Introduction of terraform

* Terraform is an infrastructure-as-code software tool created by HashiCorp.
* Users define and provide data center infrastructure using a declarative configuration language known as HashiCorp Configuration Language(HCL), or optionally JSON.
* HashiCorp maintains a Terraform Module Registry
* If we create any resource in AWS. It may create automatically by the using hashicop language.
* [Original author(s)](https://www.bing.com/ck/a?!&&p=7b7914054a021ab1JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0OA&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPVRlcnJhZm9ybStvcmlnaW5hbCthdXRob3IocykmZmlsdGVycz1zaWQ6Ijg0NzkwYjUzLTkzOTEtNmNjMS01MmEyLWY2MzdlYTQxZTEwZCI&ntb=1) Mitchell Hashimoto et al.
* [Developer(s)](https://www.bing.com/ck/a?!&&p=d2e36f100eb15ee0JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0OQ&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPVRlcnJhZm9ybStkZXZlbG9wZXIocykmZmlsdGVycz1zaWQ6Ijg0NzkwYjUzLTkzOTEtNmNjMS01MmEyLWY2MzdlYTQxZTEwZCI&ntb=1) [HashiCorp](https://www.bing.com/ck/a?!&&p=dd30fd53993adf93JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc1MA&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPUhhc2hpQ29ycCZmaWx0ZXJzPXNpZDoiNjc3MDU1YjUtYWZlZS1mNzBkLWUyMjQtNjlkYTI4MDMyYWYxIg&ntb=1)
* [Initial release](https://www.bing.com/ck/a?!&&p=c5f69b29f3bd0e6dJmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc1MQ&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPVRlcnJhZm9ybStpbml0aWFsK3JlbGVhc2UmZmlsdGVycz1zaWQ6Ijg0NzkwYjUzLTkzOTEtNmNjMS01MmEyLWY2MzdlYTQxZTEwZCI&ntb=1) 28 July 2014
* [Stable release](https://www.bing.com/ck/a?!&&p=6a86866fc55155cfJmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc1Mg&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPVRlcnJhZm9ybStzdGFibGUrcmVsZWFzZSZmaWx0ZXJzPXNpZDoiODQ3OTBiNTMtOTM5MS02Y2MxLTUyYTItZjYzN2VhNDFlMTBkIg&ntb=1) 1.8.0 / 10 April 2024
* [Repository](https://www.bing.com/ck/a?!&&p=808b3ffd13af00a0JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTczNg&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPVRlcnJhZm9ybStyZXBvc2l0b3J5JmZpbHRlcnM9c2lkOiI4NDc5MGI1My05MzkxLTZjYzEtNTJhMi1mNjM3ZWE0MWUxMGQi&ntb=1) [github.com/hashicorp/terraform](https://www.bing.com/ck/a?!&&p=5c81efd3e9340513JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTczNw&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&psq=terraform+&u=a1aHR0cHM6Ly9naXRodWIuY29tL2hhc2hpY29ycC90ZXJyYWZvcm0&ntb=1)
* [Written in](https://www.bing.com/ck/a?!&&p=a09ade7d964b88d9JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTczOA&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPVRlcnJhZm9ybSt3cml0dGVuK2luJmZpbHRlcnM9c2lkOiI4NDc5MGI1My05MzkxLTZjYzEtNTJhMi1mNjM3ZWE0MWUxMGQi&ntb=1) [Go](https://www.bing.com/ck/a?!&&p=4d325990153b2680JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTczOQ&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPUdvJmZpbHRlcnM9c2lkOiIyN2JiZGYzNi0xZDNiLWYxZWYtMTcxNy0yZGJlYTk3N2M0NjEi&ntb=1)
* [Operating system](https://www.bing.com/ck/a?!&&p=cc0d702c00646045JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0MA&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPVRlcnJhZm9ybStvcGVyYXRpbmcrc3lzdGVtJmZpbHRlcnM9c2lkOiI4NDc5MGI1My05MzkxLTZjYzEtNTJhMi1mNjM3ZWE0MWUxMGQi&ntb=1)[Linux](https://www.bing.com/ck/a?!&&p=b82ce7d2632d7be8JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0MQ&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPUxpbnV4JmZpbHRlcnM9c2lkOiJkMmY0ZTkzMy1iZmFkLWYxODQtZmNmNC0yMDA0MmU0YmVkMzYi&ntb=1), [FreeBSD](https://www.bing.com/ck/a?!&&p=750a23c9b201d935JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0Mg&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPUZyZWVCU0QmZmlsdGVycz1zaWQ6Ijc5ZjhiMzI3LTJjMzItYmI4OC0wZDcwLTVmNDFmNDA0N2U3OSI&ntb=1), [macOS](https://www.bing.com/ck/a?!&&p=200baec361027066JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0Mw&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPW1hY09TJmZpbHRlcnM9c2lkOiJhMGQyNjhlZi04MGRjLTczZTQtOWZmOS05OGIzYTIzMzIxOGEi&ntb=1), [OpenBSD](https://www.bing.com/ck/a?!&&p=c7f8ea24955b82c0JmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0NA&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPU9wZW5CU0QmZmlsdGVycz1zaWQ6IjhlZWUzNWMxLTdhYzMtYzYyMC05NTY1LTU1OTc2ZmU4ZWM2ZSI&ntb=1), [Solaris](https://www.bing.com/ck/a?!&&p=6d41bd664b554bacJmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0NQ&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&psq=terraform+&u=a1aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvU29sYXJpc18ob3BlcmF0aW5nX3N5c3RlbSk&ntb=1), and [Microsoft Windows](https://www.bing.com/ck/a?!&&p=bbe64dd724d3d5bbJmltdHM9MTcxMzIyNTYwMCZpZ3VpZD0wZDgxZDcxNy04NzBiLTY4NTUtMjcxNi1jNGM1ODY1OTY5OTAmaW5zaWQ9NTc0Ng&ptn=3&ver=2&hsh=3&fclid=0d81d717-870b-6855-2716-c4c586596990&u=a1L3NlYXJjaD9GT1JNPVNOQVBTVCZxPU1pY3Jvc29mdCtXaW5kb3dzJmZpbHRlcnM9c2lkOiIxNmFlYjZkOS05MDk4LTBhNDAtNDk3MC04ZTQ2YTRmY2VlMTIi&ntb=1)

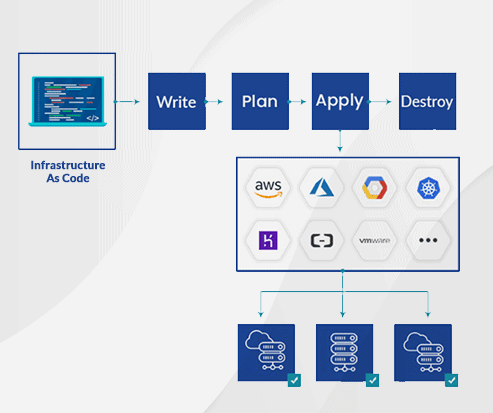
Terraform providers:

1. AWS
2. Microsoft Azure
3. Oracle
4. Alibaba
5. MongoDB
6. Google
7. Zscaler

Workflow for terraform:-

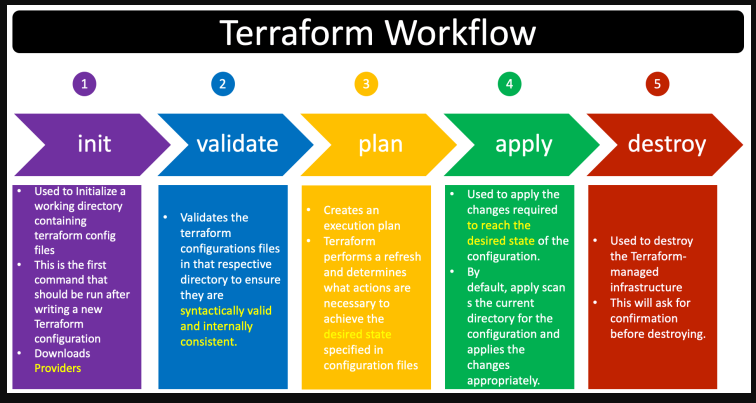
1. Workflow
2. Execution flow

Terraform workflow:



Terraform execution flow :

Init🡪format🡪validate🡪plan🡪apply🡪destroy



* Here, if we create any file we have to give .tf extension. There is no need to give any alignment terraform will take care about the alignment

Advantages of Terraform

1. Declarative configuration
2. Support for multi providers
3. Reusable infrastructure code
4. Collaboration and version
5. **Efficient Resource Management**

## Disadvantages of Terraform

## Complexity

## State management

## Performance

## Limited Error Handling

## Limited rollback capabilities

## Terraform uses:

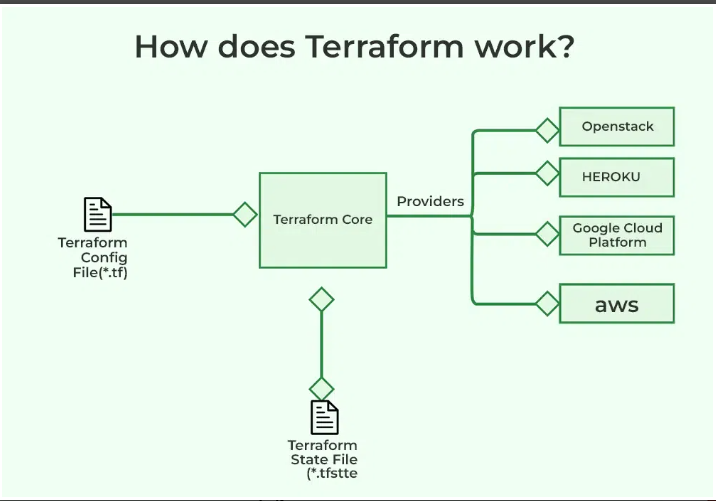
Terraform will automate the proceses of automating the infrastructure instead of doing it manually and also following are the some reasons Terraform is used in AWS

1. Consistency
2. Reproducibility
3. Efficiency
4. Flexibility

## Terraform state file(terraform.tfstate)

## The state file is used to track the resources that have been created, modified, or destroyed, and it is used to ensure that the infrastructure resources match the desired state defined in the configuration files.

* Terraform uses this state to plan and execute changes, as well as to update and manage your infrastructure.
* Were as terraform.tf state.backup file will help us to take a backup for deleted resources script.



