

## 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

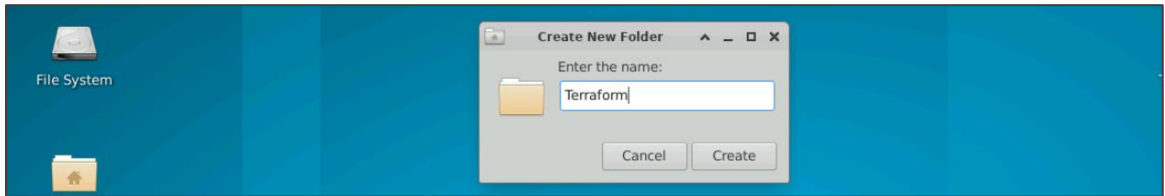
**Expected Deliverables:** Terraform configuration files for deploying a resilient AWS infrastructure with VPC, public and private subnets, Internet Gateway, and NAT Gateway. This includes HCL functions for string manipulation and collections, with clear documentation of installation and execution steps.

Steps to be followed:

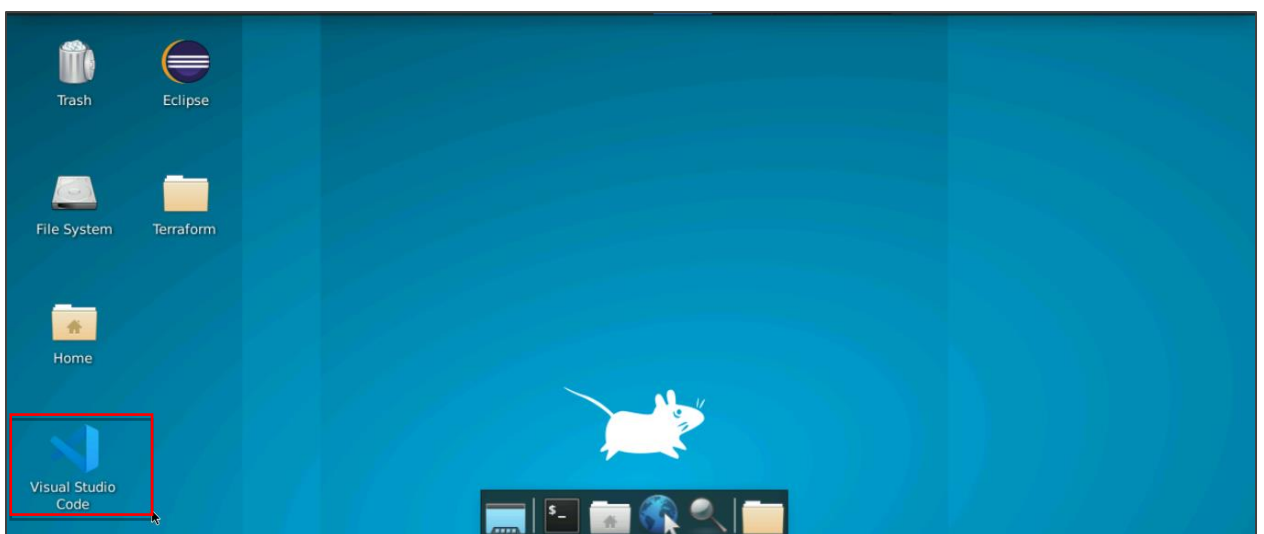
1. Prepare files and set up Terraform
2. Deploy the AWS infrastructure
3. Implement HCL functions for string manipulation, collections, and encoding
4. Validate the Terraform configuration
5. Clean up AWS resources

## Step 1: Prepare files and set up Terraform

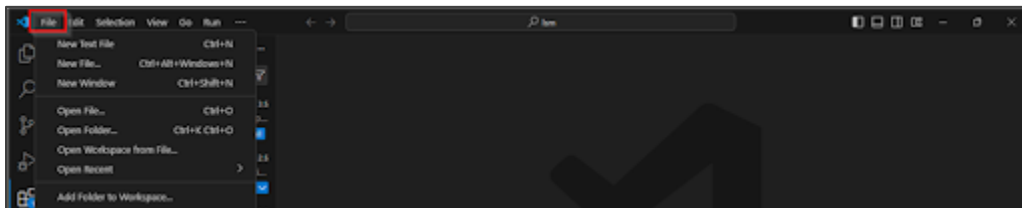
### 1.1 Create a folder named **Terraform** on the desktop



