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

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**.

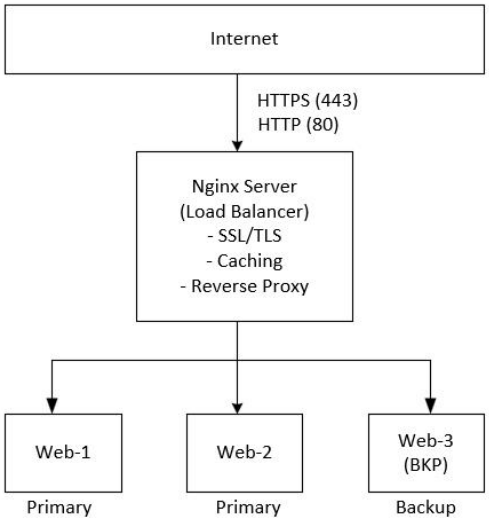
- Achieved an **A+ Security Rating** through SSL/TLS and hardened Nginx headers.
- Implemented **High Availability** with automated failover and Round Robin load balancing.
- Developed **custom health monitoring** and traffic rate-limiting to mitigate DDoS risks.

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.



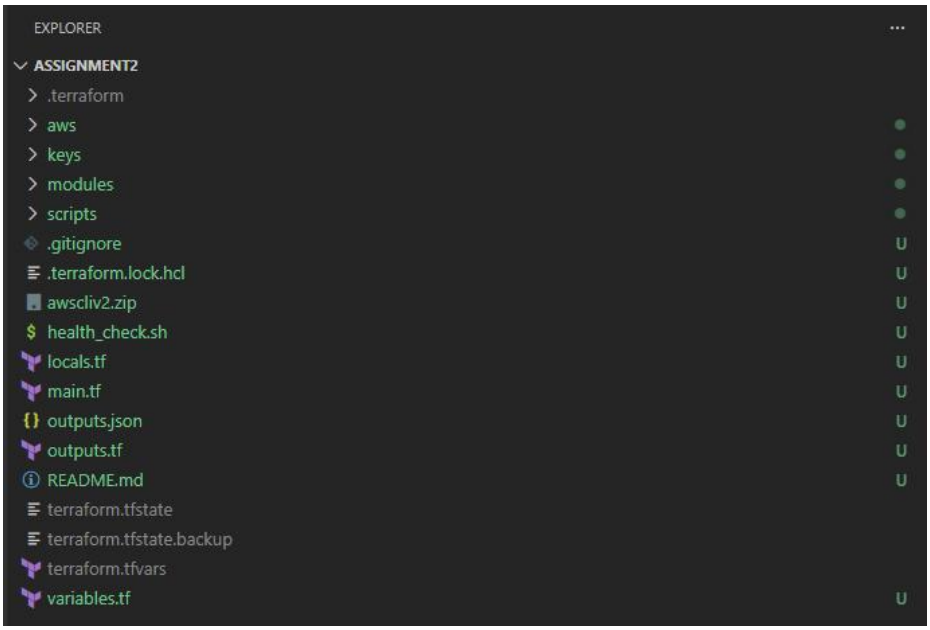
Part 1: Infrastructure Setup

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.

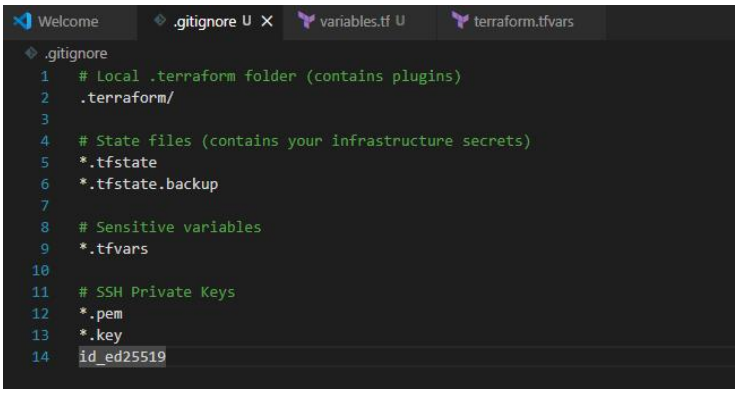
1.1 Project Structure

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

Project Directory Structure



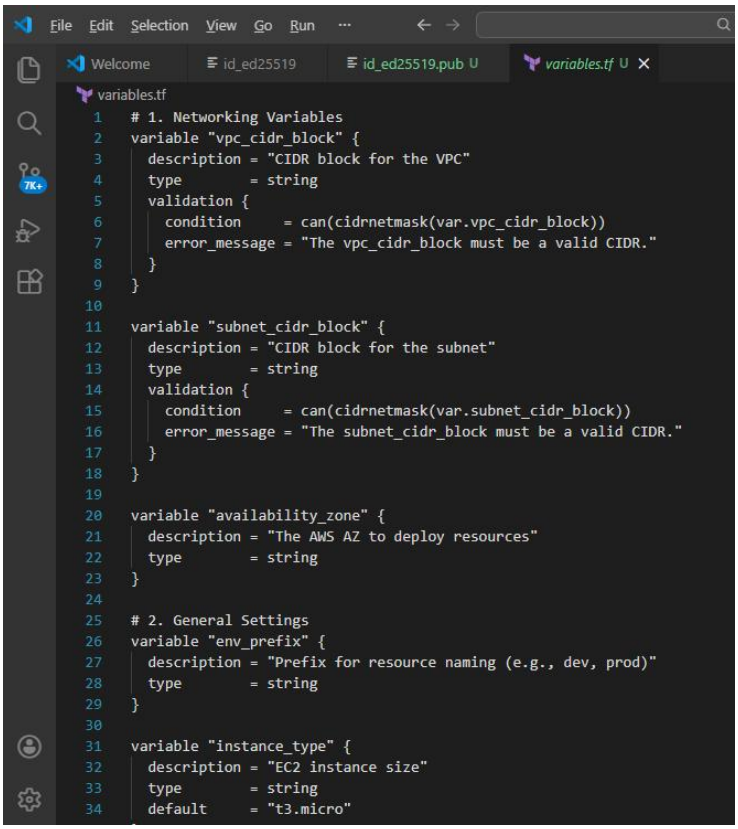
.gitignore Configuration



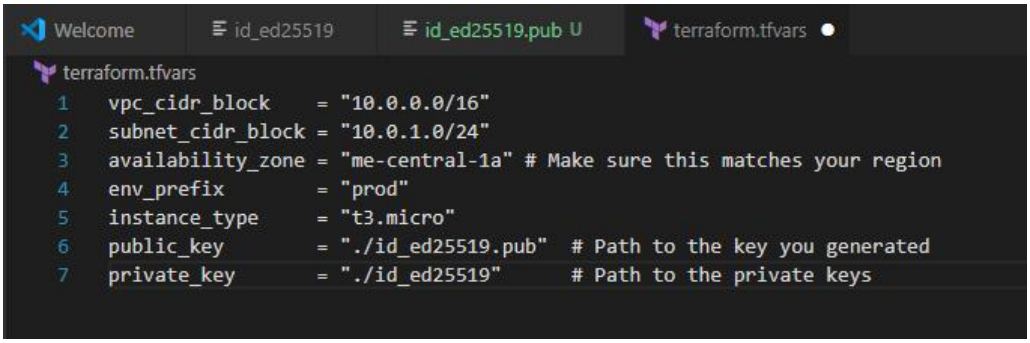
1.2 Variable Configuration

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

Terraform Variables Definition



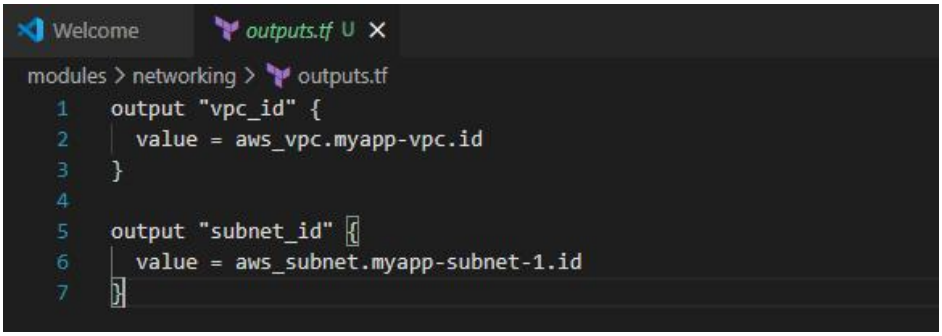
Terraform Variables Values (terraform.tfvars)



