Lesson-End Project

Optimizing Resilience in AWS Infrastructure Deployment Using Terraform

Project agenda: To deploy and manage a resilient AWS infrastructure using Terraform for optimal performance and scalability

Description: Imagine you are a cloud engineer tasked with deploying and managing a resilient AWS infrastructure using Terraform. The project involves creating a VPC with public and private subnets, deploying necessary resources such as an Internet Gateway and a NAT Gateway, and demonstrating the use of HCL functions for string manipulation, collections, and encoding using Terraform. This project aims to provide a comprehensive understanding of infrastructure as code (IaC) practices.

Tools required: AWS Account, Terraform, and VS Code

Prerequisites: You must have an AWS Account, and Terraform and VS Code installed

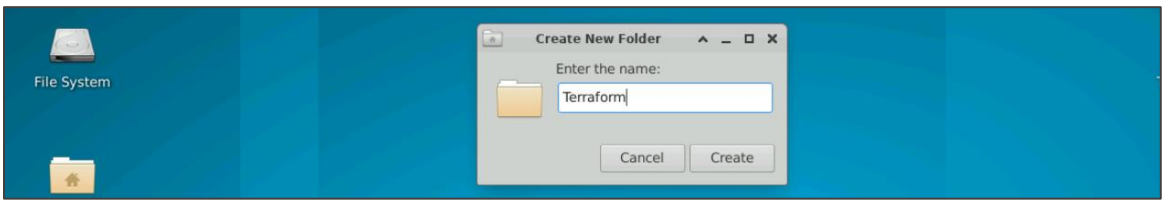
Expected Deliverables: Terraform configuration files for deploying a resilient AWS infrastructure with VPC, public and private subnets, Internet Gateway, and NAT Gateway. This includes HCL functions for string manipulation and collections, with clear documentation of installation and execution steps.

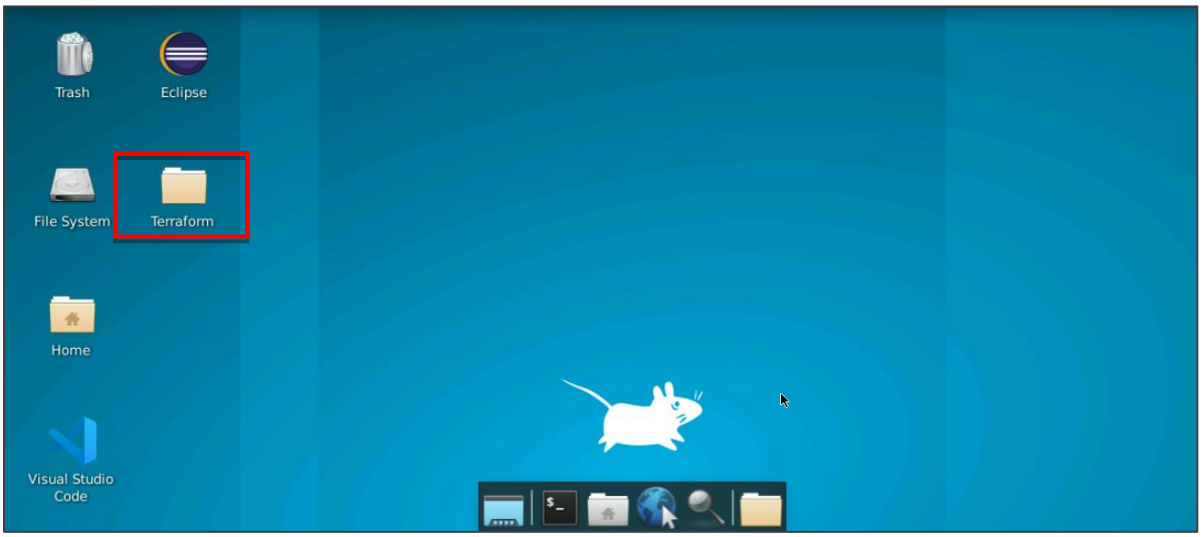
Steps to be followed:

1. Prepare files and set up Terraform
2. Deploy the AWS infrastructure
3. Implement HCL functions for string manipulation, collections, and encoding
4. Validate the Terraform configuration
5. Clean up AWS resources

