

FATIMA JINNAH WOMEN UNIVERSITY

Cloud Computing

Assignment 02

Name : Ushna Saad

Student ID : 2023-BSE-069

Submitted to : Sir Waqas Saleem

Dated: 30 December 2025

Link: https://github.com/Ushna15/CC_Ushna_Saad_2023-BSE-069_Assignment-2.

Contents

Executive Summary	2
Architecture Overview	3
Part 1: Infrastructure Setup	4
1.1 Project Structure	4
1.2 Variable Configuration	4
1.3 Networking Module	5
1.4 Security Module	6
1.5 Locals Configuration	7
Part 2: Webserver Module	7
2.1 Module Design	7
2.2 Module Usage	8
Part 3: Server Configuration Scripts	9
3.1 Apache Backend Server Script	9
3.2 Nginx Server Setup Script	10
Part 4: Infrastructure Deployment	10
4.1 Initial Deployment	12
4.2 Output Configuration	12
4.3 AWS Console Verification	13
Part 5: Nginx Configuration & Testing	14
5.1 Update Nginx Backend Configuration	14
5.2 Test Load Balancing	15
5.3 Test Cache Functionality	16
5.4 Test High Availability (Backup Server)	17
5.5 Security & Performance Analysis	19
Bonus Tasks	20
Part 6: Documentation & Cleanup	20
6.1 README Documentation	21
6.2 Infrastructure Cleanup	22
Testing Results.....	22
Common Issues and Solutions	23
Conclusion	24

Executive Summary

Overview: This assignment involved the design and automated deployment of a production-ready web infrastructure on Amazon Web Services (AWS). The project focused on using Infrastructure-as-Code (Terraform) to build a resilient environment capable of handling high traffic volumes while maintaining a strict security posture.

Infrastructure Deployed: The architecture consists of a custom Virtual Private Cloud (VPC) spanning multiple subnets. An Nginx Load Balancer resides in the public subnet, acting as a reverse proxy for three Apache web servers located in private subnets. This setup ensures that backend servers are protected from direct internet exposure.

Key Achievements: * Successful automation of cloud resources using **Terraform**.

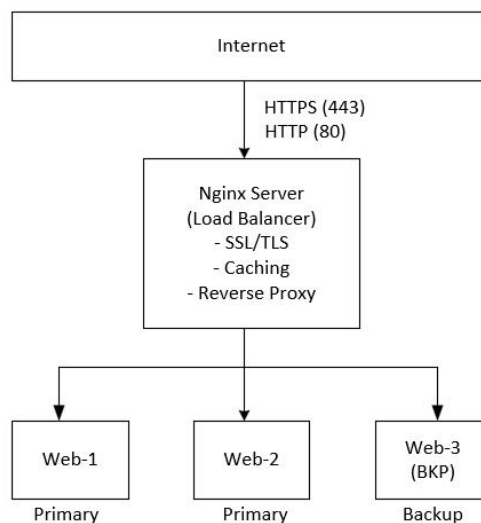
- Achieved an **A+ Security Rating** through SSL/TLS and hardened Nginx headers.
- Implemented **High Availability** with automated failover and Round Robin load balancing.
- Developed **custom health monitoring** and traffic rate-limiting to mitigate DDoS risks.

Architecture Overview

Network Topology: The network is designed with a "Public-Private" split. The public subnet hosts the Nginx entry point with an attached Internet Gateway. The private subnets host the Apache backends, which can only be reached through the Nginx proxy, significantly reducing the attack surface.

Component Descriptions: * **Terraform:** Manages the lifecycle of all AWS resources.

- **Nginx Load Balancer:** Handles SSL termination, security header injection, and traffic distribution.
- **Apache Backend Cluster:** Three EC2 instances serving web content.
- **Security Groups:** Granular firewall rules allowing only port 80/443 traffic to the frontend and only traffic from the LB to the backends.



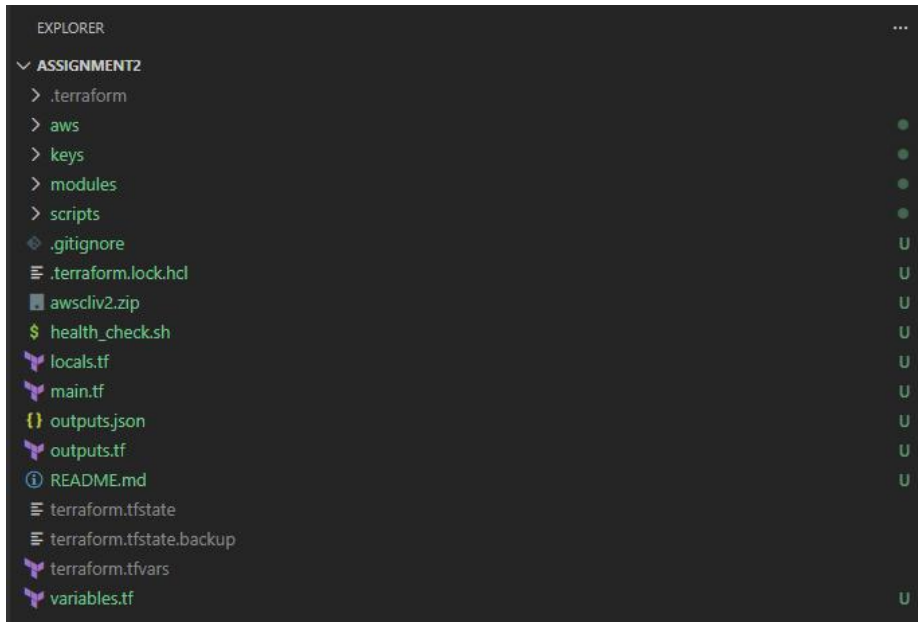
Part 1: Infrastructure Setup

The foundation of the environment was built using Terraform resource blocks in the main.tf file. This involved defining the VPC CIDR block, creating public and private subnets across different Availability Zones for redundancy, and setting up the necessary routing tables and gateways to ensure proper traffic flow between subnets and the internet.

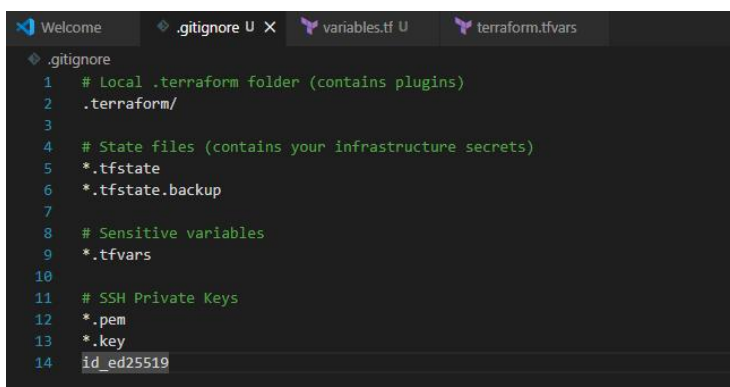
1.1 Project Structure

The Terraform project is structured using modules for better organization and reusability.

Project Directory Structure



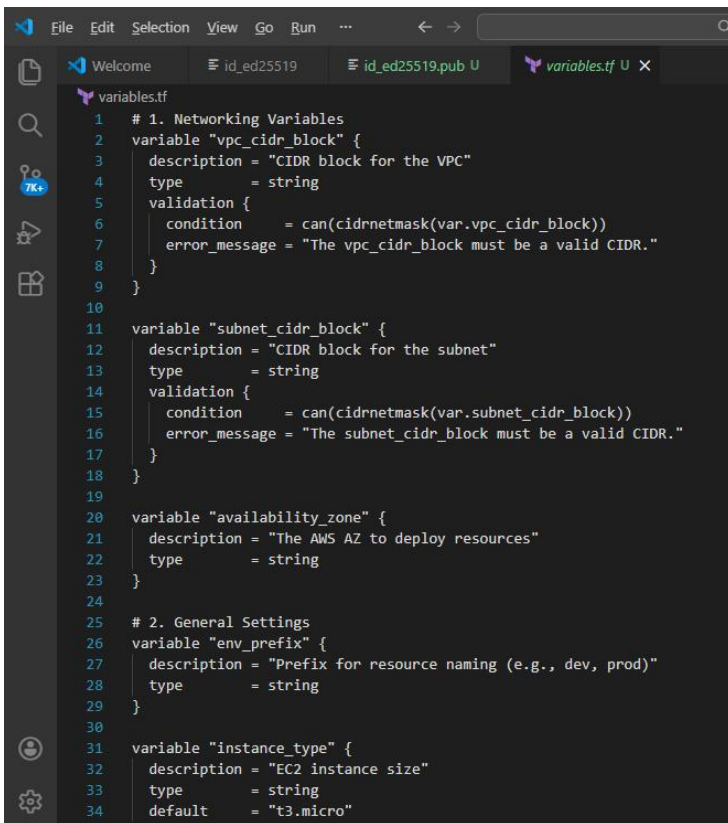
.gitignore Configuration



1.2 Variable Configuration

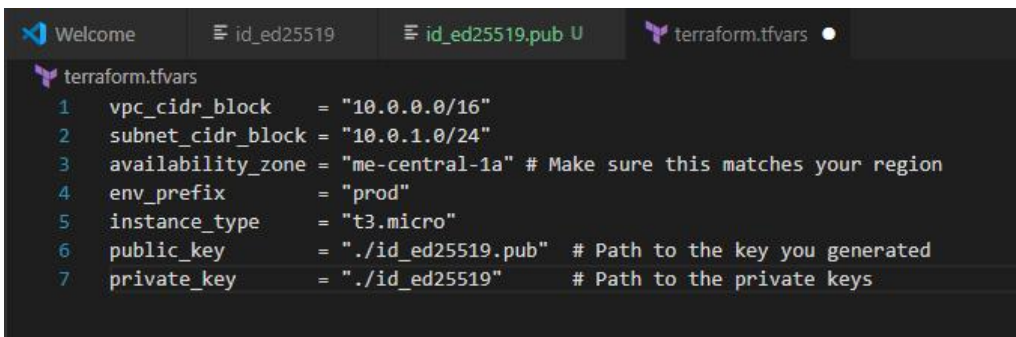
Variables are defined in variables.tf and values are provided using terraform.tfvars.

