Name: Lujain Zia

Roll no: 2023-BSE-034

Date: December 24, 2025

Lab 12

Task 0 Lab Setup (Codespace & GH CLI)

* Create Codespace & connect:

# create or open codespace via GH CLI (example)

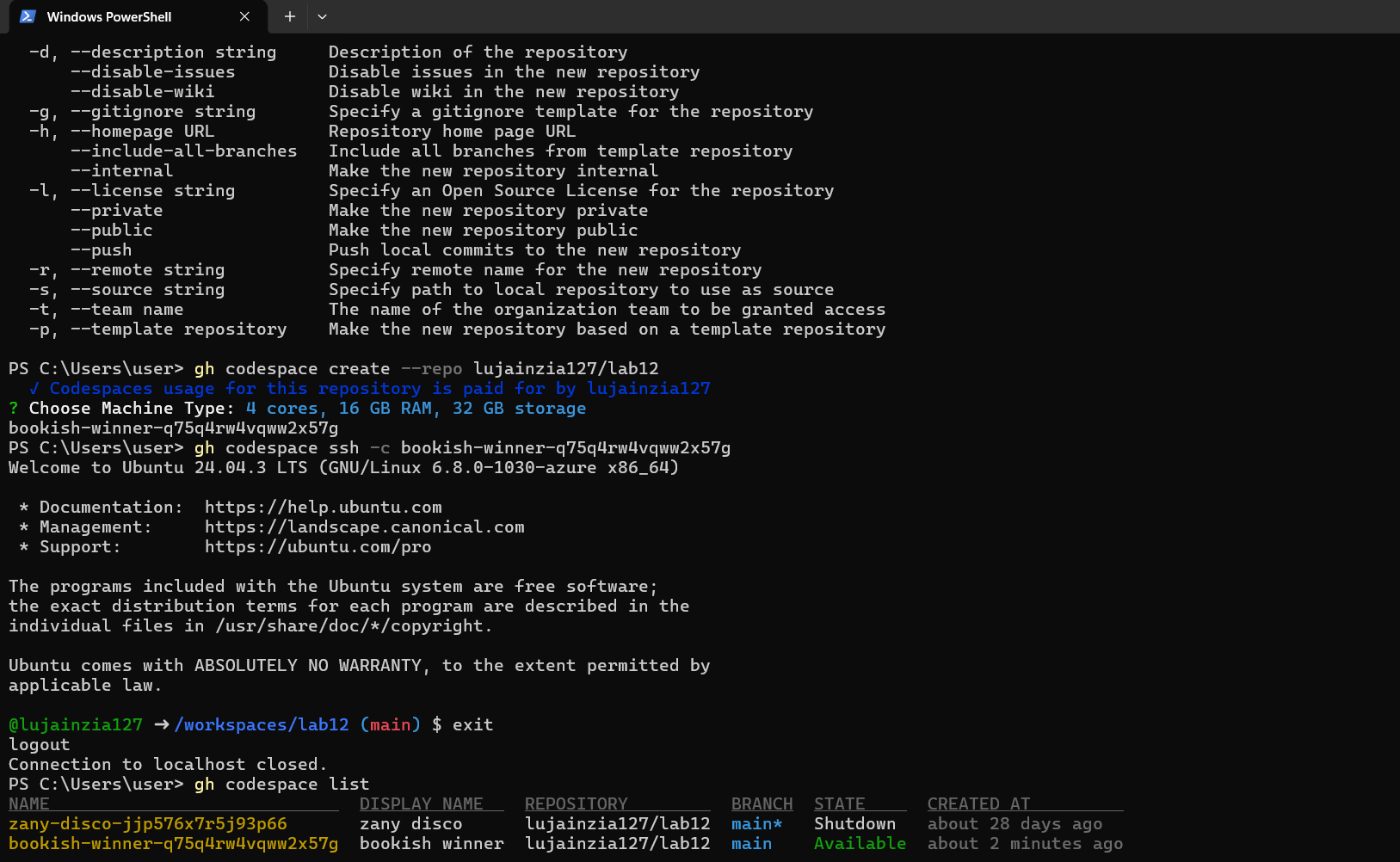
gh repo create CC\_<YourName>\_<YourRollNumber>/Lab12 --public

gh codespace create --repo <user\_name>/Lab12

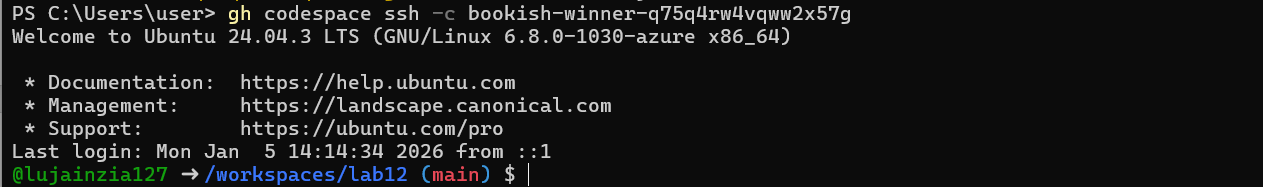
gh codespace list

* gh codespace ssh -c <your\_codespace\_name>

Save screenshot as: task0\_codespace\_create\_and\_list.png — output showing repo creation/codespace list.



Save screenshot as: task0\_codespace\_ssh\_connected.png — terminal inside the Codespace shell after ssh.



Task 1 — Organize Terraform code into separate files

1. Create the initial project structure:

mkdir -p ~/Lab12cd ~/Lab12

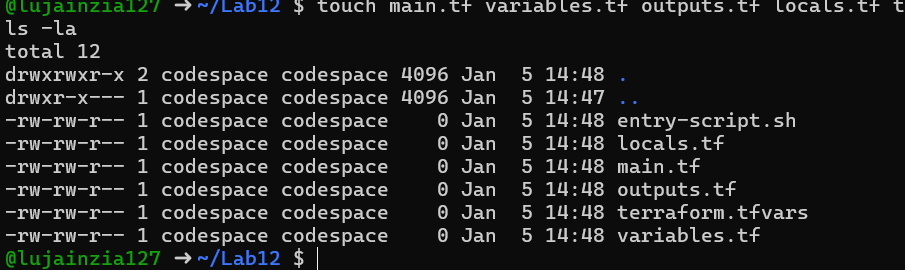
Save screenshot as: task1\_project\_directory.png — terminal showing directory creation.



1. Create all required files:

touch main.tf variables.tf outputs. tf locals.tf terraform.tfvars entry-script.sh

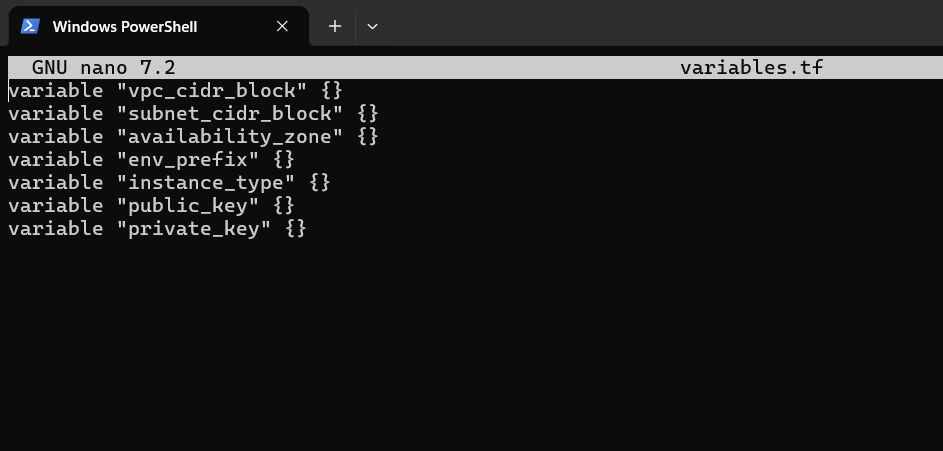
Save screenshot as: task1\_files\_created.png — terminal showing all files created (use ls -la).



1. Create variables.tf with the following content:

variable "vpc\_cidr\_block" {}variable "subnet\_cidr\_block" {}variable "availability\_zone" {}variable "env\_prefix" {}variable "instance\_type" {}variable "public\_key" {}variable "private\_key" {}

Save screenshot as: task1\_variables\_tf.png — content of variables.tf file.



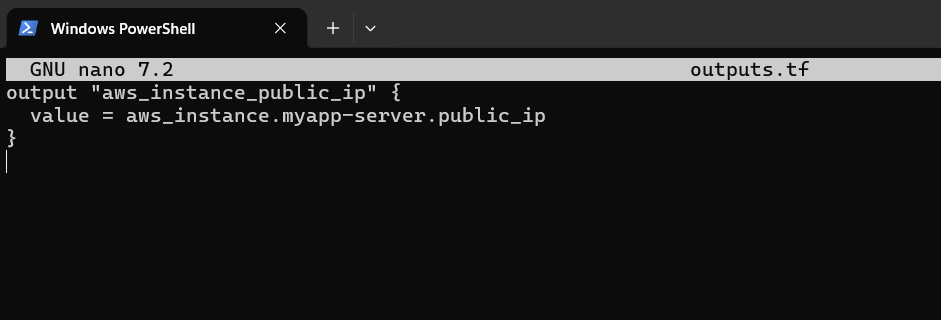
1. Create outputs.tf with the following content:

output "aws\_instance\_public\_ip" {

value = aws\_instance.myapp-server.public\_ip

}

Save screenshot as: task1\_outputs\_tf.png — content of outputs.tf file.



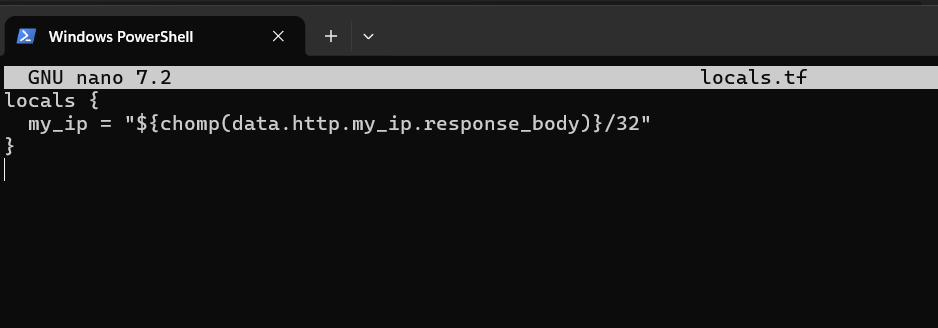
1. Create locals.tf with the following content:

locals {

my\_ip = "${chomp(data.http.my\_ip.response\_body)}/32"

}

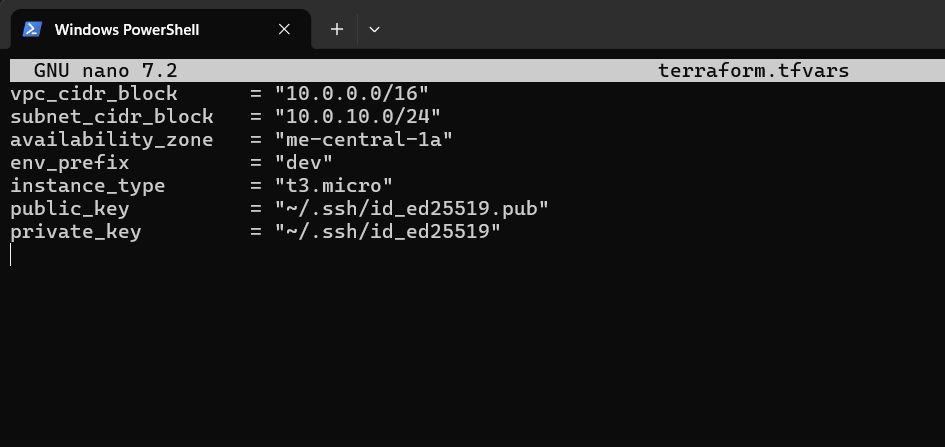
Save screenshot as: task1\_locals\_tf.png — content of locals.tf file.



1. Create terraform.tfvars with the following content:

vpc\_cidr\_block = "10.0.0.0/16"subnet\_cidr\_block = "10.0.10.0/24"availability\_zone = "me-central-1a"env\_prefix = "dev"instance\_type = "t3.micro"public\_key = "~/.ssh/id\_ed25519.pub"private\_key = "~/.ssh/id\_ed25519"

Save screenshot as: task1\_terraform\_tfvars. png — content of terraform.tfvars file.



1. Create main.tf with the following content:

provider "aws" {

shared\_config\_files = ["~/.aws/config"]

shared\_credentials\_files = ["~/.aws/credentials"]

}

resource "aws\_vpc" "myapp\_vpc" {

cidr\_block = var.vpc\_cidr\_block

tags = {

Name = "${var.env\_prefix}-vpc"

}

}

resource "aws\_subnet" "myapp\_subnet\_1" {

vpc\_id = aws\_vpc.myapp\_vpc.id

cidr\_block = var.subnet\_cidr\_block

availability\_zone = var.availability\_zone

tags = {

Name = "${var.env\_prefix}-subnet-1"

}

}

resource "aws\_default\_route\_table" "main\_rt" {

default\_route\_table\_id = aws\_vpc.myapp\_vpc.default\_route\_table\_id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.myapp\_igw.id

}

tags = {

Name = "${var.env\_prefix}-rt"

}

}

resource "aws\_internet\_gateway" "myapp\_igw" {

vpc\_id = aws\_vpc.myapp\_vpc.id

tags = {

Name = "${var.env\_prefix}-igw"

}

}

resource "aws\_default\_security\_group" "default\_sg" {

vpc\_id = aws\_vpc.myapp\_vpc.id

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = [local.my\_ip]

}

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

prefix\_list\_ids = []

}

tags = {

Name = "${var.env\_prefix}-default-sg"

}

}

resource "aws\_key\_pair" "ssh-key" {

key\_name = "serverkey"

public\_key = file(var.public\_key)

}

resource "aws\_instance" "myapp-server" {

ami = "ami-05524d6658fcf35b6" # Amazon Linux 2023 Kernel 6.1 AMI

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.myapp\_subnet\_1.id

security\_groups = [aws\_default\_security\_group.default\_sg. id]

availability\_zone = var.availability\_zone

associate\_public\_ip\_address = true

key\_name = aws\_key\_pair.ssh-key. key\_name

user\_data = file("./entry-script.sh")

tags = {

Name = "${var.env\_prefix}-ec2-instance"

}

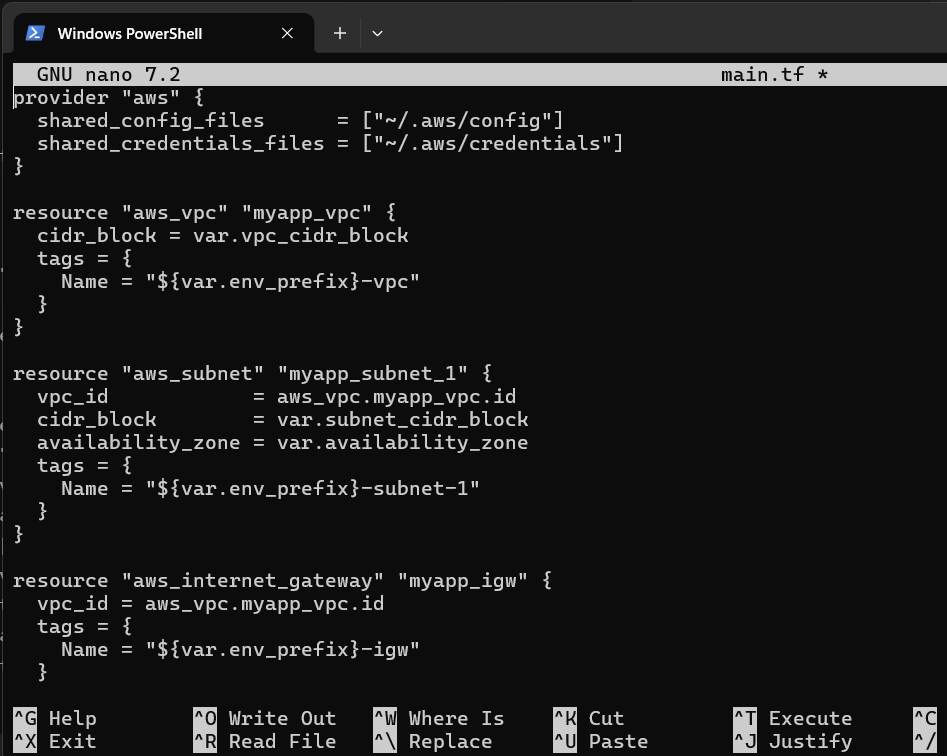
}

data "http" "my\_ip" {

url = "https://icanhazip.com"

}

Save screenshot as: task1\_main\_tf.png — content of main.tf file.



1. Create entry-script.sh with the following content:

#!/bin/bashset -e

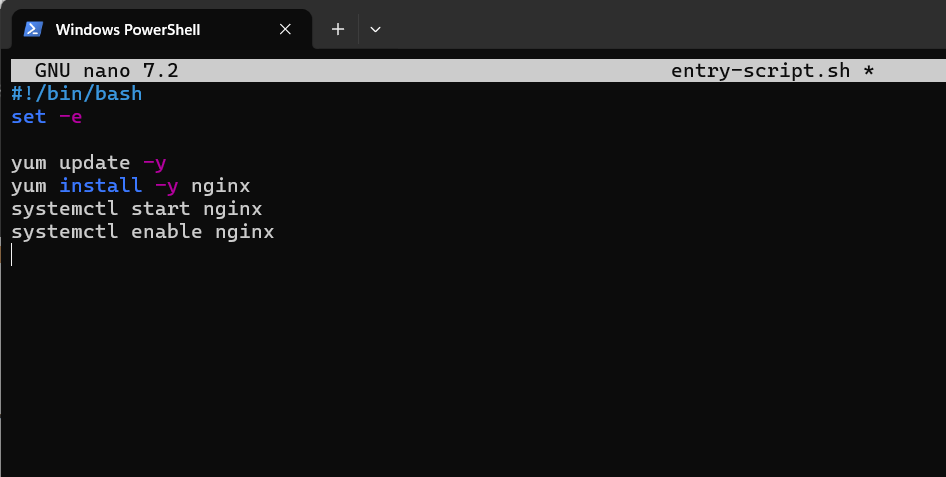
yum update -y

yum install -y nginx

systemctl start nginx

systemctl enable nginx

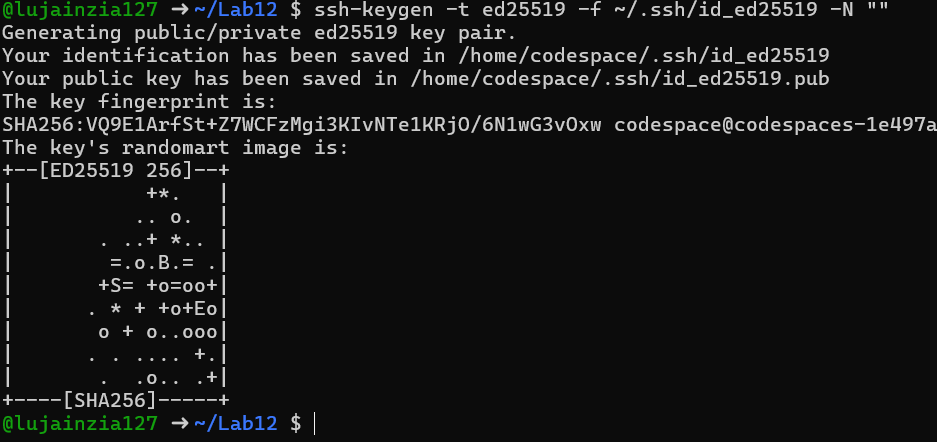
Save screenshot as: task1\_entry\_script.png — content of entry-script.sh file.



