**CLOUD COMPUTING**



**Assignment 2**

**Advanced Terraform & Nginx Multi-Tier Architecture**

**Submitted to:** Waqas Saleem

**Submitted by:** Aimen Hafeez

**Registration #:** 2023-BSE-002

**Deadline:** 30th Dec 2025

**Table of Contents**

[**Project Overview:** 2](#_Toc218002197)

[Arctitecture Diagram: 2](#_Toc218002198)

[PART 0: GitHub Repository & Codespace Setup 2](#_Toc218002199)

[PART 1 – Infrastructure Setup: 3](#_Toc218002200)

[Part 2: Webserver Module 8](#_Toc218002201)

[Part 3: Server Configuration Scripts 10](#_Toc218002202)

[Part 4: Infrastructure Deployment 12](#_Toc218002203)

[Part 5: Nginx Configuration & Testing 16](#_Toc218002204)

[Bonus Tasks 26](#_Toc218002205)

[Bonus 1: Custom Error Pages 26](#_Toc218002206)

[Bonus 2: Implement Rate Limiting 32](#_Toc218002207)

[Bonus 3: Health Check Automation 36](#_Toc218002208)

[Part 6: Documentation & Cleanup 41](#_Toc218002209)

[Github Repositry Link: 47](#_Toc218002210)

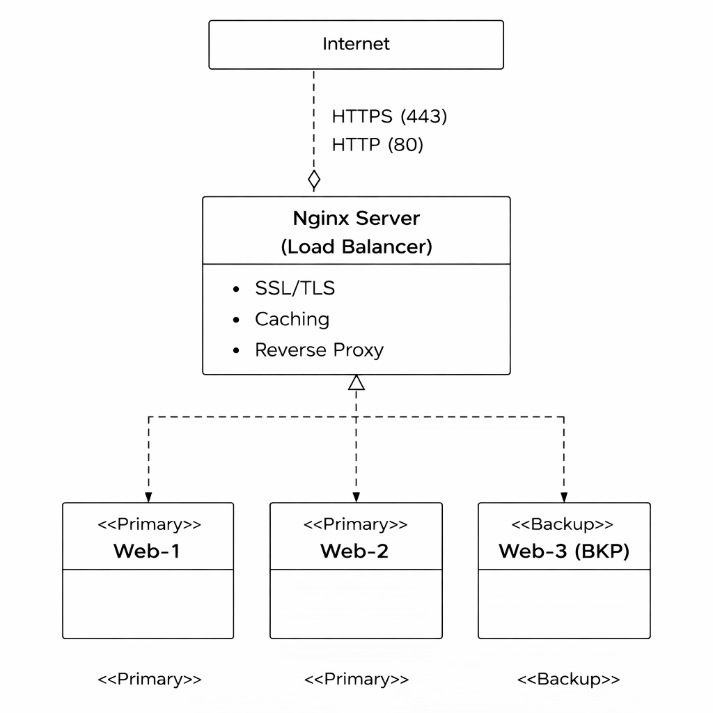
[Conclusion 47](#_Toc218002211)

# **Project Overview:**

This project focuses on designing and deploying a robust, production-ready multi-tier web infrastructure on AWS using Terraform for Infrastructure as Code (IaC) and Nginx as a reverse proxy and load balancer with advanced configurations. The assignment involves creating a high-availability environment comprising a single Nginx server for HTTPS traffic management and load balancing, three backend web servers for application hosting, and comprehensive networking and security configurations to ensure reliability and performance. Key objectives include modular Terraform code organization, dynamic variable and local configurations, automated server provisioning with reusable webserver modules, implementation of caching mechanisms, SSL/TLS encryption, and backup server setup for failover scenarios. The project emphasizes practical application of AWS services, secure SSH management, Nginx advanced settings, and monitoring, culminating in a scalable, secure, and maintainable web architecture validated through deployment, configuration testing, load balancing verification, caching assessment, and high-availability simulations.

## **Arctitecture Diagram:**

Here’s an archtitecture diagram for Multi-Tier Web Infrastructure:



## **PART 0: GitHub Repository & Codespace Setup**

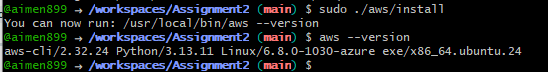
1. Authenticate GitHub CLI
2. Create GitHub Repository
3. Create GitHub Codespace
4. SSH into Codespace

