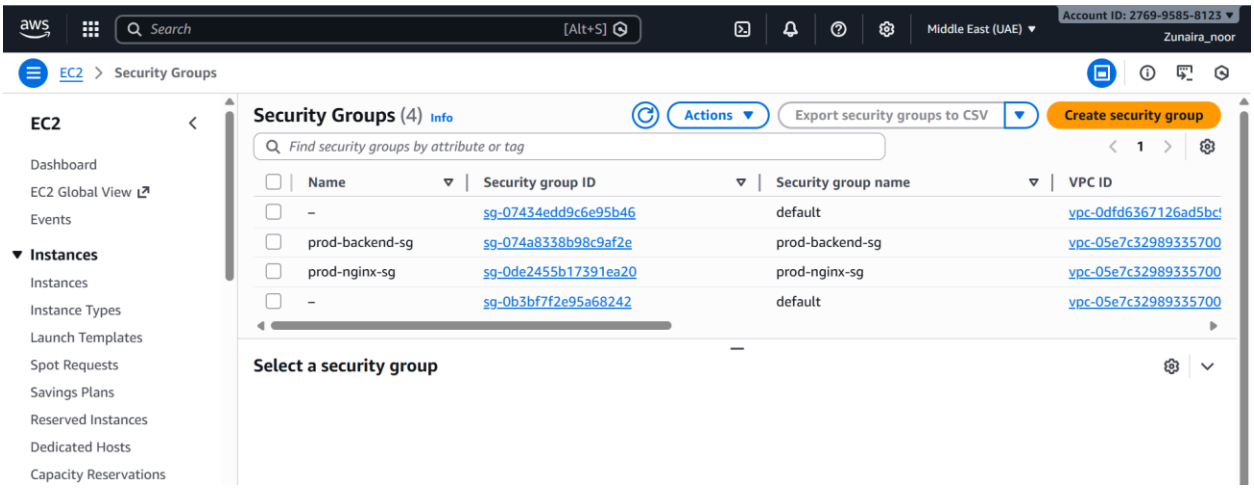
- **Nginx SG: Allows SSH from admin IP, HTTP/HTTPS from anywhere.**
- **Backend SG: Allows SSH from admin IP, HTTP only from Nginx SG.**

1.4.1 Two Security Groups:

```
# Nginx Security Group
resource "aws_security_group" "nginx_sg" {
  name        = "${var.env_prefix}-nginx-sg"
  description = "Security group for Nginx reverse proxy/load balancer"
  vpc_id      = var.vpc_id

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

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

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

  egress {
    from_port   = 0
```

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

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

  ingress {
    description = "HTTP from Nginx SG only"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    security_groups = [aws_security_group.nginx_sg.id]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
```

1.4.2 AWS Console Security Groups



1.5 Locals Configuration

locals.tf defines dynamic IP detection, resource naming, common tags, and backend server list.

```
locals {
  # Append /32 to your public IP for security group rules
  my_ip = "${chomp(data.http.my_ip.response_body)}/32"

  # Common tags for all resources
  common_tags = {
    Environment = var.env_prefix
    Project     = "Lab12-Assignment"
    ManagedBy   = "Terraform"
  }

  # Backend server configurations
  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"
    }
  ]
}
"locals.tf" 36L, 774B
```

Part 2: Webserver Module

2.1 Module Design

The webserver module provisions EC2 instances with key pairs, user data scripts, and proper tags. Variables include env_prefix, instance_name, security_group_id, and script_path.

2.1.1 Create main.tf

Windows PowerShell

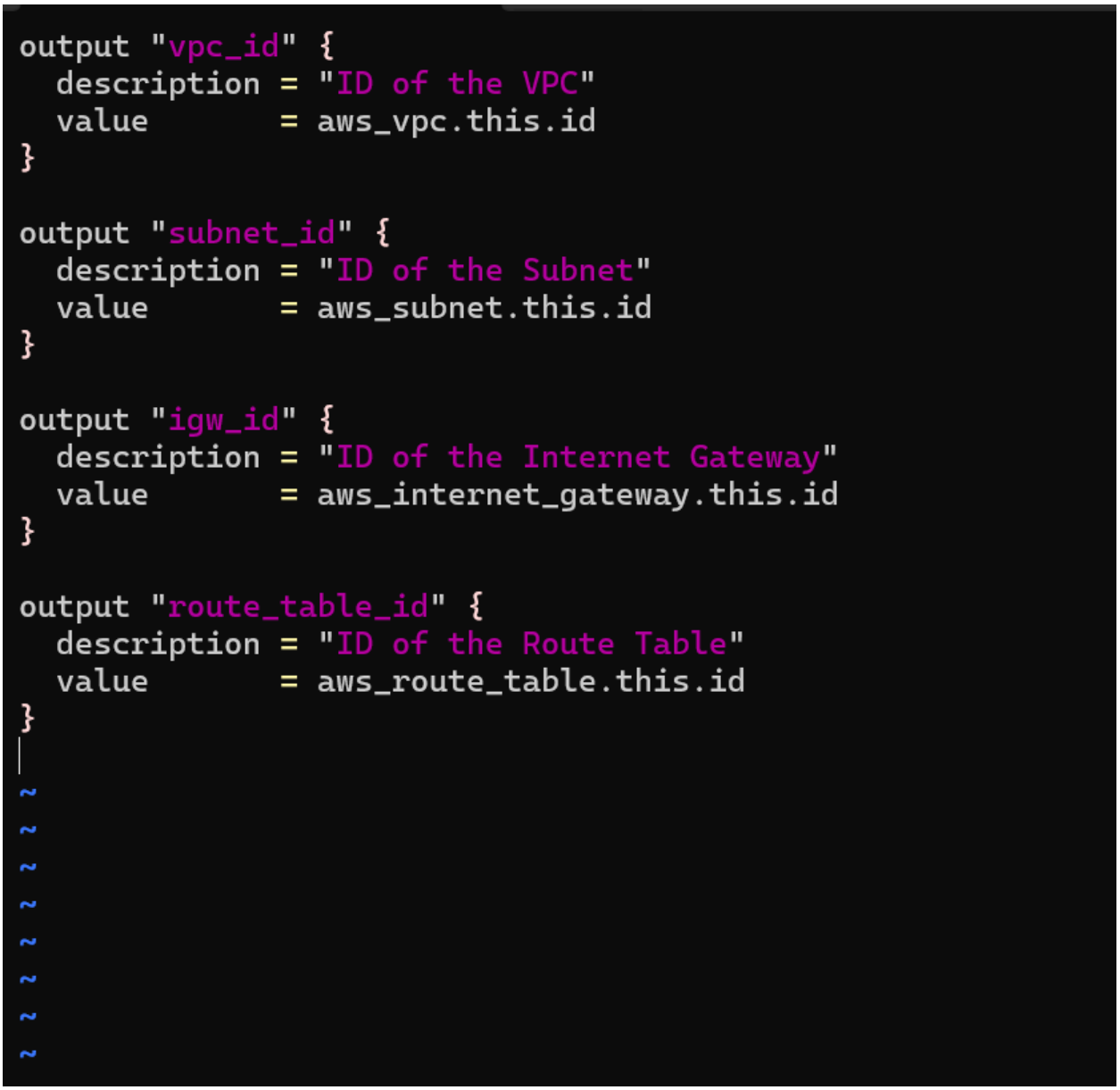
```
# Create Key Pair
resource "aws_key_pair" "this" {
  key_name   = "${var.env_prefix}-${var.instance_name}-${var.instance_suffix}"
  public_key = file(var.public_key)
  tags       = var.common_tags
}

# Create EC2 Instance
resource "aws_instance" "this" {
  ami                    = "ami-0c9fc9b90d9c91f5b" # Amazon Linux 2023 (update if needed)
  instance_type          = var.instance_type
  availability_zone      = var.availability_zone
  subnet_id              = var.subnet_id
  vpc_security_group_ids = [var.security_group_id]
  key_name               = aws_key_pair.this.key_name
  associate_public_ip_address = true

  user_data = file(var.script_path)

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

2.1.2 variables.tf

Windows PowerShell

```
variable "env_prefix" {
  type        = string
  description = "Environment prefix (e.g., prod, dev)"
}

variable "instance_name" {
  type        = string
  description = "Name of the EC2 instance"
}

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

variable "availability_zone" {
  type        = string
  description = "AWS Availability Zone"
}

variable "vpc_id" {
  type        = string
  description = "VPC ID where instance will be launched"
}

variable "subnet_id" {
  type        = string
  description = "Subnet ID for the instance"
}

variable "security_group_id" {
  type        = string
-- INSERT --
```

2.1.3 outputs.tf



```
output "vpc_id" {
  description = "ID of the VPC"
  value       = aws_vpc.this.id
}

output "subnet_id" {
  description = "ID of the Subnet"
  value       = aws_subnet.this.id
}

output "igw_id" {
  description = "ID of the Internet Gateway"
  value       = aws_internet_gateway.this.id
}

output "route_table_id" {
  description = "ID of the Route Table"
  value       = aws_route_table.this.id
}
```

2.2 Module Usage

Nginx Server: Module instantiated with nginx-setup.sh.

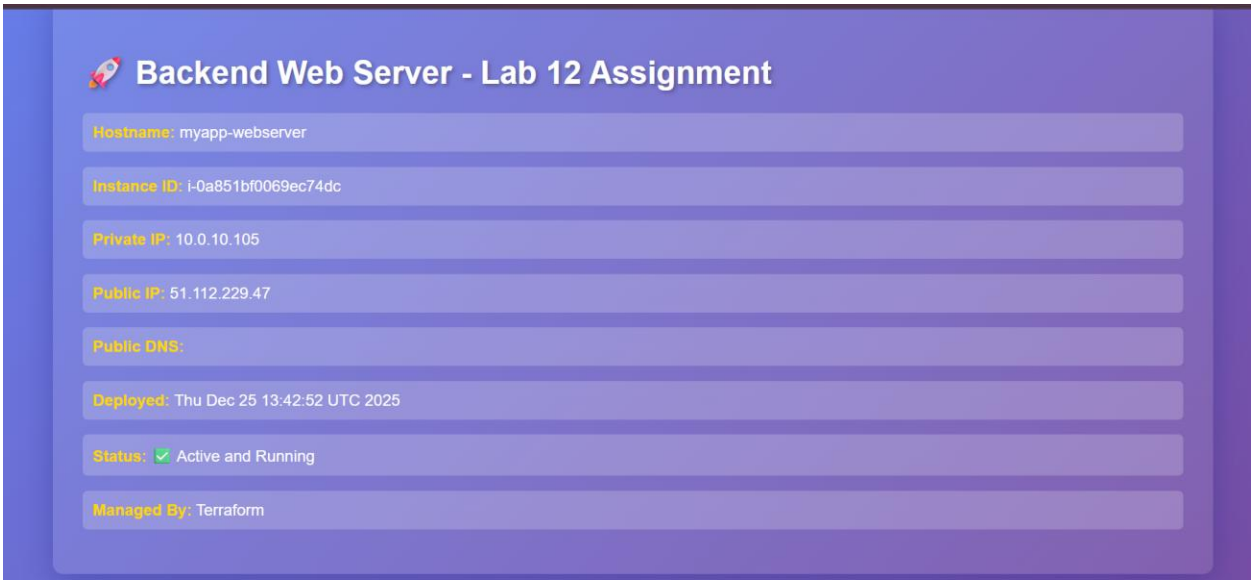
Backend Servers: Module instantiated with apache-setup.sh using for_each.