Go with a project :

3 Tier application deployment

In this 3 tier application we have 3 tiers

1) GUI (Frontend tier)

2) Processing tier (Mid tier)

3) Database tier (Backend)

* Firstly, we have to create a ec2 instance in any region & update the server
* Using below commands we have to install the Terraform package file and terrafrom to that server

sudo yum install -y yum-utils shadow-utils

sudo yum-config-manager --add-repo https://rpm.releases.hashicorp.com/AmazonLinux/hashicorp.repo

sudo yum -y install terraform

* STEP: 01

Creation of provider file using terraform(HCL-script):

Created the provider.tf file using vi provider.tf & add the script to it

* File name : provider.tf

provider "aws" {

region = "ap-south-1"

access\_key = "AKIAZQ3DORK2ZXAO3CWT"

secret\_key = "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"

}

* STEP: 02

Creation of variables file using terraform(HCL-script):

* Here, First we need to create a variable file(i.e., var.tf) because of each and every needed resources cidr\_block ids are keeping in one file due to easy way of any change in our required cidr\_blocks at any time.
* Created the var.tf file using vi var.tf and add the script to it
* File name : var.tf

# defining CIDR block for VPC

variable "vpc\_cidr" {

default = "10.0.0.0/16"

}

#defining CIDR block for 1st subnet

variable "subnet\_cidr" {

default = "10.0.1.0/24"

}

#defining CIDR block for 2nd subnet

variable "subnet1\_cidr" {

default = "10.0.2.0/24"

}

#defining CIDR block for 3rd subnet

variable "subnet2\_cidr" {

default = "10.0.3.0/24"

}

#defining CIDR block for 4th subnet

variable "subnet3\_cidr" {

default = "10.0.4.0/24"

}

#defining CIDR block for 5th subnet

variable "subnet4\_cidr" {

default = "10.0.5.0/24"

}

#defining CIDR block for 6th subnet

variable "subnet5\_cidr" {

default = "10.0.6.0/24"

}

* After create the var.tf file using vi var.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

[ Terraform init🡪terraform fmt file name 🡪

terraform validate🡪terraform plan🡪terraform apply ]

Terraform itself creates the file using the given hashicorp configuration language-HCL script

* Step : 03

Creation of vpc using terraform(HCL-script):

Created the vpc.tf file and add the script to it

File name : Vpc.tf

#creating a VPC

resource "aws\_vpc" "my\_vpc" {

cidr\_block = var.vpc\_cidr

instance\_tenancy = "default"

tags = {

Name = "MyVpc"

}

}

* After create the vpc.tf file using vi vpc.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

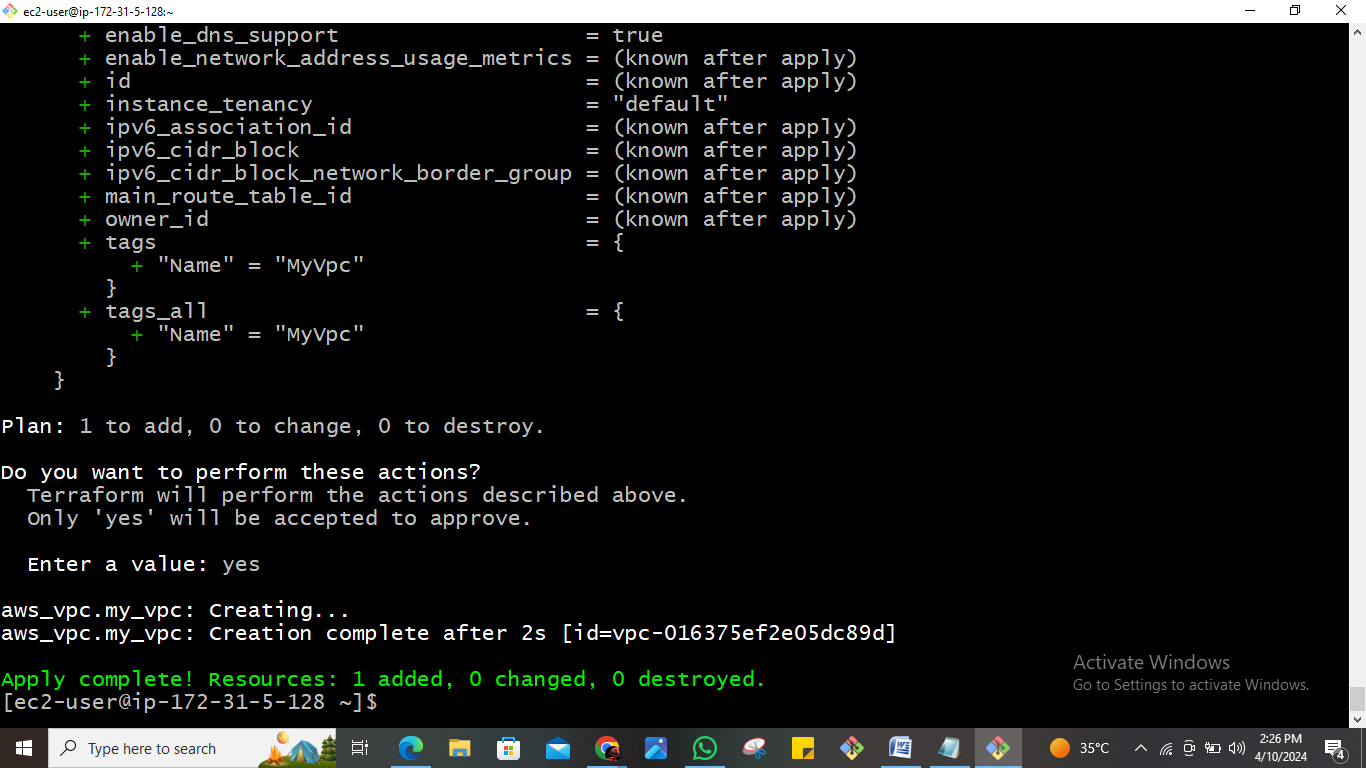
terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

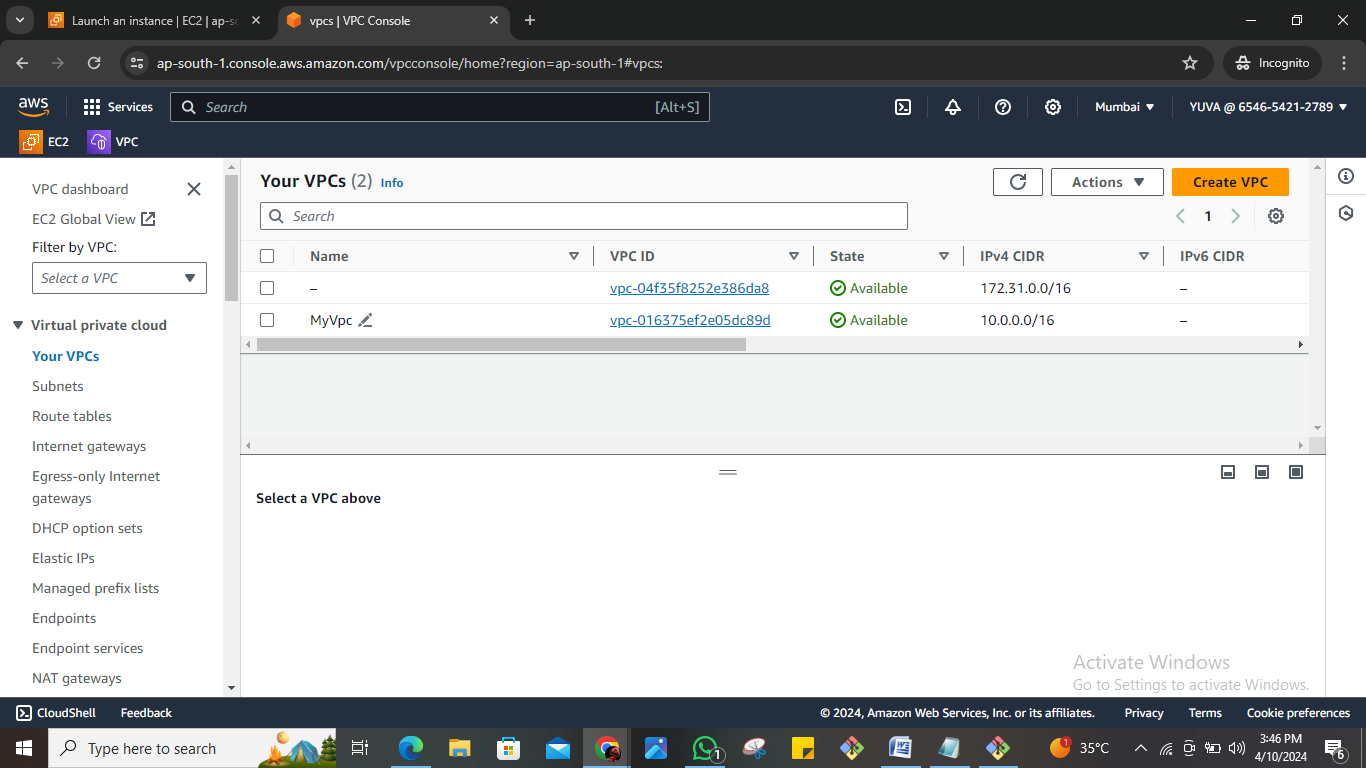
[ Terraform init🡪terraform fmt file name 🡪

terraform validate🡪terraform plan🡪terraform apply ]

Outputs:



* Once, the creation is done we have to check manually in AWS account



* Step : 04

Creation of subnet using terraform(HCL-script):

Note: Here we need to create 6 subnets(2-websubnets,2-application subnets,2-database subnets)

Created the subnet.tf file using vi.subnet.tf and add the script to it

File name: Subnet.tf

# creating 1st web subnet

resource "aws\_subnet" "public-subnet-1" {

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

cidr\_block = var.subnet\_cidr

map\_public\_ip\_on\_launch = true

availability\_zone = "ap-south-1a"

tags = {

Name = "web subnet1"

}

}

# creating 2nd web subnet

resource "aws\_subnet" "public-subnet-2" {

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

cidr\_block = var.subnet1\_cidr

map\_public\_ip\_on\_launch = true

availability\_zone = "ap-south-1b"

tags = {

Name = "web subnet2"

}

}

# creating 1st application subnet

resource "aws\_subnet" "application-subnet-1" {

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

cidr\_block = var.subnet2\_cidr

map\_public\_ip\_on\_launch = false

availability\_zone = "ap-south-1a"

tags = {

Name = "Application subnet1"

}

}

# creating 2nd Application subnet

resource "aws\_subnet" "application-subnet-2" {

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

cidr\_block = var.subnet3\_cidr

map\_public\_ip\_on\_launch = false

availability\_zone = "ap-south-1b"

tags = {

Name = "Application subnet2"

}

}

# creating Database private subnet

resource "aws\_subnet" "Database-subnet-1" {

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

cidr\_block = var.subnet4\_cidr

availability\_zone = "ap-south-1a"

tags = {

Name = "database subnet1"

}

}

# creating Database private subnet

resource "aws\_subnet" "Database-subnet-2" {

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

cidr\_block = var.subnet5\_cidr

availability\_zone = "ap-south-1b"

tags = {

Name = "database subnet2"

}

}

* After create the subnet.tf file using vi subnet.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

terraform fmt # file name

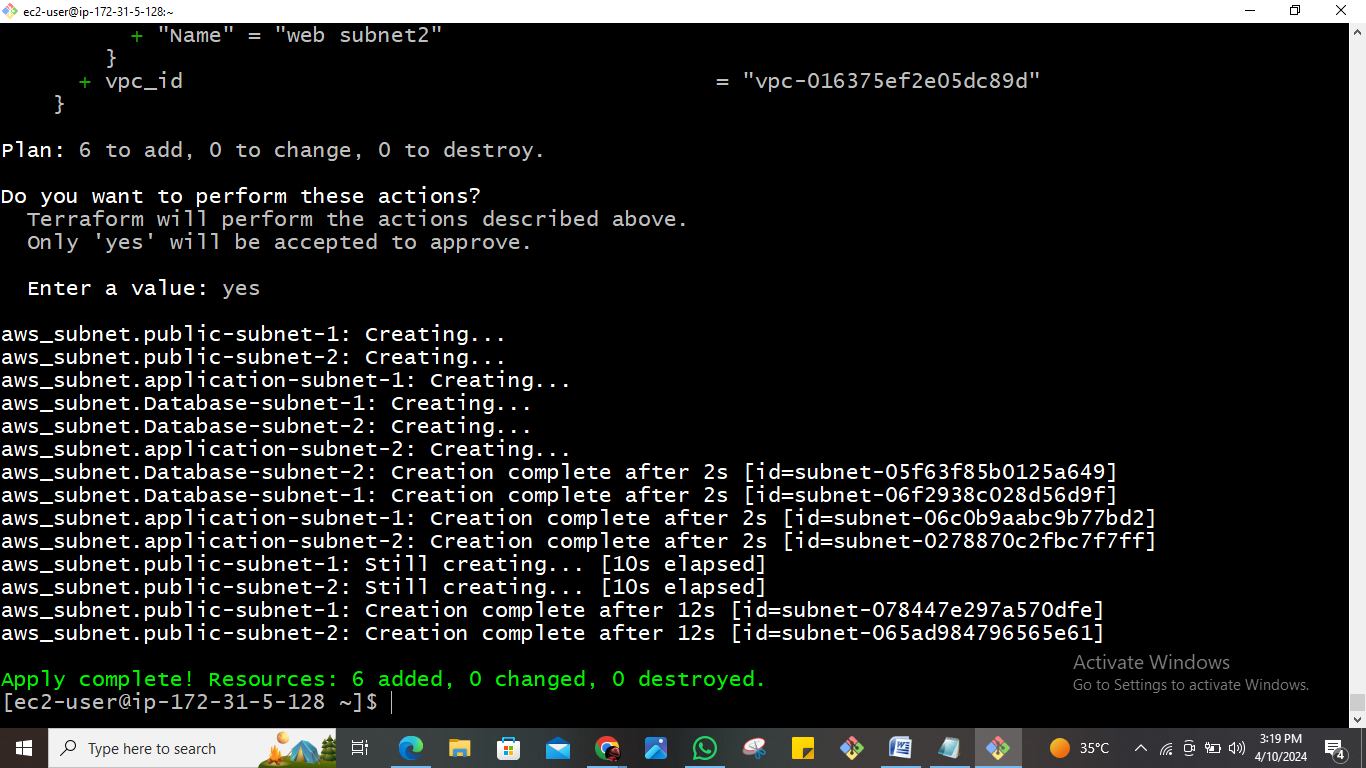
* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