## **PART 1 – Infrastructure Setup:**

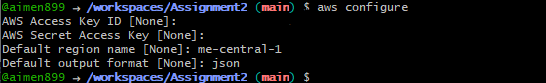
Install Terraform in Codespace



Install aws

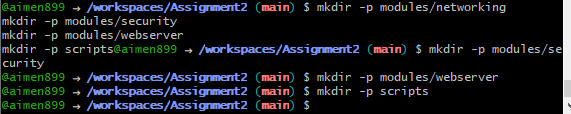


Configure AWS Credentials

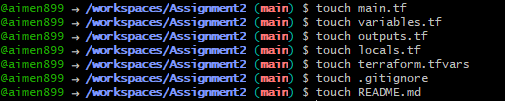


**1.1 Create Complete Folder Structure**

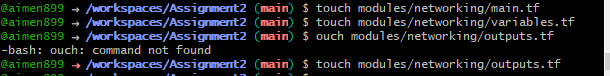
Create all directories



Create all root level files



Create networking module files



Create security module files



Create webserver module files



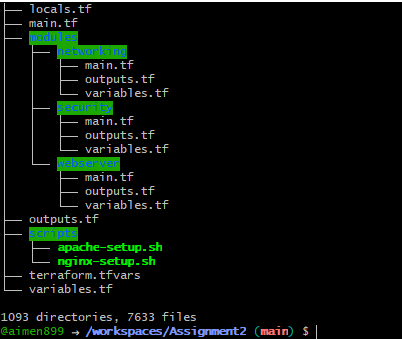
Create script files



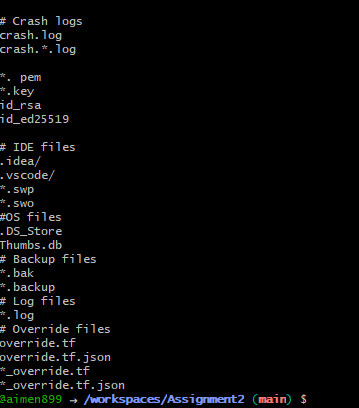
Make scripts executable



Verify structure with tree command

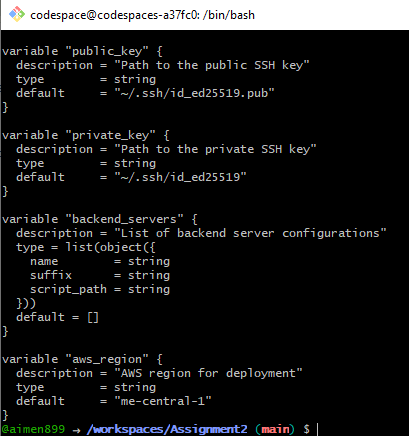


Create . gitignore File

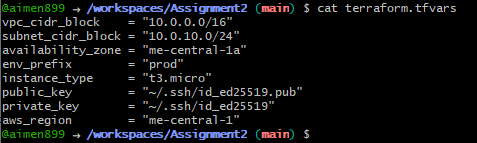


**1.2 Create File Contents**

Create variables.tf

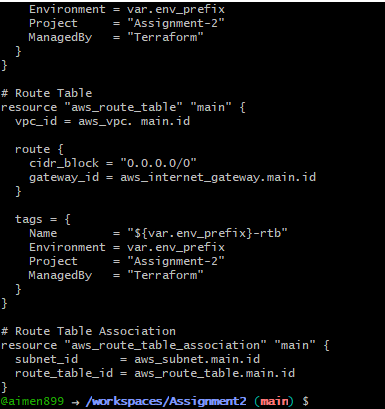


Create terraform.tfvars

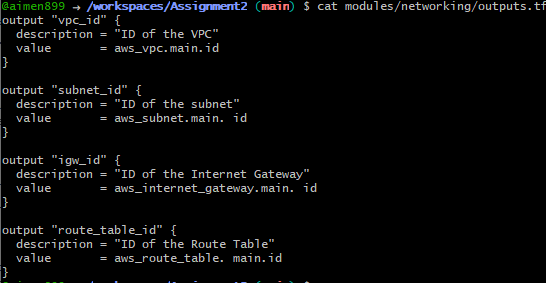


**1.3 Create Networking Module**

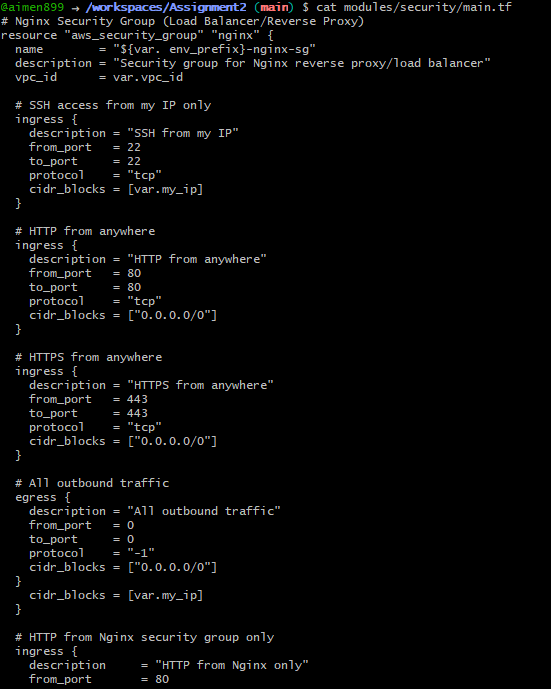
Main.tf



Outputs.tf



**1.4 Create Security Module**



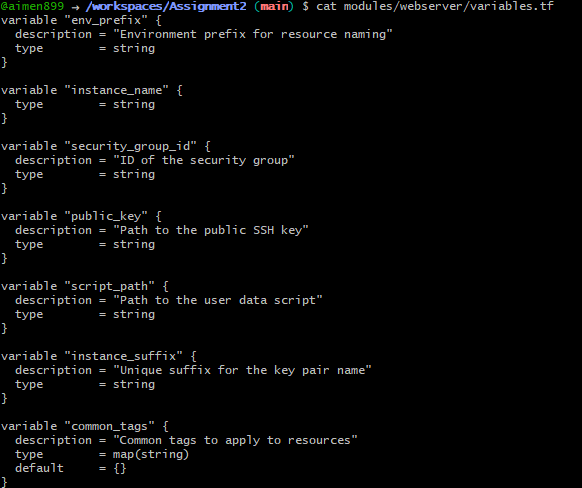
**1.5 Create locals.tf**



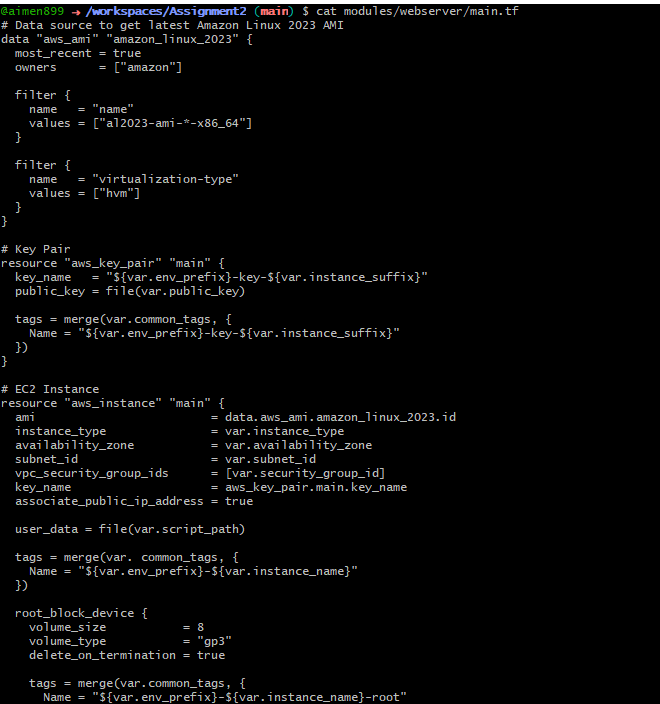
## **Part 2: Webserver Module**

**2.1 Create Webserver Module**

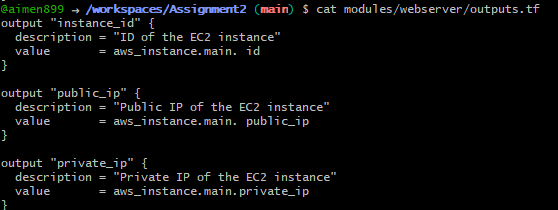
modules/webserver/variables.tf



modules/webserver/ main.tf



modules/webserver/ outputs.tf

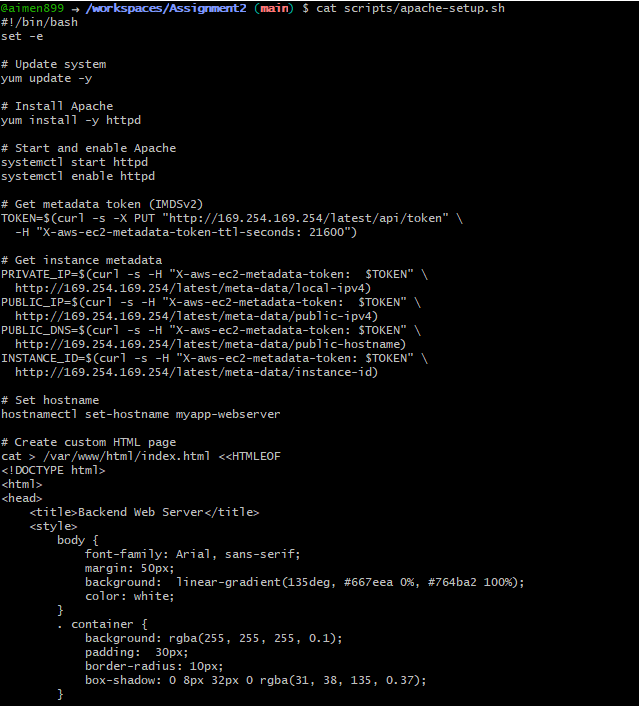


**2.2 Create Root main.tf**

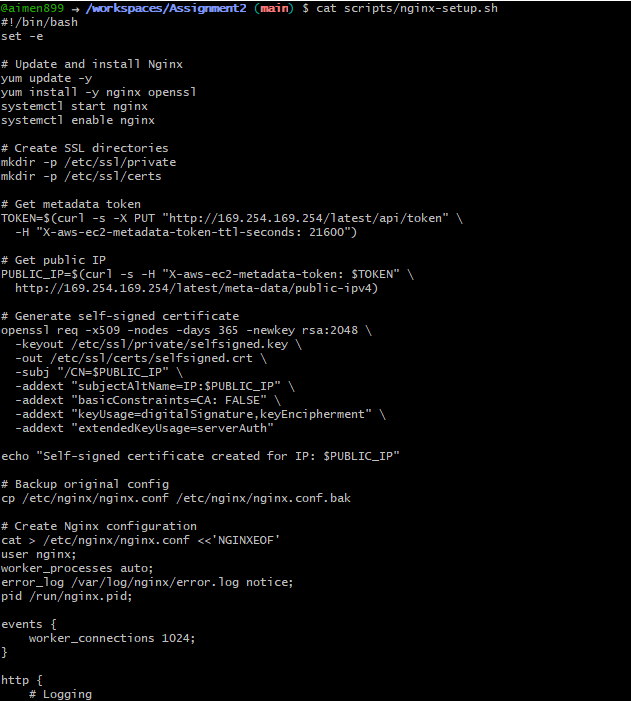


## **Part 3: Server Configuration Scripts**

**3.1 Create Apache Backend Script**



**3.2 Create Nginx Setup Script**



## **Part 4: Infrastructure Deployment**

**4.1 Generate SSH Key Pair**



**4.2 Create outputs.tf**



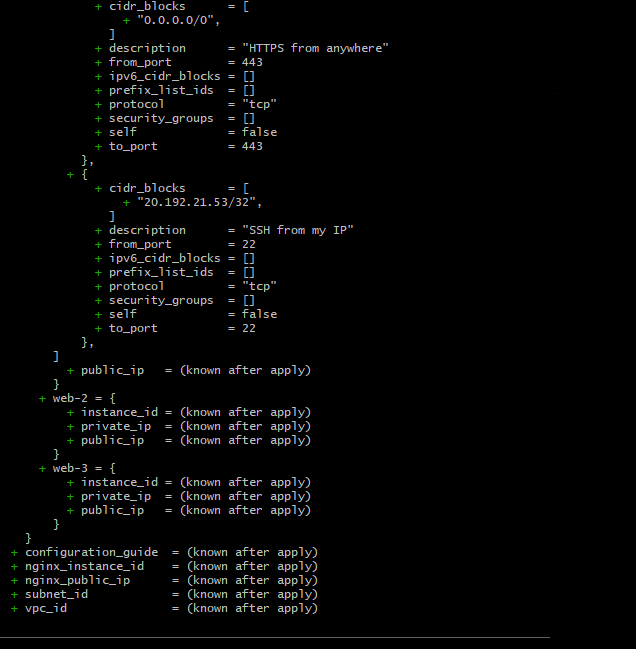
Initialize Terraform



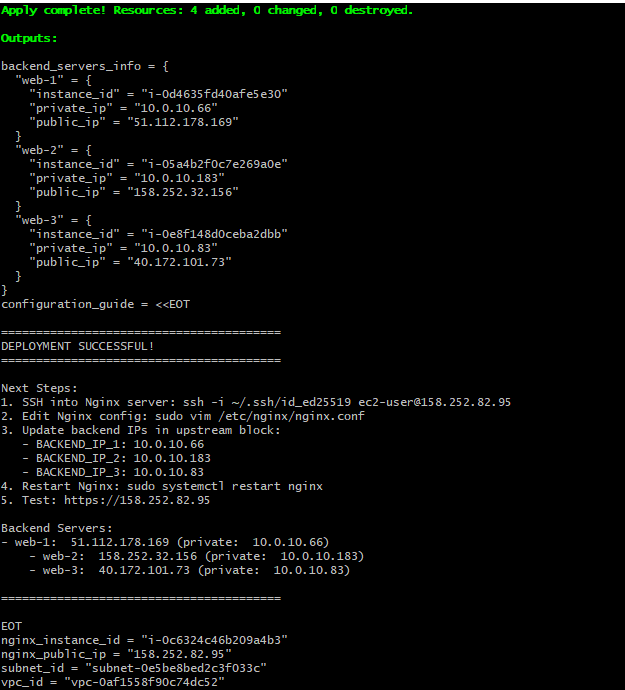
Validate Terraform Configuration



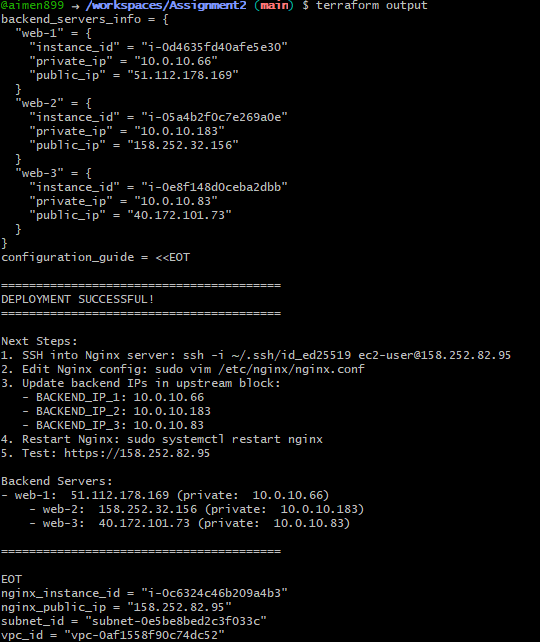
Plan Terraform Deployment



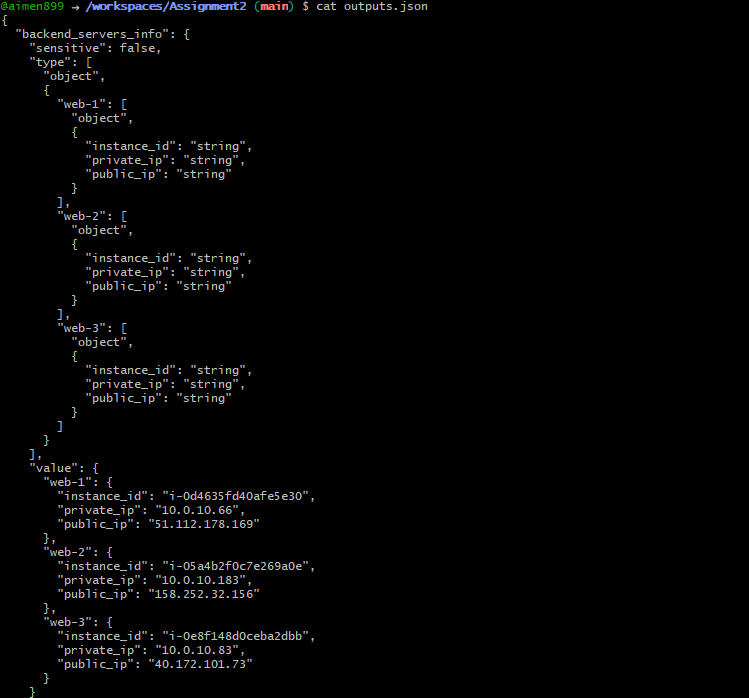
Apply Terraform Configuration



Terraform output:

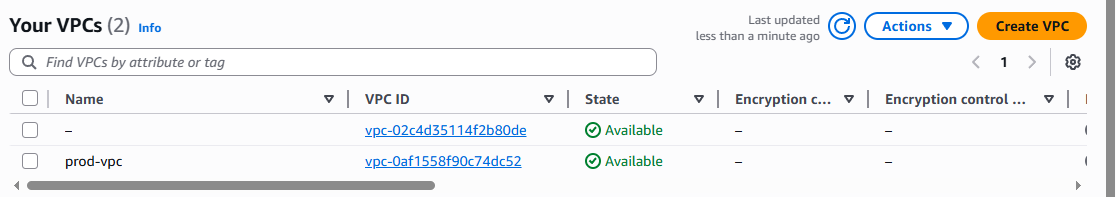


Output.json

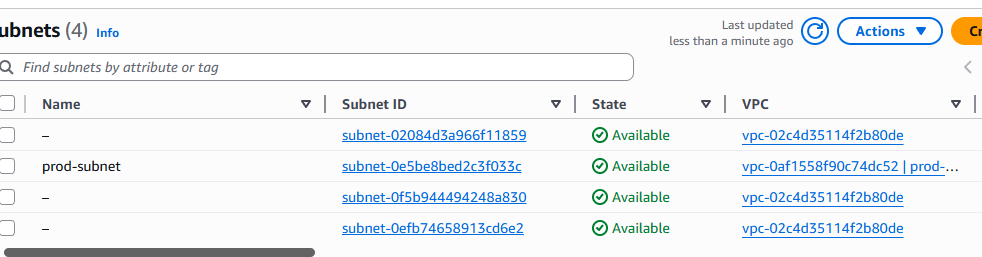


**4.3 Verify Resources in AWS Console**

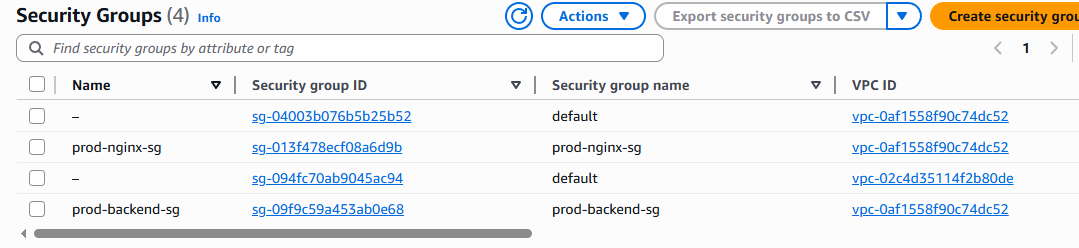
VPC:



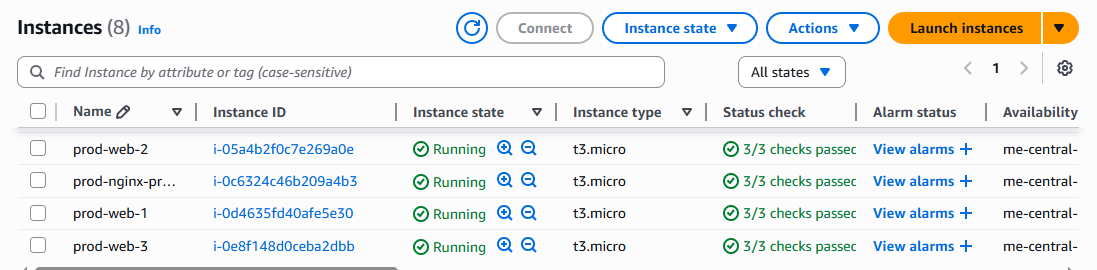
Subnet:



Security Groups:

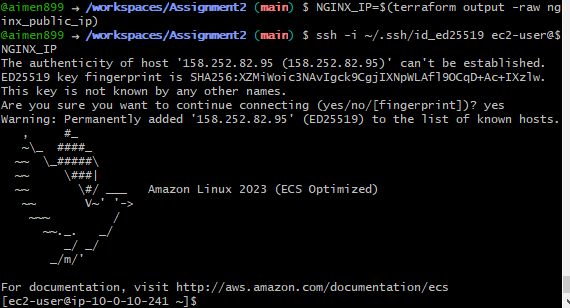


Instances:



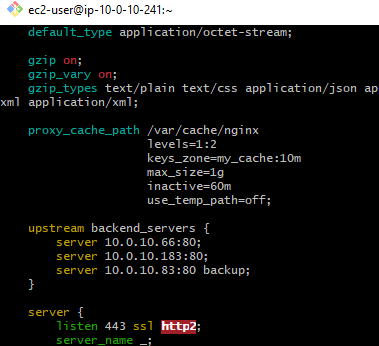
## **Part 5: Nginx Configuration & Testing**

**5.1 SSH into Nginx Server**

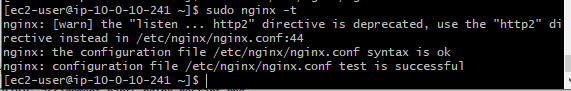


Update Nginx Configuration

On the Nginx server:



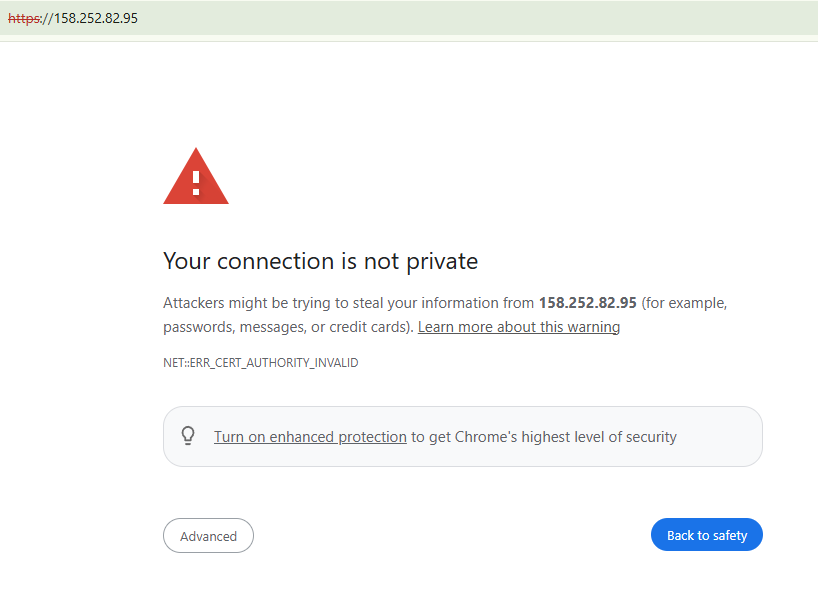
Test Nginx configuration



Nginx Restart:



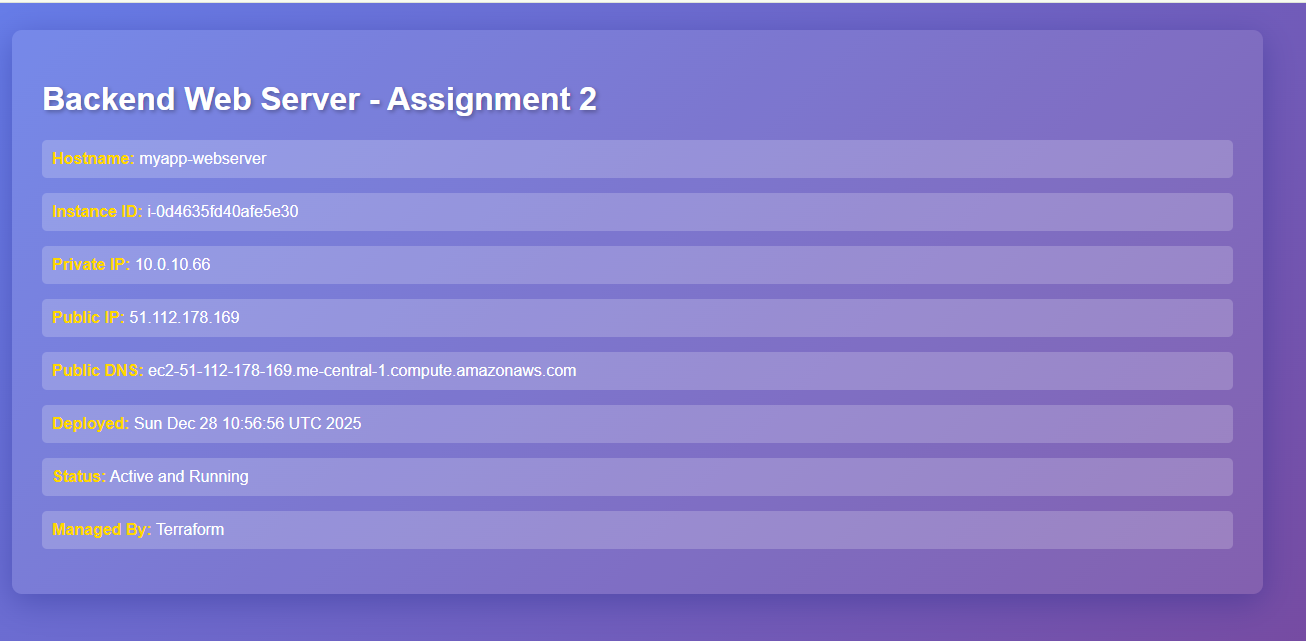
**5.2 Test Load Balancing**



Reload the page multiple times

Observe traffic alternating between web-1 and web-2

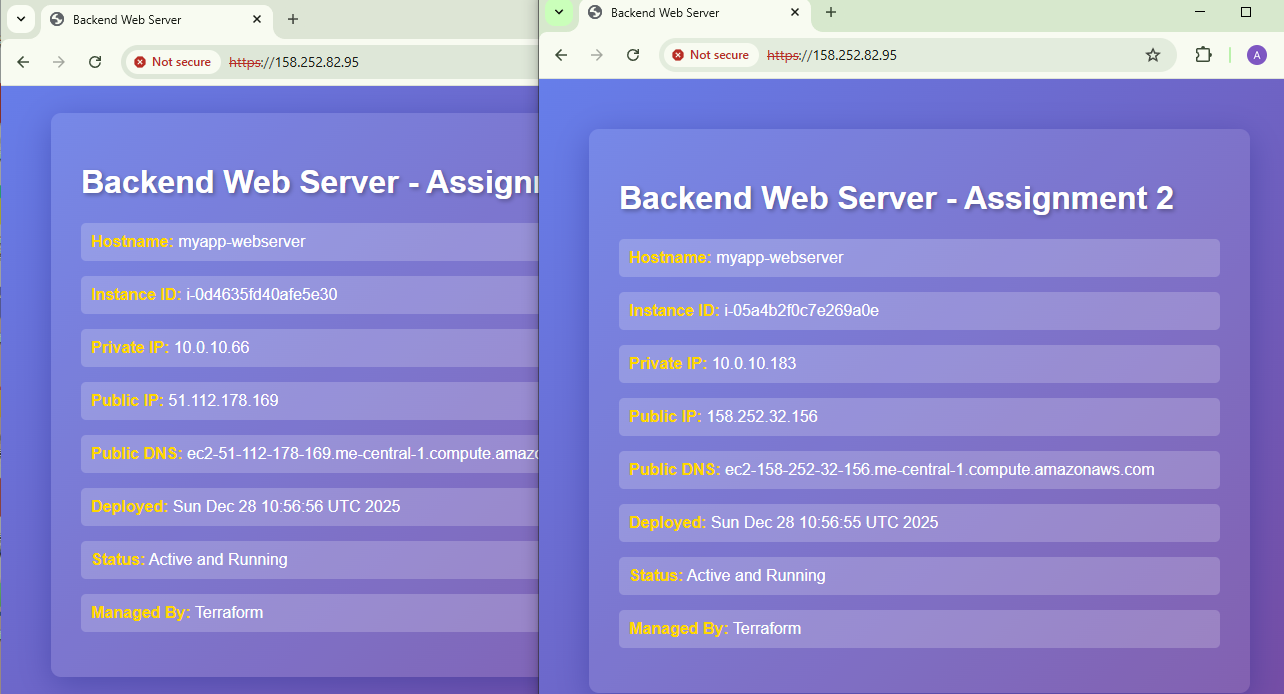
Web-1:



Web-2:

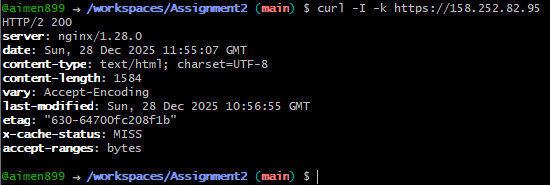


Load Balancing Demo:

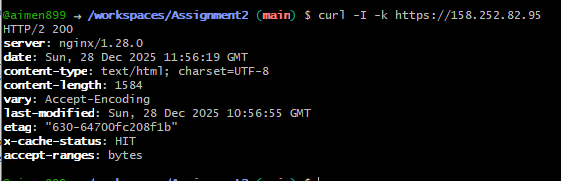


**5.3 Test Cache Functionality**

MISS on first request



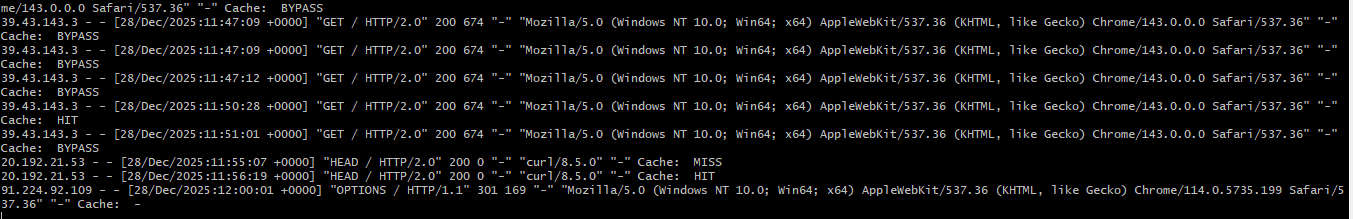
HIT on subsequent request



Cache Directory:

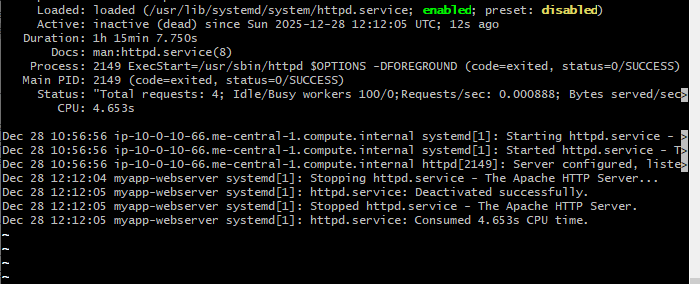
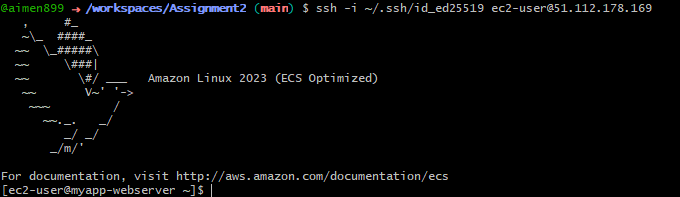