1. Generate SSH key pair if not already exists:

ssh-keygen -t ed25519 -f ~/.ssh/id\_ed25519 -N ""

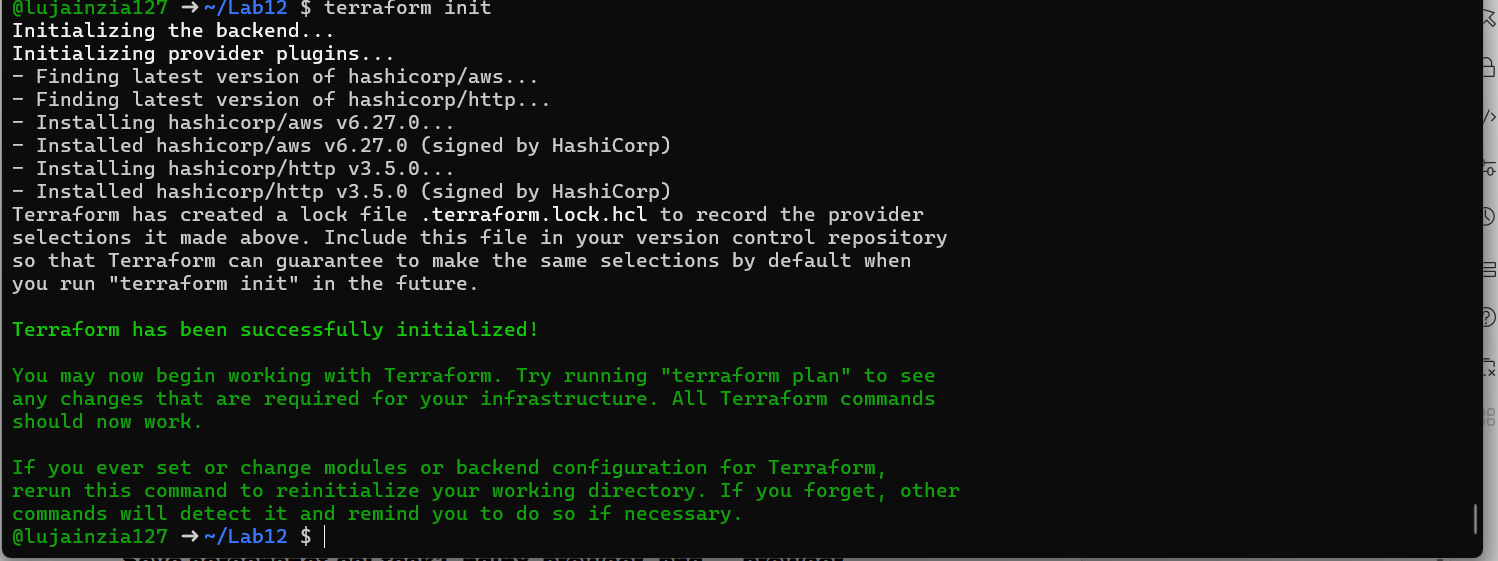
Save screenshot as: task1\_ssh\_keygen.png — terminal showing SSH key generation.



1. Initialize Terraform:

terraform init

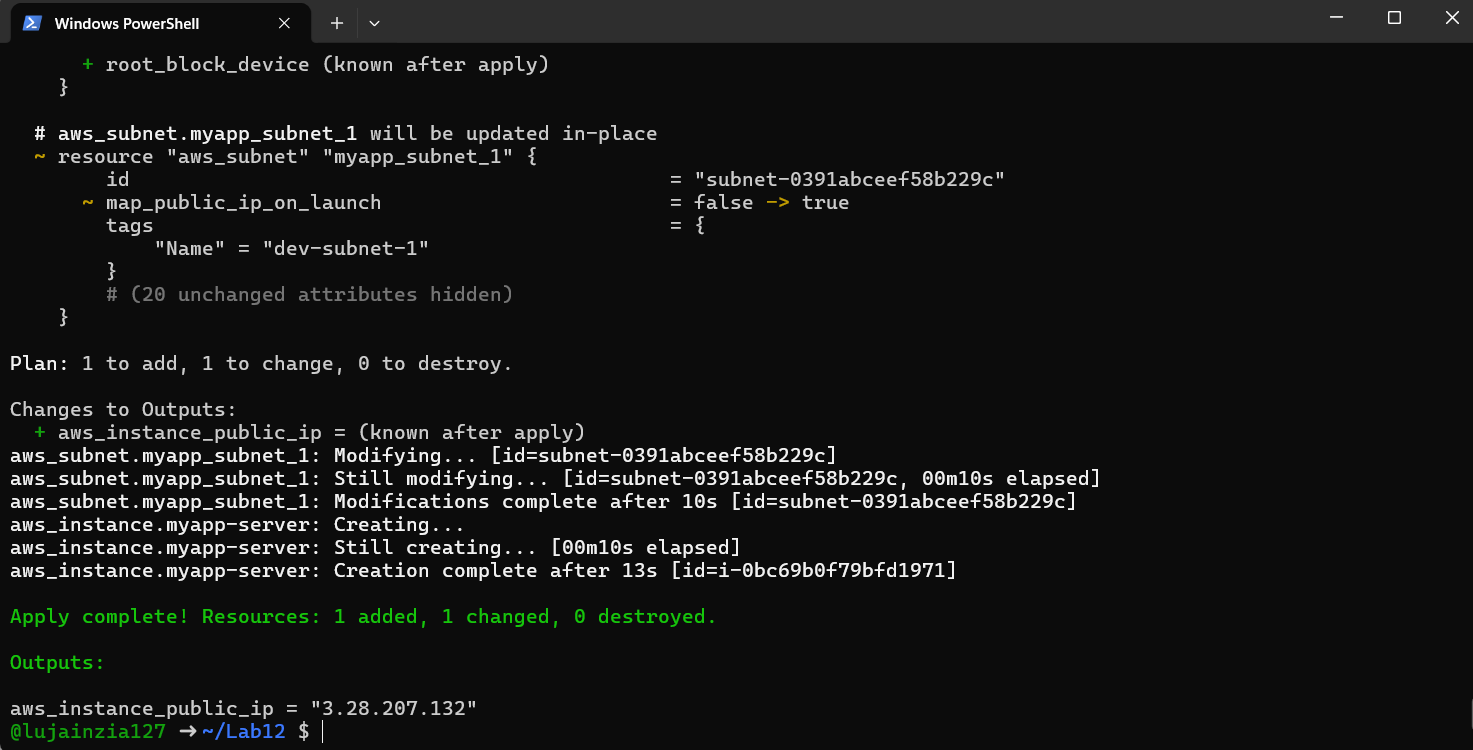
Save screenshot as: task1\_terraform\_init.png — terraform init output.



1. Apply the configuration:

terraform apply -auto-approve

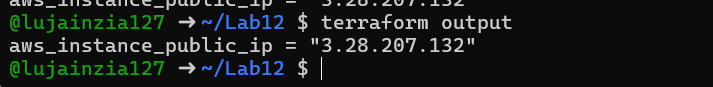
Save screenshot as: task1\_terraform\_apply.png — terraform apply output showing resources created.



1. Display the output:

terraform output

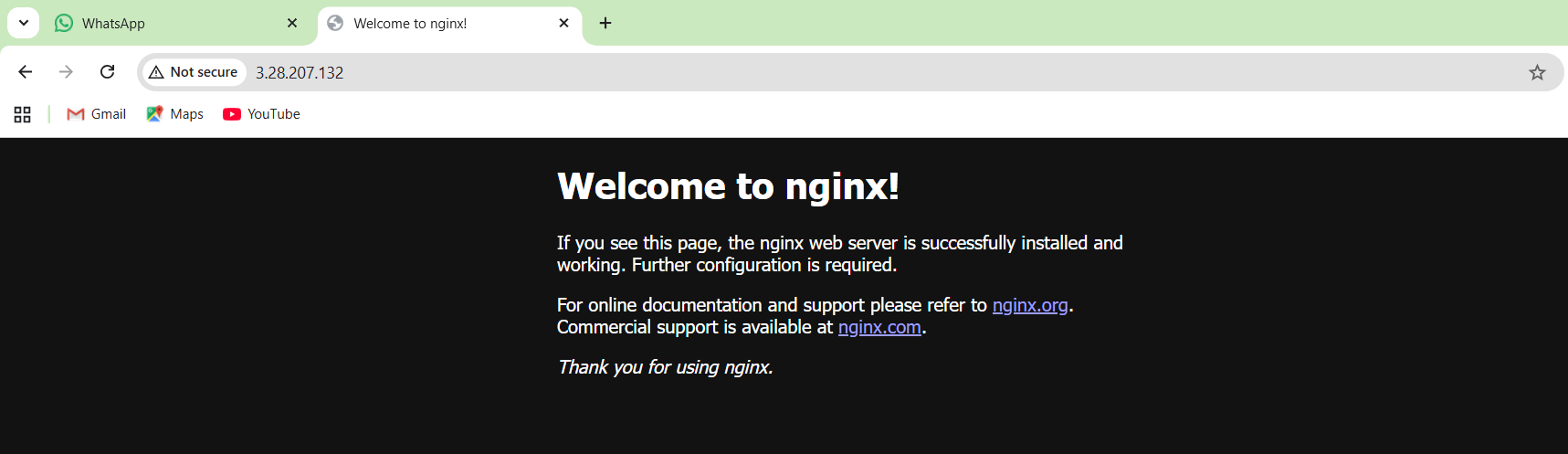
Save screenshot as: task1\_terraform\_output.png — terraform output showing public IP.



1. Test nginx in browser:

Open browser and navigate to http://<public-ip>

Save screenshot as: task1\_nginx\_browser. png — browser showing nginx default page.

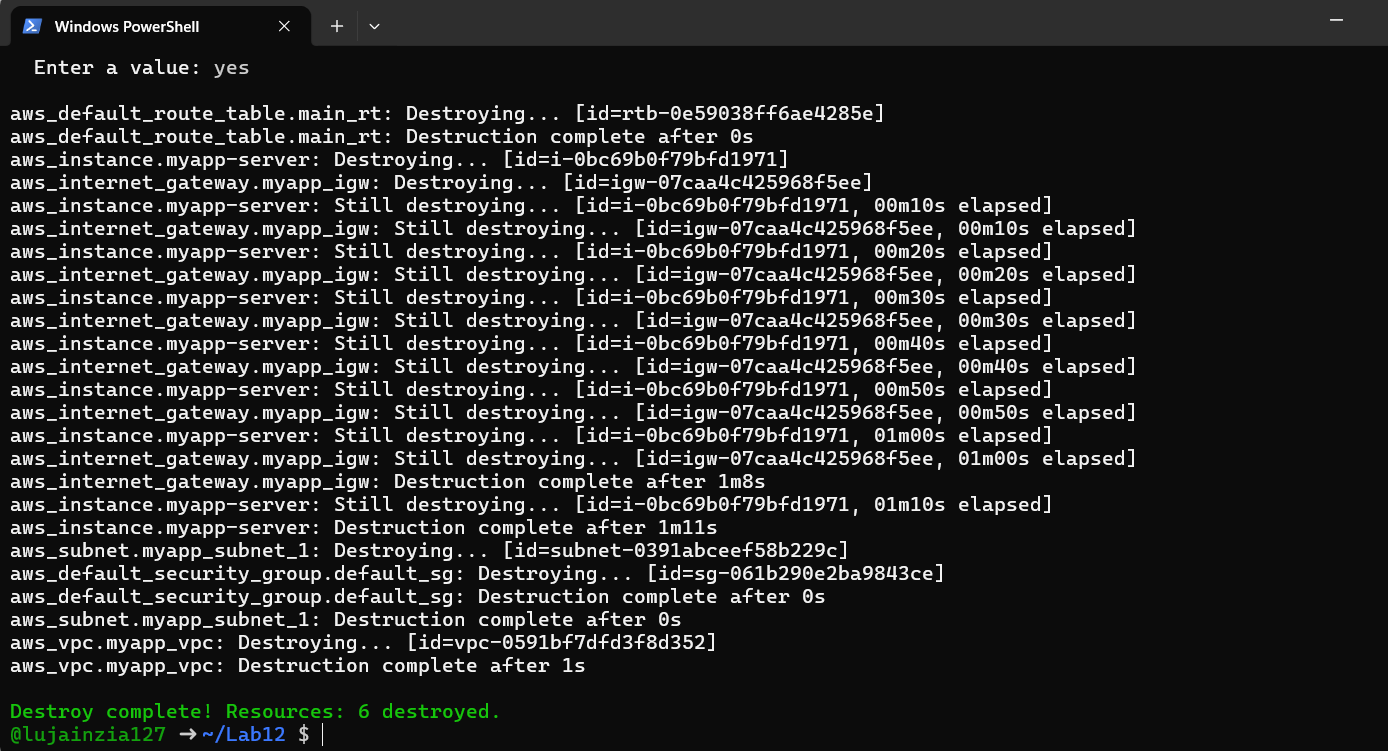


1. Destroy resources:

terraform destroy

Type yes when prompted for confirmation.

Save screenshot as: task1\_terraform\_destroy.png — terraform destroy output.



Task 2 — Use remote-exec provisioner

1. Modify the aws\_instance resource in main.tf to use remote-exec provisioner:

Replace the user\_data line with the following provisioner block:

resource "aws\_instance" "myapp-server" {

ami = "ami-05524d6658fcf35b6"

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.myapp\_subnet\_1.id

security\_groups = [aws\_default\_security\_group.default\_sg.id]

availability\_zone = var.availability\_zone

associate\_public\_ip\_address = true

key\_name = aws\_key\_pair.ssh-key.key\_name

connection {

type = "ssh"

user = "ec2-user"

private\_key = file(var.private\_key)

host = self.public\_ip

}

provisioner "remote-exec" {

inline = [

"sudo yum update -y",

"sudo yum install -y nginx",

"sudo systemctl start nginx",

"sudo systemctl enable nginx"

]

}

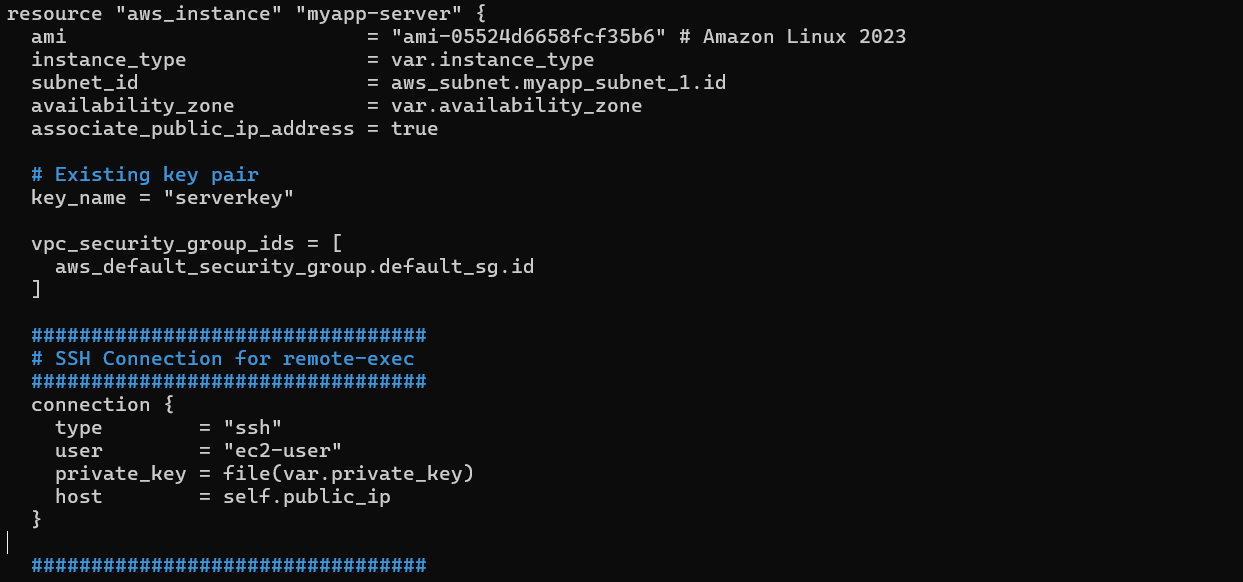
tags = {

Name = "${var. env\_prefix}-ec2-instance"

}

}

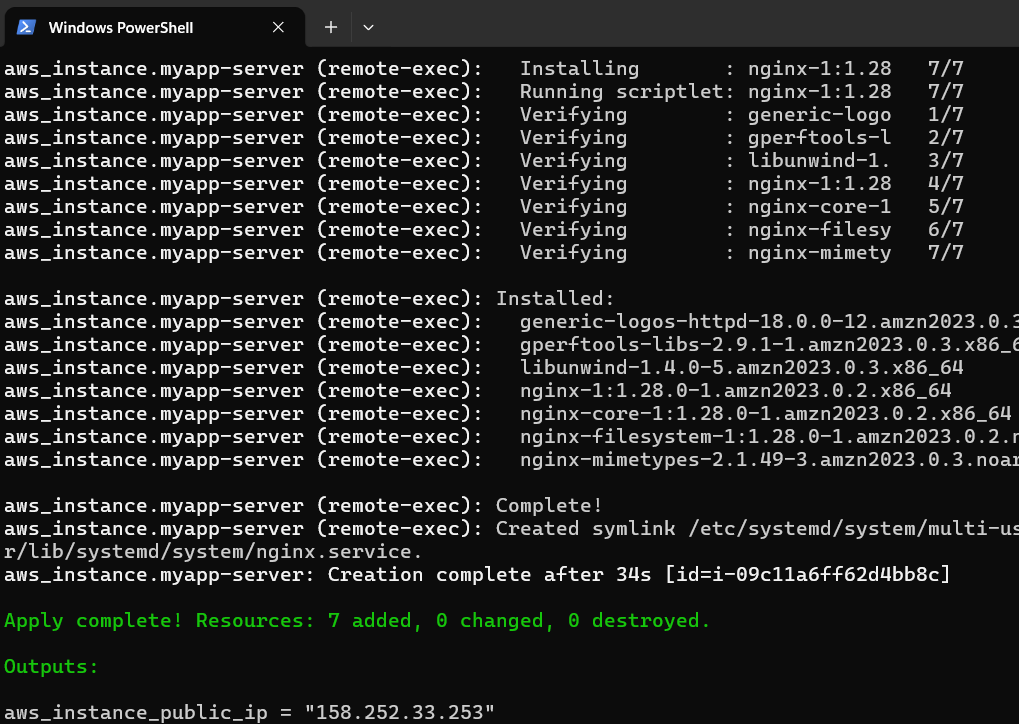
Save screenshot as: task2\_main\_tf\_remote\_exec.png — main.tf showing remote-exec provisioner.



1. Apply the configuration:

terraform apply -auto-approve

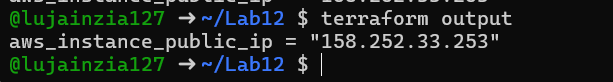
Save screenshot as: task2\_terraform\_apply.png — terraform apply output showing remote-exec execution.



1. Display the output:

terraform output

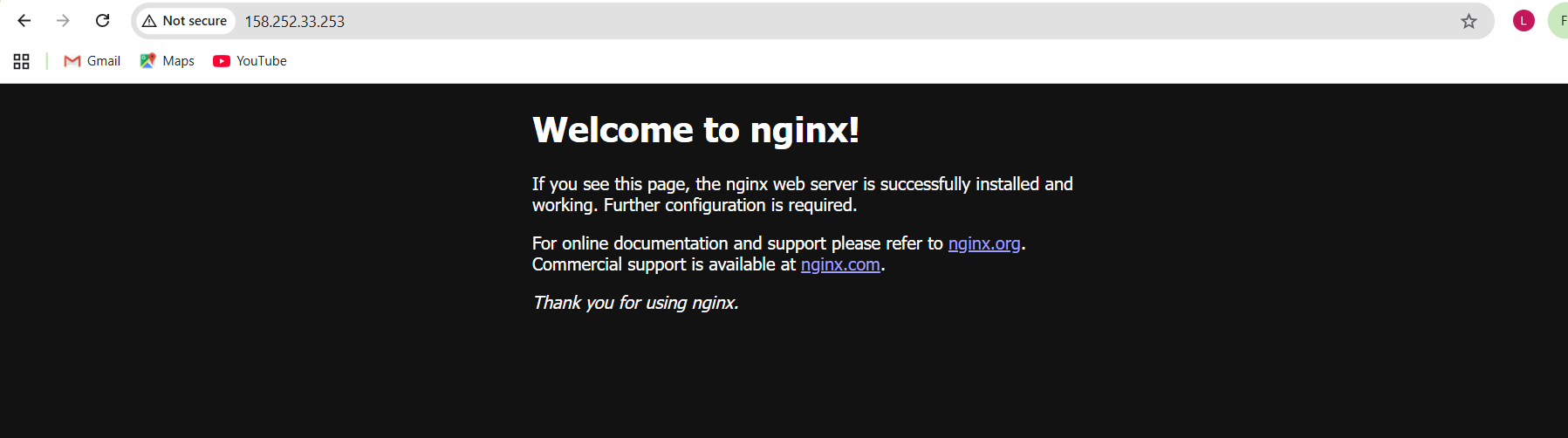
Save screenshot as: task2\_terraform\_output.png — terraform output showing public IP.



1. Test nginx in browser:

Open browser and navigate to http://<public-ip>

Save screenshot as: task2\_nginx\_browser.png — browser showing nginx default page.