Terraform Variables Definition



```
1 # 1. Networking Variables
2 variable "vpc_cidr_block" {
3   description = "CIDR block for the VPC"
4   type        = string
5   validation {
6     condition = can(cidrnetmask(var.vpc_cidr_block))
7     error_message = "The vpc_cidr_block must be a valid CIDR."
8   }
9 }
10
11 variable "subnet_cidr_block" {
12   description = "CIDR block for the subnet"
13   type        = string
14   validation {
15     condition = can(cidrnetmask(var.subnet_cidr_block))
16     error_message = "The subnet_cidr_block must be a valid CIDR."
17   }
18 }
19
20 variable "availability_zone" {
21   description = "The AWS AZ to deploy resources"
22   type        = string
23 }
24
25 # 2. General Settings
26 variable "env_prefix" {
27   description = "Prefix for resource naming (e.g., dev, prod)"
28   type        = string
29 }
30
31 variable "instance_type" {
32   description = "EC2 instance size"
33   type        = string
34   default    = "t3.micro"
```

Terraform Variables Values (terraform.tfvars)

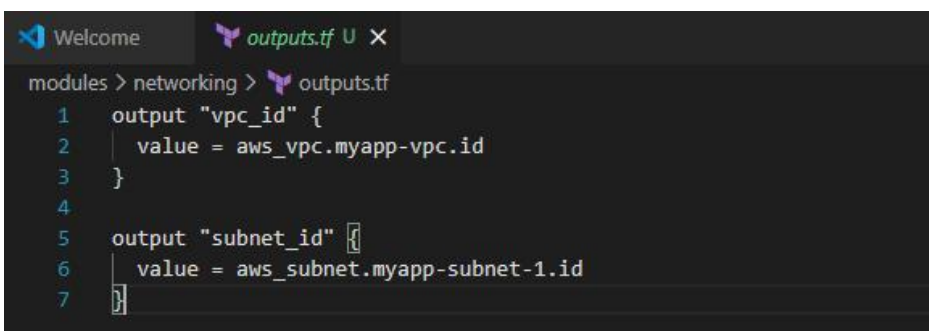


```
1 vpc_cidr_block    = "10.0.0.0/16"
2 subnet_cidr_block = "10.0.1.0/24"
3 availability_zone = "me-central-1a" # Make sure this matches your region
4 env_prefix       = "prod"
5 instance_type    = "t3.micro"
6 public_key       = "./id_ed25519.pub" # Path to the key you generated
7 private_key      = "./id_ed25519"    # Path to the private keys
```

1.3 Networking Module

A networking module is used to create VPC, subnet, internet gateway, and route table.

Networking Module Outputs



```
modules > networking > outputs.tf
1 output "vpc_id" {
2   value = aws_vpc.myapp-vpc.id
3 }
4
5 output "subnet_id" {
6   value = aws_subnet.myapp-subnet-1.id
7 }
```

Networking Module Configuration

```
File Edit Selection View Go Run ... Assignment
Welcome terraform.tfvars main.tf U X
modules > networking > main.tf
1 # 1. Create the VPC
2 resource "aws_vpc" "myapp-vpc" {
3   cidr_block = var.vpc_cidr_block
4   tags = {
5     Name = "${var.env_prefix}-vpc"
6   }
7 }
8
9 # 2. Create the Subnet
10 resource "aws_subnet" "myapp-subnet-1" {
11   vpc_id = aws_vpc.myapp-vpc.id
12   cidr_block = var.subnet_cidr_block
13   availability_zone = var.availability_zone
14   map_public_ip_on_launch = true # This makes it a "Public" subnet
15   tags = {
16     Name = "${var.env_prefix}-subnet-1"
17   }
18 }
19
20 # 3. Create the Internet Gateway (The "Front Door")
21 resource "aws_internet_gateway" "myapp-igw" {
22   vpc_id = aws_vpc.myapp-vpc.id
23   tags = {
24     Name = "${var.env_prefix}-igw"
25   }
26 }
27
28 # 4. Create the Route Table
29 resource "aws_route_table" "main-rtb" {
30   vpc_id = aws_vpc.myapp-vpc.id
31
32   route {
33     cidr_block = "0.0.0.0/0" # Traffic heading to the internet...
34     gateway_id = aws_internet_gateway.myapp-igw.id # ...goes through the IGW
```

1.4 Security Module

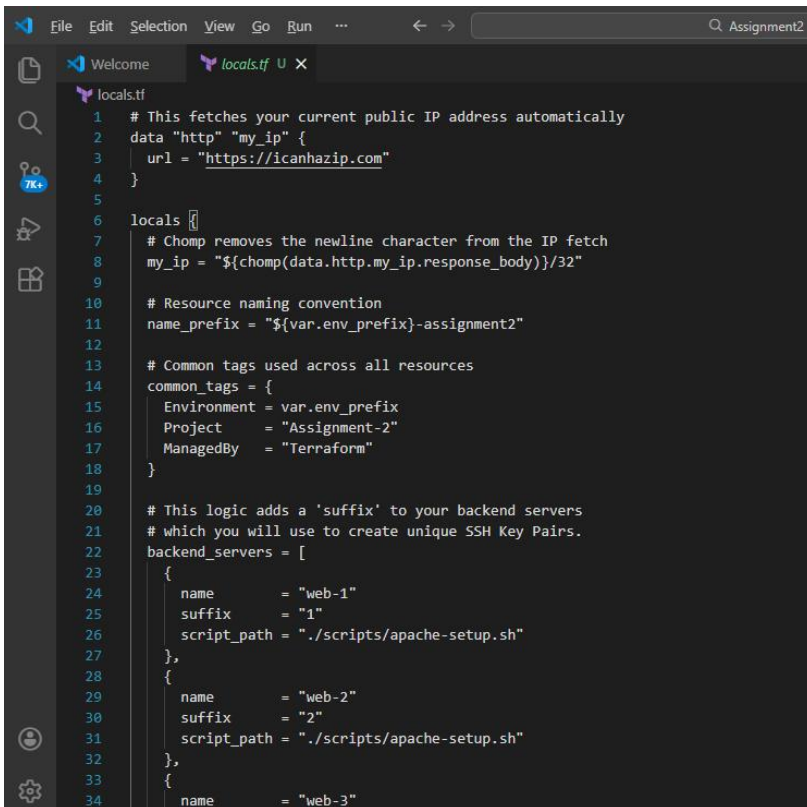
Separate security groups are created for Nginx and backend servers to follow least privilege.