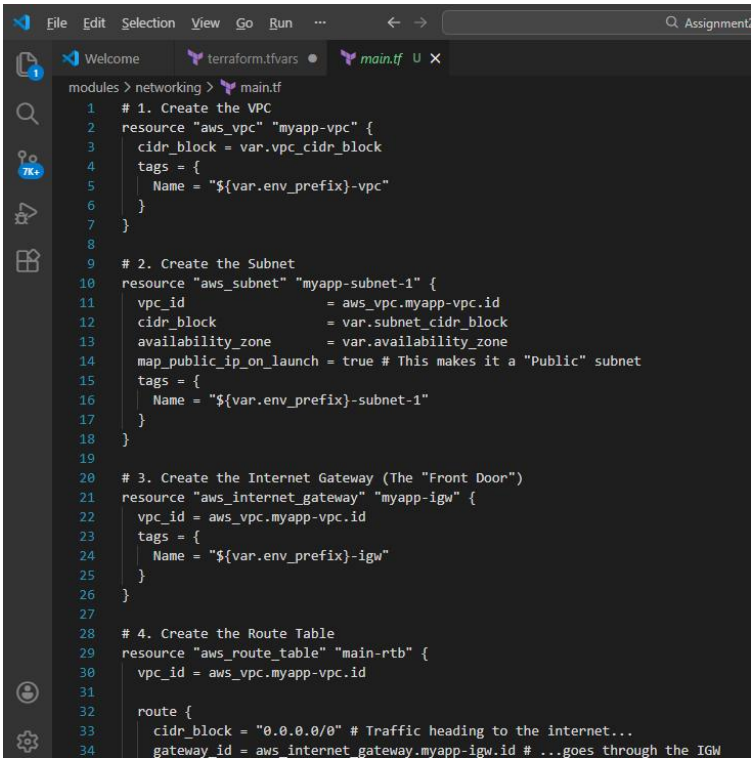
1.3 Networking Module

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

Networking Module Outputs



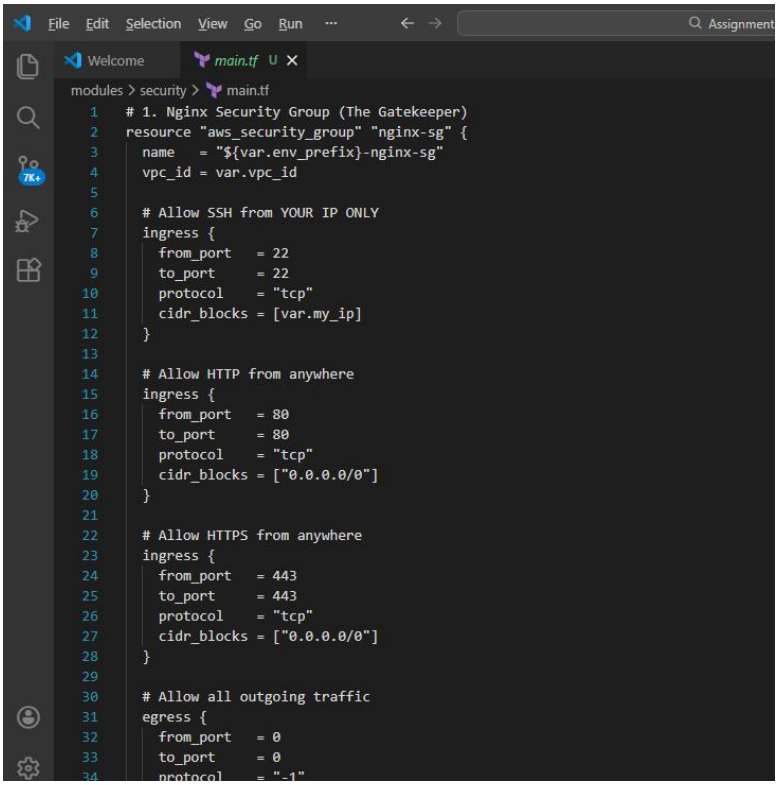
Networking Module Configuration



1.4 Security Module

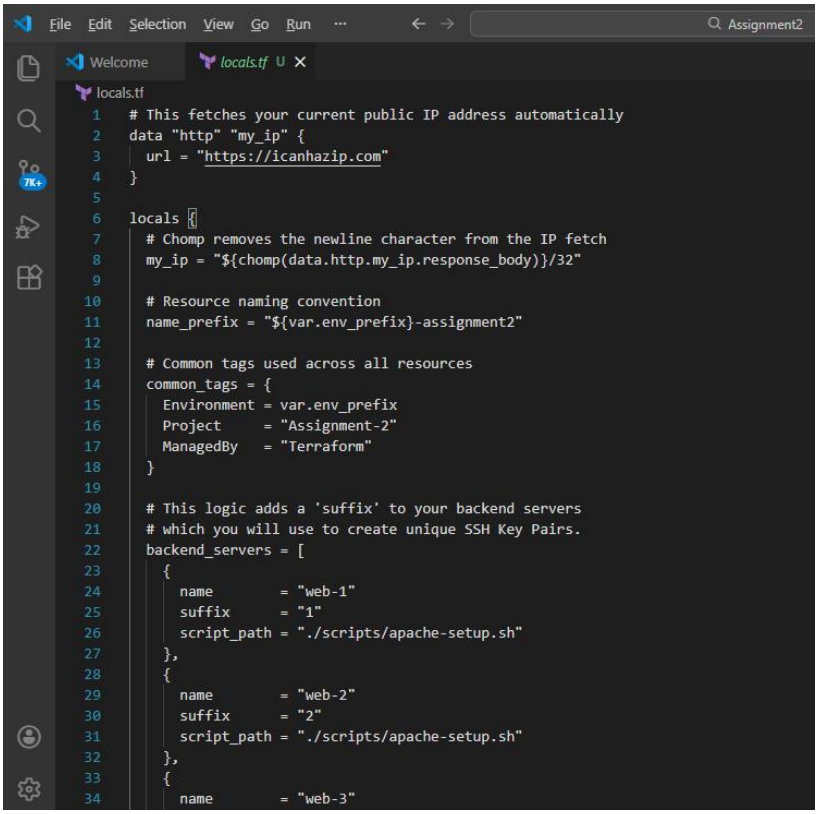
Separate security groups are created for Nginx and backend servers to follow least privilege.

Security Module Configuration



1.5 Locals Configuration

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



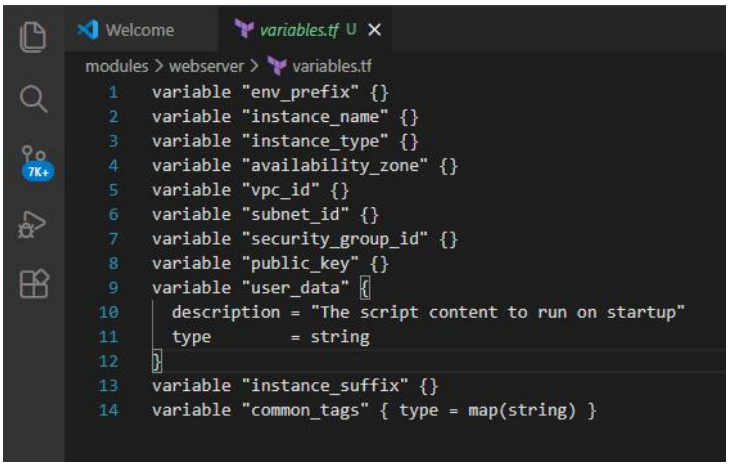
Part 2: Webserver Module

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.