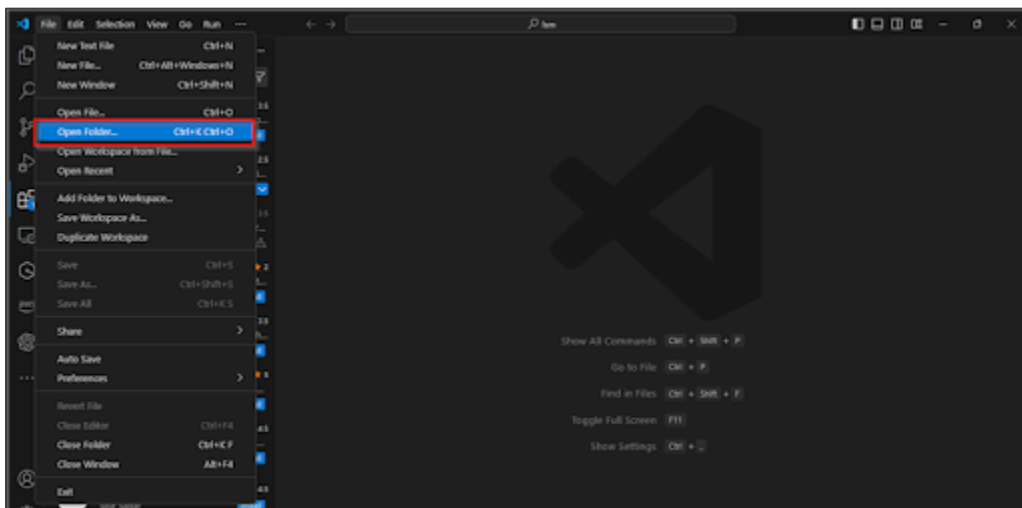
### 1.2 Open the **VS Code** editor



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:

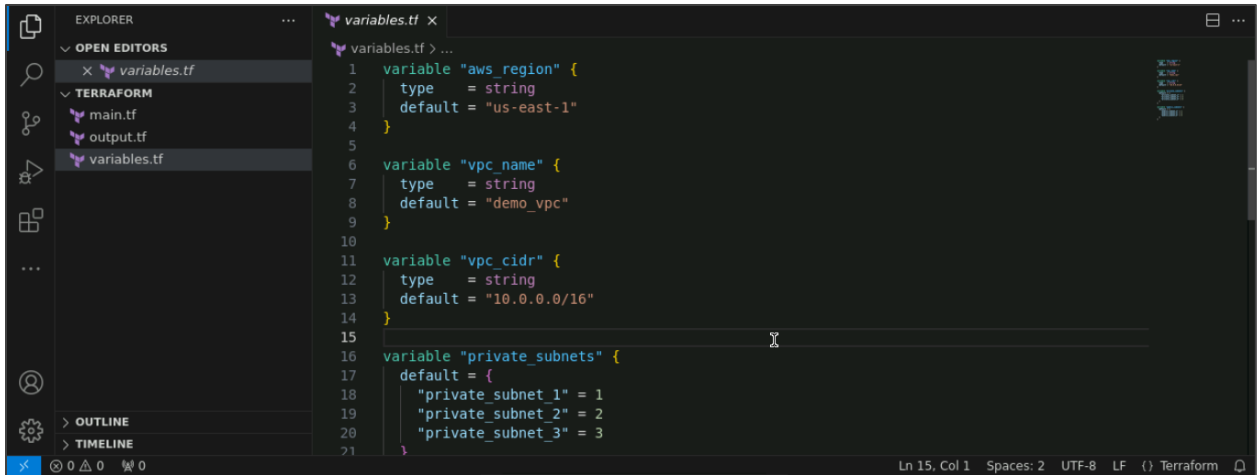
```
variable "aws_region" {  
  type    = string  
  default = "us-east-1"  
}
```

```
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"  
  }  
}
```