Task 3 — Use file and local-exec provisioners

1. Modify the aws\_instance resource in main.tf to include all three provisioners:

resource "aws\_instance" "myapp-server" {

ami = "ami-05524d6658fcf35b6"

instance\_type = var.instance\_type

subnet\_id = aws\_subnet.myapp\_subnet\_1.id

security\_groups = [aws\_default\_security\_group.default\_sg.id]

availability\_zone = var.availability\_zone

associate\_public\_ip\_address = true

key\_name = aws\_key\_pair.ssh-key.key\_name

connection {

type = "ssh"

user = "ec2-user"

private\_key = file(var.private\_key)

host = self.public\_ip

}

provisioner "file" {

source = "./entry-script.sh"

destination = "/home/ec2-user/entry-script-on-ec2.sh"

}

provisioner "remote-exec" {

inline = [

"sudo chmod +x /home/ec2-user/entry-script-on-ec2.sh",

"sudo /home/ec2-user/entry-script-on-ec2.sh"

]

}

provisioner "local-exec" {

command = <<-EOF echo Instance ${self.id} with public IP ${self.public\_ip} has been created EOF

}

tags = {

Name = "${var.env\_prefix}-ec2-instance"

}

}

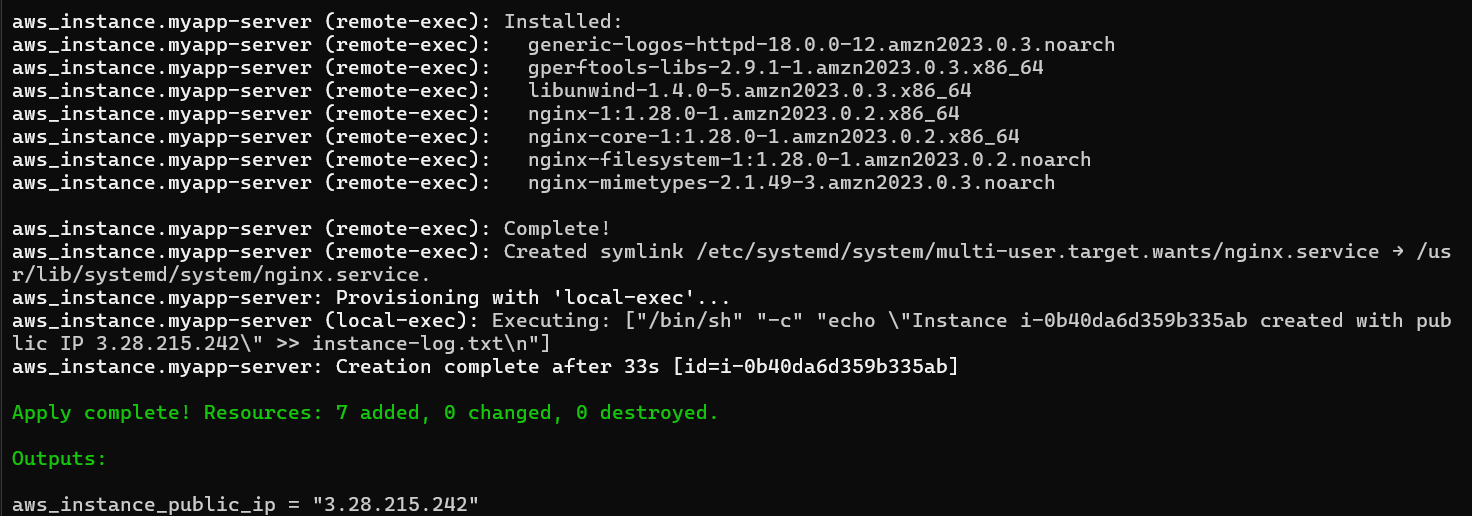
Save screenshot as: task3\_main\_tf\_all\_provisioners.png — main.tf showing file, remote-exec, and local-exec provisioners.



1. Apply the configuration:

terraform apply -auto-approve

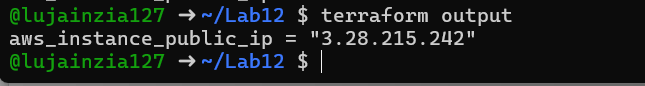
Save screenshot as: task3\_terraform\_apply.png — terraform apply output showing all provisioners execution.



1. Display the output:

terraform output

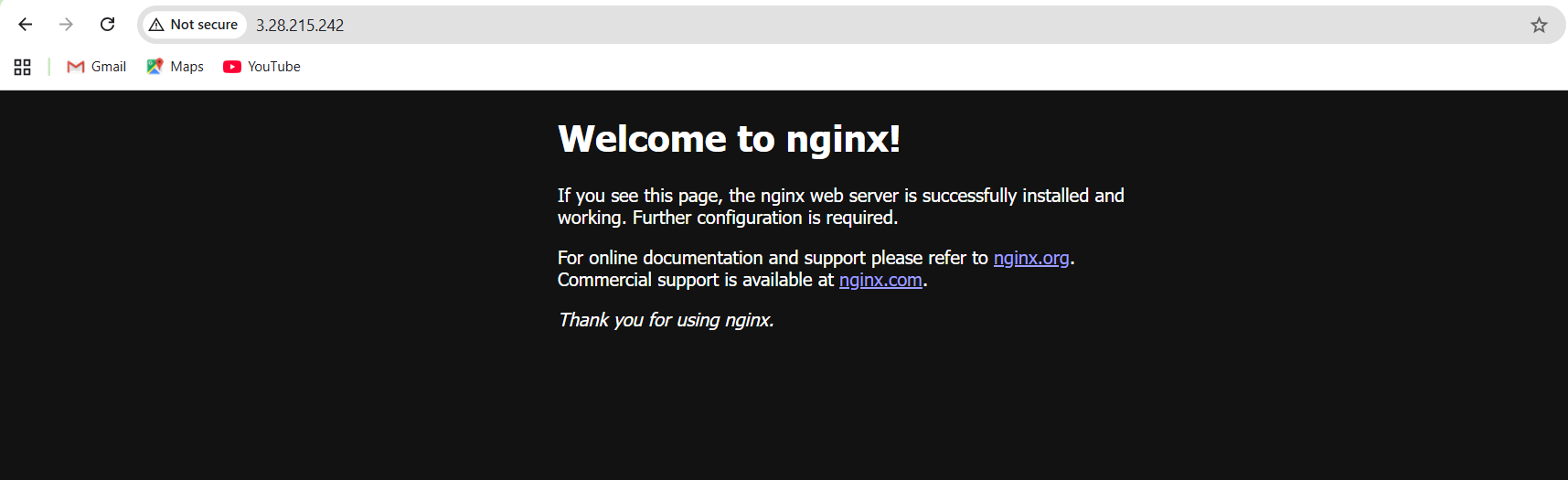
Save screenshot as: task3\_terraform\_output.png — terraform output showing public IP.



1. Test nginx in browser:

Open browser and navigate to http://<public-ip>

Save screenshot as: task3\_nginx\_browser.png — browser showing nginx default page.



1. Destroy the resources:

terraform destroy

Type yes when prompted.

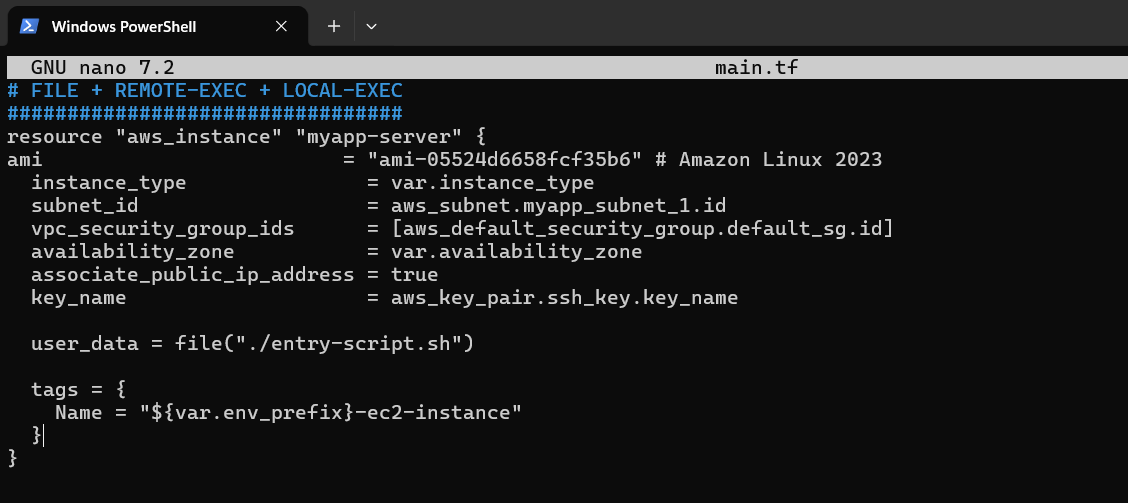
Save screenshot as: task3\_terraform\_destroy.png — terraform destroy output.

1. Remove the provisioners and restore user\_data:

Replace the connection and provisioner blocks with:

user\_data = file("./entry-script.sh")

Save screenshot as: task3\_main\_tf\_restored.png — main.tf showing user\_data restored.



Task 4 — Create Terraform modules (subnet module)

1. Create the module directory structure:

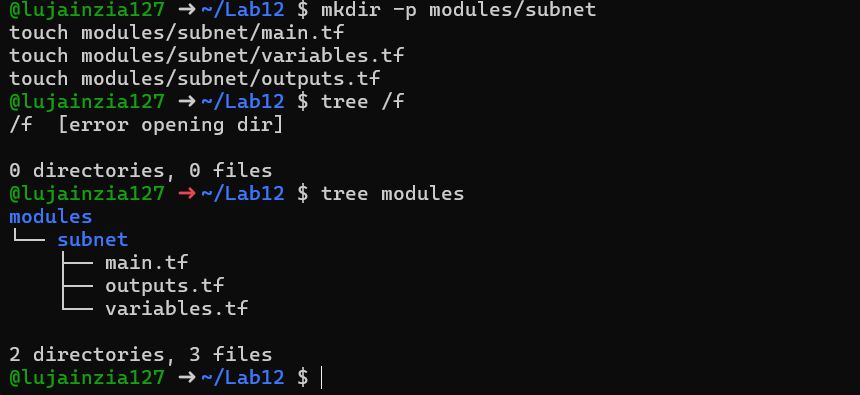
mkdir -p modules/subnet

touch modules/subnet/main.tf

touch modules/subnet/variables.tf

touch modules/subnet/outputs. tf

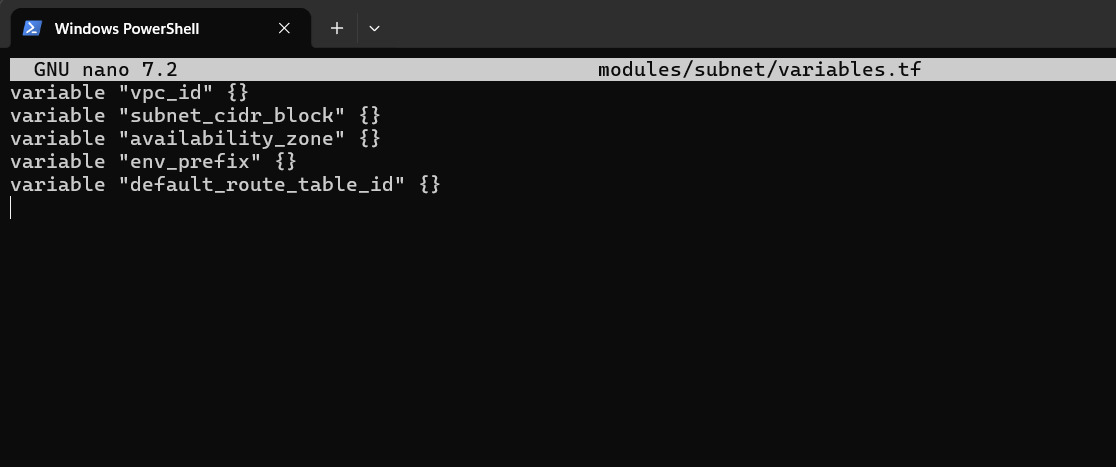
Save screenshot as: task4\_module\_structure.png — terminal showing module directory structure (use tree or ls -R).



1. Create modules/subnet/variables.tf:

variable "vpc\_id" {}variable "subnet\_cidr\_block" {}variable "availability\_zone" {}variable "env\_prefix" {}variable "default\_route\_table\_id" {}

Save screenshot as: task4\_subnet\_variables.png — content of modules/subnet/variables.tf.



1. Create modules/subnet/main.tf:

resource "aws\_subnet" "myapp\_subnet\_1" {

vpc\_id = var.vpc\_id

cidr\_block = var.subnet\_cidr\_block

availability\_zone = var.availability\_zone

map\_public\_ip\_on\_launch = true

tags = {

Name = "${var.env\_prefix}-subnet-1"

}

}

resource "aws\_default\_route\_table" "main\_rt" {

default\_route\_table\_id = var.default\_route\_table\_id

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway. myapp\_igw.id

}

tags = {

Name = "${var.env\_prefix}-rt"

}

}

resource "aws\_internet\_gateway" "myapp\_igw" {

vpc\_id = var.vpc\_id

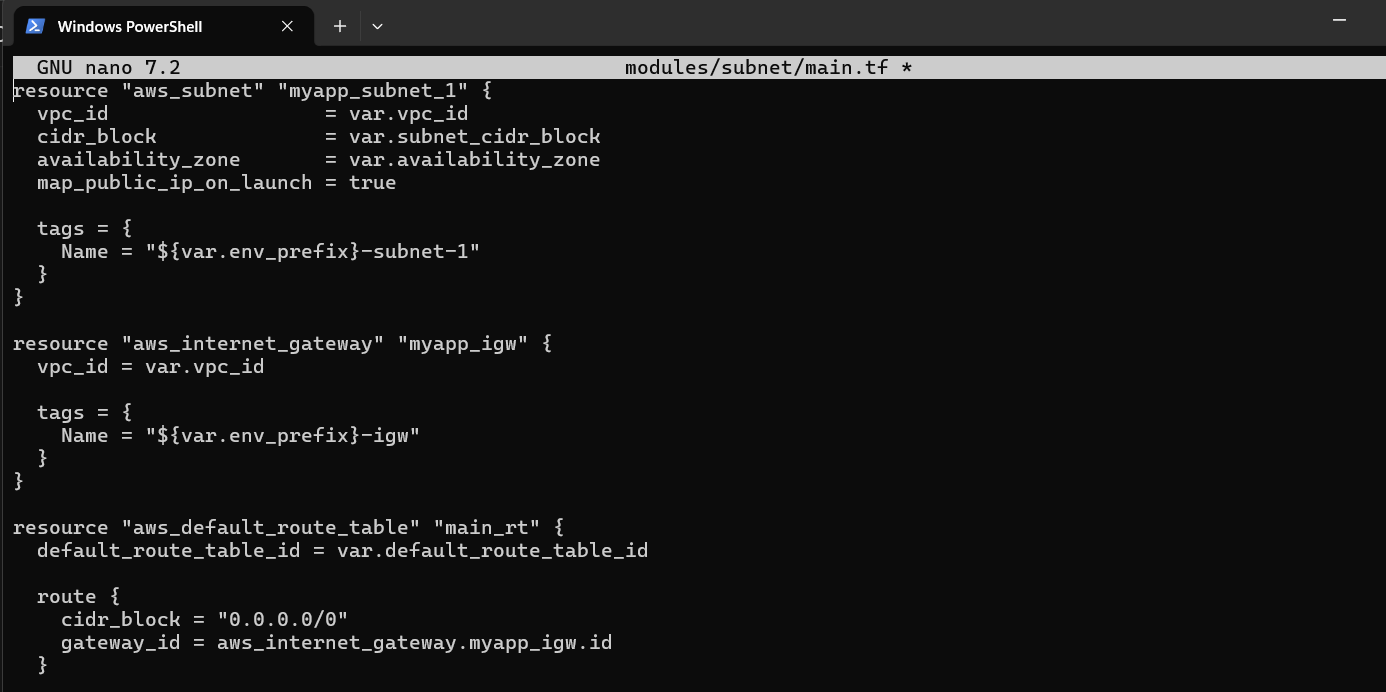
tags = {

Name = "${var.env\_prefix}-igw"

}

}

Save screenshot as: task4\_subnet\_main.png — content of modules/subnet/main.tf.



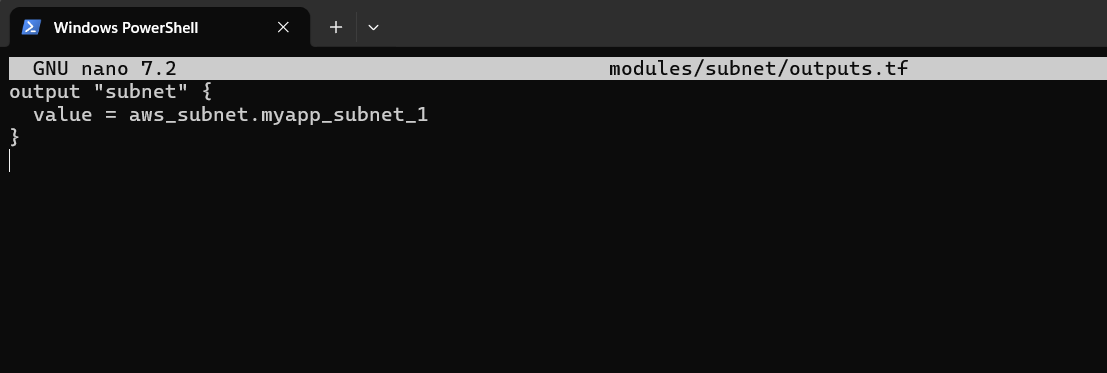
1. Create modules/subnet/outputs.tf:

output "subnet" {

value = aws\_subnet.myapp\_subnet\_1

}

Save screenshot as: task4\_subnet\_outputs.png — content of modules/subnet/outputs.tf.



1. Modify the root main.tf to use the subnet module:

Remove the subnet, route table, and internet gateway resources and replace them with:

module "myapp-subnet" {

source = "./modules/subnet"

vpc\_id = aws\_vpc.myapp\_vpc. id

subnet\_cidr\_block = var.subnet\_cidr\_block

availability\_zone = var.availability\_zone

env\_prefix = var.env\_prefix

default\_route\_table\_id = aws\_vpc.myapp\_vpc.default\_route\_table\_id

}

And update the instance resource to reference the module output:

resource "aws\_instance" "myapp-server" {

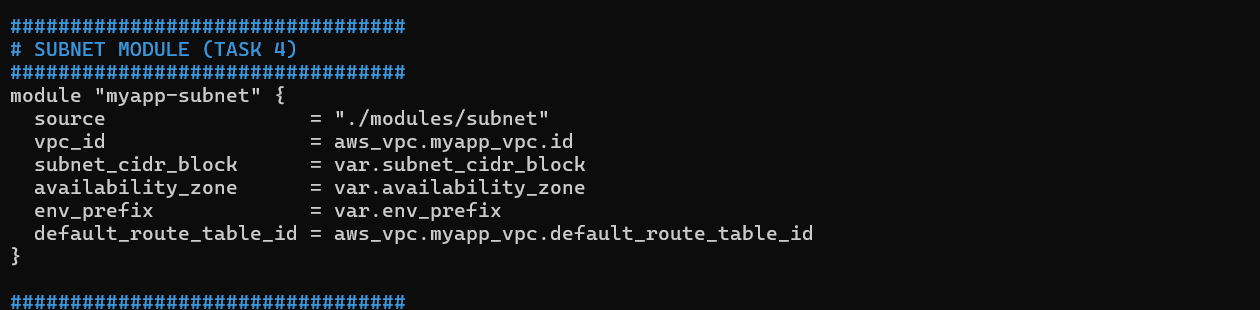
# ... other settings ...

subnet\_id = module.myapp-subnet. subnet.id

# ... rest of configuration ...

}

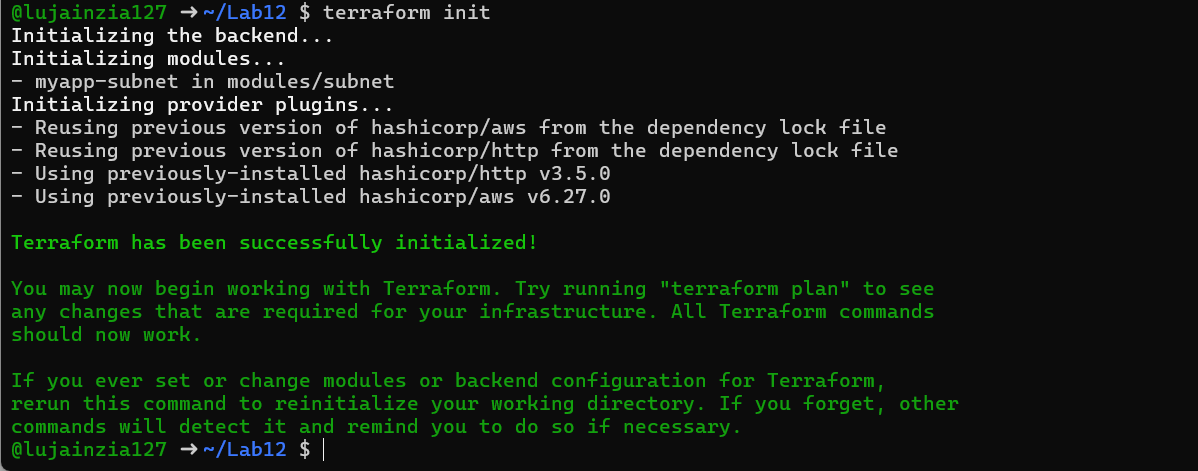
Save screenshot as: task4\_main\_tf\_with\_module.png — main.tf showing module usage.



