Lesson-End Project

Implementing and Managing Terraform Configurations

Project agenda: To deploy and manage a scalable web application on AWS infrastructure using Terraform for enhanced security, automation, and operational efficiency

Description: Imagine you are a cloud engineer tasked with implementing and managing AWS infrastructure using Terraform. The project involves setting up and initializing a Terraform configuration, defining and utilizing variables, locals, and outputs, and implementing resources with these variables and locals. Additionally, you will secure and manage sensitive data, work with collections and structure types, utilize Terraform's built-in functions and dynamic blocks, and generate and visualize a resource graph. This project aims to provide a comprehensive understanding of Terraform configuration and management practices, reinforcing key concepts and best practices.

Tools required: AWS Account, Terraform, and VS Code

Prerequisites: Refer to **Demo 01** of **Lesson 11** for creating access and secret key

Expected deliverables: A fully deployed AWS web application infrastructure using Terraform with initialized configurations, security settings for sensitive data, and a visual resource dependency graph

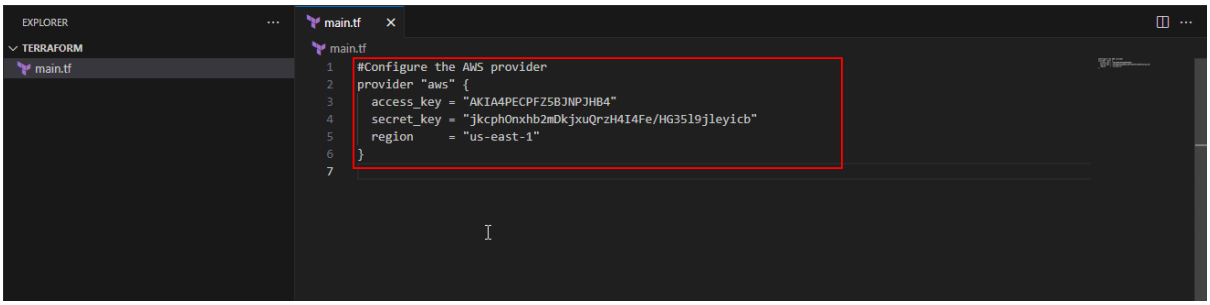
Steps to be followed:

1. Set up and initialize Terraform configuration
2. Define variables, locals, and outputs
3. Implement resources with variables and locals
4. Secure and manage sensitive data
5. Utilize collections and structure types
6. Utilize Terraform built-in functions and dynamic blocks
7. Generate and visualize the resource graph

Step 1: Set up and initialize Terraform configuration

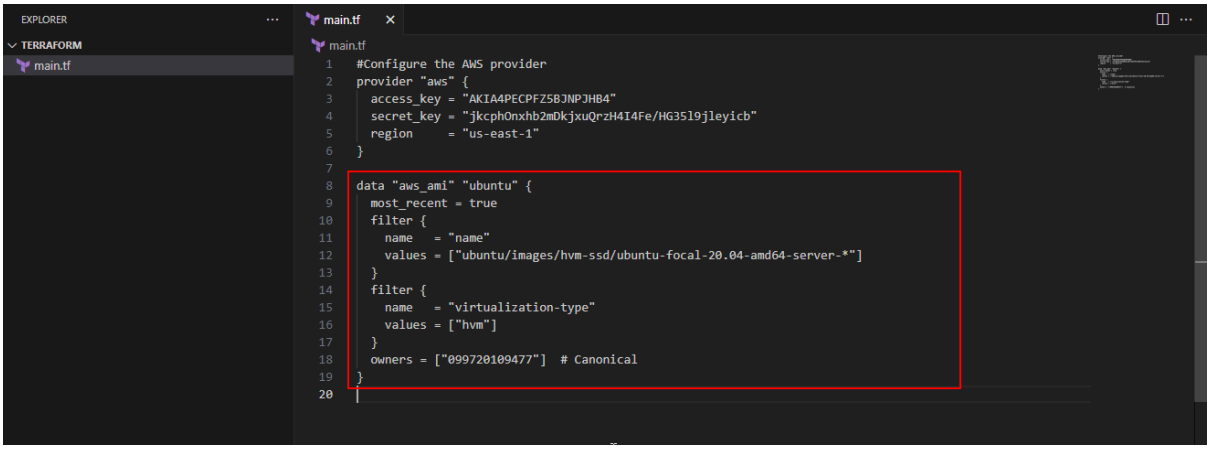
1.1 Open your Terraform configuration environment and create a file named **main.tf**. Add the following configuration block as shown in the screenshot below:

```
#Configure the AWS provider
provider "aws" {
 # Replace with your actual AWS credentials
 access_key = "YOUR_ACCESS_KEY"
 secret_key = "YOUR_SECRET_KEY"
 region     = "us-east-1" # Replace with your desired region
}
```



1.2 Add the AWS AMI data source as shown in the screenshot below:

```
data "aws_ami" "ubuntu" {
 most_recent = true
 filter {
  name   = "name"
  values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-server-*"]
 }
 filter {
  name   = "virtualization-type"
  values = ["hvm"]
 }
 owners = ["099720109477"]  # Canonical
}
```

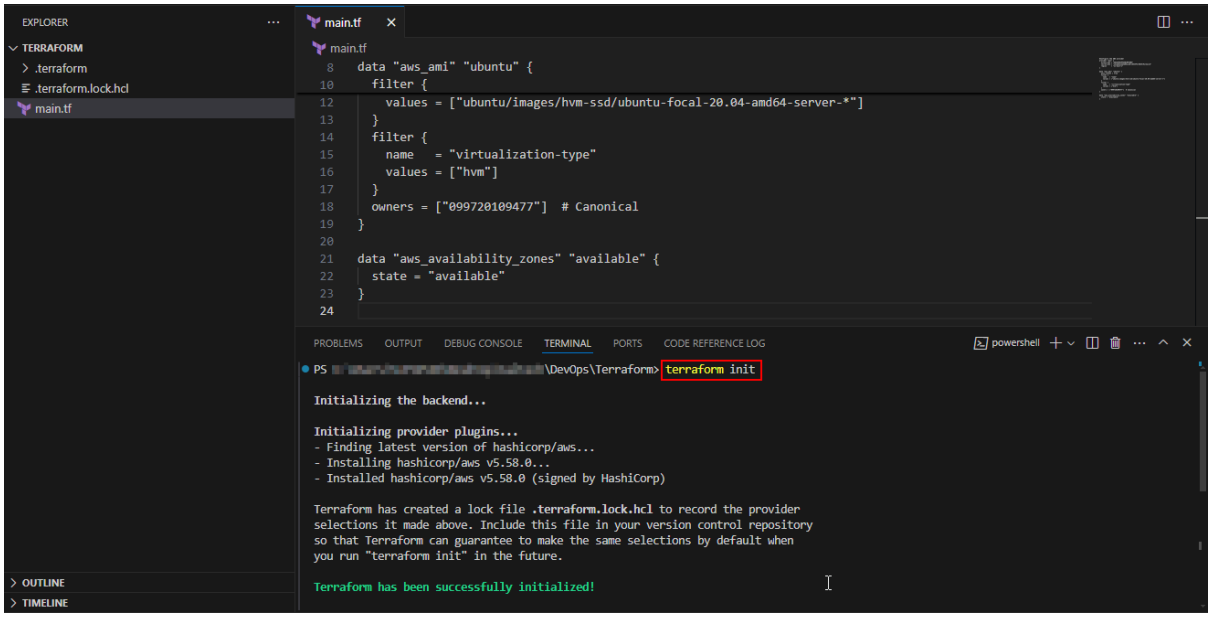


1.3 Declare the AWS availability zones data source as shown in the screenshot below:

data "aws_availability_zones" "available" {
state = "available"
}

1.4 Initialize the Terraform project using the following command to set up the necessary plugins as shown in the screenshot below:

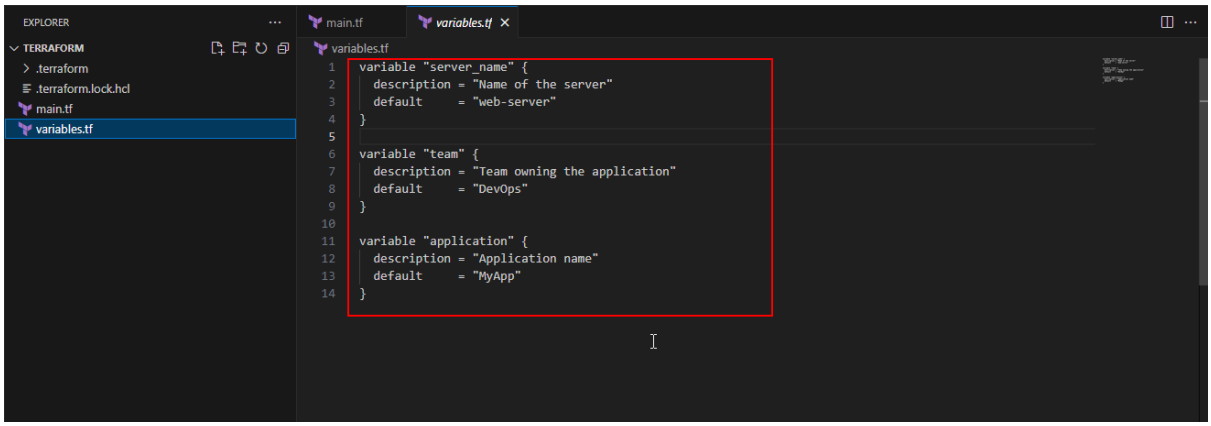
terraform init



Step 2: Define variables, locals, and outputs

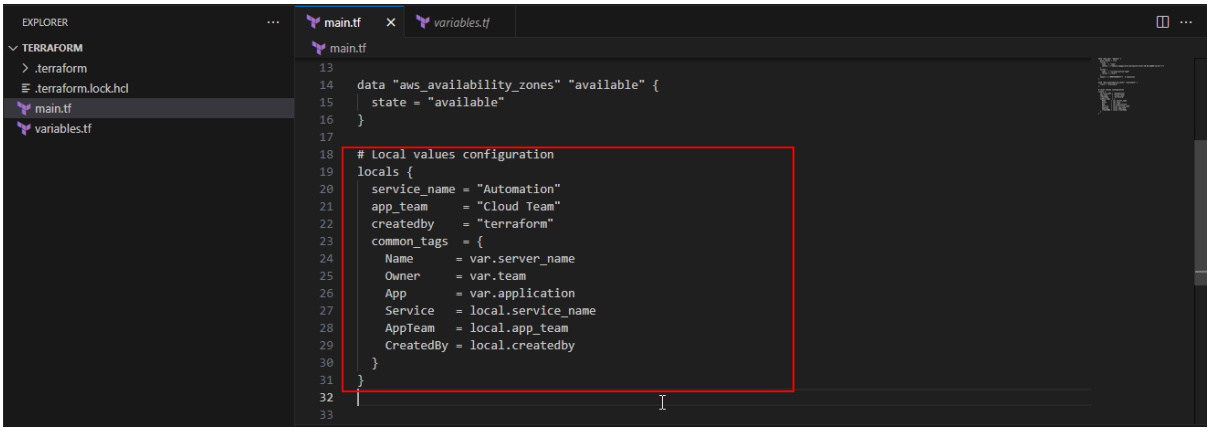
2.1 Create a file named **variables.tf** and define the basic variables as shown in the screenshot below:

```
variable "server_name" {
  description = "Name of the server"
  default     = "web-server"
}

variable "team" {
  description = "Team owning the application"
  default     = "DevOps"
}

variable "application" {
  description = "Application name"
  default     = "WebApp"
}
```