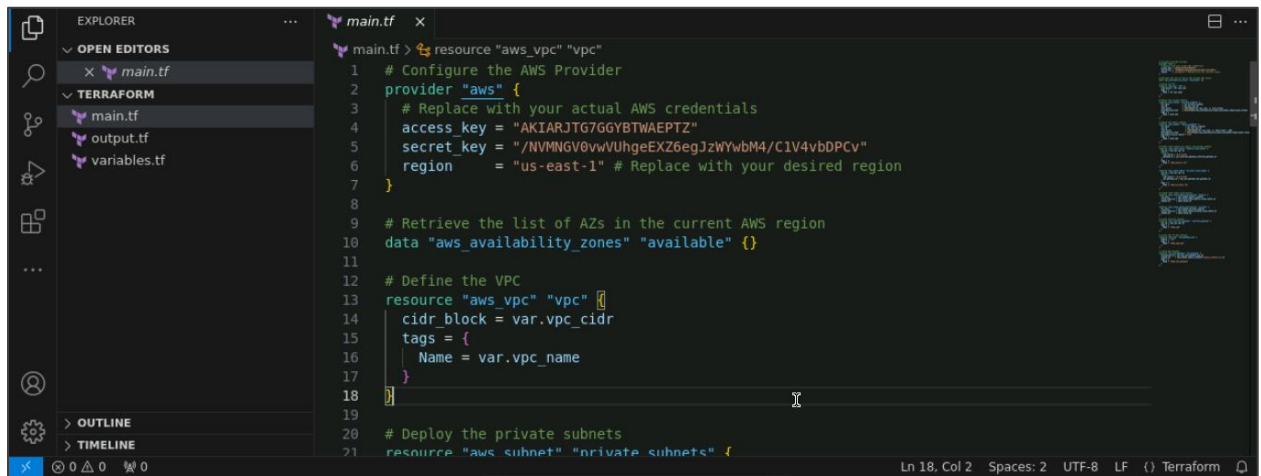
**# Create EIP for NAT Gateway**

```
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"  
  }  
}
```

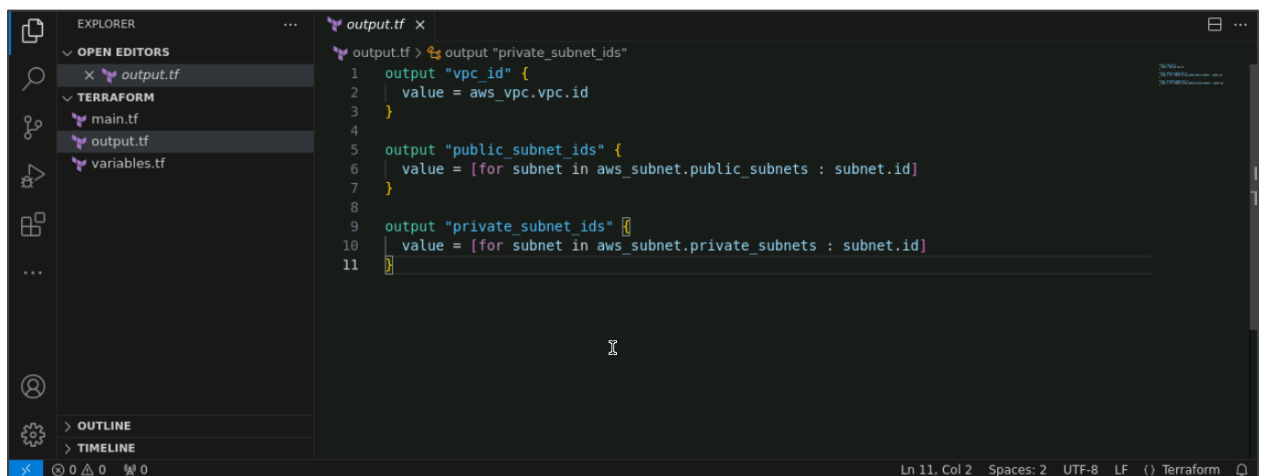
```
}  
}  
}
```



```
main.tf  
1 # Configure the AWS Provider  
2 provider "aws" {  
3   # Replace with your actual AWS credentials  
4   access_key = "AKIARJTG7GGYBTWAEPTZ"  
5   secret_key = "/NVMNGV0vwVUhgEXZ6egJzWYwbM4/C1V4vbDPCv"  
6   region     = "us-east-1" # Replace with your desired region  
7 }  
8  
9 # Retrieve the list of AZs in the current AWS region  
10 data "aws_availability_zones" "available" {}  
11  
12 # Define the VPC  
13 resource "aws_vpc" "vpc" {  
14   cidr_block = var.vpc_cidr  
15   tags = {  
16     Name = var.vpc_name  
17   }  
18 }  
19  
20 # Deploy the private subnets  
21 resource "aws_subnet" "private_subnets" {
```