1. Initialize Terraform to download the module:

terraform init

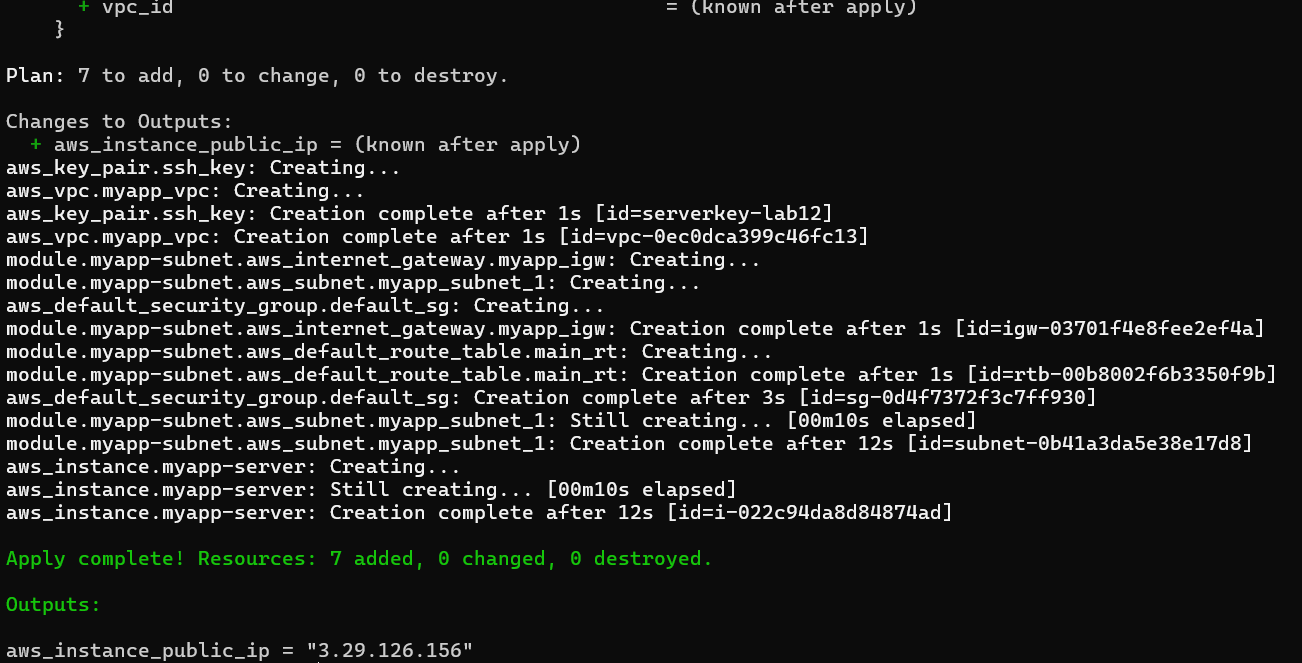
Save screenshot as: task4\_terraform\_init.png — terraform init output showing module initialization.



1. Apply the configuration:

terraform apply -auto-approve

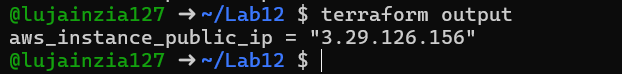
Save screenshot as: task4\_terraform\_apply.png — terraform apply output with module.



1. Display the output:

terraform output

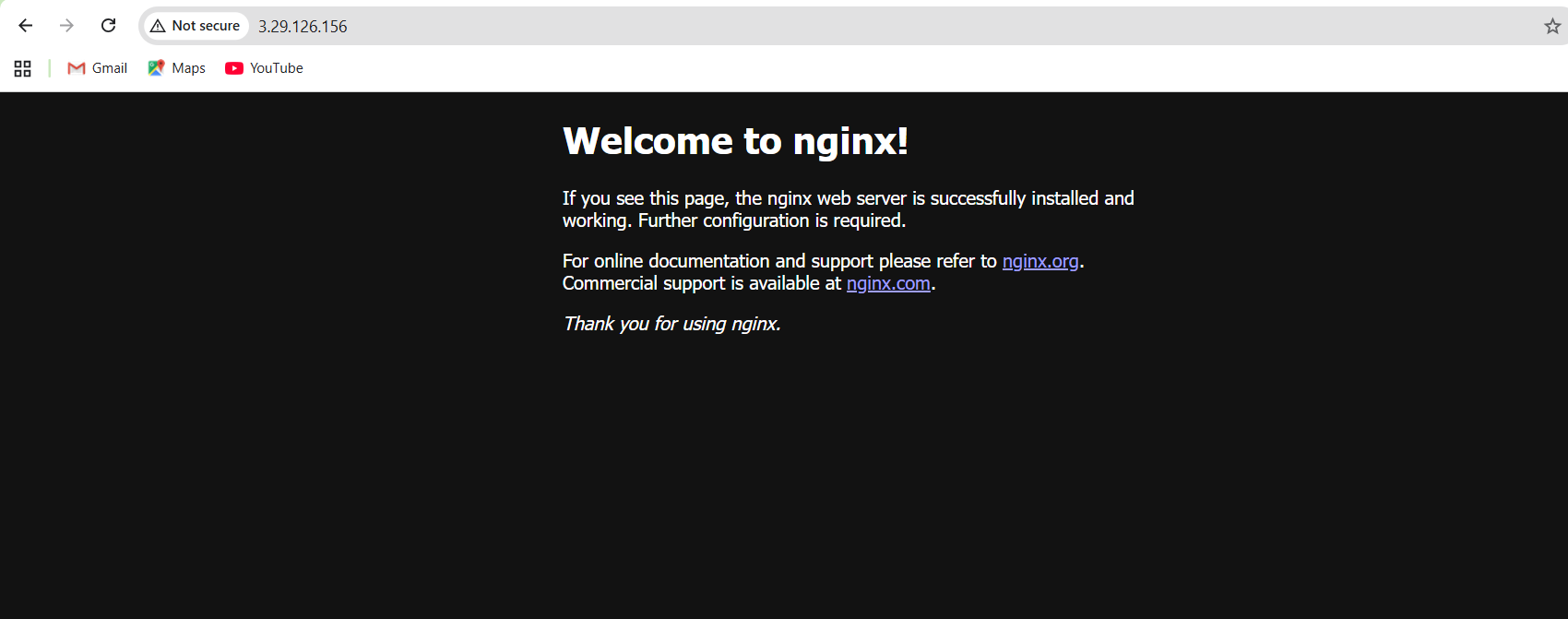
Save screenshot as: task4\_terraform\_output.png — terraform output showing public IP.



1. Test nginx in browser:

Open browser and navigate to http://<public-ip>

Save screenshot as: task4\_nginx\_browser.png — browser showing nginx default page.



Task 5 — Create webserver module

1. Create the webserver module directory structure:

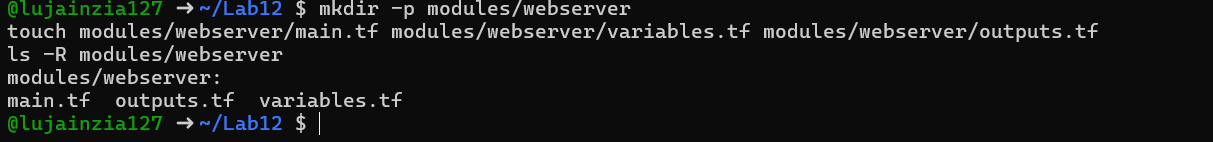
mkdir -p modules/webserver

touch modules/webserver/main.tf

touch modules/webserver/variables.tf

touch modules/webserver/outputs.tf

Save screenshot as: task5\_webserver\_module\_structure.png — terminal showing webserver module directory.



1. Create modules/webserver/variables.tf:

variable "env\_prefix" {}variable "instance\_type" {}variable "availability\_zone" {}variable "public\_key" {}variable "my\_ip" {}variable "vpc\_id" {}variable "subnet\_id" {}variable "script\_path" {}variable "instance\_suffix" {}

Save screenshot as: task5\_webserver\_variables.png — content of modules/webserver/variables. tf.



1. Create modules/webserver/main.tf:

resource "aws\_security\_group" "web\_sg" {

vpc\_id = var.vpc\_id

name = "${var.env\_prefix}-web-sg-${var.instance\_suffix}"

description = "Security group for web server allowing HTTP, HTTPS and SSH"

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = [var.my\_ip]

}

ingress {

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

prefix\_list\_ids = []

}

tags = {

Name = "${var.env\_prefix}-default-sg"

}

}

resource "aws\_key\_pair" "ssh-key" {

key\_name = "${var.env\_prefix}-serverkey-${var.instance\_suffix}"

public\_key = file(var.public\_key)

}

resource "aws\_instance" "myapp-server" {

ami = "ami-05524d6658fcf35b6" # Amazon Linux 2023 Kernel 6.1 AMI

instance\_type = var. instance\_type

subnet\_id = var.subnet\_id

security\_groups = [aws\_security\_group.web\_sg.id]

availability\_zone = var. availability\_zone

associate\_public\_ip\_address = true

key\_name = aws\_key\_pair.ssh-key.key\_name

user\_data = file(var.script\_path)

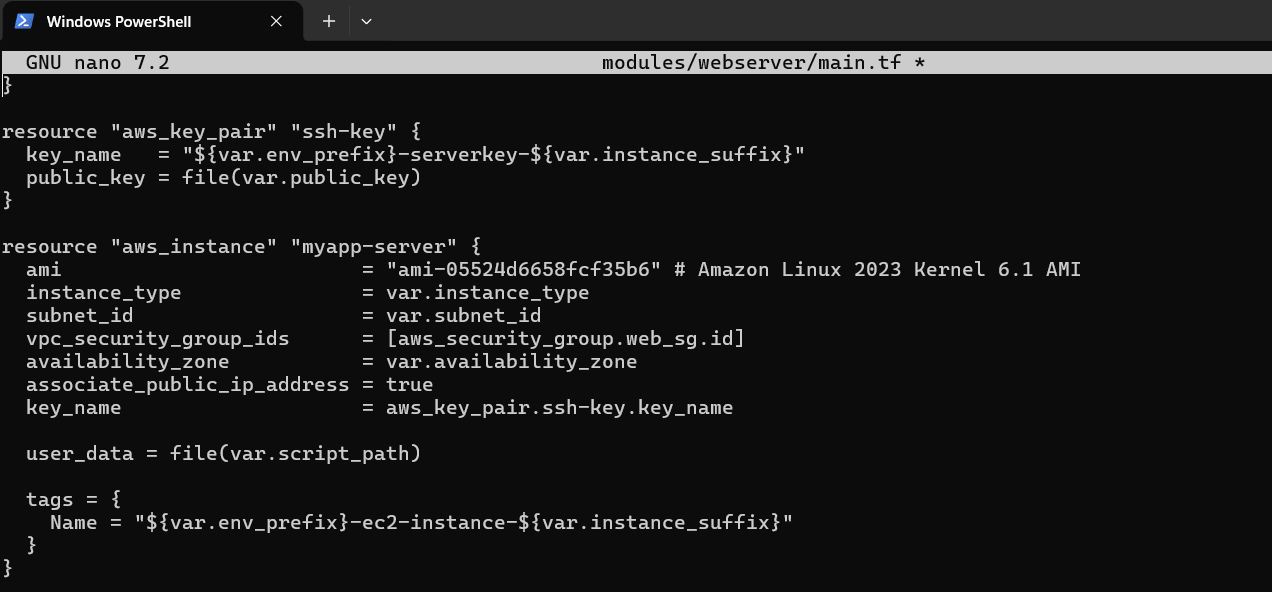
tags = {

Name = "${var.env\_prefix}-ec2-instance-${var.instance\_suffix}"

}

}

Save screenshot as: task5\_webserver\_main.png — content of modules/webserver/main.tf.



1. Create modules/webserver/outputs.tf:

output "aws\_instance" {

value = aws\_instance.myapp-server

}

Save screenshot as: task5\_webserver\_outputs.png — content of modules/webserver/outputs.tf.



1. Modify the root main.tf:

Remove the security group, key pair, and instance resources. Replace them with:

module "myapp-webserver" {

source = "./modules/webserver"

env\_prefix = var.env\_prefix

instance\_type = var. instance\_type

availability\_zone = var.availability\_zone

public\_key = var.public\_key

my\_ip = local.my\_ip

vpc\_id = aws\_vpc.myapp\_vpc.id

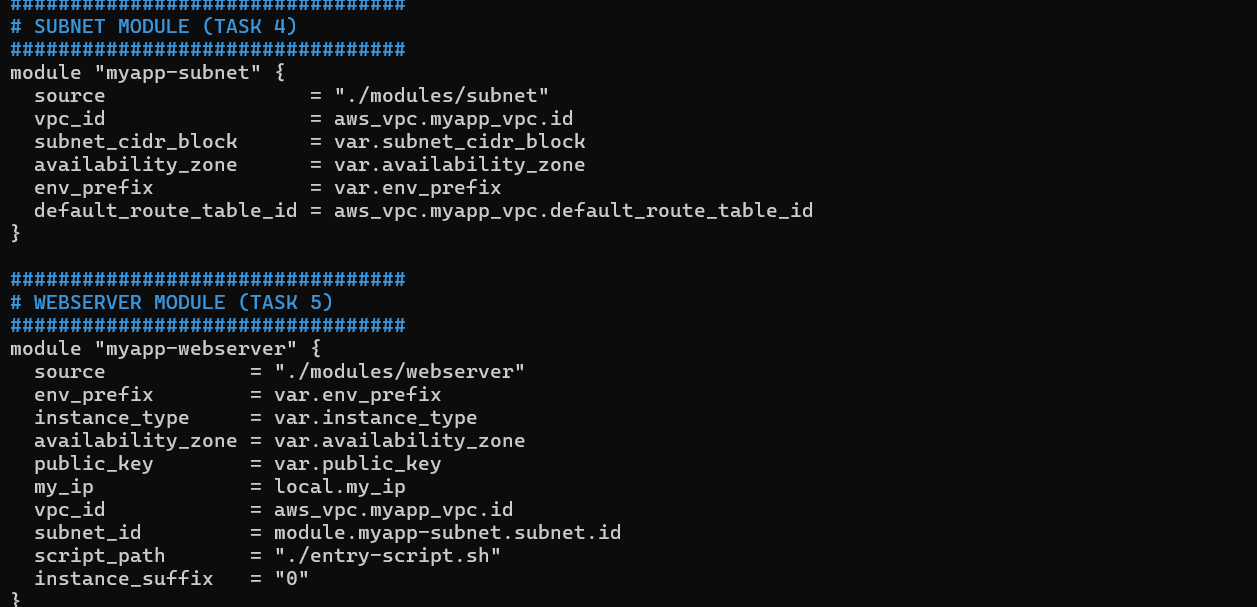
subnet\_id = module.myapp-subnet.subnet.id

script\_path = "./entry-script.sh"

instance\_suffix = "0"

}

Save screenshot as: task5\_main\_tf\_webserver\_module.png — main.tf showing webserver module usage.



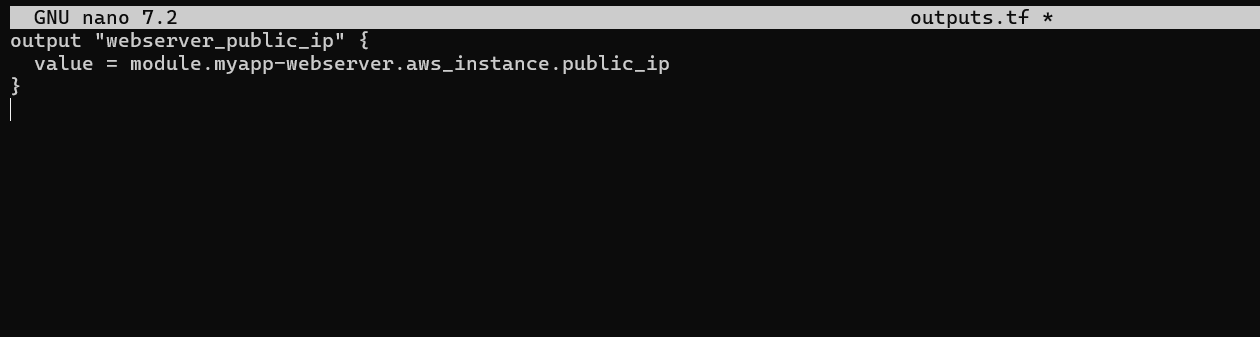
1. Update outputs.tf:

output "webserver\_public\_ip" {

value = module.myapp-webserver.aws\_instance.public\_ip

}

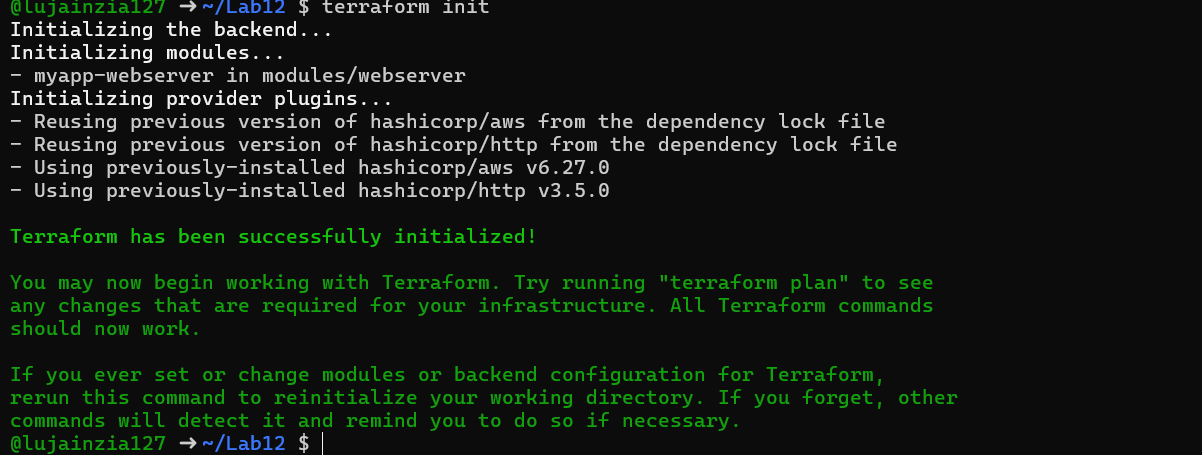
Save screenshot as: task5\_outputs\_updated.png — updated outputs.tf.



1. Initialize Terraform:

terraform init

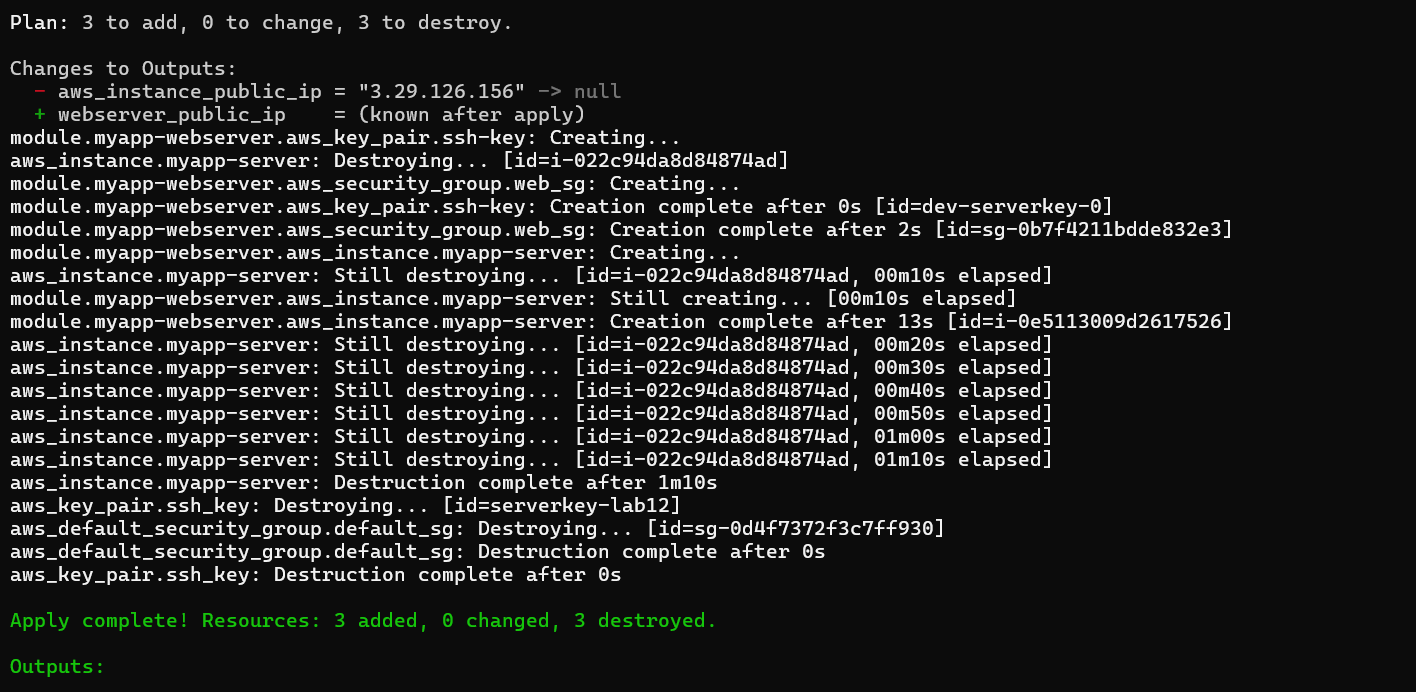
Save screenshot as: task5\_terraform\_init.png — terraform init output.