Step 1: Prepare files and set up Terraform

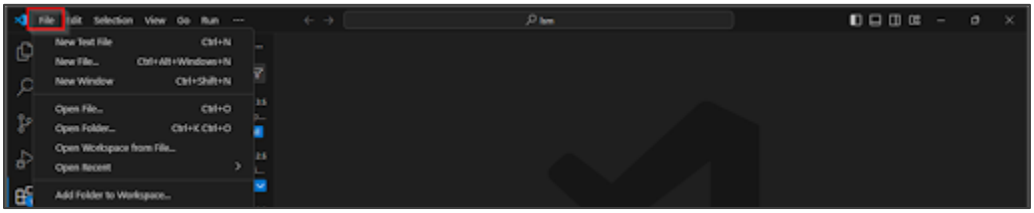
1.1 Create a folder named **Terraform** on the desktop



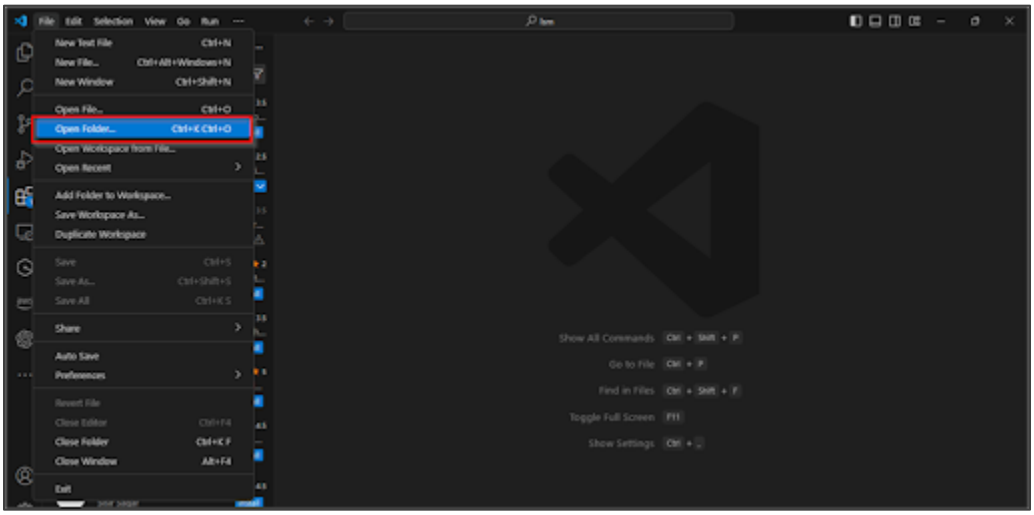


1.2 Open the **VS Code editor**

1.3 Open the **File** option present on the top console



1.4 Click on **Open Folder** in the drop-down menu to open the **Terraform** folder



The **Terraform** folder will open in the VS Code editor.

1.5 In the **Explorer** section, click on the **New File** option to create new files

1.6 In the **Terraform** folder, create files named **main.tf**, **variables.tf**, and **output.tf**

1.7 Define variables in the **variables.tf** file:

```
variable "aws_region" {
  type    = string
  default = "us-east-1"
}

variable "vpc_name" {
  type    = string
  default = "demo_vpc"
}

variable "vpc_cidr" {
  type    = string
  default = "10.0.0.0/16"
}

variable "private_subnets" {
  default = {
    "private_subnet_1" = 1
    "private_subnet_2" = 2
    "private_subnet_3" = 3
  }
}

variable "public_subnets" {
  default = {
    "public_subnet_1" = 1
    "public_subnet_2" = 2
    "public_subnet_3" = 3
  }
}
```