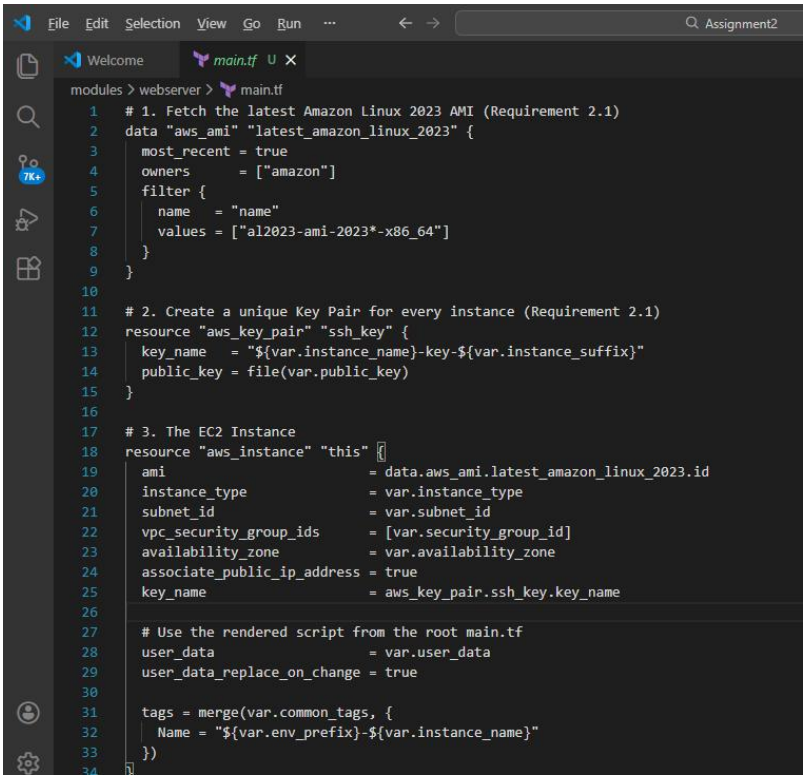
2.1 Module Design

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

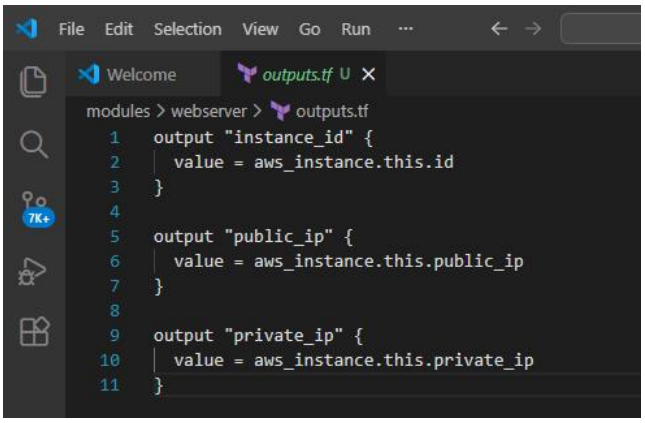
Webserver Module Variables



Webserver Module Resources



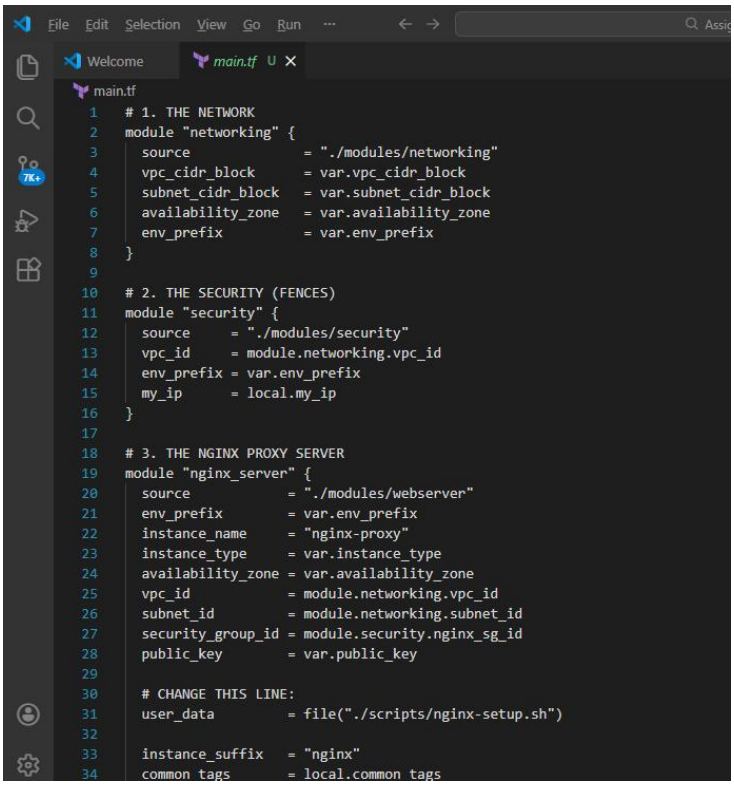
Webserver Module Outputs



2.2 Module Usage

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

Root Module Webserver Integration



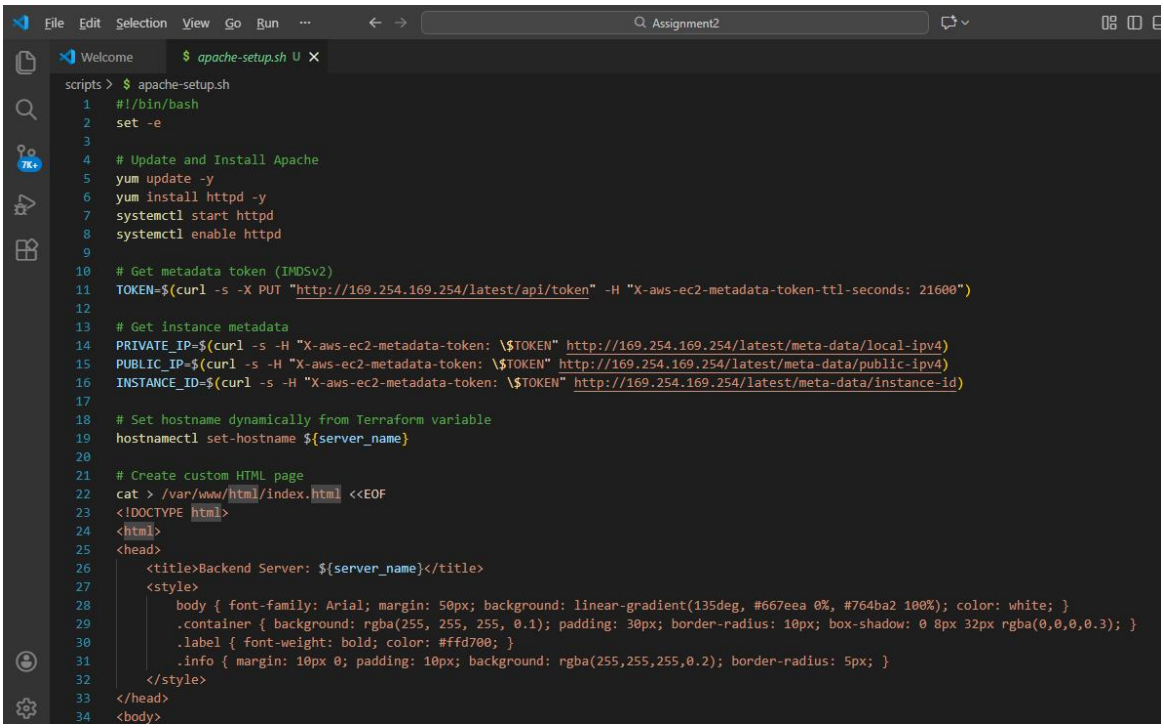
Part 3: Server Configuration Scripts

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.