[ Terraform init🡪terraform fmt file name 🡪

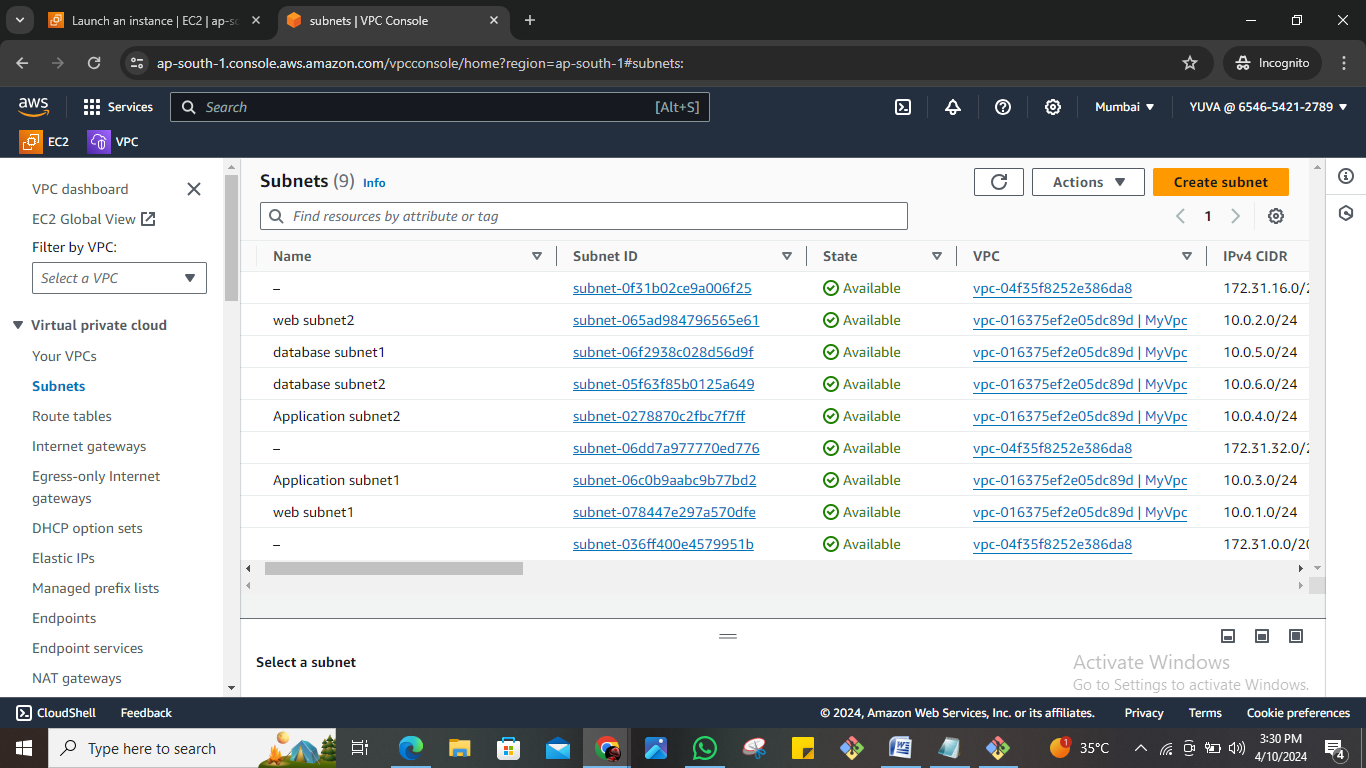
terraform validate🡪terraform plan🡪terraform apply ]

Terraform itself creates the file using the given hashicorp configuration language-HCL script

Outputs:



* Once, the creation is done we have to check manually in AWS account



* Step : 05

Creation of internet gateway using terraform(HCL-script):

Created the igw.tf file using vi igw.tf and add the script to it

File name: igw.tf

# Creating internet gateway

resource "aws\_internet\_gateway" "demogateway" {

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

tags = {

Name = " My igw "

}

}

* After create the igw.tf file using vi igw.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

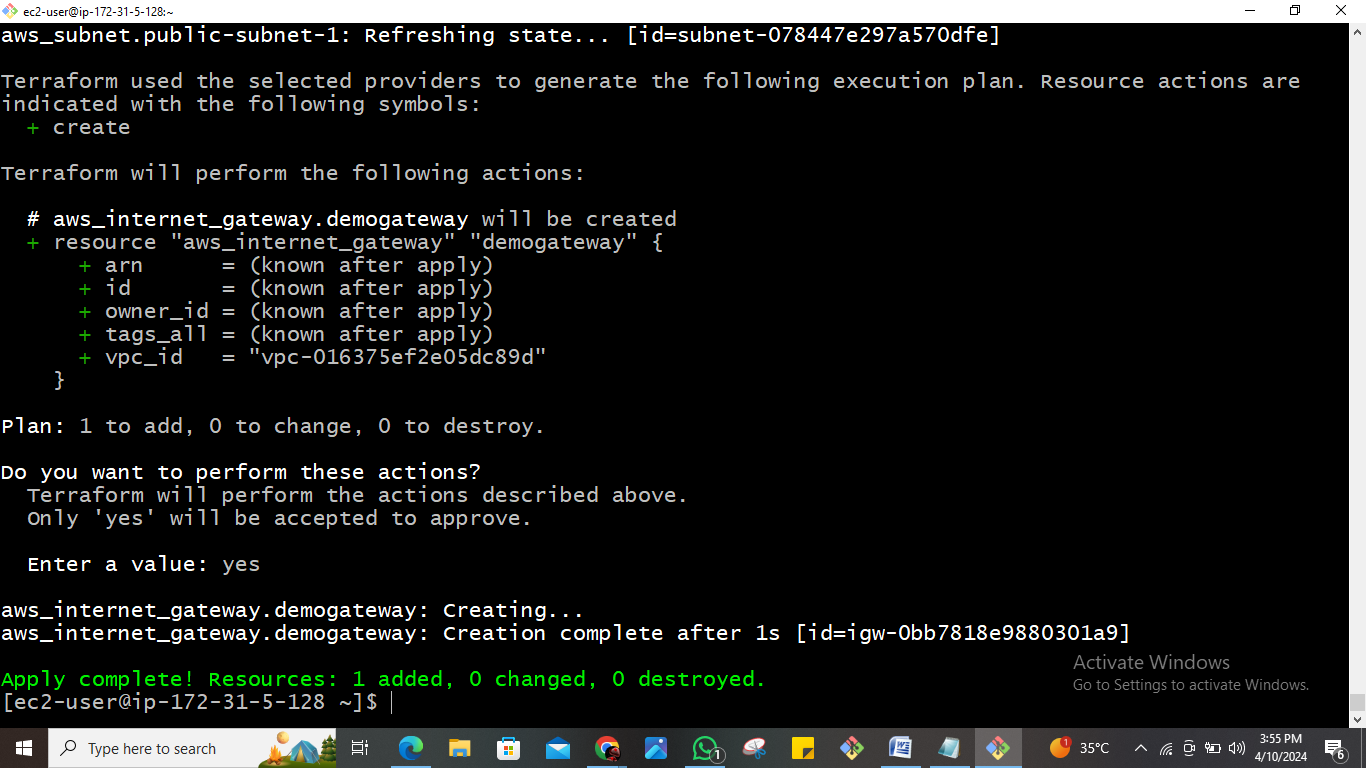
terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

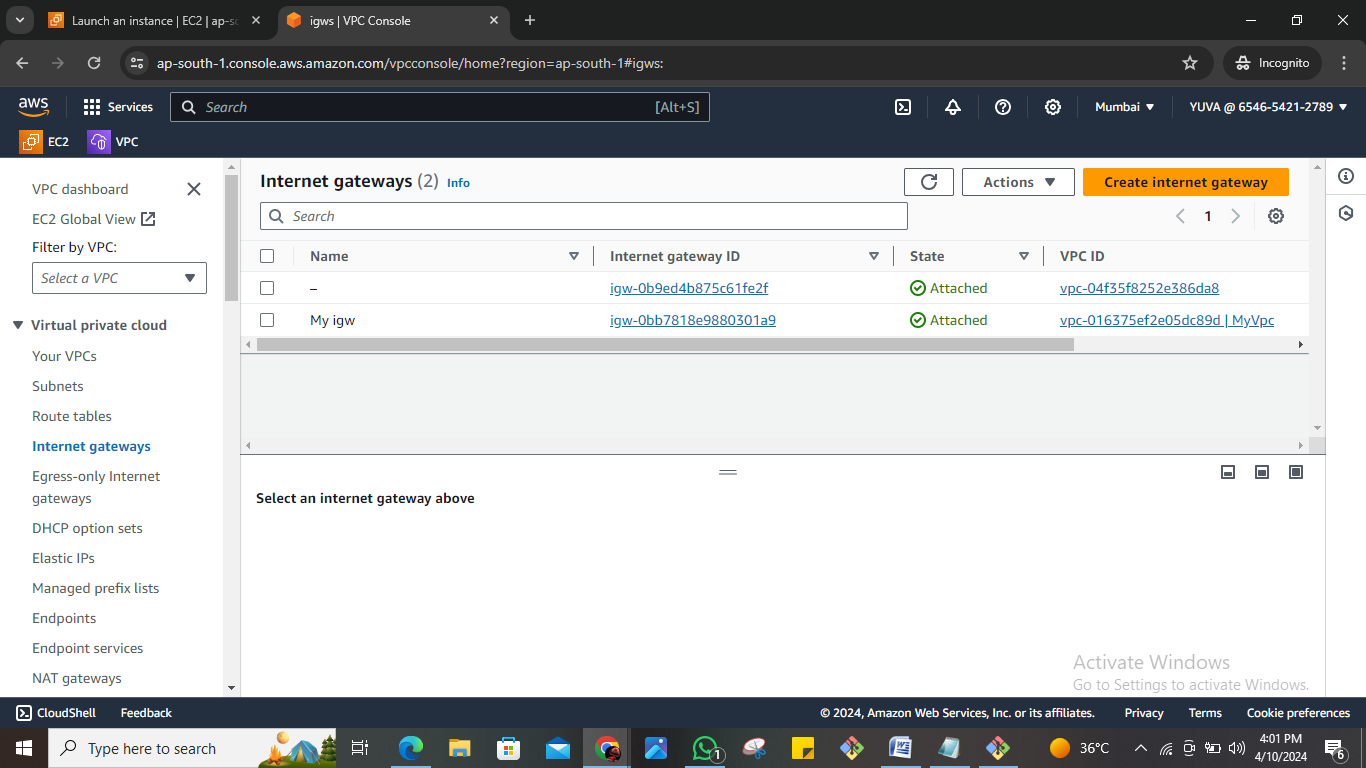
[ Terraform init🡪terraform fmt file name 🡪

terraform validate🡪terraform plan🡪terraform apply ]

Outputs:



* Once, the creation is done we have to check manually in AWS account



* Step : 06

Creation of route table & associate with using terraform(HCL-script):