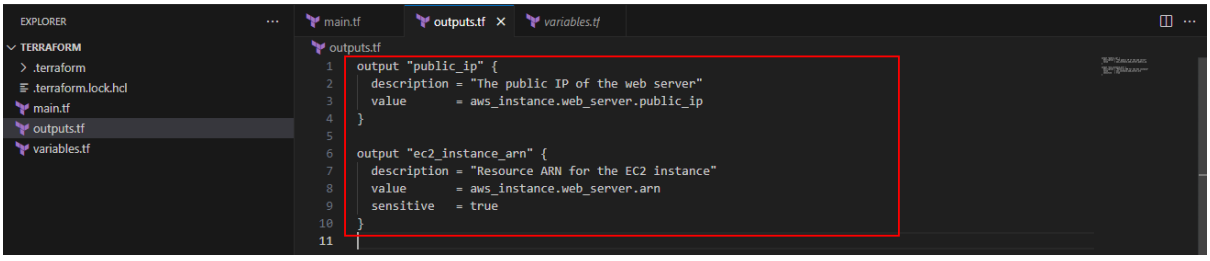
2.2 Add the local values in the **main.tf** file as shown in the screenshot below:

```
# Local values configuration
locals {
 service_name = "Automation"
 app_team     = "Cloud Team"
 createdby    = "terraform"
 common_tags  = {
  Name      = var.server_name
  Owner     = var.team
  App       = var.application
  Service   = local.service_name
  AppTeam   = local.app_team
  CreatedBy = local.createdby
 }
}
```



2.3 Create an **outputs.tf** file and add outputs as shown in the screenshot below:

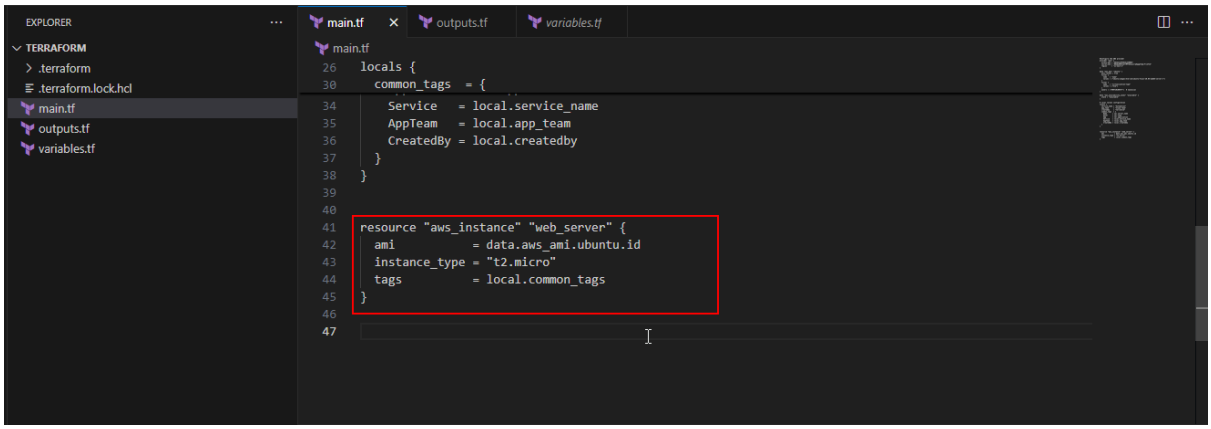
```
output "public_ip" {
 description = "The public IP of the web server"
 value       = aws_instance.web_server.public_ip
}

output "ec2_instance_arn" {
 description = "Resource ARN for the EC2 instance"
 value       = aws_instance.web_server.arn
 sensitive   = true
}
```



Step 3: Implement resources with variables and locals

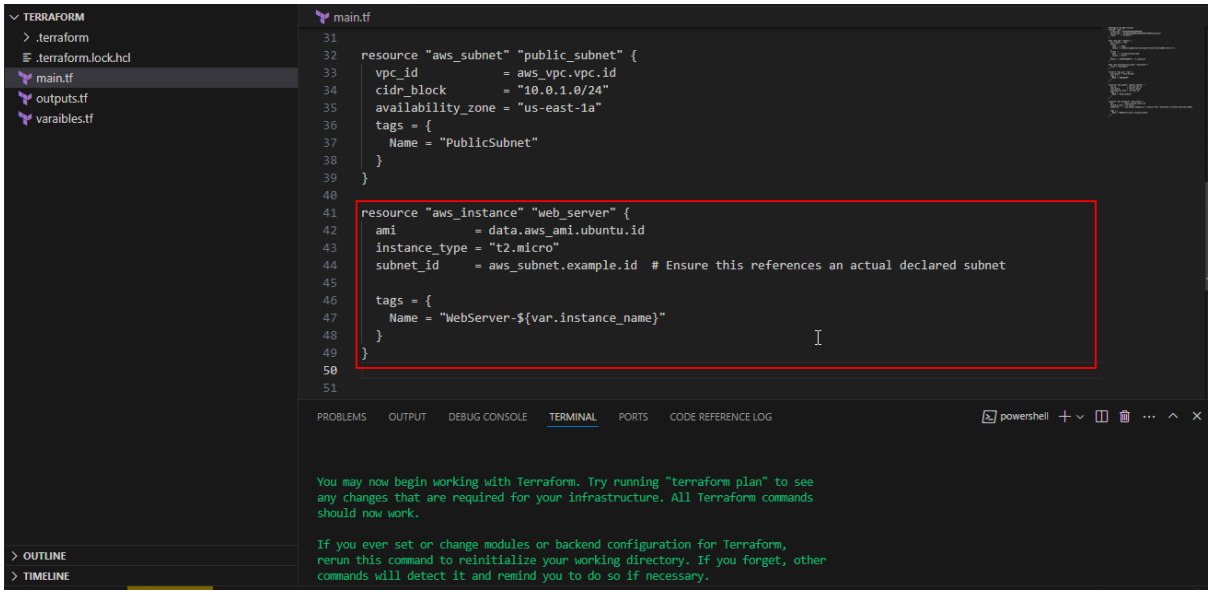
3.1 Add an AWS instance resource in **main.tf** as shown in the screenshot below:

```
resource "aws_instance" "web_server" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t2.micro"
  tags          = local.common_tags
}
```



3.2 Provision an EC2 instance using the following resource block as shown in the screenshot below:

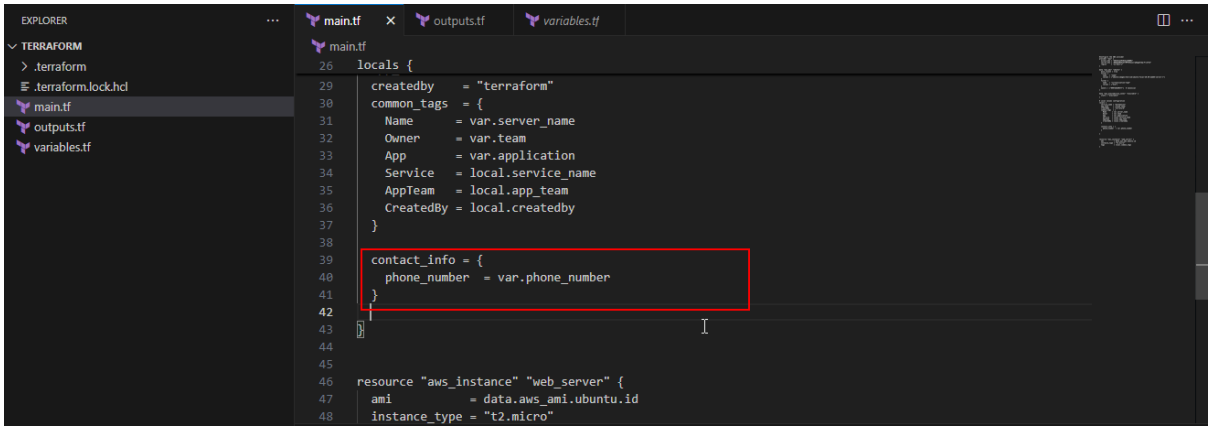
```
resource "aws_instance" "web_server" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t2.micro"
  tags          = local.common_tags
}
```



Step 4: Secure and manage sensitive data

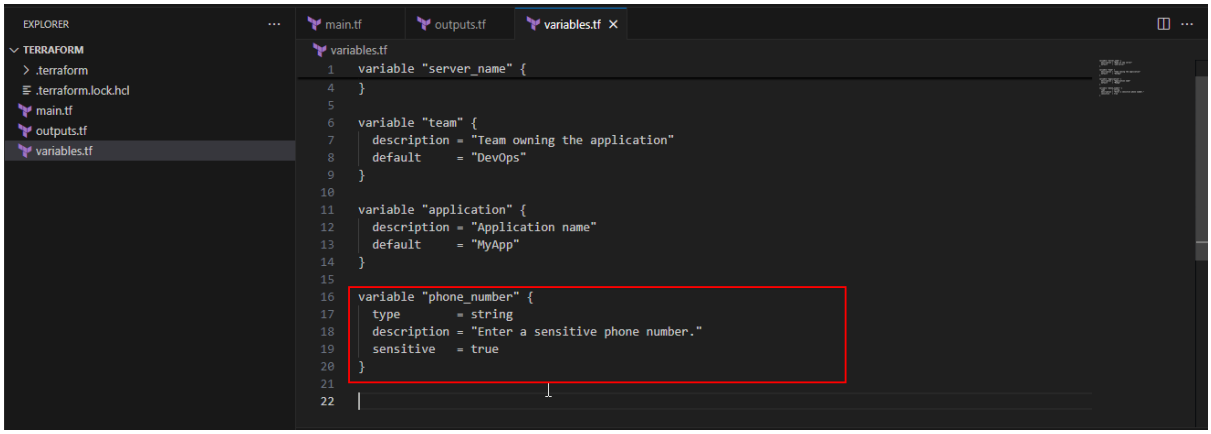
4.1 Add sensitive information such as phone number inside the **locals** block in **main.tf** as shown in the screenshot below:

```
contact_info = {
  phone_number  = var.phone_number
}
```