Access Log Cache:



**5.4 Test High Availability**

SSH into web-1 and stop Apache

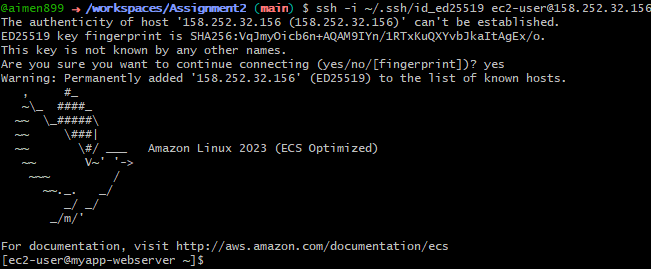


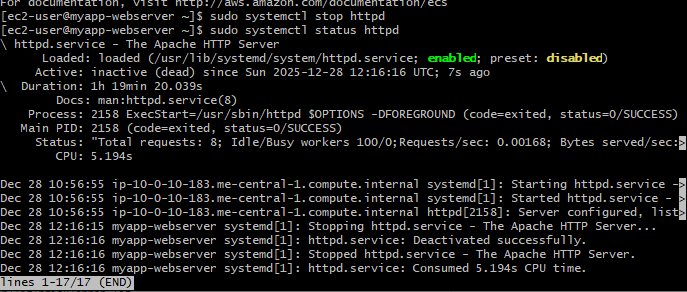
Test - should only show web-2 now

On subsequent reloads, only web-2 server is shown:



SSH into web-2 and stop Apache



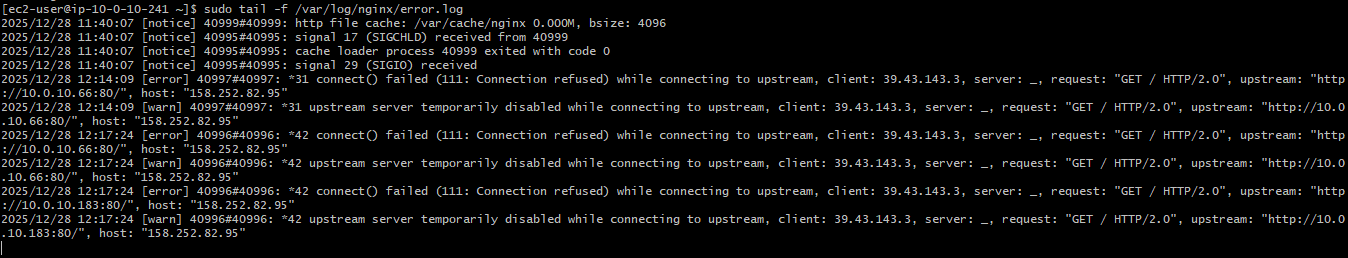


Test - should now show web-3 (backup activated)

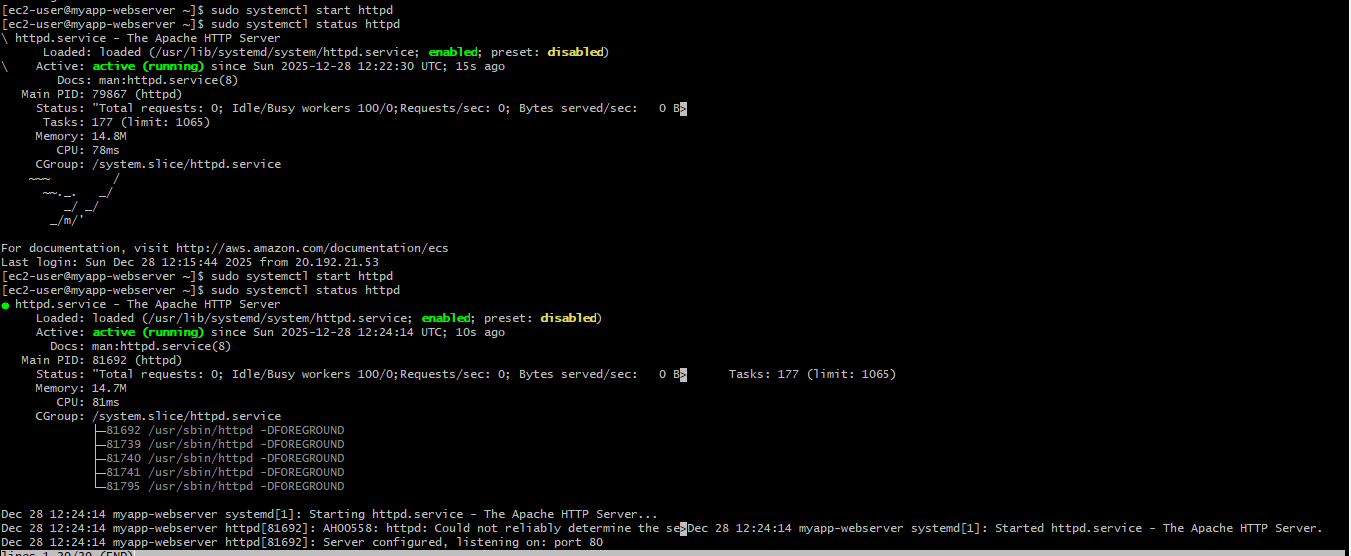
Reload browser



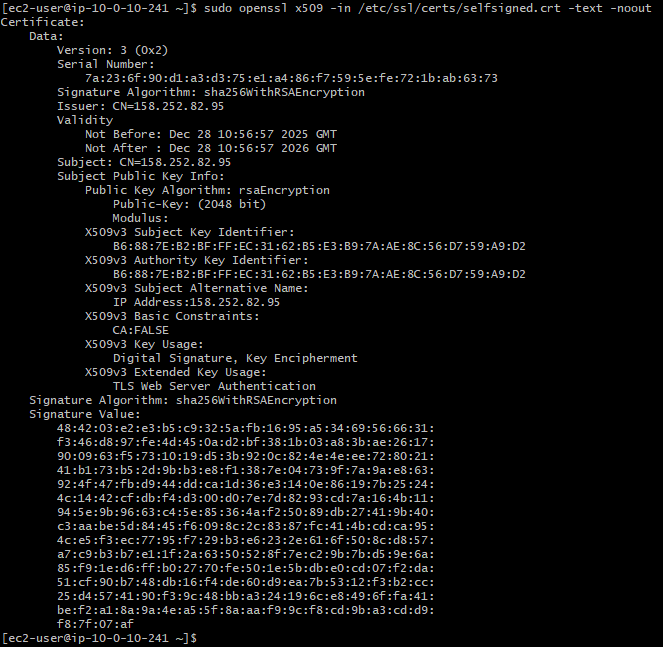
Check Nginx error logs



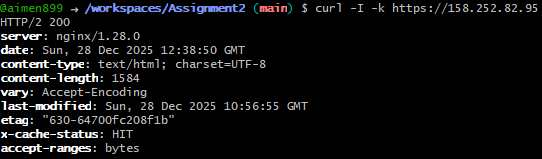
Restart services on web-1 & web-2:



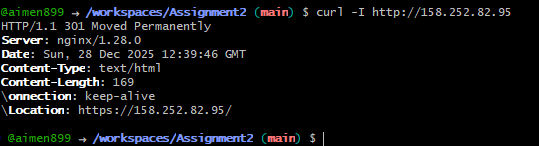
**5.5 Security & Performance Analysis**



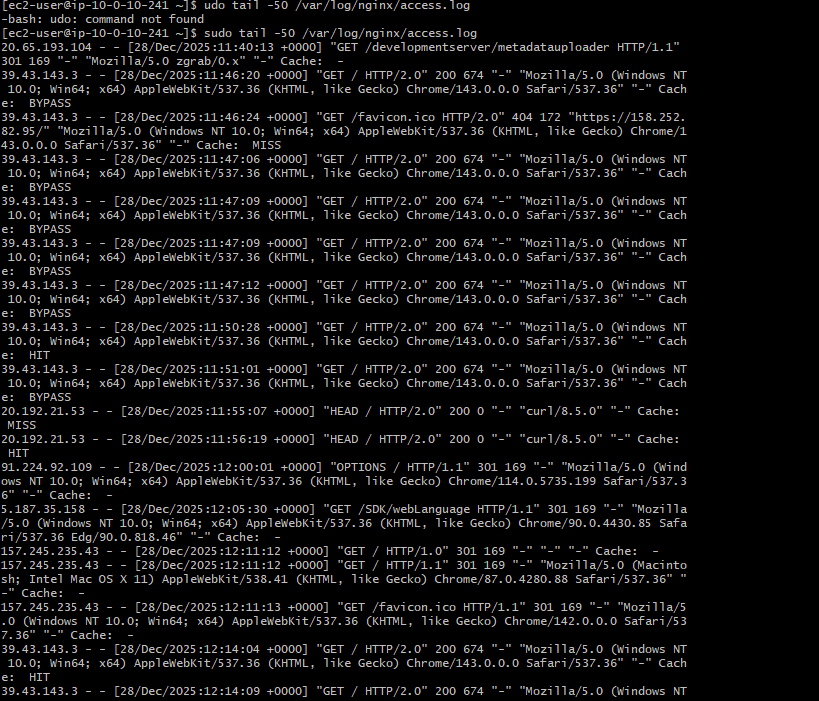
Check security headers



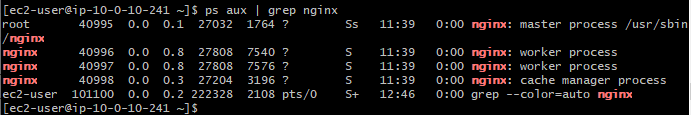
Test HTTP redirect



View logs



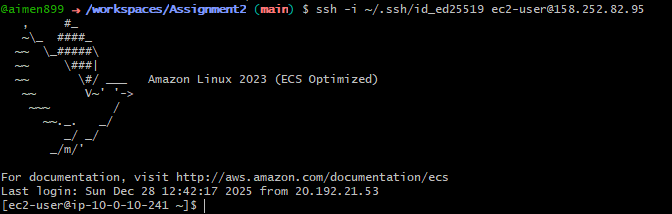
Check Nginx worker processes



# **Bonus Tasks**

## **Bonus 1: Custom Error Pages**

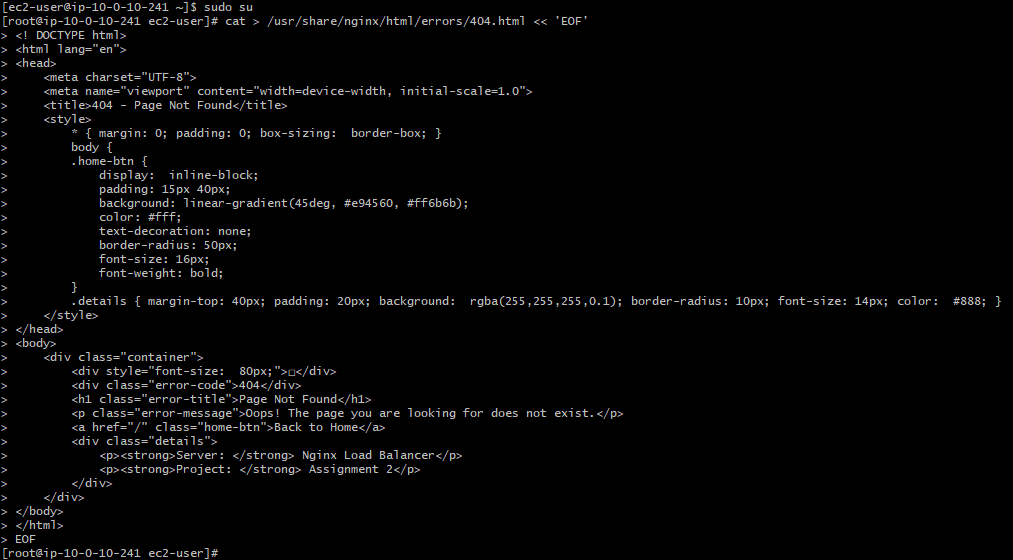
1. SSH into Nginx Server



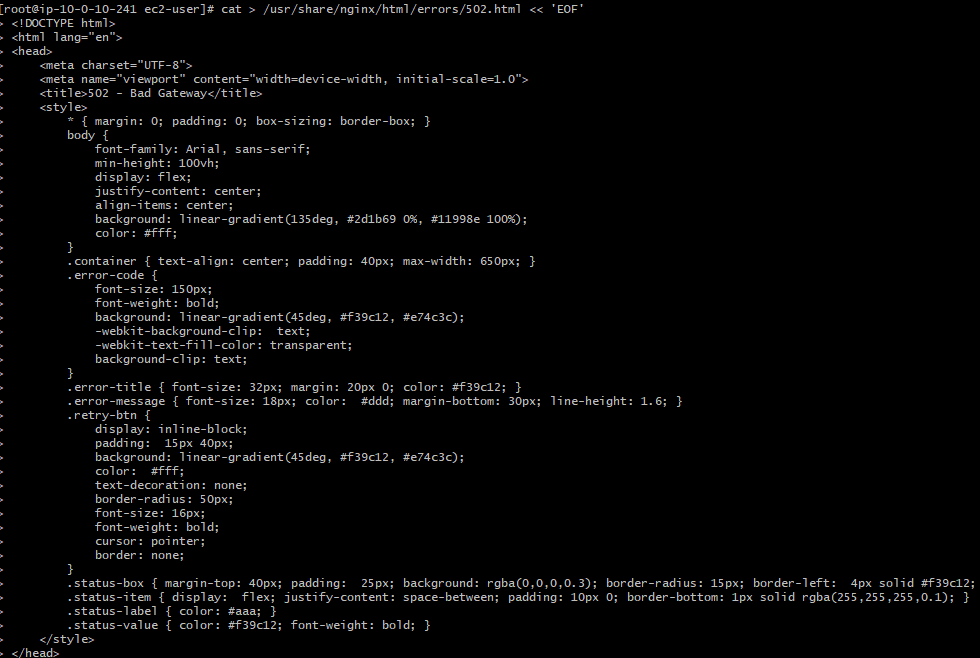
1. Create Error Pages Directory



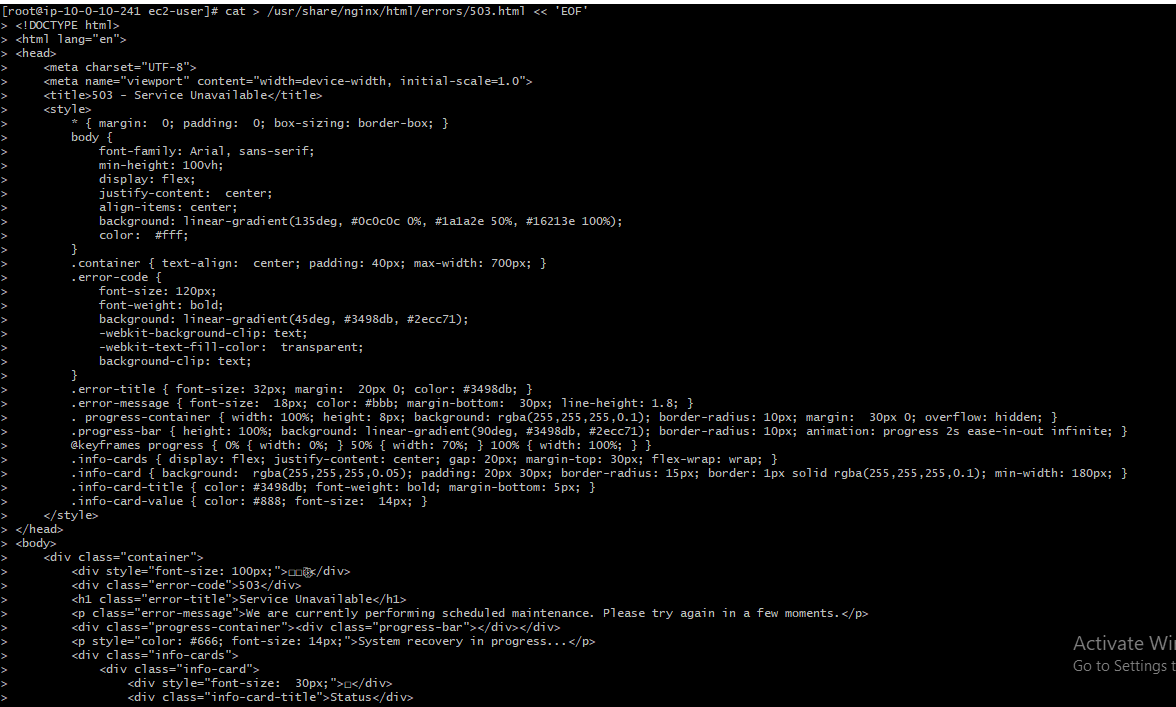
1. Create Custom 404 Error Page



1. Create Custom 502 Error Page - Bad Gateway Error Page



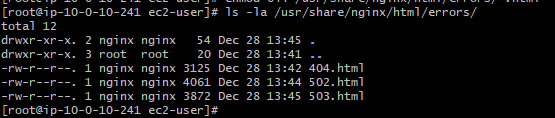
1. Create 503 error Page - Service Unavailable Error Page



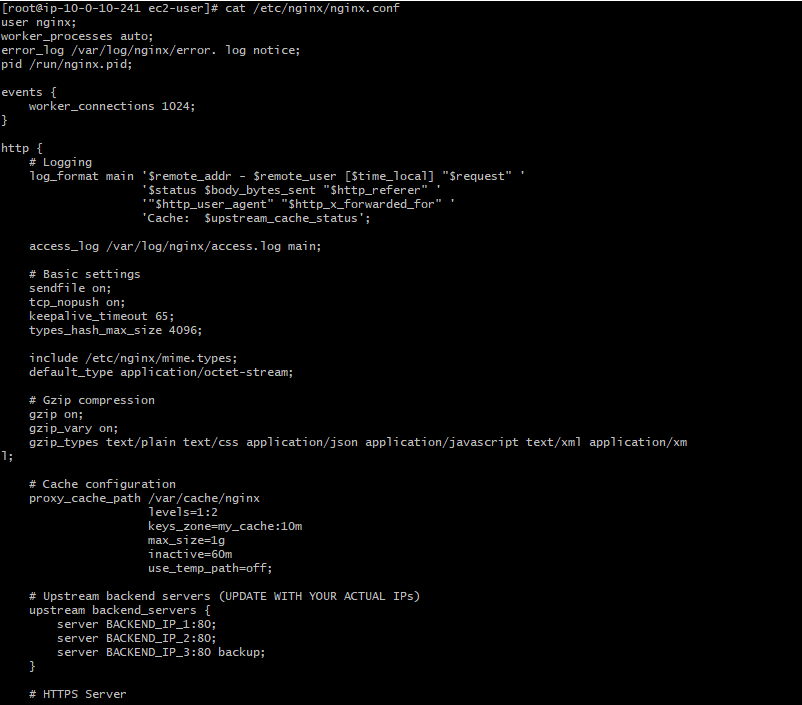
1. Set Proper Permissions



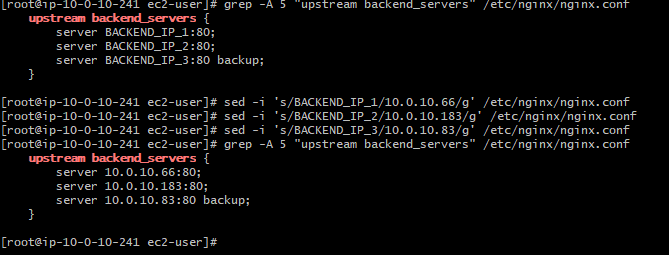
1. Verify files permissions



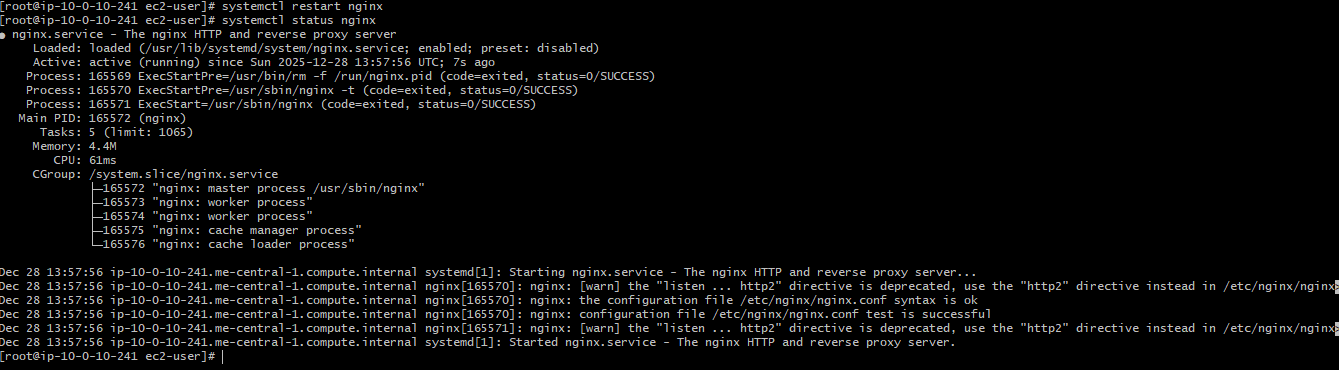
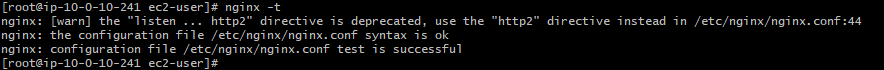
1. Update Nginx Configuration for Custom Error Pages