1. Apply the configuration:

terraform apply -auto-approve

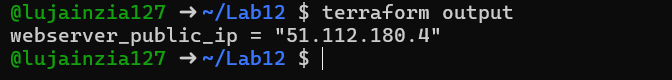
Save screenshot as: task5\_terraform\_apply.png — terraform apply output with webserver module.



1. Display the output:

terraform output

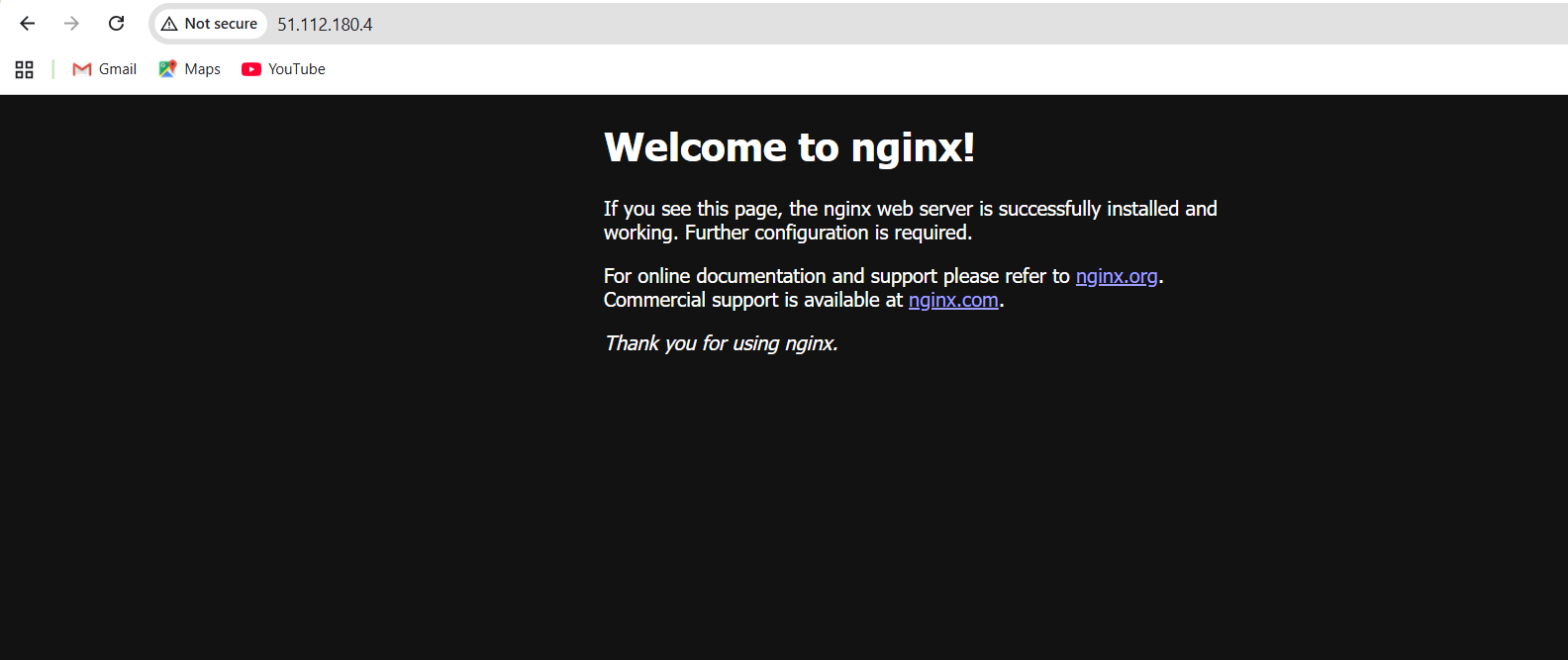
Save screenshot as: task5\_terraform\_output.png — terraform output showing webserver public IP.



1. Test nginx in browser:

Open browser and navigate to http://<public-ip>

Save screenshot as: task5\_nginx\_browser.png — browser showing nginx default page.

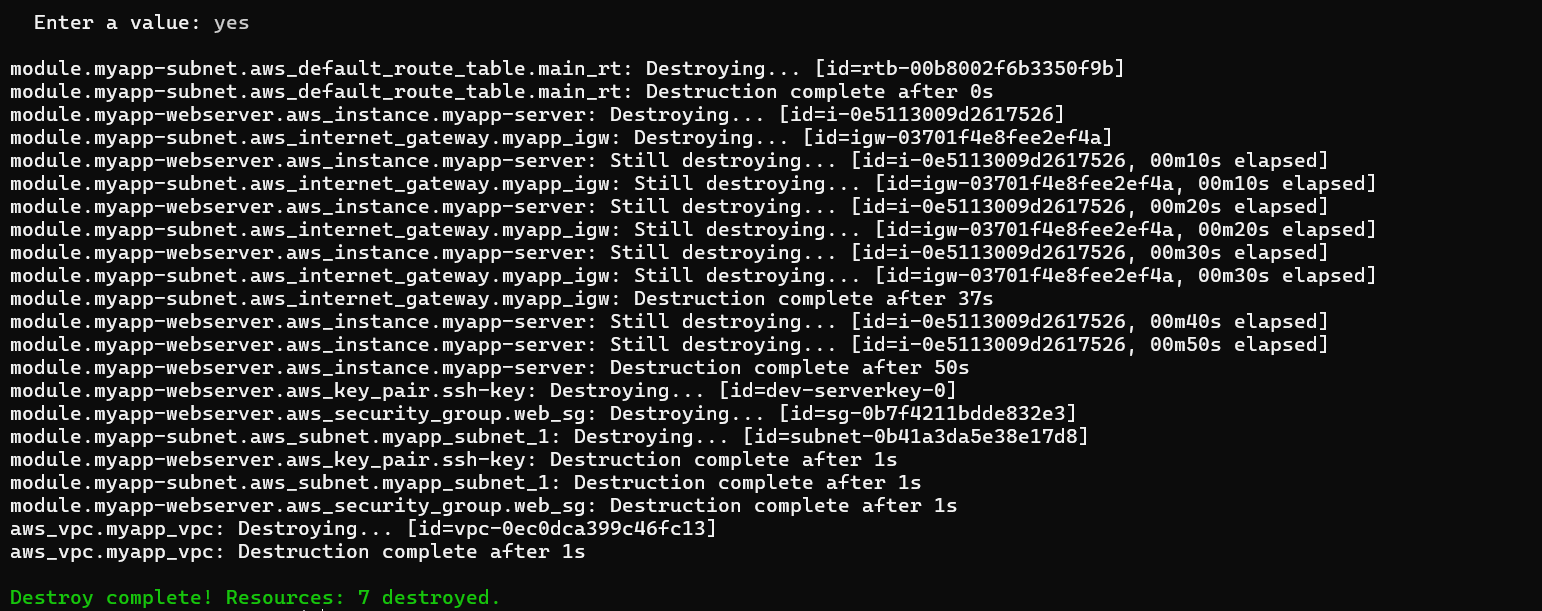


1. Destroy resources:

terraform destroy

Type yes when prompted.

Save screenshot as: task5\_terraform\_destroy.png — terraform destroy output.



Task 6 — Configure HTTPS with self-signed certificates

1. Update entry-script.sh with SSL configuration:

#!/bin/bashset -e

yum update -y

yum install -y nginx

systemctl start nginx

systemctl enable nginx

# Create directories for SSL certificates if they don't exist

mkdir -p /etc/ssl/private

mkdir -p /etc/ssl/certs

# Get IMDSv2 token

TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" \ -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")

# Get current public IP

PUBLIC\_IP=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \ http://169.254.169.254/latest/meta-data/public-ipv4)

PUBLIC\_HOSTNAME=$(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" \ http://169.254.169.254/latest/meta-data/public-hostname)

# Generate self-signed certificate with dynamic IP

openssl req -x509 -nodes -days 365 -newkey rsa:2048 \

-keyout /etc/ssl/private/selfsigned.key \

-out /etc/ssl/certs/selfsigned.crt \

-subj "/CN=$PUBLIC\_IP" \

-addext "subjectAltName=IP:$PUBLIC\_IP" \

-addext "basicConstraints=CA:FALSE" \

-addext "keyUsage=digitalSignature,keyEncipherment" \

-addext "extendedKeyUsage=serverAuth"

echo "Self-signed certificate created for IP: $PUBLIC\_IP"

# Backup existing nginx. conf

cp /etc/nginx/nginx.conf /etc/nginx/nginx.conf.bak

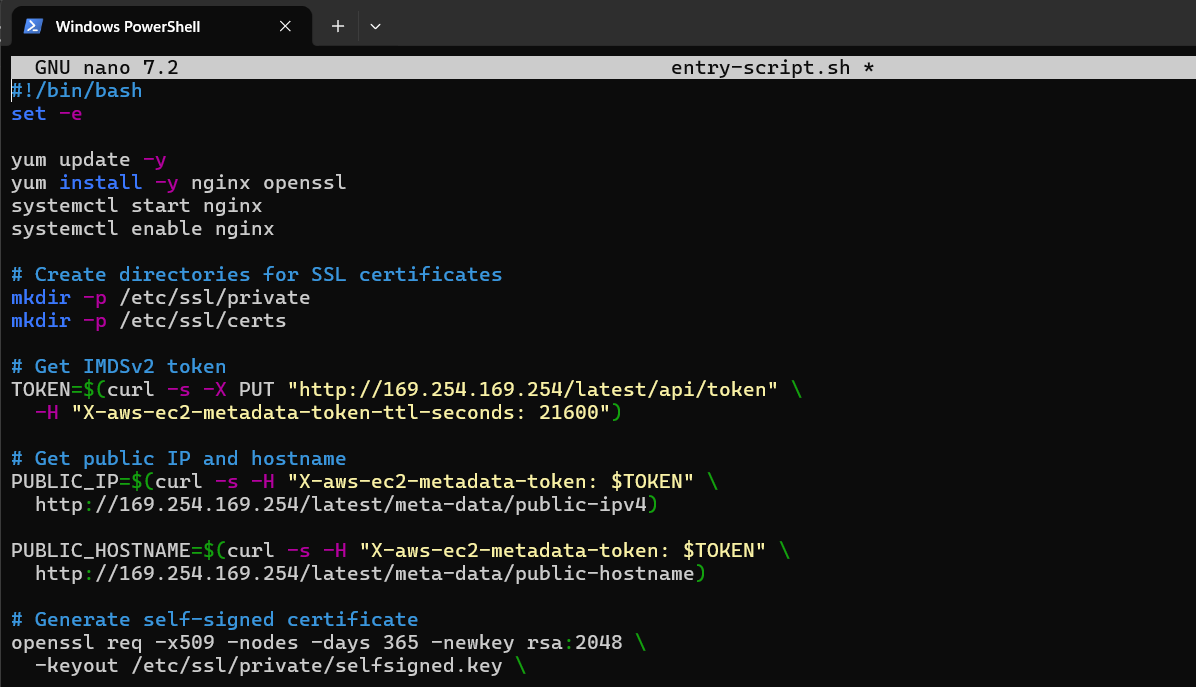
# Overwrite nginx.conf with the desired content

cat <<EOF > /etc/nginx/nginx.confuser nginx;worker\_processes auto;error\_log /var/log/nginx/error.log notice;pid /run/nginx. pid;events { worker\_connections 1024;}http { log\_format main '\$remote\_addr - \$remote\_user [\$time\_local] "\$request" ' '\$status \$body\_bytes\_sent "\$http\_referer" ' '"\$http\_user\_agent" "\$http\_x\_forwarded\_for"'; access\_log /var/log/nginx/access.log main; sendfile on; tcp\_nopush on; keepalive\_timeout 65; types\_hash\_max\_size 4096; include /etc/nginx/mime.types; default\_type application/octet-stream; upstream backend\_servers { server 158.252.94.241:80; server 158.252.94.242:80 backup; } server { listen 443 ssl; server\_name $PUBLIC\_IP; ssl\_certificate /etc/ssl/certs/selfsigned. crt; ssl\_certificate\_key /etc/ssl/private/selfsigned.key; location / { root /usr/share/nginx/html; index index.html; # proxy\_pass http://158.252.94.241:80; # proxy\_pass http://backend\_servers; } } server { listen 80; server\_name \_; return 301 https://\$host\$request\_uri; }}EOF

# Test and restart Nginx

systemctl restart nginx

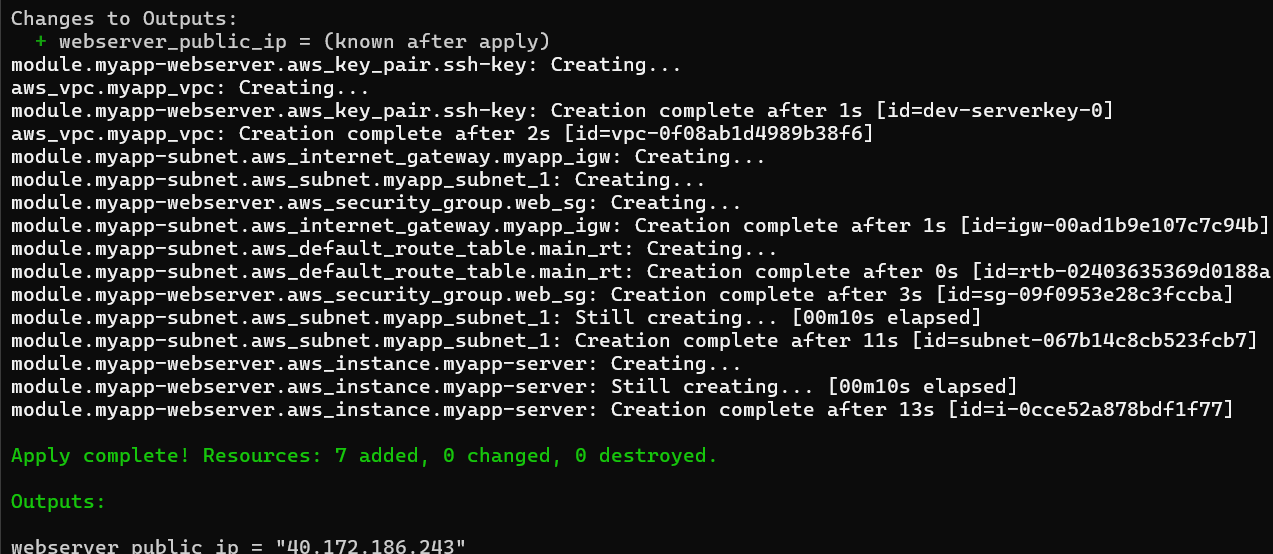
Save screenshot as: task6\_entry\_script\_https.png — updated entry-script.sh with HTTPS configuration.



1. Apply the configuration:

terraform apply -auto-approve

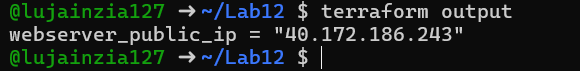
Save screenshot as: task6\_terraform\_apply.png — terraform apply output.



1. Display the output:

terraform output

Save screenshot as: task6\_terraform\_output.png — terraform output showing public IP.



1. Test HTTPS in browser:

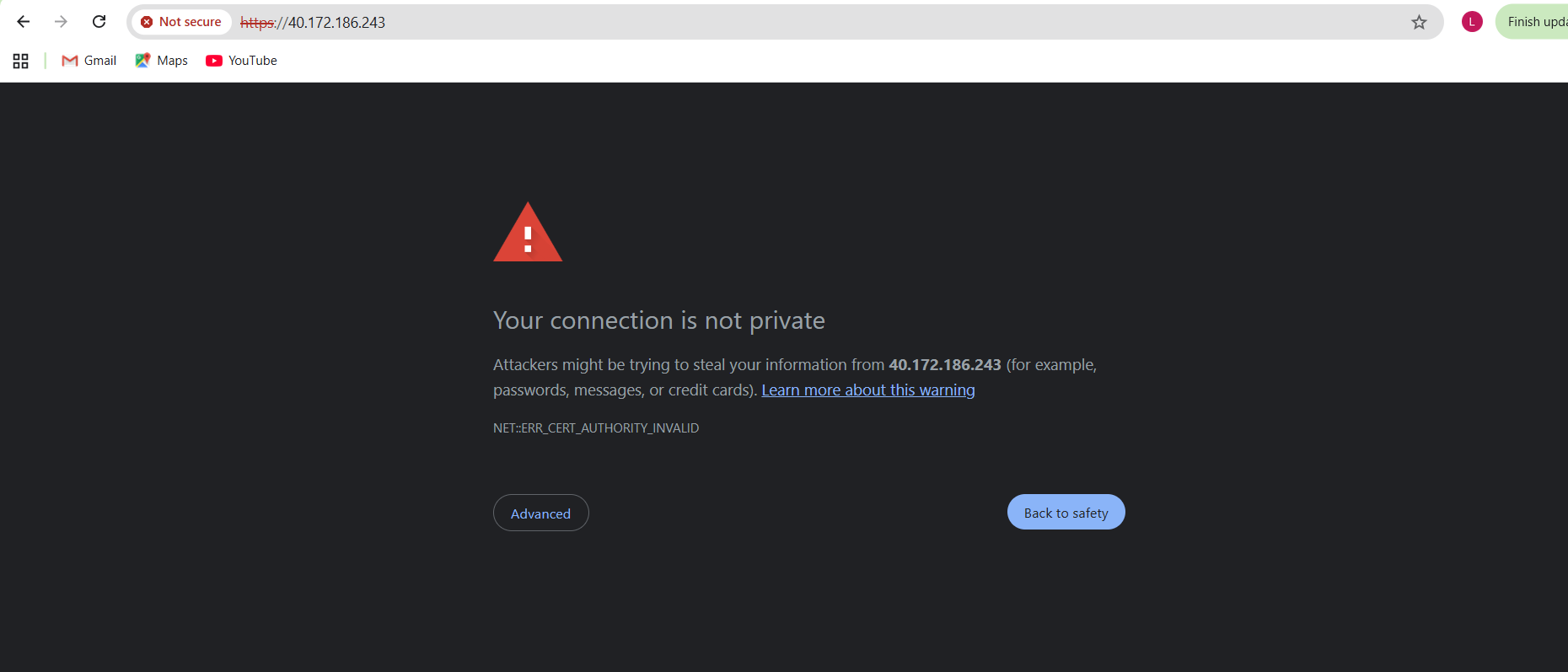
Open browser and navigate to https://<public-ip>

You will see a warning: "Warning: Potential Security Risk Ahead"

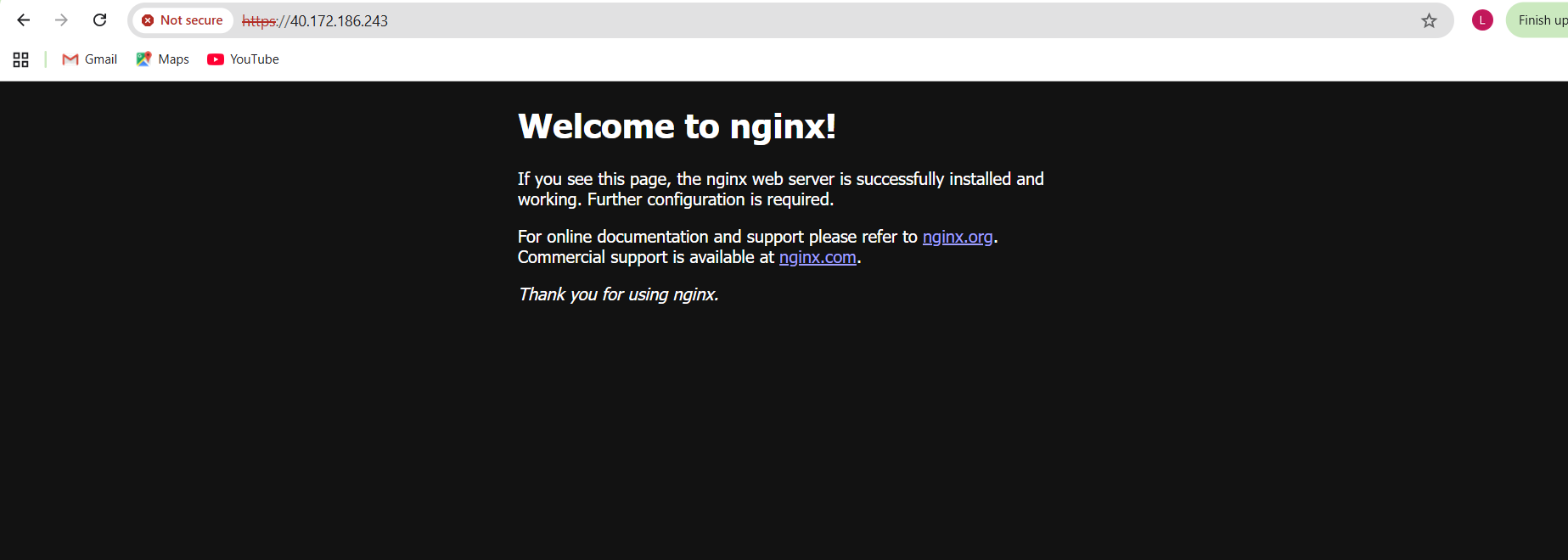
Click "Advanced" button

Click "Accept the Risk and Continue"

Save screenshot as: task6\_browser\_security\_warning.png — browser showing security warning.