Created the route.tf file using vi.route.tf and add the script to it

File name: route.tf

# creating Route table

resource "aws\_route\_table" "route" {

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

route {

cidr\_block = "0.0.0.0/0"

gateway\_id = aws\_internet\_gateway.demogateway.id

}

tags = {

Name = " Route to internet "

}

}

# Associated Route table

resource "aws\_route\_table\_association" "rt1" {

subnet\_id = aws\_subnet.public-subnet-1.id

route\_table\_id = aws\_route\_table.route.id

}

# Associated Route table

resource "aws\_route\_table\_association" "rt2" {

subnet\_id = aws\_subnet.public-subnet-2.id

route\_table\_id = aws\_route\_table.route.id

}

* After create the route.tf file using vi route.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

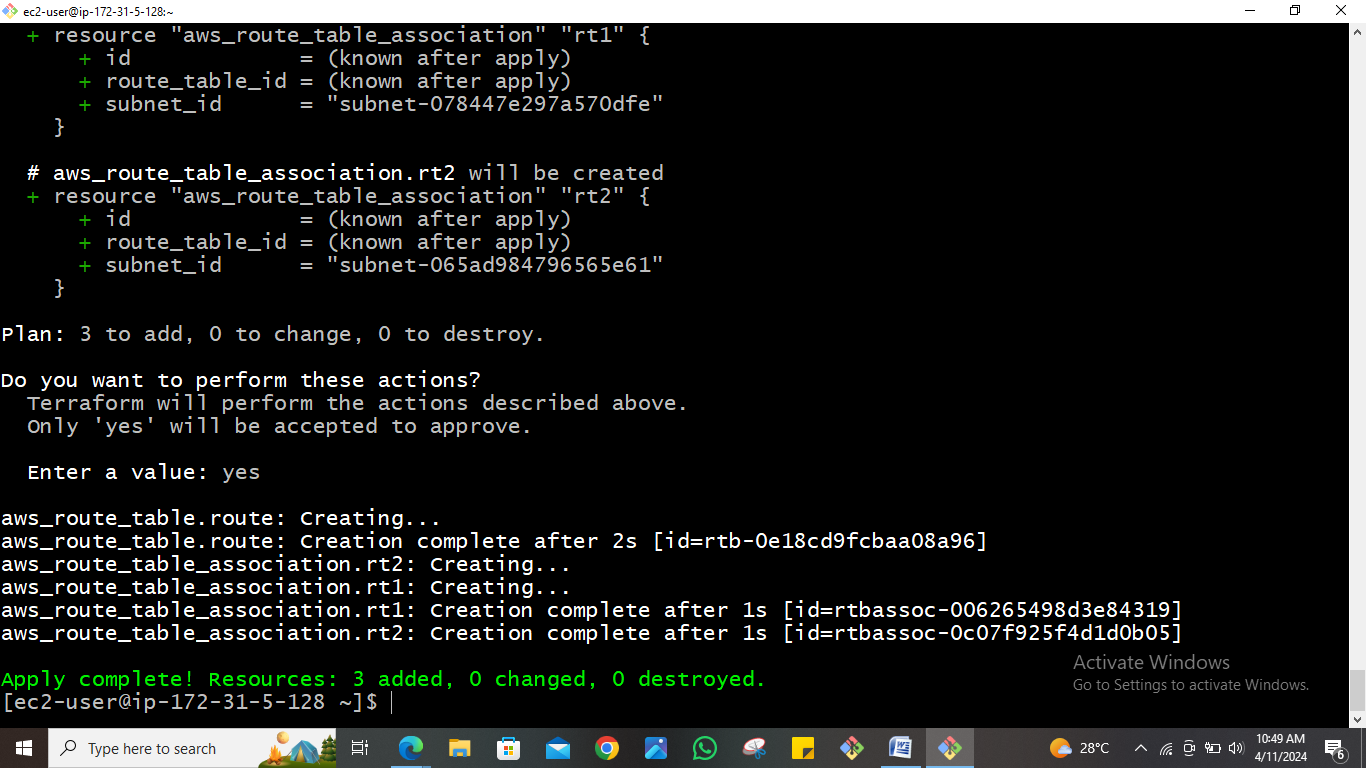
terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

[ Terraform init🡪terraform fmt file name 🡪

terraform validate🡪terraform plan🡪terraform apply ]

Outputs:



* Once, the creation is done we have to check manually in AWS account



Step : 07

Creation of security group using terraform(HCL-script) :

Created the sg.tf file using vi sg.tf and add the script to it

File name: sg.tf

# creating a security group

resource "aws\_security\_group" "demosg" {

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

# INBOUND RULES

# HTTP ACCESS FROM ANYWHERE

ingress {

from\_port = 80

to\_port = 80

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

# HTTPS ACCESS FROM ANYWHERE

ingress {

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

# SSH ACCESS FROM ANYWHERE

ingress {

from\_port = 22

to\_port = 22

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

# OUTBOUND RULES

# INPUT ACCESS TO ANYWHERE

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "websg"

}

}

* After create the sg.tf file using vi sg.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

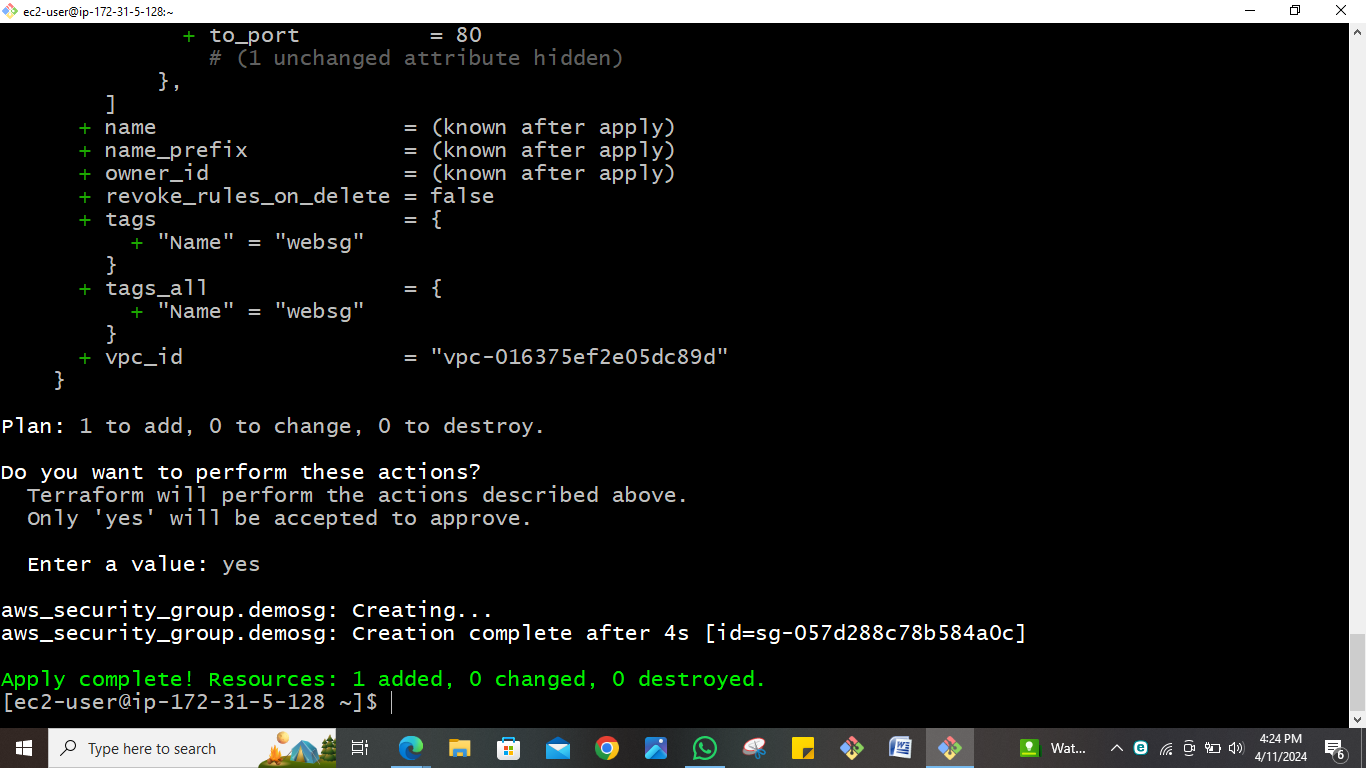
terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

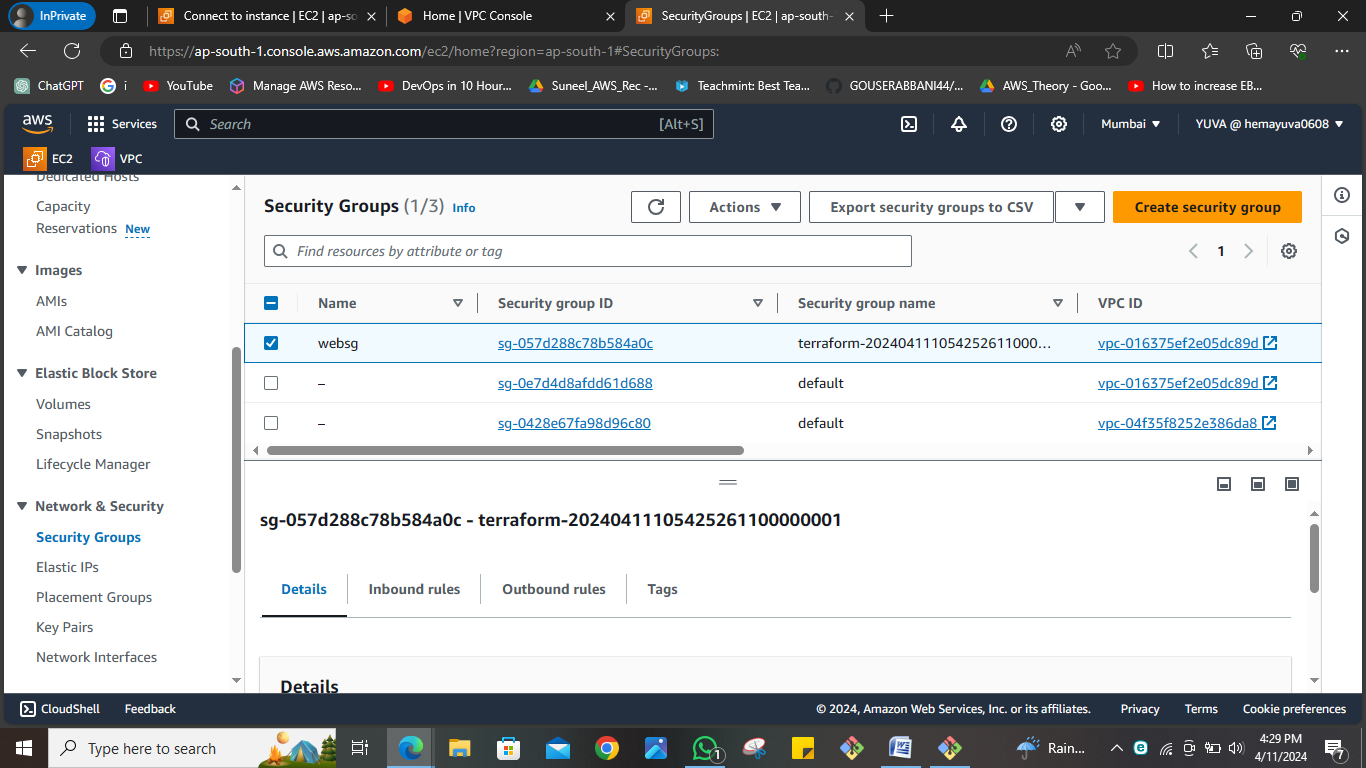
[ Terraform init🡪terraform fmt file name 🡪

terraform validate🡪terraform plan🡪terraform apply ]

Outputs:



* Once, the creation is done we have to check manually in AWS account



* Step : 08

Creation of user data file for instance using terraform (HCL-Script):