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

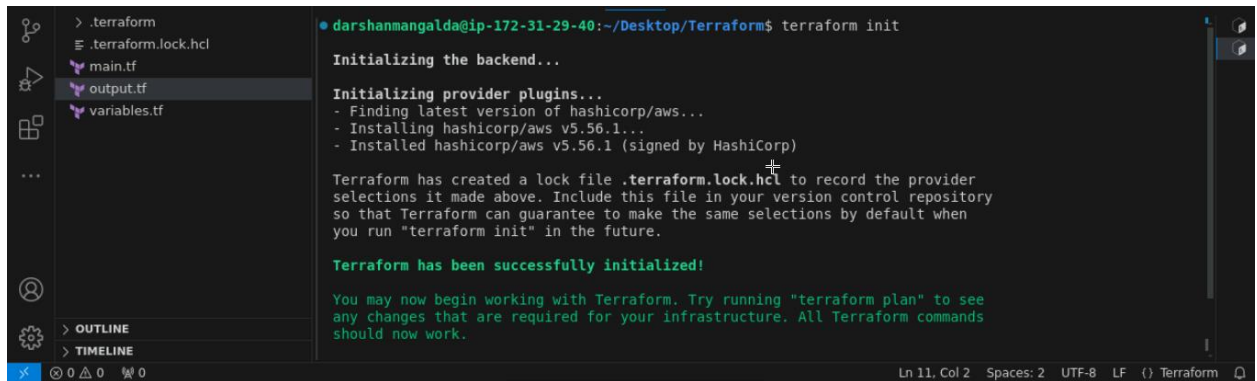
```
output "vpc_id" {  
  value = aws_vpc.vpc.id  
}  
output "public_subnet_ids"  
{  
  value = [for subnet in aws_subnet.public_subnets : subnet.id]  
}  
output "private_subnet_ids" {  
  value = [for subnet in aws_subnet.private_subnets : subnet.id]  
}
```



```
output.tf  
1 output "private_subnet_ids"  
2   output "vpc_id" {  
3     value = aws_vpc.vpc.id  
4   }  
5   output "public_subnet_ids" {  
6     value = [for subnet in aws_subnet.public_subnets : subnet.id]  
7   }  
8   output "private_subnet_ids" {  
9     value = [for subnet in aws_subnet.private_subnets : subnet.id]  
10  }  
11 }
```



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



```
darshanmangalada@ip-172-31-29-40:~/Desktop/Terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.56.1...
- Installed hashicorp/aws v5.56.1 (signed by HashiCorp)

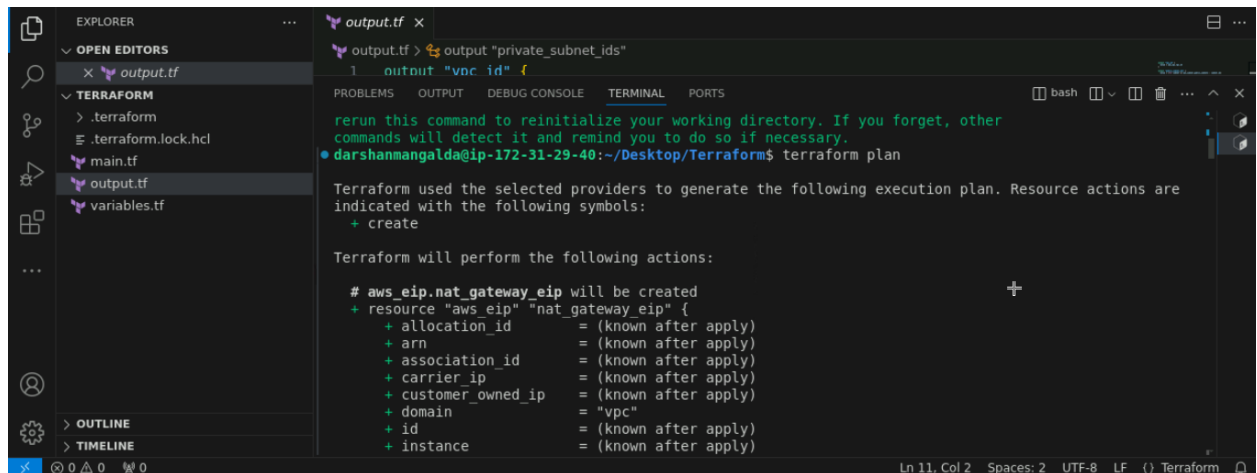
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.
```