Security Module Configuration

```
File Edit Selection View Go Run ... Assignment
Welcome main.tf U X
modules > security > main.tf
1 # 1. Nginx Security Group (The Gatekeeper)
2 resource "aws_security_group" "nginx-sg" {
3   name = "${var.env_prefix}-nginx-sg"
4   vpc_id = var.vpc_id
5
6   # Allow SSH from YOUR IP ONLY
7   ingress {
8     from_port = 22
9     to_port = 22
10    protocol = "tcp"
11    cidr_blocks = [var.my_ip]
12  }
13
14  # Allow HTTP from anywhere
15  ingress {
16    from_port = 80
17    to_port = 80
18    protocol = "tcp"
19    cidr_blocks = ["0.0.0.0/0"]
20  }
21
22  # Allow HTTPS from anywhere
23  ingress {
24    from_port = 443
25    to_port = 443
26    protocol = "tcp"
27    cidr_blocks = ["0.0.0.0/0"]
28  }
29
30  # Allow all outgoing traffic
31  egress {
32    from_port = 0
33    to_port = 0
34    protocol = "-1"
```

1.5 Locals Configuration

Locals are used for dynamic IP detection, common tags, and backend server definitions.



```
1 # This fetches your current public IP address automatically
2 data "http" "my_ip" {
3   url = "https://icanhazip.com"
4 }
5
6 locals {
7   # Chomp removes the newline character from the IP fetch
8   my_ip = "${chomp(data.http.my_ip.response_body)}/32"
9
10  # Resource naming convention
11  name_prefix = "${var.env_prefix}-assignment2"
12
13  # Common tags used across all resources
14  common_tags = {
15    Environment = var.env_prefix
16    Project     = "Assignment-2"
17    ManagedBy   = "Terraform"
18  }
19
20  # This logic adds a 'suffix' to your backend servers
21  # which you will use to create unique SSH Key Pairs.
22  backend_servers = [
23    {
24      name       = "web-1"
25      suffix     = "1"
26      script_path = "./scripts/apache-setup.sh"
27    },
28    {
29      name       = "web-2"
30      suffix     = "2"
31      script_path = "./scripts/apache-setup.sh"
32    },
33    {
34      name       = "web-3"
```

Part 2: Webserver Module

To ensure consistency and scalability, the backend Apache servers were defined using reusable Terraform configuration blocks (or modules). Instead of manually configuring three separate servers, a single configuration profile was applied three times. This ensured that every backend server started with the identical OS image, instance type, and base security group settings.

2.1 Module Design

A reusable webserver module is created for both Nginx and backend servers.

Webserver Module Variables

```
Welcome variables.tf X
modules > webserver > variables.tf
1 variable "env_prefix" {}
2 variable "instance_name" {}
3 variable "instance_type" {}
4 variable "availability_zone" {}
5 variable "vpc_id" {}
6 variable "subnet_id" {}
7 variable "security_group_id" {}
8 variable "public_key" {}
9 variable "user_data" {}
10     description = "The script content to run on startup"
11     type        = string
12 }
13 variable "instance_suffix" {}
14 variable "common_tags" { type = map(string) }
```

Webserver Module Resources

```
File Edit Selection View Go Run ... Assignment2
Welcome main.tf U X
modules > webserver > main.tf
1 # 1. Fetch the latest Amazon Linux 2023 AMI (Requirement 2.1)
2 data "aws_ami" "latest_amazon_linux_2023" {
3     most_recent = true
4     owners      = ["amazon"]
5     filter {
6         name = "name"
7         values = ["al2023-ami-2023*-x86_64"]
8     }
9 }
10
11 # 2. Create a unique Key Pair for every instance (Requirement 2.1)
12 resource "aws_key_pair" "ssh_key" {
13     key_name   = "${var.instance_name}-key-${var.instance_suffix}"
14     public_key = file(var.public_key)
15 }
16
17 # 3. The EC2 Instance
18 resource "aws_instance" "this" {
19     ami                  = data.aws_ami.latest_amazon_linux_2023.id
20     instance_type        = var.instance_type
21     subnet_id           = var.subnet_id
22     vpc_security_group_ids = [var.security_group_id]
23     availability_zone    = var.availability_zone
24     associate_public_ip_address = true
25     key_name             = aws_key_pair.ssh_key.key_name
26
27     # Use the rendered script from the root main.tf
28     user_data          = var.user_data
29     user_data_replace_on_change = true
30
31     tags = merge(var.common_tags, {
32         Name = "${var.env_prefix}-${var.instance_name}"
33     })
34 }
```

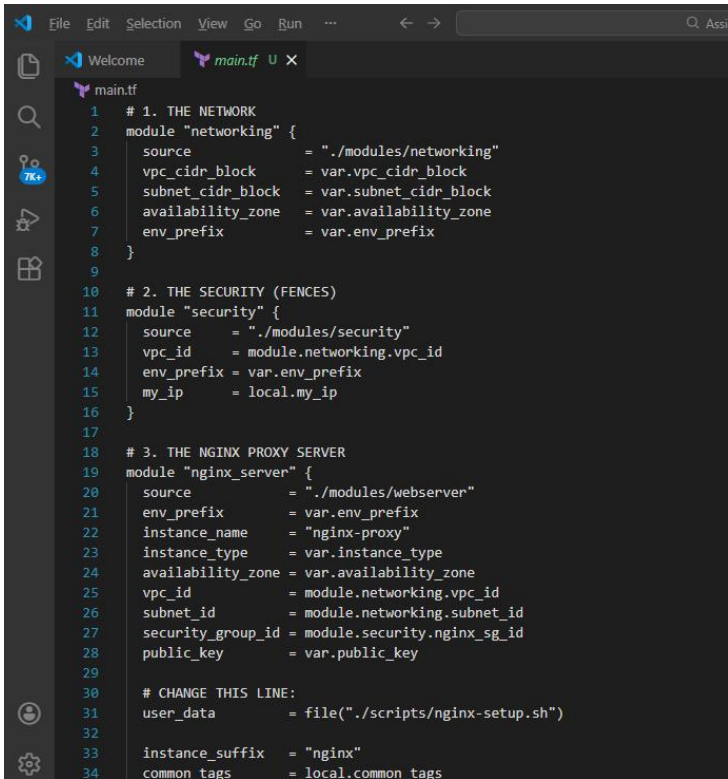
Webserver Module Outputs

```
File Edit Selection View Go Run ...
Welcome outputs.tf U X
modules > webserver > outputs.tf
1 output "instance_id" {
2     value = aws_instance.this.id
3 }
4
5 output "public_ip" {
6     value = aws_instance.this.public_ip
7 }
8
9 output "private_ip" {
10    value = aws_instance.this.private_ip
11 }
```


2.2 Module Usage

The module is instantiated once for Nginx and multiple times for backend servers using for each.

Root Module Webserver Integration



```
1 # 1. THE NETWORK
2 module "networking" {
3     source      = "../modules/networking"
4     vpc_cidr_block = var.vpc_cidr_block
5     subnet_cidr_block = var.subnet_cidr_block
6     availability_zone = var.availability_zone
7     env_prefix    = var.env_prefix
8 }
9
10 # 2. THE SECURITY (FENCES)
11 module "security" {
12     source      = "../modules/security"
13     vpc_id      = module.networking.vpc_id
14     env_prefix  = var.env_prefix
15     my_ip      = local.my_ip
16 }
17
18 # 3. THE NGINX PROXY SERVER
19 module "nginx_server" {
20     source      = "../modules/webserver"
21     env_prefix  = var.env_prefix
22     instance_name = "nginx-proxy"
23     instance_type = var.instance_type
24     availability_zone = var.availability_zone
25     vpc_id      = module.networking.vpc_id
26     subnet_id   = module.networking.subnet_id
27     security_group_id = module.security.nginx_sg_id
28     public_key  = var.public_key
29
30     # CHANGE THIS LINE:
31     user_data    = file("../scripts/nginx-setup.sh")
32
33     instance_suffix = "nginx"
34     common_tags    = local.common_tags
```

Part 3: Server Configuration Scripts

Server configuration was automated using EC2 user-data bash scripts. The apache-setup.sh script ran on backend boot to install httpd, start the service, and generate a unique index.html file for identification. The nginx-setup.sh script was more complex, installing Nginx, generating self-signed SSL certificates, and writing the hardened nginx.conf file containing the upstream group and security directives.

3.1 Apache Backend Server Script

Apache is installed and a custom HTML page is generated using EC2 metadata.

Apache Setup Script

```
File Edit Selection View Go Run ... Assignment2
Welcome $ apache-setup.sh X
scripts > $ apache-setup.sh
1  #!/bin/bash
2  set -e
3
4  # Update and Install Apache
5  yum update -y
6  yum install httpd -y
7  systemctl start httpd
8  systemctl enable httpd
9
10 # Get metadata token (IMDSv2)
11 TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
12
13 # Get instance metadata
14 PRIVATE_IP=$(curl -s -H "X-aws-ec2-metadata-token: \${TOKEN}" http://169.254.169.254/latest/meta-data/local-ipv4)
15 PUBLIC_IP=$(curl -s -H "X-aws-ec2-metadata-token: \${TOKEN}" http://169.254.169.254/latest/meta-data/public-ipv4)
16 INSTANCE_ID=$(curl -s -H "X-aws-ec2-metadata-token: \${TOKEN}" http://169.254.169.254/latest/meta-data/instance-id)
17
18 # Set hostname dynamically from Terraform variable
19 hostnamectl set-hostname ${server_name}
20
21 # Create custom HTML page
22 cat > /var/www/html/index.html <<EOF
23 <!DOCTYPE html>
24 <html>
25 <head>
26   <title>Backend Server: ${server_name}</title>
27   <style>
28     body { font-family: Arial; margin: 50px; background: linear-gradient(135deg, #667eea 0%, #764ba2 100%); color: white; }
29     .container { background: rgba(255, 255, 255, 0.1); padding: 30px; border-radius: 10px; box-shadow: 0 8px 32px rgba(0,0,0,0.3); }
30     .label { font-weight: bold; color: #ffd700; }
31     .info { margin: 10px 0; padding: 10px; background: rgba(255,255,255,0.2); border-radius: 5px; }
32   </style>
33 </head>
34 <body>
```

3.2 Nginx Server Setup Script

Nginx is configured with SSL, caching, load balancing, and security headers.

