**CLOUD COMPUTING**

**SUBMITTED BY:** ARMEEN ABID **(BSE-011)**

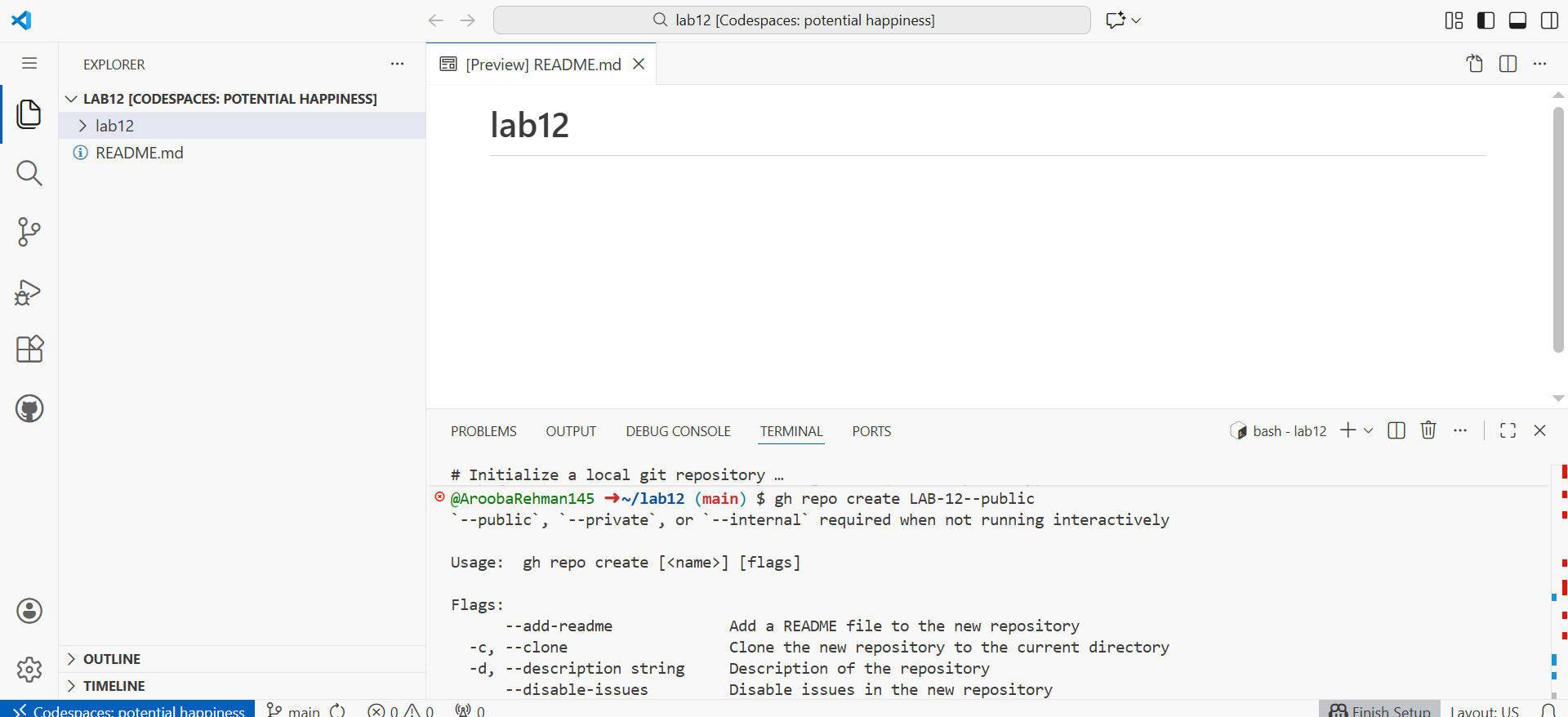
**SUBMITTED TO:** ENGR.MUHAMMAD SHOAIB

**LAB 12**

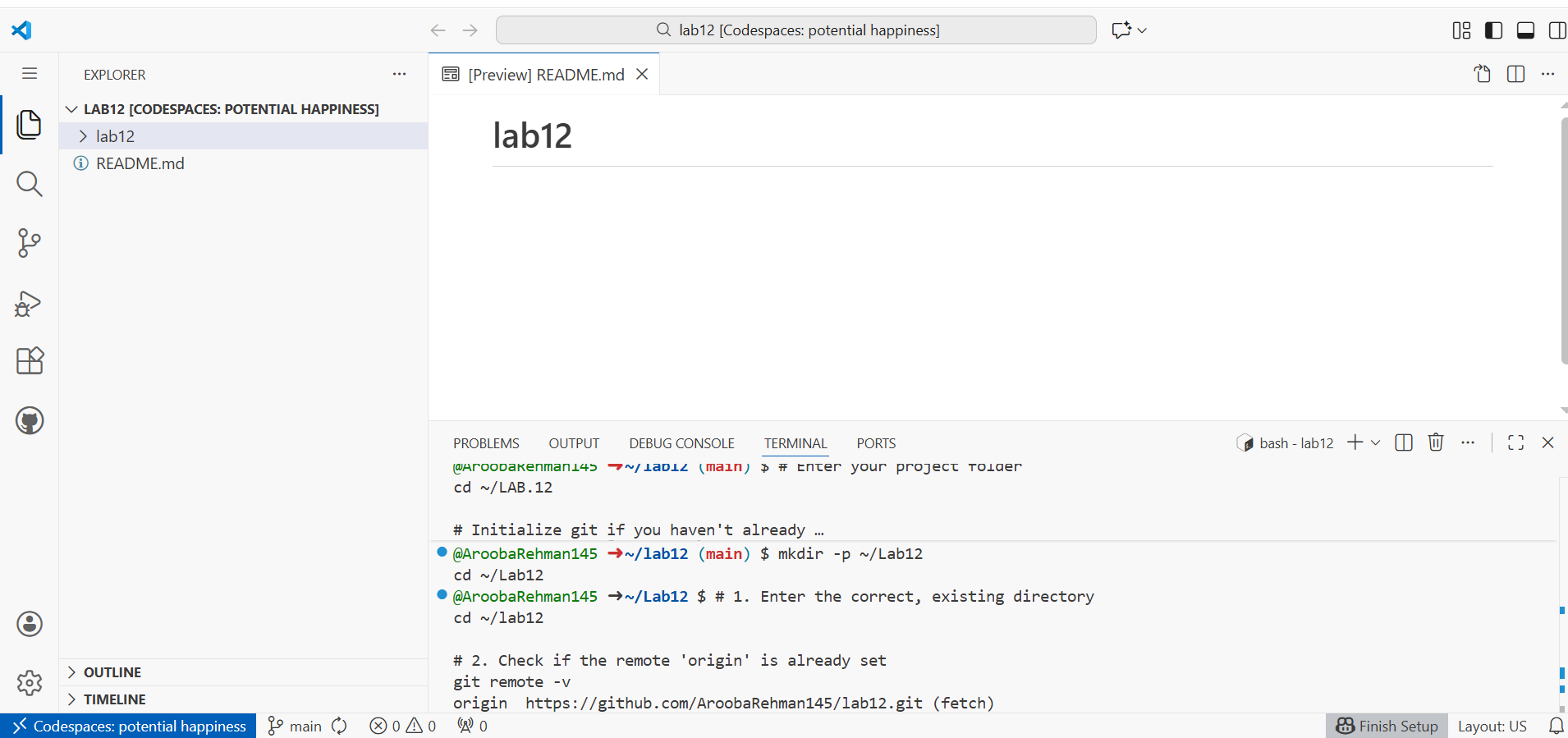
**Terraform Provisioners, Modules & Nginx Reverse Proxy/Load Balancer**

**TASK 1:**

Lab Setup

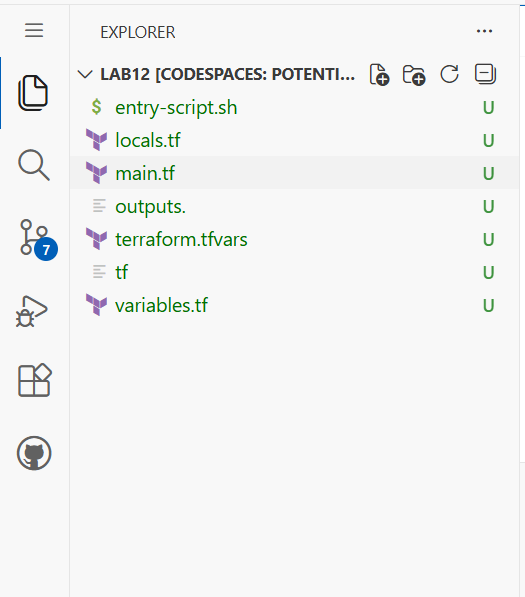


Create the initial project structure:

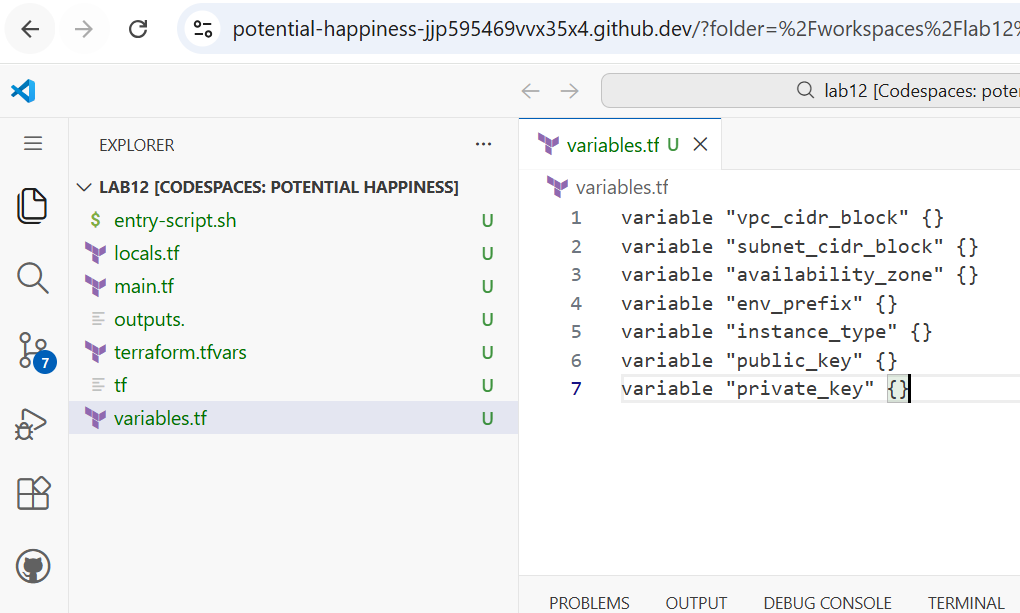


Create all required files:

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



1. Create variables.tf with the following content:
2. variable "vpc\_cidr\_block" {}
3. variable "subnet\_cidr\_block" {}
4. variable "availability\_zone" {}
5. variable "env\_prefix" {}
6. variable "instance\_type" {}
7. variable "public\_key" {}
8. variable "private\_key" {}

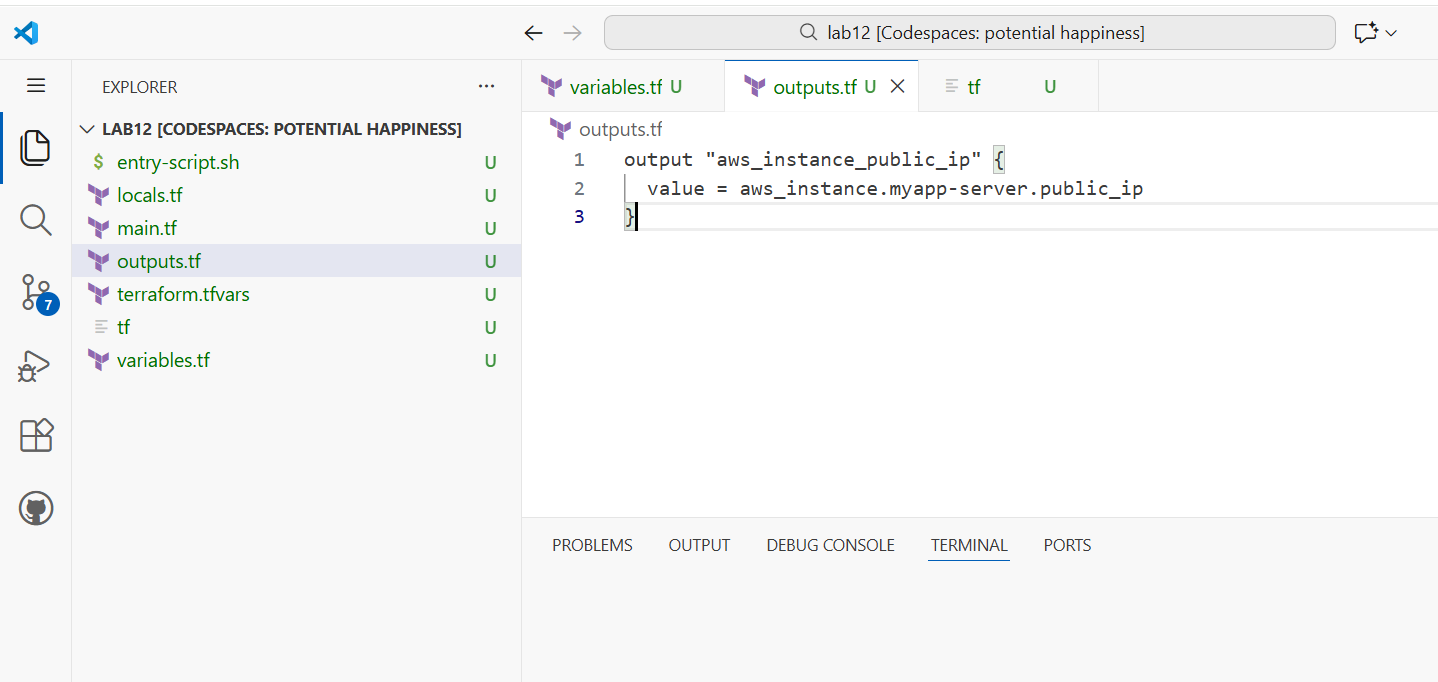


1. Create outputs.tf with the following content:

output "aws\_instance\_public\_ip" {

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

}

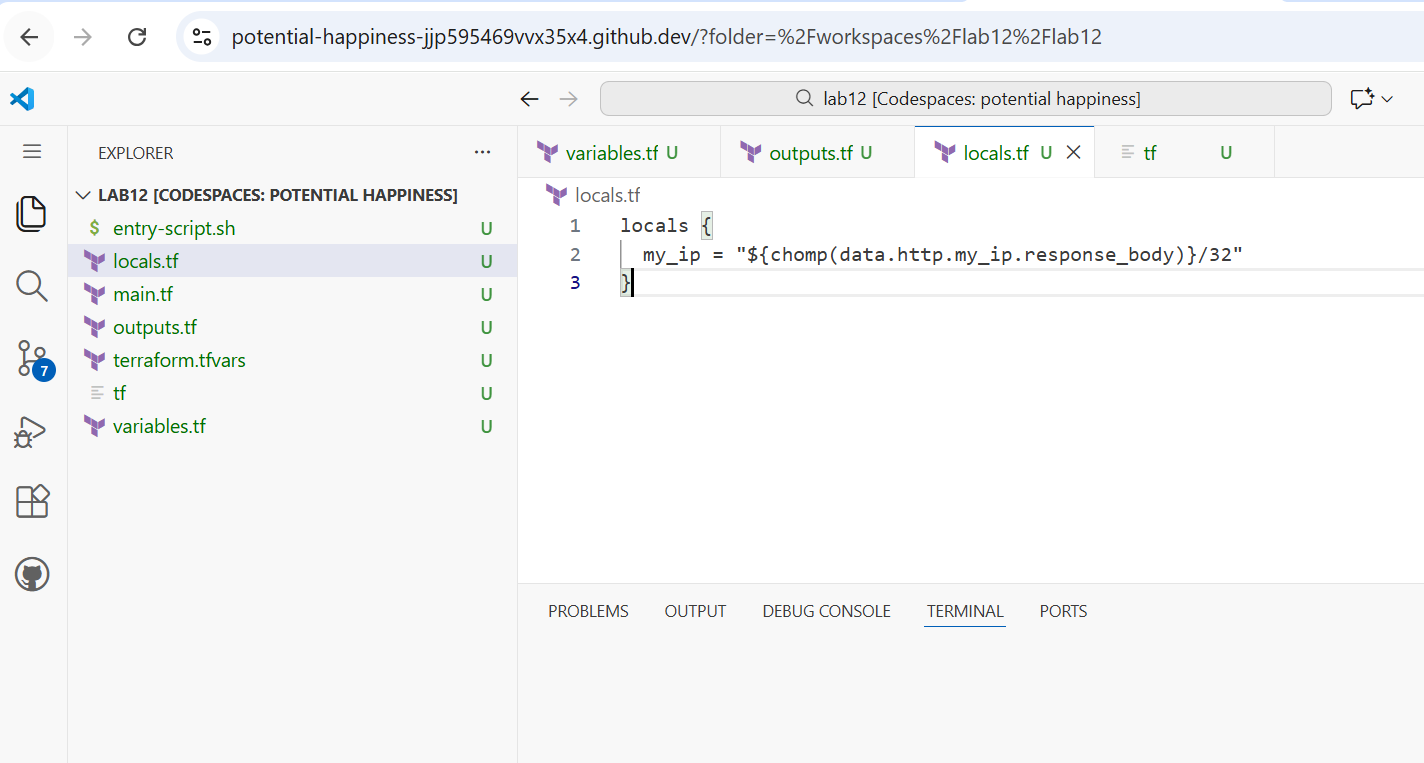


Create locals.tf with the following content

locals {

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

}



Create terraform.tfvars with the following content:

vpc\_cidr\_block = "10.0.0.0/16"

subnet\_cidr\_block = "10.0.10.0/24"