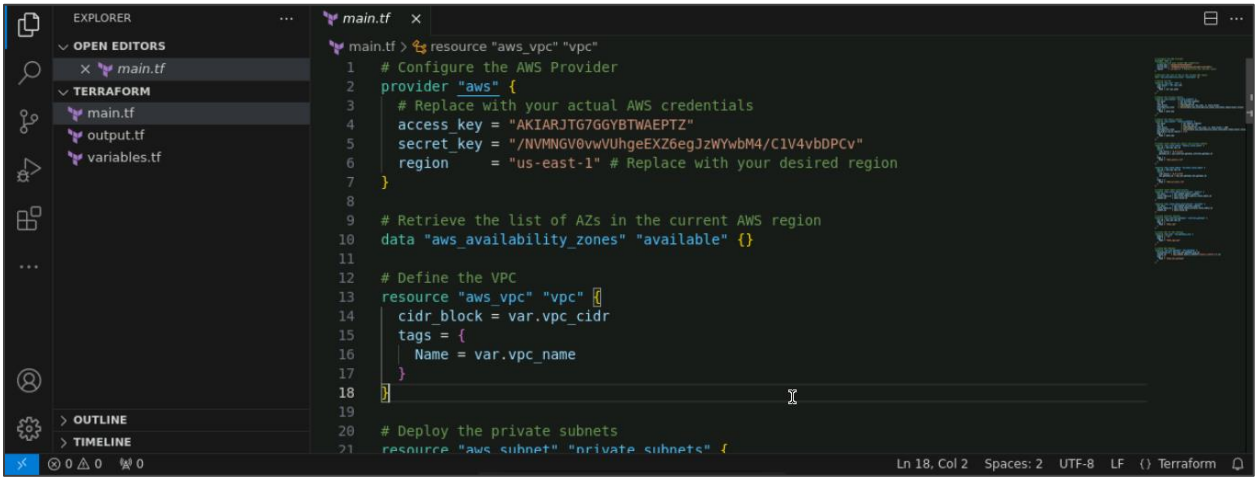
1.8 Write the main configuration in the **main.tf** file:

```
# Configure the AWS Provider
provider "aws" {
 # Replace with your actual AWS credentials
 access_key = "AKIARJTG7GGYBTWAEPTZ"
 secret_key = "/NVMNGV0vwVUhgeEXZ6egJzWYwbM4/C1V4vbDPCv"
 region     = "us-east-1" # Replace with your desired region
}

# Retrieve the list of AZs in the current AWS region
data "aws_availability_zones" "available" {}

# Define the VPC
resource "aws_vpc" "vpc" {
 cidr_block = var.vpc_cidr
 tags = {
  Name = var.vpc_name
 }
}

# Deploy the private subnets
resource "aws_subnet" "private_subnets" {
 for_each          = var.private_subnets
 vpc_id            = aws_vpc.vpc.id
 cidr_block        = cidrsubnet(var.vpc_cidr, 8, each.value)
 availability_zone    = tolist(data.aws_availability_zones.available.names)[each.value]
```

```
  tags = {
    Name = each.key
  }
}

# Deploy the public subnets
resource "aws_subnet" "public_subnets" {
  for_each              = var.public_subnets
  vpc_id                = aws_vpc.vpc.id
  cidr_block            = cidrsubnet(var.vpc_cidr, 8, each.value + 100)
  availability_zone     =
tolist(data.aws_availability_zones.available.names)[each.value]
  map_public_ip_on_launch = true
  tags = {
    Name = each.key
  }
}

# Create route tables for public and private subnets
resource "aws_route_table" "public_route_table" {
  vpc_id = aws_vpc.vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.internet_gateway.id
  }
  tags = {
    Name = "demo_public_rtb"
  }
}

resource "aws_route_table" "private_route_table" {
  vpc_id = aws_vpc.vpc.id
  route {
    cidr_block = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.nat_gateway.id
  }
  tags = {
    Name = "demo_private_rtb"
```