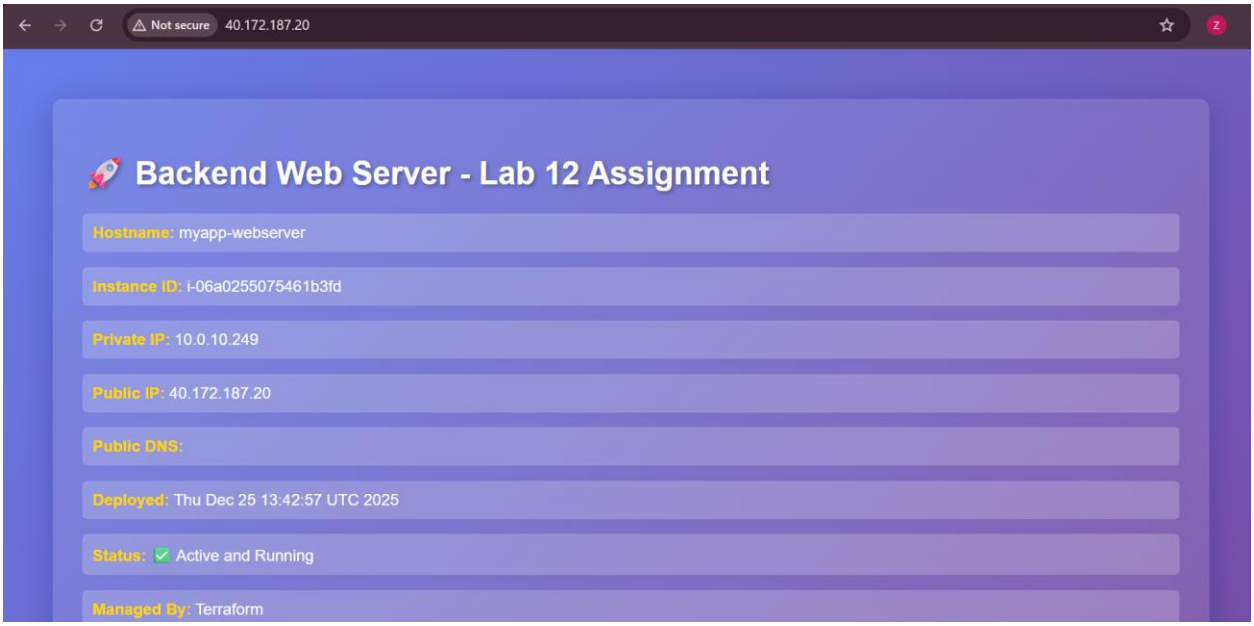
```
# Nginx Server
module "nginx_server" {
  source            = "./modules/webserver"
  env_prefix        = var.env_prefix
  instance_name     = "nginx-proxy"
  instance_type     = var.instance_type
  availability_zone = var.availability_zone
  vpc_id            = module.networking.vpc_id
  subnet_id         = module.networking.subnet_id
  security_group_id = module.security.nginx_sg_id
  public_key        = var.public_key
  script_path       = "./scripts/nginx-setup.sh"
  instance_suffix   = "nginx"
  common_tags       = local.common_tags
}

# Backend Servers (3)
module "backend_servers" {
  for_each = { for idx, server in local.backend_servers : server.name => server }

  source            = "./modules/webserver"
  env_prefix        = var.env_prefix
  instance_name     = each.value.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.backend_sg_id
  public_key        = var.public_key
  script_path       = each.value.script_path
  instance_suffix   = each.value.suffix
  common_tags       = local.common_tags
```

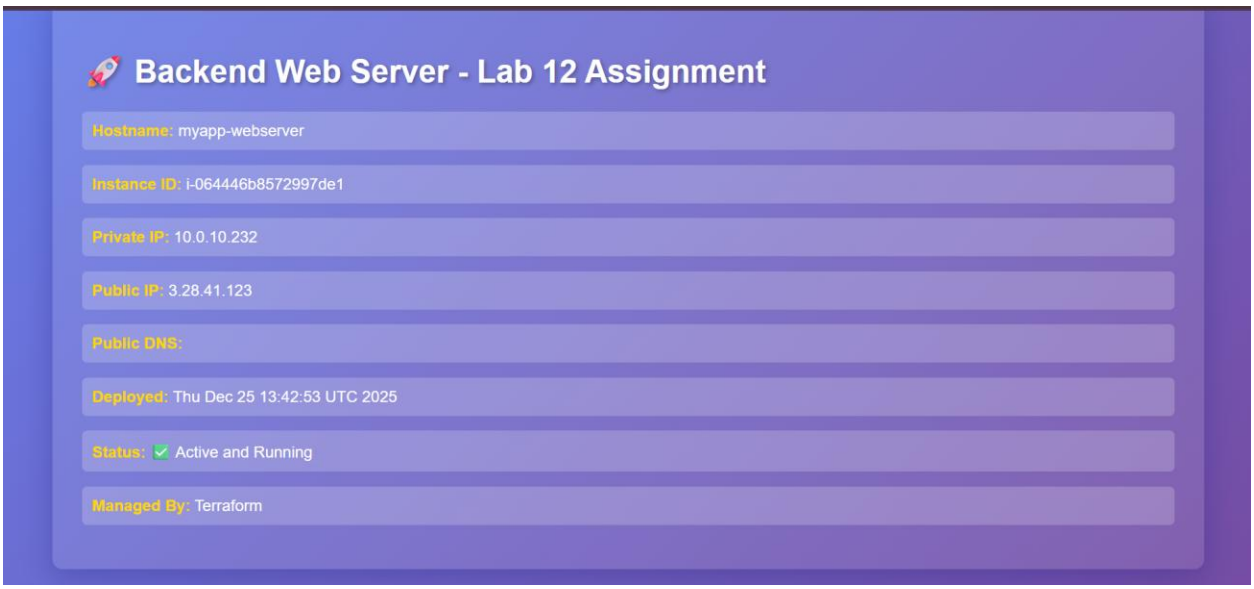
```
(and 3 more similar warnings elsewhere)

Apply complete! Resources: 8 added, 0 changed, 0 destroyed.

Outputs:

backend_sg_id = "sg-074a8338b98c9af2e"
nginx_sg_id = "sg-0de2455b17391ea20"
@Zunaira-Noor123 →/workspaces/CC_ZunairaNoor_075_Lab11/Lab12_Assignment (main) $
```

Part 3: Server Scripts

3.1 Apache Backend Script







3.2 Nginx Setup Script

scripts/nginx-setup.sh

```bash
#!/bin/bash
set -e

# ------------------------------
# Nginx Setup Script with SSL
# ------------------------------

# Update system packages and install Nginx + OpenSSL
yum update -y
yum install -y nginx openssl
systemctl start nginx
systemctl enable nginx

# Create SSL directories
mkdir -p /etc/ssl/private
mkdir -p /etc/ssl/certs

# Get metadata token for EC2
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)

# Generate self-signed certificate
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
  -keyout /etc/ssl/private/selfsigned.key \
  -out /etc/ssl/certs/selfsigned.crt \
  -subj "/CN=$PUBLIC_IP" \
  -addext "subjectAltName=IP:$PUBLIC_IP" \
  -addext "basicConstraints=CA:FALSE" \
```

```
< Last-Modified: Thu, 25 Dec 2025 13:42:52 GMT
< ETag: "603-646c6f40b5506"
< Accept-Ranges: bytes
<
<!DOCTYPE html>
<html>
<head>
    <title>Backend Web Server</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin:  50px;
            background:  linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
        }
        .container {
            background: rgba(255, 255, 255, 0.1);
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
        }
        h1 { color: #fff; text-shadow: 2px 2px 4px rgba(0,0,0,0.3); }
        .info { margin: 15px 0; padding: 10px; background: rgba(255,255,255,0.2); border-radius: 5px; }
        .label { font-weight: bold; color: #ffd700; }
    </style>
</head>
<body>
    <div class="container">
        <h1>🚀 Backend Web Server - Lab 12 Assignment</h1>
        <div class="info"><span class="label">Hostname:</span> myapp-webserver</div>
        <div class="info"><span class="label">Instance ID:</span> i-0a851bf0069ec74dc</div>
        <div class="info"><span class="label">Private IP:</span> 10.0.10.105</div>
        <div class="info"><span class="label">Public IP:</span> 51.112.229.47</div>
```

Part 4: Infrastructure Deployment

- **Initialize, validate, plan, and apply Terraform.**

- **SSH key pair generated if not existing.**
- **Generate SSH key pair (if not exists**

```
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11 (main) $ ls ~/.ssh/id_ed25519
/home/codespace/.ssh/id_ed25519
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11 (main) $ ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519 -N ""
Generating public/private ed25519 key pair.
/home/codespace/.ssh/id_ed25519 already exists.
Overwrite (y/n)? y
Your identification has been saved in /home/codespace/.ssh/id_ed25519
Your public key has been saved in /home/codespace/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:GVU2N5gRvnySFCcUyoBUbgAsrd33DKp4hv8taO4EmKE codespace@codespaces-5d58a1
The key's randomart image is:
+--[ED25519 256]--+
|   o.oooo .o@*+  |
|  . o .o + +o* . |
|.  + .  + o o    |
|.+. . ..oo o o   |
|E .    oS+  = .  |
|   .  .   o  o   |
|   o.o           |
|  o.* ..         |
|   B+....        |
+----[SHA256]-----+
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11 (main) $
```

Initialize Terraform:

```
+----[SHA256]-----+
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11 (main) $ terraform init
Initializing the backend...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.27.0

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.
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11 (main) $
```

Validate configuration:

```
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11 (main) $ terraform validate
Success! The configuration is valid.

@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11 (main) $
```

Plan deployment:

```
      ~ private_dns_name_options (known after apply)
      - private_dns_name_options {
          - enable_resource_name_dns_a_record    = false -> null
          - enable_resource_name_dns_aaaa_record = false -> null
          - hostname_type                        = "ip-name" -> null
        }

      ~ root_block_device (known after apply)
      - root_block_device {
          - delete_on_termination = true -> null
          - device_name           = "/dev/xvda" -> null
          - encrypted             = false -> null
          - iops                  = 3000 -> null
          - tags                  = {} -> null
          - tags_all              = {} -> null
          - throughput            = 125 -> null
          - volume_id             = "vol-0cba8b4f0ef4435da" -> null
          - volume_size           = 8 -> null
          - volume_type           = "gp3" -> null
            # (1 unchanged attribute hidden)
        }
    }

Plan: 1 to add, 0 to change, 1 to destroy.

Changes to Outputs:
  ~ aws_instance_public_ip = "3.29.244.20" -> (known after apply)

─────────────────────────────────────────────────────────────────────────────

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these actions if you run
"terraform apply" now.
```