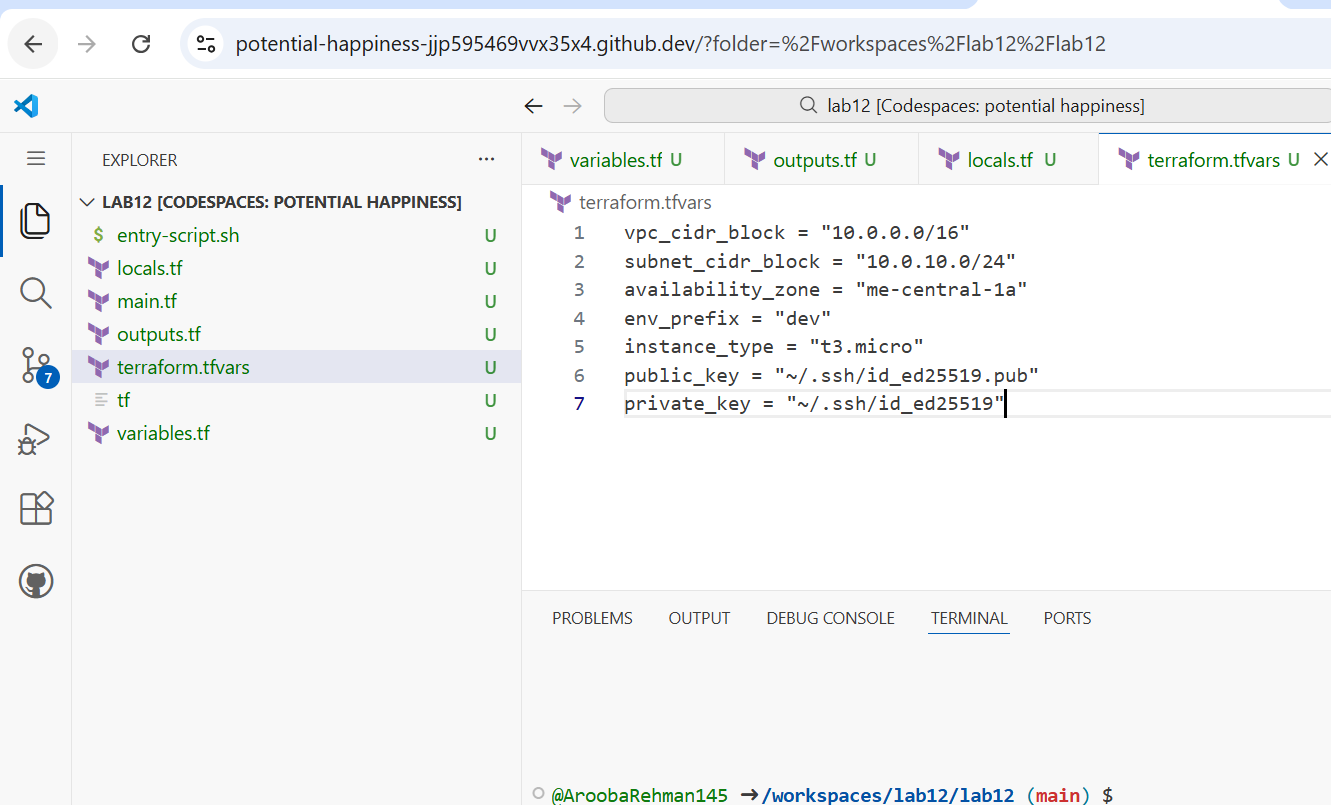
availability\_zone = "me-central-1a"

env\_prefix = "dev"

instance\_type = "t3.micro"

public\_key = "~/.ssh/id\_ed25519.pub"

private\_key = "~/.ssh/id\_ed25519"



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"

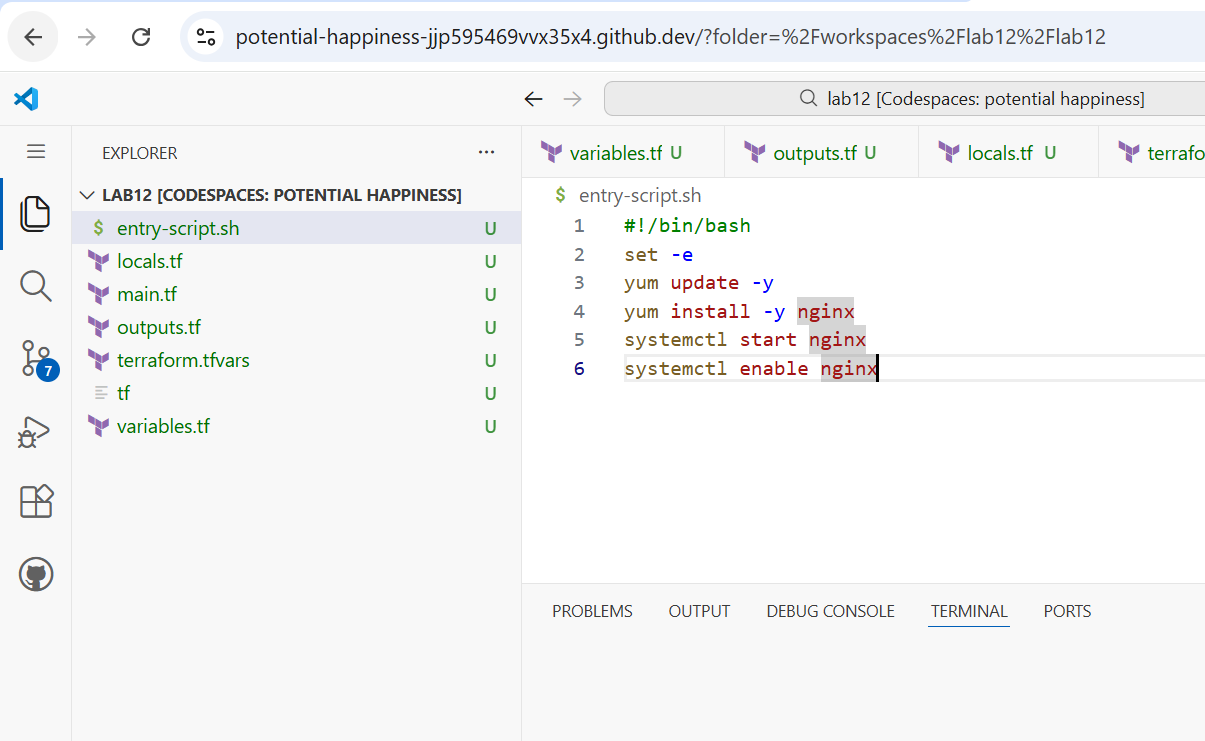
}

}

data "http" "my\_ip" {

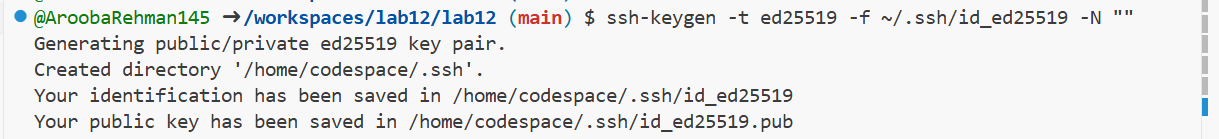
url = "https://icanhazip.com"

}



**Generate SSH key pair if not already exists**

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



**Initialize:**

terraform init

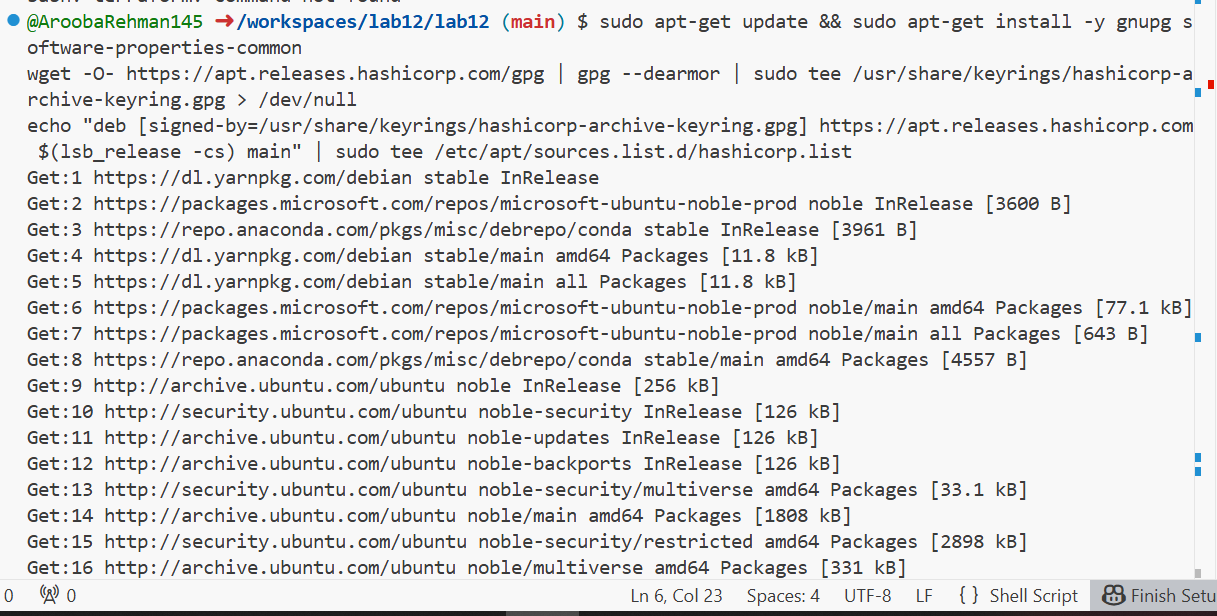
Firstly downloading terraform:

 **Add the HashiCorp repository:**

sudo apt-get update && sudo apt-get install -y gnupg software-properties-common

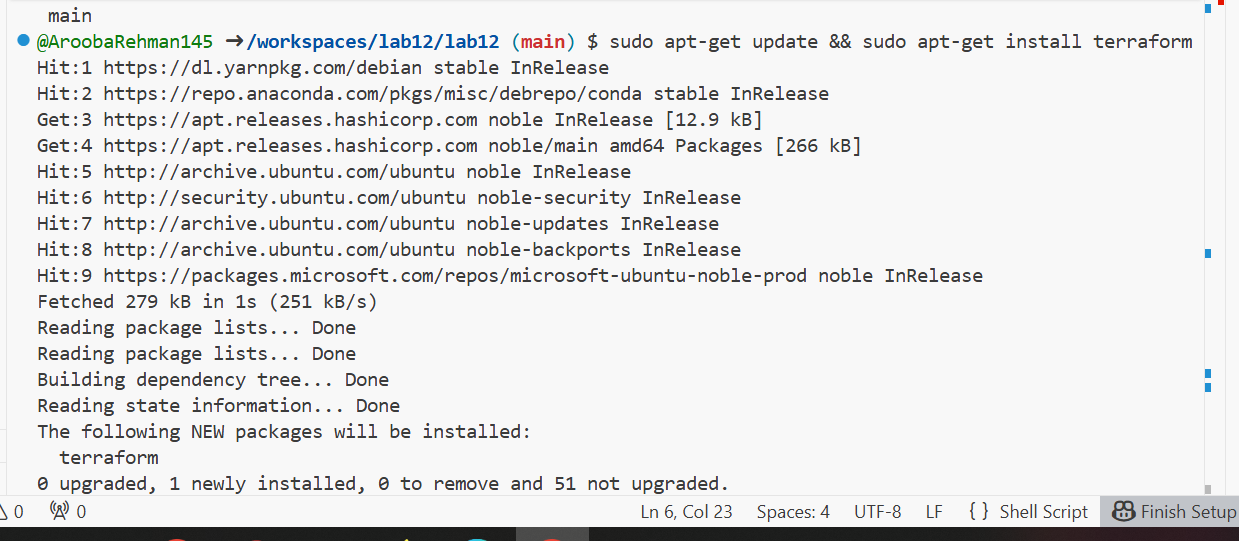
wget -O- https://apt.releases.hashicorp.com/gpg | gpg --dearmor | sudo tee /usr/share/keyrings/hashicorp-archive-keyring.gpg > /dev/null

echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com $(lsb\_release -cs) main" | sudo tee /etc/apt/sources.list.d/hashicorp.list



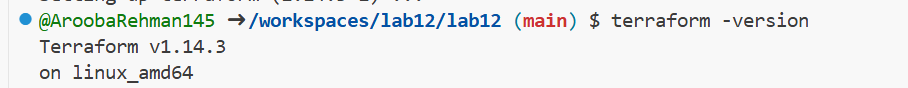
**Install the CLI:**

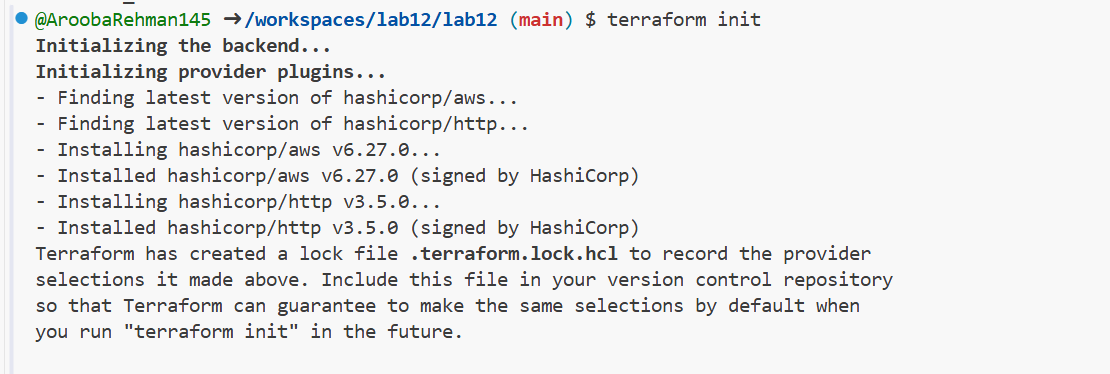
sudo apt-get update && sudo apt-get install terraform



**Verify the installation:**

terraform -version





**Apply:**

terraform apply -auto-approve

export AWS\_ACCESS\_KEY\_ID="YOUR\_ACCESS\_KEY"

export AWS\_SECRET\_ACCESS\_KEY="YOUR\_SECRET\_KEY"

export AWS\_DEFAULT\_REGION="me-central-1"

### Update your main.tf

Since you are providing the region via the terminal, you should simplify the provider block in your main.tf file. Open the file and ensure the top section looks like this:

provider "aws" {

# Region is now handled by your 'export AWS\_DEFAULT\_REGION' command

}

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"]

}

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" # Ensure this AMI exists in me-central-1

instance\_type = var.instance\_type

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

vpc\_security\_group\_ids = [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"

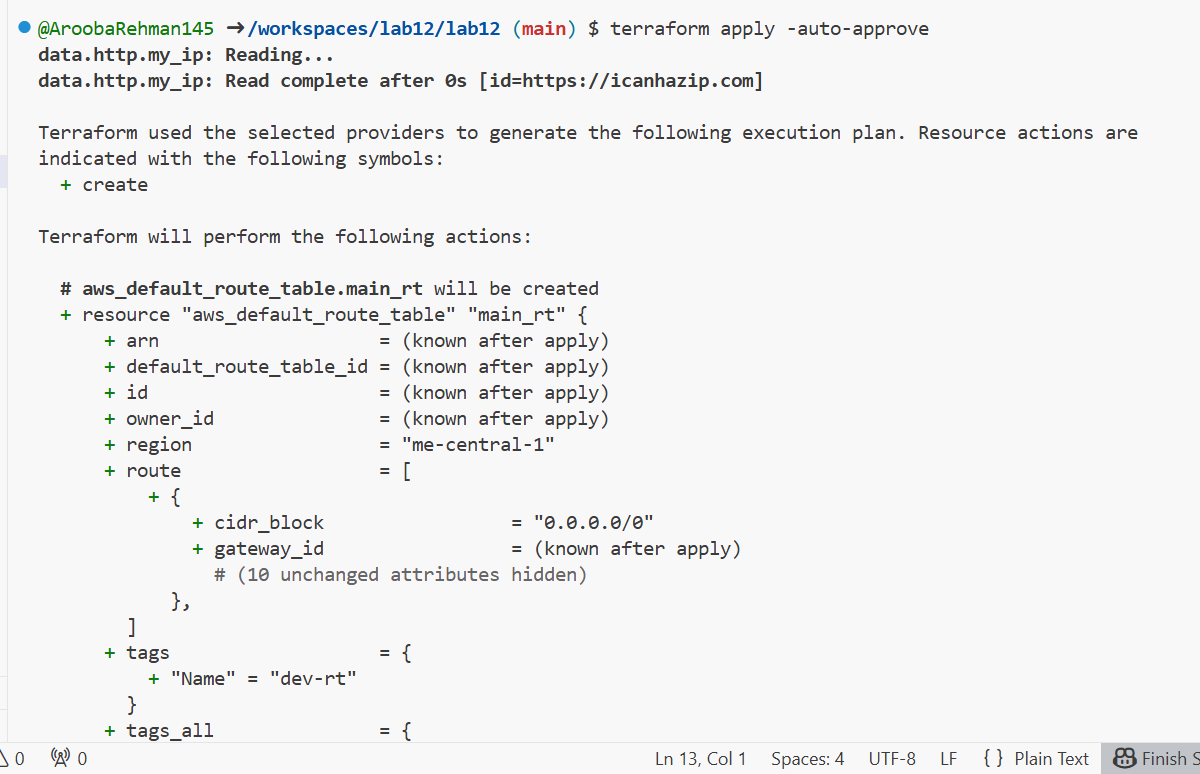
}

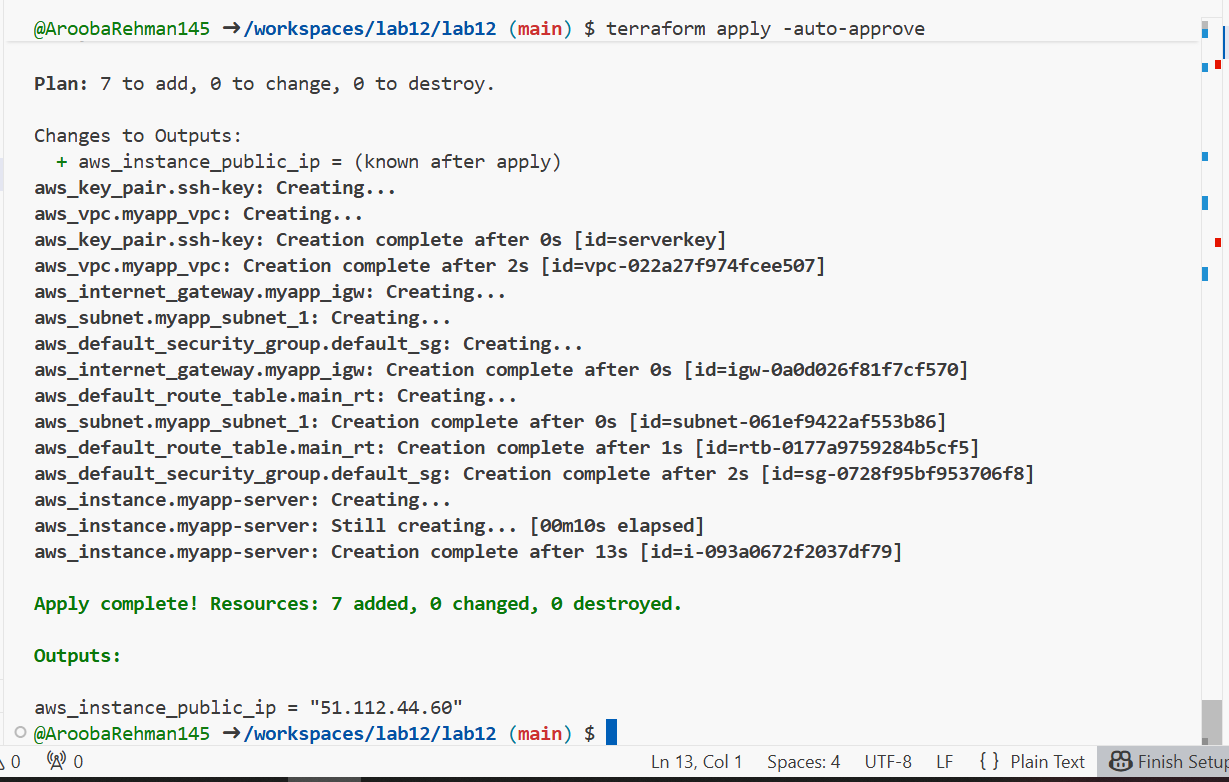
}

data "http" "my\_ip" {

url = "https://icanhazip.com"