Created the data.sh file using vi data.sh and add the script to it

File name: data.sh

#!/bin/bash

yum update -y

yum install -y httpd.x86\_64

systemctl start httpd.service

systemctl enable httpd.service

echo "hello world" from $(hostname -f) > /var/www/html/index.html

Note: If we want to deploy an application to this instance use below user data script:

#creating data1.sh for hosting a instance with a application

#!/bin/bash

yum update -y

yum install git -y

yum install httpd.x86\_64

systemctl start httpd.service

systemctl enable httpd.service

git clone http://github.com/Hemayuva/food.git /var/www/html/

* Step : 09

Creation of instance using terraform (HCL-Script):

Created the instanceec2.tf file using vi instanceec2.tf and add the script to it

File name: instanceec2.tf

# creating the instance

resource "aws\_instance" "terra1instance" {

ami = "ami-0451f2687182e0411"

instance\_type = "t2.micro"

count = 1

key\_name = "yuva-key"

security\_groups = [aws\_security\_group.demosg.id]

subnet\_id = aws\_subnet.public-subnet-1.id

associate\_public\_ip\_address = true

user\_data = file("data.sh")

tags = {

Name = " Terraform 1"

}

}

resource "aws\_key\_pair" "yuva" {

key\_name = "yuva-key"

public\_key = "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQD3F6tyPEFEzV0LX3X8BsXdMsQz1x2cEikKDEY0aIj41qgxMCP/iteneqXSIFZBp5vizPvaoIR3Um9xK7PGoW8giupGn+EPuxIA4cDM4vzOqOkiMPhz5XK0whEjkVzTo4+S0puvDZuwIsdiW9mxhJc7tgBNL0cYlWSYVkz4G/fslNfRPW5mYAM49f4fhtxPb5ok4Q2Lg9dPKVHO/Bgeu5woMc7RY0p1ej6D4CKFE6lymSDJpW0YHX/wqE9+cfEauh7xZcG0q9t2ta6F6fmX0agvpFyZo8aFbXeUBr7osSCJNgvavWbM/06niWrOvYX2xwWdhXmXSrbX8ZbabVohBK41 email@example.com"

}

#creation of 2nd instance

resource "aws\_instance" "terra2instance" {

ami = "ami-0451f2687182e0411"

instance\_type = "t2.micro"

count = 1

key\_name = "yuva-key"

security\_groups = [aws\_security\_group.demosg.id]

subnet\_id = aws\_subnet.public-subnet-2.id

associate\_public\_ip\_address = true

user\_data = file("data.sh")

tags = {

Name = " Terraform 2 "

}

}

* After create the subnet.tf file using vi var.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

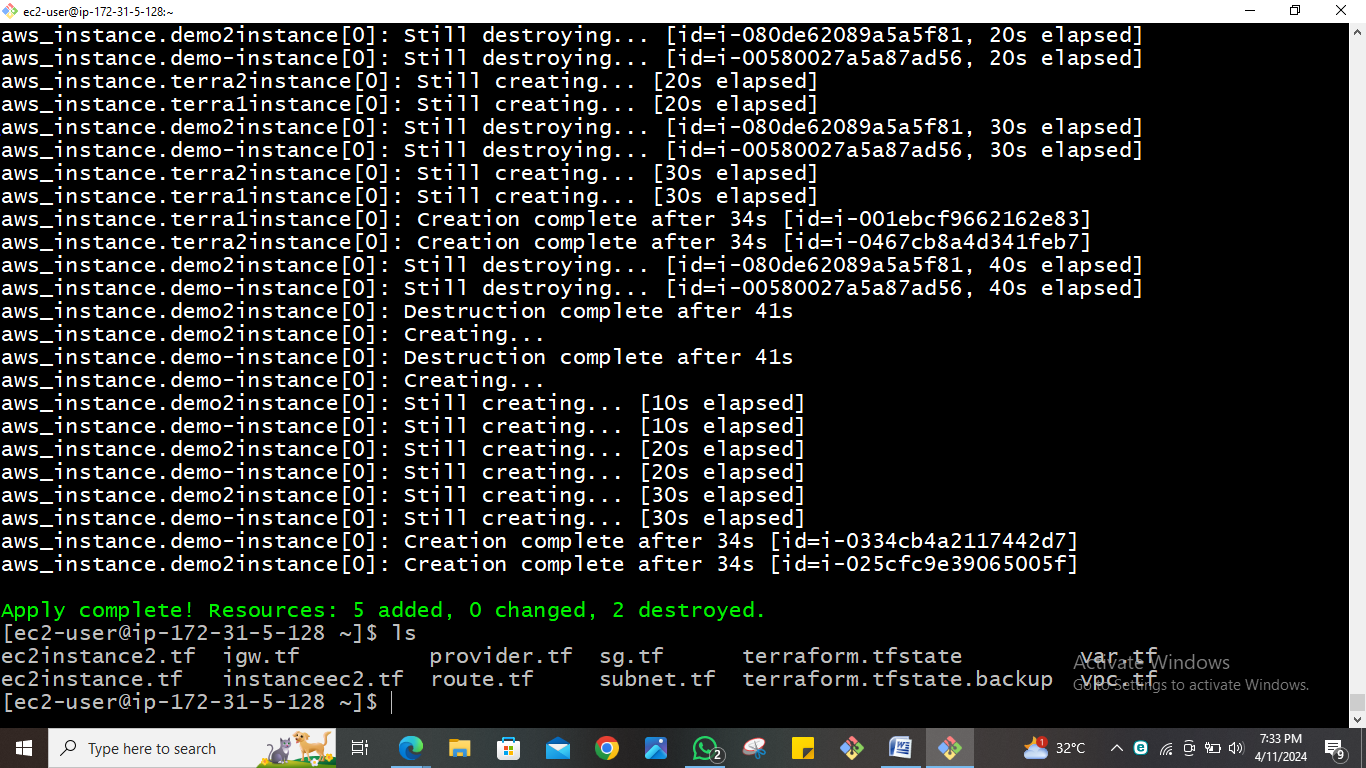
terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

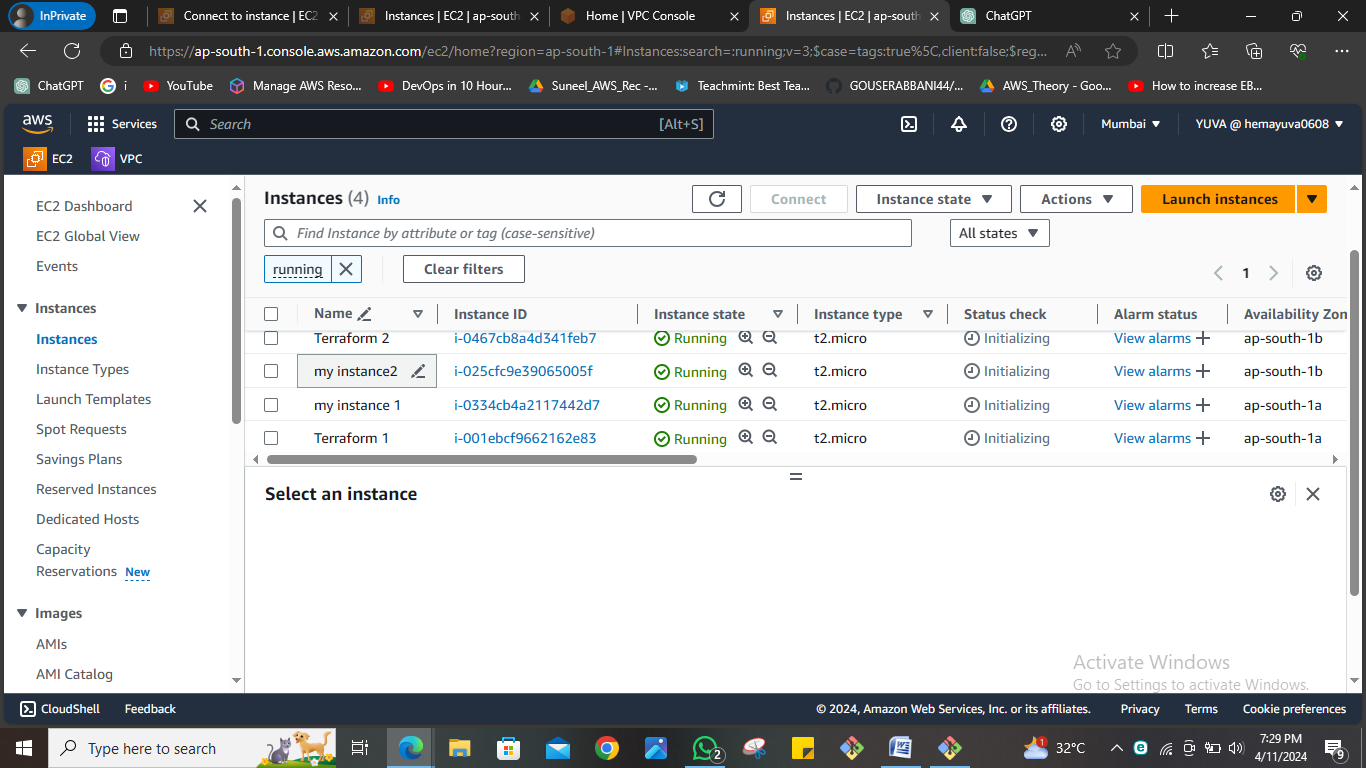
[ Terraform init🡪terraform fmt file name 🡪

terraform validate🡪terraform plan🡪terraform apply ]

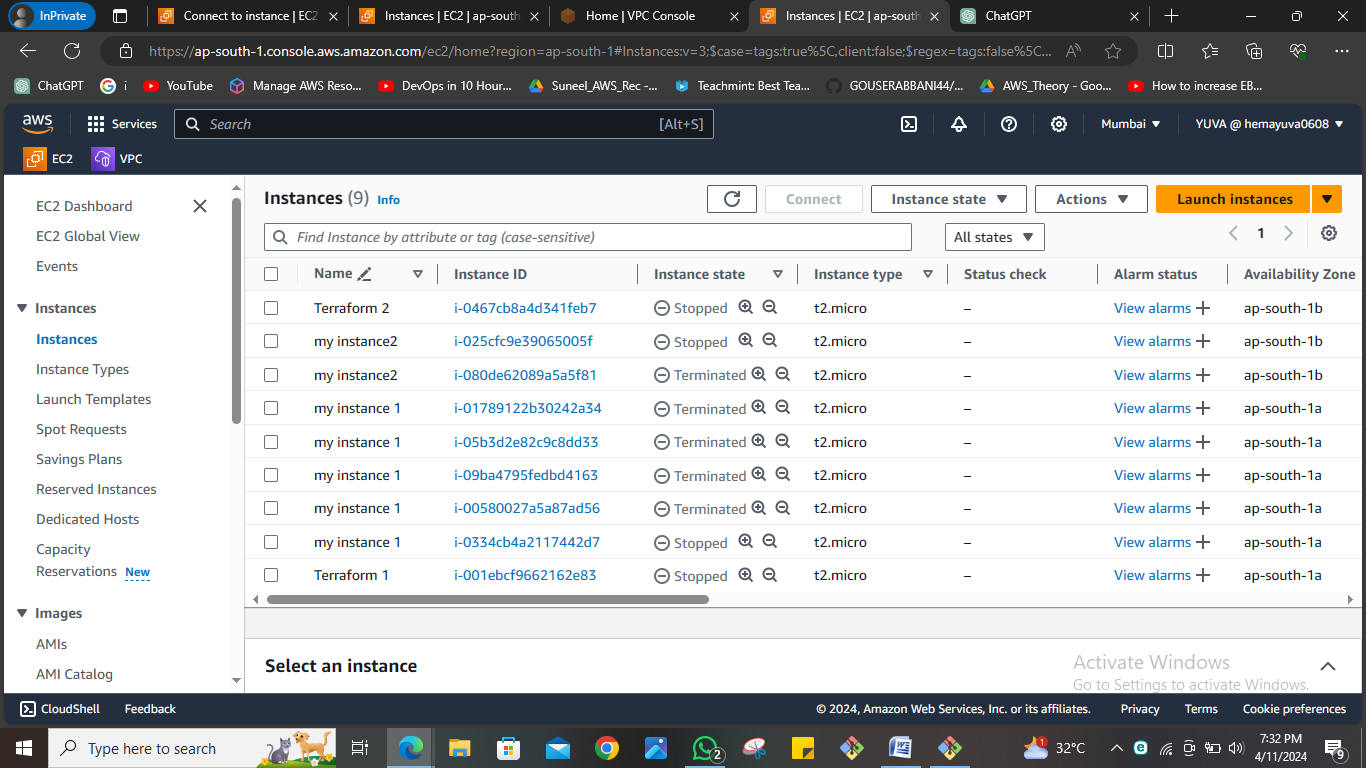
Outputs:



* Once, the creation is done we have to check manually in AWS account



Finally stopped the created instances manually to avoid billing :



* Step : 10

Creation of database security group using terraform (HCL-Script):

Created the dbsg.tf file using vi dbsg.tf and add the script to it

File name: dbsg.tf

#Create database security group

resource "aws\_security\_group" "database-sg" {

name = "database sg"

description = "Allow inbound traffic from application layer"

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

ingress {

description = "allow traffic from application layer"

from\_port = 3306

to\_port = 3306

protocol = "tcp"

security\_groups = [aws\_security\_group.demosg.id]

}

egress {

from\_port = 32768

to\_port = 65535

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

tags = {

Name = "Database sg"

}

}

* After create the subnet.tf file using vi var.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

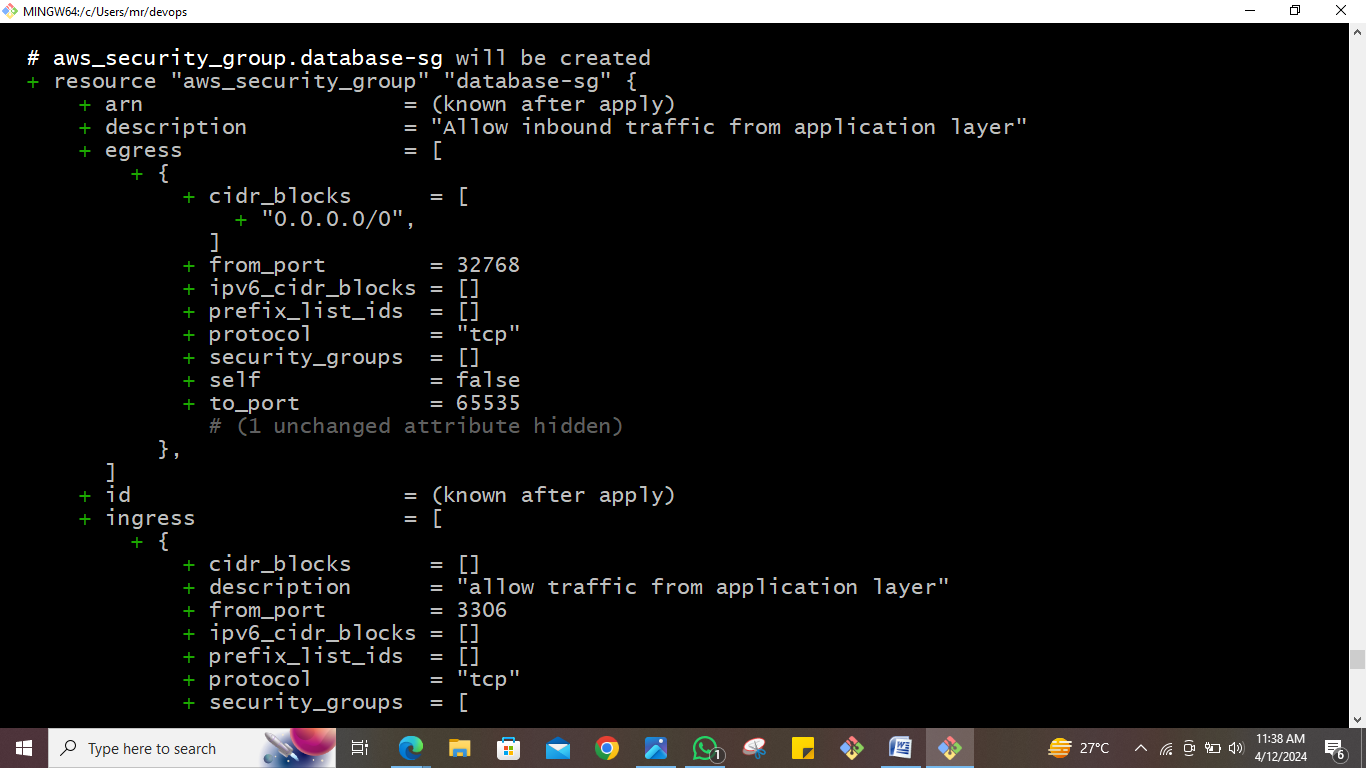
terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

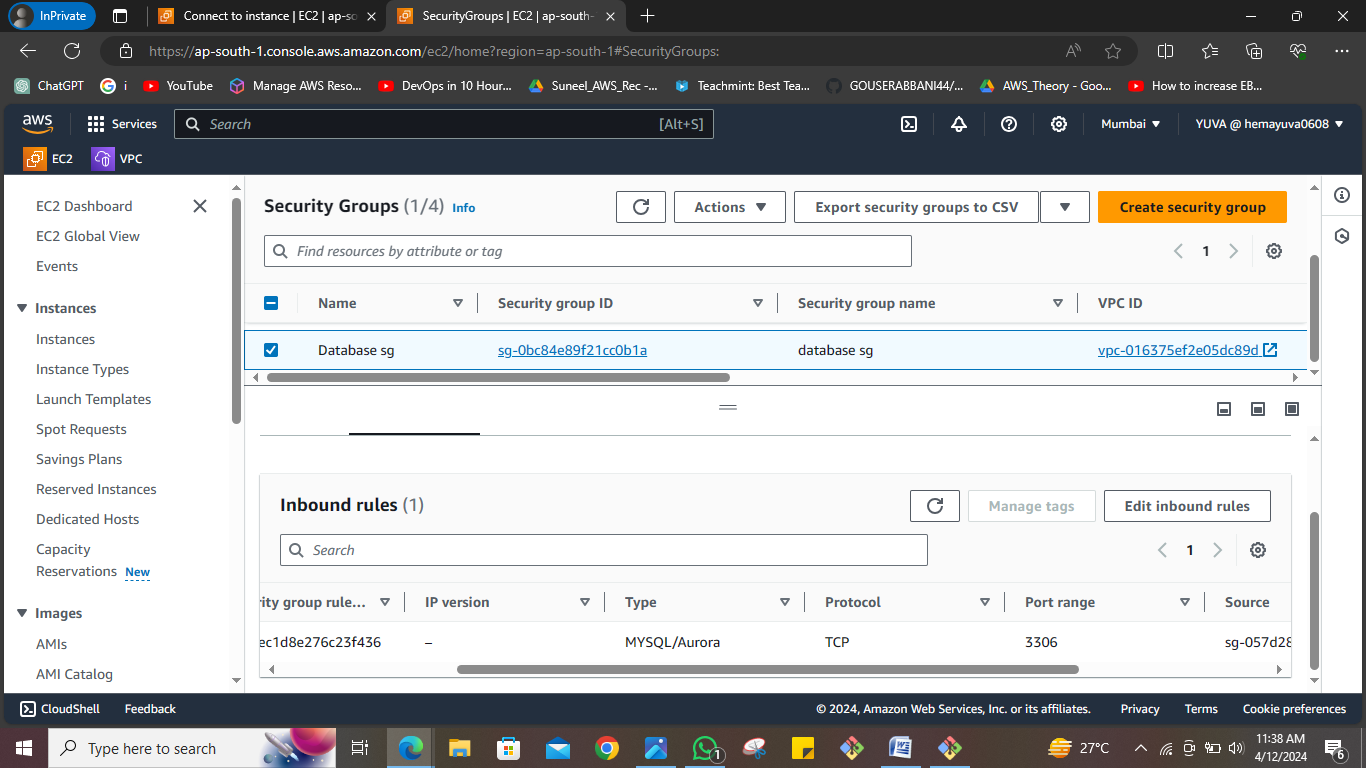
[ Terraform init🡪terraform fmt file name 🡪

Terraform validate🡪terraform plan🡪terraform apply]

Outputs:



* Once, the creation is done we have to check manually in AWS account



* Step : 11

Creation of RDS using terraform (HCL-Script):

Created the rds.tf file using vi rds.tf and add the script to it

File name: rds.tf

# creating RDS instance

resource "aws\_db\_subnet\_group" "default" {

name = "main"

subnet\_ids = [aws\_subnet.Database-subnet-1.id, aws\_subnet.Database-subnet-2.id]

tags = {

Name = "My-db-subnet-group"

}

}

resource "aws\_db\_instance" "my-rds" {

allocated\_storage = 10

db\_subnet\_group\_name = aws\_db\_subnet\_group.default.name

engine = "mysql"

engine\_version = "8.0.34"

instance\_class = "db.t3.micro"

multi\_az = "true"

identifier = "my-rds-db"

username = "Hema"

password = "Hema0608#"

skip\_final\_snapshot = true

vpc\_security\_group\_ids = [aws\_security\_group.database-sg.id]

}

* After create the subnet.tf file using vi var.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

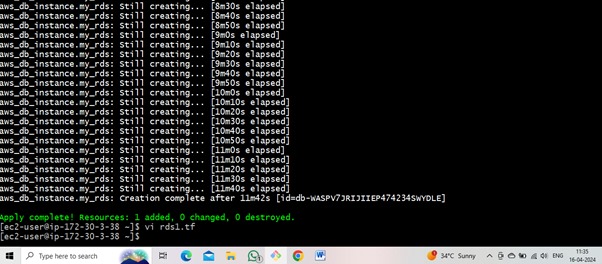
terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

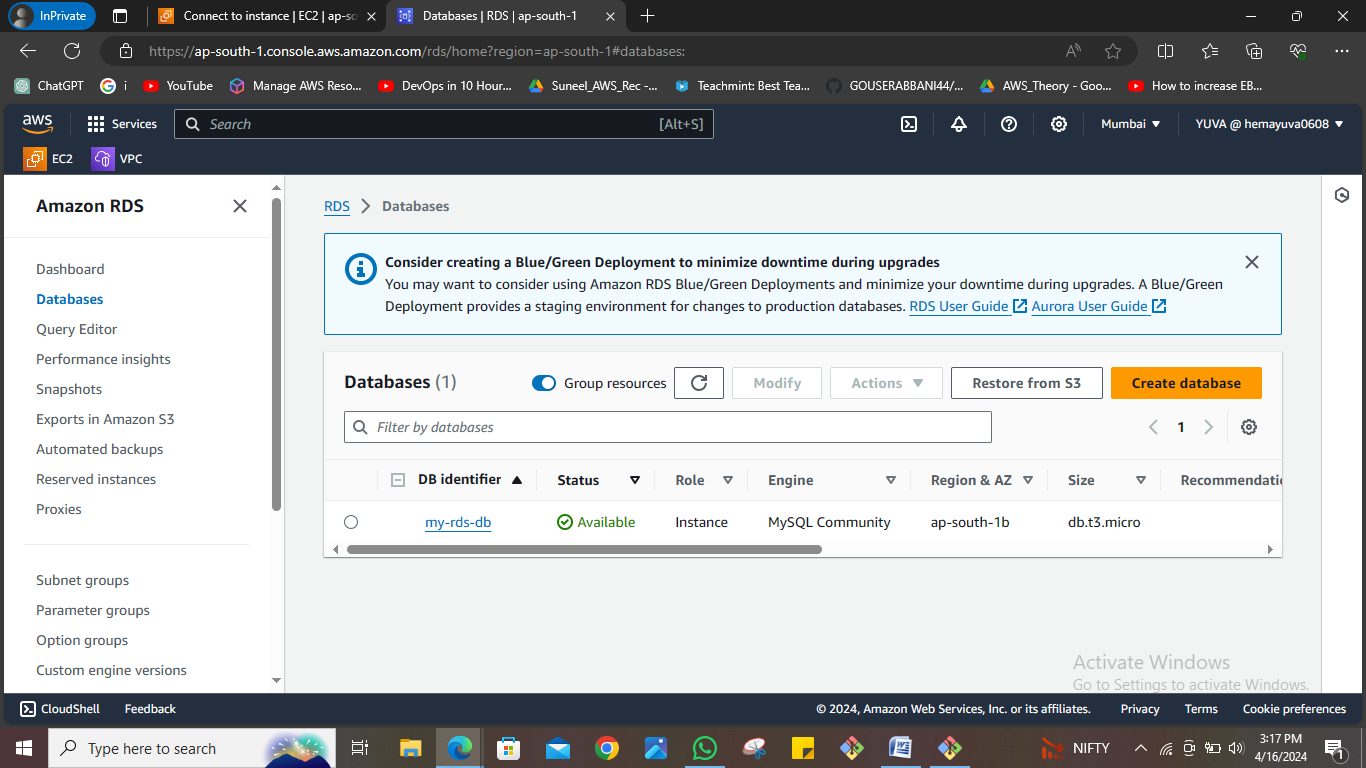
[ Terraform init🡪terraform fmt file name 🡪

terraform validate🡪terraform plan🡪terraform apply ]

