Fatima Jinnah Women University

Opening Portals of Excellence Through Higher Education

Open-Ended Lab Report
Terraform & Ansible – Frontend Backend High Availability Architecture

Student Name: Safa Jahangir

Registration No: 2023-BSE-056

Course: Cloud Computing

Lab Type: Open Ended Lab

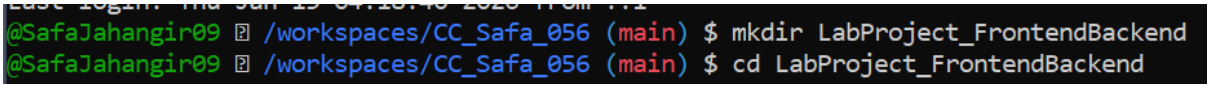
Contents

1. Introduction

This report presents the complete implementation of an open-ended cloud computing lab using Terraform and Ansible.
The goal of this lab is to provision infrastructure automatically and configure a highly available frontend-backend web architecture.

2. GitHub Repository Initialization

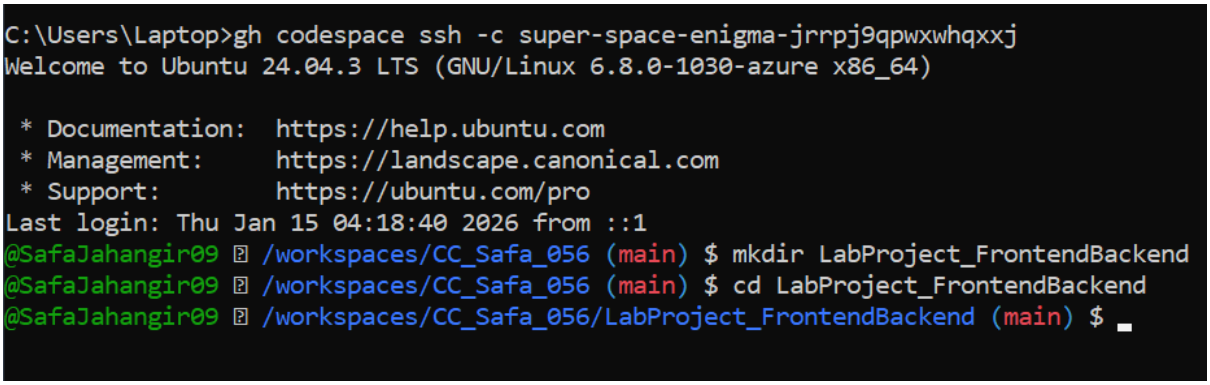
A dedicated GitHub repository was created to maintain version control and ensure clean separation of this lab from previous assignments.



(Figure 1: GitHub Repository Creation)

3. GitHub Codespaces Environment Setup

GitHub Codespaces was used as the development environment to provide a consistent Linux-based setup.



(Figure 2: Codespaces Environment)