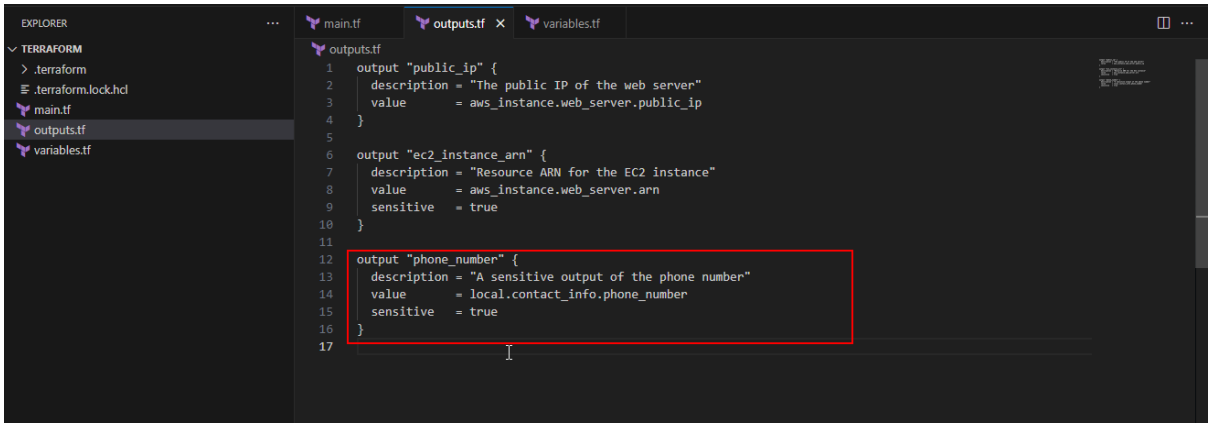
4.2 Add the variable for contact information as shown in the screenshot below:

```
variable "phone_number" {
  type        = string
  description = "Enter a sensitive phone number."
  sensitive   = true
}
```



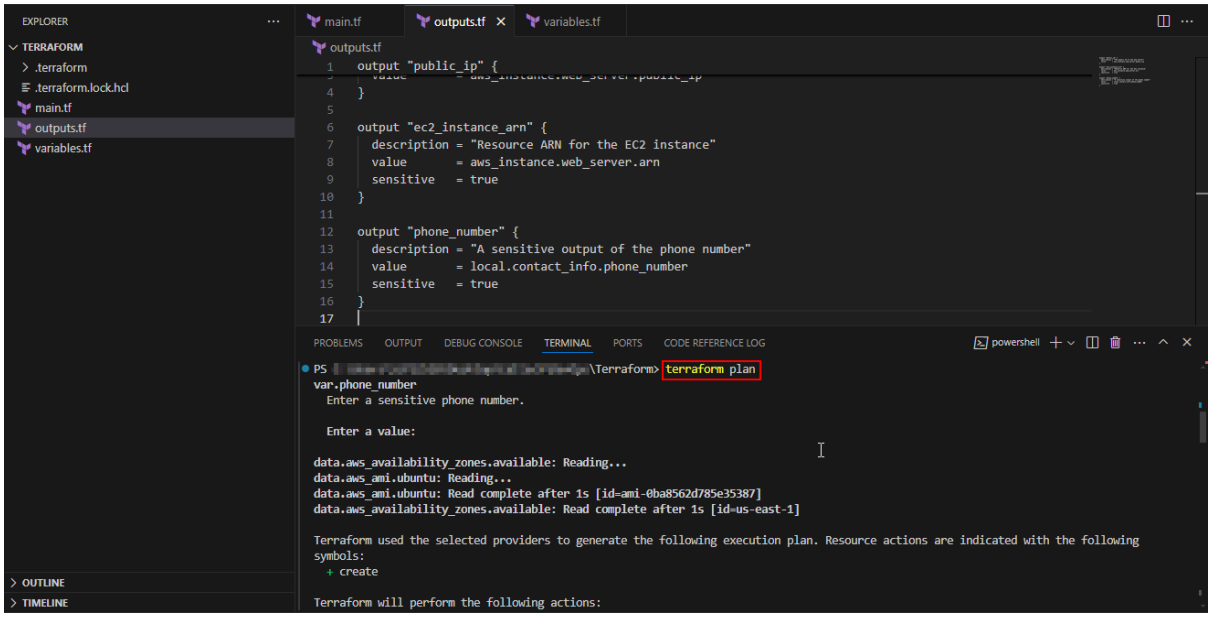
4.3 Add the following **output** block with **sensitive = true** in **output.tf** to securely output the phone number while keeping it hidden from logs:

```
output "phone_number" {
  description = "A sensitive output of the phone number"
  value       = local.contact_info.phone_number
  sensitive   = true
}
```

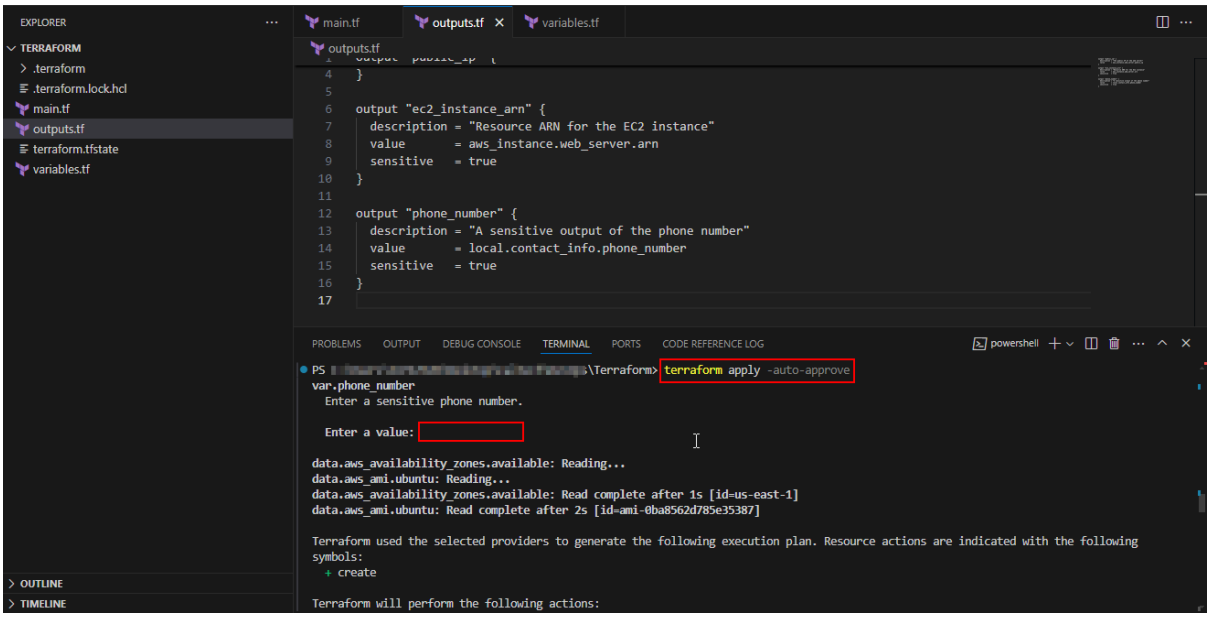


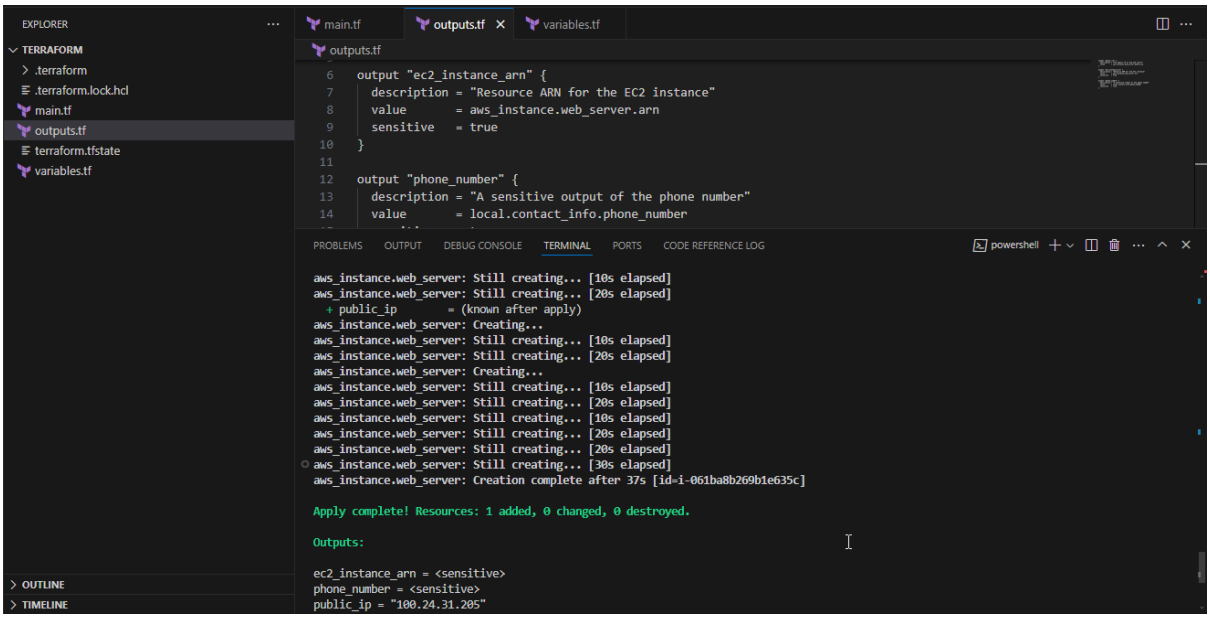
4.4 Execute the following command to preview the proposed changes in the infrastructure:

terraform plan



4.5 Apply the configuration using the following command to deploy the changes and add the desired value for the phone number as shown in the screenshot below:
terraform apply -auto-approve