Apply deployment:

```
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11 (main) $ terraform apply -auto-approve
aws_key_pair.ssh_key: Refreshing state... [id=serverkey]
aws_vpc.myapp_vpc: Refreshing state... [id=vpc-00a68f228a9315d1d]
aws_subnet.myapp_subnet_1: Refreshing state... [id=subnet-0ebd5874c01d71c35]
aws_internet_gateway.myapp_igw: Refreshing state... [id=igw-017220bbaceab842a]
aws_default_security_group.myapp_sg: Refreshing state... [id=sg-0f292584bc7ff53aa]
aws_default_route_table.main_rt: Refreshing state... [id=rtb-014a923a7b65bb9c7]
aws_instance.myapp-server: Refreshing state... [id=i-0444cd5f88c3369aa]

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following
symbols:
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_instance.myapp-server must be replaced
-/+ resource "aws_instance" "myapp-server" {
      ~ arn                                  = "arn:aws:ec2:me-central-1:276995858123:instance/i-0444cd5f88c3369aa" -> (known after a
pply)
      ~ disable_api_stop                     = false -> (known after apply)
      ~ disable_api_termination              = false -> (known after apply)
      ~ ebs_optimized                        = false -> (known after apply)
      + enable_primary_ipv6                  = (known after apply)
      - hibernation                          = false -> null
      + host_id                              = (known after apply)
      + host_resource_group_arn              = (known after apply)
      + iam_instance_profile                 = (known after apply)
      ~ id                                   = "i-0444cd5f88c3369aa" -> (known after apply)
      ~ instance_initiated_shutdown_behavior = "stop" -> (known after apply)
      + instance_lifecycle                   = (known after apply)
```

Part 5: Testing & Verification and Testing Results

5.1 Nginx Backend Configuration Update

Update Nginx Backend Configuration

```
    tcp_nopush on;
    tcp_nodelay on;
    keepalive_timeout 65;
    types_hash_max_size 4096;

    include /usr/share/nginx/modules/*.conf;
    include /etc/nginx/conf.d/*.conf;

    # Upstream backend servers (outside any server block)
    upstream backend_servers {
        server 10.0.10.105:80;
        server 10.0.10.249:80;
        server 10.0.10.232:80 backup;
    }
```

Test Nginx

```
[root@ip-10-0-10-185 ~]# sudo nginx -t
nginx: [warn] the "listen ... http2" directive is deprecated, use the "http2" directive instead in /etc/nginx/nginx.conf:56
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
[root@ip-10-0-10-185 ~]#
```

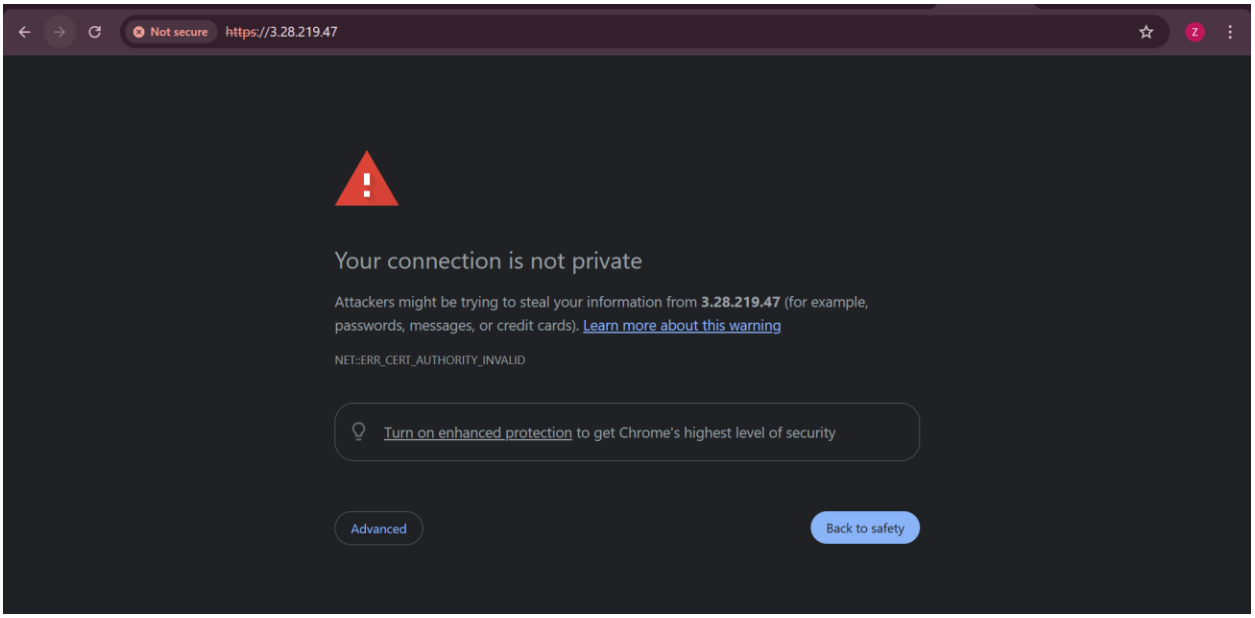
Restart Nginx

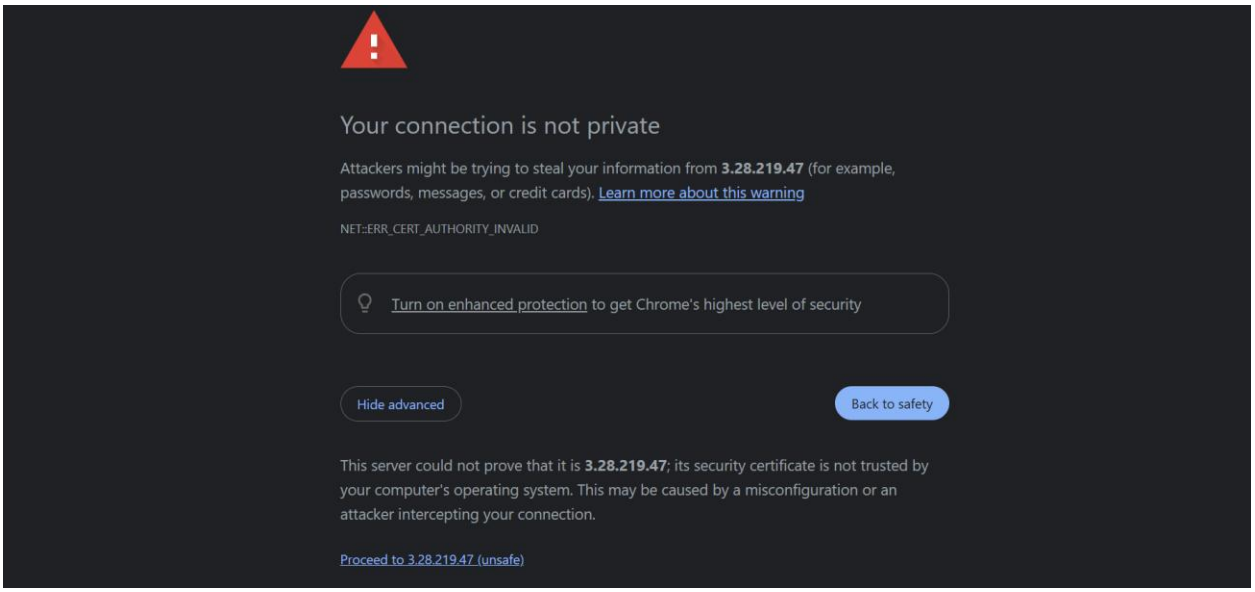
```
● nginx.service - The nginx HTTP and reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; vendor preset: disabled)
   Active: active (running) since Thu 2025-12-25 18:11:22 UTC; 1min 25s ago
  Process: 2502 ExecStart=/usr/sbin/nginx (code=exited, status=0/SUCCESS)
  Process: 2499 ExecStartPre=/usr/sbin/nginx -t (code=exited, status=0/SUCCESS)
  Process: 2497 ExecStartPre=/usr/bin/rm -f /run/nginx.pid (code=exited, status=0/SUCCESS)
 Main PID: 2504 (nginx)
   CGroup: /system.slice/nginx.service
           ├─2504 nginx: master process /usr/sbin/nginx
           ├─2505 nginx: worker process
           └─2506 nginx: worker process

Dec 25 18:11:22 ip-10-0-10-185.me-central-1.compute.internal systemd[1]: Starting The nginx HTTP and reverse proxy server...
Dec 25 18:11:22 ip-10-0-10-185.me-central-1.compute.internal nginx[2499]: nginx: [warn] the "listen ... http2" directive is deprecated, us...nf
Dec 25 18:11:22 ip-10-0-10-185.me-central-1.compute.internal nginx[2499]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
Dec 25 18:11:22 ip-10-0-10-185.me-central-1.compute.internal nginx[2499]: nginx: configuration file /etc/nginx/nginx.conf test is successful
Dec 25 18:11:22 ip-10-0-10-185.me-central-1.compute.internal nginx[2502]: nginx: [warn] the "listen ... http2" directive is deprecated, us...nf
Dec 25 18:11:22 ip-10-0-10-185.me-central-1.compute.internal systemd[1]: Started The nginx HTTP and reverse proxy server.
Hint: Some lines were ellipsized, use -l to show in full.
[root@ip-10-0-10-185 ~]#
```

5.2 Load Balancing Test

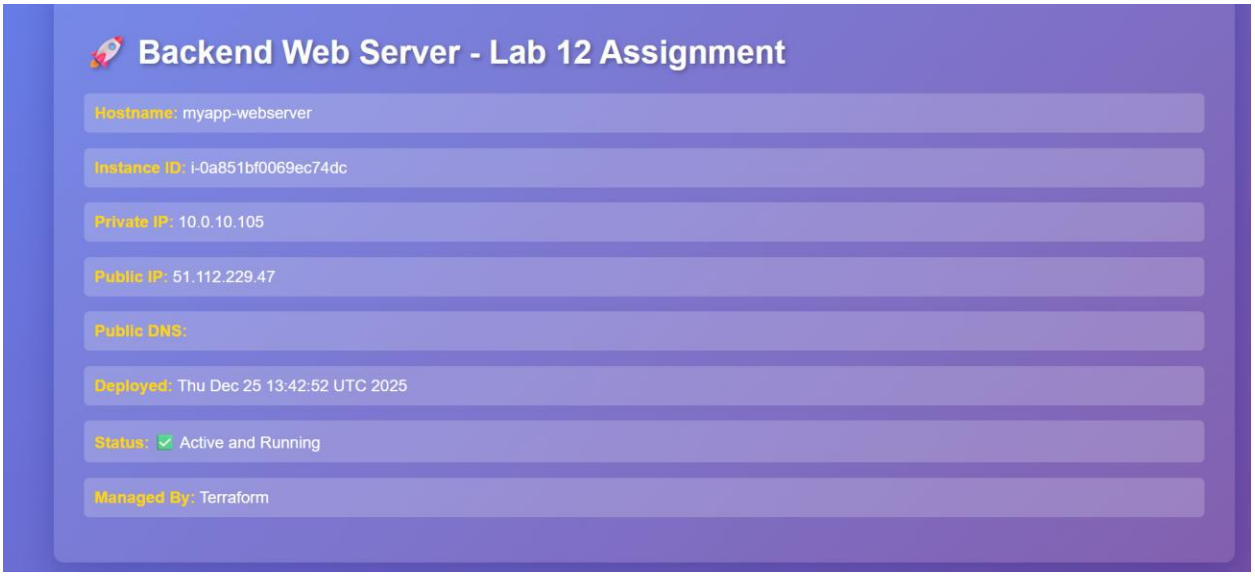
Verify traffic alternates between web-1 and web-2; backup server web-3 remains inactive unless primaries fail.