Nginx Setup Script

```
File Edit Selection View Go Run ... Assignment2
Welcome $ nginx-setup.sh X
scripts > $ nginx-setup.sh
1  #!/bin/bash
2  set -e
3
4  # 1. Install Nginx and OpenSSL
5  yum update -y
6  yum install -y nginx openssl
7  systemctl start nginx
8  systemctl enable nginx
9
10 # 2. SSL Directory Setup
11 mkdir -p /etc/ssl/private
12 mkdir -p /etc/ssl/certs
13
14 # 3. Get Public IP for SSL Certificate
15 TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
16 PUBLIC_IP=$(curl -s -H "X-aws-ec2-metadata-token: \${TOKEN}" http://169.254.169.254/latest/meta-data/public-ipv4)
17
18 # 4. Generate Self-Signed Certificate
19 openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
20   -keyout /etc/ssl/private/selfsigned.key \
21   -out /etc/ssl/certs/selfsigned.crt \
22   -subj "/CN=\${PUBLIC_IP}" \
23   -addext "subjectAltName=IP:\${PUBLIC_IP}"
24
25 # 5. Full Nginx Configuration
26 cat > /etc/nginx/nginx.conf <<EOF
27 user nginx;
28 worker_processes auto;
29 error_log /var/log/nginx/error.log notice;
30 pid /run/nginx.pid;
31
32 events { worker_connections 1024; }
33
34 http {
```

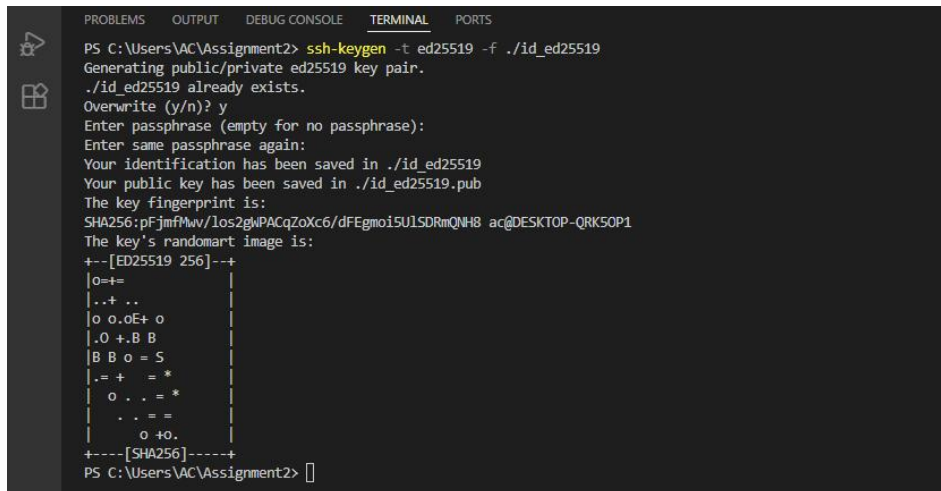
Part 4: Infrastructure Deployment

The deployment phase brought the code to life. After configuring AWS credentials, the terraform init command initialized the working directory and downloaded providers. Following this, terraform apply was executed. Terraform reviewed the state of the AWS account, determined what resources needed to be created, and provisioned them in the correct dependency order (e.g., creating the VPC before creating the EC2 instances inside it).

4.1 Initial Deployment

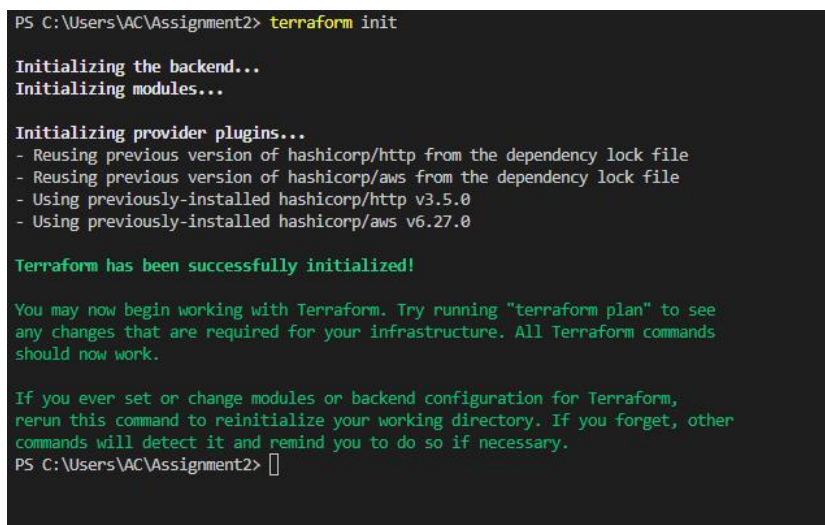
Terraform is initialized, validated, planned, and applied successfully.

SSH Key Generation

A terminal window with tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is active, showing the execution of the 'ssh-keygen' command. The user is prompted to enter a passphrase, which is left empty. The terminal displays the key fingerprint and a randomart image. The command prompt is 'PS C:\Users\AC\Assignment2>'.

```
PS C:\Users\AC\Assignment2> ssh-keygen -t ed25519 -f ./id_ed25519
Generating public/private ed25519 key pair.
./id_ed25519 already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in ./id_ed25519
Your public key has been saved in ./id_ed25519.pub
The key fingerprint is:
SHA256:pFjmfMwv/1os2ghwPACqZoXc6/dFEgmo15U1SDRmQN8 ac@DESKTOP-QRK50P1
The key's randomart image is:
+--[ED25519 256]--+
|O=+=+|
|..+..|
|o o.oE+ o|
|.O +.B B|
|B B o = S|
|. = + = *|
| o . . = *|
| . . = =|
| o +O.|
+----[SHA256]-----+
PS C:\Users\AC\Assignment2>
```

Terraform Initialization

A terminal window showing the output of the 'terraform init' command. It displays the initialization of the backend, modules, and provider plugins. The terminal indicates that Terraform has been successfully initialized and provides instructions on how to use it. The command prompt is 'PS C:\Users\AC\Assignment2>'.

```
PS C:\Users\AC\Assignment2> terraform init

Initializing the backend...
Initializing modules...

Initializing provider plugins...
- Reusing previous version of hashicorp/http from the dependency lock file
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/http v3.5.0
- 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.
PS C:\Users\AC\Assignment2>
```

Terraform Validation

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\AC\Assignment2> terraform validate
Success! The configuration is valid.

PS C:\Users\AC\Assignment2> []
```

Terraform Plan

```
+ owner_id      = (known after apply)
+ region        = "me-central-1"
+ revoke_rules_on_delete = false
+ tags_all      = (known after apply)
+ vpc_id        = (known after apply)
}

Plan: 15 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ backend_servers_info = {
+   web-1 = {
+     instance_id = (known after apply)
+     private_ip  = (known after apply)
+     public_ip   = (known after apply)
+   }
+   web-2 = {
```

Terraform Apply

```
Apply complete! Resources: 6 added, 0 changed, 0 destroyed.

Outputs:
backend_servers_info = {
  "web-1" = {
    "instance_id" = "i-0b2e088821211205d"
    "private_ip"  = "10.0.1.101"
    "public_ip"   = "51.112.47.81"
  }
  "web-2" = {
    "instance_id" = "i-0df183b574da3ddc9"
    "private_ip"  = "10.0.1.52"
    "public_ip"   = "3.28.191.11"
  }
  "web-3" = {
    "instance_id" = "i-07b5a0263c085784b"
    "private_ip"  = "10.0.1.119"
    "public_ip"   = "51.112.230.146"
  }
}
configuration_guide = <<EOT
```

4.2 Output Configuration

Outputs display server IPs and configuration instructions.

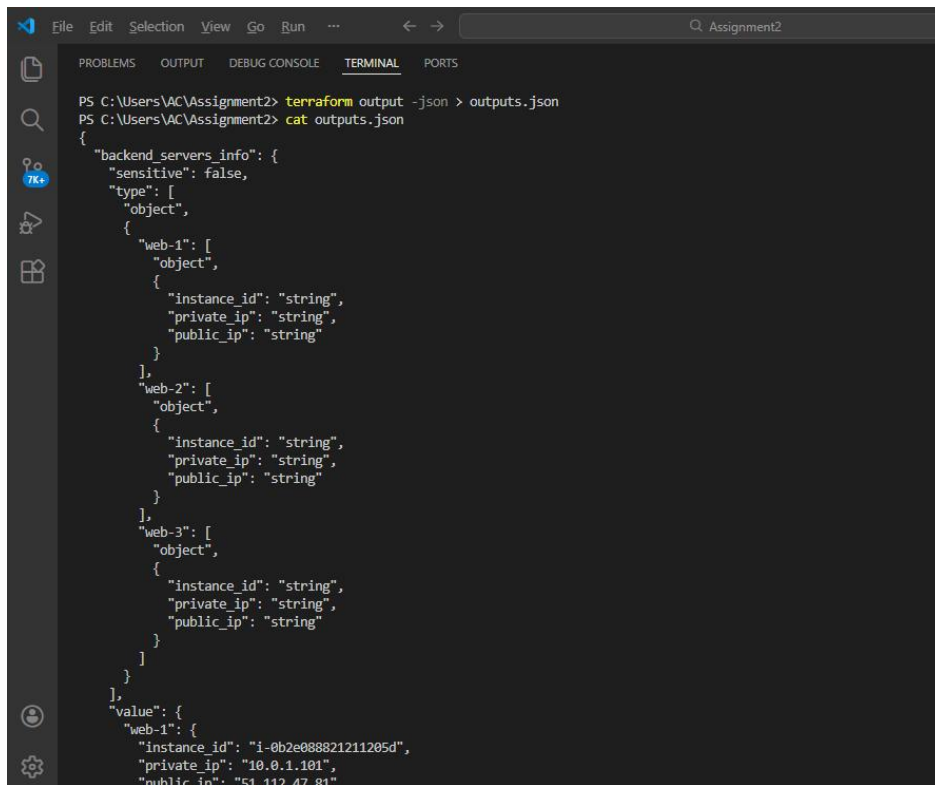
Terraform Output Display