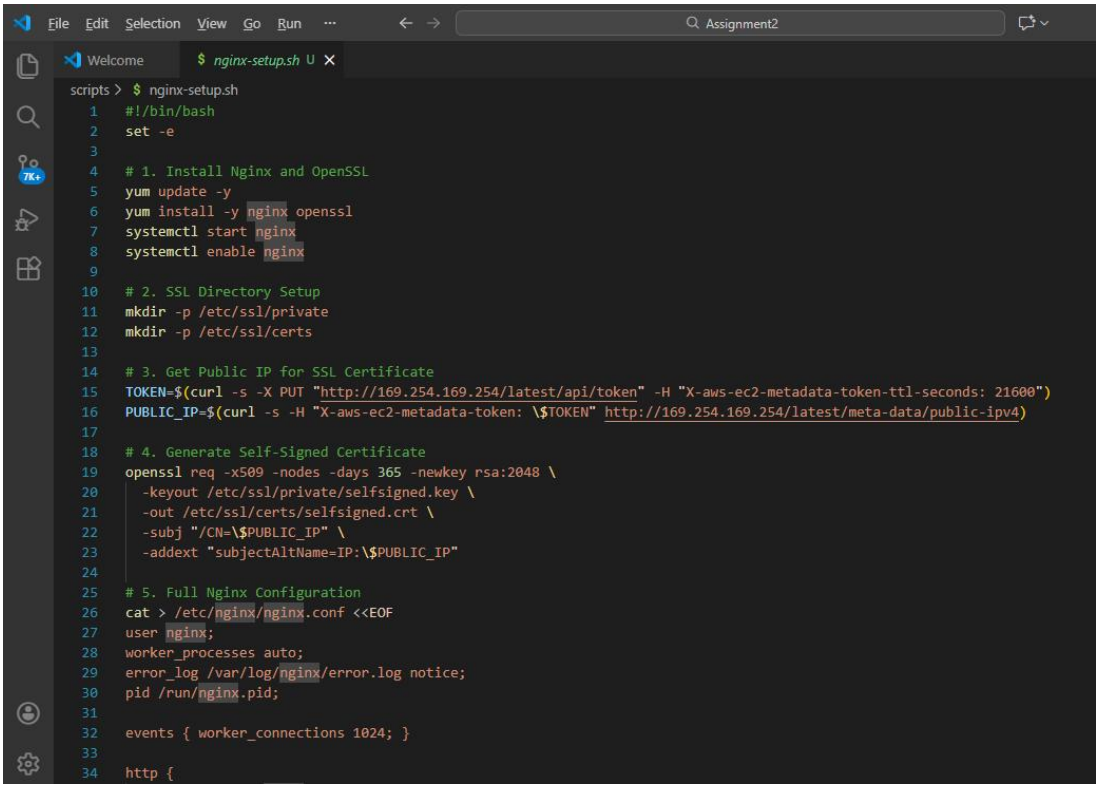
Apache Setup Script



3.2 Nginx Server Setup Script

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

Nginx Setup Script



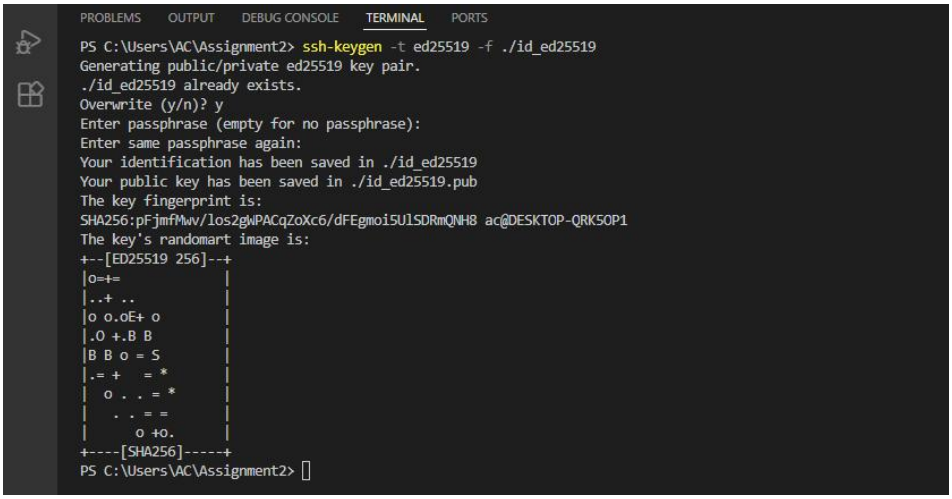
Part 4: Infrastructure Deployment

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).

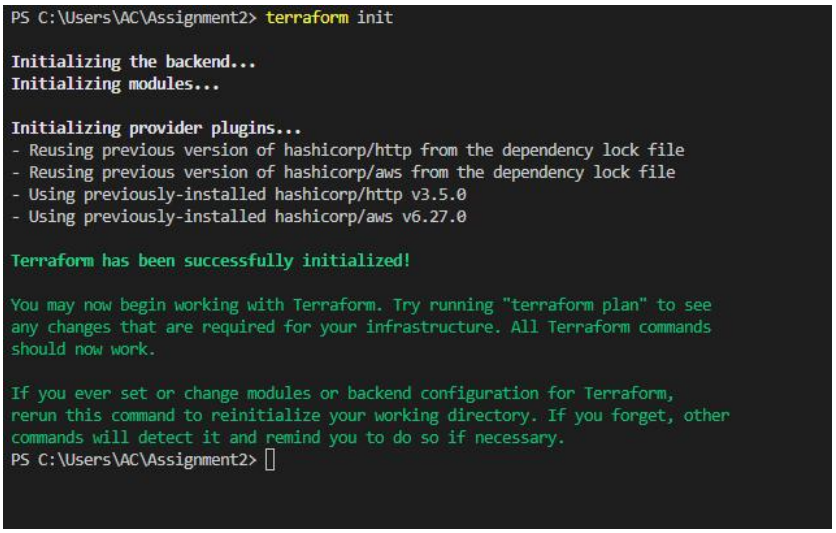
4.1 Initial Deployment

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

SSH Key Generation



Terraform Initialization



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.

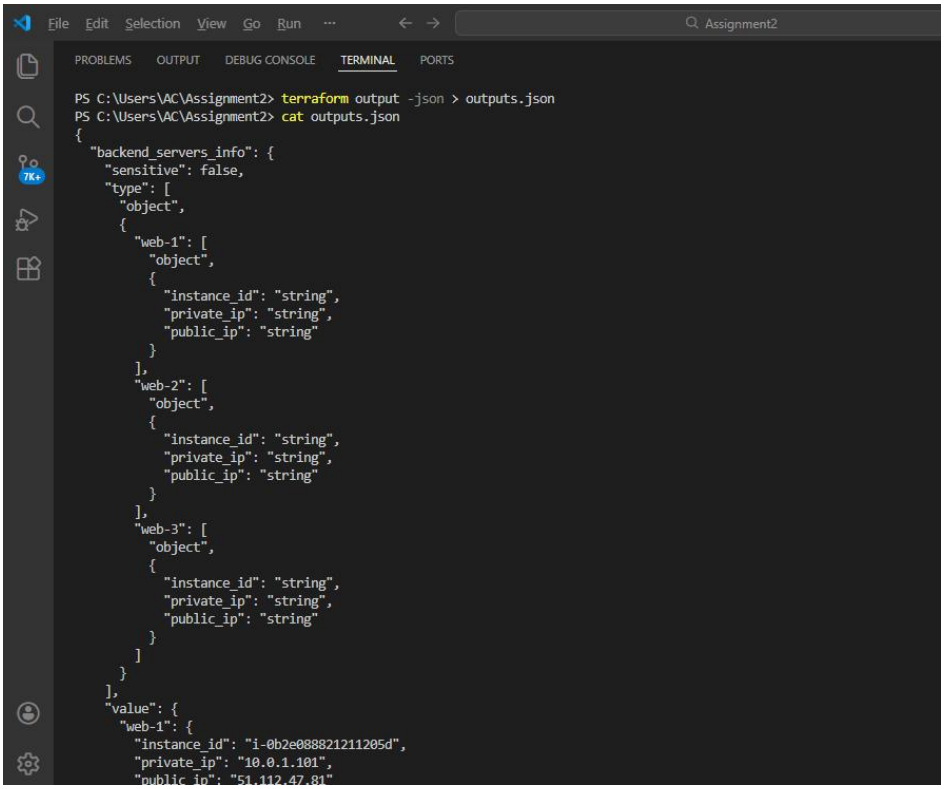
Terraform Output Display

```
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! 🚀
==========================================
```