4. Tool Installation & Verification

Required tools including Terraform, AWS CLI, Python, and Ansible were installed and verified before implementation.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend (main) $ terraform -v
Terraform v1.14.3
on linux_amd64
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend (main) $ ansible --version
ansible [core 2.16.3]
  config file = None
  configured module search path = ['/home/codespace/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/codespace/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.12.3 (main, Nov  6 2025, 13:44:16) [GCC 13.3.0] (/usr/bin/python3)
  jinja version = 3.1.6
  libyaml = True
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend (main) $ aws --version
aws-cli/2.32.32 Python/3.13.11 Linux/6.8.0-1030-azure exe/x86_64.ubuntu.24
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend (main) $ python3 --version
Python 3.12.1
```

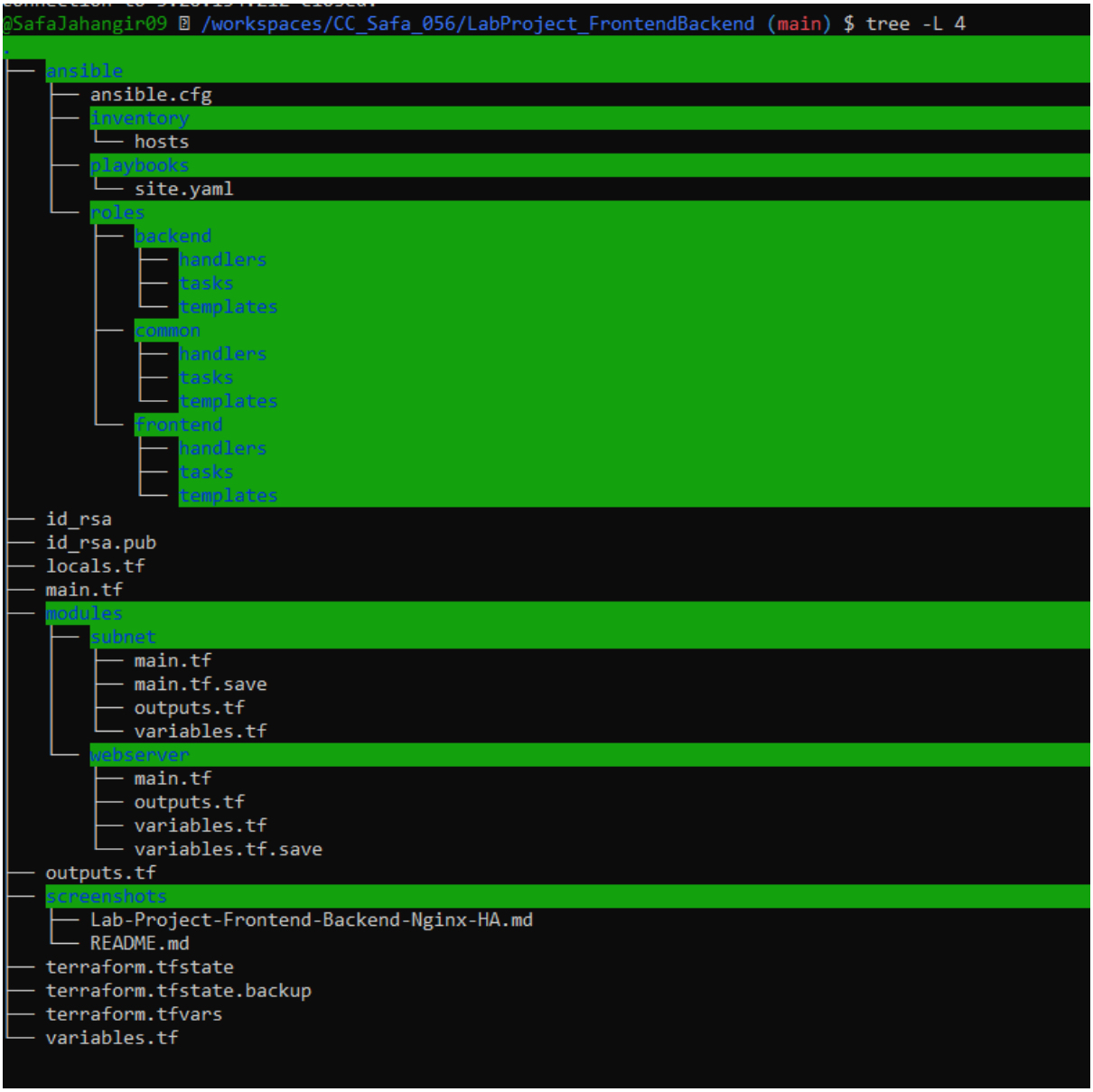
(Figure 3: Terraform Version, Ansible Version)

5. Final Project Directory Structure

The final directory structure follows Terraform module-based design and Ansible role-based configuration.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend (main) $ cat .gitignore
.terraform/
*.tfstate
*.tfstate.*
*.tfvars
.pem
.terraform.lock.hcl
```

(Figure 4: .gitignore file)



(Figure 5: Project Directory Tree)

6. Terraform Configuration

6.1 Provider & Variables Definition

Terraform provider configuration specifies the AWS region and credentials. Variables are used for reusability and flexibility.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend (main) $ cat variables.tf
variable "region" {
  default = "me-central-1"
}