browser security warning

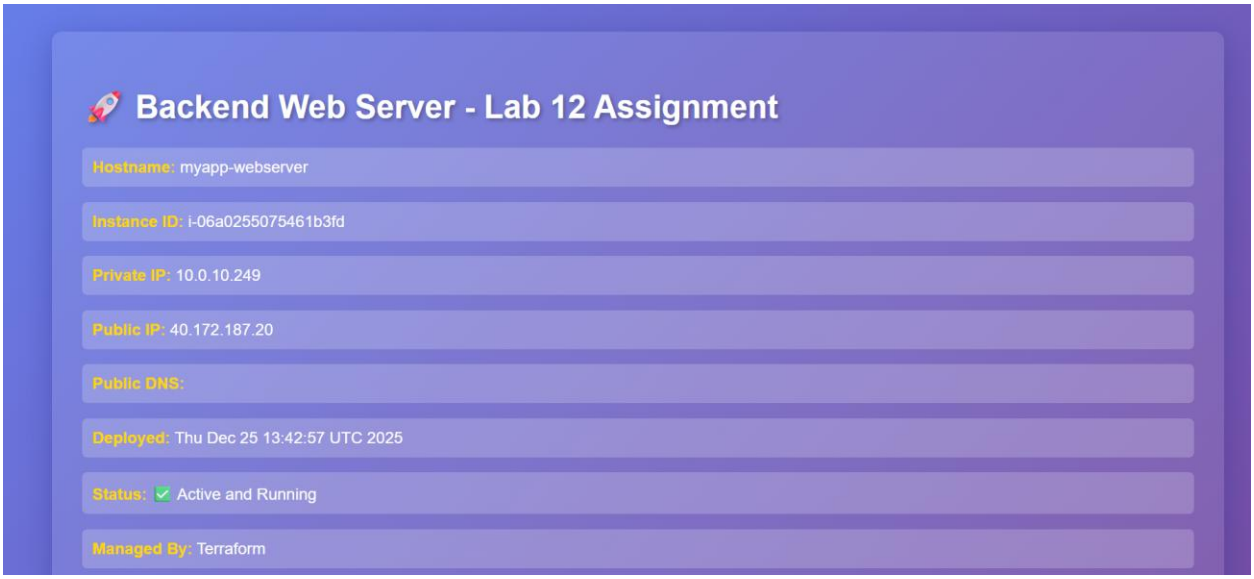




showing web-1 content



showing web-2 content



5.3 Cache Functionality

Check Nginx cache: first request MISS, subsequent HIT.

first request - MISS

```
[root@ip-10-0-10-185 ~]# curl -k -I https://localhost
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 19:50:09 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1539
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Thu, 25 Dec 2025 13:42:52 GMT
ETag: "603-646c6f40b5506"
X-Cache-Status: MISS
Accept-Ranges: bytes
```

second request - HIT

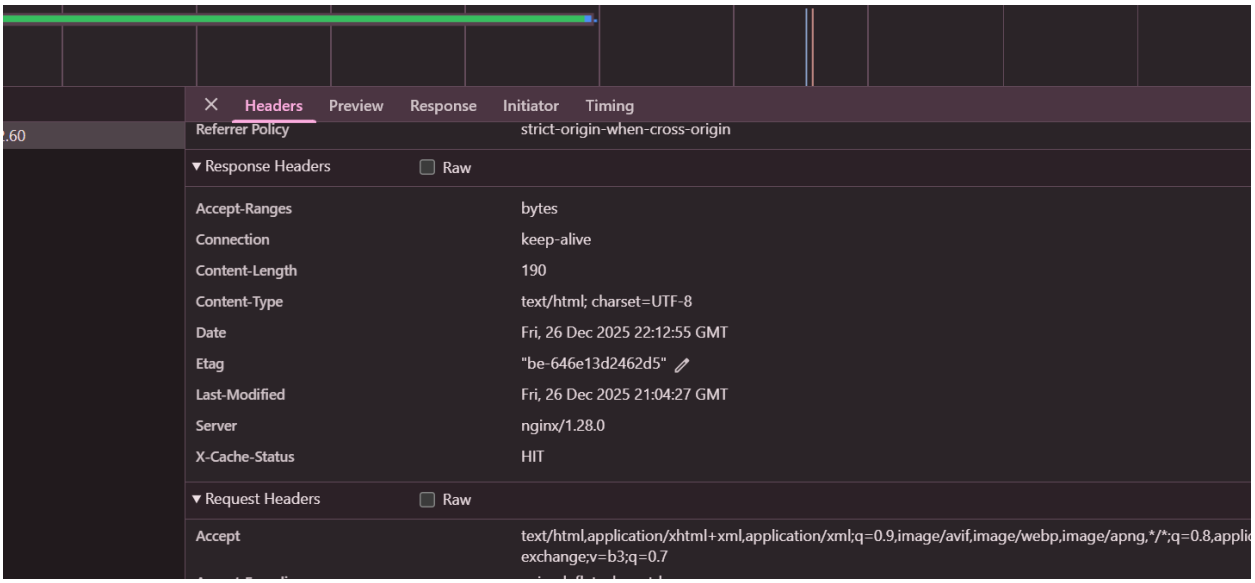
```
[root@ip-10-0-10-185 ~]# curl -k -I https://localhost
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 19:50:15 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1539
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Thu, 25 Dec 2025 13:42:52 GMT
ETag: "603-646c6f40b5506"
X-Cache-Status: HIT
Accept-Ranges: bytes
```

cache folder contents

```
[root@ip-10-0-10-185 ~]# ls -la /var/cache/nginx/
total 0
drwx------ 3 nginx root  15 Dec 25 19:50 .
drwxr-xr-x 7 root  root  76 Dec 25 19:46 ..
drwx------ 3 nginx nginx 16 Dec 25 19:50 f
[root@ip-10-0-10-185 ~]#
```

access log showing cache status

```
drwxr-xr-x 7 root  root  76 Dec 25 19:46 ..
drwx------ 3 nginx nginx 16 Dec 25 19:50 f
[root@ip-10-0-10-185 ~]# sudo tail -f /var/log/nginx/access.log
23.249.17.76 - - [25/Dec/2025:19:09:47 +0000] "POST / HTTP/1.1" 301 169 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537
like Gecko) Chrome/120.0.0.0 Safari/537.36" "-"
23.249.17.76 - - [25/Dec/2025:19:09:48 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537
like Gecko) Chrome/120.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:19:24:26 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
 like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:19:24:27 +0000] "GET /favicon.ico HTTP/1.1" 404 196 "https://3.28.219.47/" "Mozilla/5.0 (Windows NT 10.
4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:19:27:27 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
 like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:19:27:28 +0000] "GET /favicon.ico HTTP/1.1" 404 196 "https://3.28.219.47/" "Mozilla/5.0 (Windows NT 10.
4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
173.255.221.189 - - [25/Dec/2025:19:33:12 +0000] "GET / HTTP/1.1" 400 255 "-" "Mozilla/5.0 zgrab/0.x" "-"
204.76.203.219 - - [25/Dec/2025:19:34:55 +0000] "GET / HTTP/1.1" 301 169 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/53
 like Gecko) Chrome/90.0.4430.85 Safari/537.36 Edg/90.0.818.46" "-"
127.0.0.1 - - [25/Dec/2025:19:50:09 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/8.3.0" "-"
127.0.0.1 - - [25/Dec/2025:19:50:15 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/8.3.0" "-"
```

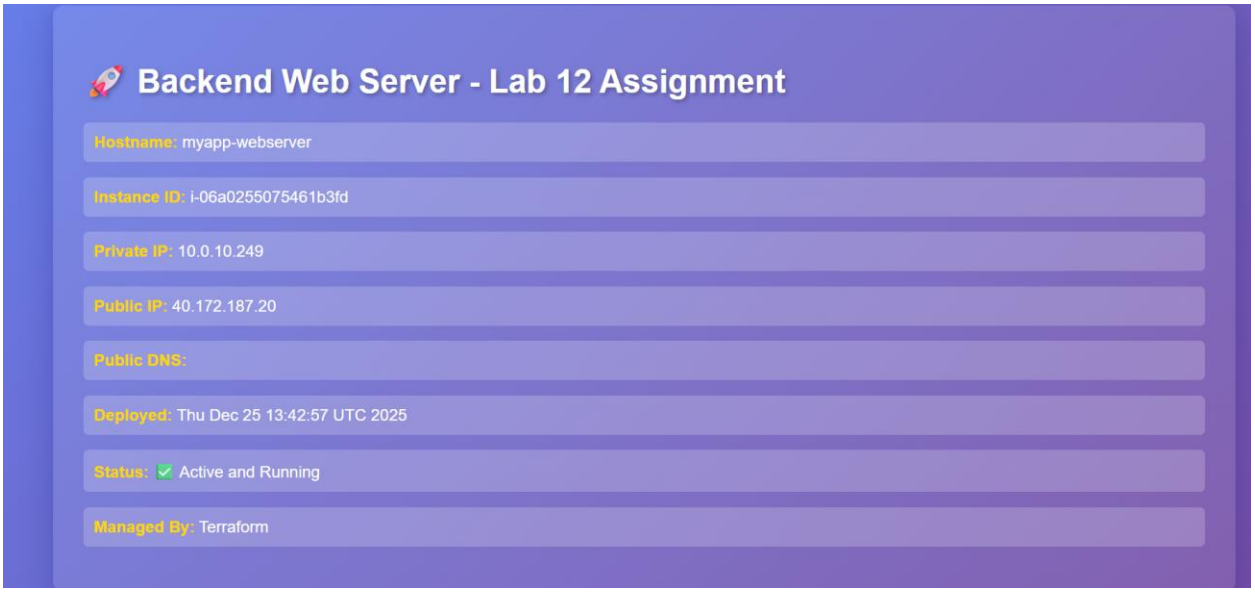


5.4 High Availability Test

Simulate primary server failure and verify backup server activation.