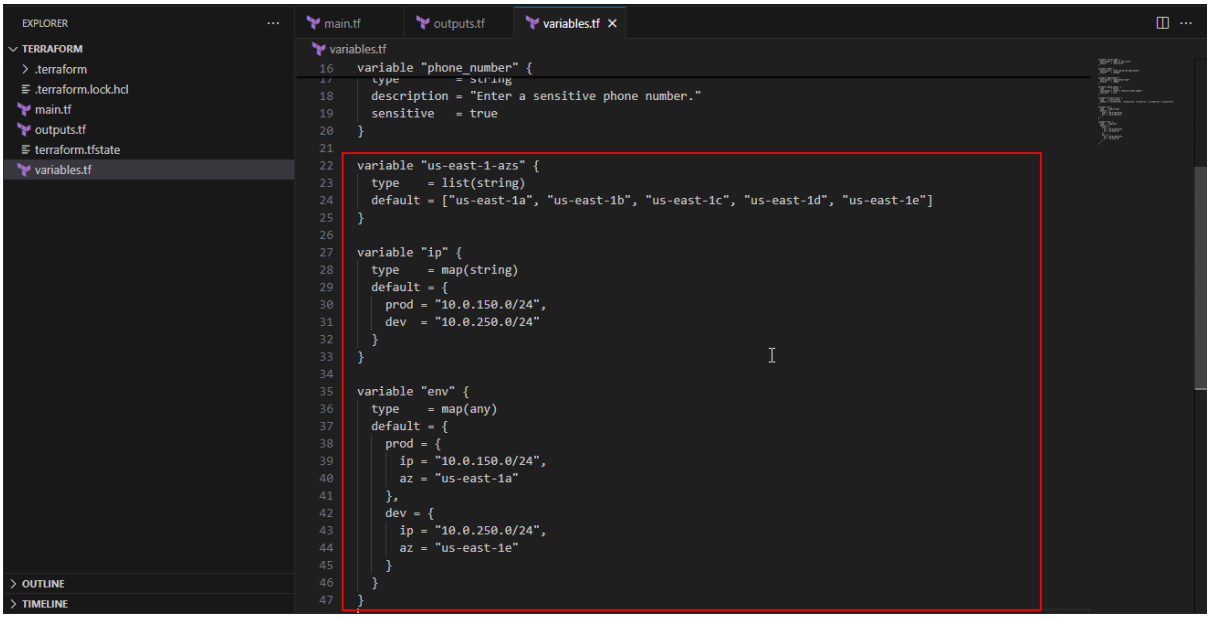


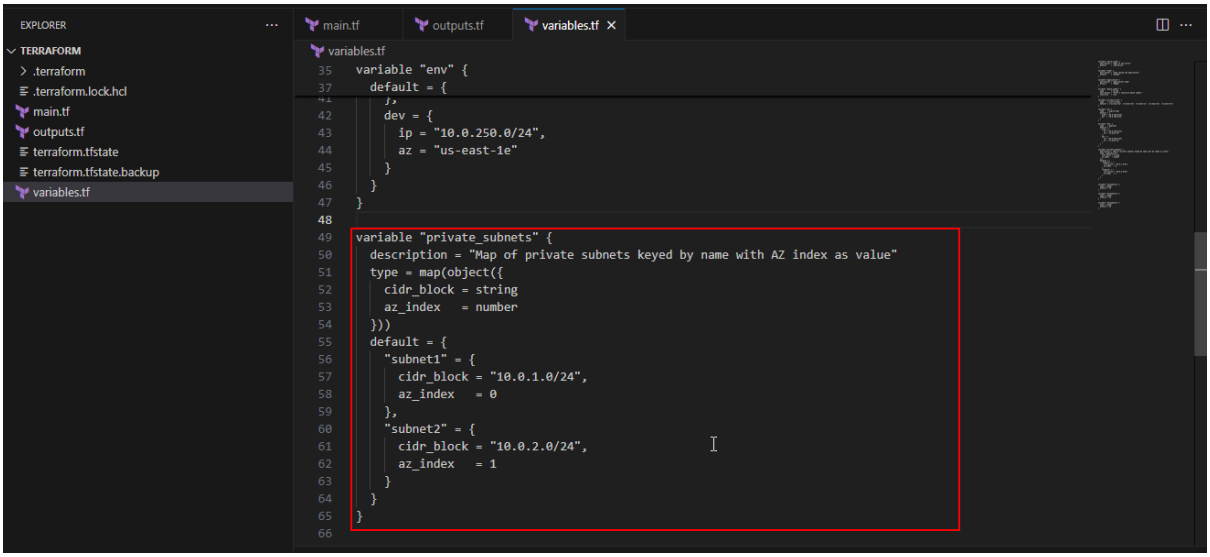
Step 5: Utilize collections and structure types

5.1 Define list and map variables in **variables.tf** as shown in the screenshot below:

```
variable "us-east-1-azs" {
  type    = list(string)
  default = ["us-east-1a", "us-east-1b", "us-east-1c", "us-east-1d", "us-east-1e"]
}

variable "ip" {
  type    = map(string)
  default = {
    prod = "10.0.150.0/24",
    dev  = "10.0.250.0/24"
  }
}

variable "env" {
  type    = map(any)
  default = {
    prod = {
      ip = "10.0.150.0/24",
      az = "us-east-1a"
    },
    dev = {
      ip = "10.0.250.0/24",
      az = "us-east-1e"
    }
  }
}

variable "private_subnets" {
  description = "Map of private subnets keyed by name with AZ index as value"
  type = map(object({
    cidr_block = string
    az_index   = number
  }))
  default = {
    "subnet1" = {
      cidr_block = "10.0.1.0/24",
      az_index   = 0
    },
    "subnet2" = {
      cidr_block = "10.0.2.0/24",
      az_index   = 1
    }
  }
```

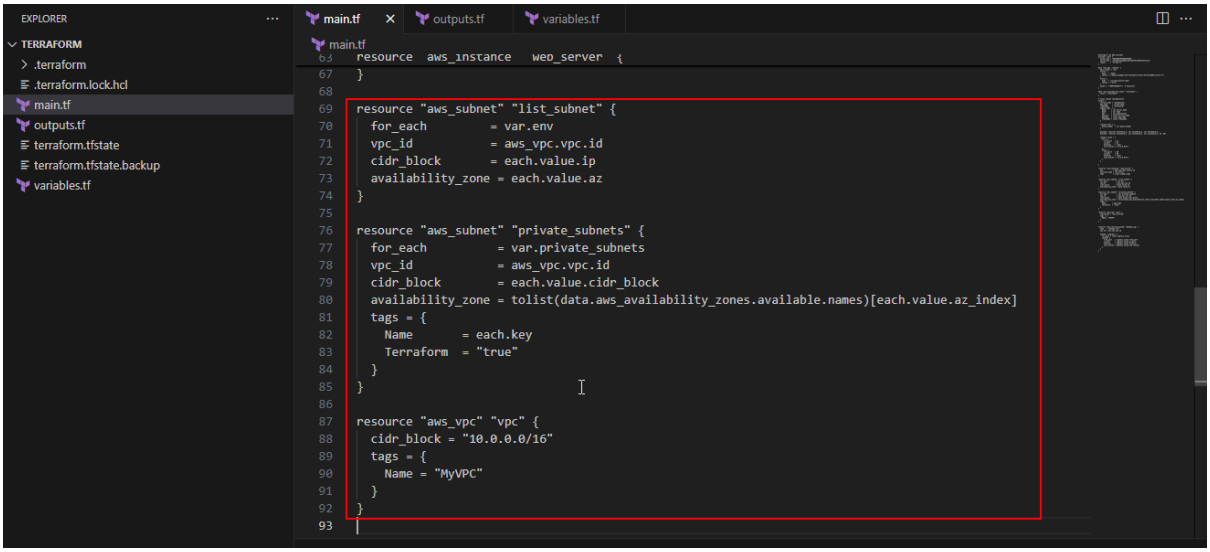
}


EXPLORER
TERRAFORM
.terraform
.terraform.lock.hcl
main.tf
outputs.tf
terraform.tfstate
variables.tf
OUTLINE
TIMELINE
main.tf
outputs.tf
variables.tf
variables.tf
variable "phone_number" {
type = string
description = "Enter a sensitive phone number."
sensitive = true
}
variable "us-east-1-azs" {
type = list(string)
default = ["us-east-1a", "us-east-1b", "us-east-1c", "us-east-1d", "us-east-1e"]
}
variable "ip" {
type = map(string)
default = {
prod = "10.0.150.0/24",
dev = "10.0.250.0/24"
}
}
variable "env" {
type = map(any)
default = {
prod = {
ip = "10.0.150.0/24",
az = "us-east-1a"
},
dev = {
ip = "10.0.250.0/24",
az = "us-east-1e"
}
}
}

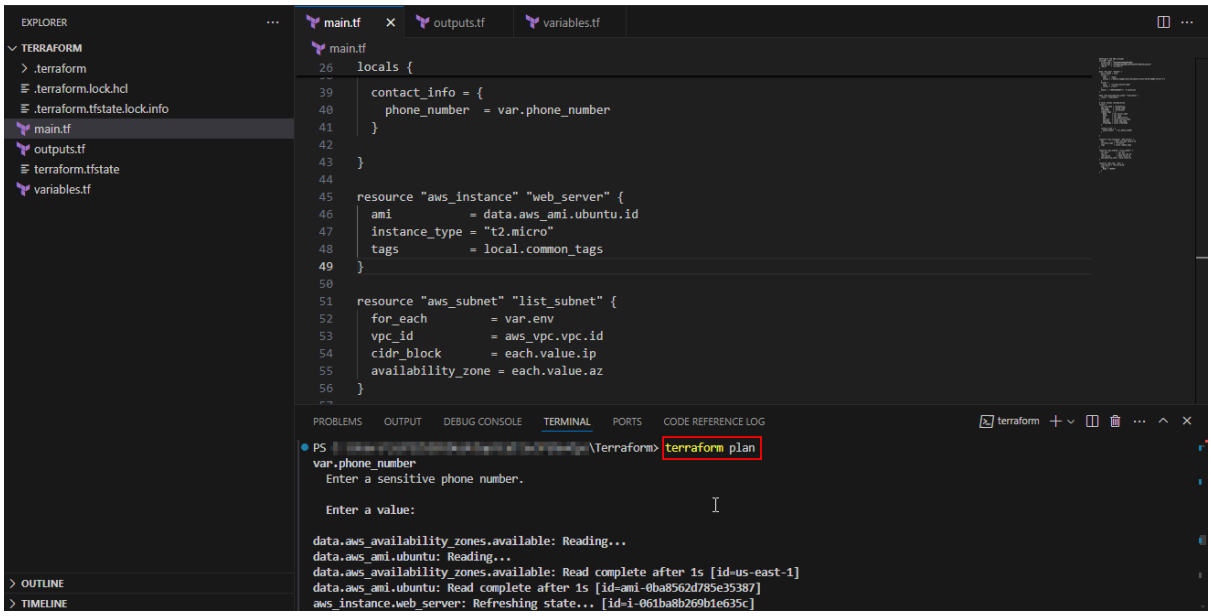

EXPLORER
TERRAFORM
.terraform
.terraform.lock.hcl
main.tf
outputs.tf
terraform.tfstate
terraform.tfstate.backup
variables.tf
main.tf
outputs.tf
variables.tf
variables.tf
variable "env" {
default = {
dev = {
ip = "10.0.250.0/24",
az = "us-east-1e"
}
}
}
variable "private_subnets" {
description = "Map of private subnets keyed by name with AZ index as value"
type = map(object({
cidr_block = string
az_index = number
}))
default = {
"subnet1" = {
cidr_block = "10.0.1.0/24",
az_index = 0
},
"subnet2" = {
cidr_block = "10.0.2.0/24",
az_index = 1
}
}
}