}



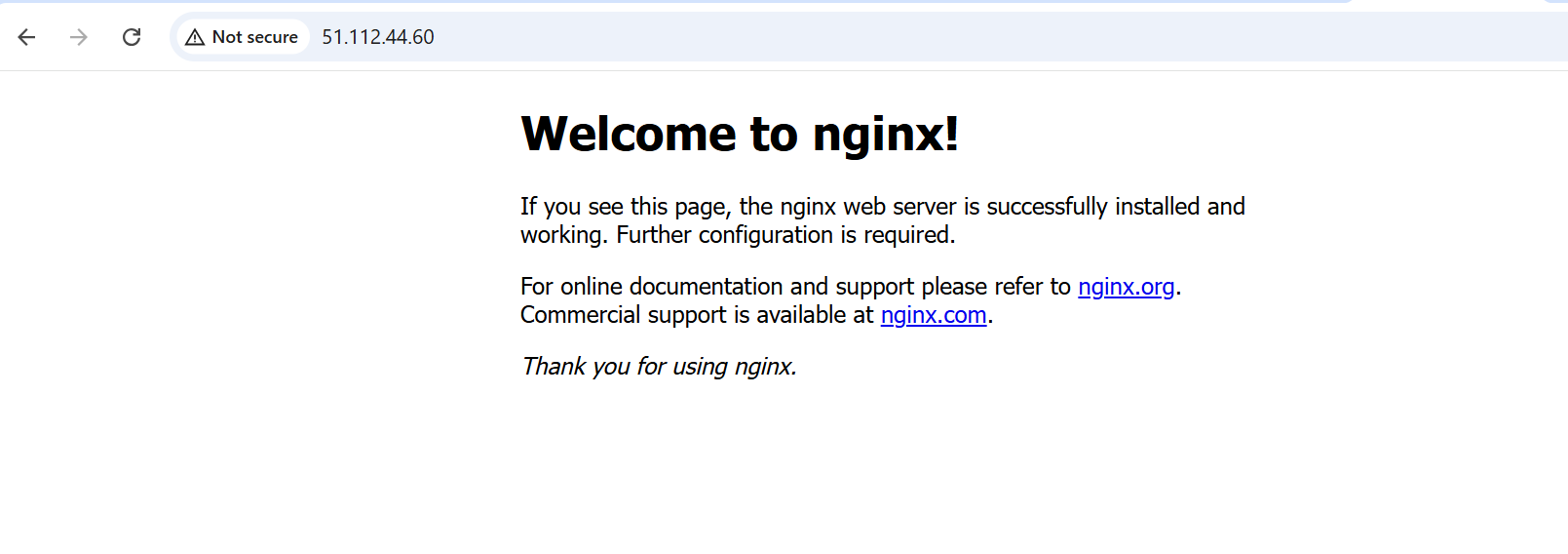


**Get IP:**

terraform output

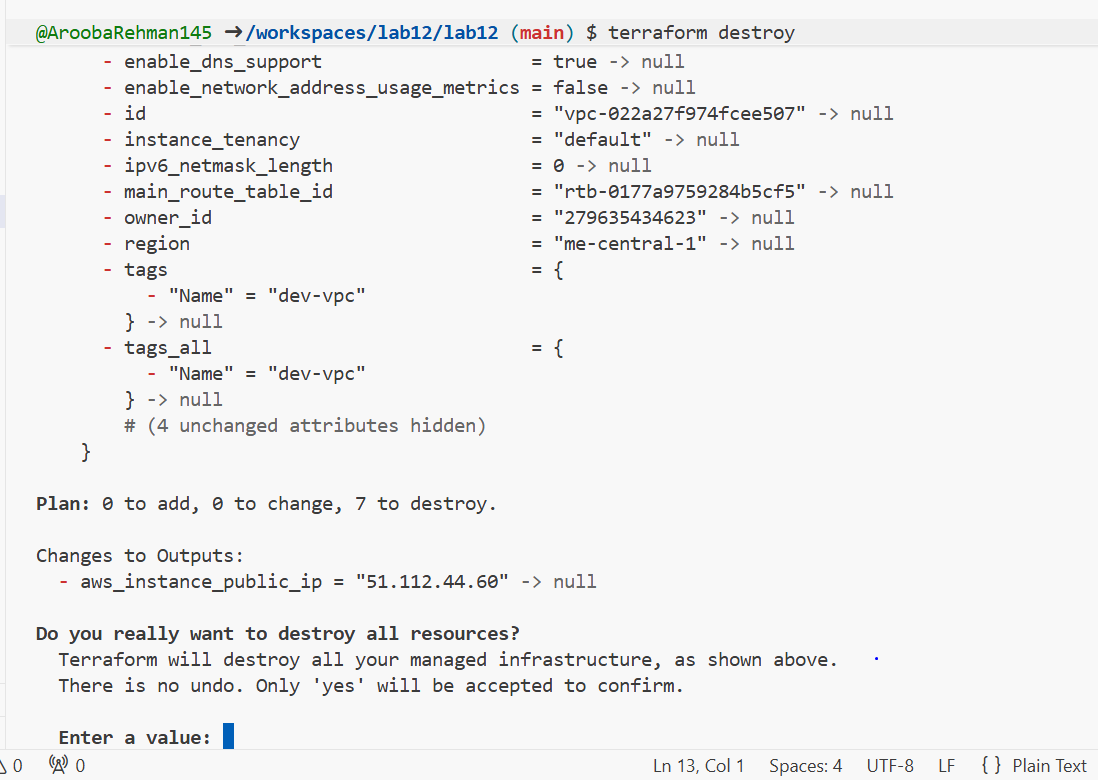


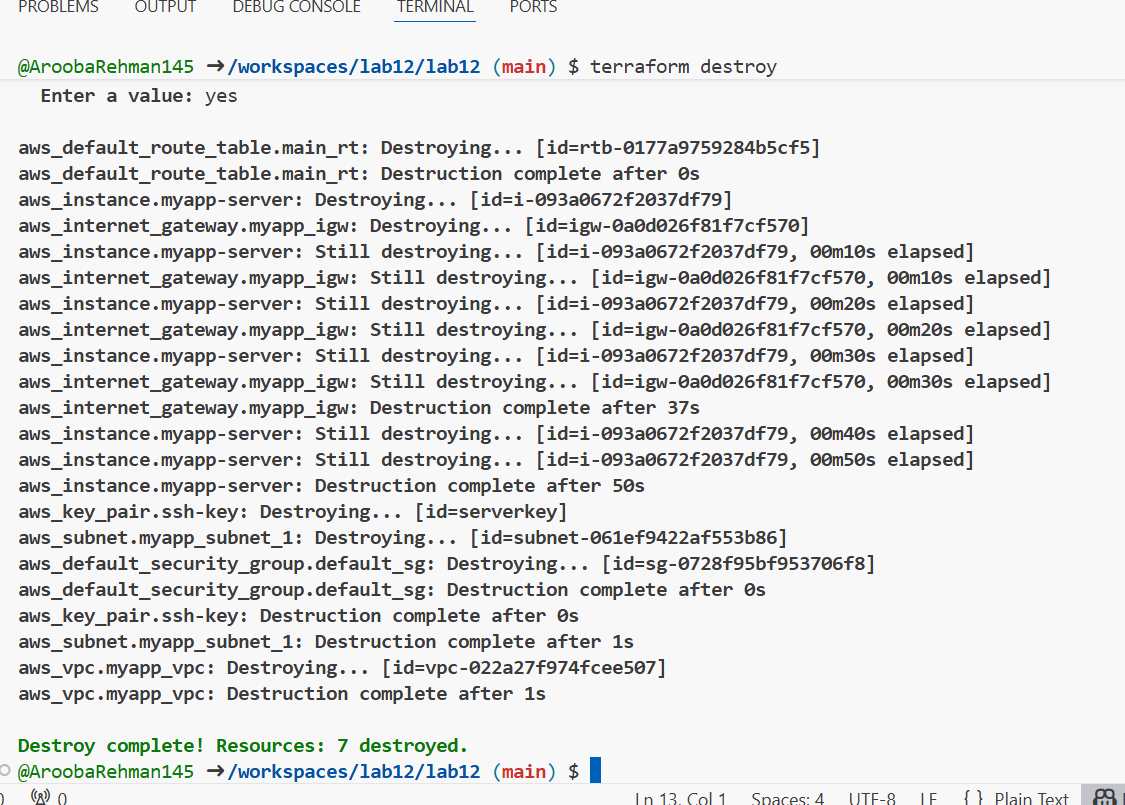
1. **Test nginx in browser:**



1. **Destroy resources:**

terraform destroy





**TASK 2:**

## **Use remote-exec provisioner**

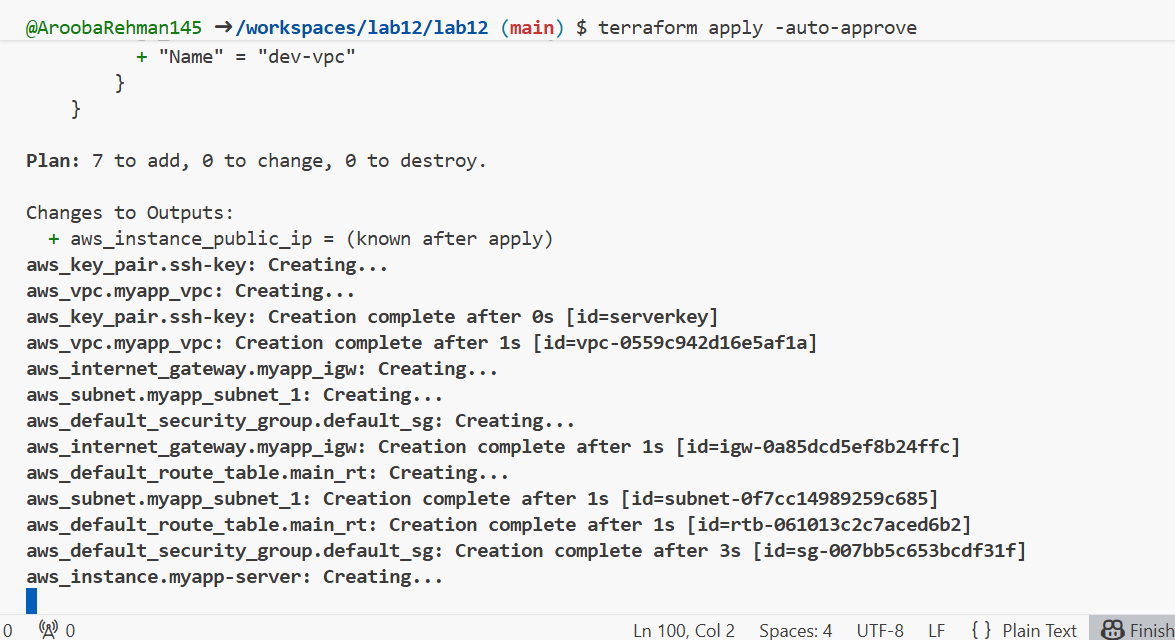
1. Modify the aws\_instance resource in main.tf to use remote-exec provisioner:

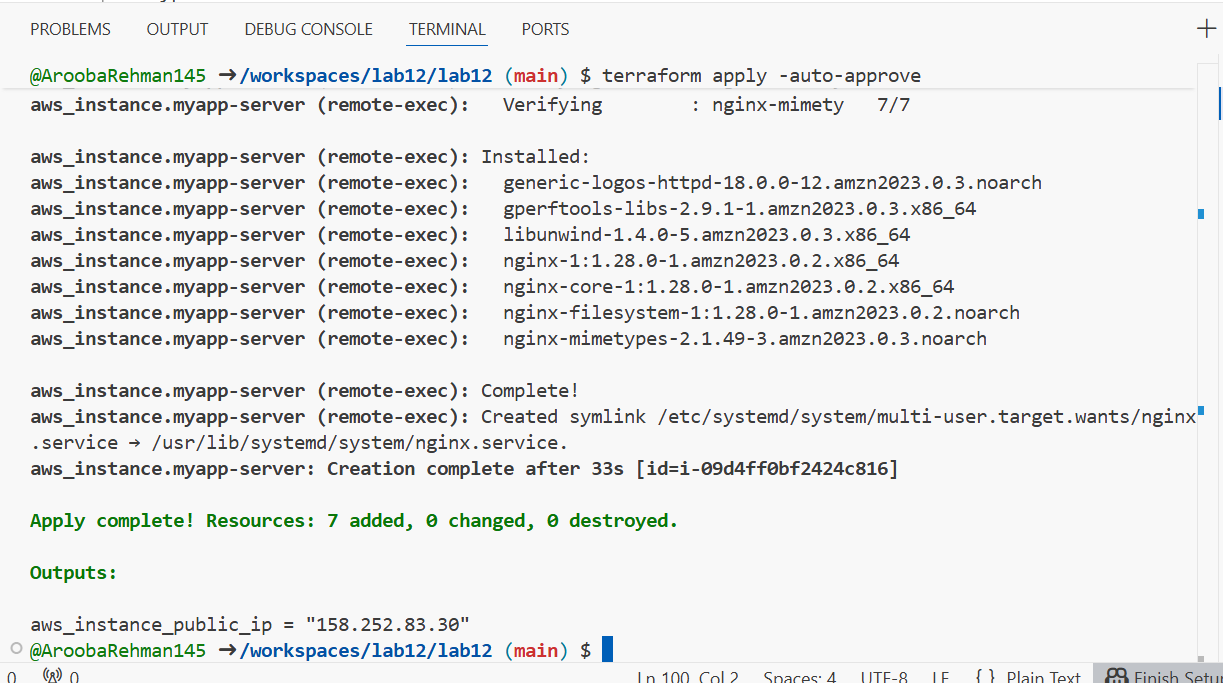
Updating main.tf file (aws\_instance block)



1. Apply the configuration:

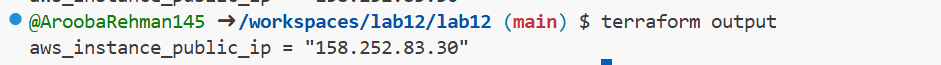
terraform apply -auto-approve

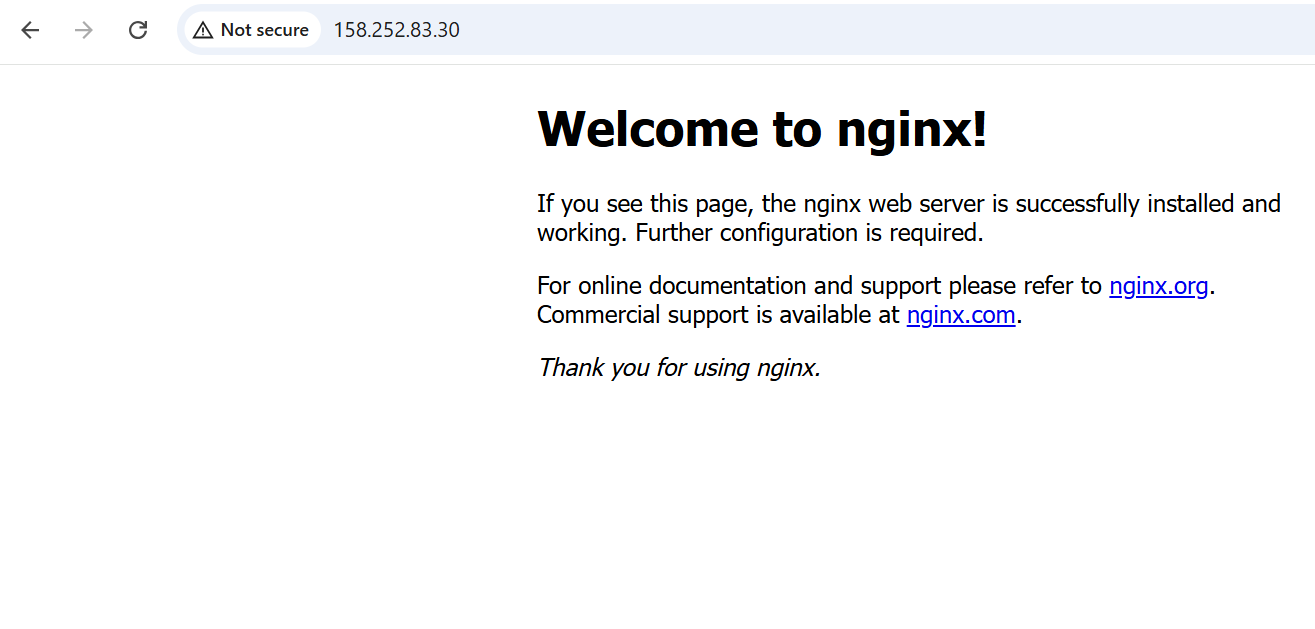




1. Display the output:

terraform output





## **Task 3 — Use file and local-exec provisioners**

n this task, you will add the file provisioner to upload the script and the local-exec provisioner to log instance information locally.