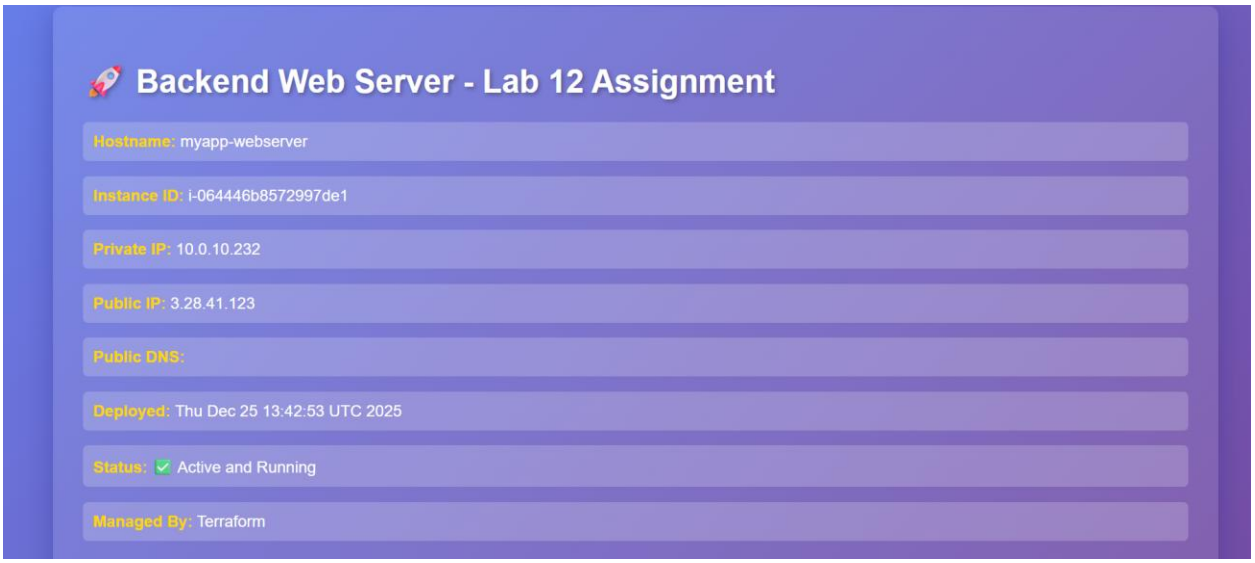
SSH (Session Manager) into web-1 and stop Apache:

```
[root@myapp-webserver ~]#  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 Thu 2025-12-25 21:11:11 UTC; 2min 30s ago
     Docs: man:httpd.service(8)
  Process: 16118 ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND (code=exited, status=0/SUCCESS)
 Main PID: 16118 (code=exited, status=0/SUCCESS)
   Status: "Total requests: 18; Idle/Busy workers 100/0;Requests/sec: 0.165; Bytes served/sec: 427 B/sec"

Dec 25 21:09:16 myapp-webserver systemd[1]: Starting The Apache HTTP Server...
Dec 25 21:09:16 myapp-webserver httpd[16118]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name, u
Dec 25 21:09:16 myapp-webserver systemd[1]: Started The Apache HTTP Server.
Dec 25 21:11:10 myapp-webserver systemd[1]: Stopping The Apache HTTP Server...
Dec 25 21:11:11 myapp-webserver systemd[1]: Stopped The Apache HTTP Server.
Hint: Some lines were ellipsized, use -l to show in full.
[root@myapp-webserver ~]#
```



Stop web-2 Apache:

```
sh-4.2$ sudo -i
[root@myapp-webserver ~]# sudo systemctl stop httpd
[root@myapp-webserver ~]# 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 Thu 2025-12-25 21:09:58 UTC; 22s ago
     Docs: man:httpd.service(8)
  Process: 5834 ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND (code=exited, status=0/SUCCESS)
 Main PID: 5834 (code=exited, status=0/SUCCESS)
   Status: "Total requests: 32; Idle/Busy workers 100/0;Requests/sec: 0.00119; Bytes served/sec:   2 B/sec"

Dec 25 13:42:57 ip-10-0-10-249.me-central-1.compute.internal systemd[1]: Starting The Apache HTTP Server...
Dec 25 13:42:57 ip-10-0-10-249.me-central-1.compute.internal systemd[1]: Started The Apache HTTP Server.
Dec 25 21:09:57 myapp-webserver systemd[1]: Stopping The Apache HTTP Server...
Dec 25 21:09:58 myapp-webserver systemd[1]: Stopped The Apache HTTP Server.
[root@myapp-webserver ~]#
```



Check Nginx error logs

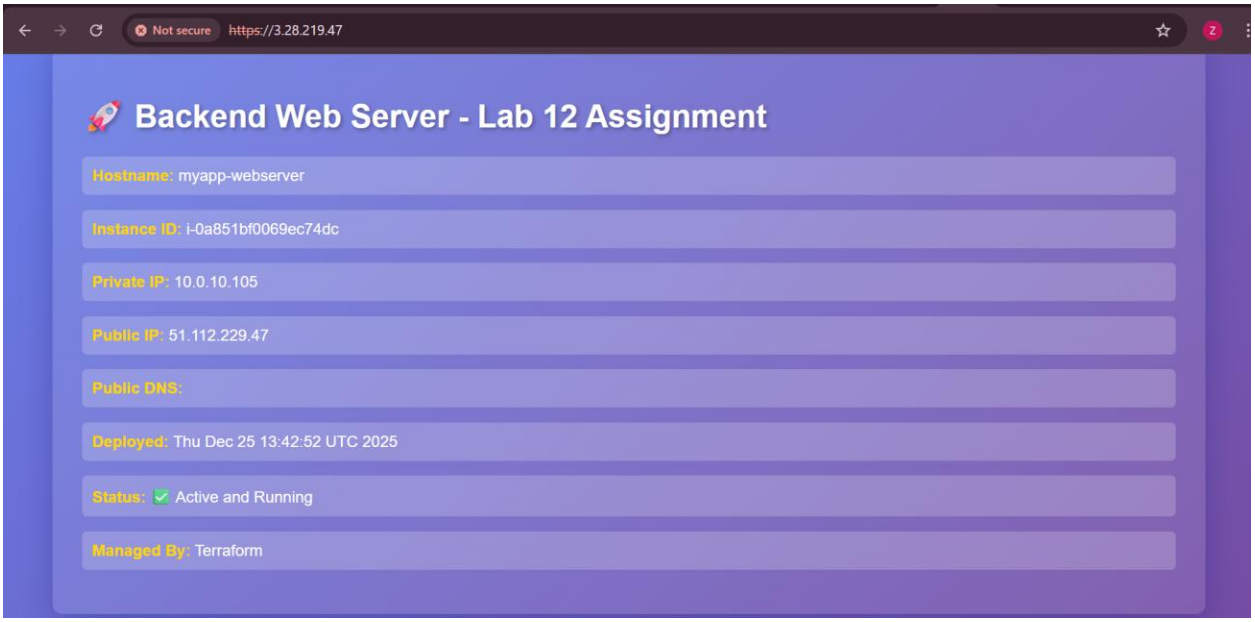
```
[root@ip-10-0-10-185 ~]# sudo tail -f /var/log/nginx/error.log
2025/12/25 21:10:30 [error] 2899#2899: *31 connect() failed (111: Connection refused) while connecting to upstream, client: 103.53.162.15, serve
 _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.249:80/", host: "3.28.219.47"
2025/12/25 21:10:30 [warn] 2899#2899: *31 upstream server temporarily disabled while connecting to upstream, client: 103.53.162.15, server: _, r
uest: "GET / HTTP/1.1", upstream: "http://10.0.10.249:80/", host: "3.28.219.47"
2025/12/25 21:11:15 [error] 2899#2899: *31 connect() failed (111: Connection refused) while connecting to upstream, client: 103.53.162.15, serve
 _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.105:80/", host: "3.28.219.47"
2025/12/25 21:11:15 [warn] 2899#2899: *31 upstream server temporarily disabled while connecting to upstream, client: 103.53.162.15, server: _, r
uest: "GET / HTTP/1.1", upstream: "http://10.0.10.105:80/", host: "3.28.219.47"
2025/12/25 21:11:15 [error] 2899#2899: *31 connect() failed (111: Connection refused) while connecting to upstream, client: 103.53.162.15, serve
 _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.249:80/", host: "3.28.219.47"
2025/12/25 21:11:15 [warn] 2899#2899: *31 upstream server temporarily disabled while connecting to upstream, client: 103.53.162.15, server: _, r
uest: "GET / HTTP/1.1", upstream: "http://10.0.10.249:80/", host: "3.28.219.47"
2025/12/25 21:12:28 [error] 2900#2900: *76 connect() failed (111: Connection refused) while connecting to upstream, client: 103.53.162.15, serve
 _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.249:80/", host: "3.28.219.47"
2025/12/25 21:12:28 [warn] 2900#2900: *76 upstream server temporarily disabled while connecting to upstream, client: 103.53.162.15, server: _, r
uest: "GET / HTTP/1.1", upstream: "http://10.0.10.249:80/", host: "3.28.219.47"
2025/12/25 21:12:28 [error] 2900#2900: *76 connect() failed (111: Connection refused) while connecting to upstream, client: 103.53.162.15, serve
 _, request: "GET / HTTP/1.1", upstream: "http://10.0.10.105:80/", host: "3.28.219.47"
2025/12/25 21:12:28 [warn] 2900#2900: *76 upstream server temporarily disabled while connecting to upstream, client: 103.53.162.15, server: _, r
```

Restart web-1 and web-2

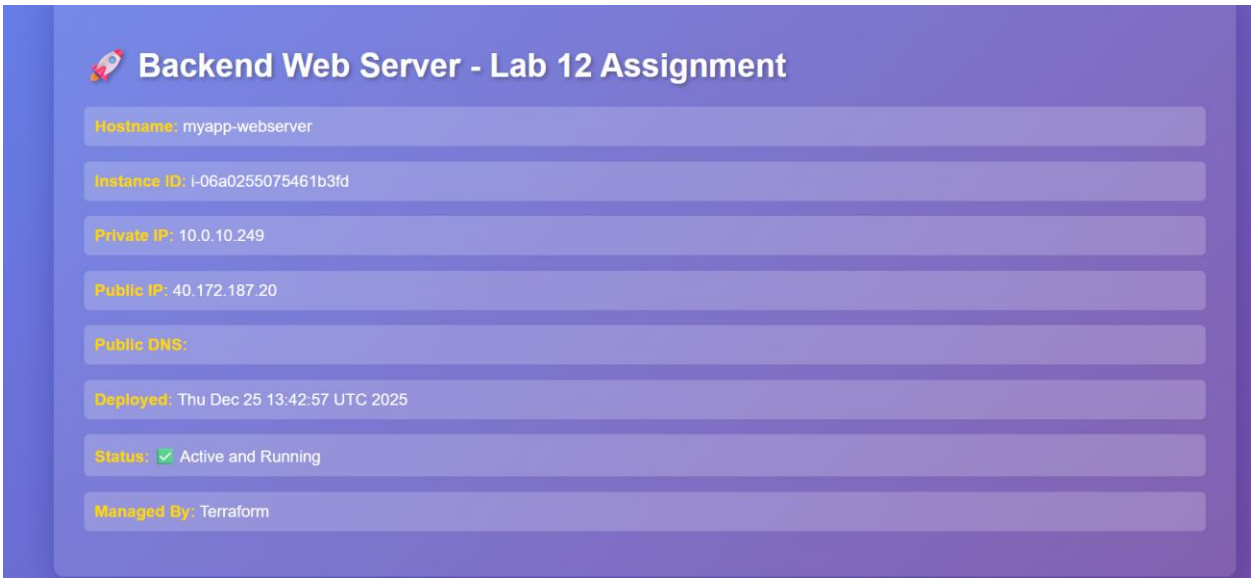
```
[root@ip-10-0-10-185 ~]# curl -k https://3.28.219.47 --http1.1
<!DOCTYPE html>
<html>
<head>
    <title>Backend Web Server</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin:  50px;
            background:  linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
        }
        .container {
            background: rgba(255, 255, 255, 0.1);
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
        }
        h1 { color: #fff; text-shadow: 2px 2px 4px rgba(0,0,0,0.3); }
        .info { margin: 15px 0; padding: 10px; background: rgba(255,255,255,0.2); border-radius: 5px; }
        .label { font-weight: bold; color: #ffd700; }
    </style>
</head>
<body>
    <div class="container">
        <h1>🚀Backend Web Server - Lab 12 Assignment</h1>
        <div class="info"><span class="label">Hostname:</span> myapp-webserver</div>
        <div class="info"><span class="label">Instance ID:</span> i-064446b8572997de1</div>
        <div class="info"><span class="label">Private IP:</span> 10.0.10.232</div>
```

Verify traffic returns to web-1 and web-2

web-1



web-2



5.5 Security & Performance Analysis

Verify SSL, security headers, HTTP→HTTPS redirect, and Nginx logs.