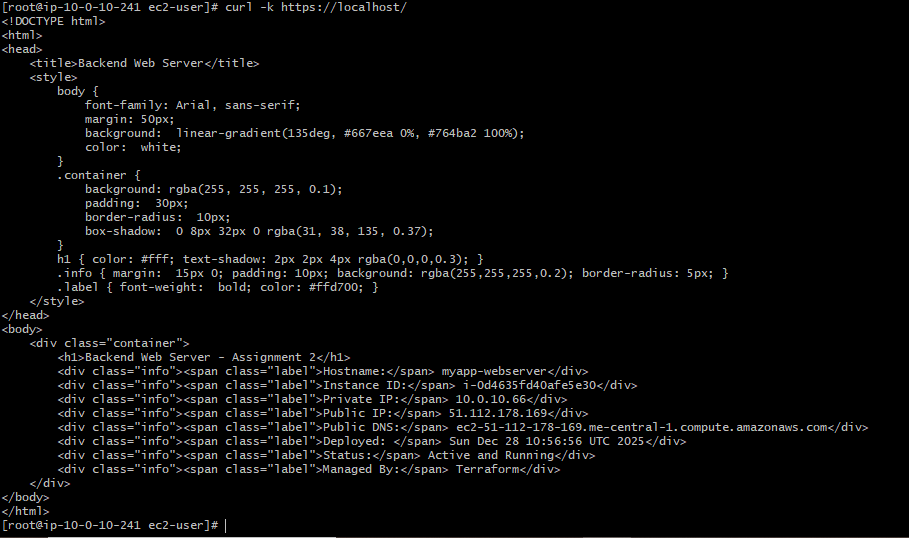
1. Update Backend IPs in Configuration



1. Test and Restart Nginx



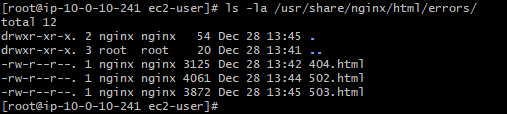
1. Verify Everything is Working



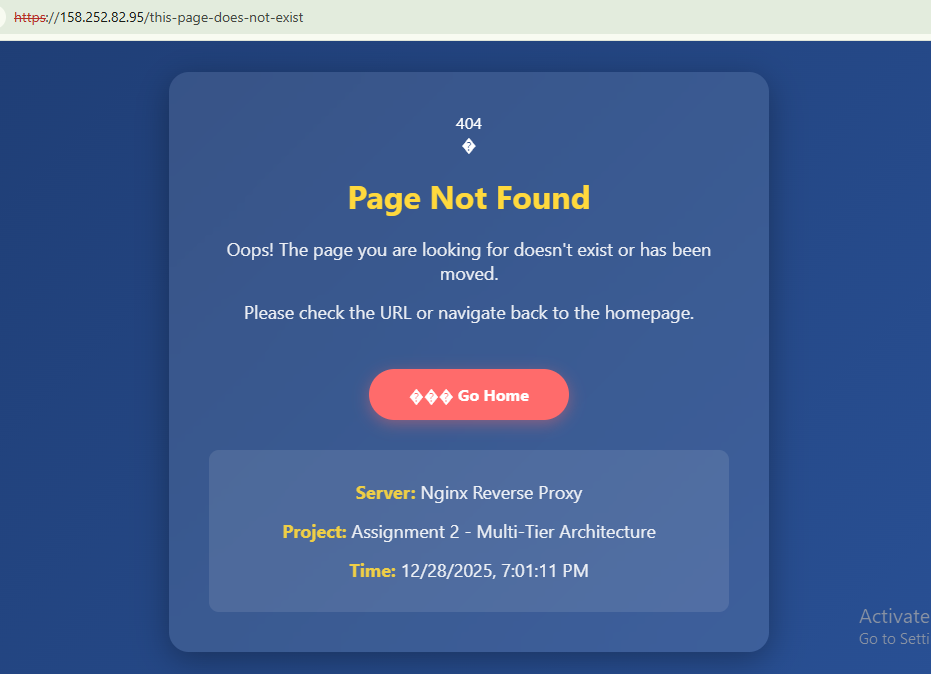
Test 404 error page



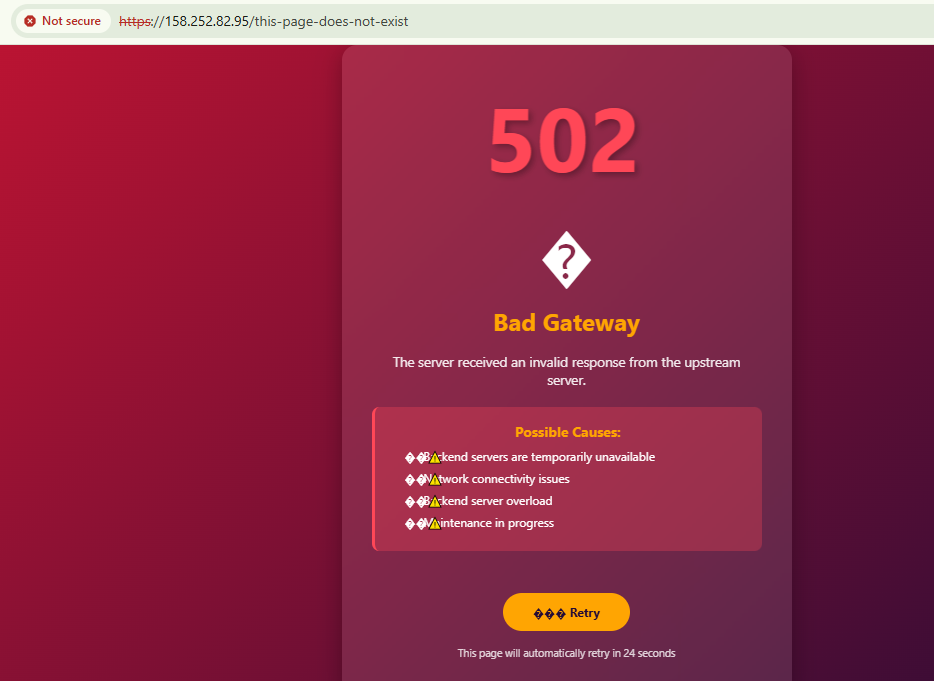
Check error pages directory



1. Test 404 Page



1. Test 502 Page



1. Restart Backend Servers



## **Bonus 2: Implement Rate Limiting**

1. SSH into Nginx Server and Become Root

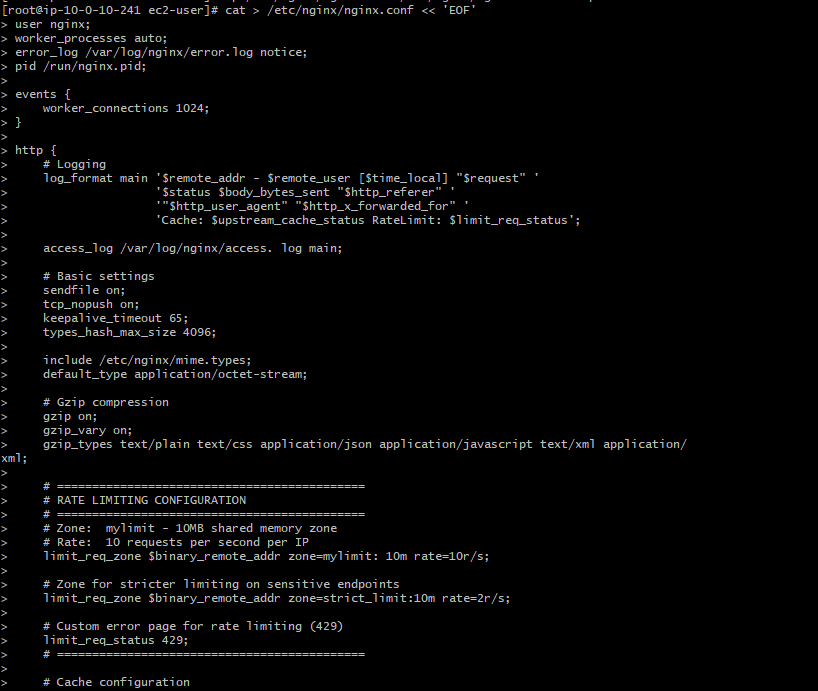


1. Update Nginx Configuration with Rate Limiting

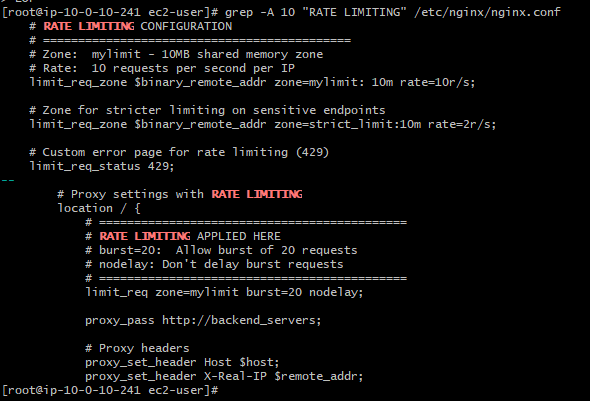
Backup current config



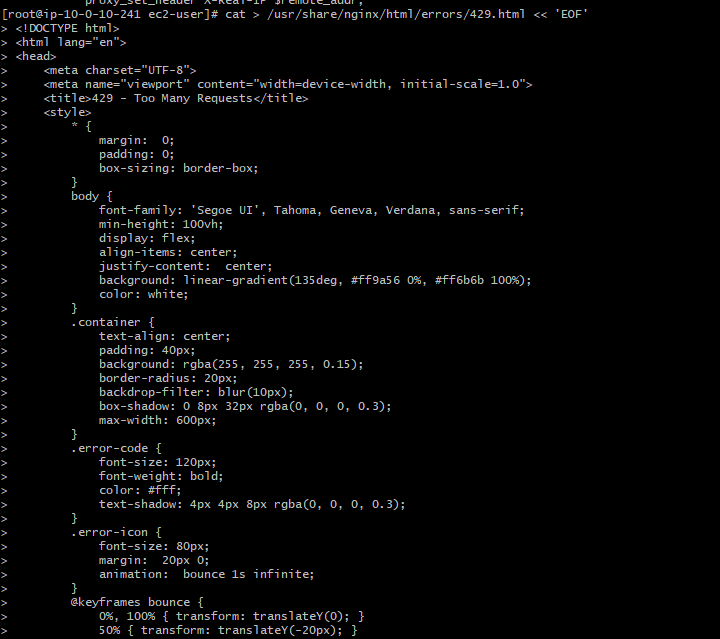
1. Create updated Nginx configuration with rate limiting



Display the rate limiting configuration



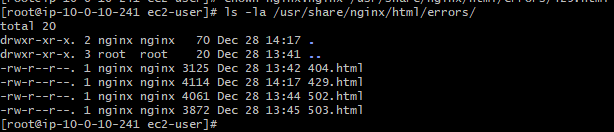
1. Create Custom 429 Error Page



Set permissions

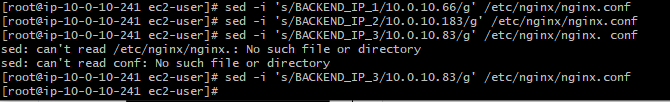


Verify file

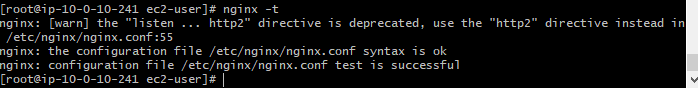


1. Update Backend IPs and Test Configuration

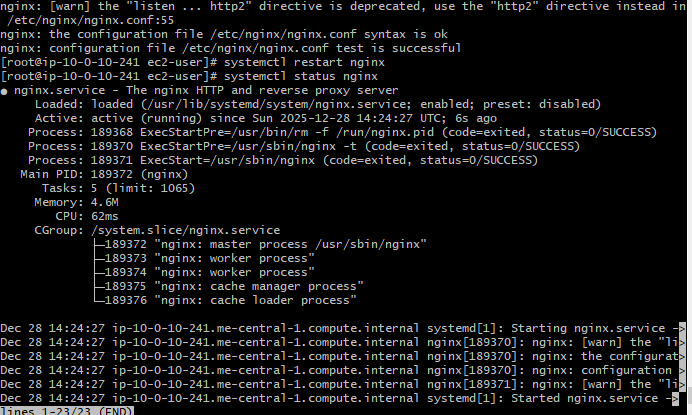
Replace with your actual backend private IPs



Test Nginx configuration

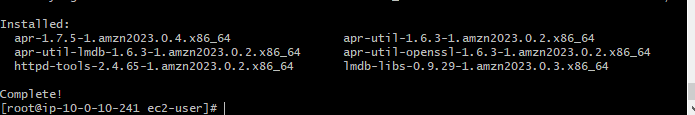


Restart Nginx

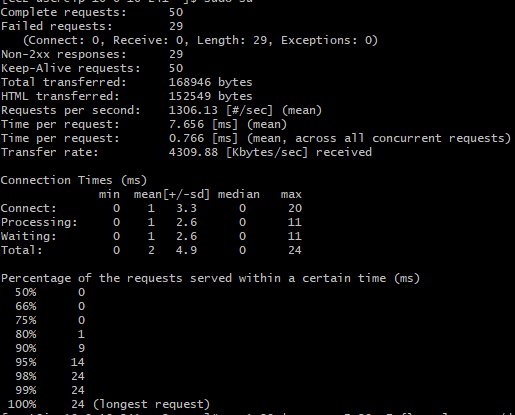


1. Test Rate Limiting with Rapid Requests

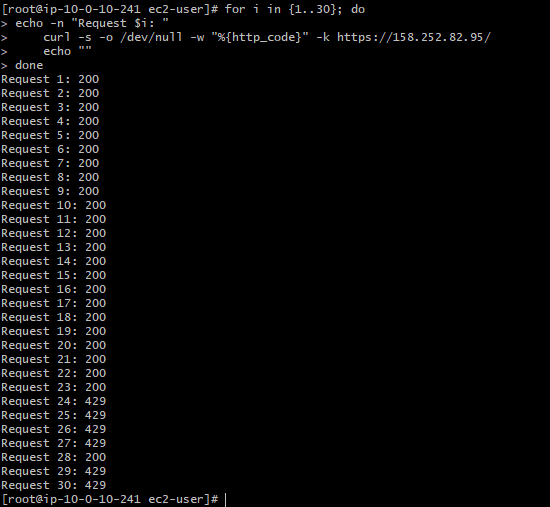
Install Apache Benchmark tool



Test rate limiting - Send 50 requests rapidly  
This should trigger rate limiting after the burst limit is exceeded