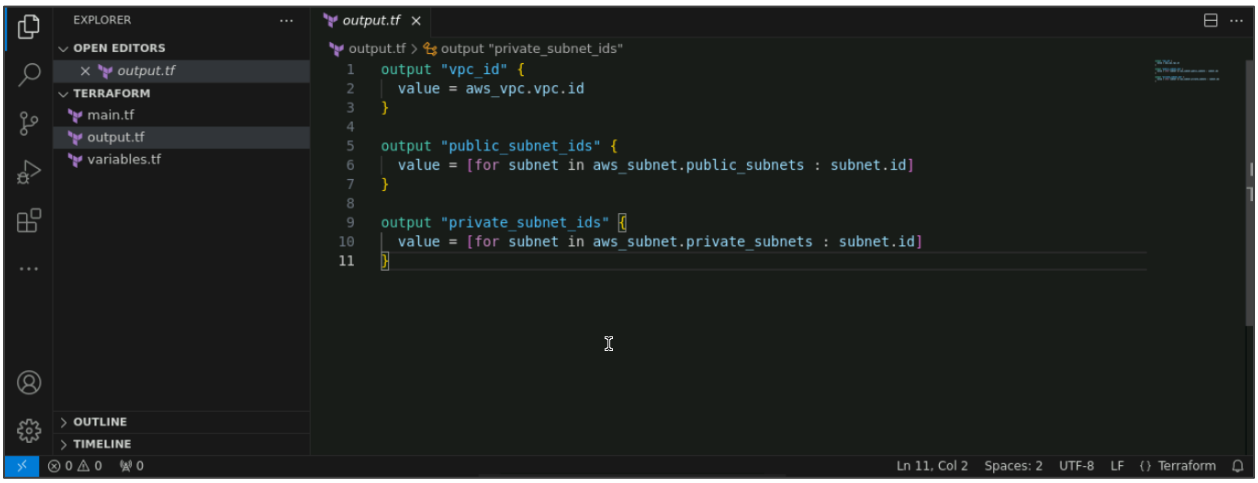
```
  }
}

# Create route table associations
resource "aws_route_table_association" "public" {
  for_each       = aws_subnet.public_subnets
  route_table_id = aws_route_table.public_route_table.id
  subnet_id      = each.value.id
}

resource "aws_route_table_association" "private" {
  for_each       = aws_subnet.private_subnets
  route_table_id = aws_route_table.private_route_table.id
  subnet_id      = each.value.id
}

# Create Internet Gateway
resource "aws_internet_gateway" "internet_gateway" {
  vpc_id = aws_vpc.vpc.id
  tags = {
    Name = "demo_igw"
  }
}

# Create EIP for NAT Gateway
resource "aws_eip" "nat_gateway_eip" {
  domain = "vpc"
  tags = {
    Name = "demo_igw_eip"
  }
}

# Create NAT Gateway
resource "aws_nat_gateway" "nat_gateway" {
  allocation_id = aws_eip.nat_gateway_eip.id
  subnet_id     = aws_subnet.public_subnets["public_subnet_1"].id
  tags = {
    Name = "demo_nat_gateway"
```

```
 }
}
```

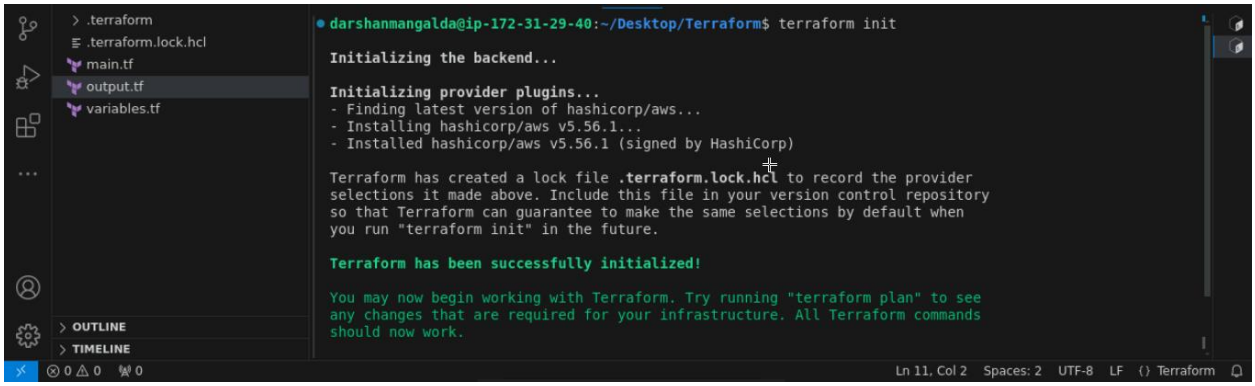


1.9 Define outputs in the **output.tf** file:

```
output "vpc_id" {
value = aws_vpc.vpc.id
}
 output "public_subnet_ids"
{
 value = [for subnet in aws_subnet.public_subnets : subnet.id]
}
 output "private_subnet_ids" {
value = [for subnet in aws_subnet.private_subnets : subnet.id]
}
```