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"

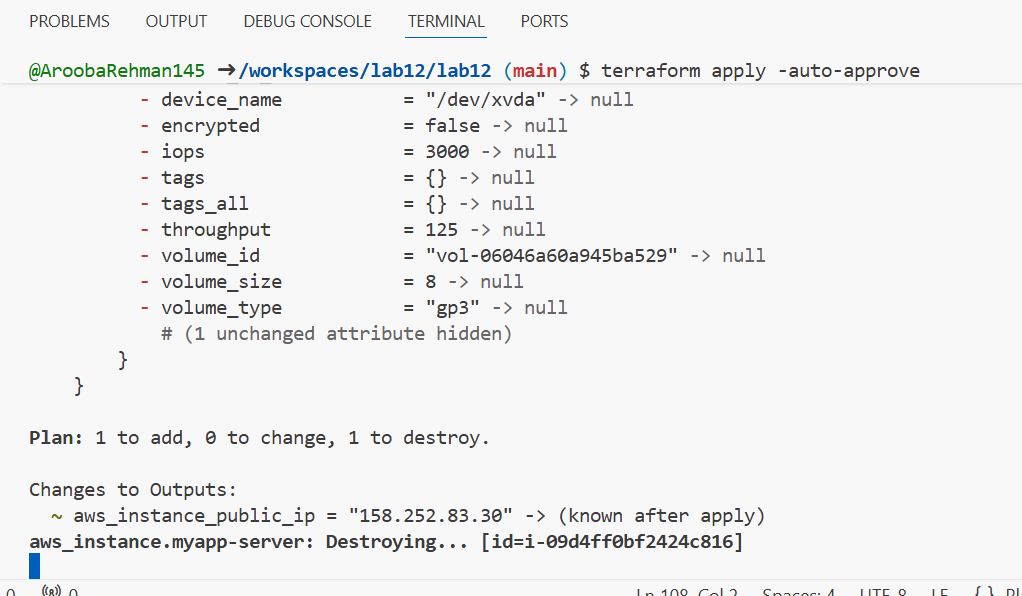
}

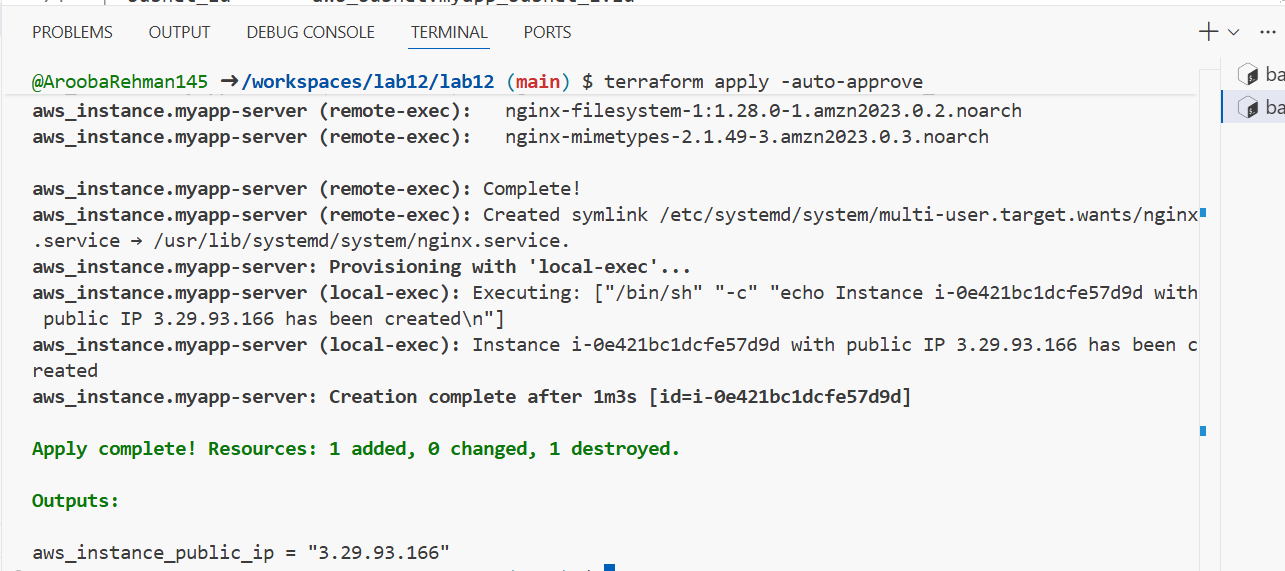
}



1. Apply the configuration:

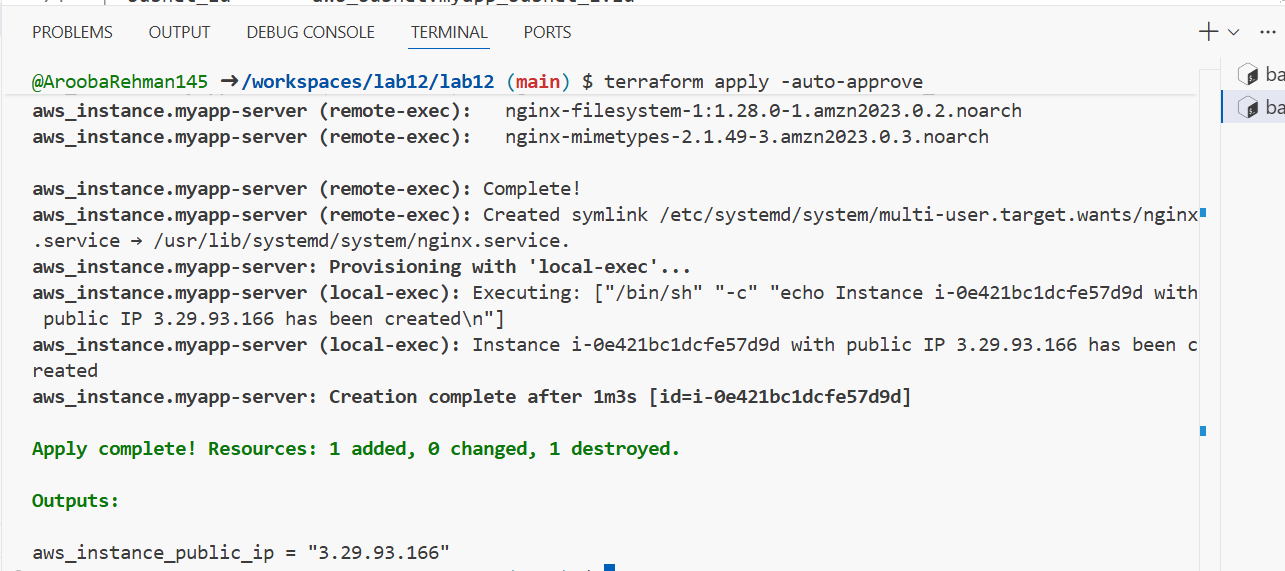
terraform apply -auto-approve

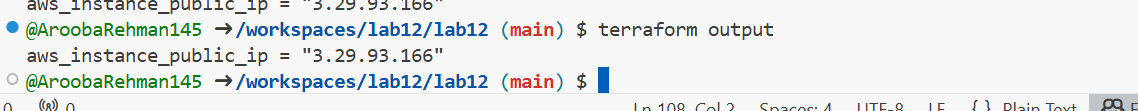


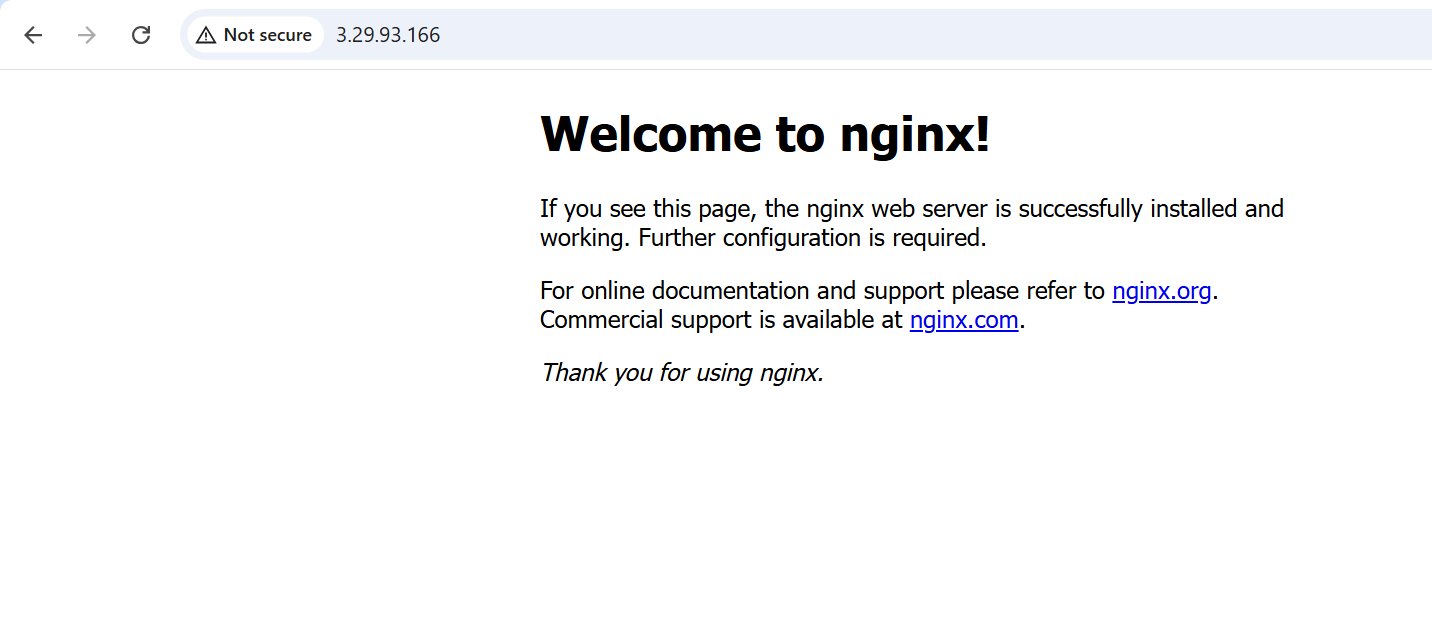


1. Display the output:

terraform output







## **Task 4 — Create Terraform modules (subnet module)**

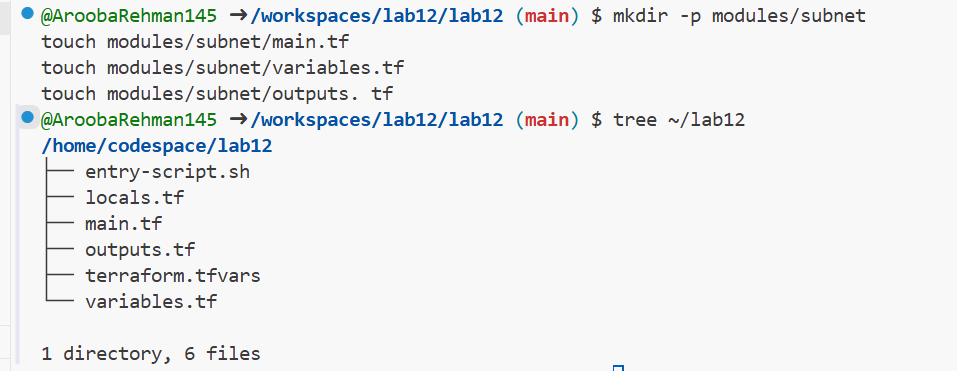
1. Create the module directory structure:

mkdir -p modules/subnet

touch modules/subnet/main.tf

touch modules/subnet/variables.tf

touch modules/subnet/outputs. Tf





1. Create modules/subnet/variables.tf:

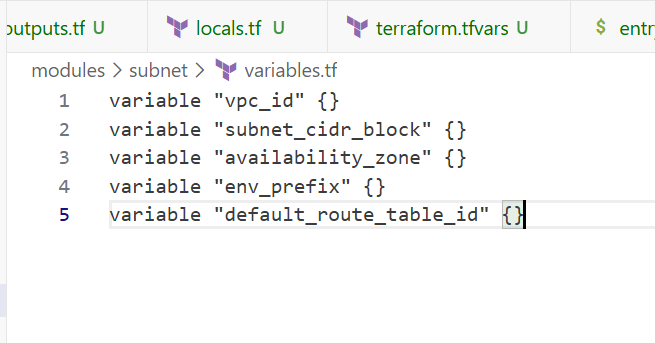
variable "vpc\_id" {}

variable "subnet\_cidr\_block" {}

variable "availability\_zone" {}

variable "env\_prefix" {}

variable "default\_route\_table\_id" {}



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" {

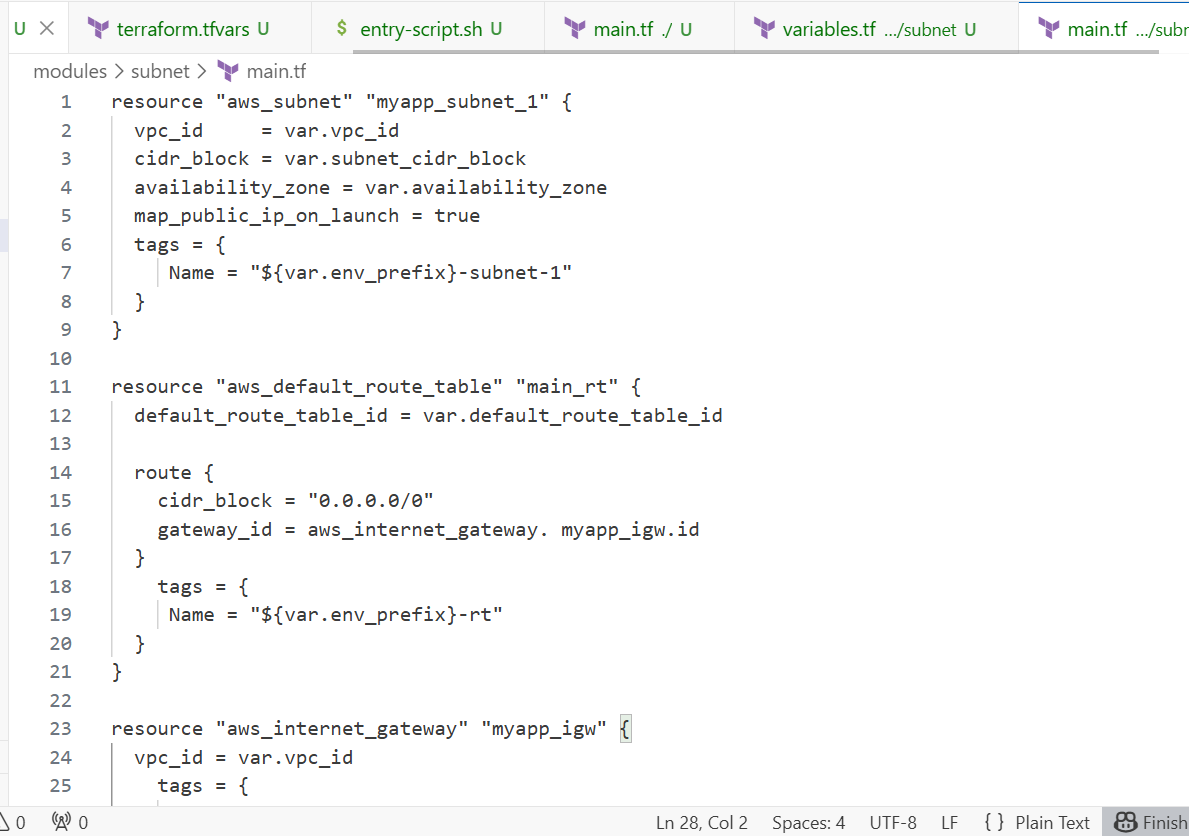
vpc\_id = var.vpc\_id

tags = {

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

}

}

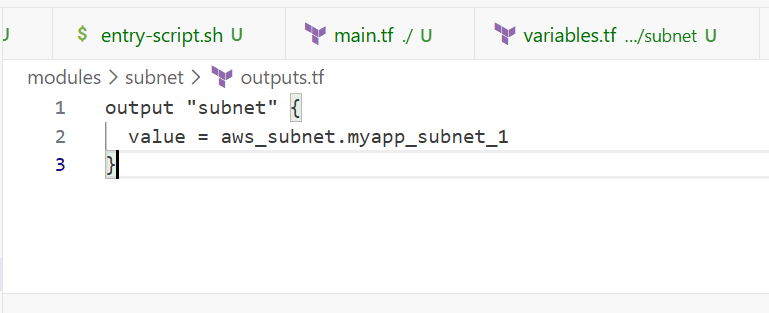


1. Create modules/subnet/outputs.tf:

output "subnet" {

value = aws\_subnet.myapp\_subnet\_1

}



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 ...

}

Full updated code:

provider "aws" {

# Region is handled by 'export AWS\_DEFAULT\_REGION'

}

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

cidr\_block = var.vpc\_cidr\_block

tags = {

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

}

}

# --- MODULE CALL START ---

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

}

# --- MODULE CALL END ---

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"]

}

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"

instance\_type = var.instance\_type

# Reference the module output here:

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

vpc\_security\_group\_ids = [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"

}

}

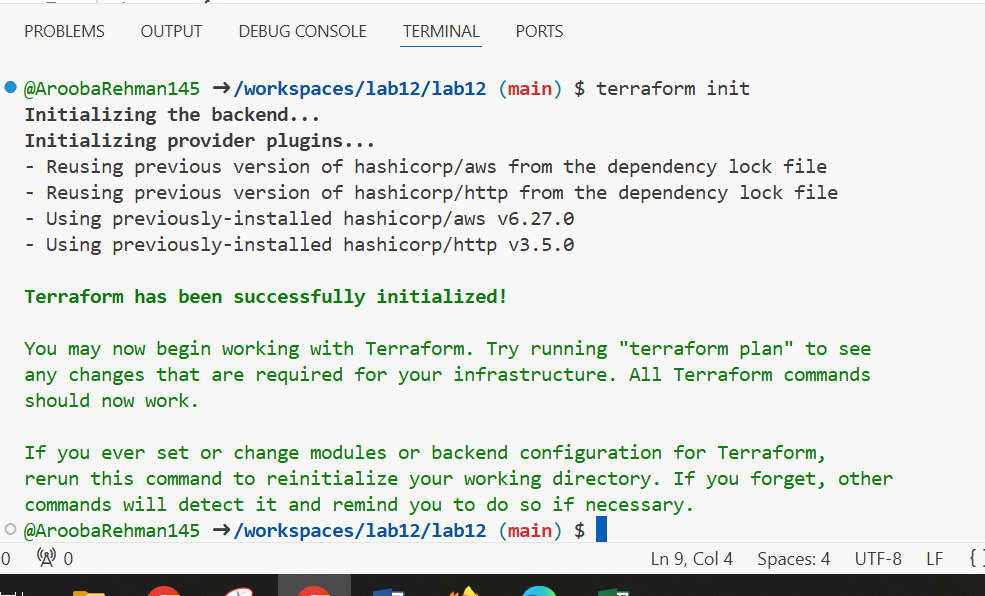
data "http" "my\_ip" {

url = "https://icanhazip.com"

}

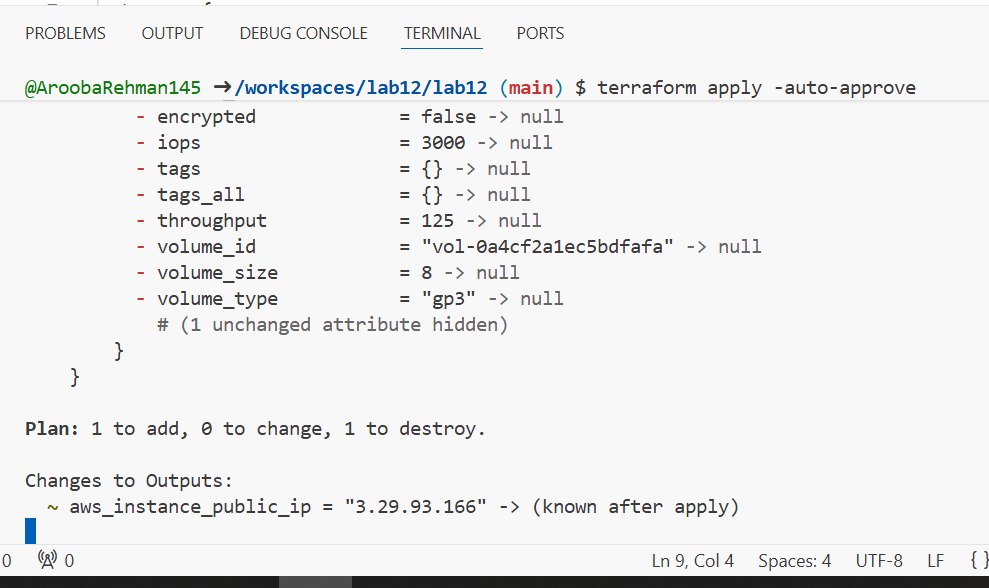
1. Initialize Terraform to download the module:

terraform init



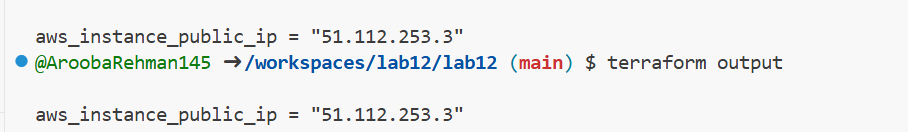
1. Apply the configuration:

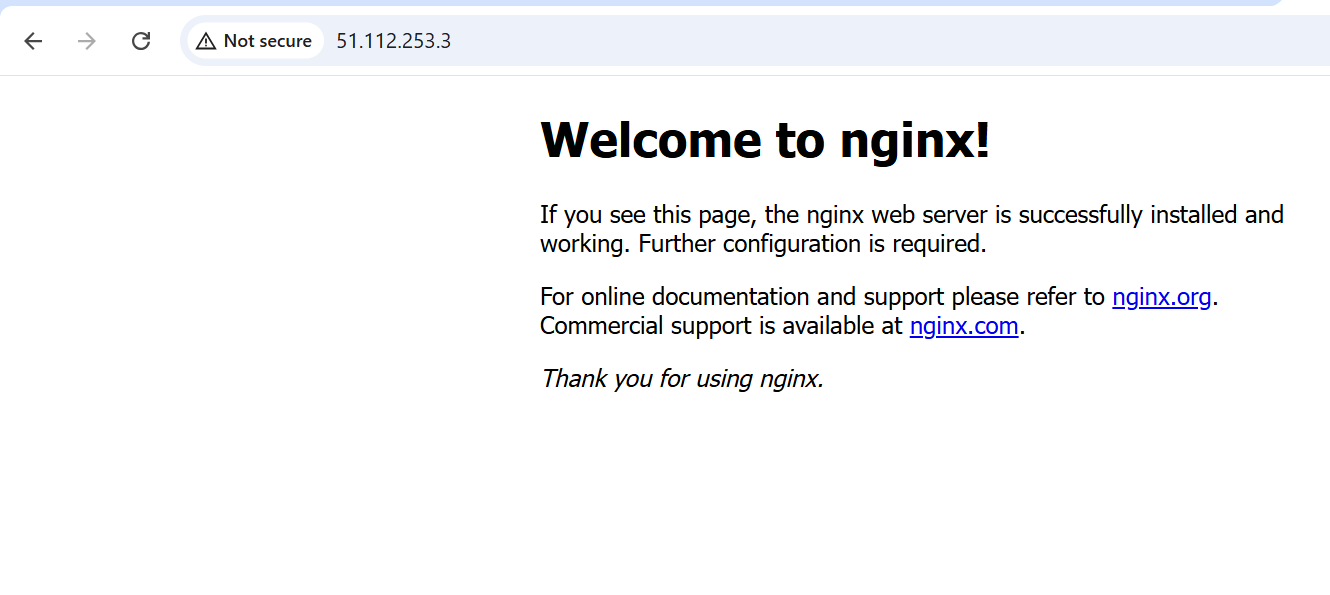
terraform apply -auto-approve



1. Display the output:

terraform output





**TASK 5:**

In this task, you will create a reusable webserver module for EC2 instances.