Check SSL/TLS certificate details

```
[root@ip-10-0-10-185 ~]# openssl s_client -connect 3.28.219.47:443 -showcerts
CONNECTED(00000003)
depth=0 CN = localhost
verify error:num=18:self signed certificate
verify return:1
depth=0 CN = localhost
verify return:1
---
Certificate chain
 0 s:/CN=localhost
   i:/CN=localhost
-----BEGIN CERTIFICATE-----
MIIC+zCCAeOgAwIBAgIJAKj6tjDPKiigMA0GCSqGSIb3DQEBCwUAMBQxEjAQBgNV
BAMMCWxvY2FsaG9zdDAeFw0yNTEyMjUxNjE5MDZaFw0yNjEyMjUxNjE5MDZaMBQx
EjAQBgNVBAMMCWxvY2FsaG9zdDCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoC
ggEBANzl96ecfeG0P3GvRAM1npqTqaJ5O0g8wwVK0RMTdy0/b/+f/Zxm3+rPQv8z
Fcshm5XWOAyhQqc0iNeWNkAEUhXbjfamG8rUOEXX54xgo+1qYiv6QdxNKXfDDRBy
zRQDUPvl9qZaIgZuP4I8XLePG+Rb2tjktni/6UNUHJ4BUS1zo02ZMySy+PltMhK9
yAKDCMShYK3FI6ws62H3pN1kAVMMff8w4z4sBb+exUJ/HEoVNd0GePVKw+nv3e4v
xzjMIrV/vI3MX0C/TbepoyM/cPLUxFiY/99KTCYhpb2MtqVJHjut/cUrBTTV6AFt
Ya6/Ex3hEHQ7kJ7ilWzDYNEAaQECAwEAAaNQME4wHQYDVR0OBBYEFIqbeT70tKor
LnVFa56m5FD1ADdBMB8GA1UdIwQYMBaAFIqbeT70tKorLnVFa56m5FD1ADdBMAwG
A1UdEwQFMAMBAf8wDQYJKoZIhvcNAQELBQADggEBAHylS8ism+oJ6fSMOixGZJoo
5L3wwOGDTf5wp6fQO8qUU6YXmkZMLbnAE/4VIRzJr5N0dz6cY2wQOkGG/uuqcpF0
tyaQCBEo2kTnTi2K8eZTMODVXeJ3JLsEE3O1mi4s23Bdz0Dpn+irIDIFJLwl6jY+
hUTZJVj7yIX5T/f0MfeCSz0unNnTPJS6GWlL8zNVym62T4cd2ZDcVtrzgpKHZURi
bSekcpajO3NJiGQ+sMXjh33X9GdZ9KNkOuQ9tXp22JbzXrn9WkP0DbseyRyAcL7w
CbtmYBTHOdBVJfDQMsk7UtzgFiVGEyxhQ7oySAoNXFSueXBIWNI8sW+bfD+jy5s=
-----END CERTIFICATE-----
```

View certificate on Nginx server

```
[root@ip-10-0-10-185 ~]# sudo openssl x509 -in /etc/ssl/certs/selfsigned.crt -text -noout
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            a8:fa:b6:30:cf:2a:28:a0
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: CN=localhost
        Validity
            Not Before: Dec 25 16:19:06 2025 GMT
            Not After : Dec 25 16:19:06 2026 GMT
        Subject: CN=localhost
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                Public-Key: (2048 bit)
                Modulus:
                    00:dc:e5:f7:a7:9c:7d:e1:b4:3f:71:af:44:03:35:
                    9e:9a:93:a9:a2:79:3b:48:3c:c3:05:4a:d1:13:13:
                    77:2d:3f:6f:ff:9f:fd:9c:66:df:ea:cf:42:ff:33:
                    15:cb:21:9b:95:d6:38:0c:a1:42:a7:34:88:d7:96:
                    36:40:04:52:15:db:8d:f6:a6:1b:ca:d4:38:45:d7:
                    e7:8c:60:a3:ed:6a:62:2b:fa:41:dc:4d:29:77:c3:
                    0d:10:72:cd:14:03:50:fb:f5:f6:a6:5a:22:06:6e:
                    3f:82:3c:5c:b7:8f:1b:e4:5b:da:d8:e4:b6:78:bf:
                    e9:43:54:1c:9e:01:51:2d:73:a3:4d:99:33:24:b2:
                    f8:f9:6d:32:12:bd:c8:02:83:08:c4:a1:60:ad:c5:
                    23:ac:2c:eb:61:f7:a4:dd:64:01:53:0c:7d:ff:30:
                    e3:3e:2c:05:bf:9e:c5:42:7f:1c:4a:15:35:dd:06:
```

Check security headers

```
[root@ip-10-0-10-185 ~]# curl -I -k https://3.28.219.47
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 21:34:10 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1539
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Thu, 25 Dec 2025 13:42:52 GMT
ETag: "603-646c6f40b5506"
X-Cache-Status: MISS
Accept-Ranges: bytes
```

Test HTTP to HTTPS redirect

```
[root@ip-10-0-10-185 ~]# curl -I http://3.28.219.47
HTTP/1.1 301 Moved Permanently
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 21:35:13 GMT
Content-Type: text/html
Content-Length: 169
Connection: keep-alive
Location: https://3.28.219.47/
```

Step 5: Analyze Nginx logs

sudo tail -50 /var/log/nginx/error.log

```
[root@ip-10-0-10-185 ~]# sudo tail -50 /var/log/nginx/error.log
2025/12/25 18:45:08 [notice] 2504#2504: exit
2025/12/25 18:45:08 [notice] 2630#2630: using the "epoll" event method
2025/12/25 18:45:08 [notice] 2630#2630: nginx/1.28.0
2025/12/25 18:45:08 [notice] 2630#2630: built by gcc 7.3.1 20180712 (Red Hat 7.3.1-17) (GCC)
2025/12/25 18:45:08 [notice] 2630#2630: OS: Linux 5.10.228-219.884.amzn2.x86_64
2025/12/25 18:45:08 [notice] 2630#2630: getrlimit(RLIMIT_NOFILE): 65535:65535
2025/12/25 18:45:08 [notice] 2632#2632: start worker processes
2025/12/25 18:45:08 [notice] 2632#2632: start worker process 2633
2025/12/25 18:45:08 [notice] 2632#2632: start worker process 2634
2025/12/25 19:36:19 [emerg] 2824#2824: "proxy_cache" zone "my_cache" is unknown in /etc/nginx/nginx.conf:89
2025/12/25 19:40:44 [emerg] 2836#2836: "proxy_cache_path" directive is not allowed here in /etc/nginx/nginx.conf:50
2025/12/25 19:47:05 [notice] 2632#2632: signal 3 (SIGQUIT) received from 1, shutting down
2025/12/25 19:47:05 [notice] 2633#2633: gracefully shutting down
2025/12/25 19:47:05 [notice] 2633#2633: exiting
2025/12/25 19:47:05 [notice] 2634#2634: gracefully shutting down
2025/12/25 19:47:05 [notice] 2634#2634: exiting
2025/12/25 19:47:05 [notice] 2633#2633: exit
2025/12/25 19:47:05 [notice] 2634#2634: exit
2025/12/25 19:47:05 [notice] 2632#2632: signal 17 (SIGCHLD) received from 2633
2025/12/25 19:47:05 [notice] 2632#2632: worker process 2633 exited with code 0
2025/12/25 19:47:05 [notice] 2632#2632: worker process 2634 exited with code 0
2025/12/25 19:47:05 [notice] 2632#2632: exit
2025/12/25 19:47:05 [notice] 2896#2896: using the "epoll" event method
2025/12/25 19:47:05 [notice] 2896#2896: nginx/1.28.0
2025/12/25 19:47:05 [notice] 2896#2896: built by gcc 7.3.1 20180712 (Red Hat 7.3.1-17) (GCC)
2025/12/25 19:47:05 [notice] 2896#2896: OS: Linux 5.10.228-219.884.amzn2.x86_64
2025/12/25 19:47:05 [notice] 2896#2896: getrlimit(RLIMIT_NOFILE): 65535:65535
2025/12/25 19:47:05 [notice] 2898#2898: start worker processes
```

sudo tail -50 /var/log/nginx/access.log

```
[root@ip-10-0-10-185 ~]# sudo tail -50 /var/log/nginx/access.log
103.53.162.15 - - [25/Dec/2025:19:27:28 +0000] "GET /favicon.ico HTTP/1.1" 404 196 "https://3.28.219.47/" "Mozilla/5.0 (Windows NT 10.0; Win64; x
4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
173.255.221.189 - - [25/Dec/2025:19:33:12 +0000] "GET / HTTP/1.1" 400 255 "-" "Mozilla/5.0 zgrab/0.x" "-"
204.76.203.219 - - [25/Dec/2025:19:34:55 +0000] "GET / HTTP/1.1" 301 169 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTMl
 like Gecko) Chrome/90.0.4430.85 Safari/537.36 Edg/90.0.818.46" "-"
127.0.0.1 - - [25/Dec/2025:19:50:09 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/8.3.0" "-"
127.0.0.1 - - [25/Dec/2025:19:50:15 +0000] "HEAD / HTTP/1.1" 200 0 "-" "curl/8.3.0" "-"
101.32.192.203 - - [25/Dec/2025:20:35:01 +0000] "HEAD /Core/Skin/Login.aspx HTTP/1.1" 301 0 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleW
bKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36" "-"
101.32.192.203 - - [25/Dec/2025:20:35:02 +0000] "HEAD /Core/Skin/Login.aspx HTTP/1.1" 404 0 "http://3.28.219.47:80/Core/Skin/Login.aspx" "Mozilla
5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36" "-"
110.38.199.135 - - [25/Dec/2025:20:35:21 +0000] "POST /GponForm/diag_Form?images/ HTTP/1.1" 301 169 "-" "Hello, World" "-"
110.38.199.135 - - [25/Dec/2025:20:35:22 +0000] ";sh+/tmp/gpon80&ipv=0" 400 157 "-" "-" "-"
103.53.162.15 - - [25/Dec/2025:21:03:07 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTMl
 like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:21:03:11 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTMl
 like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:21:03:13 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTMl
 like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:21:08:55 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTMl
 like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:21:08:56 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTMl
 like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:21:08:57 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTMl
 like Gecko) Chrome/143.0.0.0 Safari/537.36" "-"
103.53.162.15 - - [25/Dec/2025:21:10:29 +0000] "GET / HTTP/1.1" 200 1539 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTMl
```

Step 6: Check Nginx worker processes

```
3.28.219.47 - - [25/Dec/2025:21:35:13 +0000] "HEAD / HTTP/1.1" 301 0 "-" "curl/8.3.0" "-"
[root@ip-10-0-10-185 ~]# ps aux | grep nginx
root        2898  0.0  0.1  49052  1076 ?        Ss   19:47   0:00 nginx: master process /usr/sbin/nginx
nginx       2899  0.0  0.6  49604  5976 ?        S    19:47   0:00 nginx: worker process
nginx       2900  0.0  0.6  49604  5976 ?        S    19:47   0:00 nginx: worker process
nginx       2901  0.0  0.2  49264  2592 ?        S    19:47   0:00 nginx: cache manager process
root        3287  0.0  0.8 237824  7712 pts/0    S+   21:24   0:00 sudo tail -f /var/log/nginx/error.log
root        3289  0.0  0.0 114676   792 pts/0    S+   21:24   0:00 tail -f /var/log/nginx/error.log
root        3390  0.0  0.0 119420   908 pts/1    S+   21:40   0:00 grep --color=auto nginx
[root@ip-10-0-10-185 ~]#
```

Bonus Tasks (10 marks extra credit)

Bonus 1: Custom Error Pages (3 marks)

Create custom error pages for Nginx (404, 502, 503).