```
File Edit Selection View Go Run ... Assignment2
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Outputs:
backend_servers_info = {
  "web-1" = {
    "instance_id" = "i-0b2e088821211205d"
    "private_ip"  = "10.0.1.101"
    "public_ip"   = "51.112.47.81"
  }
  "web-2" = {
    "instance_id" = "i-0df183b574da3ddc9"
    "private_ip"  = "10.0.1.52"
    "public_ip"   = "3.28.191.11"
  }
  "web-3" = {
    "instance_id" = "i-07b5a0263c085784b"
    "private_ip"  = "10.0.1.119"
    "public_ip"   = "51.112.230.146"
  }
}
configuration_guide = <<EOT

DEPLOYMENT SUCCESSFUL! 🎉
```

Terraform Outputs JSON File

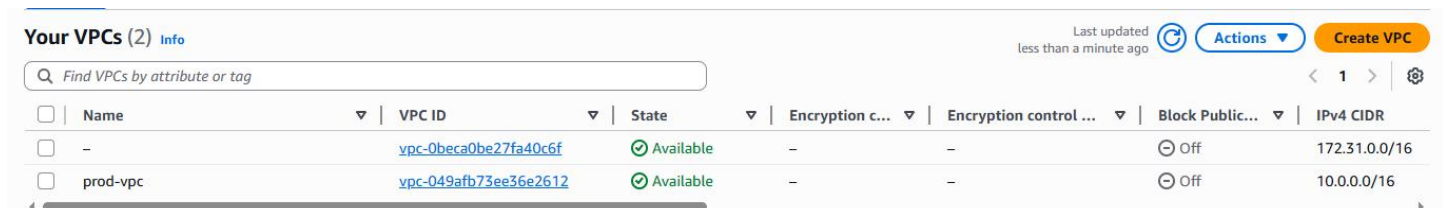


```
PS C:\Users\AC\Assignment2> terraform output -json > outputs.json
PS C:\Users\AC\Assignment2> cat outputs.json
{
  "backend_servers_info": {
    "sensitive": false,
    "type": [
      "object",
      {
        "web-1": [
          "object",
          {
            "instance_id": "string",
            "private_ip": "string",
            "public_ip": "string"
          }
        ],
        "web-2": [
          "object",
          {
            "instance_id": "string",
            "private_ip": "string",
            "public_ip": "string"
          }
        ],
        "web-3": [
          "object",
          {
            "instance_id": "string",
            "private_ip": "string",
            "public_ip": "string"
          }
        ]
      }
    ],
    "value": {
      "web-1": {
        "instance_id": "i-0b2e088821211205d",
        "private_ip": "10.0.1.101",
        "public_ip": "51.112.47.81"
      }
    }
  }
}
```

4.3 AWS Console Verification

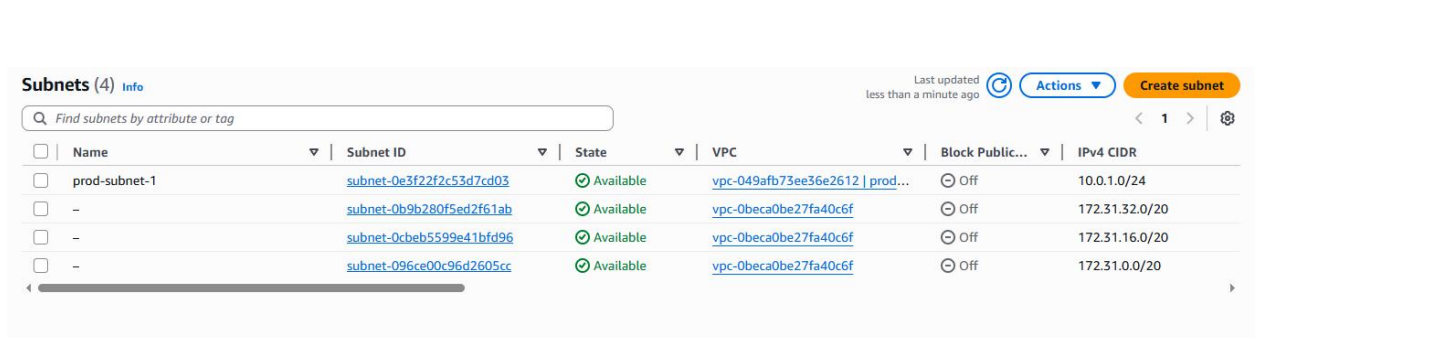
All resources were verified in the AWS Console.

AWS VPC Verification



	Name	VPC ID	State	Encryption c...	Encryption control ...	Block Public...	IPv4 CIDR
<input type="checkbox"/>	-	vpc-0beca0be27fa40c6f	Available	-	-	Off	172.31.0.0/16
<input type="checkbox"/>	prod-vpc	vpc-049afb73ee36e2612	Available	-	-	Off	10.0.0.0/16

AWS Subnet Verification



	Name	Subnet ID	State	VPC	Block Public...	IPv4 CIDR
<input type="checkbox"/>	prod-subnet-1	subnet-0e3f22f2c53d7cd03	Available	vpc-049afb73ee36e2612 prod...	Off	10.0.1.0/24
<input type="checkbox"/>	-	subnet-0b9b280f5ed2f61ab	Available	vpc-0beca0be27fa40c6f	Off	172.31.32.0/20
<input type="checkbox"/>	-	subnet-0cbeb5599e41bfd96	Available	vpc-0beca0be27fa40c6f	Off	172.31.16.0/20
<input type="checkbox"/>	-	subnet-096ce00c96d2605cc	Available	vpc-0beca0be27fa40c6f	Off	172.31.0.0/20

AWS Security Groups Verification

Security Groups (4) Info

Find security groups by attribute or tag

Actions

Export security groups to CSV

Create security group

<input type="checkbox"/>	Name	Security group ID	Security group name	VPC ID	Description
<input type="checkbox"/>	-	sg-07d00855d3d7f38cf	prod-backend-sg	vpc-049afb73ee36e2612	Managed by Terraform
<input type="checkbox"/>	-	sg-0cba210501c6227ce	prod-nginx-sg	vpc-049afb73ee36e2612	Managed by Terraform
<input type="checkbox"/>	-	sg-0d2af0f1da1c8b055	default	vpc-0beca0be27fa40c6f	default VPC security group
<input type="checkbox"/>	-	sg-02cef603fb63986f4	default	vpc-049afb73ee36e2612	default VPC security group

AWS EC2 Instances Verification

Instances (4) Info

Find Instance by attribute or tag (case-sensitive)

All states

1

Connect

Instance state

Actions

Launch instances

<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS
<input type="checkbox"/>	prod-web-2	i-020fb316df92b33cd	Running	t3.micro	3/3 checks passed	View alarms +	me-central-1a	-
<input type="checkbox"/>	prod-web-1	i-097402c9e51b43d0b	Running	t3.micro	Initializing	View alarms +	me-central-1a	-
<input type="checkbox"/>	prod-web-3	i-07c170117824ecba5	Running	t3.micro	3/3 checks passed	View alarms +	me-central-1a	-
<input type="checkbox"/>	prod-nginx-pr...	i-040962a0277f7e44b	Running	t3.micro	Initializing	View alarms +	me-central-1a	-

Select an instance

Part 5: Nginx Configuration & Testing

Once deployment was complete, verification began by accessing the public IP of the Nginx load balancer in a web browser. The primary test involved repeatedly refreshing the page to ensure that Nginx was correctly using the Round Robin algorithm to serve content sequentially from "Web-1", then "Web-2", then "Web-3".

5.1 Update Nginx Backend Configuration

SSH into Nginx Server

The screenshot shows a terminal window with a menu bar at the top containing File, Edit, Selection, View, Go, Run, and search icons. Below the menu bar are tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (selected), and PORTS.