## Step 2: Deploy the AWS infrastructure

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



```
output.tf x
output.tf > output "private_subnet_ids"
1 output "vpc_id" {

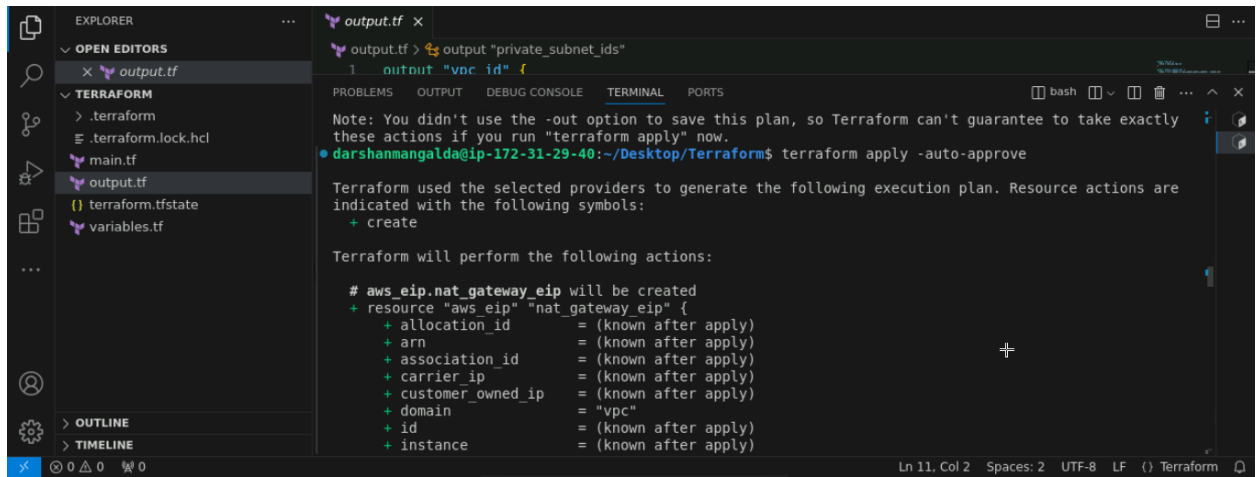
darshanmangalada@ip-172-31-29-40:~/Desktop/Terraform$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_eip.nat_gateway_eip will be created
+ resource "aws_eip" "nat_gateway_eip" {
  + allocation_id      = (known after apply)
  + arn                 = (known after apply)
  + association_id     = (known after apply)
  + carrier_ip         = (known after apply)
  + customer_owned_ip  = (known after apply)
  + domain             = "vpc"
  + id                 = (known after apply)
  + instance            = (known after apply)
```