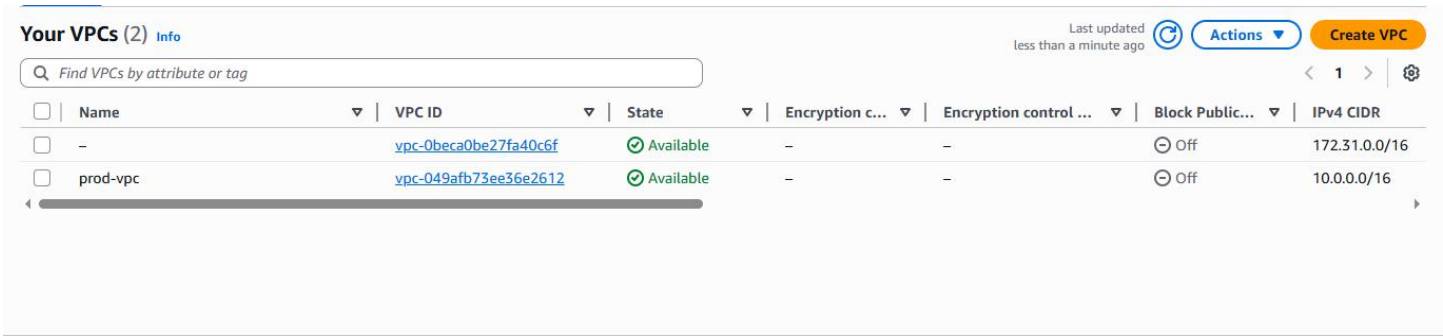
Terraform Outputs JSON File



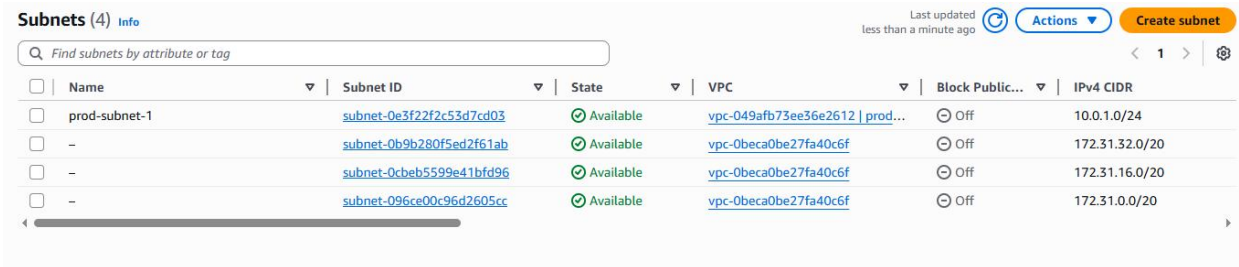
4.3 AWS Console Verification

All resources were verified in the AWS Console.

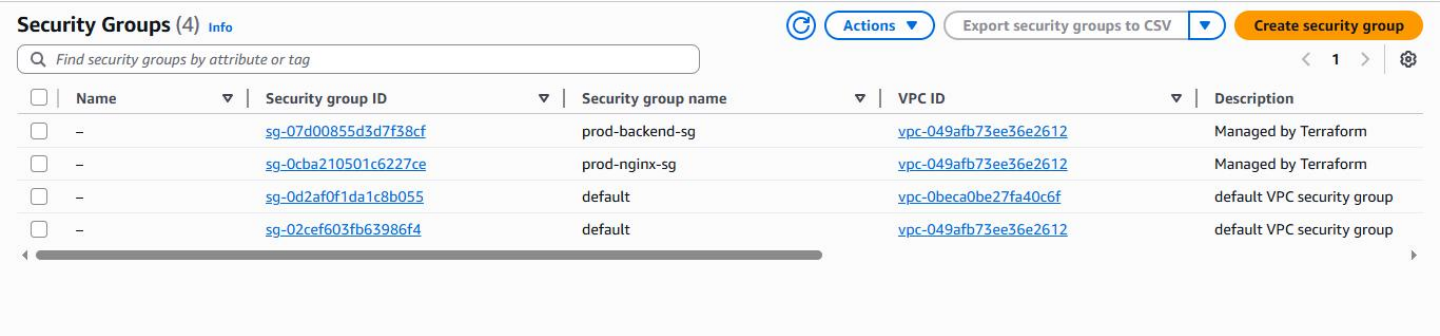
AWS VPC Verification



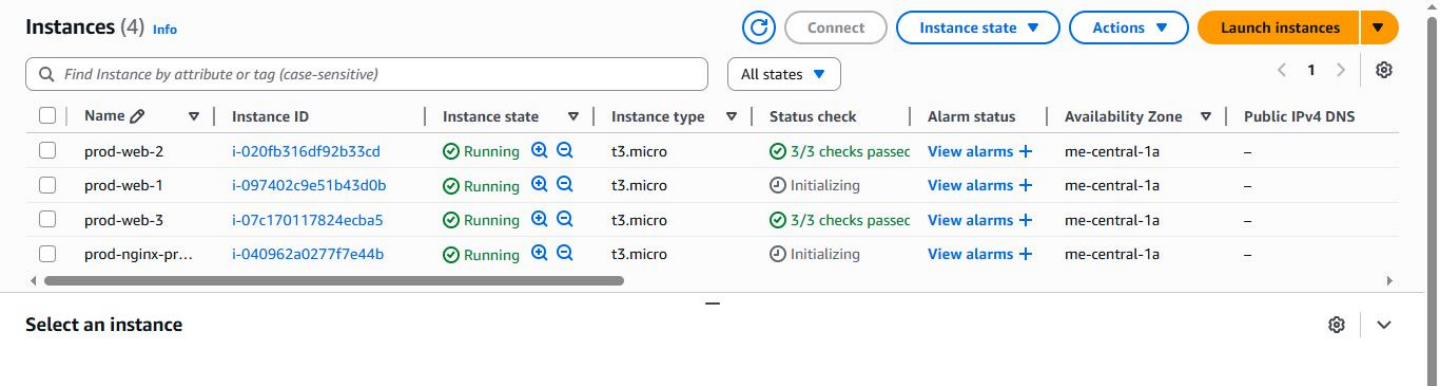
AWS Subnet Verification



AWS Security Groups Verification



AWS EC2 Instances Verification

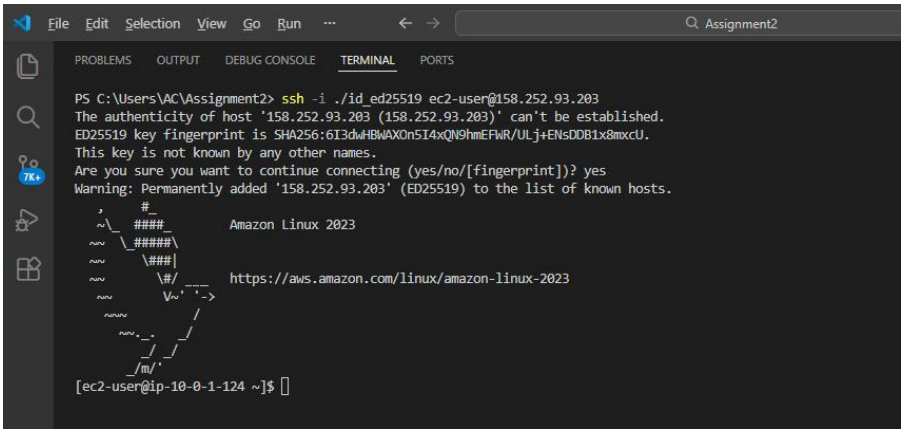


Part 5: Nginx Configuration & Testing

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



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/ngx_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