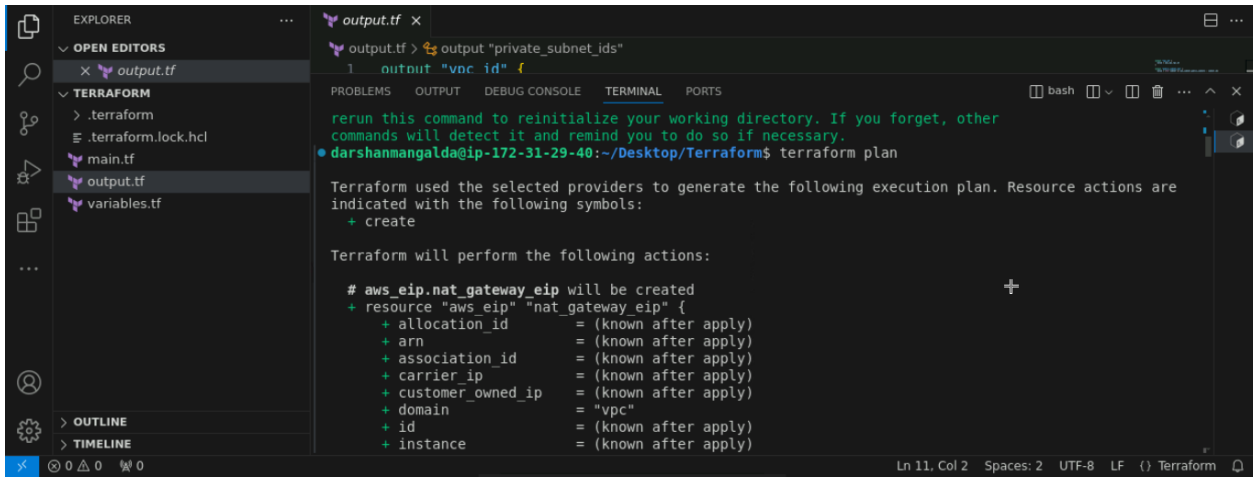


1.10 Open the terminal and run the following command:
terraform init



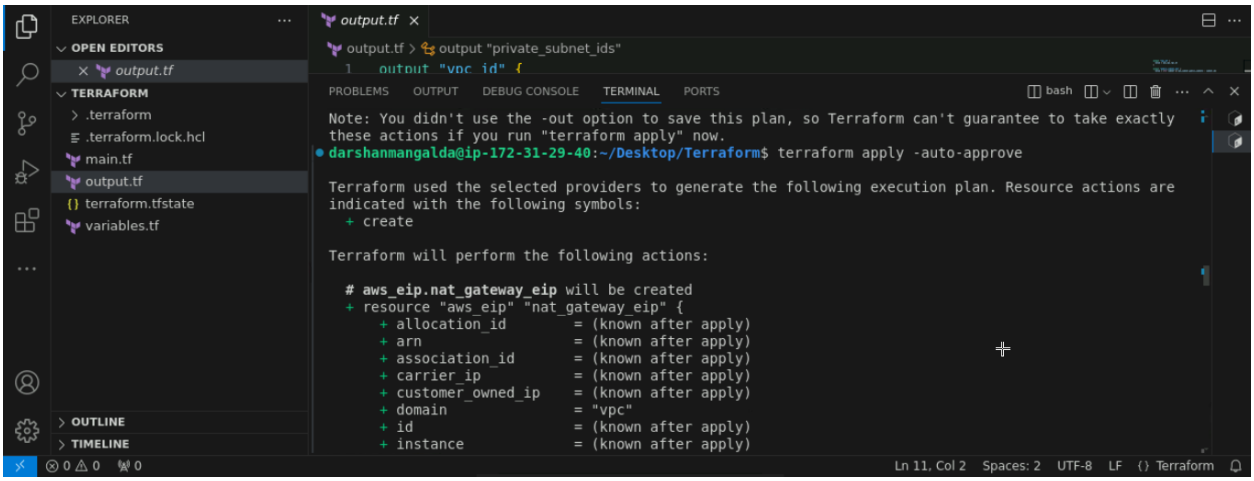
Step 2: Deploy the AWS infrastructure