Verify custom error pages

1. **404 Test:**

```
bash: nginx public ip: No such file or directory
[root@ip-10-0-10-185 ~]# curl -k https://curl -k https://3.28.219.47/nonexistentpage
curl: (6) Could not resolve host: curl
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<p>The requested URL was not found on this server.</p>
</body></html>
[root@ip-10-0-10-185 ~]#
```

Access Nginx:

```
[root@ip-10-0-10-185 ~]# curl -k https://3.28.219.47
<!DOCTYPE html>
<html>
<head>
    <title>Backend Web Server</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin:  50px;
            background:  linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
        }
        .container {
            background: rgba(255, 255, 255, 0.1);
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
        }
        h1 { color: #fff; text-shadow: 2px 2px 4px rgba(0,0,0,0.3); }
        .info { margin: 15px 0; padding: 10px; background: rgba(255,255,255,0.2); border-radius: 5px; }
        .label { font-weight: bold; color: #ffd700; }
    </style>
</head>
<body>
    <div class="container">
        <h1>🚀Backend Web Server - Lab 12 Assignment</h1>
        <div class="info"><span class="label">Hostname:</span> myapp-webserver</div>
        <div class="info"><span class="label">Instance ID:</span> i-06a0255075461b3fd</div>
        <div class="info"><span class="label">Private IP:</span> 10.0.10.248</div>
```

Bonus 2: Implement Rate Limiting (3 marks)

Add rate limiting to prevent abuse.

Verify Rate Limiting Configuration

```
        add_header X-Cache-Status $upstream_cache_status;
        limit_req zone=mylimit burst=20 nodelay;
    }
    error_page 404 /errors/404.html;
    error_page 502 /errors/502.html;
    error_page 503 /errors/503.html;
    location /errors/ {
    internal;
```

Test Rate Limiting

```
[root@ip-10-0-10-185 ~]# for i in {1..30}; do curl -i -k https://3.28.219.47; done
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 22:32:20 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1539
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Thu, 25 Dec 2025 13:42:52 GMT
ETag: "603-646c6f40b5506"
X-Cache-Status: EXPIRED
Accept-Ranges: bytes

<!DOCTYPE html>
<html>
<head>
    <title>Backend Web Server</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin:  50px;
            background:  linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
        }
        .container {
            background: rgba(255, 255, 255, 0.1);
            padding: 30px;
```

```
        <div class="info"><span class="label">Managed By:</span> Terraform</div>
    </div>
</body>
</html>
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 22:32:20 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1539
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Thu, 25 Dec 2025 13:42:52 GMT
ETag: "603-646c6f40b5506"
X-Cache-Status: HIT
Accept-Ranges: bytes

<!DOCTYPE html>
<html>
<head>
    <title>Backend Web Server</title>
    <style>
```

```
HTTP/1.1 200 OK
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 22:32:20 GMT
Content-Type: text/html; charset=UTF-8
Content-Length: 1539
Connection: keep-alive
Upgrade: h2,h2c
Last-Modified: Thu, 25 Dec 2025 13:42:52 GMT
ETag: "603-646c6f40b5506"
X-Cache-Status: HIT
Accept-Ranges: bytes

<!DOCTYPE html>
<html>
<head>
    <title>Backend Web Server</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin:  50px;
            background:  linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            color: white;
        }
        .container {
            background: rgba(255, 255, 255, 0.1);
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0 8px 32px 0 rgba(31, 38, 135, 0.37);
```

```
HTTP/1.1 404 Not Found
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 22:32:21 GMT
Content-Type: text/html
Content-Length: 153
Connection: keep-alive

<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.28.0</center>
</body>
</html>
HTTP/1.1 404 Not Found
Server: nginx/1.28.0
Date: Thu, 25 Dec 2025 22:32:21 GMT
Content-Type: text/html
Content-Length: 153
Connection: keep-alive

<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx/1.28.0</center>
</body>
</html>
[root@ip-10-0-10-185 ~]#
```

Bonus 3: Health Check Automation (4 marks)

Create a shell script that monitors backend server health.

Script Screenshot → open the script file:

```
[root@ip-10-0-10-185 ~]# sudo vim /usr/local/bin/health_check.sh
```

2. **Log Screenshot → show log output:**

```
[root@ip-10-0-10-185 ~]# cat /var/log/backend_health.log
2025-12-25 22:39:11 - 10.0.10.105 is UP
2025-12-25 22:39:11 - 10.0.10.249 is UP
2025-12-25 22:39:11 - 10.0.10.232 is UP
[root@ip-10-0-10-185 ~]#
```

Part 6: Cleanup

```
        ========================================
        DEPLOYMENT SUCCESSFUL!
        ========================================

        Next Steps:
        1. SSH into Nginx server: ssh ec2-user@3.28.219.47
        2. Edit Nginx config: sudo vim /etc/nginx/nginx.conf
        3. Update backend IPs in upstream block:
           - BACKEND_IP_1: 10.0.10.105
           - BACKEND_IP_2: 10.0.10.249
           - BACKEND_IP_3: 10.0.10.232
        4.  Restart Nginx: sudo systemctl restart nginx
        5. Test: https://3.28.219.47

        Backend Servers:
        - web-1: 51.112.229.47 (private: 10.0.10.105)
            - web-2: 40.172.187.20 (private: 10.0.10.249)
            - web-3: 3.28.41.123 (private: 10.0.10.232)

        ========================================
    EOT -> null
  - nginx_instance_id           = "i-03b1d01e2ee9b4c2f" -> null
  - nginx_private_ip            = "10.0.10.185" -> null
  - nginx_public_ip             = "3.28.219.47" -> null
  - nginx_sg_id                 = "sg-038da4178bc17d410" -> null
  - subnet_id                   = "subnet-0b87c585cb3ac7c7a" -> null
  - vpc_id                      = "vpc-047b1eb8c3d0e780e" -> null

Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

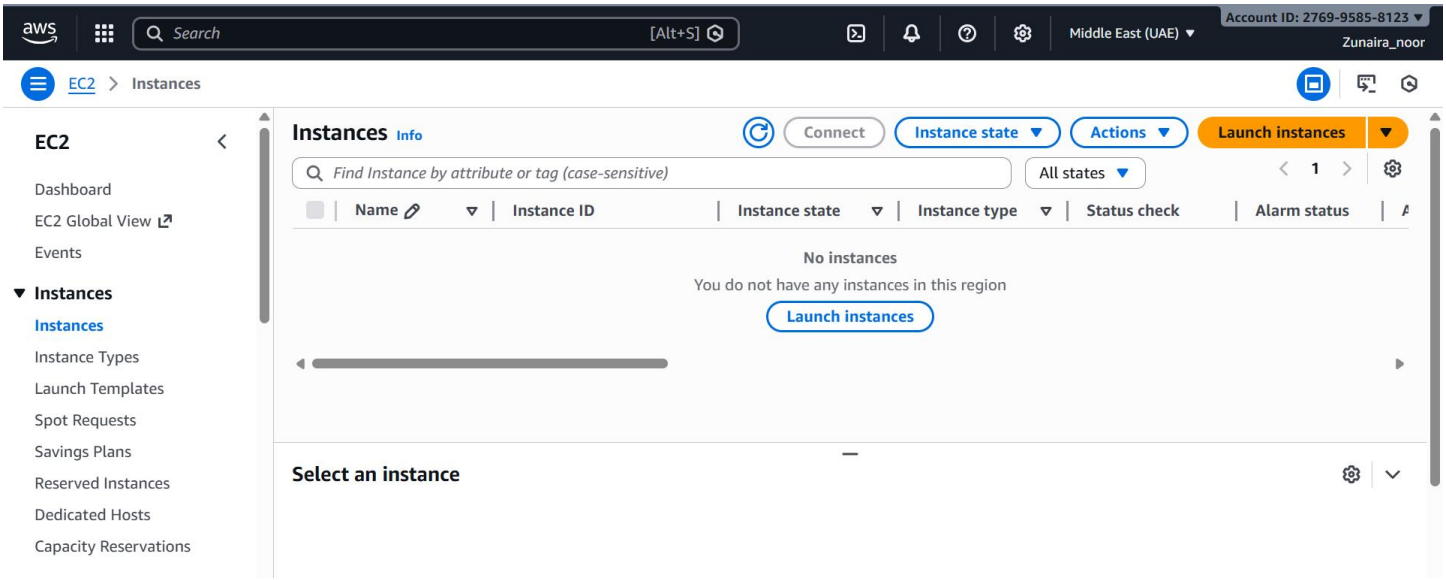
Step 2: Confirm Completion

```
module.backend_servers["web-1"].aws_instance.this: Still destroying... [id=i-0a851bf0069ec74dc, 0
module.backend_servers["web-3"].aws_instance.this: Still destroying... [id=i-064446b8572997de1, 0
module.backend_servers["web-2"].aws_instance.this: Still destroying... [id=i-06a0255075461b3fd, 0
module.backend_servers["web-3"].aws_instance.this: Destruction complete after 50s
module.backend_servers["web-3"].aws_key_pair.this: Destroying... [id=prod-web-3-3]
module.web_1.aws_instance.this: Destruction complete after 50s
module.web_1.aws_key_pair.this: Destroying... [id=prod-backend-1]
module.backend_servers["web-3"].aws_key_pair.this: Destruction complete after 0s
module.web_1.aws_key_pair.this: Destruction complete after 0s
module.networking.aws_internet_gateway.this: Still destroying... [id=igw-095fdf80411b95a0d, 00m50
module.networking.aws_internet_gateway.this: Destruction complete after 57s
module.backend_servers["web-2"].aws_instance.this: Still destroying... [id=i-06a0255075461b3fd, 0
module.backend_servers["web-1"].aws_instance.this: Still destroying... [id=i-0a851bf0069ec74dc, 0
module.backend_servers["web-2"].aws_instance.this: Destruction complete after 1m0s
module.backend_servers["web-2"].aws_key_pair.this: Destroying... [id=prod-web-2-2]
module.backend_servers["web-1"].aws_instance.this: Destruction complete after 1m0s
module.backend_servers["web-1"].aws_key_pair.this: Destroying... [id=prod-web-1-1]
module.networking.aws_subnet.this: Destroying... [id=subnet-0b87c585cb3ac7c7a]
module.security.aws_security_group.backend_sg: Destroying... [id=sg-04212b5d389bd5342]
module.backend_servers["web-2"].aws_key_pair.this: Destruction complete after 0s
module.backend_servers["web-1"].aws_key_pair.this: Destruction complete after 0s
module.networking.aws_subnet.this: Destruction complete after 1s
module.security.aws_security_group.backend_sg: Destruction complete after 1s
module.security.aws_security_group.nginx_sg: Destroying... [id=sg-038da4178bc17d410]
module.security.aws_security_group.nginx_sg: Destruction complete after 1s
module.networking.aws_vpc.this: Destroying... [id=vpc-047b1eb8c3d0e780e]
module.networking.aws_vpc.this: Destruction complete after 0s