2.2 Execute the following command to apply the configuration:  
**terraform apply -auto-approve**



```
output.tf x
output.tf > output "private_subnet_ids"
1  output "vpc_id" {

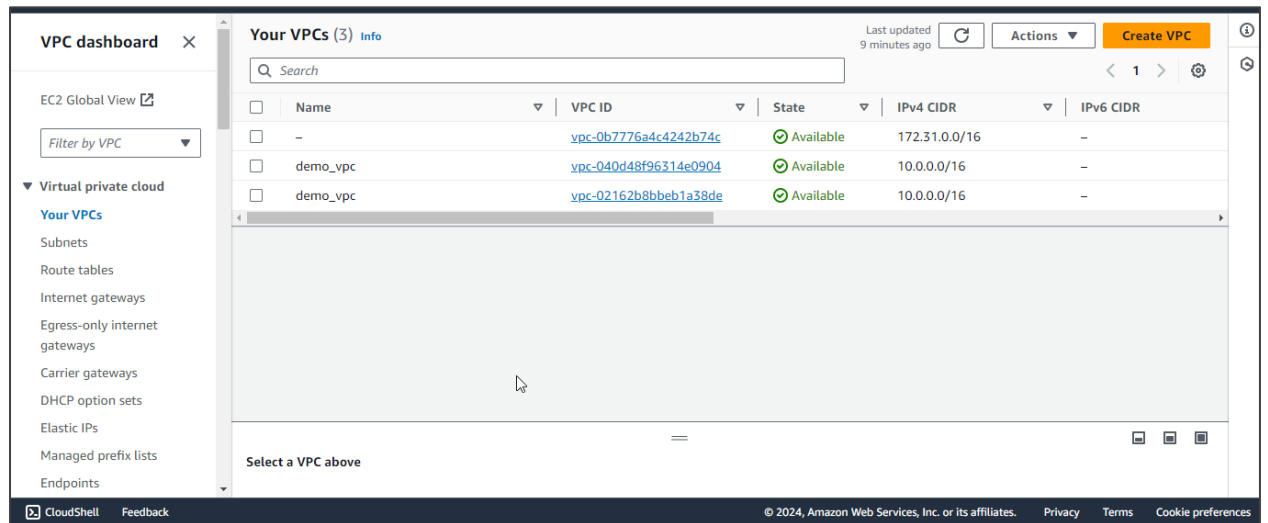
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly
these actions if you run "terraform apply" now.
• darshanmangalda@ip-172-31-29-40:~/Desktop/Terraform$ terraform apply -auto-approve

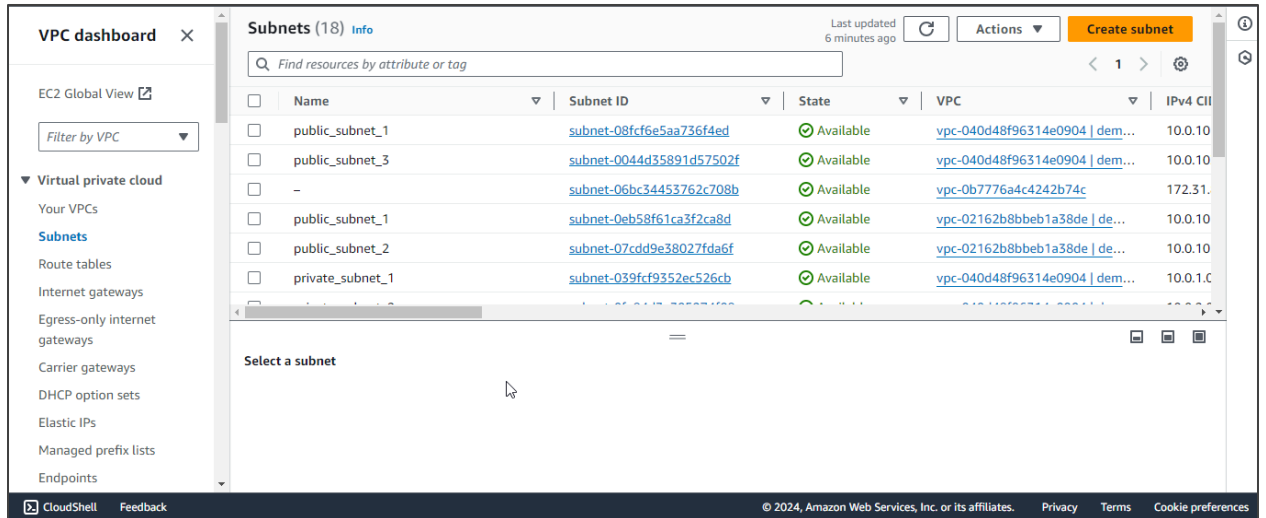
Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_eip.nat_gateway_eip will be created
+ resource "aws_eip" "nat_gateway_eip" {
  + allocation_id      = (known after apply)
  + arn                = (known after apply)
  + association_id     = (known after apply)
  + carrier_ip         = (known after apply)
  + customer_owned_ip  = (known after apply)
  + domain             = "vpc"
  + id                 = (known after apply)
  + instance           = (known after apply)
}
```