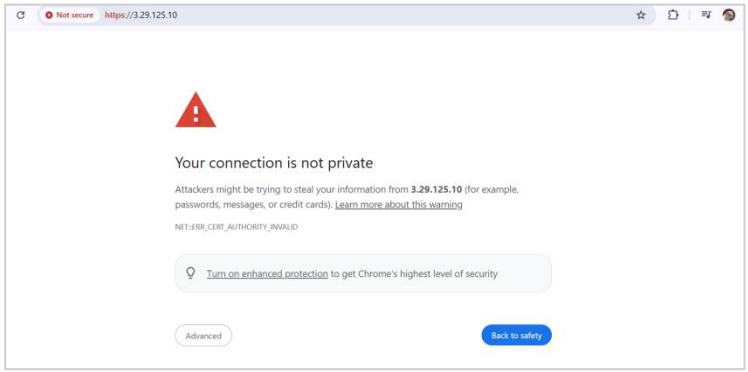
```
       _/m/'
Last login: Mon Dec 29 13:16:31 2025 from 39.49.211.159
[ec2-user@ip-10-0-1-124 ~]$ sudo vi /etc/nginx/nginx.conf
[ec2-user@ip-10-0-1-124 ~]$ [ec2-user@ip-10-0-1-124 ~]$ sudo nginx -t
nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
nginx: configuration file /etc/nginx/nginx.conf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

Nginx Restart

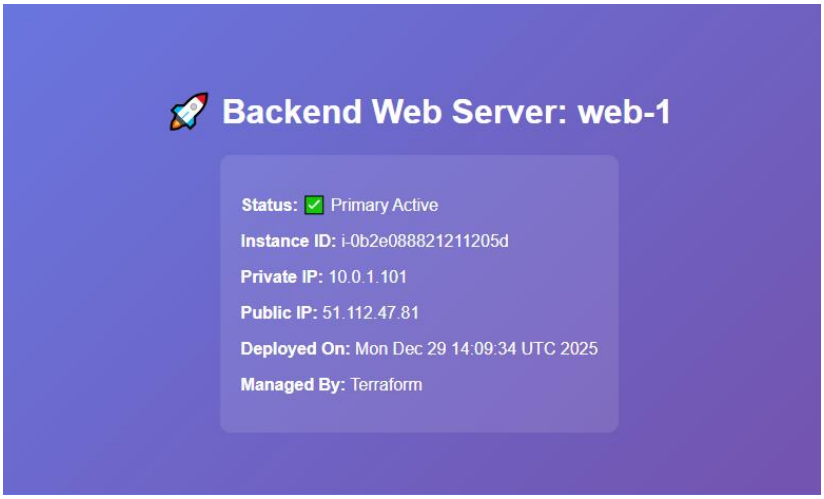
```
[ec2-user@ip-10-0-1-124 ~]$ sudo systemctl restart nginx
[ec2-user@ip-10-0-1-124 ~]$ sudo systemctl status nginx
● nginx.service - The nginx HTTP and reverse proxy server
     Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: disabled)
     Active: active (running) since Mon 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