Save screenshot as: task6\_nginx\_https\_browser.png — browser showing nginx page over HTTPS after accepting risk.

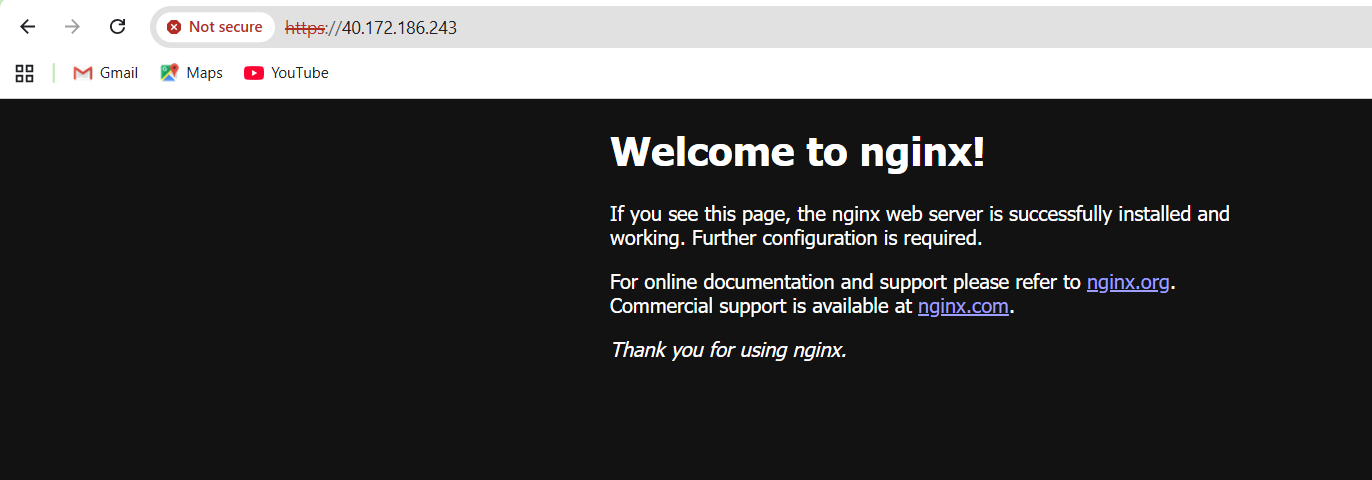
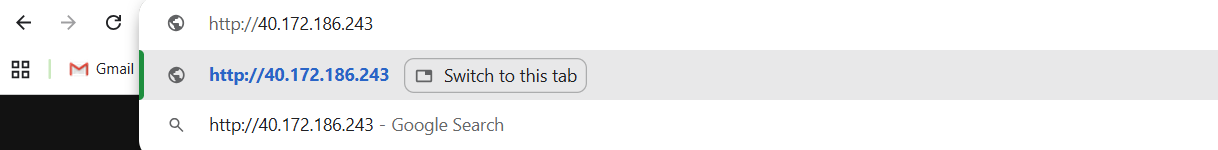


1. Verify HTTP to HTTPS redirect:

Open browser and navigate to http://<public-ip>

Verify it redirects to https://<public-ip>

Save screenshot as: task6\_http\_redirect.png — browser showing redirect from HTTP to HTTPS.



Task 7 — Configure Nginx as reverse proxy

1. Create apache.sh script for backend web server:

#!/bin/bash

yum update -y

yum install httpd -y

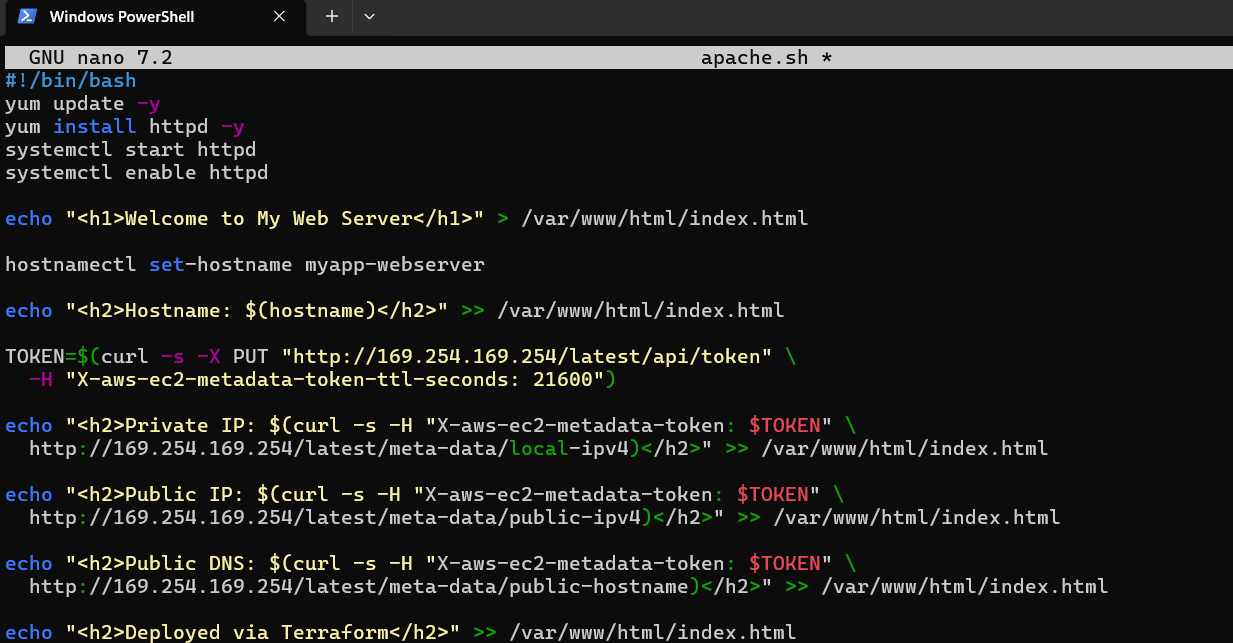
systemctl start httpd

systemctl enable httpdecho "<h1>Welcome to My Web Server</h1>" > /var/www/html/index.html

hostnamectl set-hostname myapp-webserverecho "<h2>Hostname: $(hostname)</h2>" >> /var/www/html/index.html

TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" \ -H "X-aws-ec2-metadata-token-ttl-seconds: 21600") echo "<h2>Private IP: $(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/local-ipv4)</h2>" >> /var/www/html/index.htmlecho "<h2>Public IP: $(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/public-ipv4)</h2>" >> /var/www/html/index.htmlecho "<h2>Public DNS: $(curl -s -H "X-aws-ec2-metadata-token: $TOKEN" http://169.254.169.254/latest/meta-data/public-hostname)</h2>" >> /var/www/html/index.htmlecho "<h2>Deployed via Terraform</h2>" >> /var/www/html/index. html

Save screenshot as: task7\_apache\_script.png — content of apache.sh file.



1. Add the backend web server module to main.tf:

module "myapp-web-1" {

source = "./modules/webserver"

env\_prefix = var.env\_prefix

instance\_type = var.instance\_type

availability\_zone = var.availability\_zone

public\_key = var. public\_key

my\_ip = local.my\_ip

vpc\_id = aws\_vpc. myapp\_vpc.id

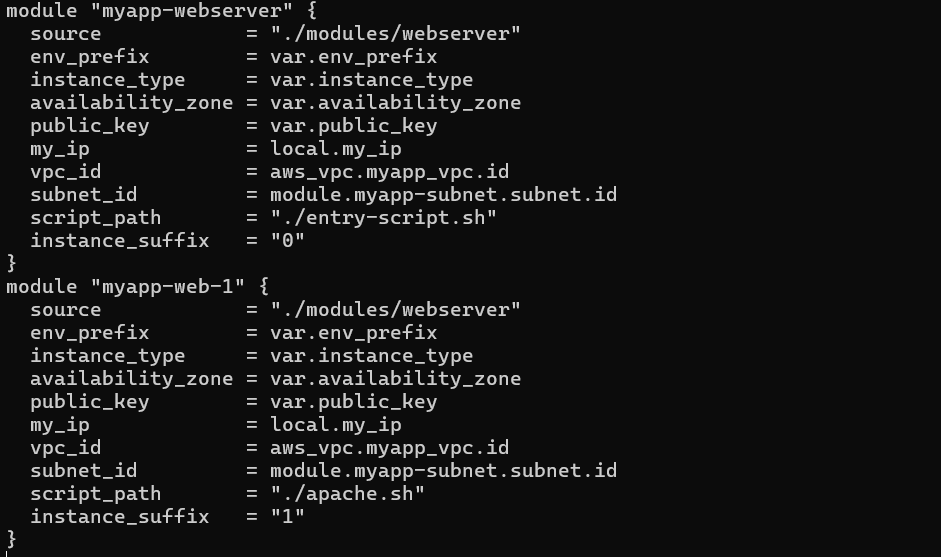
subnet\_id = module.myapp-subnet.subnet. id

script\_path = "./apache.sh"

instance\_suffix = "1"

}

Save screenshot as: task7\_main\_tf\_web1.png — main. tf showing myapp-web-1 module.



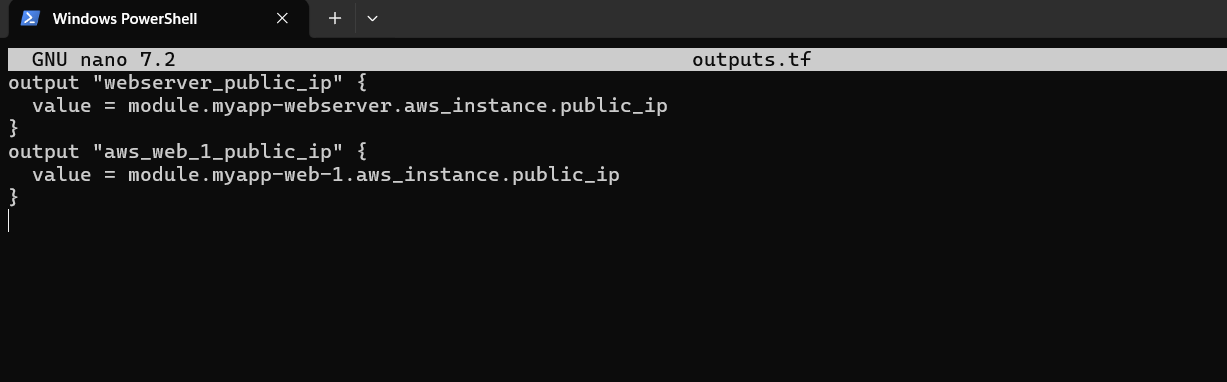
1. Update outputs.tf:

output "aws\_web-1\_public\_ip" {

value = module.myapp-web-1.aws\_instance.public\_ip

}

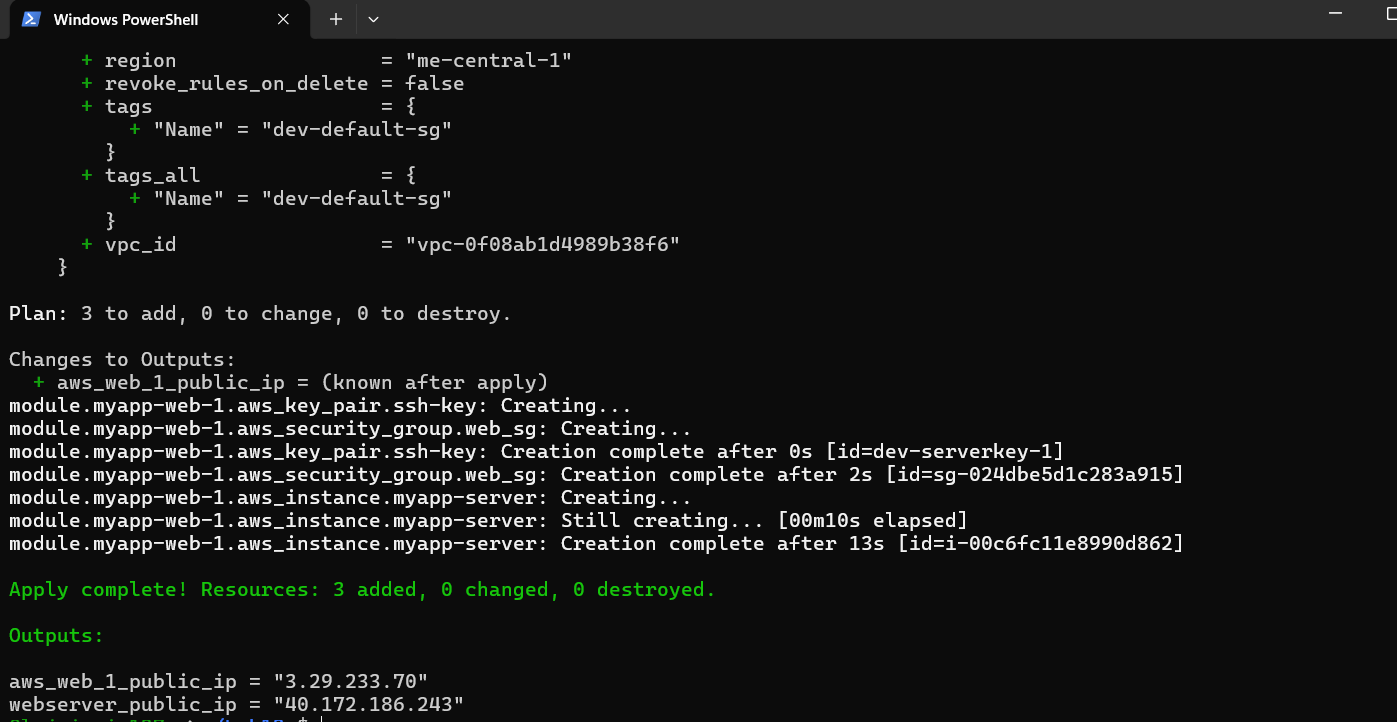
Save screenshot as: task7\_outputs\_web1.png — outputs.tf with web-1 output.



1. Apply the configuration:

terraform apply -auto-approve

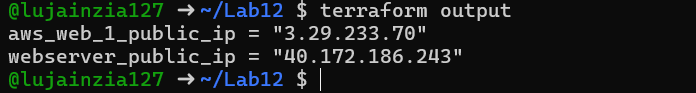
Save screenshot as: task7\_terraform\_apply.png — terraform apply output showing both instances created.



1. Get the outputs:

terraform output

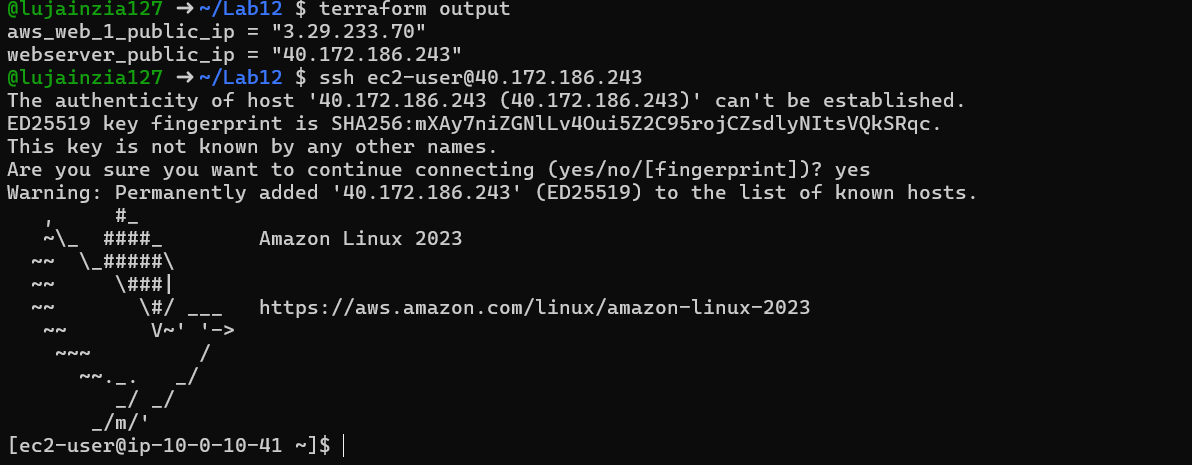
Save screenshot as: task7\_terraform\_output.png — showing both public IPs.



1. SSH into the webserver (Nginx proxy server):

ssh ec2-user@<webserver-public-ip>

Save screenshot as: task7\_ssh\_webserver. png — SSH session to webserver.



1. Edit the Nginx configuration:

sudo vim /etc/nginx/nginx.conf

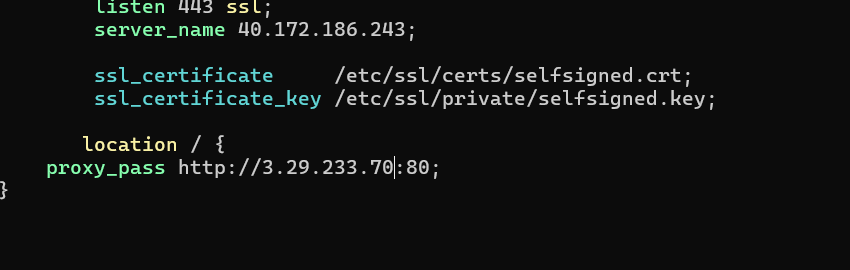
Modify the location block to proxy to web-1:

location / { # root /usr/share/nginx/html; # index index. html;

proxy\_pass http://<web-1-public-ip>:80; # proxy\_pass http://backend\_servers;

}

Save screenshot as: task7\_nginx\_conf\_reverse\_proxy.png — nginx.conf showing proxy\_pass configuration.



1. Restart Nginx:

sudo systemctl restart nginx

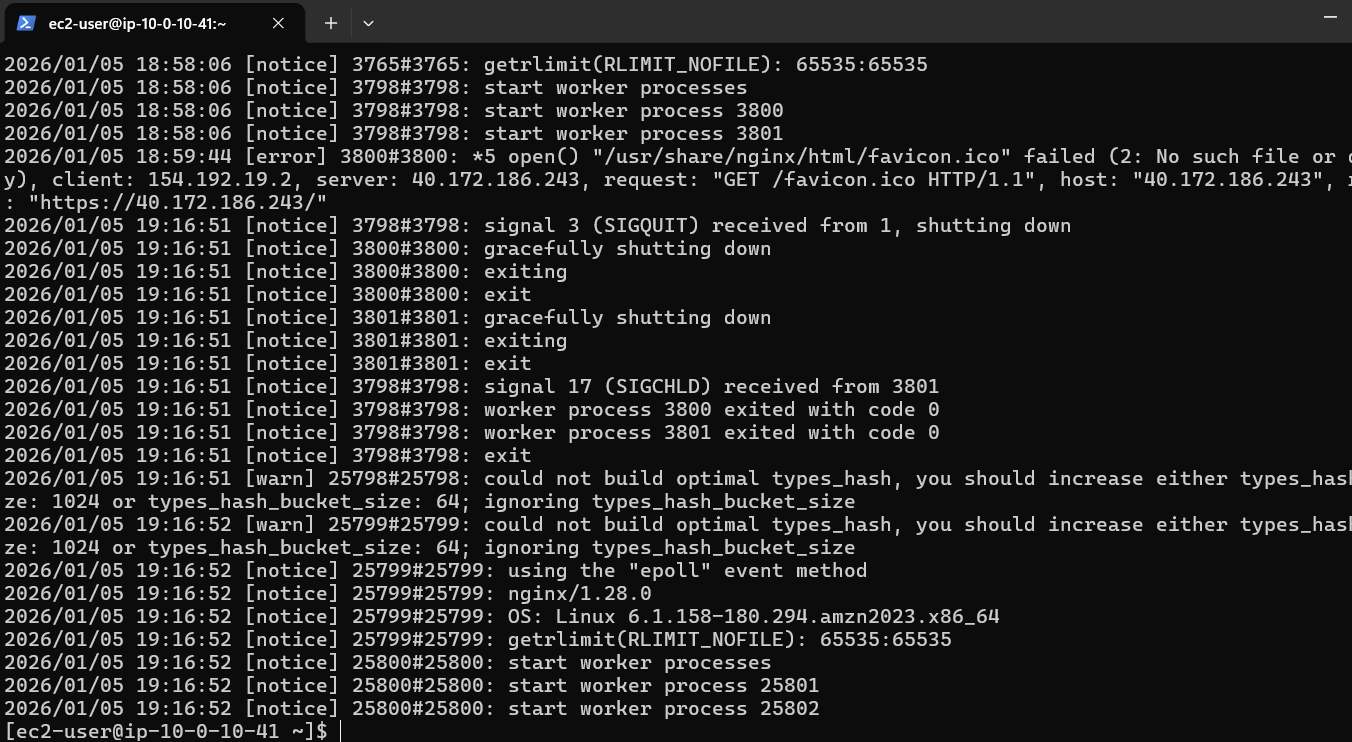
Save screenshot as: task7\_nginx\_restart.png — terminal showing nginx restart command.