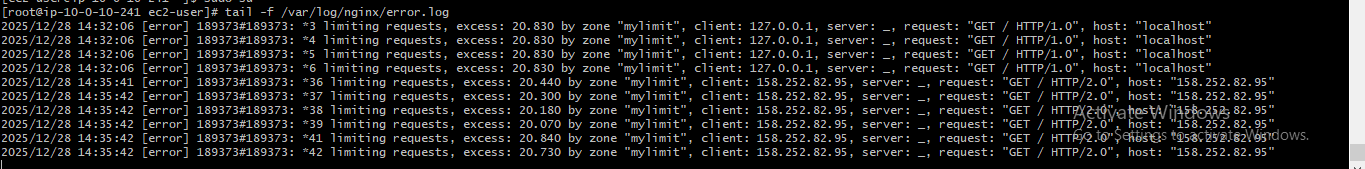
Alternative Testing with curl in rapid succession



1. Monitor Rate Limiting in Logs



Check error logs for rate limit messages

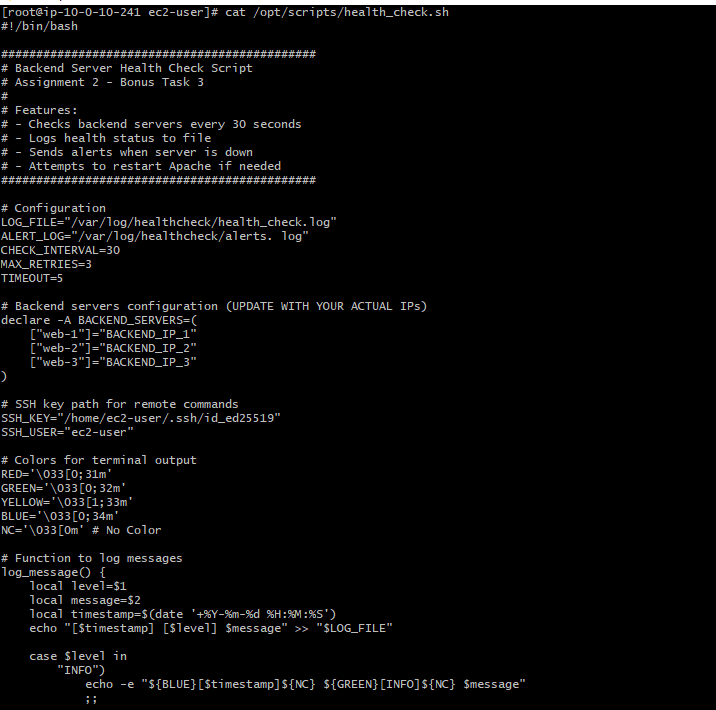


## **Bonus 3: Health Check Automation**

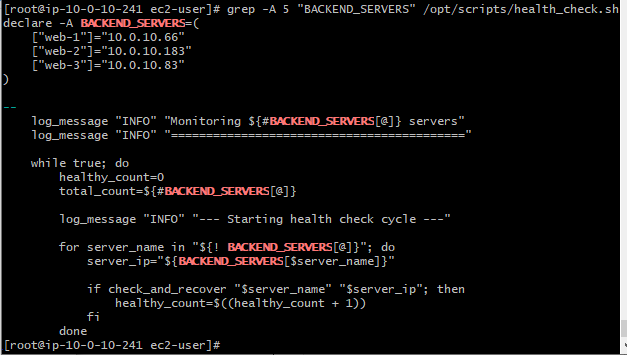
1. Create Scripts Directory



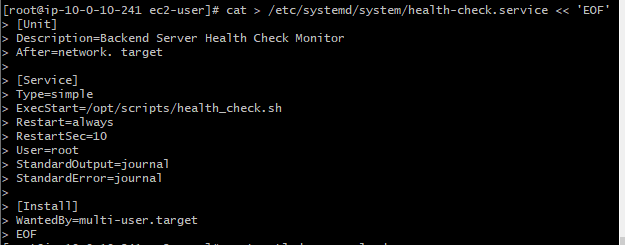
1. Create Health Check Script



1. Update Script with Actual Backend IPs



1. Create Systemd Service for Health Check



Reload systemd



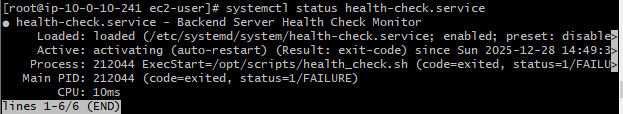
Enable service to start on boot



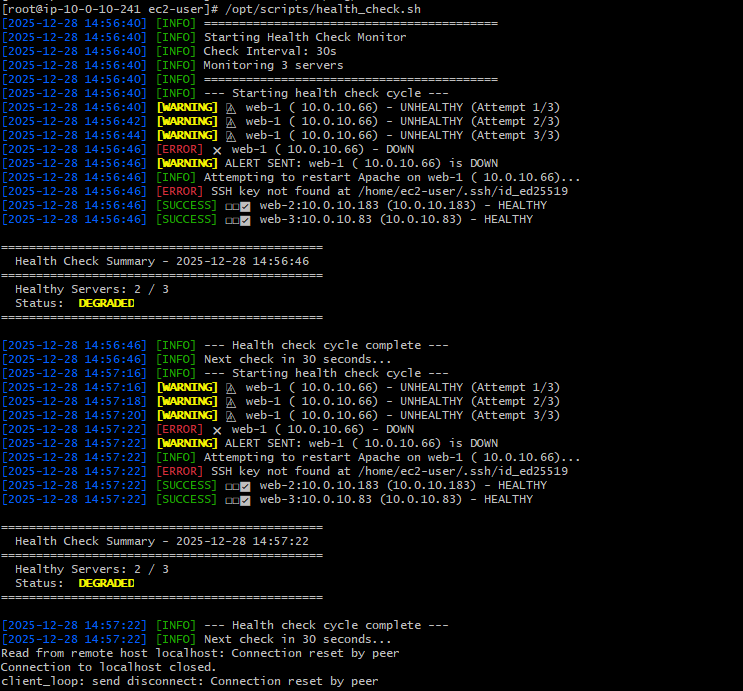
Start the service



Check status



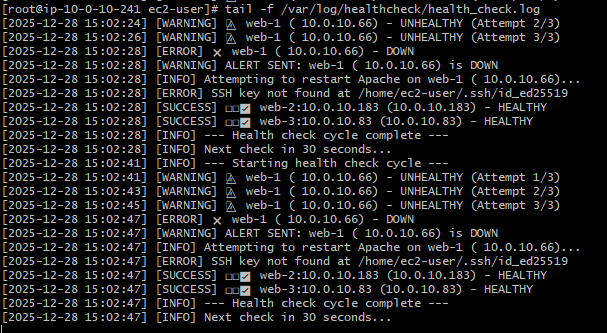
1. Run Health Check Manually for Testing



1. Start the Systemd Service

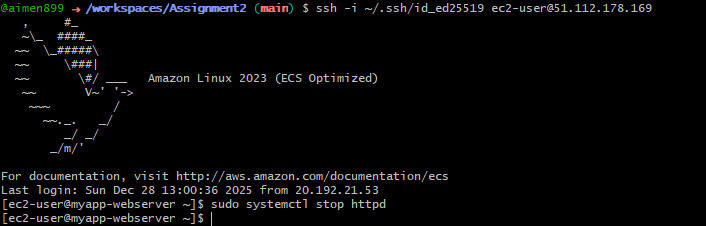


1. View the Logs



1. Test Alert by Stopping a Backend Server

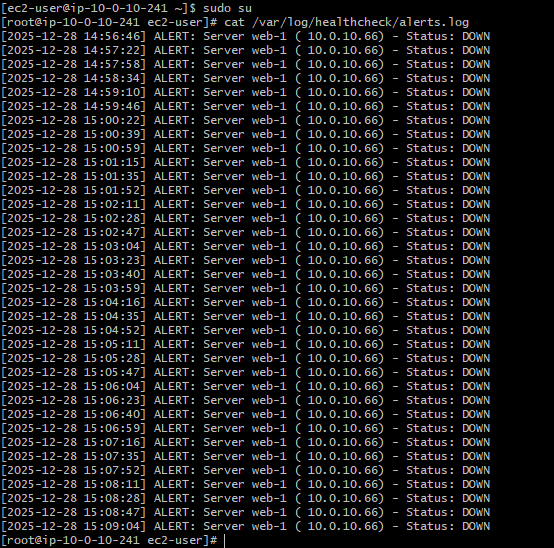
Open another terminal and stop Apache on one of the backend servers:



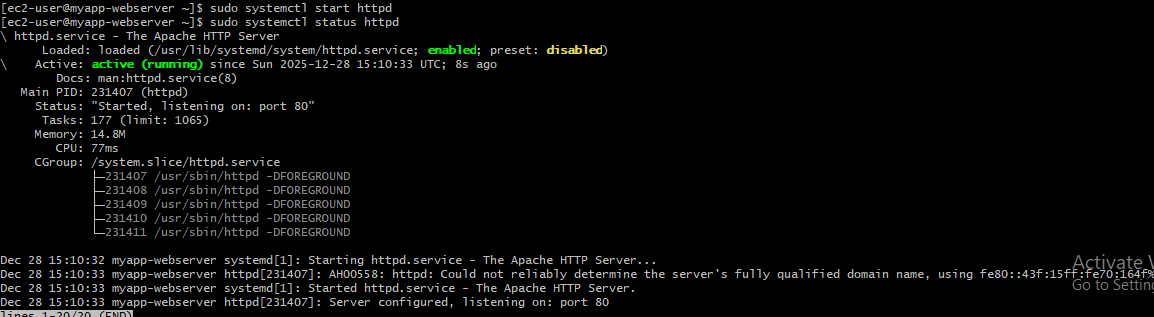
Then watch the health check logs on the Nginx server:

(ALERT: web-1 is down)