variable "availability_zone" {
  default = "me-central-1a"
}

variable "vpc_cidr_block" {
  default = "10.0.0.0/16"
}

variable "subnet_cidr_block" {
  default = "10.0.1.0/24"
}

variable "env_prefix" {
  default = "labproj"
}

variable "instance_type" {
  default = "t3.micro"
}

variable "public_key_path" {
  default = "~/.ssh/id_ed25519.pub"
}

variable "private_key_path" {
  default = "~/.ssh/id_ed25519"
}
```

(Figure 6: variables.tf)

6.2 Local Values & IP Restriction

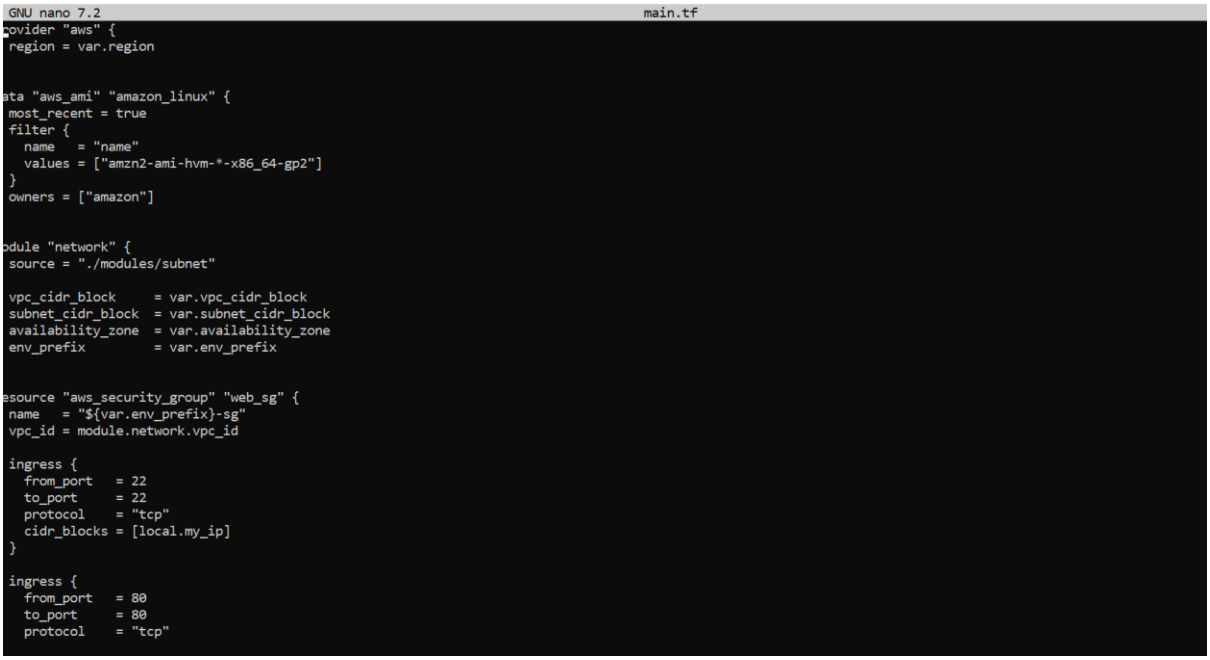
Local values dynamically retrieve the user's public IP to restrict SSH access securely.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend (main) $ cat locals.tf
data "http" "my_ip" {
  url = "https://icanhazip.com"
}

locals {
  my_ip = "${chomp(data.http.my_ip.response_body)}/32"
}
```

(Figure 7: locals.tf)

6.3 Networking (VPC, Subnet, Routing)

A VPC, public subnet, internet gateway, and route table are provisioned to allow internet access.



(Figure 8: VPC & Subnet Configuration)

6.4 EC2 Frontend & Backend Provisioning

One frontend EC2 instance and three backend EC2 instances are created using reusable Terraform modules.

```
 GNU nano 7.2                                                    main.tf

resource "aws_key_pair" "key" {
  key_name   = "${var.env_prefix}-key"
  public_key = file(var.public_key_path)
}

module "frontend" {
  source = "./modules/webserver"

  ami_id              = data.aws_ami.amazon_linux.id
  instance_type       = var.instance_type
  subnet_id           = module.network.subnet_id
  security_group_ids  = [aws_security_group.web_sg.id]
  key_name            = aws_key_pair.key.key_name
  name                = "${var.env_prefix}-frontend"
  count               = 1
}

module "backend" {
  source = "./modules/webserver"

  ami_id              = data.aws_ami.amazon_linux.id
  instance_type       = var.instance_type
  subnet_id           = module.network.subnet_id
  security_group_ids  = [aws_security_group.web_sg.id]
  key_name            = aws_key_pair.key.key_name
  name                = "${var.env_prefix}-backend"
  count               = 3
}

resource "null_resource" "ansible_run" {
  depends_on = [
    module.frontend,
    module.backend
  ]
```