1. View Nginx logs and configuration files:

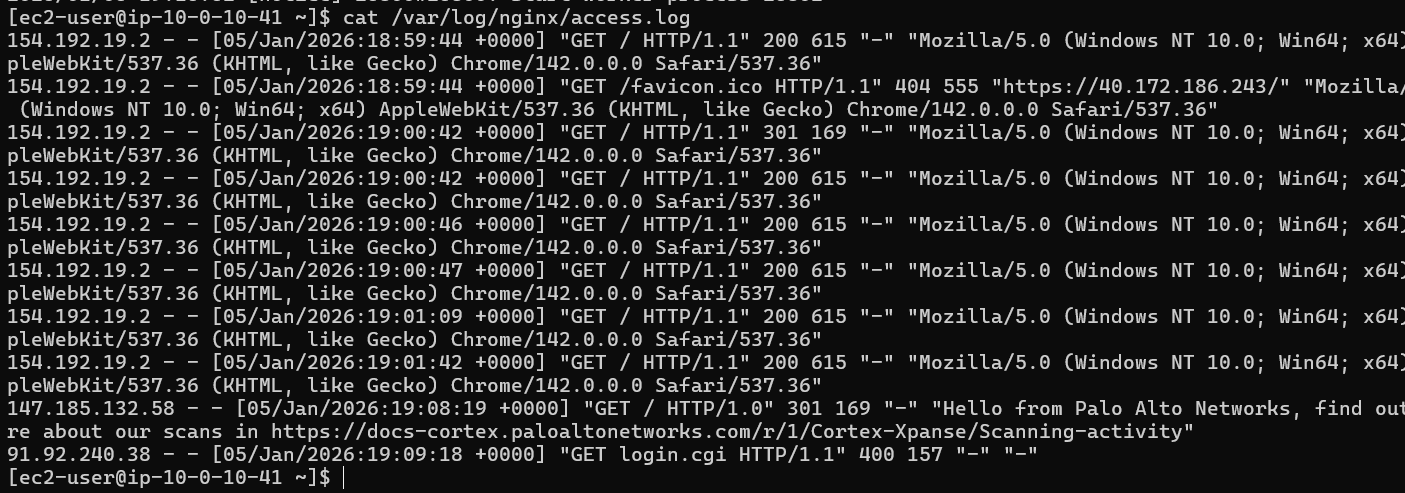
cat /var/log/nginx/error.log

Save screenshot as: task7\_error\_log.png — content of error. log.



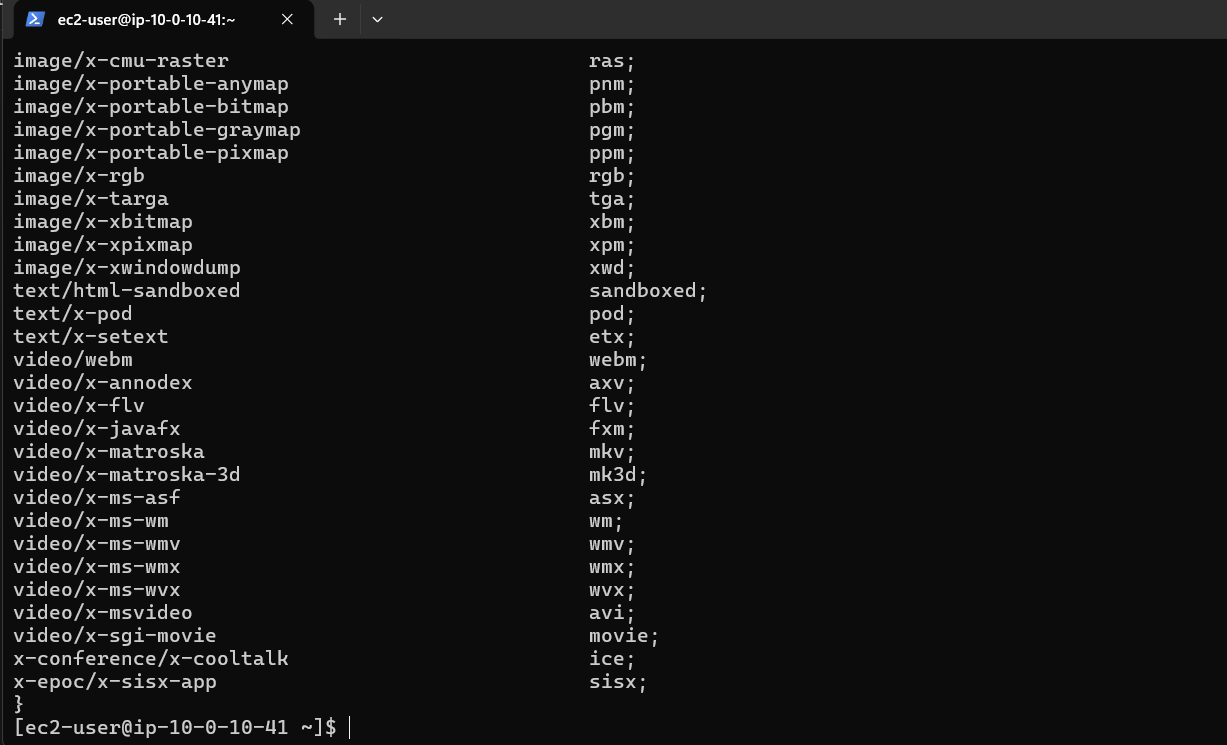
cat /var/log/nginx/access.log

Save screenshot as: task7\_access\_log.png — content of access.log.



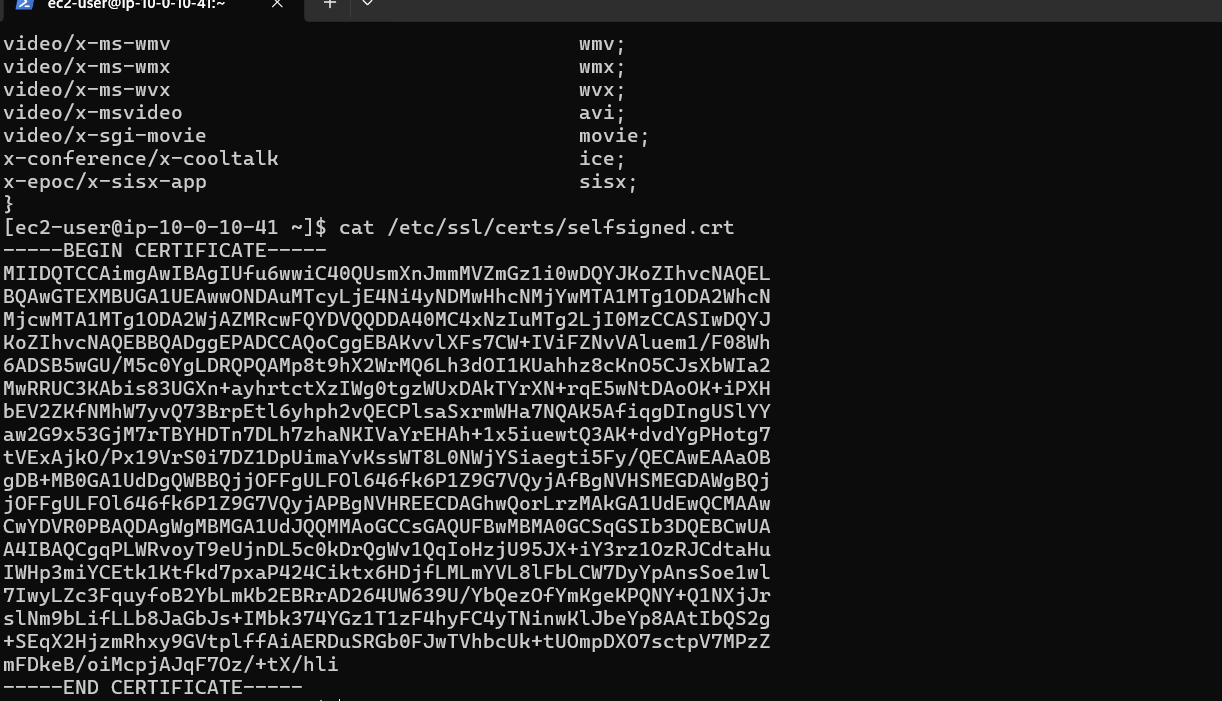
cat /etc/nginx/mime.types

Save screenshot as: task7\_mime\_types.png — content of mime.types.



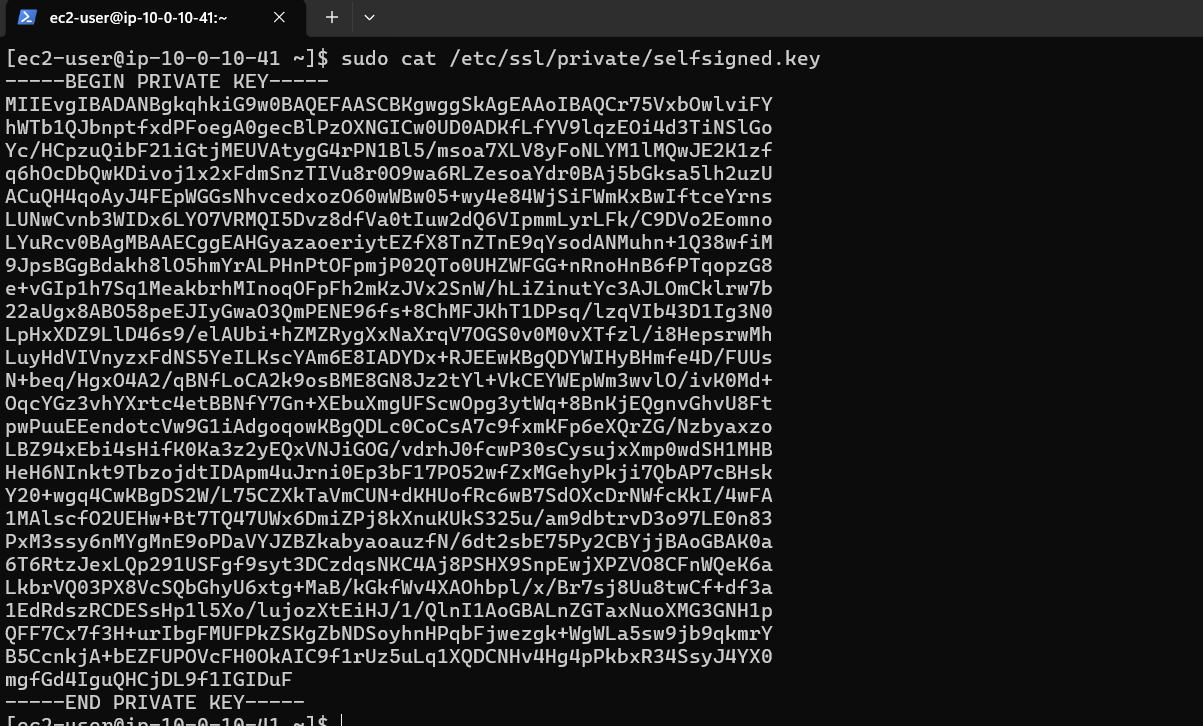
cat /etc/ssl/certs/selfsigned. crt

Save screenshot as: task7\_ssl\_cert.png — content of selfsigned.crt.



cat /etc/ssl/private/selfsigned.key

Save screenshot as: task7\_ssl\_key.png — content of selfsigned.key.

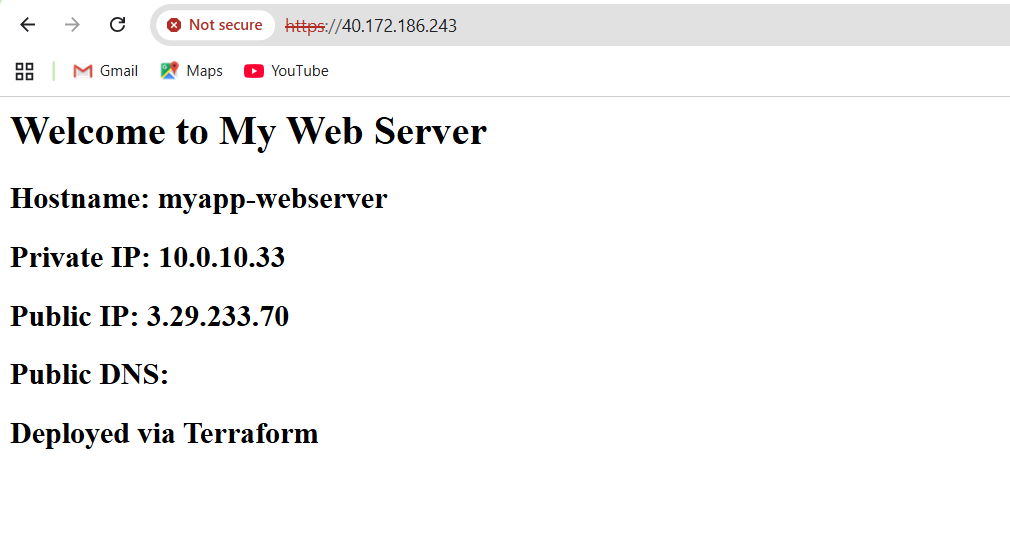


1. Test reverse proxy in browser:

Open browser and navigate to https://<webserver-public-ip>

You should see the web-1 Apache page through the Nginx proxy

Save screenshot as: task7\_reverse\_proxy\_browser.png — browser showing web-1 content through proxy.



Task 8 — Configure Nginx as load balancer

1. Add the second web server module to main.tf:

module "myapp-web-2" {

source = "./modules/webserver"

env\_prefix = var.env\_prefix

instance\_type = var.instance\_type

availability\_zone = var. availability\_zone

public\_key = var.public\_key

my\_ip = local.my\_ip

vpc\_id = aws\_vpc.myapp\_vpc. id

subnet\_id = module.myapp-subnet.subnet. id

script\_path = "./apache.sh"

instance\_suffix = "2"

}

Save screenshot as: task8\_main\_tf\_web2.png — main.tf showing myapp-web-2 module.



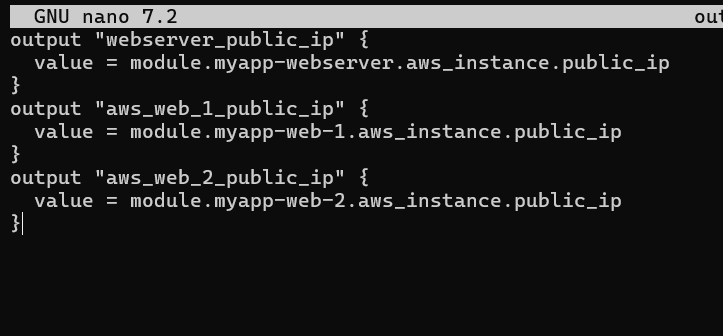
1. Update outputs.tf:

output "aws\_web-2\_public\_ip" {

value = module. myapp-web-2.aws\_instance.public\_ip

}

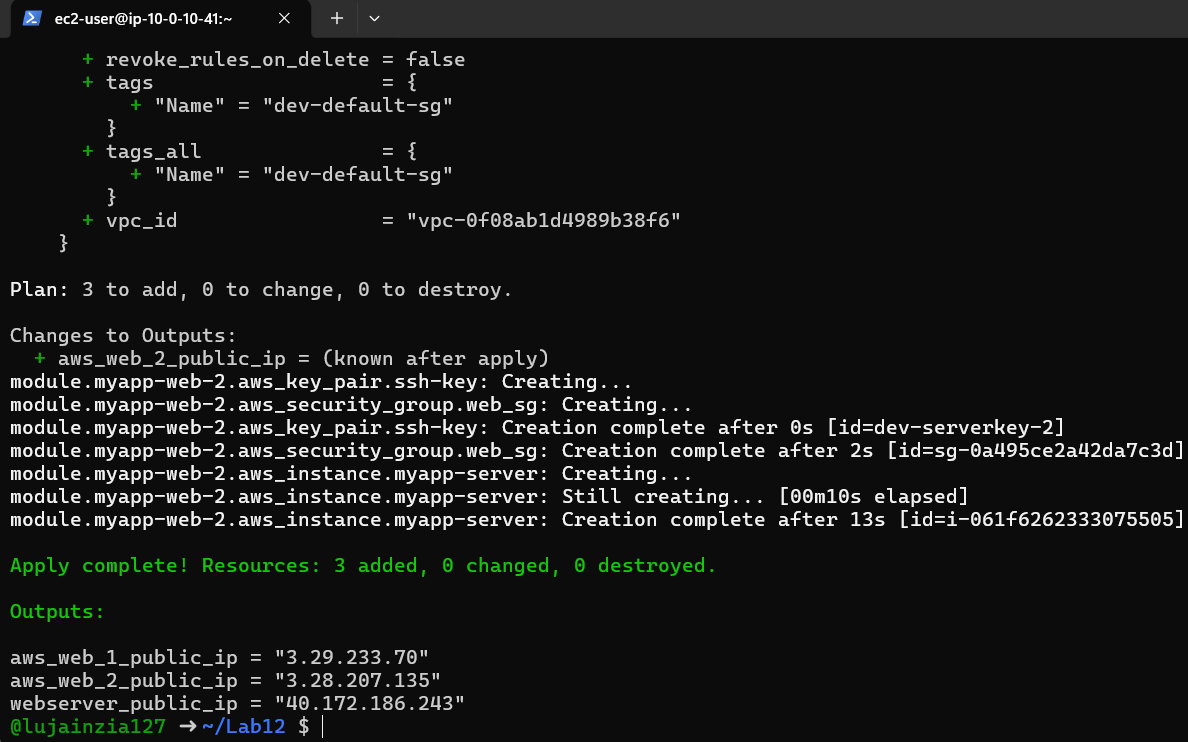
Save screenshot as: task8\_outputs\_web2.png — outputs.tf with web-2 output.



1. Apply the configuration:

terraform apply -auto-approve

Save screenshot as: task8\_terraform\_apply.png — terraform apply output showing web-2 created.



1. Get all outputs:

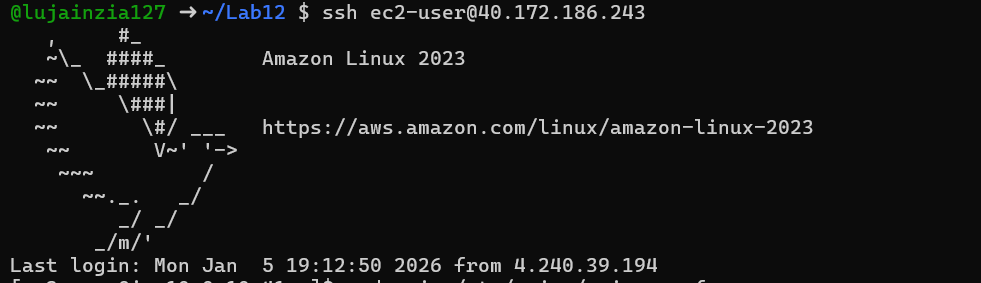
terraform output

Save screenshot as: task8\_terraform\_output.png — showing all three public IPs.