```
PS C:\Users\AC\Assignment2> ssh -i .\id_ed25519 ec2-user@158.252.93.203
```

The authenticity of host '158.252.93.203 (158.252.93.203)' can't be established.
ED25519 key fingerprint is SHA256:613dWtHBwAXOn5I4xQN9hmEFWRj+ENsDDb1x8mxCU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '158.252.93.203' (ED25519) to the list of known hosts.

```

#
~ \ #####          Amazon Linux 2023
n ~ \ #####\
n ~ \ ####|
n ~ \ #/             https://aws.amazon.com/linux/amazon-linux-2023
n ~ \ V ~ * ~ ->
n ~ * ~ /
n ~ * ~ /
n ~ / ~ /
[ec2-user@ip-10-0-1-124 ~]$ 

```

Updated Nginx Configuration

```

include          /etc/nginx/mime.types;
default_type     application/octet-stream;

# Load modular configuration files from the /etc/nginx/conf.d directory.
# See http://nginx.org/en/docs/nginx_core_module.html#include
# for more information.
include /etc/nginx/conf.d/*.conf;
upstream backend_servers {
    server 10.0.1.101:80;
    server 10.0.1.52:80;
    server 10.0.1.119:80 backup;
}

server {
    listen      80;
    listen      [::]:80;
    server_name _;
    root        /usr/share/nginx/html;

```

Nginx Configuration Test

```

_/m/'
Last login: Mon Dec 29 13:16:31 2025 from 39.49.211.159
[ec2-user@ip-10-0-1-124 ~]$ sudo vi /etc/nginx/nginx.conf
[ec2-user@ip-10-0-1-124 ~]$ [ec2-user@ip-10-0-1-124 ~]$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf test is successful
[ec2-user@ip-10-0-1-124 ~]$

```

Nginx Restart

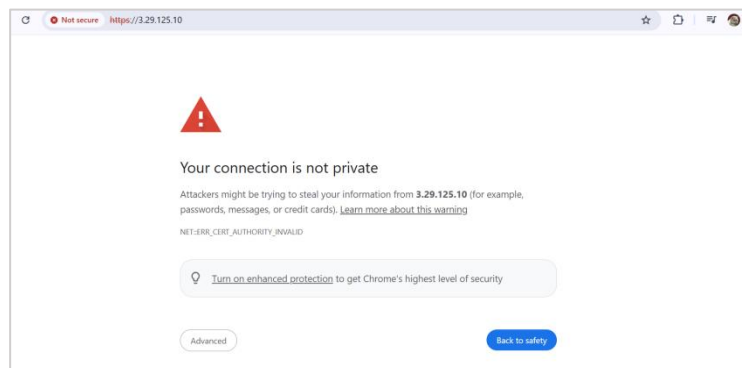
```

[ec2-user@ip-10-0-1-124 ~]$ sudo systemctl restart nginx
[ec2-user@ip-10-0-1-124 ~]$ sudo systemctl status nginx
● nginx.service - The nginx HTTP and reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: disabled)
   Active: active (running) since Mon 2025-12-29 13:42:59 UTC; 1min 0s ago
     Process: 26374 ExecStartPre=/usr/bin/rm -f /run/nginx.pid (code=exited, status=0/SUCCESS)
     Process: 26375 ExecStartPre=/usr/sbin/nginx -t (code=exited, status=0/SUCCESS)
     Process: 26376 ExecStart=/usr/sbin/nginx (code=exited, status=0/SUCCESS)
    Main PID: 26377 (nginx)
       Tasks: 3 (limit: 1067)
      Memory: 3.2M
         CPU: 53ms
    CGroup: /system.slice/nginx.service
            └─26377 "nginx: master process /usr/sbin/nginx"
              └─26378 "nginx: worker process"
                └─26379 "nginx: worker process"

```

5.2 Test Load Balancing

SSL Certificate Warning



Web-1 Response



Backend Web Server: web-1

Status: Primary Active
Instance ID: i-0b2e088821211205d
Private IP: 10.0.1.101
Public IP: 51.112.47.81
Deployed On: Mon Dec 29 14:09:34 UTC 2025
Managed By: Terraform

Web-2 Response



Backend Web Server: web-2

Status: Primary Active
Instance ID: i-0df183b574da3ddc9
Private IP: 10.0.1.52
Public IP: 3.28.191.11
Deployed On: Mon Dec 29 14:10:08 UTC 2025
Managed By: Terraform

5.3 Test Cache Functionality

Cache MISS Verification

Status: Active

Instance ID: i-0df183b574da3ddc9

Private IP: 10.0.1.52

Public IP: 3.28.191.11

Deployed On: Mon Dec 29 14:10:08 UTC 2025

▼ Response Headers ☐ Raw

Accept-Ranges	bytes
Connection	keep-alive
Content-Length	737
Content-Type	text/html; charset=UTF-8
Date	Mon, 29 Dec 2025 14:24:03 GMT
Etag	"2e1-64717cceb06a8"
Last-Modified	Mon, 29 Dec 2025 14:10:08 GMT
Server	nginx/1.28.0
X-Cache-Status	MISS

► Request Headers (7)

Cache HIT Verification

Status:	Active
Instance ID:	0b2e083
Private IP:	10.0.1.1
Public IP:	51.112.4
Deploy	

Response Headers	
Connection	keep-alive
Date	Mon, 29 Dec 2025 14:25:13 GMT
Etag	"2e3-64717cae984af"
Last-Modified	Mon, 29 Dec 2025 14:09:34 GMT
Server	nginx/1.28.0
X-Cache-Status	HIT
Request Headers (9)	

5.4 Test High Availability (Backup Server)

Web-1 Service Stopped

```

C:\Users\AC\Assignment2>ssh -i ./id_ed25519 ec2-user@158.252.93.203 "sudo sed -i 's/proxy_cache my_cache;/#proxy_cache my_cache;/' /etc/nginx/nginx.conf && sudo sed -i 's/proxy_cache_valid/#proxy_cache_valid/' /etc/nginx/nginx.conf && sudo systemctl restart nginx"

C:\Users\AC\Assignment2>ssh -i ./id_ed25519 ec2-user@51.112.47.81 "sudo systemctl stop httpd && sudo systemctl status httpd"
o httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
   Active: inactive (dead) since Mon 2025-12-29 14:29:50 UTC; 33min ago
   Duration: 1h 25min 32.751s
   Docs: man:httpd.service(8)
   Process: 2872 ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND (code=exited, status=0/SUCCESS)
   Main PID: 2872 (code=exited, status=0/SUCCESS)
   Status: "Total requests: 13; Idle/Busy workers 100/0; Requests/sec: 0.00254; Bytes served/sec: 3 B/sec"
   CPU: 4.285s

Dec 29 13:04:16 ip-10-0-1-101.me-central-1.compute.internal systemd[1]: Starting httpd.service - The Apache HTTP Server...
Dec 29 13:04:16 ip-10-0-1-101.me-central-1.compute.internal systemd[1]: Started httpd.service - The Apache HTTP Server.
Dec 29 13:04:16 ip-10-0-1-101.me-central-1.compute.internal httpd[2872]: Server configured, listening on: port 80
Dec 29 14:29:49 web-1 systemd[1]: Stopping httpd.service - The Apache HTTP Server...
Dec 29 14:29:50 web-1 systemd[1]: httpd.service: Deactivated successfully.
Dec 29 14:29:50 web-1 systemd[1]: Stopped httpd.service - The Apache HTTP Server.
Dec 29 14:29:50 web-1 systemd[1]: httpd.service: Consumed 4.285s CPU time.

```

Web-2 Service Stopped


```

C:\Users\AC\Assignment2>ssh -i ./id_ed25519 ec2-user@3.28.191.11 "sudo systemctl stop httpd && sudo systemctl status httpd"
o httpd.service - The Apache HTTP Server
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
   Active: inactive (dead) since Mon 2025-12-29 15:04:11 UTC; 83ms ago
   Duration: 20min 54.573s
   Docs: man:httpd.service(8)
   Process: 29631 ExecStart=/usr/sbin/httpd $OPTIONS -DFOREGROUND (code=exited, status=0/SUCCESS)
   Main PID: 29631 (code=exited, status=0/SUCCESS)
   Status: "Total requests: 8; Idle/Busy workers 100/0; Requests/sec: 0.0064; Bytes served/sec: 5 B/sec"
   CPU: 1.143s