(Figure 9: EC2 Instance Creation)

6.5 Terraform Outputs

Terraform outputs expose public and private IP addresses required for Ansible inventory and verification.

```
  GNU nano 7.2                                          outputs.tf
output "frontend_public_ip" {
  value = module.frontend.public_ips[0]
}

output "backend_public_ips" {
  value = module.backend.public_ips
}

output "backend_private_ips" {
  value = module.backend.private_ips
}
```

(Figure 10-A: outputs.tf)

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend (main) $ terraform output
backend_private_ips = [
  [
    "10.0.1.115",
  ],
  [
    "10.0.1.199",
  ],
  [
    "10.0.1.119",
  ],
]
backend_public_ips = [
  [
    "3.28.191.138",
  ],
  [
    "158.252.86.16",
  ],
  [
    "3.29.58.95",
  ],
]
frontend_public_ip = [
  "3.28.134.212",
]
```

(Figure 10-B: outputs.tf)

6.6 Terraform Modules Implementation

Terraform modules were used to improve code reusability, readability, and separation of concerns.
Instead of defining all resources in a single file, networking and compute logic were encapsulated into independent modules.

This approach aligns with infrastructure-as-code best practices and simplifies future scaling.

6.6.1 Subnet Module (Networking Layer)

The subnet module is responsible for provisioning the core networking components required for the project.
It creates the Virtual Private Cloud (VPC), public subnet, internet gateway, route table, and route table association.

This ensures that all EC2 instances are deployed in a properly configured public network with internet access.

```
  GNU nano 7.2                                        New Buffer *
resource "aws_vpc" "this" {
  cidr_block = var.vpc_cidr_block

  tags = {
    Name = "${var.env_prefix}-vpc"
  }
}

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

  tags = {
    Name = "${var.env_prefix}-igw"
  }
}

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

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

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

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

(Figure 12: Subnet module main.tf – VPC, subnet, and routing configuration)

```
  GNU nano 7.2                                  modules/subnet/variables.tf *
variable "vpc_cidr_block" {
  type = string
}

variable "subnet_cidr_block" {
  type = string
}

variable "availability_zone" {
  type = string
}

variable "env_prefix" {
  type = string
}
```

(Figure 13: Subnet module variables.tf)

```
  GNU nano 7.2                                  modules/subnet/outputs.tf
output "vpc_id" {
  value = aws_vpc.this.id
}

output "subnet_id" {
  value = aws_subnet.this.id
}
```

(Figure 14: Subnet module outputs.tf)