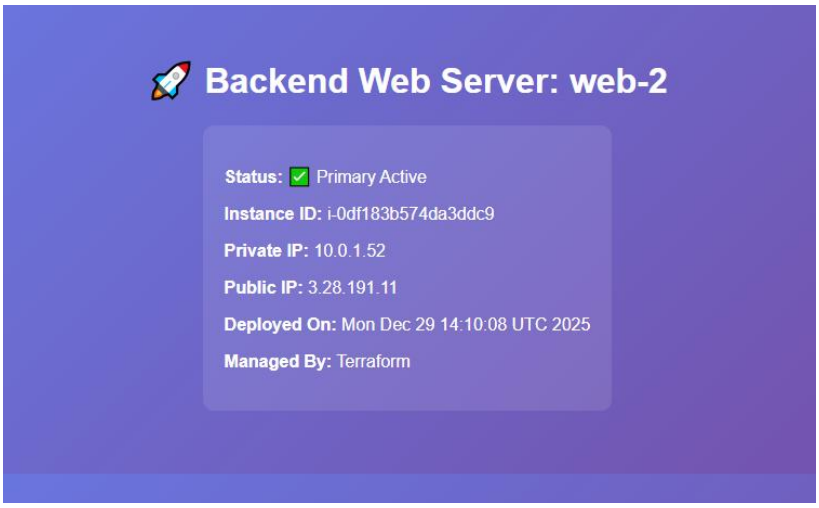
SSL Certificate Warning



Web-1 Response

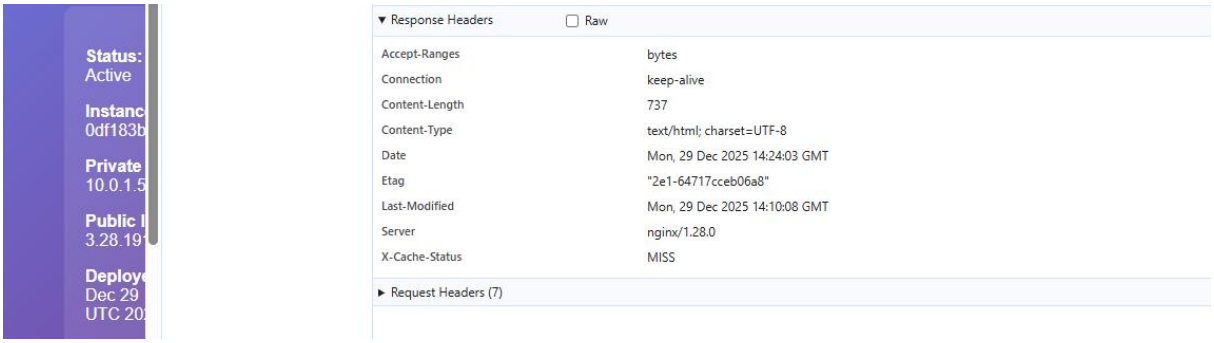


Web-2 Response

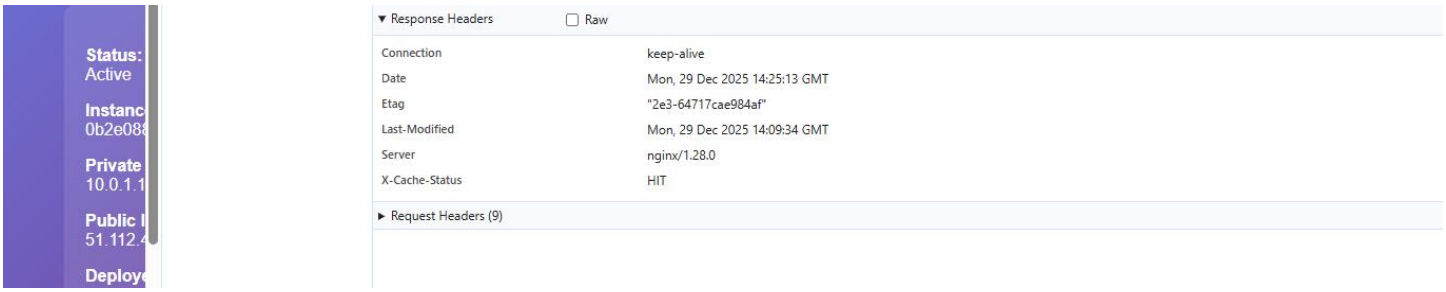


5.3 Test Cache Functionality

Cache MISS Verification

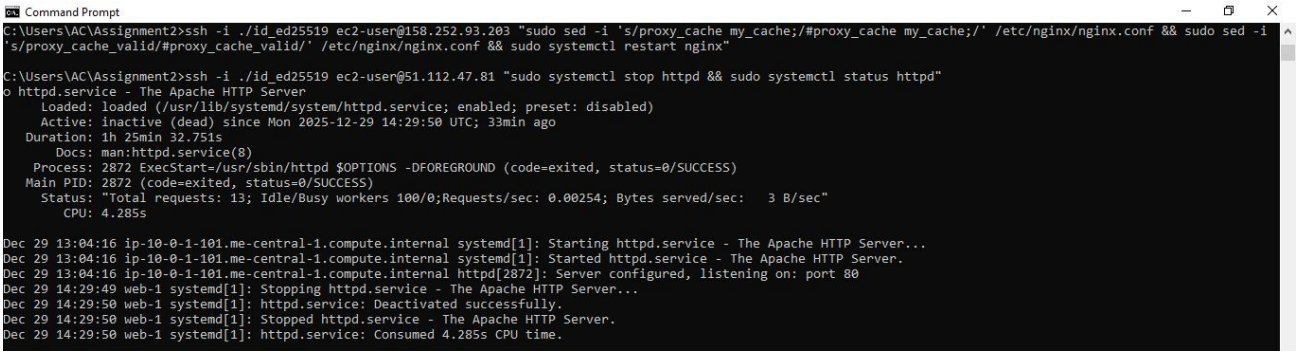


Cache HIT Verification

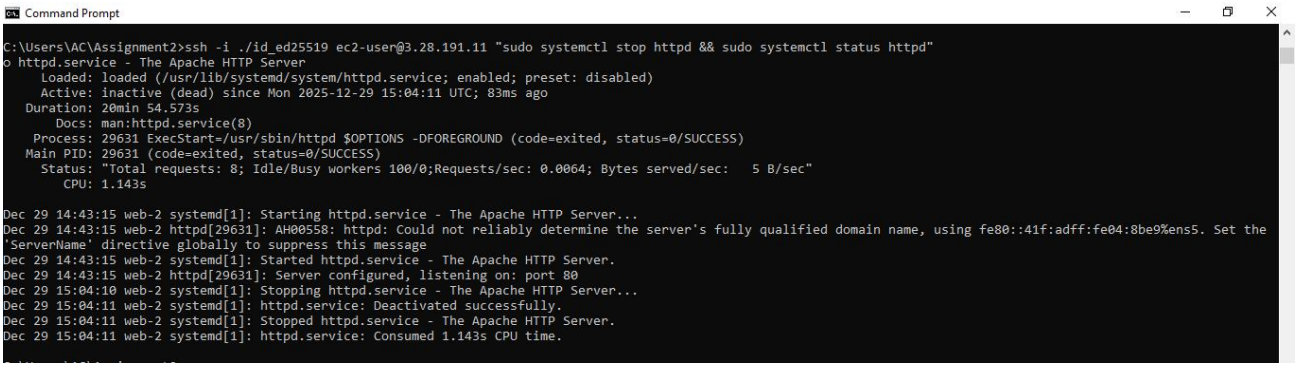


5.4 Test High Availability (Backup Server)

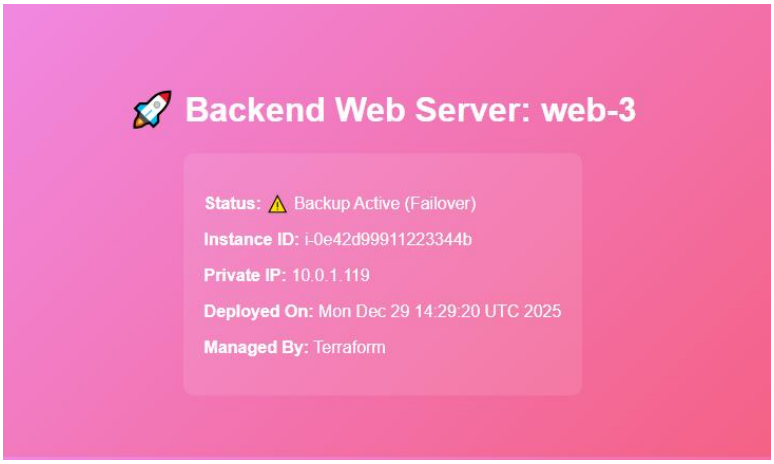
Web-1 Service Stopped



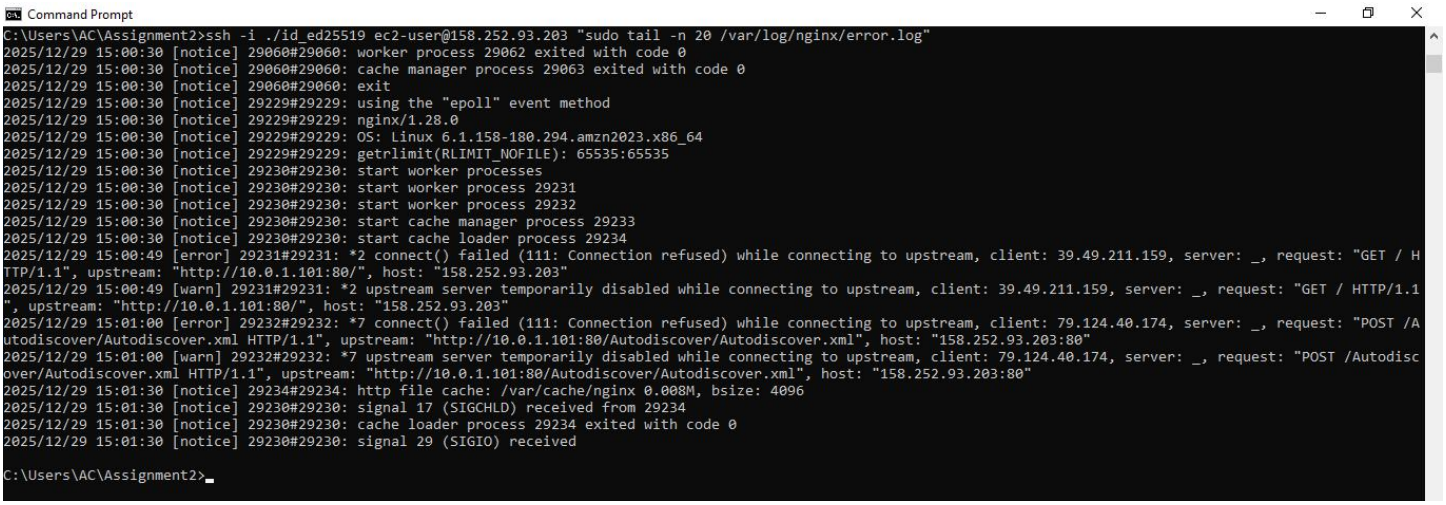
Web-2 Service Stopped



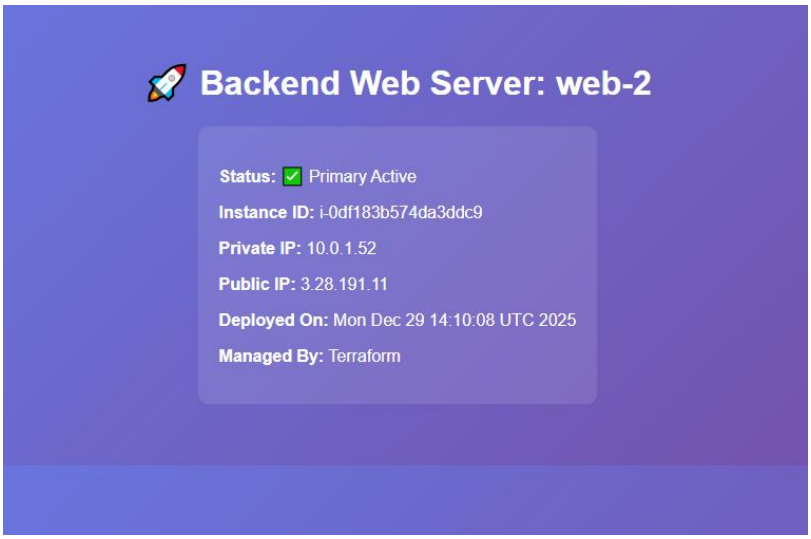
Backup Server Activated



Nginx Error Log



Services Restored



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: _, re
quest: "GET /favicon.ico HTTP/2.0", upstream: "http://10.0.10.44:80/favicon.ico", host: "3.29.125.10", referrer: "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", referrer: "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: _, re