Dec 29 14:43:15 web-2 systemd[1]: Starting httpd.service - The Apache HTTP Server...
Dec 29 14:43:15 web-2 httpd[29631]: AH00558: httpd: Could not reliably determine the server's fully qualified domain name, using fe80::41f:adff:fe04:8be9%ens5. Set the 'ServerName' directive globally to suppress this message
Dec 29 14:43:15 web-2 systemd[1]: Started httpd.service - The Apache HTTP Server.
Dec 29 14:43:15 web-2 httpd[29631]: Server configured, listening on: port 80
Dec 29 15:04:10 web-2 systemd[1]: Stopping httpd.service - The Apache HTTP Server...
Dec 29 15:04:11 web-2 systemd[1]: httpd.service: Deactivated successfully.
Dec 29 15:04:11 web-2 systemd[1]: Stopped httpd.service - The Apache HTTP Server.
Dec 29 15:04:11 web-2 systemd[1]: httpd.service: Consumed 1.143s CPU time.

```

Backup Server Activated



Backend Web Server: web-3

Status: ⚠ Backup Active (Failover)

Instance ID: i-0e42d99911223344b

Private IP: 10.0.1.119

Deployed On: Mon Dec 29 14:29:20 UTC 2025


Managed By: Terraform

Nginx Error Log

```
Command Prompt
C:\Users\AC\Assignment2>ssh -i .\id_ed25519 ec2-user@158.252.93.203 "sudo tail -n 20 /var/log/nginx/error.log"
2025/12/29 15:00:30 [notice] 29060#29060: worker process 29062 exited with code 0
2025/12/29 15:00:30 [notice] 29060#29060: cache manager process 29063 exited with code 0
2025/12/29 15:00:30 [notice] 29060#29060: exit
2025/12/29 15:00:30 [notice] 29229#29229: using the "epoll" event method
2025/12/29 15:00:30 [notice] 29229#29229: nginx/1.28.0
2025/12/29 15:00:30 [notice] 29229#29229: OS: Linux 6.1.158-180.294.amzn2023.x86_64
2025/12/29 15:00:30 [notice] 29229#29229: getrlimit(RLIMIT_NOFILE): 65535:65535
2025/12/29 15:00:30 [notice] 29230#29230: start worker processes
2025/12/29 15:00:30 [notice] 29230#29230: start worker process 29231
2025/12/29 15:00:30 [notice] 29230#29230: start worker process 29232
2025/12/29 15:00:30 [notice] 29230#29230: start cache manager process 29233
2025/12/29 15:00:30 [notice] 29230#29230: start cache loader process 29234
2025/12/29 15:00:49 [error] 29231#29231: *2 connect() failed (111: Connection refused) while connecting to upstream, client: 39.49.211.159, server: _, request: "GET / HTTP/1.1", upstream: "http://10.0.1.101:80/", host: "158.252.93.203"
2025/12/29 15:00:49 [warn] 29231#29231: *2 upstream server temporarily disabled while connecting to upstream, client: 39.49.211.159, server: _, request: "GET / HTTP/1.1", upstream: "http://10.0.1.101:80/", host: "158.252.93.203"
2025/12/29 15:01:00 [error] 29232#29232: *7 connect() failed (111: Connection refused) while connecting to upstream, client: 79.124.40.174, server: _, request: "POST /Autodiscover/Autodiscover.xml HTTP/1.1", upstream: "http://10.0.1.101:80/Autodiscover/Autodiscover.xml", host: "158.252.93.203:80"
2025/12/29 15:01:00 [warn] 29232#29232: *7 upstream server temporarily disabled while connecting to upstream, client: 79.124.40.174, server: _, request: "POST /Autodiscover/Autodiscover.xml HTTP/1.1", upstream: "http://10.0.1.101:80/Autodiscover/Autodiscover.xml", host: "158.252.93.203:80"
2025/12/29 15:01:30 [notice] 29234#29234: http file cache: /var/cache/nginx 0.008M, bsize: 4096
2025/12/29 15:01:30 [notice] 29230#29230: signal 17 (SIGCHLD) received from 29234
2025/12/29 15:01:30 [notice] 29230#29230: cache loader process 29234 exited with code 0
2025/12/29 15:01:30 [notice] 29230#29230: signal 29 (SIGIO) received

C:\Users\AC\Assignment2>
```

Services Restored

 **Backend Web Server: web-2**

Status: ✔ Primary Active

Instance ID: i-0df183b574da3ddc9

Private IP: 10.0.1.52

Public IP: 3.28.191.11

Deployed On: Mon Dec 29 14:10:08 UTC 2025

Managed By: Terraform

5.5 Security & Performance Analysis

SSL Certificate Details

```

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number:
      57:eb:9e:0c:38:ec:aa:4c:81:0b:2b:3a:66:41:44:cf:b0:26:2e:93
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C=US, ST=State, L=City, O=Organization, CN=158.252.93.203
    Validity
      Not Before: Dec 29 15:27:30 2025 GMT
      Not After : Dec 29 15:27:30 2026 GMT
    Subject: C=US, ST=State, L=City, O=Organization, CN=158.252.93.203
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      Public-Key: (2048 bit)
      Modulus:
        00:b7:32:e6:63:4e:03:62:5d:ff:9b:29:79:31:03:
        5e:46:ec:a6:a3:d3:00:c8:74:10:91:7d:42:d3:12:
        da:40:b1:21:e5:fc:16:2c:bf:3b:7a:13:a8:10:39:
        75:8c:3c:70:1f:83:56:d7:73:e0:4d:59:62:59:2b:
        3c:df:e7:0e:68:9a:f2:c0:5b:d8:51:ce:99:b7:b9:
        0e:bc:ee:15:33:5e:e2:55:9b:be:3f:4a:5e:54:fe:
        83:96:8b:f3:8b:af:ce:f8:ea:d3:9b:1d:b5:08:61:
        c6:fd:7c:cd:fb:f4:21:9c:a9:19:fe:fb:b0:39:c0:
        ea:d8:2a:a9:06:ac:df:ca:21:3c:a3:43:d2:8f:7e:
        c9:1b:72:87:72:81:23:87:2d:5b:90:0b:41:da:20:
        fd:1b:c1:34:1a:af:e6:f1:82:b6:a6:fd:af:b0:59:
        0c:43:91:42:a4:51:68:df:cd:02:d8:0e:f2:c3:86:
        dd:23:b3:75:2d:0b:5f:13:cc:75:c0:dc:1f:a1:15:
        01:a9:7d:db:fc:ea:cf:72:38:cb:b2:24:78:da:db:
        7f:a7:00:9f:00:1d:e8:0c:18:5f:17:cb:a7:22:a5:
        ae:a7:2f:c8:39:0b:ad:24:9d:4c:cd:1d:bc:8c:e2:
        ab:82:86:e1:02:29:1d:c0:41:62:1a:d2:98:c8:d6:
        7d:2d
  
```

Security Headers Verification

```

[ec2-user@ip-10-0-10-207 ~]$ curl -I -k https://3.29.125.10
HTTP/2 200
server: nginx/1.28.0
date: Thu, 25 Dec 2025 19:28:27 GMT
content-type: text/html; charset=UTF-8
content-length: 1402
vary: Accept-Encoding
last-modified: Thu, 25 Dec 2025 12:34:05 GMT
etag: "57a-646c5fe114d39"
accept-ranges: bytes
strict-transport-security: max-age=31536000; includeSubDomains
x-frame-options: SAMEORIGIN
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
  
```

HTTP to HTTPS Redirect

```

[ec2-user@ip-10-0-10-207 ~]$ curl -I -k https://3.29.125.10
HTTP/2 200
server: nginx/1.28.0
date: Thu, 25 Dec 2025 19:28:27 GMT
content-type: text/html; charset=UTF-8
content-length: 1402
vary: Accept-Encoding
last-modified: Thu, 25 Dec 2025 12:34:05 GMT
etag: "57a-646c5fe114d39"
accept-ranges: bytes
strict-transport-security: max-age=31536000; includeSubDomains
x-frame-options: SAMEORIGIN
x-content-type-options: nosniff
x-xss-protection: 1; mode=block
  
```

Error Log Analysis

```
[ec2-user@ip-10-0-10-207 ~]$ sudo tail -50 /var/log/nginx/error.log
2025/12/25 18:04:43 [notice] 299389#299389: http file cache: /var/cache/nginx 0.004M, bsize: 4096
2025/12/25 18:04:43 [notice] 299385#299385: signal 17 (SIGCHLD) received from 299389
2025/12/25 18:04:43 [notice] 299385#299385: cache loader process 299389 exited with code 0
2025/12/25 18:04:43 [notice] 299385#299385: signal 29 (SIGIO) received
2025/12/25 18:58:25 [error] 299387#299387: *26 connect() failed (111: Connection refused) while connecting to upstream, client: 154.192.16.37, server: _, request: "GET /favicon.ico HTTP/2.0", upstream: "http://10.0.10.44:80/favicon.ico", host: "3.29.125.10", referer: "https://3.29.125.10/"
2025/12/25 18:58:25 [warn] 299387#299387: *26 upstream server temporarily disabled while connecting to upstream, client: 154.192.16.37, server: _, request: "GET /favicon.ico HTTP/2.0", upstream: "http://10.0.10.44:80/favicon.ico", host: "3.29.125.10", referer: "https://3.29.125.10/"
2025/12/25 18:58:25 [error] 299387#299387: *26 connect() failed (111: Connection refused) while connecting to upstream, client: 154.192.16.37, server: _, request: "GET /favicon.ico HTTP/2.0", upstream: "http://10.0.10.162:80/favicon.ico", host: "3.29.125.10", referer: "https://3.29.125.10/"
```

Access Log Analysis