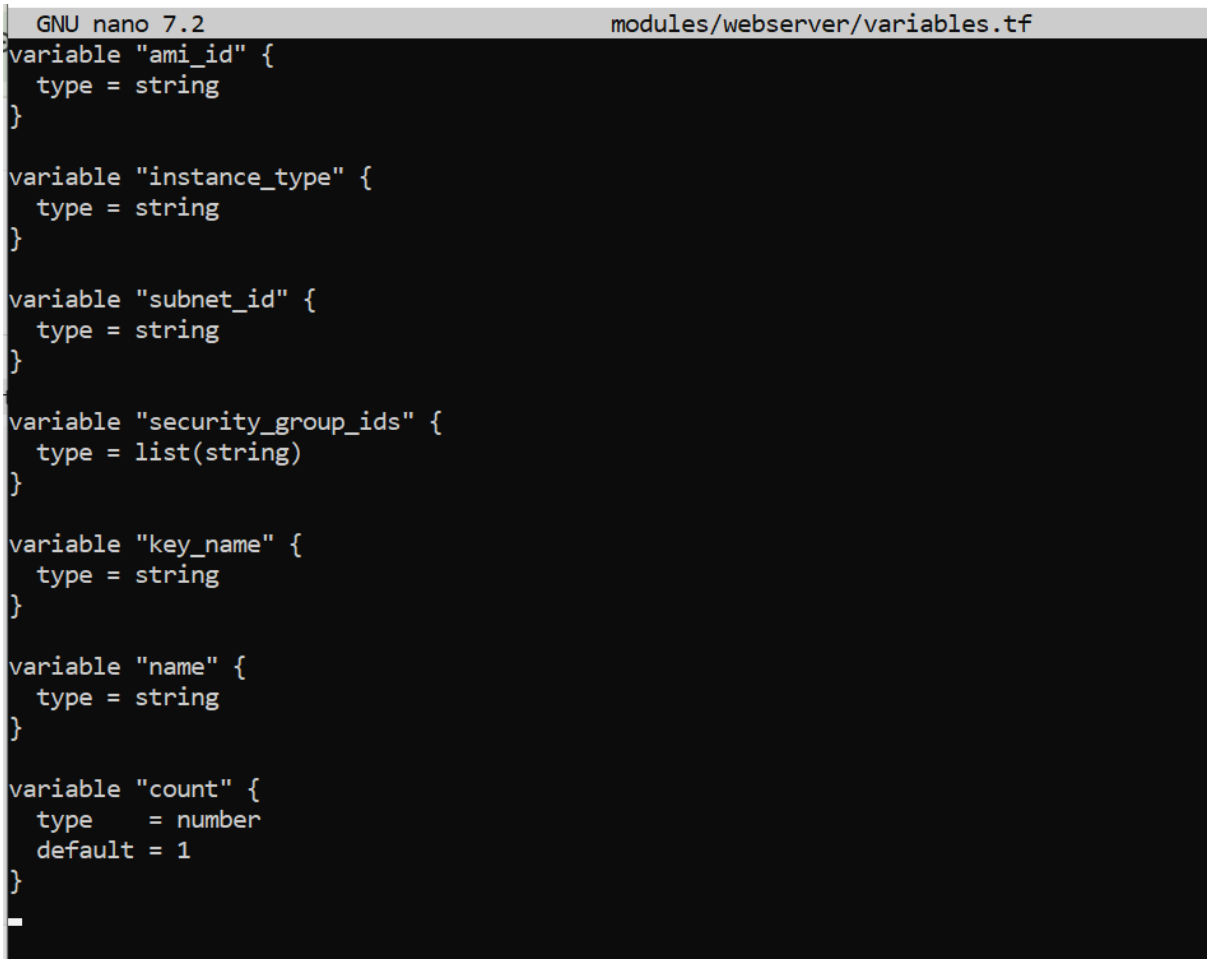
6.6.2 Webserver Module (EC2 Abstraction Layer)

The webserver module abstracts EC2 instance creation into a reusable component.
It is used to provision both frontend and backend instances by passing different parameters such as instance count and naming prefix.

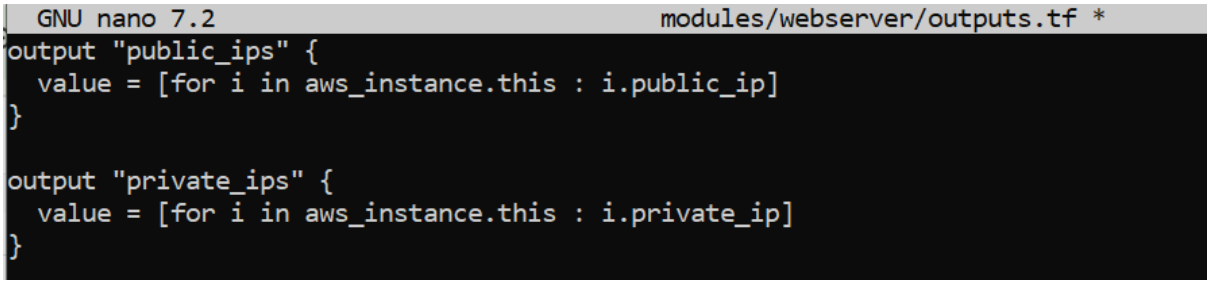
This modular design avoids duplication and ensures consistent EC2 configuration.

```
  GNU nano 7.2                                  modules/webserver/main.tf *
resource "aws_instance" "this" {
  count = var.count

  ami                    = var.ami_id
  instance_type          = var.instance_type
  subnet_id              = var.subnet_id
  vpc_security_group_ids = var.security_group_ids
  key_name               = var.key_name

  tags = {
    Name = "${var.name}-${count.index}"
  }
}
```

(Figure 15: Webserver module main.tf – EC2 instance resource definition)



(Figure 16: Webserver module variables.tf)



(Figure 17: Webserver module outputs.tf)

7. Ansible Configuration

7.1 Ansible Configuration File

The ansible.cfg file disables host key checking and specifies Python interpreter settings.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat ansible.cfg
[defaults]
# Points to your dynamic/manual inventory
inventory = ./inventory/hosts