5.2 Iterate over maps using the following resource blocks to create multiple resources in **main.tf** as shown in the screenshot below:

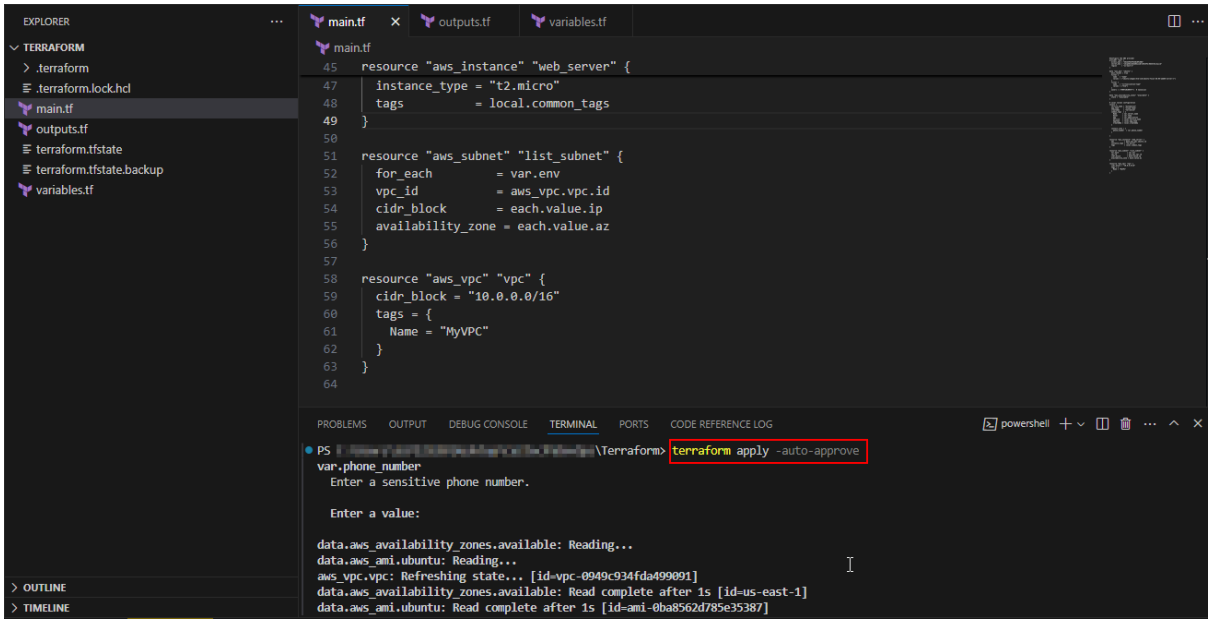
```
resource "aws_subnet" "list_subnet" {
 for_each        = var.env
 vpc_id          = aws_vpc.vpc.id
 cidr_block      = each.value.ip
 availability_zone = each.value.az
}

resource "aws_subnet" "private_subnets" {
 for_each        = var.private_subnets
 vpc_id          = aws_vpc.vpc.id
 cidr_block      = each.value.cidr_block
 availability_zone =
tolist(data.aws_availability_zones.available.names)[each.value.az_index]
 tags = {
   Name      = each.key
   Terraform  = "true"
 }
}

resource "aws_vpc" "vpc" {
 cidr_block = "10.0.0.0/16"
 tags = {
   Name = "MyVPC"
 }
}
```



5.3 Execute the following command to preview the proposed changes in the infrastructure:
terraform plan



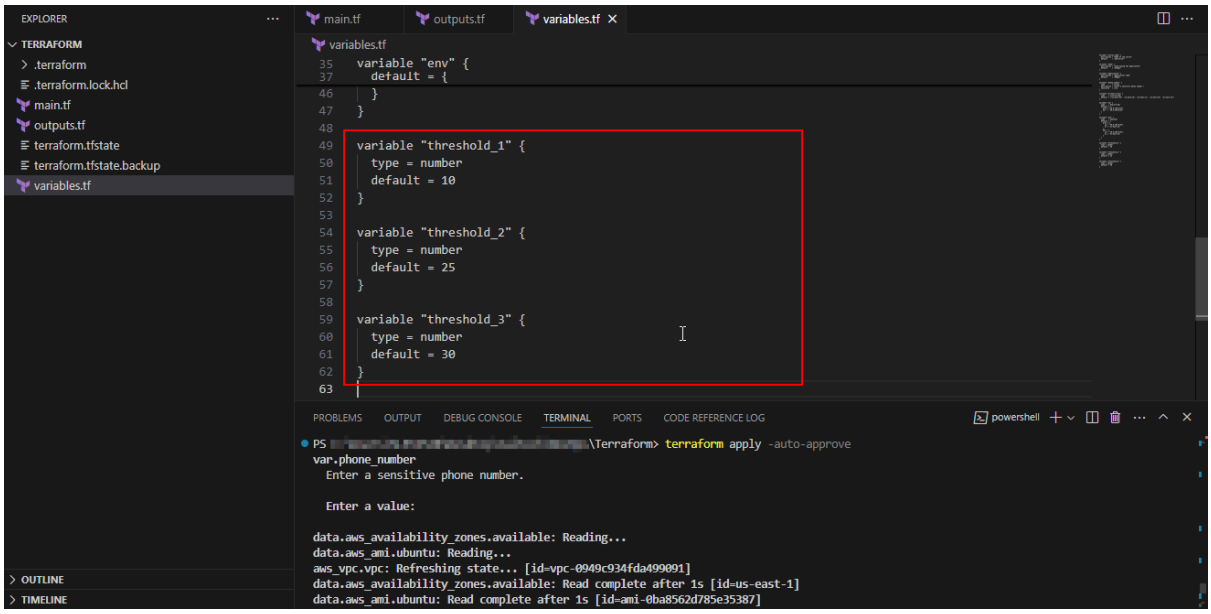
5.4 Apply the configuration using the following command to deploy the changes:
terraform apply -auto-approve



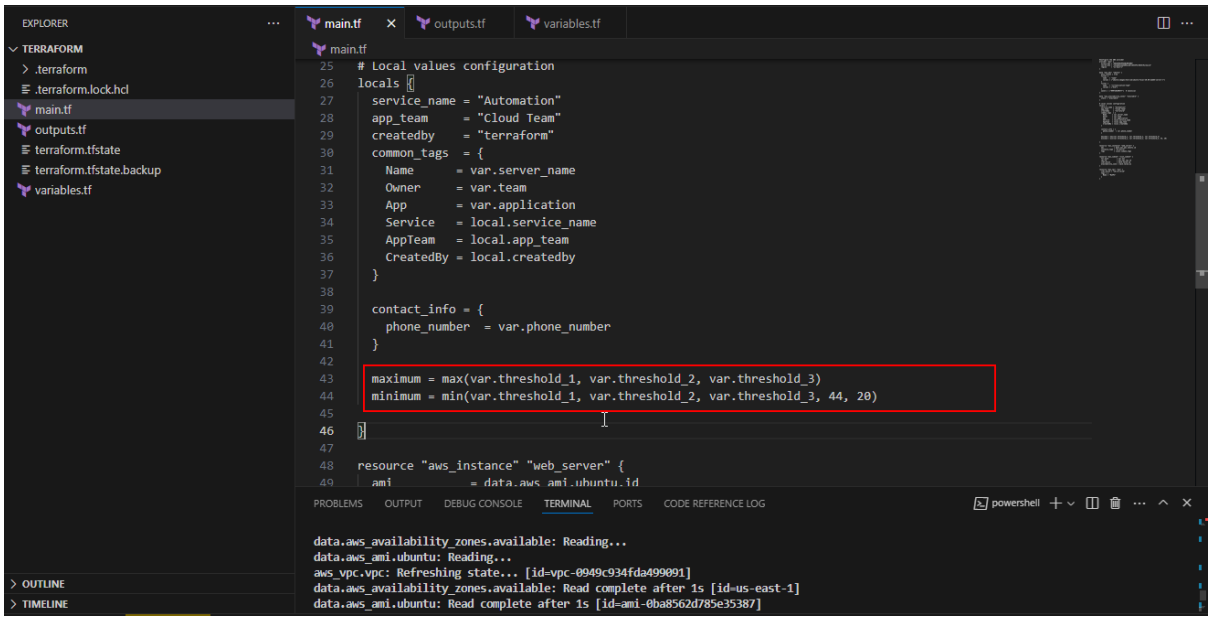
Step 6: Utilize Terraform built-in functions and dynamic blocks

6.1 Add the following numerical variables representing the threshold vaules in **varaibles.tf** as shown in the screenshot below:

```
variable "threshold_1" {
 type = number
 default = 10
}

variable "threshold_2" {
 type = number
 default = 25
}

variable "threshold_3" {
 type = number
 default = 30
}
```