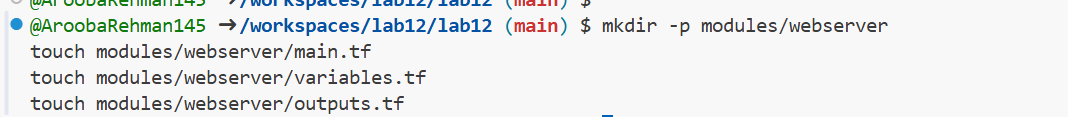
1. Create the webserver module directory structure:

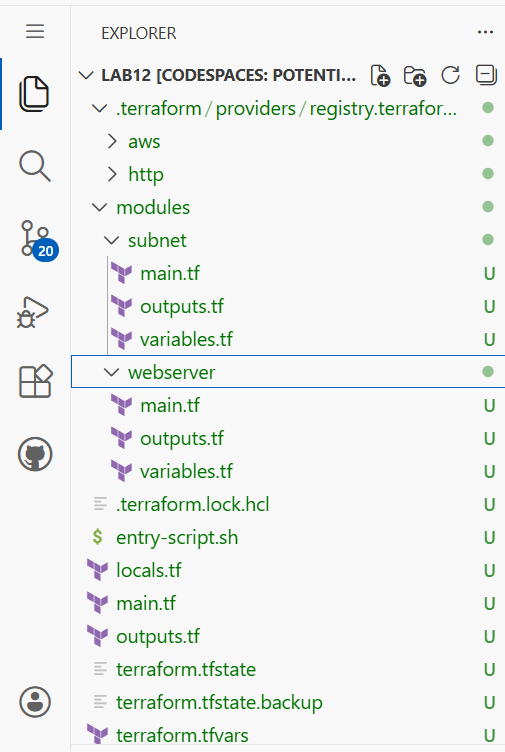
mkdir -p modules/webserver

touch modules/webserver/main.tf

touch modules/webserver/variables.tf

touch modules/webserver/outputs.tf





1. Create modules/webserver/variables.tf:

variable "env\_prefix" {}

variable "instance\_type" {}

variable "availability\_zone" {}

variable "public\_key" {}

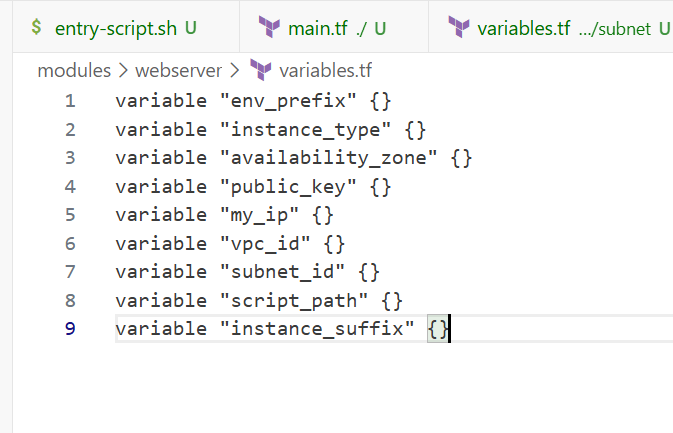
variable "my\_ip" {}

variable "vpc\_id" {}

variable "subnet\_id" {}

variable "script\_path" {}

variable "instance\_suffix" {}



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

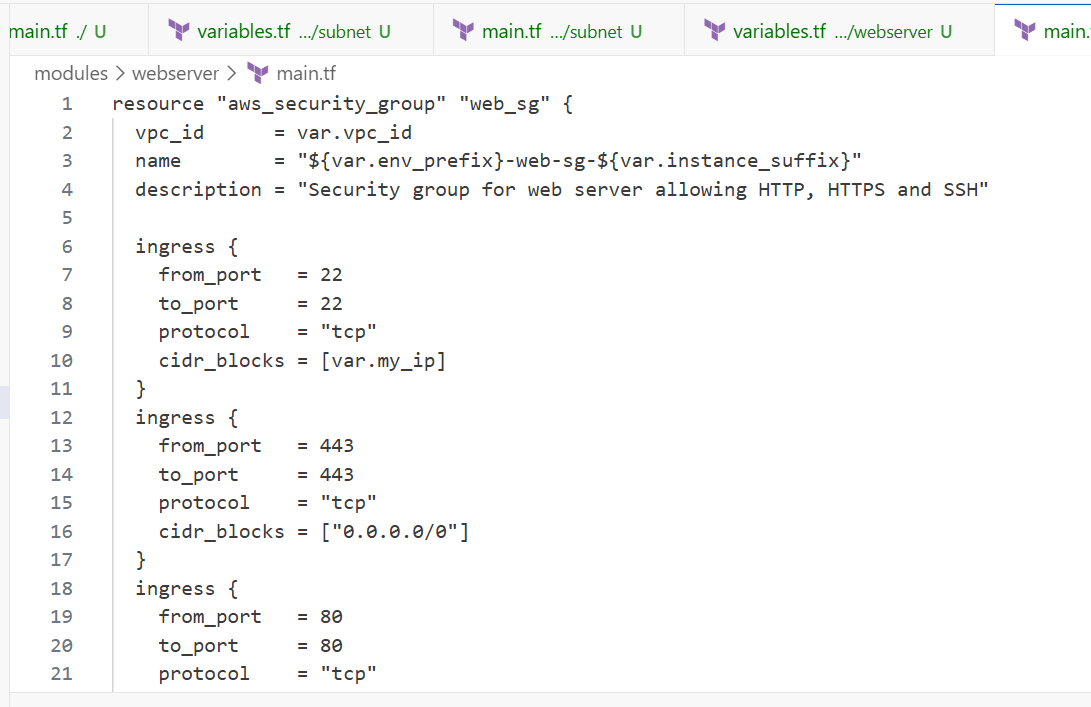
user\_data = file(var.script\_path)

tags = {

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

}

}

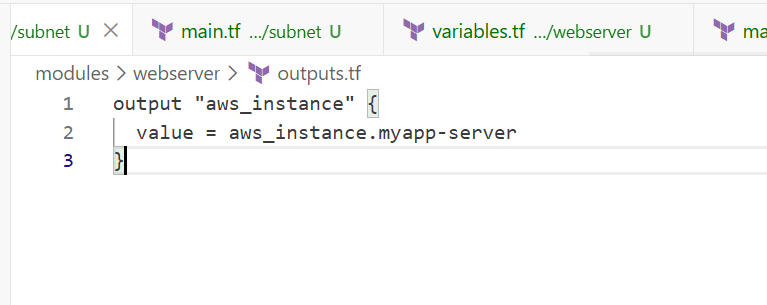


1. Create modules/webserver/outputs.tf:

output "aws\_instance" {

value = aws\_instance.myapp-server

}



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"

}

* provider "aws" {
* # Authentication is handled by terminal 'export' commands for Access Key, Secret Key, and Region.
* }
* # The VPC remains in the root main.tf as the foundation for both modules.
* resource "aws\_vpc" "myapp\_vpc" {
* cidr\_block = var.vpc\_cidr\_block
* tags = {
* Name = "${var.env\_prefix}-vpc"
* }
* }
* # --- Task 3: Subnet Module Call ---
* 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
* }
* # --- Task 3: Webserver Module Call ---
* 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"
* }
* # Data source for local IP remains in root to feed the webserver module.
* data "http" "my\_ip" {
* url = "https://icanhazip.com"
* }

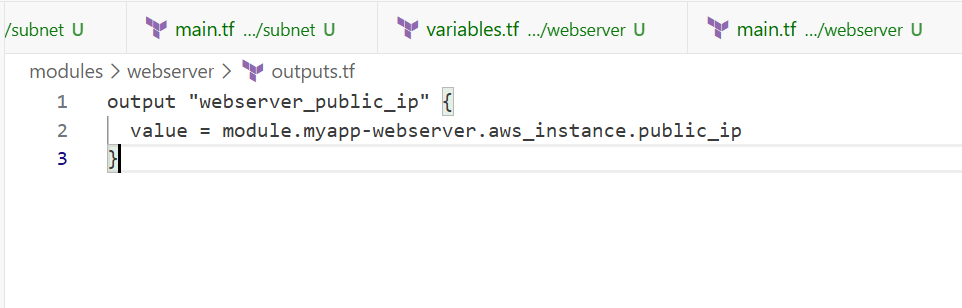


1. Update outputs.tf:

output "webserver\_public\_ip" {

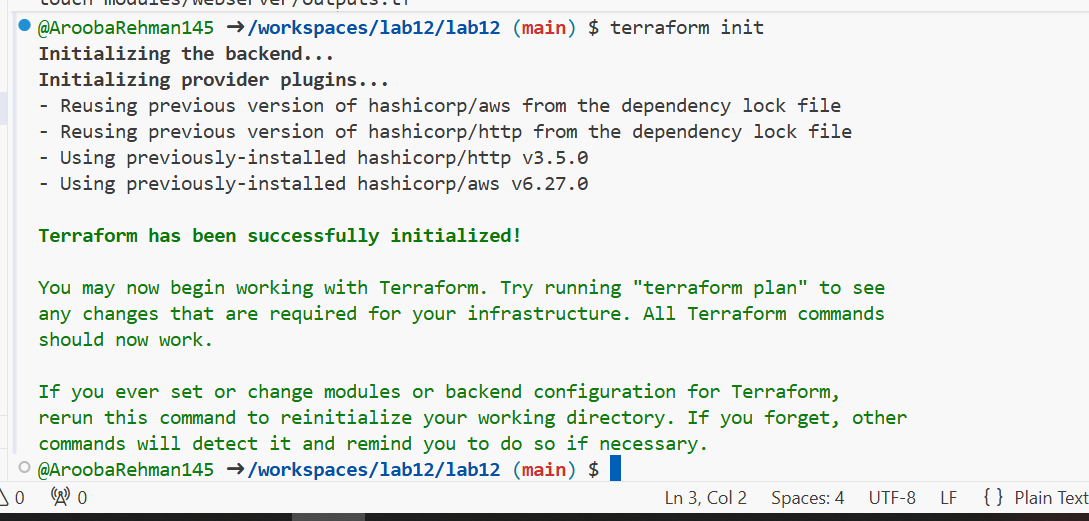
value = module.myapp-webserver.aws\_instance.public\_ip

}



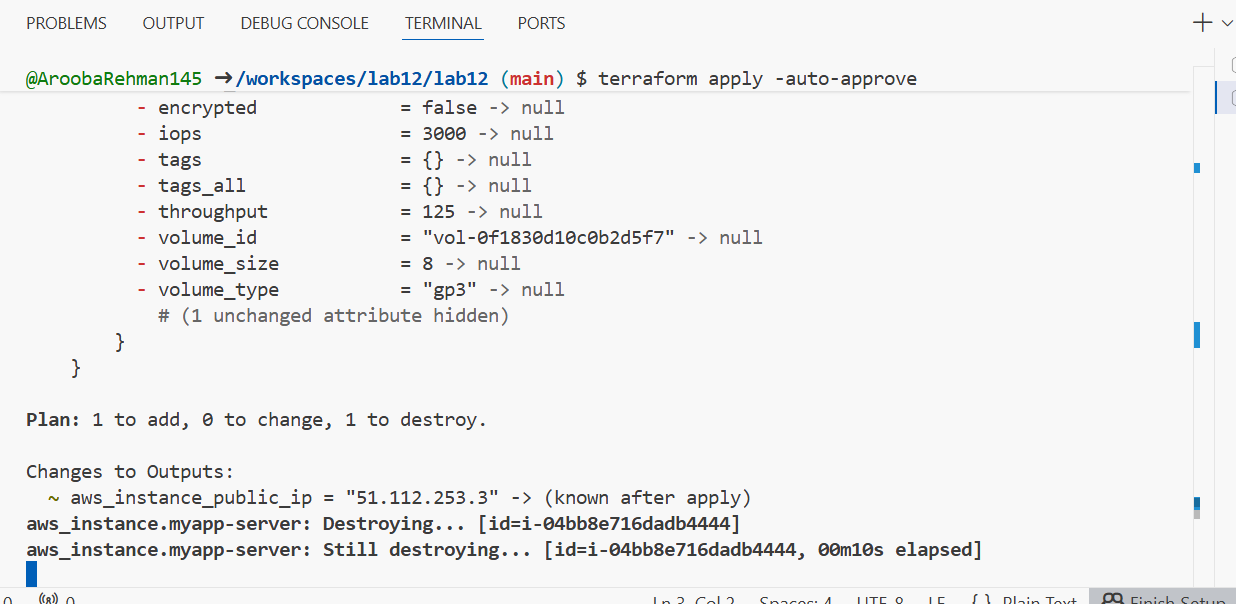
1. Initialize Terraform:

terraform init



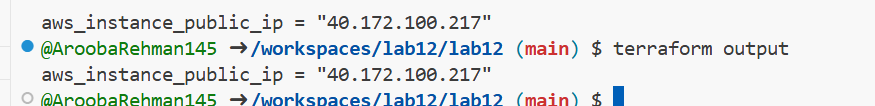
1. Apply the configuration:

terraform apply -auto-approve



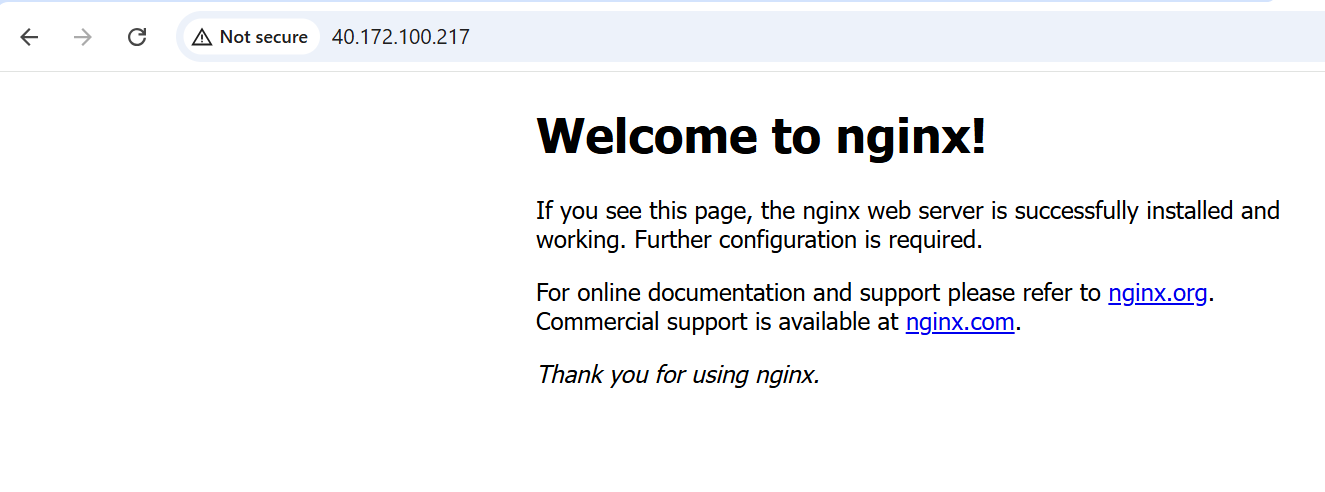
1. Display the output:

terraform output

* 

1. Test nginx in browser:

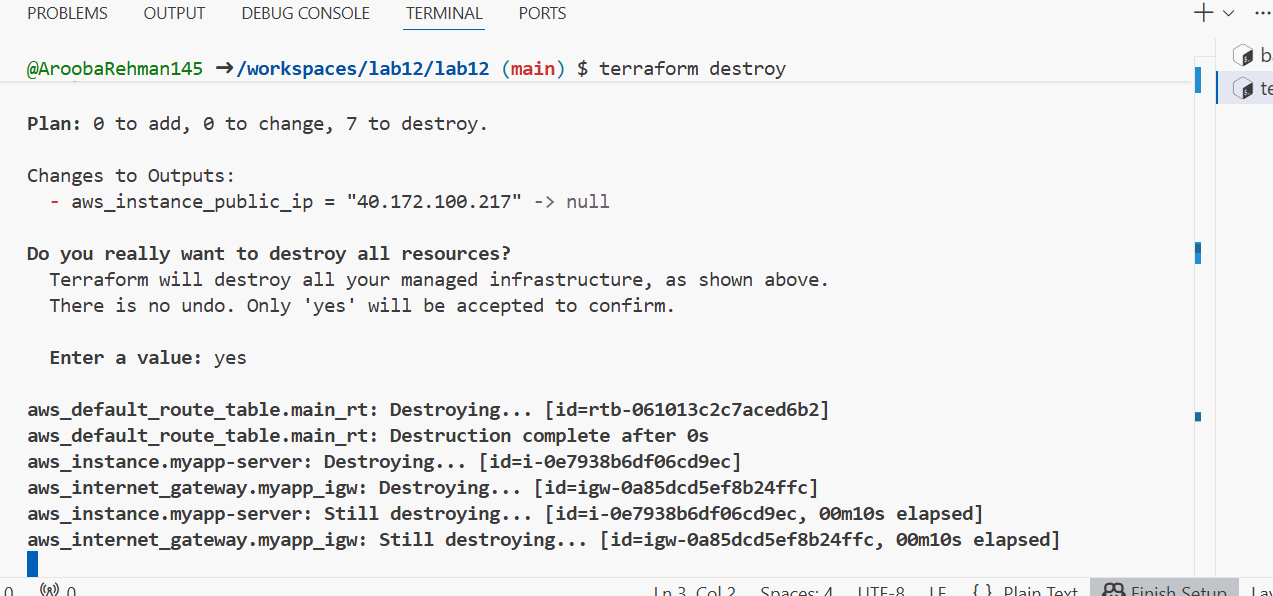
* Open browser and navigate to http://<public-ip>



1. Destroy resources:

terraform destroy

* Type yes when prompted.