# Disables the "Are you sure you want to connect (yes/no)?" prompt
host_key_checking = False

# Sets the default user for AWS EC2 (Amazon Linux 2)
remote_user = ec2-user

# Points to your private key file
private_key_file = ../id_rsa

# Improves speed by reducing the number of SSH connections
pipelining = True
```

(Figure 18: ansible.cfg)

7.2 Inventory File

The inventory file groups frontend and backend servers to enable role-based execution.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat inventory/hosts
[frontend]
frontend_server ansible_host=3.28.134.212

[backend]
backend1 ansible_host=3.28.191.138 ansible_ssh_host_private=10.0.1.115
backend2 ansible_host=158.252.86.16 ansible_ssh_host_private=10.0.1.199
backend3 ansible_host=3.29.58.95  ansible_ssh_host_private=10.0.1.119

[all:vars]
ansible_user=ec2-user
ansible_ssh_private_key_file=../id_rsa
```

(Figure 19: Inventory hosts file)

7.3 Backend Role (Apache HTTPD)

The backend role installs Apache HTTPD and deploys unique index pages on each backend server.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat roles/backend/tasks/main.yml
---
- name: Install httpd
  yum:
    name: httpd
    state: present
    update_cache: yes

- name: Start and enable httpd
  service:
    name: httpd
    state: started
    enabled: true

- name: Deploy backend page
  template:
    src: backend_index.html.j2
    dest: /var/www/html/index.html
```

(Figure 20-A: Backend Role Tasks)

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat roles/backend/templates/backend_index.html.j2

<h1>Backend {{ inventory_hostname }}</h1>
<p>Private IP: {{ ansible_default_ipv4.address }}</p>
```

(Figure 20-B: Backend Role Templates)

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat roles/backend/templates/backend_index.html.j2

<h1>Backend {{ inventory_hostname }}</h1>
<p>Private IP: {{ ansible_default_ipv4.address }}</p>
```

(Figure 20-C: Backend Role Handler)

7.4 Frontend Role (Nginx Load Balancer)

The frontend role installs Nginx and configures it as a reverse proxy with two primary and one backup backend.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat roles/frontend/tasks/main.yml
- name: Install nginx using amazon-linux-extras
  shell: "amazon-linux-extras install nginx1 -y"
  args:
    creates: /usr/sbin/nginx

- name: Start and enable nginx
  service:
    name: nginx
    state: started
    enabled: yes

- name: Deploy nginx configuration
  template:
    src: nginx_frontend.conf.j2
    dest: /etc/nginx/nginx.conf
  notify: restart nginx
```

(Figure 21-A: Nginx Configuration Tasks)

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat roles/frontend/templates/nginx_frontend.conf.j2
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log notice;
pid /run/nginx.pid;

events {
    worker_connections 1024;
}

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

    upstream backend_servers {
        # Dynamically pulling private IPs from inventory
        server {{ hostvars[groups['backend'][0]].ansible_ssh_host_private }}:80;
        server {{ hostvars[groups['backend'][1]].ansible_ssh_host_private }}:80;
        server {{ hostvars[groups['backend'][2]].ansible_ssh_host_private }}:80 backup;
    }

    server {
        listen 80;
        server_name _;

        location / {
            proxy_pass http://backend_servers;
            proxy_set_header Host $host;
```

(Figure 21-B: Nginx Configuration Template)

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat roles/frontend/handlers/main.yml
---
- name: restart nginx
  service:
    name: nginx
    state: restarted
```

(Figure 21-C: Nginx Configuration Handlers)

7.5 Main Playbook (site.yaml)

The main playbook applies backend configuration first, followed by frontend setup.

```
@SafaJahangir09 /workspaces/CC_Safa_056/LabProject_FrontendBackend/ansible (main) $ cat playbooks/site.yaml
---
- name: Configure backend servers
  hosts: backend
  become: true
  roles:
    - backend

- name: Configure frontend server
  hosts: frontend
  become: true
  vars:
    backend1_private_ip: "{{ hostvars[groups['backends'][0]].ansible_default_ipv4.address }}"
    backend2_private_ip: "{{ hostvars[groups['backends'][1]].ansible_default_ipv4.address }}"
    backup_backend_private_ip: "{{ hostvars[groups['backends'][2]].ansible_default_ipv4.address }}"
  roles:
    - frontend
```

(Figure 22: site.yaml)