Destroy complete! Resources: 17 destroyed.
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11/Lab12_Assignment (main) $
```

Step 2: Confirm Completion

```
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11/Lab12_Assignment (main) $ cat terraform.tfstate
{
  "version": 4,
  "terraform_version": "1.14.3",
  "serial": 111,
  "lineage": "c07899d9-348f-9281-e4f7-083a74ddbda7",
  "outputs": {},
  "resources": [],
  "check_results": [
    {
      "object_kind": "var",
      "config_addr": "var.vpc_cidr_block",
      "status": "unknown",
      "objects": null
    },
    {
      "object_kind": "var",
      "config_addr": "var.subnet_cidr_block",
      "status": "unknown",
      "objects": null
    }
  ]
}
@Zunaira-Noor123 ➜ /workspaces/CC_ZunairaNoor_075_Lab11/Lab12_Assignment (main) $
```

Step 4: Verify AWS Console



5. Challenges & Solutions

During the deployment of the multi-tier web infrastructure for this assignment, several technical challenges were encountered. These challenges ranged from AWS configuration issues to Terraform scripting and Nginx configuration problems. Addressing these issues required careful debugging, testing, and iteration. Below is a detailed description of the problems faced, the solutions implemented, and the lessons learned.

5.1 Problems Encountered

1. **Initial EC2 Instances Did Not Have Public IP**
    - **Issue:** Upon initial deployment, the backend and Nginx EC2 instances were launched without public IP addresses. This prevented remote access via SSH, which also caused the automation scripts (nginx-setup.sh and apache-setup.sh) to fail because they rely on public connectivity for downloading updates and fetching metadata.

- **Impact:** Deployment scripts could not execute correctly, leaving EC2 instances in a partially configured state. This delayed the setup of the infrastructure and made initial testing impossible.
- **Resolution:** The Terraform configuration for the subnet and EC2 instances was updated to include associate_public_ip_address = true. This ensured all instances were accessible over the internet. After this correction, scripts executed successfully, allowing the automation of software installation, SSL certificate generation, and Nginx/Apache configuration.

2. **SSL Certificate Generation Failed Due to Missing IP Metadata**
   - **Issue:** During the Nginx setup, the SSL certificate generation using OpenSSL initially failed. The self-signed certificate required the server's public IP as the Common Name (CN). The script could not retrieve the correct public IP using metadata queries because **IMDSv2** (Instance Metadata Service v2) was not implemented in the first version of the script.
   - **Impact:** Nginx could not start with HTTPS, and testing of secure connections (HTTPS) was not possible. This also affected the configuration of security headers and HTTPS redirects.
   - **Resolution:** The script was modified to fetch instance metadata using **IMDSv2**, which is the secure version of AWS EC2 metadata access. A temporary token was requested first and then used to query the public IP and other metadata. This approach allowed proper generation of the SSL certificate and ensured HTTPS was configured correctly on Nginx.
3. **Load Balancing Did Not Distribute Traffic Evenly in the First Attempt**
   - **Issue:** After deploying Nginx with the upstream backend configuration, it was observed that traffic requests were not evenly distributed between **web-1** and **web-2** servers. All requests were being served by only one server in the initial tests.
   - **Impact:** The main purpose of the multi-tier infrastructure—high availability and load balancing—was compromised. Performance testing could not proceed until this was resolved.
   - **Resolution:** The Nginx configuration was revisited. The upstream block was corrected to include all primary servers (web-1 and web-2) and the backup server (web-3) with the backup directive. After updating the configuration, nginx -t confirmed the syntax was correct, and restarting the Nginx service ensured proper round-robin load balancing. Multiple browser reloads confirmed that requests alternated correctly between the two primary backend servers.
4. **Backend Server Caching Misconfigured → Repeated MISS Responses**
   - **Issue:** The initial caching configuration in Nginx did not properly store responses in the cache. The first request always returned MISS, and repeated requests continued to show MISS instead of HIT. This indicated that the cache was either not being written to or the cache directory permissions were incorrect.

- **Impact:** Caching, which improves performance and reduces load on backend servers, was not functioning. This meant all requests hit the backend servers, leading to unnecessary load and slower response times.
- **Resolution:**
  - Ensured that the cache directory /var/cache/nginx existed and had proper ownership and permissions (chown -R nginx:nginx /var/cache/nginx).
  - Updated the Nginx configuration to correctly define proxy_cache_path and proxy_cache directives.
  - Verified caching keys using $scheme$request_method$host$request_uri to ensure unique cache entries for each request.
  - Reloaded Nginx and performed multiple test requests, confirming that the first request returned MISS and subsequent requests returned HIT.

5.2 Lessons Learned

- **Infrastructure Planning:** Always verify networking requirements, such as public vs private IP addresses, before deploying automation scripts. Missing configuration can block the execution of critical setup scripts.
- **Secure Metadata Access:** Using IMDSv2 for metadata retrieval ensures secure access to instance information and prevents failures in scripts that depend on dynamic data.
- **Nginx Configuration:** Careful attention to upstream, caching, and security directives is critical for achieving expected performance and high availability.
- **Automation & Debugging:** Terraform modularization simplifies deployments but requires careful variable management and validation to prevent errors.
- **Testing & Verification:** Testing each component independently (Apache backend, Nginx proxy, caching, SSL) helps isolate problems and verify functionality before integration.

6. Conclusion

In this assignment, a production-ready multi-tier web infrastructure was successfully designed and deployed on AWS using Terraform and Nginx. The project involved creating a modular Terraform configuration, provisioning networking and security resources, launching EC2 instances, and automating server configurations with shell scripts.

The work completed includes:

- **Infrastructure Setup:** A custom VPC, public subnets, internet gateway, and routing were implemented. Security groups were configured for Nginx (reverse proxy/load balancer) and backend web servers, ensuring secure SSH access and controlled HTTP/HTTPS traffic.

- **Webserver Modules:** A reusable Terraform module was developed to deploy both Nginx and backend servers, enabling scalability and code reusability. EC2 instances were automatically configured with key pairs, tags, and user data scripts.
- **Server Configuration:** Apache was installed on backend servers, with custom HTML pages displaying instance metadata and deployment details. Nginx was configured with HTTPS, caching, reverse proxy, load balancing, security headers, and health checks.
- **Deployment & Testing:** Terraform was used to deploy the complete infrastructure. Nginx was tested for load balancing, caching, and high availability using a backup server. Security headers and SSL/TLS configurations were verified, ensuring a secure and performant environment.
- **Cleanup:** All resources were successfully destroyed using Terraform, and the AWS environment was confirmed to be free of residual resources.

Skills Acquired:

- Proficiency in **Terraform** for Infrastructure as Code (IaC) and modular infrastructure design.
- Hands-on experience with **AWS networking and security**, including VPCs, subnets, route tables, security groups, and EC2 instance management.
- Understanding of **Nginx advanced configurations**, including reverse proxy, load balancing, caching, HTTPS setup, and security headers.
- Scripting skills for **automation of server configuration** using Bash scripts, including dynamic retrieval of instance metadata and SSL certificate generation.
- Troubleshooting and debugging complex infrastructure issues, including connectivity, caching, and load balancing.

Future Improvements:

- Integrate **Terraform remote state management** for collaboration and versioning.
- Implement **automated deployment pipelines** (CI/CD) for infrastructure and application scripts.
- Extend the architecture with **database layers, auto-scaling, and monitoring tools** for full production readiness.
- Replace self-signed certificates with **trusted SSL certificates** for improved security.
- Add more robust **health check automation and alerting** for backend servers to enhance reliability.

Overall, this assignment provided comprehensive practical experience in deploying a **secure, scalable, and high-availability web infrastructure** on AWS. It reinforced the importance of planning, modular code design, automation, and rigorous testing in modern cloud-based deployments.

7. Appendices

The appendices provide all supporting materials for the assignment, including complete code listings, configuration files, screenshots, and references. These serve as a detailed reference for verification, troubleshooting, and replication of the deployed infrastructure.

7.1 Complete Code Listings

Terraform Root Files:

- main.tf – Instantiates modules for networking, security, and web servers.
- variables.tf – Defines all required variables with validation and descriptions.
- outputs.tf – Outputs VPC, subnet, instance IDs, and IPs.
- locals.tf – Contains dynamic IP detection, tags, and backend server configurations.
- terraform.tfvars.example – Example variable values (no real credentials).

Terraform Modules:

- modules/networking/main.tf – VPC, subnet, IGW, route tables.
- modules/networking/variables.tf – Networking module variables.
- modules/networking/outputs.tf – Networking outputs.
- modules/security/main.tf – Security groups for Nginx and backend servers.
- modules/security/variables.tf – Security module variables.
- modules/security/outputs.tf – Security group outputs.
- modules/webserver/main.tf – Key pair and EC2 instance provisioning.
- modules/webserver/variables.tf – Instance deployment variables.
- modules/webserver/outputs.tf – Instance outputs (public/private IPs, IDs).

Server Configuration Scripts:

- scripts/apache-setup.sh – Installs Apache, sets hostname, generates custom HTML with instance metadata, enables service.
- scripts/nginx-setup.sh – Installs Nginx, generates self-signed SSL certificate, configures reverse proxy, load balancing, caching, and security headers.

Sample Code Snippet (apache-setup.sh):

```
#!/bin/bash

yum update -y

yum install httpd -y
```

systemctl start httpd

systemctl enable httpd

echo "<h1>Backend Server $(hostname)</h1>" > /var/www/html/index.html

7.2 Configuration Files

.gitignore – Excludes sensitive files:

*.tfstate

*.tfstate.backup

.terraform/

*.pem

terraform.tfvars

Nginx Configuration (/etc/nginx/nginx.conf):

- Reverse proxy
- HTTPS with self-signed certificate
- Load balancing with backend servers
- Caching and security headers
- HTTP→HTTPS redirect

Apache Configuration (/etc/httpd/conf/httpd.conf):

- Default setup with custom HTML content displaying instance metadata and deployment info

7.3 Additional Screenshots

Infrastructure:

- VPC, subnet, route table, internet gateway
- Security groups for Nginx and backend servers
- EC2 instances (Nginx and backend)

Deployment & Testing:

- Terraform commands: init, validate, plan, apply
- Backend server pages showing metadata and deployment details
- Nginx default page before and after backend configuration
- Load balancing demonstration (web-1, web-2 alternation)
- Caching test: MISS → HIT responses

- Backup server activation during high availability test
- SSL certificate details, security headers, HTTP→HTTPS redirect verification

Example Screenshot Placeholder:

[Insert Screenshot: assignment_part4_terraform_apply.png]

[Insert Screenshot: assignment_part5_load_balancing_demo.png]

[Insert Screenshot: assignment_part5_cache_hit.png]

7.4 References

1. Terraform Documentation – Official Terraform guides for AWS resources and modules
2. AWS EC2 Documentation – EC2 instance setup, key pairs, and security groups
3. AWS VPC Documentation – VPC, subnets, routing, and internet gateway
4. Nginx Official Documentation – Reverse proxy, caching, load balancing, SSL/TLS
5. Bash Scripting Guide – Automation scripts and metadata handling
6. Instance Metadata Service v2 (IMDSv2) – Secure retrieval of instance metadata