**TASK 6:**

## **Configure HTTPS with self-signed certificates**

In this task, you will configure Nginx to serve traffic over HTTPS using self-signed certificates.

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

#!/bin/bash

set -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.conf

user 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



1. Apply the configuration:

terraform apply -auto-approve

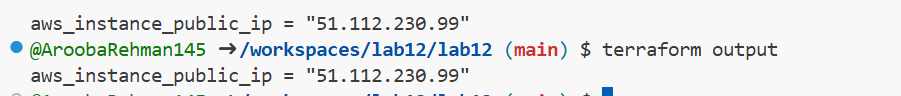
as error encountered:





1. Display the output:

terraform output



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"

Corrected root main.tf code:

provider "aws" {

# Authentication is handled via terminal export commands

}

# Foundation VPC

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

cidr\_block = var.vpc\_cidr\_block

tags = {

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

}

}

# Subnet Module Call

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

}

# Webserver Module Call (Task 6 HTTPS Configuration)

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"

}

# Data source for SSH security

data "http" "my\_ip" {

url = "https://icanhazip.com"

}

Webserver main.tf:

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

vpc\_id = var.vpc\_id

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

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = [var.my\_ip]

}

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

# Port 443 is required for Task 6 HTTPS

ingress {

from\_port = 443

to\_port = 443

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"]

}

}

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"

instance\_type = var.instance\_type

subnet\_id = var.subnet\_id

vpc\_security\_group\_ids = [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

# Reliable script execution for Task 6

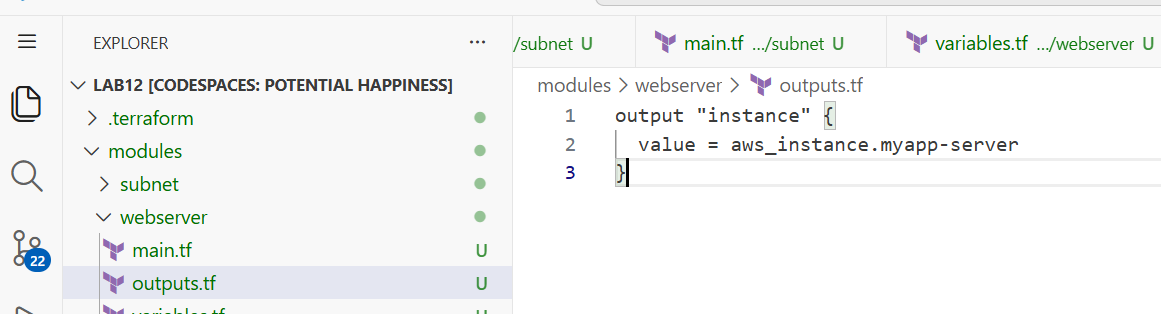
user\_data = file(var.script\_path)

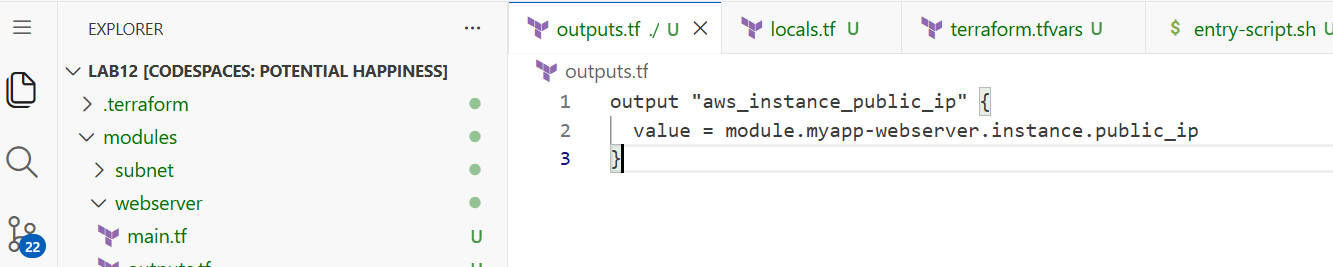
tags = {

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

}

}





Entry.script:

#!/bin/bash

set -e

# Update and Install Nginx

yum update -y

yum install -y nginx

# SSL setup

mkdir -p /etc/ssl/private /etc/ssl/certs

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

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

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"

# Overwrite Nginx config with valid syntax

cat <<EOF > /etc/nginx/nginx.conf

user nginx;

worker\_processes auto;

events { worker\_connections 1024; }

http {

    include /etc/nginx/mime.types;

    server {

        listen 443 ssl;

        server\_name \_;

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

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

        location / {

            root /usr/share/nginx/html;

            index index.html;

        }

    }

    server {

        listen 80;

        server\_name \_;

        return 301 https://\$host\$request\_uri;

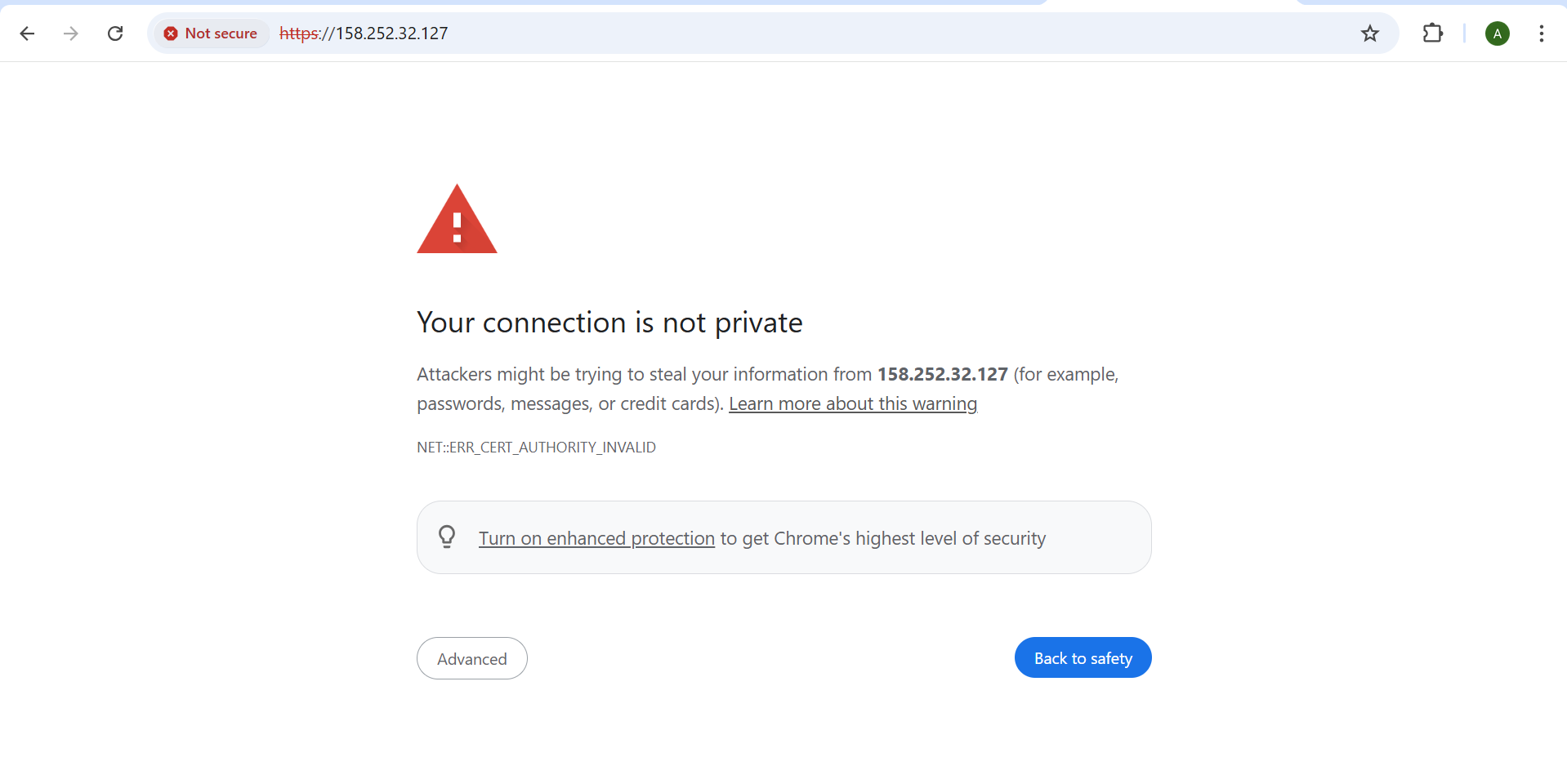
    }

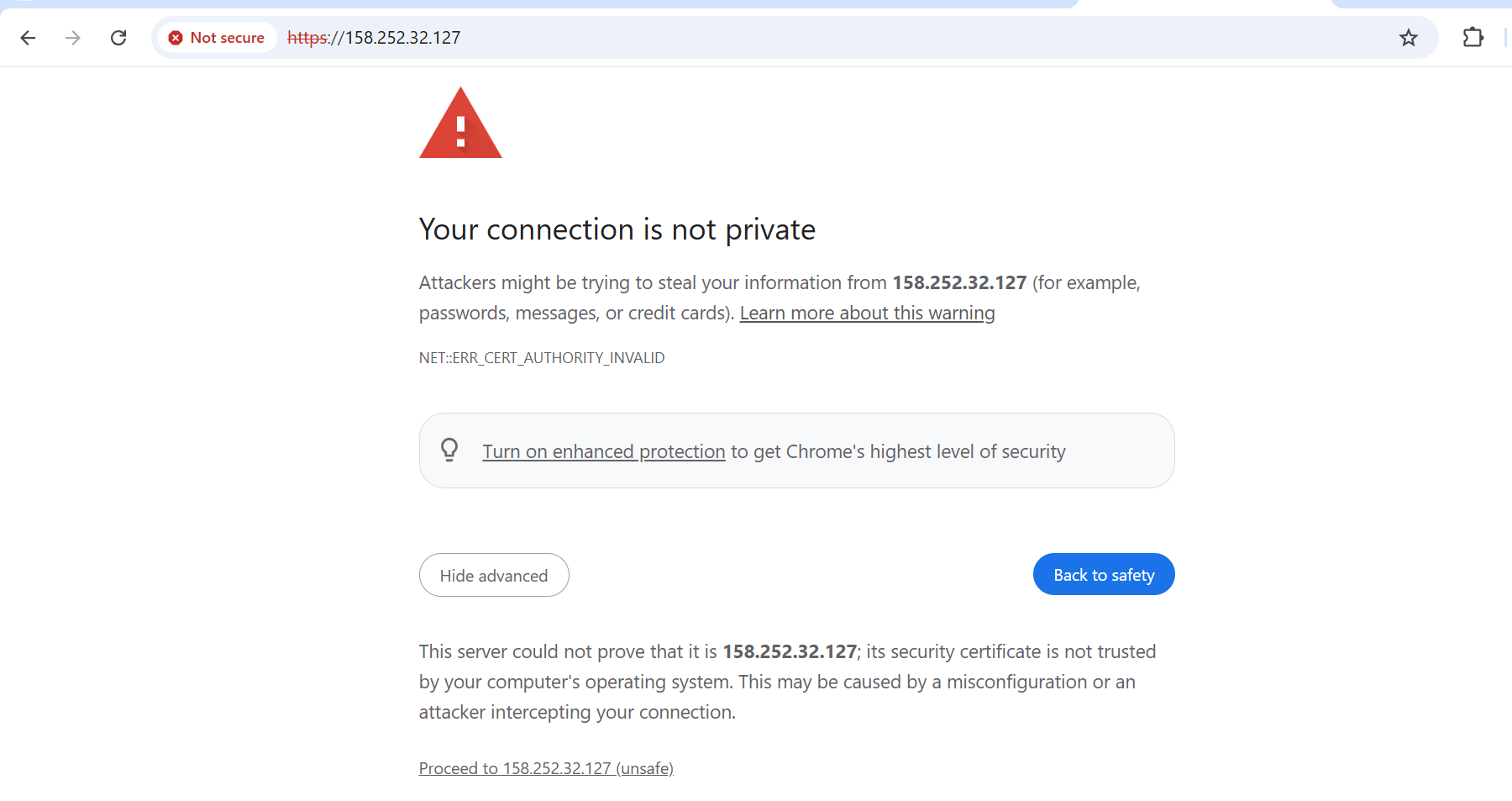
}

EOF

systemctl enable nginx

systemctl restart nginx







1. Verify HTTP to HTTPS redirect:

* Open browser and navigate to http://<public-ip>
* Verify it redirects to https://<public-ip>

## **Task 7**

## **Configure Nginx as reverse proxy**

In this task, you will create a backend web server and configure Nginx to act as a 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 httpd

echo "<h1>Welcome to My Web Server</h1>" > /var/www/html/index.html

hostnamectl set-hostname myapp-webserver

echo "<h2>Hostname: $(hostname)</h2>" >> /var/www/html/index.html

TOKEN=$(curl -s -X PUT "http://169.254.169.254/latest/api/token" \

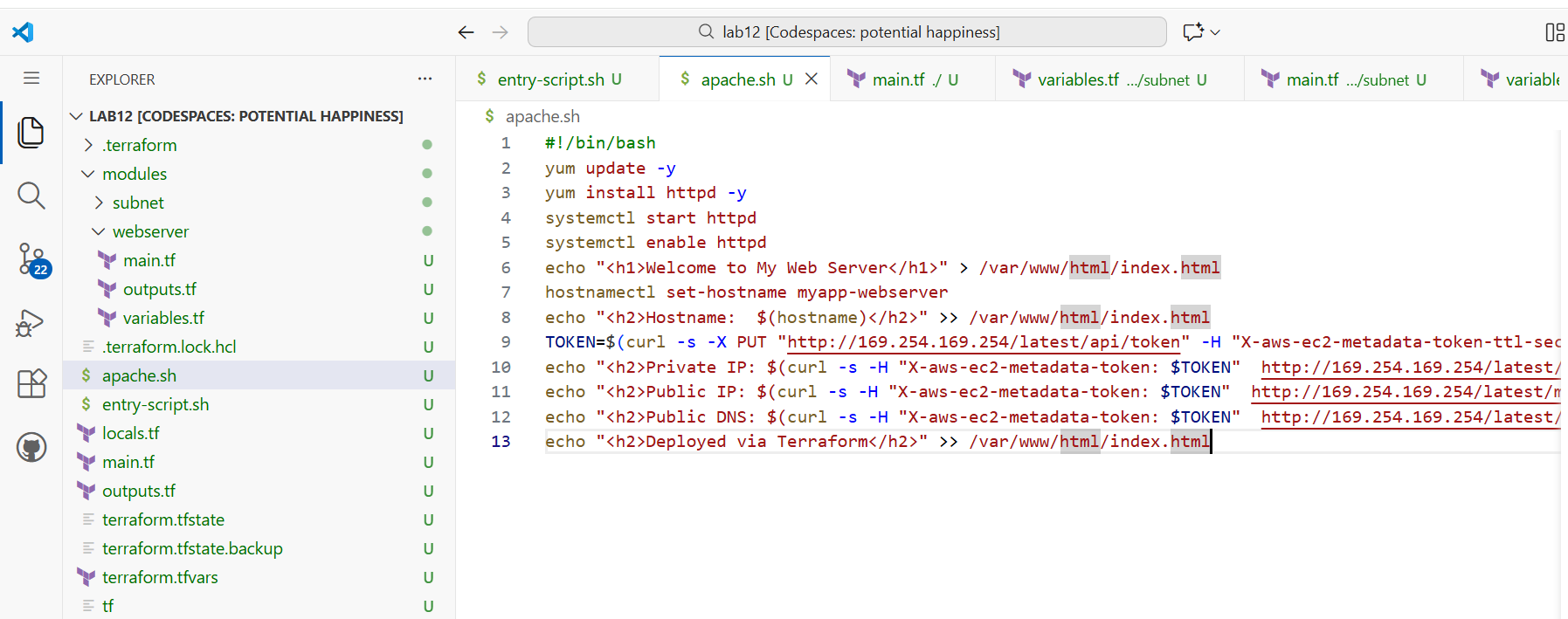
-H "X-aws-ec2-metadata-token-ttl-seconds: 21600")

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.html

echo "<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.html

echo "<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.html

echo "<h2>Deployed via Terraform</h2>" >> /var/www/html/index. html



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"

}

provider "aws" {

# Credentials handled by 'export' commands in terminal

}

# 1. Foundation VPC

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

cidr\_block = var.vpc\_cidr\_block

tags = {

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

}

}

# 2. Subnet Module - Handles IGW and Route Tables

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

}

# 3. Task 6: Nginx Proxy Server (Instance Suffix 0)

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" # Task 6 HTTPS Script

instance\_suffix = "0"

}

# 4. Task 7: Apache Backend Server (Instance Suffix 1)

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" # Task 7 Apache Script

instance\_suffix = "1"