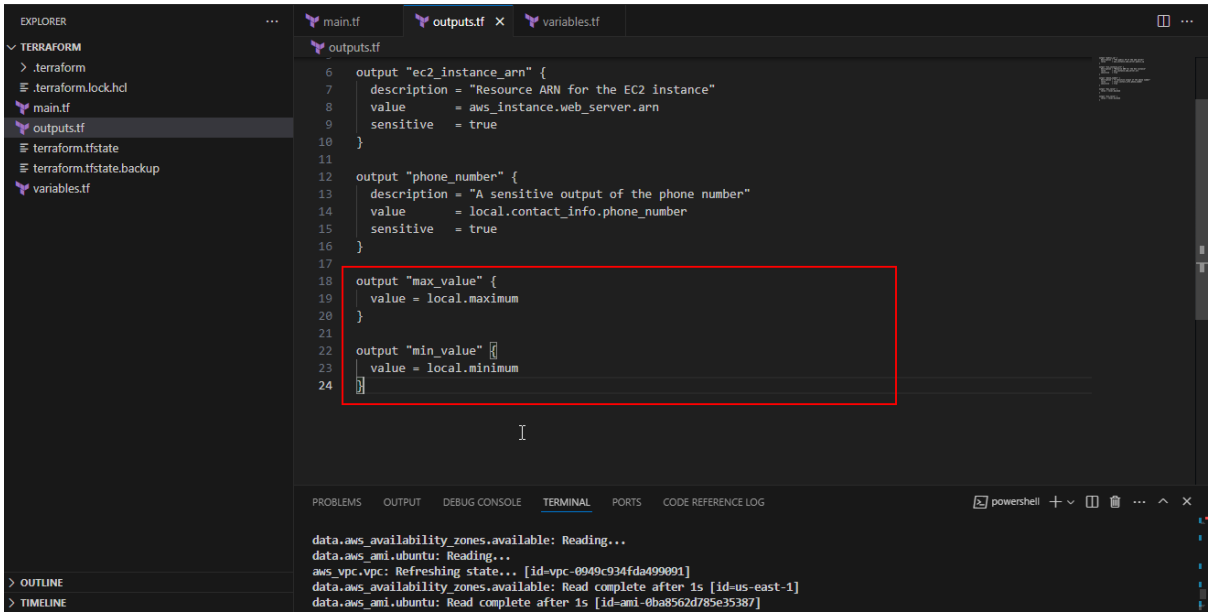


6.2 Add the following local values in the **main.tf** as shown in the screenshot below:
maximum = max(var.threshold_1, var.threshold_2, var.threshold_3)
minimum = min(var.threshold_1, var.threshold_2, var.threshold_3, 44, 20)



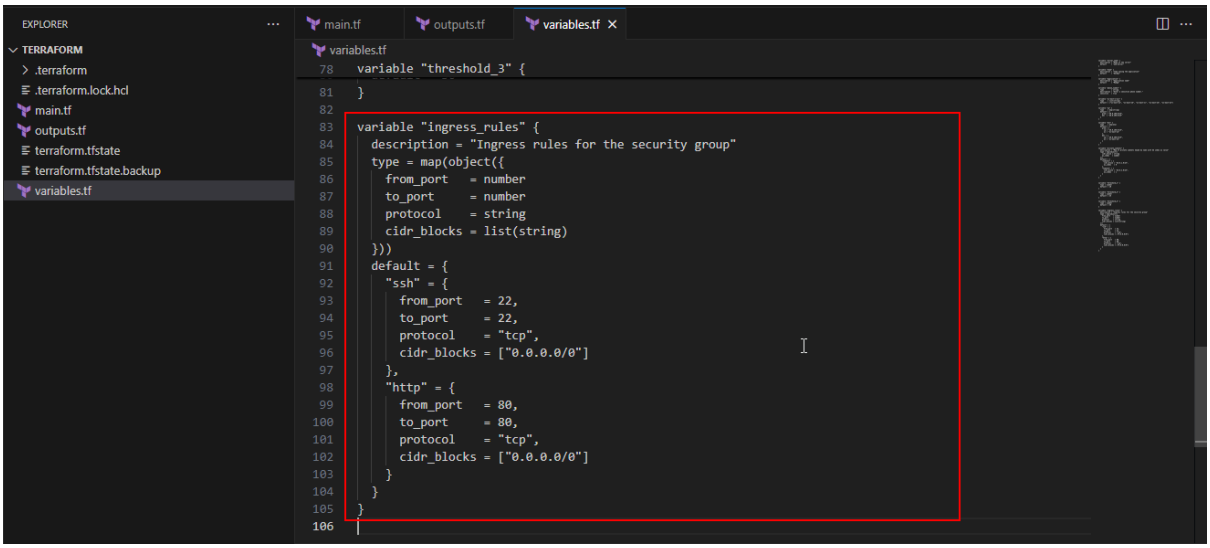
6.3 Add the following output block in **output.tf** to display the output as shown in the screenshot below:

```
output "max_value" {
  value = local.maximum
}

output "min_value" {
  value = local.minimum
}
```



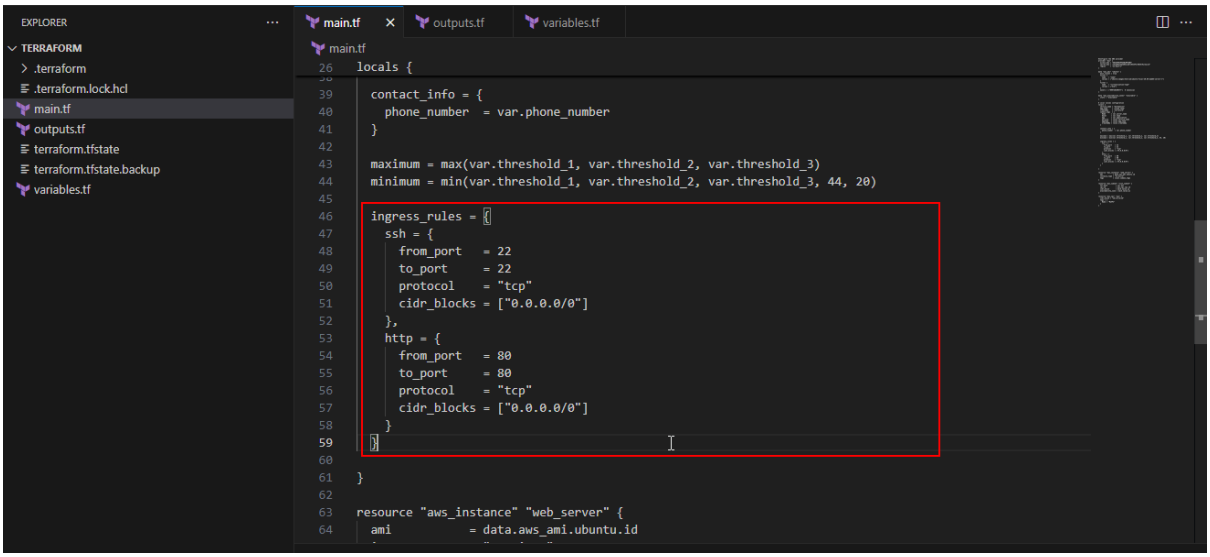
6.4 Add the following variable block in the **variables.tf** file as shown in the screenshot below:

```
variable "ingress_rules" {
 description = "Ingress rules for the security group"
 type = map(object({
  from_port   = number
  to_port     = number
  protocol    = string
  cidr_blocks = list(string)
 }))
 default = {
  "ssh" = {
   from_port   = 22,
   to_port     = 22,
   protocol    = "tcp",
   cidr_blocks = ["0.0.0.0/0"]
  },
  "http" = {
   from_port   = 80,
   to_port     = 80,
   protocol    = "tcp",
   cidr_blocks = ["0.0.0.0/0"]
  }
 }
}
```



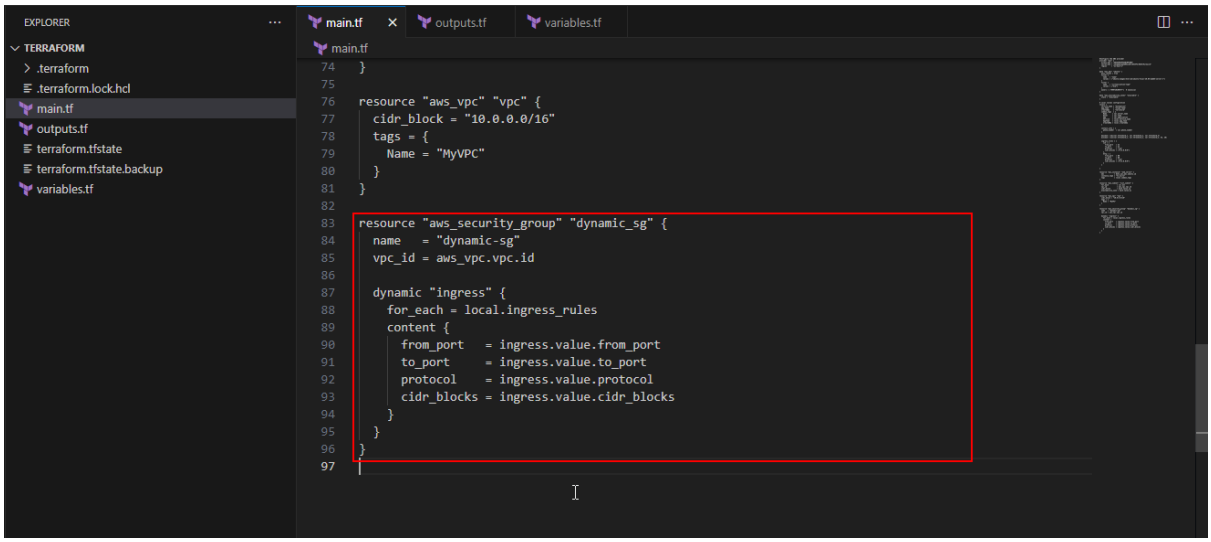
6.5 Define the ingress rules in the **locals** block as shown in the screenshot below:

```
ingress_rules = {
 ssh = {
  from_port   = 22
  to_port     = 22
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
 },
 http = {
  from_port   = 80
  to_port     = 80
  protocol    = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
 }
}
```

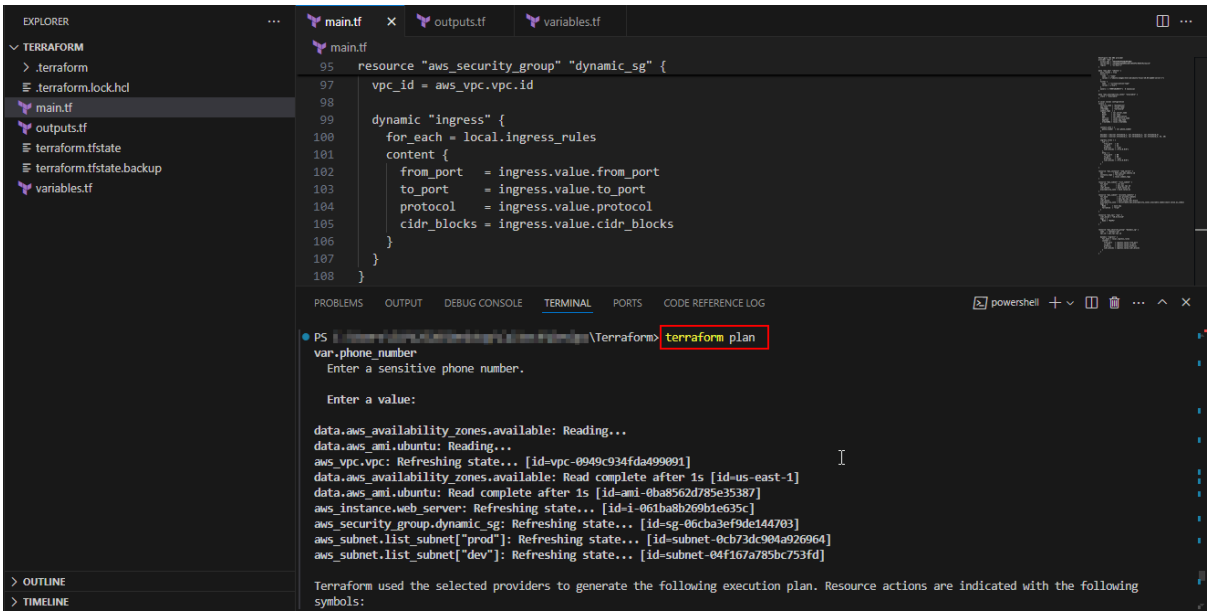


6.6 Add the following resource block in **main.tf** to implement dynamic blocks for security groups as shown in the screenshot below:

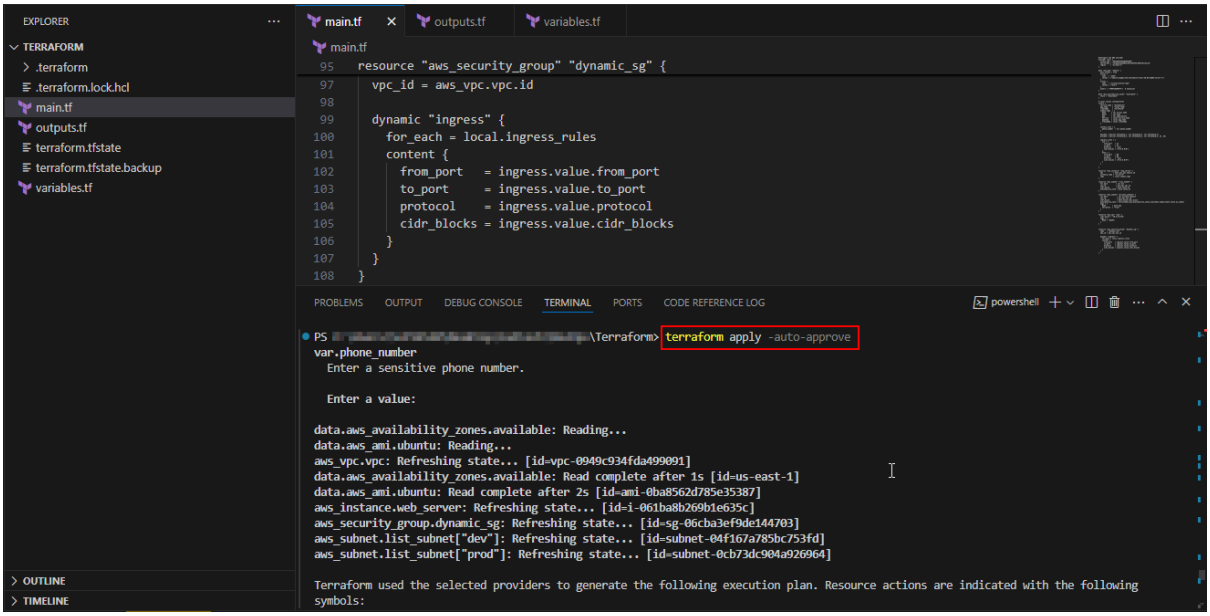
```
resource "aws_security_group" "dynamic_sg" {
  name   = "dynamic-sg"
  vpc_id = aws_vpc.vpc.id

  dynamic "ingress" {
    for_each = var.ingress_rules
    content {
      from_port   = ingress.value.from_port
      to_port     = ingress.value.to_port
      protocol    = ingress.value.protocol
      cidr_blocks = ingress.value.cidr_blocks
    }
  }
}
```