1. SSH into the webserver (Nginx proxy):

ssh ec2-user@<webserver-public-ip>



1. Edit Nginx configuration for load balancing:

sudo vim /etc/nginx/nginx.conf

Update the upstream block and location:

upstream backend\_servers {

server <web-1-public-ip>:80;

server <web-2-public-ip>: 80;

}

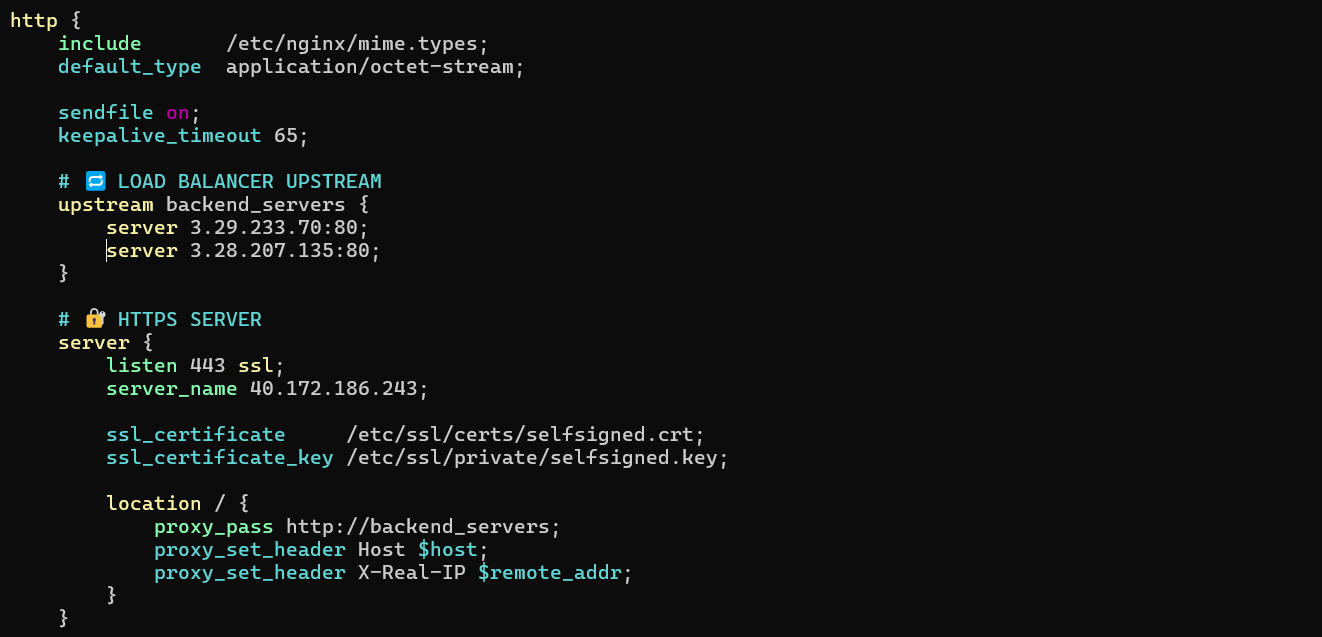
# ... in server block:

location / { # root /usr/share/nginx/html; # index index.html; # proxy\_pass http://<web-1-public-ip>:80;

proxy\_pass http://backend\_servers;

}

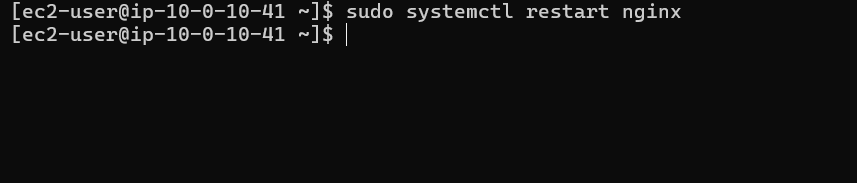
Save screenshot as: task8\_nginx\_conf\_load\_balancer.png — nginx.conf showing load balancing configuration.



1. Restart Nginx:

sudo systemctl restart nginx

Save screenshot as: task8\_nginx\_restart.png — terminal showing nginx restart.



1. Test load balancing in browser:

Open browser and navigate to https://<webserver-public-ip>

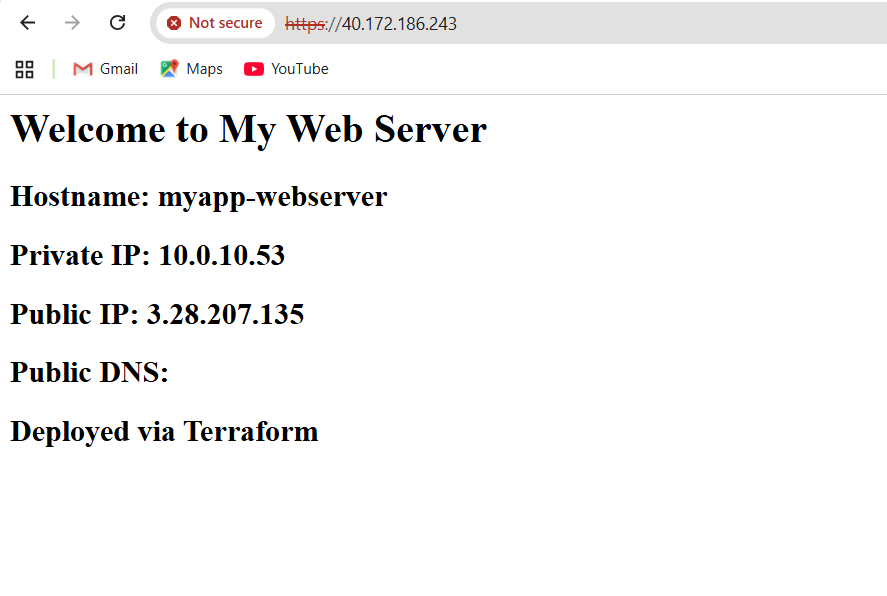
Reload the page multiple times

You should see the content alternating between web-1 and web-2 (check the hostname/IP in the page)

Save screenshot as: task8\_load\_balancer\_web1. png — browser showing web-1 content.



Save screenshot as: task8\_load\_balancer\_web2.png — browser showing web-2 content after reload.



Task 9 — Configure high availability with backup servers

1. SSH into the webserver:

ssh ec2-user@<webserver-public-ip>

Edit Nginx configuration for high availability:

sudo vim /etc/nginx/nginx.conf

Update the upstream block to make web-2 a backup:

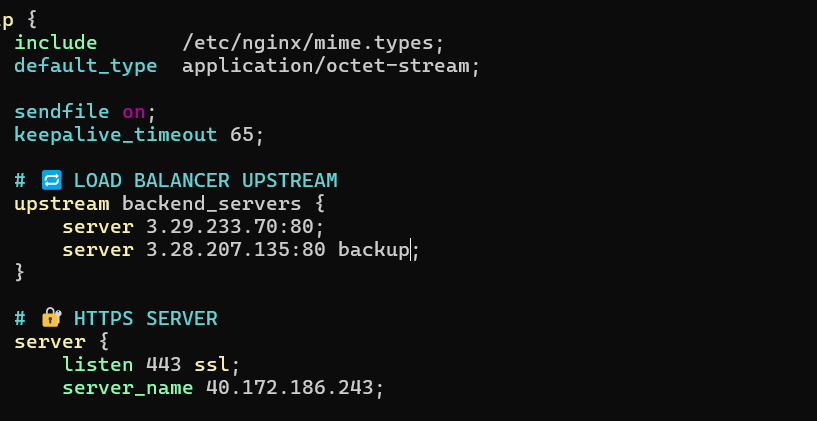
upstream backend\_servers {

server <web-1-public-ip>:80;

server <web-2-public-ip>:80 backup;

}

Save screenshot as: task9\_nginx\_conf\_ha\_web1\_primary.png — nginx.conf with web-2 as backup.



1. Restart Nginx:

sudo systemctl restart nginx

Test in browser:

Open browser and navigate to https://<webserver-public-ip>

Reload multiple times

You should ONLY see web-1 (primary server)

Save screenshot as: task9\_ha\_web1\_only.png — browser showing only web-1 content on multiple reloads.



1. Switch backup configuration:

sudo vim /etc/nginx/nginx.conf

Update to make web-1 backup:

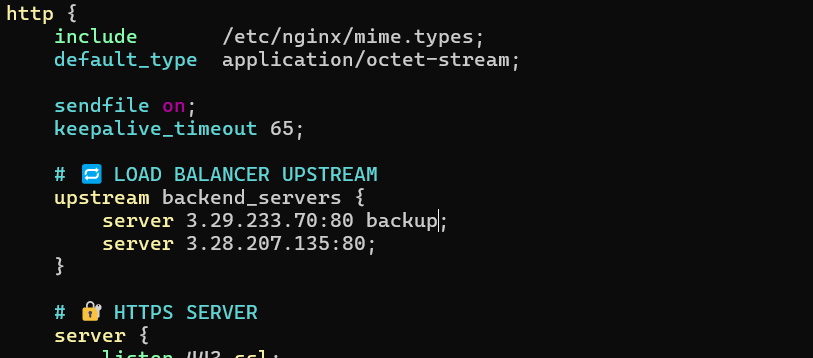
upstream backend\_servers {

server <web-1-public-ip>: 80 backup;

server <web-2-public-ip>:80;

}

Save screenshot as: task9\_nginx\_conf\_ha\_web2\_primary.png — nginx.conf with web-1 as backup.



1. Restart Nginx:

sudo systemctl restart nginx

Test in browser:

Reload multiple times

You should ONLY see web-2 (now the primary server)

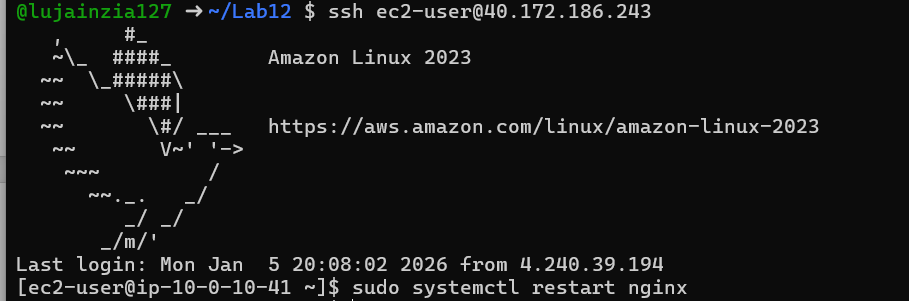
Save screenshot as: task9\_ha\_web2\_only.png — browser showing only web-2 content on multiple reloads.



Task 10 — Enable Nginx caching

1. SSH into the webserver:

ssh ec2-user@<webserver-public-ip>



1. Edit Nginx configuration to enable caching:

sudo vim /etc/nginx/nginx.conf

Add proxy cache configuration in the http block and location block:

http {

proxy\_cache\_path /var/cache/nginx levels=1:2 keys\_zone=my\_cache:10m inactive=60m max\_size=1g;

log\_format main '$remote\_addr - $remote\_user [$time\_local] "$request" '

'$status $body\_bytes\_sent "$http\_referer" '

'"$http\_user\_agent" "$http\_x\_forwarded\_for"';

# ... other settings ...

upstream backend\_servers {

server <web-1-public-ip>:80;

server <web-2-public-ip>: 80;

}

server {

listen 443 ssl;

server\_name $PUBLIC\_IP;

ssl\_certificate /etc/ssl/certs/selfsigned.crt;

ssl\_certificate\_key /etc/ssl/private/selfsigned.key;

location / { # root /usr/share/nginx/html; # index index.html; # proxy\_pass http://<web-1-public-ip>: 80;

proxy\_pass http://backend\_servers;

proxy\_cache my\_cache;

proxy\_cache\_valid 200 60m;

proxy\_cache\_key "$scheme$request\_uri";

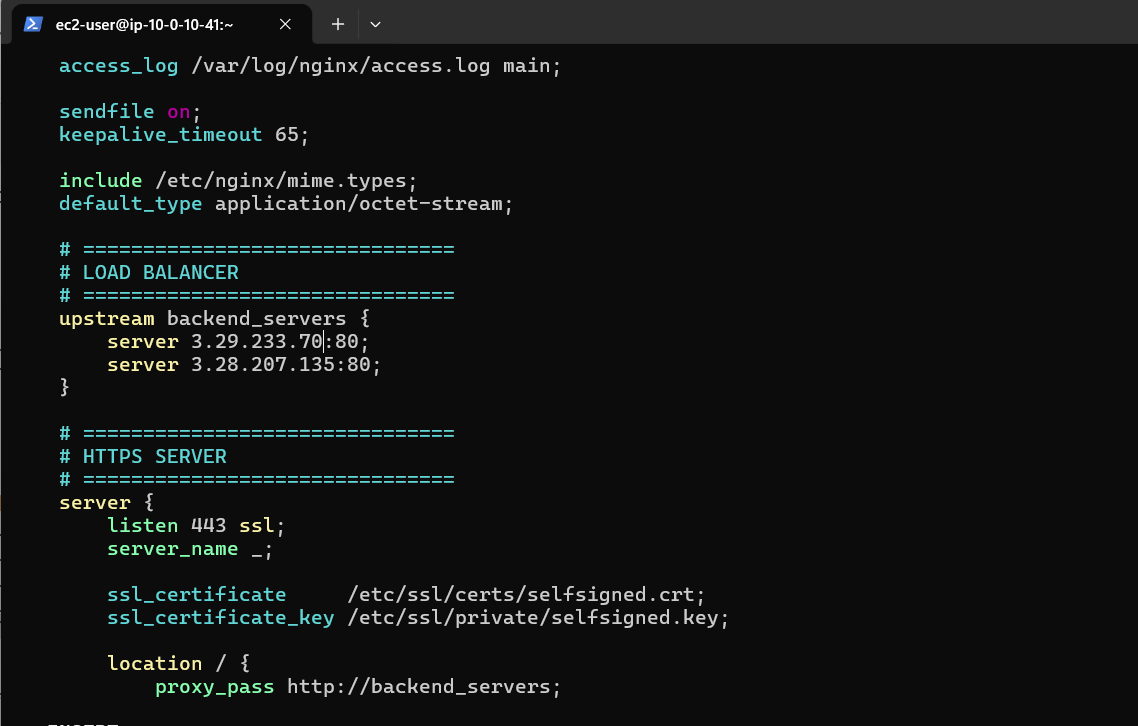
add\_header X-Cache-Status $upstream\_cache\_status;

}

}

# ... rest of config ...}

Save screenshot as: task10\_nginx\_conf\_cache.png — nginx.conf showing cache configuration.



1. Restart Nginx:

sudo systemctl restart nginx

Save screenshot as: task10\_nginx\_restart.png — terminal showing nginx restart.



1. Test caching in browser:

Open browser developer tools (F12)

Navigate to Network tab

Visit https://<webserver-public-ip>

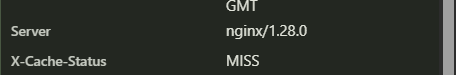
Check response headers for X-Cache-Status

First request should show MISS

Reload the page

Second request should show HIT

Save screenshot as: task10\_cache\_miss.png — browser dev tools showing X-Cache-Status: MISS on first request.



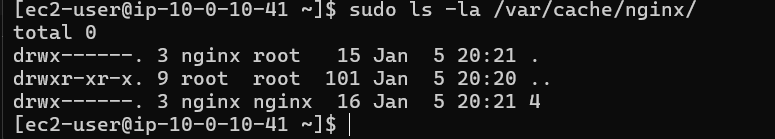
Save screenshot as: task10\_cache\_hit.png — browser dev tools showing X-Cache-Status: HIT on subsequent request.\



1. Verify cache directory:

ls -la /var/cache/nginx/

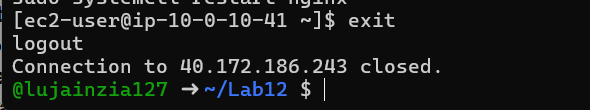
Save screenshot as: task10\_cache\_directory.png — terminal showing cache directory contents.



Cleanup:

1. Exit SSH session:

Exit

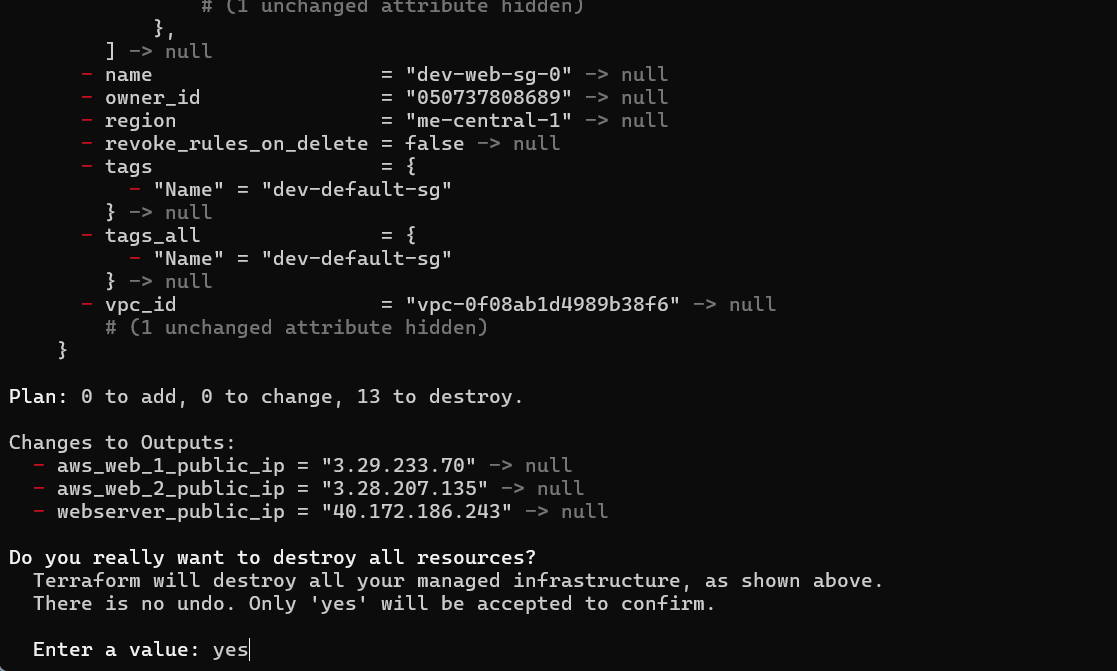


1. Destroy all resources:

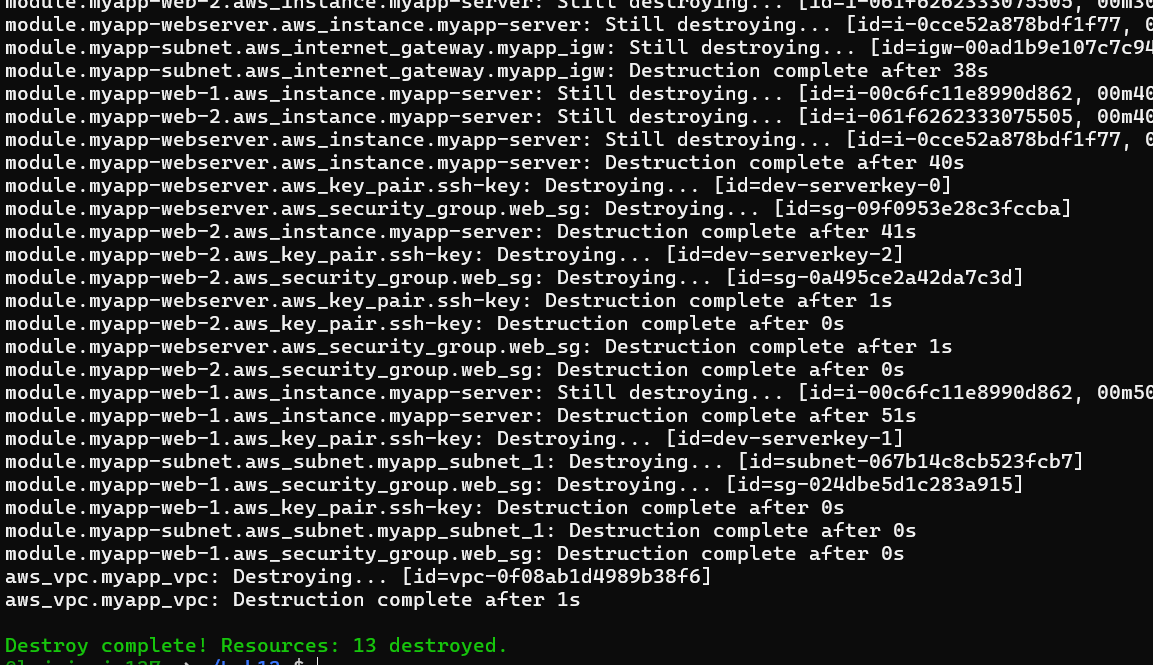
terraform destroy

Type yes when prompted for confirmation.

Save screenshot as: cleanup\_destroy\_prompt.png — terminal showing terraform destroy prompt.



Save screenshot as: cleanup\_destroy\_complete.png — terraform destroy completion output.



1. Verify state files:

cat terraform.tfstate

Save screenshot as: cleanup\_state\_empty.png — showing empty terraform. tfstate.



1. List all project files:

tree# or

ls -la

Save screenshot as: cleanup\_final\_files.png — showing final project structure.