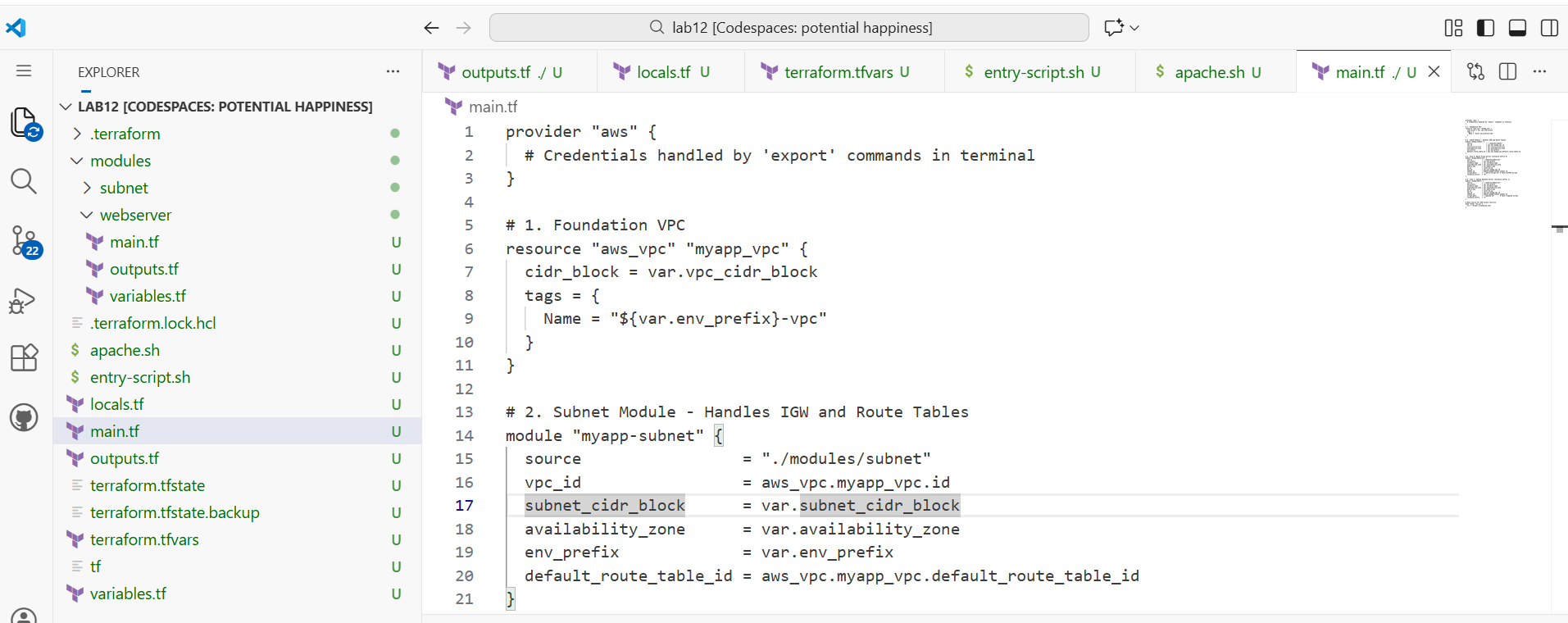
}

# Data source for SSH access security

data "http" "my\_ip" {

url = "https://icanhazip.com"

}



1. Update outputs.tf:

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

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

}

output "aws\_instance\_public\_ip" {

description = "Public IP of the Nginx Proxy Server"

value = module.myapp-webserver.instance.public\_ip

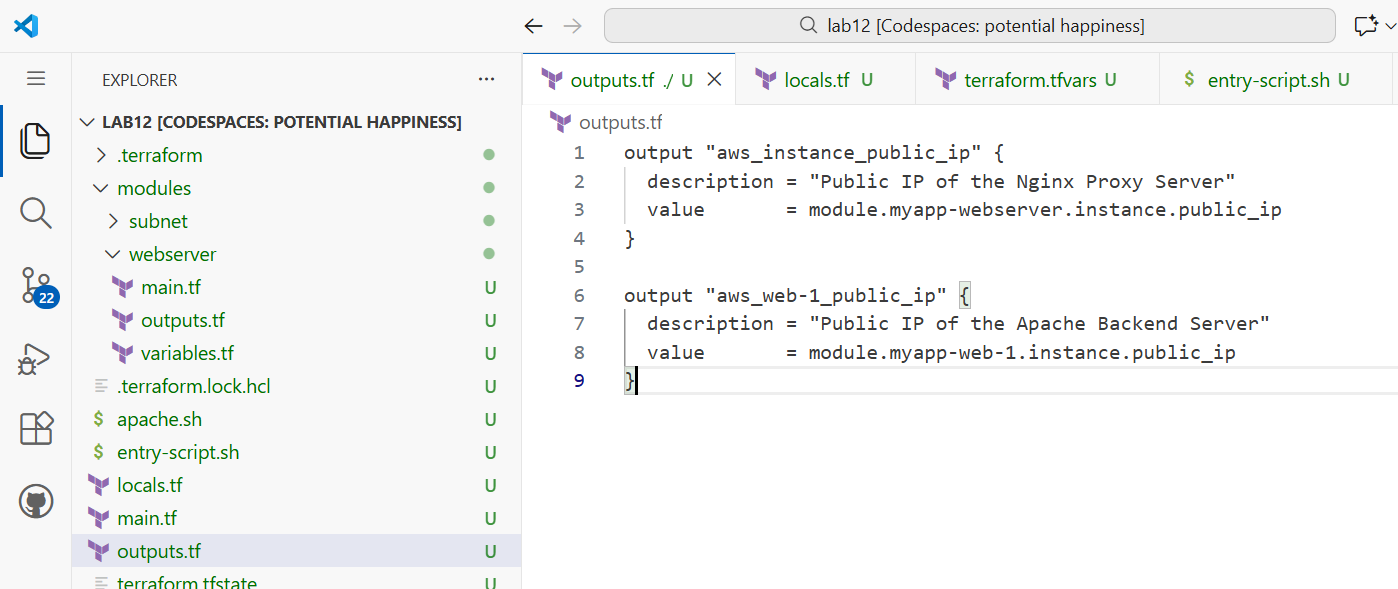
}

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

description = "Public IP of the Apache Backend Server"

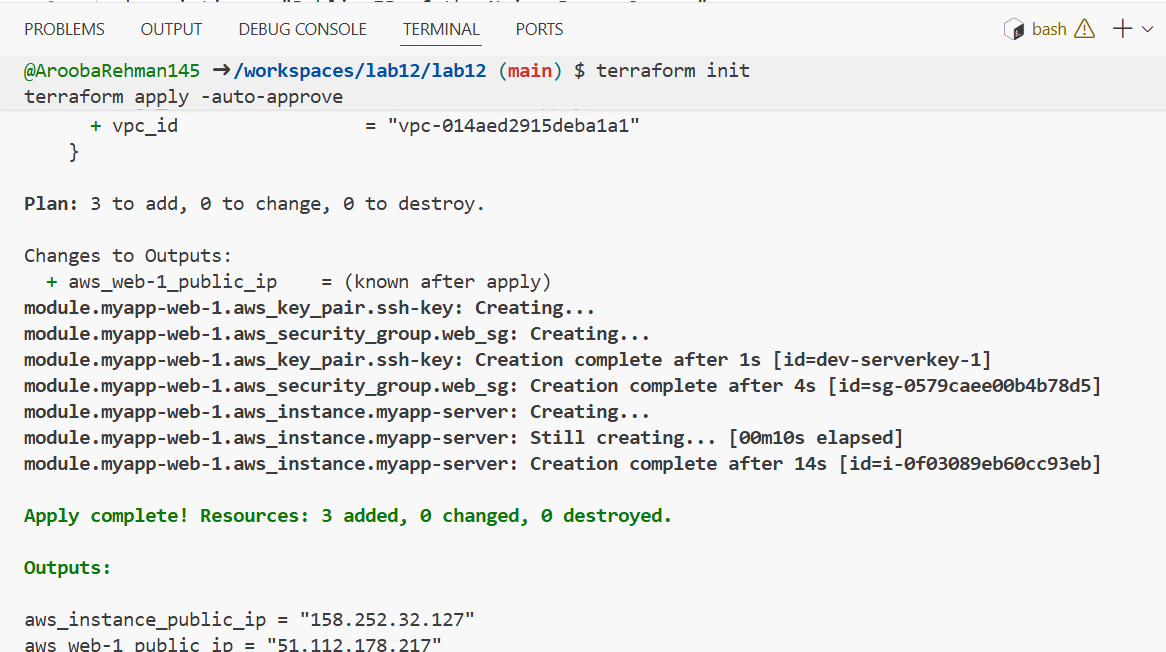
value = module.myapp-web-1.instance.public\_ip

}



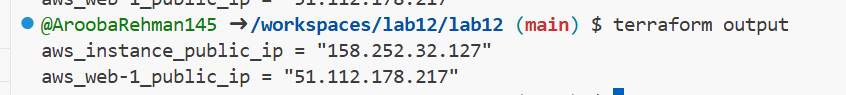
1. Apply the configuration:

terraform apply -auto-approve



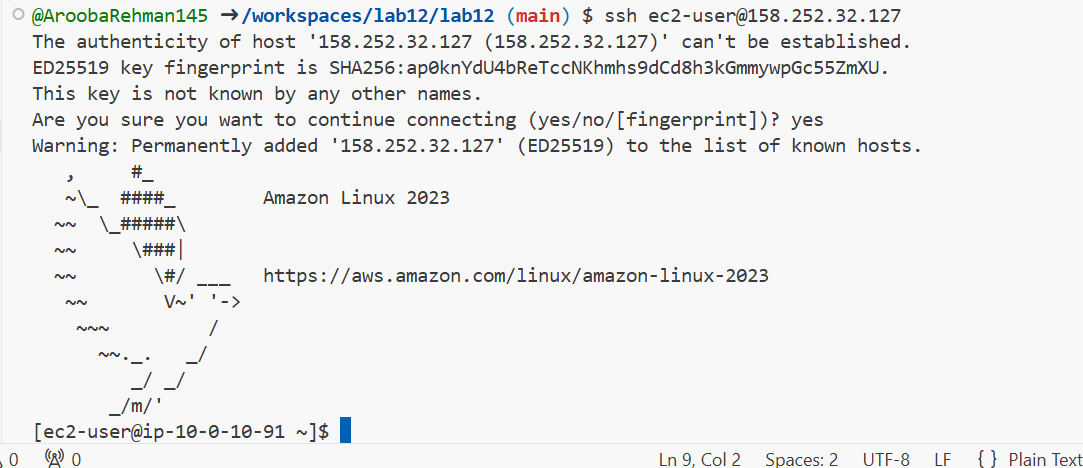
1. Get the outputs:

terraform output



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

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



1. Edit the Nginx configuration:

sudo vim /etc/nginx/nginx.conf

Modify the location block to proxy to web-1:

location / {

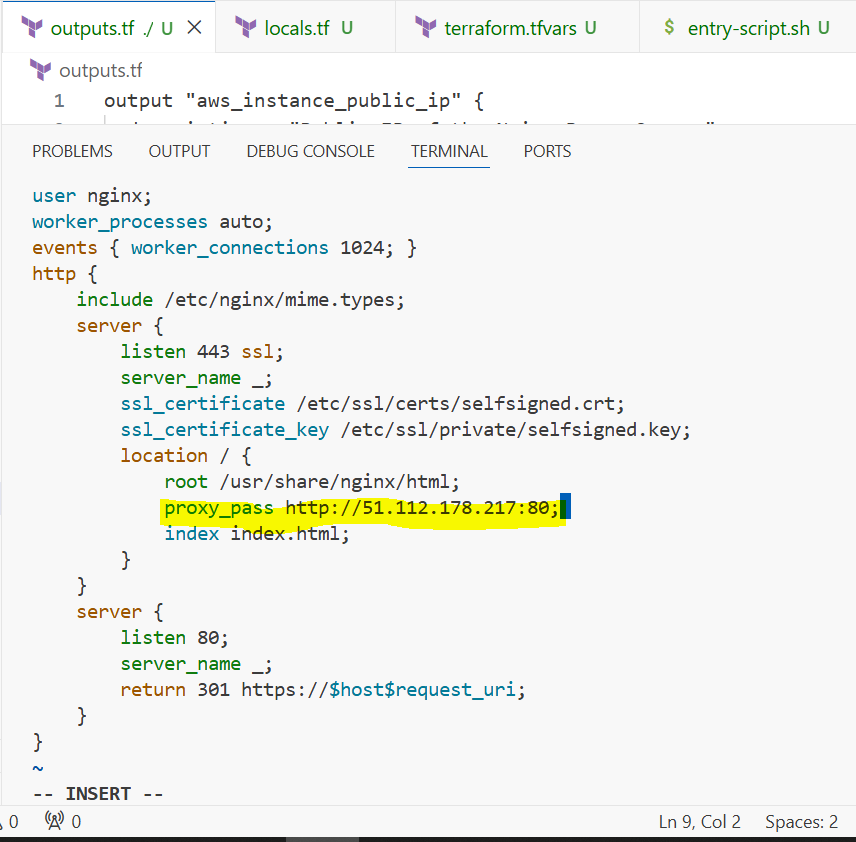
# root /usr/share/nginx/html;

# index index. html;

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

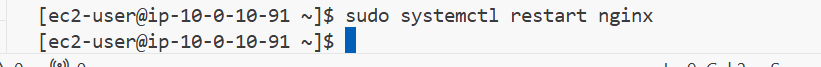
# proxy\_pass http://backend\_servers;

}



1. Restart Nginx:

sudo systemctl restart nginx

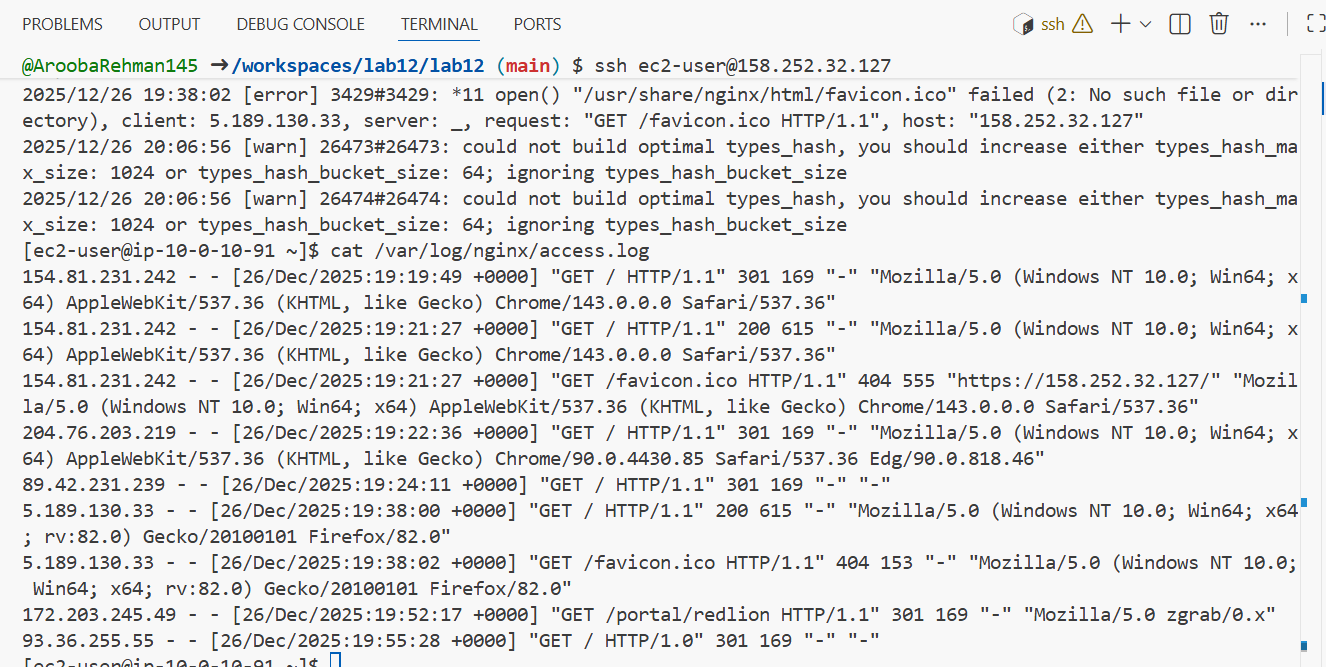


1. View Nginx logs and configuration files:

cat /var/log/nginx/error.log



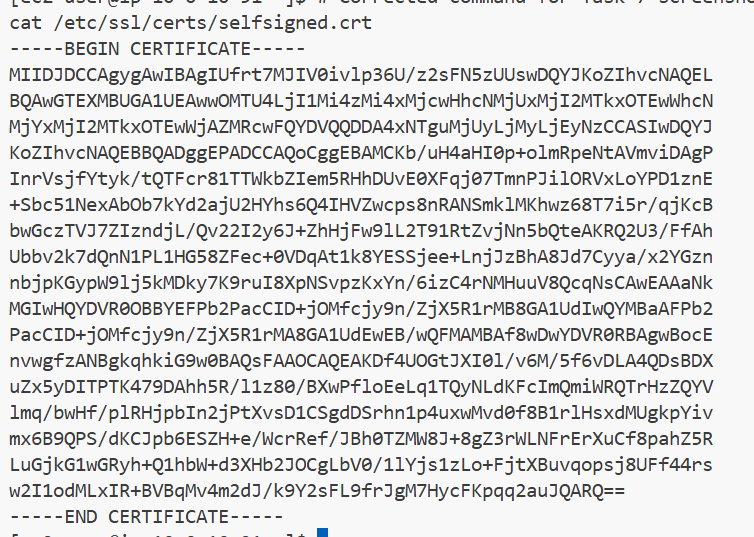
cat /var/log/nginx/access.log



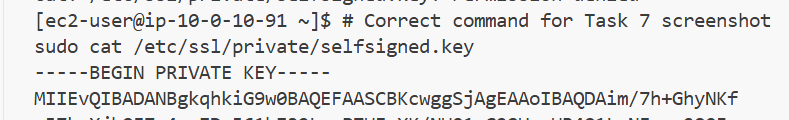
cat /etc/nginx/mime.types

* 

cat /etc/ssl/certs/selfsigned. crt



cat /etc/ssl/private/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



## **Task 8**

## **Configure Nginx as load balancer**

In this task, you will add a second backend server and configure Nginx to load balance between them.