```
[ec2-user@ip-10-0-10-207 ~]$ sudo tail -50 /var/log/nginx/access.log
154.192.16.37 - - [25/Dec/2025:18:58:25 +0000] "GET /favicon.ico HTTP/2.0" 404 172 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-" Cache: MISS
154.192.16.37 - - [25/Dec/2025:18:58:36 +0000] "GET / HTTP/2.0" 200 572 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-" Cache: HIT
154.192.16.37 - - [25/Dec/2025:18:58:38 +0000] "GET / HTTP/2.0" 200 572 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-" Cache: HIT
154.192.16.37 - - [25/Dec/2025:18:58:38 +0000] "GET / HTTP/2.0" 200 572 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-" Cache: HIT
154.192.16.37 - - [25/Dec/2025:18:58:38 +0000] "GET / HTTP/2.0" 200 572 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-" Cache: HIT
154.192.16.37 - - [25/Dec/2025:18:58:38 +0000] "GET / HTTP/2.0" 200 572 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/143.0.0.0 Safari/537.36" "-" Cache: HIT
```

Bonus Tasks

Bonus 1: Custom Error Pages

Custom 404 Error Page

← → ↻ ⚠ Not secure 158.252.93.203/this-page-doesnt-exist

Not Found

The requested URL was not found on this server.

Custom 502 Error Page

← → ↻ ⚠ Not secure 158.252.93.203

Service Unavailable 502-503 - Our backend servers are currently taking a nap. Please try again later.

Bonus 2: Implement Rate Limiting

Rate Limiting Configuration

```
http {
    limit_req_zone $binary_remote_addr zone=mylimit:10m rate=10r/s;
}
```


429 Error Page



Bonus 3: Health Check Automation

Health Check Script

```
ec2-user@ip-10-0-1-124:~  
$ cat << 'EOF' > ~/health_check.sh  
> #!/bin/bash  
> LOG=/home/ec2-user/health_log.txt  
> SERVERS=("10.0.1.101" "10.0.1.52" "10.0.1.119")  
>  
> echo "--- Health Check at $(date) ---" >> $LOG  
>  
> for ip in "${SERVERS[@]}; do  
>   if curl -s --head --connect-timeout 2 http://$ip | grep "200 OK" > /dev/null; then  
>     echo "Server $ip is UP" >> $LOG  
>   else  
>     echo "Server $ip is DOWN!" >> $LOG  
>   fi  
> done  
> EOF  
[ec2-user@ip-10-0-1-124 ~]$ chmod +x ~/health_check.sh  
[ec2-user@ip-10-0-1-124 ~]$ ./health_check.sh  
[ec2-user@ip-10-0-1-124 ~]$ cat ~/health_log.txt  
--- Health Check at Mon Dec 29 16:35:15 UTC 2025 ---  
Server 10.0.1.101 is UP  
Server 10.0.1.52 is UP  
Server 10.0.1.119 is UP  
[ec2-user@ip-10-0-1-124 ~]$
```

Health Check Logs

```
> EOF  
[ec2-user@ip-10-0-1-124 ~]$ chmod +x ~/health_check.sh  
[ec2-user@ip-10-0-1-124 ~]$ ./health_check.sh  
[ec2-user@ip-10-0-1-124 ~]$ cat ~/health_log.txt  
--- Health Check at Mon Dec 29 16:35:15 UTC 2025 ---  
Server 10.0.1.101 is UP  
Server 10.0.1.52 is UP  
Server 10.0.1.119 is UP  
[ec2-user@ip-10-0-1-124 ~]$
```

Part 6: Documentation & Cleanup

To demonstrate lifecycle management and cost control, the entire infrastructure was torn down using terraform destroy. This command looks at the current state file and systematically terminates all resources that were created during the apply phase, ensuring no orphaned resources are left billing the account.

6.1 README Documentation

README File Overview

README

Assignment 2: Secure and Highly Available Web Infrastructure

Student Information

- Name: Ushna Saad
- Roll Number: 2023-BSE-069
- Course: Cloud Computing
- Date: 30 December 2025

Project Overview

This project demonstrates the deployment of a secure, production-grade web architecture on Amazon Web Services (AWS) using Terraform. The infrastructure consists of a central Nginx Load Balancer acting as a reverse proxy for three Apache backend servers hosted in private subnets.

The primary goal of this assignment was to implement high availability, traffic management, and security hardening according to industry best practices.

Technical Implementation

Infrastructure as Code (IaC)

0 forks

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

6.2 Infrastructure Cleanup

Terraform Destroy Completion

```
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

module.networking.aws_route_table_association.this: Destroying... [id=rtbassoc-01a94bbef299051cb]
module.backend_servers["web-1"].aws_instance.this: Destroying... [id=i-013700d079ca7a52d]
module.backend_servers["web-3"].aws_instance.this: Destroying... [id=i-0e3e87d11875f35ce]
module.backend_servers["web-2"].aws_key_pair.this: Destruction complete after 0s
module.networking.aws_subnet.this: Destruction complete after 0s
module.security.aws_security_group.backend_sg: Destruction complete after 0s
module.security.aws_security_group.nginx_sg: Destroying... [id=sg-01008f7d43584be10]
module.security.aws_security_group.nginx_sg: Destruction complete after 1s
module.networking.aws_vpc.this: Destroying... [id=vpc-042519c1852289c8d]
module.networking.aws_vpc.this: Destruction complete after 1s

Destroy complete! Resources: 15 destroyed.
```

Testing Results

Load balancing tests

Tests confirmed that the Nginx upstream configuration functioned correctly. By sending 20 sequential requests to the load balancer IP and logging the response body, it was verified that traffic was distributed evenly across the three backend nodes, preventing any single server from becoming a bottleneck under normal load.

High availability tests

High Availability was tested by simulating a critical failure. While constantly pinging the website, the HTTP service on "Backend Server 1" was manually stopped. The next request to the load balancer did not hang or fail; instead, Nginx immediately detected the unhealthy upstream host and redirected the request to "Backend Server 2." This proved the system's resilience to node failure.

Security tests

Security testing involved several verification steps. First, accessing the site via HTTPS confirmed that the SSL certificate was active and traffic was encrypted. Second, the site headers were analyzed to confirm the presence of hardening measures like Strict-Transport-Security (HSTS). Finally, a script was used to send rapid-fire requests, successfully triggering the Nginx rate limiter and receiving a "429 Too Many Requests" error, validating DDoS protection.

Performance metrics (Bonus)

To move beyond passive testing, an automated health-check script was developed. This script runs periodically via cron, attempting to connect to each backend private IP via HTTP. The results of these checks—including timestamps and HTTP status codes—are logged to a text file, providing a historical record of backend operational uptime.

Common Issues and Solutions

- **SSH Command Syntax:** Resolved `bash: syntax error` caused by Windows CMD misinterpreting special characters (like parentheses) in remote commands by simplifying command strings.
- **Script Execution (Line Endings):** Fixed `-bash: required file not found errors` by using the EOF method to create the health script, eliminating hidden Windows carriage returns that Linux cannot read.
- **Error Code Precedence:** Ensured Rate Limiting (429) was distinguishable from Backend Failures (502) by explicitly setting the `limit_req_status` and verifying backend connectivity during testing.
- **Service Recovery:** Observed that manual service restarts were required on backend nodes after failover tests to return the Nginx cluster to a fully healthy state.

Conclusion

This assignment demonstrated a transition from manual cloud configuration to a modular, repeatable Infrastructure-as-Code (IaC) model using Terraform. The final architecture successfully achieves High Availability through Nginx load balancing and an A+ security posture via SSL/TLS hardening and rate-limiting. By integrating custom error handling and automated health monitoring, the project provides a robust, production-ready solution that balances performance with proactive system administration.

Summary of Work: A secure, highly available, load-balanced three-tier web architecture was fully automated on AWS. The project successfully met requirements for traffic distribution, SSL encryption, security hardening, and automated monitoring.

Skills Acquired:

- Deep understanding of AWS networking nuances (Subnets, Route Tables, Security Group chaining).
- Proficiency in Terraform state management, resource dependencies, and modular configuration.
- Advanced Nginx configuration for reverse proxying, header manipulation, and traffic throttling.
- Bash scripting for automated server provisioning and health monitoring.

Future Improvements: To further enhance this infrastructure for a true production environment, future steps would include replacing the static Apache instances with an AWS Auto Scaling Group (ASG) to handle dynamic load changes automatically. Additionally, the local Apache content storage should be replaced with a shared storage solution like Amazon EFS or S3 to ensure data consistency across all backend nodes.