2.1 Open the terminal tab and execute the following command to create an execution plan:
terraform plan

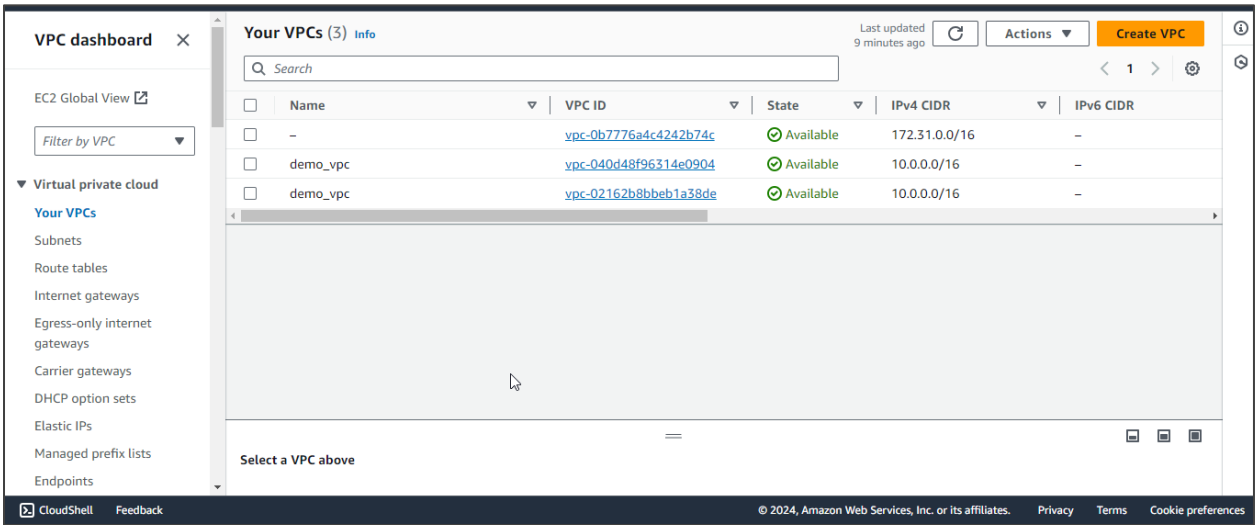


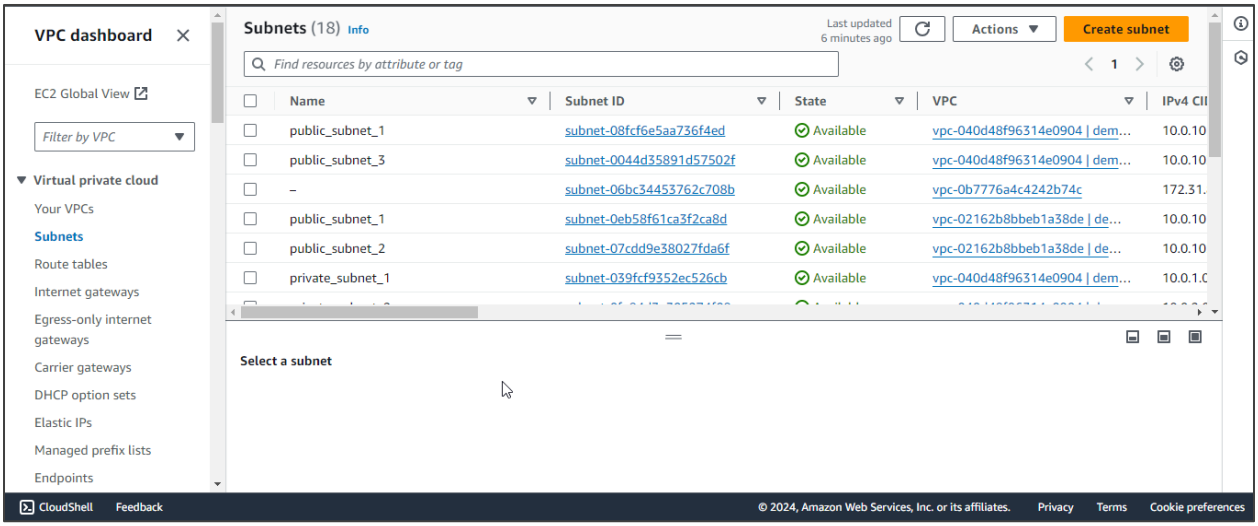
2.2 Execute the following command to apply the configuration:

terraform apply -auto-approve



2.3 Navigate to the AWS console and verify the VPC, subnets, and gateways:





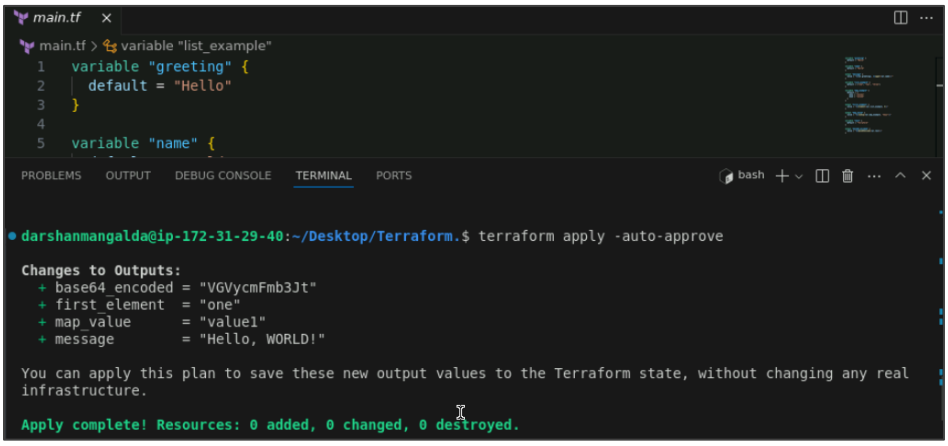
Step 3: Implement HCL Functions for string manipulation, collections, and encoding

3.1 To add HCL functions, replace the existing code in the main.tf file with the following:

```
variable "greeting" {
  default = "Hello"
}

variable "name" {
  default = "World"
}

output "message" {
  value = "${var.greeting} ${upper(var.name)}!"
}

variable "list_example" {
  default = ["one", "two", "three"]
}

variable "map_example" {
  default = {
    key1 = "value1"
    key2 = "value2"
  }
}

output "first_element" {
```

```
  value = "${element(var.list_example, 0)}"
}

output "map_value" {
  value = "${lookup(var.map_example, "key1")}"
}

variable "text" {
  default = "Terraform"
}

output "base64_encoded" {
  value = "${base64encode(var.text)}"
}
```

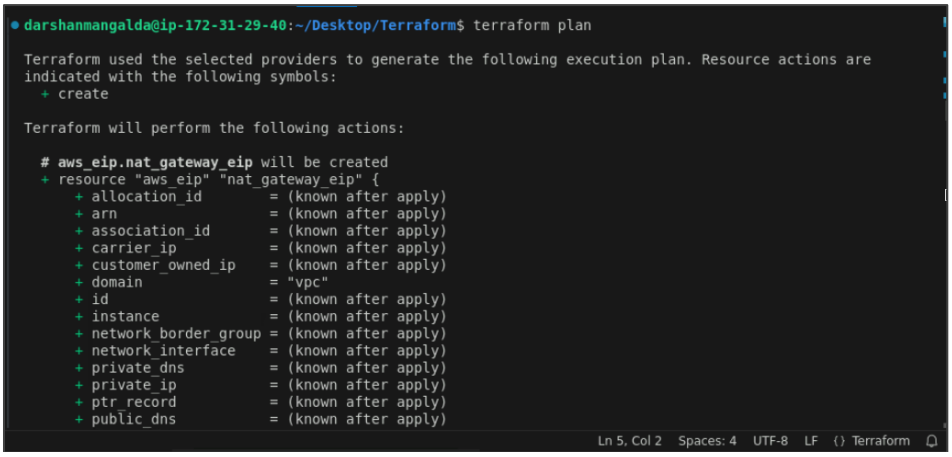
3.2 Run the following command to apply the configuration:
terraform apply -auto-approve