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)}"
  }

```

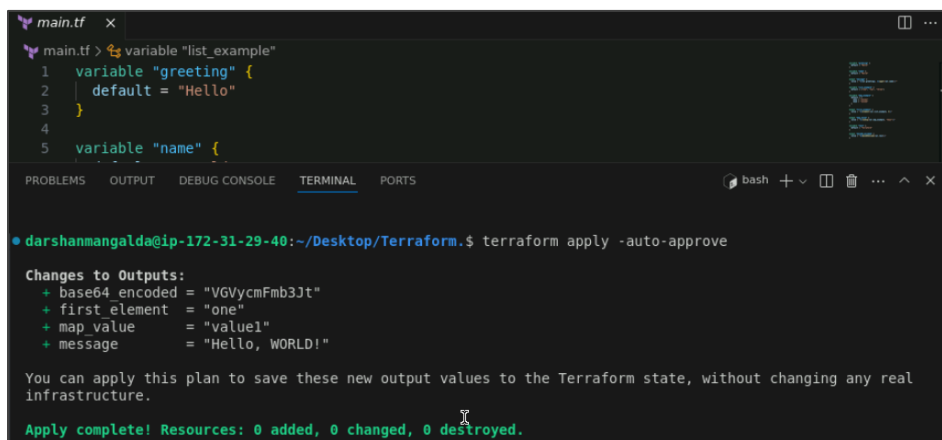


```

main.tf
1  variable "greeting" {
2    default = "Hello"
3  }
4
5  variable "name" {
6    default = "World"
7  }
8
9  output "message" {
10   value = "${var.greeting}, ${upper(var.name)}!"
11 }
12
13 variable "list_example" {
14   default = ["one", "two", "three"]
15 }
16
17 variable "map_example" {
18   default = {
19     key1 = "value1"
20     key2 = "value2"
21   }

```

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



```

main.tf
main.tf > variable "list_example"
1  variable "greeting" {
2    default = "Hello"
3  }
4
5  variable "name" {
6    default = "World"
7  }
8
9  output "message" {
10   value = "${var.greeting}, ${upper(var.name)}!"
11 }
12
13 variable "list_example" {
14   default = ["one", "two", "three"]
15 }
16
17 variable "map_example" {
18   default = {
19     key1 = "value1"
20     key2 = "value2"
21   }

```

```

darshanmangal@ip-172-31-29-40:~/Desktop/Terraform$ terraform apply -auto-approve

Changes to Outputs:
+ base64_encoded = "VG95cmFmb3Jt"
+ first_element  = "one"
+ map_value      = "value1"
+ message        = "Hello, WORLD!"

You can apply this plan to save these new output values to the Terraform state, without changing any real infrastructure.

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

```

## Step 4: Validate the Terraform configuration

4.1 Run the following command to validate the configuration:

**terraform validate**

```
Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
● darshanmangalda@ip-172-31-29-40:~/Desktop/Terraform$ terraform validate
Success! The configuration is valid.
○ darshanmangalda@ip-172-31-29-40:~/Desktop/Terraform$
```

4.2 Execute the following command to create an execution plan:

**terraform plan**