8. Terraform Apply & Automation Execution

Terraform apply provisions all resources and automatically triggers Ansible using a null_resource.

```
null_resource.ansible_run: Still creating... [00m10s elapsed]
null_resource.ansible_run (local-exec): [WARNING]: Platform linux on host frontend_server is using the discovered
null_resource.ansible_run (local-exec): Python interpreter at /usr/bin/python3.7, but future installation of another
null_resource.ansible_run (local-exec): Python interpreter could change the meaning of that path. See
null_resource.ansible_run (local-exec): https://docs.ansible.com/ansible-
null_resource.ansible_run (local-exec): core/2.16/reference_appendices/interpreter_discovery.html for more information.
null_resource.ansible_run (local-exec): ok: [frontend_server]

null_resource.ansible_run (local-exec): TASK [frontend : Install nginx using amazon-linux-extras] **********************
null_resource.ansible_run (local-exec): ok: [frontend_server]

null_resource.ansible_run (local-exec): TASK [frontend : Start and enable nginx] ************************************
null_resource.ansible_run (local-exec): ok: [frontend_server]

null_resource.ansible_run (local-exec): TASK [frontend : Deploy nginx configuration] *********************************
null_resource.ansible_run (local-exec): ok: [frontend_server]

null_resource.ansible_run (local-exec): PLAY RECAP *****************************************************************
null_resource.ansible_run (local-exec): backend1                : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
null_resource.ansible_run (local-exec): backend2                : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
null_resource.ansible_run (local-exec): backend3                : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
null_resource.ansible_run (local-exec): frontend_server         : ok=4    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

null_resource.ansible_run: Creation complete after 16s [id=995207980304899472]

Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

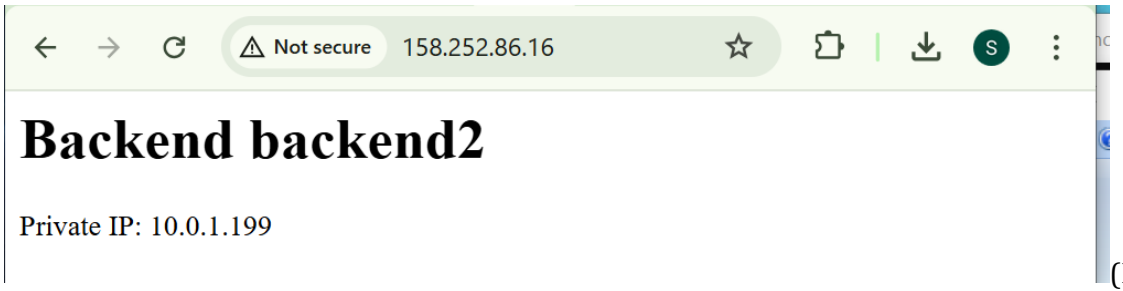
(Figure 23: Terraform Apply Output)

9. Backend Server Verification

The following unlabeled screenshots show direct access to backend servers using their public

IPs.
Each backend displays a unique response, confirming correct Apache deployment.

(Figure 24: Backend Server 1 Output)



(Figure 25: Backend Server 2 Output)

(Figure 26: Backend Server 3 Output)

10. Frontend Load Balancer Verification

These unlabeled screenshots show frontend access via the Nginx load balancer public IP. Responses alternate between primary backend servers.

Backend backend1

Private IP: 10.0.1.115

(Figure 27: Frontend Response – Backend A)

Backend backend2

Private IP: 10.0.1.199

(Figure 28: Frontend Response – Backend B)

11. Load Balancing Behaviour Validation

Repeated refreshes confirm round-robin behavior between the two primary backend servers.

Backend backend1

Private IP: 10.0.1.115

(Figure 29-A: Refresh-1)

Backend backend2

Private IP: 10.0.1.199

(Figure 29-B: Refresh-2)

Backend backend1

Private IP: 10.0.1.115

(Figure 29-C: Refresh-3)

Backend backend2

Private IP: 10.0.1.199

(Figure 29-D: Refresh-4)

12. Failover Testing (Backup Backend)

Primary backend services were stopped to validate failover. Traffic was successfully routed to the backup backend.

Backend backend3

Private IP: 10.0.1.119

(Figure 30: Backup Backend Response)

13. Conclusion

This lab successfully demonstrates automated infrastructure provisioning and configuration management.
The architecture meets all requirements of high availability, modularity, and automation.