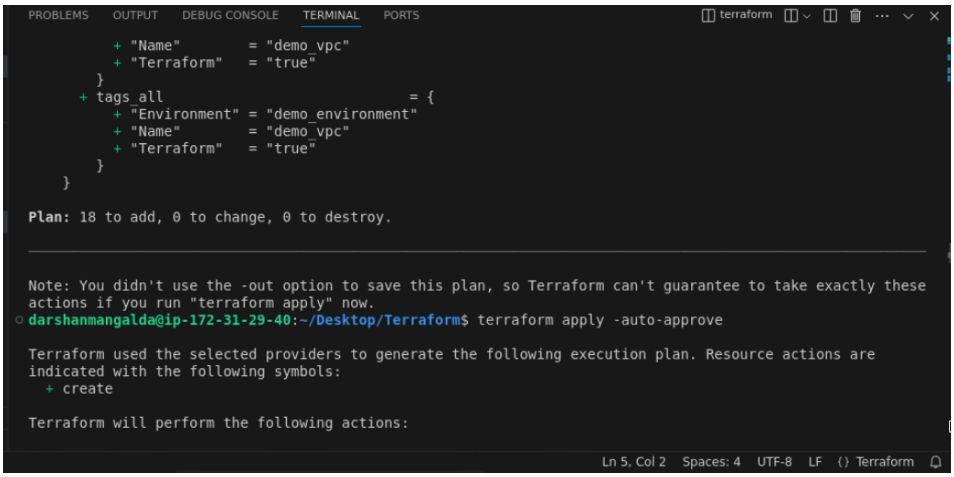
Step 4: Validate the Terraform configuration

4.1 Run the following command to validate the configuration:
terraform validate

4.2 Execute the following command to create an execution plan:
terraform plan

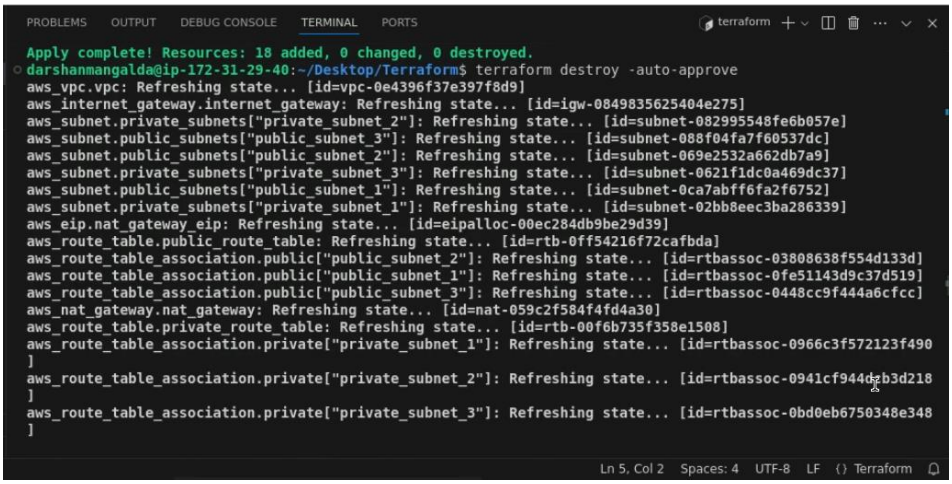


4.3 Enter the following command to execute it:
terraform apply -auto-approve



Step 5: Clean up AWS resources

5.1 To destroy your resources, execute the following command in the terminal:
terraform destroy -auto-approve



By following these steps, you have successfully deployed and managed a resilient VPC with public and private subnets across multiple availability zones in AWS. You also demonstrated the use of HCL functions for string manipulation, collections, and encoding using Terraform. Finally, you validated the Terraform configuration and efficiently managed the lifecycle of the AWS resources, including their termination.

This project consolidates various aspects of Terraform and AWS infrastructure management, providing a comprehensive understanding of infrastructure as code (IaC) practices.