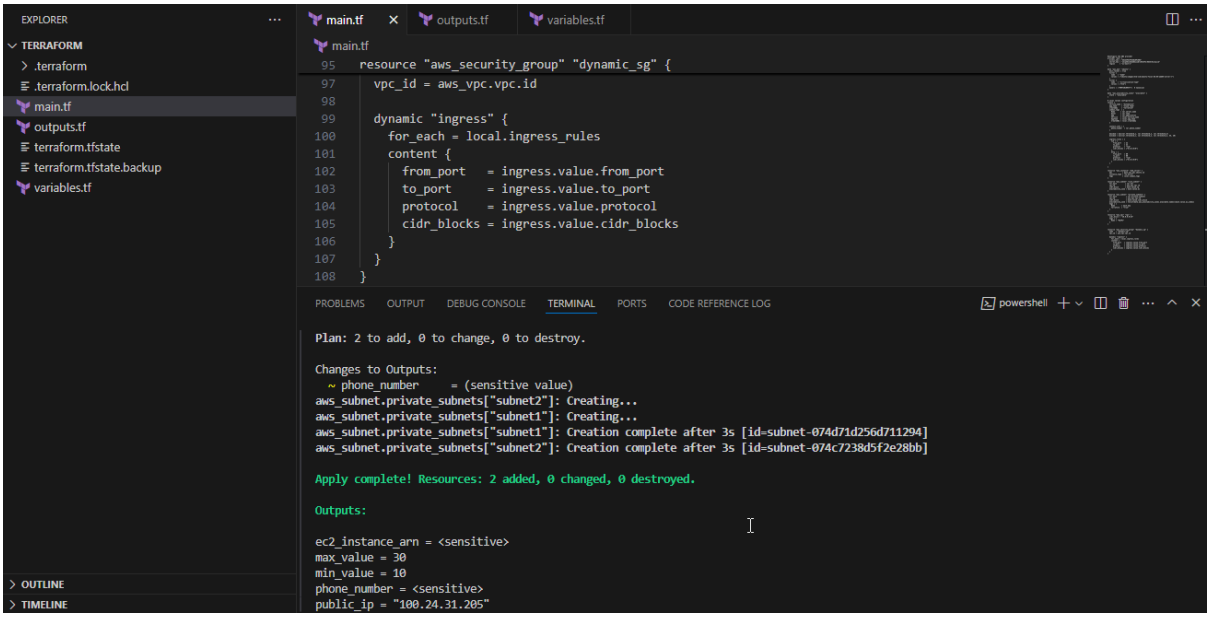


6.7 Execute the following command to preview the proposed changes in the infrastructure:
terraform plan

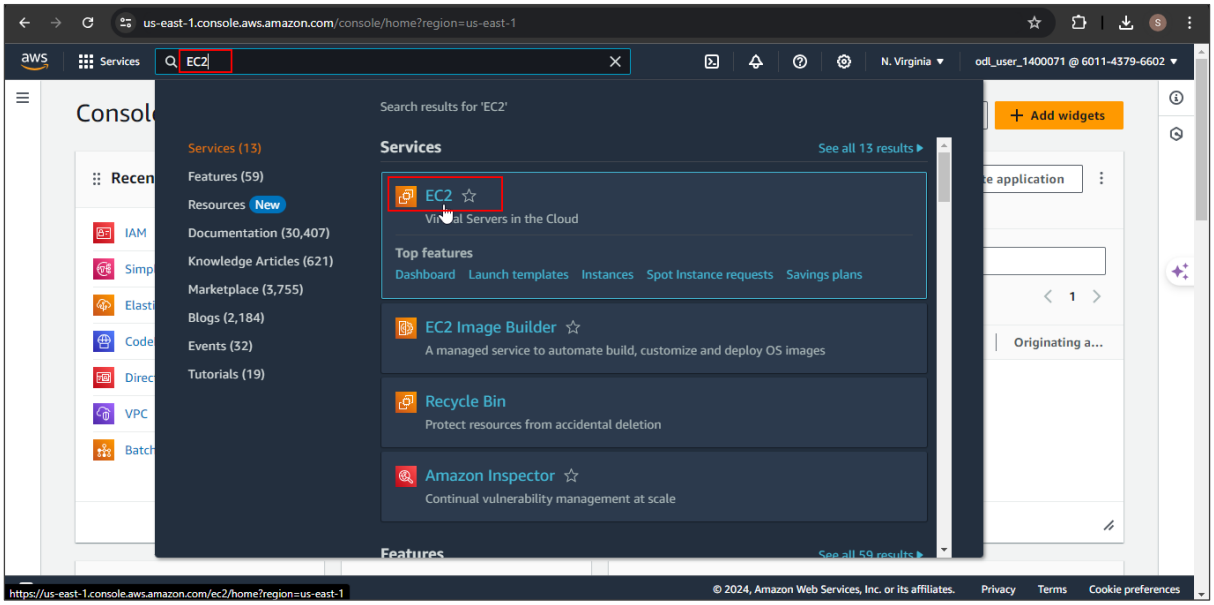


6.8 Apply the configuration using the following command to deploy the changes:
terraform apply -auto-approve

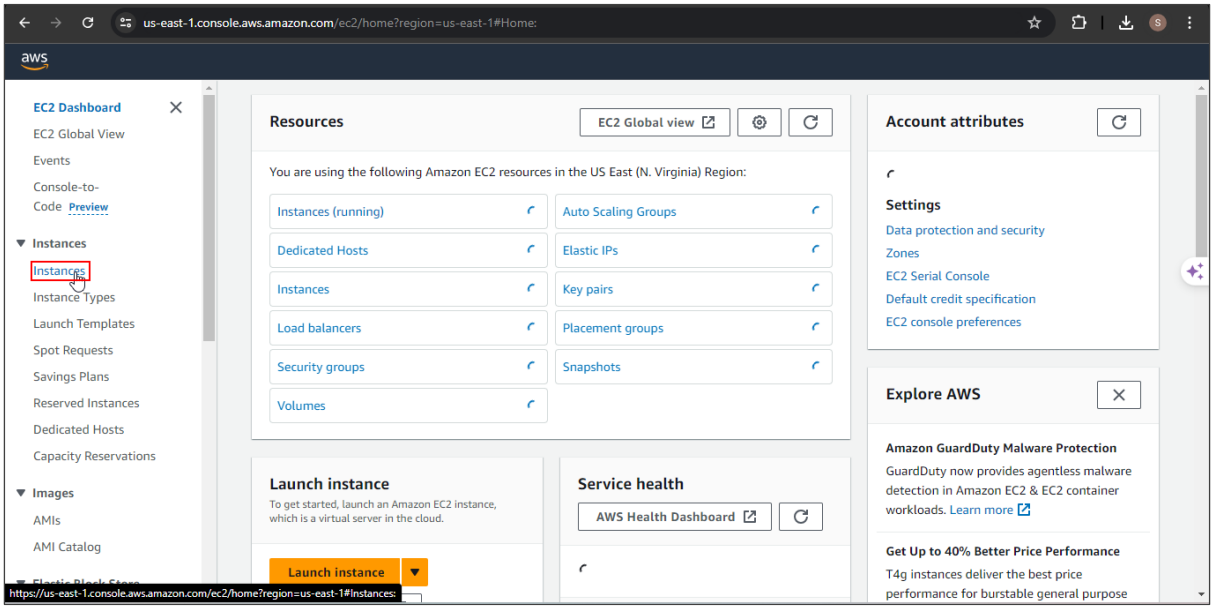




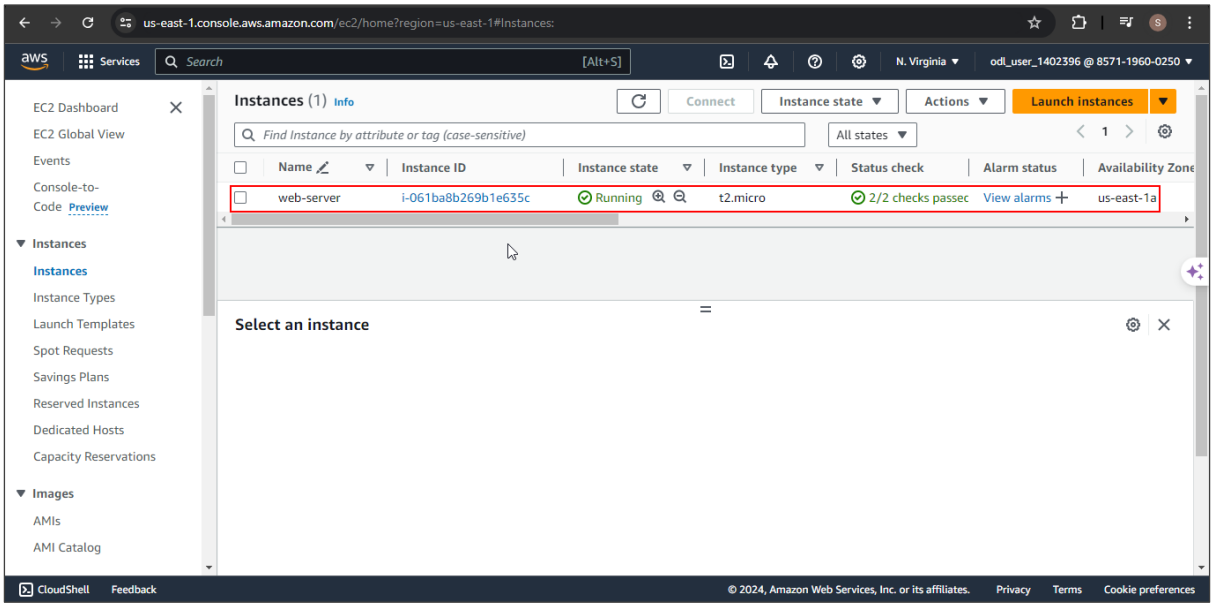
6.9 Navigate to the AWS console home, and search for and click on EC2 as shown in the screenshot below:



6.10 In the left pane, click on **Instances** as shown in the screenshot below:

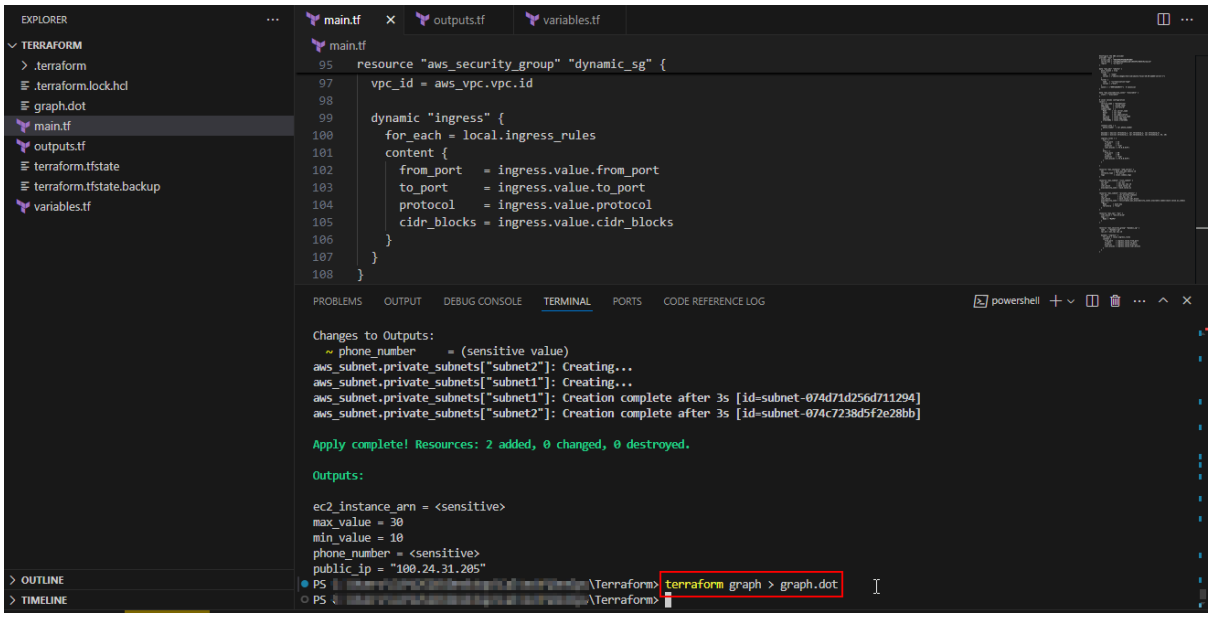


The EC2 instance has been created successfully as shown in the screenshot below:

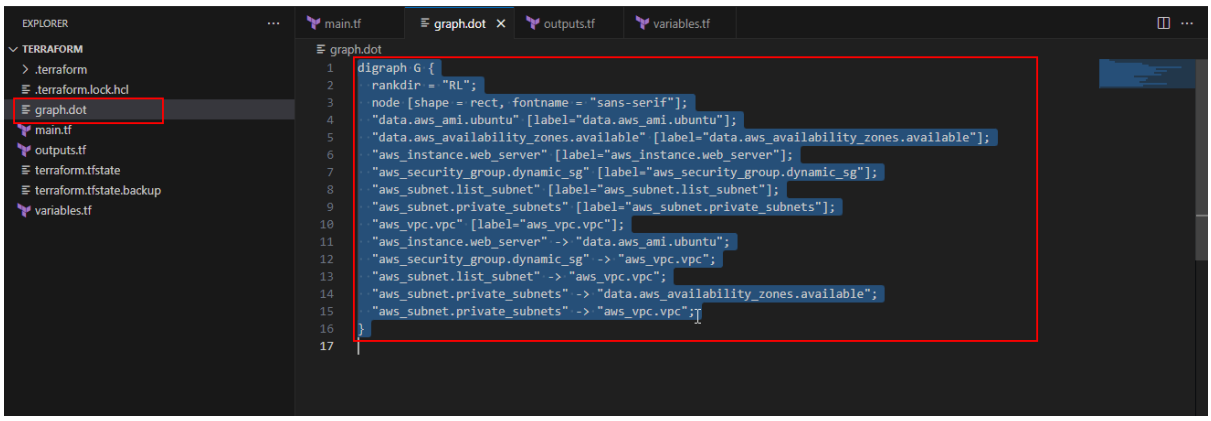


Step 7: Generate and visualize the resource graph

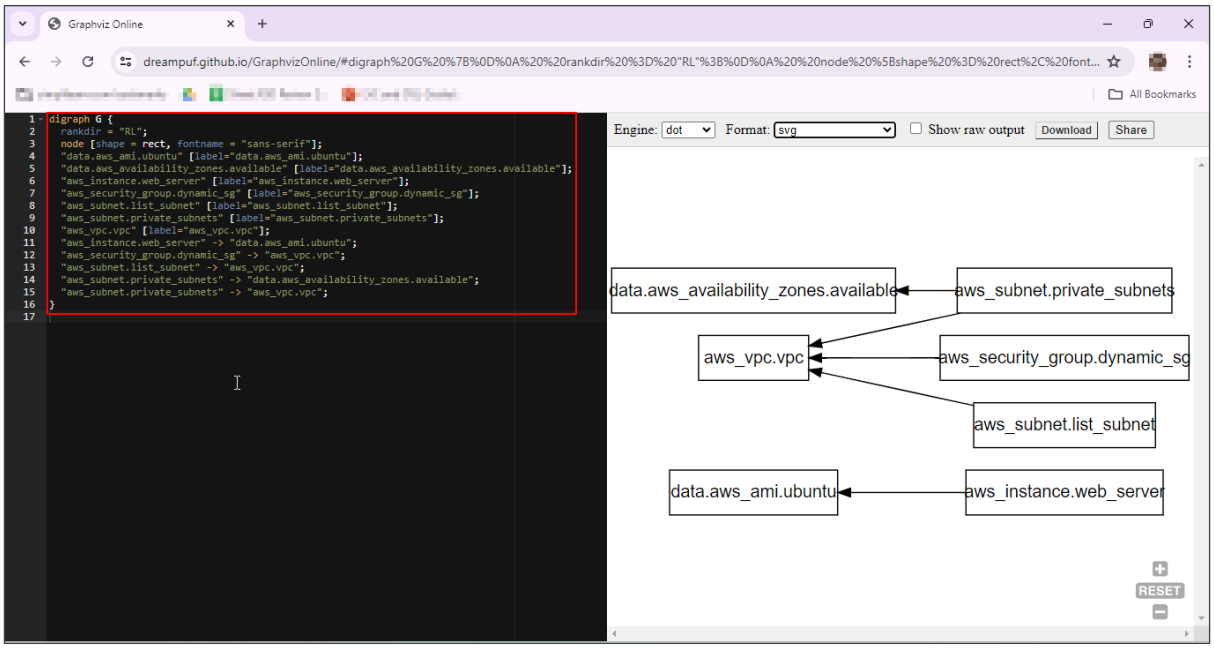
7.1 Execute the following command to generate a resource graph as shown in the screenshot below:
terraform graph > graph.dot



7.2 Click on the **graph.dot** file and copy the digraph as shown in the screenshot below:



7.3 Navigate to **https://dreampuf.github.io/GraphvizOnline/** and paste the copied digraph as shown in the screenshot below:



By following these steps, you have successfully set up and initialized a Terraform configuration, defined variables, locals, and outputs, and implemented resources using variables and locals. You have also secured and managed sensitive data, utilized collections and structure types, utilized Terraform built-in functions and dynamic blocks, and generated and visualized a resource graph.

This project consolidates various Terraform features and techniques, providing a solid foundation for managing and scaling infrastructure efficiently and securely.