Outputs:



* Once, the creation is done we have to check manually in AWS account



* Step : 12

Creation of load balancer using terraform :

Created the lb.tf file using vi lb.tf and add the script to it

File name: lb.tf

# creating External Load balancer

resource "aws\_lb" "LB\_elb" {

name = "External-LB"

internal = false

load\_balancer\_type = "application"

security\_groups = [aws\_security\_group.demosg.id]

subnets = [aws\_subnet.public-subnet-1.id, aws\_subnet.public- subnet-2.id]

}

resource "aws\_lb\_target\_group" "tg-elb" {

name = "ALB-TG1"

port = 80

protocol = "HTTP"

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

}

resource "aws\_lb\_target\_group\_attachment" "attachment" {

count = length(aws\_instance.terra1instance)

target\_group\_arn = aws\_lb\_target\_group.tg-elb.arn

target\_id = aws\_instance.terra1instance[count.index].id

port = 80

depends\_on = [aws\_instance.terra1instance]

}

resource "aws\_lb\_target\_group\_attachment" "attachment1" {

count = length(aws\_instance.terra2instance)

target\_group\_arn = aws\_lb\_target\_group.tg-elb.arn

target\_id = aws\_instance.terra2instance[count.index].id

port = 80

depends\_on = [aws\_instance.terra2instance]

}

resource "aws\_lb\_listener" "LB\_elb" {

load\_balancer\_arn = aws\_lb.LB\_elb.arn

port = 80

protocol = "HTTP"

default\_action {

type = "forward"

target\_group\_arn = aws\_lb\_target\_group.tg-elb.arn

}

}

* After create the subnet.tf file using vi var.tf
* we have to create the infrastructure using the below commands
* Initialize it using : terraform init
* Format (or)alignment(align the script using) :

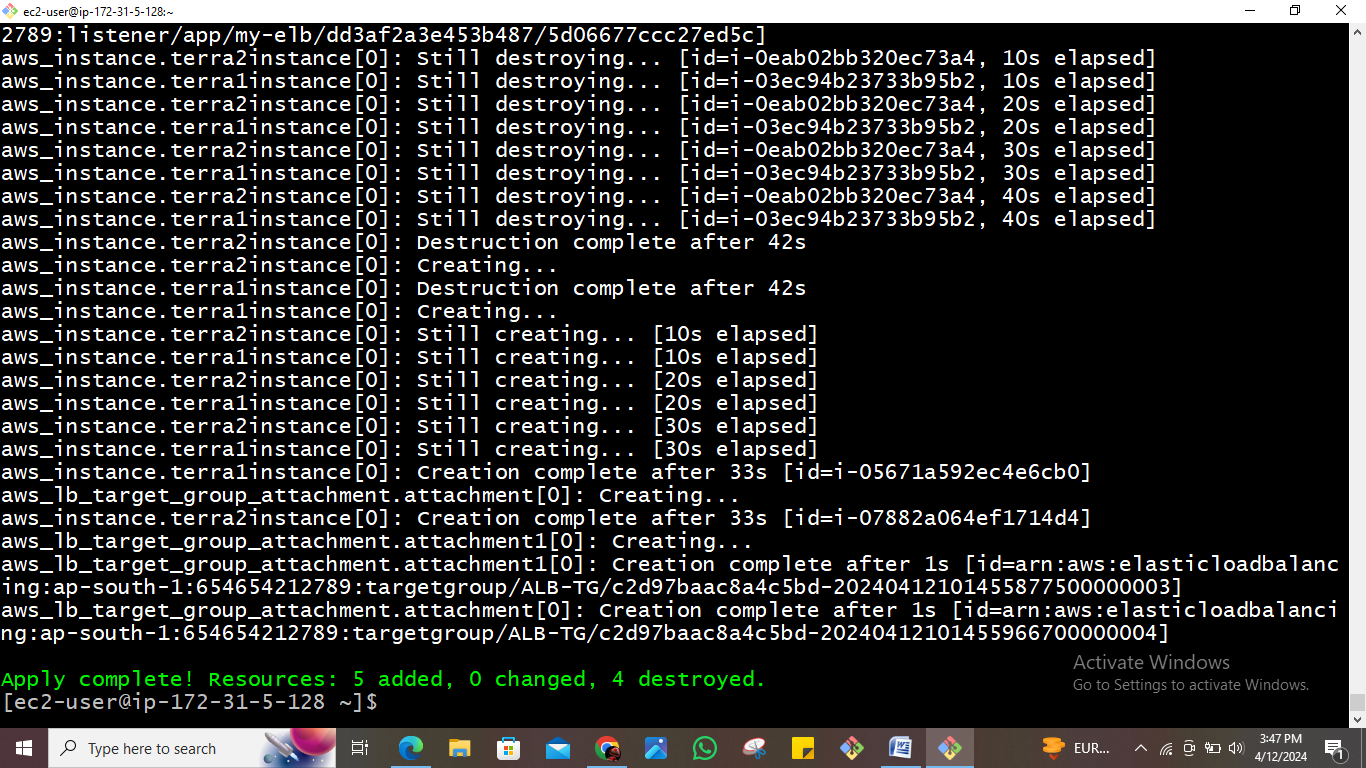
terraform fmt # file name

* Validate : terraform validate (to validate the script)
* Plan: terraform plan
* Apply : terraform apply

[ Terraform init🡪terraform fmt file name 🡪

terraform validate🡪terraform plan🡪terraform apply ]

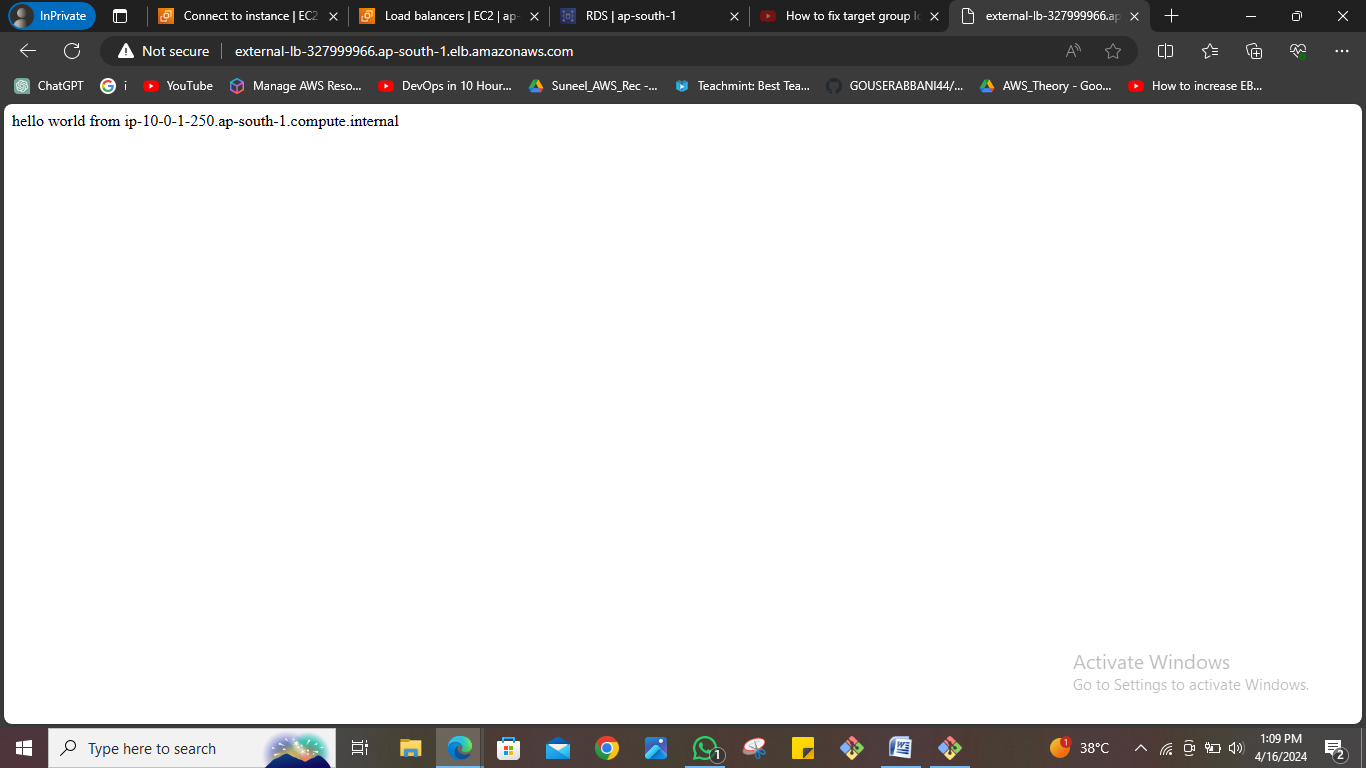
Outputs:



* Once, the creation is done we have to check manually in AWS account

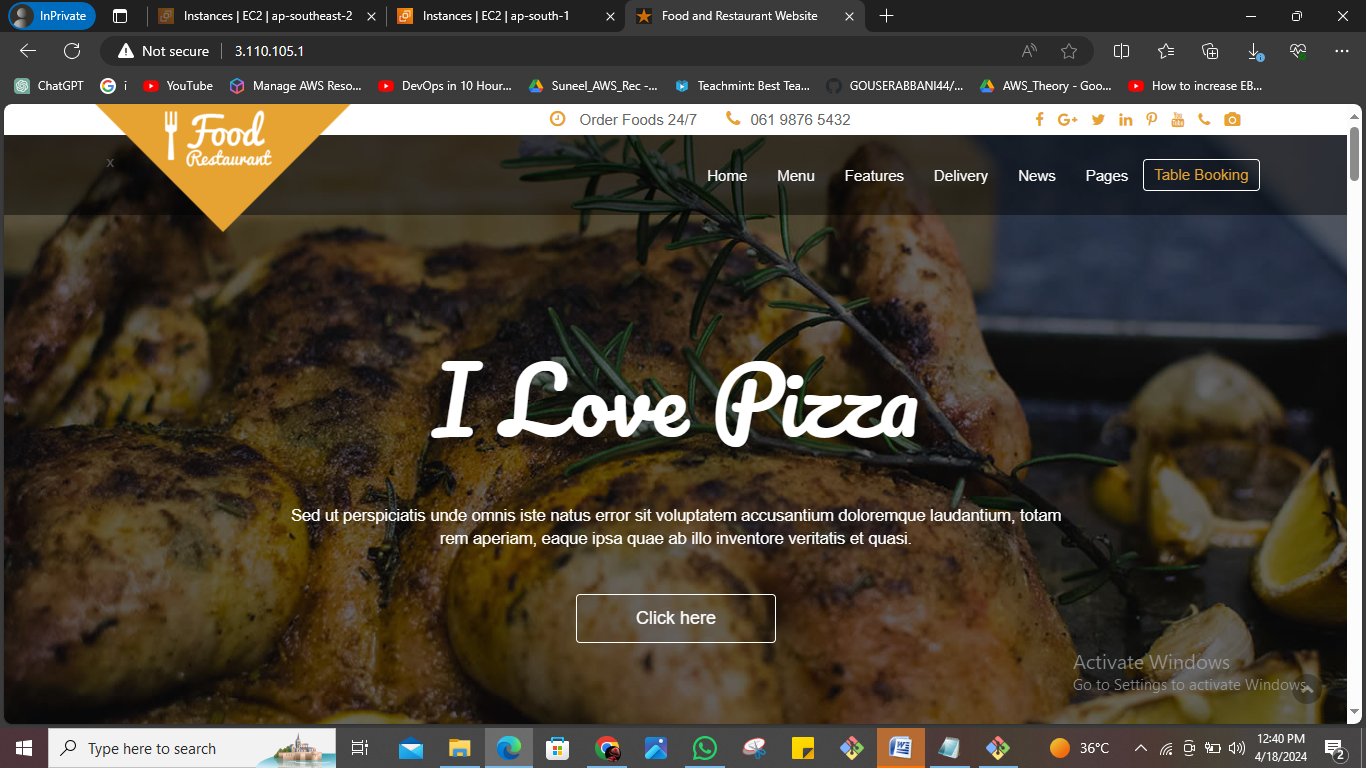


Browse through DNS:

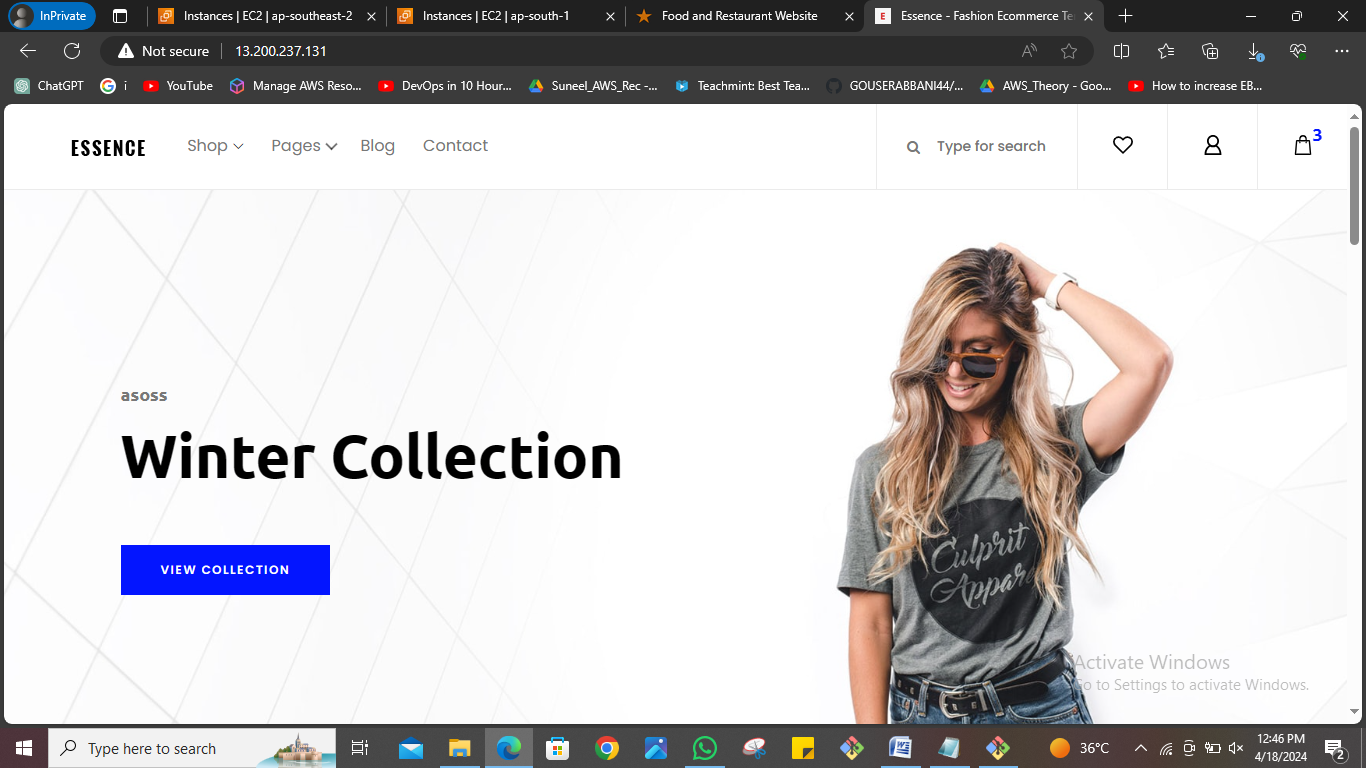


Using the another user data file Hosted the application:

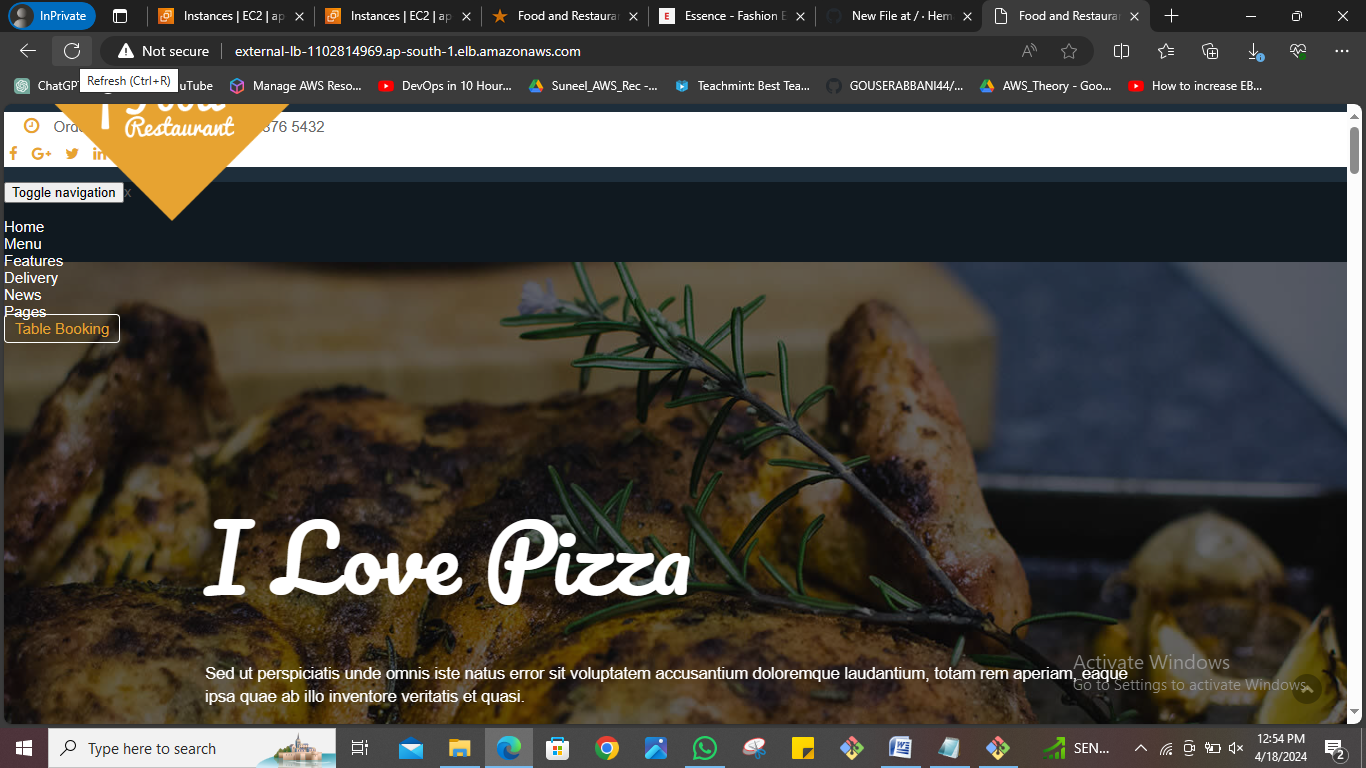
Outputs:

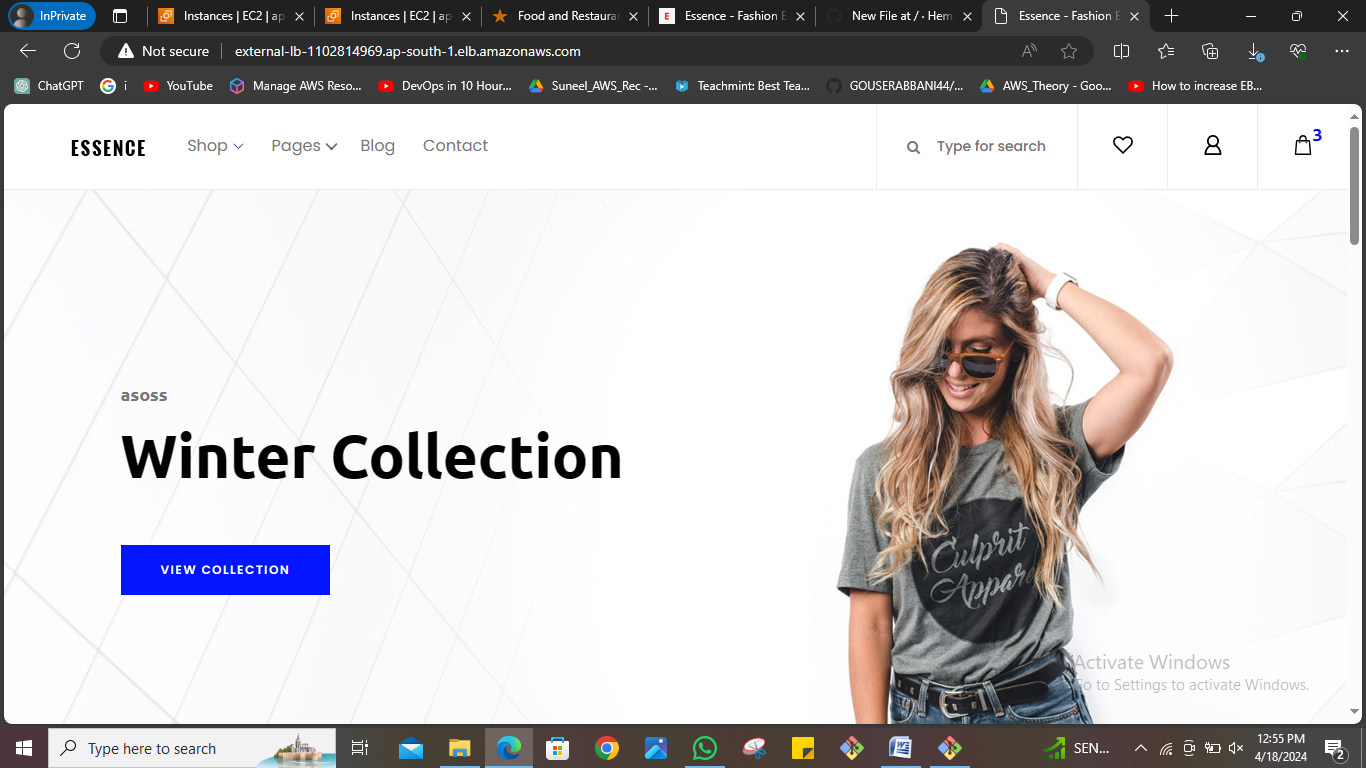


2nd instance :



Checking With DNS name whether the application is shuffled or not





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*THE END\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*