1. View Alert Log

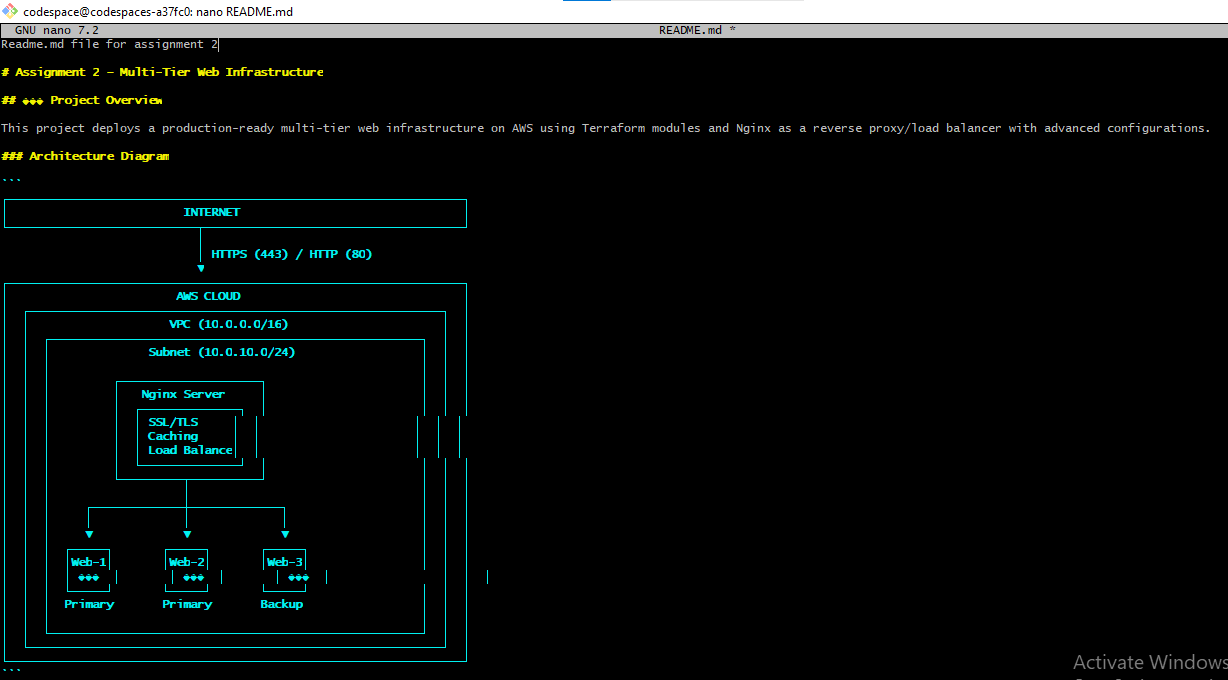


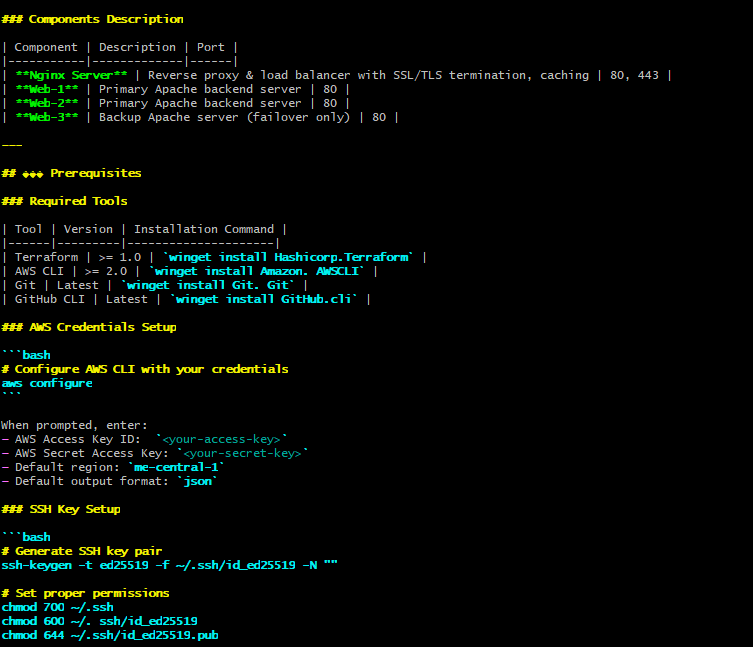
1. Restart the Backend Server

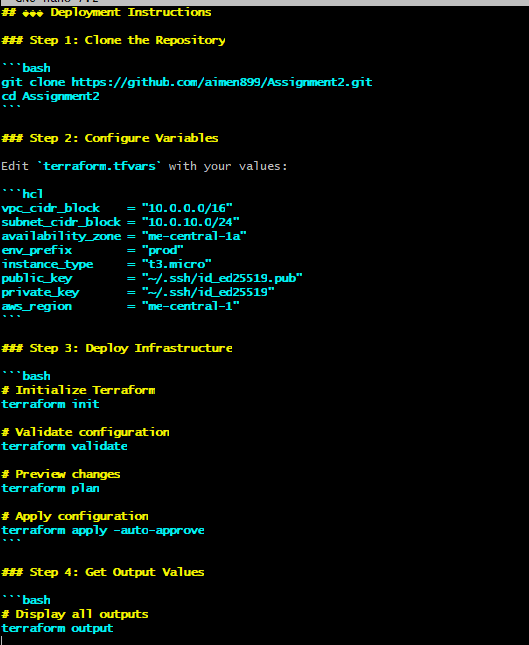


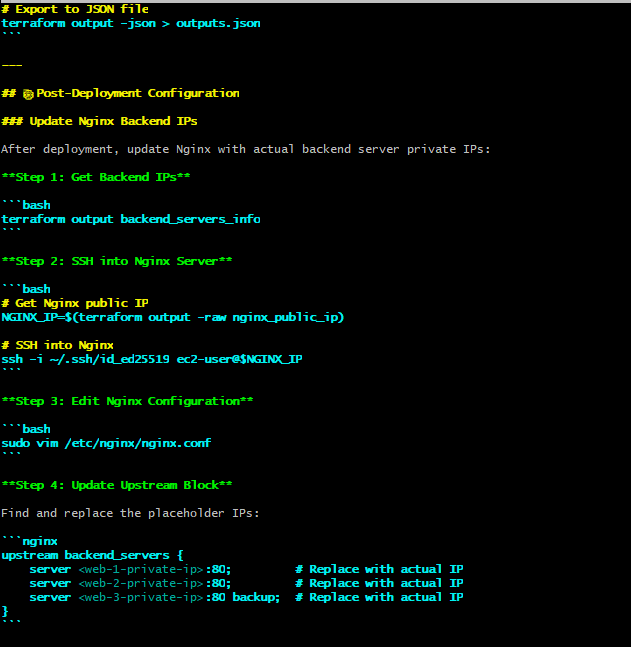
## **Part 6: Documentation & Cleanup**

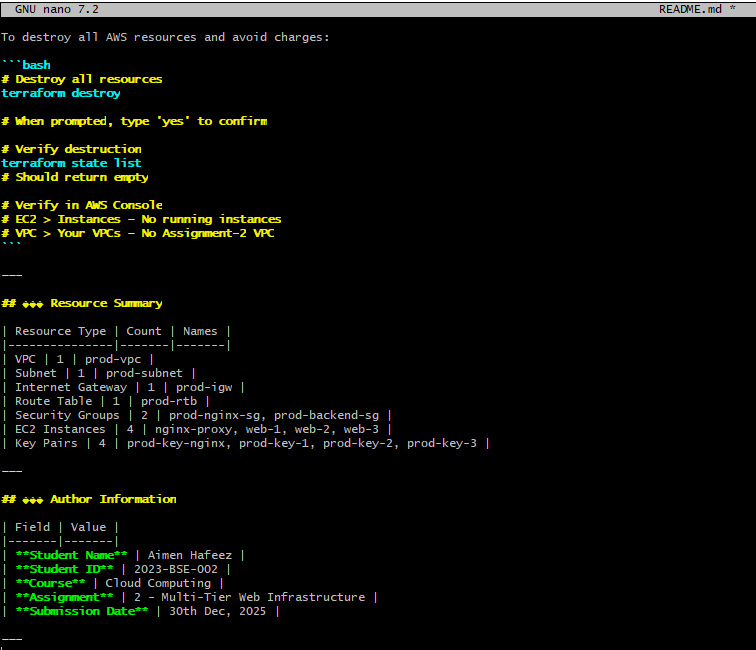
**6.1 Create README. md**



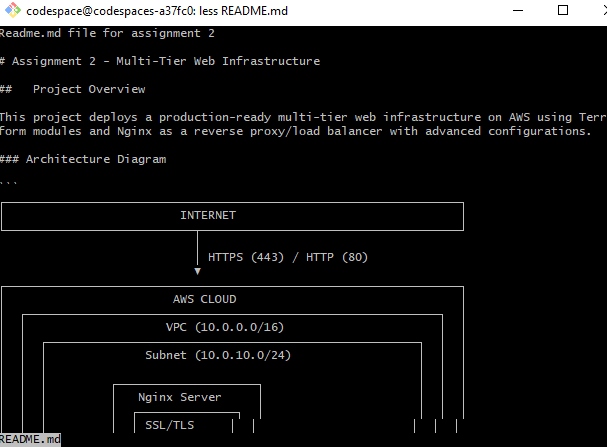




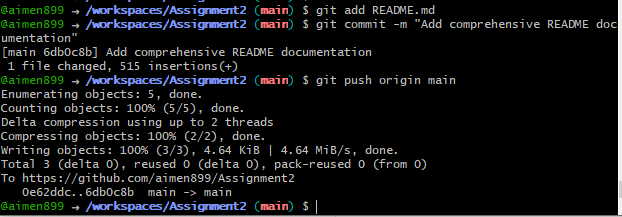




Alternatively, view README in a pager



Verify README renders correctly on GitHub



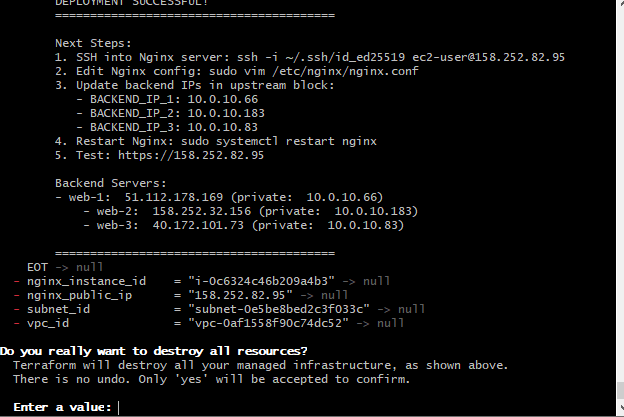
**6.2 Infrastructure Cleanup**

Prepare for Destruction

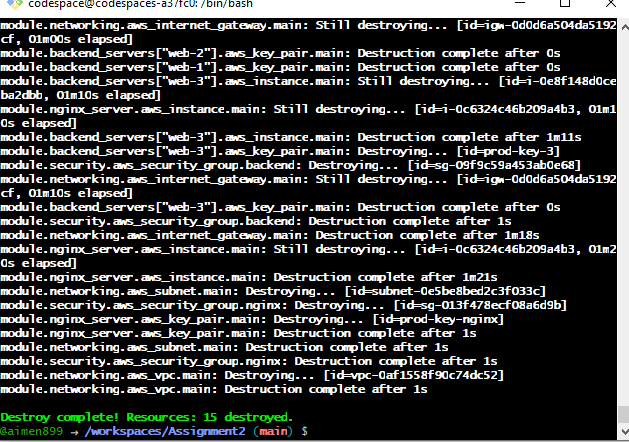
Before destroying, verify what resources exist:



Run Terraform Destroy



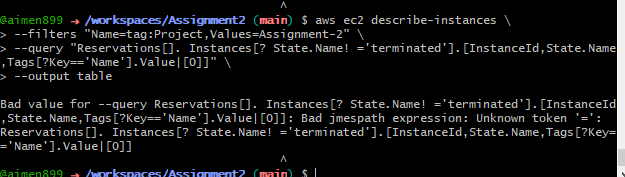
Confirm Destruction



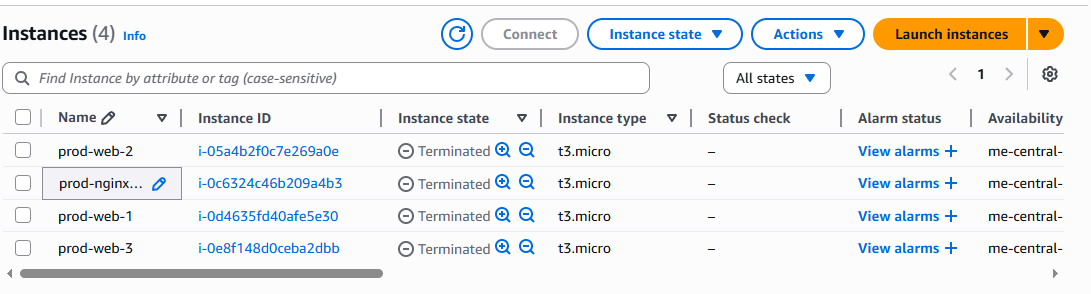
Verify State File is Empty



Verify No Resources Remain Using AWS CLI



Verify in AWS Console



## **Github Repositry Link:**

<https://github.com/aimen899/Assignment2>

# **Conclusion:**

To sum up, this assignment showed the design and deployment of a production-ready multi-tier web architecture with Terraform and Nginx on AWS successfully. The application of Infrastructure as Code principles granted the project a scalable, secure, and highly available environment with modular Terraform configurations, automated provisioning, and efficient resource management. The integration of Nginx as a reverse proxy and load balancer allowed for effective traffic distribution along with SSL/TLS security, caching, failover handling, and performance optimization. The architecture was proven to be reliable and resilient through systematic testing of load balancing, caching, high availability, and security features, thus meeting real-world cloud deployment standards and reinforcing the practical understanding of modern cloud infrastructure design.

**THE END**