```
● darshanmangalda@ip-172-31-29-40:~/Desktop/Terraform$ terraform plan

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_eip.nat_gateway_eip will be created
+ resource "aws_eip" "nat_gateway_eip" {
+   allocation_id      = (known after apply)
+   arn                 = (known after apply)
+   association_id     = (known after apply)
+   carrier_ip         = (known after apply)
+   customer_owned_ip  = (known after apply)
+   domain              = "vpc"
+   id                  = (known after apply)
+   instance            = (known after apply)
+   network_border_group = (known after apply)
+   network_interface  = (known after apply)
+   private_dns         = (known after apply)
+   private_ip          = (known after apply)
+   ptr_record          = (known after apply)
+   public_dns          = (known after apply)
}
```

4.3 Enter the following command to execute it:

**terraform apply -auto-approve**

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ "Name"      = "demo_vpc"
+ "Terraform" = "true"
}
+ tags_all = {
+   "Environment" = "demo_environment"
+   "Name"        = "demo_vpc"
+   "Terraform"   = "true"
}
}

Plan: 18 to add, 0 to change, 0 to destroy.

Note: You didn't use the -out option to save this plan, so Terraform can't guarantee to take exactly these
actions if you run "terraform apply" now.
○ darshanmangalda@ip-172-31-29-40:~/Desktop/Terraform$ terraform apply -auto-approve

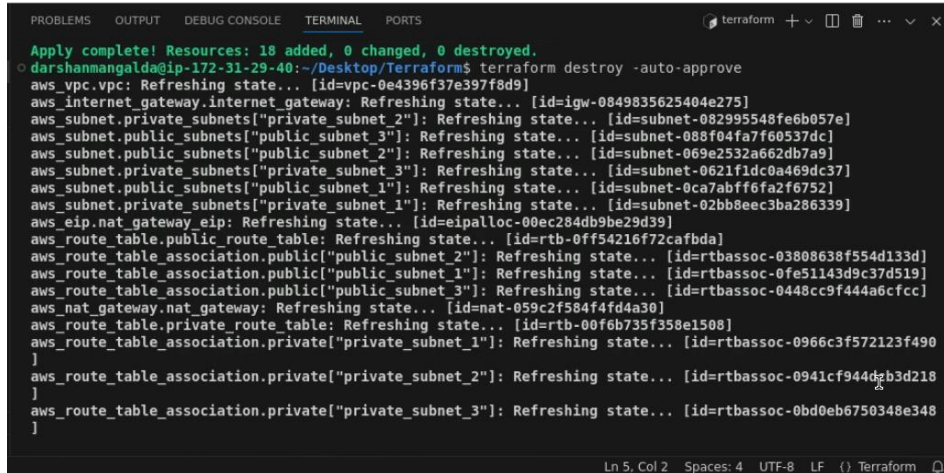
Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
+ create

Terraform will perform the following actions:
```

## Step 5: Clean up AWS resources

5.1 To destroy your resources, execute the following command in the terminal:

**terraform destroy -auto-approve**



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Apply complete! Resources: 18 added, 0 changed, 0 destroyed.
darshanmangal@ip-172-31-29-40: ~/Desktop/Terraform$ terraform destroy -auto-approve
aws_vpc.vpc: Refreshing state... [id=vpc-0e4396f37e397f8d9]
aws_internet_gateway.internet_gateway: Refreshing state... [id=igw-0849035625404e275]
aws_subnet.private_subnets["private_subnet_2"]: Refreshing state... [id=subnet-082995548fe6b057e]
aws_subnet.public_subnets["public_subnet_3"]: Refreshing state... [id=subnet-088f04fa7f60537dc]
aws_subnet.public_subnets["public_subnet_2"]: Refreshing state... [id=subnet-069e2532a662db7a9]
aws_subnet.private_subnets["private_subnet_3"]: Refreshing state... [id=subnet-0621f1dc0a469dc37]
aws_subnet.public_subnets["public_subnet_1"]: Refreshing state... [id=subnet-0ca7abff6fa2f6752]
aws_subnet.private_subnets["private_subnet_1"]: Refreshing state... [id=subnet-02bb8eec3ba286339]
aws_eip.nat_gateway_eip: Refreshing state... [id=eipalloc-00ec284db9be29d39]
aws_route_table.public_route_table: Refreshing state... [id=rtb-0ff54216f72cafbda]
aws_route_table_association.public["public_subnet_2"]: Refreshing state... [id=rtbassoc-03808638f554d133d]
aws_route_table_association.public["public_subnet_1"]: Refreshing state... [id=rtbassoc-0fe51143d9c37d519]
aws_route_table_association.public["public_subnet_3"]: Refreshing state... [id=rtbassoc-0448cc9f444a6cfc]
aws_nat_gateway.nat_gateway: Refreshing state... [id=nat-059c2f584f4fd4a30]
aws_route_table.private_route_table: Refreshing state... [id=rtb-00f6b735f358e1508]
aws_route_table_association.private["private_subnet_1"]: Refreshing state... [id=rtbassoc-0966c3f572123f490]
aws_route_table_association.private["private_subnet_2"]: Refreshing state... [id=rtbassoc-0941cf944dzb3d218]
aws_route_table_association.private["private_subnet_3"]: Refreshing state... [id=rtbassoc-0bd0eb6750348e348]
```

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.