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

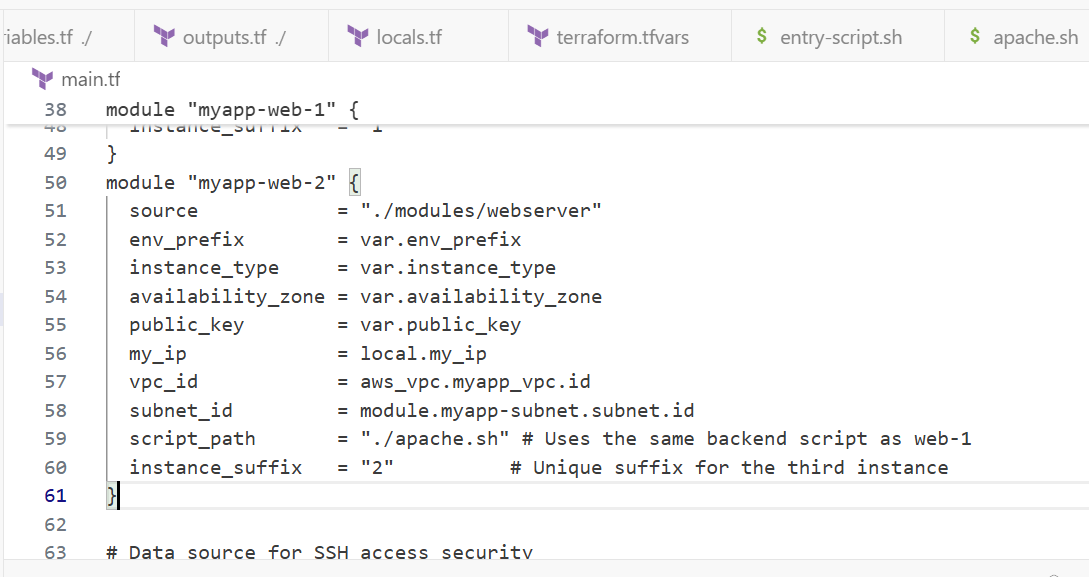
vpc\_id = aws\_vpc.myapp\_vpc. id

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

script\_path = "./apache.sh"

instance\_suffix = "2"

}

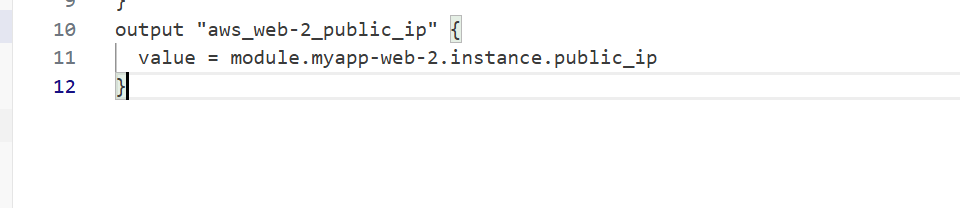


1. Update outputs.tf:

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

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

}



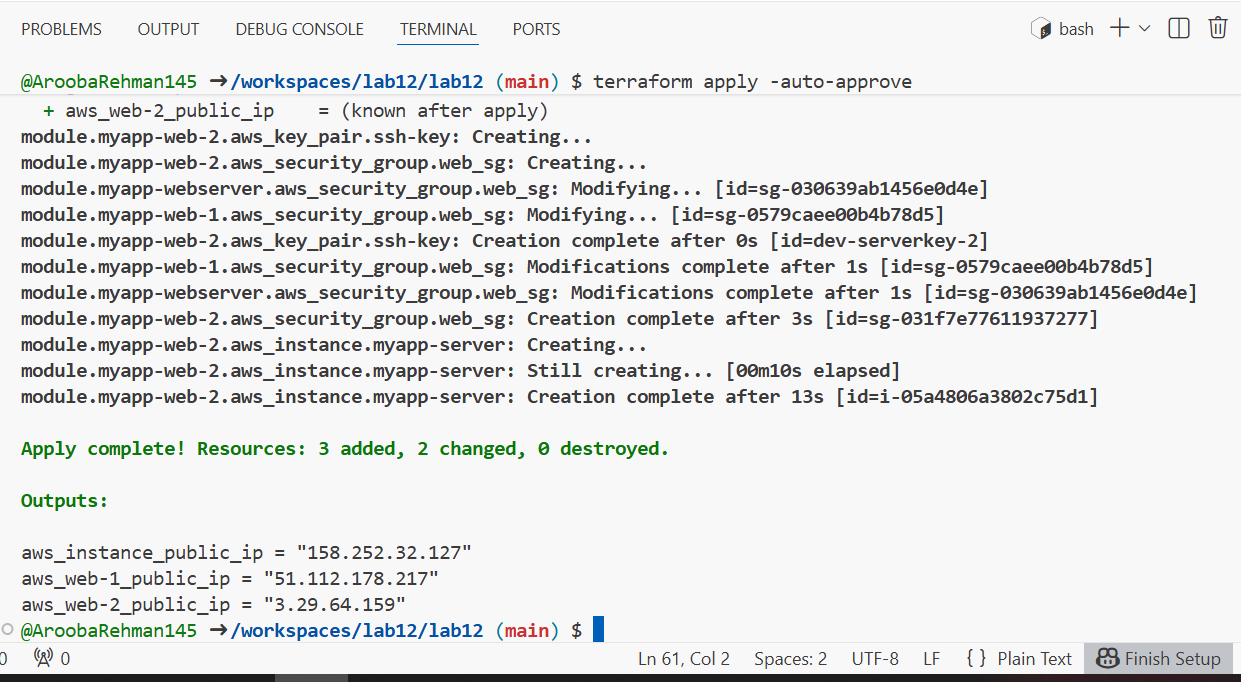
output "aws\_web-2\_public\_ip" {

value = module.myapp-web-2.instance.public\_ip

}

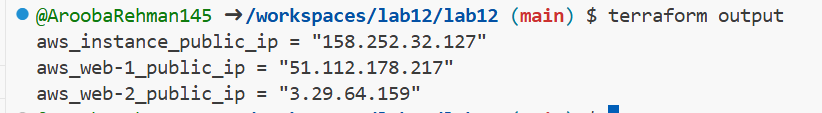
1. Apply the configuration:

terraform apply -auto-approve



1. Get all outputs:

terraform output



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

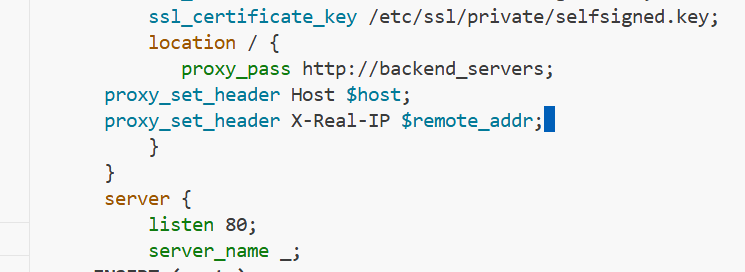


upstream backend\_servers {

server 51.112.178.217:80; # web-1 IP

server <3.29.64.159>:80; # web-2 IP from your terraform output

}



location / {

proxy\_pass http://backend\_servers;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

}

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 / {

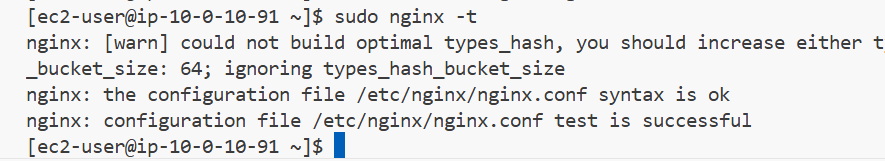
# root /usr/share/nginx/html;

# index index.html;

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

proxy\_pass http://backend\_servers;

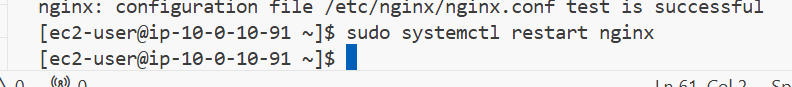
}



Sudo nginx -t

1. Restart Nginx:

sudo systemctl restart nginx

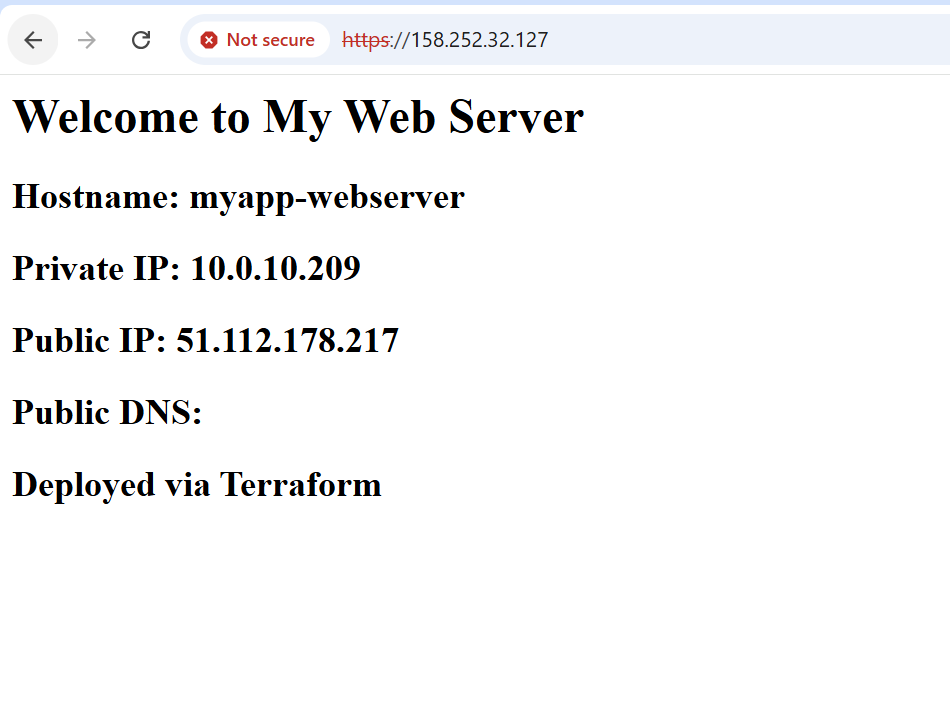


Sudo systemctl restart nginx

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)





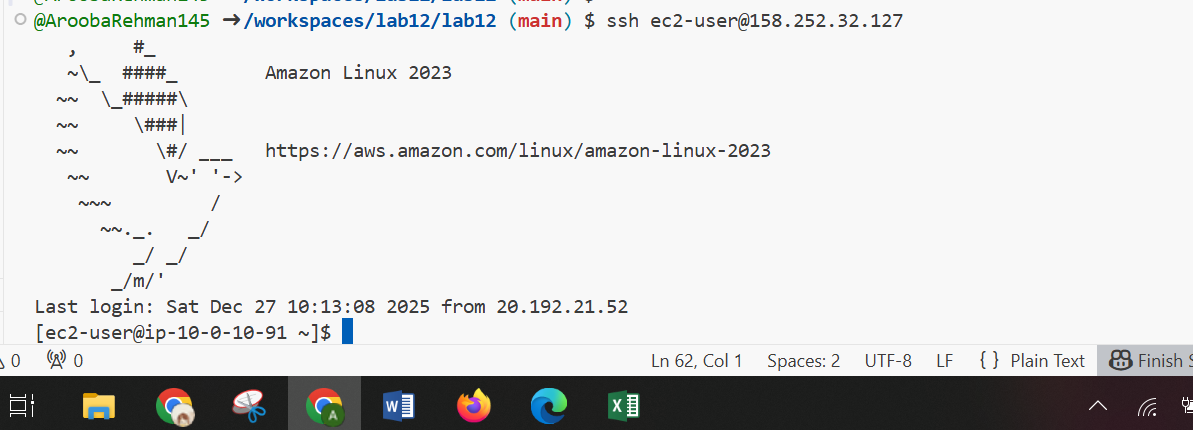
## **Task 9** Configure high availability with backup servers

In this task, you will configure one server as primary and another as backup for high availability.

1. SSH into the webserver:

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

ssh [ec2-user@158.252.32.127](mailto:ec2-user@158.252.32.127)



1. Edit Nginx configuration for high availability:

sudo vim /etc/nginx/nginx.conf

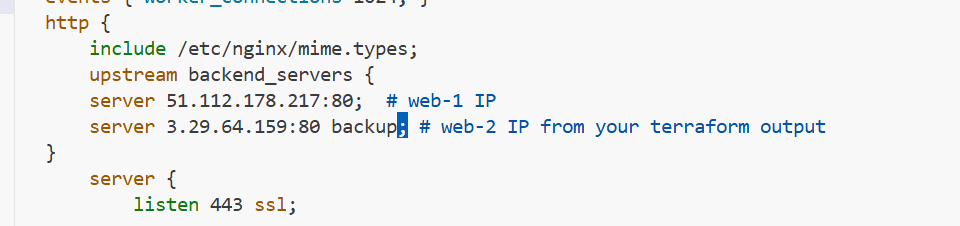
Update the upstream block to make web-2 a backup:

upstream backend\_servers {

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

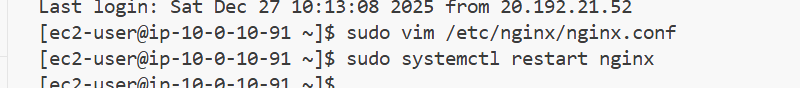
server <web-2-public-ip>:80 backup;

}



1. Restart Nginx:

sudo systemctl restart nginx



1. Test in browser:

* Open browser and navigate to https://<webserver-public-ip>
* Reload multiple times
* You should ONLY see web-1 (primary server)



1. Switch backup configuration:

sudo vim /etc/nginx/nginx.conf

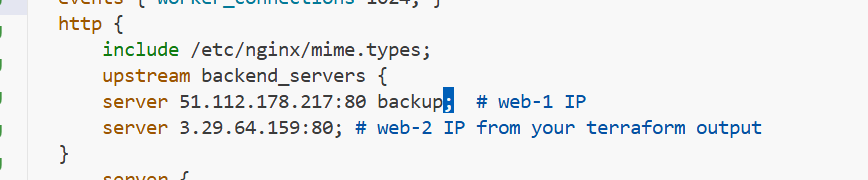
Update to make web-1 backup:

upstream backend\_servers {

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

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

}



1. Restart Nginx:

sudo systemctl restart nginx

1. Test in browser:

* Reload multiple times
* You should ONLY see web-2 (now the primary server)



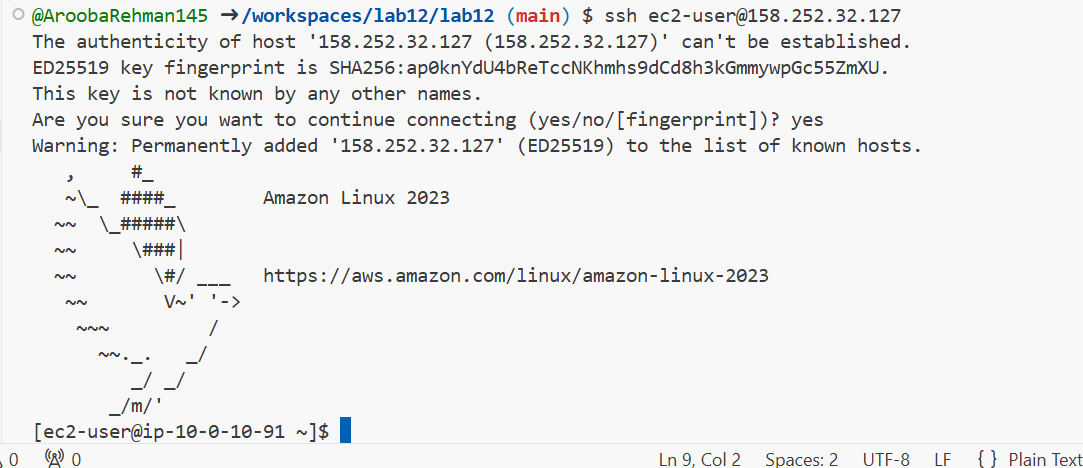
## **Task 10**

## Enable Nginx caching

In this task, you will enable caching in Nginx to improve performance.

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";

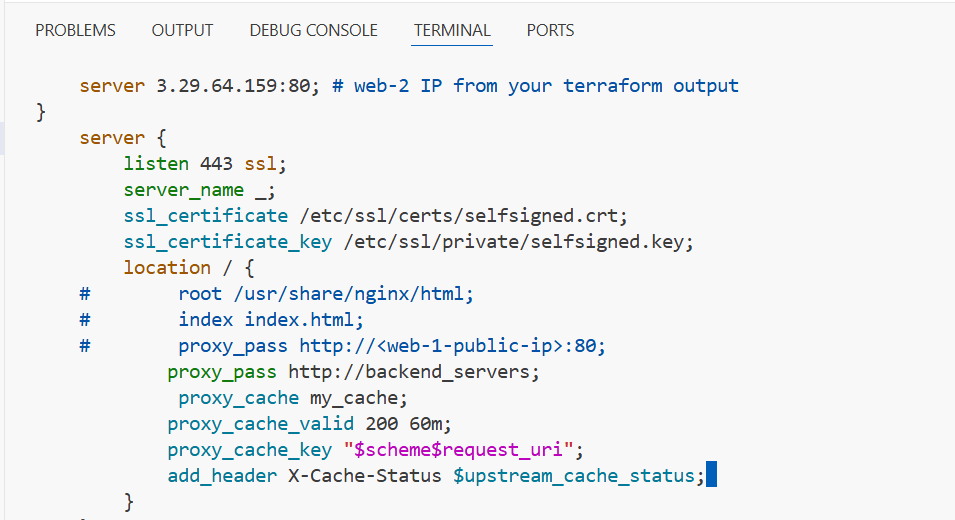
add\_header X-Cache-Status $upstream\_cache\_status;

}

}

# ... rest of config ...

}



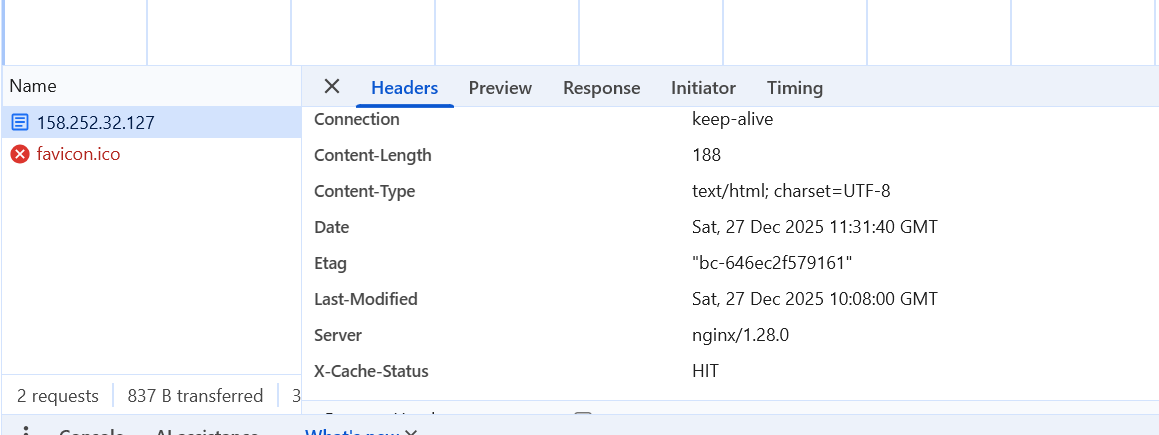


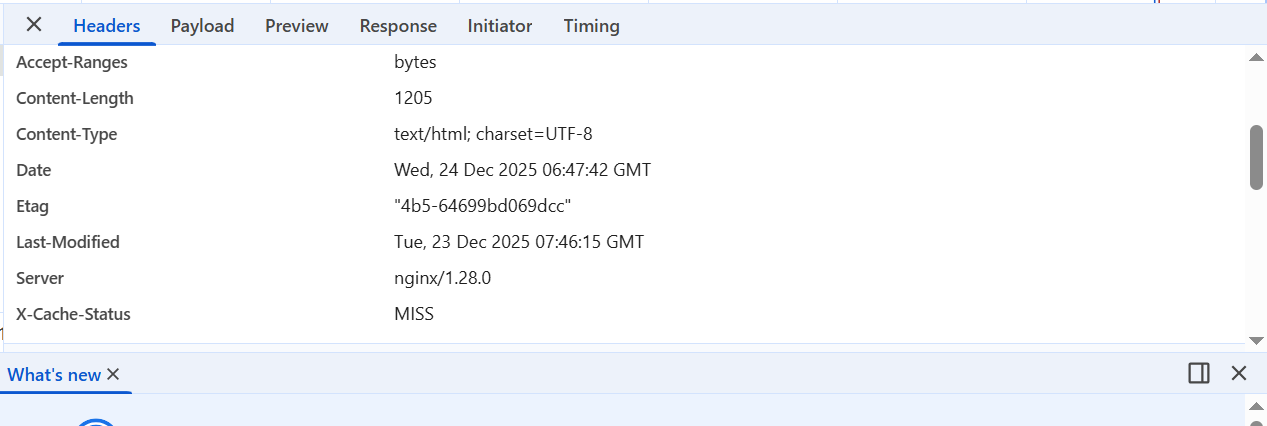
1. Restart Nginx:

sudo systemctl restart nginx

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





1. Verify cache directory:

ls -la /var/cache/nginx/