quest: "GET /favicon.ico HTTP/2.0", upstream: "http://10.0.10.162:80/favicon.ico", host: "3.29.125.10", referrer: "https://3.29.125.10/"
```

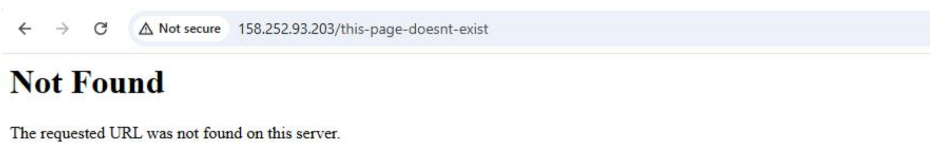
Access Log Analysis

```
[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 "https://3.29.125.10/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWe
bKit/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
```

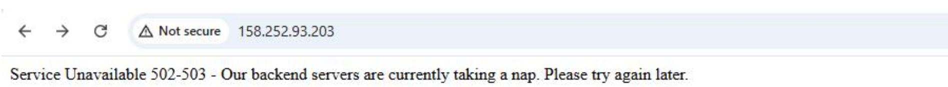
Bonus Tasks

Bonus 1: Custom Error Pages

Custom 404 Error Page



Custom 502 Error Page



Bonus 2: Implement Rate Limiting

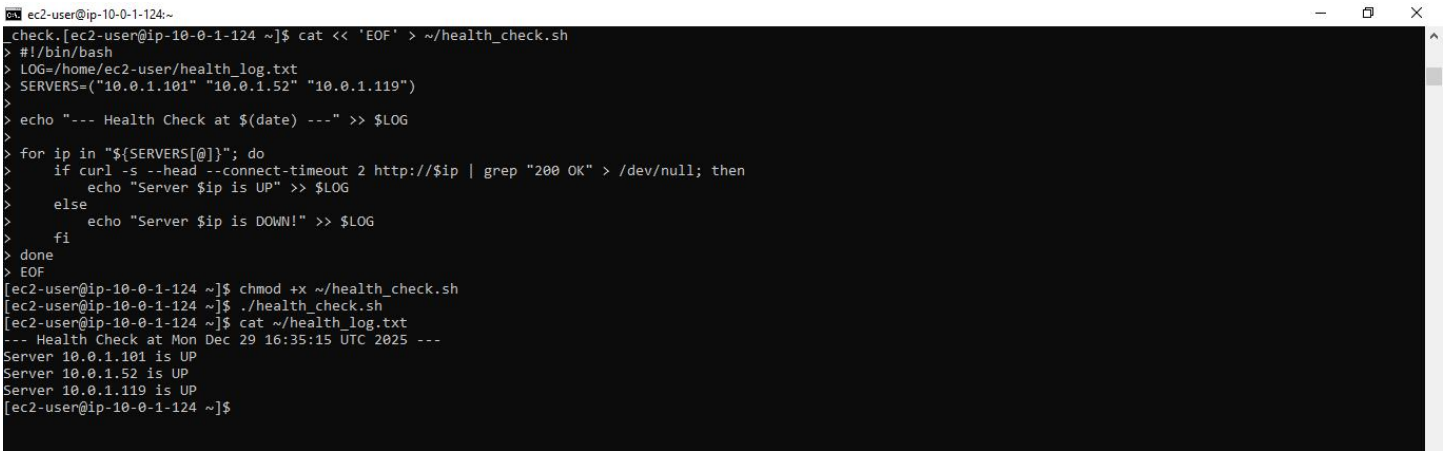
Rate Limiting Configuration

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

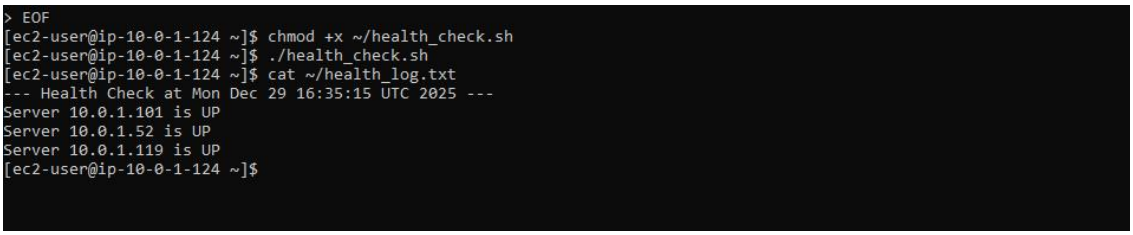
429 Error Page

Bonus 3: Health Check Automation

Health Check Script



Health Check Logs

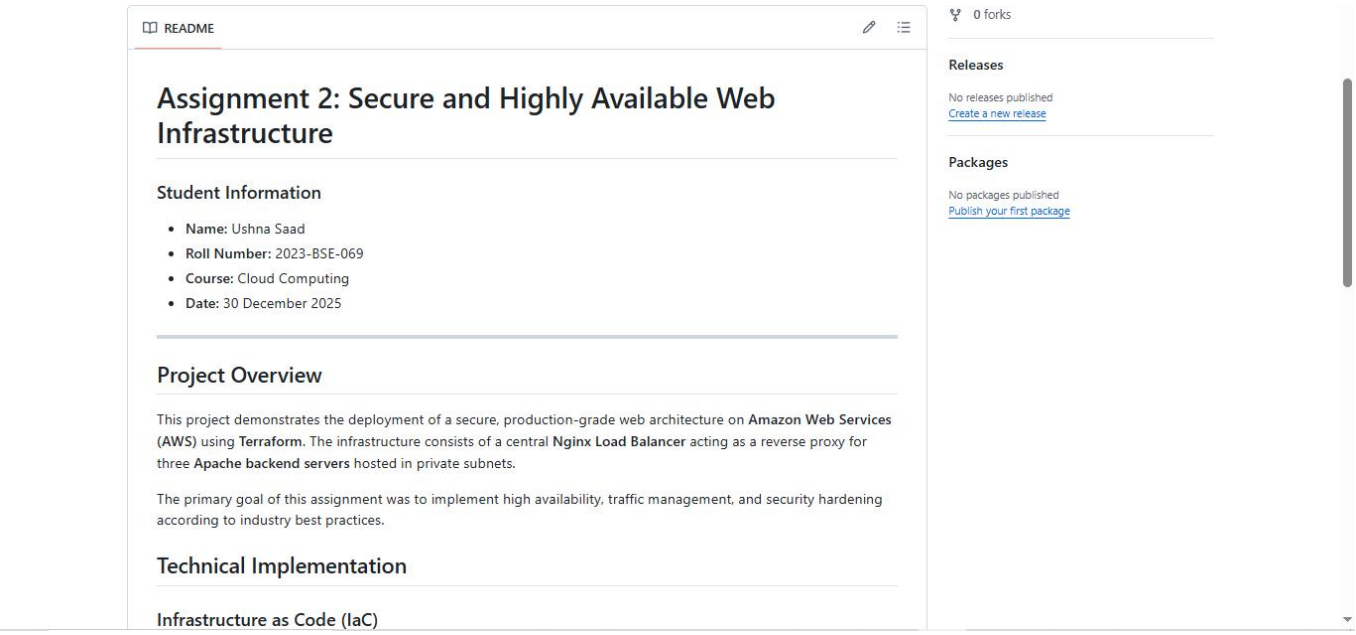


Part 6: Documentation & Cleanup

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



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.